การปรับปรุงประสิทธิภาพกราฟฟิกส์เอ็นจิ้นแบบ 3 มิติ สำหรับโปรแกรมประยุกต์แบบเรียลไทม์

Optimization of 3D Graphic Engine for Realtime Application

นายดิฐวัฒน์ ธนูสุวรรณศักดิ์
นายธนวัฒน์ แซ่อึ้ง

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2549

การปรับปรุงประสิทธิภาพกราฟฟิกส์เอ็นจิ้นแบบ 3 มิติ สำหรับโปรแกรมประยุกต์แบบเรียลไทม์

Optimization of 3D Graphic Engine for Realtime Application

โดย

นายดิฐวัฒน์ ธนูสุวรรณศักดิ์

นายธนวัฒน์ แซ่อึ้ง

อาจารย์ที่ปรึกษา

ดร. สมศักดิ์ วลัยรัชต์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2549

ปริญญานิพนธ์ปีการศึกษา 2549

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การปรับปรุงประสิทธิภาพกราฟฟิกส์เอ็นจิ้นแบบ 3 มิติสำหรับโปรแกรมประยุกต์แบบเรียลไทม์

Optimization of 3D Graphic Engine for Realtime Application

ผู้จัดทำ

1. นาย ดิฐวัฒน์ ธนุสุวรรณศักดิ์ รหัสนักศึกษา 46010239
2. นาย ธนวัฒน์ แซ่อึ้ง รหัสนักศึกษา 46010283

อาจารย์ที่ปรึกษา

(ดร. สมศักดิ์ วลัยรัชต์)

การปรับปรุงประสิทธิภาพกราฟฟิกส์เอ็นจิ้นแบบ 3 มิติ สำหรับโปรแกรมประยุกต์แบบเรียลไทม์

นาย ดิฐวัฒน์ ธนูสุวรรณศักดิ์ 46010239
นาย ธนวัฒน์ แซ่อึ้ง 46010283
ดร. สมศักดิ์ วลัยรัชต์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2549

บทคัดย่อ

ปริญญานิพนธ์นี้เกี่ยวกับการพัฒนากราฟฟิกส์เอ็นจิ้นแบบ 3 มิติ เพื่อสามารถนำไปใช้ในการพัฒนาโปรแกรมประยุกต์ประเภท 3 มิติแบบเรียล์ไทม์ โดยเนื้อหาจะคลอบคลุมถึงองค์ประกอบต่างๆ ของเอ็นจิ้น หลักการในการออกแบบ ทั้งนี้จะให้ความสำคัญกับประสิทธิภาพในการทำงานจริง นอกจากนั้นยังรวมถึงความยืดหยุ่นที่จะสามารถนำไปใช้งานได้อย่างไม่ยากนัก และสามารถที่จะปรับปรุงเปลี่ยนแปลงแก้ไขชิ้นงานได้ในอนาคตเมื่อมี เทคโนโลยีใหม่ๆ เพิ่มเข้ามา ทั้งนี้ กราฟฟิกส์เอ็นจิ้นที่ได้พัฒนาขึ้นจะสามารถใช้งานและจัดการระบบ Scene Graph ในแบบ Hierarchy, การแสดงภาพเคลื่อนไหวด้วยเทคนิคต่างๆ, การแสดงภาพภูมิประเทศ ตลอดจนถึงเทคนิคพิเศษต่างๆ ที่จะเพิ่มความสมจริงทางด้านภาพให้มากขึ้น นอกจากนั้นยังรวมไปถึงส่วนที่เป็นเครื่องมือเพื่อที่จะนำข้อมูลจากโปรแกรมสร้างโมเดล 3 มิติ เข้ามาใช้งานในระบบได้

กราฟฟิกส์เอนจิ้นนี้ใช้ภาษา C/C++ ในการพัฒนาร่วมกับ Graphic API 2 ชนิดต่างๆ คือ DirectX และ OpenGL แต่อย่างไรก็ตาม กราฟฟิกส์เอนจิ้นนี้ สามารถเรียกใช้ได้โดยออกแบบให้ส่วนหลักในการใช้งานไม่ขึ้นกับ API ที่กล่าวมาข้างต้น

Optimization of 3D Graphic Engine for Realtime Application

Dithawat Dhanusuwansak	46010239
Tanawat Sae-ung	46010283
Somsak Walairacht	Advisor
Academic Year 2005	

ABSTRACT

This thesis is about the development of 3D graphic engine development which can be used to implement any 3D graphic application. The contents cover essential components of the graphic engine, the principle of design. The developed engine is aimed for the realtime-processing performance and provides flexibility to the user. It includes the ability to change and modify for future technology. It can perform hierarchical scene graph management, various animation techniques, terrain rendering, and various visual effects. In addition, the export tool for modeling package is also provided.

This graphic engine is developed by using C/C++ as main development language together with 2 Graphic APIs (that is DirectX Graphics and OpenGL). However, the core engine can also be used in the API-independent manner.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จได้อย่างดี ด้วยคำแนะนำ และคำปรึกษาจาก ดร.สมศักดิ์ วลัยรัชต์ ซึ่งเป็นอาจารย์ผู้ควบคุมโครงงานและปริญญานิพนธ์ ข้าพเจ้ารู้สึกซาบซึ้งในความอนุเคราะห์จากท่านอาจารย์ และขอขอบพระคุณเป็นอย่างสูง

ขอขอบคุณเพื่อนๆ พี่ๆ น้องๆ ในภาควิชาวิศวกรรมคอมพิวเตอร์ สถาบันเทคโนโลยี พระจอมเกล้าเจ้าคุณทหารลาดกระบัง ทุกคนที่ให้คำแนะนำต่างๆ และคอยให้กำลังใจเสมอมา

สุดท้ายนี้ข้าพเจ้าขอกราบขอบพระคุณ บิดา มารดา และครอบครัวของข้าพเจ้าที่เป็นกำลังใจและให้การสนับสนุนในทุกเรื่องๆ ทำให้ข้าพเจ้าสามารถทำวิทยานิพนธ์ฉบับนี้สำเร็จลุล่วงด้วยดี

คุณค่าและประโยชน์อันพึงมาจากวิทยานิพนธ์ฉบับนี้ ข้าพเจ้าขอมอบแด่ผู้มีพระคุณทุกท่าน

นาย ดิฐวัฒน์ ธนูสุวรรณศักดิ์
นาย ธนวัฒน์ แซ่อึ้ง

สารบัญ

สารบัญตาราง

สารบัญรูป

บทที่ 1
บทนำ

1.1 บทนำ

ปัจจุบันนี้ คอมพิวเตอร์ได้มีส่วนในการให้ความบันเทิงกับผู้ใช้งานมากขึ้น โปรแกรมใหม่ๆ ที่ต้องการการแสดงผลแบบ 3 มิติ มีแนวโน้มมากขึ้นเรื่อยๆ อันเนื่องมาจากความสามารถต่างๆ ที่ผู้พัฒนา Graphic API พยายามที่จะใส่เพิ่มเติม ทำให้กระบวนการของการพัฒนาโปรแกรมดังกล่าวจึงทำได้ลำบากขึ้น อีกทั้งหากสั่งงานโดยไม่มีความรู้ความเข้าใจในการทำงานของอุปกรณ์พวกฮาร์ดแวร์เลย จะทำให้ประสิทธิภาพในการทำงานลดลงอย่างมาก ดังนั้น กราฟฟิกส์เอ็นจิ้นจึงถูกพัฒนาขึ้นเพื่อลดภาระดังกล่าว ของผู้พัฒนาโปรแกรม

1.2 วัตถุประสงค์ของโครงการ

1.2.1 นำไปใช้เป็นเครื่องมือ เพื่อลดความยุ่งยากซับซ้อนในการพัฒนาโปรแกรมประยุกต์เรียลไทม์แบบ 3 มิติ

1.2.2 ระบบมีความสามารถในการควบคุมจัดการ เพื่อเพิ่มประสิทธิภาพที่นอกเหนือจากการทำงานปกติทั่วไปของ Graphic API

1.2.3 สามารถนำข้อมูลจากโปรแกรมสร้างวัตถุ 3 มิติมาใช้งาน โดยป้องกันการแก้ไขจากผู้ใช้งานทั่วไป

1.3 ขอบเขตของโครงงาน

ขอบเขตของโครงงานนี้ได้แบ่งตามส่วนประกอบต่างๆ ของเอ็นจิ้นดังต่อไปนี้

1.3.1 Mathematics

เป็นส่วนที่รวบรวมข้อมูลพื้นฐานของระบบที่ใช้งานและการคำนวณทางด้านคณิตศาสตร์ต่างๆ ที่เกี่ยวข้อง เช่น Vector, Matrix และ Quaternion เป็นต้น ซึ่งเป็นส่วนที่ถูกเรียกใช้งานบ่อยครั้งที่สุด ดังนั้นจะต้องให้ความสำคัญทางด้านความเร็วในการประมวลผลเป็นหลัก

1.3.2 Graphics

เป็นส่วนที่จัดการและควบคุมการติดต่อระหว่างโปรแกรมเมอร์และ Graphic API (Renderer) รวมไปถึงระบบในการควบคุมวัตถุและจัดการความสัมพันธ์ของวัตถุต่างๆ ในฉาก (Scene Graph) นอกจากนั้น ยังเพิ่มเติมส่วนประกอบที่ทำงานร่วมอื่น เช่น ส่วนการ

แสดงภาพเคลื่อนไหว (Controller), การแสดงภาพภูมิประเทศ (Terrain), ฯลฯ โดยเน้นที่ความง่ายและความยืนหยุ่นในการใช้งาน

1.3.3 Applications

เป็นส่วนที่นำส่วนต่างๆ ที่กล่าวไว้ข้างต้นมาผสมผสานกัน โดยจัดหาองค์ประกอบพื้นฐานที่จำเป็นในการสร้างโปรแกรมประยุกต์ และสร้างระบบในการทำงานพื้นฐานเบื้องต้น

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1.4.1 ได้รับความรู้ ความเข้าใจเกี่ยวกับการแสดงผลภาพ 3 มิติในคอมพิวเตอร์

1.4.2 ได้รับความรู้ ความเข้าใจเกี่ยวกับการออกแบบกราฟฟิกส์เอ็นจิ้นแบบ 3 มิติ

1.4.3 ได้รับความรู้ ความเข้าใจเกี่ยวกับโครงสร้างของไฟล์ข้อมูลของวัตถุ 3 มิติ

1.5 ส่วนประกอบของปริญญานิพนธ์

เนื้อหาของปริญญานิพนธ์ฉบับนี้ประกอบไปด้วย 5 บทด้วยกัน คือ บทนำ, ทฤษฎีพื้นฐานและทฤษฎีที่เกี่ยวข้อง, การออกแบบและพัฒนา, การทดลองและผลการทดลอง, บทวิจารณ์และสรุป โดยสามารถจำแนกรายละเอียดได้ ดังต่อไปนี้

บทที่ 1 บทนำ กล่าวถึงจุดประสงค์ ประโยชน์ที่ได้รับและขอบเขตของโครงงาน

บทที่ 2 ทฤษฎีพื้นฐานและทฤษฎีที่เกี่ยวข้อง กล่าวถึง ทฤษฎีที่นำมาใช้ในโครงงานโดยสังเขป

บทที่ 3 การออกแบบและพัฒนา กล่าวถึง การออกแบบและโครงสร้างของชิ้นงาน

บทที่ 4 การทดลองและผลการทดลอง กล่าวถึง การทดสอบความถูกต้องตามทฤษฎีว่าได้ตามที่ต้องการหรือไม่

บทที่ 5 บทวิจารณ์และสรุป กล่าวถึง บทสรุป, ปัญหาอุปสรรค, แนวทางแก้ไข, และแนวทางพัฒนาต่อ

บทที่ 2
ทฤษฎีพื้นฐานและทฤษฎีที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงทฤษฎีพื้นฐานโดยทั่วไป และที่เกี่ยวข้องในการพัฒนา 3D Graphic Engine เพื่อที่จะทำให้ผู้อ่านได้ทราบถึงประโยชน์ และสามารถเข้าใจเนื้อหาในบทถัดไป ได้อย่างดียิ่งขึ้น โดยจะไม่กล่าวลึกลงรายละเอียดมากนัก แต่ก็ให้พอเพียงสำหรับทำความเข้าใจตามวัตถุประสงค์ได้

2.1 3D Graphic Engine

Graphic Engine (หรือพูดให้ชัดเจนยิ่งขึ้น คือ Library) เป็น การสร้างระบบที่มีงานหลักในการติดต่อกับ Graphic API ที่มีจุดประสงค์เพื่อลดความซับซ้อน, ความยุ่งยาก ในการทำงานที่จะต้องเรียกใช้งานโดยตรง ทั้งนี้เพื่อที่จะสร้างความสะดวกและความเข้าใจที่ง่ายให้กับผู้ใช้งาน

จริงๆ แล้ว Engine เอง ไม่มีคำจำกัดความที่แน่แท้และตายตัว ดังนั้น จึงจะแตกต่างกันไปในมุมมองของแต่ละคน ทั้งนี้สิ่งสำคัญ คือ "หลักการโดยทั่วไปที่ Engine ควรจะต้องทำได้" ดังนี้

(1) จัดการกับทุกๆ ข้อมูลในส่วนที่มันรับผิดชอบอยู่

(2) คำนวณทุกข้อมูลตามส่วนการทำงานของมัน

(3) ส่งทุกข้อมูลไปให้กับ Instance ที่ตามมาของมัน, ถ้าจำเป็น

(4) รับข้อมูลจาก Instance ก่อนหน้า เพื่อไปจัดการและคำนวณ

หากมองในแง่ของผู้ใช้งานว่าต้องการให้ Engine เป็นอย่างไร ก็ย่อมจะต้องการให้ง่ายและสะดวก คือ แค่ Init, Load, และ Shut เท่านั้น โดยไม่สนใจว่าการทำงานเชิงเทคนิคว่าถูก implement ได้อย่างไร ในขณะที่ผู้สร้าง Engine ยังคงมีความจำเป็นอย่างยิ่งที่จะต้องทำความเข้าใจว่าฮาร์ดแวร์ ทำงานอย่างไร เพื่อหลีกเลี่ยงปัญหาคอขวดที่จะเกิดขึ้นได้ในระบบ

ทั้งนี้ Engine ควรที่จะต้องไม่ขึ้นกับกับ Project ใดๆ และมีความสามารถที่จะรองรับ Project สำหรับ Application ทางด้าน Multimedia ต่างๆ ได้ โดยไม่จำเป็นต้องทำการปรับปรุงแก้ไขส่วนของ Engine ใดๆ เลย ดังนั้นส่วนของ Engine Code ควรที่จะต้องไม่มีการรวมสิ่งที่เกี่ยวข้องกับ Application ไว้ภายใน เพื่อที่จะทำให้มันมีความยืดหยุ่นและมีความเป็นไปได้ในการนำมาใช้ใหม่ได้

นอกจากนี้ ตัว Engine ยังจะต้องอาศัยข้อมูลจากภายนอกโดยจะอยู่ในรูปของ File Format

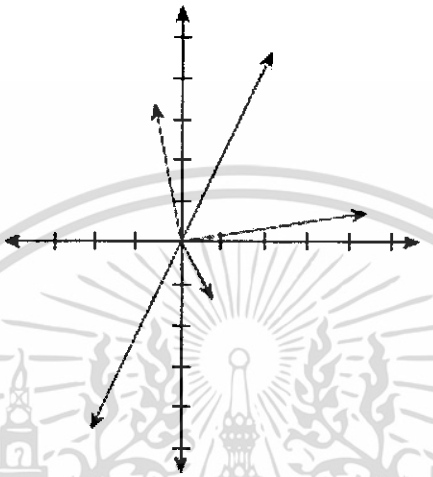
2.2 ความรู้พื้นฐาน

2.2.1 Vector

เวกเตอร์ (Vector) เป็นอาเรย์ตัวเลขหนึ่งมิติ ในกราฟฟิกสามมิติและเกมส่วนมากเวกเตอร์

จะมีองค์ประกอบภายในสอง สาม หรือสี่องค์ประกอบ โดยแต่ละอันก็จะหมายถึงแต่ละแกน แต่จริงๆแล้วเวกเตอร์นั้นจะมีองค์ประกอบจำนวนเท่าใดก็ได้ตามความต้องการ ซึ่งจะใช้ในการบอกค่าต่างๆไม่ว่าจะเป็นตำแหน่ง ทิศทาง และ ความเร็วในการเคลื่อนที่ของวัตถุ เป็นต้น

โดยเวกเตอร์นั้นจะต้องประกอบด้วยขนาดและทิศทาง หากเขียนเป็นรูปภาพก็จะสามารถแทนขนาดได้ด้วยความยาวของเส้นและทิศทางสามารถแสดงได้ด้วยหัวลูกศร ดังรูป



รูปที่ 2.1 แสดงตัวอย่างเวกเตอร์ด้วยแผนภาพ

นอกจากการแสดงด้วยรูปจะสามารถเขียนเป็นสัญลักษณ์แทนได้หลายรูปแบบ ซึ่งวิธีแสดงแบบที่ง่ายที่สุดคือการแสดงในรูป <x, y, z> ซึ่ง x, y และ z จะเป็นระยะทางตามแนวแกน x, y และ z ตามลำดับ

Magnitude and Unit Vectors

ขนาดของเวกเตอร์นั้นจะเขียนในรูปของ ||V|| ซึ่งเวกเตอร์นั้นสามารถหาขนาดได้โดยใช้สูตรในการหาความยาวดังสมการด้านล่าง ส่วนเวกเตอร์หนึ่งหน่วย (Unit Vector) ก็คือเวกเตอร์ที่มีขนาดเป็นหนึ่ง โดยทุกๆเวกเตอร์สามารถแปลงเป็นเวกเตอร์หนึ่งหน่วยได้โดยการทำ Normalization ซึ่งก็คือการนำเวกเตอร์นั้นๆไปหารด้วยขนาดของมันเอง ดังตัวอย่างด้านล่าง

$$u = \langle 3, 4 \rangle$$

$$|u| = \sqrt{3^2 + 4^2} = 5$$

$$v = \frac{u}{|u|} = \langle 0.6, 0.8 \rangle$$

$$|v| = \sqrt{0.6^2 + 0.8^2} = 1$$

รูปที่ 2.2 แสดงตัวอย่างการคำนวณ Unit Vectors

Vector Arithmetic

เวกเตอร์สามารถที่จะทำการบวก ลบ และ คูณด้วยค่า Scalar (ค่าที่มีแต่ขนาดอย่างเดียว) ได้เพื่อทำการเปลี่ยนแปลงขนาดหรือทิศทาง ในโปรแกรมสามมิติสามารถที่จะนำ Vector Arithmetic มาใช้ในการเคลื่อนที่หรือแก้ขนาดวัตถุ เนื่องจากมีข้อดีในการเก็บค่าของขนาดและทิศทางของวัตถุทั้งสองค่าได้ในตัวแปรเดียว โดยมี Operation ที่สำคัญดังนี้

- Addition and Subtraction
- Scalar Multiplication and Division

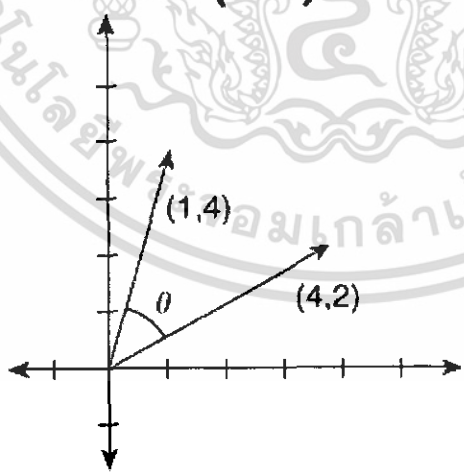
Vector Products

เวกเตอร์ไม่สามารถที่จะทำการคูณกันด้วยวิธีปกติเหมือน Scalar จึงมี 2 Operation ที่ใช้ในการทำการคูณระหว่างเวกเตอร์คือ Dot Product และ Cross Product

Dot Product จะใช้ในการหามุมระหว่างสองเวกเตอร์หรือใช้ในการตรวจสอบความเป็น Orthogonal ได้ ซึ่งเป็นประโยชน์มากในด้านสามมิติ เช่นการหาแรงลัพธ์ และ ความเร็วของวัตถุเมื่อมีแรงลม แรงโน้มถ่วงของโลกหรือแรงอื่นๆมากระทำกับวัตถุนั้นๆ เป็นต้น โดยมีวิธีการหาดังนี้

$$u \bullet v = |u||v|\cos\theta$$

$$\theta = \arccos\left(\frac{u \bullet v}{|u||v|}\right)$$



$$\theta = \arccos\left(\frac{1 \times 4 + 4 \times 2}{\sqrt{17} \times 2\sqrt{5}}\right) = 49.4°$$

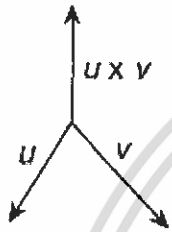
รูปที่ 2.3 แสดงตัวอย่างการหามุมระหว่างเวกเตอร์

Cross Product

Cross Product ใช้ในการคำนวณหาเวกเตอร์ที่ตั้งฉากทั้งสองเวกเตอร์ที่ทำการ Cross Product กัน มักจะใช้ในการหาเวกเตอร์ของพื้นผิววัตถุ เนื่องจากทิศของพื้นผิวจะมีทิศทางตั้งฉากกับระนาบผิววัตถุ หรือ อาจใช้ในการหาทิศทางของแสง เพราะแสงก็จะมีทิศทางตั้งฉากกับพื้นผิววัตถุเช่นกัน โดยสามารถหาค่า Cross Product ได้จากสมการ

$$uxv = (y_1z_2 - z_1y_2)i - (x_1z_2 - z_1x_2)j + (x_1y_2 - y_1x_2)k$$

จากสมการจะเห็นได้ว่า *Cross Product* นั้นจะใช้ได้กับเวกเตอร์สามมิติเท่านั้น ไม่สามารถใช้กับเวกเตอร์สองมิติได้ เนื่องจากไม่มีทิศทางการตั้งฉาก ดังรูป



$$\langle 1,5,2\rangle \times \langle 2,1,3\rangle$$
$$(5 \times 3 - 2 \times 1)i - (1 \times 3 - 2 \times 2)j + (1 \times 1 - 5 \times 2)k =$$
$$\langle 13,1,-9\rangle$$

รูปที่ 2.4 แสดงตัวอย่างการครอสเวกเตอร์

Transforming a Vector by Matrix

การย้ายตำแหน่งของจุดหรือวัตถุ ส่วนมากจะใช้เมตริกในการย้ายเวกเตอร์หรือจุด เรียกกระบวนการนี้ว่า Transformation เมตริกนั้นจะถูกใช้ในสองกรณีคือ การใช้เมตริกในการเก็บค่าของการทำ Transformation ในอดีต และ การใช้เมตริกง่ายต่อการทำ Multiplication ของเวกเตอร์และจุด

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{bmatrix} \begin{bmatrix} 3 \\ 1 \\ 4 \end{bmatrix} = \begin{bmatrix} 1\times3+2\times1+3\times4 \\ 4\times3+5\times1+6\times4 \\ 7\times3+8\times1+9\times4 \end{bmatrix} = \begin{bmatrix} 17 \\ 41 \\ 65 \end{bmatrix}$$

รูปที่ 2.5 แสดงตัวอย่างการ Transforming เวกเตอร์ด้วยเมตริก

2.2.2 Matrix

เมตริกเป็นอาเรย์ของตัวเลขขนาด N*M ทางด้านของสามมิติแล้วนั้นสามารถใช้เมตริกคู่กับหลักการทางคณิตศาสตร์มาใช้ในการแสดงหรือคำนวณค่าของ Linear equation ได้โดยใช้เมตริก Operation เนื่องจากการใช้เมตริก Operation จะช่วยให้มีการทำงานน้อยกว่าการใช้วิธีอื่นในการคำนวณ เช่นการทำ Substitution ในด้านสามมิติเมตริกจะถูกแสดงในรูปของ Transformations เป็นส่วนสำคัญในการทำ Rotation และ Translation ของวัตถุสามมิติ ข้อดีที่ใช้เมตริกในรูปของ Transformation ก็คือการรวมกันของเมตริกโดยใช้หลัก Algebra ซึ่งง่ายและทำ

ให้โปรแกรมทำงานได้เร็ว โดยเมตริก N*N (มีจำนวน Columns และ Rows เท่ากัน) เป็นพื้นฐานของเมตริกดังรูป

$$\begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}$$

รูปที่ 2.6 แสดงตัวอย่างเมตริก

The zero and The identity matrix

The zero matrix คือ เมตริกที่มีค่าทุกค่าเป็นศูนย์ ผลลัพธ์ของการบวกหรือลบเมตริกใดๆ ด้วย Zero matrix จะเป็นค่าเดิม

The identity matrix คือ เมตริกที่นำไปคูณกับเมตริกใดๆจะได้ค่าเดิม ซึ่งภายใน Identity matrix จะมีค่าเป็นศูนย์ทั้งหมดยกเว้นตำแหน่งเส้นทะแยงมุมที่เริ่มจากด้านบนซ้ายลงมาด้านล่างขวาจะมีค่าเป็นหนึ่ง ซึ่งจะใช้ Identity matrix ในการ Reset matrix

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \quad I = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

รูปที่ 2.7 แสดง The Zero matrix และ The Identity Matrix

Matrix Operation

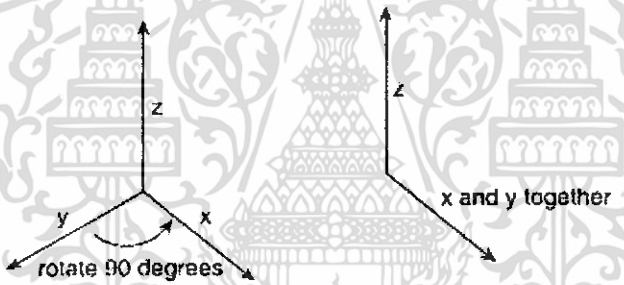
การทำ Matrix Operation เป็นพื้นฐานหลักในการคำนวณทางคณิตศาสตร์เพื่อใช้กับโมเดล โดยมี Operation ที่จำเป็นดังนี้

- Addition and Subtraction
- Scalar Multiplication
- Matrix Multiplication
- Determinant of Matrices
- Inverse of Matrix
- Transposing of Matrices

2.2.3 Quaternion

Quaternions จะประกอบด้วยชุดของตัวเลขสี่ตัวที่ใช้ในการแสดงค่าการหมุน โดยค่า x, y, z จะแทนด้วยเวกเตอร์ และค่า Scalar จะแทนด้วย w ซึ่งค่า Scalar นั้นบอกถึงองศาของการหมุนของวัตถุในรูปของ cos(angle/2) เราสามารถเขียน Quaternions ได้ในรูปของ q = [nv] โดย n คือค่า Scalar และ v คือค่าเวกเตอร์

การใช้ Quaternions ในระบบสามมิตินั้นมีข้อดีคือ จะไม่เกิดการ Gimbal lock ดังรูปที่ xxx และมีความราบรื่นและรวดเร็วของอนิเมชั่น การทำ Interpolation rotation จะง่ายกว่าการทำด้วย Matrix มากและได้ผลลัพท์ที่ดีกว่า และที่สำคัญคือการใช้ Quaternions นั้นจะใช้พื้นที่ในการเก็บข้อมูลน้อยกว่าการใช้ rotation matrix เนื่องจากใช้เพียง 4 ตำแหน่งแทนที่จะใช้ 9 ตำแหน่ง รวมถึงความเร็วในการประมวลผลของโปรเซสเซอร์ก็มีความเร็วในการคำนวณ Operations ต่างๆเร็วกว่า แต่ส่วนมากมาตรฐานโปรแกรมสำเร็จรูป (API) ต่างๆ ไม่รองรับการใช้งาน Quaternions โดยตรง จึงต้องทำการแปลงจาก Quaternions เป็น Matrices ก่อนเพื่อนำไปใช้ในส่วนอื่นๆของโปรแกรม



รูปที่ 2.8 แสดงภาพการเกิดปัญหา Gimbal Lock เนื่องจากไม่ได้ใช้ Quarternion

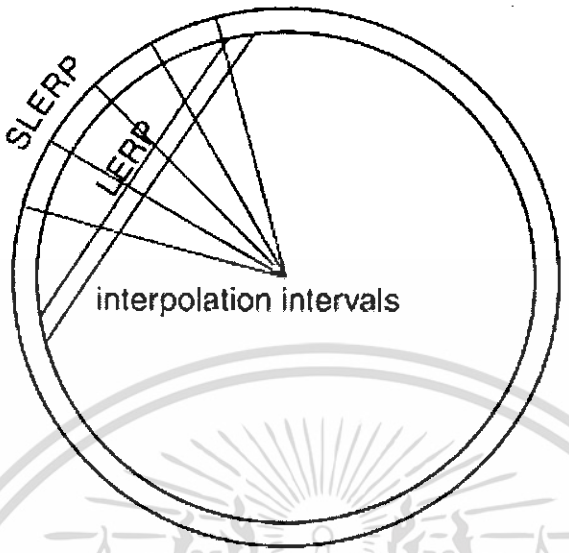
Interpolation with Quaternion

Interpolation มีความสำคัญเป็นอย่างมากในการทำงานของโมเดลสามมิติ การใช้ Interpolation จะช่วยในการคำนวณจุดสมมติขึ้นจากจุดที่ทราบแน่นอนสองจุด ซึ่งการทำด้วยวิธีนี้จะทำให้อนิเมชั่นมีความราบรื่นมากขึ้นเนื่องจากเราสามารถสร้างจุดจำนวนไม่จำกัดขึ้นระหว่างสองจุดที่กำหนดไว้ และสามารถที่จะกำหนดความถี่ในการสร้างจุดได้

Quaternion Interpolation ที่ได้ทำการศึกษานั้น ประกอบด้วย (Linear interpolation) LERP และ (Spherical linear interpolation) SLERP โดย LERP เป็นการทำ Interpolate ของเส้นตรงและ SLERP เป็นการทำ Interpolate ของเส้นโค้ง

โดยทั่วไป จะใช้ SLERP ในการทำ Interpolate ของการ Rotation มากกว่าที่จะใช้ LERP ในการทำเนื่องจากมีความราบรื่นมากกว่า แต่การใช้ SLERP จะมีความผิดพลาดได้มากกว่า ดังนั้นหากจุดที่ทราบแน่นอนสองจุดมีระยะที่ไม่ห่างกันมาก อาจจะใช้ LERP ในการทำ

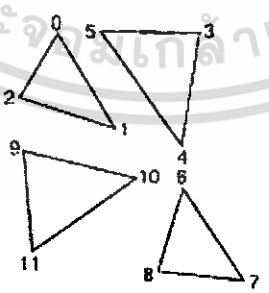
Interpolate แทนการใช้ SLERP ได้



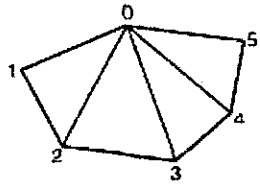
รูปที่ 2.9 แสดง 2D ของการใช้ SLERP และ LERP จะเห็นได้ว่า การใช้ SLERP จะเป็นการ interpolate ในส่วนของ วงกลม และ LERP เป็นการ Interpolate ในส่วนของเส้นตรง

2.2.4 Strips and Fans

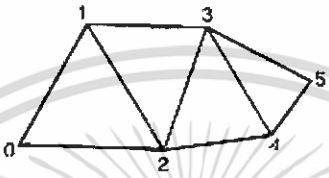
การใช้ Strips และ Fans ในการสร้าง Triangles ซึ่งจะช่วยในการเพิ่มความเร็วในการ Render 3D model ได้ โดยปกติการสร้าง Triangle นั้นจะต้องทำการระบุทั้งหมด 3 Vertices ต่อหนึ่ง Triangle และหากต้องการสร้าง 2 Triangles แล้ว ก็จะต้องระบุถึง 6 Vertices ด้วยกัน แต่การใช้แนวคิดแบบ Strips และ Fans จะช่วยในการลดการเก็บข้อมูลของ Vertices ได้ โดยให้ผลลัพธ์เหมือนกัน ซึ่งแนวคิดแบบ Strips และ Fans มีหลักการคร่าวๆดังนี้



รูปที่ 2.10 แสดงถึงการสร้าง Triangles โดยวิธีปกติจำนวน 4 Triangles จะต้องระบุ 12 Vertices



รูปที่ 2.11 แสดงถึงการสร้าง Triangles โดยวิธี Fan จำนวน 4 Triangles จะระบุเพียง 6 Vertices



รูปที่ 2.12 แสดงถึงการสร้าง Triangles โดยวิธี Strip จำนวน 4 Triangles จะระบุเพียง 6 Vertices

2.3 Rendering Pipeline

2.3.1 สถาปัตยกรรม

Graphic Rendering Pipeline ได้แนวคิดมาจากโลกของความเป็นจริง (เช่น ท่อส่งน้ำมัน) โดยที่จะประกอบไปด้วย หลายๆ Stage, ขอยกตัวอย่าง เช่น ในการขนส่งผ่านท่อน้ำมัน น้ำมันจะไม่สามารถเคลื่อนไหวจาก Stage แรก ไปยัง Stage ที่สองได้ จนกว่าน้ำมันใน Stage ที่สองจะเคลื่อนที่ไปยัง Stage ที่ สามไปเสียก่อน โดยที่ ความเร็วของของ Pipeline จะดูจาก Pipeline Stage ที่ช้าที่สุด ซึ่งเป็น ส่วนคอขวด(Bottleneck) ของ Stage ทั้งหมด

หากแบ่งอย่างหยาบๆ แล้ว จะได้เป็น Conceptual Stage ซึ่งแบ่งเป็น

Application เป็น Stage ที่ถูก implement โดยซอฟต์แวร์ เช่น Collision Detection, Acceleration Algorithm, Animations, Force Feedback, อื่นๆ

Geometry เป็น Stage ที่ถูก implement โดยทั้งซอฟต์แวร์และฮาร์ดแวร์ ขึ้นกับสถาปัตยกรรม (Architecture) เช่น Transform, Projection, Lighting ฯลฯ ทั้งนี้ จะทำการคำนวณว่า "อะไรจะถูกวาด ถูกวาดอย่างไรและที่ไหน"

Rasterizer ทำการวาดภาพจากข้อมูลที่ถูกสร้างขึ้น จาก Stage ก่อนหน้า ลงบนหน้าจอ

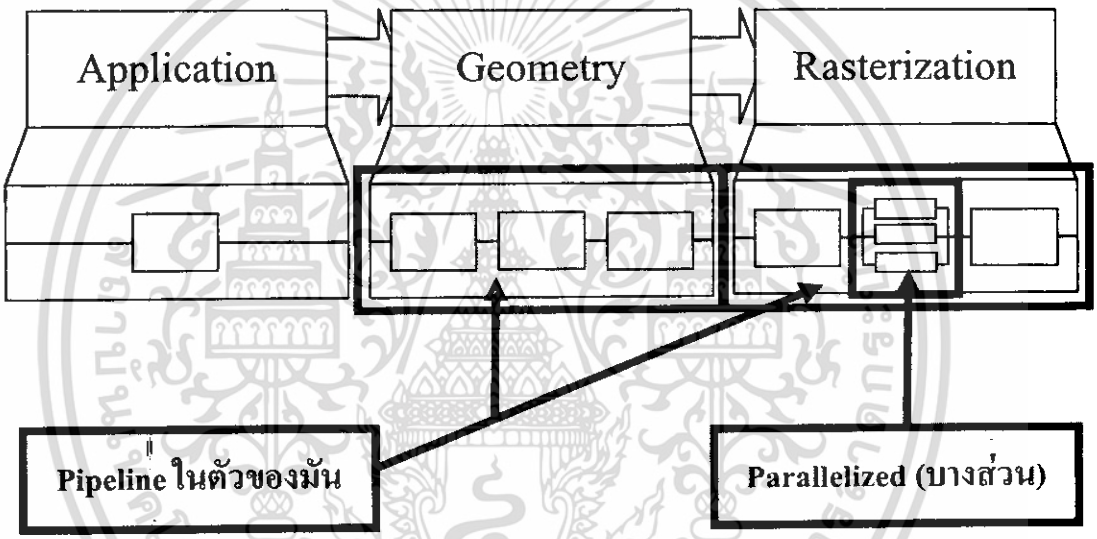
หมายเหตุ:

แต่ละ Stage มักจะเป็น Pipeline (ประกอบไปด้วย หลายๆ Stage) อยู่ภายในตัวมันเอง

Functional Stage หมายถึง "Stage ที่มีงานที่แน่นอน" (แต่ไม่ได้ระบุวิธีที่ถูกทำงาน)

Pipeline Stage หมายถึง "Stage ที่ถูกทำงานไปพร้อมๆ กัน กับ Pipeline Stage อื่นๆ ที่เหลือทั้งหมด", ซึ่งอาจจะถูกทำแบบขนาน เพื่อให้ได้ประสิทธิภาพสูงสุด

ความเร็วในการวาดภาพจะคิดหรือถูกตัดสินจาก Pipeline Stage ที่ช้าที่สุด ซึ่งก็คือ "ความเร็วในการทำการเปลี่ยนแปลงข้อมูลของภาพ" ซึ่ง ความเร็วดังกล่าว จะอยู่ในรูปของ Frame per Second หรือ Hz (บางครั้งอาจใช้คำว่า "Throughput" แทน)



รูปที่ 2.13 แสดงโครงสร้างพื้นฐานของ Rendering Pipeline

The Application Stage

เนื่องจากว่า Stage นี้ทำงานในส่วนซอฟต์แวร์ ตลอด, ดังนั้น นักพัฒนาสามารถเปลี่ยน Implementation เพื่อเปลี่ยนประสิทธิภาพได้ (ต่างจาก Stage อื่นๆ ที่ค่อนข้างยากกว่าที่จะเปลี่ยน Implementation เนื่องจาก Stage ดังกล่าวถูกสร้างขึ้นบนฮาร์ดแวร์, แต่ว่าก็ยังคงเป็นไปได้ที่จะส่งผลกระทบต่อ "เวลาที่ถูกใช้หรือโดย Geometry และ Rasterizer Stage")

เมื่อจบ Application Stage, Geometry ที่จะต้องถูก render จะถูกส่งไปให้กับ Stage ต่อไป ซึ่งนับเป็น "งานที่สำคัญที่สุดใน Application Stage"

ผลลัพธ์จากการที่เป็น Implementation ด้วยซอฟต์แวร์ ทำให้ Stage นี้ ไม่ได้ถูกแบ่งเป็น หลายๆ Stage ย่อย แต่อย่างได้ อย่างไรก็ตาม Stage นี้ก็ยังสามารถถูกทำงานแบบ

Parallel ได้บนหลายๆ Processor (CPU ที่ Design แบบ Superscalar Construction เพราะว่า มันสามารถที่จะ execute หลายๆ สิ่งในเวลาเดียวกันใน Stage เดียวกันได้)

Process ที่มักถูก implement โดยทั่วๆ ไป คือ Collision Detection (หลังจากนั้นอาจมี Response กลับไปหา Force Feedback Device) และยังมีที่สำหรับจัดการ Input จาก Source อื่นๆ (Keyboard, Mouse, Virtual Reality (VR) Helmet, อื่นๆ)

นอกจากนั้น กระบวนการอื่นๆ ยังรวม Texture Animation, Geometry Morphing หรือการคำนวณต่างๆ ที่ไม่ได้ทำให้ Stage อื่นๆ

The Geometry Stage

Geometry Stage จะรับผิดชอบหลักๆ กับการทำงานประเภท Per-Polygon หรือ Per-Vertex โดยถูกแบ่งออกเป็น Functional Stage ดังนี้

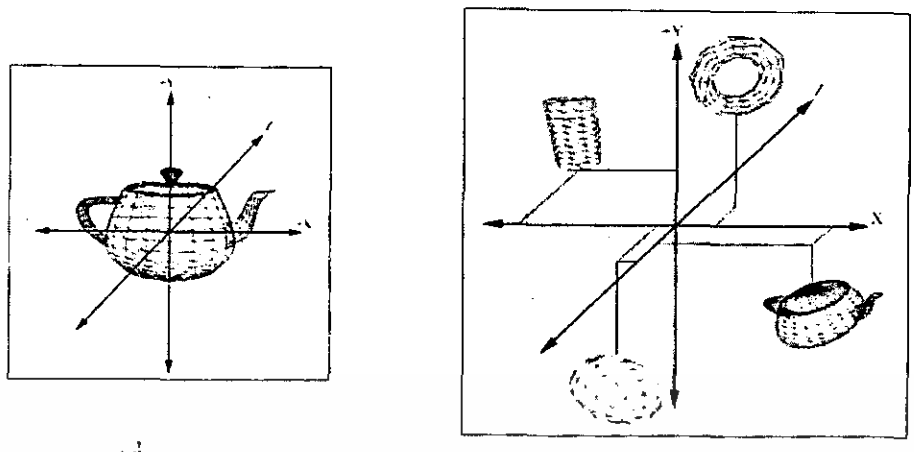
รูปที่ 2.2 แสดง Geometry Stage ที่ถูกแบ่งเป็น Pipeline ของ Functional Stages

บางกรณี Functional Stage ที่ต่อเนื่องกัน จะสร้าง Pipeline Stage เดี่ยวขึ้น (โดยทำงานอย่างขนานกับ Pipeline Stage อื่นๆ ตัวอย่าง เช่น "ทุก Stage ที่ทำงานอยู่ในส่วนซอฟต์แวร์"

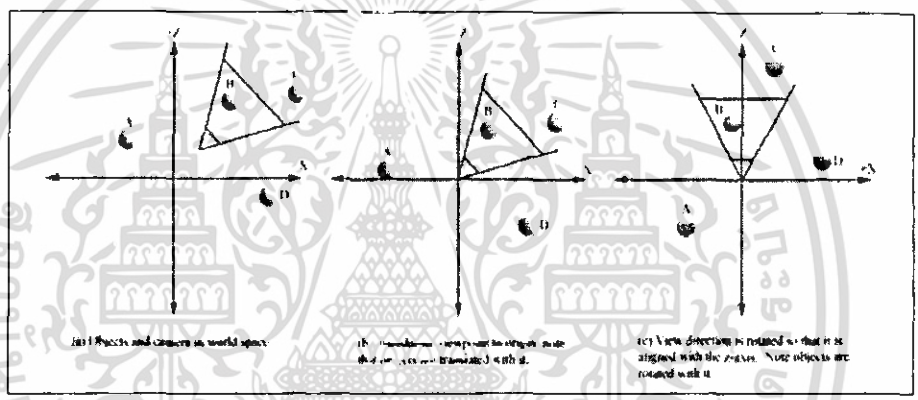
ส่วนกรณีอื่นๆ Functional Stage อาจถูกแบ่งย่อยออกเป็น Pipeline Stage ที่เล็กลง ตัวอย่าง เช่น "Functional Stage ใดๆ ที่ถูกแบ่งเพื่อที่แต่ละ Pipeline Stage จะสามารถทำงานบน FPU ปลายทาง" ได้

Model and View Transform

เริ่มต้น โมเดลจะอยู่ใน Model Space (Model Coordinate) ของตัวเอง จากนั้นจึงถูก transform ด้วย Model Transform เพื่อไปอยู่ใน World Space (World Coordinate) เพื่อที่จะทำให้ทุกโมเดลอยู่ใน Space เดียว จากนั้น เพื่อที่จะทำให้ Projection และ Clipping ง่ายขึ้น กล้องและโมเดลทั้งหมด จะต้องถูก transform ด้วย View Transform (โดยการวางกล้องไว้ที่จุด Origin และโดยชี้มันไปยังทิศทางที่ต้องการ) แล้วจึงเรียกว่าอยู่ใน Camera หรือ Eye Space นอกจากนั้น ยังมีการ Transform อื่นๆ ที่จะสามารถเกิดขึ้นได้ใน Stage นี้อีก เช่น Vertex Blending และ Transform ที่ประมวลผลด้วย Vertex Shader



รูปที่ 2.14 แสดง Geometry Stage ในส่วน Model Transformation



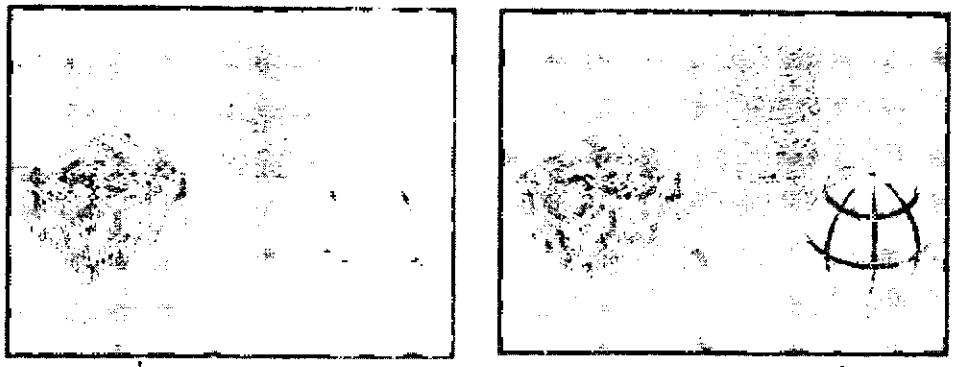
รูปที่ 2.15 แสดง Geometry Stage ในส่วน View Transformation

Lighting and Shading

โมเดลทางด้าน Geometric สามารถมีสีที่เกี่ยวข้องกับแต่ละ Vertex หรือ Texture (Image) ซึ่งสมการ Lighting จะถูกใช้เพื่อคำนวณสีที่แต่ละ Vertex ของโมเดล (โดยที่ยังไม่มีการจำลอง Phenomenon ต่างๆ เช่น True Reflection และ Shadow ใดๆ เลย)

สีของแต่ละ Vertex บนพื้นผิว จะถูกคำนวณด้วยค่าจาก Light Source และ Vertex

สีของแต่ละ Vertices ของ Triangle จะถูก interpolate บน Triangle โดยที่ Interpolation Technique นี้เรียกว่า Gouraud Shading, เพื่อที่จะทำการจำลอง Lighting Effect ที่ซับซ้อนขึ้น จะใช้ Shading Technique ระหว่าง Rasterization



รูปที่ 2.16 แสดง Geometry Stage ในส่วน Light and Shading โดยที่ ภาพซ้าย คือ ผลลัพธ์เมื่อไม่ได้มีการเปิดใช้งาน ในขณะที่ ภาพขวา คือ ผลลัพธ์เมื่อเปิดใช้งาน

Projection

เป็นการ Transform View Volume ไปเป็น "Unit Cube ที่มี Extreme Point อยู่ ณ ตำแหน่งที่ (-1, -1, -1) และ (1, 1, 1)" เรียกว่า Canonical View Volume

Projection Method คือ Orthographic และ Perspective Projection

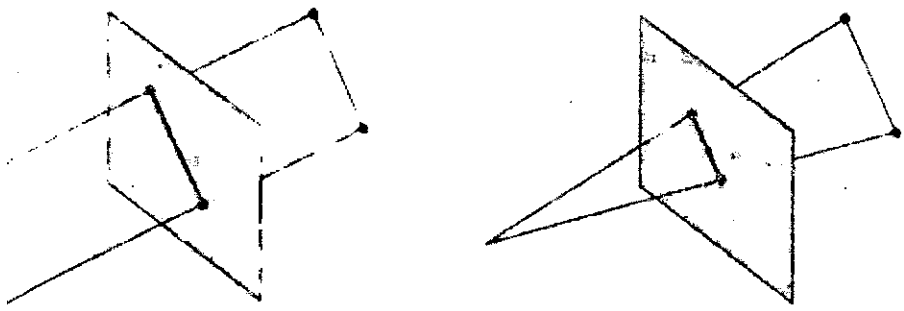
Orthographic Projection

View Volume ของ Orthographic โดยทั่วไป จะเป็น Rectangular Box แล้วถูก transform เป็น Unit Cube โดยที่คุณสมบัติหลัก คือ Parallel Line ยังคงขนาน หลังๆ ถูก transform (เป็นเพียงการทำ Translation และ Scaling เท่านั้น)

Perspective Projection

ค่อนข้างจะซับซ้อน, ยิ่ง วัตถุ อยู่ไกลกล้องเท่าไหร่, ยิ่งมี ขนาด เล็กลง หลังถูก Projection View Volume ในกรณีนี้จะเรียกว่า Viewing Frustum เป็น Pyramid ที่ถูกตัดยอดออก มีฐานเป็นสี่เหลี่ยม โดยที่จะถูก transform เป็น Unit Cube เช่นกัน

หลังจากการ Transform (ทั้ง 2 แบบ), โมเดลจะถูกเรียกว่าอยู่ใน Normal Device Coordinate ทั้งนี้ แม้ว่าจะเป็นเพียงการ transform แต่ก็ยังถูกเรียกเป็น Projection เพราะหลังจากนี้ แกน Z ไม่ได้ถูกเก็บในส่วนที่เป็นภาพ (แต่ถูกนำไปเก็บไว้ใน Z-buffer แทน)



(1) (2)

รูปที่ 2.17 แสดง Geometry Stage ในส่วน Projection โดยที่
(1) คือ Orthogonal Projection ในขณะที่ (2) คือ Perspective Projection

Clipping

เนื่องจาก Primitive มีทั้งแบบ อยู่ภายใน View Volume "ทั้งหมด" และ "บางส่วน" โดยแบ่งเป็นกรณีต่างๆ คือ อยู่ภายในทั้งหมดจะถูกส่งไปยัง Stage ถัดไป หากอยู่ภายนอกทั้งหมดจะไม่ถูกส่ง และ อยู่ภายในบางส่วนจะต้องการ Clipping

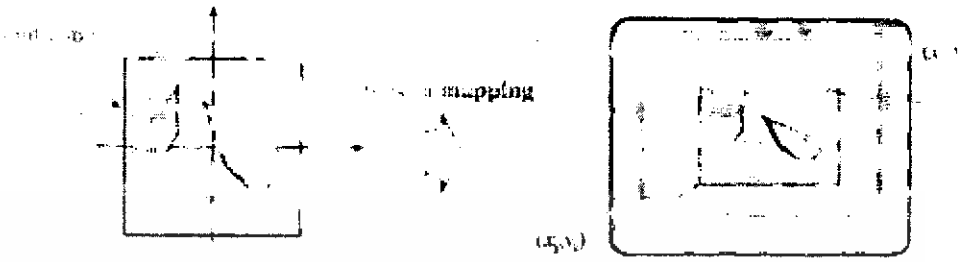
Clipping เป็น กระบวนการแทน Vertex ที่อยู่ภายนอก ด้วย Vertex ใหม่ โดยที่ประโยชน์ที่เราได้จากการทำ Viewing Transformation กับ Projection ก่อน Clipping ทำให้ "Primitive จะถูก clip กับ Unit Cube เสมอ"

รูปที่ 2.18 แสดง Geometry Stage ในส่วน Clipping

Screen Mapping

แกน X และ Y ของแต่ละ Primitive ถูก transform เพื่อสร้าง Screen Coordinates ในขณะที่ Screen Coordinates กับแกน Z เรียกรวมกันว่า Window Coordinates โดยที่มุมที่มีค่าน้อยสุดจะอยู่ที่ (x_1, y_1) และมุมที่มีค่ามากสุดจะอยู่ที่ (x_2, y_2) เมื่อ $x_1 < x_2$ และ $y_1 < y_2$

สรุปแล้ว Screen Mapping คือ การ Translation ตามด้วย Scaling, โดยที่แกน Z ไม่ได้รับผลกระทบใดๆ จากกระบวนการ Mapping เลย



รูปที่ 2.19 แสดง Geometry Stage ในส่วน Screen Mapping

The Rasterizer Stage

เป้าหมายของ Rasterizer Stage คือ "ให้ค่าสีที่ถูกต้อง ให้กับ Pixel เพื่อที่จะวาดภาพได้อย่างถูกต้อง" โดยเรียก Process นี้ว่า Rasterization หรือ Scan Conversion ซึ่งเป็น "Conversion จาก Vertices 2 มิติ ใน Screen Space กับค่า Z (ค่าความลึก) ไปเป็น Pixel บนหน้าจอ" โดยมากจะถูก Implement ในฮาร์ดแวร์

เพื่อไม่ให้มนุษย์เห็น Primitive ถูก rasterize และส่งไปยัง Screen Double Buffering จึงถูกใช้, นอกจากนั้น Stage นี้ ยังรับผิดชอบในการทำการตรวจสอบการมองเห็นโดยใช้ Z-Buffer (Depth Buffer) Algorithm คือ ถ้า ค่า Z ใหม่ มีขนาดเล็กกว่า ค่า Z เดิมใน Z-Buffer เหตุผลที่ Algorithm นี้เป็นที่นิยมอันเนื่องมาจาก "สามารถที่จะวาด Primitive ในลำดับใดๆ ก็ได้" Texturing เป็นเสมือน การแปะภาพ ลงบนวัตถุ

นอกจาก Color Buffer และ Z-Buffer ยังมี Buffer ที่สามารถถูกใช้เพื่อสร้างภาพในรูปแบบต่างๆ เช่น Stencil Buffer

2.3.2 สรุป Flow การทำงานผ่าน Pipeline

Application Stage

จะตรวจสอบและจัดการ Model ที่ transform เพื่อที่ว่า Subset ของ Model จะถูก update

Geometry Stage

จาก Stage ก่อนหน้า จะนำ View Transform ที่ถูกมาคำนวณกับ Model Transform ทำให้ Vertices และ Normal ไปใน Eye Space จากนั้นก็ถูก Lighting ที่แต่ละ Vertices ถูกคำนวณโดยใช้คุณสมบัติต่างๆ พวก Material, Textures และ Light Source

ตามด้วย Projection-โมเดลไปไว้ใน Unit Cube และทุก Primitive ที่อยู่ภายนอกจะถูกตัดทิ้ง ส่วน Primitive ที่ intersect กับ Unit Cube จะถูก clip กับ Cube แล้ว Vertices จะถูก map ไปใน Window บนหน้าจอ

ท้ายสุด จะเกิดกระบวนการพวก Per-Polygon เป็นข้อมูลส่งไปยัง Stage ถัดไป

Rasterizer Stage

ทุก Primitive ถูก rasterize (เปลี่ยนไปเป็น) Pixel บน Window โดยที่ การตรวจสอบการมองเห็น จะถูกคำนวณผ่าน Z-Buffer Algorithm

2.4 เทคนิค Culling

Culling หมายถึง การกำจัดออกจากกลุ่มก้อน โดยที่กลุ่มก้อนในที่นี้จะหมายถึง ฉากทั้งหมด ที่เราต้องการที่จะวาด ส่วนที่ถูกกำจัด คือ ส่วนของฉากที่ไม่ได้นำไปใช้สร้างภาพตอนสุดท้าย ซึ่งส่วนที่เหลือที่ไม่ได้ถูกกำจัดของฉากนั้น จะถูกส่งต่อไปใน Rendering Pipeline

สิ่งที่สนใจจะศึกษาเพื่อนำมาใช้งาน คือ เทคนิคของ Culling ที่โปรแกรมเมอร์สามารถทำการควบคุมได้อย่างเต็มที่ คือ สามารถที่จะ implement Algorithm ต่างๆ ใน Application Stage (บน CPU) ได้ ทั้งนี้ อัลกอลิทึมของ Culling แบ่งเป็น 2 ประเภท ดังนี้

Ideal Culling Algorithm

ข้อมูลที่ส่งไปให้ Pipeline จะเป็น Exact Visible Set (EVS) คือ Set ที่มองเห็นได้ของ Primitive อย่างแน่นอน โดยที่การคำนวณเพิ่มเติมใดๆ จะไม่มีผลที่ทำให้ Set เปลี่ยนแปลงอีก

Practical Culling Algorithm

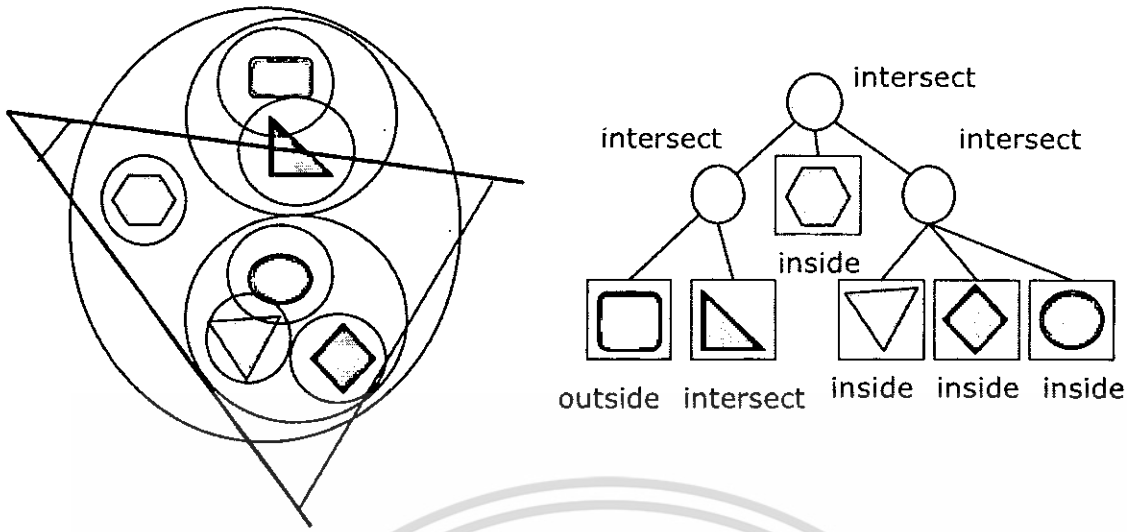
ข้อมูลที่ส่งไปให้ Pipeline จะเป็น Potentially Visible Set (PVS) ซึ่งเป็นการทำนายและประมาณของ EVS ทั้งนี้ ปัจจัยในการพิจารณาการทำงานของ PVS จะมี Conservative และ Approximate โดยที่ PVS อาจจะทำให้สร้างภาพที่ไม่ถูกต้องขึ้นมาได้

เป้าหมายของ PVS คือ ทำให้เกิดสิ่งผิดพลาดน้อยที่สุดเท่าที่จะเป็นไปได้ สิ่งสำคัญและประมาณปัจจัยทั้ง 2 เพื่อที่จะได้จุดกึ่งกลางที่เหมาะสมระหว่างคุณภาพและประสิทธิภาพที่ดีที่สุด

2.4.1 View Frustum Culling

เป็นการกำจัดวัตถุที่อยู่ภายนอกมุมกล้อง (หรือพูดให้ชัดขึ้น คือ Frustum) ออกไปจากกลุ่มข้อมูลที่จะส่งผ่านภายใน Rendering Pipeline โดยที่กระบวนการนี้จะทำการประมวลผลต่อวัตถุหรือกลุ่มของวัตถุหนึ่งๆ (Per Object หรือ Per Group) ซึ่งกระบวนการดังกล่าว

ทั้งนี้ โครงสร้างข้อมูลของวัตถุจะอยู่ในรูปของ Hierarchy ดังรูปต่อไปนี้



(1) (2)

รูปที่ 2.20 แสดง View Frustum Culling โดยที่

(1) เป็นสิ่งที่เกิดขึ้นจริงบนฉาก ซึ่ง วงกลมสีฟ้าแสดงถึง Bounding Volume ของวัตถุและกลุ่มของวัตถุ ในขณะที่ โคนสีแดง แสดงถึง ขอบเขตการแสดงผลของกล้องบนฉาก

(2) เป็นโครงสร้างข้อมูลของวัตถุในฉากซึ่งอยู่ในลักษณะของ Hierarchy รวมไปถึงผลลัพธ์เมื่อทำการตรวจสอบ Intersection กับ Frustum ของกล้อง

จากรูป วัตถุที่อยู่ภายนอก Frustum คือ สี่เหลี่ยมทึบสีฟ้าขอบมน จะถูกตัดออกจากกลุ่มที่ปรากฏในภาพ จึงไม่มีความจำเป็นที่จะส่งข้อมูลให้ฮาร์ดแวร์ไปประมวลผลต่อ ในขณะที่วัตถุที่อยู่ภายในหรือ intersect กับ Frustum นั้น จะถูกส่ง เนื่องจากว่าทั้งวัตถุ (ในกรณีของอยู่ภายใน) หรือบางส่วนของวัตถุ (ในกรณีของ intersect) ปรากฏอยู่บนภาพ

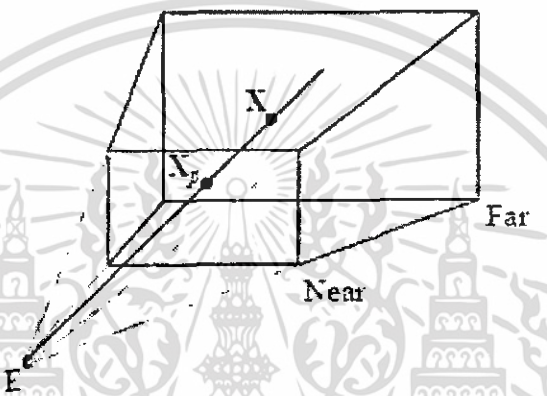
2.5 กล้องสำหรับแสดงผลภาพ

ในการแสดงผลภาพ ณ เวลาขณะใดขณะหนึ่ง จะเป็นเพียงการแสดงส่วนหนึ่งของ World โดยจะเรียก ขอบเขตของการแสดงผลนี้ว่า *View Volume* ซึ่งมีส่วนช่วยในการทำ Culling อันเป็นกระบวนการที่ตรวจสอบว่ามีวัตถุใดบ้างที่ไม่ต้องแสดงผล (หรือพูดให้ชัดว่าไม่อยู่ในขอบเขตของการแสดงผล) และการทำ Clipping ที่จะคำนวณหาส่วนที่แสดงผล (หรืออยู่ในขอบเขตการแสดงผล) ในกรณีที่มีเพียงบางส่วนของวัตถุที่อยู่ในขอบเขตการแสดงผล

ส่วนการแสดงผลข้อมูลที่อยู่ในขอบเขตนั้นจะเกิดขึ้นได้จากการโปรเจ็กต์ (Project) มันลงไปบน *View Plane* และจากการที่ View Volume นั้นถูกจำกัดขึ้นให้เป็นขอบเขตปิด ดังนั้นข้อมูลที่ถูกโปรเจ็กต์ (Project) ดังกล่าว ย่อมอยู่ในขอบเขตปิดบน View Plane โดยจะเรียกขอบเขตดังกล่าวว่า *Viewport* ซึ่งเป็นสิ่งที่ถูกนำไปวาดลงบนหน้าจอคอมพิวเตอร์จริง นั่นเอง

View Volume มาตรฐานหนึ่งที่จะนำมาใช้งาน เรียกว่า *View Frustum* โดยจะถูกสร้างจากการเลือกจุดจำลองที่อยู่ของตา (Eye Point) แล้วสร้างพีระมิดขึ้นจากระนาบ 4 ด้าน แต่ละระนาบจะประกอบไปด้วย Eye Point และเส้น Edge ของ Viewport จากนั้นจะทำการตัดพีระมิดด้วยระนาบอีก 2 ระนาบ เรียกว่า Near Plane และ Far Plane

โดยปกติทั่วไป Camera จะใช้การโปรเจ็กต์ (Project) ที่เรียกว่า Perspective Projection โดยสามารถถูกคำนวณได้จากการอินเตอร์เซคท์ (Intersect) Ray ที่เกิดจาก Origin E (Eye Point) ลากยาวไปถึง World Point X กับ View Plane ได้เป็น Intersecting Point X_p



รูปที่ 2.21 แสดง Eye Point E และ View Frustum โดยที่ World Point X ใน View Frustum ได้ถูกโปรเจ็กต์ (Project) ไปที่ X_p บน Viewport

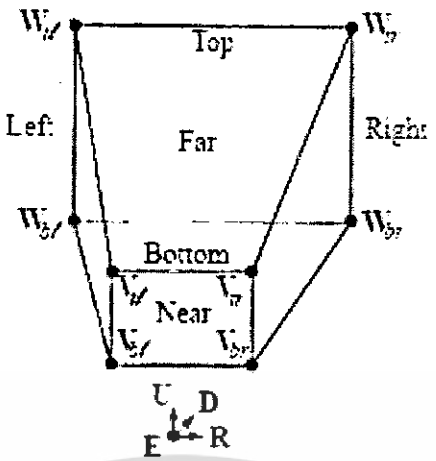
ทั้งนี้การรวมกันของ Eye Point, View Plane, Viewport และ View Frustum ทำให้เกิดเป็น Camera Model ขึ้น โดยที่ Model ดังกล่าวจะมี Coordinate System ที่เกี่ยวข้องกับมันอยู่ ดังนี้

Camera Origin คือ จุด Eye Point E

Camera Direction Vector คือ Vector ขนาดหนึ่งหน่วย D

Camera Up Vector คือ Vector ขนาดหนึ่งหน่วย U

Camera Right Vector คือ Vector ขนาดหนึ่งหน่วย R



รูปที่ 2.22 แสดงภาพของ Camera Model รวมไปถึง Camera Coordinate System และ View Frustum

จากรูปที่ Plane ทั้ง 6 ของ View Frustum นั้นจะมีชื่อกำกับไว้ คือ Near, Far, Left, Right, Bottom, Top, มีตำแหน่งของ Camera ที่ E กับแกนทิศทางของ Camera D, U และ R ระบุ และจะเห็นได้ว่า View Frustum นั้นจะประกอบไปด้วย 8 Vertices โดยที่ตรงส่วนของ Near Plane จะเรียกว่า V_{tl} , V_{bl} , V_{tr} และ V_{br} ทำนองเดียวกัน ในส่วนของ Far Plane ก็จะเรียกว่า W_{tl} , W_{bl} , W_{tr} และ W_{br} ทั้งนี้จะมีสมการที่ใช้สำหรับแต่ละ Vertices คือ

$$V_{bl} = E + d_{\min}D + u_{\min}U + r_{\min}R$$

$$V_{tl} = E + d_{\min}D + u_{\max}U + r_{\min}R$$

$$V_{br} = E + d_{\min}D + u_{\min}U + r_{\max}R$$

$$V_{tr} = E + d_{\min}D + u_{\max}U + r_{\max}R$$

$$W_{bl} = E + \frac{d_{\max}}{d_{\min}}(d_{\min}D + u_{\min}U + r_{\min}R)$$

$$W_{tl} = E + \frac{d_{\max}}{d_{\min}}(d_{\min}D + u_{\max}U + r_{\min}R)$$

$$W_{br} = E + \frac{d_{\max}}{d_{\min}}(d_{\min}D + u_{\min}U + r_{\max}R)$$

$$W_{tr} = E + \frac{d_{\max}}{d_{\min}}(d_{\min}D + u_{\max}U + r_{\max}R)$$

เมื่อ $d_{\min}$ คือ ระยะทางจาก Camera Location ไปยัง Near Plane (ในทิศทาง D)

$d_{\max}$ คือ ระยะทางจาก Camera Location ไปยัง Far Plane (ในทิศทาง D)

$u_{\min}$ คือ ระยะทางจาก Camera Location ไปยัง Bottom Plane (ในทิศทาง U)

$u_{\max}$ คือ ระยะทางจาก Camera Location ไปยัง Top Plane (ในทิศทาง U)

r_{min} คือ ระยะทางจาก Camera Location ไปยัง Left Plane (ในทิศทาง R)

r_{max} คือ ระยะทางจาก Camera Location ไปยัง Right Plane (ในทิศทาง R)

ในกระบวนการทำ Object Culling นั้น จะถูกอิมพลีเมนต์ (Implement) จะเป็นการ Culling โดยใช้ Plane ณ เวลาหนึ่งเท่านั้น โดย Frustum Plane ถูกให้ค่า Unit-Length Normal ที่ชี้ไปใน Frustum, Bounding Volume จะถูกตรวจสอบกับ Frustum Plane ถ้าเกิดว่า Bounding Volume อยู่ภายนอก Plane ใด Plane หนึ่ง จะถือว่าวัตถุนั้นไม่อยู่ในขอบเขตการแสดงผลและถูกกำจัดออกจากระบบการแสดงผล ซึ่งในการทำงานดังกล่าวนี้ จะมีสมการสำหรับ Frustum Plane ทั้ง 6 ดังต่อไปนี้

Near Plane มี D เป็น Unit-Length Normal ที่ชี้เข้าภายใน View Frustum ได้เป็นจุดบน Plane เป็น $E + d_{min}D$ ทำให้ได้สมการของ Plane เป็น

$$D \cdot X = D \cdot (E + d_{min}D) = D \cdot E + d_{min}$$

Far Plane มี –D เป็น Unit Length Normal ที่ชี้เข้าภายใน View Frustum ได้เป็นจุดบน Plane เป็น $E + d_{max}D$ ทำให้ได้สมการของ Plane เป็น

$$-D \cdot X = -D \cdot (E + d_{max}D) = -(D \cdot E + d_{max})$$

ในกรณีของ 4 Plane ที่เหลือจะคำนวณตามขั้นตอนดังต่อไปนี้ โดยจะขอยกตัวอย่างของ Left Plane เป็นตัวอย่าง คือ จากการที่ Left Plane มี 3 จุด คือ E, V_{tl}, และ V_{bl} ที่อยู่ภายบนมัน ทำให้สามารถที่จะคำนวณ Normal Vector ที่ชี้เข้าหา Frustum ได้ดังนี้

$$\begin{aligned}(V_{bl} - E) \times (V_{tl} - E) &= (d_{min}D + u_{min}U + r_{min}R) \times (d_{min}D + u_{max}U + r_{min}R) \\ &= (d_{min}D + r_{min}R) \times (u_{max}U) + (u_{min}U) \times (d_{min}D + r_{min}R) \\ &= (d_{min}D + r_{min}R) \times ((u_{max} - u_{min})U) \\ &= (u_{max} - u_{min})(d_{min}D \times U + r_{min}R \times U) \\ &= (u_{max} - u_{min})(d_{min}R - r_{min}D)\end{aligned}$$

Left Plane มี Unit Length Normal ที่ชี้เข้าภายใน และสมการเป็น

$$N_l = \frac{d_{min}R - r_{min}D}{\sqrt{d_{min}^2 + r_{min}^2}} \text{ กับ } N_l \cdot (X - E) = 0$$

Right Plane มี Unit Length Normal ที่ชี้เข้าภายใน และสมการเป็น

$$N_r = \frac{-d_{min}R - r_{max}D}{\sqrt{d_{min}^2 + r_{max}^2}} \text{ กับ } N_r \cdot (X - E) = 0$$

Bottom Plane มี Unit Length Normal ที่ชี้เข้าภายใน และสมการเป็น

$$N_b = \frac{d_{\min}U - u_{\min}D}{\sqrt{d_{\min}^2 + u_{\min}^2}} \text{ กับ } N_b \cdot (X - E) = 0$$

Top Plane มี Unit Length Normal ที่ชี้เข้าภายใน และสมการเป็น

$$N_t = \frac{-d_{\min}U - u_{\max}D}{\sqrt{d_{\min}^2 + u_{\max}^2}} \text{ กับ } N_t \cdot (X - E) = 0$$

2.6 3D File Format

การพัฒนา 3D File Format นั้นเพื่อให้มีความเหมาะสมในการใช้งานสูงสุด โดยสามารถแบ่งพื้นฐานแนวคิดในการสร้างอนิเมชั่น (animation) ของโมเดลสามมิติ (3D Model) เป็นสองแนวคิดหลักๆคือ Keyframe Animation และ Skeletal Animation ซึ่งการนำไปใช้งานนั้นจะขึ้นอยู่กับความต้องการของผู้ใช้

ประสิทธิภาพของแต่ละ File Format ต่างมีองค์ประกอบที่มีข้อดีและข้อเสียในการนำไปพิจารณา เช่น ขนาดของไฟล์, ความรวดเร็วในการเรียกใช้งาน, รูปแบบการเก็บข้อมูล, เป็นต้น

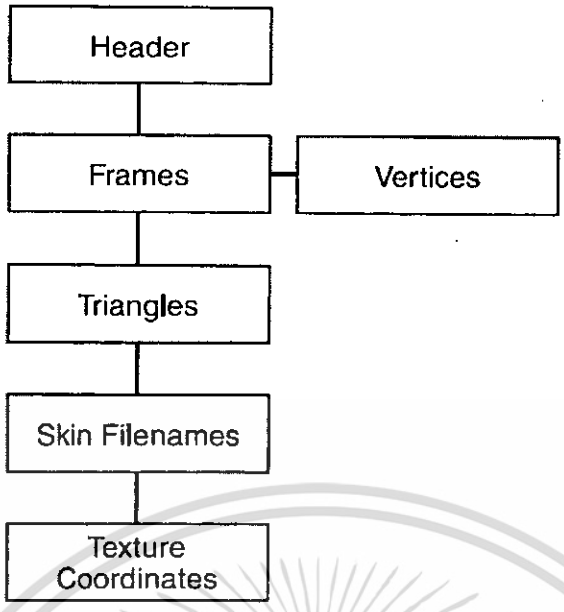
โมเดลสามมิติสามารถสร้างได้จากเครื่องมือ (Tool) หลายชนิด เช่น 3D Studio Max, Milk Shape 3D, Maya เป็นต้น ซึ่งโมเดลสามมิติที่ได้สร้างขึ้นก็จะมี File Format ตามเครื่องมือที่ใช้ในการพัฒนาขึ้นมา ทางเราจึงทำการศึกษาในข้อดีและข้อเสียของแต่ละ File Format เพื่อนำมาพัฒนาขึ้นเป็น File Format ของตนเอง เพื่อให้เหมาะสมกับการใช้งานมากยิ่งขึ้น โดย File Format ที่ได้นำมาศึกษานั้นเป็น File Format ที่รู้จักและใช้กันอย่างแพร่หลาย ซึ่งมีรายละเอียดดังนี้

2.2.1 MD2 File Format

MD2 model file format ได้ถูกนำเสนอโดยบริษัท id Software ถูกออกแบบมาเพื่อให้ง่ายต่อการใช้และเข้าใจ องค์ประกอบของ MD2 model มีส่วนหลักๆดังนี้

- รูปแบบโมเดลเป็นข้อมูลเชิงเรขาคณิต อ้างอิงจากโครงสร้างสามเหลี่ยม (triangle)
- มีการทำงานของอนิเมชั่นแบบอ้างอิงเฟรม (Frame-by-frame animations)

มีการเก็บข้อมูลของพื้นผิว (textures) แยกกับไฟล์ที่เก็บข้อมูลส่วนอื่นๆ โดย 1 MD2 model สามารถมีพื้นผิวได้เพียงพื้นผิวเดียว มีนามสกุลของไฟล์ (file's extension) เป็น "md2" เก็บข้อมูลเป็น binary file แบ่งเป็นสองส่วนหลักคือ ส่วนที่เป็นหัวไฟล์ (header) และส่วนที่เป็นเนื้อไฟล์ (data) ซึ่งส่วนหัวไฟล์ทำหน้าที่เก็บรายละเอียดของไฟล์และเนื้อข้อมูลที่ต้องใช้ทั้งหมด



รูปที่ 2.23 แสดงโครงสร้างของ md2 File Format

The MD2 Header

Header จะอยู่ในส่วนหัวของไฟล์ โดยมีโครงสร้างดังนี้

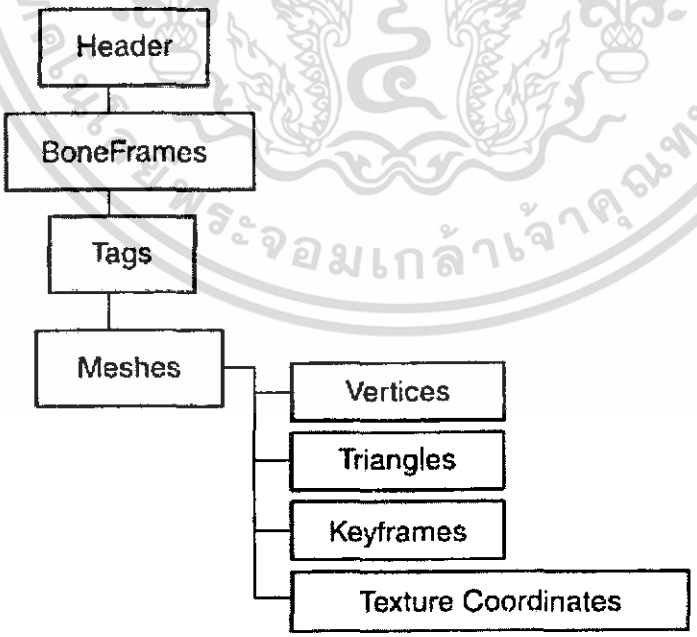
```
typedef struct
{
        int ident;          /* magic number: "IDP2" */
        int version;        /* version: must be 8 */
        int skinwidth;      /* texture width */
        int skinheight;     /* texture height */
        int framesize;      /* size in bytes of a frame */
        int num_skins;      /* number of skins */
        int num_vertices;   /* number of vertices per frame */
        int num_st;         /* number of texture coordinates */

        int num_tris;       /* number of triangles */
        int num_glcmds;     /* number of opengl commands */
        int num_frames;     /* number of frames */
        int offset_skins;   /* offset skin data */
        int offset_st;      /* offset texture coordinate data */
        int offset_tris;    /* offset triangle data */
        int offset_frames;  /* offset frame data */
        int offset_glcmds;  /* offset OpenGL command data */
        int offset_end;     /* offset end of file */
} md2_header_t;
```

- ident คือ magic number ของไฟล์ ใช้ในการระบุชนิดของไฟล์ โดยจะต้องมีค่าเป็น 844121161 หรือ ว่าเป็น "IDP2"
- version คือ เวอร์ชั่นของ file format และจะต้องมีค่าเท่ากับ 8
- skinwidth และ skinheight ใช้ระบุถึงความกว้างและความยาวของพื้นผิวโมเดลตามลำดับ
- framesize คือขนาดของเฟรมในหน่วยไบต์
- num_skins คือ จำนวนของความสัมพันธ์ของพื้นผิวโมเดล
- num_vertices คือ จำนวนของ vertices ต่อ หนึ่งเฟรม

- num_st คือ จำนวนของ texture coordinates
- num_tris คือ จำนวนของ triangles
- num_glmds คือ จำนวนของคำสั่ง OpenGL
- num_frames คือ จำนวนของเฟรมทั้งหมดในโมเดล
- offset_skins คือ ตัวแปรที่ระบุตำแหน่งในหน่วยไบต์ตั้งแต่เริ่มต้นไฟล์ถึง texture data
- offset_st คือ ตัวแปรที่ระบุตำแหน่งของ texture coordinate
- offset_tris คือ ตัวแปรที่ระบุตำแหน่งของ triangle data
- offset_frames คือ ตัวแปรที่ระบุตำแหน่งของ frame data
- offset_glcmds คือ ตัวแปรที่ระบุตำแหน่งของคำสั่ง OpenGL
- offset_end คือ ตัวแปรที่ระบุตำแหน่งของส่วนจบไฟล์

2.2.2 MD3 File Format

MD3 model File Format ได้ถูกนำเสนอโดยบริษัท id Software โดยใช้ในเกม Quake3 โดย MD3 เป็น File Format ที่พัฒนาต่อมาจาก MD2 File Format โดยมีจุดเด่นที่มีความสามารถทางด้านการเชื่อมต่อโมเดลรวมเป็นไฟล์เดียว (multiple meshes) ผ่านทางการใช้ tags ทำให้โมเดลที่สร้างขึ้น ประกอบด้วยหลายๆ part และแต่ละ part สามารถที่จะทำ animation ต่างๆกัน ในเวลาเดียวกัน โดย md3 File Format มีโครงสร้างดังนี้



ภาพที่ 2.24 แผนภาพของโครงสร้างของ MD3 File Format

The MD3 Header

MD3 Header จะมีลักษณะคล้ายกับ MD2 Header มาก โดยประกอบไปด้วย ID, a version number, information ของ chucks ใน file โดยมีโครงสร้างดังนี้

```
Struct   SMD3Header
{
        int m_iId;
        int m_iVersion;
        char m_cFilename[68];
        int m_iNumFrames;
        int m_iNumTags;
        int m_iNumMeshes;
        int m_iMaxSkins;
        int m_iHeaderSize;
        int m_iTagOffset;
        int m_iMeshOffset;
        int m_iFileSize;
};
```

- int m_iId เป็นตัวแปรที่ใช้ระบุไอดีของไฟล์ MD3 โดยค่าต้องเป็น IDP3 เท่านั้น
- int m_iVersion เป็นตัวแปรที่ใช้ระบุเวอร์ชั่น โดยต้องเป็น 15 เท่านั้น
- char m_cFilename[68] เป็นตัวแปรที่ระบุชื่อของไฟล์ MD3
- int m_iNumFrames เป็นตัวแปรที่ใช้เก็บหมายเลข keyframes ที่ใช้ในการทำอนิเมชั่น
- int m_iNumTags เป็นตัวแปรที่ใช้เก็บหมายเลข Tags ของ MD3 โมเดล
- int m_iNumMeshes เป็นตัวแปรที่ใช้เก็บ Mesh, geometry data, chuck
- int m_iMaxSkins เป็นตัวแปรที่ใช้เก็บค่าสูงสุดของ skins ที่ใช้ได้
- int m_iHeaderSize เป็นตัวแปรที่ใช้เก็บขนาดของหัวไฟล์ซึ่งเป็น offset ในหน่วย bytes
- int m_iTagOffset เป็นตัวแปรที่ใช้บอกตำแหน่งของ Tags เป็น bytes
- int m_iMeshOffset เป็นตัวแปรที่คล้าย tagoffset ใช้หาตำแหน่งของ Mesh
- int m_iFileSize เป็นตัวแปรที่ใช้เก็บขนาดทั้งหมดของไฟล์ MD3

Boneframe

แต่ละ boneframe มีทั้งหมด 56 ไบต์ โดยเก็บข้อมูลเกี่ยวกับ Bounding box ของโมเดลแต่ละ keyframes ของอนิเมชั่น จะมี 1 boneframe โดย 12 ไบต์แรกเป็นค่าน้อยสุดของแกน x,y,z ของ bounding box และ 12 ไบต์หลังคือค่าสูงสุด

MD3 TAG Structure

ใช้เชื่อมต่อโมเดลเข้าด้วยกัน แต่ละ tag ประกอบด้วยโรเทชั่นเมตริกสามคูณสาม (3 * 3 rotation matrix) , ตำแหน่งของเวกเตอร์ และชื่อขนาด 64 คาร์เรกเตอร์

```
struct SMd3Tag
{
        char m_cName[64];
        CVector3 m_vecPos;
        CQuaternion m_qRot;
};
```

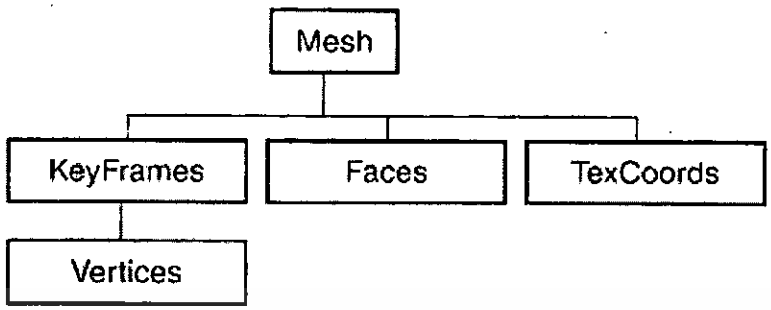
- m_cName เป็นตัวแปรชนิดสตริง ใช้เก็บชื่อ Tag
- m_qRot เป็นตัวแปรของโรเทชั่นเมตริกที่เก็บค่าการหมุนของแต่ละเฟรม โดยจะเก็บค่าเป็นแบบเมตริกสามคูณสาม เพื่อให้ง่ายต่อการเปลี่ยนแปลงตัวแปรเป็น quaternion การเปลี่ยนตัวแปรจากเมตริกเป็นโรเทชั่นเมตริก ซึ่งจะช่วยในเรื่องของ quaternion interpolation เมื่อมีการทำโมเดลอนิเมชั่น
- m_vecPos คือค่าตำแหน่งของเวกเตอร์เป็นการเก็บตำแหน่งของ tag จะถูกใช้เพื่อเก็บค่าของโมเดลที่เชื่อมต่อกัน ค่าของตำแหน่งของเวกเตอร์จะเปลี่ยนเมื่อส่วนใดส่วนหนึ่งของโมเดลเปลี่ยนไป

Meshes

Meshes ของ MD3 ประกอบด้วยทุกๆ vertex, animation, face, texture information ซึ่งหมายถึงทุกอย่างบนหน้าจอ ข้อแตกต่างระหว่าง MD3 ไฟล์กับ MD2 ไฟล์ก็คือ MD3 ไฟล์สามารถเก็บได้ทีละหลายๆ Mesh ในไฟล์เดียว (Multiple Mesh)

Multiple Meshes จะมีประโยชน์มากเมื่อต้องใช้อนิเมชั่นหลายๆรูปแบบในการแสดงผลคู่กับการเปลี่ยนแปลงหลายๆส่วนของโมเดล

The MD3 mesh layout



ภาพที่ 2.25 แผนภาพของโครงสร้างของ MD3 Mesh

2.2.3 MS3D File Format

Milkshape 3D (ms3d) File format ถูกสร้างขึ้นโดยบริษัทซอฟท์แวร์เล็กๆในประเทศสวิตเซอร์แลนด์ที่ชื่อว่า chUmbaLum sOft ซึ่ง Milkshape มีพื้นฐานการสร้างจาก low-polygon โมเดลเพื่อใช้ในส่วนเอนจิ้นของเกม Half-Life

File format ของโมเดล MilkShape 3D นี้มีจุดเด่นในเรื่องราคาที่ถูกและพัฒนาต่อได้ไม่ยากจึงเป็นที่นิยมอย่างมาก MilkShape 3D ประกอบไปด้วยความสามารถพื้นฐานทั้งหมดเช่น การเลือก (Selection), การหมุน (Rotation), การย่อขยาย (Scaling), extruding, turning edge, subdividing และอื่นๆ MilkShape 3D สามารถที่จะแก้ไขข้อมูลในระดับล่าง (low-level) ได้โดยใช้ vertex และ face tool อีกทั้งยังสามารถที่จะส่งออกไฟล์ได้อีกกว่า 70 File format

การทำอนิเมชั่นของ MilkShape 3D เป็นการใช้แนวคิดแบบ skeletal animation ซึ่งสามารถการ morph อนิเมชั่นและส่งออกเป็น skeletal อนิเมชั่นได้ โดยที่โครงสร้างของ MilkShape 3D File จะเรียงลำดับดังนี้

หมายเหตุ : เนื่องจากจะอธิบายโครงสร้างของแต่ละส่วนด้วยโครงสร้างของภาษา C/C++ จึงขอแสดงส่วนที่มีค่าเริ่มต้น (defined) Max Values, Flags, และ Types ดังนี้

```
// ค่าสูงสุดของแต่ละส่วนประกอบ
#define MAX_VERTICES        8192
#define MAX_TRIANGLES       16384
#define MAX_GROUPS 128
#define MAX_MATERIALS       128
#define MAX_JOINTS          128
#define MAX_KEYFRAMES       216     // สามารถเพิ่มได้เมื่อต้องการ
```

```
// ค่า Flag ใช้บอกสถานะ ซึ่ง MilkShape Editor จะนำไปใช้แต่ไม่ใช่ข้อมูลที่จำเป็นในการเรนเดอร์โมเดลตามปกติ
#define SELECTED    1
#define HIDDEN      2
#define SELECTED2   4
#define DIRTY  8
```

```
// ชนิดของข้อมูล
#ifndef byte
typedef unsigned char byte;
#endif  // ไบต์
#ifndef word
typedef unsigned short word;
#endif  // เวิร์ด
```

Header

MilkShape 3D File Format จะเริ่มต้นด้วยหัวไฟล์ที่มีหน้าที่ตรวจสอบว่าไฟล์นั้นมีจริงและถูกต้อง โดยจะดูจากค่า Identifier (ID) โดยหัวไฟล์นั้นจะมีขนาด 14 ไบต์และแบ่ง 10 ไบต์แรกเป็นตัวแปรที่ระบุ Character Identification String และอีก 4 ไบต์หลังเป็นตัวแปรที่ระบุ Version Number ของไฟล์ ซึ่งหัวไฟล์ที่ถูกต้องจะมีค่า Character Identification String เป็น “MS3D000000” เท่านั้นและมีค่า Version Number เป็น 3 หรือ 4 เท่านั้น จึงจะถือว่าไฟล์นั้นเป็น MS3D File format อย่างถูกต้อง

Vertices

ส่วนของ Vertex จะอยู่ต่อจากหัวไฟล์โดยจะเริ่มจาก 2 ไบต์แรกเป็นตัวแปรแบบ unsigned integer ซึ่งเป็นตัวแปรที่ใช้เก็บจำนวนของ Vertices ในโมเดล ซึ่งตัวแปร Vertices นี้นอกจากจะเก็บค่า X, Y, Z Coordinate Plane แล้ว Vertices ของ MilkShape ยังเก็บเพิ่มอีก 1 ไบต์ เป็นชนิด Signed integer ไว้เก็บจำนวนของ Vertex's Bone โดยหาก Vertex's Bone มีค่าเป็น -1 แสดงว่า Vertex ไม่มี Bone ที่ถูกแนบต่อมันและ Vertex นี้จะไม่ได้รับผลกระทบใดๆ ระหว่างกระบวนการของ Animation

```
struct SMs3dVertex
{
    unsigned char m_ucFlags;        //Editor flags, unused for the loader
    CVector3 m_vVert;               //X,Y,Z coordinates
    char m_cBone;                   //Bone ID (-1 = no bone)
    unsigned char m_ucUnused;
};
```

Faces

Faces จะเริ่มด้วยตัวแปรขนาด 2 ไบต์ชนิด unsigned integer ต่อท้ายจาก vertices เป็นตัวแปรที่ใช้เก็บจำนวนของ Triangle ที่อ่านได้จากไฟล์แล้วก็ตามด้วยชุดข้อมูลที่เก็บค่ารายละเอียดต่างๆ ประกอบด้วย editor flags, vertex indexes, texture coordinates, grouping info และ vertex normal information ซึ่งมีโครงสร้างดังนี้

```
struct SMs3dTriangle
{
    unsigned short m_usFlags;               //Editor flags
    unsigned short m_usVertIndices[3];      //Vertex indexes
    CVector3 m_vNormals[3];                 //Vertex normals;
    float m_fTexCoords[2][3];               //Texture coordinates
    unsigned char m_ucSmoothing;            //Smoothing group
    unsigned char m_ucGroup;                //Group index
};
```

Meshes

MilkShape 3D ได้รวมกลุ่มของ Triangle เข้าด้วยกันเป็นแผงเรียกว่า Meshes หรือ Groups ทำให้เกิดจุดเด่นของ MilkShape 3D ก็คือความยืดหยุ่นในการใช้ส่วนประกอบต่างๆของโมเดลดนละส่วนได้ ไม่ว่าจะเป็น Textures หรือ Materials และการ render เพียงบางส่วนของโมเดลก็สามารถทำได้ โดยเริ่มจากตัวแปรขนาด 2 ไบต์ระบุจำนวนของ Mesh แล้วตามด้วยชุดข้อมูลขนาด 35 ไบต์และตัวแปรบอกจำนวนของ Tringle indexes ซึ่งเป็นค่าที่ไม่คงที่ อาจมากน้อยแล้วแต่ Groups ในการนำเสนอข้างบนนี้เป็นเพียงให้เห็นโครงสร้าง ถ้าเป็นในการประกาศ Structure เพื่อใช้ใน Real-time Application เราควรที่จะใช้วิธีการของ dynamic Memory เราควรที่จะใช้วิธีการของ dynamic Memory Allocation ในแต่ละ Group เพื่อใช้เก็บ Indexes ดังกล่าว แต่ละค่าภายใน Array เป็น Index ไปยัง Array ของ Triangles

Materials

Materials เป็นส่วนที่ใช้ควบคุมวิธีการ Renderer จัดการกระบวนการ Texturing และ Lighting ของโมเดล จาก Texture ไป Color ไป Transparency และ Material จะจัดการควบคุมทั้งหมด เริ่มด้วยตัวแปรขนาด 2 ไบต์ ชนิด unsigned integer เป็นตัวบอกจำนวนของ Mesh แล้วก็ตามด้วยชุดข้อมูลจริงๆ ซึ่งมีการเก็บ อยู่ 6 Material Properties ดังตารางที่ 2.1

Material Properties	Description
Ambient	เป็น RGBA Color ใช้เพื่อพิจารณาว่า Polygon ที่ใช้ Material นี้จะตอบสนองต่อกระบวนการ Lighting ในฉากอย่างไร
Diffuse	เป็น RGBA Color ใช้เพื่อพิจารณาว่า Polygon ที่ใช้ Material นี้จะตอบสนองต่อกระบวนการ Lighting ใน ฉากอย่างไร
Specular	เป็น RGBA Color ของ Specular Highlights หรือ Shiny places บน Mesh ที่ใช้ Material
Emissive	เป็น RGBA Color ที่ระบุว่า Material จะ emit light ได้ Intense อย่างไร
Shininess	เป็น RGBA Color ที่ระบุว่า Specular highlights จะ Shiny เท่าไหร่
Transparency	เป็นค่า Transparency ของ Material เลย

ตารางที่ 2.1 แสดง Material Properties ในส่วน Material ของ MS3D File Format

Animation

ส่วนสุดท้ายคือส่วนของอนิเมชั่นซึ่งเป็นส่วนที่พัฒนาได้ยากที่สุดประกอบไปด้วย structure หลักสองส่วนคือ Keyframes และ Joint โดย Keyframes มีหน้าที่ในการเก็บค่า Rotation และ Translation ของ Keyframes มีโครงสร้างดังนี้

```
struct SMs3dKeyFrame
{
        float m_fTime;
        float m_fParam[3];
};
```

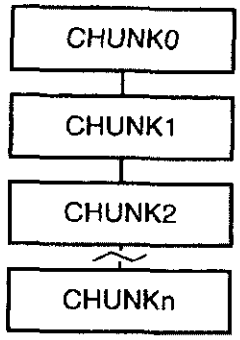
- m_ftime เป็นตัวแปรที่เก็บเวลาในหน่วยวินาที โดยวินาทีที่เก็บคือวินาทีที่ตำแหน่งของ joint ถูกเซ็ตที่ keyframe นั้นๆ
- m_fParam[3] เป็นตัวแปรที่เก็บค่าการหมุนและการเคลื่อนที่รอบแกน X, Y และ Z

โดยแต่ละ joint จะประกอบด้วยเซตของ keyframes ที่มีค่าของการหมุนและการเคลื่อนที่รวมถึงค่าที่สำคัญอีกเป็นจำนวนมาก ดังต่อไปนี้

```
struct SMs3dJoint
{
    unsigned char m_ucpFlags;              //Editor flags
    char m_cName[32];                      //Bone name
    char m_cParent[32];                    //Parent name
    float m_fRotation[3];                  //Starting rotation
    float m_fPosition[3];                  //Starting position
    unsigned short m_usNumRotFrames;       //Number of rotation frames
    unsigned short m_usNumTransFrames;     //Number of translation frames
    SMs3dKeyFrame * m_RotKeyFrames;        //Rotation keyframes
    SMs3dKeyFrame * m_TransKeyFrames;      //Translation keyframes

    short m_sParent;                       //Parent joint index
    CMatrix4X4 m_matLocal;
    CMatrix4X4 m_matAbs;
    CMatrix4X4 m_matFinal;
    unsigned short m_usCurRotFrame;
    unsigned short m_usCurTransFrame;
};
```

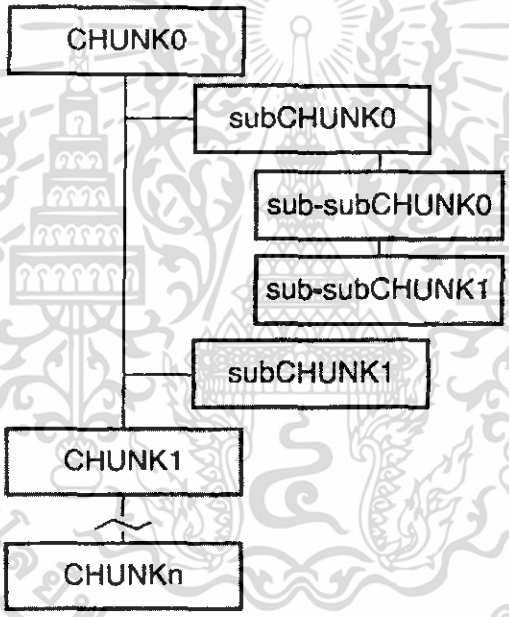
2.2.4 3DS File Format

3DS File Format เป็น File Format ที่นิยมและน่าสนใจเนื่องจากมีการใช้หลักการของ Chunk ซึ่งเป็นเอกลักษณ์ของ 3DS File Format โดยหลักการของ Chunk โดยทั่วไปคือ ในแต่ละ Chunk จะทำหน้าที่ในการระบุ identifier, length, group of bytes และจะมีหัวไฟล์เพื่อทำหน้าที่ในการบอกว่าตำแหน่งของ Chunk นั้นๆอยู่ตำแหน่งไหนหรือบอกได้ว่า Chunk นั้นๆได้ถูกนำมาใช้หรือไม่ โดยแต่ละ Chunk ไม่จำเป็นจะต้องเรียงลำดับสามารถสลับตำแหน่ง Chunk ไปมาได้ แต่สำหรับ 3DS File format จะไม่มีหัวไฟล์ในการบอกตำแหน่งของ Chunk หรือบอกว่ามีจำนวนกี่ Chunk ดังรูป



รูปที่ 2.26 แสดง 3ds File Format ที่ประกอบด้วย chunk

ซึ่งแต่ละ chunk สามารถที่จะมี Chunk ย่อยๆ (Sub chunk) ได้อีกหลายๆ Chuck โดยไม่จำเป็นต้องมีการเรียงลำดับ ดังรูป



รูปที่ 2.27 แสดง Chunk ที่ประกอบด้วย Sub-chunk

ซึ่ง Chunk ใน 3DS File format จะประกอบหัวไฟล์ด้วย 6 ไบต์ใช้เก็บ Identifier และความยาวของ Chunk ซึ่งเป็นส่วนที่ใช้ในการที่จะ *manipulate* และ *process* ข้อมูล Chunk สามารถเก็บ Meshes หรือ Triangle ของวัตถุ ซึ่ง sub-chunk ของ Chunk จะเก็บค่าของ vertices, texture coordinates, material ของ Meshes

การใช้ Chunk มีข้อดีหลายประการ เช่น อนุญาตให้ข้ามข้อมูลในส่วนของ Chunk ที่ไม่ต้องการได้ ทำให้ง่ายต่อการใช้เนื่องจากการใช้ chunk ไม่มีลำดับในการเรียกใช้ข้อมูล ทำให้สามารถที่จะแก้ไขเพิ่มเติมหรือค้นหา chunk ได้

Header

Header ของ 3ds File Format จะมี structure ดังนี้

```
struct S3dsChunkHeader
{
        unsigned short m_usID;
        unsigned int m_uiLength;
};
```

เริ่มจาก 2 ไบต์แรกของ Chunk จะเก็บค่า Identifier ซึ่งเป็นตัวแปรที่บอกว่า Chunk นั้นเก็บข้อมูลประเภทอะไร โดย Chunk สามารถที่จะเก็บข้อมูลประเภทอะไรก็ได้ ไม่ว่าจะเป็น Vertex data, Face data, Animation data หรือข้อมูลที่ไม่จำเป็น และตัวแปรถัดมาจะเก็บความยาวของ Chuck ทั้งหมดรวมถึง Header และ Sub-chunk ด้วย

ID Number	Use
0x4D4D	Used at the start of the file to signify that the file is a 3ds file.
0x0002	Holds the version number of the file.
0x4000	Contains an "object" such as a mesh, camera, or light. Each 0x4000 chunk contains sub-chunks with vertex, texture coordinate, and other information.
0x4100	A sub-chunk of 0x4000. A 0x4100 chunk contains everything needed to build a mesh of triangles.
0x4110	Contains the vertices for the object. It is a sub-chunk of 0x4100.
0x4120	Also a sub-chunk of 0x4100, a 0x4120 chunk contains the face information, including the vertex indexes that tell which vertices make up each face.
0x4130	Another sub-chunk of 0x4100, 0x4130 contains information on which materials are to be applied to which faces.
0x4140	Even another sub-chunk of 0x4100, this chunk contains the texture coordinates that allow a face to be texture mapped.
0xAFFF	A material definition is found inside 0xAFFF, colors for ambient, diffuse, and specular materials, as well as shininess and transparency are in this chunk.
0xA000	A sub-chunk of 0xAFFF that contains the material's name.
0xA010	A sub-chunk of 0xAFFF as well, contains the ambient color for the material.
0xA020	Another sub-chunk of 0xAFF, 0xA020 contains the diffuse color of the material.
0xA030	A fourth sub-chunk of 0xAFF, this contains the specular highlight colors for the specific material.
0xA040	Yet another 0xAFFF sub-chunk, this time contains the shininess of the material.
0xA050	Again, a sub-chunk of 0xAFFF that controls how transparent or opaque a material is.
0xA200	0xA200 is also a sub-chunk of the 0xAFFF chunk. This chunk stores the filename of the texture map or skin for the current material.

รูปที่ 2.28 แสดง Common 3ds Chunks Identifier

Data File

-Header 0x4D4D

Header ของส่วน Data file นั้น 6 ไบต์แรก จะเก็บค่า chunk header โดยมี ID เป็น 0x4d4d มีความยาวเป็น ความยาวทั้งหมดของไฟล์ซึ่ง chunk เก็บ sub-chunk ทั้งหมด ในส่วนหัวไฟล์นี้ไม่มีการนำไปใช้ใดๆเกิดขึ้น

-Version chunk 0x0002

เป็น sub-chunk ตัวแรกโดยมีความยาว 10 ไบต์และเก็บ Format version ใน 4 ไบต์แรก

-Objects in the 0x4000 chunk

ทำการเก็บข้อมูลของ Object ซึ่งอาจจะประกอบด้วย Triangle mesh, Light, Camera หรือ เก็บ Window setting สำหรับ Editor

-Triangular Mesh 0x4100

Mesh คือกลุ่มของ Polygon ที่ทำการต่อกันเป็นพื้นผิว (surface) ซึ่ง Triangular mesh คือ Mesh ที่ประกอบด้วย Triangles เท่านั้น

```
Struct S3dsMesh
{
    char m_cName[256];
    vector<S3dsVertex *> m_vVertices;
    vector<S3dsTexCoord *> m_vTexCoords;
    vector<S3dsFace *> m_vFaces;
    vector<S3dsObjMat *> m_vMaterials;
    unsigned short m_usNumFaces;
};
```

-Materials 0xAFFF

Materials chunk เก็บข้อมูลสี Ambient, Diffuse, Specular และข้อมูลอื่นๆ ดังนี้

```
struct S3dsMaterial
{
    char m_cName[256];          // Name of material
    float m_fAmbient[4];        // Ambient color
    float m_fDiffuse[4];        // Diffuse color
    float m_fSpecular[4];       // Specular color
    float m_fShininess;         // Material shininess
};
```

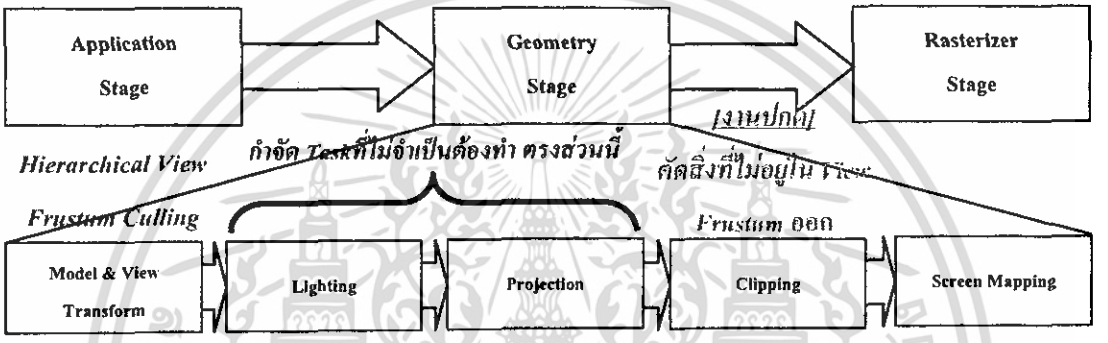
บทที่ 3
การออกแบบ, พัฒนา และทฤษฎีเพิ่มเติม

3.1 ผลลัพธ์และขอบเขตของโครงงาน

(1) สร้าง 3D File Format เพื่อนำเข้าข้อมูลจากโปรแกรมสร้างโมเดล 3 มิติ

(2) สร้าง Graphic Engine ที่มีการใช้งาน Hierarchical Viewing Frustum Culling

(3) สร้าง Demo ขึ้นมา เพื่อตรวจสอบดูผลการทำงาน



รูปที่ 3.1 แสดงถึงประโยชน์ที่ได้จากการใช้งาน View Frustum Culling

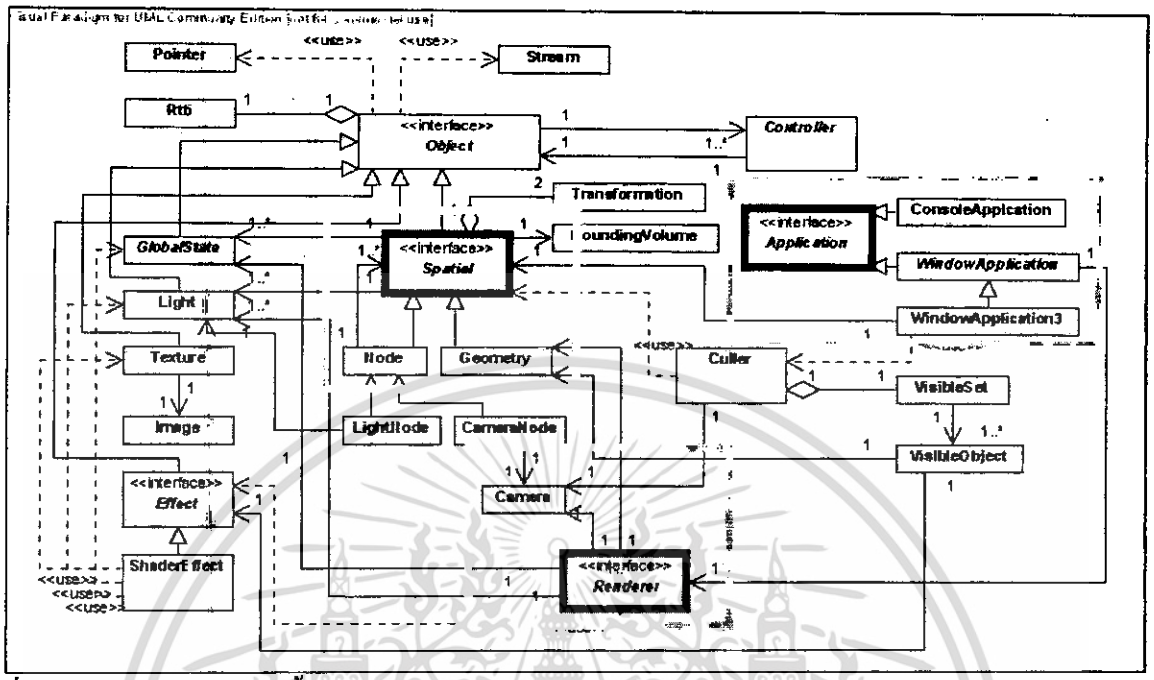
3.2 เครื่องมือที่ใช้ในการพัฒนา

(1) Visual Studio 2005 (โดยใช้ภาษา C/C++ ควบคู่กับ Win32API)

(2) DirectX SDK (สำหรับเรียกใช้งาน DirectX Graphics)

(3) Wgl (ติดตั้งมาพร้อมกับ Windows XP)

3.3 ภาพรวมของทั้งระบบโดยคร่าว



รูปที่ 3.2 แสดงภาพรวมของทั้งระบบโดยคร่าว โดยแบ่งออกเป็น 3 ระบบย่อย (ยกเว้น Foundation) คือ Spatial, Rendererของส่วน Graphics และ Renderers กับ Applciation ของส่วน Application

3.4 ส่วน Foundation

เป็นส่วนประกอบพื้นฐานหลักของระบบ (Core System) ซึ่งจะประกอบไปด้วยโครงสร้างข้อมูลที่ใช้ทั้งหมด ส่วนควบคุมจัดการกับเรื่องเวลาของระบบ การจัดการเกี่ยวกับหน่วยความจำและไฟล์ข้อมูล นอกจากนั้นยังมีส่วนสนับสนุนการทำงานทางด้านคณิตศาสตร์ (Mathematical Support) ของระบบด้วย

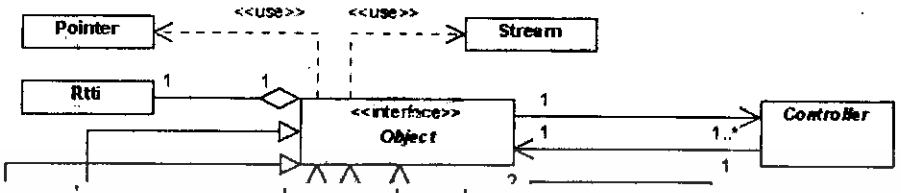
นอกจากโครงสร้างข้อมูลของคณิตศาสตร์พื้นฐาน เช่น Vector, Matrix, และ Quaternion ส่วนที่เหลือจะเป็นคลาสต่างๆที่ประกอบไปด้วยฟังก์ชั่นสมาชิกประเภท Static เพื่อให้เรียกใช้งานเท่านั้น

3.5 ส่วน Graphics

เป็นส่วนประกอบที่คอยจัดการและควบคุมระบบการแสดงผล (Graphics System) โดยออกแบบให้อยู่ในรูปแบบที่ไม่ขึ้นกับ Platform ใดๆ แบ่งออกเป็นส่วนต่างๆ ดังต่อไปนี้

3.5.1 Object System

เนื่องจาก Graphic Engine เป็น Library ที่ใหญ่และซับซ้อน จึงสมควรที่จะต้องมี กลุ่มของ Automatic Service ที่จัดการกับ Object ซึ่งมีจำนวนมาก โดยมีส่วนประกอบ ดังนี้



รูปที่ 3.3 แสดงส่วนหนึ่งจากรูปที่ 3.1 ซึ่งเป็นโครงสร้างโดยคร่าวของ Object

RTTI (Run-Time Type Information)

แม้ว่าโดยปกติแล้ว Polymorphism จะมี Abstraction ของการทำงาน ที่ทำให้การเรียกฟังก์ชั่นเป็นแบบ Polymorphic ซึ่งสามารถทำได้โดยไม่ต้องรู้ประเภท (Type) ที่แท้จริงของอ็อปเจ็กต์ที่เรียกก็ตาม, แต่ยังคงมี "กรณี ที่เราจำเป็นที่จะต้องรู้ Type ที่แน่นอน หรือพิจารณาว่าอ็อปเจ็กต์ นั้นๆ derive จากประเภท (Type) ใดๆ หรือไม่" (เช่น กรณีที่ต้องการจะทำ typecast อย่างปลอดภัยจากพอยต์เตอร์ของคลาส Base ไปเป็นพอยต์เตอร์ของคลาส Derived เรียกกระบวนการดังกล่าวว่า Dynamic Typecasting)

Single-Inheritance Class Trees

เป็นการรวมกลุ่มของ Directed Trees โดยที่ Tree Nodes เป็น คลาส และ Tree Arcs เป็นการสืบทอด (Directed Arc กำหนด ความสัมพันธ์แบบ is-a), Root Node ของ Tree จะแสดงถึงคลาส Common Base ของทุกๆ คลาส ใน Tree (แม้ว่าสามารถที่จะมีหลายๆ Tree ได้ แต่จะจำกัดให้มีเพียงแค่ Tree เดียว ในระบบ เท่านั้น) โดยที่คลาส Root จะมีส่วนช่วยงานพื้นฐานให้กับคลาส Derived ซึ่งในที่นี้จะให้ Root มีชื่อว่า Object

รูปแบบที่ง่ายที่สุด คือ ไม่มีการเก็บข้อมูลของคลาสและมีเพียงการเชื่อมไปยังคลาส Base, แต่อย่างไรก็ตาม เพื่อช่วยในการ Debug แล้ว จะเก็บ String ที่ระบุถึง ชื่อของ Class ไว้ด้วย

Names And Unique Identifiers

เพื่อช่วยในการค้าหา, โดยใช้ String Name และ Unique Integer-Valued Identifier

Name String ให้แต่ละ Object มีเดต้าสมาชิกประเภท Character String ส่วนฟังก์ชั่นสมาชิกสำหรับการค้นหาจะให้แต่ละคลาสที่สืบทอดไปทำการ override (ทั้งนี้ String ดังกล่าว ไม่จำเป็นจะต้อง Unique)

Unique Identifier จะใช้ 32-Bit Unsigned Integer โดยให้เป็น Static สำหรับคลาส Object ตัวหนึ่ง และเป็น Non-Static สำหรับ Object จริงๆ แต่ละ Object (ทั้งนี้จะไม่มีการนำ Identifier เก่ามาใช้ใหม่ ในกรณีที่อ็อปเจ็กต์ใดๆ ถูกลบทิ้งไป เพื่อความง่ายในการ implement)

Sharing and Smart Pointers

เพื่อที่ลดจำนวน Memory Use ที่เกิดขึ้น จึงใช้วิธีการ share Object โดยไม่ให้เกิดการ Lose (Object Leaking) หรือ การทำลายในขณะที่ยังมีส่วนอื่นใช้อยู่ (Premature Destruction) โดยที่ Automated System ดังกล่าว จะนำไปใช้เพื่อช่วยในการจัดการอ็อปเจ็กต์ซึ่งจะอยู่ในรูปแบบของ Reference Counter ที่จะนับจำนวน Pointer มายัง Object

จากการทำเช่นนี้ เป็นการบังคับให้ทุกอ็อปเจ็กต์ต้องถูกสร้างแบบ Dynamical Allocation

ทั้งนี้ จะต้องทำการ define คลาส Pointer ขึ้นมา (เป็น Template) เพื่อที่จะสามารถจัดการกับ Reference Counter ในคลาส Object ได้อย่างถูกต้องตามที่ต้องการ

Streaming

เพื่อที่จะ map ระหว่าง 2 Media (แม้โดยมากมักจะเป็น Disk Storage และ Memory แต่ก็สามารถประยุกต์ใช้เป็นระหว่าง Memory Block ได้เช่นกัน)

Scene Graph ถูกพิจารณาเป็น Directed Graph ของ Object โดยที่มี Node เป็น Object และ Arc เป็น Pointer ระหว่าง Object ซึ่งข้อมูลดังกล่าวจำเป็นจะต้องถูกเก็บลงไปใน Disk และถูกสร้างขึ้นมาใหม่ได้ (หมายความว่าทั้ง Nodes และ Arcs จะต้องถูกเก็บ และ Node ควรจะต้องถูกเก็บเพียงแค่ครั้งเดียวเท่านั้น) สรุปเป็นขั้นตอนได้ดังนี้

- สร้าง List ของ Unique Object ใน Graph ขึ้นมา
- เก็บ List ดังกล่าว ลงไปใน Disk
- ในกระบวนการที่เก็บลงไปใน Disk จะเก็บ Connection ระหว่าง Object ด้วย

ซึ่งคลาส Stream จะจัดการกับกระบวนการในการรวบรวมข้อมูลตลอดถึงการเก็บบันทึก

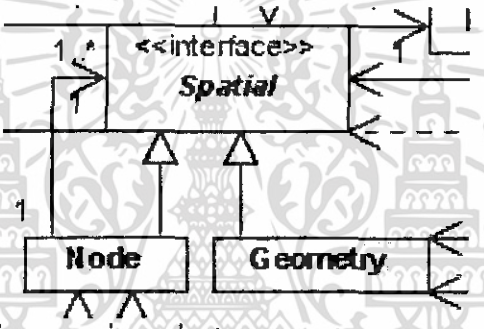
จากที่ได้กล่าวไว้แล้วว่า Stream Object นั้น มีไว้เพื่อจัดการกับ List ของอ็อปเจ็กต์ Top-Level โดยที่อ็อปเจ็กต์จะถูกใส่เข้าไปใน List โดย Insert เอาออกโดย Remove หรือ RemoveAll, GetObjectAt จะเป็นการ return อ็อปเจ็กต์ตัวที่ i ใน List Load/Save มีไว้สำหรับทั้งเพื่อ map ระหว่าง Disk กับ Memory และ ระหว่าง Memory Block ทั้งนี้ ในส่วนของคลาส Object เอง ก็ต้องมีส่วนของโครงสร้างเพื่อช่วยในการทำงานของ Stream เช่นเดียวกัน

Cloning

เพื่อทำ Deep Copy (คัดลอกทั้งอ็อปเจ็กต์) โดยพึ่งการทำงานของ Stream ที่ทำระหว่าง Memory Block (เริ่มจากการคัดลอกจาก Object ตั้งต้นไปยัง Memory Block จากนั้นจึงให้ Memory Block ทำการคัดลอกกลับไปยัง Object ใหม่), เนื่องจาก Object ทั้ง 2 ต่างต้องมีชื่อเป็นของตัวเอง ดังนั้นจึงจะมี Character พิเศษที่จะต่อท้าย เป็นข้อมูลสมาชิกประเภท Static ของคลาส Object

3.5.2 Scene Graph

คลาสในระบบการจัดการของ Scene Graph ที่สนับสนุนการทำงานของ Spatial Tree (หรือ Hierarchy) ของ Object คือ คลาส Spatial, Node และ Geometry



รูปที่ 3.4 แสดงส่วนหนึ่งจากรูปที่ 3.1 ซึ่งเป็นโครงสร้างโดยคร่าวของ Spatial และระบบ Scene Graph

แนวคิดของการออกแบบ Spatial Hierarchy

ประเด็นหลักในการออกแบบคลาส Spatial คือ เพื่อเป็น "ตัวแทนของ Coordinate System ใน Space" ทำให้ข้อมูลของสมาชิกในคลาสของมันจะต้องรวม Local และ World Transformation และ World Bounding Volume เข้าไป ทั้งนี้จะให้คลาสนี้เป็นเพียงคลาสแบบ Abstract Base เท่านั้น

จากความคิดที่ต้องการจะแยก คุณสมบัติของ "การจัดกลุ่ม" และ "การแสดงถึงข้อมูลที่เกี่ยวกับ Geometric" ออกจากกัน ซึ่งจะช่วยทำให้เกิดความเข้าใจในพฤติกรรมของของ Object ให้ง่ายขึ้นแล้ว จึงได้แยกเป็นคลาส Geometry และ Node ขึ้น โดยที่แม้ว่าจะสืบทอดจากคลาส Spatial มาเหมือนกัน แต่มีจุดประสงค์ในการใช้งานที่แตกต่างกัน คือ คลาส Geometry จะเป็นส่วนที่เกี่ยวข้องกับ Geometric Objects, Visual Appearances, Physical Properties ในขณะที่คลาส Node จะเป็นส่วนที่เกี่ยวข้องกับการจัดกลุ่มและ

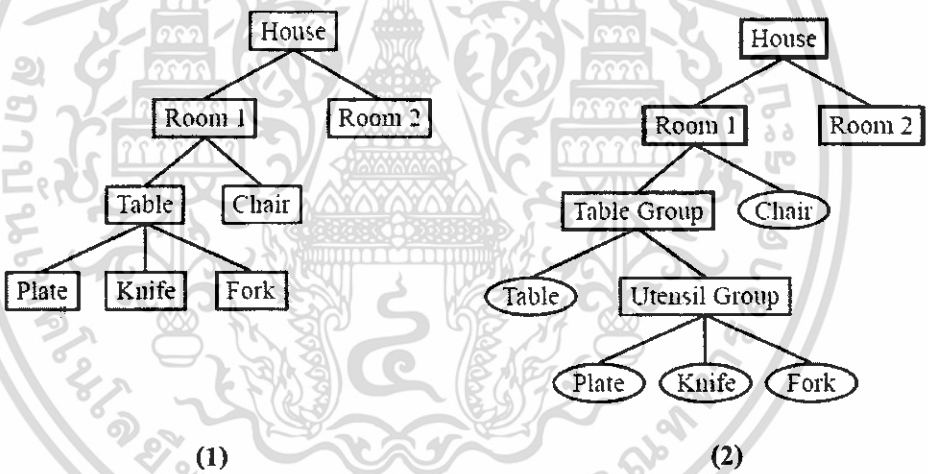
จัดการ Sub Tree ที่เชื่อมโยงอยู่กัน Node ทั้งนี้ จากการที่แยกออกเป็น 2 คลาส ทำให้เกิดผลต่อการออกแบบหลักๆ คือ

คลาส Geometry จะเก็บ Data เฉพาะของ Parent Link (จากส่วนของ Spatial) โดยให้มันเป็นได้เพียงแค่ Leaf ใน Hierarchy เท่านั้น

คลาส Node จะเก็บ Data ของ Child Link และ Parent Link (จากส่วนของ Spatial) โดยจะแบ่งย่อยได้ตามกรณี ดังนี้

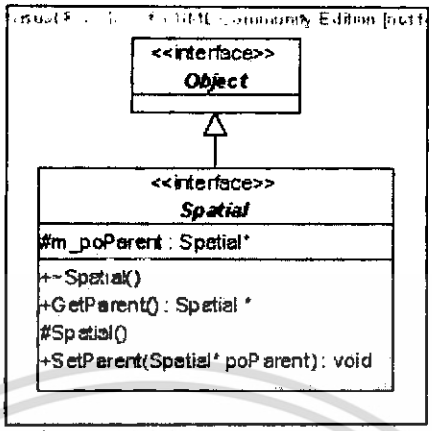
กรณีที่มีหน้าที่จัดกลุ่มของ Geometry Object เท่านั้น ส่วน Transformation ของ Node จะเป็น Identity Transformation

กรณีที่มีหน้าที่ทั้งจัดกลุ่มของ Geometry Object และสัมพันธ์กับ Geometry Object จริง ส่วน Transformation ของ Node จะเป็น Transformation ของ Geometry Object ในขณะที่ตัว Geometry Object นั้นจะเป็น Identity Transformation (ทั้งนี้เพื่อรักษา Transformation Structure ของ Hierarchy)



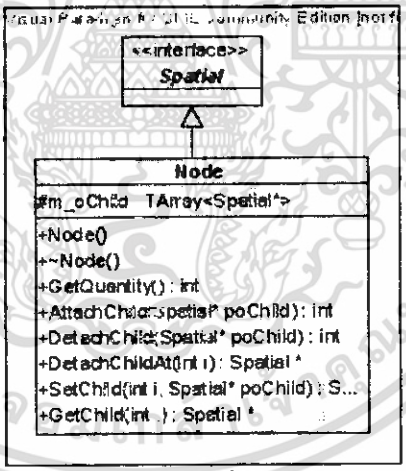
รูปที่ 3.5 (1) แสดง Spatial Hierarchy โดยทั่วไป
(2) แสดง Spatial Hierarchy เมื่อมีการนำแนวคิดที่จะแยก Node และ Geometry มาใช้

ส่วนหนึ่งของ Interface ของคลาส ที่สัมพันธ์กับ Scene Hierarchy ดังนี้



รูปที่ 3.6 แสดง ส่วนหนึ่งของ Interface ของคลาส Spatial ที่สัมพันธ์กับ Scene Hierarchy

ส่วนหนึ่งของ Node Class ที่สัมพันธ์กับการเชื่อมต่อกันของ Scene Hierarchy มีดังนี้



รูปที่ 3.7 แสดง ส่วนหนึ่งของ Node Class ที่สัมพันธ์กับการเชื่อมต่อกันของ Scene Hierarchy

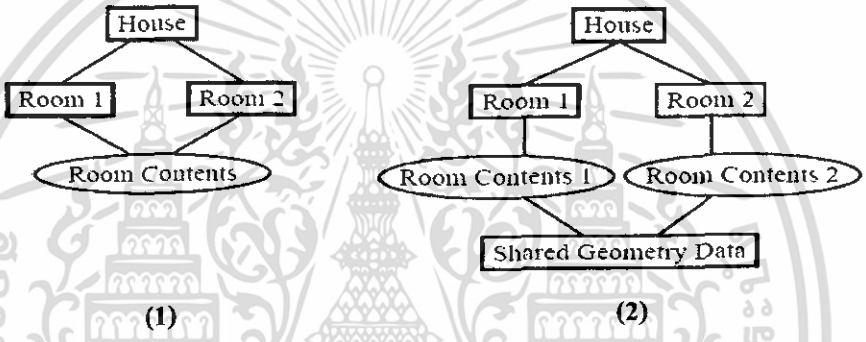
โดยที่ m_poChild เป็น Array ของ Spatial Smart Pointer และ m_iUsed ระบุว่า มีจำนวน Array Slot ที่เป็น Non-null Pointer อยู่จำนวนเท่าไหร่

Instancing

Instancing ในที่นี้ หมายถึง การทำให้เกิดการใช้งานร่วมกันของ Object ขึ้น

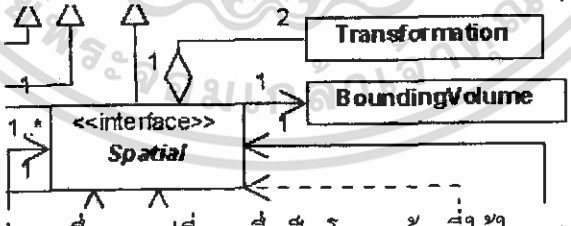
ซึ่งใน Spatial Hierarchy นี้ Object จากคลาส Node จะถูกออกแบบไม่ให้มีการ instance ได้ และจะทำการบังคับให้ Object จากคลาส Spatial มีได้เพียง Parent Link เดียวเท่านั้น

อย่างไรก็ตาม Instancing ยังคงมีความจำเป็นต่อ Engine เป็นอย่างมากสำหรับ Object ของคลาส Geometry เพื่อช่วยในการลดจำนวนการใช้งานของหน่วยความจำ ดังนั้น หากต้องการจะให้เกิด Instancing ขึ้น อาจจะแบ่งออกเป็นแต่ละ Object ที่สืบทอดต่อจากคลาส Geometry โดยจะเก็บเพียงแค่ Content (สาระสำคัญ) ที่แตกต่างกันไปของแต่ละ Object ในขณะที่ข้อมูลเกี่ยวกับ Geometric (เช่น Vertices, Texture Coordinates, Texture Image, ฯลฯ) จะใช้งานร่วมกัน



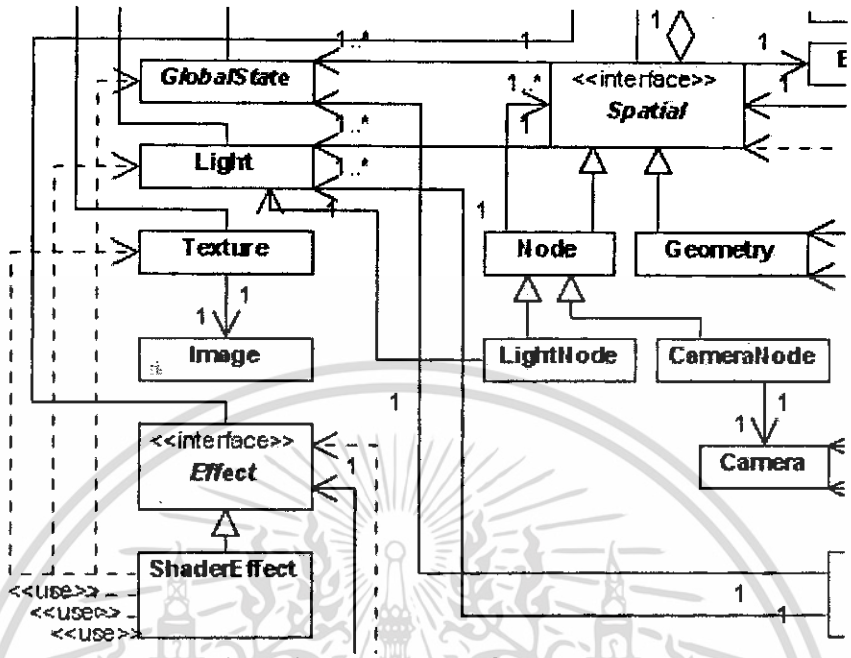
รูปที่ 3.8 (1) แสดง Instancing โดยทั่วไป
(2) แสดง Instancing เมื่อมีการนำแนวคิดที่จะ share Geometry Data มาใช้

ระบบย่อย Geometric State Update



รูปที่ 3.9 แสดงส่วนหนึ่งจากรูปที่ 3.1 ซึ่งเป็นโครงสร้างที่ใช้ในระบบย่อย Geometric State Update

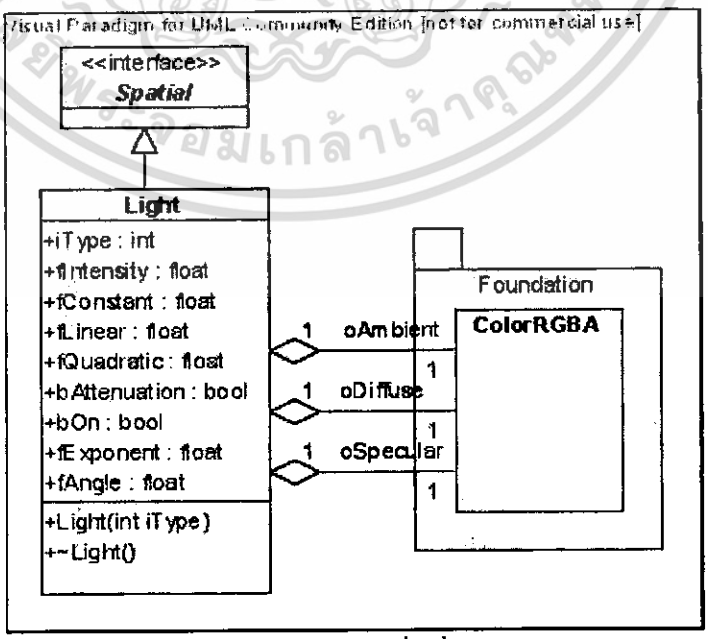
ระบบย่อย Render State Update



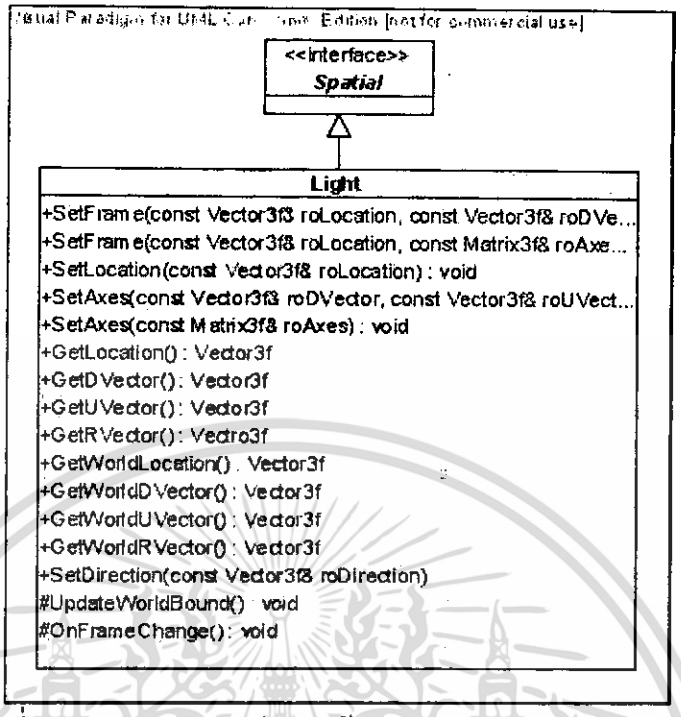
รูปที่ 3.10 แสดงส่วนหนึ่งจากรูปที่ 3.1 ซึ่งเป็นโครงสร้างที่ใช้ในระบบย่อย Render State Update

Light

คลาส Light เอง ก็ได้ถูกออกแบบให้สืบทอดจากคลาส Spatial และจาก Physical Attribute และประเภทต่างๆ ของ Light ทำให้ได้โครงสร้างเบื้องต้นของคลาส Light ดังรูปต่อไปนี้



รูปที่ 3.11 แสดงส่วนของโครงสร้างของคลาส Light (1) ซึ่งเกี่ยวข้องกับ Physical Attribute



รูปที่ 3.12 แสดงส่วนของโครงสร้างของคลาส Light (2) ซึ่งเกี่ยวข้องกับ Light Coordinate System

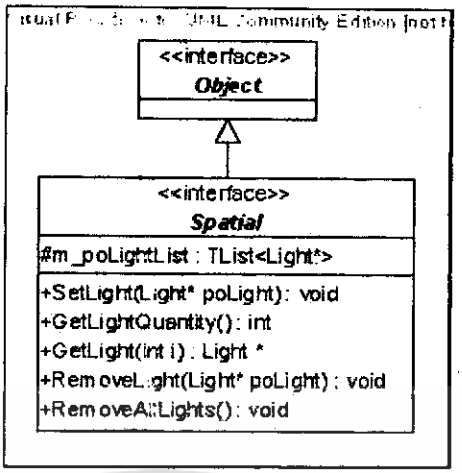
ส่วนมากจะเป็นฟังก์ชั่นที่เกี่ยวกับการให้และรับค่าที่เกี่ยวข้องกับ Coordinate System คือ Origin (Eye Point), Direction Vector, Up Vector, Right Vector ดังที่ได้กล่าวมาก่อนหน้าแล้ว โดยที่ในส่วนของการให้ค่า Coordinate จะมีให้เลือกอยู่ 2 แบบ คือใช้ 3 Vector หรือ 3*3 Matrix

สำหรับ Directional Light จะมีฟังก์ชั่น SetDirection ที่ต้องให้ค่า (เป็น Unit-length Vector)

นอกจากนั้น จะมีฟังก์ชั่น OnFrameChange ของคลาส Light จะถูกเรียกใช้งานเมื่อใดก็ตามที่มีการให้ค่า Coordinate System ขึ้น โดยฟังก์ชั่นจะทำการใช้งาน UpdateGS อีกทีหนึ่ง (จึงไม่ต้องทำการเรียกโดยตรง)

<u>ส่วนสนับสนุนการทำงานของ Light ในคลาส Spatial และ Geometry</u>

คลาส Spatial จะถูกออกแบบให้เก็บ List ของอ็อปเจ็กต์ของคลาส Light ไว้ โดยหากเมื่อไปเป็นของอ็อปเจ็กต์ของคลาส Node จะออกแบบให้ Light นั้นให้แสงไปยังทุกๆ Leaf (Geometry) ใน Subtree ที่มี Node นั้นๆ เป็น Root ทั้งนี้ จะมีฟังก์ชั่นในควบคุมการจัดการ ดังต่อไปนี้



รูปที่ 3.13 แสดงส่วนของโครงสร้างของคลาส Spatial
ที่สนับสนุนการทำงานของคลาส Light

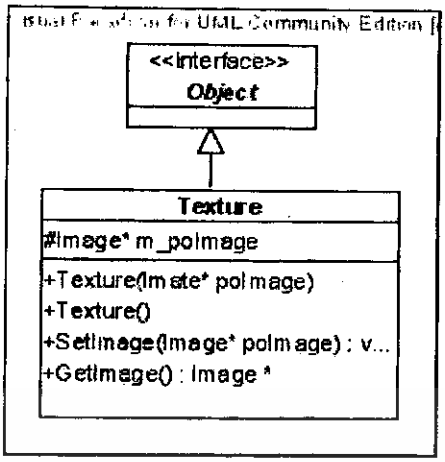
ในส่วนของคลาส Geometry นั้นจะมีการเก็บ Array ของ Light เพิ่มเติม ซึ่งเป็นพอยต์เตอร์ที่ชี้ไปยังอ็อบเจ็กต์ของ Light ที่เจอตลอดการ Traversal จาก Root Node มาถึงตัวอ็อปเจ็กต์ของ Geometry เอง (ใน Hierarchy) ทั้งนี้ เนื่องจากกลุ่มของ Light ดังกล่าว จะถูก Renderer นำไปใช้ในการให้แสงต่อ Geometric Object นั้นๆ

Texture

คลาส Texture ได้ถูกออกแบบมาเพื่อเก็บข้อมูลทั้งหมดที่จำเป็นจะต้องใช้เพื่อติดตั้ง Texture Unit บน Graphic Card โดยอย่างน้อยคลาสควรจะต้องเก็บ Texture Image อย่างไรก็ตาม Texture Coordinate ที่ถูกให้ค่ากับ Vertices ของ Geometry Object จะไม่ได้ถูกเก็บไว้ในคลาสนี้

ทั้งนี้ จะมีคลาส Effect ที่ทำการเก็บ Texture และ Texture Coordinate ที่สัมพันธ์กับมัน ด้านคลาส Geometry นั้น จะรับผิดชอบเพียงแค่ Vertex Position, Normals และ Indices เท่านั้น

ดังนั้น ส่วนของโครงสร้างของคลาส Textureที่เกี่ยวข้องกับ Image จะมีดังต่อไปนี้



รูปที่ 3.14 แสดงส่วนของโครงสร้างของคลาส Texture ที่เกี่ยวข้องกับ Image

ทั้งนี้ Graphic System ได้จัดหา Control ต่างๆ มากมายในการควบคุมว่า Image จะถูกวาดลงบนวัตถุอย่างไร ดังต่อไปนี้

FilterType

ในการที่ Texture Image จะถูก map ลงไปบน Triangle ได้นั้น จะต้อง assign ค่า Texture Coordinate ให้กับ Vertices ของ Triangle ก่อน เมื่อ Map ไปยัง Screen Space แล้ว Pixel ที่อยู่ภายใน Triangle จะต้องถูกให้ค่าสี ซึ่งกระบวนการดังกล่าวจะเกิดจากการทำ Linearly Interpolating ค่าของ Texture Coordinate ที่ Vertices เพื่อสร้าง Texture Coordinate ณ ตำแหน่ง Pixel นั้นๆ ขึ้นมา

ซึ่งเป็นไปได้ว่า Texture Coordinate ที่ถูกสร้างขึ้นมานั้น ไม่สัมพันธ์กับ Image Pixel ดังนั้นจึงจะต้องระบุถึงวิธีการสร้างสีขึ้นจากคู่ของตัวเลขกับ Image ดังกล่าว ซึ่งมีอยู่ 2 ทางเลือก คือ

Nearest (Nearest-neighbor Interpolation) จะทำการปัดค่าโดยพิจารณาเลขทศนิยมปกติ

Linear (Bilinear Interpolation) โดยที่จะให้ Real-valued Index (i', j') ตกอยู่บนสี่เหลี่ยมจัตุรัสที่มีมุมเป็น Integer-valued Index (ให้ $i_0 = \lfloor i' \rfloor$ และ $j_0 = \lfloor j' \rfloor$ มุมทั้ง 4 จะเป็น $(i_0, j_0), (i_0+1, j_0), (i_0, j_0+1), (i_0+1, j_0+1)$) และ สมการ Bilinear Interpolation จะได้เป็นดังต่อไปนี้

$$C' = (1-\Delta_i)(1-\Delta_j)C_{i_0,j_0} + (1-\Delta_i)\Delta_j C_{i_0,j_0+1}$$
$$+\Delta_i(1-\Delta_j)C_{i_0+1,j_0} + \Delta_i\Delta_j C_{i_0+1,j_0+1}$$

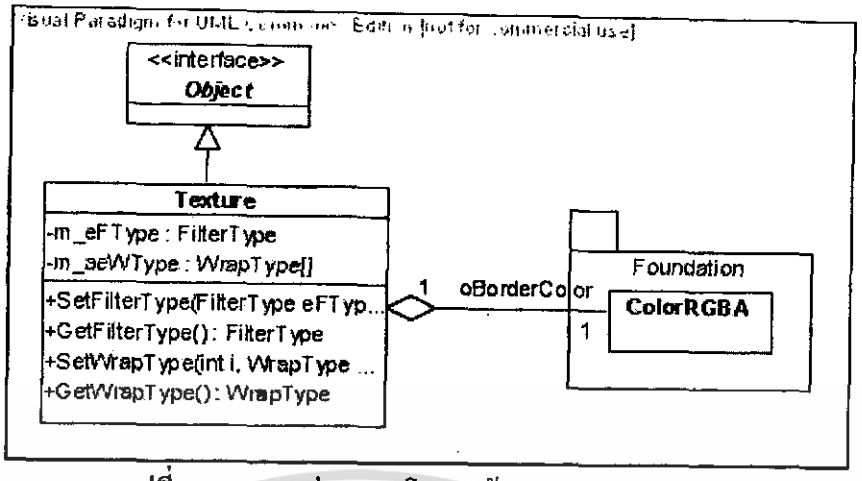
เมื่อ C' คือ สีที่ถูกสร้างขึ้น $C_{i,j}$ เป็นสีของ Image และ $\Delta_i = i' - i_0$ กับ $\Delta_j = j' - j_0$

แม้ว่าจะมีการทำ Bilinear Filtering แล้ว แต่ Texturing ยังสามารถเกิด Artifact ขึ้นได้

ทั้งนี้ ส่วนของโครงสร้างของคลาส Texture ที่ให้การสนับสนุนการทำงาน มีทางเลือกดังนี้

FilterType	คำอธิบาย
NEAREST	ใช้แค่ Texture Image เดิม ไม่ทำการสร้าง Pyramid โดยใช้กระบวนการ Nearest-neighbor Interpolation สร้างสี
LINEAR	ใช้แค่ Texture Image เดิม ไม่ทำการสร้าง Pyramid โดยใช้กระบวนการ Bilinear Interpolation สร้างสี
NEAREST_NEAREST	สร้าง Mipmap แล้วเลือก Mipmap Image และให้สี Pixel ด้วยวิธี Nearest-neighbor
NEAREST_LINEAR	สร้าง Mipmap แล้วเลือก 2 Mipmap Image ที่ปิดล้อม Pixel ไว้ ในแต่ละ Image เลือก Texel ที่ใกล้กับ Pixel ที่สุด แล้วให้สี Pixel ด้วยการทำ Linearly Interpolation จาก Texel ทั้ง 2
LINEAR_NEAREST	สร้าง Mipmap แล้วเลือก Mipmap Image ด้วยวิธี Nearest-neighbor แต่ให้สี Pixel ด้วยวิธี Linearly Interpolation
LINEAR_LINEAR	สร้าง Mipmap แล้วเลือก 2 Mipmap Image ที่ปิดล้อม Pixel ไว้ ในแต่ละ Image ให้ทำการ Bilinear Interpolation กับ Texel แล้วให้สี Pixel ด้วยการทำ Linearly Interpolation จาก Texel ทั้ง 2 (เรียกว่า Trilinear Interpolation)

ตารางที่ 3.1 แสดงทางเลือกของ Mipmap Mode ของคลาส Texture

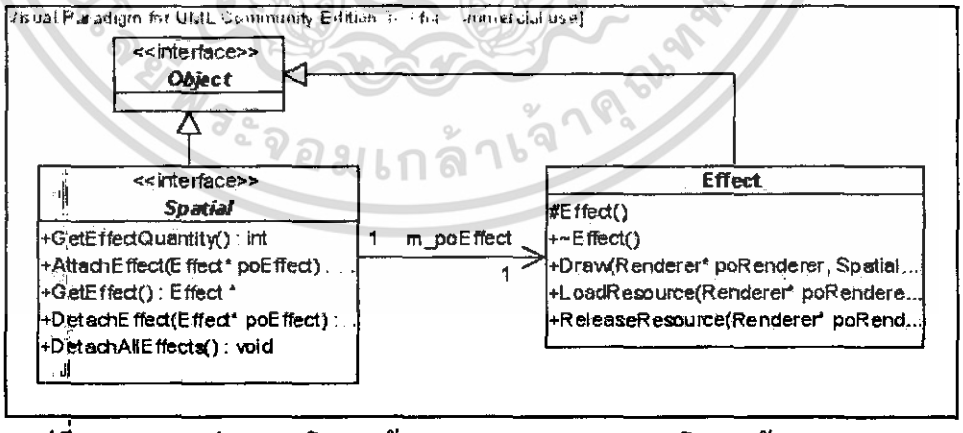


รูปที่ 3.15 แสดงส่วนของโครงสร้างของคลาส Texture ที่ช่วยสนับสนุนการทำงานของ FilterType

Effect

คลาส Effect ได้ถูกออกแบบให้สืบทอดจากคลาส Object โดยให้ Local Effect เฉพาะให้กับตัวอ็อปเจ็กต์ของ Geometry ซึ่งเป็น Leaf ใน Spatial Hierarchy นอกจากนั้น อ็อปเจ็กต์ของคลาส Effect ยังสามารถนำไปประยุกต์ใช้ให้เป็น Global Effect ได้ (เมื่อใช้กับคลาส Node)

ทั้งนี้ คลาส Effect จะเป็นเพียงคลาส Abstract Base ที่มี Virtual Function เดียว ทำงานควบคู่กับคลาส Spatial ซึ่งจะมีส่วนที่สนับสนุนการทำงานของคลาส Effect ด้วย ดังรูปต่อไปนี้

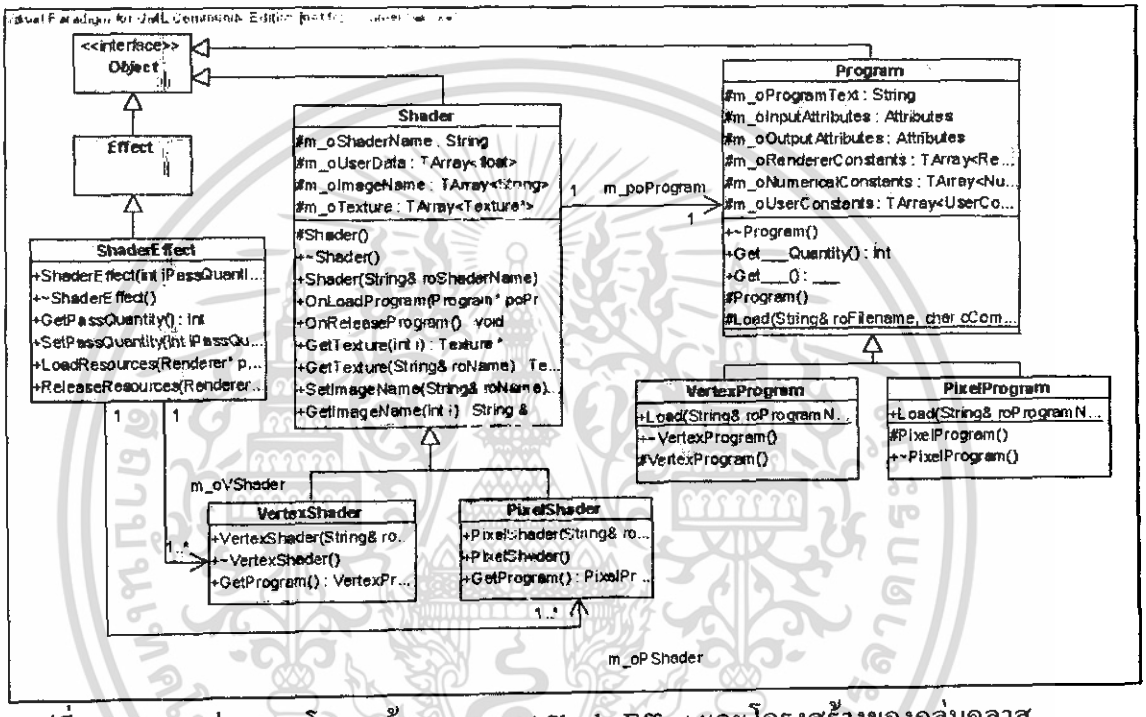


รูปที่ 3.16 แสดงส่วนของโครงสร้างของคลาส Spatial และโครงสร้างของคลาส Effect รวมไปถึงความสัมพันธ์กันของทั้ง 2 คลาส

จากรูป คลาส Effect จะมีฟังก์ชั่นสำหรับการวาดคือ Draw ซึ่งโดยปกติจะทำกา รวมเพื่อให้ค่ากับวัตถุที่มีผลกระทบจาก Effect และฟังก์ชั่น LoadResource กับ

ReleaseResource จะมีไว้เพื่อให้คลาสที่สืบทอดต่อสามารถนำไปใช้จัดการกับทรัพยากรต่างๆ ที่ Effect ต้องใช้ ส่วนทางด้านคลาส Spatial นั้นก็จะเก็บพอยต์เตอร์ไปยังอ็อปเจ็กต์ของคลาส Effect และมีฟังก์ชั่นสำหรับทำการให้ค่ารับค่าได้

ทั้งนี้ ในระบบยังมีคลาสที่สืบทอดต่อจากคลาส Effect ที่มีความสำคัญต่อระบบการวาดภาพอย่างมาก คือ คลาส ShaderEffect ซึ่งสนับสนุนการทำงานของ Vertex Shader และ Pixel Shader โดยร่วมทำงานและมีความสัมพันธ์กับคลาสอื่นๆ ดังรูปต่อไปนี้

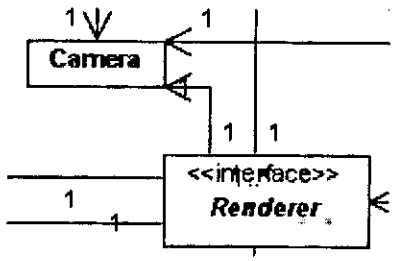


รูปที่ 3.17 แสดงส่วนของโครงสร้างของคลาส ShaderEffect และโครงสร้างของกลุ่มคลาสต่างๆ ซึ่งทำงานร่วมกัน คือ กลุ่มคลาส Shader กับกลุ่มคลาส Program

จากรูป คลาส ShaderEffect จะเป็นตัวควบคุมจัดการ Vertex และ Pixel Shader โดยที่จะเก็บกลุ่มของคลาสดังกล่าวไว้ พร้อมทั้งจัดหาฟังก์ชั่นสำหรับการใช้งานไว้ ทั้งนี้ในกลุ่มของคลาสพวก Shader จะเป็นส่วนที่จัดสรรทรัพยากรที่จะต้องใช้และทำการติดต่อกับตัวคลาส Program ที่เป็นตัวแทนของ Shader Program ที่จะเก็บค่า Attibute และ Constant ต่างๆ อีกทีหนึ่ง

ดังที่ได้กล่าวไปแล้วว่า คลาส ShaderEffect ยังมีส่วนของโครงสร้างที่สนับสนุนการทำงานของทั้ง Vertex Shader และ Pixel Shader ทั้งนี้ จะเกี่ยวข้องกับ การให้ค่ารับค่าอ็อปเจ็กต์จากกลุ่มคลาส Shader, Program, Global State, Texture, Image, รวมไปถึงค่า Constant ต่างๆ

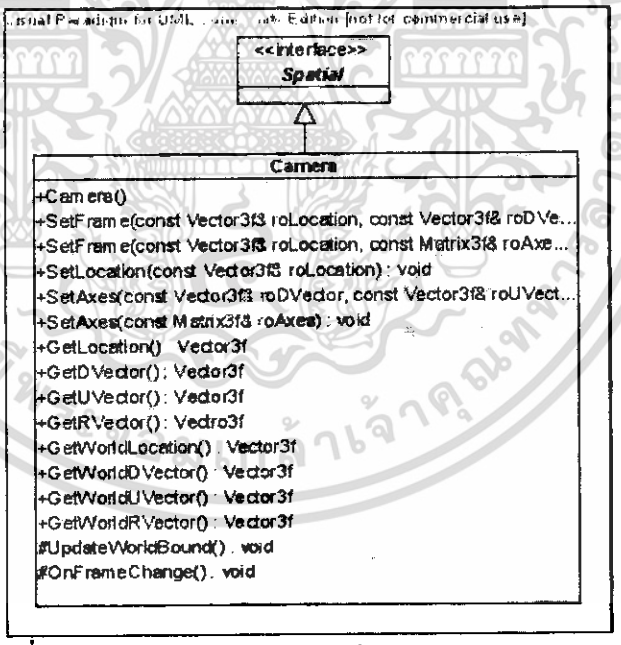
3.5.3 Rendering



รูปที่ 3.18 แสดงส่วนหนึ่งจากรูปที่ 3.1 ซึ่งเป็นโครงสร้างโดยคร่าวของ Renderer และระบบ Rendering

Camera

คลาส Camera ได้ถูกออกแบบให้สืบทอดจากคลาส Spatial เนื่องจากต้องการที่จะใช้คุณสมบัติของการมีตำแหน่งและการวางตัว ในลักษณะเดียวกันกับคลาส Light โดยจะมีโครงสร้างดังรูปต่อไปนี้



รูปที่ 3.19 แสดงส่วนของโครงสร้างของคลาส Camera (1) ซึ่งเกี่ยวข้องกับ Camera Coordinate System

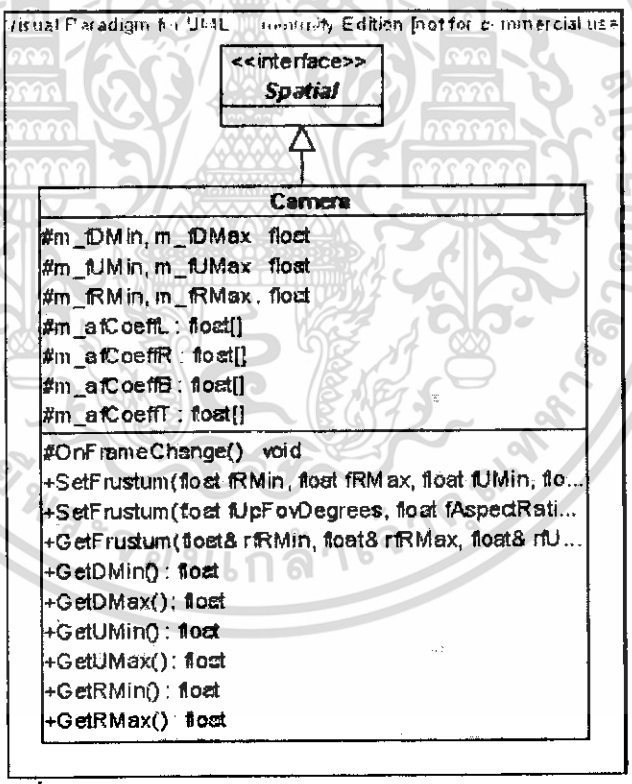
ส่วนมากจะเป็นฟังก์ชั่นที่เกี่ยวกับการให้และรับค่าที่เกี่ยวข้องกับ Coordinate System คือ Origin (Eye Point), Direction Vector, Up Vector, Right Vector ดังที่ได้กล่าว

มาก่อนหน้าแล้ว โดยที่ในส่วนของการให้ค่า Coordinate จะมีให้เลือกอยู่ 2 แบบ คือใช้ 3 Vector หรือ 3*3 Matrix

นอกจากนั้น จะมีฟังก์ชั่น OnFrameChange ของคลาส Camera จะถูกเรียกใช้งานเมื่อใดก็ตามที่มีการให้ค่า Coordinate System ขึ้น โดยฟังก์ชั่นจะทำการใช้งาน UpdateGS อีกทีหนึ่ง (จึงไม่ต้องทำการเรียกโดยตรง) ข้อแตกต่างกับฟังก์ชั่นของคลาส Light คือ ของคลาส Camera จะทำการคำนวณ World Coordinate ของ Frustum Plane ที่จะต้องใช้ในการทำ Culling และจะบอก Renderer ที่เกี่ยวข้องกับมันว่า Camera Coordinate System ได้เปลี่ยนไปแล้ว

View Frustum Parameters

ดังที่ได้กล่าวไปก่อนหน้าแล้ว ค่าที่เกี่ยวข้องกับส่วนของ View Frustum จะมี d_{min} (Near), d_{max} (Far), u_{min} (Bottom), u_{max} (Top), r_{min} (Left) และ r_{max} (Right)



รูปที่ 3.20 แสดงส่วนของโครงสร้างของคลาส Camera (2) ซึ่งเกี่ยวข้องกับ View Frustum

ในลักษณะคล้ายกับ Coordinate System คือ โดยมากจะเป็นฟังก์ชั่นที่เกี่ยวกับการให้และรับค่าที่เกี่ยวข้อง ทั้งนี้จะมีอยู่ 2 แบบ คือ ระบุโดยตรง หรือบอกเป็นค่า Field Of

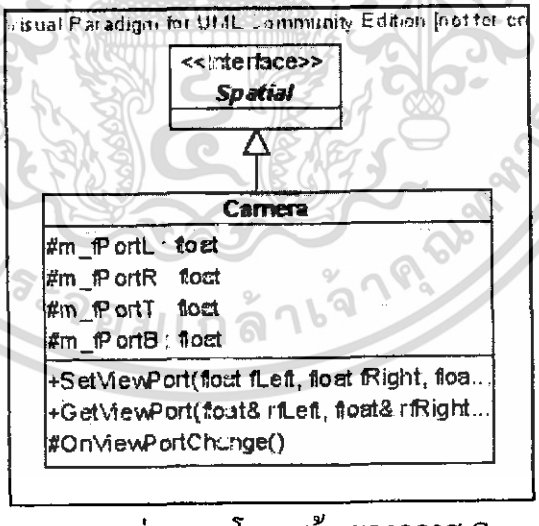
View ซึ่งเป็นค่ามุมระหว่าง Top Plane และ Bottom Plane และค่า Aspect Ration ซึ่งเป็นค่าความกว้างหารด้วยความสูง

[หมายเหตุ: ค่า Field Of View ในที่นี้ อยู่ในหน่วยดีกรี และมีค่าอยู่ในช่วง (0, 180)]

ทั้งนี้ ฟังก์ชั่น OnFrameChange ก็ยังคงมีความเกี่ยวข้องกับตรงส่วนนี้ด้วยโดยจะเป็น Callback เมื่อมีการเรียกใช้งาน SetFrustum ขึ้น ซึ่งตรงจุดนี้จะทำให้ Renderer สามารถที่จะทำการเปลี่ยนแปลงสถานะของมัน (หรือพูดให้ชัดว่าบอก Graphic API ถึง Frustum Parameter ใหม่) นอกจากนั้น Callback ยังทำการคำนวณค่าที่เกี่ยวข้องกับการทำ Culling คือ ค่า Coeff ของ Coordinate Axis Vector โดยจะเก็บไว้ในเดต้าของคลาส ได้แก่ m_afCoeffL, m_aCoeffR, m_afCoeffB, m_afCoeffT ซึ่งค่าดังกล่าวนี้จะถูกใช้เพื่อคำนวณ World Representation ของ Frustum Plane

Viewport Parameters

ส่วนของ Viewport Parameters ถูกใช้เพื่อแทนหน้าจอแสดงผลคอมพิวเตอร์ (Computer Screen) ใน Normalized Display Coordinates $(\bar{x}, \bar{y}) \in [0,1]^2$ โดยที่ ขอบซ้ายของหน้าจอ คือ $\bar{x} = 0$, ขอบขวา คือ $\bar{x} = 1$, ขอบล่าง คือ $\bar{y} = 0$, และขอบบน คือ $\bar{y} = 1$

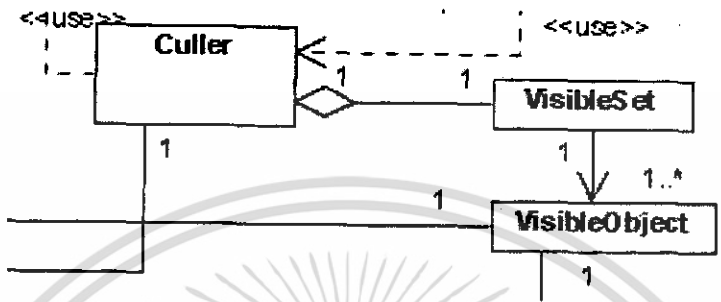


รูปที่ 3.21 แสดงส่วนของโครงสร้างของคลาส Camera (3) ซึ่งเกี่ยวข้องกับ Viewport

รูปแบบของฟังก์ชั่นให้และรับค่าจะมีอยู่รูปแบบโดยตรงแบบเดียว นอกจากนั้นแล้วจะมีฟังก์ชั่น OnVeiwPortChange ซึ่งเป็น Callback ที่จะถูกเรียกเมื่อมีการเรียก SetViewPort (หรือมีการให้ค่า) โดยที่มันจะทำการบอก Renderer ถึงการเปลี่ยนแปลงของ

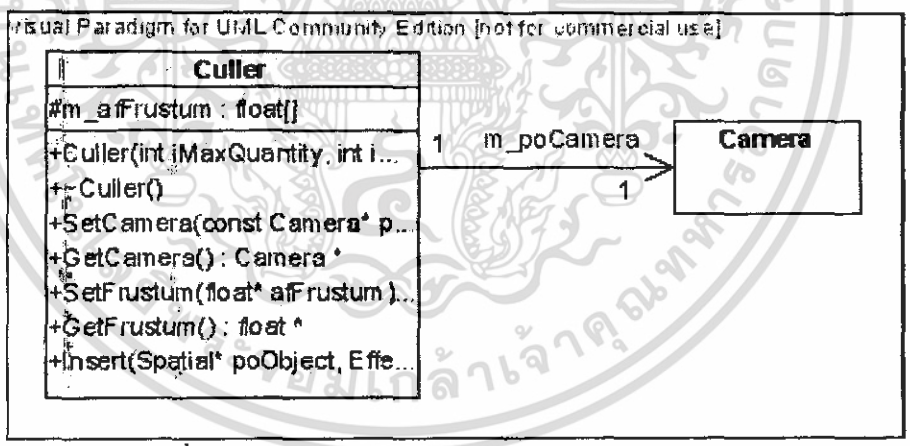
Viewport เพื่อให้ Renderer จะได้ทำการเปลี่ยนแปลงสถานะของมัน (เช่นเดียวกับรูปแบบของ OnViewFrustumChange คือ พูดให้ชัดว่าบอก Graphic API ถึง Viewport Parameter ใหม่)

Culler



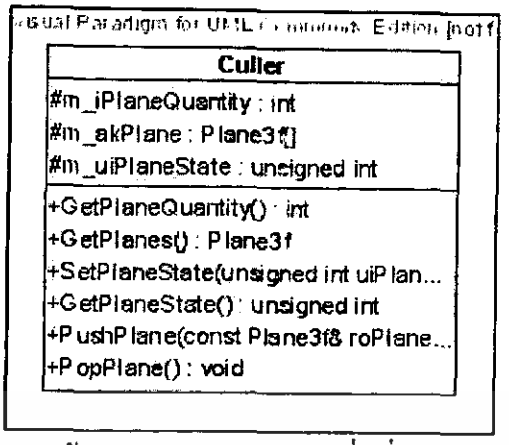
รูปที่ 3.22 แสดงส่วนหนึ่งจากรูปที่ 3.1 ซึ่งเป็นโครงสร้างโดยคร่าวของ Culler

ระบบ Object Culling ของระบบจะถูกเรียกใช้งานแยกจากการวาดรูป ทั้งนี้จะมีคลาสที่ทำงานร่วมกันอยู่หลายคลาส โดยจะมีคลาสที่ควบคุมจัดการหลัก คือ Culler ซึ่งจะมีโครงสร้าง ดังรูปต่อไปนี้



รูปที่ 3.23 แสดงส่วนของโครงสร้างของคลาส Culler (1)
ซึ่งเกี่ยวข้องกับข้อมูลเบื้องต้นและการเข้าถึงข้อมูล

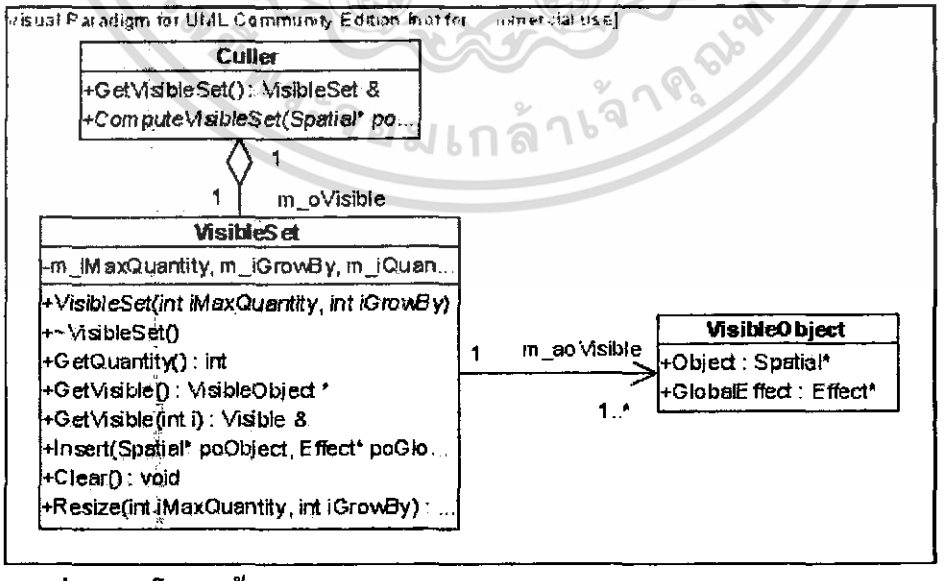
จากรูปจะเห็นได้ว่า คลาส Culler จำเป็นที่จะต้องมีการติดต่อกับตัวอ็อปเจ็กต์ของคลาส Camera เพื่อนำข้อมูลมาช่วยในการประมวลผล ทำให้มีส่วนของโครงสร้าง ดังต่อไปนี้



รูปที่ 3.24 แสดงส่วนของโครงสร้างของคลาส Culler (2) ซึ่งเกี่ยวข้องกับสถานะของ Culling Plane

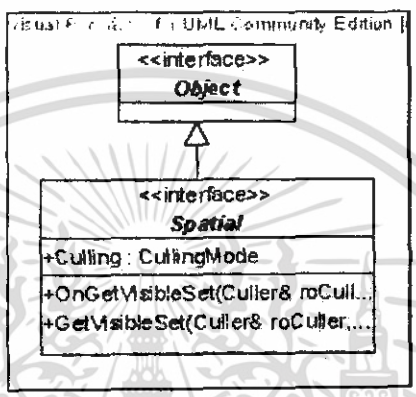
ทั้งนี้ มีจุดประสงค์เพื่อที่จะลดเวลาในการคำนวณเพื่อตรวจสอบ Bounding Volume กับ Frustum Plane โดยออกแบบให้อยู่ในรูปแบบของ Bit Flag ซึ่งหากมีค่าเป็น 1 คือ ให้ทำ แต่ 0 คือ ไม่ต้องทำ โดยที่จะมี m_uiPlaneState เก็บสถานะของแต่ละ Plane เอาไว้ในแต่ละบิต เริ่มจาก บิต 0 สำหรับ Left Plane, บิต 1 สำหรับ Right Plane, บิต 2 สำหรับ Bottom Plane, บิต 3 สำหรับ Top Plane, บิต 4 สำหรับ Near Plane และ บิต 6 สำหรับ Far Plane ในส่วนของ World Representation สำหรับ Plane นั้นจะถูกเก็บไว้ใน *m_aoPlane*

นอกจากคลาส Culler ยังมีคลาสอื่นๆ ที่นำมาใช้ร่วมกันในระบบอีก คือ คลาส VisibleObject กับ VisibleSet ซึ่งมีโครงสร้างและความสัมพันธ์ซึ่งกันและกัน ดังรูปต่อไปนี้



รูปที่ 3.25 แสดงส่วนของโครงสร้างของคลาส Culler (3) คลาส VisibleObject และคลาส VisibleSet รวมไปถึงความสัมพันธ์กันระหว่างคลาส

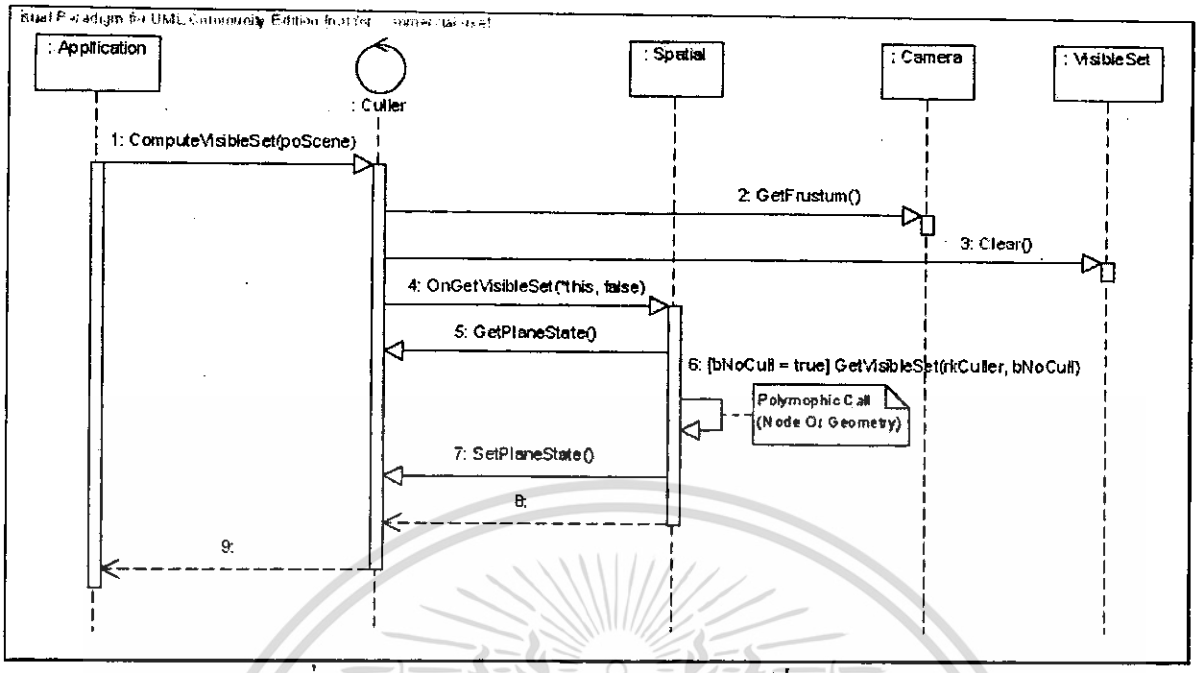
จากรูป คลาส VisibleObject จะเป็นโครงสร้างปกติทั่วไปที่เก็บพ็อยต์เตอร์ไปยังวัตถุและ GlobalEffect ที่อยู่ในมุมกล้อง ในขณะที่ VisibleSet นั้นจะเป็นเสมือน Container ของอ็อปเจ็กต์ที่เป็น VisibleObject ส่วนคลาสของ Culler ก็ถูกออกแบบให้จัดการควบคุมการทำ Culling ของฉากและเก็บวัตถุที่มีส่วนที่ปรากฏอยู่บนหน้าต่าง นอกจากนั้นทางคลาส Spatial เองก็จะต้องมีปฏิสัมพันธ์กันคลาส Culler โดยมีส่วนของโครงสร้างของคลาส ดังรูปต่อไปนี้



รูปที่ 3.26 แสดงส่วนของโครงสร้างของคลาส Spatial ที่สนับสนุนการทำงานของ Culler

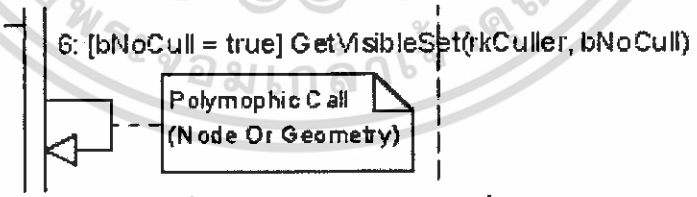
จากรูป ฟังก์ชั่น OnGetVisibleSet มีไว้เพื่อทำปฏิสัมพันธ์กับคลาส Culler (ดังจะแสดงให้เห็นในภายหลัง) ส่วนคลาส GetVisibleSet นั้นจะเป็นฟังก์ชั่นในแบบ Virtual เพื่อให้คลาสที่สืบทอดต่อ (คือ Node และ Geometry) ไป implement ต่อ นอกจากนั้นจะมีข้อมูล CullingMode อีกด้วย

จุดเริ่มต้นของ Culling นั้น จะผ่านการเรียกใช้งานฟังก์ชั่น ComputeVisibleSet เพื่อที่จะส่งไปให้กับระบบการวาดต่อไป โดยฟังก์ชั่นดังกล่าวจะมีการทำงานดังต่อไปนี้



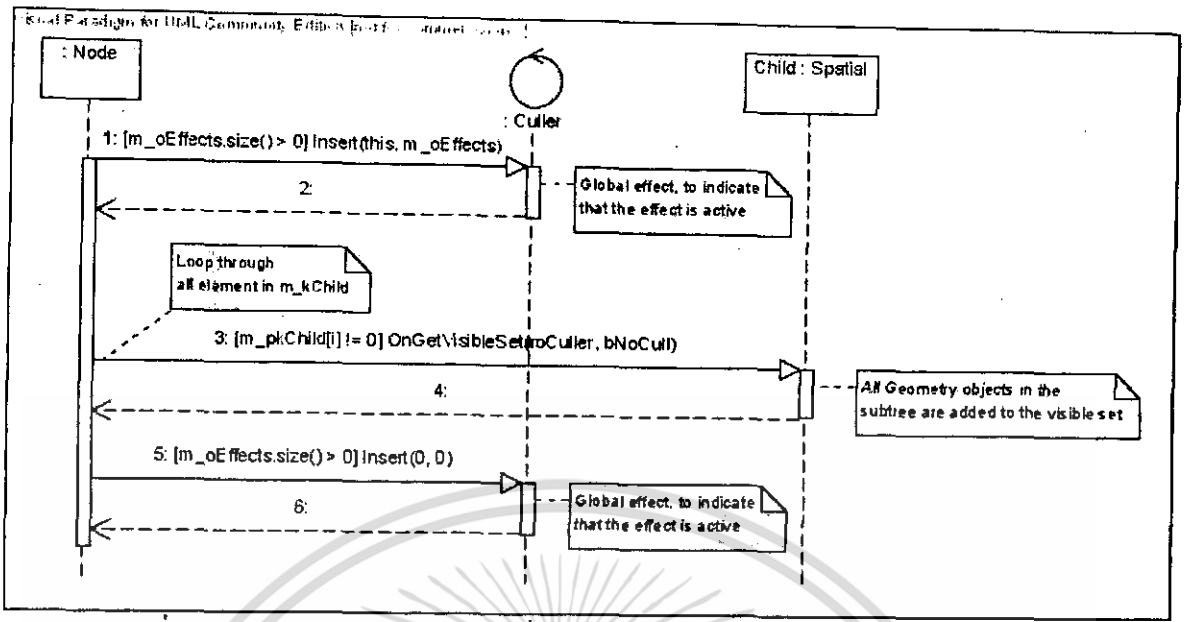
รูปที่ 3.27 แสดง Sequence Diagram ของฟังก์ชั่น ComputeVisibleSet ซึ่งเป็นจุดเริ่มต้นของ Object Culling

จะเห็นได้ว่าคลาส Culler เองจะมีฟังก์ชั่นที่ต้องใช้ระหว่างกระบวนการอยู่พอสมควร ทั้งนี้เพื่อที่จะใช้ในการเก็บและคืนค่าสถานะของ Camera ตลอด Scene ซึ่งก็คือฟังก์ชั่น GetPlaneState และ SetPlaneState ทั้งนี้หากตรวจสอบได้ผลว่า Object อยู่ในขอบเขตการแสดงผลแล้ว มันจะทำการเรียกฟังก์ชั่น GetVisibleSet ซึ่งเป็น Polymorphic Call ทำให้การทำงานที่เกิดขึ้นจะแตกต่างกันไปตามประเภท (Type) ของตัว Object นั้นจริงๆ ซึ่งแบ่งเป็น Node กับ Geometry ดังจะเห็นได้ตามรูปต่อไปนี้

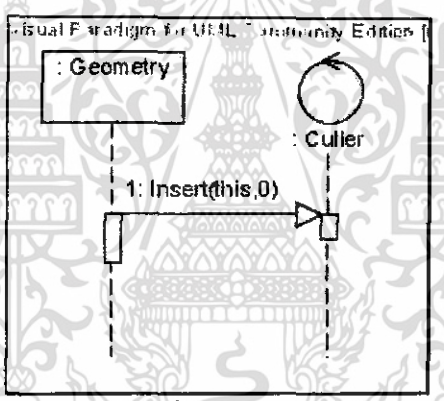


รูปที่ 3.28 แสดงส่วนหนึ่งของการทำงานของฟังก์ชั่น ComputeVisibleSet ของคลาส Spaital ซึ่งเป็นการเรียกใช้งานฟังก์ชั่น Draw ที่เป็น Polymorphic Call

ทั้งนี้การทำงานจริงที่แตกต่างกันของคลาส Node และคลาส Geometry เป็นดังรูปต่อไปนี้



รูปที่ 3.29 แสดงการทำงานของฟังก์ชั่น ComputeVisibleSet ของคลาส Node



รูปที่ 3.30 แสดงการทำงานของฟังก์ชั่น ComputeVisibleSet ของคลาส Geometry

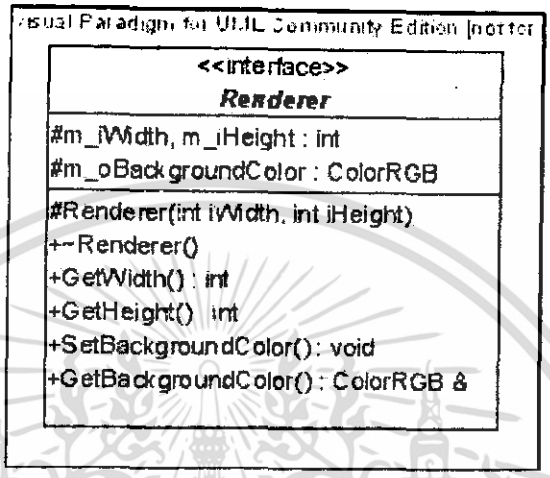
จากรูป จะเห็นได้ว่าในทั้งคลาส Node และ Geometry จะเรียกให้คลาส Culler เรียกฟังก์ชั่น Insert โดยระบุพารามิเตอร์เป็นพอยต์เตอร์เพื่อเก็บไปไว้ในอ็อปเจ็กต์ VisibleSet ของคลาส โดยที่ในส่วนของคลาส Node นั้นจะยังมีการวนลูปตลอดทุกๆ Child ใน Subtree ของมันลงไป โดยแต่ละลูกจะทำการเรียกฟังก์ชั่น OnGetVisibleSet เพื่อทำการคำนวณต่อไป หลังจากนั้นในท้ายสุด จะเรียกฟังก์ชั่น Insert โดยให้ค่าพอยต์เตอร์ของคลาส Spatial เป็น 0 เพื่อระบุว่า Global Effect จะไม่ active

Renderer

คลาส Renderer เป็น Abstract Class ทำการห่อหุ้มกาทำงานของ Graphic API และจัดการควบคุมข้อมูลต่างๆ และทำปฏิสัมพันธ์กับ Object ของคลาสอื่นๆ ในระบบของ

Scene Graph ตลอดจนถึงสร้างระบบทำงานพื้นฐานทั่วไปที่จะต้องใช้ในการแสดงผลออกทางหน้าจอด้วย

ส่วนของโครงสร้างของคลาส Renderer แรก เกี่ยวข้องกับข้อมูลคุณสมบัติทั่วไปที่สัมพันธ์กับหน้าต่างวินโดวส์ (Window) ดังรูปต่อไปนี้

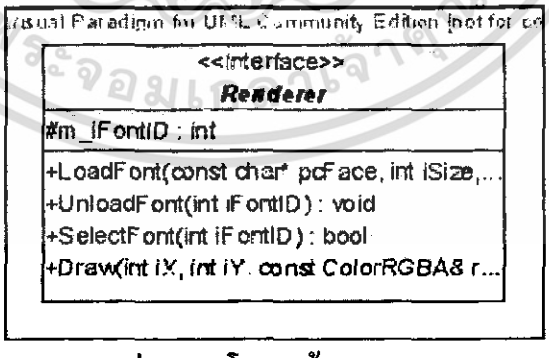


รูปที่ 3.31 แสดงส่วนของโครงสร้างของคลาส Renderer (1)

ซึ่งเป็นข้อมูลที่เกี่ยวข้องกับหน้าต่างวินโดวส์ (Window) ที่จะถูกวาด (Draw) ลงไป

ในส่วนนี้ จะตรงตัว คือทำการเก็บข้อมูลขอบเขตที่จะทำการวาด (Draw) ซึ่งเป็น Client Region ของวินโดวส์ (Window) และค่าสีพื้นหลังที่ Renderer จะใช้ในการเคลียร์พื้นหลัง

ต่อไปเป็น ส่วนของโครงสร้างคลาสที่เกี่ยวข้องกับการแสดงผลตัวอักษร



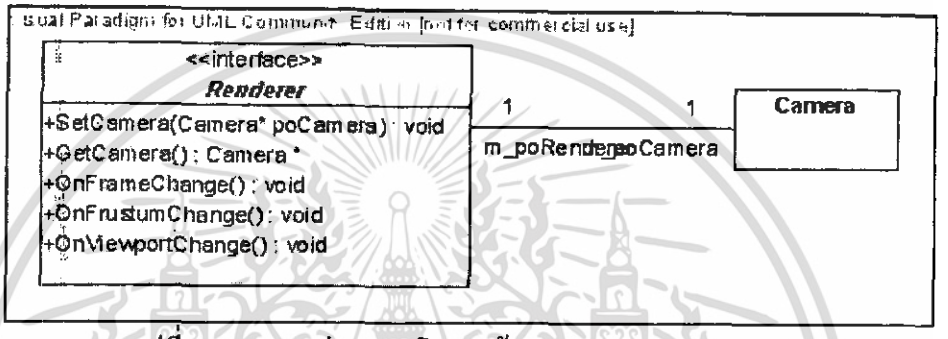
รูปที่ 3.32 แสดงส่วนของโครงสร้างของคลาส Renderer (2)

ซึ่งเกี่ยวข้องกับการวาด (Draw) ตัวอักษรลงบนหน้าจอ

ทั้งนี้ ฟังก์ชั่นจะออกแบบให้อยู่ในรูป Pure Virtual ทั้งหมด จึงไม่มีการอิมพลีเมนต์ (Implement) ใดๆ ดังนั้น คลาสที่สืบทอดต่อจากคลาส Renderer จะต้องระบุการทำงานขึ้น

โดยที่ฟังก์ชั่นต่างๆ มีจุดประสงค์ดังนี้ คือ LoadFont สำหรับสร้าง Font ขึ้นมาใช้งาน, SelectFont เพื่อให้ Renderer รู้ว่าจะวาด (Draw) ตัวอักษรโดยใช้ Font ใด, UnloadFont สำหรับทำลาย Font ที่ระบุ ในขณะที่การทำงานจริงๆ นั้นจะเกิดขึ้นที่ฟังก์ชั่น Draw ซึ่งจะสามารถระบุข้อความและสีได้

Renderer ได้ถูกออกแบบให้ต้องการกำหนดมี Object จากคลาส Camera ให้ เพื่อเป็นการระบุถึงขอบเขตของ Space ที่จะถูกเรนเดอร์ (Render) เป็นส่วนหนึ่งของคลาส ดังนี้

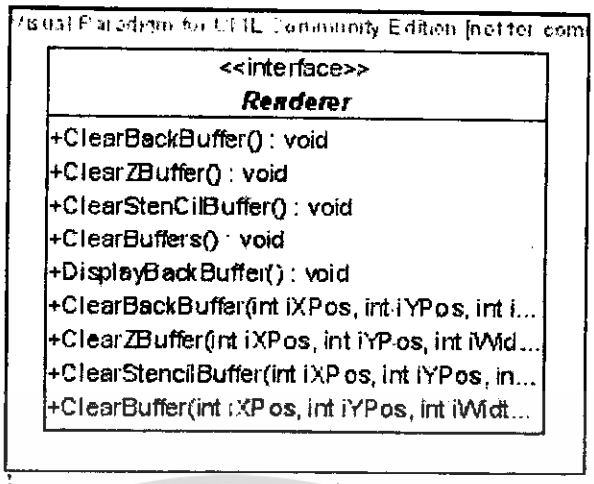


รูปที่ 3.33 แสดงส่วนของโครงสร้างของคลาส Renderer (3) ที่เกี่ยวข้องกับคลาส Camera เพื่อใช้ในการระบุของเขตการเรนเดอร์ (Render)

จากรูปจะเห็นได้ว่าทั้ง 2 คลาส จะติดต่อกันได้ทั้งไปและกลับ (Two-Way Communication) ซึ่งจำเป็นเนื่องจากคลาส Renderer มีความต้องการที่จะต้องร้องขอข้อมูลที่เกี่ยวข้องกับ Camera เช่น View Frustum Parameters และ Coordinate Frame ในขณะที่คลาส Camera เองก็มีความจำเป็นที่จะต้องเตือน Renderer เมื่อมีการเปลี่ยน Camera Parameters เกิดขึ้น

นอกจากนั้นทางคลาส Renderer ก็จะมีฟังก์ชั่นที่ตอบสนองต่อการเปลี่ยนแปลงของกล้อง คือ OnFrameChange, OnFrustumChange และ OnViewportChange เพื่อให้คลาสที่สืบทอดต่อจาก Renderer ไปทำการ implement ต่อ ทั้งนี้ กลุ่มฟังก์ชั่นดังกล่าวเป็นฟังก์ชั่นแบบ Virtual ทั้งหมด

เนื่องจากว่ามีทรัพยากรมากมายที่เกี่ยวข้องกับ Renderer ตั้งแต่ Frame Buffer ในการเก็บค่าสีของ Pixel, Back Buffer ในการทำ Double-Buffered Drawing, Depth Buffer สำหรับเก็บค่าความลึกของ Pixel นั้นๆ และ Stencil Buffer สำหรับ Effect พิเศษ ดังนั้น จึงมีโครงสร้างที่เกี่ยวข้อง ดังนี้



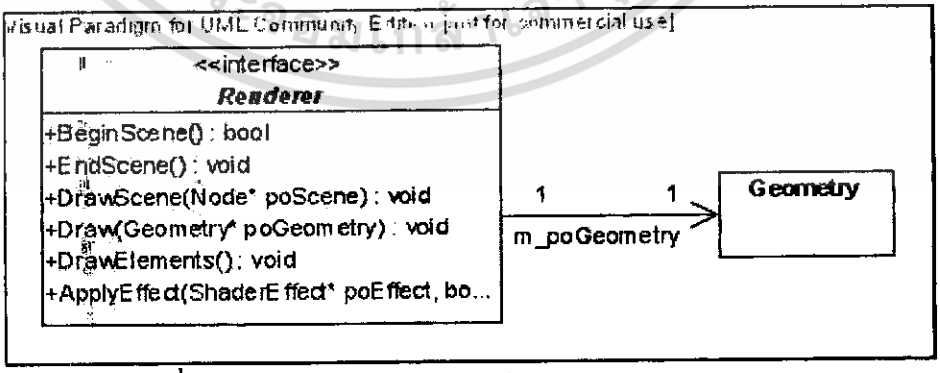
รูปที่ 3.34 แสดงส่วนของโครงสร้างของคลาส Renderer (4)
ซึ่งเกี่ยวข้องกับการจัดการกับ Buffer ที่เป็น Resource ที่เกี่ยวข้อง

ทั้งนี้ ทุกฟังก์ชั่นที่ใช้ในการเคลียร์จะเป็น Pure Virtual ทั้งหมด (ในรูปแบบเดียวกับส่วนที่เกี่ยวข้องกับ Font) ส่วนฟังก์ชั่น DisplayBackBuffer นั้น จะใช้ในการสั่งให้ Graphic System ทำการ copy Back Buffer ไปยัง Front Buffer

นอกจากนี้คลาส Renderer ยังเก็บข้อมูลเกี่ยวกับจำนวนของ Light และ Texture มากที่สุดที่รองรับได้ ซึ่งในส่วนนี้จะได้ค่าจากการตรวจสอบระบบ คลาสจะมีฟังก์ชั่นในการให้ข้อมูลเท่านั้น

Object Drawing

ในทั้งระบบนั้น หลังจากที่จบการทำ Scene Management เรียบร้อยแล้ว ก็จะเป็น Drawing System โดยมีโครงสร้างที่เป็นจุดเริ่มต้นการทำงานระบบดังกล่าว ดังรูปต่อไปนี้



รูปที่ 3.35 แสดงส่วนของโครงสร้างของคลาส Renderer (5)
ซึ่งเกี่ยวข้องกับการทำงานของ Drawing System

คู่ฟังก์ชั่น BeginScene และ EndScene มีเพื่อให้ Graphic System สามารถที่จำทำ Operation ใดๆ ก่อนและหลังการวาด (Draw) ได้ โดยที่จุดนี้คลาส Renderer จะปล่อยว่างไว้ ให้คลาสที่สืบทอดต่อจัดการ ส่วน DrawScene นั้น ได้ถูกออกแบบให้เป็น Entry Point ของ Drawing System

ตัวอย่าง Code การทำงานโดยปกติทั่วไปของการเรนเดอร์ (Render) ที่เกิดขึ้นใน Idle-Loop Callback จะอยู่ในรูปแบบต่อไปนี้

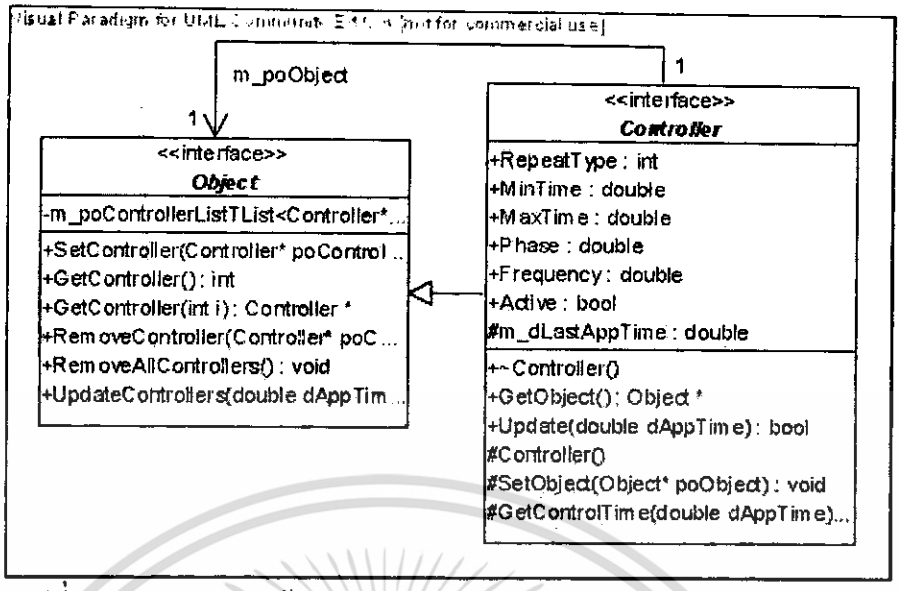
```
m_poRenderer->ClearBuffer();
if (m_poRenderer->BeginScene())
{
        m_poRenderer->DrawScene(m_poScene);
        m_poRenderer->EndScene();
}
m_poRenderer->DisplayBackBuffer();
```

การทำงานของฟังก์ชั่น DrawScene จะเป็นพื้นที่ในการจัดการเกี่ยวกับการวาดวัตถุทั้งฉาก แล้วจึงเรียกฟังก์ชั่น Draw สำหรับแต่ละอ็อปเจ็กต์ของ Geometry ซึ่งจะดึงเอาคุณสมบัติต่างๆ รวมไปถึง Effect ที่เกี่ยวข้องขึ้นมาให้ค่า ก่อนที่จะทำการวาด และคืนค่าเมื่อทำการวาดเสร็จแล้ว

ทั้งนี้ ฟังก์ชั่นที่ใช้ในการวาดภาพขึ้นจริงๆ คือ ฟังก์ชั่น DrawElements ซึ่งจะทำการตรวจสอบประเภท (Type)ของอ็อปเจ็กต์ Geometric ที่จะถูกวาด

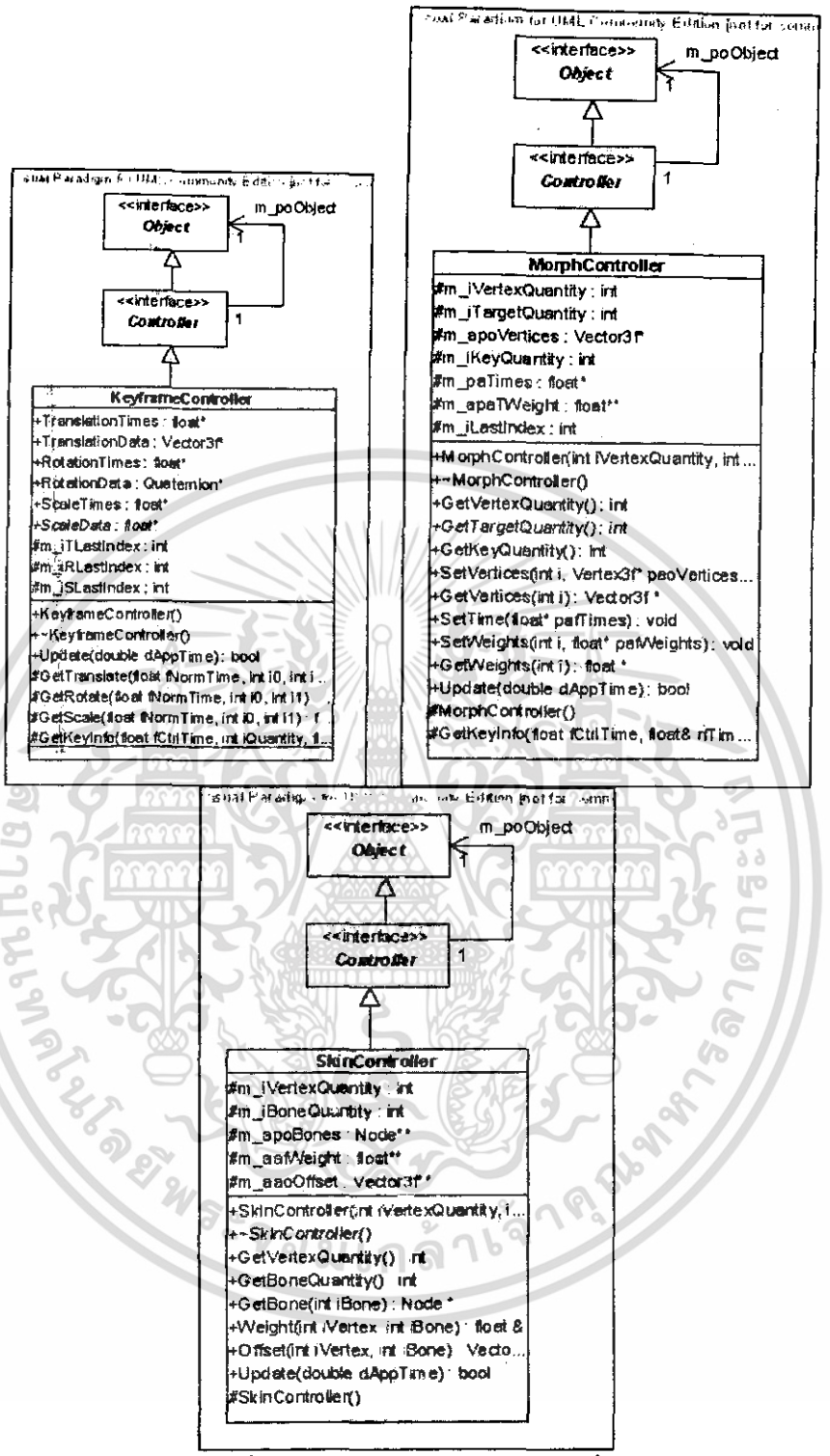
3.5.4 Controller

คลาส Controller เป็นส่วนที่ควบคุมจัดการกับค่าต่างๆ ที่เปลี่ยนไปตามเวลา โดยที่ตัวคลาสจะออกแบบมาให้เป็นคลาสแบบ Abstract Base โดยที่มีส่วนของโครงสร้างดังต่อไปนี้



รูปที่ 3.36 แสดงโครงสร้างของคลาส Controller และส่วนของโครงสร้างของคลาส Object ที่สนับสนุนการทำงานของคลาส Controller

คลาส Controller จะ ประกอบไปด้วยข้อมูลต่างๆ ที่เกี่ยวข้องกับการทำ Animation และ พอยต์เตอร์ไปยังคลาส Object ที่ตัวคลาส Controller กำลังควบคุมและจัดการอยู่ พร้อมทั้งฟังก์ชั่นในการกำหนดค่า ในขณะที่คลาส Object จะทำการเก็บ Link List ของคลาส Controller และมีฟังก์ชั่นให้ค่าคืนค่าเบื้องต้น และฟังก์ชั่นในการปรับค่า Controller ตามค่าเวลาของแอพลิเคชั่นที่เปลี่ยนแปลงไป



รูปที่ 3.37 แสดงโครงสร้างของคลาสที่สืบทอดจากคลาส Controller เพื่อรองรับ Animation ประเภทต่างๆ คือ คลาส KeyframeController, MorphController, และ SkinController จากซ้ายไปขวา

จากรูป แต่ละคลาสจะสนับสนุนการทำ Animation ต่างๆ คือ Keyframe Animation, Morph Animation, และ Skeleton Animation ซึ่งมีข้อมูลที่จะต้องใช้ในการทำงานที่แตกต่างกัน อย่างไรก็ตามทุกคลาสที่สืบทอดมาจากคลาส Controller จะต้องทำ

การโอเวอร์ไรด์ฟังก์ชั่น Update ซึ่งจะส่งค่าเวลาของแอพพลิเคชั่นเพื่อเอาไปใช้ในการคำนวณและปรับค่าข้อมูลต่างๆ ภายในคลาส

3.5.5 Terrain

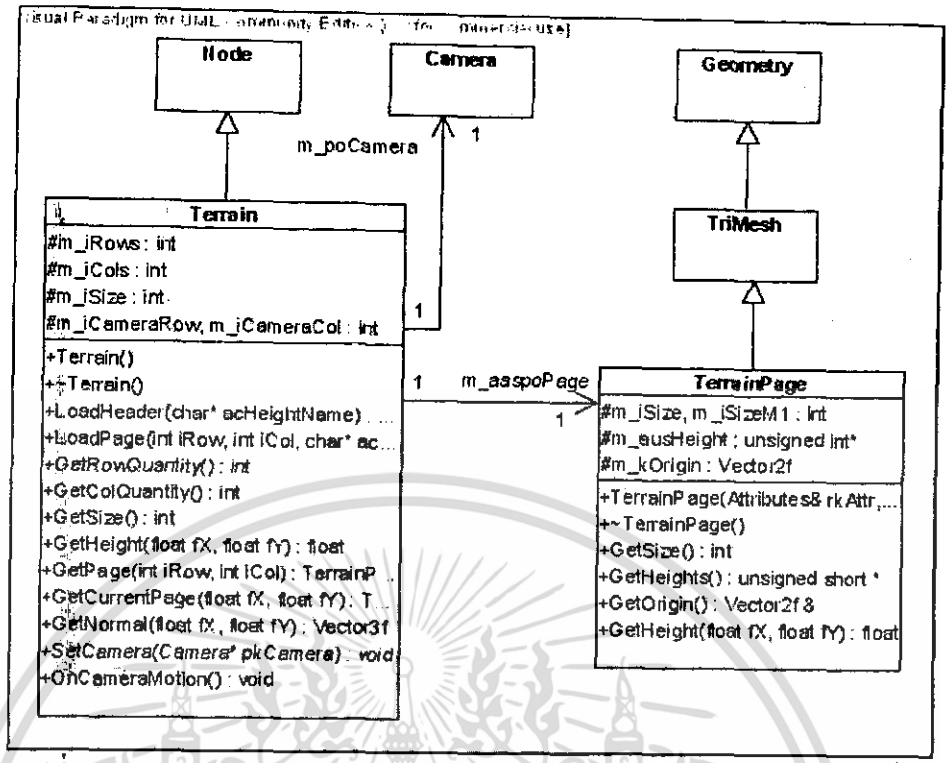
โดยปกติแล้ว Terrain จะประกอบไปด้วยสามเหลี่ยมจำนวนที่มากเมื่อเทียบโมเดล ทั้งนี้ วิธีจัดการที่ดีที่สุด คือ การแบ่ง Terrain ออกเป็นส่วนย่อยลง ซึ่งเรียกว่า Page อย่างไรก็ตาม ยังคงใช้หลักการของ Scene Graph อยู่คือ ให้ Page เก็บอยู่ในรูปของคลาส Geomerty โดยกำหนดให้เป็นสี่เหลี่ยมจัตุรัสขนาดเท่ากันทั้งหมด เพื่อง่ายต่อการคำนวณ

Height Field File Format

ในเอ็นจิ้นใช้ไฟล์นามสกุล *.pthf ซึ่งเป็นรูปแบบของไฟล์ Height Field ปกติทั่วไป โดยที่เริ่มต้นจะเป็นส่วนของ Header ซึ่งระบุถึง จำนวน Page ทั้งหมด, จำนวนแถว, จำนวนหลัก, และขนาดของ Page ต่อจากนั้นจะเป็น Height Field ของแต่ละ Page เรียงจากซ้ายไปขวา และจากบนลงล่าง

Terrain

ในการที่คลาส Terrain จะนำข้อมูลมาใช้งาน ก็จะเริ่มจากการอ่าน Header จากนั้นจึงสร้าง Page พร้อมทั้งติดตั้งให้กับ Terrain Node โดย Page ที่สร้างเป็น Plane ที่แบ่ง Segment ตามระยะห่างของแต่ละ Vertex ที่กำหนดไว้ โดยแต่ละ Vertex จะมีความสูงตามที่ปรากฏใน Height Field Data รวมถึงการคำนวณค่า U, V ให้กับแต่ละ Vertex ด้วย ทั้งนี้ ในระบบ จะประกอบไปด้วยคลาส Terrain และ TerrainPage ซึ่งมีโครงสร้างดังต่อไปนี้

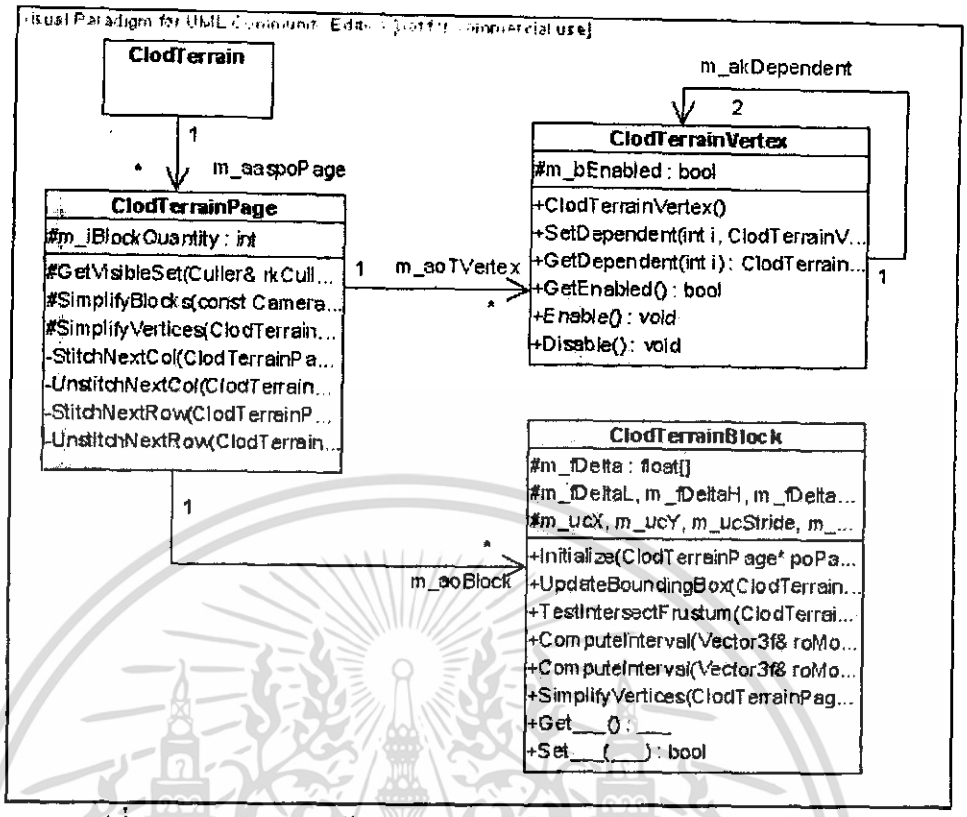


รูปที่ 3.38 แสดงโครงสร้างของคลาส Terrain, TerrainPage และความสัมพันธ์ซึ่งกันและกัน

ClodTerrain

เป็นระบบควบคุมจัดการสำหรับ Rectangular Grid ของ Terrain Page แบบสี่เหลี่ยมจัตุรัส แต่จะมีการทำ Page Tessellation อย่าง dynamic โดยใช้อัลกอลิทึม Continuous Level Of Detail เพื่อที่จะลดความละเอียดของ Terrain ในบางตำแหน่ง เช่น ตำแหน่งที่อยู่ไกลจากมุมกล้อง ตำแหน่งที่มีพื้นผิวในลักษณะราบเรียบ

แต่ละ Page หนึ่งๆ จะถูกแบ่งออกเป็น Block ซึ่งขนาดขึ้นอยู่กับจำนวน Pixel ที่ Block นั้นปรากฏบนจอภาพ ซึ่งอาจจะมีขนาดไม่เท่ากันได้ ทั้งนี้ Vertex ที่มุมและตรงกลางของ Block จะสามารถถูก Enable หรือ Disable ได้ เพื่อปรับระดับความละเอียดของ Mesh



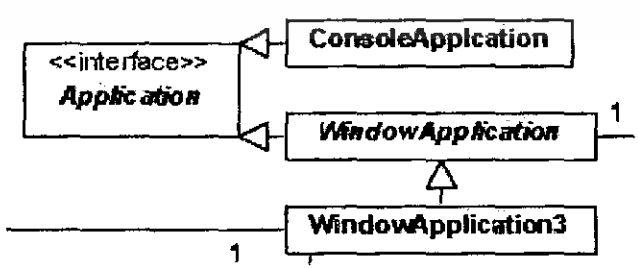
รูปที่ 3.39 แสดงโครงสร้างของคลาส ClodTerrain, ClodTerrainPage, ClodTerrainBlock, ClodTrrrainVertex และความสัมพันธ์ซึ่งกันและกัน

3.6 ส่วน Renderers

เป็นส่วนประกอบที่นำคลาส Renderer มาสืบทอดต่อเพื่อ implement ให้ใช้งาน Graphics API ต่างๆ ดังต่อไปนี้

1) Dx9Renderer จะเรียกใช้งานจาก DirectX Graphic
2) WglRenderer จะเรียกใช้งานจาก OpenGL

3.7 ส่วน Applications

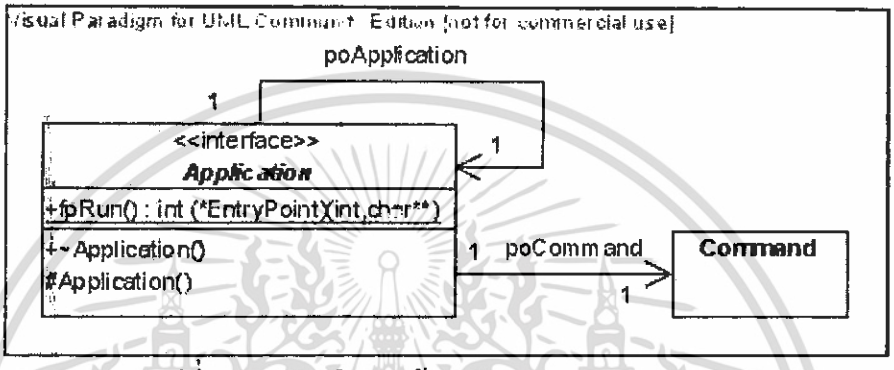


รูปที่ 3.40 แสดงส่วนหนึ่งจากรูปที่ 3.1 ซึ่งเป็นโครงสร้างโดยคร่าวของ Application

เป็นส่วนประกอบที่ห่อหุ้มข้อมูลในการสร้างโปรแกรมประยุกต์ในรูปแบบต่างๆ มีคลาสดังนี้

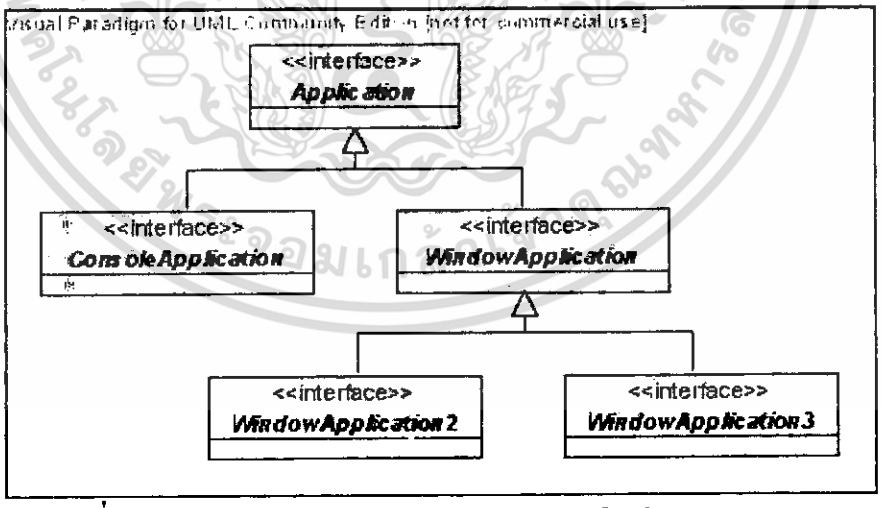
3.7.1 Application

คลาส Application เป็นคลาสแบบ Abstract Base ที่สร้างไว้เพื่อให้คลาสอื่นๆ นำไปสืบทอดต่อ โดยมีโครงสร้างดังต่อไปนี้



รูปที่ 3.41 แสดงโครงสร้างของคลาส Application

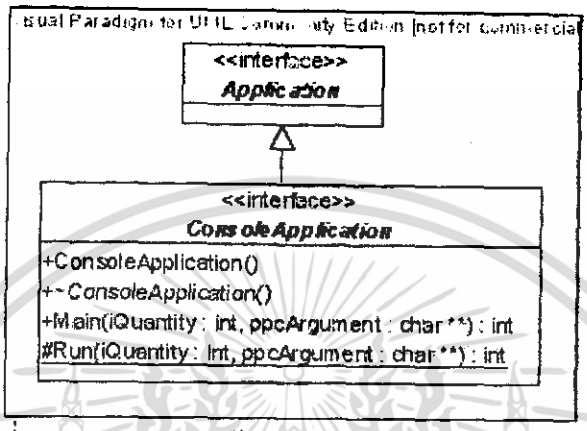
โดยที่ poCommand จะเป็นพอยต์เตอร์ที่ชี้ไปที่อ็อปเจ็กต์จากคลาส Command ที่จัดการกับ Argument ที่ผู้ใช้ป้อนเมื่อตอนสั่งแอพลิเคชั่นให้ทำงาน



รูปที่ 3.42 แสดง Hierarchy Chart ของกลุ่มคลาสในส่วน Application

3.7.2 ConsoleApplication

คลาส ConsoleApplication สืบทอดจากคลาส Application โดยเป็นเบสคลาส (Base Class) ที่ใช้ในการสร้างแอพลิเคชั่นที่อยู่ในลักษณะของคอนโซล (Console) ซึ่งไม่ได้ต้องการใช้หน้าต่างวินโดวส์ (Window) สำหรับการแสดงผล มีโครงสร้างดังรูปต่อไปนี้



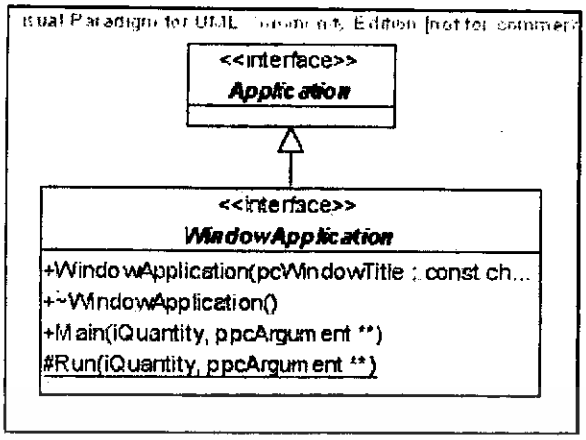
รูปที่ 3.43 แสดงโครงสร้างของคลาส ConsoleApplication

ในการใช้งานจริง ผู้ใช้จะต้องทำการสร้างคลาสสืบทอดต่อจากคลาส ConsoleApplication โดยมีส่วนที่สามารถโอเวอร์ไรด์ (Override) ได้คือ ฟังก์ชั่น Destructor และฟังก์ชั่น Main (โดยส่วนของฟังก์ชั่น Main นั้น ผู้ใช้จะต้องอิมพลีเมนต์ (Implement) ขึ้นมาเอง เนื่องจากเป็นส่วนการทำงานจริงของแอพลิเคชั่น จึงไม่ได้ระบุการทำงานใดๆ ไว้ที่คลาส ConsoleApplication

ในส่วนของฟังก์ชั่น Run จะทำการสร้างตัวออปเจ็คต์ของ ConsoleApplication ขึ้นมาและสั่งให้เรียกใช้งานฟังก์ชั่น Main ซึ่งเป็นการทำงานจริง ดังที่กล่าวไว้ข้างต้น

3.7.3 WindowApplication

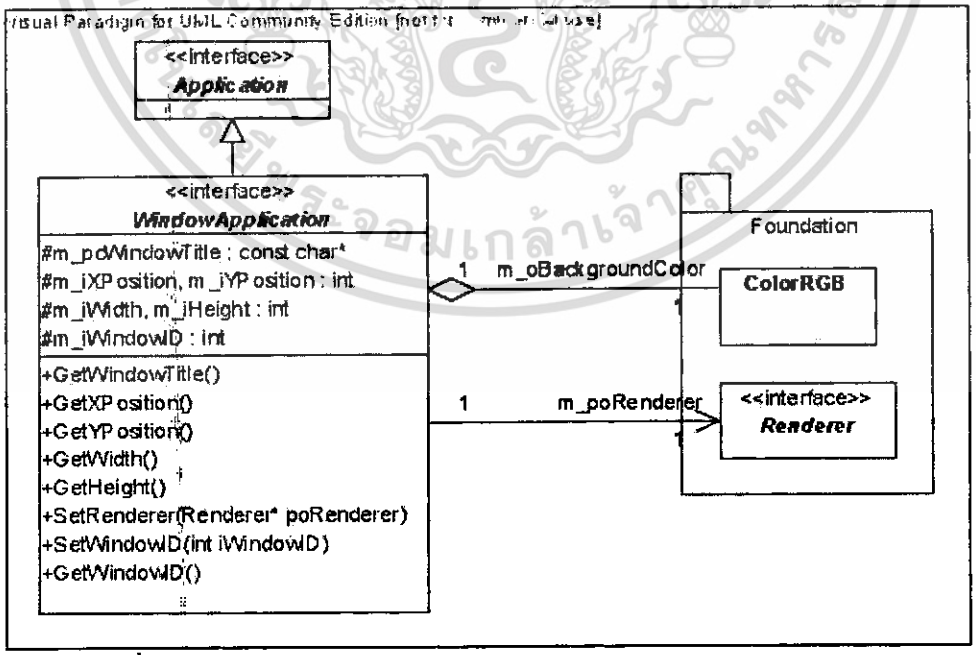
คลาส WindowApplication สืบทอดจากคลาส Application โดยเป็นเบสคลาส (Base Class) ที่ใช้ในการสร้างแอพลิเคชั่นที่ใช้หน้าต่างวินโดวส์ (Window) ในการแสดงผล ซึ่งจะมีส่วนของโครงสร้างที่คล้ายกับคลาส ConsoleApplication ดังรูปต่อไปนี้



รูปที่ 3.44 แสดงส่วนของโครงสร้างของคลาส WindowApplication (1)
โดยเป็นส่วนที่คล้ายกับคลาส ConsoleApplication

ในส่วนของฟังก์ชั่น Run ของคลาส WindowApplication นั้น จะยังคงทำงานเช่นเดียวกับของคลาส ConsoleApplication

ในการใช้งานจริง จะมีสิ่งที่แตกต่างไปจากคลาส ConsoleApplication คือ คลาสที่สืบทอดต่อจาก WindowApplication นั้นจะต้องอยู่ในรองรับรูปแบบของฟังก์ชั่น Constructor ของคลาส (กล่าวคือ จะต้องมีการเรียก Constructor ของคลาส WindowApplication ใน Constructor ของคลาสที่สืบทอดต่อ) และส่งค่าให้ด้วย เนื่องจากจะต้องนำไปให้ค่าเริ่มต้นกับเดต้าของคลาส ดังจะเห็นได้จากส่วนของโครงสร้างต่อไปนี้



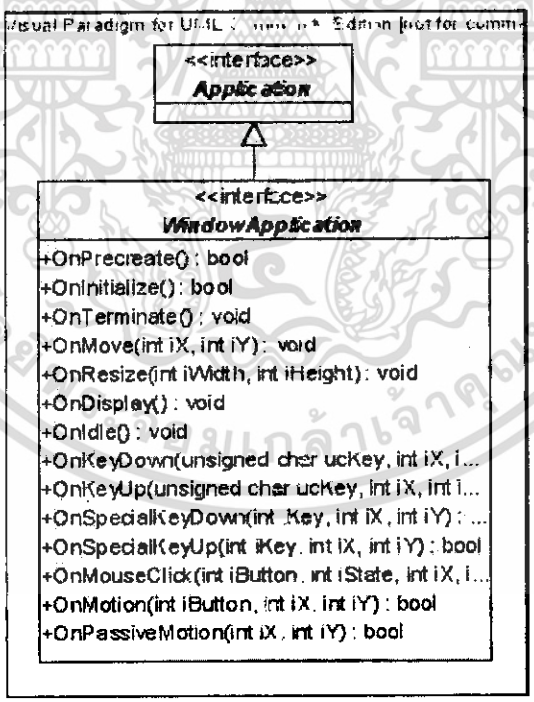
รูปที่ 3.45 แสดงส่วนของโครงสร้างของคลาส WindowApplication (2)
โดยเป็นส่วนที่เกี่ยวข้องกับเดต้าของคลาส

โดยที่เดต้าต่างๆ ภายในคลาสจะเก็บข้อมูลรายละเอียดของหน้าต่างวินโดวส์ (Window) ได้แก่ หัวข้อของ Title Bar, ตำแหน่งแกน X, ตำแหน่งแกน Y, ขนาดความกว้าง, ขนาดความสูง, สีที่ใช้ในการเคลียร์พื้นหลัง (คลาส ColorRGB) นอกจากนั้นจะเก็บเดต้าที่อ้างอิงออปเจ็กต์ของคลาส Renderer ที่เป็นคลาสที่จัดการกระบวนการดรอว์ (Draw) เพื่อแสดงภาพบนหน้าต่างการทำงาน

Event Handling

เนื่องจากทุกระบบแบบวินโดวส์ (Window) จะต้องมีกระบวนการสำหรับจัดการกับเหตุการณ์ต่างๆ (Event Handling) เช่น การกดคีย์, การคลิ๊กหรือการเคลื่อนที่ของเมาส์, รวมไปถึงการเปลี่ยนตำแหน่งและเปลี่ยนขนาดของวินโดวส์ นอกจากนั้นยังมีการวาดภาพใหม่และการปล่อยว่าง (Idle) เมื่อคิวของเหตุการณ์ (Event Queue) ว่าง

ภายในคลาสของ WindowApplication ได้สร้างกลุ่มของ Event Callback (เป็นฟังก์ชั่นที่ถูกเรียกใช้งานเมื่อเกิดเหตุการณ์ใดๆ ขึ้น) ดังรูปต่อไปนี้



รูปที่ 3.46 แสดงส่วนของโครงสร้างของคลาส WindowApplication (3) โดยเป็นส่วนที่เกี่ยวข้องกับ Event Callback

ทั้งนี้ หากเปรียบเทียบกับมาโคร Window Message ของระบบปฏิบัติการ Window แล้ว จะได้ตามตารางต่อไปนี้

ชื่อฟังก์ชั่น	มาโคร	หมายเหตุ
OnMove	WM_MOVE	
OnResize	WM_RESIZE	
OnDisplay	WM_PAINT	
OnKeyDown	WM_CHAR	Special Key จะประกอบไปด้วย Arrow Key, Insert, Delete, Home, End, Page Up, Page Down และ Function Key
OnKeyUp	WM_KEYUP	
OnSpecialKeyDown	WM_KEYDOWN	
OnSpecialKeyUp	WM_KEYUP	
OnMouseClick	WM_[L/M/R]BUTTON [DOWN/UP]	แตกต่างกันไปค่าของ Argument ที่ฟังก์ชั่นได้รับ
OnMotion	WM_MOUSEMOVE	มีการกดปุ่มเมาส์อยู่ด้วย (Drag)
OnPassiveMotion	WM_MOUSEMOVE	ไม่มีการกดปุ่มเมาส์

ตารางที่ 3.2 แสดงการเปรียบเทียบระหว่างฟังก์ชั่นในส่วน Event Callback ของคลาส WindowApplication กับมาโคร Window Message ของระบบปฏิบัติการ Window

อีกสิ่งหนึ่งที่แตกต่างไปจากคลาส ConsoleApplication ก็คือ คลาส WindowApplication นั้นได้สร้างการทำงานพื้นฐานไว้ให้บนฟังก์ชั่น Main ของคลาส ซึ่งผู้ใช้สามารถที่จะนำไปใช้เลยหรือทำการโอเวอร์ไรด์ (Override) ใหม่ตามความต้องการก็ได้ ทั้งนี้การทำงานพื้นฐานโดยทั่วไปดังกล่าว ได้นำเอาฟังก์ชั่นในส่วนที่เกี่ยวข้องกับ EventCallback ที่เหลือ (ที่ไม่ได้ปรากฏอยู่บนตารางเปรียบเทียบ คือ ฟังก์ชั่น OnInitialize, OnTerminate, OnPrecreate และ OnIdle) มาเรียกใช้ตามเหตุการณ์ต่างๆ โดยจะมีขั้นตอนดังรูปต่อไปนี้

```
int WindowApplication::Main (...)
{
    (1) งานต่างๆ ก่อนการสร้างหน้าต่างวินโดวส์ (Window) ขึ้น
       (ฟังก์ชั่น OnPrecreate)
    (2) สร้างหน้าต่างวินโดวส์ (Window) ขึ้นมา
    (3) สร้าง Renderer ขึ้นมา
    (4) ให้ค่าเริ่มต้นกับแอพพลิเคชั่น
       (ฟังก์ชั่น OnInitialize)
    (5) แสดงหน้าต่างวินโดวส์ (Window)
    do
    {
        if ( มี Message อยู่ )
        {
            (7) ถ้า Message ต้องการให้จบการทำงาน, ออกจากลูป
            (8) กำจัด Message ทิ้ง
        }
        else
        {
            (9) ปล่อยว่าง, ไม่ได้ทำอะไร
               (ฟังก์ชั่น OnIdle)
        }
    }
    (9) ออกจากแอพพลิเคชั่น
       (ฟังก์ชั่น OnTerminate)
}
```

รูปที่ 3.47 แสดงขั้นตอนการทำงานพื้นฐานในฟังก์ชั่น Main ของคลาส WindowApplication ได้ทำการสร้างไว้ให้ผู้ใช้งานไว้แล้ว

ในส่วนการใช้งานของฟังก์ชั่นพวก Event Callback นั้น ผู้ใช้งานสามารถที่จะโอเวอร์ไรด์ (Override) บนคลาสที่สืบทอดต่อจากคลาส WindowApplication โดยจะจัดการกับเหตุการณ์ใดๆ อย่างไร ได้ตามความต้องการ

3.7.4 WindowApplication3

คลาส WindowApplication3 สืบทอดจากคลาส WindowApplication โดยเป็นเบสคลาส (Base Class) ที่ใช้ในการสร้างแอพลิเคชั่นที่ใช้หน้าต่างวินโดวส์ (Window) ในการแสดงผลแบบ 3 มิติ ซึ่งในตัวคลาสจะทำการโอเวอร์ไรด์ (Override) ฟังก์ชั่นพวก Event Callback ให้ทำงานตามขั้นตอนโดยทั่วไปของการแสดงผลแบบ 3 มิติไว้บางส่วน ได้แก่ ฟังก์ชั่น OnInitialize, OnTerminate, OnDisplay, OnKeyDown, OnSpecialKeyDown, OnSpecialKeyUp, OnMouseClick, OnMotion ซึ่งหากผู้ใช้ต้องการเปลี่ยนแปลงก็สามารถที่จะอิมพลีเมนต์ (Implement) เองบนคลาสที่สืบทอดต่อจากคลาสนี้ได้

นอกจากนั้น คลาส WindowApplication3 ได้เพิ่มเติมระบบต่างๆ ที่ช่วยในการแสดงผลแบบ 3 มิติเพิ่มเข้าไปให้ คือ Camera Motion, Object Motion และ Performance Measurements ทำให้สามารถใช้งานพวกการป้อนค่าคีย์พื้นฐานได้ ซึ่งจะกล่าวถึงในบทถัดไป

3.8 Plug-in Exporter

3.8.1 Prototype1 Exporter

Prototype1 Exporter เป็นคุณสมบัติเพิ่มเติม (Plug-in) ของโปรแกรม 3Dstudio Max ที่ใช้กับโมเดลและสิ่งแวดล้อม เพื่อสร้างโมเดลและสิ่งแวดล้อมที่มี File Format และคุณสมบัติตามต้องการ โดยเริ่มจากการสร้างโมเดลและสิ่งแวดล้อมจากโปรแกรม 3Dstudio Max จากนั้นทำการ Export ก็จะได้โมเดลและสิ่งแวดล้อมนามสกุล PTOF ที่มีคุณสมบัติและโครงสร้างของไฟล์ตามที่กำหนดไว้สำหรับ Engine ที่พัฒนาขึ้นมา

3.8.2 วิธีการสร้าง Plug-in Exporter

การสร้างไฟล์ของ Plug-in Exporter นั้นจะต้องทำการสร้างไฟล์ที่เป็น Dynamic Link Libraries (DLL) ให้สามารถใช้งานได้กับโปรแกรม 3DStudio Max เพื่อทำการเชื่อม Plug-in เข้ากับโปรแกรม 3DStudio Max ขณะ Runtime หลังจากนั้น Plug-in ก็จะทำการแปลงไฟล์เป็น Prototype1 (PTOF) โดยมีวิธีการพัฒนาดังนี้

การสร้าง Plug-in และองค์ประกอบที่สำคัญ

ในส่วนนี้จะบอกถึงฟังก์ชั่นที่สำคัญที่ใช้ในการสร้าง Plug-in DLL ซึ่งเป็นตัวที่ระบุถึง Object ที่เรียกว่า Class Descriptor ซึ่งผู้พัฒนาจำเป็นจะต้องระบุรายละเอียดในแต่ละ Plug-in class

3DStudio Max จำเป็นจะต้องพัฒนาเป็น Dynamic Link Libraries (DLLs) โดย DLLs เป็น Object code libraries ที่อนุญาตให้หลายๆโปรแกรมสามารถใช้งานได้ไม่ว่าจะเป็นส่วน Code, Data และ Resources เมื่อทำการ Compile และ Link Plug-in code ผลลัพธ์ที่ได้จะเป็น DLL

การสร้าง Plug-in DLL นั้นจะต้องประกอบด้วย Function ที่สำคัญดังนี้

- DllMain() เป็นฟังก์ชั่นที่ Windows เรียกใช้ตอน Startup เมื่อ Plug-in DLL ถูกโหลด ฟังก์ชั่น DllMain จะทำการ Hook ด้วย Window เพื่อทำการ Initialize DLL โดย DllMain จะไม่ทำอะไรนอกจากเป็น Instance ของ DLL เอง

- LibDescription() เป็นฟังก์ชั่นที่จะคืนค่าเป็นสตริงเพื่อแสดงให้กับผู้ใช้หากไม่ได้ใช้ DLL

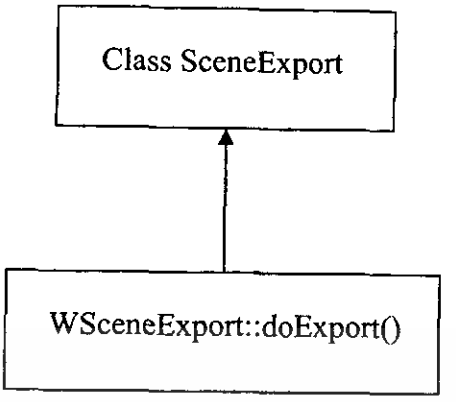
- LibNumberClasses() เป็นฟังก์ชั่นที่คืนค่าของจำนวน Plug-in ภายใน DLL

- LibClassDesc() เป็นฟังก์ชั่นที่คืนค่าของ Pointer ที่ชี้ไปยัง Object ที่เรียกว่า Class Descriptor สำหรับแต่ละ Plug-in Class ใน DLL และเป็นตัวบอก Properties ของแต่ละ Plug-in Class และวิธีการ Allocate Instance ของ Class ในหน่วยความจำ

- LibVersion() เป็นฟังก์ชั่นที่อนุญาตให้ระบบสามารถใช้กับ Plug-in DLLs ในเวอร์ชั่นเก่าๆได้

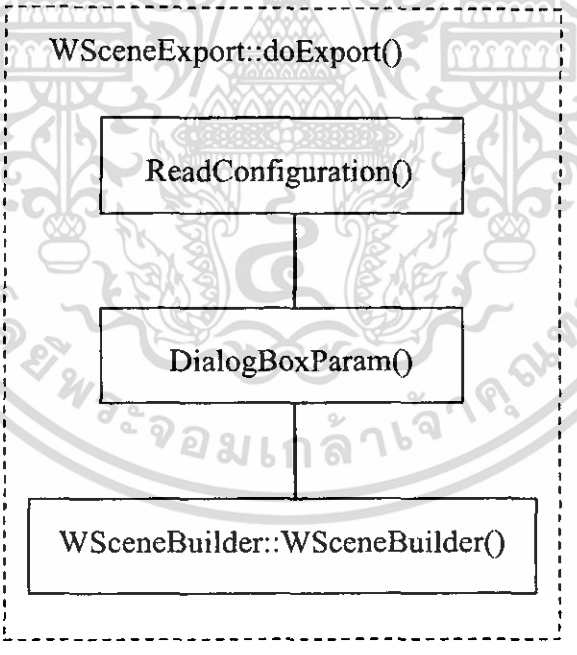
ลำดับการทำงานของ Exporter

เมื่อทำการกดปุ่มเลือก Plug-in Exporter โปรแกรมจะเรียก Function DoExport ที่ได้ทำการ Override มาจาก Class SceneExport ซึ่งเป็น Class พื้นฐานที่ใช้ในการสร้างไฟล์ Export Plug-ins โดย Plug-in ทำการ Implement method ต่างๆของคลาส SceneExport เพื่อใช้ในการอธิบายคุณสมบัติของ Export Plug-in และวิธีในการที่จะจัดการกับกระบวนการ Export จริงๆ ซึ่ง Function DoExport มีหน้าที่ โหลดค่า Configuration, การแสดง User Interface และ ทำการสร้างและบันทึก Scene graph ดังรูป



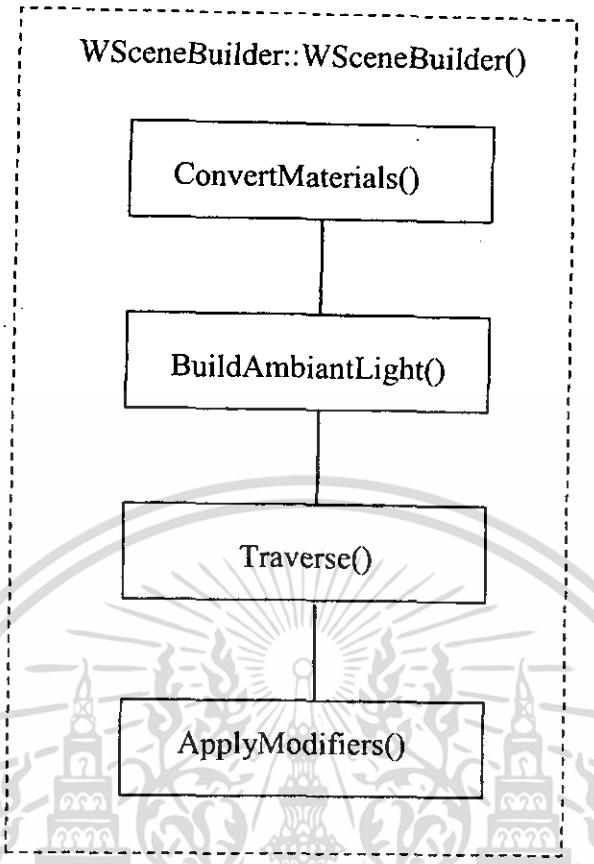
รูปที่ 3.48 แสดงคลาสหลักในการสร้าง Exporter

ภายใน Function doExport ก็จะทำการแสดงผลของ User Interface จาก Function DialogBoxParam และทำการเปลี่ยนส่วนต่างๆของไฟล์จาก 3DStudio Max เป็น Prototype1 โดยเรียก Constructor WSceneBuilder ซึ่งมีชื่อไฟล์ที่ได้กำหนดเป็นอินพุต ดังรูป



รูปที่ 3.49 แสดงการ Implement ในส่วน doExport

Class WSceneBuilder มีหน้าที่หลักในการเปลี่ยนส่วนต่างๆของไฟล์เป็น Prototype1 โดยจะทำการแปลงในส่วนของ Material, Lighting, Traverse Node และใส่ Modifier จากนั้นก็ทำการบันทึกไฟล์ที่แปลงแล้วเป็นนามสกุล PTOF



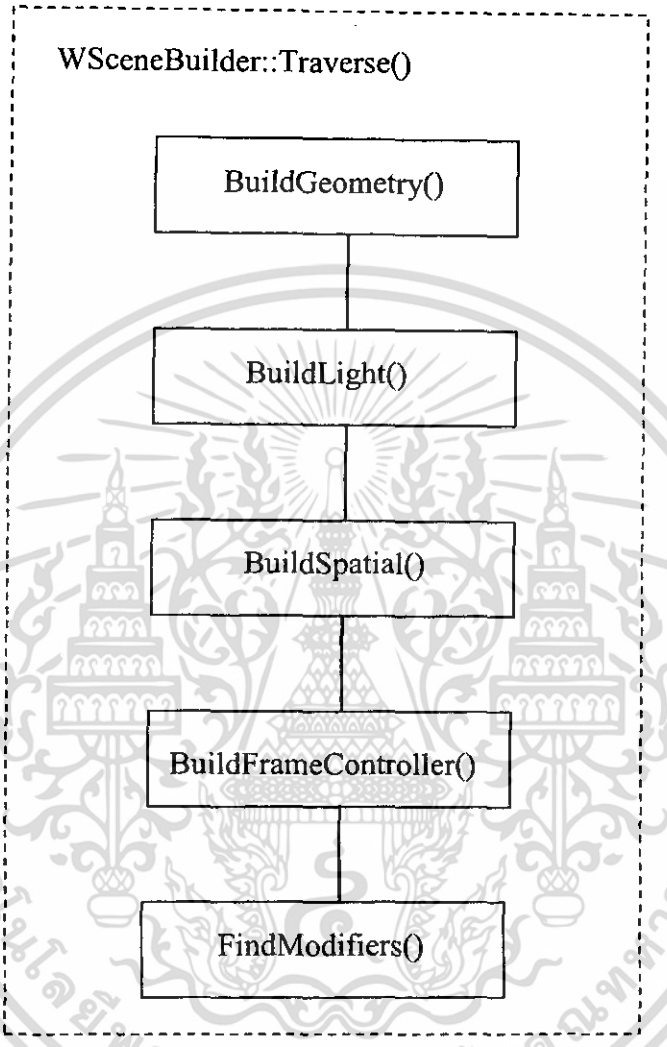
รูปที่ 3.50 แสดงการ Implement ในส่วน WSceneBuilder

การแปลง Material นั้นจะเรียก Function ConvertMaterials เพื่อทำการตรวจสอบว่ามี Material หรือไม่ ถ้ามีจะทำการคำนวณว่ามีจำนวนกี่อัน แล้วทำการสร้างเป็น Material ของ Prototype1 โดยใช้ Function ConvertMaterial เริ่มจากการสร้าง Prototype1 Material จาก 3DStudio Max Material แล้วค่อยทำการ แปลง Material จาก 3DStudio Max มาเป็น Prototype1

การใส่แสงจะทำโดยรับค่า Settings ของ Global ambient light จาก 3DStudio Max แล้วทำการสร้างใหม่ให้เหมือนกันในรูปแบบของ Prototype1 โดยเรียกจาก Function BuildAmbientLight

การทำ Traverse โหนดทุกโหนดของโมเดลและสิ่งแวดล้อมเพื่อแปลงให้เป็น Prototype1 มีการทำงานเป็น Recursive เพื่อที่จะสามารถแก้ไขในโหนดย่อยๆทั้งหมด การทำงานหลักของ Function Traverse คือจะใช้โหนดในการจัดกลุ่มของ Geometry และโหนดนั้นๆจะเป็น Parent ของ Geometry ที่ทำการจัดกลุ่ม เมื่อได้โหนดมาก็จะทำการเรียก Function Traverse แบบ Recursive เพื่อทำการแปลงส่วนประกอบของโมเดลทั้งหมด ซึ่งส่วนประกอบที่ Prototype1 ได้นำมาแปลงได้แก่ Geometry Object, Light

Object, Helper Object, Animation Information และ Modifier ซึ่งองค์ประกอบทั้งหมดก็คือความสามารถในการแปลงของ Prototype1 นั่นเอง



รูปที่ 3.51 แสดงการ Implement ในส่วน Traverse

3.8.3 ค่า Configuration ของ Exporter

ค่า Configuration ของ Exporter คือส่วนที่จะกำหนดว่าจะทำการแปลงไฟล์ในส่วนใดบ้าง โดยจะแบ่งเป็น 4 ส่วนคือ

1. Object Settings จะทำการแปลงในส่วน Object, Light, Meshes
2. Mesh Settings จะทำการแปลงในส่วน Materials, Texture Coordinates, Normal, Generate Maps
3. Modifier Settings จะทำการแปลงในส่วน Modifiers, Skins
4. Animation Settings จะทำการเลือก Frame ในการแปลงเป็นทุกๆ Frame

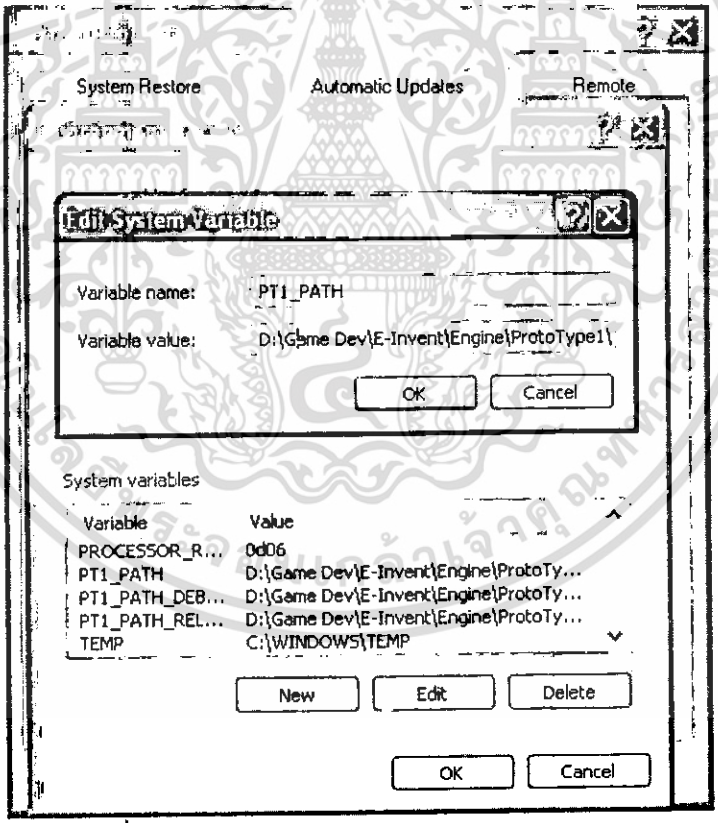
บทที่ 4
การทดลองและผลการทดลอง

4.1 การติดตั้ง

ก่อนที่จะนำใช้เอ็นจิ้นไปใช้งานนั้น มีสิ่งที่จำเป็นจะต้องติดตั้งก่อนหน้า ดังนี้

1) Visual Studio 2005 (เพื่อใช้ในการพัฒนาโปรแกรม)
2) DirectX SDK (ในกรณีที่ต้องการจะใช้ Dx9Renderer)

เพื่อความสะดวกที่จะทำการทดลองโดยไม่ต้องแก้ไข Source Code ใดๆ เลย จำเป็นจะต้องมีการสร้างตัวแปรระบบชื่อ PT1_PATH ขึ้นมาโดยให้มีค่าเป็นตำแหน่งที่ติดตั้งเอ็นจิ้นลงไป ดังรูป



รูปที่ 4.1 แสดงการสร้างตัวแปรระบบ PT1_PATH

ชิ้นงานจะประกอบไปด้วยส่วนต่างๆ ดังรูป

รูปที่ 4.2 แสดงส่วนประกอบต่างๆ ของชิ้นงาน

1) Data เป็นส่วนที่เก็บไฟล์ข้อมูลภายนอกที่นำมาใช้
2) SDK เป็นส่วนที่เก็บไฟล์ Include กับ Library ของเอ็นจิ้น
3) X – Library เป็นส่วนที่เก็บ Source Code ของเอ็นจิ้น
4) X – Sample เป็นส่วนที่เก็บ Source Code ของตัวอย่างการทดลองต่างๆ
5) X – Tool เป็นส่วนที่เก็บ Source Code ของเครื่องมือต่างๆ ที่ใช้กับเอ็นจิ้น

[หมายเหตุ: สำหรับการติดตั้งนั้น สามารถทำการคัดลอกองค์ประกอบทั้งหมดไปใช้งานได้เลย]

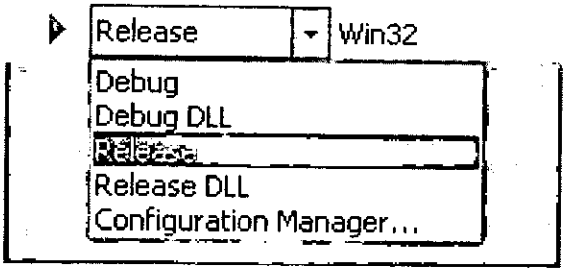
ในการที่จะสร้างเอ็นจิ้นขึ้นมา เราจะต้องคอมไพล์และสร้างส่วนต่างๆ ที่อยู่ภายใน X – Library ดังรูป

รูปที่ 4.3 แสดงส่วนประกอบต่างๆ ของ X-Library

โดยที่จะต้องสร้างตามลำดับต่อไปนี้

1) Foundation
2) Graphics
3) (Dx9 หรือ Wgl) Renderers
4) (Dx9 หรือ Wgl) Application

ทั้งนี้ ในแต่ละส่วน จะประกอบไปด้วยไฟล์ Project (*.vcproj) ที่มี Configuration ให้เลือก ดังรูป



รูปที่ 4.4 แสดง Configuration ของไฟล์ Project ใน X - Library

1) Debug สร้าง Static Library สำหรับตรวจสอบแก้ไข

2) Debug DLL สร้าง Dynamic Library สำหรับตรวจสอบแก้ไข

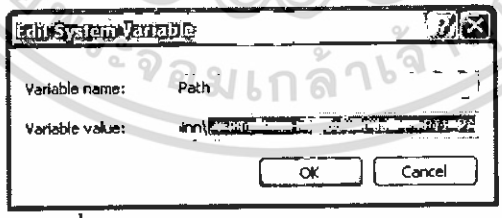
3) Release สร้าง Static Library สำหรับนำไปใช้งาน

4) Release DLL สร้าง Dynamic Library สำหรับนำไปใช้งาน

เนื่องจากในการนำ Dynamic Library ที่ได้ไปใช้งาน ไฟล์ *.dll จำเป็นที่จะต้องอยู่ในที่เดียวกับตำแหน่งที่กำลังทำงานอยู่หรือว่าอยู่ในตัวแปรระบบ PATH ดังนั้น ในกรณีที่ไม่ต้องการจะทำการคัดลอกไฟล์ *.dll ไปใช้งาน ก็สามารถปรับแต่ค่าตัวแปรระบบได้ ดังรูปต่อไปนี้

(1) (2)

รูปที่ 4.5 แสดงการสร้างตัวแปรระบบ PT1_PATH_DEBUG_DLL (1) และ PT1_PATH_RELEASE_DLL (2)

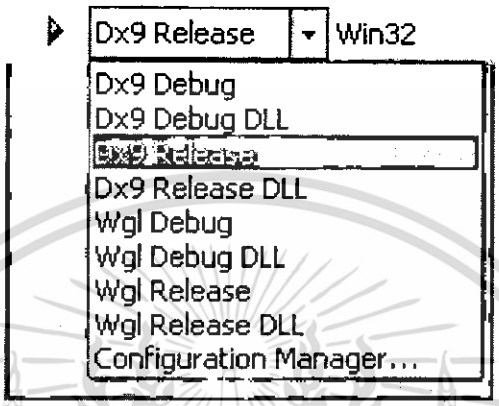


รูปที่ 4.6 แสดงการแก้ไขตัวแปร PATH

จากรูป 4.5 จะสร้างตัวแปร PT1_PATH_DEBUG_DLL และ PT1_PATH_RELEASE_DLL ขึ้นมา โดยระบุตำแหน่งของไฟล์ Dynamic Library ที่สร้างขึ้นมา ต่อจากนั้นในรูป 4.6 จะทำการแก้ไขค่า Path ให้รวมตัวแปรทั้ง 2 ที่กล่าวไว้ข้างต้นเข้าไปโดยต่อท้ายค่าเดิมที่มีอยู่ ดังนี้

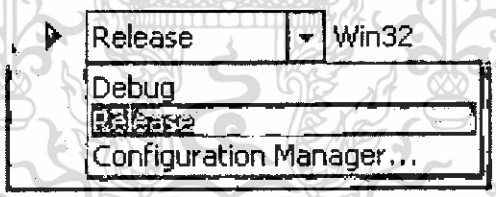
;%PT1_PATH_DEBUG_DLL%;%PT1_PATH_RELEASE_DLL%

ทั้งนี้ ในการที่จะสร้างตัวอย่างการทดลองขึ้นมา เราจะต้องคอมไพล์และสร้างส่วนต่างๆ ที่อยู่ภายใน X – Sample ซึ่งจะในแต่ละไฟล์ Project ของการทดลองจะมี Configuration ให้เลือก ดังนี้



รูปที่ 4.7 แสดง Configuration ของไฟล์ Project ใน X - Sample

ส่วนการที่จะสร้างเครื่องมือช่วยทำงานต่างๆ เราจะต้องคอมไพล์และสร้างส่วนต่างๆ ที่อยู่ภายใน X – Tool ซึ่งจะในแต่ละไฟล์ Project ของการทดลองจะมี Configuration ให้เลือก ดังนี้



รูปที่ 4.8 แสดง Configuration ของไฟล์ Project ใน X - Tool

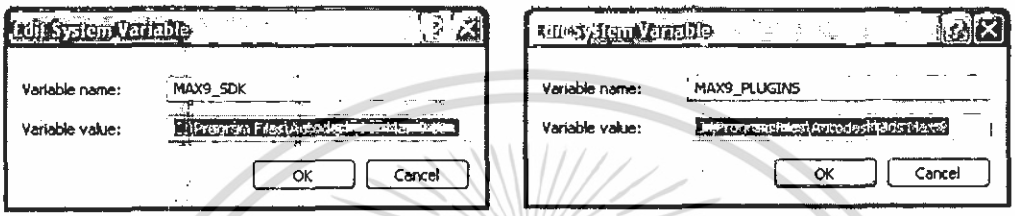
ทั้งนี้ ภายใน X – Tool จะมีอยู่ทั้งหมด 3 Project คือ

1) Max9ToPtl สำหรับสร้างไฟล์นามสกุล *.dle ซึ่งเป็น Plug-in ของโปรแกรม 3ds max เพื่อแปลงไฟล์นามสกุล *.max เป็น *.ptof
2) Bmp24ToPtif สำหรับสร้างไฟล์ Bmp24ToPtif.exe เป็น โปรแกรมประเภทคอนโซลเพื่อแปลงไฟล์นามสกุล *.bmp (24 บิต) เป็น *.ptif
3) PtifToBmp24 สำหรับสร้างไฟล์ PtifTo Bmp24.exe เป็น โปรแกรมประเภทคอนโซลเพื่อแปลงไฟล์นามสกุล *.ptif เป็น *.bmp (24 บิต)

ในการใช้งาน Bmp24ToPtif และ PtifToBmp24 จะเป็นดังนี้

- Bmp24ToPtif.exe ชื่อไฟล์นามสกุล *.bmp (24 บิต)
- PtifToBmp24.exe ชื่อไฟล์นามสกุล *.ptif

ส่วน Max9ToPtl เพื่อความสะดวกที่จะไม่ต้องแก้ไขไฟล์ Project จะต้องทำการสร้างตัวแปรระบบชื่อ MAX9_SDK และ MAX9_PLUGINS โดยให้มีค่าเป็นตำแหน่งที่ของ SDK และ Plug-In ของโปรแกรม 3ds max ดังรูป ต่อไปนี้



(1) (2)

รูปที่ 4.9 แสดงการสร้างตัวแปรระบบ MAX9_SDK (ซ้าย) และ MAX9_PLUGINS (ขวา)

4.2 คีย์พื้นฐานของทุกการทดลอง

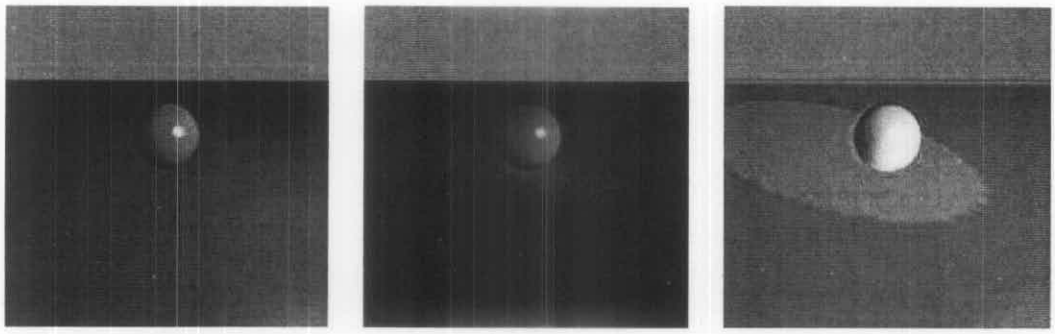
คีย์พื้นฐานของทุกการทดลองเป็นตามตารางต่อไปนี้

คีย์	ผลลัพธ์ที่เกิดขึ้นในการทดลอง
LEFT_ARROW	หมุนกล้องไปทางซ้าย
RIGHT_ARROW	หมุนกล้องไปทางขวา
UP_ARROW	เลื่อนกล้องไปข้างหน้า
DOWN_ARROW	เลื่อนกล้องไปข้างหลัง
PAGE_UP	เงยหรือหมุนกล้องขึ้นตามแนวตั้ง
PAGE_DOWN	เงยหรือหมุนกล้องลงตามแนวตั้ง
HOME	เลื่อนกล้องขึ้นตามแนวตั้ง
END	เลื่อนกล้องลงตามแนวตั้ง
F1	หมุนวัตถุตามเข็มนาฬิการอบ Direction Vector ของกล้อง (Right-Hand)
F2	หมุนวัตถุทวนเข็มนาฬิการอบ Direction Vector ของกล้อง (Right-Hand)
F3	หมุนวัตถุตามเข็มนาฬิการอบ Up Vector ของกล้อง (Right-Hand)
F4	หมุนวัตถุทวนเข็มนาฬิการอบ Up Vector ของกล้อง (Right-Hand)
F5	หมุนวัตถุตามเข็มนาฬิการอบ Right Vector ของกล้อง (Right-Hand)
F6	หมุนวัตถุทวนเข็มนาฬิการอบ Right Vector ของกล้อง (Right-Hand)
T	เพิ่มอัตราเร็วของการเคลื่อนที่ของกล้อง
t	ลดอัตราเร็วของการเคลื่อนที่ของกล้อง
R	เพิ่มอัตราเร็วของการหมุนของกล้อง
r	ลดอัตราเร็วของการหมุนของกล้อง
?	ทำการ reset ค่า Timer ของระบบ

ตารางที่ 4.1 แสดงคีย์พื้นฐานของทุกการทดลอง

4.3 การทดลอง Lighting

เป็นการแสดงผลการใช้งาน Light ประเภทต่างๆ โดยที่ในฉากจะประกอบไปด้วยพื้นราบกับวัตถุทรงกลม โดยให้มีจำนวนได้มากสุดทั้งหมด 8 Light



(1) (2) (3)

รูปที่ 4.10 แสดงผลของการทดลอง Lighting โดยที่

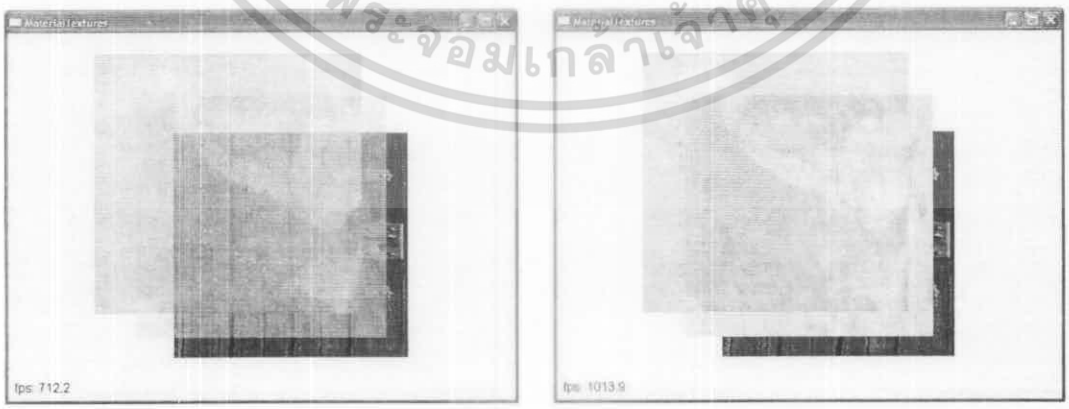
(1) มี 1 Ambient Light กับ 1 Point Light, (2) มี 1 Ambient Light กับ 2 Spot Light, และ

(3) มี ทั้ง Ambient Light, Directional Light, Point Light และ Spot Light อย่างละ 2

ในการทดลองนี้ จะสามารถที่จะป้อนคีย์ a เพื่อเพิ่ม Ambient Light, A เพื่อลด Ambient Light, d เพื่อเพิ่ม Directional Light, D เพื่อลด Directional Light, s เพื่อเพิ่ม Spot Light, S เพื่อลด Spot Light, p เพื่อเพิ่ม Point Light, และ P เพื่อลด Point Light

4.4 การทดลอง MaterialTextures

เป็นการผสม Material กับ Texture เข้าไว้ด้วยกัน แล้วทำการทดลองปรับเปลี่ยนค่า เพื่อดูผลลัพธ์ของ Alpha Blending โดยที่ในฉากจะประกอบไปด้วยสี่เหลี่ยมที่มี Texture ทั้ง 3 โดยที่สี่เหลี่ยมที่อยู่เหนือสุดจะมีค่าสี RGB ของ Texture และ Material ที่มีค่า Alpha น้อยกว่า 1 ทำให้วัตถุมีลักษณะโปร่งแสง ในขณะที่สี่เหลี่ยมที่อยู่ล่างสุดนั้นจะมีเพียง RGB Texture ซึ่งไม่มีค่า Alpha (จึงมีลักษณะที่ทึบ)



(1) (2)

รูปที่ 4.11 แสดงผลของการทดลองของ MaterialTextures โดยที่

(1) จะปรับค่าให้สี่เหลี่ยมกลางโปร่งแสง และ (2) จะปรับค่าให้สี่เหลี่ยมกลางทึบ

ในการทดลองนี้ จะสามารถที่จะป้อนคีย์ =,+ และ-,_ เพื่อเพิ่มและลดค่าของ Alpha (ความโปร่งแสง) ของสี่เหลี่ยมที่อยู่ตรงกลางได้

4.5 การทดลอง Multitextures

เป็นการทดสอบ Multiteturing โดยในฉากจะประกอบไปด้วยสี่เหลี่ยมที่มี 2 Texture Effect ติดอยู่ โดยที่ Effect แรกเป็นรูปวงกลมหน้ายิ้มหน้าเศร้า ส่วน Effect ที่ 2 เป็นรูปของแผงตารางสี่เหลี่ยม ซึ่ง Effect ทั้ง 2 จะถูก blend (ผสมกลมกลืน) เข้าด้วยกัน และถูกวาดในเวลาเดียวกัน

(1) (2)

รูปที่ 4.12 แสดงภาพเริ่มต้นในการทดลอง Multitextures
(1) แสดงภาพแรก ส่วน (2) แสดงภาพที่สอง

(1) (2) (3)

รูปที่ 4.13 แสดงผลการทดลองของ Multitextures โดยมีรูปแบบการ blend ในลักษณะต่างๆ คือ (1) ใช้ Multiplicative Blend, (2) ใช้ Hard-Additive Blend, และ (3) ใช้ Soft-Additive Blend

ในการทดลองนี้ จะสามารถที่จะป้อนคีย์ n,N เพื่อเปลี่ยนรูปแบบการ blend ดังรูปที่ 4.11 ทั้งนี้ รูปแบบการ blend ต่างๆ จะคำนวณจากสมการต่างๆ ดังต่อไปนี้

Multiplicative Blend

(Rf, Gf, Bf, Af) = (Rs*Rd, Gs*Gd, Bs*Bd, As*Ad)

Hard-Additive Blend

(Rf, Gf, Bf, Af) = (Rs+Rd, Gs+Gd, Bs+Bd, As+Ad)

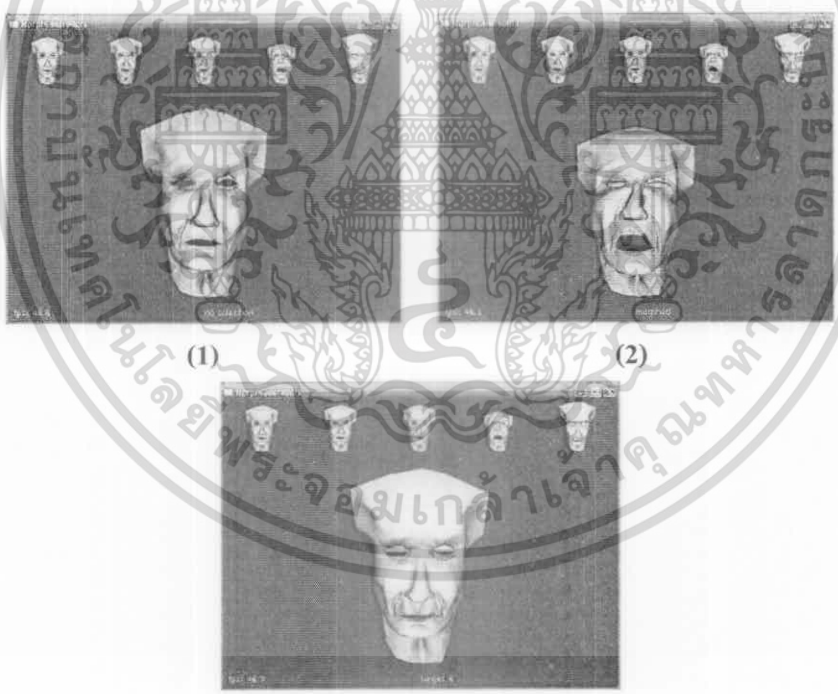
Soft-Additive Blend

(Rf, Gf, Bf, Af) = ((1-Rd)*Rs+Rd, (1-Gd)*Gs+Gd, (1-Bd)*Bs+Bd, (1-Ad)*As+Ad)

[หมายเหตุ: เมื่อ R, G, B, A คือค่า Red, Green, Blue, และ Alpha ส่วน f, s, d หมายถึง Final, Source และ Destination]

4.6 การทดลอง MorphController

เป็นการ morph Triangle Mesh โดยมีเป้าหมายอยู่ทั้งหมด 5 Mesh ซึ่งจะถูกแสดงอยู่ที่มุมบนของหน้าต่าง ส่วน Mesh ที่กำลังถูก morph นั้นจะแสดงอยู่กลางหน้าต่าง โดยจะทำการ morph จากวัตถุแรกบนมุมซ้ายบนไปถึงวัตถุหลังสุดบนมุมขวาบน วนซ้ำไปเรื่อยๆ



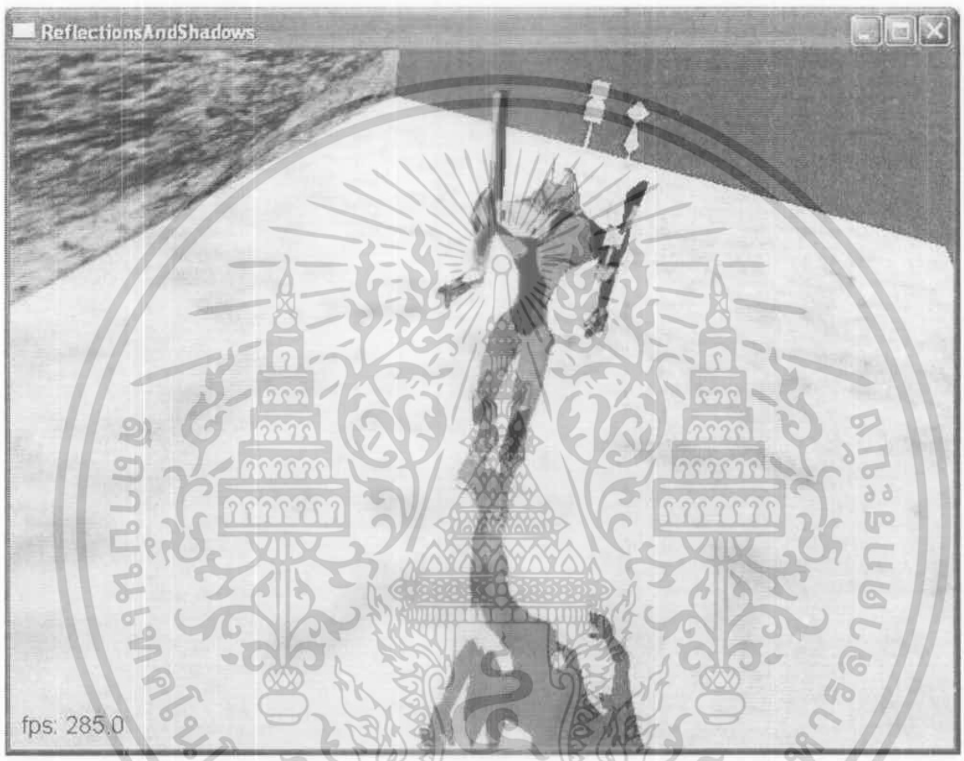
(1) (2)

(3)

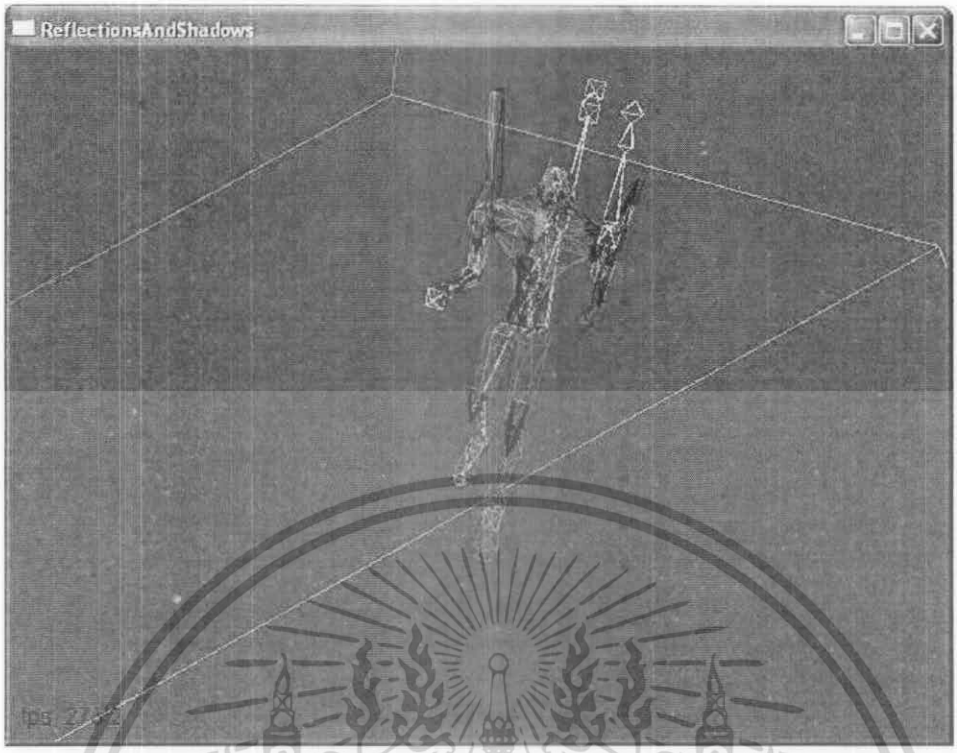
รูปที่ 4.14 แสดงผลการทดลองของ MorphController โดยที่ (1) อยู่ระหว่างวัตถุที่หนึ่งกับสอง, (2) อยู่ระหว่างวัตถุที่สามถึงห้า, และ (3) อยู่ระหว่างสี่วนกลับไปถึงหนึ่ง

4.7 การทดลอง ReflectionsAndShadows

เป็นการผสม Planar Reflection กับ Planar Shadow เข้าไว้ด้วยกัน ทั้งนี้ที่ Reflection จะเกิดขึ้นที่กำแพง (ด้านขวาบนของภาพ) ส่วน Shadow เกิดขึ้นที่พื้น โมเดลที่ใช้จะรองรับการทำงานของ Skeleton Animation ที่มี Bone (หรือ กระดูก) ช่วยในการควบคุมการเคลื่อนไหว ในการทดลองนี้ จะสามารถที่จะป้อนคีย์ i,l,k,j (รวมถึงตัวใหญ่) เพื่อขยับตัวโมเดล ไปยังทิศทางหน้า, หลัง, ซ้ายและ ขวา โดยสัมพัทธ์กับทิศทางของมุมกล้อง



(1)



(2)

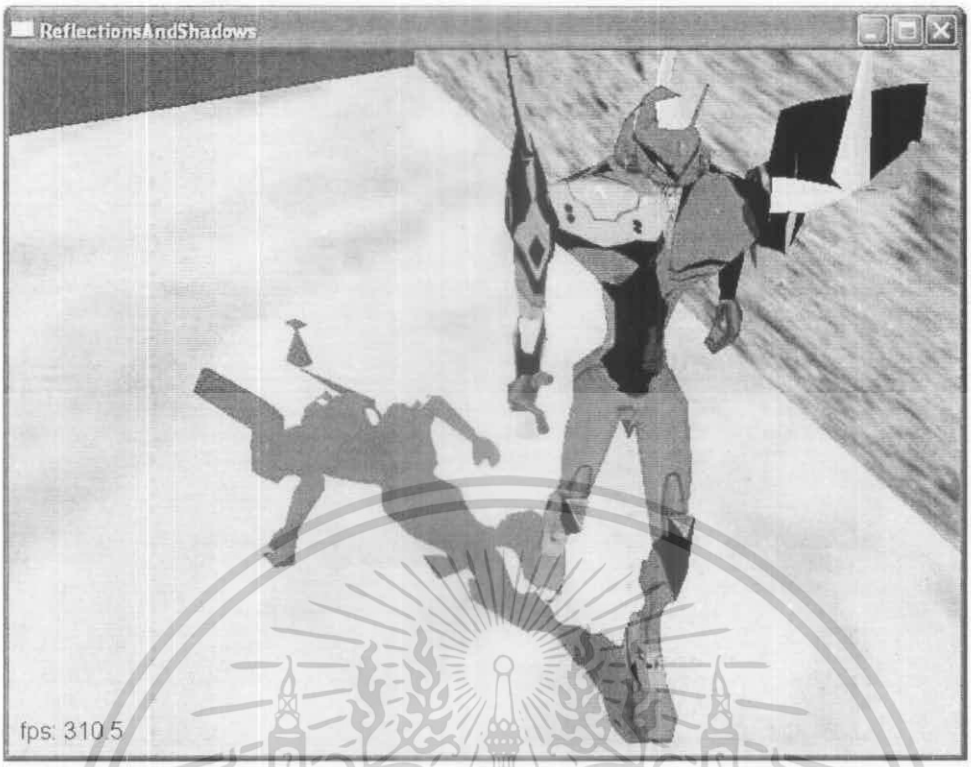
รูปที่ 4.15 แสดงผลการทดลองของ ReflectionsAndShadows โดยที่จะแสดงให้เห็นถึงการวางตัวของ Bone ในการทำ Skeleton Animation ของโมเดลตัวละครจากด้านหน้า ทั้งนี้ (1) จะเป็นการวาดแบบปกติ ในขณะที่ (2) จะเป็นการวาดเฉพาะเค้าโครง

(1)

(2)

รูปที่ 4.16 แสดงผลการทดลองของ ReflectionsAndShadows โดยที่จะแสดงให้เห็นถึงการวางตัวของ Bone ในการทำ Skeleton Animation ของโมเดลตัวละครจากด้านข้าง ทั้งนี้ (1) จะเป็นการวาดแบบปกติ ในขณะที่ (2) จะเป็นการวาดเฉพาะเค้าโครง

(1)



(2)

รูปที่ 4.17 แสดงผลการทดลองของ ReflectionsAndShadows โดยที่ (1) และ (2) แสดงให้เห็นถึง Effect จากโมเดลตัวละคร ที่เกิดขึ้นบนกำแพงและพื้น ตามลำดับ

รูปที่ 4.18 แสดงผลการทดลองของ ReflectionsAndShadows โดยที่จะแสดงถึงผลลัพธ์จากการที่ Bone ฝั่งซ้ายวางตำแหน่งผิด ซึ่งจะมีวงกลมสีแดงล้อมรอบอยู่

จากรูป 4.15 และ 4.16 จะพบว่าการวางตัวของ Bone ที่ใช้ควบคุม Skeleton Animation ทางฝั่งด้านซ้ายจะไม่ถูกต้อง ซึ่งคาดว่าน่าจะเนื่องมาจากการนำข้อมูลจากโดยผ่าน Exporter นั้นไม่สมบูรณ์ จากการทดลองเพิ่มเติม พบว่าจะเป็นกับวัตถุใดๆ ก็ตาม ที่ผ่านการใช้งานคำสั่ง Mirror ของ 3ds max

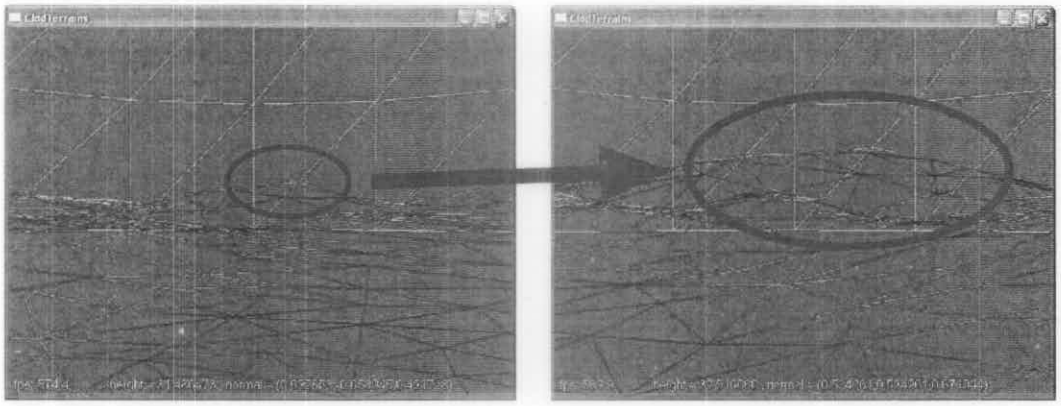
ทั้งนี้ จากรูปที่ 4.18 ในการทำ Skeleton Animation นั้น วัตถุของขาซ้ายและแขนซ้ายที่ได้รับอิทธิพลมาจาก Bone ที่วางตัวผิด จะเปลี่ยนตำแหน่งมากกว่าผลลัพธ์ที่ต้องการ เนื่องมาจากตำแหน่งของ Bone อยู่ไกลจากวัตถุมากกว่าที่ควรจะเป็น สำหรับแนวทางการแก้ไขนั้น คาดว่าจะต้องทำการศึกษาข้อมูลภายใน 3ds max SDK เพิ่มเติม ซึ่งน่าจะมีค่า Flag ที่แสดงถึงคำสั่ง Mirror เก็บไว้

4.8 การทดลอง ClodTerrains

เป็นการแสดงการทำงานระบบของ Terrain โดยใช้คลาส ClodTerrain ความสูงของกล้องจะขึ้นกับความสูงของ Terrain ที่ตำแหน่งปัจจุบัน ซึ่งจะถูกแสดงค่าไว้ที่ด้านล่างของหน้าต่าง นอกจากนั้น ค่า Normal Vector ณ ตำแหน่ง ก็จะถูกแสดงค่าด้วย เมื่อทำการเลื่อนกล้อง จะเห็นส่วนต่างๆ ของ Terrain ค่อยๆ โผล่ขึ้นมา ซึ่งจะเห็นถึงการเปลี่ยนแปลงของ Triangle Mesh ที่เกิดขึ้นอย่างต่อเนื่อง

ในการทดลองนี้ จะสามารถที่จะป้อนคีย์ + และ _ เพื่อปรับเพิ่มและลดส่วนต่างๆ ของ Terrain

(1) **(2)**



(3) (4)

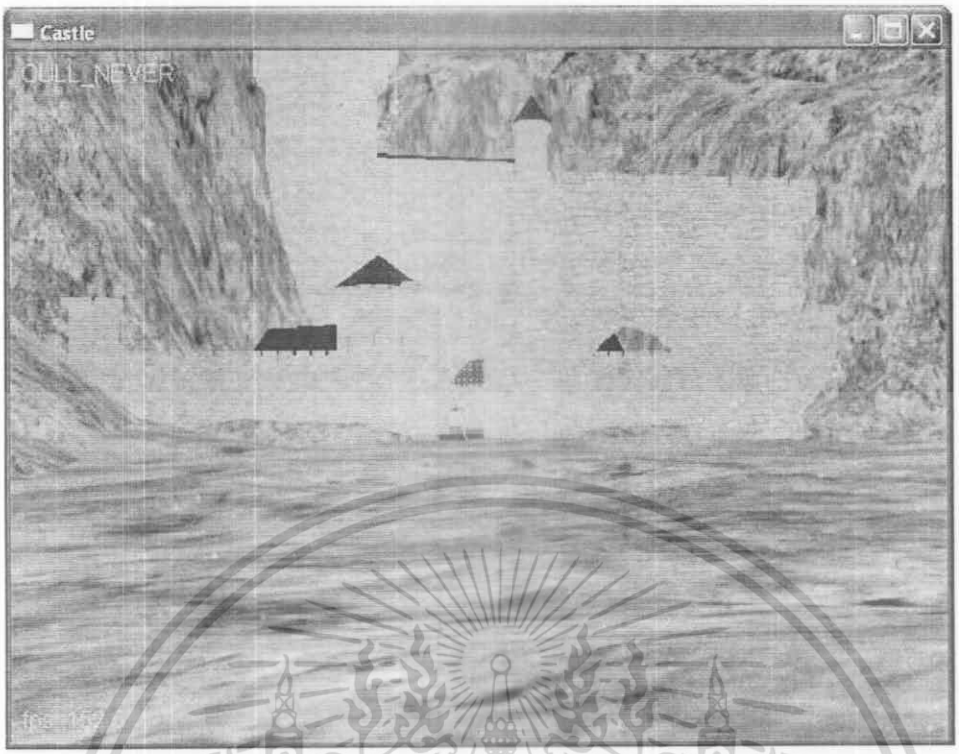
รูปที่ 4.19 แสดงผลการทดลองของ ClodTerrains โดยที่
(1) และ (3) แสดงพื้นสูงสีเทา เมื่อมองจากกล้องในระยะไกล โดยใช้การวาดแบบปกติ (1) และเส้นโครง (3)
(2) และ (4) แสดงพื้นสูงสีเทา เมื่อมองจากกล้องในระยะใกล้ โดยใช้การวาดแบบปกติ (2) และเส้นโครง (4)

เมื่อทำการทดลองโดยการเลื่อนกล้องไปยังตำแหน่งที่ใกล้ขึ้น ก็จะเห็นได้ว่ารายละเอียดของพื้นผิวของ Terrain นั้นเพิ่มขึ้นตามด้วย

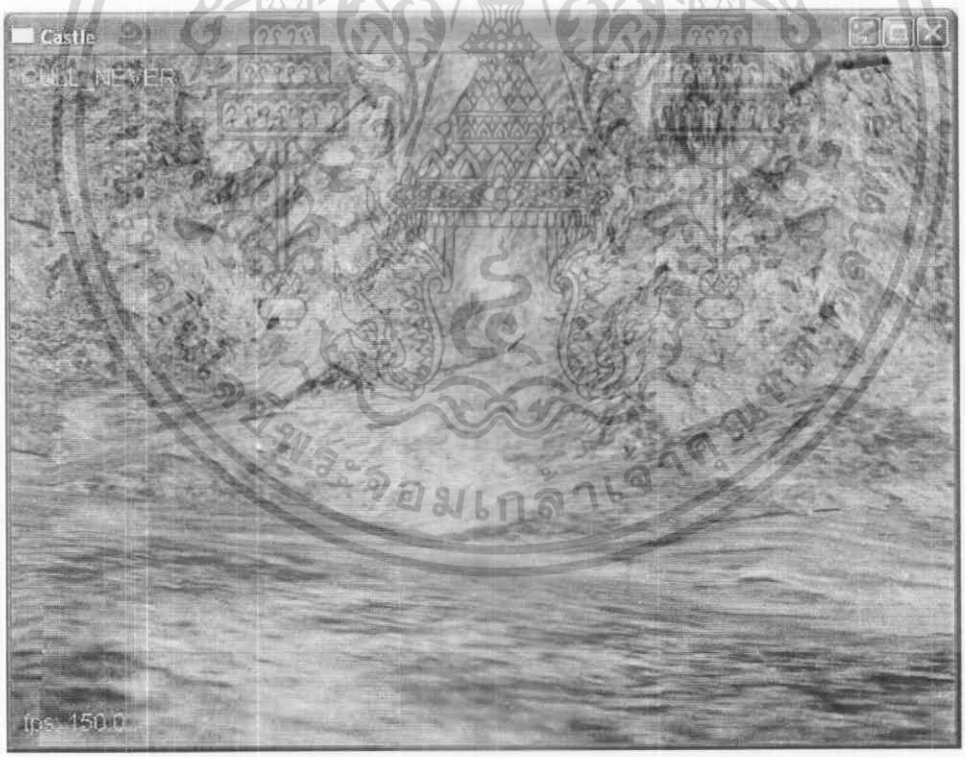
4.9 การทดลอง Castle

เป็นการทดสอบเปิด, ปิดการใช้งานของระบบ Culling เพื่อตรวจดูประสิทธิภาพที่เปลี่ยนแปลงไป โดยจะวัดได้จากค่า Frame Per Second (fps) ซึ่งแสดงอยู่บนด้านล่างซ้ายของหน้าต่าง

ในการทดลองนี้ จะสามารถที่จะป้อนคีย์ c,C เพื่อเปลี่ยน ประเภทของ Culling ของ Root ของทั้งฉาก คือ CULL_DYNAMIC (ประมวลผล) กับ CULL_NEVER (ไม่ประมวลผล) ซึ่งจะแสดงอยู่บนด้านบนซ้ายของหน้าต่าง ทั้งนี้ จากการทดลองจะได้ผลโดยคร่าว ดังรูปต่อไปนี้

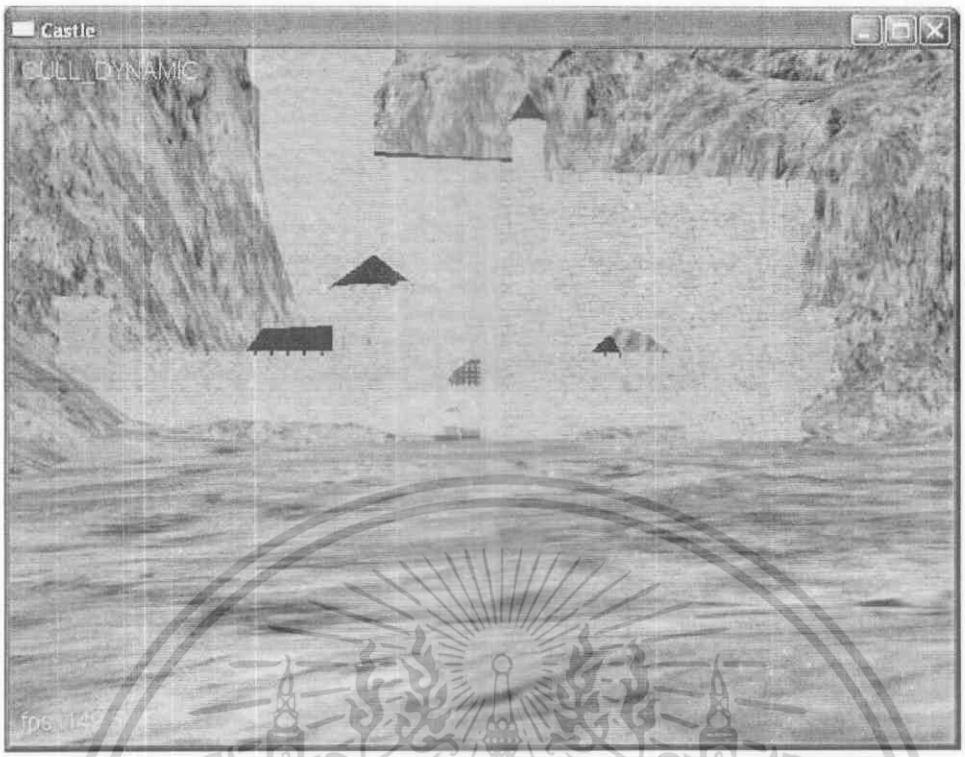


(1)

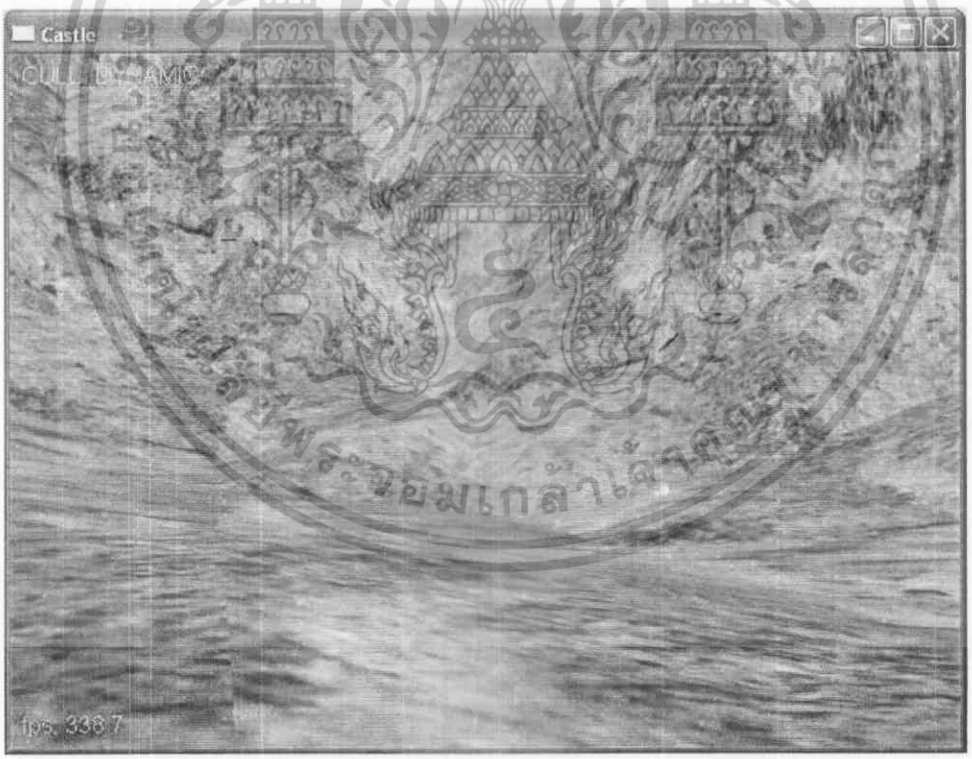


(2)

รูปที่ 4.20 แสดงผลการทดลองของ Castle โดยที่
จะใช้ CULL_NEVER ทั้งนี้ (1) คือ เมื่อมีวัตถุที่มีสามเหลี่ยมจำนวนมากอยู่ในมุมกล้อง
ส่วน (2) คือ เมื่อวัตถุที่มีสามเหลี่ยมจำนวนมากไม่ได้อยู่ในมุมกล้อง



(1)



(2)

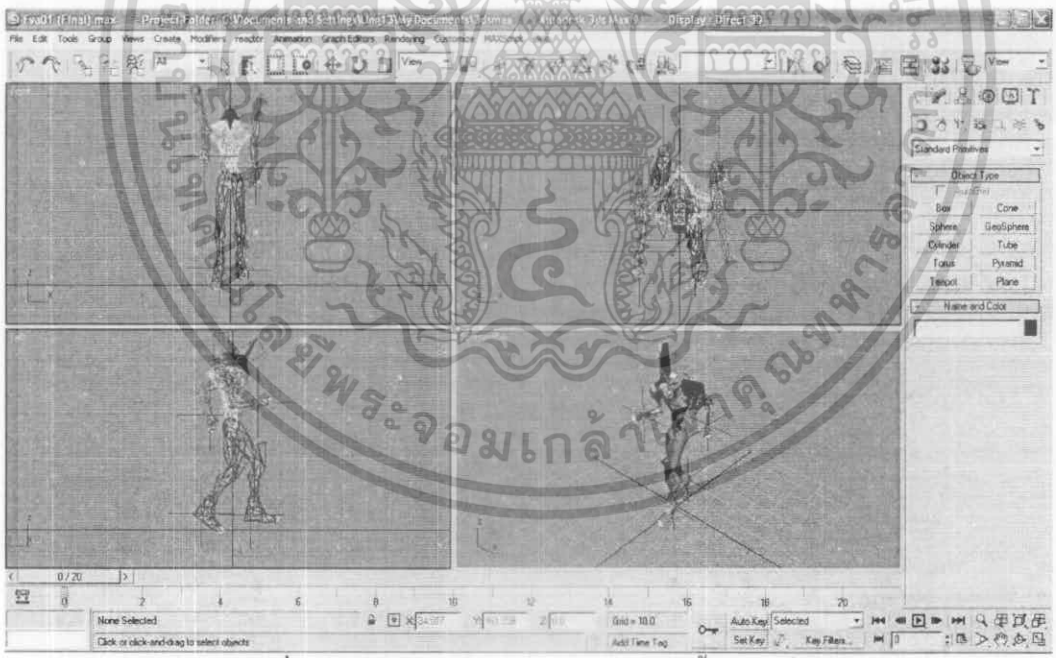
รูปที่ 4.21 แสดงผลการทดลองของ Castle โดยที่
จะใช้ CULL_DYNAMIC ทั้งนี้ (1) คือ เมื่อมีวัตถุที่มีสามเหลี่ยมจำนวนมากอยู่ในมุมกล้อง
ส่วน (2) คือ เมื่อวัตถุที่มีสามเหลี่ยมจำนวนมากไม่ได้อยู่ในมุมกล้อง

เมื่อทำการทดลองโดยใช้ Culling แบบ CULL_NEVER จะพบว่าไม่ว่าปราสาทหรือวัตถุที่มีสามเหลี่ยมจำนวนมากจะอยู่ในมุมกล้องหรือไม่ ประสิทธิภาพของการประมวลผลจะคงที่โดยค่า Frame Rate จะอยู่ประมาณ 140 – 180 fps อันเนื่องมาจากว่าไม่ได้ทำการตัดวัตถุนอกมุมกล้องออกเลย ในขณะที่เมื่อใช้ Culling แบบ CULL_DYNAMIC แล้วประสิทธิภาพของการประมวลผลจะเปลี่ยนแปลงไปเมื่อปราสาทอยู่นอกมุมกล้อง โดยที่ ค่า Frame Rate จะมากขึ้นจนไปคงที่ที่ประมาณ 330 – 370 fps แต่เมื่อปราสาทอยู่ในมุมกล้องแล้วจะได้ค่าผลลัพธ์ที่เหมือนกับ CULL_NEVER

4.10 การใช้งานไฟล์ที่ได้จาก Max9ToPt1 ในโปรแกรม 3ds max

หลังจากที่ทำการสร้างจากไฟล์ Project แล้ว จะได้ไฟล์ Plug-in นามสกุล *.dle ออกมา ทั้งนี้หากทำการตั้งค่าตัวแปรระบบดังที่ได้กล่าวไว้ในหัวข้อ 4.1 ไปแล้ว ไฟล์ผลลัพธ์จะไปอยู่ที่ตำแหน่งของตัวแปร MAX9_PLUGINS เมื่อทำการสร้างเสร็จ ทั้งนี้ ในการใช้งานนั้น จะมีขั้นตอนดังต่อไปนี้

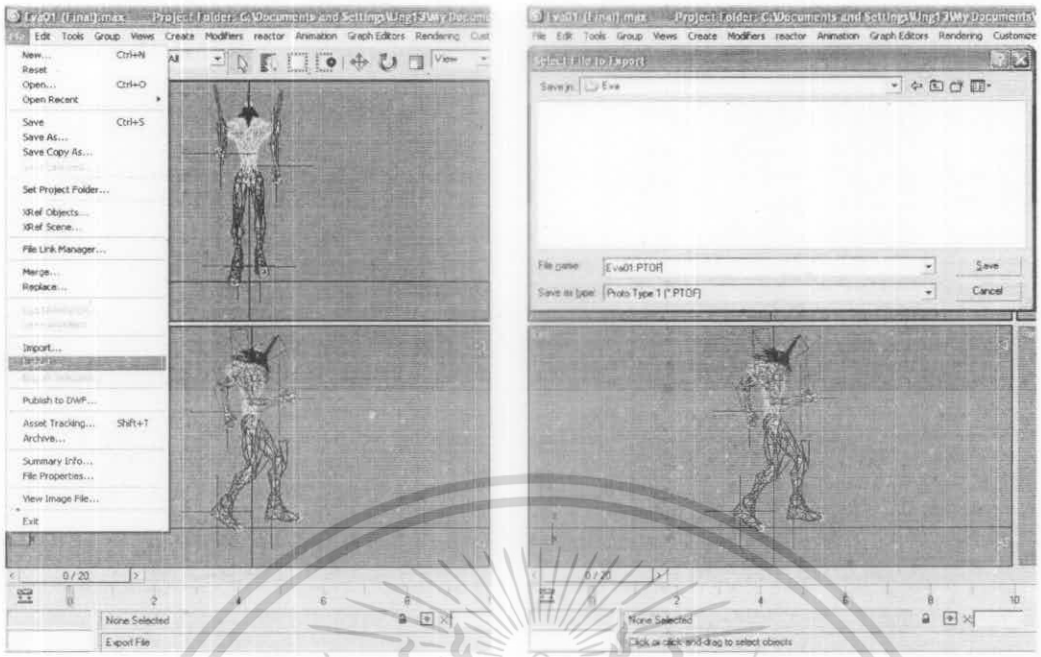
1) ทำการสร้างหรือเปิดไฟล์ใดๆ ขึ้นมาในโปรแกรม



รูปที่ 4.22 แสดงการใช้งาน Exporter ในขั้นตอนที่ 1

2) คลิกเมนู File ตามด้วย Export

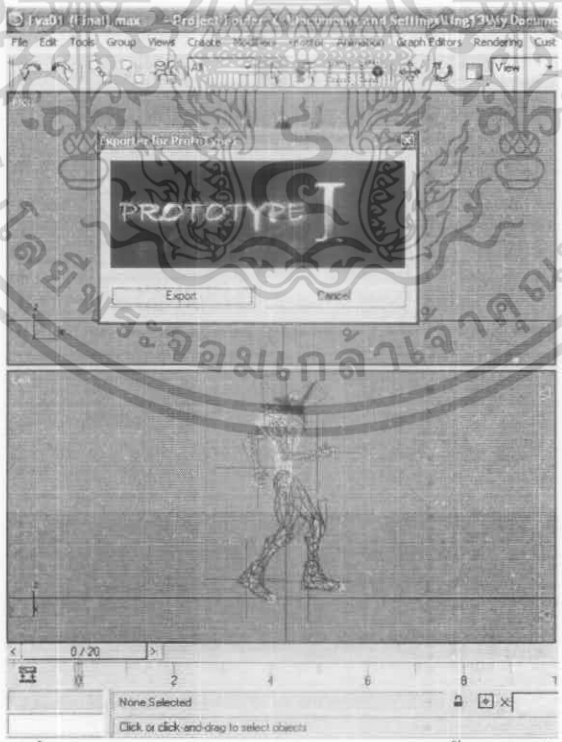
3) เลือกนามสกุลไฟล์ *.PTOF, ตำแหน่งที่จะบันทึก, และใส่ชื่อไฟล์



(1) (2)

รูปที่ 4.23 แสดงการใช้งาน Exporter ในขั้นตอนที่ 2 (1) และ 3 (2)

4) คลิ๊ก Export เพื่อดำเนินการ หรือ Cancel เพื่อยกเลิก



รูปที่ 4.24 แสดงการใช้งาน Exporter ในขั้นตอนที่ 4

บทที่ 5
บทวิจารณ์และสรุป

5.1 บทสรุป

ในปัจจุบันการสร้างแอพพลิเคชั่นประเภท 3 มิตินั้นจะต้องทำการเขียน Code เพื่อติดต่อกับฮาร์ดแวร์หลายส่วน รวมไปถึงการเขียน Code ที่เกี่ยวข้องกับการนำเข้าข้อมูลประเภทโมเดลและการจำลองการเคลื่อนที่ต่างๆ ทำให้ต้องใช้เวลาในการพัฒนานาน ดังนั้น กราฟฟิกส์เอ็นจิ้นที่พัฒนาขึ้นจะช่วยให้ผู้พัฒนาสามารถที่จะสร้างแอพพลิเคชั่นได้สะดวกและง่าย นอกจากนั้น ยังรวมไปถึงการตรวจสอบและแก้ไขอีกด้วย นอกจากนั้นเอ็นจิ้นจะมี Tool (เครื่องมือ) ต่างๆ สำหรับช่วยให้ผู้พัฒนาสามารถนำข้อมูลภายนอกเข้ามาใช้งานในระบบได้

5.2 วิจารณ์สิ่งที่ได้จากโครงงาน

จากชิ้นงานจะพบว่าตัวเอ็นจิ้นสามารถทำงานทั่วไปได้ตามที่ต้องการ แต่จากความรู้ของผู้พัฒนาชิ้นงานที่ไม่มากเพียงพอ ทำให้ยังไม่สามารถที่จะสร้างระบบดึงประสิทธิภาพออกมาได้ทุกส่วนครบถ้วน นอกจากนั้น ยังคงมีข้อบังคับในการใช้งานจริงอยู่หลายประการ จึงอาจทำให้ผู้ที่จะนำเอ็นจิ้นไปใช้งานจริงต้องทำการศึกษาเรียนรู้ระบบอยู่ ซึ่งอาจสร้างความรู้สึกว่ายุ่งยากให้กับผู้ใช้อยู่พอสมควร อย่างไรก็ตามโครงงานนี้ถือได้ว่าประสบความสำเร็จในระดับหนึ่ง โดยสามารถที่จะนำชิ้นงานไปสร้างแอพพลิเคชั่นเพื่อทดสอบจุดประสงค์ได้

5.3 ปัญหาอุปสรรคและแนวทางแก้ไข

5.3.1 ในการที่ผู้พัฒนาอื่นๆ จะนำส่วนต่างๆ ของเอ็นจิ้นไปใช้งานต่อนั้น ไม่สามารถทำได้สะดวกนัก เพราะจำเป็นที่จะต้องทำความเข้าใจและเรียนรู้ระบบพอสมควร

5.3.2 จากการที่ระบบ Scene Graph นั้น ใช้ Depth First Traverse ในการแสดงผล แม้ว่าในท้ายที่สุด Z-Buffer จะช่วยแก้ปัญหาของลำดับก่อนหลังบนฉากได้ก็ตาม แต่ในบางครั้งก็จำเป็นที่จะต้องเรียงลำดับวัตถุก่อนหลังอย่างแน่นอน เช่น การแสดงภาพวัตถุประเภทโปร่งแสง ซึ่งในที่นี้ผู้ใช้งานจำเป็นจะต้องมีความรู้ความเข้าใจมาก่อน จึงจะใช้งานได้อย่างถูกต้อง

5.3.3 แม้ว่าระบบจะสามารถตรวจสอบกลุ่มข้อมูลที่ส่งไปให้กับ VRAM โดยหวังเพื่อจะลดความจำเป็นในการส่งข้อมูลที่มีอยู่แล้ว แต่ระบบยังไม่ได้จัดกลุ่มของวัตถุที่ใช้ทรัพยากรร่วมกัน เพื่อจัดเรียงลำดับของการส่งข้อมูล ในจุดนี้จึงยังทำให้จำเป็นที่จะต้องเกิดการส่งข้อมูลที่แม้จะไม่ซ้ำกับที่อยู่บน VRAM แต่ก็ซ้ำกับที่เคยส่งไปแล้ว ทำให้เกิดงานที่ซ้ำ

ขึ้นมาโดยไม่จำเป็น ซึ่งสำหรับข้อมูลบางประเภทนั้นจะกินเวลาในการส่งมาก คือ ข้อมูลจำพวก Texture และ Shader

5.3.4 ในการพัฒนา Tool (เครื่องมือ) ที่ใช้เพื่อนำข้อมูลวัตถุ 3 มิติ จากโปรแกรมสร้างโมเดลเช่น 3ds max มาใช้งานนั้น มีจุดบกพร่องในเรื่องของขนาด ระบบหน่วย พิกัดแกน ที่ไม่เหมือนกัน ทำให้ไม่สามารถได้ผลลัพธ์ตามที่ต้องการ อีกทั้งในการดึงข้อมูลออกมานั้น ตัว Tool (เครื่องมือ) ที่พัฒนาขึ้น ยังไม่สามารถรองรับข้อมูลได้ครบถ้วน โดยยังคงเป็นข้อมูลในการแสดงผลปกติทั่วไปเท่านั้น

5.3.5 จากข้อก่อนหน้า ในการสร้างโมเดลใดๆ หากต้องการที่จะนำข้อมูลไปใช้ ให้เป็นโครงสร้างข้อมูลตามที่ระบบสามารถนำไปใช้งานได้อย่างมีประสิทธิภาพนั้น ผู้สร้างโมเดลจะต้องเข้าใจรูปแบบของ Scene Graph ด้วย เพื่อที่จะจัดวางวัตถุได้ตรงตามหลัก ซึ่งแตกต่างจากโครงสร้างโดยทั่วไปของโปรแกรมสร้างโมเดล (ในที่นี้คือ 3ds max)

5.3.6 ผู้พัฒนามีพื้นฐานความรู้ทางด้านคณิตศาสตร์ที่เกี่ยวข้องไม่มากนัก จึงต้องเสียเวลาไปกับการเรียนรู้และทำความเข้าใจ นอกจากนั้น บางข้อมูลหรือทฤษฎีจำเป็นที่จะต้องมีความรู้เฉพาะบางอย่างมาก่อน จึงทำให้หลายครั้งผู้พัฒนาได้นำมาใช้งานโดยที่ไม่ได้มีความเข้าใจแท้จริง

5.4 แนวทางพัฒนาต่อ

5.4.1 ทำการแก้ไขปัญหาที่ได้กล่าวไว้ข้างต้น รวมไปถึงการปรับแต่งชิ้นงาน เพื่อที่จะให้ใช้งานได้ง่ายและสะดวกสบายยิ่งขึ้น

5.4.2 ศึกษาถึงเทคนิคอื่นๆ ที่ผู้พัฒนาไม่มีเวลาในการศึกษาเข้ามาประกอบเพิ่มเติม เพื่อปรับปรุงระบบเดิมให้ได้ประสิทธิภาพยิ่งขึ้น

5.4.3 เพิ่มเติมส่วนองค์ประกอบอื่นๆ ที่มักเป็นที่นิยมใช้ควบคู่กับระบบ Graphic เช่น ระบบ Physic ต่างๆ

เอกสารอ้างอิง

[1] Frank D. Luna, 2003,**Introduction to 3D Game Programming with DirectX 9.0**, Wordware Publishing, Inc.

[2] Andre LaMothe, 2003, **Tricks of the 3D Game Programming Gurus: Advanced 3D Graphics And Rasterization**, Indianapolis, SAMS Publishing.

[3] Peter Walsh, 2003,**Advanced 3D Game Programming with DirectX 9.0**, Wordware Publishing, Inc.

[4] James C. Lelterman, 2002,**Learn Vertex Shader and Pixel Shader Programming with DirectX 9**, Wordware Publishing, Inc.

[5] Stefan Zerbst, 2005, **3D Game Engine Programming**, Thomson Course Technology, Premier Press.

[6] David H. Eberly, 2002, **3D Game Engine Design**, Morgan Kaufmann Technology, Elsevier.