โปรแกรมช่วยการออกแบบตกแต่งภายใน

1. นายชำนาญ ทัดสวน
2. นายประชา งามสิริศักดิ์
3. นางสาวพีระดา พานิช
4. นายอวิรุทธ์ เลี้ยงศิริ

ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตร์บัณฑิต
ภาควิชาคณิตศาสตร์ประยุกต์และวิทยาการคอมพิวเตอร์
คณะวิทยาศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้า เจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2536

Interior Design Assisting Program

1. Mr. Avirut Liangsiri
2. Mr. Chamnarn Tadsuan
3. Mr. Pracha Ngamsirisak
4. Ms. Peerada Panich

A Special Project Submitted in Partial Fulfillment of the Requirement for
the Degree of Bachelor of Science
Department of Applied Mathematics and Computer Science
Faculty of Science
King Mongkut's Institute of Technology Ladkrabang
1993

หัวข้อโครงงานพิเศษ	โปรแกรมช่วยการออกแบบตกแต่งภายใน
โดย	นายชำนาญ ทัดสวน
	นายประชา งามสิริศักดิ์
	นางสาวพีระดา พานิช
	นายอวิรุทธ์ เลี้ยงศิริ
ภาควิชา	คณิตศาสตร์ประยุกต์และวิทยาการคอมพิวเตอร์
อาจารย์ที่ปรึกษา	อ.ศรัณย์ อินทโกสุม

ภาควิชาคณิตศาสตร์ประยุกต์และวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับโครงการนี้เป็นส่วนของการศึกษาตามหลักสูตรวิทยาศาสตร์บัณฑิต

.....

(รองศาสตราจารย์ ดร. ไมตรี โพธิ์สุข)

หัวหน้าภาควิชาฯ

.....

(ผู้ช่วยศาสตราจารย์ สุนทร สุชาติเวชภูมิ)

ประธานกรรมการการสอบปัญหาพิเศษ

.....

(อ.สิริลักษณ์ เตียพิริยะกิจ)

กรรมการสอบปัญหาพิเศษ

.....

(อ.ศรัณย์ อินทโกสุม)

กรรมการและอาจารย์ที่ปรึกษาปัญหาพิเศษ

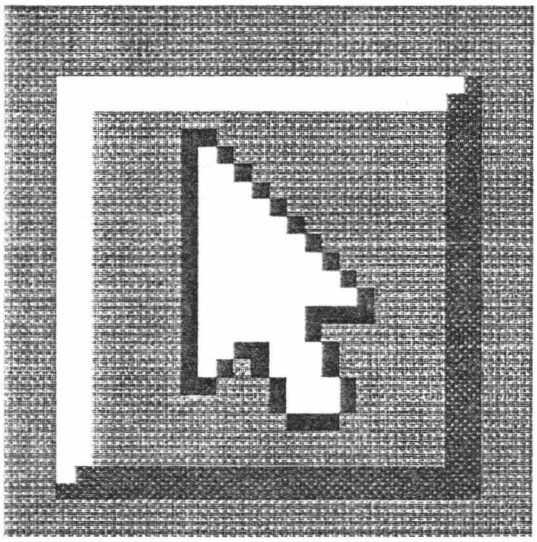
บทคัดย่อ

ทุกวันนี้ไมโครคอมพิวเตอร์มีบทบาทสำคัญในหลายๆด้าน เช่น วิศวกรรม วิทยาศาสตร์ ธุรกิจ และศิลปะ ปัจจุบันความสามารถของฮาร์ดแวร์บนเวิร์คสเตชั่นระดับสูงมีความสามารถสูงและราคาถูกลง จากวิธีการโปรแกรมต่างๆ ในการมองภาพเหมือนแบบ 3 มิติ มีวิธีทาง การติดตามรังสี รวมอยู่ด้วย นี้เป็นวิธีพื้นฐานสำคัญของการมองแบบเรขาคณิต โปรเจ็คนี้เสนอโปรแกรมสำเร็จรูปใช้ การติดตามรังสี ในการออกแบบสถาปัตยกรรมภายใน ออกแบบให้ช่วยเหลือโดยใช้เทคนิค"การติดตามรังสี" (Ray Tracing) ในการสร้างภาพมุมมอง 3 มิติ เพื่อความสะดวกของสถาปนิกที่จะออกแบบและสื่อสารกับลูกค้าเกียวกับความคิดต่างๆ ปัญหาพิเศษนี้เขียนบน Borland c++ และโปรแกรมช่วยเหลือทางกราฟิคบางตัว (GUI.LIB และ SVGA256.BGI)

Abstract

Nowaday microcomputers have an important role in many fields such as Engineering, Science, Business and Art. At present the ability of hardware that used to be on the hi-end workstation can be effort withmore performance and lower price. From this situation many programming method for 3-dimension visual realism has been invented including Ray Tracing, the method based on the principles of geometric optics. This project presenting Ray Tracing application on architect's interior design assisting by use Ray Tracing to generate the 3-D visual scene for architect's convenience to design and communicate to their customer about their idea. This project written on Borland C++ and some graphic utilities.

สารบัญ
CONTENTS



บทที่ 1 - บทนำ

บทที่ 1
บทนำ

ในปัจจุบันคอมพิวเตอร์ได้เข้ามามีบทบาทในหลาย ๆ สาขา เช่น วิศวกรรม วิทยาศาสตร์ ศิลปะ ธุรกิจ สถาปัตยกรรม และอื่น ๆ อีกมากมาย ในสาขา สถาปัตยกรรม ในระยะแรกได้นำคอมพิวเตอร์มาใช้ในการออกแบบโครงสร้างและรูปแบบของอาคาร สิ่งก่อสร้างต่าง ๆ เพราะการทำงานบนคอมพิวเตอร์สามารถทำการแก้ไขและเห็นผลได้ทันทีไม่ต้องมาแก้ไขตกแต่งใหม่ หรือวาดใหม่หมด ดังนั้นคอมพิวเตอร์จึงได้รับความนิยมอย่างมากในงานทางสถาปัตยกรรม แต่ก็เป็นเพียงเฉพาะงานการออกแบบโครงสร้างและรูปแบบภายนอกเท่านั้น เพราะการออกแบบภายในด้วยคอมพิวเตอร์ส่วนใหญ่ยังสามารถทำได้เพียงในลักษณะของการเขียนแปลนหรือแบบบนคอมพิวเตอร์เท่านั้น ยังไม่สามารถดูผลลัพธ์ได้ จึงยังจำเป็นที่สถาปนิกจะต้องทำการจินตนาการแบบแล้วนำมาเขียนเป็นรูป 3 มิติ เช่น Perspective หรือรูปในรูปแบบอื่น ๆ เพื่อนำเสนอจินตนาการในรูปแบบที่คนทั่วไปจะสามารถเข้าใจได้ ด้วยเหตุนี้ถ้าหากมีการเคลื่อนย้ายวัตถุ หรือเปลี่ยนแปลงวัตถุในแบบ สถาปนิกก็จำเป็นจะต้องวาดรูป 3 มิติใหม่ เพื่อให้ได้รูปที่ถูกต้องตามแบบ ซึ่งเป็นการเสียเวลาและสิ้นเปลืองแรงงานโดยไม่จำเป็น ด้วยเหตุนี้ทางคณะผู้จัดทำจึงได้เกิดความคิดขึ้นว่า ถ้าหากสามารถนำแบบแปลนที่ออกแบบในคอมพิวเตอร์แล้วนำไปสร้างภาพ 3 มิติในคอมพิวเตอร์เลย จะสามารถทำให้ลดเวลาในการทำงานลงได้มาก และทำให้งานการออกแบบตกแต่งภายในง่ายขึ้นจนกระทั่งเจ้าของบ้านพัก อาคาร ทั่วไปสามารถทำได้เองโดยไม่จำเป็นต้องมีความรู้ทางด้านสถาปัตยกรรมศาสตร์มากมาย ซึ่งจะเป็นประโยชน์แก่ประชาชนทั่วไปด้วย ไม่เพียงเฉพาะสถาปนิกเท่านั้น

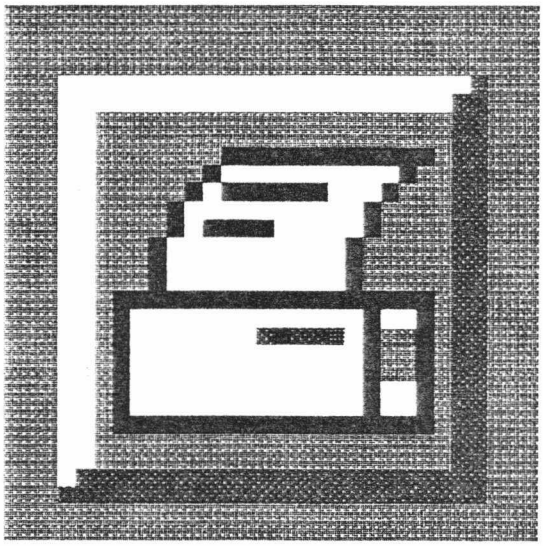
สำหรับเทคนิคที่จะได้เลือกใช้ในการสร้างภาพ 3 มิติ ซึ่งจากการคัดเลือกจากเทคนิคต่าง ๆ แล้วเห็นว่าเทคนิคการติดตามรังสีจะสามารถสร้างภาพได้คุณภาพที่ดีที่สุดถึงแม้ว่าจะต้องใช้กำลังของเครื่องคอมพิวเตอร์และเวลาในการสร้างภาพมากพอสมควรแต่ก็สมดุลกับคุณภาพของผลงานที่ได้ ซึ่งเทคนิคการติดตามรังสีใช้หลักการทางเรขาคณิตเชิงแสง(Optic Geometry) ซึ่งเป็นการจำลองเส้นทางเดินของแสงจากจุดกำเนิดไปยังทิศทางต่าง ๆ โดยคำนวณถึงจุดตัดและมุมที่แสงกระทบและมุมที่แสงสะท้อนไป โดยคำนวณซ้ำแล้วซ้ำเล่าจนกระทั่งแสงนั้นกระทบยังตาของผู้สังเกตที่สมมติขึ้นซึ่งเทคนิคนี้มีอยู่ 2 ชนิดคือ

1. การติดตามรังสีแบบไปข้างหน้า(Forward Ray Tracing)
2. การติดตามรังสีแบบย้อนหลัง(Backward Ray Tracing)

ซึ่งรายละเอียดของแต่ละชนิดจะได้กล่าวถึงอีกครั้งในบทที่ 2

จากที่กล่าวมาเราสามารถกล่าวได้ว่า การใช้คอมพิวเตอร์ในงานทางสถาปัตยกรรมในปัจจุบันยังไม่ครบ

วงจร โดยปัจจุบันได้เน้นในการใช้เพื่อออกแบบโครงสร้างและรูปแบบภายนอก โดยการออกแบบภายในซึ่งเป็นงานที่ยุ่งยากและมีความสำคัญไม่ยิ่งหย่อนไปกว่ากัน กลับถูกใช้เพียงเล็กน้อย ซึ่งปัญหาพิเศษนี้จะสามารถทำให้การออกแบบตกแต่งภายในสามารถทำได้ง่ายขึ้นกว่าแต่ก่อน และครบวงจรตั้งแต่การออกแบบ แก้ไข ตรวจสอบผลที่ได้ (ดูภาพ 3 มิติ) นำเสนอ (นำภาพที่สร้างมาแสดงต่อลูกค้า) พิมพ์แบบเพื่อนำไปสร้างจริง ซึ่งจะสามารถทำได้ในโปรแกรมเดียว



บทที่ 2 - ทฤษฎีและหลักที่เกี่ยวข้อง

บทที่ 2
ทฤษฎีและหลักที่เกี่ยวข้อง

Ray-Tracing

บทนำ

การสร้างภาพหรือรูปเหมือนจริงเป็นเรื่องที่นักคอมพิวเตอร์กราฟฟิคพยายามมากว่า 10 ปี ด้วยเหตุว่าปัจจุบันคอมพิวเตอร์นั้นมีพลังมากขึ้น และฮาร์ดแวร์ I/O กราฟฟิค มีอยู่ทั่วไป เมื่อเราต้องการสร้างแบบและสี จะสามารถทำได้ด้วยวิทยาศาสตร์คอมพิวเตอร์ โดยมีรายละเอียดมากมายจากความเป็นจริง แน่นอนว่านักคอมพิวเตอร์กราฟฟิคจะยังคงติดตามคิดค้นวิธีที่ดีขึ้นเพื่อสร้างรูปต่อไป จนกว่าจะพบวิธีที่ง่ายและใกล้เคียงกับธรรมชาติที่สุด การเลียนแบบธรรมชาติจะดำเนินต่อไปบนคอมพิวเตอร์ที่มี อัลกอริทึม และกำลังอันมากมาย

เทคนิคของการติดตามรังสีมีผลมาจากความต้องการแสดงรูปเหมือนจริงโดยไม่จบสิ้นการติดตามรังสีนั้นมีความสามารถครอบคลุมตั้งแต่การเก็บรวบรวมในครั้งเดียว ไปจนถึงการติดตามผล เหตุผลสำคัญสำหรับเรื่องนี้มีอยู่ 3 ข้อ

1. แนวความคิดของการติดตามรังสีคือง่ายต่อการทำความเข้าใจโดยบุคคลที่สนใจในเทคนิคนี้ ไม่ใช่ใช้ได้เฉพาะนักคอมพิวเตอร์กราฟฟิคเท่านั้น
2. ในช่วง 10 ปีที่ผ่านมา ประสิทธิภาพของงานเริ่มดีขึ้น มีการสร้างคอมพิวเตอร์ที่มีพลังมากขึ้น สามารถแก้ไขการขาดประสิทธิภาพของงานได้โดยใช้พลังที่เพิ่มขึ้นของคอมพิวเตอร์และเพิ่มเทคนิคใหม่ที่จะรวมไว้ในโปรแกรมการติดตามรังสีซึ่งจะทำให้มีประสิทธิภาพมากยิ่งขึ้น
3. อัลกอริทึมของการติดตามรังสี รวบรวมผลลัพธ์ที่มองเห็นได้ในวิธีที่ตรงไปตรงมา มีการเพิ่มผลลัพธ์ในเทคนิคคอมพิวเตอร์กราฟฟิค 3 มิติอื่น ๆ จะทำให้มีความยากมากขึ้น

ในทฤษฎีเบื้องต้นของ การติดตามรังสี ซึ่งเมื่อเราวาดเส้นโดยต้องเลือกใช้เทคนิคระหว่างขั้นต้นกับขั้นสูง จะขึ้นอยู่กับจุดประสงค์ อย่างไรก็ตาม จุดมุ่งหมายที่นี่คือการนำเสนอทฤษฎีที่เพียงพอสำหรับเพื่อให้คุณมีพื้นฐานที่เพียงพอเพื่อสร้างโปรแกรม การติดตามรังสี เบื้องต้นสำหรับการออกแบบ และมีข้อมูลข่าวสารที่เพียงพอ ดังนั้นคุณจะสามารถเข้าใจการวิ่งของ การติดตามรังสี ได้ โดยจะต้องอ่านและเข้าใจเนื้อหาเรื่องนี้ก่อนจะคิดหัวข้อ การติดตามรังสี ขั้นสูงต่อไป

ในขั้นแรก เราจะกล่าวถึง หลักการ การติดตามรังสี ทั่วๆไป ที่จะให้ความคิดกับคุณว่ามีกรรมวิธีทำงานอย่างไร ได้มีการอธิบายเรื่องเวกเตอร์ทางคณิตศาสตร์ขั้นต้น โดยจำเป็นที่จะเข้าใจในอัลกอริทึมของการติดตามรังสี ต่อมาจะดูทฤษฎีอื่นๆที่จะอธิบายเรื่องรังสี , การตัดกันของรังสีกับวัตถุ และเรื่องเงา ในที่สุด เราจะสามารถคิดแก้ปัญหาซึ่งมีอยู่ในเรื่อง การติดตามรังสี ได้

แนวคิดทั่วไป (General Concept)

ความมุ่งหมายของส่วนนี้คือ ให้คุณมีความเข้าใจถึงกรรมวิธีการทำงานของการติดตามรังสี ในทางปฏิบัติจะกำหนดลักษณะทั่วไปถึงวิธีที่ การติดตามรังสี ทำงาน ซึ่งเราจะพูดถึงในภายหลัง

เหตุผลของการติดตามรังสีคือสร้างรูปเหมือนจริง ซึ่งรูปนั้นจะเป็นส่วนหนึ่งในการแสดงของแสงและเงา ที่มีภาพ 3 มิติใกล้เคียงกับสิ่งที่เราจะพบได้ในธรรมชาติ หนึ่งในข้อได้เปรียบที่ใหญ่ที่สุดของ การติดตามรังสี บนเทคนิคการแปลง 3 มิติ คือความง่ายดายของผลลัพธ์ที่เปรียบเสมือนเงา,การสะท้อนกลับการติดตามรังสีสามารถสร้างรูปสมมุติที่มองสะท้อนกลับได้ เพราะการลากเส้นของรังสีคล้ายกับแสงสว่าง หรือพูดอีกอย่างหนึ่ง การติดตามรังสี จำลองส่วนของรังสีแสง(light ray) และวัตถุโดยใช้หลักการมองแบบเรขาคณิต กฎที่ใช้สำหรับ รังสีแสง/ส่วนตัดของวัตถุ คือ ทำหลักธรรมชาติให้ง่ายลง แต่ยังคงไว้ซึ่งผลลัพธ์ที่ถูกต้อง โปรแกรม การติดตามรังสี จะสมมุติว่า รังสีแสง เดินทางเป็นเส้นตรง และกระทำกับวัตถุที่พื้นผิวด้านนอกเท่านั้น คุณสมบัติของแสง เช่น diffraction, phase, polarization, ความยาวของคลื่น และ attenuation บนระยะทาง จะไม่นำไปพิจารณาเลย การสร้างแบบจำลองของปรากฏการณ์เหล่านี้ยังคงผลักดันการค้นคว้าด้านคอมพิวเตอร์กราฟฟิค เพื่อว่าให้ได้แบบที่ ทำให้ง่ายลงและผลงานคงเดิม จากผลเหล่านี้จึงถูกรวบรวมเข้าเป็นโปรแกรมการติดตามรังสี

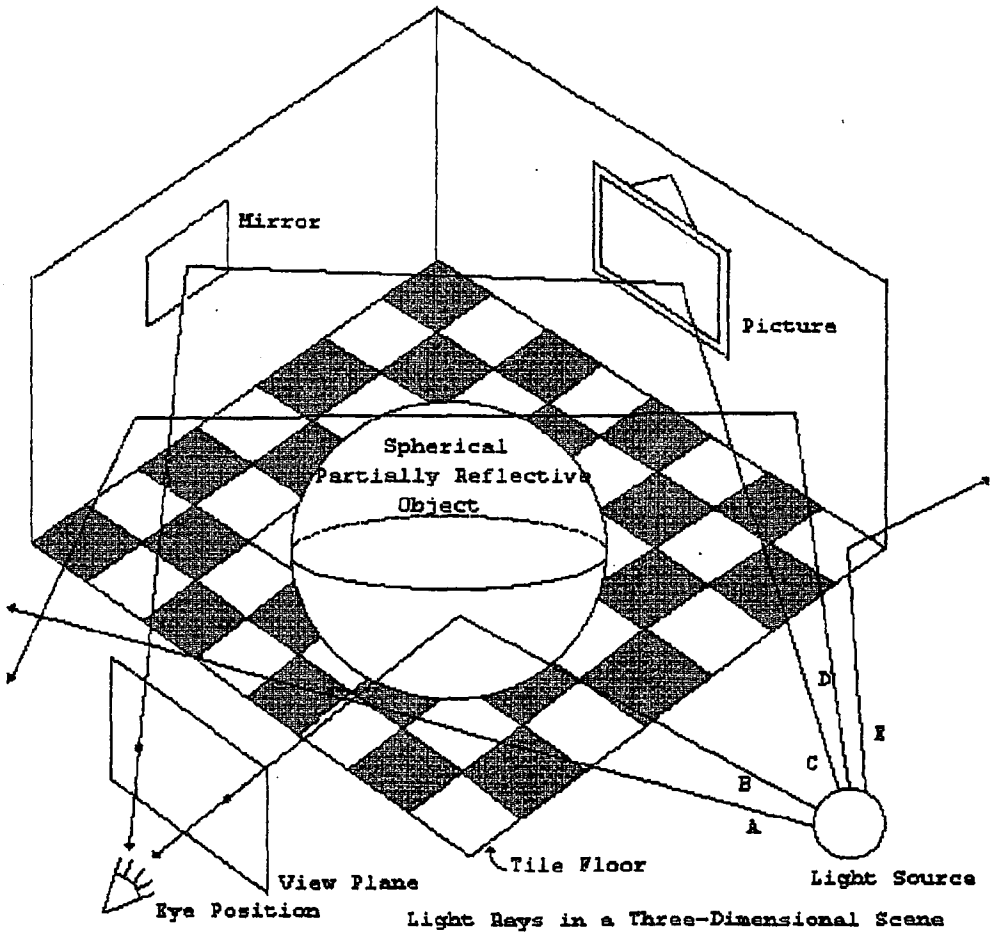
การติดตามรังสี เป็นตัวอย่างของอัลกอริทึมทางกราฟฟิค ที่อ้างถึงอัลกอริทึมการสุ่มจุด (Point Sampling Algorithm) โดยอัลกอริทึมเหล่านี้จะกำหนดการเห็นพื้นผิวของวัตถุที่ใช้จุดตัวอย่างเป็นจำนวนจำกัด และสร้างข้อกำหนดเกี่ยวกับจุดที่อยู่ระหว่างนั้นอัลกอริทึมอื่นในชั้นนี้ได้แก่ Z-buffer, painter's และ scanline ซึ่งทั้งหมดนี้เกี่ยวข้องกับพื้นผิวและการเคลื่อนย้าย ส่วนขั้นหลักอื่นๆของอัลกอริทึมเป็นการอ้างถึงอัลกอริทึมต่อเนื่องซึ่งไม่ใช่การคาดคะเน พวกเขาพยายามควบคุมการมองเห็นบนพื้นผิวทั้งหมด การคำนวณเหล่านี้ ถ้าเราลองจินตนาการดูจะรู้ว่าสิ้นเปลืองเวลามากจึงไม่เหมาะกับคอมพิวเตอร์ส่วนบุคคลในเวลานี้

ปัญหา 2 อย่างที่เป็นผลของการสุ่มจุดของการติดตามรังสีที่สืบเนื่องต่อๆ มา คือ ประสิทธิภาพ และ Aliasing การลดลงของประสิทธิภาพเกิดเนื่องมาจากจำนวนของจุดของส่วนของรังสี/วัตถุ ที่เพิ่มขึ้นทำให้เปลืองเวลาในการคำนวณ เมื่อไม่สามารถเพิ่มจำนวนของจุดขึ้นจึงยากที่จะได้ภาพเหมือนจริง ในทำนองเดียวกันหากมีการใช้แหล่งกำเนิดแสงแทรกเข้าไปในฉากมาก ๆ จะทำให้เกิดปัญหาคอขวดในเชิงประสิทธิภาพ

อะไลแอซ(Aliasing) เป็นปัญหาที่ต้องพบในทุกระบบตัวอย่าง ไม่ว่าจะเป็นกราฟฟิคหรืออิเลคทริค การอะไลแอซ เป็นสิ่งที่เกิดขึ้นจากความบกพร่องในการสร้างสัญญาณจากตัวอย่าง ความเสียหายจากปัญหาการอะไลแอซ คือความถี่ของสัญญาณที่เป็นตัวอย่างและอัตราตัวอย่าง ถ้าอัตราตัวอย่างหรือความถี่นั้นสูงเพียงพอก็จะไม่มีปัญหาการอะไลแอซอย่างไรก็ตาม โดยเหตุว่าความถี่ของตัวอย่างใกล้เคียงกับสัญญาณตัวอย่าง สัญญาณความถี่ต่ำที่เราไม่ต้องการ จะสร้างส่วนที่ไม่ใช่สัญญาณออกมาซึ่งเรียกส่วนนี้ว่าการเกิดอะไลแอซ

ในส่วนของความคิด เมื่อคุณใช้กรรมวิธีการสุ่มจุด เช่น การติดตามรังสี เพื่อสร้างภาพที่มีรายละเอียด (ส่วนที่มีความถี่สูง) บางส่วนของรายละเอียดจะถูกเปลี่ยนแปลงโดยการอะไลแอซ รูปต่างๆจะมีผลลัพธ์ที่แตกต่างกันการเกิดอะไลแอซที่อยู่ในรูป ray traced เป็นสิ่งไม่ดี แต่เมื่อรูปที่จะมีชีวิตต้องการปัญหามาประกอบกัน เนื่องจากเวลามีการแปลงรูปโดยนำการแก้ปัญหาขั้นสูงมาช่วย (ตัวอย่างความถี่สูง) แต่ก็ไม่สามารถแก้ปัญหาโดยสิ้นเชิงได้ เทคนิคกราฟฟิคที่ใช้ต่อสู้กับปัญหาการเกิดอะไลแอซ เรียกว่าเทคนิค "แอนตี้อะไลแอซ", "anti-aliasing"

การสร้างรูปติดตามรังสี เริ่มโดยแสดงความหมายของฉาก 3 มิติ ฉากประกอบด้วยแหล่งกำเนิดแสงและวัตถุ วัตถุแต่ละอย่างจะให้แต่ละตำแหน่งที่ตั้ง และลักษณะเฉพาะ(Attributes) แสดงถึงรูปร่างและคุณสมบัติของพื้นผิว(สี,การสะท้อน,รูปทรง) โดยการแสดงฉาก อย่างแรก คุณต้องสร้างภาพเรขาคณิต ที่ระบุว่าที่ใดในรูป 3 มิติ มีระยะช่องมองของตาที่ตั้งอยู่ ทิศทางตรงที่ตามองเห็นมีการเตรียมเอาไว้ ตาจะมองฉาก 3 มิติผ่านด้านตรงข้ามมุมฉากกับหน้าต่างในช่องว่าง เรียกว่าระนาบการมอง("view plane") จะจับอยู่ที่จอภาพของคอมพิวเตอร์ และนั่นคือที่ที่ตาจะมองเห็นฉากของภาพ โดยระนาบการมองจะอยู่ในแนวราบระหว่างตาและวัตถุในฉาก 3 มิติเสมอ ดังรูปข้างล่างแสดงถึงฉากกับวัตถุ แหล่งกำเนิดแสงและตำแหน่งของตาในแนวตั้ง



Light Rays in a Three-Dimensional Scene

ดูเหมือนจะเข้าใจว่า การค้นหา รังสีแสง นั้น เราต้องเริ่มที่แหล่งกำเนิดแสง รังสีที่ถูกสร้างขึ้นจะเข้าไปในฉากและมองว่าเมื่อบางรังสีผ่านระนาบการมอง หลังจากสะท้อนผ่านฉากและมองเห็นได้ด้วยตา หลักการนี้บางครั้งถูกเรียกว่า การติดตามรังสีแบบไปข้างหน้า (forward ray tracing) ได้ผลดี ถึงแม้ว่าจะต้องใช้การคำนวณจำนวนมาก เพื่อการให้ได้การแรเงาที่ละเอียดถูกต้อง จะต้องคำนวณติดตามถึง หลายพันล้านรังสีแสง ซึ่งมีเพียงไม่กี่เปอร์เซ็นต์เท่านั้นที่จะผ่านระนาบการมองซึ่งเป็นการเปลืองเวลาอย่างมากนับแรมเดือน

การคิดเปลี่ยนแปลงไปเพียงเล็กน้อย สามารถสร้างเทคนิคซึ่งมีการคำนวณจะมีประสิทธิภาพมากกว่าการติดตามรังสีแบบไปข้างหน้า เทคนิคนี้เป็นที่รู้จักในหลายๆชื่อ ได้แก่ เรย์คาสติ้ง(ray casting), การติดตามรังสีแบบย้อนหลัง (backward ray tracing) และ การติดตามรังสีแบบรีเวิร์ส (reverse ray tracing) ความคิดเกี่ยวกับการย้อนกลับของทางเดินของรังสี(แสง) เริ่มที่ตำแหน่งที่ตั้งของตา รังสีทั่วๆไปจะผ่านแต่ละจุดสีของระนาบการมองบนเส้นทางของมันในฉาก 3 มิติ คล้ายกับรังสีได้กระทบวัตถุในฉาก รังสีจะผ่านเม็ดสีและให้สีออกมาบนแนวตัดของวัตถุ ถ้าจุดที่ตัดกันระหว่างรังสีและวัตถุเป็นภาพสว่างของแหล่งกำเนิดแสง จุดของการตัดจะไม่ได้อยู่ในเงา จะเกิดแสงสว่างอย่างเต็มที่จากแหล่งกำเนิดแสง อย่างไรก็ตาม วัตถุในฉากจะอยู่ในแนวราบระหว่างจุดที่ตัดกันและแหล่งกำเนิดแสง วัตถุที่จุดจะอยู่ในเงาและความเข้มของจุดสีจะแสดงออกมา การจำกัดสีของวัตถุที่จุดตัดสามารถทำได้ง่ายถ้าไม่ใช่ความจริงว่าวัตถุสามารถสะท้อนแสงได้ หรือการสะท้อนกลับในทางบวกนั้นมีรูปทรงจริงแล้ว ความคิดเกี่ยวกับการติดตามรังสีโดยไม่มีการสะท้อนกลับ จะไม่ได้รับความสนใจที่จะมองเลย สีของจุดรังสี/รอยตัดของวัตถุ จะทำหน้าที่สีของวัตถุ, สีของแหล่งกำเนิดแสง และช่วยเหลือการสะท้อนหรือการหักเหของแสง ดังตัวอย่าง รังสี C ในภาพ 2.1 ยิงรังสีจากตาไปยังฉากจะตัดกระจกบนผนังเป็นอย่างแรก ผิวกระจกจะสะท้อนกลับได้อย่างสมบูรณ์ ซึ่งไม่มีสีจริงของตัวเองเลย สีที่เราสังเกตุเห็นโดยตาในกระจกนั้นจะสะท้อนจากรูปที่ตรงกันข้ามกับกำแพง และถูกกำหนดโดยแหล่งกำเนิดแสง ถ้าแหล่งกำเนิดแสงเป็นสีอื่นที่ไม่ใช่สีขาว ส่วนของภาพสะท้อนก็จะเป็นสีเหมือนที่เห็นด้วยตา การคำนวณของสีเป็นส่วนที่ซ้ำๆกัน มีชื่อว่า"shading" ทุกๆครั้งของการสะท้อนหรือวัตถุที่สะท้อน จะถูกตัดโดยรังสีของตา การเพิ่มของรังสีแต่ละส่วนจะลากเส้นกลับไปยังแหล่งกำเนิดแสงของมัน เพื่อกระทำกับสีของวัตถุเพราะส่วนประกอบแต่ละส่วนนั้นต่อเนื่องกัน เหตุคือ รังสีแบบต้นไม้ (tree) ที่มีกิ่งก้าน จะอธิบายเรื่องส่วนประกอบของสีของวัตถุได้ สีจะช่วยเรื่องการออกจากต้นไม้และผ่านกลับเข้าไปใหม่ รูปแบบแต่ละอย่างจะอยู่ภายใต้อิทธิพลของสีสุดท้ายของวัตถุ และรวมถึงสีของจุดสีด้วยการบรรยายจะพยายามนำเสนอส่วนที่เกี่ยวข้องกับการติดตามรังสี ในรูปแบบปกติมองดูรังสีรอบๆตัวเรา และพยายามจินตนาการถึงส่วนของ รังสีแสง ที่เข้ามาในตาและมองเห็นกริด (grid) อยู่ตรงหน้า คุณจะรู้สึกถึงภาพ 3 มิติที่ผ่านเข้าไปในระนาบการมอง เท่านั้น เราใช้หลักฟิสิกส์มาช่วยให้เข้าใจในส่วนนี้ได้

เวกเตอร์และคณิตศาสตร์เวกเตอร์(Vectors and Vector Mathematics)

ในส่วนนี้ เราจะถกปัญหาด้านคณิตศาสตร์ที่กระทำต่อการติดตามรังสี โดยเฉพาะคณิตศาสตร์นี้เป็นการเรียนพิเศษเฉพาะในส่วนที่ไม่ใช่แบบที่ใช้คอมพิวเตอร์โปรแกรมเมอร์อยู่เป็นหลักเท่านั้น อย่างไรก็ตาม ถ้าคุณมองออกจะพบว่าคณิตศาสตร์นี้ไม่ใช่เรื่องยากเลย หลักของคณิตศาสตร์ที่แก้ปัญหานี้ จะใช้โปรแกรมการติดตามรังสี ทั้งหมด ไม่ใช่เฉพาะที่นำเสนอในที่นี้เท่านั้น เวกเตอร์("vector")สร้างขึ้นโดยใช้ การติดตามรังสีเป็นฐาน เวกเตอร์เป็นปริมาณที่มีทั้งขนาดและทิศทาง เวกเตอร์ใช้ความรู้ทั้งด้านวิศวกรรมและฟิสิกส์เพื่อแสดงถึง แรง, ความเร็ว, ความเร่ง, สนามกำลังของอิเลกตรอน และแรงสนามแม่เหล็ก ของเวกเตอร์ 2 และ 3 มิติ ใช้อย่างแพร่หลายในหลายด้านของงานวิทยาศาสตร์ ในความตั้งใจของเรา เวกเตอร์มักจะใช้ปริมาณ 3 มิติกับขนาดและทิศทาง ในแนวแกน X,Y,Z และแสดงสถานะอย่างชัดเจน หรือมิฉะนั้น เวกเตอร์ทั้งหมดจะถูกสมมุติให้มีจุดกำเนิดที่ 0,0,0 และขยายระยะทางออกไปในแต่ละแนวแกน X,Y,Z การมองเวกเตอร์ให้คล้ายกับลูกศร โดยหางของมันอยู่ที่จุดกำเนิด และหัวอยู่ที่ X,Y,Z ตัวที่เกี่ยวข้องกันเวกเตอร์ที่มีจุดกำเนิด บางครั้งจะอ้างถึงตำแหน่งของเวกเตอร์ ถึงแม้ว่าจะสามารถแสดงหรือจัดให้อยู่ในพิกัดเชิงขั้วได้ เรามักใช้ระบบ Cartesian Coordinate ในการพิจารณาเนื่องจากจุดกำเนิดมีเพียงเวกเตอร์ 3 จำนวนที่ต้องการระบุในที่นี้ เราจะแสดงเวกเตอร์ในรูปของ $<V_x,V_y,V_z>$ V_x แทนเวกเตอร์จากแกน X , V_y แทนเวกเตอร์จากแกน Y , V_z แทนเวกเตอร์จากแกน Z จากรูปแบบนี้ จะสามารถแสดงถึงขนาดและทิศทางที่เวกเตอร์ชี้ถึงได้

การบวกเวกเตอร์(Vector Addition)

การกระทำขั้นพื้นฐานที่จะประยุกต์เวกเตอร์เข้ากับเวกเตอร์ คือการบวก การบวกเวกเตอร์ในสามมิติกระทำได้โดยบวกแต่ละคู่ของเวกเตอร์เข้าด้วยกัน ดังตัวอย่าง ให้เวกเตอร์ 3 มิติมา 2 อัน เวกเตอร์ A <3,4,5> และเวกเตอร์ B <6,7,8> ผลบวกของเวกเตอร์คือ <9,11,13> เพราะ

บวก X เข้าด้วยกัน คือ 3+6 = 9

บวก Y เข้าด้วยกัน คือ 4+7 = 11

บวก Z เข้าด้วยกัน คือ 5+8 = 13

มันสำคัญที่จะแสดงว่า การบวกเวกเตอร์ คือ เอาเวกเตอร์ 2 อันมารวมกัน หากว่าจำความหมายของมันไม่ได้ก็จำเพียงว่าออร์เดอร์ (order) หรือกลุ่ม (group) ใช้ในการบวกเวกเตอร์นั้นแตกต่างกัน ผลบวกจะได้เท่ากัน คุณสามารถวาดภาพการบวกเวกเตอร์โดยลากเส้นจากหางของเวกเตอร์หนึ่งไปยังหัวของอีกเวกเตอร์หนึ่ง และวาดเส้นระหว่างทั้งสองเวกเตอร์ เวกเตอร์เส้นใหม่นี้จะแสดงผลบวกของเวกเตอร์ ผลบวกของเวกเตอร์จะแสดงหรือเขียนภาพประกอบได้ในสองมิติ ถ้าคุณสามารถสร้างภาพการบวกเวกเตอร์ 2 มิติได้ คุณก็สามารถทำอย่างเดียวกันได้ในเวกเตอร์ 3 มิติเช่นกัน

การลบเวกเตอร์(Vector subtraction)

การลบเวกเตอร์ก็ง่ายเหมือนการบวกเวกเตอร์ คุณสามารถทำการลบเวกเตอร์ด้วยการบวกจำนวนลบของเวกเตอร์หนึ่งไปอีกเวกเตอร์หนึ่ง เวกเตอร์ที่เป็นลบเป็นเวกเตอร์ที่มีขนาดเดียวกัน แต่ทิศทางตรงกันข้าม การทำการลบเวกเตอร์โดยลบแต่ละคู่ของเวกเตอร์ในแบบเดียวกัน ดังตัวอย่าง ผลลัพธ์ของการลบเวกเตอร์ A<3,4,5> ออกจากเวกเตอร์ B<6,7,8> จะได้ <3,3,3> เพราะ

ผลลบของ X คือ 6-3 = 3

ผลลบของ Y คือ 7-4 = 3

ผลลบของ Z คือ 8-5 = 3

การลบเวกเตอร์ 2 มิติ เป็นการลบกันของเวกเตอร์ 2 อัน เราวาดเวกเตอร์ทั้งคู่จากจุดกำเนิด และวาดเวกเตอร์ใหม่เป็นผลลัพธ์แสดงค่าแตกต่างจากหัวของเวกเตอร์หนึ่งไปยังหัวอีกเวกเตอร์หนึ่ง การลบเวกเตอร์ 3 มิติ ทำงานในลักษณะเหมือนกับเวกเตอร์ 2 มิติ และ V_1-V_2 ไม่เท่ากับ V_2-V_1 ถ้าเวกเตอร์นั้นมีค่ากลับกัน ขนาดของผลลัพธ์จะมีค่าเดียวกัน แต่ทิศทางตรงกันข้าม

การสเกลเวกเตอร์(Vector Scaling)

การกระทำของเวกเตอร์ที่สำคัญอีกประการหนึ่งคือการสเกลลิ่ง(Scaling) เมื่อเวกเตอร์เป็นเชิงปริมาณความยาวหรือขนาดของเวกเตอร์จะเปลี่ยนแปลงโดยบางส่วนประกอบ (factor) แต่ทิศทางยังคงเดิม การสเกลลิ่ง เป็นการคูณเวกเตอร์กับสเกลาร์(scalar) สเกลาร์ไม่ใช่ปริมาณเวกเตอร์ แต่เป็นจำนวนจริง ถ้าคุณให้เวกเตอร์เป็น V_1 และ scaler factor เป็น k จะเขียนผลลัพธ์เวกเตอร์ที่สเกลแล้วคือ V_2 ดังนี้

$$V_2 = kV_1$$

V_2 มีขนาดเป็น k เท่าของขนาด V_1 V_2 ชี้ในทิศทางเดียวกับ V_1 ถ้า k เป็นจำนวนบวกหรือลบ ในทิศทางตรงข้ามจะกระทำสเกลิ่งใน 3 มิติ

$$V_{2x} = kV_{1x}$$

$$V_{2y} = kV_{1y}$$

$$V_{2z} = kV_{1z}$$

ถ้า k มีค่าเป็น 2 และ V_1 คือ<3,4,5> V_2 จะมีผลเป็น <6,8,10>

การหาขนาดของเวกเตอร์(Vector Magnitude)

การหาขนาดของเวกเตอร์มีสัญลักษณ์คือ V จะใช้ค่าสัมบูรณ์เป็นสัญลักษณ์ เพราะขนาดของเวกเตอร์มักจะเป็นปริมาณค่าบวก ทางด้าน 2 มิติ จะแสดง เวกเตอร์ V เป็นด้านตรงข้ามมุมฉากของสามเหลี่ยมที่กำหนดด้าน x และ y โดยทฤษฎีพิธากอรัสสามารถนำมาใช้คำนวณด้านตรงข้ามมุมฉากของสามเหลี่ยมมุมฉากได้ หากรู้ความยาวของด้านประกอบมุมฉาก 2 ด้าน จะได้ $V = (x^2+y^2)^{1/2}$ ในเวกเตอร์ 3 มิติที่เริ่มจากจุดกำเนิด ขนาดของเวกเตอร์ 3 มิติ คือ

$$V = (V_x^2+V_y^2+V_z^2)^{1/2}$$

ถ้าไม่ใช่เวกเตอร์ที่เริ่มที่จุดกำเนิด เราสามารถคำนวณขนาดโดยใช้ผลลบของเวกเตอร์ คือ

$$V = ((V_x-V_{x0})^2+(V_y-V_{y0})^2+(V_z-V_{z0})^2)^{1/2}$$

<u>ยูนิตเวกเตอร์และเวกเตอร์ตั้งฉาก (Unit Vector and Normalization)</u>

เราใช้เวกเตอร์ 1 หน่วย อย่างกว้างขวางใน การติดตามรังสี และอีกมากมาย เพื่อบอกทิศทาง โดยไม่สนใจขนาด การเปลี่ยนเวกเตอร์ให้เป็นเวกเตอร์ 1 หน่วย เรียกว่า การหาเวกเตอร์ตั้งฉาก ("normalization") การหาเวกเตอร์ตั้งฉากเป็นการกระทำโดยใช้การคำนวณทางเวกเตอร์เป็นอย่างแรกและจากนั้น แบ่งแต่ละเวกเตอร์โดยขนาดของมัน การเปลี่ยน เวกเตอร์ $V<V_x,V_y,V_z>$ เป็น เวกเตอร์ 1 หน่วย สามารถแสดงการคำนวณได้ดังนี้

$$V = (V_x^2+V_y^2+V_z^2)^{1/2}$$

เวกเตอร์ 1 หน่วย $U = < \frac{V_x}{|V|}, \frac{V_y}{|V|}, \frac{V_z}{|V|} >$

ดังตัวอย่าง ถ้าเรามีเวกเตอร์ <10,-29,13> ขนาดของเวกเตอร์ = 33.32 เมื่อเปลี่ยนเป็นเวกเตอร์ 1 หน่วยเพื่อดูทิศทาง เวกเตอร์ 1 หน่วยจะเป็น < +0.3 ,-0.87 ,+0.39 > เวกเตอร์ 1 หน่วย กวาดไป 360 องศา ลากเส้น วงกลม 1 หน่วย ใน 2 มิติ ถ้ากวาดไปใน 3 มิติ จะได้ผลลัพธ์เป็นรูปทรงกลม 1 หน่วย

<u>การด็อทเวกเตอร์(Vector Dot Product)</u>

ในข้างต้น เราได้อธิบายถึงการสเกลลิ่งเวกเตอร์ การสเกลลิ่งเป็นรูปแบบหนึ่งของการคูณเวกเตอร์ โดยเอาเวกเตอร์คูณกับปริมาณสเกลาร์ ได้ผลลัพธ์เป็นเวกเตอร์ที่ขนาดเปลี่ยนแปลงไป การเอาเวกเตอร์ 2 ตัวคูณกันจะพบว่าใช้ในการติดตามรังสี คือการด็อทเวกเตอร์ และการครอสเวกเตอร์ การด็อทคือการคูณ 2 เวกเตอร์ โดยผลลัพธ์เป็นสเกลาร์ การครอสคือการคูณ 2 เวกเตอร์ แต่กลับได้เวกเตอร์ใหม่มาแทนสเกลาร์ การด็อทแสดงขนาดของเวกเตอร์ทั้งสองว่าชี้ไปทิศทางเดียวกัน ในทางคณิตศาสตร์ การด็อทแสดงโดย

A dot B = AB cos Theta (0<= Theta <=Pi)

สมมุติจุดกำเนิดของเวกเตอร์ทั้งสองทับกันสนิท ถ้าเวกเตอร์ A และ B ตั้งฉากกันผลลัพธ์การด็อทจะเท่ากับศูนย์ ถ้าเวกเตอร์ A เท่ากับเวกเตอร์ B ผลด็อทจะเท่ากับขนาดของเวกเตอร์ทั้งสองยกกำลังสอง นั่นเป็นเพราะว่า เวกเตอร์ทั้งสองมีทิศทางเดียวกัน เหมือนการบวกเวกเตอร์ การด็อทมีคุณสมบัติการสลับที่ คือ

A dot B = B dot A

ส่วนที่น่าสนใจของการด็อทเวกเตอร์ คือ ถ้าเวกเตอร์ A และ B เป็นเวกเตอร์ 1 หน่วย การด็อทจะแสดงมุมระหว่างเวกเตอร์ทั้งสอง ภายใต้สภาวะนี้ ผลด็อทจะมีค่าจาก 0 ถึง 1 ผลการด็อทสามารถคำนวณได้อย่างง่ายดายคล้ายผลบวกของ ผลลัพธ์ของเวกเตอร์ ดังนี้

Vector A < A_x ,A_y ,A_z >

Vector B < B_x ,B_y ,B_z >

A dot B = A_x*B_x + A_y*B_y + A_z*B_z

ผลการด็อทใช้อย่างกว้างขวางในระหว่างการคำนวณเรื่องเงา ใน การติดตามรังสี

<u>การครอสเวกเตอร์(Vector Cross Product)</u>

การครอสเวกเตอร์(Vector Cross Product) คือการคูณสองเวกเตอร์ และกลับมาเป็นเวกเตอร์ใหม่ เวกเตอร์ใหม่จะมีขนาดเท่ากับพื้นที่ของสี่เหลี่ยมขนมเปียกปูน ใน 3 มิติสามารถใช้ภาพแสดงผลการครอส เพราะเวกเตอร์ลัพธ์เป็นการตั้งได้ฉากกับพื้นราบ ในทางคณิตศาสตร์ การครอสคือ

A cross B = AB sin Theta (0 <=Theta <=Pi >

เราสามารถใช้กฎมือขวาแสดงส่วนของการครอสได้ ถ้าคุณชี้นิ้วชี้ของมือขวาไปในทิศทางของเวก เตอร์หนึ่ง และงอนิ้วที่เหลือในอีกทิศทางหนึ่ง นิ้วหัวแม่มือของคุณจะชี้ในทิศของเวกเตอร์ลัพธ์

A cross B ไม่เท่ากับ B cross A

แต่ A cross B = -B cross A

กล่าวอีกอย่างหนึ่ง เนื่องจากผลการครอสไม่มีคุณสมบัติการสลับที่เหมือนเวกเตอร์ตัวอื่นที่ผ่านมา จึงต้องใช้ความตั้งใจในการทำ cross product มากขึ้น คุณสามารถคำนวณ cross product โดยใช้อัลกอริทึม :

Vector A <Ax,Ay,Az>

Vector B <Bx,By,Bz>

Vector C <Cx,Cy,Cz> = A cross B

Cx = Ay*Bz - Az*By

Cy = Az*Bx - Ax*Bz

Cz = Ax*By - Ay*Bx

การหาผลครอสพบว่าใช้ในการติดตามรังสี เมื่อคำนวณถึงผิวนอกที่ให้เวกเตอร์ 2 ตัวที่จะแสดงที่พื้น ราบ ส่วนในด้านอื่น การครอสเวกเตอร์ บรรยายถึง ทอร์ค(torque), มุม(angular), โมเมนตัม(momentum) และ การไหลของพลังงานอิเลคโตรแมกเนติก ในโปรแกรม การติดตามรังสี ขั้นต้น เรานำเวกเตอร์ทั้งหมดกระทำกับ C มาโคร ซึ่งจะสร้างรหัสอ่านเหนือฟังก์ชั่นที่เรียก เวกเตอร์มาโคร มีในไฟล์ "vectors.h" บนดิสก์

<u>ภูมิศาสตร์การมอง(Viewing Geometry)</u>

ก่อนที่เราจะยิงรังสีเข้าไปในภาพ 3 มิติ เพื่อสร้างภาพการติดตามรังสีนั้น เราต้องเตรียมภูมิศาสตร์ การมอง ซึ่งจะเป็นความสัมพันธ์ระหว่างตำแหน่งตาของผู้มองกับวัตถุในภาพ ซึ่งจะจำกัดลักษณะของภาพที่

ตาสามารถมองเห็น สนามของภาพในจักรวาล ทั้งในทางแนวราบและในแนวตั้งจะอยู่ในภูมิศาสตร์การมอง จะแสดงท่าทางของภาพในระบบ 3 มิติ การเลือกภูมิศาสตร์การมอง ที่เหมาะสมทำให้ภูมิศาสตร์การมองสามารถชดเชยความเสียหายสำหรับความไม่จตุรัสของเม็ดสีบนจอคอมพิวเตอร์ได้

มีหลายทางที่จะแสดงภูมิศาสตร์การมองสำหรับโปรแกรมการติดตามรังสีทางหนึ่งของการติดตามรังสีที่เราเลือกใช้ก็เพราะว่าใช้ได้อย่างง่ายดายและปรับเปลี่ยนได้ ดังที่เราเห็นการเปลี่ยนแปลงของพารามิเตอร์การมองเพราะฉะนั้นภูมิศาสตร์การมองสามารถสร้างจำนวนมุมมอง(unique view) อย่างไม่จำกัดจำนวนของวัตถุในระบบ3 มิติได้ การจัดทำที่ดีของพารามิเตอร์การมอง(viewing parameter) จะได้ผลลัพธ์ที่ดีด้วย

ถ้าเรากล่าวให้เร็วยิ่งขึ้น คือ รังสีจะยิงจากตำแหน่งของตาผ่านระนาบการมอง (เคลื่อนย้ายเหนือเม็ดสีบนจอคอมพิวเตอร์) เข้าไปในบริเวณวัตถุ("object space") อาจจะตัดหรือไม่ตัดวัตถุในจักรวาล เมื่อรังสีตัดที่วัตถุ เม็ดสีที่ตรงกับรังสีจะให้สีของวัตถุที่มันตัด ความสัมพันธ์ระหว่างตำแหน่งของตาผู้มองกับระนาบการมอง สามารถมองเห็นได้ดังรูปที่ 1(a) ภาพสามารถแสดงในรูปด้านข้างของปิรามิดโดยระนาบการมอง เป็นฐานและตำแหน่งของตาเป็นรูปปิรามิดนี้เป็นสิ่งที่อ้างถึงฟรัสตัมการมอง(viewing frustum) มีเพียงวัตถุภายในฟรัสตัม ที่สามารถมองเห็นโดยตาของผู้มองได้ ส่วนวัตถุที่อยู่นอกฟรัสตัม นั้นจะมองเห็นผู้มอง ในขณะที่ การติดตามรังสี กำลังกระทำอยู่จะไม่มีรังสีที่สร้างเส้น อยู่นอกฟรัสตัมการมองเหตุผลสำหรับเรื่องนี้คือ วัตถุที่คลิป(clipping)นั้นไม่เป็นที่ต้องการเพราะรังสีได้แสดงสนามของภาพที่สมบูรณ์แล้ว เทคนิคกราฟฟิคอื่นๆต้องการคลิป เพื่อรักษาขอบเขตของวัตถุจากการออกไปของ view port เมื่อวัตถุอยู่ในระบบ 3 มิติมันจะอยู่บนด้านตรงข้ามของระนาบการมองจากผู้มอง

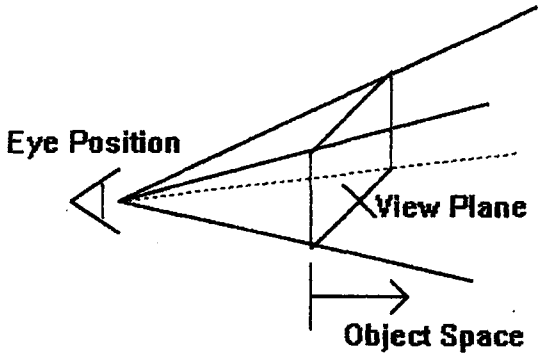


Fig 1 (a) Viewing Frustum

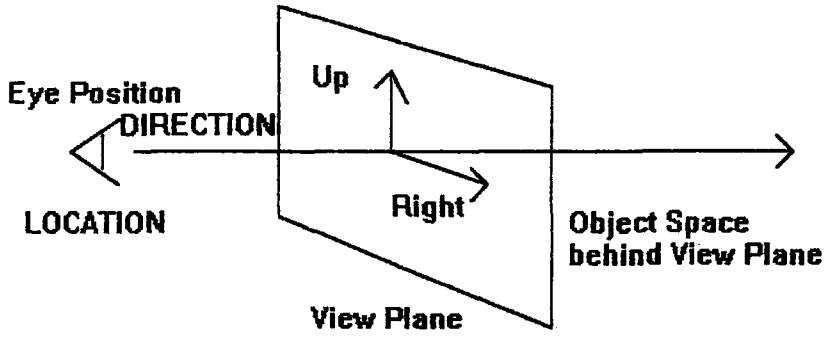


Fig 1 (b) Viewing Geometry

สำหรับการใช้ของเราระนาบการมอง เป็นฉากทั้งหมดของจอคอมพิวเตอร์ วัตถุ3มิติมองเห็นได้ในฟรัสตัมเป็นงานบนระนาบการมอง และผู้มองจะเห็นมันในการแสดงแบบ2 มิติ จากภาพ 3 มิติ คณิตศาสตร์ส่วนนี้จะอธิบายความสัมพันธ์ระหว่างที่ตั้งของตา,ระนาบการมอง และวัตถุ 3 มิติ ความสัมพันธ์นี้ก่อให้เกิด ภูมิศาสตร์การมอง(ภาพเรขาคณิต)

เราอาจใช้ เวกเตอร์ 4 อย่างเพื่ออธิบายเรื่องภูมิศาสตร์การมอง

1. ตำแหน่ง(location) ตำแหน่งแสดงที่ตาตั้งอยู่ในระบบ 3 มิติ ตำแหน่งของตาจะบอกถึงความเกี่ยวข้องกันที่วัตถุแสดงออก ตำแหน่งของตาอยู่บนแกน Z ด้านบวก

2. ทิศ(direction) ทิศแสดงความหมายของเวกเตอร์จากตำแหน่งของตาผ่านส่วนกลางของระนาบการมอง โดยไม่คำนึงถึงทิศทางที่ตามองอยู่ขนาดของเวกเตอร์จะบอกถึงระยะทางระหว่างตำแหน่งที่ตั้งของตากับระนาบการมอง

3. บน (up) เป็นเวกเตอร์ในระนาบมอง ที่เริ่มต้นที่จุดศูนย์กลาง และชี้ไปยังของของจุดสูงสุดของระนาบมอง

4.ขวา(right) เป็นเวกเตอร์ใน view plane ที่เริ่มต้นที่ศูนย์กลางและชี้ไปยังขอบด้านขวาของระนาบมอง

ความสัมพันธ์ของเวกเตอร์เหล่านี้แสดงในรูป 1(b) แขนของเวกเตอร์ทั้ง4 เราสามารถสร้างรังสีที่เราต้องการที่จะขีดเส้นเป็นภาพได้

ทุกกราฟฟิคที่แสดงผ่านเข้าไปในอะเรย์(array) ของเม็ดสี การแก้ปัญหาในการนี้คือ จำนวนของเม็ดสีที่มากมายในอะเรย์ สำหรับ PC กราฟฟิคที่แก้ไขเม็ดสีคือการนับจากซ้ายไปขวา ในทางแนวราบและจากบนลงล่างในแนวตั้ง เม็ดสีที่มีตำแหน่งระบุไว้ที่จุด (0,0)จะแสดงอยู่ที่มุมบนซ้าย ในขณะที่เม็ดสีที่ตำแหน่ง(x_{max},y_{max}) อยู่ที่มุมล่างขวาค่าของ x_{max}และy_{max} ยึดหลักบนวิธีการแสดงและการแก้ปัญหาที่ก่อให้เกิดประโยชน์

สำหรับ การติดตามรังสี ในลักษณะต่างๆ เราต้องการสมการของรังสีที่เกิดจากที่ตาผ่านเม็ดสีบนทางของวัตถุ อย่างแรกในส่วนนี้คือ การเปลี่ยนฉากสำหรับเม็ดสีในการแก้ปัญหาจำนวนอิสระที่เกี่ยวข้องกับศูนย์กลางของการแสดงที่พิจารณาว่า ค่า y ตรงไปยังฉากด้านล่างมากขึ้น กล่าวอีกทางหนึ่งคือ ให้ตำแหน่งของเม็ดสีระบุถึงฉากเราต้องเปลี่ยนไปเป็นจำนวนที่เกี่ยวข้องกับศูนย์กลางของฉาก(จุด(0,0)) การเปลี่ยนแปลงนี้เราจะใช้ผลลัพธ์ในแนว -0.5 ถึง +0.5 ทั้งในทิศทางของ x และ y ที่ -0.5 ในทิศทาง x อยู่ในด้านซ้ายของฉาก และ +0.5 อยู่ในด้านขวาในทิศของ y +0.5 อยู่บนสุดของฉาก จุด(0,0) แสดงถึงศูนย์กลางของฉาก การเปลี่ยนแปลงกระทำดังนี้

$$\text{Screen}_x = \frac{\text{Pixel}_x - \text{Display Width} / 2}{\text{Display Width}}$$

$$\text{Screen}_y = \frac{(\text{Display Height} - 1) - \text{Pixel}_Y - \text{Display Height} / 2}{\text{Display Height}}$$

เมื่อภาพการแก้ปัญหาถูกสร้างขึ้น การเปลี่ยนโคออร์ดิเนต (coordinate) ที่เม็ดสีใกล้เคียงจะทำก่อนที่ติดกันในขณะที่เหลืออยู่ในแนวที่ระบุไว้ การเปลี่ยนแปลงฉากโคออร์ดิเนตแสดงถึงทางเบี่ยงของที่ตั้งของเม็ดสีจากศูนย์กลางถึงฉาก

สนามในแนวราบและแนวตั้งของฉากจะจำกัดโดยสเกลลิ่ง ของ srceen x และ screen y มีค่าโดยเวกเตอร์ ขึ้นและขวาตามลำดับ ถ้าขนาดของขึ้นและขวานอนในแนวระนาบการมองเรารวมมันเพื่อควบคุมตำแหน่งที่รังสีจากตำแหน่งของตาจะแทงผ่านระนาบการมองสุดท้ายในการสร้างรังสีคือบวกทิศทางเวกเตอร์เข้ากับ เวกเตอร์ทางคณิตศาสตร์ ผลลัพธ์ในรังสีที่เราออกแบบจากตำแหน่งของตาผ่านฉากเข้าไปในวัตถุ ทิศทางของรังสีจะเปลี่ยนไปเป็นเวกเตอร์ หนึ่งหน่วย และจุดกำเนิดของรังสีจะตั้งในตำแหน่งของตาก่อนที่รังสีจะใช้ในการคำนวณภาพหลังทุกครั้งที่รังสียิ่งเข้าไปในระบบ 3 มิติ จะสร้างในวิธีนี้

โปรแกรม การติดตามรังสี เกือบทั้งหมด สร้างภาพเม็ดสีในเวลามาตรฐาน จากซ้ายไปขวาและจากบนลงล่าง กระทำโดยใช้การวนลูป ลูปภายนอกสำหรับแถวหรือค่า y และลูปภายในสำหรับคอลัมน์ หรือ ค่า x ค่า x และ y ที่จุดใดๆ ในเวลาที่กำหนดจะสามารถเปลี่ยนแปลงและค่า xและ y ที่ผ่านจุดสามารถสร้างได้

แบบจำลองสำหรับระนาบการมอง มีความยืดหยุ่นมากงอไปมาได้ นั่นเป็นเพราะว่ามันไม่ได้อยู่ในอาณัติที่เวกเตอร์ขึ้น, ขวาและทิศทาง อยู่ที่มุมที่ถูกต้อง เวกเตอร์เหล่านี้สามารถใช้ในทางที่ทำนายผลไว้ล่วงหน้าหรือทางที่ไม่ได้คาดหวังไว้ ที่จะได้ผลลัพธ์ที่เสียรูปไป ผลบางอย่างที่แปลกออกไป จะเกิดขึ้นได้โดยการจัดทำ vecter up และ right เมื่อใช้ในวิธีการธรรมดา เวกเตอร์เหล่านี้จะควบคุมสนามในแนวนอนและแนวตั้งของภาพและวัตถุอีกทั้งยังสามารถใช้อย่างถูกต้องสำหรับปัญหาการเปรียบเทียบลักษณะด้วย

สนามเวกเตอร์ของภาพได้ผลลัพธ์ในเทเลโฟโต้(telephoto)คล้ายกับภาพของวัตถุ ขนาดที่ใหญ่กว่าอีกผลลัพธ์หนึ่งในมุมกว้าง การเปรียบเทียบขนาดของเวกเตอร์ 2 อันนี้มีความสำคัญ เมื่อใช้วิธีที่แสดงจุดภาพจตุรัส(square pixel)ที่เป็นประโยชน์ การเปรียบเทียบของเวกเตอร์เหล่านี้จะควบคุมสนามของภาพซึ่งสามารถได้ผลลัพธ์ที่ถูกต้องได้ กล่าวอีกอย่างหนึ่งจุดภาพจตุรัส คล้ายกับการแสดงที่มีจุดภาพไม่จตุรัสสิ่งนี้หมายถึง ส่วนที่แสดงนี้จะปรากฏภาพวงรีเมื่อใช้จุดภาพไม่จตุรัสวีดีโอ VGA โหมด 13hex เป็นตัวอย่างของวีดีโอ วิธีที่จริงที่มีจุดภาพไม่จตุรัสบนจอภาพ ขนาด 14 นิ้ว mode 13 hex pixel มีความกว้างประมาณ 0.027 นิ้วและยาว 0.036 นิ้วถ้ารูปทรงกลมมีการกระทำในวิธีที่ไม่ถูกต้อง จะปรากฏการรีดให้เรียบในแนวตั้งคือทำให้ความกว้างมากกว่าความยาวการแก้ไขปรากฏการณีนี้ทำได้โดยทำ right vector ให้ยาวขึ้น จากขนาดปกติ 1.00 ถึง 1.3333ภาพส่วนมากจะมีต่อเม็ดสี ในทิศแนวราบมากกว่าแนวตั้ง

ขนาดของทิศทางเวกเตอร์จะเป็นตัวเลขในการคำนวณของสนามของภาพ คุณสามารถใช้เวกเตอร์ขึ้นและขวา สำหรับการเปรียบเทียบส่วน ถ้าหากต้องการใช้ขนาดของทิศทางเวกเตอร์ สำหรับควบคุมสนามของภาพ

คุณสามารถสร้างผลลัพธ์อื่นๆได้จาก การจัดทำ เวกเตอร์ ขึ้น และ ขวา ในการกลับทิศของภาพ เวกเตอร์ขึ้น จะใช้ <0 1 0 > เปลี่ยนเป็น < 0 -1 0> ในทางเดียวกัน ภาพซ้าย และ ขวา จะกลับด้านโดยเปลี่ยนเวกเตอร์ ขวา จาก <1 0 0> เป็น <-1 0 0>

สมการพาราเมตริคของรังสี(Parametric Equation of Rays)

โดยชื่อของการติดตามรังสีรังสีเป็นหลักของเทคนิคนี้รังสีเป็นขนาดพื้นที่ของเวกเตอร์ที่นำมาถกปัญหา ซึ่งมีทั้งทิศทางและจุดกำเนิด ตำแหน่งของเวกเตอร์จะเห็นอยู่ที่จุดกำเนิด (0,0,0) ในระบบ 3 มิติ รังสีทำจากเวกเตอร์ สองอัน อันหนึ่งบอกถึงทิศทาง อีกอันบอกถึงจุดกำเนิด ถ้าให้จุดเริ่มต้น และ ทิศของรังสีก็จะคำนวณถึงแนววิ่งของรังสี ว่ารังสีเคลื่อนที่ไปได้ทางไกลเพียงใด ถ้าให้เวลาเป็น "t" แนววิ่งของรังสีจะเป็น

Trajectory of Ray R = direction * t +-origin

ในทางคณิตศาสตร์ อธิบายเพิ่มได้ว่า

$R(t) = R_d * t + R_0 \quad \text{for } t > 0$

R(t) เป็นกลุ่มของจุดที่สร้างแนววิ่งของรังสี สมการนี้มีสถานะเป็น 3 มิติ ที่น่าสนใจ สมการของจุดที่รังสีผ่านใน 3 มิติ เป็น ฟังก์ชันของ t ได้

$P_x = X_d * t + X_0$

$P_y = Y_d * t + Y_0$

$P_z = Z_d * t + Z_0$

ทิศของรังสีบอกโดย เวกเตอร์ $<X_d,Y_d,Z_d>$ และที่จุดกำเนิด เป็น $<X_0,Y_0,Z_0>$

เราใช้รูปแบบที่ชัดเจนของสมการพาราเมตริคของรังสี (Parametric ray)

การใช้สมการพาราเมตริคของรังสี ที่เกี่ยวข้องกัน ถ้ามีการตัดกันเกิดขึ้นระหว่างรูปธรรมดา และรังสี

การคำนวณจุดตัดของรังสีและระนาบ(Ray/Plane Intersection Calculations)

ส่วนของทรงกลมที่เราจะพิจารณาต่อไปคือ พื้นที่ราบ ซึ่งเป็นเรขาคณิตที่มีอยู่เป็นการแบ่งพื้นที่ออกเป็น 2 ส่วน พื้นที่ราบไม่เหมือนทรงกลมเพราะว่าไม่มีขอบเขตจำกัด รูปทรงกลมนั้นแตกต่างที่มีการกั้นขอบเขตที่ใช้คำนวณรังสี ภาคตัดกรวยของพื้นราบจะง่ายกว่า คำนวณรังสี ภาคตัดกรวยของทรงกลมเสียอีก เราใช้สมการของพื้นราบ โดย

$Ax + By + Cz + D = 0$

จำนวนที่แสดงพื้นราบ A, B, C, D สามตัวแรกเป็นจำนวนเลข A,B,C ใช้เวกเตอร์หนึ่งหน่วย ในพื้นราบ เพราะฉะนั้น $A^2 + B^2 + C^2$ ต้องเท่ากับหนึ่ง ส่วน D ในสมการแสดงระยะทาง ของพื้นราบจากจุดกำเนิด (0,0,0) เพื่อคำนวณการตัดซึ่งกันและกันเราเปลี่ยนรูปแบบของสมการพาราเมตริคของรังสี เป็น สมการพื้นราบ และ แก้สมการเพื่อหาค่า t

$A*(X_d*t + X_0) + B*(Y_d*t + Y_0) + C*(Z_d*t + Z_0) + D = 0$

จะได้

$$t = -\frac{(A*X_0 + B*Y_0 + C*Z_0 + D)}{A*X_d + B*Y_d + C*Z_d}$$

ถ้าเราตั้งให้สมการนี้เท่ากับศูนย์ รังสีและพื้นราบจะขนานกันหรือนอนในพื้นเดียวกัน และไม่มีการตัดกันเกิดขึ้น การคำนวณสามารถหยุดที่จุดนี้ ถ้าเราให้สมการนี้ไม่เท่ากันศูนย์ การคำนวณจะมีต่อไป ถ้าให้

สมการนี้มีค่ามากกว่าศูนย์ ระนาบจะชี้ไปในทิศเดียวกับรังสี และอาจจะมีการย้อนกลับได้

การแก้ปัญหาของสมการนี้ ค่า t จะคำนวณในลักษณะตรงไปตรงมา เหมือนเช่นสมการเชิงเส้น A,B,C,D แสดงถึงพื้นราบ ในขณะที่ X_d,Y_d,Z_d,X_0,Y_0,Z_0 แสดงโดยรังสี ถ้า t น้อยกว่า 0 จะไม่มีการตัดกันปรากฏขึ้น และการคำนวณสามารถหยุดได้อีกครั้ง ถ้าเป็นจำนวนบวกจะพบการตัดซึ่งกันและกัน จุดที่ตัดจะนำมาคำนวณโดยเปลี่ยนค่าจาก หนึ่งไปเป็น สมการของรังสี และแก้ปัญหาสำหรับ P ผิวของพื้นราบไม่ได้ชี้ในทิศทางที่เหมาะสมแล้ว ที่เราต้องการคือชี้กลับไปในทิศทางที่เป็นจุดกำเนิดของรังสี สภาวะที่มีการเปลี่ยนทิศไปข้างหลังนี้จะแสดงเมื่อมีการตั้งสมการที่มีค่าตัวเลขมากกว่าศูนย์

<u>การแรเงา(Shading)</u>

ปัญหาที่ยุ่งยากปัญหาหนึ่งของ การติดตามรังสี คือ การแรเงา("shading") การแรเงาเป็นกรรมวิธีที่ใช้สีกำหนดให้กับวัตถุที่สีจากตานั้นตัด(intersection) และการแปลสีที่จะแสดงบนจอคอมพิวเตอร์ ความคิดของการแรเงาคือ ผิวส่วนต่างๆเกือบทั้งหมดของ การติดตามรังสี แบบจำลองการแรเงาใช้ในการอธิบาย การตัดของแสงโดยผิวของสิ่งของต่างๆ เรื่องนี้จะเป็นความจริงอย่างยิ่งถ้าแบบจำลองของการแรเงาใช้ในการพิจารณาความถี่ของแสงและมุมของการตัดซึ่งกันและกัน ผิวของเงาชี้ว่าเป็นการยากเพียงใดที่คอมพิวเตอร์จะเลียนแบบได้เหมือนธรรมชาติ การตัดกันของแสงและผิว มีคุณสมบัติที่ดีที่มีความเหมือนของภาพปรากฏออกมา ระบบตาและสมองของคนใช้คิว(cues) เพื่อจำกัดเกี่ยวกับการมองเมื่อคิวเกิดการผิดพลาดสมองจะสามารถบรรจุเติมลงไปได้ในบางครั้ง แต่การสูญเสียก็ยังสังเกตุเห็นได้

เทคนิคบางอย่างเป็นพื้นฐานการมองแบบเรขาคณิต และฟิสิกส์ เราจะทำความเข้าใจว่าเงาคืออะไร ความหมายของเงาเป็นการคำนวณเรื่องสีและความเข้มของแสงที่ออกจากผิวและเดินทางกลับมาสู่ตาสำหรับการมอง สีของแสงที่สังเกตุเห็นโดยตาเป็นการรวม เข้าด้วยกันของสีผิวของวัตถุ สีของแสงต่างๆ จะสะท้อนกลับโดยวัตถุและสีของแสงที่ส่งผ่านวัตถุเงาไม่ใช่การเคลื่อนไหวของพื้นผิวที่ซ่อนอยู่ ความจริงคือรังสีจากตามักจะตัดกับวัตถุอย่างใกล้ชิดซึ่งหมายความว่า เราจะประยุกต์ใช้กับผิวในแถวหน้าและการมองเห็นภาพ เงาไม่ใช้กับวัตถุในที่มืด

หลังการคำนวณเรื่องเงาสำหรับฉากแล้ว เราพบระบบคอมพิวเตอร์กราฟฟิค PC ในปัจจุบันไม่มีกามุต(gamut) ของสีที่จำเป็นสำหรับแสดงการคำนวณ เราต้องสร้างสัญลักษณ์ ในส่วนของเงาและในส่วนการแสดงผล โปรแกรม การติดตามรังสี เป็นอิสระจากการแสดงฮาร์ดแวร์ มีการระบุสีใน RGB (Red,Green และ Blue) ร่วมอยู่กับสีอื่นๆ ซึ่งคล้ายกับจุดที่อยู่ในระยะ 0 ถึง 1 ใกล้เคียงกับจำนวนอินฟินิตของสี ที่แสดงความเข้มสูงสุด ระบบสี RGB เป็นการรวมระบบที่สีไม่ได้อยู่ในนั้น ดังแสดงโดย R=G=B=0 แสดงสีดำ และสีเป็นการสร้างโดยบวกค่าของสี ดังตัวอย่าง R=1,G=B=0 จะแสดงสีแดงเป็นค่าความเข้มสูงสุด ถ้าสีทั้งหมดมีสีเท่ากัน ค่าที่แสดงจะเป็นสีเทา สีที่มีค่ามากที่สุดและเท่ากัน R=G=B=1 แสดงสีขาวค่าสี RGB ของ I*R, I*G, I*B เมื่อ I อยู่ระหว่าง 0 และ 1 พิจารณาจากเงาของสี RGB ในความเข้ม I เงาของสีเทาสามารถพิจารณาได้เหมือนเงาของสีขาวธรรมชาติของระบบสี RGB เป็นประโยชน์ดังแสดงการอธิบายในส่วนของเงา

ในส่วนของเงา เมื่อเราลากเส้น รังสีแสง ขึ้นในฉากซึ่งเป็นการแทนด้วยเม็ดสีและการแสดงความเข้ม ทิศทางทั้งสองของรังสีและคุณสมบัติของพื้นผิวจะใช้ในการพิจารณาต่อไปเมื่อรังสีแสง ตกกระทบที่พื้นผิวของวัตถุ เงาจะบอกเราว่าแสงผ่านไปเท่าไรหรือการสะท้อน (propagated) จากผิว ย้อนกลับไปยังผู้มอง การสะท้อนสามารถแตกออกเป็น 2 อย่าง คือ การสะท้อนส่องผ่านอย่างเป็นระเบียบ (specular propagation) และการสะท้อนแบบกระจัดกระจาย (diffuse propagation) รังสีที่สะท้อนอย่างเป็นระเบียบอยู่ในทิศตะวันออก ในขณะที่รังสีที่สะท้อนกระจายเป็นแสงที่เท่ากันในทุกทาง โดยไม่มีความสัมพันธ์ในทิศของแสงทั้งเป็นระเบียบและไม่เป็นระเบียบ จะต้องผ่านเกี่ยวกับการสะท้อนกลับและการส่งผ่าน การสะท้อนกลับเป็นสิ่งที่เกิดขึ้นเมื่อคุณมองในกระจก การส่งผ่านเป็นสิ่งที่เกิดขึ้นเมื่อคุณมองผ่านผิวโปร่งใส(transparent) หรือ โปร่งแสง(semi-transparent) แสงที่ออกจากตาของคุณจะไปถ่ายทอดผ่าน (transmitted) ผิวของวัตถุ ตัวอย่างการส่งผ่านของแสงคือ การมองผ่านหน้าต่างกระจก หรือมองปลาในน้ำสะอาด

มีการสะท้อนของแสง(light propagation) 4 อย่าง ที่จะอธิบายเรื่องแบบจำลองของเงา บางครั้งแสดงในรูปแบบกราฟฟิค เหมือนแบบจำลองการเดินทางของแสงทั้ง 4 แบบ กลศาสตร์ทั้ง 4 แบบ จะอธิบายถึงแสงที่มาจากแหล่งกำเนิดแสง และแสงที่มาจากวัตถุอื่นในฉาก โปรแกรม การติดตามรังสี ส่วนมากจะทำให้ง่ายลงโดยอยู่ในรูปแบบจำลองของเงา ซึ่งแสดงบางส่วนได้ดังนี้

1. การเคลื่อนที่ที่ขึ้นอยู่กับความถี่

ในการแสดงถึงแสงและพื้นผิว ความถี่ของแสงเป็นตัวกำหนดว่าแสงทำงานบนวัตถุอย่างไร ดังตัวอย่างทำงานเพราะจำนวนแสงที่เบี่ยงเบน โดยอาศัยความถี่ของตัวมัน

2. การสะท้อนภายในวัตถุและการส่งผ่านที่สนใจ

วัตถุสะท้อนและมีการช่วยส่งผ่านในฉากสว่าง เพราะวัตถุส่วนมากจะสะท้อนแสง มันช่วยให้แสงไปถึงวัตถุอื่นๆในฉาก แสงบางส่วนพบทางด้านหลังของตาทางหนึ่งของการคำนวณอ้อมฉาก คือเทคนิคที่เรียกว่า เรดิโอสิตี้ (radiosity) ซึ่งใช้กฎของแรงอนุรักษ์

3. ระยะทาง

แสงเดินทางผ่านช่องว่างจะถูกแบ่งเบาเหมือนกำลังสองของระยะทาง แบบจำลองของเงาบางส่วนไม่นำระยะทางระหว่างแหล่งกำเนิดแสงและวัตถุ หรือระยะทางระหว่างวัตถุที่สะท้อนแสงถึงวัตถุอื่นๆในการพิจารณา เรา

นำระยะทางมาพิจารณาโดยหารการคำนวณความเข้มของแสงโดยระยะทางกำลังสอง แสงจะปรากฏให้เห็นอย่างชัดเจน เราใช้เรื่องเงาในโปรแกรม การติดตามรังสี เบื้องต้น โดยไม่สนใจกับระยะทางเลย เมื่อออกจากแบบจำลองของเงา มีการแสดงการลากเส้นเบื้องต้นในแบบ

- Ambient lighting

  การส่องแสงออกไปรอบทิศ

- Diffuse reflection

  การสะท้อนแสงแบบกระจัดกระจาย

- Specular reflection

  การสะท้อนแสงแบบเป็นระเบียบ

- Specular transmission

  การส่งผ่านแสงแบบเป็นระเบียบ

<u>การส่องแสงรอบทิศทาง(Ambient Lighting)</u>

การกล่าวถึงการส่องแสงรอบทิศทาง(ambient lighting) เป็นการประดิษฐ์ที่หมายความถึงการขด เขยสำหรับการส่องสว่างภายในวัตถุ วิธีนี้ปรากฏในธรรมชาติโดยไม่จำเป็นต้องใช้การคำนวณรังสีของการ ส่องแสงรอบทิศสามารถที่จะตกกระทบวัตถุจากทุกทิศทางและสะท้อนออกไปในทุกทิศทาง ความหนา แน่นของการส่องแสง สะท้อนไม่อิสระกับทิศทางไปยังผู้สังเกตุหรือทิศทางของแหล่งกำเนิดแสง(s) การส่อง แสงสามารถคำนวนได้ในหนึ่งหรือสองทาง ทางหนึ่งคือ สูตรสำเร็จรูป

$$I_a = k_a * I_1$$

ซึ่ง I1 คือความหนาแน่นของแสงที่ส่องของต้นกำเนิด และ k_a ค่าคงที่การดูดกลืนค่าคงที่ k_a กำหนด จากจำนวนแสงส่องที่จะถูกสะท้อนจากผิวหน้า ปัญหากับการคำนวณวิธีนี้คือ ค่าของแสงที่ส่องสะท้อน ออกจากผิวหน้าวัตถุ คือฟังก์ชันของสีของแสง ไม่ใช่สีของวัตถุ คุณสามารถได้ปรากฏการณ์ที่เหมือนจริง มากกว่าโดยใช้การคำนวณแสงที่ส่องต่างๆ ดังกรณีนี้

$$I_a = k_a * I_0$$

ซึ่ง I0 คือสีของผิวหน้าก่อน และ k_a กำหนดจากจำนวนสีของผิวหน้าเห็นได้จากการส่องแสง ค่าของ k_a คือ 0.4 ด้วยวิธีนี้ วัตถุที่ปราศจากฉากจะถูกส่องแสงโดยแสงรอบทิศเท่านั้น จะแสดงเงาของสีจริงเหล่านั้น

สมการข้างบนใช้สำหรับแสงสีเดียว ทำให้เกิดประโยชน์กับมันและสมการอื่นๆ ที่ไม่มีส่วนนี้ คุณจะ ต้องประยุกต์มันกับองประกอบของสีทั้ง 3 สี (RGB) โดยแยกออกจากกัน สูตรสำหรับการให้สีจะเป็นดังนี้

$$I_{aRed} = K_{aRed} * I_{oRed}$$
$$I_{aGreen} = K_{aGreen} * I_{oGreen}$$
$$I_{aBlue} = K_{aBlue} * I_{oBlue}$$

ถึงแม้รู้ดีว่าผิวหน้าของวัสดุจะดูดกลืนแสงความถี่ต่างๆ (สีที่แตกต่างกัน) ในระดับที่แตกต่างกัน ในความ เป็นจริงแล้ว ทำได้ง่าย

<u>การสะท้อนแสงแบบไม่เป็นระเบียบ(Diffuse Reflection)</u>

แสงที่กระจัดกระจายอันเกิดจากการสะท้อนของแสง จะสะท้อนในทุกทิศทางด้วยความหนาแน่น เท่าๆกัน ทฤษฎีที่ใช้ปรากฏการณ์นี้คือ การที่โฟตอนของแสงกระทบผิวหน้า กับคุณสมบัติการสะท้อนที่ แพร่กระจายมันจะถูกดูดกลืนเพียงชั่วครู่โดยผิวหน้า การเพิ่มขึ้นของพลังงานถูกสะสมโดยอะตอมในผิว หน้าจะสูงขึ้นเพียงชั่วขณะเดียวแต่ไม่สามารถถูกทำเรื่อยไป ในที่สุดผิวหน้าก็ปลดปล่อยโฟตอนให้ต่ำลง จนกระทั่งถึงระดับพลังงานต่ำสุด สู่สภาวะเสถียร โฟตอนจะถูกแพร่จากผิวหน้าในทิศทางอย่างสุ่ม ไม่มี ความสัมพันธ์กับมุมตกกระทบของโฟตอนที่เข้ามา การสะท้อนที่แพร่กระจายของแสงไปยังผิวหน้า จะมี ปรากฏการณ์เหมือนกับการมองของตำแหน่งของผู้สังเกตุด้วยเหตุนี้ ทิศทางของตาของผู้สังเกต ไม่สามารถ นำไปสู่การคำนวณ อะไรที่สำคัญสำหรับความสัมพันธ์ระหว่างรังสีจากผิวหน้าไปยังแหล่งกำเนิด แสงและระนาบปรกติ แอมปลิจูดของแสงแสงสะท้อนที่แพร่กระจาย เป็นสัดส่วนกับค่า cosine ของ มุมระหว่างเส้นตกกระทบกับเส้นปรกติ เป็นความสัมพันธ์ของ " Lambert's Law " ถ้ารังสี L และเส้นปรกติ

N เป็นเวกเตอร์หนึ่งหน่วย ค่า cosine ของมุมระหว่างมันคือ ค่าผลการด็อท ถ้า N dot L น้อยกว่าหรือเท่ากับศูนย์ ผิวหน้าจะอยู่ทิศที่หันหน้า(face away)จากแหล่งกำเนิดแสง ดังนั้นไม่ใช่ทั้งหมดของแสง ที่กระทบกับผิวหน้าแล้วถูกสะท้อน , ค่าคงที่ของการดูดกลืนอื่นๆ ,Kd ถูกแนะนำ ,นอกจากนี้สมการโมโนโครมาติค (monochomatic) สำหรับใช้สนับสนุนสมการการแรเงา(shading equation) ทั้งหมดของการสะท้อนคือ

$$I_d = I_l * I_d * \cos Theta = I_l * I_d * N \text{ dot } L$$

สำหรับฉากที่ถูกแสงจากต้นกำเนิดหลายอัน ผลรวมของแสงจากต้นกำเนิดทั้งหมดควรถูกใช้แหล่งกำเนิดแสงหลายอันถูกรักษาประหนึ่งว่า แต่ละต้นกำเนิดคือ ต้นกำเนิดแสงที่ปราศจากฉากและ ก่อนหน้านี้ สมการ monochomatic จำเป็นต้องถูกแก้ปัญหา 3 ครั้ง เพื่อใช้กับโมเดลสีRGB (RGB Color Model) เช่นเดียวกับ Kd จะมีค่าแตกต่างกันของการดูดกลืนสำหรับแต่ละสี 3 สี ในทางปฏิบัติแสงเงา(shader) จำนวนมากเปลี่ยนค่าที่แตกต่างกันของ Kd โดยแนะนำสีของผิวหน้าที่ถูก shaded ไปเป็นสมการ

$$k_{dRed} = k_{d'} * I_{oRed}$$
$$k_{dGreen} = k_{d'} * I_{oGreen}$$
$$k_{dBlue} = k_{d'} * I_{oBlue}$$

ซึ่ง Kd' คือค่าคงที่ของการดูดกลืนถูกใช้ใน 3 กรณี และ I_0 คือสีของผิวหน้าวัตถุ โดยสัญชาติญาณ ทำให้รู้ว่า ปริมาณของแสงที่ถูกดูดกลืนโดยผิวหน้าขึ้นอยู่กับสีของแสง และสีของผิวหน้า ตัวอย่าง ถ้าแสงสีขาวถูกทำให้เกิดการสะท้อนออกจากผิวหน้าสีแดง การสะท้อนของแสงจะปรากฏเป็นสี แดงเพราะว่าสีน้ำเงินและเขียว ซึ่งเป็นส่วนประกอบของแสงจะถูกดูดกลืน (ถูกกรองออกไป) โดยผิวหน้าโดยใช้ปัจจัยในการคูณของสีของแสง โดยสีของผิวหน้าในการคำนวณเหล่านี้ ปรากฏการณ์ของการกรองทำงานดังที่ได้ถูกคาดหมายไว้

<u>การสะท้อนแสงแบบเป็นระเบียบ(Specular Reflection)</u>

การสะท้อนแบบสุ่มถูกแสดงโดยพื้นผิวเรียบ ถ้าผิวหน้ามีความเรียบเพียงพอ ภาพของแสงแบบสุ่มจะปรากฏ แสงอย่างสุ่มปรากฏบนผิวหน้าเหมือนกับรอยปุที่สว่างของแสง ซึ่งก็คือ สีของแสงที่ถูกสะท้อน ผิวหน้ายิ่งเรียบเท่าไร แสงที่ปรากฏก็จะมีความหนาแน่นมากขึ้นเท่านั้น แสงจะไม่ได้มีสีเดียวกับสีของผิวหน้า เพราะว่าโฟตอนของแสงที่กระทบกับผิวหน้าไม่ได้ถูกดูดกลืน และแพร่กระจายกลับโดยผิวหน้าเหมือนกับกรณีของการสะท้อนแบบแพร่กระจาย(Diffuse Reflection) กลับกัน มันกระทบกับผิวหน้าแข็งๆ เพียงชั่วขณะเดียว ด้วยมุมสะท้อนเท่ากับมุมตกกระทบ มันทำงานเหมือนกับ ลูกบิลเลียด กระทบกับด้านหนึ่งของโต๊ะบิลเลียด หรือหินถูกขว้างด้วยมุมแคบๆ ลงในแอ่งทะเลสาบ

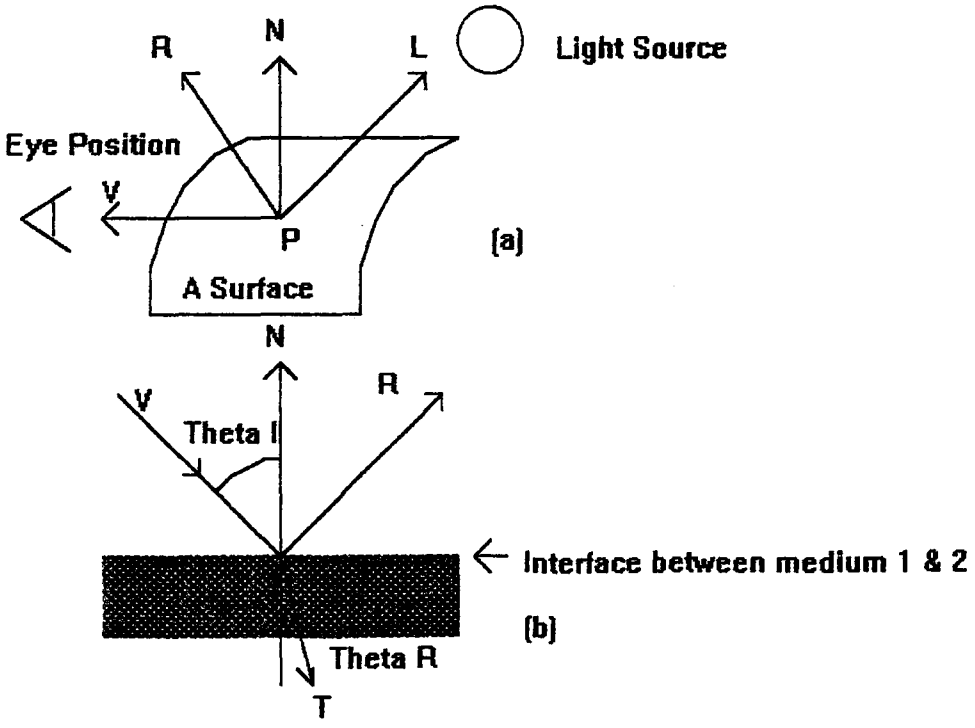
ไม่เหมือนกับการสะท้อนที่แพร่กระจาย การสะท้อนแบบสุ่มกับสีบนผิวหน้าแลเห็นโดยผู้สังเกตุในทิศทางที่สูงกว่า เพื่อที่จะเข้าใจว่าการสะท้อนแบบสุ่มทำงานอย่างไร คุณจะต้องพิจารณารูป 2.6 (a) แสง

สว่างที่เกิดแบบสุ่ม มีสาเหตุจากแสงจากต้นกำเนิดถูกสะท้อนออกจากผิวหน้าในทิศทางตาของผู้สังเกตุ มุมระหว่างการสะท้อนของแสงจากแหล่งกำเนิด (R) และรังสีไปยังผู้สังเกต(V) เป็นตัวกำหนดจำนวนสีที่สะท้อนออกจากผิวหน้าอันเกิดจากการสะท้อนแบบสุ่ม เมื่อเวกเตอร์ R และ V ทับกันสนิทจริงๆ จะเกิดผลมากที่สุด เหมือนกับเวกเตอร์เหล่านี้มี ทิศทางแยกจากกันจะทำให้เกิดผลน้อยลง ซึ่งก่อนหน้านี้ ผลคูณเชิงสเกลลาร์ถูกใช้ในการกำหนดค่า cosine ของ R และ V ว่าใกล้เคียงกันเพียงใด เพราะว่ามุมระหว่างมุมระหว่างมันทำให้เราสนใจ

<u>ข้อสังเกต</u> สำหรับผลคูณเชิงสเกลลาร์ใช้เปรียบเทียบการทำงาน ทั้ง R และ V จะต้องเป็นเวกเตอร์หนึ่งหน่วย แต่ R มาได้อย่างไร สำหรับรังสีที่ให้มากับรังสีที่ตัดกัน เราจะได้จุดตัด P และ เวกเตอร์สังเกตุ V จากเหล่านี้ เราสามารถคำนวณหาพื้นผิวปรกติ N โดยรู้ชนิดของวัตถุที่ถูกตัดผ่าน แล้วสร้างรังสี L จากจุดตัดไปยังแหล่งกำเนิดแสง L จากข่าวสารที่ให้และความจริงที่ว่า เวกเตอร์ R, N, และ L จะอยู่บนระนาบเดียวกัน และความจริงที่ว่า มุมตกกระทบ เท่ากับมุมสะท้อน เวกเตอร์หนึ่งหน่วย R ที่สะท้อนสามารถถูกคำนวณได้จาก

$$R = 2 * N *(L \text{ dot } N) - L$$

สมมุติเวกเตอร์ถูกแสดง ดังรูป



ขณะนี้เรามีทั้งเวกเตอร์ V และ R เราสามารถกำหนดความหนาแน่นที่มากที่สุดของแสงแบบสุ่ม สำหรับจุดที่ถูก shade แต่ที่กล่าวถึงก่อนหน้านี้ ขนาดของแสงสว่างคือ ฟังก์ชันของความเรียบของผิวหน้า ไม่ใช่เพียงเรขาคณิตของผู้สังเกตุกับมุมสะท้อน โมเดลของการสะท้อนของพฟอง(Phong) ถูกใช้ปรับปรุงปรากฏการณ์นี้ ด้วยโมเดลของพฟอง(Phong) แฟกเตอร์ n ตัวใหม่ถูกแนะนำ ซึ่งแสดงลักษณะผิว

หน้าของวัตถุ ยิ่งค่า n ใหญ่เท่าไร ผิวหน้าก็จะมีความมันเท่านั้น ถ้า n เท่ากับ อนันต์ ผิวหน้าจะเรียบแบบสมบูรณ์ รวมโมเดลของ Phong กับ สมการสำหรับการสะท้อนแบบสุ่ม จะได้ดังนี้

$$I_s = I_l * k_s * (R \text{ dot } V)^n$$

ค่าที่ใหญ่ของ n ทำให้แสงแบบสุ่มตกลงเร็วมาก เหมือนกับทิศทางของรังสีตกแยกออกจากรังสีสะท้อน นี้เป็นสาเหตุให้แสงแบบสุ่มบนผิวหน้าเรียบมีความหนาแน่น เป็นการคาดหวัง เขียนว่า Phong Model ไม่เป็นพื้นฐานบนฟิสิกส์ แต่อยู่บนหลักการสังเกตุ ผลที่เห็นใกล้เคียงกับสิ่งที่ปรากฏในธรรมชาติ แต่มีความต้องการน้อยกว่าการคำนวณที่ได้รับ ค่าคงที่การดูดกลืนแบบสุ่ม Ks คือ ฟังก์ชันของผิวหน้าของวัตถุ และควรจะถูกแตกให้เป็นค่าคงที่ 3 อัน สำหรับการคำนวณแต่ละอันของ RGB ในโมเดลของการแรเงาที่ซับซ้อนมากๆ ค่าคงที่การดูดกลืนเหล่านี้ จะแปรผันกับมุมของรังสีตกกระทบ

โมเดลสำหรับการสะท้อนแบบสุ่ม เป็นพื้นฐานบนทฤษฎีและไม่มีข้อมูลที่เป็นกฎเกณฑ์ตายตัว คือ โมเดล " torrance- sparrow " โมเดลนี้เป็นพื้นฐานบนแนวคิดที่ว่าพื้นผิวหน้าถูกทำด้วย ไมโครเฟเซท (microfacets) ไมโครเฟเซทเหล่านี้ เล็ก ราบเรียบ เป็นตัวสะท้อนที่สมบูรณ์ ในผิวหน้าที่ขรุขระ ไมโครเฟเซทเหล่านี้จะถูกกำหนดทิศทางอย่างสุ่ม ซึ่งที่แสงตกกระทบจะกระทบรอบๆบนผิวหน้าขณะหนึ่งๆ ก่อนจะสะท้อนกลับ นี้จะเป็นผลให้แสงสะท้อนรับเอาสีบนผิวหน้าของวัตถุจำนวนมาก ผิวหน้าที่มีความเรียบมาก ๆ จะมีไมโครเฟเซทอยู่ในแนวเส้นตรง ซึ่งที่แสงตกกระทบจะสะท้อนออกจากผิวหน้าเพียงชั่วขณะหนึ่ง และสีของมันจะไม่ถูกกระทบโดยสีของผิวหน้า โดยสัญชาติญาณ คุณสามารถเข้าใจว่าทำไม การคำนวณกับปรากฏการณ์เหล่านี้จึงใช้เวลามาก ดังนั้นเทคนิคของพฟอง จึงถูกสร้างขึ้น

<u>การถ่ายทอดแบบเป็นระเบียบ(Specular Transmission)</u>

กลไกลการเดินทางของแสงครั้งสุดท้าย ทดสอบได้โดยการถ่ายทอดแบบเป็นระเบียบโดยดูจากแสงที่ตกกระทบผิวหน้าของวัตถุและเดินทางผ่านเข้าไปในวัตถุ เมื่อเหตุการณ์นี้เกิดขึ้นจะพบว่าวัตถุไม่สามารถสกัดกั้นแสงได้ มันจะยอมให้แสงเดินทางผ่านไปโดยสะดวก เมื่อเราดูวัตถุเหล่านี้จะพบว่า รังสีของแสงจะมีการเบี่ยงเบน ปรากฏการณ์นี้เรียกว่า การหักเห (Refraction) และเมื่อแสงเดินทางจากตัวกลางหนึ่งไปยังอีกตัวกลางหนึ่ง แสงจะมีการเบี่ยงเบน ทั้งนี้เนื่องจากความเร็วของแสงได้เปลี่ยนแปลงไป เมื่อผ่านตัวกลางหนึ่ง ๆ การเบี่ยงเบนนี้ขึ้นอยู่กับ ความหนาแน่นของวัตถุ แสงที่ตกกระทบดังที่ปรากฏจากการที่เมื่อเรามองปลาในน้ำสายตาของเราจะมองเห็นปลาในตำแหน่งที่ผิดจากตำแหน่งจริงที่ปลาอยู่ในน้ำ

เราสามารถอธิบายการหักเหของแสง (Refraction) โดยหลักการทางคณิตศาสตร์ โดยกำหนดให้ตัวกลางมีดัชนีหักเห หรือ IOR ค่า IOR นี้สามารถวัดได้จาก ความเร็วของแสงที่ผ่านตังกลางนั้นเทียบกับความเร็วของแสงในอากาศ ค่า IOR ของน้ำมีค่าเท่ากับ 1.33 และในแก้วค่า IOR จะอยู่ในช่วง 1.46 - 1.66 ขึ้นอยู่กับความหนาของแก้ว การเบี่ยงเบนของแสงจะเกิดขึ้น ณ ที่ผิวของตัวกลางทั้งสองชนิด ดังในรูป 2.6 (b) กฎของสเนลล์(Snell's Law) เป็นความสัมพันธ์ ของมุมตกกระทบกับมุมสะท้อน ดังนี้

$$\frac{\sin Theta_I}{\sin Theta_R} = \frac{n_2}{n_1} = n_{21}$$

ทิศทางของรังสี T กำหนดได้โดย

$$T = n_{12} * V + (n_{12} * C - \sqrt{(1 + (n_{12}^2 * (C^2 - 1)))} * N$$

โดยที่ C = V dot N และ T ไม่เป็นเวกเตอร์หนึ่งหน่วย ถ้าค่าของ (1+ n_{12}^2*(C^2-1))น้อยกว่าศูนย์ จะแสดงให้เห็นว่าเกิด TIR (การสะท้อนกลับหมด) ค่า TIR นี้ จะเกิดขึ้นเมื่อ แสงผ่านจากตัวกลางที่มีความหนาแน่นมากไปสู่ตัวกลางที่มีความหนาแน่นน้อย ถ้ามุมที่ลำแสงตกกระทบวัตถุ ไม่สามารถผ่านเข้าไปในวัตถุ และสะท้อนกลับไปยังทิศทางเดิม ค่าของ T จะไม่สามารถคำนวณได้

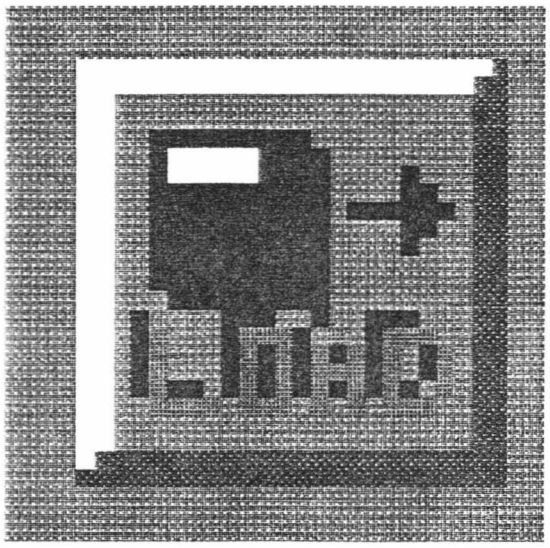
ในเทอมของสมการแรเงาโดยรวม การกระจายของแสงในการถ่ายทอดแบบสุ่ม ซึ่งมีรูปแบบคล้ายกับการสะท้อนแสงแบบสุ่มแสดงได้โดย

$$I_{st} = I_1 * k_{st} * (T \; dot \; V)^{n'}$$

ทุก ๆ เทอมในสมการนี้เป็นค่าต่อเนื่อง(analogous) สมการนี้แสดงให้เห็นว่าแสงสามารถสะท้อนกลับได้ การรวมกันของแสงเกิดจากกลไกลทางกลศาสตร์ที่ว่า เราสามารถสร้าง shading equation สำหรับใช้ในโปรแกรม ray-tracing สมการ shading equation สำหรับแสงที่เกิดจากแหล่งกำเนิดเดียว จะเป็นไปตามกฎของการ สะท้อนแบบแพร่กระจาย (diffuse reflection) การสะท้อนแบบสุ่ม (specular reflection) และ การผ่านแบบสุ่ม(specular transmission) เป็นไปตามรูปแบบ

$$I = \frac{I_1}{d + d_0} * (k_d * (L \; dot \; N) + k_s * (R \; dot \; V)^n + k_{st} * (T \; dot \; V)^{n'}) + k_a * I_a$$

ที่ซึ่งเทอมทั้งหมดจะต้องมีค่าก่อนหน้านี้ สำหรับบางการแรเงาซึ่งไม่คิดระยะทาง ซึ่งจะทำให้สมการข้างบนง่ายขึ้น คุณสามารถละบางเทอมในสมการนี้ ถ้าคุณไม่ต้องการงานเฉพาะเจาะจง หรือเฉพาะด้าน



บทที่ 3 - การวิจัยและดำเนินการ

บทที่ 3

การออกแบบ

ในขั้นแรกของการออกแบบ ได้ทำการออกแบบข้อกำหนดขึ้นมาเป็นโครงร่าง คร่าว ๆ ขึ้นมาก่อนได้ดังนี้

- เป็นโปรแกรมในลักษณะแบบGUI (Graphical User Interface)
- ใช้คุณสมบัติของความเป็นกราฟฟิคทั้งหมด
- สามารถทำการแก้ไขแปลน(แบบ)ที่เขียนในลักษณะของ 3 Plane (X-Y, X-Z, Y-Z) โดยจะเน้นที่ Plane X-Y เป็นหลัก
- สร้างภาพที่ Ray Trace แล้วขนาด 320x200 จุด 256 สี
- สามารถเก็บและอ่านภาพที่ติดตามรังสีแล้วได้ ด้วยรูปแบบ PCX
- วัตถุที่มีใช้ในโปรแกรมจะมีการแยกไว้เป็นหมวดหมู่
- วัตถุซึ่งใช้ในการสร้างภาพกำหนดให้ใช้ระนาบ"Plane" เพียงอย่างเดียวจากทั้งหมดที่มี ระนาบ(Plane), ทรงกลม(Sphere), ทรงควอดริค(Quadric), ทรงกระบอก(Cylinder) เพื่อให้โปรแกรมสามารถทำงานได้รวดเร็วขึ้น และง่ายต่อการออกแบบวัตถุ(เฟอร์นิเจอร์) ในโปรแกรม
- ผู้ใช้สามารถกำหนดขนาดของห้องเองได้

เมื่อข้อกำหนดคร่าว ๆ ของโปรแกรมเสร็จแล้ว เราจึงเริ่มทำการออกแบบหน้าที่การทำงานที่ควรจะมีของโปรแกรม เพื่อใช้กับโปรแกรม โดยเลือกแล้วนำมาใช้กับโปรแกรมได้ดังนี้


- New / สร้างแปลนว่างขึ้นมาใหม่


- Load / อ่านแปลนที่สร้างแล้วขึ้นมา


- Save / เก็บแปลนที่แก้ไขแล้วลงแผ่นดิสค์


- Point / เปลี่ยนกลับมาเป็นโหมดชี้(เป็นโหมดปริยาย)


- Property / เปลี่ยนลักษณะของวัตถุ รวมถึงการหมุน (Rotate) รูปไปทีละ 90 องศา


- Ray Trace / ทำการสร้างภาพ Ray Trace จากแปลนที่กำลังแก้ไขอยู่

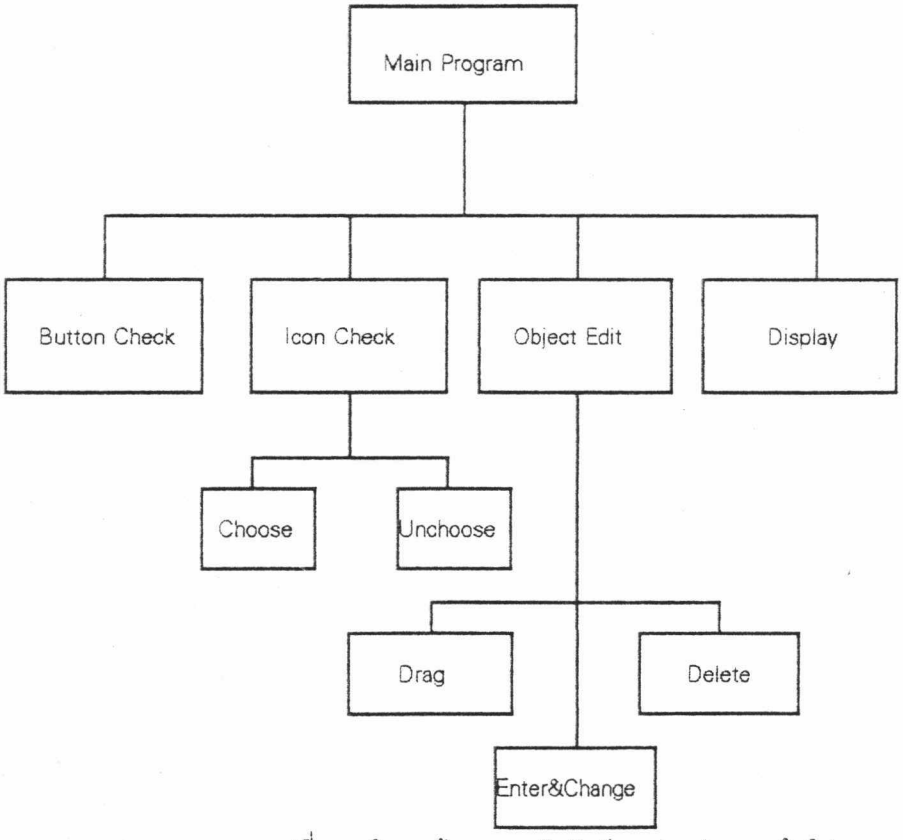
- Text / ทำการตั้งชื่อแปลน และรายละเอียดเพิ่มเติมที่เป็นข้อความ

- Eraser / ลบวัตถุที่ถูกเลือกออกไป

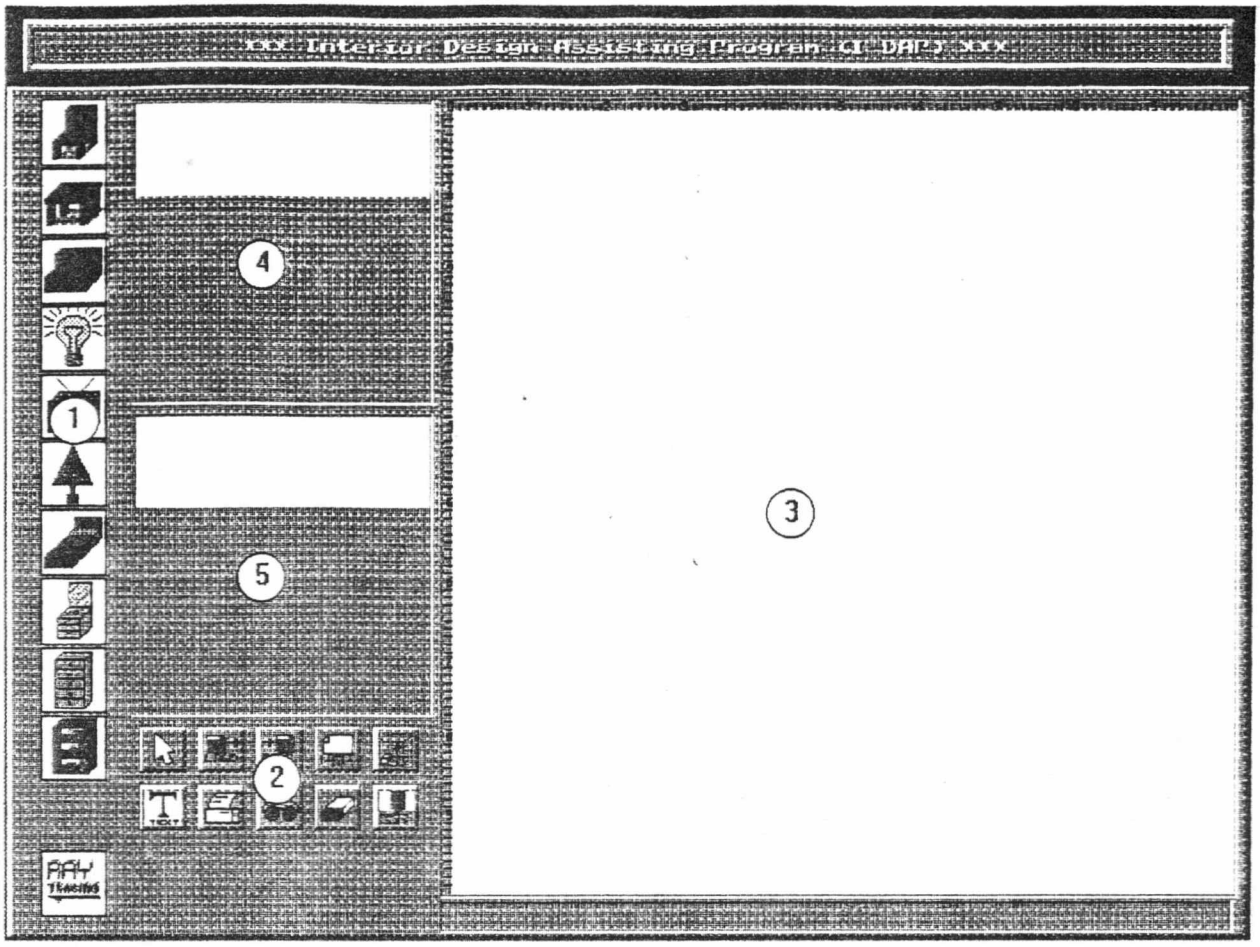
- Print / พิมพ์แปลนออกไปยังเครื่องพิมพ์

- Quit / จบการทำงานจากโปรแกรม

ด้วยหน้าที่ที่กำหนดขึ้นนี้จากการทบทวนดูหลาย ๆ รอบแล้วก็ตกลงใจว่าครบถ้วนพอสมควรแก่ลักษณะและขนาดของงานแล้ว ต่อจากนี้จึงทำการร่างการทำงานของโปรแกรมเป็นรูปเป็นร่างในกระดาษอีกครั้งหนึ่งได้รูปแบบคร่าว ๆ ดังรูปที่ 3.1 จากนั้นจึงออกแบบหน้าจอของโปรแกรม โดยในขั้นตอนนี้คณะผู้จัดทำตัดสินใจใช้การสั่งด้วยปุ่มกด เพราะว่ามีจำนวนคำสั่งไม่มาก และทำให้ง่ายต่อการเรียนรู้และใช้งาน ได้ดังนี้



รูปที่ 3.1 โครงสร้างความสัมพันธ์ของส่วนประกอบในโปรแกรม



(แสดงหน้าจอและส่วนประกอบของโปรแกรม)

มีรายละเอียดดังนี้

(1) แถวของ ICON เฟอร์นิเจอร์หรือวัตถุที่มีให้เลือก ซึ่งมีดังนี้

- เก้าอี้ (Chair)
- โต๊ะ (Table)
- โซฟา (Sofa)
- แหล่งกำเนิดแสง (Light Source)
- เครื่องใช้ไฟฟ้า (วิทยุ, โทรทัศน์, ฯลฯ) (Electric Appliances)
- ต้นไม้, พืช (Plant)
- เตียง (Bed)
- โต๊ะเครื่องแป้ง (Vanity with mirror)
- ตู้เอกสาร หรือ ตู้เสื้อผ้า (Cabinet, Drawer)
- หิ้งหนังสือ หรือ ชั้นหนังสือ (BookShelf)

(2) แถบของปุ่ม (BUTTON) คำสั่ง ซึ่งประกอบด้วยคำสั่งต่าง ๆ 10 คำสั่งดังได้กล่าวไว้แล้วข้างต้น

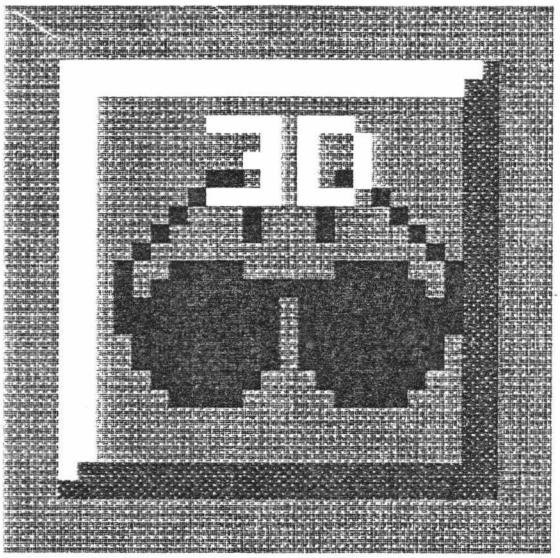
(3) แปลนแกน X-Y ใช้เพื่อแก้ไขหรือเคลื่อนย้ายวัตถุในแนวแกน X-Y เป็นโปรเจกชั่น(Projection)ของวัตถุที่เลือกในระนาบ X-Y

(4) แปลนแกน X-Z ใช้เพื่อแก้ไขหรือเคลื่อนย้ายวัตถุในแนวแกน X-Z เป็นโปรเจกชั่น(Projection)ของวัตถุที่เลือกในระนาบ X-Z

(5) แปลนแกน Y-Z ใช้เพื่อแก้ไขหรือเคลื่อนย้ายวัตถุในแนวแกน Y-Z เป็นโปรเจกชั่น(Projection)ของวัตถุที่เลือกในระนาบ Y-Z

<u>การพัฒนาและปรับปรุงโปรแกรม</u>

หลังจากได้ทำการออกแบบส่วนประกอบต่าง ๆ ทั้ง หน้าจอ หน้าที่การทำงาน ข้อกำหนดและจำกัดของโปรแกรมแล้ว จึงเริ่มทำการพัฒนาโปรแกรมให้เป็นไปตามข้อกำหนดและการออกแบบ แล้วทำการทดสอบเพื่อหาข้อผิดพลาด แล้วจึงทำการแก้ไขปรับปรุงให้ถูกต้องต่อไป

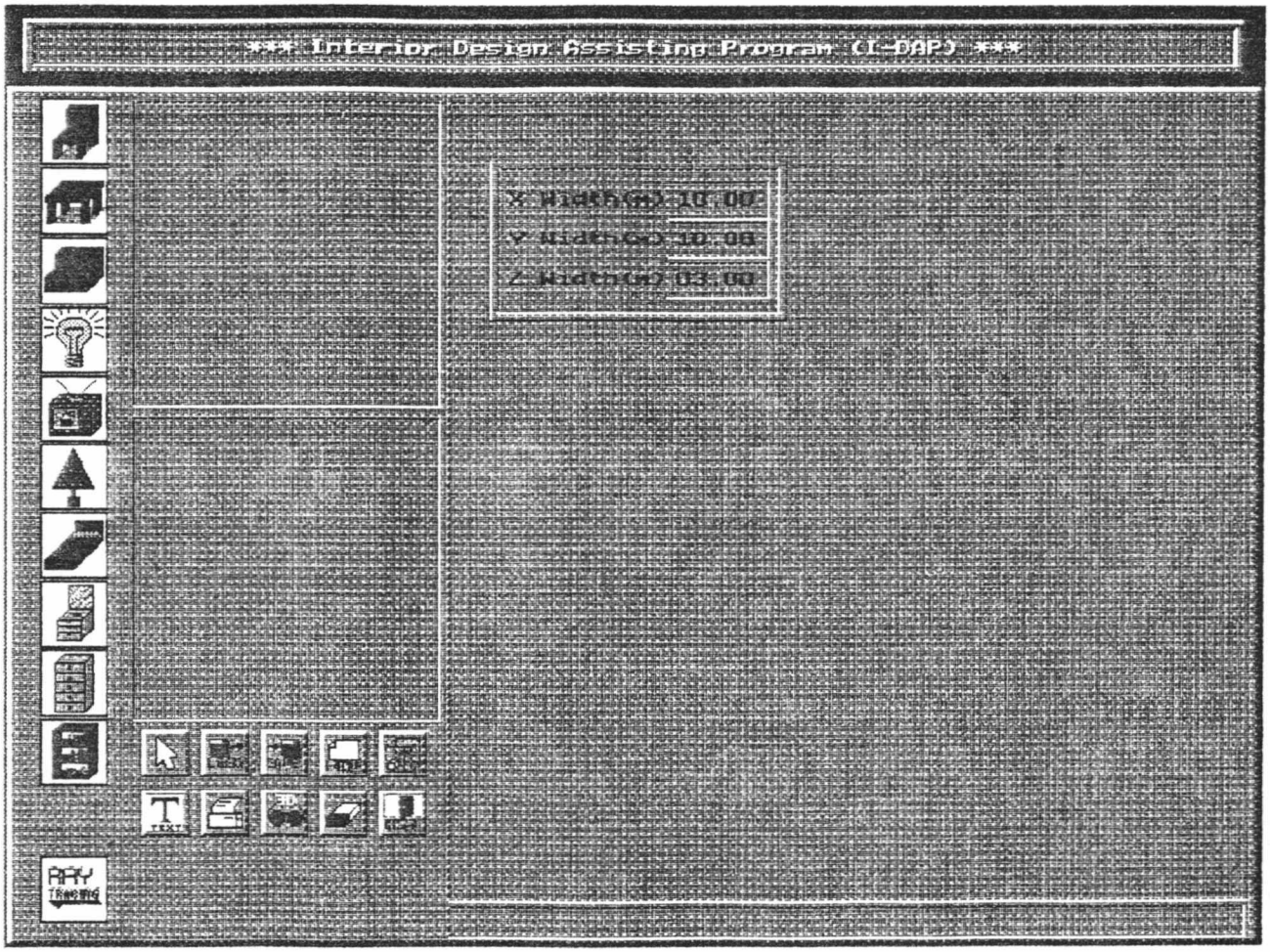


บทที่ 4 - ผลการวิจัยและวิจารณ์

บทที่ 4
ผลการวิจัยและวิจารณ์

ผลการวิจัย

สามารถทำให้โปรแกรมมีความสามารถตามที่กำหนดไว้ได้สำเร็จ และให้ผลการทำงานตามที่ได้คาดหวังไว้ โดยเป็นโปรแกรมที่ใช้งานง่าย ไม่ซับซ้อน มีการติดต่อผู้ใช้ (User Interface) ที่ดีและเข้าใจได้ง่ายพอสมควร โดยมีการใช้งานดังต่อไปนี้

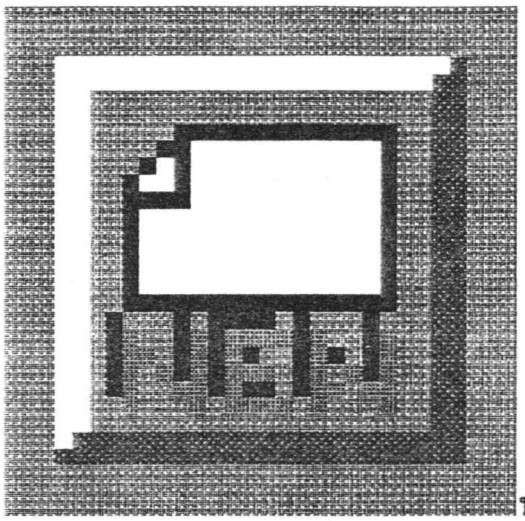


(หน้าจอแสดงหน้าจอขณะเริ่มโปรแกรม)

จากรูปแสดงหน้าจอโปรแกรมขณะเริ่มโปรแกรมโดยเรียก "IDAP.EXE" จากไดเรคทอรีของโปรแกรม เมื่อเข้าสู่โปรแกรมก็จะแสดงข้อความต้อนรับแล้วเข้าสู่โปรแกรม จะเริ่มโดยการถามขนาดมิติของห้อง โดยถามในแนวแกน

X, แกน Y, แกน Z เมื่อป้อนเสร็จแล้ว โปรแกรมจะสร้างแถบสีขาวแทนขนาดของห้องขึ้นในแปลนทั้ง 3 เพื่อแสดงอาณาเขตที่สามารถแก้ไขได้ จากนั้นก็สามารถเลือกเฟอร์นิเจอร์ที่ต้องการจากแถบตัวเลือกแนวตั้งทางด้านข้าง เมื่อเลือกแล้วจึงเลื่อนตัวชี้มายังบริเวณแปลนที่ต้องการจะวางเฟอร์นิเจอร์ แล้วจัดวางตามต้องการ เมื่อทำการจัดวางตามต้องการเสร็จแล้วจึงเลือกปุ่มคำสั่งจากแถบปุ่มคำสั่งทั้งหมดที่แถบล่าง 10 ปุ่ม โดยมีคำสั่งดังนี้

- ปุ่มเลือก
- ปุ่มอ่านไฟล์ ใช้เพื่ออ่านไฟล์ที่เก็บแปลนที่เคยสร้างเอาไว้
- ปุ่มเก็บไฟล์ ใช้เพื่อเก็บแปลนที่สร้างไว้ลงไฟล์บนดิสค์
- ปุ่มสร้างแปลนใหม่ ใช้เพื่อยกเลิกแปลนเดิมแล้วถามขนาดเพื่อกำหนดแปลนใหม่
- ปุ่มเปลี่ยนแปลงคุณสมบัติ ใช้เพื่อหมุนวัตถุ, เปลี่ยนแปลงค่ากำหนดของพื้นผิวของวัตถุ เป็นต้น
- ปุ่มตัวอักษรใช้เพื่อเพิ่มตัวอักษรบรรยายในภาพ
- ปุ่มเครื่องพิมพ์ใช้เพื่อสั่งพิมพ์แปลนที่กำหนดออกทางเครื่องพิมพ์
- ปุ่มวาดภาพ 3 มิติ ใช้เพื่อขอดูแปลนที่สร้างในมุมมองแบบ 3 มิติ
- ปุ่มยางลบ ใช้เพื่อลบวัตถุที่เลือกไว้และไม่ต้องการ
- ปุ่มออกจากโปรแกรม ใช้เพื่อยกเลิกการทำงานของโปรแกรมแล้วกลับสู่ระบบปฏิบัติการ



บทที่ 5-สรุปผลการศึกษาและข้อเสนอแนะ

บทที่ 5
สรุปผลการศึกษาพัฒนาและข้อเสนอแนะ

สรุปผลการศึกษาพัฒนา

ในปัญหาพิเศษฉบับนี้ จุดมุ่งหมายหลักก็คือการใช้เทคนิคสร้างภาพ 3 มิติแบบการติดตามรังสี เพื่อช่วยสร้างภาพ 3 มิติเหมือนจริง เพื่อใช้ในงานสถาปัตยกรรมภายใน โดยการติดตามรังสีที่ใช้นี้เป็นแบบการติดตามรังสีแบบย้อนหลัง(Backward Ray Tracing) เพื่อให้เหมาะสมกับขนาดและประสิทธิภาพของเครื่องไมโครคอมพิวเตอร์ ซึ่งไม่มีความสามารถเพียงพอในการทำงานอัลกอริทึม (Algorithm) การติดตามรังสีแบบไปข้างหน้า(Forward Ray Tracing) ได้ และในการสร้างโมเดล (Model) ก็ได้ใช้วัตถุ(Object)แบบเดียวคือ ระนาบ(Plane) เพื่อให้สามารถเพิ่มความเร็วในการสร้างภาพ(Trace)และง่ายต่อการสร้างโครงสร้างข้อมูล(Data Structure)ในโปรแกรมโดยจะแทนวัตถุต่าง ๆ โดยนำระนาบ(Plane) มาประกอบกันขึ้น ซึ่งจากผลที่ได้ก็สามารถเพิ่มความเร็วในการคำนวณได้ประมาณ 10-20% เพราะลดการคำนวณสมการของวงกลมและรูปทรงต่าง ๆ ซึ่งต้องใช้การคำนวณที่ซับซ้อนให้เหลือเพียงการคำนวณแบบง่าย ๆ ของระนาบทำให้ลดเวลาการทำงานสร้างภาพลงได้ นอกจากนี้โปรแกรมยังสามารถเก็บและอ่านภาพที่สร้างไว้แล้วได้ในแบบ PCX เพื่อความสะดวกของผู้ใช้ที่อาจไม่ต้องการรอการสร้างภาพที่ยาวนาน และต้องการใช้ภาพเดิมอีกครั้ง สรุปได้ว่าการพัฒนาสามารถสร้างโปรแกรมได้ตามจุดประสงค์เริ่มแรกอย่างถูกต้อง

ข้อเสนอแนะ

จากโปรแกรมนี้ ยังมีความบกพร่องอยู่ในหลาย ๆ จุดและควรปรับปรุงดังต่อไปนี้

- จำนวน, ชนิด และ แบบของเฟอร์นิเจอร์ที่ยังมีจำกัด ซึ่งหากจะใช้งานจริงควรจะมีมากกว่านี้และควรจะสามารถออกแบบเพิ่มเติมได้ในภายหลัง

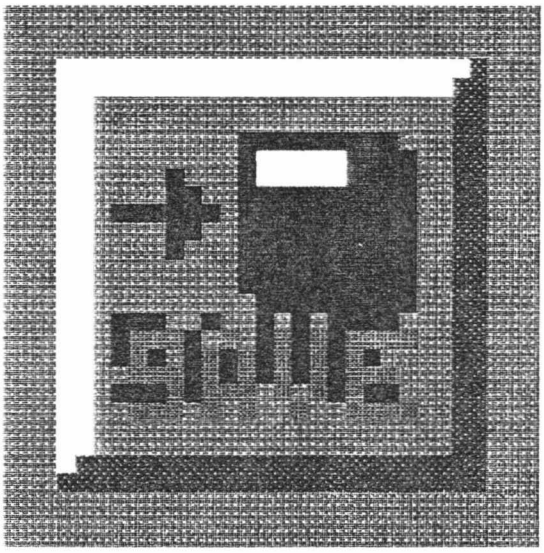
- การสร้างภาพ 3 มิติ ซึ่งยังด่อนข้างช้าอยู่ ถึงแม้ว่าภาพที่ได้จะมีคุณภาพที่ค่อนข้างดี แต่ในปัจจุบันได้มีการคิดค้นเทคนิคเพื่อเพิ่มความเร็วในการสร้างภาพของเทคนิคการติดตามรังสีอยู่มากมาย เช่น Light Buffer, Z-Buffer, Trace pipeline เป็นต้น หากจะทำการพัฒนาต่อไปจึงควรใช้เทคนิคเหล่านี้เข้ามาช่วยด้วย

- ความสามารถในการเปลี่ยนคุณสมบัติของเฟอร์นิเจอร์ยังมีเพียงการหมุนทีละ 90 องศาเท่านั้น ควรจะเพิ่มการยืดหรือหดขนาดหรือเปลี่ยนสัดส่วนของเฟอร์นิเจอร์ได้เข้าไปเพื่อให้สามารถนำวัตถุเดิมมาเปลี่ยนเพื่อให้เกิดผลลัพธ์ใหม่ที่เหมาะสมกว่าเดิมได้

- แหล่งกำเนิดแสงยังมีเพียงชนิดเดียวคือหลอดไฟ และกำหนดตำแหน่งที่เป็นไปได้เพียงไม่กี่จุด ซึ่งอยู่แต่ภายในห้อง จึงควรปรับปรุงให้มีการเพิ่มหน้าต่าง และสามารถกำหนดให้มีแสงจากภายนอกเข้ามาเพื่อเพิ่มมุมมอง

- ควรจะมีการกำหนดตำแหน่งของผู้สังเกตนอกบริเวณห้องได้

- ควรเพิ่มความสามารถในการอ่านและเขียนฟอร์แมต (Format) ของโปรแกรมสำเร็จรูปอื่น ๆ ที่เป็นที่นิยมเพื่อขยายขีดความสามารถในการทำงาน เช่น โปรแกรมออโตแคด (AutoCAD) เป็นต้น



ภาคผนวก

```
/*************************************************************/
/***                    "attrib.c"                         ***/
/***          Object Attribute Functions                   ***/
/*************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "struct.h"
#include "trace.h"


/* A function to print the surface attributes of an object */
void PrintSurfaceAttrib (OBJECT *This)  {

  printf("Surface Attributes:\n");
  printf(" Object's Color: Red=%g, Green=%g, Blue=%g\n",
           This->Properties.Color.Red,
           This->Properties.Color.Green,
           This->Properties.Color.Blue);
  printf(" Ambient=%g, Diffuse=%g, Brilliance=%g, Specular=%g, Roughness=%g\n",
           This->Properties.Ambient,
           This->Properties.Diffuse,
           This->Properties.Brilliance,
           This->Properties.Specular,
           This->Properties.Roughness);
  printf(" Reflectivity=%g\n",This->Properties.Reflection);
}

void SetSurfaceAttrib (OBJECT *This, double Red, double Green, double Blue,
                       double Ambient, double Diffuse, double Brilliance,
                       double Specular, double Roughness, double Reflection)  {

  This->Properties.Color.Red    = Red;
  This->Properties.Color.Green  = Green;
  This->Properties.Color.Blue   = Blue;
  This->Properties.Ambient      = Ambient;
  This->Properties.Diffuse      = Diffuse;
  This->Properties.Brilliance   = Brilliance;
  This->Properties.Specular     = Specular;
  This->Properties.Roughness    = Roughness;
  This->Properties.Reflection   = Reflection;
}
```

```
****** AVI.H ******
typedef struct {
     double x_width;
     double y_width;
     double z_width;
} Room;
// type define for object
typedef struct A_Plane{
     struct A_Plane *next;
     VECTOR  f_point;
     VECTOR  s_point;
     VECTOR  l_point;
     SURFACE s_attrib;
} PlaneObj;
typedef struct {
     char Obj_id[3];
     double width;
     double length;
     double height;
     int rotate;
     int Obj_num;
     struct Plane *Next;
} Obj_Header;
```

```
/*************************************************************/
/***                      "dumpio.c"                       ***/
/***         Dump format Input/Output functions            ***/
/*************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <ctype.h>
#include "misc.h"
#include "vga.h"

/* Data file header definition for the dump data file */
struct Hdr {
  unsigned Width;
  unsigned Height;
};

/*
Create a separate set of variables for dump file input and
dump file output on the chance that input and output occurs
simultaneously. Six local buffers for the storage of the pixel
color data will be established. Three for input and three for
output. Buffers are each 1024 bytes in size to accommodate the
largest raster lines supported by this code.
*/

/* Input Variables */
static FILE *InputDumpImageFile;
static struct Hdr InHdr;
static unsigned InRowNumber;
static unsigned InColumnNumber;
static BYTE RedIn   [MAXCOLS];
static BYTE GreenIn [MAXCOLS];
static BYTE BlueIn  [MAXCOLS];

/* Output Variables */
static FILE *OutputDumpImageFile;
static struct Hdr OutHdr;
static unsigned OutRowNumber;
static unsigned OutColumnNumber;
static BYTE RedOut  [MAXCOLS];
static BYTE GreenOut[MAXCOLS];
static BYTE BlueOut [MAXCOLS];

/*
Open the dump file in preparation for sequential reading or writing
the pixel values from or to disk. The Mode determines the mode in
which the file is opened.
*/
int OpenDumpFile (char *FileName, char Mode,
                  unsigned *IWidth, unsigned *IHeight)  {

  Mode = toupper(Mode);                 /* make mode upper case for compare */

  if (Mode == 'W')  {                   /* if file opened for writing */
    /* Attempt to open the output file for writing */
    if (!(OutputDumpImageFile = fopen(FileName,"wb")))
      return(EOpeningFile);

    /* Fill in and write the dump image header */
    OutHdr.Width  = *IWidth;
    OutHdr.Height = *IHeight;

    if (fwrite(&OutHdr,sizeof(struct Hdr),1,OutputDumpImageFile) != 1)
      return(EWrtFileHdr );

    OutRowNumber = 0;                   /* start write with row number zero */
    OutColumnNumber = 0;                /* start write at 1st buff position */

  }
  else if (Mode == 'R')  {              /* if file opened for reading */
    /* Attempt to open the output file for reading */
    if (!(InputDumpImageFile = fopen(FileName,"rb")))
      return(EOpeningFile);

    /* Attempt to read the dump file header */
    if (fread(&InHdr,sizeof(struct Hdr),1,InputDumpImageFile) != 1)
      return(EReadFileHdr );

    /* Retrieve and return the image parameters from the header */
    *IWidth  = InHdr.Width;
    *IHeight = InHdr.Height;

    InColumnNumber = InHdr.Width;       /* prime the pump for disk read */
  }
  else                                  /* if bad mode specified */
    return(EBadParms);

  return(NoError);
}

/*
```

```
This function closes the specified dump files after access.
*/
int CloseDumpFile(char Mode)  {

  int ErrorCode;

  Mode = toupper(Mode);                 /* make mode upper case for compare */

  switch(Mode)  {
    case 'R':                           /* close request on input file */
      if ((ErrorCode = fclose(InputDumpImageFile)) != 0)
        return(ErrorCode);
      break;

    case 'W':                           /* close request on output file */
      if ((ErrorCode = fclose(OutputDumpImageFile)) != 0)
        return(ErrorCode);
      break;

    case 'A':                           /* close request on all files */
      ErrorCode  = fclose(InputDumpImageFile);
      ErrorCode |= fclose(OutputDumpImageFile);
      if (ErrorCode != NoError)
        return(ErrorCode);
      break;
    default:
      return(EBadParms);
  }
  return(NoError);
}

/* Read the next sequential pixel from the input dump file. */

int ReadPixel (ColorRegister *Color)  {

  /* Check to see if a disk write is required */
  if (InColumnNumber == InHdr.Width)  {

    /* Yes a disk read is required. Reset ColumnNumber to zero */
    InColumnNumber = 0;

    /* First read the row number from the file and discard */
    if(fread(&InRowNumber,sizeof(unsigned),1,InputDumpImageFile) != 1)  {
      fclose(InputDumpImageFile);
      return(EReadScanLine);
    }
    /*
    Now read the Red then the Green then the Blue data
    from the file.
    */
    if (fread(RedIn,InHdr.Width,1,InputDumpImageFile) != 1)  {
      fclose(InputDumpImageFile);
      return(EReadScanLine);
    }

    if (fread(GreenIn,InHdr.Width,1,InputDumpImageFile) != 1)  {
      fclose(InputDumpImageFile);
      return(EReadScanLine);
    }

    if (fread(BlueIn,InHdr.Width,1,InputDumpImageFile) != 1)  {
      fclose(InputDumpImageFile);
      return(EReadScanLine);
    }
  }
  /* Retrieve and return the pixel's color data from the buffers */
  Color->Red   = RedIn  [InColumnNumber];
  Color->Green = GreenIn[InColumnNumber];
  Color->Blue  = BlueIn [InColumnNumber++];
  return(NoError);
}

/* Write the next sequential pixel value to the output dump file. */

int WritePixel (ColorRegister *Color)  {

  /*
  Break up the pixel into its color components and store in
  appropriate buffer awaiting write to disk.
  */
  RedOut  [OutColumnNumber]   = Color->Red;
  GreenOut[OutColumnNumber]   = Color->Green;
  BlueOut [OutColumnNumber++] = Color->Blue;

  /* Check to see if a disk write is required */
  if (OutColumnNumber == OutHdr.Width)  {

    /* Yes a disk write is required. Reset OutColumnNumber to zero */
    OutColumnNumber = 0;

    /* First write the row number to the file */
    if(fwrite(&OutRowNumber,sizeof(unsigned),1,OutputDumpImageFile) != 1)  {
```

```
      fclose(OutputDumpImageFile);
      return(EWrtScanLine);
    }
    OutRowNumber++;
    /*
    Now write the Red then the Green then the Blue data
    to the file.
    */
    if (fwrite(RedOut,OutHdr.Width,1,OutputDumpImageFile) != 1)  {
      fclose(OutputDumpImageFile);
      return(EWrtScanLine);
    }

    if (fwrite(GreenOut,OutHdr.Width,1,OutputDumpImageFile) != 1)  {
      fclose(OutputDumpImageFile);
      return(EWrtScanLine);
    }

    if (fwrite(BlueOut,OutHdr.Width,1,OutputDumpImageFile) != 1)  {
      fclose(OutputDumpImageFile);
      return(EWrtScanLine);
    }
  }
  return(NoError);
}

/*
A small driver program to test the dumpio.c code. It copies one dump
format file to another. It uses the full error checking provided by
the functions. To use this code the filenames specified in the
"OpenDumpFile" calls should be changed to names of your files. To
convert this code into a function code library, a header file called
"dumpio.h" should be created which has function prototypes for:
"OpenDumpFile", "ReadPixel", "WritePixel" and "CloseDumpFile".
*/
void main(void)  {

  unsigned ImageWidth, ImageHeight;
  unsigned long TotalPixels;
  ColorRegister PixColor;

  /* Open the input and the output dump files */
  if (OpenDumpFile("c:\\dkbtrace\\code\\colors.dis",'r',&ImageWidth,&ImageHeight) != NoError)
    exit(-1);
  if(OpenDumpFile("temp.dis",'w',&ImageWidth,&ImageHeight) != NoError)
    exit(-1);

  /* Calculate total pixels in image to copy */
  TotalPixels = (unsigned long) ImageWidth * ImageHeight;

  while(TotalPixels--)  {
    if (ReadPixel(&PixColor)  != NoError)
      exit(-1);

    if (WritePixel(&PixColor) != NoError)
      exit(-1);
  }
  /* Close both the input and the output file */
  if (CloseDumpFile('a') != NoError)
    exit(-1);
}
```

```
****** GUI.H ******

#include <dos.h>
#include <string.h>
#include <graphics.h>

#define TRUE 1
#define FALSE 0
#define IN 0
#define OUT 1
#define THICK 0
#define THIN 1
#define TEXT 0
#define IMAGE 1

#define ALTA 30
#define ALTB 48
#define ALTC 46
#define ALTD 32
#define ALTE 18
#define ALTF 33
#define ALTG 34
#define ALTH 35
#define ALTI 23
#define ALTJ 36
#define ALTK 37
#define ALTL 38
#define ALTM 50
#define ALTN 49
#define ALTO 24
#define ALTP 25
#define ALTQ 16
#define ALTR 19
#define ALTS 31
#define ALTT 20
#define ALTU 22
#define ALTV 47
#define ALTW 17
#define ALTX 45
#define ALTY 21
#define ALTZ 44

#define ALT1 120
#define ALT2 121
#define ALT3 122
#define ALT4 123
#define ALT5 124
#define ALT6 125
#define ALT7 126
#define ALT8 127
#define ALT9 128
#define ALT0 129

#define ALTMINUS  130
#define ALTPLUS   131

//LIB
void dlay(int ticks);
long getticks();
int altkey();
int ctrlkey();
int lshiftkey();
int rshiftkey();
void flushkeys();
unsigned char getvidmode();

//MOUSECLASS
class Mcursor {
      int shown;
      int current;
      int xpos;
      int ypos;
      void mouse_interrupt();
      int m1;
      int m2;
      int m3;
      int m4;
      int button;
      int count;
      long tickcount;
      int disabled;
    public:
      int  init();
      void changeto(int);
      void get_status();
      void show();
      void hide();
      void set_hor_bounds(int,int);
      void set_ver_bounds(int,int);
      void Mcursor::conditional_off(int x1,int y1,int x2,int y2);
      void Mcursor::position(int,int);
      int  LBP();
```

```
        int  RBP();
        int  getcurrent();
        int  mousex();
        int  mousey();
        int  mx();
        int  my();
        void unarm();
        void arm();
        int  LBDCLK();
        };

#define UPARROW 1
#define DOT 2
#define PENCIL 3
#define CROSSHAIR 4
#define ARROW 5
#define FINGER 6
#define POINT 7
#define CLOCK 8
#define DISK 9
#define IBAR 10
#define PAINTCAN 11
#define HAND 12
#define ERASOR 13
#define GUNSIGHT 14
#define SCISSORS 15
#define JAWS 16

//INPUT
void beep();
char *strdel(char *,int);
char *strins(char *,int,char);

class Gstring {
    protected:
            int x;
            int y;
            int xpos;
            int length;
            int ucase;
            char laststring[81];
            int shown;
            int escape;
            int retrn;
            int tab;
            int uparrow;
            int dnarrow;
            int infgd;
            int inbgd;
            int firstchar;
            int curpos;
            int curson;
            void showcurs();
            void hidecurs();
    public:
            Gstring();
           ~Gstring();
            void init(int,int,int,int);
            void show();
            void input();
            void get_input();
            void get_form_input();
            void get_form_mouse_input();
            char *getstring();
            void reset();
            void preset(char *);
            int isshown();
            void check_for_blink();
            int returnhit();
            int escapehit();
            int uparrowhit();
            int dnarrowhit();
            int tabhit();
            void setincolors(int,int);
            int hit();
            };

//GROBJECT
class Point
{
    protected:
            int x,y,color;
            viewporttype vref;
    public:
            Point();
            void move(int ptx,int pty);
            virtual void draw();
            void create(int ptx,int pty,int c);
            void restoreviewport();
            void Setcolor(int c);
            virtual void setloc(int ptx,int pty);
            virtual void erase();
```

```
            int Getcolor();
            int Getx();
            int Gety();
            };

//*****************************************************************************

class Colorbutton:public Point
{
    protected:
            int color;
            int width;
            int height;
    public:
            void init(int,int,int,int,int);
            void show(int);
            int  clicked();
            int      hit();
            int getcolor();
            };

//*****************************************************************************

class Closebutton:public Colorbutton
{
    public:
            void  show();
            };

//*****************************************************************************

class Icon:public Point
{
    protected:
            int state;
    public:
            void far *picture;
            Icon();
            ~Icon();
            void init(int,int,char*);
            void show();
            void choose();
            int hit();
            int clicked();
            int ispressed();
            };

//*****************************************************************************

class Gcheckbox:public Point
{
    protected:
            int checked;
            char *desc;
            int length;
    public:
            Gcheckbox();
            ~Gcheckbox();
            void init(int,int,char *);
            void show();
            void check();
            void uncheck();
            int is_checked();
        int hit();
            };

//*****************************************************************************

class Gradio:public Gcheckbox
{
    public:
            void show();
            void check();
            void uncheck();
            };

//*****************************************************************************

class Acticon:public Icon
{
    protected:
            void *picture[32];
            int state;
            int numpix;
    public:
            Acticon();
            ~Acticon();
            void init(int,int,char*);
            void show(int);
            void choose();
            int ispressed();
            void animate(int);
```

```
            void backforth(int);
            };

//*************************************************************************

class Button:public Point
{
    protected:
            int state,sizex,sizey;
            char btntxt[40];
            int  file_text;
            void far *picture;
            void getpic(char*);
    public:
            Button();
            ~Button();
            virtual void show();
            virtual void press();
            void init(int ptx,int pty,char* text,int);
            int hit();
        int pressed();
            };

//*************************************************************************

class Bitmap:public Point
{
    protected:
            int sizex,sizey;
            int size;
            void far *picture;
            int shown;
            int clickcount;
        long last_tick;
    public:
            Bitmap();
            ~Bitmap();
            void init(int,int);
            void load(char *);
            void save(char *);
            void show_XOR();
            void show_COPY();
            void show_AND();
            void show_OR();
            void show_NOT();
            int is_shown();
            void hide();
            void moveto(int,int);
            int capture(int,int,int,int);
            int hit();
            int clicked();
            int LBDCLK();
            int LBSCLK();
            int xsize();
            int ysize();
            int bitmapx();
            int bitmapy();
            void changexy(int,int);
            };

//*************************************************************************

class Panel:public Point
{
    protected:
            int w;
            int h;
            int in_or_out;
            int thick_or_thin;
    public:
            Panel();
            ~Panel();
            virtual void show();
            void init(int,int,int,int,int,int);
            };

//*************************************************************************

class Bevel:public Point
{
    private:
            int w;
            int h;
            int thick_or_thin;
            Panel outerbevel;
            Panel innerbevel;
    public:
            void init(int,int,int,int,int);
            virtual void show();
            };

//*************************************************************************
```

```
class OKbox:public Point
{
    protected:
            Button OKbutton;
            Button ESCbutton;
            Panel panel;
            char text[90];
            void *screen;
            int OKorNOT;
    public:
            OKbox();
            ~OKbox();
            void init(int,int,char*);
            int show();
            };

//*************************************************************************

class Gwindow:public Point
{
    protected:
            Closebutton closebox;
            int                     w,h;
            int                     fgd,bgd;
            int                     tfgd,tbgd;
            int                     active;
            char            title[48];
            void            *beneath;
    public:
            Gwindow();
            ~Gwindow();
            void init(int,int,int,int,int,int,int,int,char *);
            void show();
            void redraw();
            void hide();
            int     closeboxhit();
            int  sizecornerhit();
            int  titlebarhit();
            void resize();
            void move();
            int write_to_disk();
            int read_from_disk();
            int totalGwindows();
            };

//GMENU
typedef char gitemarray[80][10];


class Gmenu {
    protected:
            int on;
            int x,y,w,h;
            int num;
            gitemarray gitems;
            int menuchoice;
            int oldbarx,oldbary;
            void *ptr;
            void *menubar;
    public:
            Gmenu();
            ~Gmenu();
            void init(int xloc,int yloc,int numentries,gitemarray gitem);
            int show();
            void hide();
            int isshown();
            };

//*************************************************************************

class Gmenubutton {
    protected:
            int on;
            int x,y;
            int offfgd,offbgd;
            int onfgd,onbgd;
            char id[20];
    public:
            Gmenubutton();
            ~Gmenubutton();
            void init(int xloc,int yloc,int ffgd,int fbgd,
            int nfgd,int nbgd,char txt[20]);
            void show();
            void press();
            int hit();
            };

//GPRINT
void gprintf(int xloc,int yloc,char *fmt,...);
void gprintc(int xloc, int yloc, char *fmt,...);
void gprintxy(int xloc,int yloc,char *fmt,...);
```

```
//SOUNDQ
#define TimerTick 0x8
#define noisemax 8192
#define ON 1
#define OFF 0

#define C 523
#define CS 554
#define D 587
#define DS 622
#define E 659
#define F 698
#define FS 740
#define G 784
#define GS 831
#define A 880
#define AS 932
#define B 988
#define C1 1046

#define SN 32
#define EN 63
#define QN 125
#define HN 250
#define WN 500

#define ENT 20
#define QNT 41
#define HNT 83

#define BN 30000,5
#define SR 30000,32
#define ER 30000,63
#define QR 30000,125
#define HR 30000,250
#define WR 30000,500
#define MM 30000,1

class SoundQ{
    protected:
            float speed_factor;
    public:
            SoundQ();
            ~SoundQ();
            void play(int,int);
            void adjust_speed(float);
            };

typedef struct
{
  int duration;
  int freq;
} noise;

void empty_sound_queue();
void init_sound(void);
void restore_sound(void);
int submit_sound(int freq,int delay);
void interrupt soundsystem(...);

//SCREEN
extern "C" void     _Cdecl vga256_driver();

class Screen {
    protected:
            static int huge alwayszero();
    public:
            Screen();
            ~Screen();
            int VGA_480_16();
            int VGA_350_16();
            int VGA_200_16();
            int VGA_200_256();
            void fill(int);
                };
```

```
****** LIGHT.C ******
/*************************************************************/
/***                      "light.c"                        ***/
/***          Light Rav / Light Source Code                ***/
/*************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "mymem.h"
#include "struct.h"
#include "trace.h"
#include "vectors.h"

/*
The following two functions are used in testing for shadow. They
calculate information about lightray/object intersections.
*/
double MakeLightRay (OBJECT *LightPtr, VECTOR Point, RAY *Ray)  {

  double DistanceT;

  /*
  Create a ray from the point of intersection to this light source.
  Ray origin is the point. Ray direction is direct ray from center
  of light source to point. Since both rays are positional, the new
  ray is just the difference between the other rays.
  */
  Ray->Origin = Point;

  VSub(LightPtr->ObjSpecific.L.Center,Point,Ray->Direction);
  /*
  Find and return the length or distance of this ray. Make unit vector of
  direction.
  */
  VLength(Ray->Direction,DistanceT);
  VNormalize(Ray->Direction,Ray->Direction);
  return(DistanceT);
}


COLOR GetLightColor (OBJECT *LightPtr, OBJECT *ObjectPtr, RAY *Ray,
                     double Distance)  {

  OBJECT *ShadowObjPtr;
  COLOR   Color;
  double  DistanceT;

  /* Assume light source is obscured by some object */
  Color.Red = Color.Green = Color.Blue = 0.0;

  /*
  Traverse the object list looking for obscuring object. Do not test
  to see if object itself is obscuring the light source.
  */
  ShadowObjPtr = ObjectList;
  while (ShadowObjPtr)  {
    if (ShadowObjPtr != ObjectPtr)  {
      DistanceT = ShadowObjPtr->Intersect(ShadowObjPtr,Ray);
      /*
      If an intersection is found and it is between the light and
      the object and not past it then return black for color.
      */
      if ((DistanceT > EPSILON) && (DistanceT < Distance))
        return(Color);
    }
    ShadowObjPtr = ShadowObjPtr->Next;
  }
  /*
  If we got here, no objects obscure the light from the specified
  light source to the intersection point. Therefore, return the
  colors specified for the light source.
  */
  Color.Red   = LightPtr->Properties.Color.Red;
  Color.Green = LightPtr->Properties.Color.Green;
  Color.Blue  = LightPtr->Properties.Color.Blue;
  return(Color);
}

/* Declare a light source */
OBJECT *MakeLight (double X, double Y, double Z,
                   double Red, double Green, double Blue)   {

  OBJECT *ObjectPtr = NULL;

  ObjectPtr = malloc(sizeof(OBJECT));
  if (ObjectPtr != NULL)  {
    memset(ObjectPtr,'\0',sizeof(OBJECT));
    ObjectPtr->Type = Light;
    ObjectPtr->ObjSpecific.L.Center.X = X;
    ObjectPtr->ObjSpecific.L.Center.Y = Y;
    ObjectPtr->ObjSpecific.L.Center.Z = Z;
```

```
    ObjectPtr->Properties.Color.Red   = Red;
    ObjectPtr->Properties.Color.Green = Green;
    ObjectPtr->Properties.Color.Blue  = Blue;
    AddToList(&LightList,ObjectPtr);
  }
  return(ObjectPtr);
}
```

```
/*************************************************************/
/***                    "linklist.c"                       ***/
/***                Linked List Functions                  ***/
/*************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "mymem.h"
#include "struct.h"
#include "trace.h"

/*
Given a ptr to the ptr to the list and a ptr to an object,
add the object at the head of the list.
*/
void AddToList (OBJECT **ListPtr, OBJECT *Item)   {

  if (*ListPtr == NULL)   {
    *ListPtr = Item;
    Item->Next = NULL;
  }
  else   {
    Item->Next = *ListPtr;
    *ListPtr = Item;
  }
}

/*
Given a ptr to ptr to the list, traverse the list deleting each
object on the list. Return the number of objects deleted.
*/
unsigned FreeList (OBJECT **ListPtr)   {

  OBJECT *TempPtr;
  unsigned Number = 0;

  while (*ListPtr)   {
    Number++;
    TempPtr = *ListPtr;
    *ListPtr = (*ListPtr)->Next;
    free (TempPtr);
  }
  ListPtr = NULL;
  return (Number);
}

/*
Given a ptr to a list, traverse the list and print out the
information about each object on the list. The function called to
print an object is contained in the object itself. Return the number
of objects in the list.
*/
unsigned PrintList (OBJECT *ListPtr)   {

  unsigned Number = 0;

  while (ListPtr)   {
    Number++;
    ListPtr->Print(ListPtr);
    ListPtr = ListPtr->Next;
  }
  return (Number);
}
```

****** MISC.H ******

```
/*************************************************************/
/***                      "misc.h"                         ***/
/***          Miscellaneous item include file              ***/
/*************************************************************/

/* Define new types */
#ifndef __BYTE
#define __BYTE
typedef unsigned char BYTE;
#endif

#ifndef _CompletionCode_
#define _CompletionCode_
typedef int CompletionCode;
#endif


#define TRUE        1
#define FALSE       0

#define VIDEO    0x10

/* Common Macros */
#define MIN(a,b) ((a)>(b)) ? (b):(a)
#define MAX(a,b) ((a)>(b)) ? (a):(b)
#define SQUARE(x) (x)*(x)

#define MAXSCREENWIDTH              1024
#define MAXBYTESPERSCAN    MAXSCREENWIDTH
#define MAXSCREENHEIGHT              768
#define MAXPALETTECOLORS              16
#define MAX256PALETTECOLORS          256

/* Error Bit Definitions */
#define NoError           0
#define EBadParms        -1
#define EFileNotFound    -2
#define EOpeningFile     -3
#define EReadFileHdr     -4
#define ENotPCXFile      -5
#define ECorrupt         -6
#define EWrtFileHdr      -7
#define EWrtOutFile      -8
#define EReadScanLine    -9
#define EWrtScanLine    -10
#define EPCCFile        -11
#define EGraphics       -12
#define ENoMemory       -13
#define EWrtExtPal      -14
#define ENotGIFFile     -15
#define EReadRowNum     -16
#define EReadData       -17
```

```
/*************************************************************/
/***                      "mquan.c"                        ***/
/***         Median Cut Color Quantizer Program            ***/
/***             produces 256 color images                 ***/
/***     from "dump" format color image data files         ***/
/*************************************************************/

#include <stdio.h>
#include <conio.h>
#include <dos.h>
#include <process.h>
#include <string.h>
#include <alloc.h>
#include "misc.h"
#include "vga.h"
#include "gif.h"
#include "pcx.h"

/*
Memory allocation bits used to keep track of how much far heap memory
has been allocated by this program.
*/

#define  RGBMEM         1
unsigned MemoryAlloc = 0;              /* variable used to keep track */

#define SQUARE(x)    (x)*(x)
#define COLLEVELS      32              /* number of brightness levels */
#define NUMAXIS         3              /* num of axes in RGB cube */

/* Local global variables */
/*
Data file structure information
*/
static struct {
   unsigned Width;
   unsigned Height;
} Hdr;

static unsigned Ver = 1;
static unsigned Rev = 0 ;
static unsigned ImageWidth;
static unsigned ImageHeight;
static unsigned long ImagePixels;
static int MaxValue;

/* Single row buffers for reading raw data */
static BYTE RedRowBuf[MAXCOLS];
static BYTE GreenRowBuf[MAXCOLS];
static BYTE BlueRowBuf[MAXCOLS];

/*
RGBCube: Three dimensional array implemented in such a way as to work well
on a machine that has trouble with objects greater than 64K.  The indices
into this array are the color components, rgb, normalized to fit in
the range defined by COLLEVELS.  The values in the array are frequency counts
of the particular color.  The array is set up here to be an array of pointers
to smaller 2 dimensional arrays, so no object is greater than 64K.
*/

static unsigned long far *RGBCube[COLLEVELS];
static unsigned NumBoxes;
static unsigned Verbose;
static unsigned CorrectGamma;
static unsigned GenComFile;
static unsigned GenGIFFile;
static unsigned GenPCXFile;
static unsigned InstallColors;

/* Gamma correction table - GAMMA = 2.222 */
static BYTE GammaTable[] =   {
   0, 9,13,16,18,20,21,23,24,26,27,28,29,30,32,33,
  34,34,35,36,37,38,39,40,40,41,42,43,43,44,45,45,
  46,47,47,48,48,49,50,50,51,51,52,53,53,54,54,55,
  55,56,56,57,57,58,58,59,59,60,60,61,61,62,62,63
};

/*
Boxes: Structure holding the unsorted generated color boxes.  Includes the
low and high value along each axis of RGBCube, and the number of elements
in the box.
*/

struct Box  {
   unsigned Lo[3];
   unsigned Hi[3];
   unsigned long NumElements;
}  Boxes[MAXNUMCOLREGS];

/*
Sorted version of Boxes.
*/
```

```
struct Box SBoxes[MAXNUMCOLREGS];

/*
ColRegs: Holds the determined values of the color registers.
*/
ColorRegister ColRegs[MAXNUMCOLREGS];

/*
SColRegs: Sorted version of ColRegs.
*/

ColorRegister SColRegs[MAXNUMCOLREGS];


/* beginning of program functions */

/*
This function will output the message string passed to it if the Verbose
option variable is set true.
*/

void Message(char *String)  {
   if (Verbose)
      printf("%s",String);
}

/*
This function deallocates all far heap memory that has been allocated
*/

void DeAllocMemory( void )  {

   register unsigned Index;

   /* test MemoryAlloc bit to see what was allocated then dispose */

   Message("Deallocating program memory\n");

   if (MemoryAlloc & RGBMEM)
      for (Index=0; Index < COLLEVELS; Index++)
         if (RGBCube[Index] != NULL)
           farfree((char far *) RGBCube[Index]);

}

/*
Allocate all memory for the entire program. The MemoryAlloc variable is
used to keep track of all memory that has been allocated from the far
heap. That way DeAllocMemory can give it all back to the system when the
program terminates.
*/

unsigned AllocMemory( void )  {

   register unsigned Index;

   Message("Allocating program memory\n");

   /* create COLLEVELS number of pointer to 2 D arrays */

   /*
   Set the initial values of the RGBCube sub-array pointers to NULL,
   so we can do proper checks later to see if they have been allocated
   or not.
   */

   for (Index=0; Index < COLLEVELS; Index++)
      RGBCube[Index] = NULL;

   for (Index=0; Index < COLLEVELS; Index++)  {
      RGBCube[Index] = (unsigned long far *) farcalloc(COLLEVELS*COLLEVELS,
       (unsigned long) sizeof(unsigned long));

      if (RGBCube[Index] == NULL)  {
         printf("RGBCube memory allocation failed\n");
         printf("\t only %ld bytes of memory available\n",farcoreleft());
         DeAllocMemory();
         return(FALSE);
      }
      /* clear it to all zeros */
      _fmemset((char far *) RGBCube[Index],'\0',
                COLLEVELS*COLLEVELS*sizeof(unsigned long));
   }
   MemoryAlloc |= RGBMEM;              /* indicate success */

   return(TRUE);
}


/*
This function sets the indices to the numbers of the other axis after
```

```
a main axis has been selected.
*/

void OtherAxes(unsigned MainAxis, unsigned *Other1, unsigned *Other2)  {
   switch (MainAxis)  {
   case 0:
     *Other1 = 1;
     *Other2 = 2;
     break;
   case 1:
     *Other1 = 0;
     *Other2 = 2;
     break;
   case 2:
     *Other1 = 0;
     *Other2 = 1;
   }
}


/*
This function takes a index value into the Boxes array, and shrinks the
specified box to tightly fit around the input color frequency data (eg.
there are no zero planes on the sides of the box).
*/

void Shrink(unsigned BoxIndex)  {

   unsigned axis,aax1,aax2;
   register unsigned ind[3], flag;

   /* Along each axis: */

   for (axis=0; axis < NUMAXIS; axis++)  {
      OtherAxes(axis,&aax1,&aax2);

     /* Scan off zero planes on from the low end of the axis */
     flag = 0;
     for (ind[axis]=Boxes[BoxIndex].Lo[axis];
          ind[axis] <= Boxes[BoxIndex].Hi[axis]; ind[axis]++)  {
        for (ind[aax1]=Boxes[BoxIndex].Lo[aax1];
             ind[aax1] <= Boxes[BoxIndex].Hi[aax1]; ind[aax1]++)  {
           for (ind[aax2]=Boxes[BoxIndex].Lo[aax2];
                ind[aax2] <= Boxes[BoxIndex].Hi[aax2]; ind[aax2]++)
             if (RGBCube[ind[0]][ind[1]*COLLEVELS+ind[2]])  {
               flag=1;
               break;
             }
           if (flag) break;
        }
       if (flag) break;
     }
     Boxes[BoxIndex].Lo[axis] = ind[axis];

     /* Scan off zero planes from the high end of the axis */
     flag = 0;
     for (ind[axis]=Boxes[BoxIndex].Hi[axis];
          ind[axis]+1 >= Boxes[BoxIndex].Lo[axis]+1; ind[axis]--)  {
        for (ind[aax1]=Boxes[BoxIndex].Hi[aax1];
             ind[aax1]+1 >= Boxes[BoxIndex].Lo[aax1]+1; ind[aax1]--)  {
           for (ind[aax2]=Boxes[BoxIndex].Hi[aax2];
                ind[aax2]+1>=Boxes[BoxIndex].Lo[aax2]+1; ind[aax2]--)
             if (RGBCube[ind[0]][ind[1]*COLLEVELS+ind[2]])  {
                flag = 1;
                break;
             }
           if (flag) break;
        }
        if (flag) break;
     }
     Boxes[BoxIndex].Hi[axis] = ind[axis];
   }
}

/*
This function selects the optimum colors from the color frequency data,
using the Median Cut algorithm.  It prints the number of colors used at
its termination.
*/

void SelectColorBoxes( void )  {

   register unsigned SelectedBox, c;
   register unsigned ind[3], Max, axis, TargetBox, k;
   unsigned aax1,aax2;
   unsigned long LongMax, PlaneSum, ElementSum;

   /*
   Initialize the first and only box in the array to contain the entire RGBCube,
   then discard unused zero planes surrounding it.
   */
```

```
for (c=0; c < NUMAXIS; c++)  {
  Boxes[0].Lo[c] = 0;
  Boxes[0].Hi[c] = COLLEVELS-1;
}
Boxes[0].NumElements = ImagePixels;
NumBoxes = 1;

Shrink(0);

/* Perform the following until all color registers are used up */

while(NumBoxes < MAXNUMCOLREGS)  {
   /*
   Pick the box with the maximum number of elements that is not a single
   color value to work with.  It will be the box we will split.
   */

   LongMax = 0;
   SelectedBox = 1000;
   for (c=0; c < NumBoxes; c++)  {
     if ((Boxes[c].NumElements > LongMax) &&
        ((Boxes[c].Lo[0] != Boxes[c].Hi[0]) ||
         (Boxes[c].Lo[1] != Boxes[c].Hi[1]) ||
         (Boxes[c].Lo[2] != Boxes[c].Hi[2])))  {
       LongMax = Boxes[c].NumElements;
       SelectedBox = c;
     }
   }

   /*
   If we couldn't find any box that was not a single color, we don't
   need to assign any more colors, so we can terminate this loop.
   */

   if (SelectedBox == 1000)
     break;

   /* Choose the longest axis of the box to split it along */

   axis = 0;
   Max = Boxes[SelectedBox].Hi[axis] - Boxes[SelectedBox].Lo[axis];
   for (k=1; k < NUMAXIS; k++)  {
      if (Max < (c=(Boxes[SelectedBox].Hi[k]-Boxes[SelectedBox].Lo[k])))  {
         Max = c;
         axis = k;
      }
   }

   /*
   Check to see if any of our previously assigned boxes have zero elements
   (may happen in degenerate cases), if so, reuse them.  If not, use the
   next available box.
   */

   TargetBox = NumBoxes;
   for (c=0; c < NumBoxes; c++)  {
     if (Boxes[c].NumElements == 0)  {
         TargetBox = c;
         break;
     }
   }

   OtherAxes(axis,&aax1,&aax2);
   if (Boxes[SelectedBox].Hi[axis] != Boxes[SelectedBox].Lo[axis])  {

      /*
      Sum planes of box from low end until the sum exceeds half the total
      number of elements in the box.  That is the point where we will
      split it.
      */

      ElementSum = 0;
      for (ind[axis]=Boxes[SelectedBox].Lo[axis];
           ind[axis] <= Boxes[SelectedBox].Hi[axis]; ind[axis]++)  {
         PlaneSum = 0;
         for (ind[aax1]=Boxes[SelectedBox].Lo[aax1];
              ind[aax1] <= Boxes[SelectedBox].Hi[aax1]; ind[aax1]++)
            for (ind[aax2]=Boxes[SelectedBox].Lo[aax2];
                 ind[aax2] <= Boxes[SelectedBox].Hi[aax2]; ind[aax2]++)
              PlaneSum += RGBCube[ind[0]][ind[1]*COLLEVELS+ind[2]];
         ElementSum += PlaneSum;
         if (ElementSum > Boxes[SelectedBox].NumElements/2)
           break;
      }
      /*
      If we did not exceed half the total until we added the last plane
      (such as in a case where the last plane contains the bulk of the data
      points), back up so we do not create the new box as a degenerate box.
      */

      if (ind[axis] == Boxes[SelectedBox].Hi[axis])  {
         ind[axis]--;
```

```
            ElementSum -= PlaneSum;
         }

         /*
         The new box has most of the data the same as the old box, but its low
         extent is the index above the point where we needed to split, and its
         number of elements is the total number of elements in this whole box,
         minus the number in the planes we just summed.
         */

         for (c=0; c < NUMAXIS; c++)  {
            Boxes[TargetBox].Lo[c] = Boxes[SelectedBox].Lo[c];
            Boxes[TargetBox].Hi[c] = Boxes[SelectedBox].Hi[c];
         }
         Boxes[TargetBox].Lo[axis] = ind[axis]+1;
         Boxes[TargetBox].NumElements = Boxes[SelectedBox].NumElements -
                                        ElementSum;

         /*
         The high extent of our old box is now cut off at the plane we just
         split at and the number of elements in it is the number we just
         summed.
         */

         Boxes[SelectedBox].Hi[axis] = ind[axis];
         Boxes[SelectedBox].NumElements = ElementSum;

         /* Discard zero planes around both our new boxes */

         Shrink(SelectedBox);
         Shrink(TargetBox);

         /*
         If we used the top box in our list, we have to increment the
         total number of boxes used, to make ready for the use of the next
         free box.
         */

         if (TargetBox == NumBoxes)
            NumBoxes++;
      }
   }

   /* show number of display colors to be used if requested to */
   if (Verbose)
      printf("%d colors will be used for display of the image\n",NumBoxes);
}


/*
This function calculates the actual color register values for each box,
based on the weighted distribution of data in the box.  It then sorts the
color registers by brightness (using a calculation described by the VGA
technical reference for calculating brightness).
*/

void SortColors( void )  {

   register unsigned Index,c,flag,temp,r,b,g;
   unsigned indices[MAXNUMCOLREGS];
   unsigned long weightedcolor[MAXNUMCOLREGS],rsum,bsum,gsum,tmp;

   for (Index=0; Index < NumBoxes; Index++)  {

      /* Calculate a weighted sum of the color values in the box */

      rsum = bsum = gsum = 0;
      for (r=Boxes[Index].Lo[0]; r<=Boxes[Index].Hi[0]; r++)
         for (b=Boxes[Index].Lo[1]; b<=Boxes[Index].Hi[1]; b++)
            for (g=Boxes[Index].Lo[2]; g<=Boxes[Index].Hi[2]; g++)  {
               tmp = RGBCube[r][b*COLLEVELS+g];
               rsum += r*tmp;
               bsum += b*tmp;
               gsum += g*tmp;
            }

      /* Pick the actual color for that box based on the weighted sum */

      ColRegs[Index].Red   = rsum/Boxes[Index].NumElements;
      ColRegs[Index].Blue  = bsum/Boxes[Index].NumElements;
      ColRegs[Index].Green = gsum/Boxes[Index].NumElements;
   }
   /*
   Set up for an index sort of the brightness by first calculating the
   weighted brightness of each color (based on the calculation described
   in the VGA manual.
   */

   for (Index=0; Index < NumBoxes; Index++)  {
      indices[Index] = Index;
      weightedcolor[Index] = ColRegs[Index].Red  *30 +
                             ColRegs[Index].Blue *11 +
```

```
                                ColRegs[Index].Green*59;
    }

    /*
    Do a bubble sort of the weighted colors via indices. Sort is done in
    ascending order.
    */

    flag = 1;
    while (flag)  {
       flag = 0;
       for (Index=0; Index < NumBoxes-1; Index++)
          if (weightedcolor[indices[Index]] > weightedcolor[indices[Index+1]])  {
             temp = indices[Index];
             indices[Index] = indices[Index+1];
             indices[Index+1] = temp;
             flag = 1;
          }
    }

    /*
    Remap the boxes and the color registers into SBoxes and SColRegs via the
    sorted indices found above.
    */

    for (Index=0; Index < NumBoxes; Index++)  {
       SColRegs[Index].Red   = ColRegs[indices[Index]].Red;
       SColRegs[Index].Blue  = ColRegs[indices[Index]].Blue;
       SColRegs[Index].Green = ColRegs[indices[Index]].Green;
       SBoxes[Index].NumElements = Boxes[indices[Index]].NumElements;
       for (c=0; c < NUMAXIS; c++)  {
          SBoxes[Index].Hi[c] = Boxes[indices[Index]].Hi[c];
          SBoxes[Index].Lo[c] = Boxes[indices[Index]].Lo[c];
       }
    }
}


/*
This function maps the raw image pixel data from the raw data file into the
new color map we've come up with in the SColRegs array.
*/

int DisplayImageData(char *FileName )  {

   FILE *ImageFile;
   unsigned Row, Col;
   register unsigned c,k,goodindex, PixVal;
   unsigned long minerror,error;
   unsigned RowNum, Red, Green, Blue;
   register int ReturnCode,r,b,g,i;

   /*
   Set the RGBCube array to a value that can't be a color register index
   (MAXNUMCOLREGS*2) so we can detect when we hit on a part of the array that
   is not included in a color box.
   */

   for (c=0; c < COLLEVELS; c++)
      for (k=0; k < COLLEVELS*COLLEVELS; k++)
        RGBCube[c][k] = MAXNUMCOLREGS*2;

   /*
   Fill the boxes in the RGBCube array with the index number for that box, so
   we can tell what box a particular color index into the RGBCube array is in
   by a single access.
   */

   for (i=0; i < NumBoxes; i++)
     for (r=SBoxes[i].Lo[0]; r <= SBoxes[i].Hi[0]; r++)
       for (b=SBoxes[i].Lo[1]; b <= SBoxes[i].Hi[1]; b++)
          for (g=SBoxes[i].Lo[2]; g <= SBoxes[i].Hi[2]; g++)
            RGBCube[r][b*COLLEVELS+g] = i;

   /*
   Rescan the raw image file so that it can now be displayed.
   */

   printf("\nPass 2\n");
   if (Verbose)
      printf("Reading file: %s\n",FileName);

   /* Attempt to open the file */
   if (!(ImageFile = fopen(FileName,"rb")))   {
      printf("File %s not found in Pass 2!\n",FileName);
      ReturnCode = EFileNotFound;
      goto ErrorExit;
   }

  /* Read in the header */
  if (fread(&Hdr,sizeof(Hdr),1,ImageFile) != 1)   {
      ReturnCode = EReadFileHdr;
```

```
        goto ErrorExit;
    }

    /*
    Now attempt to read the raw image file into three
    separate buffers: Red, Green & Blue a row at a time.
    */

    for (Row=0; Row < ImageHeight; Row++)   {
       if (fread(&RowNum,sizeof(unsigned),1,ImageFile) != 1)   {
         printf("\nError reading image row number !\n");
         ReturnCode = EReadRowNum;
         goto ErrorExit;
       }

       /* Now read a line of red data */
       if (fread(RedRowBuf,ImageWidth,1,ImageFile) != 1)   {
         printf("\nError reading red data in Pass 2!\n");
         ReturnCode = EReadData;
         goto ErrorExit;
       }

       /* Now read a line of green data */
       if (fread(GreenRowBuf,ImageWidth,1,ImageFile) != 1)   {
         printf("\nError reading green data in Pass 2!\n");
         ReturnCode = EReadData;
         goto ErrorExit;
       }

       /* Now read a line of blue data */
       if (fread(BlueRowBuf,ImageWidth,1,ImageFile) != 1)   {
         printf("\nError reading blue data in Pass 2!\n");
         ReturnCode = EReadData;
         goto ErrorExit;
       }

       /*
       For each RGB triplet in this row of image data. Scale values
       accordingly.
       */
       for (Col=0; Col < ImageWidth; Col++)  {
         Red   = RedRowBuf[Col]   >> 3;
         Green = GreenRowBuf[Col] >> 3;
         Blue  = BlueRowBuf[Col]  >> 3;

         /*
         If this particular color is inside one of the boxes,
         assign the color index for this pixel to the value at that
         spot in the cube.
         */
         if (RGBCube[Red][(Blue*COLLEVELS)+Green] != MAXNUMCOLREGS*2)  {
           PixVal = RGBCube[Red][Blue*COLLEVELS+Green];
           SetPixelValue(Col,Row,PixVal);
         }
         else  {
           /*
           Otherwise, we need to scan the array of colors to find which is
           the closest to our prospective color.
           */

           goodindex = 0;
           minerror = SQUARE(Red-SColRegs[goodindex].Red)+
                      SQUARE(Blue-SColRegs[goodindex].Blue)+
                      SQUARE(Green-SColRegs[goodindex].Green);
          /*
          Scan all color registers to find which has the smallest error
          when it is used for this pixel.
          */

          for (k=1; k < NumBoxes; k++)  {
            error = SQUARE(Red-SColRegs[k].Red)+
                    SQUARE(Blue-SColRegs[k].Blue)+
                    SQUARE(Green-SColRegs[k].Green);
            if (error < minerror)  {
              minerror = error;
              goodindex = k;
            }
         }
         /* Assign that register to this pixel */
         SetPixelValue(Col,Row,goodindex);
       }
    }
    }
    /* Normal function exit point */
    printf("\n");
    fclose(ImageFile);
    return(0);

ErrorExit:
    fclose(ImageFile);
    return(ReturnCode);
```

```
}

/*
The purpose of this function is to partially make up
for the normalization of pixel values performed previously.
The values in the color registers are scaled upwards toward
the maximum value of 63 to increase image brightness. Then, if
requested, the color data is gamma corrected.
*/

void ScaleColRegisters(void)  {

   register unsigned Index;
   register unsigned Temp;

   /* Find the maximum value of any RGB component value */
   MaxValue = -1;
   for (Index = 0; Index < MAXNUMCOLREGS; Index++)  {
      if (SColRegs[Index].Red > MaxValue)
        MaxValue = SColRegs[Index].Red;
      if (SColRegs[Index].Green > MaxValue)
        MaxValue = SColRegs[Index].Green;
      if (SColRegs[Index].Blue > MaxValue)
        MaxValue = SColRegs[Index].Blue;
   }
   /* Scale all color register components accordingly */
   for (Index = 0; Index < MAXNUMCOLREGS; Index++)  {

      /* temp used to prevent overflow of BYTE value */
      Temp = SColRegs[Index].Red * (unsigned) MAXCOLREGVAL;
      Temp /= MaxValue;
      if (CorrectGamma)
        SColRegs[Index].Red = GammaTable[Temp];
      else
        SColRegs[Index].Red = Temp;

      Temp = SColRegs[Index].Green * (unsigned) MAXCOLREGVAL;
      Temp /= MaxValue;
      if (CorrectGamma)
        SColRegs[Index].Green = GammaTable[Temp];
      else
        SColRegs[Index].Green = Temp;

      Temp = SColRegs[Index].Blue * (unsigned) MAXCOLREGVAL;
      Temp /= MaxValue;
      if (CorrectGamma)
        SColRegs[Index].Blue = GammaTable[Temp];
      else
        SColRegs[Index].Blue = Temp;
   }
}

/*
This function produces an executable .COM file for display of the
digitized color image. It writes a small code segment followed by
the 256 color register value followed by the image data to the
specified file.
*/

void WriteComFile(char *FileName)  {

   FILE *OutPutFile;
   char     String[80];
   unsigned Index, PixelValue, Col, Row;
   BYTE FileCode[] =
     {0xB4,0x0F,0xCD,0x10,0xA2,0x37,0x01,0xB4,0x00,0xB0,0x13,0xCD,0x10,
      0xB8,0x12,0x10,0xBB,0x00,0x00,0xB9,0x00,0x01,0xBA,0x38,0x01,0xCD,
      0x10,0xB9,0x00,0xFA,0xBE,0x38,0x04,0xB8,0x00,0xA0,0x8E,0xC0,0xBF,
      0x00,0x00,0xF3,0xA4,0xB4,0x00,0xCD,0x16,0xB4,0x00,0xA0,0x37,0x01,
      0xCD,0x10,0xC3,0x00};


   if (!strchr(FileName,'.'))  {        /* is there an ext ? */
      strcpy(String,FileName);          /* copy filename to buffer */
      FileName = String;                /* FileName now points at buffer */
      strcat(FileName,".com");          /* if not add .com ext */
   }
    /* open the output file */

    if ((OutPutFile = fopen(FileName,"wb")) == NULL)  {
       SetTextMode();
       DeAllocMemory();
       printf("Cannot open Image .COM file\n");
       exit(1);
    }

    /* write code segment to the file */
    for (Index=0; Index < sizeof(FileCode); Index++)
       if (fputc(FileCode[Index],OutPutFile) != FileCode[Index])  {
          SetTextMode();
          DeAllocMemory();
          printf("Error writing Image .COM code seg\n");
```

```
            exit(1);
         }

      /* now write the color register rgb values to the file */
      for (Index = 0; Index < MAX256PALETTECOLORS; Index++)  {
         if (fputc(SColRegs[Index].Red,OutPutFile) != SColRegs[Index].Red)  {
            SetTextMode();
            DeAllocMemory();
            printf("Error writing Image .COM red color reg\n");
            exit(1);
         }
         if (fputc(SColRegs[Index].Green,OutPutFile) != SColRegs[Index].Green)  {
            SetTextMode();
            DeAllocMemory();
            printf("Error writing Image .COM green color reg\n");
            exit(1);
         }
         if (fputc(SColRegs[Index].Blue,OutPutFile) != SColRegs[Index].Blue)  {
            SetTextMode();
            DeAllocMemory();
            printf("Error writing Image .COM blue color reg\n");
            exit(1);
         }
      }
      /* now write the actual image data to the file */
      for (Row=0; Row < 200; Row++)
        for (Col=0; Col < 320; Col++)  {
           PixelValue = GetPixelValue(Col,Row); /* read the value from display */
           fputc(PixelValue,OutPutFile);
         }

      fclose(OutPutFile);
}

/*
Pass 1 through the data reads the raw data file, scales the raw
data and builds the RGBCube.
*/

int ProcessPass1(char *FileName)   {

   FILE *ImageFile;
   register unsigned Row, Col;
   unsigned RowNum, NumColors, c,k;
   unsigned Red, Green, Blue;
   int ReturnCode;

   printf("\nPass 1\n");
   if (Verbose)
      printf("Reading file: %s\n",FileName);

   /* attempt to allocate required memory for RGBCube */
   if (!AllocMemory())  {
      printf("\nMemory allocation error in Pass 1!\n");
      ReturnCode = ENoMemory;
      goto ErrorExit;
   }

   /* Attempt to open the file */
   if (!(ImageFile = fopen(FileName,"rb")))   {
      printf("File %s not found in Pass 1!\n",FileName);
      ReturnCode = EFileNotFound;
      goto ErrorExit;
   }

  /* Read in the header */
  if (fread(&Hdr,sizeof(Hdr),1,ImageFile) != 1)   {
      printf("Error reading image header in Pass 1!\n",FileName);
      ReturnCode = EReadFileHdr;
      goto ErrorExit;
   }
   /* Make local copies of image parameters */
   ImageWidth  = Hdr.Width;
   ImageHeight = Hdr.Height;
   ImagePixels = (unsigned long) ImageWidth * ImageHeight;
   /*
   Now scan complete image scaling the RGB values from
   the range 0..FF to 0..1F by right shifting three bits.
   Each row of raw data is comprised of a RowNum and then
   a row of RGB color values.
   */

   Message("Reading Line:   0");
   for (Row=0; Row < ImageHeight; Row++)   {

      if (fread(&RowNum,sizeof(unsigned),1,ImageFile) != 1)   {
        printf("\nError reading image row number in Pass 1!\n");
        ReturnCode = EReadRowNum;
        goto ErrorExit;
      }
      if (Verbose)
        printf("\b\b\b%3d",RowNum);     /* output row number to display */
```

```
      /* Now read a line of red data */
      if (fread(RedRowBuf,ImageWidth,1,ImageFile) != 1)   {
        printf("\nError reading red data in Pass 1!\n");
        ReturnCode = EReadData;
        goto ErrorExit;
      }

      /* Now read a line of green data */
      if (fread(GreenRowBuf,ImageWidth,1,ImageFile) != 1)   {
        printf("\nError reading green data in Pass 1!\n");
        ReturnCode = EReadData;
        goto ErrorExit;
      }

      /* Now read a line of blue data */
      if (fread(BlueRowBuf,ImageWidth,1,ImageFile) != 1)   {
        printf("\nError reading blue data in Pass 1!\n");
        ReturnCode = EReadData;
        goto ErrorExit;
      }

      /*
      For each RGB triplet in this row of image data. Scale values
      accordingly.
      */
      for (Col=0; Col < ImageWidth; Col++)  {
        Red   = RedRowBuf[Col]   >> 3;
        Green = GreenRowBuf[Col] >> 3;
        Blue  = BlueRowBuf[Col]  >>3;

       /* Add one to the count of pixels with that color */
       (RGBCube[Red][(Blue*COLLEVELS)+Green])++;
      }
    }
    /*
    Count and print the number of unique colors in the input by scanning the
    RGBCube array and looking for non-zero frequencies.
    */

    NumColors = 0;
    for (c=0; c < COLLEVELS; c++)
      for (k=0; k < COLLEVELS*COLLEVELS; k++)
        if (RGBCube[c][k])
          NumColors++;

    if (Verbose)
       printf("\n%d unique colors in image data\n",NumColors);

    if (Verbose)
      printf("Pass 1 completed.\n\n");

    /* Normal function exit point */
    fclose(ImageFile);
    return(NoError);

ErrorExit:
    printf("\n");
    DeAllocMemory();
    fclose(ImageFile);
    return(ReturnCode);

}

/*
This function provides help in the advent of operator error. Program
terminates after help is given
*/

void ShowHelp( void )  {

  printf("\nThis program quantizes, displays and saves a 256 color image.\n");
  printf("It is invoked as follows:\n\n");
  printf("Usage: mquan [-c -g -h -i -p -v -x] infilename [outfilename]<cr>\n");
  printf("  -c inhibit gamma correction\n");
  printf("  -g create a GIF output file\n");
  printf("  -h or ? show help\n");
  printf("  -i prevents the color reg values from being installed\n");
  printf("  -p create PCX output file\n");
  printf("  -v displays program progress information\n");
  printf("  -x create executable display program\n");
  printf(" outfilename is name given to generated display file(s).\n\n");
  exit(1);
}

/* main quantizer program */

void main(short argc, char *argv[])  {

    unsigned FileNameCounter, ArgIndex;
    char     *ImageFileName;
    char     *OutputFileName;
```

```
    clrscr();
    printf("MQUAN -- Median Cut Color Quantizer Program.  Ver: %d.%d\n",Ver,Rev);
    printf("from the book \"Practical Ray Tracing in C\"\n");
    printf("Written by Craig A. Lindley\n\n");
    printf("This program converts 16.7 million 24 bit color images\n");
    printf("in dump format to 256 color images for display on a VGA\n");
    printf("equipped computer\n\n");

    /* install default options */
    Verbose       = FALSE;              /* don't be wordy */
    GenComFile    = FALSE;              /* don't generate an .COM file */
    GenGIFFile    = FALSE;              /* don't generate a GIF file */
    GenPCXFile    = FALSE;              /* don't generate a PCX file */
    InstallColors = TRUE;               /* install calculated colors */
    CorrectGamma  = TRUE;               /* do gamma correction */

    /* parse all command line arguments */

    FileNameCounter = 0;                /* count of user specified filenames */
    for (ArgIndex=1; ArgIndex < argc; ArgIndex++)  {
      if (*argv[ArgIndex] != '-')  {    /* if not a cmd line switch */
                                        /* must be a filename */
        if (*argv[ArgIndex] == '?')     /* help requested ? */
          ShowHelp();
        if (FileNameCounter > 2)        /* only two filenames allowed */
          ShowHelp();                   /* if more then error exit */
        if (FileNameCounter == 0)
          ImageFileName = argv[ArgIndex];  /* save input image filename */
        else
          OutputFileName = argv[ArgIndex]; /* save output image filename */

        FileNameCounter++;              /* inc count for error check */
      }
      else  {                           /* its a cmd line switch */
        switch (*(argv[ArgIndex]+1))  {  /* parse the cmd line */
          case 'c':
          case 'C':
            CorrectGamma = FALSE;
            break;
          case 'g':
          case 'G':
            GenGIFFile = TRUE;
            break;
          case 'h':
          case 'H':
            ShowHelp();
            break;
          case 'i':
          case 'I':
            InstallColors = FALSE;
            break;
          case 'p':
          case 'P':
            GenPCXFile = TRUE;
            break;
          case 'v':
          case 'V':
            Verbose    = TRUE;
            break;
         case 'x':
         case 'X':
           GenComFile = TRUE;
           break;
         default:
           printf("Error - invalid cmd line switch encountered\n");
           ShowHelp();
        }
      }
    }
    if (FileNameCounter < 1)  {
      printf("Error - input filename required\n");
      ShowHelp();
    }
    if ((GenComFile | GenGIFFile | GenPCXFile) && (FileNameCounter != 2))  {
      printf("Error - input and output filename required\n");
      ShowHelp();
    }

    if (ProcessPass1(ImageFileName) == NoError)  {
      Message("Selecting optimum color palette\n");
      SelectColorBoxes();
      SortColors();

      /* Check for VESA compliance */
//      CheckForVESA();
      /* display the resultant color image with the proper palette */
      SelectVideoMode(ImageWidth,ImageHeight,TRUE);
      DisplayImageData(ImageFileName);
      /*
      If we want real colors InstallColors is TRUE. If we want pseudo
      colors for special effects, InstallColors is FALSE. In this case
```

```
      we must call GetAllColorRegs to store the default RGB values of the
      color registers into the SColReg array.
      */
      if (InstallColors)   {
        ScaleColRegisters();
        SetAllColorRegs(SColRegs);
      }
      else
        GetAllColorRegs(SColRegs);

      if (GenGIFFile)   {
        /*
        Create a 256 color GIF file of displayed image. Assumes
        single bit plane with one byte per pixel.
        */
        WriteGIFFile(OutputFileName,ImageWidth,ImageHeight,
                                    ImageWidth,ImageHeight,0,0);
      }
      if (GenPCXFile)   {
        /*
        Create a 256 color PCX file of displayed image. Assumes
        single bit plane with one byte per pixel.
        */
        WritePCXFile(OutputFileName, ImageWidth, ImageHeight);
      }
      if (GenComFile && (ImageWidth <= 320))  {
        /* create an executable file for display of color image */
        WriteComFile(OutputFileName);
      }
    }
    /* prepare to return to dos */
    getch();
    SetTextMode();
    DeAllocMemory();
}
```

```
/*************************************************************/
/***                      "pcx.c"                          ***/
/***                  PCX Function Library                 ***/
/*************************************************************/

#include <stdio.h>
#include <string.h>
#include <process.h>
#include <conio.h>
#include <dos.h>
#include "misc.h"
#include "vga.h"
#include "pcx.h"

/* Externally Accessible Global Variables */
struct   PCX_File PCXData;        /* PCX File Hdr Variable */
unsigned ImageWidth, ImageHeight;

/* Variables global to this file only */
static FILE    *PCXFile;       /* file handle */
static BYTE     ScanLine[MAXBYTESPERSCAN];
static struct   ExtendedPalette Color256Palette;


/* Start of Functions */
/*
This function opens an image file and attempts to read the
PCX header information.
*/
CompletionCode ReadPCXFileHdr (char *FileName, int Verbose)  {

   char     String[80];

   if (!strchr(FileName,'.'))  {        /* is there an ext ? */
       strcpy(String,FileName);         /* copy filename to buffer */
      FileName = String;                /* FileName now points at buffer */
      strcat(FileName,".pcx");          /* if not add .pcx ext */
   }
   /* try to open the PCX file */
   if ((PCXFile = fopen(FileName,"rb")) == NULL)  {

      printf("PCX file: %s not found\n",FileName);
      return(EFileNotFound);
   }
   /* try to read the file header record */
   if (fread(&PCXData,sizeof(struct PCX_File),1,PCXFile) != 1)  {
      printf("Error reading PCX file header\n");
      return(EReadFileHdr);
   }
   /* check to make sure its a PCX file */
   if (PCXData.PCXHeader.Header != PCXHdrTag)  {

      printf("Error not a PCX file\n");
      return(ENotPCXFile);
   }
   /* Yep, we've got a PCX file OK. Display info if requested */
   if (Verbose)  {

      clrscr();
      printf("PCX Image Information for file: %s\n\n",FileName);
      printf("\tVersion: %d\n", PCXData.PCXHeader.Version);
      printf("\tCompression: %s\n",
              PCXData.PCXHeader.Encode == 0 ? "None":"RLL");
      printf("\tBits Per Pixel: %d\n",PCXData.PCXHeader.BitPerPix);
      printf("\tX1: %d\n",PCXData.PCXHeader.X1);
      printf("\tY1: %d\n",PCXData.PCXHeader.Y1);
      printf("\tX2: %d\n",PCXData.PCXHeader.X2);
      printf("\tY2: %d\n",PCXData.PCXHeader.Y2);
      printf("\tHoriz Resolution: %d\n",PCXData.PCXHeader.Hres);
      printf("\tVert  Resolution: %d\n",PCXData.PCXHeader.Vres);
      printf("\tVMode: %d\n",PCXData.Info.Vmode);
      printf("\tNumber of Planes: %d\n",PCXData.Info.NumOfPlanes);
      printf("\tBytes Per Scan Line One Plane: %d\n",PCXData.Info.BytesPerLine);
      printf("\nHit <Enter> to proceed - ^C to abort\n");
      getchar();                        /* wait for operator input */
   }
   return(NoError);
}

static CompletionCode ExpandScanLine (FILE *InFile)  {

   register short CharRead;
   unsigned       InPtr,RepCount;
   unsigned       BytesToRead;

   BytesToRead = PCXData.Info.NumOfPlanes * PCXData.Info.BytesPerLine;

   InPtr = 0;                           /* initialize input ptr */
   do  {

      CharRead = getc(InFile);          /* read a byte from the file */
      if (CharRead == EOF)              /* error should never read EOF */
```

```
            return(FALSE);                  /* abort picture */

         if ((CharRead & 0xC0) == 0xC0) {   /* a repeat tag ? */

            RepCount = CharRead & ~0xC0;    /* repeat 1..63 */
            CharRead = getc(InFile);        /* read byte to repeat */
            if (CharRead == EOF)            /* error should never read EOF */
               return(FALSE);               /* abort picture */

            while (RepCount--)              /* expand byte */
               ScanLine[InPtr++] =          /* RepCount times */
                 CharRead;
         }
         else                               /* just a byte of data */
           ScanLine[InPtr++] = CharRead;    /* store in buffer */
      } while (InPtr < BytesToRead);        /* expand a full scan line */
/*
When we get here, we have an array, ScanLine, which is composed of
NumOfPlanes sections each BytesPerLine long. For a 256 color VGA images
it is 1 plane of 320, 640, 800 or 1024 bytes. For a 256 color image,
the ScanLine contains one line of raster data. Return an indication
that this operation went smoothly.
*/

   return(TRUE);
}

/*
This function reads and displays a 256 color PCX file.
*/

void DisplayPCXFile (char *FileName, int Verbose)
{
   register unsigned ScanNum;           /* scan line being expanded
                                           and displayed */
   register unsigned ColNum, Index;     /* pixel being read */
   int      PCXError;


   if ((PCXError = ReadPCXFileHdr(FileName,Verbose)) != NoError)
      exit(PCXError);

   /* Header has been read, now we are ready to display the PCX image */
   /* PCC files cannot be displayed */

   if ((PCXData.PCXHeader.X1 != 0) || (PCXData.PCXHeader.Y1 != 0))
   {
      printf("Error PCC file not PCX file\n");
      exit(EPCCFile);
   }

   /* Copy image specs to local storage */
   ImageWidth  = PCXData.PCXHeader.Hres;
   ImageHeight = PCXData.PCXHeader.Vres;

   /* Select the correct video mode if VESA device else abort */
   CheckForVESA();
   SelectVideoMode(ImageWidth,ImageHeight,TRUE);

    /* proceed to unpack and display the PCX file */
   for (ScanNum=0; ScanNum < ImageHeight; ScanNum++)  {
     if (ExpandScanLine(PCXFile) != TRUE)  {
       printf("Scanline corrupt in PCX file\n");
       exit(ECorrupt);
     }
     for (ColNum=0; ColNum < ImageWidth; ColNum++)
       SetPixelValue(ColNum,ScanNum,ScanLine[ColNum]);
   }
   /* Now read the extended palette structure from the end of the file */

   if (fread(&Color256Palette,sizeof(struct ExtendedPalette),1,PCXFile) == 1)
      /* Extended palette read ok. Now check tag. */
     if (Color256Palette.ExtendedPalette == PCX256ColorTag)  {
       for(Index=0; Index < MAX256PALETTECOLORS; Index++)  {
         Color256Palette.Palette[Index].Red   >>= 2;
         Color256Palette.Palette[Index].Green >>= 2;
         Color256Palette.Palette[Index].Blue  >>= 2;
       }
       SetAllColorRegs((ColorRegister *) &(Color256Palette.Palette));
     }
   fclose(PCXFile);
}


/*
The following functions create a PCX file from a raster image
on the display and writes it to disk.
*/

CompletionCode WritePCXHdr(char *FileName, unsigned BitsPerPixel,
   unsigned MaxX,  unsigned MaxY, unsigned Planes,
   unsigned BytesPerLine)  {
```

```
    unsigned    Index;
    char        String[80];

    if (!strchr(FileName,'.'))  {       /* is there an ext ? */
                                        /* if not ... */
       strcpy(String,FileName);         /* copy filename to buffer */
       FileName = String;               /* FileName now points at buffer */
       strcat(FileName,".pcx");         /* add .pcx ext */
    }

    if ((PCXFile = fopen(FileName,"w+b")) == NULL)
       return (EFileNotFound);

    /* initialize the PCX file header info */
    PCXData.PCXHeader.Header    = PCXHdrTag;
    PCXData.PCXHeader.Version   = 5;
    PCXData.PCXHeader.Encode    = 1;
    PCXData.PCXHeader.BitPerPix = BitsPerPixel;
    PCXData.PCXHeader.X1        = 0;
    PCXData.PCXHeader.Y1        = 0;
    PCXData.PCXHeader.X2        = MaxX-1;
    PCXData.PCXHeader.Y2        = MaxY-1;
    PCXData.PCXHeader.Hres      = MaxX;
    PCXData.PCXHeader.Vres      = MaxY;
    PCXData.Info.Vmode          = 0;
    PCXData.Info.NumOfPlanes    = Planes;
    PCXData.Info.BytesPerLine   = BytesPerLine;

    /*
    Initialize the 16 color palette structure in the PCX file data.
    This 16 color palette will be written to the PCX file even though
    the images are 256 color images containing an extended palette.
    The extended palette will be written at the end of the PCX raster data.
    */

    for (Index = 0; Index < MAXPALETTECOLORS; Index++)  {
       PCXData.Palette[Index].Red   = 0;
       PCXData.Palette[Index].Green = 0;
       PCXData.Palette[Index].Blue  = 0;
    }

    /* clear the unused area at the end of the PCX header */
    memset(&PCXData.Info.unused,'\0',sizeof(PCXData.Info.unused));

    /* now write the file header to the physical file */
    if (fwrite(&PCXData,sizeof(struct PCX_File),1,PCXFile) != 1)
       return(EWrtFileHdr);

    return(NoError);
}


static CompletionCode CompressScanLine(FILE *OutFile)  {

    register unsigned OutPtr,RepCount,RepChar;
    register unsigned BytesToWrite;

    BytesToWrite = PCXData.Info.NumOfPlanes * PCXData.Info.BytesPerLine;

    OutPtr = 0;                         /* ptr to data to compress */
    do  {

       RepChar = ScanLine[OutPtr++];    /* get byte to start compression */
       RepCount = 1;                    /* byte seen once at this point */
       while ((ScanLine[OutPtr]==RepChar) &&
              (RepCount < MaxRepCount)    &&
              (OutPtr < BytesToWrite))  {
          RepCount++;                   /* count all repetitions of char */
          OutPtr++;                     /* bump ptr and check again */
       }

       /*
       Repeat sequence found or if chars has either or both MSBs set
       than must process as a repetition count and char sequence.
       */

       if ((RepCount > 1) || (RepChar > 0xBF))  {
          RepCount |= 0xC0;             /* set two MSBs */
          if (putc(RepCount,OutFile) == EOF) /* write count to file */
             return(FALSE);             /* if error return error */
       }
       if (putc(RepChar,OutFile) == EOF)/* write char to file */
          return(FALSE);                /* if error return error */
    } while (OutPtr < BytesToWrite);    /* until all bytes in scan
                                           are compressed */
    return(TRUE);                       /* indicate operation successful */
}

/*
This function writes a PCX file to disk from the image currently
being displayed on the monitor. The image data is read directly
off of the screen and the palette information is read from the
```

```
DAC.
*/

void WritePCXFile (char *FileName, unsigned MaxX, unsigned MaxY)  {

   register unsigned ScanLineNum, PixelNum, Index;
   int PCXError;

   /* write out PCX header and palette */
   if ((PCXError = WritePCXHdr(FileName, 8, MaxX, MaxY, 1, MaxX)) != NoError)
      exit(PCXError);

   ImageWidth = MaxX;
   ImageHeight = MaxY;

/*
At this point we will read the displayed image from the screen a scanline
at time. For 256 color images there is only a single bit plane
so the data read from the screen is placed directly into the ScanLine array
for compressing.
*/

   for (ScanLineNum=0; ScanLineNum < ImageHeight; ScanLineNum++)  {
     for (PixelNum=0; PixelNum < ImageWidth; PixelNum++)
       ScanLine[PixelNum] = GetPixelValue(PixelNum,ScanLineNum);

     if (CompressScanLine(PCXFile) != TRUE) /* compress a complete scan */
       exit(EWrtScanLine);
   }
   /*
   Write an extended palette record to the file after the
   raster data. Read the 256 color register RGB values and
   store them in the Color256Palette structure before writing
   them to the PCX file. This structure is tagged to assure validity.
   */

   Color256Palette.ExtendedPalette = PCX256ColorTag;
   GetAllColorRegs((ColorRegister *) &(Color256Palette.Palette));

   for (Index = 0; Index < MAX256PALETTECOLORS; Index++)  {
     Color256Palette.Palette[Index].Red   <<= 2;
     Color256Palette.Palette[Index].Green <<= 2;
     Color256Palette.Palette[Index].Blue  <<= 2;
   }
   /*
   With all of the color register values read, write the
   extended palette structure to the PCX file.
   */

   if (fwrite(&Color256Palette,
              sizeof(struct ExtendedPalette),1,PCXFile) != 1)  {
     fclose(PCXFile);                    /* close the completed PCX file */
     exit(EWrtExtPal);
   }
   /* file has been written prepare to close up shop */

   fclose(PCXFile);                    /* close the completed PCX file */
}
```

```
/*************************************************************/
/***                      "pcx.h"                          ***/
/***                  PCX include file                     ***/
/***              for PCX access functions                 ***/
/*************************************************************/

#define MAXSCREENWIDTH              1024
#define MAXBYTESPERSCAN    MAXSCREENWIDTH
#define MAXSCREENHEIGHT              768
#define MAXPALETTECOLORS              16
#define MAX256PALETTECOLORS          256

/* PCX File Structures and Defines */

#define PCXHdrTag        10    /* tag in valid PCX file */
#define MaxRepCount      63    /* max # of repeat bytes */
#define PCX256ColorTag   12    /* tag for extended palette in PCX file */

struct PCXFileHeader  {

   BYTE     Header;            /* marks file as PCX file */
   BYTE     Version;           /* 0 = version 2.5 */
                               /* 2 = 2.8 with palette info */
                               /* 3 = no palette info 2.8 or 3.0 */
                               /* 5 = 3.0 with palette info */
   BYTE     Encode;            /* File encoding mode */
   BYTE     BitPerPix;         /* Bits per pixel */
   unsigned X1;                /* Picture dimensions inclusive */
   unsigned Y1;
   unsigned X2;
   unsigned Y2;
   unsigned Hres;              /* Graphics adapter Horiz    resolution */
   unsigned Vres;              /* Graphics adapter Vertical resolution */
};

struct PCXInfo  {

   BYTE     Vmode;             /* Ignore should always be zero */
   BYTE     NumOfPlanes;       /* Number of bit planes */
   unsigned BytesPerLine;      /* Bytes Per Line in picture */
   BYTE     unused[60];        /* fills out header to 128 bytes */
};


struct PCX_File  {

   struct PCXFileHeader PCXHeader;
   ColorRegister        Palette[MAXPALETTECOLORS]; /* Max size of 48 bytes */
   struct PCXInfo       Info;
};


/* Extended palette data structure for 256 color PCX files */

struct ExtendedPalette  {

   BYTE ExtendedPalette;
   ColorRegister Palette[MAX256PALETTECOLORS]; /* Max size of 768 bytes */
};


/* PCX C Function Prototype Declarations */
void DisplayPCXFile (char *FileName, int Verbose);

void WritePCXFile (char *FileName, unsigned MaxX, unsigned MaxY);
```

```
/*************************************************************/
/***                      "plane.c"                        ***/
/***              Plane Object Functions                   ***/
/*************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include "mymem.h"
#include <math.h>
#include "struct.h"
#include "trace.h"
#include "vectors.h"

/*
The following plane intersection function is based upon the
substitution of the 3D parametric equation of a line into the formula
for a plane. See text for details.
*/
double PlaneIntersect (OBJECT *This, RAY *Ray)  {

  double Vd, Vo, T;

  /*
  Calculate the dot product of the planes normal and the
  rays direction.
  */
  VDot(This->ObjSpecific.P.Normal,Ray->Direction,Vd);

  if (Vd <= EPSILON)          /* ray is parallel to plane. No intersection */
    return(0.0);              /* return t=0 */

  VDot(This->ObjSpecific.P.Normal,Ray->Origin,Vo);
  Vo += This->ObjSpecific.P.Distance;
  Vo *= -1.0;

  T = Vo/Vd;
  if (T < 0.0)                /* intersection behind ray origin */
    return(0.0);              /* return t=0 */
  return(T);                  /* else return t */
}

/*
The following function returns a unit normal to a plane. The normal to
the plane is that given in its definition.
*/

VECTOR PlaneNormal (OBJECT *This, VECTOR Point)  {

  VECTOR Normal;

  Normal.X = This->ObjSpecific.P.Normal.X;
  Normal.Y = This->ObjSpecific.P.Normal.Y;
  Normal.Z = This->ObjSpecific.P.Normal.Z;
  return(Normal);
}

/* A function to print the contents of a plane object */
void PlanePrint (OBJECT *This)  {

  printf("\n\nPlane's normal is: <A=%g,B=%g,C=%g> with distance D=%g\n",
         This->ObjSpecific.P.Normal.X,
         This->ObjSpecific.P.Normal.Y,
         This->ObjSpecific.P.Normal.Z,
        -1.0*This->ObjSpecific.P.Distance);
  PrintSurfaceAttrib(This);
}


/* Declare an object of type plane */
OBJECT *MakePlane (double A, double B, double C, double D)   {

  OBJECT *ObjectPtr = NULL;

  ObjectPtr = malloc(sizeof(OBJECT));
  if (ObjectPtr != NULL)  {
    memset(ObjectPtr,'\0',sizeof(OBJECT));
    ObjectPtr->Type = Plane;
    /* Assign functions for intersection, normal and printing */
    ObjectPtr->Intersect = PlaneIntersect;
    ObjectPtr->Normal    = PlaneNormal;
    ObjectPtr->Print     = PlanePrint;
    ObjectPtr->ObjSpecific.P.Normal.X = A;
    ObjectPtr->ObjSpecific.P.Normal.Y = B;
    ObjectPtr->ObjSpecific.P.Normal.Z = C;
    ObjectPtr->ObjSpecific.P.Distance = -D;
    /* Make the normal in case it wasn't entered that way */
    VNormalize(ObjectPtr->ObjSpecific.P.Normal,ObjectPtr->ObjSpecific.P.Normal);
    AddToList(&ObjectList,ObjectPtr);
  }
  return(ObjectPtr);
}
```

```
/*************************************************************/
/***                        "rays.c"                       ***/
/***        Ray Creation and Shading Functions             ***/
/*************************************************************/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <conio.h>
#include <time.h>
#include "struct.h"
#include "trace.h"
#include "vectors.h"

/*
Global variables used throughout this file and program
*/
DISPLAY Display;
VIEWPOINT View;
COLOR BackGround;
static unsigned RecurseLevel;
static unsigned Ver = 1;
static unsigned Rev = 0;
double VTemp;                   /* temp variable used by macros */


/* Definition of the two major lists we will use */
OBJECT *ObjectList;             /* list of objects */
OBJECT *LightList;              /* list of light sources */


/*
Shade the point at which the ray strikes the object.
*/
void Shade(OBJECT *ObjectPtr, RAY *Ray, VECTOR Normal,
           VECTOR Point, COLOR *Color)  {

  double K, Ambient, Diffuse, Specular, DistanceT;
  RAY LightRay;
  RAY ReflectedRay;
  OBJECT *LightSourcePtr;
  COLOR LightColor, NewColor;

  /*
  Calculate the reflected ray R from the normal N and incoming ray V. The
  formula for the reflected ray is generally:
                        R = 2(V dot N)N - V
  assuming V points away from the point of intersection, which it
  does not do in our model since the ray originates at the eye. We
  must therefore reverse V to correspond to our model. To do this,
  we must scale V by -1.0. The result is then:
                       R = -2N(V dot N) + V
  which is the formula we will use here.

  The reflected ray is that which will be traced to find color contributions
  to the point being shaded if the surface being shaded is reflective. The
  reflected ray starts at Point and has a direction away from the surface.
  */
  VDot(Ray->Direction,Normal,K);
  K *= -2.0;

  ReflectedRay.Origin = Point; /* structure copy */

  ReflectedRay.Direction.X = K * Normal.X + Ray->Direction.X;
  ReflectedRay.Direction.Y = K * Normal.Y + Ray->Direction.Y;
  ReflectedRay.Direction.Z = K * Normal.Z + Ray->Direction.Z;
  /*
  We do not have to normalize the reflected ray because
  both V and N were unit vectors. The calculated reflected
  ray is then automatically a unit vector.
  */
  /*
  Start the calculation of the color with the ambient contribution.
  This is light that is everywhere in a scene.
  */
  Ambient = ObjectPtr->Properties.Ambient;
  Color->Red   = ObjectPtr->Properties.Color.Red   * Ambient;
  Color->Green = ObjectPtr->Properties.Color.Green * Ambient;
  Color->Blue  = ObjectPtr->Properties.Color.Blue  * Ambient;

  /*
  Now we must traverse the LightList to find the contribution
  of each light source to the pixel color we are calculating.
  */
  LightSourcePtr=LightList;  /* get us a ptr to list of sources */
  while (LightSourcePtr)  {  /* for each source */
    /*
    Generate a ray from Point to light source of interest. DistanceT
    is the length of the ray from Point to light source. Call
    GetLightColor to see if there are any blocking objects along this
    ray.
    */
```

```
    DistanceT  = MakeLightRay(LightSourcePtr,Point,&LightRay);
    LightColor = GetLightColor(LightSourcePtr,ObjectPtr,&LightRay,DistanceT);

    /* Check to see if our object faces the light */
    VDot(Normal,LightRay.Direction,Diffuse);
    if ((Diffuse > 0.0) && (ObjectPtr->Properties.Diffuse > 0.0)) {
      Diffuse = pow(Diffuse,ObjectPtr->Properties.Brilliance) *
                ObjectPtr->Properties.Diffuse;
      Color->Red   += (LightColor.Red   * ObjectPtr->Properties.Color.Red   * Diffuse);
      Color->Green += (LightColor.Green * ObjectPtr->Properties.Color.Green * Diffuse);
      Color->Blue  += (LightColor.Blue  * ObjectPtr->Properties.Color.Blue  * Diffuse);
    }
    /*
    Note: usually the specular component is calculate by seeing how
    the incident ray V lines up with the reflected light ray.
    The closer they are to being equal, the bigger contribution
    made to the color of the object at Point. In our case, we will
    do the same thing by comparing how the reflected incident ray
    calculated above and called ReflectedRay lines up with the
    LightRay. This requires less computation because ReflectedRay and
    LightRay already point in the same direction. This works because the
    angle between the two sets of rays is the same.
    */
    VDot(ReflectedRay.Direction,LightRay.Direction,Specular);
    /* add highlight */
    if ((Specular > 0.0) && (ObjectPtr->Properties.Specular > 0.0)) {
      Specular = pow(Specular,ObjectPtr->Properties.Roughness) *
      ObjectPtr->Properties.Specular;
      Color->Red   += (LightColor.Red   * Specular);
      Color->Green += (LightColor.Green * Specular);
      Color->Blue  += (LightColor.Blue  * Specular);
    }
    LightSourcePtr = LightSourcePtr->Next;
  }
  /* Now consider reflection */
  K = ObjectPtr->Properties.Reflection;
  if (K > 0.0)  {
    RecurseLevel++;
    Trace(&ReflectedRay,&NewColor);
    Color->Red   += NewColor.Red   * K;
    Color->Green += NewColor.Green * K;
    Color->Blue  += NewColor.Blue  * K;
    RecurseLevel--;
  }
}


/*
Create a Ray from the eye position through the display
at CurX, CurY.
*/
void MakeRay (unsigned CurX,  unsigned CurY, RAY *Ray)   {

  double ScrnX, ScrnY;
  VECTOR TempV1, TempV2;

  /* Convert the X Coordinate to be a double from -0.5 to 0.5 */
  ScrnX  = (CurX - (double) Display.DisplayWidth / 2.0) /
           (double) Display.DisplayWidth;

  /* Convert the Y Coordinate to be a double from -0.5 to 0.5 */
  ScrnY  = (((double)(Display.DisplayHeight -1) - CurY) -
             (double) Display.DisplayHeight / 2.0) /
             (double) Display.DisplayHeight;

  VScale (View.Up,    ScrnY, TempV1);
  VScale (View.Right, ScrnX, TempV2);
  VAdd (TempV1, TempV2, Ray->Direction);
  VAdd (View.Direction, Ray->Direction, Ray->Direction);
  VNormalize (Ray->Direction, Ray->Direction);
  Ray->Origin.X = View.Location.X;
  Ray->Origin.Y = View.Location.Y;
  Ray->Origin.Z = View.Location.Z;
}

/*
Given a ray from the eye position through the screen, trace that ray
into the 3D universe to see what it intersects, if anything. Call
shader to figure out the color of what is hit.
*/
void Trace (RAY *Ray, COLOR *Color)  {

  double T, MinT, NormalDir;
  OBJECT *MinObjectPtr;
  OBJECT *ObjectPtr;
  VECTOR Point;
  VECTOR Normal;

  /* Initialize color to black */
  Color->Red = Color->Green = Color->Blue = 0.0;

  /* Recursion limit check */
```

```
    if (RecurseLevel > MAXRECURSELEVEL)
      return;

    MinT = BIG;
    MinObjectPtr = NULL;
    ObjectPtr = ObjectList;
    /*
    What objects does this ray intersect?  T is the parameter of
    the intersection. A T of 0.0 means nothing was intersected
    by Ray
    */
    while(ObjectPtr)  {
      T = (ObjectPtr->Intersect(ObjectPtr,Ray));
      if ((T > EPSILON) && (T < MinT))  {
        /* Save ptr to closest object intersected */
        MinT = T;
        MinObjectPtr = ObjectPtr;
      }
      ObjectPtr = ObjectPtr->Next;
    }
    /* if nothing was intersected, return background color */
    if (MinT == BIG )  {
      Color->Red   = BackGround.Red;
      Color->Green = BackGround.Green;
      Color->Blue  = BackGround.Blue;
      return;
    }
    /*
    If we get here, we know that our Ray has intersected an object
    use the parameter MinT to calculate the point of intersection.
    */
    Point.X = MinT * Ray->Direction.X + Ray->Origin.X;
    Point.Y = MinT * Ray->Direction.Y + Ray->Origin.Y;
    Point.Z = MinT * Ray->Direction.Z + Ray->Origin.Z;
    /*
    Now find the normal to the intersected object at the point
    of intersection.
    */
    Normal = MinObjectPtr->Normal(MinObjectPtr, Point);
    /*
    Check to see if Normal is pointing towards Ray. If not
    we must reverse the direction of the Normal.
    */
    VDot(Normal,Ray->Direction,NormalDir);
    if (NormalDir > 0.0)
      VNegate(Normal,Normal);

    /* Shade this point */
    Shade(MinObjectPtr,Ray,Normal,Point,Color);
}


/*
This function provides help in the advent of operator error. Program
terminates after help is given
*/

void ShowHelp( void )  {

  printf("Trace is invoked as follows:\n\n");
  printf("Usage: trace [-l]<cr>\n");
  printf("  -l or L renders the larger 320x200 pixel image\n");
  exit(1);
}

/*
Parse the command line to see if larger image format is
required.
*/
void ParseCmdLine(short argc, char *argv[])  {

  unsigned ArgIndex;

  /* Assume we'll render a small 80x50 pixel image with 256 colors */
  Display.DisplayWidth  = 80;
  Display.DisplayHeight = 50;

  printf("\n\nA First Ray Tracer Program - Version: %d.%d\n",Ver,Rev);
  printf("from the book \"Practical Ray Tracing in C\"\n");
  printf("Written by Craig A. Lindley\n\n");

  /* parse all command line arguments */
  for (ArgIndex=1; ArgIndex < argc; ArgIndex++)  {
    if (*argv[ArgIndex] != '-')      /* if not a cmd line switch */
      ShowHelp();                    /* then its an error */
    else  {                          /* its a cmd line switch */
      switch (*(argv[ArgIndex]+1)) { /* parse the cmd line switch */
        case 'h':
        case 'H':
          ShowHelp();
          break;
        case 'l':
```

```
        case 'L':
          /* Select the large image format */
          Display.DisplayWidth  = 320;
          Display.DisplayHeight = 200;
          break;
        default:
          printf("Error - invalid cmd line switch encountered\n");
          ShowHelp();
      }
    }
  }
}

/*
Perform the actual ray tracing of the scene described in the
file "trace.c".
*/
void DoRayTrace(void)  {

  unsigned X, Y;
  RAY Ray;
  COLOR Color;

  time_t timer;
  struct tm *tblock;

  /* Initialize the recurselevel to zero */
  RecurseLevel = 0;

  /* Open the output file for this image */
  OpenOutputFile("ray.raw",Display.DisplayWidth,Display.DisplayHeight);

  /* get the time the ray tracing was started */
  timer = time(NULL);

  /* converts date/time to a structure */
  tblock = localtime(&timer);

  printf("\nBegin Ray Trace on %s",asctime(tblock));
  printf("Tracing Row\n");        /* print the identifier */

  /* Do the ray tracing pixel by pixel, line by line */
  for (Y = 0; Y < Display.DisplayHeight && (!kbhit()); Y++)   {
    printf("%4d",Y);              /* print the current row */
    for (X = 0 ; X < Display.DisplayWidth ; X++)  {
      MakeRay (X,Y,&Ray);         /* calculate eye ray for this pixel */
      Trace (&Ray, &Color);       /* trace it through the scene */
      WritePixelColor(&Color,X);  /* output pixel color to disk */
    }
  }
  /* get the time the ray tracing was completed */
  timer = time(NULL);

  /* converts date/time to a structure */
  tblock = localtime(&timer);

  printf("\nEnd Ray Trace on %s\n",asctime(tblock));
  CloseOutputFile();
  FreeList(&ObjectList);    /* return memory */
  FreeList(&LightList);
}
```

****** STRUCT.H ******

```
/*************************************************************/
/***                    "struct.h"                         ***/
/***          Data Structure Definitions                   ***/
/*************************************************************/

/* Define a three dimensional vector */
typedef struct  {
  double X;                   /* X, Y , Z components */
  double Y;
  double Z;
} VECTOR;

/* Define a color structure */
typedef struct  {
  double Red;                 /* RGB components of a color */
  double Green;
  double Blue;
} COLOR;

/* Define a ray */
typedef struct  {
  VECTOR Origin;              /* a ray has an origin and direction */
  VECTOR Direction;
} RAY;

/* Define the position and direction of our view point */
typedef struct  {
  VECTOR Location;            /* Where our eye is */
  VECTOR Direction;           /* Which direction we are looking */
  VECTOR Up;                  /* Which direction is up */
  VECTOR Right;               /* Which direction is to the right of us */
} VIEWPOINT;

/* Define the display environment */
typedef struct  {
  unsigned DisplayWidth;
  unsigned DisplayHeight;
} DISPLAY;

/* All objects have a set of these surface properties */
typedef struct  {
  COLOR  Color;               /* color of object in RGB */
  double Ambient;             /* ambient factor */
  double Diffuse;             /* diffuse factor */
  double Brilliance;          /* brilliance factor */
  double Specular;            /* specular RGB */
  double Roughness;           /* specular coefficient */
  double Reflection;          /* reflection 0.0 - 1.0 */
} SURFACE;

/* All currently defined object types */
enum ObjectType {NoTAnObject,Light,Sphere,Plane};

/* Define sphere specific info */
struct ASphere  {
  VECTOR  Center;             /* It's location */
  double  Radius;             /* It's radius */
  double  Radius2;            /* It's radius squared */
};

/* Define plane specific info */
struct APlane  {
  VECTOR  Normal;             /* It's defining normal */
  double  Distance;           /* Distance from origin */
};

/* Define light source specific info */
struct ALight  {
  VECTOR  Center;             /* It's location */
};


/* The generic object definition */
typedef struct AnObject  {
  struct AnObject  *Next;     /* ptr to next object in list */
  enum   ObjectType Type;     /* object type for debugging */

  /* function pointers for operations on objects */
  double  (*Intersect)(struct AnObject *, RAY *);
  VECTOR  (*Normal)(struct AnObject *, VECTOR);
  void    (*Print)(struct AnObject *);

  SURFACE Properties;         /* object's properties */
  union  {
    struct ASphere S;         /* object specific info goes here */
    struct APlane  P;
    struct ALight  L;
  } ObjSpecific;
} OBJECT;
```

```
/*********************************************/
/*                                           */
/*       SuperVGA 256 BGI driver defines     */
/*          Copyright (c) 1991               */
/*       Jordan Hargraphix Software          */
/*                                           */
/*********************************************/

#include <dos.h>

typedef unsigned char DacPalette256[256][3];

extern int far _Cdecl Svga256_fdriver[];

/* These are the currently supported modes */
#define     SVGA320x200x256         0       /* 320x200x256 Standard VGA */
#define     SVGA640x400x256         1       /* 640x400x256 Svga/VESA */
#define     SVGA640x480x256         2       /* 640x480x256 Svga/VESA */
#define     SVGA800x600x256         3       /* 800x600x256 Svga/VESA */
#define SVGA1024x768x256    4       /* 1024x768x256 Svga/VESA */

#ifndef XNOR_PUT
#define XNOR_PUT    5
#define NAND_PUT    6
#define NOR_PUT             7
#endif
#define TRANS_COPY_PUT      8

/* Setvgapalette256 sets the entire 256 color palette */
/* PalBuf contains RGB values for all 256 colors      */
/* R,G,B values range from 0 to 63                */
/* Usage:                                         */
/*  DacPalette256 dac256;                         */
/*                                                */
/* setvgapalette256(&dac256);                             */
void setvgapalette256(DacPalette256 *PalBuf)
{
  struct REGPACK reg;

  reg.r_ax = 0x1012;
  reg.r_bx = 0;
  reg.r_cx = 256;
  reg.r_es = FP_SEG(PalBuf);
  reg.r_dx = FP_OFF(PalBuf);
  intr(0x10,&reg);
}
```

```
****** TRACE.H ******
/*************************************************************/
/***                      "trace.h"                        ***/
/***      Misc. Definitions and Function Prototypes        ***/
/*************************************************************/

/* Define a new BYTE type */
#ifndef __BYTE
#define __BYTE
typedef unsigned char BYTE;
#endif

#define TRUE        1
#define FALSE       0

#define BIG       1e10
#define EPSILON   1e-03
#define MAXRECURSELEVEL 5

/* Max width of the display row is set at 640 pixels */
#define MAXLINEWIDTH 640

/* Common Macros */
#define MIN(a,b) ((a)>(b)) ? (b):(a)
#define MAX(a,b) ((a)>(b)) ? (a):(b)
#define SQUARE(x) (x)*(x)

/* Declarations of lists and other globals used in ray tracer */
extern OBJECT *ObjectList;
extern OBJECT *LightList;
extern DISPLAY Display;
extern VIEWPOINT View;
extern COLOR BackGround;
extern double VTemp;

/* Function prototypes for linked listed */
void AddToList (OBJECT **ListPtr, OBJECT *Item);
unsigned FreeList (OBJECT **ListPtr);
unsigned PrintList (OBJECT *ListPtr);

/* Function prototypes for attributes */
void PrintSurfaceAttrib (OBJECT *ObjectPtr);
void SetSurfaceAttrib (OBJECT *ObjectPtr,
                       double Red, double Green, double Blue,
                       double Ambient, double Diffuse, double Brilliance,
                       double Specular, double Roughness, double Reflection);

/* Function prototypes for objects */
OBJECT *MakeSphere (double X, double Y, double Z, double Radius);
OBJECT *MakePlane  (double A, double B, double C, double D);
OBJECT *MakeLight  (double X, double Y, double Z,
                    double Red, double Green, double Blue);

/* Miscellaneous Function Prototypes */
void ParseCmdLine(short argc, char *argv[]);
void Trace (RAY *Ray, COLOR *Color);
void DoRayTrace(void);
double MakeLightRay (OBJECT *LightPtr, VECTOR Point, RAY *Ray);
COLOR GetLightColor (OBJECT *LightPtr, OBJECT *ObjectPtr, RAY *Ray,
                     double Distance);

/* Function prototypes for file output */
int OpenOutputFile (char *FileName, unsigned IWidth, unsigned IHeight);
int CloseOutputFile (void);
int WritePixelColor (COLOR *Color, unsigned Col);
```

```
/*************************************************************/
/***                      "vectors.h"                      ***/
/***             Vector Manipulation Macros                ***/
/*************************************************************/

/*
Various Vector manipulation MACROS
*/

/* Vector Add: a + b = c */
#define VAdd(a, b, c) {(c).X=(a).X+(b).X;(c).Y=(a).Y+(b).Y;(c).Z=(a).Z+(b).Z;}

/* Vector Subtract: c = a - b */
#define VSub(a, b, c) {(c).X=(a).X-(b).X;(c).Y=(a).Y-(b).Y;(c).Z=(a).Z-(b).Z;}

/* Vector Negate: b = -a */
#define VNegate(a, b) {(b).X=-(a).X;(b).Y=-(a).Y;(a).Z=-(a).Z;}

/* Vector Scale: b = k * a */
#define VScale(a, k, b) {(b).X=(a).X*(k);(b).Y=(a).Y*(k);(b).Z=(a).Z*(k);}

/* Vector Dot Product: c = a dot b */
#define VDot(a, b, c) {c=(a).X*(b).X+(a).Y*(b).Y+(a).Z*(b).Z;}

/* Vector Cross Product: c = a cross b */
/* c must be different than a and b */
#define VCross(a,b,c) {(c).X=(a).Y*(b).Z-(a).Z*(b).Y; \
                       (c).Y=(a).Z*(b).X-(a).X*(b).Z; \
                       (c).Z=(a).X*(b).Y-(a).Y*(b).X;}

/* Vector Length: l = len(a) */
#define VLength(a, l) {l=sqrt((a).X*(a).X+(a).Y*(a).Y+(a).Z*(a).Z);}

/* Vector Normalize: u = |a| */
#define VNormalize(a,u) {VTemp=sqrt((a).X*(a).X+(a).Y*(a).Y+(a).Z*(a).Z);\
                         (u).X=(a).X/VTemp;\
                         (u).Y=(a).Y/VTemp;\
                         (u).Z=(a).Z/VTemp;}
```

```
/*************************************************************/
/***                      "vga.c"                          ***/
/***       VGA/SuperVGA Graphic Adapter Functions          ***/
/*************************************************************/

#include <stdio.h>
#include <process.h>
#include <dos.h>
#include "misc.h"
#include "vga.h"

static union REGS regs;

/* Start of VGA/SuperVGA functions */

/*
This function restores the text mode of operation after the display
adapter has been in the graphics mode. This function should always
be called when a graphic program exits.
*/

void SetTextMode( void ) {

  regs.h.ah = 0;              /* set video mode function # */
  regs.h.al = 3;              /* 80x25 text 16 color */
  int86(VIDEO,&regs,&regs);
}

/*
This function determines if a VESA compliant video adapter is present.
If so it return to the calling program. If not, it outputs an error
message and terminates program operation. If this message is received
by the user it is an indication that either the display adapter is not
VESA compliant or that the TSR which makes it so has not been loaded.
If neither is the case, the function "SelectVideoMode" will have to
be rewritten to make the appropriate video mode selections using the
mode numbers for the specific display adapter.
*/

void CheckForVESA (void) {

  unsigned char GLOBALINFO[256];

  /* Determine if a VESA compliant video adapter is present */
  regs.h.ah = VESAFUNCTIONID;
  regs.h.al = 0;              /* return SuperVGA mode function */
  _ES =       FP_SEG(GLOBALINFO);
  regs.x.di = FP_OFF(GLOBALINFO);
  int86(VIDEO,&regs,&regs);
  if ((regs.h.al != VESAFUNCTIONID) || (regs.h.ah != 0)) {
    SetTextMode();
    printf("\nDisplay adapter not VESA compliant !!!\n");
    printf("See text for details\n\n");
    exit(-1);
  }
}

/*
These function get/set the video mode from the VGA
video controller. The first two function deal with
standard VGA devices whereas the second two deal with
VESA SuperVGA devices.
*/

unsigned GetStandardVideoMode(void) {

  regs.h.ah = 0x0F;           /* request the current video mode */
  int86(VIDEO,&regs,&regs);
  return(regs.h.al);          /* return it to caller */
}

void SetStandardVideoMode(unsigned Mode) {

  regs.h.ah = 0;              /* set mode function code */
  regs.h.al = Mode;           /* this is the requested mode */
  int86(VIDEO,&regs,&regs);   /* ask nicely */
}

int GetVESAVideoMode(void) {

  regs.h.ah = VESAFUNCTIONID;          /* request SuperVGA function */
  regs.h.ah = 0x03;                    /* return video mode cmd */
  int86(VIDEO,&regs,&regs);            /* execute request and check result */
  if ((regs.h.al != VESAFUNCTIONID) || (regs.h.ah != 0))
    return(-1);                        /* if error return -1 */
  else
    return(regs.x.bx);                 /* else return video mode to caller */
}

int SetVESAVideoMode(unsigned Mode, unsigned ClearVMemFlag) {

  unsigned TempMode;
```

```
  TempMode = Mode & 0x7FFF;              /* clear bit 15 to clear memory */
  if (!ClearVMemFlag)                    /* if request to not clear memory */
    TempMode |= 0x8000;                  /* set bit 15 to indicate no clear */

  regs.h.ah = VESAFUNCTIONID;            /* request SuperVGA function */
  regs.h.al = 0x02;                      /* set video mode cmd */
  regs.x.bx = TempMode;                  /* this is the requested mode */
  int86(VIDEO,&regs,&regs);              /* ask nicely */
  if ((regs.h.al != VESAFUNCTIONID) || (regs.h.ah != 0))
    return(-1);                          /* if error return -1 */
  else
    return(0);                           /* else return 0 */
}

/* Set a VESA compatible VGA card into an appropriate 256 color mode. */
int SelectVideoMode(unsigned Width, unsigned Height,
                    unsigned ClearVMemFlag)  {

  int ReturnCode = 0;

  if ((Width <= 320) && (Height <= 200))
    ReturnCode = SetVESAVideoMode(Mode320x200,ClearVMemFlag);
  else
  if ((Width <= 640) && (Height <= 400))
    ReturnCode = SetVESAVideoMode(Mode640x400,ClearVMemFlag);
  else
  if ((Width <= 640) && (Height <= 480))
    ReturnCode = SetVESAVideoMode(Mode640x480, ClearVMemFlag);
  else
  if ((Width <= 800) && (Height <= 600))
    ReturnCode = SetVESAVideoMode(Mode800x600,ClearVMemFlag);
  else
    ReturnCode = SetVESAVideoMode(Mode1024x768, ClearVMemFlag);

  return(ReturnCode);
}


/* Set an individual VGA color register */
void SetAColorReg(unsigned RegNum, unsigned Red,
                  unsigned Green,  unsigned Blue)  {

  union REGS regs;

  /*
  With graphics mode set, we can load a color register
  in the DAC.
  */

  /* set a Color Register */
  regs.h.ah = 0x10;
  regs.h.al = 0x10;
  regs.x.bx = RegNum;
  regs.h.dh = Red;
  regs.h.ch = Green;
  regs.h.cl = Blue;
  int86(VIDEO,&regs,&regs);
}

/* Get the color components of a VGA color register */
void GetAColorReg(unsigned RegNum, unsigned *Red,
                  unsigned *Green, unsigned *Blue)  {

  union REGS regs;
  /*
  With graphics mode set, we can read a color register
  from the DAC.
  */

  /* get a Color Register's components */
  regs.h.ah = 0x10;
  regs.h.al = 0x15;
  regs.x.bx = RegNum;
  int86(VIDEO,&regs,&regs);
  /* store the returned values at the pointers */
  *Red   = regs.h.dh;
  *Green = regs.h.ch;
  *Blue  = regs.h.cl;
}
/*
Set and get the color value of the specified pixel on VGA screen. These
functions will work regardless of current video mode. That is why they
are used here.
*/
void SetPixelValue(unsigned Col, unsigned Row, unsigned Value)  {

  /* set a graphic pixel */
  regs.h.ah = 0x0C;
  regs.h.al = Value;
  regs.x.bx = 0;
  regs.x.dx = Row;
```

```
  regs.x.cx = Col;
  int86(VIDEO,&regs,&regs);
}

unsigned GetPixelValue(unsigned Col, unsigned Row)  {

  /* set a graphic pixel */
  regs.h.ah = 0x0D;
  regs.x.bx = 0;
  regs.x.dx = Row;
  regs.x.cx = Col;
  int86(VIDEO,&regs,&regs);
  regs.h.ah = 0;
  return(regs.x.ax);
}

/*
Set all 256 color registers
*/
void SetAllColorRegs(ColorRegister *Palette) {

  /* set a block of Color Registers */
  regs.h.ah = 0x10;
  regs.h.al = 0x12;
  regs.x.bx = 0;
  regs.x.cx = MAX256PALETTECOLORS;
  _ES = FP_SEG(Palette);
  regs.x.dx =FP_OFF(Palette);
  int86(VIDEO,&regs,&regs);
}

/*
Read all 256 color registers
*/
void GetAllColorRegs(ColorRegister *Palette) {

  /* set a block of Color Registers */
  regs.h.ah = 0x10;
  regs.h.al = 0x17;
  regs.x.bx = 0;
  regs.x.cx = MAX256PALETTECOLORS;
  _ES = FP_SEG(Palette);
  regs.x.dx =FP_OFF(Palette);
  int86(VIDEO,&regs,&regs);
}
```

```
****** VGA.H ******

/*************************************************************/
/***                      "vga.h"                          ***/
/***      VGA/SuperVGA Graphic Adapter Header File         ***/
/*************************************************************/

#define VESAFUNCTIONID 0x4F             /* VESA function identifier */

#define MAXNUMCOLREGS 256               /* max number of color registers */
#define MAXCOLREGVAL   63
#define MAXCOLS      1024
#define MAXROWS       768
/*
The following define the VGA 256 color modes. The first is a standard VGA
mode whereas all others are VESA SuperVGA modes.
*/
#define Mode320x200  0x13               /* 320x200  256 color VGA video mode */
#define Mode640x400  0x100              /* 640x400  256 color VESA video mode */
#define Mode640x480  0x101              /* 640x480  256 color VESA video mode */
#define Mode800x600  0x103              /* 800x600  256 color VESA video mode */
#define Mode1024x768 0x105              /* 1024x768 256 color VESA video mode */


#ifndef __BYTE
#define __BYTE
typedef unsigned char BYTE;
#endif

#ifndef __ColorRegister
#define __ColorRegister
typedef struct  {
   BYTE Red;                    /* RGB components of color */
   BYTE Green;                  /* register */
   BYTE Blue;
} ColorRegister;
#endif

/* VGA C Function Prototypes */
void SetTextMode (void);
void CheckForVESA (void);
unsigned GetstandardVideoMode(void);
void SetStandardVideoMode (unsigned Mode);
int  GetVESAVideoMode(void);
int  SetVESAVideoMode (unsigned Mode, unsigned ClearVMemFlag);
int  SelectVideoMode(unsigned Width, unsigned Height,
                     unsigned ClearVMemFlag);
void SetAColorReg (unsigned RegNum, unsigned Red,
                   unsigned Green,  unsigned Blue);
void GetAColorReg (unsigned RegNum, unsigned *Red,
                   unsigned *Green, unsigned *Blue);
void SetPixelValue (unsigned Col, unsigned Row, unsigned Value);
unsigned GetPixelValue (unsigned Col, unsigned Row);
void SetAllColorRegs (ColorRegister *Palette);
void GetAllColorRegs (ColorRegister *Palette);
```

บรรณานุกรม

1 Craig A Lindley "PRACTICAL RAY TRACING IN C" John Wiley & Sons, Inc 1992

2. James Foley, Andries van Dam, Steven Feiner, John Hughes "Computer Graphics PRINCIPLES AND PRACTICE" 2nd Edition , ADDISON WESLEY 1990, 1992

3 เสถียรพงศ์ หุยะนันท์ "RAY TRACING การติดตามรังสี เพื่อสร้างความสมจริง" วารสารคอมพิวเตอร์รีวิว ปีที่ 8 ฉบับที่ 80 เมษายน 2534 หน้า 148-152

4. "EPSON PRINTER LQ-550 USER'S MANUAL" , EPSON COMPUTER CO.,LTD.