การพัฒนาเกม 3 มิติ
3D Game Development

นายชินโชติ พิทยานนท์
นายณรงค์ วังชัยศรี

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2544

การพัฒนาเกม 3 มิติ

3D Game Development

โดย

นายชินโชติ พิทยานนท์

นายณรงค์ วังชัยศรี

อาจารย์ที่ปรึกษา

ดร. วรวัฒน์ ลิ้มโภคา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

ปริญญานิพนธ์ ปีการศึกษา 2544

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะ วิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาเกม 3 มิติ

3D Game Development

ผู้จัดทำ นายชินโชติ พิทยานนท์ รหัสประจำตัว 41014107

นายณรงค์ วังชัยศรี รหัสประจำตัว 41014125

อาจารย์ที่ปรึกษา

(ดร. วรวัฒน์ ลิ้มโภคา)

การพัฒนาเกม 3 มิติ

นาย ชินโชติ พิทยานนท์
นาย ณรงค์ วังชัยศรี
ดร. วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษา
ปีการศึกษา 2544

บทคัดย่อ

ขณะนี้ตลาดวงการเกม PC ได้ก้าวหน้าขึ้นมาก ยอดการขายสูงขึ้น ประชาชนทั่วไปต่างนิยมเล่นกันแพร่หลายทั้งตามร้านค้าให้เช่าและตามบ้านเรือนต่างๆ อย่างไรก็ตามตลาดเกม PC ของเมืองไทยก็ยังมีการเติบโตที่น้อยมาก แม้ว่าจะมีเกมภาษาไทยเพิ่มมากขึ้นแต่ก็เป็นเพียงที่ตัวแทนจำหน่ายนำเข้ามาเท่านั้นการพัฒนาเกมของคนไทยเองจริงๆนั้นยังเพิ่งอยู่ในระยะเริ่มต้น

จึงเป็นโอกาสดีที่จะได้ทำการศึกษาการพัฒนาเกมในระดับเริ่มต้นคือตั้งแต่ระดับเอ็นจิ้นของโปรแกรมโมเดลที่ใช้ ระบบเสียงระบบเอาต์พุตและอินพุตของเกมในระบบเกม3 มิติที่ใช้ไดเร็กเอ็กซ์เป็นตัวกลางการเชื่อมต่อกับฮาร์ดแวร์ของระบบ

วิทยานิพนธ์ฉบับนี้เป็นการศึกษาถึงการสร้างโมเดลและอนิเมชั่นเพื่อนำไปใช้ในเกมที่เราได้สร้างขึ้น โดยเริ่มจากการศึกษาความสามารถของโปรแกรมสตูดิโอแม็กซ์ (STUDIO MAX) ซึ่งเป็นโปรแกรมที่แพร่หลายทั่วไปสำหรับการสร้างโมเดล3มิติ และการนำโมเดล3มิติที่สร้างสำเร็จแล้วไปใช้ในเกมของเรา

ในวิทยานิพนธ์จะพูดถึงการสร้างวัตถุ3มิติ เทคนิคที่ใช้โดยทั่วไปในการสร้างวัตถุ 3มิติอย่างง่ายๆเพื่อใช้สำหรับโปรแกรมเกม และการนำวัตถุที่ได้ไปใช้กับโปรแกรม และสุดท้ายคือการโปรแกรมเกมจากเอ็นจิ้นและโมเดลที่ได้สร้างขึ้นเอง เพื่อใช้สำหรับนักพัฒนาที่สนใจในการสร้างเกม3มิติต่อไป

3D Game Development

Chinnachote Pittayanont
Narong Wangchaisri
Dr. Worawat Limpoka Advisor

ABSTRACT

In this time, PC Game's business have a higher improve rating from the pass very much. The people, Adults or children can play it in the Game Shop or they can play it at home. However, In Thai Game's business they still have a little growing rate. although, they have Thai's language game but they just distributed from the right's distribution agent, not create from the Thai's people whole the software. Developmentation process in Thailand just in the beginning.

Then it's a chance to learn the process to make the game from The begin. It's means The Engine of the game's program, The Model, Input-Output of the 3D's game system that using Direct X and OpenGL to be interface with the hardware.

This thesis is concerned about how to create game's model, animation model to use for the our game engine by studying on the function and performance of 3D Studio Max. 3D Studio Max is a powerful and famous program to create the 3D object model

We study on the topic that how to make 3D object, the general technique to make easy 3D object for use in our game.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยความกรุณาของบุคคลหลายฝ่ายผู้ให้ความรู้ข้อมูลและช่วยเหลือทั้งจากเพื่อนๆพี่ๆและคณาจารย์ผู้เป็นที่เคารพรักในการสร้างผลงานขึ้นมาให้สำเร็จลงได้ และพ่อแม่พี่น้องที่เป็นกำลังใจให้ตลอดมา จึงขอขอบพระคุณไว้ในที่นี้ ด้วยรักและเคารพ

ชินโชติ พิทยานนท์

ณรงค์ วังชัยศรี

สารบัญ

หน้าที่

สารบัญ (ต่อ)

สารบัญภาพ

หน้าที่

สารบัญภาพ (ต่อ)

สารบัญภาพ (ต่อ)

สารบัญภาพ (ต่อ)

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ในปัจจุบันการพัฒนาทางด้านเทคโนโลยีคอมพิวเตอร์นั้นได้ก้าวหน้าไปอย่างรวดเร็ว โดยเฉพาะในด้านเกมคอมพิวเตอร์ ฮาร์ดแวร์ได้มีการพัฒนาระบบการแสดงผลให้สามารถประมวลผลภาพในลักษณะสามมิติ ที่มีทั้งความเร็วและความสวยงาม ไม่ว่าจะเป็นการ์ดแสดงผลสามมิติที่จัดการด้านการแสดงผลสามมิติโดยตรง หรือหน่วยประมวลผลกลางที่พัฒนาคำสั่งต่างๆ มารองรับการประมวลผลข้อมูลสามมิติ ทางด้านซอฟแวร์ก็มีโปรแกรมต่างๆ เข้ามาช่วยสร้างสรรค์ภาพสามมิติ ทำให้มีการสร้างเกมในลักษณะสามมิติออกมามากมาย ทั้งนี้ทั้งนั้น ส่วนที่ช่วยให้การทำเกมนั้นสำเร็จโดยง่าย หรือผู้อยู่เบื้องหลัง ส่วนหนึ่งก็คือ "เกมเอนจินต์", "โอเพนจีแอล" และ "ไดเร็กเอ็กซ์"

เกมเอนจินต์นั้น ช่วยอำนวยความสะดวกให้กับการสร้างเกมอย่างมาก มีฟังก์ชันเกี่ยวกับเกมต่างๆไว้ให้เรียกใช้งาน ช่วยในการสร้างภาพเคลื่อนไหวในเกม ส่วนไดโอเพนจีแอลและไดเร็กเอ็กซ์นั้น เป็นตัวกลางช่วยในการติดต่อกับฮาร์ดแวร์ เพื่อการจัดการกับภาพ เสียง การรับอินพุท ได้โดยตรงและรวดเร็ว

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

การทำงานวิจัยนี้ นอกจากได้ศึกษาการใช้งานโอเพนจีแอลและไดเร็กเอ็กซ์ในติดต่อกับฮาร์ดแวร์มัลติมีเดียและนำมาใช้ในการดึงความสามารถทางฮาร์ดแวร์ต่างๆได้อย่างเต็มประสิทธิภาพแล้ว 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 ด้านมัลติมิเดีย เช่น การ์ดจอ การ์ดเสียง อุปกรณ์อินพุท เป็นต้น เพื่อเป็นพื้นฐานในการพัฒนาเกมสามมิติ

1.2.3 นำโอเพนจีแอลและไดเร็กเอ็กซ์มาใช้ในการพัฒนาเกมสามมิติ

1.2.4 ออกแบบและพัฒนาเกมเอนจินต์ โดยใช้ โอเพนจีแอล, ไดเร็กเอ็กซ์ และ ซีพลัสพลัส

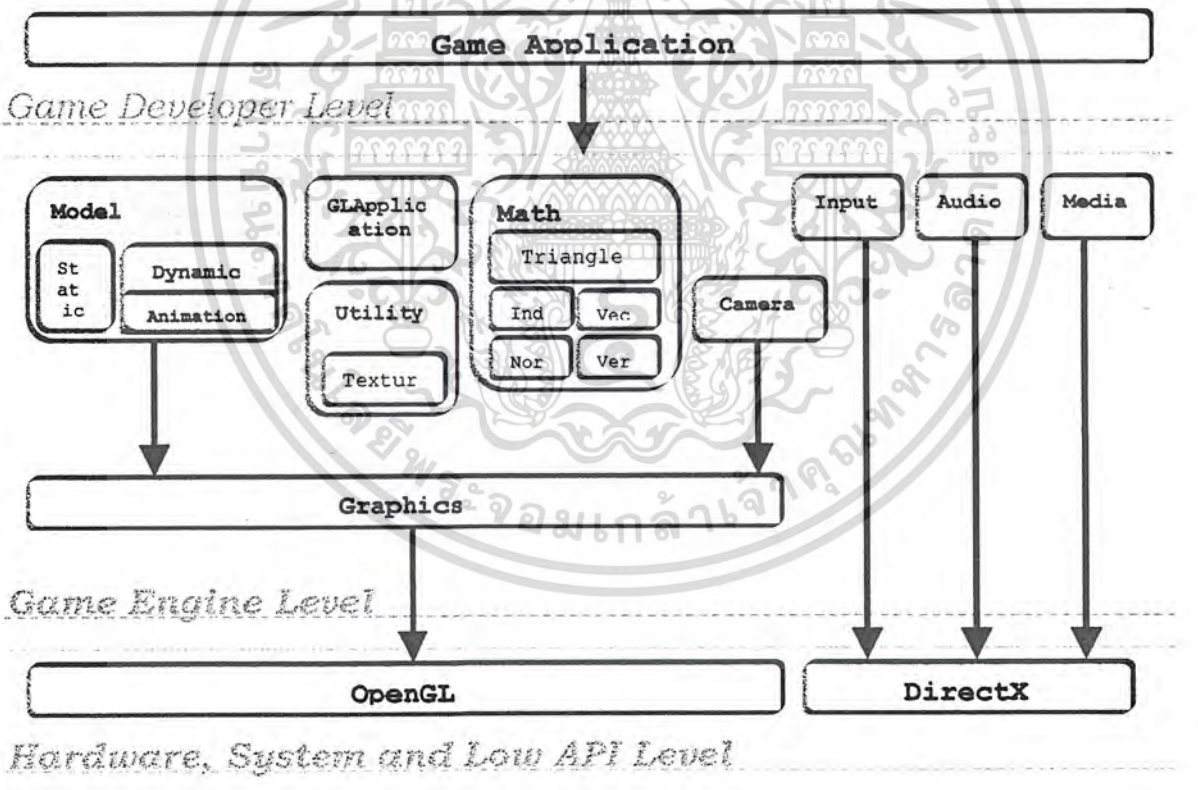
1.2.5 สร้างเกมสามมิติ จากเกมเอนจินต์ที่ได้ออกแบบเสร็จแล้ว

1.2.6 เพื่อพัฒนา Library tool รวมถึง Software tool ที่จะใช้ในการพัฒนาเกม

1.3 ขอบเขตของงานวิจัย

งานวิจัยนี้เน้นที่การศึกษาและสร้างเกมเอนจินต์สามมิติ โดยใช้เทคโนโลยี โอเพนจีแอล, ไดเร็กเอ็กซ์ และซีพลัสพลัส โปรแกรมมิ่ง เริ่มตั้งแต่การออกแบบเอนจินต์ การสร้างเกมเอนจินต์โดยหลักการโปรแกรมเชิงวัตถุ จนสามารถนำมาช่วยสร้างเกมสามมิติได้ จากนั้นนำเกมเอนจินต์ที่ได้สร้างเสร็จแล้ว มาสร้างเกมสามมิติขึ้นมา เพื่อทดสอบใช้งานเอนจินต์ และหาข้อบกพร่อง เพื่อนำไปสู่การพัฒนาให้สมบูรณ์ยิ่งขึ้นต่อไป

สำหรับในงานวิจัยนี้ได้เกิดจากการรวมตัวของกลุ่มนักศึกษาที่มีความสนใจในด้านเดียวกันหลายกลุ่ม จึงได้แบ่งกลุ่มร่วมกันทำงานเป็น 4 กลุ่มย่อย หลังจากได้มีการออกแบบร่วมกันแล้ว ได้มีการแบ่งงานกัน โดยนักศึกษาในกลุ่มนี้ได้รับผิดชอบในส่วน โมเดล และระบบวัตถุไม่เคลื่อนไหว ดังแสดงในรูปที่ 1-1



รูปที่ 1-1 แสดงโครงสร้างของเกมเอนจินต์ และส่วนที่กลุ่มรับผิดชอบ

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1.4.1 ทักษะในการพัฒนาเกม โดยนำเสนอในรูปแบบ 3 มิติ เช่น การสร้างตัวละคร การวางโครงเรื่องของเกม การสร้างภาพสามมิติ เป็นต้น

1.4.2 ความรู้ในด้านการประมวลผลภาพสามมิติ เช่น การหมุนภาพ การสร้างภาพเคลื่อนไหว การใช้คณิตศาสตร์มาช่วยจัดการภาพสามมิติ เป็นต้น

1.4.3 เทคนิคในการเขียนโปรแกรมประมวลผลภาพสามมิติ

1.4.4 การใช้โอเพนจีแอลและไดเร็กเอ็กซ์ติดต่อกับอุปกรณ์มัลติมิเดีย

1.4.5 การออกแบบโปรแกรมเชิงวัตถุ

1.4.6 การเขียนโปรแกรมซีพลัสพลัส เพื่อติดต่อกับโอเพนจีแอลและไดเร็กเอ็กซ์

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:

- การสร้างโมเดล 3 มิติ ด้วย 3D Studio MAX

กลุ่มที่ 4:

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

<u>โดยปริญญานิพนธ์ฉบับนี้จะได้นำเสนอเนื้อหาในกลุ่มที่ 3</u>

1.5.3 ออกแบบเกมเอนจินต์ จะอธิบายโครงสร้างของเกมเอนจินต์แต่ละส่วน และส่วนประกอบต่างๆของเกมเอนจินต์

1.5.4 พัฒนาส่วนประกอบต่างๆที่ได้ออกแบบไว้แล้ว

1.5.5 จากนั้นนำเอนจินต์ในแต่ละส่วนมาประกอบรวมกัน และกำหนดรูปแบบไฟล์ข้อมูลที่จะนำไปใช้กับเกมเอนจินต์ จัดทำแอพพลิเคชันเฟรมเวิร์ค สำหรับผู้นำเกมเอนจินต์ไปพัฒนาเกมในขั้นตอนต่อไป

1.5.6 เมื่อได้เกมเอนจินต์ที่ใช้งานได้แล้ว จึงนำเกมเอนจินต์ที่ได้ ไปสร้างเป็นเกมสามมิติเพื่อทดสอบการใช้งานเกมเอนจินต์

1.5.7 รวบรวมผลการทดสอบ วิเคราะห์ปัญหา และหาแนวทางการแก้ปัญหาต่างๆ เพื่อนำมาพัฒนาหรือเป็นข้อมูลในการพัฒนาขั้นต่อไป

บทที่ 2
ทฤษฎีและหลักการพัฒนาเกม 3 มิติ

2.1 โปรแกรม 3D Studio MAX

ขอบข่ายงาน

งานส่วนนี้เป็นส่วนหนึ่งของการพัฒนาเกม 3 มิติโดยใช้ ไดเร็กเอ็กซ์ เวอร์ชัน 8.0 โดยโครงงานนี้เป็นการจัดการเรื่อง โมเดล, การจำลองเกมและบรรยากาศในเกมทั้งหมดเพื่อให้ game engine สามานำไปใช้ได้ โดยสร้างโมเดลจากโปรแกรม 3D Studio Max แล้ว Export เป็น file ในรูปแบบ format ต่างๆที่เหมาะสมที่สุดที่จะนำไปใช้ใน game engine

3D Studio Max

โปรแกรม 3D Studio Max เป็นโปรแกรม ที่ใช้ในการสร้างงาน 3 มิติ แบบ Animation ซึ่งในเวอร์ชัน Relcase 3.1 นี้ สามารถรันได้บน Windows 95/98/ME และ Windows NT/2000 ซึ่งในเวอร์ชันก่อน Relcase 2.5 จะรันได้เฉพาะ Windows NT เท่านั้น ทำให้ 3D MAX เวอร์ชันนี้เริ่มใช้กันกว้างขวางมากขึ้นใน user ระดับ Home PC ทั่วไป

ด้วยประสิทธิภาพที่สูงอย่างมากในหลายๆ ด้านทั้งในการสร้าง Model(รูปทรง 3 มิติ)รูปแบบของพื้นผิววัตถุที่มากขึ้น การคำนวณสภาพการชนของวัตถุ(Simulation) และคำสั่งที่เพิ่มขึ้นมากเมื่อเทียบกับ 3DMAX ในรุ่นเก่าๆ ทำให้เราสามารถขึ้นรูปทรง 3 มิติด้วยการใช้ mouse click ในการสร้างได้ตามจินตนาการของเรา อีกทั้งสามารถกำหนดพื้นผิวให้กับวัตถุได้อย่างไร้ขอบเขตจำกัด

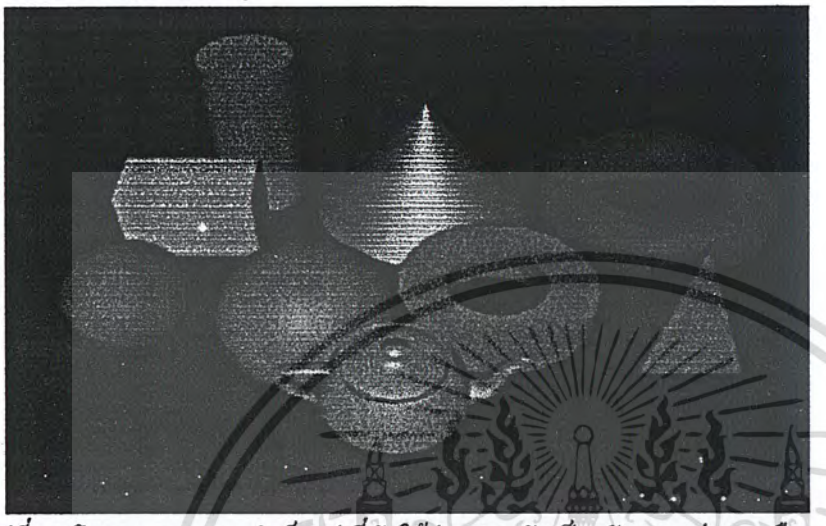
ข้อได้เปรียบของ 3D Studio Max

จากการศึกษาพบว่าโปรแกรมทางด้าน 3 มิติหลายโปรแกรม ไม่ว่าจะเป็น Light Wave , Soft Image, Maya และอื่นๆ นั้นมีความสามารถในการสร้างวัตถุ 3มิติคล้ายกันแต่โปรแกรมที่เห็นว่าเหมาะสมที่สุดคือ 3D Studio Max เนื่องจาก รูปแบบคำสั่ง ง่าย มี การติดต่อที่หน้าจอ ง่ายกว่าเมื่อเทียบกับ Soft Image, Maya สามารถใช้ PC ที่อยู่ตามบ้านทั่วไป ซึ่งไม่จำเป็นต้องมี คุณสมบัติของเครื่องที่สูงเหมือน ใน โปรแกรมอื่นๆเช่น Maya ที่ต้องมี Ram อย่างน้อย 128 และต้องใช้ mouse แบบ 3 ปุ่มเท่านั้น และไม่สามารถ สร้างไฟล์ที่ Export ด้วยตนเองได้อย่างสมบูรณ์(Maya Script) ขณะที่ 3D Studio MAX ทำได้ด้วย MAX Script และ คุณสมบัติเครื่อง ใช้เพียง Ram 64 ก็ใช้งานได้แล้ว สำหรับ โปรแกรม 3D Studio MAX จะนิยมใช้มากในแถบ ยุโรป อเมริกา และไทยก็มีใช้กันมากด้วย

ความสามารถของ 3D Studio Max และตัวอย่าง Model

สำหรับความสามารถของ 3D Studio Max มีมากมาย ในที่นี้ได้ยกตัวอย่างมาพอสังเขป

- โมเดล มาตรฐานสำเร็จรูป



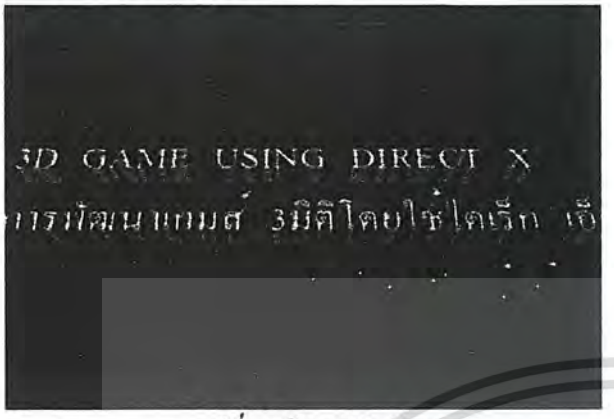
รูปที่2-1 โมเดลมาตรฐาน สำเร็จรูป ที่มักใช้ประกอบกันเป็น ตัวละครต่างๆ หรือ ฉากต่างๆ ในระดับ ธรรมดา

- โมเดล ขั้นสูงสำเร็จรูป

รูปที่2-2 ตัวอย่างโมเดลขั้นสูง

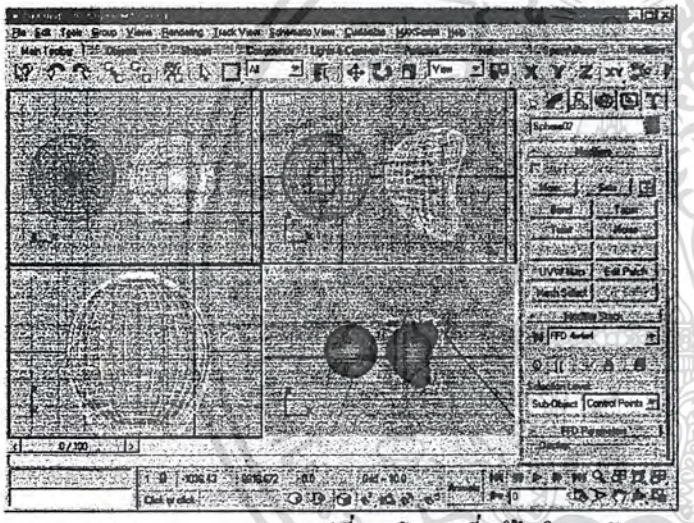
เป็น โมเดลมาตรฐาน สำเร็จรูป ที่มักใช้ประกอบกันเป็น ตัวละครต่างๆ หรือ ฉากต่างๆ ในระดับสูง ส่วนใหญ่มักร่วมกันทั้ง 2 แบบ

- โมเดล ตัวอักษรทุกชนิด



รูปที่2-3 โมเดลตัวอักษร

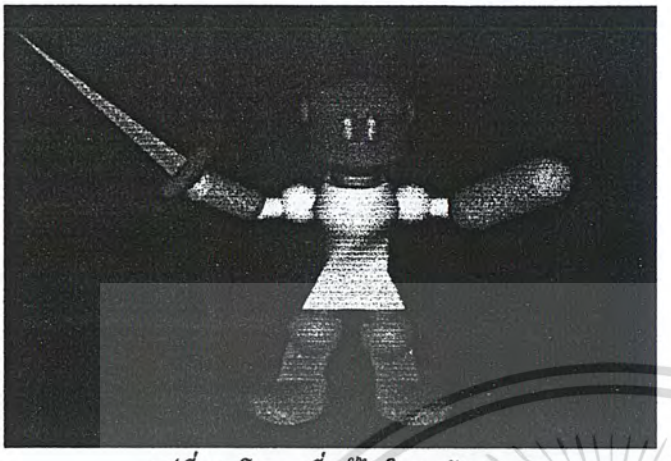
- ความสามารถในการเปลี่ยนแปลงโมเดลในระดับ Pixel



รูปที่2-4 โมเดลที่แก้ไขในระดับ Pixel

มีการแก้ไขในระดับจุด ซึ่งทำให้โมเดลมีความคล่องตัวมาก

- โมเดล ตัวละครหลังจากมีการ แก้ไขในระดับ Pixel



รูปที่2-5 โมเดลที่แก้ไขในระดับ Pixel

โมเดลนี้ มีการแก้ไขในระดับ Pixel คือ ท้อง และ เท้า

รูปที่2-6 โมเดลที่มีการปะ Texture

สัตว์ประหลาด จะมีการปะพื้นผิว ชนิดบิตแมปที่สร้างมาจาก โฟโตชอป (Adobe Photoshop)

● ความสามารถในการเล่นแสงเงา และ การปะพื้นผิว ได้อย่างไม่จำกัด

รูปที่2-7 รูปการปะพื้นผิววัตถุ

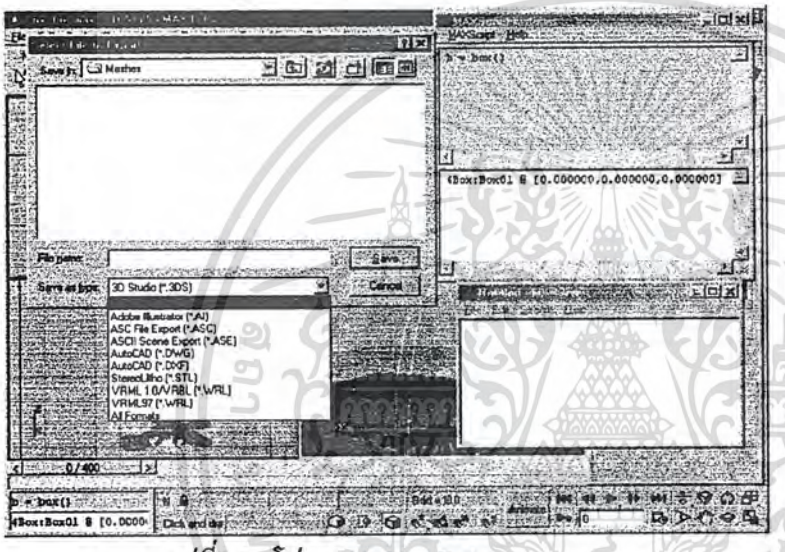
รูปที่2-8 ตัวอย่าง โมเดล

รูปที่2-9 ตัวอย่าง โมเดล

นอกจากนี้ยังมีความสามารถในการ ทำ Animation , File Video, การใช้ Bone และการตั้งกล้อง ซึ่งดูได้จาก Demo

การนำไปใช้และ FILE EXPORT

สำหรับลักษณะการนำไปใช้ จะเป็นลักษณะการ export file เป็น format file ที่เราเข้าใจและนำไปใช้ได้ง่ายที่สุด แบ่งเป็น 2 ประเภท คือ format file ที่มีมาให้อยู่แล้วได้แก่ *.ASE ,*.3DS,*.DXF เป็นต้น และอีกแบบคือ เขียน format เองตามที่เราต้องการ นั่นคือ การใช้ Function MAX Script ในการเก็บข้อมูลแบบของเรา นั่นเอง



รูปที่2-10 โปรแกรม Max Script

เนื่องจาก format file สำเร็จรูป จะถูกจัดวางข้อมูล เป็นลักษณะ โครงสร้างซึ่งเราจะต้องเขียนโปรแกรมตัดสตริงเอง และมีข้อมูล การทำ Animation ที่ซ้ำซ้อน ดังนั้น การเขียน format file เองจึงเป็นวิธีที่น่าสนใจวิธีหนึ่งเลย นอกจากนี้ถ้า engine game ที่พัฒนามาจาก Open GL จะไม่สามารถใช้ format file *.X ได้ ขณะที่ Direct X เอาไปใช้ได้ ดังนั้น Max Script จึงเป็นอีกทางเลือกที่ต้องการ

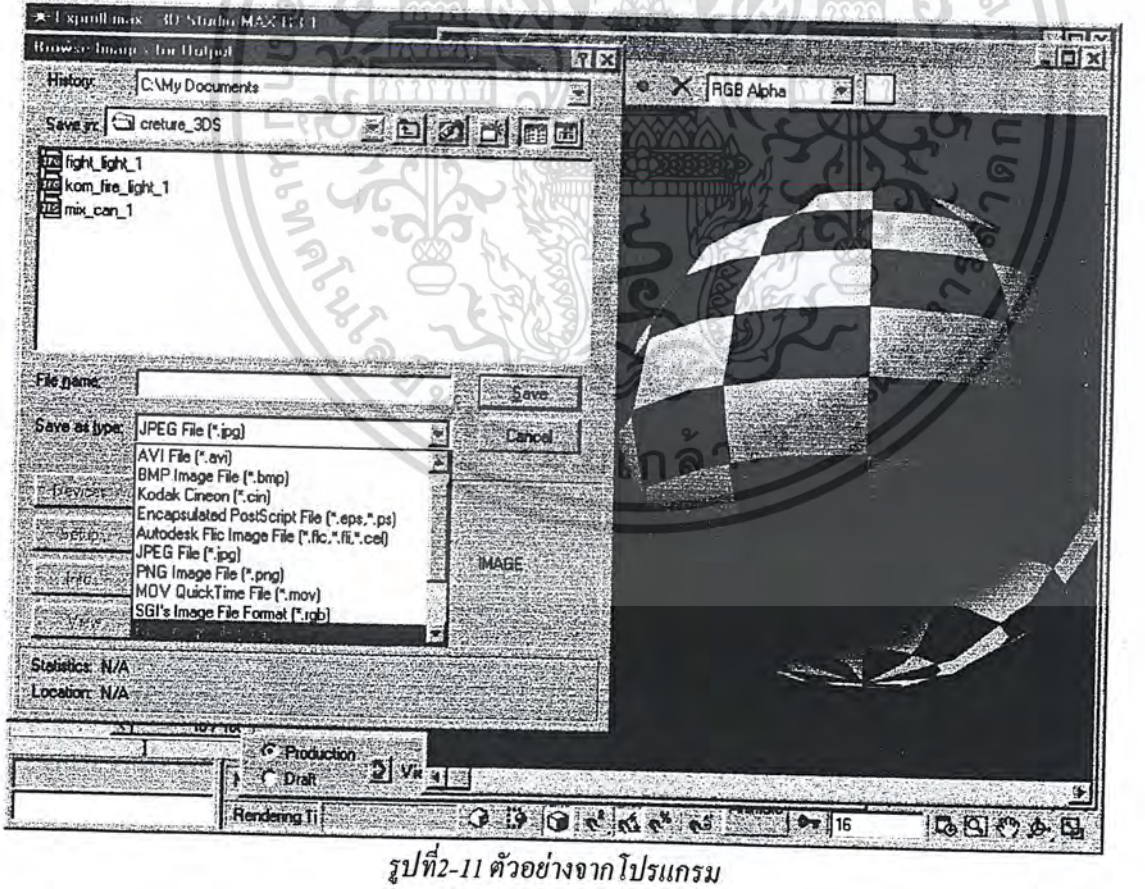
2.2 การสร้างโมเดลด้วย 3D Studio MAX

- การใช้งาน 3D Studio MAX

ใน 3D Studio MAX เราจะสร้าง วัตถุ(object) ใน viewports โดยใช้ เครื่องมือ(tools) และ Import ออกมาได้ด้วย สามารถคอนโทรลวัตถุด้วยค่าต่างๆใน command-panel rollouts ซึ่งเกิดจาก เส้นตรง, เส้นโค้ง, รูปทรง 2 มิติ หรือรูปทรง 3มิติ

วัตถุแสง(light objects) คือการเพิ่มเงาและแสงใน วัตถุ, คาเมร่า(cameras) จะถูกสร้างสำหรับการทำหนัง(movies), ผิวของวัตถุจะถูกตกแต่งด้วย วัสดุ(material) ที่สร้างขึ้น และ ถูกแปลงในฉาก, Effect ต่างๆ เช่น บรรยากาศต่างๆ หิมะ การระเบิด ก็จะสามารถนำมาใช้เพิ่มเข้าไปได้เช่นกัน

นอกจากนี้ การแสดงการเคลื่อนไหวจะถูกสร้างด้วย keyframe animation, มีการจำลอง แรงดึงดูด การชนกันแบบ dynamic เกิดขึ้น Motion controllers จะเพิ่ม animation tracks เพื่อทำ animation ได้ดียิ่งขึ้น เช่นการเคลื่อนที่ไปตามเส้นทางที่กำหนด camera ก็จะทำ animation สำหรับทำ วีดีโอ และที่สุดของการ Render ภาพ Animation คือ การนำ output ของการ render ไปใช้ ได้แก่ ไฟล์วีดีโอ และรูปภาพ เป็นไฟล์ฟอแมตต่างๆ มักจะเอาไปใช้ใน วีดีโอเกม ดังรูป



รูปที่2-11 ตัวอย่างจากโปรแกรม

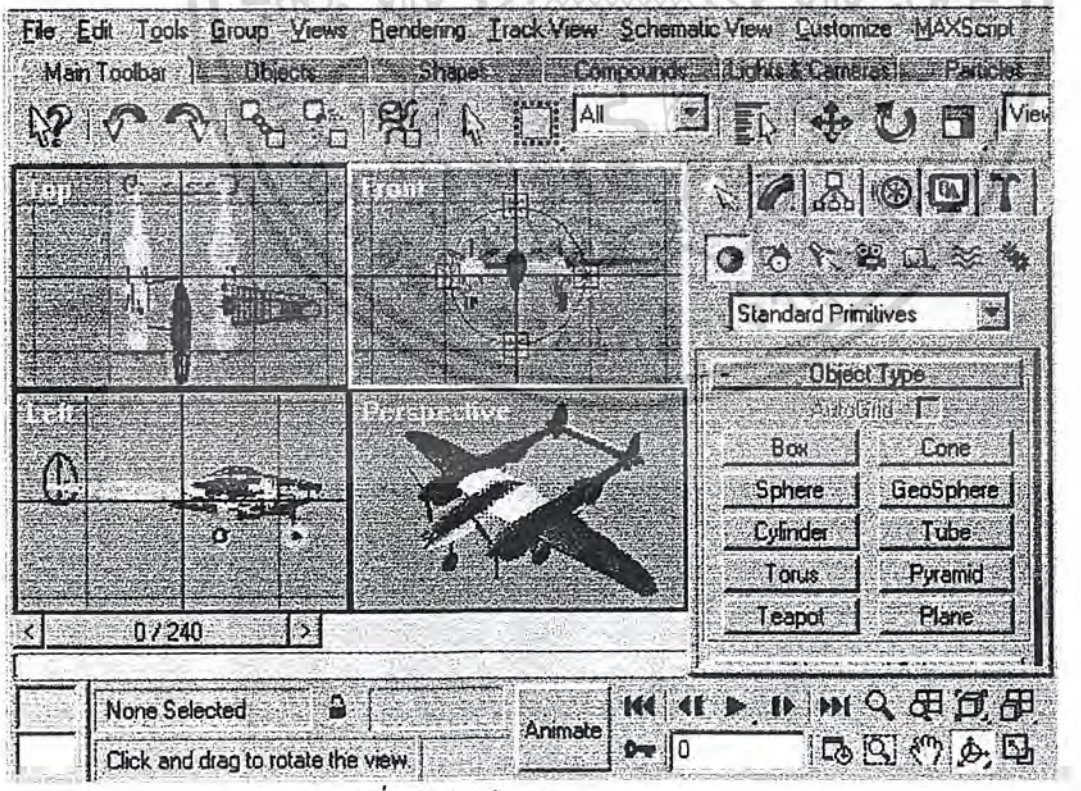
จากรูปข้างบนพบว่า output หลังจากทำการ render สามารถนำไปทำเป็น ไฟล์ วีดีโอ หรือรูปภาพ ต่างๆ ได้ เช่น *.AVI, *.BMP, *.JPG เป็นต้น

ส่วนของงาน 3D Studio MAX ดังที่กล่าวมาจะถูกนำไปใช้ในโครงงานนี้เป็นดังรูปต่อไปนี้

รูปที่2-12 แผนภูมิการนำโมเดลไปใช้

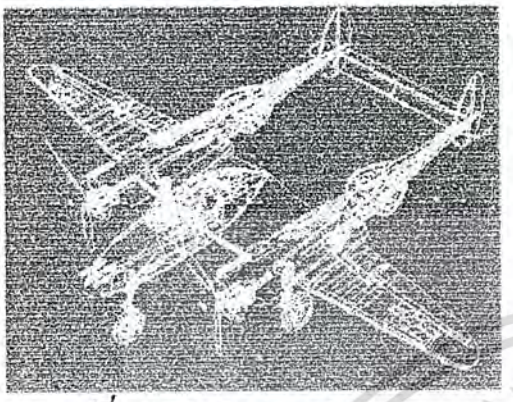
- The Viewports

ใน 3DS MAX โมเดลวัตถุจะถูก แสดงในลักษณะ default user interface ประกอบไปด้วย 4 viewports คือ มุมมอง TOP, LEFT, FRONT และ Perspective ซึ่งสามารถเซตเลือก viewports ตามต้องการได้

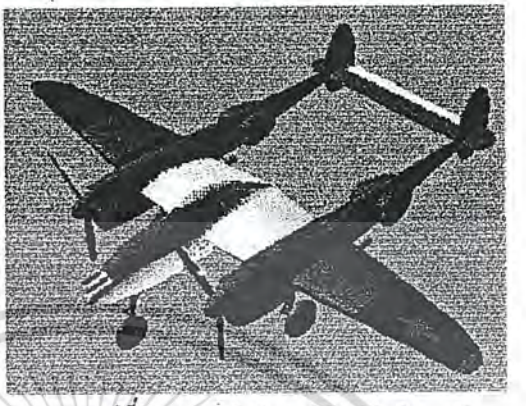


รูปที่2-13 ตัวอย่าง Viewport

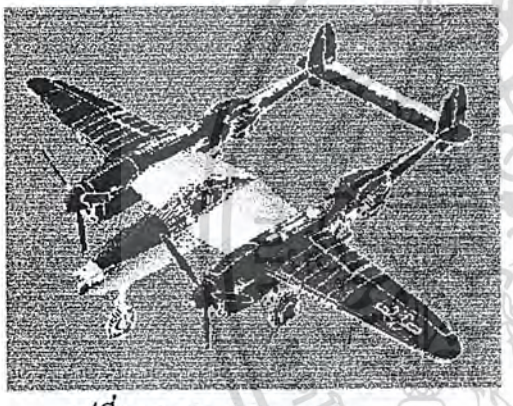
และสามารถเลือกได้ว่าจะ display geometry ใน โหมด wireframe, shaded, หรือ โหมด Edged-face หรือโหมดอื่นๆได้ตามต้องการ



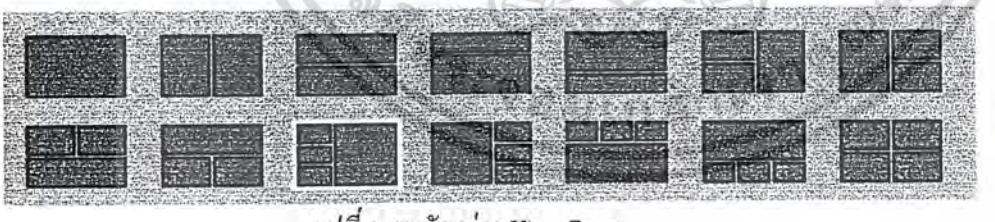
รูปที่2-14 Wrieframe mode



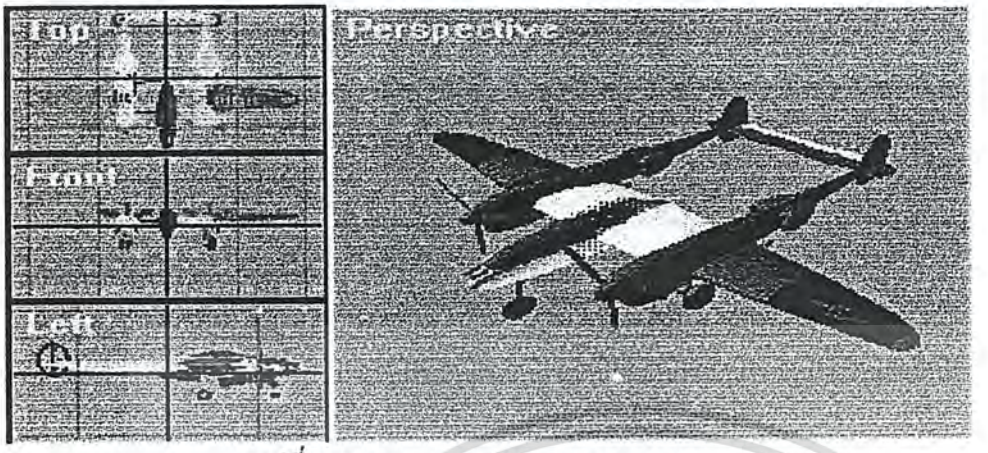
รูปที่2-15 Smooth and highlight mode



รูปที่2-16 Edged-face mode



รูปที่ 2-17 ตัวอย่าง View Port

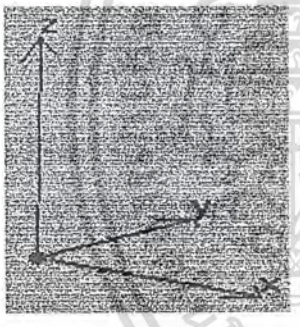


รูปที่2-18 An alternative viewport layout

- **The Vocabulary of 3D**

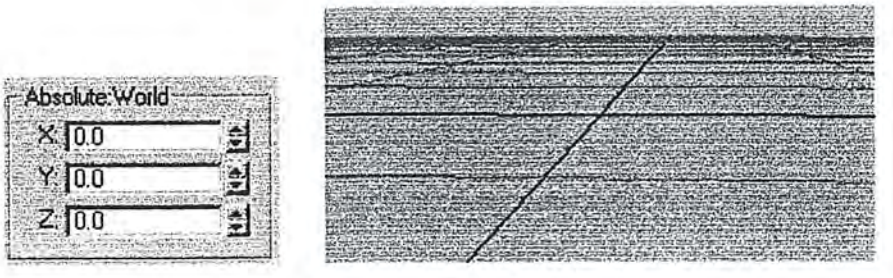
The vocabulary of 3D คือศัพท์ ทางคณิตศาสตร์ที่อธิบาย ส่วนต่างๆใน 3D geometry , ส่วนที่เล็กที่สุดในโลก 3มิติ คือ vertex เป็นจุดใน space.

Vertex



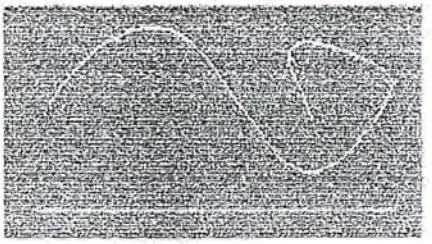
รูปที่2-19 vecter

The Vertex เป็นจุดที่ไม่มีความกว้างและความยาวได้แก่ 3D vertices จะใช้สร้าง 3D geometry, 2D vertices ใช้สร้างเส้น และหน้า(shapes) ตำแหน่งใน space สำหรับ 3D vertex มี 3ค่าคือ X, Y และ Z โดยค่าดังกล่าวบอกถึงระยะทางจากจุด world space origin(0,0,0) ซึ่งเป็นจุดศูนย์กลางของ universe



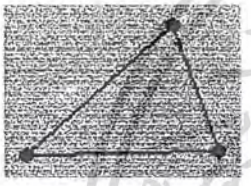
รูปที่2-20 The world space origin point

Lines

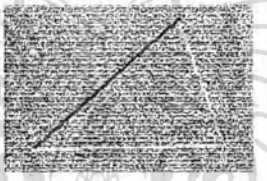


รูปที่2-21 2D lines

Gaining Face, Edge and Polygon

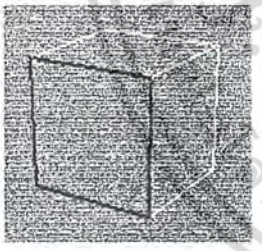


รูปที่2-22 A face



รูปที่2-23 An edge

A polygon เกิดจาก 2 faces หรือมากกว่ามาต่อกัน



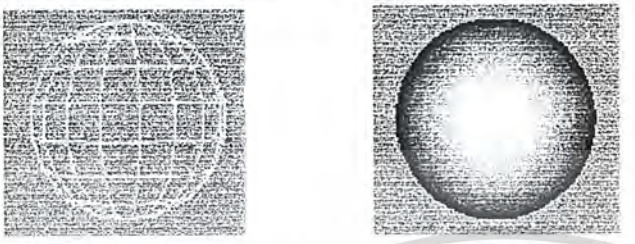
รูปที่2-24 A polygon

Polygon มีการเชื่อมต่อกันเป็น A 3D element

รูปที่2-25 A 3D element

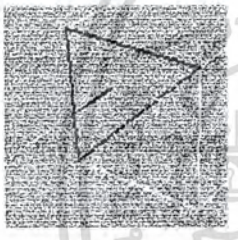
Surface and Object Properties

Faces มีเพิ่ม information ในการกำหนด smoothing ระหว่าง edges, Smoothing groups จะสร้าง smoothing ทำให้ทรงกลมรูปนี้เหมือนทรงกลม มากกว่าการรวมกันของสามเหลี่ยม



รูปที่2-26 Smoothing creates the illusion of roundness

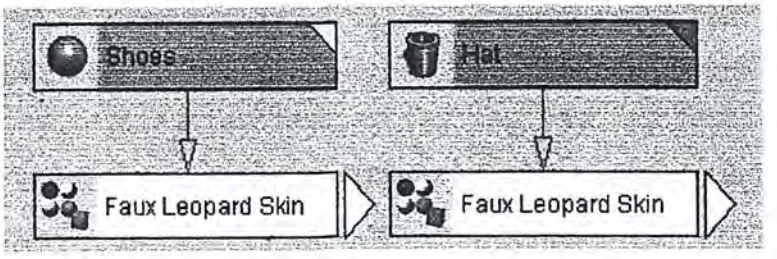
ทุกๆ สามเหลี่ยมจะมี 2ด้านคือด้านในและ ด้านนอก ข้อมูลที่บอกถึงทิศทางพุ่งไปข้างนอกที่อยู่ในรูปเวกเตอร์ในสามเหลี่ยมใดๆ ถูกเรียกว่า face normal



รูปที่2-27 Face normals point which way is out

Modifiers and the Modifier Stack

เมื่อใดที่มีการปรับปรุง หรือแก้ไขวัตถุที่เราสร้าง 3DMAX จะเก็บบันทึกการเปลี่ยนแปลงแก้ไขทั้งหมดที่เกิดขึ้นกับวัตถุนั้นไว้ใน Modifier Stack ที่ทำหน้าที่คล้ายๆกันกับ สมุดเก็บประวัติ การเปลี่ยนแปลงของวัตถุชิ้นนั้นๆ ทุกขั้นตอน โดยเราสามารถเข้าไปแก้ไขเปลี่ยนแปลงลำดับคำสั่งที่ใช้ในการแก้ไขวัตถุ หรือยกเลิกบางคำสั่งใน Modifier Stack ได้ Schematic View โชว์ modifiers shared โดย multiple objects ซึ่งมีการทำงานในลักษณะแบบขนานกันไป Modifier จะทำการ copy ค่าmodifiers จาก object ไปอีก object หนึ่ง ตามลำลับของ modifier ที่เราแก้ไขใน stack



รูปที่2-28 Schematic View shows shared materials

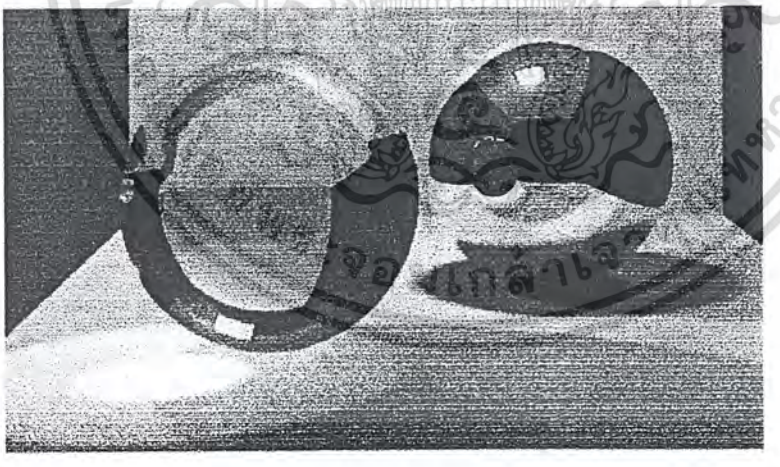
Non-Geometric Objects

Lights

รูปที่2-29 ไอคอนในโปรแกรม

แหล่งกำเหนิดแสงของ 3DMAX มีด้วยกันหลายชนิดเพื่อทำให้เกิดลักษณะ และรูปแบบของแสงต่างๆ หลายรูปแบบดังนี้

- Ambient แสงในบรรยากาศ
- Omni แสงแบบจุดกระจายรอบทิศ
- Directional แสงขนานส่องเป็นลำแบบกลมและสี่เหลี่ยมเป็นแนวตรง
- Target Spot แสงจากจุดกำเหนิดส่องเป็นรูปทรงกรวย
- Free Spot แสงในลักษณะกรวย



Cameras

ชนิดของกล้องจะมีอยู่ 2 ลักษณะคือแบบ Target และ แบบ Free

กล้องแบบ Target

กล้องแบบ Target นี้เราสามารถควบคุมตำแหน่งของตัวกล้อง(camera) และกรอบเป้าหมาย(Target) ได้อย่างอิสระ เมื่อเราย้ายตำแหน่งของกรอบเป้าหมาย ตัวกล้องจะหมุนตัวเองตามกล้องเป้าหมายไปด้วย มี

ประโยชน์ในการส่องภาพเคลื่อนไหว โดยการย้ายกรอบของเป้าไปตามตำแหน่งของวัตถุที่เคลื่อนที่ ในขณะที่ตัวกล้อง ยืนอยู่ที่ตำแหน่งเดิม

กล้องแบบ Free

กล้องแบบ Free นี้จะมีคุณสมบัติทุกอย่างเหมือนกับกล้องแบบ Target แต่จะไม่มีกรอบ เป้าหมายควบคุมทิศทางกล้อง ในการเปลี่ยนตำแหน่งการถ่ายภาพนั้น เราต้องควบคุมโดยการหมุนตัวกล้องด้วยคำสั่งหมุน (Rotate) เพื่อควบคุมทิศทางการถ่ายภาพ

เราใช้กล้องแบบ Free ในการมองภาพในลักษณะที่กล้องเคลื่อนที่ เช่นเราถ่ายภาพด้วยกล้อง วีดีโอเมื่อเรานั่งอยู่ในรถ เมื่อรถเคลื่อนที่ภาพที่เราเห็นก็จะเคลื่อนตาม

องค์ประกอบของวัตถุ 3 มิติใน 3D Studio MAX

ใน 3D studio MAX 3 นี้ วัตถุที่ถูกสร้างขึ้น จะประกอบด้วยองค์ประกอบย่อยๆอีกหลายองค์ประกอบ ดังนี้

- โครงสร้างของวัตถุที่ประกอบขึ้นจาก Vertex, Edge, Faces
- จุดอ้างอิงวัตถุ (Pivot Point)
- ชื่อของวัตถุ
- สี
- คุณสมบัติพื้นผิวของวัตถุ (Material)

องค์ประกอบของพื้นผิวประกอบวัตถุ

ตามที่เราได้ทราบกันแล้วว่าพื้นผิวของวัตถุ 3 มิติใน 3DMAX โดยพื้นฐานแล้วประกอบขึ้นด้วยพื้นผิวประกอบวัตถุเป็นรูปสามเหลี่ยม หรือสี่เหลี่ยมเล็กๆเรียงตัวเป็นรูปทรงต่างๆซึ่งรูปทรงของแผ่นสี่เหลี่ยมเล็กๆ นี้เราเรียกว่า พื้นผิวประกอบวัตถุ (Face) ประกอบด้วยจุด Vertex และเส้นเชื่อมระหว่าง Vertex เรียกว่า Edge ซึ่งถ้าเรามองลึกลงไปในส่วนย่อยแล้ว แผ่น Face นี้เกิดจากแผ่นสามเหลี่ยม 2 แผ่นประกบกัน แต่โดยปกติแล้ว 3DMAX จะไม่แสดงเส้น Edge ที่ว่านี้ จึงเรียกเส้นนี้ว่า Invisible Edge (หรือเส้น Edge ที่มองไม่เห็น

Face มี 2 ด้าน

โดยปกติแล้วเราเรียกพื้นผิวแบบปิดของวัตถุว่า Polygon ขอให้เข้าใจว่าการสร้างเส้นแบบปิด จะยังไม่มีพื้นผิวเชื่อมต่อระหว่าง Edge จนกว่าเราจะสร้างพื้นผิวเชื่อมระหว่างเส้น Spline ปิด ถึงจะเรียกว่า Polygon และมีพื้นผิวของวัตถุ

Polygon Face ใน 3DMAX เราจะสามารถมองเห็นได้เพียงด้านเดียวเรียกว่า Single-Side Polygon นั่นคือ Face จะมีด้านหน้าและด้านหลัง เมื่อ Face ด้านหน้าหันออกทางด้านหน้าจอ เราจะเห็นเป็นแผ่นตัน แต่ละ Face

ถูกหันหลังให้หน้าจอ เราจะมองไม่เห็น Face นั้นเลย ในกรณีที่เกิดเราสร้างวัตถุ 3 มิติโดยมีบางส่วนของพื้นผิววัตถุที่ถูกหันด้านหลังออกมา ภาพที่ปรากฏบนหน้าจอ และคำนวณในการเรนเดอร์ภาพจะไม่ปรากฏขึ้นมา

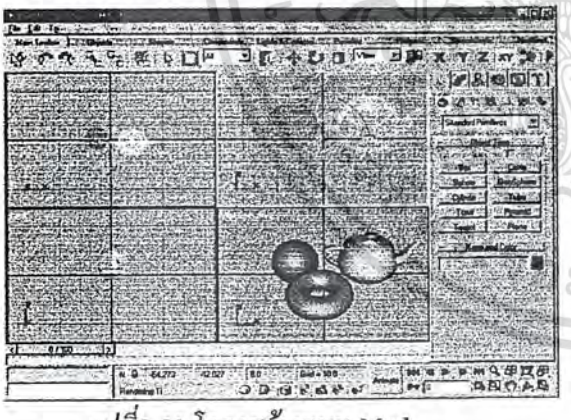
เราสามารถกำหนดให้พื้นผิวของ Face มองเห็นได้ทั้งด้านหน้า และด้านหลังได้ โดยแยกออกเป็น 2 ส่วน คือ ส่วนแสดงในจอภาพย่อยบนหน้าจอที่ตัวเลือก Force-2 Side ซึ่งจะส่งผลให้เราสามารถเห็น Face ได้ทั้ง 2 ด้านบนจอภาพย่อย และภาพที่ได้จากการ Render ถึงแม้จะกำหนดการ Force-2 Side ในจอภาพย่อยแล้วก็ตาม เราต้องกำหนดให้สามารถมอง Face ได้ทั้ง 2 ด้าน ในหน้าต่างควบคุมการ Render เมื่อมีการ Render ภาพด้วย

<u>รูปแบบของโครงสร้าง 3 มิติใน 3DMAX</u>

ใน 3DMAX จะแบ่งรูปแบบของโครงสร้างวัตถุทาง 3 มิติออกเป็นหลายชนิด ซึ่งในแต่ละรูปแบบจะมีลักษณะในการคำนวณพื้นผิวของวัตถุแตกต่างกัน ซึ่งวัตถุที่มีรูปร่างเดียวกัน อาจสามารถเปลี่ยนโครงสร้างไปมาระหว่างกันได้ โดยจะขอแบ่งออกเป็น 4 กลุ่มหลักใหญ่ๆ

<u>1. โครงสร้างวัตถุแบบ Mesh</u>

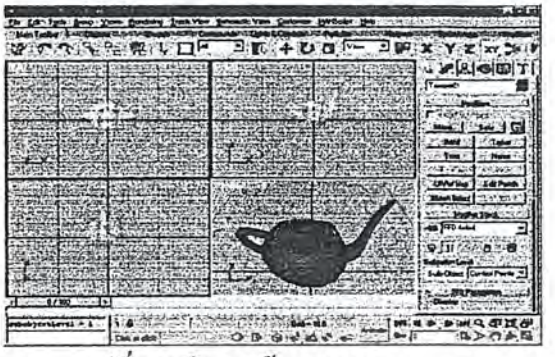
เป็นโครงสร้างของพื้นผิววัตถุแบบขั้นพื้นฐานที่สุด คือ เราเห็นวัตถุประกอบด้วยพื้นผิว Faceหลายพื้นผิวประกอบขึ้นเป็นวัตถุ 3 มิติ ซึ่งการแก้ไขวัตถุแบบ Mesh นี้ เราจะเข้าไปแก้ไขได้ที่พื้นผิวของวัตถุโดยตรง



รูปที่2-31 โครงสร้างแบบ Mesh

<u>2. โครงสร้างวัตถุแบบ Patch</u>

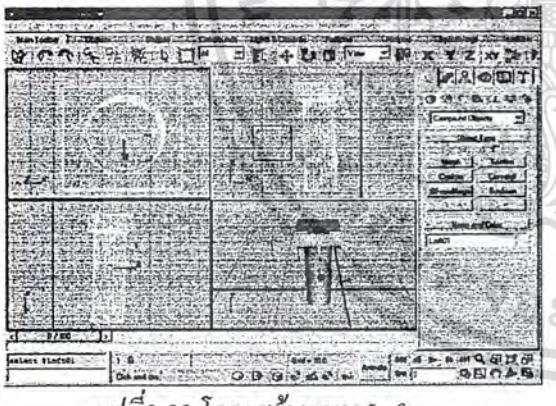
เป็นโครงสร้างวัตถุที่ประกอบขึ้นจากการกำหนดโครงสร้างครอบวัตถุ ประกอบด้วยจุด Vertex และ Lattice Handle ซึ่งเป็นเวคเตอร์ในการควบคุม และคำนวณพื้นผิวของโครงสร้าง 3 มิติที่สร้างขึ้น



รูปที่2-31 โครงสร้างแบบ Patch

3.โครงสร้างวัตถุแบบ Loft

เป็นโครงสร้างวัตถุที่เกิดจากภาคตัดขวางของวัตถุที่ถูกดึงตามเส้นทาง ซึ่งเราสามารถสร้างขึ้นด้วยเส้น 2 มิติ โดยเราจะเรียกภาคตัดขวางของวัตถุ Loft นี้ เราจะแก้ไขที่รูป Shape ของภาคตัดขวางกับเส้น Path ใน 3DMAX จะมีกลุ่มคำสั่งในการปรับปรุงแก้ไขรูปทรงของวัตถุ Loft ด้วยกลุ่มคำสั่ง Deformation ซึ่งจะเป็นกลุ่มคำสั่งเฉพาะของวัตถุ Loft โดยมีหลักการแก้ไข Shape ตามระยะ Path

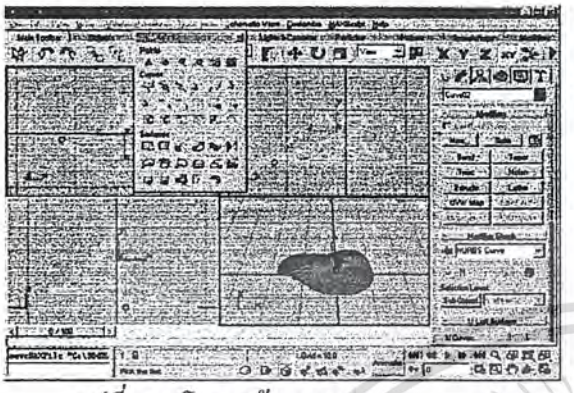


รูปที่2-32 โครงสร้างแบบ Loft

4.โครงสร้างวัตถุแบบ NURBS

เป็นโครงสร้างวัตถุแบบใหม่ที่มีขึ้นเต็มรูปแบบใน 3DMAX โดยมีหลักการเชื่อมเส้นพื้นผิวของวัตถุด้วยเส้นรอบรูป ซึ่งทำให้เราสามารถสร้างพื้นผิวของวัตถุใดๆ ก็ได้ โดยการสร้างเส้นรอบรูปของวัตถุ แล้วค่อยเชื่อมพื้นผิวของวัตถุด้วยคำสั่งในการสร้างพื้นผิว โดยมีข้อแม้ว่าเส้นที่เราต้องการสร้างวัตถุ NURBS จะต้องเป็น

เส้น NURBS เท่านั้น แต่ใน 3DMAX นี้จะมีคำสั่งในการเปลี่ยนเส้น Spline หรือเส้น 2 มิติแบบธรรมดาเป็นแบบ NURBS ได้



รูปที่2-33 โครงสร้างแบบ Nurbs

ในโครงสร้างวัตถุทั้ง 4 แบบนี้ เราสามารถที่จะปะภาพ และ Render วัตถุได้เหมือนกันทุกอย่าง แต่การที่ 3DMAX มีโครงสร้างของวัตถุในหลายๆรูปแบบ เพื่อให้เราสามารถสร้างและแก้ไขรูปร่างของวัตถุที่ต้องการสร้างได้อย่างเหมาะสม ซึ่งการสร้างรูปทรงของวัตถุขั้นหนึ่งอาจง่ายเมื่อสร้างด้วยโครงสร้างแบบ NURBS แต่อาจยากต่อการขึ้นรูปทรงวัตถุในรูปแบบอื่นๆ ที่จุดนี้ก็ให้พิจารณาเอาว่าควรจะขึ้นรูปทรงวัตถุที่ต้องการด้วยโครงสร้างแบบใด

ในการขึ้นรูปทรง 3 มิติในคอมพิวเตอร์ ถือว่าเป็นศาสตร์อีกแขนงหนึ่ง ซึ่งไม่ใช่ว่าเขียนรูปเก่งจะสามารถสร้างงาน 3 มิติได้ดีกว่าผู้ที่เขียนรูปได้เก่งๆ ตรงนี้เราจะต้องมาเริ่มต้นที่จุดศูนย์ แล้วค่อยๆเริ่มฝึกฝนไปจนชำนาญ ซึ่งจะพบว่าผู้ที่สร้างงาน 3 มิติในคอมพิวเตอร์หลายๆ คนยังคงเขียนรูปบนกระดาษไม่เก่ง หรืออาจลงสีไม่เป็นเลย แต่จะมีพื้นฐานทางด้านคอมพิวเตอร์เพิ่มขึ้น

<u>แนวการศึกษาการสร้างงานทางกราฟฟิก 3 มิติ</u>

แนวการศึกษาการสร้างงานทางกราฟฟิก 3 มิติ บนคอมพิวเตอร์ โดยหลักการทั้งหมดแล้ว จะประกอบด้วย 4 ขั้นตอนตามลำดับ ดังนี้

1. สร้างวัตถุ 3 มิติขึ้นตามที่เราต้องการ ซึ่งรวมไปถึงการวางตำแหน่ง การจัดองค์ประกอบ การจัดการปรับปรุงรูปร่างของวัตถุ เพื่อให้ได้วัตถุที่เราต้องการ

2. กำหนดพื้นผิวให้วัตถุแต่ละชิ้นที่เราสร้างขึ้นมา ซึ่งอาจประกอบด้วยลักษณะพื้นฐานทางกายภาพของวัตถุ เช่น การสะท้อนแสง ความมันวาวของพื้นผิว ความโปร่งใสของวัตถุรวมทั้งยังสามารถปะภาพในรูปแบบต่างๆ

3. กำหนดมุมมองของภาพโดยมองผ่านเลนส์กล้อง ซึ่งเราสามารถกำหนดโดยเลือกขนาดของเลนส์กล้องได้หลายขนาด นอกจากนี้เรายังสามารถมองภาพแบบทัศนียภาพ (Perspective) จากเลนส์กล้อง และยัง

สามารถมองผ่านแหล่งกำเนิดแสงได้ด้วย โดยกำหนดมุมมองให้กว้างหรือแคบตามความกว้างของลำแสง ซึ่งเราสามารถกำหนดได้อย่างอิสระ

4. กำหนดสภาพแวดล้อม องค์ประกอบของภาพ เช่น ภาพฉากหลัง (Background) หมอกในบรรยากาศ สภาพแสงในบรรยากาศ (Ambient) เป็นต้น

ขั้นตอนข้างต้นนี้เป็นแนวทางในการสร้างวัตถุ 3 มิติขั้นพื้นฐานเท่านั้น ซึ่งอาจสลับขั้นตอนหรือมีแนวทางในการสร้างงาน 3 มิติในแบบที่แตกต่างกันออกไป

2.3 ฟังก์ชัน MAX Script ใน 3D Studio MAX

การศึกษา Format file 3DS

เมื่อได้สร้างโมเดลขึ้นมาจากโปรแกรม 3d studio แล้วก็จะต้องมีการจัดเก็บเป็นไฟล์ไว้เพื่อเรียกใช้ต่อไป ในการจัดเก็บไฟล์นั้นเราจะต้องคำนึงถึงการเรียกใช้ไฟล์ ให้สามารถมีการจัดการได้ง่าย เรียกใช้ข้อมูลได้อย่างรวดเร็วและต้องสามารถเก็บข้อมลที่ต้องการได้อย่างครบถ้วน เราจะมาศึกษาไฟล์ ฟอร์แมตต่างๆ ของโมเดล 3มิติกัน

3DS file format

(รายละเอียดบางส่วนของ file ยังไม่สามารถทำความเข้าใจได้ และ Autodesk 3ds ไม่ได้ เปิดเผย source และข้อมูลของ file format ของตนแต่อย่างใด)

3DS เป็น format file ที่ใช้ใน auto-desk 3d studio จะเก็บข้อมูลทั้งหมดที่โปรแกรมได้สร้างขึ้นมีรายละเอียดดังนี้ 3DS file format ประกอบไปด้วยกลุ่มก้อนใหญ่ๆ เรียกว่า chunk มีหน้าที่อธิบายข้อมูลว่าเป็นข้อมูลของอะไร จะเก็บ ID และชี้ว่าส่วนไหนคือ ส่วนต่อไป ข้อมูล binary ใน 3DS file ถูกเขียนขึ้นในลักษณะพิเศษ byte ที่มีความสำคัญน้อยกว่าจะอยู่ข้างหน้าและ ตัวอย่าง 4A 5C (2byte ในระบบ hex) จะเป็น 5C byteสูง และ 4A byte ต่ำ และอีก ตัวอย่างหนึ่ง 4A 5C 3B 8F โดย 5C 4Aเป็น word ต่ำกว่าและ 8F 3B เป็น word สูงกว่า ต่อไปเราจะมาดูที่ chunk ซึ่ง chunk จะถูกระบุไว้ดังนี้

start end size name

0 1 2 Chunk ID

2 5 4 pointer ที่ชี้ไปที่ chunk ต่อไป โดยชี้ไปที่ chunk ID หรือก็คือ ความยาวของ chunk

chunk มีลำดับ ID ที่เริ่มจาก primary chunk 4D4Dh, ซึ่งจะเป็น chunk แรกของ file เสมอโดย primary chunk เป็น main chunk หรือ chunk หลัก ด้านล่างคือ diagram ของลำดับ chunk ที่จะแสดงอยู่ใน file

MAIN3DS (0x4D4D)

|

+--EDIT3DS (0x3D3D)

| |

| +--EDIT_MATERIAL (0xAFFF)

| | |

| | +--MAT_NAME01 (0xA000) (See mli Doc)

| |

| +--EDIT_CONFIG1 (0x0100)

| +--EDIT_CONFIG2 (0x3E3D)

| +--EDIT_VIEW_P1 (0x7012)

| | |

| | +--TOP (0x0001)

| | +--BOTTOM (0x0002)

| | +--LEFT (0x0003)

| | +--RIGHT (0x0004)

| | +--FRONT (0x0005)

| | +--BACK (0x0006)

| | +--USER (0x0007)

| | +--CAMERA (0xFFFF)

| | +--LIGHT (0x0009)

| | +--DISABLED (0x0010)

| | +--BOGUS (0x0011)

| |

| +--EDIT_VIEW_P2 (0x7011)

| | |

| | +--TOP (0x0001)

| | +--BOTTOM (0x0002)

| | +--LEFT (0x0003)

| | +--RIGHT (0x0004)

```
| | +--FRONT        (0x0005)
| | +--BACK         (0x0006)
| | +--USER         (0x0007)
| | +--CAMERA       (0xFFFF)
| | +--LIGHT        (0x0009)
| | +--DISABLED     (0x0010)
| | +--BOGUS        (0x0011)
| |
| +--EDIT_VIEW_P3  (0x7020)
| +--EDIT_VIEW1    (0x7001)
| +--EDIT_BACKGR   (0x1200)
| +--EDIT_AMBIENT  (0x2100)
| +--EDIT_OBJECT   (0x4000)
| | |
| | +--OBJ_TRIMESH  (0x4100)
| | | |
| | | +--TRI_VERTEXL        (0x4110)
| | | +--TRI_VERTEXOPTIONS  (0x4111)
| | | +--TRI_MAPPINGCOORS   (0x4140)
| | | +--TRI_MAPPINGSTANDARD (0x4170)
| | | +--TRI FACEL1         (0x4120)
| | | | |
| | | | +--TRI_SMOOTH         (0x4150)
| | | | +--TRI_MATERIAL       (0x4130)
| | | |
| | | +--TRI_LOCAL          (0x4160)
| | | +--TRI_VISIBLE        (0x4165)
| | |
| | +--OBJ_LIGHT   (0x4600)
| | | |
| | | +--LIT_OFF          (0x4620)
```

```
| | | +--LIT_SPOT          (0x4610)
| | | +--LIT_UNKNWN01      (0x465A)
| | |
| | +--OBJ_CAMERA  (0x4700)
| | | |
| | | +--CAM_UNKNWN01      (0x4710)
| | | +--CAM_UNKNWN02      (0x4720)
| | |
| | +--OBJ_UNKNWN01 (0x4710)
| | +--OBJ_UNKNWN02 (0x4720)
| |
| +--EDIT_UNKNW01  (0x1100)
| +--EDIT_UNKNW02  (0x1201)
| +--EDIT_UNKNW03  (0x1300)
| +--EDIT_UNKNW04  (0x1400)
| +--EDIT_UNKNW05  (0x1420)
| +--EDIT_UNKNW06  (0x1450)
| +--EDIT_UNKNW07  (0x1500)
| +--EDIT_UNKNW08  (0x2200)
| +--EDIT_UNKNW09  (0x2201)
| +--EDIT_UNKNW10  (0x2210)
| +--EDIT_UNKNW11  (0x2300)
| +--EDIT_UNKNW12  (0x2302)
| +--EDIT_UNKNW13  (0x2000)
| +--EDIT_UNKNW14  (0xAFFF)
|
+--KEYF3DS (0xB000)
  |
  +--KEYF_UNKNWN01 (0xB00A)
  +--..... (0x7001) ( viewport, same as editor )
  +--KEYF_FRAMES  (0xB008)
```

+--KEYF_UNKNWN02 (0xB009)

+--KEYF_OBJDES (0xB002)

|

+--KEYF_OBJHIERARCH (0xB010)

+--KEYF_OBJDUMMYNAME (0xB011)

+--KEYF_OBJUNKNWN01 (0xB013)

+--KEYF_OBJUNKNWN02 (0xB014)

+--KEYF_OBJUNKNWN03 (0xB015)

+--KEYF_OBJPIVOT (0xB020)

+--KEYF_OBJUNKNWN04 (0xB021)

+--KEYF_OBJUNKNWN05 (0xB022)

chunk ที่จะพบในทุก file คือ color chunk

COL_RGB

COL_TRU

COL_UNK

รายละเอียดของ chunk แต่ละส่วน

*Main chunk

เมน chunk (คือ primary chunk 0x4D4D) ขนาดของ chunk ส่วนนี้จะเป็นขนาด fileเล็กที่สุดของheader ของ main chunkที่เป็นได้

และ file ยังมีอีก 2 main chunk คือ 3d-editor chunk และ keyframe chunk

id

3D3D start of Editor data (ตรงนี้เป็นจุดบอกว่า วัตถุอยู่ตรงไหน)

B000 start of Keyframe data (ส่วนระบุ keyframe)

ส่วนต่อขาก main chunk สามารถเป็น chunk ใดๆต่อไปก็ได้ตามแต่ระบุไว้ จาก diagram

*Subchunks of 3D3D (ส่วนย่อยขออง editor chunk)

id Description

0100 Part of configuration

1100 unknown

1200 Background Color

1201 unknown

1300 unknown

1400 unknown

1420 unknown

1450 unknown

1500 unknown

2100 Ambient Color Block

2200 fog ?

2201 fog ?

2210 fog ?

2300 unknown

3000 unknown

3D3E Editor configuration main block

4000 Definition of an Object

AFFF Start of material list

* Subchunks of AFFF - ระบุรายละเอียดของ รายการmaterial

* A000 - material name ระบุชื่อของ material

- chunkนี้ประกอบด้วยชื่อของ material ในรูปแบบ ASCIIZ สตริง

* Subchunks of 3D3E - Editor configuration (ระบุการ edit)

id Description

7001 Start of viewport indicator

7011 Viewport definition (type 2)

7012 Viewport definition (type 1)

7020 Viewport definition (type 3)

3D3E chunk เป็นส่วนที่ยังคลุมเครืออยู่มากเพราะประกอบด้วยส่วนข้อมูลที่ซ้ำซ้อนมาก chunk ที่สำคัญที่สุดคือ 7020 chunk นี้อธิบายรายละเอียด ของ 4 viewport ใน editor สิ่งที่จะพบได้ใน chunk นี้คือที่ byte 6และ 7 (นับจาก 6byteแรกของ header chunk และ pointer)
byte นี้จะมีข้อมูลที่ view ต้องใช้ ดังนี้

id	Description
0001	Top
0002	Bottom
0003	Left
0004	Right
0005	Front
0006	Back
0007	User
FFFF	Camera
0009	Light
10	Disabled

* Subchunks of 4000 - Object description Block (ข้อมูลรายละเอียดของวัตถุ)
ส่วนแรกของ subchunk 4000 คือ ASCIIZ สตริงของชื่อวัตถุ
ASCIIZ หมายถึง สตริงของตัวอักษร ที่ลงท้ายด้วย 0

วัตถุ สามารถเป็น ได้ทั้ง กล้อง แสง หรือ mesh

id	Description
4010	unknown
4012	shadow ?
4100	Triangular Polygon List (Contains only subchunks)
4600	Light
4700	Camera

* Subchunks of 4100 - Triangular Polygon List (รายละเอียดของโพลีกอนแบบ 3เหลี่ยม)

id	Description

4110 Vertex List

4111 Vertex Options

4120 Face List

4130 Face Material

4140 Mapping Coordinates

4150 Face smoothing group

4160 Translation Matrix

4165 Object visible/invisble

4170 Standard Mapping

* 4110 - Vertex List (รายละเอียดของ vertex)

start	end	size	type	name
0	1	2	unsigned int	Total vertices in object
2	5	4	float	X-value
6	9	4	float	Y-value
10	13	4	float	Z-value

byte ที่ 2..13 คือเลขจำนวนครั้งการเขียนซ้ำของ vertices ทั้งหมดในวัตถุ(object)

* 4111 - Vertex Options

2byte แรก : จำนวนของ vertices.

ต่อ short int ของแต่ละ vertex:

bit 0-7 0

bit 8-10 x

bit 11-12 0

bit 13 vertex selected in selection 3

bit 14 vertex selected in selection 2

bit 15 vertex selected in selection 1

บิตที่ 8-10 เป็นบิตสุ่ม จากการทดลอง save เป็น fileใหม่ โดยใช้ scene เดิม แสดงว่ามันสามารถเปลี่ยนแปลงได้

บิต 0-7 และ 11-12 มีผลกับ vertex ที่สามารถมองเห็นได้

4111 chunk สามารถลบออกได้โดยปราศจากผลกระทบมากนัก 3ds จะยังสามารถโหลด file ได้

* 4120 - Face list

start end size type name

0 1 2 unsigned int total polygons in object (numpoly)

2 3 2 unsigned int number of vertex A

4 5 2 unsigned int number of vertex B

6 7 2 unsigned int number of vertex C

8 9 2 unsigned int face info (*)

ซ้ำส่วน 'numpoly' สำหรับแต่ละ polygon

ค่า int 3 ตัวแรกคือ 3 vertices ของ face

0 หมายถึง vertex แรกที่ระบุไว้ใน vertex list

ความหมายของลำดับ จะเป็นทิศทางปกติของแต่ละ face

สมมติเป็นการขันน็อต (น็อตมาตรฐานทั่วๆไป) ถ้าขันน็อตไปในทางที่ vertices ชี้จะเป็น ทิศปกติ

ถ้า vertices ให้ลำดับมาเป็น A B C:

จากรูปหมายความว่า การขันน็อตออก จุดปกติจะชี้ออกมาจาก จาก

(*)เลขหมายนี้เป็น เลข binary ที่ขยายออกได้เป็น 3ค่า คย. 0x0006 สามารถแปลงเป็น

ค่า 110 binary เราจะอ่านค่า binary เป็น 1 1 0, ค่านี้สามารถพบใน 3d-studio ascii file

ในลักษณะ AB:1 BC:1 AC:0 อาจจะเป็นการระบุลำดับของ vertices ตย. AB:1 เป็น เส้นปกติ (normal line) จาก A ไป B แต่ AB:0 จะหมายถึงเส้นจาก B ไป A

bit 0 AC visibility

bit 1 BC visibility

bit 2 AB visibility

bit 3 Mapping (if there is mapping for this face)

bit 4-8 0 (not used ?)

bit 9-10 x (chaotic ???)

bit 11-12 0 (not used ?)

bit 13 face selected in selection 3

bit 14 face selected in selection 2

bit 15 face selected in selection 1

* 4130 - Face Material Chunk

ถ้าวัตถุใช้ material ชนิด default จะ ไม่มี chunkเบอร์ 4130
ในความเป็นจริง จะมี chunk เบอร์ 4130 สำหรับแต่ละ material ที่ใช้สำหรับวัตถุ

4130 chunk แต่ละตัวจะเริ่มด้วย ASCIIZ ของ material หลังจากตัวอักขระ null จะเป็น short int ที่ให้จำนวนของ face ของวัตถุที่เกี่ยวข้องกับ material จากนั้นจะเป็นส่วนรายละเอียดของ face นี้, 0000หมายถึง face แรกของ face list (4120)

* 4140 Mapping coordinates.

2byte แรก: จำนวนของ vertices
ต่อจากนั้น จะเป็น ค่า float 2ค่าของแต่ละ vertexที่แสดงตำแหน่งของการ mapping coordinates
หมายความว่า ถ้าจุดจุดหนึ่งอยู่ที่กึ่งกลางของ map มันจะเป็นค่า 0.5,0.5 ทีการ mapping coordinates

* 4150 - Face Smoothing Group

nfaces*4 bytes

ถ้าอ่านค่าออกมาเป็น long int, bit nth จะระบุว่ามี face เป็นของ nth smoothing group หรือไม่

* 4160 Local axis

ข้อมูลของ Local axis

3 block แรกจะบอก local axis X Y Z (ตำแหน่งของวัตถุ)

และ block สุดท้ายจะบอก จุดกึ่งกลางของวัตถุ

* 4170 Standard mapping

2 bytesแรก: ชนิดของการ mapping

0 => พื้นเรียบหรือชนิดพิเศษ(ในกรณีนี้จะเป็นพวกmappingจาก lofterข้อมูลของ chunkนี้จะไม่เกี่ยวเนื่องกัน)

1 -> ทรงกระบอก

2 => ทรงกลม

ค่า float 21 ค่าถัดมาจะอธิบายการ mapping

* 4600 - Light

start	end	size	type	name
0	3	4	float	Light pos X
4	7	4	float	Light pos Y
8	11	4	float	Light pos Z

หลังจากโครงสร้างส่วนนี้จะเป็นรายละเอียดของ chunk

id	Description (full description later)
0010	RGB color
0011	24 bit color
4610	Light is a Spot light
4620	Light is off/on (Boolean)

* 4610 - Spot Light

start	end	size	type	name
0	3	4	float	Target pos X
4	7	4	float	Target pos X
8	11	4	float	Target pos X
12	15	4	float	Hotspot
16	19	4	float	Falloff

* 0010 - RGB Color

start	end	size	type	name
0	3	4	float	Red
4	7	4	float	Green
8	11	4	float	Blue

* 0011 - RGB Color - 24 bit

start	end	size	type	name
0	1	1	byte	Red
1	1	1	byte	Green
2	2	1	byte	Blue

* 4700 - Camera

อธิบายรายละเอียดของกล้องในฉาก

start	end	size	type	name
0	3	4	float	Camera pos X
4	7	4	float	Camera pos Y
8	11	4	float	Camera pos Z
12	15	4	float	Camera target X
16	19	4	float	Camera target X

20 23 4 float Camera target X

24 27 4 float Camera bank (rotation angle)

28 31 4 float Camera lens

ต่อไปเราจะมาดูส่วนสำคัญอีกส่วนคือ keyframe chunk

* Keyframer chunk

id	Description
B00A	unknown
7001	See first description of this chunk
B008	Frames
B009	unknown
B002	Start object description

* B008 - Frame information

โครงสร้างพื้นฐานอธิบายข้อมูลของเฟรม

start	end	size	type	name
0	3	4	unsigned long	start frame
4	7	4	unsigned long	end frame

*B002 - Start of Object info

chunkย่อย (Subchunk)

id	Description
B010	Name & Hierarchy
B011*	Name Dummy Object
B013	unknown
B014*	unknown
B015	unknown
B020	Objects pivot point ?
B021	unknown

B022 unknown

* B010 - Namc & Hicrarchy dcscriptor

start	end	size	type	name
0	?	?	ASCIIZ	Object name
?	?	2	unsigned int	unknown
?	?	2	unsigned int	unknown
?	?	2	unsigned int	Hierarchy of Object

ลำดับความสำคัญของวัตถุเป็นบิตที่ซับซ้อนแต่มีประสิทธิภาพ
วัตถุแต่ละชิ้นในฉากจะมี หมายเลขระบุลำดับภายใน tree วัตถุเกือบทั้งหมดที่อยู่ภายใน file 3DS จะถูกจัดลำดับไว้ใน tree วัตถุที่อยู่ที่ root จะมีหมายเลขเป็น –1 (FFFF), ขณะที่อ่าน file จะมีตัวนับเก็บจำนวนวัตถุไว้ และจะบวกเพิ่มขึ้นเมื่อถึงวัตถุที่เป็น children node ของวัตถุชิ้นก่อนหน้า
ตัวอย่าง

ลำดับขั้นวัตถุ ชื่อวัตถุ		
A	-1	
B	0	
C	1	
D	2	
E	1	สังเกตที่ลำดับของวัตถุแต่ละชิ้น
F	4	
G	5	
H	1	
I	7	
J	8	
K	0	

L 10

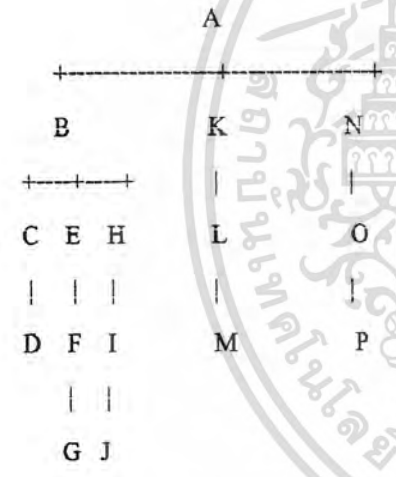
M 11

N 0

O 13

P 14

เป็นแผนภาพ tree ดังนี้



ถ้าวัตถุชื่อว่า $$$DUMMY จะมี chunk เพิ่มขึ้นมา

* B011 - Dummy objects name.

ชื่อ object dummy เป็น ASCIIZ สตริง

* B020 - Pivot Point (จุดหมุน)

จุดหมุนของวัตถุ

start end size type name

0 3 4 float unknown

4 7 4 float unknown

8 11 4 float unknown

12 16 4 float unknown

16 19 4 float unknown

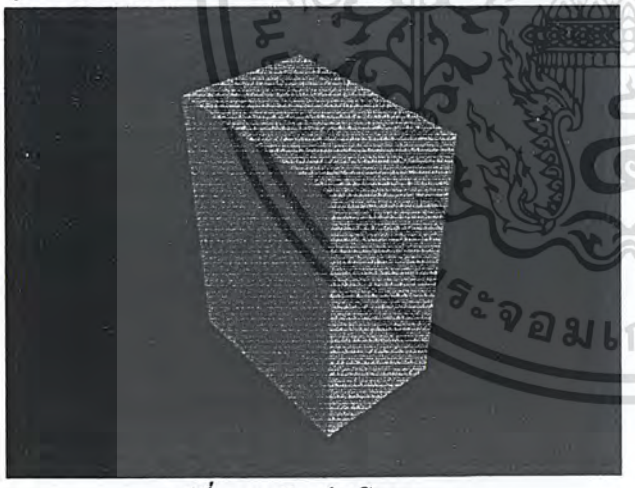
20 23 4 float unknown

24 27 4 float Pivot Y

28 32 4 float Pivot X

File Export Property

ข้อมูลต่างๆของ Object จะปรากฏใน Format ไฟล์ต่างๆที่ Export โดย 3D Studio Max ซึ่ง Format ไฟล์หลาย Format ไม่สามารถนำไปใช้ในเกมเอ็นจิน ได้ เช่น Format X ที่ใช้ได้ใน Direct X แต่กลับใช้ไม่ได้กับ Open GL เป็นต้น จึงต้องศึกษา ไฟล์ Format อื่นโดยเขียนโปรแกรมตัดสตริงเพื่อนำข้อมูลไปใช้ ในที่นี้จะขออธิบายข้อมูลส่วนต่างๆด้วย Format ไฟล์ *.ASE



รูปที่2-34 ตัวอย่างโมเดล

จากรูป เราทำการสร้างโมเดล สี่เหลี่ยม 1 รูปที่มีจุดศูนย์กลางอยู่ที่ (0,0,0) ขนาด กว้าง 10.0 หน่วย ยาว 15.0 หน่วย และ สูง 20.0 หน่วย ซึ่ง export ได้ไฟล์ box.ASE มีข้อมูลดังข้างล่างนี้

// File box.ASE

*3DSMAX_ASCIIEXPORT 200

*COMMENT "AsciiExport Version 2.00 - Wed Sep 19 14:41:29 2001"

```
*SCENE {
        *SCENE_FILENAME ""
        *SCENE_FIRSTFRAME 0
        *SCENE_LASTFRAME 100
        *SCENE_FRAMESPEED 30
        *SCENE_TICKSPERFRAME 160
        *SCENE_BACKGROUND_STATIC 0.0000         0.0000  0.0000
        *SCENE_AMBIENT_STATIC 0.0431    0.0431  0.0431
}
*MATERIAL_LIST {
        *MATERIAL_COUNT 0
}
*GEOMOBJECT {
        *NODE_NAME "Box"
        *NODE_TM {
                *NODE_NAME "Box"
                *INHERIT_POS 0 0 0
                *INHERIT_ROT 0 0 0
                *INHERIT_SCL 0 0 0
                *TM_ROW0 1.0000     0.0000  0.0000
                *TM_ROW1 0.0000     1.0000  0.0000
                *TM_ROW2 0.0000     0.0000  1.0000
                *TM_ROW3 0.0000     0.0000  0.0000
                *TM_POS 0.0000      0.0000  0.0000
                *TM_ROTAXIS 0.0000  0.0000  0.0000
                *TM_ROTANGLE 0.0000
                *TM_SCALE 1.0000    1.0000  1.0000
                *TM_SCALEAXIS 0.0000        0.0000  0.0000
                *TM_SCALEAXISANG 0.0000
        }
        *MESH {
```

```
*TIMEVALUE 0
*MESH_NUMVERTEX 8
*MESH_NUMFACES 12
*MESH_VERTEX_LIST {
        *MESH_VERTEX  0    -7.5000 -5.0000 0.0000
        *MESH_VERTEX  1    7.5000  -5.0000 0.0000
        *MESH_VERTEX  2    -7.5000 5.0000  0.0000
        *MESH_VERTEX  3    7.5000  5.0000  0.0000
        *MESH_VERTEX  4    -7.5000 -5.0000 20.0000
        *MESH_VERTEX  5    7.5000  -5.0000 20.0000
        *MESH_VERTEX  6    -7.5000 5.0000  20.0000
        *MESH_VERTEX  7    7.5000  5.0000  20.0000
}
*MESH_FACE_LIST {
        *MESH_FACE  0:  A:  0 B:  2 C:  3 AB:  1 BC:  1 CA:  0
*MESH_SMOOTHING 2 *MESH_MTLID 1
        *MESH_FACE  1:  A:  3 B:  1 C:  0 AB:  1 BC:  1 CA:  0
*MESH_SMOOTHING 2 *MESH_MTLID 1
        *MESH_FACE  2:  A:  4 B:  5 C:  7 AB:  1 BC:  1 CA:  0
*MESH_SMOOTHING 3 *MESH_MTLID 0
        *MESH_FACE  3:  A:  7 B:  6 C:  4 AB:  1 BC:  1 CA:  0
*MESH_SMOOTHING 3 *MESH_MTLID 0
        *MESH_FACE  4:  A:  0 B:  1 C:  5 AB:  1 BC:  1 CA:  0
*MESH_SMOOTHING 4 *MESH_MTLID 4
        *MESH_FACE  5:  A:  5 B:  4 C:  0 AB:  1 BC:  1 CA:  0
*MESH_SMOOTHING 4 *MESH_MTLID 4
        *MESH_FACE  6:  A:  1 B:  3 C:  7 AB:  1 BC:  1 CA:  0
*MESH_SMOOTHING 5 *MESH_MTLID 3
        *MESH_FACE  7:  A:  7 B:  5 C:  1 AB:  1 BC:  1 CA:  0
*MESH_SMOOTHING 5 *MESH_MTLID 3
```

```
                *MESH_FACE  8:  A:  3 B:  2 C:  6 AB:  1 BC:  1 CA:  0
*MESH_SMOOTHING 6 *MESH_MTLID 5
                *MESH_FACE  9:  A:  6 B:  7 C:  3 AB:  1 BC:  1 CA:  0
*MESH_SMOOTHING 6 *MESH_MTLID 5
                *MESH_FACE 10:  A:  2 B:  0 C:  4 AB:  1 BC:  1 CA:  0
*MESH_SMOOTHING 7 *MESH_MTLID 2
                *MESH_FACE 11:  A:  4 B:  6 C:  2 AB:  1 BC:  1 CA:  0
*MESH_SMOOTHING 7 *MESH_MTLID 2
        }
        *MESH_NUMTVERTEX 0
        *MESH_NUMCVERTEX 0
        *MESH_NORMALS {
            *MESH_FACENORMAL 0      0.0000  0.0000  -1.0000
                *MESH_VERTEXNORMAL 0    0.0000  0.0000  -1.0000
                *MESH_VERTEXNORMAL 2    0.0000  0.0000  -1.0000
                *MESH_VERTEXNORMAL 3    0.0000  0.0000  -1.0000
            *MESH_FACENORMAL 1      0.0000  0.0000  -1.0000
                *MESH_VERTEXNORMAL 3    0.0000  0.0000  -1.0000
                *MESH_VERTEXNORMAL 1    0.0000  0.0000  -1.0000
                *MESH_VERTEXNORMAL 0    0.0000  0.0000  -1.0000
            *MESH_FACENORMAL 2      0.0000  0.0000  1.0000
                *MESH_VERTEXNORMAL 4    0.0000  0.0000  1.0000
                *MESH_VERTEXNORMAL 5    0.0000  0.0000  1.0000
                *MESH_VERTEXNORMAL 7    0.0000  0.0000  1.0000
            *MESH_FACENORMAL 3      0.0000  0.0000  1.0000
                *MESH_VERTEXNORMAL 7    0.0000  0.0000  1.0000
                *MESH_VERTEXNORMAL 6    0.0000  0.0000  1.0000
                *MESH_VERTEXNORMAL 4    0.0000  0.0000  1.0000
            *MESH_FACENORMAL 4      0.0000  -1.0000 0.0000
                *MESH_VERTEXNORMAL 0    0.0000  -1.0000 0.0000
                *MESH_VERTEXNORMAL 1    0.0000  -1.0000 0.0000
```

```
        *MESH_VERTEXNORMAL 5     0.0000  -1.0000 0.0000
*MESH_FACENORMAL 5          0.0000  -1.0000 0.0000
        *MESH_VERTEXNORMAL 5     0.0000  -1.0000 0.0000
        *MESH_VERTEXNORMAL 4     0.0000  -1.0000 0.0000
        *MESH_VERTEXNORMAL 0     0.0000  -1.0000 0.0000
*MESH_FACENORMAL 6          1.0000  0.0000  0.0000
        *MESH_VERTEXNORMAL 1     1.0000  0.0000  0.0000
        *MESH_VERTEXNORMAL 3     1.0000  0.0000  0.0000
        *MESH_VERTEXNORMAL 7     1.0000  0.0000  0.0000
*MESH_FACENORMAL 7          1.0000  0.0000  0.0000
        *MESH_VERTEXNORMAL 7     1.0000  0.0000  0.0000
        *MESH_VERTEXNORMAL 5     1.0000  0.0000  0.0000
        *MESH_VERTEXNORMAL 1     1.0000  0.0000  0.0000
*MESH_FACENORMAL 8          0.0000  1.0000  0.0000
        *MESH_VERTEXNORMAL 3     0.0000  1.0000  0.0000
        *MESH_VERTEXNORMAL 2     0.0000  1.0000  0.0000
        *MESH_VERTEXNORMAL 6     0.0000  1.0000  0.0000
*MESH_FACENORMAL 9          0.0000  1.0000  0.0000
        *MESH_VERTEXNORMAL 6     0.0000  1.0000  0.0000
        *MESH_VERTEXNORMAL 7     0.0000  1.0000  0.0000
        *MESH_VERTEXNORMAL 3     0.0000  1.0000  0.0000
*MESH_FACENORMAL 10         -1.0000 0.0000  0.0000
        *MESH_VERTEXNORMAL 2     -1.0000 0.0000  0.0000
        *MESH_VERTEXNORMAL 0     -1.0000 0.0000  0.0000
        *MESH_VERTEXNORMAL 4     -1.0000 0.0000  0.0000
*MESH_FACENORMAL 11         -1.0000 0.0000  0.0000
        *MESH_VERTEXNORMAL 4     -1.0000 0.0000  0.0000
        *MESH_VERTEXNORMAL 6     -1.0000 0.0000  0.0000
        *MESH_VERTEXNORMAL 2     -1.0000 0.0000  0.0000
    }
}
```

*PROP_MOTIONBLUR 0

*PROP_CASTSHADOW 1

*PROP_RECVSHADOW 1

*WIREFRAME_COLOR 0.8392 0.8980 0.6510

}

ข้อมูลไฟล์นี้ เป็นเพียงข้อมูลของ โมเดลรูป สี่เหลี่ยมเท่านั้นซึ่งบอกค่าต่างๆ ได้แก่ จำนวน vertex, ตำแหน่งของ vertex, จำนวน face, ค่า vertex ที่ทำให้เกิด face, matrix transform, สถานะ edge, ค่า vector normal ต่างๆ และ รายละเอียดส่วนของ สีแสงพื้นผิวต่างๆ

จากรายละเอียดดังกล่าว พบว่าโมเดลนี้เป็นเพียง 1 object เท่านั้นและยังไม่มีข้อมูลส่วนของการทำ Animation ซึ่งการเอาไปใช้งานจริง จำนวน object จะมีมาก เมื่อรวมกับข้อมูลส่วนของแสง สี พื้นผิว และข้อมูลการทำ animation Formatไฟล์นี้จะมีขนาดใหญ่มาก ยิ่งโมเดลที่มีลักษณะที่เนียนโค้งมากเพียงใดก็ยิ่งมี vertex มาก และมีข้อมูลส่วนอื่นๆมาก ทำให้ไฟล์มีขนาดใหญ่มากเกินที่จะเอาไปใช้งานได้ เนื่องจากไฟล์มีข้อมูลที่ซ้ำซ้อนมาก เมื่อ object มีการทำ animation

X File format

Format ไฟล์อีกชนิดที่เป็นที่นิยมใช้กันคือ file .X ของ Microsoft ซึ่งรวมไว้ใน SDK Direct X ไฟล์ชนิดนี้ถูกออกแบบมาเพื่อเก็บข้อมูลภาพ 3 มิติ โดยเฉพาะเพื่อใช้งานร่วมกับ Direct 3D โดยมีแนวคิดหลักคือต้องเก็บวัตถุไว้ใช้ในการแสดงผลต่อไปได้ และต้องแสดงความสัมพันธ์ระหว่างวัตถุได้ด้วย ข้อมูลจะถูกเก็บในลักษณะความสัมพันธ์แบบ parent, child และนำความสัมพันธ์นี้ไปแสดงผลเช่น meshที่ประกอบเป็น frames หรือ materials ที่ประกอบเป็น meshs และ file format ควรที่จะมีความสามารถมากพอที่จะสนับสนุนการทำ forward หรือ backward ได้ด้วย เช่นการคำนวณการเคลื่อนที่ไปข้างหน้าและถอยหลังย้อนกลับได้

x file format สามารถเก็บได้ทั้งแบบ text file และแบบ binary file ด้วยกัน x file format สามารถสร้างได้จากการใช้โปรแกรม 3D Studio max สร้างวัตถุที่เราต้องการแล้ว export เก็บไว้เป็น file นามสกุล 3DS จากนั้นใช้ โปรแกรม conv3ds แปลง file นามสกุล 3DS นี้เป็น file นามสกุล .X เพื่อใช้งานร่วมกับ Direct 3D

ตัวอย่าง File .X

```
xof 0302txt 0064      // File Header
Header {
1;
0;
```

```
1;      }
Frame x3ds_Box01 {                          // Transform Matrix for Frame Animation
FramcTransformMatrix {
 1.000000, 0.000000, 0.000000, 0.000000,
 0.000000, 0.000000, -1.000000, 0.000000,
 0.000000, 1.000000, 0.000000, 0.000000,
 -1.324503, 3.311258, -0.000000, 1.000000;; }
Mesh Box01 {                                //Mesh Object
 8;
 -25.827814; 0.000000; -23.841055;,
 25.827814; 0.000000; -23.841055;,
 -25.827814; -0.000000; 23.841055;,
 25.827814; -0.000000; 23.841055;,
 -25.827814; -56.953636; -23.841055;,
 25.827814; -56.953636; -23.841055;,
 -25.827814; -56.953632; 23.841055;,
 25.827814; -56.953632; 23.841055;;

 12;
 3;2,3,0;,
 3;1,0,3;,
 3;5,7,4;,
 3;6,4,7;,
 3;1,5,0;,
 3;4,0,5;,
 3;3,7,1;,
 3;5,1,7;,
 3;2,6,3;,
 3;7,3,6;,
 3;0,4,2;,
 3;6,2,4;;
```

```
MeshNormals {                                   //Mesh Normal use for Lighting
 24;
 0.000000;1.000000;0.000000;,
 0.000000;0.000000;-1.000000;,
 -1.000000;0.000000;0.000000;,
 0.000000;1.000000;0.000000;,
 0.000000;0.000000;-1.000000;,
 1.000000;0.000000;0.000000;,
 0.000000;1.000000;0.000000;,
 0.000000;0.000000;1.000000;,
 -1.000000;0.000000;0.000000;,
 0.000000;1.000000;0.000000;,
 1.000000;0.000000;0.000000;,
 0.000000;0.000000;1.000000;,
 0.000000;-1.000000;0.000000;,
 0.000000;0.000000;-1.000000;,
 -1.000000;0.000000;0.000000;,
 0.000000;-1.000000;0.000000;,
 0.000000;0.000000;-1.000000;,
 1.000000;0.000000;0.000000;,
 0.000000;-1.000000;0.000000;,
 0.000000;0.000000;1.000000;,
 -1.000000;0.000000;0.000000;,
 0.000000;-1.000000;0.000000;,
 1.000000;0.000000;0.000000;,
 0.000000;0.000000;1.000000;;
 12;
 3;6,9,0;,
 3;3,0,9;,
 3;15,21,12;,
 3;18,12,21;,
```

```
 3;4,16,1;,
 3;13,1,16;,
 3;10,22,5;,
 3;17,5,22;,
 3;7,19,11;,
 3;23,11,19;,
 3;2,14,8;,
 3;20,8,14;;
 }
}
}
```

Max script overview

เป็น script language ที่อยู่ใน 3ds maxมีความสามารถในการสร้าง ระบบ coordinate, การสร้างวัตถุ, การใช้ material, การใช้ mode animation กับ automatic keyframing (สร้าง keyframe แบบอัตโนมัติ), และการเรียกใช้ object โดยใช้ลำดับขั้นตาม 3ds max object hierachy

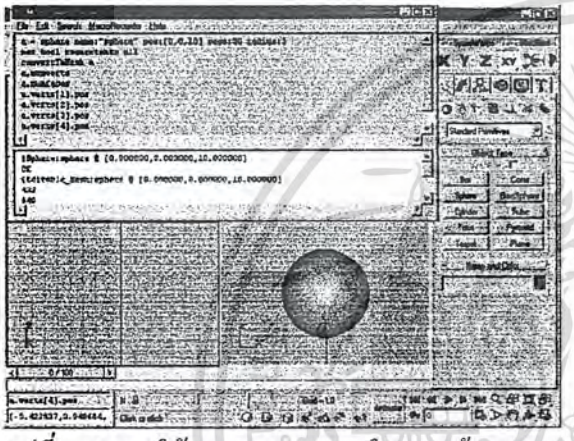
syntax ของภาษาเป็นแบบง่ายๆใช้คู่กับ listenner window มีลักษณะการทำงานแบบ command linc, ความสามารถในการติดตั้ง script เป็นปุ่ม toolbar และจดจำการทำงานของ user ไว้เป็น คำสั่งของ maxscript

MAXScript

maxscript มีความสามารถเพียงพอในการโปแกรมงานที่ซับซ้อนด้วยคุณสมบัติ เช่น 3D เวคเตอร์, เมตริกซ์, คณิตศาสตร์ quartemion, maxscript เหมาะกับงานที่ใช้จำนวน object ขนาดใหญ่, ตัวอย่าง เช่นการ selection ที่ซับซ้อน, การสร้าง ดวงดาว แบบสุ่ม, หรือการวางวัตถุในpatternที่ถูกต้องแน่นอน maxscript สามารถใช้ในการทำ object, material, textures, การrender และการทำ atmosphereหรือ การสร้าง mesh object maxscript สนับสนุน format text input และ ouput ทำให้สามารถสร้าง report ได้โดยตรงจาก 3ds max file และอ่าน file ที่ประกอบไปด้วย layout, naming, texture, และรายละเอียดอื่นๆ รวมทั้งยังสามารถ export จาก โปรแกรมการจัดการโปรเจคต์ อื่นๆได้อีกด้วย maxscript สามารถใช้เป็นtool ในการ importได้ โดยการนำ output ของ maxscript ที่ประกอบไปด้วย ชุดคำสั่งในการสร้าง object

MAX Script คือ ภาษาเฉพาะ(built-in scripting language) ของ 3D Studio MAX และ 3D Studio VIZ ใช้สำหรับสร้างโมเดล 3มิติ ด้วยภาษาเขียนเฉพาะของโปรแกรม ซึ่งจะมีประโยชน์มาก กับเกมเอ็นจินที่ต้องการ import ข้อมูลโมเดลเองตามต้องการ จาก โมเดลที่สร้างโดย โปรแกรม 3D Studio MAX

ข้อมูลของโมเดลต่างๆที่นำไปใช้ในเกมเอ็นจินจะถูก export ใน Format ของเราเองตามต้องการ ซึ่งจะคล่องตัวมาก สำหรับเกมเอ็นจินที่พัฒนาเองโดยใช้ Open GL เพราะ Open GL ไม่สามารถนำ file.X มาใช้ได้เหมือน Direct X ในรายงานฉบับนี้จะขอแสดงการใช้งาน พอสังเขป



รูปที่ 2-35 การใช้งาน MAX Script ในการสร้าง ทรงกลมด้วย ภาษา MAX Script

จากรูปเราได้ทำการสร้าง ทรงกลมโดยการเขียนโปรแกรมที่มี source code ดังนี้

```
a = sphere name:"sphere" pos:[0,0,10] segs:30 radius:5  //1
max tool zoomextents all
convertToMesh a
a.numverts
a.numfaces
a.verts[1].pos
a.verts[2].pos
a.verts[3].pos
a.verts[4].pos                                          //9
```

จะเห็นได้ว่าไม่ต่างจากภาษาโปรแกรมอื่นทั่วไปมากนัก โดย

บรรทัดที่ 1 เป็นการสร้าง ทรงกลมตามชื่อ, ตำแหน่ง, ความละเอียด, และขนาด ตามลำดับ

บรรทัดที่ 2 เป็นการปรับกล้องให้เห็นได้เหมาะสม

บรรทัดที่ 3 ทำการ Convert โมเดลให้เป็น mesh เพื่อดูข้อมูลในระดับ vertex

บรรทัดที่ 4 หาจำนวน vertexs ทั้งหมด

บรรทัดที่ 5 หาจำนวน faces ทั้งหมด

,บรรทัดที่ 6-9 ทำการหาตำแหน่งของ vertexs ในแต่ละ vertex

จาก source code ดังกล่าวมี output ออกดังนี้

```
$Sphere:sphere @ [0.000000,0.000000,10.000000]
OK
$Editable_Mesh:sphere @ [0.000000,0.000000,10.000000]
422
840
[0,0,15]
[0,1.03956,14.8907]
[-0.216136,1.01684,14.8907]
[-0.422827,0.949684,14.8907]
```

จะเห็นว่า เราสามารถดูข้อมูลได้ตามต้องการ นำไปสู่การ export ไฟล์ตาม Format ไฟล์ที่เราต้องการ โดยการเซฟข้อมูลต่างๆของโมเดลลงสู่ไฟล์ตาม Format ที่เราเข้าใจ และทำการ optimize ให้ข้อมูลซ้ำซ้อนน้อยที่สุด หรือว่าจะเขียนแยกไฟล์เป็นส่วนๆ อาจจะเป็น ไฟล์ animation ที่มีแต่ข้อมูล animation อย่างเดียว กับข้อมูลเนื้ออีกส่วนหนึ่ง เพื่อว่าจะมี โมเดลอื่นต้องการ แชร์ข้อมูลส่วน animation ก็จะทำให้ประหยัด space ในการเก็บไปด้วย

2.4 Low Polygon Modeling

สิ่งสำคัญในการสร้าง Model ที่จะใช้ในเกมนั้นก็คือการกำหนดว่าควรจะใช้ Polygon ทั้งหมดเท่าไร เกมในปัจจุบันจำนวน polygon ต่อ object นั้นได้เพิ่มมากขึ้น เนื่องจากความเร็วในการประมวลผลที่เพิ่มมากขึ้นของ Hardware ในปัจจุบัน แต่เราก็ยังต้องกำหนดว่า Polygon เท่าไรถึงจะถือว่ามีความเหมาะสมต่อการใช้งาน ยกตัวอย่าง เช่น หินที่วางข้างทาง ไม่ได้มีความสำคัญต่อตัวเกมหลักเท่าไร Polygon ที่เหมาะสมควรจะประมาณเพียง 10-30 Polygons ก็พอ อย่างไรก็ตามวัตถุอย่างเช่น น้ำพุ หรือ ตึก จะเป็นจุดสนใจของสายตา จึงต้องเพิ่มรายละเอียดเข้าไปให้มากกว่าซึ่งจำนวน Polygon ควรจะมีขนาดเพิ่มถึง 350-550 Polygons เลยทีเดียว ประการสุดท้ายที่จะใช้ในการพิจารณาคือความเร็วของเกม เมื่อวัตถุทั้งหมดถูกรวมเข้าด้วยกันไว้ใน view window คอมพิวเตอร์รุ่นเก่าบางชนิดไม่สามารถที่จะแสดงผลฉากที่มี 5000 Polygons ใน 1 view window นั้น

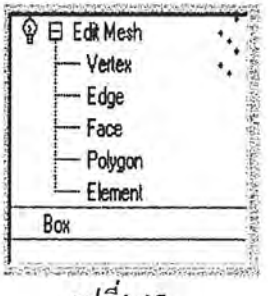
ในขณะที่คอมพิวเตอร์รุ่นใหม่สามารถรองรับการแสดงผลได้ถึง 6000-7000 Polygons อย่างไม่มีปัญหา มันเป็นเรื่องดีที่จะสร้างเกมที่สามารถเล่นได้อย่างกว้างขวางในหลายๆ users และเล่นได้ในความเร็วต่ำๆขณะที่ users ที่มีเครื่องระดับสูงสามารถเล่นเกมได้ใน frame rates ที่สูงๆ ในส่วนนี้เราจะได้มาพูดถึงวิธีการสร้าง Model ที่มีจำนวน Polygons น้อยกัน

1.ขั้นแรกสร้าง Box ขึ้นมาจาก Standard Primitives ที่ Top Viewport ให้มีขนาดดังนี้ Length=40, Width=40, Height=43, Length Segs=1, Width Segs=2, Height Segs=4

2.ใส่ Edit mesh Modifier ให้กับ Box ที่สร้างขึ้น

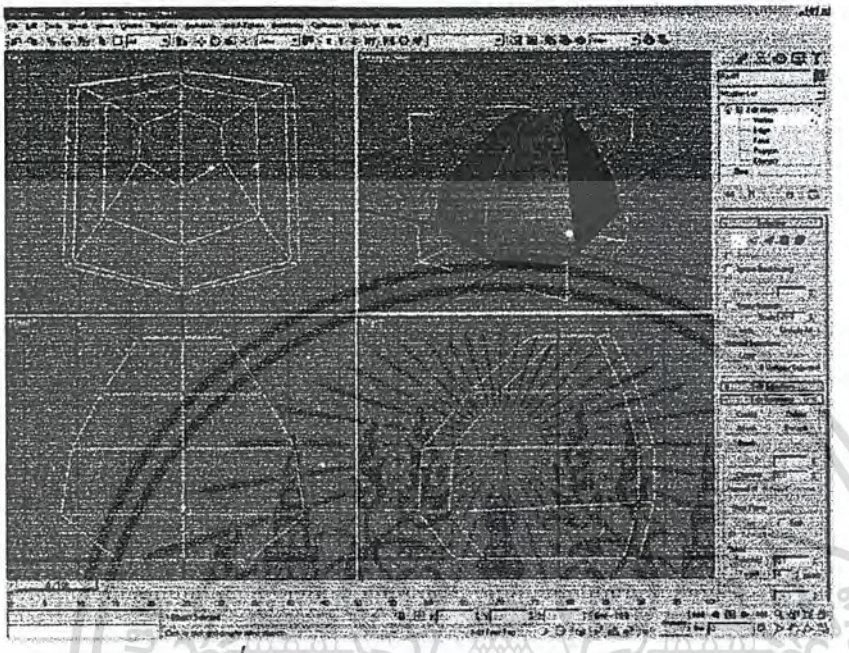
รูปที่2-36

3.เปิดตัวเลือกของ Edit mesh Modifier เลือกโหมด Vertex เพื่อจะสามารถทำงานกับ Vertex (จุด)ได้



รูปที่2-37

4.ต่อไปใช้ Non Uniform Scale Tool, Move Tool เพื่อทำการย้ายตำแหน่ง Vertex ของ Model ดึง Vertices ด้านหน้าออกมาเพื่อทำเป็นหน้าอก และดึงเส้นด้านหลังออกมาเล็กน้อยเป็นแนวแผ่นหลัง ปรับให้ส่วนบนของ Box เล็กลงตามรูป



รูปที่2-38 Low Polygon

5.ใส่ Mesh smooth Modifier ให้กับ Model การทำเช่นนี้เพื่อให้สามารถมองเห็น Model ในรูปแบบที่มี Resolution สูงๆพร้อมไปกับ Model ในรูปแบบที่ Resolution ต่ำๆได้ในเวลาเดียวกัน

5.1 selectที่ Model แล้วเลือก Vertex เพิ่ม Surfaces > MeshSmooth Modifier เข้าไปที่บนสุดของ Stack

5.2 ปรับค่า Iterations ไปที่ 2

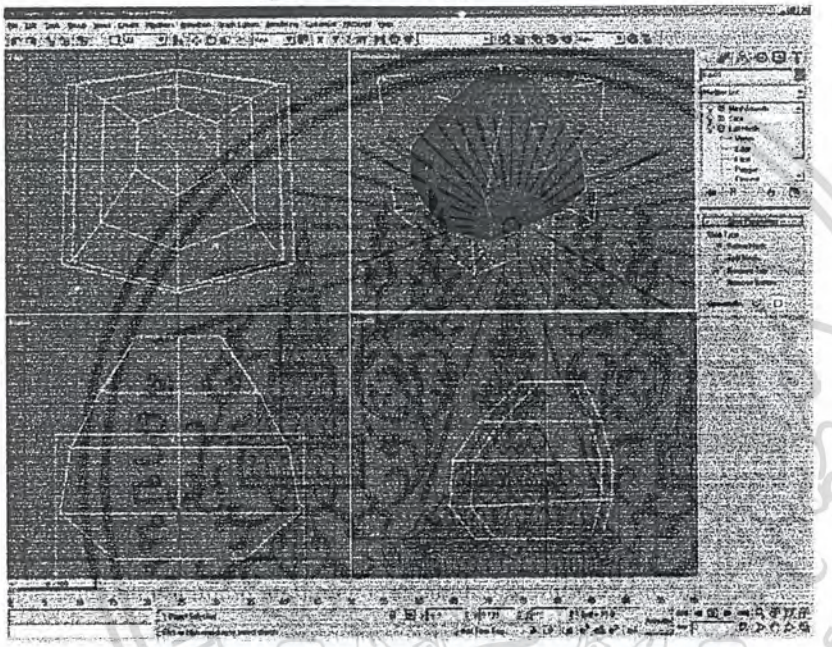
5.3 เห็นได้ว่า model กลายเป็นพื้นผิวเรียบ เราสามารถจะปรับความเรียบของ Model ให้มากขึ้นหรือน้อยลงได้ตามค่า Iterations อย่างไรก็ตามการทำให้ Model มีความเรียบมาก ย่อมหมายถึงมีจำนวน Vertex เพิ่มมากขึ้น ตามไปด้วย

6.กดเลือก Edit Mesh ด้านล่างของ Mesh Smooth ภายใน Stack Modifier สังเกตว่า Model จะกลับมาอยู่ในสภาพ Low Resolution คุณอาจจะต้องใช้ป่ ม Tooggle เพื่อสลับปิดเปิดระหว่าง Hi resolution กับ Low Resolution อย่าไรก็ตามคุณจะต้องเพิ่ม Modifier ข้างล่าง Mesh Smooth ที่ Stack Modifier เสมอ เพื่อให้ Mesh Smooth มีผลกับ Model

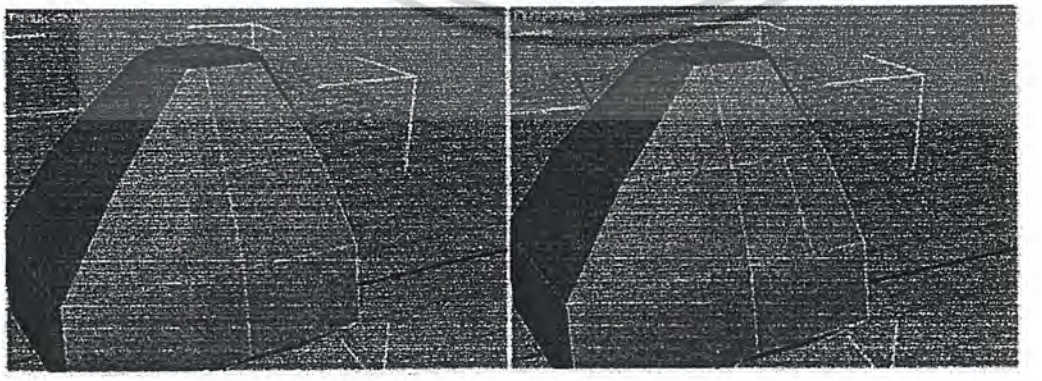
7.จาก Modifier List เลือก Slice เพื่อทำการ Slice ตัว Model ออก Slice คือการแบ่ง Edge ออกจากกันหรือจะพูดว่าเป็นการเพิ่ม Vertex เข้าไปใน Edge ก็ว่าได้ ทำการ Slice ตัดขวาง Model ตามแนวตั้ง ดังรูปตัวอย่าง

(เหตุผลที่ไม่ใช้ Slice จาก Editable mesh แต่ใช้ slice Plane จาก Slice Modifierแทน เพราะว่า Slice Modifier จะไม่ยุ่งกับ Hidden Edge แต่ Slice ของ Editable mesh จะทำการ Slice Hidden Edge ไปด้วย ซึ่งเราต้องมาทำการเอาออกโดยการทำ Target Weld ซึ่งไม่สะดวก)

รูปที่2-39 Low Polygon

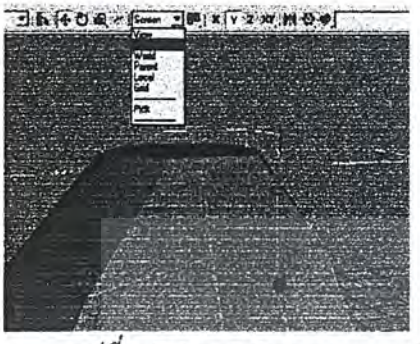


8.ที่ Edit Geometry ของ Mesh Modifier ให้ใช้ กดปุ่ม Cut เพื่อทำการ Cut ด้านข้างของ Model (การ Cut ก็เหมือนการ Slice คือเป็นการตัด Edge ออกเป็น 2ส่วนโดยการเพิ่ม Vertex เข้าไปภายใน Edge นั้น) ให้ทำการ Cut ตัว Model ดังรูป



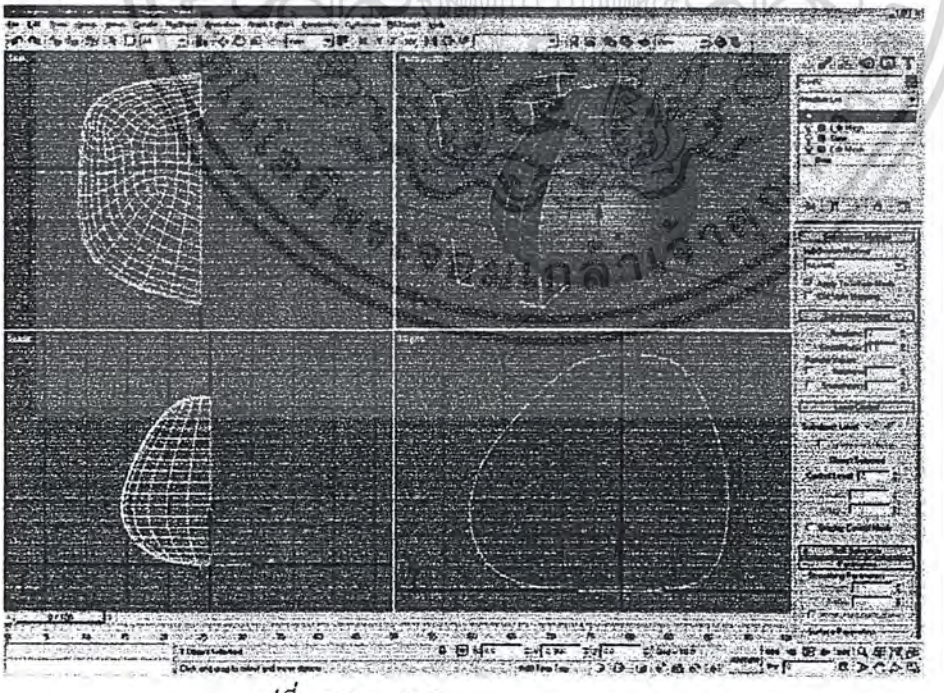
รูปที่2-40 Low Polygon

หลังจากการ Cut จะเห็นว่ามี Edge ใหม่เกิดขึ้นและมี Vertices เกิดเพิ่มขึ้นมาด้วย เราจะใช้ส่วนนี้สร้างเป็นแขนให้กับ Model ปรับ Vertices ดังรูปเพื่อเตรียมทำแขน



รูปที่2-41 Low Polygon

9.ต่อไปจะเป็นการทำ Mirror การทำ Mirror คือการทำให้ Model ที่เราสร้างมีความเป็นสมมาตร (Symmetric) ยกตัวอย่างเช่นการที่เราทำแขนซ้ายก่อนครั้งหนึ่ง แล้วทำแขนขวาก่อนครั้งหนึ่งอาจทำให้แขนทั้ง2ข้างไม่เท่ากันได้ เราจะแก้ปัญหานี้โดยใช้วิธี สะท้อน (Mirror) โดยให้เข้าสู่โหมด Vertex ใน Editable mesh เลือก Vertex ทางฝั่งซ้ายทั้งหมดของ Model หลังจากนั้นให้ลบ Vertex ทั้งหมดนั้นออกไปเหลือเพียง Vertex ทางฝั่งขวาเท่านั้น ต่อไปให้ออกจาก Editable mesh แล้ว Select ตัว Model ที่เหลือ กดปุ่ม Mirror เลือก Instance จะปรากฏ model อีกส่วนขึ้นมา Instance หมายความว่าเป็น Model ชั่วคราวเพื่อให้เราเห็นการเปลี่ยนแปลง การกระทำใดๆที่ทำกับ Model ฝั่งซ้ายจะเกิดการเปลี่ยนแปลงกับ Instance ฝั่งขวาด้วย

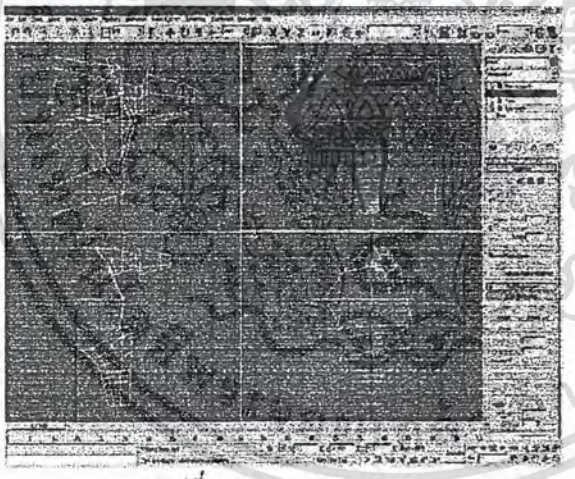


รูปที่2-42 Low Polygon

10.กลับมาทำแขนของ Model โดยการ Select Polygon 5 เหลี่ยม (โหมด Polygon) จากนั้น Extrude และ Bevel มันออกมา ทำการปรับขนาดและรูปร่างให้กลายเป็นแขนสำหรับ Model

รูปที่2-43 Low Polygon

11.Model ของเราเริ่มจะเป็นรูปเป็นร่างขึ้นมาแล้ว ต่อไปเป็นการเพิ่มขาเข้าไปให้กับ Model โดยการ Extrude และ Bevel เช่นเดียวกับที่ทำที่แขน ควรจะ Extrude ตรงข้อต่อ เช่น เข่า หรือ ข้อศอกไว้เป็นชั้นๆ (Layer) เผื่อเอาไว้เวลาทำ Animation ด้วย



รูปที่2-44 Low Polygon

12.Extrude ส่วนคอของ Model ออกมาด้วยการทำเช่นเดียวกับที่แขนและที่ขา ให้ทำการลบ Polygon ด้านในของคอที่ Extrude ออกมาด้วย ไม่อย่างนั้นหลังจากรวมตัว Mirror เข้าไปแล้วทำการ Attach จะเกิดข้อผิดพลาดกับ Model ได้

รูปที่2-45 Low Polygon

13.สร้างมือและเท้าให้กับ Model ด้วยการทำ Extrude เช่นกัน หลังจากนั้นให้ออกจาก Modifier แล้วลบ Instance ซ้ายมือทิ้งไป Select ตัว Model ขวามืออีกครั้ง แล้วกดที่ Mirror ตรงนี้ให้เลือก Copy แล้ว OK
เลื่อนซีก Copy กับ Model ดั้งเดิมให้เข้าใกล้กันที่สุดและต่อกันพอดี ปรับส่วนคอให้ชิดติดกัน ตรวจรอยต่อให้ชิดกันพอดี

รูปที่2-46 Low Polygon

14.หลังจากสร้างทุกชิ้นส่วนครบแล้วให้ Select ชิ้นส่วนด้านขวามือ ไปที่ Editable mesh ที่ Geometry Roll out เลือก Attach แล้วกดที่ชิ้น Copy ซ้ายมือ ตอนนี้ Model ของเราจะกลายเป็นชิ้นเดียวกันแล้ว ขั้นสุดท้ายคือทำการรวม Vertex ที่อยู่ใกล้กันให้เป็น Vertex เดียวกัน จุดประสงค์เพื่อทำให้ Model เรียบขึ้นไม่มี polygon ส่วนเกินที่ไม่ต้องการ ให้ Select Vertex กึ่งกลางตามแนวขวางของ Modelทั้งหมด เพื่อทำการ Weld Vertex ของชิ้น Copy และชิ้นดั้งเดิม แล้วไปที่ Edit geometry แล้วไปที่ weld target ปรับค่า selected ไปที่ 0.2 (ค่าโดยทั่วไปของการทำ weld vertex) หลังจากการ weld แล้วก็เสร็จการทำ Model ให้ตรวจสอบ Model ทั้งชิ้นปรับแก้ตามต้องการ โดยใช้ editable mesh เมื่อเสร็จเรียบร้อยทำการ ลบ Stack Modifier ทิ้งเพื่อคืน memory แล้ว save ไฟล์ไว้ทำชิ้นส่วนต่อไป

Make the Head

Low Polygon Make the Human's Head

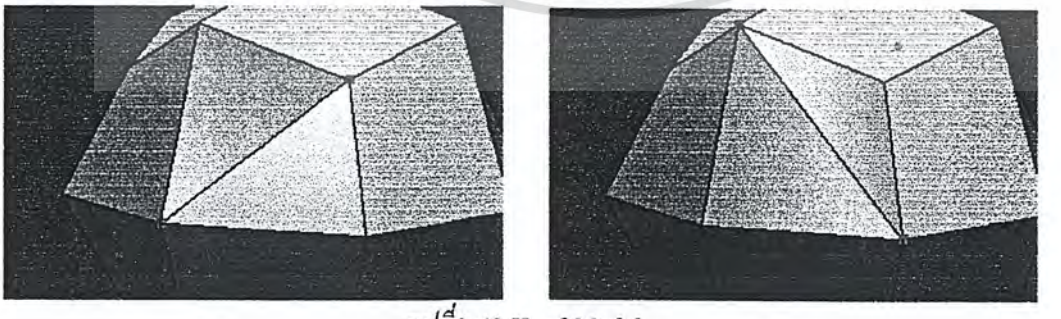
การสร้างหัวของมนุษย์จะมีรายละเอียดมากกว่าการสร้างร่างกายแขนขา ซึ่งต้องใช้ความชำนาญในการสร้างแล้วให้ทำ Texture Mapping ได้อย่างพอดี ในบทนี้จะไม่พูดถึงพื้นฐานการใช้โปรแกรม MAX และฟังก์ชั่นการทำงาน แต่จะพูดถึงเพียงแนวการสร้าง Model ศรีษะเท่านั้น

1.เริ่มด้วยการสร้าง Box ลูกบาศก์ขึ้นมาเหมือนกับการสร้างร่างกาย ลูกบาศก์จะมี 6 หน้า ในการทำให้ลูกบาศก์นี้เป็นรูปสามเหลี่ยมจะต้องแบ่งครึ่ง ตามเส้นกึ่งกลาง edge แต่ละด้าน โดยการแบ่ง Section ของลูกบาศก์ออกตามรูป

รูปที่2-47 Head Model

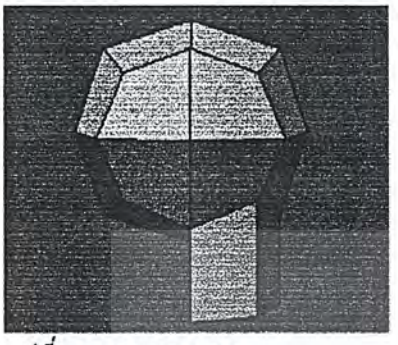
2.ใช้โหมด Vertices แล้ว Select จุดยอด Transform ให้ลูกบาศก์กลายเป็นรูปเหลี่ยม

3.ขณะนี้ Model ไม่ได้เป็นผิวเรียบ (Planer) อีกต่อไปแล้ว เพราะได้กลายเป็น Polygon ที่มี สามเหลี่ยม 48 ชิ้นประกอบกันเข้า ให้เปลี่ยนจาก Mode ของ Vertices เป็น Mode ของ Edge แล้วไปที่ Edit ทำการ Select all จะเห็นว่า Edge ทั้งหมดถูกทำให้ปรากฏขึ้น เส้นประหมายถึง Hide Edge เป็น Edge ที่ซ่อนไว้ไม่ให้เห็น
ปรับ Edge ตามรูปเพื่อสร้างด้านหน้าของศรีษะ



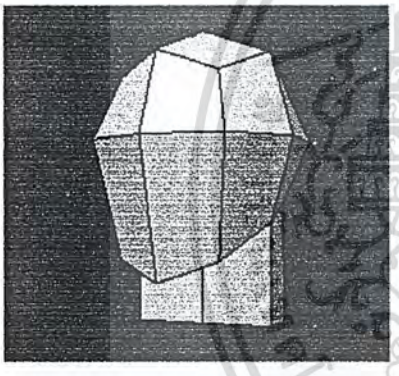
รูปที่2-48 Head Model

4.ต่อไปให้ Extrude ส่วนด้านล่างของ Polygon ลงมาตามรูปเพื่อสร้างเป็นคอ คุณอาจจะลบมันออกเมื่อเสร็จสิ้น หรือจะรวมมันเข้ากับส่วนของร่างกายก็ได้

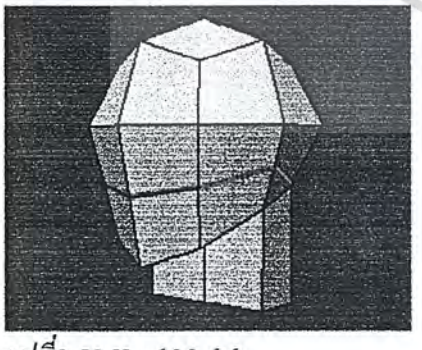


รูปที่2-49 Head Model

5.ดึงขอบล่างของ Model เพื่อสร้างเป็นคางและกรามตามตัวอย่าง

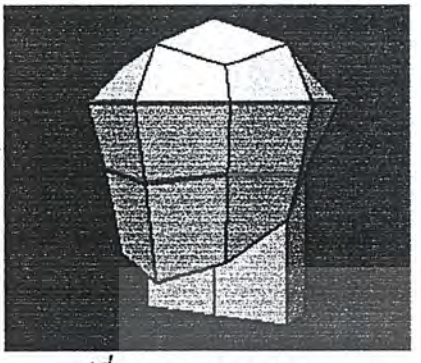


6.ส่วนที่สำคัญสำหรับลักษณะของศีรษะก็คือจมูก เราจะใช้พื้นที่ด้านบนเป็นหน้าผากและทำการ Divide Edge ใหม่เพื่อสร้างแกนนอนเส้นใหม่ซึ่งจะใช้เป็นจุดอ้างอิงสร้างจมูกขึ้น



รูปที่2-50 Head Model

7.ปรับ Edge ที่เป็นเส้นอ้างอิงบริเวณหน้าผาก และจมูกขึ้นไปยังบริเวณตำแหน่งที่ถูกต้อง

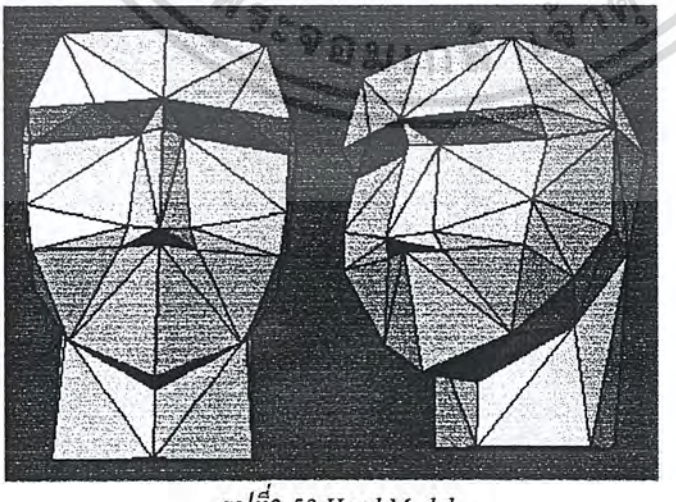


รูปที่2-51 Head Model

8.ให้ทำการ Select all ให้ปรากฏเส้น Edges ทั้งหมดอีกครั้งปรับตำแหน่งและรูปร่างของ Edges และ Vertices (ทำการพลิกด้าน Edge ได้ด้วยการ Turn Edge)

รูปที่2-52 Head Model

9.ขณะนี้เรามีสามเหลี่ยมด้วยกันรวม 66 ชิ้น แต่นี่เป็นเพียงการเริ่มต้นเท่านั้น ต่อไปให้ทำการปรับ Edges, Vertices ตามภาพ ขั้นตอนเหล่านี้ต้องอาศัยความชำนาญ ในการที่จะเลือกว่าจะทำการปรับแต่งและ Transform ตัววัตถุขนาดไหน ให้เหมาะสมกับ Model ที่เราได้ออกแบบไว้



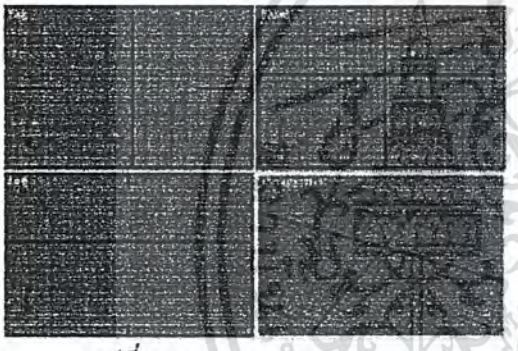
รูปที่2-53 Head Model

10.รูปร่างเมื่อสำเร็จ วิธีการสร้าง Low Polygon Head Modeling นี้เป็นเพียงวิธีพื้นฐานที่พอจะใช้สำหรับสร้าง Model ที่ใช้ในเกมได้เท่านั้นยังมีการสร้าง Model ชั้นสูงอีกซึ่งใช้วิธี Nurbs Model สามารถศึกษาได้จากคู่มือการใช้ 3d Studio Max

Make the tree

ต่อไปจะพูดถึงเทคนิควิธีการสร้าง Model องก์ประกอบฉาก Model ประกอบฉากของเกมจำเป็นที่จะต้องเป็นชนิด Low-Polygon ด้วยเพื่อจะได้ไม่ดึงความเร็วของการทำงานของโปรแกรม ต่อไปเป็นเทคนิคการสร้าง Model ต้นไม้ ซึ่งเป็นเทคนิคที่ใช้โดยทั่วไปในเกมประเภทบุคคลที่1 หรือ3 สามารถพบเห็นได้ทั่วไป จึงได้นำมากล่าวถึงไว้ในที่นี้ด้วย

1.เริ่มด้วยการเปิด โปรแกรม 3D Studio MAX ขึ้นมา สร้าง Object โดยใช้ Plane 2 ชิ้นตามรูป ใช้ section ทั้ง height และ width เป็น1 ทั้งคู่



รูปที่2-54 Make the Tree

2.คุณสามารถสร้าง Plane หลายๆ Plane ได้เพื่อให้ต้นไม้มีความสมจริงมากยิ่งขึ้นแต่จำนวน Polygon ก็เพิ่มมากขึ้นด้วยซึ่งการใช้ขนาดไหนจะถือว่าพอดีก็ขึ้นอยู่กับการออกแบบเกมของเรา หลังจากสร้าง Plane เรียบร้อยแล้วให้ใช้ editable mesh ทำการ Attach เข้าด้วยกันกลายเป็น Plane เดียว

3.กด M เปิด Material Editor เลือกใช้ Material ชนิด Bitmap เลือก File ภาพรูปต้นไม้ที่ได้เตรียมไว้

5.ปรับค่า Map ดังนี้

5.1ที่ Shader basic parameter ให้เลือก 2 sided

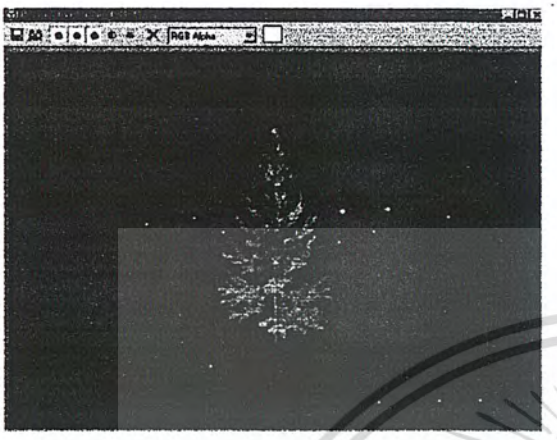
5.2ที่ Specular highlights ปรับ Glossiness เป็น 0

5.3ที่ Map เลือก Diffuse Color เลือก map file เป็น file ภาพต้นไม้ที่เตรียมไว้

และที่ Opacity bitmap เลือก map file เป็นภาพต้นไม้ที่เตรียมไว้รูปเดียวกัน

5.4ปรับค่า Mono Channel Output ที่ Opacity bitmap เป็น Alpha และที่ RGB Channel Output เลือกเป็น Alpha as Gray

6.ใส่ Texture ให้กับ Plane ที่สร้าง



รูปที่2-55 Make the Tree

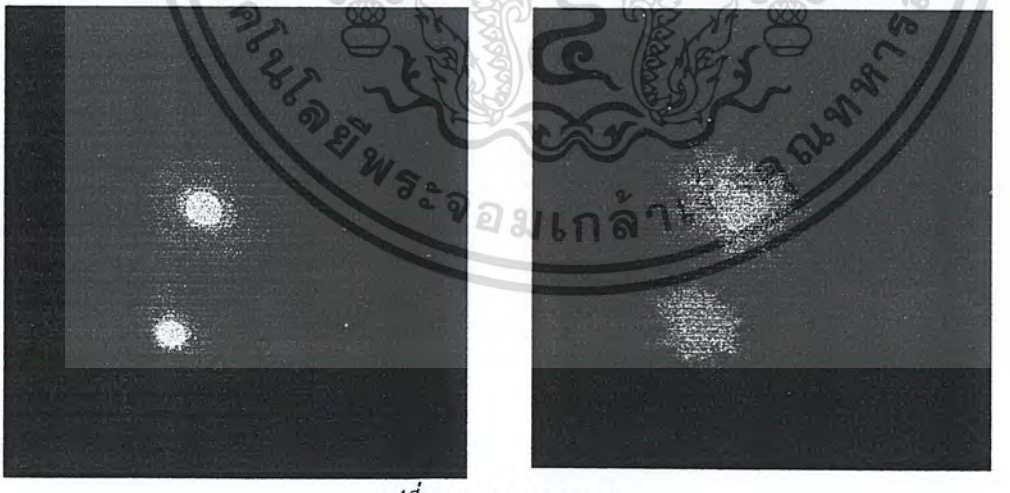
7.ทดสอบโดยการ Render เมื่อเป็นที่พอใจแล้วก็ Export ออกไปใช้ได้เลย

Make the Terrain

Terrain คือพื้นผิวในฉากเกม อาจเรียกว่าเป็น Landscape หรือภูมิประเทศในฉากเช่นเป็น ภูเขา เป็น เนิน หรือเป็น แอ่งน้ำ การสร้าง Terrain เป็นสิ่งสำคัญในเกมเพราะต้องใช้เป็นฉากเป็นพื้นผิวอ้างอิงที่สำคัญ ในส่วนนี้ เราต้องใช้โปรแกรม 2 ชนิดค้วยกัน คือ โปรแกรม 3D Studio และ โปรแกรม Paint เช่น โปรแกรม Adobe PhotoShop ก่อนอื่นต้องทำความเข้าใจก่อนว่า map ที่เราได้ทำการระบายจะทำหน้าที่เป็นตัวนำเสนอความสูงของพื้นที่ ณ จุดใดๆ โดยส่วนที่มืดจะหมายถึง พื้นที่ที่ต่ำ และส่วนที่สว่างจะแสดงถึงพื้นที่ที่สูงขึ้นมา

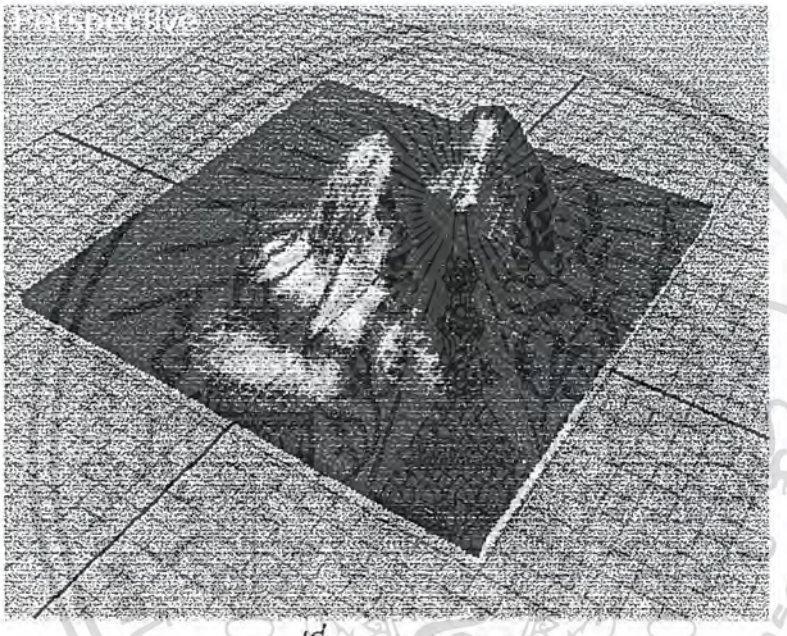
รูปที่ 2-56 Make Terrain

ดังนั้น ก่อนอื่นเราต้องสร้าง Map ขึ้นมาก่อน ในที่นี้ใช้โปรแกรม PhotoShop ในการสร้าง Map ขึ้นมาดังตัวอย่าง จุดที่สว่างที่สุดก็คือ จุดที่มีความสูงมากที่สุดไล่ลงมาจนถึงจุดที่ต่ำที่สุด เริ่มจากใช้ Brush ในการร่างกำหนดคร่าวๆก่อนแล้วใช้ filter Blur เข้าช่วย สามารถใช้ลูกเล่นต่างๆของ PhotoShop เข้าช่วยได้



รูปที่ 2-57 Make Terrain

ต่อไปก็ถึงการนำไปใช้ร่วมกับ Studio Max เปิดโปรแกรม Studio Max สร้าง Box ขึ้นมา กำหนดขนาดที่ต้องการ โดยใช้ Top Viewport และให้ height มีขนาดประมาณ 5 และมี Segments ประมาณ 50x50 segments ในหน่วย Length / Width จากนั้น ให้ใช้ Displace modifier ที่มีใน MAX ไปที่ tab modifier เปิด Displace modifier ขึ้นมา ปรับค่า Strength = 100 กดเลือกใช้ image map เลือก file ที่เราได้สร้างมาจาก PhotoShop จะเห็นว่า Box ที่เราสร้างขึ้น กลายสภาพไปตาม image map ที่เราสร้าง จากนั้น Export Terrain ที่ได้ เข้าสู่การสร้างเกมต่อไป



รูปที่ 2-58 Make Terrain

การสร้าง Terrain โดยใช้ MAX สามารถทำได้หลายวิธี อีกวิธี 1 ในการสร้างพื้นเป็นผิวขรุขระคล้ายผิวดินก็คือการใช้ Noise Modifier

1.สร้าง Box ขึ้นมาขนาดตามที่ต้องการนำไปใช้ (Height ขนาด 5 ที่ Top viewport) และกำหนด 50x50 segments

2.ใส่ Noise Modifier ให้กับ Box ปรับค่า Seed, Strength และ ปรับแกนตามความเหมาะสม

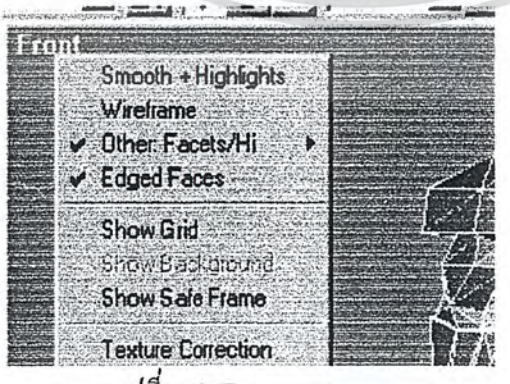
2.5 Texture Mapping

เมื่อสร้าง Model เสร็จแล้ว ขั้นต่อไปเป็นการทำให้ Model เสร็จอย่างสมบูรณ์เพื่อจะนำไปใช้ในเกมซึ่งก็คือการใส่ texture หรือเหมือนกับการระบายสีนั่นเอง

1.นำโมเดลที่สร้างเสร็จแล้วมาเปิดด้วย MAX ดังตัวอย่าง

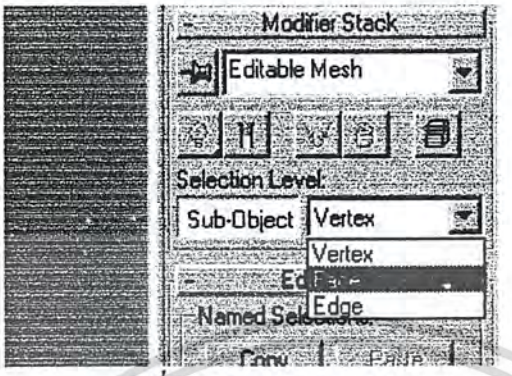
รูปที่2-59 Texture Mapping

2.คลิ๊กเมาส์ขวาที่ชื่อ viewport เปลี่ยนโหมดเป็น Other:Factes และ Edged Faces on จะทำให้สามารถมองเห็น polygon และ faces ได้ดีขึ้น



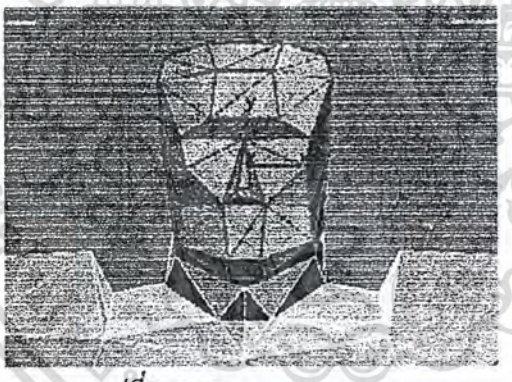
รูปที่2-60 Texture Mapping

3.เพิ่ม Editable mesh ใน modifier เลือก faces mode



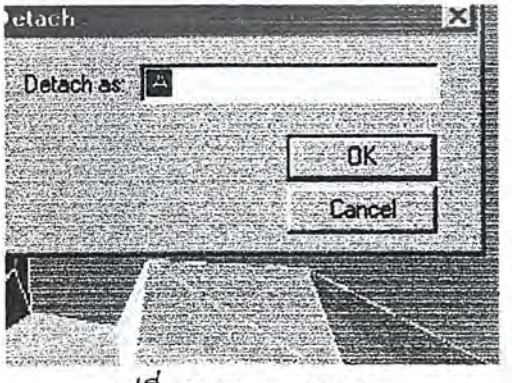
รูปที่2-61 Texture Mapping

4.จุดประสงค์ในการเลือก faces คือการเลือกกลุ่มของ faces ที่จะนำมา map ในลักษณะ planer หรือผิวเรียบโดย กด Ctrl ค้างไว้และเลือกกลุ่มของ faces ที่ต้องการ จากตัวอย่างเป็นการเลือกกลุ่ม faces ที่ด้านหน้าของส่วนหัวเพื่อที่เราจะนำไปจัดเป็นกลุ่มๆเพื่อทำ mapping ในขั้นตอนต่อไป ต้องแน่ใจว่าเลือก faces ได้ถูกต้องและไม่ได้เลือกส่วนหลังของหัวติดมาด้วยไม่เช่นนั้นส่วนที่ map ให้ด้านหน้าจะไปปรากฏในส่วนหลังด้วย



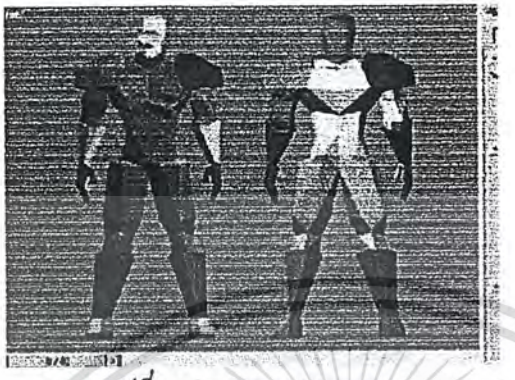
รูปที่2-62 Texture Mapping

5.ต่อไปให้เราทำการ 'DETACH' กลุ่มของ faces ที่เราเลือกไว้ให้ทำการใส่ชื่อที่เรียงลำดับไปเรื่อยๆได้ เช่น A, B, C, ... จะทำให้สามารถจัดการไปตามลำดับได้ง่ายขึ้นในขั้นตอนต่อไป



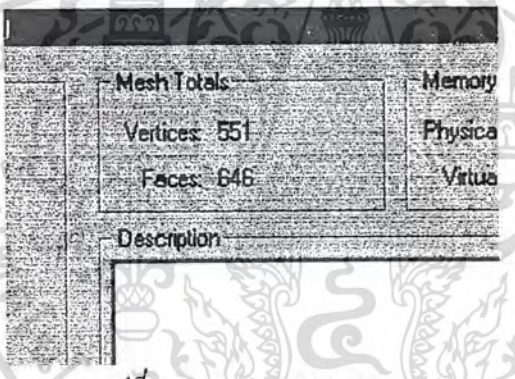
รูปที่2-63 Texture Mapping

6.ทำแบบเดียวกันกับส่วนที่เหลือ เลือกทั้งด้านหลัง ด้านข้าง แขน และขา ให้ครบ Model ที่ Detach แล้วควรได้คล้ายรูปตัวอย่าง



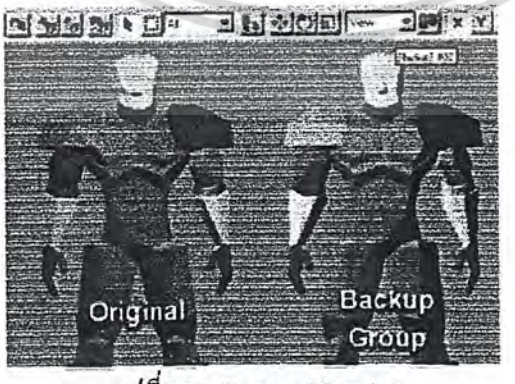
รูปที่2-65 Texture Mapping

7.เมื่อทำการ Detach หมดแล้วแต่ก็จะยังคงเหลือ Object อยู่ ให้ทำการลบ Object เปล่าๆนั้นทิ้งไป ให้ไปดูที่ค่า Summary info สังเกตว่ามีจำนวน vertices และ faces เท่าไร ให้จดเอาไว้



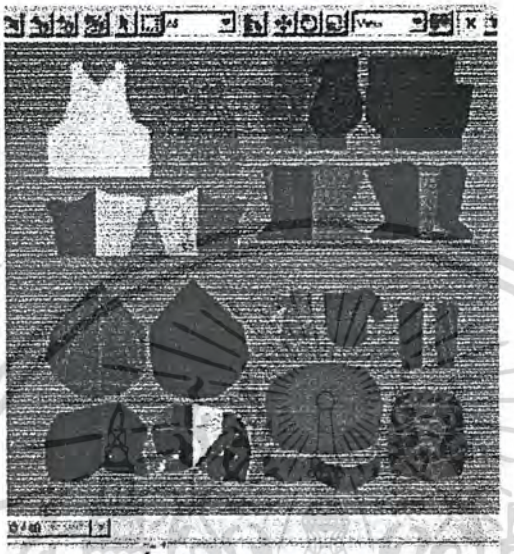
รูปที่2-66 Texture Mapping

8.ทำการ Select all และ Clone ตัว Object ออกมา ให้ชื่อ Group ว่า 'Backup' การทำเช่นนี้เป็นการเก็บตัวโคลนแยกไว้จากตัวแม่แบบ ถ้าไม่ทำการโคลนไว้ในขั้นตอนต่อๆไปจะไม่สามารถทำได้ จัดการ Hide ตัว Clone เอาไว้



รูปที่2-67 Texture Mapping

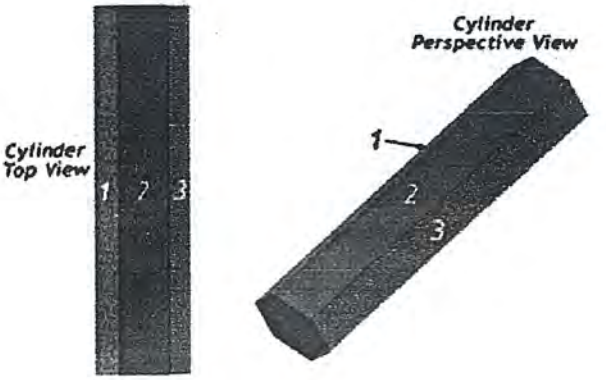
9.จากนี้ไปห้ามลบ Faces หรือ Vertices ใดๆทั้งสิ้น แต่ละชิ้นส่วนที่ได้ทำการ Detach เอาไว้ จัดการจับมันมาเรียงใหม่ไว้ภายในบริเวณสี่เหลี่ยม ให้แน่ใจว่า Object ทุกชิ้นได้หันให้เห็น Faces ของแต่ละชิ้นอย่างชัดเจน (เช่น ต้องทำการหมุนชิ้นส่วนด้านหลังให้หันออกมาเพื่อให้มองเห็นได้)

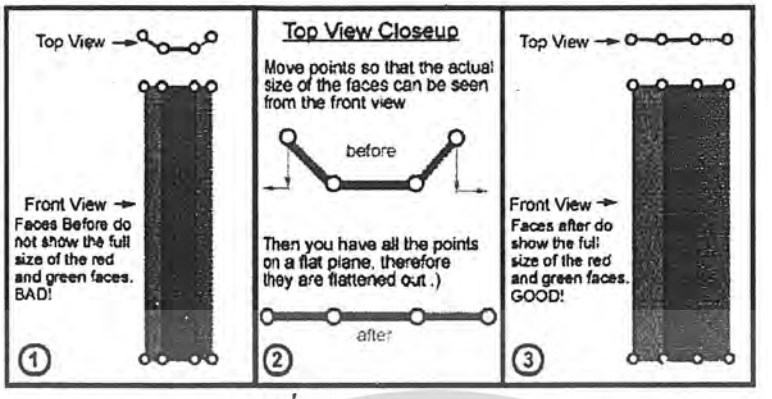


รูปที่2-68 Texture Mapping

10.ต่อไปทำการจัดวัตถุให้เรียบลงไปเพื่อให้ Texture ห่อวัตถุได้อย่างถูกต้องถูกเหลี่ยมและมุม การทำให้เรียบลงไปนั้นหมายความว่าให้กางตัวชิ้นส่วนออกจากวัตถุที่มีความหนาก็ต้องจัดมันให้เป็นแนวราบ การทำเช่นนี้ให้คุณแน่ใจว่าเมื่อเวลาคุณเคลื่อน Point แต่ละ Point ของชิ้นส่วนออกไปให้มั่นใจว่า มันจะยังอยู่ในขนาดเท่าเดิมก่อนการย้าย

10-1.การทำให้เรียบ (Flatten, Layout) มีส่วนเกี่ยวข้องกับการทำ Planer Map จากรูป ยกตัวอย่างว่าเราต้องการที่จะ Map วัตถุนี้ แต่มีปัญหาที่ว่าถ้าเรา Map ดังรูปเมื่อหันมุมดังรูปด้านขวาจะเกิดการผิดขนาดขึ้นเพราะเราไม่ได้ Map วัตถุไว้ในขนาดเดียวกัน วิธีแก้คือเราจะต้องทำให้การ Map วัตถุที่เห็นด้านหน้าตรงๆมีขนาดที่ถูกต้อง ซึ่งมันจะทำให้เมื่อหมุนวัตถุไปในด้านใดๆแล้วก็จะยังมีขนาดที่ถูกต้องเป็นผลให้การ Map ถูกต้องด้วย

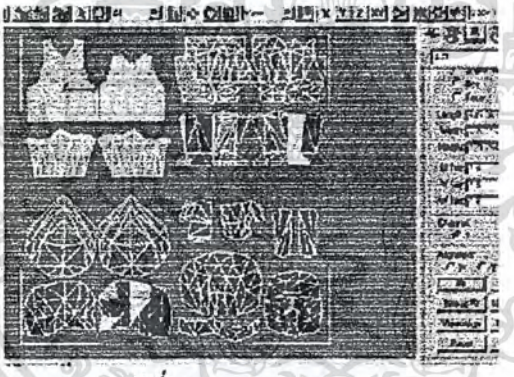




รูปที่2-60 Texture Mapping

11.เมื่อถึงขั้นนี้เราก็จะได้ Object แต่ละชิ้นถูก Layout ภายในพื้นที่สี่เหลี่ยม ต่อไปคุณจะต้อง Re-Attach มันกลับเป็นชิ้นเดียวกัน ให้ทำการ Attach แต่ละชิ้นเรียงตามลำดับจากที่เราได้ Detach เอาไว้ เช่น A ไป B และ B ไป C

12.เมื่อ Attach เรียบร้อยแล้ว ให้ใช้ UVW Map modifier กับ Object ให้เลือกใช้ 'Fit' คุณก็จะได้ 1 Bitmap ที่จะใช้สำหรับทำ Map Planer สำหรับทุกส่วน ปัญหาต่อไปคือจะทำยังไงให้แต่ละส่วนกลับมาเป็นตัว Model ของคุณ?



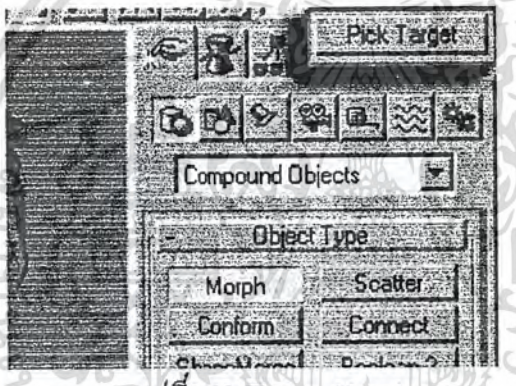
รูปที่2-61 Texture Mapping

13.ทำการ Unhide ตัว Clone จัดการ Ungroup ตัว Clone แล้ว Attach ชิ้นส่วนของตัว Clone เรียงตามลำดับเหมือนข้อที่11

รูปที่2-62Texture Mapping

14.ถึงขั้นนี้คุณควรจะมีเพียง Object 2 ชิ้นเท่านั้นคือ Model ที่ถูก Lay out แล้ว กับ Model ชิ้นที่เป็น Clone ขั้นตอนนี้เป็นการทำให้มั่นใจว่ากระบวนการ Map จะไม่ผิดพลาด ให้คุณทำการลบตัว Clone ออกไป และดูที่ Summary Info ตรวจสอบจำนวน Faces และ Points ทำการ Undo delete ให้ Model Clone กลับมาต่อไปลบ Model ที่ถูก Lay out ออกบ้าง ตรวจจำนวน Faces และ Points ของตัว Clone เปรียบดูว่าเท่ากับจำนวนของ Model ที่ถูก Lay out เมื่อกี้หรือไม่ จำนวนทั้งสองควรจะเท่ากันจึงจะทำให้ขั้นตอนนี้ถูกต้อง ถ้ามันไม่เท่าแสดงว่าคุณได้ทำผิดพลาดในการในการลบ Vertex หรือ Face ออกไป

15.เลือก Compound Object และเลือก Morph กดปุ่ม 'Pick Target' และกดที่ตัว Model Clone Model จะกลับมาอยู่ในรูปร่างเดิม Max จะทำการเก็บข้อมูลของ UVW map ไว้จาก Model ที่ถูก Lay out ไว้ ดังนั้นต่อไปคุณก็พร้อมที่จะทำการแต่ง Model ของคุณแล้ว

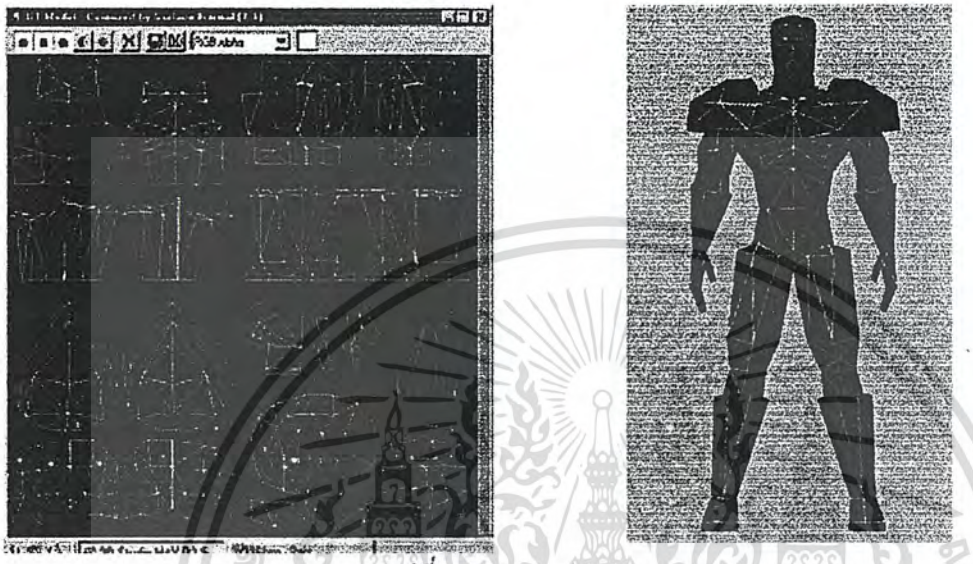


รูปที่2-63 Texture Mapping

16.ทำการ Selected ตัว Model และ Weld Vertex เข้าด้วยกัน การ Weld Vertex คือการทำให้ Vertex อยู่ใกล้กันรวมเป็น Vertex เดียวโดยเรากำหนดระยะห่างว่า Vertex ที่ห่างกันเท่าไรที่จะถูก Weld เข้าด้วยกัน ค่าโดยทั่วไปจะประมาณ 0.2 การ Weld Vertex จะทำให้ Model มีความเรียบลบรอยต่อระหว่าง Faces ที่เป็นขอบมุมออกไปได้

17.ขั้นตอนสุดท้ายคือการใช้ Textporter ให้ไปที่ Utilities Tab และเลือก Morph Option แล้วเลือก Textporter เลือกขนาดของภาพที่ต้องการ เป็นหน่วย pixels ควรจะเลือกขนาดใหญ่ก่อนเพื่อสะดวกในการลงรายละเอียดแล้วจึงลดขนาดมันในภายหลัง ประมาณ 700x700 กดปุ่ม Pick Object และกดที่ Model ของคุณ แล้วคุณควรจะได้ภาพพร้อมด้วยชิ้นส่วนทุกชิ้นที่ถูก Lay out แล้ว ทำการ Save เป็น Tiff และนำมันไปใช้เป็น Bitmap Texture ของคุณ สังเกตว่าจะเห็นเส้นต่างๆ รวมทั้ง mesh linc อย่างชัดเจน นำรูปที่ได้ไปตกแต่งด้วย PhotoShop หรือ Paint Program ใดๆ

Note: TextPorter เป็น Plugin ตัวหนึ่งสำหรับ MAX ทำการ Export ตัว Object ใน MAX ออกเป็น File ภาพใช้สำหรับในการทำ Texture Mapping



รูปที่2-64 Texture Mapping

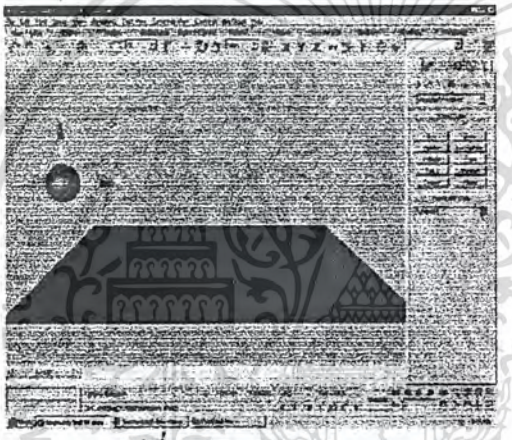
2.6 Animation

Basic KeyFrame

การทำ Animation นั้นจะใช้ MAX สร้างขึ้นซึ่ง MAX ก็มีการสนับสนุนการทำ Animation อยู่แล้วนั่นก็คือการทำ KeyFraming หมายถึงการ Render ภาพออกมาเป็นแต่ละ Keyframe ซึ่งมีการเคลื่อนที่ ขยับตำแหน่ง เปลี่ยนแปลงรูปร่าง หรือลักษณะอื่นใดเป็น 1 KeyFrame และเมื่อ render ออกมาเรียงตามลำดับกันออกมาแล้วก็จะสามารถมองเป็นภาพ Animation ทำให้วัตถุมีการเคลื่อนไหวได้

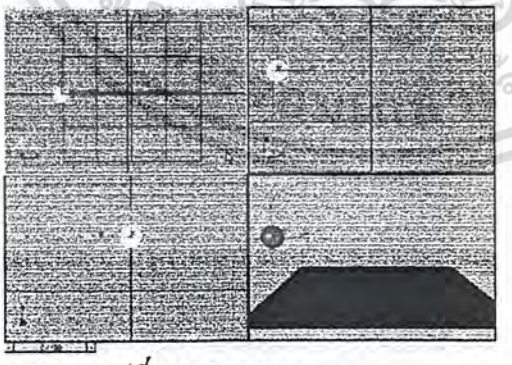
ซึ่งในบทนี้ก็จะมาพูดถึงวิธีการพื้นฐานของการใช้ KeyFrame เพื่อจะได้สามารถเข้าใจในบทต่อไป

1.เปิดโปรแกรม MAX สร้างลูกทรงกลมขึ้น 1ลูก (Sphere) ขนาดประมาณ 10 units ใน Top Viewport



รูปที่2-65 Basic Keyframe

2.สร้าง Plane ขึ้นมา 1 Plane ขนาดประมาณ 200x200 units เราจะใช้เป็นพื้นอ้างอิง



รูปที่2-66 Basic Keyframe

3.เคลื่อนทรงกลมขึ้นไปเหนือ Plane ซ้ายมือ

4.สังเกตขอบล่างตรงส่วนที่แสดง Frame เป็นบริเวณที่จะแสดงจำนวน Frame และลำดับ Frame ขณะนั้น ซึ่งตอนนี้จะมีค่าอยู่ที่ 0 เพราะเรายังไม่ได้ทำ Animation ใดๆ คลิ๊กเมาส์ขวาที่กรอบนี้จะมี Dialog Box ขึ้นมาซึ่งจะเขียนไว้ว่า 'Source Time:0, Destination Time:0' กด OK เพื่อสร้าง KeyFrame สำหรับการ หมุน การ ย่อขยายขนาด และการย้ายตำแหน่งที่ Frame0 สังเกตที่ วงกลมรูปไข่สีเทาที่ดานซ้ายมือ นี่คือ Key ที่คุณต้องสร้างขึ้นจะปรากฏอยู่ใน Track Bar

5.กดปุ่ม Animate ให้กลายเป็นสีแดง ตรง 'current time field' ให้ใส่ค่า 50 Time Slider จะเลื่อนไปที่ Frame 50 เลื่อนทรงกลมให้อยู่เหนือกึ่งกลางของ Plane โดยใช้ Transform Gizmo แกน X

รูปที่2-67 Basic Keyframe

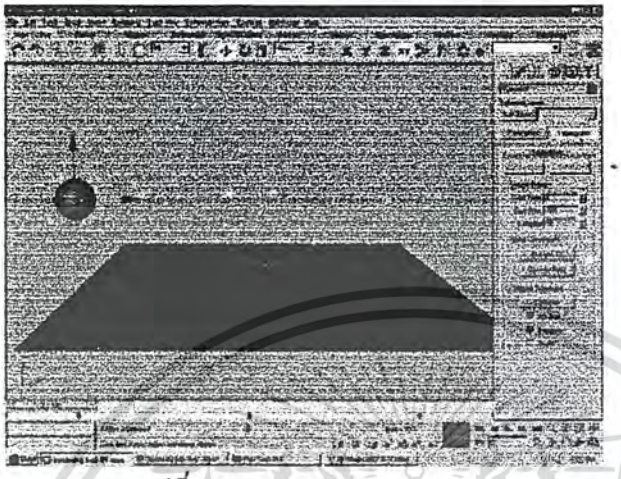
6.ในขณะที่ทรงกลมถูก 'selected' อยู่ให้กดที่เครื่องหมาย Align ตัวชี้จะเปลี่ยนเป็นรูปของ Align กดที่ Plane จะมี Box ข้อความขึ้นมา ตรง 'Under Align Position(World)' ให้เลือก Z, ตรง 'Current Object' ให้เลือก Minimum แล้วกด OK

รูปที่2-68 Basic Keyframe

7.ย้อน Animation ไปที่ Frame 0 และทำการ Play ทรงกลมที่ได้จะหล่นลงมาจากจุดเริ่มต้นและตกลงมาหยุดที่ Frame 50

8.เลื่อนไปที่ Frame 100 และย้ายลูกทรงกลมไปที่จุดบนขวา Key Frame จะถูกสร้างขึ้นอย่างอัตโนมัติที่ Frame 100 ให้ลองเล่น Animation ดูจะเห็นว่าลอยเป็นมุมขึ้นไป ถ้าต้องการให้มันโค้งจะต้องเพิ่ม KeyFrame เข้าไป

9.ไปที่ Frame 15 เคลื่อนทรงกลมไปตามแกน Z ให้สูงท่ากับที่ Frame0 ทำเช่นนี้ที่ Frame 85 ด้วย สังเกตว่า Animation จะโค้งขึ้นมานิดหนึ่ง

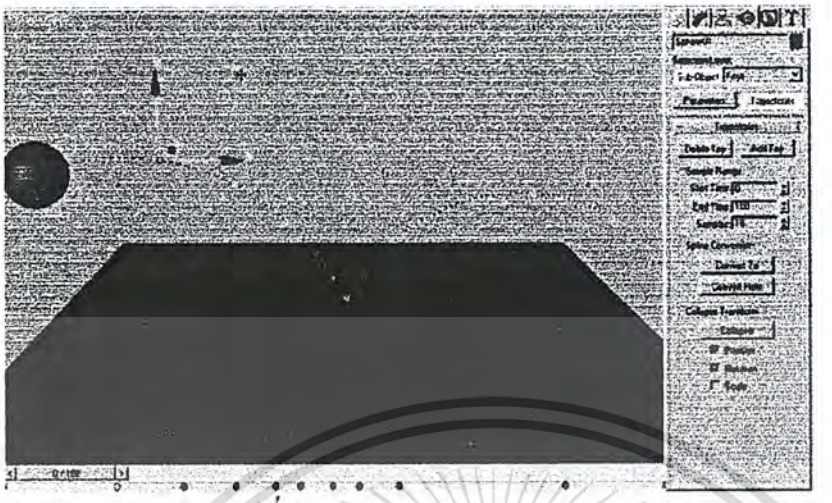


รูปที่2-69 Basic Keyframe

10.ให้กดเลือกทรงกลมไว้แล้วเปิด Motion panel เลือก trajectories MAX จะแสดงเส้นทางการเดินของลูกบอลให้เห็น

11.ที่ Motion panel เลือก sub-object keys ที่ Trajectories ให้ทำการ Add key เลื่อนตำแหน่งของเคอร์เซอร์ไปอยู่เหนือทรงกลมสีน้ำเงิน เคอร์เซอร์เปลี่ยนเป็นรูปเครื่องหมายบวก เพิ่ม Key เข้าไปใกล้จุดโค้งอีกทั้ง 2 ด้าน

12.ปิด Add Key ให้เรียบร้อยแล้วทำการปรับ KeyFrame ใหม่โดยใช้คำสั่ง Move เพื่อให้แน่ใจว่าทรงกลมของคุณจะวิ่งในแนวเดียวกันให้ระวังที่จะเลื่อนทรงกลมไปในแกน XZ เท่านั้น เลือกมุมแดงและน้ำเงินของ Transform Gizmo แล้วย้ายตำแหน่งไปในจุดที่คุณต้องการ



รูปที่2-70 Basic Keyframe

13.เล่น Animation เพื่อตรวจสอบว่าลูกบอลได้โค้งอย่างที่คุณต้องการ Keys สามารถเคลื่อนย้ายได้โดยใช้ Transform Gizmo ในพื้นที่ space หรือย้ายโดยลากเครื่องหมายรูปไข่ใน Track Bar ปรับตำแหน่ง หรือ เคลื่อนย้าย Frame ต่างๆตามแต่ที่คุณต้องการ จุดสีขาวที่ปรากฏบนจอแสดงความต่างและความสัมพันธ์ระหว่าง Frames กับ Keys ถ้าจุดมีน้อยวัตถุก็จะเคลื่อนที่ไปได้รวดเร็ว

14.นี่เป็นเพียงพื้นฐานของการใช้ KeyFrame เพื่อนำไปใช้ทำ Animation ให้กับ Model เท่านั้น ตัวโปรแกรม MAX ยังมีความสามารถมากมายที่จะช่วยให้คุณทำ Animation ต่างๆที่ต้องการ

Character Studio

การทำ Model ในเกมนั้นจะขาดความน่าสนใจและความสวยงามไปอย่างยิ่ง ถ้า Model ไม่มีการเคลื่อนไหว สิ่งสำคัญที่สุดในเกมก็คือความน่าสนใจในตัวเกม ดังนั้น Model ควรจะต้องมีการเคลื่อนไหว หรือ Animation ที่สวยงาม เช่นการเดินของมนุษย์ ล้อรถ หมุน หรือการก้าวเดินของสัตว์4เท้า เป็นเรื่องยากที่ คนทั่วๆไปที่ไม่มีความรู้ด้านกราฟฟิก 3 มิติและการคำนวณชั้นสูง จะสามารถสร้างการเคลื่อนไหวในลักษณะเสมือนจริงขึ้นมาได้ เนื่องจากจะต้องทำการคำนวณชั้นสูงทั้งทางด้าน Forward Kinematics และ Inverse Kinematics ถึงแม้ทางผู้ผลิต MAX จะบอกว่าเป็นเครื่องมือที่ใช้ได้ง่ายก็ตามที ทางผู้ผลิตจึงได้ทำการออก Tool อีกตัวที่ใช้สร้าง Animation ได้ง่ายมากขึ้น นั่นคือ Character Studio ซึ่งรองรับการทำ IK แบบอัตโนมัติไว้ทำให้บุคคลทั่วไปสามารถเรียนรู้และใช้งานได้อย่างรวดเร็วง่ายดาย ในบทนี้เราจะกล่าวถึงคอนเซปต์และวิธีการใช้โปรแกรมสร้าง Animation ที่เราต้องการ โดยใช้ Character Studio

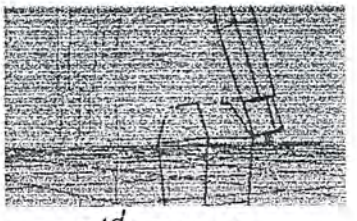
Character Studio มีการทำงานหลักๆ 3 ส่วนคือ Biped, Physique และ Crowd ในที่นี้เราจะพูดถึงเพียง Biped และ Physique เท่านั้นส่วน Crowd นั้นเป็นการทำ Group ตัว Object เข้าด้วยกันซึ่งไม่ได้ใช้ในการทำโครงงาน รายละเอียดสามารถหาได้ในคู่มือการใช้ Character Studio

Biped

Biped เป็นการสร้าง Skeletal พื้นฐานอย่างง่ายๆทั่วไป (Skeletal หมายถึง โครงกระดูกที่เปรียบเหมือนแกนหลักของ Model ที่ใช้ในการทำการเคลื่อนไหว หรือ Animation นั่นเอง) โดยปกติเราสามารถสร้าง Skeletal ขึ้นมาได้เอง แต่ต้องใช้ความรู้ขั้นสูงทางด้านการคำนวณคณิตศาสตร์และความรู้ด้านกราฟฟิกดีไซน์ เป็นการสะดวกที่เราจะใช้ Biped ในการสร้าง Animation อย่างง่ายๆสำหรับเกม เพราะ Model ที่ใช้ในเกมต้องการข้อมูลที่น้อย ไม่เหมือนการสร้างภาพยนต์ที่ไม่มีข้อจำกัดในการ Render ภาพ

Biped ถูกออกแบบมาให้มีข้อต่อตามลักษณะข้อต่อของมนุษย์ โดยปกติ biped จะถูกจัดอยู่ในลักษณะของ Human skeleton และทำ IK โดยอัตโนมัติไว้อยู่แล้ว หมายความว่าเมื่อคุณเคลื่อนไหวมือหรือเท้า ข้อเข่า หรือ ข้อศอกจะถูกเคลื่อนที่ไปโดยอัตโนมัติตามการเคลื่อนที่จาก มือและเท้า สามารถใช้ Tool ของ MAX ในการทำการ Transform เคลื่อนย้ายตำแหน่งต่างๆของ biped

Biped Center of Mass เป็นจุดศูนย์กลางของ Biped ใช้สำหรับอ้างอิงตำแหน่ง เคลื่อนย้าย Biped หรือทำ Transform ใดๆ รวมทั้งใช้ในการกำหนด Biped ให้กับตัว Model



รูปที่2-71 Biped

Keyframing the Biped

การทำ Animation ให้กับ Biped มี2วิธีด้วยกัน คือ Footsteps Method และ Freeform Method สำหรับ Footsteps Method นั้นเป็นคล้ายๆกับเครื่องมืออัตโนมัติที่มีมากับ Character Studio ที่ใช้สำหรับการเคลื่อนที่โดยทั่วๆไป ซึ่งเราจะไม่ใช้ในที่นี้ จึงไม่ขอกล่าวถึง Freeform Method นั้นเป็นการทำ Animation โดยการทำ Transform ให้กับ biped ผสมกับเทคนิคการใช้ Keyframing โดยผู้สร้างต้องกำหนดการเคลื่อนไหวหรือจำนวนเฟรมหรือท่าทางเอง จึงเหมาะกับการทำเกมมากกว่า เพราะสามารถทำการเคลื่อนไหวได้อย่างอิสระ Biped สามารถใช้ได้กับตัว Model ที่มี2ขาใดๆ รวมไปถึง Model 4 ขาด้วย

Working With Biped

1.เปิด Model ที่คุณสร้างขึ้นมาแล้วทำการ Freeze ตัว Model ไว้ก่อน

2.หลังจากทำการติดตั้งโปรแกรม Character Studio แล้ว ให้ไปที่รูป เฟือง2ตัวบนเมนู เลือก Systems

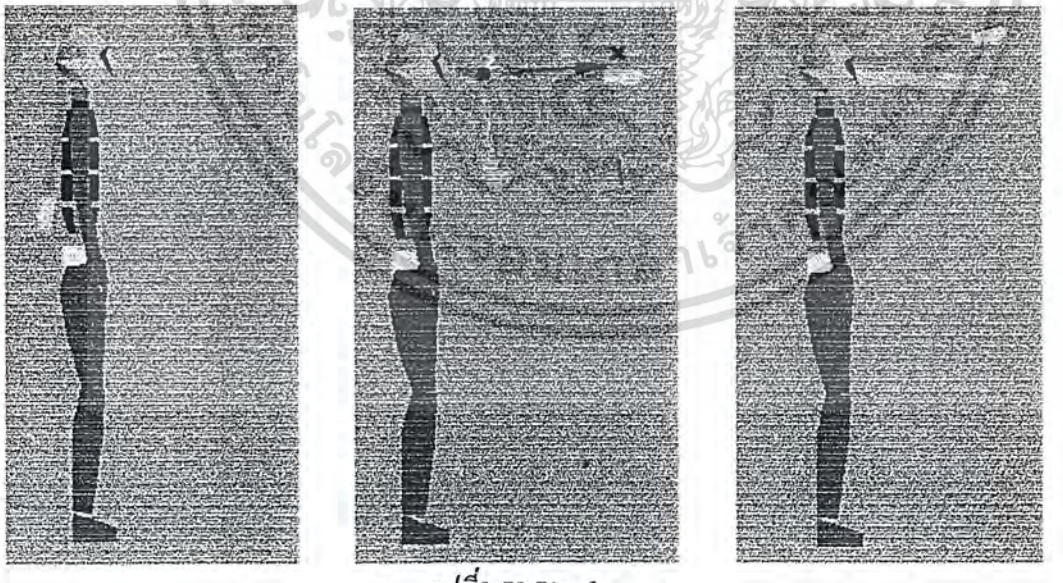
3.กด Biped ที่เมนูให้เลือก จะปรากฏแถบคุณลักษณะของ Biped ขึ้นมา

4.ไปที่ Viewport เลื่อนตัวชี้ไปที่ Viewport กดและลากขึ้นมาจะสังเกตเห็น Box เกิดขึ้นตรงจุดที่ลาก

5.ให้ลาก Box ขึ้นมาจน Box มีความสูงเท่ากับตัว Model

6.ไปที่ Motion Panel รูปล้อ แล้วเลือกเป็น Figure Mode ปรับ Biped ให้มีคุณสมบัติตามที่ต้องการเช่นจำนวนข้อต่อ ความสูง หาง หรือ อื่นๆ

7.สามารถนำส่วนหาง (Tail) หรือหางกระต่าย (Pony Tail) มาใช้เป็นชิ้นส่วนอื่นๆ ได้



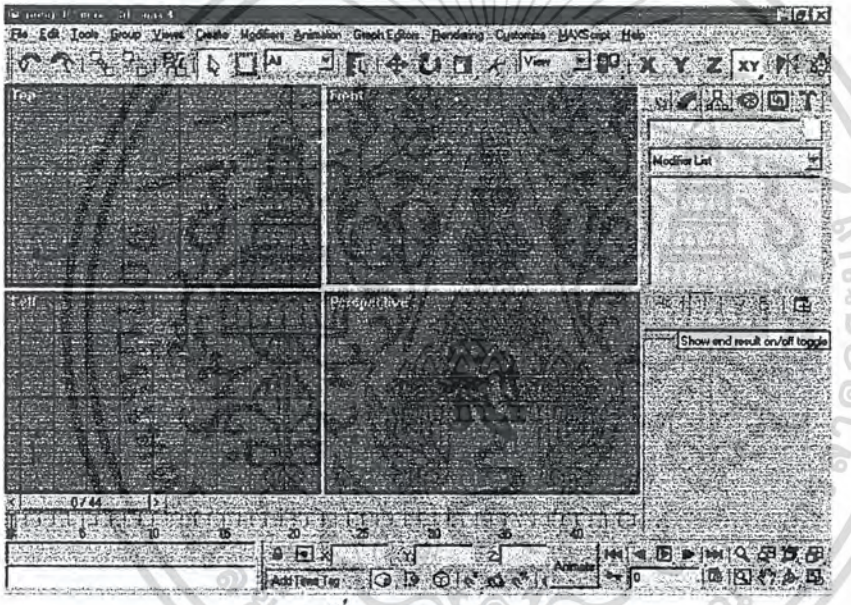
รูปที่2-72 Biped

Physique

หมายถึงการทำให้ Biped กับ mesh Model รวมเข้าด้วยกัน หลังจากได้ทำ Physique แล้วตัว mesh Model จะเคลื่อนไหวไปตาม Biped ต่อไปเราจะมาดูถึงการกำหนด Biped เข้าไปให้กับ mesh Model

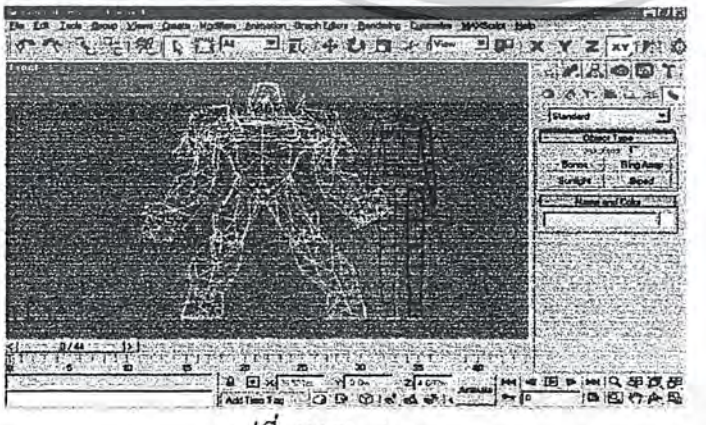
Physique เป็น Modifier ที่ใช้กับ Mesh Model ต่างๆ สำหรับการใช้ Skeleton กับ mesh Model ให้เคลื่อนที่ไปด้วยกันได้ Physique จะมีผลกับ mesh ต่อเมื่อได้ทำการ Attach Node แล้ว เมื่อทำการ Attach การทำงานของ Physique จะทำการ Link ส่วนต่างๆของ mesh กับ biped ไปตามลำดับขั้น

1.เปิด Model ที่คุณต้องการทำ Animation ขึ้นมาแล้ว Freeze ไว้ก่อน



รูปที่2-73 Physique

2.สร้าง Biped ขึ้นมาให้มีความสูงเท่ากับ Model แล้วตั้งค่าโครงสร้างเริ่มต้นของ Biped ให้เรียบร้อย



รูปที่2-74 Physique

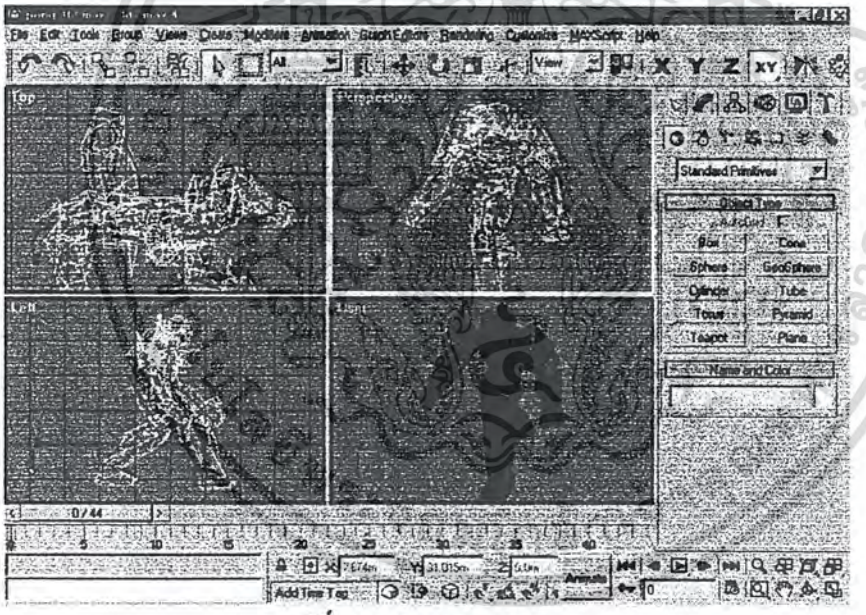
3.เข้าสู่ Figure Mode

4.เลื่อน Center of Mass ไว้ที่จุด Pelvis ของ Model (จุด Pelvis คือจุดที่ใช้เป็นจุดอ้างอิงของ Model)

5.ปรับ Biped ให้เข้ากับแขนขา ของ Model โดย Select ส่วนที่ต้องการ แล้วใช้ Non-Uniform Scale ปรับแต่งขนาด Biped ให้เท่ากับ Model

6.ปรับ Viewport เป็นด้านข้างทำการ ปรับ Biped อีกให้เข้ากับ Model อาจใช้โหมด Symmetric ในการ ทำ Transform ชิ้นส่วนที่เป็นคู่เช่นแขน หรือขา ซึ่งจะทำให้ปรับได้อย่างเท่ากัน 2ข้าง

7.ปรับดู Model ในหลายๆมุมมอง ให้แน่ใจว่า Biped ทุกส่วนได้ Fit เข้ากับ mesh Model



รูปที่2-75 Physique

เมื่อได้มั่นใจว่าปรับขนาดของ Biped จนพอเหมาะพอดีกับ mesh Model แล้ว ต่อไปเป็นการรวม Biped กับ Physique ให้เป็นชิ้นเดียวกัน

1.อันดับแรกให้ปิด Figure Mode ก่อน

2.Unfreeze Model ของเรา เลือกที่ COM ของ Biped จัดวางตำแหน่งให้ตรงกับ Pelvis Area

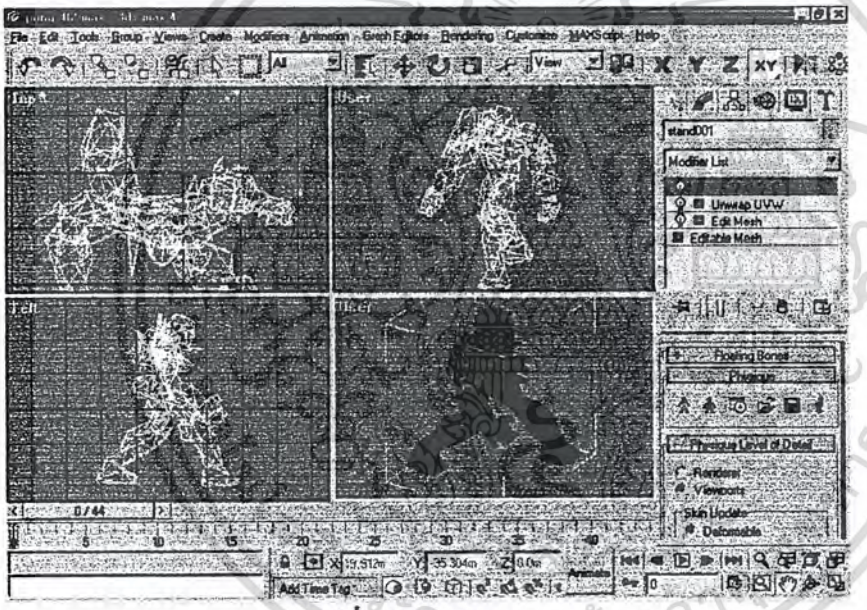
3.ให้ใส่ Physique Modifier ให้กับ Model เลือก Attach to Node

4.จะมีตัวเลือกปรากฏขึ้นมาให้กดที่ Pelvis object (Box กึ่งกลางของ Biped)

5.จะมีตัวเลือกปรากฏขึ้นมา ให้ตรวจสอบให้แน่ใจว่าตัวเลือกต่อไปนี้ถูกต้อง

- Deformable
- N Links
- Create Envelopes
- Object Bounding Box

6.กดปุ่ม Initialize ที่ด้านล่าง สักครู่จะปรากฏ เส้น Physique Link สีส้มขึ้นมา แต่ละ Link จะดูด Vertices ใกล้เคียงเอาไว้กับแต่ละส่วนของ Biped Skeleton



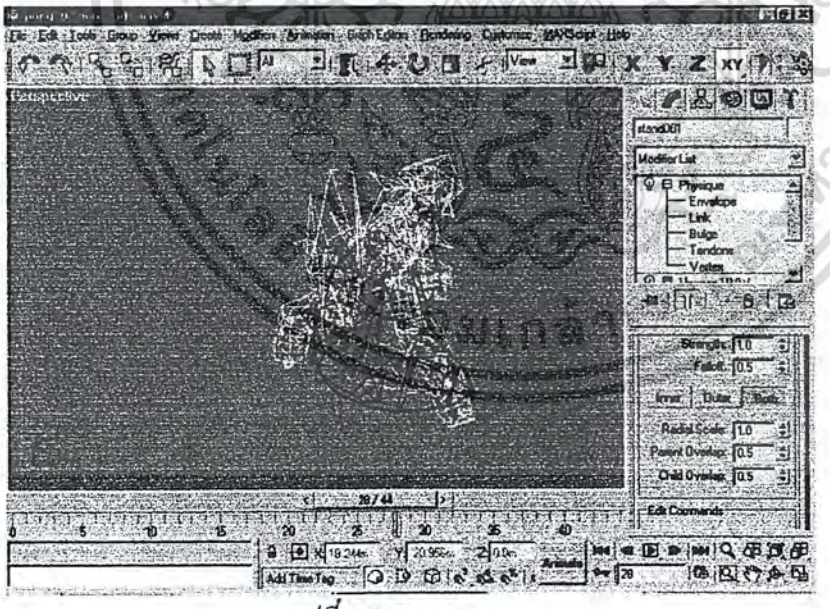
รูปที่2-76 Physique

Envelope

ต่อไปเป็นการปรับแต่งขนาด และค่า Overlap เพื่อที่จะทำให้ mesh ถูก Link อย่างถูกต้องกับ Biped ในขณะทำการปรับค่า Envelope ให้เลื่อน Time Slider และหมุน View port ไปทั่วๆเพื่อสังเกตว่า Model ที่เคลื่อนไหวได้เคลื่อนไหวอย่างถูกต้องทุก mesh ทุกจุดใดๆ

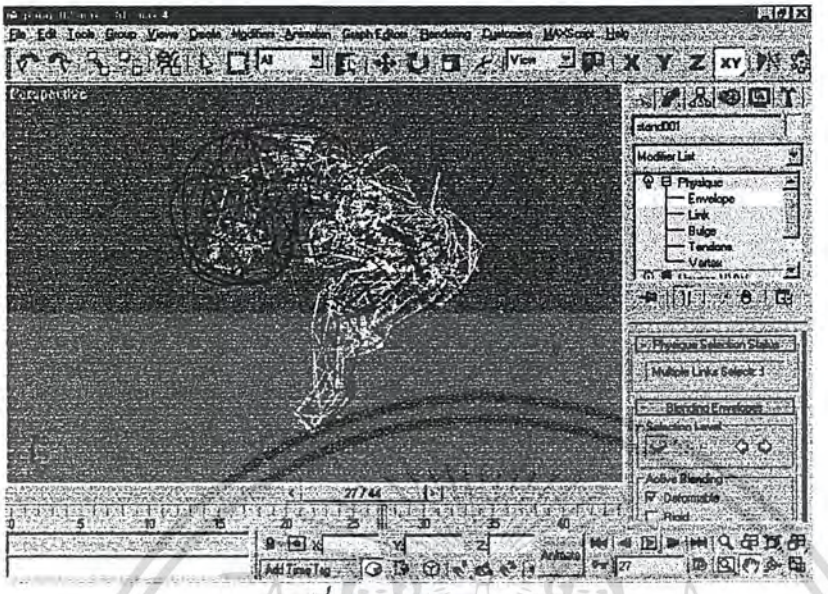
หลังจากได้ Attach ตัว mesh กับ biped เข้าด้วยกันและ ทำ Animation แล้ว mesh ควรจะต้องเคลื่อนไปตามที่เรากำหนดให้กับ biped ไว้ แต่จะสังเกตว่าจะมีบาง Vertices ที่ไม่ถูกรวมไว้ใน Envelopes และยังอยู่ที่ตำแหน่งเดิม ไม่เคลื่อนไปตาม Biped ทำให้เกิดเป็น Polygon รูปร่างผิดเพี้ยนไป จึงต้องทำการปรับขนาด Radius เพื่อให้ Vertices ถูกรวมอยู่ใน Envelopes อย่างถูกต้อง ทำให้การแสดงผล Animation เป็นไปอย่างถูกต้อง

1.เริ่มการทำ Envelopes ให้เปิด Model ที่ได้ทำการ Attach กับ Biped ไว้แล้วขึ้นมา
2.ให้แน่ใจว่าปิด Figure Mode ไว้แล้ว เนื่องจากไม่สามารถทำ Animation ได้ใน Figure Mode
3.เปิด Animation หรือสร้าง Animation ใดๆที่ต้องการให้กับ Model cและสังเกตดู Polygon ของ Model ในแต่ละเฟรม แต่ละ Viewport ว่าแสดงผลอย่างถูกต้อง
4.ถ้าเกิดจุด Vertices ใดๆที่แสดงผลไม่ถูกต้องดังในภาพ ดังนั้นเราจึงต้องทำการปรับ Envelopes ให้กับ Model



รูปที่2-77 Envelope

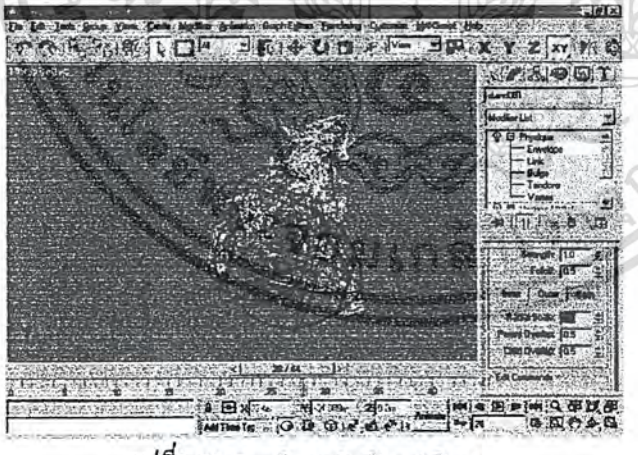
5.ให้ทำการ Select mesh และไปที่ Physique modifier
6.เลือก Hide Attach nodes เพื่อให้เห็น mesh ได้ถนัดขึ้น
7.เลื่อน Time Slider ไปที่ Frame ที่มีปัญหา



รูปที่2-77 Envelope

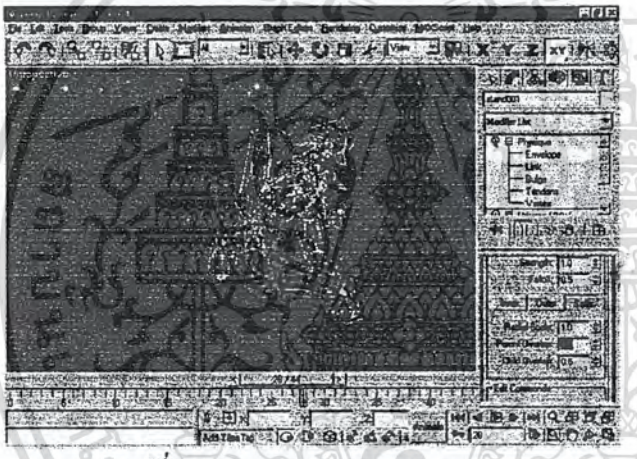
8.เลือก Envelopes จาก Physique Modifier

9.Select จุดที่ mesh เกิดผิดไปจากรูปร่างปกติ ให้ปรับขนาด Radial ขนาด Parent\Child Overlao จนกระทั่งความผิดปกติที่ mesh หายไป



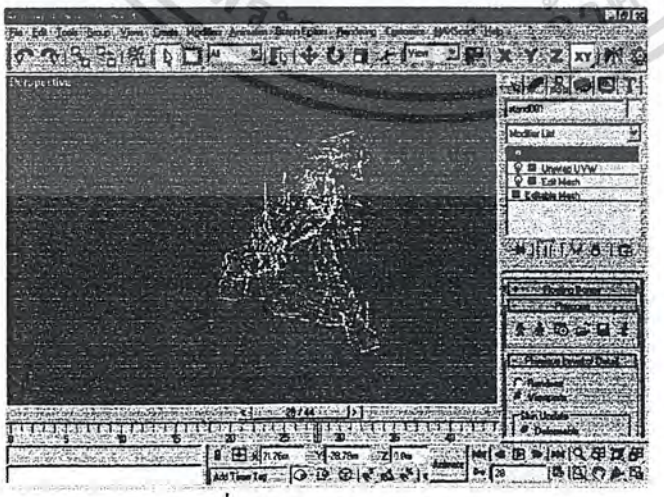
รูปที่2-78 Envelope Radius Adjust

รูปที่2-79 Envelope Parent Adjust



รูปที่2-80 Envelope Child Adjust

10.ปรับเช่นนี้ที่ทุกจุดของ Model ที่ผิดปกติ



รูปที่2-81 Envelope Finish

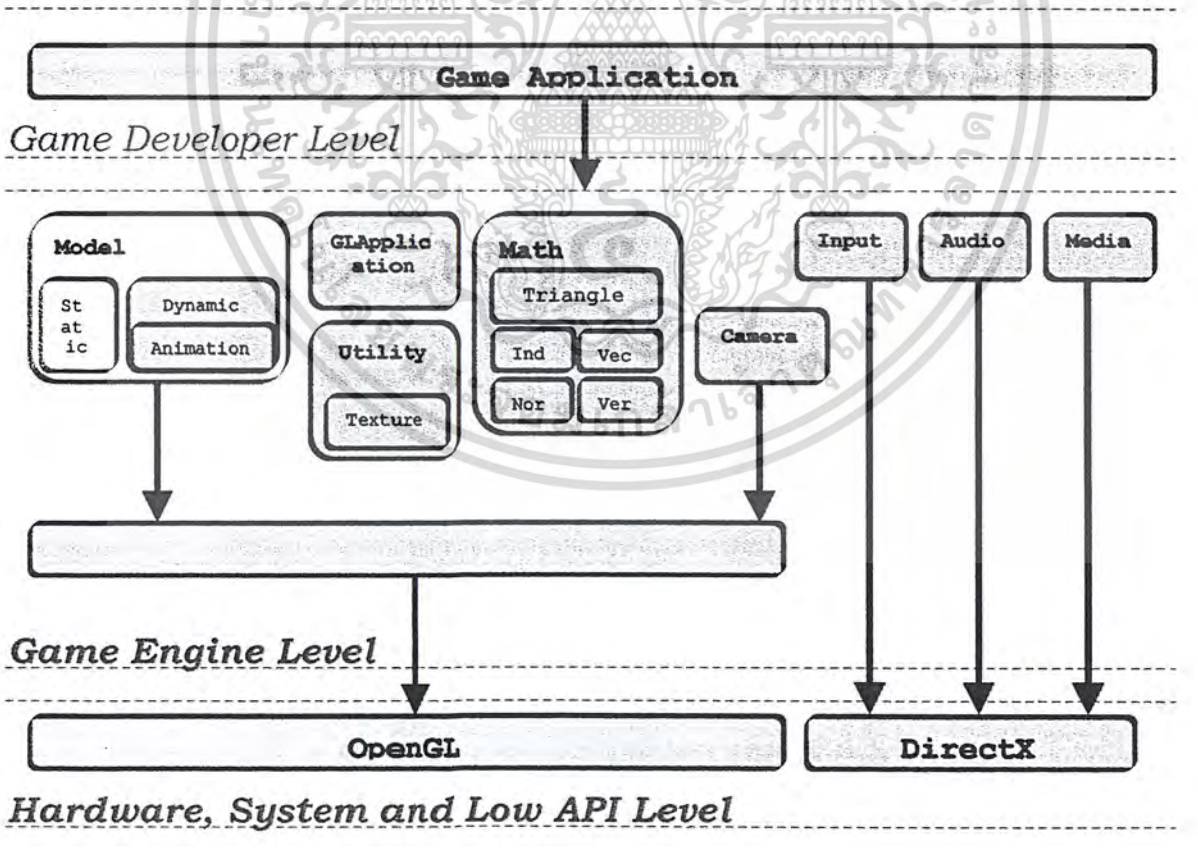
บทที่ 3
การออกแบบเอนจินต์สำหรับเกม 3 มิติ

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 โมเดล (Model)

สำหรับ Model Engine จะเน้นหน้าที่ในการจัดการข้อมูลต่างๆที่จะนำมาแสดงผลซึ่งแบ่งออกเป็นสองส่วน คือ Model ประเภท Static และ Model ประเภท Dynamic โดยทั้งสองส่วนจะใช้ข้อมูลเริ่มต้นที่เป็น Model เดียวกัน ซึ่งประกอบไปด้วยคลาส ต่างๆ ดังต่อไปนี้

- CModel

  เป็น Base Class ของ Class CStaticModel และ CDynamicModel

3.2.2 สแตติกโมเดล (StaticModel)

สแตติกโมเดลเป็นส่วนที่จะใช้ในการจัดการตัวละครหรือวัตถุในเกมที่ไม่มีการทำ แอนิเมชัน ในส่วนของสแตติกโมเดลนี้มีคลาสดังนี้

- CUseStaticModelData

  ทำหน้าที่เป็นตัวจัดการ Access ข้อมูลประเภท Static Model ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model

- CStaticModel

  ทำหน้าที่จัดการกับ Model ที่เป็นแบบ Static มีการเรียกค่าต่างๆ เกี่ยวกับ Model นั้น และ Interface เพื่อเรียกใช้งานสะดวกในการเข้าถึงข้อมูลต่างๆของ Model Engine

- CStaticModelDataManage

  ทำหน้าที่เป็นตัวจัดการข้อมูลประเภท Static ที่จะถูกนำไปใช้ในการกำหนดวัตถุแบบ อยู่นิ่ง ซึ่งในบางครั้งอาจจะมีการเรียกใช้งานข้อมูล Static นั้นซ้ำๆ เช่นจอง Memory สำหรับ Model เดิม CStaticModelDataManage ก็จะมาคอยช่วยจัดการไม่ให้มีการ Load ข้อมูลขึ้นมาซ้ำๆ เพื่อประหยัดการใช้งาน Memory

- CStaticModelData

  ทำหน้าที่จัดการข้อมูลโมเดลของวัตถุ มี Interface เพื่อสะดวกในการเข้าถึงข้อมูลต่างๆของ Model Engine ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

- CSLandModel

  เป็น Class interface สำหรับผู้ใช้เพื่อสะดวกในการนำไปใช้

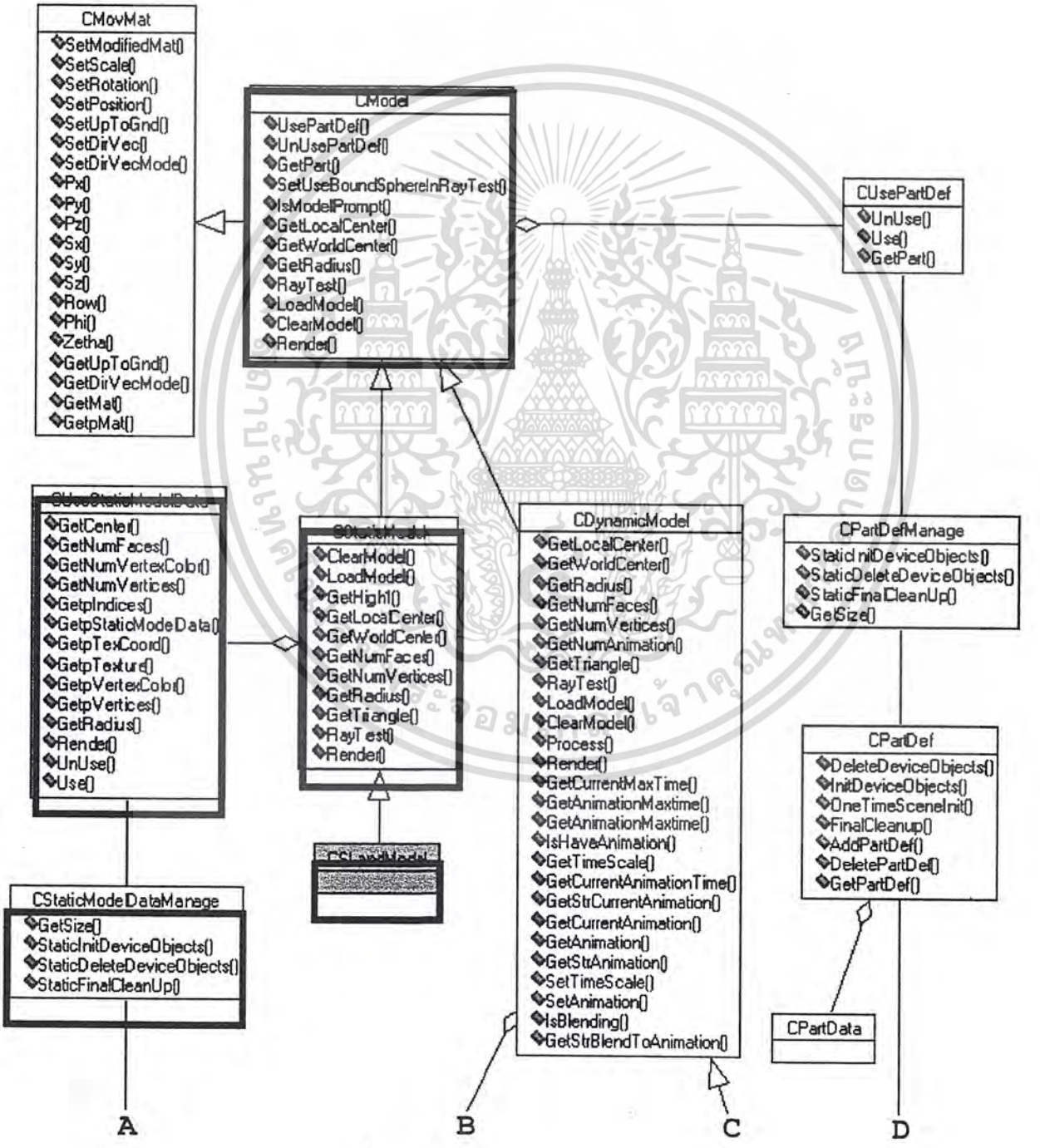
เป็น Class interface สำหรับผู้ใช้เพื่อสะดวกในการนำไปใช้

3.3 คลาสไดอะแกรมของเกมเอนจินท์ (Class Diagram of Game Engine)

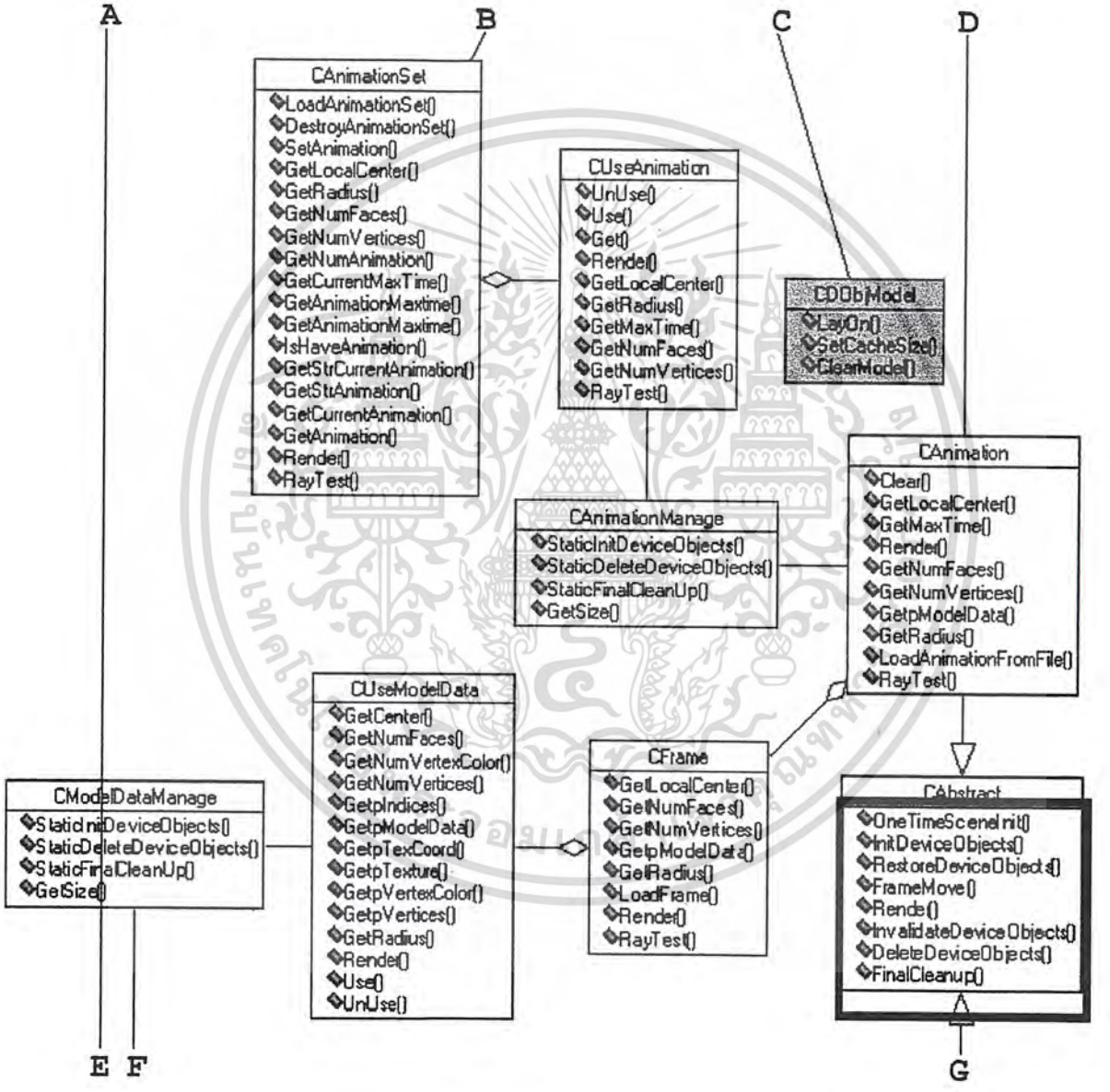
สัญลักษณ์ Class สี □ หมายถึง Internal Class of Game Engine

สัญลักษณ์ Class สี ■ หมายถึง Interface Class of Developer

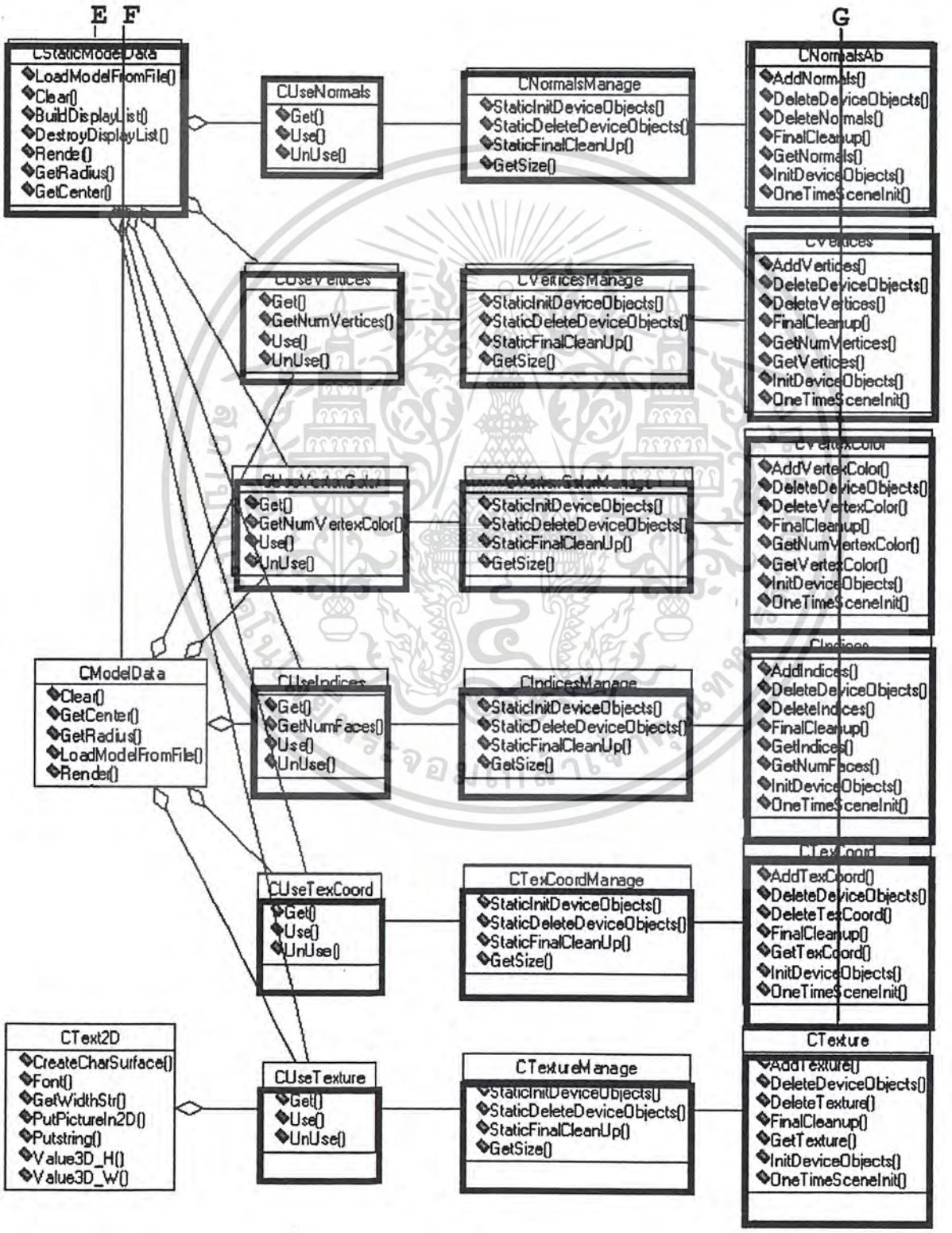
สัญลักษณ์เส้นทึบ ▢ ล้อมรอบ หมายถึง Class ที่จะอธิบายในปริญญานิพนธ์ฉบับนี้



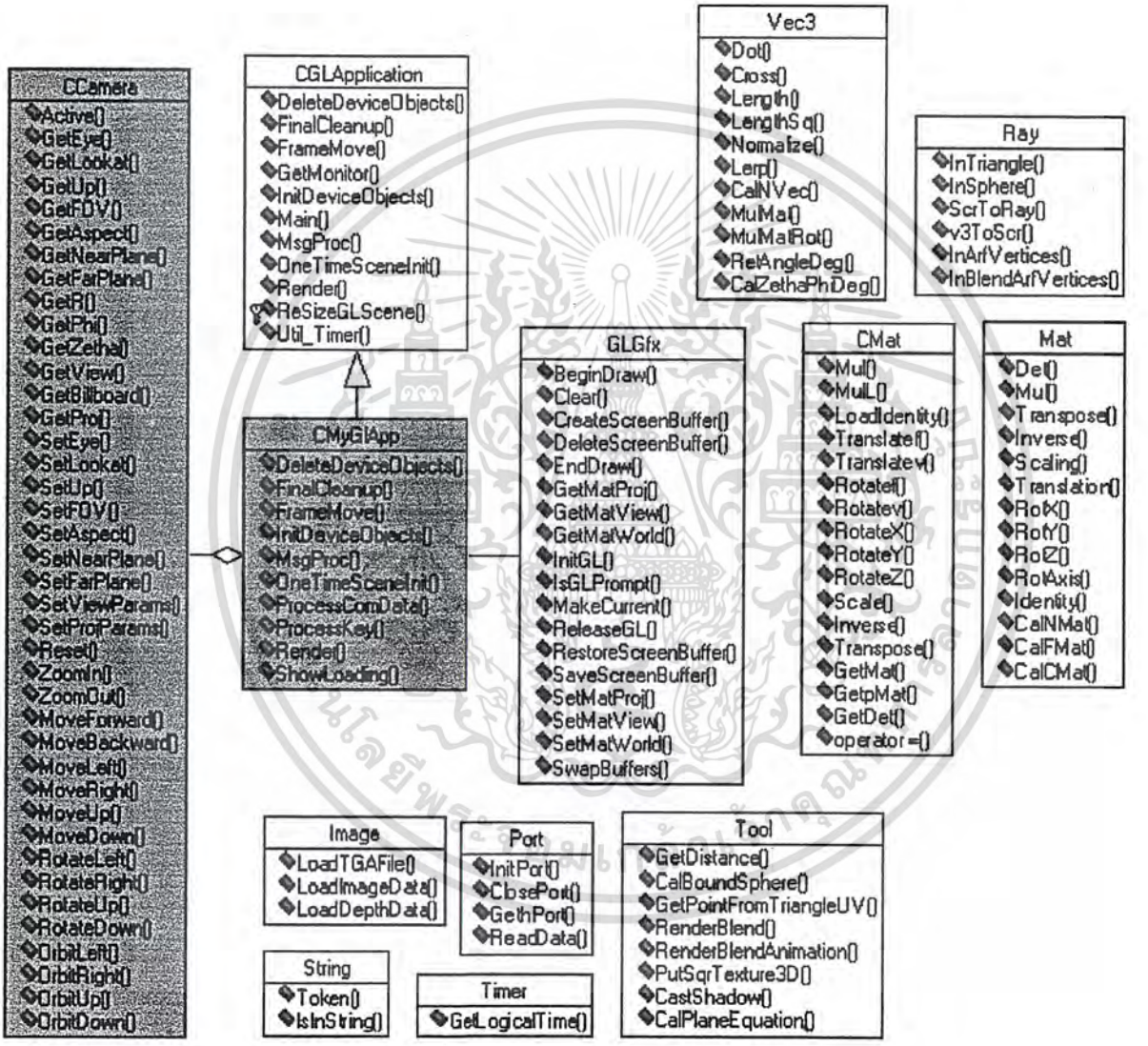
รูปที่ 3-2ก แสดง Class Diagram ของ Game Engine ส่วนที่ 1



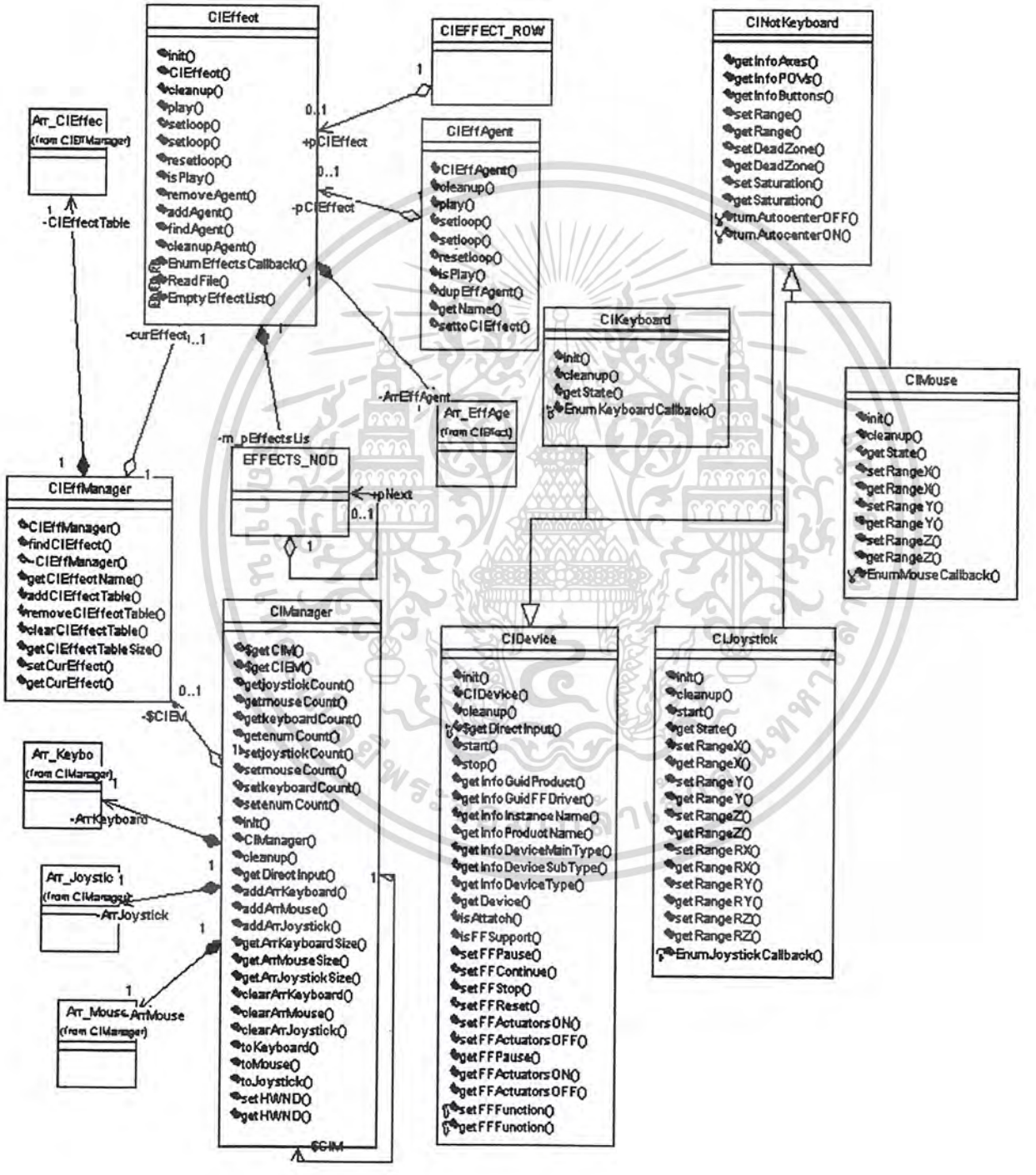
รูปที่3-2ข แสดง Class Diagram ของ Game Engine ส่วนที่ 2


E F
G
CStaticModelData
LoadModelFromFile()
Clear()
BuildDisplayList()
DestroyDisplayList()
Render()
GetRadius()
GetCenter()
CUseNormals
Get()
Use()
UnUse()
CNormalsManage
StaticInitDeviceObjects()
StaticDeleteDeviceObjects()
StaticFinalCleanUp()
GetSize()
CNormalsAb
AddNormals()
DeleteDeviceObjects()
DeleteNormals()
FinalCleanup()
GetNormals()
InitDeviceObjects()
OneTimeSceneInit()
CUseVertices
Get()
GetNumVertices()
Use()
UnUse()
CVerticesManage
StaticInitDeviceObjects()
StaticDeleteDeviceObjects()
StaticFinalCleanUp()
GetSize()
CVertices
AddVertices()
DeleteDeviceObjects()
DeleteVertices()
FinalCleanup()
GetNumVertices()
GetVertices()
InitDeviceObjects()
OneTimeSceneInit()
CUseVertexColor
Get()
GetNumVertexColor()
Use()
UnUse()
CVertexColorManage
StaticInitDeviceObjects()
StaticDeleteDeviceObjects()
StaticFinalCleanUp()
GetSize()
CVertexColor
AddVertexColor()
DeleteDeviceObjects()
DeleteVertexColor()
FinalCleanup()
GetNumVertexColor()
GetVertexColor()
InitDeviceObjects()
OneTimeSceneInit()
CModelData
Clear()
GetCenter()
GetRadius()
LoadModelFromFile()
Render()
CUseIndices
Get()
GetNumFaces()
Use()
UnUse()
CIndicesManage
StaticInitDeviceObjects()
StaticDeleteDeviceObjects()
StaticFinalCleanUp()
GetSize()
CIndices
AddIndices()
DeleteDeviceObjects()
DeleteIndices()
FinalCleanup()
GetIndices()
GetNumFaces()
InitDeviceObjects()
OneTimeSceneInit()
CUseTexCoord
Get()
Use()
UnUse()
CTexCoordManage
StaticInitDeviceObjects()
StaticDeleteDeviceObjects()
StaticFinalCleanUp()
GetSize()
CTexCoord
AddTexCoord()
DeleteDeviceObjects()
DeleteTexCoord()
FinalCleanup()
GetTexCoord()
InitDeviceObjects()
OneTimeSceneInit()
CText2D
CreateCharSurface()
Font()
GetWidthStr()
PutPictureIn2D()
Putstring()
Value3D_H()
Value3D_W()
CUseTexture
Get()
Use()
UnUse()
CTextureManage
StaticInitDeviceObjects()
StaticDeleteDeviceObjects()
StaticFinalCleanUp()
GetSize()
CTexture
AddTexture()
DeleteDeviceObjects()
DeleteTexture()
FinalCleanup()
GetTexture()
InitDeviceObjects()
OneTimeSceneInit()

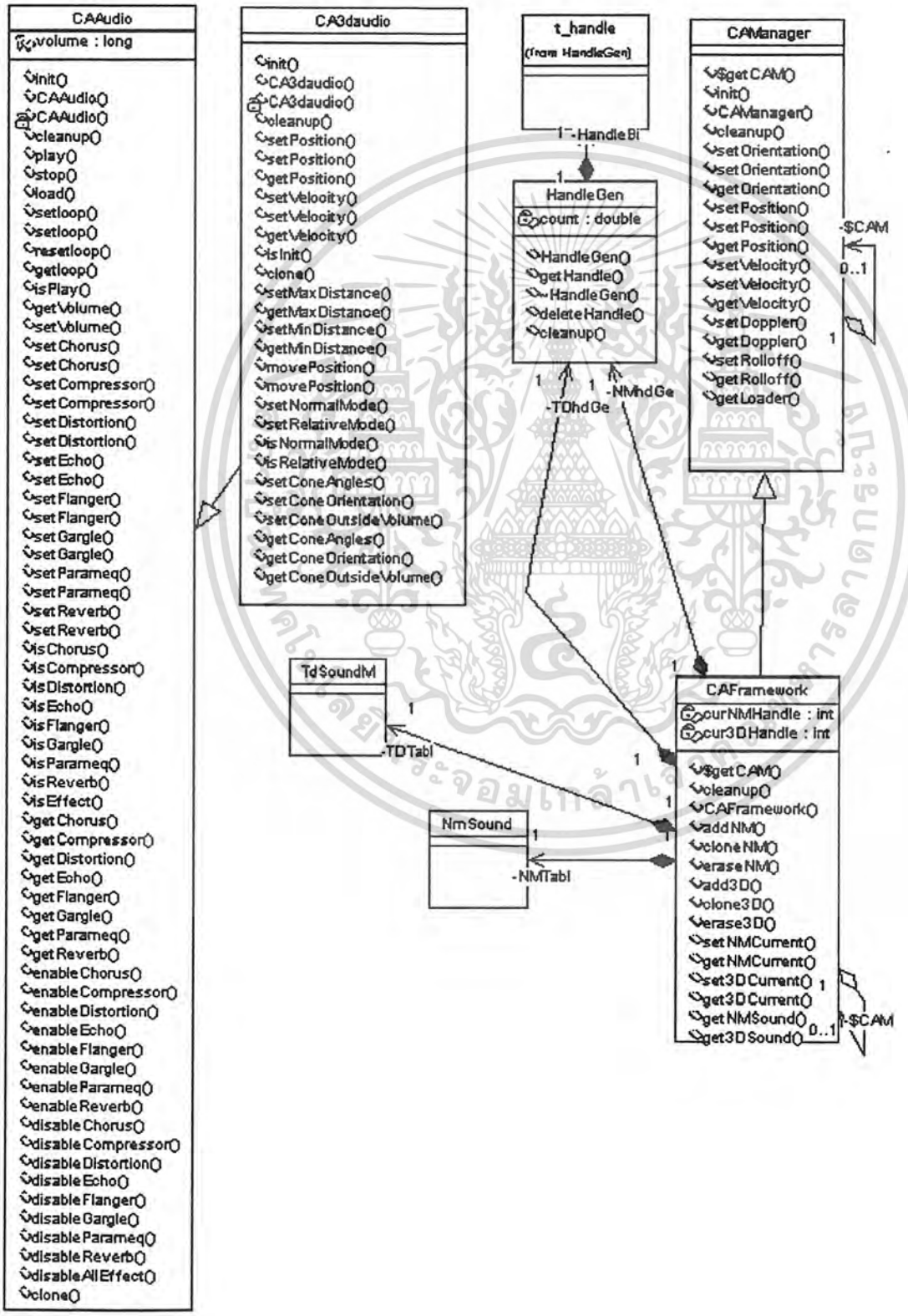
รูปที่3-2ค แสดง Class Diagram ของ Game Engine ส่วนที่ 3



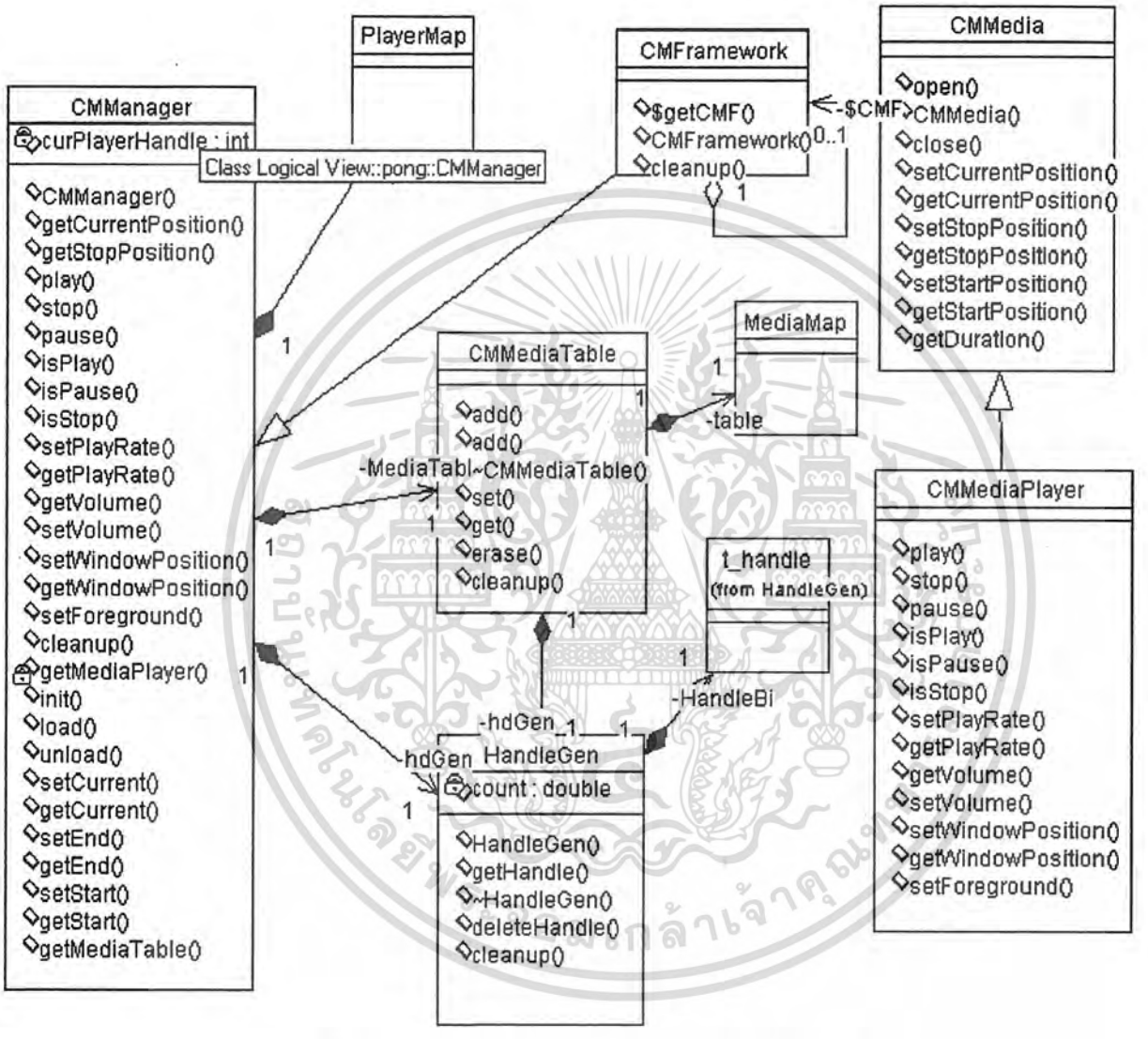
รูปที่3-2ง แสดง Class Diagram ของ Game Engine ส่วนที่ 4


CIEffect
init()
CIEffect()
cleanup()
play()
setloop()
setloop()
resetloop()
isPlay()
removeAgent()
addAgent()
findAgent()
cleanupAgent()
EnumEffectsCallback()
ReadFile()
EmptyEffectList()
CIEFFECT_ROW
CIEffAgent
CIEffAgent()
cleanup()
play()
setloop()
setloop()
resetloop()
isPlay()
dupEffAgent()
getName()
settoCIEffect()
CINotKeyboard
getInfoAxes()
getInfoPOVs()
getInfoButtons()
setRange()
getRange()
setDeadZone()
getDeadZone()
setSaturation()
getSaturation()
EnumAutocenterOFF()
EnumAutocenterON()
CIKeyboard
init()
cleanup()
getState()
EnumKeyboardCallback()
CIMouse
init()
cleanup()
getState()
setRangeX()
getRangeX()
setRangeY()
getRangeY()
setRangeZ()
getRangeZ()
EnumMouseCallback()
Arr_CIEffec
(from CIEffManager)
-CIEffectTable
-pCIEffect
-curEffect
-ArrEffAgent
Arr_EffAge
(from CIEffect)
-m_pEffectsLis
EFFECTS_NOD
+pNext
CIEffManager
CIEffManager()
findCIEffect()
~CIEffManager()
getCIEffectName()
addCIEffectTable()
removeCIEffectTable()
clearCIEffectTable()
getCIEffectTableSize()
setCurEffect()
getCurEffect()
-$CIEM
CIManager
$getCIM()
$getCIEM()
getjoystickCount()
getmouseCount()
getkeyboardCount()
getenumCount()
setjoystickCount()
setmouseCount()
setkeyboardCount()
setenumCount()
init()
CIManager()
cleanup()
getDirectInput()
addArrKeyboard()
addArrMouse()
addArrJoystick()
getArrKeyboardSize()
getArrMouseSize()
getArrJoystickSize()
clearArrKeyboard()
clearArrMouse()
clearArrJoystick()
toKeyboard()
toMouse()
toJoystick()
setHWND()
getHWND()
$CIM
CIDevice
init()
CIDevice()
cleanup()
$getDirectInput()
start()
stop()
getInfoGuidProduct()
getInfoGuidFFDriver()
getInfoInstanceName()
getInfoProductName()
getInfoDeviceMainType()
getInfoDevioeSubType()
getInfoDeviceType()
getDevice()
isAttatch()
isFFSupport()
setFFPause()
setFFContinue()
setFFStop()
setFFReset()
setFFActuatorsON()
setFFActuatorsOFF()
getFFPause()
getFFActuatorsON()
getFFActuatorsOFF()
setFFFunction()
getFFFunction()
CIJoystick
init()
cleanup()
start()
getState()
setRangeX()
getRangeX()
setRangeY()
getRangeY()
setRangeZ()
getRangeZ()
setRangeRX()
getRangeRX()
setRangeRY()
getRangeRY()
setRangeRZ()
getRangeRZ()
EnumJoystickCallback()
Arr_Keybo
(from CIManager)
-ArrKeyboard
Arr_Joystic
(from CIManager)
-ArrJoystick
Arr_Mouse
(from CIManager)
-ArrMouse

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
การพัฒนาเอนจินต์ สำหรับ เกม 3 มิติ

การพัฒนาเอนจินต์สำหรับปริญญานิพนธ์ฉบับนี้ จะอธิบายรายละเอียดเฉพาะในส่วนของ Static Model Engine และ Model อันประกอบไปด้วยคลาสต่างๆดังต่อไปนี้

CModel

CUseStaticModelData

CStaticModel

CStaticModelData

CStaticModelDataManage

CSLandModel

CAbstract

CVertices

CVerticesManage

CUseVertices

CVertexColor

CVertexColorManage

CIndices

CIndicesManage

CUseIndices

CTexCoord

CTexCoordManage

CUseTexCoord

CNormalsAb

CNormalsManage

CUseNormals

CTexture

CTextureManage

CUseTexture

4.1 โมเดล (Model)

สำหรับ Model Engine จะเน้นหน้าที่ในการจัดการข้อมูลต่างๆที่จะนำมาแสดงผลซึ่งแบ่งออกเป็นสองส่วนคือ Model ประเภท Static และ Model ประเภท Dynamic โดยทั้งสองส่วนจะใช้ข้อมูลเริ่มต้นที่เป็น Model เดียวกัน ซึ่งประกอบไปด้วยคลาส ต่างๆ ดังต่อไปนี้

Class CModel

CModel เป็น Base Class ของ Class CStaticModel และ CDynamicModel

CModel::UsePartDef

ฟังก์ชันเพื่อการเข้าถึงหรือ Access Part ของ Model นั้นๆ ซึ่งจะเป็นตัว เรียกสร้าง หรือ Create Part ที่จะถูกใช้งานซึ่งเรียกมาจาก User ในแต่ละครั้ง

HRESULT CModel::UsePartDef

(

const char *strFileName

);

Paramcters

strFileName

String ซึ่งใช้เป็น Key สำหรับการ Search เพื่อที่จะเรียกใช้ Use กับ Part นั้นๆ

Return Values

-

Remarks

ทำหน้าที่เป็น Reference Count Increment สำหรับการ Access ขอใช้งาน Part

CModel::UnUsePartDef

ฟังก์ชันเพื่อยกเลิกการเข้าถึงหรือ Access Part ของ Model นั้นๆ

HRESULT CModel::UnUsePartDef

(

void

);

Parameters

-

Return Values

-

Remarks

ทำหน้าที่เป็น Reference Count Decrement สำหรับยกเลิกการ Access ใช้งาน Part

CModel::GetPart

ฟังก์ชันเพื่อ Access Part โดยจะคืนค่ามาเป็น Pointer ของ Part ของ Model นั้นๆ ที่ถูกเรียก Use ไว้ก่อนหน้านี้

```
float* Get
(
    void
);
```

Parameters

-

Return Values

Pointer to float ที่ชี้ไปยัง Part ของ Model ขณะนั้น

Remarks

เป็นฟังก์ชันสำหรับเรียก Access ไปยัง Part ที่ถูก Use อยู่อีกที

CModel::SetUseBoundSphereInRayTest

ฟังก์ชันเพื่อ กำหนดขอบเขต Bounding Sphere ให้กับวัตถุนั้นๆ

```
inline void SetUseBoundSphereInRayTest
(
    BOOL bUse
);
```

Parameters

bUse

กำหนดให้มีการใช้ Bounding Sphere

Return Values

-

Remarks

เป็นฟังก์ชันสำหรับกำหนด Bounding Sphere สำหรับการเช็คชน

CModel::IsModelPrompt

ฟังก์ชันที่คอยบอกถึง Model ที่พร้อมที่จะถูกนำมาใช้

```
inline BOOL IsModelPrompt
```

{

void

);

Parameters

-

Return Values

Bool เป็นผลลัพธ์บอกถึงสถานะของ Model

Remarks

ใช้สำหรับการบอกถึงความพร้อมของ Model

CModel::GetLocalCenter

ฟังก์ชันสำหรับรับค่า ตำแหน่งศูนย์กลางอ้างอิงภายใน (Local) ของวัตถุซึ่งอยู่ระหว่างสองเฟรม โดยผู้ใช้จะต้องส่งค่า Weight เพื่อที่จะคำนวณหา Local Center ของช่วงของเฟรมนั้นๆ

D3DXVECTOR3 GetLocalCenter

(

void

);

Parameters

-

Return Values

D3DXVECTOR3 ซึ่งเป็นตำแหน่ง Local Center เป็นผลลัพธ์ที่ได้จากคำนวณ

Remarks

ใช้สำหรับการคำนวณหาตำแหน่งปัจจุบันของจุดระหว่าง 2 จุด ซึ่งถูกกำหนดโดย Weight

CModel::GetWorldCenter

ฟังก์ชันสำหรับรับค่า ตำแหน่งศูนย์กลางอ้างอิงของพื้นที่ฉาก (World)

D3DXVECTOR3 GetWorldCenter

(

void

);

Parameters

-

Return Values

D3DXVECTOR3 ซึ่งเป็นตำแหน่ง World Center เป็นผลลัพธ์ที่ได้จากคำนวณ

Remarks

-

CModel::Raytest

ฟังก์ชันสำหรับทดสอบการชนของวัตถุ (Collision Detection)

```
BOOL RayTest
(
    const D3DXVECTOR3&      RayOrigin,
    const D3DXVECTOR3&      RayDirection,
    const D3DXMATRIX*       matWorld,
    float                   fBPos,
    BOOL                    bUseBoundSphereInRayTest,
    int*                    pFaceIndex      = NULL,
    CTriangle*              pTriangle       = NULL,
    float*                  pDistance       = NULL,
    float*                  pU              = NULL,
    float*                  pV              = NULL
);
```

Parameters

RayOrigin

เป็น Vector สำหรับระบุตำแหน่งจุดกำเนิดหรือจุดเริ่มต้นของแสง หรือเส้นตรง

RayDirection

เป็น Vector สำหรับระบุทิศทางของแสง หรือเส้นตรง โดยมีขนาดเท่ากับระยะห่างของแสง

matWorld

เป็น Matrix สำหรับระบุ World เพื่อใช้สำหรับการคำนวณทดสอบ RayTest

fBPos

เป็นค่า Weight สำหรับการคำนวณหาการชนแบบ Sphere Bounding

bUseBoundSphereInRayTest

เป็น Flag ระบุเพื่อบอกว่าจะมีการเช็คชนโดยใช้ Bound Sphere หรือไม่

pFaceIndex

คือ Output ซึ่งเป็น Pointer ที่ชี้ไปยัง Index ของ Face ของวัตถุที่ทำการเช็คชน

pTriangle

คือ Output ซึ่งเป็น Pointer ที่ชี้ไปยัง CTriangle ที่ระบุ Polygon ที่ทำการเช็คชนกับแสง

pDistance

คือ Output ซึ่งเป็น Pointer ที่ชี้ไปยังค่าที่เก็บระยะ Line of sight หรือระยะห่างของการชน

pU

คือ Output ซึ่งเป็น Pointer ที่ชี้ไปยังขนาดของ U vector

pV

คือ Output ซึ่งเป็น Pointer ที่ชี้ไปยังขนาดของ V vector

Return Values

ถ้าการทดสอบแล้วเกิดการชนระหว่างแสงกับวัตถุแล้วฟังก์ชันจะคืนค่า booleadn 'true'

ถ้าการทดสอบแล้วไม่เกิดการชนระหว่างแสงกับวัตถุแล้วฟังก์ชันจะคืนค่า booleadn 'false'

Remarks

-

CModel::LoadModel

ฟังก์ชันสำหรับบรรจุวัตถุเคลื่อนไหวจากไฟล์

```
void LoadModel
(
    const char*     strFileName
);
```

Parameters

strFileName

เป็น Pointer Handle ที่ต้องการโหลดข้อมูลวัตถุเคลื่อนไหว

Return Values

-

Remarks

-

CModel::ClearModel

ฟังก์ชันสำหรับลบเฟรมของภาพเคลื่อนไหว

```
void Clear
(
    void
);
```

Parameters

-

Return Values

-

Remarks

-

CModel::Render

ฟังก์ชันสำหรับแสดงผลวัตถุออกทางจอภาพ

HRESULT Render

(

const D3DXMATRIX* matCustom

);

Parameters

matCustom

ค่าเวลาที่จะแสดงภาพ

Return Values

S_OK หากไม่มีการผิดพลาดของการแสดงผล

Remarks

ใช้สำหรับการสั่งให้แสดงวัตถุออกทางจอภาพ

4.2 สแตติกโมเดล (StaticModel)

สแตติกโมเดลเป็นส่วนที่จะใช้ในการจัดการตัวละครหรือวัตถุในเกมที่ไม่มีการทำ แอนิเมชัน ในส่วนของสแตติกโมเดลนี้มีคลาสดังนี้

Class CUseStaticModelData

CUseStaticModelData ทำหน้าที่เป็นตัวจัดการ Acccss ข้อมูลประเภท Static Model ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CUseStaticModelData::GetCenter

ฟังก์ชันสำหรับรับค่า ตำแหน่งศูนย์กลางของโมเดล

D3DXVECTOR3 GetCenter

(

void

);

Parameters

-

Return Values

D3DXVECTOR3 ซึ่งเป็นตำแหน่งศูนย์กลางของโมเดล

Remarks

-

CUseStaticModelData::GetNumFaces

ฟังก์ชันเพื่อรับค่าจำนวนของ Indices ทั้งหมดของ Model นั้นๆ ที่ถูกเรียก Use ไว้ก่อนหน้านี้

int GetNumFaces

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกจำนวนของ Indices ทั้งหมดของ Model ขณะนั้น

Remarks

เป็นฟังก์ชันสำหรับเรียก Access ไปยัง Indices ที่ถูก Use อยู่อีกที

CUseStaticModelData::GetNumVertexColor

ฟังก์ชันเพื่อรับค่าจำนวนของ Vertices ทั้งหมดของ Model นั้นๆ

int GetNumVertexColor

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกจำนวนของ Vertices ทั้งหมดของ Model ขณะนั้น

Remarks

-

CUseStaticModelData::GetNumVertices

ฟังก์ชันเพื่อรับค่าจำนวนของ Model ทั้งหมดของ Model นั้นๆ ที่ถูกเรียก Use ไว้ก่อนหน้านี้

int GetNumVertices

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกจำนวนของ Model ทั้งหมดของ Model ขณะนั้น

Remarks

เป็นฟังก์ชันสำหรับเรียก Access ไปยัง Model ที่ถูก Use อยู่อีกที

CUseStaticModelData::GetpIndices

ฟังก์ชันเพื่อ Access Indices โดยจะคืนค่ามาเป็น Pointer ของ Indices ของ Model นั้นๆ

unsigned int* GetpIndices

(

void

);

Parameters

-

Return Values

Pointer to unsigned int ที่ชี้ไปยัง Indices ของ Model ขณะนั้น

Remarks

มีไว้เพื่อบางครั้งผู้ใช้ต้องการที่จะทราบข้อมูล Indices เพื่อนำไปใช้งานบางอย่างตามต้องการ

CUseStaticModelData::GetpStaticModelData

ฟังก์ชันสำหรับ Access ไปยัง Model data ของแต่ละเฟรม

CStaticModelData * GetpStaticModelData()

(

void

);

Parameters

-

Return Values

Pointer ที่ชี้ไปยัง CStaticModelData ของเฟรมนั้นๆ ที่ต้องการจะ Access

Remarks

ถูกเรียกใช้ในกรณีที่ผู้ใช้ต้องการนำไปใช้งานหรือจัดการ Frame ด้วยตัวเอง

CUseStaticModelData::GetpTextCoord

ฟังก์ชันเพื่อ Access Part โดยจะคืนค่ามาเป็น Pointer ของ Part ของ Texture Coordinate นั้นๆ

float * GetpTexCoord

(

int nIndex

);

Parameters

int nIndex

เก็บจำนวน index ที่ชี้ไปยัง Texture Coordinate

Return Values

Pointer to float ชี้ไปยัง Part ของ Texture Coordinate ขณะนั้น

Remarks

มีไว้เพื่อบางครั้งผู้ใช้ต้องการที่จะทราบข้อมูล Part เพื่อนำไปใช้งานบางอย่างตามต้องการ

CUseStaticModelData::GetpTexture

ฟังก์ชันเพื่อ Access Part โดยจะคืนค่ามาเป็น Pointer ของ Part ของ Texture นั้นๆ

GLuint GetpTexture

(

int nIndex

);

Parameters

int nIndex

เก็บจำนวน index ที่ชี้ไปยัง Texture Coordinate

Return Values

Pointer to GLuint ชี้ไปยัง Part ของ Texture ขณะนั้น

Remarks

มีไว้เพื่อบางครั้งผู้ใช้ต้องการที่จะทราบข้อมูล Part เพื่อนำไปใช้งานบางอย่างตามต้องการ

CUseStaticModelData::GetpVertexColor

ฟังก์ชันเพื่อ Access Vertices โดยจะคืนค่ามาเป็น Pointer ของ Vertices ของ Model นั้นๆ

float* GetpVertexColor

(

void

);

Parameters

-

Return Values

Pointer to float ที่ชี้ไปยัง Vertices ของ Model ขณะนั้น

Remarks

มีไว้เพื่อบางครั้งผู้ใช้ต้องการที่จะทราบข้อมูล Vertices เพื่อนำไปใช้งานบางอย่างตามต้องการ

CUseStaticModelData::GetpVertices

ฟังก์ชันเพื่อ Access Vertices โดยจะคืนค่ามาเป็น Pointer ของ Vertices ของ Model นั้นๆ

float* GetpVertices

(

void

);

Parameters

-

Return Values

Pointer to float ที่ชี้ไปยัง Vertices ของ Model ขณะนั้น

Remarks

มีไว้เพื่อบางครั้งผู้ใช้ต้องการที่จะทราบข้อมูล Vertices เพื่อนำไปใช้งานบางอย่างตามต้องการ

CUseStaticModelData::GetRadius

ฟังก์ชันสำหรับรับค่ามุมของ Model data ซึ่งมีหน่วยเป็น Radius ของแต่ละเฟรม

float GetRadius

(

void

);

Parameters

fBPos

ไม่ถูกนำมาใช้งานสำหรับ GetRadius ซึ่งเป็น member ของ CFrame นี้

Return Values

float ซึ่งเป็นค่าค่ามุมของ Model data ซึ่งมีหน่วยเป็น Radius

Remarks

-

CUseStaticModelData::Render

ฟังก์ชันสำหรับวาดภาพแสดงผลของ Frame ออกทางจอภาพ

void Render

(

void

);

Parameters

fBPos

ตำแหน่งอ้างอิงสำหรับการ Render ถูกใช้เป็นค่า Weight หรือ Scale ที่จะนำมาคำนวณด้วยการ Interpolate หาค่าตำแหน่งของการ Render วัตถุ

fBUV

ตำแหน่งอ้างอิงสำหรับการ Map texture coordiantes ซึ่งถูกใช้เป็นค่า Weight หรือ Scale ที่จะนำมาคำนวณด้วยการ Interpolate สำหรับการทำ Texture Map Blending

Return Values

-

Remarks

ฟังก์ชันที่ถูกเรียกภายในสำหรับวาดภาพของแต่ละ Frame

CUseStaticModelData::Use

ฟังก์ชันเพื่อการเข้าถึงหรือ Access Model ของ Model นั้นๆ ซึ่งจะเป็นตัว เรียกสร้าง หรือ Create Model ที่จะถูกใช้งานซึ่งเรียกมาจาก User ในแต่ละครั้ง

void Use

(

const CString& pName

);

Parameters

pName

String ซึ่งใช้เป็น Key สำหรับการ Search เพื่อที่จะเรียกใช้ Use กับ Model นั้นๆ

Return Values

-

Remarks

ทำหน้าที่เป็น Reference Count Increment สำหรับการ Access ขอใช้งาน Model

CUseStaticModelData::UnUse

ฟังก์ชันเพื่อยกเลิกการเข้าถึงหรือ Access Model ของ Model นั้นๆ

```
void UnUse
(
    void
);
```

Parameters

-

Return Values

-

Remarks

ทำหน้าที่เป็น Reference Count Decrement สำหรับยกเลิกการ Access ใช้งาน Model

---

Class CStaticModel

CStaticModel ทำหน้าที่จัดการกับ Model ที่เป็นแบบ Static มีการเรียกค่าต่างๆ เกี่ยวกับ Model นั้น และ Interface เพื่อเรียกใช้งานสะดวกในการเข้าถึงข้อมูลต่างๆของ Model Engine ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CStaticModel::GetLocalCenter

ฟังก์ชันสำหรับรับค่า ตำแหน่งศูนย์กลางอ้างอิงภายใน (Local) ของวัตถุซึ่งอยู่ระหว่างสองเฟรม โดยผู้ใช้จะต้องส่งค่า Weight เพื่อที่จะคำนวณหา Local Center ของช่วงของเฟรมนั้นๆ

```
D3DXVECTOR3 GetLocalCenter
(
    void
);
```

Parameters

-

Return Values

D3DXVECTOR3 ซึ่งเป็นตำแหน่ง Local Center เป็นผลลัพธ์ที่ได้จากคำนวณ

Remarks

ใช้สำหรับการคำนวณหาตำแหน่งปัจจุบันของจุดระหว่าง 2 จุด ซึ่งถูกกำหนดโดย Weight

CStaticModel::GetHigh1

ฟังก์ชันสำหรับรับค่าความสูงของ วัตถุโมเดล

```
OOL CStaticModel::GetHigh1
(                       // In
                        float px, float pz,
                        // Out
                        float*    pHigh,
                        int*      pFaceIndex,
                        CTriangle* pTriangle
);
```

Parameters

px ,pz เป็นค่าพิกัด ด้านกว้างและด้านยาว

Return Values

pHigh ซึ่งเป็น ค่าความสูงของวัตถุ

Remarks

ใช้สำหรับการคำนวณหาตำแหน่งความสูงของวัตถุ

CStaticModel::GetWorldCenter

ฟังก์ชันสำหรับรับค่า ตำแหน่งศูนย์กลางอ้างอิงของพื้นที่ฉาก (World)

```
D3DXVECTOR3 GetWorldCenter
(
    void
);
```

Parameters

-

Return Values

D3DXVECTOR3 ซึ่งเป็นตำแหน่ง World Center เป็นผลลัพธ์ที่ได้จากคำนวณ

Remarks

-

CStaticModel::GetRadius

ฟังก์ชันสำหรับรับค่ามุมของ Model data ซึ่งมีหน่วยเป็น Radius ของแต่ละเฟรม

```
float GetRadius
(
    void
);
```

Parameters

-

Return Values

float ซึ่งเป็นค่าค่ามุมของ Model data ซึ่งมีหน่วยเป็น Radius

Remarks

-

CStaticModel::GetNumFaces

ฟังก์ชันสำหรับรับค่า จำนวน Faces ของแต่ละเฟรม

```
int GetNumFaces
(
    void
);
```

Parameters

-

Return Values

Integer ซึ่งเป็นจำนวน Faces ของเฟรมนั้นๆ

Remarks

ถูกเรียกใช้เพื่อต้องการทราบจำนวน Faces เพื่อใช้งานบางอย่าง เช่น collision detection

CStaticModel::GetNumVertices

ฟังก์ชันสำหรับรับค่า จำนวน Vertices ของแต่ละเฟรม

```
int GetNumVertices
(
    void
```

);

Parameters

-

Return Values

Integer ซึ่งเป็นจำนวน Vertices ของเฟรมนั้นๆ

Remarks

ถูกเรียกใช้เพื่อต้องการทราบจำนวน Vertices เพื่อใช้งานบางอย่างแล้วแต่ผู้ใช้ต้องการนำไปใช้งาน

CStaticModel::GetTriangle

ฟังก์ชันสำหรับหาจำนวนรูปสามเหลี่ยมที่ประกอบในวัตถุเคลื่อนไหว

```
int GetTriangle
(
    void
);
```

Parameters

-

Return Values

จำนวนสามเหลี่ยม

Remarks

-

CStaticModel::Raytest

ฟังก์ชันสำหรับทดสอบการชนของวัตถุ (Collision Detection)

```
BOOL RayTest
(
    const D3DXVECTOR3&      RayOrigin,
    const D3DXVECTOR3&      RayDirection,
    const D3DXMATRIX*       matWorld,
    float                   fBPos,
    BOOL                    bUseBoundSphereInRayTest,
    int*                    pFaceIndex      = NULL,
    CTriangle*              pTriangle       = NULL,
    float*                  pDistance       = NULL,
```

float* **pU** **= NULL,**

float* **pV** **= NULL**

);

Parameters

RayOrigin

เป็น Vector สำหรับระบุตำแหน่งจุดกำเนิดหรือจุดเริ่มต้นของแสง หรือเส้นตรง

RayDirection

เป็น Vector สำหรับระบุทิศทางของแสง หรือเส้นตรง โดยมีขนาดเท่ากับระยะห่างของแสง

matWorld

เป็น Matrix สำหรับระบุ World เพื่อใช้สำหรับการคำนวณทดสอบ RayTest

fBPos

เป็นค่า Weight สำหรับการคำนวณหาการชนแบบ Sphere Bounding

bUseBoundSphereInRayTest

เป็น Flag ระบุเพื่อบอกว่าจะมีการเช็คชนโดยใช้ Bound Sphere หรือไม่

pFaceIndex

คือ Output ซึ่งเป็น Pointer ที่ชี้ไปยัง Index ของ Face ของวัตถุที่ทำการเช็คชน

pTriangle

คือ Output ซึ่งเป็น Pointer ที่ชี้ไปยัง CTriangle ที่ระบุ Polygon ที่ทำการเช็คชนกับแสง

pDistance

คือ Output ซึ่งเป็น Pointer ที่ชี้ไปยังค่าที่เก็บระยะ Line of sight หรือระยะห่างของการชน

pU

คือ Output ซึ่งเป็น Pointer ที่ชี้ไปยังขนาดของ U vector

pV

คือ Output ซึ่งเป็น Pointer ที่ชี้ไปยังขนาดของ V vector

Return Values

ถ้าการทดสอบแล้วเกิดการชนระหว่างแสงกับวัตถุแล้วฟังก์ชันจะคืนค่า booleadn 'true'

ถ้าการทดสอบแล้วไม่เกิดการชนระหว่างแสงกับวัตถุแล้วฟังก์ชันจะคืนค่า booleadn 'false'

Remarks

-

CStaticModel::LoadModel

ฟังก์ชันสำหรับบรรจุวัตถุเคลื่อนไหวจากไฟล์

void LoadModel

(

const char* strFileName

);

Parameters

strFileName

เป็น Pointer Handle ที่ต้องการโหลดข้อมูลวัตถุเคลื่อนไหว

Return Values

-

Remarks

-

CStaticModel::ClearModel

ฟังก์ชันสำหรับลบเฟรมของภาพเคลื่อนไหว

void Clear

(

void

);

Parameters

-

Return Values

-

Remarks

-

CStaticModel::Render

ฟังก์ชันสำหรับแสดงผลวัตถุออกทางจอภาพ

HRESULT Render

(

const D3DXMATRIX* matCustom

);

Parameters

matCustom

ค่าเวลาที่จะแสดงภาพ

Return Values

S_OK หากไม่มีการผิดพลาดของการแสดงผล

Remarks

ใช้สำหรับการสั่งให้แสดงวัตถุออกทางจอภาพ

Class CStaticModelDataManage

CStaticModelDataManage ทำหน้าที่เป็นตัวจัดการข้อมูลประเภท Static ที่จะถูกนำไปใช้ในการกำหนดวัตถุแบบ อยู่นิ่ง ซึ่งในบางครั้งอาจจะมีการเรียกใช้งานข้อมูล Static นั้นซ้ำๆ เช่นจอง Memory สำหรับ Model เดิม CStaticModelDataManage ก็จะมาคอยช่วยจัดการไม่ให้มีการ Load ข้อมูลขึ้นมาซ้ำๆ เพื่อประหยัดการใช้งาน Memory ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CStaticModelDataManage::StaticInitDeviceObjects

ทำหน้าที่เป็นส่วน ควบคุมการการ Initialize Object ก่อนที่จะนำ Object นั้นไปใช้งาน ของ Model นั้นๆ เพื่อควบคุมการ InitDeviceObjects ซ้ำๆของ Model

void StaticInitDeviceObjects

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ InitDeviceObjects ของ CStaticModel อีกที

CStaticModelDataManage::StaticDeleteDeviceObjects

ทำหน้าที่ควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาที่ CStaticModelData ของ Model เพื่อควบคุมการ DeleteDeviceObjects ซ้ำๆของ Model

void StaticDeleteDeviceObjects

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ DeleteDeviceObjects ของ CStaticModelData อีกที

CStaticModelDataManage::StaticFinalCleanUp

ทำหน้าที่เป็นส่วนควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาที่ CStaticModelData ของ Model เพื่อควบคุมการ FinalCleanup ชั้ๆของ Model

void StaticFinalCleanUp

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ FinalCleanup ของ CStaticModelData อีกที

CStaticModelDataManage::GetSize

ฟังก์ชันเพื่อรับค่าเพื่อบอกขนาดของ Part ทั้งหมดของ Model นั้นๆ ที่ได้มีการจองเอาไว้ในตอนเริ่มต้นก่อนหน้านี้

int GetSize

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกขนาดของ Part ทั้งหมดของ Model ที่ได้มีการจองเอาไว้ก่อนหน้านี้

Remarks

-

Class CStaticModelData

CStaticModelData ทำหน้าที่จัดการข้อมูลโมเดลของวัตถุ มี Interface เพื่อสะดวกในการเข้าถึงข้อมูลต่างๆของ Model Engine ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CStaticModelData::Clear

ฟังก์ชันสำหรับลบเฟรมของภาพเคลื่อนไหว

void Clear

(

void

);

Parameters

-

Return Values

-

Remarks

-

CStaticModelData::GetCenter

ฟังก์ชันสำหรับรับค่า ตำแหน่งศูนย์กลางของโมเดล

D3DXVECTOR3 GetCenter

(

void

);

Parameters

-

Return Values

D3DXVECTOR3 ซึ่งเป็นตำแหน่งศูนย์กลางของโมเดล

Remarks

-

CStaticModelData::BuildDisplayList

ฟังก์ชันสำหรับสร้าง DisplayList เพื่อง่ายต่อการนำไปใช้

void CStaticModelData::BuildDisplayList

(

void

);

Parameters

-

Return Values

-

Remarks

-

CStaticModelData::DestroyDisplayList

ฟังก์ชันสำหรับทำลาย DisplayList

void CStaticModelData::DestroyDisplayList

(

void

);

Parameters

-

Return Values

-

Remarks

-

CStaticModelData::GetRadius

ฟังก์ชันสำหรับรับค่ามุมของ Model data ซึ่งมีหน่วยเป็น Radius ของแต่ละเฟรม

float GetRadius

(

void

);

Parameters

-

Return Values

float ซึ่งเป็นค่าค่ามุมของ Model data ซึ่งมีหน่วยเป็น Radius

Remarks

-

CStaticModelData::LoadModelFromFile

ฟังก์ชันสำหรับบรรจุวัตถุเคลื่อนไหวจากไฟล์

void LoadAnimationFromFile

(

const char* strModelFileName

);

Parameters

strModelFileName

เป็น Pointer Handle ที่ต้องการ โหลดข้อมูลวัตถุเคลื่อนไหว

Return Values

-

Remarks

-

CStaticModelData::Render

ฟังก์ชันสำหรับวาดภาพแสดงผลของ Frame ออกทางจอภาพ

void Render

(

void

);

Parameters

-

Return Values

-

Remarks

ฟังก์ชันที่ถูกเรียกภายในสำหรับวาดภาพของแต่ละ Frame

Class CSLandModel

เป็น Class interface สำหรับผู้ใช้เพื่อสะดวกในการนำไปใช้

Class CAbstract

CAbstract ทำหน้าที่เป็น Base Class ซึ่งจะเป็นตัวกำหนด Overide functions เพื่อให้ Derived Class ที่สืบทอดไปนั้นมี Interface เพื่อเรียกใช้งานที่เหมือนกัน เพื่อสะดวกในการเข้าถึงข้อมูลต่างๆของ Model Engine ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CAbstract::OneTimeSceneInit

ทำหน้าที่เป็น Base Virtual Function สำหรับถูกเรียกเพื่อทำการ Initialize Object เพียงครั้งเดียวก่อนที่จะเริ่มต้น ฉากในเกม

HRESULT OneTimeSceneInit

(

void

);

Parameters

-

Return Values

S_OK

Remarks

ยังไม่ถูก Implement ใน CAbstract นี้

CAbstract::InitDeviceObjects

ทำหน้าที่เป็น Base Virtual Function สำหรับถูกเรียกเพื่อทำการ Initialize Object ก่อนที่จะนำ Object นั้นไปใช้งาน

HRESULT InitDeviceObjects

(

void

);

Parameters

-

Return Values

S_OK

Remarks

ยังไม่ถูก Implement ใน CAbstraet นี้

CAbstract::RestoreDeviceObjects

ทำหน้าที่เป็น Base Virtual Function สำหรับถูกเรียกเพื่อทำการ Recover Object เพราะเนื่องจากในบางครั้งอาจจะมีการเปลี่ยน Display Mode ในการแสดงผล ดังนั้นจึงต้องมีการ ปรับขนาด Buffer ที่จองเอาไว้ก่อนหน้านี้เพื่อให้เหมาะสมสำหรับการใช้งานในแต่ละ Mode การแสดงผล

HRESULT RestoreDeviceObjects

(

void

);

Parameters

-

Return Values

S_OK

Remarks

ยังไม่ถูก Implement ใน CAbstract นี้

CAbstract::FrameMove

ทำหน้าที่เป็น Base Virtual Function มีไว้สำหรับให้ผู้ใช้นำไป Implement ต่อซึ่งจะเป็นฟังก์ชันสำหรับการคำนวณและประมวลผลสำหรับ แต่ละ Model เช่น การเคลื่อนที่, AI โดยฟังก์ชันนี้จะถูกเรียกให้ทำงานทุกๆครั้งในแต่ละรอบของ Game Loop

HRESULT FrameMove

(

void

);

Parameters

-

Return Values

S_OK

Remarks

ยังไม่ถูก Implement ใน CAbstract นี้

CAbstract::Render

ทำหน้าที่เป็น Base Virtual Function มีไว้สำหรับให้ผู้ใช้นำไป Implement ต่อซึ่งจะเป็นฟังก์ชันสำหรับการวาดภาพสำหรับ แต่ละ Model ซึ่งบางครั้งผู้ใช้อาจจะต้องการเป็นผู้กำหนดวิธีวาดภาพเองเช่นการทำ Effects ต่างๆ โดยฟังก์ชันนี้จะถูกเรียกให้ทำงานทุกๆครั้งในแต่ละรอบของ Game Loop

HRESULT Render

(

void

);

Parameters

-

Return Values

S_OK

Remarks

ยังไม่ถูก Implement ใน CAbstract นี้

CAbstract::InvalidateDeviceObjects

ทำหน้าที่เป็น Base Virtual Function มีไว้สำหรับให้ผู้ใช้นำไป Implement ต่อซึ่งจะเป็นฟังก์ชันสำหรับการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ โดยฟังก์ชันนี้จะถูกเรียกให้ทำงานในกรณีที่ก่อนจะทำการ Switch Display Mode หรือกรณีที่จะมีการเปลี่ยนฉากหรือยกเลิกใช้ Object นั้นๆ

HRESULT InvalidateDeviceObjects

(

void

);

Parameters

-

Return Values

S_OK

Remarks

ยังไม่ถูก Implement ใน CAbstract นี้

CAbstract::DeleteDeviceObjects

ทำหน้าที่เป็น Base Virtual Function มีไว้สำหรับให้ผู้ใช้นำไป Implement ต่อซึ่งจะเป็นฟังก์ชันสำหรับการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ โดยฟังก์ชันนี้จะถูกเรียกให้ทำงานในกรณีที่จะยกเลิกใช้ Object นั้นๆซึ่งในบ้างครั้งผู้ใช้อาจจะมีการจอง Memory เอาไว้ในตอน InitDeviceObjects

HRESULT DeleteDeviceObjects

(

void

);

Parameters

-

Return Values

S_OK

Remarks

ยังไม่ถูก Implement ใน CAbstract นี้

CAbstract::FinalCleanup

ทำหน้าที่เป็น Base Virtual Function มีไว้สำหรับให้ผู้ใช้นำไป Implement ต่อซึ่งจะเป็นฟังก์ชันสำหรับการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ของ Objcct นั้นๆ โดยฟังก์ชันนี้จะถูกเรียกให้ทำงานในกรณีที่ก่อนจะจบเกมจะถูกเรียกเพียงครั้งสุดท้ายเพียงครั้งเดียว

HRESULT FinalCleanup

(

void

);

Parameters

-

Return Values

S_OK

Remarks

ยังไม่ถูก Implement ใน CAbstract นี้

Class CVertices

CVertices ทำหน้าที่เป็น เก็บข้อมูลและจัดการข้อมูลประเภท Vertex ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ที่ถูกอ่านขึ้นมาจากไฟล์ข้อมูลประเภท .Map โดยเป็น Derived Class ของ CAbstract ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CVertices::AddVertices

ทำหน้าที่เพิ่ม Vertex เข้าไปเก็บยัง List ของ Vertices ทั้งหมดของ Model นั้น

BOOL AddVertices

(

void

);

Parameters

-

Return Values

TRUE

Remarks

การทำงานภายในจะเป็นการ Increment จำนวนของ Vertices ทั้งหมดขณะนั้นของ Model

CVertices::DeleteDeviceObjects

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ โดยฟังก์ชันนี้จะถูกเรียกให้ทำงานในกรณีที่จะยกเลิกใช้ Object นั้นๆ และจำนวนของ Vertices ยังมากกว่าศูนย์

HRESULT DeleteDeviceObjects

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ DeleteDeviceObjects ของ Vertices

CVertices::DeleteVertices

ทำหน้าที่ควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ ของ Vertices นั้นๆ ซึ่งจะเป็นการ Decrease จำนวนของ Vertices ทั้งหมดลงทีละ 1 vertex เรียกจนกว่าจะหมดจึงจะ Cleanup Memory ให้

int DeleteVertices

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกจำนวนของ Vertices ทั้งหมดของ Model ขณะนั้น

Remarks

ภายในจะเรียก DeleteDeviceObjects และ FinalCleanup เพื่อคืน Memory ของ Vertices นั้นๆ

CVertices::FinalCleanup

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ของ Object นั้นๆ โดยฟังก์ชันนี้จะถูกเรียกให้ทำงานในกรณีที่ต้องการให้คืน Memory ทั้งหมดทันทีที่ถูกเรียก

HRESULT FinalCleanup

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ FinalCleanup ของ Vertices

CVertices::GetNumVertices

ฟังก์ชันเพื่อรับค่าจำนวนของ Vertices ทั้งหมดของ Model นั้นๆ

int GetNumVertices

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกจำนวนของ Vertices ทั้งหมดของ Model ขณะนั้น

Remarks

-

CVertices::GetVertices

ฟังก์ชันเพื่อ Access Vertices โดยจะคืนค่ามาเป็น Pointer ของ Vertices ของ Model นั้นๆ

float* GetVertices

(

void

);

Parameters

-

Return Values

Pointer to float ที่ชี้ไปยัง Vertices ของ Model ขณะนั้น

Remarks

มีไว้เพื่อบางครั้งผู้ใช้ต้องการที่จะทราบข้อมูล Vertices เพื่อนำไปใช้งานบางอย่างตามต้องการ

CVertices::OneTimeSceneInit

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการ Initialize Object เพียงครั้งเดียวก่อนที่จะเริ่มต้นฉากในเกม ของ Model นั้นๆ

HRESULT OneTimeSceneInit

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ OneTimeSceneInit ของ Vertices

CVertices::InitDeviceObjects

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการ Initialize Object ก่อนที่จะนำ Object นั้นไปใช้งาน ของ Model นั้นๆ

HRESULT InitDeviceObjects

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ InitDeviceObjects ของ Vertices

Class CVerticesManage

CVerticesManage ทำหน้าที่เป็นตัวจัดการข้อมูลประเภท Vertex ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ซึ่งในบางครั้งอาจจะมีการเรียกใช้งานข้อมูลของ Vertices นั้นซ้ำๆ เช่นจอง Memory สำหรับ Vertices เดิม CVerticesManage ก็จะมาคอยช่วยจัดการไม่ให้มีการ Load ข้อมูลขึ้นมาซ้ำๆ เพื่อประหยัดการใช้งาน Memory ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CVerticesManage::StaticInitDeviceObjects

ทำหน้าที่เป็นส่วน ควบคุมการการ Initialize Vertices Object ก่อนที่จะนำ Object นั้นไปใช้งาน ของ Model นั้นๆ เพื่อควบคุมการ InitDeviceObjects ซ้ำๆของ Model

void StaticInitDeviceObjects

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ InitDeviceObjects ของ CVertices อีกที

CVerticesManage::StaticDeleteDeviceObjects

ทำหน้าที่ควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาที่ CVertices ของ Model เพื่อควบคุมการ DeleteDeviceObjects ซ้ำๆของ Model

void StaticDeleteDeviceObjects

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ DeleteDeviceObjects ของ CVertices อีกที

CVerticesManage::StaticFinalCleanUp

ทำหน้าที่เป็นส่วนควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาที่ CVertices ของ Model เพื่อควบคุมการ FinalCleanup ซ้ำๆของ Model

void StaticFinalCleanUp

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ FinalCleanup ของ CVertices อีกที

CVerticesManage::GetSize

ฟังก์ชันเพื่อรับค่าเพื่อบอกขนาดของ Vertices ทั้งหมดของ Model นั้นๆ ที่ได้มีการจองเอาไว้ในตอนเริ่มต้นก่อนหน้านี้

int GetSize

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกขนาดของ Vertices ทั้งหมดของ Model ที่ได้มีการจองเอาไว้ก่อนหน้านี้

Remarks

-

Class CUseVertices

CUseVertices ทำหน้าที่เป็นตัวจัดการ Access ข้อมูลประเภท Vertices ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CUseVertices::Use

ฟังก์ชันเพื่อการเข้าถึงหรือ Access Vertices ของ Model นั้นๆ ซึ่งจะเป็นตัว เรียกสร้าง หรือ Create Vertices ที่จะถูกใช้งานซึ่งเรียกมาจาก User ในแต่ละครั้ง

void Use

(

const CString& pName

);

Parameters

pName

String ซึ่งใช้เป็น Key สำหรับการ Search เพื่อที่จะเรียกใช้ Use กับ Vertices นั้นๆ

Return Values

-

Remarks

ทำหน้าที่เป็น Reference Count Increment สำหรับการ Access ขอใช้งาน Vertices

CUseVertices::UnUse

ฟังก์ชันเพื่อยกเลิกการเข้าถึงหรือ Access Vertices ของ Model นั้นๆ

void UnUse

(

void

);

Parameters

-

Return Values

-

Remarks

ทำหน้าที่เป็น Reference Count Decrement สำหรับยกเลิกการ Access ใช้งาน Vertices

CUseVertices::GetNumVertices

ฟังก์ชันเพื่อรับค่าจำนวนของ Vertices ทั้งหมดของ Model นั้นๆ ที่ถูกเรียก Use ไว้ก่อนหน้านี้

int GetNumVertices

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกจำนวนของ Vertices ทั้งหมดของ Model ขณะนั้น

Remarks

เป็นฟังก์ชันสำหรับเรียก Access ไปยัง Vertices ที่ถูก Use อยู่อีกที

CUseVertices::Get

ฟังก์ชันเพื่อ Access Vertices โดยจะคืนค่ามาเป็น Pointer ของ Vertices ของ Model นั้นๆ ที่ถูกเรียก Use ไว้ก่อนหน้านี้

float* Get

(

void

);

Parameters

-

Return Values

Pointer to float ที่ชี้ไปยัง Vertices ของ Model ขณะนั้น

Remarks

เป็นฟังก์ชันสำหรับเรียก Access ไปยัง Vertices ที่ถูก Use อยู่อีกที

Class CVertexColor

CVertexColor ทำหน้าที่เป็น เก็บข้อมูลและจัดการข้อมูลประเภท Vertex Color ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ที่ถูกอ่านขึ้นมาจากไฟล์ข้อมูลประเภท .Map และ .Mac โดยเป็น Derived Class ของ CAbstract ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CVertexColor::AddVertexColor

ทำหน้าที่เพิ่ม Vertex เข้าไปเก็บยัง List ของ Vertices ทั้งหมดของ Model นั้น

BOOL AddVertexColor

(

void

);

Parameters

-

Return Values

TRUE

Remarks

การทำงานภายในจะเป็นการ Increment จำนวนของ Vertices ทั้งหมดขณะนั้นของ Model

CVertexColor::DeleteDeviceObjects

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ โดยฟังก์ชันนี้จะถูกเรียกให้ทำงานในกรณีที่จะยกเลิกใช้ Object นั้นๆ และจำนวนของ Vertices ยังมากกว่าศูนย์

HRESULT DeleteDeviceObjects

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ DeleteDeviceObjects ของ Vertices

CVertexColor::DeleteVertexColor

ทำหน้าที่ควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ ของ Vertices นั้นๆ ซึ่งจะเป็นการ Decrease จำนวนของ Vertices ทั้งหมดลงทีละ 1 vertex เรียกจนกว่าจะหมดจึงจะ Cleanup Memory ให้

int DeleteVertexColor

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกจำนวนของ Vertices ทั้งหมดของ Model ขณะนั้น

Remarks

ภายในจะเรียก DeleteDeviceObjects และ FinalCleanup เพื่อคืน Memory ของ Vertices นั้นๆ

CVertexColor::FinalCleanup

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ของ Object นั้นๆ โดยฟังก์ชันนี้จะถูกเรียกให้ทำงานในกรณีที่ต้องการให้คืน Memory ทั้งหมดทันทีที่ถูกเรียก

HRESULT FinalCleanup

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ FinalCleanup ของ Vertices

CVertexColor::GetNumVertexColor

ฟังก์ชันเพื่อรับค่าจำนวนของ Vertices ทั้งหมดของ Model นั้นๆ

int GetNumVertexColor

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกจำนวนของ Vertices ทั้งหมดของ Model ขณะนั้น

Remarks

-

CVertexColor::GetVertexColor

ฟังก์ชันเพื่อ Access Vertices โดยจะคืนค่ามาเป็น Pointer ของ Vertices ของ Model นั้นๆ

float* GetVertexColor

(

void

);

Paramctcrs

-

Return Values

Pointer to float ที่ชี้ไปยัง Vertices ของ Model ขณะนั้น

Remarks

มีไว้เพื่อบางครั้งผู้ใช้ต้องการที่จะทราบข้อมูล Vertices เพื่อนำไปใช้งานบางอย่างตามต้องการ

CVertexColor::OneTimeSceneInit

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการ Initialize Object เพียงครั้งเดียวก่อนที่จะเริ่มต้นฉากในเกม ของ Model นั้นๆ

HRESULT OneTimeSceneInit

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ OneTimeSceneInit ของ Vertices

CVertexColor::InitDeviceObjects

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการ Initialize Object ก่อนที่จะนำ Object นั้นไปใช้งาน ของ Model นั้นๆ

HRESULT InitDeviceObjects

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ InitDeviceObjects ของ Vertices

---

Class CVertexColorManage

CVertexColorManage ทำหน้าที่เป็นตัวจัดการข้อมูลประเภท Vertex ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ซึ่งในบางครั้งอาจจะมีการเรียกใช้งานข้อมูลของ Vertices นั้นซ้ำๆ เช่นจอง Memory สำหรับ Vertices เดิม CVertexColorManage ก็จะมาคอยช่วยจัดการไม่ให้มีการ Load ข้อมูลขึ้นมาซ้ำๆ เพื่อประหยัดการใช้งาน Memory ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CVertexColorManage::StaticInitDeviceObjects

ทำหน้าที่เป็นส่วน ควบคุมการการ Initialize Vertices Object ก่อนที่จะนำ Object นั้นไปใช้งาน ของ Model นั้นๆ เพื่อควบคุมการ InitDeviceObjects ซ้ำๆของ Model

void StaticInitDeviceObjects

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ InitDeviceObjects ของ CVertexColor อีกที

CVertexColorManage::StaticDeleteDeviceObjects

ทำหน้าที่ควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาที่ CVertexColor ของ Model เพื่อควบคุมการ DeleteDeviceObjects ซ้ำๆของ Model

void StaticDeleteDeviceObjects

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ DeleteDeviceObjects ของ CVertexColor อีกที

CVertexColorManage::StaticFinalCleanUp

ทำหน้าที่เป็นส่วนควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาที่ CVertexColor ของ Model เพื่อควบคุมการ FinalCleanup ซ้ำๆของ Model

void StaticFinalCleanUp

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ FinalCleanup ของ CVertexColor อีกที

CVertexColorManage::GetSize

ฟังก์ชันเพื่อรับค่าเพื่อบอกขนาดของ Vertices ทั้งหมดของ Model นั้นๆ ที่ได้มีการจองเอาไว้ในตอนเริ่มต้นก่อนหน้านี้

int GetSize

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกขนาดของ Vertices ทั้งหมดของ Model ที่ได้มีการจองเอาไว้ก่อนหน้านี้

Remarks

-

Class CIndices

CIndices ทำหน้าที่เป็น เก็บข้อมูลและจัดการข้อมูลประเภท Index ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ที่ถูกอ่านขึ้นมาจากไฟล์ข้อมูลประเภท .Map โดยเป็น Derived Class ของ CAbstract ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CIndices::AddIndices

ทำหน้าที่เพิ่ม Index เข้าไปเก็บยัง List ของ Indices ทั้งหมดของ Model นั้น

BOOL AddIndices

(

void

);

Paramcters

-

Return Values

TRUE

Remarks

การทำงานภายในจะเป็นการ Increment จำนวนของ Indices ทั้งหมดขณะนั้นของ Model

CIndices::DeleteDeviceObjects

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ โดยฟังก์ชันนี้จะถูกเรียกให้ทำงานในกรณีที่จะยกเลิกใช้ Object นั้นๆ และจำนวนของ Indices ยังมากกว่าศูนย์

HRESULT DeleteDeviceObjects

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ DeleteDeviceObjects ของ Indices

CIndices::DeleteIndices

ทำหน้าที่ควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ ของ Indices นั้นๆ ซึ่งจะเป็นการ Decrease จำนวนของ Indices ทั้งหมดลงทีละ 1 Index เรียกจนกว่าจะหมดจึงจะ Cleanup Memory ให้

int DeleteIndices

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกจำนวนของ Indices ทั้งหมดของ Model ขณะนั้น

Remarks

ภายในจะเรียก DeleteDeviceObjects และ FinalCleanup เพื่อคืน Memory ของ Indices นั้นๆ

CIndices::FinalCleanup

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ของ Object นั้นๆ โดยฟังก์ชันนี้จะถูกเรียกให้ทำงานในกรณีที่ต้องการให้คืน Memory ทั้งหมดทันทีที่ถูกเรียก

HRESULT FinalCleanup

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ FinalCleanup ของ Indices

CIndices::GetNumFaces

ฟังก์ชันเพื่อรับค่าจำนวนของ Indices ทั้งหมดของ Model นั้นๆ

int GetNumFaces

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกจำนวนของ Indices ทั้งหมดของ Model ขณะนั้น

Remarks

-

CIndices::GetIndices

ฟังก์ชันเพื่อ Access Indices โดยจะคืนค่ามาเป็น Pointer ของ Indices ของ Model นั้นๆ

float* GetIndices

(

void

);

Parameters

-

Return Values

Pointer to float ที่ชี้ไปยัง Indices ของ Model ขณะนั้น

Remarks

มีไว้เพื่อบางครั้งผู้ใช้ต้องการที่จะทราบข้อมูล Indices เพื่อนำไปใช้งานบางอย่างตามต้องการ

CIndices::OneTimeSceneInit

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการ Initialize Object เพียงครั้งเดียวก่อนที่จะเริ่มต้นฉากในเกม ของ Model นั้นๆ

HRESULT OneTimeSceneInit

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ OneTimeSceneInit ของ Indices

CIndices::InitDeviceObjects

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการ Initialize Object ก่อนที่จะนำ Object นั้นไปใช้งาน ของ Model นั้นๆ

HRESULT InitDeviceObjects

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ InitDeviceObjects ของ Indices

---

Class CIndicesManage

CIndicesManage ทำหน้าที่เป็นตัวจัดการข้อมูลประเภท Index ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ซึ่งในบางครั้งอาจจะมีการเรียกใช้งานข้อมูลของ Indices นั้นซ้ำๆ เช่นจอง Memory สำหรับ Indices เดิม CIndicesManage ก็จะมาคอยช่วยจัดการไม่ให้มีการ Load ข้อมูลขึ้นมาซ้ำๆ เพื่อประหยัดการใช้งาน Memory ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CIndicesManage::StaticInitDeviceObjects

ทำหน้าที่เป็นส่วน ควบคุมการการ Initialize Indices Object ก่อนที่จะนำ Object นั้นไปใช้งาน ของ Model นั้นๆ เพื่อควบคุมการ InitDeviceObjects ซ้ำๆของ Model

void StaticInitDeviceObjects

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ InitDeviceObjects ของ CIndices อีกที

CIndicesManage::StaticDeleteDeviceObjects

ทำหน้าที่ควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาที่ CIndices ของ Model เพื่อควบคุมการ DeleteDeviceObjects ซ้ำๆของ Model

void StaticDeleteDeviceObjects

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ DeleteDeviceObjects ของ CIndices อีกที

CIndicesManage::StaticFinalCleanUp

ทำหน้าที่เป็นส่วนควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาที่ CIndices ของ Model เพื่อควบคุมการ FinalCleanup ซ้ำๆของ Model

void StaticFinalCleanUp

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ FinalCleanup ของ CIndices อีกที

CIndicesManage::GetSize

ฟังก์ชันเพื่อรับค่าเพื่อบอกขนาดของ Indices ทั้งหมดของ Model นั้นๆ ที่ได้มีการจองเอาไว้ในตอนเริ่มต้นก่อนหน้านี้

```
int GetSize
(
    void
);
```

Parameters

-

Return Values

Integer ซึ่งบอกขนาดของ Indices ทั้งหมดของ Model ที่ได้มีการจองเอาไว้ก่อนหน้านี้

Remarks

-

Class CUseIndices

CUseIndices ทำหน้าที่เป็นตัวจัดการ Access ข้อมูลประเภท Indices ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CUseIndices::Use

ฟังก์ชันเพื่อการเข้าถึงหรือ Access Indices ของ Model นั้นๆ ซึ่งจะเป็นตัว เรียกสร้าง หรือ Create Indices ที่จะถูกใช้งานซึ่งเรียกมาจาก User ในแต่ละครั้ง

```
void Use
(
    const CString&  pName
);
```

Parameters

pName

String ซึ่งใช้เป็น Key สำหรับการ Search เพื่อที่จะเรียกใช้ Use กับ Indices นั้นๆ

Return Values

-

Remarks

ทำหน้าที่เป็น Reference Count Increment สำหรับการ Access ของใช้งาน Indices

CUseIndices::UnUse

ฟังก์ชันเพื่อยกเลิกการเข้าถึงหรือ Access Indices ของ Model นั้นๆ

void UnUse

(

void

);

Paramcters

-

Return Values

-

Remarks

ทำหน้าที่เป็น Reference Count Decrement สำหรับยกเลิกการ Access ใช้งาน Indices

CUseIndices::GetNumFaces

ฟังก์ชันเพื่อรับค่าจำนวนของ Indices ทั้งหมดของ Model นั้นๆ ที่ถูกเรียก Use ไว้ก่อนหน้านี้

int GetNumFaces

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกจำนวนของ Indices ทั้งหมดของ Model ขณะนั้น

Remarks

เป็นฟังก์ชันสำหรับเรียก Access ไปยัง Indices ที่ถูก Use อยู่อีกที

CUseIndices::Get

ฟังก์ชันเพื่อ Access Indices โดยจะคืนค่ามาเป็น Pointer ของ Indices ของ Model นั้นๆ ที่ถูกเรียก Use ไว้ก่อนหน้านี้

float* Get

(

void

);

Parameters

-

Return Values

Pointer to float ที่ชี้ไปยัง Indices ของ Model ขณะนั้น

Remarks

เป็นฟังก์ชันสำหรับเรียก Access ไปยัง Indices ที่ถูก Use อยู่อีกที

Class CTexCoord

CTexCoord ทำหน้าที่เป็น เก็บข้อมูลและจัดการข้อมูลประเภท Texture Coordinate ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ที่ถูกอ่านขึ้นมาจากไฟล์ข้อมูลประเภท .Map โดยเป็น Derived Class ของ CAbstract ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CTexCoord::AddTexCoord

ทำหน้าที่เพิ่ม Texture Coordinate เข้าไปเก็บยัง List ของ Texture Coordinate ทั้งหมดของ Model นั้น

```
BOOL AddTexCoord
(
    void
);
```

Parameters

-

Return Values

TRUE

Remarks

การทำงานภายในจะเป็นการ Increment จำนวนของ Texture Coordinate ทั้งหมดขณะนั้นของ Model

CTexCoord::DeleteDeviceObjects

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ โดยฟังก์ชันนี้จะถูกเรียกให้ทำงานในกรณีที่จะยกเลิกใช้ Object นั้นๆ และจำนวนของ Texture Coordinate ยังมากกว่าศูนย์

```
HRESULT DeleteDeviceObjects
(
    void
);
```

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ DeleteDeviceObjects ของ Texture Coordinate

CTexCoord::DeleteTexCoord

ทำหน้าที่ควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ ของ Texture Coordinate นั้นๆ ซึ่งจะเป็นการ Decrease จำนวนของ Texture Coordinate ทั้งหมดลงทีละ 1 Texture Coordinate เรียกจนกว่าจะหมดจึงจะ Cleanup Memory ให้

int DeleteTexCoord

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกจำนวนของ Texture Coordinate ทั้งหมดของ Model ขณะนั้น

Remarks

ภายในจะเรียก DeleteDeviceObjects และ FinalCleanup เพื่อคืน Memory ของ Texture Coordinate นั้นๆ

CTexCoord::FinalCleanup

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ของ Object นั้นๆ โดยฟังก์ชันนี้จะถูกเรียกให้ทำงานในกรณีที่ต้องการให้คืน Memory ทั้งหมดทันทีที่ถูกเรียก

HRESULT FinalCleanup

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ FinalCleanup ของ Texture Coordinate

CTexCoord::GetNumTexCoord

ฟังก์ชันเพื่อรับค่าจำนวนของ Texture Coordinate ทั้งหมดของ Model นั้นๆ

int GetNumTexCoord

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกจำนวนของ Texture Coordinate ทั้งหมดของ Model ขณะนั้น

Remarks

-

CTexCoord::GetTexCoord

ฟังก์ชันเพื่อ Access Texture Coordinate โดยจะคืนค่ามาเป็น Pointer ของ Texture Coordinate ของ Model นั้นๆ

float* GetTexCoord

(

void

);

Parameters

-

Return Values

Pointer to float ที่ชี้ไปยัง Texture Coordinate ของ Model ขณะนั้น

Remarks

มีไว้เพื่อบางครั้งผู้ใช้ต้องการที่จะทราบข้อมูล Texture Coordinate เพื่อนำไปใช้งานบางอย่างตามต้องการ

CTexCoord::OneTimeSceneInit

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการ Initialize Object เพียงครั้งเดียวก่อนที่จะเริ่มต้นฉากในเกม ของ Model นั้นๆ

HRESULT OneTimeSceneInit

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ OneTimeSceneInit ของ Texture Coordinate

CTexCoord::InitDeviceObjects

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการ Initialize Object ก่อนที่จะนำ Object นั้นไปใช้งาน ของ Model นั้นๆ

HRESULT InitDeviceObjects

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ InitDeviceObjects ของ Texture Coordinate

---

Class CTexCoordManage

CTexCoordManage ทำหน้าที่เป็นตัวจัดการข้อมูลประเภท Texture Coordinate ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ซึ่งในบางครั้งอาจจะมีการเรียกใช้งานข้อมูลของ Texture Coordinate นั้นซ้ำๆ เช่นจอง Memory สำหรับ Texture Coordinate เดิม CTexCoordManage ก็จะมาคอยช่วยจัดการไม่ให้มีการ Load ข้อมูลขึ้นมาซ้ำๆ เพื่อประหยัดการใช้งาน Memory ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CTexCoordManage::StaticInitDeviceObjects

ทำหน้าที่เป็นส่วน ควบคุมการการ Initialize Texture Coordinate Object ก่อนที่จะนำ Object นั้นไปใช้งาน ของ Model นั้นๆ เพื่อควบคุมการ InitDeviceObjects ซ้ำๆของ Model

void StaticInitDeviceObjects

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ InitDeviceObjects ของ CTexCoord อีกที

CTexCoordManage::StaticDeleteDeviceObjects

ทำหน้าที่ควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาที่ CTexCoord ของ Model เพื่อควบคุมการ DeleteDeviceObjects ซ้ำๆของ Model

void StaticDeleteDeviceObjects

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ DeleteDeviceObjects ของ CTexCoord อีกที

CTexCoordManage::StaticFinalCleanUp

ทำหน้าที่เป็นส่วนควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาที่ CTexCoord ของ Model เพื่อควบคุมการ FinalCleanup ซ้ำๆของ Model

void StaticFinalCleanUp

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ FinalCleanup ของ CTexCoord อีกที

CTexCoordManage::GetSize

ฟังก์ชันเพื่อรับค่าเพื่อบอกขนาดของ Texture Coordinate ทั้งหมดของ Model นั้นๆ ที่ได้มีการจองเอาไว้ในตอนเริ่มต้นก่อนหน้านี้

int GetSize

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกขนาดของ Texture Coordinate ทั้งหมดของ Model ที่ได้มีการจองเอาไว้ก่อนหน้านี้

Remarks

-

Class CUseTexCoord

CUseTexCoord ทำหน้าที่เป็นตัวจัดการ Access ข้อมูลประเภท Texture Coordinate ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CUseTexCoord::Use

ฟังก์ชันเพื่อการเข้าถึงหรือ Access Texture Coordinate ของ Model นั้นๆ ซึ่งจะเป็นตัว เรียกสร้างหรือ Create Texture Coordinate ที่จะถูกใช้งานซึ่งเรียกมาจาก User ในแต่ละครั้ง

void Use

(

const CString& pName

);

Parameters

pName

String ซึ่งใช้เป็น Key สำหรับการ Search เพื่อที่จะเรียกใช้ Use กับ Texture Coordinate นั้นๆ

Return Values

-

Remarks

ทำหน้าที่เป็น Reference Count Increment สำหรับการ Access ขอใช้งาน Texture Coordinate

CUseTexCoord::UnUse

ฟังก์ชันเพื่อยกเลิกการเข้าถึงหรือ Access Texture Coordinate ของ Model นั้นๆ

void UnUse

(

void

);

Parameters

-

Return Values

-

Remarks

ทำหน้าที่เป็น Reference Count Decrement สำหรับยกเลิกการ Access ใช้งาน Texture Coordinate

CUseTexCoord::GetNumTexCoord

ฟังก์ชันเพื่อรับค่าจำนวนของ Texture Coordinate ทั้งหมดของ Model นั้นๆ ที่ถูกเรียก Use ไว้ก่อนหน้านี้

int GetNumTexCoord

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกจำนวนของ Texture Coordinate ทั้งหมดของ Model ขณะนั้น

Remarks

เป็นฟังก์ชันสำหรับเรียก Access ไปยัง Texture Coordinate ที่ถูก Use อยู่อีกที

CUseTexCoord::Get

ฟังก์ชันเพื่อ Access Texture Coordinate โดยจะคืนค่ามาเป็น Pointer ของ Texture Coordinate ของ Model นั้นๆ ที่ถูกเรียก Use ไว้ก่อนหน้านี้

float* Get

(

void

);

Parameters

-

Return Values

Pointer to float ที่ชี้ไปยัง Texture Coordinate ของ Model ขณะนั้น

Remarks

เป็นฟังก์ชันสำหรับเรียก Access ไปยัง Texture Coordinate ที่ถูก Use อยู่อีกที

Class CNormalsAb

CNormalsAb ทำหน้าที่เป็น เก็บข้อมูลและจัดการข้อมูลประเภท Normal Vertex ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ที่ถูกอ่านขึ้นมาจากไฟล์ข้อมูลประเภท .Map โดยเป็น Derived Class ของ CAbstract ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CNormalsAb::AddNormals

ทำหน้าที่เพิ่ม Normal Vertex เข้าไปเก็บยัง List ของ Normal Vertices ทั้งหมดของ Model นั้น

BOOL AddNormals

(

void

);

Parameters

-

Return Values

TRUE

Remarks

การทำงานภายในจะเป็นการ Increment จำนวนของ Normal Vertices ทั้งหมดขณะนั้นของ Model

CNormalsAb::DeleteDeviceObjects

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ โดยฟังก์ชันนี้จะถูกเรียกให้ทำงานในกรณีที่จะยกเลิกใช้ Object นั้นๆ และจำนวนของ Normal Vertices ยังมากกว่าศูนย์

HRESULT DeleteDeviceObjects

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ DeleteDeviceObjects ของ Normal Vertices

CNormalsAb::DeleteNormals

ทำหน้าที่ควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ ของ Normal Vertices นั้นๆ ซึ่งจะเป็นการ Decrease จำนวนของ Normal Vertices ทั้งหมดลงทีละ 1 vertex เรียกจนกว่าจะหมดจึงจะ Cleanup Memory ให้

Int DeleteNormals

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกจำนวนของ Normal Vertices ทั้งหมดของ Model ขณะนั้น

Remarks

ภายในจะเรียก DeleteDeviceObjects และ FinalCleanup เพื่อคืน Memory ของ Normal Vertices นั้นๆ

CNormalsAb::FinalCleanup

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ของ Object นั้นๆ โดยฟังก์ชันนี้จะถูกเรียกให้ทำงานในกรณีที่ต้องการให้คืน Memory ทั้งหมดทันทีที่ถูกเรียก

HRESULT FinalCleanup

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ FinalCleanup ของ Normal Vertices

CNormalsAb::GetNormals

ฟังก์ชันเพื่อ Access Vertices โดยจะคืนค่ามาเป็น Pointer ของ Normal Vertices ของ Model นั้นๆ

float* GetNormals

(

void

);

Parameters

-

Return Values

Pointer to float ที่ชี้ไปยัง Normal Vertices ของ Model ขณะนั้น

Remarks

มีไว้เพื่อบางครั้งผู้ใช้ต้องการที่จะทราบข้อมูล Normal Vertices เพื่อนำไปใช้งานบางอย่างตามต้องการ

CNormalsAb::OneTimeSceneInit

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการ Initialize Object เพียงครั้งเดียวก่อนที่จะเริ่มต้นฉากในเกม ของ Model นั้นๆ

HRESULT OneTimeSceneInit

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ OneTimeSceneInit ของ Normal Vertices

CNormalsAb::InitDeviceObjects

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการ Initialize Object ก่อนที่จะนำ Object นั้นไปใช้งาน ของ Modcl นั้นๆ

HRESULT InitDeviceObjects

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ InitDeviceObjects ของ Normal Vertices

---

Class CNormalsManage

CNormalsManage ทำหน้าที่เป็นตัวจัดการข้อมูลประเภท Normal Vertex ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ซึ่งในบางครั้งอาจจะมีการเรียกใช้งานข้อมูลของ Normal Vertices นั้นซ้ำๆ เช่น จอง Memory สำหรับ Normal Vertices เดิม CNormalsManage ก็จะมาคอยช่วยจัดการไม่ให้มีการ Load ข้อมูลขึ้นมาซ้ำๆ เพื่อประหยัดการใช้งาน Memory ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CNormalsManage::StaticInitDeviceObjects

ทำหน้าที่เป็นส่วน ควบคุมการการ Initialize Normal Vertices Object ก่อนที่จะนำ Object นั้นไปใช้งาน ของ Model นั้นๆ เพื่อควบคุมการ InitDeviceObjects ซ้ำๆของ Model

void StaticInitDeviceObjects

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ InitDeviceObjects ของ CNormals อีกที

CNormalsManage::StaticDeleteDeviceObjects

ทำหน้าที่ควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาที่ CNormals ของ Model เพื่อควบคุมการ DeleteDeviceObjects ซ้ำๆของ Model

void StaticDeleteDeviceObjects

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ DeleteDeviceObjects ของ CNormals อีกที

CNormalsManage::StaticFinalCleanUp

ทำหน้าที่เป็นส่วนควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาที่ CNormals ของ Model เพื่อควบคุมการ FinalCleanup ซ้ำๆของ Model

void StaticFinalCleanUp

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ FinalCleanup ของ CNormals อีกที

CNormalsManage::GetSize

ฟังก์ชันเพื่อรับค่าเพื่อบอกขนาดของ Normal Vertices ทั้งหมดของ Model นั้นๆ ที่ได้มีการจองเอาไว้ในตอนเริ่มต้นก่อนหน้านี้

int GetSize

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกขนาดของ Normal Vertices ทั้งหมดของ Model ที่ได้มีการจองเอาไว้ก่อนหน้านี้

Remarks

-

Class CUseNormals

CUseNormals ทำหน้าที่เป็นตัวจัดการ Access ข้อมูลประเภท Normal Vertices ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CUseNormals::Use

ฟังก์ชันเพื่อการเข้าถึงหรือ Access Normal Vertices ของ Model นั้นๆ ซึ่งจะเป็นตัว เรียกสร้าง หรือ Create Normal Vertices ที่จะถูกใช้งานซึ่งเรียกมาจาก User ในแต่ละครั้ง

void Use

(

const CString& pName

);

Parameters

pName

String ซึ่งใช้เป็น Key สำหรับการ Search เพื่อที่จะเรียกใช้ Use กับ Normal Vertices นั้นๆ

Return Values

-

Remarks

ทำหน้าที่เป็น Reference Count Increment สำหรับการ Access ขอใช้งาน Normal Vertices

CUseNormals::UnUse

ฟังก์ชันเพื่อยกเลิกการเข้าถึงหรือ Access Normal Vertices ของ Model นั้นๆ

void UnUse

(

void

);

Parameters

-

Return Values

-

Remarks

ทำหน้าที่เป็น Reference Count Decrement สำหรับยกเลิกการ Access ใช้งาน Normal Vertices

CUseNormals::Get

ฟังก์ชันเพื่อ Access Normal Vertices โดยจะคืนค่ามาเป็น Pointer ของ Normal Vertices ของ Model นั้นๆ ที่ถูกเรียก Use ไว้ก่อนหน้านี้

float* Get

(

void

);

Parameters

-

Return Values

Pointer to float ที่ชี้ไปยัง Normal Vertices ของ Model ขณะนั้น

Remarks

เป็นฟังก์ชันสำหรับเรียก Access ไปยัง Normal Vertices ที่ถูก Use อยู่อีกที

Class CTexture

CTexture ทำหน้าที่เป็น เก็บข้อมูลและจัดการข้อมูลประเภท Texture ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ที่ถูกอ่านขึ้นมาจากไฟล์ข้อมูลประเภท .Map โดยเป็น Derived Class ของ CAbstract ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CTexture::AddTexture

ทำหน้าที่เพิ่ม Texture เข้าไปเก็บยัง List ของ Texture ทั้งหมดของ Model นั้น

BOOL AddTexture

(

void

);

Parameters

-

Return Values

TRUE

Remarks

การทำงานภายในจะเป็นการ Increment จำนวนของ Texture ทั้งหมดขณะนั้นของ Model

CTexture::DeleteDeviceObjects

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ โดยฟังก์ชันนี้จะถูกเรียกให้ทำงานในกรณีที่จะยกเลิกใช้ Object นั้นๆ และจำนวนของ Texture ยังมากกว่าศูนย์

HRESULT DeleteDeviceObjects

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ DeleteDeviceObjects ของ Texture

CTexture::DeleteTexture

ทำหน้าที่ควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ ของ Texture นั้นๆ ซึ่งจะเป็นการ Decrease จำนวนของ Texture ทั้งหมดลงทีละ 1 texture เรียกจนกว่าจะหมดจึงจะ Cleanup Memory ให้

int DeleteTexture

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกจำนวนของ Texture ทั้งหมดของ Model ขณะนั้น

Remarks

ภายในจะเรียก DeleteDeviceObjects และ FinalCleanup เพื่อคืน Memory ของ Texture นั้นๆ

CTexture::FinalCleanup

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาไว้ของ Object นั้นๆ โดยฟังก์ชันนี้จะถูกเรียกให้ทำงานในกรณีที่ต้องการให้คืน Memory ทั้งหมดทันทีที่ถูกเรียก

HRESULT FinalCleanup

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ FinalCleanup ของ Texture

CTexture::GetTexture

ฟังก์ชันเพื่อ Access Vertices โดยจะคืนค่ามาเป็น Pointer ของ Texture ของ Model นั้นๆ

float* GetTexture

(

void

);

Parameters

-

Return Values

Pointer to float ที่ชี้ไปยัง Texture ของ Model ขณะนั้น

Remarks

มีไว้เพื่อบางครั้งผู้ใช้ต้องการที่จะทราบข้อมูล Texture เพื่อนำไปใช้งานบางอย่างตามต้องการ

CTexture::OneTimeSceneInit

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการ Initialize Object เพียงครั้งเดียวก่อนที่จะเริ่มต้นฉากในเกม ของ Model นั้นๆ

HRESULT OneTimeSceneInit

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ OneTimeSceneInit ของ Texture

CTexture::InitDeviceObjects

ทำหน้าที่เป็นส่วน Implement ต่อ เพื่อควบคุมการการ Initialize Object ก่อนที่จะนำ Object นั้นไปใช้งาน ของ Model นั้นๆ

HRESULT InitDeviceObjects

(

void

);

Parameters

-

Return Values

S_OK

Remarks

เป็นส่วน Implement ต่อจาก CAbstract เพื่อควบคุมการ InitDeviceObjects ของ Texture

Class CTextureManage

CTextureManage ทำหน้าที่เป็นตัวจัดการข้อมูลประเภท Texture ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ซึ่งในบางครั้งอาจจะมีการเรียกใช้งานข้อมูลของ Texture นั้นซ้ำๆ เช่นจอง Memory สำหรับ Texture เดิม CTextureManage ก็จะมาคอยช่วยจัดการไม่ให้มีการ Load ข้อมูลขึ้นมาซ้ำๆ เพื่อประหยัดการใช้งาน Memory ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CTextureManage::StaticInitDeviceObjects

ทำหน้าที่เป็นส่วน ควบคุมการการ Initialize Texture Object ก่อนที่จะนำ Object นั้นไปใช้งาน ของ Model นั้นๆ เพื่อควบคุมการ InitDeviceObjects ซ้ำๆของ Model

void StaticInitDeviceObjects

(

void

);

Paramcters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ InitDeviceObjects ของ CTexture อีกที

CTextureManage::StaticDeleteDeviceObjects

ทำหน้าที่ควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาที่ CTexture ของ Model เพื่อควบคุมการ DeleteDeviceObjects ซ้ำๆของ Model

void StaticDeleteDeviceObjects

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ DeleteDeviceObjects ของ CTexture อีกที

CTextureManage::StaticFinalCleanUp

ทำหน้าที่เป็นส่วนควบคุมการการคืน Memory ในส่วนที่ได้มีการจองเอาที่ CTexture ของ Model เพื่อควบคุมการ FinalCleanup ซ้ำๆของ Model

void StaticFinalCleanUp

(

void

);

Parameters

-

Return Values

-

Remarks

เป็นส่วน Implement เพื่อควบคุมการ FinalCleanup ของ CTexture อีกที

CTextureManage::GetSize

ฟังก์ชันเพื่อรับค่าเพื่อบอกขนาดของ Texture ทั้งหมดของ Model นั้นๆ ที่ได้มีการจองเอาไว้ในตอนเริ่มต้นก่อนหน้านี้

int GetSize

(

void

);

Parameters

-

Return Values

Integer ซึ่งบอกขนาดของ Texture ทั้งหมดของ Model ที่ได้มีการจองเอาไว้ก่อนหน้านี้

Remarks

-

Class CUseTexture

CUseTexture ทำหน้าที่เป็นตัวจัดการ Access ข้อมูลประเภท Texture ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ซึ่งประกอบไปด้วยฟังก์ชันต่างๆดังนี้

CUseTexture::Use

ฟังก์ชันเพื่อการเข้าถึงหรือ Access Texture ของ Model นั้นๆ ซึ่งจะเป็นตัว เรียกสร้าง หรือ Create Texture ที่จะถูกใช้งานซึ่งเรียกมาจาก User ในแต่ละครั้ง

void Use

(

const CString& pName

);

Parameters

pName

String ซึ่งใช้เป็น Key สำหรับการ Search เพื่อที่จะเรียกใช้ Use กับ Texture นั้นๆ

Return Values

-

Remarks

ทำหน้าที่เป็น Reference Count Increment สำหรับการ Access ขอใช้งาน Texture

CUseTexture::UnUse

ฟังก์ชันเพื่อยกเลิกการเข้าถึงหรือ Access Texture ของ Model นั้นๆ

void UnUse

(

void

);

Parameters

-

Return Values

-

Remarks

ทำหน้าที่เป็น Reference Count Decrement สำหรับยกเลิกการ Access ใช้งาน Texture

CUseTexture::Get

ฟังก์ชันเพื่อ Access Texture โดยจะคืนค่ามาเป็น Pointer ของ Texture ของ Model นั้นๆ ที่ถูกเรียก Use ไว้ก่อนหน้านี้

float* Get

(

void

);

Parameters

-

Return Values

Pointer to float ที่ชี้ไปยัง Texture ของ Model ขณะนั้น

Remarks

เป็นฟังก์ชันสำหรับเรียก Access ไปยัง Texture ที่ถูก Use อยู่อีกที

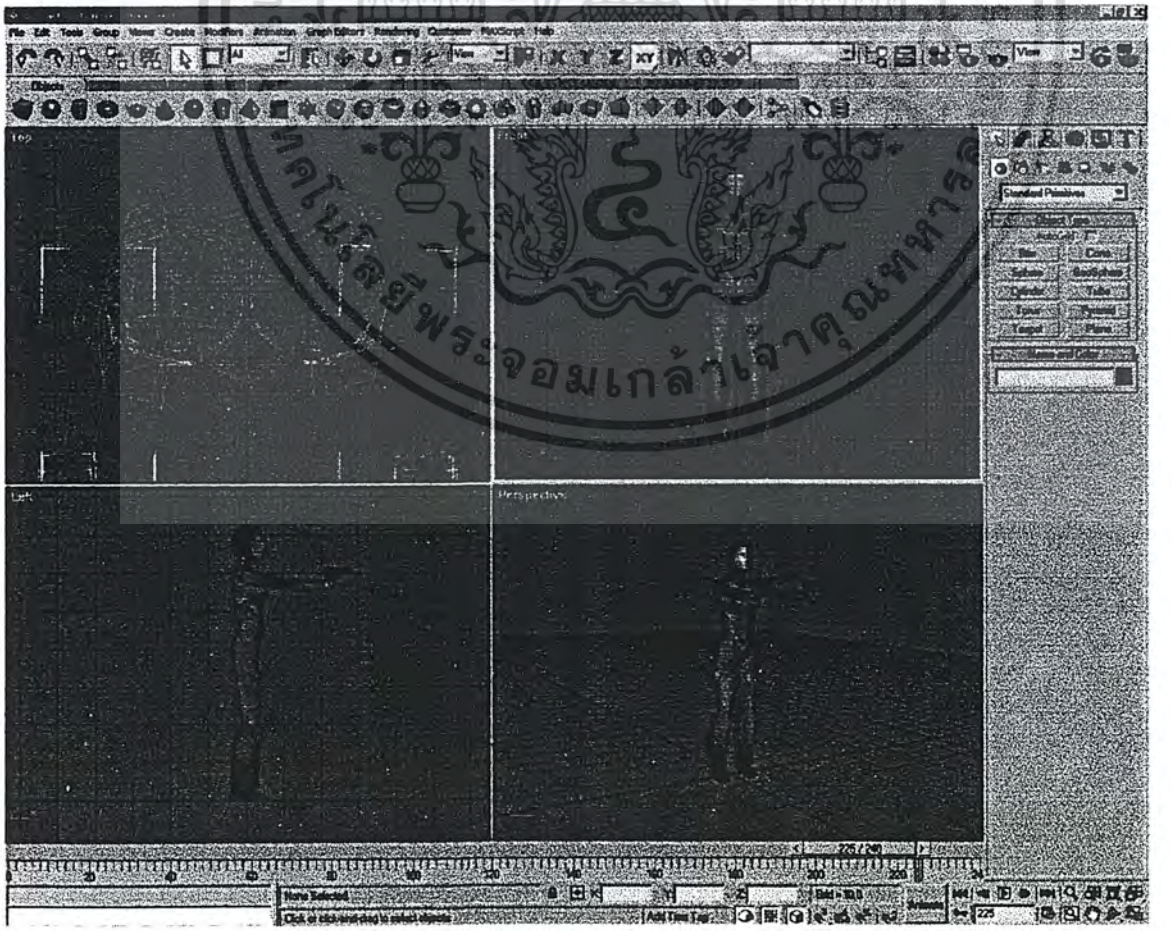
บทที่ 5
การใช้งานเกมเอนจินต์ (Using Game Engine)

5.1 ขั้นตอนการจัดเตรียมข้อมูลเพื่อทำแอนิเมชันของตัวละคร

การเคลื่อนไหวในรูปแบบเวอร์เท็กซ์บเลนดิง ข้อมูลของวัตถุที่อ่านหรือโหลดขึ้นมาได้นั้นจะเป็นลักษณะของวัตถุที่อยู่นิ่งกับที่ ข้อมูลที่จัดเก็บจะเก็บเป็นเวอร์เท็กซ์ของวัตถุที่อยู่นิ่งกับที่ โดยแบ่งเป็นช่วงของการเคลื่อนไหวในลักษณะต่างๆ การทำการเคลื่อนไหวจึงจะต้องอาศัยเทคนิคการอินเตอร์โพเลตมาช่วยทำให้การทำการเคลื่อนไหวมีความราบรื่น จึงเป็นหน้าที่ของผู้พัฒนาเอง โดยมีหลักการและวิธีการที่จะต้องสร้างและจัดเตรียมข้อมูลการเคลื่อนไหวให้มีความเหมาะสมกับลักษณะท่าทางที่แตกต่างกันของตัวละคร ที่มีกลไกในการเคลื่อนไหวของวัตถุนั้นเองโดยอาศัยข้อมูลการเคลื่อนไหวของวัตถุที่เตรียมไว้ การจัดเก็บข้อมูลการเคลื่อนไหวนี้จึงเป็นขั้นตอนหนึ่งในการพัฒนาแอนิเมชันของตัวละครในเกมสามมิติ

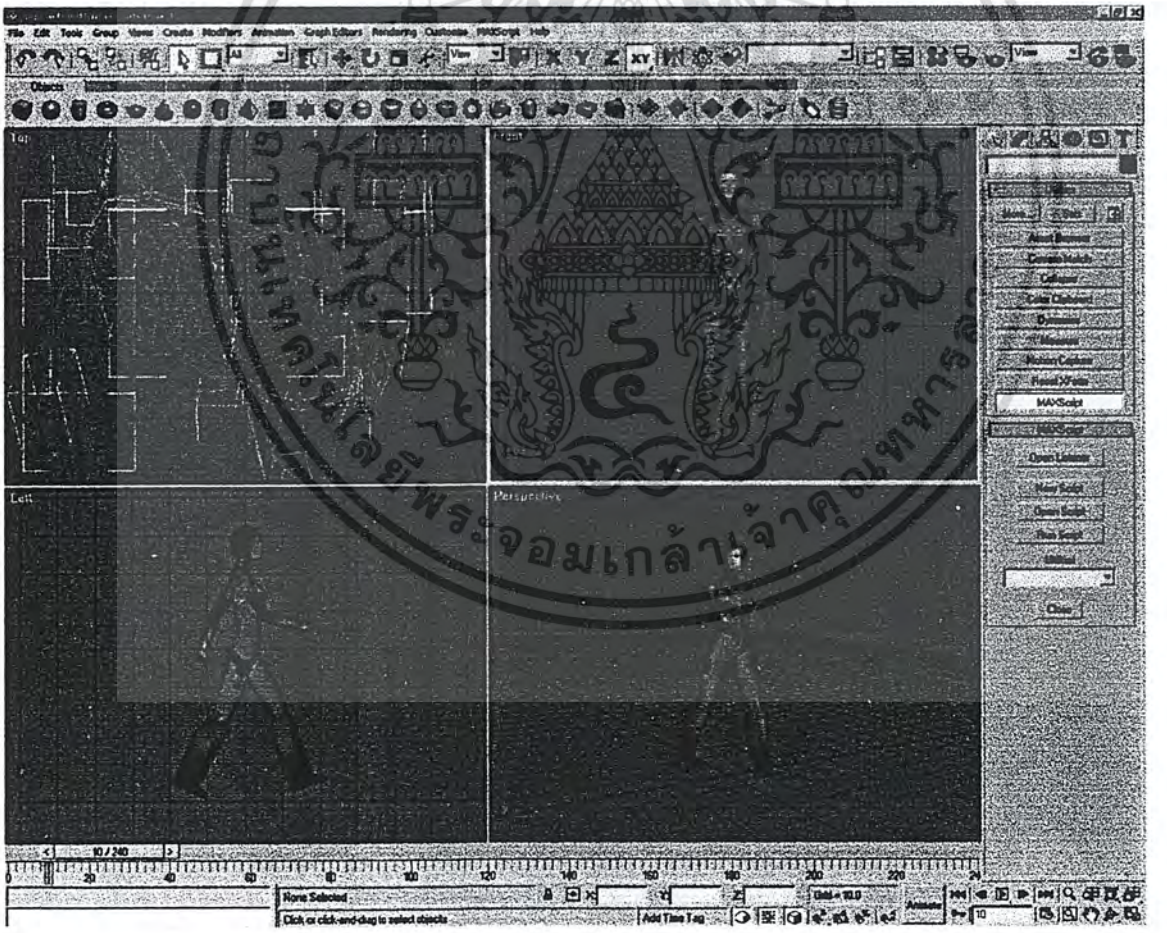
5.1.1 เอ็กพอร์ตแอนิเมชันด้วย Maxscript ของ 3ds-max

สร้างแอนิเมชันของตัวละครด้วยโปรแกรม 3ds-max ก่อนที่จะทำการเอ็กพอร์ต ในกริยาท่าทางต่างๆ ของตัวละคร



รูปที่ 5-1 ทำแอนิเมชันของตัวละครด้วย 3ds-max

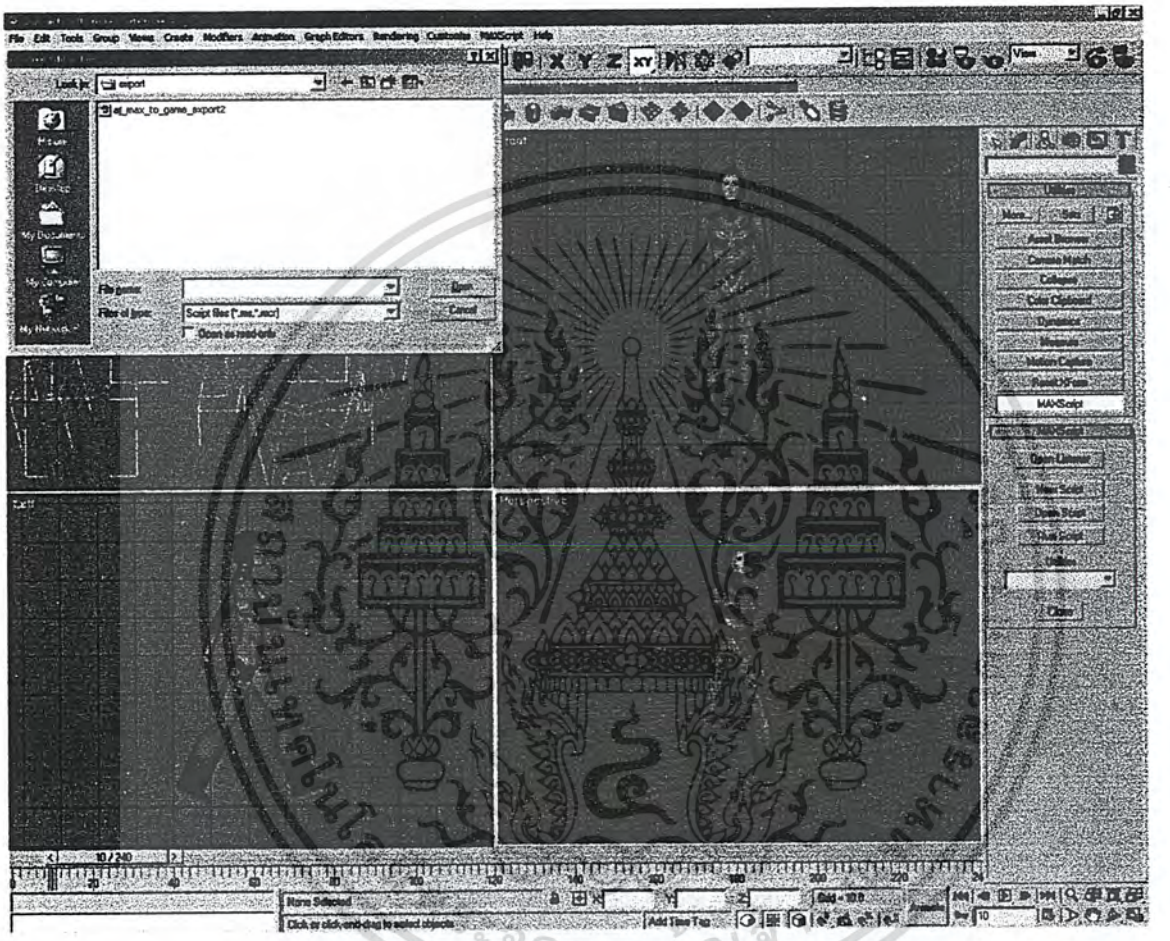
เมื่อได้แอนิเมชันของตัวละครแล้วจะเป็นขั้นตอนการเอ็กพอร์ตไฟล์ฟอร์แมตของ 3ds-max เพื่อให้ได้ไฟล์ฟอร์แมต 5 แบบที่มีส่วนขยายดังนี้ mac, maf, mai, map และ mau ซึ่งเป็นฟอร์แมตที่จะสามารถนำไปใช้ในการพัฒนาเกม เลือกเฟรมที่จะทำการเอ็กพอร์ตแล้ว ใช้เครื่องมือ Max Script ในแท็ปยูทิลิตี้  ของ 3ds-max ซึ่งเป็น Script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



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

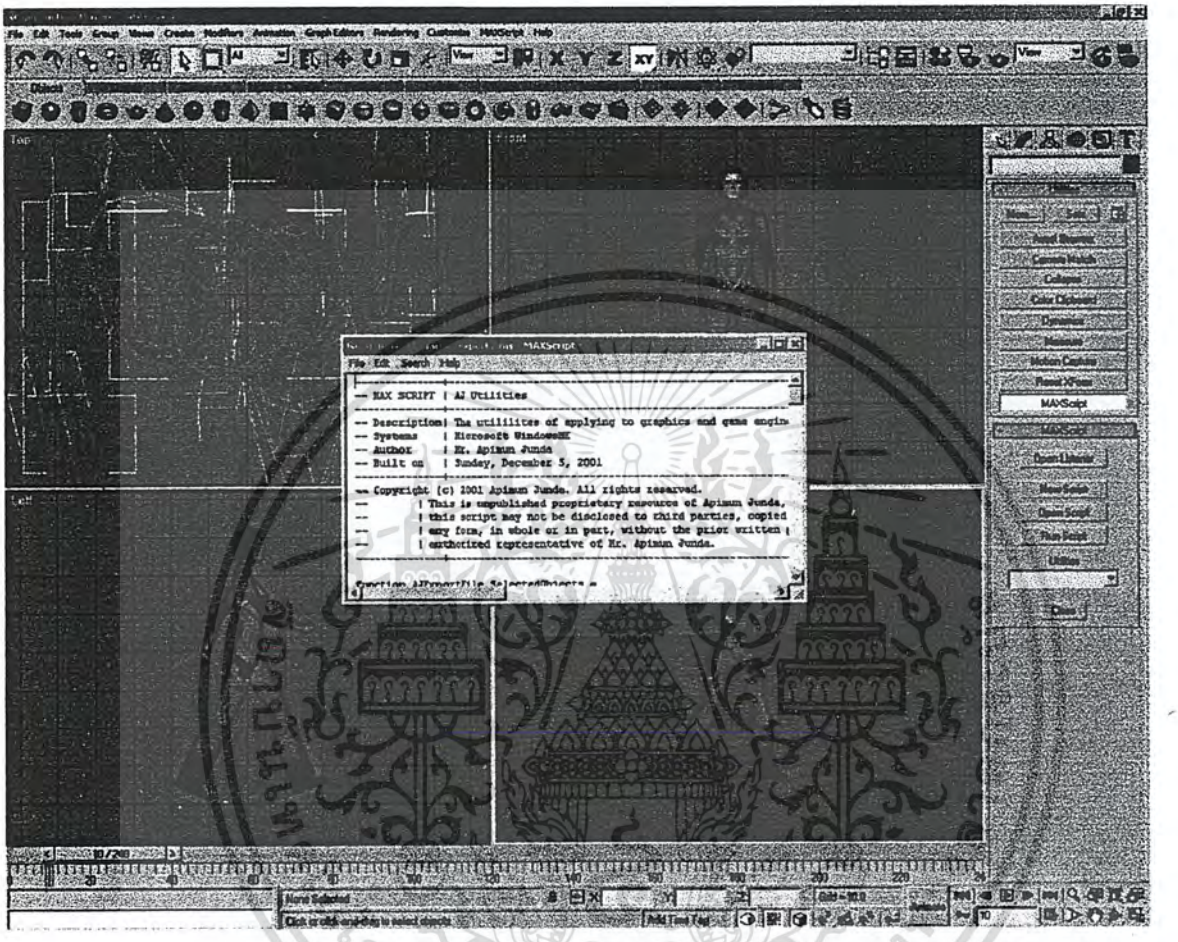
5.1.1.1 เปิดสคริปต์ (Open script) ที่ใช้ในการเอ็กพอร์ต

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

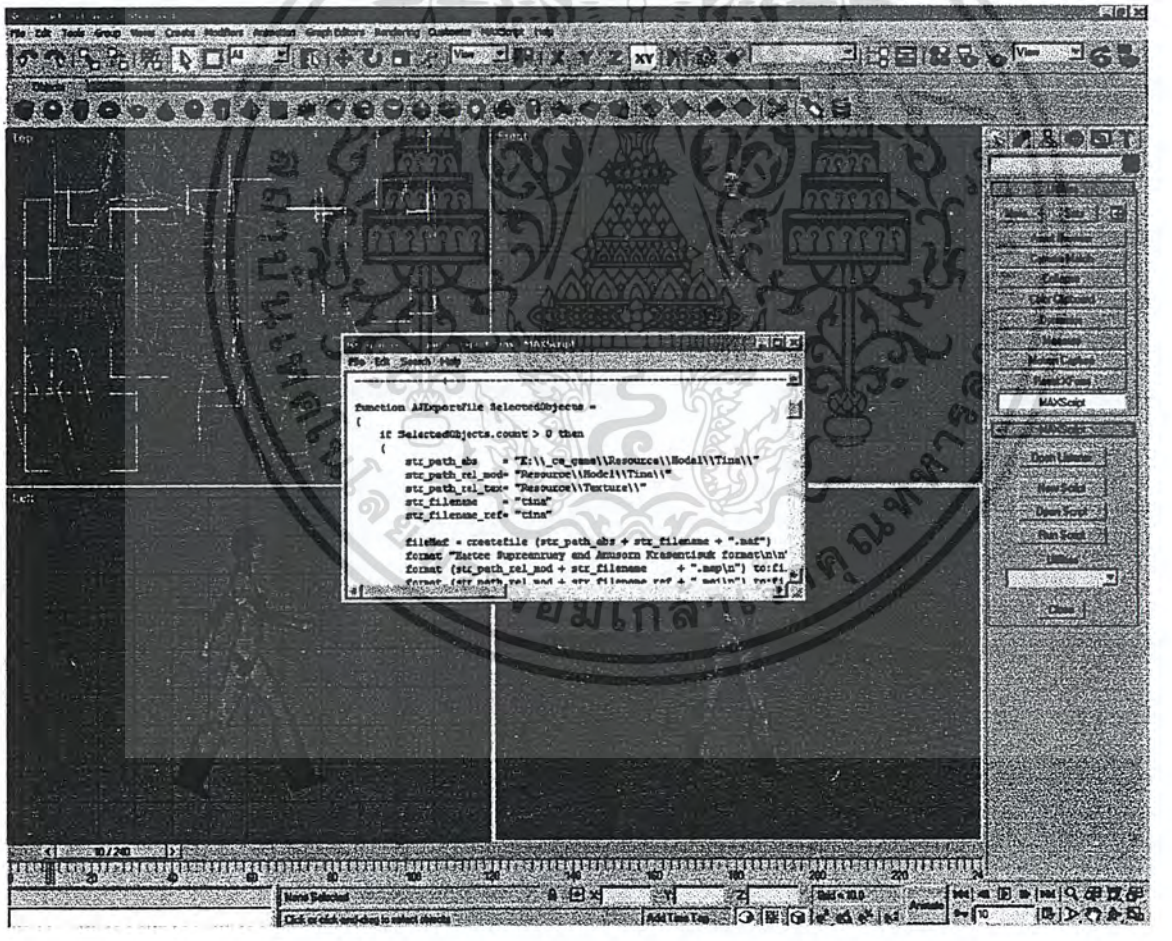
ทำการแก้ไข Path และชื่อ ไฟล์ที่จะทำการเอ็กพอร์ต ใน MAX Script

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

str_path คือ Path ของไฟล์ที่จะทำการเอ็กพอร์ต

str_filename คือชื่อของไฟล์ที่จะทำการเอ็กพอร์ต

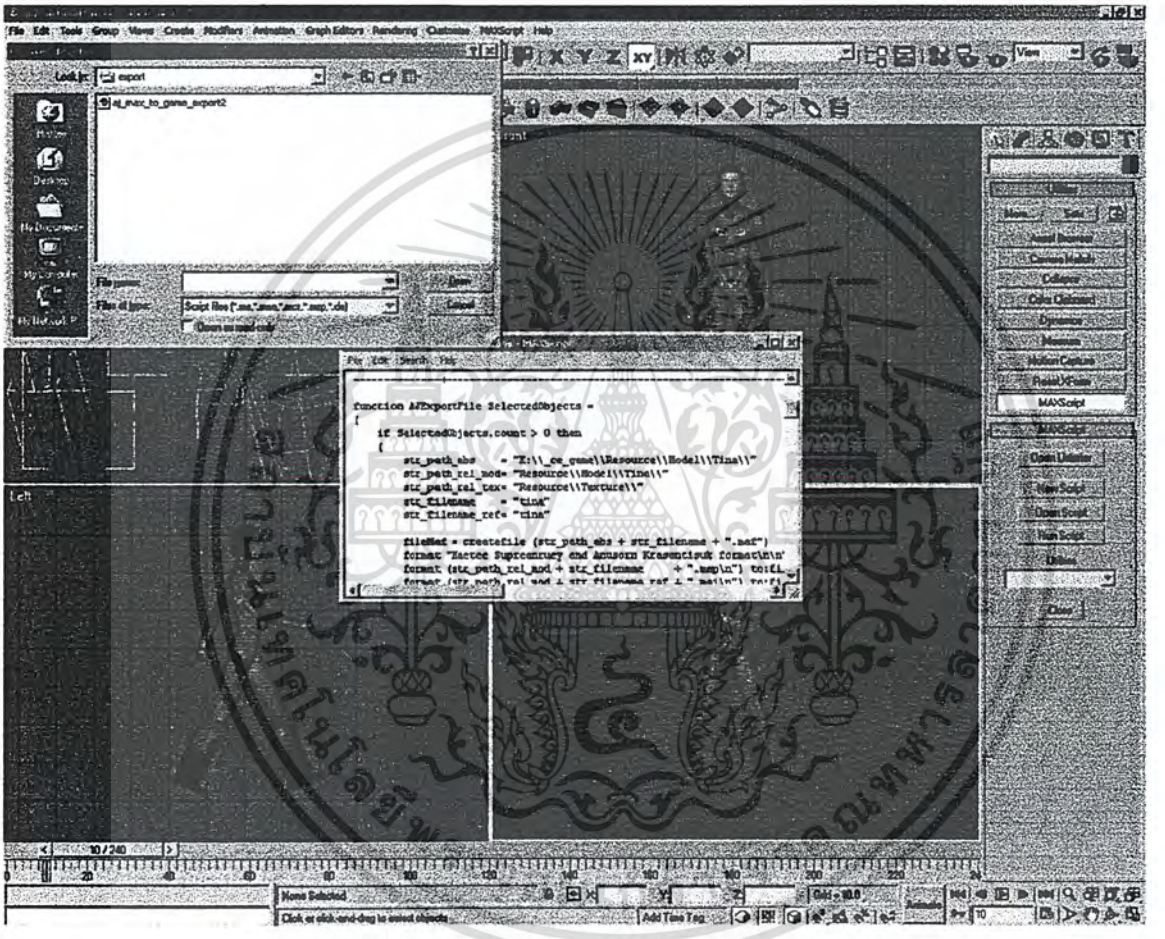
จากนั้นทำการบันทึก MAX Script ไฟล์



รูปที่ 5-5 แก้ไข Path และชื่อไฟล์ใน MAX Script

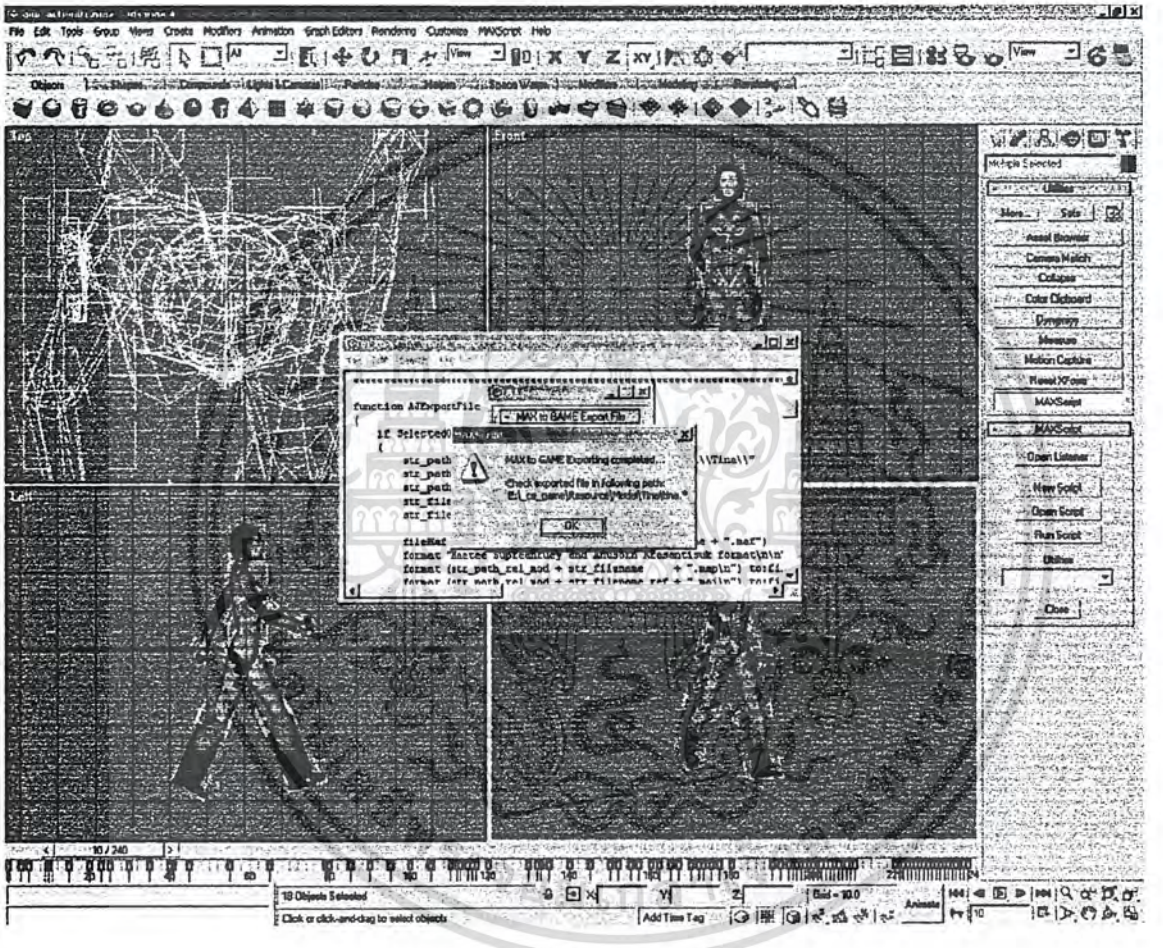
5.1.1.3 รันสคริปต์ (Run script) ที่ใช้ในการเอ็กพอร์ต

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

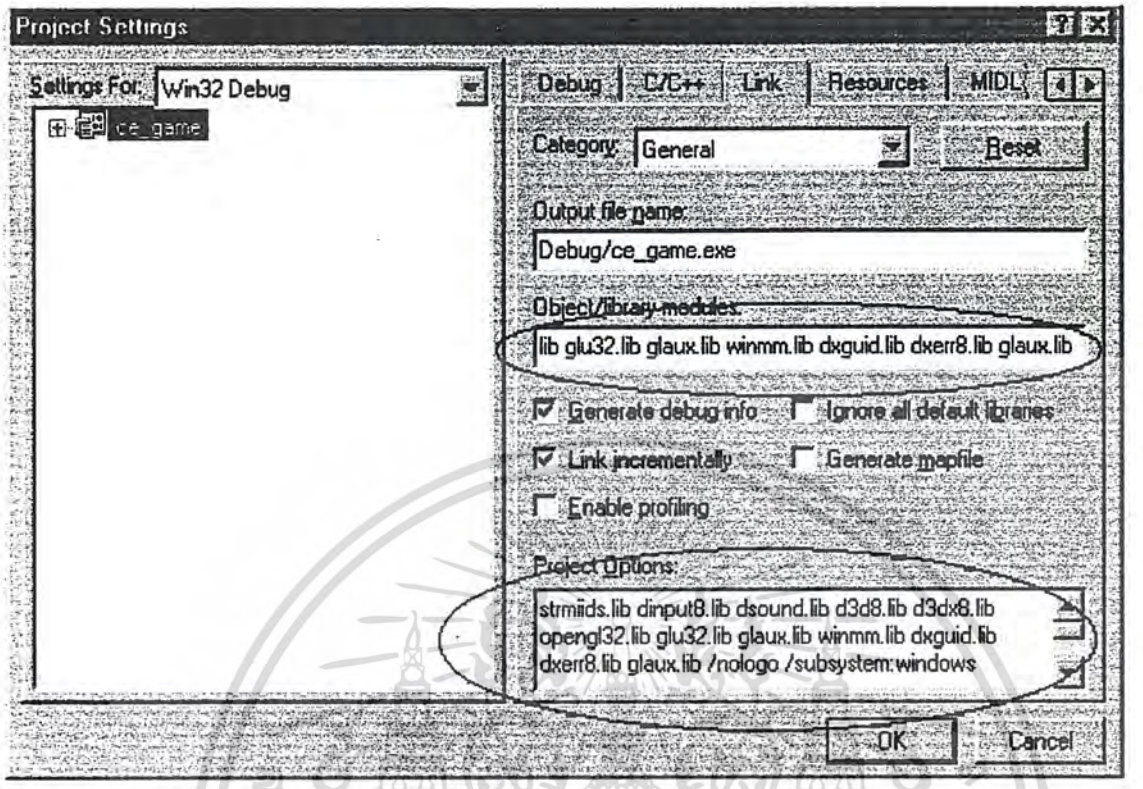
เราสามารถควบคุมการเคลื่อนไหวของตัวละครโดยการสร้างไฟล์ประเภท .MAA ซึ่งภายในจะประกอบไปด้วยรูปแบบที่กำหนดไว้ดังนี้

```
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

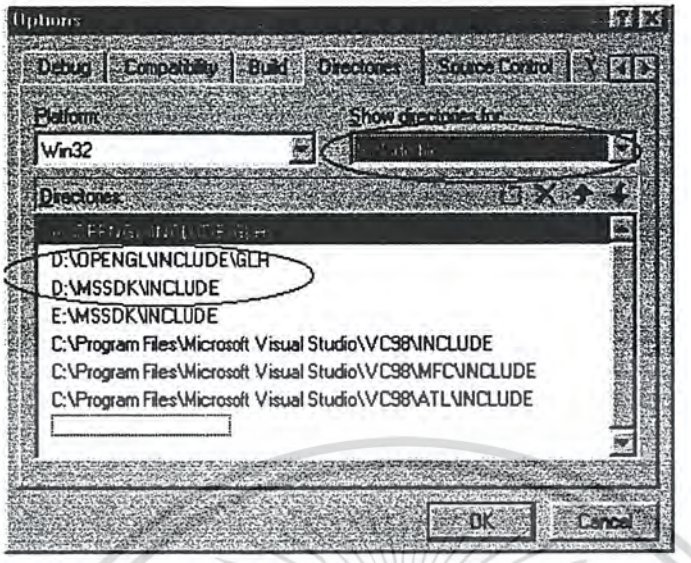
หลังจากที่ได้เตรียมข้อมูลต่างๆ สำหรับการสร้างเกมแล้ว ขั้นตอนต่อไปคือการสร้างเกม ในการสร้างเกมจะใช้เฟรมเวิร์ค ซึ่งประกอบไปด้วยคลาสต่างๆ ที่เตรียมไว้ให้ผู้ใช้สามารถสืบทอด หรือเรียกใช้ฟังก์ชันได้ ในการใช้เฟรมเวิร์คสร้างเกมนั้น มีขั้นตอนดังนี้



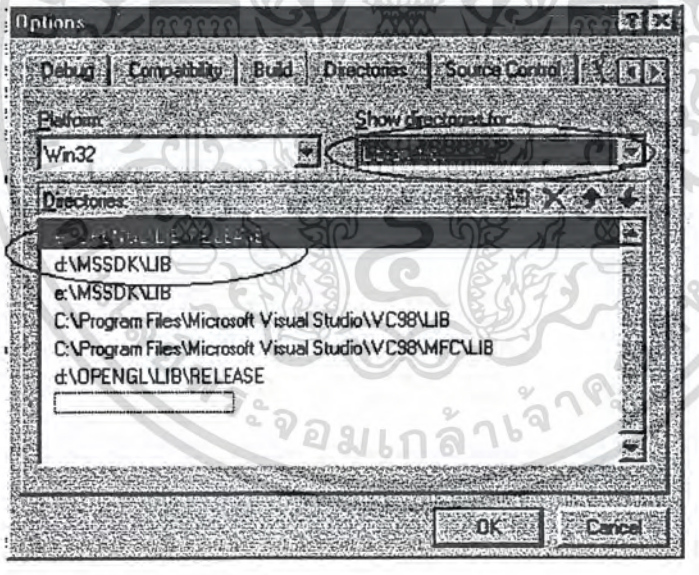
รูปที่ 5-8 ไฟล์ไลบรารีที่ต้องการ

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)

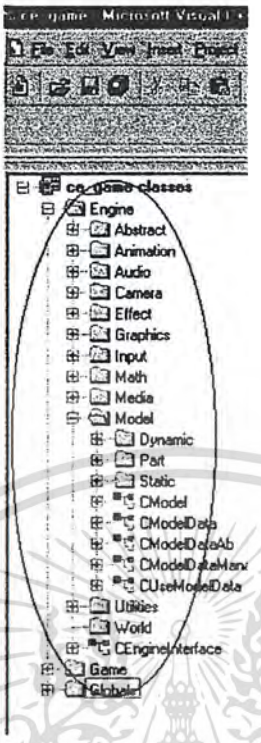


รูปที่ 5-9 Directory ของ Include Files



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

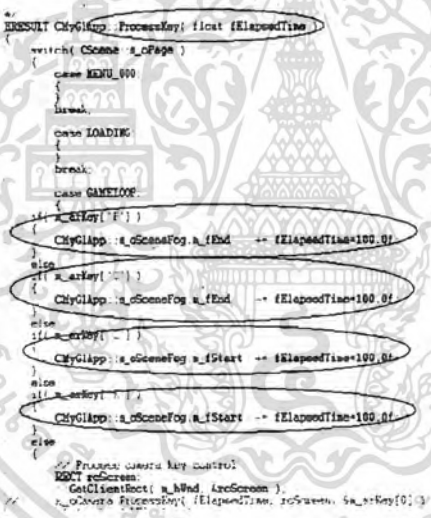
```
HRESULT CMyGlApp::InitDeviceObjects(GLvoid)
{
    ShowLoading(0.0f,"Wait for loading");

    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

รูปที่ 5-14 แสดงตัวอย่าง Source Code ส่วน Render

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



รูปที่ 5-15 แสดงตัวอย่าง Source Code ในส่วนการกดปุ่ม

รูปที่ 5-16 แสดงตัวอย่าง Source Code ในส่วนประมวลผล

บทที่ 6
การพัฒนาเกมเพื่อทดสอบเกมเอนจินต์

6.1 ประเภทเกมที่พัฒนา

สำหรับเกมที่พัฒนานี้เป็นเกมแนวแอ็คชั่นชู้ตติ้ง โดยเอามุมมองบุคคลที่1 มาใช้ โดยรายละเอียดของเกมคือจำลองผู้เล่นเป็นเสมือนคนขับยานยิงยานลำอื่นๆที่ลอยอยู่ ภายในควงดาว ด้านซ้ายล่างจะแสดงเรดาห์ของตำแหน่งยานผู้เล่น ตรงกลางเป็นจำนวนยานผู้เล่นที่เหลืออยู่ ด้านขวาเป็นอาวุธที่มี จอจะใช้การ Render ของ Engine3 มิติ และเมนูด้านขอบล่างเป็นการ Render แบบ 2 มิติ การเล่นเกมจะใช้ปุ่ม 8,4,6,2 ในการควบคุมยาน การหันหรือการเลี้ยว เมาส์ปุ่มซ้ายใช้ยิงปืนและปุ่มขวาใช้ยิงลูกระเบิด

รูปที่6.1 ตัวอย่างเกม

รูปที่6.2 ตัวอย่างเกม

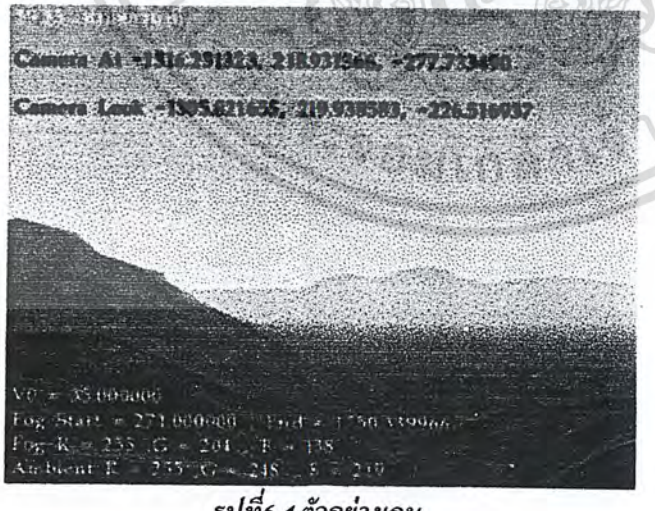
6.2 การนำเอนจินต์มาพัฒนาเกม

ในเกม ตัวอย่างนี้ ได้นำเอ็นจิ้นส่วนโมเดลเข้ามาใช้ในการโหลดข้อมูล การเรียบเรียงโครงสร้างข้อมูล การจัดการตำแหน่งวัตถุ การเข้าถึง Data member ต่างๆของ โมเดลดัง และสุดท้ายก็คือการ Render โมเดลออกมาตามที่ต้องการ ดั่งที่กล่าวไว้แล้วในการพัฒนา Class Model บทก่อนหน้านี้ และยังมีการนำเอาโมเดลที่ได้ลองพัฒนาเข้ามาใช้ด้วย และการเคลื่อนที่ของตัวยานก็คือกานำเอาการควบคุมกล้องมาใช้นั่นเอง

6.3 ผลการทดสอบ

ผลการทดสอบ โปรแกรมสามารถดึงข้อมูลไฟล์โมเดลที่เรา Export ออกมาจาก 3D Studio ได้สำเร็จ โปรแกรมสามารถแสดงผลได้ถูกต้อง

รูปที่6.3 ตัวอย่างการโหลดโมเดล



รูปที่6.4 ตัวอย่างเกม

บทที่ 7
บทวิจารณ์และสรุป

7.1 สรุปผลการวิจัย

- จากการทดสอบพบว่าการ Render ของโปรแกรมทำได้ค่อนข้างดี ตัวโมเดลที่ถูก Render ไม่มีส่วนใดเสียหาย ความเร็วในการทำอนิเมชั่นอยู่ในขั้นน่าพอใจ
- ข้อผิดพลาดที่พบคือการ Render ผิดตำแหน่ง การ Render ที่เกิดภาพโมเดลแตกออก และการ Map Texture ที่ผิดพลาดไปจากโมเดลใน 3D Studio ซึ่งต้องทำการแก้ไขในบางส่วนต่อไป

7.2 แนวทางในการพัฒนาต่อ

แนวทางต่อไปที่จะทำคือต้องแก้ไขข้อผิดพลาดในการ Render เสียก่อน หลังจากนั้นก็จะเป็นการลองสร้าง Model ที่มีรายละเอียดสูงเพื่อใช้ในโปรแกรมและทดสอบความเร็วพร้อมความเป็นไปได้ในการนำมาใช้ในเกมนอกจากจะนำไปใช้ในมูฟวี่เท่านั้น และต้องการจะนำเอ็นจิ้นที่มีไปลองพัฒนาในเกมอื่นๆที่ไม่ใช่เกมแอ็คชั่นเท่านั้นด้วย

ภาคผนวก

ภาคผนวก ก

การติดตั้ง 3D Studio MAX บนระบบปฏิบัติการวินโดว์

1. เตรียมการติดตั้ง

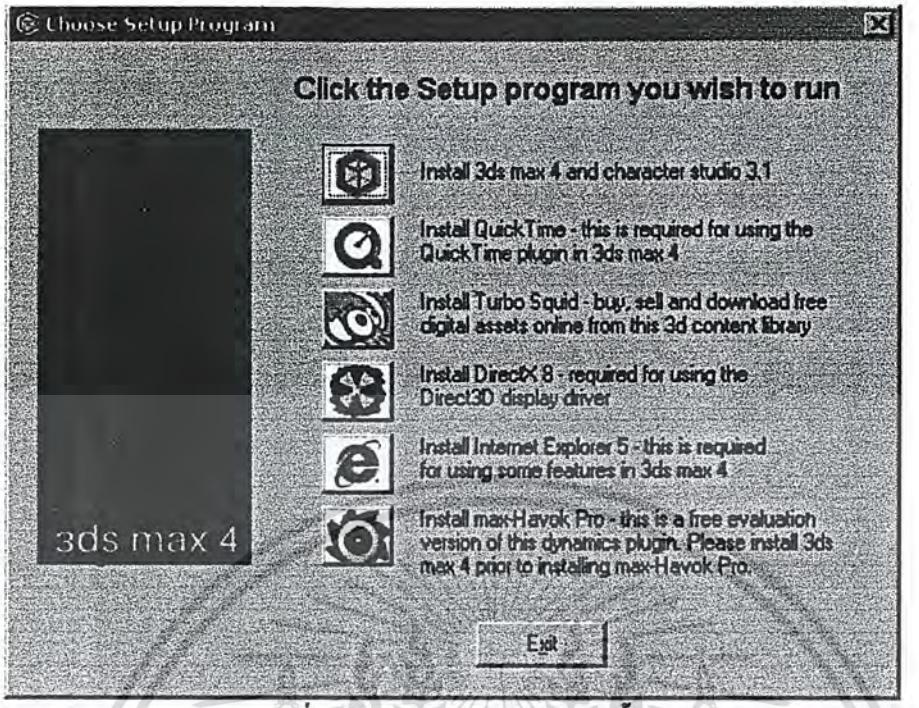
ในการติดตั้งบน Windows ทั้ง Windows 98/2000/NT/ME สามารถติดตั้งได้ โดยก่อนอื่นต้องสำรวจความพร้อมของทั้งฮาร์ดแวร์ และซอฟต์แวร์ต่างๆ ดังนี้.

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

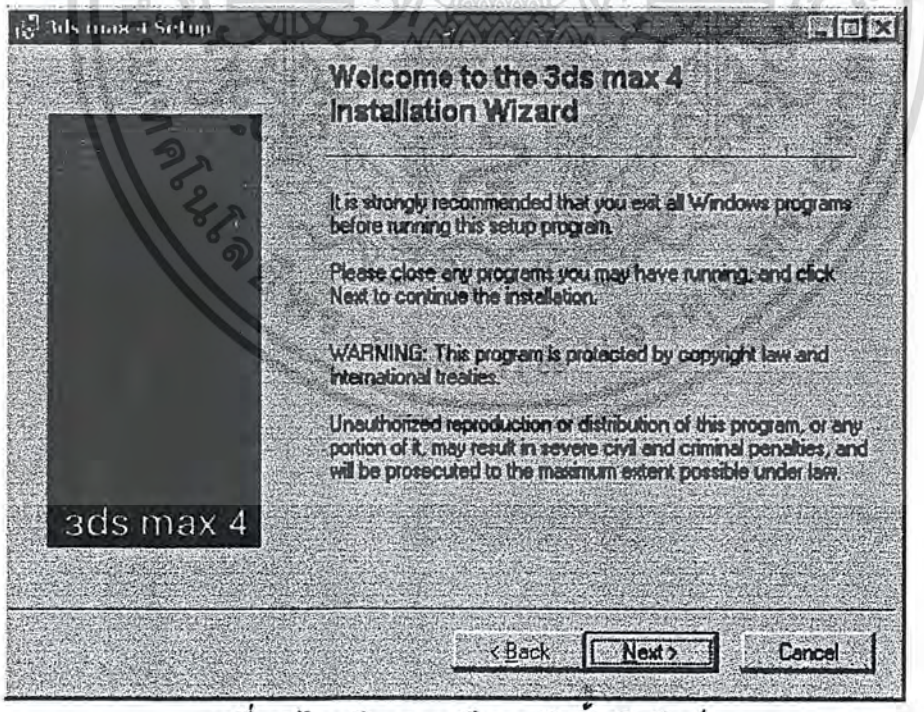
สำหรับขั้นตอนในการติดตั้งมีรายละเอียดดังนี้

2.1 นำแผ่นติดตั้งของ Visual C++ 6.0 เข้าไปในซีดีรอมไดรว์ รอสักครู่มันจะเรียกโปรแกรมเซตอัพโดยอัตโนมัติ



รูปที่ ก-1 ไดอะล็อกการเลือกชนิดติดตั้ง

2.2 คลิกเลือกไอคอนการติดตั้งโปรแกรม 3D Studio MAX 4



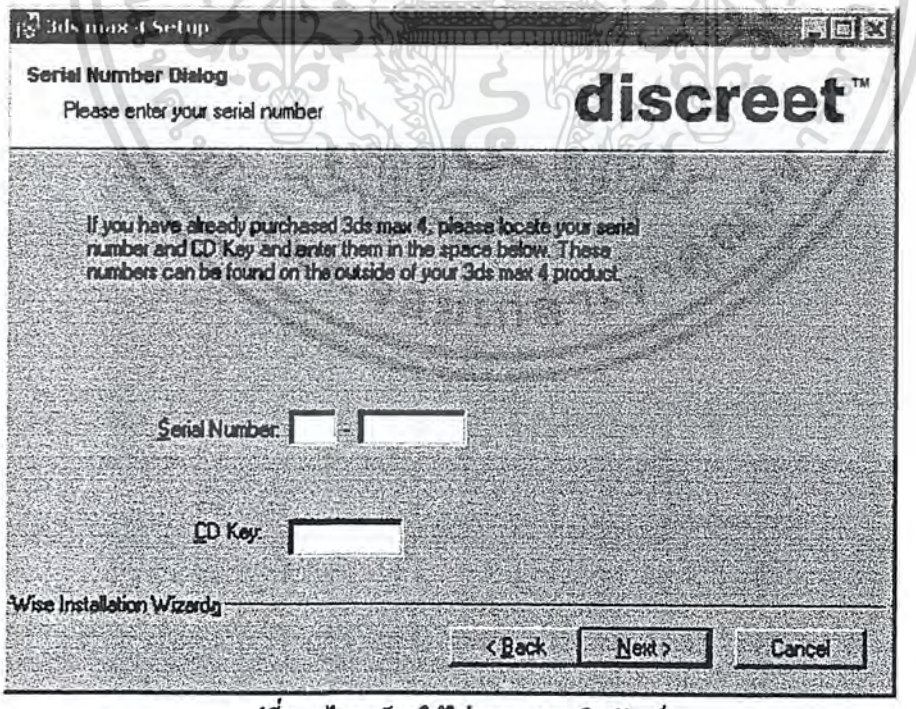
รูปที่ ก-2 ไดอะล็อกรายละเอียดการติดตั้งผลิตภัณฑ์

2.3 คลิกเลือก  เพื่อติดตั้งโปรแกรมจะขึ้นไดอะล็อกเกี่ยวกับข้อตกลงการใช้ผลิตภัณฑ์นี้ คลิกที่หัวข้อ I accept

รูปที่ ก-3 ไดอะล็อกเกี่ยวกับข้อตกลงการใช้ผลิตภัณฑ์

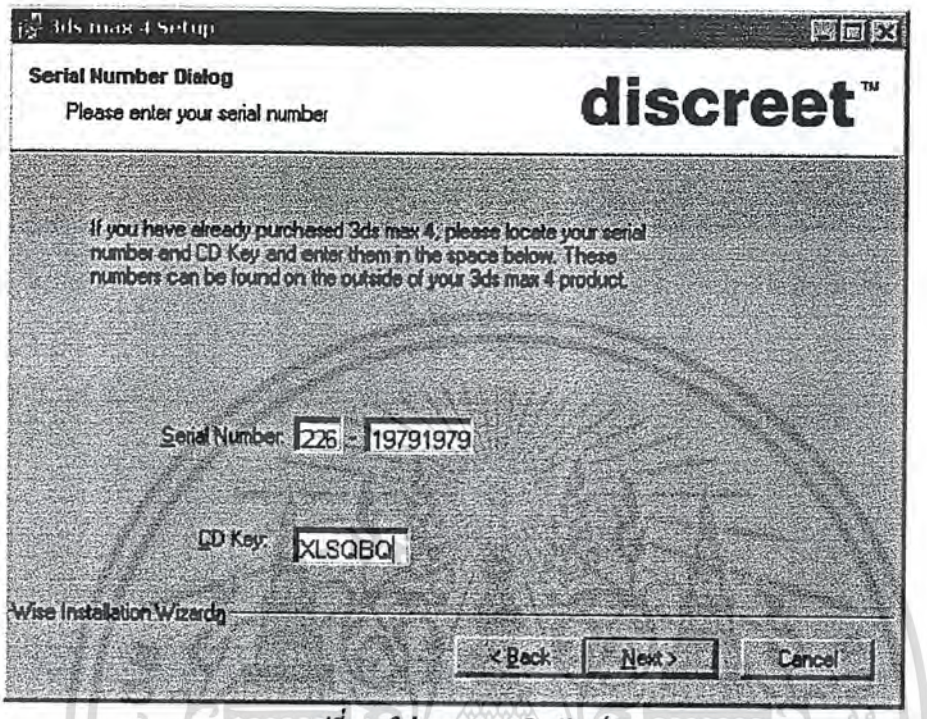
2.4 จากนั้นคลิกปุ่ม Next > อีกครั้ง

2.5 พิมพ์ข้อมูลหมายเลขของผลิตภัณฑ์นี้



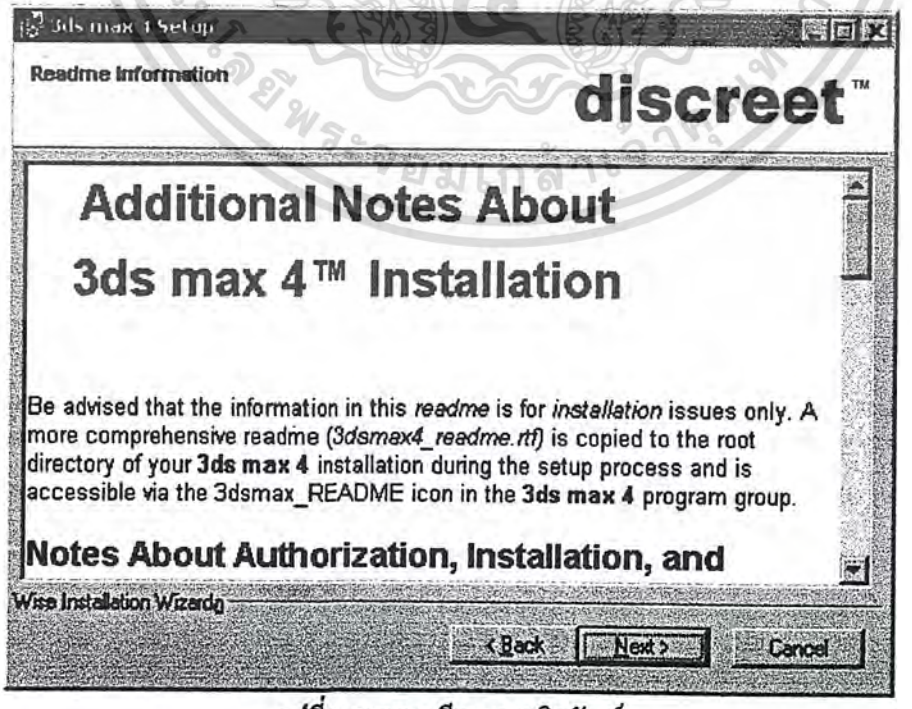
รูปที่ ก-4 ไดอะล็อกให้ใส่หมายเลขผลิตภัณฑ์

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



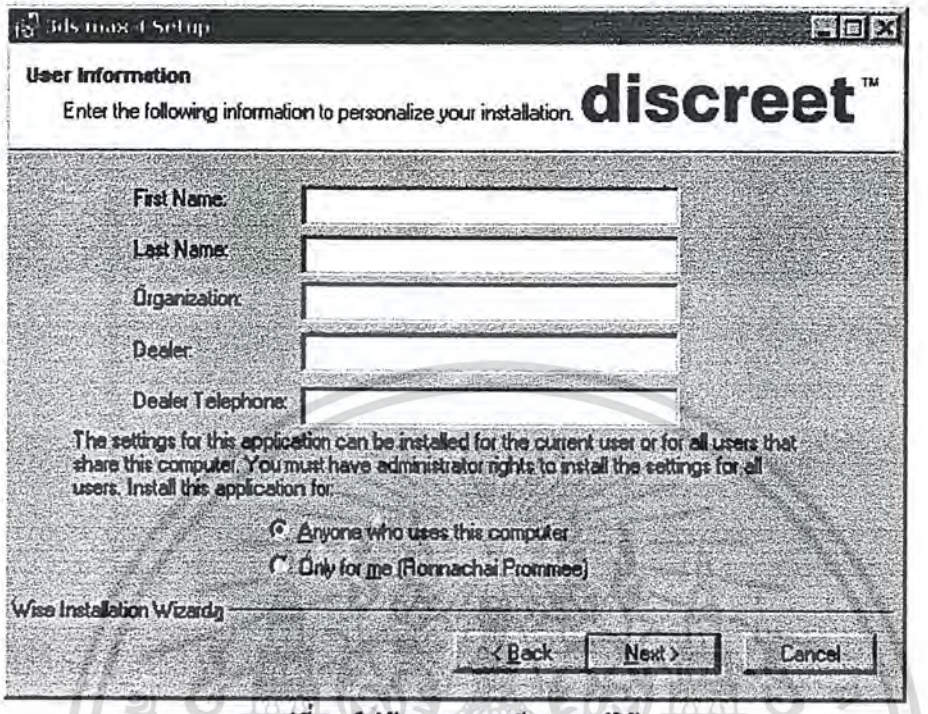
รูปที่ ก-5 ใส่หมายเลขผลิตภัณฑ์

2.7 จากนั้นจะขึ้นไดอะล็อกข้อมูลรายละเอียดของผลิตภัณฑ์ แล้วคลิกปุ่ม Next >



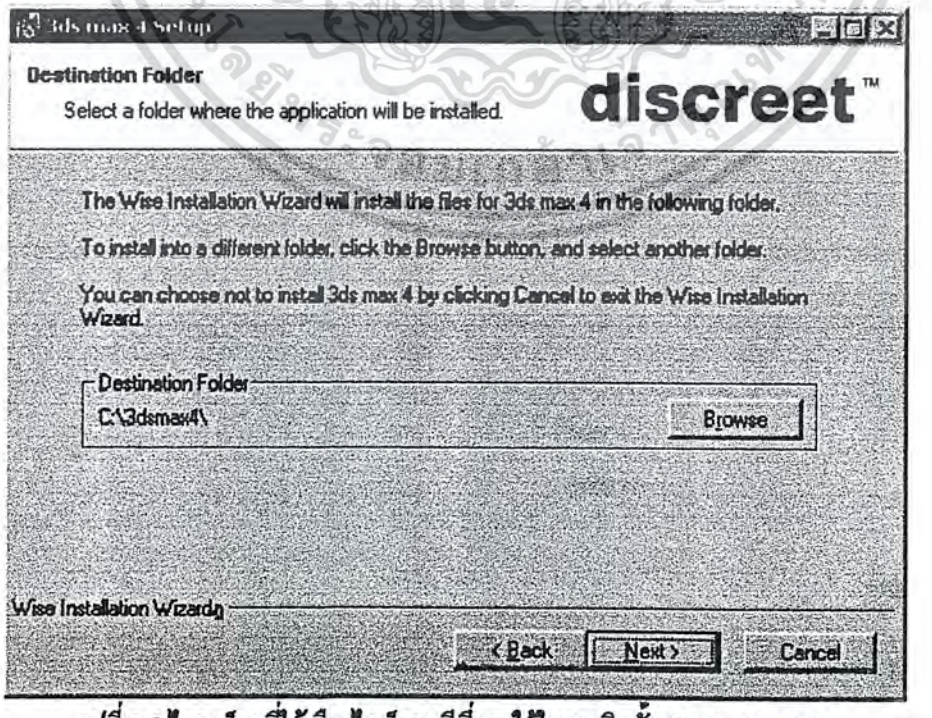
รูปที่ ก-6 รายละเอียดของผลิตภัณฑ์

2.8 จากนั้นจะขึ้นไดอะล็อกให้พิมพ์ข้อมูลรายละเอียดของผู้ใช้ แล้วคลิกปุ่ม Next >



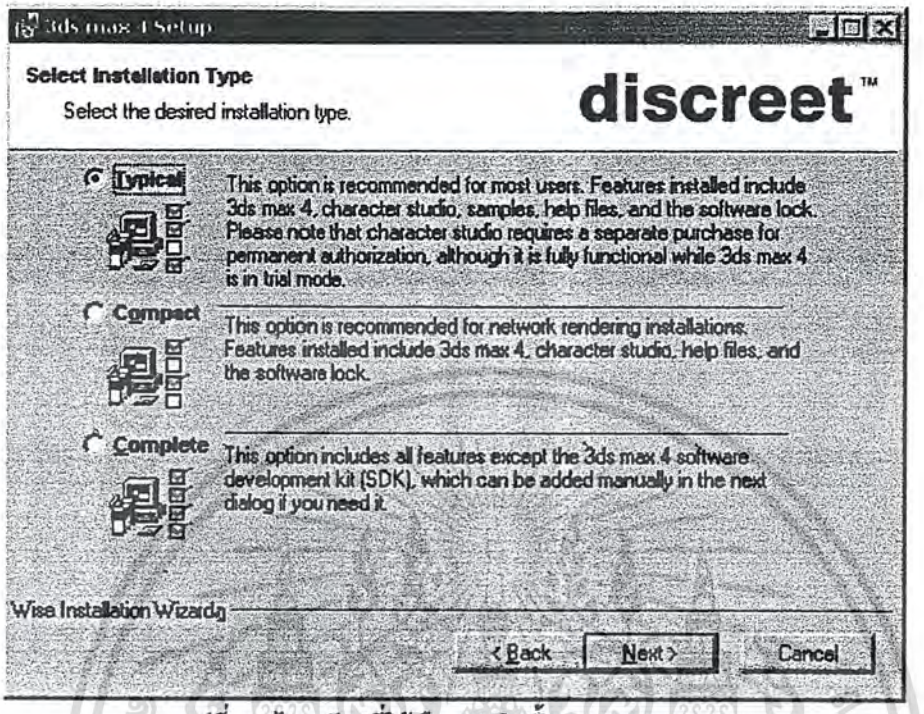
รูปที่ ก-7 ใส่ข้อมูลรายละเอียดของผู้ใช้

2.9 จากนั้นจะถามไดเร็คทอรีที่จะลงโปรแกรมในการติดตั้ง โดยถ้าไม่ต้องการไดเร็คทอรีดีฟอลต์นี้ ให้คลิกปุ่ม Browse...



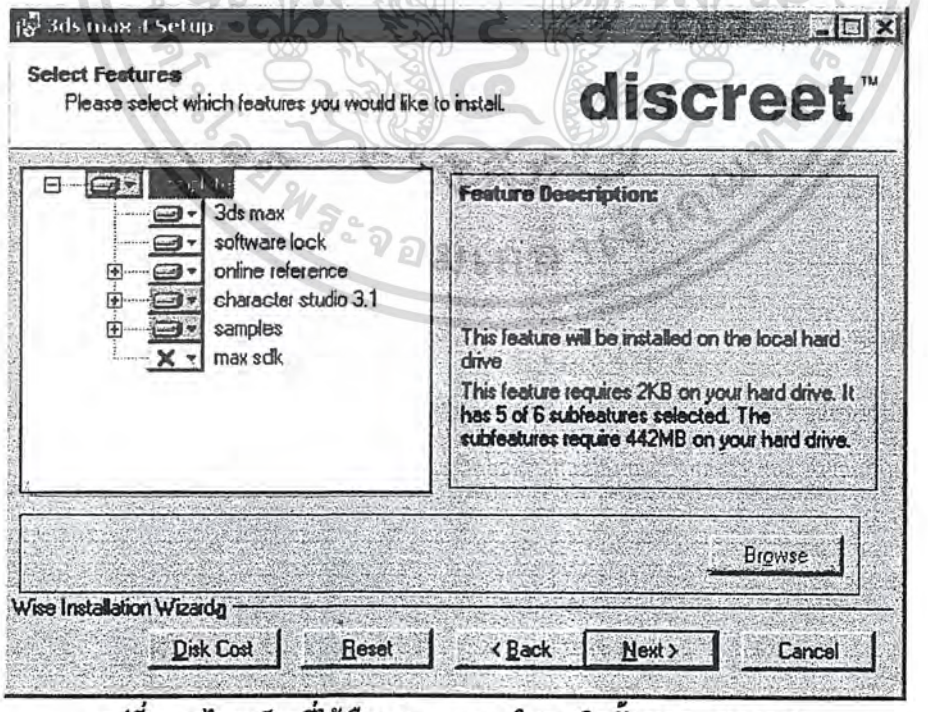
รูปที่ ก-8 ไดอะล็อกที่ให้เลือกไดเร็คทอรีซึ่งจะใช้ในการติดตั้ง 3D Studio MAX

2.10 จากนั้นจะให้เลือกชนิด Typical ในการติดตั้ง 3D Studio MAX แล้วคลิกปุ่ม Next >



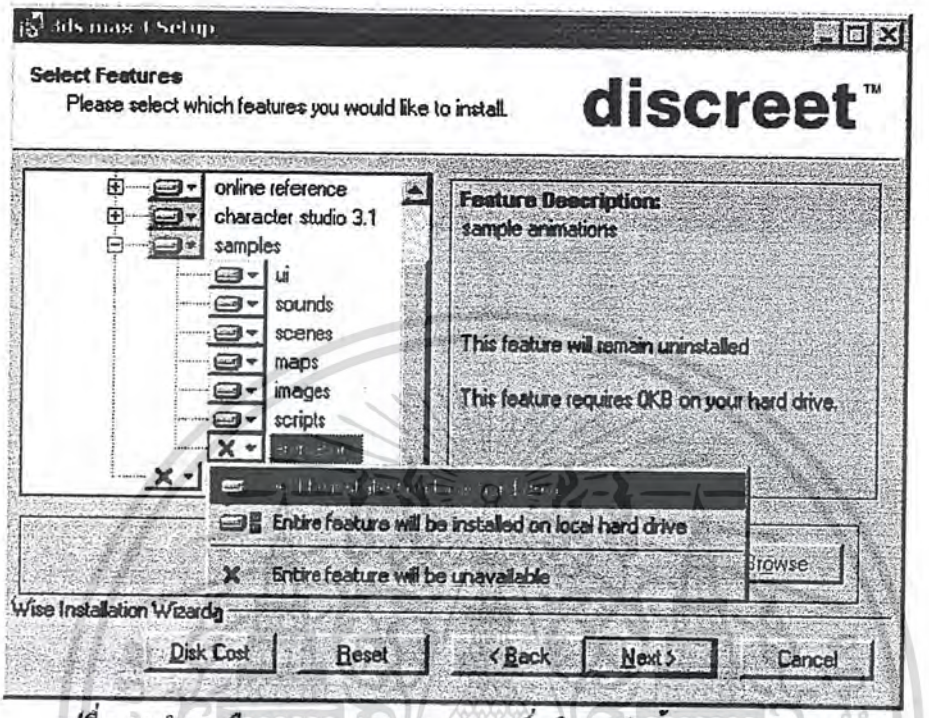
รูปที่ ก-9 ไดอะล็อกที่ให้เลือกการติดตั้ง 3D Studio MAX

2.11 จากนั้นจะให้เลือก Component ในการติดตั้ง 3D Studio MAX



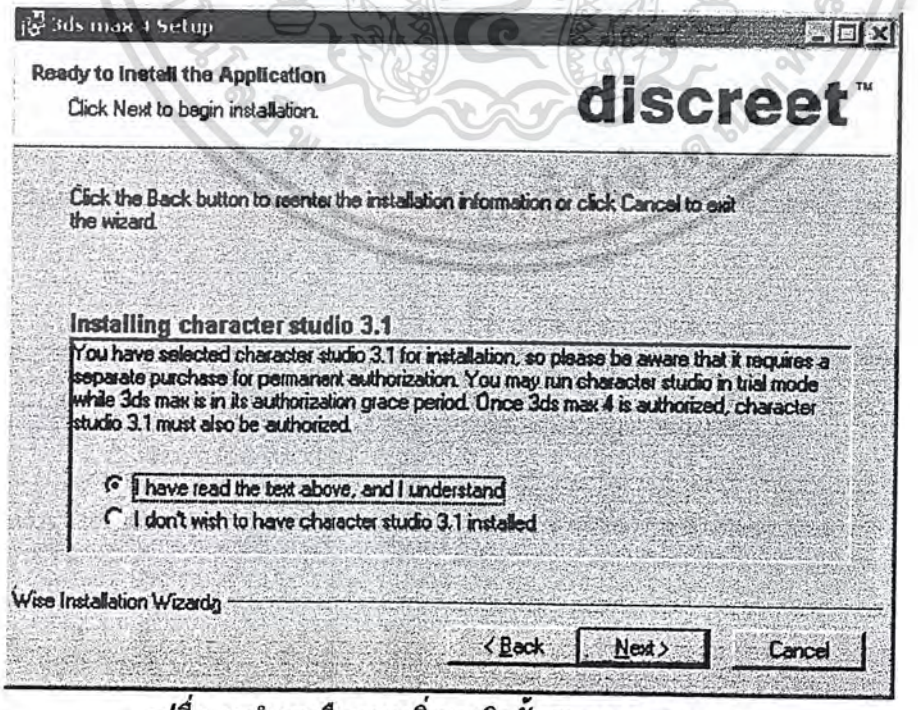
รูปที่ ก-10 ไดอะล็อกที่ให้เลือก Component ในการติดตั้ง 3D Studio MAX

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

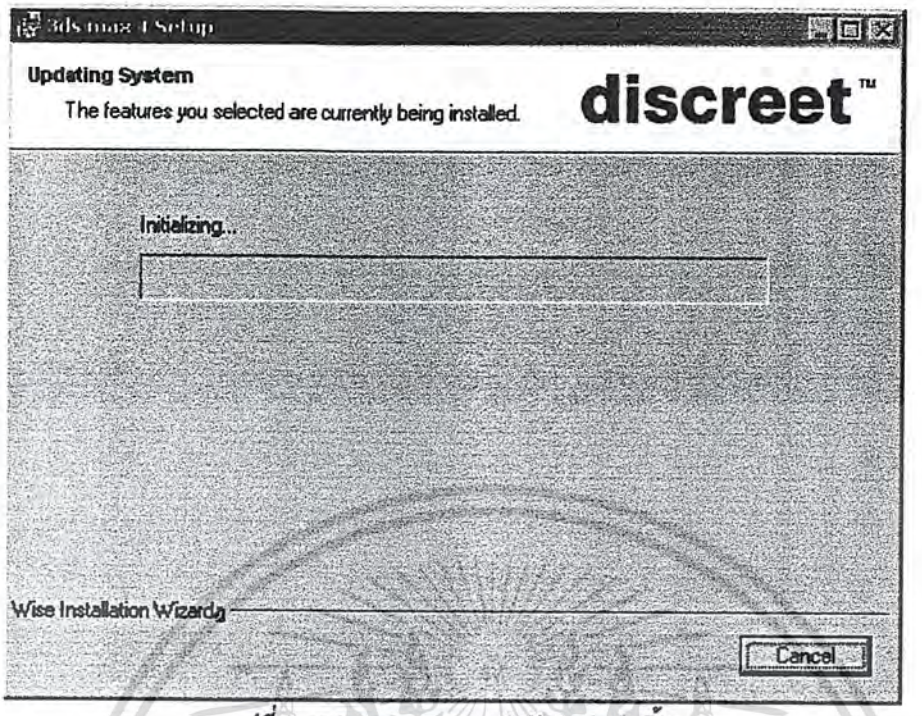


รูปที่ ก-11 ทำการเลือก Component Animation เพิ่ม ในการติดตั้ง 3D Studio MAX

2.13 จากนั้นจะขึ้นไดอะล็อกให้เลือกการติดตั้ง Character studio 3.1 ให้คลิกปุ่ม 

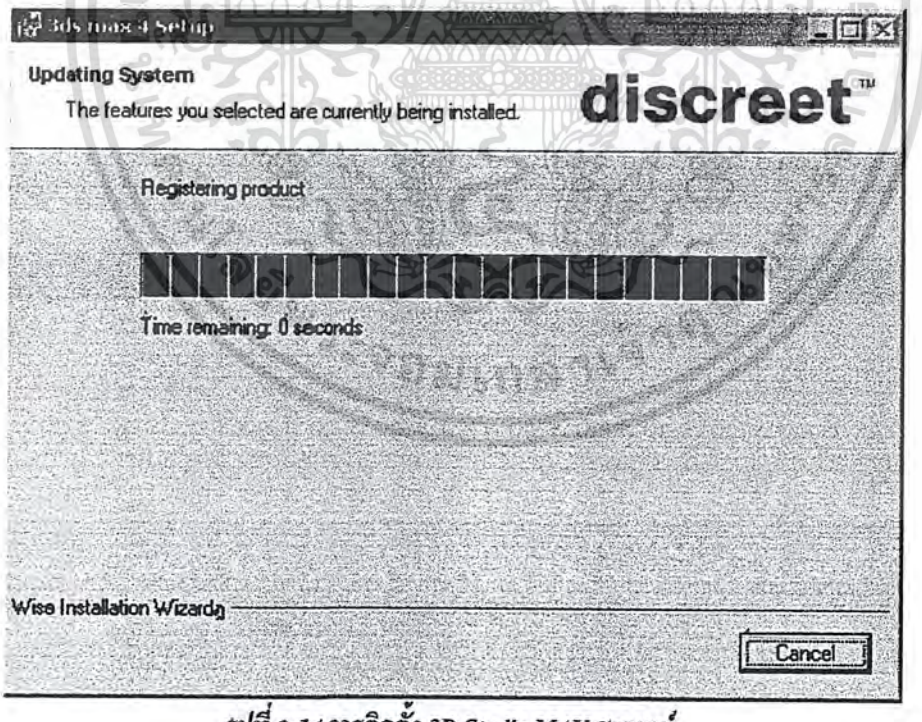


รูปที่ ก-12 ทำการเลือกการ เพิ่มการติดตั้ง Character studio 3.1

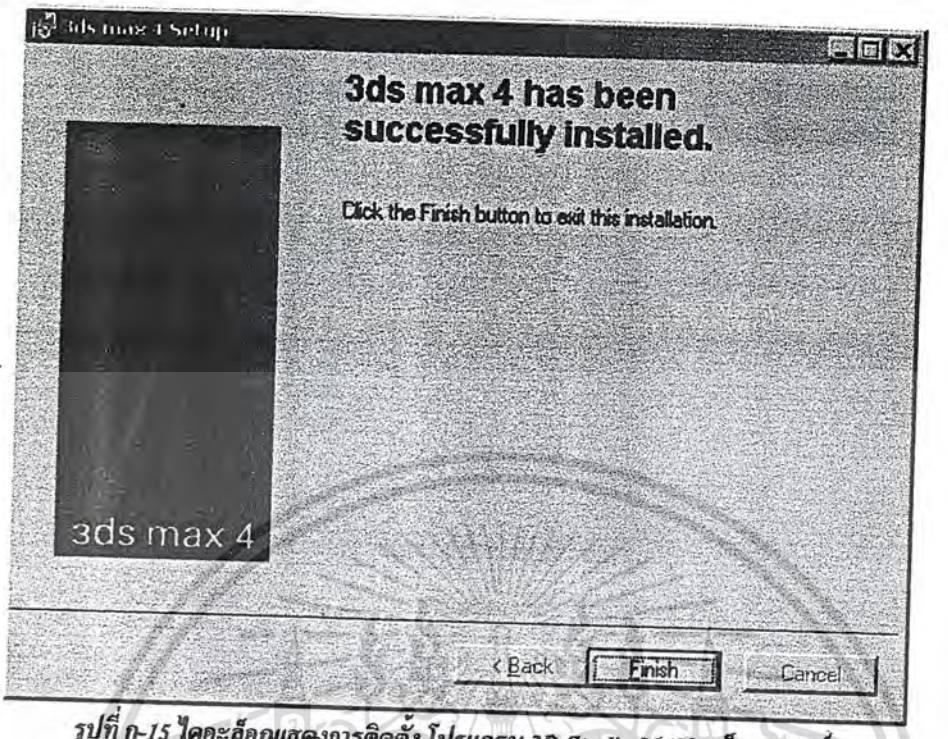


รูปที่ ก-13 3D Studio MAX กำลังทำการติดตั้ง

2.14 จากนั้นจะขึ้นไดอะล็อกแสดงสภาวะการติดตั้งโปรแกรม 3D Studio MAX ให้รอสักครู่

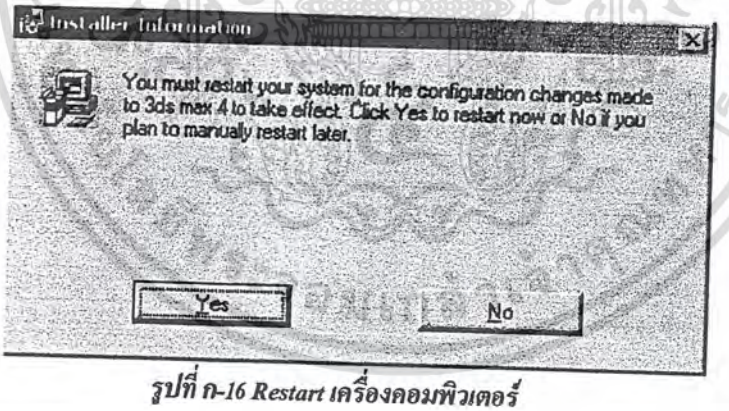


รูปที่ ก-14 การติดตั้ง 3D Studio MAX สมบูรณ์



รูปที่ ก-15 ไดอะล็อกแสดงการติดตั้ง โปรแกรม 3D Studio MAX เสร็จสมบูรณ์

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



รูปที่ ก-16 Restart เครื่องคอมพิวเตอร์

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.