การพัฒนาเกม 3 มิติ 3

3D Game Development 3

นายศรุต พจนปรีชากุล
นายศักดิ์พจน์ ทองเลี่ยมนาก

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2543

การพัฒนาเกม 3 มิติ 3

3D Game Development 3

โดย

นายศรุต พจนปรีชากุล

นายศักดิ์พจน์ ทองเลี่ยมนาค

อาจารย์ที่ปรึกษา

ดร. วรวัฒน์ ลิ้มโภคา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2543

ปริญญานิพนธ์ปีการศึกษา2543

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาเกม 3 มิติ 3

3D Game Development 3

ผู้จัดทำ

1. นาย ศรุต พจนปรีชากุล รหัสประจำตัว 40010766
2. นาย ศักดิ์พจน์ ทองเลี่ยมนาค รหัสประจำตัว 40010775

อาจารย์ที่ปรึกษา

(ดร. วรวัฒน์ ลิ้มโภคา)

การพัฒนาเกม 3 มิติ 3

นายศรุต พจนปรีชากุล 40010766
นายศักดิ์พจน์ ทองเลี่ยมนาค 40010775
ดร. วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษา
ปีการศึกษา 2543

บทคัดย่อ

การพัฒนาเกม 3 มิติ 3 เป็นโครงงานที่กล่าวถึงการเขียนโปรแกรมเกม 3 มิติ โดยใช้ไดเร็กเอ็กซ์ และ มอร์ฟิทเอนจิ้น เป็นเครื่องมือช่วยในการพัฒนาโปรแกรม ไดเร็กเอกซ์เป็นเทคโนโลยีที่ถูกสร้างขึ้นมาเพื่อช่วยให้ผู้พัฒนาซอฟต์แวร์ ดึงความสามารถของอุปกรณ์ฮาร์ดแวร์ต่างๆ มาใช้อย่างมีประสิทธิภาพ และให้เกิดประโยชน์สูงสุด ส่วน มอร์ฟิทเอนจิ้น เป็น แอพพลิเคชั่นโปรแกรมอินเทอร์เฟส ที่ใช้ในการพัฒนาโปรแกรม 3 มิติ

ขอบเขตของโครงงานนี้โครงงานนี้เป็นการสร้างเกมในรูปแบบ เกมยิงในมุมมองบุคคลที่ 1 บนระบบปฏิบัติการวินโดวส์ โดยจะใช้มอร์ฟิทเอนจิ้น เป็นหลักในการพัฒนากราฟิก 3 มิติ และในส่วนที่เหลือจะใช้ไดเร็กเอกซ์ 8 เป็นเครื่องมือในการช่วยเขียนโปรแกรม เริ่มจากการสร้างโมเดลจำลองฉากภายในเกมด้วยโปรแกรม สามดี สตูดีโอ แม็กซ์ แล้วจึงใช้มอร์ฟิทเอนจิ้นควบคุมวัตถุ 3 มิติต่างๆ ภายในฉาก รวมถึงการควบคุมลำดับเหตุการณ์ที่เกิดขึ้นภายในเกม และการตอบสนองอินพุตจากผู้เล่นโดยใช้ไดเร็กอินพุต

3D Game Development 3

Sarut Potjanapreshakul

Sakpod Tongleamnak

Dr. Worawat Limpoka Advisor

ABSTRACT

3D Game Development 3 is the project that describes 3D game programming that uses DirectX SDK (Software Development Kit) and Morfit engine. DirectX technology is produced to provide service for software developer to get the hardware optimize performance. Morfit engine is application program interface for develop 3D application.

This thesis is concerned with 3D first person shooting game on windows platform. Using Morfit engine for develop 3D graphic and the rest using DirectX 8 as a programming tool. First, build a scene using 3D Studio MAX. Then control 3D objects in the scene and game event through Morfit engine. Finally response user input via through DirectInput.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือ และร่วมมือจากหลาย ๆ ฝ่ายด้วยกัน บุคคลแรกที่ต้องกล่าวถึงเพราะเป็นส่วนสำคัญที่ทำให้ปริญญานิพนธ์นี้เสร็จลงได้ก็คือ อาจารย์ วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษาปริญญานิพนธ์ ที่ให้ความเอาใจใส่ แนะนำ และช่วยเหลือเสมอมา, ฝ่ายสนับสนุนของเกมเอนจินมอร์ฟิทที่คอยช่วยเหลือผ่านทางจดหมายอิเล็กทรอนิค รวมถึงกลุ่มพัฒนาเกม 3 มิติ อีกกลุ่ม ที่ช่วยกันแก้ไขปัญหาที่เกิดขึ้นในการดำเนินงาน ซึ่งต้องขอขอบพระคุณเป็นอย่างมาก

และต้องขอขอบพระคุณบุคคลสำคัญที่สุดที่ทำให้ข้าพเจ้ามีวันนี้ ก็คือ บิดา มารดา อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูผู้เขียนมาเป็นอย่างดี พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจเอาใจใส่เสมอมา ในทุก ๆ ด้านอันหาที่เปรียบมิได้ ข้าพเจ้าขอระลึกในพระคุณอันสุดประมาณ และขอกราบขอบพระคุณมา ณ ที่นี้

นายศรุต พจนปรีชากุล
นายศักดิ์พจน์ ทองเลี่ยมนาค

สารบัญ

สารบัญ
(ต่อ)

สารบัญ
(ต่อ)

สารบัญ
(ต่อ)

สารบัญภาพ

สารบัญภาพ
(ต่อ)

สารบัญภาพ

(ต่อ)

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

เกมเป็นแอปพลิเคชั่นที่ถูกสร้างขึ้นเพื่อความบันเทิง ซึ่งเครื่องคอมพิวเตอร์ส่วนบุคคลนั้นก็มีแอปพลิเคชั่นทางด้านเกมอยู่มากมายมาย ตั้งแต่สมัยที่เครื่องคอมพิวเตอร์ส่วนบุคคลยังใช้ระบบปฏิบัติการ MS-DOS จนมาถึงปัจจุบันบนระบบปฏิบัติการ Windows ซึ่งเกมนั้นถือได้ว่าเป็นแอปพลิเคชั่นที่ต้องการประสิทธิภาพการประมวลผลที่สูง โดยการพัฒนาของเครื่องคอมพิวเตอร์ส่วนบุคคลส่วนหนึ่งก็ได้รับแรงกระตุ้นจากแอปพลิเคชั่นทางด้านเกม ตามความเป็นจริงแล้วแอปพลิเคชั่นทางด้านเกมถือได้ว่าเป็นแอปพลิเคชั่นทางด้านมัลติมีเดียอย่างหนึ่งเพราะมีการประมวลผลทั้งทางด้านภาพ เสียง และ ส่วนที่ติดต่อกับผู้ใช้งาน

ในอดีตนั้นเกมถูกพัฒนาบนระบบปฏิบัติการ MS-DOS ซึ่งมีข้อดีคือผู้พัฒนานั้นสามารถที่จะเข้าถึงการทำงานของฮาร์ดแวร์ในระดับต่ำ (low-level) ได้โดยตรง ซึ่งทำให้แอปพลิเคชั่นมีประสิทธิภาพการทำงานที่สูง แต่มีข้อเสียคือ ผู้พัฒนาแอปพลิเคชั่นต้องพัฒนาให้แอปพลิเคชั่นของตนสามารถรองรับการทำงานกับอุปกรณ์ของบริษัทต่างๆที่มีอยู่มากมายอยู่ในท้องตลาด เช่น การ์ดแสดงผล และ การ์ดเสียง ที่มีอยู่มากมายหลายยี่ห้อเพื่อให้ครอบคลุมกลุ่มผู้ใช้งานให้ได้มากที่สุด ซึ่งเป็นงานที่ยากลำบากและสิ้นเปลืองงบประมาณเป็นอย่างมาก

ต่อมาบริษัทไมโครซอฟต์ได้เปิดตัวระบบปฏิบัติการ Windows ออกมาซึ่งระบบปฏิบัติการ Windows นี้เป็นระบบปฏิบัติการแบบ GUI (Graphical User Interface) มีข้อดีคือผู้พัฒนาแอปพลิเคชั่นไม่จำเป็นต้องพัฒนาแอปพลิเคชั่นของตน ให้สนับสนุนอุปกรณ์ของบริษัทต่างๆอีกต่อไปเพราะเมื่ออุปกรณ์เหล่านั้นถูกพัฒนามาเพื่อใช้งานกับระบบปฏิบัติการ Windows แล้วผู้พัฒนาแอปพลิเคชั่นเพียงแต่พัฒนาโดยยึดรูปแบบมาตรฐานของระบบปฏิบัติการ Windows ก็เพียงพอ ซึ่งเป็นคุณสมบัติแบบ "device-independent" แต่ระบบปฏิบัติการ Windows นั้นเป็นระบบปฏิบัติการที่มีการแสดงผลด้านกราฟิกที่ช้า ดังนั้นในยุคแรกๆของระบบปฏิบัติการ Windows ผู้พัฒนาแอปพลิเคชั่นทางด้านเกมจึงยังคงยึดติดอยู่กับระบบปฏิบัติการ MS-DOS อยู่ ซึ่งแอปพลิเคชั่นทางด้านเกมที่ทำงานอยู่บนระบบปฏิบัติการ Windows นั้นก็พอมีอยู่บ้างแต่จะเป็นพวกที่ไม่ต้องการการแสดงผลทางด้านกราฟฟิกที่รวดเร็ว เช่น เกมหมากกระดาน และ เกมแนวผจญภัย

ทางบริษัทไมโครซอฟต์ได้เล็งเห็นว่าแอปพลิเคชั่นทางด้านเกมนั้นมีผู้ใช้งานกันอย่างแพร่หลาย จึงมีแนวความคิดที่จะให้ผู้พัฒนาแอปพลิเคชั่นทางด้านเกมและมัลติมีเดียหันมาพัฒนาบนระบบปฏิบัติการ Windows ดังนั้นบริษัทไมโครซอฟต์จึงพัฒนา DirectX ขึ้นมา ซึ่งเป็น libraries ทางด้านมัลติมีเดียที่มีจุดมุ่งหมายหลักๆดังต่อไปนี้

- DirectX ประกอบด้วย libraries ที่ทำงานในระดับต่ำ ซึ่งทำงานได้รวดเร็วและไม่มีข้อจำกัดในการพัฒนาแอปพลิเคชั่นทางด้านเกม
- โครงสร้างของ DirectX ต้องยกภาระในเรื่องฮาร์ดแวร์จากผู้พัฒนาแอปพลิเคชั่นไปสู่ผู้ผลิตฮาร์ดแวร์ ซึ่งผู้ผลิตฮาร์ดแวร์ต้องเป็นผู้สร้าง drivers สำหรับผลิตภัณฑ์ของตน และให้ผู้พัฒนาแอปพลิเคชั่นนั้นสามารถใช้ความสามารถล่าสุดที่มีอยู่ในฮาร์ดแวร์นั้นๆได้
- DirectX นั้นต้องอนุญาตให้ผู้พัฒนาแอปพลิเคชั่นสามารถพัฒนาแอปพลิเคชั่นออกมาในรูปแบบของ Windows application ที่สามารถทำงานบน Desktop ได้และสามารถทำงานร่วมกับฟังก์ชันต่างๆที่มีอยู่บนระบบปฏิบัติการ Windows ได้
- แอปพลิเคชั่นที่ถูกพัฒนาโดยใช้ DirectX นั้นต้องมีประสิทธิภาพที่อยู่สูงกว่าหรืออย่างน้อยที่สุดต้องเทียบเท่ากับประสิทธิภาพของแอปพลิเคชั่นที่ทำงานบนระบบปฏิบัติการ MS-DOS

ซึ่งในปัจจุบันแอปพลิเคชั่นทางด้านเกมและมัลติมีเดียได้หันมาพัฒนาบนระบบปฏิบัติการ Windows กันหมดแล้วเพราะ DirectX นั้นทำให้แอปพลิเคชั่นมีประสิทธิภาพมาก เช่น สามารถใช้ความสามารถของการ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ ที่ติดตั้งอยู่บนเครื่องคอมพิวเตอร์และชุดคำสั่งพิเศษของไมโครโปรเซสเซอร์ได้ อีกทั้งยังช่วยให้ผู้พัฒนาแอปพลิเคชั่นไม่ต้องมายุ่งเกี่ยวในส่วนของฮาร์ดแวร์ที่มีอยู่มากมายหลายยี่ห้ออีกต่อไป โดยที่ยังสามารถเข้าถึงการทำงานในระดับต่ำของฮาร์ดแวร์ได้

ต่อมามีบริษัทซอฟต์แวร์ส่วนใหญ่ได้เล็งเห็นถึงการใช้งาน DirectX ในการพัฒนาโปรแกรม 3 มิติว่าเป็นงานที่ต้องใช้เวลาสูง ทั้งในด้านการศึกษาและพัฒนา หลาย ๆ บริษัทจึงพัฒนา 3D engine ของตัวเองออกมา ซึ่งช่วยลดเวลาในการพัฒนาโปรแกรม 3 มิติลงได้มาก หนึ่งในบริษัทเหล่านั้นได้พัฒนา 3D engine ที่ค่อนข้างแตกต่างจาก engine ทั่วไปซึ่งมักเป็น engine ที่ใช้พัฒนาโปรแกรมเฉพาะด้าน แต่ engine ที่จะใช้ในโครงงานนี้เป็น engine ที่สามารถใช้ในงานพัฒนาโปรแกรม 3 มิติทั่วไปไม่เฉพาะเจาะจง 3D engine ตัวนี้มีชื่อว่า Morfit

1.2 จุดประสงค์ของโครงงาน

1. ศึกษาและทำความเข้าใจในการพัฒนาโปรแกรม 3 มิติด้วยการใช้มอร์ฟิทและ DirectX
2. พัฒนาแอปพลิเคชั่นทางด้านเกมในรูปแบบ 3 มิติ โดยจะเน้นการใช้มอร์ฟิทเป็นหลักในการพัฒนาด้าน graphics 3 มิติ
3. พัฒนาส่วนอื่น ๆ ของเกมในส่วนที่ไม่ใช่ graphics 3 มิติ ด้วย DirectX

1.3 ขอบเขตของโครงงาน

โครงงานนี้เป็นการสร้างเกมในรูปแบบ First's person shooting 3 มิติ บนระบบปฏิบัติการ Windows โดยจะใช้มอร์ฟิทเป็นหลักในการพัฒนากราฟิก 3 มิติ และในส่วนที่เหลือจะใช้ DirectX

1.4 แนวทางดำเนินงาน

1. ทำการค้นหา และศึกษาเครื่องมือต่างๆ ที่ใช้ในการพัฒนาเกม 3 มิติ ทั้งใน internet และหนังสือ
2. ศึกษาข้อดี ข้อด้อย ของเครื่องมือที่หามาได้ รวมทั้งความเข้ากันได้กับเครื่องมือที่ใช้ในการทำโมเดล 3 มิติ ต่างๆ
3. เลือกและศึกษาการใช้งานเครื่องมือที่จะใช้ในการพัฒนา
4. ทำการวางรูปแบบของเกม กฎ กติกา และวิธีการเล่น
5. ทำโมเดลของ วัตถุต่างๆ ตัวละคร และฉากที่มีในเกม รวมทั้งทดสอบการ import เข้าไปใน 3D Engine
6. เขียนโปรแกรมตามให้ได้ตามรูปแบบของเกมที่วางไว้
7. ทดสอบการเล่นเบื้องต้น รวมทั้งหาข้อผิดพลาดของโปรแกรม
8. ทำการแก้ข้อผิดพลาด อาจจะเพิ่มฉาก และตัวละครที่มี และพัฒนาความสวยงามของเกม
9. เสร็จสิ้นและสรุป การพัฒนา

ในภาคเรียนนี้เราได้ทำการพัฒนาจนเสร็จสมบูรณ์แล้ว

1.5 สาเหตุที่เลือกใช้กราฟิกเอนจิ้นในการพัฒนา

1. เวลา เนื่องจากมีเวลาจำกัด การพัฒนา เอนจิ้น 3 มิติใช้เองนั้น เป็นเรื่องยากมาก
2. ความง่าย เนื่องจากพัฒนา เอนจิ้น 3 มิติ ขึ้นใหม่ เป็นเรื่องยากมาก
3. ประสิทธิภาพ ของเอนจิ้นที่พัฒนาขึ้นเอง อาจจะไม่ดีเท่าที่มีอยู่แล้ว
4. มาตรฐาน เอนจิ้นที่พัฒนาขึ้นเอง จะไม่มีมาตรฐาน และไม่แพร่หลาย
5. คุณภาพ ของเอนจิ้นที่พัฒนาขึ้นเอง อาจจะไม่ดีเท่าที่มีอยู่แล้ว

1.6 สาเหตุที่เลือกใช้มอร์ฟิท (Morfit) เอนจิ้นในการพัฒนา

1. ประสิทธิภาพ เนื่องจากมอร์ฟิท ให้ประสิทธิภาพในการแสดงผล 3 มิติ และคุณภาพที่พอรับได้
2. เทคโนโลยี เนื่องจากมอร์ฟิท ใช้เทคโนโลยีที่เป็นมาตรฐานในปัจจุบัน จึงมั่นใจได้ว่าจะใช้กับเครื่องมือต่างๆ ในปัจจุบันได้
3. ความง่าย ในการใช้งาน สามารถแสดงผลโลก 3 มิติได้ด้วย การเขียนโปรแกรมเพียงไม่กี่บรรทัด
4. สามารถควบคุมได้ทั้งหมด เนื่องจาก มอร์ฟิท สามารถควบคุมโลก 3 มิติ ได้ในทุกแง่มุมทั้งแต่ โพลีกอน เล็กๆ ไปจนถึง วัตถุขนาดใหญ่
5. ใช้งานได้หลากหลาย เนื่องจาก มอร์ฟิท สามารถพัฒนาโปรแกรมได้หลากหลาย ไม่จำเป็นต้องเป็น เกม

6. ใช้เวลาในการเรียนรู้น้อย
7. ประกอบไปด้วยเครื่องมือ ที่ครบถ้วน
8. ความยืดหยุ่น สามารถพัฒนาได้หลาย ภาษา เช่น VC++, Borland C Builder, Visual Basic
9. สามารถอัพเกรด ได้ง่าย เพียงแค่แทนที่ dll และ header file
10. ฟรี !!!

1.7 สาเหตุที่เลือกใช้ไดเร็กเอกซ์

1. ประสิทธิภาพ ให้ประสิทธิภาพสูงสุดเท่าที่ ฮาร์ดแวร์จะรองรับ
2. มีเอกสารให้ศึกษามากมาย
3. เป็นมาตรฐานในปัจจุบัน
4. มีการใช้งานอย่างแพร่หลาย
5. สามารถอัพเกรดได้ง่าย มีความเข้ากันได้ย้อนหลัง

บทที่ 2
แนะนำไดเร็กเอ็กซ์

2.1 ไดเร็กเอ็กซ์คืออะไร

ไมโครซอฟต์ไดเร็กเอ็กซ์ (Microsoft DirectX) คือ กลุ่มของแอปพลิเคชั่นโปรแกรมมิ่งอินเตอร์เฟส (Application Programming Interfaces) หรือ เอพีไอ (APIs) ที่ช่วยสนับสนุนการสร้างแอปพลิเคชั่นแบบเรียลไทม์ (Real-Time Applications) ด้าน เกม และ มัลติมีเดียที่ต้องการประสิทธิภาพการทำงานสูง

2.2 จุดประสงค์ของไดเร็กเอ็กซ์

ไมโครซอฟต์พัฒนาไดเร็กเอ็กซ์เพื่อ เพิ่มประสิทธิภาพการทำงานด้านมัลติมีเดียของแอปพลิเคชั่นที่ทำงาน
อยู่บนระบบปฏิบัติการไมโครซอฟต์วินโดวส์ (Microsoft Windows) ให้มีความสามารถเทียบเท่าหรือสูงกว่า ประสิทธิภาพของแอปพลิเคชั่นที่ทำงานอยู่บนระบบปฏิบัติการ MS-DOS หรือเครื่องเกมประเภทคอนโซล (Game Consoles)

2.3 ประโยชน์ของการพัฒนาแอปพลิเคชั่นโดยใช้ไดเร็กเอ็กซ์

ไมโครซอฟต์พัฒนาไดเร็กเอ็กซ์ขึ้นมาโดยมีจุดประสงค์หลักคือเพื่อสนับสนุนการพัฒนาแอปพลิเคชั่นทาง
ด้านเกมบนระบบปฏิบัติการไมโครซอฟต์วินโดวส์ ก่อนที่จะมีไดเร็กเอ็กซ์นั้นการพัฒนาแอปพลิเคชั่นทางด้านเกมสำหรับเครื่องคอมพิวเตอร์ส่วนบุคคลนั้นจะกระทำบนระบบปฏิบัติการ MS-DOS ซึ่งผู้ที่ทำการพัฒนาแอปพลิเคชั่นเกมต้องทำการพัฒนาแอปพลิเคชั่นให้สามารถใช้ได้กับอุปกรณ์ยี่ห้อต่างๆที่อยู่มากมายในท้องตลาด ส่วนการพัฒนาแอปพลิเคชั่นโดยใช้ไดเร็กเอ็กซ์นั้นผู้พัฒนาจะได้รับประโยชน์จากดีไวซ์อินดีเพนเดนท์ (device-independent) แต่ยังคงคุณสมบัติไดเร็กแอ็คเซส (Direct Access) ของอุปกรณ์ไว้ ซึ่งหมายความว่าเราสามารถพัฒนาแอปพลิเคชั่นโดยได้ไม่ต้องคำนึงถึงว่าจะมีอุปกรณ์อยู่มากมายหลายยี่ห้อในท้องตลาด เพราะแอปพลิเคชั่นของเราจะสามารถทำงานได้กับอุปกรณ์ทุกยี่ห้อที่สนับสนุนไดเร็กเอ็กซ์และจะสามารถเข้าถึงการทำงานในระดับต่ำ (low-level) ของอุปกรณ์นั้นๆได้ อีกทั้งยังสามารถใช้ความสามารถจากการเร่งความเร็วโดยใช้ฮาร์ดแวร์ที่มีอยู่บนอุปกรณ์ต่างๆ เช่น การ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ และ ชุดคำสั่งพิเศษของไมโครโพรเซสเซอร์ เช่น ชุดคำสั่ง MMX, 3D NOW!, SSE ได้อีกด้วย

ไดเร็กเอ็กซ์จะให้ประโยชน์ใน 2 ด้านคือ

1. ประโยชน์ในด้านการพัฒนา DirectX Windows Application
2. เป็นมาตรฐานสำหรับการพัฒนาฮาร์ดแวร์

2.4 ส่วนประกอบของ DirectX

ในส่วนนี้จะให้รายละเอียดของส่วนประกอบต่างๆของ DirectX อย่างคร่าวๆ ส่วนประกอบของ DirectX ได้แก่

- **DirectDraw** จะเป็นเทคนิคการเร่ง ฮาร์ดแวร์ และซอฟต์แวร์อนิเมชั่น โดยการเข้าถึงรูปภาพบิตแมปในหน่วยความจำ(ไม่ต้องผ่านระบบปฏิบัติการ Windows) ทำให้แอปพลิเคชั่นของเราต้องการเพียงการจัดการฮาร์ดแวร์พื้นฐานง่ายๆ เราไม่ต้องเรียก procedure ที่เจาะจงสำหรับจัดการกับสีต่างๆ รวมทั้งรองรับการจัดการกับ palettes, clipping และ animation การใช้ DirectDraw ทำให้จัดการกับหน่วยความจำได้ง่ายขึ้น

  DirectDraw จะช่วยผู้เขียนโปรแกรมดังนี้

  - Hardware abstraction layer (HAL) ของ DirectDraw จะติดต่อโดยรงกับฮาร์ดแวร์โดยตรง ทำให้ฮาร์ดแวร์สามารถแสดงผลที่สูงสุดได้
  - DirectDraw จะประเมินความสามารถของฮาร์ดแวร์ของเรา และใช้งานให้มีประสิทธิภาพสูงสุด นอกจากนั้น DirectDraw ยังมี Hardware emulation layer (HEL) ที่รองรับการทำงานเมื่อไม่มีฮาร์ดแวร์มารองรับ
  - DirectDraw อยู่ภายใต้ระบบปฏิบัติการ Windows ดังนั้นจะได้รับข้อดีในการอ้างอิงหน่วยความจำ 32-bit และ flat memory model ที่ระบบปฏิบัติการจัดหาให้
    DirectDraw แสดงหน่วยความจำวีดีโอ (video memory) และหน่วยความจำระบบ (system memory)เป็นบล็อกขนาดใหญ่ ไม่ใช่เป็น segment เล็กๆ ที่ใช้อ้างอิงหน่วยความจำเป็น segment:offset เหมือนสมัยก่อน
  - รองรับแอปพลิเคชั่นทั้งแบบเต็มหน้าจอ (full-screen mode) และไม่เต็มหน้าจอ (windowed mode)
  - รองรับ 3D Z-buffers
  - รองรับการช่วยเหลือจากฮาร์ดแวร์ด้วย Z-Ordering
  - เข้าถึงหน่วยความจำมาตรฐาน และหน่วยความจำของส่วนแสดงผลได้พร้อมกัน

- **DirectSound** ทำให้ใช้ฮาร์ดแวร์ และซอฟต์แวร์ทางด้านเสียง(การ mix และเล่น)ได้ และเข้าถึงฮาร์ดแวร์ด้านเสียงได้โดยตรง DirectSound จะทำให้การทำงานของ drivers ต่างๆ สามารถเข้ากันได้ DirectSound สามารถรับข้อมูลเสียง และเล่นได้ DirectSound ยังรองรับคุณสมบัติต่างๆที่จะทำให้ผู้พัฒนาโปรแกรมนำข้อดีต่างๆที่เพิ่มขึ้นมาจากการ์ดเสียง และ drivers ต่างๆที่เกี่ยวข้อง

DirectSound ยังช่วยให้ทำงานง่ายขึ้นเช่น

- สามารถตอบฮาร์ดแวร์ขณะที่ทำงานอยู่ เพื่อตัดสินหาวิธีการตั้งค่าต่างๆ ที่ดีที่สุดสำหรับเครื่องคอมพิวเตอร์ส่วนบุคคลแต่ละเครื่อง
- สามารถใช้คุณสมบัติต่างๆ ของฮาร์ดแวร์ใหม่ แม้ว่ามันจะไม่ได้รองรับ DirectSound โดยตรง
- จัดการกับเสียงในระบบ 3 มิติได้
- รับข้อมูลเสียงได้

- **DirectMusic** เป็นอุปกรณ์ทางด้านเสียงเพลงของ DirectX ไม่เหมือนกับ DirectSound ตรงที่ DirectMusic จะเล่นข้อมูลที่เป็นเพลง

  DirectMusic รองรับมาตรฐาน Musical Instrument Digital Interface (MIDI) และ downloadable sounds (DLS), DirectMusic จะมี tools สำหรับแต่ง และเล่นเพลง

- **DirectPlay** ทำให้การสร้างเกมระบบผู้เล่นหลายๆ คนง่ายขึ้น และสะดวกในการใช้งานร่วมกับโมเด็ม DirectPlay จะจัดหาการติดต่อสื่อสารที่มีประสิทธิภาพให้กับแอปพลิเคชั่นทำให้ผู้พัฒนาแอปพลิเคชั่นไม่ต้องมาพะวงกับการเชื่อมต่อโพรโตคอลต่างๆ เข้าด้วยกัน ทำให้พัฒนาแอปพลิเคชั่นของตนได้ง่ายขึ้น

- **Direct3D** ใช้ในการสร้างเกมในรูปแบบ 3 มิติ บนระบบปฏิบัติการ Windows ซึ่งได้เปรียบจากการที่สามารถใช้งานร่วมกับการ์ดเร่งความเร็วกราฟิกแบบ 3 มิติ ได้อย่างมีประสิทธิภาพเพราะมีฟังก์ชันคอยสนับสนุนนั่นเอง นอกจากนั้นสามารถรองรับการทำงานบนระบบปฏิบัติการ Windows NT ได้ และใช้เทคโนโลยี MMX ได้โดยตรง Direct3D มี 2 ส่วนประกอบ คือ Direct3D Retained-Mode API และ Direct3D Immediate Mode API

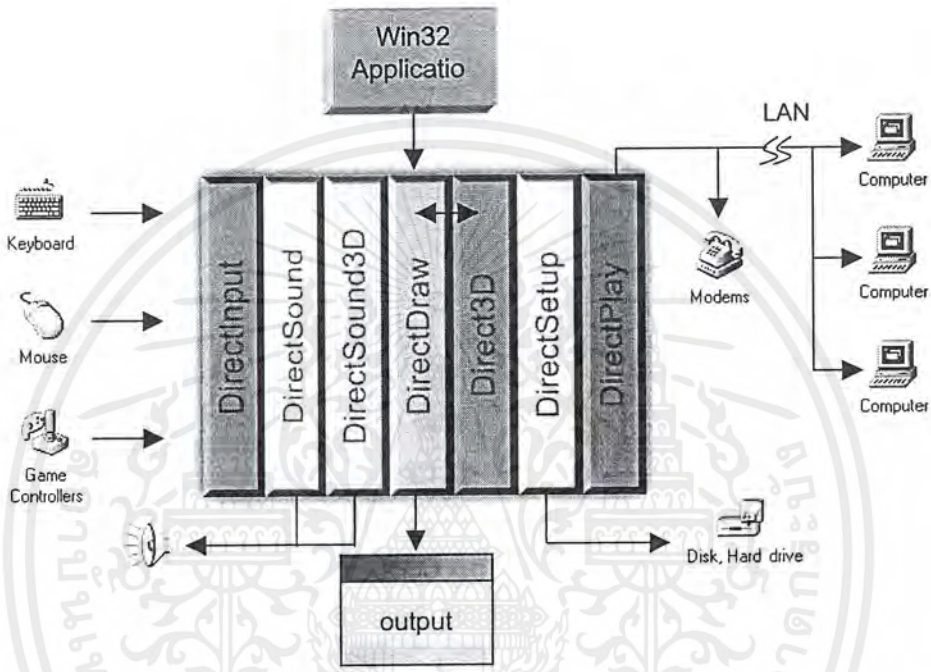
  ผู้พัฒนาส่วนมากจะใช้ Immediate Mode แทน Retained-Mode สำหรับโปรแกรมที่ต้องการความละเอียดทางกราฟิกสูง

- **DirectInput** ทำให้การสร้างเกมเพื่อรองรับระบบการสั่งงานจากผู้เล่นไม่ว่าจะเป็นจอยสติ๊ก เมาส์ หรือคีย์บอร์ด ทำได้ง่ายและมีประสิทธิภาพเป็นอย่างยิ่ง อีกทั้งยังมีฟังก์ชันที่สนับสนุนการเพิ่มเติมอุปกรณ์การเล่นที่จะมี ใหม่ในอนาคต

  DirectInput จะทำให้สามารถเข้าถึงข้อมูลอินพุตได้เร็วกว่าเดิม เพราะมันไม่ต้องพึ่งพาการทำงานของระบบปฏิบัติการ Windows แต่จะสื่อสารโดยตรงกับฮาร์ดแวร์

- **DirectSetup** ช่วยในการติดตั้งส่วนประกอบที่จำเป็นของ DirectX ให้กับผู้ใช้อย่างอัตโนมัติ ทำให้สะดวกมาก DirectSetup จะจัดหาวิธีการอัตโนมัติที่จะติดตั้งแอปพลิเคชั่นให้เหมาะสมกับระบบปฏิบัติการ Windows

- **AutoPlay** เป็นส่วนประกอบสำคัญในระบบปฏิบัติการ Windows ทำให้แอปพลิเคชั่นสามารถที่จะทำการติดตั้งลงไปในฮาร์ดิสก์ได้อย่างอัตโนมัติ โดยแอปพลิเคชั่นจะเริ่มทำงานเมื่อเรานำแผ่นโปรแกรมเกมใส่เข้าไปในไดรฟ์ซีดีรอม AutoPlay เป็นส่วนของ Microsoft Win32 API ใน SDK และไม่เหมือนกับใน DirectX



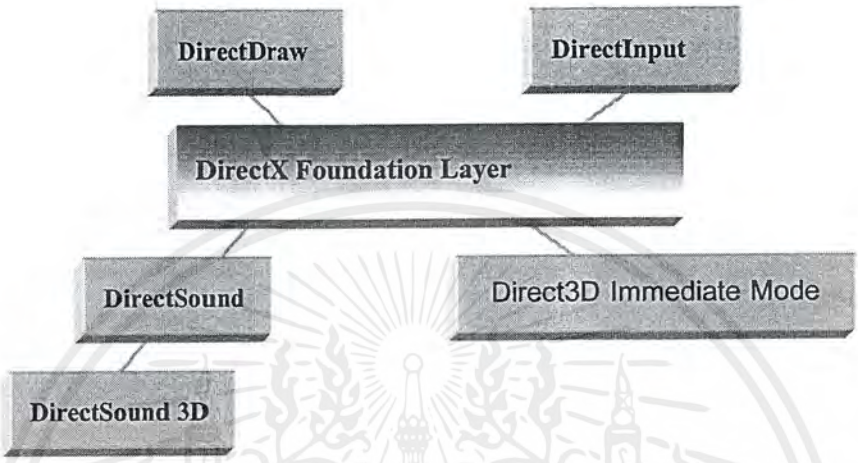
รูปที่ 2-1 ส่วนประกอบของ DirectX

โดยที่ในโครงงานนี้จะพัฒนาแอปพลิเคชั่นเกมแบบ 3 มิติ ในมุมมองของบุคคลที่หนึ่ง ในส่วนของกราฟิกจะเป็นหน้าที่ของมอร์ฟิทเอนจิ้น และจะใช้ DirectInput เพื่อใช้จัดการเกี่ยวกับการรับอินพุต ทั้งจากคีย์บอร์ดและเมาส์

2.5 การแบ่ง Layers ของ DirectX components

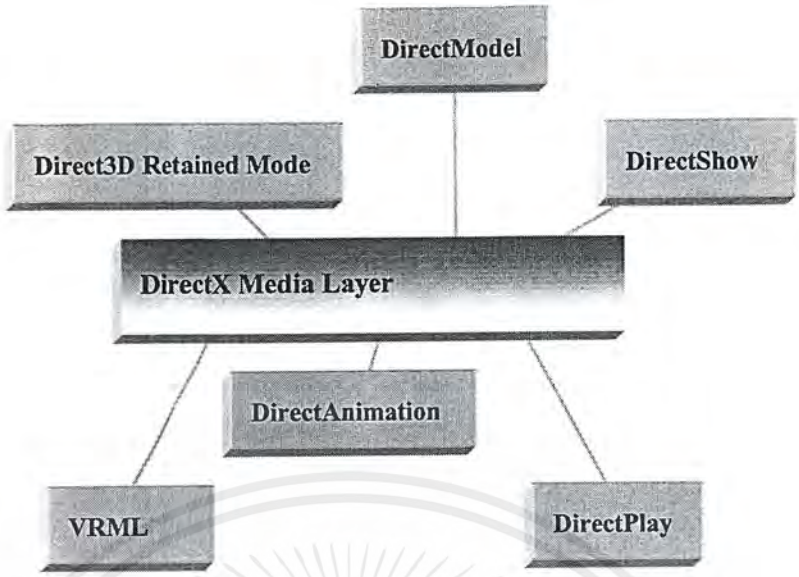
DirectX นั้นแบ่งออกได้เป็น 2 ส่วนหลักๆ คือ DirectX Foundation Layer จะทำหน้าที่เป็นตัวกลางในการติดต่อระหว่างแอปพลิเคชั่นกับฮาร์ดแวร์ ผู้ผลิตซอฟต์แวร์ก็เพียงแต่เขียนโปรแกรมให้สนับสนุนมาตรฐานทั้ง 4 ชนิดของ DirectX Foundation Layer โดยไม่จำเป็นต้องคำนึงถึงฮาร์ดแวร์ที่ต้องทำงานด้วย เพราะผู้ผลิตฮาร์ดแวร์ต่างๆที่จะใช้ทำงานร่วมกับระบบปฏิบัติการ Windows จะต้องผลิตฮาร์ดแวร์ให้

สนับสนุนมาตรฐานของ DirectX Foundation Layer นี้เช่นกัน DirectX Foundation Layer แสดงในรูปที่ 2-2



รูปที่ 2-2 Components ของ DirectX Foundation Layer

อีกส่วนได้แก่ DirectX Media Layer เป็นส่วนที่นำฟีเจอร์ต่างๆของ DirectX Foundation Layer มาประสานงานใช้ร่วมกันทำให้เกิดฟีเจอร์ใหม่ๆ ซึ่ง DirectX Foundation Layer จะสั่งงานฮาร์ดแวร์โดยรับคำสั่งจาก DirectX Media Layer ซึ่งรับคำสั่งจากซอฟต์แวร์อีกทีหนึ่ง DirectX Media Layer แสดงดังรูปที่ 2-3



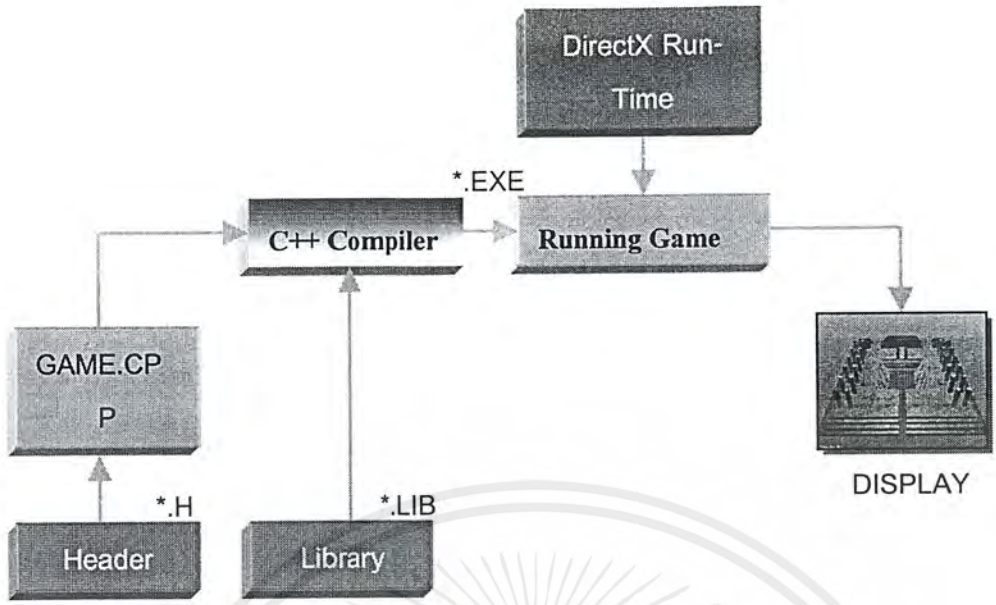
รูปที่ 2-3 Components ของ DirectX Media Layer

2.6 ประเภทของ DirectX library

DirectX Library นั้นมีอยู่ 2 รูปแบบด้วยกัน คือ

1. **DirectX Run-Time** เป็น driver สำหรับผู้ใช้ (end-user) แอปพลิเคชั่นที่ถูกพัฒนาโดยใช้ DirectX โดยที่ DirectX Run-Time นั้นจะอยู่ในรูปแบบของไฟล์ *.DLL ซึ่งจะถูกเรียกใช้โดยแอปพลิเคชั่นเมื่อแอปพลิเคชั่นทำงาน โดยทั่วไปแล้วผู้พัฒนาแอปพลิเคชั่นที่ใช้ DirectX นั้นจะให้ DirectX Run-Time มาพร้อมกับแอปพลิเคชั่นอยู่แล้ว

2. **DirectX SDK** เป็นส่วนที่มีไว้สำหรับผู้พัฒนาแอปพลิเคชั่นโดยที่ DirectX SDK นี้จะประกอบด้วยไฟล์ *.LIB และ ไฟล์ *.H สำหรับการพัฒนาแอปพลิเคชั่นโดยใช้คอมไพเลอร์ภาษาซีพลัสพลัส (C++ Compiler) มีเอกสารสำหรับช่วยเหลือในการพัฒนาแอปพลิเคชั่น มีตัวอย่างแอปพลิเคชั่นที่แสดงถึงฟังก์ชันการทำงานต่างๆของ DirectX พร้อม source code โดยที่เราสามารถดาวน์โลด DirectX SDK ได้จาก Web site ของไมโครซอฟต์

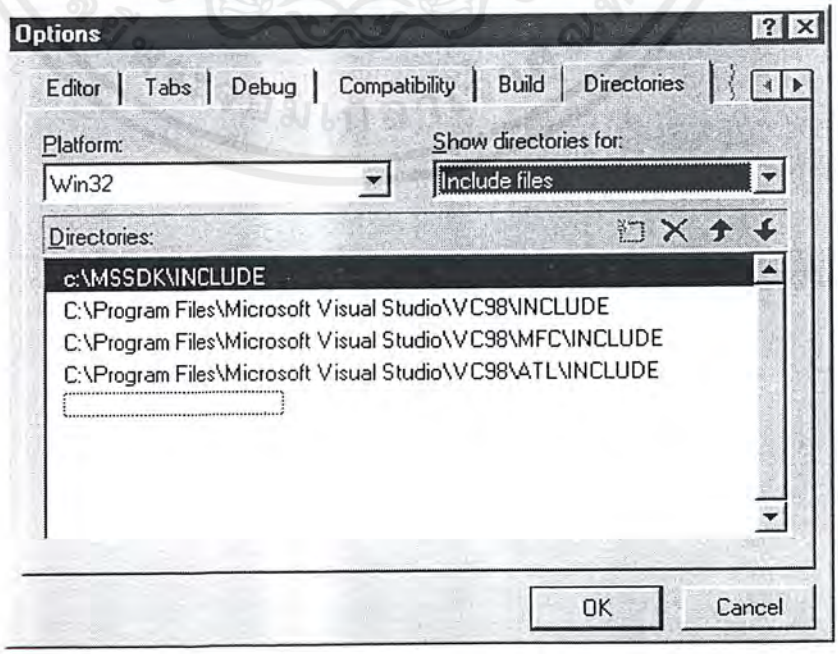


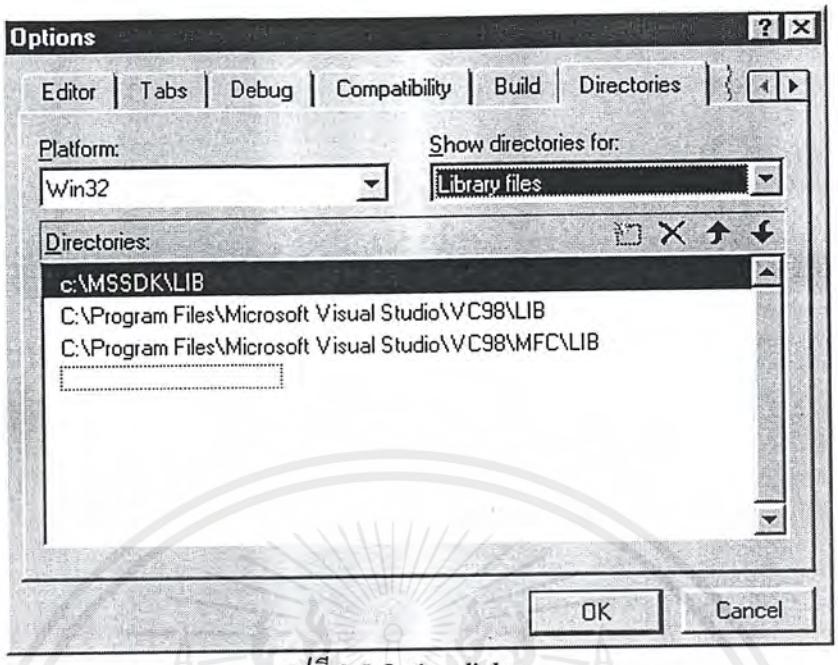
รูปที่ 2-4 โครงสร้างของแอปพลิเคชั่นและ DirectX Library

2.7 การใช้ Visual C++ กับ DirectX SDK

การใช้ DirectX SDK ร่วมกับคอมไพเลอร์ Visual C++ นั้นเราบอกให้คอมไพเลอร์ทราบว่า directory ของ

Header files และ Library files นั้นอยู่ที่ใด โดยเราสามารถทำได้โดยเลือก "Options" จาก เมนู "Tool" ของคอมไพเลอร์ Visual C++ จะปรากฏ Options dialog ดังรูปที่ 2-5

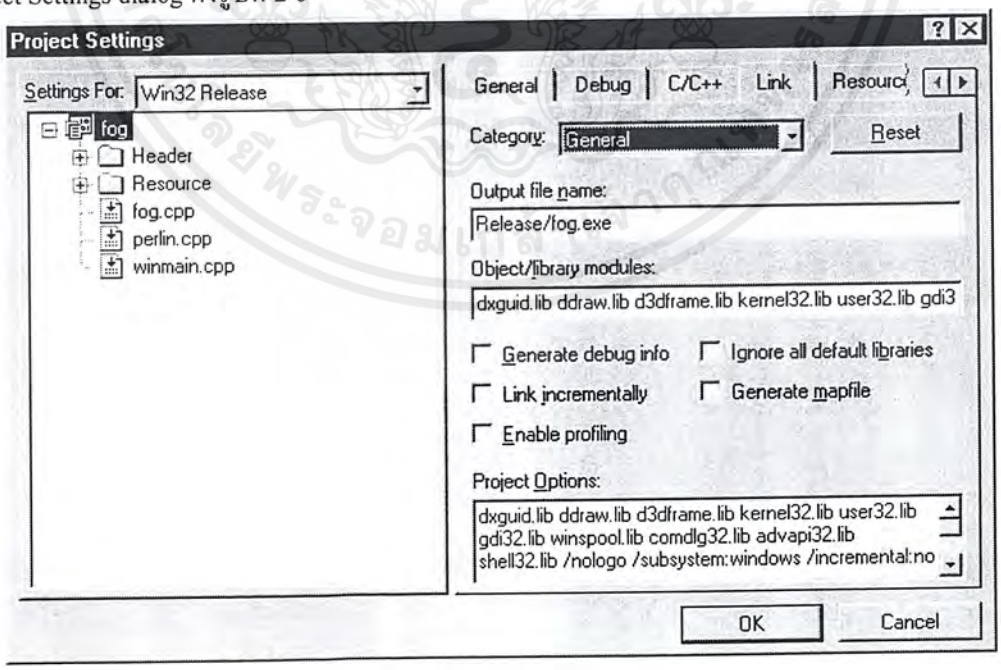




รูปที่ 2-5 Option dialog

เราต้องเลื่อน directory ที่เราเพิ่มเข้าไปใหม่นั้นให้อยู่บนสุดมิเช่นนั้น Visual C++ จะใช้ DirectX Library ที่ถูกติดตั้งมาพร้อมกับตัวคอมไพเลอร์เองซึ่งอาจเป็นเวอร์ชันที่เก่ากว่าเวอร์ชันที่เราต้องการใช้งาน

นอกจากนี้เราต้องเซต Project link libraries เพื่อให้คอมไพเลอร์ลิงค์ DirectX Library เข้ากับแอปพลิเคชั่นของเรา โดยเลือก "Settings" จาก เมนู "Project" ของคอมไพเลอร์ Visual C++ จะปรากฏ Project Settings dialog ดังรูปที่ 2-6



รูปที่ 2-6 Project Settings dialog

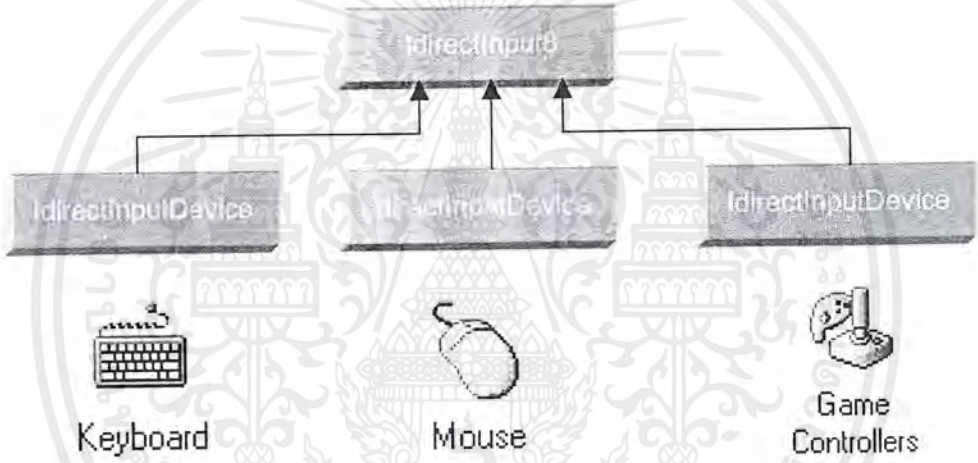
บทที่ 3

DirectInput

สถาปัตยกรรมพื้นฐานของ DirectInput ประกอบด้วย DirectInput object ซึ่งรองรับ COM interface และ object สำหรับ input device แต่ละตัวที่ส่งข้อมูลเข้ามา โดย device แต่ละอันจะมี "object instances" ของมันเอง ซึ่งแต่ละตัวจะมีการควบคุม ที่แตกต่างกัน เช่น ปุ่ม, แกน เป็นต้น

คำว่า "object" ในที่นี้ใช้แสดงสิ่งต่างๆ ที่สร้างโดย ระบบ DirectInput ที่รองรับ method ของ COM interface โดย method เหล่านี้ไม่ได้ถูกใช้เรียกผ่าน OOP เช่น ภาษา VC++

ในด้านความเร็ว DirectInput ทำงานโดยตรงกับ อุปกรณ์ต่างๆ โดยผ่านระบบ message ของ Windows



รูปที่ 3-1 การติดต่อของ DirectInput

3.1 ขั้นตอนการสร้าง Direct Input

1. **สร้าง Direct Input** ขึ้นมา โดยเรียกฟังก์ชัน DirectInput8Create โดยการ return pointer ไป IdirectInput8 Interface เพื่อสร้าง object
2. **Enumerating Device** คือการร้องขอ Direct Input ให้หา input device ที่ต่ออยู่

Note: 1. Mouse และ keyboard อาจไม่ต้องใช้

2. Joystick จำเป็นเพราะ joystick มีหลายชนิด ทำให้ GUID ไม่ว่างพอเหมือน mouse และ keyboard เราต้อง query ว่า joystick ที่ต่ออยู่ แล้วรับ GUID มาแล้วจึงใช้ ID เหล่านี้

การ Enumerate device ใดๆ จะเรียก callback function ก่อนแล้วจึงเรียก EnumDevice method

```
HRESULT EnumDevices( DWORD dwDevType,           //ชนิดของ device ที่หา
                     LPDIENUMCALLBACK lpCallback,   //address ของ callback function
                     LPVOID pvRef,              //ค่า 32-bit ที่ใช้ส่งกลับ
                     DWORD dwFlags );           //flag ว่าจะ enumerate แบบไหน
```

3. **Setting up Device**
   - Create Device: สร้าง input device ต้องมี GUID สำหรับแต่ละ device โดยอาจจะร้องขอจาก Direct Input Enumerate

     Note: ถ้าไม่ได้ enumerate จาก guidInstance ของ DIDEVICEINSTANCE structure ที่ DirectInput ส่งมายัง callback function ของเรา เราจะต้องกำหนดเองก่อน เช่น GUID_SysMouse or GUID_SysKeyboard
   - Setting Data Format: ต้องรู้โครงสร้างของข้อมูลที่มาจาก device ต่างๆ เพราะ device ต่างๆ มีขนาดข้อมูลไม่เท่ากัน
   - Getting Data Format about a device
   - Identify Device Things: มี 2 แบบ คือ โดย Offset กับโดย ID
   - Set Property of device
   - Set Cooperative level: set เป็น DISCL_NONEXCLUSIVE | DISCL_BACKGROUND

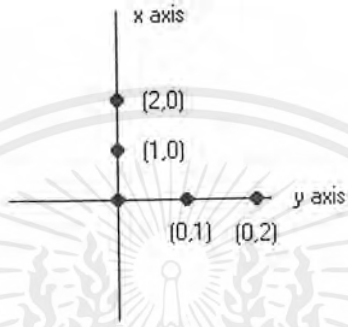
4. **Acquire** การอนุญาตให้ใช้ device และแจ้ง Direct Input ว่าต้องการข้อมูลจาก device ตามรูปแบบของข้อมูลที่กำหนดไว้
5. **Unacquire** การยกเลิกการใช้ Device จาก Direct Input
6. **Release** การยกเลิกการใช้ Direct Input

Note: Poll สำหรับ Joystick โดย Direct Input จะตรวจสอบว่า device ต้อง polling หรือไม่ ถ้าต้องการจะเก็บ state ที่เปลี่ยนไป(โดย Hardware Interrupt) ทันที

บทที่ 4
พื้นฐานมอร์ฟิท

4.1 ระบบโคออดิเนต 2 มิติ

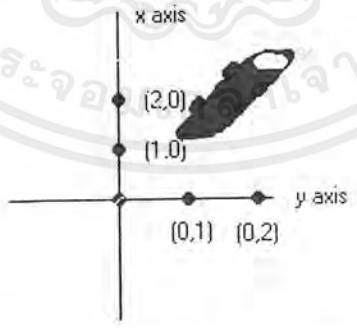
การที่จะเข้าใจระบบโคออดิเนต 3 มิติ ต้องเข้าใจระบบโคออดิเนต 2 มิติก่อน จากรูปที่ 4-1 แกน x ค่าบวก คือ เส้นตรงตรงที่เริ่มจากจุดออริจินผ่านจุด (1,0) หรือสามารถเขียนได้อีกอย่างว่า เวคเตอร์ [1,0] ส่วนแกน y ค่าบวก คือ เส้นตรงตรงที่เริ่มจากจุดออริจินผ่านจุด (0,1) หรือสามารถเขียนได้อีกอย่างว่า เวคเตอร์ [0,1] ส่วนแกน x ค่าลบจะอยู่ใต้จุดออริจิน และ แกน y ค่าลบจะอยู่ทางด้านซ้ายของจุดออริจิน



รูปที่ 4-1 แกน x-y ของระบบโคออดิเนต 2 มิติ

4.1.1 World Space กำหนดให้ World coordinate system หรือ World Space มีแกน X ด้านบวกชี้ขึ้นไปข้างบนเสมอ และ แกน y ด้านบวกชี้ไปทางขวาเสมอ

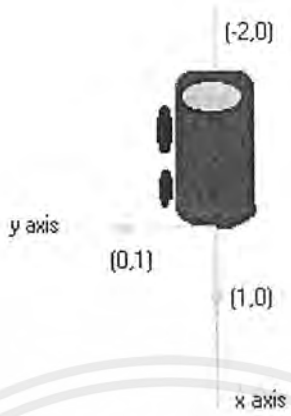
เพื่อให้เข้าใจง่ายขึ้น ยกตัวอย่างให้มีรถคันหนึ่งจอดอยู่ที่จุดออริจิน และหันหน้าไปทางแกน y ด้านลบ หรือ หันหน้าไปข้างล่าง เพราะฉะนั้นทิศทางของรถ คือ [-1,0] สมมุติรถคันนั้นเลี้ยวขวา รถก็จะหันหน้าขึ้นข้างบนทิศทางของรถก็จะเปลี่ยนเป็น [1,0]



รูปที่ 4-2 ระบบโคออดิเนตของวัตถุเทียบกับโลก (World Space)

4.1.2 Object Space เป็นระบบโคออดิเนตของออบเจ็คนั้นเมื่อรถเคลื่อนที่ไปข้างหน้า แสดงว่ามันมีทิศทางในแกน x เสมอ การที่จะทำเช่นนี้จึงต้องมีแกนอีกแกนที่เป็นอิสระจากแกนอื่นๆ กำหนดให้แกน

นั้นเป็นแกนเฉพาะของรถเท่านั้น กำหนดให้แกน x ด้านบวกชี้ออกทางด้านหลังของรถ และแกน y ชี้ออกจากทางซ้ายของรถดังรูปที่ 4-3 และทิศทางทีรถหันหน้าไป คือ [-1,0] เสมอ



รูปที่ 4-3 Object Space

ตามรูปที่ 4-2 จะพบว่าในระบบโคออดิเนตของวัตถุรถ รถจะเคลื่อนที่ไปข้างหน้าตามทิศทาง [-1,0] แต่ในโคออดิเนตของโลก รถมีทิศทาง [1,1]

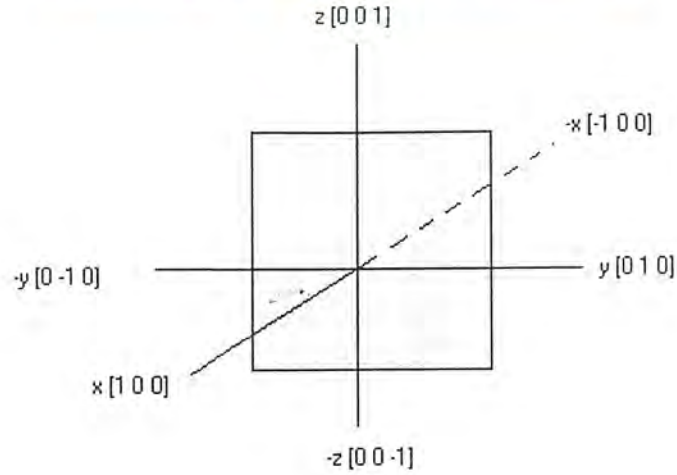
ระบบโคออดิเนตของโลกแกน x และ y จะแทนถึง เวคเตอร์ [1,0] และ [0,1] ตามลำดับ และแกนของวัตถุก็จะเคลื่อนตามวัตถุไปเสมอ จากรูปที่ 4-1 แกน x-y ของรถจะเท่ากับ เวคเตอร์โคออดิเนตของโลก [-1,-1] และ [-1,1] ตามลำดับ

ในเอนจิ้นมอร์ฟีทนั้นจะสามารถเปลี่ยนเวคเตอร์ระหว่าง 2 ระบบนี้ได้ทำให้ผู้โปรแกรมสามารถเลือกใช้ระบบโคออดิเนตที่เหมาะสมต่อสถานะการณ์ได้

4.2 ระบบโคออดิเนต 3 มิติ

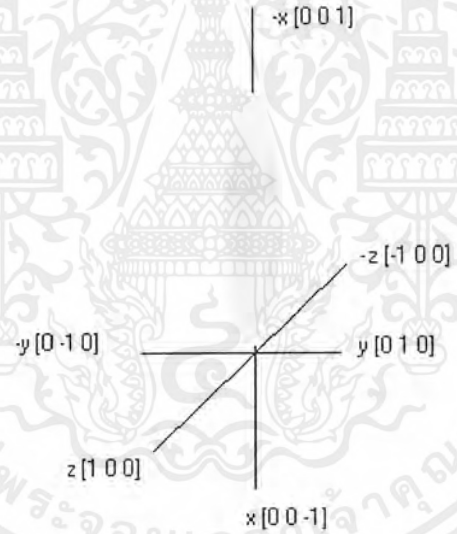
ในระบบ โคออดิเนต 3 มิติ จะต้องตั้งอยู่บนกฎมือซ้าย หรือ มือขวาอย่างใดอย่างหนึ่ง ใน DirectX จะใช้ระบบโคออดิเนตตามกฎมือซ้าย แต่ใน มอร์ฟีท และในโปรแกรมสร้างภาพ 3 มิติ โดยทั่วไปจะยึดระบบโคออดิเนต 3 มิติ ตามกฎมือขวา

ตามกฎมือขวาถ้าสมมุติให้จุดกลางจอมอนิเตอร์เป็นจุดออริจิน จุดที่อยู่บนแกน x ด้านบวก คือ จุดที่อยู่หลังหน้าจอมอนิเตอร์ จุดที่อยู่บนแกน x ด้านลบ คือ จุดที่ชี้ออกมาด้านหน้าแกน y ด้านบวกคือจุดที่ชี้ไปทางขวาของหน้าจอ ในแกนลบจะชี้ไปทางซ้าย ส่วนแกน z ด้านบวกคือจุดที่อยู่ด้านบนของจอ ในด้านลบก็จะอยู่ด้านใต้ของจอ



รูปที่ 4-4 ระบบโคออดิเนต 3 มิติตามกฎมือขวา

ถ้าเป็นในระบบโคออดิเนตนี้ รถที่วิ่งเข้าไปในหน้าจอจะมีทิศ [-1,0,0] และ รถที่วิ่งไปทางขวาจะมีทิศ [0,1,0] เฮลิคอปเตอร์ที่บินขึ้นจากพื้นไปด้านบนของจอก็จะมีทิศ [0,0,1] นี่เป็นการอ้างอิงทิศทางเทียบกับระบบโค ออดิเนตของโลก



รูปที่ 4-5 ระบบโคออดิเนต 3 มิติ ตามกฎมือขวาเทียบกับวัตถุ

ถ้าเปรียบเทียบกับระบบโคออดิเนตของโลกแกน x จะชี้ออกมาจากด้านหลัง แกน y จะชี้ไปทางขวา แกน z จะชี้ขึ้นข้างบน ยกตัวอย่าง เช่น จรวดที่ยิงขึ้นสู่ท้องฟ้าจะมีค่าแกน x, y และ z เป็น [0 0 –1], [0 1 0] และ [1 0 0] ตามลำดับ

4.2.1 การบวกเวกเตอร์ ให้โคออดิเนตของวัตถุด้านหน้าห่างออกไป n ในระบบโคออดิเนตของวัตถุ จุดๆ นั้น จะมีค่าเป็น [-n,0,0] และให้วัตถุด้านหน้ามีทิศทางเดียวกัน คือ มีทิศทาง [-1. 0, 0] ตำแหน่งในระบบโคออดิเนตของโลกของวัตถุที่อยู่ข้างหน้าจะเป็น [current_x_location – n, current_y_location,

current_z_location] ส่วนทิศทางก็จะเป็น [dir_x, dir_y, dir_z]/ $(x^2+y^2+x^2)^{1/2}$ ถ้า [dir_x, dir_y, dir_z] เป็นทิศทางของวัตถุ สาเหตุที่ต้องหารด้วย $(x^2+y^2+x^2)^{1/2}$ เพราะค่านี้เป็นค่า magnitude ของเวคเตอร์ การหารด้วยค่านี้จะทำให้ค่า $(dirm\_x^2+dirm\_y^2+dirm\_x^2)^{1/2}= 1$ (ค่า dirm_x คือค่า dir_X ที่ถูกหารด้วยค่า magnitude แล้ว เพราะฉะนั้น

location_in_front = current_location + *n* * unit_direction

แต่ถ้าจะคำนวนตำแหน่งถัดไปของวัตถุโดยมีตำแหน่งปัจจุบันกับความเร็วของวัตถุ

new_location = current_location + speed * direction

<u>4.2.2 การลบเวคเตอร์</u> การเลื่อนวัตถุไปข้างหน้าเป็นตัวอย่างของการเพิ่มเวคเตอร์ (การบวกผลคูณของเวคเตอร์ทิศทางกับตำแหน่งปัจจุบัน)

แต่ในการหาเวคเตอร์ระหว่าง 2 วัตถุ ต้องใช้การลบเคเตอร์ สมมุติให้กล้องอยู่ที่ cam_pos = [cx, cy, cx] และ วัตถุอยู่ที่จุด obj_pos = [x, y, z] ถ้าต้องการหาเวคเตอร์ที่กล้องชี้ลงมาที่วัตถุ เพื่อที่จะแสดงผลวัตถุนั้น ทำได้โดยเอาตำแหน่งวัตถุตั้งแล้วลบด้วยตำแหน่งวัตถุ

direction = obj_pos – cam_pos

หรือถ้านำไปเขียนแยกทีละแกนได้ดังนี้

dir_x = x – cx

dir_y = y – cy

dir_z = z – cz

ค่าระยะห่างระหว่างวัตถุกับกล้องก็คือค่า magnitude ของเวคเตอร์ หรือ $(dir\_x^2+dir\_y^2+dir\_x^2)^{1/2}$ นั่นเอง

<u>4.2.3 ระนาบ</u> เรื่องนี้เป็นเรื่องสุดท้ายเกี่ยวกับระบบโคออดิเนต 3 มิติ ที่จะกล่าวถึง ระนาบ 3 มิติ จะแทนด้วยสมการ

Ax + By + Cz + D = 0

ถ้าเป็นระนาบที่ทุกๆ จุดบนระนาบมีค่า x = -4 ก็จะสามารถเขียนได้เป็น

x + 0y + 0z + 4 = 0 หรือ

x + 4 = 0

ในระนาบโดยทั่วไปจะถูกอ้างถึงโดยค่าเวคเตอร์ที่ตั้งฉากกับระนาบนั้นๆ โดยนิยามให้ระนาบเกิดจากเส้นตรงจำนวนอนันต์มารวมกัน ยกตัวอย่างเช่นในระนาบ xy ที่แกน z มีค่าเป็น 0 ก็จะมีแกน z

เป็นเวคเตอร์ตั้งฉากนั้นเอง ([0, 0, 1] หรือ [0, 0, -1]) โดยทั่วไปจะหาค่าสมการระนาบได้โดยแทนค่าเวคเตอร์ตั้งฉาก [A, B, C] ในสมการ Ax + By + Cz + D = 0

4.3 แฮนเดิล (HANDLES)

ในการแยกแยะวัตถุทุกอย่างในมอร์ฟิทเอพีไอจำเป็นจะต้องรู้แฮนเดิลของวัตถุนั้นๆ กล้องทุกๆ ตัว และ วัตถุทั้งหมดจะต้องมีแฮนเดิลเป็นของตัวเอง ไม่ว่าจะเป็นบิตแมป (Bitmaps) หรือ โพลีกอนก็ตาม

แฮนเดิลในมอร์ฟิทเป็นค่า DWORD (หรือ unsigned int) นั่นก็หมายความว่าถ้าส่งค่า unsigned int แทนแฮนเดิลของวัตถุผ่านฟังก์ชันไป คอมไพเลอร์ก็จะทำงานตามปกติเพราะไม่รู้ข้อแตกต่าง แต่ในช่วงรันไทม์ มอร์ฟิท จะ pop-up ข้อความขึ้นมาแสดงข้อผิดพลาด และ ถ้าค่าแฮนเดิลเป็น NULL หรือแฮนเดิลของวัตถุถูกทำลาย มอร์ฟิท จะ pop-up ข้อความขึ้นมาแสดงข้อผิดพลาดเช่นกัน

ถ้าสงสัยว่าค่าแฮนเดิลไม่ถูกต้องสามารถตรวจสอบได้โดยใช้ฟังก์ชัน "is" เช่น Morfit_camera_is_camera(), Morfit_object_is_object(), ฯลฯ

ส่วนการอ่านค่าแฮนเดิลของวัตถุทำได้โดยใช้ฟังก์ชัน เช่น แฮนเดิลของกล้อง Morfit_camera_get_default_camera() หรือ ในกรณีที่จะอ่านค่าแฮนเดิลของกล้องที่มีชื่อตามที่เราต้องการก็ให้ใช้ฟังก์ชัน Morfit_camera_get_using_name("my_camera") ในการเขียนโปรแกรมควรเรียกฟังก์ชันอ่านค่า Handle เพียงครั้งเดียวก็พอ ไม่ควรเรียกทุกครั้งก่อนที่จะทำอะไรกับวัตถุนั้น

4.4 ภาพรวมของเกมเอนจิ้นมอร์ฟิท

มอร์ฟิทเอพีไอเป็นกลุ่มของฟังก์ชัน โดยในแต่ละกลุ่มของฟังก์ชันจะมีหน้าที่ต่างๆ กันในการกระทำกับโลก 3 มิติ กลุ่มที่สำคัญ ได้แก่ camera API, object API, engine API, polygon API, and the bitmap API

โดยฟังก์ชันทุกๆ ฟังก์ชันของมอร์ฟิทจะนำหน้าด้วยคำว่า Morfit เช่น Morfit_camera_get_location(), Morfit_engine_load_world() และ Morfit_object_set_direction().

และในมอร์ฟิทได้เตรียม Help ที่อธิบายหน้าที่ของฟังก์ชันทั้งหมดไว้แล้ว การใช้ Help ของ มอร์ฟิท ทำได้โดยทำเครื่องหมายที่ "Use Extension Help" ใน Help menu ของคอมไพเลอร์

และในเฮดเดอร์ไฟล์ mrft_api.h ของ Mofit ก็มีคำอธิบายหน้าที่ของฟังก์ชันเช่นกัน

ในส่วนต่อไปจะอธิบายถึงหน้าที่ของเอพีไอสำคัญๆ ของ มอร์ฟีท โดยในรายละเอียดนั้นจะไม่กล่าวถึงในรายงานฉบับนี้

Object API

ทำหน้าที่ควบคุมวัตถุในโลก 3 มิติที่สร้างขึ้น ทุกๆ วัตถุไม่ว่าจะเป็น คน, สิ่งก็สร้าง, พาหนะ, กำแพง หรือ ต้นไม้ โดยแต่ละวัตถุจะถูกสร้าง, ทำลาย, หมุน หรือ เคลื่อนที่โดยใช้เอพีไอนี้ นอกจากนั้นเอพีไอนี้ยังมีการเตรียมฟังก์ชันสำหรับตรวจสอบการชน ว่าจะสามารถเคลื่อนวัตถุไปได้หรือไม่ และ ยังสามารถกำหนดรายละเอียดทางฟิสิกส์ให้วัตถุแต่ละตัวได้ด้วย เช่น ความยืดหยุ่น, ความเสียดทาน และแรง เป็นต้น โดยแต่ละวัตถุต้องอยู่ภายใต้กฎฟิสิกส์ และ ยังมีฟังก์ชันที่ทำหน้าที่เปลี่ยนรูปร่างวัตถุเพื่อนำมาทำอนิเมชั่นด้วย

Camera API

ทำหน้าที่ควบคุมกล้องในโลกที่สร้างขึ้นมา ไม่ว่าจะเป็นการย้าย, หมุน, ขยายภาพ, ย่อภาพ หรือโฟกัสไปที่วัตถุใด โดยกล้องแต่ละตัวสามารถควบคุมโดยโปรแกรมอย่างอิสระ หรือ ควบคุมให้เคลื่อนไปตาม Track หรือ Chase ก็ได้

Group API

เป็นเอพีไอที่อนุญาติให้รวมโพลีกอนต่างเข้าไว้เป็นกลุ่มเดียวกัน ทำให้สามารถหมุน, เลื่อน, ย้าย หรือ ลดขนาดของกลุ่มของวัตถุนั้นพร้อมๆ กันได้ เราสามารถอ่านขนาดของกลุ่มของวัตถุได้ โดยใน มอร์ฟีท จะเรียกว่า "Bounding Box" และประกอบด้วยฟังก์ชันอื่น เช่น ฟังก์ชันที่ทำหน้าที่กำหนดจุดกึ่งกลางของกลุ่ม (เพื่อนำไปใช้หมุน และ ขยายวัตถุเทียบกับจุดกึ่งกลางนี้) และสามารถกำหนดลายละเอียดทางฟิสิกส์ให้กลุ่มของวัตถุนี้เหมือน Object API

Engine API

เป็นเอพีไอที่ประกอบด้วยฟังก์ชันต่างๆ ที่จัดการกับปรากฏการณ์ทั้งหมดในโลกที่สร้างขึ้น และการแสดงผลออกทางหน้าจอ ไม่ว่าจะเป็นฟังก์ชันโหลดโลก 3 มิติที่สร้างขึ้น หลายฟังก์ชันทำหน้าที่เพิ่มประสิทธิภาพของเกม หรือ เพิ่มคุณภาพของภาพ ตัวอย่างฟังก์ชันก็เช่น ฟังก์ชันที่ทำหน้าที่เรียกใช้ Z-buffer, ฟังก์ชันที่เปลี่ยนความละเอียดที่จะแสดงผล หรือ ฟังก์ชันที่ใช้จัดการปรากฏการณ์ของบรรยากาศในโลกที่สร้างขึ้น

Bitmap API

เป็นเอพีไอที่ทำหน้าที่โหลด หรือ เปลี่ยนบิตแมปที่ในเกม บิตแมปที่ว่าเป็นบิตแมปที่ทำหน้าที่ให้สี และ เท็กซ์เจอร์กับวัตถุ

Animation API

เป็นเอพีไอที่ทำหน้าที่แสดงอนิเมชั่นของชุดของโพลีกอน ที่เปลี่ยนไปตามเวลาต่างๆ อนิเมชั่นสามารถใช้สร้างเอฟเฟคต่างๆ เช่น หมอก, แสดงวัตถุคนวิ่งในโลก หรือ วัตถุที่ได้รับความเสียหาย

Polygon API

วัตถุทุกๆ วัตถุประกอบขึ้นมาจากโพลีกอน สำหรับเอพีไอจะทำหน้าที่ควบคุมรูปร่าง, สี และบิตแมปของโพลีกอน เราสามารถตั้งค่าคุณสมบัติเกี่ยวกับแสง ความใส หรือ ความสว่าง เราสามารถใช้ฟังก์ชันนี้นิยามโพลีกอนเพิ่มจุดเข้าไป ในนั้น หมุน, ย้าย หรือ ขยายโพลีกอนนั้น หรือแม้กระทั่งหาจุดตัดในโพลีกอนหาเส้นตรงที่โพลีกอนตัดกัน และยังสามารถนำไปใช้แสดงผลแสงและเงาให้กับวัตถุได้ด้วย

เอพีไออื่นๆ

เช่น Point API, Background API, Track API และ 3D Card API เป็น API ที่มีความสำคัญรองลงมาแต่ก็เป็น API ที่ยังมีประโยชน์อยู่มาก

บทที่ 5

Object

5.1 Viewer Mode และ Dynamic Objects

ในการที่จะแสดง world ที่เราสร้างขึ้น เราสามารถกำหนดรูปแบบการแสดงผลโดยใช้คำสั่งดังนี้

```
Morfit_engine_load_world(world_file_name, world_directory_path, bitmaps_directory_path,
world_mode);
```

ซึ่งรูปแบบของการแสดงผลจะขึ้นอยู่กับค่า world_mode โดยจะเป็น

- Viewer Mode เมื่อเรากำหนดค่านี้เป็น USER_DEFINE_BEHAVIOR
- Editor Mode เมื่อเรากำหนดค่านี้เป็น EDITOR_MODE

โดยข้อแตกต่างของทั้ง 2 รูปแบบเป็นดังนี้

Viewer Mode	Editor Mode
ประสิทธิภาพ ใช้เวลานานมากในการ load worldเพราะจะต้อง load และคำนวณจนครบทั้ง world แต่หลังจากนั้นจะไม่ต้องคำนวณอีก จึงทำงานได้เร็วมาก ในการใช้ software render แต่ในการใช้ hardware จะต่างกันน้อยมาก	ไม่จำเป็นต้องสร้างและคำนวณ world ทั้งหมดในตอนแรก ดังนั้นจึง load ได้เร็วกว่ามาก แต่ในการทำงานหลังจากนั้นจำเป็นต้องคำนวณ จึงทำงานได้ช้ากว่ามากในการใช้ software render แต่ในการใช้ hardware จะต่างกันน้อยมาก
การควบคุม เนื่องจากโครงสร้างของ world ถูกกำหนดไว้ตายตัว object ต่าง ๆ จึงเป็น Static Object ไม่สามารถเคลื่อนย้ายหรือควบคุมได้ เราจึงต้องใช้ Dynamic Object ซึ่งจะเก็บแยกออกไปต่างหาก และเราสามารถควบคุมได้อย่างอิสระโดยใช้ Object API (**เราทำให้วัตถุเป็น Dynamic Object ได้โดยการตั้งคลิ้กไปที่วัตถุแล้วตั้งค่า "Properties of Object" ให้เป็น "dynamic"**)	เราสามารถควบคุม object ได้ทั้งหมดไม่ว่าจะเป็น static หรือ dynamic เพราะเราไม่ได้สร้าง world ที่ตายตัวไว้ในตอนแรก โดยในการควบคุมนั้นเราจะใช้ Group API

5.2 การตั้งชื่อและการเข้าใช้วัตถุ

มีขั้นตอนดังนี้

1. ใน World Builder เราสามารถตั้งชื่อวัตถุได้ใน dialog box "Properties of Object"
2. จากนั้นเราสามารถเข้าถึงวัตถุได้โดยอ้าง handle ซึ่งสร้างจากการกำหนดค่าดังต่อไปนี้

```
DWORD Morfit_object_get_object_using_name(char *object_name);
```

นอกจากนี้ยังมีรูปแบบการใช้งานที่น่าสนใจเกี่ยวกับการเข้าใช้วัตถุอีกหลายรูปแบบเช่น

- **การท่องไปใน world ที่เราสร้างขึ้น** ซึ่งผลลัพธ์ที่ได้จะเป็นค่า handle ของทุกวัตถุ เราอาจจะเขียนโปรแกรมเพื่อการนี้ได้ดังต่อไปนี้

```
for(handle=Morfit_object_get_first_object() ; handle!=NULL ; handle =
Morfit_object_get_next_object(handle) )
        {
        ...
        }
```

โดย DWORD Mofit_object_get_first_object(void) จะส่งค่า handle ของวัตถุแรกใน world กลับมาส่วน DWORD Morfit_object_get_next_object(DWORD object_handle) จะรับค่า handle ของวัตถุหนึ่งแล้วส่งค่า handle ของวัตถุถัดไปใน world กลับมา

- **การตรวจสอบสถานะว่า handle ยังถูกต้องอยู่หรือไม่** (ตรวจสอบว่าวัตถุนั้น ๆ ไม่ได้ถูกลบหรือ handle ไม่ได้ชี้ไปยังที่อื่น) จะได้ผลลัพธ์เป็น YES ถ้าถูกต้อง หรือเป็น NO ถ้าไม่ถูกต้องซึ่งเราตรวจสอบด้วยการเรียกใช้

```
int Morfit_object_is_object(DWORD object_handle, char *function_asking);
```

- **การเรียกดูชื่อของวัตถุ** ทำได้โดยการเรียกใช้

```
function char * Morfit_object_get_name(DWORD object_handle)
```

- **การเปลี่ยนชื่อของวัตถุ** ทำได้โดยการเรียกใช้

```
Morfit_object_set_name(DWORD object_handle, char * name);
```

5.3 การเคลื่อนย้ายวัตถุ

เราสามารถทำการเคลื่อนย้ายวัตถุได้ 4 รูปแบบได้แก่

1. **การควบคุมโดยตรง** โดยการกำหนดตำแหน่งและทิศทางของวัตถุ
2. **การกำหนดให้วัตถุเคลื่อนที่ไปตามเส้นทาง (Track)**
3. **การกำหนดให้วัตถุเคลื่อนที่ตามวัตถุอื่นไป**
4. **การกำหนดให้วัตถุเคลื่อนที่ตามกฎฟิสิกส์**

5.3.1 การควบคุมโดยตรง

สามารถใช้ควบคุมวัตถุได้อย่างอิสระมากที่สุด มีการทำงานที่สำคัญ ๆ ได้แก่

- **การกำหนดตำแหน่งให้วัตถุ** คือการทำการวางวัตถุลงบนตำแหน่งตามค่าในแกน [x,y,z] ดังนี้

```
Morfit_object_set_location(DWORD object_handle, double x, double y, double z);
```

- **การกำหนดทิศทางให้วัตถุ** จะใช้คำสั่งดังนี้

```
Morfit_object_set_direction(DWORD object_handle, double x, double y, double z);
```

โดย [x,y,z] จะเป็นเวกเตอร์ที่ชี้ไปยังทิศที่เราต้องการให้วัตถุหันหน้าไป

- **การหาตำแหน่งของวัตถุในขณะนั้น** จะได้ผลลัพธ์เป็นตำแหน่งปัจจุบันมาเก็บไว้ในตัวแปร x, y และ z ดังนี้

```
Morfit_object_get_location(DWORD object_handle, double *x, double *y, double *z)
```

ตัวอย่าง ถ้าเราต้องการเคลื่อนวัตถุขึ้นด้านบนตรง ๆ (ตามแกน z) เราอาจเขียนโปรแกรมได้ดังนี้

```
double x,y,z;
Morfit_object_get_location(my_object,&x,&y,&z);
z += 10;
Morfit_object_set_location(my_object,x,y,z);
```

- **การเคลื่อนที่โดยอ้างอิงตำแหน่งสัมพัทธ์** คือการเคลื่อนวัตถุโดยอ้างจากตำแหน่งปัจจุบันของวัตถุ เช่น

```
Morfit_object_move(my_object,WORLD_SPACE,0,0,10);
```

จุดสำคัญคือการเลือกใช้ระบบ coordinate โดย

1) world space จะชี้อิงกับ world เป็นหลักเช่น [-1 0 0] จะทำการเลื่อนวัตถุตามทิศทางเข้าหาหน้าจอตรง ๆ (ตามแกน x ลบ)
2) object space จะชี้อิงกับวัตถุนั้น ๆ เป็นหลักเช่น [-1 0 0] จะทำการลื่อนวัตถุตามทิศทางด้านหน้าของวัตถุไม่ว่าวัตถุจะหันหน้าไปทางใดอยู่ก็ตาม

- **การหมุนวัตถุ** จะทำการหมุนวัตถุไปตามแกนที่เราต้องการ ดังนี้

```
Morfit_object_rotate_x(DWORD object_handle, double degrees, int space_flag);
```

โดย degrees คือจำนวนองศาที่ต้องการหมุนไป และเลือกค่าของ space_flag จาก WORLD_SPACE หรือ OBJECT_SPACE

5.3.2 การกำหนดให้วัตถุเคลื่อนที่ไปตามเส้นทาง (Track)

ในบางครั้งถ้าเราไม่ต้องการกำหนดการเคลื่อนไหวของวัตถุด้วยตนเอง เราอาจกำหนดให้วัตถุเคลื่อนไปตามสิ่งอื่น ๆ ได้ โดยการกำหนดให้เคลื่อนที่ไปตามเส้นทางก็เป็นวิธีหนึ่ง จะมีรูปแบบและขั้นตอนการทำงานที่สำคัญ ๆ ได้แก่

1. เมื่อเรากำหนดให้วัตถุเคลื่อนที่ไปตามเส้นทางแล้วมอร์ฟิทจะทำการเลื่อนวัตถุไปตามเส้นทางทุกครั้งที่เราเรียกใช้ Morfit_engine_render()
2. เราต้องสร้าง track ขึ้นมาก่อน วิธีที่ง่ายที่สุดคือใช้ World Builder (ดูรายละเอียดในบทที่ 1)
3. วัตถุที่จะใช้ต้องกำหนดให้เป็น dynamic object
4. กำหนดให้วัตถุเคลื่อนที่ สามารถทำได้ 2 วิธี คือ

- ใช้ dialog box ใน World Builder ซึ่งเป็นวิธีที่ง่ายที่สุด โดยกำหนดชื่อของ track ที่จะให้วัตถุเคลื่อนที่ไปตาม และกำหนดรูปแบบการเคลื่อนที่ตาม
- ใช้การเขียนโปรแกรมกำหนดด้วยตนเอง โดยเราจะต้องมี handle ของวัตถุก่อน (ดู 5.2 ประกอบ) 1. ส่วนการสร้าง handle ของ track ทำได้ดังนี้ (สมมุติว่าเราตั้งชื่อ track ว่า mytrack และ วัตถุชื่อ man)

```
DWORD track=Morfit_track_get_using_name("mytrack");
```

เราจะได้ handle ของ mytrack ชื่อว่า track และสมมุติให้ handle ของวัตถุชื่อว่า man

จากนั้นทำการกำหนดรูปแบบการเคลื่อนที่ตามได้โดย

```
Morfit_set_chase_type(object_handle, chase_type)
```

ซึ่ง chase_type จะเป็นตัวระบุรูปแบบการเคลื่อนที่ตาม ที่เราสนใจในที่นี้ได้แก่

- CHASE_TRACK จะทำการเคลื่อนที่ตามแบบที่เน้นความแม่นยำ และเลี้ยวแบบหักตรง
- CHASE_TRACK_AIRPLANE จะทำการเลี้ยวแบบค่อย ๆ เลี้ยวหักมุม คล้ายกับเครื่องบิน
- แบบอื่น ๆ ดูรายละเอียดใน 5.3.3

จากนั้นเราก็กำหนดว่าจะให้วัตถุใด เคลื่อนไปตาม track ไหน (เพราะอาจจะมีหลาย track ใน world) โดย

```
Morfit_object_set_track(object_handle,track_handle);
```

เมื่อเรากำหนด track ที่ต้องการแล้วเราก็ทำการกำหนดความเร็วให้กับวัตถุดังนี้

```
Morfit_object_set_absolute_speed(man,20);
```

ตัวอย่าง จากขั้นตอนการทำงานข้างต้นเราสามารถนำมาเขียนเป็นโปรแกรมได้ดังโปรแกรมที่ 5-3 โดยมีวัตถุชื่อ man (ซึ่งมี handle ชื่อ man) และมี track ชื่อ mytrack (ซึ่งมี handle ชื่อ track)

```
#include "..\\..\\..\\Engine\\Include\\mrft_api.h"
#include <iostream.h>
```

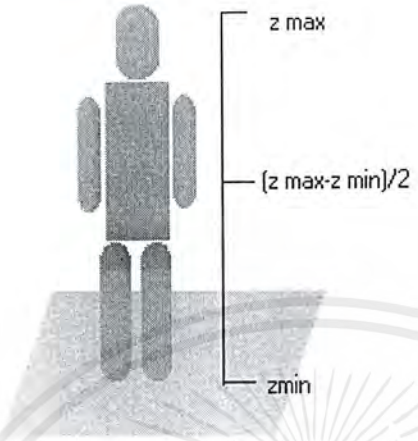
```
void main(void)
{
        void Move_man_on_track(DWORD man,DWORD track);

        int rc=Morfit_engine_load_world("track.wld",".","Bitmaps",USER_DEFINED_BEHAVIOR);
        if(rc==VR_ERROR) {
                cout << "Failed to load world, aborting\n";
                cout << "look at error.log to see why\n";
                return;
        }


        Morfit_engine_set_default_rendering_window_title("esc to exit");
        Morfit_engine_maximize_default_rendering_window();
        ShowCursor(FALSE);  // Hide the cursor

        DWORD camera=Morfit_camera_get_default_camera();
        DWORD man=Morfit_object_get_object_using_name("man");
        DWORD track=Morfit_track_get_using_name("mytrack");
        Morfit_object_set_chase_type(man, CHASE_TRACK);
        Morfit_object_set_track(man,track);
        Morfit_object_set_absolute_speed(man,20);

        while( (GetAsyncKeyState(VK_ESCAPE)&1) ==0 ) {
                Morfit_engine_render(NULL,camera);
        }

}
```

โปรแกรมที่ 5-3 แสดงตัวอย่างการกำหนดให้วัตถุเคลื่อนที่ไปตาม track

5. ทำการตรวจสอบค่า offset ของวัตถุ จากตัวอย่างโปรแกรมที่ 5-3 จะเห็นว่าเมื่อประมวลผลโปรแกรมแล้ว บางครั้งวัตถุ (ในที่นี้เราเลือกรูปคนมา) จะเดินแบบเท้าทะลุลงไปใต้ดิน ในการแก้ปัญหานี้เราต้องใช้การตรวจสอบค่า offset ของวัตถุนั่นคือเราสามารถสั่งให้มอร์ฟิทำการเลื่อนวัตถุไปตาม track โดยคำนึงถึงค่า offset ซึ่งเป็นค่าความสูงที่วัดจากแนวระดับ

ขึ้นไปยังจุดศูนย์กลางของวัตถุ เช่นถ้าเรากำหนด offset vector เป็น [0 0 150] เราจะได้วัตถุที่เคลื่อนที่ไปตาม track โดยที่จุดศูนย์กลางของวัตถุจะอยู่สูงจากแนวระดับ 150 หน่วยเสมอ

ตัวอย่าง เพื่อความเข้าใจเรื่อง offset เราจะใช้ตัวอย่างที่วัตถุเป็นรูปคน ดังรูปที่ 5-3



รูปที่ 5-3 แสดง offset ของวัตถุรูปคน

จากรูป เราต้องการให้จุดศูนย์กลางของคนเคลื่อนที่ไปตาม track ที่ระยะสูงจากแนวระดับครึ่งหนึ่งของความสูงของคนจากแนวระดับ (ระยะ [z max -z min]/2) ซึ่งจะทำให้เท้าของคนอยู่บนพื้นเสมอ

ในการหาความสูงของคนเราใช้

```
Morfit_object_get_bounding_box(DWORD object_handle, double box[2][3]);
```

ซึ่งจะคำนวณค่าสูงสุดและค่าต่ำสุดของ coordinates x, y และ z และเก็บลงใน *box* array ในรูปแบบต่อไปนี้

box[0][0] เป็นค่า minimum X

box[0][1] เป็นค่า minimum Y

box[0][2] เป็นค่า minimum Z

box[1][0] เป็นค่า max X

box[1][1] เป็นค่า max Y

box[1][2] เป็นค่า max Z

ดังนั้นความสูงของคนจะคำนวณได้ดังนี้

```
Morfit_object_get_bounding_box(man,box);
double height = (box[1][2]-box[0][2]);
```

จากนั้นเราก็กำหนด track offset โดยกำหนด vector ขนาดสูงครึ่งหนึ่ง (ตามแกน z) ของความสูงของคนดังนี้

```
double up[3]={0,0,height/2};
Morfit_object_set_track_offset(man,up);
```

** ถ้าเราใช้ World Builder ในการกำหนดให้วัตถุเคลื่อนที่ตาม track เราไม่ต้องห่วงเรื่อง offset เพราะมอร์ฟิทจะจัดการให้โดยอัตโนมัติ

6. เมื่อเราต้องการหยุดวัตถุไม่ให้เคลื่อนที่ไปตาม track ทำได้โดย

```
Morfit_object_set_chase_type(man, NO_CHASE);
```

ทั้งหมดที่กล่าวมาแสดงให้เห็นว่าการใช้ track เป็นเรื่องง่าย เพียงแค่เรากำหนดให้วัตถุเคลื่อนไปตาม track, กำหนดความเร็ว และ offset นอกนั้นมอร์ฟิทจะจัดการให้เราเอง

5.3.3 การกำหนดให้เคลื่อนที่ไปตามวัตถุ

เราสามารถกำหนดให้ dynamic object ใด ๆ เคลื่อนที่ไปตาม dynamic object ใด ๆ ก็ได้ โดยจะมีปัจจัยสำคัญ ๆ 4 ประการ ได้แก่

- วัตถุที่กำหนดให้ถูกเคลื่อนที่ตาม (Chased Object)
- ระยะการเคลื่อนที่ตาม (Chase Offset)
- รูปแบบการเคลื่อนที่ตาม (Chase Type)
- ความผ่อนปรนในการเคลื่อนที่ตาม (Chase Softness)

1. **วัตถุที่กำหนดให้ถูกเคลื่อนที่ตาม (Chased Object)**

   วัตถุที่กำหนดให้ถูกเคลื่อนที่ตามจะเป็น dynamic object, camera, หรือกรุ๊ปก็ได้ การทำงานที่สำคัญ ๆ ได้แก่

- การกำหนดให้วัตถุเคลื่อนที่ตามสิ่งต่าง ๆ ที่กล่าวข้างต้น ทำได้โดย

```
Morfit_object_set_object_to_chase(DWORD object_handle, DWORD object_to_chase_handle);
Morfit_object_set_camera_to_chase(DWORD object_handle, DWORD camera_to_chase_handle);
Morfit_object_set_group_to_chase(DWORD object_handle, DWORD group_to_chase_handle);
```

- การตรวจสอบว่าวัตถุที่ถูกเคลื่อนที่ตามคืออะไร ทำได้โดย

```
DWORD Morfit_object_get_chased_object(DWORD object_handle);
DWORD Morfit_object_get_chased_camera(DWORD object_handle);
DWORD Morfit_object_get_chased_group(DWORD object_handle);
```

** **วัตถุสามารถเคลื่อนที่ตามสิ่งอื่น ๆ ได้คราวละ 1 อย่างเท่านั้น** เช่น ถ้าวัตถุเคลื่อนที่ตามกรุ๊ปหนึ่งอยู่ ถ้าเราเรียกใช้ Morfit_object_set_camera_to_chase() วัตถุนั้นจะหยุดเคลื่อนที่ตามกรุ๊ปและหันไปเคลื่อนที่ตาม camera แทน

- การหยุดไม่ให้วัตถุเคลื่อนที่ตามวัตถุอื่น ทำได้โดย

```
Morfit_object_set_chase_type(object_handle, NO_CHASE);
```

เรายังสามารถกำหนด chased object ได้จาก World Builder เพียงแค่เลือก object's properties, กำหนดให้เป็น dynamic object, และเลือกวัตถุที่จะถูกเคลื่อนที่ตามจากรายการที่ปรากฏ

2. **ระยะการเคลื่อนที่ตาม (Chase Offset)**
   เป็นตัวกำหนดว่าวัตถุที่กำลังเคลื่อนที่ตามอยู่ห่างจากวัตถุที่ถูกเคลื่อนตามเท่าไร เช่นถ้า chase offset เป็น [0, 0, 20] แล้ววัตถุที่เคลื่อนที่ตามจะเคลื่อนที่ไปพร้อมกับวัตถุที่ถูกตาม แต่จะอยู่เหนือวัตถุที่ถูกตามอยู่ 20 หน่วย หรือถ้า chase offset เป็น [10, 10, 0] แล้วค่า coordinates x และ y ของวัตถุที่กำลังเคลื่อนที่ตามจะมากกว่าของวัตถุที่ถูกตามอยู่ 10 เสมอ

   ** **chase offset ไม่ขึ้นอยู่กับความเร็วของวัตถุที่กำลังเคลื่อนที่ตาม** ระยะที่กำหนดจะเท่าเดิมเสมอ

- การกำหนด object's chase offset ทำได้โดย

```
Morfit_object_set_chase_offset(DWORD object_handle, double offset[3]);
```

- การตรวจสอบ chase offset ปัจจุบัน ทำได้โดย

```
Morfit_object_get_chase_offset(DWORD object_handle, double offset[3]);
```

** **ใน World Builder ไม่จำเป็นต้องกำหนด chase offset** มอร์ฟิทจะทำให้เอง

- การกำหนด object's chase distance จะเป็นตัวกำหนดระยะห่างว่าห่างจากวัตถุที่ถูกตามเท่าไร มีประโยชน์มากกว่า chase offset โดยจะสังเกตได้ในขณะที่วัตถุที่ถูกตามกำลังเลี้ยว ถ้าเราใช้ chase distance มอร์ฟิทจะคำนวณตำแหน่งของวัตถุที่กำลังตามให้ใหม่ เพื่อให้วัตถุที่ตามอยู่ในตำแหน่งเดิมตามหลังวัตถุที่ถูกตามเสมอ เราสามารถเรียกใช้ chase distance ได้ในลักษณะเดียวกันกับ chase offset ดังนี้

```
Morfit_object_set_chase_distance(DWORD object_handle, double chase_distance);
double Morfit_object_get_chase_distance(DWORD object_handle);
```

3. **รูปแบบการเคลื่อนที่ตาม (Chase Type)**

มีทั้งหมด 3 รูปแบบได้แก่

- Chase Precise จะทำให้วัตถุตามอยู่ในตำแหน่งที่สัมพันธ์กับตำแหน่งของวัตถุที่ถูกตามเสมอ เช่นถ้ากำหนดให้ เอลิคอปเตอร์เคลื่อนที่ตามรถ โดยอยู่เหนือกว่ารถ(สมมุติ offset = [10, 0, 10]) เมื่อรถเลี้ยว จะทำให้เฮลิคอปเตอร์ตีวงไปตามหลังรถให้ได้ระยะ offset ที่เท่าเดิมทันที แทนที่จะตรงไปเหมือนรถแล้วจึงเลี้ยวตาม และถ้ารถหมุนเฮลิคอปเตอร์จะหมุนตามในลักษณะดาวเทียมโคจรรอบโลกเพื่อรักษาระยะให้ตรงตาม offset ที่เรากำหนด

- Chase Location จะทำงานเหมือน precise แต่เฮลิคอปเตอร์จะไม่หมุนตามรถ เพียงแค่รักษาระยะ offset ให้เท่าเดิมเท่านั้น

- Chase Flexible จะไม่เลี้ยวเพื่อตามวัตถุอย่างแม่นยำแต่จะค่อย ๆ ตามวัตถุไป แม้ว่าจะทำให้ค่า offset คลาดเคลื่อนไปเล็กน้อย

การเลือก chase type ทำได้โดย

```
Morfit_object_set_chase_type(DWORD object_handle, int chase_type);
```

ซึ่ง chase_type จะมีเป็นค่าคงที่ได้ดังต่อไปนี้ CHASE_PRECISE, CHASE_LOCATION, CHASE_FLEXIBLE, และ NO_CHASE (เราใช้ NO_CHASE เมื่อต้องการยกเลิกการเคลื่อนที่ตาม)

การตรวจสอบ chase type ปัจจุบันทำได้โดย

```
int Morfit_object_get_chase_type(DWORD object_handle);
```

4. **ความผ่อนปรนในการเคลื่อนที่ตาม (Chase Softness)**

จะเป็นค่าที่ใช้ในการพิจารณาว่าวัตถุตามจะหักเลี้ยวตามวัตถุนำ โดยเลี้ยวสะบัดแรงเท่าไร ซึ่งปกติจะมีค่าอยู่ระหว่าง 0 กับ 1 แต่ใน World Builder จะเห็นว่า เป็น 0-10 โดยค่าที่ใส่เข้าไปจะถูกหารด้วย 10 เพื่อให้ได้ค่า chase softness จริง

เราสามารถควบคุมการทำงานของ chase softness ได้โดย

```
Morfit_object_set_chase_softness(DWORD object_handle, double softness);
double Morfit_object_get_chase_softness(DWORD object_handle);
```

5.3.4 การกำหนดให้วัตถุเคลื่อนที่ตามกฎฟิสิกส์

เราสามารถกำหนดให้วัตถุเคลื่อนที่ไปตามกฎฟิสิกส์ได้ โดยเราเพียงแค่กำหนดความเร็วต้นกับทิศทาง, กำหนดคุณสมบัติต่าง ๆ ทางฟิสิกส์ จากนั้นมอร์ฟิทจะจัดการส่วนที่เหลือให้เอง การทำงานที่สำคัญ ๆ ได้แก่

1. กำหนดให้ chase type เป็น CHASE_PHYSICS และเมื่อต้องการยกเลิกให้กำหนดเป็น NO_CHASE

2. กำหนดความเร็วต้น ซึ่งจะอยู่ในรูปเวกเตอร์ [x y z] ซึ่งจะแทนค่าของความเร็วในแกนต่าง ๆ เช่น [10 0 10] จะทำให้วัตถุเคลื่อนที่ขึ้นและเข้าหาตัวเรา ซึ่งการกำหนดความเร็วนั้นทำได้ 2 วิธีคือ

- แบบความเร็วและทิศทางรวมกัน ดังนี้

```
double speed[3]={10 0 10};
Morfit_object_set_speed(object_handle,speed);
```

- แบบความเร็วและทิศทางแยกกัน โดยเราสามารถกำหนด magnitude ได้หลายค่า ทำให้เราสามารถกำหนดให้วัตถุหลาย ๆ วัตถุเคลื่อนที่ไปในทิศทางเดียวกัน ที่ความเร็วแตกต่างกันได้ ดังนี้

```
double speed[3]={1 0 1};
Morfit_object_set_speed(object_handle,speed);
Morfit_object_set_absolute_speed(14.14);
```

ค่า magnitude 14.14 มาจาก $\text{sqrt}(x^2 + y^2 + z^2)$ ซึ่งในที่นี้เราได้จากการคำนวณ $\text{sqrt}(10^2 + 0 + 10^2)$

ตัวอย่าง การกำหนดให้หลายวัตถุเคลื่อนที่ในทางเดียวกัน แต่มีความเร็วต่างกัน

```
Double direction[3]={.4, .6, .8};
for (j=0;j<5;j++);
{
Morfit_object_set_speed(object[j],direction);
Morfit_object_set_absolute_speed(object[j],rand());
}
```

** การกำหนดความเร็วต้นแบบสุ่มตามตัวอย่างข้างบนอาจมีข้อจำกัดบางอย่างซึ่งเราต้องควบคุมไว้ก่อน เช่น ถ้าวัตถุวางอยู่บนพื้น เราต้องแน่ใจว่าวัตถุจะไม่เคลื่อนที่ต่ำลงไปกว่าพื้นอีก ดังนั้นเราต้องควบคุมให้ค่า z เป็นบวกเสมอ อย่างเช่น

```
double direction[3]={rand(),rand(),rand()};
```

```
if (direction[2]<0) direction[2]=-direction[2];
Morfit_object_set_speed(object,direction);
Morfit_object_set_absolutee_speed(object,50);
```

3. กำหนดคุณสมบัติต่าง ๆ ทางฟิสิกส์ ได้แก่

- max_speed เป็นการกำหนดความเร็วสูงสุดสัมบูรณ์ ทำได้โดย

```
Morfit_object_set_max_speed(DWORD object_handle, double max_speed);
```

- ความเสียดทาน (Friction) จะมีผลต่อความเร็วของวัตถุดังนี้

0:	ไม่มีผลของ Friction
ระหว่าง 0 กับ 1:	วัตถุจะช้าลงทุก ๆ ครั้งที่ render world
1:	Infinite Friction วัตถุจะหยุดทันที
น้อยกว่า 0:	วัตถุจะมีความเร่งทุก ๆ ครั้งที่ render world

ค่า (1-Friction) จะใช้เป็นตัวคูณกับความเร็วของวัตถุทุก ๆ ครั้งที่มีการเรียกใช้ Morfit_engine_render() ค่าทั่วไปของ Friction จะมีค่าระหว่าง -.03 ถึง .03

เราสามารถกำหนดค่า Friction ได้โดย

```
Morfit_object_set_friction(DWORD object_handle, double friction);
```

- ความยืดหยุ่น (elasticity) จะใช้เป็นตัวคูณกับความเร็วของวัตถุเมื่อมีการชนเกิดขึ้น เช่นถ้าเป็น 1 วัตถุมีความเร็วเท่าเดิม, ถ้าเป็น .5 วัตถุจะช้าลงครึ่งหนึ่ง และถ้าเป็น 2 วัตถุจะเร็วเป็น 2 เท่า เราสามารถกำหนด elasticity ได้ดังนี้

```
Morfit_object_set_elasticity(DWORD object_handle, double friction);
```

- แรง (Force) คือเวกเตอร์ที่ใช้จำลองผลของแรงดึงดูดต่อวัตถุ เช่น [0 0 -1] จะจำลองเหมือนว่าวัตถุถูกกดลง (โดนแรงดึงดูด ดูดลงไปตามแกน z), ถ้าเป็น [0 0 1] วัตถุจะถูกดูดขึ้นตามแกน z, [1 1 0] จะถูกดูดไปที่มุม เป็นต้น เราสามารถกำหนด Force ได้โดย

```
Morfit_object_set_force(DWORD object_handle, double[3] force);
```

5.4 การตรวจสอบการชน (Collision Detection)

ทำได้โดยการเรียกใช้ดังนี้

```
int Morfit_object_is_movement_possible(DWORD object_handle ,double start_location[3], double
end_location[3], DWORD *intersected_polygon, double intersection[3], DWORD *blocking_object);
```

ฟังก์ชันจะคืนค่า YES ถ้าสามารถเคลื่อนที่ไปได้ (movement possible) และจะเป็น NO ถ้าเคลื่อนที่ไปไม่ได้พร้อมกับคืนค่าจุดที่เกิดการชน (จุดที่วัตถุอื่น ๆ ตัดกับเส้นตรงที่เรากับหนดจุดต้นและจุดสิ้นสุดเอาไว้ ซึ่งโดยปกติจะกำหนดให้ลากผ่านวัตถุที่เราต้องการตรวจสอบการชน และอาจกำหนดไว้หลายเส้นเพื่อครอบคลุมจุดหลัก ๆ ของวัตถุ เช่นกำหนดไว้ที่รถ 2-3 เส้น ถ้าต้องการตรวจสอบว่ารถชนกับอะไรบ้าง), ค่า handle ของโพลีกอนที่มาตัดกับเส้น และค่า handle ของวัตถุที่เป็นเจ้าของโพลีกอนนั้น (blocking_object)

**ถ้าฟังก์ชันนี้คืนค่า NO แต่ blocking_object เป็น NULL แสดงว่าวัตถุชนกับส่วนของ world ที่เป็น static

การทำงานอื่น ๆ ที่สำคัญได้แก่

- การกำหนดให้วัตถุสามารถถูกทะลุผ่านได้เช่น เมฆ, ควัน ทำได้โดย

```
Morfit_object_make_non_collisional(DWORD object_handle);
```

ซึ่งเมื่อวัตถุอื่นเรียกใช้ Morfit_object_is_movement_possible() วัตถุนี้จะไม่ถูกตรวจสอบการชน

- ค่าโดยปริยายของทุกวัตถุจะกำหนดไว้ว่าไม่สามารถถูกทะลุผ่านได้ ซึ่งเป็นดังนี้

```
Morfit_object_make_collisional(DWORD object_handle);
```

- การตรวจสอบค่า collisional (ที่ถูกกำหนดไว้ตามข้างต้น) ของแต่ละวัตถุ (คืนค่าเป็น YES หรือ NO) ทำได้โดย

```
int Morfit_object_is_collisional(DWORD object_handle);
```

5.5 การทำงานอื่น ๆ

ยังมีอีกหลาย ๆ ฟังก์ชันที่น่าสนใจใน Object API ในหัวข้อนี้จะยกมาเพียงบางส่วน โดยส่วนที่เหลือสามารถดูรายละเอียดได้จาก file "mrft_api.h"

- Copy วัตถุขึ้นมาอีกวัตถุหนึ่ง ทำได้โดย

```
DWORD Morfit_object_duplicate(DWORD object_handle, int duplicate_polygons_flag);
```

ซึ่ง duplicate_polygon_flag จะเป็นตัวบอกว่าเป็นการ copy ไปทุกโพลีกอนหรือไม่ โดยปกติควรเป็น YES แต่ถ้าเป็น NO เมื่อวัตถุต้นแบบเปลี่ยนบิตแมปไป วัตถุที่ copy ขึ้นมาจะถูกเปลี่ยนบิตแมปด้วย เพราะใช้ polygon เดียวกัน

- Enable หรือ Disable วัตถุ จะใช้เมื่อเราต้องการให้วัตถุหายไปชั่วคราว (disable) โดยสามารถเรียกกลับมาแสดงผลใหม่ได้ (enable) การทำแบบนี้จะใช้เวลาน้อยกว่าลบแล้วสร้างวัตถุใหม่มาก

```
Morfit_object_disable(DWORD object_handle);
```

```
Morfit_object_enable(DWORD object_handle);
int Morfit_object_is_enabled(DWORD object_handle);
```

- Expiration Timer ใช้กำหนดให้มีการลบ หรือ disable วัตถุหลังจากเวลาที่กำหนด มีการเรียกใช้ดังนี้

```
Morfit_object_set_event(DWORD object_handle, int time_in_milliseconds, int event);
```

ซึ่ง event เป็นได้ทั้ง MORFIT_DELETE หรือ MORFIT_DISABLE ถ้าเป็นการลบ (MORFIT_DELETE) เราอาจต้องการลบ handle ของวัตถุด้วยโดยใช้ object_handle=NULL

- การตกจาก track ถ้า world ของเราไม่ได้แบนราบ การกำหนด track ที่ถูกต้องให้วัตถุเคลื่อนที่ไปตาม track จะเป็นเรื่องที่ยากมาก เช่น ถ้า track เป็นรูปสี่เหลี่ยมและมันต้องผ่านเนินเขาหรือยอดเขา บางครั้งวัตถุอาจจะทะลุผ่านเนินเขา หรือลอยอยู่บนอากาศ ในการหลีกเลี่ยงปัญหานี้ เราอาจกำหนดให้ track อยู่เหนือพื้น (หรือกำหนด track บนพื้นแล้วใช้ค่า offset ยกขึ้นมา) แล้วเรียกใช้ฟังก์ชันการตกจาก track (falling from track) เพื่อทำให้ตกลงมาบนพื้นด้านล่าง track โดยตรง การทำแบบนี้จะทำให้วัตถุสามารถเคลื่อนที่ผ่านเนินเขาและยอดเขาได้ โดยมอร์ฟิทจะปรับค่า coordinate z ให้โดยอัตโนมัติเพื่อให้แน่ใจว่าวัตถุจะอยู่บนพื้นเสมอ การเรียกใช้ฟังก์ชันนี้ทำได้โดย

```
Morfit_object_set_falling_from_track(DWORD object_handle, double  height_above_ground, int
fall_through_dynamic_objects);
```

วัตถุจะตกลงมาจนกระทั่งถึงค่า height_above_ground เหนือพื้นดิน เมื่อ fall_trough_dynamic_object
เป็น YES วัตถุจะทะลุผ่าน dynamic object ที่อยู่ด้านล่างของมัน (เช่น เครื่องบิน) แต่ยังคงลงสู่พื้นผิวของ static object (เช่น สิ่งก่อสร้าง หรือพื้น) แต่ถ้าเป็น NO วัตถุจะลงสู่พื้นผิวของวัตถุแรกที่มันผ่าน

ตัวอย่าง การใช้ฟังก์ชันนี้เพื่อให้แน่ใจว่าวัตถุ คน จะวิ่งไปบนพื้นตาม track ที่เรากำหนด

```
Morfit_object_set_chase_type(man, CHASE_TRACK);
Morfit_object_set_track(man,track);
double up[3]={0,0,200};
Morfit_object_set_track_offset(man,up);
Morfit_object_set_falling_from_track(man,0,YES);
```

- การกำหนดชื่อประเภทของวัตถุ (type name) คือข้อความที่เชื่อมโยงอยู่กับวัตถุที่ยังไม่ได้ใช้ในขณะนั้น ใช้ในการเก็บข้อมูลเกี่ยวกับวัตถุตามที่เราต้องการ ฟังก์ชันที่ใช้กำหนดและเรียก type name ของวัตถุมาใช้ เป็นดังนี้

```
int Morfit_object_set_type_name(DWORD object_handle, char *new_name);
char * Morfit_object_get_type_name(DWORD object_handle);
```

- Automatically Advancing Objects ใน Viewer Mode ทุกครั้งที่เราเรียกใช้ Morfit_engine_render() ในการแสดง world วัตถุใด ๆ ที่ถูกกำหนดให้เคลื่อนที่ตาม ไม่ว่าจะเป็น track หรือ physics ก็จะเคลื่อนที่ไปตามค่า parameter ของมันตามปกติ แต่ในบางครั้งเราอาจต้องการที่จะหยุดวัตถุจากการเคลื่อนที่ไปทุก ๆ ครั้งที่ render world ซึ่งทำได้โดย

```
Morfit_engine_advance_objects_automatically(int yes_no_flag);
```

เมื่อเรา disable การเคลื่อนที่โดยอัตโนมัติแล้ว เราจะ enable มันได้โดย

```
Morfit_object_advance(DWORD object_handle);
```

หรือเราอาจจะ enable การเคลื่อนที่โดยอัตโนมัติของทุกวัตถุได้โดย

```
Morfit_object_advance_all();
```

บทที่ 6

กล้อง (Camera)

การแสดงผลโลก 3 มิติ นั้นจะต้องแสดงผ่านกล้องที่มีอยู่ในโลก 3 มิติ ซึ่งในบทนี้เราจะกล่าวถึงวิธีสร้างและควบคุมกล้อง

ภาพที่ถูกแสดงผลออกมาผ่านกล้องจะเป็นเช่นไร ก็ขึ้นอยู่กับปัจจัยหลักๆ ดังนี้คือ ตำแหน่งของกล้อง, ทิศทางที่กล้องชี้ไป, การซูม (Zoom), คุณภาพของภาพ

6.1 การตั้งชื่อ Camera และ Handles

คำสั่งที่ใช้กับกล้องจะเหมือนกับของวัตถุโดยจะต่างกันเล็กน้อย เช่น

```
for(handle=Morfit_object_get_first_object() ; handle!=NULL ; handle =
        Morfit_object_get_next_object(handle) )
        {
        ...
        }
```

นี่เป็นโค้ดของการอ่านค่าแฮนเดิลของกล้อง

```
for(handle=Morfit_camera_get_first_camera() ; handle!=NULL ; handle =
        Morfit_camera_get_next_camera(handle) )
        {
        ...
        }
```

การตรวจสอบชื่อกล้องก็เหมือนกับวัตถุดังตัวอย่าง

```
char * Morfit_object_get_name(DWORD object_handle);
char * Morfit_camera_get_name(DWORD camera_handle);
```

ส่วนนี้เป็นการอ่านค่าแฮนเดิลของกล้องโดยอ้างอิงจากชื่อ

```
DWORD Morfit_object_get_object_using_name(char *object_name);
DWORD Morfit_camera_get_using_name(char * camera_name);
```

นอกจากนี้เรายังสามารถสร้างกล้องเองโดยใช้คำสั่ง

```
DWORD Morfit_camera_create(char *camera_name);
```

แต่กล้องจะถูกสร้างขึ้นตามค่าปริยาย เราต้องมากำหนดตำแหน่งและทิศทางก่อนจะนำไปใช้งาน โดยฟังก์ชันนี้จะคืนค่าแฮนเดิลของกล้องที่สร้างขึ้นมาใหม่ให้ ส่วนการทำลายกล้องที่สร้างขึ้นมาจะใช้คำสั่ง

```
int Morfit_camera_delete(DWORD camera_handle); หรือ
Morfit_camera_delete_all(void);
```

หลังจากคุณทำลายกล้องไปแล้ว จะไม่สามารถเรียกใช้กล้องนั้นๆ ได้อีก

นอกจากนี้ยังมีแฮนเดิลของกล้องที่สำคัญๆ ที่เราควรรู้ คือ

ค่าปริยายของคาเมรา(Default Camera) หมายถึง กล้องที่สร้างขึ้นเป็นตัวแรกในโลกที่สร้างขึ้นโดยใช้ World Builder หรือ ในโปรแกรม เรียกใช้โดยใช้คำสั่ง

```
Morfit_camera_get_default_camera(void);
```

คาเมราปัจจุบัน (Current Camera) หมายถึง กล้องที่ใช้แสดงผลตัวสุดท้าย (ที่ใช้อยู่ขณะนั้นๆ) ฟังก์ชันบางตัวของมอร์ฟีทนั้นจะไม่สามารถใช้กับกล้องที่เรากำหนด แต่จะใช้ได้กับคาเมราปัจจุบันเท่านั้น เช่น Morfit_engine_3D_point_to_2D การใช้คำสั่งพวกนี้เราต้องแน่ใจว่าเรียกใช้กับกล้องที่เราต้องการ ซึ่งสามารถตรวจสอบ หรือ เปลี่ยนคาเมราปัจจุบันได้โดยใช้คำสั่ง

```
int Morfit_camera_set_current(DWORD camera_handle);
DWORD Morfit_camera_get_current(void);
```

6.2 ตำแหน่งของ Camera

ตำแหน่งกล้องในโลก 3 มิติ อ้างอิงโดยใช้โคออดิเนต 3 มิติ [x,y,z] เดียวกับของวัตถุ และสามารถเปลี่ยนแปลงได้เช่นกัน

การกำหนดตำแหน่งกล้องจะใช้คำสั่ง

```
Morfit_camera_set_location(DWORD camera_handle, double x, double y, double z);
```

ส่วนฟังก์ชันนี้ใช้สำหรับหาตำแหน่งกล้องในขณะนั้น

```
Morfit_camera_get_location(DWORD camera_handle, double *x, double *y, double *z);
```

ในการเลื่อนกล้องโดยอ้างอิงจากตำแหน่งปัจจุบัน ใช้

```
Morfit_camera_move(DWORD camera_handle, int space_flag, double x, double y, double z);
```

space_flag จะต้องถูกกำหนดเป็นอย่างใดอย่างหนึ่งระหว่าง WORLD_SPACE และ CAMERA_SPACE ซึ่ง camera space ก็เหมือนกับ object space เช่น เวกเตอร์ของกล้องใน camera space [-1,0,0] จะชี้ไปด้านหน้าของกล้องเสมอ

ส่วนการเลื่อนกล้องโดยอัตโนมัติให้เคลื่อนไปตามทางที่กำหนด หรือ ตามวัตถุใดๆ ก็สามารถทำได้เช่นกัน กลับไปดูตัวอย่างในบทที่ 3 และเพิ่มโค้ดนี้เข้าไปหลังจากโหลดค่าแฮนเดิลของโลกและวัตถุแล้ว

```
Morfit_camera_set_chase_type(camera,CHASE_PRECISE);
Morfit_camera_set_object_to_chase(camera, car);
Morfit_camera_set_chase_distance(camera, 500);
Morfit_object_set_direction(car,-1,0,0);
```

กล้องจะตามรถโดยห่างจากรถอยู่ 500

สำหรับรายละเอียดเพิ่มเติมเกี่ยวกับการเลื่อนกล้องแบบ track, chase และ physics ให้กลับไปดูในบทที่ 3 โดยเปลี่ยนโค้ดจาก Morfit_object เป็น Morfit_camera

6.3 ทิศทางของ Camera

ทิศทางของกล้องจะเป็นตัวบอกว่ากล้องหันไปที่ใด เล็งไปที่วัตถุใดอยู่ โดยสามารถกำหนดทิศทางของกล้องได้โดยใช้คำสั่ง

```
Morfit_camera_set_direction(camera, -1, 0, 1);
```

ในคำสั่งนี้จะทำให้กล้องมีทิศทางชี้ไปหน้าจอเป็นมุมเงย 45 องศา หรือจะสั่งให้กล้องชี้ไปที่วัตถุใดที่ต้องการก็ได้โดยปกติแล้ว เราต้องใช้วิธีคำนวณหาเวกเตอร์ระหว่างวัตถุกับกล้องตามตัวอย่าง

```
Morfit_object_get_location(object, &x, &y, &z);
Morfit_camera_get_location(camera, &cx, &cy, &cz);
Morfit_camera_set_direction(camera, x-cx, y-cy, z-cz);
```

แต่ในมอร์ฟิทมีการเตรียมคำสั่งที่จำเป็นให้เกือบทั้งหมดแล้ว ในกรณีที่คุณต้องการให้กล้องชี้ไปที่วัตถุใดๆ คุณเพียงแต่ใช้คำสั่ง

```
Morfit_object_get_location(object, &x, &y, &z);
```

```
Morfit_camera_point_at(camera, x, y, z);
```

ฟังก์ชันที่เกี่ยวข้องอีกอย่างที่สำคัญก็คือ Morfit_camera_point_at_2D() คำสั่งนี้ทำหน้าที่แปลงตำแหน่งบนหน้าจอที่ใช้โคออดิเนต (x,y) ว่าจุดๆ นั้นชี้ไปที่วัตถุใดในโลก 3 มิติ ทำให้คุณสามารถเลื่อนวัตถุ, หมุน หรือ เลื่อนวัตถุตามที่เมาส์คุณชี้อยู่ หรือ จุดที่กำหนดบนหน้าจอชี้อยู่ก็ได้ (เช่น เป้าเล็งกระสุน)

ในเรื่องการหมุนกล้องในมอร์ฟิทได้จัดเตรียม API ที่ใช้ในการหมุนกล้องอยู่ 2 แบบ คือ ตามมุมที่มีหน่วยเป็นเรเดียน และ องศา นอกจากนั้นยังสามารถหมุนตามแกนของ Object space หรือ World space ก็ได้

```
Morfit_camera_rotate_x(DWORD camera_handle, double degrees, int space_flag);
Morfit_camera_rotate_y(DWORD camera_handle, double degrees, int space_flag);
Morfit_camera_rotate_z(DWORD camera_handle, double degrees, int space_flag);

Morfit_camera_rotate_x_radians(DWORD camera_handle, double radians, int space_flag);
Morfit_camera_rotate_y_radians(DWORD camera_handle, double radians, int space_flag);
Morfit_camera_rotate_z_radians(DWORD camera_handle, double radians, int space_flag);
```

นอกจากนี้เรายังสามารถหมุนกล้องตามมุม Tilt, Head และ Bank โดย

การหมุนตามมุม Tilt จะเทียบเท่าการหมุนรอบแกน Y

tilt = 0: camera looks straight.

tilt = 90: camera looks up

tilt –90: camera looks down

การหมุนตามมุม Head จะเทียบเท่าการหมุนรอบแกน Z ของโลก 3 มิติ

head angle = 0: camera looks forward into the screen

head angle = 90: camera looks left

head angle = -90 (or 270): camera looks right

head angle = 180: camera looks from the screen out

การหมุนตามมุม Bank จะเทียบเท่าการหมุนรอบแกน X

bank angle = 90: camera is turned 90 degrees counterclockwise

bank angle = -90 (or 270): camera is turned 90 degrees clockwise

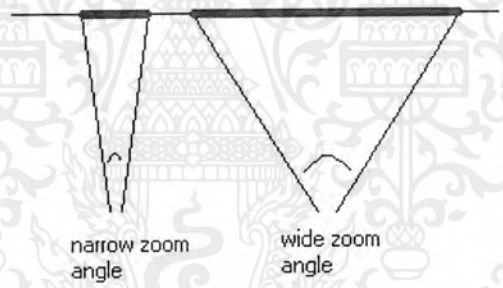
bank angle = 180: camera is upside down

ตัวอย่างโค้ดการตั้ง, แก้ไข และ อ่านค่ามุม Tilt, Head และ Bank ของ Morfit

```
double Morfit_camera_set_tilt(DWORD camera_handle, double angle);
double Morfit_camera_get_tilt(DWORD camera_handle);
double Morfit_camera_modify_tilt(DWORD camera_handle, double change);

double Morfit_camera_set_head_angle(DWORD camera_handle, double angle);
double Morfit_camera_get_ head_angle (DWORD camera_handle);
double Morfit_camera_modify_ head_angle (DWORD camera_handle, double change);

double Morfit_camera_set_bank(DWORD camera_handle, double angle);
double Morfit_camera_get_bank (DWORD camera_handle);
double Morfit_camera_modify_bank (DWORD camera_handle, double change);
```

6.4 การซูม (Zoom)

การซูมหมายถึงพื้นที่ที่เรามองเห็นนั้นเอง ยิ่งพื้นที่ที่เรามองเห็นน้อย ภาพที่เห็นก็จะใหญ่ หรืออีกนัยหนึ่งก็คือความกว้างของมุมที่มองเห็นนั้นเอง



รูปที่ 6-1 การซูม

ตัวอย่างโค้ด

```
double Morfit_camera_set_zoom(DWORD camera_handle, double field_of_view_angle);
double Morfit_camera_get_zoom(DWORD camera_handle);
double Morfit_camera_modify_zoom(DWORD camera_handle, double field_of_view_change);
```

ในมอร์ฟิทสามารถตั้งค่ามุมของพื้นที่ที่มองเห็นได้ตั้งแต่ 1 องศา – 179 องศา

6.5 ตัวอย่างโปรแกรม Camera Control

โปรแกรมนี้จะแสดงถึงวิธีการหมุน และ เลื่อนกล้อง โดยจะให้ผู้ใช้เป็นผู้ควบคุมกล้องโดยใช้ปุ่มลูกศรขึ้น และ ลง ในการควบคุมกล้อง โดยมีโหมดการควบคุมดังนี้

Key	Mode
M	**M**ove camera forward and backward
X	Rotate camera around its **x**-axis
Y	Rotate camera around its **y**-axis
Z	Rotate camera around its **z**-axis
B	Increase \ decrease **b**ank angle
H	Increase \ decrease **h**ead angle
T	Increase \ decrease **t**ilt angle
F	Increase \ decrease zoom (**f**ield of view)
O	Point camera at next **o**bject
C	Switch to next **c**amera

ในขั้นแรกให้คุณสร้างโลก 3 มิติขึ้นมาโดยให้มีไดนามิกออบเจ็คและกล้องอย่างน้อย 1 ชิ้น แล้วให้บันทึกไฟล์เป็น camera.wld หลังจากนั้นสร้างโปรเจ็คใน Visual C++ แล้วพิมพ์โค้ดต่อไปนี้ลงไป

```
#include "..\\..\\..\\Engine\\Include\\mrft_api.h"
#include <iostream.h>

void main(void)
{
        DWORD MoveCamera();
        int
rc=Morfit_engine_load_world("camera.wld",".","Bitmaps",USER_DEFINED_BEHAVIOR);
        if(rc==VR_ERROR) {
                cout << "Failed to load world, aborting\n";
                cout << "look at error.log to see why\n";
                return;
        }

        Morfit_engine_set_default_rendering_window_title("esc to exit");
        Morfit_engine_maximize_default_rendering_window();
        ShowCursor(FALSE);  // Hide the cursor
        DWORD camera;

        while( (GetAsyncKeyState(VK_ESCAPE)&1) ==0 ) {
                camera = MoveCamera();
```

```
            Morfit_engine_render(NULL,camera);
        }
}

DWORD MoveCamera()
{
        static DWORD camera = Morfit_camera_get_default_camera();
        static DWORD object_handle = Morfit_object_get_first_object();
        enum movement_mode {Move, RotateX, RotateY, RotateZ, Bank,HeadAngle, Tilt, Zoom};
        static movement_mode mode=Move;
        const speed = 10;

        if (GetAsyncKeyState('C')<0)
        {
                camera=Morfit_camera_get_next_camera(camera);
                if (camera==NULL) camera=Morfit_camera_get_first_camera();
                mode = Move;
        }
        if (GetAsyncKeyState('O')<0)
        {
                object_handle=Morfit_object_get_next_object(object_handle);
                if (object_handle==NULL) object_handle=Morfit_object_get_first_object();
                double x,y,z;
                Morfit_object_get_location(object_handle,&x,&y,&z);
                Morfit_camera_point_at(camera,x,y,z);
        }



        if (GetAsyncKeyState('M')<0)
                mode=Move;
        else if (GetAsyncKeyState('X')<0)
                mode=RotateX;
        else if (GetAsyncKeyState('Y')<0)
```

```
            mode=RotateY;
    else if (GetAsyncKeyState('Z')<0)
            mode=RotateZ;
    else if (GetAsyncKeyState('B')<0)
            mode=Bank;
    else if (GetAsyncKeyState('H')<0)
            mode=HeadAngle;
    else if (GetAsyncKeyState('T')<0)
            mode=Tilt;
    else if (GetAsyncKeyState('F')<0)
            mode=Zoom;


    if (GetAsyncKeyState(VK_UP)<0)
    {
            if (mode==Move)
                    Morfit_camera_move(camera, CAMERA_SPACE, -speed, 0, 0);
            else if (mode==RotateX)
                    Morfit_camera_rotate_x(camera,speed,CAMERA_SPACE);
            else if (mode==RotateY)
                    Morfit_camera_rotate_y(camera,speed,CAMERA_SPACE);
            else if (mode==RotateZ)
                    Morfit_camera_rotate_z(camera,speed,CAMERA_SPACE);
            else if (mode==Bank)
                    Morfit_camera_modify_bank(camera,speed);
            else if (mode==HeadAngle)
                    Morfit_camera_modify_head_angle(camera,speed);
            else if (mode==Tilt)
                    Morfit_camera_modify_tilt(camera,speed);
            else if (mode==Zoom)
                    Morfit_camera_modify_zoom(camera,speed);
    }
    if (GetAsyncKeyState(VK_DOWN)<0)
    {
```

```
        if (mode==Move)
                Morfit_camera_move(camera, CAMERA_SPACE, speed, 0, 0);
        else if (mode==RotateX)
                Morfit_camera_rotate_x(camera,-speed,CAMERA_SPACE);
        else if (mode==RotateY)
                Morfit_camera_rotate_y(camera,-speed,CAMERA_SPACE);
        else if (mode==RotateZ)
                Morfit_camera_rotate_z(camera,-speed,CAMERA_SPACE);
        else if (mode==Bank)
                Morfit_camera_modify_bank(camera,-speed);
        else if (mode==HeadAngle)
                Morfit_camera_modify_head_angle(camera,-speed);
        else if (mode==Tilt)
                Morfit_camera_modify_tilt(camera,-speed);
        else if (mode==Zoom)
                Morfit_camera_modify_zoom(camera,-speed);
    }


    return camera;
}
```

6.6 การควบคุมคุณภาพของภาพ

Morfit อนุญาติให้เราสามารถตั้งค่าคุณภาพของภาพที่จะแสดงออกมาทางหน้าจอได้ ถ้าเราตั้งค่านี้ไว้สูงก็จะทำให้ความเร็วในการแสดงผลตกลง แต่ถ้าตั้งค่านี้ไว้ต่ำก็จะทำให้แสดงผลได้เร็วขึ้น

ขนาดของภาพที่มาใช้เป็น Texture ก็มีผลต่อคุณภาพของภาพเช่นกัน ถ้าใช้ภาพที่มีขนาดเล็กในโหมดการแสดงผลความละเอียดสูง (Resolution) ก็จะทำให้ภาพที่ได้แตก แต่ใช้โหมดแสดงผลต่ำภาพก็จะไม่แตกมาก การที่จะอ่านค่าความกว้าง ความยาวของหน้าจอให้ใช้คำสั่ง

```
int Morfit_camera_get_width(DWORD camera_handle);
int Morfit_camera_get_height(DWORD camera_handle);
```

ในการตั้งค่าความละเอียดของหน้าจอให้ใช้ค่า Height และ Width ที่ได้จากคำสั่งทั้งสอง โดยค่า quality จะเท่ากับ Width*Height

หมายเหตุ : อย่าตั้งค่าเองเพราะอัตราส่วนความกว้างยาวอาจจะผิดพลาดได้

```
Morfit_engine_set_picture_quality(int quality);
Morfit_engine_increase_picture_quality();
Morfit_engine_decrease_picture_quality();
```

บทที่ 7
การใช้ MFC

7.1 ทำไมจึงต้องใช้ MFC ?

ตัวอย่างโปรแกรมทั้งหมดในบทที่ผ่านมาถูกเขียนอยู่ในรูปของ Windows console applications เพื่อความง่ายต่อการเข้าใจ console applications มักมีขนาดสั้น ๆ และเพียงแค่เรามีความรู้ในการเขียนภาษา C++ หรือ C ก็เพียงพอแล้ว อย่างไรก็ตาม console applications มีข้อเสียหลายอย่างเช่น ต้องมีการเปิดหน้าต่าง DOS ขึ้นมาทุกครั้งที่เรารันโปรแกรม ในทางกลับกัน applications ที่เขียนในรูปแบบของ MFC มีลักษณะและการทำงานเหมือนโปรแกรมบน Windows ทั่ว ๆ ไป มีการตอบสนองต่อการอินพุตของผู้ใช้ก็เป็นเรื่องที่ง่ายกว่าทั้งในการใช้คีย์บอร์ดและเมาส์ และมันยังอนุญาตให้เราสร้างกรอบโต้ตอบและเมนูในโปรแกรมของเราได้อีกด้วย

ในบทนี้จะเป็นการแนะนำการใช้ Mofit SDK กับ MFC แบบคร่าว ๆ แต่จะไม่ขออธิบาย MFC โดยละเอียด เพราะจะต้องใช้เวลานานมาก ในที่นี้จะเป็นเพียงการสอนความารู้ที่จำเป็นในการสร้าง MFC projects ที่ใช้ Morfit 3D graphics tools

7.2 เริ่มด้วยการสร้าง World ใหม่ก่อน

โปรแกรมในบทนี้จะใช้ world ชื่อ "shapes.wld" ให้เราสร้าง world โดยใช้ BlankFloor template วางรูปร่าง 3 มิติอะไรก็ได้ลงบน world สัก 3 ชิ้นแล้วทำให้เป็น dynamic object ทั้งหมด สร้าง camera จากนั้นลบพื้นลายหมากรุกโดยการ release แทนที่จะ save as จะมีกรอบโต้ตอบขึ้นมาให้เราเลือก "delete template polygon" และไม่เลือก "Subgroup to flat mode" โดยเด็ดขาดเพราะถ้าเลือกแล้วจะทำให้วัตถุใน world ไม่เป็น dynamic และไม่สามารถเลือกใช้ทีละวัตถุได้ แล้วจึงกด OK

7.3 สร้าง Hello, World! โดยใช้ Morfit Application Wizard

ใน Microsoft Visual Studio เลือก File->New คราวนี้ในแท็บ projects ให้เลือก Morfit AppWizard ตั้งชื่อ project ว่า "mfcdemo" แล้วกด OK Visual Studio จะสร้าง application ใหม่ขึ้นมาซึ่งจะสนับสนุน MFC และ Morfit 3D Engine จากนั้นให้เลือก Project->Settings และเลือกแท็บ Debug ตั้ง working directory ให้ตรงกับที่เราใช้เก็บ "shapes.wld" สังเกตได้ว่า AppWizard ได้เชื่อมโยงกับ Morfit library และ Header File ไว้แล้ว

เมื่อ AppWizard สร้าง project ขึ้นมาแล้ว มันจะเพิ่ม code ตัวอย่างบางส่วนลงไปในบาง function ซึ่งเราจำเป็นต้องลบบางส่วนของมันทิ้งเพื่อสร้างโปรแกมของเราให้สมบูรณ์

เรามาทบทวนความจำสักเล็กน้อย ด้านล่างนี้เป็นรูปแบบของ console applications:

```
// Initialization
Load the world
```

```
// Main rendering loop
Repeat until user presses <esc>:
        Move camera and/or objects
        Display world
```

ปัญหาที่สำคัญของรูปแบบนี้คือมันจะครอบครองใช้งาน CPU อยู่ตลอดเวลา เพราะมันจะคอยตรวจสอบอินพุตจากผู้ใช้อยู่โดยตลอดแม้ว่าจะไม่มีอินพุตเข้ามา รวมทั้งมันยัง render world อยู่ซ้ำ ๆ ไปเรื่อย ๆ แม้ว่ารูปบนหน้าจอจะไม่มีการเปลี่ยนแปลงใด ๆ เลยก็ตาม

ใน MFC (รวมทั้งโปรแกรมบน Windows ส่วนใหญ่) เราไม่จำเป็นต้องทำการหน่วงเวลาเพื่อตรวจสอบอินพุต Windows จะเป็นตัวบอกเราเองเมื่อมี "event" เกิดขึ้น โดย Windows จะบอกโปรแกรมของเราในแต่ละครั้งที่ผู้ใช้กดปุ่มคีย์บอร์ด, เคลื่อนเมาส์ หรือคลิ้กเมาส์ เรายังสามารถเรียกใช้ Windows ให้บอกโปรแกรมของเราเมื่อมี event อื่น ๆ เกิดขึ้นได้อีกเช่น การตั้งเวลา, การคลิ้กบน menu

โครงสร้างของ MFC โปรแกรมเป็นดังนี้:

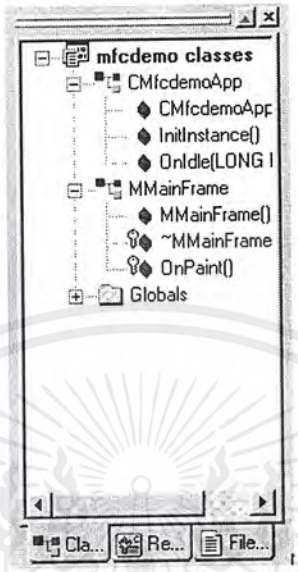
```
// Initialization
Load the world
Display the world
// Respond to Events
Event 1:
        Modify the world
        Display the world
Event 2:
        Modify the world
        Display the world
.
.
.
```

ให้พยายามทำความเข้าใจความแตกต่างของ program model ทั้ง 2 แบบ จะเห็นได้ว่าไม่มี "rendering loop" ใน MFC เราเพียงแค่จัดเตรียม world และตอบสนองต่อ events ต่าง ๆ เมื่อ Windows บอกเราว่ามันเกิดขึ้น

7.3.1 การกำหนดค่าเริ่มต้น

MFC เป็นมากกว่าคลาส มันเป็น "application framework" ใน programming model ทั่ว ๆ ไป libraries จะให้กลุ่มของ function ที่เราเรียกใช้ในโปรแกรมได้ แต่ใน application framework เราจะเป็นคน

กำหนดฟังก์ชันให้มันเรียกใช้ เราไม่จำเป็นต้องกำหนดฟังก์ชัน main (หรือ winmain) MFC จะเป็นตัวจัดการให้ อาจจะมีคำถามขึ้นมาว่า "แล้วเราจะกำหนดค่าเริ่มต้นไว้ที่ไหนล่ะ ?" คำตอบก็คือในฟังก์ชัน InitInstance() ซึ่ง MFC รับประกันว่าจะถูกเรียกเป็นตัวแรกเมื่อเราเริ่มรันโปรแกรมในหน้าต่าง Workspace เลือกแท็บ ClassView ขยายสาขาของ tree ลงมาจะเป็นดังรูป 7-1



รูปที่ 7-1 ClassView

**MmainFrame อาจจะเป็น CmainFrame() ก็ได้

ในมุมมองนี้จะแสดงคลาสต่าง ๆ ในโปรแกรมของเรารวมทั้งฟังก์ชันและตัวแปรที่มันเก็บ ฟังก์ชัน InitInstance() จะถูกสร้างโดย AppWizard ดับเบิ้ลคลิ้กมันเพื่อเรียกฟังก์ชัน body ของมันขึ้นมา แล้วลองดูผ่าน ๆ เพื่อหาส่วนที่มีการเรียกใช้ฟังก์ชัน LoadWorld แล้วแทนที่ด้วย:

```
if(OK != Morfit_engine_load_world("shapes.wld", "", "Bitmaps",
USER_DEFINED_BEHAVIOR ))
        return false;
```

ฟังก์ชัน InitInstance() นี้จะเป็นส่วนที่เราสามารถกำหนดค่าเริ่มต้นและการทำงานต่างที่เราต้องการให้มีการทำงานเพียงครั้งเดียวในตอนเริ่มรันโปรแกรม

จากนั้น ดับเบิ้ลคลิ้กฟังก์ชัน OnIdle() ใน ClassView แล้วลบ Code ตัวอย่างทั้งหมดที่อยู่ระหว่างบรรทัด

```
//**** MORFIT CODE ****
//******* BEGIN *******
```

กับบรรทัด

```
// **** MORFIT CODE ****
// ******** END ********
```

7.3.2 Rendering the World

ส่วนที่เหลือจะเป็นการ render world ใน MFC เราต้องกำหนดฟังก์ชัน OnPaint() ซึ่งจะถูกเรียกทุกครั้งที่หน้าต่างของโปรแกรมของเราต้องการการอัพเดต ฟังก์ชันนี้จะถูกเรียกเมื่อหน้าต่างถูกสร้างขึ้นมาในตอนแรก, เมื่อมันเปลี่ยนขนาด หรือเมื่อไรก็ตามที่เราบอก Windows ว่าสิ่งที่อยู่บนหน้าต่างนั้นต้องการการอัพเดต เรียกฟังก์ชัน OnPaint() ขึ้นมาจากแท็บ Class View แล้วสังเกตบรรทัด:

```
if (Morfit_engine_render(m_hWnd, Morfit_camera_get_default_camera()) != OK)
return;
```

พารามิเตอร์ตัวแรกคือ HWND (เป็น window handle) ของหน้าต่างที่เราต้องการ render และพารามิเตอร์ตัวที่สองคือ camera ที่เราต้องการใช้

เราไม่จำเป็นต้องทำเปลี่ยนแปลงอะไรในฟังก์ชันนี้ คอมไพล์แล้วรันโปรแกรม เราควรจะเห็น world แสดงอยู่ในหน้าต่างโปรแกรมของเรา ลองเปลี่ยนขนาดหน้าต่างดู จากนั้นเลือก File->Exit MFC AppWizard จะสร้าง code เพื่อจัดการการทำงานนี้รวมทั้งการ Minimize, Maximize และปุ่มปิดที่อยู่มุมขวาบนของหน้าต่าง

7.4 สร้างโปรแกรมที่ซับซ้อนกว่า Hello, World!

ในหัวข้อนี้เราจะทำให้ world ซับซ้อนและเป็น dynamic ขึ้นเล็กน้อย โดยเพิ่มฟังก์ชันเอตอบสนองต่อ events จากผู้ใช้ โดยเราจะเลือกให้ตอบสนองต่อ timers และ mouse clicks แต่ก่อนอื่นเรามาเขียน code ที่จะรันเมื่อมี "event" ที่ไม่มีอะไรเกิดขึ้น (Idle State)

7.4.1 การประมวลผลในสถานะที่ว่าง (Idle Processing)

application ของเราใช้เวลาส่วนใหญ่ไปกับการตอบสนองต่อ Windows Messages เมื่อ message queue ว่างและโปรแกรมไม่มีอะไรจะทำ application จะเรียกใช้ฟังก์ชัน OnIdle เรามาลองกำหนดฟังก์ชัน OnIdle ที่จะหมุนรูปทรงดู ซึ่งจะทำการหมุนรูปทรงใน world ไปเรื่อย ๆ

ดับเบิ้ลคลิ้กฟังก์ชัน OnIdle() และระหว่าง MORFIT CODE BEGIN และ MORFIT CODE END ให้แทรก code ด้านล่างนี้เข้าไป:

```
for(DWORD shape=Morfit_object_get_first_object() ; shape!=NULL ;
shape=Morfit_object_get_next_object(shape) )
{
        Morfit_object_rotate_x(shape,.5,WORLD_SPACE);
        Morfit_object_rotate_y(shape,.5,WORLD_SPACE);
        Morfit_object_rotate_z(shape,.5,WORLD_SPACE);
```

```
}
```

code นี้จะหมุนแต่ละวัตถุใน world ไป .5 องศาไปตาแต่ละแกน แต่เพียงแค่การหมุนวัตถุยังไม่เพียงพอ เพราะในขณะที่เราอาจจะเปลี่ยนสถานะของ world เราจำเป็นที่จะต้องแน่ใจว่าการเปลี่ยนแปลงนั้นต้องแสดงให้เห็นบนหน้าจอด้วย ซึ่งสิ่งที่เราต้องทำก็คือเรียกใช้ฟังก์ชัน rendering (OnPaint) อย่างไรก็ตามแทนที่เราจะเรียกใช้ฟังก์ชัน OnPaint ตรง ๆ เราจะเรียกใช้ฟังก์ชัน Invalidate() แทน ซึ่งจะเป็นการบอกให้ Windows รู้ว่าสิ่งที่อยู่ในหน้าต่างของเรายังไม่ได้อัพเดต และจำเป็นต้องได้รับการวาดขึ้นใหม่ แต่การกระทำนี้ OnIdle() ได้ทำให้เราในตอนท้ายก่อนจะ return แล้วดังนี้:

```
m_pMainWnd->Invalidate(false);
```

สร้างแล้วรันโปรแกรม รูปทรง 3 มิติควรจะหมุนรอบแกนของมันแล้ว

7.4.2 การตั้งเวลา (Timer)

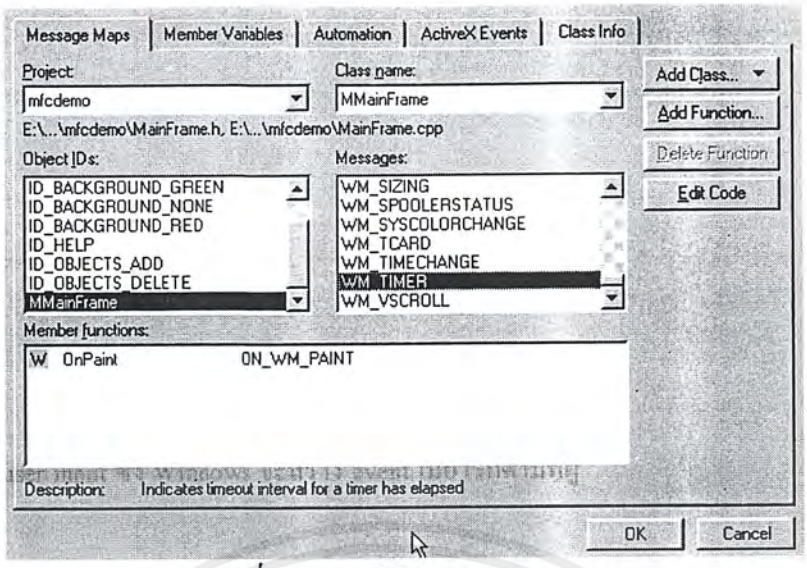
ฟังก์ชัน OnIdle จะมีประโยชน์เมื่อเราต้องการทำกิจกรรมที่ต่อเนื่องอยู่ตลอดเวลา แต่เราควบคุมความถี่ในการรันของมันได้น้อยมาก ถ้าเราต้องการทำอะไรที่เกิดขึ้นตามช่วงเวลาหนึ่ง ๆ เราควรจะใช้ timer ต่อไปเรามาลองดัดแปลงโปรแกรมของเราเพื่อให้รูปทรงเปลี่ยนตำแหน่งไปทุก ๆ 5 วินาที โดยเราจะใช้ Windows Timer

หลักการของ Windows Timer นั้นง่ายมาก เราตั้ง timer ให้สร้าง event ทุก ๆ n milliseconds และกำหนดฟังก์ชันที่จะให้ Windows เรียกใช้เพื่อตอบสนองต่อ event นั้น ให้เพิ่มบรรทัดด้านล่างเข้าไปในฟังก์ชัน InitInstance():

```
m_pMainWnd->SetTimer(1,5000,NULL);
```

พารามิเตอร์ตัวแรกคือ event ID ที่ unique ในการบ่งชี้ timer แต่ละตัว มันช่วยให้เราใช้งาน timer ได้พร้อมกันหลาย ๆ ตัว พารามิเตอร์ตัวที่สองคือคาบเวลาของ timer ในหน่วย milliseconds

จากนั้นเราจะสร้างฟังก์ชันที่ตอบสนองต่อ timer ให้เปิด Class Wizard (Ctrl-W) เลือกคลาสชื่อ "MmainFrame" (ในที่นี้อาจจะเป็น "CmainFrame" ก็ได้) ในรายการของ Object Ids เลือก "MmainFrame" แล้วดูรายการ Message และเลือก WM_TIMER message ซึ่งกรอบโต้ตอบควรจะเป็นดังรูป 7-2



รูปที่ 7-2 กรอบโต้ตอบ Class Wizard

ดับเบิ้ลคลิ้ก WM_TIMER message หรือกด "Add Function" จากนั้นดับเบิ้ลคลิ้ก OnTimer ในรายการฟังก์ชันสมาชิก หรือกด "Edit Code" แล้วเพิ่ม code ด้านล่างซึ่งจะทำการสลับตำแหน่งของแต่ละวัตถุ:

```
KillTimer(1);
DWORD shape1=Morfit_object_get_first_object();
DWORD shape2=Morfit_object_get_next_object(shape1);
DWORD shape3=Morfit_object_get_next_object(shape2);
double loc1[3],loc2[3],loc3[3],temp[3];
Morfit_object_get_location(shape1,&loc1[0],&loc1[1],&loc1[2]);
Morfit_object_get_location(shape2,&loc2[0],&loc2[1],&loc2[2]);
Morfit_object_get_location(shape3,&loc3[0],&loc3[1],&loc3[2]);
temp[0]=loc1[0];
temp[1]=loc1[1];
temp[2]=loc1[2];
Morfit_object_set_location(shape1,loc2[0],loc2[1],loc2[2]);
Morfit_object_set_location(shape2,loc3[0],loc3[1],loc3[2]);
Morfit_object_set_location(shape3,temp[0],temp[1],temp[2]);
Invalidate(false);
SetTimer(1,5000,NULL);
```

สังเกตว่าเราจะหยุด timer ที่จุดเริ่มต้นของฟังก์ชันและรีเซ็ตมันในตอนท้าย ซึ่งจะเป็นการป้องกันฟังก์ชันจากการถูกเรียกครั้งที่สองในขณะที่กำลังจัดการกับ timer event อันแรกอยู่ ถึงแม้ว่าในที่นี้จะยังไม่เกิดปัญหานี้แน่ ๆ (เพราะทำงานเพียงเล็กน้อย และมีเวลาทำถึง 5 วินาทีระหว่าง event) แต่เป็นสิ่งที่ควรฝึก

ให้เป็นนิสัย จากนั้นเมื่อเราทำการเปลี่ยนแปลง world แล้วเราก็ต้องเรียก Invalidate(False) อีกเช่นเคย เพื่อบังคับให้ Windows วาดหน้าต่างของเราใหม่

คอมไพล์แล้วรันโปรแกรม จะพบว่านอกจากวัตถุจะหมุนแล้ว มันจะสลับตำแหน่งไปทุก ๆ 5 วินาทีด้วย แต่ถ้าวัตถุหมุนไปตามแกนเดียวกัน และเมื่อครบ 5 วินาทีแล้วพบกรอบแสดงความผิดพลาดว่า Object Handle is NULL ให้ลองกลับไปดูว่าเรา release world ถูกหรือยัง ดู 7.2 ประกอบ

7.4.3 อินพุตจากผู้ใช้

event ประเภทสุดท้ายที่เราจะแนะนำคือ user input ซึ่ง Windows จะสร้าง event เมื่อไรก็ตามที่ผู้ใช้กดคีย์บอร์ด, เคลื่อนเมาส์ หรือคลิ้กเมาส์

เราจะแสดงว่าจะจัดการกับ user input ได้อย่างไร โดยอนุญาตให้ผู้ใช้เลื่อนรูปทรง 3 มิติโดยใช้เมาส์คลิ้กเลือกวัตถุ การคลิ้กครั้งที่ 2 จะเคลื่อนวัตถุไปยังตำแหน่งใหม่

ก่อนอื่น ให้เราเพิ่มตัวแปรเข้าไปยังคลาส MmainFrame เพื่อเก็บค่า handle ของวัตถุที่ถูกเลือก โดยการ
คลิ้กขวาบนคลาส MmainFrame แล้วเลือก “Add member variable” เพิ่มตัวแปร protected ชื่อ selected มี type DWORD ใน MmainFrame() constructor กำหนดค่าเริ่มต้นของ selected เป็น NULL:

```
MMainFrame::MMainFrame()
{
        selected = NULL;
}
```

เราต้องการที่จะตอบสนองต่อการคลิ้กซ้าย ให้เปิด Class Wizard เลือกคลาสชื่อ "MmainFrame" (ในที่นี้อาจจะเป็น "CmainFrame" ก็ได้) ในรายการของ Object Ids เลือก "MmainFrame" อีกครั้ง คราวนี้เลือก WM_LBUTTONDOWN message เลือก "Add Function" แล้วเลือก "Edit Code" เพื่อเรียกฟังก์ชัน OnLButtonDown() ขึ้นมา

ต่อไปนี้เป็น algorithm ที่เราจะใช้:

```
In response to the first click:
        If the user clicked on an object, select it
        If the user clicked on an empty spot, ignore
In response to the second click:
```

```
Calculate the 3D coordinates of the point that the user clicked
Move the object to the new location
```

จากนั้นเรามาทำการตอบสนองต่อการคลิ้กเมาส์ครั้งแรก เราจะแยกแยะการคลิ้กครั้งแรกและครั้งที่สองได้อย่างไร ? การคลิ้กครั้งแรกจะเกิดขึ้นเมื่อไม่มีวัตถุใดถูกเลือกอยู่ (selected=NULL) ในขณะที่ครั้งที่สองจะเกิดขึ้นเมื่อวัตถุถูกเลือกอยู่แล้ว

ในการที่จะหาว่าวัตถุใดถูกคลิ้ก เราจะใช้ฟังก์ชัน Morfit_engine_2D_to_3D() ซึ่งรายละเอียดให้ดูในบทที่ 7 ในที่นี้จะอธิบายเพียงคร่าว ๆ คือถ้ามีวัตถุอยู่บนจุดที่คลิ้ก (x,y) มันจะทำการเปลี่ยนให้จุดมาอยู่ในระบบ 3D coordinates และยังส่งค่า handle ของวัตถุและโพลีกอนที่ถูกคลิ้กกลับมาด้วย แต่ถ้าไม่มีวัตถุในตำแหน่งนั้นมันจะคืนค่า [0, 0, 0] และ handle เป็น NULL ซึ่งเราจะตรวจสอบจุดที่ผู้ใช้คลิ้กบนพิกัด 2 มิติได้จากพารามิเตอร์ point ของฟังก์ชัน OnLButtonDown() และ code ด้านล่างนี้จะใช้ในการตอบสนองต่อการคลิ้กครั้งแรก:

```
double result[3];
DWORD polygon,object;
if (selected==NULL)
{
        if (Morfit_engine_2D_point_to_3D(point.x, point.y, result, &object, &polygon)==OK)
                if (object!=NULL)
                {
                        selected=object;
                        KillTimer(1);
                }
}
```

เมื่อผู้ใช้คลิ้กบนวัตถุ เราจะเก็บค่า handle ของวัตถุไว้ในตัวแปรสมาชิก selected และเรายังหยุด timer เพื่อที่จะให้รูปทรง 3 มิติหยุดการสลับตำแหน่งในขณะที่วัตถุใดวัตถุหนึ่งกำลังถูกเลือก

แล้วเราจะตอบสนองต่อการคลิ้กครั้งที่สอง ? ถ้าดูผิวเผินเพียงแค่ code ด้านล่างนี้ก็น่าจะเพียงพอแล้ว:

```
else
{
Morfit_engine_2D_point_to_3D(point.x,point.y,result,&object, &polygon);
Morfit_object_set_location(selected,result[0],result[1],result[2]);
}
```

แต่โชคไม่ดีที่ ถ้าผู้ใช้คลิ้กบนพื้นที่ว่างเปล่าของหน้าต่างแล้ว Morfit_engine_2D_to_3D() จะไม่สามารถคืนค่าพิกัด 3 มิติกลับมาได้ เป็นเพราะมีจุด 3 มิติจำนวนนับไม่ถ้วนที่สอดคล้องกับแต่ละพิกัด 2 มิติ (ดูรายละเอียดเกี่ยวกับปัญหานี้ในบทที่ 7) ในการคลิ้กครั้งแรกจะไม่เกิดปัญหานี้ เพราะถ้าผู้ใช้ไม่ได้คลิ้กบนวัตถุ เราก็แค่ไม่สนใจการคลิ้กนั้น แต่ในที่นี้เราต้องการให้ผู้ใช้เคลื่อนวัตถุไปยังที่ใดก็ได้ ไม่ใช่เพียงแค่บนวัตถุอื่น ๆ (ซึ่งเป็นพิกัด 3 มิติ) เท่านั้น

ในการแก้ปัญหานี้ก่อนอื่นเราจะยังใช้ฟังก์ชัน Morfit_engine_2D_to_3D() ถ้าผู้ใช้คลิ้กบนวัตถุก็จะไม่มีปัญหา มันจะคืนค่าพิกัด 3 มิติกลับมาได้ แต่ถ้าผู้ใช้คลิ้กบนพื้นหลังเราจะต้องบังคับ engine ให้เลือกจุดในพิกัด 3 มิติมา 1 จุดจากความเป็นไปได้ที่นับไม่ถ้วน โดยใช้ฟังก์ชัน Morfit_engine_2D_point_to_3D_point_on_plane() ซึ่งจะคืนค่าพิกัด 3 มิติของจุด 2 มิติที่อยู่บนระนาบที่เรากำหนดให้ ในที่นี้เราจะเลือก (แบบไร้เหตุผล) ระนาบ x=1000 ซึ่งสามารถเขียนในรูปทั่วไปได้ว่า 1x + 0y + 0z – 1000 = 0 และด้านล่างจะเป็น code ในส่วนนี้ พยายามอย่าเพิ่งใส่ใจกับฟังก์ชันเฉพาะเหล่านี้มากนัก เราจะมาดูรายละเอียดเกี่ยวกับมันอีกครั้งในบทที่ 7 ในที่นี้ให้ทำความเข้าใจว่าโปรแกรมที่จะใช้โครงสร้าง application แบบ MFC เพื่อตอบสนองต่ออินพุตจากเมาส์ได้อย่างไร

```
else
{
        if (Morfit_engine_2D_point_to_3D(point.x,point.y,result,&object,
         &polygon)==OK)
        Morfit_object_set_location(selected,result[0],result[1],result[2]);
        else
        {
                double plane[4]={1,0,0,-1000};
                if(Morfit_engine_2D_point_to_3D_point_on_plane(point.x, point.y,
plane,result)==OK)
                Morfit_object_set_location(selected,result[0],result[1],result[2]);
        }
        selected=NULL;
        SetTimer(1,5000,NULL);
}
```

หลังจากวางวัตถุไปบนตำแหน่งใหม่แล้ว เราจะรีเซ็ต timer เพื่อให้วัตถุสลับตำแหน่งต่อไปทุก ๆ 5 วินาที

รันโปรแกรม คลิ้กบนวัตถุเพื่อเลือก แล้วคลิ้กที่ใดก็ได้บนหน้าต่างเพื่อย้ายมัน

** ถ้าพบปัญหา NULL object handle ให้ดู 7.2 ประกอบ ถ้าเลื่อนตำแหน่งวัตถุไปยังตำแหน่งพื้นหลังของหน้าต่างแล้ววัตถุหายไปแสดงว่า camera อยู่บนระนาบ x ที่ห่างจาก x=1000 มากเช่นอาจอยู่ในช่วง 600 เป็นต้นจึงทำให้มองไม่เห็นวัตถุที่ย้ายไปบนระนาบ x=1000 ให้เราใช้ World Builder แก้ค่าตำแหน่งของ camera ให้ x=2000 เพื่อให้เรามองเห็นระนาบ x=1000 แน่ ๆ ก็จะแก้ปัญหานี้ได้ (ถ้ากำหนดตำแหน่ง camera ให้ x=1000จะไปทับกับระนาบที่เราจะย้ายวัตถุไปพอดีทำให้เราอาจมองไม่เห็นวัตถุเหมือนเดิม)

บทที่ 8

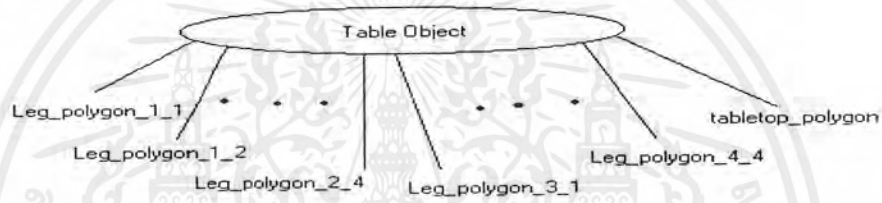
Group API

ในบทที่ 3 เราได้ศึกษาถึงวิธีการควบคุมวัตถุด้วย Object API ไปแล้ว ซึ่งการควบคุมวัตถุใน Viewer Mode ในบทนี้จะกล่าวถึงการควบคุมวัตถุใน Editor Mode ซึ่งการควบคุมวัตถุในโหมดนี้จะสามารถทำให้เราใช้ Group API ได้ แต่ว่า Editor Mode มีความเร็วต่ำกว่า Viewer mode มาก จึงควรหลีกเลี่ยงการใช้ Group API ถ้าไม่จำเป็น

8.1 อะไรคือ Group

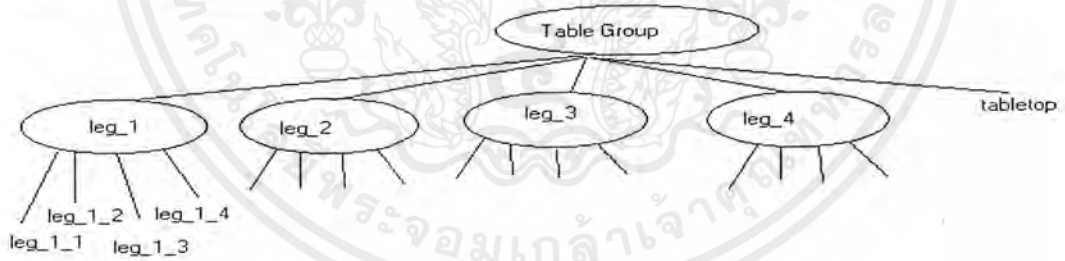
กรุ๊ป คือ โพลีกอนที่สะสมรวมกันขึ้นมาเป็นวัตถุ หรือ ส่วนย่อยของวัตถุกรุ๊ปสามารถเป็นส่วนประกอบของกรุ๊ปอื่น ทำให้เราสามารถสร้างความสัมพันธ์อย่างเป็นลำดับระหว่างกรุ๊ปได้

ยกตัวอย่างเช่น โต๊ะตัวหนึ่งประกอบด้วยขา 4 ขา และ พื้นโต๊ะด้านบน แต่ละขาประกอบด้วย 4 โพลีกอน เท่ากับว่าโต๊ะตัวนี้มี 17 โพลีกอน หากไม่มีการใช้กรุ๊ปโต๊ะก็จะมีรูปแบบดังรูปที่ 8-1



รูปที่ 8-1 ตัวอย่างการไม่ใช้ group

จะเห็นว่าไม่มีการแสดงลำดับชั้นความสัมพันธ์ระหว่างโพลีกอน โพลีกอนทั้งหมดเป็นของวัตถุโต๊ะ แต่ถ้าหากมีการจัดกรุ๊ปก็จะเป็นดังรูป 8-2



รูปที่ 8-2 ตัวอย่างการใช้ group

จากรูปแสดงให้เห็นได้ว่า leg_1_1, leg_1_2, leg_1_3 และ leg_1_4 เป็นของ group leg_1 ส่วน Group leg_1, leg_2, leg_3, leg_4, และ โพลีกอนพื้นโต๊ะด้านบน (tabletop) รวมกันเป็น Table Group

Group มีประโยชน์อะไร? เราลองนึกว่ามีโต๊ะกับเก้าอี้อยู่ แล้วเราเอาเก้าอี้ไปวางไว้บนโต๊ะ เมื่อคุณเลื่อนโต๊ะไปก็หมายความว่าเก้าอี้ที่อยู่บนโต๊ะก็ต้องเลื่อนตามไปด้วยเหมือนเป็นของชิ้นเดียวกัน (ถ้าคุณไม่ได้กรุ๊ปโต๊ะกับเก้าอี้ ตอนที่คุณเลื่อนโต๊ะ เก้าอี้จะไม่เลื่อนตามไปด้วย)

กรุ๊ปแต่ละกรุ๊ปจะมีตัวชี้ไปยัง Father Group จากรูป 8.2 leg_1_1 จะมี leg_1 เป็น Father Group และ leg_1 ก็จะมีวัตถุโต๊ะเป็น Father Group ส่วนตัวชี้ของวัตถุโต๊ะจะเป็น NULL หมายความว่าโต๊ะไม่ได้เป็นส่วนหนึ่งของวัตถุใดที่ใหญ่กว่า

8.2 การจัดการกับ Group

การสร้าง Group

อย่าลืม! การใช้กรุ๊ปจะต้องโหลดโลก 3 มิติ ในแบบ Editor mode โดยในพารามิเตอร์สุดท้ายในคำสั่ง Morfit_engine_load_world() จะต้องเป็น EDITOR_MODE.

การสร้างกรุ๊ปจะใช้คำสั่ง

```
DWORD Morfit_group_create(char *name);
```

หลังจากนั้นการเพิ่มโพลีกอนเข้าไปเป็นส่วนหนึ่งของกรุ๊ปนั้นจะใช้คำสั่ง

```
Morfit_polygon_set_group(DWORD polygon_handle, DWORD group_handle);
```

ส่วนการเพิ่มกรุ๊ปเข้าไปในกรุ๊ปจะใช้

```
Morfit_group_set_father_group(DWORD group_handle, DWORD father_group);
```

ในโค้ดนี้จะแสดงถึงวิธีสร้างกรุ๊ปของวัตถุโต๊ะที่ประกอบไปด้วย 4 ขา และ พื้นโต๊ะด้านบน (สมมุติให้มีการกำหนดค่าแฮนเดิลให้ไอเทมต่างๆ ทั้งหมดแล้ว)

```
DWORD table = Morfit_group_create("table");
Morfit_group_set_father_group(leg_1, table);
Morfit_group_set_father_group(leg_2, table);
Morfit_group_set_father_group(leg_3, table);
Morfit_group_set_father_group(leg_4, table);
Morfit_polygon_set_group(tabletop, table);
```

ลบ Group

ในการลบกรุ๊ปหรือโพลีกอน ออกจากกรุ๊ปให้ตั้งค่า Father Group ให้เป็น NULL

```
Morfit_group_set_father_group(leg_1, NULL) ย้าย leg_1 ออกจาก table group
Morfit_polygon_set_group(tabletop, NULL) ย้าย tabletop ออกจาก table group
```

แต่ถ้าต้องการลบสมาชิกทั้งหมดออกจากกรุ๊ปและ โลก 3 มิติ ให้ใช้คำสั่ง

```
Morfit_group_delete_members(DWORD group_handle)
```

การก็อบปี้ Group

```
DWORD Morfit_group_duplicate_tree(DWORD group_handle)
```

คำสั่งจะส่งค่าแฮนเดิลของกรุ๊ปใหม่คืนมา

การตรวจสอบ Group

ในการตรวจสอบว่าโพลีกอนเป็นสมาชิกของกรุ๊ปใดทำได้โดยใช้คำสั่ง

```
DWORD Morfit_polygon_get_group(DWORD polygon_handle);
```

ฟังก์ชันนี้จะส่งค่าแฮนเดิลของกรุ๊ปที่โพลีกอนนั้นๆ เป็นสมาชิกอยู่กลับมา ถ้าค่าที่ส่งกลับเป็น NULL แสดงว่าโพลีกอนนั้นไม่ได้เป็นของกรุ๊ปใดเลย ตามตัวอย่างที่ได้ที่ผ่านมาถ้าเรียกใช้ฟังก์ชัน

Morfit_polygon_get_group(tabletop) จะส่งค่าแฮนเดิลของ *table* กลับมา

Morfit_polygon_get_group(leg_polygon_1_1) จะส่งค่าแฮนเดิลของ *leg_1* กลับมา

การตรวจสอบว่าโพลีกอนเป็นของกรุ๊ปหรือไม่ จะใช้ฟังก์ชัน

```
Boolean Morfit_polygon_is_in_group(DWORD polygon_handle, DWORD group_handle)
```

Morfit_polygon_is_in_group(leg_polygon_1_1, leg_1) จะส่งกลับค่า True

Morfit_polygon_is_in_group(leg_polygon_1_1, table) จะส่งกลับค่า True เช่นกันเพราะ leg_1 เป็นสมาชิก

ของ Group table โพลีกอนที่เป็นสมาชิกของ leg_1 ก็ถือว่าเป็นสมาชิกของ table ด้วย

Morfit_polygon_is_in_group(polygon_handle, NULL) จะส่งกลับค่า True เสมอเพราะ NULL หมายถึงโลก 3 มิติ

การตรวจสอบ Father Group

```
DWORD Morfit_group_get_father_group(DWORD group_handle);
```

การตรวจสอบกรุ๊ปว่าเป็นของกรุ๊ปใด

```
Morfit_group_is_groupA_included_in_groupB(DWORD groupA_handle, DWORD groupB_handle);
```

ตัวอย่างการใช้

Morfit_group_get_father_group(leg_1) จะส่งกลับค่าแฮนเดิลของ Group table

Morfit_group_is_groupA_included_in_groupB(leg_1,table) จะส่งกลับค่า True

Morfit_group_is_groupA_included_in_groupB(leg_1,leg_2) จะส่งกลับค่า False

Morfit_group_is_groupA_included_in_groupB(group1_handle,group1_handle) จะส่งกลับค่า True เสมอ

Morfit_group_is_groupA_included_in_groupB(group_handle, NULL) จะส่งกลับค่า True เสมอเช่นกัน

การตรวจสอบว่าในกรุ๊ปมีโพลีกอนทั้งหมดเท่าไร

```
int Morfit_group_get_number_of_polygons(DWORD group_handle);
```

Morfit_group_get_number_of_polygons(NULL) จะส่งกลับจำนวนโพลีกอนทั้งหมดในโลก 3 มิติ

8.3 การทำงานเกี่ยวกับ Group

เราสามารถควบคุมกรุ๊ปได้เหมือนกับวัตถุ เช่น หมุน, เลื่อน, สั่งให้เคลื่อนที่ไปตาม Track หรือสั่งให้ Chasing ตามวัตถุ หรือ กล้อง โดยคุณจะต้องตั้งค่า properties ต่างๆ ใน Editor mode ก่อน หลังจากนั้นจึงค่อยไปรันใน Viewer Mode

ถ้าใช้ Editor Mode เราสามารถแก้ไขกรุ๊ปได้ เช่น ขนาด,สี, คุณสมบัติแสง และ ภาพบิตแมปที่เป็นพื้นผิว

แฮนเดิลและชื่อของ Group

การอ่านค่าแฮนเดิลของกรุ๊ปใช้วิธีเดียวกับวัตถุ และ กล้อง แต่ฟังก์ชันที่ใช้จะต่างกันเล็กน้อยดังตัวอย่าง

```
DWORD Morfit_group_get_first_group(void);
DWORD Morfit_group_get_next(DWORD group_handle);
DWORD Morfit_group_get_using_name(char *name);
DWORD Morfit_group_create();
```

ในการอ้างอิงโดยใช้ชื่อ

```
char * Morfit_group_get_name(DWORD group_handle);
Morfit_group_set_name(DWORD group_handle, char *name);
```

การอ่านค่าแฮนเดิลของโพลีกอนที่รวมกันเป็น Group

```
DWORD Morfit_group_get_using_name(char *name);
DWORD Morfit_group_get_first_polygon(DWORD group_handle);
```

การเคลื่อนย้าย Group

การกำหนดตำแหน่ง Group

```
Morfit_group_set_location(DWORD group_handle, double location[3]);
```

จะตั้งศูนย์กลางของกรุ๊ปไว้ที่ จุดที่กำหนด ในการตรวจสอบตำแหน่งจะต้องใช้

```
Morfit_group_get_center_of_mass(DWORD group_handle, double center[3]);
```

แต่จริงแล้วคำสั่งนี้จะส่งกลับค่า Center of Volume โดยมอร์ฟิทจะตรวจสอบจาก bounding box ในการหาจุดศูนย์กลาง

ฟังก์ชันที่ใช้อ่านค่า Bounding box

```
Morfit_group_ge_bounding_box(DWORD group_handle, double box[2][3]);
box[0][0] is the minimum X
box[0][1] is the minimum Y
box[0][2] is the minimum Z
box[1][0] is the max X
box[1][1] is the max Y
box[1][2] is the max Z
```

เราสามารถเลื่อนวัตถุโดยอ้างอิงจากตำแหน่งเดิมได้โดยใช้ฟังก์ชัน

```
Morfit_group_move(DWORD group_handle, double step[3], int space_flag);
```

นอกจากนี้ใน Group API ยังมีฟังก์ชันที่ใช้เลื่อนกรุ๊ปอีกอย่าง คือ

```
Morfit_group_move_to_match_point(DWORD group_handle, DWORD point_belongs_to_group,
DWORD point_to_match);
```

โดย point_belongs_to_group หมายถึงจุดที่อยู่ในกรุ๊ปและ point_to_match หมายถึงจุดที่อยู่นอกกรุ๊ปคำสั่งนี้จะเลื่อนกรุ๊ปโดยอ้างอิงจากจุด point_belongs_to_group ที่กำหนดไปยังจุด point_to_match

การหมุน Group

Group API มีฟังก์ชันหลายๆ แบบสำหรับการหมุนวัตถุ เช่น การหมุนวัตถุรอบแกนใดๆ

```
Morfit_group_rotate(DWORD group_handle, double degrees, int space_flag, int axis_flag);
```

โดย axis_flag = 0 จะหมายถึงแกน X, axis_flag = 1 จะหมายถึงแกน Y และ axis_flag = 2 จะหมายถึงแกน Z ยกตัวอย่างเช่น

```
Morfit_group_rotate(group, 45, WORLD_SPACE, 1)  หมุนกรุ๊ป 45°รอบแกน Y ของโลก 3 มิติ
Morfit_group_rotate(group, 30, OBJECT_SPACE, 0) หมุนกรุ๊ป 30°รอบแกน X ของวัตถุ
```

การหมุนกรุ๊ปนั้น จะหมุนรอบจุดศูนย์กลาง โดยเราสามารถกำหนดจุดศูนย์กลางใหม่ได้ดังนี้

```
Morfit_group_set_rotate_reference_point(DWORD group_handle, double center[3]);
Morfit_group_get_rotate_reference_point(DWORD group_handle, double center[3]);
```

ปกติแล้วค่าที่ส่งกลับจาก Morfit_group_get_rotate_reference_point() จะเป็นค่าเดียวกับ Morfit_group_get_center_of_mass() แต่ถ้า Reference point ถูกเปลี่ยนแปลงโดย Morfit_group_set_rotate_reference_point() ค่าที่ได้จะเป็นคนละค่ากันโดยค่า Center of volume จะเป็นค่าเดิม

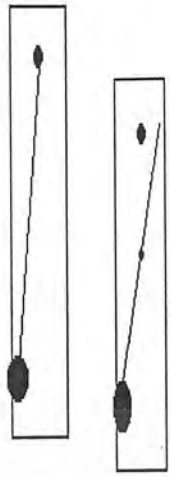
ทำไมถึงต้องตั้งค่าจุดหมุนใหม่ ลองนึกถึงการแกว่งของลูกตุ้มนาฬิกาดู โดยลูกตุ้มมี center of volume อยู่ที่จดกึ่งกลางดังรูป

รูปที่ 8-3 ภาพแสดงลูกตุ้ม

ถ้าหมุนลูกตุ้มรอบแกน X ก็จะได้ผลดังรูป 8-4

รูปที่ 8-4 ภาพแสดงลูกตุ้มหมุนรอบแกน X

จะเห็นว่าผลที่ได้ไม่เป็นอย่างที่ต้องการ เราจึงต้องเลื่อนจุดหมุนไปที่ปลายด้านบนของลูกตุ้ม จึงจะได้ผลตามที่ต้องการดังรูป 8-5



รูปที่ 8-5 การเลื่อนจุดหมุน

ในการหมุนวัตถุรอบเส้นตรงใดๆ ก็ทำได้โดยใช้ฟังก์ชัน

```
Morfit_group_rotate_around_line(DWORD group_handle, double p1[3], double p2[3], double degrees);
```

โดย p1 และ p2 จะเป็นจุดบนเส้นตรงที่ต้องการ

Group โอเรียนติง

ในขณะที่เราปล่อยวัตถุ หรือกรุ๊ปลงในโลก 3 มิติ โปรแกรมจะรู้ได้อย่างไรว่าส่วนไหนเป็นด้านล่าง หรือ ด้านบน

Group API สามารถกำหนดสิ่งเหล่านี้ได้โดยใช้ฟังก์ชัน

```
Morfit_group_set_orientation(DWORD group_handle, DWORD polygon_handle, int
orientation_value);
```

ถ้า orientation_value มีค่าเป็น ORIENTATION_TOP หรือ ORIENTATION_BOTTOM โพลีกอนจะถูกกำหนดค่าโอเรียนเทชันตามฟังก์ชัน ส่วนโพลีกอนอื่นจะถูกตั้งค่าโอเรียนเทชันเป็น ORIENTATION_UNKNOWN เพราะจะมีด้านบนหรือด้านล่างได้เพียงหนึ่งเท่านั้น การกำหนดด้านหน้าด้านหลัง ก็เช่นกัน ORIENTATION_FRONT or ORIENTATION_BACK

ส่วนการตรวจสอบว่าโพลีกอนใดเป็นด้านใดของกรุ๊ปจะใช้ฟังก์ชัน

```
DWORD Morfit_group_get_bottom_polygon(DWORD group_handle);
DWORD Morfit_group_get_top_polygon(DWORD group_handle);
DWORD Morfit_group_get_front_polygon(DWORD group_handle);
DWORD Morfit_group_get_back_polygon(DWORD group_handle);
```

โดยฟังก์ชันดังกล่าวจะส่งกลับค่าแฮนเดิลของโพลีกอนแรกในกรุ๊ปที่ถูกโอเรียนเทชัน

หลังจากกรุ๊ปถูกกำหนดด้านหน้า, ด้านหลัง, ด้านบน และ ด้านล่างแล้ว เราก็จะสามารถใช้ระบบแกนของ

กรุ๊ปได้

```
Morfit_group_calculate_axis_system(DWORD group_handle, double x_axis[3], double y_axis[3], double z_axis[3]);
```

ฟังก์ชันจะส่งกลับค่าแกน X, Y และ Z โดยแกน X จะมีทิศทางจากด้านหน้าไปยังด้านหลังของ Group, แกน Y จะมีทิศทางจากด้านซ้ายไปยังด้านขวาของกรุ๊ปและ แกน Z จะมีทิศทางจากด้านล่างไปยังด้านบนของกรุ๊ปหลังจากนี้คุณก็จะสามารถหมุนกรุ๊ปให้ตรงตามระบบแกนที่ใช้

```
Morfit_group_rotate_to_match_axis_system(DWORD group_handle, double x_axis[3], double y_axis[3], double z_axis[3]);
```

คุณสามารถเปลี่ยนจุดบน World space ไปเป็นจุดบน Group space (Object space) ได้โดยใช้ฟังก์ชัน

```
Morfit_group_convert_point_to_world_space(DWORD group_handle, double group_space_point[3], double result[3]);
Morfit_group_convert_point_to_group_space(DWORD group_handle, double world_space_point[3], double result[3]);
```

ตัวอย่างการหาจุดบน World space ที่อยู่ด้านหน้ากรุ๊ปอยู่ 5 หน่วย

```
DWORD gs[3]={-5,0,0};
DWORD ws[3];
Morfit_group_convert_point_to_world_space(group, gs, ws);
```

การเปลี่ยนขนาดของ Group

```
Morfit_group_scale(DWORD group_handle, int space_flag, double scale_x, double scale_y, double scale_z);
```

scale_x, scale_y, และ *scale_z* จะเป็นตัวกำหนดถึงขนาดของวัตถุในแกนนั้นๆ ตามที่ระบบแกนที่ *space_flag* อ้างถึงว่าจะเป็นแกนของโลก 3 มิติ หรือ แกนของวัตถุ ดังตัวอย่าง

Morfit_group_scale(group, OBJECT_SPACE, 2, 1, 1) จะเพิ่มขนาดตามแกน X เป็น 2 เท่า

Morfit_group_scale(group, OBJECT_SPACE, .5, .5, .5) จะลดขนาดกรุ๊ปเป็นครึ่งหนึ่ง

สีและแสง

เราสามารถกำหนดสีของกรุ๊ปตามแม่สีของแสง (แดง, เขียว, น้ำเงิน) ได้โดยใช้ฟังก์ชัน

```
Morfit_group_set_color(DWORD group_handle, int red, int green, int blue);
```

รวมถึงคุณสมบัติแสงของกรุ๊ปนั้นๆ ด้วย โดยในมอร์ฟิทมีแสงอยู่ 2 ชนิด คือ แสงที่มีทิศทาง หรือ แอมเบียนไลท์ (Ambient Light) และ แสงที่มีทิศทางซึ่งความสว่างของแสงแบบนี้จะขึ้นอยู่กับมุมที่แสงตก

กระทบกับผิวโพลีกอน ถ้ามุมของแสงที่สะท้อนไม่ตรงกับกล้อง แสงที่เห็นก็จะมีแต่แอมเบียนไลท์ แต่ถ้าแสงสะท้อนเข้าลำกล้องพอดี ความสว่างก็จะเท่ากับ

```
final_intensity = ambient + angle_between_ploygon_and_light_direction * light_intensity
```

การกำหนดคุณสมบัติของแสงจะใช้ฟังก์ชัน

```
Morfit_group_light_group(DWORD group_handle, double direction[3], double ambient, double
light_intensity);
```

ถ้า ambient หรือ light_intensity มีค่าเป็นลบก็จะทำให้เกิดเงามืดแทนที่จะเป็นแสง ในรายละเอียดของเรื่องแสงจะกล่าวถึงในบทที่ 11 ต่อไป

บิทแมป

เราสามารถกำหนดบิทแมบที่จะคลุมกรุ๊ปได้ ส่วนค่าแฮนเดิลของบิทแมบสามารถอ่านได้โดย Morfit_bitmap_load() ในรายละเอียดจะกล่าวถึงในบทที่ 8 ต่อไป การคลุมกรุ๊ปจะใช้ฟังก์ชัน

```
Morfit_group_wrap_a_bitmap(DWORD group_handle, DWORD bitmap_handle, int repeat_x, int
repeat_y);
```

ฟังก์ชันนี้จะทำงานได้ดีกว่าถ้ากรุ๊ปมีลักษณะแบน ส่วนค่า repeat_x และ repeat_y หมายถึงจำนวนบิตแมปที่มีซ้ำได้ตามแกน x และ y

แต่ในความจริงกรุ๊ปหนึ่งๆ ส่วนมากจะไม่ใช้บิทแมบ หรือ สี เพียง 1 ภาพ ในบทที่ 9 เราจะกล่าวถึงการกำหนดสี หรือ บิตแมปให้แต่ละโพลีกอน

คุณสมบัติของ Group

เราสามารถกำหนดคุณสมบัติ เช่นกรุ๊ปเป็นแบบสเตติก หรือ ไดนามิก แต่คุณสมบัติดังกล่าวจะมีผลก็ต่อเมื่อใช้ใน Viewer mode เท่านั้น

```
Morfit_group_set_static(DWORD group_handle);
Morfit_group_set_dynamic(DWORD group_handle);
int Morfit_group_is_static(DWORD group_handle);
```

การ disable วัตถุจะทำให้มองไม่เห็น และ ไม่มีส่วนในการตรวจสอบการชน ในการโหลดโลก 3 มิติวัตถุ และกรุ๊ปจะถูก enable ตามค่าดีฟอล์ท ถ้าเราต้องการโหลดกรุ๊ปโดย disable ทันทีที่โหลดเสร็จให้ใช้คำสั่ง

```
Morfit_group_load_as_disabled(DWORD group_handle, int YES_or_NO);
int Morfit_group_get_load_as_disabled_status(DWORD group_handle);
```

บทที่ 9

Engine API

9.1 การ load world

มีการใช้งานที่สำคัญ ๆ ดังนี้

- การ load world เข้ามาใน engine ทำได้โดย

```
int Morfit_engine_load_world(char *world_file_name, char *world_directory_path, char
*bitmaps_directory_path, int world_mode);
```

ฟังก์ชันจะคืนค่า OK ถ้า load สำเร็จ และ VR_ERROR ถ้าไม่สำเร็จ ส่วนค่า world_mode มีค่าได้ดังนี้ (อาจใช้ "|" เพื่อกำหนดมากกว่า 1 ค่าร่วมกันก็ได้):

USER_DEFINE_BEHAVIOR	load world ใน viewer mode
AUTO_BEHAVIOR01	เคลื่อนวัตถุหรือ camera ตามปุ่มทิศทางโดยอัตโนมัติ
EDITOR_MODE	load world ใน editor mode
DONT_USE_ZBUFFER	ไม่ใช้ z buffer

การ load world ใหม่ในขณะที่มี world ถูก load อยู่ก่อนหน้าแล้วจะเป็นการแทนที่ world เก่าด้วย world ใหม่

- การเพิ่ม world เข้าไปในขณะที่มี world ถูก load อยู่แล้ว (จะไม่ทับ world เดิม) ทำได้โดย:

```
DWORD Morfit_engine_add_world(char *world_file_name, char *world_directory_path, char
*bitmaps_directory_path, double position[3]);
```

ตัวอย่างเช่นเราสามารถ load world ที่มีรถอยู่เพิ่มเข้าไปใน world ที่เป็นถนน

** ค่า position เป็นค่าที่กำหนดตำแหน่งที่จะวางจุดกำเนิดของ world ใหม่ลงไป ฟังก์ชันนี้จะคืนค่า handle ของกรุ๊ปที่เกิดจาก world ใหม่ และส่วนใหญ่เรามักจะใช้ Morfit_group_set_location() ประกอบด้วยเพื่อให้ตำแหน่งของกรุ๊ปที่เราต้องการเป็นค่าเดียวกันกับ position ที่เราจะวางลงไปแทนที่จะเป็นจุดกำเนิดของ world ใหม่ซึ่งไม่ใช่ตำแหน่งของกรุ๊ปที่เราต้องการ

- การตรวจสอบว่ามี world ถูก load อยู่แล้วหรือไม่ ทำได้โดย:

```
int Morfit_engine_is_engine_empty(void);
```

ฟังก์ชันนี้จะคืนค่า YES ถ้ายังไม่มี world ใน engine และเป็น NO ถ้ามี world ถูก load อยู่แล้ว

- การบันทึก world จะทำให้เรานำ world กลับมาสร้างใหม่ได้อีกครั้งด้วย Morfit_engine_load_world() สามารถทำได้โดย:

```
int Morfit_engine_save(char *file_name, DWORD group_to_save, int save_flags);
```

file_name คือชื่อ file ที่เราต้องการบันทึก world ลงไป ค่าโดยปริยายคือ .wld

group_to_save คือค่า handle ของกรุ๊ปที่เราต้องการบันทึกลง disk ใช้ค่า NULL ถ้าต้องการบันทึกทั้ง world

save_flag มีค่าเป็นได้ดังนี้ (ใช้ "|" เพื่อกำหนดหลายค่าร่วมกัน):

SAVE_DEFAULT

SAVE_ONLY_WHATS_NEED ไม่บันทึกส่วนที่ไม่จำเป็นต่อการบันทึกกรุ๊ปเช่นจะบันทึกแต่ภาพเคลื่อนไหวที่จำเป็นสำหรับโพลีกอนในกรุ๊ปที่เราต้องการ cameras, พื้นหลังและ tracks จะไม่ถูกบันทึก

SAVE_FLAT_BITMAP_PATH ไม่บันทึก path สำหรับแต่ละโพลีกอนโดยอนุมาณว่าอยู่ path เดียวกัน

SAVE_BITMAP_AS_JPG บันทึกบิตแมปในรูปแบบ JPG

SAVE_BITMAP_AS_BMP บันทึกบิตแมปในรูปแบบ BMP

SAVE_BITMAP_AS_JBM บันทึกบิตแมปในรูปแบบ JBM

SAVE_IN_MULTIPLE_FILES บันทึกแต่ละวัตถุใน file ของตนเอง

SAVE_ONLY_MODELS

SAVE_RELEASE วัตถุทั้งหมดถูกรวมไว้ในโมดูล main

SAVE_BITMAP_OPTIMAZED บันทึกบิตแมปทั้งหมดในรูปแบบ JPG ยกเว้นที่เป็นแบบโปร่งใส จะบันทึกเป็น BMP

แต่ละรูปแบบของการบันทึกบิตแมปเป็นดังนี้

- JPG ขนาดเล็กที่สุดแต่คุณภาพต่ำกว่า และใช้เวลา load นานกว่า
- BMP ขนาดใหญ่กว่าแต่คุณภาพดีกว่า และใช้โปรแกรมภายนอกตกแต่งแก้ไขได้ด้วย
- JBM เป็นรูปแบบภายใน engine เอง load ได้เร็วที่สุด

9.2 การแสดง world

เมื่อเรา load world เข้ามาใน engine แล้ว เราสามารถแสดง world และควบคุมหน้าต่างการแสดงผลได้ด้วย การทำงานที่สำคัญ ๆ มีดังนี้

- การ render world ทำได้โดย:

```
Morfit_engine_render(HWND hwnd, DWORD camera);
```

ฟังก์ชันนี้อาจเป็นฟังก์ชันที่สำคัญที่สุดใน Morfit SDK เลยก็ว่าได้ เพราะมันจะนำสถานะปัจจุบันของ world ที่ถูก load อยู่ขึ้นแสดงผลบนจอภาพ ค่า hwnd คือค่า handle ของหน้าต่างที่เราต้องการจะ render ค่าโดยปริยายคือ NULL ส่วนใน MFC ถ้าเราต้องการ render ที่หน้าต่างหลักของโปรแกรม เราสามารถทำได้โดย:

```
Morfit_engine_render(m_hWnd, camera_handle);
```

** ถ้าเราเรียกใช้ Morfit_engine_render() ใน viewer mode นอกจากที่มันจะแสดง world แล้วมันยังทำการเคลื่อนวัตถุในแบบอัตโนมัติด้วย ถ้าเรากำหนดไว้ก่อน เช่น ตาม track, physicsm, การเคลื่อนที่ของ camera เป็นต้น ถ้าเราต้องการยกเลิกการเคลื่อนที่แบบอัตโนมัติเหล่านั้น สามารถทำได้โดยการเรียกฟังก์ชันต่อไปนี้ด้วยพารามิเตอร์ NO:

```
Morfit_engine_advance_objects_automatically(yes_or_no_flag);
Morfit_engine_advance_cameras_automatically(yes_or_no_flag);
```

- การควบคุมหน้าต่างโดยปริยายที่เรา render มีดังนี้:

```
Morfit_engine_set_default_rendering_window_size(int left_x, int top_y, int width, int height);
Morfit_engine_maximize_default_rendering_window(void);
Morfit_engine_minimize_default_rendering_window(void);
Morfit_engine_set_default_rendering_window_title(char *text);
Morfit_engine_set_default_rendering_window_size(int left_x, int top_y, int width, int height);
HWND Morfit_engine_get_default_rendering_window_hwnd(void);
```

- การแสดงเพียงบางส่วนของ world ทำได้โดย:

```
Morfit_engine_set_group_to_render(DWORD grp_to_render_handle);
```

ถ้าต้องการแสดงผลทั้ง world เหมือนเดิมให้ใช้พารามิเตอร์ NULL

- การกำหนดค่าความละเอียดในการแสดงผลและความลึกของสีทำได้โดย:

```
Morfit_engine_set_resolution(int width, int height, int bits_per_pixel);
Morfit_engine_get_resolution(int *width, int *height, int *bits_per_pixel);
Morfit_engine_set_color_depth(int bits_per_pixel);
```

ค่า bit_per_pixel เป็นได้ทั้ง 8,16 หรือ 24 มอร์ฟิททำงานได้เร็วที่สุดที่ 16 bpp ซึ่งเป็นค่าโดยปริยาย ส่วนค่าความละเอียดของการแสดงผลโดยปกติคือ 640 x 480, 800 x 600 และ 1024 x 768

- การ render ไปยัง memory ทำได้โดย:

```
HDC Morfit_engine_render_on_memCDC(HWND hwnd, DWORD camera_handle);
```

ฟังก์ชันนี้จะคืนค่า Hardware Device Context (HDC) ของภาพที่ render ค่า HDC นี้จะถูกนำไปใช้ในหลาย ๆ คำสั่งที่ใช้ในการวาดภาพของ Windows API

หลังจากแก้ไขภาพใน memory เรียบร้อยแล้วเราสามารถนำภาพมาแสดงผลได้โดย:

```
Morfit_engine_bitblt();
```

9.3 ความแม่นยำในการแสดงผล

ในการ render เราต้องเลือกระหว่างคุณภาพรวมทั้งความแม่นยำของภาพกับความเร็ว ถ้าเราต้องการภาพคุณภาพดีมีความแม่นยำสูงเราก็ต้องสูญเสียความเร็วในการ render ไปในทางกลับกันก็เช่นกัน ในหัวข้อนี้จะกล่าวถึง tools ที่ใช้กำหนดค่าต่าง ๆ ซึ่งมีผลต่อประสิทธิภาพและคุณภาพของภาพ ให้เราทดลองและเลือกกำหนดให้เหมาะสมกับโปรแกรมของเรามากที่สุด การทำงานที่สำคัญ ๆ ได้แก่:

- คุณภาพของภาพ คือพื้นที่ทั้งหมดของ world ที่ camera กำลังแสดงผล ถ้าเราแสดงพื้นที่เล็ก ๆ ในหน้าต่างใหญ่ ๆ คุณภาพของภาพจะต่ำมาก ในทางกลับกันถ้าเราแสดงพื้นที่ใหญ่บนหน้าต่างขนาดเล็กคุณภาพของภาพจะสูง เราสามารถกำหนดคุณภาพของภาพได้โดย:

```
Morfit_engine_set_picture_quality(int quality);
int Morfit_engine_get_picture_quality();
Morfit_engine_increase_picture_quality();
Morfit_engine_decrease_picture_quality();
```

- การแสดงวัตถุที่อยู่ไกล ๆ บางครั้งเราไม่จำเป็นต้องใช้ค่าความแม่นยำเดียวกันทั้งภาพ ตาของมนุษย์จะเห็นวัตถุที่อยู่ไกลไม่ชัดเท่าวัตถุที่อยู่ใกล้ รายละเอียดของวัตถุที่อยู่ไกล ๆ จึงไม่จำเป็นต้องมีมากนัก เราจะประหยัดเวลาได้มากถ้าเรา render ในลักษณะนั้น คือซ่อนรายละเอียดของวัตถุที่อยู่ไกลและใช้เวลาที่ได้มาแสดงวัตถุที่อยู่ใกล้ให้มีคุณภาพสูง ซึ่งมีอยู่หลายวิธีได้แก่

1) กำหนดให้บิตแมปของวัตถุที่อยู่ไกลถูกแทนที่ด้วยสีเต็ม ๆ (ไม่มีลายของบิตแมปเลย) ทำได้โดย

```
Morfit_engine_set_far_objects_color_accuracy(int value);
int Morfit_engine_get_far_objects_color_accuracy();
Morfit_engine_increase_far_objects_color_accuracy();
Morfit_engine_decrease_far_objects_color_accuracy();
```

value มีค่าอยู่ระหว่าง 0 และ NUMBER_OF_PALETTES -1

2) เลือกแสดงวัตถุที่อยู่ใน ระยะที่กำหนดเท่านั้นและจะไม่แสดงวัตถุที่อยู่ไกลกว่าระยะนั้น ทำได้โดย:

```
Morfit_engine_set_culling_depth(int value);
int Morfit_engine_get_culling_depth();
Morfit_engine_increase_culling_depth();
Morfit_engine_decrease_culling_depth();
```

3) การใช้ atmospheric effect หรือ fog ในการบังบิตแมปของวัตถุที่อยู่ไกล ๆ (ซึ่งจะทำให้สมจริงกว่าการให้วัตถุปรากฏขึ้นมาทันทีเมื่อเข้าใกล้ในแบบข้อ 2)) แล้วค่อย ๆ แสดงรายละเอียดให้มากขึ้นเมื่อเข้าใกล้วัตถุนั้น ทำได้โดย:

```
Morfit_engine_set_atmospheric_effect_intensity(double value);
double Morfit_engine_get_atmospheric_effect_intensity();
Morfit_engine_increase_atmospheric_effect_intensity();
Morfit_engine_decrease_atmospheric_effect_intensity();
Morfit_engine_toggle_atmospheric_effect();
```

เราสามารถดู effect นี้ใน World Builder ได้โดยการคลิ้กที่ปุ่ม "Increase Atmosphere"

ส่วน
การกำหนดสีของ fog ใน World Builder ทำได้โดยเลือก Custom Color tools แล้วคลิ้กที่พื้นหลังแล้วเราจะสามารถเลือกได้ว่าจะกำหนดสีพื้นหลังหรือ atmospheric effect ซึ่งโดยทั่วไปเรามักจะกำหนดให้สีของ fog เข้ากันกับสีของพื้นหลัง ส่วนการเรียกใช้คำสั่งโดยตรงทำได้โดย:

```
Morfit_engine_set_atmospheric_effect(int red, int green, int blue);
Morfit_engine_get_atmospheric_effect(int *red, int *green, int *blue);
```

ค่า red,green และ blue ควรเป็นจำนวนเต็มระหว่าง 0 ถึง 255

ฟังก์ชันดังข้างบนจะทำการกำหนดให้สีพื้นหลังเป็นสีเดียวกันกับ fog ในการกำหนดสีพื้นหลังต่างหากทำได้โดย:

```
Morfit_engine_set_background_color(int red, int green, int blue);
Morfit_engine_get_background_color(int *red, int *green, int *blue);
```

- การใช้ Z Buffer คือ algorithm ที่ engine ใช้ในการตัดสินใจว่าจะแสดงหรือซ่อนพื้นผิวใดบ้าง ส่วนใหญ่แล้วการไม่ใช้ z buffer จะให้ประสิทธิภาพที่ดีกว่า แต่ในบางครั้งการใช้ z buffer อาจจะดีกว่าก็ได้ เราควรจะทดสอบและเลือกใช้ z buffer ตามความเหมาะสมต่อโปรแกรมของเราให้มากที่สุด

ใน editor mode จะมีการใช้ z buffer เสมอ แต่ใน viewer mode เราสามารถกำหนดได้โดย:

```
int Morfit_engine_use_zbuffer(int yes_no_flag);
int Morfit_engine_is_zbuffer(); // returns YES or NO
```

- Rendering Mode มีอยู่ 3 modes ได้แก่

1) "normal" เป็นค่าโดยปริยาย จะแสดงวัตถุและบิตแมปทั้งหมด

2) "color fill" จะแสดงสีเต็ม ๆ แทนที่บิตแมปทั้งหมด

3) "wire frame" จะแสดงแต่เส้นรอบนอกของวัตถุเท่านั้น

ส่วนใหญ่เรามักจะไม่ค่อยเปลี่ยนเป็น mode อื่นนอกจาก normal มากนัก เราสามารถดู effects เหล่านี้ได้ใน World Builder โดยการเลือก camera mode แล้วคลิ้กขวาบนภาพเลือก "Render's Info" เราสามารถเปลี่ยน mode โดยเลือก "Full Render", "Without Bitmaps" หรือ "Wire-frame" ส่วนการกำหนดโดยตรงทำได้โดย:

```
Morfit_engine_set_normal_rendering_mode();
Morfit_engine_set_color_fill_rendering_mode();
Morfit_engine_set_wire_frame_rendering_mode();
```

9.4 2D และ 3D Space

มีการใช้งานที่สำคัญ ๆ ดังนี้

9.4.1 การเปลี่ยน 3D เป็น 2D

- การแปลงจุด

จุด 3D ทุก ๆ จุดใน world สามารถแปลงเป็นจุด 2D ได้โดยใช้:

```
int Morfit_engine_3D_point_to_2D(double p3D[3], int p2D[2]);
```

ฟังก์ชันนี้จะคืนค่ากลับมาโดย p2D[0] จะเป็นค่าแกน x และ p2D[1] จะเป็นค่าแกน y และจะคืนค่า int ต่าง ๆ ด้วย ดังนี้

-1: เกิด error ได้ p2D เก็บ 0,0

0: OK p2D เก็บค่าจุดในหน้าต่าง

1: จุดที่กำหนดอยู่นอกหน้าต่าง p2D เก็บค่าที่ถูกต้อง

2: จุดที่กำหนดอยู่หลัง camera ผลใน p2D ไม่ถูกต้อง

3: จุดที่กำหนดทั้งอยู่นอกหน้าต่างและอยู่หลัง camera ผลใน p2D ไม่ถูกต้อง

- การแปลงเส้น

ให้เรากำหนดจุดปลายทั้งสองของเส้นตรงเป็น p1 และ p2 แล้ว engine จะคืนค่าจุดปลายใน 2 มิติเป็น p1_2D และ p2_2D ดังนี้:

```
int Morfit_engine_3D_edge_to_2D(double p1[3], double p2[3], int p1_2D[2], int p2_2D[2]);
```

และจะคืนค่าได้ดังนี้:

EDGE_FULLY_SEEN: เส้นตรงอยู่ในหน้าต่างทั้งหมด ผลถูกต้อง

EDGE_PARTIALLY_SEEN: ส่วนของเส้นตรงอยู่ในหน้าต่าง ค่า p1_2D และ p1_3D แสดงค่าส่วนของเส้นตรงที่อยู่ในหน้าต่าง

EDGE_NOT_SEEN: เส้นตรงอยู่นอกหน้าต่างทั้งเส้น ผลไม่ถูกต้อง

VR_ERROR: เกิด error ผลไม่ถูกต้อง

- การตัดแต่งเส้น

  คล้ายกับการแปลงเส้นแต่จะคืนค่าจุดปลายมาในรูป 3D แทน และจะตัดให้เสนอยู่ในหน้าต่างพอดี ดังนี้:

```
int Morfit_engine_clip_edge(double p1[3], double p2[3], double clipped_p1[3], double clipped_p2[3]);
```

ฟังก์ชันนี้จะให้ผลเหมือน Morfit_engine_3D_edge_to_2D() ค่า clipped_p1 และ clipped_p2 เก็บค่าจุดปลายของเส้นที่ตัดแต่งแล้ว

9.4.2 การเปลี่ยน 2D เป็น 3D

- การแปลงจุดในกรณีที่จุดที่กำหนดมีโพลีกอนอยู่

  ในกรณีนี้ฟังก์ชันจะคืนค่าจุด 3D ที่ถูกต้องให้ได้:

```
int Morfit_engine_2D_point_to_3D(int x, int y, double result[3], DWORD *selected_object_handle,
DWORD *selected_polygon_handle);
```

ถ้ามีโพลีกอนอยู่ที่จุดที่กำหนดให้ ฟังก์ชันนี้จะคืนค่า OK, ค่าตามแกน 3D, และ handle ของโพลีกอนและวัตถุที่อยู่ตำแหน่งนั้น ถ้ามีมากกว่าหนึ่งโพลีกอนก็จะคืนค่าของโพลีกอนที่อยู่ใกล้ camera มากที่สุด แต่ถ้าไม่มีโพลีกอนอยู่เลยจะคืนค่า VR_ERROR

- การแปลงจุดในกรณีที่จุดที่กำหนดไม่มีโพลีกอนอยู่

  เนื่องจากจุด 2D จุดหนึ่งมีโอกาสแปลงเป็นจุด 3D หลาย ๆ จุดได้ไม่จำกัดการที่เราจะแปลงมาเป็น 3D เราจึงต้องกำหนดระนาบที่จะแปลงจุดไปวางบนระนาบนั้นด้วย โดยใช้:

```
int Morfit_engine_2D_point_to_3D_point_on_plane(int x, int y, double polygons_plane[4],double p3d
[3]); // returns SUCCESS or VR_ERROR
```

ซึ่งระนาบจะกำหนดจากรูปทั่วไปโดย Ax + By + Cz + D = 0 ดังนี้:

polygons_plane[0] = A

polygons_plane[1] = B

polygons_plane[2] = C

polygons_plane[3] = D

ตัวอย่างเช่นระนาบ x = -20 มาจาก 1x + 0y + 0z + 20 = 0 ได้ระนาบ = {1, 0, 0, 20}

- การตรวจสอบว่ามีวัตถุหรือโพลีกอนอยู่ที่จุด 2D หรือไม่

  แต่ไม่ต้องการจุด 3D กลับมา ทำได้โดย:

```
DWORD Morfit_engine_get_object_at_point_2D(int x,int y);
```

```
DWORD Morfit_engine_get_polygon_at_point_2D(int x,int y);
```

- การแปลงการเคลื่อนที่

เช่นใช้ในการลากวัตถุจากจุดหนึ่งไปยังอีกจุดหนึ่ง ทำได้โดย:

```
int Morfit_engine_translate_movement_on_screen_to_world(double p3D[3], double result_p3D[3], int
delta_x, int delta_y);
```

เรากำหนดค่าของ p3D เป็นจุดต้น, delta_x และ delta_y เป็นขนาดการเคลื่อนที่ แล้วฟังก์ชันนี้จะคืนค่าจุดที่เคลื่อนที่ไปเป็น result_p3D

ตัวอย่าง ถ้าวัตถุอยู่ที่ [x, y, z] จุดที่เมาส์อยู่เดิมเป็น (oldx, oldy) ผู้ใช้ลากเมาส์ไปยังจุด (newx, newy) ในการเคลื่อนวัตถุไปตามเมาส์ทำได้โดย:

```
int delta_x = newx – oldx;
int delta_y= newy – oldy;
double obj_loc[3] = {x,y,z};
double new_loc[3];
Morift_engine_translate_movement_on_screen_to_movement_in_world(obj_loc, new_loc, delta_x,
delta_y);
Morfit_object_set_location(object_handle,new_loc[0],new_loc[1],new_loc[2]);
```

9.5 การเคลื่อนที่

9.5.1 การตรวจสอบการชน

ถ้าเราต้องการเคลื่อนวัตถุจากจุด start_location ไปยัง end_location เราจะตรวจสอบว่ามีโพลีกอนขวางทางอยู่หรือไม่ ได้โดย:

```
int Morfit_engine_is_movement_possible(double start_location[3], double end_location[3], DWORD
*intersected_polygon, double intersection[3], DWORD *blocking_object);
```

ถ้าไม่มีอะไรขวางทางจะได้ค่า YES แต่ถ้ามีโพลีกอนมาขวางจะคืนค่า NO และคืนค่า intersected_polygon และ blocking_object ซึ่งเป็น handle ของโพลีกอนที่ขวางและวัตถุที่โพลีกอนนั้นอยู่ รวมทั้งค่า intersection ซึ่งเป็นจุดที่โพลีกอนตัดขวางการเคลื่อนที่ โดยการทำงานของฟังก์ชันนี้จะคล้ายกับ Morfit_object_is_movement_possible แต่ฟังก์ชันนั้นจะยอมให้เรายกเว้นการตรวจสอบการชนกับวัตถุที่กำหนดได้ ดูรายละเอียดในบทที่ 3

Morfit_engine_is_movement_possible() จะคืนค่าเกี่ยวกับโพลีกอนแรกที่ขวางเท่านั้น ถ้าเราต้องการจะตรวจสอบว่ามีโพลีกอนกี่โพลีกอนที่ขวางการเคลื่อนที่อยู่ ทำได้โดย:

```
int Morfit_engine_get_number_of_collisions(double point1[3], double point2[3], double
combined_normal[3]);
```

ฟังก์ชันนี้ยังคำนวณค่าเส้นตั้งฉากเฉลี่ยของโพลีกอนเหล่านั้นด้วย

9.5.2 ความเร็วในการเคลื่อนที่

ถ้าต้องการเคลื่อนวัตถุไป 50 หน่วยทุก ๆ ครั้งที่ world ถูก render ความเร็วของวัตถุจะขึ้นอยู่กับความเร็วของเครื่องคอมพิวเตอร์ที่ใช้ แต่ถ้าเราต้องการกำหนดให้ความเร็วมีค่าคงที่ไม่ขึ้นกับเครื่อง เราสามารถคำนวณความเร็วสัมพัทธ์ของหน่วยประมวลผลและควบคุมให้เคลื่อนที่ด้วยความเร็วที่ถูกต้องได้จาก:

```
factor=Morfit_engine_get_computer_speed_factor();//คืนค่าความเร็วของ CPU
corrected_number= 50*factor;
move_my_object(corrected_number);
```

**ความเร็วจะแปรเปลี่ยนไปตาม load ของ CPU ในขณะนั้น ไม่ควรใช้ค่าเดิมตลอด เราควรเรียกฟังก์ชันทุกครั้งที่ต้องการใช้

9.6 Log Window

เราสามารถควบคุม Log Windows ที่จะปรากฏเมื่อเราต้องประมวลผลอะไรเป็นเวลานาน ๆ ได้ดังนี้

- สร้าง log window:

```
Morfit_engine_set_log_window_visible();
```

- ลบ log window:

```
Morfit_engine_hide_log_window();
```

- minimize log window:

```
Morfit_log_window_minimize();
```

- ตรวจสอบว่า log window ว่า enable อยู่หรือไม่:

```
int Morfit_engine_is_log_window_visible();
```

- เพิ่มข้อความเข้าไปในหน้าต่าง:

```
Morfit_engine_log_window_output(char *text);
```

- แทนที่ข้อความปัจจุบันด้วยข้อความใหม่:

```
Morfit_engine_log_window_set_text(char *new_text);
```

- กำหนดชื่อของ log window:

```
Morfit_engine_log_window_set_title(char *new_caption);
```

- ควบคุมค่า progress และ control:

```
Morfit_engine_set_log_window_progress(int value);
Morfit_engine_set_log_window_target(int value);
int Morfit_engine_get_log_window_progress();
int Morfit_engine_get_log_window_target();
```

- การตรวจสอบค่า handle ของ log window ซึ่งมักจะใช้บ่อยใน Windows API:

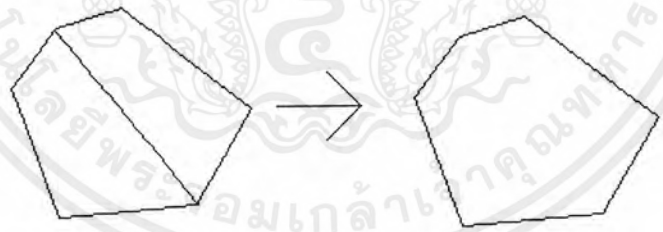
```
HWND Morfit_engine_log_window_get_hwnd();
```

9.7 ฟังก์ชันทั่ว ๆ ไป

- เพิ่มโพลีกอนเข้าไปใน world สามารถทำได้ใน editor mode เท่านั้นโดยเราต้องสร้างโพลีกอนก่อนด้วย Morfit_polygon_create() จากนั้นเพิ่มจุดให้โพลีกอนนั้นโดย Morfit_polygon_add_point() หลังจากเราปรับแต่งจนพอใจแล้วจึงใช้ฟังก์ชันด้านล่างเพื่อแสดงโพลีกอน:

```
int Morfit_engine_add_polygon(DWORD polygon_handle);
```

- ลบวัตถุที่ไม่จำเป็น คือโพลีกอนที่อยู่บนระนาบเดียวกันสามารถรวมกันได้ดังรูปที่ 9-1



รูปที่ 9-1 การรวมโพลีกอน

การทำแบบนี้จะประหยัดหน่วยความจำและเวลาที่ใช้ประมวลผล ซึ่งทำได้โดย:

```
int Morfit_engine_reduce_polygons_count(double accuracy);
```

ฟังก์ชันนี้ทำงานใน editor mode เท่านั้น และจะคืนจำนวนของโพลีกอนที่กำจัดไป ค่า accuracy คือค่าความใกล้เคียงที่จะรวมโพลีกอนเข้าด้วยกัน ถ้าเป็น 0 จะทำให้โพลีกอนที่อยู่บนระนาบเดียวกันเท่านั้นที่จะรวมกัน ถ้าเส้นตั้งฉากของทั้งสองระนาบเป็น 1.0 และ .9 แล้วโพลีกอนจะรวมกันเมื่อ accuracy เป็น .1 เท่านั้น

นอกจากนี้เรายังสามารถประหยัดการคำนวณได้โดย tools อื่น ๆ อีก เช่น

- unload บิตแมปที่ไม่ใช้แล้ว:

```
int Morfit_engine_unload_unused_bitmaps();
```

- คำนวณบิตแมปที่ถูก unload แล้ว:

```
Morfit_engine_unload_all_bitmaps();
```

- การใช้งานลำโพง เมื่อเกิด error ขึ้นจะเกิดเสียง beep และ engine จะเขียน"morfit.log" และ "error.log" เสมอ เราสามารถเปิด/ปิดการทำงานโดยที่ยังมีการเขียน log files อยู่ได้โดย:

```
Morfit_engine_set_speaker_mode(int yes_no_flag);
```

บทที่ 10
บิตแมป และ อนิเมชั่น

อย่างที่ทราบกันดีว่าพื้นผิวของโมเดลสามารถถูกเติมได้ด้วยสี หรือ บิตแมปก็ได้ ในบทนี้เราจะกล่าวถึงการทำงานกับบิตแมป ไม่ว่าจะเป็นการสร้าง, แปะมันลงบนวัตถุ หรือ ย้ายบิตแมป รวมไปถึงการทำอนิเมชั่น, การตั้งค่าบิตแมปที่ทำให้วัตถุดูเหมือนมีการเคลื่อนไหว หรือ เปลี่ยนไป โดยจะใช้ World builder ในการทำความเข้าใจกับ บิตแมป และ อนิเมชั่น รวมถึงศึกษา API ที่ใช้ในการควบคุมในโปรแกรมของเรา

10.1 การวางบิตแมป

หลังจากที่เรามีภาพบิตแมปที่ต้องการแล้วให้เปิด World Builder ขึ้นมา สร้างโลกขึ้นแบบ Blank Floor หลังจากนั้น ให้เราวาง Morfit cube ที่อยู่ในซับโฟลเดอร์ชื่อ 3Dforms ของโมเดลแกลเลอรี่ โมเดลที่ว่าจะผิวเป็นรูปโลโก้ของมอร์ฟิท

รูปที่ 10-1 รูปที่เลือกมาจากฮาร์ดดิสก์

ให้เราลองวางรูปที่สร้างขึ้นเองบนโมเดลดู โดยคลิกที่ปุ่ม Select Bitmap from Harddisk เลือกบิตแมปที่เราต้องการ หลังจากนั้นเคอร์เซอร์จะกลายเป็นรูปถังใส่สี ให้เราคลิกที่ด้านที่เราต้องการจะวางบิตแมปลงไปจะได้ผลดังรูป

รูปที่ 10-2 การระบายบิตแมป

ถ้ารูปที่วางลงไปเอียงข้างดังรูปให้คุณหมุนบิตแมป โดยเลือกที่โหมดบิตแมปแล้วเลือกที่บิตแมปที่ต้องการจะหมุน แล้วคลิกที่ปุ่มหมุน โดยให้กดปุ่ม Shift ค้างไว้ด้วย (จะหมุนทีละ 90 องศา)

10.2 การวางบิตแมปซ้อน ๆ กัน (Tilling Bitmap)

ใน World builder เมื่อเราวางรูปลงบนวัตถุ World Builder จะปรับขนาดภาพให้พอดีกับโพลีกอนที่วางลงไป แต่ถ้าเราไปแก้ไขบิตแมปให้มีขนาดเล็กลง เอนจินก็จะต้องวางบิตแมปลงไปหลายๆ ภาพ เพื่อให้เต็มโพลีกอนนั้นๆ เราเรียกการกระทำนี้ว่าทิลลิงบิตแมป (Tilling Bitmap) จริงๆ แล้วเอนจินไม่ได้ลด

ขนาดบิตแมป แต่เอนจินไปแก้ไขค่าบิตแมปโคออดิเนตของโพลีกอนนั้น เราจะอธิบายเพิ่มเติมในภายหลัง การเปลี่ยนขนาดทำได้โดยเลือกบิตแมปที่ต้องการเปลี่ยนขนาดแล้วคลิกที่ปุ่มแว่นขยาย หรือ กดคลิกขวาที่บิตแมปเมื่ออยู่ในโหมดบิตแมปเพื่อเข้าไปแก้ไขค่าคุณสมบัติต่างๆ เช่น จำนวนบิตแมปตามแนวตั้ง และแนวนอน

10.3 ความโปร่งใส

การที่เราวางบิตแมปลงไปบนพื้นผิว มันก็จะบังสิ่งที่อยู่ด้านหลังทั้งหมด แต่บางครั้งเราก็ต้องการให้มีบางส่วนที่ใสสามารถมองทะลุได้ ถ้าเราต้องการให้บิตแมปที่เราวางลงไปมองทะลุได้ ก็เพียงแต่เลือก Most Frequent Color ที่มุมล่างขวาสุดของไดอะล็อกซ์ที่แสดงขึ้นมาหลังกดปุ่ม Load Bitmap from Hard Disk บริเวณที่มีสีเป็นสีใสของภาพที่โหลดขึ้นมาก็จะสามารถมองทะลุไปได้เหมือนไม่มีอะไรอยู่ ส่วนสีที่ส่วนมากนิยมใช้เป็นสีใส ก็คือสีที่มีค่า RBG อย่างใดอย่างหนึ่งหรือมากกว่าเป็น 255 แล้วที่เหลือเป็น 0 หมด เช่น แดง, น้ำเงิน, เขียง และ เหลือง เป็นต้น

ใน World builder สีที่จะเป็นสีใสได้จะต้องเป็นสีที่นิยมใช้เป็นสีใสเท่านั้น แต่ถ้าคุณใช้ Bitmap API ในการตั้งค่าสีใส คุณจะตั้งให้สีใดเป็นสีใสก็ได้ ในการสร้างรูปที่จะมีสีใส ให้ระวังบริเวณขอบของรูปที่ติดกับบริเวณที่เราต้องการให้ใส เนื่องจากเอนจินสามารถตั้งสีใสได้เพียงสีเดียว เช่น ในภาพที่สีขาว (255,255,255) และ (254,255,253) อยู่ด้วยกัน แม้เราจะมองเป็นเป็นสีขาวเข้มทั้งหมด แต่เอนจินจะมองเป็น 2 สี และ สีที่ไม่ได้เป็นสีใสก็จะถูกแสดงผลออกมา

10.4 รูปแบบของแฟ้มข้อมูล Bitmaps

เอนจินมอร์ฟิทสนับสนุนบิตแมป 3 ฟอร์แมท คือ BMP, JPEG และ JBM โดยแต่ละฟอร์แมทก็มีข้อดีข้อเสียต่างกันไป

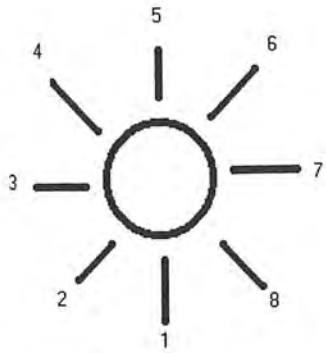
BMP มีคุณภาพสูง ง่ายต่อการแก้ไขเพราะสามารถใช้ Paint แก้ไขได้ แต่ก็กินเนื้อที่มาก

JPEG มีคุณภาพต่ำกว่า แต่ก็มีขนาดเล็กเช่นกัน อย่างไรก็ดีเนื่องจาก JPEG ใช้แอนติเอเลียสซิง

JBM เป็นฟอร์แมทของมอร์ฟิทเอง ถ้าเซฟเป็นไฟล์ชนิดนี้จะโหลดได้เร็วกว่าฟอร์แมทอื่น

10.5 ภาพเคลื่อนไหว

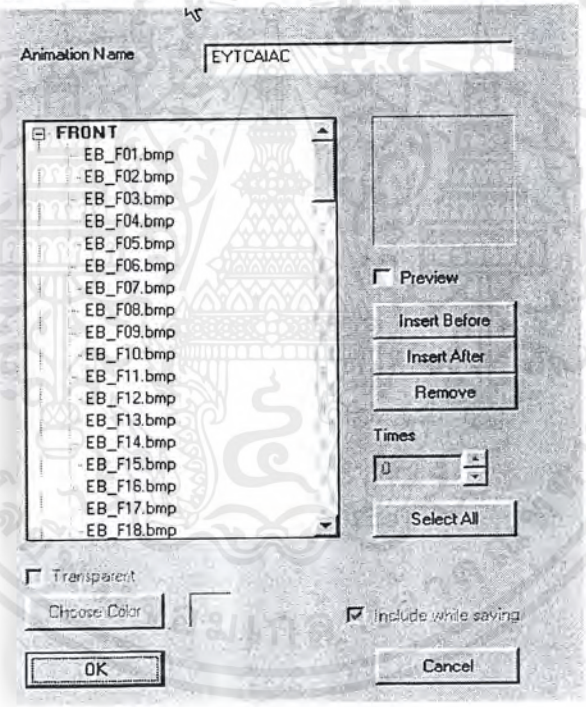
อนิเมชั่นก็คือเซตของบิตแมปที่เปลี่ยนตัวเองไปเรื่อยในช่วงเวลาหนึ่ง อย่างเช่นภาพที่แสดงออกมาบนจอทีวีเป็นต้น เนื่องจากบิตแมปอนิเมชั่นเป็นภาพ 2 มิติ เราจึงทำอนิเมชั่นไว้หลอกตา 8 ด้าน เพื่อที่จะมองอนิเมชั่นเสมือนเป็นภาพ 3 มิติ



รูปที่ 10-3 ด้านต่าง ๆ ของอนิเมชั่น

แต่ถ้าพิจารณาดูจะพบว่าด้านบางด้าน เช่น 2 กับ 8, 3 กับ 7 และ 4 กับ 6 ก็คือรูปในเงาสะท้อนนั้นเอง ซึ่งเอนจินจะจัดการรูปที่เป็นเสมือนเงาสะท้อนให้ เราจึงเตรียมอนิเมชั่นเพียง 5 ด้านก็เพียงพอแล้ว คือ Front, Front-side, Back, Back-side และ Back

ในการแก้ไขอนิเมชั่นใน World Builder ให้เราไปที่แทบที่อยู่ด้านซ้ายของจอ เลือกที่แทบบิตแมป แล้วคลิกขวาที่รูปที่เราต้องการ เลือก Edit Animation โปรแกรมจะแสดงหน้าจอดังรูป 10-4 ขึ้นมา



รูปที่ 10-4 หน้าจอ Edit Animation

เราสามารถใช้บิตแมปมาทำเป็นอนิเมชั่นได้มากตามที่เราต้องการแต่มีข้อยกเว้น คือ เราไม่สามารถเซทจำนวนบิตแมปที่เป็นอนิเมชั่นไม่เท่ากันในแต่กลุ่ม ยกเว้นแต่ว่ากลุ่มที่ไม่เท่ากับกลุ่มอื่นไม่มีอนิเมชั่นเลยส่วนเวลาที่จะแสดงผลภาพนั้นนานเท่าไรสามารถตั้งค่าได้ที่ Times โดยค่า 100 เท่ากับ 1 วินาที หลังจากที่เราแก้ไขเสร็จแล้วให้กดที่ปุ่ม OK เพื่อบันทึกสิ่งที่เราได้แก้ไขไป

ในการสร้างอนิเมชั่นขึ้นมาใหม่ให้เราคลิกขวาที่บริเวณใดก็ได้ในแทบบิตแมปแล้วเลือก Add Animation เลือกด้านที่ต้องการจากทั้ง 5 ด้าน แล้วคลิกที่ Insert After หลังจากที่เราได้ใส่อนิเมชั่นในด้านที่ต้องการทั้งหมดแล้ว ให้ตั้งชื่ออนิเมชั่นแล้วกอ OK เพื่อบันทึก

10.6 Bitmap API

การโหลดบิตแมป

การที่เราจะใช้บิตแมปได้ต้องโหลดมันขึ้นมาจากดิสก์ก่อนโดยใช้ฟังก์ชัน

DWORD Morfit_bitmap_load(char *file_name, int transparent_index);

ฟังก์ชันนี้จะส่งค่าแฮนเดิลของบิตแมปที่โหลดขึ้นมา

ชื่อไฟล์ที่เราจะแทนใน file_name ไม่ต้องใส่นามสกุล แต่ถ้าเราใส่เอนจินจะไม่สนใจอยู่ดี ในขั้นตอนการโหลดเอนจินจะหาไฟล์ที่มีนามสกุลเป็น JBM, BMP และ JPEG ตามลำดับ เพราะฉะนั้นถ้าคุณมีรูปชื่อเดียวกัน 2 รูป ที่มีนามสกุลเป็น BMP และ JPEG เอนจินจะโหลดรูป BMP ก่อนเสมอ ส่วนตำแหน่งของไฟล์เราสามารถใช้ตำแหน่งเต็ม เช่น "c:\windows\bitmaps\mypicture" หรือ ตำแหน่งอ้างอิงก็ได้ เช่น "bitmaps\mypictures" ส่วนค่า *transparent_index* จะอนุญาติให้เราใช้สีใสกับรูปที่โหลดขึ้นมาได้ โดยถ้ามีค่าเป็น –1 หมายถึงไม่ใช้สีใส ส่วนถ้ามีค่าตั้งแต่ 0 ขึ้นไป หมายถึง ใช้สีใสโดยกำหนดให้สีใสเป็นสีใสที่ใช้บ่อยๆ เป็นอันดับที่ 1 เป็นต้นไป ถ้าเราโหลดภาพเดียวกันแต่ใช้ *transparent_index* คนละค่า เอนจินจะมองว่าเป็นภาพคนละภาพกัน และ จะส่งค่าแฮนเดิลค่ากลับมา 2 ค่า

เมื่อเราโหลดบิตแมปขึ้นมาแล้วครั้งหนึ่ง เราสามารถนำไปใช้กับวัตถุอื่นได้โดยไม่ต้องโหลดซ้ำ ให้ใช้ฟังก์ชัน Morfit_bitmap_get_first_bitmap() และ Morfit_bitmap_get_next(DWORD bitmap_handle) หรือ DWORD Morfit_bitmap_get_handle(char *file_name, int transparent_index) ในการอ่านค่าแฮนเดิลบิตแมปโดยกำหนดชื่อที่เราโหลดขึ้นมาแล้ว

ส่วนการบันทึกบิตแมปที่เราแก้ไขไปทำได้โดยใช้ฟังก์ชัน

Morfit_bitmap_save(DWORD bitmap_handle, char *full_path_name);

ในฟังก์ชันนี้เราต้องระบุนามสกุลของไฟล์ที่จะบันทึกด้วย ดังตัวอย่าง

Morfit_bitmap_save(my_bitmap_handle, "c:\\John\\bitmaps\\grass.bmp");

ตามตัวอย่างจะบันทึกบิตแมปเป็นไฟล์แบบ BMP และถ้า full_path_name เป็น NULL จะบันทึกไฟล์ที่ ที่โหลดไฟล์นั้นขึ้นมา (อ้างอิงจากค่าแฮนเดิล)

สำหรับการอันโหลดบิตแมปออกจากหน่วยความจำให้ใช้

Morfit_bitmap_unload(DWORD bitmap_handle);

สีของบิตแมป

ในเรื่องที่ผ่านมาเราจะเห็นว่าค่า 0 ใน *transparent_index* หมายถึง สีที่ใช้บ่อยมากเป็นอันดับ 1 ถ้าเป็น 1 ก็จะหมายถึง สีที่ใช้บ่อยมากเป็นอันดับ 2 เราสามารถตรวจสอบว่าสีที่ใช้เป็นสีอะไรได้โดยใช้ฟังก์ชัน

int Morfit_bitmap_index_to_rgb(DWORD bitmap_handle, BYTE index ,BYTE *red, BYTE *green, BYTE *blue);

ฟังก์ชันนี้จะส่งค่า RGB กลับมา โดยให้คุณกำหนดค่า *transparent_index* ที่ต้องการทราบ ตัวอย่างเช่น Morfit_bitmap_index_to_rgb(my_bitmap, 0, &red, &green, &blue) ฟังก์ชันนี้จะส่งกลับค่าสี RBG ของสีที่ใช้บ่อยมากเป็นอันดับ 1

นอกจากนี้เรายังสามารถตรวจสอบได้ว่าสี RGB ที่เรากำหนดมีค่า index_color เป็นเท่าไร โดยใช้ฟังก์ชัน

int Morfit_bitmap_rgb_color_to_index(DWORD bitmap_handle, BYTE red, BYTE green, BYTE blue);

หากเราโหลดบิตแมปขึ้นมาแล้วต้องการให้สีแดงสด (R=255, G=0, B=0) เป็นสีใส แต่ก็ไม่ทราบว่าค่า *transparent_index* เป็นเท่าใด ให้เราทำดังตัวอย่างนี้

```
DWORD temp_handle = Morfit_bitmap_load("picture",-1);
int red_index = Morfit_rgb_color_to_index(temp_handle, 255, 0, 0);
DWORD my_bitmap = Morfit_bitmap_load("picture", red_index);
Morfit_bitmap_unload(temp_handle);
```

ส่วนการตรวจสอบว่าบิทแมปรูปนี้ใช้สีใสอะไรให้ใช้

int Morfit_bitmap_get_transparent_rgb(DWORD bitmap_handle, int *red, int *green, int *blue) ;

ฟังก์ชันนี้จะส่งกลับค่า VR_ERROR ถ้าบิตแมปไม่มีการให้สีใส ส่วนการตรวจค่า *transparent_index* ให้ใช้

int Morfit_bitmap_get_transparent_index(DWORD bitmap_handle);

เปลี่ยนขนาดบิตแมป

ฟังก์ชันนี้จะสร้างบิตแมปขึ้นมาใหม่จากบิตแมปที่เรากำหนดโดยจะส่งค่าแฮนเดิลของบิตแมปใหม่กลับมา

DWORD Morfit_bitmap_resample(DWORD source_bitmap_handle, int new_width, int new_height);

เราสามารถสร้างสำเนาของบิตแมปได้โดยใช้ค่า new_width และ new_height เป็นค่าเก่า ส่วนการตรวจสอบค่าทำได้โดยใช้ฟังก์ชัน

```
int Morfit_bitmap_get_width(DWORD bitmap_handle);
int Morfit_bitmap_get_height(DWORD bitmap_handle);
```

ฟังก์ชันบิตแมปอื่นๆ

ฟังก์ชันนี้ใช้ตรวจสอบจำนวนโพลีกอนที่บิตแมปวางอยู่

int Morfit_bitmap_get_number_of_polygons_using_bitmap(DWORD bitmap_handle);

ส่วนการวางบิตแมปลงบนโพลีกอนเราจะกล่าวถึงในบทถัดไป

BYTE * Morfit_bitmap_get_memory(DWORD bitmap_handle);

ฟังก์ชันนี้จะรับค่าแฮนเดิลบิตแมป และ ส่ง BYTE array of size *width*height* กลับมา ทำให้เราแก้ไขพิกเซลแต่ละพิกเซลของบิตแมปได้ ข้อมูลของบิตแมปถูกเก็บเป็นแอร์เรย์ของแถวโดยมีจุดเริ่มที่ด้านบนซ้าย

ของรูป การหาตำแหน่งพิกเซลที่ต้องการทำได้โดยใช้สมการ *array*[*width* * (row-1) + column] โดยค่า BYTE ที่ส่งกลับมาที่มีค่า 0 จะหมายถึงสีที่ใช้มากที่สุด ส่วน 255 จะหมายถึงสีใสถ้ามีการใช้สีใส

ค่าของ index color จะถูกแปลงเป็นสี RGB จริง ผ่าน Palette โดย Palette จะเก็บค่าสี RGB ที่ค่า index color อ้างถึง เราสามารถเปลี่ยน Palette โดยใช้ฟังก์ชัน

int Morfit_bitmap_set_palette_color(DWORD bitmap_handle, BYTE index_of_color_to_be_changed, BYTE red, BYTE green, BYTE blue);

การตรวจว่าจำนวนสีทั้งหมดที่ใช้ใน Palette มีเท่าไรทำได้โดย

int Morfit_bitmap_get_palette_size(DWORD bitmap_handle);

การตรวจสอบว่าค่าอินเด็กซ์ของค่า RGB ที่เราระบุทำได้โดย

int Morfit_get_palette_index(DWORD bitmap_handle, BYTE red, BYTE green, BYTE blue);

ฟังก์ชันนี้จะส่งค่าอินเด็กซ์ที่มีสีใกล้เคียงกับค่า RGB ที่เราระบุให้มากที่สุด

เราสามารถแก้ไขบิตแมปได้ โดยแก้ไขอาร์เรย์ของบิตแมป ยกตัวอย่างเช่น การเขียนเส้นสีเขียวบนแถวบนสุดทำได้ดังนี้

```
BYTE green_index;
Morfit_bitmap_get_palette_color(my_bitmap, green_index, 0, 255, 0);
BYTE *bmp_ptr=Morfit_bitmap_get_memory(my_bitmap);
int width =Morfit_bitmap_get_width();
for(int i=0; i<width; i++)
        bmp_ptr[i]=green_index;
```

ข้อควรระวัง : การแก้ไข Palettes และ บิตแมปจะทำได้ในโหมดซอฟท์แวร์เรนเดอร์เท่านั้น

อนิเมชั่น

การสร้างอนิเมชั่น

ทำได้โดยใช้ DWORD Morfit_animation_create(char *name);

ฟังก์ชันจะสร้างอนิเมชั่นขึ้นมาใหม่ และ ส่งค่าแฮนเดิลกลับมาให้ และ เหมือนกับวัตถุอื่นใน Morfit ที่มีคำสั่งอ่านค่าแฮนเดิลวัตถุเริ่มต้น และ ถัดไป คือ Morfit_animation_get_first_anima() และ Morfit_animation_get_next(DWORD animation_handle)

char * Morfit_animation_get_name(DWORD animation_handle) ใช้ตรวจสอบชื่อของอนิเมชั่น

Morfit_animation_set_name(DWORD animation_handle, char * name) ใช้แก้ไขชื่อของอนิเมชั่น

DWORD Morfit_animation_get_handle(char *animation_name); ใช้อ่านค่าแฮนเดิลของอนิเมชั่นที่มีชื่อตามที่ระบุ

Morfit_animation_delete(DWORD animation_handle); ใช้สำหรับลบอนิเมชั่นออกจากหน่วยความจำ

การเพิ่มบิตแมป

หลังจากที่เราสร้างอนิเมชั่นขึ้นมา เราก็ต้องใส่บิตแมปให้เซตทั้ง 5 ซึ่งเคยกล่าวไว้ในตอนแรก

int Morfit_animation_add_bitmap(DWORD animation_handle, int bitmap_list ,DWORD bitmap_handle, int bitmap_position_index);

animation_handle คือ แฮนเดิลของอนิเมชั่นที่จะใส่บิตแมปเข้าไป

bitmap_list คือค่าที่อ้างถึงเซต 5 เซตดังนี้ คือ

BITMAP_LIST_FRONT

BITMAP_LIST_FRONT_SIDED

BITMAP_LIST_SIDE

BITMAP_LIST_BACK_SIDED

BITMAP_LIST_BACK

bitmap_handle แฮนเดิลของบิตแมปที่จะใส่เข้าไป

bitmap_position_index ค่าที่ระบุว่าเป็นลำดับขั้นการแสดงภาพที่เท่าไร 0 หมายถึงภาพที่จะแสดงผลในตอนเริ่มแสดง, 1 หมายถึงภาพที่จะแสดงผลถัดมาจากภาพที่ 0 ค่า –1 หมายถึงหมายถึงภาพบิตแมปสุดท้ายในลิตส์ ฟังก์ชันนี้จะส่งกลับค่าลำดับขั้นการแสดงภาพจริงที่กลับมา ถ้าค่าเป็น –1 หมายถึง ลำดับที่เราระบุมากเกินไป หรือ อื่นๆ

ถ้าค่า *bitmap_position_index* เป็น 5 แต่มีรูปในเซตแค่ 2 รูป หลังจากที่คุณวางรูปที่ 3 หลังรูปที่สอง ฟังก์ชันจะส่งกลับค่า 2

การจะลบบิตแมปออกจากอนิเมชั่นให้ใช้

int Morfit_animation_remove_bitmap(DWORD animation_handle, int bitmap_list , int bitmap_position_index);

bitmap_postion_index หมายถึงลำดับของบิตแมปที่จะลบ โดย –1 หมายถึงลำดับท้ายสุด

การตรวจสอบค่าแฮนเดิลของบิตแมป ณ ตำแหน่งที่กำหนด

DWORD Morfit_animation_get_bitmap(DWORD animation_handle, int bitmap_list ,int bitmap_position_index);

การตรวจสอบว่าในอนิเมชั่นมีบิตแมปเท่าไรให้ใช้

int Morfit_animation_get_number_of_frames(DWORD animation_handle);

int Morfit_animation_get_number_of_bitmaps(DWORD animation_handle);

ฟังก์ชันแรกจะส่งจำนวนเฟรม (จำนวนบิตแมปในเซต) กลับ ส่วนฟังก์ชันที่สองจะส่งจำนวนบิตแมปทั้งหมดในอนิเมชั่นกลับ ยกตัวอย่าง เช่น ถ้าเรามีอยู่ 4 บิตแมปใน FRONT, SIDE, และ BACK แต่ไม่มีเลยใน FRONT_SIDED และ BACK_SIDED ฟังก์ชันแรกจะส่งกลับค่า 4 ฟังก์ชันที่สองจะส่งกลับค่า 12

ฟังก์ชันนี้จะเขียนค่าแฮนเดิลของบิตแมปทั้งหมดในอนิเมชั่นลงอาร์เรย์

```
Morfit_animation_get_all_bitmaps(DWORD animation_handle, DWORD bitmaps_array[]);
```

นี่เป็นตัวอย่างการใช้งาน

```
int number = Morfit_animation_get_number_of_bitmaps(my_animation);
DWORD *handles = (DWORD *) malloc(number * sizeof(DWORD));
Morfit_animation_get_all_bitmaps(my_animation, number);
```

(ใน C++, ให้เปลี่ยนบรรทัดที่มี malloc เป็น: DWORD *handles = new DWORD[number];)

ระยะเวลาของแต่ละเฟรม

เฟรมไทม์ (Frame time) หมายถึงเวลาที่จะแสดงผลบิตแมปนั้นค้างไว้ก่อนจะแสดงผลบิตแมปถัดไป หน่วยที่ใช้ในเอนจินมอร์ฟิทคือ 1/100 วินาที การกำหนดเฟรมไทม์จะใช้ฟังก์ชัน

```
int Morfit_animation_set_frame_time(DWORD animation_handle, int frame_index, DWORD time);
```

ยกตัวอย่างเช่น Morfit_animation_set_frame_time(my_animation, 0, 200) จะกำหนดให้แสดงบิตแมปแรกค้างไว้ 2 วินาทีก่อนจะเปลี่ยนไปบิตแมปถัดไป

ส่วนการตรวจสอบค่าเฟรมไทม์ของบิตแมปให้ใช้

```
DWORD Morfit_animation_get_frame_time(DWORD animation_handle, int frame_index);
```

เราสามารถตั้งค่าเฟรมทั้งหมดได้โดยสร้างอาร์เรย์ที่เก็บค่าเฟรมไทม์ แล้วใช้ฟังก์ชัน

```
int Morfit_animation_set_times(DWORD animation_handle, DWORD time_for_each_frame[], int size_of_array);
```

ตัวอย่างเช่น

```
DWORD times[3];
times[0]=300;
times[1]=200;
times[2]=200;
Morfit_animation_set_times(my_animation, times, 3);
```

ส่วนการอ่านค่าเฟรมไทม์ทั้งหมดให้สร้างอาร์เรย์ไว้ก่อนแล้วค่อยใช้ฟังก์ชัน

int Morfit_animation_get_times(DWORD animation_handle, DWORD time_for_each_frame[ ], int size_of_array);

นอกจากนั้นเรายังสามารถกำหนดสเกลของเฟรมไทม์ได้โดยใช้ฟังก์ชัน

Morfit_animation_factor_speed(DWORD animation_handle, double factor);

ตัวอย่างเช่น ถ้าอนิเมชั่นมีบิตแมปอยู่ 3 เฟรม โดยมีค่าเฟรมไทม์ 40, 4, 2 ตามลำดับ ถ้าเราใช้เรียกฟังก์ชันดังนี้

Morfit_animation_factor_speed(my_animation, 2);

ค่าเฟรมไทม์ก็จะเป็น 80, 8, 2 แต่ถ้าเราเรียกฟังก์ชันเป็น

Morfit_animation_factor_speed(my_animation, 0.5);

ค่าเฟรมไทม์ก็จะเป็น 20, 2, 1

ฟังก์ชันที่เกี่ยวกับอนิเมชั่นอื่นๆ

ฟังก์ชันนี้จะส่งกลับค่า True เมื่อ อนิเมชั่นถูกใช้ในโลกที่โหลดไว้ ถ้าไม่ก็จะส่งกลับค่า False

int Morfit_animation_is_part_of_the_last_world(DWORD animation_handle);

การตรวจสอบจำนวนโพลีกอนที่วางอนิเมชั่นนั้นลงไปใช้

int Morfit_animation_get_number_of_polygons_with_animation(DWORD animation_handle);

บทที่ 11

Polygon API

11.1 รู้จักกับโพลีกอน

โพลีกอนเป็นส่วนประกอบที่สำคัญในแต่ละวัตถุ โดยวัตถุจะประกอบด้วยโพลีกอนจำนวนมากต่อเข้าด้วยกันกลายเป็นรูปทรงขึ้นมา ซึ่งโพลีกอนแต่ละชิ้นก็คือกลุ่มของเส้นตรงที่เชื่อมต่อกันและอยู่บนระนาบเดียวกัน โดยจะต้องต่อกันเป็นวงปิดพอดี และแต่มุมภายในแต่ละมุมจะต้องมีค่าน้อยกว่า 180 องศา

11.2 การสร้างโพลีกอน

มีการทำงานที่สำคัญ ๆ ดังนี้

- การสร้างโพลีกอนใหม่ ทำได้โดย

```
DWORD Morfit_polygon_create(void);
```

ฟังก์ชันนี้จะสร้างโพลีกอนเปล่าและคืนค่าแฮนเดิลของโพลีกอนมา ซึ่งจะยังไม่มีจุดเลย และยังไม่ถูกรวมไว้ใน world ซึ่งเราสามารถรวมโพลีกอนนั้นเข้าไปใน world ได้โดย

```
Morfit_engine_add_polygon()
```

- การหาค่าแฮนเดิลของโพลีกอน ทำได้โดย

```
DWORD Morfit_polygon_get_handle_using_name(char *polygon_name);
```

- การหาค่าแฮนเดิลของโพลีกอนไปเรื่อยใน world ทำได้โดย

```
DWORD Morfit_polygon_get_first_polygon(void);
DWORD Morfit_polygon_get_next(DWORD polygon_handle);
```

- การกำหนดและการหาชื่อของโพลีกอน ทำได้โดย

```
int Morfit_polygon_set_name(DWORD poly_handle, char *polygon_name);
char * Morfit_polygon_get_name(DWORD poly_handle);
```

- การ copy โพลีกอน ทำได้โดย

```
DWORD Morfit_polygon_duplicate(DWORD polygon_handle);
```

- การลบโพลีกอน ทำได้โดย

```
void Morfit_polygon_delete(DWORD polygon_handle);
```

- การรวมโพลีกอน 2 ชิ้นเพื่อสร้างเป็นโพลีกอนใหม่ที่ใหญ่ขึ้น ทำได้โดย

```
DWORD Morfit_polygon_join(DWORD polygon1_handle, DWORD polygon2_handle);
```

ถ้าฟังก์ชันนี้ทำงานสำเร็จมันจะลบแฮนเดิลของโพลีกอนเดิมออกไปทั้งสองตัว แล้วคืนค่าแฮนเดิลของโพลีกอนใหม่กลับมา แต่ถ้าทำไม่สำเร็จจะคืนค่า NULL และจะไม่มีการเปลี่ยนแปลงใด ๆ กับโพลีกอนทั้งสองตัวนั้น

โพลีกอนจะมีการเก็บค่าจุดแต่ละจุดของมัน ซึ่งจะเชื่อมถึงกันด้วยเส้นตรงโดยจุดสุดท้ายจะเชื่อมต่อกับจุดแรก การทำงานที่สำคัญ ๆ กับโครงสร้างข้อมูลของโพลีกอนนี้ได้แก่

- การเพิ่มจุดไปยังท้าย list ของโพลีกอน ทำได้โดย

```
DWORD Morfit_polygon_add_point(DWORD polygon_handle, double xyz_bmX_bmY_brightness
[6]);
```

Array xyz_bmX_bmY_brightness ประกอบด้วย

xyz_bmX_bmY_brightness[0] = ค่าในแนวแกน x ของจุด

xyz_bmX_bmY_brightness[1] = ค่าในแนวแกน y ของจุด

xyz_bmX_bmY_brightness[2] = ค่าในแนวแกน z ของจุด

xyz_bmX_bmY_brightness[3] = ค่าในแนวแกน x ของจุดที่จะแปะบิตแมปลงไป

xyz_bmX_bmY_brightness[4] = ค่าในแนวแกน y ของจุดที่จะแปะบิตแมปลงไป

xyz_bmX_bmY_brightness[5] = ค่าความสว่างของจุด

ตย. ในการสร้างโพลีกอนและเพิ่มจุดเข้าไปที่ตำแหน่ง (10, 10, 10), (20,10,10), และ (20,20,20) โดยไม่ใช้บิตแมป:

```
DWORD polygon = Morfit_polygon_create();
double point1[6] = {10,10,10,0,0,0};
double point2[6] = {20,10,10,0,0,0};
double point3[6] = {20,20,20,0,0,0};
Morfit_polygon_add_point(polygon,point1);
Morfit_polygon_add_point(polygon,point2);
Morfit_polygon_add_point(polygon,point3);
```

- การเพิ่มจุดไปที่ต้น list ทำได้โดย

```
DWORD Morfit_polygon_add_point_to_head(DWORD polygon_handle, double
xyz_bmX_bmY_brightness[6]);
```

- การหาจำนวนจุดทั้งหมดใน list ทำได้โดย

```
int Morfit_polygon_get_num_of_points(DWORD polygon_handle);
```

- การลบจุดจากโพลีกอน ทำได้โดย

```
Morfit_polygon_delete_point(DWORD polygon_handle, DWORD point_handle);
```

- การตรวจสอบว่าโพลีกอนที่สร้างขึ้นนั้นถูกต้องสมบูรณ์หรือไม่ ทำได้โดย

```
int Morfit_polygon_is_valid(DWORD polygon_handle); // คืนค่า YES หรือ NO
```

แต่การตรวจสอบนี้จะทำอย่างอัตโนมัติเมื่อเราเรียกใช้ Morfit_engine_add_polygon() อยู่แล้ว

11.3 การหมุนและเลื่อนโพลีกอน

มีการทำงานที่สำคัญ ๆ ได้แก่

- การหมุนโพลีกอน ทำได้โดย

```
Morfit_polygon_rotate_x(DWORD polygon_handle, double degrees);
Morfit_polygon_rotate_y(DWORD polygon_handle, double degrees);
Morfit_polygon_rotate_z(DWORD polygon_handle, double degrees);
```

- การหมุนโพลีกอนให้มันตรงกับแนวของอีกโพลีกอนหนึ่ง ทำได้โดย

```
Morfit_polygon_rotate_to_match_polygon(DWORD polygon_handle, DWORD
reference_polygon_handle);
```

- การเลื่อนโพลีกอนโดยสัมพันธ์กับตำแหน่งปัจจุบันของมัน ทำได้โดย

```
Morfit_polygon_move(DWORD polygon_handle, double step[3]);
```

- การหาตำแหน่งกึ่งกลางของโพลีกอน ทำได้โดย

```
Morfit_polygon_get_location(DWORD polygon_handle, double location[3]);
```

** ใน viewer mode เท่านั้น ถ้าโพลีกอนเป็นส่วนหนึ่งของ dynamic object ตำแหน่งของมันจะอ้างอิงโดยสัมพันธ์กับจุดกึ่งกลางของ object นั้น อย่างเช่นถ้าฟังก์ชันนี้คืนค่า {0, 0, 0} แสดงว่าโพลีกอนนั้นอยู่ที่จุดกึ่งกลางของวัตถุไม่ใช่จุดกำเนิดของ world ดังนั้นในการหาตำแหน่งจริง ๆ เราจะต้องบวกตำแหน่งของวัตถุนั้นเข้าไปด้วย:

```
actual_x_location = object_x_location + pc :ygon_x_location
actual_y_location = object_y_location + polygon_y_location
actual_z_location = object_z_location + polygon_z_location
```

- การเลื่อนโพลีกอนไปยังจุดที่ต้องการ ทำได้โดย

```
Morfit_polygon_set_location(DWORD polygon_handle, double location[3]);
```

- การเลื่อนโพลีกอนโดยให้จุดใดจุดหนึ่งของโพลีกอนนั้นเลื่อนไปยังจุดที่เราต้องการ ทำได้โดย

```
Morfit_polygon_move_to_match_point(DWORD polygon_handle, DWORD
point_belongs_to_polygon, DWORD point_to_match);
```

11.4 คุณสมบัติของโพลีกอน

11.4.1 คุณสมบัติด้านเดียว โดยปกติโพลีกอนจะมีเพียงด้านเดียว ถ้าเรามองจากอีกด้านที่ไม่มีบิตแมปเราจะมองทะลุผ่านโพลีกอนนั้นไปเลย (World Builder จะสร้างโพลีกอนที่มีทั้ง 2 ด้านโดยการประกบโพลีกอน 2 อันเข้าด้วยกันโดยใช้ด้านหลังของโพลีกอนประกบกัน) มีการทำงานที่สำคัญ ๆ เกี่ยวกับคุณสมบัตินี้ดังนี้

- การสลับด้านของโพลีกอนที่เรามองเห็น ทำได้โดย

```
Morfit_polygon_flip_visible_side(DWORD polygon_handle);
```

- การสร้างโพลีกอนให้เป็นแบบที่มีทั้ง 2 ด้าน ทำได้ดังตัวอย่าง:

```
DWORD flipped = Morfit_polygon_duplicate(original); /* flipped อาจจะเป็นโพลีกอนที่มาจากวิธีอื่นก็
ได้*/
Morfit_polygon_flip_visible_side(flipped);
```

** ถ้าเราต้องการเลื่อนหรือหมุนโพลีกอนอันใดอันหนึ่งเราจะต้องหมุนอีกอันหนึ่งด้วย ดังนั้นเราจึงควรกรุ๊ปมันไว้ด้วยกันจะทำให้ง่ายขึ้น

11.4.2 คุณสมบัติการหมุน คือ "โพลีกอนหมุน" จะหมุนด้านที่กำหนดให้มองเห็นได้เข้าหา camera โดยอัตโนมัติ โดยเราควรใช้กับโพลีกอนที่เราต้องการให้มองเห็นทั้งด้านข้างและด้านหลังด้วยเช่น ต้นไม้ (ถ้าเราไม่กำหนดคุณสมบัติ "การหมุน (rotated)" ให้กับมันแล้วเมื่อเรามองโพลีกอนนี้จากด้านข้างหรือด้านหลัง เราจะมองไม่เห็นมันเลย) การกำหนดให้โพลีกอนหมุนทำได้สองวิธีโดยใน World Builder ทำได้โดยกำหนดจาก properties polygon ของวัตถุที่ check box "rotated" หรือกำหนดในโปรแกรมได้ดังนี้

```
Morfit_polygon_set_rotated(DWORD polygon_handle, int yes_no_flag);
int Morfit_polygon_is_rotated(DWORD polygon_handle); // returns YES or NO
```

11.4.3 คุณสมบัติอื่น ๆ มีการทำงานที่สำคัญ ๆ ได้แก่

- การกำหนดให้โพลีกอนอยู่ในการตรวจสอบการชนหรือไม่ มีดังนี้

```
Morfit_polygon_make_non_collisional(DWORD polygon_handle);
//ค่าปกติมอร์ฟิทจะกำหนดให้โพลีกอนอยู่ในการตรวจสอบการชนอยู่แล้ว//
Morfit_polygon_make_collisional(DWORD polygon_handle);
```

```
int Morfit_polygon_is_collisional(DWORD polygon_handle);
```

- การ enable และ disable โพลีกอน มีดังนี้

```
Morfit_polygon_disable(DWORD polygon_handle);
Morfit_polygon_enable(DWORD polygon_handle);
int Morfit_polygon_is_disabled(DWORD polygon_handle); // returns YES or NO
```

11.5 การเติมสี

- การกำหนดและหาสีของโพลีกอนทำได้โดย

```
Morfit_polygon_set_color_fill(DWORD polygon_handle, int red, int green, int blue);
Morfit_polygon_get_color(DWORD polygon_handle, int *red, int *green, int *blue);
```

red, *green*, and *blue* คือค่า RGB ของสีที่เราต้องการ

11.6 การแปะบิตแมปและภาพเคลื่อนไหว

- การแปะบิตแมปบนโพลีกอน ทำได้โดย

```
Morfit_polygon_set_bitmap_fill(DWORD polygon_handle, DWORD bitmap_handle);
```

- การหาชื่อและแฮนเดิลของบิตแมปที่แปะอยู่กับโพลีกอน ทำได้โดย

```
char * Morfit_polygon_get_bitmap_name(DWORD polygon_handle);
DWORD Morfit_polygon_get_bitmap_handle(DWORD polygon_handle);
```

** จะคืนค่า NULL ถ้าไม่มีบิตแมปแปะอยู่

11.6.1 สีและความสว่างของบิตแมป มีการทำงานที่สำคัญ ๆ ดังนี้

- การใช้สีโปร่งใสของบิตแมป (ดูบทที่ 8 ประกอบ) ทำได้โดย

```
Morfit_polygon_get_transparent_rgb(DWORD polygon_handle, int *red, int *green, int *blue);
int Morfit_polygon_get_transparent_index(DWORD polygon_handle);
```

- การกำหนดความสว่างของบิตแมป จะมีประโยชน์มากโดยเฉพาะงานที่เราต้องการแปะบิตแมปไปที่ฌพลีกอนอื่นโดยให้มีความสว่างต่างกัน เราไม่จำเป็นต้องสร้างบิตแมปขึ้นใหม่เพียงแค่แปะบิตแมปนั้นลงไปแล้วกำหนดค่าความสว่างใหม่เท่านั้น โดย:

```
Morfit_polygon_set_brightness(DWORD polygon_handle, int value);
```

** *value* เป็นค่าความสว่างใหม่เมื่อเทียบกับความสว่างเดิม

value = 100: ความสว่างเท่าเดิม

value = 200: ความสว่างเป็น 2 เท่าของเดิม

value = 50: ความสว่างเป็นครึ่งหนึ่ง

- การตรวจสอบว่าความสว่างของทุกจุดใน list ของโพลีกอนเท่ากันหรือไม่ ทำได้โดย

```
int Morfit_polygon_is_uniform_brightness(DWORD polygon_handle);
```

ถ้าแต่ละจุดมีค่าความสว่างไม่เท่ากัน บิตแมปจะสว่างขึ้นหรือมืดลงเมื่อเข้าใกล้แต่ละจุด ทำให้การ render ช้าลงใน mode software render

- การหาค่าความสว่างเฉลี่ยของโพลีกอน ทำได้โดย

```
double Morfit_polygon_get_brightness(DWORD polygon_handle);
```

- การกำหนดและหาค่า light diminution ของโพลีกอน โดยค่านี้จะมีผลต่อ fog กับโพลีกอน ซึ่งค่ามากจะทำให้มองเห็นได้ยากมากใน fog เช่นก้อนหินตอนกลางคืน แต่ค่าน้อยจะทำให้มองเห็นได้ง่าย เช่นกองไฟตอนกลางคืน เป็นต้น:

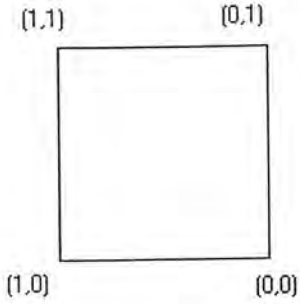
```
double Morfit_polygon_get_light_diminution(DWORD polygon_handle);
Morfit_polygon_set_light_diminution(DWORD polygon_handle, double value);
```

11.6.2 การพลิกบิตแมป บิตแมปจะต้องเป็น 4 เหลี่ยมมุมฉากแต่โพลีกอนนั้นไม่จำเป็น โดยมอร์ฟิทจะใช้พิกัดบิตแมป (bitmap coordinates) ในการระบุว่าบิตแมปจะถูกแปะลงไปอย่างไร โดยทุก ๆ บิตแมปซึ่งเป็น 4 มุมฉากจะมีมุมบนซ้ายเป็นจุด (0, 0) และล่างซ้ายเป็นจุด (1, 1) ถ้ายังจำกันได้ ฟังก์ชัน Morfit_polygon_add_point() array ที่เป็นพารามิเตอร์ตัวที่สองจะมีการเก็บค่าพิกัดของจุดในโพลีกอนและพิกัดบิตแมปของจุดนั้นด้วยซึ่งจะระบุว่าจะแปะบิตแมปที่พิกัดใดลงไปบนจุดนั้น

ตัวอย่างเช่นถ้าเรามีบิตแมปดังรูป:

รูปที่ 11-1 บิตแมปตัวอย่าง

และเรามีโพลีกอนซึ่งระบุพิกัดบิตแมปของจุดยอดไว้ดังรูป:



รูปที่ 11-2 จุดยอดของบิตแมป

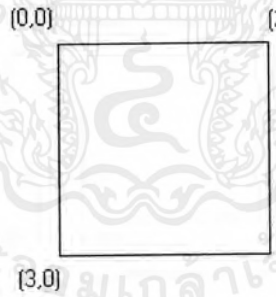
แล้วบิตแมปจะถูกแปะลงไปดังรูป:

รูปที่ 11-3 การแปะบิตแมปตามจุดยอดที่กำหนด

**จะเห็นด้ายว่ามุมซ้ายบนของบิตแมปจะถูกแปะลงไปที่ (0, 0) ส่วนมุมขวาล่างที่จุด (1, 1)

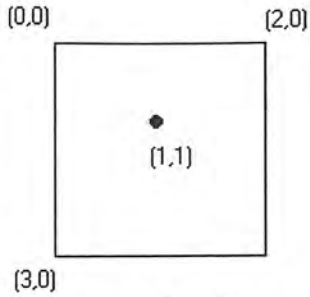
**เราเพียงกำหนดค่าพิกัดไป 3 จุดแล้ว engine จะคำนวณจุดที่เหลือให้เอง ถ้าเราต้องการให้ engine คำนวณพิกัดบิตแมปของจุดยอดให้ด้วยก็ให้ใช้พิกัด (-1 ,-1)

เราสามารถปูบิตแมปไปตามขวางของโพลีกอนได้ด้วยวิธีดังนี้:



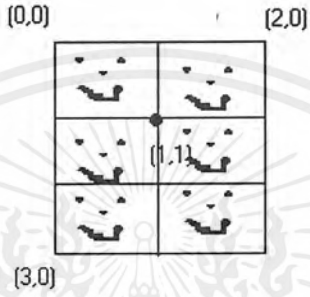
รูปที่ 11-4 การแปะบิตแมปโดยใช้จุดยอด 3 จุด

แล้ว engine จะคำนวณจุด (1, 1):



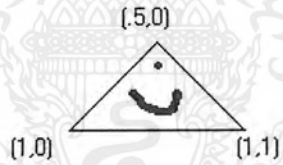
รูปที่ 11-5 จุดยอดที่เอนจิ้นคำนวณให้

จากนั้นจะปูบิตแมปไปดังนี้:



รูปที่ 11-6 การแปะบิตแมปเรียงกันตามจุดยอด

และเรายังสามารถวางบิตแมปลงไปบนโพลีกอนที่ไม่เป็น 4 เหลี่ยมได้ดังนี้:



รูปที่ 11-7 การแปะบิตแมปบนโพลีกอนที่ไม่เป็น 4 เหลี่ยม

- ในการหมุนบิตแมป ทำได้โดย

```
Morfit_polygon_rotate_bitmap(DWORD polygon_handle);
```

จะเป็นการหมุนพิกัดบิตแมปของแต่ละจุดยอดในโพลีกอน

- ในการพลิกบิตแมปไป 180 องศา ทำได้โดย

```
Morfit_polygon_mirror_horizontal_bitmap(DWORD polygon_handle);
Morfit_polygon_mirror_vertical_bitmap(DWORD polygon_handle);
```

- ในการแปะบิตแมปไปยังโพลีกอนถัดไปที่อยู่ติดกันไปเรื่อย ๆ ทำได้โดย

```
Morfit_polygon_match_bitmap_cords(DWORD source_polygon_handle, DWORD
target_polygon_handle);
```

11.6.3 ภาพเคลื่อนไหว

- การติดภาพเคลื่อนไหวไปกับโพลีกอน ทำได้โดย

```
Morfit_polygon_set_animation(DWORD polygon_handle, DWORD animation_handle);
```

- การหาแฮนเดิลของภาพเคลื่อนไหวที่ติดกับโพลีกอน (คืนค่า NULL ถ้าไม่มีภาพเคลื่อนไหว) ทำได้โดย

```
DWORD Morfit_polygon_get_animation(DWORD polygon_handle);
```

- การกำหนดว่าจะแสดงเฟรมใดในขณะนั้น ทำได้โดย

```
Morfit_polygon_set_animation_frame(DWORD polygon_handle, int animation_frame);
```

0 จะเป็นเฟรมแรก 1 เป็นเฟรมที่สอง... และ -1 เป็นเฟรมสุดท้าย

- การหาเฟรมปัจจุบันของภาพเคลื่อนไหว ทำได้โดย

```
int Morfit_polygon_get_animation_frame(DWORD polygon_handle);
```

11.6.4 ความโปร่งแสง (Translucency)

โพลีกอนที่อยู่ไกลมาก เราจะมองเห็นเป็นสีทึบ โดยเราอาจจะเห็นสี, บิตแมป หรือภาพเคลื่อนไหว โพลีกอนสามารถเป็นได้ทั้งโปร่งใสหรือโปร่งแสง (นั่นคือปล่อยให้แสงผ่านได้หมดหรือเพียงบางส่วน) อย่างเช่น กระจกของหน้าต่างซึ่งเราสามารถมองเห็นบานหน้าต่างเล็กน้อย แต่เราจะเห็นสิ่งที่อยู่ข้างหลังมัน

- การสร้างโพลีกอนที่โปร่งแสง (ทำงานใน mode hardware render เท่านั้น) ทำได้โดย

```
Morfit_polygon_set_translucent(DWORD polygon_handle, BYTE glass_type);
```

glass_type จะเป็นค่าคงที่ได้ดังต่อไปนี้:

DISABLE_BLENDING

DARK_FILTER_GLASS

FILTER_GLASS

OPAQUE_FILTER_GLASS

NORMAL_GLASS

CONTROL_GLASS

BRIGHT_TRANSPARENT_GLASS

BRIGHT_TRANSLUCENT_GLASS

BRIGHT_OPAQUE_GLASS

LIGHT_SOURCE_GLASS

INVERSE_GLASS

**ดูรายละเอียดเพิ่มเติมในไฟล์ mrft_api.h

- การหา glass type ของวัตถุ ทำได้โดย

```
BYTE Morfit_polygon_get_translucent(DWORD polygon_handle);
```

11.7 การทำงานทางคณิตศาสตร์ของโพลีกอน

- แต่ละจุดในโพลีกอนจะต้องอยู่บนระนาบเดียวกัน เราสามารถหาสมการของระนาบนั้นได้โดย:

```
Morfit_polygon_get_plane(DWORD polygon_handle, double plane[4]);
```

ซึ่งสมการจะเป็น:

plane[0]*x + plane[1]*y + plane[2]*z + plane[3] = 0

และมีเวกเตอร์ปกติเป็น (plane[0], plane[1], plane[2])

- การตรวจสอบว่าจุด 3 มิติใด ๆ อยู่ในโพลีกอนหรือไม่ ทำได้โดย

```
int Morfit_polygon_is_point_inside_polygon(DWORD polygon_handle, double pnt[3]); // returns YES or NO
```

- การหาจุดยอดที่ใกล้เคียงที่สุดกับจุดที่กำหนด ทำได้โดย

```
DWORD Morfit_polygon_get_closest_point(DWORD polygon_handle, double p3d[3]);
```

มันจะคืนค่าแฮนเดิลของจุดยอดที่ใกล้ที่สุดมาให้

11.8 โพลีกอนและ Group

- การเพิ่มโพลีกอนเข้าไปในกรุ๊ปทำได้โดย

```
Morfit_polygon_set_group(DWORD polygon_handle, DWORD group_handle);
```

- การหากรุ๊ปที่เป็นเจ้าของโพลีกอนใด ๆ ทำได้โดย

```
DWORD Morfit_polygon_get_group(DWORD polygon_handle);
```

- การตรวจสอบว่ามีโพลีกอนใดอยู่ในกรุ๊ปหรือไม่ ทำได้โดย

```
int Morfit_polygon_is_in_group(DWORD polygon_handle, DWORD group_handle);
```

- การกำหนดค่า orientation ของโพลีกอน (ดู บทที่ 6 ประกอบ) ทำได้โดย

```
Morfit_polygon_set_orientation(DWORpolygon_handle, int orientation_value);
```

11.9 Patches

patches เป็นทางเดียวที่จะทำให้เราสร้างโพลีกอนขึ้นใหม่ได้ใน Viewer mode โดยจะเป็นโพลีกอนที่ติดอยู่บนโพลีกอนอื่น อย่างเช่น การสร้างรอยจากกระสุนบนกำแพง เป็นต้น

- การสร้าง patch แบบง่าย ๆ ทำได้โดย

```
DWORD Morfit_polygon_add_patch_easy(DWORD father_polygon, double center[3], double radius,
double direction[3], DWORD bitmap_handle, int rgb[3]);
```

father_polygon เป็นโพลีกอนที่เราจะติด patch ลงไป

center เป็นจุดกึ่งกลางของ patch

radius บอกว่า patch ควรจะมีขนาดเท่าไร (ในที่นี้เป็นรูป 4 เหลี่ยมจึงมักหมายถึงจาก จุดศูนย์กลางไปยังขอบ) *direction* บอกทิศทางด้านหน้าของ patches ใช้ในการจัดวางบิตแมป แต่ถ้าจะเป็นอย่างไรก็ได้ให้ใช้ {0,0,0}

bitmap_handle คือบิตแมปที่จะแปะบน patch แต่ถ้าใช้การเติมสีแทนให้ใช้ NULL

rgb ถ้า *bitmap_handle* เป็น NULL, *rgb* จะระบุสีที่จะเติมลงบน patch แต่ถ้าเราใช้บิตแมปแล้วค่านี้จะไม่นำมาคำนวณ

- การลบ patches ของโพลีกอน ทำได้โดย

```
Morfit_polygon_delete_patches(DWORD polygon_handle, int
type_of_patches_to_delete_or_serial_number);
```

พารามิเตอร์ตัวสุดท้ายจะมีค่าได้ดังต่อไปนี้:

ALL_PATCHES – ลบ patches ทั้งหมดของโพลีกอน

REGULAR_PATCHES – ลบ patches ทั้งหมดที่ถูกสร้างโดย Morfit_polygon_add_patch() หรือ Morfit_polygon_add_patch_easy()

SHADOW_PATCHES – ลบ shadow patches ทั้งหมด ดูรายละเอียดของ Shadow patches ได้ในบทที่ 11

บทที่ 12
API อื่นๆ

ในบทที่ผ่านๆ มา เราได้กล่าวถึง API ที่สำคัญของมอร์ฟิทแล้ว ในบทนี้เราจะกล่าวถึง API ที่สำคัญรองลงมา บาง API อาจจะไม่จำเป็นต้องใช้ในโปรแกรมของคุณก็ได้ ซึ่งเราจะกล่าวถึงอย่างคร่าวๆ ในรายละเอียดให้อ่านใน Morfit Extension Help หรือในเฮดเดอร์ไฟล์ชื่อ mrft_api.h

12.1 Point API

อย่างที่ทราบดีว่าวัตถุในโลก 3 มิติ ประกอบขึ้นมาจากโพลีกอน ซึ่งโพลีกอนก็ประกอบด้วยจุดอย่างน้อย 1 จุดเช่นกัน จุดเหล่านี้เป็นตัวกำหนดเวอร์เท็กซ์ของโพลีกอน โดยแต่ละจุดจะมีคุณสมบัติต่างๆ เช่น ตำแหน่ง, ความสว่าง และ โคออดิเนตของบิตแมปเป็นต้น Point API จะอนุญาติให้คุณตั้งค่าเหล่านี้ได้

12.2 Track API

วัตถุ, กล้อง และกรุ๊ปสามารถตั้งให้วิ่งไปตามแทร็กได้ แทร็กก็เป็นเพียงจุดบนแกน 3 มิติ หลายๆ จุด มาเชื่อมโยงกันเป็นทางเท่านั้นเอง Track API สามารถใช้สร้าง และ แก้ไขแทร็กได้

12.3 3D Card API

เราสามารถเร่งความเร็วของการแสดงผลได้ถ้าเครื่องที่รันโปรแกรมมีการ์ดเร่ง 3 มิติ นอกจากนี้การแสดงผลแอฟเฟคบางอย่างก็จำเป็นต้องใช้การ์ด 3 มิติเช่นกัน เราสามารถเปลี่ยนการแสดงผลระหว่างโหมดซอฟท์แวร์ และ ฮาร์ดแวร์ โดยกดปุ่ม Crtl+Alt+V หรือสั่งจากในโปรแกรมโดยใช้ฟังก์ชัน

```
int Morfit_3D_card_use(int YES_or_NO);
```

ก็ได้ ใน 3D Card API จะประกอบไปด้วย ฟังก์ชันที่ใช้ตรวจสอบว่ามีการ์ด 3 มิติหรือไม่ และ ฟังก์ชันที่ใช้เปลี่ยนการแสดงผลโดยใช้การ์ด 3 มิติ กับ ซอฟท์แวร์

12.4 Background API

ฉากหลัง (Background) ก็คือภาพบิทแมบธรรมดาที่ถูกนำมาแปะไว้รอบๆ โลก Background API ใช้สำหรับกำหนดระยะห่างของฉากหลัง, การแสดงผลในระยะห่างระหว่างกล้องกับฉากหลังที่ต่างๆ กัน, ฉากหลังที่จะแสดงเมื่อกล้องเงยขึ้น หรือ ลง, ความสูงของฉากหลังตั้งแต่พื้น รวมถึง ความสว่าง และ บรรยากาศ (ความหนาหมอก)

12.5 Math API

API นี้จะประกอบไปด้วยฟังก์ชันทางคณิตศาสตร์ต่างที่ใช้ในการคำนวณทาง 3 มิติ เช่น การหาระยะห่างระหว่างจุดสองจุด, หาจุดตัวระหว่างเส้นกับระนาบ 3 มิติ, ฟังก์ชันที่ใช้คำนวณหาสมการระนาบจากจุด 3 จุด และ สมการที่ใช้คำนวณหาแสงที่สะท้อนจากพื้นผิว

12.6 Utilities API

ส่วนมาก API นี้จะอำนวยความสะดวกเกี่ยวกับการจัดการกับภาพบิตแมป เช่น การโหลดภาพจากดิสก์, การเปลี่ยนจากภาพ BMP เป็น JPEG, เปลี่ยนภาพสี 16 บิต เป็น 24 บิต

12.7 Profiler API

Profiler API ประกอบด้วย 2 คำสั่งที่สำคัญ คือ MORFIT_START_TAKING_TIME(char * log_output _string), and MORFIT_STOP_TAKING_TIME ที่ทำให้เราสามารถตรวจสอบประสิทธิภาพของโปรแกรม โดยผลลัพธ์จะถูกบันทึกลงในๆ ล็อกไฟล์ (log files) ของ Morfit

บทที่ 13

แสง และเงา

เราจะใช้ patch ในการสร้างแสงและเงา เพราะในงานดังกล่าวไม่จำเป็นต้องมีการสร้างรูปทรงใหม่ขึ้นมา

13.1 เงา

การสร้างเงาต้องใช้วัตถุ 3 อย่างได้แก่ แหล่งกำเนิดแสง, โพลีกอนที่จะส่องเงาลงไป และโพลีกอนที่จะรับเงา ดังรูป:

รูปที่ 13-1 การเกิดเงา

เราสามารถสร้างเงาได้โดยใช้:

```
Morfit_engine_create_shadow(DWORD entity_receiving_shadow, DWORD entity_creating_shadow,
double light_src[3], BYTE serial_number);
```

ทั้ง *entity_receiving_shadow* และ *entity_creating_shadow* เป็นได้ทั้ง โพลีกอน, วัตถุหรือ กรุ๊ป และไม่จำเป็นต้องเป็นประเภทเดียวกันก็ได้ **โดยถ้า handle ตัวใดตัวหนึ่งเป็น NULL จะถือว่าเป็น world

light_src[3] เก็บตำแหน่งของแหล่งกำเนิดแสงตามพิกัด x, y, z และเนื่องจากใช้ในการสร้างเงาจึงควรเป็นแหล่งกำเนิดแบบจุด

serial_number มีค่าระหว่าง 10-255 ใช้ในการแบ่งกลุ่มของเงา เช่นถ้าเรามีวัตถุที่สร้างเงาอยู่ 2 วัตถุ โดยฉายเงาไปบนวัตถุเดียวกัน และวัตถุที่ฉายเงาอันหนึ่งเคลื่อนที่ เราไม่จำเป็นต้องสร้างเงาขึ้นมาใหม่ทั้งหมด เราจะสร้างเพียงเงาที่เกิดจากวัตถุที่เคลื่อนที่เท่านั้น ดังตัวอย่าง:

```
Morfit_engine_create_shadow(room_group, person_object, light, 255);
Morfit_engine_create_shadow(room_group, piano_group, light, 254);
.
.
.
// Person moves
Morfit_polygon_delete_patches(room_group,255);
```

```
Morfit_engine_create_shadow(room_group, person_object, light, 255);
```

เรายังสามารถใช้ serial number ในการเปลี่ยนสีหรือบิตแมปของแต่ละกลุ่มของ patch ได้อีกด้วย

เราสามารถลบเงาได้ 2วิธีโดย:

```
Morfit_polygon_delete_patches(DWORD polygon_handle, int
type_of_patches_to_delete_or_serial_number);
Morfit_group_delete_patches(DWORD group_handle, int
type_of_patches_to_delete_or_serial_number);
```

ฟังก์ชันแรกจะลบ patch ทั้งหมดจากโพลีกอนกลุ่มเดียว ฟังก์ชันหลังจะลบ patch ทั้งหมดจากทั้งกลุ่ม

13.1.1 สีของเงา

เงาจะสืบทอดสมบัติมาจากวัตถุที่ฉายเงาเช่นวัตถุโปร่งแสงจะให้เงาที่โปร่งแสง, คนจะให้เงาเป็นรูปบิตแมปของคน เป็นต้น ซึ่งในกรณีหลังเรามักไม่ต้องการ แต่เราสามารถกำหนดสีของเงาให้เป็นสีทึบเพื่อความถูกต้องได้โดย:

```
Morfit_group_set_patches_color(DWORD group_handle, int red, int green, int blue, int patches_type);
```

การเติมบิตแมปไปบนเงาทำได้โดย:

```
Morfit_group_set_patches_bitmap(DWORD group_handle, DWORD bitmap_handle, int patches_type);
```

เรามักต้องการให้เงามีสมบัติโปร่งแสง เพราะเงาเพียงแค่ทำให้วัตถุที่มันฉายลงไปมืดลงเท่านั้น ไม่ใช่บังจนเปลี่ยนเป็นสีดำทั้งหมด โดยเราสามารถเปลี่ยนค่าความโปร่งแสงของ patch ได้โดย:

```
Morfit_group_set_patches_translucent(DWORD group_handle, BYTE translucent_type, int
patches_type);
```

โดยจะเห็นการโปร่งแสงได้ใน hardware rendering เท่านั้น

*ค่า *patches_type* ในแต่ละฟังก์ชันที่กล่าวมาเป็นได้ทั้ง serial number, ALL_PATCHES, NORMAL_PATCHES หรือ SHADOW_PATCHES

13.1.2 Dynamic Patches

การสร้าง patches อาจเป็นงานที่ช้ามาก เราสามารถทำให้เร็วขึ้นได้โดยใช้โพลีกอนเป็นทั้งตัวสร้างเงาและตัวรับเงา แต่ในบางครั้งการทำเช่นนี้ก็ยังช้าเกินไป ดังนั้นสำหรับวัตถุและแหล่งกำเนิดแสงที่ไม่เคลื่อนไหว เราสามารถสร้างเงาเพียงครั้งเดียวในตอนเริ่มโปรแกรม

ส่วนการสร้างเงาจากวัตถุหรือแหล่งกำเนิดแสงที่เคลื่อนที่นั้น เราสามารถทำให้เร็วขึ้น โดยใช้:

```
Morfit_engine_create_dynamic_shadow(DWORD entity_receiving_shadow, DWORD
entity_creating_shadow, double light_src[3], BYTE serial_number);
```

จะเห็นว่าฟังก์ชันนี้เกือบเหมือนกับ Morfit_engine_create_shadow() แต่มีข้อดีคือเร็วกว่า โดยแลกกับข้อเสียเมื่อเราต้องการลบบาง patches เราต้องลบทั้งหมด ซึ่งทำได้โดย:

```
Morfit_engine_delete_dynamic_shadows(void);
```

13.2 แสง

แสงสร้างโดยวิธีการใกล้เคียงกับเงา โดยเทียบแสงได้กับ "เงาสว่าง" (ดูรูปด้านล่างประกอบ):

รูปที่ 13-2 การเกิดแสง

เราจะเห็นได้ว่าแหล่งกำเนิดฉายแสงไปบนโพลีกอนเป้าหมาย แต่เนื่องจากมอร์ฟีทเอนจิ้นใช้แหล่งกำเนิดแสงแบบจุด เราจะสร้างแสงได้ดังรูปที่ 13-3:

รูปที่ 13-3 แหล่งกำเนิดแสงแบบจุด

จะเห็นได้ว่าเหมือนกับการสร้างเงา โดยเราใช้ แหล่งกำเนิดแสงแบบจุด, โพลีกอนคั่นกลาง และโพลีกอนรับแสง ต่างกันเพียงแค่แสงจะสร้างเงาที่ทำให้วัตถุที่รับแสงสว่างขึ้นไม่ใช่มืดลง

การสร้างแสงทำได้โดยใช้ Morfit_engine_create_shadow() แต่ก่อนจะสร้างเราต้องกำหนดสีของ patches ที่สว่างขึ้น หรือใช้บิตแมปที่สว่าง และกำหนดให้ patches โปร่งแสงก่อน:

```
Morfit_engine_create_shadow(floor_polygon, light_polygon, light_point, serial_number);
```

```
Morfit_group_set_patches_color(floor_polygon, 200,200, 200, serial_number);
Morfit_group_set_patches_translucent(floor_polygon, NORMAL_GLASS, serial_number);
```

**แต่เนื่องจาก patches จะสืบทอดสมบัติมาจากโพลีกอนที่สร้างมัน แทนที่เราจะกำหนดสมบัติของ patches ทุกครั้งที่สร้างมัน เราเพียงแค่กำหนดสมบัติของโพลีกอนคั่นกลางครั้งเดียวเท่านั้น:

```
DWORD light_polygon = Morfit_polygon_get_handle_using_name("light");
Morfit_polygon_set_color(light_polygon, 200, 200, 200);
// OR, to use a bitmap instead of a solid color:
// DWORD bitmap = Morfit_bitmap_load("\\Bitmaps\\light", 0);
// Morfit_polygon_set_bitmap(light_polygon, bitmap);
Morfit_polygon_set_translucent(light_polygon, NORMAL_GLASS);
```

เมื่อเรากำหนดสมบัติต่าง ๆ ของ patches หรือโพลีกอนคั่นกลางดังข้างต้นแล้วเราสามารถสร้างแสงได้โดย:

```
Morfit_engine_create_shadow(floor_polygon, light_polygon, light_point, serial_number);
```

การกำหนดสมบัติทางแสงของโพลีกอนดังข้างต้นจะมีประโยชน์ในกรณีที่เราต้องสร้างแสงขึ้นใหม่บ่อยเช่น แหล่งกำเนิดแสงเคลื่อนที่ได้ เป็นต้น

*การสร้างแสงจะซับซ้อนกว่าเงาเล็กน้อยตรงที่เราต้องสร้างโพลีกอนคั่นกลาง ซึ่งทำได้โดยวางโพลีกอนคั่นระหว่างแหล่งกำเนิดแสงกับส่วนที่เหลือของโลก แล้วใช้ Morfit_polygon_disable() เพื่อทำให้มองไม่เห็นมัน

บทที่ 14
ข้อแตกต่างระหว่าง Editor mode และ Viewer mode

ในการแสดงผลโลก 3 มิติ เราจำเป็นต้องเลือกโหมดการแสดงผลระหว่าง Editor mode และ Viewer mode การเลือกโหมดการแสดงผลจะมีผลต่อความเร็ว และ ความสามารถในการจัดการวัตถุ 3 มิติ รวมถึงความสามารถที่จะใช้ API บางตัว เราจำเป็นต้องทราบข้อแตกต่างระหว่างโหมดการแสดงผลทั้งสองนี้เพื่อที่จะได้เลือกโหมดการแสดงผลที่เหมาะสมกับโปรแกรมของเรา

ในขั้นแรกเมื่อเราโหลดโลก 3 มิติขึ้นมาใน Viewer mode เอนจินจะสร้างดาต้าสตรักเจอร์ชื่อ BSP-Tree โดยขนาดของดาต้าสตรักเจอร์จะขึ้นอยู่กับจำนวนโพลีกอนที่มีอยู่ในโลก เนื่องจากในโลกที่มีโพลีกอนมากๆ บางครั้งการสร้าง BSP-Tree อาจจะใช้เวลาเป็นชั่วโมง เอนจินจึงบันทึก BSP- Tree นี้เป็นไฟล์หลังจากสร้างเสร็จเพื่อที่จะไม่ต้องสร้าง BSP-Tree ซ้ำอีกครั้งเมื่อโหลดโลกนี้อีกในคราวต่อไป ตราบใดที่เราไม่ได้แก้ไขโลกก็ไม่จำเป็นต้องสร้างไฟล์นี้ใหม่อีกครั้ง (BSP-Tree จะถูกบันทึกเป็นไฟล์นามสกุล .bsp)

BSP-Tree ใช้สำหรับตรวจสอบว่าพื้นผิวใดที่จะต้องถูกแสดงผลออกมา และ พื้นผิวใดไม่ต้องเพื่อลดภาระการแสดงผลลง เราสามารถทำอะไรก็ได้ที่ไม่มีการเปลี่ยนแปลงวัตถุที่มองเห็น การสร้างโพลีกอนใดๆ ขึ้นมาใหม่ทำให้จำเป็นต้องสร้าง BSP-Tree ขึ้นมาใหม่ด้วย ทำให้ไม่สามารถสร้างวัตถุขึ้นมาใหม่ใน Viewer mode ได้ รวมถึงการย่อขยาย และ หมุนวัตถุก็เช่นกัน ยกเว้นแต่วัตถุแบบไดนามิกที่ไม่ได้อยู่ BSP-Tree

14.1 ทำไมจึงต้องใช้ Viewer Mode ?

การใช้ BSP-Tree ทำให้เรนเดอร์เร็วขึ้นประมาณ 40% เพราะไม่ต้องเรนเดอร์วัตถุที่ถูกบังอยู่คุณควรใช้ Viewer mode ถ้าเป็นไปได้เพราะใน Editor mode ไม่ได้ใช้ BSP-Tree ร่วมในการแสดงผล

นอกจากนี้ถ้าคุณปล่อยให้วัตถุเคลื่อนไหวเองโดยอัตโนมัติ เช่น Chasing, แทร็ก หรือ ปล่อยให้วัตถุเคลื่อนไหวเองตามกฎฟิสิกส์ เพราะว่าใน Editor mode ไม่สามารถใช้การเคลื่อนโดยใช้วิธีดังกล่าวได้ เนื่องจาก วัตถุแบบสเตติกเท่านั้นที่ถูกนำไปใช้ในการคำนวณสร้าง BSP-Tree ถ้าโลกที่คุณสร้างมีแต่วัตถุแบบไดนามิกการใช้ BSP-Tree ก็ไม่ได้ทำให้เรนเดอร์ได้เร็วขึ้น

คุณไม่ต้องสร้างวัตถุขึ้นในช่วงเวลาที่รันโปรแกรม (Run-time)

บางครั้งเราต้องการที่จะทำลายวัตถุและสร้างขึ้นมาใหม่ เช่น เมื่อฆ่าศัตรู หรือ มีศัตรูแบบใหม่โผล่ออกมา เราไม่จำเป็นต้องทำลาย หรือ สร้างวัตถุขึ้นมาใหม่จริงๆ เราเพียงแต่เอนเนเบิล กับ ดิสเอเบิลวัตถุนั้นก็พอ

14.2 ทำไมจึงต้องใช้ Editor Mode ?

เราต้องสร้างวัตถุขึ้นมาใหม่ขณะรันไทม์

ต้องย่อขยาย หรือ แก้ไขวัตถุ

โลกที่คุณต้องการแสดงมีโพลีกอนมาก แม้ว่าคุณจะใช้ Viewer mode ก็ตามหากมีโพลีกอนมากเกินไปก็ทำให้การแสดงผลช้าได้ นอกจากนั้นการสร้าง BSP-Tree อาจจะใช้เวลาเป็นวัน หรือ เป็นไปไม่ได้เลย ถ้าคุณใช้ Editor mode คุณเพียงแต่สร้างวัตถุในบริเวณของตัวละครของคุณ และ ทำลายวัตถุรอบๆ ทิ้งเมื่อตัวละครออกจากออกบริเวณนั้น เพียงเท่านี้เราก็จะสามารถแสดงผลโลกที่มีขนาดใหญ่ได้ โดยบริเวณไกลเราก็เพียงใช้เอฟเฟคหมอกบังไป ดังรูปตัวอย่างสมมุติให้โลก 3 มิติเป็นรูปบิตแมปขนาดใหญ่ จุดสีขาวกลางบริเวณทึบเป็นตัวละคร บริเวณที่เป็นสีทึบก็คือบริเวณที่อยู่รอบตัวละครของเรา โดยถ้าวัตถุอยู่นอกบริเวณทึบก็ทำลายทิ้ง ถ้าวัตถุเข้ามาอยู่ในบริเวณที่เรากำหนดนี้ก็สร้างขึ้นมา

รูปที่ 14-1 การเรนเดอร์ในขอบเขตที่กำหนด

สรุป

เกมส่วนใหญ่มักจะใช้ Viewer mode ซึ่งมีแทบทุกอย่างที่จำเป็นแล้วหากไม่มีการแก้ไขวัตถุ นอกจากนั้นยังมีความเร็วสูง ส่วน Editor mode เหมาะสำหรับการแสดงผลโลกขนาดใหญ่ที่ไม่สนใจถึงเรื่องความเร็วเป็นปัจจัยสำคัญ หรือ ต้องการสร้างหรือแก้ไขวัตถุขณะรันไทม์ โปรแกรมที่ใช้ Editor mode ก็เช่น Word Builder เป็นต้น

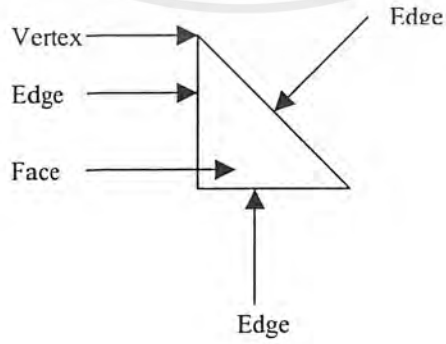
บทที่ 15
การสร้างโมเดล 3 มิติ

15.1 หลักการสร้างกราฟิก 3 มิติด้วยโปรแกรม 3D Studio Max

ในการทำงานบนโปรแกรม 3D Studio Max นั้นจะต้องทำความเข้าใจก่อนว่า กำลังติดต่อกับโลกเสมือนที่ถูกสร้างขึ้นโดยคอมพิวเตอร์ ดังนั้นจึงต้องเข้าใจวิธีการแสดงและเก็บวัตถุต่างๆ ภายในโลกเสมือนนี้ด้วย คือ วัตถุทุกๆ ชิ้นมีโคออร์ดิเนตของตัวมันเอง และถูกสร้างขึ้นจากรูปทรงทางคณิตศาสตร์ หรือกล่าวได้ว่า วัตถุมีลักษณะเป็น 3D space บน cyberspace (สร้างและอยู่เฉพาะบนซอฟต์แวร์เท่านั้น) และทุกๆ จุดภายใน cyberspace มี 3 โคออร์ดิเนต คือ การแสดงความสูง ความกว้าง และความลึกของจุด ซึ่งแต่ละ coordinate จะเรียงตัวไปตามแกนเฉพาะหนึ่งแกน (axis) คือ ความกว้างถูกแสดงด้วยแกน x ความลึกแสดงด้วยแกน y ความสูงแสดงด้วยแกน z

รูปที่ 15-1 ระบบพิกัด 3 มิติ

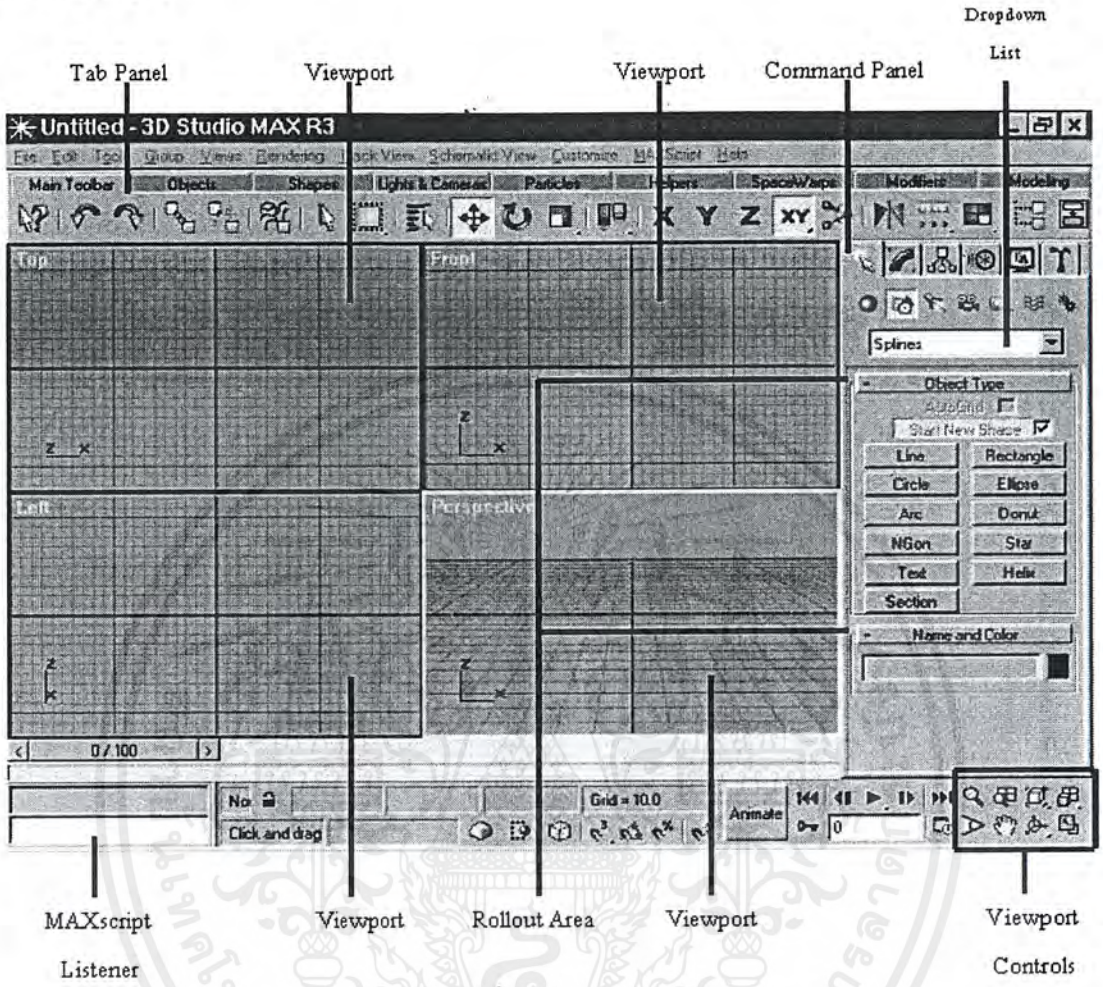
- Line : เส้นที่เกิดจากการเชื่อมจุด 2 จุด
- Polyline : เส้นหลายเส้นที่ทำให้เกิดเป็นด้าน (segment)
- Spline : ใน 3D Studio Max หมายถึง line และ polyline
- Closed shape : เกิดจากการสร้าง polyline และเชื่อมจุดเริ่มต้นกับจุดสุดท้าย และ closed shape อย่างง่ายที่สุด คือ โพลีกอน (polygon) ที่มี 3 ด้าน หรือสามเหลี่ยม หรือเรียกว่า face ซึ่ง face เป็นวัตถุพื้นฐานในระบบ และการสร้างวัตถุที่มีความซับซ้อนมากขึ้นใน 3D Studio Max ล้วนแต่สร้างมาจากโพลีกอนทั้ง 3 ด้านและ 4 ด้าน (สี่เหลี่ยม)



รูปที่ 15-2 Face

15.2 อินเทอร์เฟซภายในโปรแกรม 3D Studio Max

15.2.1 Viewport



รูปที่ 15-3Viewport

ในหน้าจอหลัก 3D Studio Max สามารถเลือกที่จะแสดง viewportได้พร้อมกัน 4 viewport หรือแสดงเพียง 1 viewport เท่านั้นได้เช่นกัน viewport ใน 3D Studio Max ได้แก่ Top Bottom Left Right Perspective และ camera (ถ้ามี) และคลิกเปลี่ยนให้แสดง viewport ใดๆ ได้โดยกดปุ่มตัวอักษรบนคีย์บอร์ด เช่น Top กด T Bottom กด B Left กด L Right กด R Front กด F Back กด K Perspective กด P Camera กด C และ Spotlight กด $ หรือคลิกเมาส์ขวาที่มุมบนซ้ายของ viewport ใดก็ได้ แสดงเมนู pop-up เลือก View และเลือก viewport ที่ต้องการ

Command Panel : เป็นอินเทอร์เฟซหลักที่ติดต่อกับผู้ใช้ ประกอบด้วย 6 commands คือ Create/Modify/*Hierarchy/Motion/Display/Utility*

สำหรับโมเดลที่สร้างในโครงงานนี้จะใช้เพียง Create และ Modify command เท่านั้น

- Create command : เป็น command ที่ใช้ในการสร้างวัตถุทั้งหมดใน 3D Studio Max และส่วนบนเป็นปุ่มคำสั่ง 7 ปุ่ม คือ Geometry/Shapes/Lights/Cameras/Helpers/Space Warps/Systems และภายในแต่

ละปุ่มประกอบด้วย dropdown list ซึ่งแบ่งรูปทรงออกเป็นประเภทหลัก และส่วน rollout ซึ่งรวมคำสั่งย่อยที่ช่วยในการสร้างวัตถุ

- Modify command : เป็น command ที่ใช้แก้ไขวัตถุที่สร้างขึ้นจาก create command

การเรียกใช้งานส่วน rollout สามารถทำได้ 2 วิธี คือ

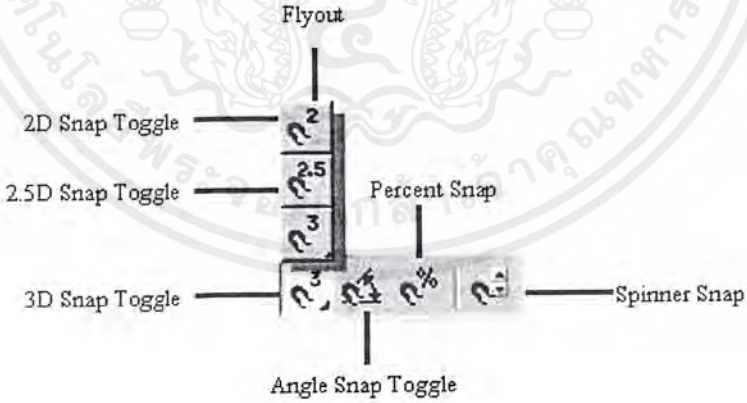
- คลิกเปิดปิดเพื่อขยายส่วน rollout
- คลิกเมาส์ขวาบริเวณ rollout area แสดงเมนู pop-up เพื่อเลือกเปิดและปิด rollout ทั้งหมดที่อยู่ภายใต้ปุ่มคำสั่งต่างๆ ได้

Tab Panel : เป็นอินเทอร์เฟซที่ติดต่อกับผู้ใช้อย่างหนึ่งที่เพิ่มขึ้นใน 3D Studio Max R3.0 ประกอบด้วย 11 tab หลัก คือ Main Toolbar/Objects/Shapes/Lights&Cameras/Participles/Helpers/Spacewarps/Modifiers /Modeling/Rendering และการคลิกเมาส์ขวาบริเวณ tab panel แสดงเมนู pop-up เพื่อเพิ่มเติม ลบหรือเปลี่ยนชื่อ tab สามารถทำได้โดย

15.2.2 เครื่องมือช่วยในการวาดอื่นๆ

เครื่องมือช่วยในการวาดอื่นๆ ที่ใช้ในโครงงานนี้ประกอบด้วย

- การกำหนดหน่วย : เลือก Customize (ในเมนู pull-down) >Unit Setup ซึ่งช่วยในการกำหนดประเภทของหน่วยที่ใช้ในการสร้างวัตถุ
- การกำหนด snap : เลือกการเลื่อนของเคอร์เซอร์ในแบบต่างๆ ซึ่งส่วนใหญ่จะใช้แบบ vertex คือ เคลื่อนเคอร์เซอร์ทีละหนึ่งช่อง (grid) ทำได้โดยเลือก Customize>Grid and Snap Settings>Snaps Tab>Vertex check box หรือเลือกปุ่ม snap เพื่อกำหนด/ยกเลิกการกำหนด snap และถ้าคลิกเมาส์ค้างไว้จะแสดงการกำหนด snap ทีละ Vertex แบบอื่นๆ ให้เลือกในลักษณะ flyout



รูปที่ 15-4 อินเทอร์เฟซของการกำหนด snap แบบต่างๆ

การกำหนด snap แบบอื่นๆ ได้แก่

- Angle Snap Toggle : การปรับมุมทีละ 5 องศา เมื่อมีการหมุนวัตถุ
- Percent Snap : การปรับขนาดทีละ 10 เปอร์เซ็นต์

- Spinner Snap : การปรับค่าใน spinner (เป็นช่องว่างสำหรับเติมค่าใดๆ หรือคลิกเมาส์ที่ปุ่มเพิ่มค่า/ลดค่า) ทีละ 1

<u>เพิ่มเติม</u> : ซึ่งสามารถปรับเปลี่ยนค่า snap เหล่านี้ให้เพิ่ม/ลดทีละเท่าใดได้โดยเลือก Customize>Grid and Snap Settings>Options Tab ส่วนการกำหนด snap แบบ Vertex ให้เพิ่ม/ลดเท่าใด คือ การกำหนดขนาด grid

- การกำหนดขนาดช่อง (grid) ใน 3D Studio Max : เลือก Customize>Grid and Snap Settings>Home Grid Tab ซึ่งสามารมเติมค่าขนาดช่องใน Grid Spacing spinner (ค่าเริ่มต้นที่กำหนดไว้ใน 3D Studio Max เป็น 10) และเติมจำนวนเส้น/ช่องเล็กๆ ในหนึ่งช่องใหญ่ใน Major Lines every Nth spinner (เริ่มต้นเป็น 10)
- การรวมวัตถุให้เป็นก้อนเดียวกัน : เลือกวัตถุที่ต้องการรวมกลุ่มทั้งหมดก่อน จากนั้นเลือก Group (ในเมนู pull-down) >Group และตั้งชื่อกลุ่มวัตถุ
- การเลือกวัตถุ : มี 5 แบบ คือ
  - การเลือกโดยใช้เมาส์คลิกเลือกวัตถุที่ต้องการ (สังเกตได้จากวัตถุที่ยังไม่ถูกเลือกเมาส์เคอร์เซอร์จะเป็นเครื่องหมายบวก) และสามารถเลือกวัตถุได้หลายๆ ชิ้นโดยการกดปุ่ม Ctrl บนคีย์บอร์ดค้างไว้แล้วคลิกเมาส์เลือกวัตถุอื่นต่อไป
  - การเลือกแบบ windowed ทำได้โดยการลากเมาส์ซ้ายเป็นกรอบสี่เหลี่ยม ซึ่งแบ่งเป็น 2 ประเภทย่อย คือ windowed เลือกวัตถุนอกกรอบ และ crossing เลือกวัตถุที่อยู่ภายใน และสัมผัสกรอบ โดยเลือก Edit (เมนู pull-down) >Region>windowed/crossing ซึ่งการเลือกวัตถุแบบนี้ควรเลือกประเภทวัตถุใน Selection Filter drop-down list ก่อน
  - การเลือกจากชื่อวัตถุ โดยคลิกปุ่ม Select by Name ใน Main Toolbar และเลือกวัตถุจากรายการ หรือเลือก Edit (เมนู pull-down) >Select by>Name
  - การเลือกจากสีวัตถุ โดยเลือก Edit (เมนู pull-down) >Select by>Color
  - การเลือกจากกลุ่มวัตถุ โดยเลือก Edit (เมนู pull-down) >Edit Named Selections

15.2.3 การ Merge ไฟล์

การโหลดไฟล์และรวมเข้ากับไฟล์ที่แสดงอยู่ หรือเลือกรวมเฉพาะวัตถุได้เช่นกัน โดยเลือก File (เมนู pull-down) >Merge และ browse ไฟล์ที่ต้องการ แล้วเลือกวัตถุที่ต้องการจากรายการ

15.2.4 การ Import/Export ไฟล์

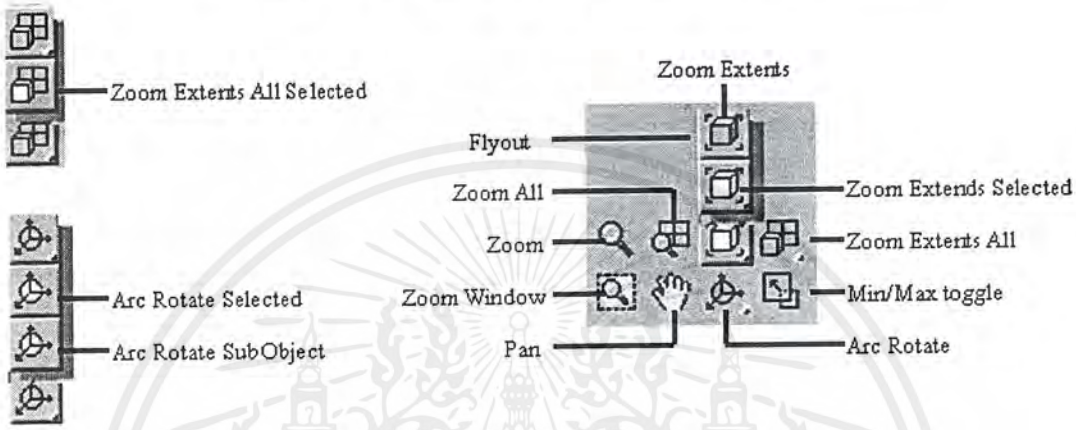
สามารถโหลดไฟล์ในรูปแบบอื่นได้ ซึ่งใน 3D Studio Max R3.0 สามารถ import และรองรับการแสดงไฟล์ประเภท 3D Studio Max R4.0 ASE DXF STL DWG และ VRML และ export เป็นไฟล์ประเภท 3D Studio Max R4.0 (3DS PRJ และ SHP) ถ้าเราติดตั้งปลั๊กอินของมอร์ฟิทเข้าไปก็จะ export เป็น WLD ได้ด้วย

15.2.5 การ Convert ไฟล์เป็น WLD

เราสามารถ convert จากไฟล์ 3DS เป็น WLD ได้โดยใช้เครื่องมือที่มากับมอร์ฟิทคือ 3DS2WLD โดยมันจะแปลงบิตแมปที่จำเป็นไว้ในโฟลเดอร์บิตแมปซึ่งจะอยู่ในโฟลเดอร์เดียวกันกับไฟล์ WLD ที่ convert ไป

15.2.6 การใช้งาน Viewport Controls

เครื่องมือ หรือคำสั่งควบคุม viewport เป็นอินเทอร์เฟซอยู่ที่มุมล่าง-ขวาของหน้าจอ 3D Studio Max และอินเทอร์เฟซควบคุม camera viewport มีความแตกต่างจาก viewport อื่นๆ ทั้งหมด



รูปที่ 15-5 อินเทอร์เฟซของ Viewport Controls

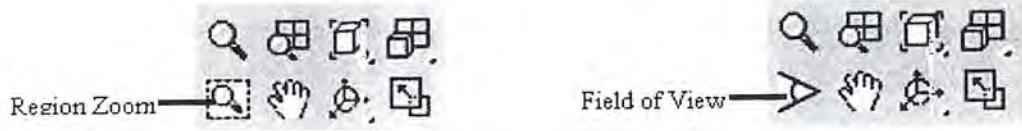
Zoom : เป็นคำสั่งย่อ-ขยายภาพบน viewport ที่ active อยู่ คลิกเมาส์ลากขึ้น คือ ขยายภาพ (Zoom-In) ลากลง คือ ย่อภาพ (Zoom-out)

Zoom All : คำสั่งนี้เหมือนกับคำสั่งที่แล้ว แต่มีผลกับทุก viewport

Zoom Extents : เป็นของคำสั่งย่อ-ขยายภาพ แต่จะทำจนกว่าจะมองเห็นวัตถุทั้งหมดใน viewport และมีคำสั่ง Zoom Extents Selected เป็น flyout สำหรับเลือกเฉพาะวัตถุที่ต้องการ การเลือกคำสั่งให้คลิกเมาส์ค้างไว้ที่ปุ่มจะปรากฏ flyout ขึ้นมาให้เลือก (ให้สังเกตว่า หากที่มุมล่าง-ขวาของปุ่มใดมีรูปสามเหลี่ยมเล็กๆ อยู่ แสดงว่า มีปุ่ม flyout สำหรับเลือกคำสั่งเพิ่มเติม)

Zoom Extents All : คำสั่งนี้เหมือนกับคำสั่งที่แล้ว แต่มีผลกับทุก viewport ในเวลาเดียวกัน และมีคำสั่ง Zoom Extents All Selected เป็น flyout สำหรับเลือกเฉพาะวัตถุที่ต้องการ

Zoom Window : คำสั่งนี้คล้ายกับคำสั่ง Zoom แต่จะเป็นการปรับค่าพื้นที่ในการมองเห็นของ viewport เท่านั้น ซึ่งถ้าใช้คำสั่งบน Planar viewport เช่น Top Right หรือ Front จะถูกแทนที่ด้วย Region Zoom หรือถ้าใช้คำสั่งบน Perspective viewport จะถูกแทนที่ด้วย Field of View



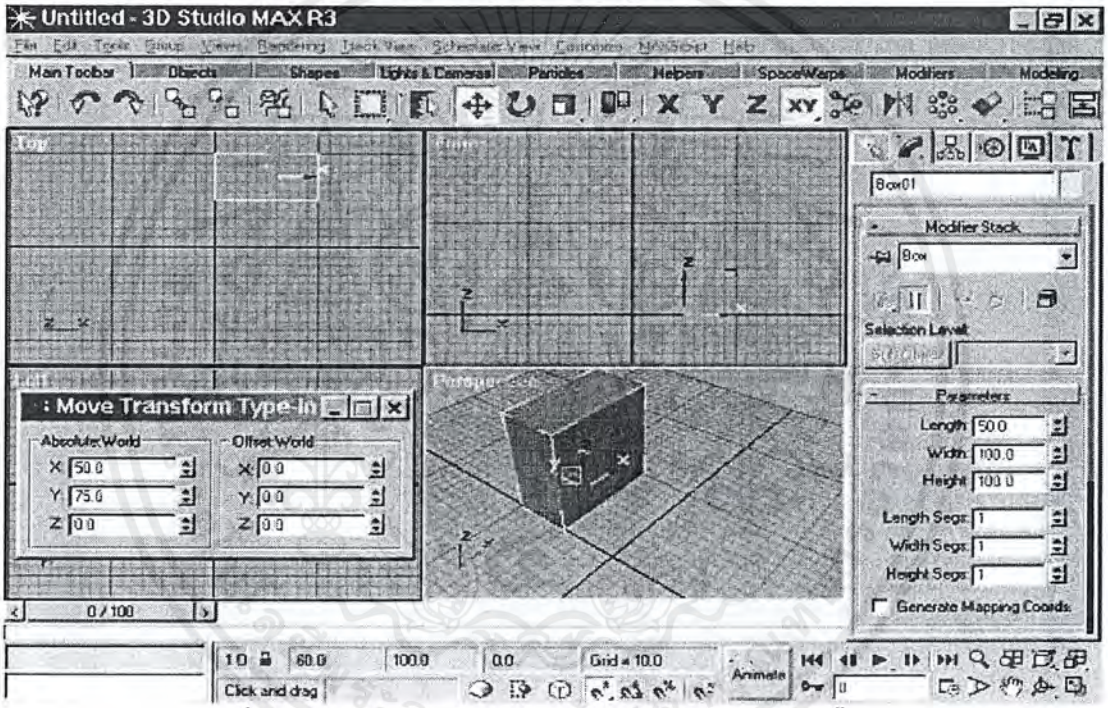
รูปที่ 15-6 Zoom Window

Pan : คำสั่งให้เลื่อน viewport โดยไม่มีผลต่อการย่อ-ขยายภาพ

Arc Rotate : คำสั่งนี้ใช้หมุน viewport ไปรอบๆ โดยใช้จุดศูนย์กลางของ viewport เป็นแกนในการหมุน การใช้งานเมื่อคลิกเลือกคำสั่งนี้ จะปรากฏไอคอนสีเขียวบน viewport คลิกเมาส์ค้างไว้แล้วลากเมาส์ไปรอบๆ ทำให้ viewport เปลี่ยนไปตามเมาส์ และมี Arc Rotate Selected เป็น flyout สำหรับเลือกเฉพาะวัตถุที่ต้องการเป็นแกนหมุน และ Arc Rotate SubObject เป็น flyout สำหรับเลือก material ที่กำหนดลงบนวัตถุ หรือวัตถุย่อย (Sub-Object) เป็นแกนหมุน

15.3 ตัวอย่างการใช้งานโปรแกรม 3D Studio Max

15.3.1 ตัวอย่างการสร้างและปรับคุณสมบัติวัตถุในระดับพื้นฐาน



รูปที่ 15-7 ตัวอย่างการสร้างและปรับคุณสมบัติวัตถุในระดับพื้นฐาน

- Active ที่ Top viewport เลือก Create Command>ปุ่ม Geometry>Object Type rollout>ปุ่ม Box สร้าง box01
- Active ที่ Perspective viewport เลือก Modify Command>Parameters rollout>ใส่ค่า/ปรับค่าใน Length Width และ Height spinner (50, 100, 100)
- ปรับโคออร์ดิเนต โดยกดปุ่ม Select and Move (Main Toolbar Tab)>Tool (เมนู pull-down)>Move Transform Type-In dialogue>ใส่ค่า/ปรับค่าใน X Y และ Z spinner (50, 75, 0)

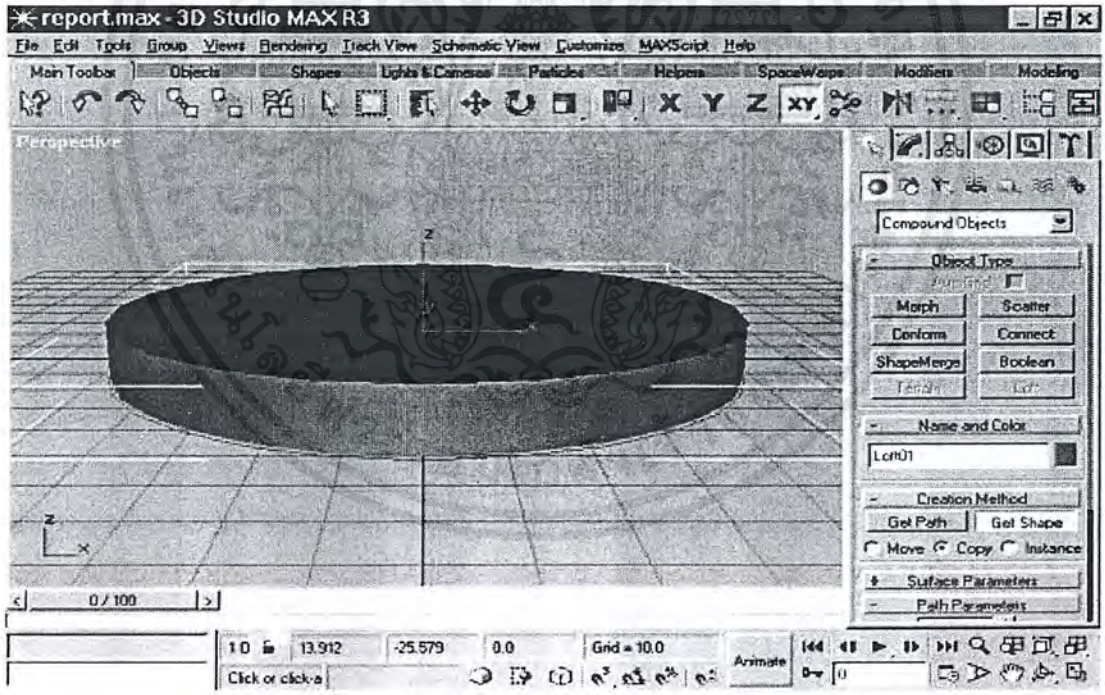
15.3.2 ตัวอย่างการสร้างวัตถุระดับสูงแบบ Loft

เริ่มสร้างวัตถุตัวอย่าง (วงกลม) โดยเลือก Create Command>ปุ่ม Shapes>Object Type rollout>ปุ่ม Circle จากนั้นปรับค่ารัศมี โดยคลิกขยาย Parameters rollout และใส่/ปรับค่ารัศมีใน Radius spinner (50) และสร้างเส้นตรง (ให้ตั้งฉากกับวงกลมที่สร้างไว้) คลิกเมาส์ active Top viewport และเลือก Create Command>ปุ่ม Shapes>

- ขยาย Creation Method rollout>Initial Type>Corner option
- ขยาย Keyboard Entry rollout>ใส่ค่า 0 ใน X spinner Y spinner และ Z spinner>คลิกปุ่ม Add Point และใส่ค่า 10 (ความสูงเส้นตรงที่ต้องการ)ใน z spinner คลิกปุ่ม Add Point

ได้เส้นตรงตั้งฉากกับวงกลม ซึ่งมีโคออร์ดิเนต (0, 0, 0) และเราจะนำวงกลมกับเส้นตรงมาสร้างเป็นวัตถุทรงกระบอกได้ โดยการสร้างวัตถุ loft ซึ่งมีขั้นตอนดังนี้

- Main Toolbar Tab>Select and Move>เลือกเส้นตรงเพื่อสร้างวัตถุ loft (ได้ทรงกระบอกตัน ถ้าเลือกวงกลม จะได้ทรงกระบอกกลวง)
- Create command>ปุ่ม Geometry>Compound Objects ใน drop-down list>Object Type rollout>ปุ่ม Loft
  - Creation Method rollout>ปุ่ม Get Shape>เลือก Copy option กรณีสร้างวัตถุ loft เสร็จจะคัดลอกวงกลมเก็บ แต่วงกลมจะมีการเปลี่ยนขนาดอย่างไรไม่ทำให้วัตถุ loft เปลี่ยนนขนาดตาม

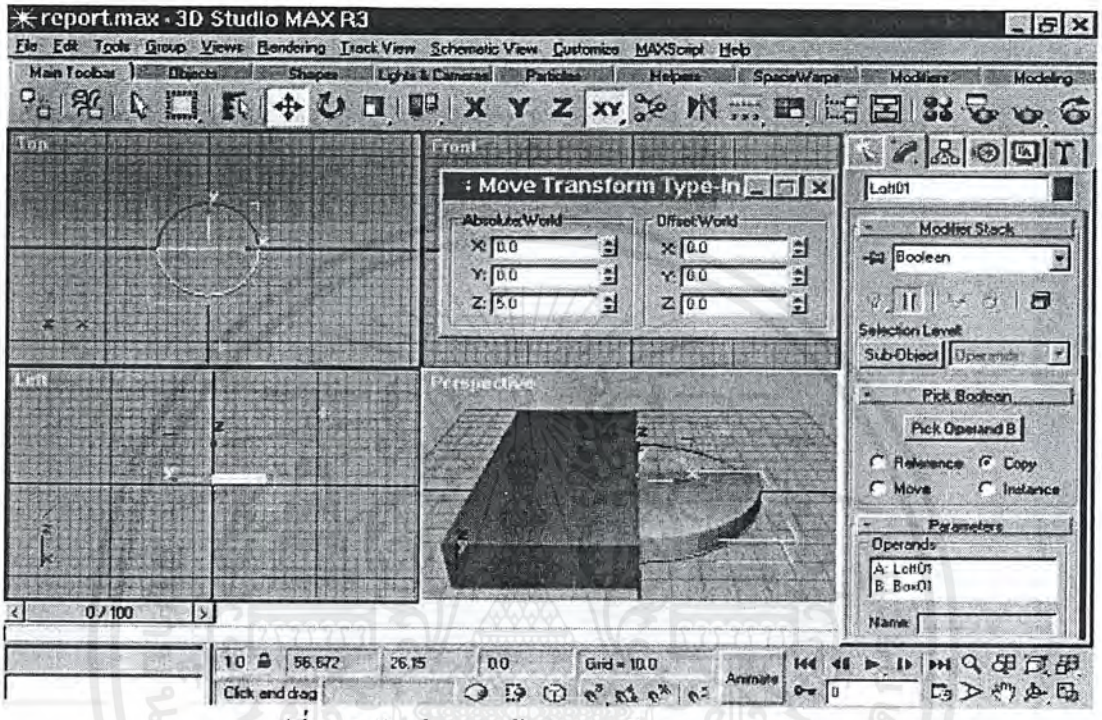


รูปที่ 15-8 ตัวอย่างการสร้างวัตถุระดับสูงแบบ Loft

15.3.3 ตัวอย่างการสร้างวัตถุระดับสูงแบบ Boolean

- สร้าง box ซึ่งมีขนาด 60 ×120 ×20 และย้ายไปที่โคออร์ดิเนต (0, 30, -5)

- คลิกปุ่ม Select and Move (ใน Main Toolbar Tab)>คลิกเมาส์เลือกวัตถุ loft>Create Command> Object Type rollout>คลิกปุ่ม Boolean
  - Operation>Subtraction (A-B) option ซึ่งวัตถุ A คือ วัตถุ loft
  - Pick Boolean rollout>คลิกปุ่ม Pick Operand B และคลิกเมาส์เลือก box ที่สร้างไว้ คือ ให้วัตถุ B เป็น box

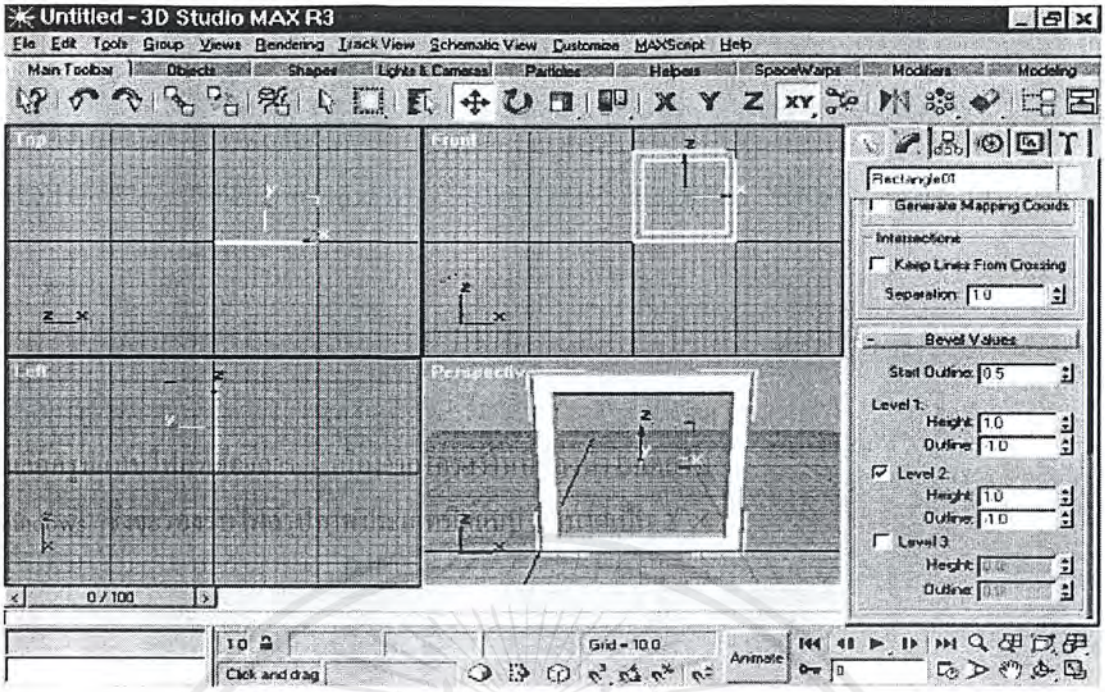


รูปที่ 15-9 ตัวอย่างการสร้างวัตถุระดับสูงแบบ Boolean

สังเกตเห็นว่าวัตถุ loft ถูกตัดไป ½ ส่วน หรือเป็นวัตถุ boolean ทรงกระบอกที่สร้างขึ้นเพียง ½ ส่วน

15.3.4 ตัวอย่างการสร้างวัตถุระดับสูงแบบ Edit Spline และ Bevel

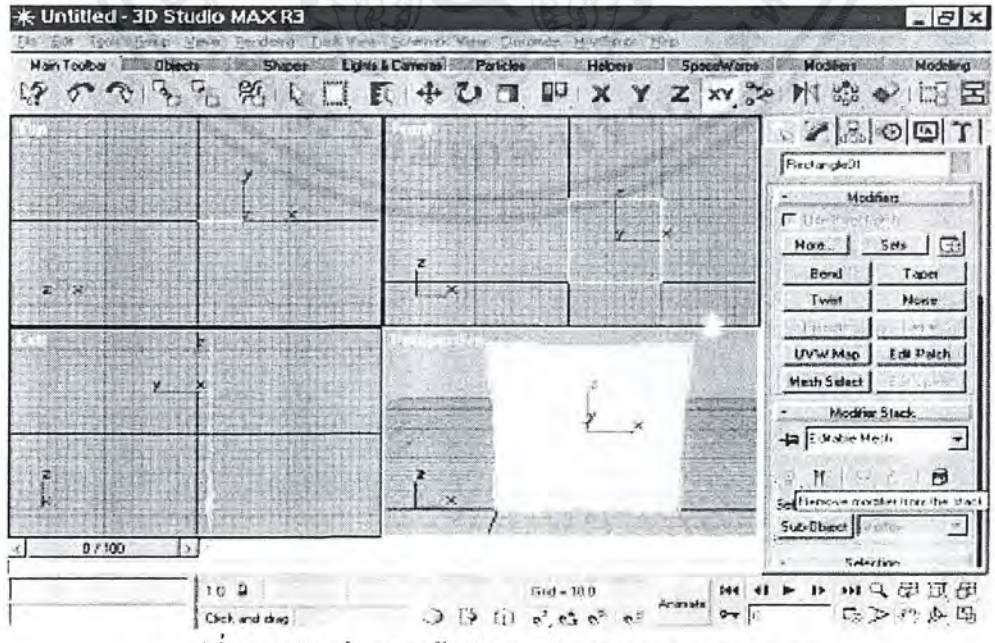
- สร้างรูปสี่เหลี่ยม 2 รูปซ้อนกันให้เป็นวัตถุชิ้นเดียวกัน โดย active ที่ Top viewport เลือก Create Command>ปุ่ม Shapes คลิกปลด check box ด้านขวาของปุ่ม Start New Shape ออก คลิกปุ่ม Rectangle สร้างรูปสี่เหลี่ยมซ้อนกัน 2 รูปเป็น Rantangle01
- แปลง Rantangle01 เป็น Editable Mesh โดย Modify C ›mmand>Modifier Stack rollout>ปุ่ม Edit Spline สังเกตว่าค่าใน drop-down list เป็น Edit Spline
- สร้าง Rantangle01 เป็น Bevel โดย Modify Command>Modifier rollout>ปุ่ม More คลิกเลือก Bevel ขยาย Bevel Values rollout ใส่ค่า/ปรับค่าใน Start Outline spinner (0.5) Level1 Height spinner (1.0) Level1 Outline spinner (-1.0) Level2 Height spinner (1.0) Level2 Outline spinner (-1.0)



รูปที่ 15-10 ตัวอย่างการสร้างวัตถุระดับสูงแบบ Edit Spline และ Bevel

15.3.5 ตัวอย่างการสร้างวัตถุระดับสูงแบบ Editable Mesh

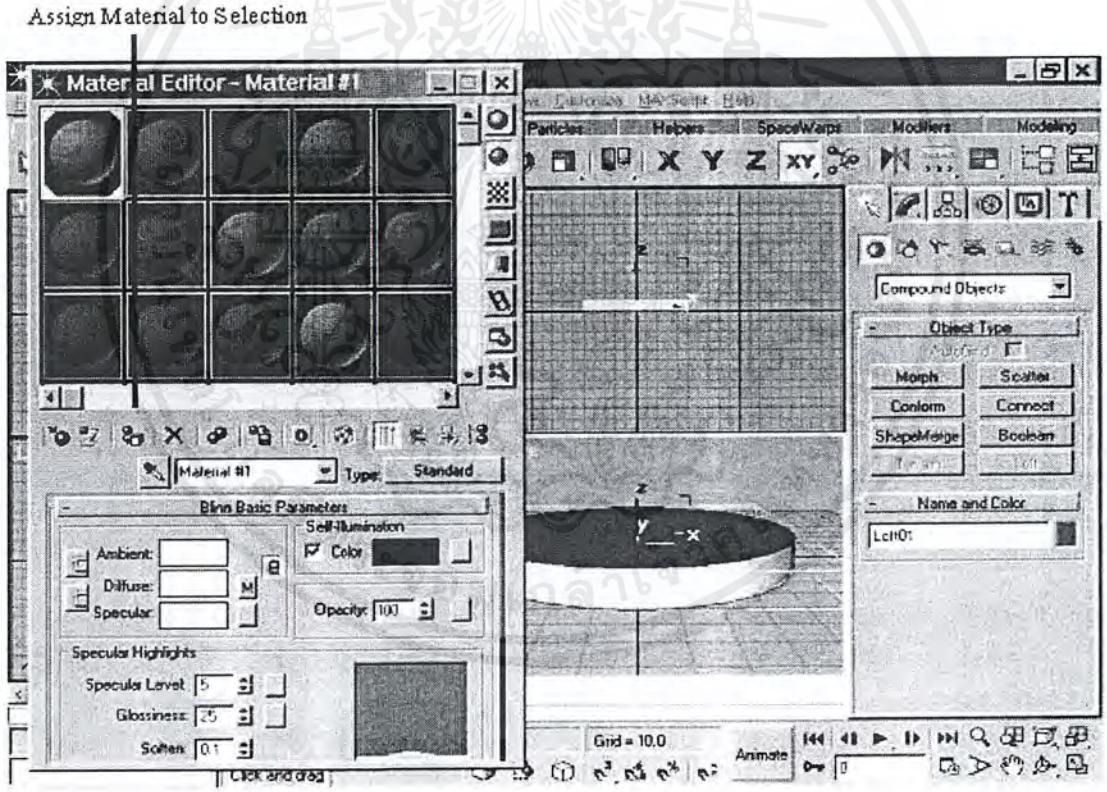
- active ที่ Top viewport> Create Tab>ปุ่ม Shapes>Object Type rollout>ปุ่ม Rectangle และสร้างรูปสี่เหลี่ยม หรือ Rantangle01
- Modify Tab>Modifiers rollout>ปุ่ม Edit Stack แปลงเป็น Editable Mesh หมายถึง แปลงวัตถุจากรูปสี่เหลี่ยมที่สร้างไว้เป็นแผ่นสี่เหลี่ยม หรือ editable mesh ซึ่งเป็นกฎของ 3D Studio Max ที่สามารถมองเห็นพื้นผิวได้เพียงด้านเดียว แสดงผลดังรูป



รูปที่ 15-11ตัวอย่างการสร้างวัตถุระดับสูงแบบ Editable Mesh

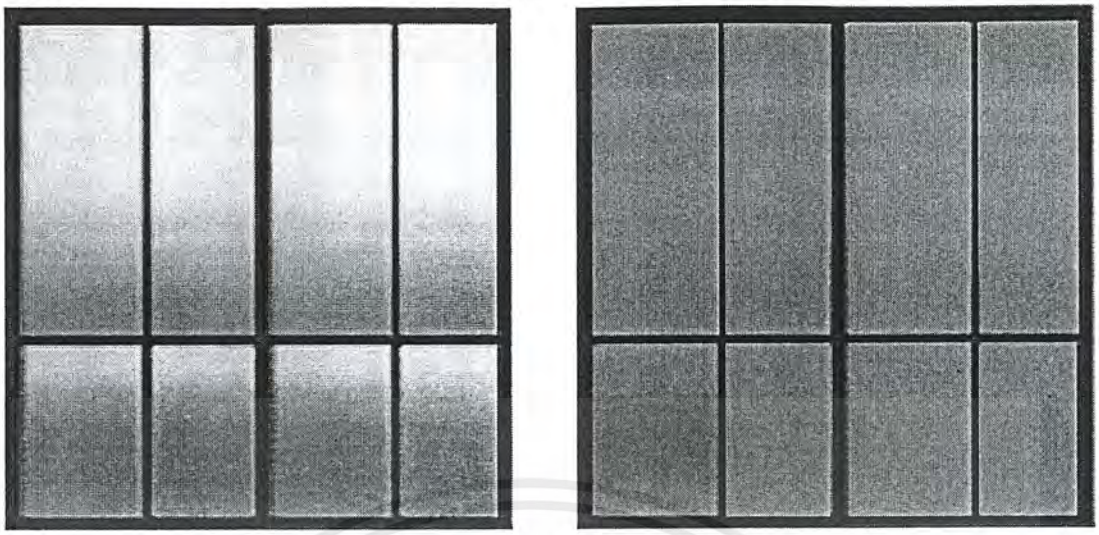
15.3.6 ตัวอย่างการกำหนด Material ให้กับวัตถุ

- Main Toolbar Tab>ปุ่ม Select and Move>เลือกวัตถุ loft
- เลือก Tool (เมนู pull-down) >Material Editor>เลือก Material จากช่องแสดง Material สังเกตว่า Material ที่เลือกมีกรอบสีขาวล้อมรอบ
- Binn Basic Parameters rollout>คลิกเมาส์ที่ช่อง Ambient Color แสดง Color Selector dialogue เพื่อปรับสีตามต้องการ แต่ในที่นี้จะนำไฟล์รูปแบบอื่น (bitmap) จากภายนอก 3D Studio Max มาเป็น Material จึ่งต้องปรับ Ambient Color ให้เป็นสีขาว และปรับสีที่ช่อง Diffuse Color ให้เป็นสีขาวด้วย
- คลิกเมาส์ที่ปุ่มด้านขวา Diffuse Color เพื่อ browse ไฟล์ Material แสดง Material/Map Browser dialogue คลิกเลือกบิตแมปเพื่อเลือกรูปแบบไฟล์ที่ต้องการ เนื่องจากโครงงานนี้ต้องนำโมเดลไปพัฒนาเกมโดยใช้ DirectX บน Windows ซึ่งจะรองรับไฟล์รูปภาพประเภทบิตแมป มีขนาดเป็น $2^n \times 2^n$ (เป็นสี่เหลี่ยมจัตุรัส) และคลิกปุ่ม OK เพื่อแสดง Select Bitmap Image File dialogue คลิกเมาส์เลือกไฟล์ที่ต้องการ และคลิกปุ่ม Open และจะกลับมาที่ Material/Map Browser dialogue และคลิกปุ่ม Assign Material to Selection



รูปที่ 15-12 ตัวอย่างการกำหนด Material ให้กับวัตถุ

15.3.6.1 ตัวอย่างการใช้เทคนิคบน PhotoShop ในการสร้าง Material เพื่อเพิ่มความสมจริงให้กับวัตถุ



รูปที่ 15-13 ตัวอย่างการใช้เทคนิคบน PhotoShop เพื่อเพิ่มความสมจริงให้กับวัตถุ

สำหรับพื้นผิวัตถุที่เป็นกระจก ในการแต่งบิตแมปบน PhotoShop จะนำ Linear Gradient Tool มาใช้ในการแสดงสีในความเข้มไม่เท่ากัน เพื่อให้หน้าต่างที่เป็นกระจกมีเงา

- ปรับโหมดของบิตแมปให้เป็น RGB Color ก่อน
- ใช้ Rectangular Marquee Tool เลือกส่วนที่ต้องการทำ Gradient
- ใช้ Linear Gradient Tool คลิกเมาส์ค้างและลากบนส่วนที่ต้องการ โดยจุดเริ่มต้นลากเมาส์เป็นสีในช่อง foreground และจุดสุดท้ายเป็นสีในช่อง Background และไล่ความเข้มสีในปริมาณที่เหมาะสมกับระยะที่ลากเมาส์ค้างตั้งแต่จุดแรกถึงจุดสุดท้าย
- ปรับโหมดของบิตแมปให้เป็น Indexed Color

15.3.7 ตัวอย่างการ Map Material

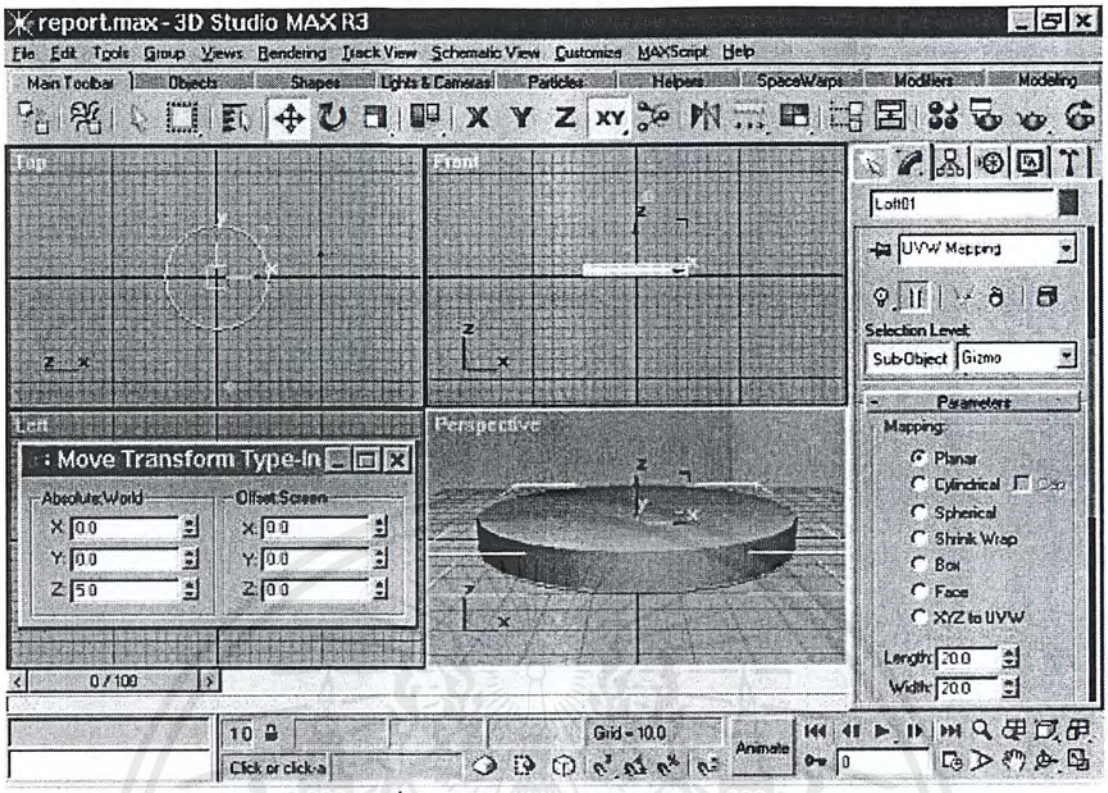
ขั้นตอนนี้สำคัญมาก คือถ้าทำเพียงการกำหนด material ให้กับวัตถุ สามารถเรนเดอร์โดยที่เห็นลาย material ได้เฉพาะใน 3D Studio Max เท่านั้น และลาย material บนวัตถุจะไม่เป็นระเบียบตามที่ต้องการ

- เริ่มต้นเลือก Modify Command>Modifier rollout>ปุ่ม UVW Map แสดง Parameters rollout
- เลือก Planar option (ให้ map วัตถุด้วย material ที่เลือกไว้ เพียง 2 ด้าน คือ หน้า-หลัง/ซ้าย-ขวา/บน-ล่าง) และใส่ค่าใน Length spinner และ Width spinner ซึ่งเป็นความลึก กว้างของ material ที่จัดเรียงลงบนวัตถุ และใน Alignment ให้เลือก X/Y/Z option (เป็นการเลือกให้ material วางตัวตามแกน X/Y/Z)

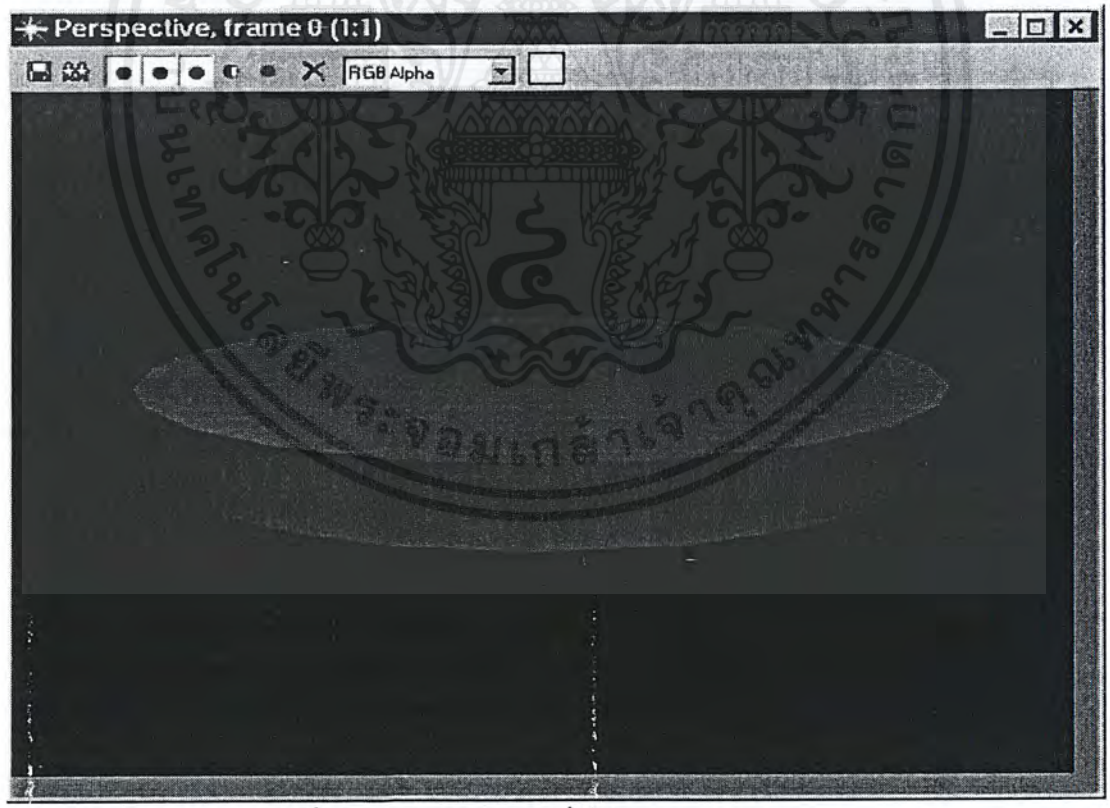
การจัดเรียง material บนวัตถุ การเพิ่มขนาด material และการหมุน material สามารถทำได้ โดยมอง material เป็นวัตถุย่อยๆ (Sub-Object) ซึ่งมีขั้นตอนดังนี้

- Modify Command>Modifier Stack rollout>เลือก UVW Mapping และคลิกปุ่ม Sub-Object

- การเคลื่อน การหมุน การปรับขนาด material (วัตถุย่อย) ใช้หลักการเช่นเดียวกับวัตถุ

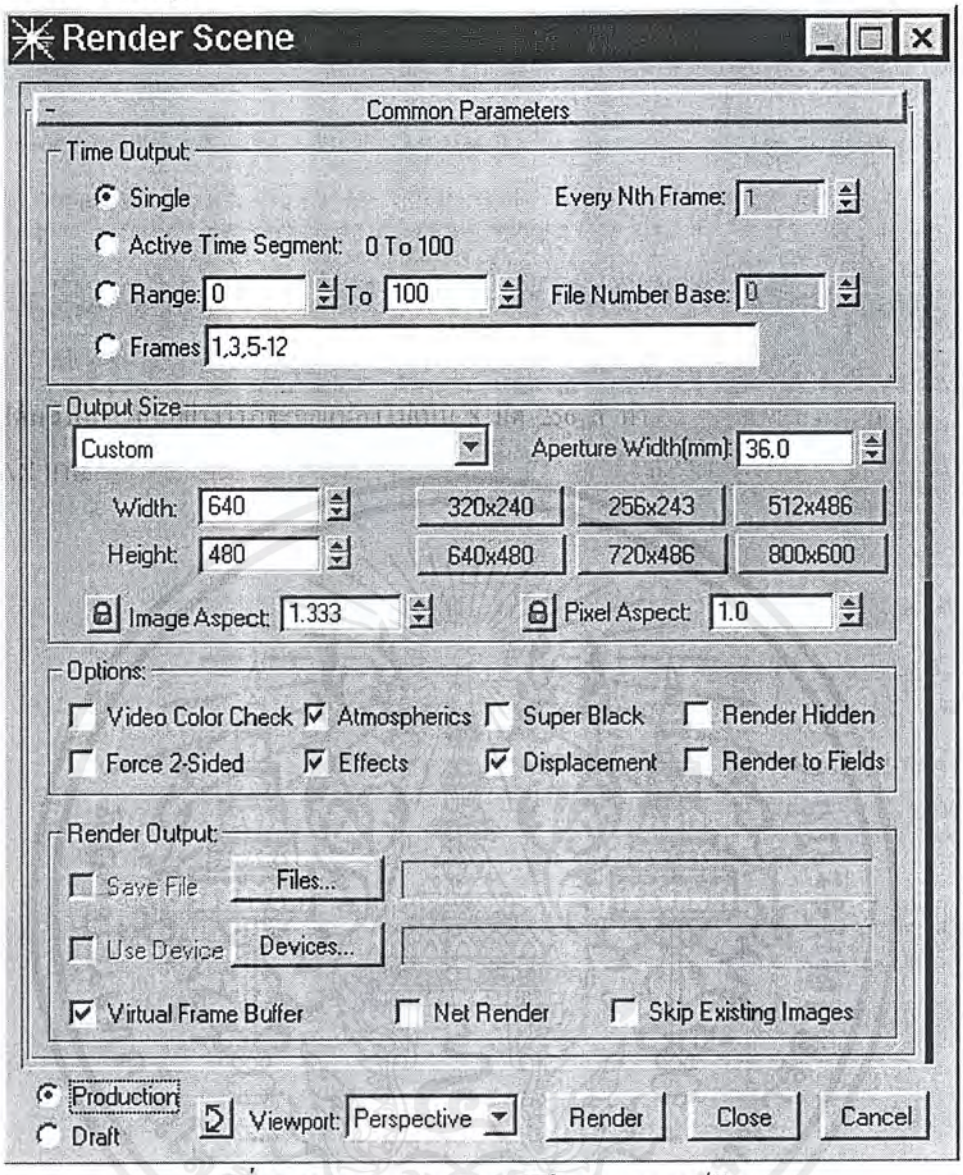


รูปที่ 15-14 ตัวอย่างการ Map Material



รูปที่ 15-15 ผลการเรนเดอร์ซึ่งได้จากการ map material

15.3.8 การเรนเดอร์วัตถุ



รูปที่ 15-16 การกำหนดคุณสมบัติในการเรนเดอร์วัตถุ

การ map ลาย material ลงบนวัตถุ จำเป็นต้องทำการเรนเดอร์ เพื่อแสดงผล ซึ่งมี 3 วิธี คือ

- เลือก Render (เมนู pull-down) >Render แสดง Render Scene dialogue
  - Time Output rollout>single option
  - Output Size rollout>คลิกเลือกปุ่มขนาดหน้าจอที่ต้องการให้เรนเดอร์ (ในที่นี้เลือก 640 ×480) ซึ่งหมายถึง ค่าใน Length spinner และ Height spinner ตามลำดับ
  - Render Output>คลิกปุ่ม Files.. ถ้าต้องการ capture ผลจากการเรนเดอร์เก็บเป็นไฟล์รูปภาพ
  - Viewport spinner หมายถึง เลือกเรนเดอร์จาก viewport ใด
  - ปุ่ม Render
- เลือก viewport ที่ต้องการเรนเดอร์>ปุ่ม Quick Render (ใน Main Toolbar Tab)

- กดปุ่ม Shift และ Q พร้อมกัน

15.4 ปัญหาในการสร้างโมเดล

- การจัดแสงและกล้องใน 3D Studio Max จะไม่แสดงผล เมื่อนำมาโปรแกรมเกมด้วยมอร์ฟิท
- การสร้างเกมในโครงงานนี้ประกอบด้วยวัตถุจำนวนมาก เพราะต้องคำนึงถึง concept หลักของการสร้างโมเดลที่ต้องการความเหมือนจริงให้ได้มากที่สุด ทำให้การรันเกม (การเรนเดอร์วัตถุในเกม) ช้าลงไปมาก
- การเรนเดอร์ใน 3D Studio Max มีการใช้เทคนิคต่าง ๆ ช่วยมากมายทำให้ภาพที่ได้ดูดี แต่ในมอร์ฟิทจะเน้นไปที่ความเร็วในการเรนเดอร์ อีกทั้งบิตแมปจะถูกแปลงไปเป็นรูปแบบที่เหมาะสมกับมอร์ฟิทมากที่สุดซึ่งไม่ว่าความลึกสีของบิตแมปต้นฉบับจะเป็นอย่างไรก็จะถูกแปลงไปเป็น 8 บิต 256 สี ทำให้ภาพที่ได้ดูด้อยกว่าใน 3D Studio MAX มาก

บทที่ 16
แนวคิดและการออกแบบ

16.1 หลักการของโลก 3 มิติและ Static Object

ในขณะที่เราสร้างและแก้ไขโลก 3 มิติในไฟล์ .wld ซึ่งเราทำผ่านทาง WorldBuilder นั้นเรากำลังทำงานใน Editor Mode ซึ่งจะมีจุดเด่นคือ สามารถเข้าถึงได้ในระดับโพลีกอน แต่ก็มีข้อเสียคือมีประสิทธิภาพในการเรนเดอร์ที่ค่อนข้างช้า เมื่อเราแก้ไขโลก 3 มิติให้เป็นไปตามที่เราต้องการแล้วเราอาจต้องการการปรับแต่งเพิ่มเติมซึ่งไม่สามารถทำไปเลยใน WorldBuilder ได้ เช่น การเลื่อนหรือเปลี่ยนมุมกล้อง ในแบบเรียลไทม์ เป็นต้น ซึ่งเราจะพิจารณาในหัวข้อต่อ ๆ ไป

สรุปคือโลก 3 มิติที่เราสร้างขึ้นใน WorldBuilder จะยังเป็นโลกที่แข็ง ๆ ไม่มีการเคลื่อนไหว เราจึงเรียก object ในลักษณะนี้ว่า Static Object ซึ่งเรามักใช้ object แบบนี้กับโมเดลที่ไม่ต้องการการเคลื่อนไหว เช่น ตึก, สิ่งก่อสร้าง ฯลฯ มอร์ฟิทจะสร้าง BSP Tree ขึ้นสำหรับวัตถุแบบนี้ในโลก 3 มิติของเรา เพื่อช่วยในการเรนเดอร์ให้เร็วขึ้น โดยการทำงานในรูปแบบนี้เรียกว่า Viewer Mode ซึ่งมีจุดเด่นคือประสิทธิภาพสูง แต่มีจุดด้อยคือไม่สามารถเข้าถึงโลก 3 มิติในระดับโพลีกอนได้

16.2 หลักการของ Dynamic Object และ MD2

โลก 3 มิติของเรา ย่อมต้องมีการเคลื่อนไหว ไม่ว่าจะเป็นวัตถุต่าง ๆ ในฉาก, กล้อง หรือแม้แต่แสงและเงา ก็ต้องมีการเคลื่อนที่ สิ่งเหล่านี้ต้องการการทำงานแบบเรียลไทม์ ตรงจุดนี้เองที่มอร์ฟิทเอพีไอจะเข้ามามีบทบาทมากในการควบคุมวัตถุให้เคลื่อนที่, หมุน, ติดตามวัตถุอื่น, เคลื่อนที่ไปตามเส้นทางที่กำหนดไว้ ฯลฯ เราเรียกวัตถุที่มีการเคลื่อนไหว ต้องการการเปลี่ยนแปลงนี้ว่า Dynamic Object

เราสั่งการควบคุม dynamic object ผ่านทางฟังก์ชันต่าง ๆ ซึ่งเป็นไลบรารีเพิ่มเติมของมอร์ฟิทใน Visual C++ จึงจำเป็นที่จะต้องมีการเพิ่ม DLL และเฮดเดอร์ไฟล์ที่จำเป็นในเวลาคอมไพล์และรันโปรแกรมด้วยได้แก่ morfit.dll และ mrft_api.h

MD2 เป็นมาตรฐานของโมเดลที่เก็บลำดับของการเคลื่อนไหวของโมเดลไว้ในตัวมันเองมากมาย เช่นเดิน, ยืน, วิ่ง, โจมตี ฯลฯ จะเห็นได้ว่าเหมาะกับการนำมาใช้เป็น dynamic object มาก และมอร์ฟิทยังสนับสนุนการใช้งาน MD2 ในหลายเอพีไอของมันเช่น Object API สามารถควบคุมและจัดการเกี่ยวกับลำดับการเคลื่อนไหวของ MD2 ได้ในระดับที่น่าพอใจทีเดียว

16.3 การตรวจสอบการชน

ในโครงงานนี้มีหลายส่วนด้วยกันที่จะต้องใช้การตรวจสอบการชนกันระหว่างวัตถุ ตัวอย่างเช่นการตรวจสอบการชนระหว่างกระสุนกับกำแพงหรือ static object ใด ๆ มีหลักการพื้นฐานอยู่สองอย่างคือแบบเส้นและแบบกล่องในที่นี้เนื่องจากกระสุนมีการวิ่งเป็นเส้นตรงเราจึงเลือกใช้แบบเส้น โดยเราจะกำหนดเวกเตอร์ที่ไปตามแนวของกระสุนขึ้นมาเส้นหนึ่ง เมื่อใดก็ตามที่มีการยิงเกิดขึ้น ก็จะทำการลากเวกเตอร์นี้

ไปตามทิศทางของกระสุนในขณะนั้น ถ้าเวกเตอร์นี้ไปชนกับอะไรเข้ามันจะคืนค่าแฮนเดิลของวัตถุที่มันไปชนกลับมาให้เรา ซึ่งจะทำให้เรารู้ว่าชนกับอะไรและดำเนินการกับการชนนั้นต่อไป

16.4 การเคลื่อนที่ของวัตถุและการตรวจสอบระดับพื้น

เป็นการตรวจสอบว่าวัตถุ หรือตัวละครนั้นจะสามารถผ่านเข้าไปยังตำแหน่งที่ต้องการได้หรือไม่ เช่น สามารถขึ้นบันไดได้ แต่ไม่สามารถขึ้นไปยังหลังคาตึก เป็นต้น โดยใช้หลักกการตรวจสอบการชนมาใช้ในการ ตรวจสอบตำแหน่งพื้น โดยใน โลก 3 มิติอย่างง่ายๆ จะใช้การลาก เวกเตอร์ จากจุดสูดสุด (ค่า z มากๆ หรือ อินฟินิตี้) มายังจุดต่ำสุด (ค่า z น้อยมากๆ หรือ ลบอินฟินิตี้) แล้วหาจุดตัดกับ พื้นโลกหรือวัตถุ จะได้ตำแหน่งที่ตัดมา เปรียบเทียบความสูง(ค่า z) ของวัตถุกับจุดตัดนั้น ถ้ามากกว่าที่วัตถุจะขึ้นได้ (ในที่นี้ใช้ 1/3 ของความสูงตัวละคร) ก็จะไม่สามารถเดินไปยังตำแหน่งนั้นได้

แต่การตรวจสอบนี้มีข้อด้อยคือใน โลกที่ซับซ้อนขึ้น เช่น มีหลังคา เราจะไม่สามารถลอดเข้าไปข้างล่างได้ เนื่องจากจุดตัดที่ได้จะเป็นจุดบนหลังคาซึ่งสูงเกินกว่าวัตถุจะขึ้นได้ จึงทำการแก้ไข การตรวจสอบตำแหน่งโดย ให้ลากเวกเตอร์ออกจากจุด ที่วัตถุสามารถขึ้นได้ (1/3 ของความสูง) ไปสองทิศทางคือ ด้านบนถึง อินฟินิตี้ และด้านล่างถึงลบอินฟินิตี้ ถ้าจุดตัดด้านบนอยู่สูงกว่าความสูงของวัตถุ(หลังคา) และมีจุดตัดด้านล่าง เราก็จะสามารถเข้าไปได้

16.5 การจัดกล้อง

ในโลก 3 มิติ นั้นกล้องถือเป็นสิ่งสำคัญมาก เพราะ ถึงแม้ว่าเราจะสามารถสร้างโลกที่เหมือนจริงและสวยงามขนาดไหน แต่เราไม่สามารถแสดงโลกนั้นผ่านทางอื่นได้เลย นอกจากผ่านกล้องเพียงทางเดียวเท่านั้น ดังนั้นการจัดกล้องจึงเป็นสิ่งสำคัญมากที่จะทำให้ โลกเราออกมาดูดี และสวยงามสมจริง

แต่ในเกมรูปแบบ มุมมองบุคคลที่หนึ่งนั้น มุมกล้องถูกยึดติดกับตัวผู้เล่น ทำให้เราไม่สามารถทำการจัดตำแหน่งหรือทิศทางได้ เนื่องจากกล้องต้องขยับ แหละหมุนตาม การบังคับของผู้เล่น เช่น ผู้เล่นเลื่อน เมาส์ไปทางซ้ายกล้องต้องหมุนซ้าย เลื่อนไปขวาหมุนขวา เลื่อนขึ้นลง หรือการเดินไปข้างหน้า ถอยหลัง กล้องก็ต้องเลื่อนไปข้างหน้าและถอยหลัง เป็นต้น

16.6 ระบบ AI

ศัตรู จะเดินไปยังทิศทางที่ผู้เล่นอยู่ และเมื่อเข้าใกล้ถึงระยะหนึ่ง(หวังผลที่จะยิงได้) ก็จะทำการยิงใส่ผู้เล่น โดยทำการสุ่ม ความแม่นยำ ในการยิง

16.7 การใช้ Effect

เทคนิคพิเศษที่ใช้ในเกมหลักๆ มีอยู่ 3 ประเภทคือ

1. Alpha Blending เป็นความสามารถในการแสดงวัตถุให้โปร่งแสง คล้ายๆ ฟิลเตอร์ที่ใช้ในกล้องถ่ายภาพ จะได้ในการสร้าง ประกายไฟให้สมจริง รวมถึงแสดงอาการ บาดเจ็บ

2. Atmospheric Effect เป็นการสร้างหมอก ขึ้นมาเพื่อให้วัตถุที่อยู่ไกลๆ มีสีที่ซีดจางลงเพื่อความสมจริง และยังช่วยเพิ่มประสิทธิภาในการ แสดงผล เนื่องจากวัตถุที่อยู่ไกลจะถูกหมอกบังจนหมด
3. Lighting Effect เป็นการใส่แสงลงใน โลก 3 มิติ เพื่อให้สมจริงยิ่งขึ้น รวมทั้งสามารถจัดให้มืด หรือสว่างตามสภาพความเป็นจริงได้ เช่น เราสามารถทำให้โลกที่สร้างมืด เพื่อให้เป็นกลางคืน หรือ จัดแสงให้สว่างมากๆ เหมือนกลางวัน และการยิงปืนในเวลากลางคืนก็จะเกิดประกายไฟขึ้นสว่าง เพื่อเพิ่มความสวยงามและสมจริง

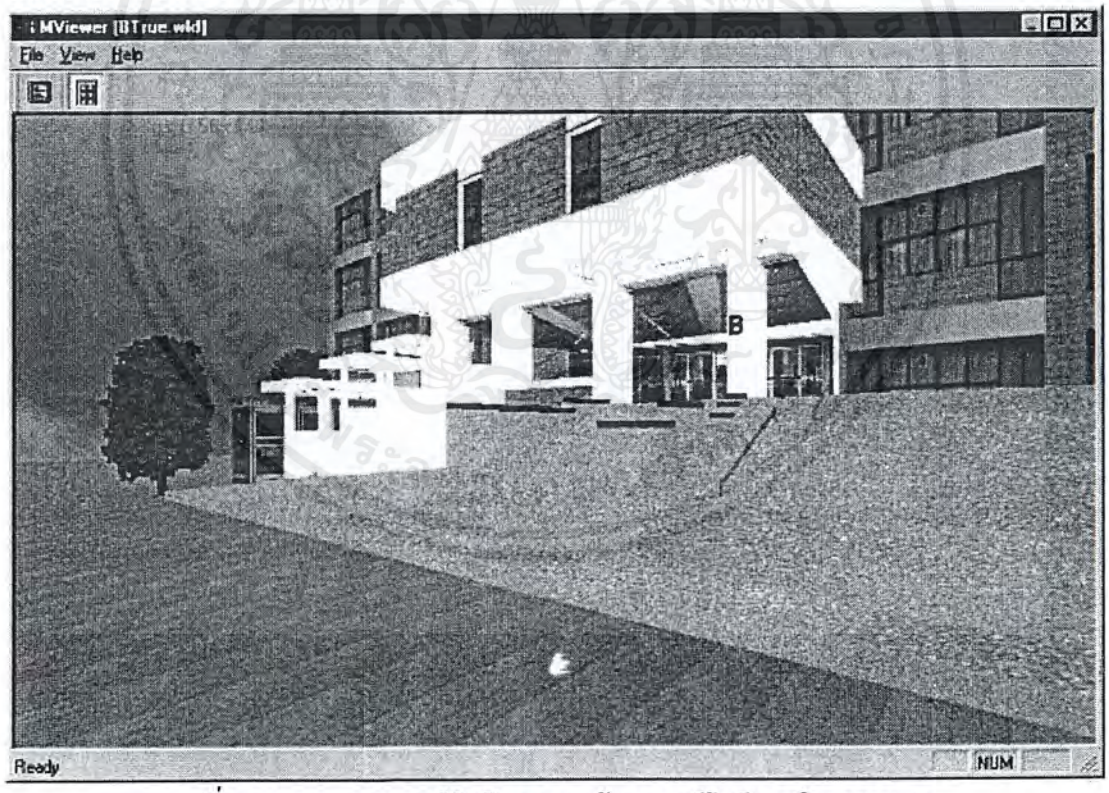
บทที่ 17
ขีดจำกัดและปัญหาที่พบ

17.1 เฟรมเรตต่อวินาที (Frame per Second/FPS)

มอร์ฟีทสามารถเรนเดอร์โลก 3 มิติได้ในด้วยความเร็วในระดับหนึ่ง ซึ่งจะมีผลโดยตรงมาจากจำนวนโพลีกอนที่มันต้องเรนเดอร์ในหน้าจอที่เราเห็นในขณะนั้น (Field Of View หรือ FOV), จำนวนโพลีกอนที่ต้องเรนเดอร์ของ dynamic object, dynamic object นั้นเป็น MD2 หรือไม่ ฯลฯ แต่ในที่นี้เราจะพิจารณาการเรนเดอร์โลก 3 มิติที่มีแต่ static object ก่อน โดยการทดลองที่มีค่าเริ่มต้นดังนี้ :

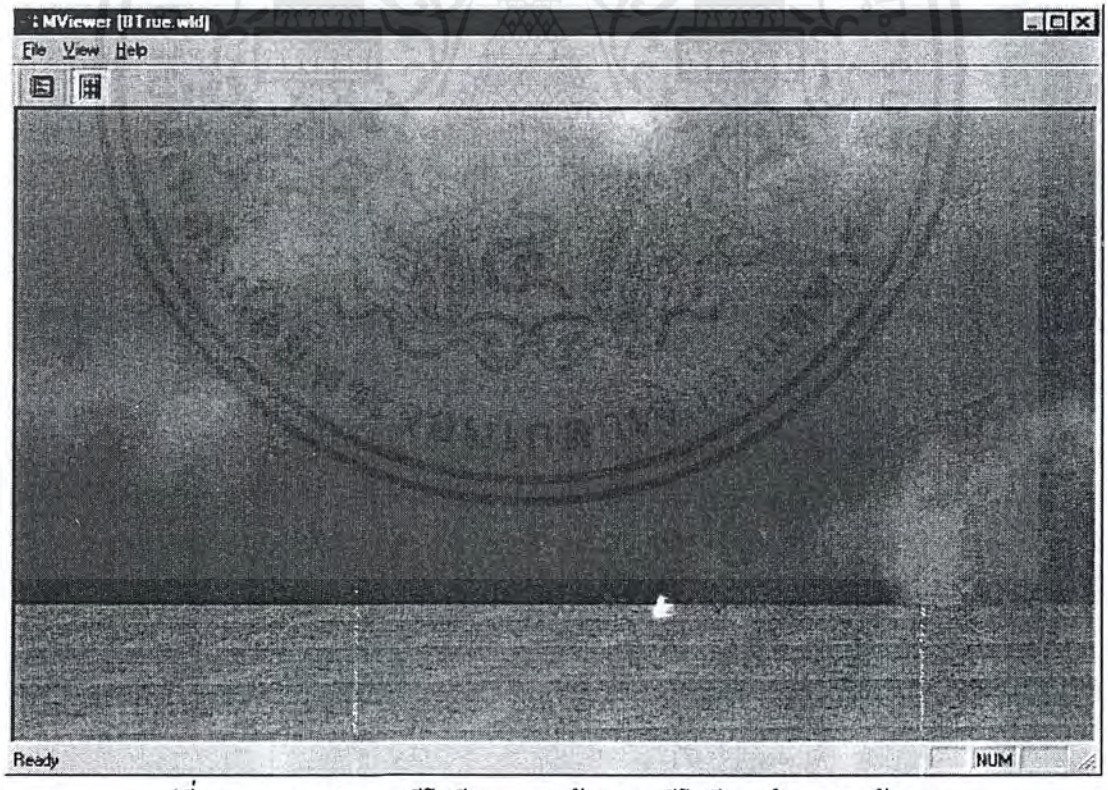
รายละเอียดระบบ	:	AMD Duron 600 MHz, 128 MB SDRAM PC133, Nvidia TNT2/M64 32MB SDRAM, HDD 20 GB, MS Windows 98se
ความละเอียดของหน้าจอ	:	1,024 x 768 (เฉพาะ window ที่แสดงผล 756 x 440)
ความลึกของสี	:	16 บิต

ได้ผลการทดลองดังนี้ :

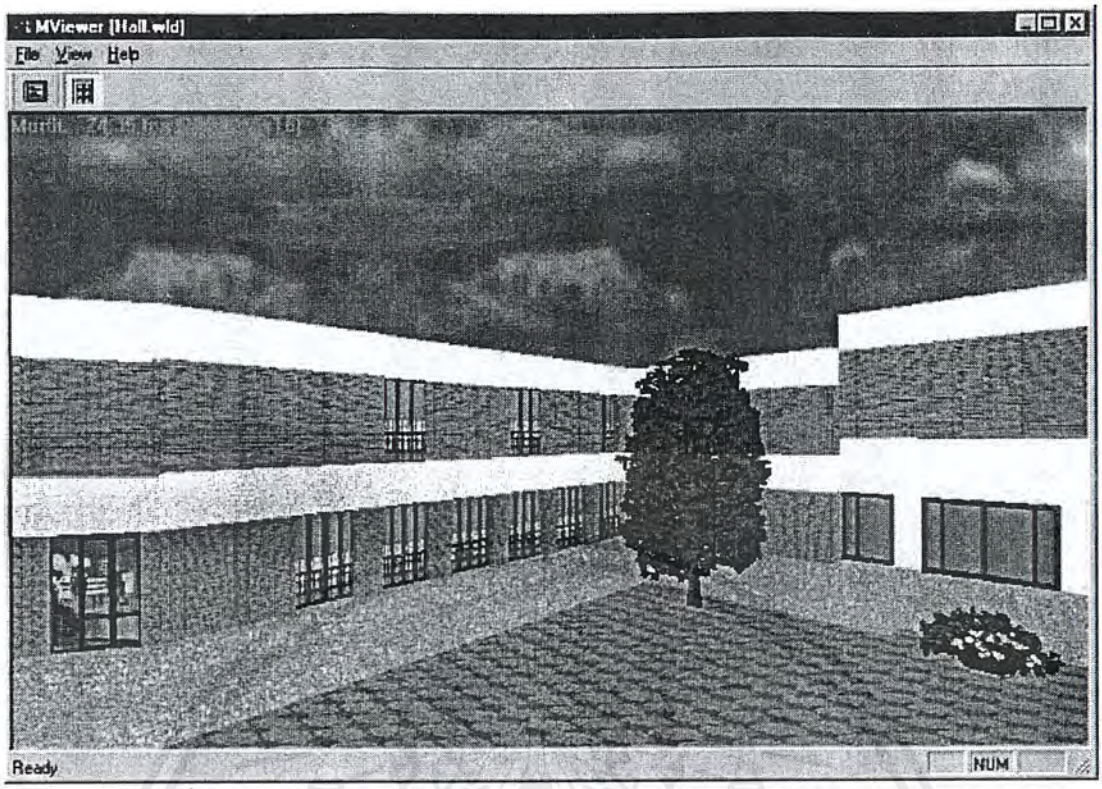


รูปที่ 17-1 แสดง FPS กรณีโพลีกอนรวมน้อย และมีโพลีกอนใน FOV มาก

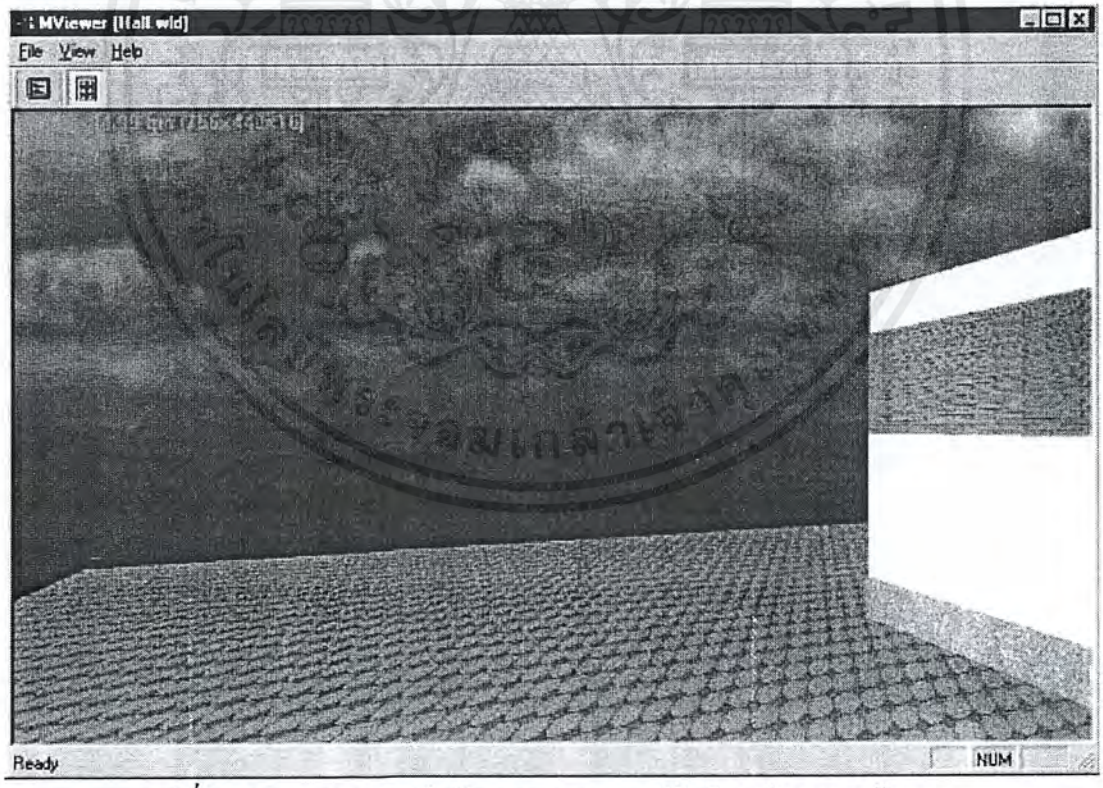
รูปที่ 17-2 แสดง FPS กรณีโพลีกอนรวมน้อย และมีโพลีกอนใน FOV ปานกลาง



รูปที่ 17-3 แสดง FPS กรณีโพลีกอนรวมน้อย และมีโพลีกอนใน FOV น้อยมาก



รูปที่ 17-4 แสดง FPS กรณีโพลีกอนรวมมาก และมีโพลีกอนใน FOV ปานกลาง



รูปที่ 17-5 แสดง FPS กรณีโพลีกอนรวมมาก และมีโพลีกอนใน FOV น้อยมาก

สรุปผลการทดลอง

จากรูป 17-1 – 17-3 ได้ FPS ประมาณ 13, 30 และ 50 FPS ตามลำดับจะเห็นได้ว่าที่ FOV ต่าง ๆ กันจะมีผลต่อ FPS มากในจำนวนโพลีกอนรวมที่เท่ากัน และจำนวนของโพลีกอนรวมเพียงอย่างเดียวไม่ได้บ่งชี้ค่า FPS จะเห็นได้จากรูป 17-4 – 17-5 ที่มีจำนวนโพลีกอนรวมมากกว่าโลก 3 มิติ ที่ใช้ในรูป 17-1 – 17-3 แต่มี FPS ที่ดีกว่าใน FOV ที่พอ ๆ กัน ในที่นี้เราสามารถสรุปได้ว่า FOV มีผลต่อ FPS มากกว่าโพลีกอนรวม แต่มันไม่ได้เป็นปัจจัยเดียวที่จะกำหนด FPS ซึ่งเราจะมาพิจารณาผลกระทบจากด้านอื่นในหัวข้อถัดไป

แนวทางการแก้ไขปัญหา

การสร้างโลก 3 มิติควรจะสร้างให้มีจำนวนโพลีกอนที่น้อย ๆ แต่ใช้ภาพบิตแมปที่สมจริงเข้าช่วย แต่เรื่องที่สำคัญยิ่งกว่าคือ เราต้องพยายามให้จำนวนโพลีกอนใน FOV ของช่วงเวลาใด ๆ ในโปรแกรมให้น้อยที่สุด แต่ฉากต้องยังดูดี ซึ่งเป็นเรื่องที่ยากมาก

17.2 การคำนวณโพลีกอนในโมเดลที่ซับซ้อน

จากหัวข้อที่ผ่านมา อาจเกิดคำถามขึ้นว่าถ้าจำนวนโพลีกอนใน FOV พอ ๆ กันแล้ว เหตุใดจึงทำให้ค่า FPS ต่างกันมาก สาเหตุของกรณีนี้เกิดจากในโลก 3 มิติแต่ละโลกย่อมต่างกัน และมีความซับซ้อนของโมเดลในฉากไม่เหมือนกัน เช่นในโลก 3 มิติที่ใช้ในรูปที่ 17-1 – 17-3 มีจำนวนโพลีกอนรวมน้อยกว่าโลก 3 มิติที่ใช้ในรูป 17-4 – 17-5 และในกรณี FOV มีโพลีกอนพอ ๆ กันตามปกติควรจะได้ค่า FPS ที่ดีกว่า แต่เนื่องจากความซับซ้อนของโมเดลที่ใช้ในฉากมีมากกว่ามากจึงมีผลต่อ FPS มากกว่า

แนวทางการแก้ไขปัญหา

ควรสร้างโลก 3 มิติให้มีความซับซ้อนในระดับที่จำเป็นเท่านั้น แต่ถ้ามีความจำเป็นต้องใช้โมเดลที่ซับซ้อน ควรตรวจสอบว่ามีโพลีกอนที่ไม่จำเป็นมากเกินไปหรือเปล่า เช่น โพลีกอนที่มองไม่เห็นจากมุมใด ๆ , โพลีกอนที่มีเวอร์เทกซ์ร่วมกับโพลีกอนอื่นหลายเวอร์เทกซ์, โพลีกอนที่ควรจะอยู่บนพื้นผิวในระนาบเดียวกันแต่กลับเป็นคนละชิ้น ฯลฯ

17.3 ผลกระทบจากการเรนเดอร์ Dynamic Object

การเรนเดอร์ dynamic object จะใช้ประสิทธิภาพของระบบมากกว่า static object มากเนื่องจากไม่มี BSP ช่วย การเรนเดอร์จึงต้องทำแบบเรียลไทม์ตลอด อีกทั้งถ้า dynamic object เป็น MD2 ซึ่งมีการเคลื่อนไหวอยู่ในตัวของมันเองโดยตลอดจะยิ่งใช้ประสิทธิภาพของระบบมากขึ้นอีก

แนวทางการแก้ไขปัญหา

ควรลดโพลีกอน (และเฟรมของ MD2) ที่ไม่จำเป็นของ dynamic object ออกไป และลดการเคลื่อนไหวที่ไม่จำเป็นออกไป แต่นั่นก็ช่วยไม่ได้มากถ้าจำนวนโพลีกอนของ dynamic object มีมากตั้งแต่ 600 โพลีกอนขึ้นไป

17.4 การลดความลึกของสีของบิตแมปใน Morfit

มอร์ฟิทจะทำการลดความลึกของสีของบิตแมปที่เราใช้ในการทำ texture โดยไม่ว่าเราจะใช้บิตแมปต้นฉบับที่มีความลึกของสีมากเพียงใด มอร์ฟิทก็จะแปลงไปเป็นความลึก 8 บิตเท่านั้นเพื่อประสิทธิภาพที่ดีที่สุด แต่ไม่ใช่เพื่อคุณภาพที่ดีที่สุด

แนวทางการแก้ไขปัญหา

สิ่งที่เราสามารถทำได้ดีที่สุดคือควรเปลี่ยนความลึกของสีภายนอกมอร์ฟิท เช่น อาจทำใน Photoshop จะได้ผลลัพธ์ที่ดีกว่า แต่แนวทางที่จะแก้ให้มอร์ฟิทใช้บิตแมปที่ความลึก 24 บิตในขณะนี้ยังไม่มี

17.5 การควบคุมการเคลื่อนไหวของโมเดลเอ็มดีทู (MD2)

มอร์ฟิทใช้หลาย ๆ เอพีไอในการควบคุมการเคลื่อนไหวของ MD2 เช่น Object API แต่มันยังมีขีดจำกัดในหลาย ๆ เรื่อง ได้แก่

1. ชื่อที่จะนำมาสร้างแฮนเดิลของอนิเมชั่นมีข้อจำกัดว่าต้องเป็นชื่อของอนิเมชั่นที่เป็นไปตามมาตรฐานของ MD2
2. การตรวจสอบเฟรมปัจจุบันว่าเป็นเฟรมที่เท่าไรเป็นในระดับทศนิยมซึ่งไม่เป็นที่ต้องการเพราะเราต้องการตรวจสอบลำดับของเฟรมที่เป็นจำนวนเต็ม
3. การตรวจสอบว่าอนิเมชั่นใด ๆ เล่นครบรอบไปหรือยัง
4. การสั่งให้กระทำการใด ๆ หลังจากอนิเมชั่นนั้น ๆ ครบรอบ

แนวทางการแก้ไขปัญหา

1. ควรใช้อนิเมชั่นเท่าที่จำเป็นและเป็นไปตามมาตรฐาน
2. ใช้การตรวจสอบเป็นเรนจ์ของเฟรมแทนจำนวนเต็ม
3. ตัดเฟรมสุดท้ายของอนิเมชั่นที่ต้องการมา 1 เฟรม แล้วสร้างเป็นอนิเมชั่นใหม่ซึ่งเราให้แทนที่อนิเมชั่นเดิมเมื่อเล่นครบรอบ จากนั้นตรวจสอบจากแฮนเดิลของอนิเมชั่นหลัง
4. ใช้การตรวจสอบแบบในข้อ 4 แล้วค่อยสั่งให้กระทำการใด ๆ

17.6 การเข้าถึงในระดับโพลีกอนของโมเดลเอ็มดีทู

ในขณะนี้มอร์ฟิทยังไม่เปิดเผยการทำงานในระดับโพลีกอนของ MD2 ออกมาเลย จึงเป็นไปไม่ได้ที่จะสั่งการต่อโมเดล MD2 ในระดับโพลีกอน ซึ่งทำให้การเปลี่ยนอาวุธให้กับโมเดล, การตรวจสอบการชนในระดับโพลีกอน, การเปลี่ยน texture ในแบบเรียลไทม์ ฯลฯ เป็นเรื่องที่ยากมากหรือแทบเป็นไปไม่ได้ในปัจจุบัน

17.7 การจัดการกล้องกับโมเดล

เมื่อเราเลื่อนกล้องเข้าไปใกล้โมเดลใด ๆ ไม่ว่าจะเป็น static หรือ dynamic มาก ๆ แล้วจะทำให้การแสดงผลโมเดลนั้น ๆ ผิดพลาดคือ ไม่แสดงเลย หรือแสดงเพียงบางส่วน ซึ่งเราไม่ต้องการ

แนวทางการแก้ไขปัญหา

พยายามอย่าให้กล้องกินเข้าไปในเนื้อของโมเดล โดยการกำหนดระยะห่างที่พอเหมาะ

17.8 การตรวจสอบการชนโดยใช้ฟังก์ชันของมอร์ฟิท

โดยปกติการตรวจสอบการชนของมอร์ฟิทจะใช้เส้นหรือใช้กล่องในการตรวจสอบ ซึ่งค่อนข้างเพียงพอในการใช้กับโปรแกรมทั่ว ๆ ไป แต่ในบางโปรแกรมที่ต้องใช้การตรวจสอบการชนในปริมาณมาก และมีความถี่ในการตรวจสอบที่สูง จะใช้ประสิทธิภาพของระบบค่อนข้างมาก จนแทบจะรันไม่ได้ในทางปฏิบัติ

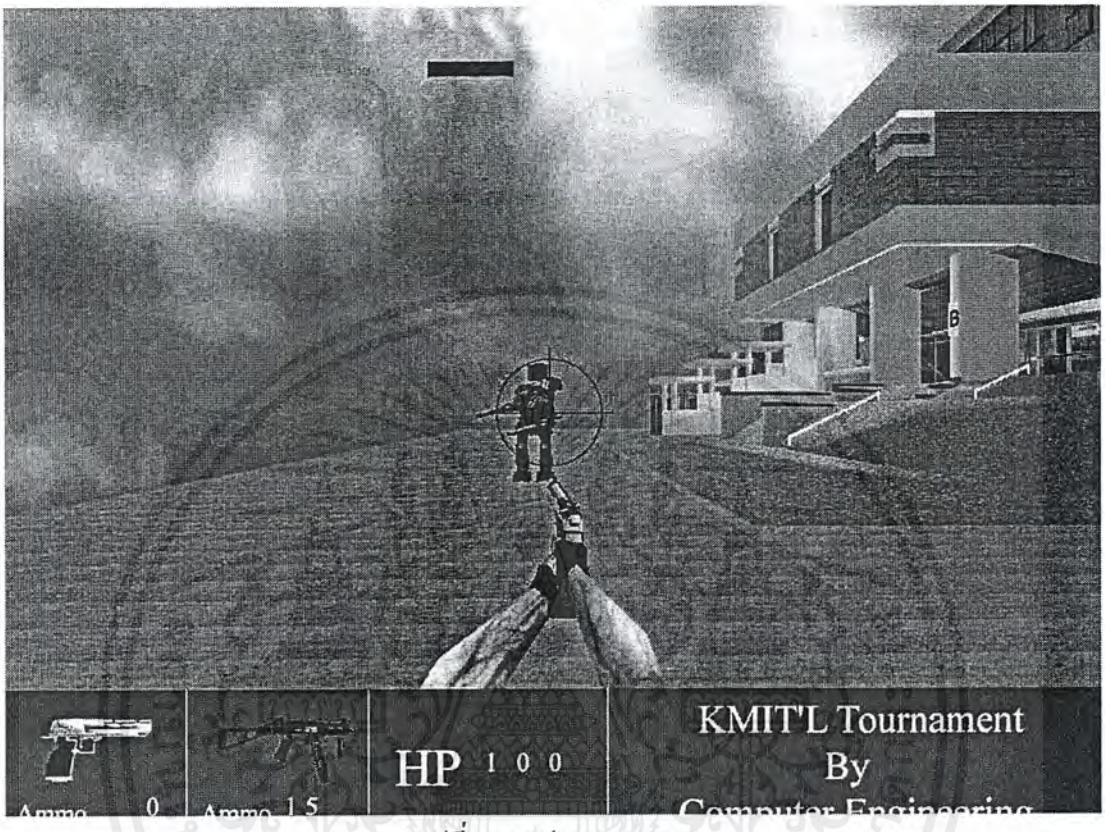
แนวทางการแก้ไขปัญหา

ถ้าจำเป็นจริง ๆ ควรลดปริมาณการตรวจสอบการชนในขณะใดขณะหนึ่งให้น้อยที่สุดเท่าที่จะเป็นไปได้

บทที่ 18
ผลลัพธ์ของโครงงาน

18.1 รูปแบบของเกม

เป็น First-Person Shooting ที่กำลังเป็นที่นิยม ดังรูป



รูปที่ 18.1 รูปแบบของเกม

18.2 การควบคุมเกม

ปุ่ม W, S, A, D เป็นการเดินไปบนล่างซ้ายขวาตามลำดับ

ปุ่ม R เป็นการบรรจุกระสุนใหม่

การหมุนเมาส์ เป็นการหมุนทิศทางของตัวละคร

การคลิ้กเมาส์ซ้ายเป็นการยิง

ปุ่ม 1, 2 เป็นการเลือกอาวุธ

ปุ่ม N เป็นการกำหนดแสง

ปุ่ม P ใช้เริ่มเกมใหม่สำหรับผู้เล่น

ปุ่ม M ใช้เริ่มเกมใหม่สำหรับศัตรู

บทที่ 19
บทวิจารณ์และสรุป

19.1 ประเมินผล

โครงงานนี้นับว่าประสบความสำเร็จในระดับหนึ่ง ถึงแม้ว่า FPS ที่ได้จะค่อนข้างน้อย แต่ก็ทำให้ได้เข้าใจถึงหลักการเบื้องต้นเกี่ยวกับระบบ 3 มิติ และที่สำคัญคือได้ศึกษาและทดลองใช้เทคโนโลยีที่มีอยู่จริงในปัจจุบัน

ในขณะนี้มอร์ฟิทยังมีความสามารถทั่ว ๆ ไปไม่มากนัก แต่มีความง่ายในการใช้งาน จึงเหมาะกับงานเพื่อการศึกษามากกว่าที่จะนำไปใช้เขียนเกมให้ได้อรรถรสจริง ๆ ส่วนไดเร็กเอกซ์นั้นเป็นเทคโนโลยีที่มีมานานและค่อนข้างลงตัวแล้วจึงเหมาะสมในการใช้งานมาก แม้ว่าบางส่วนประกอบจะใช้งานยากแต่ก็เหมาะที่จะนำไปใช้พัฒนาโปรแกรมที่ต้องการความสามารถสูง (แต่จะต้องใช้เวลาในการพัฒนานานมาก) ดังนั้นถ้าต้องการทำโครงงานที่ใช้ไดเร็กเอกซ์ทั้งหมดภายในเวลาหนึ่งปีจึงควรเป็นโปรแกรมง่าย ๆ ที่ไม่มีส่วนประกอบซับซ้อนจะเหมาะสมกว่า

19.2 แนวทางการพัฒนาต่อ

- โปรแกรมเพื่อการศึกษาโพลีกอนไม่มาก -> มอร์ฟิท เอพีไอ
- โปรแกรมที่ใช้เวลาพัฒนาได้นานมาก -> ไดเร็กเอกซ์
- โปรแกรมเกม First Person Shooting -> เอนจิ้นที่ได้มาตรฐานเช่น Quake3 engine, Quake2 engine
- Modification จากเกมต่าง ๆ -> Half-Life SDK

บรรณานุกรม

[1] Andre LaMothe (1998) : *"Windows Game Programming for Dummies"*, IDG Books Worldwide, Inc. , 1998.

[2] Bradley Bargen and Peter Donnelly (1998) : *"Inside DirectX"*, Microsoft Press, 1998

[3] Stan Trujillo (1996) : *"Cutting Edge Direct3D Programming"*, Coriolis Group Book,1996.

[4] ชัยวัฒน์ คำรัตน์ (1999) : *"3D Game Programming with DirectX"*, Computer Age Technology Co,. Ltd. , 1999.

[5] Morfit Staff (1999): *"3D Programming Using the Morfit 3D Engine"*, http://www.3dstate.com/