เกมเอนจินสามมิติ

3D GAME ENGINE

นายขจรศักดิ์ สิริวัฒนานุกูล
นายคณิต เมฆฤทธิไกร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
พ.ศ.2548

เกมเอนจินสามมิติ

3D GAME ENGINE

โดย

นายชจรศักดิ์ สิริวัฒนานุกูล

นายคณิต เมฆฤทธิไกร

อาจารย์ที่ปรึกษา

ดร. สมศักดิ์ วลัยรัชต์

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

พ.ศ.2548

ปริญญานิพนธ์ปีการศึกษา 2548

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เกมเอนจินสามมิติ

3D GAME ENGINE

ผู้จัดทำ

1. นายขจรศักดิ์ สิริวัฒนานุกูล รหัสนักศึกษา 45010073

2. นายคณิต เมฆฤทธิไกร รหัสนักศึกษา 45010081

_____ อาจารย์ที่ปรึกษา

(ดร. สมศักดิ์ วลัยรัชต์)

เกมเอนจินสามมิติ

นาย ขจรศักดิ์ สิริวัฒนานุกูล 45010073
นาย คณิต เมฆฤทธิไกร 45010081
ดร. สมศักดิ์ วลัยรัชต์ อาจารย์ที่ปรึกษา
ปีการศึกษา 2548

บทคัดย่อ

ปริญญานิพนธ์นี้เกี่ยวกับการพัฒนาเกมเอนจินสามมิติ ซึ่งสามารถนำไปใช้พัฒนาเกม หรือโปรแกรมที่ใช้กราฟิกสามมิติได้ โดยเนื้อหาจะประกอบไปด้วยส่วนประกอบต่างๆของเกมเอนจิน หลักการในการออกแบบเพื่อให้เกมเอนจินสามารถนำไปใช้ได้อย่างไม่ยากนักแต่ยังคงประสิทธิภาพและความยืดหยุ่นในการทำงานอยู่ และสามารถปรับปรุงแก้ไขเปลี่ยนแปลงได้ในอนาคตเมื่อมีเทคโนโลยีใหม่ๆเพิ่มเข้ามา เกมเอนจินที่ได้พัฒนาขึ้นจะมีความสามารถในการทำ Hierarchical Scene Management (Scene graph) การแสดงภาพเคลื่อนไหวด้วยเทคนิคต่างๆ การแสดงภาพภูมิประเทศ การจำลองการเคลื่อนที่ของวัตถุแข็งคงรูป และการตรวจจับการชนกันของวัตถุด้วยวิธีต่างๆ ตลอดจนถึงเทคนิคพิเศษต่างๆเพื่อเพิ่มความสมจริงทางด้านภาพมากขึ้น และยังรวมไปถึงส่วนที่ช่วยอำนวยความสะดวกในการพัฒนาเกมด้านต่างๆ เช่นส่วนจัดการกับอินพุท ส่วนจัดการด้านเสียง ส่วนจัดการด้านการติดต่อสื่อสารผ่านระบบเครือข่ายอย่างง่ายๆ โครงการนี้ใช้ภาษา C++ ในการพัฒนา ร่วมกับ API ชนิดต่างๆ เช่น Direct Graphic Direct Sound Direct Input เป็นต้น แต่ทั้งนี้การออกแบบจะแสดงให้เห็นว่าการทำงานของส่วนหลักนั้นจะไม่ขึ้นกับ API ที่ใช้แต่อย่างใด

3D GAME ENGINE

Kajornsak Siriwattananukul	45010073
Kanit Mekritthikrai	45010081
Somsak Walairacht	Advisor
Academic Year 2004	

ABSTRACT

This thesis is about 3D game engine development which can be used as a game or 3D graphic application development toolkit. The thesis will cover each component of the game engine, the design pattern of game engine. The design is easy to use, high performance and flexible. This engine will be able to do hierarchical scene management, scene graph, animation technique, terrain rendering, rigid body simulation, collision detection and many effects related. It's also included the other part such as sound module. This engine will use C++ as the main language with many APIs. However the design is API independent.

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

นายชจรศักดิ์ สิริวัฒนานุกูล

นายคณิต เมฆฤทธิไกร

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

สารบัญรูป

สารบัญรูป (ต่อ)

รูปที่ หน้า

สารบัญรูป (ต่อ)

รูปที่ หน้า

บทที่ 1
บทนำ

1.1. บทนำ

ปัจจุบันนี้ คอมพิวเตอร์ได้มีส่วนในการให้ความบันเทิงกับผู้ใช้มากขึ้น เกมคอมพิวเตอร์ในปัจจุบันจึงมีความสามารถด้านต่างๆมากขึ้น ไม่ว่าจะเป็นด้านภาพ เสียง และปฏิสัมพันธ์กับผู้เล่น การพัฒนาเกมให้มีประสิทธิภาพจึงทำได้ลำบากมากขึ้นหากผู้พัฒนาไม่มีความรู้ทางด้านต่างๆที่เกี่ยวข้องเพียงพอ จะทำให้การพัฒนาเกมเป็นไปได้อย่างยากลำบาก และใช้ทรัพยากรต่างๆที่มีได้ไม่เต็มความสามารถ ดังนั้นเกมส์เอนจินจึงถูกพัฒนาขึ้นเพื่อให้ผู้พัฒนาเกมส์ใช้งานได้ง่ายขึ้นในการสร้างเกมส์ ทำให้ผู้พัฒนาสามารถดึงความสามารถออกมาได้สูงสุดตามความสามารถของเอนจินที่ใช้

1.2. วัตถุประสงค์ของโครงงาน

1.2.1. พัฒนา 3D Game Engine ที่สามารถนำไปพัฒนาเกมสามมิติ

1.2.2. ใช้เป็นเครื่องมือในการพัฒนาโปรแกรมประยุกต์ทางด้านกราฟิกสามมิติได้

1.2.3. ใช้เป็นเครื่องมือในการจำลองการเคลื่อนที่ของวัตถุแข็งคงรูปได้

1.3. ขอบเขตของโครงงาน

ขอบเขตของโครงงานนี้ได้แบ่งตามส่วนประกอบต่างๆของเกมเอนจินดังต่อไปนี้

1.3.1 Mathematical Module

ใช้สำหรับทำการคำนวณทางคณิตศาสตร์ต่างๆ ที่เกี่ยวข้อง เช่น Vector, Matrix และ Quaternion เป็นต้น ส่วนนี้จะเป็นส่วนที่ถูกใช้บ่อยที่สุดในส่วนประกอบทั้งหมดของเกมเอนจิน ดังนั้นในการพัฒนาจึงถูกให้ความสำคัญทางด้านความเร็วในการดำเนินงานมากกว่าส่วนอื่นๆ โดยได้นำข้อดีของเทคโนโลยี SIMD (ในที่นี้คือ Intel SSE) มาใช้เพื่อช่วยเพิ่มความเร็วในการประมวลผลในส่วนของฟังก์ชั่นการทำงานที่เห็นว่าการใช้คำสั่งประเภท SIMD จะให้ความเร็วในการทำงานที่มากกว่าการใช้คำสั่งพื้นฐานของ FPU เช่น Matrix multiplication, Matrix inversion และ Vector crossing เป็นต้น

1.3.2. Graphics Module

ใช้สำหรับเพิ่มความสะดวกในการติดต่อระหว่างโปรแกรมเมอร์และ Graphic API และมีความยืดหยุ่นโดยไม่ขึ้นกับชนิดของ Graphic API โดยอาศัยหลัก Polymorphism ทำการสร้างแม่แบบของคลาสอุปกรณ์แสดงผล (Abstract) ซึ่งถูกติดต่อโดย Graphic API ชนิดต่างๆ ขึ้น คลาสแม่แบบจะเป็นผู้กำหนดหน้าที่การทำงานหลักของอุปกรณ์แสดงผล

1.3.3. Supporting Module

ใช้สำหรับเพิ่มประสิทธิภาพ และความสามารถในการแสดงผลรูปแบบต่างๆของเอนจิน และGraphic module เช่นการทำHierarchical scene management การเลือกวัตถุ (Picking) การทำ Visibility testing การแสดงภาพเคลื่อนไหว (Animation) และการแสดงภาพภูมิประเทศ (Terrain rendering) เป็นต้น โดยจะกล่าวถึงรายละเอียดและวิธีการในรายงานส่วนต่อไป

1.3.4. Physically Base Simulation Module

สามารถทำการจำลองการเคลื่อนไหวของวัตถุชนิดแข็งไม่ยืดหยุ่นให้ใกล้เคียงความเป็นจริงมากที่สุดโดยอาศัยหลักทฤษฎีทางฟิสิกส์เกี่ยวกับจลพลศาสตร์มาช่วยในการจำลอง โดยประกอบไปด้วยส่วนของการตรวจจับการชนกันของวัตถุ (Collision detection) และการจำลองการเคลื่อนที่ของวัตถุแข็ง (Rigid body simulation) โดยจะกล่าวถึงรายละเอียดและวิธีการในส่วนต่อไป

1.3.5. Sound Module

ใช้สำหรับเพิ่มความสะดวกในการสร้างเสียงประกอบแบบ 3 มิติ โดยจะใช้หลักการเดียวกันกับส่วนของ Graphic module ในการติดต่อกับ API ที่จัดการเกี่ยวกับด้านเสียง โดยมีการกำหนดตำแหน่งของแหล่งกำเนิดเสียง และตำแหน่งของผู้ฟัง เพื่อให้สามารถจำลองทิศทางการได้ยินเสียงได้อย่างถูกต้อง

1.4.ประโยชน์ที่คาดว่าจะได้รับ

1.4.1. ได้รับความรู้ ความเข้าใจเกี่ยวกับการแสดงภาพสามมิติในคอมพิวเตอร์

1.4.2. ได้รับความรู้ ความเข้าใจเกี่ยวกับกระบวนการออกแบบเกมส์เอนจินสามมิติ

1.4.3. ได้รับความรู้ ความเข้าใจเกี่ยวกับการเคลื่อนที่และการตรวจจับการชนของวัตถุ

1.4.4. สามารถสร้างวัตถุสามมิติจากเอนจินได้

1.4.5. สามารถจำลองการเคลื่อนที่ของวัตถุแข็งไม่เปลี่ยนสภาพได้ใกล้เคียงความเป็นจริง

1.5.ส่วนประกอบของปริญญานิพนธ์

เนื้อหาของปริญญานิพนธ์ฉบับนี้ประกอบไปด้วย 5 บทด้วยกันก็คือ บทนำ, ทฤษฎีที่เกี่ยวข้อง ,การออกแบบ ,การทดลอง และ บทสรุป โดยสามารถจำแนกรายละเอียดได้ตามนี้

บทที่ 1 บทนำ กล่าวถึงจุดประสงค์ ประโยชน์ที่ได้รับ และขอบเขตของโครงงาน

บทที่ 2 ทฤษฎีที่เกี่ยวข้อง กล่าวถึง ทฤษฎีโดยคร่าวๆ ที่นำมาใช้กับโครงงาน

บทที่ 3 การออกแบบ กล่าวถึง การออกแบบคลาสที่นำไป สร้างเป็น เอนจินสามมิติ

บทที่ 4 การทดลอง กล่าวถึง การทดสอบทฤษฎีว่าถูกต้องตามที่ต้องการหรือไม่

บทที่ 5 บทวิจารณ์และบทสรุป กล่าวถึง บทสรุป ปัญหาอุปสรรค และแนวทางแก้ไข แนวทางการพัฒนาต่อ

บทที่ 2
ความรู้เบื้องต้นเกี่ยวกับ 3D Game Engine

ในหัวข้อนี้จะกล่าวถึงทฤษฎีพื้นฐานโดยทั่วไป และที่เกี่ยวข้องในการพัฒนา 3D Game Engine จำทำให้ผู้อ่านทราบถึงประโยชน์ และสามารถเข้าใจในเนื้อหาในส่วนของบทที่ 4 ได้อย่างดียิ่งขึ้น ซึ่งทฤษฎีต่างๆในบทนี้จะไม่ได้กล่าวถึงอย่างละเอียดมากนัก แต่มากพอที่จะช่วยให้บรรลุวัตถุประสงค์ข้างต้นได้

2.1 ส่วนประกอบหลักของ 3D Game Engine

Game Engine เป็นเครื่องมือที่สามารถช่วยให้ผู้พัฒนาเกม สามารถมุ่งเน้นไปที่การพัฒนาเกมได้มากกว่า การพัฒนาส่วนประกอบอื่นๆให้สามารถทำงานได้ ส่วนประกอบเหล่านั้นได้แก่ ส่วนของ Graphics ซึ่งจะเกี่ยวข้องโดยตรงกับการใช้งาน Graphics API ต่างๆ ส่วนของ Physics ซึ่งทำการจำลองการเคลื่อนที่และการชนกันของวัตถุโดยใช้วิธีต่างๆ ส่วนของ Sound ซึ่งใช้ในการเล่นเสียงประกอบในเกม

ใน 3D Game Engine นั้นควรประกอบไปด้วยส่วนที่จำเป็นดังต่อไปนี้

2.1.1 Mathematical Module

การพัฒนา 3D Game Engine ที่ไม่ขึ้นตรงกับ API ใดนั้น จำเป็นต้องมีส่วนของ Mathematical Module เป็นของตนเอง เนื่องจากไม่สามารถที่จะเรียกใช้เครื่องมือช่วยเหลือจาก API ใดได้อย่างเฉพาะเจาะจง ซึ่งในส่วนนี้จะต้องสามารถช่วยให้การคำนวณทางคณิตศาสตร์ที่เกี่ยวข้อง เป็นไปได้อย่างถูกต้อง สะดวก แม่นยำ และรวดเร็ว เนื่องจากส่วนนี้จะถูกเรียกใช้งานมากและบ่อยที่สุดเมื่อเทียบกับส่วนประกอบอื่น ความเร็วที่ใช้ในการทำงานจึงเป็นสิ่งสำคัญ โดยทั่วไปแล้วการทำ Profile เพื่อช่วยในการ Optimization จะแสดงให้เห็นว่าส่วนนี้จะส่งผลกับประสิทธิภาพโดยรวมของระบบมากที่สุด ส่วนนี้จึงต้องใช้ความระมัดระวังในการออกแบบและเขียนโปรแกรมอย่างมาก เพื่อให้เป็นไปตามจุดมุ่งหมายที่วางไว้ หัวข้อการคำนวณทางคณิตศาสตร์ที่สำคัญได้แก่ Vector, Matrix และ Quaternion เป็นต้น

2.1.2 Graphics Module

Module นี้ถือว่าเป็นส่วนที่สำคัญที่สุดส่วนหนึ่งในส่วนประกอบทั้งหมด เนื่องจาก เป็นส่วนแสดงผลภาพ นอกจากจะมีหน้าที่ติดต่อกับ Graphics API โดยตรงแล้ว ยังสามารถมีส่วนขยายเพื่ออำนวยความสะดวกให้กับผู้ใช้ หรือเพื่อให้มีลูกเล่นที่สวยงามน่าชมยิ่งขึ้นโดยที่ผู้ใช้ไม่ต้องมีความรู้ทางทฤษฎีเกี่ยวกับลูกเล่นเหล่านั้นทั้งหมด เพียงแค่เข้าใจในหลักการง่ายๆ ก็สามารถ

ใช้งานได้ นอกจากนี้อาจจะมีส่วนของ Scene Management ที่ช่วยให้การแสดงผลเป็นไปอย่างมีประสิทธิภาพมากขึ้น

2.1.3 Physically Base Simulation Module

สามารถทำการจำลองการเคลื่อนไหวของวัตถุชนิดแข็งไม่ยืดหยุ่นให้ใกล้เคียงความเป็นจริงมากที่สุดโดยอาศัยหลักทฤษฎีทางฟิสิกส์เกี่ยวกับจลพลศาสตร์มาช่วยในการจำลอง โดยประกอบไปด้วยส่วนของการตรวจจับการชนกันของวัตถุ (Collision detection) การตรวจหาปฏิสัมพันธ์อย่างง่ายหลังการชนของวัตถุแข็ง (Collision response of Rigid body) และ การจำลองการเคลื่อนที่ของวัตถุแข็ง (Rigid body simulation) โดยจะกล่าวถึงรายละเอียดและวิธีการในส่วนต่อไป

2.1.4 Sound Module

ส่วนนี้ใช้สำหรับ เล่นไฟล์เสียงที่ต้องการ และอาจใช้ควบคุมระดับและทิศทางของเสียงที่เล่นได้ โดยส่วนมาก API ที่ติดต่อจะมีความสามารถค่อนข้างสูง และสามารถใช้งานได้โดยง่าย ส่วนนี้จึงไม่มีความสลับซับซ้อนใดๆ เพียงแค่สามารถเรียกใช้ API ที่ต้องการได้เท่านั้น โดยส่วนที่อาจจะเพิ่มเติมขึ้นมาคือส่วนของการควบคุมการใช้ทรัพยากรเท่านั้น

2.2 ลักษณะของ Graphics Module ใน Game Engine ที่มีอยู่ในปัจจุบัน

ในปัจจุบันส่วนของ Graphics Module ใน Game Engine ชั้นนำได้ถูกพัฒนาไปอย่างรวดเร็ว ทั้งนี้เนื่องจากความสวยงามของภาพสามารถนำไปใช้เป็นจุดเด่นทางการค้าได้ดี ในหัวข้อนี้จะแสดงถึงคุณสมบัติเด่นๆ ที่จำเป็น

ตารางต่อไปนี้แสดงคุณสมบัติทางด้านการแสดงผลและส่วนที่เกี่ยวข้องของ Source Engine ของ บริษัท Value ที่ใช้ในเกม Half-life 2 เทียบกับความสามารถของ PC Engine (ชื่อโครงงานนี้) ซึ่งมีเฉพาะความสามารถที่สำคัญพื้นฐานของ Game Engine ในปัจจุบัน

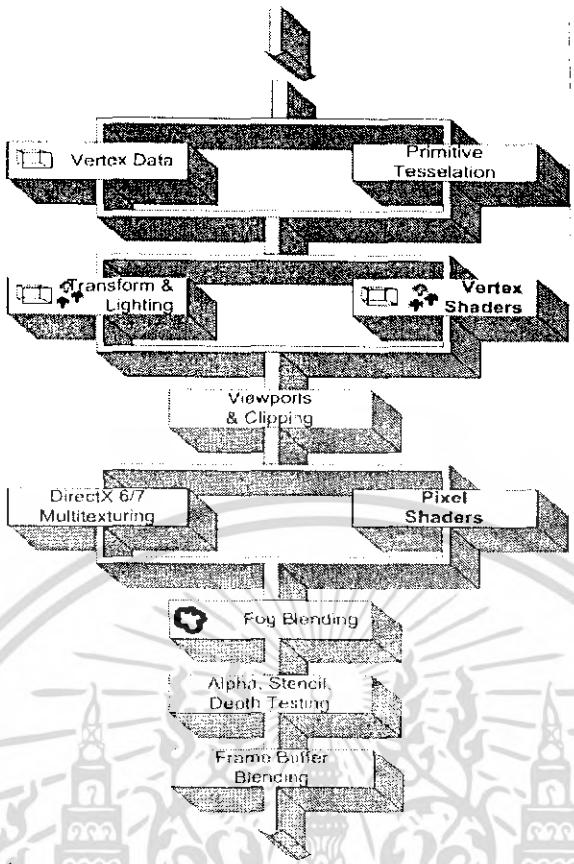
Feature	Source Engine	PC Engine
Lighting	Per-vertex, Per-pixel, Light mapping, Radiosity, Gloss maps:	Per-vertex, Light mapping
Shadow	Shadow Mapping:	-
Texturing	Basic, Multi-texturing, Bumpmapping, Mipmapping:	Basic, Multi-texturing, Bumpmapping
Shaders	Vertex, Pixel, High Level:	Vertex, Pixel

Scene Management	BSP, Portals, Occlusion Culling, LOD:	Scene graph (Basic Frustum Culling)
Animation	Skeletal Animation, Morphing, Facial Animation, Animation Blending:	Skeletal Animation
Meshes	Mesh Loading, Skinning, Progressive, Deformation:	Mesh Loading, Skinning
Special Effects	Environment Mapping, Billboarding, Particle System:	Billboarding
Terrain	Rendering, CLOD:	Rendering, CLOD

ตารางที่ 2.1 เปรียบเทียบความสามารถระหว่าง Source Engine กับ PC Engine [ข้อมูลอ้างอิงจาก 3D Game Engine Database – http://www.devmaster.net/engines]

2.3 แนะนำกระบวนการแสดงภาพ 3 มิติ

ในปัจจุบันเทคโนโลยีใหม่ๆของการ์ดแสดงผลอย่าง Vertex/Pixel Shader ได้เปิดโอกาสให้เรากำหนดกระบวนการการแสดงภาพสามมิติได้ด้วยตนเอง (Programmable Pipeline) ในบางขั้นตอนของกระบวนการ ดังนั้นผู้ใช้ Engine ควรจะเข้าใจถึง 3D Pipeline ในปัจจุบัน และมีความรู้พื้นฐานเกี่ยวกับเรื่อง Computer Graphics บ้าง ในส่วนนี้จึงขออนุญาตนำเสนอความรู้เบื้องต้นเหล่านี้



รูปที่ 2.1 3D Graphic Pipeline (With Vertex/Pixel Shader)

จากรูปที่ 2.1 จะเห็นได้ว่าในช่วงของกระบวนการ Transform & Lighting และTexturing นั้น จะเปิดโอกาสให้ผู้ใช้สามารถทำการ Program เองได้ด้วย Vertex/Pixel Shader ดังนั้นจึงต้องเข้าใจเกี่ยวกับส่วนประกอบของ 3D Pipeline เพื่อที่จะสามารถใช้งานร่วมกันกับขั้นตอนอื่นๆได้อย่างถูกต้อง ส่วนเรื่องเกี่ยวกับ Vertex/Pixel Shader จะกล่าวถึงในส่วนถัดไป

2.3.1 Vertex Data

ข้อมูล Vertex จัดเป็นข้อมูลที่เป็น Input ให้กับ Graphic Pipeline องค์ประกอบของข้อมูลอาจจะแตกต่างกันออกไปตามลักษณะการใช้งาน อย่างไรก็ตามองค์ประกอบของข้อมูล Vertex ที่ขาดไม่ได้คือ Vertex Position หรือตำแหน่งของ Vertex นั่นเอง โดยข้อมูล Position นั้นสามารถเก็บอยู่ในรูปแบบของ Vector3 (x, y, z)

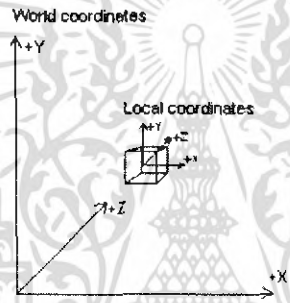
องค์ประกอบของ Vertex Data อื่นๆ ที่จำเป็นได้แก่ Vertex Color(r, g, b, a), Vertex Normal (x, y, z) และ Texture Coordinate (u, v) ซึ่งในส่วนของ Vertex Color นั้นจะใช้ก็ต่อเมื่อต้องการกำหนดสีให้กับ Vertex โดยตรงเท่านั้นและจะต้องไม่มีแสงเข้ามาเกี่ยวข้อง หากมีแสงเข้ามาเกี่ยวข้องจะต้องใช้ Vertex Normal ซึ่งแสดงถึงทิศทางของ Vertex เพื่อที่จะนำไปคำนวณผลกระทบที่เกิดจากแสง

ตัวอย่าง Structure ของ Vertex ที่ใช้โดยทั่วไปสามารถกำหนดได้ดังต่อไปนี้

```
struct Vertex
{
    Vector3 Position;        //Position
    Vector3 Normal;          //Normal
    Vector2 UV;              //Texture Coordinate
};
```

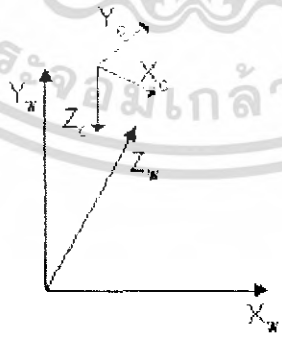
2.3.2 Transformation

ในขั้นแรกของ Pipeline ข้อมูล Vertex (Vertex Position) ส่วนมากจะอยู่ใน Local Coordinate System (Model Space) ซึ่งของแต่ละวัตถุจะแยกออกจากกันTransformation ครั้งแรกของ Pipelineจะเป็นการแปลงจาก Model Space ไปเป็น World Space ซึ่งจะเป็น Unify Space ของทุกวัตถุ เรียกว่า World Transform



รูปที่ 2.2 World และ Local Coordinate (World Transform สามารถเป็นได้ทั้ง Translation Rotation และ Scaling)

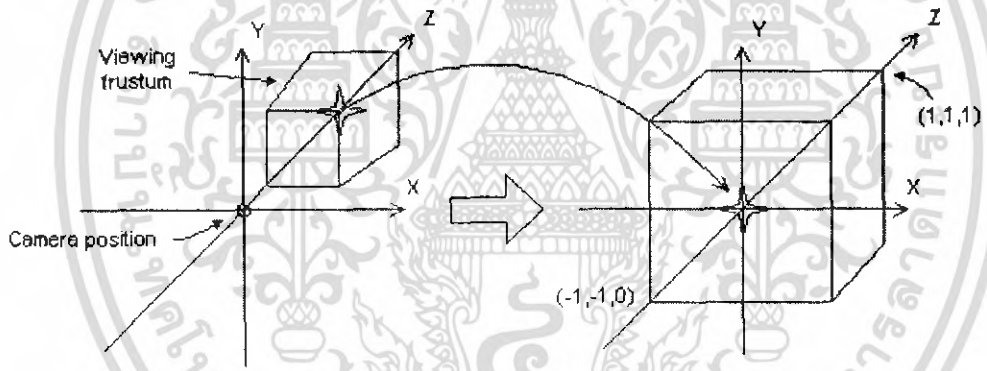
ในลำดับต่อมาจะกล่าวถึงการแปลงจาก World Space ไปสู่ Camera Space เรียกว่า View Transform ซึ่งจะแสดงถึงการจัดวางตำแหน่งและทิศทางของกล้องที่ใช้แสดงผลนั่นเอง โดย Matrix Formula แสดงถึง View Transformation (V) = T*Rz*Ry*Rx



รูปที่ 2.3 View และ World Coordinate

```
//ตัวอย่างการสร้าง View Transform Matrix
zaxis = normal(Eye - At)
xaxis = normal(cross(Up, zaxis))
yaxis = cross(zaxis, xaxis)

xaxis.x     yaxis.x     zaxis.x     0
xaxis.y     yaxis.y     zaxis.y     0
xaxis.z     yaxis.z     zaxis.z     0
-dot(xaxis, eye)  -dot(yaxis, eye)  -dot(zaxis, eye)  1
```

ต่อมาคือ Projection Transform ซึ่งในส่วนนี้วัตถุต่างๆจะถูก Scale ตามระยะห่างของวัตถุกับ Viewer เพื่อที่จะสามรถแสดงถึงความลึกของภาพได้ วัตถุที่อยู่ใกล้กว่าจะมีขนาดใหญ่กว่า และเช่นกันวัตถุที่อยู่ไกลกว่าจะมีขนาดเล็กกว่า เรียกว่า Homogeneous Space (Projection Space) แต่ก็มีบางชนิดของ Projection Transform ที่ไม่ทำการ Scale เช่น Affine หรือ Orthogonal Projection



รูปที่ 2.4 Projection Transform

```
//ตัวอย่างการสร้าง Projection Matrix จาก Field of View (FOV)
xScale    0        0          0
0      yScale      0          0
0         0     zf/(zf-zn)     1
0         0     -zn*zf/(zf-zn) 0
where:
yScale = cot(fovY/2)
xScale = aspect ratio * yScale
```

ขั้นสุดท้ายคือการทำ Clipping เพื่อที่จะตัด Vertex ที่มองไม่เห็นออก เพื่อที่ Rasterizer จะไม่ต้องเสียเวลาคำนวณสีและแสงเงา ถัดจาก Clipping ก็คือ Transformation ครั้งสุดท้ายที่ Vertices จะถูก Scale ตาม Viewport ที่กำหนดและเปลี่ยนเป็น Screen Coordinate เพื่อทำ Rasterization ให้ปรากฏเป็นภาพบนจอ

ตามปกติแล้ว Transform Matrix แบบต่างๆจะถูกนำมารวมเข้าด้วยกันเป็น World-View-Projection Matrx (WVP) เพื่อทำการ Apply ให้กับแต่ละ Vertex อย่างไรก็ตามผู้ใช้ยังมีความจำเป็นต้องรู้ถึงพื้นฐานของ Transformation Matrix ที่อยู่ในรูปแบบของ Matrix 4x4 ซึ่งใช้ในการ Translation Rotation และ Scaling ด้วยสามารถสรุปได้ดังนี้

//Translation

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ Tx & Ty & Tz & 1 \end{bmatrix}$$

//Scaling

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} Sx & 0 & 0 & 0 \\ 0 & Sy & 0 & 0 \\ 0 & 0 & Sz & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

//Rotation about x-Axis

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & \sin\theta & 0 \\ 0 & -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

//Rotation about y-Axis

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

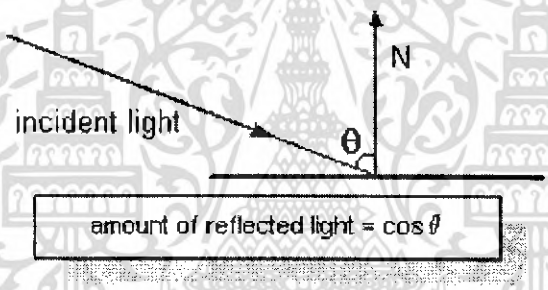
//Rotation about z-Axis

$$\begin{bmatrix} x' & y' & z' & 1 \end{bmatrix} = \begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

2.3.3 Material & Lighting

ก่อนที่จะสามารถพูดถึงเรื่องแสงได้นั้น ต้องพูดถึงเรื่องของ Material ก่อน เนื่องจาก Material แสดงถึงการที่ Polygon จะสะท้อน (Reflect) หรือเปล่งแสง (Emit) ออกมาในภาพ คุณสมบัติของ Material ได้แก่ Diffuse Reflection, Ambient Reflection, Light Emission และ Specular Highlight ซึ่งแต่ละคุณสมบัติสามารถแสดงได้ด้วย ColorARGB ซึ่งประกอบไปด้วยสี และค่า Alpha ซึ่งจะนำไปใช้ในกระบวนการ Alpha Blending ด้วย

Diffuse และ Ambient Reflection จะแสดงถึงคุณสมบัติในการสะท้อน Ambient และ Diffuse Light ในฉากของวัตถุ เนื่องจากว่าในฉากจะประกอบไปด้วย Diffuse Light มากกว่า Ambient Light การสะท้อน Diffuse Light จึงเป็นตัวกำหนดสีของวัตถุ และเนื่องจากว่า Diffuse Light มีทิศทาง มุมระหว่างแสงและ Normal Vector จึงมีผลกับความเข้มของการสะท้อน โดยการสะท้อนจะมากที่สุดเมื่อทิศของแสงขนานกับทิศของ Normal Vector เมื่อมุมมากขึ้นก็จะสะท้อนน้อยลงดังรูปที่ 2.5



รูปที่ 2.5 แสดงการคำนวณความเข้มของการสะท้อน

ส่วน Ambient Reflection จะเหมือนกับ Ambient Light คือไม่มีทิศทางและมีผลต่อวัตถุทั้งชิ้นเท่าเทียมกันซึ่งจะเห็นได้ชัดเมื่อวัตถุไม่มี Diffuse light มากระทำ

Emission คือการที่วัตถุสามารถเปล่งแสงออกมาได้ โดยเราสามารถใช้ Emissive Property ทำให้ดูเหมือนวัตถุเป็นแหล่งกำเนิดแสงได้ ส่วน Specular Reflection สามารถทำให้วัตถุดูเป็นมันเงา โดยทั่วไป Material Structure จะสามารถกำหนดสี และค่าความเงาได้

องค์ประกอบแสง หรือ Light ประกอบไปด้วย 4 ส่วนเช่นเดียวกันกับ Material คือ Ambient Diffuse Emissive และ Specular โดยทั้งสี่ส่วนจะมีวิธีการคำนวณแตกต่างกันออกไป แต่ผลลัพธ์จะถูกนำมารวมกันดังต่อไปนี้

```
//การรวมแต่ละองค์ประกอบของแสงเป็น Global Illumination
Global Illumination = Ambient Light + Diffuse Light + Specular Light + Emissive Light
//Ambient Light
Ambient Lighting = MaterialAmbientColor*[Global AmbientColor + LightAmbientColor]
```

```
//Diffuse light
Diffuse Lighting = DiffuseColor*LightDiffuseColor*(Normal.Direction)
//Specular Light
Specular Lighting = SpecularColor * [LightSpecularColor * (Normal • HalfWay)^Power *]
//Emissive Light
Emissive Lighting = EmissiveColor
//Attenuation
Attenuation = 1/( ConstantAttenuate + LinearAttenuate * Distance + QuadraticAttenuate * d^2)
```

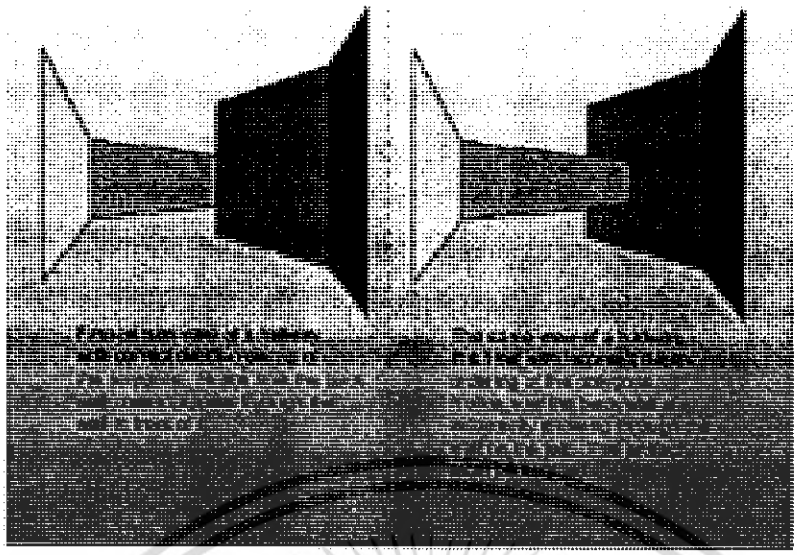
แสงยังสามารถแบ่งได้หลายประเภทโดยประเภทที่นิยมได้แก่ Point และ Directional Light ซึ่งมีคุณสมบัติคล้ายคลึงกัน Point Light จะเทียบได้กับหลอดไฟ คือมีจุดกำเนิด และเปล่งแสงไปรอบๆ แต่ Directional จะเทียบได้กับดวงอาทิตย์ส่องแสงมายังโลก คือมีแต่ทิศทางของแสง แต่ไม่คิดตำแหน่งของแสงเนื่องจากถือว่าอยู่ห่างจากวัตถุมาก

2.3.4 Texture

Texture mapping คือการนำภาพ 2มิติ มา map เป็นลวดลายให้กับวัตถุ ข้อมูลที่สำคัญได้แก่ Texture Image หรือรูปที่จะใช้ และ Texture Coordinate ของแต่ละ Vertex โดย Texture Coordinate แสดงถึงบริเวณของรูปที่จะใช้ในการ Map ให้กับ Vertex นั้นโดยในกระบวนการ Rasterization จะมีการทำ Sampling Texture Image เพื่อนำค่าสีของรูปมาใส่ เนื่องจาก Texture mapping มีลูกเล่นมากมาย อีกทั้งยังสามารถใช้หลาย Texture พร้อมกันได้จึงขออธิบายในส่วนนี้ไว้เพียงเท่านี้ และจะอธิบายเพิ่มเติมควบคู่ไปกับการออกแบบTexture Class

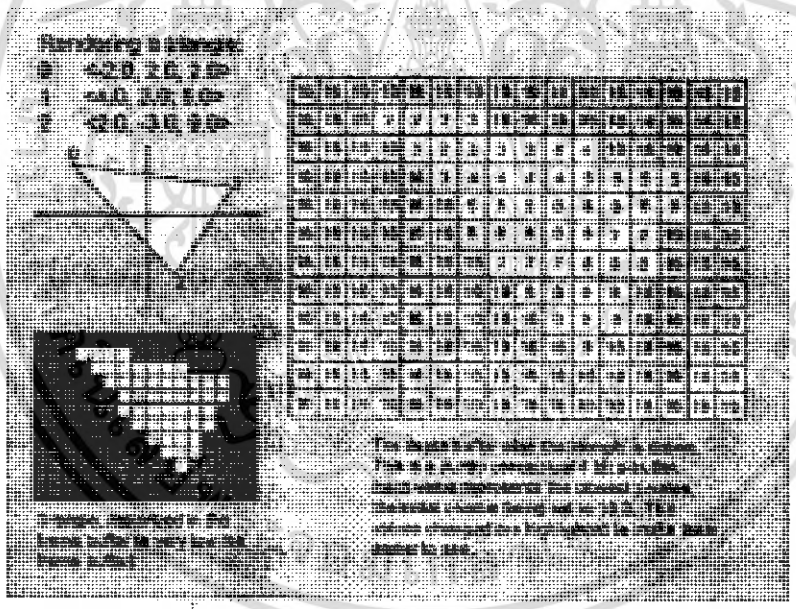
2.3.5 Buffers

Buffer ที่ใช้โดยทั่วไปสามารถแบ่งได้สามประเภทคือ Front/Back Buffer หรือ Frame Buffer, Depth Buffers และ Stencil Buffer ซึ่งแต่ละชนิดมีหน้าที่ต่างๆกัน โดย Frame Buffer จะเก็บค่าสีและค่า Alpha ส่วน Depth Buffer จะใช้ในการแก้ปัญหา Depth Problem ดังในรูปที่ 2.6 โดยจะเก็บค่าความลึกไว้ดังในรูปที่ 2.7 ส่วน Stencil Buffer เป็น Buffer อเนกประสงค์สามารถใช้ในการเลือกส่วนที่ต้องการแสดงผลด้วยวิธีต่างๆได้ เปรียบได้กับการทาสีโดยปิดส่วนที่ไม่ต้องการเอาไว้ก่อน



A B

รูปที่ 2.6 แบบ Aแสดงผลลัพธ์ที่ต้องการ และ แบบ B แสดงการแก้ปัญหาโดยการทำ Polygon Depth Ordering



รูปที่ 2.7 ตัวอย่างปัญหาในการใช้ Depth Buffer

2.3.6 Graphics API

ในการเขียนโปรแกรมติดต่อ Graphic Hardware (Graphics Card)ให้แสดงผลภาพสามมิตินั้นจำเป็นต้องอาศัยตัวกลางอย่าง Graphics API เข้ามาช่วยทำให้ง่ายขึ้น ในปัจจุบันมีทางเลือกใหญ่ๆเพียง 2 ทางเลือกคือ Direct Graphics (Microsoft) และ OpenGL (Silicon Graphics) อย่างไรก็ตามผู้พัฒนาก็สามารถเลือกที่จะเขียน Driver และ Software 3D Graphics and

Rasterization เองได้ Game Engine ที่ดีควรที่จะออกแบบให้สามารถเลือก API ในการทำงานได้อย่างอิสระ โดยให้ส่วนหลักของ Engine ไม่ขึ้นตรงกับ Graphics API ใดอย่างเฉพาะเจาะจง

2.4. ลักษณะของ Physically Based Simulation Module ใน Game Engine ที่มีอยู่ในปัจจุบัน

ในปัจจุบัน Physics Engine ได้มีการพัฒนาอย่างมากซึ่งได้มีความสามารถนอกเหนือจากการ ตรวจสอบการชน ,การเคลื่อนที่ของวัตถุแข็ง , ฟิสิกส์พื้นฐาน คือสามารถที่จะทำการตอบสนองหลังจากเกิดการชนกันได้กับวัตถุเกือบทุกชนิด สามารถทำการเปลี่ยนแปลง ภาพและเสียงจาก การคำนวณทางฟิสิกส์ มีความสามารถหาความสัมพันธ์ของพลังงานในการเคลื่อนที่ อนิเมชั่นของโบน ฟิสิกส์เกี่ยวกับยานพาหนะ แรงลอยตัวของวัตถุในของเหลว

2.5. แนะนำกระบวนการ Physically Based Simulation

ในการจำลองนั้นจะมีสามส่วนด้วยกันคือ

1. การเคลื่อนที่เชิงเส้น
2. การเคลื่อนที่เชิงมุม
3. การตรวจสอบการชน

2.5.1. การเคลื่อนที่เชิงเส้น

เริ่มจาก วัตถุที่คลื่นที่ได้ต้องมีมวล ของวัตถุและถ้าไม่มีความเร็วเริ่มต้นวัตถุจะไม่เคลื่อนที่แต่ถ้ามีแรงไปกระทำกับวัตถุ ทำให้วัตถุเกิดการเคลื่อนที่ตาม หลัก ทฤษฎีทางฟิสิกส์ โดยการจำลองจะนำแรงที่กระทำกับวัตถุทั้งหมด มารวมกัน เป็นแรงลัพธ์ แล้วนำแรงลัพธ์ที่ได้ไปคำนวณหาความเร็วที่กระทำวัตถุ เมื่อได้ความเร็วที่กระทำกับวัตถุ ณ. เวลานั้นแล้วก็จะเอาไปรวมกับความเร็วเดิมของวัตถุทำให้ได้ความเร็วใหม่ และเมื่อได้ความเร็วใหม่ก็จะนำความเร็วตัวนี้ไปบวกกับตำแหน่ง เดิมของวัตถุ ได้ตำแหน่งใหม่ของวัตถุในระนาบสามมิตินั้นเอง โดยทุกขั้นตอนจะเป็นการกระทำ ณ เวลาหนึ่งเท่านั้น

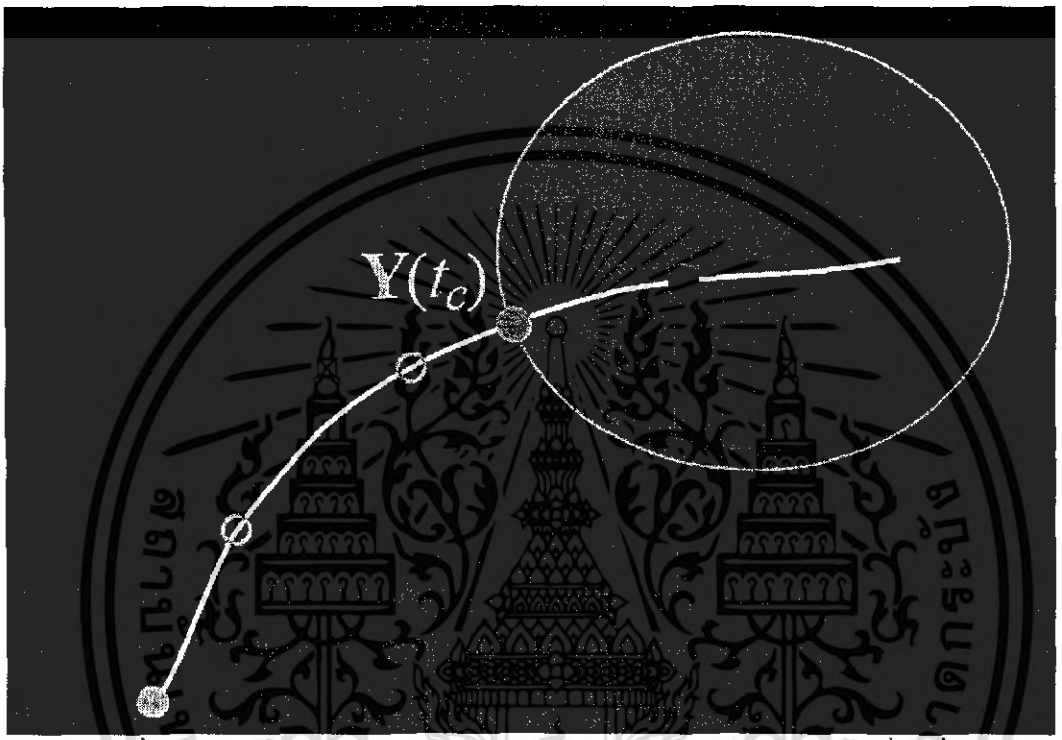
2.5.2. การเคลื่อนที่เชิงมุม

ในการเคลื่อนที่เชิงมุมก็เช่นเดียวกับการเคลื่อนที่เชิงเส้น โดยเราจะนำแรงที่กระทำมารวมเป็นแรงลัพธ์ แล้วนำมาหาว่าแรงลัพธ์นั้นกระทำผ่านจุดศูนย์กลางวัตถุรึไม่เพราะถ้ากระทำผ่านจุดศูนย์กลางของวัตถุ จะทำให้แรงที่กระทำนั้นไม่ทำให้เกิดความเร็วเชิงมุม แต่ถ้าไม่ผ่านจุดศูนย์กลางเราก็จะนำแรงลัพธ์นั้นมาหาความเร็วเชิงมุมตาม หลักทฤษฎีทางฟิสิกส์ แล้วนำความเร็วที่ได้ไปทำการบวกเพิ่มกับความเร็วเชิงมุมเดิม จากนั้นก็หมุนวัตถุ

ด้วยความเร็วเชิงมุมใหม่ที่ได้ทำให้วัตถุที่เคลื่อนที่นั้นหมุนไปด้วย โดยทุกขั้นตอนจะเป็นการกระทำ ณ เวลาหนึ่งเท่านั้น

2.5.3. การตรวจสอบการชน

เมื่อวัตถุเคลื่อนที่ได้ ก็จะมีโอกาสที่จะเคลื่อนที่มาชนกันดังนั้นจึงต้องมีการตรวจสอบการชนกันของวัตถุในการเคลื่อนที่แต่ละ ครั้งด้วย เพราะถ้าไม่มีการตรวจสอบการชนกันจะเกิดการทับกันของวัตถุที่ชนกันเหมือนกับว่าไม่มีวัตถุอีกชิ้นอยู่ทำให้ไม่สมจริง ซึ่งการตรวจสอบการชนสามารถดูได้จากรูปที่ 2.8



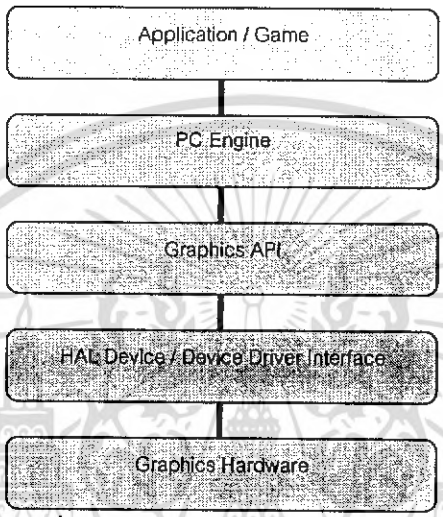
รูปที่ 2.8 แสดงการจำลองการตรวจสอบการชนกันของวัตถุในแต่ละการเคลื่อนที่

จากรูปที่ 2.8 จะเห็นได้ ว่าในการเคลื่อนที่แต่ละครั้งนั้นจะมีการตรวจสอบว่าวัตถุนั้นได้เคลื่อนที่เข้าไปใน วงกลมรีรัง ซึ่งถ้ามีการเคลื่อนที่เข้าไปแล้วก็จะทำการขยับจุดนั้นออกมาเป็นเวลา เท่ากับ Contact time เวลาที่วัตถุเริ่มชน เพื่อให้จุดที่ได้นั้นอยู่ที่พื้นผิวของวัตถุ ซึ่งหลังจากที่ตรวจสอบได้ว่าวัตถุสองชิ้นชนกันแล้วก็ต้องมาคำนวณว่าจะเกิดอะไรหลังจากการชนกัน โดยในส่วนนี้สามารถกำหนดได้หลายรูปแบบ เช่น ทิศทางของวัตถุที่เคลื่อนที่ชนกันเกิดการเปลี่ยนแปลงหลังจากที่ชนรวมไปถึงความเร็วของวัตถุก่อนชนกับหลังชนด้วย หรือ อาจจะเกิดการเปลี่ยนสีของวัตถุหลังจากที่ชนกันซึ่งในส่วนนี้ขึ้นกับผู้ออกแบบว่าต้องการให้เป็นไปในทิศทางใด

บทที่ 3
การออกแบบและทฤษฎีเพิ่มเติม

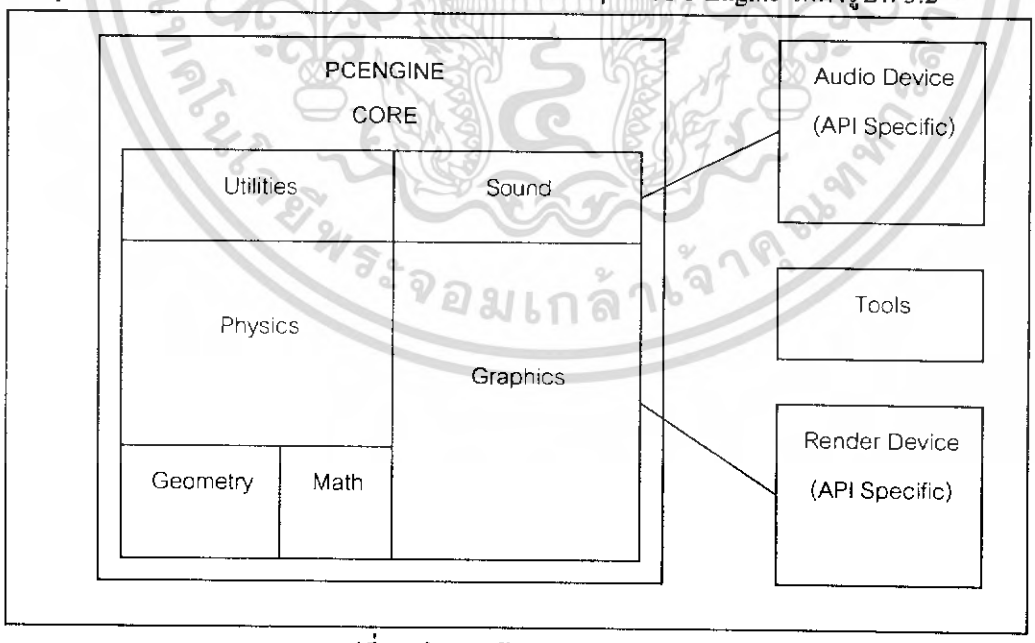
3.1 แนวทางการออกแบบCore Architecture

การทำงานของ PC Engine นั้นสามารถจะอยู่ระหว่าง Application Layer และ API Layer ดังรูปที่ 3.1



รูปที่ 3.1 PC Engine Operating Layer

จากองค์ประกอบหลักๆของ Game Engine ซึ่งกำหนดให้ส่วนกลาง (Core) เป็น API Independent จึงสามารถกำหนดลักษณะโครงสร้างคร่าวๆ ของ PC Engine ได้ดังรูปที่ 3.2



รูปที่ 3.2 โครงสร้างของ PC Engine

ส่วนประกอบต่างๆของส่วนกลางทั้งหมดจะอยู่ในข้อกำหนดของ API Independent ซึ่งจะต้องไม่มีส่วนใดอ้างถึง API ชนิดใดๆเลย ส่วนใดที่ต้องทำการติดต่อกับ API จะต้องออกแบบให้อยู่ในรูปของ Interface ที่สามารถติดต่อกับ APIชนิดต่างๆได้โดยสมบูรณ์ ซึ่งได้แก่ ส่วนประกอบจำพวก Render Device และ Audio Device

3.2 Mathematical Module

ส่วนนี้จะต้องช่วยอำนวยความสะดวกในการคำนวณทางคณิตศาสตร์ต่างๆ การที่จะให้ใช้งานได้ง่ายนั้นจะใช้ความสามารถของ Overloaded Operator เท่าที่จะเป็นไปได้ เพราะนอกจากผู้ใช้จะเห็นภาพของการกระทำได้ชัดเจนยิ่งขึ้นและยังง่ายต่อการเขียนสมการที่ต่อเนื่องกันอีกด้วย โดยคลาสที่สำคัญพื้นฐานได้แก่ Vector2, Vector3, Vector4, Martix3, Matrix4 และ Quaternion

Vector2, Vector3 และ Vector4เป็นคลาสที่ใช้ในการคำนวณ Vector Arithmetic ต่างๆมี Operation ที่จำเป็นได้แก่ +, -, *(scalar), Vector Magnitude, Normalize, Cross และ Dot Product

Matrix3 และ Matrix4 เป็นคลาสที่ใช้ในการคำนวณ Matrix Operation ต่างๆ ได้แก่ +, -, *(Matrix), *(Scalar), / (Scalar), Determinant, Adjoint, Transpose และ Inverse นอกจากนี้ Matrix3 ถูกนำไปใช้งานส่วนมากเป็น Rotation Matrix จึงต้องมีเมธอดช่วยในการสร้าง Rotation Matrix ตามแกนที่ระบุดังนี้ RotateX, RotateY, RotateZ และ RotateArbitary

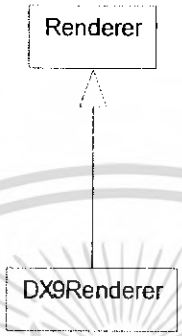
Quaternion ถูกนำไปใช้ใน Rigid body Simulation เพื่อที่จะป้องกันปัญหาที่เกิดจากความคลาดเคลื่อนของการคำนวณ Floating Point ที่จะส่งผลให้เกิด Scaling และยังใช้ใน Animation อีกด้วยเพื่อป้องกันปัญหา Gimbals' Lock ที่จะเกิดเมื่อใช้ Rotation Matrix นอกจากนี้ยังใช้ทรัพยากรน้อยกว่า Rotation Matrix และให้การ Interpolation ที่ Smooth กว่าอีกด้วย โดยมี Operation ที่สำคัญดังนี้ +, -, *(Quaternion), *(Scalar), /, Length, Dot, Normalize, Conjugate และ Inverse นอกจากนี้ยังต้องมีส่วนช่วยในการสร้าง Rotation แบบต่างๆ ตลอดจนถึงการแปลงกลับระหว่าง Quaternion และ Rotation Matrix และการทำ Interpolation ได้แก่ FromAxisAngle, ToRotationMatrix, FromRotationMatrix และ Slerp(linear spherical interpolation)

เนื่องจากส่วนนี้จะเป็นส่วนที่ถูกใช้บ่อยที่สุดในส่วนประกอบทั้งหมดของเกมเอนจิน ดังนั้นในการพัฒนาจึงถูกให้ความสำคัญทางด้านความเร็วในการดำเนินงานมากกว่าส่วนอื่นๆ โดยได้นำข้อดีของเทคโนโลยี SIMD (ในที่นี้คือ Intel SSE) มาใช้เพื่อช่วยเพิ่มความเร็วในการประมวลผลในส่วนของฟังก์ชั่นการทำงานที่เห็นว่าการใช้คำสั่งประเภท SIMD จะให้ความเร็วในการทำงานที่มากกว่าการใช้คำสั่งพื้นฐานของ FPU เช่น Matrix multiplication, Matrix inversion และ Vector crossing เป็นต้น

3.3 Graphics Module

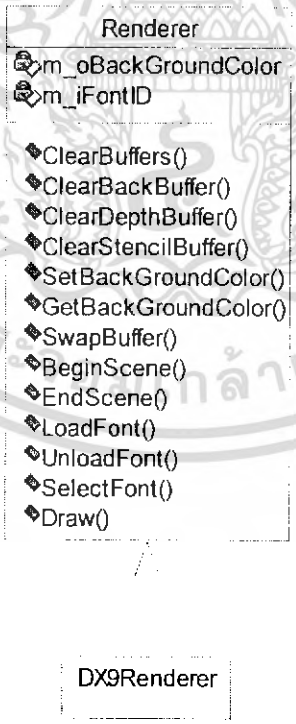
3.3.1 Basic Capability of Renderer

เพื่อให้ได้มาซึ่งความเป็น API Independent ของ Graphics Module ส่วนของ RenderDevice (Renderer) จึงเป็นเพียงแค่ Interface Class สำหรับ user ที่ติดต่อกับ Class Renderer อื่นที่ implement ด้วย API ที่เฉพาะเจาะจง



รูปที่ 3.3 คลาส DX9Renderer จะ Implement Method ต่างๆที่กำหนดโดยคลาส Renderer

นอกจากการ Initialize และการหา Device Capability แล้วความสามารถเบื้องต้นของคลาส Renderer ได้แก่การเคลียร์หน้าจอเป็นสีที่กำหนดได้, สามารถล้างค่าต่างๆใน Buffer ต่างๆ ได้, สามารถสลับ Back/Front Buffer ได้ และสามารถแสดงผลตัวอักษร (Draw) ด้วยสีต่างๆที่ตำแหน่งที่กำหนดได้ ซึ่งทั้งหมดนี้จะถูก Implement ในคลาส DX9Renderer ด้วย DirectX



รูปที่ 3.4 เมธอดพื้นฐานของคลาส Renderer

3.3.2 Object System

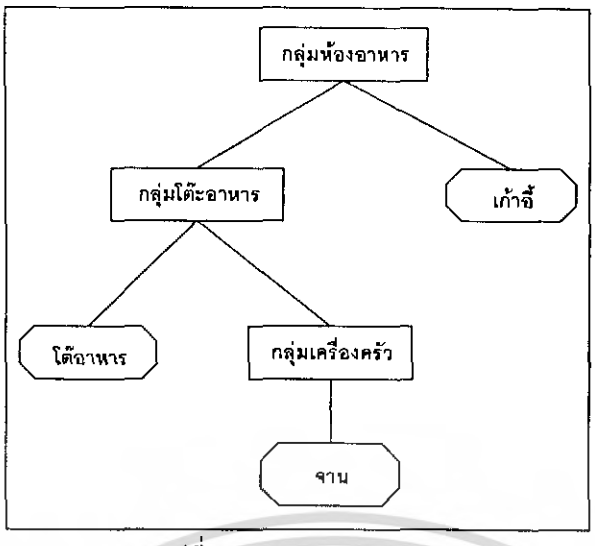
ใน Game Engine นั้นการจัดการกับวัตถุต่างๆมีความสำคัญมาก เนื่องจากโครงสร้างจะมีความซับซ้อน และผู้ใช้มีความต้องการที่จะใช้วัตถุต่างๆอยู่เสมอ การกำหนดชื่อและหมายเลขให้กับวัตถุจึงจะช่วยให้การค้นหาเป็นไปอย่างมีประสิทธิภาพยิ่งขึ้น นอกจากนี้ในบางกรณีผู้ใช้อาจต้องการทราบถึงชนิดของวัตถุเหล่านั้นในขณะนั้น (โดยเฉพาะในกรณีที่เกิดการ Casting จาก Base Class)โดยอาศัยกระบวนการ Real-time Type Information รวมไปถึงความสามารถในการใช้วัตถุต่างๆร่วมกันอย่างมีประสิทธิภาพ เช่นการนับจำนวนการถูกอ้างอิงถึงของวัตถุหนึ่งๆ (Reference counting) เพื่อที่จะสามารถทำลายวัตถุทิ้งโดยอัตโนมัติ เมื่อไม่ถูกอ้างถึงอีกต่อไป (ทั้งนี้เนื่องจากพัฒนาด้วยภาษา C++ ซึ่งไม่มีระบบ Garbage collection) โดยการอ้างถึงเหล่านี้จะกระทำผ่าน Pointer ซึ่ง Pointer เหล่านี้ต้องมีความสามารถในการนับ Reference เพิ่มเติมให้กับวัตถุ

รูปที่ 3.5 Object System

ในส่วนของ Object Retrieval หรือส่วนช่วยในการค้นหา Object นั้นจะเห็นได้ว่า Object บางชนิดอาจจะบรรจุ Object ชนิดอื่นๆอยู่ จึงจะต้องทำการ Override Method ที่เกี่ยวข้องให้ทำการสืบค้นได้อย่างถูกต้อง

3.3.3 Basic Scene graph

เพื่อที่จะอธิบายฉากที่ซับซ้อนให้สะดวกและมีประสิทธิภาพยิ่งขึ้น จึงได้นำแนวคิดของ Hierarchical scene management มาใช้ โดยมีลักษณะดังภาพต่อไปนี้



รูปที่ 3.6 Scene Hierarchy

จากรูปที่ 3.6 แสดงให้เห็นถึงฉากหนึ่งในห้องอาหารซึ่งมีโต๊ะและเก้าอีตั้งอยู่ บนโต๊ะมีชุดของเครื่องครัวซึ่งประกอบไปด้วยจาน

เมื่อกำหนดคุณลักษณะต่างๆให้กับNodeใดๆแล้ว ลูกของNodeนั้นจะต้องมีลักษณะเดียวกันด้วย เช่นกำหนดให้กลุ่มของเครื่องครัวมีลักษณะพื้นผิว (Material) ที่เป็นมันวาว ไม่ว่าจะเพิ่มช้อนหรือส้อมเข้ามาก็จะมีลักษณะมันวาวเช่นเดียวกันกับจาน หรือแม้แต่การกำหนดแสงให้กับกลุ่มห้องอาหาร วัตถุทั้งหมดในห้องอาหารก็จะได้รับผลจากแสงนั้นด้วยเป็นต้น โดยตัวอย่างที่เห็นได้อย่างชัดเจนที่สุดคือการคำนวณเกี่ยวกับ Transformation และ Bounding Volume ของแต่ละNode

เมื่อพิจารณาในส่วนของ Transformation (World and local transformation) สามารถอธิบายได้ดังต่อไปนี้ ซึ่ง World Transformation จะถูกนำไปใช้กับ Graphic API ในการกำหนดตำแหน่งของวัตถุ

World(Room) = Local(Room)

World(Chair) = World(Room) . Local(Chair)
= Local(Room) . Local(Chair)

World(Table Group) = World(Room) . Local(Table Group)
= Local(Room) . Local(Table Group)

World(Table) = World(Table Group) . Local(Table)
= Local(Room) . Local(Table Group) . Local(Table)

World(Utensil Group) = World(Table Group) . Local(Utensil Group)
= Local(Room) . Local(Table Group) . Local(Uten.)
World(Disc) = World(Utensil Group) . Local(Disc)
= Local(Room) . Local(Table Group) . Local(Utensil Group) . Local(Disc)

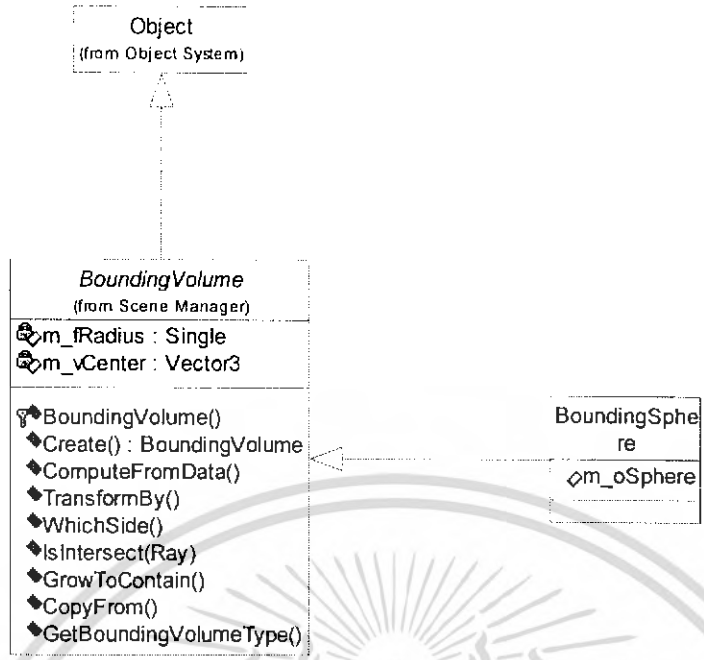
จะเห็นได้ว่าสามารถจัดการการเคลื่อนย้ายวัตถุที่มีความสัมพันธ์กันได้โดยง่าย เช่นการย้ายกลุ่มของโต๊ะจะทำให้ทุกสิ่งที่อยู่บนโต๊ะถูกเคลื่อนย้ายไปด้วยเป็นต้น

นอกจากการใช้ประโยชน์ในเรื่องของความสามรถในการอธิบายฉากต่างๆที่มีความซับซ้อนได้แล้ว Scene graph ยังสามารถเพิ่มประสิทธิภาพให้กับเอนจิน โดยจะสามารถเห็นได้ชัดเมื่อมีวัตถุจำนวนมากอยู่ในฉาก การส่งวัตถุทั้งหมดไปให้กับการ์ดแสดงผลจะมีผลกระทบกับประสิทธิภาพการทำงานได้ (ถึงแม้ว่าการ์ดแสดงผลจะมีกระบวนการ Culling อยู่บน Pipeline ก็ตาม แต่ก็ยังจะต้องเสียเวลาทำการ Transform กับทุกๆ Vertices ของวัตถุก่อนอยู่ดีจึงจะทราบว่าวัตถุนั้นอยู่นอกบริเวณที่สามารถมองเห็นได้) การทำ Visibility test นั้นจะสามารถทำได้เมื่อเราใช้ประโยชน์จาก Bounding Volume ของแต่ละNode เราจะสามารถทราบได้ว่าวัตถุที่อยู่ภายในห้องอาหารทั้งหมดไม่สามารถมองเห็นได้จากการตรวจสอบเพียงครั้งเดียวในNodeกลุ่มห้องอาหาร โดยวิธีการตรวจสอบจะกล่าวถึงในส่วนของ Frustum Culling การออกแบบเชิงวัตถุของ Scene graph จะถูกกำหนดตามคุณสมบัติต่างๆตามที่ได้กล่าวมาดังต่อไปนี้

Spatial มีคุณสมบัติพื้นฐานได้แก่การเปลี่ยนจาก Local transformation -> World transformation และคำนวณจาก World bounding volume จาก Local Bounding Volume

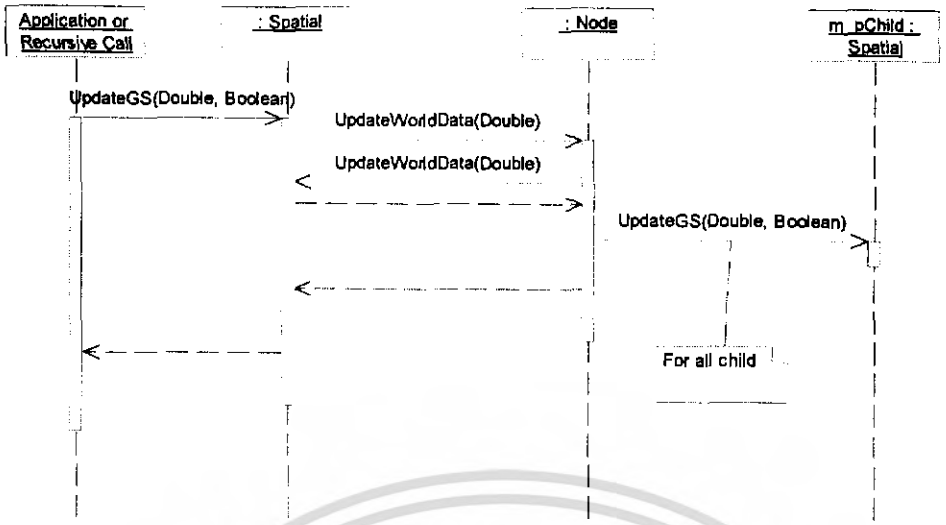
Node จะทำหน้าที่กระจายกระบวนการ Update State ต่างๆไปยัง children ทุกตัว (เฉพาะวัตถุชนิดนี้เท่านั้นที่สามารถมีลูกได้)

Geometry เป็นคลาสสำหรับวัตถุซึ่งจะปรากฏที่ leaf node ของ tree เท่านั้นโดยจะเก็บข้อมูลเกี่ยวกับ Local bounding volume (Model bound) และข้อมูลเกี่ยวกับวัตถุได้แก่ Vertices, Indices และ Normal โดยการคำนวณขนาดและรูปร่าง Model Bound สามารถเลือกทำได้หลายรูปแบบและวิธี ในที่นี้เลือกใช้ Bounding Sphere โดยแต่ละชนิดก็จะใช้วิธีที่แตกต่างกันออกไปในการคำนวณ นอกจากนี้ Bounding Volume ยังต้องสามารถขยายตัวเองให้ครอบคลุม Bounding Volume ในระดับต่ำกว่าและย้อนกลับไปยัง Root Nodeได้ด้วย และยังต้องสามารถตรวจได้ว่าอยู่ด้านใดของ Plane เพื่อใช้ในการทำ Frustum Culling และสามารถตรวจจับการชนกันกับ Ray สำหรับการทำ Picking

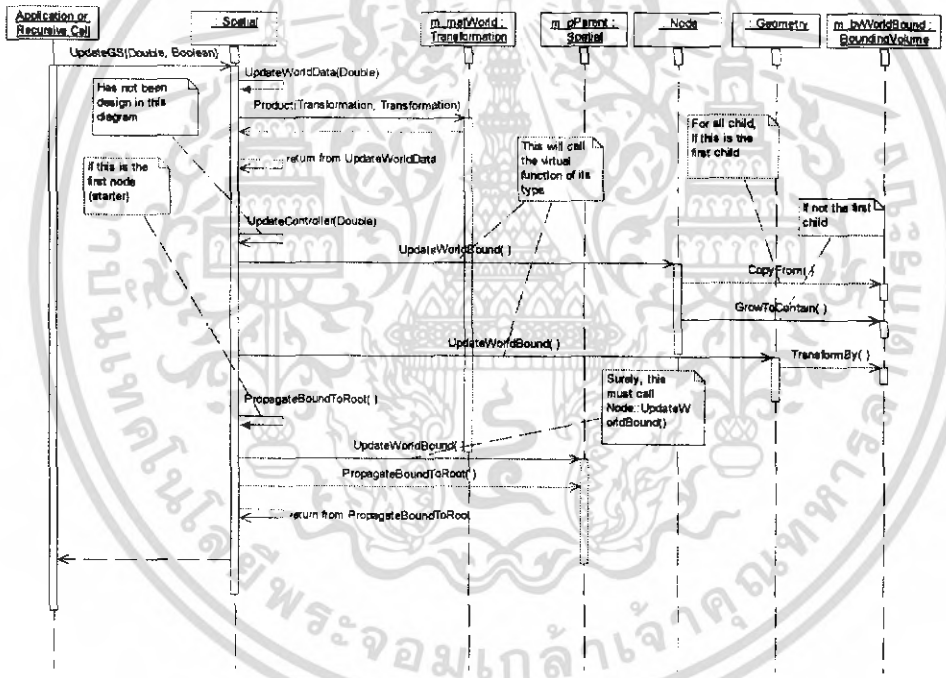


รูปที่ 3.7 BoundingVolume และ BoundingSphere

รูปที่ 3.8 Basic Scene Graph



รูปที่ 3.9 Sequence Diagram เมื่อทำการเรียก UpdateGS บน Node



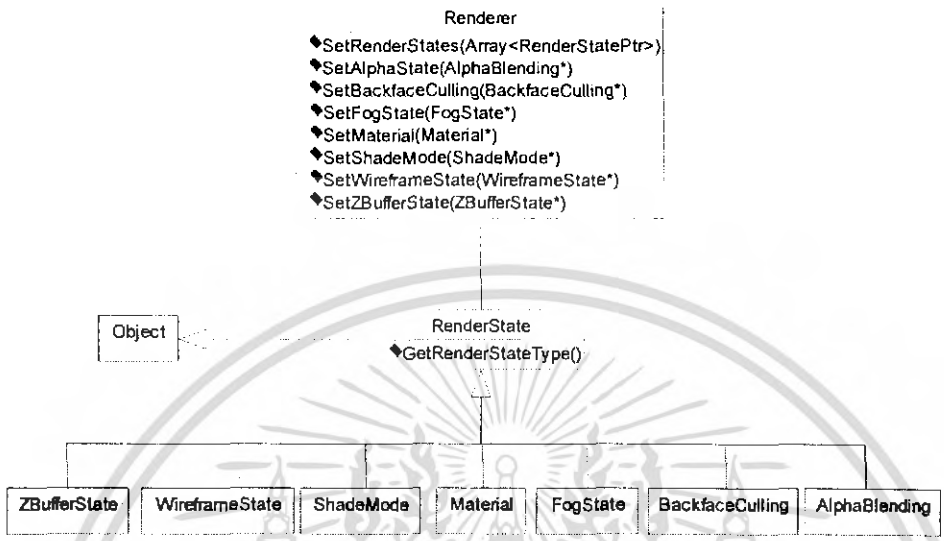
รูปที่ 3.10 Sequence Diagram เมื่อทำการเรียก UpdateGS บน Spatial Derived Classes

3.3.4 Render States

ในการแสดงผลภาพสามมิติ (Draw Primitive)การแสดงผลของ Graphics API ส่วนมากจะใช้รูปแบบของ State Machine การกำหนด Render State ต่างๆก็เช่นเดียวกัน โดย Render State ประกอบไปด้วยส่วนต่างๆได้แก่ การแสดงผลแบบ Wireframe, การทำ Back Face Culling, การ

กำหนด Shade mode, การกำหนดค่า Alpha Blending, การกำหนดค่า Fog Effect, การกำหนด Material และการกำหนดสถานะของ Z-Buffer

ส่วนของคลาส Renderer จะต้องสามารถกำหนด Render State ต่างๆได้อย่างสะดวก เนื่องจาก Render State ชนิดต่างๆมีรายละเอียดและวิธีการกำหนดค่า ต่างๆกันไปในแต่ละ API



รูปที่ 3.11 โครงสร้างการทำงานของ Render State

เมธอดในการติดตั้ง Render State ต่างๆเช่น SetAlphaState, SetMaterial จะ Implement ไว้ใน Derived Renderer Class เนื่องจากแต่ละ API มีวิธีที่แตกต่างกันออกไป

3.3.4.1 Alpha Blending

กระบวนการ Alpha Blending ใน Render State คือการกำหนดค่าสำหรับการทำ Frame Buffer Alpha Blending ซึ่งจะใช้ค่า Alpha ที่รับมาจาก Vertex Color หรือ Material หรือ Texture ซึ่งจะกำหนดค่า Alpha (Transparency Value) สำหรับในแต่ละวัตถุ มาใส่ใน Frame Buffer

เมื่อทำ Alpha Blending สีสองสีจะถูกรวมเข้าด้วยกัน คือ Source Color และ Destination Color โดย Source Color มาจากวัตถุที่โปร่งแสง และ Destination Color มาจากค่าที่อยู่บน Frame Buffer อยู่แล้วซึ่งมาจากผลของการ Render วัตถุอื่นๆก่อนหน้านี้ โดยการควบคุมกระบวนการเป็นไปตามสูตรต่อไปนี้

$$\mathbf{FinalColor = ObjColor * SrcBlendFactor + PixelColor * DestBlendFactor} \qquad (3.1)$$

ObjectColor คือสีของวัตถุที่กำลังจะถูก Render ที่ตำแหน่งนั้น ส่วน PixelColor คือสีปัจจุบันที่ปรากฏอยู่บน Frame Buffer ที่ตำแหน่งนั้น โดยค่า Blend Factor ต่างๆสามารถเป็นไปตามตารางที่ 3.1

Blend Factor	รายละเอียด
ZERO	(0, 0, 0, 0)
ONE	(1, 1, 1, 1)
SRCCOLOR	(Rs, Gs, Bs, As)
INVSRCCOLOR	(1-Rs, 1-Gs, 1-Bs, 1-As)
SRCALPHA	(As, As, As, As)
INVSRCALPHA	(1-As, 1-As, 1-As, 1-As)
DESTALPHA	(Ad, Ad, Ad, Ad)
INVDESTALPHA	(1-Ad, 1-Ad, 1-Ad, 1-Ad)
DESTCOLOR	(Rd, Gd, Bd, Ad)
INVDESTCOLOR	1-Rd, 1-Gd, 1-Bd, 1-Ad)
SRCALPHASAT	(f, f, f, 1); f = min(As, 1-Ad)

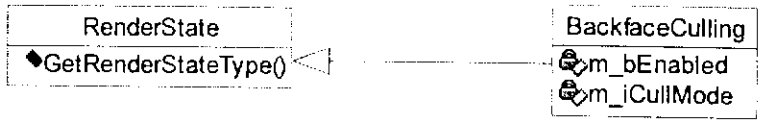
ตารางที่ 3.1 รายละเอียดของ Blend Factor

นอกคลาส AlphaBlending จำเป็นต้องเก็บค่าต่างๆ ที่สมการต้องการได้ ซึ่งได้แก่ Source และ Destination Blend Factor แล้วยังต้องมีความสามารถในการทดสอบว่าควรจะทำการเขียนค่า Alpha ลงบน Frame Buffer หรือไม่ (Alpha Testing) โดยทำการเทียบค่าAlphaบนFrame Buffer กับค่าReference ค่าหนึ่งด้วย Condition ต่างๆได้แก่ Never, Always, Less, Less Equal, Equal, Greater, Greater Equal และ Not Equal

รูปที่ 3.12 AlphaBlending Class

3.3.4.2 Back-face Culling

BackfaceCulling เป็นการตัดPolygon ที่มองไม่เห็นออกเนื่องจากหันหลังให้กับกล้อง โดยจะดูจากค่า Normal และทิศทางของกล้องเป็นหลักซึ่งสามารถกำหนดทิศได้ตามลักษณะIndex ของการวาดว่าเป็นการวาดทวนเข็ม หรือตามเข็มนาฬิกา (Cull ModeJ)



รูปที่ 3.13 BackfaceCulling Class

3.3.4.3 Fog State

Graphic Hardware ในปัจจุบันสนับสนุนการทำงานของ Pixel Based Fog ซึ่งจะกระทำในขั้นตอนของ Rasterization โดยจะทำการ Blend ค่าสีของ Fog เข้ากับค่าสีบน Frame Buffer ดังสมการต่อไปนี้

$$I = FogFunction(Distance)$$

$$FinalColor = I \times CurrentColor + (1-I) \times FogColor \qquad (3.2)$$

Fog Function ต่างๆได้แก่ Linear, Exponential และ Exponential Squared โดยมีสูตรดังต่อไปนี้

//Linear Fog Function

$$I = \frac{fogend - dis\tan ce}{fogend - fogstart}$$

//Exponential Fog Function

$$I = \frac{1}{e^{dis\tan ce \times density}}$$

//Exponential Squared Fog Function

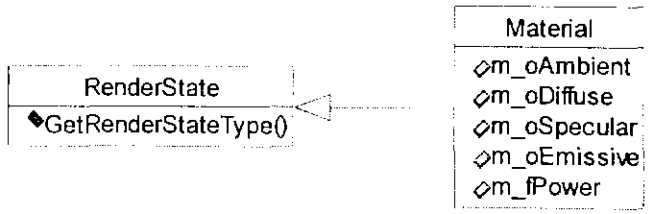
$$I = \frac{1}{e^{(dis\tan ce \times density)^2}}$$

ดังนั้นคลาส Fog จะต้องเก็บข้อมูลได้แก่ Fog Mode (Function), Fog Color, Fog Start, Fog End และ Fog Density

รูปที่ 3.14 FogState Class

3.3.4.4 Material

ความรู้เบื้องต้นเกี่ยวกับ Material ได้อธิบายไว้แล้วในบทที่ 2 การออกแบบจะต้องสามารถเก็บข้อมูลทั้งหมดที่เป็นคุณสมบัติของ Material ไว้ให้ได้เพื่อที่ Derived Renderer จะสามารถนำไปใช้ได้อย่างถูกต้อง



รูปที่ 3.15 Material Class

3.3.4.5 Shade Mode

Shade mode ใช้ระบุถึงความเข้มของแสงที่จุดใดๆบน Face ของ Polygon โดยส่วนมาก Graphics API จะสนับสนุนการทำงานสองแบบ คือ Flat Shading และ Gouraud Shading

Flat Shading จะใช้สีของ Material หรือสีของ Vertex แรกของ Face สำหรับทั้ง Face ส่วน Gouraud Shading จะใช้ Vertex Normal และค่าคุณสมบัติของแสงมาคำนวณสีของแต่ละ Vertex จากนั้นทำการ Interpolate สีด้วย Linear Interpolation จะทำให้ดูนุ่มนวลกว่า

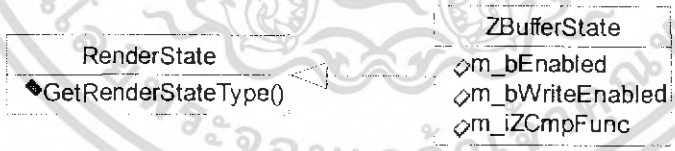
การออกแบบคลาส Shade Mode จะต้องเก็บคุณลักษณะที่สำคัญเพียงอย่างเดียวคือ Shade Mode เท่านั้น

3.3.4.6 Wireframe State

Wireframe State ใช้แสดงถึงวิธีในการวาดภาพว่าจะทำการ Fill Polygon หรือให้ทำ Line Drawing เพียงอย่างเดียว คุณสมบัติของคลาสที่สำคัญจึงมีเพียงแค่สถานะเปิดหรือปิดเท่านั้น

3.3.4.7 Z-Buffer State

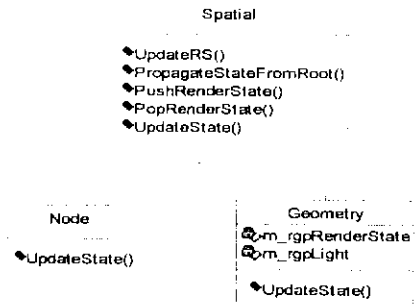
ในที่นี้หมายถึงเพียงแค่ส่วนของ Depth Buffer ซึ่งได้อธิบายไปแล้วในบทที่ 2 โดยคุณสมบัติของ Class ที่สำคัญได้แก่ Z-Compare Function ที่ใช้ในการทดสอบเพื่อเขียนลงบน Frame Buffer ได้แก่ Equal, Less, Less Equal, Greater, Greater Equal และ Not Equal, สถานะภาพใช้งาน และสถานะภาพเขียนค่า Z ลงบน Buffer เท่านั้น



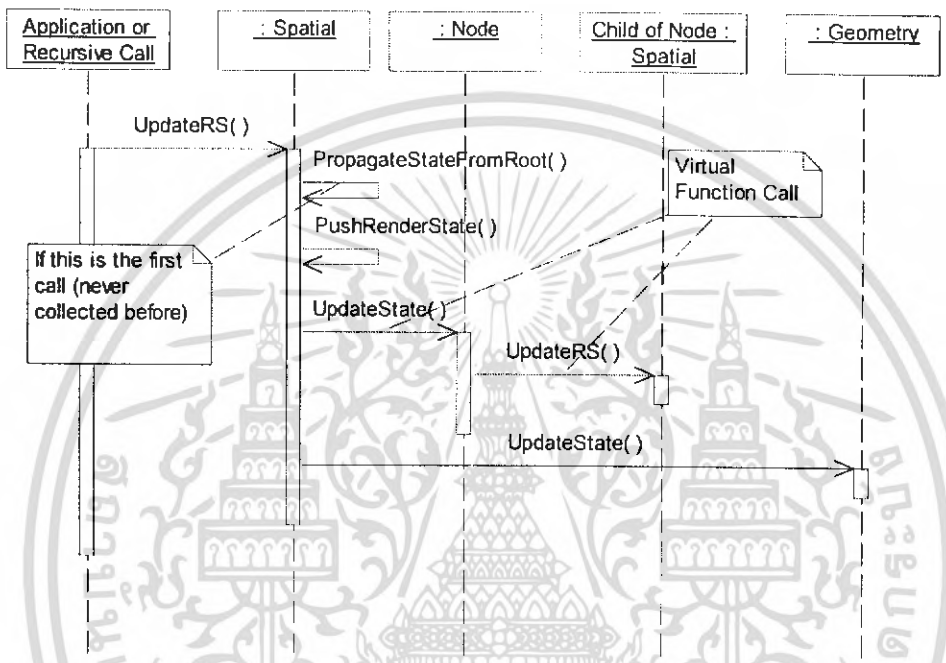
รูปที่ 3.16 ZBufferState Class

3.3.4.8 Update Render State ใน Scene graph

การ Update Render State ใน Scene graph สามารถทำได้โดยไล่สะสม State มาจาก Root Node โดยจะรวมไปถึงสะสม Light ที่ได้ติดตั้งไว้ตั้งแต่ Root Node ลงมาด้วย โดยเมื่อถึง Geometry จะต้องนำ State ล่าสุดที่ผ่านมาใส่ให้กับ Geometry เพื่อนำไปใช้ในการ Render



รูปที่ 3.17 Update Render State



รูปที่ 3.18 Sequence Diagram: Update Render State

3.3.5 Effect & Texture Mapping

PC Engine สนับสนุนการใช้ Vertex Color และ Multi-texture ผ่านทางคลาส Effect (แต่เดิมควรจะชื่อ Skin เพียงแต่จะสร้างความสับสนให้กับผู้ใช้ ระหว่าง Skin ใน Skeletal Animation) โดยคุณสมบัติของคลาส Effect จะประกอบไปด้วย Vertex Color, Textures และ UVs (Texture Coordinates) ส่วนของ Vertex Color มีความตรงไปตรงมาคือ Color ของแต่ละ Vertex แต่ในส่วนของคลาส Texture นอกจากคุณสมบัติพื้นฐานอย่างเช่น Imageที่ใช้ Filterที่ใช้และ UV Mode (Addressing Mode) แล้วจะต้องสามารถให้ข้อมูลในการทำ Multi-texture ได้ด้วย Apply Mode แบบต่างๆได้แก่ Replace, Modulate, Decal, Blend, Add และ Combine ซึ่งเป็นตัวกำหนด Operation ที่Texture Stage ต่างๆ (กรณีใช้ Multi-texture) ได้อย่างมีประสิทธิภาพ โดยในกรณีของ Combine Apply Mode จะมีดังต่อไปนี้

Apply Mode	คำอธิบาย
REPLACE	FinalColor = TextureColor, FinalAlpha = TextureAlpha
DECAL	FinalColor = (1-TextureAlpha) x CurrentColor + TextureAlpha x TextureColor, FinalAlpha = CurrentAlpha
MODULATE	FinalColor = TextureColor x CurrentColor, FinalAlpha = TextureAlpha x CurrentAlpha
BLEND	FinalColor = (1 – TextureColor) x CurrentColor + TextureColor x CurrentColor, FinalAlpha = TextureAlpha x CurrentAlpha
ADD	FinalColor = CurrentColor + TextureColor, FinalAlpha = CurrentAlpha x TextureAlpha
COMBINE	Customizable Textures Blending

ตารางที่ 3.2 Texture Apply Mode

ผู้ใช้สามารถกำหนดการใช้ Textureได้เองอย่างอิสระทั้งในส่วนของ Color และ Alpha Channel ในกรณีที่กำหนด Apply Mode เป็น COMBINE ซึ่งจะขึ้นกับ Combine Function ต่างๆ ดังนี้

Combine Function	สมการ
REPLACE	v0
MODULATE	v0 x v1
ADD	v0 + v1
ADD_SIGNED	v0 + v1 – ½
SUBSTRACT	v0 – v1
INTERPOLATE	v0 x v2 + v1 x (1 –v2)

ตารางที่ 3.3 Combine Function

โดย vi คือ Operand ซึ่งจะกำหนดโดย Combine Source และ Combine Operation ต่างๆ ด้วยกันดังนี้

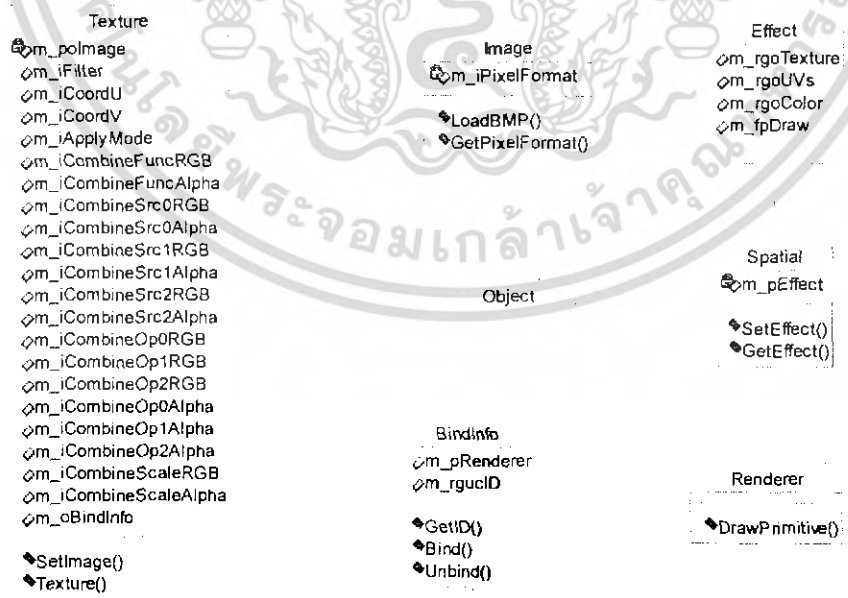
Source Operation	Source Color	1- Source Color	Source Alpha	1 – Source Alpha
TEXTURE	TextureColor	1- TexColor	TextureAlpha	1- TexAlpha
PRIMARY_COLOR	CurrentColor	1- CurrColor	CurrentAlpha	1- CurrAlpha
CONSTANT	ConstantColor	1- ConstColor	ConstantAlpha	1- ConstAlpha
PREVIOUS	PreviousColor	1- PrevColor	PreviousAlpha	1- PrevAlpha

ตารางที่ 3.4 Combine Function Operand

จากตารางที่ 3.3 จะเห็นได้ว่าข้อมูลด้านคุณสมบัติที่ใช้ในการทำ Texture Combine สามารถประกอบไปด้วยจำนวนสูงสุดถึง 3 ชุด คือ v0 v1 และ v2 ซึ่งแต่ละชุดประกอบไปด้วย RGB และ Alpha Channel ทั้ง Source และ Operation นอกจากนี้จะต้องสามารถทำการ Scale ผลลัพธ์ได้ด้วยเป็นจำนวนเท่าได้แก่ 1 2 และ 4 เท่า

นอกจากคุณสมบัติดังกล่าวมาทั้งหมดของ Class Effect แล้วจะเห็นได้ว่าในบางกรณี Drawing Function ที่ใช้งานอยู่อาจจะไม่ตรงความต้องการของผู้ใช้ เช่นการใช้ Texture ในทางอื่นที่นอกเหนือจากปกติ หรือการทำ Multi-pass Rendering ซึ่งต้องการผลจากการวาด Node อื่นก่อน (Global Effect) คลาส Effect จึงต้องสามารถกำหนดให้ผู้ใช้สามารถใช้ Drawing Function ของตัวเองได้ด้วย อย่างไรก็ตามโดยปกติแล้วจะเป็น Drawing Function พื้นฐานของ PC Engine

การที่ Texture ใดถูกเรียกใช้โดย Renderer แล้วครั้งหนึ่งแสดงว่า Texture นั้นถูกจัดการและเก็บบน Video Memory หรือส่วนอื่นๆที่จัดการโดย Graphics API แล้วไม่มีความจำเป็นที่จะสร้าง Texture นั้นซ้ำอีกครั้งจึงทำการ Bind Texture Object ที่ใช้บน API นั้นเข้ากับ Texture Object ของ PC Engine

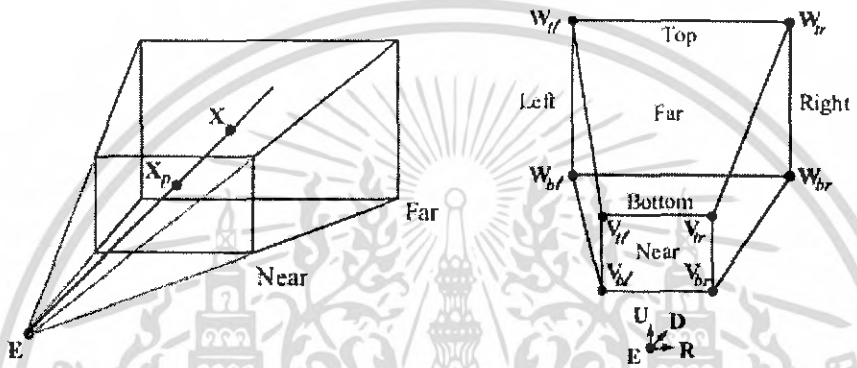


รูปที่ 3.19 Effect Class

3.3.6 Scene Graph Rendering

การแสดงฉากต่างๆด้วย Scene graph นั้นเริ่มต้นจากคลาส Camera ซึ่งจะถูกติดตั้งให้กับ Render Device โดยคุณลักษณะของ Camera ประกอบไปด้วย View Frustum, Viewport, Frame และมีหน้าที่ทำ Visibility testing (Culling), สร้าง Ray สำหรับนำไปใช้ในการทำPicking ด้วย

การที่ Camera มีคุณสมบัติเช่นเดียวกันกับSpatial คือมีตำแหน่ง Location และ Orientation ทำให้เราสามารถสืบทอดคุณสมบัติมาจากคลาสSpatial ได้ การทำเช่นนี้ยังทำให้ Camera เป็นส่วนหนึ่งของฉากนั้นๆด้วยเช่นกล้องวงจรปิดที่ติดอยู่ในห้องแต่ละห้องเป็นต้น โดยส่วนของ Frame ของกล้องซึ่งก็คือ Location, Direction, Up, Right สามารถใช้ Location, Rotation Matrix ในส่วนของ Transformation แทนได้ให้ใช้หน่วยความจำอย่างมีประสิทธิภาพ



รูปที่ 3.20 Standard Camera Model

รูปที่ 3.20 แสดงถึง Camera Frustum และ Plane ที่เกี่ยวข้องซึ่งตำแหน่งที่ต้องคำนวณจาก Frustum Paremeter ทั่วไปเช่น Near Far Left Right Top Bottom ได้แก่ตำแหน่ง Vtl Vtr Vbl และ Vbr เป็น Vertexของ Near Plane ส่วนของ Far Plane ก็จะใช้ลักษณะเดียวกันได้แก่ Wtl Wtr Wbl และ Wbr

$$Vbl = E + dminD + uminU + rminR$$

$$Vtl = E + dminD + umaxU + rminR$$

$$Vbr = E + dminD + uminU + rmaxR$$

$$Vtr = E + dminD + umaxU + rmaxR$$

$$Wbl = E + dmax/dmin\ (dminD + uminU + rminR)$$

$$Wtl = E + dmax/dmin\ (dminD + umaxU + rminR)$$

$$Wbr = E + dmax/dmin\ (dminD + uminU + rmaxR)$$

$$Wtr = E + dmax/dmin\ (dminD + umaxU + rmaxR) \qquad (3.3)$$

ส่วนของการทำ Frustum Culling จะเป็นการทดสอบกับแต่ละ Frustum Plane ที่มี Normal ชี้เข้าด้านใน ถ้า Bounding Volume ใดอยู่หลัง Plane ก็จะมองไม่เห็น โดยสมการของ Plane สามารถหาได้จากสูตรทั่วไปของ Plane ซึ่งจุด X ใดๆบน Near Plane คือ E + dminD

$$D \cdot X = D \cdot (E + dminD) = D \cdot E + dmin \qquad (3.4)$$

ในส่วนของ Far Plane ก็เช่นเดียวกัน X คือ E + dmaxD

$$-D \cdot X = -D \cdot (E + dmaxD) = -D \cdot E - dmax \qquad (3.5)$$

ส่วนของ Left Plane จะประกอบไปด้วยจุด 3 จุดคือ E, Vtl และ Vbl ดังนั้น Normal Vector คือ

$$\begin{aligned}(Vbl - E) \times (Vtl - E) &= (dminD + uminU + rminR) \times (dminD + umaxU + rminR) \\ &= (dminD + rminR) \times (umaxU) + (uminU) \times (dminD + rminR) \\ &= (dminD + rminR) \times ((umax - umin)U) \\ &= (umax - umin)(dminD \times U + rminR \times U) \\ &= (umax - umin)(dminR - rminD) \qquad (3.6)\end{aligned}$$

ดังนั้นเวกเตอร์ Normal หนึ่งหน่วยของ Left Plane และ สมการของ Left Plane คือ

$$N_l = \frac{d_{min}R + r_{min}D}{\sqrt{d_{min}^2 + r_{min}^2}}, N_r.(X - E) = 0 \qquad (3.7)$$

ในทำนองเดียวกันก็จะสามารถหาสมการของ Right Bottom และ Top Plane ได้ดังนี้

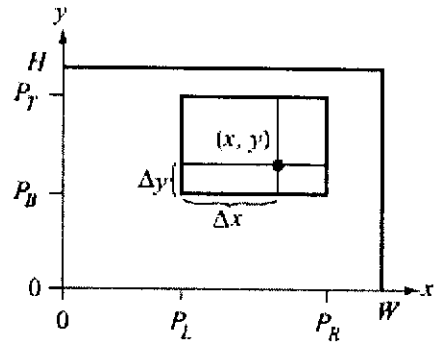
$$N_r = \frac{-d_{min}R + r_{max}D}{\sqrt{d_{min}^2 + r_{max}^2}}, N_r.(X - E) = 0 \qquad (3.8)$$

$$N_b = \frac{d_{min}U + u_{min}U}{\sqrt{d_{min}^2 + u_{mix}^2}}, N_b.(X - E) = 0 \qquad (3.9)$$

$$N_t = \frac{-d_{min}R + u_{max}D}{\sqrt{d_{min}^2 + u_{max}^2}}, N_r.(X - E) = 0 \qquad (3.10)$$

สมการ Plane ต่างๆจะถูกเก็บไว้เฉพาะส่วนของ Coefficient เพื่อนำไปใช้คำนวณหา World Plane ที่ใช้ในการ Cull จริงๆ (World Plane เก็บในรูปของ Plane Object) ส่วนของ Viewport Parameters จะต้องถูก Normalize เนื่องจากว่ายังไม่ทราบความกว้างและความสูงของส่วนที่ใช้แสดงผลจริง (ขึ้นกับ Renderer) คุณสมบัติที่สำคัญได้แก่ Left Right Top และ Bottom โดยการเปลี่ยนแปลงทั้งหมดที่เกิดขึ้นกับ Camera จะต้องแจ้งให้ Renderer ทราบด้วยเพื่อที่จะส่งต่อให้กับ API ไปจัดการ

นอกจากการทำ Frustum Culling แล้ว Camera ยังสนับสนุนการทำ Picking ด้วยการช่วยสร้าง Picking Ray จากตำแหน่งของ Camera Ffpมีทิศเดียวกันกับบริเวณที่กำหนดบน Viewport ดังนี้



รูปที่ 3.21 กรณีของ Non-fullscreen viewport

เริ่มจาก Normalize ตำแหน่งที่กำหนดบน Screen Coordinate (x, y) โดยการ Inverse ค่า y ด้วยเนื่องจาก Screen Coordinate ถือว่า y = 0 อยู่ด้านบน

$$x' = x / (W - 1),\ y' = H - 1 - y / (H - 1) \qquad (3.11)$$

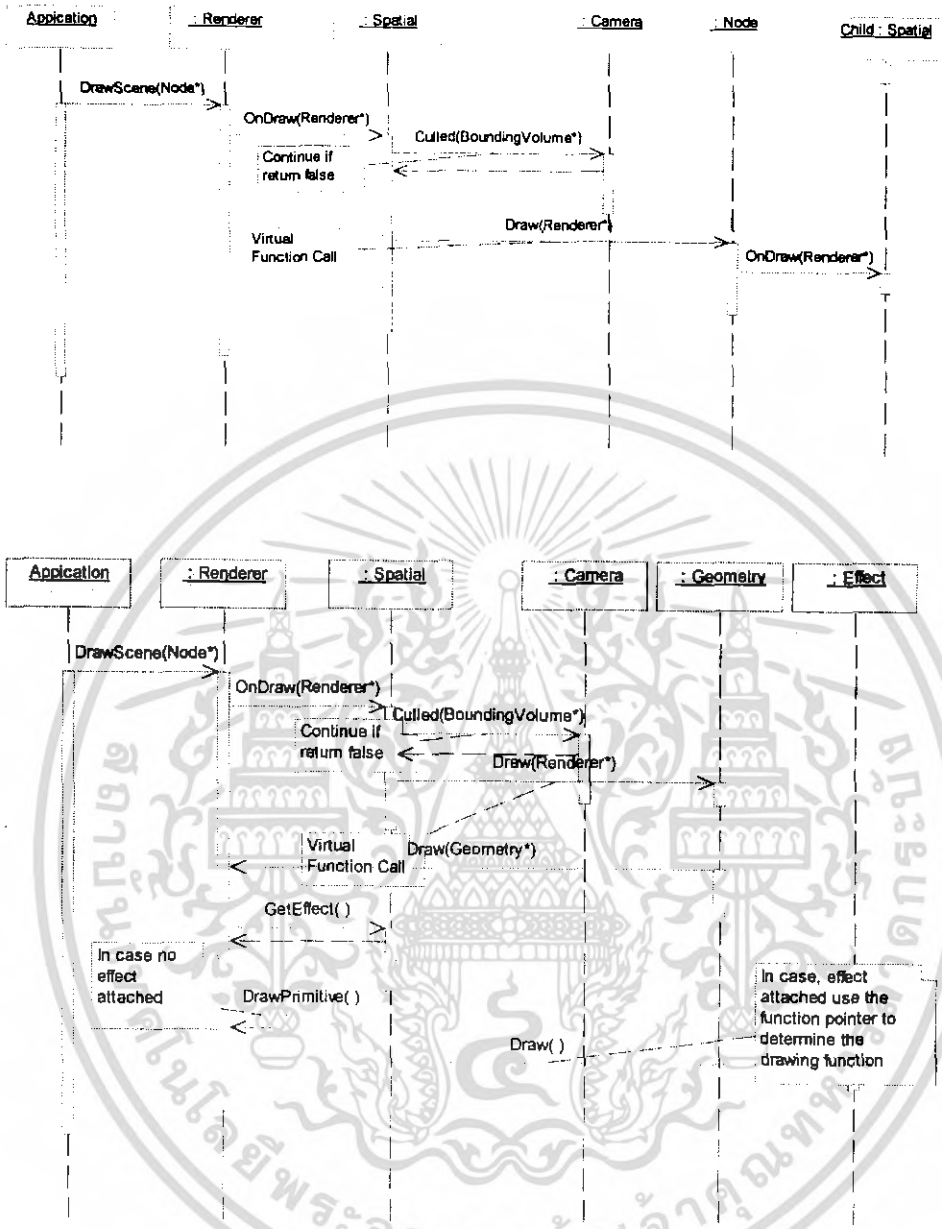
$$\Delta x = \frac{x' - P_L}{P_R - P_L}, \Delta y = \frac{y' - P_B}{P_T - P_B} \qquad (3.12)$$

Pick Ray = E + tU

$$U = d\min D + ((1 - \Delta x)r\min + \Delta x r\max)R + ((1 - \Delta y)u\min + \Delta y u\max)U \quad (3.13)$$

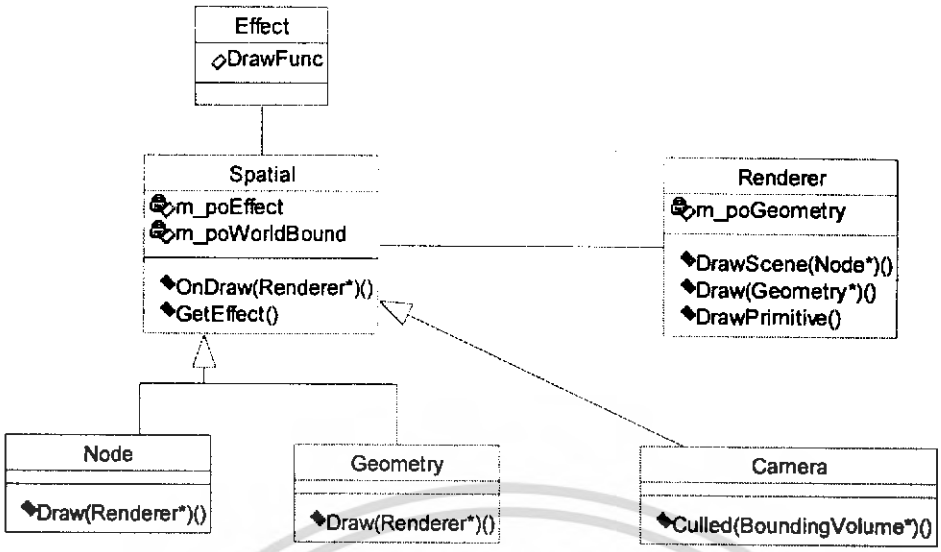
รูปที่ 3.22 Camera Class

การทำการ Render Scene graph นั้นจะเป็นการทำตามลำดับการเดินทางแบบ Depth first Traversal การวาดแบบ Single pass มีลำดับขั้นตอนดังต่อไปนี้



รูปที่ 3.23 Sequence Diagram: Single Pass Drawing

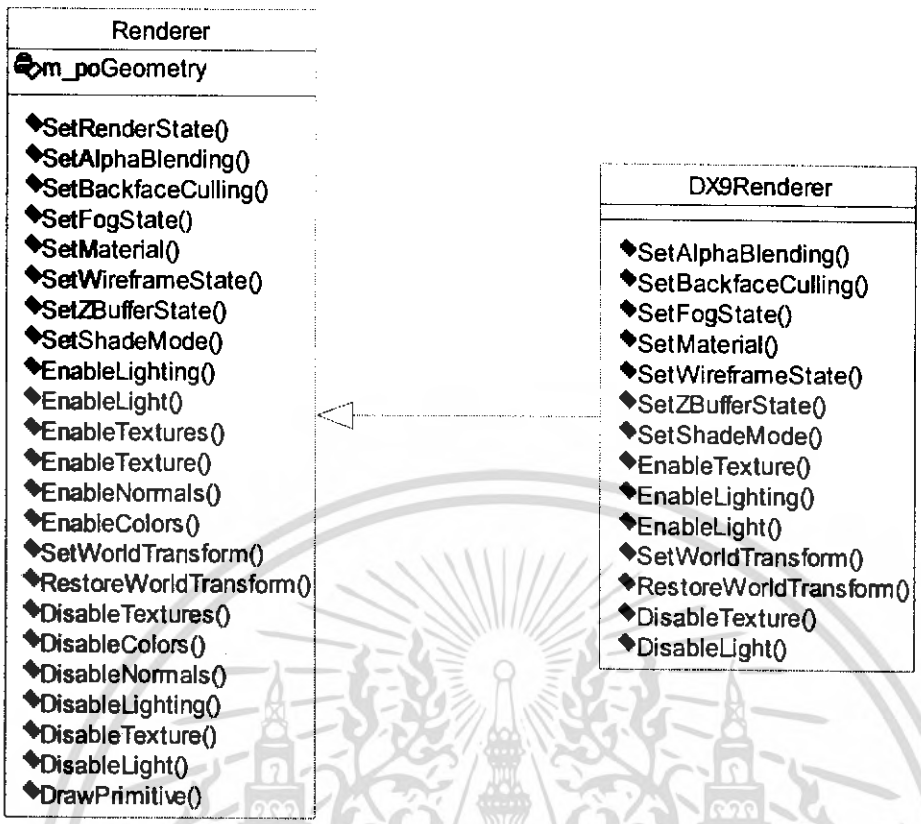
ในบางกรณีที่ต้องการสร้าง Effect ที่อาศัย Multi-pass Rendering (Global Effect) Effect Object จะต้องใช้ Drawing Function พิเศษที่กำหนดขึ้นเอง อย่างไรก็ตาม Effect ประเภทนี้จะติดตั้งอยู่บน Node ไม่ใช่ Geometry ดังนั้นในช่วงของการเรียก Node::Draw(Renderer*) จะแบ่งเป็นสองกรณีคือมี Effect และไม่มีในกรณีที่มี Effect ก็จะต้องเรียก Renderer::Draw(Node*) ก่อนเพื่อทำการ Draw Effect ที่ติดตั้งไว้ด้วย Drawing Function พิเศษ ซึ่งอาจรวมไปถึงการ Draw ลูกๆ ด้วยก็ได้



รูปที่ 3.24 Drawing Related Class Diagram

ในเมธอด DrawPrimitive จะดึงเอาคุณสมบัติต่างๆ ที่ติดมากับ Geometry เพื่อนำไปกำหนดให้กับ Derived Renderer มีขั้นตอนต่างๆดังนี้

- Set Render States
- Enable Lights
- Enable Vertices
- Enable Vertex Normals
- Enable Vertex Colors
- Enable Textures
- Apply World Transformation
- Create and Draw Vertex/Index Buffer (Array in OpenGL)
- Restore World Transformation
- Disable Textures
- Disable Vertex Colors
- Disable Vertex Normals
- Disable Vertices
- Disable Lighting



รูปที่ 3.25 Renderer Class เพิ่มเติม

3.3.7 Billboard

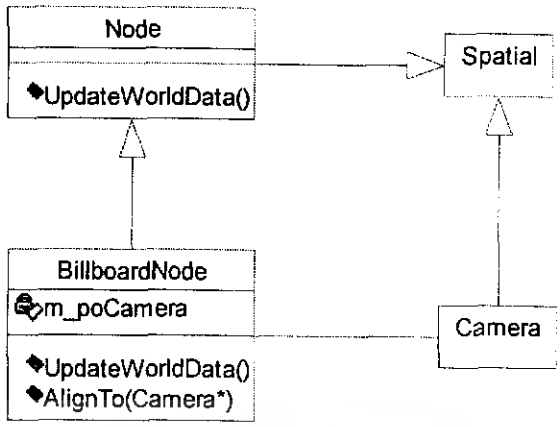
การทำ Billboard ไม่ได้มีอะไรมากไปกว่าการกำหนด World Transformation ของวัตถุให้มีทิศทางเข้าหากล้องซึ่งเดิมการทำ World Transformation เป็นหน้าที่ของ Scene graph (UpdateWorldData) อยู่แล้วการทำ Billboard ในระบบ Scene graph จึงไม่ใช่เรื่องยากเมื่อสามารถใช้การ Overridden method ที่ทำหน้าที่นั้นบนNodeได้ โดยสามารถคำนวณหามุมที่จะทำการหมุนวัตถุรอบแกน Y (Up) ของวัตถุนั้นได้ดังต่อไปนี้

ตำแหน่งของกล้องในเฟรมของ Billboard

$$CamLoc = BillboardRotMat^{-1} (CameraWorldLocation - BillboardWorldLocation) \quad (3.14)$$

ทิศของกล้องในแกน Y

$$Angle = ATan(CamLoc.X/CamLoc.Z) \quad (3.15)$$



รูปที่ 3.26 BillboardNode

3.3.8 Terrain

3.3.8.1 Large Terrain Rendering

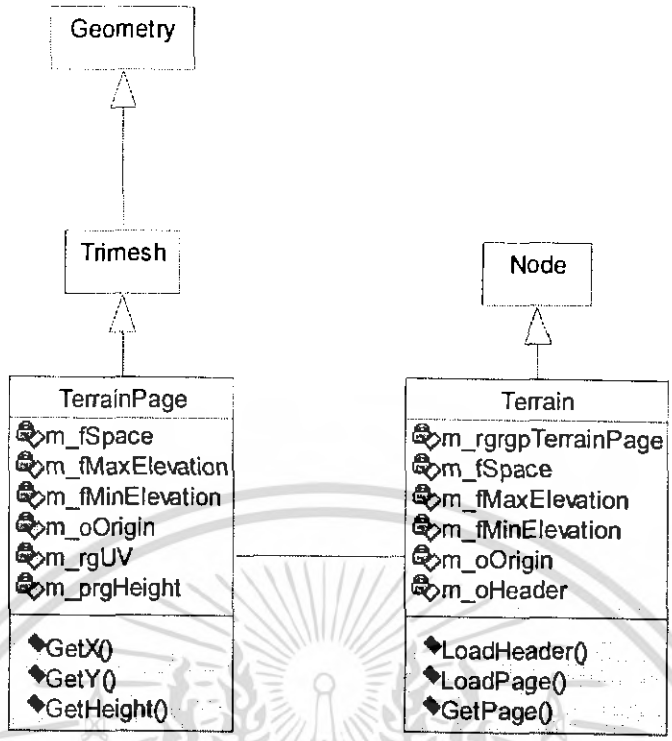
เมื่อข้อมูลที่ใช้ใน Terrain ที่สวยงามใช้ได้จริงจะมีจำนวนมหาศาลวิธีการจัดการที่ดีที่สุดคือการแบ่ง Terrain ออกเป็นส่วนย่อยๆเรียกว่า Page ซึ่งตาม Scene Graph Hierarchy จะจัดไว้เป็นลูกของ Terrain (Node) และเป็น leaf node เสมอ (Geometry) ซึ่งใน Height Field Header ก็ต้องบอกด้วยว่ามีกี่ page (แถวและหลัก) และ page มีขนาดเท่าใดเพื่อความง่ายต่อการคำนวณในภายหลัง (กำหนดให้เป็นสี่เหลี่ยมจัตุรัสขนาดเท่ากันหมด)

3.3.8.2 Height Field File Format

นอกจากจะให้ข้อมูลด้านความสูงของพื้นที่แล้ว ในส่วนหัวก็ต้องบรรจุ Header ลงไปด้วยเพื่อที่ Terrain Loader จะสามารถนำข้อมูลมาใช้ได้อย่างถูกต้อง กำหนด Header ดังต่อไปนี้

```
struct TerrainHeader
{
        int iNumPages;          //จำนวน Page ทั้งหมด
        int iNumRows;           //จำนวนแถว
        int iNumCols;           //จำนวนหลัก
        int iPageSize;          //ขนาดของ Page
};
```

ถัดจาก Header จะเป็น Height Field ของแต่ละ Page เรียงจากซ้ายไปขวา และจากบนลงล่างเพื่อความสะดวกในการนำไปใช้ การสร้าง Terrain เริ่มจากอ่าน Header ว่ามีจำนวน Page เท่าใด จากนั้นสร้าง Page และติดตั้งเข้ากับ Terrain Node โดย Page ที่สร้างเป็น Plane ที่แบ่ง Segment ตามระยะห่างของแต่ละ Vertex ที่กำหนด โดยแต่ละ Vertex จะมีความสูงตามที่ปรากฏใน Height Field Data รวมถึงการคำนวณ UV ให้กับแต่ละ Vertex ด้วย

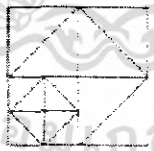


รูปที่ 3.27 Terrain

3.3.8.3 Continuous Level of Detail

ในการแสดงผล Terrain ที่มีความละเอียดสูงจะพบว่าในกรณีที่เป็นพื้นที่ราบไม่มีความจำเป็นต้องใช้ความละเอียดมากขนาดนั้น และบริเวณที่อยู่ไกลก็เช่นกัน ดังนั้นจึงมีวิธีที่จะลดความละเอียดของ Terrain ในกรณีที่เหมาะสมได้

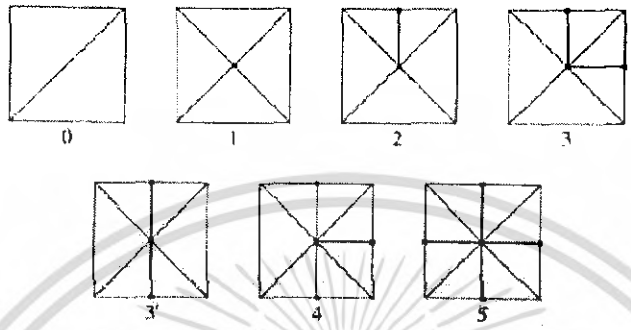
Pageหนึ่งPageจะถูกแบ่งเป็นBlock ซึ่งขนาดขึ้นอยู่กับจำนวน pixel ที่ Block นั้นปรากฏบนจอภาพ Pageหนึ่งๆอาจจะถูกแบ่งเป็นหลาย Block ที่มีขนาดไม่เท่ากัน (Block Simplification) ดังภาพต่อไปนี้



รูปที่ 3.28 Block

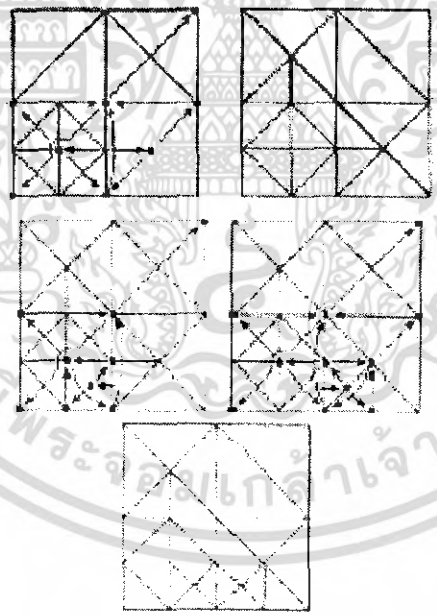
จากภาพด้านบนแสดง Page ซึ่งถูกแบ่งเป็น 7 Blockย่อย เมื่อต้องการจะแบ่ง Page ให้เป็น Block ได้อย่างลงตัว ซึ่งBlockมีขนาด 3x3 Page จึงถูกกำหนดให้มีขนาดเท่ากับ 2^n+1; $0<n<8$ ซึ่งขนาด Page ใหญ่ที่สุดคือ 129x129 สามารถแทนด้วย Block เพียงอันเดียวก็ได้ซึ่งก็คือที่ตำแหน่ง 0, 64, 128 ทั้งแนวตั้งและแนวนอน การแบ่งจะทำทีละ 4 ซึ่งเห็นได้ชัดรูปแบบของ Quadtree โดย Block ที่กล่าวไปข้างต้นเป็น Root (Stride = 64) และแบ่งต่อมาเรื่อยๆ (Stride = Stride/2) จนถึงขั้น

ย่อยสุด คือใช้ทุก Vertex (Stride = 1) จากนั้นพิจารณาแต่ละ Block โดยใช้ Depth First Search เริ่มคำนวณขนาด Bounding Box ของแต่และ Block ไปจนถึง leaf node ให้พิจารณาว่า Pixel Interval มีค่ามากถึงระดับหนึ่ง (Threshold) หรือไม่ถ้าถึงแสดงว่า ต้องการความละเอียดในระดับนี้ หากไม่ถึงให้พิจารณา Sibling node ดูว่าทุกตัวมีลักษณะเดียวกันหรือไม่ ถ้าใช่ให้ใช้ Parent Block นั่นเอง แต่ละ Block สามารถมีรูปแบบต่างๆกันขึ้นอยู่กับความเหมาะสมดังรูปต่อไปนี้



รูปที่ 3.29 Block Detail

จะเห็นได้ว่าVertex บน Edge และ Center ของ Block สามารถ Enable หรือ Disable ได้ เพื่อให้เกิดเป็น Level of Detail Mesh ได้ดังรูป

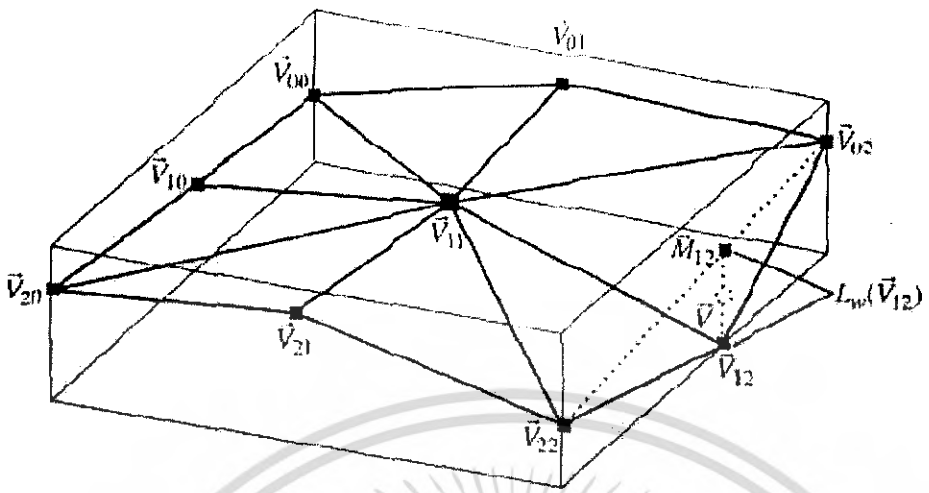


รูปที่ 3.30 Vertex Dependency

จากภาพด้านบนเห็นได้ว่าหาก Vertex หนึ่งหนึ่ง Enable จะต้องมี Vertex จำนวนหนึ่ง Enable ด้วย (Vertex Dependency) ซึ่งการทำเช่นนี้เรียกว่า Vertex Simplification

เมื่อแต่ละ Page มี Level of Detail ที่ไม่เท่ากันจึงต้องมีการปรับแก้โดยการสร้างและกำหนด Vertex Dependency เมื่อนำ Page มาเชื่อมต่อกัน (Page Stitching) อีกที เมื่อเชื่อมต่อกันได้

อย่างกลมกลืนแล้ว จะทำ Tessellation (Create Index) เป็นขั้นตอนสุดท้าย เพื่อส่งวัตถุเข้าสู่ Graphic pipeline ต่อไป

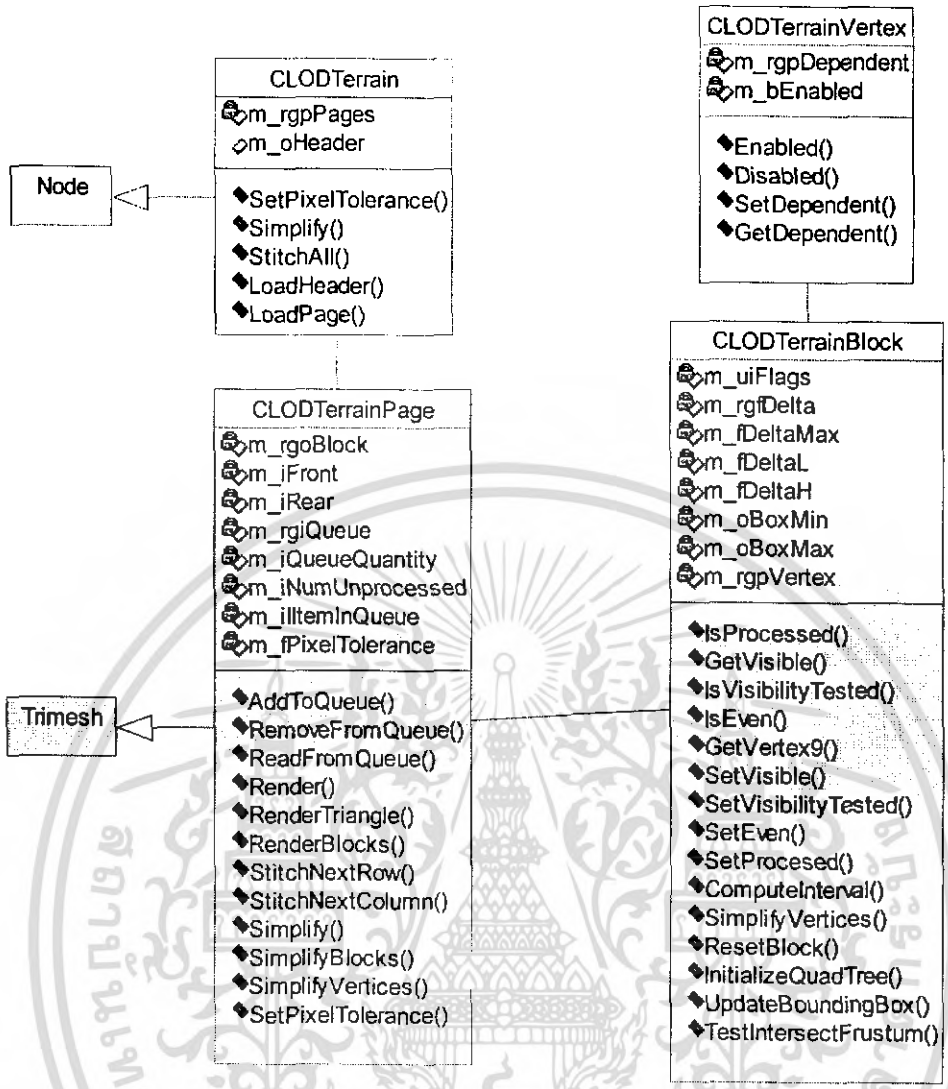


รูปที่ 3.31 Even Block

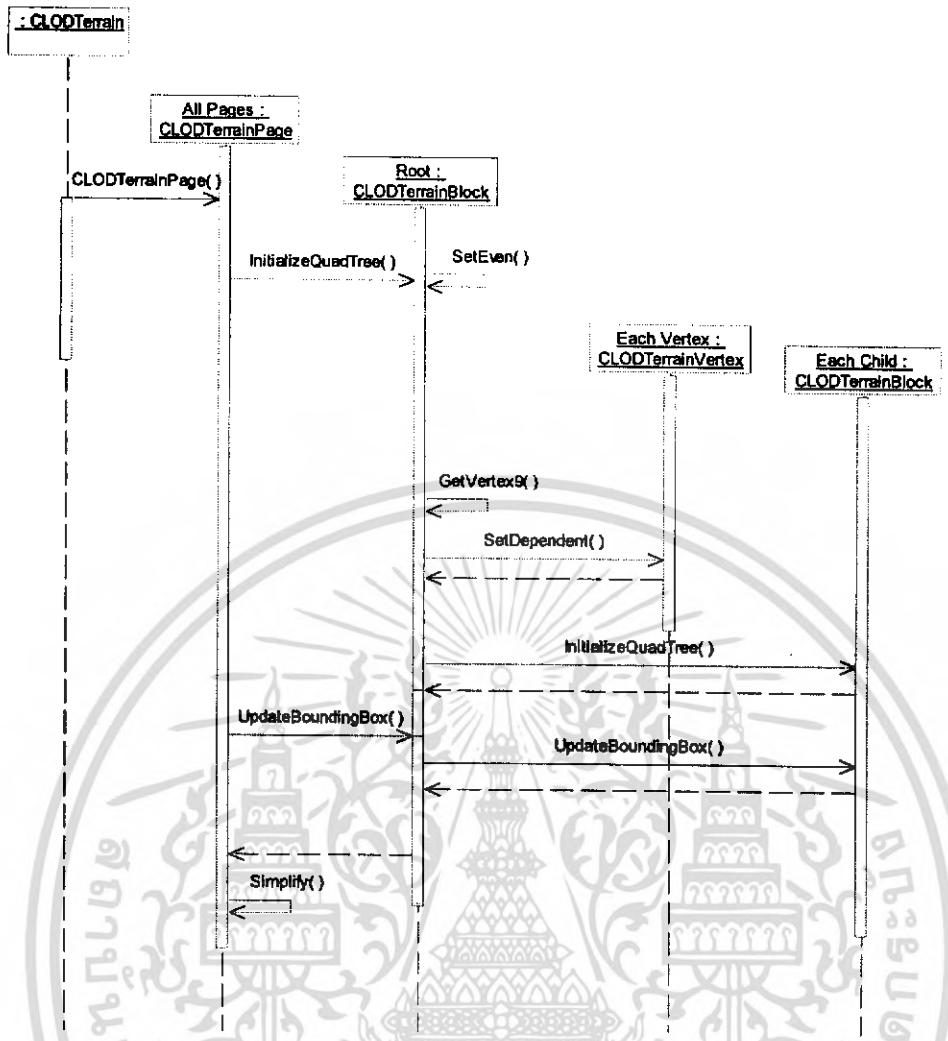
ในการทำ Block Based Simplification ภายหลังจากได้ Quad-tree ของ Block ทั้งหมด และใส่ Root Block เข้าไปใน Queue แล้วกระบวนการก็จะเริ่มจากทำการโปรเซส Block ที่เหลือใน Queue โดยตรวจดูว่าเป็นลูกตัวแรกหรือไม่ ในกรณีของ Root Node ไม่ใช่ จึงตรวจสอบต่อว่าอินเตอร์เซคกับ View Frustum หรือไม่ หากใช่จะตรวจสอบต่อว่าขนาดในปัจจุบันยังสามารถแบ่งต่อได้อีกหรือไม่ ก็จะทำการคำนวณค่าPixel Interval ของลูกแต่ละตัว หากมีตัวใดตัวหนึ่งที่มีค่า Interval สูงกว่าที่กำหนดก็จะทำการใส่ลูกทั้งหมดเข้าไปใน Queue และเริ่มตรวจใหม่เข้าในกรณีของลูกตัวแรกโดยหาก Nodeในชั้นเดียวกันมีค่า Interval ต่ำกว่าที่กำหนดทั้งหมดก็จะทำการแทนค่าด้วย Parent ในกรณีนี้จะไม่เข้าเงื่อนไขจึงทำการ Sub divide ต่อไปอีกอย่างเช่นที่เกิดกับ Root และสิ้นสุดกระบวนการ (จำนวน Block ใน Queue ที่ยังไม่ได้ Process เป็น 0)

ในการทำ Vertex Based Simplification ภายหลังจากทำ Block Based Simplification แล้วจะมี Block ที่ใช้งานอยู่จริงใน Queue จะกระทำการต่อไปใน Block ที่มองเห็นได้ ดังในภาพที่ 3.31 เมื่อ V12 เป็น Candidate Vertex (เช่นเดียวกันกับ V01, V10 และ V21) ว่าจะสามารถ Disable ได้หรือไม่โดยการแปลงระยะ (M12, V12) เป็นระยะบน Screen Space (Lw) และเทียบ Lw ว่ามากกว่า Pixel Tolerance หรือไม่หากมากกว่าก็จะ Enable Vertex นั้นและส่งผลกับ Dependent Vertex ด้วย

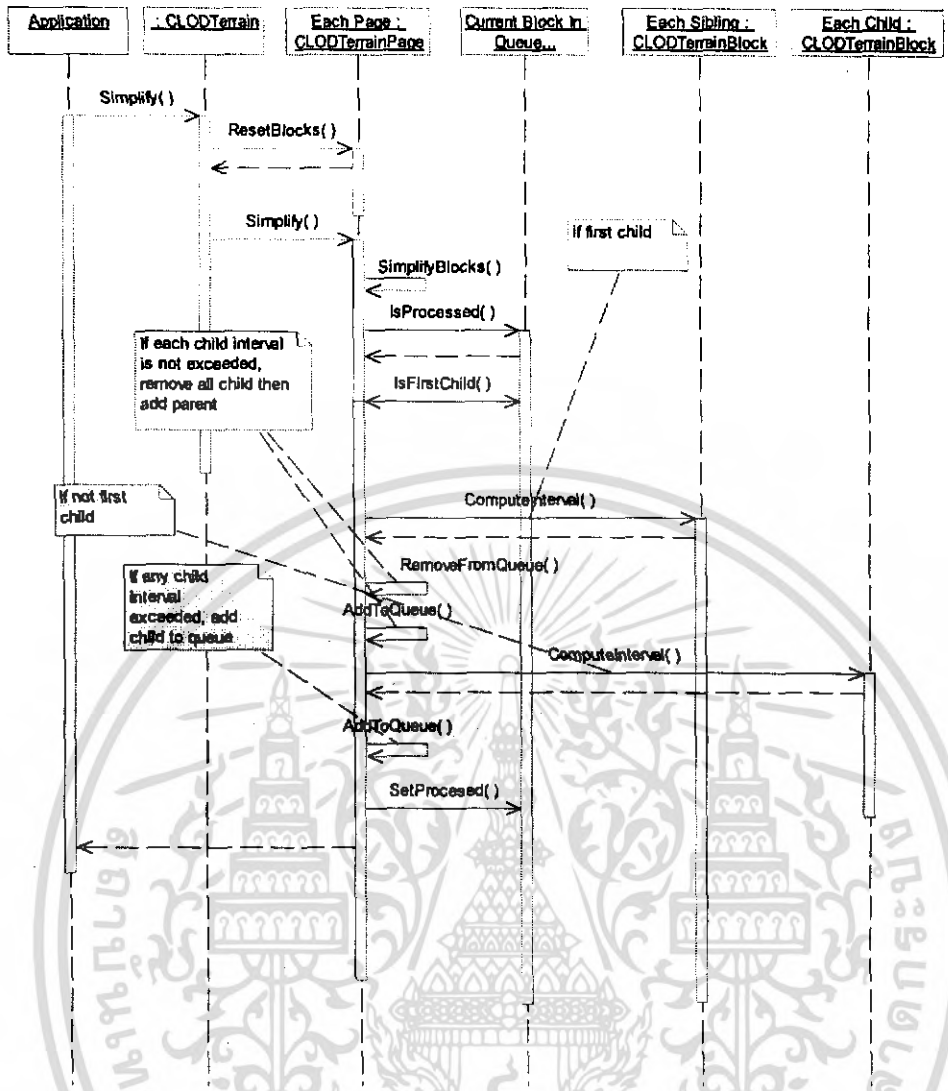
ส่วนในขั้นตอนของการ Tessellation นั้นจะทำการประมวล Block ทั้งหมดที่อยู่ใน Queue ในแต่ละ Block จะถูกวาดจาก 2 Triangle ในลักษณะ Recursive คือหาก Triangle ใดที่สามารถแบ่งต่อได้ และVertexที่จะใช้แบ่งอยู่ในสถานะ Enabled ก็จะทำการเรียก RenderTriangle ต่อๆไปจนกระทั่งแบ่งต่อไม่ได้ แล้วจึงสร้าง Index ให้กับการแสดงผล



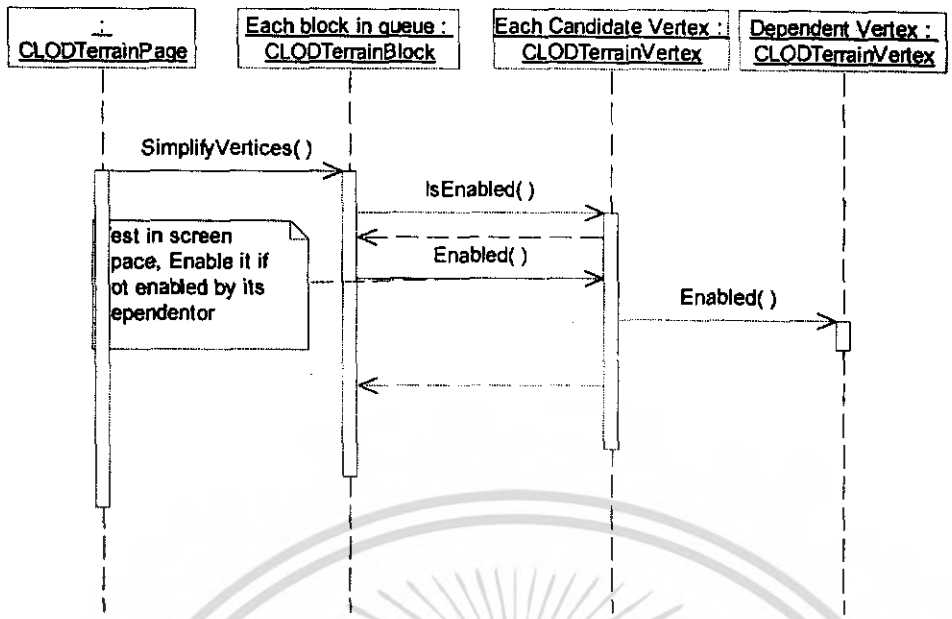
รูปที่ 3.32 CLOD Terrain



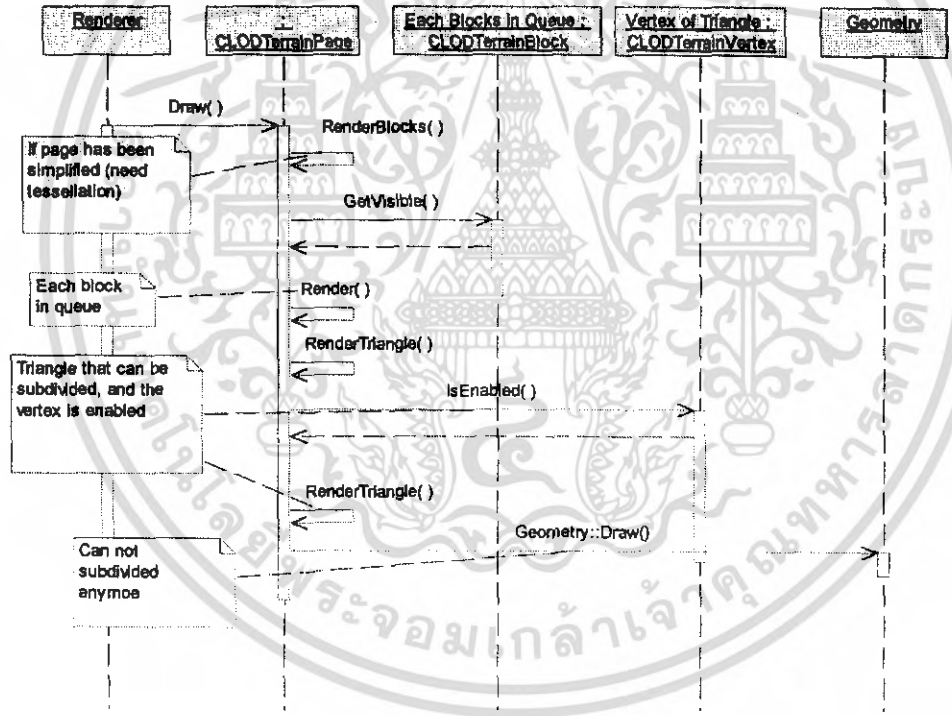
รูปที่ 3.33 Sequence diagram: Initialize



รูปที่ 3.34 Sequence diagram: Block based simplification



รูปที่ 3.35 Sequence diagram: Vertex Simplification



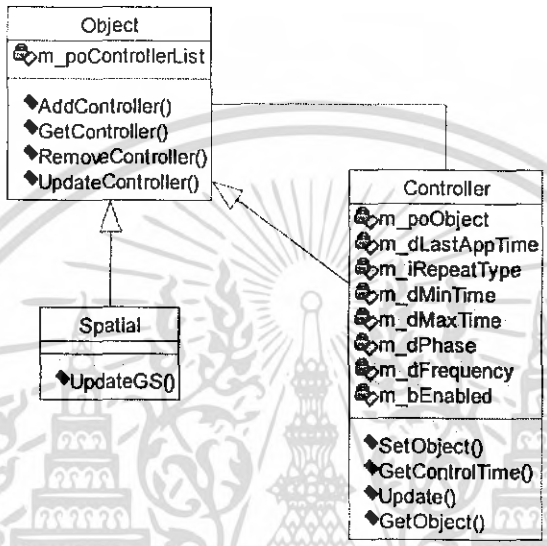
รูปที่ 3.36 Sequence diagram: Page Drawing

3.3.9 Controller

Animation หมายถึงการเปลี่ยนแปลงคุณสมบัติของวัตถุเมื่อเวลาเปลี่ยนแปลงไป ดังนั้นใน PC Engine จึงไม่ได้กำหนดให้การเปลี่ยนแปลงเหล่านี้สามารถเกิดขึ้นได้กับ Geometry เท่านั้น แต่ยังสามารถเกิดกับวัตถุชนิดต่างๆได้ โดยทำการกำหนดคลาส Controller ขึ้นมาควบคุมการเปลี่ยนแปลงเชิงเวลาที่เกิดขึ้นของวัตถุ โดยรับค่าเวลาผ่านทางการ Update Geometry State

ของ Scene graph เพื่อที่จะมาปรับปรุงค่าต่างๆของ Object โดยการนำไปใช้ประโยชน์ที่เห็นได้ชัดเจนที่สุดคือการนำไปทำภาพเคลื่อนไหวนั่นเอง

ส่วนของคลาส Controller มีหน้าที่เพียงดูแล Controller Time เท่านั้น โดยจะมีวิธีการวนรอบแบบต่างๆเช่น การวนกลับมาจุดแรกใหม่(Repeat) การย้อนกลับมาจนถึงจุดแรกเมื่อถึงจุดจบ(Cycle) หรือการจบเลย(Clamp) อย่างไรก็ตามในส่วนของเมธอด UpdateWorldData หรือ UpdateGS ในคลาส Spatial ก็ต้องทำการเรียก UpdateController ด้วยเพื่อให้นำค่าเวลาในขณะนั้นไปแปลงเป็นค่าเวลาเฉพาะของ Controller

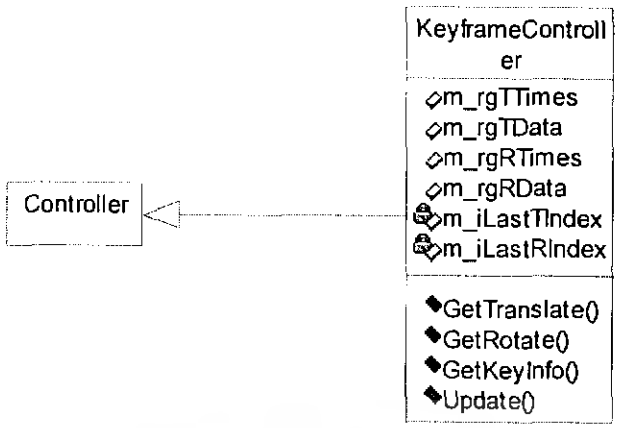


รูปที่ 3.37 Controller Class

3.3.10 Keyframe Animation

คือการกำหนดการเคลื่อนไหวของวัตถุในแต่ละ Shot ที่สำคัญออกมา (คล้ายกับการวาดการ์ตูนเคลื่อนไหว) โดยสามารถกำหนด Local Transformation ของวัตถุในแต่ละ frame ได้ โดยกำหนดเวลาที่จะใช้ของแต่ละ frame ไปด้วย หากเวลาที่ระบุโดย Application มีค่าระหว่างช่วงเวลาที่กำหนด ก็จะทำการ Linear Interpolate ค่า Transformation ระหว่างช่วงเวลานั้นและทำการ Apply ให้กับวัตถุ

Transformation ที่สนับสนุนใน Keyframe ได้แก่ Translation ซึ่งอยู่ในรูปของ Vector และ Rotation ซึ่งอยู่ในรูปของ Quaternion



รูปที่ 3.38 Keyframe Controller

3.3.11 Skin Controller

จากข้อเสียของ Keyframe Animation คือต้องเก็บ Transformation ของแต่ละ Vertices ในวัตถุ หากนำไปใช้ในโมเดลรูปคนซึ่งมีจำนวน Vertices มากๆ ก็จะต้องเปลืองทรัพยากรในการเก็บ และการ Apply Transformation มาก จึงได้นำแนวคิดของ Bone มาใช้โดยจะทำ Keyframe animation เฉพาะส่วนที่เป็น Boneของวัตถุเท่านั้น และใช้ความสัมพันธ์ระหว่าง Vertex กับ Bone เป็นตัวกำหนด Transformation ของแต่ละ vertex อีกทีความสัมพันธ์มีดังต่อไปนี้

- B : Bone(s) ที่มีผลต่อ Vertex
- W : Weight ของ Bone(s) ที่มีผลต่อ Vertex
- O : Offset ของ Vertex จาก Bone(s)

ยกตัวอย่างตารางความสัมพันธ์

Vertex	Bone0	Bone1	Bone2	Bone3
V0	W_{00},O_{00}	-	W_{02},O_{02}	W_{03},O_{03}
V1	-	W_{11},O_{11}	-	W_{13},O_{13}
V2	W_{20},O_{20}	-	-	W_{23},O_{23}

ตารางที่ 3.5 ตัวอย่าง Bone Offset และ Weight

เมื่อทำการ Update ความสัมพันธ์จะได้สมการดังต่อไปนี้

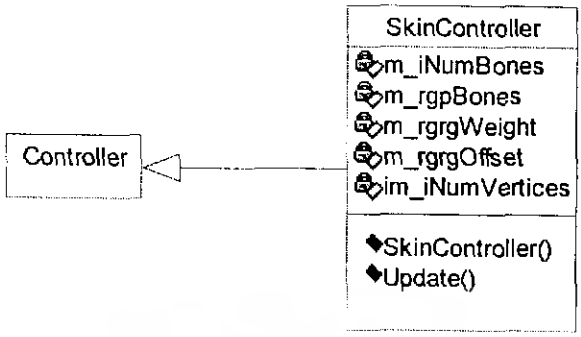
$$V0 = W_{00}(S_0R_0O_{00} + T_0) + W_{02}(S_2R_2O_{02} + T_2) + W_{03}(S_3R_3O_{03} + T_3)$$

$$V1 = W_{11}(S_1R_1O_{11} + T_1) + W_{13}(S_3R_3O_{13} + T_3)$$

$$V2 = W_{20}(S_2R_2O_{20} + T_2) + W_{23}(S_3R_3O_{23} + T_3) \qquad (3.16)$$

Skin Controller จึงไม่ได้ยุ่งเกี่ยวกับทางด้านเวลาแต่จะทำการปรับ Vertex Data ให้อยู่ในตำแหน่งที่กำหนดจาก Offset และ Weight จึงต้องทำการแก้ไข World Transformation ให้เป็น

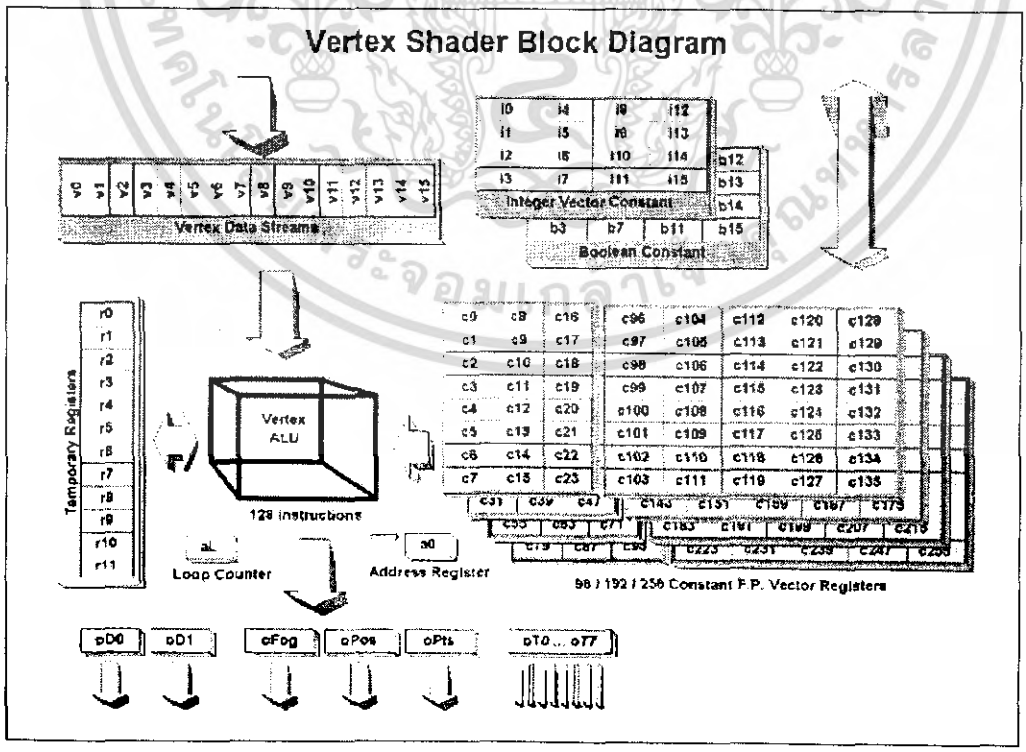
Identity และกำหนดไม่ให้เปลี่ยนแปลง ซึ่งต้องปรับปรุง Class Spatial เสียก่อนเพื่อสามารถบอกได้ว่าไม่ต้องทำการ Apply World Transformation ให้

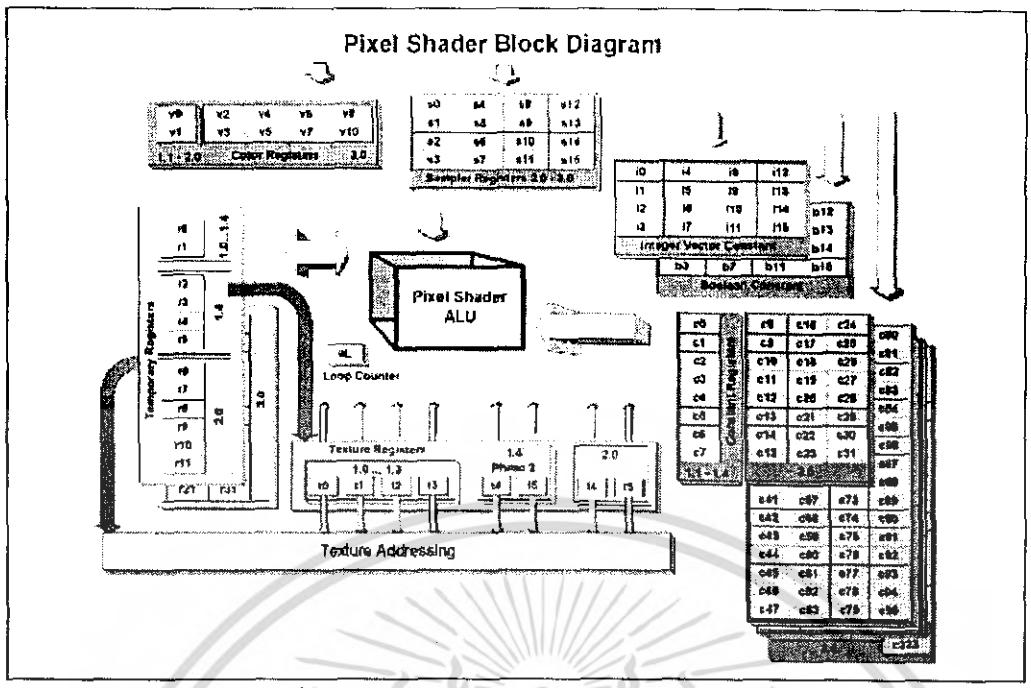


รูปที่ 3.39 Skin Controller

3.3.12 Shader

ใน 3D Pipeline ช่วงของกระบวนการ Transform & Lighting และTexturing นั้นจะเปิดโอกาสให้ผู้ใช้สามารถทำการ Program เองได้ด้วย Vertex/Pixel Shader เนื่องจากการใช้ Vertex หรือ Pixel Shader ไม่สามารถใช้ Drawing Function เดิม (DrawPrimitive) ได้เนื่องจากต้องติดตั้ง Shader Program ให้กับ Derived Renderer จึงจัดให้ Shader เป็น Effect ชนิดหนึ่ง และเพิ่ม DrawShader เป็น Drawing Function ขึ้นใน Renderer โดยเพิ่มส่วนของการติดตั้ง Shader Program และใน ShaderEffect ก็จะประกอบไปด้วย VertexShader และ PixelShader ซึ่งสามารถทำ Generalize ได้เป็น Class Shader ที่จะประกอบไปด้วยส่วน Code ของโปรแกรม และลิสต์ของ Shader Constants ต่างๆ





รูปที่ 3.40 Vertex/Pixel Shader Block Diagram

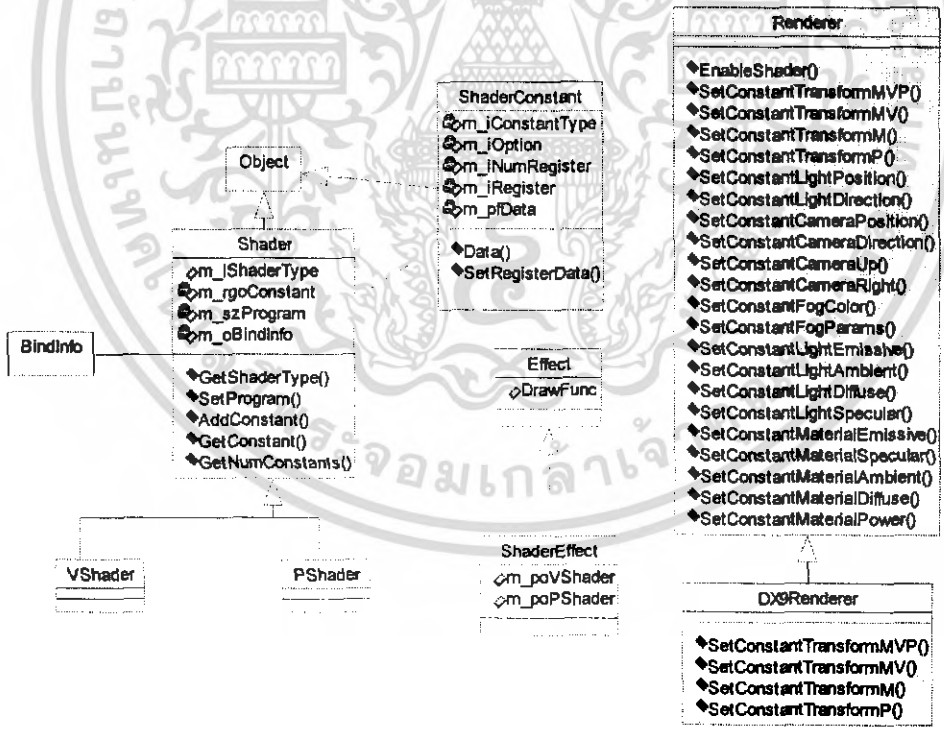
ส่วนของ Shader Constants ในการใช้ Vertex และ Pixel Shader จะต้องมีการกำหนดค่าให้ Constant Register (c) เพื่อนำไปใช้ในโปรแกรมอย่างเหมาะสม เพื่อความสะดวกของผู้ใช้งานค่าโดยทั่วไปที่จะทำการตั้งให้กับ Constant Register เช่น World-View-Projection Matrix ก็เป็นค่าที่มีการเปลี่ยนแปลงอยู่ตลอดเวลา และเป็นสิ่งที่ Renderer รู้อยู่แล้วจึงกำหนดให้ Renderer ตรวจประเภทของ Constant ที่ติดตั้งไว้และทำการกำหนดค่าให้ตามความเหมาะสมดังตารางที่ 3.6 โดยในกรณีที่เป็นMatrix จะต้องสามารถเลือกได้ว่าจะให้อยู่ในรูปแบบใดคือ Inverse Transpose หรือ Inverse และ Transpose ส่วนในกรณีที่เกี่ยวกับแสงก็จะต้องระบุลำดับที่ด้วย

ประเภทของ Constant	คำอธิบาย
TRANSFORM_M	Model Transformation
TRANSFORM_P	Projection Transformation
TRANSFORM_MV	Model-View Transformation
TRANSFORM_MVP	Model-View-Projection Transformation
CAMERA_POSITION	ตำแหน่งกล้อง
CAMERA_DIRECTION	ทิศทางกล้อง
CAMERA_UP	ทิศทางด้านบนของกล้อง
CAMERA_RIGHT	ทิศทางด้านขวาของกล้อง
FOG_COLOR	สีของหมอก
FOG_PARAMS	(start, end, density, enabled)
MATERIAL_EMISSIVE	Material Emissive

MATERIAL_AMBIENT	Material Ambient
MATERIAL_DIFFUSE	Material Diffuse
MATERIAL_SPECULAR	Material Specular
MATERIAL_SHININESS	Material Shininess
LIGHT_POSITION	ตำแหน่งของแสง
LIGHT_DIRECTION	ทิศทางของแสง
LIGHT_AMBIENT	แสง Ambient
LIGHT_DIFFUSE	แสง Diffuse
LIGHT_SPECULAR	แสง Specular
LIGHT_ATTENPARAMS	ค่า Attenuate ของแสง
NUMERICAL_CONSTANT	ค่าคงที่ตัวเลข
USER_DEFINED	ค่าที่ User กำหนดเอง

ตารางที่ 3.6 Shader Constant มาตรฐาน

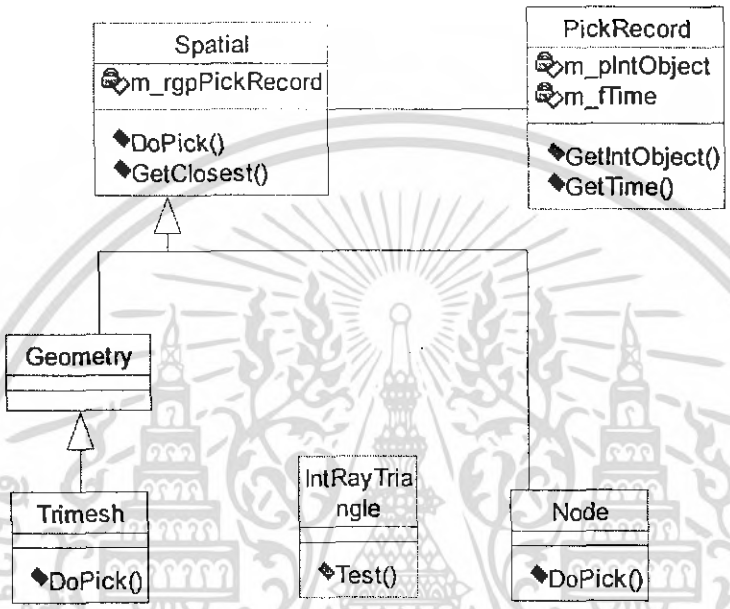
เช่นเดียวกันกับ Texture เพื่อเพิ่มประสิทธิภาพในการจัดการหน่วยความจำ Shader Program ใดที่ถูกคอมไพล์และติดตั้งแล้วก็ไม่จำเป็นต้องคอมไพล์และติดตั้งอีก จึงเพิ่ม BindInfo ให้กับคลาส Shader



รูปที่ 3.41 Shader Support

3.3.13 Detailed Picking

ภายหลังจากการสร้าง Picking Ray จากคลาส Camera แล้ว ผู้ใช้สามารถที่จะทำ Picking แบบหยาบได้โดยตรวจสอบกับ Bounding Volume ของแต่ละวัตถุโดยตรง แต่จากคุณลักษณะของ Scene Graph สามารถทำ Picking แบบลำดับขั้นได้ โดยใช้คุณสมบัติ Virtual Function ที่จะเลือกการกระทำได้ ได้แก่ถ้าเป็น Node ก็จะถ่ายทอดคำสั่งต่อไปให้ลูก และถ้าเป็น Trimesh ก็จะต้องทำการตรวจกับ Bounding Volume ก่อนแล้วจึงตรวจสอบในระดับ Triangle ผลของการทำ Picking คือ Pick Record Array ซึ่งประกอบไปด้วยวัตถุที่ตัด และค่าของเวลาที่ Ray ตัด



รูปที่ 3.42 Picking Classes

3.4 Physically Base Simulation Module

หลักการออกแบบส่วนของ Physically Base Simulation Module ประกอบไปด้วยส่วนหลักสามส่วนด้วยกันคือ

1. ส่วนตรวจหาการชนกันของวัตถุ(Collision Detection)
2. ส่วนคำนวณการเคลื่อนที่ของวัตถุ (Rigidbody System)
3. ส่วนหา ปฏิสัมพันธ์ของวัตถุหลังการชนกันของวัตถุ(Collision Response)

โดยทั้งสามส่วนจำเป็นที่จะต้องพึ่งพาอาศัยกัน ในทุกครั้งที่มีการเคลื่อนที่จะมีการตรวจสอบการชนกันของวัตถุ และเมื่อชนกันก็จะส่งวัตถุที่ชนกันนั้นไป ส่วน ปฏิสัมพันธ์ เพื่อคำนวณหาทิศทางใหม่ของวัตถุที่ชนกันต่อไป โดยในส่วน คำนวณการเคลื่อนที่กับส่วนหาปฏิสัมพันธ์หลังการชน เนื่องจากว่า ส่วนปฏิสัมพันธ์นั้นจะรับค่าเป็นวัตถุแข็งแกร่ง ที่ชนกันการการตรวจสอบ การชนนั้นเอง

3.4.1 ส่วนตรวจหาการชนกันของวัตถุ (Collision Detection)

ใช้หลักการในการหาการซ้อนทับกันของวัตถุสองชิ้น โดยตรวจสอบจากข้อมูลของวัตถุทั้งสองซึ่งองค์ประกอบหลักของส่วนนี้จะถูกออกแบบให้เป็น คลาส ในการออกแบบคลาสนี้ ได้คำนึงถึงวัตถุที่จะมาหาว่าชนกันนั้นมีทั้งแบบที่เคลื่อนที่และแบบที่หยุดนิ่งดังนั้นเราจึงจำแนกการตรวจจับการชนออกมาเป็น การตรวจจับการชนของวัตถุที่เคลื่อนที่ (มีความเร็ว) และการตรวจจับการชนของวัตถุที่หยุดนิ่ง (ไม่มีความเร็ว) และเพื่อประสิทธิภาพเอนจินนี้ได้ทำการแยกการทดสอบและการหาตำแหน่งที่ชน ออกจากกันเพื่อบางทีอาจต้องการรู้ว่าชนกับไม่ชนจะได้ไม่ต้องไปเข้าสมการคณิตศาสตร์เพื่อหาจุดของตำแหน่งที่ชน ดังนั้นคลาสทุกคลาสก็จะมีฟังชั่นที่เหมือนๆ กันและตัวแปลบางตัวแปลที่เหมือนกัน เราจึงได้สร้างคลาสต้นแบบขึ้นเพื่อให้คลาสต่างๆมา derive จากคลาสนี้

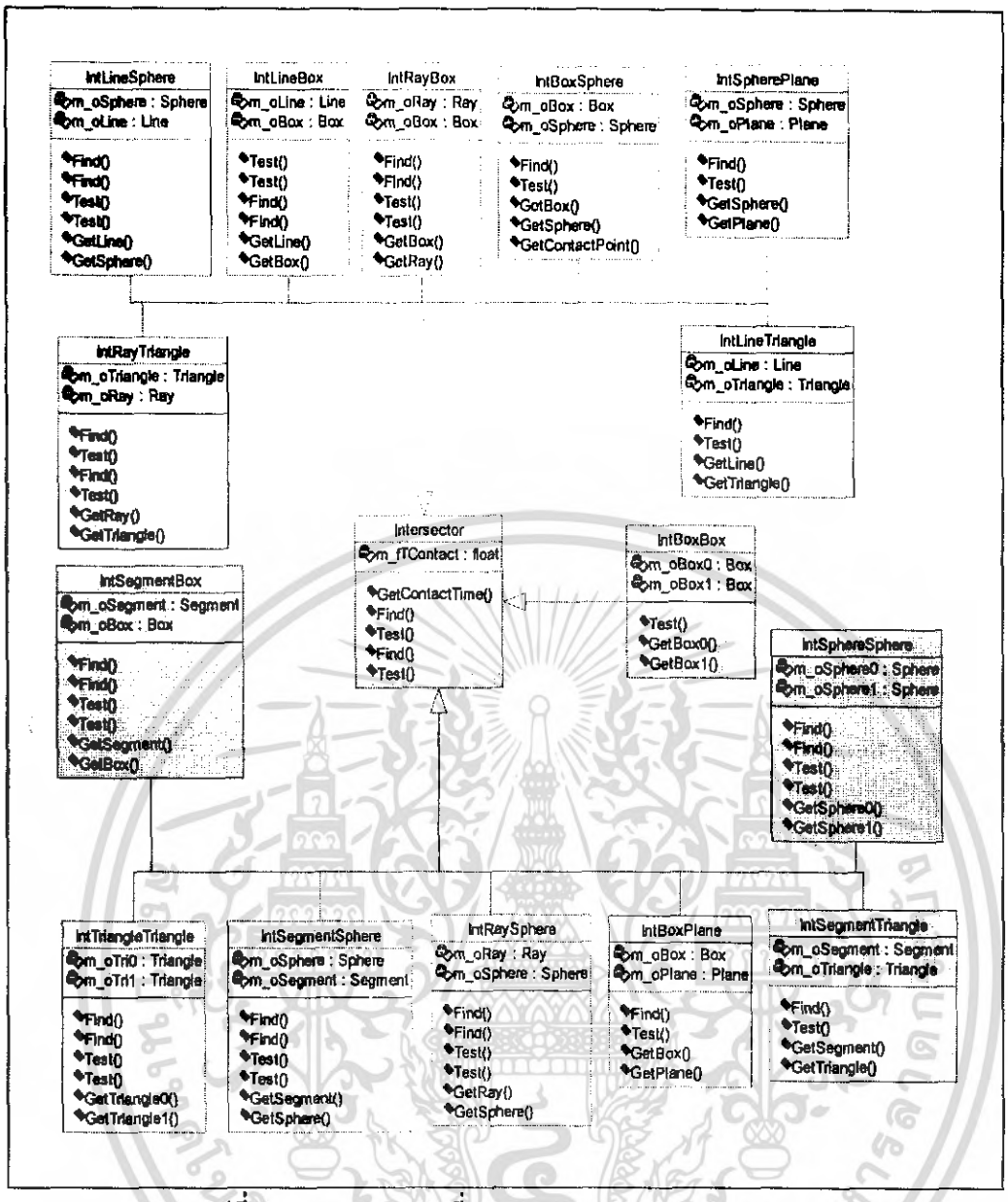
```
Class Intersector
{
public:
        virtual ~Intersector();

        //static intersection queries
        virtual bool Test();
        virtual bool Find();

        //dynamic intersection queries
        virtual bool Test(const float fTMax,const Vector3& vVelocity0,const Vector3& vVelocity1);
        virtual bool Find(const float fTMax,const Vector3& vVelocity0,const Vector3& vVelocity1);

        //time at which two object are in first constant for dynemic intersection
        float GetContactTime() const;

protected:
        Intersector();
        float m_fContactTime;
};
```

จากโค้ด จะเห็นได้ว่า ได้ออกแบบให้ มีฟังก์ชั่น Test() และ Find () สองลักษณะด้วยกันคือ แบบไม่มีการเคลื่อนที่ของวัตถุกับแบบที่มีการเคลื่อนที่ของวัตถุโดยถ้าเป็นแบบ

ที่มีการเคลื่อนที่ของวัตถุจะให้ใส่ค่าของความเร็วของวัตถุทั้งสองและเวลาในส่วนของ ฟังก์ชั่น Test() เป็นการตรวจว่ามีการชนกันหรือไม่ ถ้ามีการชนจะคืนค่า true ถ้าไม่มีการชนจะคืน false โดยไม่มีการเก็บค่าใดๆเกี่ยวกับการชนที่เกิดขึ้น และ ในส่วนของ ฟังก์ชั่น Find() ถ้ามีการชนจะมีการเก็บเวลาที่ชน และค่าต่างๆ ที่เกี่ยวกับการชนโดยค่าต่างๆนั้นขึ้นอยู่กับว่าเป็นการชนระหว่าง วัตถุประเภทใด โดยคลาสที่ Derive มาจากคลาส Intersector ประกอบไปด้วย

1. การตรวจสอบการชนระหว่างเส้นตรงกับสามเหลี่ยม
2. การตรวจสอบการชนระหว่างเรย์ กับสามเหลี่ยม
3. การตรวจสอบการชนระหว่างส่วนของเส้นตรงกับสามเหลี่ยม
4. การตรวจสอบการชนระหว่างเส้นตรงกับทรงกลม
5. การตรวจสอบการชนระหว่างเรย์ กับทรงกลม
6. การตรวจสอบการชนระหว่างส่วนขอเส้นตรงกับทรงกลม
7. การตรวจสอบการชนระหว่างเส้นตรงกับ สี่เหลี่ยม
8. การตรวจสอบการชนระหว่างเรย์ กับ สี่เหลี่ยม
9. การตรวจสอบการชนระหว่างส่วนของเส้นตรงกับสี่เหลี่ยม
10. การตรวจสอบการชนระหว่างสามเหลี่ยมกับสามเหลี่ยม
11. การตรวจสอบการชนระหว่างทรงกลมกับทรงกลม
12. การตรวจสอบการชนระหว่างสี่เหลี่ยมกับสี่เหลี่ยม
13. การตรวจสอบการชนระหว่างสี่เหลี่ยมกับทรงกลม
14. การตรวจสอบการชนระหว่างสี่เหลี่ยมกับระนาบ
15. การตรวจสอบการชนระหว่างทรงกลมกับระนาบ

รูปที่ 3.43 เป็นตัวอย่างการออกแบบคลาสที่ใช้ในการตรวจสอบการชนที่ Derive มาจาก คลาส Intersector()



รูปที่ 3.43 แสดงคลาสที่ Derive มาจาก คลาส Intersector

ในการตรวจสอบวัตถุที่ชนกันนั้นสามารถทำได้ละเอียดสุดในระดับ สามเหลี่ยมกับสามเหลี่ยม โดยอาศัยการสร้าง Bounding Volume Tree มาช่วยในการเพิ่มประสิทธิภาพในการตรวจสอบ ซึ่งจะมีการเก็บสามเหลี่ยมที่เป็นองค์ประกอบของวัตถุให้อยู่ในรูปของ Tree เพื่อลดการวนรอบในการตรวจสอบระหว่างสามเหลี่ยมกับสามเหลี่ยม โดยมีคลาสที่ทำหน้าที่รับผิดชอบคือ Record ซึ่ง Record นี้จะเป็นคลาสที่ทำการเก็บ สมเหลี่ยมทั้งหมดของวัตถุนั้นๆ องค์ประกอบของคลาสมีดังนี้

```
Class CollisionRecord
{
Public:
        typedef void (*Callback) (CollisionRecord& roRecord0, int iT0,
    CollisionRecord& roRecord1, int iT1,Intersector* pIntersector);

        CollisionRecord(Trimesh* pTrimesh, BoundingVolumeTree* pTree,
                        Vector3* pvVelocity, Callback oCallback,void* poCallbackData);

        ~CollisionRecord();

        Trimesh* GetMesh();
        Vector3* GetVelocity();
        void* GetCallbackData();

        void TestIntersection (CollisionRecord& roRecord);
        void FindIntersection (CollisionRecord& roRecord);
        void TestIntersection (float fTMax, CollisionRecord& roRecord);
        void FindIntersection (float fTMax ,CollisionRecord& roRecord);
protected:
        Trimesh* m_pTrimesh;
        BoundingVolumeTree* m_pTree;
        Vector3* m_pvVelocity;
        Callback m_oCallback;
        void* m_pvCallbackData;
}
```

หลักการตรวจสอบก็คือ จะทำการตรวจสอบสามเหลี่ยมทุกสามเหลี่ยมระหว่างสองวัตถุและเมื่อพบว่าชนกันก็จะเรียก CallBack ฟังก์ชั่น โดยเราสามารถกำหนดฟังก์ชั่นนี้ได้ว่าต้องการให้ทำอะไรกับ วัตถุรีสามเหลี่ยมนั้น เนื่องจากการตรวจสอบแบบนี้จะใช้ความสามารถของเครื่องสูงเราจึงมีการ นำเทคนิคมาใช้ในการตรวจสอบการชนคือการแบ่งกลุ่มการตรวจสอบการชนกับระหว่างวัตถุ โดยอาศัยหลักการที่ว่า ถ้าวัตถุที่ไม่สามารถจะเคลื่อนที่มาชนกันได้เราก็จะไม่ทำการตรวจสอบการชน โดยการจัดกลุ่มของวัตถุที่สามารถเคลื่อนที่มาชนกันได้เป็นกลุ่มๆ แล้วทำการตรวจสอบการชนกันเฉพาะภายในกลุ่มเท่านั้น เป็นการช่วยเพิ่มประสิทธิภาพในการตรวจสอบการชนซึ่งเราได้ทำการออกแบบคลาส กลุ่มของวัตถุที่มีความเป็นไปได้ที่จะชนกันขึ้นโดยรูปแบบของคลาสเป็นดังนี้

```
Class CollisionGroup
{
public:
        CollisionGroup();
        ~CollisionGroup();
        bool Add(CollisionRecord* pRecord);
        bool Remove(CollisionRecord* pRecord);
        void TestIntersection();
        void FindIntersection();
        void TestIntersection (float fTMax);
        void FindIntersection (float fTMax);
protected:
        Array<CollisionRecord*> m_prgRecord;
}
```

ภายในฟังก์ชั่น TestIntersect นั้นจะทำการวนตรวจสอบระหว่างวัตถุทั้งหมดที่อยู่ในกลุ่มนี้ โดยการเพิ่มวัตถุ จะทำผ่านฟังก์ชั่น Add และการเอาออกก็จะผ่านฟังชั่น Remove ซึ่งวัตถุที่นำเข้าไปนั้นต้องเป็น Collision Record เพราะว่าเป็นคลาสที่สร้างขึ้นมาเพื่อสนับสนุน Collision Record นั้นเอง

3.4.2. ส่วนคำนวณการเคลื่อนที่ของวัตถุ (Rigid Body Simulation)

ในส่วนนี้จะเป็นการสร้างวัตถุแข็ง โดยจะจำลองโดยมีเก็บค่าของ ตำแหน่ง ความเร็วเชิงเส้น ความเร็ว เชิงมุม โมเมนตั้มเชิงเส้น โมเมนตั้มเชิงมุม น้ำหนัก ค่าความเฉื่อยของวัตถุ แรงธรรมชาติ ทั้งในเชิงเส้นและเชิงมุม แล้วนำค่าเล่านี้มาคำนวณตำแหน่งของวัตถุ ณ เวลาถัดไปร่วมไปถึง ลักษณะการเคลื่อนที่ของวัตถุด้วย ซึ่งคลาสนี้จะมีรูปแบบดังนี้

```
Class RigidBody
{
public:
        RigidBody();
        virtual ~RigidBody();

        Vector3& Position();

        void ComputeMassProperties (const Vector3* rgvVertex,int iTQuantity,const unsigned int*
rgiIndex,Vector3& rvCenter);
        //set rigid body state
```

```
void SetMass (float fMass);
void SetBodyInertia (const Matrix3& roInertia);
void SetPosition (const Vector3& roPos);
void SetQOrientation (const Quaternion& roQOrient);
void SetLinearMomentum (const Vector3& roLinMom);
void SetAngularMomentum (const Vector3& roAngMom);
void SetROrientation (const Matrix3& roROrient);
void SetLinearVelocity (const Vector3& roLinVel);
void SetAngularVelocity (const Vector3& roAngVel);

//get rigid body state
float GetMass () const;
float GetInverseMass () const;
const Matrix3& GetBodyInertia () const;
const Matrix3& GetBodyInverseInertia ()const;
Matrix3 GetWorldInertia ()const;
Matrix3 GetWorldInverseInertia ()const;
const Vector3& GetPosition() const;
const Quaternion& GetQOrientation() const;
const Vector3& GetLinearMomentum() const;
const Vector3& GetAngularMomentum() const;
const Matrix3& GetROrientation () const;
const Vector3& GetLinearVelocity() const;
const Vector3& GetAngularVelocity() const;

typedef Vector3 (*Function)
(
        float,          // time of application
        float,          // mass
        const Vector3&,   // position
        const Quaternion&, // orientation
        const Vector3&,   // linear momentum
        const Vector3&,   // angular momentum
        const Matrix3&,   // orientation
        const Vector3&,   // linear velocity
        const Vector3&    // angular velocity
);
```

```
        Function Force;
        Function Torque;

        void Update (float fT,float fDT);
protected:
        float m_fMass,m_fInvMass;
        Matrix3 m_omatrixInertia,m_omatrixInvInertia;

        Vector3 m_vPos;
        Vector3 m_vLinMom;
        Vector3 m_vAngMom;
        Quaternion m_vQOrient;

        Matrix3 m_omatrixROrient;
        Vector3 m_vLinVel;
        Vector3 m_vAngVel;
};
```

จะเห็นว่ามีการตั้งค่าให้กับตัวแปรต่างๆ ให้กับวัตถุซึ่งค่าบางอย่างจำเป็นต้องกำหนดให้กับวัตถุไม่มีการตั้งค่าเริ่มต้นอาจทำให้ไม่สามารถนำไปคำนวณการเคลื่อนที่ของวัตถุได้ และนอกจากเราสามารถตั้งค่าเหล่านั้นได้แล้วเรายังสามารถนำค่าต่างๆออกมาใช้ได้ด้วย ส่วนในการใช้งาน ต้องมีการสร้าง ฟังก์ชั่น Force และ ฟังก์ชั่น Torque ให้กับ RigidBody เพื่อนำไปใช้ในการคำนวณตำแหน่งและตัวแปลต่างๆ ณ. เวลาถัดไป โดยที่หนึ่ง Rigid Body จะแทนแค่ หนึ่งวัตถุเท่านั้น

3.4.3. ส่วนคำนวณปฏิสัมพันธ์หลังการชน (Collision Response)

ในส่วนนี้จะใช้ควบคู่กับคลาส Contact ซึ่งคลาสนี้จะทำหน้าที่เก็บข้อมูลหลังการชนกันของวัตถุเพื่อนำไปเป็น Input ให้กับคลาส Collision Response ไปคำนวณแรงและทิศทางร่วมไปถึงความเร็วหลังจากที่ชนกัน รูปแบบของคลาส Contact จะเป็นดังนี้

```
Class Contact
{
Public:
        RigidBody A;    //Rigid Body A at Collision
        RigidBody B;    //Rigid Body B at Collision
        Vector3 N;      // Normal of face Rigid Body A
        Vector3  PA;    //Contact Point A;
```

```
        Vector3 PB;     //Contact Point B;
}
```

จะเห็นได้ว่ามีการเก็บค่าของ RigidBody ของวัตถุ แรก และเก็บค่าของ RigidBody ของวัตถุที่สอง และเก็บค่าของ Normal หน้าที่เกิดการชนกันของวัตถุ และเก็บจุดที่ชนกัน โดยทั้งหมดนี้ผู้ใช้งานต้องเป็นคนป้อนค่าเอง ซึ่งค่าเล่านี้จะเป็นอินพุทของ CollisionResponse โดยมีรูปแบบของคลาสดังนี้

```
class CollisionResponse
{
public:

        CollisionResponse(std::vector<Contact>& oContact);



        void ComputePreimpulseVelocity (float* rgfPreRelVel);
        void ComputeImpulseMagnitude (float* rgfPreRelVel, float* rgfImpulseMag);
        void DoImpulse (float* fImpulseMag);


        void Compute ();
        void SetContact(std::vector<Contact>& roContact);
        void AddContact(Contact roContact);
        std::vector<Contact> GetContact();
        int GetiNumContact();
        void Clear();
        void SetRestitution(float fRestitution);
        float GetRestitution() const;
        std::vector<Contact>& m_roContact;
private:
        float m_fRestitution;
        int m_iNumContacts;
};
```

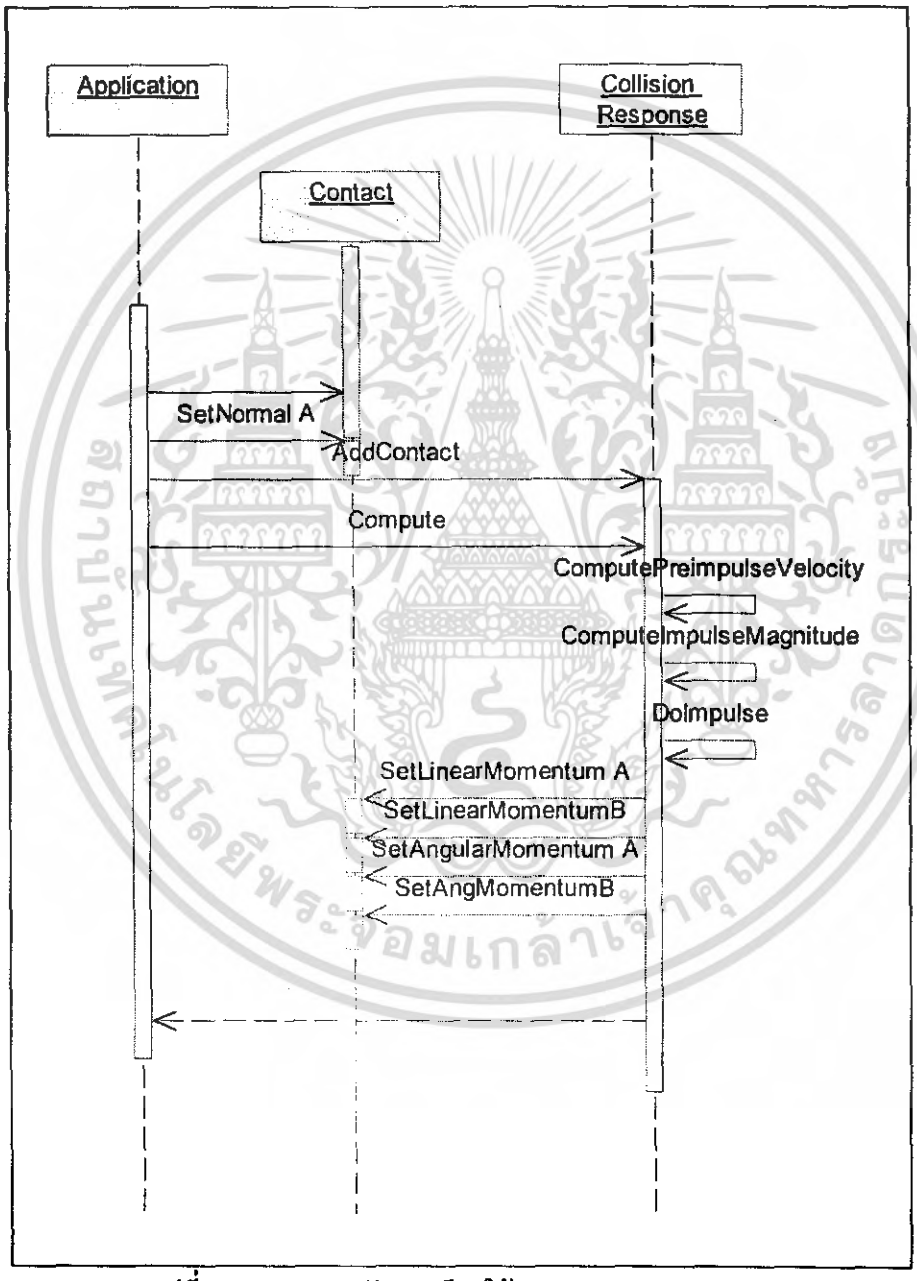
ในการคำนวณจะเรียกผ่านฟังชั่น Compute โดยในฟังก์ชั่นนี้จะทำการ อัพเดทค่าทิศทางและความเร็วใช้กับ RigidBody ทั้งสองโดยการเรียก ฟังชั่นอีกสามตัวคือ ComputePreimpulseVelocity , ComputeImpulseMagnitude , DoImpulse ซึ่งในแต่ละตัวทำหน้าที่ดังนี้

ComputePreimpulseVelocity – ทำหน้าที่คำนวณความเร็วลัพธ์ก่อนการชนกันในแนว normal เพื่อดูว่าบอลที่ชนกันนั้นมีโอกาสที่จะเคลื่อนที่ตามกันรึไม่

ComputeImpulseMagnitude - ทำหน้าที่ คำนวณ impulse ที่เกิดขึ้นระหว่างสองวัตถุ

DoImpulse – ทำการนำ impulse ที่ได้มา โมเมนตั้มให้กับแต่ละวัตถุ

โดยจะเป็นไปตามรูปที่ 3.44



รูปที่ 3.44 แสดงลำดับการเรียกใช้งาน Collision Response

เมื่อได้ตามนี้จะทำให้เราสามารถหาได้ว่าหลังจากชนวัตถุจะไปในทิศทางใด ซึ่งถ้าต้องการผลลัพธ์ที่ถูกต้องจะต้องมีการป้อนตัวแปลให้ถูกตามไปด้วย

3.5 ทฤษฎีที่เกี่ยวข้องกับ Physically Base Simulation

3.5.1.ส่วนตรวจสอบการชนกัน

ในส่วนนี้เป็นการนำสมการทางคณิตศาสตร์มาคำนวณหาการชนกันของวัตถุต่างๆ ซึ่งมีความซับซ้อนอยู่บ้าง โดยจะขอยกตัวอย่างขึ้นมาบางตัวอย่างดังนี้

3.5.1.1. การตรวจสอบการชนระหว่างเส้นตรงกับสามเหลี่ยม

ในส่วนของการตรวจจับการชนระหว่างเส้นตรงกับสามเหลี่ยมนั้น สมการที่นำมาแสดงเส้นตรงคือ $X(t) = P + tD$ โดย D คือเวคเตอร์หนึ่งหน่วยแสดงทิศทางของเส้นตรง , P เป็นจุดเริ่มของเส้นตรง , t เป็นจำนวนเต็มใดๆ โดยสามารถบอกประเภทของเส้นตรงได้จากค่าของ t ซึ่ง ถ้าเป็น เรย์ ค่า t จะมากกว่าหรือเท่ากับ 0 แต่ถ้าเป็นส่วนของเส้นตรง ค่า t จะเป็นสมาชิกของ t เริ่มต้นและ t สุดท้ายของส่วนของเส้นตรง สำหรับสามเหลี่ยมที่ประกอบไปด้วยจุด V_0, V_1, V_2 จะถูกแสดงอยู่ในรูป barycentric coordinate ตามสมการที่ (3.17)

$$b_0V_0 + b_1V_1 + b_2V_2 = V_0 + b_1E_1 + b_2E_2 \qquad (3.17)$$

โดยที่ $b_0 + b_1 + b_2 = 1$ และ $E_1 = V_1 - V_0, E_2 = V_2 - V_0$ และ $0 \le b_1, 0 \le b_2$ แต่ $b_1 + b_2 \le 1$ เมื่อนำสมการทั้งสองมารวมกันเพื่อจะใช้หาจุดตัด และกำหนดให้ $Q = P - V_0$ จะได้สมการที่ (3.18)

$$Q + tD = b_1E_1 + b_2E_2 \qquad (3.18)$$

ซึ่งในสมการ(2.4)นี้มีค่าสามค่าที่เราไม่ทราบคือค่าของ t , b_1 และ b_2 ดังนั้นจึงต้องทำการแก้สมการเพื่อหาค่าทั้งสามทีละค่าโดยเริ่มจาก กำหนดให้ $N = E_1 \times E_2$ จากนั้นทำการครอส E_2 กับด้านขวาของทุกเทอม ในสมการที่ (3.18) จะได้สมการที่ (3.19)

$$Q \times E_2 + tD \times E_2 = b_1E_1 \times E_2 + b_2E_2 \times E_2 \qquad (3.19)$$

จัดรูปสมการที่ (2.5) ได้ เนื่องจาก $E_2 \times E_2 = 0$ ดังนั้นจะได้สมการที่ (3.20)

$$Q \times E_2 + tD \times E_2 = b_1N \qquad (3.20)$$

จากนั้นทำการ ดอท สมการ(3.20) ด้วยค่า D จะได้ สมการที่ (3.21)

$$D \cdot Q \times E_2 + tD \cdot D \times E_2 = b_1 D \cdot N \tag{3.21}$$

จัดรูปสมการที่ (3.21) ใหม่ได้สมการที่ (3.22)

$$D \cdot Q \times E_2 = b_1 D \cdot N \tag{3.22}$$

เนื่องจากว่า เส้นตรงจะตัดกับสามเหลี่ยมได้นั้นเส้นตรงต้องไม่ขนาน กับ สามเหลี่ยม แปลว่า $N \cdot D \neq 0$ และจากสมการที่ (3.22) จะได้ค่าของ สมการที่ (3.23)

$$b_1 = \frac{D \cdot Q \times E_2}{D \cdot N} \tag{3.23}$$

ทำ เหมือนเดิมอีกครั้งตั้งแต่สมการที่ (3.19)-(3.22) แต่คราวนี้ทำการครอส E_1 กับด้านซ้ายของทุกเทอมในสมการ (3.19) เสร็จแล้วทำการดอทผลลัพท์ที่ได้ด้วย D ก็จะได้เป็นสมการที่ (3.24)

$$b_2 = \frac{D \cdot E_1 \times Q}{D \cdot N} \tag{3.24}$$

สุดท้ายจะหา ค่า t โดยการนำเวคเตอร์ N ดอทเข้าไปกับทุกเทอมในสมการที่ (3.19) เมื่อเวคเตอร์ E_1 และ E_2 ดอทกับเวคเตอร์ N แล้วจะได้ เท่ากับ 0 เพราะว่า ทั้ง E_1 และ E_2 ตั้งฉากกับ N นั้นเองจะได้ ดังสมการที่ (3.25)

$$t = \frac{-Q \cdot N}{D \cdot N} \tag{3.25}$$

ค่าที่ได้จากสมการที่ (3.23) , (3.24) , (3.25) เป็นตัวแปลของจุดที่ชนกันระหว่างเส้นตรงและสามเหลี่ยม ซึ่งถ้าเป็น เรย์ จะตรวจสอบจากค่า t โดย t จะต้องมากกว่าเท่ากับ 0 แต่ถ้าเป็นส่วนของเส้นตรง t จะต้องมีค่าระหว่าง t เริ่มต้นและ t สุดท้ายของส่วนของเส้นตรง ถ้าไม่ตรงตามเงื่อนไขดังกล่าวก็แสดงว่าไม่เกิดการชน แต่ถ้าตรงตามเงื่อนไขจุดที่ชนจะเป็นจุดเดียวกับเส้นตรงนั้นเอง

3.5.1.2. การตรวจสอบการชนระหว่างเส้นตรงกับทรงกลม

ในการตรวจสอบนั้นได้ใช้ สมการเส้นตรง คือ $X(t) = P + tD$ โดย D คือเวคเตอร์หนึ่งหน่วยแสดงทิศทางของเส้นตรง , P เป็นจุดเริ่มของเส้นครง , t เป็นจำนวนเต็มใดๆ

โดยสามารถบอกประเภทของเส้นตรงได้จากค่าของ t ซึ่ง ถ้าเป็น เรย์ ค่า t จะมากกว่าหรือเท่ากับ 0 แต่ถ้าเป็นส่วนของเส้นตรง ค่า t จะเป็นสมาชิกของ t เริ่มต้นและ t สุดท้ายของส่วนของเส้นตรง และนำสมการอีกสมการหนึ่งคือ สมการวงกลม คือ $|X-C|=r$ โดยที่ C คือจุดศูนย์กลางของทรงกลมและมีรัศมีเท่ากับ r จากนั้นเราจะทำการคำนวณหาจุดที่เส้นตัดกับทรงกลมได้จากการนำสมการเส้นตรงเข้ามาแทนที่ X ในสมการทรงกลมจะได้ดังสมการที่ 3.26

$$\begin{aligned} 0 &= |X-C|^2 - r^2 = |tD+P-C|^2 - r^2 = |tD+\Delta|^2 - r^2 \\ &= t^2 + 2D \bullet \Delta t + |\Delta|^2 - r^2 \end{aligned} \tag{3.26}$$

เมื่อทำการแก้สมการหาค่าของที่จะได้ดังสมการที่ 3.27

$$t = -D \bullet \Delta \pm \sqrt{(D \bullet \Delta)^2 - (|\Delta|^2 - r^2)} \tag{3.27}$$

จากสมการที่ 3.27 ทำการกำหนดตัวแปล A = $\sqrt{(D \bullet \Delta)^2 - (|\Delta|^2 - r^2)}$ ซึ่งถ้า A มีค่ามากกว่าศูนย์แสดงว่าจะมีจุดของเส้นที่ตัดกับทรงกลมสองจุด แต่ถ้า A เท่ากับ 0 จะมีจุดตัดกับทรงกลมเพียง 1 จุด และถ้าหาก A น้อยกว่าศูนย์ ซึ่งจะทำให้ผลลัพธ์ที่ได้ไม่ใช่จำนวนจริง แสดงว่าเส้นตรงไม่ตัดกับทรงกลม เราจึงสรุปได้ว่า ถ้า A เท่ากับหรือมากกว่า 0 แสดงว่าเส้นตรงตัดกับทรงกลม แต่ถ้าน้อยกว่า 0 แสดงว่าเส้นตรงไม่ตัดกับทรงกลม

3.5.1.3. การตรวจสอบการชนระหว่างเส้นตรงกับกล่อง

ในการตรวจสอบนี้จะใช้การประมวลผลในแบบสองมิติ โดยมีสมการดังนี้

$$\begin{aligned} |U_0 \bullet D \times \Delta| &> e_1 |D \bullet U_2| + e_2 |D \bullet U_1| \\ |U_1 \bullet D \times \Delta| &> e_0 |D \bullet U_2| + e_2 |D \bullet U_0| \\ |U_2 \bullet D \times \Lambda| &> e_0 |D \bullet U_1| + e_1 |D \bullet U_0| \end{aligned} \tag{3.28}$$

โดยที่ $\Delta = P - C$, D เท่ากับทิศทางของเส้นตรง U เท่ากับแกนของสี่เหลี่ยมซึ่งมีทั้งหมดสามแกนตามลำดับ e เท่ากับขนาดของแต่ละด้านตามแนวแกนของสี่เหลี่ยม ถ้าตรงตามเงื่อนไขใดเงื่อนไขหนึ่งตามสมการที่ แสดงว่าเส้นตรงไม่ตัดกับกล่อง แต่ถ้าไม่ตรงทุกเงื่อนไขแสดงว่าเส้นตรงตัดกับกล่อง

ในตัวอย่างทั้งสามนี้เป็นตัวอย่างที่ยกมาให้เห็นว่าในการหาการชนกันของวัตถุนั้นได้อาศัยสมการทางคณิตศาสตร์เข้ามาช่วยในการคำนวณ รวมไปถึงรูปทรงเลขาคณิตเข้ามาช่วย

3.5.1.4. การตรวจสอบการชนกันระหว่าง ทรงกลมกับทรงกลม

ในการตรวจสอบนั้นสามารถหาได้โดยอาศัยสมการ $|X - C| = r$ โดยที่ C คือจุดศูนย์กลางของทรงกลมและมีรัศมีเท่ากับ r จากนั้นทำการแทนค่า X ด้วย C + r ของอีกทรงกลมหนึ่ง เราจะได้สมการที่ 3.29

$$C_1 - C_2 = r_1 + r_2 \qquad (3.29)$$

จากสมการที่ ทำการยกกำลังทั้งสองข้างเพื่อให้ค่าที่ได้ออกมาเป็นบวกเท่านั้น จะได้เป็นสมการที่

$$(C_1 - C_2)^2 = (r_1 + r_2)^2 \qquad (3.30)$$

ซึ่งจากสมการที่ ถ้า $(C_1 - C_2)^2$ น้อยกว่าหรือเท่ากับ $(r_1 + r_2)^2$ แสดงว่าทรงกลมทั้งสอง Intersect กัน แต่ถ้ามากกว่าแสดงว่าไม่ Intersect กัน

3.5.2. ส่วนการจำลองการเคลื่อนที่ (Rigid body simulation)

การจำลองการเคลื่อนที่ของวัตถุแข็งนั้นจะใช้การเลียนแบบการเคลื่อนที่ตาม สมการฟิสิกส์ โดยดูจากความสัมพันธ์ระหว่างสมการฟิสิกส์ ซึ่งมีอยู่สองประเภทด้วยกันคือ การเคลื่อนที่เชิงเส้น,การเคลื่อนที่เชิงมุม

3.5.2.1.การเคลื่อนที่เชิงเส้น

ในการเคลื่อนที่เชิงเส้นนั้นจะนำสมการของ โมเมนตั้มเชิงเส้นมาใช้ตามสมการที่ (3.31)

$$P(t) = mV(t) \qquad (3.31)$$

โดยที่ P = โมเมนตั้มของวัตถุ ณ. เวลานั้น

m = มวลของวัตถุ

V = ความเร็วของวัตถุ ณ. เวลานั้น

เนื่องจาก ถ้าเรารู้โมเมนตั้มและรู้มวลเราจะหาความเร็วของวัตถุได้ และจากสมการที่ (3.31) ถ้านำมาหาอนุพันธ์จะได้เป็นแรงที่กระทำกับวัตถุ

$$\dot{P} = F \qquad (3.32)$$

จากสมการที่ (3.32) เรารู้แรงเราจะสามารถหาความเร่งของวัตถุ จากความคิดดังกล่าว จะได้ตัวที่มีผลทำให้วัตถุเคลื่อนที่ในเชิงเส้น คือ ค่าความเร็ว และแรงที่กระทำกับวัตถุ

3.5.2.2. การเคลื่อนที่เชิงมุม

ในการเคลื่อนที่เชิงมุมจะอาศัยสมการของ โมเมนตั้มเชิงมุมมาใช้ตามสมการที่3.33

$$L(t) = I(t)\omega(t) \tag{3.33}$$

โดยที่ L = โมเมนตั้มเชิงมุม

I = Inertia Tensor

w = ความเร็วเชิงมุม

เมื่อนำสมการที่ (3.33) มาหาอนุพันธ์จะได้ ค่าของความเร่งเชิงมุมตามสมการ(3.34)

$$\dot{L}(t) = \tau(t) \tag{3.34}$$

ซึ่ง Inertia Tensor I(t) ในสมการที่ (3.33) หมายถึงปริมาณที่บอกสมบัติในการต้านการเปลี่ยนสภาพ การหมุนของวัตถุ สามารถคำนวณได้จาก สมการที่ (3.35)

$$I(t) = \sum \begin{pmatrix} m_i\left(r_{iy}^{'2} + r_{iz}^{'2}\right) & -m_i r_{ix}^{'} r_{iy}^{'} & -m_i r_{ix}^{'} r_{iz}^{'} \\ -m_i r_{iy}^{'} r_{ix}^{'} & m_i\left(r_{ix}^{'2} + r_{iz}^{'2}\right) & -m_j r_{iy}^{'} r_{iz}^{'} \\ -m_i r_{iz}^{'} r_{ix}^{'} & -m_i r_{iz}^{'} r_{iy}^{'} & m_j\left(r_{ix}^{'2} + r_{iy}^{'2}\right) \end{pmatrix} \tag{3.35}$$

ในความเป็นจริง I(t) ไม่ใช่ค่าคงที่ ในการเปลี่ยนแปลงเกิดขึ้นตลอดเวลาเนื่องจากการหมุนของวัตถุดังนั้นเราจึงต้องหาความสัมพันธ์ระหว่าง I ที่เขียนอยู่ในรูปของ Body space และ I(t) ซึ่งได้ความสัมพันธ์ดังสมการที่ (3.36) และ (3.37)

$$I(t) = R(t) I_{body} R(t)^T \tag{3.36}$$

$$I^{-1}(t) = \left(R(t) I_{body} R(t)^T\right)^{-1} = R(t) I_{body}^{-1} R(t)^T \tag{3.37}$$

ถึงจุดนี้เราสามารถสรุปสถานะของวัตถุให้อยู่ในรูปของเวกเตอร์ได้โดยทำการเก็บค่าที่แสดงสถานะของวัตถุดังสมการที่ (3.38)

$$Y(t) = \begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix} \tag{3.38}$$

เวกเตอร์นี้ประกอบไปด้วยตำแหน่งวัตถุ การหมุนวัตถุ Momentum เชิงเส้น Momentum เชิงมุม ซึ่ง Momentum เชิงเส้น Momentum เชิงมุม จะแสดงข้อมูลด้านความเร็ว ซึ่งมีความสัมพันธ์ตามสมการ ที่ (3.39) (3.40) (3.41)

$$v(t) = \frac{P(t)}{M} \tag{3.39}$$

$$\omega(t) = I(t)^{-1} L(t) \tag{3.40}$$

อนุพันธ์ของ Y(t) ในสมการที่ (3.38) จะอยู่ในรูป

$$\frac{d}{dt}Y(t) = \frac{d}{dt}\begin{pmatrix} x(t) \\ R(t) \\ P(t) \\ L(t) \end{pmatrix} = \begin{pmatrix} v(t) \\ \omega(t) * R(t) \\ F(t) \\ \tau(t) \end{pmatrix} \tag{3.41}$$

ค่าของอนุพันธ์ของ Y(t) ในสมการที่ (3.41) จะสามารถบอกตำแหน่งและความสัมพันธ์ของการเคลื่อนที่ ณ เวลาต่อไปทำให้เราสามารถทำนายการเคลื่อนที่ของวัตถุได้

3.2.5.3.ส่วนคำนวณปฏิสัมพันธ์หลังการชน (Collision Response)

สามารถคำนวณทิศทางของความเร็วหลังจากการชนกันได้จากสมการ

$$\hat{n} \bullet (p_a^{+} - p_b^{+}) = -\varepsilon(\hat{n} \bullet (p_a^{-} - p_b^{-})) \tag{3.42}$$

โดยที่ $\hat{n}$ คือ normal ของวัตถุ b และ p_a^{+} คือความเร็วที่เกิดหลังจากการชนของวัตถุ a , p_b^{+} คือความเร็วที่เกิดหลังจากการชนของวัตถุ b , ε เท่ากับค่าคงที่ในการสูญเสียแรงในการชน , p_a^{-} คือความเร็วก่อนเกิดการชนของวัตถุ a , p_b^{-} คือความเร็วก่อนเกิดการชนของวัตถุ b ซึ่ง รายละเอียดที่มาของสมการข้างบนได้มาจากการคำนวณหาค่าของ v_{rel} (velocity relative) โดยค่านี้คำนวณมาจากสมการที่ 3.43

$$p_a(t_0) = v_a(t_0) + \omega_a(t_0) \times (p_a(t_0) - x_a(t_0)) \tag{3.43}$$

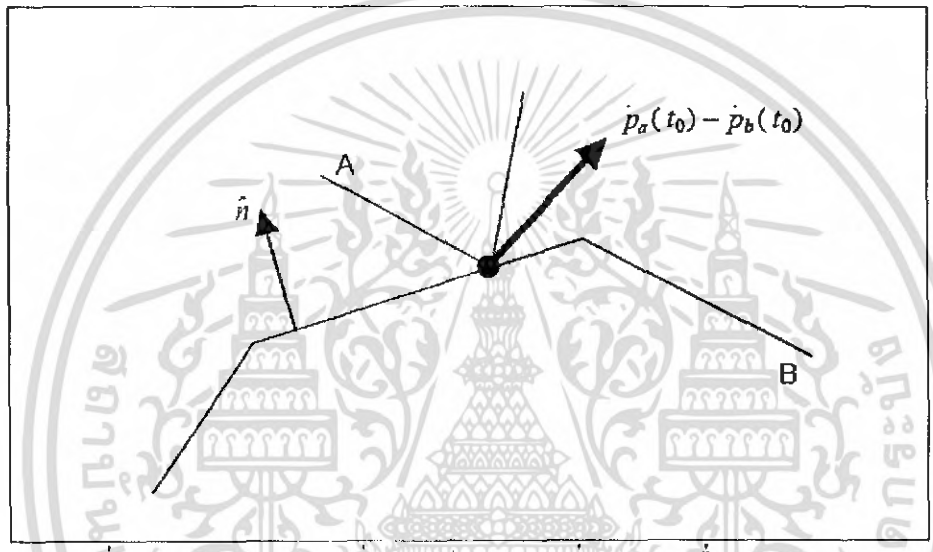
ซึ่งเป็นค่าความเร็ว ณ จุดที่ชนกันของวัตถุ a ก่อนการชน โดยที่ v_a คือความเร็วเชิงเส้นของวัตถุ a และ ω_a คือความเร็วเชิงมุมของวัตถุ a คอสกับ จุดที่ชนกันของวัตถุ(p_a) ลบกับตำแหน่งจุดศูนย์กลางของวัตถุ a (x_a) ณ เวลาเดียวกันคือ (t_0) และสมการที่

$$p_b(t_0) = v_b(t_0) + \omega_b(t_0) \times (p_b(t_0) - x_b(t_0)) \tag{3.44}$$

ซึ่งเป็นค่าความเร็ว ณ จุดที่ชนกันของวัตถุ b ก่อนการชน โดยที่ v_b คือความเร็วเชิงเส้นของวัตถุ b และ ω_b คือความเร็วเชิงมุมของวัตถุ b คอสกับ จุดที่ชนกันของวัตถุ(p_b) ลบกับตำแหน่งจุดศูนย์กลางของวัตถุ b (x_b) ณ เวลาเดียวกันคือ (t_0) จากนั้นนำสมการทั้งสองมาหา v_{rel} (velocity relative) ซึ่งเป็นไปตามสมการที่

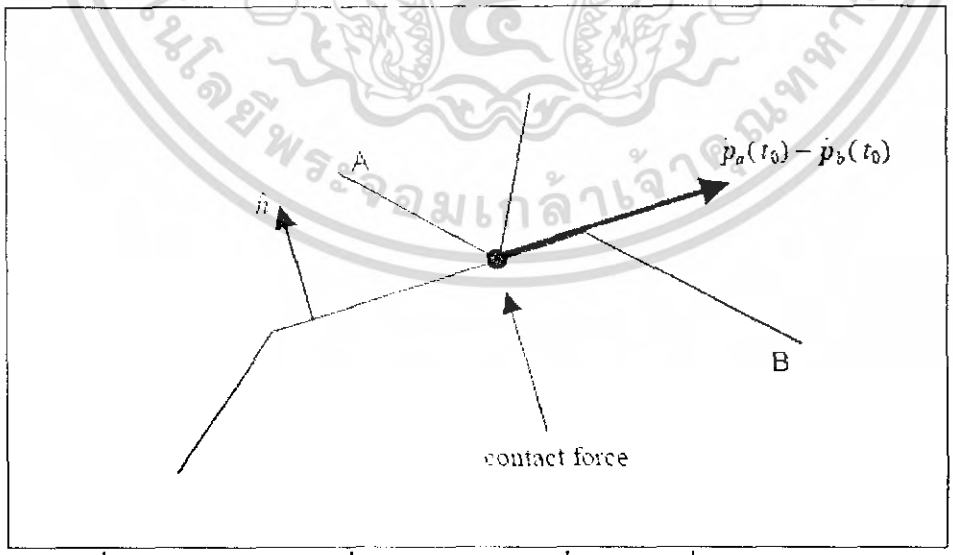
$$v_{rel} = \hat{n}(t_0) \bullet (p_a(t_0) - p_b(t_0)) \qquad (3.45)$$

โดยที่ v_{rel} คือ (velocity relative) $\hat{n}$ คือ normal หนึ่งหน่วย p_a คือค่าที่ได้จากสมการ ที่ และ p_a คือค่าที่ได้จากสมการ ที่ ที่เราต้องทำการหา v_{rel} ก็เพื่อต้องการรู้ว่าวัตถุกำกลังเคลื่อนที่ในทิศทางใด ซึ่งถ้า v_{rel} มากกว่าศูนย์แสดงว่าความเร็วนี้มีทิศทางเดียวกับ normal สามารถดูได้จากรูปที่ 3.45



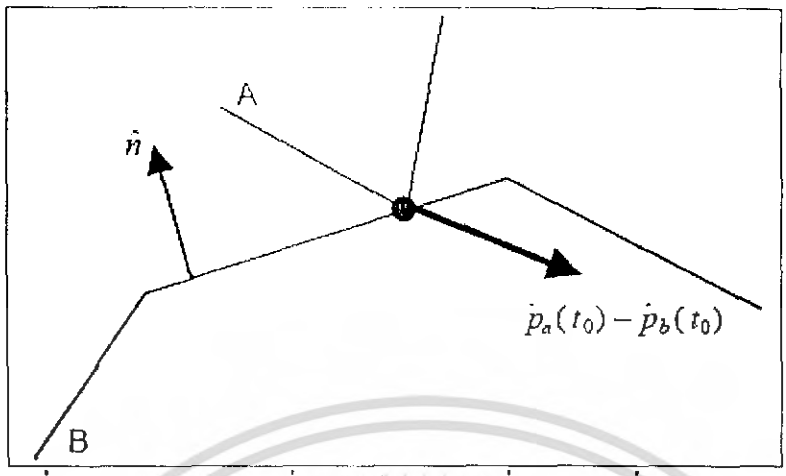
รูปที่ 3.45 แสดงความเร็วที่สัมพันธ์กันของจุดที่สัมผัสกันเมื่อ v_{rel} เป็นบวก

แต่ถ้า v_{rel} เท่ากับ 0 แสดงว่าวัตถุนั้นไถลอยู่บนอีกวัตถุหนึ่งดังรูปที่ 3.46



รูปที่ 3.46 แสดงความเร็วที่สัมพันธ์กันของจุดที่สัมผัสกันเมื่อ v_{rel} เท่ากับ ศูนย์

แต่ถ้า v_{rel} ติดลบแสดงว่าวัตถุนั้นเคลื่อนที่เข้าไปในวัตถุอีกวัตถุหนึ่งดังตัวอย่างในรูปที่ 3.47



รูปที่ 3.47 แสดงความเร็วที่สัมพันธ์กันของจุดที่สัมผัสกันเมื่อ v_{rel} ติดลบ

โดย impulse ที่เกิดจะเป็นแรงที่กระทำกับจุดสัมผัส ณ. เวลาที่น้อยมากซึ่งจะได้สมการดังสมการที่ 3.46

$$F\Delta t = J \tag{3.46}$$

J คือ impulse , F คือแรงที่กระทำกับจุด Δt คือเวลาที่น้อยมากเกือบเข้าใกล้ ศูนย์ ตัวอย่างเช่น ถ้าเราต้องการนำแรง impulse ที่ได้ไปกระทำกับวัตถุมวล M เราจะได้ความเร็วของมวลนั้นเท่ากับสมการที่

$$\Delta v = \frac{J}{M} \tag{3.47}$$

จากสมการที่ คือความเร็วที่เปลี่ยนไปจะเท่ากับ impulse หารด้วย มวล ซึ่งเราสามารถสรุปได้ว่า J คือโมเมนตัมที่เปลี่ยนไปของมวล แต่ถ้าเราต้องการหาทอร์กที่เกิดจาก impulse ด้วยก็สามารถคำนวณได้จาก สมการที่

$$\tau_{impulse} = (p \quad x(t)) \times J \tag{3.48}$$

$\tau_{impulse}$ คือ ทอร์กที่เกิดจาก impulse

P คือ ตำแหน่งที่ชนกัน

X คือ จุดศูนย์กลางวัตถุ

J คือ impulse ที่เกิดขึ้น

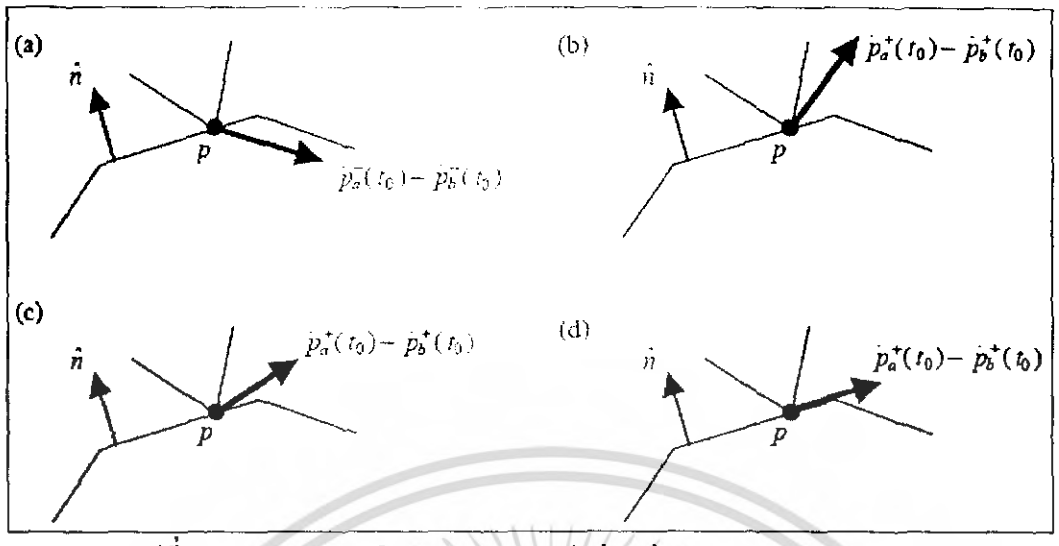
ดังนั้นถ้าเราสามารถรู้ความเร็วและทิศทางหลังจากการชนได้จากสมการที่

$$v_{rel}^{+} = -\varepsilon v_{rel}^{-} \tag{3.49}$$

โดยที่ v_{rel}^{+} คือความเร็วและทิศทางหลังการชน

v_{rel}^{-} คือ ความเร็วและทิศทางก่อนการชน

ε คือค่าการสูญเสียที่เกิดในการชน โดยค่านี้ จะเป็นตัวกำหนดว่ารูปแบบของแรงหลังการชนนั้นจะออกมาในลักษณะใด โดยสามารถดูได้จากรูปที่ 3.48



รูปที่ 3.48 A แสดงทิศทางของความเร็วก่อนผ่านการคำนวณ impulse
B แสดง ทิศทางหลังจากผ่านการคำนวณ impulse โดยมี $\varepsilon = 1$
C แสดง ทิศทางหลังจากผ่านการคำนวณ impulse โดยมี ε ระหว่าง 0 ถึง 1
D แสดง ทิศทางหลังจากผ่านการคำนวณ impulse โดยมี $\varepsilon = 0$

จากทั้งหมดนี้สามารถสรุปเป็น สูตรสมการที่ และเมื่อผ่านการคำนวณ impulse แล้วก็ทำการ อัพเดทค่าให้กับวัตถุที่ชนกันทั้งค่า ของ โมเมนตั้มเชิงเส้น และโมเมนตั้มเชิงมุม ทำให้วัตถุมีการเคลื่อนที่เปลี่ยนแปลงไปจากเดิม

3.5 Drawing Tools

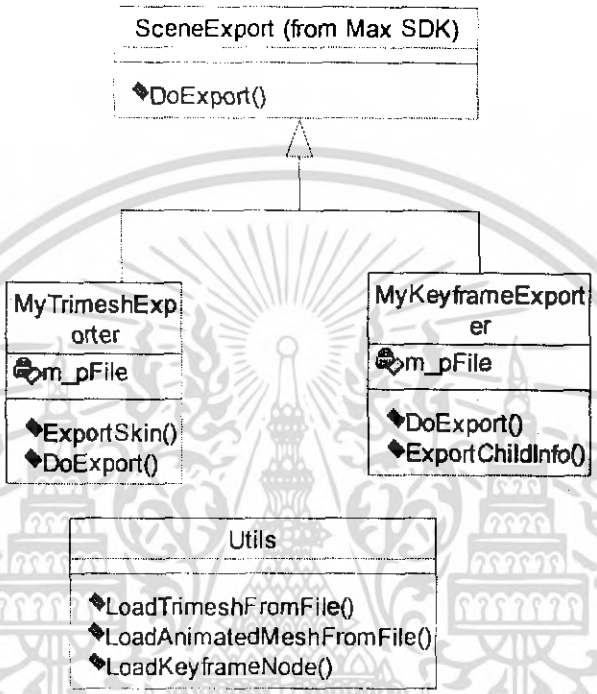
มีไว้เพื่อช่วยในการจัดรูปแบบของไฟล์ที่สามารถนำมาใช้ใน PC Engineเป็นหลักได้แก่ Terrain Height Field ที่สามารถแปลงจาก Photoshop RAW Format ไปเป็นรูปแบบที่กำหนดไว้ในส่วนของ Terrain ได้ การออกแบบไม่ซับซ้อนเขียนอยู่ในรูปแบบของ C-Style function แต่กำหนดให้เป็น static member ของคลาส Tools เพื่อสำหรับเครื่องมือที่ถูกพัฒนาในอนาคต

PC Mesh Exporter เป็น Plug-in ของ 3Ds Studio Max8 ซึ่งพัฒนาโดยใช้ MAX SDK ซึ่งมีมาพร้อมกับโปรแกรม ทำให้สามารถเข้าถึงข้อมูลของ Scene ที่ปรากฏได้โดยง่าย โดยการที่จะสร้าง Plug-in สำหรับ 3Ds Studio Max นั้นจะต้อง Derive คลาสต่างๆมาจาก MAX SDK และทำการสร้างเป็น DLL เพื่อให้ 3Ds Max นำไปใช้ต่อไป ตัว Plug-in ที่สร้างจะทำการ Export Vertices และ Indices ของ Mesh ที่กำหนดรวมไปถึง Texture Coordinate ด้วยหากต้องการ

นอกจากที่ PC Mesh Exporter จะสามารถทำการ Export Model ง่ายๆแล้ว ต้องสนับสนุนการ Export Skin ด้วย โดยให้ Export weight และ offset ของแต่ละ Vertex มาด้วยในรูปแบบที่สามารถนำไปใช้ได้ง่าย

PC Keyframe Exporter เป็น Plug-in ของ 3Ds Studio Max เช่นเดียวกันแต่ใช้สำหรับ Export Keyframe animation ของแต่ละวัตถุได้ โดยหากวัตถุใดอยู่ในลำดับชั้นเช่น Bone ก็จะทำการ Export ลูกให้โดยอัตโนมัติ

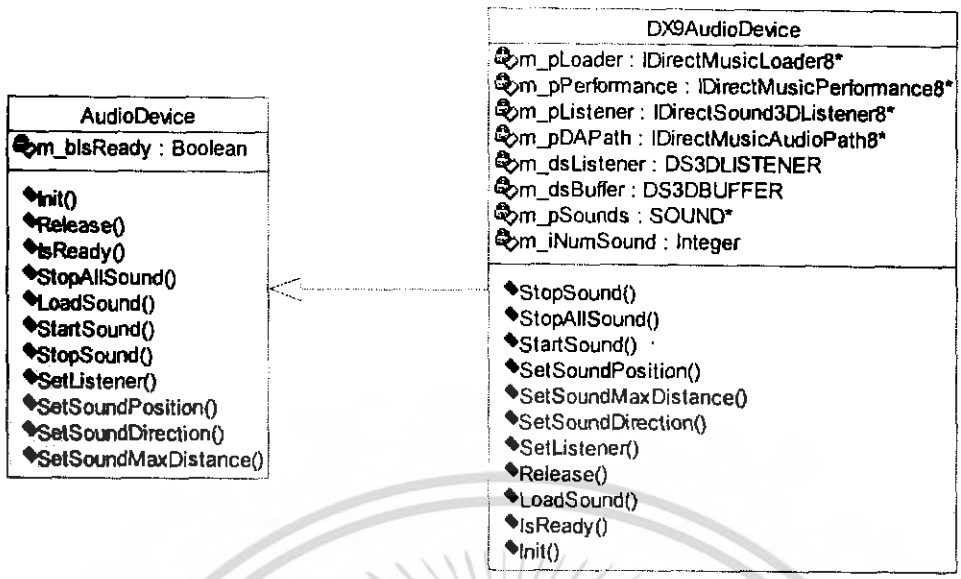
ส่วนของ Utility จะช่วยในการอ่านไฟล์รูปแบบต่างๆที่สร้างขึ้นซึ่งได้แก่ Trimesh Loader, Skin Controller Loader, Keyframe Loader การออกแบบจะง่ายไม่ซับซ้อนด้วยการสร้างแต่ละหน้าที่เป็น static member ของ Class Utils



รูปที่ 3.49 ต้นแบบ Utility และ Tools (ไม่ได้คำนึงถึงความถูกต้องในการอิมพิลเมนต์จริงตามข้อกำหนดของ 3Ds Max SDK)

3.6. Sound Utility Module

เป็นส่วนที่ทำหน้าที่รับผิดชอบเกี่ยวกับเสียง ทั้งหมด ซึ่งในส่วนนี้จะเป็น Wrapper Class เพื่อให้ผู้พัฒนาใช้งานง่ายขึ้น โดยมี Interface ชื่อ Audio Device ซึ่ง Interface นี้สามารถ Load Sound , Start Sound (เล่นเสียง) , Stop Sound (หยุดเล่น) , Stop All Sound (หยุดเล่นทุกเสียง) Set Listener (ตั้งตำแหน่งผู้ฟัง) , Set Sound Position (ตั้งตำแหน่งของเสียง) , Set Sound Direction (ตั้ง ทิศทางของเสียง) Set Sound Max Distance (ตั้งระยะที่ไกลที่สุดที่จะได้ยินเสียง) ส่วนการนำไปใช้งานจะทำการ Derive Interface นี้ออกมาเป็น DirectX Audio Device หรือว่า OpenGL Audio Device ขึ้นอยู่กับผู้พัฒนา ซึ่งภายใน Audio Device และคลาสที่ Derive จะมีหน้าตา ดัง รูปที่ 3.50



รูปที่ 3.50 แสดงสมาชิกของ Audio Device และสมาชิกของคลาส DX9Audio Device ที่ Derive ออกมา

บทที่ 4

การทดลองและผลการทดลอง

4.1. การทดลอง Vertex Color

เป็นการใส่สีให้กับจุด ที่เป็นส่วนประกอบของวัตถุ ซึ่งสามารถดูได้จากรูปที่ 4.1

รูปที่ 4.1 เป็นการใส่สีให้กับ จุดที่เป็นองค์ประกอบของ สามเหลี่ยม

จากรูปที่ 4.1 ได้ทำการใส่สีให้กับจุดสามจุดคือสี แดง สีเขียว และสีน้ำเงิน โดยสีที่เกิดระหว่างจุดนั้นเป็นการแสดงผลโดย DirectX ซึ่งได้ทำการไล่สีให้โดยอัตโนมัติ

4.2. การทดลอง Alpha Blending

การทดลองนี้เป็นการใช้ค่า Alpha ทำให้วัตถุโปร่งแสง ซึ่งสามารถดูได้จากรูปที่ 4.2

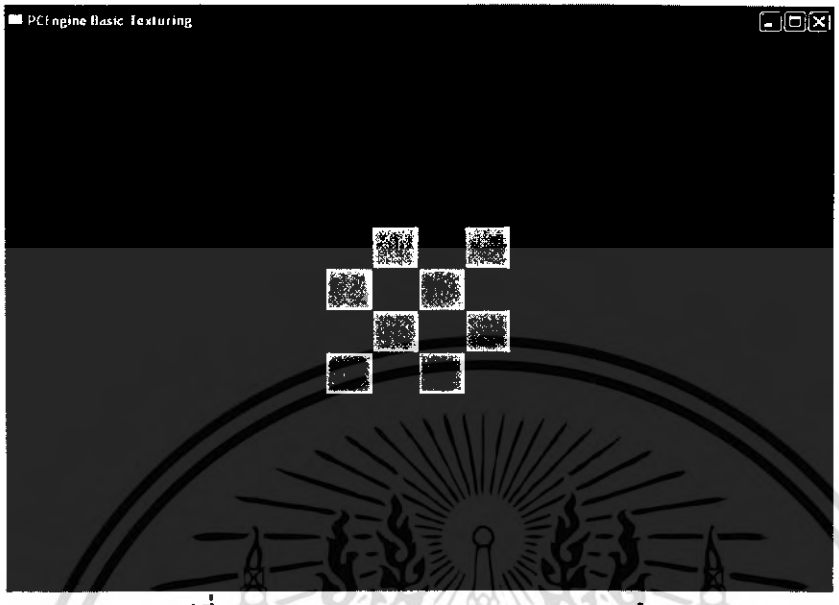
รูปที่ 4.2 แสดงการใช้ Alpha Blending สองแบบคือ Material Alpha Blending และ Texture Alpha Blending

จากรูปที่ 4.2 มีการตั้งค่า Alpha ให้กับ Material สีเขียวเท่ากับ 0.5 และตั้งค่า Alpha ให้กับส่วนของ Texture เท่ากับ 1 และส่วนที่ไม่ใช่ Texture เท่ากับ 0 ซึ่งตัวอย่าง Alpha texture เป็นไปตามรูปที่ 4.3

รูปที่ 4.3 แสดง Alpha ของ Texture ซึ่งส่วนที่เป็นสีดำคือส่วนที่มีค่าเท่ากับ 0 และส่วนที่เป็นสีขาวเท่ากับ 1

4.3 การทดลอง Texture Mapping

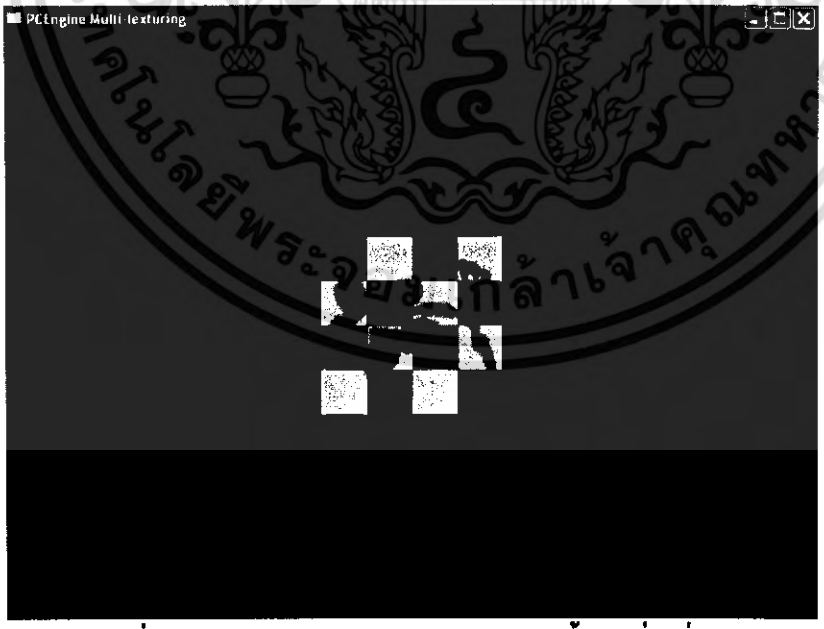
เป็นการแสดง Texture บนพื้นผิวซึ่งเป็นไปตามรูปที่ 4.4



รูปที่ 4.4 แสดงการ Map Texture ลงบนกล่อง

4.3. การทดลอง MultiTexture

การทำ MultiTexture คือการรวม Texture มากกว่าหนึ่ง Texture บน พื้นผิวเดียวกัน สามารถดูตัวอย่างได้จากรูปที่ 4.5



รูปที่ 4.5 แสดงการทำ MultiTexture เป็นพื้นผิวสี่เหลี่ยม

จากรูปที่ 4.5 จะเห็นได้ว่ามีรูปอยู่สองรูปคือ รูปลายหมากรุก กับรูปดอกไม้ซึ่งสองรูปนี้ เป็น คนละรูปกันแต่เราสามารถนำมาแสดงผลซ้อนทับกันได้บนพื้นที่เดียวกัน

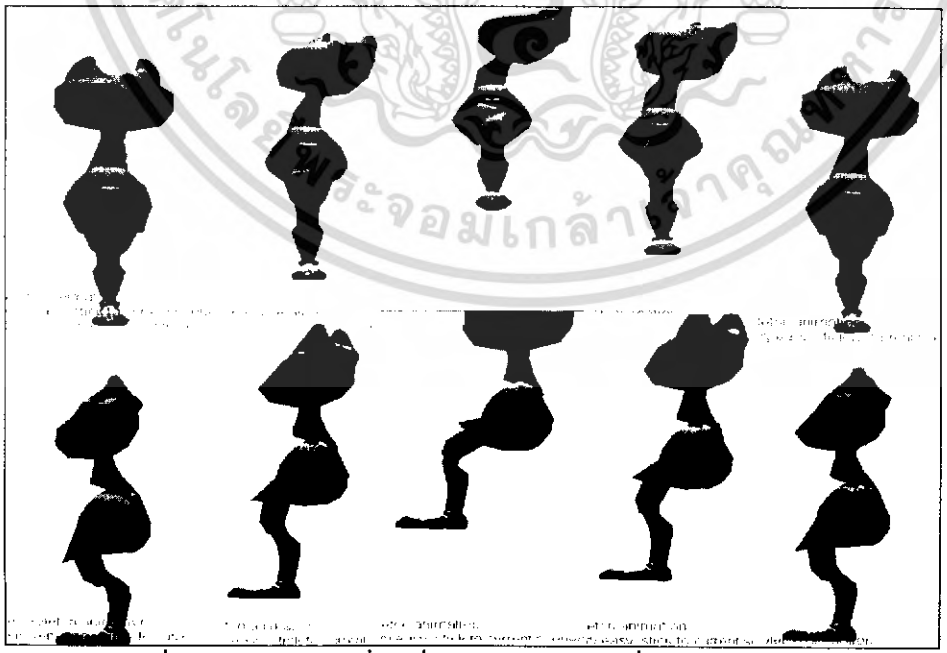
4.4. การทดลอง Bum Mapping

เป็นการจำลองภาพที่เป็น Texture นั้นให้ดูนูนขึ้นเหมือนกับพื้นผิวอยู่คนละระดับกันเป็นการทดสอบ การใช้ Shader Effect โดยผลลัพธ์เป็นไปตามรูปที่ 4.6

รูปที่ 4.6 แสดงการใช้ Shader Effect แบบ Bump Mapping

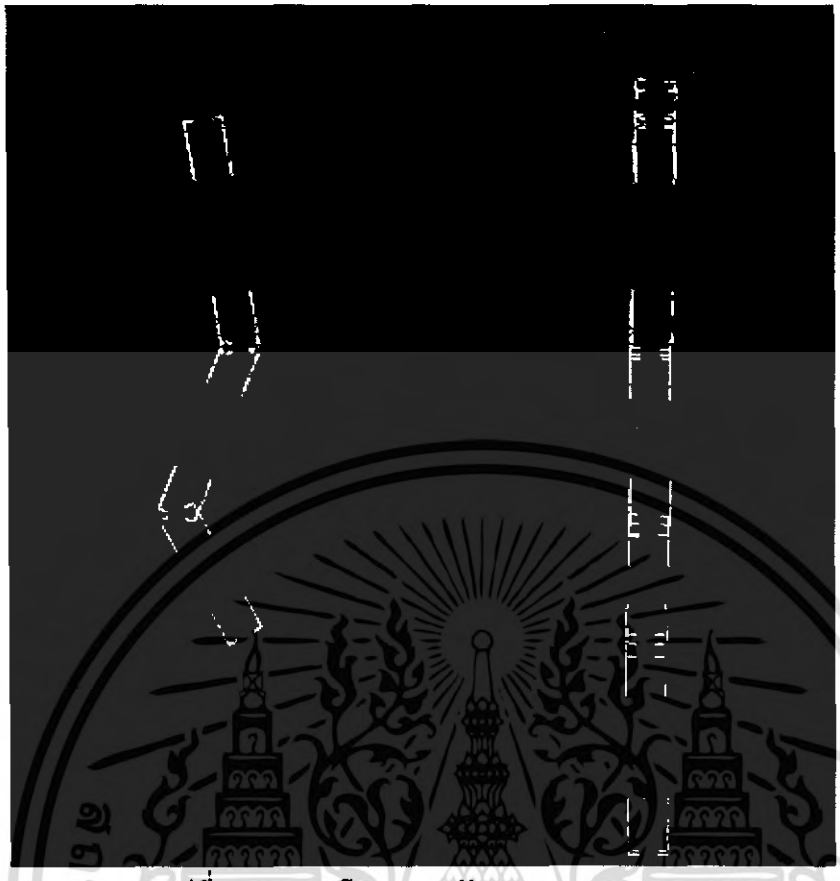
4.5. การทดลอง Animation

การเคลื่อนที่ของ Animation เป็นไปตามรูปที่ 4.7



รูปที่ 4.7 แสดงการเคลื่อนที่ของ Animation เมื่อมีการกด ปุ่ม A

ในรูป 4.7 วัตถุเคลื่อนที่ตามที่กำหนดค่าไว้โดยมี โบนของ ตัวละคร ตามรูปที่ 4.8

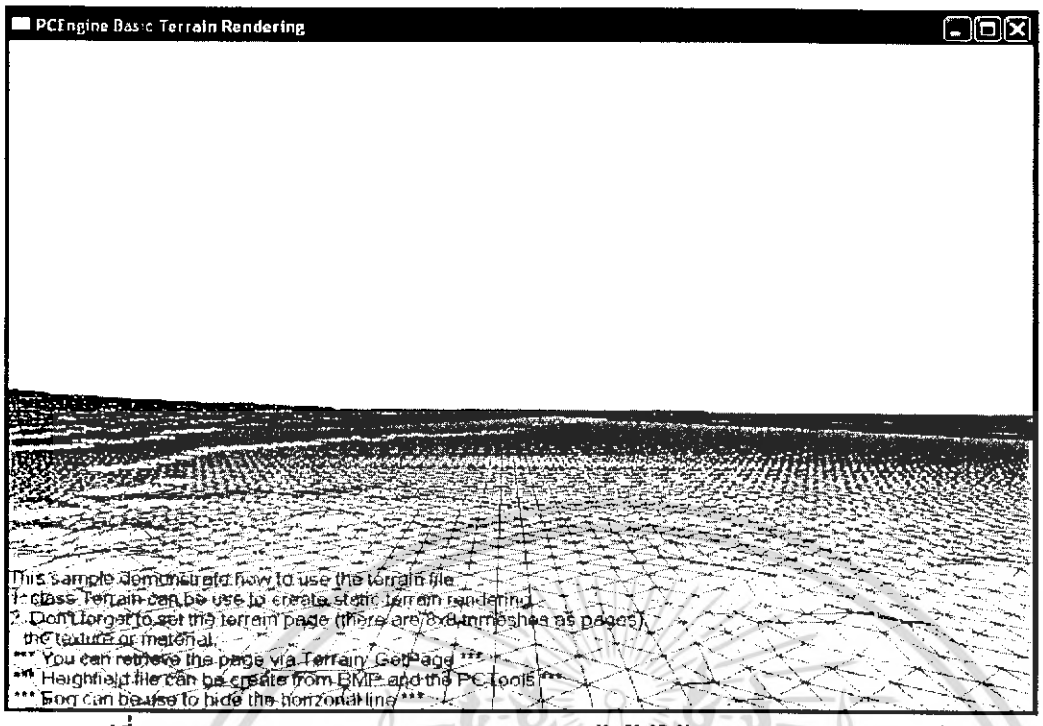


รูปที่ 4.8 แสดงโบน ของ ตัวละคร Animation

จากรูปที่ 4.8 ส่วนสีแดงคือส่วนที่เป็น โบนของ ตัวละคร ซึ่งการเคลื่อนที่ต่างๆของตัวละครจะเป็นไปตามการเคลื่อนที่ของโบนที่เรากำหนดให้กับละครนั้น

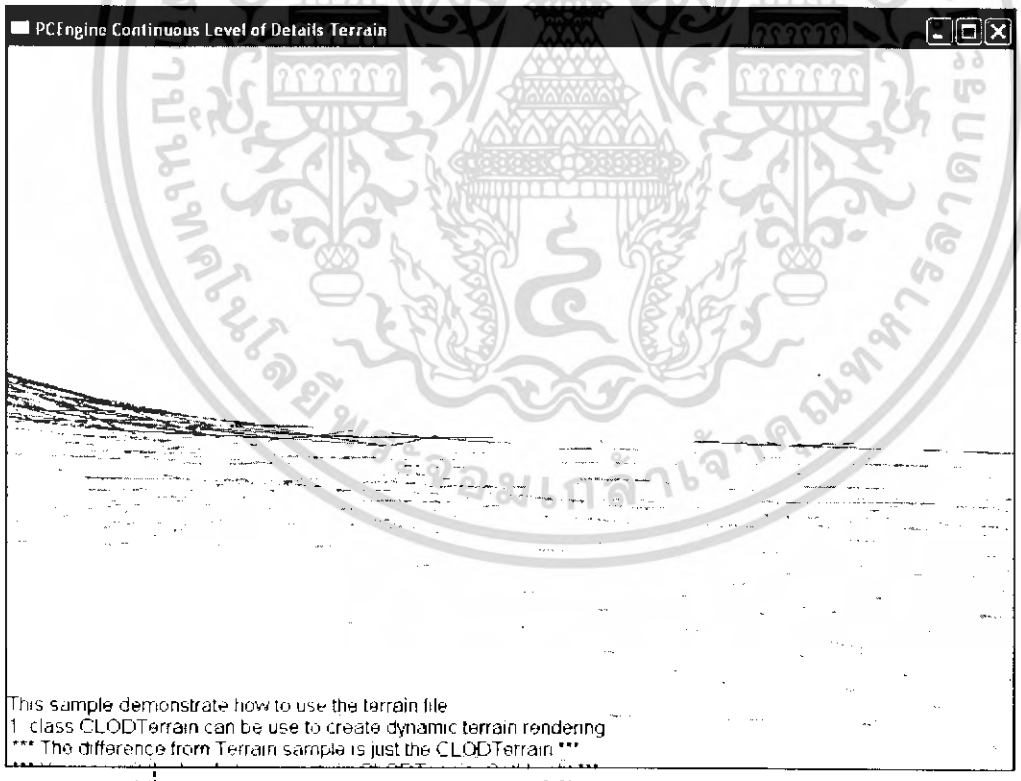
4.6. การทดสอบ Terrain

เนื่องจาก Terrain ที่ทำมานั้นประกอบไปด้วย สอง แบบด้วยกัน คือ แบบธรรมดา กับ แบบ CLOD Terrain ซึ่งแบบ ธรรมดานั้นจะทำการวาด สามเหลี่ยมขนาดเท่ากันทั้ง Terrain ส่วนแบบ CLOD Terrain นั้นจะมีการเพิ่มประสิทธิภาพ โดยจะทำการพิจารณาว่า ณ. พื้นที่ใน Terrain ตรงนั้นจะใช้สามเหลี่ยมขนาดเท่าใดในการแสดงผล ดังที่ได้กล่าวมาในส่วนของทฤษฎีแล้ว ซึ่งสามารถดูได้จาก รูปที่ 4.9 และ รูปที่ 4.10



รูปที่ 4.9 แสดงการวาด Terrain แบบธรรมดาไม่ได้ใช้ CLOD Algorithm เข้ามาช่วย

จากรูปที่ 4.9 จะเห็นได้ว่าสามเหลี่ยมที่เป็นส่วนประกอบของ Terrain นั้นมีขนาดที่เท่ากันทุกสามเหลี่ยม ซึ่งทำให้เสียเวลาในการวาด



รูปที่ 4.10 แสดงการวาด Terrain แบบใช้ CLOD Algorithm เข้ามาช่วย

ซึ่งเมื่อเปรียบเทียบระหว่างรูปที่ 4.9 กับรูปที่ 4.10 จะเห็นว่าความละเอียดของสามเหลี่ยมที่ใช้วาด Terrain ในรูปที่ 4.9 และ รูปที่ 4.10 มีจำนวนที่แตกต่างกันมากแต่ผลที่ได้ออกมาใกล้เคียงกัน คือ

ความสูงของ Terrian ณ. ตำแหน่งเดียวกันก็ทั้งสองรูปก็ใกล้เคียงกัน แต่ในเรื่องของความละเอียด CLOD จะไม่ละเอียดเท่า แบบธรรมดา แต่ในการใช้ CLOD จะเป็นการเพิ่มประสิทธิภาพให้กับโปรแกรมขึ้นมากเพราะการแสดงผลลัพธ์เดียวกันแต่วาดสามเหลี่ยมน้อยกว่าทำให้เปลืองทรัพยากรน้อยกว่า

4.8 การทดสอบการตรวจสอบการชนกันของวัตถุ พื้นฐานต่างๆ

การทดสอบนี้เป็นการทดสอบ ทฤษฎี การตรวจสอบการชนกันของวัตถุในระบบ สามมิติ ซึ่งในการทดสอบจะมีวัตถุดังนี้คือ ทรงกลม กล่อง และ ระนาบ ดังแสดงในรูปที่ 4.11

รูปที่ 4.11 ภาพของตัวอย่างในการทดสอบการตรวจสอบการชนกันของวัตถุ

จากรูปที่ 4.11 จะมีการตรวจสอบการชนกันของวัตถุดังนี้ คือ

การตรวจสอบการชนระหว่าง กล่อง กับ กล่อง

การตรวจสอบการชนระหว่าง กล่อง กับ ทรงกลม

การตรวจสอบการชนระหว่าง ทรงกลม กับ ทรงกลม

การตรวจสอบการชนระหว่าง กล่อง กับ ระนาบ

การตรวจสอบการชนระหว่าง ทรงกลม กับ ระนาบ

การตรวจสอบการชนระหว่าง เรย์ กับ กล่อง

การตรวจสอบการชนระหว่าง เรย์ กับ ทรงกลม

โดยเราสามารถเลือกวัตถุที่ต้องการให้เคลื่อนที่ได้โดย คลิกเมาส์ ไปบนวัตถุนั้น แล้วส่วนของการตรวจสอบจะนำเรย์ ไปตรวจดูว่าเรย์ที่มีทิศทางไปตรงจุดที่คลิก นั้นชนกับวัตถุใดแล้วจะนำวัตถุนั้นมาแสดงตรง select ซึ่งในตัวอย่างเป็น Sphere0 และเมื่อมีการกดตัวอักษร A วัตถุที่เลือกจะเคลื่อนที่ไปทางขวา และเมื่อกดตัวอักษร B วัตถุจะเคลื่อนที่ไปทางซ้าย และเมื่อกดตัวอักษร C และ D จะเคลื่อนที่ขึ้นลงตามลำดับ โดยทุกครั้งที่มีการวาดภาพจะมีการตรวจสอบการชนกันระหว่างวัตถุต่างๆ ทุกครั้ง ซึ่งถ้าหากมีการชนก็จะไม่ทำการย้ายตำแหน่งของวัตถุไปยังตำแหน่งใหม่ ซึ่งอยู่ใน Test\TestBasicCollision\

4.9. การทดสอบการตรวจสอบการชนกันแบบละเอียดระดับรูปทรงสามเหลี่ยม

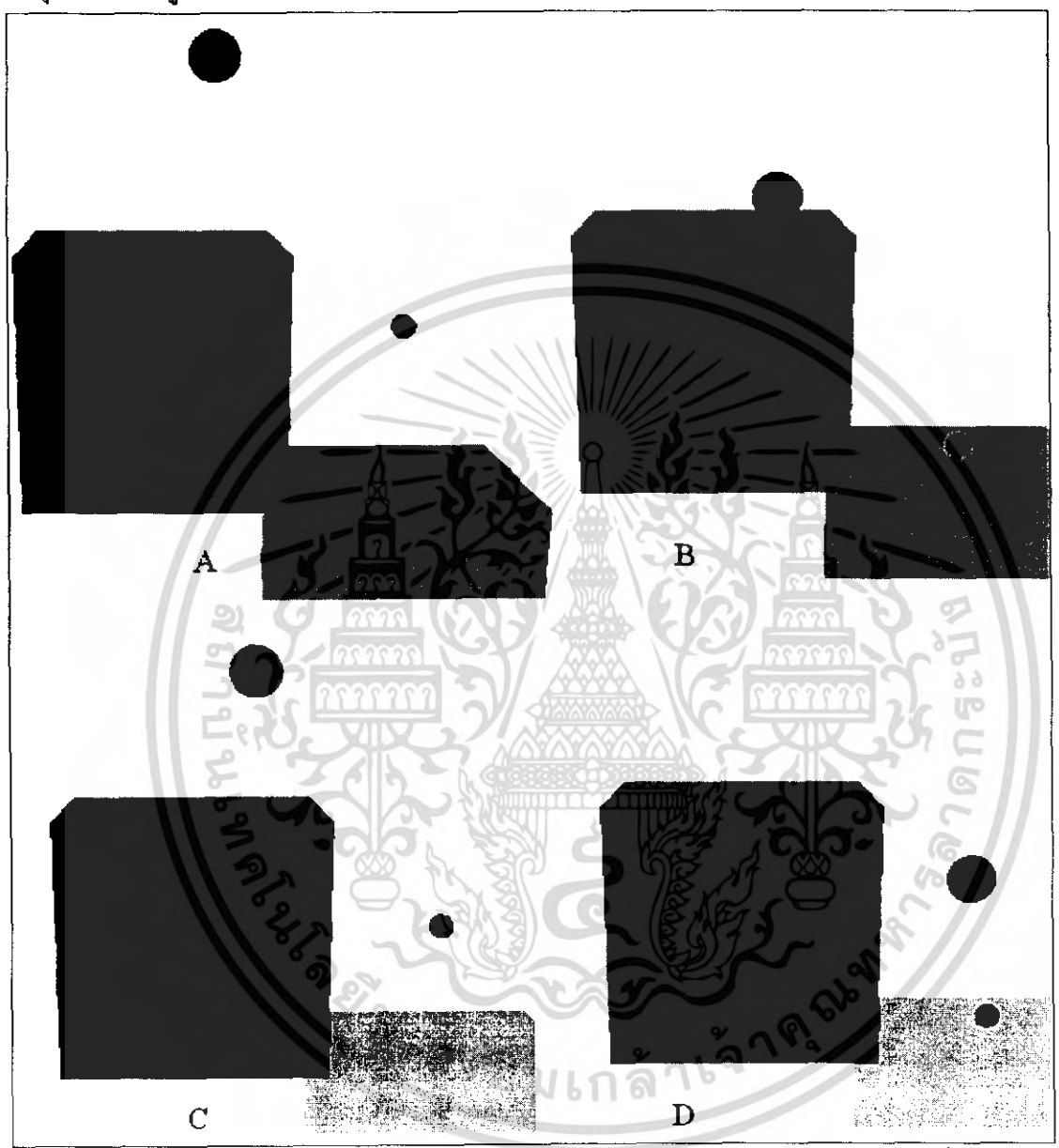
การทดสอบนี้เป็นการทดสอบความสามารถในการตรวจหาการชนกันในระดับสามเหลี่ยมที่เป็นออกประกอบของแต่ละวัตถุ ซึ่งผลลัพธ์ได้ดังรูปที่ 4.12

รูปที่ 4.12 แสดงผลลัพธ์จากการทดสอบการตรวจหาการชนกันแบบละเอียดระดับรูปทรงสามเหลี่ยม

จากรูปที่ 4.12 ผลลัพธ์ ที่ได้คือ รูปทรงสามเหลี่ยมของวัตถุที่ Intersect กันนั้นจะทำการเปลี่ยนสี ซึ่งในส่วนนี้เป็นส่วนที่ผู้พัฒนาโปรแกรมเป็นผู้กำหนดว่าต้องการให้เมื่อชนกันแล้วจะทำอะไรต่อไป จากการทดสอบนี้ ทำให้เราสามารถหาการชนกันของวัตถุที่มีรูปทรงไม่ใช่รูปทรงเลขาคณิตได้ แต่ถ้านำไปใช้กับวัตถุที่ประกอบไปด้วยสามเหลี่ยมจำนวนมากจะทำให้การประมวลผลนั้นใช้เวลามากขึ้น สามารถดูได้ใน Test\TestCollision\

4.10. การทดสอบการเคลื่อนที่ของวัตถุ แข็ง

ผลการทดสอบการเคลื่อนที่ของวัตถุแข็ง ที่กระทบลงบนกล่องโดยมีแรงดึงดูดกระทำกับวัตถุเป็นไปตามรูปที่ 4.13



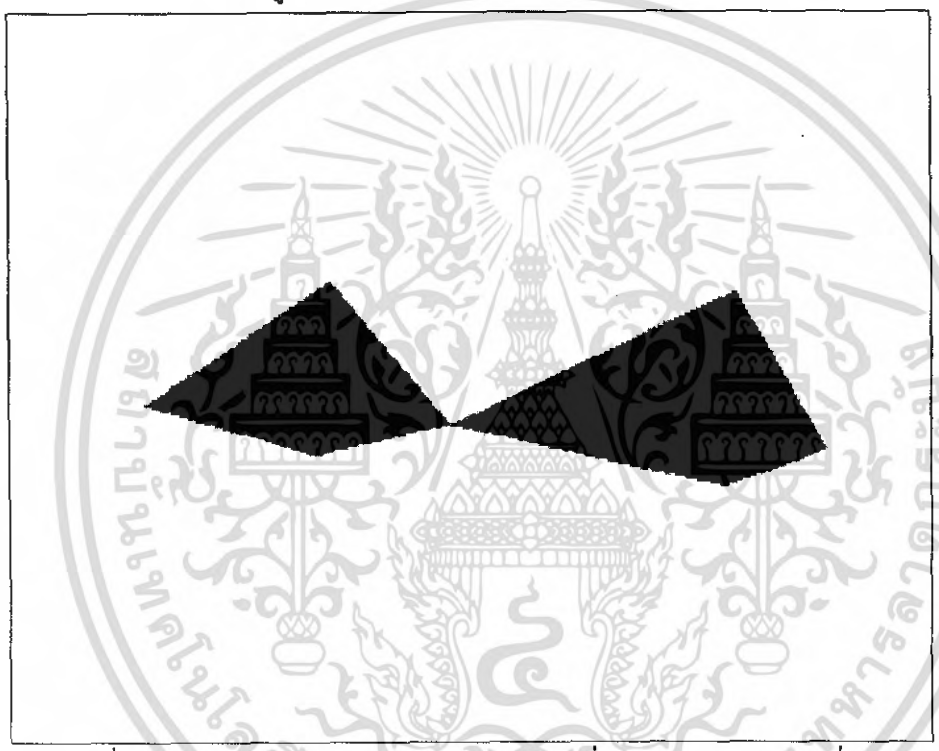
รูปที่ 4.13 แสดงการเคลื่อนที่ของลูกบอลและการตรวจสอบการชนของบอลกับกล่องซึ่ง A ลูกบอลเริ่มเคลื่อนที่ลงตามแรงโน้มถ่วง B ลูกบอลกระทบกับกล่อง C ลูกบอลเคลื่อนที่ขึ้นจากการกระทบกล่อง และ D เมื่อมีการคลิกที่ลูกบอลทำให้ลูกบอลเคลื่อนที่ไปทางขวาตกไปยังกล่องข้างล่าง

จากรูปที่ 4.13 จะเห็นได้ว่าลูกบอลสามารถเคลื่อนที่ได้โดยการจำลองสภาพแรงโน้มถ่วงให้กับมัน ซึ่งเราได้ใส่มวลให้กับลูกบอลทำให้ลูกบอลเคลื่อนที่ลงตามแรงโน้มถ่วง และเมื่อกระทบกับกล่อง ก็มีแรงที่ลูกบอลกระทำกับกล่องแต่เนื่องจากเราไม่ได้ทำการใส่มวลให้กับกล่อง ทำให้

กล่องไม่สามารถเคลื่อนที่ได้ดังนั้นแรงทั้งหมดจึงกลับไปที่ลูกบอลทำให้ลูกบอลเคลื่อนที่ขึ้น แต่ในการชนแต่ละครั้งจะมีการสูญเสียแรงทำให้ท้ายที่สุดลูกบอลจะหยุดนิ่งบนกล่อง แต่เมื่อนำเมาส์ไปคลิกที่ลูกบอล จะมีแรงไปกระทำให้ลูกบอลเคลื่อนที่ไปทางซ้าย โดยจะเคลื่อนที่ไปบนพื้นผิวของกล่อง และไม่จมลงไปในกล่อง ดูได้ใน Test\TestBallBox

4.11. การทดสอบการตรวจสอบการชนกันของรูปทรงสามเหลี่ยม

การทดสอบนี้เป็นการตรวจสอบการชนระหว่างรูปทรงสามเหลี่ยมกับรูปทรงสามเหลี่ยมในระบบ สามมิติ ซึ่งเป็นไปตามรูปที่

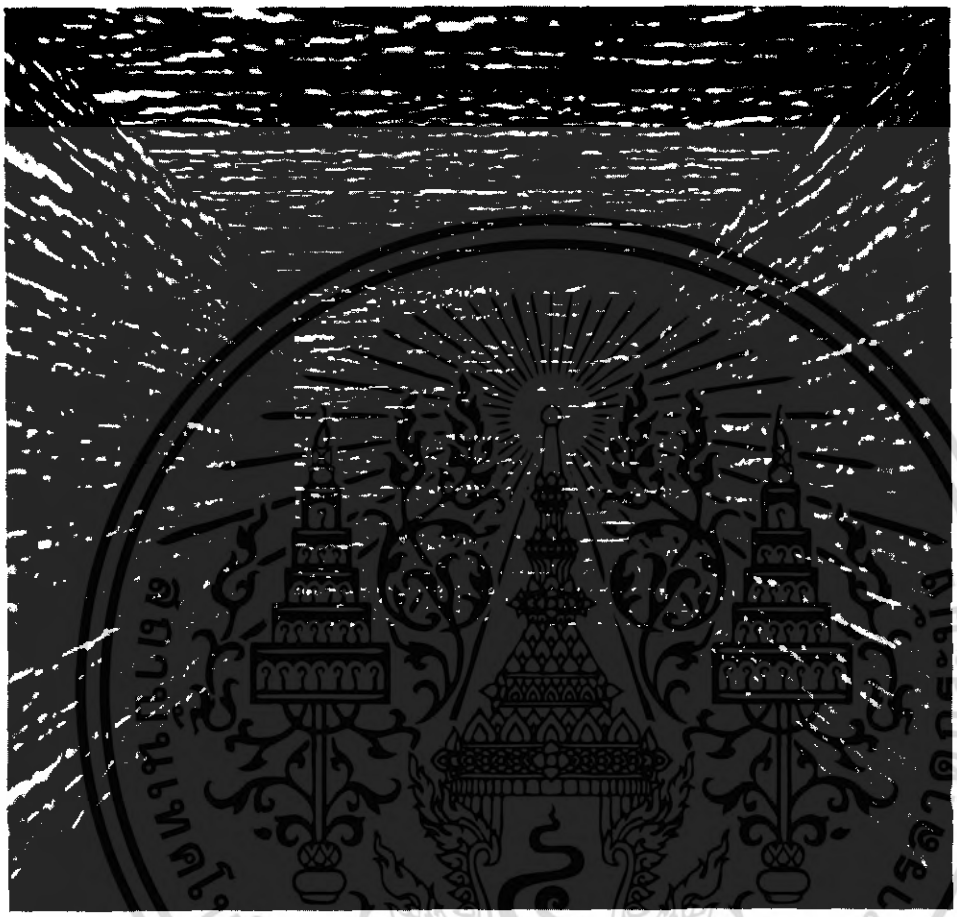


รูปที่ 4.14 แสดงการชนกันของรูปทรงสามเหลี่ยมกับรูปทรงสามเหลี่ยม

ในการทดสอบสามารถทำการเคลื่อนรูปทรงสามเหลี่ยมได้โดยการกดอักษร A และ B โดยวัตถุจะเคลื่อนที่ทางซ้ายละขวาตามลำดับ ซึ่งเมื่อเคลื่อนที่ไปชนกับรูปทรงสามเหลี่ยมก็จะมีการผลักรูปทรงสามเหลี่ยมออกมา โดยสามารถดูได้ใน Test\TestTriInt

4.12. การทดสอบแรงปฏิกิริยาหลังการชนระหว่างลูกบอลในทุกทิศทาง

ในการทดสอบนี้จะมีลูกบอลในกล่องทั้งหมด 6 ลูก โดยแต่ละลูกจะมีมวลเท่ากันและความเร็วต้นไม่เหมือนกันพร้อมทั้งอยู่คนละตำแหน่งกันภายในกล่อง โดยมีแรงดึงดูดกระทำกับลูกบอลทุกลูก ดังรูปที่ 4.15



รูปที่ 4.15 แสดงผลการทดสอบ การชนกันและแรงปฏิกิริยาหลังการชนของบอลกับกล่อง

ผลที่ได้ คือ ลูกบอลเคลื่อนที่ด้วยความเร็วต้นและชนกับบอลลูกอื่น โดยที่หลังการชนนั้นเกิดการถ่ายแรงให้กัน ทำให้มีผลกับความเร็วและทิศทางของลูกบอลที่ชนกันทั้งสองลูก ร่วมไปถึงเมื่อมีการชนระหว่างลูกบอลกับผนังก็เช่นกัน แล้วหลังจากนั้นลูกบอลจะเริ่มเคลื่อนที่ช้าลงเนื่องจากมีความเฉื่อยและแรงเสียดทาน ซึ่งเป็นส่วนที่ ผู้พัฒนาจะเป็นคนจัดการ ดูตัวอย่างได้จาก

Test/TestBallPhysic

บทที่ 5
บทวิจารณ์และสรุป

5.1 บทสรุป

ในปัจจุบันการสร้างเกมส์นั้นต้องทำการเขียนโค้ต ติดต่อกับ Hardware หลายส่วน ร่วมไปถึงการเขียนโค้ตเกี่ยวกับ การนำเข้า โมเดล และการจำลองการเคลื่อนที่ต่างๆ ทำให้กว่าที่ผู้พัฒนาจะสร้างเกมส์ขึ้นมาได้นั้นใช้เวลานาน

ดังนั้น เกมส์เอนจินที่พัฒนาขึ้นสามารถช่วยให้ผู้พัฒนาเกมส์สร้างเกมส์ได้ง่าย และทำให้โค้ตของเกมส์นั่นสั้นลง ง่ายต่อการตรวจสอบและแก้ไข โดยภายในเกมส์เอนจินจะประกอบไปด้วย Tool ต่างๆให้ผู้พัฒนานำไปใช้ ซึ่งผู้พัฒนาไม่จำต้องเขียนโค้ตเองทั้งหมด แต่เกมส์เอนจินก็มีความยืดหยุ่นที่อนุญาตให้ผู้พัฒนาโปรแกรมนั้น เขียนส่วนเพิ่มเติมที่นอกเนื่องจากที่มีในเกมส์เอนจินได้

5.2 วิจารณ์สิ่งที่ได้จากโครงงาน

จากที่ทำการพัฒนาเอนจินมา พบว่าเอนจินนั้นสามารถทำงานได้ในแบบพื้นฐานทั่วไปและยังมีบางส่วนที่ยังไม่สมบูรณ์ เนื่องจากในการพัฒนาเกมส์เอนจินนั้นถ้าต้องการให้ออกมาสมบูรณ์แบบ ควรจะทำการศึกษาและพัฒนาเฉพาะส่วนที่มีอยู่ในเกมส์เอนจิน เพื่อให้ได้ผลลัพธ์ในส่วนนั้นดีที่สุด ซึ่งถ้าแบ่งกันพัฒนาแต่ละส่วนแล้วนำมาร่วมกันก็จะได้เกมส์เอนจินที่ออกมาสมบูรณ์แบบในทุกด้าน อย่างไรก็ตามโครงงานที่ได้ถือว่าประสบความสำเร็จในระดับหนึ่งโดยสามารถนำไปสร้างตัวอย่างเกมส์เล็กๆ ได้ถึงแม้จะไม่สะดวกนักก็ตาม เนื่องจากชิ้นงานยังไม่ได้รับการปรับแต่งให้ง่ายต่อการใช้

5.3 ปัญหาอุปสรรค์และแนวทางแก้ไข

5.3.1 การใช้ Scene Graph แม้จะช่วยจัดการเรื่องของ Transformation ได้ดี แต่การแสดงผลในแต่ละโหนดนั้นได้เป็นไปตามลำดับล่วงหน้าแล้ว (Depth First Traverse) ถึงแม้ว่า Depth Buffer จะช่วยแก้ปัญหาใน Depth Sorting ได้แต่ในบางกรณีมีความจำเป็นที่ต้องแสดงผลตามลำดับก่อนหลัง เช่น การแสดงภาพวัตถุ โปร่งแสงเป็นต้นทำให้ไม่สะดวกต่อผู้ใช้ แนวทางแก้ไขคือ มีอัลกอริทึม ที่ช่วยในการจัดเรียง ความลึกของวัตถุมาช่วยจัดการลำดับของโหนด ใน Scene Graph เช่น BSP Tree เป็นต้น

5.3.2. นอกจากข้อแรก ลำดับในการวาดของ Scene Graph ที่กำหนดให้สร้าง Vertex และ index Buffer สำหรับแต่ละวัตถุแล้วทำการวาด พบได้ว่าในบาง Graphic API สามารถที่จะ Render วัตถุ หรือ Primitive ในปริมาณมากๆ ต่อการวาด หนึ่งครั้งได้ดีกว่าการแสดงผลทีละน้อย และ ยังสูญเสียเวลาในการเปลี่ยนแปลง Render State แนวทางการแก้ปัญหา มีการสะสมวัตถุที่มีคุณสมบัติคล้ายกันเช่นใช้ Texture เดียวกัน หรือใช้ Render State เหมือนกันรวมเข้าไว้ในการวาดเพียงครั้งเดียว

5.3.3. ในการพัฒนาเครื่องที่ใช้ร่วมกับโปรแกรมสามมิติ อื่นๆ เช่นการพัฒนาเครื่องมือที่ใช้การการ Export file จาก 3Ds Studio Max มีความยากลำบากในเรื่องของระบบพิกัดแกนไม่เหมือนกันเป็นผลให้การพัฒนาเป็นไปโดยไม่สมบูรณ์ แนวทางแก้ปัญหาเพิ่มเติมส่วนที่ช่วยในการแปลงระบบพิกัดขึ้น

5.3.4. การทำ Global Effect หรือ Effect ที่มีผลมาจากการแสดงวัตถุอื่นหรือมีผลกับการแสดงวัตถุอื่น สามารถทำได้ไม่สะดวกนัก ไม่มีทางแก้ไขให้สะดวกขึ้นได้ เนื่องจาก Scene Graph มีลำดับการวาด นอกเสียจากจะต้องสร้าง เมธทอด ที่ใช้ในการวาดและจัดการผลที่เกิดขึ้นเหล่านั้นด้วยตนเองซึ่งผู้พัฒนาจะต้องมีความรู้เกี่ยวกับระบบของเอนจินได้เป็นอย่างดี

5.3.5. ในการกำหนด Force และ Torque ให้กับ Rigid Body ตั้งแต่สองชิ้นขึ้นไป ถ้าต้องการให้ Force และ Torque แตกต่างกันในการออกแบบกำหนดให้ผู้ใช้สามารถใส่ฟังก์ชั่นในการคำนวณ Force และ Torque เองได้เพื่อความยืดหยุ่น แต่ก็ส่งผลให้ ต้องทำการสร้างฟังก์ชั่นสำหรับ Force และ Torque ขึ้นมาใหม่แล้วกำหนดให้กับ Rigid Body ที่เราต้องการ ทำให้ไม่สะดวกนัก แนวทางการแก้ไข ทำการสร้าง State ขึ้นมาว่าในแต่ละ State ต้องการให้ Rigid Body ชิ้นนั้นมี Force และ Torque เป็นอย่างไร โดย State นี้จะถูกเก็บอยู่ใน ฟังก์ชั่น Force และ Torque ด้วย ทำให้เราสามารถมีฟังก์ชั่น Force และ Torque เพียงฟังก์ชั่นเดียวในการกำหนดแรงให้กับวัตถุหลายชิ้น

5.3.6. ในการคำนวณ Inertia รูปทรงพื้นฐานต่างๆ ควรจะมีฟังก์ชั่นในการคำนวณอย่างง่าย เช่น วงกลม กล่อง สามเหลี่ยม

5.3.7. Input Register ของ Vertex Shader มีความหลากหลายมากกว่า Vertex Data Format ที่ใช้ในปัจจุบันส่งผลให้ใช้ความสามารถของ Shader ได้ไม่เต็มที่ เนื่องจากการออกแบบ

ส่วนที่สำหรับสนับสนุน Shader นั้น กระทำขึ้นหลังจากส่วนของ Scene Graph ที่ดำเนินงานได้อย่างสมบูรณ์แล้วจึงพยายามที่จะคงรูปแบบเดิมไว้ แนวทางแก้ไขใช้รูปแบบการกำหนดVertex Format ใหม่สำหรับการแสดงผลที่ใช้ Shader (ปัญหานี้เกิดขึ้นใน DirectX)

5.4 แนวทางพัฒนาต่อ

5.4.1. ทำการปรับแต่ง เอนจินให้ใช้งานได้ง่ายขึ้น รวมไปถึงแก้ปัญหาต่างๆ ในส่วนที่กล่าวมาแล้ว

5.4.2. เพิ่มเทคนิคพิเศษต่างๆ ที่เป็นที่นิยม เนื่องจากผู้พัฒนามีเวลาไม่มากพอในการพัฒนา

5.4.3. เทคนิค CLOD (Continuous Level Of Detail) ที่ใช้อยู่ในตอนนี้นั้นเป็นเทคนิคที่ค่อนข้างล้าหลังเนื่องจาก Graphic Card ในปัจจุบันมีความสามารถในการแสดงผลมากขึ้น อีกทั้งยังมี Pixel และ Vertex Process ที่ดียิ่งขึ้น จึงควรที่จะปรับปรุงวิธีการใหม่และนำไปประมวลผลบน Graphic Card แทนที่จะคำนวณบน CPU

5.4.4. ในการพัฒนาเกมส์จริงจริงนั้นคือการทำ interpolation ในรูปแบบต่างๆจึงสมควรที่จะมีคลาสที่สนับสนุนเรื่องเหล่านี้ไว้ด้วย

5.4.5. ย้ายการคำนวณเกี่ยวกับ Skin ใน Skeletal animation ไว้บน GPU

5.4.6. ในการพัฒนาเกมส์ การออกแบบฉากมีความสำคัญจึงควรพัฒนาเครื่องมือในการออกแบบฉากและพัฒนาในส่วนของ Scene Graph ในสามารถ บันทึก หรือ โหลด ฉาก ในปัจจุบันได้

5.4.7. เพิ่มการคำนวณ ฟิสิกส์ที่หันสมัยอื่นๆ เช่น แรงลอยตัว ของวัตถุในของเหลว ข้อต่อรูปแบบต่างๆ รวมถึง ฟิสิกส์สำหรับ ยานพาหนะ

บรรณานุกรม

[1] David Baraff and Andrew Witkin "Physically Based Modeling" Carnegie Mellon University.

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

[3] John P. Flynt, 2005, "Software Engineering for Game Developers", Thomson Course Technology, Premier Press.

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

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

[6] Gregory Snook, 2005, "Real-Time 3D Terrain Engine using C++ and DirectX9", Game Development Series, Charles River Media.

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

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

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