การพัฒนาเกม 3 มิติ

3D GAME DEVELOPMENT

นาย อนุสรณ์ กระสานติสุข

นาย เอกพล อนันตพรกิจ

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

การพัฒนาเกม 3 มิติ

3D GAME DEVELOPMENT

โดย

นาย อนุสรณ์ กระสานติสุข

นาย เอกพล อนันตพรกิจ

อาจารย์ที่ปรึกษา

ดร. วรวัฒน์ ลิ้มโภคา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

ปริญญานิพนธ์ปีการศึกษา 2544

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาเกม 3 มิติ

3D GAME DEVELOPMENT

ผู้จัดทำ

1. นาย อนุสรณ์ กระสานติสุข รหัสประจำตัว 41014511
2. นาย เอกพล อนันตพรกิจ รหัสประจำตัว 41014548

_____อาจารย์ที่ปรึกษา

(ดร. วรวัฒน์ ลิ้มโภคา)

การพัฒนาเกม 3 มิติ

นาย อนุสรณ์ กระสานติสุข 41014511
นาย เอกพล อนันตพรกิจ 41014548
ดร. วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษา
ปีการศึกษา 2544

บทคัดย่อ

การพัฒนาเกม 3 มิติ เป็นโครงงานที่กล่าวถึง การพัฒนาโปรแกรมคอมพิวเตอร์ประเภทเกม 3 มิติ โดยใช้โอเพนจีแอลและไดเร็กเอ็กซ์มาเป็นส่วนประกอบของโปรแกรมในด้านมัลติมีเดีย ซึ่งเริ่มต้นตั้งแต่การพัฒนาเกมเอนจินต์ซึ่งเป็นหัวใจสำคัญสำหรับการสร้างเกม สำหรับการเขียนโปรแกรมเพื่อพัฒนาเกม 3 มิติ ทั้งหมดนี้อาศัยการโปรแกรมในเชิงวัตถุซึ่งภาษาที่ใช้ในการพัฒนาคือ “ซีพลัสพลัส”

ขอบเขตการทำงานของโครงงานนี้: ในขั้นแรก ศึกษาถึงสถาปัตยกรรมของโอเพนจีแอลและไดเร็กเอ็กซ์เพื่อนำมาช่วยในการพัฒนาโปรแกรมเกมในด้านต่างๆ อาทิ การแสดงผลทางด้านภาพ, เสียง และการติดต่อกับอุปกรณ์อินพุต รวมถึงศึกษาทฤษฎีพื้นฐานที่จะนำไปสู่การพัฒนาเกมเอนจินต์ จากนั้นจึงทำการออกแบบวางแนวความคิดโครงสร้างและส่วนประกอบของเกมเอนจินต์ และเริ่มพัฒนาส่วนประกอบของเอนจินต์ควบคู่ไปด้วยเพื่อศึกษาถึงผลลัพธ์ของวิธีการต่างๆแล้วเลือกใช้อย่างเหมาะสม โดยเริ่มจากด้านกราฟิก, เสียง, อินพุตของเกม และส่วนประกอบอื่นๆที่จำเป็นในตอนเริ่มต้นของการพัฒนาเกมเอนจินต์ รวมถึงเริ่มทำการสร้างและการจัดการงานด้านโมเดล 3 มิติ

ในโครงงานนี้จะเน้นการออกแบบและพัฒนาส่วนแสดงผลภาพเสริมจาก API ที่มีอยู่ และส่วนตรวจสอบการชนกันระหว่างวัตถุภายในฉาก รวมถึงเครื่องมือต่าง ๆ ที่ช่วยอำนวยความสะดวกในการพัฒนาเกมและการพัฒนาเฟรมเวิร์คสำหรับพัฒนาเกมโดยเฉพาะ

ในขั้นที่สอง ทำการพัฒนาเกมเอนจินต์ในส่วนอื่นๆที่เหลือ อาทิ การออกแบบและอ่านแผนที่ การตรวจสอบการชนกันของวัตถุสำหรับฉากต่าง ๆ จากนั้นจึงนำส่วนประกอบต่างๆของเกมเอนจินต์มารวมกันทั้งหมดเพื่อทดสอบและแก้ไขส่วนที่บกพร่อง รวมถึงปรับปรุงเพิ่มเติมส่วนอื่นๆเพื่อความสมบูรณ์ของเกมเอนจินต์ ในขณะเดียวกันก็จะเป็นการนำเอาเกมเอนจินต์ที่ได้พัฒนาแล้วนั้น มาสร้างผลงานที่เป็นเกม 3 มิติ สุดท้ายเป็นการนำผลงานไปทดสอบกับฮาร์ดแวร์คอมพิวเตอร์อื่น ๆ เพื่อหาประสิทธิภาพและจัดทำส่วนติดตั้งโปรแกรมให้สมบูรณ์

3D GAME DEVELOPMENT

Anusorn	Krasarntisuk	41014511
Ekapol	Anantapornkich	41014548
Dr. Worawat	Limpoka	Advisor

ABSTRACT

3D game development is the project that describes the use of OpenGL and DirectX Application Programming Interface (API) as the multimedia component to develop 3D game which emphasize on game engine development that is the core of game creating. The object oriented programming language "C++" is used for the development of project.

This project covers: first of all, studying the architecture of OpenGL and DirectX and the way to implement them for game development in many ways such as a graphics system, sound effects and interfacing with input devices and also studying the basic theory of game engine development. Then use the concept to design the game engine components and beginning to develop that component for learning the result of many methods to choose the appropriate way at the same time. The programming will begin from graphics, sound, input interfacing and the other components that are necessary in the beginning step of game engine development and also 3D modeling field.

The project focuses on the design and development of Graphics API from existing API and collision detection tool to detect collision among objects in the game's scene including useful tools for developing game and the game framework development.

After that, develop the left engine components such as game mapping and collision detection. Then integrate all engine components for engine testing and improving to final completed. In the meantime create 3D games by using the developed 3D games project. Finally, test the game with various class of PC for measuring performance and packaging the 3D game so that they are ready to install.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือ และความร่วมมือจากหลาย ๆ ฝ่ายด้วยกัน บุคคลสำคัญที่ผลักดันให้ปริญญานิพนธ์นี้สำเร็จลงได้ก็คือ ดร.วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษาปริญญานิพนธ์ ที่ให้ความเอาใจใส่ แนะนำ และช่วยเหลือเสมอมา ซึ่งต้องขอขอบพระคุณเป็นอย่างมาก

ขอขอบพระคุณบุคคลสำคัญที่สุดที่ทำให้ข้าพเจ้ามีวันนี้ ก็คือ บิดา มารดา อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูผู้เขียนมาเป็นอย่างดี พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจ เอาใจใส่เสมอมาในทุก ๆ ด้านอันหาที่เปรียบมิได้ ข้าพเจ้าขอระลึกในพระคุณอันสุดประมาณ และขอกราบขอบพระคุณมา ณ ที่นี้ นอกจากนี้ขอขอบพระคุณคณาจารย์ทุกท่านที่ประสิทธิประสาทวิชาความรู้ให้แก่ข้าพเจ้าตลอดมา

ขอขอบคุณทุกคนในห้องวิจัย OLALA ที่คอยซักถามความคืบหน้า คอยกระตุ้น และคอยเป็นกำลังใจในการทำงานจนสำเร็จได้ และขอขอบคุณสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ที่เปิดโอกาสให้ผู้จัดทำได้เข้ามาศึกษาเล่าเรียนวิชาจนมีวันนี้

สุดท้ายขอขอบคุณเพื่อน ๆ ในแต่ละกลุ่มที่ร่วมพัฒนาเกมด้วยกันในภาควิชาตลอดระยะเวลาเกือบ 1 ปี ที่ให้ความช่วยเหลือสนับสนุนในเรื่องการหาข้อมูลทางด้านทฤษฎี และช่วยเหลือในด้านการออกแบบสร้างแบบจำลอง 3 มิติของตัวละครต่าง ๆ และช่วยพัฒนาโปรแกรมในส่วนต่าง ๆ ด้วย ทำให้แบ่งเบางานลงไปบางส่วน ซึ่งต้องยอมรับว่าถ้าไม่ทำงานร่วมกันแล้วการจะทำให้ปริญญานิพนธ์นี้เสร็จสมบูรณ์ได้คงต้องลำบากขึ้นอีกมาก

อนุสรณ์ กระสานติสุข
เอกพล อนันตพรกิจ

สารบัญ

หน้าที่

สารบัญ (ต่อ)

หน้าที่

สารบัญ (ต่อ)

หน้าที่

สารบัญภาพ

หน้าที่

สารบัญภาพ (ต่อ)

หน้าที่

สารบัญภาพ (ต่อ)

หน้าที่

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ในปัจจุบันการพัฒนาทางด้านเทคโนโลยีคอมพิวเตอร์นั้นได้ก้าวหน้าไปอย่างรวดเร็ว โดยเฉพาะในด้านเกมคอมพิวเตอร์ ฮาร์ดแวร์ได้มีการพัฒนาระบบการแสดงผลให้สามารถประมวลผลภาพในลักษณะสามมิติ ที่มีทั้งความเร็วและความสวยงาม ไม่ว่าจะเป็นการ์ดแสดงผลสามมิติที่จัดการด้านการแสดงผลสามมิติโดยตรง หรือหน่วยประมวลผลกลางที่พัฒนาคำสั่งต่าง ๆ มารองรับการประมวลผลข้อมูลสามมิติ ทางด้านซอฟแวร์ก็มีโปรแกรมต่าง ๆ เข้ามาช่วยสร้างสรรค์ภาพสามมิติ ทำให้มีการสร้างเกมในลักษณะสามมิติออกมามากมาย ทั้งนี้ทั้งนั้น ส่วนที่ช่วยให้การทำเกมนั้นสำเร็จโดยง่าย หรือผู้อยู่เบื้องหลัง ส่วนหนึ่งก็คือ “เกมเอนจินต์” “โอเพนจีแอล” และ “ไดเร็กเอ็กซ์”

เกมเอนจินต์นั้น ช่วยอำนวยความสะดวกให้กับการสร้างเกมอย่างมาก มีฟังก์ชันเกี่ยวกับเกมต่างๆไว้ให้เรียกใช้งาน ช่วยในการสร้างภาพเคลื่อนไหวในเกม ส่วนไดเร็กเอ็กซ์นั้น เป็นตัวกลางช่วยในการติดต่อกับฮาร์ดแวร์เพื่อการจัดการกับภาพ เสียง การรับอินพุต ได้โดยตรงและรวดเร็ว

เกมเอนจินต์ที่ดี นอกจากต้องใช้งานง่ายแล้ว เรื่องความเร็วในเกมก็เป็นสิ่งสำคัญ ความรู้พื้นฐานในเรื่องการทำเกมสามมิติจึงเป็นสิ่งสำคัญ ที่ต้องนำมาใช้พัฒนาเกมสามมิติให้มีประสิทธิภาพ ไม่ว่าจะเป็นเทคนิคในการคำนวณทางคณิตศาสตร์ ไปจนถึงอัลกอริทึมในการเขียนโปรแกรมกับระบบสามมิติ ดังนั้น ในการพัฒนาเกมสามมิติให้ได้ดียิ่ง การศึกษาให้ได้ลึกและใกล้ชิดอย่างหนึ่งก็คือ เริ่มตั้งแต่การทำเกมเอนจินต์ขึ้นมาเอง และใช้เกมเอนจินต์นั้นในการพัฒนาเกมสามมิติต่อไป

การทำงานวิจัยนี้ นอกจากได้ศึกษาการใช้งานโอเพนจีแอลและไดเร็กเอ็กซ์ในติดต่อกับฮาร์ดแวร์มัลติมีเดียและนำมาใช้ในการดึงความสามารถทางฮาร์ดแวร์ต่าง ๆ ได้อย่างเต็มประสิทธิภาพแล้ว ยังได้ศึกษาเกี่ยวกับการประมวลผลภาพสามมิติพื้นฐาน เช่น พิกัดในระบบสามมิติ การหมุนภาพสามมิติ การทำภาพเคลื่อนไหว และเทคนิคต่างๆที่ใช้ในการประมวลผล เช่น การทำงานกับมุมกล้อง การประมวลผลภาพเคลื่อนไหวให้รวดเร็ว การดึงความสามารถทางด้านฮาร์ดแวร์และซอฟแวร์ของคอมพิวเตอร์ออกมาใช้อย่างเต็มที่ เป็นต้น

1.2 วัตถุประสงค์ของงานวิจัย

1.2.1 เพื่อศึกษาและพัฒนาเกม โดยนำเสนอในรูปแบบ 3 มิติ

1.2.2 ศึกษาการทำงานของโอเพนจีแอลและไดเร็กเอ็กซ์ ในการติดต่อกับอุปกรณ์ฮาร์ดแวร์ ด้านมัลติมีเดีย เช่น การ์ดจอ การ์ดเสียง อุปกรณ์อินพุท เป็นต้น เพื่อเป็นพื้นฐานในการพัฒนาเกมสามมิติ

1.2.3 นำโอเพนจีแอลและไดเร็กเอ็กซ์ มาใช้ในการพัฒนาเกมสามมิติ

1.2.4 ออกแบบและพัฒนาเกมเอนจินต์ โดยใช้โอเพนจีแอล, ไดเร็กเอ็กซ์ และ ซีพลัสพลัส

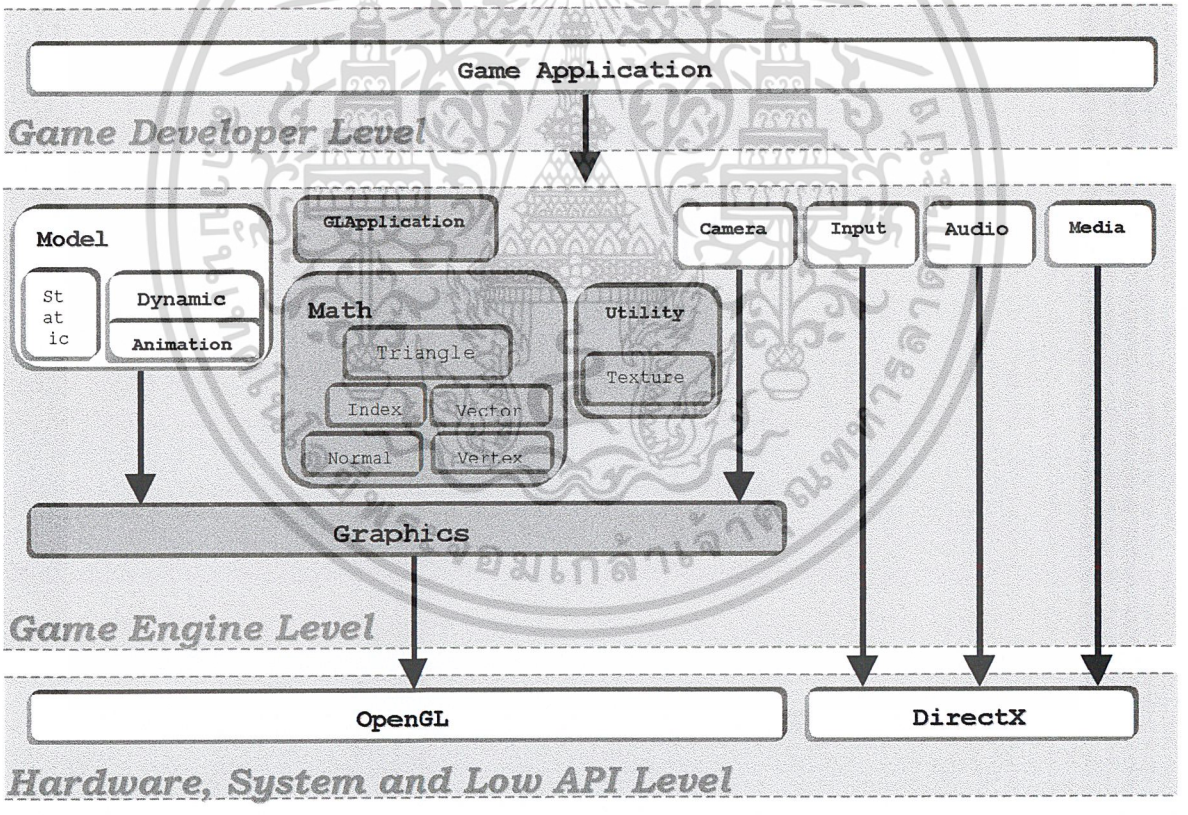
1.2.5 สร้างเกมสามมิติ จากเกมเอนจินต์ที่ได้ออกแบบเสร็จแล้ว

1.2.6 เพื่อพัฒนา Library tool รวมถึง Software tool ที่จะใช้ในการพัฒนาเกม

1.3 ขอบเขตของงานวิจัย

งานวิจัยนี้เน้นที่การศึกษาและสร้างเกมเอนจินต์สามมิติ โดยใช้เทคโนโลยีโอเพนจีแอล ไดเร็กเอ็กซ์ และภาษาโปรแกรมซีพลัสพลัส เริ่มตั้งแต่การออกแบบเอนจินต์ การสร้างเกมเอนจินต์โดยหลักการโปรแกรมเชิงวัตถุ จนสามารถนำมาช่วยสร้างเกมสามมิติได้ จากนั้นนำเกมเอนจินต์ที่ได้สร้างเสร็จแล้ว มาสร้างเกมสามมิติขึ้นมา เพื่อทดสอบใช้งานเอนจินต์ และหาข้อบกพร่อง เพื่อนำไปสู่การพัฒนาให้สมบูรณ์ยิ่งขึ้นต่อไป

สำหรับในงานวิจัยนี้ได้เกิดจากการรวมตัวของกลุ่มนักศึกษาที่มีความสนใจในด้านเดียวกันหลายกลุ่ม จึงได้แบ่งกลุ่มร่วมกันทำงานเป็น 4 กลุ่มย่อย หลังจากได้มีการออกแบบร่วมกันแล้ว ได้มีการแบ่งงานกัน โดยนักศึกษาในกลุ่มนี้ได้รับผิดชอบในส่วน ระบบกราฟฟิก, แอพพลิเคชันเฟรมเวิร์ค และ ยูทิลิตี้อื่นที่เป็นประโยชน์ ดังแสดงในรูปที่ 1-1



รูปที่ 1-1 แสดงโครงสร้างของเกมเอนจินต์ และส่วนที่กลุ่มรับผิดชอบ

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1.4.1 ทักษะในการพัฒนาเกม โดยนำเสนอในรูปแบบ 3 มิติ เช่น ก ารสร้างตัวละคร การวางโครงเรื่องของเกม การสร้างภาพสามมิติ เป็นต้น

1.4.2 ความรู้ในด้านการประมวลผลภาพสามมิติ เช่น การหมุนภาพ การสร้างภาพเคลื่อนไหว การใช้คณิตศาสตร์มาช่วยจัดการภาพสามมิติ เป็นต้น

1.4.3 เทคนิคในการเขียนโปรแกรมประมวลผลภาพสามมิติ

1.4.4 การใช้ไดเร็กเอ็กซ์ ติดต่อกับอุปกรณ์มัลติมิเดีย

1.4.5 การออกแบบโปรแกรมเชิงวัตถุ

1.4.6 การเขียนโปรแกรมซีพลัสพลัส เพื่อติดต่อกับไดเร็กเอ็กซ์

1.5 วิธีการดำเนินงาน

1.5.1 เริ่มจากการศึกษาทฤษฎีในการสร้างเกมสามมิติ เช่น ระบบพิกัด เวกเตอร์ การหมุนกล้องตลอดจนแนวความคิดในการออกแบบ พัฒนา ซึ่งได้กล่าวไว้ในบทที่ 2 สำหรับในบทที่ 3 นั้น กล่าวถึงเรื่อง แนวความคิดในการออกแบบ เกมเอนจินต์ สำหรับนำไปพัฒนาเกม 3 มิติ

1.5.2 ทฤษฎีที่เกี่ยวข้องกับการพัฒนาเกมเอนจินต์ ในบทที่ 2 แบ่งออกเป็น 4 กลุ่มดังนี้

กลุ่มที่ 1:

- ทฤษฎีและหลักการกล้องในระบบ 3 มิติ สำหรับเกมเอนจินต์
- ทฤษฎีและหลักการ การสร้างแอนิเมชันสำหรับเกมเอนจินต์

กลุ่มที่ 2:

- ทฤษฎีและหลักการของ Direct Audio
- ทฤษฎีและหลักการของ Direct Input
- ทฤษฎีและหลักการของ Direct Show

กลุ่มที่ 3:

- การสร้าง โมเดล 3 มิติ ด้วย 3D Studio MAX

กลุ่มที่ 4:

- ทฤษฎีการแสดงผลภาพ 3 มิติ (3D Rendering)
- เทคนิคการตรวจสอบการชนสำหรับเกม 3 มิติ

โดยปริญญานิพนธ์ฉบับนี้จะได้นำเสนอเนื้อหาในกลุ่มที่ 4

1.5.3 ออกแบบเกมเอนจินต์ จะอธิบายโครงสร้างของเกมเอนจินต์แต่ละส่วน และส่วนประกอบต่าง ๆ ของเกมเอนจินต์

1.5.4 พัฒนาส่วนประกอบต่างๆที่ได้ออกแบบไว้แล้ว

1.5.5 จากนั้นนำเอนจินต์ในแต่ละส่วนมาประกอบรวมกัน และกำหนดรูปแบบไฟล์ข้อมูลที่จะนำไปใช้กับเกมเอนจินต์ จัดทำแอพพลิเคชันเฟรมเวิร์ค สำหรับผู้นำเกมเอนจินต์ไปพัฒนาเกมในขั้นตอนต่อไป

1.5.6 เมื่อได้เกมเอนจินต์ที่ใช้งานได้แล้ว จึงนำเกมเอนจินต์ที่ได้ ไปสร้างเป็นเกมสามมิติเพื่อทดสอบการใช้งานเกมเอนจินต์

1.5.7 รวบรวมผลการทดสอบ วิเคราะห์ปัญหา และหาแนวทางการแก้ปัญหาต่าง ๆ เพื่อนำมาพัฒนาหรือเป็นข้อมูลในการพัฒนาขั้นต่อไป

บทที่ 2
ทฤษฎีและหลักการพัฒนาเกม 3 มิติ

2.1 ทฤษฎีการแสดงผลภาพ 3 มิติ (3D Rendering)

ในการพัฒนาโปรแกรมเกม 3มิติ จำเป็นอย่างยิ่งที่จะต้องมีความเข้าใจถึงทฤษฎีพื้นฐานที่จำเป็นสำหรับนำไปประยุกต์ใช้ โดยเฉพาะทฤษฎีทางด้านคอมพิวเตอร์กราฟฟิก ซึ่งเป็นพื้นฐานสำหรับการพัฒนาส่วนประกอบอื่น ๆ ในการพัฒนาเอนจินต์ได้ เช่น การสร้างภาพเคลื่อนไหว (Animation) ให้กับวัตถุภายในเกม การเขียนโปรแกรมติดต่อกับกราฟฟิกเอนจินต์ การสร้างแบบจำลอง 3 มิติ เป็นต้น

2.1.1 พิกัดในระบบ 3 มิติ (3D Coordinate System)

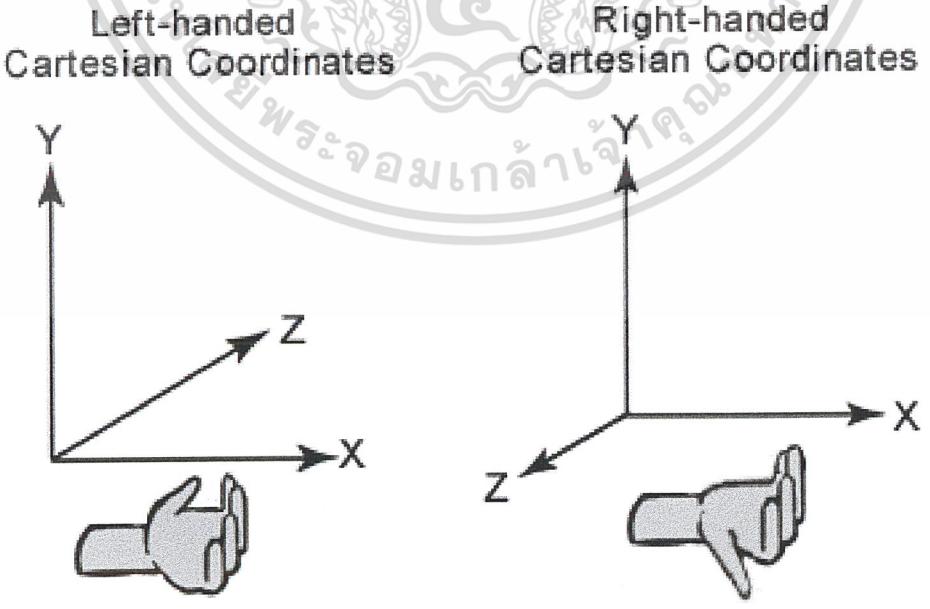
2.1.1.1 ระบบพิกัดคาร์ทีเซียน (Cartesian Coordinate System)

เป็นระบบพิกัดที่ทำความเข้าใจได้ง่ายที่สุด เพราะเป็นระบบพิกัดที่คนส่วนใหญ่คุ้นเคย และมีที่ใช้แพร่หลายที่สุด อีกทั้งยังเหมาะสมในการใช้สำหรับสร้างโปรแกรม ระบบพิกัด คาร์ทีเซียนประกอบด้วยสามแกนที่ตั้งฉากซึ่งกันและกันสำหรับกำหนดพิกัด โดยมักจะตั้งชื่อแกนดังกล่าวให้เป็น X, Y และ Z ระบบพิกัดนี้โดยทั่วไปมักมีการกำหนดแนวแกนได้ 2 แบบคือ

1) ระบบพิกัดคาร์ทีเซียนแบบมือซ้าย (Left-handed Cartesian Coordinate System)

2) ระบบพิกัดคาร์ทีเซียนแบบมือขวา (Right-handed Cartesian Coordinate System)

โดยแสดงได้ดังภาพดังต่อไปนี้

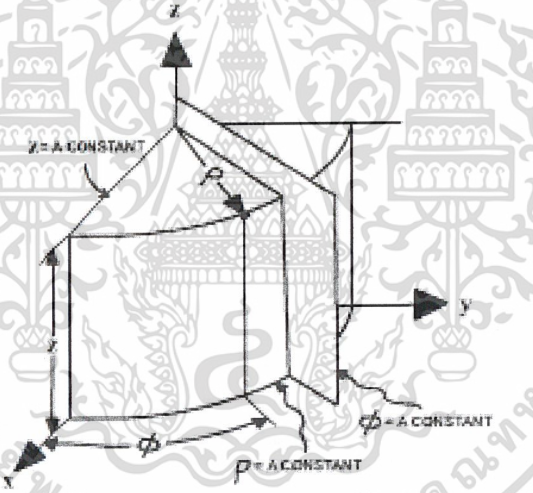


รูปที่ 2-1 แสดงระบบพิกัดคาร์ทีเซียนแบบมือซ้ายและแบบมือขวา

โดยทั่วไป DirectX Graphics API นั้นระบบพิกัดที่ใช้จะเป็นระบบมือซ้าย ส่วนใน OpenGL API นั้นจะใช้ระบบมือขวา ในการแสดงผลภาพ 3 มิตินั้นเราจะต้องส่ง Vertex Array ที่เราต้องการแสดงผลเข้าไปใน Geometry Pipeline สุดท้ายเราก็จะได้ภาพ 2 มิติออกมาแสดงบนจอภาพ เหตุที่เราต้องแปลงข้อมูลที่เป็น 3 มิติให้เป็น 2 มิตินั้นเพราะหน้าจอของเราไม่สามารถแสดงภาพ 3 มิติตรง ๆ ได้จึงต้องทำการส่งข้อมูลเข้า Geometry Pipeline ซึ่งมีขั้นตอนต่าง ๆ มากมาย เพื่อให้ได้ผลลัพธ์เป็นภาพ 2 มิติที่สอดคล้องกับข้อมูลนั้นบนมุมมองที่ต้องการ ซึ่งสามารถนำมาแสดงบนหน้าจอได้ ดังจะได้กล่าวในรายละเอียดต่อไปในบทนี้

2.1.1.2 ระบบพิกัดทรงกระบอก (Cylindrical Coordinate System)

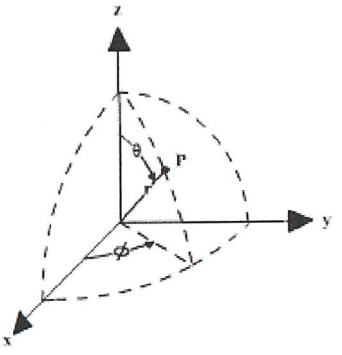
ในระบบพิกัดนี้ เราสามารถระบุพิกัดของจุดแต่ละจุดได้โดยใช้ระยะห่างระหว่างจุดนั้น ๆ กับจุดกำเนิด (Origin) ซึ่งแทนด้วยสัญลักษณ์ ρ มุมระหว่างแกนอ้างอิงในแนวระดับ (มักสัมพันธ์กับแกน X และระนาบ XY ในระบบพิกัดคาร์ทีเซียน) กับเส้นตรงที่ลากจากจุดกำเนิดไปยังตำแหน่งของจุดนั้น ๆ ในระนาบของจุดกำเนิดกับแกนอ้างอิงในแนวระดับ (ซึ่งสัมพันธ์กับระนาบ XY ในระบบพิกัดคาร์ทีเซียน) ซึ่งแทนด้วยสัญลักษณ์ ϕ และระดับความสูงของจุดนั้น ๆ ซึ่งมักแทนด้วย Z ดังแสดงในรูป



รูปที่ 2-2 แสดงระบบพิกัดทรงกระบอก

2.1.1.3 ระบบพิกัดทรงกลม (Spherical Coordinate System)

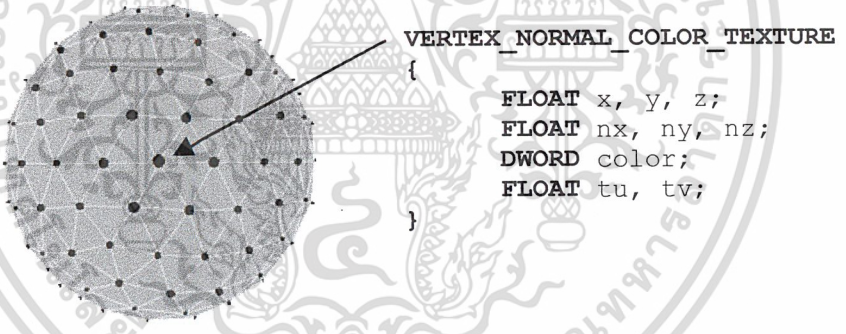
ในระบบพิกัดนี้ เราสามารถระบุพิกัดของจุดแต่ละจุดได้โดยใช้ระยะห่างระหว่างจุดนั้น ๆ กับจุดกำเนิด (Origin) ซึ่งแทนด้วยสัญลักษณ์ r มุมระหว่างแกนอ้างอิงในแนวระดับ (มักสัมพันธ์กับแกน X และระนาบ XY ในระบบพิกัดคาร์ทีเซียน) กับเส้นตรงที่ลากจากจุดกำเนิดไปยังตำแหน่งของจุดนั้น ๆ ในระนาบของจุดกำเนิดกับแนวระดับ (ซึ่งสัมพันธ์กับระนาบ XY ในระบบพิกัดคาร์ทีเซียน) ซึ่งแทนด้วยสัญลักษณ์ ϕ และมุมระหว่างแกนตั้งฉากแนวระดับ (แกน Z) กับเส้นตรงที่ลากจากจุดกำเนิดไปยังตำแหน่งของจุดนั้น ๆ ดังแสดงในรูป



รูปที่ 2-3 แสดงระบบพิกัดทรงกลม

2.1.2 เวอร์เท็กซ์ (Vertex)

เวอร์เท็กซ์เป็นจุดพิกัดในระบบ 3 มิติ ซึ่งใช้ในการอ้างอิงตำแหน่ง ซึ่งในการประมวลผลทางด้านกราฟฟิกนั้นเรามักจะกำหนดคุณสมบัติเพิ่มเติมให้กับเวอร์เท็กซ์ เช่น Color, Normal, Texture Coordinate เป็นต้น ซึ่งคุณสมบัติเหล่านี้เมื่อประกอบกับตำแหน่งของเวอร์เท็กซ์แล้วจะนำไปใช้ในการเก็บข้อมูลที่แทนแบบจำลอง 3 มิติได้ โดยแสดงได้ดังรูปตัวอย่าง



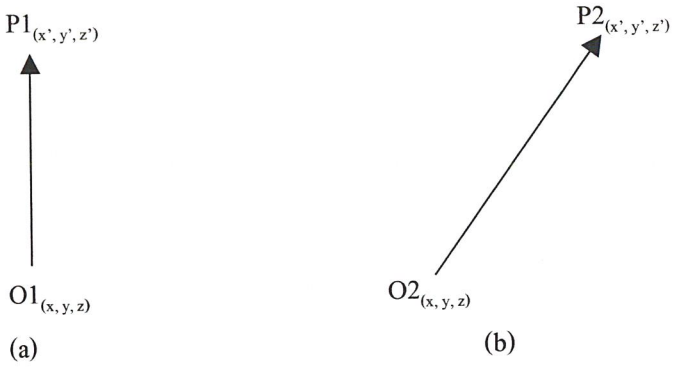
```
VERTEX_NORMAL_COLOR_TEXTURE
{
    FLOAT x, y, z;
    FLOAT nx, ny, nz;
    DWORD color;
    FLOAT tu, tv;
}
```

รูปที่ 2-4 แสดงตัวอย่างการนำเวอร์เท็กซ์มาใช้ในการเก็บข้อมูลที่แทนแบบจำลอง 3 มิติ

2.1.3 เวกเตอร์ (Vector)

เวกเตอร์เป็นปริมาณที่มีทั้งขนาดและทิศทาง เป็นองค์ประกอบทางคณิตศาสตร์ที่มีความสำคัญมากด้านการประยุกต์ใช้ในงานการประมวลผลกราฟฟิก ซึ่งจะต่างกับปริมาณสเกล่าร์ตรงที่เวกเตอร์นั้นจะมีทิศทาง หรือกล่าวได้อีกอย่างหนึ่งว่าเวกเตอร์มีองค์ประกอบมาจากปริมาณสเกล่าร์ n ตัว เพื่อแทนขนาดและทิศทางในระบบ n มิติ

เรามักจะแทนเวกเตอร์โดยใช้สัญลักษณ์ $\overrightarrow{OP}$ โดยหมายถึงเวกเตอร์นี้มีทิศทางจากจุด O ไปยังจุด P และมีขนาดเท่ากับระยะห่างระหว่างจุด O และจุด P



รูปที่ 2-5 แสดงตัวอย่างเวกเตอร์

หากเราทำการบวกเวกเตอร์ทั้งสองจะได้ค่าดังนี้

$R = V1 + V2$

$R = (V1_x + V2_x, V1_y + V2_y, V1_z + V2_z)$

เมื่อทำการคูณเวกเตอร์ด้วยปริมาณสเกล่าร์ ซึ่ง ทำได้โดยการคูณปริมาณสเกลล่าร์กับทุกคอมโพแนนท์ของเวกเตอร์นั้น ๆ จะได้

$V * s = (V_x * s, V_y * s, V_z * s)$

สำหรับขนาด (Magnitude/length) ของเวกเตอร์ใด ๆ นั้น สามารถเขียนแทนด้วย $|V|$ ซึ่งสามารถหาได้โดยใช้กฎของปิทากอรัส ได้สมการดังนี้

$$|V| = \sqrt{(V_X^2 + V_Y^2 + V_Z^2)}$$

เวกเตอร์ที่ขนานกัน (Parallel Vector) หมายถึง เวกเตอร์คู่ใด ๆ ที่มีทิศทางเดียวกันหรือทิศทางตรงข้ามกัน

เวกเตอร์ร่วมระนาบ (Coplanar Vector) หมายถึง เวกเตอร์ตั้งแต่ 3 ตัวขึ้นไปที่อยู่ในระนาบเดียวกัน

2.1.4 เมทริกซ์ (Matrix)

เมทริกซ์มิติ m × n ประกอบด้วยจำนวนจริงที่เขียนเรียงเป็นแถว (row) m แถว และเขียนในแนวดิ่ง n หลัก (column) โดยปิดล้อมจำนวนจริงเหล่านี้ด้วยเครื่องหมาย [ ] หรือ ( ) จำนวนแต่ละจำนวนในเมทริกซ์เรียกว่า สมาชิกของเมทริกซ์

$\begin{bmatrix} 2 & 1 & 0 \\ 5 & 3 & 2 \end{bmatrix}$ เป็นเมทริกซ์มิติ 2 × 3 $\begin{bmatrix} 1 \\ 0 \\ 2 \\ 4 \end{bmatrix}$ เป็นเมทริกซ์มิติ 4 × 1

ถ้าเมทริกซ์ที่มีจำนวนแถวและจำนวนหลักเท่ากันเท่ากับ n จะเรียกเมทริกซ์นั้นว่า เมทริกซ์จัตุรัสมิติ n และเรียก $a_{11}, a_{12}, ..., a_{nn}$ ว่าเป็นสมาชิกในแนวทแยงของ A เมื่อ A เป็นเมทริกซ์จัตุรัสมิติ n

a_{11}, a_{22}, a_{33} เป็นสมาชิกในแนวทแยงของ A หากเมทริกซ์จัตุรัส $A = [a_{ij}]$ ซึ่งมีสมาชิกทุกตัวนอกจากแนวทแยงเป็นศูนย์ทั้งหมด (นั่นคือ $a_{ij} = 0$ ถ้า $I \neq j$) เรียก A ว่าเป็น เมทริกซ์ทแยง (Diagonal Matrix)

ให้ $A = [a_{ij}]_{m \times n}$ และ $B = [b_{ij}]_{m \times n}$ เป็นเมทริกซ์ที่มีมิติเท่ากัน เรียก A เท่ากับ B ก็ต่อเมื่อ $a_{ij} = b_{ij}$ สำหรับ $1 \leq i \leq m, 1 \leq j \leq n$ และเขียนแทนด้วย A = B

ถ้า $A= [a_{ij}]_{m \times n}$ และ $B = [b_{ij}]_{m \times n}$ แล้วผลบวกของ A และ B เขียนแทนด้วย A + B หมายถึง $C = [c_{ij}]_{m \times n}$ ซึ่ง

$c_{ij} = a_{ij} + b_{ij}$ $(1 \leq I \leq m, 1 \leq j \leq n)$

ให้ $A = [a_{ij}]_{m \times n}$ เป็นเมทริกซ์ และ k เป็นจำนวนจริงใดๆ แล้ว ผลคูณสเกลาร์ kA จะเป็นเมทริกซ์ $[ka_{ij}]_{m \times n}$ เช่น

ให้ $A = \begin{bmatrix} 2 & 1 & 0 \\ 5 & 3 & 2 \end{bmatrix}$ และ k = 3

ดังนั้น $kA = \begin{bmatrix} 3(2) & 3(1) & 3(0) \\ 3(5) & 3(3) & 3(2) \end{bmatrix} = \begin{bmatrix} 6 & 3 & 0 \\ 15 & 9 & 6 \end{bmatrix}$

ให้ $A = [a_{ij}]_{m \times p}$ และ $B = [b_{ij}]_{p \times n}$ แล้ว ผลคูณของ A และ B เขียนแทนด้วย AB หมายถึง

$$c = [a_{ij}]_{m \times n} \text{ โดย } c_{ij} = a_{i1}b_{1j} + a_{i2}b_{i2} + \ldots + a_{ip}b_{ip}$$

$$= \sum_{k=1}^{p} a_{ik}\, b_{kj} \quad (1 \leq i \leq m, 1 \leq j \leq n)$$

เช่น

$$AB = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} b_{11} & b_{12} & b_{13} \\ b_{21} & b_{22} & b_{23} \end{bmatrix}$$

$$= \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} & a_{11}b_{13} + a_{12}b_{23} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} & a_{21}b_{13} + a_{22}b_{23} \end{bmatrix}$$

ผลคูณ AB ไม่มี หรือไม่นิยาม (undefine) ถ้า A เป็นเมทริกซ์ขนาด $m \times p$ และ B เป็นเมทริกซ์ขนาด $q \times n$ เมื่อ $p \neq q$

การคูณเมทริกซ์นั้นไม่มีคุณสมบัติการสลับที่ซึ่งหมายถึง $A \times B \neq B \times A$ เมื่อ A และ B เป็น Matrix จัตุรัส

เมทริกซ์เอกลักษณ์ (Identity Matrix) คือ เมทริกซ์จัตุรัสที่มีสมาชิกในแนวทแยงเป็น 1 ทั้งหมด ซึ่งเขียนแทนด้วย I หรือ I_n แทนเมทริกซ์เอกลักษณ์มิติ n เช่น

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ถ้า A เป็นเมทริกซ์มิติ $m \times n$ แล้วจะได้ว่า $AI_n = A$ และ $I_m A = A$

B เป็นอินเวอร์สการคูณของเมทริกซ์ A (B เป็นอินเวอร์สของ A) เมื่อ B เป็นเมทริกซ์ซึ่ง $AB = BA = I$ โดยที่ A เป็นเมทริกซ์จัตุรัสใดๆ

ให้ A เป็นเมทริกซ์จัตุรัสมิติ n จะกล่าวว่า A มีตัวผกผัน (Invertible) หรือ A เป็นเมทริกซ์ซึ่งมิใช่เอกฐาน (Non-Singular Matrix) ถ้าสามารถหาเมทริกซ์จัตุรัส B ได้ ซึ่งทำให้

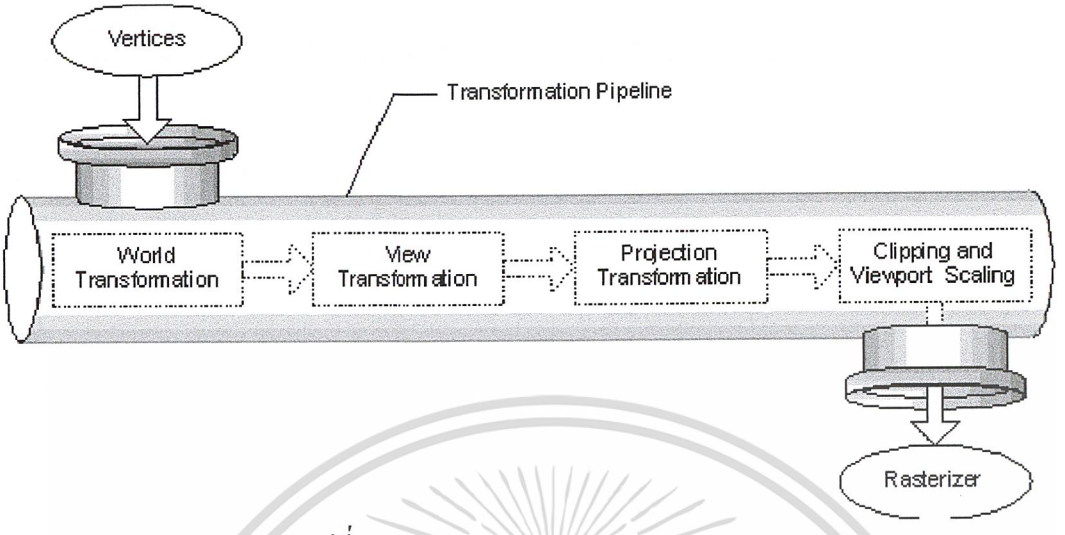
$$AB = I_n = BA$$

และเรียกเมทริกซ์ B ว่าเป็นอินเวอร์สการคูณ ของ A เขียนแทนด้วยสัญลักษณ์ A^{-1} นั่นคือ ถ้า A เป็นเมทริกซ์ซึ่งมิใช่เมทริกซ์เอกฐานมิติ n แล้ว จะได้ว่า

$$AA^{-1} = A^{-1}A = I_n$$

และถ้า A ไม่มีอินเวอร์ส แล้วจะเรียก A ว่าเป็นเมทริกซ์เอกฐาน (Singular Matrix)

2.1.5 Geometry Pipeline



รูปที่ 2-6 DirectX Graphics Geometry Pipeline

ในการแสดงผลภาพ 3 มิตินั้น เราจะต้องนำ Vertex Array ป้อนเข้าไปใน Geometry Pipeline ดังรูป ซึ่ง Graphics Engine ที่ Implement ใน API จะทำกระบวนการที่เรียกว่า 3D Transformation เพื่อแปลงข้อมูล 3 มิติเหล่านี้ให้เป็นจุดบนหน้าจอซึ่งเป็นภาพ 2 มิติ

ขั้นตอนต่าง ๆ ใน Geometry Pipeline มีดังนี้

1) World Transformation
2) View Transformation
3) Projection Transformation
4) Clipping and Viewport Scaling

2.1.5.1 World Transformation

ขั้นตอนนี้จะเป็นขั้นตอนในการแปลง Local Coordinate ไปเป็น World Coordinate ซึ่งจะเป็นตำแหน่งจริงๆ ในปริภูมิ 3 มิติ ซึ่งวิธีการนั้นเราจะใช้การคูณตำแหน่งใน Vertex ด้วย Transformation Matrix ต่าง ๆ เพื่อให้ได้ตำแหน่งที่เราต้องการ ซึ่งโดยทั่วๆ ไปจะมี Transformation ในรูปแบบต่าง ๆ ดังนี้

1) Translation
2) Rotation
3) Scaling

1) Translation

เป็นการเคลื่อนย้ายตำแหน่งของวัตถุ ซึ่งใช้ Translation Matrix ดังรูป

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

Translation Matrix

2) Rotation

เป็นการหมุนวัตถุรอบแกน X, Y หรือ Z ซึ่งใช้ Rotation Matrix ดังรูป

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

Rotation around X Axis Matrix

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

Rotation around Y Axis Matrix

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

Rotation around Z Axis Matrix

3) Scaling

เป็นการย่อหรือขยายวัตถุ ซึ่งใช้ Scaling Matrix ดังรูป

$$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Scaling Matrix

ในการแปลง Vertex จาก Local Coordinate ให้เป็น World Coordinate นั้นในบางครั้งถ้าการแปลงของเรามีความซับซ้อนมากเราอาจจะต้องใช้ Matrix หลายตัวในการแปลงคูณต่อกันไป (Matrix Concatenation) เพื่อสร้าง Matrix สุดท้ายสำหรับใช้เป็น Transformation Matrix ก่อนแล้วจึงจะเอามาคูณกับตำแหน่งใน Vertex จริงเพื่อให้ได้ผลลัพธ์สุดท้ายก็ได้

การคูณ Matrix กับตำแหน่งของ Vertex จะเป็นดังต่อไปนี้

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{bmatrix}$$

$$x' = (x \times M_{11}) + (y \times M_{21}) + (z \times M_{31}) + (1 \times M_{41})$$

$$y' = (x \times M_{12}) + (y \times M_{22}) + (z \times M_{32}) + (1 \times M_{42})$$

$$z' = (x \times M_{13}) + (y \times M_{23}) + (z \times M_{33}) + (1 \times M_{43})$$

จากรูป x, y, z คือ ตำแหน่งของ Vertex พอคูณกับ Matrix ดังรูปแล้ว จะได้ผลลัพธ์ เป็น x', y', z' ซึ่งเป็นผลลัพธ์ของการแปลงพิกัด

2.1.5.2 View Transformation

ขั้นตอนนี้ใช้สำหรับแปลง World Coordinate ให้สอดคล้องกับการเห็นของผู้สังเกต เนื่องจาก World Coordinate บอกเพียงตำแหน่งจริง ๆ ใน 3D Space เท่านั้น ยังไม่มีการอ้างอิงจากผู้สังเกตเลย ดังนั้นจึงต้องมีการบอก ว่าผู้สังเกตอยู่ที่ไหน และมองไปทางไหน ขั้นตอนนี้มักใช้ในการปรับมุมมองของกล้อง โดยการนำ View Matrix มาคูณกับ World Coordinate จากการทำ World Transformation

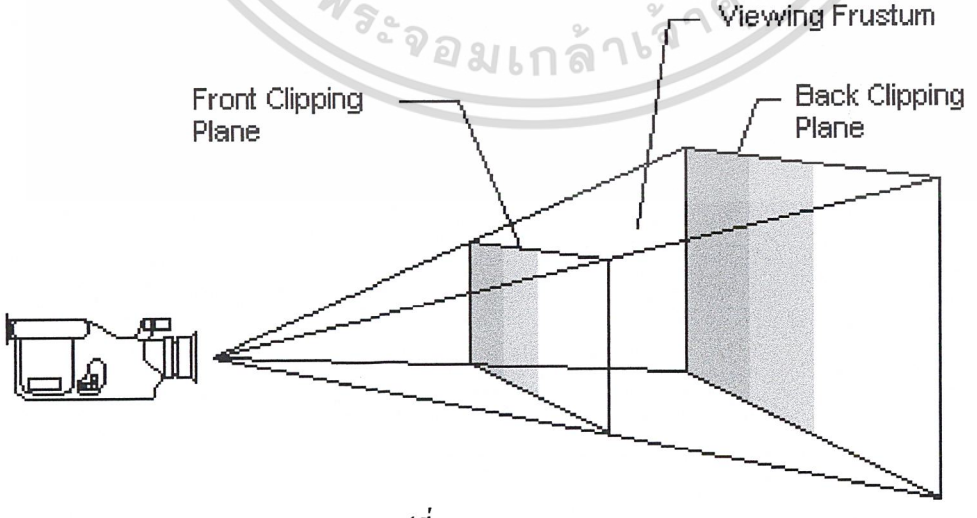
$$\begin{bmatrix} u_x & v_x & n_x & 0 \\ u_y & v_y & n_y & 0 \\ u_z & v_z & n_z & 0 \\ -(u \cdot c) & -(v \cdot c) & -(n \cdot c) & 1 \end{bmatrix}$$

จากรูป View Matrix ประกอบด้วย Vector 3 ตัวบอกทิศทางของ ผู้สังเกตว่าหันหน้าไปทางใด ซึ่งมี Vector u, v, n บอกถึง Up, Right, และ View Direction ตามลำดับ และ Vector c ซึ่งบอกตำแหน่งของ ผู้สังเกตใน World Space

View Matrixนั้นให้ข้อมูลที่จำเป็นในการปรับมุมมองของกล้องใน Game Engine ได้ซึ่งใช้ในการบอกตำแหน่งและทิศทางของกล้อง

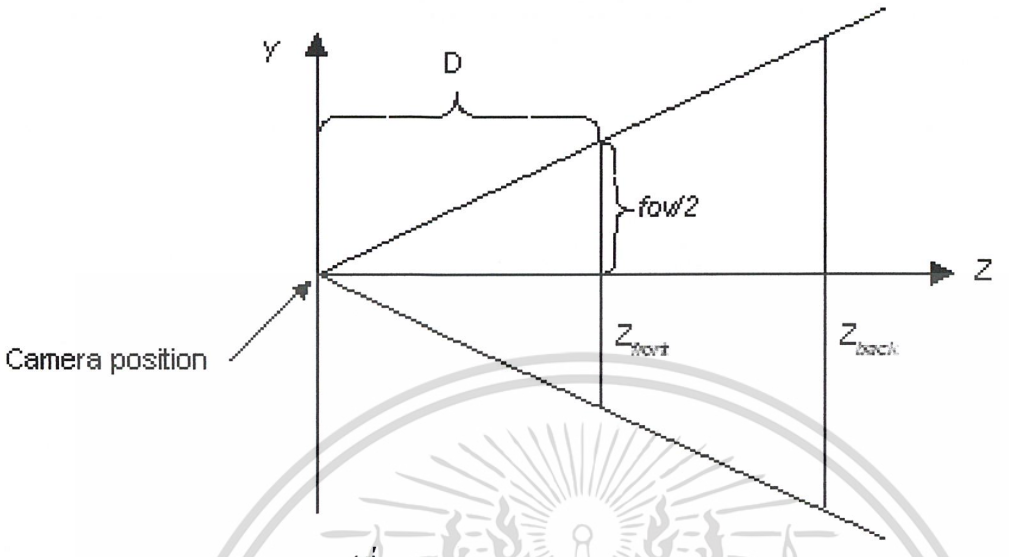
2.1.5.3 Projection Transformation

เมื่อเราได้ตำแหน่งซึ่งอ้างอิงกับผู้สังเกตเรียบร้อยแล้ว ขั้นตอนต่อไปก็คือการแปลง View Coordinate มาเป็นพิกัด 2 มิติ ซึ่งสามารถนำมาวาดได้จริง ๆ บนจอภาพ ซึ่งโดยทั่วไปแล้วเราต้องการให้ได้ภาพที่สมจริง คือ วัตถุที่อยู่ไกลจากสายตาจะมีขนาดเล็กลง และวัตถุที่อยู่ใกล้สายตาจะมีขนาดใหญ่ขึ้น ซึ่งเรียกว่า Perspective Projection Transformation ซึ่งเป็นการสร้างภาพฉายมาตกกระทบฉากคล้าย ๆ กับการฉายภาพจากโปรเจคเตอร์

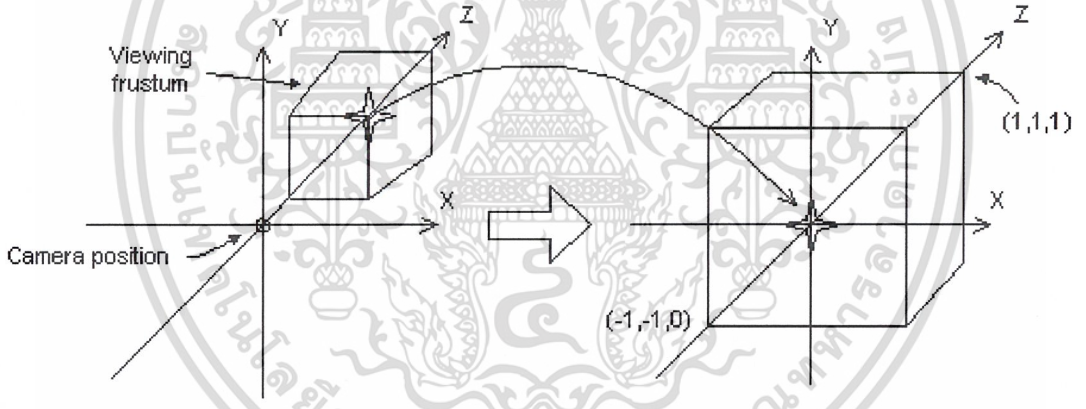


รูปที่ 2-7 Viewing Frustum

วัตถุที่อยู่ใน Viewing Frustum จะถูกนำมาแสดงผล เนื่องจากเป็นส่วนที่จอภาพสามารถเห็นได้ ส่วนวัตถุที่อยู่ภายนอก Viewing Frustum จะถูกคัดออกระหว่างการประมวลผลก่อนที่จะนำไปแสดงผล



รูปที่ 2-8 Viewing Frustum มองจากแนวแกน X



รูปที่ 2-9 การแปลงจาก Viewing Frustum (จากภาพด้านซ้าย) ไปเป็น cuboid shape (จากภาพด้านขวา) ซึ่งทำให้วัตถุที่อยู่ไกลมีขนาดเล็กลง และวัตถุที่อยู่ใกล้มีขนาดใหญ่ขึ้นตามสัดส่วนของระยะห่างจากสายตาผู้สังเกต

เราสามารถทำ Perspective Projection Transformation ได้โดยการใช้ Perspective Projection Matrix ดังนี้

$$\begin{bmatrix} w & 0 & 0 & 0 \\ 0 & h & 0 & 0 \\ 0 & 0 & Q & 1 \\ 0 & 0 & -QZ_n & 0 \end{bmatrix}$$

$$w = \cot\left(fov_w/2\right)$$

$$h = \cot\left(fov_h/2\right)$$

$$Q = \frac{Z_f}{Z_f - Z_n}$$

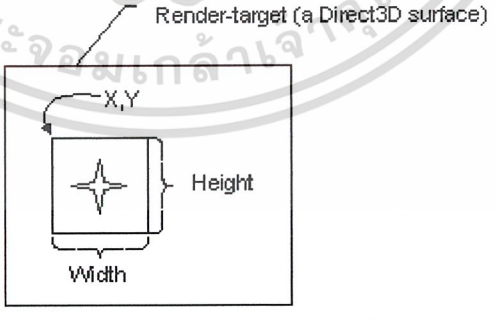
โดยทั่วไปแล้ว Graphics Engine นั้นมักจะมีฟังก์ชันที่ใช้สำหรับทำการคำนวณหา Perspective Projection Matrix โดยให้เรากำหนดค่า Field of View (fov) ตามแนวความสูงของฉาก (โดยทั่วไปมักเป็นแกน Y) ค่า Aspect Ratio (w/h) ค่า Near Clipping Plane (Z_n) และค่า Far Clipping Plane (Z_f)

นอกจากการทำ Perspective Projection Transformation แล้วยังมีการทำ Projection Transformation แบบอื่น ๆ อีก เช่น Isometric Projection Transformation, Oblique Projection Transformation เป็นต้น รายละเอียดสามารถหาอ่านเพิ่มเติมได้จากหนังสือพื้นฐานคอมพิวเตอร์พื้นฐานทั่ว ๆ ไปได้

2.1.5.4 Clipping and Viewport Scaling

Viewportในที่นี้หมายถึงพื้นที่สี่เหลี่ยมผืนผ้าที่ใช้ในการฉายภาพที่เกิดจากการทำ Projection Transformation ซึ่งเป็นขั้นตอนสุดท้ายใน Geometry Pipeline และทำให้ได้ภาพฉายตกลงบน Viewport เพื่อทำกระบวนการ Rasterization ต่อไปในการสร้างภาพเสมือนจริงบนหน้าจอ

Viewportอาจหมายถึงหน้าจอทั้งหน้าจอหรือบางส่วนของหน้าจอก็ได้ ทั้งนี้ขึ้นอยู่กับโปรแกรมซึ่งโดยทั่วไปแล้วเกมมักจะแสดงผลแบบเต็มจอ แต่การใช้ Viewport อาจมีประโยชน์เมื่อเราต้องการการแสดงผล 3 มิติหลายหน้าต่าง



รูปที่ 2-10 Direct3D Viewport

โดยทั่วไปแล้วเราจะทำการตัดทอน (Clipping) สิ่งที่มองไม่เห็นบนหน้าจอออกไป ในกรณีที่ Viewport แสดงถึงบางส่วนของหน้าจอนั้น เราจะตัดทอนส่วนของภาพที่อยู่นอกกรอบ Viewport ออกไป ทั้งนี้รวมถึงการตัดทอนในแนวแกน Z ด้วย (เราจะตัดทอนส่วนของภาพที่อยู่ไกลเกิน Viewport ที่กำหนด หรือส่วนของภาพที่อยู่ด้านหลัง Near Clipping Plane ของ Viewing Frustum ออก)

กระบวนการนี้โดยมากมักจัดการโดย Graphics Engine โดยเราเพียงแค่กำหนดค่าขอบเขตของ Viewport และระยะการตัดทอนตามแนวแกน Z ก็เพียงพอแล้ว

2.2 เทคนิคการตรวจสอบการชนสำหรับเกม 3 มิติ

ในการพัฒนาเกม 3 มิตินั้นปัญหาที่สำคัญอย่างหนึ่งคือ เราจะรู้ได้อย่างไรว่าวัตถุที่เคลื่อนไหวอยู่มันเคลื่อนไหวมาชนกัน ซึ่งถ้าจะใช้การตรวจสอบ Polygon Intersection จริง ๆ จะต้องใช้เวลาในการคำนวณอย่างมาก (Complexity ประมาณ $O(n^2)$ เมื่อ n เป็นจำนวนสามเหลี่ยมทั้งหมด) เป็นผลให้การแสดงการเคลื่อนไหวแบบเรียลไทม์ดูไม่สมจริง ซึ่งในทางปฏิบัติแล้วมักจะหลีกเลี่ยงการใช้วิธีนี้ แต่จะให้การคำนวณโดยประมาณเท่านั้น ซึ่งมีหลายวิธีดังต่อไปนี้

2.2.1 ใช้ทรงกลมครอบวัตถุ (Bounding Sphere)

วิธีนี้เราจะเอาทรงกลมมาครอบวัตถุไว้แล้วการเช็คว่าวัตถุชนกันรึเปล่าเราจะทำโดยการตรวจสอบว่าทรงกลมทั้งสองทับซ้อนกันรึเปล่าเท่านั้น ถ้าทรงกลมทั้ง 2 รูปที่ครอบวัตถุ 2 วัตถุทับซ้อนกันแล้วเราจะได้ว่าวัตถุทั้ง 2 มีโอกาสทับซ้อนกันด้วย (โดยประมาณ) ซึ่งจะทำให้การคำนวณเร็วขึ้นมาก แต่มีข้อเสียคือ ความละเอียดในการตรวจสอบจะค่อนข้างต่ำ แต่จะใช้ได้ดีกับวัตถุที่มีรูปร่างคล้ายทรงกลม ซึ่งมีวิธีการคำนวณดังต่อไปนี้

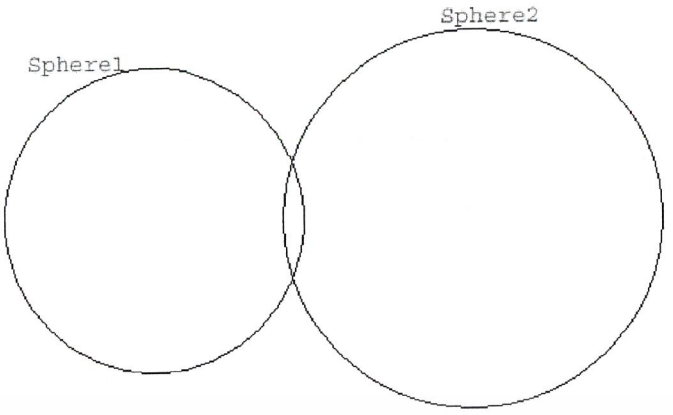
กำหนด

ทรงกลม 1 มีจุดศูนย์กลางอยู่ที่ (x_1, y_1, z_1) และมีรัศมี R_1

ทรงกลม 2 มีจุดศูนย์กลางอยู่ที่ (x_2, y_2, z_2) และมีรัศมี R_2

จะได้ว่า ทรงกลมทั้ง 2 รูปจะทับซ้อนกัน ก็ต่อเมื่อ

$$((x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2)^{1/2} < R_1 + R_2$$



รูปที่ 2-11 แสดงการทับซ้อนกันของทรงกลม

2.2.2 ใช้ทรงกระบอกครอบวัตถุ (Bounding Cylinder)

วิธีนี้เราจะใช้ทรงกระบอกมาครอบวัตถุ แล้วตรวจสอบว่าทรงกระบอกชนกันรึเปล่าแค่นั้น ซึ่งการคำนวณถือว่าเร็วมากเมื่อเทียบกับการตรวจสอบ Polygon Intersection แต่ถ้าจะให้ได้ผลดีต้องใช้กับวัตถุที่มีรูปร่างคล้ายทรงกระบอก ซึ่งมีวิธีการดังนี้

กำหนดให้ทรงกระบอกที่ 1มีจุดต่ำสุดอยู่ที่ $Y = Low_1$

มีจุดสูงสุดอยู่ที่ $Y = High_1$

วงกลมบนระนาบ XZ มีจุด Center อยู่ที่ x_1, z_1 และ รัศมี = R_1

กำหนดให้ทรงกระบอกที่ 2มีจุดต่ำสุดอยู่ที่ $Y = Low_2$

มีจุดสูงสุดอยู่ที่ $Y = High_2$

วงกลมบนระนาบ XZ มีจุด Center อยู่ที่ x_2, z_2 และ รัศมี = R_2

Pseudo Code แสดงขั้นตอนการคำนวณ

(กำหนดให้ทรงกระบอกทั้ง 2 รูปวางอยู่บนแนวระนาบเดียวกัน)

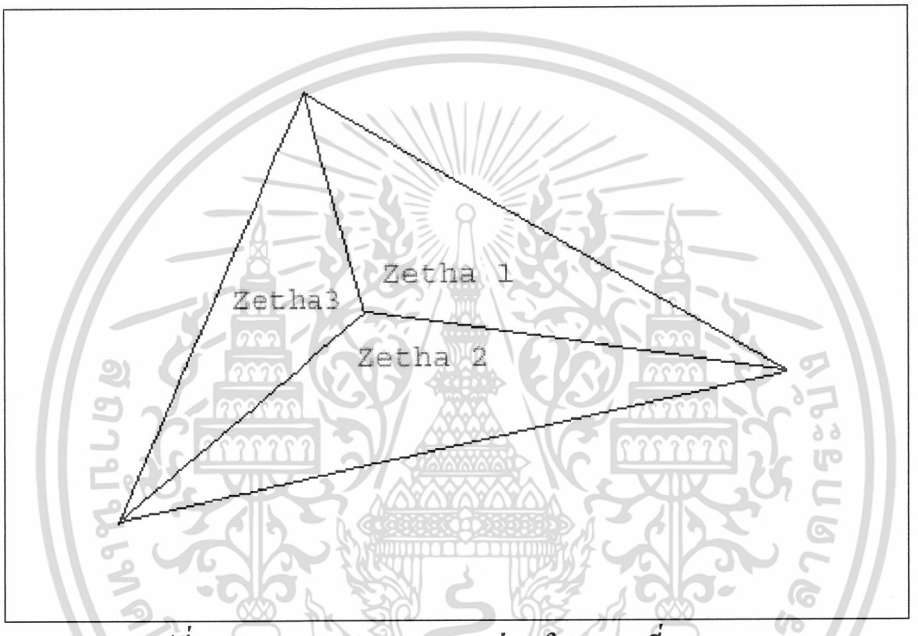
```
If (Min(High1, High2) and Max(Low1, Low2) is Intersect)
{
        If (Circle1 and Circle2 is Intersect)
        {
                Return True;
        }
}

Return False;
```

ในที่นี้จะไม่กล่าวถึงการตรวจสอบการชนระหว่าง ทรงกระบอกใด ๆ 2 รูป ที่อาจไม่ได้วางตัวอยู่บนแนวระนาบเดียวกัน

2.2.3 การหาความสูงของจุดบนสามเหลี่ยม

ในการสร้างเกม 3 มิตินั้น ก่อนที่เราจะวางวัตถุลงบนพื้นที่ที่เราต้องการ เราต้องรู้ความสูงของพื้นผิวนั้นก่อนว่าสูงจากระดับอ้างอิงมากน้อยเพียงใด เราถึงจะวางวัตถุลงบนพื้นที่นั้นได้โดยไม่จมลงใต้พื้นที่นั้น ๆ ซึ่งเราจะต้องรู้อีกว่าตอนนี้เราอยู่ที่สามเหลี่ยมไหนบนพื้นผิวที่ต้องการวางวัตถุ ดังนั้นตอนแรกเราต้องหาก่อนว่าตอนนี้เราอยู่ที่ สามเหลี่ยมไหนของพื้นผิว ซึ่งมีวิธีการดังต่อไปนี้

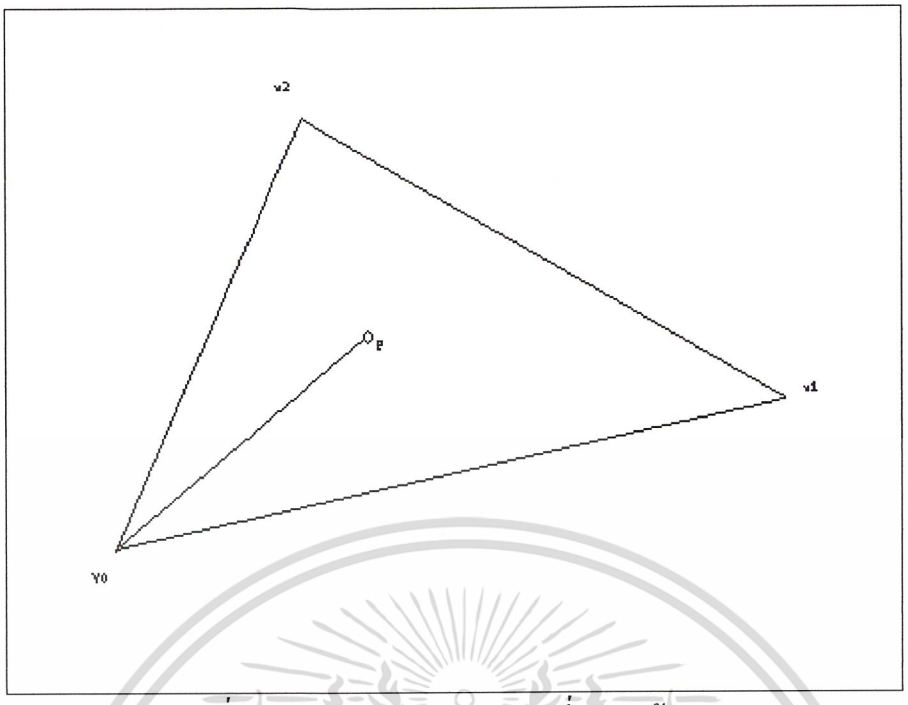


รูปที่ 2-12 แสดงการทดสอบการอยู่ภายในสามเหลี่ยมของจุด

ตอนแรกให้เราลากเส้นจากจุดที่เราอยู่ ไปที่จุดยอดทุกจุดยอดของสามเหลี่ยมนั้น แล้วก็หามุม $Zetha_1$, $Zetha_2$, $Zetha_3$ ดังรูป จุดที่เราอยู่จะอยู่ในสามเหลี่ยมก็ต่อเมื่อ

$$Zetha_1 + Zetha_2 + Zetha_3 = 360^{\circ}$$

เมื่อเรารู้แล้วว่าเราอยู่สามเหลี่ยมไหนแล้วตอนนี้เราก็สามารถดึงสมการระนาบของสามเหลี่ยมนั้นออกมา แล้วเราก็คำนวณหาค่าความสูง (ค่า y) โดยการส่ง Input เป็น x, z เข้าไปได้ เมื่อเรารู้ความสูงของพื้นที่ที่เราจะเอาไปวางแล้วตอนนี้เราก็สามารถเอาวัตถุที่เราต้องการไปวางบนพื้นที่นั้นได้แล้ว ซึ่งวิธีการดังกล่าวมีรายละเอียดดังนี้



รูปที่ 2-13 แสดงภาพฉายของสามเหลี่ยมจากด้านบน

V_0, V_1, V_2 เป็นจุดยอดทั้งสามของสามเหลี่ยม

P เป็นจุดที่เราอยู่

ขั้นแรกเราต้องหา Normal Vector ของสามเหลี่ยมก่อนซึ่งได้ดังนี้

Normal Vector = $(V_2 - V_0)X(V_1 - V_0)$; Cross Product

ต่อมาเราต้องหา Vector ที่ลากจากจุดยอดใดก็ได้ไปที่จุด P เช่น เลือกใช้ V_0 ซึ่งจะได้สมการดังนี้

Plane Vector = $P - V_0$ = $(P.x - V_0.x)i + (P.y - V_0.y)j + (P.z - V_0.z)k$

เนื่องจาก Vector ที่ตั้งฉากกัน Dot Product = 0 ดังนั้น

(Normal Vector)●(Plane Vector) = 0; Dot Product

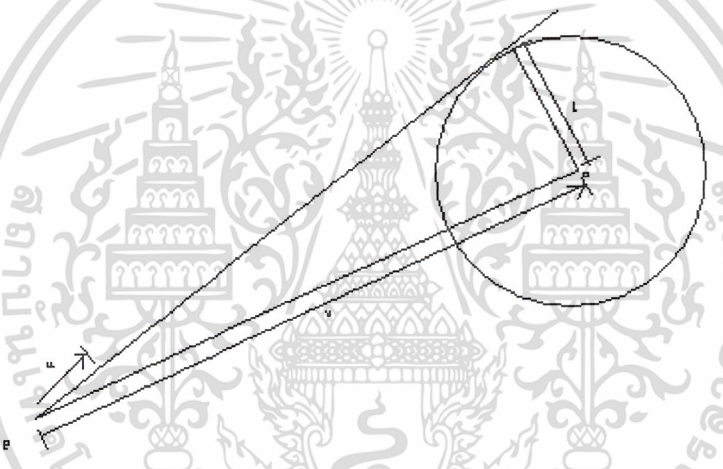
ดังนั้นเราจึงสามารถหาค่า p.y ได้จาก Input p.x และ p.z ซึ่ง p.y นั้นก็คือความสูงของจุดบนระนาบนั่นเอง

2.2.4 Ray Intersection Testing: Ray-Object Intersection

ในการพัฒนาเกมที่เป็นแนวยิงปืนนั้น เราจะต้องตรวจสอบว่ากระสุนที่ยิงออกไปนั้น ถูกเป้าหมายหรือไม่ และถูกส่วนไหนของเป้าหมาย ตัวอย่างของเกมประเภทนี้ได้แก่ Half-life: Counter Strike, Unreal ซึ่งเป็นเกมแนวยิงปืน โดยภายในเกมเหล่านี้ข้างในจะมีการคำนวณว่ากระสุนโดนส่วนไหนของวัตถุ ซึ่งวิธีการคำนวณจะใช้การตรวจสอบ Ray-Object Intersection ซึ่งมีรายละเอียดดังต่อไปนี้

2.2.4.1 Ray Intersection Testing: Ray-Sphere Intersection

วิธีนี้ใช้การตรวจสอบว่า Ray Vector ที่เรายิงออกไปนั้น Intersect ทรงกลมหรือไม่ ซึ่งความละเอียดในการตรวจสอบจะค่อนข้างต่ำถ้าเอามาใช้กับ Polygon ที่มีรูปทรงต่างจากทรงกลมมากแต่มีข้อดีคือการคำนวณเร็วกว่าการตรวจสอบ Ray-Polygon Intersection มากซึ่งมีวิธีการดังนี้



รูปที่ 2-14 การทดสอบการทับซ้อนของเส้นตรงกับทรงกลม

จากภาพ

P คือจุดเริ่มต้นของ Ray Vector

U คือ Unit Vector ซึ่งบอกทิศทางของ Ray Vector

$U_1 = U * |V|$

$V_1 = V - U_1$

$L = |V_1|$

Ray จะทับซ้อนกันกับทรงกลมก็ต่อเมื่อ L < รัศมีของทรงกลม

ซึ่งหลักการนี้ใช้ในการตรวจสอบการชนขั้นหยาบ ๆ เท่านั้น แต่เราอาจนำวิธีนี้ไปใช้กับ วัตถุที่มีรูปร่างคล้ายทรงกลมได้

2.2.4.2 Ray Intersection Testing: Ray-Polygon Intersection

วิธีนี้ใช้การตรวจสอบว่า Ray ที่ยิงออกไปนั้นทับซ้อนกับ Polygon หรือไม่ ซึ่ง Polygon นั้นจะประกอบไปด้วยสามเหลี่ยมหลาย ๆ รูป มาประกอบกันดังนั้น Ray นั้นทับซ้อนกับ Polygon ก็ต่อเมื่อมีการทับซ้อนระหว่าง Ray กับสามเหลี่ยมใด ๆ ใน Polygon นั้น ดังนั้นเราจะเริ่มจากการคำนวณว่า Ray นั้นทับซ้อนกับสามเหลี่ยมหรือไม่ ซึ่งมีรายละเอียดดังนี้

รูปที่ 2-15 การทดสอบการทับซ้อนของเส้นตรงกับสามเหลี่ยม

จากภาพ

P คือจุดเริ่มต้นของ Ray

U คือ Unit Vector ซึ่งบอกทิศทางของ Ray Vector

ซึ่งเส้นตรงที่ Ray ผ่านจะนิยามได้โดย

$$L(t) = P + U(t)$$

จุดใด ๆ บนสามเหลี่ยมนิยามได้โดย

$$T(t) = V_0 + (A*V_1 + B*V_2)$$

Ray จะทับซ้อนกับสามเหลี่ยมก็ต่อเมื่อสมการด้านล่างนี้เป็นจริง

$$L(t) = T(t)$$

ซึ่งขั้นตอนต่อไปเราจะต้องแก้สมการหาค่า t, A, B ซึ่งถ้า A + B <= 0.5 แล้ว จะได้ว่า Ray ทับซ้อนกับสามเหลี่ยม

ในการคำนวณหา Ray-Polygon Intersection นั้น เราจะต้องทำการวน Loop ทุก สามเหลี่ยมแล้ว ก็ใช้ตรวจสอบ Ray-Triangle Intersection ไปเรื่อยๆ ถ้าเจอ สามเหลี่ยมที่ Ray ทับซ้อนแสดงว่า Ray นั้นทับซ้อนกับ Polygon

ในการใช้งานจริง ๆ นั้น ทั้ง 2 วิธีต่างก็มีข้อดีและข้อเสียที่ต่างกัน ในส่วนของการหา Ray-Sphere Intersectionนั้นมีข้อดีคือเร็วเพราะมันไม่ต้องวนลูปทุกสามเหลี่ยม แต่มีข้อเสียคือความละเอียดในการตรวจสอบค่อนข้างต่ำส่วนการหา Ray-Polygon Intersection นั้นมีข้อดีคือตรวจสอบการชนได้ละเอียด คือตรวจสอบการทับซ้อนจริง ๆ ได้ แต่มีข้อเสียคือทำให้ใช้เวลาในการประมวลมาก ดังนั้นในการพัฒนาเกมจะใช้ข้อดีของทั้ง 2 วิธีมารวมกันคือ ตอนแรกเราจะตรวจสอบการชนด้วย Ray-Sphere Intersection ก่อนแล้ว ถ้าพบว่ามีการทับซ้อนเราถึงตรวจสอบการชนอย่างละเอียดด้วย Ray-Polygon Intersection ซึ่งจะทำให้สามารถประหยัดเวลาในการประมวลผลไปได้มาก

สำหรับการที่จะตรวจสอบว่า Ray ยิงโดนส่วนใดของ Polygon นั้น เราจำเป็นจะต้องมีสิ่งที่บอกว่าสามเหลี่ยมไหนคือส่วนไหน เช่น บอกว่าที่หัวของตัวละครมีสามเหลี่ยมใดอยู่บ้างเป็นต้น แล้วเวลาตรวจสอบการชนก็เอาข้อมูลนี้ไปใช้ก็ทำให้เรารู้ว่าเรายิงโดนส่วนไหนของวัตถุ ซึ่งบางครั้งเราก็ต้องสร้างเครื่องมือในพัฒนาขึ้นมาโดยเฉพาะสำหรับกำหนดข้อมูลที่บอกว่าส่วนของวัตถุส่วนนี้มีสามเหลี่ยมอะไรอยู่บ้าง

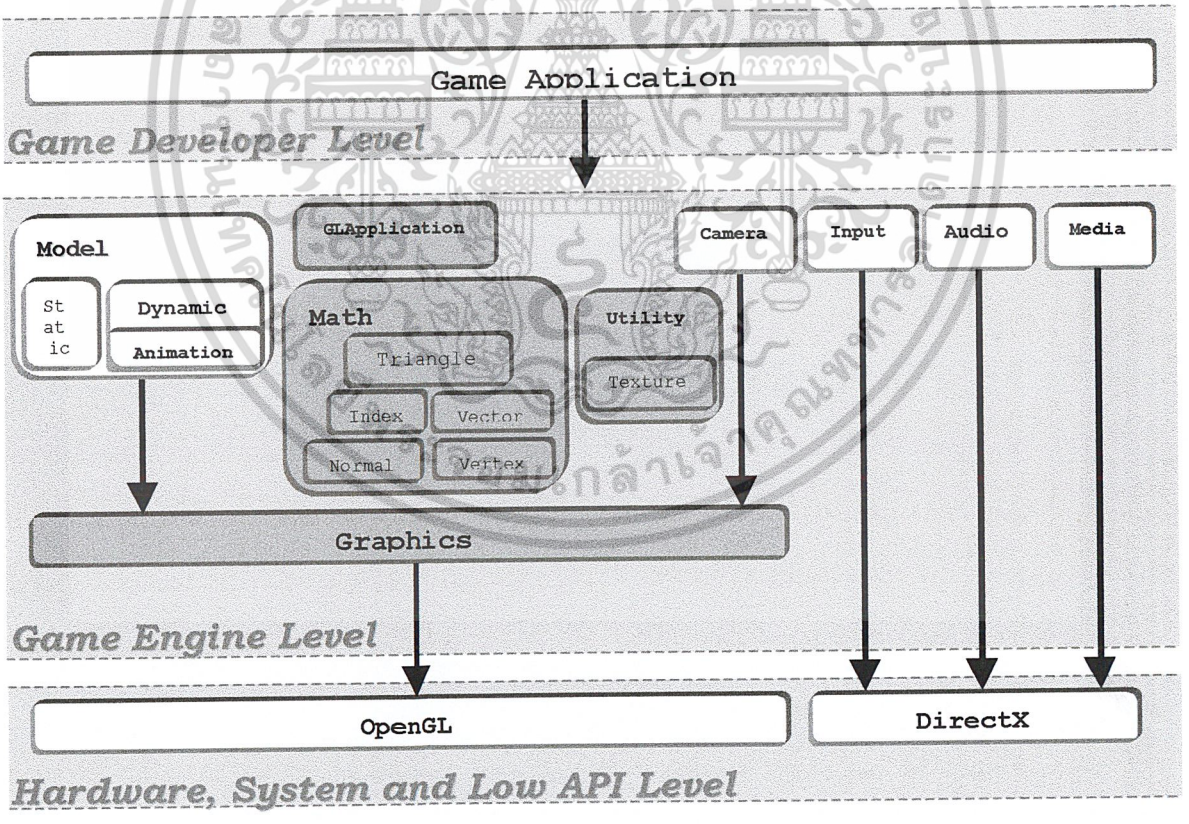
บทที่ 3
การออกแบบเอนจินต์สำหรับเกม 3 มิติ

3.1 โครงสร้างเกมเอนจินต์ (Engine Structure)

แนวคิดในการออกแบบเกมเอนจินต์ คือการช่วยให้ผู้พัฒนาเกมลดความซับซ้อนในการที่ต้องเขียนโปรแกรมเพื่อติดต่อกับ API ระดับล่างรวมถึงอุปกรณ์ฮาร์ดแวร์โดยตรง และเพิ่มฟังก์ชันที่จำเป็นต้องใช้ที่ยังไม่มีอยู่ใน API ของระบบ โครงสร้างของโปรแกรมที่พัฒนาโดยใช้เกมเอนจินต์แบ่งออกเป็น 3 ระดับ ได้แก่

- Game Developer Level
- Game Engine Level
- Hardware, System and Low API Level

โครงงานนี้จะพัฒนาในส่วนของ Game Developer Level และ Game Engine Level โดยมีโครงสร้างดังนี้



รูปที่ 3-1 แสดงโครงสร้างของเกมเอนจินต์

3.2 เอนจินต์คอมโพเนนต์ (Engine Component)

สำหรับรายละเอียดของเอนจินต์คอมโพเนนต์ต่างๆนั้นจะได้กล่าวถึงในบทที่ 4 ของปริญญานิพนธ์ เฉพาะในส่วนที่ได้รับผิดชอบเท่านั้น โดยคอมโพเนนต์ส่วนอื่นๆจะได้แสดงไว้ใน ปริญญานิพนธ์ เรื่อง การพัฒนาเกม 3 มิติ (ดร. วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษา, ปีการศึกษา 2544)

3.2.1 กราฟฟิก (Graphics)

กราฟฟิกเป็นส่วนที่ใช้ในการ Render วัตถุให้แสดงผลบนจอภาพ ซึ่งจะเป็น Layer ที่อยู่บน OpenGL API ใช้ในการจัดการ Initialize และ Cleanup ระบบ Graphics ของ OpenGL ในส่วนของ Graphics ซึ่งเราจะใช้คลาสนี้เป็นตัวแทน OpenGL Render Context ซึ่งประกอบไปด้วยคลาสหลักที่ใช้งานดังนี้

- GLGfx

  ใช้สำหรับการวาดภาพ 3 มิติ และฟังก์ชันช่วยเหลืออื่น ๆ ที่เกี่ยวกับระบบกราฟฟิก

3.2.2 คณิตศาสตร์ (Math)

Math เป็นส่วนที่จะใช้ในการจัดการเกี่ยวกับฟังก์ชันต่างๆ ทางคณิตศาสตร์ ซึ่งประกอบไปด้วยคลาสหลักที่ใช้งานดังนี้

- CMat

  เป็น class ที่ช่วยในการคำนวณการ Transformation เช่น Translation, Rotation, Scaling ซึ่งเป็น operation พื้นฐานสำหรับการประมวลผลทางด้าน Computer Graphics

- CMovMat

  ช่วยในการคำนวณและจัดเก็บข้อมูลที่เกี่ยวกับ Local Coordinate ของวัตถุ ช่วยอำนวยความสะดวกในการใช้งานมากกว่าคลาส CMat

- Vec3

  เป็น class vector 3 มิติ เพื่อใช้ในการคำนวณเกี่ยวกับตำแหน่งต่างๆของ พิกัดในระบบ 3 มิติ ตามทฤษฎีที่ได้กล่าวไว้ในบทที่ 2 ของปริญญานิพนธ์ฉบับนี้

3.2.3 ยูทิลิตี้ (Utilities)

Utilities เป็นส่วนที่จะใช้ในการจัดการส่วนอื่นประกอบในเกมนอกเหนือจากกลุ่มต่าง ๆ ที่ได้มีมาแล้ว ในส่วนของ Utilities นี้มีคลาสดังนี้

- Timer

  ประโยชน์หลัก ๆ ก็คือใช้สำหรับจับเวลาสำหรับการประมวลผลต่าง ๆ ที่ต้องใช้ภายในเกม

- Image

  ใช้สำหรับอ่านข้อมูลรูปภาพเข้ามาในหน่วยความจำโดยจะมีฟังก์ชันที่ใช้ในการโหลดภาพตระกูล Bitmap (.bmp) อยู่ ซึ่งเราสามารถนำข้อมูลนี้ไปใช้ในการทำเป็น Texture ต่อไป

- Tool

  Class ซึ่งจะประกอบไปด้วย ฟังก์ชันที่เติมเต็ม function ทางด้าน Math และการแสดงผลภาพต่าง ๆ ให้สามารถใช้งานเอนจินต์ได้สะดวกขึ้น

3.2.4 แอพพลิเคชันเฟรมเวิร์ค (Application Framework)

เอนจินต์เฟรมเวิร์คใช้สำหรับ อำนวยความสะดวกในการสร้างโปรแกรม และช่วยจัดการ Initialize และ Cleanup API ต่างๆ ที่อยู่ภายใต้เอนจินต์ให้พร้อมใช้งาน ซึ่งมีคลาสหลักที่ใช้งานดังนี้

- CGLApplication

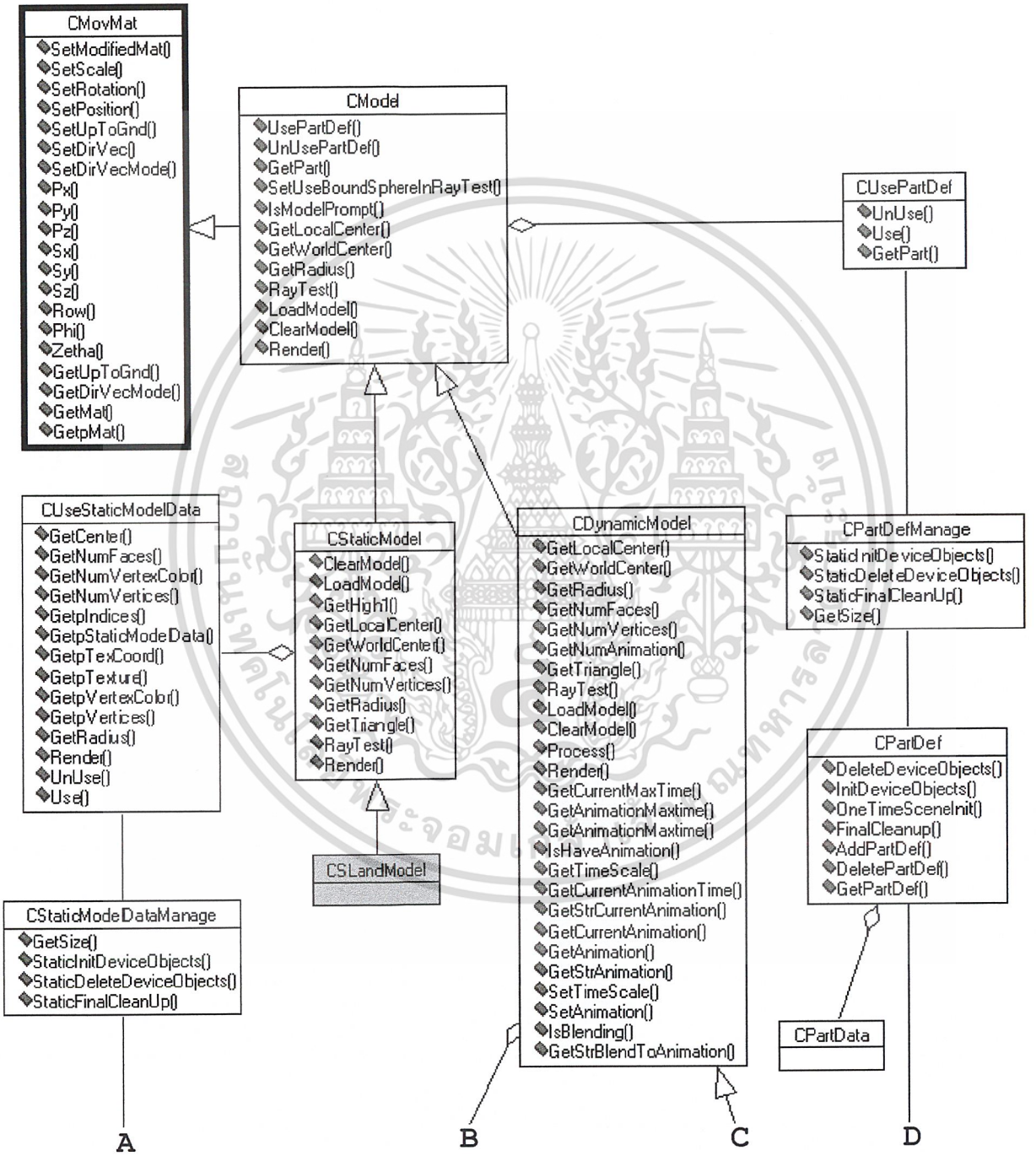
  เฟรมเวิร์คสำหรับผู้ใช้เกมเอนจินต์ ให้พร้อมใช้งาน โดยผู้พัฒนาเกมจะสืบทอดจากคลาส CGLApplication นี้ก่อนที่จะ Implement ในส่วนของผู้พัฒนาเอง

3.3 คลาสไดอะแกรมของเกมเอนจินท์ (Class Diagram of Game Engine)

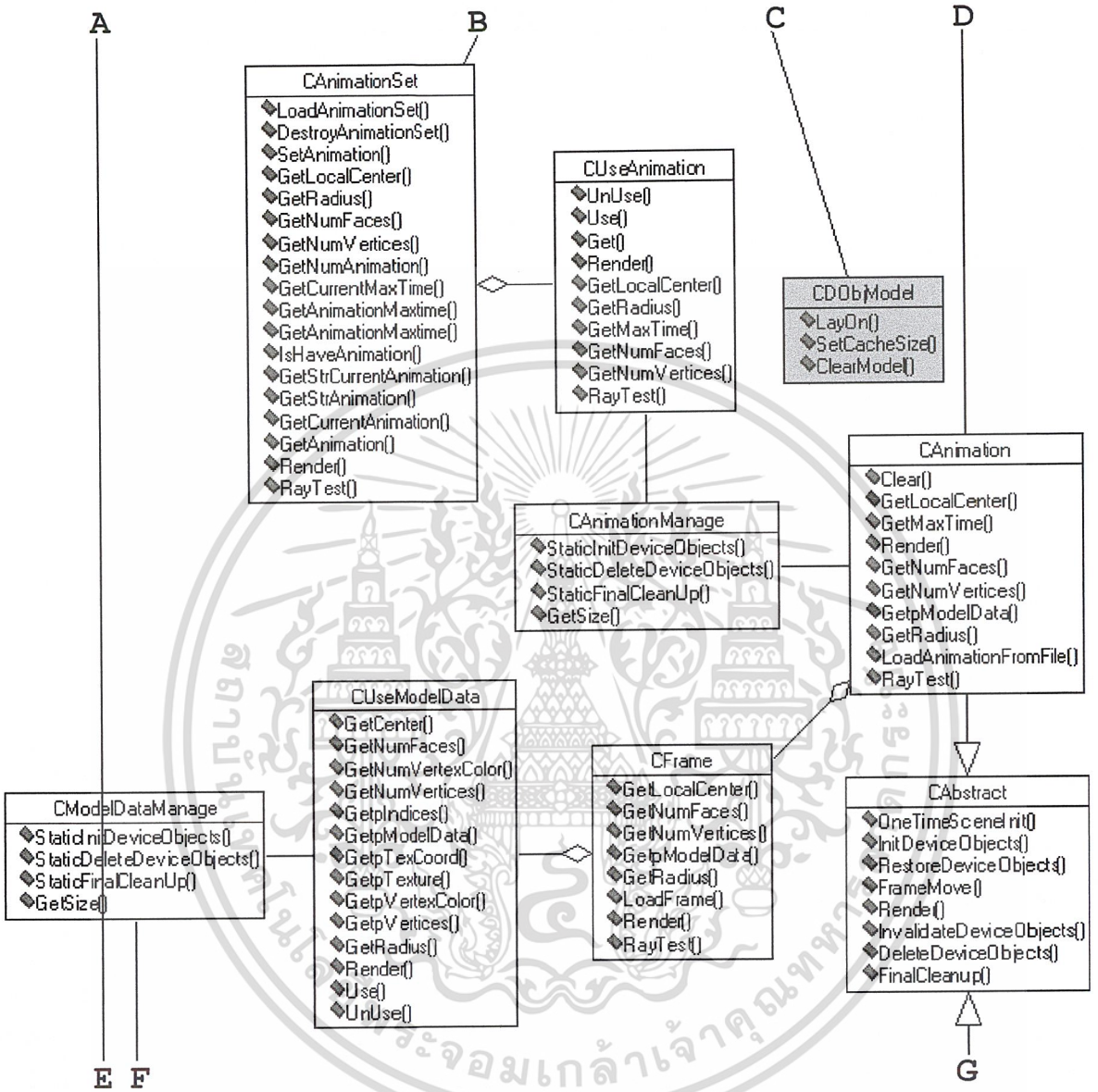
สัญลักษณ์ Class สี  หมายถึง Internal Class of Game Engine

สัญลักษณ์ Class สี  หมายถึง Interface Class of Developer

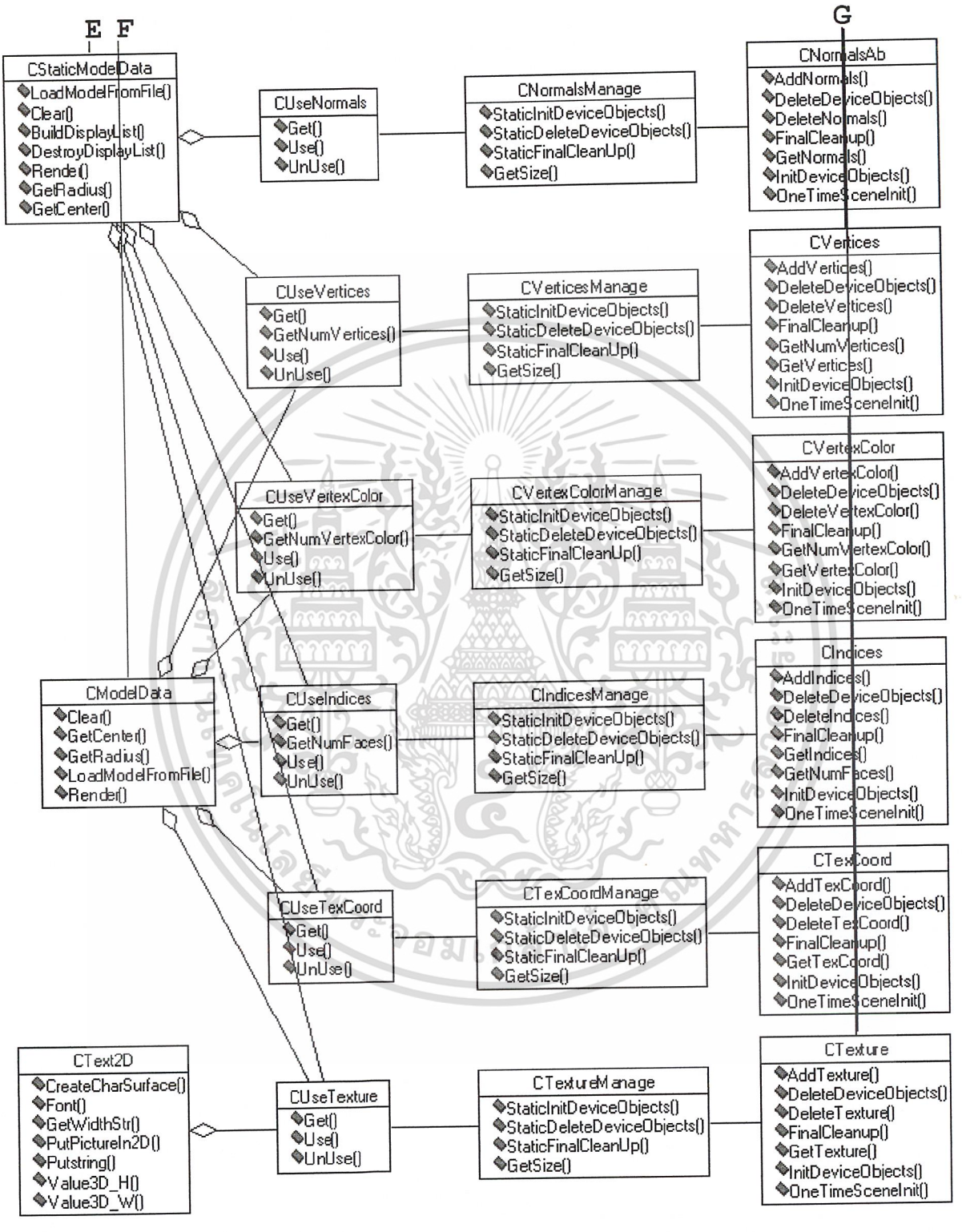
สัญลักษณ์เส้นทึบ  ล้อมรอบ หมายถึง Class ที่จะอธิบายในปริญญานิพนธ์ฉบับนี้



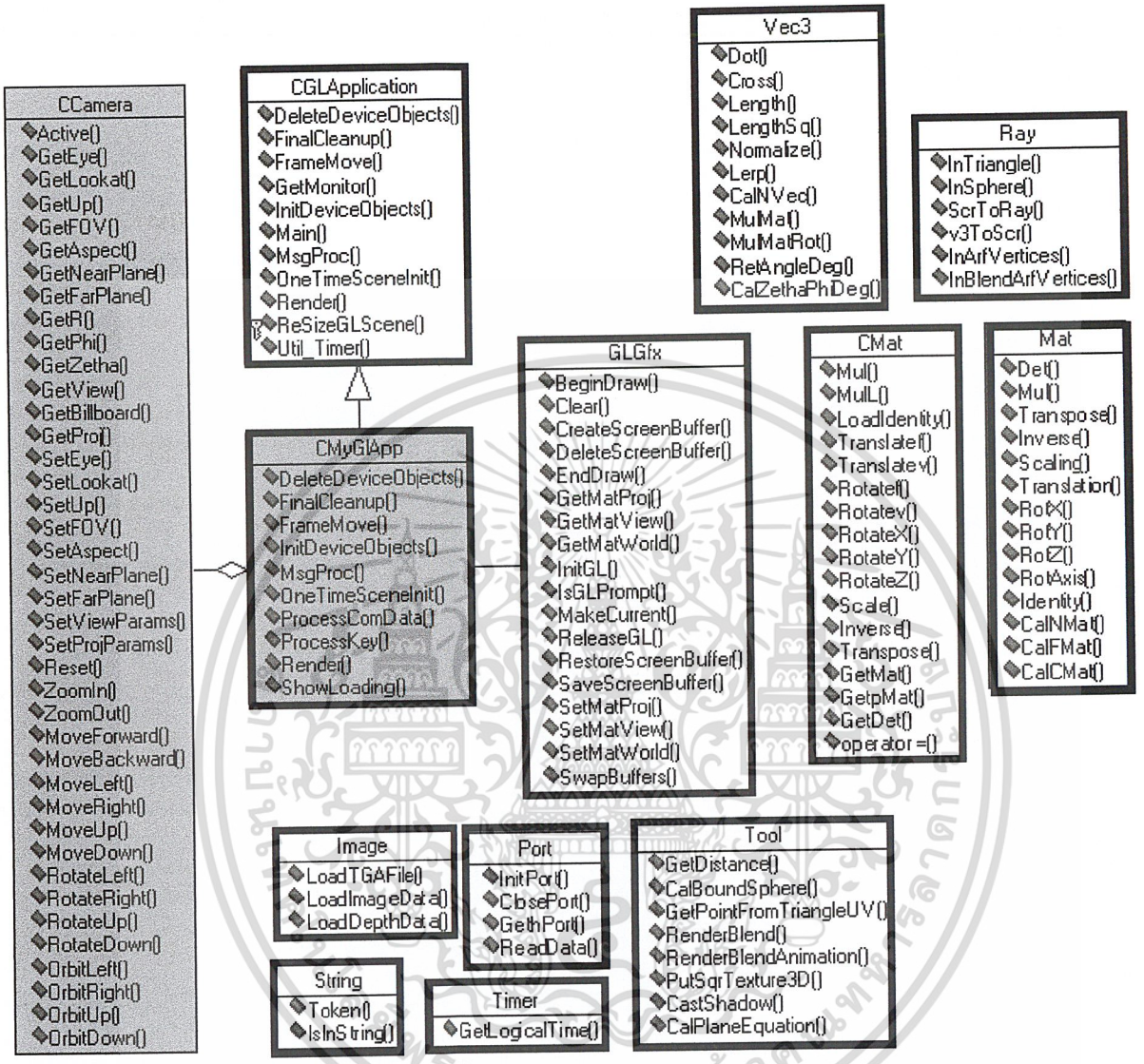
รูปที่ 3-2ก แสดง Class Diagram ของ Game Engine ส่วนที่ 1



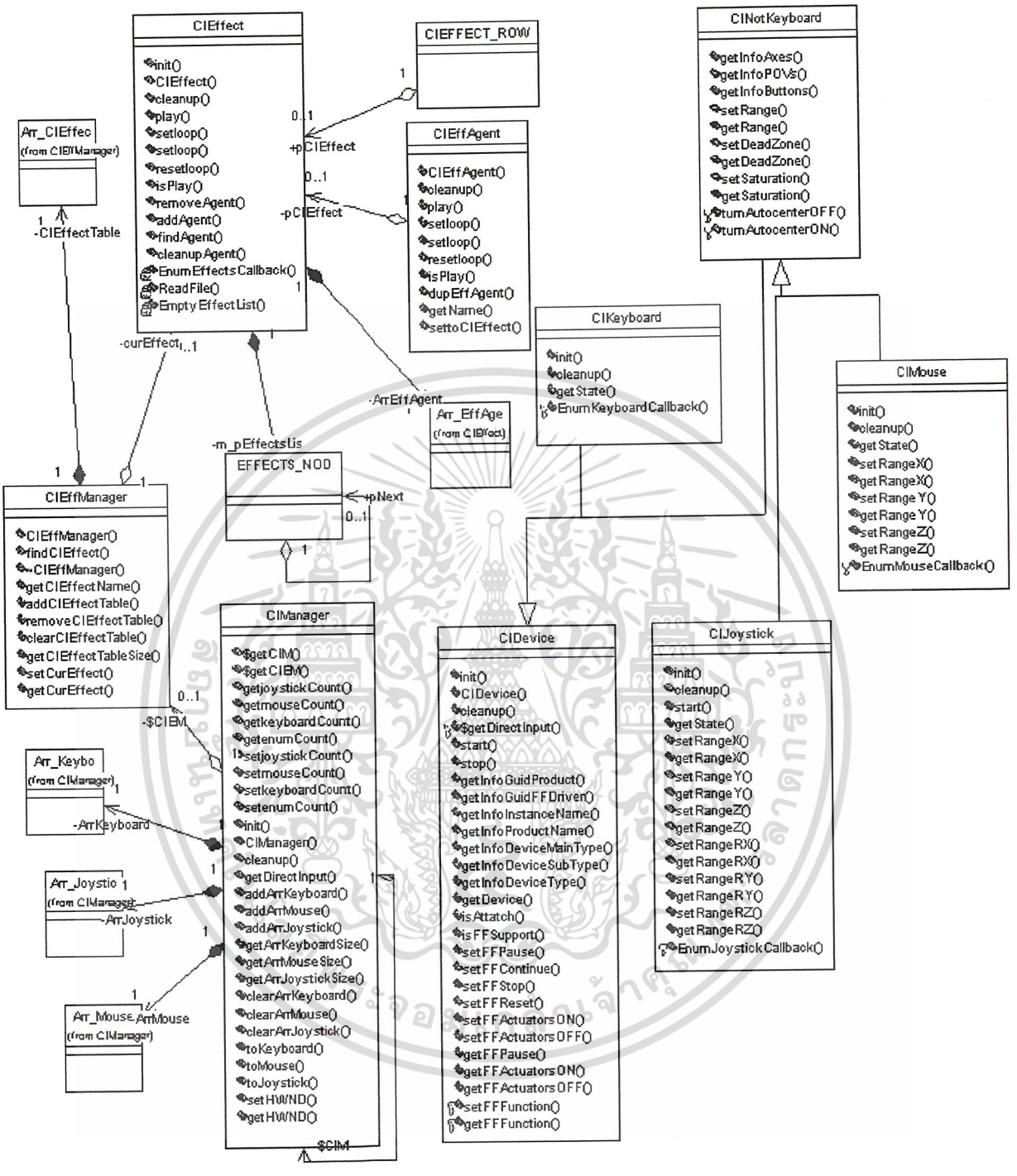
รูปที่3-2ข แสดง Class Diagram ของ Game Engine ส่วนที่ 2



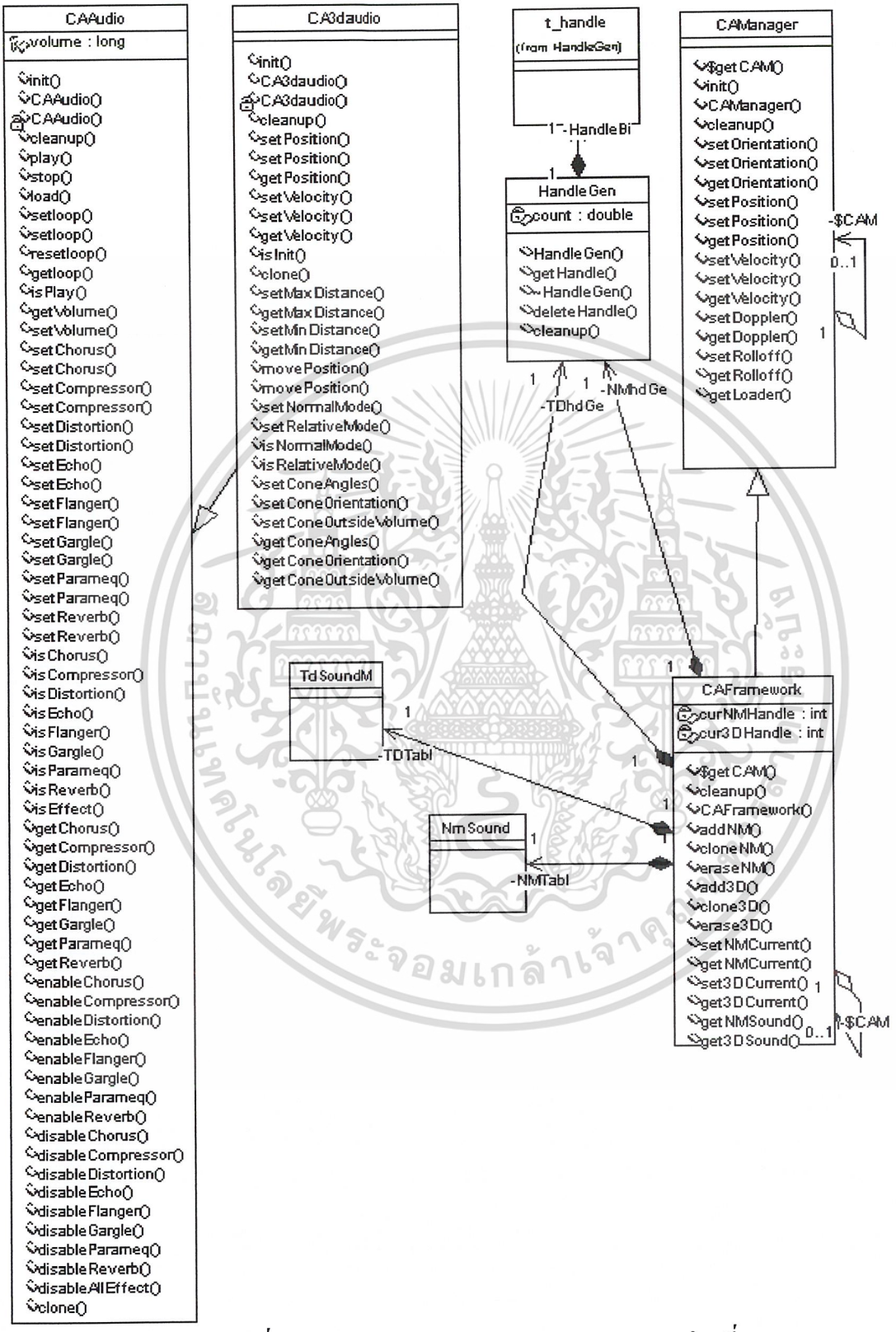
รูปที่3-2ค แสดง Class Diagram ของ Game Engine ส่วนที่ 3



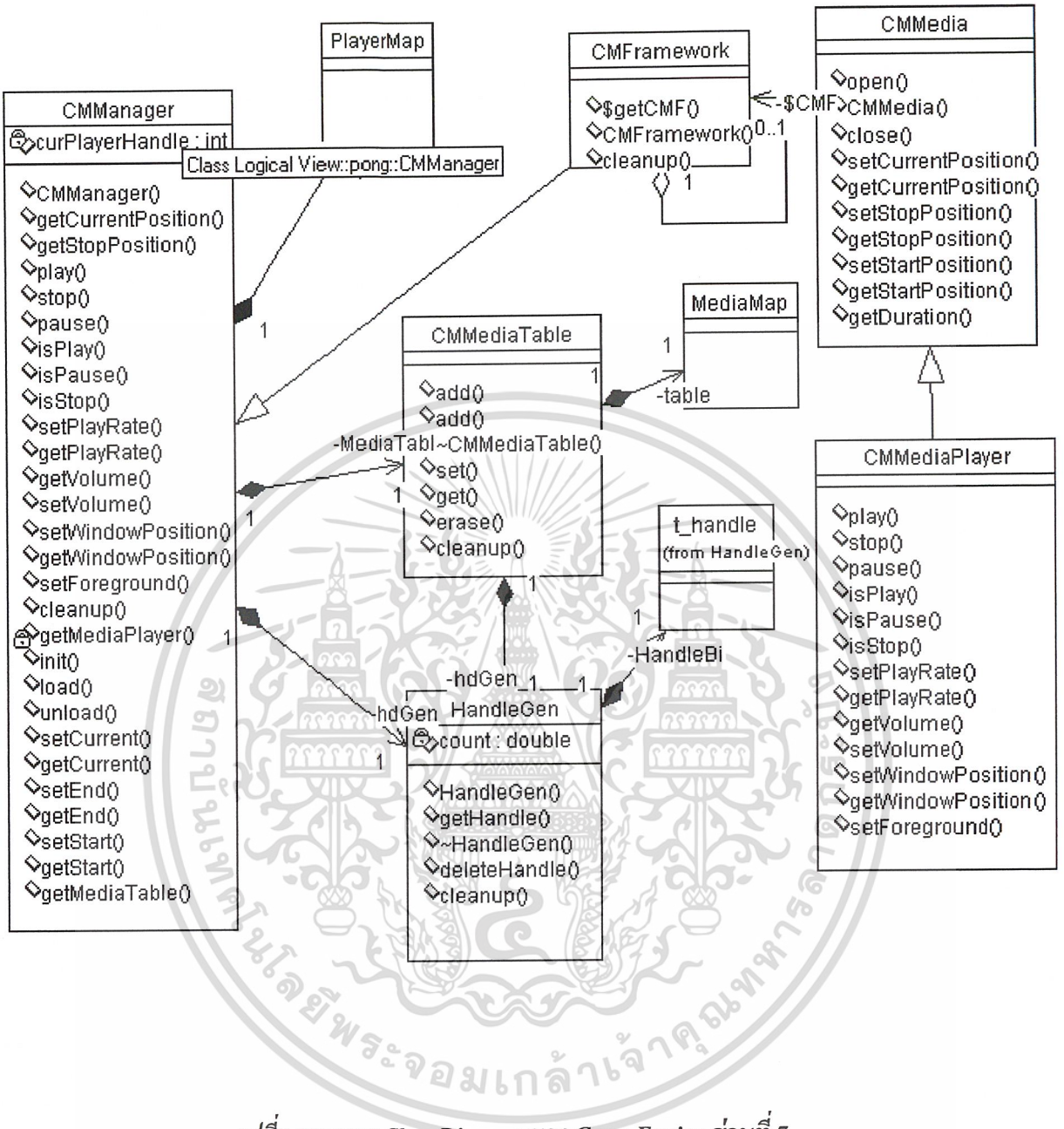
รูปที่3-2ง แสดง Class Diagram ของ Game Engine ส่วนที่ 4



รูปที่3-2จ แสดง Class Diagram ของ Game Engine ส่วนที่ 5



รูปที่3-2ฉ แสดง Class Diagram ของ Game Engine ส่วนที่ 6



รูปที่3-2ช แสดง Class Diagram ของ Game Engine ส่วนที่ 7

บทที่ 4
การพัฒนาเอนจินต์สำหรับเกม 3 มิติ

ในบทนี้เราจะกล่าวถึงรายละเอียดปลีกย่อยในบางส่วนของคลาสที่ประกอบขึ้นมาเป็นเอนจินต์ซึ่งผู้ใช้งานมักจะต้องติดต่อด้วยเป็นหลัก โดยจะอธิบายรายละเอียดเฉพาะในส่วนของ Graphics, Math Library, Utility และ Engine Framework อันประกอบไปด้วยคลาสต่างๆดังต่อไปนี้

4.1 กราฟฟิก (Graphics)

เอนจินต์ในส่วนกราฟฟิกนั้นเป็นส่วนที่ใช้ในการ Render วัตถุให้แสดงผลบนจอภาพ ซึ่ งจะเป็น Layer ที่อยู่บน OpenGL API ใช้ในการจัดการ Initialize และ Cleanup ระบบ Graphics ของ OpenGL ในส่วนของ Graphics ซึ่งเราจะใช้คลาสนี้เป็นตัวแทน OpenGL Render Context

เราเลือกออกแบบเอนจินต์โดยใช้ OpenGL API เนื่องจากเป็น Graphics API ที่นิยมใช้ตัวหนึ่ง และการใช้งานก็ไม่ยากเกินไปนัก อีกทั้งยังทำงานได้อย่างมีประสิทธิภาพ โดยคลาสที่ทำหน้าที่เป็นส่วนกราฟฟิกของเอนจินต์นั้นก็คือคลาส **GLGfx** ซึ่งมี Public Member Function ของคลาส GLGfx ดังนี้

HRESULT

GLGfx::InitGL(HWND hWnd, int colorBits = 16);

ใช้สร้าง OpenGL render context

Parameters:

hWnd - handleของ window ที่ใช้ในการ Render หรือ แสดงผลกราฟฟิก 2 มิติ / 3มิติ โดยใช้ OpenGL

colorBits - จำนวนบิตต่อพิกเซลของบัฟเฟอร์ที่ใช้ในการแสดงผลกราฟฟิกโดยใช้ OpenGL

Return Value:

ถ้ามีข้อผิดพลาดเกิดขึ้น ฟังก์ชันนี้จะ return ค่าติดลบออกมา เราสามารถตรวจสอบได้ว่าการสร้าง OpenGL render context สำเร็จหรือไม่

HRESULT

GLGfx::ReleaseGL();

ใช้คืน resource ที่เกี่ยวข้องกับ OpenGL และ OpenGL render context เมื่อเลิกใช้งาน

Parameters:

(none)

Return Value:

ถ้ามีข้อผิดพลาดเกิดขึ้น ฟังก์ชันนี้จะ return ค่าติดลบออกมา เราสามารถตรวจสอบได้ว่าการ cleanup OpenGL สำเร็จหรือไม่

HRESULT

GLGfx::Clear(GLbitfield mask = GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);

ใช้ลบ buffer ของ OpenGL เช่น color buffer หรือ depth buffer

Parameters:

mask - mask bit สำหรับบอก OpenGL ว่าต้องการ clear buffer ตัวใดบ้าง

Return Value:

ถ้ามีข้อผิดพลาดเกิดขึ้น ฟังก์ชันนี้จะ return ค่าติดลบออกมา เราสามารถตรวจสอบได้ว่าการติดต่อกับ OpenGL สำเร็จหรือไม่

HRESULT

GLGfx::BeginDraw(GLenum mode);

ใช้บอกให้ OpenGL เตรียมการจัดการ Geometry Pipeline

Parameters:

mode - polygon mode ที่ต้องการแสดงผล เช่น triangles, triangle fan, triangle strip เป็นต้น

Return Value:

ถ้ามีข้อผิดพลาดเกิดขึ้น ฟังก์ชันนี้จะ return ค่าติดลบออกมา เราสามารถตรวจสอบได้ว่าการติดต่อกับ OpenGL สำเร็จหรือไม่

HRESULT

GLGfx::EndDraw();

ใช้บอกให้ OpenGL เตรียมการส่งข้อมูลผ่าน Geometry Pipeline ไปยัง render buffer

Parameters:

(none)

Return Value:

ถ้ามีข้อผิดพลาดเกิดขึ้น ฟังก์ชันนี้จะ return ค่าติดลบออกมา เราสามารถตรวจสอบได้ว่าการติดต่อกับ OpenGL สำเร็จหรือไม่

BOOL

GLGfx::SwapBuffers(HDC hTargetDC = NULL);

ใช้บอกให้ OpenGL เตรียมการส่งข้อมูลจาก render buffer ไปวาดลงบนหน้าจอภายใน window ที่กำหนด โดยใช้ Device Context ของ window นั้น ๆ

Parameters:

HTargetDC - Device Context ของ windows ที่ต้องการนำภาพที่ประมวลผลโดย OpenGL อยู่ภายใน buffer มาวาดลง

Return Value:

ยังไม่มีการกำหนดความหมายของค่าที่ return ออกมา

BOOL

GLGfx::MakeCurrent(HDC hDC, HGLRC hRC);

ใช้เลือกให้ OpenGL render context ตัวใดตัวหนึ่งทำงานภายใน thread ปัจจุบัน

Parameters:

HTargetDC - Device Context ของ windows ที่ต้องการนำภาพที่ประมวลผลโดย OpenGL อยู่ภายใน buffer มาวาดลง

Return Value:

ถ้ามีข้อผิดพลาดเกิดขึ้น ฟังก์ชันนี้จะ return ค่า false เราสามารถตรวจสอบได้ว่าสามารถใช้งาน OpenGL ได้หรือไม่ ถ้าทุกอย่างปกติ ฟังก์ชันนี้จะ return ค่า true

BOOL

GLGfx::SetMatWorld(const D3DXMATRIX *pMatWorld);

ใช้ในการกำหนดค่า World Matrix ของ OpenGL geometry pipeline เพื่อนำไปใช้ในการทำ transformation

Parameters:

PMatWorld - World Matrix ที่ต้องการใช้ในการทำ Transformation

Return Value:

ฟังก์ชันนี้จะ return ค่า true เสมอ

BOOL

GLGfx::SetMatView (const D3DXMATRIX *pMatView);

ใช้ในการกำหนดค่า View Matrix ของ OpenGL geometry pipeline เพื่อนำไปใช้ในการ ทำ transformation

Parameters:

PMatView - View Matrix ที่ต้องการใช้ในการทำ Transformation

Return Value:

ฟังก์ชันนี้จะ return ค่า true เสมอ

BOOL

GLGfx::SetMatProj (const D3DXMATRIX *pMatProj);

ใช้ในการกำหนดค่า Projection Matrix ของ OpenGL geometry pipeline เพื่อนำไปใช้ในการทำ transformation

Parameters:

PMatProj - Projection Matrix ที่ต้องการใช้ในการทำ Transformation

Return Value:

ฟังก์ชันนี้จะ return ค่า true เสมอ

void

GLGfx::GetMatWorld(D3DXMATRIX* pMatWorld);

ใช้ในการตรวจสอบค่า World Matrix ของ OpenGL geometry pipeline ปัจจุบันในการทำ transformation

Parameters:

PMatWorld - World Matrix ที่ต้องการใช้ในการทำ Transformation

Return Value:

(none)

void

GLGfx::GetMatView (D3DXMATRIX* pMatView);

ใช้ในการตรวจสอบค่า View Matrix ของ OpenGL geometry pipeline ปัจจุบันในการ ทำ transformation

Parameters:

PMatView - View Matrix ที่ต้องการใช้ในการทำ Transformation

Return Value:

(none)

void

GLGfx::GetMatProj (D3DXMATRIX* pMatProj);

ใช้ในการตรวจสอบค่า Projection Matrix ของ OpenGL geometry pipeline ปัจจุบันในการ ทำ transformation

Parameters:

PMatProj - Projection Matrix ที่ต้องการใช้ในการทำ Transformation

Return Value:

(none)

BOOL

GLGfx::IsGLPrompt();

ใช้ตรวจสอบสถานะปัจจุบันของ OpenGL

Parameters:

(none)

Return Value:

return false ถ้า OpenGL ไม่อยู่ในสถานะที่ใช้งานได้ นอกจากนี้จะ return true

4.2 ฟังก์ชันการคำนวณ (Math Library)

Math เป็นส่วนที่จะใช้ในการจัดการเกี่ยวกับฟังก์ชันต่างๆ ทางคณิตศาสตร์ มีคลาสต่าง ๆ ดังนี้

CMat

เป็น class ที่ช่วยในการคำนวณการ Transformation เช่น Translation, Rotation, Scaling ซึ่งเป็น operation พื้นฐานสำหรับการประมวลผลทางด้าน Computer Graphics มี member function ดังนี้

void

CMat::Mul(const CMat *pcMat);

ใช้คูณ Matrix ที่เข้ามาเป็นพจน์ขวาของ CMat Object และให้ CMat Objectเก็บผลลัพธ์เอาไว้ โดยทำงานตามสมการ (*this) = (*this) * (*pcMat);

Parameters:

pcMat - Matrix ที่เป็นตัวคูณร่วม

Return Value:

(none)

void

CMat::Mul(const D3DXMATRIX *pMat);

ใช้คูณ Matrix ที่เข้ามาเป็นพจน์ขวาของ CMat Object และให้ CMat Objectเก็บผลลัพธ์ เอาไว้โดยทำงานตามสมการ (*this) = (*this) * (*pMat);

Parameters:

pMat - Matrix ที่เป็นตัวคูณร่วม

Return Value:

(none)

void

CMat::MulL(const CMat *pcMat);

ใช้คูณ Matrix ที่เข้ามาเป็นพจน์ซ้ายของ CMat Object และให้ CMat Objectเก็บผลลัพธ์เอาไว้ โดยทำงานตามสมการ (*this) = (*pcMat) * (*this);

Parameters:

pcMat - Matrix ที่เป็นตัวคูณร่วม

Return Value:

(none)

void

CMat::MulL(const D3DXMATRIX *pMat);

ใช้คูณ Matrix ที่เข้ามาเป็นพจน์ซ้ายของ CMat Object และให้ CMat Objectเก็บผลลัพธ์ เอาไว้โดยทำงานตามสมการ (*this) = (*pMat) * (*this);

Parameters:

pMat - Matrix ที่เป็นตัวคูณร่วม

Return Value:

(none)

void

CMat::LoadIdentity();

ใช้ปรับค่า Matrix ให้เป็น Matrix เอกลักษณ์

Parameters:

(none)

Return Value:

(none)

void

CMat::Translatef(float x, float y, float z);

ใช้ปรับค่า Matrix ให้เป็น Translation Matrix

Parameters:

x - ค่าการย้ายตำแหน่งในแนวแกน X

y - ค่าการย้ายตำแหน่งในแนวแกน Y

z - ค่าการย้ายตำแหน่งในแนวแกน Z

Return Value:

(none)

void

CMat::Translatev(const D3DXVECTOR3 *pvTran);

ใช้ปรับค่า Matrix ให้เป็น Translation Matrix

Parameters:

pvTran - ค่าการย้ายตำแหน่ง แทนข้อมูลด้วยเวคเตอร์ 3 มิติ

Return Value:

(none)

void

CMat::Rotatef(float x, float y, float z, float AngleDeg);

ใช้ปรับค่า Matrix ให้เป็น Rotation Matrix

Parameters:

x - ค่าเวคเตอร์แกนหมุนในแนวแกน X

y - ค่าเวคเตอร์แกนหมุนในแนวแกน Y

z - ค่าเวคเตอร์แกนหมุนในแนวแกน Z

AngleDeg - ค่ามุมที่ต้องการหมุนในหน่วยองศา

Return Value:

(none)

void

CMat::Rotatev(const D3DXVECTOR3 *pvAxis, float AngleDeg);

ใช้ปรับค่า Matrix ให้เป็น Rotation Matrix

Parameters:

pvAxis - ค่าเวคเตอร์แกนหมุน แทนข้อมูลด้วยเวคเตอร์ 3 มิติ

AngleDeg - ค่ามุมที่ต้องการหมุนในหน่วยองศา

Return Value:

(none)

void

CMat::RotateX(float AngleDeg);

ใช้ปรับค่า Matrix ให้เป็น Rotation Matrix แทนการหมุนในแนวแกน X

Parameters:

AngleDeg - ค่ามุมที่ต้องการหมุนในหน่วยองศา

Return Value:

(none)

void

CMat::RotateY(float AngleDeg);

ใช้ปรับค่า Matrix ให้เป็น Rotation Matrix แทนการหมุนในแนวแกน Y

Parameters:

AngleDeg - ค่ามุมที่ต้องการหมุนในหน่วยองศา

Return Value:

(none)

void

CMat::RotateZ(float AngleDeg);

ใช้ปรับค่า Matrix ให้เป็น Rotation Matrix แทนการหมุนในแนวแกน Z

Parameters:

AngleDeg - ค่ามุมที่ต้องการหมุนในหน่วยองศา

Return Value:

(none)

void

CMat::Scale(float sx, float sy, float sz);

ใช้ปรับค่า Matrix ให้เป็น Scaling Matrix

Parameters:

sx - ค่าการขยายในแนวแกน X

sy - ค่าการขยายในแนวแกน Y

sz - ค่าการขยายในแนวแกน Z

Return Value:

(none)

void

CMat::Inverse();

ใช้ปรับค่า Matrix ให้เป็น Inverse Matrix จากค่าปัจจุบัน

Parameters:

(none)

Return Value:

(none)

void

CMat::Transpose();

ใช้ปรับค่า Matrix ให้เป็น Transpose Matrix จากค่าปัจจุบัน

Parameters:

(none)

Return Value:

(none)

float

CMat::GetDet() const;

ใช้คำนวณหาดีเทอร์มิแนนต์ของ Matrix

Parameters:

(none)

Return Value:

ค่า Determinant ของ CMat Object

CMovMat

ช่วยในการคำนวณและจัดเก็บข้อมูลที่เกี่ยวกับ Local Coordinate ของวัตถุ ช่วยอำนวยความสะดวกในการใช้งานมากกว่าคลาส CMat มี member function ดังนี้

HRESULT

CMovMat::SetScale(float Sx, float Sy, float Sz);

ใช้ปรับค่าอัตราส่วนขยายของวัตถุ

Parameters:

Sx - ค่าการขยายในแนวแกน X

Sy - ค่าการขยายในแนวแกน Y

Sz - ค่าการขยายในแนวแกน Z

Return Value:

ฟังก์ชันนี้ return ค่า S_OK เสมอ

HRESULT

CMovMat::SetRotation(float Row, float Phi, float Zetha);

ใช้ปรับค่าการหมุนรอบแนวแกน X, Y, Z ตามลำดับ

Parameters:

Row - ค่าการหมุนรอบแนวแกน X

Phi - ค่าการหมุนรอบแนวแกน Y

Zetha - ค่าการหมุนรอบแนวแกน Z

Return Value:

ฟังก์ชันนี้ return ค่า S_OK เสมอ

HRESULT

CMovMat::SetPosition(float Px, float Py, float Pz);

ใช้ปรับตำแหน่งของวัตถุ

Parameters:

Px - ค่าการย้ายตำแหน่งในแนวแกน X

Py - ค่าการย้ายตำแหน่งในแนวแกน Y

Pz - ค่าการย้ายตำแหน่งในแนวแกน Z

Return Value:

ฟังก์ชันนี้ return ค่า S_OK เสมอ

void

CMovMat::SetUpToGnd(float fUpToGnd);

ใช้ปรับค่าการวางตำแหน่งสูงจากพื้น

Parameters:

fUpToGnd - ค่าระดับความสูง

Return Value:

(none)

HRESULT

CMovMat::SetDirVec(const D3DXVECTOR3* pVDir);

ใช้ปรับทิศทางของวัตถุ

Parameters:

PVDir - เวคเตอร์ทิศทางของวัตถุ

Return Value:

ฟังก์ชันนี้ return ค่า S_OK เสมอ

void

CMovMat::SetDirVecMode(int nDirMode);

ใช้ปรับโหมดการปรับทิศทาง

Parameters:

nDirMode - โหมดทิศทางของวัตถุ มีค่าดังนี้

1. DV_UNUSE, // Default +Y is Up(Not Use m_DirVec)
2. DV_FRONTVEC, // m_DirVec is Front
3. DV_NORMALVEC // m_DirVec is Normal(Up)

Return Value:

(none)

float

CMovMat::Px();

ใช้ตรวจสอบค่าการย้ายตำแหน่งตามแนวแกน X

Parameters:

(none)

Return Value:

return ค่าการย้ายตำแหน่งตามแนวแกน X

float

CMovMat::Py();

ใช้ตรวจสอบค่าการย้ายตำแหน่งตามแนวแกน Y

Parameters:

(none)

Return Value:

return ค่าการย้ายตำแหน่งตามแนวแกน Y

float

CMovMat::Pz();

ใช้ตรวจสอบค่าการย้ายตำแหน่งตามแนวแกน Z

Parameters:

(none)

Return Value:

return ค่าการย้ายตำแหน่งตามแนวแกน Z

float

CMovMat::Sx();

ใช้ตรวจสอบค่าอัตราส่วนขยายตามแนวแกน X

Parameters:

(none)

Return Value:

return ค่าอัตราส่วนขยายตามแนวแกน X

float

CMovMat::Sy();

ใช้ตรวจสอบค่าอัตราส่วนขยายตามแนวแกน Y

Parameters:

(none)

Return Value:

return ค่าอัตราส่วนขยายตามแนวแกน Y

float

CMovMat::Sz();

ใช้ตรวจสอบค่าอัตราส่วนขยายตามแนวแกน Z

Parameters:

(none)

Return Value:

return ค่าอัตราส่วนขยายตามแนวแกน Z

float

CMovMat::Row();

ใช้ตรวจสอบค่าการหมุนรอบแนวแกน X

Parameters:

(none)

Return Value:

return ค่าการหมุนรอบแนวแกน X

float

CMovMat::Phi();

ใช้ตรวจสอบค่าการหมุนรอบแนวแกน Y

Parameters:

(none)

Return Value:

return ค่าการหมุนรอบแนวแกน Y

float

CMovMat::Zetha();

ใช้ตรวจสอบค่าการหมุนรอบแนวแกน Z

Parameters:

(none)

Return Value:

return ค่าการหมุนรอบแนวแกน Z

float

CMovMat::GetUpToGnd();

ใช้ตรวจสอบค่าการวางตำแหน่งสูงจากพื้น

Parameters:

(none)

Return Value:

return ค่าระดับความสูง

int

CMovMat::GetDirVecMode();

ใช้ตรวจสอบโหมดการปรับทิศทาง

Parameters:

(none)

Return Value:

โหมดทิศทางของวัตถุ มีค่าดังนี้

1. DV_UNUSE, // Default +Y is Up(Not Use m_DirVec)
2. DV_FRONTVEC, // m_DirVec is Front
3. DV_NORMALVEC // m_DirVec is Normal(Up)

Vec3

เป็น class vector 3 มิติ เพื่อใช้ในการคำนวณเกี่ยวกับตำแหน่งต่างๆของ พิกัดในระบบ 3 มิติ ตามทฤษฎีที่ได้กล่าวไว้ในบทที่ 2 ของปริญญานิพนธ์ฉบับนี้ โดย class นี้มี member function ดังนี้

float

Vec3::Dot(const D3DXVECTOR3* pV1, const D3DXVECTOR3* pV2);

ใช้คำนวณค่า dot product

Parameters:

pV1 - เวคเตอร์ 1

pV2 - เวคเตอร์ 2

Return Value:

return ค่า (*pV1) dot (*pV2)

D3DXVECTOR3*

Vec3::Cross(D3DXVECTOR3* pOut, const D3DXVECTOR3* pV1, const D3DXVECTOR3* pV2);

ใช้คำนวณค่า cross product

Parameters:

pOut - เวคเตอร์ผลลัพธ์

pV1 - เวคเตอร์ 1

pV2 - เวคเตอร์ 2

Return Value:

return ค่า (*pOut) = (*pV1) cross (*pV2)

float

Vec3::Length(const D3DXVECTOR3* pV);

ใช้คำนวณค่า magnitude ของเวคเตอร์

Parameters:

pV - เวคเตอร์ที่ต้องการคำนวณค่า

Return Value:

return ค่า |(*pV)|

float

Vec3::LengthSq(const D3DXVECTOR3* pV);

ใช้คำนวณค่ากำลังสองของ magnitude ของเวคเตอร์

Parameters:

pV - เวคเตอร์ที่ต้องการคำนวณค่า

Return Value:

return ค่า $|(*pV)|^2$

D3DXVECTOR3*

Vec3::Normalize(D3DXVECTOR3* pOut, CONST D3DXVECTOR3* pV);

ใช้คำนวณ normalize vector

Parameters:

pOut - เวคเตอร์ผลลัพธ์

pV - เวคเตอร์ที่ต้องการคำนวณค่า

Return Value:

return ค่า (*pOut) = (*pV) / |(*pV)|

D3DXVECTOR3*

Vec3::Lerp(

D3DXVECTOR3* pOut,

const D3DXVECTOR3* pV1,

const D3DXVECTOR3* pV2,

float s

);

ใช้คำนวณค่า linear interpolation ระหว่างสองเวคเตอร์

Parameters:

pOut - เวคเตอร์ผลลัพธ์

pV1 - เวคเตอร์ 1

pV2 - เวคเตอร์ 2

Return Value:

return ค่า (*pOut) = (s * (*pV2)) + ((1-s) * (*pV1))

HRESULT

Vec3::CalNVec(

D3DXVECTOR3* pVOut,

const D3DXVECTOR3* v0,

const D3DXVECTOR3* v1,

const D3DXVECTOR3* v2

);

ใช้คำนวณค่า normal vector ของหน้าสามเหลี่ยม

Parameters:

pOut - เวคเตอร์ผลลัพธ์

v0 - เวคเตอร์ 0

v1 - เวคเตอร์ 1

v2 - เวคเตอร์ 2

Return Value:

return ค่า (*pOut) = ((*v1) - (*v0)) cross ((*v2) - (*v0))

HRESULT

Vec3::MulMat(

D3DXVECTOR3* pVOut,

const D3DXVECTOR3* pVIn,

const D3DXMATRIX* pMat

);

ใช้คำนวณค่า vector หลังการทำ homogeneous transformation

Parameters:

pVOut - เวคเตอร์ผลลัพธ์

pVIn - เวคเตอร์ที่ต้องการคำนวณค่า

pMat - Transformation Matrix

Return Value:

return ค่า (*pVOut) = (*pVIn) X ($*pMat_{4x4}$)

HRESULT

Vec3::MulMatRot(

D3DXVECTOR3* pVOut,

const D3DXVECTOR3* pVIn,

const D3DXMATRIX* pMat

);

ใช้คำนวณค่า vector หลังการทำ non-homogeneous transformation

Parameters:

pVOut - เวคเตอร์ผลลัพธ์

pVIn - เวคเตอร์ที่ต้องการคำนวณค่า

pMat - Transformation Matrix

Return Value:

return ค่า (*pVOut) = (*pVIn) X ($*pMat_{3x3}$)

double

Vec3::RetAngleDeg(const D3DXVECTOR3* pV1, const D3DXVECTOR3* pV2);

ใช้คำนวณค่ามุมระหว่างสองเวคเตอร์

Parameters:

pV1 - เวคเตอร์ 1

pV2 - เวคเตอร์ 2

Return Value:

return ค่ามุมระหว่างสองเวคเตอร์

HRESULT

Vec3::CalZethaPhiDeg(const D3DXVECTOR3* pV, double* Zetha, double* Phi);

ใช้คำนวณค่ามุมของเวคเตอร์ในระบบพิกัดทรงกลม

Parameters:

pV - เวคเตอร์ที่ต้องการคำนวณค่า

Zetha - มุม Zetha

Phi - มุม Phi

Return Value:

return S_OK เสมอ

Ray

เป็น class เพื่อช่วยในการคำนวณหา Collision ระหว่าง เส้นตรง กับ Polygon ตามทฤษฎีที่ได้กล่าวไว้ในบทที่ 2 ของปริญญานิพนธ์ฉบับนี้ โดย class นี้มี member function ดังนี้

BOOL

Ray::InTriangle(

const D3DXVECTOR3& orig, const D3DXVECTOR3& dir,

D3DXVECTOR3& v0, D3DXVECTOR3& v1, D3DXVECTOR3& v2,

FLOAT* t, FLOAT* u, FLOAT* v

);

ใช้ตรวจว่ารังสีตัดกับสามเหลี่ยมหรือไม่

Parameters:

orig - จุดกำเนิดรังสี

dir - ทิศทางของรังสี

v0, v1, v2 - จุดสามจุดของสามเหลี่ยม

t - ระยะห่างจากจุดกำเนิดรังสีถึงจุดตัดกับสามเหลี่ยมถ้ารังสีและสามเหลี่ยมตัดกัน

u, v - พิกัดของจุดตัดบนสามเหลี่ยม

Return Value:

true ถ้ารังสีและสามเหลี่ยมตัดกัน false ถ้ารังสีและสามเหลี่ยมไม่ตัดกัน

BOOL

Ray::InSphere(

const D3DXVECTOR3* vCenter,

float fRadius,

const D3DXMATRIX* matWorld,

const D3DXVECTOR3& RayOrigin,

const D3DXVECTOR3& RayDirection,

float* Distance = NULL

);

ใช้ตรวจว่ารังสีตัดกับทรงกลมหรือไม่

Parameters:

vCenter - จุดศูนย์กลางของทรงกลม

fRadius - รัศมีของทรงกลม

matWorld - Transformation Matrix ของทรงกลม

RayOrigin - จุดกำเนิดรังสี

RayDirection - ทิศทางของรังสี

Distance - ระยะทางจากจุดกำเนิดรังสีถึงจุดตัดกับทรงกลม

Return Value:

true ถ้ารังสีและทรงกลมตัดกัน false ถ้ารังสีและทรงกลมไม่ตัดกัน

HRESULT

Ray::ScrToRay(

int ptCursorX, int ptCursorY, int Width, int Height, CCamera *pCamera,

D3DXVECTOR3 *vRayOrigin, D3DXVECTOR3 *vRayDir

);

ใช้แปลงพิกัดของหน้าจอให้เป็นพิกัดในฉาก 3 มิติ

Parameters:

ptCursorX, ptCursorY - ตำแหน่งจุดบนจอภาพ

Width, Height - ความกว้างและความสูงของจอภาพ

pCamera - มุมกล้องปัจจุบันภายในฉาก

vRayOrigin - จุดกำเนิดรังสีผลลัพธ์

vRayDir - ทิศทางของรังสีผลลัพธ์

Return Value:

return S_OK เสมอ

BOOL

Ray::v3ToScr(

const D3DXVECTOR3 *v3DPoint,

const D3DXMATRIX* MatWorld,

const D3DXMATRIX* MatView,

const D3DXMATRIX* MatProj,

const int ScreenWidth, const int ScreenHeight,

int *pScreenX, int *pScreenY,

float *pZ = NULL

);

ใช้แปลงพิกัดในฉาก 3 มิติให้เป็นพิกัดของหน้าจอ

Parameters:

v3DPoint – จุดบนฉาก 3 มิติ

MatWorld - World Matrix

MatView - View Matrix

MatProj - Projection Matrix

ScreenWidth, ScreenHeight - ความกว้างและความสูงของจอภาพ

pScreenX, pScreenY, pZ - ตำแหน่งจุดบนจอภาพ

Return Value:

return true ถ้าจุดอยู่ใน view volume, false ถ้าจุดไม่อยู่ใน view volume

BOOL

Ray::InArfVertices(

const D3DXVECTOR3& RayOrigin,

const D3DXVECTOR3& RayDirection,

const D3DXMATRIX* matWorld,

float* pVertices,

unsigned int* pIndices,

int NumFaces,

int* pFaceIndex = NULL,

CTriangle* pTriangle = NULL,

FLOAT* pDistance = NULL,

FLOAT* pU = NULL,

FLOAT* pV = NULL

);

ใช้ตรวจว่ารังสีตัดกับรูป polygon หรือไม่

Parameters:

RayOrigin - จุดกำเนิดรังสี

RayDirection - ทิศทางของรังสี

matWorld - Transformation Matrix ของรังสี

pVertices - จุดที่ประกอบเป็น polygon

pIndices - Vertex Indices

NumFaces - จำนวนสามเหลี่ยมที่แทนพื้นผิวของ polygon

pFaceIndex - การเชื่อมต่อของจุดเป็น polygon

pTriangle - สามเหลี่ยมที่ตัดกับรังสี

pDistance - ระยะทางจากจุดกำเนิดรังสีถึงจุดตัด polygon

pU, pV - พิกัดของจุดตัดบนสามเหลี่ยม

Return Value:

true ถ้ารังสีและ polygon ตัดกัน false ถ้ารังสีและ polygon ไม่ตัดกัน

BOOL

Ray::InBlendArfVertices(

const D3DXVECTOR3& RayOrigin,

const D3DXVECTOR3& RayDirection,

const D3DXMATRIX* matWorld,

float* pVertices1, float* pVertices2, float Factor,

unsigned int* pIndices, int NumFaces,

int* pFaceIndex,

CTriangle* pTriangle,

FLOAT* pDistance,

FLOAT* pU,

FLOAT* pV

);

ใช้ตรวจว่ารังสีตัดกับรูป polygon หรือไม่โดยแปลงหารูป polygon ระหว่าง polygon 2 รูปก่อนคำนวณ

Parameters:

RayOrigin - จุดกำเนิดรังสี

RayDirection - ทิศทางของรังสี

matWorld - Transformation Matrix ของรังสี

pVertices1, pVertices2 - จุดที่ประกอบเป็น polygon รูปที่ 1 และ 2

Factor - ค่าน้ำหนักการเฉลี่ยระหว่าง polygon 2 รูป

pIndices - Vertex Indices

NumFaces - จำนวนสามเหลี่ยมที่แทนพื้นผิวของ polygon

pFaceIndex - การเชื่อมต่อของจุดเป็น polygon

pTriangle - สามเหลี่ยมที่ตัดกับรังสี

pDistance - ระยะทางจากจุดกำเนิดรังสีถึงจุดตัด polygon

pU, pV - พิกัดของจุดตัดบนสามเหลี่ยม

Return Value:

true ถ้ารังสีและ polygon ตัดกัน false ถ้ารังสีและ polygon ไม่ตัดกัน

4.3 ยูทิลิตี้ (Utilities)

Utilities เป็นส่วนที่จะใช้ในการจัดการส่วนอื่นประกอบในเกมนอกเหนือจากกลุ่มต่างๆ ที่ได้มีมาแล้ว ในส่วนของ Utilities นี้มีคลาสดังนี้

Timer

ใช้สำหรับจับเวลามี member function ดังนี้

float

Timer::GetLogicalTime();

ใช้ตรวจสอบค่าเวลานับปัจจุบัน

Parameters:

(none)

Return Value:

ค่าเวลานับปัจจุบัน

Image

ใช้สำหรับอ่านข้อมูลรูปภาพเข้ามาในหน่วยความจำโดยจะมีฟังก์ชันที่ใช้ในการโหลดภาพตระกูล TGA (.tga) ซึ่งเราสามารถนำข้อมูลนี้ไปใช้ในการทำเป็น Texture ต่อไป class นี้มี โครงสร้างข้อมูลที่เกี่ยวข้อง คือ TGAFILE ใช้อ่านข้อมูลรูปภาพประเภท .tga เข้ามาในหน่วยความจำ เช่นเดียวกันกับ bitmap file แต่ไฟล์ภาพตระกูลนี้จะมีข้อดีตรงที่สามารถเก็บค่า Alpha ของแต่ละ pixel ในรูปได้ จึงเหมาะสมในการใช้ทำ Texture ที่ต้องใช้เทคนิคการทำ Alpha Blending มีโครงสร้างดังนี้

```
struct TGAFILE
{
        unsigned char   imageTypeCode;
        short int       imageWidth;
        short int       imageHeight;
        unsigned char   bitCount;
        unsigned char*  imageData;
};
```

ส่วน class Image นั้นมี member function ดังต่อไปนี้

int

Image::LoadTGAFile(const char* filename, TGAFILE *tgafile);

ใช้ในการอ่านภาพตระกูล TGA ขึ้นมาบน memory เพื่อนำไปใช้งาน

Parameters:

filename – ชื่อไฟล์ที่จะอ่าน

tgafile - memory ที่จะใช้เก็บข้อมูล

Return Value:

return ค่า 1 เสมอ

int

Image::LoadImageData(const char* filename, float*& pfRGBA);

ใช้อ่านภาพในฟอร์แมต RGBA ขึ้นมาบน memory เพื่อนำไปใช้งาน

Parameters:

filename – ชื่อไฟล์ที่จะอ่าน

pfRGBA - memory ที่จะใช้เก็บข้อมูล

Return Value:

return ค่า 1 เสมอ

int

Image::LoadDepthData(const char* filename, float*& pfDepth);

ใช้อ่านภาพข้อมูลแทนความลึกของแต่ละ pixel ของภาพขึ้นมาบน memory เพื่อนำไปใช้งาน

Parameters:

filename – ชื่อไฟล์ที่จะอ่าน

pfDepth - memory ที่จะใช้เก็บข้อมูล

Return Value:

return ค่า 1 เสมอ

Tool

เป็น utility สำหรับใช้คำนวณสำหรับการประมวลผลเกม ประกอบไปด้วย function ดังต่อไปนี้

float

Tool::GetDistance(float x1, float y1, float z1, float x2, float y2, float z2);

ใช้คำนวณระยะทางระหว่างจุดสองจุด

Parameters:

x1, y1, z1 - พิกัดของจุดที่ 1

x2, y2, z2 - พิกัดของจุดที่ 2

Return Value:

ระยะทางระหว่างจุดสองจุด

void

Tool::CalBoundSphere(float* pVertices, int numVertices, float* pcx, float* pcy, float* pcz, float* pr);

ใช้คำนวณหาทรงกลมที่ครอบวัตถุที่แทนด้วย Polygon

Parameters:

pVertices - pointer ไปยัง vertex array

numVertices - จำนวน vertex

pcx, pcy, pcz - จุดศูนย์กลางของทรงกลม

pr - รัศมีของทรงกลม

Return Value:

(none)

void

Tool::GetPointFromTriangleUV(

const CTriangle& Triangle,

float U, float V,

float* pX, float* pY, float* pZ

);

ใช้แปลงพิกัดจุดบนสามเหลี่ยมให้เป็นพิกัดจริง

Parameters:

Triangle - สามเหลี่ยมที่ใช้ระบุพิกัดจุด

U, V - พิกัดของจุดบนสามเหลี่ยม

pX, pY, pZ - พิกัดของจุดบนสามเหลี่ยมตามแนวแกน X, Y, Z

Return Value:

(none)

void

Tool::RenderBlend(

float *pVertices0, float *pVertices1,

float *pTexCoord0, float *pTexCoord1,

float fBPos, float fBUV,

unsigned int *pIndices,

int NumFaces,

float* pVertexColor = NULL,

BOOL bBlendPosition = TRUE, BOOL bBlendUV = FALSE

);

เป็นฟังก์ชันสำหรับวาดภาพเฟรมที่อยู่ระหว่าง 2 เฟรมโดยใช้ factor 0 ถึง 1

Parameters:

pVertices0, pVertices1 - vertex ทั้งหมดของภาพเฟรมที่ 1 และ 2 ตามลำดับ

pTexCoord0, pTexCoord1 - texture coordinate ทั้งหมดของภาพเฟรมที่ 1 และ 2 ตามลำดับ

fBPos, fBUV - น้ำหนักการเฉลี่ย vertex และ texture coordinate ตามลำดับ มีค่าอยู่ระหว่าง 0-1

pIndices - Vertex Indices

NumFaces - จำนวนสามเหลี่ยมที่แทนพื้นผิวของ polygon

pVertexColor - สีของแต่ละจุดของสามเหลี่ยมบนรูป polygon

bBlendPosition, bBlendUV - ค่าที่บอกว่าจะเฉลี่ย vertex และ texture coordinate ตามลำดับ

Return Value:

(none)

void

Tool::RenderBlendAnimation(

CAnimation* pAnimation1, float fTime1,

CAnimation* pAnimation2, float fTime2,

float fBPos,float fBUV = 0.0f,

BOOL bBlendPosition = TRUE, BOOL bBlendUV = FALSE

);

เป็นฟังก์ชันสำหรับวาดภาพเฟรมที่อยู่ระหว่าง 2 เฟรมโดยใช้ factor 0 ถึง 1 จากข้อมูล Animation

Parameters:

pAnimation1, fTime1 - ข้อมูลเฟรมจาก Animation ที่ 1 และเวลาที่บอกเฟรมตามลำดับ

pAnimation2, fTime2 - ข้อมูลเฟรมจาก Animation ที่ 2 และเวลาที่บอกเฟรมตามลำดับ

fBPos, fBUV - น้ำหนักการเฉลี่ย vertex และ texture coordinate ตามลำดับ มีค่าอยู่ระหว่าง 0-1

bBlendPosition, bBlendUV - ค่าที่บอกว่าจะเฉลี่ย vertex และ texture coordinate ตามลำดับ

Return Value:

(none)

void

Tool::PutSqrTexture3D(

CUseTexture *pUseTexture,

float x, float y, float z,

float fSize,

const D3DXVECTOR3* pvFront,

float Alpha = 1.0f,

float Angle = 0.0f

);

ใช้วาดรูปสี่เหลี่ยมจัตุรัส 2 มิติลงในฉาก 3 มิติ

Parameters:

pUseTexture - texture ที่ใช้ในการวาด

x, y, z - จุดบนซ้ายของขอบเขตการวาด

fSize - ขนาดของสี่เหลี่ยมจัตุรัสบนฉาก 3 มิติ

pvFront - normal vector ของหน้าสี่เหลี่ยม

Alpha - ค่า Alpha ของรูปสี่เหลี่ยมที่จะวาด

Angle - มุมที่ทำกับหน้าจอ

Return Value:

(none)

void

Tool::CalPlaneEquation(

float v0x, float v0y, float v0z,

float v1x, float v1y, float v1z,

float v2x, float v2y, float v2z,

float *a, float *b, float *c, float *d

);

ใช้คำนวณหาสมการระนาบในระบบพิกัด 3 มิติ

Parameters:

v0x, v0y, v0z - จุดที่ 1 ของสามเหลี่ยม

v1x, v1y, v1z - จุดที่ 2 ของสามเหลี่ยม

v2x, v2y, v2z - จุดที่ 3 ของสามเหลี่ยม

a, b, c, d - ค่าสัมประสิทธิ์ของสมการระนาบ

Return Value:

(none)

4.4 เอนจินต์เฟรมเวิร์ค (Engine Framework)

เอนจินต์เฟรมเวิร์คใช้สำหรับ อำนวยความสะดวกในการสร้างโปรแกรม และช่วยจัดการ Initialize และ Cleanup API ต่างๆ ที่อยู่ภายใต้เอนจินต์ให้พร้อมใช้งาน โดยผู้พัฒนาเกมจะสืบทอดจากคลาส **CGLApplication** และต้อง Override member function ดังนี้

LRESULT

CGLApplication::MsgProc(HWND hWnd, UINT uMsg, WPARAM wParam, LPARAM lParam);

ใช้จัดการตอบสนองกับ Windows Messages

Parameters:

HWnd - Window Handle

uMsg - ชนิดของ message

wParam - ตัวแปรข้อมูลของ message

lParam - ตัวแปนข้อมูลเสริมของ message

Return Value:

return ค่าที่บอกสถานะการจบฟังก์ชันให้กับระบบ Windows

HRESULT

CGLApplication::InitDeviceObjects()

ใช้สำหรับจอง Graphics Buffer ของ Graphics API ตามความต้องการในการใช้งาน

Parameters:

(none)

Return Value:

สำเร็จ return S_OK, ล้มเหลว return E_FAIL

HRESULT

CGLApplication::DeleteDeviceObjects()

คืน Graphics Buffer ให้กับระบบ โดยต้องเรียกใช้ก่อนออกจากโปรแกรม หรือเมื่อมีการ Initialize Graphics API ในโหมดความละเอียดอื่นๆ

Parameters:

(none)

Return Value:

สำเร็จ return S_OK, ล้มเหลว return E_FAIL

HRESULT

CGLApplication::OneTimeSceneInit()

ใช้สำหรับอ่านข้อมูลบางอย่างที่จำเป็นต้องใช้ตลอดทั้งโปรแกรม

Parameters:

(none)

Return Value:

สำเร็จ return S_OK, ล้มเหลว return E_FAIL

HRESULT

CGLApplication::FinalCleanup()

คืน Memory ทั้งหมดให้กับระบบ โดยต้องเรียกใช้ก่อนออกจากโปรแกรม

Parameters:

(none)

Return Value:

สำเร็จ return S_OK, ล้มเหลว return E_FAIL

HRESULT

CGLApplication::FrameMove()

ใช้ในการควบคุมเกม Loop ซึ่งเป็นส่วนที่ใช้ในการประมวลผลการดำเนินเรื่องของเกม เช่น การควบคุมตัวละคร, AI, ฟังก์ชันการคำนวณต่างๆ ที่เกี่ยวข้องกับเกม

Parameters:

(none)

Return Value:

สำเร็จ return S_OK, ล้มเหลว return E_FAIL

HRESULT

CGLApplication::Render()

ใช้สำหรับวาดภาพเพื่อแสดงผลวัตถุทั้ง 2 มิติ และ 3 มิติ รวมทั้ง GUI และสร้างภาพ Effect

Parameters:

(none)

Return Value:

สำเร็จ return S_OK, ล้มเหลว return E_FAIL

โดยทั่วไปโปรแกรมที่ใช้ CGLApplication จะมีขั้นตอนการทำงานโดยสังเขปดังนี้

1. เรียกฟังก์ชัน OneTimeSceneInit
2. เรียกฟังก์ชัน InitDeviceObjects
3. เข้าสู่เกมลูป
    3.1 เรียกฟังก์ชัน FrameMove
    3.2 เรียกฟังก์ชัน Render
4. ระหว่างเกมลูปจะมีการประมวลผล Windows Message ไปด้วย ทำให้ Message จากระบบ Windows เข้ามาเรียกในฟังก์ชัน WndProc
5. ถ้าเรารับอินพุทที่บอกกับโปรแกรมว่าให้ออกจากโปรแกรม ก็ให้ทำการคืนทรัพยากรทั้งหมดให้กับระบบก่อนออกจากโปรแกรมโดย
    5.1 เรียกฟังก์ชัน DeleteDeviceObjects
    5.2 เรียกฟังก์ชัน FinalCleanup

ระหว่างการทำงานถ้ามีการเปลี่ยนโหมดการทำงานของ OpenGL และจำเป็นต้องสร้าง Render context ขึ้นใหม่ เราจะต้องเรียก DeleteDeviceObjects จากนั้นสร้าง OpenGL Render Context ขึ้นมาใหม่ แล้วจึงเรียก InitDeviceObjects เพื่อโหลดทรัพยากรที่จำเป็นต้องใช้กลับเข้าสู่ระบบ

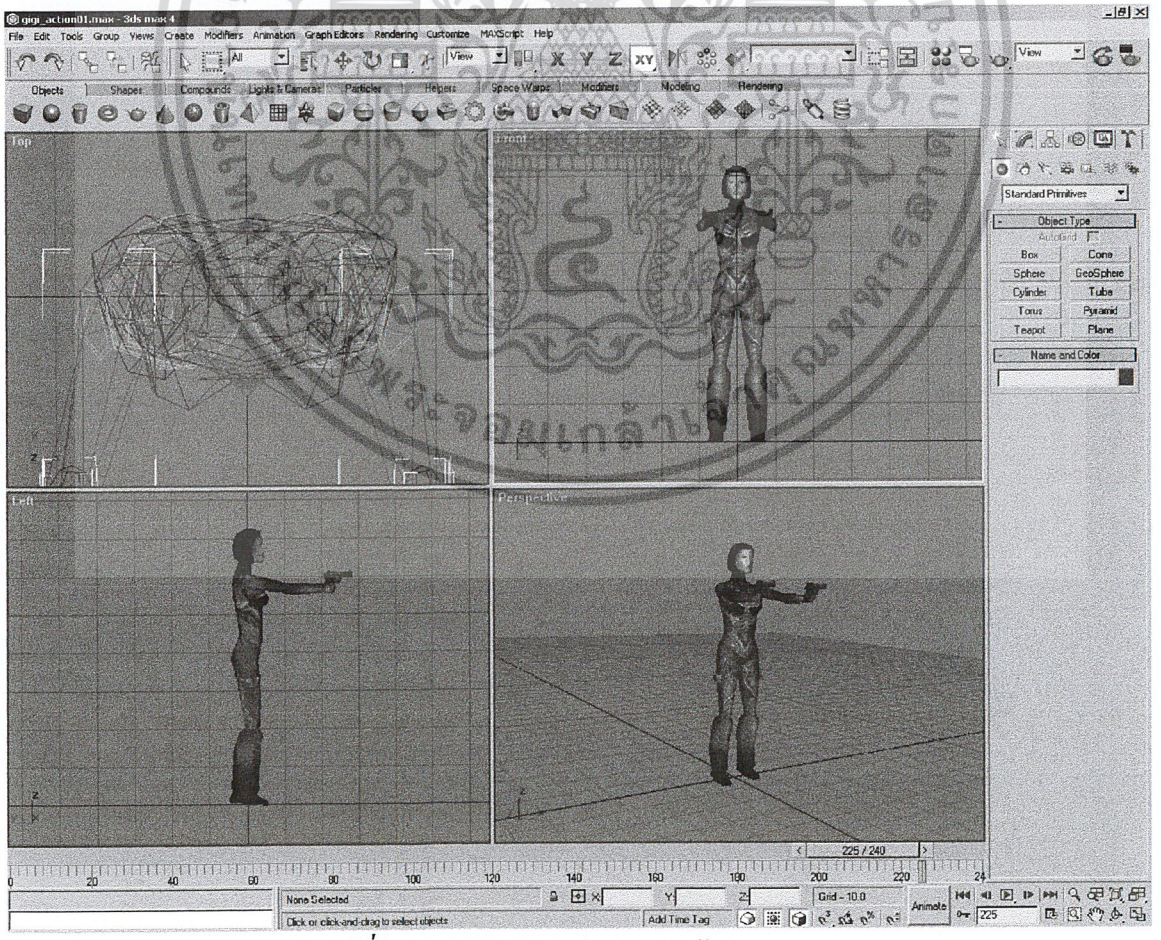
บทที่ 5
การใช้งานเกมเอนจินต์ (Using Game Engine)

5.1 ขั้นตอนการจัดเตรียมข้อมูลเพื่อทำแอนิเมชันของตัวละคร

การเคลื่อนไหวในรูปแบบเวอร์เท็กซ์บเลนดิง ข้อมูลของวัตถุที่อ่านหรือโหลดขึ้นมาได้นั้นจะเป็นลักษณะของวัตถุที่อยู่นิ่งกับที่ ข้อมูลที่จัดเก็บจะเก็บเป็นเวอร์เท็กซ์ของวัตถุที่อยู่นิ่งกับที่ โดยแบ่งเป็นช่วงของการเคลื่อนไหวในลักษณะต่างๆ การทำการเคลื่อนไหวจึงจะต้องอาศัยเทคนิคการอินเตอร์โพเลตมาช่วยทำให้การทำการเคลื่อนไหวมีความราบรื่น จึงเป็นหน้าที่ของผู้พัฒนาเอง โดยมีหลักการและวิธีการที่จะต้องสร้างและจัดเตรียมข้อมูลการเคลื่อนไหวให้มีความเหมาะสมกับลักษณะท่าทางที่แตกต่างกันของตัวละคร ที่มีกลไกในการเคลื่อนไหวของวัตถุนั้นเองโดยอาศัยข้อมูลการเคลื่อนไหวของวัตถุที่เตรียมไว้ การจัดเก็บข้อมูลการเคลื่อนไหวนี้จึงเป็นขั้นตอนหนึ่งในการพัฒนาแอนิเมชันของตัวละครในเกมสามมิติ

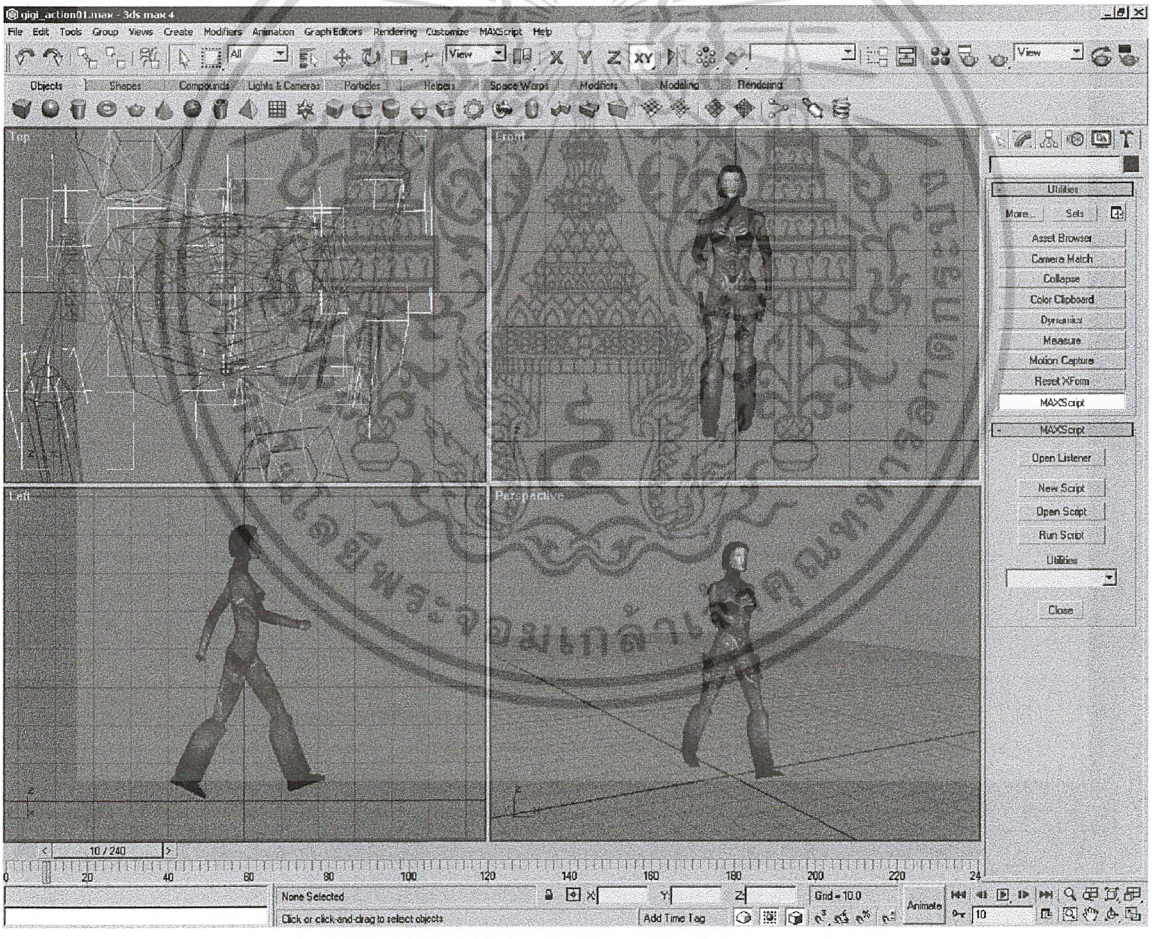
5.1.1 เอ็กพอร์ตแอนิเมชันด้วย Maxscript ของ 3ds-max

สร้างแอนิเมชันของตัวละครด้วยโปรแกรม 3ds-max ก่อนที่จะทำการเอ็กพอร์ต ในกริยาท่าทางต่างๆ ของตัวละคร



รูปที่ 5-1 ทำแอนิเมชันของตัวละครด้วย 3ds-max

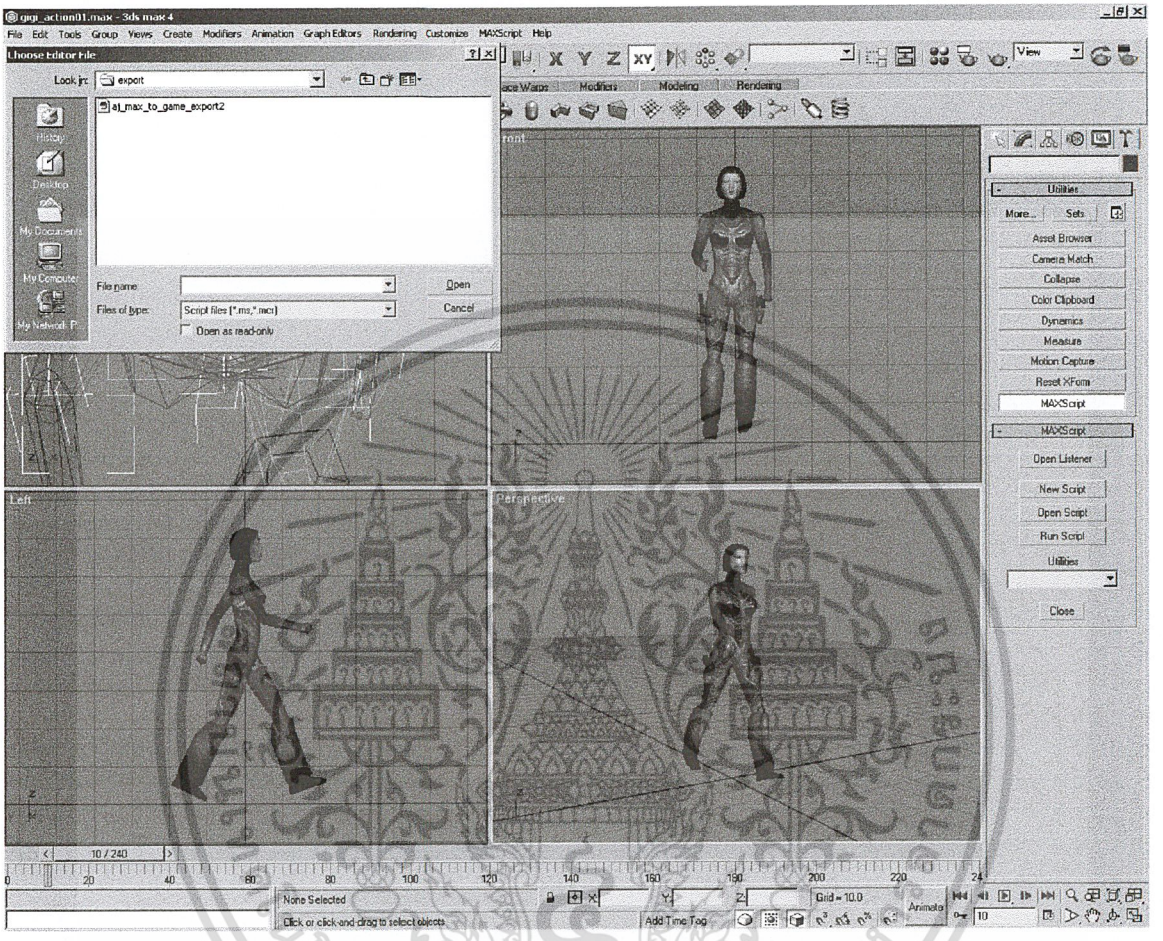
เมื่อได้แอนิเมชันของตัวละครแล้วจะเป็นขั้นตอนการเอ็กพอร์ตไฟล์ฟอร์แมตของ 3ds-max เพื่อให้ได้ไฟล์ฟอร์แมต 5 แบบที่มีส่วนขยายดังนี้ mac, maf, mai, map และ mau ซึ่งเป็นฟอร์แมตที่จะสามารถนำไปใช้ในการพัฒนาเกม เลือกเฟรมที่จะทำการเอ็กพอร์ตแล้ว ใช้เครื่องมือ Max Script ในแท็ปยูทิลิตี้  ของ 3ds-max ซึ่งเป็น Script ที่ได้จากการพัฒนาขึ้นเพื่อให้เป็นเครื่องมือที่ใช้เป็นเครื่องมือในการเอ็กพอร์ตไฟล์ ในการพัฒนาส่วนแอนิเมชันของเกมนั้น จะต้องทำการเอ็กพอร์ตบ่อยครั้ง ยกตัวอย่างท่าเดินของตัวละครซึ่งจะต้องทำการเอ็กพอร์ตสามเฟรมเป็นอย่างน้อย เริ่มจากเฟรมที่เป็นท่ายืนปกติ เฟรมก้าวเท้าซ้าย และเฟรมก้าวเท้าขวา ใช้เทคนิคการอินเตอร์โพเลตเข้าช่วย ผู้จัดทำได้มีไลบรารีที่ทำส่วนนี้ไว้แล้ว จึงจะได้เป็นแอนิเมชันของการเดิน ซึ่งจากการยกตัวอย่างนั้นยังน้อยไปสำหรับการเอ็กพอร์ตเพียงสามเฟรม เพราะจะทำให้ท่าการเดินของตัวละครไม่ราบรื่นเท่าที่ควร การเอ็กพอร์ทเฟรมออกมามากขึ้นจะช่วยให้แอนิเมชันของตัวละครในเกมมีความราบรื่น แต่หากมีการเฟรมที่เอ็กพอร์ตมาใช้มากเกินไปก็อาจมีผลกระทบกับความเร็วในการแสดงผลของเกม ทั้งนี้แล้วผู้พัฒนาเกมจึงต้องใช้ดุลพินิจจัดการให้เฟรมที่ได้มีความเหมาะสมกับแอนิเมชันของตัวละคร



รูปที่ 5-2 เครื่องมือของแม็กสคริปต์ (MAX Script)ของ 3ds-max

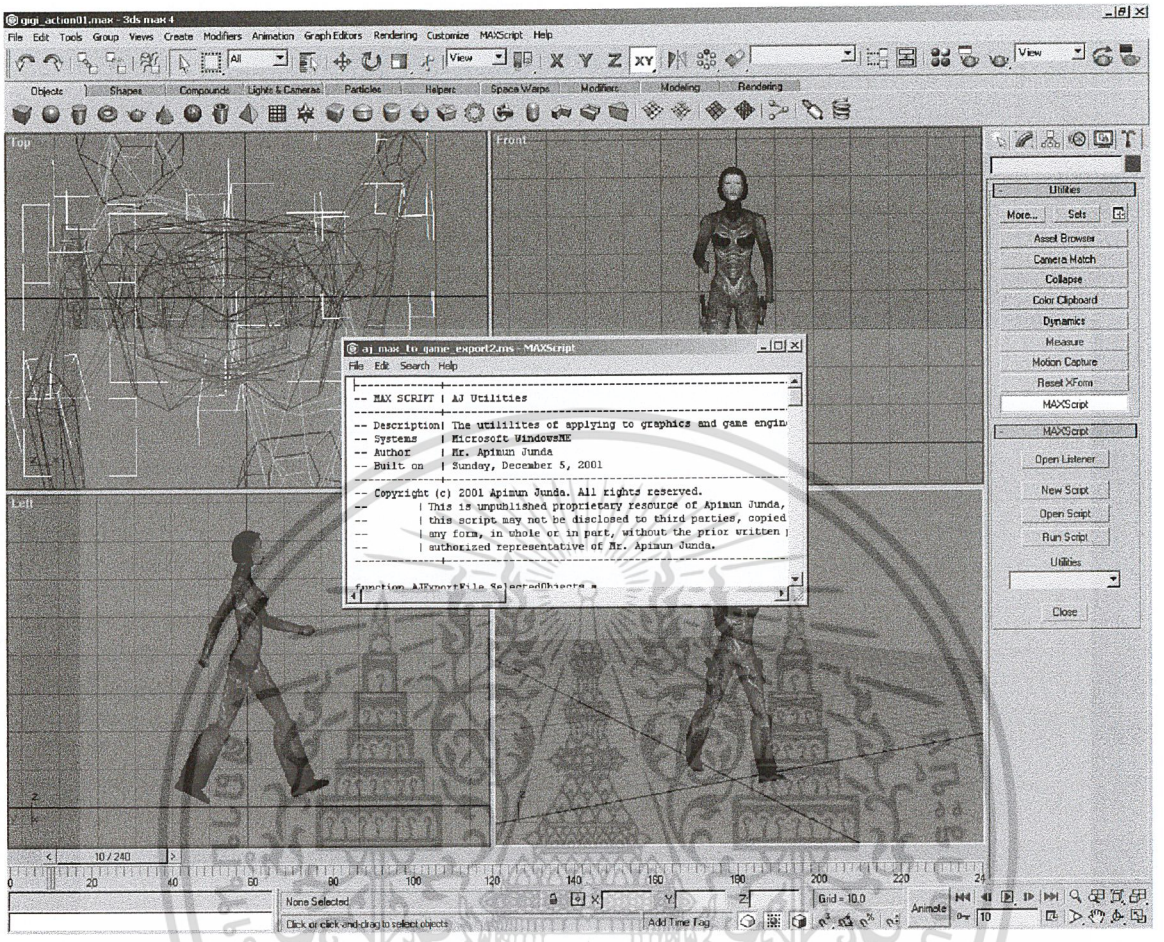
5.1.1.1 เปิดสคริปต์ (Open script) ที่ใช้ในการเอ็กพอร์ต

เมื่อเลือกเฟรมที่จะเอ็กพอร์ตแล้ว เลือก Open Script ในเครื่องมือ Max Script ในแท็ปยูทิลิตี้ ของ 3ds-max แล้วทำการเลือก Script file(*.ms) ที่ได้จัดเตรียมไว้แล้ว



รูปที่ 5-3 เปิดสคริปต์ (Open MAX Script)

จะได้ Dialog box เล็กๆ ขึ้นมาใหม่ ซึ่งจะเป็นไฟล์ Max Script ที่ได้ทำการเปิด



รูปที่ 5-4 MAX Script Dialog box ที่ได้จากการเปิด

5.1.1.2 แก้ไขสคริปต์ที่จะใช้ในการเอ็กพอร์ต

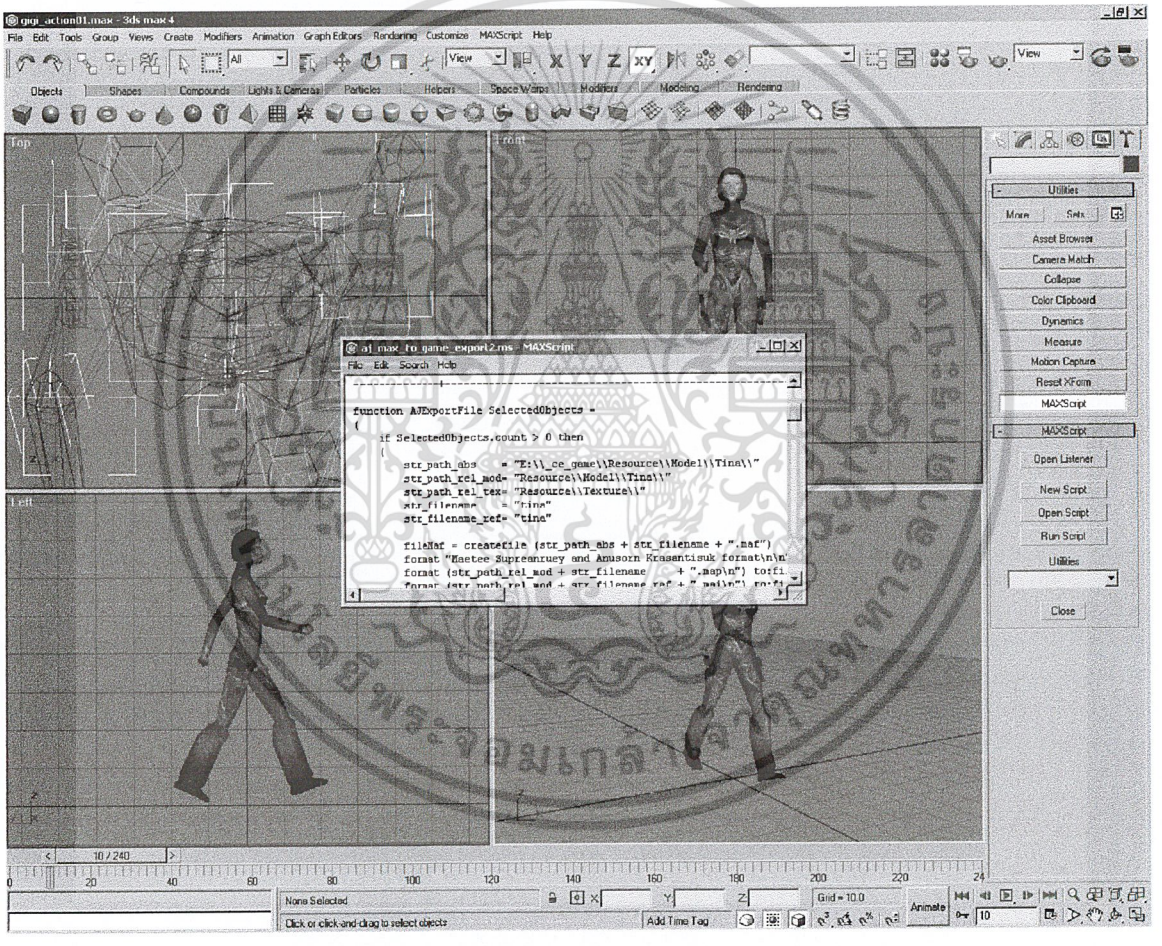
ทำการแก้ไข Path และชื่อไฟล์ที่จะทำการเอ็กพอร์ต ใน MAX Script

```
str_path     = " E:\\_ce_game\\Resource\\Model\\Tina\\"
str_filename = "tina"
```

str_path คือ Path ของไฟล์ที่จะทำการเอ็กพอร์ต

str_filename คือชื่อของไฟล์ที่จะทำการเอ็กพอร์ต

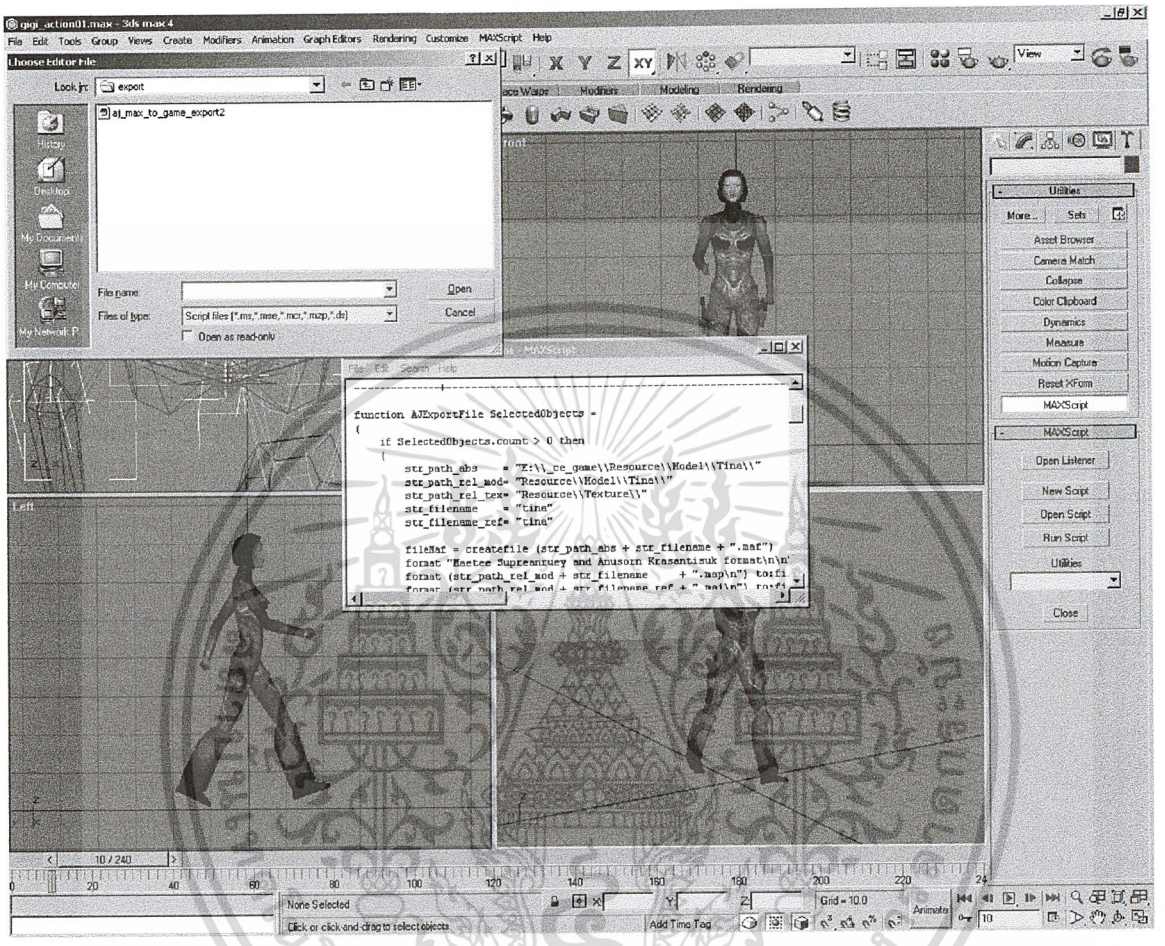
จากนั้นทำการบันทึก MAX Script ไฟล์



รูปที่ 5-5 แก้ไข Path และชื่อไฟล์ใน MAX Script

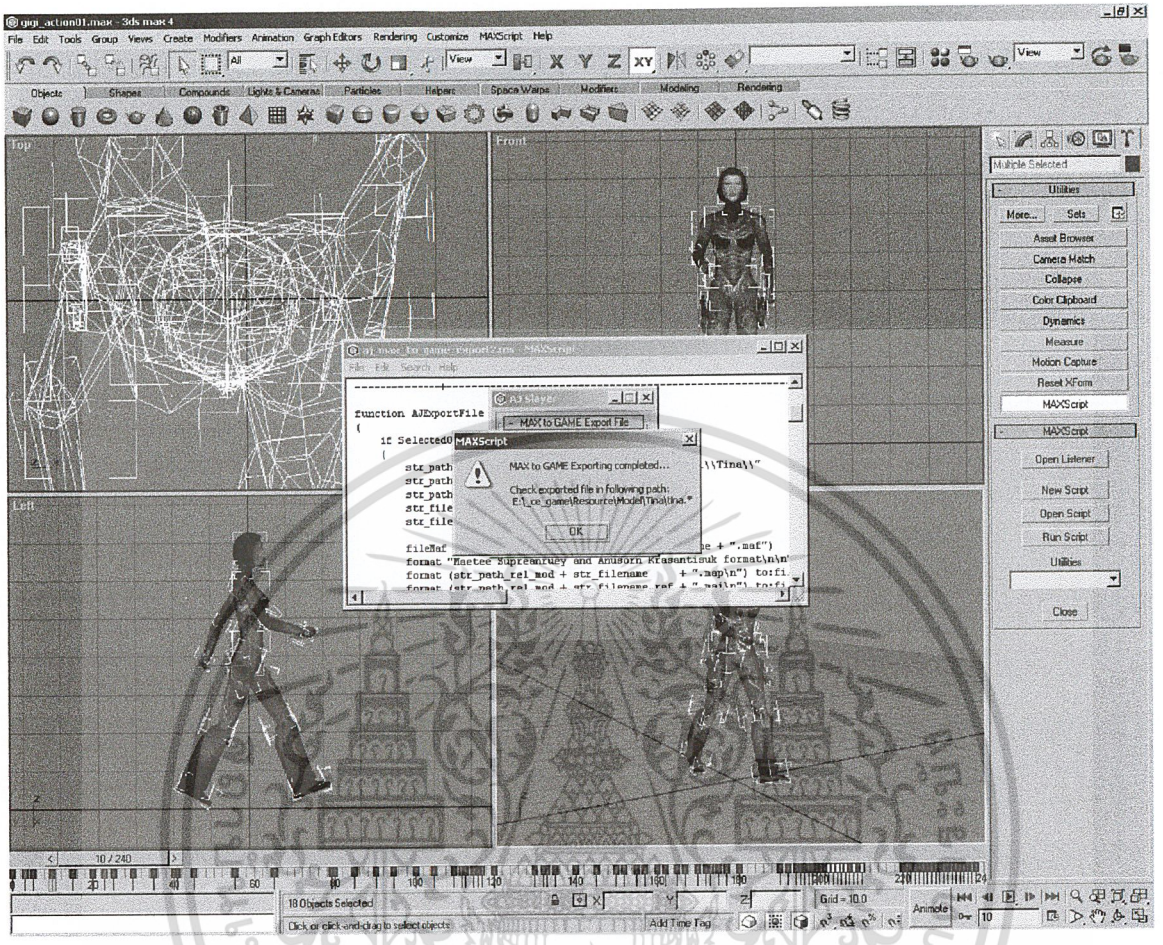
5.1.1.3 รันสคริปต์ (Run script) ที่ใช้ในการเอ็กพอร์ต

เลือก Run Script ในเครื่องมือ Max Script ในแท็ปยูทิลิตี้ ของ 3ds-max แล้วทำการเลือก Script file (*.ms) ที่ได้จัดเตรียมไว้แล้ว ซึ่งก็เป็นไฟล์เดียวกับที่ทำการเปิดมาแก้ไขแล้ว



รูปที่ 5-6 รัน MAX Script

เลือก Export scene จากนั้นจะได้ Dialog box แสดงผลการเอ็กพอร์ตครั้งนั้น



รูปที่ 5-7 ผลที่ได้จากการ Export Scene สมบูรณ์

จากการรัน Max Script แล้วจะมีไฟล์ที่เกิดขึ้นใหม่ 5 ไฟล์ด้วยกัน ซึ่งเป็นไฟล์ที่มีชื่อที่ได้แก้ไขแล้วใน MAX Script ไฟล์

- Filename.MAC
- Filename.MAF
- Filename.MAI
- Filename.MAP
- Filename.MAU

5.2 รูปแบบของไฟล์ข้อมูลที่เอ็กพอร์ต

- Filename.MAS

  สำหรับบอกวัตถุที่เป็นแอนิเมชัน บอกว่ามีแอคชั่นใดบ้าง ชื่อของแอคชั่นนั้น และส่วนที่อ้างอิงไปยังไฟล์ .MAA

- Filename.MAA

  บอกว่าแอนิเมชันในแอคชั่นนั้นมีกี่เฟรม โดยแต่ละช่วง บอกช่วงของความเร็ว และส่วนที่อ้างอิงไปยังไฟล์ .MAF

- Filename.MAF

  เป็นไฟล์ที่บอกจุดอ้างอิงไปยังไฟล์อื่นที่เกี่ยวข้อง ที่ใช้เก็บแอนิเมชันเฟรมของรูป

  fileMaf = createfile "C:\export\frame\man_010.maf"

- Filename.MAP

  เซตของจุดที่ประกอบเป็นเวอร์เท็กซ์ บอกจำนวนของเวอร์เท็กซ์, ตำแหน่งของเวอร์เท็กซ์ ที่แสดงรูปร่างของวัตถุเฟรมนั้น

  format "model\\man_001.map\n" to:fileMaf

- Filename.MAI

  เซตของดัชนีการเชื่อมต่อของจุด (Topology Index) ที่ประกอบเป็นเวอร์เท็กซ์ จะเป็นอินเดกซ์ที่บอกเฟส, จำนวนเฟสที่ Render รายละเอียดในไฟล์แต่ละบรรทัดจะเป็นอินเดกซ์ ซึ่งชี้ไปยังไตรเฟสในไฟล์ .MAP ว่าจุดใดบ้างที่จะถูกวาดต่อหนึ่งสามเหลี่ยม

  format "model\\man_shr.mai\n" to:fileMaf

- Filename.MAU

  เป็นเท็กเจอร์โคออดิเนท เพื่อใช้แมพรูปภาพไปยังพื้นผิวของวัตถุ รายละเอียดในไฟล์ในบรรทัดแรกจะบอกจำนวนของเท็กเจอร์โคออดิเนท ในบรรทัดต่อไปจะบอกเท็กเจอร์โคออดิเนทของ U และ V

  format "model\\man_shr.mau\n" to:fileMaf

  ชื่อไฟล์รูปที่ใช้เป็น Texture

  format "Texture\\man.bmp\n" to:fileMaf

- Filename.MAC

  เซตของสีของวัตถุ (ปกติถ้ามีการใช้เท็กเจอร์เราจะไม่ใช้ไฟล์นี้)

  format "model\\man_shr.mac\n" to:fileMaf

สำหรับรายละเอียดในการเขียนและใช้งาน Max Script เพื่อทำงานร่วมกับโปรแกรม 3D Studio MAX อ้างอิงจาก Max Script ซึ่งมาพร้อมกับ 3D Studio Max แต่ทั้งนี้ผู้พัฒนาได้จัดเตรียมสร้าง Max Script สำหรับ Export ไฟล์ ดังกล่าวไว้แล้ว ถ้าหากผู้ใช้ไม่ต้องการปรับปรุงหรือเปลี่ยนแปลงส่วนใด ก็สามารถนำไปใช้ได้เลย โดยใช้ได้ร่วมกับ โปรแกรม 3D Studio MAX ตั้งแต่ Version 4.0 ขึ้นไป

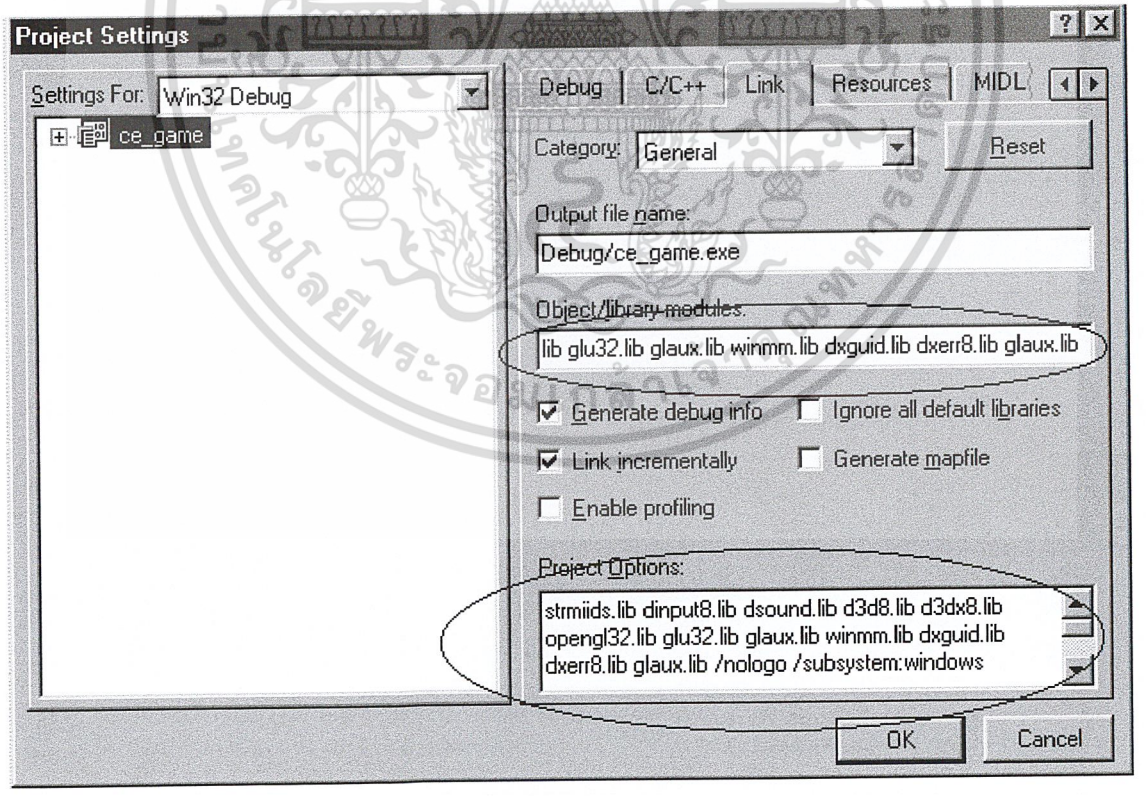
เราสามารถควบคุมการเคลื่อนไหวของตัวละครโดยการสร้างไฟล์ประเภท .MAA ซึ่งภายในจะประกอบไปด้วยรูปแบบที่กำหนดไว้ดังนี้

```
3 Loop // คือจำนวนเฟรมทั้งหมดและ Loop flag สำหรับบอกให้ทำ Animation แบบ วนไปเรื่อยๆ
Frame\\ man_001.maf Frame\\ man_002.maf  1.0   // คือเฟรมที่ 1 - 2
Frame\\ man_002.maf Frame\\ man_003.maf  0.5   // คือเฟรมที่ 2 – 3
Frame\\ man_003.maf Frame\\ man_001.maf  0.5   // คือเฟรมที่ 3 – 1 (วนรอบ)
```

ตัวเลขด้านหลังจะแสดงความห่างของระยะเวลาระหว่างสองเฟรมต่อกัน แต่ทั้งนี้ยังสามารถปรับแต่งได้ในขณะที่เรียกใช้งาน Engine จากโปรแกรม

5.3 การใช้เกมเอนจินต์เฟรมเวิร์ค

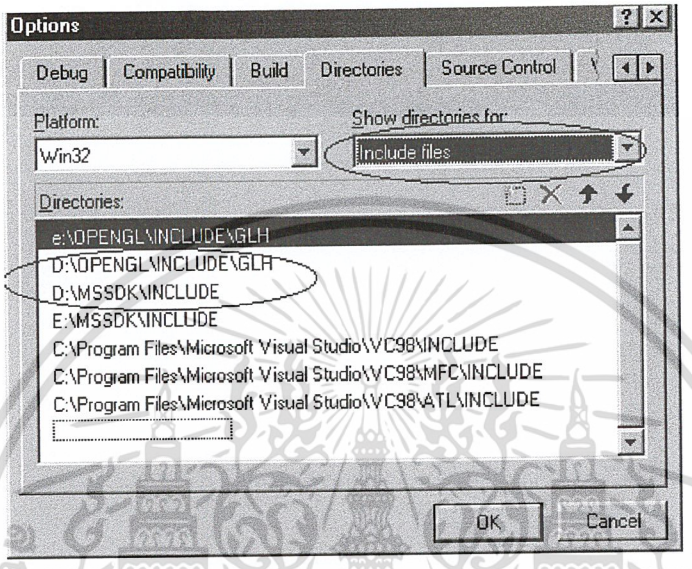
หลังจากที่ได้เตรียมข้อมูลต่างๆ สำหรับการสร้างเกมแล้ว ขั้นตอนต่อไปคือการสร้างเกม ในการสร้างเกมจะใช้เฟรมเวิร์ค ซึ่งประกอบไปด้วยคลาสต่างๆ ที่เตรียมไว้ให้ผู้ใช้สามารถสืบทอด หรือเรียกใช้ฟังก์ชันได้ ในการใช้เฟรมเวิร์คสร้างเกมนั้น มีขั้นตอนดังนี้



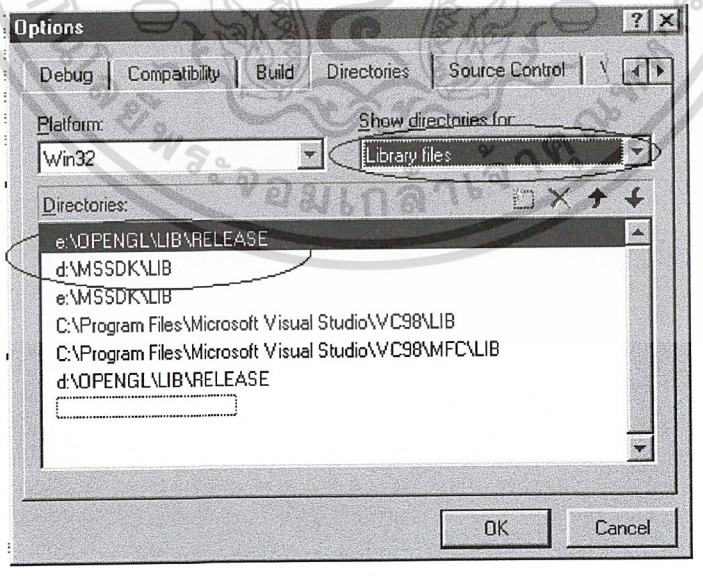
รูปที่ 5-8 ไฟล์ไลบรารีที่ต้องการ

เปิดเฟรมเวิร์คด้วย VC++ ตั้งค่าต่างๆดังนี้

เพิ่มไลบรารี dinput8.lib, dsound.lib, d3d8.lib, d3dx8.lib, opengl32.lib, glu32.lib, glaux.lib, winmm.lib, dxguid.lib, dxerr8.lib และglaux.lib โดยเลือกที่ Project แล้วเลือก Settings ที่แถบลิ้ง เพิ่มไลบรารีดังกล่าวลงไป (ไลบรารีเหล่านี้ได้จากการติดตั้งโปรแกรมต่างๆ ดังแสดงในภาคผนวก)

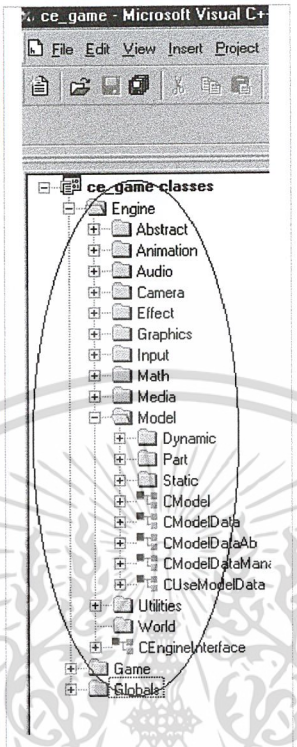


รูปที่ 5-9 Directory ของ Include Files



รูปที่ 5-10 Directory ของ Include Files

ระบุที่อยู่ของไฟล์ไลบรารี โดยเลือกที่ Tools แล้วเลือก Options เลือกที่แถบ Directories จากนั้นให้ระบุตำแหน่งที่อยู่ของไลบรารี MSSDK และ OPENGL ลงไป



รูปที่ 5-11 แสดง Workspace ของเอนจินต์

```
Copyright (c) 2001 Aj Slayer Team. All rights reserv
-------------------------------------------------------
*
class CMyGlApp : public CGLApplication
{
public:
    //---members variable-----------------------------
    static CColor4f s_oSceneColor;
    static CLight   s_oSceneLight;
    static CFog     s_oSceneFog;
    CDMyCamera      m_oCamera;

    //---constructors & destructor--------------------
    CMyGlApp();
    virtual ~CMyGlApp();

    //---members function-----------------------------
    virtual HRESULT InitDeviceObjects(GLvoid);
    virtual HRESULT DeleteDeviceObjects(GLvoid);
    virtual HRESULT OneTimeSceneInit(void);
    virtual HRESULT FinalCleanup(void);
    virtual HRESULT FrameMove(GLvoid);
    virtual HRESULT Render(GLvoid);

    static void ShowLoading(float,const CString&);
    HRESULT ProcessKey(float);
    LRESULT CALLBACK MsgProc(HWND,UINT,WPARAM,LPARAM)

protected:
    //---members variable-----------------------------
```

รูปที่ 5-12 แสดงส่วนสำคัญของคลาส CMyGlApp

สร้างคลาสหลักของเกมที่จะเป็นส่วนควบคุมการดำเนินของเกม โดยการสืบทอดมาจากคลาส CGLApplition และสร้างตัวแปรที่จำเป็นต่างๆ เช่น กล้อง เมาส์ ในคลาสนี้จะมีฟังก์ชันสำคัญ 3 ตัวที่จะต้องเขียนขึ้นทับ คือ InitDeviceObjects, FrameMove และRender

เขียนฟังก์ชัน InitDeviceObjects ขึ้นใหม่ ส่วนนี้เป็นส่วนของการโหลดข้อมูลต่างๆ ที่เตรียมไว้ เข้ามาไว้ใช้ในเกม และการตั้งค่าเริ่มต้นของเกม เช่น ตำแหน่งเริ่มของกล้อง ตำแหน่งเริ่มของตัวละคร

```
HRESULT CMyGlApp::InitDeviceObjects(GLvoid)
{
    ShowLoading(0.0f,"Wait for loading");

    ShowLoading(0.0f,"Generate camera");

    static BOOL  bFirstTime = TRUE;

    m_Camera.SetProjParams( D3DX_PI / 4.0f, 1.33f * m_fMoni
    //m_Camera.SetProjParams( D3DX_PI * 14.0f / 180.0f, 1.6

      ShowLoading(5.0f,"Load model : Land");
      ModelLand.LoadModel( "Frame\\RealLand_01.maf" );

      // Projectile Object
      ProjectileObject.LoadModel( "Frame\\BigBullet.maf"
      arProjectileObject.SetLand( &ModelLand );

      ShowLoading(10.0f,"Load model : Water");
      ModelWater.LoadModel("Frame\\Water.maf");
      //ModelLand.LoadModel( "Frame\\Land.maf" );

      ShowLoading(15.0f,"Load model : Sky");
      ModelSky.LoadModel( "Frame\\Sky.maf" );

      ShowLoading(20.0f,"Load model : Tank");
//    ModelBox.LoadModel( "Frame\\Tank.maf" );
```

รูปที่ 5-13 แสดงตัวอย่าง Source Code ในส่วนโหลดข้อมูล

เขียนฟังก์ชัน Render ขึ้นใหม่ ส่วนนี้จะเป็นส่วนแสดงผลของวัตถุต่างๆ ออกทางจอภาพ ต้องการให้วัตถุชิ้นไหนแสดงผล ก็ให้ใช้ฟังก์ชัน Render ของวัตถุนั้นในฟังก์ชันนี้

```
CEffectBillboardList CloudList;

HRESULT CMyGlApp::Render(GLvoid)
{


    //Camera View
    m_Camera.View();

    // Clear Scene
//  ModelBox.Render();

    GLGfx::Clear(GL_COLOR_BUFFER_BIT | GL

    ModelSky.Render()
    glEnable(GL_FOG);

|
    ModelLand.Render()
    //=================================
    //ModelLand.Render();
    //glColorMask(TRUE,TRUE,TRUE,TRUE);


    //Soldier01.SetPosition( -2, 0, 0 );
    //Soldier01.Render();

    //=====================Render Sold
    SoldierList.Render(&m_Camera);
    glDisable(GL_STENCIL_TEST);
    //=================================

    arProjectileObject.Render();
```

รูปที่ 5-14 แสดงตัวอย่าง Source Code ส่วน Render

เขียนฟังก์ชัน FrameMove ขึ้นใหม่ สำหรับจัดการการดำเนินการต่างๆของเกม เช่น การเคลื่อนไหวของตัวละคร เมื่อมีการกดปุ่ม การตรวจสอบการกดปุ่มต่างๆ การเปลี่ยนแปลงค่าต่างๆ ตามเวลาที่ผ่านไปในแต่ละรอบการคำนวณ และอาจสร้างฟังก์ชัน ProcessKey ขึ้นมาอีกเพื่อง่ายต่อการจัดการกับการตรวจสอบปุ่มกด

```
*/
HRESULT CMyGlApp::ProcessKey( float fElapsedTime )
{
    switch( CScene::s_oPage )
    {
        case MENU_000:
        {
        }
        break;

        case LOADING:
        {
        }
        break;

        case GAMELOOP:
        {
    if( m_arKey['P'] )
    {
        CMyGlApp::s_oSceneFog.m_fEnd    += fElapsedTime*100.0f;
    }
    else
    if( m_arKey['O'] )
    {
        CMyGlApp::s_oSceneFog.m_fEnd    -= fElapsedTime*100.0f;
    }
    else
    if( m_arKey['L'] )
    {
        CMyGlApp::s_oSceneFog.m_fStart  += fElapsedTime*100.0f;
    }
    else
    if( m_arKey['K'] )
    {
        CMyGlApp::s_oSceneFog.m_fStart  -= fElapsedTime*100.0f;
    }
    else
    {
        // Process camera key control
        RECT rcScreen;
        ::GetClientRect( m_hWnd, &rcScreen );
//      m_oCamera.ProcessKey( fElapsedTime, rcScreen, &m_arKey[0] );
```

รูปที่ 5-15 แสดงตัวอย่าง Source Code ในส่วนการกดปุ่ม

```
*/
HRESULT CMyGlApp::FrameMove(GLvoid)
{
    // Local static data
    static double   s_dAngle = 0;
    static float    s_fX = 5.0f;
    static float    s_fZ = 5.0f;
    static float    s_fHigh;


    if(!CSoundContainer::s_oSound[SOUND_BACKGROUND0(
        CSoundContainer::s_oSound[SOUND_BACKGROUND0(


    // Activate to engine interface
    CEngineInterface::AddLogicalTime( m_fElapsedTime


    // Insert key processing
    ProcessKey( m_fElapsedTime );
    CScene::FrameMove( m_fElapsedTime );

    m_oCamera.LayOn( &g_oLand001 );

    BYTE Data[10];
    if( Port::ReadData( 10, Data ) )
    {
        ProcessComData( Data );
    }

    arProjectileObject.Process( m_fElapsedTime );
```

รูปที่ 5-16 แสดงตัวอย่าง Source Code ในส่วนประมวลผล

บทที่ 6
การพัฒนาเกมเพื่อทดสอบเกมเอนจินต์

6.1 ประเภทเกมที่พัฒนา

เกมสาธิตในงานวิจัยชิ้นนี้ได้พัฒนาขึ้นมาเพื่อทดสอบเกมเอนจินต์ว่าสามารถนำไปใช้สร้างเกมได้สะดวกยิ่งขึ้นซึ่ง เกมเอนจินต์ที่ได้ออกแบบไว้ข้างต้นนั้นเราได้นำมาใช้พัฒนาเป็นเกมสาธิตขึ้นมา 4 โปรแกรม ซึ่งก็มีแนวคิดในการสร้างและการเล่นที่แตกต่างกัน แต่ตัวเกมจะไม่ซับซ้อนมากนักเนื่องจากเราใช้เวลาในการพัฒนาเอนจินต์ค่อนข้างนานมาก จึงทำให้มีเวลาพัฒนาเกมสาธิตขึ้นมาได้น้อยและอาจจะยังใช้ความสามารถของเอนจินต์ได้ไม่ครบในบางส่วนสำหรับในบางเกม แต่ในเกมสาธิตที่พัฒนาขึ้นมาทั้ง 4 โปรแกรม จะใช้ความสามารถในส่วนต่าง ๆ ของเกมเอนจินต์ในแนวที่แตกต่างกันออกไป

สำหรับเกมที่ผู้จัดทำได้พัฒนาขึ้นโดยใช้เกมเอนจินต์ร่วมกันกับกลุ่มการพัฒนาเกม 3 มิติโดยมี ดร. วรวัฒน์ ลิ้มโภคาเป็นอาจารย์ที่ปรึกษา เกมนี้มีชื่อว่า Military Strike เป็นเกมแนว 3D Shooting First Person ที่พัฒนาเพื่อทดสอบความสามารถของ Library ที่ช่วยในการพัฒนาเกม กติกาของเกมนี้จึงยังไม่ซับซ้อนมากนัก ในเกมผู้เล่นจะต้องยิงศัตรูให้ได้มากที่สุด ภายในเวลาที่กำหนด ถ้าผู้เล่นยิงที่จุดสำคัญเช่น ที่หัวใจ หรือ ที่ศีรษะ ก็จะได้คะแนนมาก แต่ถ้ายิงโดนที่จุดอื่นก็จะได้คะแนนลดลงตามลำดับ ถ้ายิงพลาดก็จะโดนหักคะแนน พอหมดเวลาก็จะรายงานว่าผู้เล่นสามารถยิงได้กี่คะแนน

<u>การควบคุมในเกม</u>

ในเกมนี้ผู้เล่นจะควบคุมเกมโดยใช้ Keyboard และ Mouse โดยมีปุ่มที่ใช้ในการควบคุมเกมดังนี้

การเปลี่ยนมุมมอง (Viewing):

- ปุ่ม A: เงยหน้า

ใช้ในการเงยหน้ามองขึ้นไปข้างบน สามารถใช้ Mouse เลื่อนขึ้นทางด้านบนสุดขอบจอแทนก็ได้

- ปุ่ม Z: ก้มหน้า

ใช้ในการก้มหน้ามองลงไปข้างล่างหรือที่พื้น สามารถใช้ Mouse เลื่อนลงทางด้านล่างสุดขอบจอแทนก็ได้

- ปุ่ม 7: หันซ้าย

ใช้ในการหันหาเป้าหมายทางด้านซ้าย สามารถใช้ Mouse เลื่อนไปทางด้านซ้ายสุดขอบจอแทนก็ได้ ซึ่งการใช้ Mouse จะช่วยอำนวยสะดวกในการเล็งหาเป้าหมาย

- ปุ่ม 9: หันขวา

ใช้ในการหันหาเป้าหมายทางด้านขวา สามารถใช้ Mouse เลื่อนไปทางด้านขวาสุดขอบจอแทนก็ได้ ซึ่งการใช้ Mouse จะช่วยอำนวยสะดวกในการเล็งหาเป้าหมาย

การเคลื่อนที่ (Movement):

- ปุ่ม 4: ขยับไปทางซ้าย (left shift)
- ปุ่ม 6: ขยับไปทางขวา (right shift)
- ปุ่ม 8: เดินหน้า (move forward)
- ปุ่ม 2: ถอยหลัง (move backward)

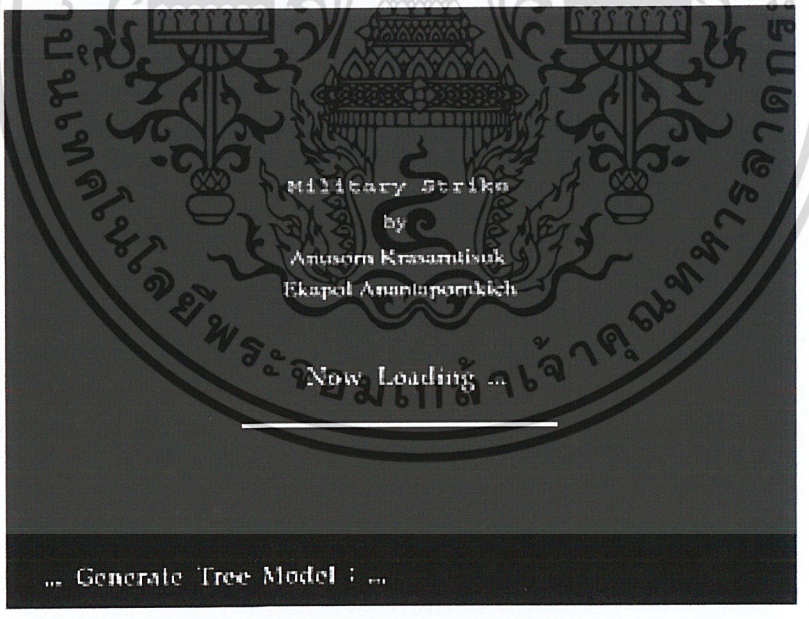
ปุ่มควบคุมการเคลื่อนที่เหล่านี้เมื่อกดค้างไว้จะมีความเร่งสูงขึ้นเรื่อย ๆ

การยิง:

- Right Mouse Button:

ยิงปืนยาวระยะไกล ผู้เล่นจะต้องเล็งไปยังทหารที่เป็นเป้าหมายที่ต้องการกำจัด โดยใช้การควบคุม Mouse ไปหาเป้าหมายที่ต้องการยิง โดย Mouse Cursor จะเป็นรูปศูนย์เล็งสำหรับปืนยาว

เมื่อเริ่มต้นเปิดโปรแกรมจะขึ้นหน้าจอ แสดงการดำเนินการอ่านข้อมูลฉากขึ้นมาประมวลผลในเกมดังแสดงในรูป

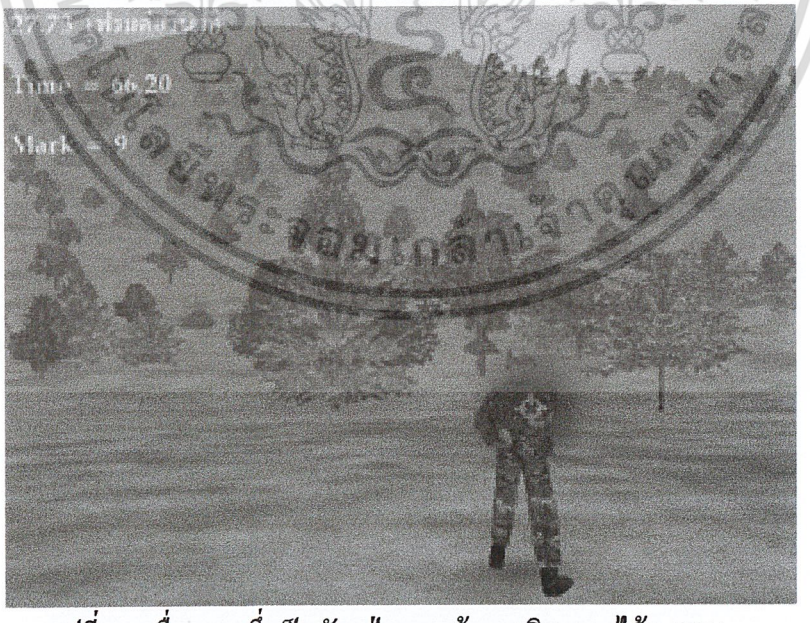


รูปที่ 6-1 หน้าจอแรกเมื่อเริ่มต้นเกม

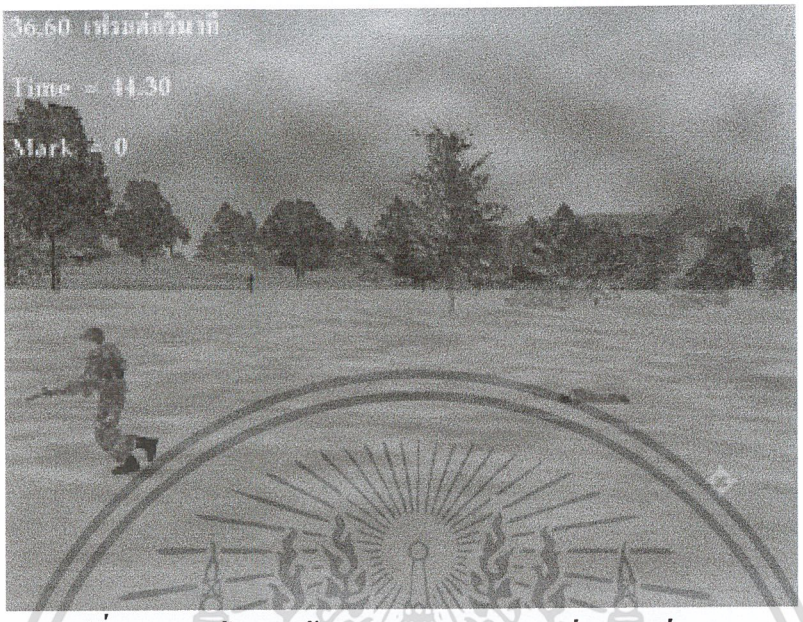
Screenshots

รูปต่าง ๆ นี้เป็นรูป Screenshot ของตัวเกม Military Strike

รูปที่ 6-2 โฉมหน้าของทหารซึ่งเป็นศัตรูฝ่ายตรงข้าม



รูปที่ 6-3 เมื่อทหารซึ่งเป็นศัตรูฝ่ายตรงข้ามถูกยิงเราจะได้คะแนน



รูปที่ 6-4 ทหารฝ่ายตรงข้ามจะคอยลาดตระเวนอยู่ตามจุดต่าง ๆ

รูปที่ 6-5 แสดงบรรยากาศของเกม ที่เต็มไปด้วยทุ่งหญ้าและป่าเขารกทึบ

6.2 การนำเอนจินต์มาพัฒนาเกม

สำหรับเกมนี้เราใช้คอมโพเนนท์ต่าง ๆ ของเกมเอนจินต์ต่อไปนี้เป็นตัวหลักในการพัฒนาเกม

- ส่วนของกล้องเราใช้ฟังก์ชันคำนวณที่เกี่ยวข้องกับการควบคุมกล้องในสไตล์ First Person Shooting

- โปรแกรมนี้ใช้ความสามารถของเอนจินต์ในการแสดงผลภาพเคลื่อนไหวอย่างเต็มรูปแบบ ทั้งการใช้งาน Animation และ Model Data

- โปรแกรมนี้ใช้งานส่วนออดิโอเพื่อสร้างเสียงประกอบเท่านั้นเนื่องจากขาดดนตรีประกอบฉาก

- โปรแกรมมีการใช้งาน Math และ Utility อย่างเต็มที่ทั้งในการตรวจสอบวิถีการยิงว่าถูกเป้าหมายหรือไม่

6.3 ผลการทดสอบ

เกมนี้สามารถแสดงผลภาพได้ประมาณ 30-40 เฟรมต่อวินาที ที่ความละเอียดจอภาพ 800x600 ในโหมดสี 32 บิต ซึ่งนับว่าน่าพอใจระดับหนึ่ง การเคลื่อนไหวของทหารเป็นไปอย่างนิ่มนวลโดยสามารถดึงประสิทธิภาพการประมวลผลในส่วนของ Animation และ Model Data มาใช้ได้อย่างเต็มที่

บทที่ 7
บทวิจารณ์และสรุป

7.1 สรุปผลการวิจัย

จากการพัฒนาโปรแกรมพบว่าเอนจินต์ที่พัฒนาขึ้นสามารถทำงานได้รวดเร็วในระดับที่น่าพอใจพอสมควรแต่ก็ยังมีข้อจำกัดตรงที่การออกแบบตัวเกมรวมทั้งการนำโมเดลที่สร้างขึ้นมาไปใช้นั้นยังทำได้ค่อนข้างยากและไม่ค่อยสะดวกนัก ทำให้ไม่เหมาะสำหรับพัฒนาเกมที่มีกติกาซับซ้อนและมีโครงสร้างฉากที่ซับซ้อน

ตลอดเวลาที่ได้พัฒนาเกมเอนจินต์ขึ้นมานั้นมีอุปสรรคและปัญหามากมายรวมทั้งเวลาอันจำกัดและข้อมูลที่มีอยู่น้อยนิด โดยเฉพาะที่เป็นทฤษฎีที่เกี่ยวข้องกับการออกแบบและพัฒนาเกมโดยตรงนั้นหายากมาก

7.2 แนวทางในการพัฒนาต่อ

- สร้างเครื่องมือสำหรับช่วยเหลือในการแปลงข้อมูลให้เอนจินต์ใช้งานได้ในรูปแบบ Graphical User Interface
- การออกแบบ Script สำหรับใช้ในการปรับแต่งเกมโดยไม่ต้องแก้ไขโปรแกรม
- เพิ่มฟังก์ชันการตรวจสอบการชนในระดับสูงขึ้นเพื่อให้สามารถสร้างโปรแกรมที่มีการโต้ตอบในระดับที่มีความเป็นธรรมชาติมากขึ้น
- เพิ่มระบบจัดการการจัดเก็บฉากที่มีขนาดใหญ่อย่างมีประสิทธิภาพขึ้นกว่าเดิม
- ปรับปรุงให้สามารถแสดงผลภาพได้เร็วขึ้น
- เพิ่มคุณสมบัติการเล่นเกมผ่านเน็ตเวิร์ค

ภาคผนวก

ภาคผนวก ก

การติดตั้ง 3D Studio MAX บนระบบปฏิบัติการวินโดว์

1. เตรียมการติดตั้ง

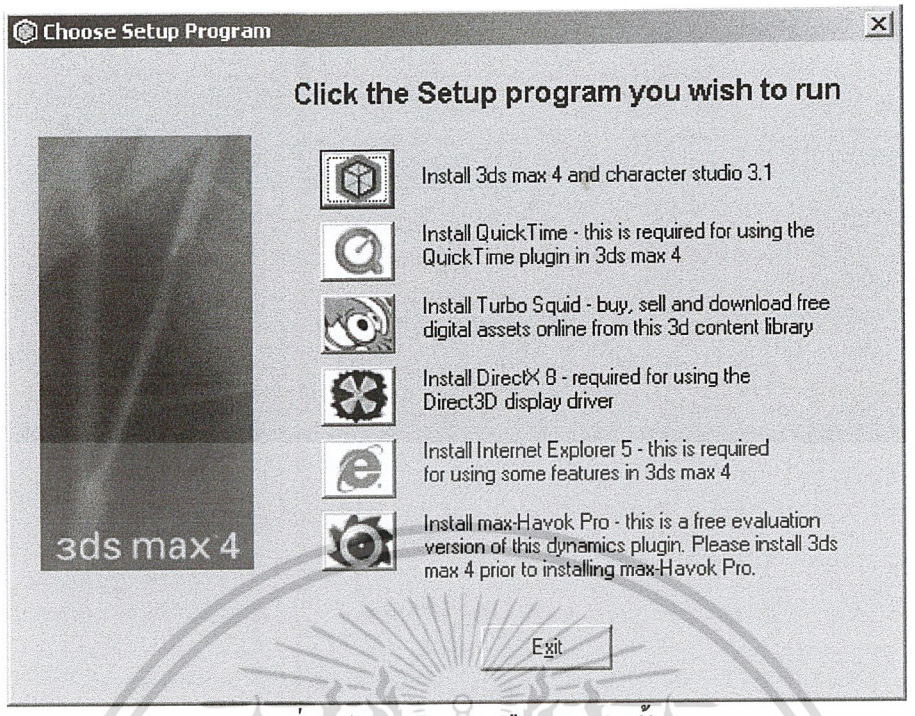
ในการติดตั้งบน Windows ทั้ง Windows 98/2000/NT/ME สามารถติดตั้งได้ โดยก่อนอื่นต้องสำรวจความพร้อมของทั้งฮาร์ดแวร์ และซอฟต์แวร์ต่างๆ ดังนี้.

ฮาร์ดแวร์/ซอฟต์แวร์	รายละเอียด
ซีพียู	Intel Pentium 233 MHz ขึ้นไป หรือซีพียู Compatible อื่นๆ เช่น AMD, Cyrix
แรม	หน่วยความจำ RAM ขนาดอย่างน้อย 64 MB แนะนำว่าควรเป็นขนาด 128 MB ขึ้นไป
ฮาร์ดดิสก์	ควรพื้นที่ว่างในการติดตั้งประมาณ 500 MB ทั้งนี้ยังไม่ได้รวมพื้นที่ชั่วคราวในการทดสอบระหว่างใช้งานอีกประมาณ 500 MB ต้องใช้พื้นที่ประมาณ 1 GB
ไดรว์ซีดีรอม	ต้องมี
การ์ดแสดงผล	ควรเป็นการ์ดแสดงผลที่สามารถแสดงภาพ 3 มิติได้
จอมอนิเตอร์	ควรเป็นจอขนาด 15 นิ้ว แนะนำให้ใช้ 17 นิ้วเพื่อความสะดวกในการเขียนโปรแกรม
บราวเซอร์	ต้องติดตั้งบราวเซอร์ Internet Explorer 4.01 ขึ้นไป

2. ขั้นตอนการติดตั้ง

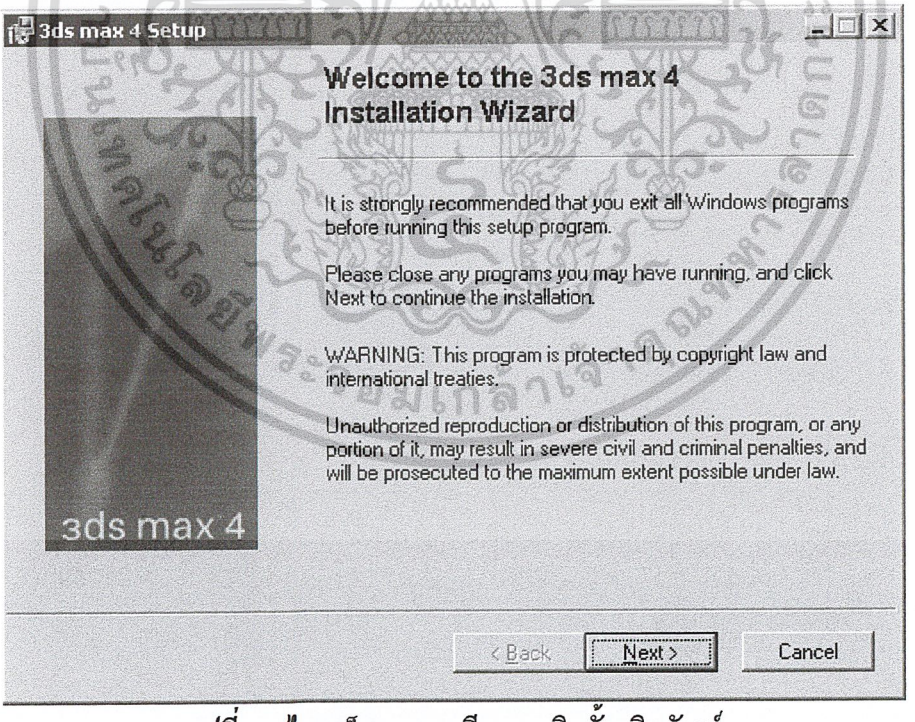
สำหรับขั้นตอนในการติดตั้งมีรายละเอียดดังนี้

2.1 นำแผ่นติดตั้งของ Visual C++ 6.0 เข้าไปในซีดีรอมไดรว์ รอสักครู่มันจะเรียกโปรแกรมเซตอัพโดยอัตโนมัติ



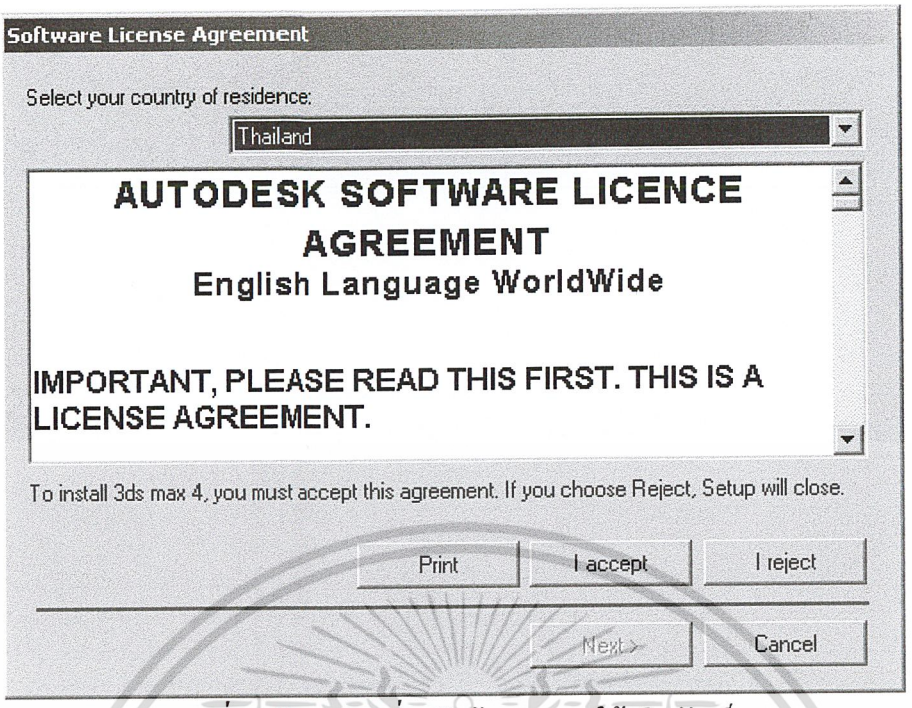
รูปที่ ก-1 ไดอะล็อกการเลือกชนิดติดตั้ง

2.2 คลิกเลือกไอคอนการติดตั้งโปรแกรม 3D Studio MAX 4



รูปที่ ก-2 ไดอะล็อกรายละเอียดการติดตั้งผลิตภัณฑ์

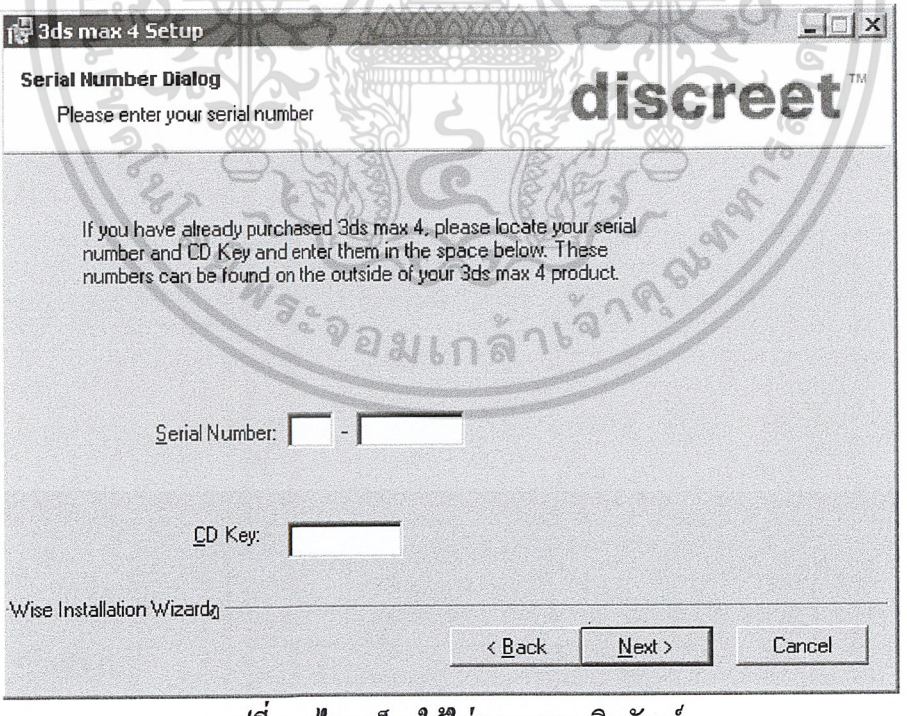
2.3 คลิกเลือก Next > เพื่อติดตั้งโปรแกรมจะขึ้นไดอะล็อกเกี่ยวกับข้อตกลงการใช้ผลิตภัณฑ์นี้ คลิกที่หัวข้อ I accept



รูปที่ ก-3 ไดอะล็อกเกี่ยวกับข้อตกลงการใช้ผลิตภัณฑ์

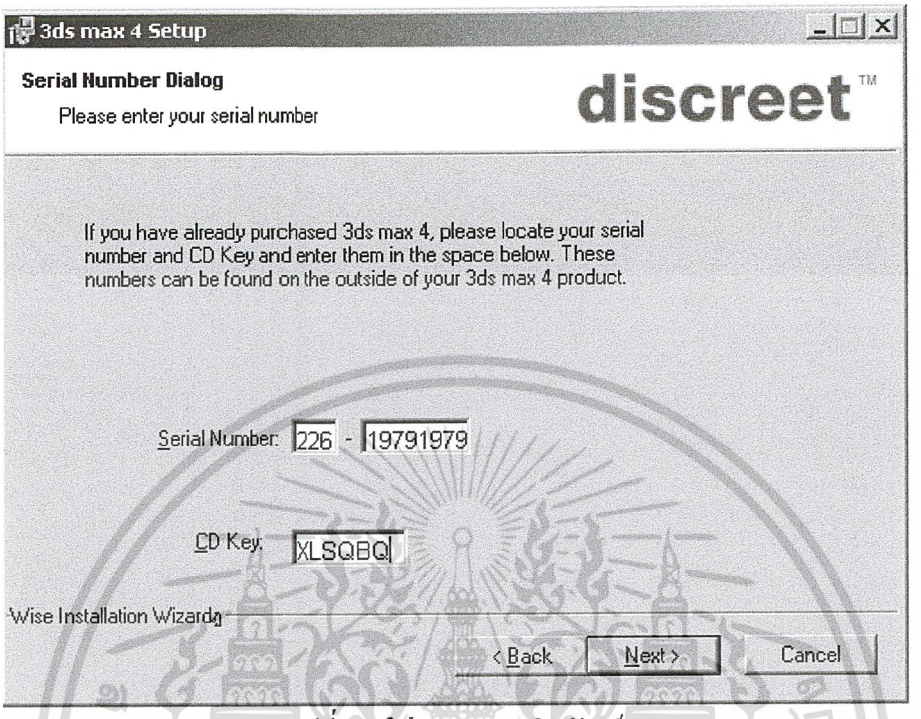
2.4 จากนั้นคลิกปุ่ม Next > อีกครั้ง

2.5 พิมพ์ข้อมูลหมายเลขของผลิตภัณฑ์นี้



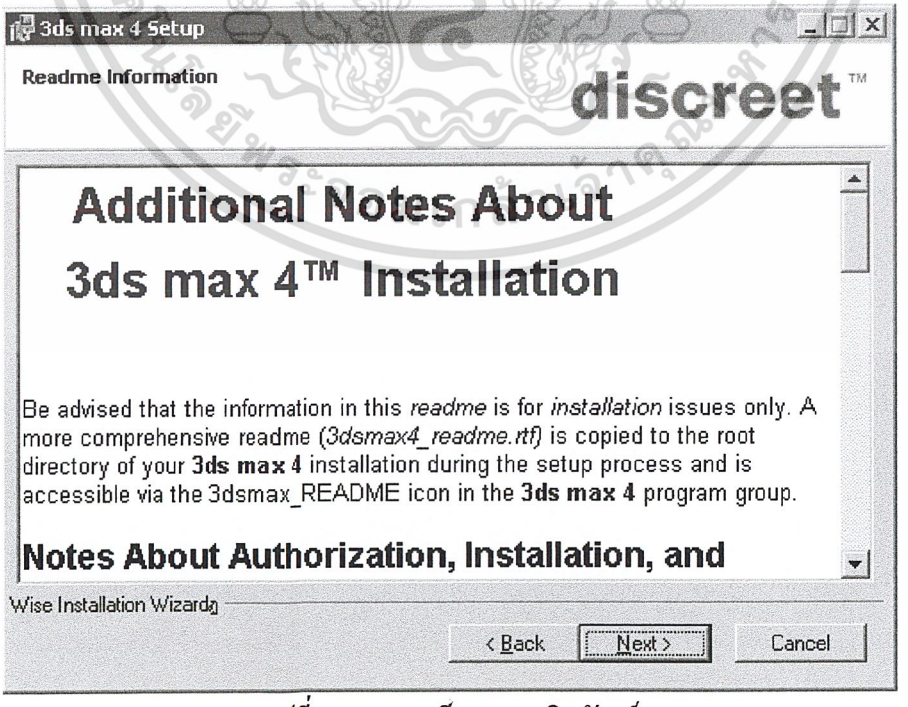
รูปที่ ก-4 ไดอะล็อกให้ใส่หมายเลขผลิตภัณฑ์

2.6 ใส่หมายเลขของผลิตภัณฑ์ แล้วคลิกปุ่ม Next > รายละเอียดของหมายเลขผลิตภัณฑ์สามารถดูได้จาก CD-ROM ของ 3D Studio MAX



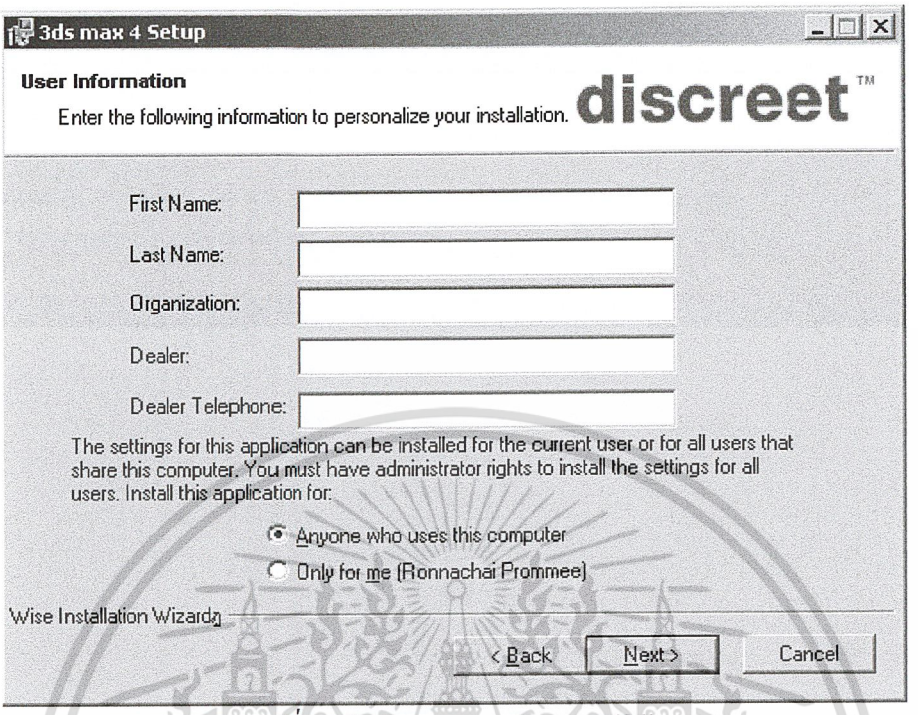
รูปที่ ก-5 ใส่หมายเลขผลิตภัณฑ์

2.7 จากนั้นจะขึ้นไดอะล็อกข้อมูลรายละเอียดของผลิตภัณฑ์ แล้วคลิกปุ่ม Next >



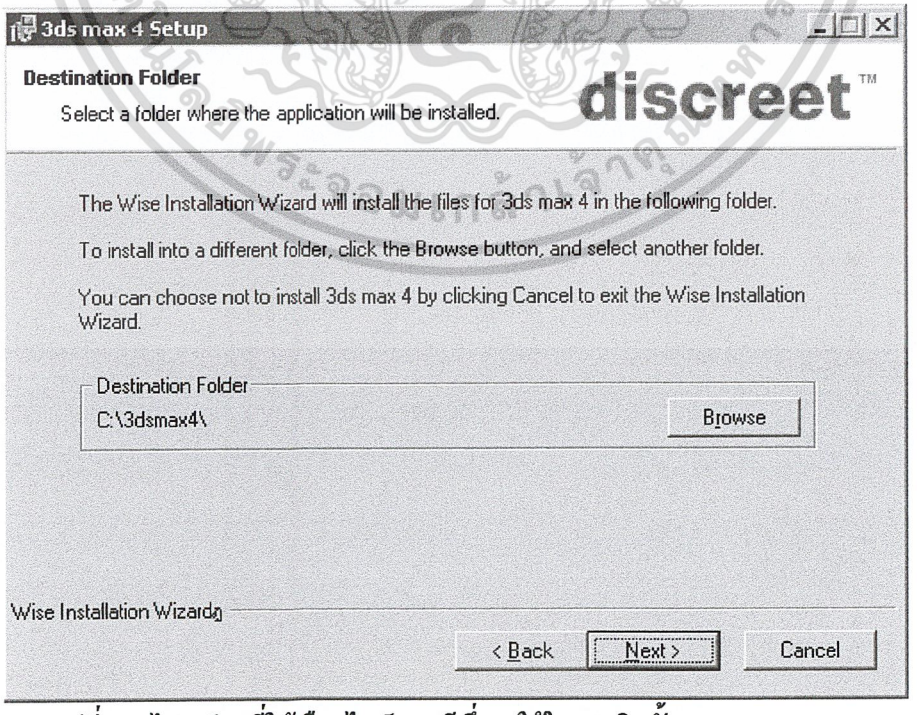
รูปที่ ก-6 รายละเอียดของผลิตภัณฑ์

2.8 จากนั้นจะขึ้นไดอะล็อกให้พิมพ์ข้อมูลรายละเอียดของผู้ใช้ แล้วคลิกปุ่ม Next >



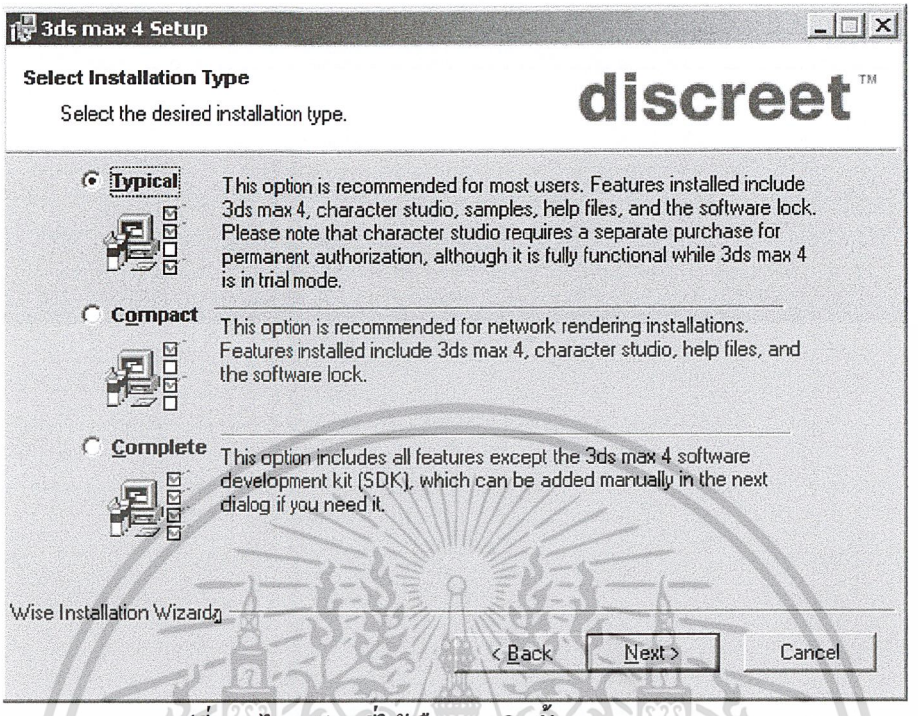
รูปที่ ก-7 ใส่ข้อมูลรายละเอียดของผู้ใช้

2.9 จากนั้นจะถามไดเร็คทอรีที่จะลงโปรแกรมในการติดตั้ง โดยถ้าไม่ต้องการไดเร็คทอรีดีฟอลต์นี้ ให้คลิกปุ่ม Browse...



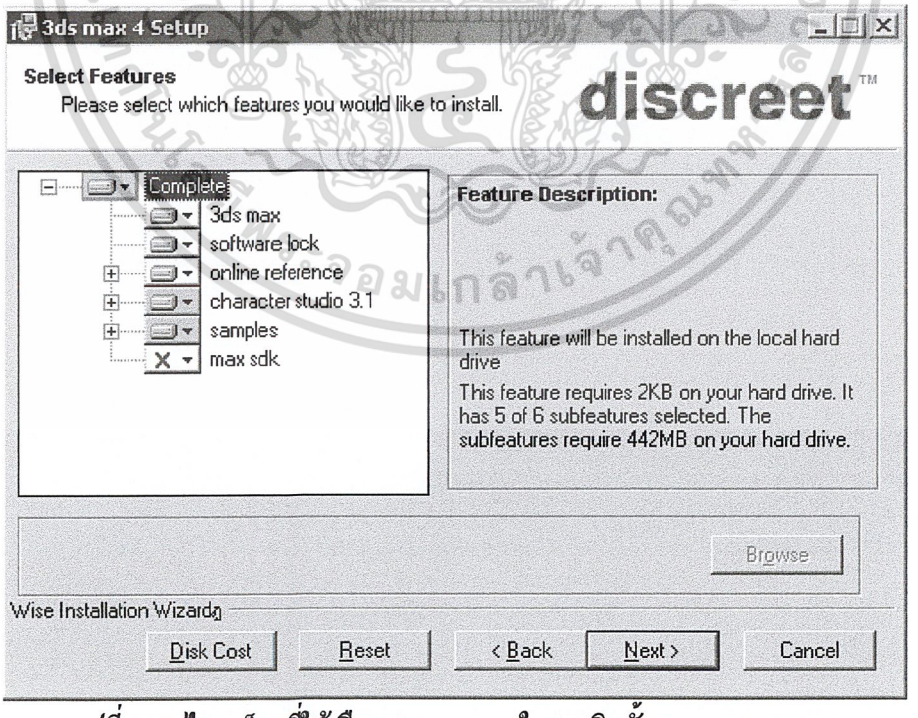
รูปที่ ก-8 ไดอะล็อกที่ให้เลือกไดเร็คทอรีซึ่งจะใช้ในการติดตั้ง 3D Studio MAX

2.10 จากนั้นจะให้เลือกชนิด Typical ในการติดตั้ง 3D Studio MAX แล้วคลิกปุ่ม 



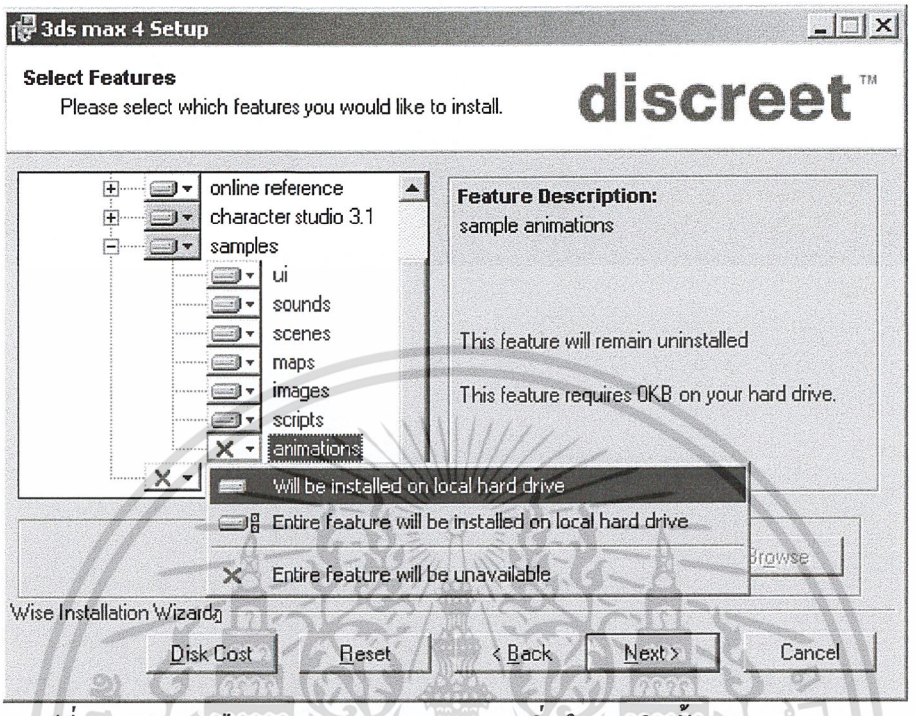
รูปที่ ก-9 ไดอะล็อกที่ให้เลือกการติดตั้ง 3D Studio MAX

2.11 จากนั้นจะให้เลือก Component ในการติดตั้ง 3D Studio MAX



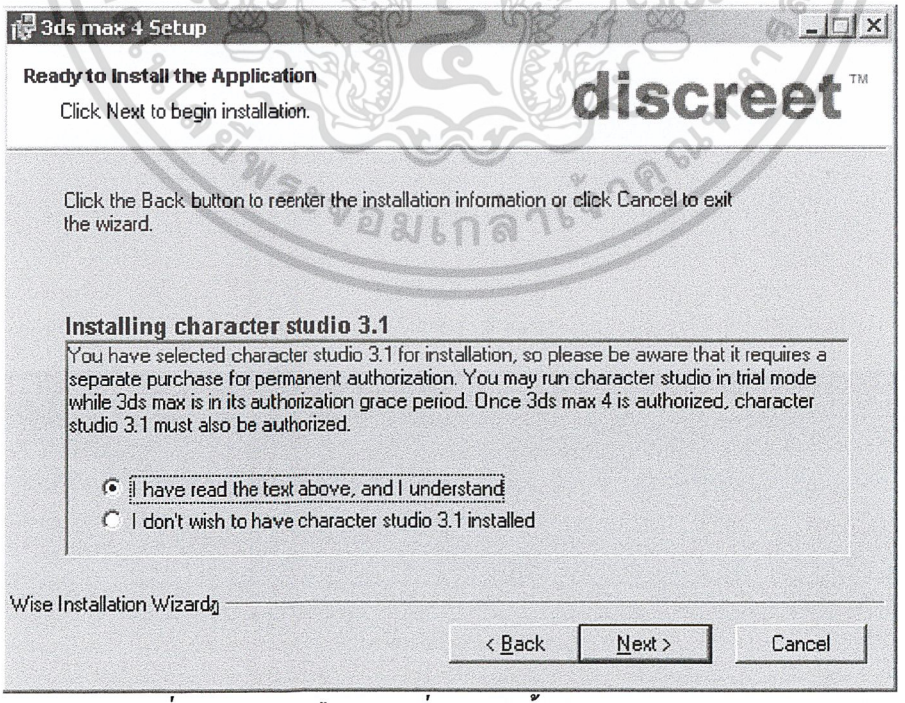
รูปที่ ก-10 ไดอะล็อกที่ให้เลือก Component ในการติดตั้ง 3D Studio MAX

2.12 ให้เลือก Component ในการติดตั้ง 3D Studio MAX ที่เป็น Animation เพิ่ม โดยเลือกเป็น Will be installed on local hard drive แล้วคลิกปุ่ม Next >

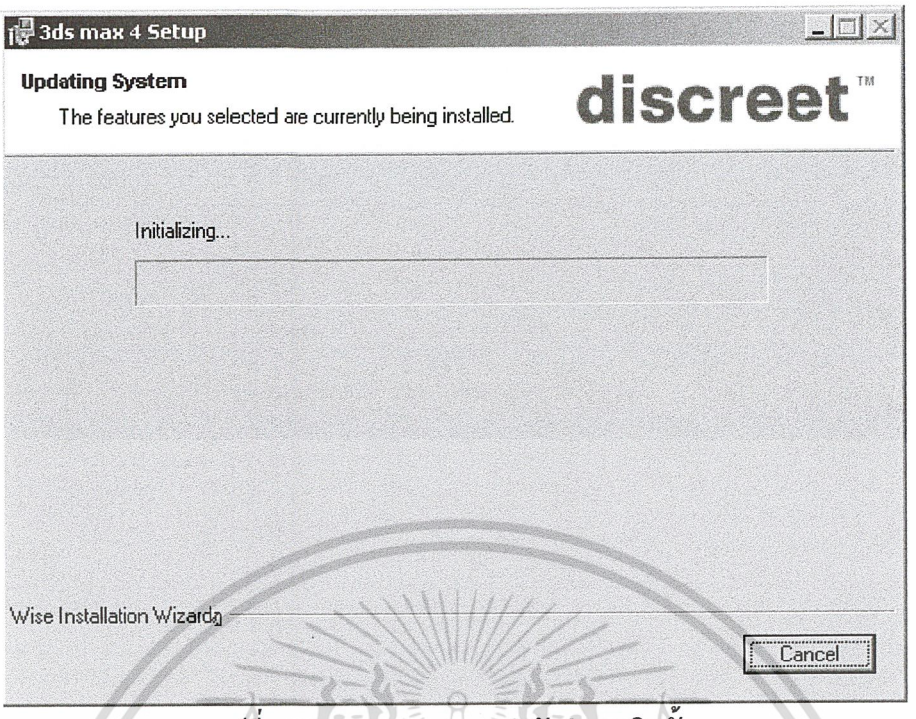


รูปที่ ก-11 ทำการเลือก Component Animation เพิ่ม ในการติดตั้ง 3D Studio MAX

2.13 จากนั้นจะขึ้นไดอะล็อกให้เลือกการติดตั้ง Character studio 3.1 ให้คลิกปุ่ม Next >

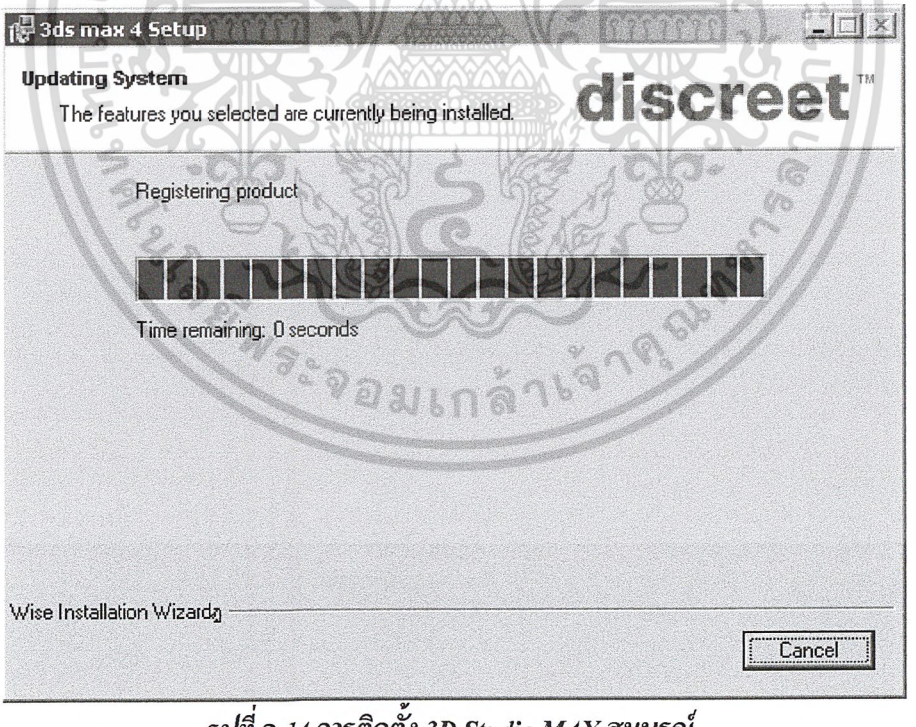


รูปที่ ก-12 ทำการเลือกการ เพิ่มการติดตั้ง Character studio 3.1

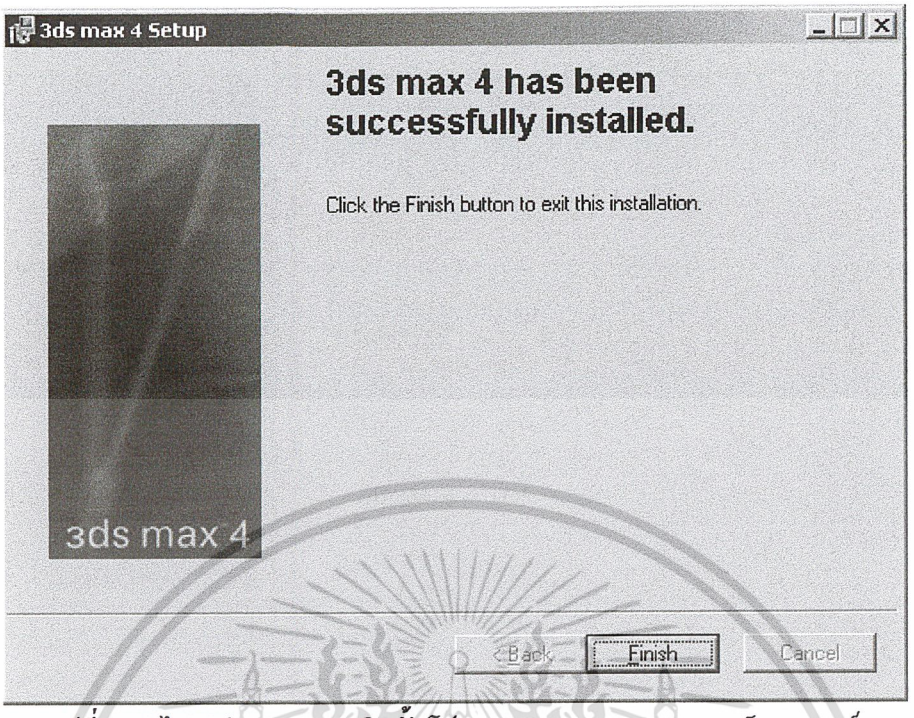


รูปที่ ก-13 3D Studio MAX กำลังทำการติดตั้ง

2.14 จากนั้นจะขึ้นไดอะล็อกแสดงสภาวะการติดตั้งโปรแกรม 3D Studio MAX ให้รอสักครู่

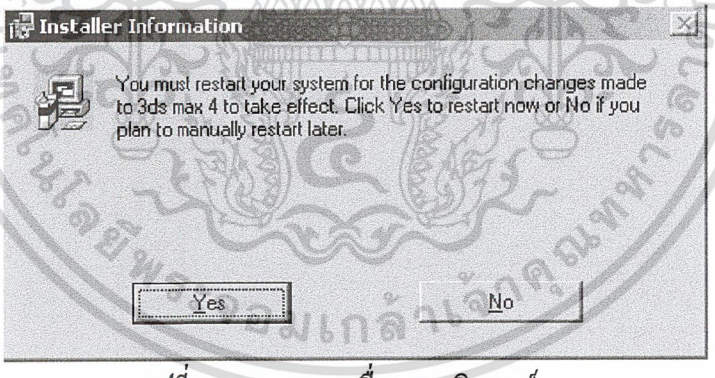


รูปที่ ก-14 การติดตั้ง 3D Studio MAX สมบูรณ์



รูปที่ ก-15 ไดอะล็อกแสดงการติดตั้ง โปรแกรม 3D Studio MAX เสร็จสมบูรณ์

2.15 การติดตั้งโปรแกรม 3D Studio MAX เสร็จสมบูรณ์ ให้คลิกที่ปุ่ม Finish แล้ว Restart เครื่อง



รูปที่ ก-16 Restart เครื่องคอมพิวเตอร์

ภาคผนวก ข

การติดตั้ง Microsoft Visual C++ 6.0

1. เตรียมการติดตั้ง

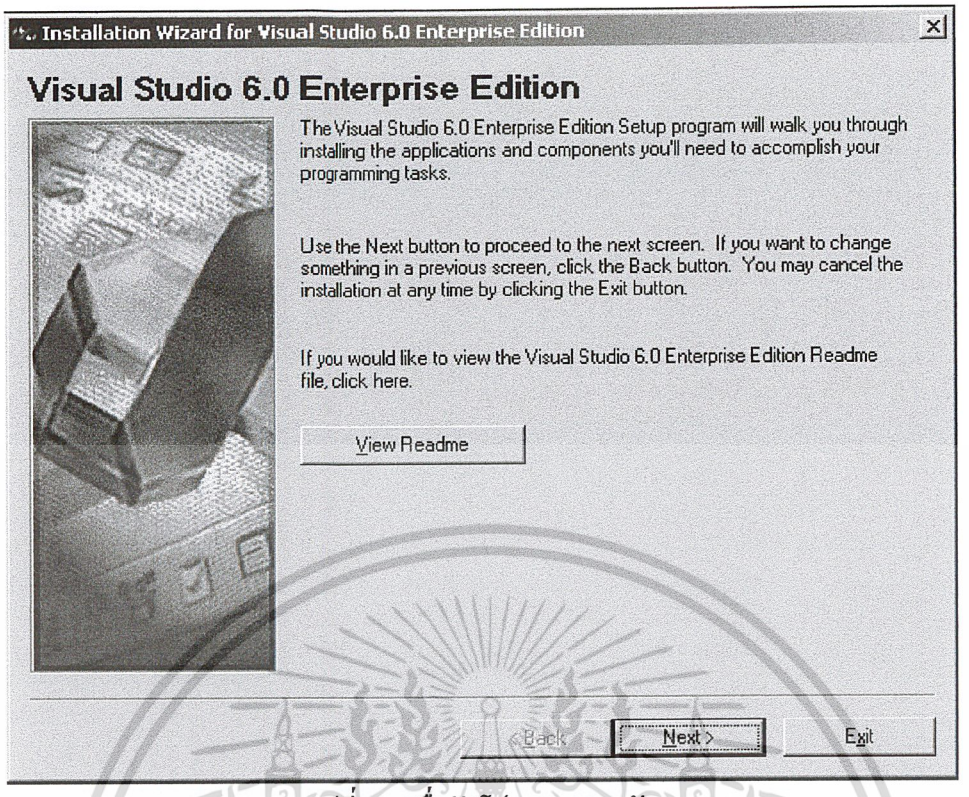
ก่อนการติดตั้งและใช้งาน Visual C++ 6.0 ต้องสำรวจความพร้อมของทั้งฮาร์ดแวร์ และซอฟต์แวร์ต่างๆ ดังนี้

ฮาร์ดแวร์/ซอฟต์แวร์	รายละเอียด
ซีพียู	Intel Pentium 166 MHz ขึ้นไป หรือซีพียู Compatible อื่นๆ เช่น AMD, Cyrix
แรม	หน่วยความจำ RAM ขนาดอย่างน้อย 32 MB แนะนำว่าควรเป็นขนาด 64 MB ขึ้นไป
ฮาร์ดดิสก์	ควรพื้นที่ว่างในการติดตั้งประมาณ 550 MB ทั้งนี้ยังไม่ได้รวมพื้นที่ชั่วคราวในการทดสอบ/สร้างแอพพลิเคชันอีกประมาณ 500 MB และการติดตั้งไลบรารี่ MSDN (Microsoft Developer Network) ซึ่งถ้าติดตั้ง MSDN ที่สมบูรณ์ ต้องใช้พื้นที่ประมาณ 1.2 GB
ไดรว์ซีดีรอม	ต้องมี
จอมอนิเตอร์	ควรเป็นจอขนาด 15 นิ้ว แนะนำให้ใช้ 17 นิ้วเพื่อความสะดวกในการเขียนโปรแกรม
ระบบปฏิบัติการ	Windows 95, Windows 98, Windows NT หรือ Windows 2000
บราวเซอร์	ต้องติดตั้งบราวเซอร์ Internet Explorer 4.01 ขึ้นไป

2. ขั้นตอนการติดตั้ง

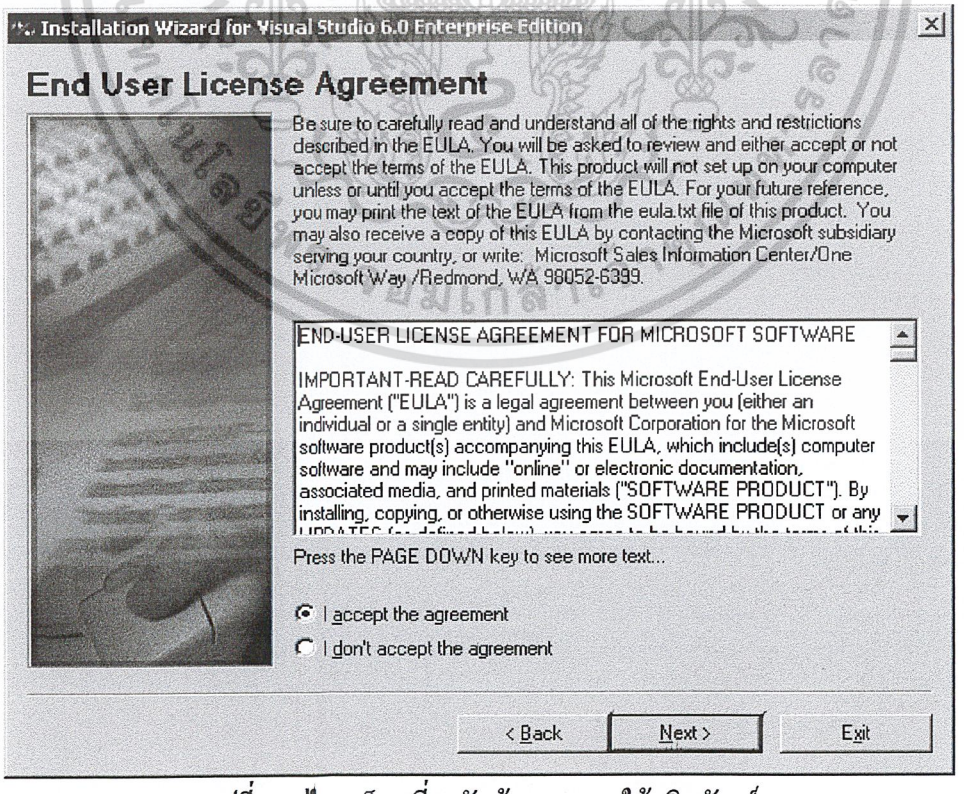
สำหรับขั้นตอนในการติดตั้งมีรายละเอียดดังนี้

2.1 นำแผ่นติดตั้งของ Visual C++ 6.0 เข้าไปในซีดีรอมไดรว์ รอสักครู่มันจะเรียกโปรแกรมเซตอัพโดยอัตโนมัติ



รูปที่ ข-1 เมื่อรันโปรแกรมเซตอัพ

2.2 คลิกปุ่ม Next > จะขึ้นไดอะล็อกเกี่ยวกับข้อตกลงการใช้ผลิตภัณฑ์นี้ คลิกที่หัวข้อ I accept the agreement.

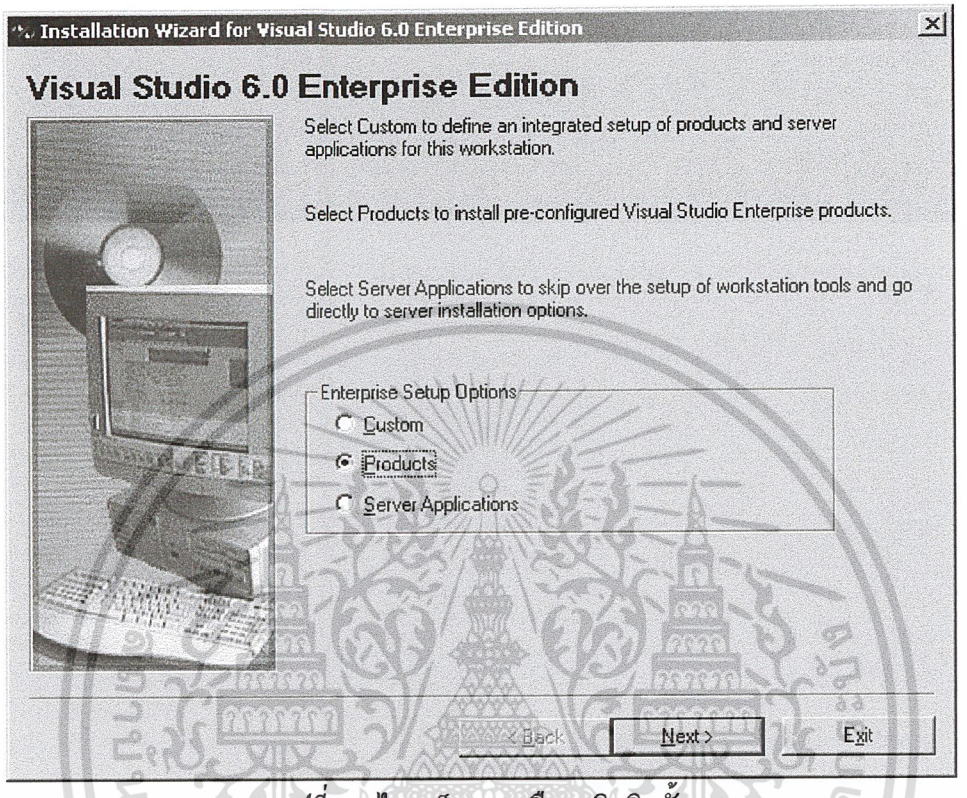


รูปที่ ข-2 ไดอะล็อกเกี่ยวกับข้อตกลงการใช้ผลิตภัณฑ์

2.3 จากนั้นคลิกปุ่ม Next > อีกครั้ง

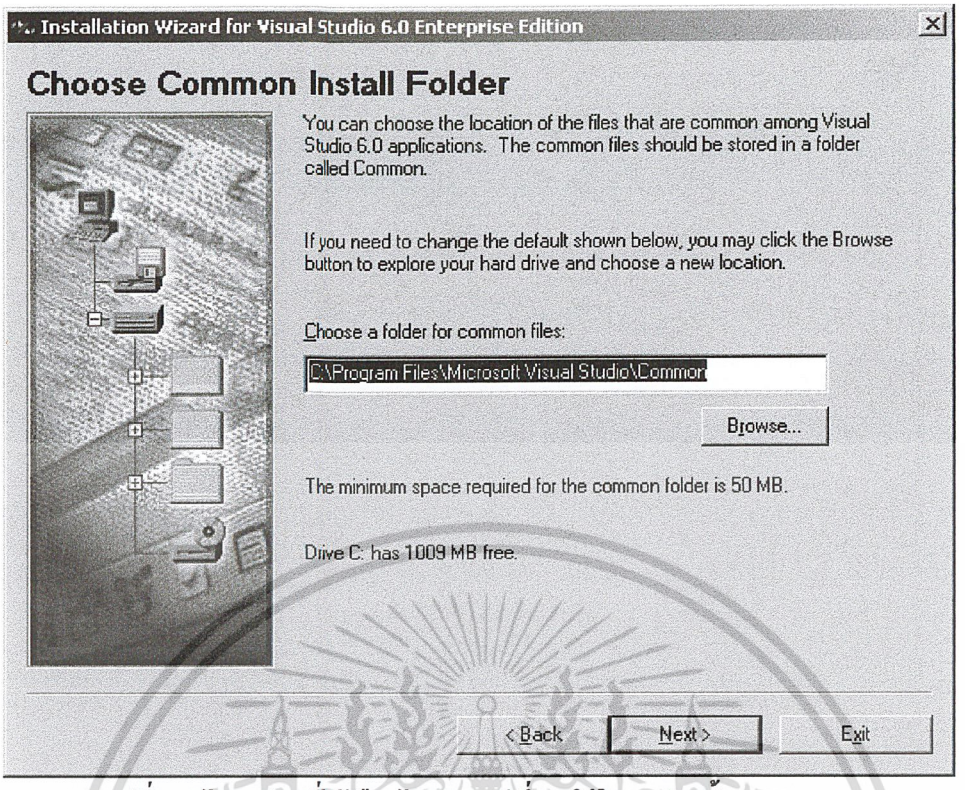
2.4 พิมพ์ข้อมูลรายละเอียดเกี่ยวกับผลิตภัณฑ์นี้

2.5 คลิกปุ่ม Next > เพื่อเลือกชนิดการติดตั้ง



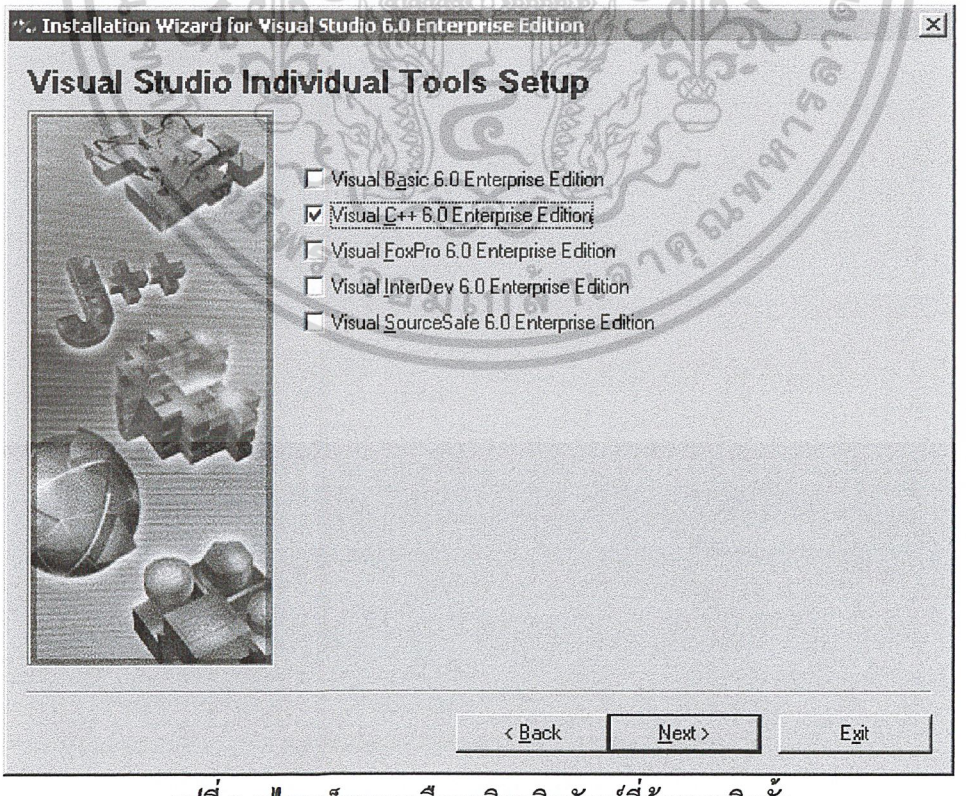
รูปที่ ข-3 ไดอะล็อกการเลือกชนิดติดตั้ง

2.6 คลิกที่หัวข้อ Products และคลิกปุ่ม Next > แล้วจะถามไดเร็คทอรีที่จะลงโปรแกรมในการติดตั้ง โดยถ้าไม่ต้องการไดเร็คทอรีดีฟอลต์นี้ ให้คลิกปุ่ม Browse...



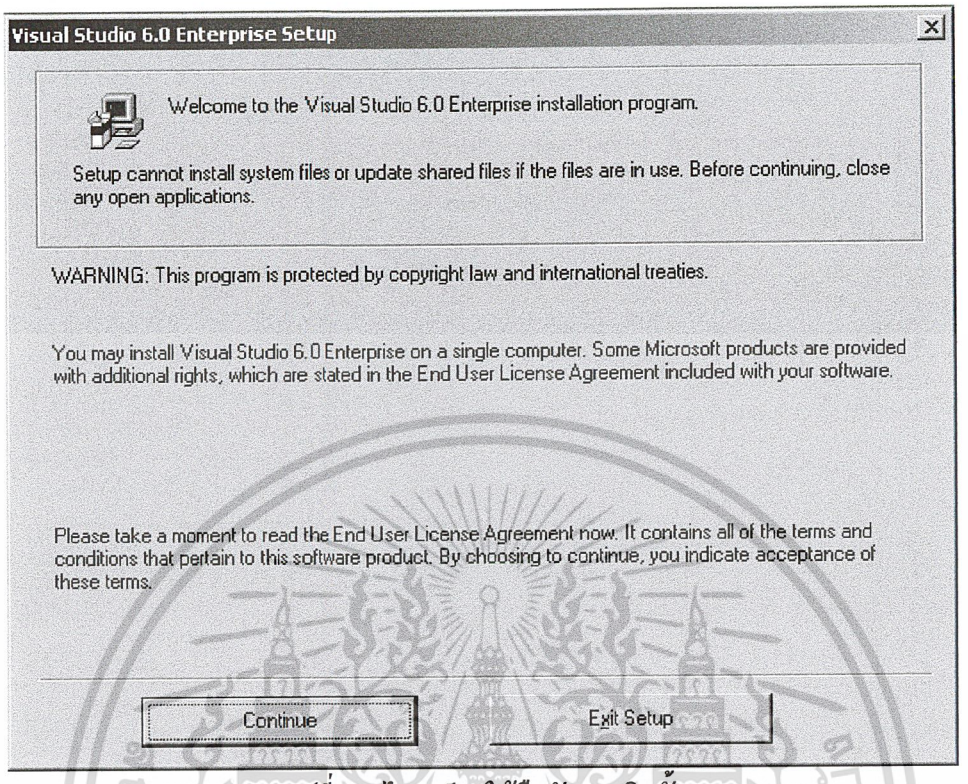
รูปที่ ข-4 ไดอะล็อกที่ให้เลือกไดเร็คทอรีซึ่งจะใช้ในการติดตั้ง Visual C++

2.7 เมื่อเลือกไดเร็คทอรีได้แล้ว คลิกปุ่ม Next > อีกครั้ง มันก็จะขึ้นไดอะล็อกชนิดผลิตภัณฑ์ที่ต้องการติดตั้ง ให้เลือก Visual C++ 6.0 Enterprise Edition



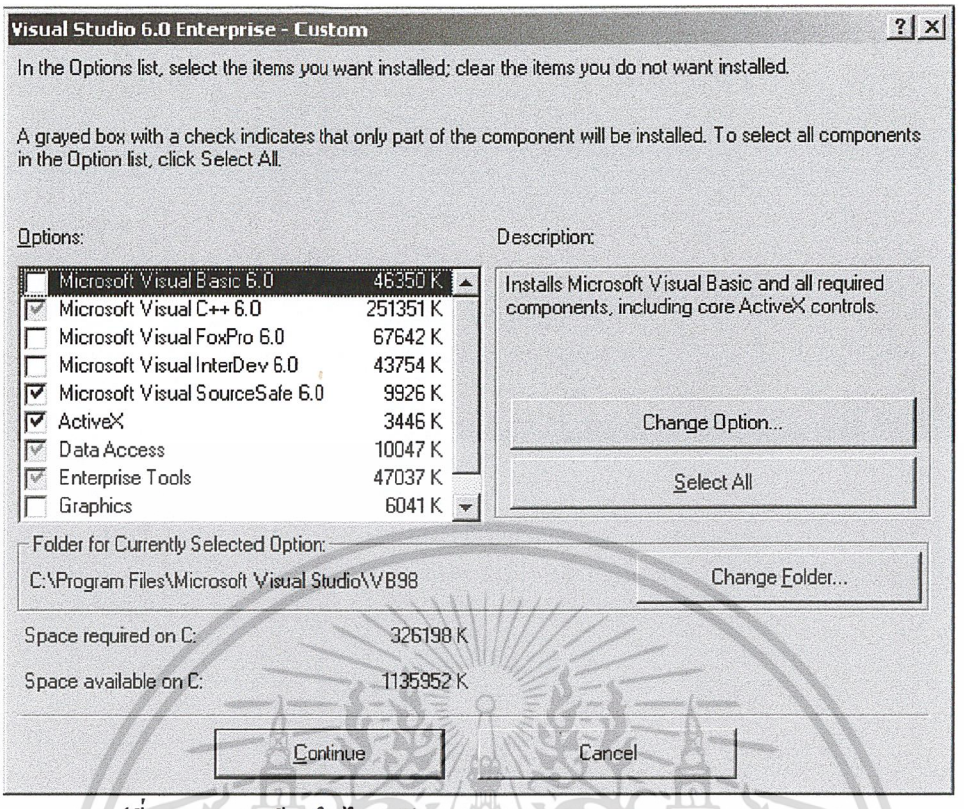
รูปที่ ข-5 ไดอะล็อกการเลือกชนิดผลิตภัณฑ์ที่ต้องการติดตั้ง

2.8 คลิกปุ่ม Next > จะขึ้นไดอะล็อกให้ยืนยันการติดตั้ง แล้วคลิกปุ่ม Continue



รูปที่ ข-6 ไดอะล็อกให้ยืนยันการติดตั้ง

2.9 เมื่อคลิกปุ่ม Continue แล้วโปรแกรมเซตอัพจะขึ้นไดอะล็อกใหม่เป็น Visual Studio 6.0 Enterprise – Custom



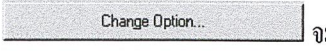
รูปที่ ข-7 รายละเอียดในไดอะล็อก Visual Studio 6.0 Enterprise - Custom

คอมโพเนนต์ที่แสดงอยู่ในไดอะล็อกจะรวมสภาพแวดล้อมหลักสำหรับการพัฒนาทั้งหมดไว้ ซึ่งมาพร้อมกับ Microsoft Visual Studio 6.0 (Microsoft Visual Basic 6.0, Microsoft Visual C++ 6.0 และอื่นๆ) บวกด้วยเครื่องมือต่างๆ และ Accessory ดังต่อไปนี้

- ActiveX
- Data Access
- Enterprise Tools
- Graphics
- Tools

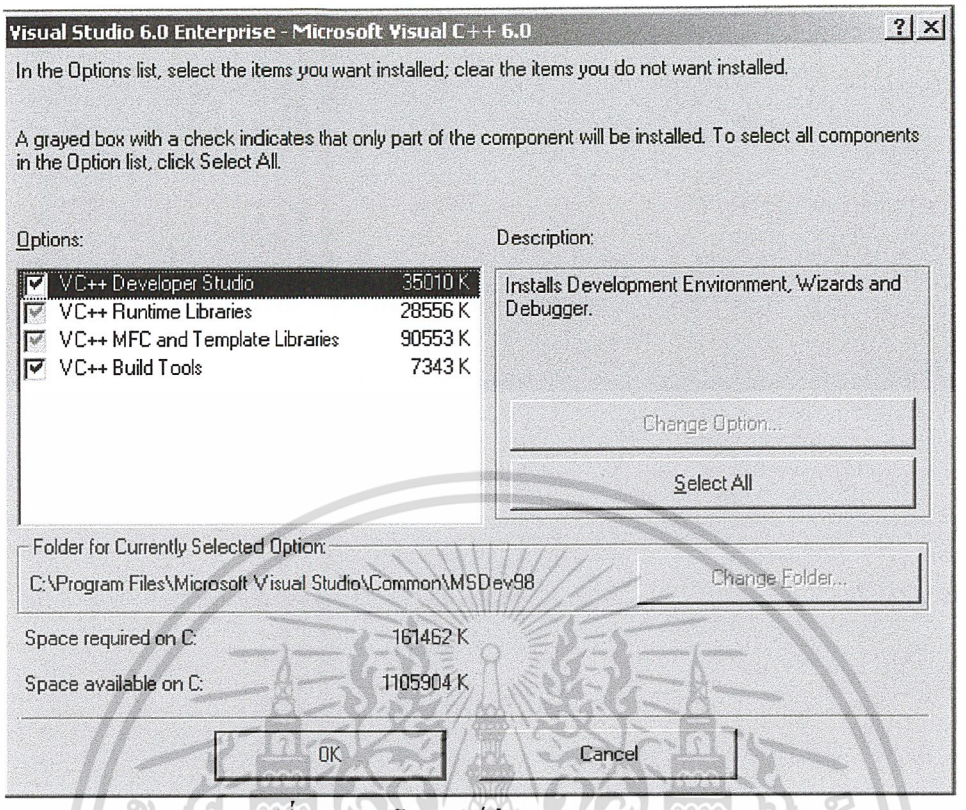
คลิกแต่ละคอมโพเนนต์ที่เราต้องการรวมเข้าไว้ในการติดตั้งนี้ และถ้าต้องการติดตั้งคอมโพเนนต์ทั้งหมด เราก็คลิกที่ปุ่ม Select All (ในที่นี้เราไม่คลิกรายการ Microsoft Visual Basic 6.0, FoxPro 6.0, InterDev 6.0 และ SourceSafe 6.0 นอกนั้นเราคลิกหมด)

แต่ละคอมโพเนนต์ของ Visual Studio ประกอบด้วยกลุ่มคอมโพเนนต์ย่อยอีก ซึ่งสามารถเลือกรายการเดียวก็ได้ (ยกเว้น ActiveX จะไม่มีคอมโพเนนต์ย่อย)

รายการที่เลือกแล้วแสดงว่าคอมโพเนนต์นั้นจะถูกติดตั้ง แต่ไม่ทั้งหมด เพื่อที่จะระบุว่าคอมโพเนนต์ไหนจะถูกติดตั้ง ใช้เมาส์คลิกรายการนั้น ต่อไปคลิกปุ่ม 

จะปรากฏไดอะล็อกใหม่และเลือกคอมโพเนนต์ย่อยที่เราต้องการในไดอะล็อกใหม่นี้ ตัวอย่างเช่น ถ้าเราคลิกรายการ Microsoft Visual C++ 6.0 และคลิกปุ่ม 

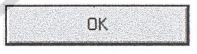
จะปรากฏไดอะล็อกใหม่ดังนี้



รูปที่ ข-8 คอมโพเนนต์ต่างๆ ของ Visual C++

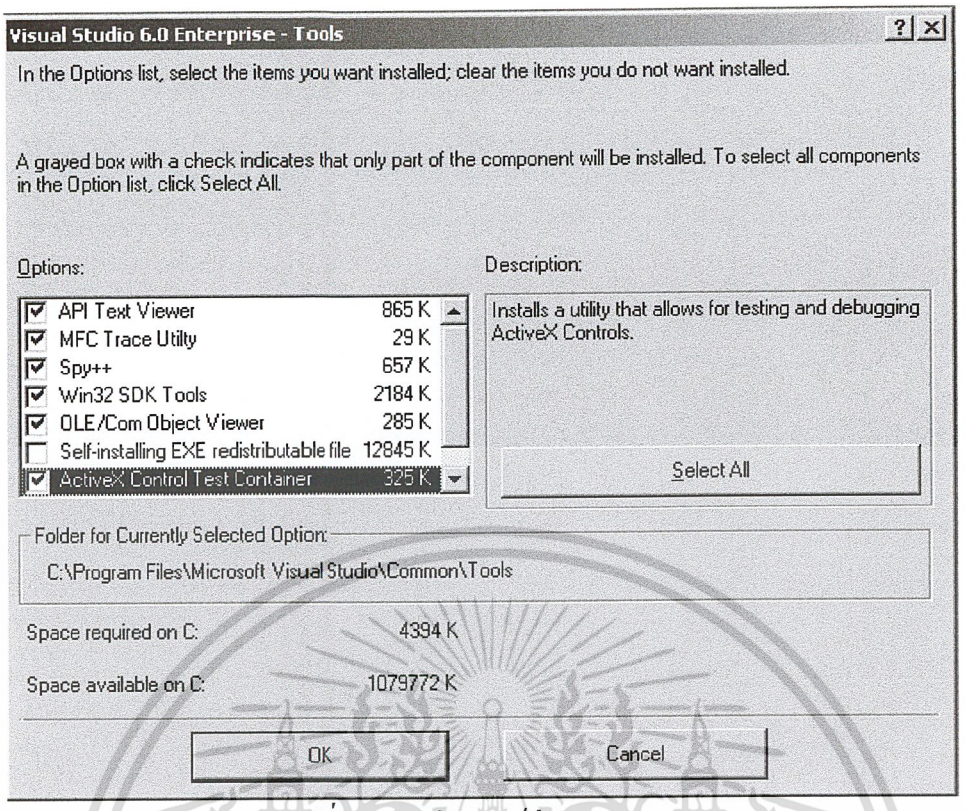
จากรูปนี้ แสดงรายการคอมโพเนนต์ย่อยที่สามารถติดตั้งได้ ซึ่งก็คือการระบุลงใน Visual C++ คอมโพเนนต์ย่อยมีดังนี้

- VC++ Developer Studio
- VC++ Runtime Libraries
- VC++ MFC and Template Libraries
- VC++ Build Tools

ในที่นี้เลือกทุกคอมโพเนนต์ ซึ่งเมื่อเลือกแล้ว มันจะทำเครื่องหมายไว้แล้วคลิกปุ่ม 

ต่อไปคลิกที่รายการ Tools แล้วตามด้วยปุ่ม 

ซึ่งจะแสดงคอมโพเนนต์ย่อยต่างๆ

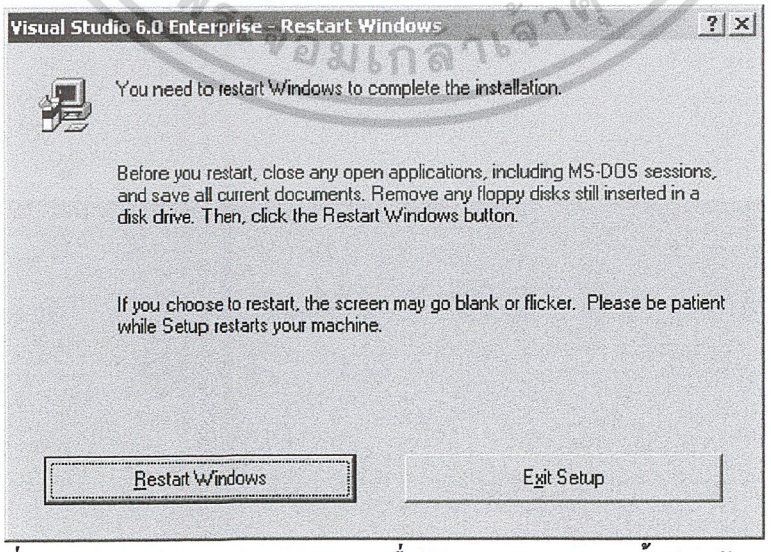


รูปที่ ข-9 คอมโพเนนต์ย่อยของ Tools

จากรูปข้างบนนี้เราต้องคลิกรายการ ActiveX Control Test Container ซึ่งใช้ในการทดสอบ ActiveX Control และคลิกปุ่ม OK มันจะกลับมายังไดอะล็อก Visual Studio 6.0 Enterprise – Custom (มีหลายๆ คอมโพเนนต์ประกอบด้วยคอมโพเนนต์ย่อยๆ อีก เราใช้วิธีการดังข้างต้นในการเลือกคอมโพเนนต์ที่ต้องการ)

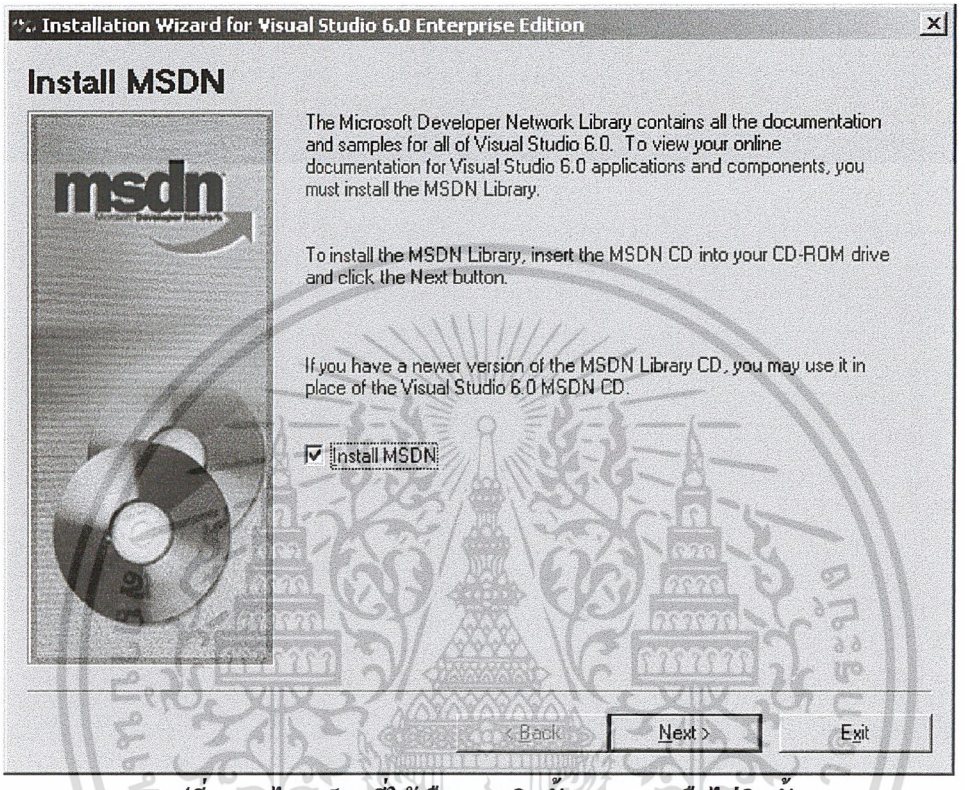
2.10 เมื่อคลิก Continue แล้วโปรแกรมเซตอัพจะเริ่มทำการติดตั้ง

2.11 เมื่อติดตั้งเรียบร้อยแล้ว โปรแกรมเซตอัพจะให้เราทำการคลิกปุ่ม Restart Windows



รูปที่ ข-10 ไดอะล็อก Restart Windows เมื่อโปรแกรมเซตอัพติดตั้งเรียบร้อยแล้ว

2.12 หลังจากที่ Restart Windows แล้ว มันจะให้เราติดตั้งไลบรารี MSDN ถ้าต้องการติดตั้ง คลิกที่รายการ Install MSDN และคลิกที่ปุ่ม Next > แต่ถ้าไม่ต้องการให้คลิกที่ปุ่ม Exit (แนะนำว่าควรติดตั้งไลบรารี MSDN เนื่องจากมีข้อมูลจำนวนมากที่เป็นประโยชน์ต่อการสร้างและการพัฒนาแอพพลิเคชัน)



รูปที่ ข-11 ไดอะล็อกที่ให้เลือกการติดตั้ง MSDN หรือไม่ติดตั้ง

2.13 ติดตั้งไลบรารี MSDN เสร็จแล้ว ต่อไปโปรแกรมเซตอัพจะติดตั้งเครื่องมือไคลเอนท์อื่นๆ

เมื่อทำการติดตั้ง Microsoft Visual C++ และคอมโพเนนต์ต่างๆ เรียบร้อยแล้ว ต่อไปถ้ามี Service Pack 5 ควรติดตั้งต่อ ในการติดตั้งนั้นให้ดูไฟล์ Readme ก่อนเพื่อติดตั้งได้อย่างถูกต้อง สำหรับ Service Pack 5 ถ้าไม่มี สามารถดาวน์โหลดได้จากโฮมเพจไมโครซอฟท์

บรรณานุกรม

[1] Peter Donnelly and DirectX Team (1998): "Inside DirectX", Microsoft Press, 3 1998.

[2] Andre LaMothe (1999): "Tricks of the Windows Game Programming Gurus", Sams, 9 1999.

[3] Mark DeLoura (2000): "Game Programming Gems", Charles River Media, 8 2000.

[4] Mark DeLoura (2001): "Game Programming Gems 2", Charles River Media, 8 2001.

[5] Adrain Perez, Dan Royer: "Advance 3-D Game Programming with DirectX 7.0", Wordware Publishing Inc., 6 2000.