สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

การพัฒนาเกม 3 มิติ (1)

3D Game Development (1)

นายอวิรุทธ์ อำพันธุ์รัตน์

นายอาทิตย์ ศรีทองอินทร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เลขหมู่.....
เลขทะเบียน..... 42758
วัน, เดือน, ปี 10 มิ.ย. 2545

b.....
i.....

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

กณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2543

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การพัฒนาเกม 3 มิติ (1)

3D Game Development (1)

โดย

นายอวิรุทธ์ อำพันธุ์รัตน์

นายอาทิตย์ ศรีทองอินทร์

อาจารย์ที่ปรึกษา

ดร. วรวัฒน์ ลิ้มโภคา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2543

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ปริญญานิพนธ์ปีการศึกษา2543

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาเกม 3 มิติ

3D Game Development

ผู้จัดทำ

1. นาย อวิรุทธ์ อำพันธุ์รัตน์ รหัสประจำตัว 40010977
2. นาย อาทิตย์ ศรีทองอินทร์ รหัสประจำตัว 40010998

อาจารย์ที่ปรึกษา

(ดร. วรวัฒน์ ลิ้มโภคา)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

การพัฒนาเกม 3 มิติ

นาย อวิรุทธ์ อำพันธุ์รัตน์ 40010977
นาย อาทิตย์ ศรีทองอินทร์ 40010998
ดร. วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษา
ปีการศึกษา 2543

บทคัดย่อ

ในปัจจุบันความก้าวหน้าทางด้านฮาร์ดแวร์ของคอมพิวเตอร์นั้นเป็นไปอย่างรวดเร็วจนสามารถรองรับเทคโนโลยีการแสดงผลแบบ 3 มิติ แบบเรียลไทม์ได้ ด้วยเหตุนี้จึงทำให้งานในหลายๆ ด้าน นำเอาความสามารถนี้ไปอิมพลีเมนต์ ไม่ว่าจะเป็น งาน CAD/CAM, 3D Animation, 3D Modeling, Hardware Prototype, การนำเสนอ หรือ แม้กระทั่งการผ่อนคลาย ก็เช่นกัน ความต้องการในสิ่งเหล่านี้ทำให้ในหลายๆ หน่วยงานพยายามพัฒนาการนำเทคโนโลยีนี้ไปใช้

ซึ่งในโครงงานนี้จะศึกษาในเรื่องการสร้างแอพพลิเคชันที่แสดงผลภาพ 3 มิติ ที่เป็นที่รู้จักกันอย่างกว้างขวาง และ มีผู้บริโภคเป็นจำนวนมาก แอพพลิเคชันนี้ก็คือ เกมนั่นเอง

เนื่องจากเกมเป็นแอพพลิเคชันชนิดหนึ่ง เพราะฉะนั้นการพัฒนาเกมจึงมีปัญหาเช่นเดียวกันกับการพัฒนาแอพพลิเคชัน เช่น ข้อจำกัดด้านเวลา, ทรัพยากรที่ต้องใช้ และ อื่นๆ ผู้ดำเนินธุรกิจในการผลิตเกมจึงพยายามนำวิธีการที่ใช้แก้ไขปัญหาในการผลิตซอฟท์แวร์ เช่น การโปรแกรมเชิงวัตถุ หรือ เฟรมเวิร์ค เข้ามาใช้ แต่เนื่องจากผู้ใช้แอพพลิเคชันนี้มักไม่ค่อยมีความต้องการเชิงฟังก์ชันจากแอพพลิเคชัน ในขณะที่มีความต้องการแบบไม่เป็นฟังก์ชันจากแอพพลิเคชันเป็นหลัก เช่น ความสวยงาม และ การเคลื่อนไหวของตัวละคร, เพลงประกอบ และ ความน่าสนใจของเนื้อเรื่อง

ด้วยเหตุผลนี้เองปริญญานิพนธ์ฉบับนี้จึงจะไม่เน้นแต่เพียงการโปรแกรมมิงเท่านั้น แต่จะกล่าวถึงภาพรวมของแนวทางในการพัฒนาเกม 3 มิติ ในปัจจุบัน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3D Game Development

Awirutdh Amphunrat

Arthid Srithongin

Prof. Dr. Vorawat Limpoka Advisor

Now, the grow of hardware technology of computer is not a linear function any more. It's greater than. The hardware technology now enough to implement 3D rendering and display in real-time. Because of this many of occupation use computer that have a 3D capacity to implement his work. Example CAD/CAM, 3D Animation, 3D Modeling, Hardware Prototype and Entertainment. The demand of this make provider and researcher try to develop and implement rapidly.

This project is aim to study about how to develop 3D-Application that very wide recognize and have a greater of supplier called "3D-Games"

"Games" is an application. Because of this Game Development have a same kind of trouble as general Application Development (such as: Develop Time, Resource consumption and eg..). So the Game Development Company use the same technique, that can reduce problem, as application development (such as: Object Oriented Programming and Framework). By the way the user didn't have much functional-requirement, All they care is the game meet them un-fuctional requirement (such as: did character in game beautiful or handsome, did the movement of object in game is so real?, Soundtrack is melody? or Is story good plot?)

So this Thesis is not intend to research in 3D Game Development in term of application only but in term of necessary thing that effect to 3D Game Development.

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือ และ ร่วมมือจากหลาย ๆ ฝ่ายด้วยกัน บุคคลแรกที่ต้องกล่าวถึงเพราะเป็นส่วนสำคัญที่ทำให้ปริญญานิพนธ์นี้เสร็จลงได้ก็คือ อาจารย์ วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษาปริญญานิพนธ์ ที่ให้ความเอาใจใส่ แนะนำ และช่วยเหลือเสมอมา, ฝ่ายสนับสนุนของเกมเอนจินมอร์ฟีตที่คอยช่วยเหลือผ่านทางจดหมายอิเล็กทรอนิค รวมถึงกลุ่มพัฒนาเกม 3 มิติ อีกกลุ่ม ที่ช่วยกันแก้ไขปัญหาที่เกิดขึ้นในการดำเนินงาน ซึ่งต้องขอขอบพระคุณเป็นอย่างมาก

และต้องขอขอบพระคุณบุคคลสำคัญที่สุดที่ทำให้ข้าพเจ้ามีวันนี้ ก็คือ บิดา มารดา อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูผู้เขียนมาเป็นอย่างดี พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่ และ ยังให้กำลังใจ เอาใจใส่เสมอมา ในทุก ๆ ด้านอันหาที่เปรียบมิได้ ข้าพเจ้าขอระลึกในพระคุณอันสุดประมาณ และขอกราบขอบพระคุณมา ณ ที่นี้

นายอวิรุทธ์ อำพันธุ์รัตน์
นายอาทิตย์ ศรีทองอินทร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ

(ต่อ)

หน้าที่

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ
(ต่อ)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ

(ต่อ)

หน้าที่

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญ

(ต่อ)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สารบัญภาพ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ในปัจจุบันอุตสาหกรรมเกมเป็นอีกอุตสาหกรรมซอฟท์แวร์หนึ่งที่ให้ผลตอบแทนสูง และ มีการแข่งกันกันมาก แต่ในประเทศไทยกลับไม่มีการสนับสนุนการพัฒนาซอฟท์แวร์ประเภทนี้อย่างจริงจังอย่างมากที่สุดก็เป็นการจัดประกวดซอฟท์แวร์ที่อนุญาติให้มีผู้ร่วมโครงการเพียง 2-3 คน ให้ทุนเพียงเล็กน้อยในการพัฒนา หรือ ยกภาระหน้าที่ในด้านค่าใช้จ่ายทั้งหมดให้ทีมพัฒนา พวกผมผู้ดำเนินงานวิจัยชิ้นนี้จึงเห็นว่าสังคมไทยยังไม่ตระหนักถึงส่วนต่างๆ ที่สำคัญในการพัฒนาซอฟท์แวร์ประเภทนี้ไม่ว่าจะเป็น ผู้ออกแบบตัวละคร (Character Design), ผู้ออกแบบกราฟฟิค (Graphic Design), ผู้แต่งเพลง, ผู้แต่งเรื่อง, ผู้ออกแบบเกม (Game Concept Design), ผู้สร้างโมเดล 3 มิติ (3D Modeler), แอนิเมเตอร์ 2 มิติ และ 3 มิติ (2D Animator and 3D Animator) และ อื่นๆ ทำให้เกมที่คนไทยผลิตขึ้นมาไม่สามารถแข่งขันในตลาดได้ เนื่องจากหลายสาเหตุที่เกิดจากการละเลยความสำคัญในด้านอื่นๆ นอกจากการโปรแกรม ทำให้เกิดข้อเสียเปรียบในการแข่งขัน เช่น ตัวละครขาดความดึงดูด, ยูสเซอร์อินเทอร์เฟสของเกมอยากต่อการใช้, ภาพไม่สวยงาม, ภาพ หรือ เพลงที่ประกอบในเกมนำมาจากแหล่งอื่นๆ และไม่ได้ขอลิขสิทธิ์ อาจจะทำให้เกิดปัญหาการฟ้องร้องเมื่อนำออกไปวางตลาด, โมเดล 3 มิติ ไม่สวยงาม ไม่เหมือนจริง หรือ เหมือนเกม 3 มิติพ้นสมัย, การเคลื่อนไหวของวัตถุในเกมไม่สมจริง ไม่ต่อเนื่อง ก่อให้เกิดความวิงเวียน หรือ รำคาญขณะเล่น, เนื้อเรื่องไม่น่าติดตาม หรือ น้ำเน่า และ อื่นๆ

ในงานวิจัยนี้จึงกล่าวถึงความจำเป็นอื่นๆ ในการพัฒนาเกมนอกจากการโปรแกรม และ ชี้ให้เห็นถึงความสำคัญของบุคคลากรที่พึงมีในการพัฒนาซอฟท์แวร์ชนิดนี้ รวมถึงปัจจัยทางเทคโนโลยีอื่นๆ ไม่ว่าจะเป็นเรื่องการเลือกใช้เอพีไอ, การเลือก 3D โมเดลลิงทูล, หลักการเลือกเกมเอนจิ้น, แนวโน้มความต้องการของผู้ใช้ที่มีต่อซอฟท์แวร์ชนิดนี้, ความต้องการของผู้ผลิตที่มีต่อเกมเอนจิ้น ซึ่งสิ่งเหล่านี้ล้วนมีส่วนที่จะทำให้เกมที่ผลิตขึ้นมาสามารถแข่งขันกับผู้พัฒนารายอื่นๆ ได้

1.2 จุดประสงค์ของงานวิจัย

1.ศึกษาแนวทางการพัฒนาแอพพลิเคชันเกม 3 มิติ ที่เป็นที่นิยมใช้ในอุตสาหกรรมเกม ตลอดจนในหมู่ผู้พัฒนารายย่อย

2.ศึกษาส่วนประกอบ และ สิ่งที่จำเป็นในการพัฒนา และ ผลิตเกมในเชิงอุตสากรรม

3.นำความรู้ที่ได้จากการดำเนินงานเพื่อตอบสนองต่อวัตถุประสงค์ข้อที่ 1 และ 2 มาใช้ค้นหา และเลือกเอพีไอที่เหมาะสมพัฒนาแอพพลิเคชันเกม 3 มิติ ในโลกปัจจุบัน

4.พัฒนาแอพพลิเคชันเกม 3 มิติ แบบต่อสู้กัน 2 คน โดยใช้เอพีไอที่ได้เลือกไว้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

1.3 ขอบเขตของงานวิจัย

1.งานวิจัยชิ้นนี้มุ่งที่จะวิจัย และ นำเสนอถึงส่วนต่างๆ ที่จำเป็นในการผลิตเกม ไม่ใช่ซอฟท์แวร์ จึงไม่เน้นพัฒนาความสมบูรณ์ของซอฟท์แวร์เป็นหลัก แต่เป็นการทดสอบว่าแนวทางในการพัฒนาเกม 3 มิติ ที่ได้จากการศึกษานั้นเหมาะสม หรือไม่ แต่เนื่องจากทางผู้วิจัยไม่มีประสบการณ์ในการโปรแกรมเป็นหลัก งานในส่วนอื่นๆ นอกเหนือจากด้านโปรแกรมจึงไม่ได้มีคุณภาพถึงระดับที่ผู้ใช้ทั่วไปคาดหวัง

2.สร้างโมเดลตัวละคร และ ฉากขึ้นเองทั้งหมดโดยฉากจะให้เป็นบริเวณใดบริเวณหนึ่งของสถาบัน แต่เนื่องจากผู้วิจัยไม่ใช่ผู้เชี่ยวชาญทางด้านการสร้างวัตถุ 3 มิติ ทำให้ ฉาก และ ตัวละครที่ได้ไม่ดีเท่าที่ควร

3.เกมที่พัฒนามีลักษณะคล้ายเกม Street Fighter โดยตัวละครในเกมจะต้องสามารถโจมตีฝ่ายตรงข้ามได้ และ แสดงถึงอาการบาดเจ็บได้เมื่อถูกโจมตี

1.4 ผลที่คาดว่าจะได้รับ

1.แนวทางการพัฒนาเกม 3 มิติในปัจจุบัน

2.บุคลากร และ ส่วนที่จำเป็นในการพัฒนาเกมที่ตอบสนองต่อความต้องการของตลาดในปัจจุบัน

3.คู่มือการใช้งานเอพีไอ หรือ เกมเอนจิ้นที่เลือกใช้

4.แอพพลิเคชันเกมต่อสู้ 3 มิติ

5.แนวทางการแก้ไขปัญหาบางปัญหาที่เกิดขึ้นในการดำเนินงาน หากปัญหานั้นสามารถแก้ไขได้

1.5 วิธีการดำเนินงาน

1.ค้นคว้าข้อมูลจากอินเทอร์เนตตามเวปเพจของผู้ผลิต หรือ เวปเพจที่สนับสนุนการพัฒนาแอพลิเคชันชนิดนี้ รวมถึงการสอบถามข้อมูลจากบุคคลเหล่านี้ผ่านทางจดหมายอิเล็กทรอนิค

2.สอบถาม หรือ สัมภาษณ์ ผู้ผลิต, ผู้พัฒนา, ผู้ที่สนใจ หรือ ผู้ที่ชอบใช้แอพพลิเคชันประเภทนี้ถึงส่วนประกอบต่างๆ ที่คิดว่าเป็นส่วนสำคัญ รวมถึงเหตุผลประกอบ

3.เลือกเครื่องมือที่ใช้ในการพัฒนาแอพพลิเคชัน และ ศึกษาถึงวิธีการใช้งาน

4.ออกแบบโครงสร้าง และ รูปแบบของเกม

5.จัดเตรียม หรือ สร้างสิ่งที่เป็นองค์ประกอบของเกม เช่น ภาพประกอบ, วัตถุ 3 มิติ และ เสียงแอฟเฟค

6.พัฒนาเกมในเชิงโปรแกรม (โค้ดดิง)

7.ตรวจสอบ และ แก้ไขข้อผิดพลาด

1.6 สาเหตุที่เลือกใช้เกมเอนจิ้นในการพัฒนา

ในปัจจุบันการผลิตแอพพลิเคชันขนาดกลางถึงใหญ่ หรือ แม้กระทั่งขนาดเล็กก็ตามที นิยมใช้เฟรมเวิร์ค หรือ เอพีไอ เข้ามาช่วยในการผลิต เพื่อลดความซับซ้อนของโค้ด และ ทำให้สามารถผลิตเกมออกมาได้มากขึ้นโดยใช้ทรัพยากรน้อยลง และ เนื่องจากเกมเป็นแอพพลิเคชันรูปแบบหนึ่ง จึงมีหลายบริษัทที่

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ผลิตเฟรมเวิร์คสำหรับการพัฒนาเกมออกมา โดยเฟรมเวิร์คสำหรับพัฒนาเกมนี้ถูกเรียกกันในหมู่นักพัฒนาเกมว่า "เกมเอนจิ้น"

1. มาตรฐานซอฟท์แวร์ และ ฮาร์ดแวร์มีการเปลี่ยนแปลงอย่างไม่หยุดยั้ง ทั้งการ์ดเร่งการแสดงผล 3 มิติ และ API ต่างก็พัฒนากันอย่างรวดเร็ว ในทุกๆ ไตรมาส ถ้าไม่มีการ์ดรุ่นใหม่ออกมา ก็มีการปล่อย API ใหม่ๆ หรือ เวอร์ชันใหม่ ออกมา การที่จะทำเกมให้สนับสนุน API และ การ์ดแสดงผล 3 มิติ ให้มากที่สุด นั้นต้องใช้เวลามากเกินไป ทำให้ผู้พัฒนามีเวลาที่จะไปพัฒนาตัวเนื้อเกมน้อยลง

2.เวลาที่ใช้ในการศึกษา เทคโนโลยีที่ดีนอกจากจะต้องตอบสนองความต้องการของเราได้แล้ว ควรจะง่ายต่อการทำความเข้าใจด้วย เพราะการที่จะเข้าใจและเป็นผู้เชี่ยวชาญ API ได้นั้นจำเป็นต้องใช้เวลา โดยเฉพาะอย่างยิ่งถ้าจะศึกษา API ทั้งหมดที่มีอยู่ในท้องตลาดจนเชี่ยวชาญย่อมเป็นไปได้ยาก

3. เวลาที่ใช้ในการผลิต ถึงแม้คุณจะเป็นผู้เชี่ยวชาญ API นั้นก็ตาม แต่ถ้าการสร้างซอฟท์แวร์โดยใช้ API นั้น ใช้เวลามาก และต้องเขียนทุกๆ ส่วนขึ้นมาใหม่ทุกครั้ง ทั้งๆ ที่มันควรจะนำกลับมาใช้ได้ ทำให้เสียเวลาส่วนนั้นไปโดยเปล่าประโยชน์ ในการสร้างเกมก็เช่นกัน การตรวจสอบการชนกันของวัตถุ, การบริหารหน่วยความจำสำหรับเท็กซ์เจอร์ (Texture) และซอฟท์แวร์เรนเดอร์ (Software Render) ล้วนเป็นสิ่งที่มีผู้พัฒนามาอยู่แล้ว การพัฒนาขึ้นจาก 0 ใหม่ นอกจากจะเสียเวลาแล้ว ยังอาจทำไม่ได้เท่าของที่มีอยู่เดิมด้วย

4. ความสามารถ API บางชนิดที่ทำหน้าที่ติดต่อกับฮาร์ดแวร์เป็นหลัก เช่น DirectX RM (Retain Mode) มีประสิทธิภาพต่ำ และ ไม่สามารถจัดการกับโลก 3 มิติ ที่มีขนาดใหญ่มากๆ ได้ และ DirectX IM ที่มีประสิทธิภาพสูงกว่าแต่ไม่สนับสนุนการทำแอนิเมชันโมเดล 3 มิติ

5. เครื่องมือ เอนจิ้นที่ดีมักจะมีทูลช่วยพัฒนาที่เป็นกูอิ (GUI หรือ Graphic User Interface) มาด้วย ทำให้ลดเวลาในการพัฒนาลงไปได้มาก เช่น การโมเดลตึกขึ้นมาตึกหนึ่งโดยใช้ทูลที่มีลายผนัง, หน้าต่าง, กระจก, โต๊ะ, เก้าอี้ และอื่นๆ แบบ Drag-Drop โดยใช้กูอิทูลกับ การโมเดลตึกโดยใช้ AutoCAD ซึ่งต้องวาดทุกอย่างขึ้นมาทั้งหมด

1.7 สาเหตุที่เลือกเกมเอนจิ้นมอร์ฟิท (Morfit) ในการพัฒนา

1. เกมเอนจิ้นมอร์ฟิทเป็นฟรีแวร์ เนื่องจากเกมเอนจิ้นที่มีขายในปัจจุบันที่น่าเชื่อถือก็มีราคามากกว่า 5,000 เหรียญสหรัฐ ในขณะที่เอนจิ้นที่มีราคาต่ำกว่านั้นมีคุณภาพพอๆ กับ มอร์ฟิท มอร์ฟิทจึงเป็นทางเลือกที่ดีที่สุด แต่ก็หมายได้หมายความว่ามอร์ฟิทเป็นเอนจิ้นที่ดีพอสำหรับการพัฒนาเกม 3 มิติ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

2. ทางมอร์ฟิทยังมีทีมงาน และ เวปเพจสนับสนุนที่จะให้ความช่วยเหลือ เมื่อเกิดปัญหาในการใช้เกมเอนจิ้น เราสามารถปรึกษาหาวิธีแก้ไขจากทางมอร์ฟิท หรือ ขอความช่วยเหลือจากผู้ใช้รายอื่นที่เคยพบปัญหาเดียวกัน

3. มีผู้ใช้แพร่หลายอยู่ในระดับหนึ่ง นอกจากจะทำให้วางใจในระดับหนึ่งเนื่องจากถ้ามอร์ฟิทไม่ใช่เอนจิ้นที่ดีในระดับที่น่าพอใจ ก็ไม่น่าจะมีผู้ใช้ขนาดนี้ นอกจากนั้นยังมีข้อดีอื่นอีก คือ เมื่อเอนจิ้นมีข้อผิดพลาดผู้ใช้ก็จะรายงานไปยังผู้ผลิต เมื่อมีผู้ใช้จำนวนมาก ก็หมายวามว่ามีผู้ตรวจสอบ และ รายงานข้อผิดพลาดไปยังผู้ผลิต เพื่อให้ผู้ผลิตได้แก้ไขข้อผิดพลาดนั้น ข้อผิดพลาดของเอนจิ้นก็ต้องน้อยลง เมื่อผู้ผลิตออกเอนจิ้นรุ่นใหม่มา (ปัจจุบันมอร์ฟิทออกถึงเวอร์ชัน 5 แต่ ในเวอร์ชัน 5 เป็นเวอร์ชันที่ผลิตออกขายซึ่งไม่ได้เป็นฟรีแวร์แล้ว)

4. สนับสนุน และ สามารถเข้าถึงข้อมูลแอนิเมชันของโมเดล 3 มิติ เอ็มดี2 (.md2) ซึ่งเป็นโมเดล 3 มิติ ที่นิยมใช้ในเกม 3 มิติ บนพีซี โดยโมเดล 3 มิติชนิดนี้สามารถสร้างได้โดย 3D โมเดลลิงทูลชื่อ "มิลค์เชปทรีดี" (Milkshape 3D) ซึ่งมีราคาถูก ใช้งานง่าย

1.8 ปัญหาที่คาดว่าจะพบ

1. ไม่สามารถแก้ไขปัญหาบางอย่างได้ หากปัญหาที่เกิดมีสาเหตุมาจากข้อจำกัด หรือ ข้อผิดพลาดของเกมเอนจิ้น เนื่องจากมอร์ฟิทเปิดเผยแต่เพียงอินเทอร์เฟสของเอพีไอเท่านั้น

2. เกมที่พัฒนามีข้อจำกัดหลายด้าน เนื่องจากข้อจำกัดบางประการของมอร์ฟิท เช่น สนับสนุนสีของเท็กซ์เจอร์ 8 บิทเท่านั้น, ไม่สนับสนุนโมเดลเอ็มดีทรี (.md3) ซึ่งเป็นโมเดลที่ใช้กันอยู่ในปัจจุบัน (เช่น เกม Quake III), ไม่สามารถเรียกใช้คุณสมบัติเฉพาะของการ์ด 3 มิติได้

3. ประสิทธิภาพของเกมต่ำ เนื่องจากสาเหตุหลายอย่าง ไม่ว่าจะเกิดจากการขาดประสบการณ์ของผู้พัฒนา หรือ เนื่องจากเกมเอนจิ้นมอร์ฟิทมีประสิทธิภาพต่ำ

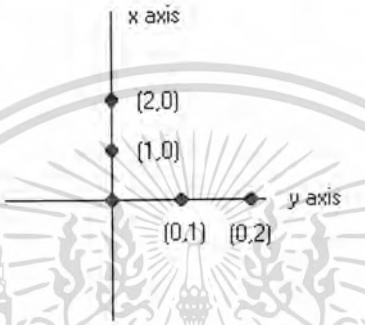
4. วัตถุ 3 มิติในเกมดูไม่เหมือนจริง, ไม่สวยงาม หรือ การเคลื่อนไหวของวัตถุไม่สมจริง เนื่องจากผู้พัฒนาไม่ได้เป็นผู้เชี่ยวชาญในการสร้างโมเดล 3 มิติ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 2
พื้นฐานมอร์ฟิท

2.1 โคออดิเนต 2 มิติ

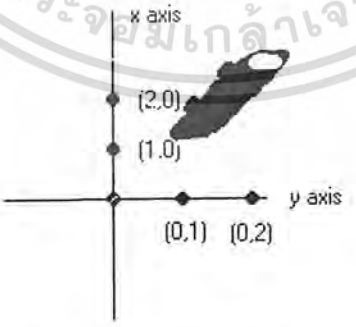
การที่จะเข้าใจระบบโคออดิเนต 3 มิติ ต้องเข้าใจระบบโคออดิเนต 2 มิติก่อน จากรูปที่ 9 แกน x ค่าบวก คือ เส้นตรงตรงที่เริ่มจากจุดออริจินผ่านจุด (1,0) หรือสามารถเขียนได้อีกอย่างว่า เวคเตอร์ [1,0] ส่วนแกน y ค่าบวก คือ เส้นตรงตรงที่เริ่มจากจุดออริจินผ่านจุด (0,1) หรือสามารถเขียนได้อีกอย่างว่า เวคเตอร์ [0,1] ส่วนแกน x ค่าลบจะอยู่ใต้จุดออริจิน และ แกน y ค่าลบจะอยู่ทางด้านซ้ายของจุดออริจิน



รูปที่ 2-1 แกน x-y ของระบบโคออดิเนต 2 มิติ

2.1.1 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 2-2 ระบบโคออดิเนตของวัตถุเทียบกับโลก (World Space)

2.1.2 Object Space เป็นระบบโคออดิเนตของออบเจ็คนั้นเมื่อรถเคลื่อนที่ไปข้างหน้า แสดงว่ามันมีทิศทางในแกน x เสมอ การที่จะทำเช่นนี้จึงต้องมีแกนอีกแกนที่เป็นอิสระจากแกนอื่นๆ กำหนดให้แกน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

นั้นเป็นแกนเฉพาะของรถเท่านั้น กำหนดให้แกน x ด้านบวกชี้ออกทางด้านหลังของรถ และแกน y ชี้ออกจากทางซ้ายของรถดังรูปที่ 2-3 และทิศทางที่รถหันหน้าไป คือ [-1,0] เสมอ

รูปที่ 2-3 Object Space

ตามรูปที่ 2-2 และ 2-3 จะพบว่าในระบบโคออดิเนตของวัตถุรถ รถจะเคลื่อนที่ไปข้างหน้าตามทิศทาง [-1,0] แต่ในโคออดิเนตของโลก รถมีทิศทาง [1,1]

ระบบโคออดิเนตของโลกแกน x และ y จะแทนถึง เวคเตอร์ [1,0] และ [0,1] ตามลำดับ และแกนของวัตถุก็จะเคลื่อนตามวัตถุไปเสมอ จากรูปที่ 9 แกน x-y ของรถจะเท่ากับ เวคเตอร์โคออดิเนตของโลก [-1,-1] และ [-1,1] ตามลำดับ

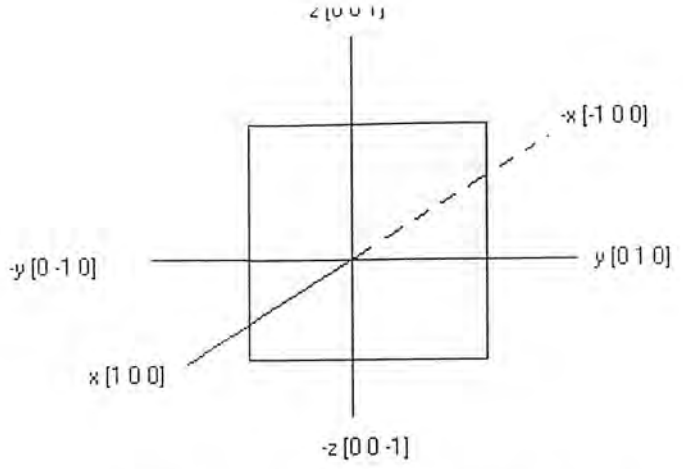
ในเอนจิ้นมอร์ฟิทนั้นจะสามารถเปลี่ยนเวคเตอร์ระหว่าง 2 ระบบนี้ได้ทำให้ผู้โปรแกรมสามารถเลือกใช้ระบบโคออดิเนตที่เหมาะสมต่อสถานะการณ์ได้

2.2 ระบบโคออดิเนต 3 มิติ

ในระบบ โคออดิเนต 3 มิติ จะต้องตั้งอยู่บนกฎมือซ้าย หรือ มือขวาอย่างใดอย่างหนึ่ง ใน DirectX จะใช้ระบบโคออดิเนตตามกฎมือซ้าย แต่ใน มอร์ฟิท และ ในโปรแกรมสร้างภาพ 3 มิติ โดยทั่วไปจะยึดระบบโคออดิเนต 3 มิติ ตามกฎมือขวา

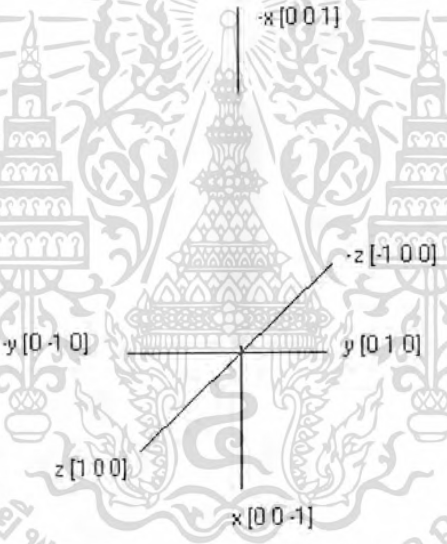
ตามกฎมือขวาถ้าสมมุติให้จุดกลางจอมอนิเตอร์เป็นจุดออริจิน จุดที่อยู่บนแกน x ด้านบวก คือ จุดที่อยู่หลังหน้าจอมอนิเตอร์ จุดที่อยู่บนแกน x ด้านลบ คือ จุดที่ชี้ออกมาด้านหน้า แกน y ด้านบวกคือจุดที่ชี้ไปทางขวาของหน้าจอ ในแกนลบจะชี้ไปทางซ้าย ส่วนแกน z ด้านบวกคือจุดที่อยู่ด้านบนของจอ ในด้านลบก็จะอยู่ด้านใต้ของจอ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 2-4 ระบบโคออดิเนต 3 มิติตามกฎมือขวา

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



รูปที่ 2-5 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

<u>2.2.1 การบวกเวกเตอร์</u> 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] เป็น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ทิศทางของวัตถุ 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

2.2.2 การลบเวคเตอร์ 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

2.2.3 ระนาบ 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 ยกตัวอย่างเชนในระนาบ xy ที่แกน z มีค่าเป็น 0 ก็จะมีแกน z

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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

2.3 แฮนเดิล (HANDLES)

ในการแยกแยะวัตถุทุกอย่างในมอร์ฟิทเอพีไอจำเป็นจะต้องรู้แฮนเดิลของวัตถุนั้นๆ กล้องทุกๆ ตัว และ วัตถุทั้งหมดจะต้องมีแฮนเดิลเป็นของตัวเอง ไม่ว่าจะเป็นบิตแมบ (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

2.4 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 Morfit ก็มีคำอธิบายหน้าที่ของฟังก์ชันเช่นกัน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ในส่วนต่อไปจะอธิบายถึงหน้าที่ของเอพีไอสำคัญๆ ของ มอร์ฟิท 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 ยังมีฟังก์ชันที่ทำหน้าที่เปลี่ยนรูปร่างวัตถุเพื่อนำมาทำแอนิเมชันด้วย

Camera API

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

Group API

เป็นเอพีไอที่อนุญาติให้รวมโพลีกอนต่างๆ เข้าไว้เป็นกลุ่มเดียวกัน ทำให้สามารถหมุน, เลื่อน, ย้าย หรือ ลดขนาดของกลุ่มของวัตถุนั้นพร้อมๆ กันได้ เราสามารถอ่านขนาดของกลุ่มของวัตถุได้ โดยในมอร์ฟิท 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 เปลี่ยนบิตแมบที่ในเกม บิตแมบที่ว่าเป็นบิตแมบที่ทำหน้าที่ให้สี และ เท็กซ์เจอร์กับวัตถุ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Animation API

เป็นเอพีไอที่ทำหน้าที่แสดงแอนิเมชันของชุดของโพลีกอน ที่เปลี่ยนไปตามเวลาต่างๆ แอนิเมชันสามารถใช้สร้างเอฟเฟคต่างๆ เช่น หมอก, แสดงวัตถุคนวิ่งในโลก หรือ วัตถุที่ได้รับความเสียหาย

Polygon API

วัตถุทุกๆ วัตถุประกอบขึ้นมาจากโพลีกอน สำหรับเอพีไอจะทำหน้าที่ควบคุมรูปร่าง, สี และบิตแมบของโพลีกอน 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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 3
การจัดการวัตถุบนโลก 3 มิติ โดยใช้เอพีไอวัตถุของมอร์ฟิท

3.1 โหมดแสดงผล และ วัตถุที่มีการเคลื่อนที่

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_DEFINED_BEHAVIOR
- Editor Mode เมื่อเรากำหนดค่านี้เป็น EDITOR_MODE

โดยข้อแตกต่างของทั้ง 2 รูปแบบเป็นดังตารางที่ 3.1.1

ตารางที่ 3.1.1 เปรียบเทียบ Viewer Mode กับ Editor Mode

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 ได้โดยการคลิ้กไปที่วัตถุ แล้วตั้งค่า "Properties of Object" ให้เป็น "dynamic"**)	เราสามารถควบคุม object ได้ทั้งหมดไม่ว่าจะเป็น static หรือ dynamic เพราะเราไม่ได้สร้าง world ที่ตายตัวไว้ในตอนแรก โดยในการควบคุมนั้นเราจะใช้ Group API

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.2 การเข้าถึง และ กำหนดชื่อให้วัตถุ

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);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.3 การเคลื่อนวัตถุ

เราสามารถทำการเคลื่อนย้ายวัตถุได้ 4 รูปแบบได้แก่

- การควบคุมโดยตรง โดยการกำหนดตำแหน่งและทิศทางของวัตถุ
- การกำหนดให้วัตถุเคลื่อนที่ไปตามเส้นทาง (Track)
- การกำหนดให้วัตถุเคลื่อนที่ตามวัตถุอื่นไป
- การกำหนดให้วัตถุเคลื่อนที่ตามกฎฟิสิกส์

3.3.1 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);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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] จะทำการเลื่อนวัตถุตามทิศทางด้านหน้าของวัตถุไม่ว่าวัตถุจะหันหน้าไปทางใดอยู่ก็ตาม

- 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

3.3.2 การกำหนดให้วัตถุเคลื่อนที่ไปตามเส้นทาง (Track)

ในบางครั้งถ้าเราไม่ต้องการกำหนดการเคลื่อนไหวของวัตถุด้วยตนเอง เราอาจกำหนดให้วัตถุเคลื่อนไปตามสิ่งอื่น ๆ ได้ โดยการกำหนดให้เคลื่อนที่ไปตามเส้นทางก็เป็นวิธีหนึ่ง จะมีรูปแบบและขั้นตอนการทำงานที่สำคัญ ๆ ได้แก่

1. เมื่อเรากำหนดให้วัตถุเคลื่อนที่ไปตามเส้นทางแล้ว Morfit จะทำการเลื่อนวัตถุไปตามเส้นทางทุกครั้งที่เราเรียกใช้ 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 3.2 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)
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ซึ่ง chase_type จะเป็นตัวระบุรูปแบบการเคลื่อนที่ตาม ที่เราสนใจในที่นี้ได้แก่

- CHASE_TRACK จะทำการเคลื่อนที่ตามแบบที่เน้นความแม่นยำ และเลี้ยวแบบหักตรง
- CHASE_TRACK_AIRPLANE จะทำการเลี้ยวแบบค่อย ๆ เลี้ยวหักมุม คล้ายกับเครื่องบิน
- แบบอื่น ๆ ดูรายละเอียดใน 3.3.3

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 3.3.2.1 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();
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

สำนักวิทยบริการ พระจอมเกล้าลาดกระบัง

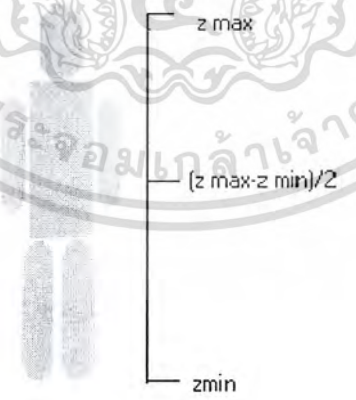

```
        DWORD man=Morfit_object_get_object_using_name("man");
        DWORD track=Morfit_track_get_using_name("mytrack");
        Morfit_object_set_chase_type(man, CHASE_TRACK);
        Morfit_object_set_track(man,track);
        Morfit_object_set_absolute_speed(man,20);

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

}
```

โปรแกรมที่ 3.3.2.1 แสดงตัวอย่างการกำหนดให้วัตถุเคลื่อนที่ไปตาม track

5. ทำการตรวจสอบค่า offset ของวัตถุ จากตัวอย่างโปรแกรมที่ 3.3.2.1 จะเห็นว่าเมื่อประมวลผลโปรแกรมแล้ว บางครั้งวัตถุ (ในที่นี้เราเลือกรูปคนมา) จะเดินแบบเท้าทะลุลงไปใต้ดิน ในการแก้ปัญหานี้เราต้องใช้การตรวจสอบค่า offset ของวัตถุนั่นคือเราสามารถสั่งให้ Morfit ทำการเลื่อนวัตถุไปตาม track โดยคำนึงถึงค่า offset ซึ่งเป็นค่าความสูงที่วัดจากแนวระดับขึ้นไปยังจุดศูนย์กลางของวัตถุ เช่นถ้าเรากำหนด offset vector เป็น [0 0 150] เราจะได้วัตถุที่เคลื่อนที่ไปตาม track โดยที่จุดศูนย์กลางของวัตถุจะอยู่สูงจากแนวระดับ 150 หน่วยเสมอ ตัวอย่าง เพื่อความเข้าใจเรื่อง offset เราจะใช้ตัวอย่างที่วัตถุเป็นรูปคน ดังรูปที่ 3-1



รูปที่ 3-1 แสดง offset ของวัตถุรูปคน

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

42758

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 เพราะ Morfit จะจัดการให้โดยอัตโนมัติ

6. เมื่อเราต้องการหยุดวัตถุไม่ให้เคลื่อนที่ไปตาม track ทำได้โดย

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

ทั้งหมดที่กล่าวมาแสดงให้เห็นว่าการใช้ track เป็นเรื่องง่าย เพียงแค่เรากำหนดให้วัตถุเคลื่อนไปตาม track, กำหนดความเร็ว และ offset นอกนั้น Morfit จะจัดการให้เราเอง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

3.3.3 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)

- วัตถุที่กำหนดให้ถูกเคลื่อนที่ตาม (Chased Object)

  วัตถุที่กำหนดให้ถูกเคลื่อนที่ตามจะเป็น dynamic object, camera, หรือ group ก็ได้ 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 ถ้าวัตถุเคลื่อนที่ตาม group หนึ่งอยู่ ถ้าเราเรียกใช้ Morfit_object_set_camera_to_chase() วัตถุนั้นจะหยุดเคลื่อนที่ตาม group และหันไปเคลื่อนที่ตาม 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

- ระยะการเคลื่อนที่ตาม (Chase Offset)

  เป็นตัวกำหนดว่าวัตถุที่กำลังเคลื่อนที่ตามอยู่ห่างจากวัตถุที่ถูกเคลื่อนตามเท่าไร เช่นถ้า chase offset เป็น [0, 0, 20] แล้ววัตถุที่เคลื่อนที่ตามจะเคลื่อนที่ไปพร้อมกับวัตถุที่ถูกตาม แต่

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จะอยู่เหนือวัตถุที่ถูกตามอยู่ 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** Morfit จะทำให้เอง

- การกำหนด object's chase distance จะเป็นตัวกำหนดระยะห่างว่าห่างจากวัตถุที่ถูกตามเท่าไร มีประโยชน์มากกว่า chase offset โดยจะสังเกตได้ในขณะที่วัตถุที่ถูกตามกำลังเลี้ยว ถ้าเราใช้ chase distance Morfit จะคำนวณตำแหน่งของวัตถุที่กำลังตามให้ใหม่ 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);
```

- **รูปแบบการเคลื่อนที่ตาม (Chase Type)**
  มีทั้งหมด 3 รูปแบบได้แก่

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

- Chase Location จะทำงานเหมือน precise แต่เฮลิคอปเตอร์จะไม่หมุนตามรถ เพียงแค่รักษาระยะ offset ให้เท่าเดิมเท่านั้น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- 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);
```

- **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);
```

3.3.4 การกำหนดให้วัตถุเคลื่อนที่ตามกฎฟิสิกส์

เราสามารถกำหนดให้วัตถุเคลื่อนที่ไปตามกฎฟิสิกส์ได้ โดยเราเพียงแค่กำหนดความเร็วต้นกับทิศทาง, กำหนดคุณสมบัติต่าง ๆ ทางฟิสิกส์ จากนั้น Morfit จะจัดการส่วนที่เหลือให้เอง 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};
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
Morfit_object_set_speed(object_handle,speed);
```

- แบบความเร็วและทิศทางแยกกัน โดยเราสามารถกำหนด magnitude ได้หลายค่า ทำให้เราสามารถกำหนดให้วัตถุหลาย ๆ วัตถุเคลื่อนที่ไปในทิศทางเดียวกัน ที่ความเร็วแตกต่างกันได้ดังนี้

```
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 sqrt($x^2 + y^2 + z^2$) ซึ่งในที่นี้เราได้จากการคำนวณ 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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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);
```

3.4 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 ตัดกับเส้นตรงที่เรากำหนดจุดต้นและจุดสิ้นสุดเอาไว้ ซึ่งโดยปกติจะกำหนดให้ลากผ่านวัตถุที่เราต้องการตรวจสอบการชน และอาจกำหนดไว้หลายเส้นเพื่อครอบคลุมจุดหลัก ๆ ของวัตถุ เช่นกำหนดไว้ที่รถ 2-3 เส้น ถ้าต้องการตรวจสอบว่ารถชนกับอะไรบ้าง), ค่า handle ของโพลีกอน(polygon) ที่มาตัดกับเส้น และค่า handle ของวัตถุที่เป็นเจ้าของโพลีกอนนั้น (blocking_object)

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

การทำงานอื่น ๆ ที่สำคัญได้แก่

- การกำหนดให้วัตถุสามารถถูกทะลุผ่านได้เช่น เมฆ, ควัน ทำได้โดย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
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);
```

3.5 ฟังก์ชันที่มีประโยชน์อื่นๆ

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 เมื่อวัตถุต้นแบบเปลี่ยน bitmap ไป วัตถุที่ copy ขึ้นมาจะถูกเปลี่ยน bitmap ด้วย เพราะใช้โพลีกอนเดียวกัน

- 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 บางคร้างวัตถุอาจจะทะลุผ่านเนินเขา หรือลอยอยู่บนอากาศ ในการหลีกเลี่ยง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ปัญหานี้ เราอาจกำหนดให้ track อยู่เหนือพื้น (หรือกำหนด track บนพื้นแล้วใช้ค่า offset ยกขึ้นมา) แล้วเรียกใช้ฟังก์ชันการตกจาก track (falling from track) เพื่อทำให้ตกลงมาบนพื้นด้านล่าง track โดยตรง การทำแบบนี้จะทำให้วัตถุสามารถเคลื่อนที่ผ่านเนินเขาและยอดเขาได้ โดย Morfit จะปรับค่า 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();
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่4

การควบคุมกล้องโดยใช้ คาเมราเอพีไอของมอร์ฟิท

การแสดงผลโลก 3 มิติ นั้นจะต้องแสดงผ่านกล้องที่มีอยู่ในโลก 3 มิติ ซึ่งในบทนี้เราจะกล่าวถึงวิธีสร้างและควบคุมกล้อง

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

4.1 แฮนเดิล และ ชื่อของกล้อง

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);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

แต่กล้องจะถูกสร้างขึ้นตามค่าดีฟอล์ท (Default) เราต้องมากำหนดตำแหน่งและทิศทางก่อนจะนำไปใช้งาน โดยฟังก์ชันนี้จะคืนค่าแฮนเดิลของกล้องที่สร้างขึ้นมาใหม่ให้ ส่วนการทำลายกล้องที่สร้างขึ้นมาจะใช้คำสั่ง

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

หลังจากคุณทำลายกล้องไปแล้ว จะไม่สามารถเรียกใช้กล้องนั้นๆ ได้อีก

นอกจากนี้ยังมีแฮนเดิลของกล้องที่สำคัญๆ ที่เราควรรู้ คือ

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

```
Morfit_camera_get_default_camera(void);
```

เคอร์เรนท์คาเมรา (Current Camera) หมายถึง กล้องที่ใช้แสดงผลตัวสุดท้าย (ที่ใช้อยู่ขณะนั้นๆ) ฟังก์ชันบางตัวของ Morfit นั้นจะไม่สามารถใช้กับกล้องที่เรากำหนด แต่จะใช้ได้กับเคอร์เรนท์คาเมราเท่านั้น เช่น Morfit_engine_3D_point_to_2D การใช้คำสั่งพวกนี้เราต้องแน่ใจว่าเรียกใช้กับกล้องที่เราต้องการ ซึ่งสามารถตรวจสอบ หรือ เปลี่ยนเคอร์เรนท์คาเมราได้โดยใช้คำสั่ง

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

4.2 ตำแหน่งกล้อง

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 จะใช้ฟังก์ชัน

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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

4.3 ทิศทางกล้อง

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);
```

แต่ใน Morfit มีการเตรียมคำสั่งที่จำเป็นให้เกือบทั้งหมดแล้ว 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)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ในเรื่องการหมุนกล้องใน Morfit ได้จัดเตรียม 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 Moffit

```
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);
```

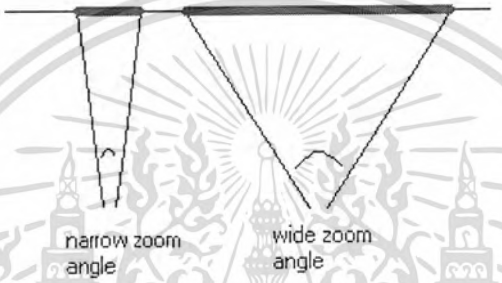
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
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);
```

4.4 ซูม (Zoom)

การซูมหมายถึงพื้นที่ที่เรามองเห็นนั้นเอง ยิ่งพื้นที่ที่เรามองเห็นน้อย ภาพที่เห็นก็จะใหญ่ หรืออีกนัยหนึ่งก็คือความกว้างของมุมที่มองเห็นนั้นเอง



รูปที่4-1

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);
```

ใน Morfit สามารถตั้งค่ามุมของพื้นที่ที่มองเห็นได้ตั้งแต่ 1 องศา – 179 องศา

4.5 ตัวอย่างโปรแกรม Camera Control

โปรแกรมนี้จะแสดงถึงวิธีการหมุน และ เลื่อนกล้อง โดยจะให้ผู้ใช้เป็นผู้ควบคุมกล้องโดยใช้ปุ่มลูกศรขึ้น และ ลง ในการควบคุมกล้อง โดยมีโหมดการควบคุมดังนี้

Key Mode

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

M	Move 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 bank angle
H	Increase \ decrease head angle
T	Increase \ decrease tilt angle
F	Increase \ decrease zoom (field of view)
O	Point camera at next object
C	Switch to next camera

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);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
        }
}

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;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
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)
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
                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;
}
```

4.6 คุณภาพของภาพ

มอร์ฟิทอนุญาติให้เราสามารถตั้งค่าคุณภาพของภาพที่จะแสดงออกมาทางหน้าจอได้ ถ้าเราตั้งค่านี้ไว้สูงก็จะทำให้ความเร็วในการแสดงผลตกลง แต่ถ้าตั้งค่านี้ไว้ต่ำก็จะทำให้แสดงผลได้เร็วขึ้น

ขนาดของภาพที่มาใช้เป็นเท็กซ์เจอร์ก็มีผลต่อคุณภาพของภาพเช่นกัน 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);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
Morfit_engine_increase_picture_quality();
Morfit_engine_decrease_picture_quality();
```


เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 5
การใช้ MFC

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 ในที่นี้จะเป็นเพียงการสอนความรู้ที่จำเป็นในการสร้าง MFC projects ที่ใช้ Morfit 3D graphics tools

5.1 การสร้างโลก 3 มิติ

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

5.2 โปรแกรมเบื้องต้น 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 ซึ่งเราจำเป็นต้องลบบางส่วนของมันทิ้งเพื่อสร้างโปรแกรมของเราให้สมบูรณ์

เรามาทบทวนความจำสักเล็กน้อย ด้านล่างนี้เป็นรูปแบบของ console applications:

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

// 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

.

.

.

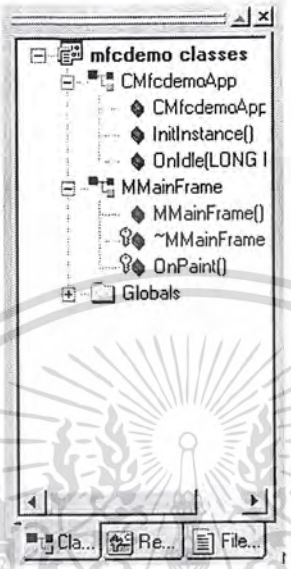
ให้พยายามทำความเข้าใจความแตกต่างของ program model ทั้ง 2 แบบ จะเห็นได้ว่าไม่มี "rendering loop" ใน MFC เราเพียงแค่จัดเตรียม world และตอบสนองต่อ events ต่าง ๆ เมื่อ Windows บอกเราว่ามันเกิดขึ้น

5.2.1 การกำหนดค่าเริ่มต้น

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กำหนดฟังก์ชันให้มันเรียกใช้ เราไม่จำเป็นต้องกำหนดฟังก์ชัน main (หรือ winmain) MFC จะเป็นตัวจัดการให้ อาจจะมีคำถามขึ้นมาว่า "แล้วเราจะกำหนดค่าเริ่มต้นไว้ที่ไหนล่ะ ?" คำตอบก็คือในฟังก์ชัน InitInstance() ซึ่ง MFC รับประกันว่าจะถูกเรียกเป็นตัวแรกเมื่อเราเริ่มรันโปรแกรมในหน้าต่าง Workspace เลือกแท็บ ClassView ขยายสาขาของ tree ลงมาจะเป็นดังรูป 5-1



รูปที่ 5-1 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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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

5.2.2 Rendering the World

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

```
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

5.3 การพัฒนาโปรแกรมที่ได้จาก Morfit Application Wizard

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

5.3.1 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);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
}
```

code นี้จะหมุนแต่ละวัตถุใน world ไป .5 องศาไปตามแต่ละแกน 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

5.3.2 การตั้งเวลา (Timer)

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

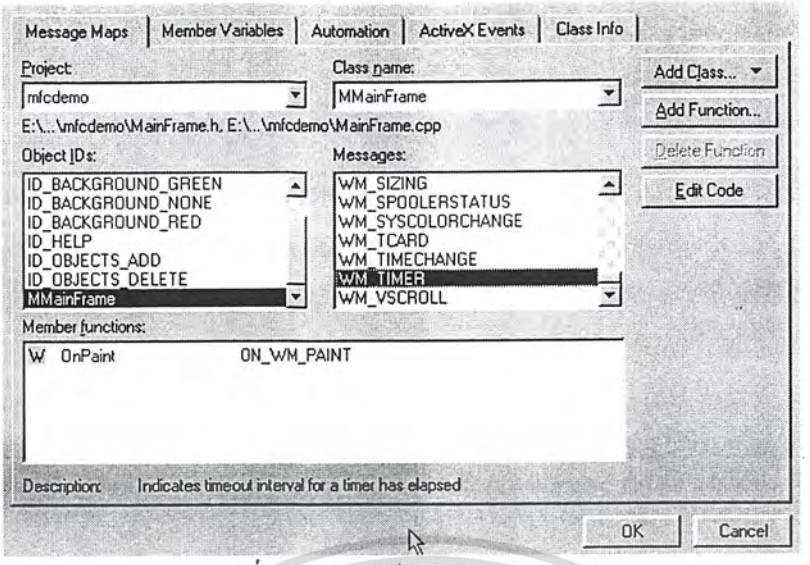
หลักการของ Windows Timer นั้นง่ายมาก เราตั้ง timer ให้สร้าง event ทุก ๆ n milliseconds และกำหนดฟังก์ชันที่จะให้ Windows เรียกใช้เพื่อตอบสนองต่อ event นั้น ให้เพิ่มบรรทัดด้านล่างเข้าไปในฟังก์ชัน InitInstance():

```
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 messege ซึ่งกรอบโต้ตอบควรจะเป็นดังรูป 5-2

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 5-2 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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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

5.3.3 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 โดยการคลิ้กขวาบนคลาส 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

ต่อไปนี้เป็นอัลกอริทึมที่เราจะใช้:

```
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
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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 ของวัตถุและ polygon ที่ถูกคลิ้กกลับมาด้วย 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_lo :ation(selected,result[0],result[1],result[2]);
}
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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 5.2 ประกอบ ถ้าเลื่อนตำแหน่งวัตถุไปยังตำแหน่งพื้นหลังของหน้าต่างแล้ววัตถุหายไปแสดงว่า camera อยู่บนระนาบ x ที่ห่างจาก x=1000 มากเช่นอาจอยู่ใน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ช่วง 600 เป็นต้นจึงทำให้มองไม่เห็นวัตถุที่ย้ายไปบนระนาบ x=1000 ให้เราใช้ World Builder แก้ค่าตำแหน่งของ camera ให้ x=2000 เพื่อให้เรามองเห็นระนาบ x=1000 แน่ ๆ ก็จะแก้ปัญหานี้ได้ (ถ้ากำหนดตำแหน่ง camera ให้ x=1000จะไปทับกับระนาบที่เราจะย้ายวัตถุไปพอดีทำให้เราอาจมองไม่เห็นวัตถุเหมือนเดิม)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

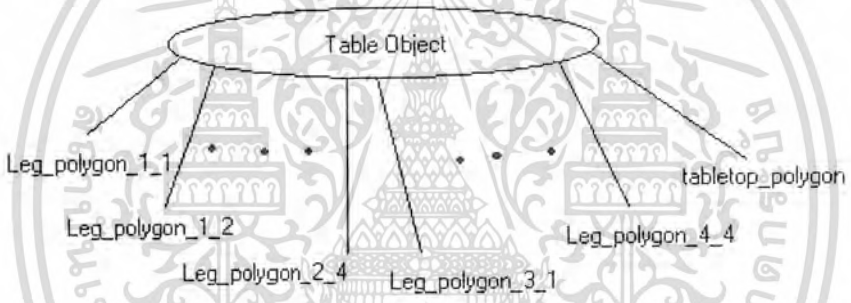
บทที่ 6
กรุ๊ปเอพีไอ

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

6.1 กรุ๊ป คือ อะไร

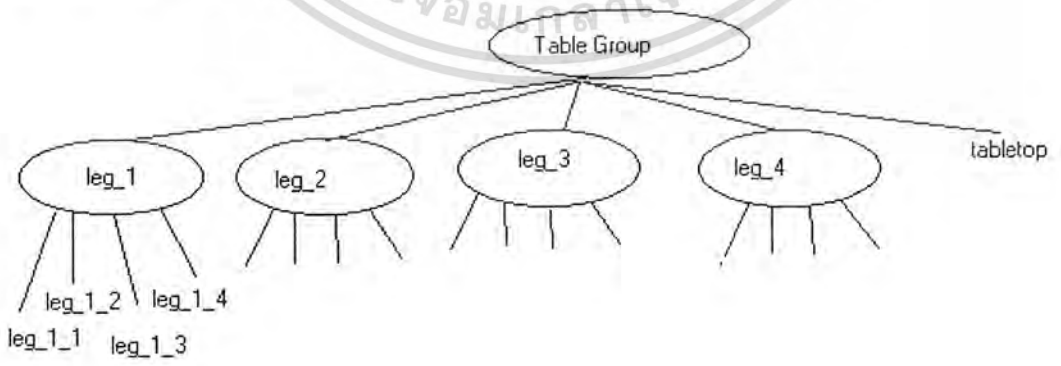
Group คือ โพลีกอนที่สะสมรวมกันขึ้นมาเป็นวัตถุ หรือ ส่วนย่อยของวัตถุ Group สามารถเป็นส่วนประกอบของ Group อื่น ทำให้เราสามารถสร้างความสัมพันธ์อย่างเป็นลำดับระหว่าง Group ได้

ยกตัวอย่างเช่น โต๊ะตัวหนึ่งประกอบด้วยขา 4 ขา และ พื้นโต๊ะด้านบน แต่ละขาประกอบด้วย 4 โพลีกอน เท่ากับว่าโต๊ะตัวนี้มี 17 โพลีกอน หากไม่มีการใช้ Group โต๊ะก็จะมีรูปแบบดังรูปที่ 6-1



รูปที่ 6-1

จะเห็นว่าไม่มีการแสดงลำดับชั้นความสัมพันธ์ระหว่างโพลีกอน โพลีกอนทั้งหมดเป็นของวัตถุโต๊ะ แต่ถ้าหากมีการจัด Group ก็จะเป็นดังรูป 6-2



รูปที่ 6-2

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

กรุ๊ปมีประโยชน์อะไร? เราลองนึกว่ามีโต๊ะกับเก้าอี้อยู่ แล้วเราเอาเก้าอี้ไปวางไว้บนโต๊ะ เมื่อคุณเลื่อนโต๊ะไปก็หมายความว่าเก้าอี้ที่อยู่บนโต๊ะก็ต้องเลื่อนตามไปด้วยเหมือนเป็นของชิ้นเดียวกัน (ถ้าคุณไม่ได้กรุ๊ป โต๊ะกับเก้าอี้ ตอนที่คุณเลื่อนโต๊ะ เก้าอี้จะไม่เลื่อนตามไปด้วย)

กรุ๊ปแต่ละกรุ๊ปจะมีตัวชี้ไปยัง Father Group จากรูป 6.2 leg_1_1 จะมี leg_1 เป็น Father Group และ leg_1 ก็จะมีวัตถุโต๊ะเป็น Father Group ส่วนตัวชี้ของวัตถุโต๊ะจะเป็น NULL หมายความว่าโต๊ะไม่ได้เป็นส่วนหนึ่งของวัตถุใดที่ใหญ่กว่า

6.2 การสร้างกรุ๊ป

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

การสร้าง Group จะใช้คำสั่ง

```
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);
```

ในโค้ดนี้จะแสดงถึงวิธีสร้าง Group ของวัตถุโต๊ะที่ประกอบไปด้วย 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);
```

ลบกรุ๊ป

ในการลบกรุ๊ป หรือโพลีกอน 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
```

แต่ถ้าต้องการลบสมาชิกทั้งหมดออกจาก Group และ โลก 3 มิติ ให้ใช้คำสั่ง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Morfit_group_delete_members(DWORD group_handle)

การก็อบปี้กรุ๊ป

DWORD Morfit_group_duplicate_tree(DWORD group_handle)

คำสั่งจะส่งค่าแฮนเดิลของกรุ๊ปใหม่คืนมา

การตรวจสอบกรุ๊ป

ในการตรวจสอบว่าโพลีกอนเป็นสมาชิกของกรุ๊ปใดทำได้โดยใช้คำสั่ง

DWORD Morfit_polygon_get_group(DWORD polygon_handle);

ฟังก์ชันนี้จะส่งค่าแฮนเดิลของกรุ๊ปที่โพลีกอนนั้นๆ เป็นสมาชิกอยู่กลับมา ถ้าค่าที่ส่งกลับเป็น NULL แสดงว่าโพลีกอนนั้นไม่ได้เป็นของ Group ใดเลย ตามตัวอย่างที่ได้ที่ผ่านมาถ้าเรียกใช้ฟังก์ชัน

Morfit_polygon_get_group(tabletop) จะส่งค่าแฮนเดิลของ *table* กลับมา

Morfit_polygon_get_group(leg_polygon_1_1) จะส่งค่าแฮนเดิลของ *leg_1* กลับมา

การตรวจสอบว่าโพลีกอนเป็นของกรุ๊ป หรือไม่ 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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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

6.3 การใช้กรุ๊ป

เราสามารถควบคุม Group ได้เหมือนกับวัตถุ 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 ภาพบิตแมบที่เป็นพื้นผิว

แฮนเดิลและชื่อของกรุ๊ป

การอ่านค่าแฮนเดิลของกรุ๊ปใช้วิธีเดียวกับวัตถุ และ กล้อง แต่ฟังก์ชันที่ใช้จะต่างกันเล็กน้อยตามตัวอย่าง

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);

การอ่านค่าแฮนเดิลของโพลีกอนที่รวมกันเป็นกรุ๊ป

DWORD Morfit_group_get_using_name(char *name);

DWORD Morfit_group_get_first_po ygon(DWORD group_handle);

การเคลื่อนย้ายกรุ๊ป

การกำหนดตำแหน่งกรุ๊ป

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จะตั้งศูนย์กลางของกรุ๊ปไว้ที่ จุดที่กำหนด ในการตรวจสอบตำแหน่งจะต้องใช้

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

แต่จริงแล้วคำสั่งนี้จะส่งกลับค่า Center of Volume โดย Morfit จะตรวจสอบจาก 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 หมายถึงจุดที่อยู่ใน Group และ point_to_match หมายถึงจุดที่อยู่นอก Group คำสั่งนี้จะเลื่อน Group โดยอ้างอิงจากจุด point_belongs_to_group ที่กำหนดไปยังจุด point_to_match

การหมุนกรุ๊ป

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) หมุน Group 45° รอบแกน Y ของโลก 3 มิติ

Morfit_group_rotate(group, 30, OBJECT_SPACE, 0) หมุน Group 30° รอบแกน X ของวัตถุ

การหมุนกรุ๊ปนั้น จะหมุนรอบจุดศูนย์กลาง โดยเราสามารถกำหนดจุดศูนย์กลางใหม่ได้ดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
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() ค่าที่ได้จะเป็นคนละค่ากัน โดยค่า Center of volume จะเป็นค่าเดิม

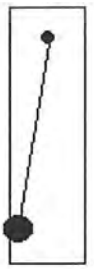
ทำไมถึงต้องตั้งค่าจุดหมุนใหม่ ลองนึกถึงการแกว่งของลูกตุ้มนาฬิกาดู โดยลูกตุ้มมี center of volume อยู่ที่จดกึ่งกลางดังรูป

รูปที่ 6-3

ถ้าหมุนลูกตุ้มรอบแกน X ก็จะได้ผลดังรูป 6-4

รูปที่ 6-4

จะเห็นว่าผลที่ได้ไม่เป็นอย่างที่ต้องการ เราจึงต้องเลื่อนจุดหมุนไปที่ปลายด้านบนของลูกตุ้ม จึงจะได้ผลตามที่ต้องการดังรูป 6-5



รูปที่ 6-5

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ในการหมุนวัตถุรอบเส้นตรงใดๆ ก็ทำได้โดยใช้ฟังก์ชัน

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

โดย p1 และ p2 จะเป็นจุดบนเส้นตรงที่ต้องการ

กรุ๊ปโอเรียนติง

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);

โดยฟังก์ชันดังกล่าวจะส่งกลับค่าแฮนเดิลของโพลีกอนแรกใน Group ที่ถูกโอเรียนเทชัน

หลังจาก Group ถูกกำหนดด้านหน้า, ด้านหลัง, ด้านบน และ ด้านล่างแล้ว เราก็จะสามารถใช้ระบบแกนของ Group ได้

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 จะมีทิศทางจากด้านหน้าไปยังด้านหลังของกรุ๊ป, แกน Y จะมีทิศทางจากด้านซ้ายไปยังด้านขวาของ Group และ แกน Z จะมีทิศทางจากด้านล่างไปยังด้านบนของ Group หลังจากนี้คุณก็จะสามารถหมุน Group ให้ตรงตามระบบแกนที่ใช้

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
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);
```

การเปลี่ยนขนาดของกรุ๊ป

```
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 แสงที่มีทิศทางซึ่งความสว่างของแสงแบบนี้จะขึ้นอยู่กับมุมที่แสงตกกระทบกับผิวโพลีกอน 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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บิตแมบ

เราสามารถกำหนดบิตแมบที่จะคลุมกรุ๊ปได้ ส่วนค่าแฮนเดิลของบิตแมบสามารถอ่านได้โดย 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 หมายถึงจำนวนบิตแมบที่มีซ้ำได้ตามแกน x และ y

แต่ในความจริงกรุ๊ปหนึ่งๆ ส่วนมากจะไม่ใช้บิตแมบ หรือ สี เพียง 1 ภาพ ในบทที่ 9 เราจะกล่าวถึงการกำหนดสี หรือ บิตแมบให้แต่ละโพลีกอน

คุณสมบัติของกรุ๊ป

เราสามารถกำหนดคุณสมบัติ เช่น Group เป็นแบบสเตติก 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 และ กรุ๊ปจะถูก enable ตามค่าดีฟอล์ท ถ้าเราต้องการโหลดกรุ๊ป โดย 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);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 7

Engine API

Engine API จะเกี่ยวข้องกับการควบคุม world โดยรวมและแนวทางที่จะแสดงผล world ด้วย ในที่นี้จะกล่าวถึงแต่ฟังก์ชันที่ใช้งานบ่อย ๆ เท่านั้น ดูรายละเอียดส่วนที่เหลือได้จาก Morfit Extension Help หรือ Morfit Header File

7.1 การโหลดโลก 3 มิติ

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 ของ group ที่เกิดจาก world ใหม่ และส่วนใหญ่เรามักจะใช้ Morfit_group_set_location() ประกอบด้วยเพื่อให้ตำแหน่งของ group ที่เราต้องการเป็นค่าเดียวกันกับ position ที่เราจะวางลงไปแทนที่จะเป็นจุดกำเนิดของ world ใหม่ซึ่งไม่ใช่ตำแหน่งของ group ที่เราต้องการ

- การตรวจสอบว่ามี world ถูก load อยู่แล้วหรือไม่ ทำได้โดย:

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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 ของ group ที่เราต้องการบันทึกลง disk ใช้ค่า NULL ถ้าต้องการบันทึกทั้ง world

save_flag มีค่าเป็นได้ดังนี้ (ใช้ "|" เพื่อกำหนดหลายค่าร่วมกัน):

SAVE_DEFAULT

SAVE_ONLY_WHATS_NEED ไม่บันทึกส่วนที่ไม่จำเป็นต่อการบันทึก group เช่นจะบันทึกแต่ภาพเคลื่อนไหวที่จำเป็นสำหรับ polygon ใน group ที่เราต้องการ cameras, พื้นหลังและ tracks จะไม่ถูกบันทึก

SAVE_FLAT_BITMAP_PATH ไม่บันทึก path สำหรับแต่ละ polygon โดยอนุมาณว่าอยู่ path เดียวกัน

SAVE_BITMAP_AS_JPG บันทึก bitmaps ในรูปแบบ JPG

SAVE_BITMAP_AS_BMP บันทึก bitmaps ในรูปแบบ BMP

SAVE_BITMAP_AS_JBM บันทึก bitmaps ในรูปแบบ JBM

SAVE_IN_MULTIPLE_FILES บันทึกแต่ละวัตถุใน file ของตนเอง

SAVE_ONLY_MODELS

SAVE_RELEASE วัตถุทั้งหมดถูกรวมไว้ในโมดูล main

SAVE_BITMAP_OPTIMAZED บันทึก bitmaps ทั้งหมดในรูปแบบ JPG ยกเว้นที่เป็นแบบโปร่งใส จะบันทึกเป็น BMP

แต่ละรูปแบบของการบันทึก bitmaps เป็นดังนี้

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

7.2 การแสดงผลโลก 3 มิติ

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

- การ render world ทำได้โดย:

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
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 Morfit ทำงานได้เร็วที่สุดที่ 16 bpp ซึ่งเป็นค่าโดยปริยาย ส่วนค่าความละเอียดของการแสดงผลโดยปกติคือ 640 x 480, 800 x 600 และ 1024 x 768

- การ render ไปยัง memory ทำได้โดย:

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
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();
```

7.3 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) กำหนดให้ bitmap ของวัตถุที่อยู่ไกลถูกแทนที่ด้วยสีเต็ม ๆ (ไม่มีลายของ bitmap เลย) 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) เลือกแสดงวัตถุที่อยู่ในระยะที่กำหนดเท่านั้นและจะไม่แสดงวัตถุที่อยู่ไกลกว่าระยะนั้น ทำได้โดย:

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
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 ในการบัง bitmap ของวัตถุที่อยู่ไกล 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" ส่วนการกำหนดสีของ 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
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- Rendering Mode มีอยู่ 3 modes ได้แก่
  1) "normal" เป็นค่าโดยปริยาย จะแสดงวัตถุและ bitmaps ทั้งหมด
  2) "color fill" จะแสดงสีเต็ม ๆ แทนที่ bitmaps ทั้งหมด
  3) "wire frame" จะแสดงแต่เส้นรอบนอกของวัตถุเท่านั้น

ส่วนใหญ่เรามักจะไม่ค่อยเปลี่ยนเป็น mode อื่นนอกจาก normal มากนัก เราสามารถดู effects เหล่านี้ได้ใน World Builder โดยการเลือก camera mode แล้วคลิ๊กขวาบนภาพเลือก "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();
```

7.4 โลก 2 มิติ และ โลก 3 มิติ

มีการใช้งานที่สำคัญ ๆ ดังนี้

7.4.1 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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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 และจะตัดให้สเนอยู่ในหน้าต่างพอดี 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

7.4.2 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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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]);
```

7.5 การเคลื่อนที่

7.5.1 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:

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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

ฟังก์ชันนี้ยังคำนวณค่าเส้นตั้งฉากเฉลี่ยของโพลีกอนเหล่านั้นด้วย

7.5.2 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

7.6 ล็อกวินโดว์ (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:

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
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:

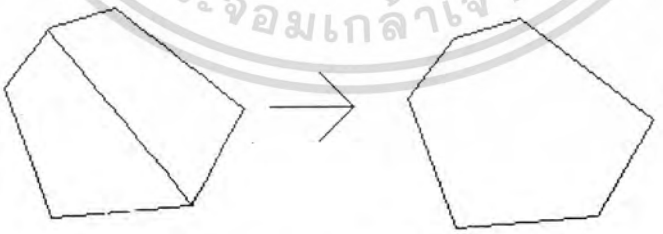
```
HWND Morfit_engine_log_window_get_hwnd();
```

7.7 ฟังก์ชันที่มีประโยชน์อื่นๆ

- 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 9.8.1



รูปที่ 7-1 การรวมโพลีกอน

การทำแบบนี้จะประหยัดหน่วยความจำและเวลาที่ใช้ประมวลผล ซึ่งทำได้โดย:

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

ฟังก์ชันนี้ทำงานใน editor mode เท่านั้น และจะคืนจำนวนของโพลีกอนที่กำจัดไป ค่า accuracy คือค่าความใกล้เคียงที่จะรวมโพลีกอนเข้าด้วยกัน ถ้าเป็น 0 จะทำให้โพลีกอนที่อยู่บนระนาบเดียวกันเท่า

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

นั้นที่จะรวมกัน ถ้าเส้นตั้งฉากของทั้งสองระนาบเป็น 1.0 และ .9 แล้วโพลีกอนจะรวมกันเมื่อ accuracy เป็น .1 เท่านั้น

นอกจากนี้เรายังสามารถประหยัดการคำนวณได้โดย tools อื่น ๆ อีก เช่น

- unload bitmap ที่ไม่ใช้แล้ว:

```
int Morfit_engine_unload_unused_bitmaps();
```

- คำนวณ bitmaps ที่ถูก 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);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 8

การจัดการบิตแมบ และ แอนิเมชันโดยใช้มอร์ฟท

อย่างที่ทราบกันดีว่าพื้นผิวของโมเดลสามารถถูกเติมได้ด้วยสี หรือ บิตแมบก็ได้ ในบทนี้เราจะกล่าวถึงการทำงานกับบิตแมบ ไม่ว่าจะเป็นการสร้าง, แปะมันลงบนวัตถุ หรือ ย้ายบิตแมบ รวมไปถึงการทำแอนิเมชัน, การตั้งค่าบิตแมบที่ทำให้วัตถุดูเหมือนมีการเคลื่อนไหว หรือ เปลี่ยนไป โดยจะใช้ World builder ในการทำความเข้าใจกับ บิตแมบ และ แอนิเมชัน รวมถึงศึกษา API ที่ใช้ในการควบคุมในโปรแกรมของเรา

8.1 การแปะบิตแมบลงบนวัตถุหรือโพลีกอน

หลังจากที่เรามีภาพบิตแมบที่ต้องการแล้วให้เปิด World Builder ขึ้นมา สร้างโลกขึ้นแบบ Blank Floor หลังจากนั้น ให้เราวาง Morfit cube ที่อยู่ในซับโฟลเดอร์ชื่อ 3Dforms ของโมเดลแกลเลอรี่ โมเดลที่ว่าจะผิวเป็นรูปโลโก้ของ Morfit

รูปที่8-1 รูปที่เลือกมาจากฮาร์ดดิสก์

ให้เราลองวางรูปที่สร้างขึ้นเองบนโมเดลดู โดยคลิกที่ปุ่ม Select Bitmap from Harddisk เลือกบิตแมบที่เราต้องการ หลังจากนั้นเคอร์เซอร์จะกลายเป็นรูปถังใส่สี ให้เราคลิกที่ด้านที่เราต้องการจะวางบิตแมบลงไปจะได้ผลดังรูป

รูปที่ 8-2

ถ้ารูปที่วางลงไปเอียงข้างดังรูปให้คุณหมุนบิตแมบ โดยเลือกที่โหมดบิตแมบแล้วเลือกที่บิตแมบที่ต้องการจะหมุน แล้วคลิกที่ปุ่มหมุน โดยให้กดปุ่ม Shift ค้างไว้ด้วย (จะหมุนทีละ 90 องศา)

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

8.2 การทิลลิ่งบิตแมบ (Tiling)

ใน World builder เมื่อเราวางรูปลงบนวัตถุ World Builder จะปรับขนาดภาพให้พอดีกับโพลีกอนที่วางลงไป แต่ถ้าเราไปแก้ไขบิตแมบให้มีขนาดเล็กลง เอนจิ้นก็จะต้องวางบิตแมบลงไปหลายๆ ภาพ เพื่อให้เต็มโพลีกอนนั้นๆ เราเรียกการกระทำนี้ว่าทิลลิงบิตแมบ (Tilling Bitmap) จริงๆ แล้วเอนจิ้นไม่ได้ลดขนาดบิตแมบ แต่เอนจิ้นไปแก้ไขค่าบิตแมบโคออดิเนตของโพลีกอนนั้น เราจะอธิบายเพิ่มเติมในภายหลัง การเปลี่ยนขนาดทำได้โดยเลือกบิตแมบที่ต้องการเปลี่ยนขนาดแล้วคลิกที่ปุ่มแว่นขยาย หรือ กดคลิกขวาที่บิตแมบเมื่ออยู่ในโหมดบิตแมบเพื่อเข้าไปแก้ไขค่าคุณสมบัติต่างๆ เช่น จำนวนบิตแมบตามแนวตั้ง และแนวนอน

8.3 บิตแมบที่มีบางส่วนโปร่งใส (Transparency)

การที่เราวางบิตแมบลงไปบนพื้นผิว มันก็จะบังสิ่งที่อยู่ด้านหลังทั้งหมด แต่บางครั้งเราก็ต้องการให้มีบางส่วนที่ใสสามารถมองทะลุได้ ถ้าเราต้องการให้บิตแมบที่เราวางลงไปมองทะลุได้ ก็เพียงแต่เลือก Most Frequent Color ที่มุมล่างขวาสุดของไดอะล็อกที่แสดงขึ้นมาหลังกดปุ่ม Load Bitmap from Hard Disk บริเวณที่มีสีเป็นสีใสของภาพที่โหลดขึ้นมาก็จะสามารถมองทะลุไปได้เหมือนไม่มีอะไรอยู่ ส่วนสีที่ส่วนมากนิยมใช้เป็นสีใส ก็คือสีที่มีค่า RBG อย่างใดอย่างหนึ่งหรือมากกว่าเป็น 255 แล้วที่เหลือเป็น 0 หมด เช่น แดง, น้ำเงิน, เขียว และ เหลือง เป็นต้น

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

8.4 ฟอร์แมทของบิตแมบ

เอนจิ้น Morfit สนับสนุนบิตแมบ 3 ฟอร์แมท คือ BMP, JPEG และ JBM โดยแต่ละฟอร์แมทก็มีข้อดีข้อเสียต่างกันไป

BMP มีคุณภาพสูง ง่ายต่อการแก้ไขเพราะสามารถใช้ Paint แก้ไขได้ แต่ก็กินเนื้อที่มาก

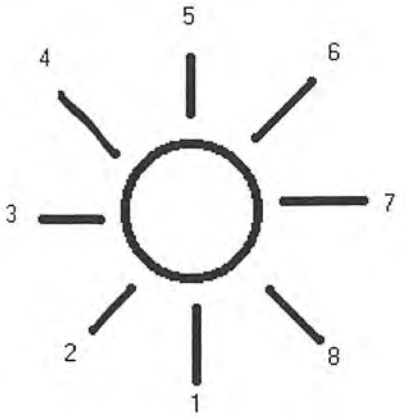
JPEG มีคุณภาพต่ำกว่า แต่ก็มีข้อดีคือมีขนาดเล็ก

JBM เป็นฟอร์แมทของ Morfit เอง ถ้าเซฟเป็นไฟล์ชนิดนี้จะโหลดได้เร็วกว่าฟอร์แมทอื่น

8.5 นิยามของแอนิเมชัน

แอนิเมชันก็คือเซตของบิตแมบที่เปลี่ยนตัวเองไปเรื่อยในช่วงเวลาหนึ่ง อย่างเช่นภาพที่แสดงออกมาบนจอทีวีเป็นต้น เนื่องจากบิตแมบแอนิเมชันเป็นภาพ 2 มิติ เราจึงทำแอนิเมชันไว้หลอกตา 8 ด้าน เพื่อที่จะมองแอนิเมชันเสมือนเป็นภาพ 3 มิติ

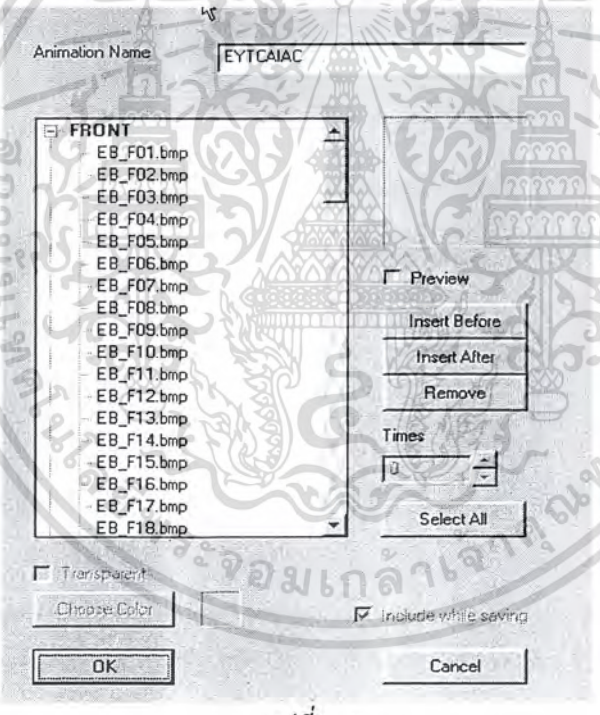
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 8-3

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

ในการแก้ไขแอนิเมชันใน World Builder ให้เราไปที่แทบที่อยู่ด้านซ้ายของจอ เลือกที่แทบบิตแมบแล้วคลิกขวาที่รูปที่เราต้องการ เลือก Edit Animation โปรแกรมจะแสดงหน้าจอดังรูป 8.4 ขึ้นมา



รูปที่ 8-4

เราสามารถใช้บิตแมบมาทำเป็นแอนิเมชันได้มากตามที่เราต้องการแต่มีข้อยกเว้น คือ เราไม่สามารถเซทจำนวนบิตแมบที่เป็นแอนิเมชันไม่เท่ากันในแต่กลุ่ม ยกเว้นแต่ว่ากลุ่มที่ไม่เท่ากับกลุ่มอื่นไม่มีแอนิเมชันเลยส่วนเวลาที่จะแสดงผลภาพนั้นนานเท่าไรสามารถตั้งค่าได้ที่ Times โดยค่า 100 เท่ากับ 1 วินาที หลังจากที่เราแก้ไขเสร็จแล้วให้กดที่ปุ่ม OK เพื่อบันทึกสิ่งที่เราได้แก้ไขไป

ในการสร้างแอนิเมชันขึ้นมาใหม่ให้เราคลิกขวาที่บริเวณใดก็ได้ในแทบบิตแมบแล้วเลือก Add Animation เลือกด้านที่ต้องการจากทั้ง 5 ด้าน แล้วคลิกที่ Insert After หลังจากที่เราได้ใส่แอนิเมชันในด้านที่ต้องการทั้งหมดแล้ว ให้ตั้งชื่อแอนิเมชันแล้วกอ OK เพื่อบันทึก

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

8.6 บิตแมบเอพีไอ (Bitmap API)

8.6.1 การโหลดบิตแมบ

การที่เราจะใช้บิตแมบได้ต้องโหลดมันขึ้นมาจากดิสก์ก่อนโดยใช้ฟังก์ชัน

DWORD Morfit_bitmap_load(char *file_name, int transparent_index);

ฟังก์ชันนี้จะส่งค่าแฮนเดิลของบิตแมบที่โหลดขึ้นมา

ชื่อไฟล์ที่เราจะแทนใน file_name ไม่ต้องใส่นามสกุล แต่ถ้าเราใส่เอนจิ้นจะไม่สนใจอยู่ดี ในขั้นตอนการโหลดเอนจิ้นจะหาไฟล์ที่มีนามสกุลเป็น JBM, BMP และ JPEG ตามลำดับ เพราะฉะนั้นถ้าคุณมีรูปชื่อเดียวกัน 2 รูป ที่มีนามสกุลเป็น BMP และ JPEG เอนจิ้นจะโหลดรูป 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 เอนจิ้นจะมองว่าเป็นภาพคนละภาพกัน และ จะส่งค่าแฮนเดิลค่ากลับมา 2 ค่า

เมื่อเราโหลดบิตแมบขึ้นมาแล้วครั้งหนึ่ง 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) ในการอ่านค่าแฮนเดิลบิตแมบโดยกำหนดชื่อที่เราโหลดขึ้นมาแล้ว

ส่วนการบันทึกบิตแมบที่เราแก้ไขไปทำได้โดยใช้ฟังก์ชัน

Morfit_bitmap_save(DWORD bitmap_handle, char *full_path_name);

ในฟังก์ชันนี้เราต้องระบุนามสกุลของไฟล์ที่จะบันทึกด้วย ดังตัวอย่าง

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

ตามตัวอย่างจะบันทึกบิตแมบเป็นไฟล์แบบ BMP และถ้า full_path_name เป็น NULL จะบันทึกไฟล์ที่ ที่โหลดไฟล์นั้นขึ้นมา (อ้างอิงจากค่าแฮนเดิล)

สำหรับการอันโหลดบิตแมบออกจากหน่วยความจำให้ใช้

Morfit_bitmap_unload(DWORD bitmap_handle);

8.6.2 สีของบิตแมบ

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);

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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);

หากเราโหลดบิตแมบขึ้นมาแล้วต้องการให้สีแดงสด (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);
```

ส่วนการตรวจสอบว่าบิทแมทรูปนี้ใช้สีใสอะไรให้ใช้

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

ฟังก์ชันนี้จะส่งกลับค่า VR_ERROR ถ้าบิตแมบไม่มีการให้สีใส ส่วนการตรวจค่า *transparent_index* ให้ใช้

```
int Morfit_bitmap_get_transparent_index(DWORD bitmap_handle);
```

8.6.3 เปลี่ยนขนาดบิตแมบ

ฟังก์ชันนี้จะสร้างบิตแมบขึ้นมาใหม่จากบิตแมบที่เรากำหนดโดยจะส่งค่าแฮนเดิลของบิตแมบใหม่กลับมา

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

เราสามารถสร้างสำเนาของบิตแมบได้โดยใช้ค่า new_width และ new_height เป็นค่าเก่า ส่วนการตรวจสอบค่าทำได้โดยใช้ฟังก์ชัน

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

8.6.4 ฟังก์ชันบิตแมบอื่นๆ

ฟังก์ชันนี้ใช้ตรวจสอบจำนวนโพลีกอนที่บิตแมบวางอยู่

```
int Morfit_bitmap_get_number_of_polygons_using_bitmap(DWORD bitmap_handle);
```

ส่วนการวางบิตแมบลงบนโพลีกอนเราจะกล่าวถึงในบทถัดไป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
BYTE * Morfit_bitmap_get_memory(DWORD bitmap_handle);
```

ฟังก์ชันนี้จะรับค่าแฮนเดิลบิตแมบ และ ส่ง BYTE array of size *width*height* กลับมา ทำให้เราแก้ไขพิกเซลแต่ละพิกเซลของบิตแมบได้ ข้อมูลของบิตแมบถูกเก็บเป็นแอร์เรย์ของแถวโดยมีจุดเริ่มที่ด้านบนซ้ายของรูป 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

เราสามารถแก้ไขบิตแมบได้ โดยแก้ไขอาร์เรย์ของบิตแมบ 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 บิตแมบจะทำได้ในโหมดซอฟท์แวร์เรนเดอร์เท่านั้น

8.7 การจัดการกับแอนิเมชัน

8.7.1 การสร้างแอนิเมชัน

ทำได้โดยใช้ DWORD Morfit_animation_create(char *name);

ฟังก์ชันจะสร้างแอนิเมชันขึ้นมาใหม่ และ ส่งค่าแฮนเดลกลับมาให้ 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) ใช้ตรวจสอบชื่อของแอนิเมชัน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

Morfit_animation_set_name(DWORD animation_handle, char * name) ใช้แก้ไขชื่อของแอนิเมชัน

DWORD Morfit_animation_get_handle(char *animation_name); ใช้อ่านค่าแฮนเดิลของแอนิเมชันที่มีชื่อตามที่ระบุ

Morfit_animation_delete(DWORD animation_handle); ใช้สำหรับลบแอนิเมชันออกจากหน่วยความจำ

8.7.2 การเพิ่มบิตแมบ

หลังจากที่เราสร้างแอนิเมชันขึ้นมา เราก็ต้องใส่บิตแมบให้เซตทั้ง 5 ซึ่งเคยกล่าวไว้ในตอนแรก

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

animation_handle คือ แฮนเดิลของแอนิเมชันที่จะใส่บิตแมบเข้าไป

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 แฮนเดิลของบิตแมบที่จะใส่เข้าไป

bitmap_position_index ค่าที่ระบุว่าเป็นลำดับขั้นการแสดงภาพที่เท่าไร 0 หมายถึงภาพที่จะแสดงผลในตอนเริ่มแสดง, 1 หมายถึงภาพที่จะแสดงผลถัดมาจากภาพที่ 0 ค่า –1 หมายถึงหมายถึงภาพบิตแมบสุดท้ายใน List ฟังก์ชันนี้จะส่งคลับค่าลำดับขั้นการแสดงภาพจริงที่กลับมา ถ้าค่าเป็น –1 หมายถึง ลำดับที่เราระบุมากเกินไป หรือ อื่นๆ

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

การจะลบบิตแมบออกจากแอนิเมชันให้ใช้

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

bitmap_postion_index หมายถึงลำดับของบิตแมบที่จะลบ โดย –1 หมายถึงลำดับท้ายสุด

การตรวจสอบค่าแฮนเดิลของบิตแมบ ณ ตำแหน่งที่กำหนด

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

การตรวจสอบว่าในแอนิเมชันมีบิตแมบเท่าไรให้ใช้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
int Morfit_animation_get_number_of_frames(DWORD animation_handle);
int Morfit_animation_get_number_of_bitmaps(DWORD animation_handle);
```

ฟังก์ชันแรกจะส่งจำนวนเฟรม (จำนวนบิตแมบในเซต) กลับ ส่วนฟังก์ชันที่สองจะส่งจำนวนบิตแมบทั้งหมดในแอนิเมชันกลับ ยกตัวอย่าง เช่น ถ้าเรามีอยู่ 4 บิตแมบใน FRONT, SIDE, และ BACK แต่ไม่มีเลยใน FRONT_SIDED และ BACK_SIDED ฟังก์ชันแรกจะส่งกลับค่า 4 ฟังก์ชันที่สองจะส่งกลับค่า 12

ฟังก์ชันนี้จะเขียนค่าแฮนเดิลของบิตแมบทั้งหมดในแอนิเมชันลงอาร์เรย์

```
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];)

8.7.3 ระยะเวลาของแต่ละเฟรม

เฟรมไทม์ (Frame time) หมายถึงเวลาที่จะแสดงผลบิตแมบนั้นค้างไว้ก่อนจะแสดงผลบิตแมบถัดไป หน่วยที่ใช้ในเอนจิ้น Morfit คือ 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) จะกำหนดให้แสดงบิตแมบแรกค้างไว้ 2 วินาทีก่อนจะเปลี่ยนไปบิตแมบถัดไป

ส่วนการตรวจสอบค่าเฟรมไทม์ของบิตแมบให้ใช้

```
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;
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
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 ถ้าแอนิเมชันมีบิตแมบอยู่ 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

8.8 ฟังก์ชันที่มีประโยชน์อื่นๆ

ฟังก์ชันนี้จะส่งกลับค่า True เมื่อ แอนิเมชันถูกใช้ในโลกที่โหลดไว้ ถ้าไม่ก็จะส่งกลับค่า False

```
int Morfit_animation_is_part_of_the_last_world(DWORD animation_handle);
```

การตรวจสอบจำนวนโพลีกอนที่วางแอนิเมชันนั้นลงไปใช้

```
int Morfit_animation_get_number_of_polygons_with_animation(DWORD animation_handle);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 9
โพลีกอนเอพีไอ (Polygon API) ของมอร์ฟท

9.1 นิยามของโพลีกอน

โพลีกอนเป็นส่วนประกอบที่สำคัญในแต่ละวัตถุ โดยวัตถุจะประกอบด้วยโพลีกอนจำนวนมากต่อเข้าด้วยกันกลายเป็นรูปทรงขึ้นมา ซึ่งโพลีกอนแต่ละชิ้นก็คือกลุ่มของเส้นตรงที่เชื่อมต่อกันและอยู่บนระนาบเดียวกัน โดยจะต้องต่อกันเป็นวงปิดพอดี และมุมภายในแต่ละมุมจะต้องมีค่าน้อยกว่า 180 องศา

9.2 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);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- 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 ของจุดที่จะแปะบิตแมบลงไป

xyz_bmX_bmY_brightness[4] = ค่าในแนวแกน y ของจุดที่จะแปะบิตแมบลงไป

xyz_bmX_bmY_brightness[5] = ค่าความสว่างของจุด

ตัวอย่าง ในการสร้างโพลีกอนและเพิ่มจุดเข้าไปที่ตำแหน่ง (10, 10, 10), (20,10,10), และ (20,20,20) โดยไม่ใช้บิตแมบ:

```
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);
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- 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

9.3 การหมุนและเคลื่อนโพลีกอน

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 + polygon_x_location
actual_y_location = object_y_location + polygon_y_location
actual_z_location = object_z_location + polygon_z_location
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- 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);
```

9.4 การกำหนดคุณสมบัติสีให้โพลีกอน

9.4.1 คุณสมบัติด้านเดียว โดยปกติโพลีกอนจะมีเพียงด้านเดียว ถ้าเรามองจากอีกด้านที่ไม่มีบิตแมบเราจะมองทะลุผ่านโพลีกอนนั้นไปเลย (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 อาจจะเป็นโพลีกอนที่มาจากวิธีอื่นก็ได้*/
Morfit_polygon_flip_visible_side(flipped);
```

** ถ้าเราต้องการเลื่อนหรือหมุนโพลีกอนอันใดอันหนึ่งเราจะต้องหมุนอีกอันหนึ่งด้วย ดังนั้นเราจึงควร group มันไว้ด้วยกันจะทำให้ง่ายขึ้น

9.4.2 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
```

9.4.3 คุณสมบัติอื่น ๆ มีการทำงานที่สำคัญ ๆ ได้แก่

- การกำหนดให้โพลีกอนอยู่ในการตรวจสอบการชนหรือไม่มีดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
Morfit_polygon_make_non_collisional(DWORD polygon_handle);
//ค่าปกติ Morfit จะกำหนดให้โพลีกอนอยู่ในการตรวจสอบการชนอยู่แล้ว//
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
```

9.5 การกำหนดสีให้โพลีกอน

- 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

9.6 บิตแมบ และ แอนิเมชัน

- การแปะบิตแมบบนโพลีกอน ทำได้โดย

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

- การหาชื่อและแฮนเดิลของบิตแมบที่แปะอยู่กับโพลีกอน ทำได้โดย

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

** จะคืนค่า NULL ถ้าไม่มีบิตแมบแปะอยู่

9.6.1 สีและความสว่างของบิตแมบ มีการทำงานที่สำคัญ ๆ ดังนี้

- การใช้สีโปร่งใสของบิตแมบ (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);
```

- การกำหนดความสว่างของบิตแมบ จะมีประโยชน์มากโดยเฉพาะงานที่เราต้องการแปะบิตแมบไปที่โพลีกอนอื่นโดยให้มีความสว่างต่างกัน เราไม่จำเป็นต้องสร้างบิตแมบขึ้นใหม่เพียงแค่แปะบิตแมบนั้นลงไปแล้วกำหนดค่าความสว่างใหม่เท่านั้น โดย:

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

** *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 บิตแมบจะสว่างขึ้นหรือมืดลงเมื่อเข้าใกล้แต่ละจุด 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);
```

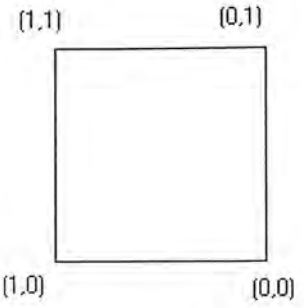
9.6.2 การพลิกบิตแมบ บิตแมบจะต้องเป็น 4 เหลี่ยมมุมฉากแต่โพลีกอนนั้นไม่จำเป็น โดย Morfit จะใช้พิกัดบิตแมบ (bitmap coordinates) ในการระบุว่าบิตแมบจะถูกแปะลงไปอย่างไร โดยทุก ๆ บิตแมบซึ่งเป็น 4 มุมฉากจะมีมุมบนซ้ายเป็นจุด (0, 0) และล่างซ้ายเป็นจุด (1, 1) ถ้ายังจำกันได้ ฟังก์ชัน Morfit_polygon_add_point() array ที่เป็นพารามิเตอร์ตัวที่สองจะมีการเก็บค่าพิกัดของจุดในโพลีกอนและพิกัดบิตแมบของจุดนั้นด้วยซึ่งจะระบุว่าจะแปะบิตแมบที่พิกัดใดลงไปบนจุดนั้น

ตัวอย่างเช่นถ้าเรามีบิตแมบดังรูป:

รูปที่ 9-1

และเรามีโพลีกอนซึ่งระบุพิกัดบิตแมบของจุดยอดไว้ดังรูป:

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 9-2

แล้วบิตแมบจะถูกแปะลงไปดังรูป:

รูปที่ 9-3

**จะเห็นได้ว่ามุมซ้ายบนของบิตแมบจะถูกแปะลงไปที่ (0, 0) ส่วนมุมขวาล่างที่จุด (1, 1)

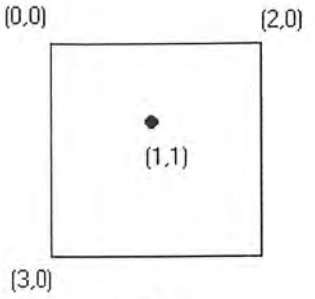
**เราเพียงกำหนดค่าพิกัดไป 3 จุดแล้ว engine จะคำนวณจุดที่เหลือให้เอง ถ้าเราต้องการให้ engine คำนวณพิกัดบิตแมบของจุดยอดให้ด้วยก็ให้ใช้พิกัด (-1 ,-1)

เราสามารถแปะบิตแมบไปตามขวางของโพลีกอนได้ด้วยวิธีดังนี้:

รูปที่ 9-4

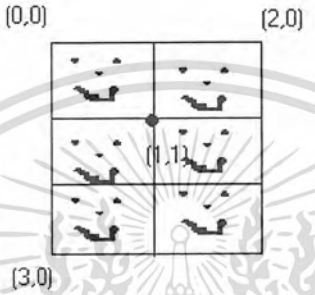
แล้ว engine จะคำณวนจุด (1, 1):

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 9-5

จากนั้นจะแปะบิตแมบไปดังนี้:



รูปที่ 9-6

และเรายังสามารถวางบิตแมบลงไปบนโพลีกอนที่ไม่เป็น 4 เหลี่ยมได้ดังนี้:

รูปที่ 9-7

- ในการหมุนบิตแมบ ทำได้โดย

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

จะเป็นการหมุนพิกัดบิตแมบของแต่ละจุดยอดในโพลีกอน

- ในการพลิกบิตแมบไป 180 องศา ทำได้โดย

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

- ในการแปะบิตแมบไปยังโพลีกอนถัดไปที่อยู่ติดกันไปเรื่อย ๆ ทำได้โดย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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

9.6.3 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);
```

9.6.4 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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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);
```

9.7 ฟังก์ชันทางคณิตศาสตร์ของโพลีกอน

- 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

9.8 โพลีกอน และ กรุ๊ป

- การเพิ่มโพลีกอนเข้าไปใน group ทำได้โดย

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

- การหา group ที่เป็นเจ้าของโพลีกอนใด ๆ ทำได้โดย

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

- การตรวจสอบว่ามีโพลีกอนใดอยู่ใน group หรือไม่ ทำได้โดย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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);

9.9 Patches

patches เป็นทางเดียวที่จะทำให้เราสร้างโพลีกอนขึ้นใหม่ได้ใน Viewer mode โดยจะเป็นโพลีกอนที่ติดอยู่บนโพลีกอนอื่น อย่างเช่น การสร้างรอยจากกระสุนบนนกำแพง 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 ใช้ในการจัดวางบิตแมบ แต่ถ้าจะเป็นอย่างไรก็ได้ให้ใช้ {0,0,0}

bitmap_handle คือบิตแมบที่จะแปะบน patch แต่ถ้าใช้การเติมสีแทนให้ใช้ NULL

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

- 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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 10
แนะนำเอพีไออื่นๆ ของมอร์ฟิท

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

10.1 พอย์ทเอพีไอ (Point API)

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

10.2 แทร็กเอพีไอ (Track API)

วัตถุ, กล้อง และ Group สามารถตั้งให้วิ่งไปตามแทร็กได้ แทร็กก็เป็นเพียงจุดบนแกน 3 มิติหลายๆ จุด มาเชื่อมโยงกันเป็นทางเท่านั้นเอง Track API สามารถใช้สร้าง และ แก้ไขแทร็กได้

10.3 เอพีไอการ์ด 3 มิติ (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

10.4 เอพีไอฉากหลัง (Background API)

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

10.5 เอพีไอคณิตศาสตร์ (Math API)

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

10.6 ยูทิลลิตี้เอพีไอ (Utility API)

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

10.7 โปรไพเลอร์เอพีไอ (Profiler API)

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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 11

การจัดการกับเงา และ แสง โดยใช้มอร์ฟิท

เราจะใช้ patch ในการสร้างแสงและเงา เพราะในงานดังกล่าวไม่จำเป็นต้องมีการสร้างรูปทรงใหม่ขึ้นมา

11.1 เงา

การสร้างเงาต้องใช้วัตถุ 3 อย่างได้แก่ แหล่งกำเนิดแสง, โพลีกอนที่จะส่องเงาลงไป และโพลีกอนที่จะรับเงา ดังรูป:

รูปที่ 11-1 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 จะถือว่าเป็นทั้ง world

light_src[3] เก็บตำแหน่งของแหล่งกำเนิดแสงตามพิกัด x, y, z และเนื่องจากใช้ในการสร้างเงาจึงควรเป็นแหล่งกำเนิดแบบจุด

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

```
Morfit_engine_create_shadow(room_group, person_object, light, 255);
Morfit_engine_create_shadow(room_group, piano_group, light, 254);
.
.
.
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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

เรายังสามารถใช้ serial number ในการเปลี่ยนสีหรือบิตแมบของแต่ละกลุ่มของ 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

11.1.1 สีของเงา

เงาจะสืบทอดคุณคุณสมบัติมาจากวัตถุที่ฉายเงาเช่นวัตถุโปร่งแสงจะให้เงาที่โปร่งแสง, คนจะให้เงาเป็นรูปบิตแมบของคน เป็นต้น ซึ่งในกรณีหลังเรามักไม่ต้องการ แต่เราสามารถกำหนดสีของเงาให้เป็นสีทึบเพื่อความถูกต้องได้โดย:

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

การเติมบิตแมบไปบนเงาทำได้โดย:

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

เรามักต้องการให้เงามีคุณสมบัติโปร่งแสง 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

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

11.1.2 Dynamic Patches

การสร้าง patches อาจเป็นงานที่ช้ามาก เราสามารถทำให้เร็วขึ้นได้โดยใช้โพลีกอนเป็นทั้งตัวสร้างเงาและตัวรับเงา แต่ในบางครั้งการทำเช่นนี้ก็ยังช้าเกินไป ดังนั้นสำหรับวัตถุและแหล่งกำเนิดแสงที่ไม่เคลื่อนไหว เราสามารถสร้างเงาเพียงครั้งเดียวในตอนเริ่มโปรแกรม

ส่วนการสร้างเงาจากวัตถุหรือแหล่งกำเนิดแสงที่เคลื่อนที่นั้น เราสามารถทำให้เร็วขึ้นโดยใช้:

```
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);
```

11.2 แสง

แสงสร้างโดยวิธีการใกล้เคียงกับเงา โดยเทียบแสงได้กับ "เงาสว่าง" (ดูรูปด้านล่างประกอบ):

รูปที่ 11-2 การเกิดแสง

เราจะเห็นได้ว่าแหล่งกำเนิดฉายแสงไปบนโพลีกอนเป้าหมาย แต่เนื่องจากมอร์ฟิทเอนจิ้นใช้แหล่งกำเนิดแสงแบบจุด เราจะสร้างแสงได้ดังรูปที่ 11.3:

รูปที่ 11-3

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จะเห็นได้ว่าเหมือนกับการสร้างเงา โดยเราใช้ แหล่งกำเนิดแสงแบบจุด, โพลีกอนคั่นกลาง และโพลีกอนรับแสง ต่างกันเพียงแค่แสงจะสร้างเงาที่ทำให้วัตถุที่รับแสงสว่างขึ้นไม่ใช่มืดลง

การสร้างแสงทำได้โดยใช้ Morfit_engine_create_shadow() แต่ก่อนจะสร้างเราต้องกำหนดสีของ patches ที่สว่างขึ้น หรือใช้บิตแมบที่สว่าง 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 จะสืบทอดคุณสมบัติมาจากโพลีกอนที่สร้างมัน แทนที่เราจะกำหนดคุณสมบัติของ patches ทุกครั้งที่สร้างมัน เราเพียงแค่กำหนดคุณคุณสมบัติของโพลีกอนคั่นกลางครั้งเดียวเท่านั้น:

```
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);
```

เมื่อเรากำหนดคุณสมบัติต่าง ๆ ของ patches หรือโพลีกอนคั่นกลางดังข้างต้นแล้วเราสามารถสร้างแสงได้โดย:

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

การกำหนดคุณสมบัติทางแสงของโพลีกอนดังข้างต้นจะมีประโยชน์ในกรณีที่เราต้องสร้างแสงขึ้นใหม่บ่อยเช่น แหล่งกำเนิดแสงเคลื่อนที่ได้ เป็นต้น

*การสร้างแสงจะซับซ้อนกว่าเงาเล็กน้อยตรงที่เราต้องสร้างโพลีกอนคั่นกลาง ซึ่งทำได้โดยวางโพลีกอนคั่นระหว่างแหล่งกำเนิดแสงกับส่วนที่เหลือของโลก แล้วใช้ Morfit_polygon_disable() เพื่อทำให้มองไม่เห็นมัน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 12

โหมดแก้ไข (Editor Mode) และ โหมดแสดงผล (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 เอนจิ้นจะสร้างดาต้าสตรักเจอร์ชื่อ BSP-Tree โดยขนาดของดาต้าสตรักเจอร์จะขึ้นอยู่กับจำนวนโพลีกอนที่มีอยู่ในโลก เนื่องจากในโลกที่มีโพลีกอนมากๆ บางครั้งการสร้าง BSP-Tree อาจจะใช้เวลาเป็นชั่วโมง เอนจิ้นจึงบันทึก 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

12.1 สาเหตุที่ควรเลือกใช้โหมดแสดงผล (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 วัตถุแบบสแตติกเท่านั้นที่ถูกนำไปใช้ในการคำนวณสร้าง BSP-Tree ถ้าโลกที่คุณสร้างมีแต่วัตถุแบบไดนามิกการใช้ BSP-Tree ก็ไม่ได้ทำให้เรนเดอร์ได้เร็วขึ้น

คุณไม่ต้องสร้างวัตถุขึ้นในช่วงเวลาที่รันโปรแกรม (Run-time)

บางครั้งเราต้องการที่จะทำลายวัตถุและสร้างขึ้นมาใหม่ เช่น เมื่อฆ่าศัตรู หรือ มีศัตรูแบบใหม่โผล่ออกมา เราไม่จำเป็นต้องทำลาย หรือ สร้างวัตถุขึ้นมาใหม่จริงๆ เราเพียงแต่เอนเนเบิล กับ ดิสเอเบิลวัตถุนั้นก็พอ

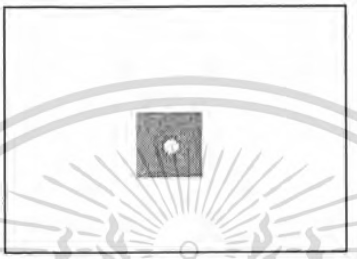
12.2 สาเหตุที่ควรเลือกใช้โหมดแก้ไข (Editor mode)

เราต้องสร้างวัตถุขึ้นมาใหม่ขณะรันไทม์

ต้องย่อขยาย หรือ แก้ไขวัตถุ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

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



รูปที่ 12-1

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 13
ปัญหาที่พบ

13.1 ประสิทธิภาพ

เกมเอนจิ้นมอร์ฟิทนั้นมีความเร็วอยู่ในระดับน่าพอใจเมื่อแสดงผลวัตถุ 3 มิติ ที่ไม่มีการเปลี่ยนแปลง แต่เมื่อเรียกใช้โมเดลเอ็มดีทู ซึ่งเป็นโมเดลที่สามารถเก็บข้อมูลแอนิเมชันของวัตถุ จะทำให้เฟรมเรทตกลงอย่างมาก แม้ว่าโมเดลเอ็มดีทูที่เรียกใช้จะมีโพลีกอนเพียงไม่กี่พันโพลีกอน จากการทดสอบพบว่าเราสามารถโหลดโลก 3 มิติ ที่มีโพลีกอนหลายหมื่นโพลีกอนขึ้นมา และสามารถแสดงผลโดยที่มีเฟรมเรทอยู่ในช่วง 20 - 30 เฟรมต่อวินาทีแล้ว แต่ถ้าโหลดโมเดลเอ็มดีทู ที่ใช้โพลีกอน 2,000 –3,000 โพลีกอนก็ทำให้เฟรมเรทตกลงเหลือเพียง 10-18 เฟรมต่อวินาทีแล้ว แม้ว่าจะเปลี่ยนความละเอียดของการแสดงผลเป็น 1024 x 768 แต่ก็ทำให้เฟรมเรทตกลงไปไม่ถึง 2 เฟรม ทางผู้วิจัยจึงคาดว่าน่าจะเป็นเพราะอัลกอลิทึมของเอนจิ้นที่ใช้ในการทรานซ์ฟอร์มโพลีกอน มีประสิทธิภาพต่ำ และ ใช้ทรัพยากรของระบบสูงเกินไป นอกจากนี้การทำให้ภาพที่แสดงผลออกมาดูดีต้องใช้เทคนิคหลายอย่างโดยที่บางเทคนิค ต้องสร้างแหล่งกำเนิดแสงขึ้นมาก่อน (ไลท์ติง หรือ Lighting) อย่างเช่น เงา (Shadow) หรือ เฉดดิง (Shading) แต่มอร์ฟิทใช้ทรัพยากรสูงมากในการทำไลท์ติง แค่สร้างแหล่งกำเนิดแสงขึ้นมาเฟรมเรทก็ตกลงไป 2 เฟรมแล้ว

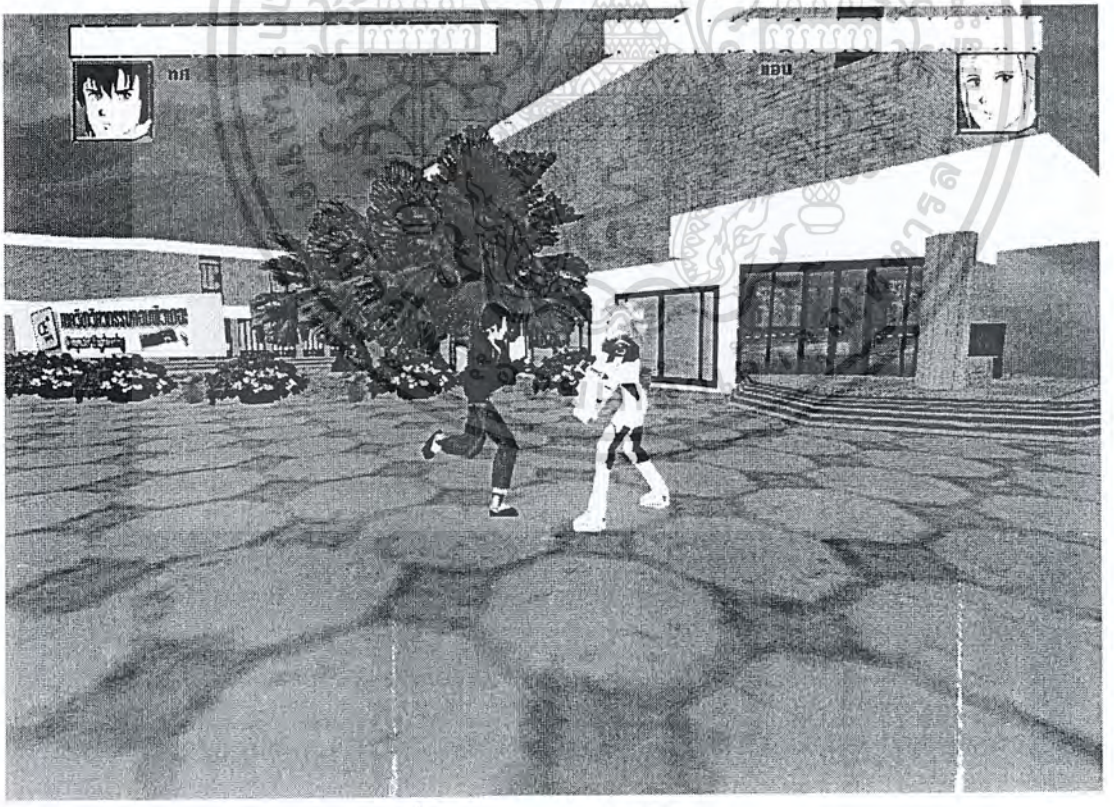
13.2 เอเลียสซิง (Aliasing)

ปัญหานับว่าเป็นปัญหาคลาสสิคของการแสดงผลกราฟฟิคบนคอมพิวเตอร์เลยทีเดียว หากสามารถแอนติเอเลียสซิงได้ ภาพที่แสดงผลออกมาจะดูดีโดยที่ไม่ต้องแสดงผลที่ความละเอียดสูง แต่ภาพที่มอร์ฟิทแสดงกลับเอเลียสซิงมากจนอยู่ในระดับผิดปกติ กล่าวได้ว่าหากการ์ด 3 มิติ ที่ใช้ในระบบนั้นไม่มีคุณสมบัติการทำแอนติเอเลียสซิงแล้วละก็ภาพที่แสดงออกมาจะดูแย่มาก เนื่องจากตัวการ์ดแสดงผล 3 มิติ GeForce 256 ที่ทางสถาบันให้การสนับสนุนนั้นมีความสามารถที่เรียกว่า FSAA (Full Screen Anti-Aliasing) จึงสามารถทดสอบกับมอร์ฟิทได้ ต้องขอขอบพระคุณมา ณ ที่นี้อีกครั้ง แม้มอร์ฟิทไม่สามารถเรียกใช้ความสามารถนี้ได้ แต่ไดรเวอร์ของ GeForce สามารถบังคับให้แอพพลิเคชันที่มีการเรียกใช้ไดเรกทรีดี หรือ โอเพนจีเอล แสดงผลโดยใช้ความสามารถแอนติเอเลียสซิงได้ โดยได้ผลดังรูป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 13-1 การแสดงผลโดยไม่ได้ใช้ FSAA

ในรูปบริเวณที่เขียนล้อมไว้เป็นบริเวณที่มีการแสดงผลผิดพลาด เนื่องจากการเอเลียสซิง ในรูปถัดไปแม้จะใช้ FSAA แล้ว จะลดความผิดพลาด และ ทำให้ภาพดูดีขึ้น แต่ก็ยังเหลือปัญหาอยู่



รูปที่ 13-2 การแสดงผลโดยใช้ FSAA

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

13.3 จำนวนโพลีกอน

แม้ว่าโพลีกอนในโลก 3 มิติ ทั้งหมดจะเป็นโพลีกอนที่ไม่มีการเคลื่อนไหว หรือ เปลี่ยนแปลงรูปร่างเลย แต่ถ้าจำนวนโพลีกอนมากกว่า 100,000 โพลีกอน เฟรมเรทก็จะลดลงเหลือไม่ถึง 10 เฟรมต่อวินาทีทั้งๆ ที่การ์ดแสดงผลที่ใช้สามารถแสดงผลโพลีกอนได้ถึงหลักล้านโพลีกอน เนื่องจากมอร์ฟิทพัฒนาขึ้นบนไดเรกซ์เอ็กซ์ และ ผู้วิจัยยังไม่เคยพบเกมสามมิติที่พัฒนาโดยใช้ไดเรกซ์ทรีดีที่สามารถแสดงผลโพลีกอนหลายล้านโพลีกอนได้เรียลไทม์ จึงไม่สามารถระบุได้ว่าข้อจำกัดนี้เกิดจากไดเรกซ์ทรีดี หรือ มอร์ฟิท

13.4 ขีดจำกัดของจำนวนสีของบิตแมบที่ใช้เป็นพื้นผิว หรือ ฉากหลัง

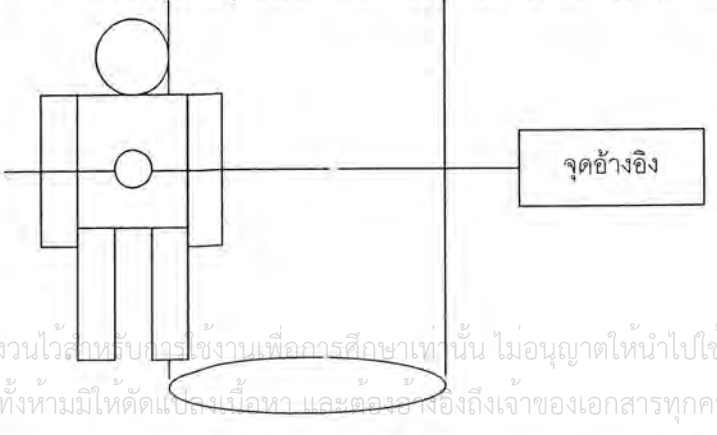
ในเกม 3 มิติยุคปัจจุบันจะใช้เท็กซ์เจอร์ขนาด 16 - 24 บิทสี หรือ 65536 – 16.7ล้านสี เป็นมาตรฐานไปแล้ว จึงสามารถกล่าวได้ว่าเกม 3 มิติที่ใช้เท็กซ์เจอร์ที่มีสีเพียง 256 สี เป็นเกมที่พัฒนาขึ้นโดยใช้เทคโนโลยีพ้นสมัยไปแล้ว และ มอร์ฟิทสามารถเรียกใช้เท็กซ์เจอร์ 256 สีได้เท่านั้น แม้ว่าจะสามารถเรียกใช้รูปที่มีสีมากกว่า 256 สี ก็ตาม แต่เมื่อนำไปใช้จริงมอร์ฟิทจะลดสีลงเหลือ 256 สี

13.5 การเข้าถึงแอนิเมชันของโมเดลเอ็มดีทู (MD2)

หลังจากโหลดโมเดลเอ็มดีทูขึ้นมาแล้ว หากต้องการโหลดโมเดลเอ็มดีทูขึ้นมาอีกตัวสามารถทำได้ แต่จะไม่สามารถเข้าถึงข้อมูลแอนิเมชันของโมเดลเอ็มดีทูตัวที่สองได้หากโมเดลเอ็มดีทูตัวที่สองนั้นเป็นโมเดลเดียวกับตัวแรก แม้ว่าจะโหลดมาจากไฟล์สำเนาที่มีชื่อต่างกันก็ตาม หลังจากส่งจดหมายอิเล็กทรอนิคไปสอบถาม ทางมอร์ฟิทกล่าวว่า " ปัญหานี้จะถูกแก้ไขอย่างแน่นอนในเวอร์ชันที่ 5 นอกจากนี้เรายังเพิ่มความสามารถที่น่าสนใจหลายอย่างเข้าไป " อย่างไรก็ดีในเวอร์ชันที่ 5 ไม่ได้เป็นฟรีแวร์ และ มีราคา 6,500 เหรียญสหรัฐ

13.6 ข้อบกพร่องของการตรวจสอบการชนโดยใช้ฟังก์ชันของมอร์ฟิท

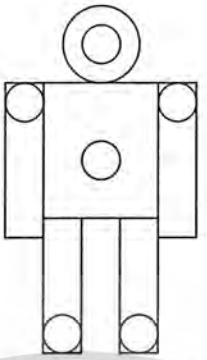
เนื่องจากฟังก์ชันตรวจสอบการชนบนมอร์ฟิท (Morfit_object_is_movement_possible) มีลักษณะการทำงานดังนี้ คือ ให้กำหนดจุดเริ่มต้น และ จุดปลายของวัตถุที่ต้องการจะตรวจสอบ โดยที่ต้องบอกค่าแฮนเดิลของวัตถุที่ต้องการจะตรวจสอบ จากการที่ทราบจุดเริ่มกับจุดปลายก็จะสามารถนำจุดทั้ง 2 มาสร้างเส้นตรงได้ แล้วนำเส้นตรงมาตรวจสอบว่ามีการตัดกับโพลีกอนใดหรือไม่ ถ้าไม่ฟังก์ชันจะส่งกลับค่า YES แต่ถ้ามีการตัดจะส่งกลับค่า NO พร้อมทั้งบอกจุดตัดกับโพลีกอน, แฮนเดิลของโพลีกอน และวัตถุที่เส้นตรงตัดกลับคืนมา แต่เนื่องจากวัตถุในโลก 3 มิติเป็นปริมาตรไม่ใช่เส้นตรง ทำให้เกิดปัญหาขึ้น



เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

รูปที่ 13-3 ตัวละครสามารถเดินผ่านเลยไปได้ ทั้งๆ มีบางส่วนของร่างกายติดเสาอยู่

จากรูปกำหนดให้วงกลมกลางลำตัวคนเป็นจุดอ้างอิง ที่จะเป็นจุดเริ่ม และ จุดปลาย จะเห็นว่าตัวคนกับเสามีส่วนที่ทับกันอยู่ คนจึงไม่ควรจะเดินผ่านเสาไปได้ แต่ถ้าใช้คำสั่งนี้ จะพบว่าเส้นที่ใช้ตรวจสอบนั้นไม่ได้ตัดกับเสาเลยทำให้คนสามารถเดินผ่านเสาไปได้



รูปที่ 13-4 การกำหนดจุดอ้างอิงมากกว่า 1 จุด

อย่างไรก็ดีการแก้ไขปัญหาได้โดยเพิ่มจุดอ้างอิงที่ใช้ตรวจสอบให้มากขึ้นดังรูปที่ 13.6.2 จะทำให้ปัญหานี้ทุเลาลง แต่ก็เพิ่มปัญหาใหม่มาอีก 2 อย่าง คือ

1 ใช้การประมวลผลมากขึ้นทำให้เฟรมเรทลดลง

2 ไม่สามารถแก้ไขปัญหาได้สมบูรณ์ ถ้าเจอวัตถุขนาดเล็กขวางอยู่โดยที่เส้นตรงที่เกิดจากจุดอ้างอิงไม่ได้ตัดกับวัตถุนั้นเลย ก็จะสามารถเดินผ่านวัตถุนั้นไปได้อีก

13.7 การตรวจสอบการชนระหว่างโมเดลเอ็มดีทู กับ โมเดลเอ็มดีทูโดยใช้ฟังก์ชันของมอร์ฟิท

ฟังก์ชันตรวจสอบการชนของมอร์ฟิท เมื่อใช้กับโมเดลเอ็มดีทู พบว่าเมื่อโมเดลเอ็มดีทูเดินไปชนโมเดลเอ็มดีทูอีกตัวฟังก์ชันนี้จะสามารถตรวจการชนได้ไม่เสมอไป บางครั้งโมเดลเอ็มดีทูจะสามารถเดินทะลุกันได้

13.8 การโหลดบิตแมบมากว่าหนึ่งบิตแมบ

การโหลดบิตแมบทำได้โดยใช้ฟังก์ชัน Morfit_bitmap_load ฟังก์ชันนี้จะทำงานโดยโหลดบิตแมบจากไฟล์เข้ามาเก็บไว้ที่หน่วยความจำและจะส่งกลับค่าแฮนเดิลที่จะใช้เข้าถึงบิตแมบนั้น แต่พบว่าเมื่อโหลดบิตแมบมากกว่าหนึ่งโดยใช้คำสั่งนี้ ในการโหลดบิตแมบถัดมา ฟังก์ชันนี้จะส่งกลับค่าแฮนเดิลเดียวกับแฮนเดิลของบิตแมบแรกสุด ทำให้ไม่สามารถโหลดบิตแมบได้มากกว่าหนึ่ง แต่ทางผู้วิจัยพบว่าถ้าตั้งค่า transparent_index ให้ไม่ซ้ำกันจะสามารถโหลดบิตแมบได้มากกว่าหนึ่ง แต่ก็จะมีปัญหาเพราะในบิตแมบที่โหลดเข้าไปจะมีบางส่วนที่เป็นสีใสตามค่า transparent_index

13.9 การใช้ความสามารถเฉพาะตัวของการ์ด 3 มิติ

การที่ไม่สามารถเข้าถึงความสามารถเฉพาะตัวของการ์ดแสดงผล 3 มิติ ทำให้มีผลต่อคุณภาพของภาพที่แสดงออกมา และ ประสิทธิภาพของเกม ความสามารถเฉพาะของการ์ดหลายอย่างที่ทำให้ภาพที่

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

แสดงออกมาดูเหมือนจริง เช่น บัมพ์แมบบิง (Bump mapping) ประเภทต่างๆ, เอฟเฟคหมอก (Fog Effect) ส่วนที่มีผลต่อประสิทธิภาพของเกมก็เช่น ทรานซ์ฟอร์ม และ ไลท์ติงโดยใช้ฮาร์ดแวร์ (Hardware T&L) และ ไทล์เบสเรนเดอร์ริง (Tile-Based Rendering) แม้มอร์ฟิทจะพัฒนาบนไดเรกเอ็กซ์ แต่ในมอร์ฟิทเวอร์ชันที่ใช้พัฒนายังไม่สามารถเรียกใช้ความสามารถเฉพาของการ์ด 3 มิติได้

13.10 การเล่นแอนิเมชันซ้ำ 2 รอบ

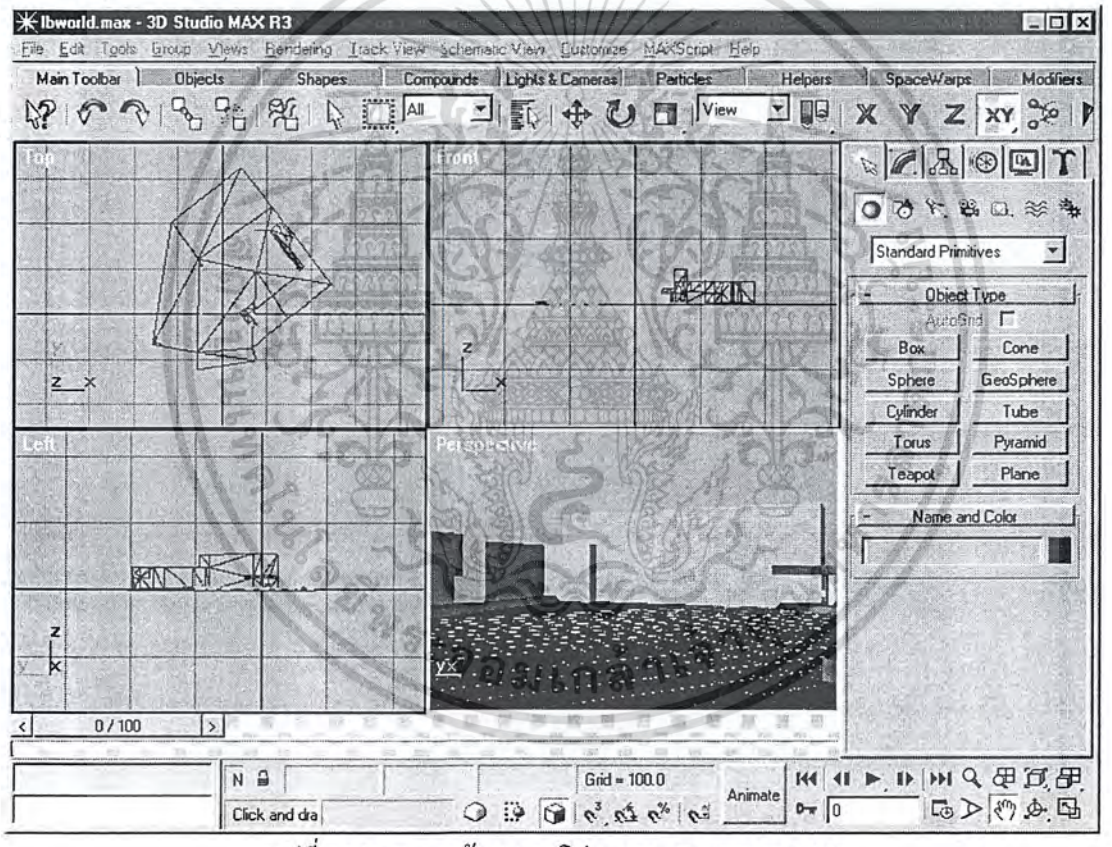
ปัญหานี้เกิดขึ้นเมื่อใช้ฟังก์ชัน Morfit_object_replace_3D_sequence_when_finished การทำงานของฟังก์ชันนี้ จะเปลี่ยนแอนิเมชันเป็นแอนิเมชันที่เรากำหนด เมื่อแอนิเมชันปัจจุบันเล่นจบ แต่เมื่อศึกษาจากคู่มือการใช้พบว่าในการเล่นแอนิเมชัน เอนจิ้นมอร์ฟิทจะทำอินเทอร์โพเลชัน (Interpolation) ระหว่างเฟรมเข้าด้วยกัน (เพราะฉะนั้นเมื่อเราใช้ฟังก์ชัน Morfit_3D_sequence_get_current_frame เพื่อตรวจสอบว่าขณะนี้แอนิเมชันเล่นไปถึงเฟรมเท่าไร ฟังก์ชันนี้จะส่งกลับค่า double กลับมา) ทางผู้วิจัยจึงสรุปว่าสาเหตุที่ทำให้เล่นแอนิเมชัน 2 รอบ เป็นแอนิเมชันที่เล่นไปถึงเป็นเฟรมสุดท้ายก่อนเปลี่ยนเป็นแอนิเมชันถัดไป เป็นเฟรมที่ A.BCDE... โดยที่ A เป็นเฟรมสุดท้ายของแอนิเมชันแรก โดยที่ 0.BCDE > 0.0 ถ้าเป็นอย่างที่กล่าวมอร์ฟิทจะอินเทอร์โพเลชันเฟรมสุดท้ายกับเฟรมแรกเข้าด้วยกัน ทำให้ดูเหมือนแอนิเมชันเล่นซ้ำอีกครั้งหนึ่งก่อนจะเปลี่ยนเป็นแอนิเมชันถัดไป

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 14
การสร้างโมเดล 3 มิติ เพื่อใช้กับมอร์ฟิท

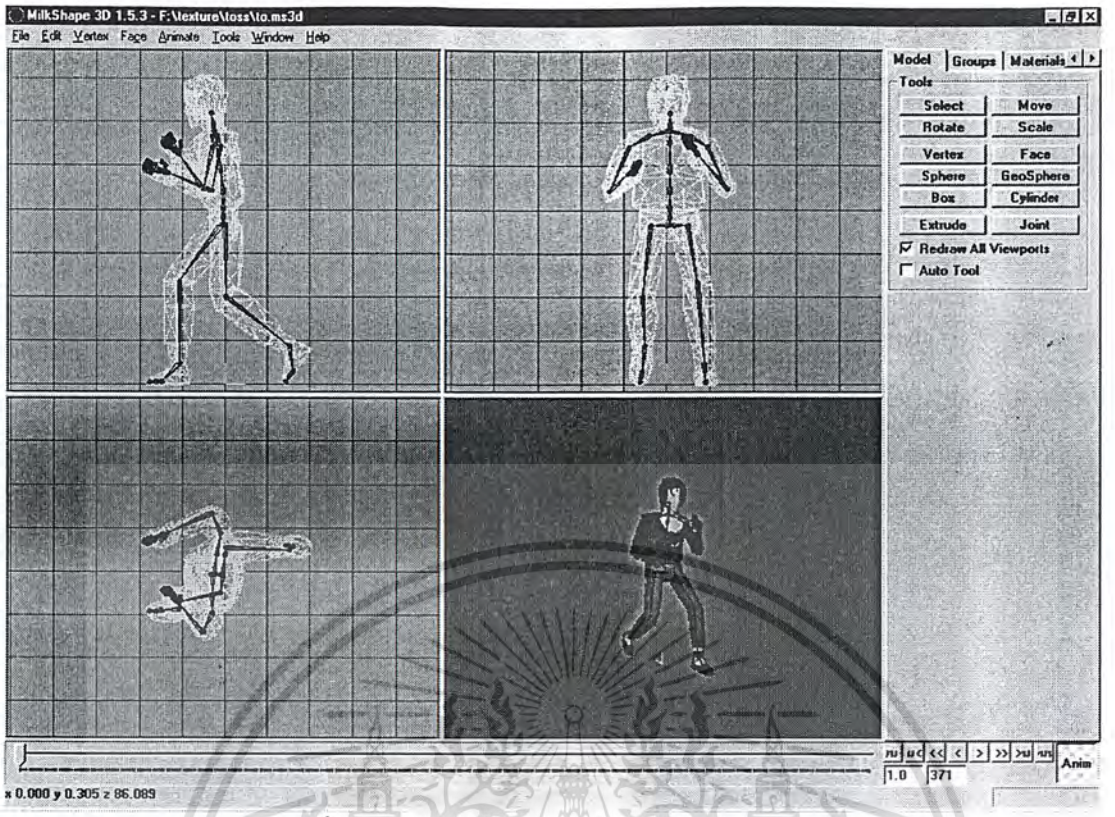
14.1 โปรแกรมที่ใช้ในการสร้างโมเดล 3 มิติ

สำหรับโครงงานนี้ ได้เลือกใช้โปรแกรม 3D Studio MAX และ Milkshape 3D โดยโปรแกรม 3D Studio MAX ใช้ในการสร้างฉาก และโปรแกรม Milkshape 3D ใช้ในการสร้างตัวละคร เนื่องจากทั้งสองโปรแกรมนี้มีจุดเด่นที่แตกต่างกัน โปรแกรม 3D Studio MAX จะสามารถจัดการกับโพลีกอนที่ซับซ้อนและมีจำนวนมากได้ดีจึงเหมาะกับการสร้างฉาก ส่วนโปรแกรม Milkshape 3D จะจัดการแอนิเมชันได้ดีและเอ็กซ์พอร์ตอิมพอร์ตไฟล์โมเดล 3 มิติได้หลายฟอร์แมต จึงเหมาะกับการทำโมเดลตัวละคร



รูปที่ 14-1 แสดงหน้าจอของโปรแกรม 3D Studio MAX

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

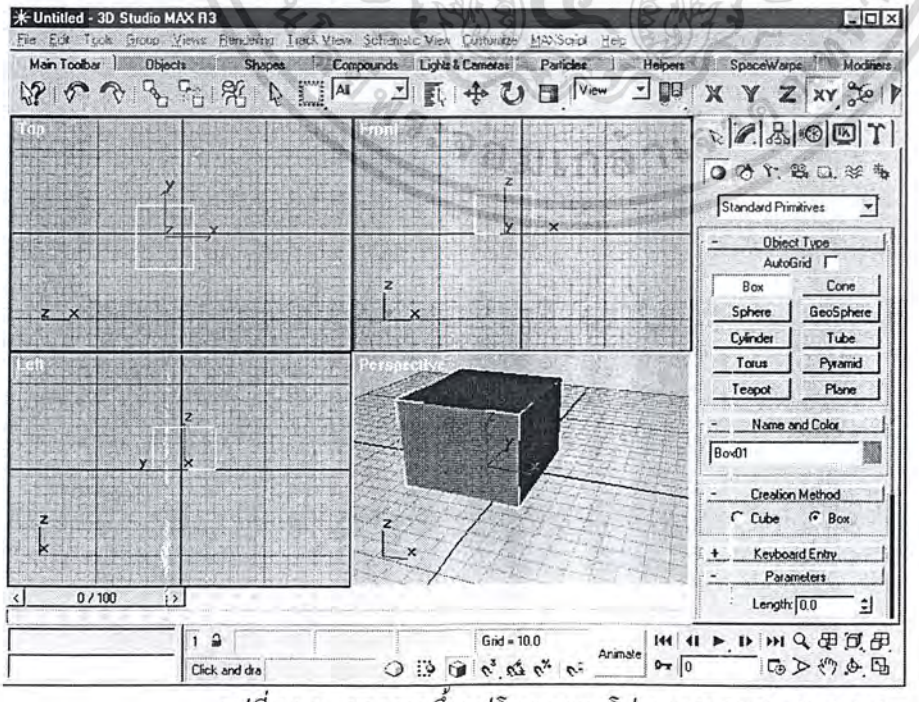


รูปที่ 14-2 แสดงหน้าจอของโปรแกรม Mlkshape 3D

14.2 โปรแกรม 3D Studio MAX

14.2.1 การขึ้นรูปโมเดล 3 มิติ

การขึ้นรูปโมเดลในโปรแกรม 3D Studio MAX จะเริ่มนำรูปทรงเรขาคณิตที่ใกล้เคียงกับโมเดลที่ต้องการนำมาติดกันหรือดัดแปลงให้ได้รูปทรงที่ต้องการ

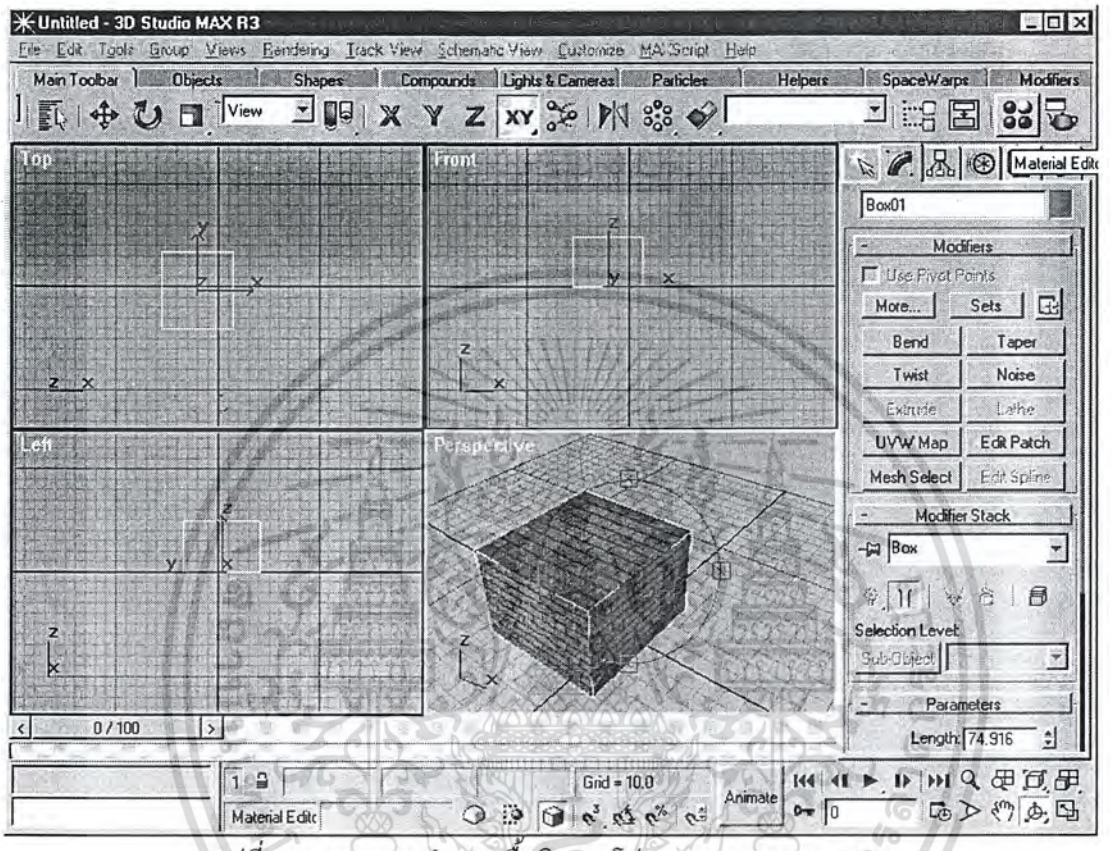


รูปที่ 14-3 แสดงการขึ้นรูปโมเดลของโปรแกรม 3D Studio MAX

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

14.2.2 การกำหนดพื้นผิว (Texture Mapping)

การกำหนดพื้นผิวในโปรแกรม 3D Studio Max จะเป็นการกำหนดพื้นผิวทั้งวัตถุ ถ้าหากเราต้องการให้วัตถุชิ้นหนึ่งมีหลายพื้นผิวจะทำได้ค่อนข้างลำบาก จุดเด่นของโปรแกรมคือสามารถเล่นเอฟเฟคบนพื้นผิวได้หลากหลาย ไม่จำเป็นต้องแต่งภาพก่อนที่จะนำมาแปะบนผิวโพลีกอน



รูปที่ 14-4 แสดงการกำหนดพื้นผิวของโปรแกรม 3D Studio MAX

14.2.3 การเอ็กซ์พอร์ตเพื่อนำไปใช้กับมอร์ฟิทเอนจิ้น

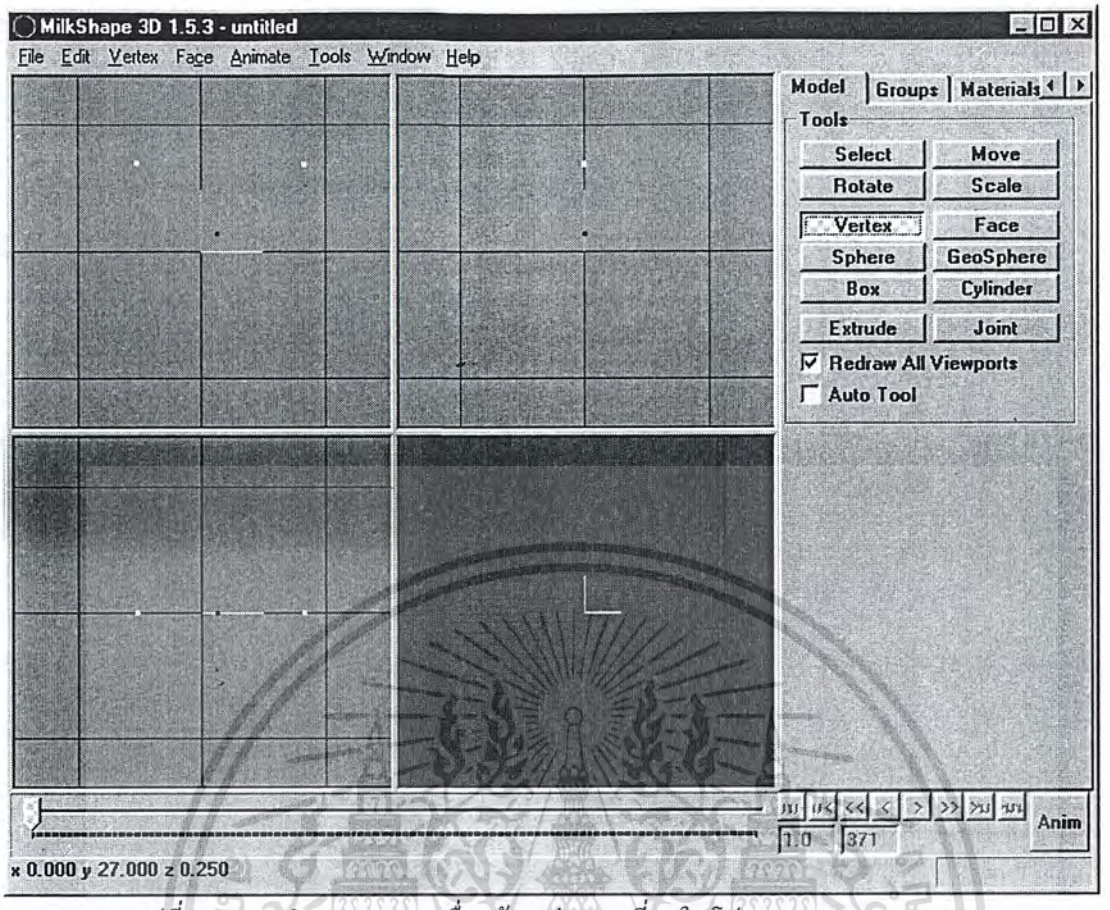
เมื่อเราทำการลง plugin ของมอร์ฟิทแล้วเราก็จะสามารถเอ็กพอร์ตโมเดล 3 มิติในรูปแบบไฟล์ .wld ได้ โดยไฟล์ .wld จะทำการบันทึกเพียงรูปทรงและพื้นผิวของโมเดลเท่านั้น ไม่มีแอนิเมชันการเคลื่อนไหว

14.3 โปรแกรม Milkshape 3D

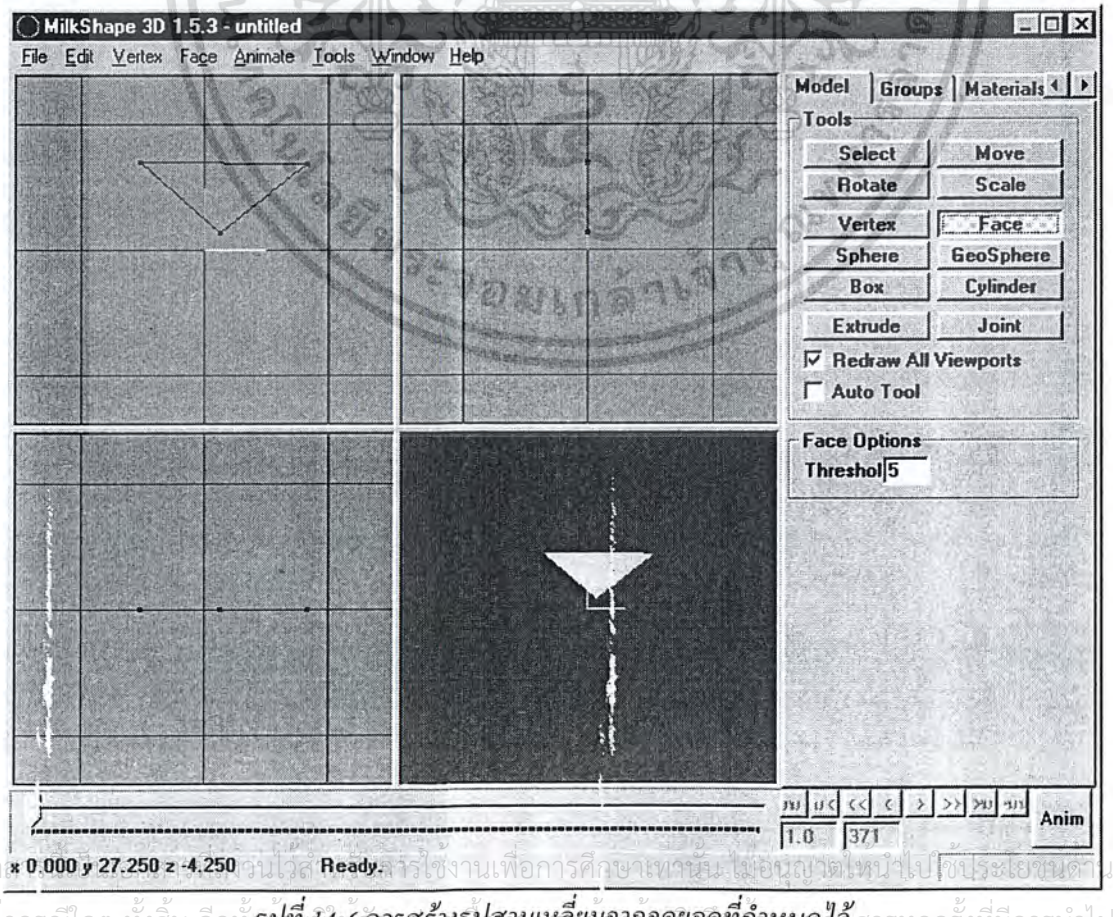
14.3.1 การขึ้นรูปโมเดล 3 มิติ

ในโปรแกรม Milkshape 3D สามารถขึ้นรูปโมเดลได้ 2 แบบ คือขึ้นรูปแบบอิสระโดยการวาดสามเหลี่ยมทีละอันแล้วประกอบขึ้นมาเป็นโมเดล หรือเริ่มต้นจากรูปทรงเรขาคณิตที่ใกล้เคียงโมเดลที่ต้องการก็ได้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

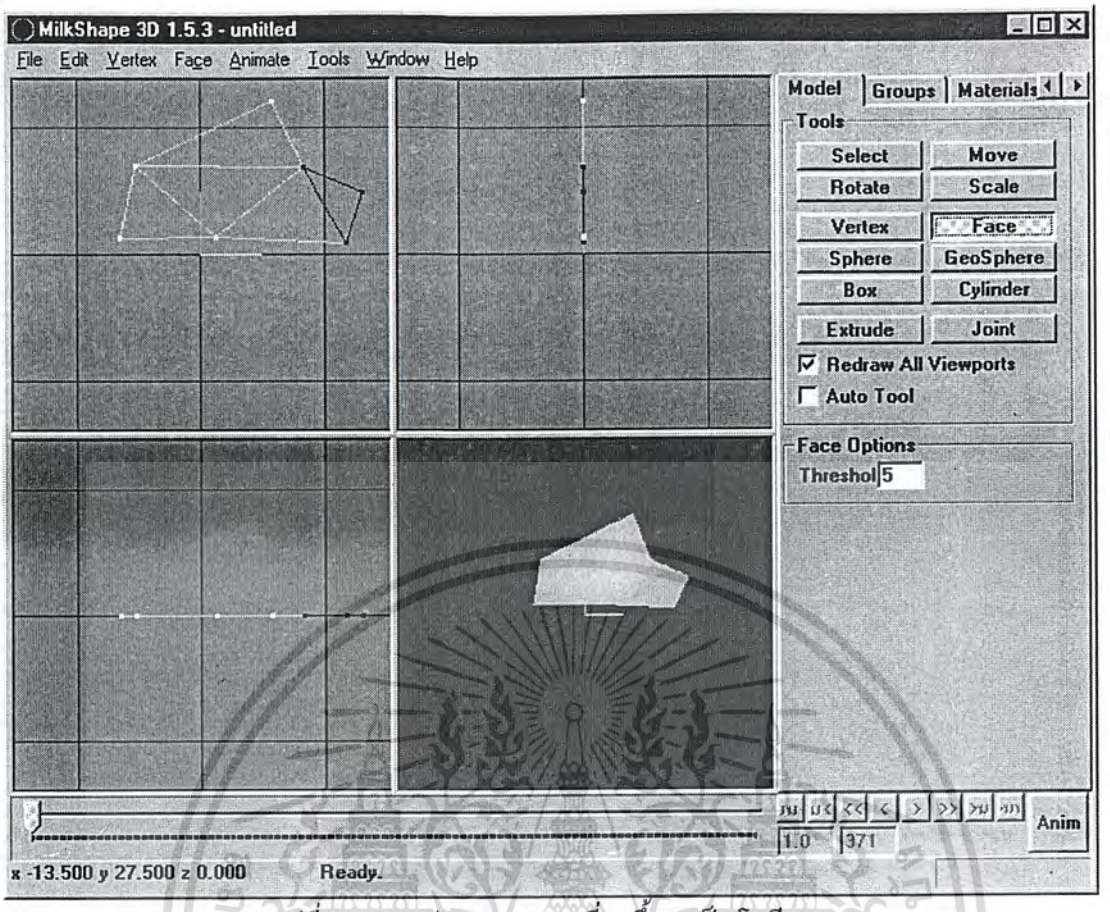


รูปที่ 14-5 การกำหนดจุดยอดเพื่อสร้างรูปสามเหลี่ยมในโปรแกรม Milkshape 3D



รูปที่ 14-6 การสร้างรูปสามเหลี่ยมจากจุดยอดที่กำหนดไว้

เอกสารนี้มีลิขสิทธิ์ สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งไม่อนุญาตให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเอกสารทุกครั้งที่มีการนำไปใช้

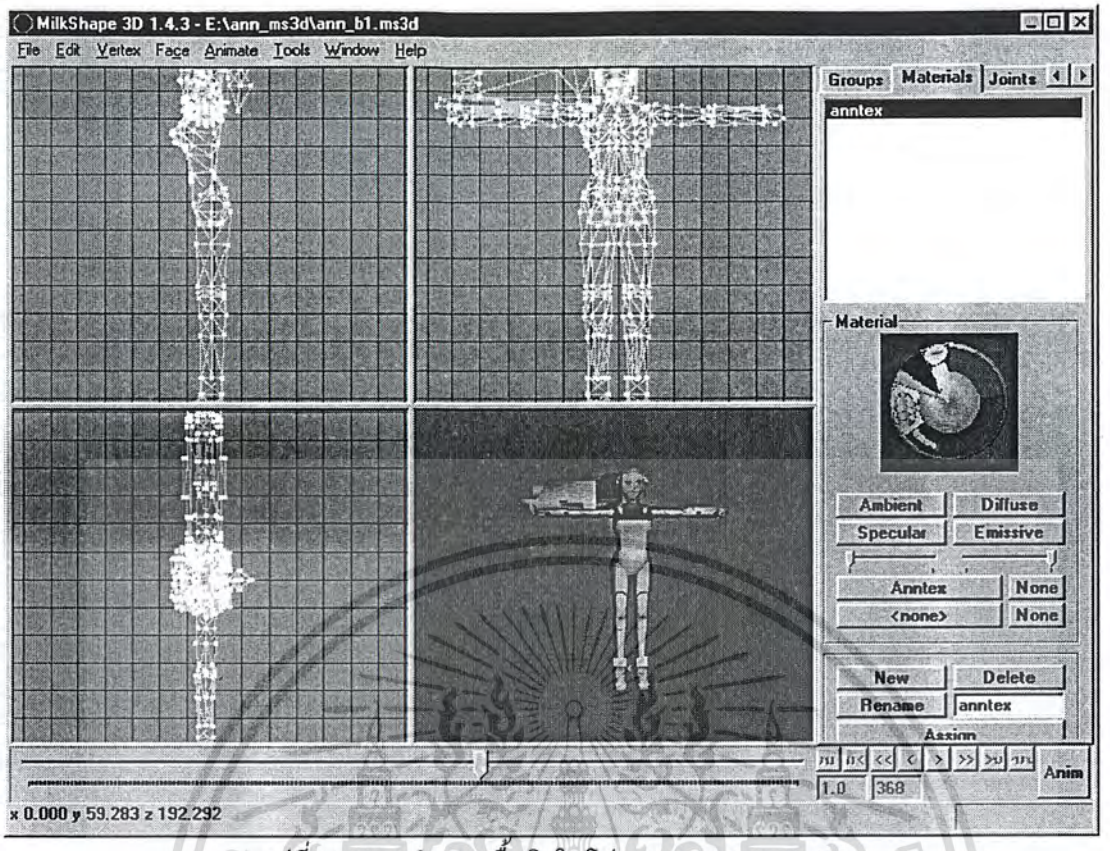


รูปที่ 14-7 การประกอบสามเหลี่ยมขึ้นมาเป็นโพลีกอน

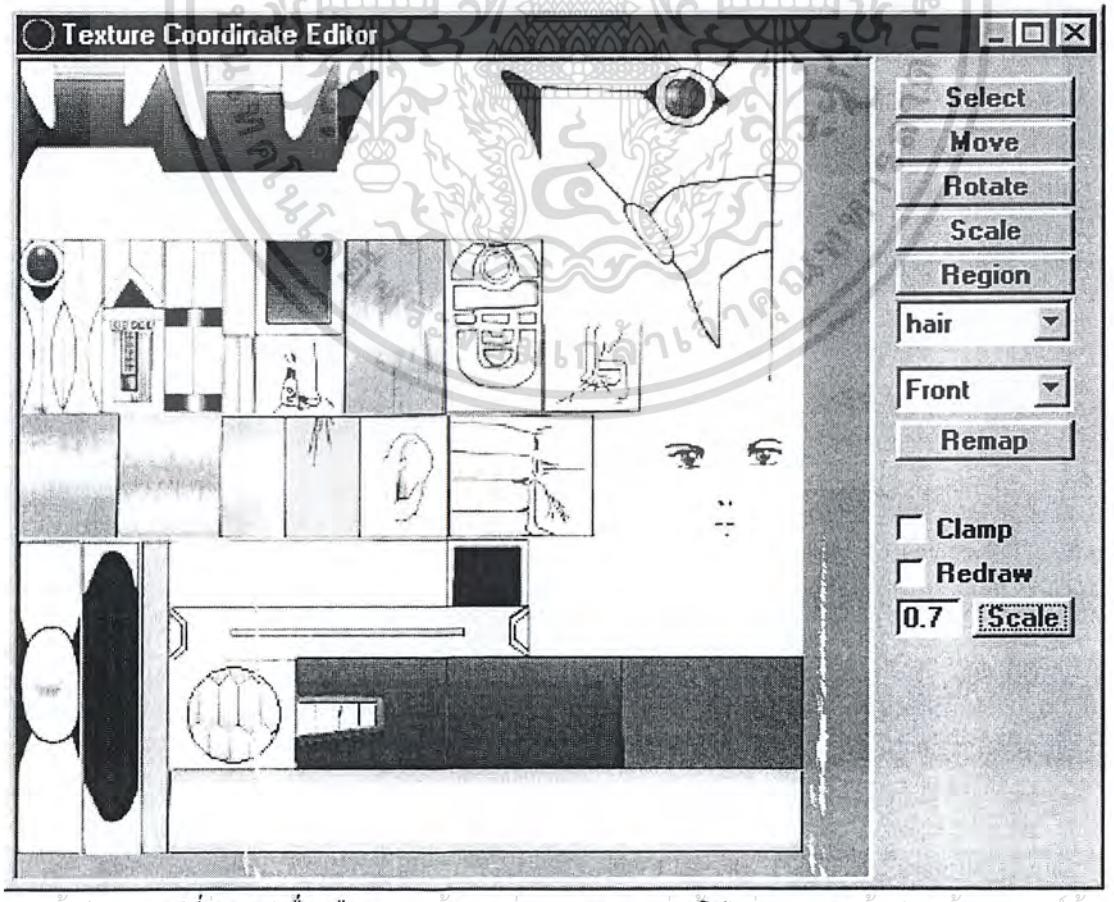
14.3.2 การกำหนดพื้นผิว (Texture Mapping)

การกำหนดพื้นผิวในโปรแกรม Milkshape 3D นั้น สามารถทำได้อิสระกว่า 3D Studio MAX เพราะแต่ละ face จะมีข้อมูลของ material และ coordinate เป็นของตัวเอง เราสามารถที่จะกำหนดให้โมเดลทั้งหมดของเรามี material เป็นภาพ bitmaps ภาพเดียวก็ได้ นอกจากนี้การจัด coordinate ของภาพที่อยู่บนพื้นผิววัตถุก็ทำได้อย่างอิสระมาก เนื่องจากมีเครื่องมือ Texture Coordinate Editor มาช่วยในการทำพื้นผิววัตถุ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้



รูปที่ 14-8 การกำหนดพื้นผิวในโปรแกรม Milkshape 3D

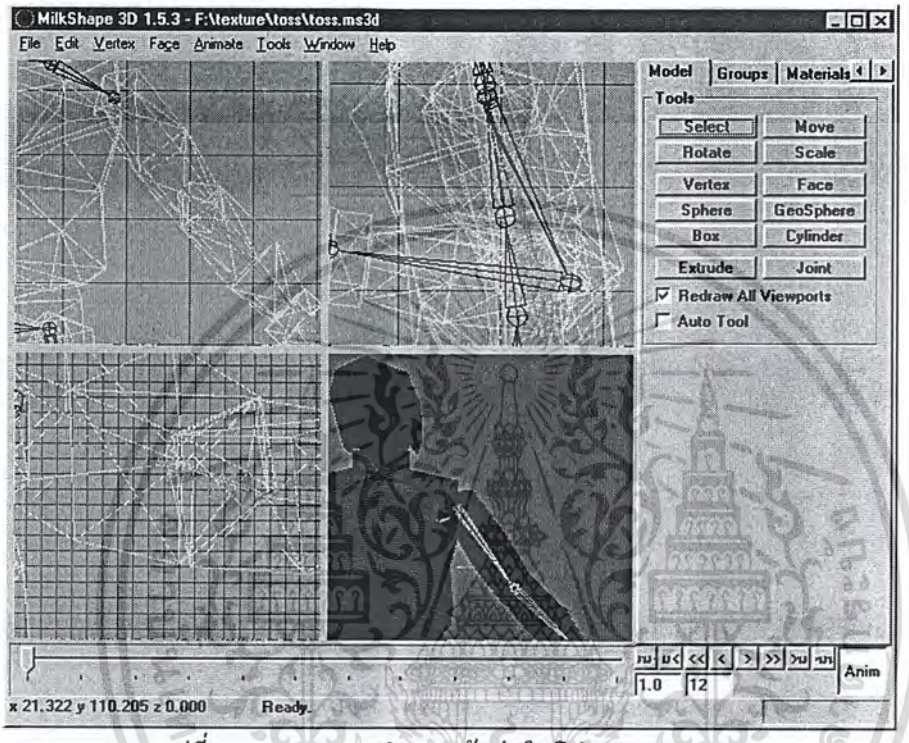


รูปที่ 14-9 เครื่องมือ Texture Coordinate Editor ของโปรแกรม Milkshape 3D

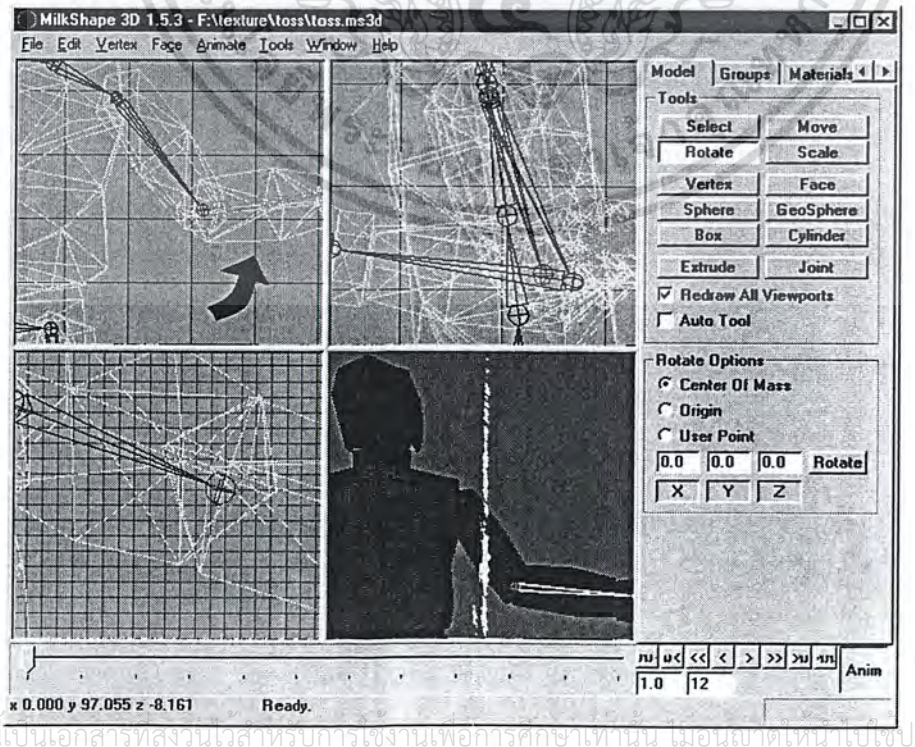
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

14.3.3 การกำหนดกระดูกและข้อต่อให้โมเดล 3 มิติ

หากต้องการให้โมเดล 3 มิติทำการเคลื่อนไหว เราจะต้องทำการบันทึกการเคลื่อนไหวเอาไว้ก่อน เมื่อเกิดเหตุการณ์ใดๆจึงแสดงท่าทางตามเงื่อนไข ในโปรแกรม Milkshape 3D หากต้องการกำหนดการเคลื่อนไหวให้กับโมเดล 3 มิติ จำเป็นจะต้องกำหนดแกนการเคลื่อนไหวหรือกระดูกให้โมเดลเสียก่อน โดยโมเดลจะทำการหมุนได้ตามข้อต่อของกระดูก รูปที่ 14-10 แสดงกระดูกของโมเดล 3 มิติ และรูปที่14-11 แสดงการหมุนข้อต่อ



รูปที่ 14-10 แสดงการกำหนดข้อต่อในโปรแกรม Milkshape 3D



รูปที่ 14-11 การหมุนข้อต่อ

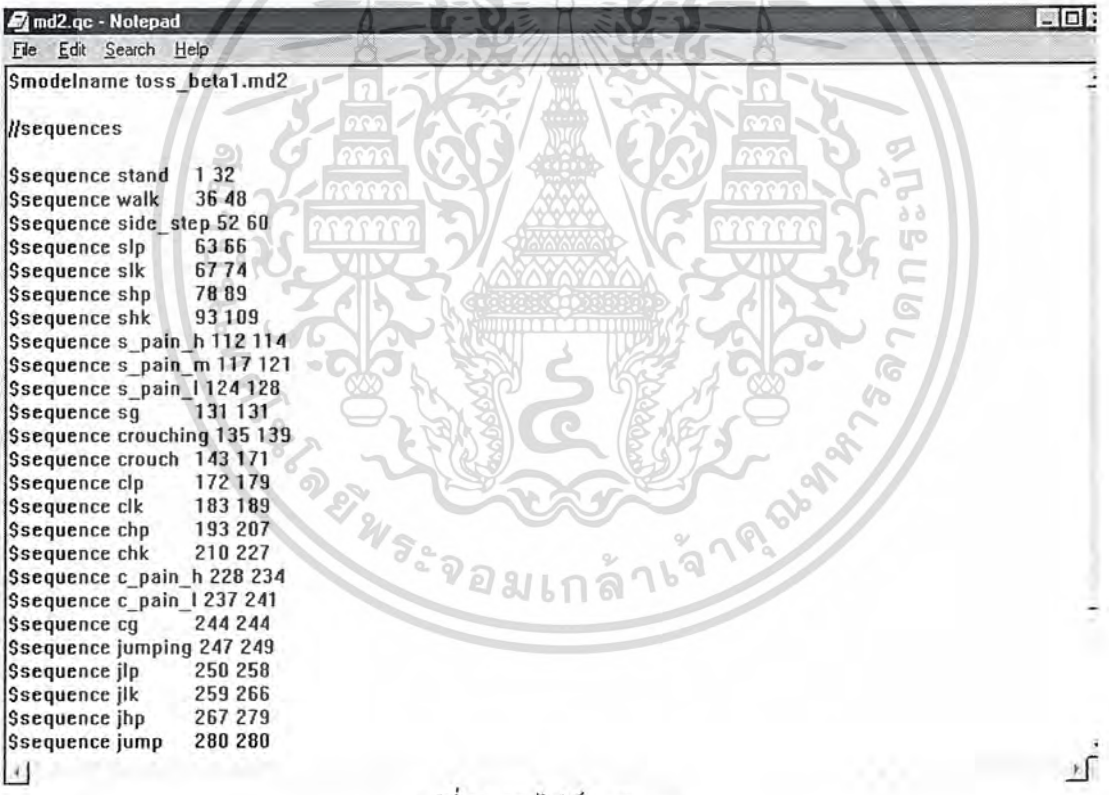
เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

14.3.4 การจัดการแอนิเมชันของโมเดล 3 มิติ

เมื่อเราทำการกำหนดกระดูกและข้อต่อแล้ว เราก็สามารถบันทึกการเคลื่อนไหวของโมเดล 3 มิติได้โดยทำการกำหนดท่าทางในเฟรมแรกและเฟรมสุดท้าย โปรแกรมจะทำการคำนวณเฟรมตรงกลางทั้งหมดให้

14.3.5 การเอ็กซ์พอร์ตเพื่อนำไปใช้กับมอร์ฟิทเอนจิ้น

มอร์ฟิทเอนจิ้นจะรู้จักไฟล์โมเดล 3 มิติที่มีนามสกุล .md2 โดยมีการบันทึกทั้งรูปร่างโมเดลและแอนิเมชันการเคลื่อนไหวทั้งหมดไว้ภายในไฟล์ โดยแต่ละแอนิเมชันจะมีชื่อแอนิเมชันให้ทำการเรียกใช้ได้ ในความเป็นจริงแล้ว รูปแบบไฟล์ .md2 จะทำการเรียงต่อแอนิเมชันทั้งหมดเป็นแอนิเมชันเดียว โดยแต่ละเฟรมจะมีชื่อเฟรมอยู่ เฟรมที่มีชื่อนำหน้าเหมือนกันจะถูกจัดให้อยู่ในกลุ่มเดียวกัน ถือเป็นแอนิเมชันหนึ่ง เช่น ท่ายืน มีชื่อเรียกคือ “stand” มีแอนิเมชันในช่วงเฟรมที่ 1- 10 ของไฟล์สกุล .md2 ก็จะมีชื่อเฟรมที่ 1-10 คือ “stand01” , “stand02” , “stand03” , …”stand10” หรือ ท่าวิ่ง มีชื่อเรียกคือ “run” มีแอนิเมชันในช่วงเฟรมที่ 11-22 ของไฟล์สกุล .md2 ก็จะมีชื่อเฟรมที่ 11-22 คือ “run01” , “run02” , …”run12”



รูปที่ 14-12 ไฟล์ md2 . qc

ในโปรแกรม Milkshpae 3D นั้นสามารถเอ็กพอร์ตโมเดล 3 มิติในรูปแบบไฟล์สกุล .md2 ได้ โดยจะต้องทำการเขียน QC ไฟล์ชื่อ md2.qc เพื่อกำหนดชื่อเฟรมแต่ละเฟรม ส่วนแอนิเมชันให้ทำการเรียงต่อกันทั้งหมดเป็นแอนิเมชันเดียว ตัวอย่างของไฟล์ md2.qc แสดงไว้ดังรูปที่ xxx

สำหรับความหมายของคำสั่งในไฟล์ md2.qc นั้น อธิบายได้ดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

$modelname <ชื่อไฟล์โมเดล> เป็นการกำหนดชื่อไฟล์ของโมเดลที่จะทำการเอ็กพอร์ต

$sequence <ชื่อแอนิเมชัน> <เฟรมเริ่มต้น> <เฟรมสุดท้าย> เป็นการกำหนดเฟรมให้แอนิเมชัน ว่ามีช่วงเฟรมที่เท่าไหร่บ้าง โดยหลังจากเอ็กพอร์ตแล้วจะมีชื่อเฟรมเป็น <ชื่อแอนิเมชัน><หมายเลขเฟรมที่>

//<ข้อความ> เป็นหมายเหตุ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

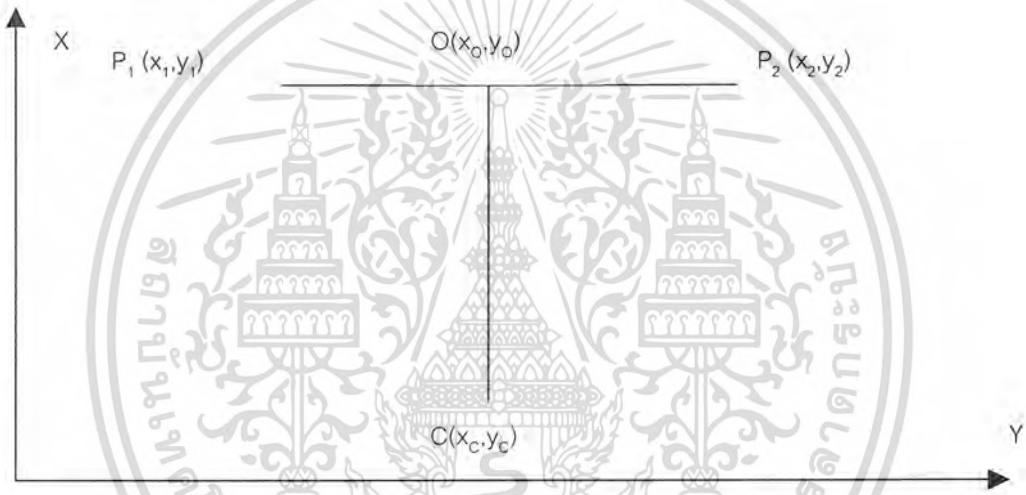
บทที่ 15

สิ่งที่มอร์ฟิทไม่ได้จัดเตรียมไว้ หรือ จัดเตรียมไว้แต่ไม่สามารถนำมาใช้ได้

15.1. การจัดตำแหน่งกล้องสำหรับเกมต่อสู้ 3 มิติ

เนื่องจากมอร์ฟิทสนับสนุนการเลื่อนตำแหน่งกล้องแบบติดตาม ซึ่งใช้ได้กับวัตถุเท่านั้น แต่ในเกมต่อสู้แบบ 1 ต่อ 1 กล้องจะต้องอยู่ระหว่างตัวละครทั้ง 2 และ ห่างจากจุดกึ่งกลางระหว่างตัวละครทั้ง 2 ในระดับหนึ่ง เพราะฉะนั้นทางผู้วิจัยจึงคิดอัลกอริทึมที่ใช้จัดการกับกล้องขึ้นเองโดยมีหลักดังนี้

15.1.1 วิธีเลื่อนตำแหน่ง



รูปที่ 15-1 ตำแหน่งของตัวละครทั้ง 2 กับ กล้อง

กำหนดจุด P_1 เป็นตำแหน่งของผู้เล่นที่อยู่ด้านซ้ายมือของกล้อง

กำหนดจุด P_2 เป็นตำแหน่งของผู้เล่นที่อยู่ด้านขวามือของกล้อง

กำหนดจุด O เป็นจุดกึ่งกลางระหว่าง P_1 และ P_2

$$x_O = (x_1 + x_2)/2 \qquad y_O = (y_1 + y_2)/2$$

กำหนดจุด C เป็นตำแหน่งของกล้อง

กำหนดเวกเตอร์ $U(A_1,B_1)$ เป็นเวกเตอร์ที่พุ่งจากจุด C ไปจุด O

กำหนดเวกเตอร์ $V(A_2,B_2)$ เป็นเวกเตอร์ที่พุ่งจากจุด O ไปจุด P_1

กำหนดเวกเตอร์ $u(a_1,b_1)$ เป็นยูนิตเวกเตอร์ของเวกเตอร์ U

$$a_1 = (x_O - x_C)/ |CO| \qquad b_1 = (y_O - y_C)/ |CO|$$

กำหนดเวกเตอร์ $v(a_2,b_2)$ เป็นยูนิตเวกเตอร์ของเวกเตอร์ V

$$a_2 = (x_1 - x_O)/ |OP_1| \qquad b_2 = (y_1 - y_O)/ |OP_1|$$

$|CO|$ เป็นระยะห่างระหว่างจุด C และ O ซึ่งกำหนดให้แปรผันตรงกับ $|OP_1|$ โดยสัมพันธ์กันดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

$|CO| = |OP_1|$ * ค่าแฟคเตอร์ที่กำหนด + ค่าคงที่ (ป้องกันกรณีที่ค่า $|OP_1|$ ต่ำมากๆ)

$|OP_1|$ เป็นระยะห่างระหว่างจุด O และ P_1 มีค่าเท่ากับ $|P_1 P_3| /2$

เราต้องการตำแหน่งของจุด C ซึ่งจุด C มีความสัมพันธ์ทางคณิตศาสตร์กับจุดอื่นๆ ดังนี้คือ

ถ้าไม่คำนึงถึงค่าแกน Z ค่า c_1 และ c_2 จะเท่ากับ 0 เราจะพบว่า

1) $u \times v = (0,0,1)$

2) $u \cdot v = \cos 90^\circ = 0$

จากความสัมพันธ์นี้เองทำให้สามารถหาจุด C มาได้ โดยแก้สมการดังนี้คือ

จาก $u \times v = (0,0,1) = |b_1 c_1| i - |a_1 c_1| j - |a_1 b_1| k$

$|b_2 c_2| \quad |a_2 c_2| \quad |a_2 b_2|$

$1 = a_1 b_2 - a_2 b_1$ ---------- (1)

จาก $u \cdot v = a_1 a_2 + b_1 b_2 + c_1 c_2 = \cos x^\circ = 0$ เนื่องจาก $x = 90^\circ$ จะได้

$0 = a_1 a_2 + b_1 b_2$ ---------- (2)

เนื่องจาก a_1 และ b_1 ซึ่งไม่สามารถหาค่าได้เพราะไม่รู้ค่า x_c และ y_c แต่ค่า a_2 และ b_2 เป็นค่าคงที่ เพราะเราทราบค่าที่ใช้ในการคำนวณทั้งหมด จึงเหลือตัวแปรเพียง 2 ตัว

จากสมการที่ (1)

$a_1 = (a_2 b_1 + 1)/ b_2$ ---------- (3)

$b_1 = (a_1 b_2 - 1)/ a_2$ ---------- (4)

บวก 1 เข้าไปที่สมการที่ (2)

$1 = a_1 a_2 + b_1 b_2 + 1$ ---------- (5)

สมการที่ (1) = (5)

$a_1 b_2 - a_2 b_1 = a_1 a_2 + b_1 b_2 + 1$

$a_1 b_2 - a_1 a_2 = b_1 b_2 + a_2 b_1 + 1$

เมื่อจัดสมการให้อยู่ในเทอมของ a_1 และ b_1 จะได้

$a_1 (b_2 - a_2) = b_1 b_2 + a_2 b_1 + 1$ ---------- (6)

$b_1 (a_2 + b_2) = a_1 b_2 - a_1 a_2 - 1$ ---------- (7)

แทนค่า a_1 จากสมการ (3) ในสมการที่ (6)

$((a_2 b_1 + 1)/ b_2) (b_2 - a_2) = b_1 b_2 + a_2 b_1 + 1$

$a_2 b_1 + 1 - (a_2^2 b_1 + a_2)/ b_2 = b_1 b_2 + a_2 b_1 + 1$

$b_1 b_2 = -(a_2^2 b_1 + a_2)/ b_2$

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

$b_1 b_2^2 = - a_2^2 b_1 - a_2$

$b_1(a_2^2 + b_2^2) = - a_2$

เนื่องจาก v เป็นยูนิตเวกเตอร์ เพราะฉะนั้น $|v| = sqrt(a_2^2 + b_2^2) = (a_2^2 + b_2^2) = 1$ ---------- (8)

$b_1 = - a_2$ ---------- (9)

$(y_O - y_C)/ |CO| = - (x_1 - x_O)/ |OP_1|$

$y_C = y_O + \{(x_1 - x_O) |CO| / |OP_1|\}$ ---------- (9.1)(Ans)

แทนค่า b_1 ในสมการที่ (9) ลงในสมการที่ (4) จะได้

$a_1 = (a_2 (-a_2)+ 1)/ b_2$

$a_1 = (1- a_2^2)/b_2$

เนื่องจากต้องนำสมการนี้ไปใช้ในการประมวลผลในคอมพิวเตอร์จึงจำเป็นต้องหลีกเลี่ยงการหาร เพราะอาจเกิดกรณีหารด้วย 0 ให้แทนค่า 1 ในสมการที่ (9) ลงในสมการจะได้

$a_1 = ((a_2^2 + b_2^2) - a_2^2)/b_2$

$a_1 = b_2^2/b_2 = b_2$ ---------- (10)

$(x_O - x_C)/ |CO| = (y_1 - y_O)/ |OP_1|$

$x_C = x_O + \{(y_1 - y_O) |CO| / |OP_1|\}$ ---------- (10.1)(Ans)

15.1.2 ทิศทางของกล้อง สามารถกำหนดได้โดยใช้ฟังก์ชันของมอร์ฟิทชื่อ Morfti_camera_point_at (DWORD camera_handle, double x, double y, double z) แล้วส่งค่าตำแหน่ง O เข้าไปให้

แนวคิดจาก ข้อที่ 15.1 ถูกนำไปอิมพลีเมนท์เป็นฟังก์ชัน

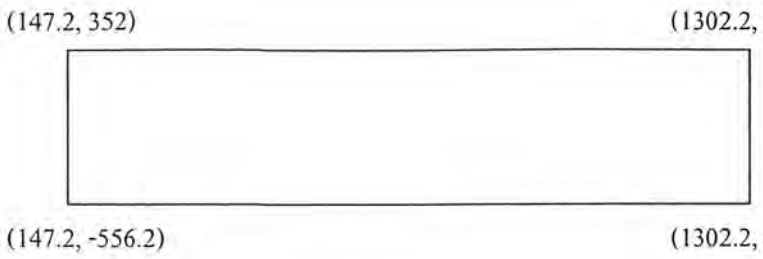
set_location_and_direction(double location_of_left_player[],double location_of_right_player [],double abs_of_vectorP1P2) ในวัตถุ camera

15.2 การตรวจสอบการชน

จากการวิเคราะห์ฟังก์ชันตรวจสอบการชนบนมอร์ฟิท พบว่า ฟังก์ชันของมอร์ฟิท ไม่สามารถทำงานได้อย่างสมบูรณ์ จากการประเมินหากจะตรวจสอบการชนกับวัตถุอื่นอย่างถูกต้อง โดยใช้เครื่องมือของมอร์ฟิทที่มีอยู่นั้น มีความเป็นไปต่ำมาก ที่การตรวจสอบอย่างละเอียดที่คิดขึ้นใหม่จะไม่ทำให้เฟรมเรทตกลงจนยอมรับไม่ได้ และ นอกจากนั้นมอร์ฟิทยังไม่ได้เปิดเผยซอร์สโค้ด จึงมีความเป็นไปได้ต่ำมากที่จะแก้ไข ฟังก์ชันของมอร์ฟิท (รายละเอียดอยู่ในบทที่ 13) ทางผู้วิจัยจึงออกแบบอัลกอริทึมที่เหมาะสมกับเกมของตนขึ้นเองโดยมีหลักการดังนี้

15.2.1 การตรวจสอบการชนกับวัตถุ 3 มิติในฉาก เนื่องจากไม่สามารถตรวจสอบวัตถุการชนกับวัตถุในฉากได้ ทางผู้วิจัยจึงจำกัดบริเวณที่ตัวละครจะสามารถเดินไปถึงได้ โดยในฉากของลาดกระบังที่ทำ มีขอบเขตบริเวณ ดังนี้

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

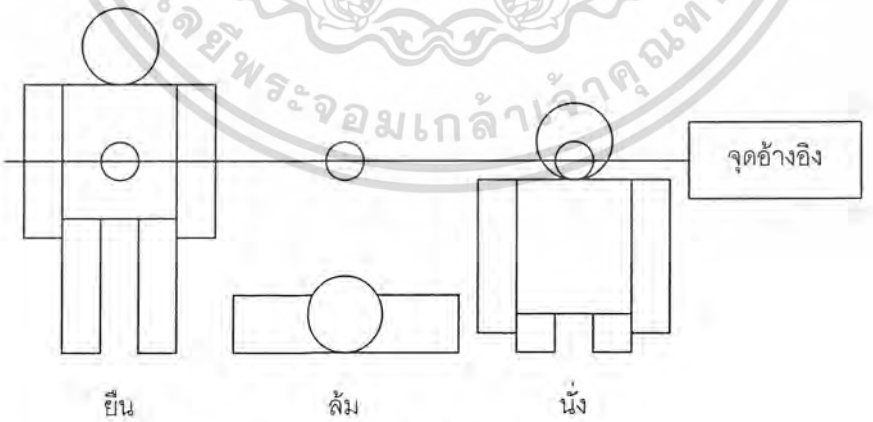


15.2.2 การตรวจสอบการชนระหว่างโมเดลเอ็มดี 2 เนื่องจากฟังก์ชันตรวจสอบการชนของมอร์ฟีทไม่สามารถตรวจสอบการชนได้หากวัตถุที่ขวางอยู่เป็นโมเดลเอ็มดี 2 ทางผู้วิจัยจึงกำหนดระยะใกล้ที่สุดที่ตัวละคร 2 ตัวจะเข้าใกล้กันได้ ระยะห่างนี้จะคำนวณโดยไม่สนใจค่าของแกน Z และ ไม่อนุญาติให้ตัวละครเคลื่อนไหวไปข้างหน้าหากระยะทางน้อย หรือ เท่ากับที่กำหนด วิธีนี้อาจจะเป็นวิธีที่หยาบ และ ดูเหมือนไม่น่าจะใช้ในทางปฏิบัติได้ แต่เนื่องจาก ในโปรแกรมจะกำหนดทิศทางของตัวละครให้หันหน้าไปทางฝ่ายตรงข้ามเสมอ จึงเสมือนว่าตัวละครทั้งสองอยู่บนเส้นเดียวกันตลอดไม่มีการเปลี่ยนแปลง

ส่วนการกระโดดเข้าหากันทางผู้วิจัยกำหนดให้ตรวจสอบผลต่างของระยะห่างตามแกน Z ก่อนโดยไม่สนใจค่าบนแกน X และ Y หากผลต่างความสูงมากกว่ากำหนดก็จะไม่จำเป็นต้องตรวจสอบระยะห่าง

แนวคิดจาก ข้อที่ 15.2.1 และ 15.2.2 ถูกนำไปอิมพลีเมนท์เป็นฟังก์ชัน move(double x_speed, double y_speed, double z_speed) ในวัตถุชื่อ character

15.2.3 การตรวจสอบการถูกโจมตี ถึงฟังก์ชันตรวจสอบการชนของมอร์ฟีทจะใช้กับโมเดลเอ็มดี 2 ได้สมบูรณ์ แต่เนื่องจากฟังก์ชันของมอร์ฟีทไม่ได้ตรวจสอบการชนระหว่างโพลีกอนอย่างแท้จริง



รูปที่ 15-2 ตำแหน่งจุดอ้างอิงเมื่อตัวละครเปลี่ยนท่าทาง

เพราะท่าโจมตีอย่างเช่นการเตะ หรือ ต่อย เป็นการยื่นบางส่วนของร่างกายออกไปเพื่อโจมตีฝ่ายตรงข้ามเท่านั้น ไม่ได้เป็นการเคลื่อนตัวผ่านไปทั้งตัว และ ตำแหน่งอ้างอิงก็ไม่ได้อยู่ติดกับตัวละคร สมมุติ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ว่าจุดอ้างอิงอยู่ที่ท้องเมื่อตัวละครยืนอยู่ แต่เมื่อตัวละครก้มลงจุดอ้างอิงก็ยังคงอยู่ตำแหน่งเดิม ทำให้เมื่อมองเทียบกับตัวละครจุดอ้างอิงจึงไปอยู่ที่หัวแทน เพราะฉะนั้นจึงไม่อาจจะกำหนดจุดอ้างอิงไปไว้ที่ส่วนต่างๆ ตามร่างกายที่ใช้โจมตีได้

จากสาเหตุดังกล่าวทางผู้วิจัยจึงคิดอัลกอลิทึ่มในการตรวจสอบการชนในกรณีนี้ขึ้นเอง โดยใช้หลักการตรวบสอบการอินเทอร์เซก (Intersect) ระหว่างปริมาตร โดยสร้างปริมาตรสมมุติขึ้นมาแล้วตรวจสอบว่าปริมาตรนี้อินเทอร์เซกกับบาวน์ดิงบ็อกซ์ของตัวละครฝ่ายตรงข้ามหรือเปล่า เมื่อเฟรมปัจจุบันมากกว่าเฟรมที่กำหนด ถ้ามีก็จะอนุมานว่าโจมตีถูกฝ่ายตรงข้าม โดยกำหนดคุณสมบัติของค่าโจมตีไว้ คุณสมบัติที่สำคัญมีดังนี้ คือ

ค่า radian เป็นค่า cos X° โดยที่ X° เป็นความกว้างของมุมโจมตี

ค่า range_distance_max เป็นค่าระยะห่างมากที่สุดที่ท่านั้นโจมตีถึง

ค่า range_in_axis_z_min เป็นตำแหน่งต่ำสุดที่ท่านั้นโจมตีโดน

ค่า range_in_axis_z_max เป็นตำแหน่งสูงสุดที่ท่านั้นโจมตีโดน

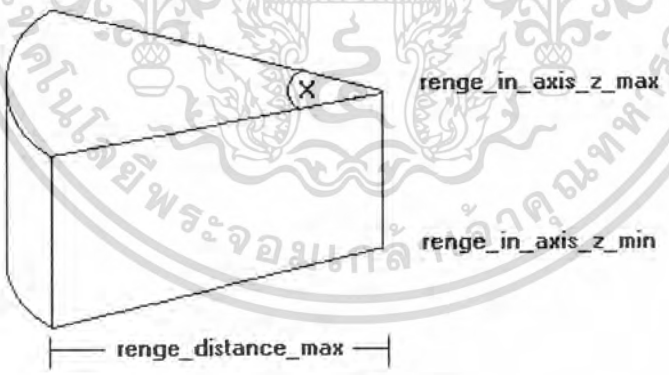
ค่า block_with_crouch_guard เป็นค่าที่บอกว่าท่าโจมตีนั้นสามารถการ์ดด้วยการนั่งการ์ดหรือไม่

ค่า block_with_stand_guard เป็นค่าที่บอกว่าท่าโจมตีนั้นสามารถการ์ดด้วยการยืนการ์ดหรือไม่

ค่า damage เป็นค่าพลังโจมตีของท่านั้น

ค่า attackable_frame จะบอกถึงเฟรมที่เริ่มตรวจสอบการชน

ค่า stagger_type จะบอกถึงลักษณะการเซของฝ่ายตรงข้าม หากฝ่ายตรงข้ามโดนการโจมตีและไม่ได้ป้องกัน



รูปที่ 15-3 ปริมาตรสมมุติที่ใช้ตรวจสอบการชน

อัลกอริทึมนี้ถูกนำไปอิมพลีเมนท์ในฟังก์ชัน receive_attack_data วัตถุ charactor ส่วนข้อมูลของท่าโจมตีต่างจะถูกกำหนดใน ฟังก์ชัน initialize ของวัตถุ charactor เช่นกัน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

15.3 ไดเรกท์อินพุต (Direct Input)

ไดเร็กท์อินพุต เป็นเอพีไอสำหรับอุปกรณ์อินพุตเช่น เมาส์ , คีย์บอร์ด , จอยสติ๊ก , อุปกรณ์ควบคุมเกมอื่นๆ รวมถึงอุปกรณ์ฟอร์ซ – ฟีดแบค(force – feedback)

การจัดการไดเร็กท์อินพุต ทำผ่านไดเร็กท์อินพุต ออบเจ็กท์ โดยอินเตอร์เฟส IDirectInput8

ไดเร็กท์อินพุต ออบเจ็กท์หนึ่งจะทำการเก็บข้อมูลของอุปกรณ์ควบคุมหนึ่งตัว สมมติว่าเราต้องการสร้างออบเจ็กท์สำหรับจอยสติ๊ก 2 ตัว เราก็ต้องทำการสร้างไดเร็กท์อินพุต ออบเจ็กท์ 2 อัน

ไดเร็กท์อินพุต ทำการติดต่อกับไดร์เวอร์ของอุปกรณ์ควบคุมโดยตรง จึงไม่สนใจแมสเซสของเมาส์และคีย์บอร์ดจากวินโดว์ และการปรับความเร็วหรือลักษณะต่างๆของเมาส์และคีย์บอร์ดในวินโดว์ก็ไม่มีผลกับไดเร็กท์อินพุต ยกเว้นการปรับค่าของจอยสติ๊กเท่านั้นที่ถูกนำมาใช้

ไดเร็กท์อินพุต ไม่ถือว่าการกดคีย์ค้างเป็นการกดซ้ำ ไดเร็กท์อินพุต จะจำสถานะของคีย์ว่ากดอยู่หรือไม่กดเท่านั้น เราสามารถเลือกได้ว่าจะใช้ข้อมูลสถานะแบบบัฟเฟอร์ (buffer) หรือแบบอิมมีเดียต (immediate) นอกจากนี้ปุ่มพิเศษพวก SHIFT,ALT,CTRL จะถูกมองว่าเป็นคีย์ปกติเหมือนปุ่มอักษรทั่วไป ทั้งนี้ข้อมูลในแต่ละคีย์นั้นจะแตกต่างกันไปตามเขตประเทศที่ตั้งไว้ในวินโดว์

เนื่องจากโครงงานนี้นำไดเร็กท์อินพุต มาใช้ในส่วนของจอยสติ๊กเท่านั้น จึงจะกล่าวเฉพาะวิธีใช้จอยสติ๊กเท่านั้น

<u>ขั้นตอนการใช้ไดเร็กท์อินพุต</u>

- <u>ขั้นตอนที่1 : การอีนิวมูเรตจอยสติ๊ก</u>

เริ่มแรกก็คือการอีนิวมูเรต (Enumerate)จอยสติ๊ก เพื่อตรวจสอบว่าเครื่องนั้นมีจอยสติ๊กอยู่หรือไม่ โดยทำการเรียกเมธอด IDiectInput8:EnumDevies ตามตัวอย่างด้านล่าง

```
//g_pDI เป็นพอยน์เตอร์ไปยัง IDirectInput8
g_pDI -> EnumDevices(DI8DEVCLASS_GAMECTRL , EnumJoysticksCallback ,
                                NULL , DIEDFL_ATTACHEDONLY)
```

ค่าคงที่ DI8DEVCLASS_GAMECTRL ซึ่งเป็นพารามิเตอร์แรก เป็นการบอกกลุ่มของอุปกรณ์ที่ทำการอีนิวมูเรต

EnumJoysticksCallback เป็นแอดเดรสสำหรับคอลแบคฟังก์ชันเพื่อใช้ในการคอลในแต่ละครั้งที่พบจอยสติ๊ก ซึ่งจะเกิดขึ้นเมื่อทำการ create devices

พารามิเตอร์ตัวที่สามเป็นเลข 32 บิตอะไรก็ได้ เพื่อกำหนดจำนวนของคอลแบคฟังก์ชัน ซึ่งในกรณีนี้ไม่มีการกำหนด

พารามิเตอร์ตัวสุดท้ายคือ DIEDFL_ATTACHEDONLY คือแฟลกจำกัดว่าทำการอีนิวมูเรตเฉพาะอุปกรณ์ที่ต่ออยู่กับคอมพิวเตอร์เท่านั้น แม้ว่าอุปกรณ์จะลงไดร์เวอร์ไว้แต่ไม่ได้ต่อกับพอร์ตใดๆ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

ของคอมพิวเตอร์ก็จะไม่ถูกอีนิวมูเรต ถ้าหากต้องการให้อีนิวมูเรตอุปกรณ์ทั้งหมดให้ใช้ DIEDFL_ALLDEVICES

- ขั้นตอนที่2 : การสร้าง DirectInput Joystick Device

ในขั้นตอนการอีนิวมูเรตจอยสติ๊กนั้น เราได้ทำการเรียกฟังก์ชัน EnumJoysticksCallback ซึ่งฟังก์ชันนี้ทำการสร้าง

จอยสติ๊กดีไวซ์ออบเจ็กท์ขึ้นมา ฟังก์ชัน EnumJoysticksCallback แสดงไว้ด้านล่าง

```
BOOL CALLBACK EnumJoysticksCallback( const DIDEVICEINSTANCE*
                                    PdidInstance , VOID *pcontext )
{
        HRESULT hr;
        // รับอินเตอร์เฟสเพื่อทำการอีนิวมูเรตจอยสติ๊ก
        hr = g_pDI -> CreateDevice( pdidInstance -> guidInstance ,
                                    &g_pJoystick , NULL)
        if ( FAILED (hr) )
                return DIENUM_CONTINUE;
        return DIENUM_STOP;
...
```

พารามิเตอร์ต่างๆของฟังก์ชัน EnumJoysticksCallback อธิบายได้ดังนี้

- พอยน์เตอร์ไปยังดีไวซ์อินสแตนซ์ ซึ่งถูกกำหนดไว้แล้วโดยไดเร็กท์อินพุตเมื่ออุปกรณ์ถูกอีนิวมูเรต
- พอยน์เตอร์ไปยังค่า 32 บิต ซึ่งเป็นพารามิเตอร์ของ IDirectInput8::EnumDevices ในกรณีนี้ค่านี้เป็น NULL

การทำงานของฟังก์ชันนี้คือการ create device เมธอด IDirectInput8::CreateDevice มีพารามิเตอร์ดังนี้ พารามิเตอร์แรกเป็นการอ้างอิงถึง globally unique identifier (GUID) สำหรับอินสแตนซ์ของอุปกรณ์ ในกรณีนี้ GUID ได้จากโครงสร้าง DIDEVICEINSTANCE ซึ่งกำหนดไว้แล้วโดยไดเร็กท์อินพุต

พารามิเตอร์ตัวที่สองเป็นแอดเดรสของตัวแปรซึ่งจะรับค่าอินเตอร์เฟสพอยน์เตอร์หากการคอลสำเร็จ

พารามิเตอร์ตัวที่สามเป็นแอดเดรสของตัวควบคุมออบเจ็กท์ของอินเตอร์เฟส Iunknown ในกรณีนี้ให้ค่าเป็น NULL

หากไม่สามารถ create device ได้ฟังก์ชันจะทำการรีเทิร์นแฟลก DIENUM_CONTINUE ซึ่งเป็นการบอกไดเร็กท์อินพุตให้ทำการอีนิวมูเรตต่อไปจนกว่าจะครบทุกอุปกรณ์ของคอมพิวเตอร์

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- ขั้นตอนที่ 3 : การจัดรูปแบบข้อมูลของจอยสติ๊ก

เมื่อเรามีพอยน์เตอร์ของไดเร็กท์อินพุตดีไวซ์แล้ว เราสามารถเรียกเมธอดของ IDirectInputDevice8 เพื่อทำการควบคุมอุปกรณ์ได้ ในขั้นตอนนี้เราจะทำการจัดรูปแบบข้อมูลของจอยสติ๊ก เพื่อให้ไดเร็กท์อินพุตจัดรูปแบบข้อมูลอินพุตได้อย่างถูกต้อง แสดงตัวอย่างได้ดังนี้

```
if ( FAILED ( hr = g_pJoystick -> SetDataFormat( &c_dfDIJoystick2) ) )
        return  hr;
```

สำหรับพารามิเตอร์มีเพียงตัวเดียวคือ c_dfDIJoystick2 ซึ่งถูกกำหนดไว้แล้วโดยไดเร็กท์อินพุต เป็นรูปแบบของข้อมูลซึ่งใช้โครงสร้าง DIJOYSTATE2 ในการรับข้อมูล

- ขั้นตอนที่ 4 : การจัดการ Joystick Behavior

ขั้นตอนต่อไปคือการตั้งค่า cooperative level ตามตัวอย่างไว้ด้านล่าง โดย g_pJoystick เป็นพอยน์เตอร์ของอินเตอร์เฟสของอุปกรณ์

```
if ( FAILED( hr = g_pJoystick -> SetCoorperativeLevel ( hDlg ,
                DISCL_EXCLUSIVE | DISCL_FOREGROUND ) ) )
        return  hr;
```

พารามิเตอร์แรกของ IDirectInputDevice8::SetCooperativeLevel คือวินโดว์แฮนเดิล

พารามิเตอร์ตัวสุดท้ายคือแฟลกที่ทำการตั้งค่า Cooperative Level ในที่นี้เราให้จอยสติ๊กรับอินพุตเมื่อแอพพลิเคชันของจอยสติ๊กทำงานโฟร์กราวน์อยู่เท่านั้น แล้วก็จะทำการติดต่อกับแอพพลิเคชันนั้นเท่านั้น จึงตั้งค่าแฟลกเป็น DISCL_EXCLUSIVE | DISCL_FOREGROUND

ขั้นตอนต่อไปคือการรับค่าลักษณะต่างๆของจอยสติ๊กที่ผู้ใช้ทำการตั้งไว้ในวินโดว์ ซึ่งรวมถึง มีแกนกี่แกน , จำนวนปุ่ม , และส่วนควบคุมพิเศษอื่นๆ โครงสร้าง DIDEVCAPS ใช้เก็บข้อมูลเหล่านี้ ก่อนอื่นทำการกำหนดค่าของ dwSize ซึ่งเป็นสมาชิกของ DIDEVCAPS จากนั้นจึงเรียก IDirectInputDevice8::GetCapabilities แสดงไว้ดังตัวอย่างข้างล่าง g_diDevCaps เป็น DIDEVCAPS

```
g_diDevCaps.dwsize = sizeof ( DIDEVCAPS );
if ( FAILED ( hr = g_pJoystick -> GetCapabilities ( &g_diDevCaps ) ) )
        return  hr;
```

ขั้นตอนต่อไปเป็นการหาจำนวนแกนของจอยสติ๊กรวมถึงระยะของแต่ละแกน โดยทำการเรียก IDirectInputDevice8::EnumObjects เราไม่ทำขั้นตอนนี้จอยสติ๊กจะใช้ค่าดีฟอลต์แทน

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
if ( FAILED (  hr = g_pJoystick -> EnumObjects( EnumAxesCallback ,
        (VOID*)hDlg , DIDFT_AXIS ) ) )
        return  hr;
```

พารามิเตอร์แรกของเมธอด EnumObjects คือ EnumAxesCallback เป็นแอดเดรสของคอลแบคฟังก์ชันซึ่งจะทำการอีนิวมูเรตแกนให้

พารามิเตอร์ตัวที่สองคือ เลข 32 บิตใดๆซึ่งให้กับคอลแบคฟังก์ชัน ในที่นี้เราให้แฮนเดิลของวินโดว์เพื่อสามารถอัพเดทค่าบนหน้าจอได้เมื่อพบแกนใดๆของจอยสติ๊ก

พารามิเตอร์ตัวสุดท้าย คือ DIDFT_AXIS เป็นการบอกคอลแบคฟังก์ชันให้ทำการอีนิวมูเรตเฉพาะแกนของจอยสติ๊กเท่านั้น

ในการกำหนดระยะของจอยสติ๊ก จะเป็นค่าต่ำสุดและสูงสุดของแต่ละแกน ถ้าเราตั้งค่า –1000 ถึง +1000 สำหรับแกน x หมายถึงเมื่อเราดันคันโยกไปทางซ้ายจนสุดจะรีเทิร์นค่า –1000 เมื่อดันคันโยกไปทางขวาสุดจะรีเทิร์นค่า +1000 และเมื่อไม่ดันคันโยกจะรีเทิร์นค่า 0

ตัวอย่างด้านล่างเป็นการตั้งระยะของแกนสำหรับจอยสติ๊ก ตัวแปร pdidoi เป็นแอดแดรสของโครงสร้าง DIDEVICEOBJECTINSTANCE ซึ่งเก็บข้อมูลของออบเจ็กท์ที่กำลังทำการอีนิวมูเรต

```
DIPROPRANGE  diprg;

diprg.diph.dwSize       =  sizeof ( DIPROPRANGE );
diprg.diph.dwHeaderSize  = sizeof ( DIPROPHEADER );
diprg.diph.dwHow        = DIPH_BYID;
diprg.diph.dwObj         = pdidoi -> dwType;
diprg.lMin              = -1000;
diprg.lMax              = +1000;
-
-
hr = g_pJoystick -> SetProperty ( DIPROP_RANGE , diprg.didh );
if ( FAILED ( hr ) )
        return DIENUM_STOP;
```

เราทำการตั้งระยะโดยกำหนดค่า lMir คือค่าต่ำสุด และ lMax คือค่าสูงสุด จากนั้นจึงทำการเรียก IDirectInputDevice8::SetProperty เพื่อทำการกำหนดค่าแกนของจอยสติ๊กตามที่เราต้องการ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

- ขั้นตอนที่ 5 : การเข้าถึงจอยสติ๊ก

หากเราต้องการข้อมูลอินพุตจากจอยสติ๊ก เราต้องเรียกเมธอด IDirectInputDevice8::Acquire เสียก่อน หลังจากนั้นจึงจะทำการดึงข้อมูลอินพุตจากจอยสติ๊กได้ และในโปรแกรมเราจะต้องมีการ acquire จอยสติ๊กใหม่ตลอดเวลา เผื่อว่าจอยสติ๊กหลุดจากแอพพลิเคชันด้วยสาเหตุใดสาเหตุหนึ่ง ซึ่งจะมีแฟลก DIERR_INPUTLOST แจ้งให้ทราบว่าแอพพลิเคชันสูญเสียจอยสติ๊กไป แสดงตัวอย่างได้ดังนี้

```
hr = g_pJoystick -> Poll ( );
if ( FAILED ( hr ))
{
        hr = g_pJoystick -> Acquire( );
        while( hr = = DIERR_INPUTLOST )
                hr = g_pJoystick -> Acquire( );
        return  S_OK;
}
```

- ขั้นตอนที่ 6 : การรับข้อมูลจากจอยสติ๊ก

ในกรณีของจอยสติ๊กนั้นแนะนำว่าให้ใช้ immediate data ดีกว่า buffered data เพราะเรามักต้องการการตอบสนองแบบทันทีทันใด เราทำการดึงข้อมูลจากจอยสติ๊กโดยเรียกเมธอด IDirectInputDevice8::GetDeviceState ซึ่งก่อนเราจะทำการดึงข้อมูลจากจอยสติ๊ก เราควรทำการเรียกเมธอด IDirectInputDevice8::Poll เพื่อตรวจสอบให้แน่ใจว่าแอพพลิเคชันสามารถรับข้อมูลจากจอยติ๊กได้ แสดงโค้ดตัวอย่างได้ดังนี้

```
HRESULT UpdateInputState( HWND hDlg)
{
        HRESULT         hr ;
        CHAR            strText[128] ;    // Device state text
        DIJOYSTATE2     js ;              // DirectInput joystick state
        CHAR*           str ;

        If ( NULL = = g_pJoystick)
                Return  S_OK ;
        // Poll the device to read the current state
        hr = g_pJoystick -> Poll ( ) ;
        if ( FAILED ( hr ) )
```

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

```
{
        //Attemt to reacquire joystick
}
hr = g_pJoystick -> GetDeviceState( sizeof ( DIJOYSTATE2 ) , &js );
if ( FAILED ( hr ) )
        return hr ;
```

พารามิเตอร์ตัวแรกคือขนาดของโครงสร้างที่ข้อมูลจะถูกรีเทิร์นกลับมา และพารามิเตอร์ตัวที่สองคือแอดเดรสของโครงสร้างนั้น ซึ่งโครงสร้าง DIJOYSTATE2 สนับสนุนจอยสติ๊กได้ถึง 6 แกน กับปุ่มอีก 128 ปุ่ม

สำหรับการตรวจสอบว่าปุ่มถูกกดหรือไม่ ให้ตรวจสอบบิตสูงสุดของแต่ละปุ่มที่รีเทิร์นกลับมา ถ้าเป็น 1 คือถูกกด

รายละเอียดเพิ่มเติมของไดเร็กท์อินพุตนั้น กรุณาอ่าน help ของ Microsoft DirectX 8.0 SDK

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 16

ความสัมพันธ์ระหว่างความต้องการของผู้ใช้ กับ บุคลากร

ในบทนี้เราจะกล่าวถึงความต้องการของผู้เล่นที่สำคัญ และ บุคคลากรที่จะสามารถทำให้เกมที่สร้างสามารถตอบสนองความต้องการของผู้เล่นได้

16.1 ความสนุก

การที่เกมจะทำให้ผู้เล่นรู้สึกสนุก และ อยากเล่นนั้น มีสาเหตุหลายประการ ไม่ว่าจะเป็นเนื้อเรื่องของเกม, รูปแบบ, รสนิยมของผู้เล่น หรือ แม้กระทั่งความยากง่าย เพราะฉะนั้นการสร้างเกมจึงไม่สามารถมองเป็นเพียงแค่การสร้างแอพพลิเคชัน แต่ยังต้องมองถึงส่วนประกอบอื่นๆ เช่น

16.1.1. เราอาจจะมองเกมเป็นนวนิยาย หรือ ภาพยนต์ ก็ได้ คณะผู้จัดทำอาจจะต้องการถ่ายทอดความคิด หรือ เรื่องราว ผ่านออกมาทางการดำเนินเรื่องในเกม เพราะฉะนั้นการที่จะทำให้ส่วนเนื้อเรื่องสมบูรณ์ คณะผู้จัดทำจะต้องมีผู้มีความสามารถด้านการวางโครงเรื่อง และ การแต่งเรื่อง ถ้าพิจารณาให้ดีจะพบว่าการอิมพลีเมนต์ส่วนนี้ให้สมบูรณ์ก็ไม่ต่างกับการผลิตภาพยนต์เลย แน่นอนว่าบุคคลกรที่จะทำงานส่วนนี้ให้สมบูรณ์ก็เหมือนกับบุคคลากรที่จำเป็นต้องมีในการผลิตภาพยนต์นั่นเอง

16.1.2. รูปแบบของเกม หรือ รูปแบบการเล่น ในปัจจุบันตั้งแต่เริ่มมีเกมเกิดขึ้นมาเราจะพบว่าเกมที่มีรูปแบบใหม่ก็จะมีโอกาสประสบความสำเร็จสูง ต่างจากเกมที่มีรูปแบบเดิมๆ ที่ทำให้ผู้เล่นรู้สึกจำเจน่าเบื่อ เพราะการที่เราจะสร้างเกมให้ประสบความสำเร็จนอกจากจะต้องมีความสมบูรณ์ในด้านๆ อื่นแล้ว ในเกมควรจะมีสิ่งใหม่ให้ผู้เล่นได้ทำ ซึ่งงานในส่วนนี้จะตกเป็นของ เกมคอนเซปดีไซน์ (Game Concept Design) และ เกมดีไซน์ (Game Design)

16.1.3. รสนิยมของผู้เล่น การที่จะสร้างเกมที่ประสบความสำเร็จ ไม่ใช่การสร้างเกมให้เสร็จ หรือ สร้างให้มีคุณสมบัติที่ดี แต่ต้องสร้างเกมที่จะมีผู้ซื้อ การที่เกมจะมีผู้ซื้อ ก็เพราะเกมเป็นที่น่าสนใจต่อผู้เล่นคนนั้นๆ ผู้สร้างต้องเข้าใจความต้องการของผู้เล่นส่วนใหญ่ เพราะฉะนั้นจึงต้องมีผู้วิเคราะห์ตลาดเพื่อที่จะรู้ความต้องการของผู้เล่น

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

16.2 วัตถุในเกมดูเหมือนจริง สวยงาม หรือ น่ารัก

วัตถุในเกม โดยทั่วไปอาจจะประกอบด้วยโมเดล 3 มิติ ซึ่งอาจจะสร้างโดยใช้โพลีกอน (หรือ เนิร์บ) และ เท็กซ์เจอร์ที่จะใช้แปะลงบนวัตถุ หรือ ภาพ 2 มิติ ซึ่งงานด้านการสร้างโมเดล 3 มิติ ก็จะเป็นหน้าที่ของทรีดีโมเดลเลอร์ (3D Modeler) แต่งานในสาขานี้ยังไม่ค่อยเป็นที่รู้จักในเมืองไทย และ มีคนที่มีความสามารถในด้านนี้อยู่ไม่มาก ส่วนในด้านการสร้างเท็กซ์เจอร์ และ ภาพ 2 มิติ ก็จะเป็นงานของกราฟฟิคดีไซน์เนอร์ (Graphic Designer) หรือ จิตรกรนั่นเอง แต่เป็นที่น่าเสียดายที่ในประเทศมักจะดูถูกผู้ประกอบอาชีพในสายงานนี้, ไม่ให้ความสำคัญเท่าที่ควร รวมถึงไม่มีการพัฒนาบุคลากรในสายอาชีพนี้อย่างจริงจัง นอกจากนี้ก่อนที่จะสร้างตัวละคร ก็ยังต้องมีการออกแบบตัวละครก่อน ไม่ว่าจะเป็นรูปร่าง, หน้าตา หรือ การแต่งตัว รวมถึงอุปนิสัยของตัวละครด้วย งานชิ้นนี้จะมีผู้ออกแบบตัวละคร (Character Design) เป็นผู้จัดการ

16.3 มีการเคลื่อนไหวของวัตถุในเกมสมจริง หรือ นุ่มนวล

งานในส่วนนี้จะเป็นหน้าที่ของแอนิเมเตอร์ (Animator) แต่เนื่องจากในปัจจุบันแอนิเมชันมีทั้งงานแอนิเมชัน 2 มิติ และ 3 มิติ นอกจากนั้นยังมีการประยุกต์งานแอนิเมชันทั้ง 2 มิติ และ 3 มิติ เข้าด้วยกัน เพราะฉะนั้นบางเกมที่เป็นเกม 2 มิติ จึงอาจจะต้องการ ทรีดีแอนิเมเตอร์ (3D Animator) ด้วยก็ได้

การทำแอนิเมชัน 3 มิติ นั้นต้องใช้เวลาสูงหากไม่มีเครื่องมือที่ดีเข้ามาสนับสนุน โดยเฉพาะอย่างยิ่งการทำแอนิเมชันของคน ไม่ว่าจะเป็นการเดิน, วิ่ง, ท่าทางต่างๆ และ การเปลี่ยนแปลงสีหน้า ซึ่งในปัจจุบันการทำแอนิเมชันประเภทนี้มีเครื่องมือที่เรียกว่าโมชันแคพเจอร์ (Motion Capture) เข้ามาช่วย แม้ว่าโมชันแคพเจอร์จะมีราคาแพง แต่มันก็ช่วยประหยัดต้นทุนในการพัฒนาในระยะยาว และ ให้การเคลื่อนไหวสมจริง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 17
คุณสมบัติของเกมเอนจิ้นที่ดี

17.1 ประสิทธิภาพของเกมเอนจิ้น

ถ้ามองในด้านประสิทธิภาพการแสดงผล เมื่อพิจารณาหลักการทำแอนิเมชันในปัจจุบัน การที่จะทำให้ผู้ดูรู้สึกว่าการเคลื่อนไหวนั้นนุ่มนวล, สมจริง และ ไม่ดูตะกุกตะกัก แอนิเมชันจะต้องมีเฟรมเรทประมาณ 25-30 เฟรมต่อวินาที แต่ในการแสดงผลจริงๆ คอมพิวเตอร์ไม่สามารถประมวลผลให้ได้เฟรมสูงกว่า 25 เฟรมต่อวินาทีตลอดเวลา บางครั้งเฟรมเรทก็ตกลงมาต่ำกว่า 25 เฟรมต่อวินาที การที่เฟรมเรทเปลี่ยนแปลงไปจากค่าเฉลี่ยเฟรมเรทของระบบนั้นๆ ก็จะทำให้การแสดงผลช่วงนั้นดูตะกุกตะกัก

เพราะฉะนั้นจากที่ได้กล่าวข้างต้นคุณลักษณะของเกมเอนจิ้นที่ดีจะต้องสามารถแสดงผลให้ได้มากกว่า 25 เฟรมเป็นอย่างต่ำ และ ต้องมีอัตราเปลี่ยนแปลงของเฟรมเรทเมื่อเทียบกับค่าเฉลี่ย หรือ อัตราการกระจัด อยู่ในเกณฑ์ต่ำด้วย

17.2 เวลาที่ใช้ในการพัฒนาเกมโดยใช้เกมเอนจิ้นนั้นๆ

เกมเอนจิ้นที่ดีจะต้องช่วยลดเวลาที่ใช้ในการพัฒนาลงโดยมีประสิทธิภาพอยู่ในเกณฑ์ที่ยอมรับได้ ตัวอย่างเช่น

1. เอนจิ้นที่ดีต้องลดขั้นตอนการโปรแกรมมิงที่ไม่จำเป็น หรือ ลดขั้นตอนให้มากที่สุดเท่าที่จะลดได้เพื่อที่จะลดเวลาที่ใช้ในการโปรแกรมลง

2. มีแอพพลิเคชันที่ช่วยในการสร้างวัตถุ หรือ โลก 3 มิติ มากับเกมเอนจิ้นด้วย ถ้าไม่มี ก็ควรจะสนับสนุนในการเรียกใช้ไฟล์วัตถุ 3 มิติ ของโปรแกรมขึ้นโมเดล 3 มิติ อื่นๆ (เช่น 3D Studio Max, Light Wave หรือ Maya) เพื่อที่ผู้พัฒนาจะได้ไม่เสียเวลาสร้างส่วนที่จะใช้โหลดไฟล์วัตถุ 3 มิติ นั้นๆ รวมถึงข้อมูลแอนิเมชันในวัตถุนั้นๆ ด้วย

17.3 ง่ายต่อการทำความเข้าใจ

คำสั่ง หรือ ฟังก์ชันของเกมเอนจิ้นควรอยู่ในรูปภาษาพูดที่ง่ายต่อการเข้าใจ เช่น ในมอร์ฟิทฟังก์ชันที่ใช้โหลดเสียงมีชื่อ DWORD Morfit_sound_load(char *file_name, DWORD entity_handle, double distance_reach); โดย entity_handle เป็นค่าแฮนเดิลของวัตถุ 3 มิติที่เป็นแหล่งกำเนิดเสียง เมื่อกำหนดค่านี้ให้เป็นค่าแฮนเดิลของวัตถุโดยไม่ปล่อยให้เป็น NULL ก็จะสามารถ เล่นเสียงโดยมีคุณสมบัติเป็นเสียง 3 มิติ ได้ มอร์ฟิทจะคำนวณ ความดังของเสียงจากระยะห่างของแหล่งกำเนิดเสียงให้เอง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

จากที่ยกตัวอย่างจะเห็นว่าฟังก์ชันนี้มีชื่อ ที่เข้าใจง่าย นอกจากนี้ยังลดขั้นตอนที่ไม่จำเป็นไปมากเมื่อเทียบกับไดเรกท์ซาว์นทรีดี (DirectSound3D) การที่รูปแบบ และ คำสั่งของเอนจิ้นง่ายต่อการเข้าใจนอกจากจะทำให้เกิดผลดีในเรื่องการลดเวลาที่ใช้ศึกษา และ ลดเวลาที่ใช้พัฒนาเกมแล้ว ยังทำให้โค้ดมีขนาดเล็ก และ ง่ายต่อการหา และ แก้ไขข้อผิดพลาด

17.4 มีฝ่ายสนับสนุนผู้สร้างเกม

เกมเอนจิ้นที่ดีต้องมีการสนับสนุนการใช้ที่ดีด้วยไม่ว่าจะเป็นการสนับสนุนในเรื่องการให้ความช่วยเหลือผ่านเวป จดหมายอิเล็กทรอนิก หรือ คู่มือการใช้งานที่สมบูรณ์ ต่อให้เป็นเกมเอนจิ้นที่ดีเลิศขนาดไหน หากไม่มีคนที่สามารถนำเอนจิ้นนั้นไปใช้ได้ เพราะถ้าผู้ใช้ไม่รู้วิธีการใช้ก็เปล่าประโยชน์

17.5 สนับสนุนการจัดการด้วยตัวเอง หากเห็นว่าสิ่งที่เอนจิ้นจัดเตรียมให้มีความไม่เหมาะสม

แต่ในความเป็นจริงแทบจะไม่มีเอนจิ้นใดเลยที่อนุญาติให้ผู้ใช้ทำได้ เพราะการกระทำเช่นนี้ ย่อมหมายความว่าผู้พัฒนาเอนจิ้นต้องเปิดเผยซอร์สโค้ด บางส่วนหรือทั้งหมด เพราะฉะนั้นเกนเอนจิ้นโดยทั่วไปจึงอนุญาติให้ผู้ใช้จัดการสิ่งที่ไม่เหมาะสมด้วยตนเองได้เพียงบางส่วนเท่านั้น

17.6 สนับสนุนเทคโนโลยีการแสดงผล 3 มิติ ที่หลากหลาย

เกมเอนจิ้นต้องสามารถเรียกใช้ความสามารถพิเศษของการ์ดแสดงผลแต่ละชนิดได้ เอนจิ้นจะไม่ช่วยให้การสร้างเกมที่มีกราฟิคสวยงามง่ายขึ้น หากไม่สนับสนุนการเรียกใช้ความสามารถเฉพาะของการ์ดแสดงผล เพราะความสามารถเฉพาะของ การ์ด 3 มิติหลายอย่าง ที่มีส่วนทำให้ภาพที่ออกมาดูดี เช่น บัมแมบบิงแบบต่างๆ, แอนติเอเลียสแบบเต็มหน้าจอ, เอฟเฟคภาพยนต์ต่างๆ (เช่น ระยะชัดลึก และ โมชันเบลอ)

17.7 สนับสนุนการแสดงผล และ จัดการกับพื้นผิวเนิร์บ (Nurb)

ในปัจจุบันการมองวัตถุ 3 มิติ เป็นโพลีกอน นั้นมีข้อจำกัดอยู่มาก เพราะเมื่อมองวัตถุเป็นโพลีกอน การทำแอนิเมชันจะต้องกำหนดข้อต่อ และ กระดูกให้ เพื่อที่จะมาขยับ และ ทำแอนิเมชัน แต่ถ้ามองในโลกจริงๆ ก็จะพบว่ามีวัตถุหลายอย่างที่ไม่มีกระดูก และ ข้อต่อ แต่งอขยับได้เหมือนมีข้อต่อนับไม่ถ้วน ตัวอย่างเช่น ผม, กระดาษ, ผ้า และ ผิวหนัง แต่ในความเป็นจริงแล้วของทุกอย่างในโลกสามารถมองในรูปของ เนิร์บได้ ในขณะที่พบว่าโพลีกอนสามารถแทนวัตถุที่เป็นของแข็งเท่านั้น เพราะฉะนั้นเอนจิ้นในอนาคตที่ดีจึงควรจะสนับสนุนวัตถุที่สร้างขึ้นมาโดยใช้พื้นผิวเนิร์บ

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

17.8 จัดการกับวัตถุบางส่วน และ เอฟเฟคต่างๆ โดยการซิมิวเลชัน (Simulation)

เหตุการบางอย่างที่เกิดขึ้นในเกมหากเป็นการจัดเตรียมมาก่อน ไม่ใช่การซิมิวเลชัน สิ่งแสดงผลออกมาจะดูไม่สมจริง อย่างเช่น การเคลื่อนไหวของเส้นผม หรือ เสื้อผ้า, การกระโดด, การตกลงตามแรงโน้มถ่วงของวัตถุ, คลื่นในทะเล และ การลุกไหม้ของเปลวไฟ การทำแอนิเมชันวัตถุ หรือ สิ่งเหล่านี้โดยสร้างข้อมูลแอนิเมชันเตรียมไว้ก่อน จะทำให้ผู้เล่นรู้สึกถึงความไม่เหมือนจริง เนื่องจากการเคลื่อนไหวของวัตถุ จะวนลูปแสดงซ้ำไปเรื่อยๆ ตามข้อมูลแอนิเมชันที่มี หรือแม้แต่การเคลื่อนไหวของเสื้อผ้า หรือ เส้นผมของตัวละคร

ตัวอย่างหากเราจะทำโลก 3 มิติ ที่มีลมพัดมาเป็นระยะๆ นั้นก็หมายความว่าวัตถุเบา อย่างเช่น เสื้อผ้า และ เส้นผมต้องปลิวไปตามลมด้วย หมายความว่าในโมเดล 3 มิติ ที่เราสร้างจะต้องมีข้อมูลบอกเราว่า ตรงส่วนไหนของโมเดล ที่จะต้องมีการเคลื่อนไหวสัมพันธ์กับปรากฏการณ์รอบข้าง และ สัมพันธ์กันอย่างไร ในส่วนการทำแอนิเมชันนั้น ผู้พัฒนาก็ควรจะต้องทำแค่การเคลื่อนไหวหลักของตัวละคร ในส่วนวัตถุที่ติดกับตัวละครที่จะต้องมีการเลื่อนไหวสัมพันธ์กับสิ่งแวดล้อมนั้น เอนจิ้นควรจะต้องเป็นฝ่ายจัดการเอง

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บทที่ 18

แนวทางการพัฒนางานวิจัย, ข้อแนะนำ และ บทสรุป

18.1 สร้างเกม 3 มิติ

ถ้าจะพัฒนาเกม 3 มิติ ต่อทางผู้วิจัยแนะนำ ให้พัฒนาโดยใช้เกมเอนจิ้น แต่แนะนำให้ใช้เกมเอนจิ้นอื่นนอกจากมอร์ฟิท ตัวอย่างเช่น LithTech แม้ LithTech จะไม่เป็นฟรีแวร์ แต่ก็มีราคาเพียง 399 เหรียญสหรัฐ ซึ่งราคานี้รวมค่าฝ่ายสนับสนุนแล้ว หากทางสถาบันสนใจน่าจะติดต่อสั่งซื้อกับทางบริษัทแบบเพื่อการศึกษาได้ในราคาพิเศษ โดยติดต่อได้ที่ sales@lithtech.com สาเหตุที่แนะนำให้ใช้เอนจิ้นตัวอื่น เนื่องมอร์ฟิทมีข้อจำกัดอยู่มากตามที่ได้กล่าวไว้ในบทที่ 13

ในเรื่องการพัฒนาควรจะแบ่งทีมพัฒนาเป็น 2 ส่วน คือ ส่วนที่ทำงานด้านโปรแกรม กับ ส่วนที่ทำหน้าที่สร้างรีซอร์ส ไม่ว่าจะเป็นวัตถุ ที่ใช้ในเกม ทั้ง 3 มิติ และ 2 มิติ เพื่อที่จะได้เกมที่มีความสมบูรณ์ และใกล้เคียงกับเกมที่วางจำหน่ายในตลาดมากขึ้น หรือ เพลง และ เสียงประกอบในเกม

18.2 สร้างเอนจิ้นเกม 3 มิติ

แนะนำให้พัฒนาโดย OpenGL เพื่อทดสอบหาทางเลือกอื่นๆ และในการ์ด 3 มิติ เกือบทั้งหมด ต่างก็สนับสนุน OpenGL นอกจาก OpenGL ยังมีความเร็วสูงกว่า Direct3D ด้วย โดยให้เน้นที่การแสดงผล 3 มิติ เพียงอย่างเดียวก่อน ขอบเขตของงานวิจัยที่ควรจะพัฒนาในปีหน้าจึงควรเลือกศึกษาโมเดล 3 มิติ และเขียนโปรแกรมที่ใช้โหลดโมเดล 3 มิติ พร้อมข้อมูลแอนิเมชัน ขึ้นมาแสดงผลได้

หากต้องการจะพัฒนาการเล่นเกมผ่านระบบเครือข่าย ควรจะแยกทีมพัฒนามาอีกเพื่อ ศึกษา DirectPlay เพียงอย่างเดียว เพราะ DirectPlay มีความซับซ้อนสูง พอๆ กับ Direct3D เลยทีเดียว

18.3 สรุป

ผลงานวิจัยชิ้นนี้ นอกเหนือจากเรื่องความเร็วในการแสดงผลหรือเฟรมเรท ในส่วนอื่นๆ ก็ถือได้ว่าประสบผลสำเร็จตามความคาดหมาย อย่างไรก็ดีในเรื่องความสวยงาม วัตถุในเกมอาจจะดูไม่สวยงามเนื่องจากทีมงานบุคลากรที่มีความชำนาญในด้านนี้ รวมถึงเรื่องการเคลื่อนไหวของตัวละครด้วย

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้

บรรณานุกรม

ชัยวัฒน์ คำรัตน์ : "*Game Engine เล่ม 1*", บริษัท คอมพิวเตอร์ เอจ เทคโนโลยี จำกัด, พ.ศ. 2542

"*Milkshape 3D Tutorials*", http://www.swissquake.ch/chumbalum-soft , 2001.

"*Thai-3D Tutorials*", www.thai-3d.com , พ.ศ.2544

"*Message Board*", www.thaigamedevx.com , พ.ศ.2544

"*Microsoft DirectX 8.0 Visual C++ Help*" , Microsoft Corporation , 2001.

"*3D Programming with Morfit*" , www.morfit.com , 2001.

.max : "*Computer Time*" , ฉบับที่ 71 กันยายน 2542, หน้า 10 –16

.max : "*Computer Time*" , ฉบับที่ 78 กันยายน 2542, หน้า 10 –15

Andre LaMothe : "*Windows GAME PROGRAMMING for DUMMIES*", IDG Books Worldwide, Inc.1998.

นิรุธ อำนวยศิลป์ : "*คู่มือการเขียนโปรแกรม Microsoft Visual C++ Version 6.0*", บริษัท ซัคเซส มีเดีย จำกัด

เอกสารนี้เป็นเอกสารที่สงวนไว้สำหรับการใช้งานเพื่อการศึกษาเท่านั้น ไม่อนุญาตให้นำไปใช้ประโยชน์ด้านการค้า ไม่ว่ากรณีใดๆ ทั้งสิ้น อีกทั้งห้ามมิให้ดัดแปลงเนื้อหา และต้องอ้างอิงถึงเจ้าของเอกสารทุกครั้งที่มีการนำไปใช้