การพัฒนาเกม 3 มิติ

3D Game Development

นายนิธิวัฒน์ เสนาะน้อย
นายพลเดช วรสุทธยางกูร

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2544

การพัฒนาเกม 3 มิติ

3D Game Development

นายนิธิวัฒน์ เสนาะน้อย

นายพลเดช วรสุทธยางกูร

อาจารย์ที่ปรึกษา

ดร. วรวัฒน์ ลิ้มโภคา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

ปริญญานิพนธ์ปีการศึกษา 2544

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาเกม 3 มิติ

3D Game Development

ผู้จัดทำ

1. นาย นิธิวัฒน์ เสนาะน้อย รหัสประจำตัว 41014222

2. นาย พลเดช วรสุทธยางกูร รหัสประจำตัว 41014300

อาจารย์ที่ปรึกษา

(ดร.วรวัฒน์ ลิ้มโภคา)

การพัฒนาเกม 3 มิติ

นายนิธิวัฒน์ เสนาะน้อย
นายพลเดช วรสุทธยางกูร
ดร.วรวัฒน์ ลิ้มไภคา
อาจารย์ที่ปรึกษา
ปีการศึกษา 2544

บทคัดย่อ

การพัฒนาเกม 3 มิติ โดย เป็นโครงงานที่กล่าวถึง การพัฒนาโปรแกรมคอมพิวเตอร์ประเภทเกม 3 มิติ โดยใช้ไดเร็กเอ็กซ์มาเป็นส่วนประกอบของโปรแกรมในด้านมัลติมีเดีย ซึ่งเริ่มต้นตั้งแต่การพัฒนาเกมเอนจินต์ซึ่งเป็นหัวใจสำคัญสำหรับการสร้างเกม สำหรับการเขียนโปรแกรมเพื่อพัฒนาเกม 3 มิติ ทั้งหมดนี้อาศัยการโปรแกรมในเชิงวัตถุซึ่งภาษาที่ใช้ในการพัฒนาคือ "ซีพลัสพลัส"

ขอบเขตการทำงานของโครงงานนี้: ในขั้นแรก, จะทำการศึกษาถึงสถาปัตยกรรมของไดเร็กเอ็กซ์ และวิธีการนำไดเร็กเอ็กซ์มาช่วยในการพัฒนาโปรแกรมเกมในด้านต่างๆ อาทิ การแสดงผลทางด้านภาพ, เสียง และการติดต่อกับอุปกรณ์อินพุต รวมถึงศึกษาทฤษฎีพื้นฐานที่จะนำไปสู่การพัฒนาเกมเอนจินต์ จากนั้นจึงทำการออแบบวางแนวความคิดโครงสร้างและส่วนประกอบของเกมเอนจินต์ และเริ่มพัฒนาส่วนประกอบของเอนจินต์ควบคู่ไปด้วยเพื่อศึกษาถึงผลลัพธ์ของวิธีการต่างๆแล้วเลือกใช้อย่างเหมาะสม โดยเริ่มจากด้านกราฟิก, เสียง, อินพุตของเกม และส่วนประกอบอื่นๆที่จำเป็นในตอนเริ่มต้นของการพัฒนาเกมเอนจินต์ รวมถึงเริ่มทำการสร้างและการจัดการงานด้านโมเดล 3 มิติ

ปริญญานิพนธ์ฉบับนี้ เป็นส่วนหนึ่งของปริญญานิพนธ์ในโครงงานเดียวกันทั้งหมด 4 ฉบับ ในฉบับนี้ได้แสดงในส่วนของการพัฒนาส่วนสำหรับการจัดการออดิโอ,อินพุตและมิเดีย

ในขั้นที่สองถึงปรับปรุงเพิ่มเติมส่วนอื่นๆเพื่อความสมบูรณ์ของเกมเอนจินต์ ในขณะเดียวกันก็จะเป็นการนำเอาเกมเอนจินต์ที่ได้พัฒนาแล้วนั้น มาสร้างผลงานที่เป็นเกม 3 มิติ สุดท้ายเป็นการนำผลงานไปทดสอบกับฮาร์ดแวร์คอมพิวเตอร์อื่นๆ เพื่อหาประสิทธิภาพและจัดทำส่วนติดตั้งโปรแกรมให้สมบูรณ์

3D Game Development

Nitiwat Sanornoi

Phondech Warasutthayanggoonr

Dr. Voravat Limpoka

Advisor

ABSTRACT

Developing 3D Games is the project that describes the use of DirectX Application Programming Interface (API) as the multimedia component in Developing 3D Game which is starting with game engine development that is the core of game creating. The object oriented programming language in "C++" is used for developing this project.

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

The 3D Game project consist of four part, this thesis is one part of Audio, Input and Media.

After that, Integrate all engine components for engine testing and improving to final completed. In the meantime create 3D games by using the developed 3D games project . Finally, test the game with various hardware for measuring performance and packaging the 3D game so that ready to installation.

กิตติกรรมประกาศ

ขอขอบพระคุณอาจารย์วรวัฒน์ ลิ้มโภคา ที่สละเวลาให้คำปรึกษาช่วยเหลือและแนะนำในการจัดทำวิทยานิพนธ์เป็นประจำทุกสัปดาห์ซึ่งเป็นแนวทางในการพัฒนา และเอาใจใส่ต่อวิทยานิพนธ์ซึ่งทำให้พวกข้าพเจ้ามีกำลังใจในการทำงานจนสำเร็จ

และต้องขอขอบคุณเหล่าเพื่อน และพี่ที่ทำวิทยานิพนธ์ในเรื่องเดียว ที่ร่วมกันทำงาน ช่วยเหลือและร่วมทุกข์ร่วมสุข นางสาวนิสา พัฒนพงษ์อนันต์ และนายพิพัฒน์ ศรีมัธยกุล ที่ให้ความช่วยเหลือในการแปล หากไม่ได้ความร่วมมือจากเพื่อนและพี่ วิทยานิพนธ์ของข้าพเจ้าคงไม่อาจสำเร็จลงได้

ขอขอบคุณกลุ่มวิทยานิพนธ์ในห้อง OLALA ที่เอื้อเฟื้ออาหารและเครื่องดื่ม และช่วยผ่อนคลายความเครียดในการจัดทำ

ขอขอบพระคุณบิดาและมารดาของข้าพเจ้า ที่ให้กำลังใจและสนับสนุนข้าพเจ้าอยู่เบื้องหลังเสมอตลอดมา

นิธิวัฒน์ เสนาะน้อย

พลเดช วรสุทธยางกูร

สารบัญ

สารบัญตาราง

หน้าที่

สารบัญภาพ

หน้าที่

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

ในปัจจุบันการพัฒนาทางด้านเทคโนโลยีคอมพิวเตอร์นั้นได้ก้าวหน้าไปอย่างรวดเร็ว โดยเฉพาะในด้านเกมคอมพิวเตอร์ ฮาร์ดแวร์ได้มีการพัฒนาระบบการแสดงผลให้สามารถประมวลผลภาพในลักษณะสามมิติ ที่มีทั้งความเร็วและความสวยงาม ไม่ว่าจะเป็นการ์ดแสดงผลสามมิติที่จัดการด้านการแสดงผลสามมิติโดยตรง หรือหน่วยประมวลผลกลางที่พัฒนาคำสั่งต่างๆ มารองรับการประมวลผลข้อมูลสามมิติ ทางด้านซอฟแวร์ก็มีโปรแกรมต่างๆเข้ามาช่วยสร้างสรรค์ภาพสามมิติ ทำให้มีการสร้างเกมในลักษณะสามมิติออกมามากมาย ทั้งนี้ทั้งนั้น ส่วนที่ช่วยให้การทำเกมนั้นสำเร็จโดยง่าย หรือผู้อยู่เบื้องหลัง ส่วนหนึ่งก็คือ "เกมเอนจินต์" และ "ไดเร็คเอ็กซ์"

เกมเอนจินต์นั้น ช่วยอำนวยความสะดวกให้กับการสร้างเกมอย่างมาก มีฟังก์ชันเกี่ยวกับเกมต่างๆไว้ให้เรียกใช้งาน ช่วยในการสร้างภาพเคลื่อนไหวในเกม ส่วนไดเร็คเอ็กซ์นั้น เป็นตัวกลางช่วยในการติดต่อกับฮาร์ดแวร์ เพื่อการจัดการกับภาพ เสียง การรับอินพุต ได้โดยตรงและรวดเร็ว

เกมเอนจินต์ที่ดี นอกจากต้องใช้งานง่ายแล้ว เรื่องความเร็วในเกมก็เป็นสิ่งสำคัญ ความรู้พื้นฐานในเรื่องการทำเกมสามมิติจึงเป็นสิ่งสำคัญ ที่ต้องนำมาใช้พัฒนาเกมสามมิติให้มีประสิทธิภาพ ไม่ว่าจะเป็นเทคนิคในการคำนวณทางคณิตศาสตร์ ไปจนถึงอัลกอริทึมในการเขียนโปรแกรมกับระบบสามมิติ ดังนั้นในการพัฒนาเกมสามมิติให้ได้ดียิ่ง การศึกษาให้ได้ลึกและใกล้ชิดอย่างหนึ่งก็คือ เริ่มตั้งแต่การทำเกมเอนจินต์ขึ้นมาเอง และใช้เกมเอนจินต์นั้นในการพัฒนาเกมสามมิติต่อไป

การทำงานวิจัยนี้ นอกจากได้ศึกษาการใช้งานไดเร็คเอ็กซ์ในติดต่อกับฮาร์ดแวร์มัลติมีเดียและนำมาใช้ในการดึงความสามารถทางฮาร์ดแวร์ต่างๆได้อย่างเต็มประสิทธิภาพแล้ว ยังได้ศึกษาเกี่ยวกับการประมวลผลภาพสามมิติพื้นฐาน เช่น พิกัดในระบบสามมิติ การหมุนภาพสามมิติ การทำภาพเคลื่อนไหว และเทคนิคต่างๆที่ใช้ในการประมวลผล เช่น การทำงานกับมุมกล้อง การประมวลผลภาพเคลื่อนไหวให้รวดเร็ว การดึงความสามารถทางด้านฮาร์ดแวร์และซอฟแวร์ของคอมพิวเตอร์ออกมาใช้อย่างเต็มที่ เป็นต้น

1.2 วัตถุประสงค์ของงานวิจัย

1.2.1 เพื่อศึกษาและพัฒนาเกม โดยนำเสนอในรูปแบบ 3 มิติ

1.2.2 ศึกษาการทำงานของไดเร็คเอ็กซ์ ในการติดต่อกับอุปกรณ์ฮาร์ดแวร์ ด้านมัลติมิเดีย เช่น การ์ดจอ การ์ดเสียง อุปกรณ์อินพุท เป็นต้น เพื่อเป็นพื้นฐานในการพัฒนาเกมสามมิติ

1.2.3 นำไดเร็คเอ็กซ์ มาใช้ในการพัฒนาเกมสามมิติ

1.2.4 ออกแบบและพัฒนาเกมเอนจินต์ โดยใช้ไดเร็คเอ็กซ์ และ ซีพลัสพลัส

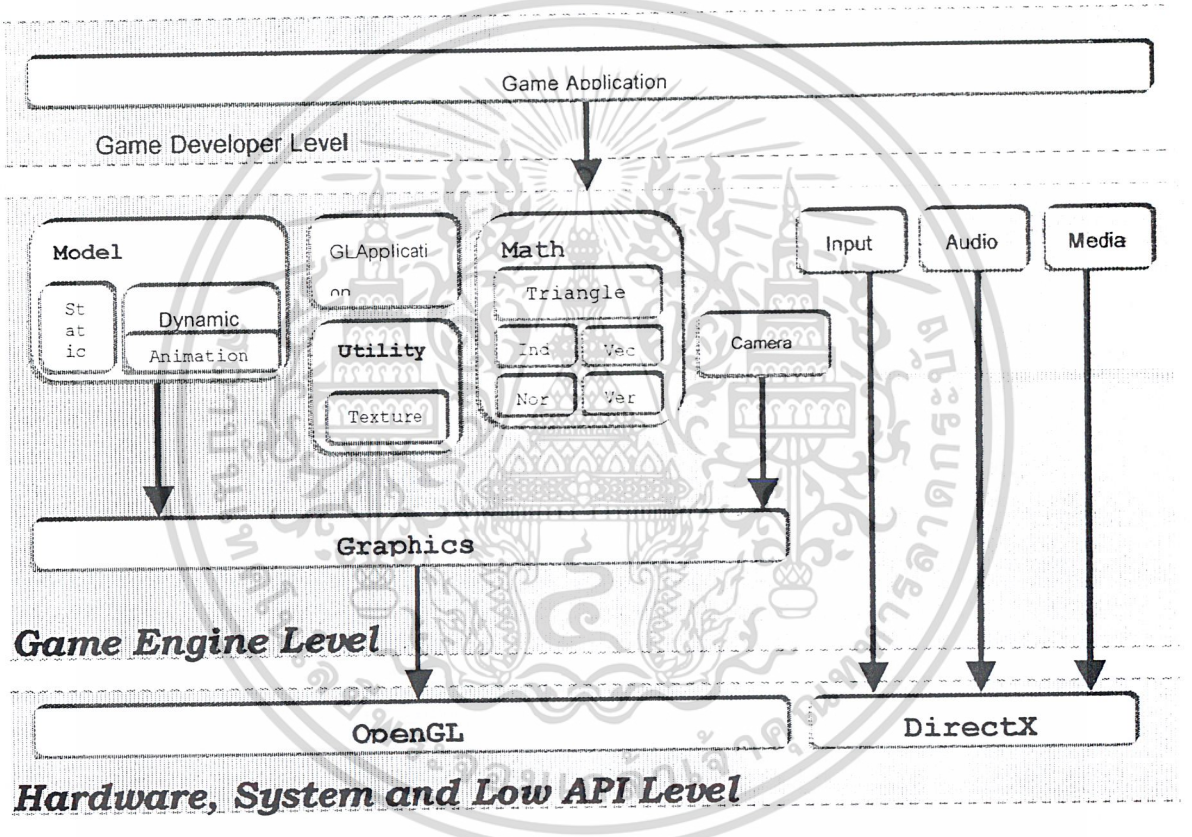
1.2.5 สร้างเกมสามมิติ จากเกมเอนจินต์ที่ได้ออกแบบเสร็จแล้ว

1.2.6 เพื่อพัฒนา Library tool รวมถึง Software tool ที่จะใช้ในการพัฒนาเกม

1.3 ขอบเขตของงานวิจัย

งานวิจัยนี้เน้นที่การศึกษาและสร้างเกมเอนจินต์สามมิติ โดยใช้เทคโนโลยีไดเร็คเอ็กซ์ และซีพลัสพลัส โปรแกรมมิ่ง เริ่มตั้งแต่การออกแบบเอนจินต์ การสร้างเกมเอนจินต์โดยหลักการโปรแกรมเชิงวัตถุ จนสามารถนำมาช่วยสร้างเกมสามมิติได้ จากนั้นนำเกมเอนจินต์ที่ได้สร้างเสร็จแล้ว มาสร้างเกมสามมิติขึ้นมา เพื่อทดสอบใช้งานเอนจินต์ และหาข้อบกพร่อง เพื่อนำไปสู่การพัฒนาให้สมบูรณ์ยิ่งขึ้นต่อไป

สำหรับในงานวิจัยนี้ได้เกิดจากการรวมตัวของกลุ่มนักศึกษาที่มีความสนใจในด้านเดียวกันหลายกลุ่ม จึงได้แบ่งกลุ่มร่วมกันทำงานเป็น 4 กลุ่มย่อย หลังจากได้มีการออกแบบร่วมกันแล้ว ได้มีการแบ่งงานกัน โดยนักศึกษาในกลุ่มนี้ได้รับผิดชอบในส่วน ระบบออดิโอ อินพุต และมิเดีย ดังแสดงในรูปที่ 1-1



รูปที่ 1.1 แสดงโครงสร้างของเกมเอนจินต์ และส่วนที่กลุ่มรับผิดชอบ

1.4 ประโยชน์ที่คาดว่าจะได้รับ

1.4.1 ทักษะในการพัฒนาเกม โดยนำเสนอในรูปแบบ 3 มิติ เช่น การสร้างตัวละคร การวางโครงเรื่องของเกม การสร้างภาพสามมิติ เป็นต้น

1.4.2 ความรู้ในด้านการประมวลผลภาพสามมิติ เช่น การหมุนภาพ การสร้างภาพเคลื่อนไหว การใช้คณิตศาสตร์มาช่วยจัดการภาพสามมิติ เป็นต้น

1.4.3 เทคนิคในการเขียนโปรแกรมประมวลผลภาพสามมิติ

1.4.4 การใช้ไดเร็คเอ็กซ์ ติดต่อกับอุปกรณ์มัลติมิเดีย

1.4.5 การออกแบบโปรแกรมเชิงวัตถุ

1.4.6 การเขียนโปรแกรมซีพลัสพลัส เพื่อติดต่อกับไดเร็คเอ็กซ์

1.5 วิธีการดำเนินงาน

1.5.1 เริ่มจากการศึกษาทฤษฎีในการสร้างเกมสามมิติ เช่น ระบบพิกัด เวกเตอร์ การหมุนกล้อง ตลอดจน แนวความคิดในการออกแบบ พัฒนา ซึ่งได้กล่าวไว้ในบทที่ 2 สำหรับในบทที่ 3 นั้น กล่าวถึง เรื่อง แนวความคิดในการออกแบบ เกมเอนจินต์ สำหรับนำไปพัฒนาเกม 3 มิติ

1.5.2 ทฤษฎีที่เกี่ยวข้องกับการพัฒนาเกมเอนจินต์ ในบทที่ 2 แบ่งออกเป็น 4 กลุ่มดังนี้

กลุ่มที่ 1:

- ทฤษฎีและหลักการกล้องในระบบ 3 มิติ สำหรับเกมเอนจินต์
- ทฤษฎีและหลักการ การสร้างแอนิเมชันสำหรับเกมเอนจินต์

กลุ่มที่ 2:

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

กลุ่มที่ 3:

- การสร้างโมเดล 3 มิติ ด้วย 3D Studio MAX

กลุ่มที่ 4:

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

โดยปริญญานิพนธ์ฉบับนี้จะได้นำเสนอเนื้อหาในกลุ่มที่ 2

1.5.3 ออกแบบเกมเอนจินต์ จะอธิบายโครงสร้างของเกมเอนจินต์แต่ละส่วน และส่วนประกอบต่างๆของเกมเอนจินต์

1.5.4 พัฒนาส่วนประกอบต่างๆที่ได้ออกแบบไว้แล้ว

1.5.5 จากนั้นนำเอนจินต์ในแต่ละส่วนมาประกอบรวมกัน และกำหนดรูปแบบไฟล์ข้อมูลที่จะนำไปใช้กับเกมเอนจินต์ จัดทำแอพพลิเคชันเฟรมเวิร์ค สำหรับผู้นำเกมเอนจินต์ไปพัฒนาเกมในขั้นตอนต่อไป

1.5.6 เมื่อได้เกมเอนจินต์ที่ใช้งานได้แล้ว จึงนำเกมเอนจินต์ที่ได้ ไปสร้างเป็นเกมสามมิติเพื่อทดสอบการใช้งานเกมเอนจินต์

1.5.7 รวบรวมผลการทดสอบ วิเคราะห์ปัญหา และหาแนวทางการแก้ปัญหาต่างๆ เพื่อนำมาพัฒนาหรือเป็นข้อมูลในการพัฒนาขั้นต่อไป

บทที่ 2
ทฤษฎีและหลักการพัฒนาเกม 3 มิติ

2.1 ทฤษฎีและหลักการของ Direct Audio

DirectAudio เป็น API ส่วนหนึ่งของ DirectX ซึ่งจัดการเกี่ยวส่วนของเสียงหรือ Audio ซึ่งช่วยในการเขียนโปรแกรมเกมเพราะความสามารถของ DirectAudio นั้นช่วยให้การเล่นเสียง Effect และดนตรีประกอบได้ดีขึ้น เนื่องจากไม่ได้ผ่านทาง Multimedia Control API ของ Windows ซึ่งมีข้อจำกัดมากมาย เช่น สามารถเล่นเสียงได้จากแหล่งกำเนิดเดียว ไม่สามารถ Share ซึ่งเป็นข้อจำกัดอย่างยิ่งในการเขียนโปรแกรมเกม ดังนั้น DirectAudio จึงมีส่วนช่วยได้มาก

DirectAudio เป็นส่วนการพัฒนาที่มีใน DirectX8 SDK โดยเป็นการรวม DirectMusic และ DirectSound ของ DirectX SDK version ก่อนเข้าด้วยกัน ซึ่งลดความซับซ้อนลงไปได้พอสมควร แม้แต่รูปแบบของสถาปัตยกรรมก็ถูกออกแบบใหม่ แม้การทำงานเราอาจจะแยกเป็น DirectSound และ DirectMusic แต่จริงๆแล้วทั้งสองเป็นส่วนหนึ่งของ DirectAudio

ในที่นี้จะกล่าวถึง DirectAudio ในด้านที่เกี่ยวข้องกับเกมเท่านั้น

2.1.1 ความสามารถของ DirectAudio

- เล่นเสียงจาก file format wav, midi และจาก DirectMusic Producer ได้
- เล่นเสียงจากหลายๆแหล่งพร้อมๆกันได้
- บริหารเกี่ยวกับการเล่น midi ได้
- โดยปกติ การเล่นเสียง MIDI จะต้องใช้ synthesizer ของ soundcard หรือ sound module ซึ่งเสียงเครื่องดนตรีจะขึ้นอยู่กับ hardware ที่ใช้ แต่ DirectAudio มี DLS synthesizer ซึ่งทำให้ไม่ยึดติดกับ hardware สามารถใช้เล่นเครื่องใดๆก็ได้ และได้เสียงเหมือนกัน
- สามารถใส่ Effect ได้
- สามารถเล่นเสียงในสภาพแวดล้อม 3 มิติได้
- สามารถ capture เสียงจาก Capture Device ได้

2.1.2 สิ่งที่ควรรู้เกี่ยวกับ DirectAudio

Loader

เป็น object ทีสร้างไว้สำหรับดึง object ชนิดอื่นๆขึ้นมา เราอาจจะพูดได้ว่าเป็น object ที่ต้องสร้างก่อน object อื่นๆที่ต้องการการ load

ใน DirectAudio Loader จะเป็น object ชนิด IdirectMusicLoader8

Segment

เป็น object ที่เก็บลำดับของเสียงเอาไว้ โดยข้อมูลอาจจะเป็นเสียงใดๆก็ได้ เช่น Wave, MIDI หรือข้อมูลต่างๆที่อยู่ในรูป file format ของ DirectMusic Producer

Segment จะมี State ของตัวมันเอง ซึ่งสามารถดึงข้อมูลของ State มาใช้งานได้

Performance

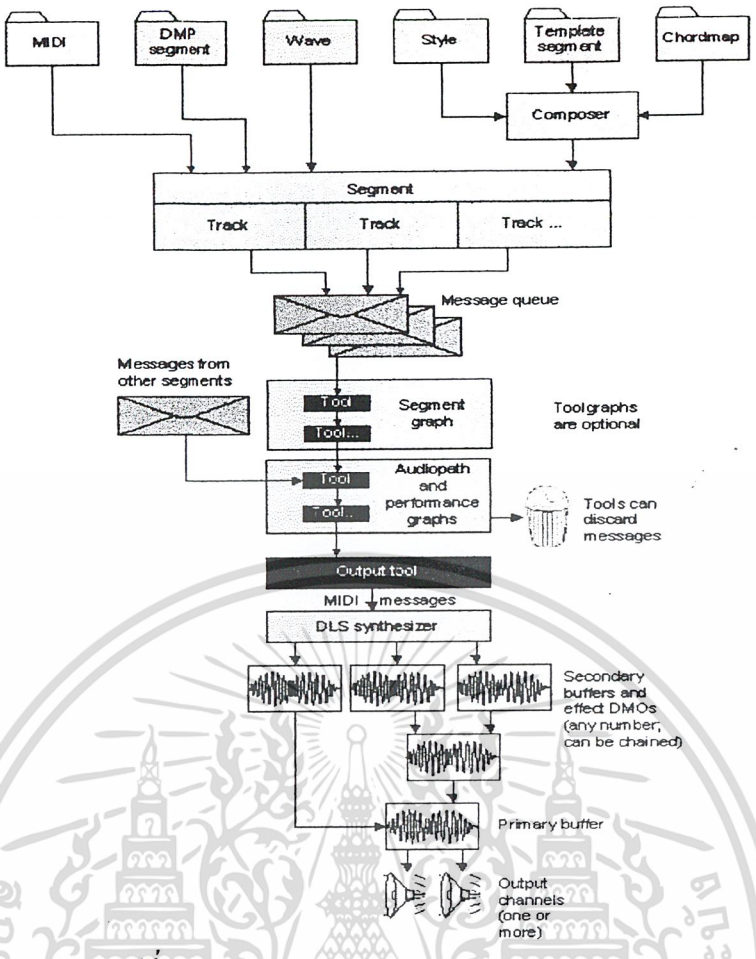
เป็น object ไว้สำหรับบริหารจัดการกระแสข้อมูลเสียงจากแหล่งกำเนิด เช่น การเล่นเสียง การหยุดเล่น ฯลฯ

Audiopaths และ Buffers

Segment จะเล่นเสียงบน Audiopath ซึ่งควบคุมกระแสของข้อมูลจาก Performance ไปยังตัวสังเคราะห์เสียง,แล้วไปยัง Buffer ซึ่ง Effect หรือตำแหน่งในรูป 3 มิติหรือทั้งสองได้ถูกใส่เข้าไป ก่อนที่จะผสมเสียงต่างๆขั้นสุดท้ายใน Primary Buffer

Buffer ของเสียงจะมีทั้ง Primary Buffer ซึ่งเป็นส่วนสุดท้ายและมีเพียงหนึ่งเดียว ซึ่งจะผสมเสียง (Mix) ขั้นตอนสุดท้าย และ Secondary Buffer ซึ่งเป็นส่วนเก็บเสียงของแต่ละแหล่งกำเนิดเสียง โดย Secondary Buffer นั้นสามารถปรับแต่งได้ เช่น ใส่ Effect

2.1.3 Overview ของ DirectAudio



รูปที่ 2.1 แสดง Architecture ของ Direct Audio

ที่มาของข้อมูลจะมาจาก MIDI, Wave, Segment ต่างๆ ซึ่งอาจจะประกอบไปด้วย track หลายๆ track ซึ่งจะเป็นข้อมูลที่ซับซ้อน ซึ่งมีการเล่นจะถูกแปลงมาเป็นลักษณะ Message ส่งมาเป็นแถวๆ โดยจะผ่าน tool ต่างๆซึ่งจะอาจจะทำการแก้ไขหรือลบ message ที่ผ่านเข้า(เช่น การใส่ Effect) โดยสุดท้ายจะแปลงเป็นรูปแบบ MIDI ในขั้นตอนสุดท้าย แล้วไปผ่านตัวสังเคราะห์เสียง แล้วไป Sound Buffer ต่างๆ ก่อนจะถูก Mix รวมที่ Primary Buffer

2.1.4 ขั้นตอนต่างๆที่เกิดขึ้นในการใช้งาน DirectAudio

การ Load object

ในการ Load object ต่างๆขึ้นมาจาก file หรือ resource ก่อนที่จะเอา object นั้นไปติดต่อกับส่วน Performance โดยต้องอาศัย IDirectMusicLoader8 ในการ load object ต่างๆ

ใน Application หนึ่งควรจะมี IDirectMusicLoader8 เพียงตัวเดียว โดยอาจจะเป็นลักษณะ Global object เนื่องจากไม่มีความจำเป็นที่จะสร้างขึ้นมาหลายๆ object แล้วยกเลิกการใช้เมื่อไม่มีการ load object ใดๆแล้ว

นี้คือตัวอย่างการสร้าง object Loader

```
IDirectMusicLoader8* m_pLoader;
```

```
CoInitializeEx(NULL, 0);
HRESULT hr = CoCreateInstance(
   CLSID_DirectMusicLoader,
   NULL,
   CLSCTX_INPROC,
   IID_IDirectMusicLoader8,
   (void**)&m_pLoader);
```

การตั้งค่าและวิเคราะห์ก่อนการ Load object

IDirectMusicLoader8 มีความสามารถในการ search ชื่อ file ในระดับโดยการ load จะ file อาจจะไม่ต้องทำการใส่ชื่อแบบเต็ม (full path) ในการ load ยกตัวอย่างเช่น

```
HRESULT mySetLoaderPath (
   IDirectMusicLoader8 *pILoader)   // Previously created.
{
   return pILoader->SetSearchDirectory(
         CLSID_DirectMusicStyle,
         L"c:\\mymusic\\funky",
         FALSE);
}
```

function นี้เป็นการตั้ง Directory ที่เริ่มต้นการค้นหาไว้ที่ c:\mymusic\funky\ ดังนั้นหากทำการ load object จาก file จะเริ่มค้นหาจาก path ที่ตั้งไว้

IdirectMusicLoader8 ยังมี method ไว้สำหรับค้นหา file คือ ScanDirectory และ method ใช้ในการวิเคราะห์รูปแบบ file คือ ParseDescriptor ซึ่งอาจจะใช้ method เหล่านี้ก่อนที่จะทำการ LoadObject จาก file

การ Enumerate object

EnumObject เป็น method ไว้ทำการ enumerate object โดยระบุประเภทที่ต้องการ ซึ่งอาจจะใช้ร่วมกับ ScanDirectory เพื่อจำกัดขอบเขตในการ Enumerate โดย ตัวอย่างข้างล่างเป็นตัวอย่างการ Enumerate Object ที่อยู่ใน Directory หนึ่ง

```
void myListStyles(
    IDirectMusicLoader *pILoader)

{
  HRESULT hr = pILoader->SetSearchDirectory(
      CLSID_DirectMusicStyle,
      L"c:\\mymusic\\wassup",
      TRUE);
  if (SUCCEEDED(hr))
  {
    hr = pILoader->ScanDirectory(
        CLSID_DirectMusicStyle,
        L"sty",
        L"stylecache");
    if (hr == S_OK)   // Only if files were found.
    {
      DWORD dwIndex;
      DMUS_OBJECTDESC Desc;
      Desc.dwSize = sizeof(DMUS_OBJECTDESC);
      for (dwIndex = 0; ;dwIndex++)
      {
        if (S_OK ==(pILoader->EnumObject(
            CLSID_DirectMusicStyle,
            dwIndex, &Desc)))
        {
          TRACE("Name: %S, Category: %S, Path: %S\n",
              Desc.wszName,
              Desc.wszCategory,
              Desc.wszFileName);
        }
        else break;
      }
    }
```

```
    }
}
```

ผลที่ได้คือรายชื่อของ file ที่เป็น class CLSID_DirectMusicStyle (ดูรายละเอียดได้ที่ dmusic.h) ซึ่งเป็น class ที่เป็น Style ของเพลงต่างๆ

การ Load object จาก file

Method LoadObjectFromFile มีไว้สำหรับ load file ขึ้นมา เช่น load file ที่เป็น MIDI file format มาเป็น Segment เพื่อนำไปเล่น ซึ่งหากมีการใช้ method SetSearchDirectory แล้วก็อาจจะไม่ต้องใส่ชื่อ file แบบ full path ก็ได้ เช่น

```
IDirectMusicSegment8 * m_pSegments[4];

static WCHAR  wszNames[4][MAX_PATH] = {
    L"AudioPath1.sgt",
    L"AudioPath2.sgt",
    L"AudioPath3.wav",
    L"AudioPath4.sgt"
};

for (DWORD dwIndex = 0; dwIndex < 4; dwIndex++)
{
    hr = m_pLoader->LoadObjectFromFile(
            CLSID_DirectMusicSegment,
            IID_IDirectMusicSegment8,
            wszNames[dwIndex],
            (void**) &m_pSegments[dwIndex]);
}
```

แสดงการ Load file ทั้ง 4 ขึ้นมาเก็บไว้ใน Array ของ Segment

ในการ Load object ที่ไม่ใช่ Segment อาจจะใช้ method GetObject เช่น ในตัวอย่างข้างล่างเป็นการ Load object ชนิด Style ขึ้นมาจาก file polka.sty จาก Directory c:\mymusic\funky\

```
void myLoadStyle(
   IDirectMusicStyle8 **ppStyle, IDirectMusicLoader8 *pLoader)
{
   if (pLoader)
   {
      DMUS_OBJECTDESC Desc;

      // Start by initializing Desc with the file name and
      // class GUID for the style object.

      wcscpy(Desc.wszFileName,L"c:\\mymusic\\funky\\polka.sty");
      Desc.guidClass = CLSID_DirectMusicStyle;
      Desc.dwSize = sizeof (DMUS_OBJECTDESC);
      Desc.dwValidData = DMUS_OBJ_CLASS |
                  DMUS_OBJ_FILENAME |
                  DMUS_OBJ_FULLPATH;

      pLoader->GetObject(&Desc, IID_IDirectMusicStyle8,
            (void **) ppStyle);
   }
}
```

การ Load object ที่ไม่ใช่จาก file

ในกรณีที่เราต้องการดึง object จาก resource ที่มีอยู่แล้วใน Memory โดยใช้ Method GetObject เหมือนกับการดึง object จาก file ซึ่งเพียงเปลี่ยน parameter บางตัวใน structure DMUS_OBJECTDESC

Object ชนิด IDirectMusicContainer8 เป็น object เก็บ object ต่างๆ ซึ่งสามารถ load ได้จาก file ที่ทำขึ้นจากโปรแกรม DirectMusic Producer ก็สามารถดึง object โดยใช้ method GetObject ได้เช่นเดียวกัน

Cache Management

การ Cache เป็นประโยชน์ในการจัดการ Memory ซึ่งหากใช้ Method GetObject จะมีการ Cache เกิดขึ้น โดยจะช่วยประหยัดหน่วยความจำไปได้มาก

เช่น หากมีการใช้ Segment ที่ใช้ Style เดียวกัน เมื่อ Segment ตัวที่สองมีการใช้ Style เดียวกัน ก็จะใช้ Style ตัวเดิมเคยถูก load ขึ้นมาแล้ว

หลักการทำงานของ Cache จะเป็นการเก็บ Pointer ที่ชี้ไปยัง object ซึ่งจะคอยบริหารการใช้ object หากไม่มีการใช้ object นั้นก็จะถูกลบออกไป แต่หากมีการอ้างอิงอยู่ก็จะเก็บไว้

การ Cache นั้นสามารถ Disable หรือ Enable ได้โดยใช้ method DisableCache และ EnableCache

การเล่นเสียง

เกี่ยวกับ Performance

ในการเล่นเสียงเมื่อมี Performance เพื่อทำการเล่นเสียงก่อน

Performance สามารถมีได้หลายตัวในหนึ่ง Application โดยอิสระต่อกัน โดยตัวอย่างการสร้างมีดังต่อไปนี้

```
IDirectMusicPerformance8* pPerf;

if (FAILED(CoCreateInstance(
      CLSID_DirectMusicPerformance,
      NULL,
      CLSCTX_INPROC,
      IID_IDirectMusicPerformance8,
      (void**)&pPerf
   )))
{
   pPerf = NULL;
}
```

เมื่อมีการสร้างแล้ว จะต้องมีการ Initialize เสมอ โดยใช้ Method Init

Parameter ของ Performance สามารถใช้ SetParam ในการตั้งค่าและ SetGlobalParam ในการตั้งค่า Performance ทุกตัว และในขณะเดียวกันก็สามารถดูค่าของ Parameter ได้จาก GetParam

เกี่ยวกับ Audiopath

Performance สามารถมี Audiopath ได้หนึ่งหรือหลายๆตัว โดย Audiopath นั้นมี object มากมายที่เกี่ยวข้องกับตัว Audiopath เอง

การสร้าง Audiopath นั้นสามารถสร้างโดย object Performance IDirectMusicPerformance8) โดย CreateStandardAudioPath เพื่อสร้าง Audiopath แบบมาตฐานหรือ InitAudio เพื่อสร้าง Audiopath ที่เป็น default หรือ CreateAudioPath จาก file ของ DirectMusic Producer

Audiopath ที่เป็นตัว default จะเป็นตัวที่เล่น segment เมื่อใช้คำสั่ง PlaySegment หรือ PlaySegmentEx ซึ่งสามารถ Set Audiopath อื่นๆให้เป็นตัว default ได้โดย Method SetAudioPath และดึง Default Audiopath ได้โดย GetDefaultAudioPath

Standard Audiopath คือ Auidopath ที่เป็นค่ามาตฐานซึ่งมีอยู่ด้วยกัน 4 ชนิด ซึ่งมีคุณสมบัติแตกต่างกันไปดังนี้

Standard Audiopath	Description	Comment
3D-Dry	Mono-3D buffer	มี IDirectSound3Dbuffer8 เป็น object ภายใน
Reverb	Stereo buffer with music reverberation effect	มี IDirectSoundFXWavesReverb8
Stereo	Stereo buffer with no effects	
Mono	Mono buffer with no effects	

ตารางที่ 2.1 แสดงชนิดของ Audiopath

การเล่นเสียงจาก Segment

การเล่นเสียงทำได้โดยใช้ method PlaySegment หรือ PlaySegmentEx โดยระบุ Audiopath ที่ต้องการและ Segment ที่เล่น ซึ่งการเล่นนั้นจะต้องใช้ Audiopath ในการเล่น โดย PlaySegment นั้นจะใช้ Audiopath ที่เป็น default ในขณะที่ PlaySegmentEx นั้นสามารถระบุ Audiopath ได้ ซึ่งมีสองวิธี

- โดยใช้ Audiopath ที่สร้างไว้แล้ว ซึ่งอาจจะใช้ Audiopath ที่เป็น default หรือระบุ audiopath ที่ต้องการ
- โดยใช้ Temopary audiopath โดยใช้ parameter ที่ชื่อ DMUS_SEGF_USE_AUDIOPATH โดย segment จะสร้างเมื่อมีการเล่น และจะทำลายเมื่อหยุดเล่น

2.1.5 ขั้นตอนคร่าวๆในการทดลองเล่นเสียง

เราสามารถสรุปการใช้งานจากหัวข้อบนได้อย่างคร่าวๆดังนี้

1. Initialize COM
2. สร้างและ Initialize Performance
3. สร้าง Loader
4. นำ Loader ไป load segment
5. Download ข้อมูล ลงไปยังตัวสังเคราะห์เสียง
6. ใช้ Performance สั่งเล่นเสียง

2.1.6 เสียง 3 มิติ

พิกัดของพื้นที่ 3 มิติ

ในพิกัด 3 มิติของส่วน DirectAudio จะใช้พิกัดแบบ Cartesian ซึ่งมีสามแกนคือ x,y และ z โดยแกน x แทนระยะซ้ายขวา แกน y แทนระยะบนล่าง แกน z แทนระยะใกล้ไกล

ใน DirectAudio จะใช้ Structure ที่ชื่อ D3DVECTOR ซึ่งใช้ในการเก็บพิกัดดังกล่าว ซึ่งจะแยกได้เป็น (x, y, z) แบบทั่วไป

ในหน่วยของการวัดระยะจะใช้มาตราเมตรเป็นค่า default ซึ่งผู้ใช้อาจจะใช้หน่วยอื่นก็ได้ เช่น หน่วยฟุต จะต้องทำการตั้งค่า Distance factor ให้เป็น 0.3048 เพื่อแปลงหน่วยทั้งหมดให้เป็นฟุต (หรือหน่วยเมตรมีค่า Distance factor เป็น 1) โดยใช้ method ของ IDirectSound3Dlistener ที่ชื่อ SetDistanceFactor และใช้ GetDistanceFactor ในการดูค่า Distance factor ปัจจุบันที่ใช้อยู่

ความเข้าใจในตำแหน่งของเสียง

ในโลกของความเป็นจริง ผลกระทบที่เกิดขึ้นจากตำแหน่งของเสียงจะได้รับอิทธิพลจากสิ่งต่างๆ มากมาย ปัจจัยต่างๆดังเช่น

- ความดังของเสียง หากแหล่งกำเนิดเสียงไกลจากผู้ฟังเท่าใด ก็จะได้ยินค่อยเท่านั้น ซึ่งจะมีระยะหนึ่งที่หากไกลจากระยะนี้จะไม่ได้ยินเสียงจากแหล่งนั้น เราเรียกว่า roll off
- ความรู้สึกแตกต่างของหู หากแหล่งกำเนิดใกล้หูข้างขวามากกว่า ย่อมได้ยินเสียงที่หูด้านขวาดังกว่าหูด้านซ้าย
- ความรู้สึกแตกต่างในด้านเวลา การแผ่ของคลื่นเสียงจะมีเวลาในการเดินทาง หากเสียงมาจากทางด้านขวา ย่อมจะมาถึงหูด้านขวาก่อนหูข้างซ้าย
- เสียงที่อู้อี้ รูปร่างและการหันของศีรษะผู้ฟังจะมีผลกระทบต่อเสียง เช่น ถ้าเสียงมาจากทางขวา เสียงที่ด้านซ้ายจะเกิดการอู้อี้โดยมวลของศีรษะและการหันของผู้ฟัง

ใน DirectAudio จะมีการช่วยให้การคำนวณผลกระทบเกี่ยวกับปัจจัยของเสียงให้กับผู้ใช้ โดยจะเรียกว่า head-related transfer function (HRTF)

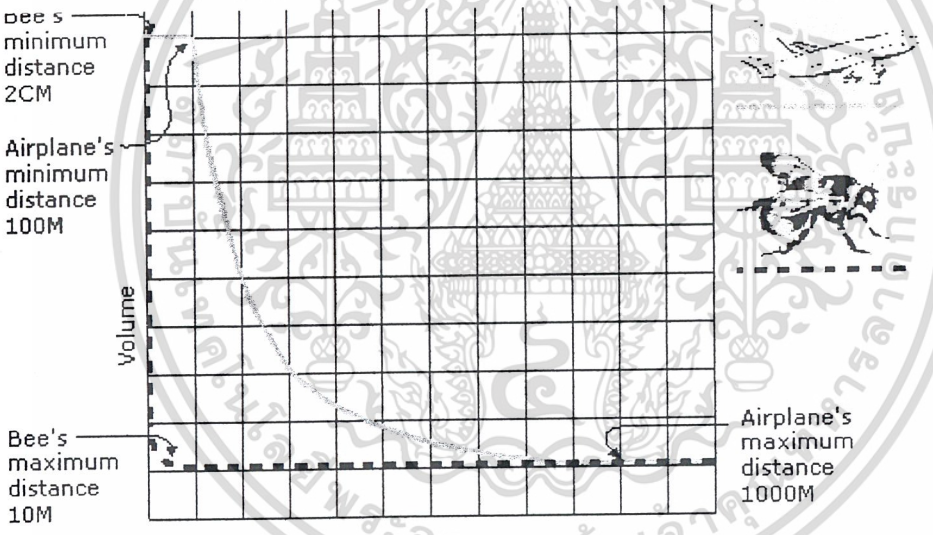
ระยะทางต่ำสุดและสูงสุดของเสียง

เมื่อผู้ฟังได้เข้าใกล้แหล่งกำเนิดเสียง เสียงที่ได้ยินจะดังขึ้นเรื่อยๆ เสียงจะดังเป็นสองเท่าเมื่อระยะทางห่างจากเดิมครึ่งหนึ่ง และเมื่อผ่านจุดๆหนึ่ง เสียงก็จะไม่ดังเพิ่มอีกต่อไป เราเรียกระยะนั้นว่า ระยะทางต่ำสุดของเสียง (minimum distance)

ระยะทางต่ำสุดเป็นประโยชน์อย่างมากในการชดเชยเสียงที่แตกต่างกันของแหล่งกำเนิดเสียง เช่น อาจจะมีระยะต่ำสุดเป็น 100 เมตรหากต้นเสียงเป็นเครื่องยนต์ Jet หรือระยะต่ำสุดอาจจะเป็นแค่ 2 เซนติเมตรหากเสียงเป็นเพียงแมลง

ระยะทางสูงสุดของเสียงคือระยะที่เสียงจะไม่เบาไปกว่านี้อีกแล้ว ค่า default ที่ตั้งมาคือ 1 ล้าน

ในตัวอย่าง เสียงเครื่องยนต์ หากตั้งระยะสูงสุดไว้ที่ 1000 เมตร ระยะต่ำสุด 100 เมตร ก็จะมีระดับของความดังเป็นดังรูป เสียงผึ้งก็เช่นเดียวกัน ซึ่งใน object IDirectSound3Dlistener มี method ในการตั้งค่าคือ SetMinDistance, SetMaxDistance และดูค่าโดย GetMinDistance และ GetMaxDistance



รูปที่ 2.2 ตัวอย่างของระยะทางต่ำสุดและระยะทางสูงสุด

Processing Mode

Mode ในการประมวลผลของ DirectAudio ใน 3 มิติมีอยู่สามแบบคือ Normal ซึ่งเป็นค่า Default , Mode Head-relative ซึ่งจะคำนวณผลกระทบของจาก Listener มากยิ่งขึ้น ซึ่งก็จะทำให้ประมวลผลมากขึ้นด้วยและ Disable mode ซึ่งจะเป็นการปิดการประมวลผล 3 มิติของเสียง

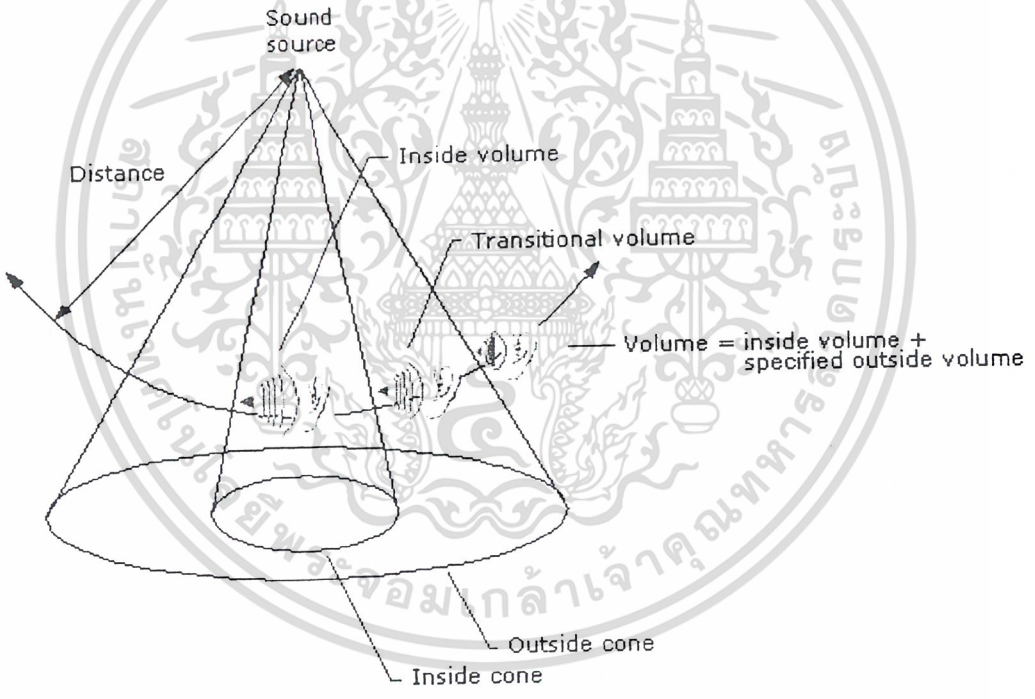
ตำแหน่งและความเร็ว

ตำแหน่ง(Position)จะแทนด้วยค่าของ Vector ซึ่งจะมีความสัมพันธ์ต่อผู้ฟังและสภาพแวดล้อม โดยเป็นอิสระต่อ Processing Mode และความเร็ว(Velocity)ก็จะแทนด้วย Vector เช่นกัน แต่มีหน่วยเป็นหน่วยต่อวินาที(หาก Distance factor เป็น 1 จะเท่ากับ เมตรต่อวินาที) ซึ่งใช้ในการคำนวณเกี่ยวกับผลกระทบต่อเสียงของการเคลื่อนที่

กรวยของเสียง(Sound Cone)

ความดังของเสียงจะดังในทิศทางที่แหล่งกำเนิดหันไป แบบจำลองที่บอกถึงความดังที่เกี่ยวข้องกับกันหันจะเป็นรูปแบบกรวย เรียกว่า Sound Cone ซึ่งจะแบ่งได้เป็นในและนอกกรวย

จากรูปเราจะอธิบายเสียงที่อยู่ในกรวยจะมีเสียงที่ดังสุด และระหว่างข้างนอกและในของกรวยจะแปรผันตามมุม ส่วนความดังที่อยู่นอกกรวยจะมีค่าตามที่ตั้งไว้ โดยค่า default จะเป็น 0 หรือไม่มีเสียง



รูปที่ 2.3 แสดงรูปของกรวยเสียง

ในค่า default นั้นจะไม่มีการตังการหันและกรวย ดังนั้นแหล่งกำเนิดเสียงจะเป็นลักษณะลูกบอลหรือไม่มีการหัน เสียงจะกระจายไปทุกทิศ ดังนั้นจึงไม่เป็นลักษณะที่เป็นกรวย

แต่ใน Application ที่ต้องการความสมจริงๆในด้านเสียง เช่น เสียงตะโกนจากที่ๆหนึ่งจะต้องตั้งค่าของการหันและกรวยให้เหมาะสม จะทำให้เสียงที่ได้ 3 มิติสมจริงมากยิ่งขึ้น

การหันของผู้ฟัง

ผู้ฟังจะแทนได้เป็นความสัมพันธ์ของสอง Vector ซึ่งใช้จุดกำเนิดที่เดียวกันซึ่งแทนเป็นจุดศูนย์กลางของศีรษะผู้ฟัง โดย vector หนึ่งจะพุ่งออกไปทางข้างหน้า อีก vector จะพุ่งออกไปทางด้านบน ซึ่งทั้ง
สองจะแทนการหันของศีรษะได้ดังรูป

รูปที่ 2.4 แสดงการหันของผู้ฟัง

Doppler Factor และ Rolloff Factor

ใน DirectAudio นั้นจะมีการคำนวณ Effect เกี่ยวกับความสัมพันธ์ในสภาพแวดล้อม 3 มิติ ซึ่ง Doppler และ Rolloff นั้นเป็น Effect ดังกล่าว ซึ่งสามารถตั้งค่าให้ factor ของ effect ทั้งสองได้ ซึ่งบาง Application ต้องการ เช่น ค่า factor ที่ต้องตั้งเพราะผู้ฟังอยู่ในน้ำ ซึ่งต่างจากอยู่บนพื้นดิน

DirectSound3Dbuffer และ DirectSound3DListener

แหล่งกำเนิดเสียงใน 3 มิติจะแสดงตัวเป็น object IDirectSound3Dbuffer8 โดยสามารถสร้างใน Standard Audiopath แบบ 3D-Dry เท่านั้น ซึ่งจะมี method ต่างๆที่จำเป็นซึ่งเกี่ยวข้องกับทฤษฎีข้างต้น โดยการสร้าง IDirectSound3Dbuffer8 นั้นเป็นการ interface กับ Secondary Buffer เดิมซึ่งมี flag DSBCAPS_CTRL3D ซึ่งสามารถดึง object จาก Segment โดยใช้ method GetObjectInPath หรือ Audiopath โดยใช้ method GetObject

```
HRESULT                 hr;
IDirectMusicAudioPath8*  g_p3DAudioPath;
IDirectSound3DBuffer8*   g_pDS3DBuffer;
```

```
if (FAILED(hr = pPerformance->CreateStandardAudioPath(
      DMUS_APATH_DYNAMIC_3D, 64, TRUE, &g_p3DAudioPath)))
   return hr;

if (FAILED(hr = g_p3DAudioPath->GetObjectInPath(
      DMUS_PCHANNEL_ALL, DMUS_PATH_BUFFER, 0,
      GUID_NULL, 0, IID_IDirectSound3DBuffer8,
      (LPVOID*) &g_pDS3DBuffer)))
   return hr;
```

หรือจาก Secondary Buffer โดยใช้ QueryInterface

```
LPDIRECTSOUND3DBUFFER8 lpDs3dBuffer;
HRESULT hr = lpDsbSecondary->QueryInterface(IID_IDirectSound3DBuffer8, (LPVOID
*)&lpDs3dBuffer);
```

ส่วน object IDirectSound3Dlistener นั้นเป็น object แทนตัวผู้ฟัง ซึ่งมี method ต่างๆที่ใช้โดยเกี่ยวข้องกับทฤษฎีเช่นกัน โดยแนะนำว่าควรจะสร้างเพียง object เดียวใน Application จะทำให้ไม่สับสน

IDrectSound3Dlistener จะ interface กับ Primary Buffer สามารถใช้ method QueryInterface ในการดึง object ออกมา

```
DSBUFFERDESC        dsbd;
LPDIRECTSOUNDBUFFER   lpdsbPrimary;
LPDIRECTSOUNDLISTENER8 lp3DListener;

ZeroMemory(&dsbd, sizeof(DSBUFFERDESC));
dsbd.dwSize = sizeof(DSBUFFERDESC);
dsbd.dwFlags = DSBCAPS_CTRL3D | DSBCAPS_PRIMARYBUFFER;
if SUCCEEDED(lpds->CreateSoundBuffer(&dsbd, &lpdsbPrimary, NULL))
{
   // Get listener interface.
   if FAILED(lpdsbPrimary->QueryInterface(
```

```
        IID_IDirectSound3DListener8,
        (LPVOID *)&lp3DListener))
    {
    lpdsbPrimary->Release();
    }
}
```

หรือใช้ method GetObjectInPath ของ Segment หรือ AudioPath ก็ได้

```
IDirectSound3DListener8* g_pDSListener;

HRESULT hr = g_p3DAudioPath->GetObjectInPath(
    DMUS_PCHANNEL_ALL, DMUS_PATH_PRIMARY_BUFFER, 0,
    GUID_NULL, 0, IID_IDirectSound3DListener8,
    (LPVOID*) &g_pDSListener));
```

2.1.7 การใช้ Effect

Effect ใน DirectAudio สามารถใช้ในการแต่งเสียงได้ ซึ่งมีหลากหลายชนิดและสามารถผสมได้ด้วย โดย DirectAudio จะมี Interface และ Structure ที่ใช้กับ Effect โดยมี Standard Effect ดังนี้

- Chorus

เป็นการเพิ่มเสียงเสมือนมีหลายๆเสียงแปร่งออกมาพร้อมกัน เช่น การร้องประสานเสียง โดยแทนได้ object เป็น IDirectSoundFXChorus8 และมี parameter เป็น DSFXChorus

- Compressor

เป็นการทำให้การแกว่งของเสียงอยู่เหนือของความกว้างของคลื่นที่กำหนดหรือการบีบอัดเสียง โดยแทนได้ object เป็น IDirectSoundFXCompressors8 และมี parameter เป็น DSFXCompressor

- Distortion

เป็นการเพิ่ม Harmonics ให้ไปในทางเดียว เช่น การสร้างเสียง Distortion ของ Guitar โดยแทนได้ object เป็น IDirectSoundFXDistortion8 และมี parameter เป็น DSFXDistortion

- Echo

เป็นการทำเสียงให้เล่นซ้ำในช่วงเวลาที่กำหนด โดยแทนได้ object เป็น IDirectSoundFXEcho8 และมี parameter เป็น DSFXEcho

- Flange

เป็นการ Echo ที่มีการหน่วงเวลาระหว่างสัญญาณเดิมและเป็น Echo ที่มีช่วงเวลาสั้นๆและเวลาไม่คงที่ โดยแทนได้ object เป็น IDirectSoundFXFlanger8 และมี parameter เป็น DSFXFlanger

- Gargle

เป็น Effect ที่เป็นความลึกของคลื่น โดยแทนได้ object เป็น IDirectSoundFXGargle8 และมี parameter เป็น DSFXGargle

- Parametric Equalizer

เป็นการขยายสัญญาณในช่วงความถี่หนึ่ง โดยแทนได้ object เป็น IDirectSoundFXParamEq8 และมี parameter เป็น DSFXParamEq

- Wave Reverberation

เป็นการทำให้เสียงก้องขึ้น โดยแทนได้ object เป็น IDirectSoundFXWavesReverb8 และมี parameter เป็น DSFXWavesReverb

การตั้ง Effect ที่ Sound Buffer จะใช้ Method ที่ชื่อ SetFX ในการตั้งค่า Effect โดยต้องใส่ Parameter ให้ตาม Effect ที่ต้องการตั้ง เช่น

```
DSEFFECTDESC dsEffect;
dsEffect.dwSize = sizeof(DSEFFECTDESC);
dsEffect.dwFlags = 0;
dsEffect.guidDSFXClass = GUID_DSFX_STANDARD_ECHO; dsEffect.dwReserved1 = 0;
dsEffect.dwReserved2 = 0;

DWORD dwResults;
// Set the effect
if (FAILED(hr = g_pDSBuffer->SetFX(1, &dsEffect, &dwResults)))
return hr;
// You can check the value of dwResults here to see if and how
// the effect was allocated.
// Activate the path.
g_p3DAudioPath->Activate(TRUE);
```

สังเกตว่าการตั้ง Effect จะทำให้ Audiopath มีการ Activate เป็น False ซึ่งเสียงที่กำลังอยู่จะมีผลทำให้หยุดโดยปริยาย ดังนั้นจะต้องทำการ Activate ใหม่

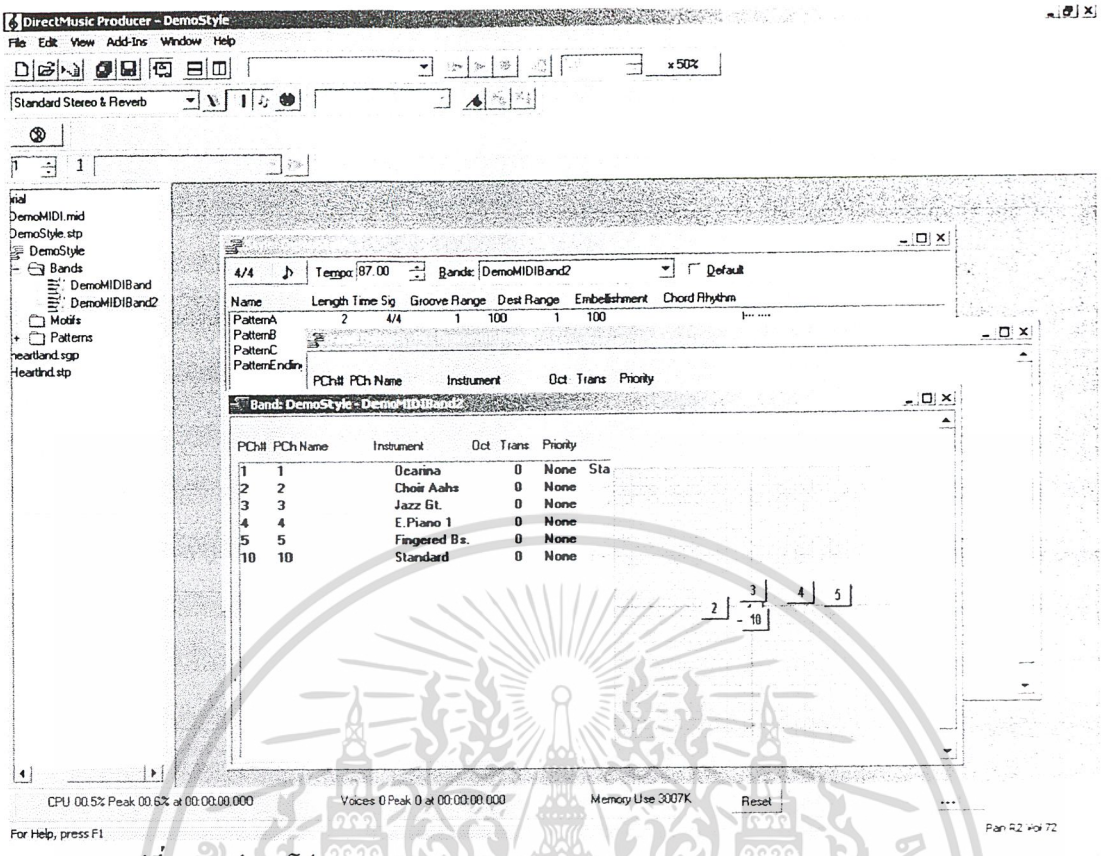
การดูหรือตั้งค่า Parameter ที่ทำการใส่ Effect ไปแล้วนั้นต้องอาศัย object ที่เป็น parameter ของ Effect นั้น ซึ่งดูได้จากด้านบน

```
HRESULT hr;
IDirectSoundFXGargle8* pEffectDMO;
DSFXGargle FXParams;
hr = g_p3DAudioPath->GetObjectInPath(DMUS_PCHANNEL_ALL,
DMUS_PATH_BUFFER_DMO, 0, GUID_All_Objects, 0, IID_IDirectSoundFXGargle8, (LPVOID*)
&pEffectDMO);
hr = pEffectDMO->GetAllParameters(&FXParams); FXParams.dwRateHz =
DSFXGARGLE_RATEHZ_MIN;
hr = pEffectDMO->SetAllParameters(&FXParams);
```

จากตัวอย่างเป็นการตั้งค่า Effect ชนิด Gargle โดยดึง object IDirect SoundFXGargle8 จาก Audiopath แล้วตั้งค่ากลับไป

2.1.8 DirectMusic Producer

DirectMusic Producer เป็นโปรแกรมไม่ได้แนบกับทาง DirectX8 SDK ซึ่งเป็นโปรแกรม Utility ไว้สำหรับการทำ file format ที่ DirectAudio Support ต่างๆ นอกเหนือจาก MIDI และ Wave file format ซึ่งมีความสามารถมากมาย โดยสามารถ download ได้ www.microsoft.com



รูปที่ 2.5 รูปของโปรแกรม Direct Music Producer

2.2 ทฤษฎีและหลักการของ Direct Input

DirectInput เป็นส่วนหนึ่งของ DirectX ซึ่งใช้สำหรับติดต่อกับกับ Hardware ส่วนการ Input ได้โดยผ่านทาง DirectX API เช่น keyboard, mouse, joystick และอุปกรณ์ input หลายอย่าง โดยธรรมดาแล้ว การรับการ Input จะผ่านทาง Win32 API โดยใช้การรับผ่านทาง message จาก Windows ซึ่งอุปกรณ์ Input บางอย่างไม่สามารถดึงค่าจะจาก message ได้ เช่น joystick ดังนั้นจึงมีความจำเป็นที่ต้องใช้ DirectInput ในการเขียนโปรแกรมเกมแทบทุกชนิดก็ว่าได้ ซึ่งข้อดีของ DirectInput API คือจะติดต่อเข้าถึง Hardware ได้ตรงกว่าการใช้ Windows message

2.2.1 สิ่งที่ควรรู้ในส่วน DirectInput

- DirectInput object เป็น object หลักของ DirectInput interface
- Device เป็น hardware ที่ใช้ในการรับ input เช่น keyboard หรือ mouse
- DirectInput Device object เป็น object ที่ใช้แทนตัวของ device
- Device object คือ object ส่วนประกอบของ device เช่น ปุ่ม ทิศทาง ฯลฯ
- Cooperative Levels เป็นสิ่งที่พิจารณาว่า device จะทำการ share กับ application อื่นอย่างไร

2.2.2 ขั้นตอนอย่างคร่าวๆในการนำ DirectInput มาใช้กับการเขียนโปรแกรมเกม

ในการใช้ DirectX SDK ส่วน DirectInput โดยทั่วไปแล้วการนำข้อมูลที่ผู้เล่น Input เข้ามาประมวลผลจะมีขั้นตอนดังนี้

1. สร้าง DirectInput object ซึ่งเป็น object หลักของ DirectInput Interface ซึ่งใช้ในการสร้างและแจกแจง DirectInput device objects
2. Enumerate device การที่เราจะสร้าง DirectInput Device object ขึ้นมาได้นั้น เราจะต้องค้นหาว่าในระบบของเรามี device อะไรบ้าง จึงจำเป็นต้องค้นหาก่อน หากว่า device อยู่ในระบบ ก็จะสามารถสร้าง object ขึ้นมาแทนตัว device ได้
3. สร้าง DirectInput Device object ที่เราต้องการ โดยหากเราสามารถค้นหา device ว่ามีอยู่ในระบบ ก็จะสร้าง object ที่เราต้องการหรือทั้งหมดได้ ขึ้นมาติดต่อกับ device ตัวนั้น เช่น เราสามารถสร้าง object ของ keyboard ได้เมื่อเรา Enumerate จนพบ device ที่เป็น keyboard
4. Setup device โดยเราต้องทำการ setup device ที่เราติดต่อด้วย ผ่านทาง DirectInput Device object เช่น การตั้งค่า cooperative level, ตั้งค่ารูปแบบข้อมูลของ device ฯลฯ
5. เข้าถึง device ซึ่งเมื่อเราเริ่มต้องการที่จะดึงค่า input จาก device
6. ดึงข้อมูลจาก device โดยผ่านทาง DirectInput Device object
7. กระทำการต่างๆ เมื่อเรารู้ค่าที่ input เข้า เช่น หากมีการกดปุ่มขวา(เรารู้จากข้อมูลที่ดึงมา) ก็ให้ทำการเดินหน้า
8. ปิดส่วน input เมื่อเราไม่ต้องการติดต่อกับ device ส่วน input แล้ว

2.2.3 ขั้นตอนต่างๆที่เกิดขึ้นในการใช้งาน DirectInput

สร้าง DirectInput

เริ่มแรกเราสามารถดึง Interface ของ IDirectInput8 ซึ่งเป็น DirectInput object โดยใช้ Function ที่ชื่อ DirectInput8Create ในการสร้าง โดย IDirectInput8 ควรจะมีเพียง object เดียวและ release เมื่อก่อนจบโปรแกรม

การแจกแจง DirectInput Device

การแจกแจง (Enumeration) เป็นการดูว่ามี Device อะไรบ้างที่ทำการติดต่อกับระบบ และให้ข้อมูลเกี่ยวกับ Device นั่น

เช่น การ Enumerate keyboard ที่มีอยู่ โดยใช้ Function EnumDevices หรือ EnumDevicesBySemantics ของ IDirectInput8 object โดยระบุประเภท Device ที่ต้องการซึ่งเป็น keyboard ของระบบ

```
GUID KeyboardGUID = GUID_SysKeyboard;

lpdi->EnumDevices(DIDEVTYPE_KEYBOARD, DIEnumDevicesCallback, &KeyboardGUID,
DIEDFL_ATTACHEDONLY);
```

หากดูจากสามารถอธิบายได้ดังนี้ โดย lpdi คือ IDirectInput8 object ใช้ function EnumDevices โดย parameter แรกแสดงถึงประเภทของ Device(ในที่นี้คือ keyboard)

parameter ต่อมาคือ Address ของ Callback function เมื่อมีการ Enumerate โดยเมื่อจะถูกเรียกเมื่อมีการเรียก

parameter ที่สามเป็นค่า 32 bits ใดๆที่จะถูกส่งเข้าไปใน Callback function เพื่อใช้ประโยชน์ต่างๆ ในที่นี้ เราจะส่งค่า pointer ของ GUID เข้าไปเพื่อการสร้าง DirectInput device object

สุดท้ายแสดงถึงการ Enumerate เฉพาะ keyboard ที่ต่ออยู่เท่านั้น

นี่คือตัวอย่างของ Callback function ซึ่งสามารถสร้างเองได้ตามความต้องการ

```
BOOL  hasEnhanced;

BOOL CALLBACK DIEnumKbdCallback(LPCDIDEVICEINSTANCE lpddi,
                    LPVOID pvRef)
{
  *(GUID*) pvRef = lpddi->guidInstance;
  if (GET_DIDEVICE_SUBTYPE(lpddi->dwDevType) ==
                DIDEVTYPEKEYBOARD_PCENH)
  {
    hasEnhanced = TRUE;
    return DIENUM_STOP;
  }
  return DIENUM_CONTINUE;
} // End of callback
```

หากดู Source Code อธิบายได้ว่า หากค่า dwDevType ซึ่งเป็นส่วนย่อยของ GUID ที่ pass เข้ามาตรงกับ DIDEVTYPEKEYBOARD_PCENH ก็จะหยุดการ Enumerate (โดยการ return DIENUM_STOP เพื่อบอกว่าหยุด) หากไม่ใช่ก็จะ Enumerate ต่อไป

การสร้าง DirectInput Device

สามารถสร้างได้โดยใช้ Function CreateDevice ของ IDirectInput8 เช่น การสร้าง keyboard โดยดูจาก code ตัวอย่างนี้

```
LPDIRECTINPUTDEVICE8  lpdiKeyboard;
lpdi->CreateDevice(GUID_SysKeyboard, &lpdiKeyboard, NULL);
```

โดย Parameter แรกเป็น constant GUID ซึ่งเป็นของ System keyboard โดยจะ return object ไปยัง lpdiKeyboard หากทำสำเร็จ

โดยทั่วไป เราสามารถใช้การ CreateDevice เมื่อมีการ Enumerate โดยใช้ GUID ที่ pass เข้าไปยัง callback function เมื่อมีการ Enumerate พบที่ต้องการก็จะทำการสร้าง Device ทันที

ความสามารถของ Device

ก่อนที่จะทำการอ่านค่า Input จากผู้เล่น เราควรจะรู้รายละเอียดของ Device นั้นๆให้ดีเสียก่อน เช่น มีกี่ปุ่ม ชนิดอะไร ฯลฯ

เราสามารถใช้ GetCapabilities ของ IDirectDevice8 โดยค่าที่ return กลับมาจะเป็น structure แบบ DIDEVCAPS เช่น

```
DIDEVCAPS  DIMouseCaps;
HRESULT    hr;
BOOLEAN    WheelAvailable;

DIMouseCaps.dwSize = sizeof(DIDEVCAPS);
hr = lpdiMouse->GetCapabilities(&DIMouseCaps);
WheelAvailable = ((DIMouseCaps.dwFlags & DIDC_ATTACHED)
    && (DIMouseCaps.dwAxes > 2));
```

หากดูจาก Source code จะเป็นการตรวจสอบว่า mouse มี wheel หรือไม่ โดย mouse ทั่วไปจะมี 2 แกน คือ X และ Y หาก dwAxes มีมากกว่า 2 แสดงว่า mouse มี wheel

Cooperative Levels

เป็นระดับการจำกัดการ Share device ให้กับ Application โดยใช้ Function SetCooperativeLevels ของ IDirectDevice8 ในการเลือกระดับ

```
/* hwnd is the top-level window handle. */
lpdiDevice->SetCooperativeLevel(hwnd, DISCL_NONEXCLUSIVE | DISCL_FOREGROUND)
```

Cooperative จะแบ่งออกเป็นสองส่วนใหญ่ๆ ซึ่ง Device จะสามารถเลือกได้ว่าจะเลือกใช้ mode foreground หรือ background, exclusive หรือ non-exclusive

Foreground หมายความว่าจะอ่านค่า Input ที่ได้จาก Device เมื่อ Application ที่ใช้อยู่ Active เท่านั้น โดยจะหยุดการเข้าถึง Device อย่างอัตโนมัติ ส่วน Background นั่นสามารถรับค่า Device ได้ตลอดแม้ว่า Application ของตัวเองจะไม่ Active ซึ่งในการเขียนโปรแกรมเกมนั่น ควรจะใช้ Mode foreground แต่ว่าหากอยู่ในขบวนการ debug แล้ว จะต้องใช้ background เท่านั้นเพราะว่า Application จะมองโปรแกรม Compiler (VC) เป็นอีกโปรแกรม จึงไม่สามารถรับค่า Input ได้

Exclusive หมายความว่า Application อื่นจะไม่สามารถดึงค่าจาก Device ได้หากใช้ mode นี้ โดย Non-exclusive จะเป็นลักษณะตรงข้ามกัน ในกรณีที่ Device สนับสนุนระบบ Force Feedback จะต้องใช้ Exclusive mode เท่านั้น

รูปแบบของข้อมูลที่ดึงจาก Device

การรับข้อมูลจาก Device จะต้องมีรูปแบบข้อมูลที่รับเข้ามาเป็น Structure เพราะแต่ละ Device จะมีคุณสมบัติแตกต่างกันไป ไม่ว่าจะเป็น Keyboard, mouse หรือ joystick โดยใช้ Function SetDataFormat ของ IDirectInputDevice8 เช่น

```
lpdiMouse->SetDataFormat(&c_dfDIMouse);
```

โดย c_dfDIMouse เป็นตัวแปรชนิด Global ซึ่งเป็น Default format data ของ Mouse

ส่วน Properties ของ device จะประกอบไปด้วยขอบเขตของแกน,จำนวนแกน,ขนาดของ buffer ที่ใช้อยู่ ฯลฯ ซึ่งสามารถตั้งค่าและดูค่าของ Device นั่นๆได้ เพื่อปรับเปลี่ยนให้เหมาะสมกับ Application ของเราเอง เช่น ตัวเลขของขอบเขตสูงสุด ซึ่งผู้พัฒนาอาจจะต้องการกำหนดเอง โดยใช้ Function ของ IDirectInputDevice8 ซึ่งมีทั้ง GetProperty และ SetProperty

การเข้าถึง Device

การที่จะดึงข้อมูล Input ของ Device ได้นั้น เราจะต้องเข้าถึงตัว Device ก่อน (Acquiring Devices) โดยใช้คำสั่ง Acquire และเมื่อไม่ต้องการดึงข้อมูลก็ต้องทำการยกเลิกการเข้าถึง(Unacquiring Devices) โดยใช้คำสั่ง Unacquire

โดยหากใช้ Cooperative level แบบ Foreground เมื่อ Application ไม่ได้ Active ก็จะทำการยกเลิกการเข้าถึงอย่างอัตโนมัติ ดังนั้นอาจจะเกิดการ Device-Lost ได้ดังนั้นเมื่อมีการ Active ของ Application เมื่อใด ซึ่งอาจจะใช้ Handle WM_ACTIVE ในการเข้าถึง Device อีกครั้ง

ใน Property ของ Device บางอย่างจะต้องทำการเข้าถึงตัว Device ก่อนถึงจะทำการตั้งค่าหรือรับค่า เช่น ค่า Gain ของ Device ที่สนับสนุน Force-Feedback

การดึงข้อมูลจาก Device เพื่อนำมาใช้งาน

เมื่อ Acquiring Device แล้วก็สามารถดึงข้อมูลจาก Device เพื่อนำมาใช้งานได้ โดยทั่วไปแล้วเราจะนำข้อมูลที่ได้มาตรวจสอบดูว่าเกิดการกระทำอะไรบ้างที่สอดคล้องกับโปรแกรมของเรา แล้วจึงทำตามคำสั่งนั้นๆ

ยกตัวอย่างเช่น โปรแกรมเกมเป็นเกมต่อสู้ เมื่อเราดึงข้อมูลที่ได้จาก Device มาแล้วพบว่ามีการกดปุ่ม Enter ที่ Device Keyboard เราก็ให้โปรแกรมทำท่าการต่อย เป็นต้น

2.2.4 สิ่งที่ควรรู้เกี่ยวกับ DirectInput Device Data

Buffered Data กับ Immediate Data

ในการดึงข้อมูลเพื่อนำไปใช้งานนั้น มีรูปแบบในการดึงข้อมูลอยู่สองแบบ โดย Buffered Data นั้นจะบันทึกเหตุการณ์ที่เกิดขึ้นไว้ใน Buffer จนกว่าจะเต็มหรือถูกอ่านนำไปใช้ ส่วน Immediate Data คือการดึงข้อมูลตอนที่เราต้องการตอนนั้น เดี๋ยวนั้น

จากการทำงานเราจะเห็นได้ว่า Buffered Data นั้นจะเหมาะสมกับข้อมูลที่ต่อเนื่อง เช่น การเคลื่อนที่ของ Mouse ส่วน Immediate Data นั้นเหมาะกับข้อมูลที่มีลักษณะเป็นแบบ State เช่น การกดปุ่มของ Joystick ซึ่งหลักการทำงานทั้งสองแบบเราไม่จำเป็นต้องเลือก สามารถใช้ได้ทั้งสองแบบใน Device เดียวกัน เช่น อาจจะใช้ Buffered Data ในการเคลื่อนที่ของ Joystick โดยใช้ Immediate Data ในการกดปุ่ม

Buffered Data สามารถใช้คำสั่ง GetDeviceData ของ IDirectInputDevice8 ในการอ่านค่าใน Buffer โดยเหตุการณ์ต่างๆ เช่น การเลื่อนของ mouse หรือการกดปุ่มจะบันทึกไว้ใน Buffer โดยจะถูกล้างเมื่อมีการอ่านออกไป หาก Buffer เต็มก็จะไม่บันทึกเหตุการณ์ใหม่เข้าไป(สามารถตั้งค่า Buffer ได้โดยคำสั่ง SetProperty) โดย Data ที่อยู่ใน Buffer จะอยู่ในรูปแบบของ DIDEVICEOBJECTDATA

Immediate Data สามารถใช้คำสั่ง GetDeviceState ของ IDirectInputDevice8 ในการอ่านค่า หากมีเหตุการณ์เกิดขึ้นในระหว่างการอ่านค่า เช่น การกดปุ่ม ก็จะอ่าน Data ได้ดังนั้น หมายความว่า หากเกิดเหตุการณ์ที่ไม่ได้อยู่ในระหว่างการ GetDeviceState เหตุการณ์จะเสมือนไม่ได้เกิดขึ้น

ในความเป็นจริง Joystick บางชนิดไม่สามารถใช้ Buffered Data ได้ เพราะไม่มีHardware Interrupt

Polling และ Event Notification

สองอย่างเป็นวิธีการที่จะดึง Input Data ออกมา

การ Poll เป็นวิธีการที่พื้นฐานในการดึงข้อมูล โดยในโปรแกรมเกมจะมี Loop ของเกมอยู่ โดยจะมี Function ในการดึงข้อมูลซึ่งอยู่ใน Loop นั้น โดยวิธีนี้จะเป็นวิธีที่ใช้บ่อยมากที่สุดและสามารถใช้ได้ทั้งกับ Buffered Data และ Immediate Data (Buffered ใช้ในการอ่านข้อมูลที่อยู่ใน Buffer แต่ไม่ได้ใช้ในการบันทึก)

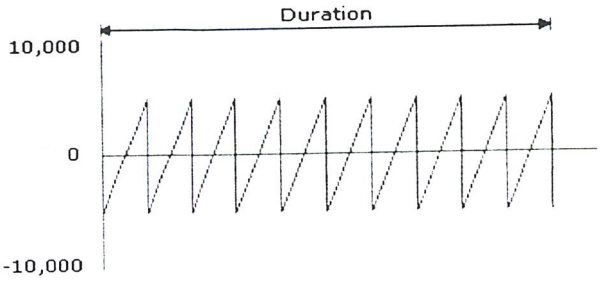
คำสั่ง Poll ของ IDirecInputDevice8 เป็นการเตรียม Device ในการอ่านข้อมูล แต่มิใช่การอ่านข้อมูล ซึ่ง Joystick บางชนิดจะต้องใช้คำสั่งนี้ก่อนที่จะอ่านข้อมูล

การ Poll ควรใช้เมื่อต้องการรับ Input อย่างต่อเนื่อง เช่น ในขณะเล่นเกม แต่เมื่อไม่มีการใช้ค่า Input Data ก็ไม่ควรจะมีการ Poll เกิดขึ้น เช่น ตอนการเล่าเรื่องของเกม

Event Notification เป็นการตั้งค่าให้ IDirectInputDevice8 เกิดการ thread-synchronization กับ Hardware โดยใช้ CreateEvent โดยใช้คำสั่ง SetEventNotification ในการตั้งค่า ดังนั้นการใช้ Event จึงไม่มีการทำงานในส่วนการ Input หากไม่เกิด Event ขึ้นมา จึงทำให้โปรแกรมทำงานได้เร็วกว่า เพราะไม่ผ่าน Function Input แต่หากมีการรับ Input ต่อเนื่องโปรแกรมอาจจะทำงานได้ช้ากว่าการ Poll ดังนั้นจึงเหมาะสมกับการรับ Input ที่มีไม่มาก เช่น การกดปุ่ม esc เพื่อข้ามฉากเล่าเรื่อง

2.2.5 ทฤษฎีของ Force Feedback

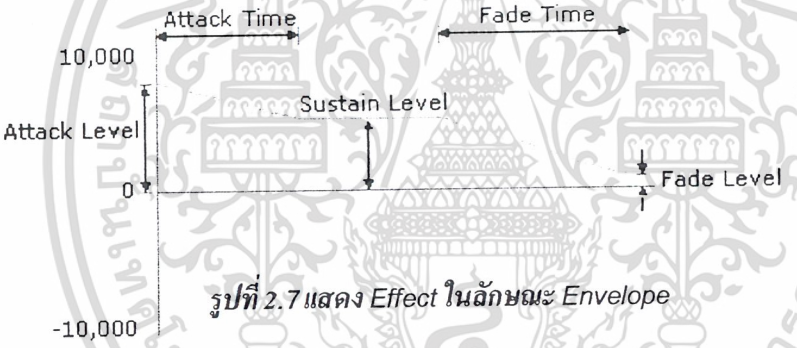
Force Feedback เป็นการสร้างการตอบสนองจากโปรแกรมไปยัง Device ที่เป็น Input โดยใช้มอเตอร์ในการสร้างแรง เช่น เมื่อมีการกดปุ่มยิงปืน โปรแกรมก็จะส่งให้มีการสร้างแรงเสมือนการยิงปืนที่ Joystick ของผู้เล่น ซึ่ง Device จะต้องสนับสนุน Force Feedback ด้วย เราจะเรียกแรงที่เกิดขึ้นว่า Effect



รูปที่ 2.6 แสดงลักษณะของ Force-Feedback Effect

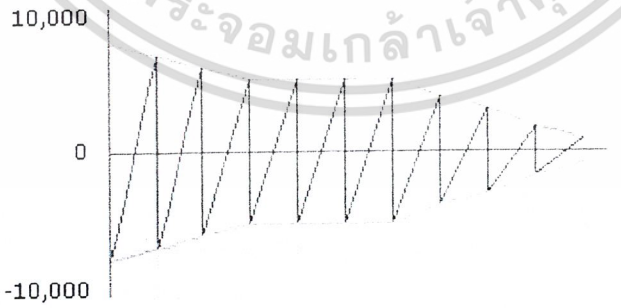
จากรูป จะเห็นได้ว่าลักษณะของ Effect จะเป็นรูปแบบคลื่น โดยค่า Magnitude ซึ่งมีค่าระหว่าง -1000 ถึง 10000 โดยค่าเป็นลักษณะ Linear ซึ่งค่าบวกลบแสดงถึงทิศทาง

Duration เป็นเวลาที่เกิด Effect ขึ้นมีหน่วยเป็น microsecond โดยเวลาจะดำเนินไปเรื่อยๆ หากดูจากรูป เมื่อเราดูจากรูปจะสรุปได้ว่า Effect นี้มีการส่งแรงจากทิศทางลบไปบวกทุกๆช่วงหนึ่งของ Duration เราเรียก Effect ในรูปได้ว่าเป็นรูปแบบ Periodic



รูปที่ 2.7 แสดง Effect ในลักษณะ Envelope

Effect สามารถอธิบายเป็นลักษณะแบบ Envelope ได้ โดยมีค่า Attack และ fade เป็นตัวกำหนด



รูปที่ 2.8 แสดงตัวอย่างของ Effect ในรูป Periodic

หากดูจากรูปแสดงให้เห็นว่า Attack level จะเริ่มที่ค่า 8000 และ fade ลงมาเรื่อยๆจนถึง 1000

เราสามารถในรูปแบบของ Envelope ให้อยู่ในรูป Periodic ได้ดังรูป

ทิศทางของ Effect (Effect Direction) โดยทิศทางสามารถอธิบายในลักษณะของแกน ค่าลบบวกของแกน X แสดงถึงซ้ายขวา แกน Y แสดงถึงสูงต่ำ แกน Z แสดงถึงความใกล้ไกล ซึ่ง Effect ก็เช่นเดียวกัน

ทิศทางของ Effect นั้นช่วยให้ผู้เล่นสามารถรู้สึกถึงทิศทางของ Effect ที่เกิดขึ้น เช่น การสั่นซ้ายทีขวาที ทำให้รู้สึกสมจริงมากกว่าการไม่มีทิศทาง ซึ่ง Device นอกจากจะต้องสนับสนุน Force Feedback แล้ว จะต้องสนับสนุนแกนของทิศทางด้วย เช่น หาก Effect มีการเกิดแบบ 3 แกน แต่ Device สนับสนุนแค่แกน X อย่างเดียว ผู้เล่นก็ไม่รู้สึกถึงแกนที่เหลือ หมายความว่า รู้สึกเพียงซ้ายขวา แต่ไม่รู้สึกถึงสูงต่ำและใกล้ไกล

2.2.6 สิ่งที่ควรรู้เมื่อใช้งาน Force Feedback

การแจกแจง(Enumerate) Effect

การ Enumerate Effect จะคล้ายกับการ Enumerate Device โดย Effect จาก Enumerate จาก Device ว่า Effect ที่เสนอไปนั้น มีอยู่ใน Device หรือไม่ ซึ่งใช้ในการสร้าง Effect ต่อไป

IDrectInputDevice8 มี method ที่ชื่อว่า EnumEffects ในการ Enumerate โดยลักษณะจะคล้ายกับ Enumerate ตัว Device ซึ่งใช้ Callback function เช่นเดียวกัน เช่น

```
HRESULT hr = g_lpdid->EnumEffects(&DIEnumEffectsCallback, g_lpdid, DIEFT_ALL);
```

หมายถึงการ Enumerate Effect ที่มี GUID แบบ DIEFT_ALL (Effect ทุกชนิด) โดยเรียก Callback function ที่ชื่อ DIEnumEffectsCallback โดย pass ค่าของ g_lpdid ซึ่งเป็น object device เข้าไปใน Callback function

```
BOOL CALLBACK DIEnumEffectsCallback(LPCDIEFFECTINFO pdei, LPVOID pvRef)
{
        HRESULT hr; LPDIRECTINPUTDEVICE8 lpdid = (LPDIRECTINPUTDEVICE8)pvRef;
        if (DIEFT_GETTYPE(pdei->dwEffType) == DIEFT_CONSTANTFORCE)
        hr = lpdid->CreateEffect(pdei->guid, &diEffect, &lpdiEffect, NULL); . . . }
        return DIENUM_CONTINUE; }
```

หากดูจาก callback function จะเห็นได้ว่าหาก Enumerate พบ Effect แบบ DIEFT_CONSTANTFORCE ก็ให้สร้าง Effect นั้นขึ้นมา

การ Enumerate Effect สามารถทำได้จาก file ที่บันทึกอยู่ในรูป *.ffe ซึ่งใช้โปรแกรม Force Editor ในการสร้างและบันทึก (โปรแกรม Force Editor แถมมาให้กับ DirectX8 SDK) โดยใช้ function EnumEffectsInFiles แทน โดยใช้ Callback function เช่นเดียวกัน

```
LPDIRECTINPUTEFFECT pEff[3];

g_lpdid->EnumEffectsInFile("FEdit1.ffe", EnumEffectsInFileProc, NULL,
DIFEF_MODIFYIFNEEDED);

BOOL CALLBACK EnumEffectsInFileProc(LPCDIFILEEFFECT lpdife,
      LPVOID pvRef)

{
   HRESULT     hr;
   static int  i;


   hr = g_lpdid->CreateEffect(lpdife->GuidEffect,
                        lpdife->lpDiEffect,
                        &pEff[i],
                        NULL);
   if (FAILED(hr))
   {
      // Error handling
   }
   if (++i > 2) return DIENUM_STOP;
   else return DIENUM_CONTINUE;
}
```

ใน file .ffe นั้นมิได้เก็บเพียง Effect เดียวต่อหนึ่ง file การ Enumerate ย่อมทำได้หลายๆครั้ง เหมือนการ Enumerate จาก Device

ข้อมูลเกี่ยวกับ Effect ที่ Support ต่อ Device

เราสามารถดู Effect ที่สนับสนุนต่อ Device นั้น มีข้อมูลอะไรบ้าง โดยใช้ GUID ของ Effect เป็นตัวอ้างอิง โดยใช้ method GetEffectInfo ของ IDirectInputDevice8 object ซึ่งจะ return ค่าเป็นรูปแบบของ DIEFFECTINFO

```
DIEFFECTINFO diEffectInfo;
diEffectInfo.dwSize = sizeof(DIEFFECTINFO);
g_lpdid->GetEffectInfo(&diEffectInfo, EffectGuid);
if (diEffectInfo.dwDynamicParams & DIEP_DIRECTION)
{
  // Can reset parameter dynamically
}
```

การสร้าง Effect

การสร้าง Effect สามารถสร้างได้โดย CreateEffect ของ IDirectInputDevice8 เช่น

```
g_lpdid->CreateEffect(GUID_ConstantForce, &diEffect, &lpdiEffect, NULL);
```

โดย Parameter แรกคือ GUID ซึ่งเป็น Constant GUID โดยระบุว่าเป็น Effect ชนิด Constant Force และ Parameter ที่สองคือตัวแปรชนิด DIEFFECT ซึ่งเป็น parameter ของ Effect นั่นๆ

Parameter ต่อมาคือ Address ของ object ที่เป็น IDirectInputEffect8 ซึ่งจะส่งค่าไปหากสามารถสร้าง Effect ได้สำเร็จ

การสร้าง Effect นั้นสัมพันธ์อย่างยิ่งกับการ Enumerate โดยสามารถ Enumerate ได้ ย่อมจะสามารถสร้าง Effect ได้เช่นกัน ในการสร้าง Effect จะต้องการข้อมูลในการสร้าง คือ GUID และค่า parameter ต่างๆ ซึ่งสามารถรับค่าได้จากการ Callback function ของการ Enumerate ไม่ว่าจาก file หรือจากการโปรแกรมมิ่ง

การเล่น Effect

โดยใช้ method Start ของ IDirectInputEffect8 โดยสามารถสั่งให้เล่นจำนวนครั้งที่ต้องการได้

การที่จะเล่น Effect ได้นั้น ขึ้นอยู่กับตัว Device เป็นสำคัญ หากสถานะ Device ไม่พร้อม ย่อมไม่สามารถเล่น Effect ได้

การ Download และ Unload Effect

ก่อนที่จะเล่น Effect จะต้องมีการ Download Effect ลงไปยัง Device ก่อน การที่ใช้ method Start นั้น Effect จะถูก Download ลงไปอย่างอัตโนมัติ (หากไม่ได้ยกเลิกการ Download แบบอัตโนมัติ)

เมื่อ Device เกิดการ Unacquire นั้น เช่น การสลับ Windows ไปยัง Application อื่นๆ โดยตั้งค่า Cooperative Level ไว้ที่ Exclusive foreground นั้น Effect จะถูก Unload อย่างอัตโนมัติ ดังนั้น Effect จะต้องการ Download ลงไปใหม่อีกครั้ง

Device สามารถเกิดการเต็มของ Effect ได้ ดังนั้น อาจจะต้องใช้การ Unload เพื่อจัดการให้ Effect ที่ยังไม่ได้เล่นออกไปก่อน

Parameter ที่เกี่ยวข้องกับ Effect

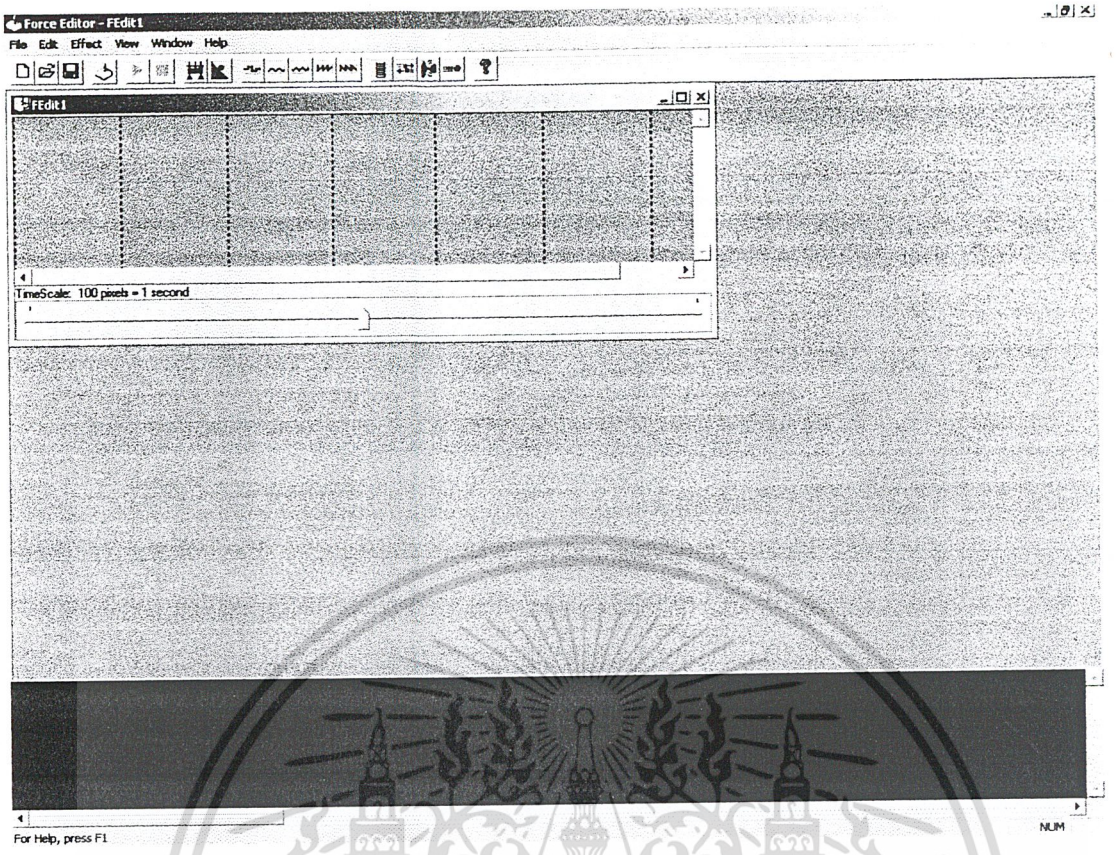
เราสามารถ Set ค่าต่างๆของ Effect ได้แม้ว่าจะทำการสร้างไปแล้ว โดยใช้ method SetParameter และสามารถดูค่า Parameter ต่างๆของ Effect ที่มีอยู่ได้จาก GetParameter ของ IDirectInputEffect8

ค่า Gain ของ Effect คือค่าความแรงของการเล่น Effect ซึ่งสามารถปรับระดับได้ โดยมีค่า 0 ถึง 10000 หมายความว่า หากตั้งค่า Gain ไว้ที่ 7500 หมายความว่า ตั้งค่าแรงไว้ที่ 75 percent ของความแรงปกติ การตั้งค่า Gain สามารถตั้งได้โดยใช้ method SetProperty

Device ที่สนับสนุน Force-Feedback สามารถตั้งค่าสถานะต่างๆได้ ไม่ว่าจะเป็นการ Mute, Play, Stop, Pause, Continue ได้จากการใช้ Method SendForceFeedbackCommand ของ IDirectInputDevice8 ซึ่งก็สามารถดูค่าสถานะต่างๆได้จาก GetForceFeedbackState

2.2.7 โปรแกรม Force Editor

เป็นโปรแกรม Utility ที่แถมมาให้กับ DirectX8 SDK โดยสามารถสร้างและแก้ไข Force-Feedback Effect ได้อย่างง่าย โดยสามารถบันทึกเป็น file .ffe เพื่อใช้ในการเขียนโปรแกรมได้ ซึ่งมีความสามารถเหมือนกับการสร้าง Effect จากการโปรแกรมมิ่ง แต่สะดวกกว่ามาก ดังนั้นการสร้าง Effect เพื่อใช้กับเกมนั้น จึงนิยมใช้วิธีนี้มาก



รูปที่ 2.9 โปรแกรม Force Editor

2.3 ทฤษฎีและหลักการของ Direct Show

Microsoft DirectShow คือส่วนที่มาจัดการกับ streaming media บน Windows platform.ช่วยในการเล่นและ capture multimedia streams. แถมยังรองรับ format ได้หลากหลาย ซึ่งประกอบด้วย Advanced Streaming Format (ASF), Motion Picture Experts Group (MPEG), Audio-Video Interleaved (AVI), MPEG Audio Layer-3 (MP3), and WAV files. สนับสนุนการ capture โดยใช้ Windows Driver Model (WDM) devices หรือ Video for Windows devices. DirectShow จะทำการค้นหาHardware ที่เกี่ยวข้องโดยอัตโนมัติ

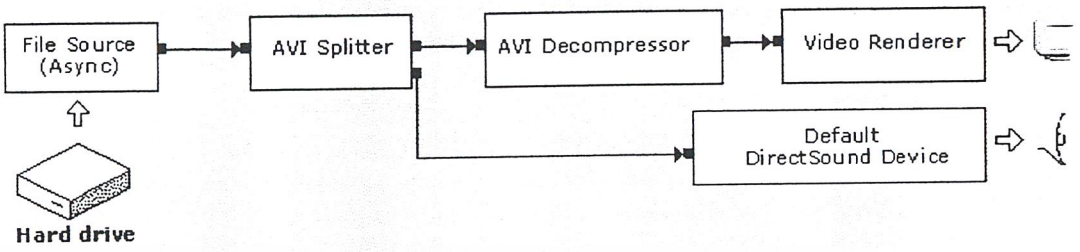
2.3.1 สิ่งที่ควรรู้ในส่วนของ DirectShow

DirectShow Application Programming

สิ่งที่เราควรจะรู้ก่อนที่เราจะมาทำการProgrammingในส่วนของDirectShow นั่นก็คือ Filter Graphs

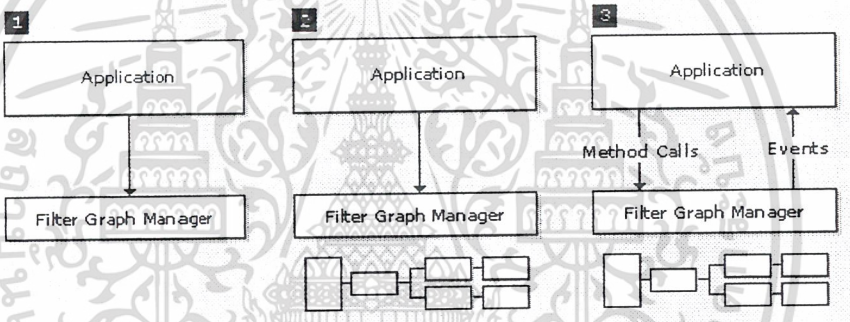
ตัว DirectShow จะสร้าง block component มาเรียกว่า Filter เพื่อที่มาจัดการกับ Multimedia Stream โดยจะทำการ อ่านไฟล์ และทำการ Decodeเฉพาะ Stream Format แล้วส่ง ข้อมูลไปยัง Graphic & Sound Card

Set ของ Filter จะเรียกว่า Filter Graphs โดยแต่ละ Filter จะทำงานในหน้าที่แตกต่างกัน ดัง Diagram ข้างล่าง



รูปที่ 2.10 แสดงภาพของ Filter

ทั้งนี้ ตัว DirectShow เองยังทำการจัดการกับ Filter เองโดยเพียงเราเขียน Application ครอบลงไปบนตัว Filter Graphs Manager แค่นั้นเอง มันจะทำการจัดการกับ Flow ต่างๆเอง ดังรูป



รูปที่ 2.11 แสดงรูปแบบการเขียนโปรแกรมกับ Direct Show

การสร้างสภาพแวดล้อมในการใช้งาน Direct Show

ในการสร้าง DirectShow Application นั้น จะต้องประกอบด้วย

Header File

Header File	Required For
Dshow.h	All DirectShow applications

ตารางที่ 2.2 แสดง Includes File ที่ต้องการ

Library File

Library File	Description
Strmiids.lib	เพื่อใช้ class identifiers (CLSIDs) and interface identifiers (IIDs). DirectShow applications ทั้งหมดต้องการ.

Quartz.lib เพื่อใช้ *AMGetErrorText* function. ถ้าไม่ใช้ ก็ไม่จำเป็นต้องมีไว้.

Strmbase.lib ถ้าจะต้องการใช้ในส่วนของDirectShow *base classes*.

ตารางที่ 2.3 แสดง Library Files ที่ต้องการ

การเล่นไฟล์

ในการเล่นไฟล์โดยใช้ DirectShow ในแต่ละครั้ง จะต้องทำตามขั้นตอน 4 ขั้นตอนคือ

1. ต้องสร้าง Filter Graphs Manager มาจัดการ
2. Filter Graphs Manager จะทำการสร้าง Filter Graphs
3. ให้ตัว Filter Graphs Manager เป็นตัวเล่นไฟล์
4. รอให้มันจัดการเล่นไฟล์จบ

โดยใช้ COM Interface ของตัว DirectShow ในการ สร้างและควบคุม Filter Graphs

IGraphBuilder : เป็นตัวสร้าง Filter Graph

IMediaControl : เป็นเครื่องมือจัดการเกี่ยวกับ Media Streamming ใน Filter Graph

IMediaEvent : เป็นตัวจัดการเกี่ยวกับ event ต่างๆใน Filter Graph

ในการสร้าง Filter Graph Manager ต้องใช้ interface ทั้งหมดที่กล่าวมาในการสร้าง ดัง code

```
IGraphBuilder *pGraph;
CoInitialize(NULL);
CoCreateInstance(CLSID_FilterGraph, NULL, CLSCTX_INPROC_SERVER,
        IID_IGraphBuilder, (void **)&pGraph);
```

CoInitialize ใช้ในการเริ่มต้นค่าให้กับ COM Library ส่วน CoCreateInstance ใช้ในการสร้าง Filter Graph Manager โดยจะ return เป็น pointer ออกไป โดยส่วนนี้ ยังต้องมี interface อีก 2 ตัวข้างต้น

```
IMediaControl *pMediaControl;
IMediaEvent   *pEvent;
pGraph->QueryInterface(IID_IMediaControl, (void **)&pMediaControl);
pGraph->QueryInterface(IID_IMediaEvent, (void **)&pEvent);
```

ส่วนสำคัญของโปรแกรมก็คือ ส่วนของการ เล่นไฟล์ ดู Code

```
pGraph->RenderFile(L"C:\\Hello_World.avi", NULL);
pMediaControl->Run();
pEvent->WaitForCompletion(INFINITE, &evCode);
```

IGraphBuilder ::RenderFile – เป็น methode ที่ใช้สร้าง Filter Graph เพื่อเล่นไฟล์

ImediaControl::Run – เป็น methode ที่ควบคุมให้ Filter อยู่ใน Mode ของการเล่น โดยให้ Application อ้างถึง Methode นี้ ในการเล่นไฟล์

ImediaEvent:: WaitForCompletion – เป็น methode ที่ทำกับ block file จนกว่าจะทำการเล่นเสร็จเรียบร้อยแล้ว

ในการใช้งานจริง เราจะ return ค่าของ RenderFile ออกมาเพื่อยืนยันการสร้าง Filter Graph ถ้าเกิด error จะ return ออกเป็น error code VFW_E_NOT_FOUND

ในการทำลาย Filter Graph Manager ทำโดยการ Release มันทิ้งไป

```
pMediaControl->Release();
pEvent->Release();
pGraph->Release();
CoUninitialize();
```

การใช้ IVideoWindow Interface

ในการ render video ไฟล์ Filter Graph จะทำการสร้าง Filter สำหรับการ Render เป็นส่วนที่ต้องสร้างด้วย Application ที่เราทำขึ้น

โดยใช้ IVideoWindow ซึ่งมี methode ในการตั้งค่า และปรับปรุงคุณสมบัติต่างๆ ของ window

1.เซ็ทให้มันเปิดอยู่ใน window เดิม แทน parent window

```
IVideoWindow    *pVidWin = NULL;
pGraph->QueryInterface(IID_IVideoWindow, (void **)&pVidWin);
pVidWin->put_Owner((OAHWND)g_hwnd);
```

methode จะเปลี่ยนไปตามtype ของ OAHWND

2. ทำการเปิด window ใหม่ โดย WS_CHILD จะกำหนดให้มันเป็น window ลูกอยู่แล้ว ส่วน WS_CLIPSIBLINGS จะป้องกันการวาดลงไปใน window ที่เปิดใหม่อยู่

```
pVidWin->put_WindowStyle(WS_CHILD | WS_CLIPSIBLINGS);
```

3.เซ็ทให้มันเปิดใน window เดิม แต่มีการเซ็ทค่าแกน X,Y โดยวาง video window ไว้อยู่บน parent window

```
RECT grc;
GetClientRect(g_hwnd, &grc);
pVidWin->SetWindowPosition(0, 0, grc.right, grc.bottom);
```

2.3.2 Event ใน Direct Show

การตอบสนองต่อ Event ที่ตัว Application ส่งมาให้ และส่ง message Event คืนให้กับตัว Application

หลักการทำงานของ Event Notification

ขณะที่ DirectShow Application ทำการเล่นไฟล์ ณ เวลาใดๆ อยู่นั้น สถานะของ Filter อาจจะเปลี่ยนแปลงได้ เช่น การหยุดการเล่น เป็นต้น เพื่อนที่จะส่งสัญญาณให้ Filter Graph Manager ทราบถึงการเปลี่ยนแปลง Filter จะทำการส่ง Event Notification ซึ่งประกอบด้วย Event Code พร้อมด้วยชนิดของ Event และยังมี Parameter อีก2ตัว ซึ่งจะขึ้นอยู่กับ Event Code (ดูจากภาคผนวค)

Filter Graph Manager สามารถจัดการกับบาง Event ได้โดยไม่ต้องยุ่งกับ Application เลย ส่วน Event อื่นๆ ก็จะจัดอยู่ในคิว คล้ายๆกับ Window Message แต่ Filter Graph Manager จะทำการส่ง Window Message ก็ต่อเมื่อ เกิด Event ใหม่ๆเท่านั้น Application จึงจัดการกับ Event ใหม่ๆได้จาก Message loop ของ Window

ยกตัวอย่าง Event Code : EC_COMPLETE , EC_USERABORT

EC_COMPLETE : จะส่ง Event นี้ก็ต่อเมื่อ Renderer Filter ทำงานในการอ่านไฟล์ Stream เสร็จเรียบร้อยแล้ว ก็จะส่งมายัง Filter Graph Manager โดยแต่ละ Filter Graph อาจจะต้องรอหลายๆ

Filter ทำงานเสร็จเพราะว่ามีหลาย Stream ในหนึ่ง Graph เมื่อ Application ได้รับ Event Code นี้จาก Filter Graph Manager ก็จะทราบว่าทุกๆ Stream ถูกอ่านเรียบร้อยแล้ว

EC_USERABORT : ในกรณีที่ User เกิด Interrupt การเล่นไฟล์ Video Renderer ก็จะทำการปิด Video Window

การใช้ Event Notification

การจัดการกับ Event โดย Window Message loop หลัก จะต้อง Define Message ที่จะส่ง โดย Application สามารถใช้จำนวน Message ได้ในช่วงจาก WM_APP ถึง 0xBFFF เป็น Message ส่วนตัว :

```
#define WM_GRAPHNOTIFY  WM_APP + 1
```

จากนั้นก้อทำการสร้าง Filter Graph Manager ที่จะทำการส่ง Message Event ไปยัง Main Window :

```
pEvent->SetNotifyWindow((OAHWND)g_hwnd, WM_GRAPHNOTIFY, 0);
```

IMediaEventEx::SetNotifyWindow เป็นการกำหนดให้ตัวเองเป็นตัวรับ Message โดยต้องเรียก Methode นี้หลังการสร้าง Filter Graph และ Window

ใน function WindowProc จะต้องเพิ่ม case Statement ให้กับ WM_GRAPHNOTIFY ก่อน

```
case WM_GRAPHNOTIFY:
HandleEvent();
break;
```

WM_GRAPHNOTIFY เป็น Windows message แยกจาก Event Notification Queue ของ DirectShow:

1. filter จะทำการส่ง event notification ไปยัง filter graph manager.
2. ถ้า filter graph manager ไม่สามารถจัดการกับ event นั้นได้ ก็จะทำการใส่ลงไปใน event queue.
3. filter graph manager จะส่ง WM_GRAPHNOTIFY message ไปยัง application window.
4. application ตอบสนองกับ message จากภายใน window's message loop.
5. application จะไปเอา event notification จาก queue มาทำการจัดการต่อไป.

IMediaEvent::GetEvent คือ Function ที่ทำการนำเอา event notification จาก queue :

```
long evCode, param1, param2;
HRESULT hr;
while (hr = pEvent->GetEvent(&evCode, &param1, &param2, 0),  SUCCEEDED(hr))
{
hr = pEvent->FreeEventParams(evCode, param1, param2);
if ((EC_COMPLETE == evCode) || (EC_USERABORT == evCode))
{
CleanUp();
break;
}
}
```

GetEvent คือ method ที่เอา event code กับอีก 2 event parameters มาใช้. Parameter สุดท้าย ของ GetEvent จะกำหนดระยะเวลาของการรอ event. เพราะว่า application จะเรียก method มาใน การตอบสนอง WM_GRAPHNOTIFY message, event ที่พร้อมอยู่ใน queued.0 คือค่าของ time-out

2.3.3 DirectShow Solution

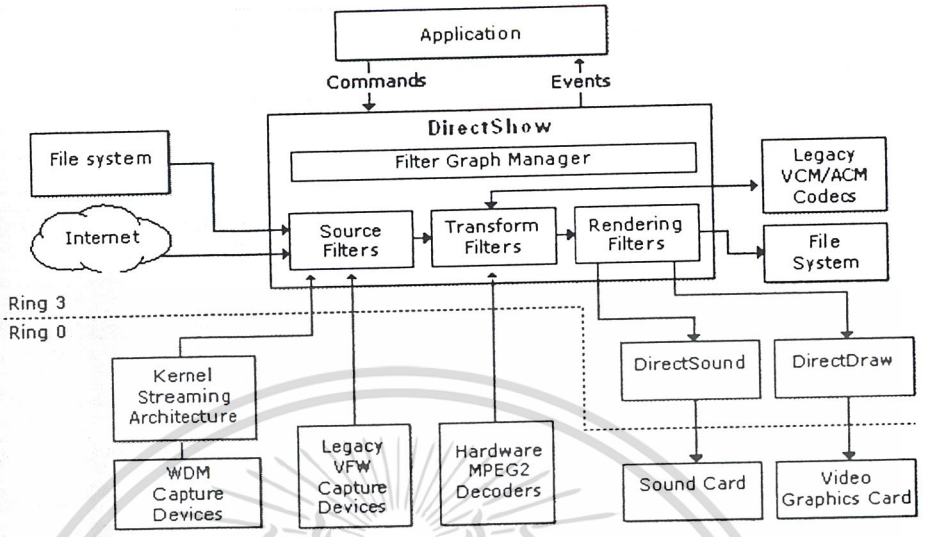
จุดประสงค์หลักของ DirectShow ในการออกแบบคือการสร้าง Multimedia Application ที่เป็นเอกเทศ แยกออกจากการที่มี data transport ที่ซับซ้อน ความแตกต่างของ Hardware และส่วนที่มันเกี่ยวข้องกันอยู่

DirectShow เองใช้ส่วนของ DirectDraw และ DirectSound ในการ Render ข้อมูลอย่างมีประสิทธิภาพ

เพื่อที่จะจัดการกับ source , formats, and hardware devices ที่แตกต่างกัน, DirectShow จะใช้โครงสร้างสำหรับจัดการ component ของระบบนั่นก็คือ *filters* มันสามารถ mixed และ matched เพื่อที่จะสนับสนุนสำหรับสถานการณ์ที่แตกต่างกันไป

DirectShow ประกอบด้วย filters ที่สนับสนุน cutting-edge multimedia capture and tuning devices บน พื้นฐานของ Windows Driver Model (WDM) เสมือนว่า filters นั้นสนับสนุน Video for Windows (VfW) capture cards และ codecs ที่เก่ากว่าถูกเขียนเพื่อ Audio Compression Manager (ACM) และ Video Compression Manager (VCM) interfaces

ตาม diagram จะแสดงความสัมพันธ์ระหว่าง Application กับ DirectShow Component บาง Hardware และ Software ที่มัน support อยู่



รูปที่ 2.12 แสดง Architecture ของ Direct Show

2.3.4 Filters

DirectShow จะแบ่งส่วนสำหรับกระบวนการทำข้อมูล multimedia ในแบบ discrete step. Application สามารถ Mix และ Match Filter ให้อยู่ในชนิดของ operation ที่แตกต่างกันในต่างๆ format ใน class ที่แตกต่างกันของ Hardware และ Software Device

แม้ว่าในแต่ละ Filter จะมีความ Unique มากในตัวเองก็ตาม DirectShow จะจัดการผ่าน IBaseFilter Interface จะแยกออกเป็น 3 หัวข้อใหญ่ๆ:

Source Filters

คือข้อมูล Multimedia ดิบที่ต้องการ Process

Transform Filters

เป็น Filter ที่รับทั้งข้อมูลดิบและกระบวนการที่จะจัดการกับข้อมูลนั้นเพื่อที่จะทำให้เป็น frame หรือ sample (จะกล่าวถึงในภายหลัง) หรือ Compressor กับ Decompressor กับ Format Converter. AVI Splitter และ AVI decompressor เป็น Transform Filter

Renderer Filters

เป็นส่วนที่ข้อมูลถูกแปลงมาเพื่อพร้อมจะเล่นออกทางอุปกรณ์ภายนอก(monitor, speaker)

Pins

เป็นตัวจัดการในระดับ low-level detail ระหว่าง Filter โดยที่ pin จัดเป็น COM object และยังเป็นตัวกำหนด direction ของ input กับ output ด้วย

Media Samples

หลังจากที่ข้อมูลดิบ ถูกทำให้เป็น Graph โดยข้อมูลถูกกระจายเข้าไปในหน่วยความจำ จะถูกเรียกว่า media sample ซึ่งเป็น object ที่ถูกกำหนดชนิดของ media และ synchronization time

Allocators

เมื่อ 2 Filter มาเชื่อมต่อกัน pin ของมันก็ต้องยอมรับรายละเอียดว่า media sample object มันจะเชื่อมต่อกันระหว่าง filter อย่างไร การ connect มันจะกำหนดทั้งขนาด ตำแหน่ง จำนวนของ sample ที่ถูกใช้ ขนาดของ sample ขึ้นอยู่กับชนิด และ format ของ media ตัว buffer อาจจะอยู่ในหน่วยความจำหลัก หรือว่าอยู่บน Hardware Device การสร้างและการจัดการ sample จะถูกกระทำโดยตัว allocator

Clocks

Operation ต่างๆที่เกี่ยวกับ mutimedia การ synchronize กันของ sample สำคัญมาก ว่า video frame จะต้องอยู่ใน rate ที่เหมาะสม ดังนั้น filter graph จะต้องมี clock ที่แน่นอน มีผลต่อ timestamp, กระบวนการ render media sample โดยทั้งหมด ตัว filter graph manager จะเป็นตัวจัดการทั้งหมด

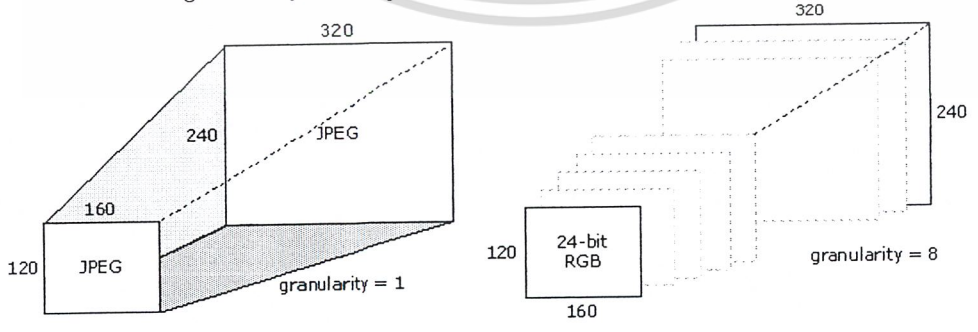
2.3.5 Video Capabilities

IAMStreamConfig::GetStramCaps เป็น method ที่แสดงความสามารถทางด้าน video ด้วยโครงสร้างของ AM_MEDIA_TYPE และ VIDEO_STREAM_CONFIG_CAPS . โดยสามารถเปิดได้ทุก format และความละเอียดที่ต้องการและยังสนับสนุนกับพวก pin ด้วย

สมมติว่าcapture card สนับสนุน JPG format ที่ความละเอียด 160 × 120 pixels และ 320 × 240 pixels โดยความแตกต่างในการ support ความละเอียดที่แตกต่างกัน จะถูกเรียกว่า granularity

สมมติว่า card support ความละเอียดที่ 640*480 มันจะอยู่ในช่วงของความละเอียด(ทั้ง 160 × 120 pixels และ320 × 240 pixels).

และสมมติว่ามัน support 24-bit color RGB format ที่ความละเอียดระหว่าง 160x120 และ 320x240 แต่ด้วย granularity = 8 ดังรูป.



รูปที่ 2.12 แสดงสัดส่วนการแสดงผลของ Video

2.3.6 Audio Capabilities

IAMStreamConfig::GetStramCaps จะใช้ AM_MEDIA_TYPE และ AUDIO_STREAMCONFIG_CAPS คล้ายๆกับวีดีโอ จะสามารถแจกแจงชนิดความสามารถของ audio ได้บน pin เช่น data rate หรือแม้แต่กระทั่งการสนับสนุนเสียงแบบ mono หรือว่า stereo

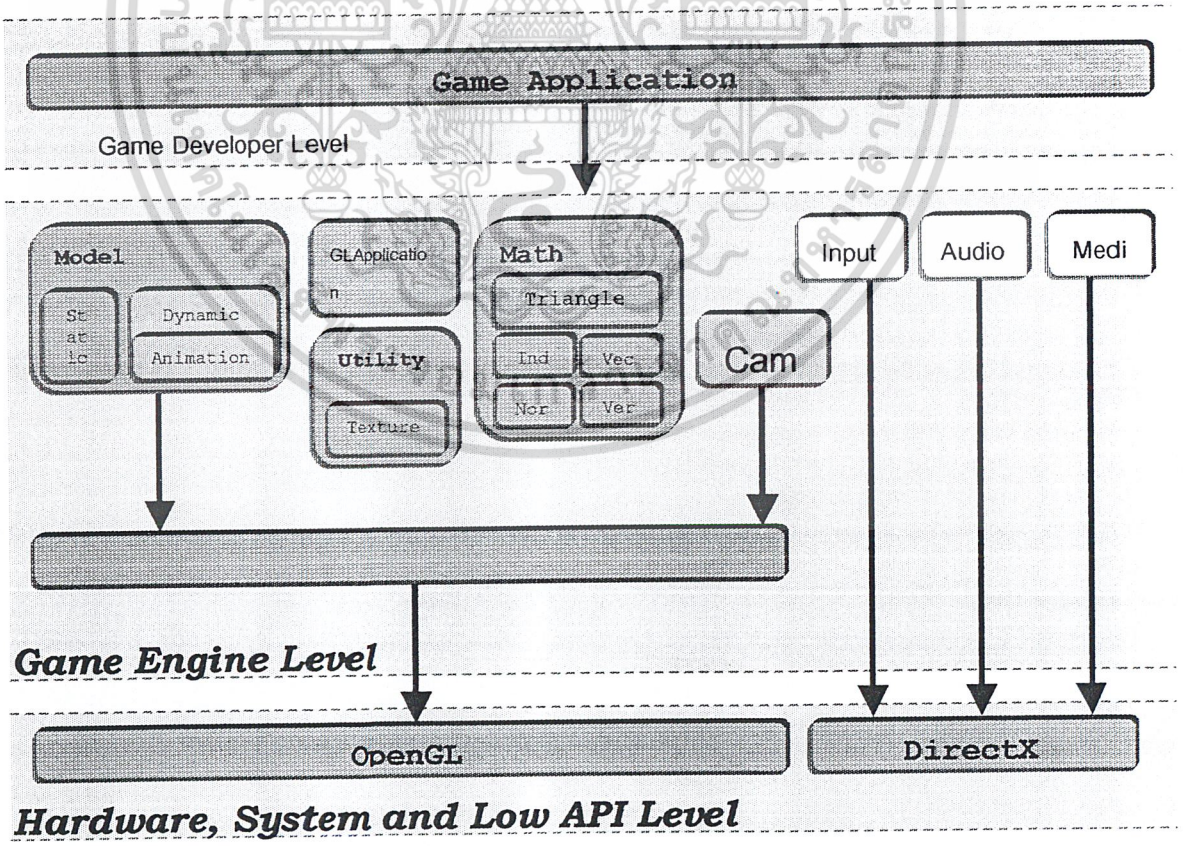
บทที่ 3
การออกแบบเอนจินต์สำหรับเกม 3 มิติ

3.1 โครงสร้างเกมเอนจินต์ (Engine Structure)

แนวคิดในการออกแบบเกมเอนจินต์ คือการช่วยให้ผู้พัฒนาเกมลดความซับซ้อนในการที่ต้องเขียนโปรแกรมเพื่อติดต่อกับ API ระดับล่างรวมถึงอุปกรณ์ฮาร์ดแวร์โดยตรง และเพิ่มฟังก์ชันที่จำเป็นต้องใช้ที่ยังไม่มีอยู่ใน API ของระบบ โครงสร้างของโปรแกรมที่พัฒนาโดยใช้เกมเอนจินต์แบ่งออกเป็น 3 ระดับ ได้แก่

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

โครงงานนี้จะพัฒนาในส่วนของ Game Developer Level และ Game Engine Level โดยมีโครงสร้างดังนี้



รูปที่ 3-1 แสดงโครงสร้างของเกมเอนจินต์

3.2 เอนจินต์คอมโพเนนต์ (Engine Component)

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

3.2.1 คาเมรา (Camera)

คาเมราเป็นส่วนที่จะใช้ในการจัดมุมมองภายในเกม ซึ่งในส่วนของคาเมรานี้มีเพียงคลาส Camera เท่านั้น ซึ่งประกอบไปด้วยฟังก์ชันหลักที่ใช้งานดังนี้

- SetAspect / GetAspect
  ใช้กำหนด / หาอัตราส่วนระหว่างสองแกนของ Resolution ของ Monitor
- SetEye / GetEye
  ใช้กำหนด / หาตำแหน่งปัจจุบันของกล้องเพื่อทำการหลบหลีกสิ่งกีดขวาง
- SetLookat / GetLookat
  ใช้กำหนด / หาตำแหน่งปัจจุบันของจุดมอง (Look at point) ของกล้อง
- SetFarPlane / GetFarPlane
  ใช้กำหนด / หาระนาบที่ไกลที่สุดของมุมกล้องที่จะนำมา Rendering
- SetNearPlane / GetNearPlane
  ใช้กำหนด / หาระนาบที่ใกล้ที่สุดของมุมกล้องที่จะนำมา Rendering
- Move
  ใช้เปลี่ยนพิกัดของ Eye ในทิศทางต่างๆ ทั้งเดินหน้า, ถอยหลัง, เคลื่อนขึ้นและลง
- Orbit
  ใช้เคลื่อนกล้องในทิศทางต่างๆ เมื่อเทียบกับจุดมอง โดยที่ยังมองอยู่ที่จุดเดิม
- Rotate
  ใช้กำหนดให้จุดมอง เคลื่อนที่ในทิศทางต่างๆ เมื่อเทียบกับ Eye โดยยังมีตำแหน่งกล้องอยู่ที่พิกัดเดิม
- SetProjParams
  ใช้กำหนดแอททริบิวต์ของ Matrix Projection
- SetViewParams
  ใช้ตั้งค่าพารามิเตอร์ที่เกี่ยวกับมุมมองของกล้องทั้งจุด Eye, Lookat, Up
- ZoomIn / ZoomOut
  ใช้ปรับมุมมองของกล้องให้น้อยลงหรือมากขึ้น หรือปรับค่า Field Of View ของกล้องให้แคบลงหรือกว้างขึ้น

3.2.2 แอนิเมชัน (Animation)

แอนิเมชันเป็นส่วนที่จะใช้ในการจัดการความเคลื่อนไหวของตัวละครหรือวัตถุในเกม ในส่วนของแอนิเมชันนี้มีคลาสดังนี้

- CDynamicModel

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

- CDObjModel

  เป็น New version ที่พัฒนาขึ้นมาเพื่อเกมประเภท Outdoor Landscape เดินอยู่บนพื้นที่มีลักษณะสูงต่ำ เช่น ภูเขา ได้พัฒนาฟังก์ชันนี้เพื่อการเช็คชนในลักษณะต่างๆ

- CAnimation

  ใช้สำหรับ อ่านข้อมูล Animation จากไฟล์ที่ Export (.MAA) เข้ายัง Memory ไว้พร้อมสำหรับใช้งาน เช่นการอ่านข้อมูล Vertex, Face และ ข้อมูลอื่นๆที่เกี่ยวข้องกับการแสดงผลภาพ Animation แต่ละ Frame

- CAnimationManage

  เป็นตัวจัดการเก็บ Animation Frame ต่างๆ ไว้เป็นคลังข้อมูล เพื่อ ให้ผู้ใช้เข้าถึงข้อมูลเหล่านั้นได้

- CAnimationSet

  เป็นตัวเก็บข้อมูลที่เกี่ยวกับการเคลื่อนไหวท่าทางในหนึ่งท่าของตัวละครในแสดงภาพ Animation ท่านั้นๆ

- CFrame

  เก็บข้อมูลสำหรับการ Blending ของช่วงใดในแต่ละท่าทาง และถูกควบคุมโดย CAnimation

- CUseAnimation

  เป็น class ที่ทำหน้าที่เป็น Interface ให้กับ Programmer ในการใช้งาน Animation

- CModelData

  ทำหน้าที่จัดการข้อมูลโมเดลของวัตถุ มี Interface เพื่อสะดวกในการเข้าถึงข้อมูลต่างๆของ Model Engine

- CModelDataManage

  ทำหน้าที่เป็นตัวจัดการข้อมูลประเภท Model ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model โดยก็จะมาคอยช่วยจัดการไม่ให้มีการ Load ข้อมูลขึ้นมาซ้ำๆ เพื่อประหยัดการใช้งาน Memory

- CUseModelData

  ทำหน้าที่เป็นตัวจัดการ Access ข้อมูลประเภท Model ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model

- CPartDef

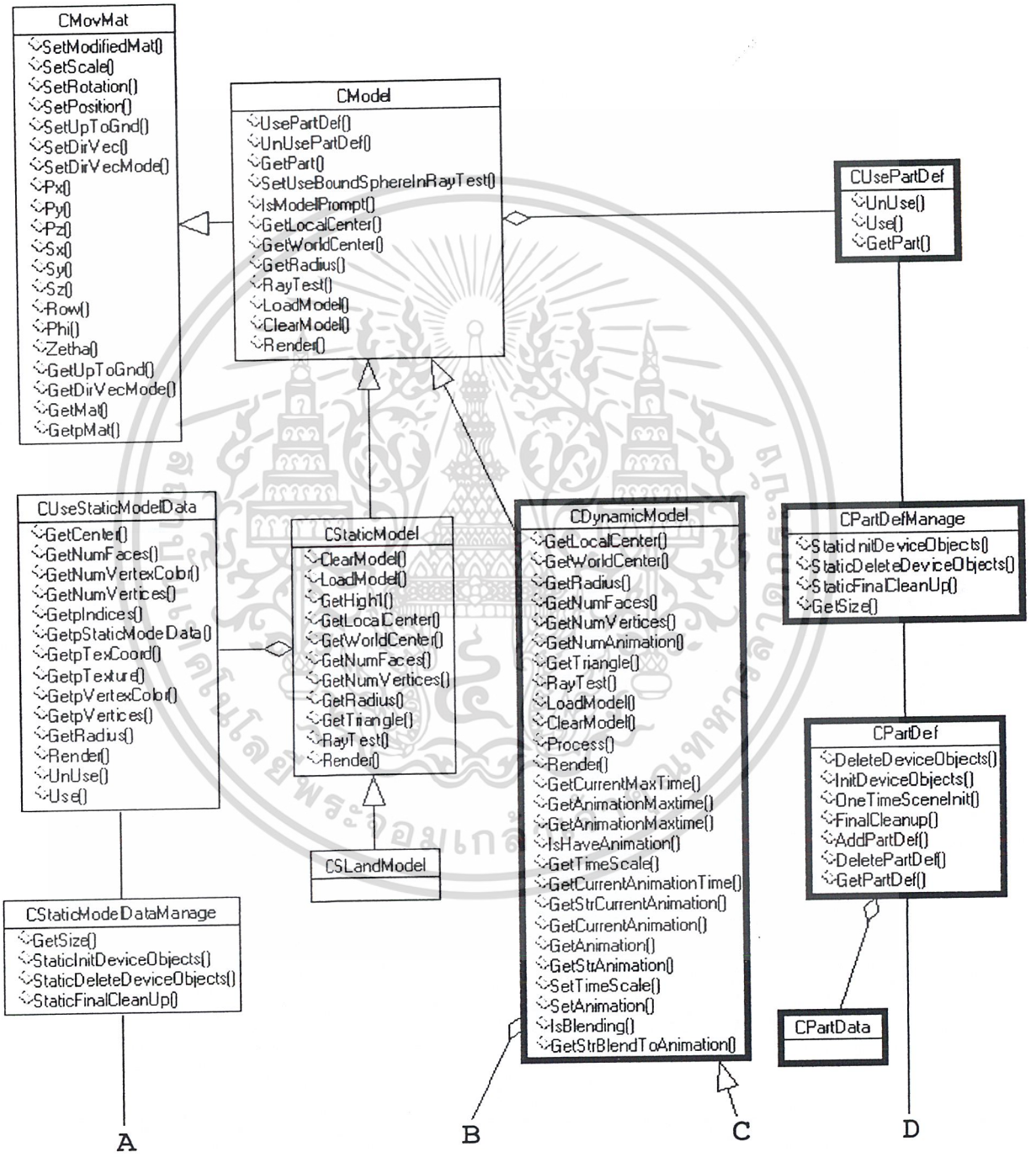
  ทำหน้าที่เป็น เก็บข้อมูลและจัดการข้อมูลประเภท Part ที่จะถูกนำไปใช้ในการกำหนดรูปร่างของตัว Model ที่ถูกอ่านขึ้นมาจากไฟล์ข้อมูลประเภท .Map โดยเป็น Derived Class ของ CAbstract

3.3 คลาสไดอะแกรมของเกมเอนจินท์ (Class Diagram of Game Engine)

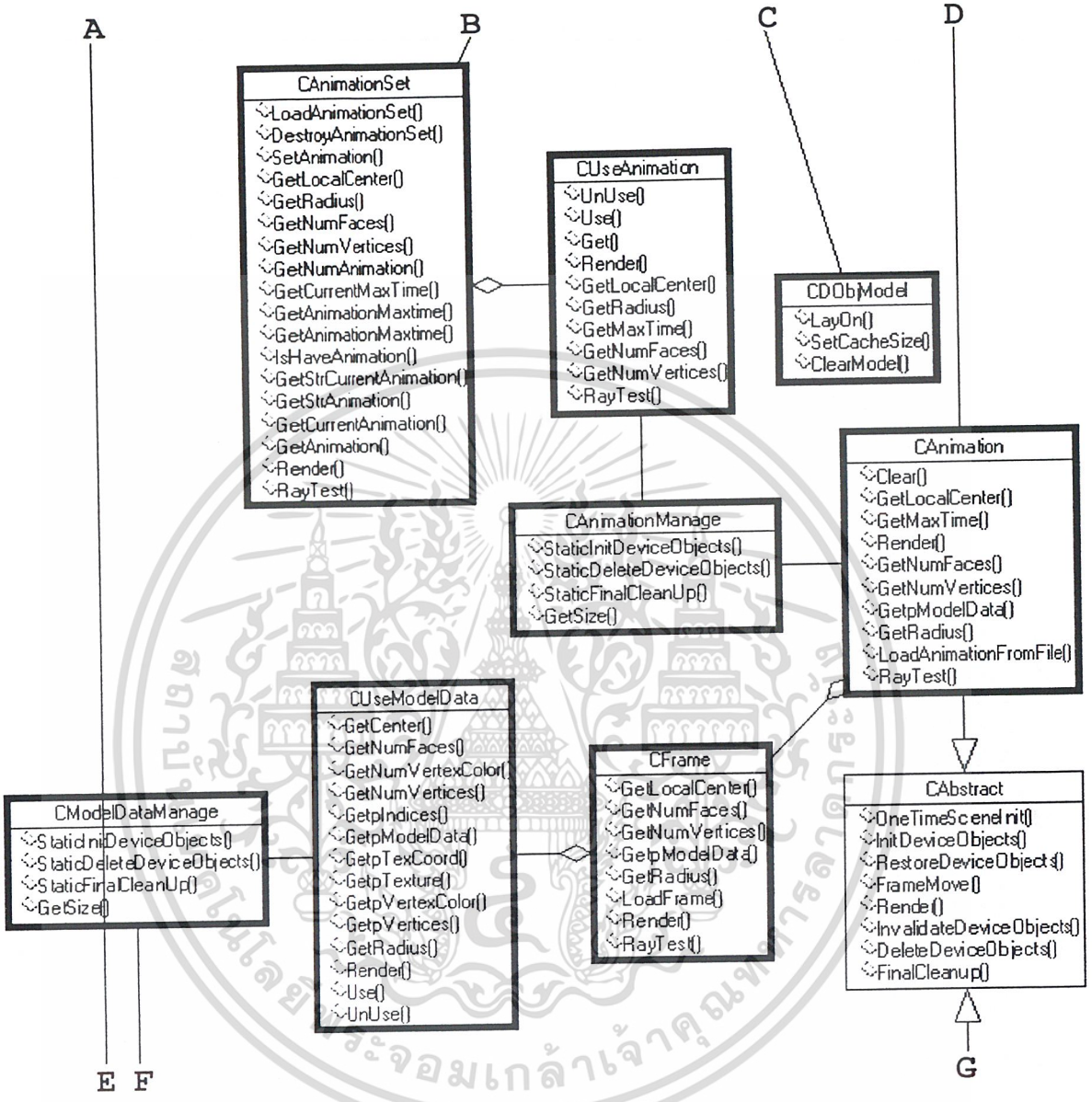
สัญลักษณ์ Class สี ☐ หมายถึง Internal Class of Game Engine

สัญลักษณ์ Class สี ▩ หมายถึง Interface Class of Developer

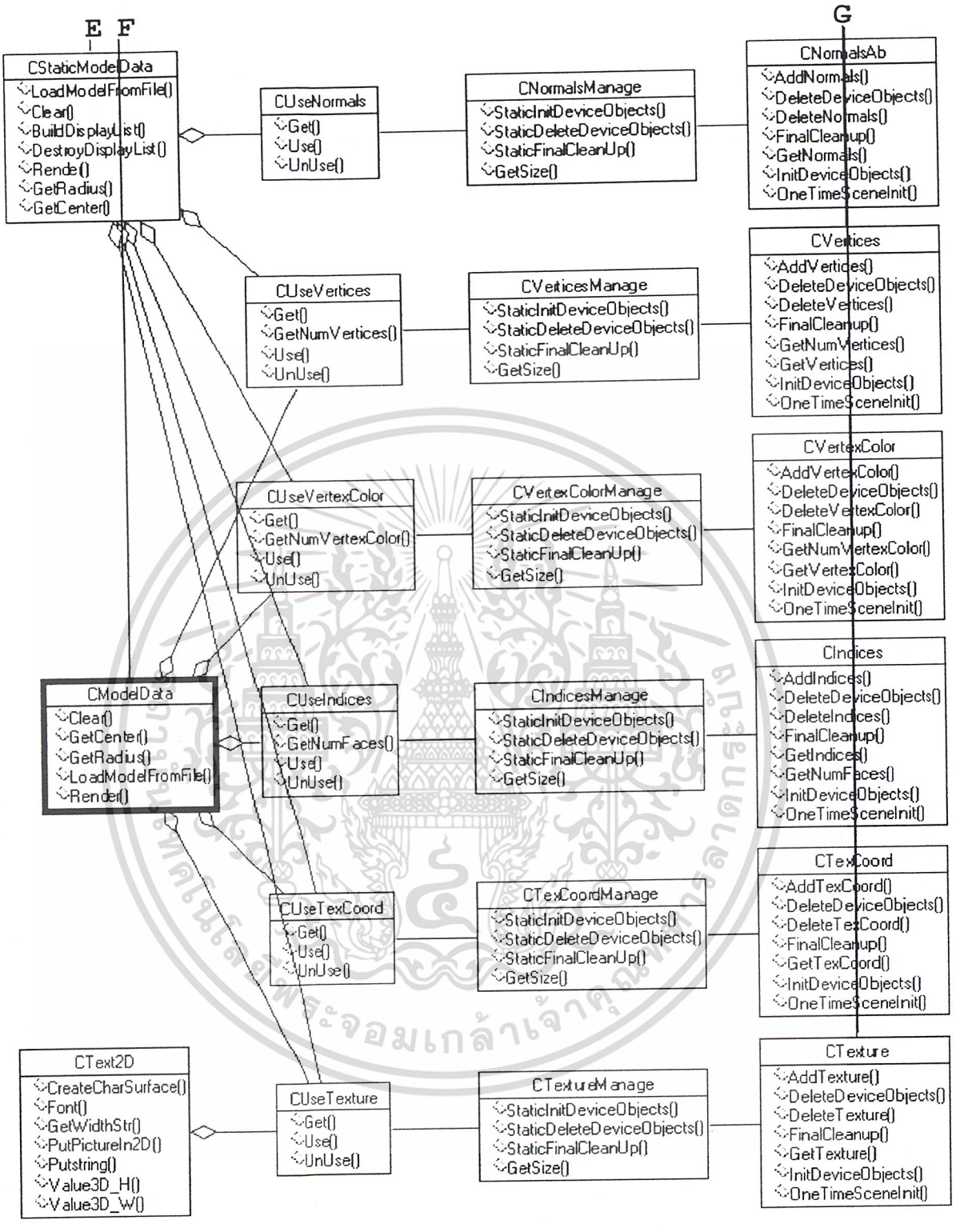
สัญลักษณ์เส้นทึบ ☐ ล้อมรอบ หมายถึง Class ที่จะอธิบายในปริญญานิพนธ์ฉบับนี้



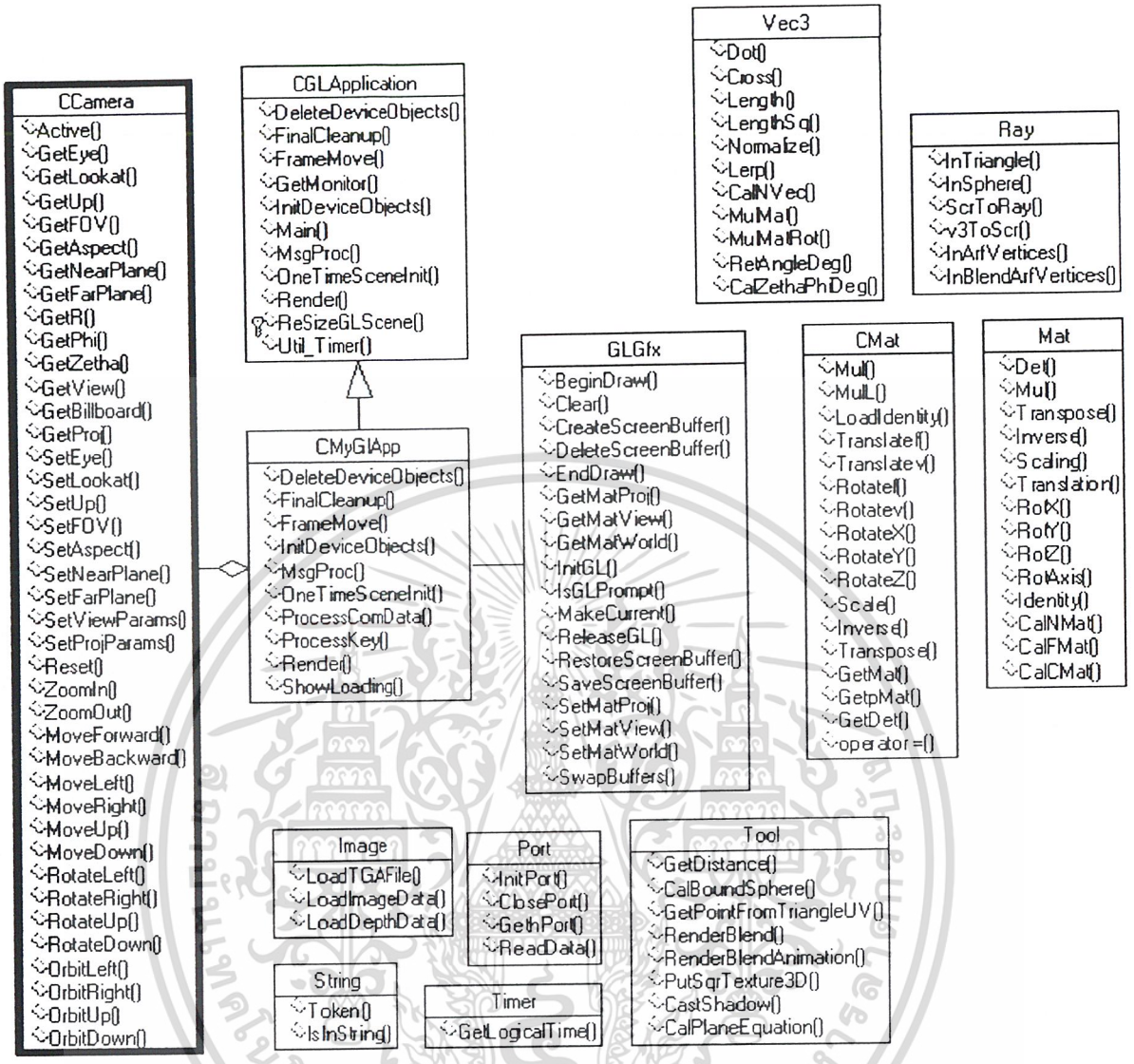
รูปที่ 3-2ก แสดง Class Diagram ของ Game Engine ส่วนที่ 1



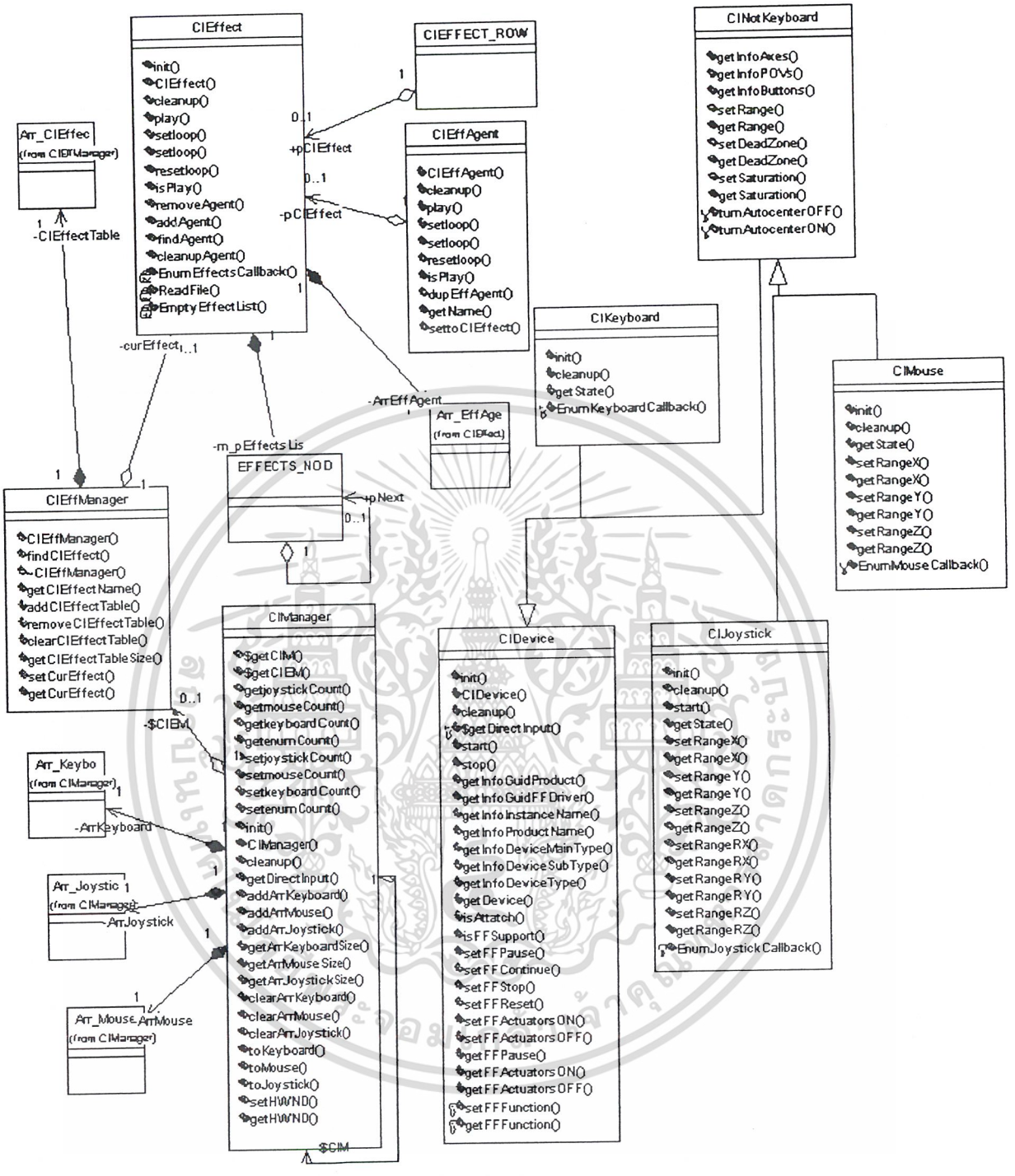
รูปที่3-2ข แสดง Class Diagram ของ Game Engine ส่วนที่ 2



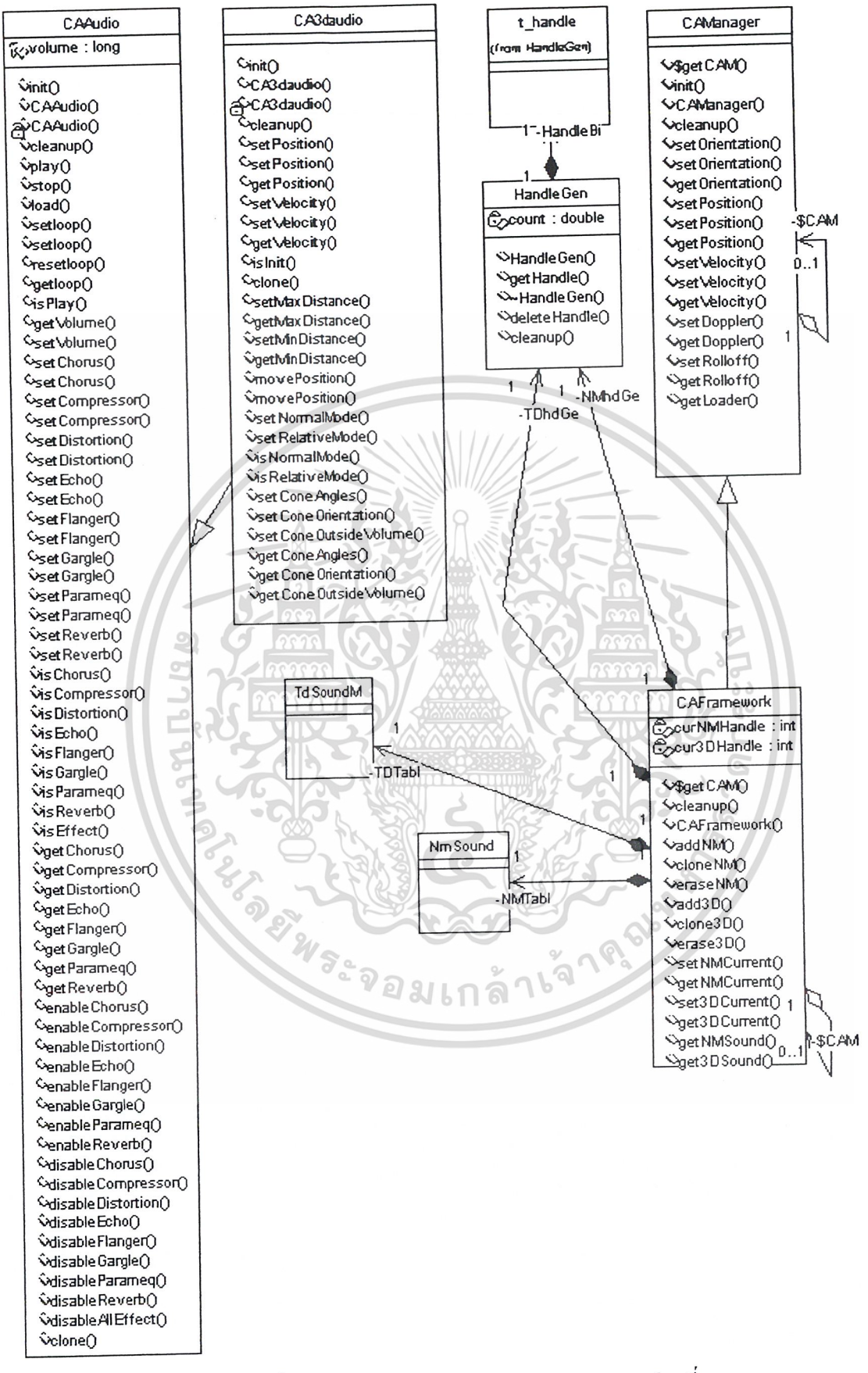
รูปที่3-2ค แสดง Class Diagram ของ Game Engine ส่วนที่ 3



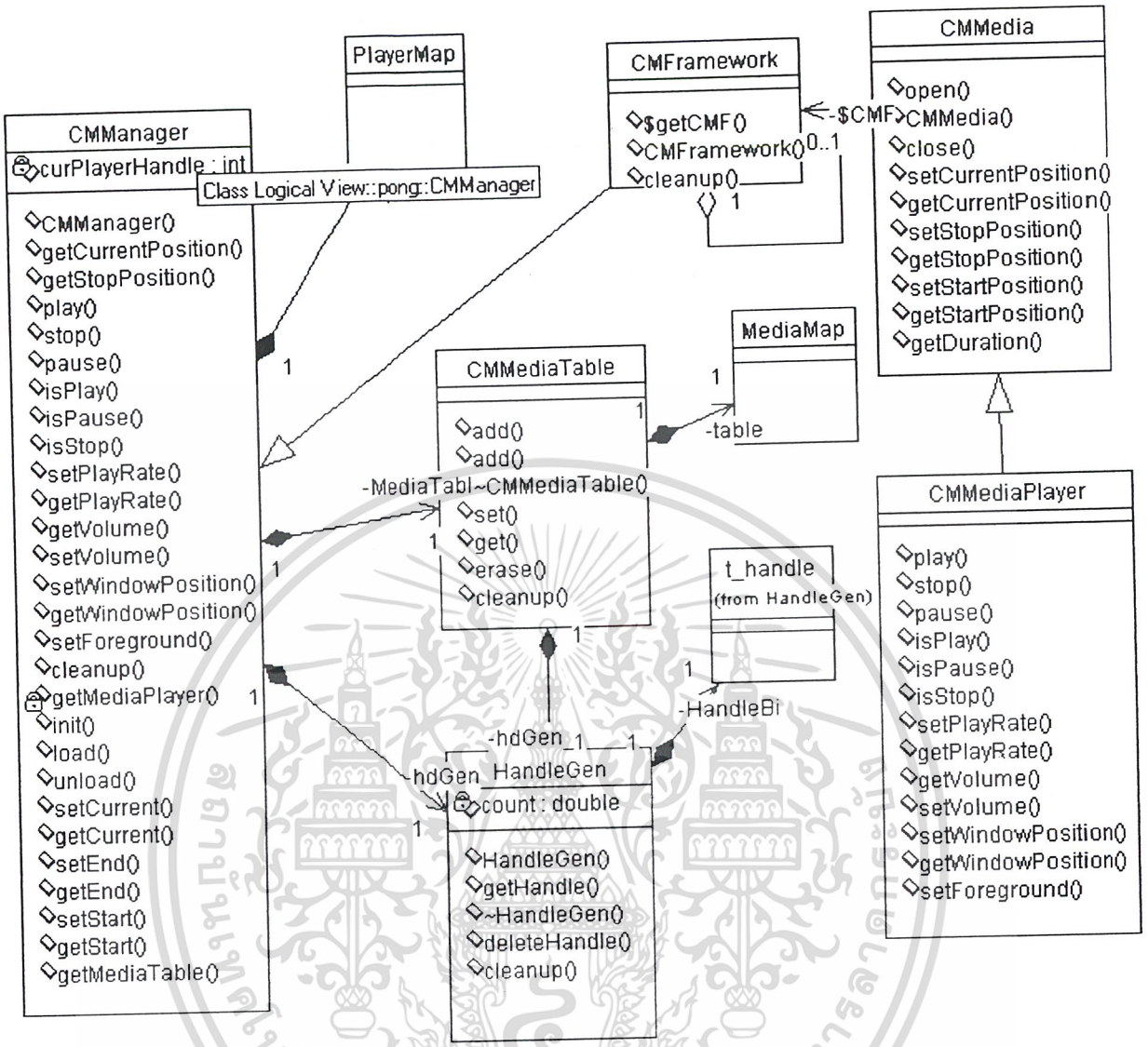
รูปที่3-24 แสดง Class Diagram ของ Game Engine ส่วนที่ 4



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



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



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

บทที่ 4
การพัฒนาเอนจินต์สำหรับเกม 3 มิติ

4.1 ออดิโอ (Audio)

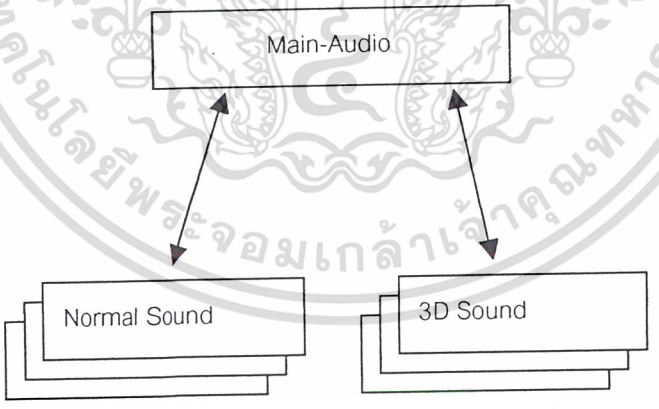
เมื่อเรารู้หลักการทำงาน Direct Audio และทฤษฎีต่างๆแล้ว เราสามารถออกแบบ Audio API ของตัวเองได้ โดยใช้ Object จาก Direct Audio มาใช้ในการเขียนโปรแกรมให้ออกมาเป็นเครื่องมือที่ใช้ในการพัฒนาเกมได้

หลักการออกแบบ

1. ทำให้เขียนโปรแกรมได้ง่ายขึ้น
2. มีความสามารถครบถ้วน เหมาะสำหรับพัฒนาเกม
3. รวดเร็ว
4. มีความเป็นเชิงวัตถุ

4.1.1 Audio API Version 1

Version แรกจะเป็นลักษณะเป็น class 3 อย่าง คือ CAManager ซึ่งมี Function ของผู้ฟังและ Utility ต่างๆ โดยมีลักษณะเป็น Static คือ มีเพียง Object เดียว ส่วน CAAduio และ CA3daudio เป็น class แทนตัวเสียงธรรมดาและ 3 มิติ โดยไม่มีตัวจัดการบริหาร object



รูปที่ 4.1 แสดงความสัมพันธ์ของ Class

CAManager(MainAudio)

เป็น Object ที่ประกาศเป็น Singleton มีหนึ่งเดียวจาก Class CAManager โดยก่อนที่ใช้งาน Audio API จะต้องทำการ Initialize ก่อนและเมื่อสิ้นสุดการทำงานจะทำการ Cleanup ตัวเองด้วย ซึ่ง Object Normal Sound(CAAudio) และ 3D Sound (CA3daudio) จะเรียกใช้ Function อัตถะประโยชน์ต่างๆ ที่จำเป็นจากตัว CAManager ด้วย

ใน CAManager จะประกอบไปด้วย Function แทนตัวผู้ฟัง(Listener)ที่ใช้ตั้งค่าต่างๆในส่วนที่ใช้ในเสียง 3 มิติด้วย

CAAudio (Normal Sound)

เป็น Object แทนเสียงในลักษณะทั่วไป ไม่อยู่ในโลก 3 มิติ หมายถึง มิติจะอยู่ในระบบ Stereo ตามต้นกำเนิดเสียงที่เป็น Wave File ซึ่งจะไม่มี Parameter เกี่ยวกับเสียง 3 มิติ

นอกจากจะมีคุณสมบัติทั่วไปๆ เช่น เล่น,หยุด,ตั้งค่าความดัง ฯลฯ แล้วยังสามารถตั้งค่า Effect ได้ด้วย

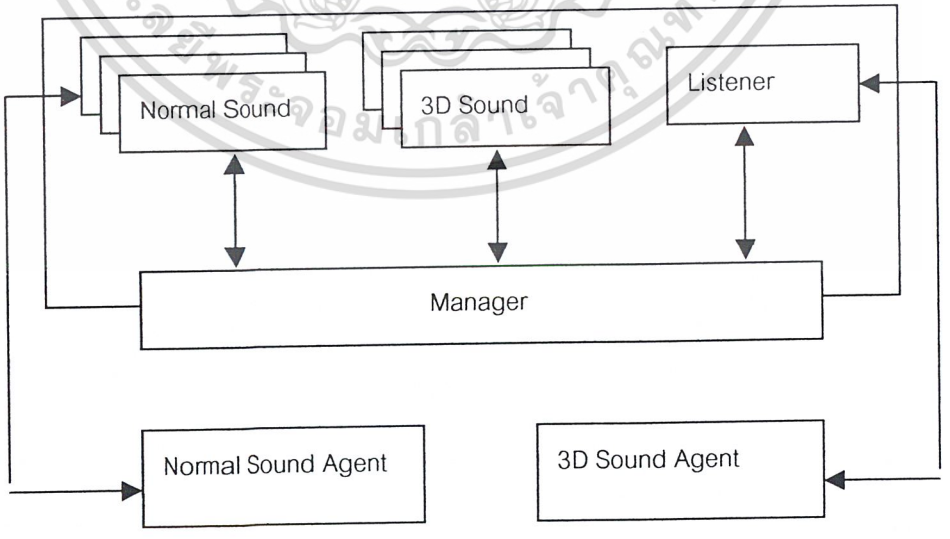
ส่วน Function ซึ่งเกี่ยวกับ Effect จะมีการ Enable, Disable, Set Parameter และ Get Parameter โดยเมื่อต้องการใช้ Effect ใดๆ ก็สามารถ Enable ได้ แต่การที่จะตั้งค่าและดูค่า Parameter ได้ จะต้องทำการ Enable ก่อน และสามารถยกเลิก Effect ได้ด้วยการ Disable

CA3daudio (3D Sound)

เป็น Object เสียง 3 มิติ โดยสืบทอดมาจาก CAAudio ดังนั้นจะมี Function ต่างๆที่เพิ่มเข้ามาจะเป็น Function ที่เกี่ยวข้องกับเสียงใน 3 มิติทั้งสิ้น

4.1.2 Audio API Version 2

จาก Version แรกจะเห็นได้ว่าไม่มีส่วนจัดการบริหาร Object โดยซ่อน CAAudio และ CA3daudio เอาไว้และติดต่อ โดยการอ้างอิงเป็นชื่อและใช้ Agent เป็นตัวแทน



รูปที่ 4.2 แสดง Architecture ของ Audio API Version 2

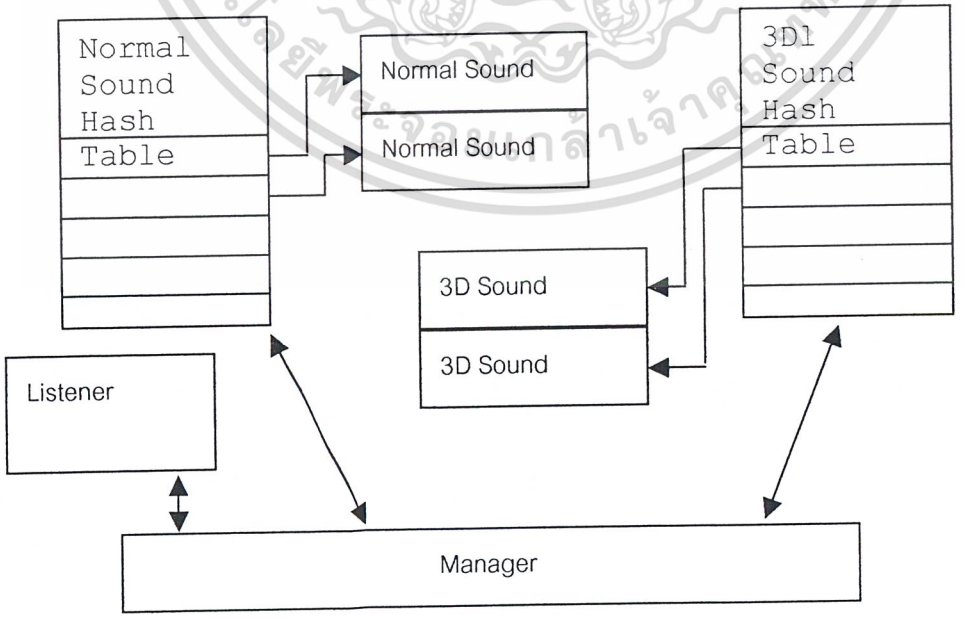
อธิบาย Architecture ของ Audio API version 2

- Manager เป็นตัวจัดการและตัวกลางระหว่าง Application กับ Object ที่อยู่ภายใน ซึ่งมี Function ที่ไว้จัดการเกี่ยวกับ Object ภายในโดยตรง
- Normal Sound เป็น Object จาก Class CAAudio
- 3D Sound เป็น Object จาก Class CA3daudio
- Agent เป็น Object เสมือนเป็นตัวกลางของ Object เสียงที่อยู่ภายใน แต่ไม่ได้เก็บเสียงจริงๆไว้
- Listener เป็น Object แทนตัวผู้ฟังในโลก 3 มิติ

4.1.3 Audio API Version 3

จาก Audio Library Version 2 ได้ปรับปรุงจนมาเป็น version ล่าสุด โดยลดส่วนที่ซับซ้อนลง และให้ใช้งานได้ง่ายยิ่งขึ้น โดยใช้ Namespace มาช่วยในการลดความยาวของชื่อ Function และตัดส่วนที่เป็น Agent ออกไป แล้วมาใช้การอ้างอิง Object โดยใช้ Map ของ Standard Template Library แทน

Map มีลักษณะเป็น Hash โดยมี key เป็นตัวอ้างอิง ทำให้ทำการ Search ได้เร็ว ไม่ว่า object จะเก็บอยู่ส่วนไหนก็ตาม โดนเราใช้ handle number ของ object sound เป็น key ทำให้ไม่จำเป็นต้องมี object agent ต่อไป โดยเราใช้ Handle ในการอ้างอิงแทน



รูปที่ 4.3 แสดง Architecture ของ Auido API Version 3

อธิบาย Architecture ของ Audio API version 3

- Manager เป็นตัวจัดการและตัวกลางระหว่าง Application กับ Object ที่อยู่ภายใน ซึ่งมี Function ที่ไว้จัดการเกี่ยวกับ Object ผ่านทาง Hash Table
- Hash Table จะมีสองตารางคือ Normal Sound และ 3D Sound ซึ่งจะมี Key เป็น Handle Number แล้วจะมี Pointer ชี้ไปยัง Sound Object อีกที
- Normal Sound เป็น Object จาก Class CAAudio
- 3D Sound เป็น Object จาก Class CA3daudio
- Listener เป็น Object แทนตัวผู้ฟังในโลก 3 มิติ

4.2 อินพุต (Input)

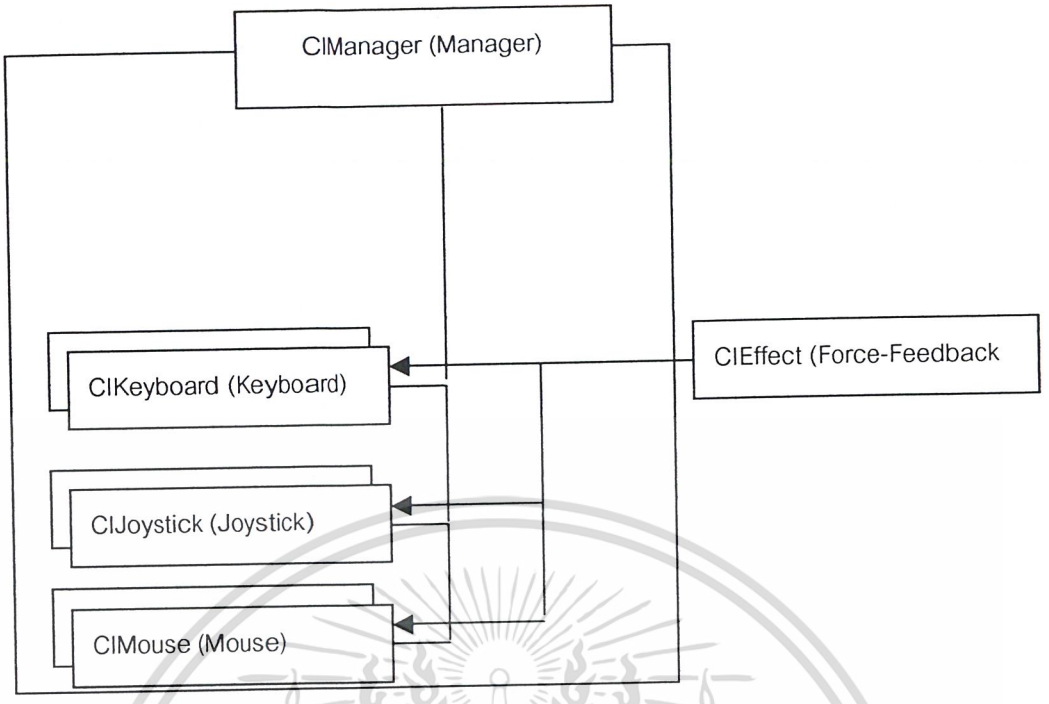
เมื่อเราพิจารณาโครงสร้างและการทำงานของ Direct Input แล้ว ก็สามารถออกแบบส่วน Input API โดยนำ Direct Input มาเป็นพัฒนาเพิ่มเป็น API ของเรา เพื่อให้โปรแกรมเมอร์ใช้งานได้สะดวกมากยิ่งขึ้น

หลักการออกแบบ

1. ให้โปรแกรมเมอร์ใช้งานได้ง่าย
2. มีความเหมาะสมกับการเขียนโปรแกรมเกม
3. มีความสามารถเพียงพอในการใช้งาน

4.2.1 Input API Version 1

ในการพัฒนาขั้นแรก เรานำ Direct Input มาสร้างเป็น Class ที่เป็นพื้นฐานก่อน โดยให้สามารถใช้งานได้ในระดับหนึ่ง ก่อนที่จะนำไปพัฒนาให้ดีขึ้นในขั้นต่อไป



รูปที่4.4 แสดงความสัมพันธ์ของ Class ใน Input API version 1

อธิบาย Architecture ของ Input API version 1

- Manager (CIManager)

เป็น Object ที่ประกาศเป็น Singleton มีหนึ่งเดียวจาก Class CIManager โดยจะเก็บ Pointer ของ Device Object(Keyboard ,Mouse ,Joystick) ไว้ โดยอ้างอิงจาก Application โดยใช้ Handle แทน Pointer ซึ่ง Manager จะเป็นจัดการ บริหาร Device Object รวมทั้ง Function อรรถประโยชน์ต่างๆอีกด้วย

- Device Object(CIKeyboard, CIMouse, CIJoystick)

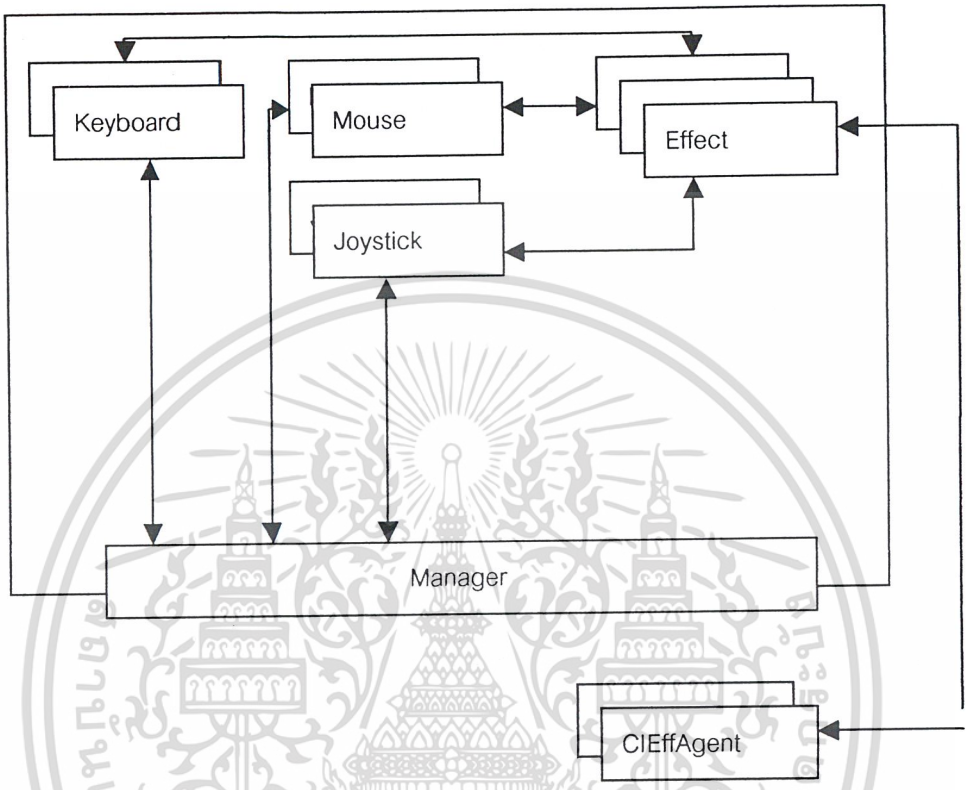
เป็น Object แทนตัว Device โดยแบ่งออกเป็น 3 ชนิดคือ Keyboard, Mouse และ Joystick โดยจะมี Function ต่างๆ แทนตัว Device นั้นๆ โดยจะถูกซ่อนเอาไว้โดย Manager โดยติดต่อผ่านทาง Manager โดยการใส่ Handle ที่ต้องการแล้ว Return Pointer

- Effect

แทนตัว Force-Feedback ซึ่งสร้างจาก File.FFE ซึ่งสร้างจากโปรแกรม Force-Ediotr โดยจะยึดติดกับ Device ที่สนับสนุน Force-Feedback หมายความว่า หากจะสร้าง Effect ได้จะต้องมี Device ที่สนับสนุน Force-Feedback ก่อน

4.2.2 Input API Version 2

มีการพัฒนาโดยจะมีการใช้ติดต่อกับ Object ที่เป็น Input Device โดยอ้างอิงเป็น Handle โดยแบ่งออกเป็น Keyboard, Mouse, Joystick โดยมีส่วน Force Feedback Effect ด้วยโดยมี Class CIEffAgent เป็นตัวกลางติดต่อกับผ่าน Object Effect ภายใน



รูปที่ 4.5 แสดง Architecture ของ Input API version 2

อธิบาย Architecture ของ Input API Version 2

- Manager เป็นส่วนติดต่อกับ Application โดยโปรแกรมเมอร์สามารถเรียก Function ต่างๆ ผ่านทาง Manager เพื่อเรียกบริการจาก Object ภายในได้
- Device(Keyboard,Mouse,Joystick) แทน Input Device ต่างๆ โดยแต่ละชนิดสามารถมีได้หลายๆ Device เช่น Joystick สองตัว เป็นต้น
- Effect เป็น Object แทนตัว Force Feedback Effect โดยจะติดต่อการสร้าง Effect กับ Device ผ่านทาง Manager หรือ CIEffAgent
- Effect Agent หรือ CIEffAgent Class เป็นเสมือนตัวแทนของ Effect ซึ่งสามารถสร้างจาก Effect เพื่อใช้งานและทำลายโดยไม่กระทบต่อ Effect จริงๆ โดยมีความสัมพันธ์แบบ Many-to-One กับ Effect(Effect สามารถมีได้หลาย Agent)

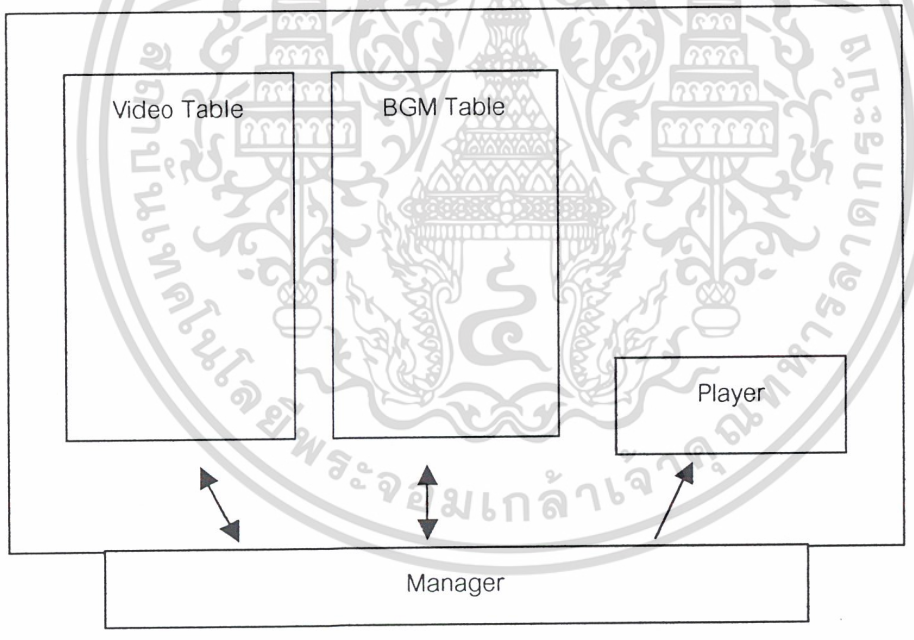
4.3 มิเดีย (Media)

เมื่อเราพิจารณาถึงการใช้งานในส่วนของการใช้งาน Direct Show ในเกมต่างๆ เกมทั่วไปจะใช้ส่วนของการเล่น Video หรือ Background Music เท่านั้น โดยขณะที่เล่น Video นั้นจะแทบไม่มีการทำงานอื่นๆอยู่เลย เช่น การเล่น Video ตอนเริ่มเกม, Logo ของบริษัท, ฉากจบของเกม ซึ่งจะเป็นการเล่น Video ทีละ 1 file ดังนั้นการออกแบบ Media API ในช่วงแรกจะใช้ในส่วนที่จำเป็นเท่านั้น โดยตัดส่วนที่ไม่จำเป็นออกไป

หลักการออกแบบ

1. สามารถใช้งานได้ง่าย
2. มีความสามารถเล่น Video ใน Mode Full Screen และ BGM ในขณะเล่นเกมได้
3. มีส่วนจัดการที่ดี

4.3.1 Media API Version 1



รูปที่ 4.6 แสดง Architecture ของ Media API version 1

Architecture ของ Media API version 1

จะเห็นได้ว่าโครงสร้างของ API นั้นไม่ซับซ้อน โดยเป็นตัวจัดการเกี่ยวกับการเล่น BGM และ Video ครั้งละ 1 file

- Manager เป็นตัวติดต่อกับ Application และเป็นสั่งการต่างๆ ไม่ว่าจะเป็นการเล่น Video และ BGM หรือจัดการข้อมูลต่างๆ โดย Application จะติดต่อส่วนที่อยู่ข้างในผ่านทาง Manager เท่านั้น
- Video Table และ BGM Table เป็นตารางเก็บข้อมูลที่จำเป็นต่อการเล่น file เช่น ชื่อ File ฯลฯ
- Player เป็นตัวเล่น file โดย Manager จะนำข้อมูลจากตารางมาสร้างเป็น Player เป็นใช้ในการเล่น 1 ครั้ง

รายละเอียดของ Manager

Manager จะเป็น Object เสมือน โดยจะประกอบไปด้วย function ต่างๆที่ใช้ในการบริหารจัดการกับ Video Table และ BGM Table โดยจะเป็นสร้าง Player เมื่อมีการเล่นเกิดขึ้นและทำลายเมื่อการเล่นสิ้นสุด

รายละเอียดของ Video Table และ BGM Table

จาก Architecture ไม่มีความจำเป็นที่จะเก็บ Video และ BGM เป็นลักษณะ Object เพราะการเล่นจะกระทำเพียงหนึ่งครั้งในช่วงหนึ่งเวลา และเมื่อมีการเล่นก็จะสร้าง Player ซึ่งเป็น Temporary Object ขึ้นมาและถูกทำลายทิ้งโดย Manager โดยรายละเอียดของตาราง Video และ BGM เป็นดังนี้

Video Table

Name	Filename	SrcRect	DesRect	Volume	Rate	Start	End
Open	Open.avi	0,0,320,240	0,0,320,240	100	1	0	2.3×10^7
Ending	End.mpg	0,0,320,240	0,0,320,240	100	1	0	100×10^7
Logo	Logo.avi	0,0,320,240	0,0,320,240	80	1.2	0	3.1×10^7

ตารางที่ 4.1 ตัวอย่างตาราง Video

Name เป็นชื่อใช้ในการอ้างอิง

Filename เป็นชื่อ file ของ Media ซึ่งสนับสนุน MPEG, AVI, QuickTime, WAV,AIFF,AU,SND และ MIDI

SrcRect เป็นกรอบของสี่เหลี่ยมของขอบเขตของต้นฉบับของ Video

DesRect เป็นกรอบของสี่เหลี่ยมของขอบเขตของขนาดที่แสดงผลบนหน้าจอ

Volume เป็นความดังของเสียง มีค่าระหว่าง 0-100

Rate คือค่าอัตราการเล่นของเสียง ปกติจะมีค่าเป็น 1.0

Start, End จุดเริ่มและจุดสิ้นสุด หน่วยเป็น 100 nanoseconds

Background Music Table

Name	Filename	Volume	Rate	Start	End	Loop
World Map	Map.mp3	100	1	0	182.2×10^7	Yes
Dead	Dead.wav	100	1	0	10×10^7	No
Fight	Fight.mp3	73	1.5	0	221.23×10^7	Yes

ตารางที่ 4.2 ตัวอย่างตาราง Background Music

Name เป็นชื่อใช้ในการอ้างอิง

Filename เป็นชื่อ file ของ Media ซึ่งสนับสนุน MPEG, AVI, QuickTime, WAV,AIFF,AU,SND และ MIDI

Volume เป็นความดังของเสียง มีค่าระหว่าง 0-100

Rate คือค่าอัตราการเล่นของเสียง ปกติจะมีค่าเป็น 1.0

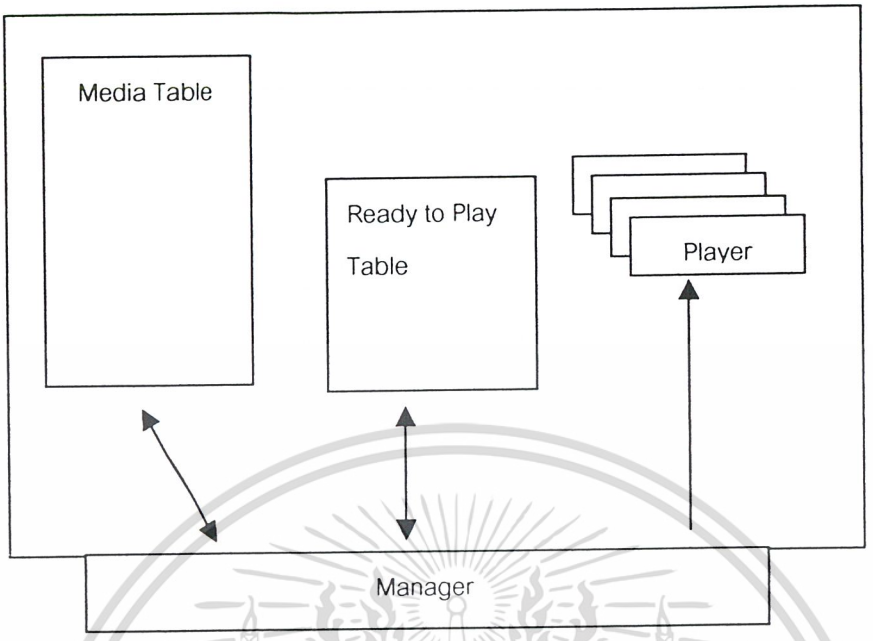
Start, End จุดเริ่มและจุดสิ้นสุด หน่วยเป็น 100 nanoseconds

4.3.2 Media API Version 2

จาก Version ที่ผ่านมา จะเห็นได้ออกว่าออกแบบมามีข้อเสียอยู่ คือ ออกแบบไม่ให้เล่น Video และ BGM ไปพร้อมๆกัน และไม่สามารถเล่นพร้อมๆกันหลาย file ได้

จึงได้ออกแบบให้สามารถเปิดเล่นพร้อมๆกันได้หลายๆ file และไม่แยกชนิดของ Media เป็น BGM และ Video เช่นเดิม โดยมองว่าเป็น Media แบบเดียวกัน โดยหากเป็น file ที่ไม่มีภาพ Video เมื่อใช้ Function ที่เกี่ยวกับภาพ เช่น ตั้งค่าขนาด, ตั้งตำแหน่งของการแสดง ฯลฯ จะไม่เกิดผล

และเช่นเดียวกับ Library อื่นๆ โดยปรับปรุงโดยใช้ Namespace ในการย่อชื่อให้สั้นลงและแบ่ง Section ได้สะดวกยิ่งขึ้น



รูปที่ 4.7 แสดง Architecture ของ Media API version 2

Architecture ของ Media API version 2

จะเห็นได้ว่า Table จะรวมเป็นตารางเดียว ไม่แยกว่าเป็น Video หรือ BGM และ Player จะมีจำนวนเท่ากับ Media ที่ Load ขึ้นมาเล่น ซึ่งเป็นส่วนแก้ไขจาก Version แรก โดย Structure ของตารางมีการแก้ไขดังนี้

Handle	filename	volume	rate	start	end	Window_pos
0	Open.avi	100	1.2	0	172×10^7	0,0,320,240
1	End.mpg	100	1.0	0	122×10^7	0,0,320,240
2	Wolf.mp3	100	1.0	0	332×10^7	0,0,0,0

ตารางที่ 4.3 ตัวอย่างตารางใน Media API version 2

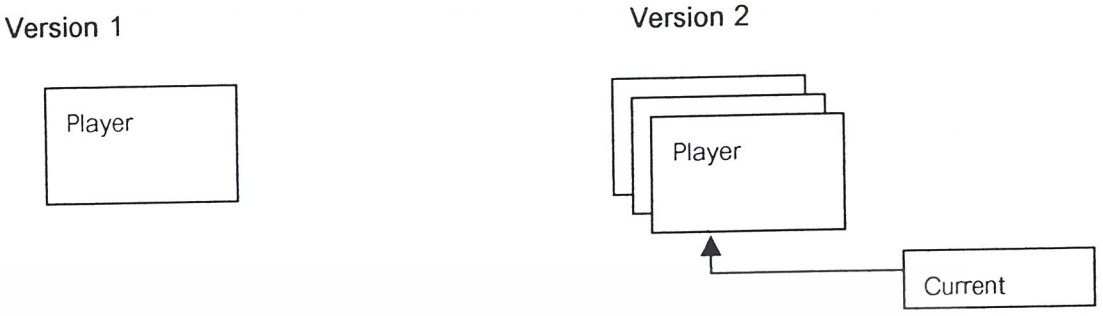
จะเห็นได้ว่าหากเป็น File ที่ไม่ใช่ Video, Structure ของ Window จะไม่ได้ใช้

4.3.3 จุดแก้ไขจาก Version 1 มายัง Version 2

การเล่น File หลายๆ File พร้อมๆกัน

ในการเล่น File ซึ่งสามารถ Load ขึ้นมาเล่นได้ทีละ 1 file ไม่สามารถเล่น file หลายๆ file พร้อมกันได้ โดยใน Version 2 แก้ไขให้สามารถเล่นพร้อมๆกัน โดยใน Version 2 จะสร้าง Object Player เพื่อ

ใช้ในการเล่นตามจำนวนที่ Load ขึ้นมา โดยการสั่งการ Media ต่างๆกันทำได้โดยอ้างอิง Handle ที่ได้จากการ Load ขึ้นมายัง Playing Table



รูปที่ 4.8 เปรียบเทียบการเล่นระหว่าง Version 1 และ 2

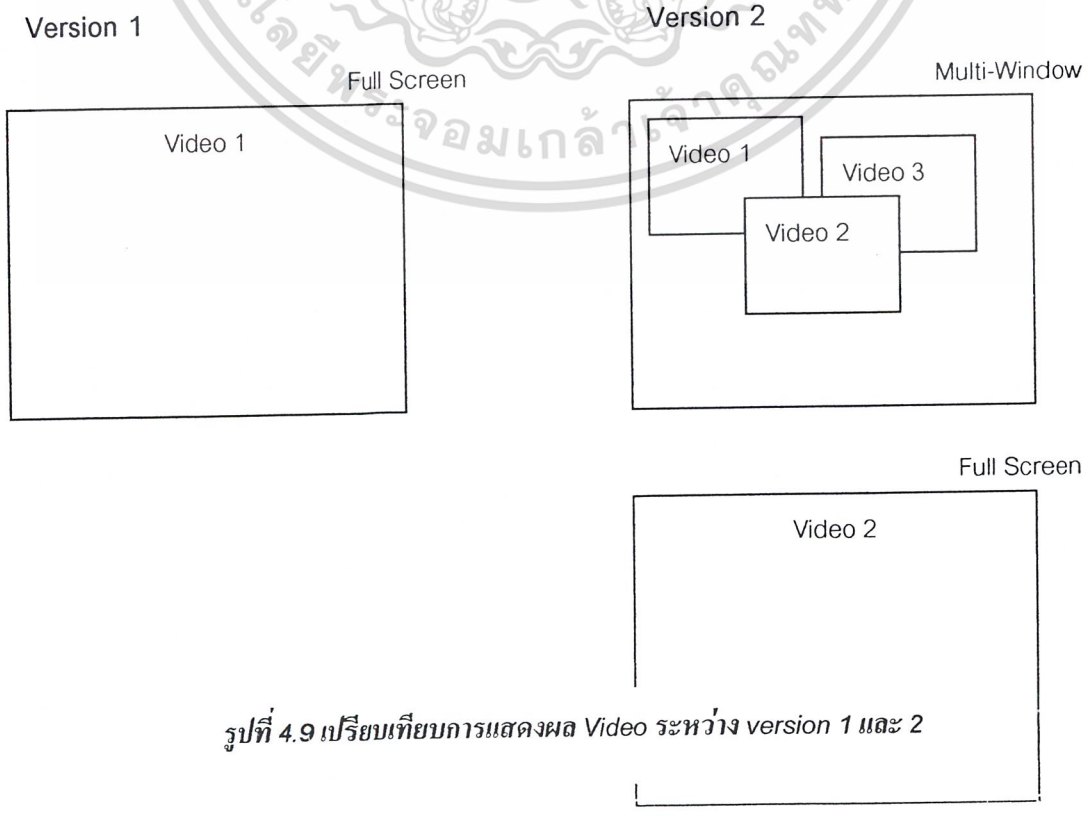
การเล่น Video

ใน Version แรกการเล่น Video จะเป็นลักษณะ Full Screen ซึ่งจะมีขนาดเต็มจอ แต่ใน Version 2 การเล่น Video สามารถเล่นได้พร้อมๆกัน โดยใช้การสร้าง Child Window ซึ่งเป็น Window ที่อยู่ใน Window Application อีกที โดยใน Child Window จะแสดงผลเป็นภาพ Video โดยสามารถขยายขนาดได้ตามใจชอบ โดย Child Window สามารถตั้งการซ้อนทับได้โดย

int DM::setForeground();

ซึ่งจะนำ Video ที่เป็น Current ไปไว้บนสุด

หากต้องการ Mode Full-Screen ก็สามารถทำได้โดย ขยาย Window ให้เต็มจอ และขยาย Child Window ที่ต้องการให้เท่ากัน



รูปที่ 4.9 เปรียบเทียบการแสดงผล Video ระหว่าง version 1 และ 2

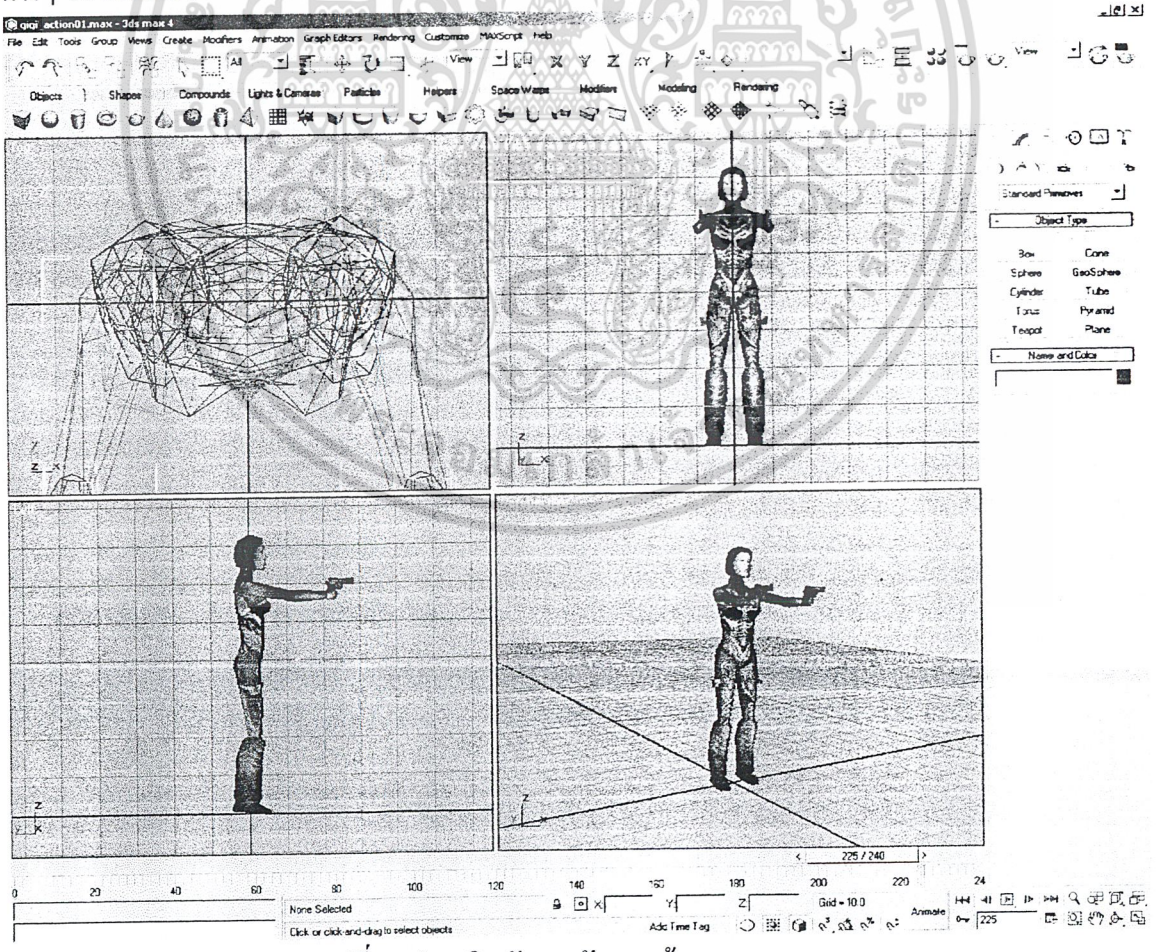
บทที่ 5
การใช้งานเกมเอนจินต์ (Using Game Engine)

5.1 ขั้นตอนการจัดเตรียมข้อมูลเพื่อทำแอนิเมชันของตัวละคร

การเคลื่อนไหวในรูปแบบเวอร์เท็กซ์บเลนดิง ข้อมูลของวัตถุที่อ่านหรือโหลดขึ้นมาได้นั้นจะเป็นลักษณะของวัตถุที่อยู่นิ่งกับที่ ข้อมูลที่จัดเก็บจะเก็บเป็นเวอร์เท็กซ์ของวัตถุที่อยู่นิ่งกับที่ โดยแบ่งเป็นช่วงของการเคลื่อนไหวในลักษณะต่างๆ การทำการเคลื่อนไหวจึงจะต้องอาศัยเทคนิคการอินเตอร์โพเลตมาช่วยทำให้การทำการเคลื่อนไหวมีความราบรื่น จึงเป็นหน้าที่ของผู้พัฒนาเอง โดยมีหลักการและวิธีการที่จะต้องสร้างและจัดเตรียมข้อมูลการเคลื่อนไหวให้มีความเหมาะสมกับลักษณะท่าทางที่แตกต่างกันของตัวละคร ที่มีกลไกในการเคลื่อนไหวของวัตถุนั้นเองโดยอาศัยข้อมูลการเคลื่อนไหวของวัตถุที่เตรียมไว้ การจัดเก็บข้อมูลการเคลื่อนไหวนี้จึงเป็นขั้นตอนหนึ่งในการพัฒนาแอนิเมชันของตัวละครในเกมสามมิติ

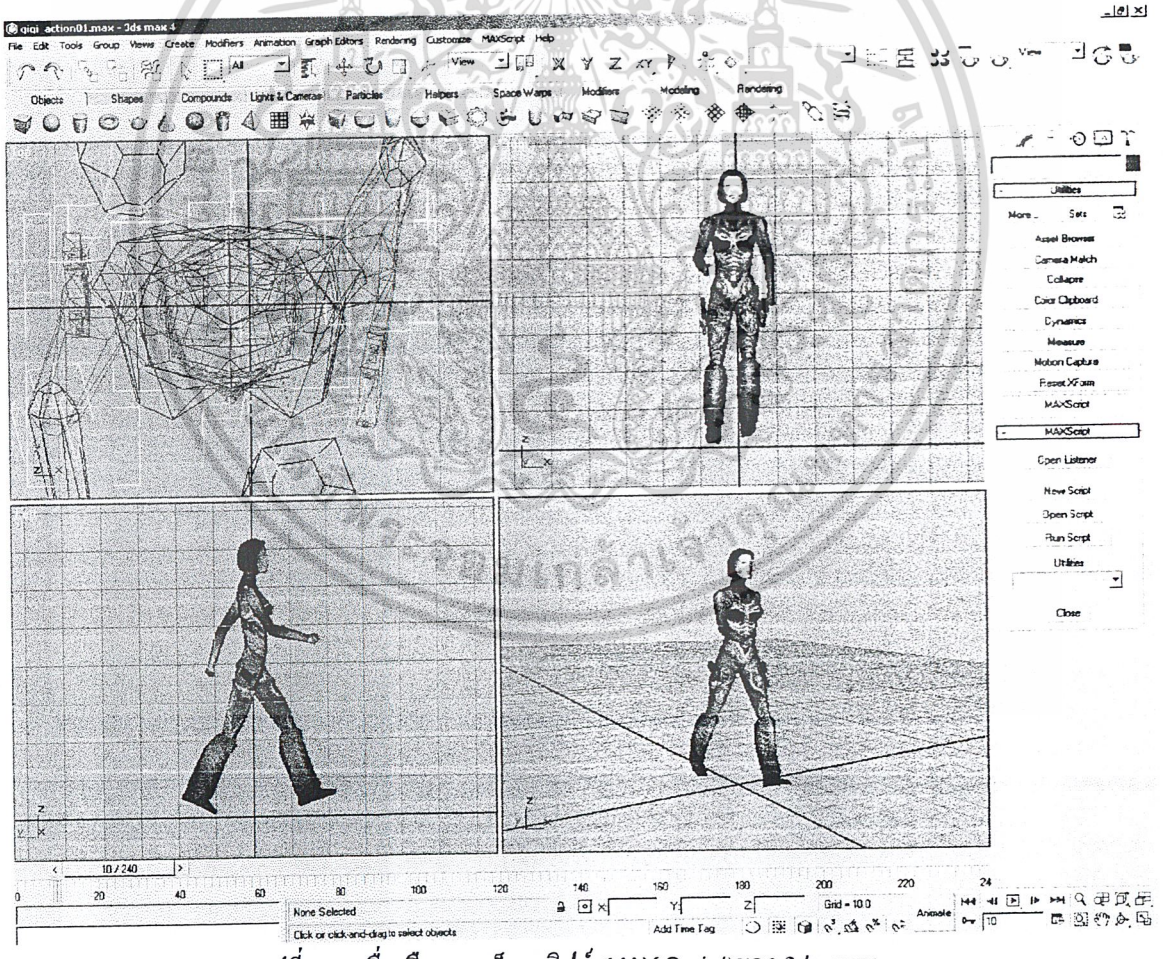
5.1.1 เอ็กพอร์ตแอนิเมชันด้วย Maxscript ของ 3ds-max

สร้างแอนิเมชันของตัวละครด้วยโปรแกรม 3ds-max ก่อนที่จะทำการเอ็กพอร์ต ในกริยาท่าทางต่างๆ ของตัวละคร



รูปที่ 5.1 ทำแอนิเมชันของตัวละครด้วย 3ds-max

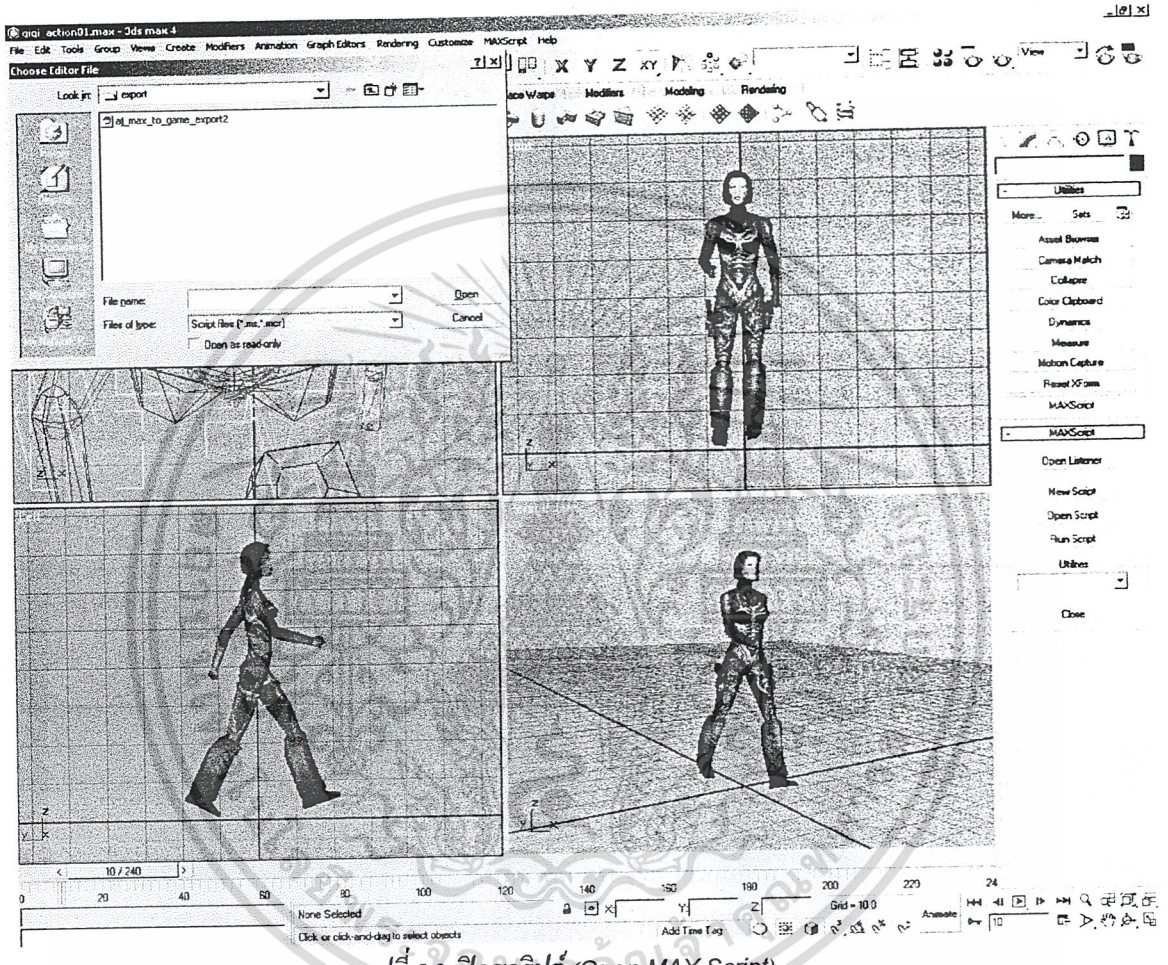
เมื่อได้แอนิเมชันของตัวละครแล้วจะเป็นขั้นตอนการเอ็กพอร์ตไฟล์ฟอร์แมตของ 3ds-max เพื่อให้ได้ไฟล์ฟอร์แมต 5 แบบที่มีส่วนขยายดังนี้ mac, maf, mai, map และ mau ซึ่งเป็นฟอร์แมตที่จะสามารถนำไปใช้ในการพัฒนาเกม เลือกเฟรมที่จะทำการเอ็กพอร์ตแล้ว ใช้เครื่องมือ Max Script ในแท็ปยูทิลิตี้  ของ 3ds-max ซึ่งเป็น Script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



รูปที่ 5.2 เครื่องมือของแม็กสคริปต์ (MAX Script)ของ 3ds-max

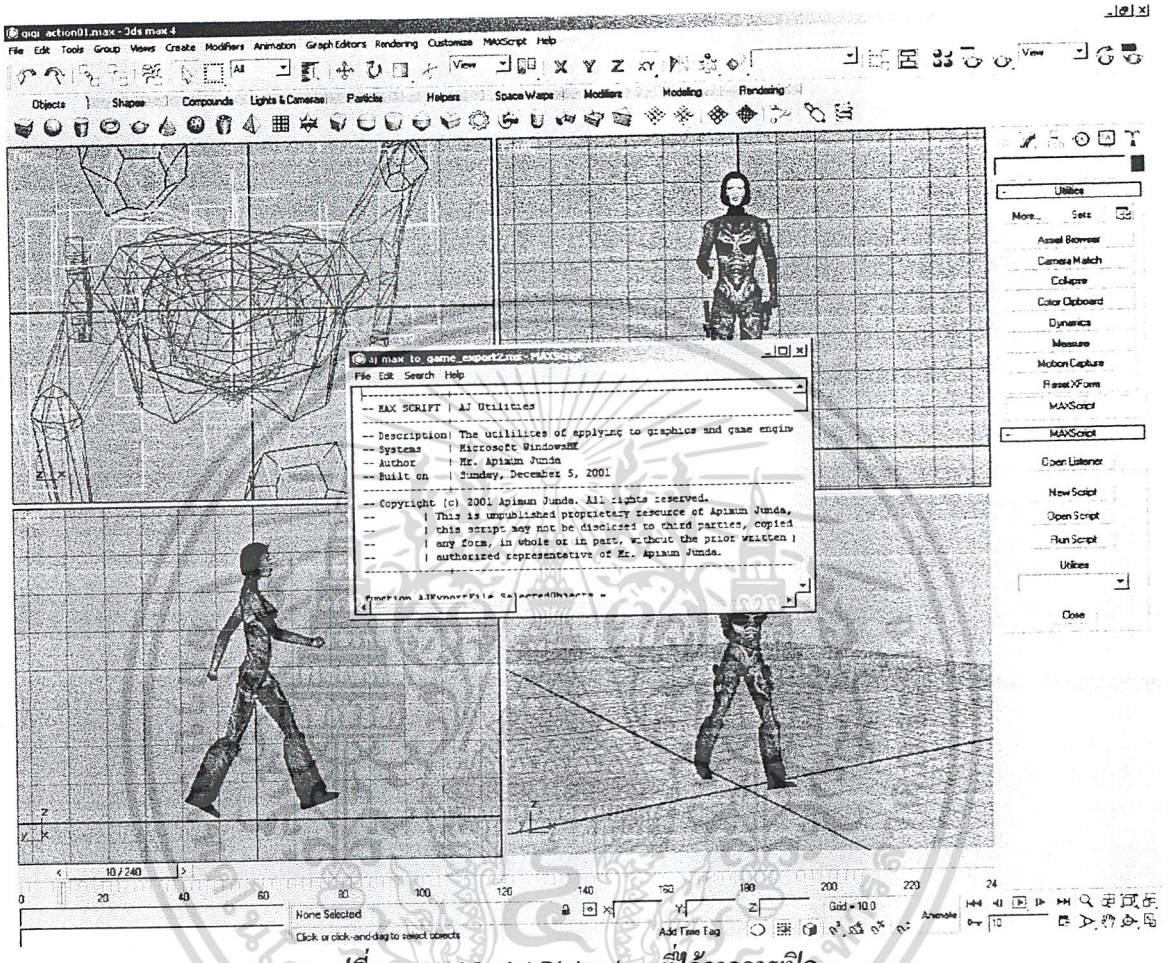
5.1.1.1 เปิดสคริปต์ (Open script) ที่ใช้ในการเอ็กพอร์ต

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



รูปที่ 5.3 เปิดสคริปต์ (Open MAX Script)

จะได้ Dialog box เล็กๆ ขึ้นมาใหม่ ซึ่งจะเป็นไฟล์ Max Script ที่ได้ทำการเปิด



รูปที่ 5.4 MAX Script Dialog box ที่ได้จากการเปิด

5.1.1.2 แก้ไขสคริปต์ที่จะใช้ในการเอ็กพอร์ต

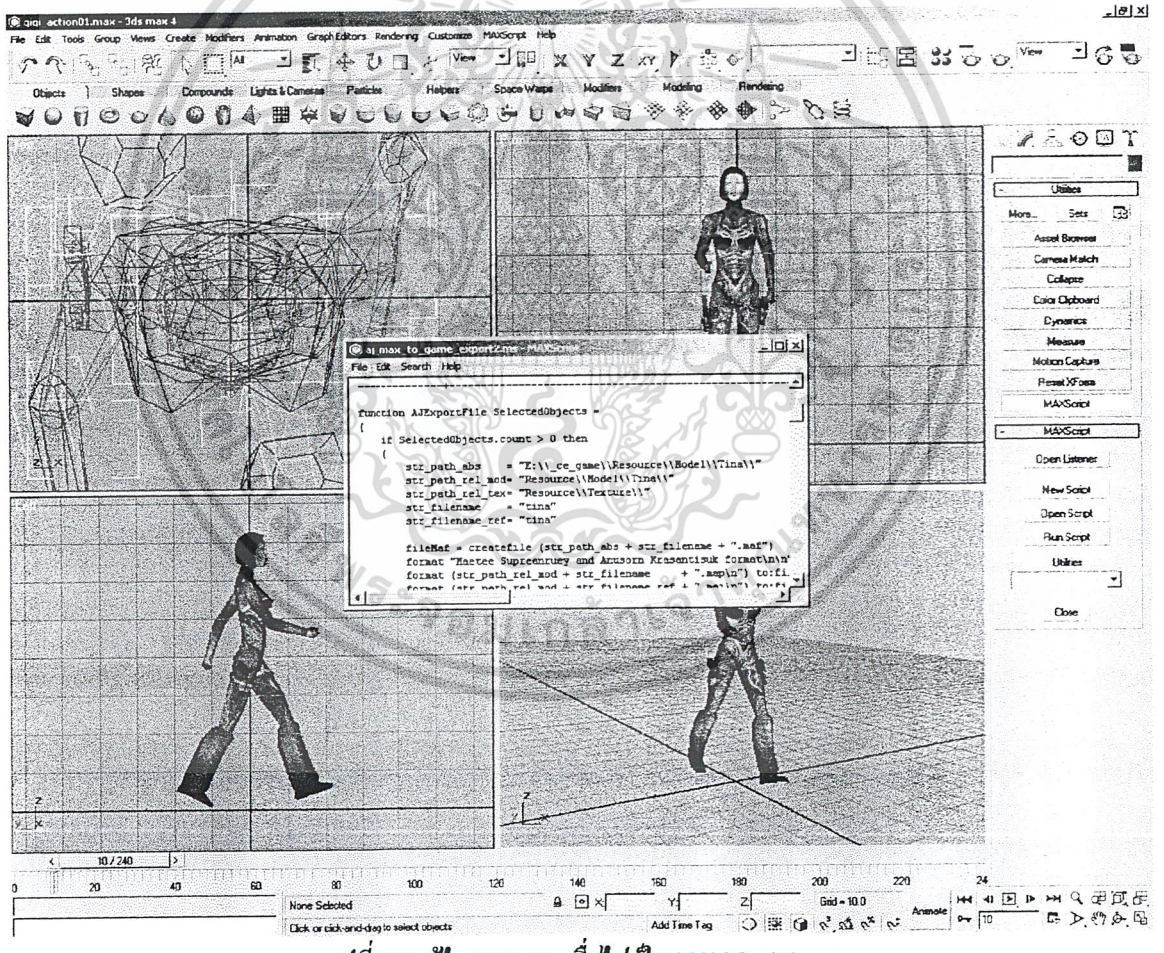
ทำการแก้ไข Path และชื่อไฟล์ที่จะทำการเอ็กพอร์ต ใน MAX Script

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

str_path คือ Path ของไฟล์ที่จะทำการเอ็กพอร์ต

str_filename คือชื่อของไฟล์ที่จะทำการเอ็กพอร์ต

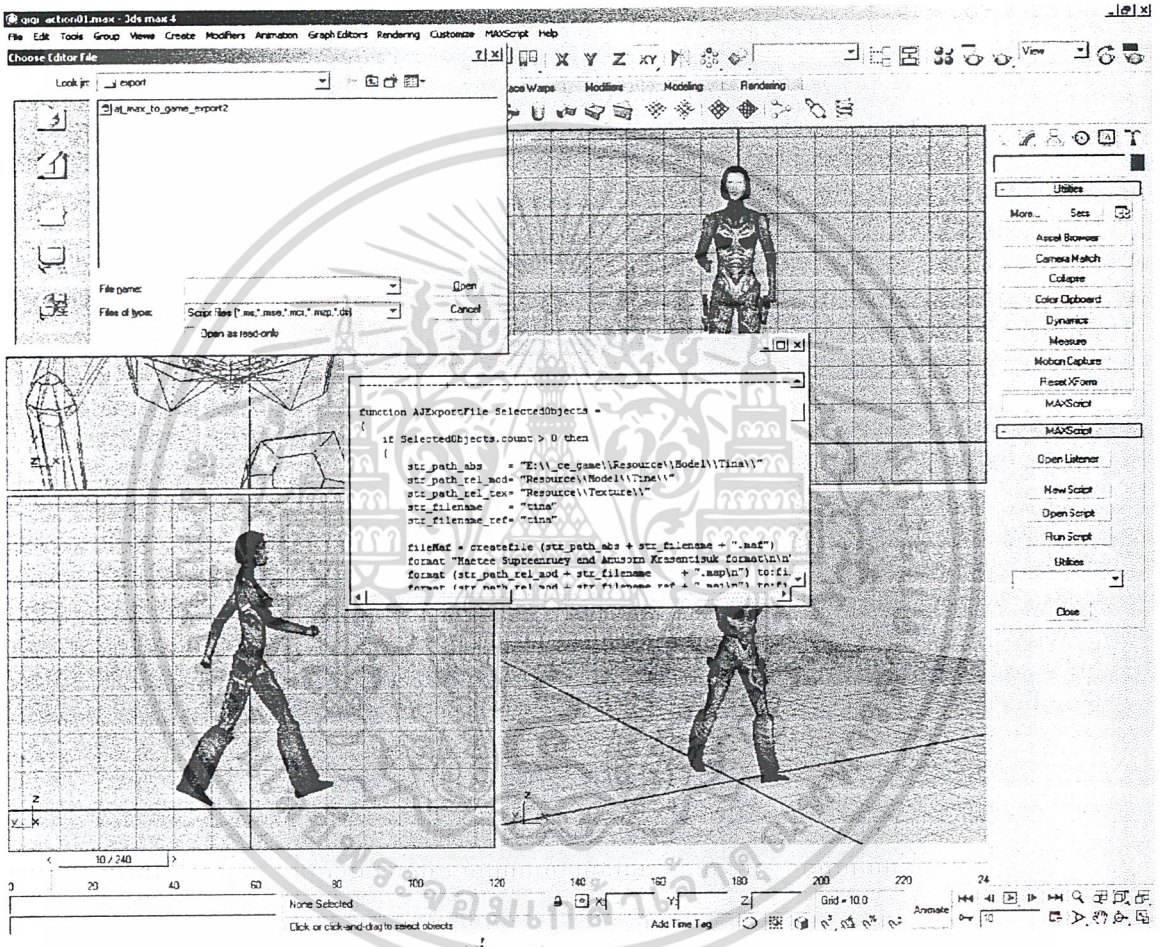
จากนั้นทำการบันทึก MAX Script ไฟล์



รูปที่ 5.5 แก้ไข Path และชื่อไฟล์ใน MAX Script

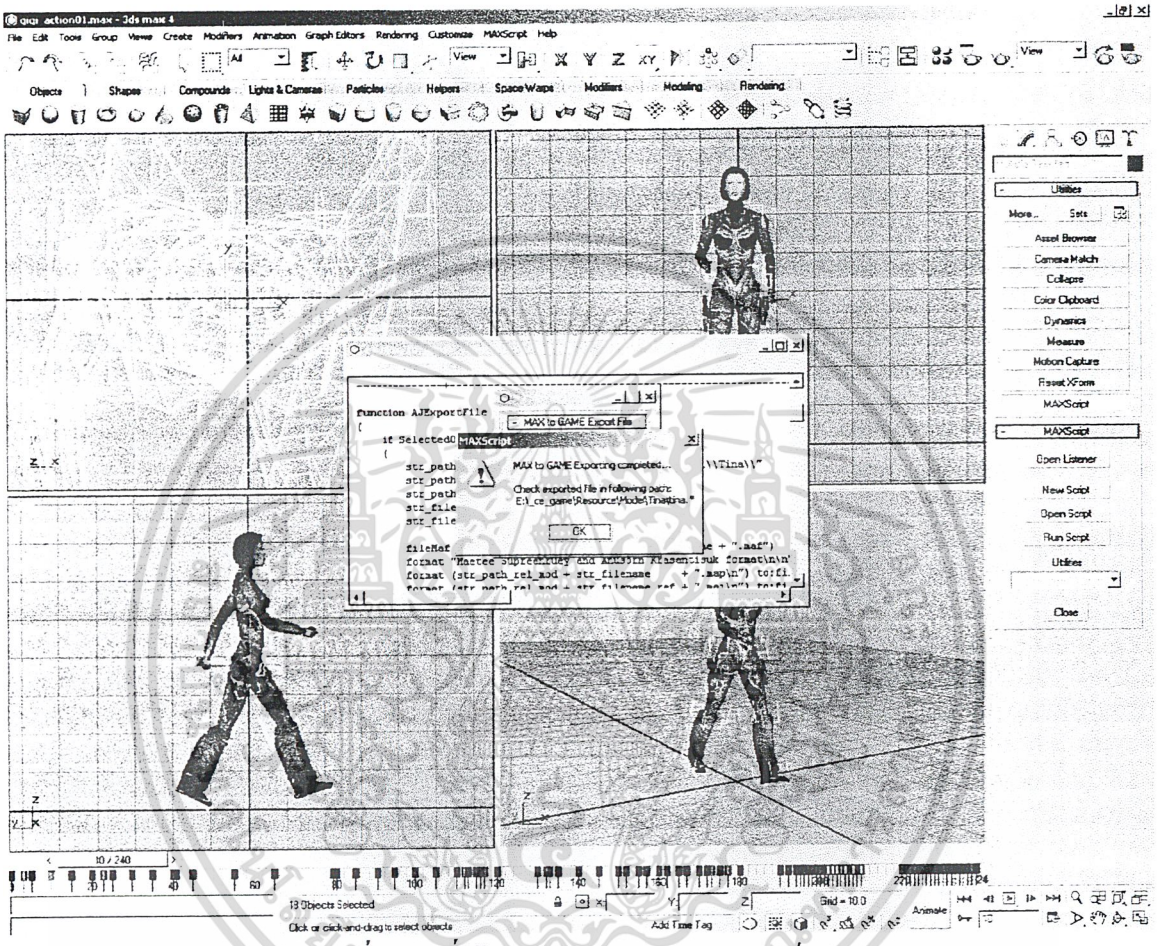
5.1.1.3 รันสคริปต์ (Run script) ที่ใช้ในการเอ็กพอร์ต

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



รูปที่ 5.6 รัน MAX Script

เลือก Export scene จากนั้นจะได้ Dialog box แสดงผลการเอ็กพอร์ตครั้งนั้น



รูปที่ 5.7 ผลที่ได้จากการ Export Scene สมบูรณ์

จากการรัน Max Script แล้วจะมีไฟล์ที่เกิดขึ้นใหม่ 5 ไฟล์ด้วยกัน ซึ่งเป็นไฟล์ที่มีชื่อที่ได้แก้ไขแล้วใน MAX Script ไฟล์

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

5.2 รูปแบบของไฟล์ข้อมูลที่เอ็กพอร์ต

- Filename.MAS

  สำหรับบอกวัตถุที่เป็นแอนิเมชัน บอกว่ามีแอคชั่นใดบ้าง ชื่อของแอคชั่นนั้น และส่วนที่อ้างอิงไปยังไฟล์ .MAA

- Filename.MAA

  บอกว่าแอนิเมชันในแอคชั่นนั้นมีกี่เฟรม โดยแต่ละช่วง บอกช่วงของความเร็ว และส่วนที่อ้างอิงไปยังไฟล์ .MAF

- Filename.MAF

  เป็นไฟล์ที่บอกจุดอ้างอิงไปยังไฟล์อื่นที่เกี่ยวข้อง ที่ใช้เก็บแอนิเมชันเฟรมของรูป

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

- Filename.MAP

  เซตของจุดที่ประกอบเป็นเวอร์เท็กซ์ บอกจำนวนของเวอร์เท็กซ์, ตำแหน่งของเวอร์เท็กซ์ ที่แสดงรูปร่างของวัตถุเฟรมนั้น

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

- Filename.MAI

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

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

- Filename.MAU

  เป็นเท็กเจอร์โคออดิเนท เพื่อใช้แมพรูปภาพไปยังพื้นผิวของวัตถุ รายละเอียดในไฟล์ในบรรทัดแรกจะบอกจำนวนของเท็กเจอร์โคออดิเนท ในบรรทัดต่อไปจะบอกเท็กเจอร์โคออดิเนทของ U และ V

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

  ชื่อไฟล์รูปที่ใช้เป็น Texture

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

- Filename.MAC

  เซตของสีของวัตถุ (ปกติถ้ามีการใช้เท็กเจอร์เราจะไม่ใช้ไฟล์นี้)

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

```
close fileMaf

fileMap = createfile "C:\export\model\man_010.map"
fileMai = createfile "C:\export\model\man_shr.mai"
fileMau = createfile "C:\export\model\man_shr.mau"
fileMac = createfile "C:\export\model\man_shr.mac"
bMesh = Obj.mesh

-- write file header
format "Anusorn Krasantisuk and Ekapol Anantapornkich format\n\n" to:fileMap
format "Anusorn Krasantisuk and Ekapol Anantapornkich format\n\n" to:fileMai
format "Anusorn Krasantisuk and Ekapol Anantapornkich format\n\n" to:fileMau
format "Anusorn Krasantisuk and Ekapol Anantapornkich format\n\n" to:fileMac

format "%\n" bMesh.numverts to:fileMap
for i = 1 to bMesh.numverts do
(
        verts = getvert bMesh i
        x = ( Obj.transform[1].x * verts.x ) + ( Obj.transform[2].x * verts.y ) + (
Obj.transform[3].x * verts.z ) + Obj.transform[4].x
        y = ( Obj.transform[1].y * verts.x ) + ( Obj.transform[2].y * verts.y ) + (
Obj.transform[3].y * verts.z ) + Obj.transform[4].y
        z = ( Obj.transform[1].z * verts.x ) + ( Obj.transform[2].z * verts.y ) + (
Obj.transform[3].z * verts.z ) + Obj.transform[4].z
        format "% % %\n"  x z y to:fileMap
)
format "%\n" bMesh.numfaces to:fileMai
for x = 1 to bMesh.numfaces do
(
        face = getface bMesh x
        p0 = face.x as integer
        p2 = face.y as integer
```

```
        p1 = face.z as integer
        format "% % %\n" (p0-1) (p1-1) (p2-1) to:fileMai
)

format "%\n" (bMesh.numfaces*3) to:fileMau
for abc = 1 to bMesh.numfaces do
(
        test = getTVface bMesh abc

        p0 = test.x  as integer
        p2 = test.y  as integer
        p1 = test.z  as integer

        face0 = gettvert bMesh p0
        format "% %\n" face0.x face0.y  to:fileMau

        face1 = gettvert bMesh p1
        format "% %\n" face1.x face1.y  to:fileMau

        face2 = gettvert bMesh p2
        format "% %\n" face2.x face2.y  to:fileMau
)

nvc = getNumCpvVerts bMesh

format "%\n" (nvc) to:fileMac
for abc = 1 to nvc do
(
        cv = getvertcolor bMesh abc
        format "%\n" cv to:fileMac
)

close fileMap
```

```
close fileMai
close fileMau
close fileMac
```

สำหรับรายละเอียดในการใช้งาน Max Script สามารถอ้างอิงได้จาก Max Script ซึ่งมาพร้อมกับ 3D Studio Max

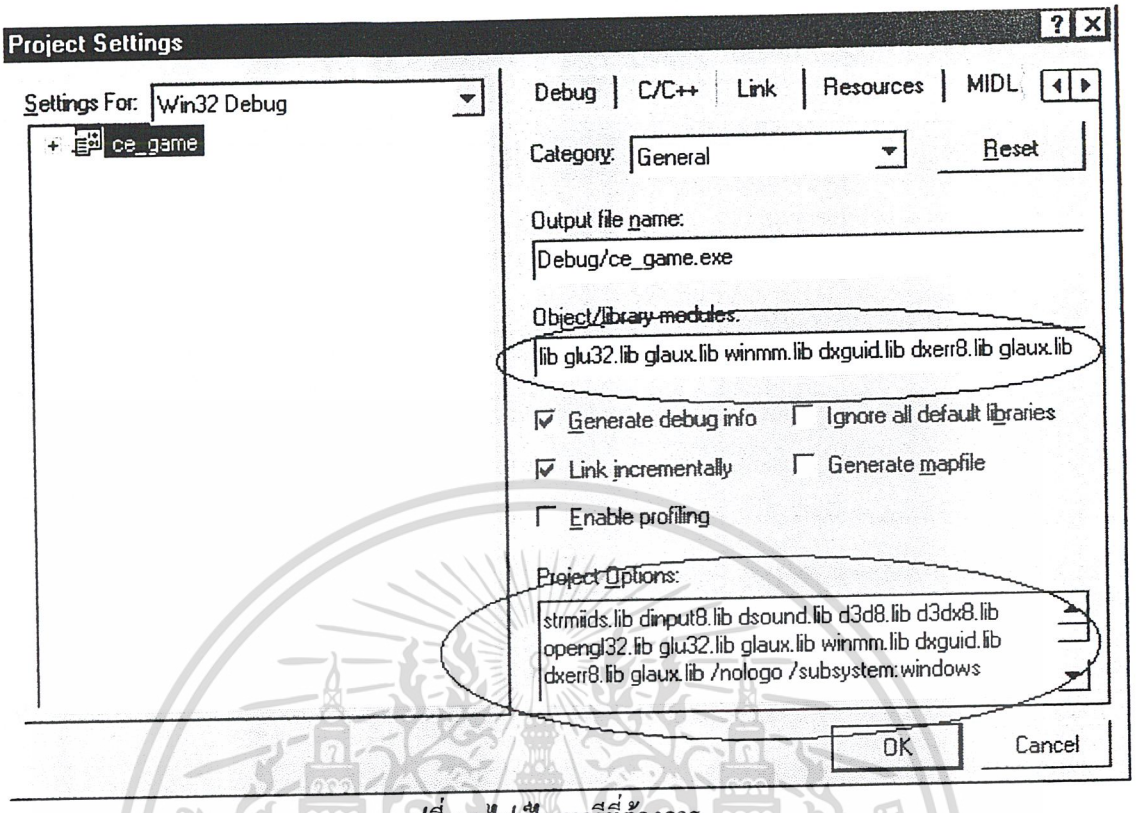
เราสามารถควบคุมการเคลื่อนไหวโดยการสร้างไฟล์ดังนี้

```
3 // จำนวนเฟรมทั้งหมด
Frame\\ man_001.maf  Frame\\ man_002.maf   1.0 // เฟรมที่ 1 - 2
Frame\\ man_002.maf  Frame\\ man_003.maf   0.5 // เฟรมที่ 2 – 3
Frame\\ man_003.maf  Frame\\ man_001.maf   0.5 // เฟรมที่ 3 – 1 (วนรอบ)
```

ตัวเลขด้านหลังจะแสดงความห่างของระยะเวลาระหว่างสองเฟรมต่อกัน

5.3 การใช้เกมเอนจินต์เฟรมเวิร์ค

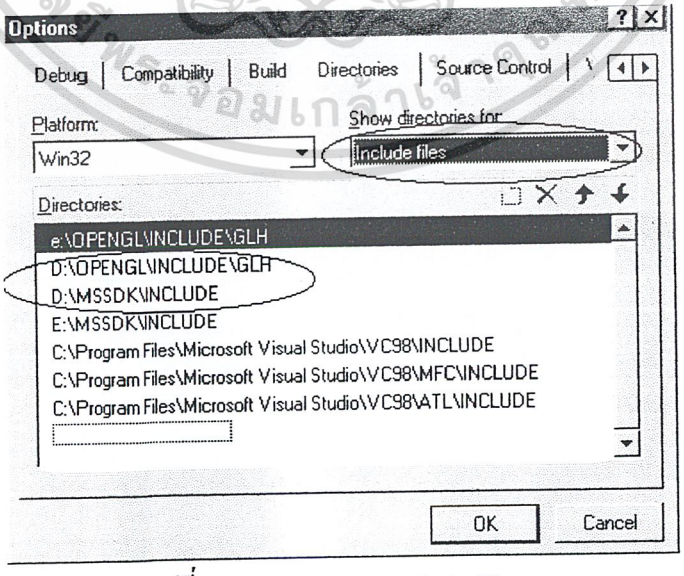
หลังจากที่ได้เตรียมข้อมูลต่างๆ สำหรับการสร้างเกมแล้ว ขั้นตอนต่อไปคือการสร้างเกม ในการสร้างเกมจะใช้เฟรมเวิร์ค ซึ่งประกอบไปด้วยคลาสต่างๆ ที่เตรียมไว้ให้ผู้ใช้สามารถสืบทอด หรือเรียกใช้ฟังก์ชันได้ ในการใช้เฟรมเวิร์คสร้างเกมนั้น มีขั้นตอนดังนี้



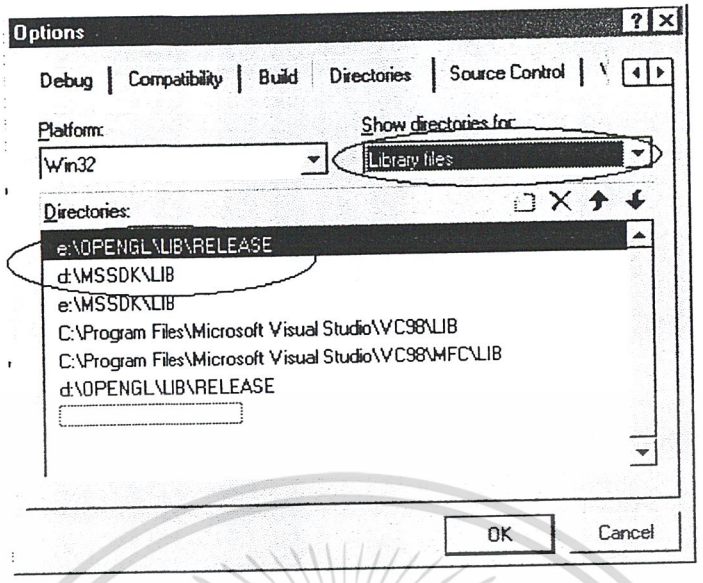
รูปที่ 5.8 ไฟล์ไลบรารีที่ต้องการ

เปิดเฟรมเวิร์คด้วย VC++ ตั้งค่าต่างๆดังนี้

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

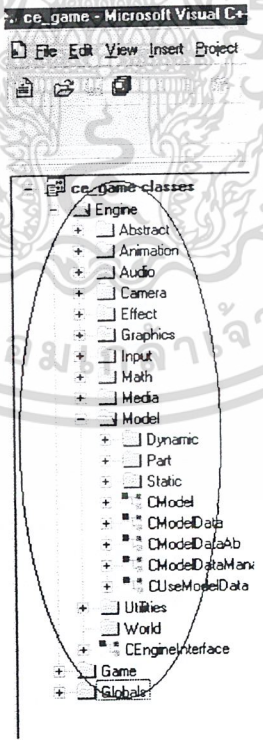


รูปที่ 5.9 Directory ของ Include Files



รูปที่ 5.10 Directory ของ Include Files

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



รูปที่ 5.11 แสดง Workspace ของเอนจินต์

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

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

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

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

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

รูปที่ 5.12 แสดงส่วนสำคัญของคลาส CMyGlApp

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

เขียนฟังก์ชัน InitDeviceObjects ขึ้นใหม่ ส่วนนี้เป็นส่วนของการโหลดข้อมูลต่างๆ ที่เตรียมไว้เข้ามาไว้ใช้ในเกม และการตั้งค่าเริ่มต้นของเกม เช่น ตำแหน่งเริ่มของกล้อง ตำแหน่งเริ่มของตัวละคร

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

    ShowLoading(0.0f, "Generate camera");

    static BOOL  bFirstTime = TRUE;

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

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

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

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

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

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

รูปที่ 5.13 แสดงตัวอย่าง Source Code ในส่วนโหลดข้อมูล

เขียนฟังก์ชัน Render ขึ้นใหม่ ส่วนนี้จะเป็นส่วนแสดงผลของวัตถุต่างๆ ออกทางจอภาพ ต้องการให้วัตถุชิ้นไหนแสดงผล ก็ให้ใช้ฟังก์ชัน Render ของวัตถุนั้นในฟังก์ชันนี้

```
CEffectBillboardList CloudList;

HRESULT CMyGlApp::Render(GLvoid)
{
    m_Camera.View();

    // Clear Scene
    ModelBox.Render();
    GLGfx::Clear(GL_COLOR_BUFFER_BIT | GL

    ModelSky.Render();
    glEnable(GL_FOG);

    ModelLand.Render();
    //-------------------------------------
    // ModelLand Render
    //glColorMask(TRUE,TRUE,TRUE,TRUE);

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

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

    arProjectileObject.Render();
```

รูปที่ 5.14 แสดงตัวอย่าง Source Code ส่วน Render

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

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

        case LOADING
        {
        }
        break;

        case GAMELOOP
        {
    if( m_arKey[ 5 ] )
    {
        CMyGlApp::s_oSceneFog.m_fEnd   -= fElapsedTime*100.0f;
    }
    else
    if( m_arKey[ 6 ] )
    {
        CMyGlApp::s_oSceneFog.m_fEnd   -= fElapsedTime*100.0f;
    }
    else
    if( m_arKey[ 7 ] )
    {
        CMyGlApp::s_oSceneFog.m_fStart  -= fElapsedTime*100.0f;
    }
    else
    if( m_arKey[ 8 ] )
    {
        CMyGlApp::s_oSceneFog.m_fStart  -= fElapsedTime*100.0f;
    }
    else
    {
        // Process other key control
        RECT rcScreen;
        GetClientRect( m_hWnd, &rcScreen );
//      m_oCamera.ProcessKey( fElapsedTime, rcScreen, m_arKey[0],
```

รูปที่ 5.15 แสดงตัวอย่าง Source Code ในส่วนการกดปุ่ม

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


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

/*
    // Activate to engine interface
    CEngineInterface::AddLogicalTime( m_fElapsedTime
*/
    // Insert key processing
    ProcessKey( m_fElapsedTime );
    CScene::FrameMove( m_fElapsedTime );

    m_oCamera.LayOn( &g_oLand001 );

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

    arProjectileObject.Process( m_fElapsedTime );
```

รูปที่ 5.16 แสดงตัวอย่าง Source Code ในส่วนประมวลผล

บทที่ 6
การพัฒนาเกมเพื่อทดสอบเกมเอนจินต์

เมื่อได้พัฒนา API ต่างๆแล้ว เราก็ควรที่จะนำมาทดสอบในการเขียนโปรแกรมจริงๆ โดยเขียนโปรแกรมเกม 3 มิติแล้วนำ API ไปใช้ในการพัฒนาเพื่อให้เกมที่พัฒนาสมบูรณ์มากยิ่งขึ้น โดยได้จากที่พัฒนา Media API, Input API และ Auido API version ล่าสุด

6.1 ประเภทของเกมที่พัฒนา

เกมล่าหมาป่าอสูร

เป็นเกม 3 มิติ โดยเป็นเกมลักษณะ Real-time Stratagy โดยผู้เล่นต้องควบคุมทหาร 3 นายปราบหมาป่าอสูร โดยผู้เล่นจะต้องควบคุมทหารได้สลับทีละนายให้ยิงปืนกลยิงใส่หมาป่า โดยหมาป่าจะเข้ามาโจมตีทหารด้วย โดยหากผู้เล่นสามารถปราบหมาป่าได้ก็ถือว่าชนะ แต่หากไม่สำเร็จก็ถือว่าแพ้

เนื้อเรื่อง

ในปี 2010 ได้มีการคิดค้นวิจัยสัตว์ทดลองพันธ์ใหม่ ซึ่งมีความแข่งแกร่งกว่าพันธ์ดั้งเดิมและเชื่อฟังมนุษย์ เพื่อนำไปใช้ในการทหาร แต่แล้วจากอุบัติเหตุ ทำให้สัตว์ทดลองตนหนึ่ง ได้หลุดออกมาทำร้ายผู้คน ดังนั้นกองกำลังพิเศษจึงได้ส่งหน่วยรบออกไล่ล่าเพื่อรักษาความลับ

จนในที่สุดสัตว์ทดลองซึ่งมีรูปร่างเป็นหมาป่า ได้มาจนมุมที่ดาดฟ้าตึกแห่งหนึ่ง กองกำลังพิเศษจึงต้องปราบหมาป่าอสูรตนนี้ให้ได้เพื่อความสงบสุขของเมือง

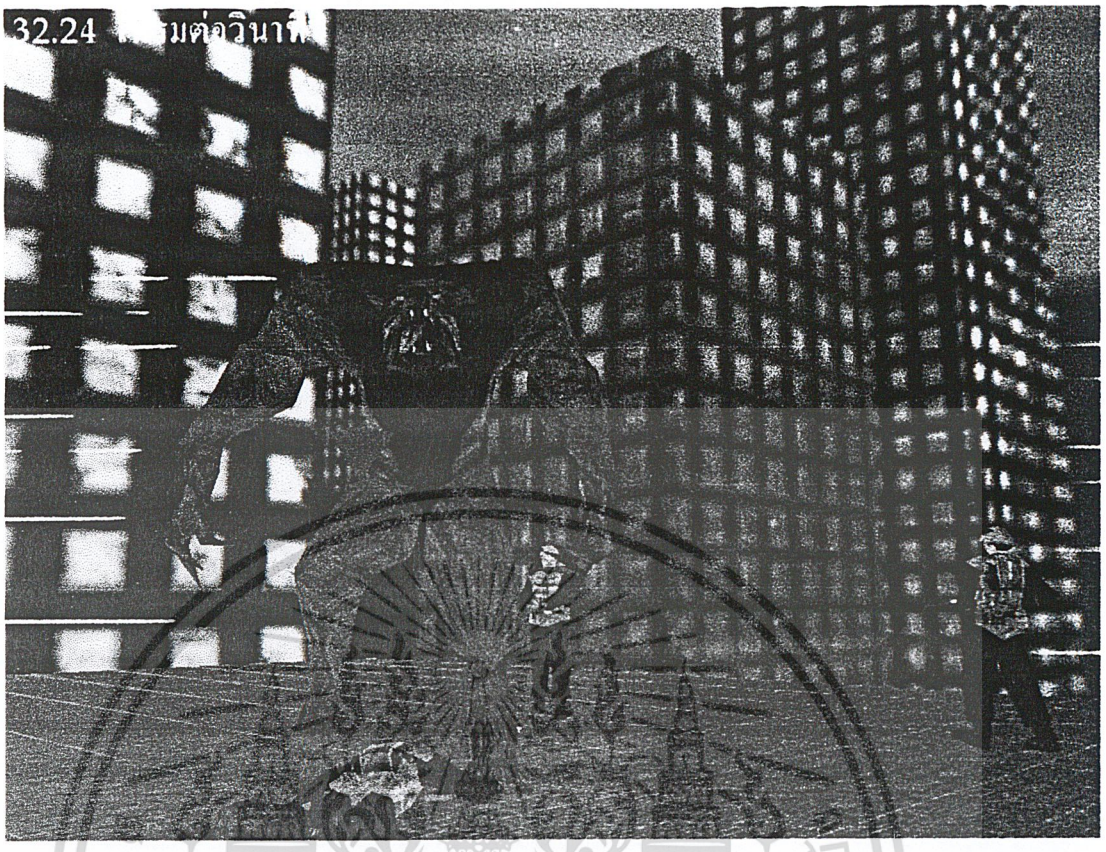
6.2 การนำเอนจินต์มาพัฒนาเกม

สิ่งที่ต้องการในการพัฒนาเกมคือ

- Video ต้นเกม ซึ่งต้องใช้ Media API ที่พัฒนา
- เสียง Effect ต่างๆ ใน 3 มิติการและเล่นเพลงประกอบ ต้องใช้ Audio API
- การควบคุมตัวละครและกล้อง, การส่ง Force-Feedback ไปยัง Joypad เมื่อมีการยิงปืนของตัวละครจะต้องใช้ Input API

6.3 ผลการทดสอบ

ผลการทดสอบได้ผลที่น่าพอใจ เกมสามารถเล่นได้อย่างสนุกสนาน โดยไม่มีปัญหาเกี่ยวกับประสิทธิภาพ อัตราเฟรมต่อวินาทีมากพอจนไม่เห็นความสะดุด และสามารถใช้งานได้ตามที่ได้ระบุไว้ แม้ว่าเกมล่าหมาป่าอสูรจะใช้ครบทุกฟังชั่น



รูปที่ 6.1 ภาพจากเกมล่าหมาป่าอสูร

บทที่ 7
วิจารณ์และสรุป

7.1 สรุปผลการวิจัย

จากการศึกษา DirectX เพื่อนำมาใช้ในการพัฒนาเกมนั้น เกมทุกประเภทจะต้องการใช้ส่วน API พื้นฐานในการสร้างเกม ดังนั้นเมื่อศึกษา DirectX ในส่วน DirectAudio, DirectInput และ DirectShow ทำให้สามารถนำ DirectX API มาพัฒนาต่อจากเดิม โดยลดขั้นตอนที่ยุ่งยาก ซับซ้อน มาเป็นง่ายๆ ซึ่งเหมาะสมในการพัฒนา โดยได้ทำการออกแบบ Architecture ของ API ต่างๆ และพัฒนาออกมา ซึ่งสามารถนำไปใช้ในการเขียนโปรแกรมได้เป็นอย่างดี

7.2 แนวทางในการพัฒนาต่อ

- ปรับปรุงให้มีการจัดการเฟรมเวิร์คแบบอัตโนมัติ
- พัฒนาให้มีประสิทธิภาพให้สูงขึ้น
- ปรับปรุงส่วนออดิโอให้มีความสามารถอีเอเอ็กซ์ (EAX)

ภาคผนวก ก.

พารามิเตอร์ของ Effect

DSFXChorus

Syntax

```
typedef struct _DSFXChorus {
  FLOAT   fWetDryMix;
  FLOAT   fDepth;
  FLOAT   fFeedback;
  FLOAT   fFrequency;
  LONG    lWaveform;
  FLOAT   fDelay;
  LONG    lPhase;
} DSFXChorus, *LPDSFXChorus;

typedef const DSFXChorus *LPCDSFXChorus;
```

fWetDryMix

เป็นอัตราของสัญญาณที่ประมวลผมแล้วไปยังสัญญาณที่ยังไม่ได้ประมวลผล ต้องมีค่าอยู่ระหว่าง DSFXCHORUS_WETDRYMIX_MIN กับ DSFXCHORUS_WETDRYMIX_MAX โดยค่า Default จะอยู่ที่ 50

fDepth

เปอร์เซ็นต์ของการหน่วงเวลาที่ถูก Modulate โดย Oscillator ความถี่ต่ำ ต้องอยู่ในช่วงระหว่าง DSFXCHORUS_DEPTH_MIN กับ DSFXCHORUS_DEPTH_MAX โดยค่า Default อยู่ที่ 10

fFeedback

เปอร์เซ็นต์ของสัญญาณออกไป Feedback สู่สัญญาณขาเข้า โดยจะอยู่ในช่วงระหว่าง DSFXCHORUS_FEEDBACK_MIN กับ DSFXCHORUS_FEEDBACK_MAX โดยค่า Default คือ 25

fFrequency

ความถี่ของ LFO ในช่วงของ DSFXCHORUS_FREQUENCY_MIN และ DSFXCHORUS_FREQUENCY_MAX โดยค่า Default อยู่ที่ 1.1

lWaveform

รูปแบบคลื่นของ LFO มีอยู่สองรูปแบบคือ DSFXCHORUS_WAVE_TRIANGLE และ DSFXCHORUS_WAVE_SIN ค่า Default จะอยู่ในรูป sine

fDelay

ขนาดเวลาในหน่วย Milliseconds ของสัญญาณขาเข้าที่ถูกหน่วงเวลาก่อนที่จะถูกเล่น อยู่ในช่วงของ DSFXCHORUS_DELAY_MIN ถึง DSFXCHORUS_DELAY_MAX โดยค่า deault อยู่ที่ 16 ms

lPhase

ความแตกต่างของเฟสระหว่างซ้ายกับขวา มีค่าระหว่าง DSFXCHORUS_PHASE_MIN ถึง DSFXCHORUS_PHASE_MAX โดยค่าที่เป็นไปได้อยู่ที่

Value

DSFXCHORUS_PHASE_NEG_180

DSFXCHORUS_PHASE_NEG_90

DSFXCHORUS_PHASE_ZERO

DSFXCHORUS_PHASE_90

DSFXCHORUS_PHASE_180

ค่า Default อยู่ที่ DSFXCHORUS_PHASE_90

DSFXCompressor

Syntax

```
typedef struct _DSFXCompressor {
  FLOAT  fGain;
  FLOAT  fAttack;
  FLOAT  fRelease;
  FLOAT  fThreshold;
  FLOAT  fRatio;
  FLOAT  fPredelay;
} DSFXCompressor, *LPDSFXCompressor;

typedef const DSFXCompressor *LPCDSFXCompressor;
```

fGain

Gain ของสัญญาณขาออกหลังจากบีบอัด มีค่าในช่วง DSFXCOMPRESSOR_GAIN_MIN ถึง DSFXCOMPRESSOR_GAIN_MAX ค่า Default อยู่ที่ 0dB

fAttack

เวลาหลังจากการช่วงเวลาการบีบอัดถึงพิกัด มีค่าในช่วง DSFXCOMPRESSOR_ATTACK_MIN ถึง DSFXCOMPRESSOR_ATTACK_MAX ค่า Default อยู่ที่ 10 ms

fRelease

ความเร็วของการบีบอัดที่ถูกหยุดหลังจากสัญญาณขาเข้าตกต่ำกว่า fThreshold มีค่าอยู่ในช่วง DSFXCOMPRESSOR_RELEASE_MIN ถึง DSFXCOMPRESSOR_RELEASE_MAX ค่า Default อยู่ที่ 200 ms

fThreshold

จุดที่การบีบอัดเริ่มต้นในหน่วยเดซิเบล อยู่ในช่วง DSFXCOMPRESSOR_THRESHOLD_MIN ถึง DSFXCOMPRESSOR_THRESHOLD_MAX ค่า Default อยู่ที่ –20dB

fRatio

อัตราการบีบอัด อยู่ในช่วง DSFXCOMPRESSOR_RATIO_MIN ถึง DSFXCOMPRESSOR_RATIO_MAX ค่า Default อยู่ที่ 3 หมายความว่าอัตราอยู่ที่ 3 ต่อ 1

fPredelay

เวลาหลังจากขอบเขต lThreshold ก่อนเฟส Attack เริ่มต้น ในหน่วย Milliseconds อยู่ในช่วง DSFXCOMPRESSOR_PREDELAY_MIN ถึง DSFXCOMPRESSOR_PREDELAY_MAX ค่า Default อยู่ที่ 4 ms

DSFXDistortion

```
typedef struct _DSFXDistortion {
  FLOAT  fGain;
  FLOAT  fEdge;
  FLOAT  fPostEQCenterFrequency;
  FLOAT  fPostEQBandwidth;
  FLOAT  fPreLowpassCutoff;
} DSFXDistortion, *LPDSFXDistortion;

typedef const DSFXDistortion *LPCDSFXDistortion;
```

fGain

จำนวนของสัญญาณที่เปลี่ยนแปลงหลังจาก Distort อยู่ในช่วง DSFXDISTORTION_GAIN_MIN ถึง DSFXDISTORTION_GAIN_MAX ค่า Default อยู่ที่ –18 dB

fEdge

เปอร์เซ็นต์ของความแรงของการ Distort อยู่ในช่วง DSFXDISTORTION_EDGE_MIN ถึง DSFXDISTORTION_EDGE_MAX ค่า Default คือ 15 เปอร์เซ็นต์

fPostEQCenterFrequency

ความถี่กลางของการเพิ่มช่วง Harmonic อยู่ในช่วง

DSFXDISTORTION_POSTEQCENTERFREQUENCY_MIN ถึง

DSFXDISTORTION_POSTEQCENTERFREQUENCY_MAX โดยค่า Default อยู่ที่ 2400 Hz

fPostEQBandwidth

ความกว้างของความถี่ที่อยู่ในช่วงการเพิ่มของ Harmonic

DSFXDISTORTION_POSTEQBANDWIDTH_MIN ถึง

DSFXDISTORTION_POSTEQBANDWIDTH_MAX ค่า Default อยู่ที่ 2400 Hz

fPreLowpassCutoff

ฟิลเตอร์เพื่อการตัด Harmonic คลื่นของถี่สูง อยู่ในช่วง

DSFXDISTORTION_PRELOWPASSCUTOFF_MIN ถึง

DSFXDISTORTION_PRELOWPASSCUTOFF_MAX ค่า Default อยู่ที่ 8000 Hz

DSFXEcho

```
typedef struct _DSFXEcho {
  FLOAT  fWetDryMix;
  FLOAT  fFeedback;
  FLOAT  fLeftDelay;
  FLOAT  fRightDelay;
  LONG   lPanDelay;
} DSFXEcho, *LPDSFXEcho;

typedef const DSFXEcho *LPCDSFXEcho;
```

fWetDryMix

อัตราของสัญญาณที่ประมวลผลแล้วกับสัญญาณดั้งเดิม อยู่ในช่วง

DSFXECHO_WETDRYMIX_MIN ถึง DSFXECHO_WETDRYMIX_MAX โดยค่า Default อยู่ที่ 50

fFeedback

เปอร์เซ็นต์ของสัญญาณขาออก Feedback กลับไปยังขาเข้า มีค่าอยู่ในช่วง

DSFXECHO_FEEDBACK_MIN ถึง DSFXECHO_FEEDBACK_MAX ค่า Default อยู่ที่ 50

fLeftDelay

ค่าการหน่วงเวลาของช่องด้านซ้าย ในหน่วย Milliseconds อยู่ในช่วง

DSFXECHO_LEFTDELAY_MIN ถึง DSFXECHO_LEFTDELAY_MAX ค่า Default อยู่ที่ 500 ms

fRightDelay

ค่าการหน่วงเวลาของช่องด้านขวา ในหน่วย Milliseconds อยู่ในช่วง DSFXECHO_RIGHTDELAY_MIN ถึง DSFXECHO_RIGHTDELAY_MAX ค่า Default อยู่ที่ 500 ms

lPanDelay

ค่าที่ตั้งไว้เพื่อใช้ที่การหน่วงเวลาในตอนสลับระหว่างซ้ายกับขวา ค่า Default อยู่ที่ 0 หมายถึงไม่มีการสลับช่อง โดยค่าจะมีอยู่สองค่าคือ DSFXECHO_PANDELAY_MIN (เท่ากับ FALSE) และ DSFXECHO_PANDELAY_MAX (เท่ากับ TRUE).

DSFXFlanger

```
typedef struct _DSFXFlanger {
  FLOAT  fWetDryMix;
  FLOAT  fDepth;
  FLOAT  fFeedback;
  FLOAT  fFrequency;
  LONG   lWaveform;
  FLOAT  fDelay;
  LONG   lPhase;
} DSFXFlanger, *LPDSFXFlanger;

typedef const DSFXFlanger *LPCDSFXFlanger;
```

fWetDryMix

อัตราของสัญญาณที่ประมวลผลแล้วกับสัญญาณที่ไม่ได้ประมวลผล ต้องอยู่ในช่วง DSFXFLANGER_WETDRYMIX_MIN ถึง DSFXFLANGER_WETDRYMIX_MAX ค่า Default อยู่ที่ 50

fDepth

เปอร์เซ็นของเวลาที่หน่วงที่ถูก Modulate โดย Oscillator ความถี่ต่ำ (low-frequency oscillator หรือ LFO) อยู่ในช่วง DSFXFLANGER_DEPTH_MIN ถึง DSFXFLANGER_DEPTH_MAX ค่า Default คือ 100

fFeedback

เปอร์เซ็นต์ของสัญญาณขาออกที่ Feedback กลับไปยัง Effect ขาเข้า มีค่าในช่วง DSFXFLANGER_FEEDBACK_MIN ถึง DSFXFLANGER_FEEDBACK_MAX ค่า Default อยู่ที่ 50

fFrequency

ความถี่ของ LFO ในช่วงของ DSFXFLANGER_FREQUENCY_MIN ถึง DSFXFLANGER_FREQUENCY_MAX ค่า Default อยู่ที่ 0.25

lWaveform

รูปแบบคลื่นของ LFO โดยค่า Default จะอยู่ในรูป Sine โดยค่าที่เป็นไปได้มีดังนี้

Value	Description
DSFXFLANGER_WAVE_TRIANGLE	Triangle.
DSFXFLANGER_WAVE_SIN	Sine.

fDelay

เวลาในหน่วย Milliseconds ของขาเข้าที่ถูกหน่วงเวลาก่อนที่จะถูกเล่นกลับ อยู่ในช่วงของ DSFXFLANGER_DELAY_MIN ถึง DSFXFLANGER_DELAY_MAX ค่า Default อยู่ที่ 2 ms

lPhase

ความแตกต่างระหว่งเฟส LFO ซ้ายและขวา อยู่ในช่วง DSFXFLANGER_PHASE_MIN ถึง DSFXFLANGER_PHASE_MAX โดยค่าที่เป็นไปได้อยู่ที่

Value
DSFXFLANGER_PHASE_NEG_180
DSFXFLANGER_PHASE_NEG_90
DSFXFLANGER_PHASE_ZERO
DSFXFLANGER_PHASE_90
DSFXFLANGER_PHASE_180

DSFXGargle

```
typedef struct _DSFXGargle {
  DWORD  dwRateHz;
  DWORD  dwWaveShape;
} DSFXGargle, *LPDSFXGargle;

typedef const DSFXGargle *LPCDSFXGargle;
```

dwRateHz

อัตราของการ Modulate ในหน่วย Hertz อยู่ในช่วง DSFXGARGLE_RATEHZ_MIN ถึง DSFXGARGLE_RATEHZ_MAX โดยค่า Default ที่ 20

dwWaveShape

รูปร่างของคลื่นที่ Modulate โดยค่าที่เป็นไปอยู่ที่

Value	Description
DSFXGARGLE_WAVE_TRIANGLE	Triangular wave.
DSFXGARGLE_WAVE_SQUARE	Square wave.

ค่า Default อยู่ที่ DSFXGARGLE_WAVE_TRIANGLE

DSFXParamEq

```
typedef struct _DSFXParamEq {
  FLOAT  fCenter;
  FLOAT  fBandwidth;
  FLOAT  fGain;
} DSFXParamEq, *LPDSFXParamEq;
```

fCenter

ความถี่กลาง ที่อยู่ช่วง DSFXPARAMEQ_CENTER_MIN ถึง DSFXPARAMEQ_CENTER_MAX โดยค่า Default อยู่ที่ 8000

fBandwidth

ค่า Bandwidth มีหน่วย Semitones อยู่ในช่วง DSFXPARAMEQ_BANDWIDTH_MIN ถึง DSFXPARAMEQ_BANDWIDTH_MAX ค่า Default อยู่ที่ 12

fGain

Gain ของสัญญาณ อยู่ในช่วง DSFXPARAMEQ_GAIN_MIN ถึง DSFXPARAMEQ_GAIN_MAX ค่า Default อยู่ที่ 0

DSFXWavesReverb

```
typedef struct _DSFXWavesReverb {
  FLOAT  fInGain;
  FLOAT  fReverbMix;
  FLOAT  fReverbTime;
  FLOAT  fHighFreqRTRatio;
} DSFXWavesReverb, *LPDSFXWavesReverb;

typedef const DSFXWavesReverb *LPCDSFXWavesReverb;
```

fInGain

Gain ของ สัญญาณขาเข้า ในหน่วยเดซิเบล (dB) อยู่ในช่วง DSFX_WAVESREVERB_INGAIN_MIN ถึง DSFX_WAVESREVERB_INGAIN_MAX ค่า Deault คือค่า DSFX_WAVESREVERB_INGAIN_DEFAULT หรือ 0 dB

fReverbMix

ค่า Reverb Mix ในหน่วย dB อยู่ในช่วง DSFX_WAVESREVERB_REVERBMIX_MIN ถึง DSFX_WAVESREVERB_REVERBMIX_MAX ค่า Default คือค่า DSFX_WAVESREVERB_REVERBMIX_DEFAULT หรือ 0 dB

fReverbTime

ค่าเวลา Reverb ในหน่วย Milliseconds อยู่ในช่วง DSFX_WAVESREVERB_REVERBTIME_MIN ถึง DSFX_WAVESREVERB_REVERBTIME_MAX โดยค่า Default คือค่า DSFX_WAVESREVERB_REVERBTIME_DEFAULT หรือ 1000.

fHighFreqRTRatio

อยู่ในช่วง DSFX_WAVESREVERB_HIGHFREQRTRATIO_MIN ถึง DSFX_WAVESREVERB_HIGHFREQRTRATIO_MAX โดยค่า Default อยู่ที่ DSFX_WAVESREVERB_HIGHFREQRTRATIO_DEFAULT, หรือ 0.001

หมายเหตุ ค่า Constant ต่างๆ สามารถดูได้ใน Dsound.h

ภาคผนวก ข.

Error Handling

ค่า Error Handling มาตฐานใน Audio API, Media API และ Input API มีค่าเป็น integer (int) โดยมีค่าสามค่าดังนี้

Value	ความหมาย 1	ความหมาย 2
0	True	ไม่มีข้อผิดพลาด
1	False	ไม่มีข้อผิดพลาด
-1	-	มีข้อผิดพลาดเกิดขึ้น ไม่อาจจะจะทำกระบวนให้สำเร็จได้

ภาคผนวก ค.

แฟลกใน Input API

ค่า Flag ที่ใช้ใน Input API ซึ่งชนิดของตัวแปรเป็น DEVICETYPE มีดังนี้

Constant	Value	ความหมาย	หมายเหตุ
ALL_TYPE	0	ทุกชนิด	ในกรณีที่ต้องการกระทำกับทุกๆ Device
KEY_TYPE	1	Keyboard	
MOUSE_TYPE	2	Mouse	
JOY_TYPE	3	Joystick	

หมายเหตุ ALL_TYPE ไม่สามารถใช้ได้ทุกกรณี หากกรณีใดใช้ไม่ได้ ค่า Error Handling จะ Return เป็น –1

ภาคผนวก ง.

ค่าพารามิเตอร์ใน Media API

MEDIA_ROW

โดย Structure ของ MEDIA_ROW ที่เป็น Row เก็บในตาราง โดยอ้างอิงแต่ละ Row โดยใช้ค่า Handle และมีข้อมูลเป็น MEDIA_ROW มีลักษณะดังนี้

```
struct MEDIA_ROW
{
TCHAR filename[MAX_PATH];
LONG volume;
double rate;
LONGLONG start;
LONGLONG end;
WINDOW_POS pos;
};
```

โดยความหมายมีดังนี้

filename เป็นชื่อ file ของ Media ซึ่งสนับสนุน MPEG, AVI, QuickTime, WAV,AIFF,AU,SND และ MIDI

volume เป็นความดังของเสียง มีค่าระหว่าง 0-100

rate คือค่าอัตราการเล่นของเสียง ปกติจะมีค่าเป็น 1.0

start,end จุดเริ่มและจุดสิ้นสุด หน่วยเป็น 100 nanoseconds

pos เป็นค่าความกว้างยาวและตำแหน่งของ Window ที่แสดงผล โดยมี Struct เป็น

WINDOW_POS

เป็น Structure บอกขนาดของการแสดงผล Video

```
struct WINDOW_POS
{
        long left;
        long top;
        long width;
```

```
        long height;
};
```

left ตำแหน่งของขอบการแสดงผลด้านซ้าย

top ตำแหน่งของขอบการแสดงผลด้านบน

width ขนาดในแนวนอน

height ขนาดในแนวตั้ง

ภาคผนวก จ.

ค่าสถานะที่รับจาก ดีไวซ์ ใน Input API

Keyboard Device Data

ค่าต่างๆสามารถดูได้ใน Dinput.h โดยจะเป็น 256 byte array (BYTE[256]) ซึ่งแทนข้อมูลจาก Keyboard โดยค่าต่างๆจะมีดังนี้

Constant	Note
DIK_0	On main keyboard
DIK_1	On main keyboard
DIK_2	On main keyboard
DIK_3	On main keyboard
DIK_4	On main keyboard
DIK_5	On main keyboard
DIK_6	On main keyboard
DIK_7	On main keyboard
DIK_8	On main keyboard
DIK_9	On main keyboard
DIK_A	
DIK_ABNT_C1	On numeric pad of Brazilian keyboards
DIK_ABNT_C2	On numeric pad of Brazilian keyboards
DIK_ADD	PLUS SIGN (+) on numeric keypad
DIK_APOSTROPHE	
DIK_APPS	Application key
DIK_AT	On Japanese keyboard
DIK_AX	On Japanese keyboard
DIK_B	
DIK_BACK	BACKSPACE
DIK_BACKSLASH	
DIK_C	
DIK_CALCULATOR	
DIK_CAPITAL	CAPS LOCK
DIK_COLON	On Japanese keyboard
DIK_COMMA	

DIK_CONVERT	On Japanese keyboard
DIK_D	
DIK_DECIMAL	PERIOD (decimal point) on numeric keypad
DIK_DELETE	
DIK_DIVIDE	Forward slash (/) on numeric keypad
DIK_DOWN	DOWN ARROW
DIK_E	
DIK_END	
DIK_EQUALS	On main keyboard
DIK_ESCAPE	
DIK_F	
DIK_F1	
DIK_F2	
DIK_F3	
DIK_F4	
DIK_F5	
DIK_F6	
DIK_F7	
DIK_F8	
DIK_F9	
DIK_F10	
DIK_F11	
DIK_F12	
DIK_F13	On NEC PC-98 Japanese keyboard
DIK_F14	On NEC PC-98 Japanese keyboard
DIK_F15	On NEC PC-98 Japanese keyboard
DIK_G	
DIK_GRAVE	Grave accent (`)
DIK_H	
DIK_HOME	
DIK_I	
DIK_INSERT	
DIK_J	

DIK_K	
DIK_KANA	On Japanese keyboard
DIK_KANJI	On Japanese keyboard
DIK_L	
DIK_LBRACKET	Left square bracket [
DIK_LCONTROL	Left CTRL
DIK_LEFT	LEFT ARROW
DIK_LMENU	Left ALT
DIK_LSHIFT	Left SHIFT
DIK_LWIN	Left Microsoft® Windows® logo key
DIK_M	
DIK_MAIL	
DIK_MEDIASELECT	Media Select key, which displays a selection of supported media players on the system
DIK_MEDIASTOP	
DIK_MINUS	On main keyboard
DIK_MULTIPLY	Asterisk (*) on numeric keypad
DIK_MUTE	
DIK_MYCOMPUTER	
DIK_N	
DIK_NEXT	PAGE DOWN
DIK_NEXTTRACK	Next track
DIK_NOCONVERT	On Japanese keyboard
DIK_NUMLOCK	
DIK_NUMPAD0	
DIK_NUMPAD1	
DIK_NUMPAD2	
DIK_NUMPAD3	
DIK_NUMPAD4	
DIK_NUMPAD5	
DIK_NUMPAD6	
DIK_NUMPAD7	
DIK_NUMPAD8	

DIK_NUMPAD9	
DIK_NUMPADCOMMA	On numeric keypad of NEC PC-98 Japanese keyboard
DIK_NUMPADENTER	
DIK_NUMPADEQUALS	On numeric keypad of NEC PC-98 Japanese keyboard
DIK_O	
DIK_OEM_102	On British and German keyboards
DIK_P	
DIK_PAUSE	
DIK_PERIOD	On main keyboard
DIK_PLAYPAUSE	
DIK_POWER	
DIK_PREVTRACK	Previous track; circumflex on Japanese keyboard
DIK_PRIOR	PAGE UP
DIK_Q	
DIK_R	
DIK_RBRACKET	Right square bracket]
DIK_RCONTROL	Right CTRL
DIK_RETURN	ENTER on main keyboard
DIK_RIGHT	RIGHT ARROW
DIK_RMENU	Right ALT
DIK_RSHIFT	Right SHIFT
DIK_RWIN	Right Windows logo key
DIK_S	
DIK_SCROLL	SCROLL LOCK
DIK_SEMICOLON	
DIK_SLASH	Forward slash (/) on main keyboard
DIK_SLEEP	
DIK_SPACE	SPACEBAR
DIK_STOP	On NEC PC-98 Japanese keyboard
DIK_SUBTRACT	MINUS SIGN (-) on numeric keypad
DIK_SYSRQ	
DIK_T	
DIK_TAB	

DIK_U	
DIK_UNDERLINE	On NEC PC-98 Japanese keyboard
DIK_UNLABELED	On Japanese keyboard
DIK_UP	UP ARROW
DIK_V	
DIK_VOLUMEDOWN	
DIK_VOLUMEUP	
DIK_W	
DIK_WAKE	
DIK_WEBBACK	
DIK_WEBFAVORITES	Displays the Microsoft Internet Explorer Favorites list, the Windows Favorites folder, or the Netscape Bookmarks list.
DIK_WEBFORWARD	
DIK_WEBHOME	
DIK_WEBREFRESH	
DIK_WEBSEARCH	
DIK_WEBSTOP	
DIK_X	
DIK_Y	
DIK_YEN	On Japanese keyboard
DIK_Z	

ค่า Constant อาจจะมีชื่ออื่นอีก ดังเช่น

Alternate name	Regular name	Note
DIK_BACKSPACE	DIK_BACK	BACKSPACE
DIK_CAPSLOCK	DIK_CAPITAL	CAPS LOCK
DIK_CIRCUMFLEX	DIK_PREVTRACK	On Japanese keyboard
DIK_DOWNARROW	DIK_DOWN	On arrow keypad
DIK_LALT	DIK_LMENU	Left ALT
DIK_LEFTARROW	DIK_LEFT	On arrow keypad
DIK_NUMPADMINUS	DIK__SUBTRACT	MINUS SIGN (-) on numeric keypad
DIK_NUMPADPERIOD	DIK_DECIMAL	PERIOD (decimal point) on numeric keypad

DIK_NUMPADPLUS	DIK_ADD	PLUS SIGN (+) on numeric keypad
DIK_NUMPADSLASH	DIK__DIVIDE	Forward slash (/) on numeric keypad
DIK_NUMPADSTAR	DIK_MULTIPLY	Asterisk (*) on numeric keypad
DIK_PGDN	DIK_NEXT	On arrow keypad
DIK_PGUP	DIK_PRIOR	On arrow keypad
DIK_RALT	DIK_RMENU	Right ALT
DIK_RIGHTARROW	DIK_RIGHT	On arrow keypad
DIK_UPARROW	DIK_UP	On arrow keypad

Mouse Device Data

โดย Structure ที่ใช้คือ DIMOUSESTATE2

```
typedef struct DIMOUSESTATE2 {
  LONG lX;
  LONG lY;
  LONG lZ;
  BYTE rgbButtons[8];
} DIMOUSESTATE2, *LPDIMOUSESTATE2;
```

lX

ค่าแกน X

lY

ค่าแกน Y

lZ

ค่าแกน Z ซึ่งเป็นค่าสำหรับ Wheel Mouse หากไม่มี ค่าจะเป็น 0

rgbButtons

เป็น Array ของปุ่ม หากส่วน hight bit ของ byte ถูก set แสดงว่ามีการกดปุ่ม

ค่า Constant แทนปุ่มต่างๆจะอยู่ใน Dinput.h โดยมีค่าต่างๆดังนี้

Constant	Note
DIMOFS_BUTTON0	

DIMOFS_BUTTON1

DIMOFS_BUTTON2

DIMOFS_BUTTON3

DIMOFS_BUTTON4

DIMOFS_BUTTON5

DIMOFS_BUTTON6

DIMOFS_BUTTON7

DIMOFS_X

DIMOFS_Y

DIMOFS_Z

Joystick Device Data

Data ในการรับข้อมูลจาก Joystick จะใช้ Structure DIJOYSTATE2

```
typedef struct DIJOYSTATE2 {
    LONG lX;
    LONG lY;
    LONG   lZ;
    LONG    lRx;
    LONG    lRy;
    LONG    lRz;
    LONG    rglSlider[2];
    DWORD   rgdwPOV[4];
    BYTE    rgbButtons[128];
    LONG    lVX;
    LONG    lVY;
    LONG    lVZ;
    LONG    lVRx;
    LONG    lVRy;
    LONG    lVRz;
    LONG    rglVSlider[2];
    LONG    lAX;
    LONG    lAY;
```

```
    LONG    lAZ;
    LONG    lARx;
    LONG    lARy;
    LONG    lARz;
    LONG    rglASlider[2];
    LONG    lFX;
    LONG    lFY;
    LONG    lFZ;
    LONG    lFRx;
    LONG    lFRy;
    LONG    lFRz;
    LONG    rglFSlider[2];
} DIJOYSTATE2, *LPDIJOYSTATE2;
```

lX

แกน X หรือการเคลื่อนที่ซ้ายขวา

lY

แกน Y หรือการเคลื่อนหน้าหลัง

lZ

แกน Z หากไม่มีแกนนี้ ค่าจะเป็น 0

lRx

การหมุนแกน X หากไม่มีแกนนี้ ค่าจะเป็น 0

lRy

การหมุนแกน Y หากไม่มีแกนนี้ ค่าจะเป็น 0

lRz

การหมุนแกน Z หากไม่มีแกนนี้ ค่าจะเป็น 0

rglSlider[2]

แกนเสริม 2 แกน (แกน U และ แกน V)

rgdwPOV[4]

ทิศทาง หรือเรียกว่า Point-of-view โดยทิศทางจะมีค่าเป็นองศาจากทิศเหนือไปตามเข็มนาฬิกา คือ 0, 9,000, 18,000, หรือ 27,000 หากไม่มีการกดค่าจะเป็น -1

rgbButtons[128]

Array ของปุ่ม หากส่วน hight bit ของ byte ถูก set แสดงว่ามีการกดปุ่ม

lVX

ความเร็วแกน X

lVY

ความเร็วแกน Y

lVZ

ความเร็วแกน Z

lVRx

ความเร็วเชิงมุมแกน X

lVRy

ความเร็วเชิงมุมแกน Y

lVRz

ความเร็วเชิงมุมแกน Z

rglVSlider[2]

ความเร็วแกนเสริม

lAX

ความเร่งแกน X

lAY

ความเร่งแกน Y

lAZ

ความเร่งแกน Z

lARx

ความเร่งเชิงมุมแกน X

lARy

ความเร่งเชิงมุมแกน Y

lARz

ความเร่งเชิงมุมแกน Z

rglASlider[2]

ความเร่งเชิงมุมแกนเสริม

lFX

แรงของแกน X

lFY

แรงของแกน Y

lFZ

แรงของแกน Z

lFRx

Torque แกน X

lFRy

Torque แกน Y

lFRz

Torque แกน Z

rglFSlider[2]

Torque แกนเสริม

ค่า Constant ของ Joystick แทนปุ่มต่างๆมีดังนี้

Constant	Device object
DIJOFS_BUTTON0 *to* DIJOFS_BUTTON31 *or* DIJOFS_BUTTON(*n*)	Button
DIJOFS_POV(*n*)	Point-of-view indicator
DIJOFS_RX	X-axis rotation
DIJOFS_RY	Y-axis rotation
DIJOFS_RZ	Z-axis rotation (rudder)
DIJOFS_X	X-axis
DIJOFS_Y	Y-axis
DIJOFS_Z	Z-axis
DIJOFS_SLIDER(*n*)	Slider axis

ภาคผนวก ฉ.

Device Type

ข้อมูล Main-Type และ Sub-Type ซึ่งเป็นข้อมูลระบุชนิดของ Device ในรายละเอียดที่ลึกยิ่งขึ้น โดย

int DI::Param::getInfoDeviceMainType(DEVICETYPE const type,DWORD const handle,DWORD& manintype);

int DI::Param::getInfoDeviceSubType(DEVICETYPE const type,DWORD const handle,DWORD& subtype);

int DI::Param::getInfoDeviceType(DEVICETYPE const type,DWORD const handle,DWORD& alltype);

โดยค่าที่ได้จาก getInfoDeviceType จะเป็นการเอาค่า Main-Type และ Sub-Type มา AND กัน โดยค่า Constant จะถูก Define ไว้ใน Dinput.h

DI8DEVTYPE_1STPERSON First-person action game device

DI8DEVTYPE1STPERSON_LIMITED

Device that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPE1STPERSON_SHOOTER

Device designed for first-person shooter games.

DI8DEVTYPE1STPERSON_SIXDOF

Device with six degrees of freedom; that is, three lateral axes and three rotational axes.

DI8DEVTYPE1STPERSON_UNKNOWN

Unknown subtype.

DI8DEVTYPE_DEVICE Device that does not fall into another category.

DI8DEVTYPE_DEVICECTRL Input device used to control another type of device from within the context of the application.

DI8DEVTYPEDEVICECTRL_COMMSSELECTION

Control used to make communications selections.

DI8DEVTYPEDEVICECTRL_COMMSSELECTION_HARDWIRED

Device that must use its default configuration and cannot be remapped.

DI8DEVTYPEDEVICECTRL_UNKNOWN

Unknown subtype.

DI8DEVTYPE_DRIVING Device for steering

DI8DEVTYPEDRIVING_COMBINEDPEDALS

Steering device that reports acceleration and brake pedal values from a single axis.

DI8DEVTYPEDRIVING_DUALPEDALS

Steering device that reports acceleration and brake pedal values from separate axes.

DI8DEVTYPEDRIVING_HANDHELD

Hand-held steering device.

DI8DEVTYPEDRIVING_LIMITED

Steering device that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPEDRIVING_THREEPEDALS

Steering device that reports acceleration, brake, and clutch pedal values from separate axes.

DI8DEVTYPE_FLIGHT Controller for flight simulation.

DI8DEVTYPEFLIGHT_LIMITED

Flight controller that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPEFLIGHT_RC

Flight device based on a remote control for model aircraft.

DI8DEVTYPEFLIGHT_STICK

Joystick.

DI8DEVTYPEFLIGHT_YOKE

Yoke.

DI8DEVTYPE_GAMEPAD Gamepad. The following subtypes are defined.

DI8DEVTYPEGAMEPAD_LIMITED

Gamepad that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPEGAMEPAD_STANDARD

Standard gamepad that provides the minimum number of device objects for action mapping.

DI8DEVTYPEGAMEPAD_TILT

Gamepad that can report x-axis and y-axis data based on the attitude of the controller.

DI8DEVTYPE_JOYSTICK Joystick

DI8DEVTYPEJOYSTICK_LIMITED

Joystick that does not provide the minimum number of device objects for action mapping.

DI8DEVTYPEJOYSTICK_STANDARD

Standard joystick that provides the minimum number of device objects for action mapping.

DI8DEVTYPE_KEYBOARD Keyboard or keyboard-like device.

DI8DEVTYPEKEYBOARD_UNKNOWN

Subtype could not be determined.

DI8DEVTYPEKEYBOARD_PCXT

IBM PC/XT 83-key keyboard.

DI8DEVTYPEKEYBOARD_OLIVETTI

Olivetti 102-key keyboard.

DI8DEVTYPEKEYBOARD_PCAT

IBM PC/AT 84-key keyboard.

DI8DEVTYPEKEYBOARD_PCENH

IBM PC Enhanced 101/102-key or Microsoft Natural® keyboard.

DI8DEVTYPEKEYBOARD_NOKIA1050

Nokia 1050 keyboard.

DI8DEVTYPEKEYBOARD_NOKIA9140

Nokia 9140 keyboard.

DI8DEVTYPEKEYBOARD_NEC98

Japanese NEC PC98 keyboard.

DI8DEVTYPEKEYBOARD_NEC98LAPTOP

Japanese NEC PC98 laptop keyboard.

DI8DEVTYPEKEYBOARD_NEC98106

Japanese NEC PC98 106-key keyboard.

DI8DEVTYPEKEYBOARD_JAPAN106

Japanese 106-key keyboard.

DI8DEVTYPEKEYBOARD_JAPANAX

Japanese AX keyboard.

DI8DEVTYPEKEYBOARD_J3100

Japanese J3100 keyboard.

DI8DEVTYPE_MOUSE A mouse or mouse-like device (such as a trackball

DI8DEVTYPEMOUSE_ABSOLUTE

Mouse that returns absolute axis data.

DI8DEVTYPEMOUSE_FINGERSTICK

Fingerstick.

DI8DEVTYPEMOUSE_TOUCHPAD

Touchpad.

DI8DEVTYPEMOUSE_TRACKBALL

Trackball.

DI8DEVTYPEMOUSE_TRADITIONAL

Traditional mouse.

DI8DEVTYPEMOUSE_UNKNOWN

Subtype could not be determined.

DI8DEVTYPE_REMOTE Remote-control device.

DI8DEVTYPEREMOTE_UNKNOWN

Subtype could not be determined.

DI8DEVTYPE_SCREENPOINTER Screen pointer

DI8DEVTYPESCREENPTR_UNKNOWN

Unknown subtype.

DI8DEVTYPESCREENPTR_LIGHTGUN

Light gun.

DI8DEVTYPESCREENPTR_LIGHTPEN

Light pen.

DI8DEVTYPESCREENPTR_TOUCH

Touch screen.

DI8DEVTYPE_SUPPLEMENTAL Specialized device with functionality unsuitable for the main control of an application, such as pedals used with a wheel.

DI8DEVTYPESUPPLEMENTAL_2NDHANDCONTROLLER

Secondary handheld controller.

DI8DEVTYPESUPPLEMENTAL_COMBINEDPEDALS

Device whose primary function is to report acceleration and brake pedal values from a single axis.

DI8DEVTYPESUPPLEMENTAL_DUALPEDALS

Device whose primary function is to report acceleration and brake pedal values from separate axes.

DI8DEVTYPESUPPLEMENTAL_HANDTRACKER

Device that tracks hand movement.

DI8DEVTYPESUPPLEMENTAL_HEADTRACKER

Device that tracks head movement.

DI8DEVTYPESUPPLEMENTAL_RUDDERPEDALS

Device with rudder pedals.

DI8DEVTYPESUPPLEMENTAL_SHIFTER

Device that reports gear selection from an axis.

DI8DEVTYPESUPPLEMENTAL_SHIFTSTICKGATE

Device that reports gear selection from button states.

DI8DEVTYPESUPPLEMENTAL_SPLITTHROTTLE

Device whose primary function is to report at least two throttle values. It may have other controls.

DI8DEVTYPESUPPLEMENTAL_THREEPEDALS

Device whose primary function is to report acceleration, brake, and clutch pedal values from separate axes.

DI8DEVTYPESUPPLEMENTAL_THROTTLE

Device whose primary function is to report a single throttle value. It may have other controls.

DI8DEVTYPESUPPLEMENTAL_UNKNOWN

Unknown subtype.

ภาคผนวก ช.

วิธีใช้งานออดิโอ

Direct Audio จะใช้สถาปัตยกรรม COM ของ Microsoft ซึ่ง Audio API ที่พัฒนาขึ้นมาก็พัฒนาจาก Direct Audio ดังนั้น ก่อนการใช้จะต้อง Initialize COM ก่อน โดยใช้คำสั่ง

CoInitialize(NULL);

เมื่อเริ่มการใช้งาน Audio Library โดยเรียก init (HWND คือ window handle) และเมื่อต้องการสิ้นสุดการใช้ก็เรียกคำสั่ง cleanup

```
int     DA::init(HWND hWnd);
void    DA::cleanup();
```

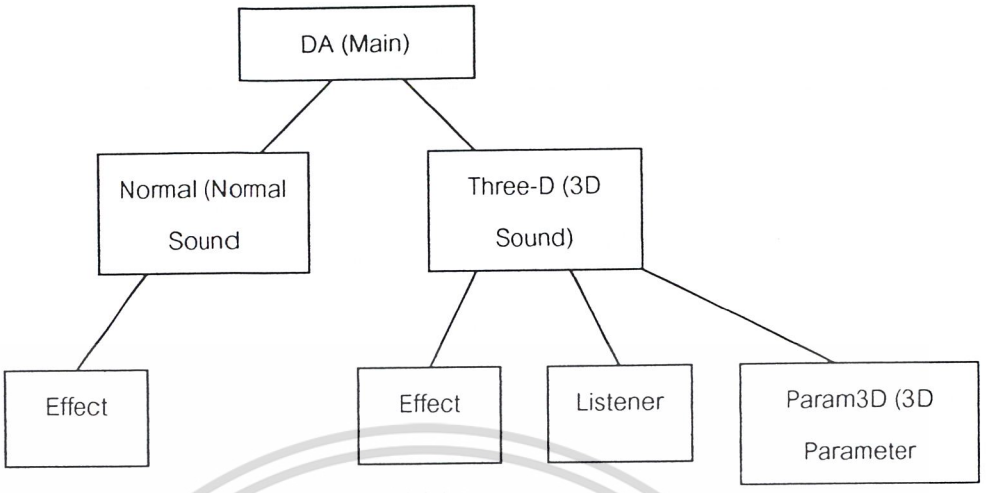
หลังการ Cleanup ต้องไม่ลืมที่จะ UnInitialize COM โดยใช้คำสั่ง

CoUninitialize();

ใน Audio Library จะแยกออกเป็นออกเป็น 2 ส่วนคือ Normal และ ThreeD โดย Normal จะเป็นส่วนของเสียงทั่วไป และ ThreeD จะเป็นส่วนของเสียง 3 มิติ ซึ่งจะมี Function จัดการเกี่ยวกับเสียง 3 มิติเพิ่มเข้ามา แต่จะมี Function ที่เกี่ยวกับบริหาร object,Effect และการเล่นเหมือนกัน โดยแบ่งออกเป็นส่วนๆตาม Namespace ดังนี้

DA

- Normal: ส่วนของ Normal Sound Object
  - Effect: ส่วนของ Effect ของ Normal Sound
- ThreeD: ส่วนของ 3D Sound Object
  - Effect: ส่วนของ Effect ของ 3D Sound
  - Listener: ส่วนของการตั้งค่าผู้ฟัง
  - Param3D: ส่วนของ 3D Parameter ของ 3D Sound



ส่วนการบริหาร Object

int DA::Normal::create(TCHAR* filename,int& handle);

int DA::ThreeD::create(TCHAR* filename,int& handle);

สร้าง Object sound โดยจะได้ handle number ออกไป โดยใช้ค่า handle number ที่ได้ในการอ้างอิงต่อ object ที่สร้าง โดยต้องระบุชื่อ file ซึ่งเป็น WAVE หรือ MIDI

int DA::Normal::duplicate(int const s_handle,int& t_handle);

int DA::ThreeD::duplicate(int const s_handle,int& t_handle);

คัดลอก Object โดยจะคล้ายกับการสร้างใหม่ โดย object จะมีคุณสมบัติเหมือนต้นฉบับ โดยจะได้ค่า handle number เหมือนกับการสร้างใหม่

int DA::Normal::destroy(int const handle);

int DA::ThreeD::destroy(int const handle);

ทำลาย object

int DA::Normal::check(int const handle);

7int DA::ThreeD::check(int const handle);

ตรวจสอบว่ามี object อยู่หรือไม่

ส่วนการเล่นเสียง

void DA::Normal::play(int const handle);

void DA::ThreeD::play(int const handle);

เล่นเสียง

void DA::Normal::stop(int const handle);

void DA::ThreeD::stop(int const handle);

หยุดการเล่นเสียง

int DA::Normal::load(int const handle,TCHAR* filename);

int DA::ThreeD::load(int const handle,TCHAR* filename);

Load เสียงขึ้นมาใหม่ โดยระบุชื่อ file ที่เป็น WAVE หรือ MIDI

int DA::Normal::setloop(int const handle);

int DA::ThreeD::setloop(int const handle);

ตั้งการเล่นเสียงเป็นลักษณะวน Loop โดยวนไม่มีที่สิ้นสุด

int DA::Normal::setloop(int const handle,DWORD const loop);

int DA::ThreeD::setloop(int const handle,DWORD const loop);

ตั้งการเล่นเสียงเป็นลักษณะวน Loop โดยระบุจำนวนรอบ

int DA::Normal::resetloop(int const handle);

int DA::ThreeD::resetloop(int const handle);

ยกเลิกการเล่นเสียงลักษณะวน

int DA::Normal::getloop(int const handle,DWORD& loop);

int DA::ThreeD::getloop(int const handle,DWORD& loop);

ดูจำนวน Loop การเล่นของ Sound Object

int DA::Normal::isPlay(int const handle);

int DA::ThreeD::isPlay(int const handle);

ตรวจสอบว่าสถานะขณะนั้น เล่นเสียงอยู่หรือเปล่า

int DA::Normal::getVolume(int const handle,LONG& vol);

int DA::ThreeD::getVolume(int const handle,LONG& vol);

ตรวจสอบค่าความดังของเสียง โดยค่าสูงสุดคือ 100 และค่าต่ำสุดคือ 0

int DA::Normal::setVolume(int const handle,LONG const vol);

int DA::ThreeD::setVolume(int const handle,LONG const vol);

ตั้งค่าความดังของเสียง โดยค่าสูงสุดคือ 100 และค่าต่ำสุดคือ 0

ส่วนของ Effect

ในส่วน Effect จะเป็นส่วนย่อยของทั้ง Normal และ ThreeD โดยใช้ Namespace ในการแบ่ง โดยจะมี Function แบบเดียวกัน แต่อยู่คนละส่วนคือ DA::Normal::Effect และ DA::ThreeD::Effect โดยมี Function ดังนี้

int enableChorus(int const handle);

int enableCompressor(int const handle);

int enableDistortion(int const handle);

int enableEcho(int const handle);

int enableFlanger(int const handle);

int enableGargle(int const handle);

int enableParameq(int const handle);

int enableReverb(int const handle);

เปิด Effect ของ Sound Object

int disableChorus(int const handle);

int disableCompressor(int const handle);

int disableDistortion(int const handle);

int disableEcho(int const handle);

int disableFlanger(int const handle);

int disableGargle(int const handle);

int disableParameq(int const handle);

int disableReverb(int const handle);

int disableAllEffect(int const handle);

ปิด Effect ของ SoundObject

```
int     isChorus(int const handle);
int     isCompressor(int const handle);
int     isDistortion(int const handle);
int     isEcho(int const handle);
int     isFlanger(int const handle);
int     isGargle(int const handle);
int     isParameq(int const handle);
int     isReverb(int const handle);
int     isEffect(int const handle);
```

ตรวจสอบว่าเปิด Effect นั้นๆอยู่หรือไม่

```
int     setChorus(int const handle);
int     setCompressor(int const handle);
int     setDistortion(int const handle);
int     echo(int const handle);
int     setFlanger(int const handle);
int     setGargle(int const handle);
int     setParameq(int const handle);
int     setReverb(int const handle);
```

ตั้งค่า Parameter ของ Effect นั้นๆเป็น Default (ต้อง Enable Effect นั้นๆก่อน)

```
int     setChorus(int const handle,DSFXChorus const p);
int     setCompressor(int const handle,DSFXCompressor const p);
int     setDistortion(int const handle,DSFXDistortion const p);
int     setEcho(int const handle,DSFXEcho const p);
int     setFlanger(int const handle,DSFXFlanger const p);
int     setGargle(int const handle,DSFXGargle const p);
int     setParameq(int const handle,DSFXParamEq const p);
int     setReverb(int const handle,DSFXWavesReverb const p);
```

ตั้งค่า Parameter ของ Effect นั้นๆ โดยระบุ Parameter เอง (ต้อง Enable Effect นั้นๆก่อน)

```
int     getChorus(int const handle,DSFXChorus& p);
```

int getCompressor(int const handle,DSFXCompressor& p);

int getDistortion(int const handle,DSFXDistortion& p);

int getEcho(int const handle,DSFXEcho& p);

int getFlanger(int const handle,DSFXFlanger& p);

int getGargle(int const handle,DSFXGargle& p);

int getParameq(int const handle,DSFXParamEq& p);

int getReverb(int const handle,DSFXWavesReverb& p);

ดูค่า Parameter ที่ได้ตั้งเอาไว้

หมายเหตุ ความหมายของค่า Parameter ต่างๆ สามารถดูได้ที่ภาคผนวก

ส่วนของเสียง 3 มิติ

ในส่วนของ ThreeD จะมี Function เกี่ยวกับเสียง 3 มิติขึ้นมา โดยแบ่งเป็นสองส่วนคือ Listener (DA::ThreeD::Listener) ซึ่งจะเป็น Function เกี่ยวกับ Parameter เกี่ยวกับผู้ฟัง และ Param3D (DA::ThreeD::Param3D) ซึ่งเป็น Parameter ของ Sound Object ที่อยู่ในโลก 3 มิติ

int DA::ThreeD::Listener ::setOrientation(D3DVECTOR const front,D3DVECTOR const top);

int DA::ThreeD::Listener ::setOrientation(D3DVALUE const fx,D3DVALUE const fy,D3DVALUE const fz,D3DVALUE const tx,D3DVALUE const ty,D3DVALUE const tz);

int DA::ThreeD::Listener ::getOrientation(D3DVECTOR& fort,D3DVECTOR& tort);

ตั้งและรับค่า Head Orientation ของผู้ฟัง โดย D3DVECTOR เป็นค่าแทน Vector ใน direct X โดย D3DVALUE แทนค่า x, y, z

int DA::ThreeD::Listener ::setPosition(D3DVECTOR const pos);

int DA::ThreeD::Listener ::setPosition(D3DVALUE const x,D3DVALUE const y,D3DVALUE const z);

int DA::ThreeD::Listener ::getPosition(D3DVECTOR& pos);

ตั้งและรับค่าของตำแหน่งผู้ฟัง

int DA::ThreeD::Listener ::setVelocity(D3DVECTOR const vel);

int DA::ThreeD::Listener ::setVelocity(D3DVALUE const x,D3DVALUE const y,D3DVALUE const z);

int DA::ThreeD::Listener ::getVelocity(D3DVECTOR& vel);

ตั้งค่าและรับค่าของความเร็วของผู้ฟัง

int DA::ThreeD::Listener ::setDoppler(D3DVALUE const dop);

int DA::ThreeD::Listener ::getDoppler(D3DVALUE& dop);

ตั้งและรับค่า Doppler (Doppler คือค่า factor ที่เกี่ยวกับการปิดเปี้ยวของคลื่นเสียงเมื่อมีการเคลื่อนที่) โดยค่า Default จะอยู่ที่ 1 (1 เท่า) โดยมีค่าระหว่าง 0.0 ถึง 10.0

int DA::ThreeD::Listener ::setRolloff(D3DVALUE const rol);

int DA::ThreeD::Listener ::getRolloff(D3DVALUE& rol);

ตั้งและรับค่า Rolloff factor โดยมีค่าอยู่ระหว่าง 0 ถึง 10 เท่า (ค่า Default เป็น 1.0)

int DA::ThreeD::Param3D ::setMaxDistance(int const handle,D3DVALUE const maxDist);

int DA::ThreeD::Param3D ::getMaxDistance(int const handle,D3DVALUE& maxDist);

int DA::ThreeD::Param3D ::setMinDistance(int const handle,D3DVALUE const minDist);

int DA::ThreeD::Param3D ::getMinDistance(int const handle,D3DVALUE& minDist);

ตั้งค่า MaxDistance และ MinDistance

int DA::ThreeD::Param3D ::setPosition(int const handle,D3DVECTOR const pos);

int DA::ThreeD::Param3D ::setPosition(int const handle,D3DVALUE const x,D3DVALUE const y,D3DVALUE const z);

int DA::ThreeD::Param3D ::movePosition(int const handle,D3DVECTOR const pos);

int DA::ThreeD::Param3D ::movePosition(int const handle,D3DVALUE const x,D3DVALUE const y,D3DVALUE const z);

int DA::ThreeD::Param3D ::getPosition(int const handle,D3DVECTOR& pos);

ตั้งและรับค่าตำแหน่งของเสียง

int DA::ThreeD::Param3D ::setNormalMode(int const handle);

int DA::ThreeD::Param3D ::setRelativeMode(int const handle);

int DA::ThreeD::Param3D ::isNormalMode(int const handle);

int DA::ThreeD::Param3D ::isRelativeMode(int const handle);

ตั้งและรับค่า Mode ของเสียง โดย Mode Relative จะมีการปรับค่า Cone, Velocity, Position จะถูก Update อย่างอัตโนมัติเมื่อมีการเปลี่ยนแปลง Parameter ของ Listener โดยค่า Default จะเป็น Mode Normal

int DA::ThreeD::Param3D ::setVelocity(int const handle,D3DVECTOR const vel);

int DA::ThreeD::Param3D ::setVelocity(int const handle,D3DVALUE const x,D3DVALUE const y,D3DVALUE const z);

int DA::ThreeD::Param3D ::getVelocity(int const handle,D3DVECTOR& vel);

ตั้งค่าความเร็วของวัตถุเสียง

int DA::ThreeD::Param3D ::setConeAngles(int const handle,DWORD const in,DWORD const out);

int DA::ThreeD::Param3D ::setConeOrientation(int const handle,D3DVECTOR const ort);

int DA::ThreeD::Param3D ::setConeOutsideVolume(int const handle,LONG const vol);

int DA::ThreeD::Param3D ::getConeAngles(int const handle,DWORD& in,DWORD& out);

int DA::ThreeD::Param3D ::getConeOrientation(int const handle,D3DVECTOR& ort);

int DA::ThreeD::Param3D ::getConeOutsideVolume(int const handle,LONG& vol);

ตั้งและรับค่าของ Sound Cone

ภาคผนวก ซ.

วิธีใช้งานอินพุต

การอ้างอิงต่อ Device Object ใน API จะใช้ค่า Flag เป็นตัวอ้างอิง ดังต่อไปนี้

ALL_TYPE = 0: ทุกชนิด

KEY_TYPE = 1: Keyboard

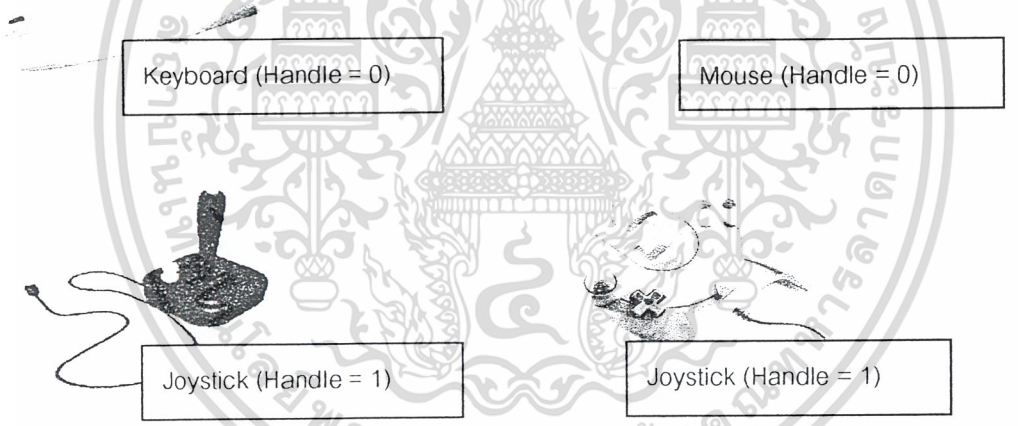
MOUSE_TYPE = 2: Mouse

JOY_TYPE = 3: Joystick

โดยแต่ละชนิดจะมีจำนวนได้มากกว่าหนึ่ง เช่น สามารถอ้างอิง Joystick ตัวที่สองได้โดยใช้ Flag JOY_TYPE และระบุหมายเลข 1(0 คือ Joystick ตัวแรก)

แต่คุณสมบัติของ Device แต่ละชนิดไม่เหมือนกัน ดังนั้น Function บาง Function จะสามารถใช้ได้เมื่ออ้างอิงต่อ Device ถูกชนิดเท่านั้น หาก Device ไม่สนับสนุนก็จะ Return ค่า Error ออกมา

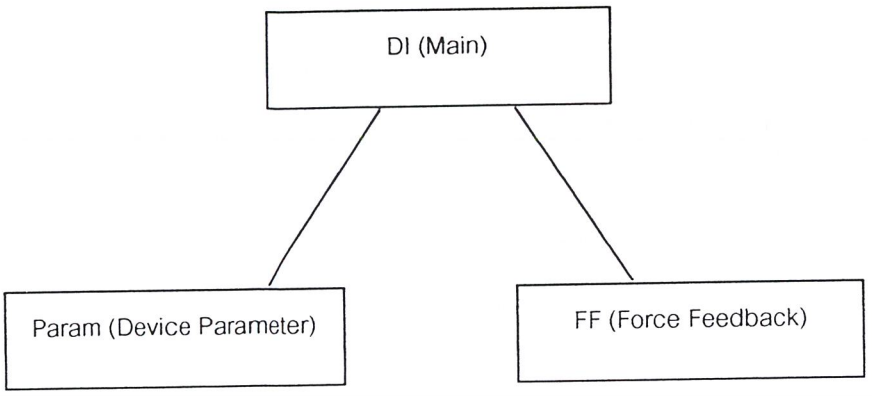
เช่น PC เครื่องหนึ่งมี Keyboard หนึ่งตัว, Mouse หนึ่งตัว, Joystick สองตัวเป็นต้น



Namespace แบ่งออกเป็นดังนี้

DI : จะเป็นส่วนหลักของ Input API

- Param : เกี่ยวกับ Parameter ของ Device
- FF : เกี่ยวกับ Force Feedback



หมายเหตุ: ค่าที่ Return จาก Function นั้นจะใช้ค่า integer มีอยู่สามค่าคือ

0: ค่าเป็น True ไม่มีข้อผิดพลาดและ

1: ค่าเป็น False และไม่มีข้อผิดพลาด,

-1: มีข้อผิดพลาดเกิดขึ้น ไม่อาจจะจะทำกระบวนให้สำเร็จได้

ส่วนหลัก

ส่วนหลักจะประกอบไปด้วย Function Initialize ,Cleanup ,บริหารจัดการ Device และดึงค่า Input State จาก Device

โดยเริ่มต้นใช้งาน Input API จะต้องทำการ Initiliaze ก่อนโดย HWND คือ window handle

```
int DI::init(HWND const hwnd);
```

เมื่อต้องการยกเลิกการใช้งาน ก็ต้องทำการ Cleanup โดย

```
void DI::cleanup();
```

เมื่อทำการ Initiliaze แล้วเราต้องสร้าง Device ขึ้นมาก่อน โดยค่าที่ keyNum, mouseNum, joystickNum คือค่าที่ return ออกมาเพื่อบอกจำนวนของ Device ชนิดๆนั้นที่สร้างขึ้นได้ เช่น PC เครื่องหนึ่งมี Keyboard หนึ่งตัว, Mouse หนึ่งตัว, Joystick สองตัว จะได้ค่า keyNum เท่ากับ 1, mouseNum เท่ากับ 1 และ joystickNum เท่ากับ 2

```
int DI::createDevice(DWORD& keyNum,DWORD& mouseNum,DWORD& joystickNum);
```

และเมื่อต้องการทำลาย Device ทั้งหมดก็

```
void DI::destroyDevice();
```

เมื่อทำการ Create Device แล้ว เราสามารถดูจำนวน Device ทั้งหมดได้โดย

int DI::getDeviceNum(DWORD& num);

หรือหากต้องการระบุชนิดของ Device ก็สามารถทำได้โดยระบุ Flag

int DI::getDeviceNum(DEVICETYPE const type,DWORD& num);

เมื่อต้องการใช้งาน Device ใดๆ ต้องทำการ Enable ก่อน โดยต้องระบุ Flag ของชนิด Device (ALL_TYPE ไม่สามารถใช้ได้) และ handle ของ Device นั้นๆ

int DI::enableDevice(DEVICETYPE const type,DWORD const handle);

และเมื่อต้องการยกเลิกการใช้ชั่วคราวก็สามารถ Disable ได้โดยการระบุ Flag และ Handle เช่นกัน

int DI::disableDevice(DEVICETYPE const type,DWORD const handle);

Device ที่สร้างขึ้นได้นั้น หมายถึง Device นั้นๆ ได้ติดตั้ง Driver นั้นๆเอาไว้ ซึ่งไม่จำเป็นจะต้องติดตั้งอยู่จริงๆ เช่น Joystick ที่ไม่ได้ต่อเอาไว้ เราสามารถดูสถานะว่า Device นั้นๆ ว่าต่อไว้หรือไม่โดยต้องระบุ Flag และ Handle (ALL_TYPE ไม่สามารถใช้ได้)

int DI::isAttatch(DEVICETYPE const type,DWORD const handle);

การดึงข้อมูล Input จาก Device ต่างๆ เมื่อนำมาใช้ในการประมวลผลนั้นสามารถทำได้ โดยระบุ Handle ของ Device นั้นๆ

สังเกตว่าไม่มีการระบุ Flag เพราะ Device แต่ละชนิดจะมีชนิดของข้อมูลไม่เหมือนกัน Keyboard จะใช้ Array ของ Byte 256 ตัว, Mouse จะใช้ DIMOUSESTATE2 และ Joystick จะใช้ DIJOYSTATE2

หมายเหตุ รายละเอียดของชนิดข้อมูลต่างๆ สามารถดูได้ที่ภาคผนวก

int DI::getState(DWORD const handle,BYTE* pBuffer);

int DI::getState(DWORD const handle,DIMOUSESTATE2& buffer);

int DI::getState(DWORD const handle,DIJOYSTATE2& buffer);

ส่วน Parameter ของ Device

เราสามารถดู GUID ของ Device และ Force-Feedback Device ได้โดย

int DI::Param::getInfoGuidProduct(DEVICETYPE const type,DWORD const handle,GUID& guidProduct);

int DI::Param::getInfoGuidFFDriver(DEVICETYPE const type,DWORD const handle,GUID& guidFFDriver);

ข้อมูลที่บ่งบอกชื่อของ Device ชนิดนั้นๆ สามารถดูได้โดย

int DI::Param::getInfoInstanceName(DEVICETYPE const type,DWORD const handle,TCHAR* tszInstanceName);

int DI::Param::getInfoProductName(DEVICETYPE const type,DWORD const handle,TCHAR* tszProductName);

ข้อมูล Main-Type และ Sub-Type ซึ่งเป็นข้อมูลระบุชนิดของ Device ในรายละเอียดที่ลึกยิ่งขึ้นก็สามารถทำได้โดย

int DI::Param::getInfoDeviceMainType(DEVICETYPE const type,DWORD const handle,DWORD& manintype);

int DI::Param::getInfoDeviceSubType(DEVICETYPE const type,DWORD const handle,DWORD& subtype);

int DI::Param::getInfoDeviceType(DEVICETYPE const type,DWORD const handle,DWORD& alltype);

โดยการ getInfoDeviceType เป็นการผนวกทั้ง Main-Type และ Sub-Type เข้าด้วยกัน

หมายเหตุ รายละเอียดสามารถดูได้ที่ภาคผนวก

เราสามารถดูข้อมูลจำนวนแกนและจำนวนปุ่ม(เฉพาะ Mouse กับ Joystick) ได้โดย

int DI::Param::getInfoAxes(DEVICETYPE const type,DWORD const handle,DWORD& numAxes);

int DI::Param::getInfoPOVs(DEVICETYPE const type,DWORD const handle,DWORD& numPOVs);

int DI::Param::getInfoButtons(DEVICETYPE const type,DWORD const handle,DWORD& numButtons);

ส่วน Parameter Dead Zone และ Saturation (เฉพาะ Mouse กับ Joystick)สามารถตั้งค่าและดูค่าได้โดย

int DI::Param::setDeadZone(DEVICETYPE const type,DWORD const handle,LONG const deadzone);

int DI::Param::getDeadZone(DEVICETYPE const type,DWORD const handle,LONG& deadzone);

int DI::Param::setSaturation(DEVICETYPE const type,DWORD const handle,LONG const saturation);

int DI::Param::getSaturation(DEVICETYPE const type,DWORD const handle,LONG& saturation);

การตั้งค่าและดูค่าขอบเขต(Range) ของ Mouse หรือ Joystick ก็สามารถทำได้โดย

int DI::Param::setRange(DEVICETYPE const type,DWORD const handle,LONG const range);

int DI::Param::getRange(DEVICETYPE const type,DWORD const handle,LONG& range);

หรือเลือกเฉพาะแกนที่ต้องการได้โดย

int DI::Param::setRangeX(DEVICETYPE const type,DWORD const handle,LONG const range);

int DI::Param::getRangeX(DEVICETYPE const type,DWORD const handle,LONG& range);

int DI::Param::setRangeY(DEVICETYPE const type,DWORD const handle,LONG const range);

int DI::Param::getRangeY(DEVICETYPE const type,DWORD const handle,LONG& range);

int DI::Param::setRange(DEVICETYPE const type,DWORD const handle,LONG const range);

int DI::Param::getRange(DEVICETYPE const type,DWORD const handle,LONG& range);

int DI::Param::setRangeZ(DEVICETYPE const type,DWORD const handle,LONG const range);

int DI::Param::getRangeZ(DEVICETYPE const type,DWORD const handle,LONG& range);

int DI::Param::setRangeRX(DEVICETYPE const type,DWORD const handle,LONG const range);

int DI::Param::getRangeRX(DEVICETYPE const type,DWORD const handle,LONG& range);

int DI::Param::setRangeRY(DEVICETYPE const type,DWORD const handle,LONG const range);

int DI::Param::getRangeRY(DEVICETYPE const type,DWORD const handle,LONG& range);

int DI::Param::setRangeRZ(DEVICETYPE const type,DWORD const handle,LONG const range);

int DI::Param::getRangeRZ(DEVICETYPE const type,DWORD const handle,LONG& range);

ส่วนของParameter ของ Device ที่สนับสนุน Force-Feedback จะมี Function ดังต่อไปนี้

int DI::Param::getFFSupport(DEVICETYPE const type,DWORD const handle);
ดูว่า Device นั้นๆ สนับสนุน Force-Feedback หรือไม่

ตั้งค่าให้ Device นั้นๆ อยู่ในสภาพต่างๆ เช่น
int DI::Param::setFFPause(DEVICETYPE const type,DWORD const handle);
หยุดการเล่น Effect ชั่วคราว
int DI::Param::setFFContinue(DEVICETYPE const type,DWORD const handle);
เล่น Effect ต่อไป
int DI::Param::setFFStop(DEVICETYPE const type,DWORD const handle);
หยุดการเล่น Effect โดยสิ้นเชิง
int DI::Param::setFFReset(DEVICETYPE const type,DWORD const handle);
ให้ Effect ที่เล่นอยู่ กลับไปจุดเริ่มต้น

สามารถตั้ง Actuators ให้อยู่ในสภาพติด-ดับ(คล้ายกับ Mute ในเสียง)ได้โดย
int DI::Param::setFFActuatorsON(DEVICETYPE const type,DWORD const handle);
int DI::Param::setFFActuatorsOFF(DEVICETYPE const type,DWORD const handle);
โดยเมื่ออยู่ในสภาวะ OFF ไม่ว่าจะเล่น Effect ใดๆก็จะมีผลต่อ Device จนกว่าจะทำการ ON

เราสามารถดูว่า Device นั้นๆ อยู่ในสภาวะนั้นๆหรือไม่ ก็สามารถทำได้โดย
int DI::Param::getFFPause(DEVICETYPE const type,DWORD const handle);
int DI::Param::getFFActuatorsON(DEVICETYPE const type,DWORD const handle);
int DI::Param::getFFActuatorsOFF(DEVICETYPE const type,DWORD const handle);

ส่วนของ Force-Feedback Effect

เป็นส่วนจัดการเกี่ยวกับ Force-Feedback Effect โดยเฉพาะ

เมื่อต้องการสร้าง Effect ต้องใช้ Function createFFEffect

int DI::FF::createFFEffect(DEVICETYPE const type,DWORD const handle,TCHAR* name,const TCHAR* filename);

โดยระบุชื่อ File .FFE และชื่อที่ใช้ในการอ้างอิง Effect นั้นๆ และระบุ Device ที่จะเชื่อมต่อด้วย โดยการสร้าง Effect นั้น จะต้องสร้างโดย Device ที่สนับสนุน Force-Feedback เท่านั้น

และสามารถทำลาย Effect ที่สร้างขึ้นมาได้โดยการระบุชื่อ

int DI::FF::destroyFFEffect(TCHAR* name);

หรือหากต้องการจะทำลาน Effect ทั้งหมด ก็สามารถทำได้โดย

int DI::FF::clearFFEffect();

เราสามารถสั่งการ Effect ต่างๆโดยการระบุชื่อของ Effect นั้นๆ โดยมี Function ดังนี้

int DI::FF::playFFEffect(TCHAR* name);

เล่น Effect ไปยัง Device

int DI::FF::setloopFFEffect(TCHAR* name);

int DI::FF::setloopFFEffect(TCHAR* name,DWORD const dloop);

int DI::FF::resetloopFFEffect(TCHAR* name);

ตั้งค่าการวนรอบของการเล่น Effect โดยหากไม่ระบุจำนวนจะเป็นการวน Infinite และเมื่อทำการ Reset จะเป็นการตั้งค่าให้เล่นแค่รอบเดียว

หากต้องการรู้ว่า Effect นั้นๆ ทำการเล่นอยู่หรือเปล่าก็สามารถตรวจได้โดย

int DI::FF::isPlayFFEffect(TCHAR* name);

หรือต้องการดูว่า Effect นั้นๆ มีตัวตนหรือไม่ ก็ใช้ Function นี้

int DI::FF::checkFFEffect(TCHAR* name);

Function ที่สั่งการ Effect ต่างๆสามารถไม่ใส่ชื่อของ Effect ได้ดังเช่น

int DI::FF::playFFEffect();

int DI::FF::setloopFFEffect();

int DI::FF::setloopFFEffect(DWORD const dloop);

int DI::FF::resetloopFFEffect();

int DI::FF::isPlayFFEffect();

โดยต้องระบุชื่อ Current Effect ก่อน Function ที่กล่าวไว้ด้านบนจะทำการสั่งการ Current Effect ที่ตั้งเอาไว้ และเราก็สามารถดูชื่อ Current Effect ได้ด้วย

int DI::FF::setCurrentFFEffect(TCHAR* name);

int DI::FF::getCurrentFFEffect(TCHAR* name);

ส่วน Effect Agent สามารถสร้างได้จาก Effect ที่สร้างขึ้นแล้ว โดย

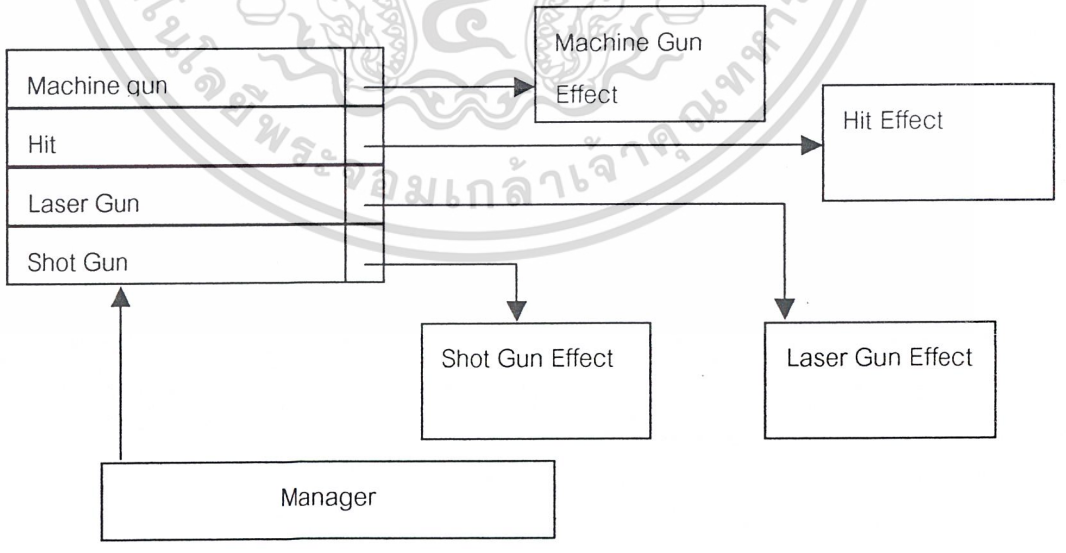
CIEffAgent* DI::FF::cloneFFEffectAgent(TCHAR* name);

Effect Agent(CIEffAgent)

Agent นั้นเสมือนเป็นตัวแทนของ Effect Object หมายความมีความสามารถทำได้ทุกอย่างเสมือนอย่างที่ Effect Object ทำได้

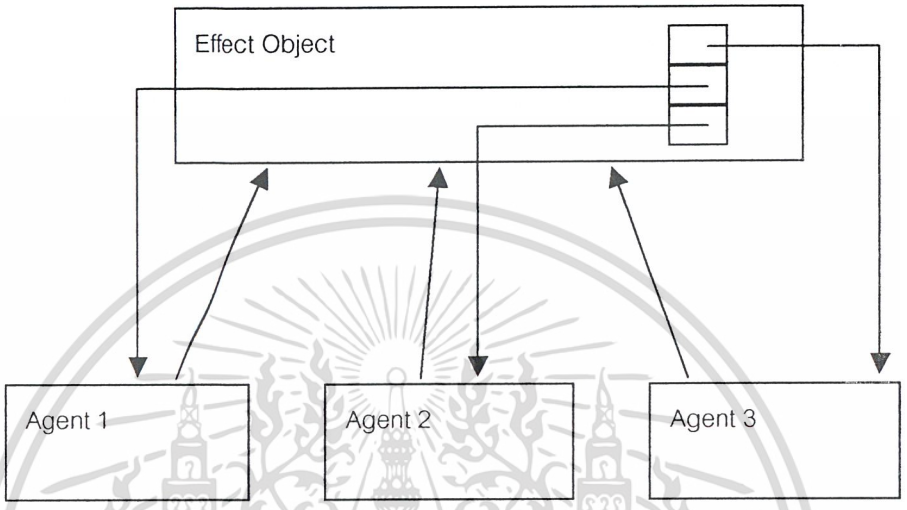
Effect Agent นั้นสามารถคัดลอกได้ หมายความว่า Effect Agent สามารถมีได้หลายตัว ในขณะที่ Effect Object มีได้เพียงตัวเดียว ซึ่งการกระทำใดๆต่อตัว Effect Agent ตัวหนึ่งจะกระทบต่อ Effect Object ด้วย และการกระทำใดๆต่อ Effect Object ก็จะมีผลต่อ Effect Agent ทุกตัวเช่นเดียวกัน

โครงสร้างของ Effect Object นั้นจะมีตารางอ้างอิง ซึ่งเก็บชื่อและ Pointer ของ Effect Object ดังนั้น การติดต่อกับ Effect Object ผ่านทาง Manager จะต้องระบุชื่อของ Effect นั้นๆด้วย



แต่ Effect Agent นั้นจะเก็บ Pointer ของ Effect Object อยู่ และ Effect Object ก็จะเก็บ Pointer ของ Effect Agent เช่นเดียวกัน ดังนั้นการติดต่อ Effect Object ผ่านทาง Effect Agent จะรวดเร็วกว่า

Effect Object จะเก็บ Pointer ของ Effect Agent ที่อ้างอิงต่อตัวมันเอง ซึ่งเก็บเป็นตารางอ้างอิง ทำให้การติดต่อระหว่าง Effect Object กับ Effect Agent เป็น Two-way pointer



สำหรับ Function ต่างๆของ CIEffAgent จะมีดังนี้

int play();

เล่น Effect

int setloop();

int setloop(DWORD const dloop);

int resetloop();

ตั้งค่า Loop ในการเล่น

int isPlay();

ตรวจสอบว่าเล่น Effect อยู่หรือไม่

int dupEffAgent(CIEffAgent& agent);

คัดลอก Agent โดย CIEffAgent ที่คัดลอกจะเหมือนกับต้นฉบับทุกอย่าง

int getName(TCHAR* name);

ดูชื่อ Effect Object ที่ Agent ติดต่ออยู่

int settoCIEffect(TCHAR* name);

เปลี่ยน Effect ที่ติดต่ออยู่

void cleanup();

ยกเลิกการใช้ Agent โดยต้องทำการ Cleanup ก่อนการยกเลิกการใช้งาน

ภาคผนวก ญ
วิธีใช้งานมิเดีย

ค่าที่ Return จาก Function นั้นจะใช้ค่า integer มีอยู่สามค่าคือ

0: ค่าเป็น True ไม่มีข้อผิดพลาดและ

1: ค่าเป็น False และไม่มีข้อผิดพลาด,

-1: มีข้อผิดพลาดเกิดขึ้น ไม่อาจจะจะทำกระบวนให้สำเร็จได้

เหมือนกับ Version 1 ซึ่งจะต้อง Initialize COM ก่อน โดย

CoInitialize(NULL);

และเริ่มต้นการใช้งานโดย Initialize Media API โดยค่า hwnd คือ Handle window ของ Application ที่ใช้อยู่

int DM::init(HWND const hwnd);

และเมื่อต้องการสิ้นสุดการทำงาน Media API ก็ทำการ Cleanup โดย

void DM::cleanup();

และอย่าลืมเมื่อสิ้นสุดการใช้ COM ก็ต้อง Uninitialize ด้วยคำสั่ง

CoUninitialize();

การบริหารจัดการตาราง

ในส่วนของ Table (DM::Table) เป็นส่วนที่เก็บข้อมูลของ Media ซึ่งผู้ใช้สามารถ Define เองได้ โดยจะเก็บข้อมูลเป็น Struct MEDIA_ROW โดยจะยังไม่ได้ทำการ load ขึ้นมาแต่อย่างใด

โดย Structure การเก็บจะเป็นลักษณะตาราง โดยอ้างอิงแต่ละ Row โดยใช้ค่า Handle และมีข้อมูลเป็น MEDIA_ROW มีลักษณะดังนี้

```
struct MEDIA_ROW
{
TCHAR filename[MAX_PATH];
LONG volume;
double rate;
LONGLONG start;
LONGLONG end;
WINDOW_POS pos;
};
```

filename เป็นชื่อ file ของ Media ซึ่งสนับสนุน MPEG, AVI, QuickTime, WAV,AIFF,AU,SND และ MIDI

volume เป็นความดังของเสียง มีค่าระหว่าง 0-100

rate คือค่าอัตราการเล่นของเสียง ปกติจะมีค่าเป็น 1.0

start,end จุดเริ่มและจุดสิ้นสุด หน่วยเป็น 100 nanoseconds

pos เป็นค่าความกว้างยาวและตำแหน่งของ Window ที่แสดงผล โดยมี Struct เป็น WINDOW_POS ดังนี้

```
struct WINDOW_POS
{
        long left;
        long top;
        long width;
        long height;
};
```

การบริหารจัดการกับ Media ก็สามารถทำได้ เช่น

```
int DM::Table ::add(MEDIA_ROW const row);
```

เพิ่มข้อมูลลงตาราง โดยกำหนดของข้อมูลเอง โดยจะได้ค่า Handle number กลับไป

```
int DM::Table ::add(TCHAR* filename);
```

เพิ่มข้อมูลลงตาราง โดยระบุชื่อ file เท่านั้น โดยจะได้ค่า Handle number กลับไป

```
HRESULT DM::Table ::set(int const handle,MEDIA_ROW const row);
```

เปลี่ยนแปลงข้อมูลของ MEDIA_ROW

```
HRESULT DM::Table ::get(int const handle,MEDIA_ROW& row);
```

ดูข้อมูลของ MEDIA_ROW นั้นๆ

```
HRESULT DM::Table ::erase(int const handle);
```

ลบข้อมูล MEDIA_ROW ที่ต้องการ

การเล่น Media ต่างๆ

ในการเล่น Media จะต้องทำการ Load Media ขึ้นมาจากตาราง Media Table ก่อน โดยจะได้ handle number ของส่วนการตารางการเล่น(Playing Table) ซึ่งโดยไม่เกี่ยวข้องกับ Media Table แต่อย่างใด

เมื่อต้องการ Load Media จากตาราง Media Table ขึ้นมาเล่น โดยจะได้ค่า Handle number อีกตารางการเล่นใช้ในการอ้างอิง

int DM::load(int const loadhd,int& playhd);

ในการสั่งการ Media ที่ทำการ load ขึ้นมาจะต้องระบุ handle number ที่ได้จากการ load ก่อน โดยเป็นการระบุ Current Media โดย

int setCurrent(int const playhd);

ระบุ Media ที่จะสั่งการ

หากต้องการดูว่าค่า Current เป็นค่าใดก็สามารถดูได้โดย

int getCurrent(int& playhd);

ส่วน Function อื่นๆ จะเป็นการสั่งการ Media ที่ถูกระบุโดย setCurrent

int DM::play();

int DM::pause();

int DM::stop();

เล่น, หยุดชั่วคราว , หยุด

int DM::isPlay();

int DM::isPause();

int DM::isStop();

ตรวจสอบสถานะปัจจุบัน

int DM::getCurrentPosition(LONGLONG& pos);

ดูตำแหน่งการเล่นปัจจุบัน มีหน่วยเป็น 100 nanosecond

int DM::getStopPosition(LONGLONG& pos);

ดูตำแหน่งที่จะทำการหยุดเล่นเล่น มีหน่วยเป็น 100 nanosecond

int DM::setPlayRate(double const rate);

int DM::getPlayRate(double& rate);

ตั้งและดูอัตราการเล่น

int DM::setEnd(LONGLONG const end);

int DM::getEnd(LONGLONG& end);

ตั้งและดูจุดสิ้นสุดการเล่น (ไม่สามารถตั้งค่าขณะทำการ Play อยู่ได้)

int DM::setStart(LONGLONG const start);

int DM::getStart(LONGLONG& start);

ตั้งและดูจุดเริ่มต้นการเล่น (ไม่สามารถตั้งค่าขณะทำการ Play อยู่ได้)

int DM::setVolume(LONG const volume);

int DM::getVolume(LONG& volume);

ตั้งและรับค่าความดัง Volume

int DM::setWindowPosition(WINDOW_POS const pos);

int DM::getWindowPosition(WINDOW_POS& pos);

ตั้งและรับค่าตำแหน่งและขนาดของ Window ที่แสดงผล

int DM::setForeground();

ตั้งค่าให้ Media นั้นๆ อยู่บนสุดในการแสดงผล เมื่อมีการแสดงผลทับซ้อนกัน

เมื่อต้องการ Unload Media ก็สามารถโดยระบุ Handle number ที่ได้มาจากการ Load

int DM::unload(int const playhd);

บรรณานุกรม

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

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

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

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

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