การพัฒนาเกมโดยใช้ไดเร็กเอ็กซ์
GAME DEVELOPMENT USING DIRECTX

โดย
นายทิวากร โกมุท
นายเทวินทร์ ทัศนเจริญ

อาจารย์ที่ปรึกษา
ดร. วรวัฒน์ ลิ้มโภคา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2542

ปริญญานิพนธ์ปีการศึกษา 2542

ภาควิชา วิศวกรรมคอมพิวเตอร์

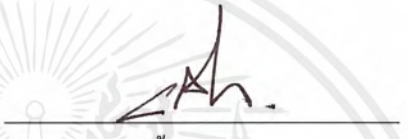
คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาเกมโดยใช้ไดเร็กเอ็กซ์

GAME DEVELOPMENT USING DIRECTX

ผู้จัดทำ

1. นาย ทิวากร โกมุท รหัสประจำตัว 39014200
2. นาย เทวินทร์ ทัศนเจริญ รหัสประจำตัว 39014203



อาจารย์ที่ปรึกษา

(ดร. วรวัฒน์ ลิ้มโภคา)

การพัฒนาเกมโดยใช้ไดเร็กเอ็กซ์

นายทิวากร โกมุท 39014200
นายเทวินทร์ หัสนเจริญ 39014203
ดร. วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษา
ปีการศึกษา 2542

บทคัดย่อ

การพัฒนาเกมโดยใช้ไดเร็กเอ็กซ์ เป็นโครงงานที่กล่าวถึง การพัฒนาเกมโดยใช้ ไดเร็กเอกซ์ เป็นตัวช่วยเหลือในการพัฒนาโปรแกรมด้านมัลติมีเดีย, ไดเร็กเอ็กซ์ เป็นเทคโนโลยีที่ถูกสร้างขึ้นมา เพื่อช่วยให้ผู้พัฒนา และผู้ใช้โปรแกรมสามารถได้ประโยชน์สูงสุดจาก อุปกรณ์ฮาร์ดแวร์ต่างๆ โดยไม่สูญเสียประโยชน์ในการเข้าถึงฮาร์ดแวร์ได้โดยตรง

ขอบเขตการทำงานของโครงงานนี้: ในขั้นแรก, จะทำการศึกษาถึงสถาปัตยกรรมของไดเร็กเอ็กซ์ และศึกษาถึงความสัมพันธ์ระหว่าง ไดเร็กเอ็กซ์ กับฮาร์ดแวร์ ซึ่งได้แก่ การ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ และหน่วยประมวลผล จากนั้นจึงทำการพัฒนาโปรแกรมโดยใช้ไดเร็กเอ็กซ์, โดยเริ่มต้นจากการใช้ ไดเร็กดรอว์(2 มิติ) และส่วนประกอบต่างๆ ของไดเร็กเอ็กซ์

ในขั้นที่สอง, จะสร้างโปรแกรมที่ใช้ในการออกแบบแผนที่ จัดหาวิธีในการตรวจสอบการชนกันของวัตถุสำหรับฉากต่างๆ สร้างโปรแกรมที่ดึงเอาโมเดล 3 มิติ และเสียงต่างๆ เข้าไปในเกม สร้างโปรแกรมเพื่อรองรับอุปกรณ์อินพุตต่างๆ รวมทั้ง สร้างโมเดล 3 มิติ และ จัดหาข้อมูลด้านเสียงสำหรับเกม จากนั้นจึงทำการรวมโปรแกรมทั้งหมด และ ตรวจสอบโปรแกรม โดยการเขียนโปรแกรมทั้งหมดจะเขียนในเชิงวัตถุ

Game Development Using DirectX

Tiwakorn Komut

Tawin Tusnajareon

Dr. Worawat Limpoka Advisor

ABSTRACT

Game Development Using DirectX is the project that describes about game programming that used by DirectX Application Programming Interface (API), DirectX is technology that produced to service for user programming and developer programming to get the benefits of device independence without losing the benefits of direct access to the hardware.

This project covers: In First step, Begins with studying architecture of DirectX and studying relations between DirectX and hardware such as accelerating card-3D and CPU. After that researcher will develop program with DirectX, starts with using DirectDraw(2D) and other components.

In Second step, Begins with program that helps to design map in game and then develops program for check collision in game and develops program that gets model 3D and sound files in game. And develop program that supports input device including builds model 3D and collects sound files. After that is assemble all programs into one program and testing. All of programming will program in Object Oriented Programming.

สารบัญ

สารบัญตาราง

หน้าที่

สารบัญภาพ

หน้าที่

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

เกมเป็นแอปพลิเคชั่นที่ถูกสร้างขึ้นเพื่อความบันเทิง ซึ่งเครื่องคอมพิวเตอร์ส่วนบุคคลนั้นก็มีแอปพลิเคชั่นทางด้านเกมอยู่มากมายมาย ตั้งแต่สมัยที่เครื่องคอมพิวเตอร์ส่วนบุคคลยังใช้ระบบปฏิบัติการ MS-DOS จนมาถึงปัจจุบันบนระบบปฏิบัติการ Windows ซึ่งเกมนั้นถือได้ว่าเป็นแอปพลิเคชั่นที่ต้องการประสิทธิภาพการประมวลผลที่สูง โดยการพัฒนาของเครื่องคอมพิวเตอร์ส่วนบุคคลส่วนหนึ่งก็ได้รับแรงกระตุ้นจากแอปพลิเคชั่นทางด้านเกม ตามความเป็นจริงแล้วแอปพลิเคชั่นทางด้านเกมถือได้ว่าเป็นแอปพลิเคชั่นทางด้านมัลติมีเดียอย่างหนึ่งเพราะมีการประมวลผลทั้งทางด้านภาพ เสียง และ ส่วนที่ติดต่อกับผู้ใช้งาน

ในอดีตนั้นเกมถูกพัฒนาบนระบบปฏิบัติการ MS-DOS ซึ่งมีข้อดีคือผู้พัฒนานั้นสามารถที่จะเข้าถึงการทำงานของฮาร์ดแวร์ในระดับต่ำ (low-level) ได้โดยตรง ซึ่งทำให้แอปพลิเคชั่นมีประสิทธิภาพการทำงานที่สูง แต่มีข้อเสียคือ ผู้พัฒนาแอปพลิเคชั่นต้องพัฒนาให้แอปพลิเคชั่นของตนสามารถรองรับการทำงานกับอุปกรณ์ของบริษัทต่างๆที่มีอยู่มากมายอยู่ในท้องตลาด เช่น การ์ดแสดงผล และ การ์ดเสียง ที่มีอยู่มากมายหลายยี่ห้อเพื่อให้ครอบคลุมกลุ่มผู้ใช้งานให้ได้มากที่สุด ซึ่งเป็นงานที่ยากลำบากและสิ้นเปลืองงบประมาณเป็นอย่างมาก

ต่อมาบริษัทไมโครซอฟต์ได้เปิดตัวระบบปฏิบัติการ Windows ออกมาซึ่งระบบปฏิบัติการ Windows นี้เป็นระบบปฏิบัติการแบบ GUI (Graphical User Interface) มีข้อดีคือผู้พัฒนาแอปพลิเคชั่นไม่จำเป็นต้องพัฒนาแอปพลิเคชั่นของตนให้สนับสนุนอุปกรณ์ของบริษัทต่างๆอีกต่อไปเพราะเมื่ออุปกรณ์เหล่านั้นถูกพัฒนามาเพื่อใช้งานกับระบบปฏิบัติการ Windows แล้วผู้พัฒนาแอปพลิเคชั่นเพียงแต่พัฒนาโดยยึดรูปแบบมาตรฐานของระบบปฏิบัติการ Windows ก็เพียงพอ ซึ่งเป็นคุณสมบัติแบบ "device-independent" แต่ระบบปฏิบัติการ Windows นั้นเป็นระบบปฏิบัติการที่มีการแสดงผลด้านกราฟิกที่ช้า ดังนั้นในยุคแรกๆของระบบปฏิบัติการ Windows ผู้พัฒนาแอปพลิเคชั่นทางด้านเกมจึงยังคงยึดติดอยู่กับระบบปฏิบัติการ MS-DOS อยู่ ซึ่งแอปพลิเคชั่นทางด้านเกมที่ทำงานอยู่บนระบบปฏิบัติการ Windows นั้นก็พอมีอยู่บ้างแต่จะเป็นพวกที่ไม่ต้องการการแสดงผลทางด้านกราฟฟิกที่รวดเร็ว เช่น เกมหมากกระดาน และ เกมแนวผจญภัย

ทางบริษัทไมโครซอฟต์ได้เล็งเห็นว่าแอปพลิเคชั่นทางด้านเกมนั้นมีผู้ใช้งานกันอย่างแพร่หลาย จึงมีแนวความคิดที่จะให้ผู้พัฒนาแอปพลิเคชั่นทางด้านเกมและมัลติมีเดียหันมาพัฒนาบนระบบปฏิบัติการ Windows ดังนั้นบริษัทไมโครซอฟต์จึงพัฒนา DirectX ขึ้นมา ซึ่งเป็น libraries ทางด้านมัลติมีเดียที่มีจุดมุ่งหมายหลักๆดังต่อไปนี้

- DirectX ประกอบด้วย libraries ที่ทำงานในระดับต่ำ ซึ่งทำงานได้รวดเร็วและไม่มีข้อจำกัดในการพัฒนาแอปพลิเคชั่นทางด้านเกม

- โครงสร้างของ DirectX ต้องยกภาระในเรื่องฮาร์ดแวร์จากผู้พัฒนาแอปพลิเคชั่นไปสู่ผู้ผลิตฮาร์ดแวร์ ซึ่งผู้ผลิตฮาร์ดแวร์ต้องเป็นผู้สร้าง drivers สำหรับผลิตภัณฑ์ของตน และให้ผู้พัฒนาแอปพลิเคชั่นนั้นสามารถใช้ความสามารถล่าสุดที่มีอยู่ในฮาร์ดแวร์นั้นๆได้
- DirectX นั้นต้องอนุญาตให้ผู้พัฒนาแอปพลิเคชั่นสามารถพัฒนาแอปพลิเคชั่นออกมาในรูปแบบของ Windows application ที่สามารถทำงานบน Desktop ได้และสามารถทำงานร่วมกับฟังก์ชันต่างๆที่มีอยู่บนระบบปฏิบัติการ Windows ได้
- แอปพลิเคชั่นที่ถูกพัฒนาโดยใช้ DirectX นั้นต้องมีประสิทธิภาพที่อยู่สูงกว่าหรืออย่างน้อยที่สุดต้องเทียบเท่ากับประสิทธิภาพของแอปพลิเคชั่นที่ทำงานบนระบบปฏิบัติการ MS-DOS

ซึ่งในปัจจุบันแอปพลิเคชั่นทางด้านเกมและมัลติมีเดียได้หันมาพัฒนาบนระบบปฏิบัติการ Windows กันหมดแล้วเพราะ DirectX นั้นทำให้แอปพลิเคชั่นมีประสิทธิภาพมาก เช่น สามารถใช้ความสามารถของการ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ ที่ติดตั้งอยู่บนเครื่องคอมพิวเตอร์และชุดคำสั่งพิเศษของไมโครโปรเซสเซอร์ได้ อีกทั้งยังช่วยให้ผู้พัฒนาแอปพลิเคชั่นไม่ต้องมายุ่งเกี่ยวในส่วนของฮาร์ดแวร์ที่มีอยู่มากมายหลายยี่ห้ออีกต่อไป โดยที่ยังสามารถเข้าถึงการทำงานในระดับต่ำของฮาร์ดแวร์ได้

1.2 จุดประสงค์ของโครงงาน

1.2.1 ศึกษาโครงสร้างและส่วนประกอบต่างๆของ DirectX libraries ว่ามีส่วนประกอบอะไรบ้างและแต่ละส่วนนั้นมีการทำงานอย่างไร

1.2.2 พัฒนาแอปพลิเคชั่นทางด้านเกมในรูปแบบ 3 มิติ โดยใช้ส่วนประกอบต่างๆของ DirectX libraries

1.3 ขอบเขตของโครงงาน

โครงงานนี้เป็นการสร้างเกมในรูปแบบ 3 มิติ ในมุมมองของบุคคลที่หนึ่ง (first-person-perspective 3D game) ที่สามารถควบคุมได้ทั้งจาก คีย์บอร์ด, เมาส์, และ จอยสติ๊ก และมีเสียงเพลงประกอบขณะเล่นเกม โดยเกมที่สร้างออกมานั้นก็เพื่อทดสอบและทดลองว่า DirectX libraries นั้นมีคุณสมบัติและประสิทธิภาพการทำงานเป็นอย่างไรโดยไม่ได้เน้นในด้านความสนุกของเกม DirectX libraries นั้นมีส่วนประกอบหลายอย่าง แต่ส่วนที่จะนำมาใช้ในโครงงานนี้ได้แก่ DirectDraw ซึ่งเป็นส่วนพื้นฐานในการจัดการด้านกราฟฟิก Direct3D Retained-Mode เป็นส่วนที่จัดการเกี่ยวกับการแสดงผลกราฟิกในรูปแบบ 3 มิติ DirectInput เป็นส่วนที่จัดการเกี่ยวกับอินพุตจากผู้ใช้ และ DirectSound จัดการเกี่ยวกับเสียง โดยการพัฒนาแอปพลิเคชั่นทั้งหมดจะทำในรูปแบบของการเขียนโปรแกรมเชิงวัตถุ (Object-Oriented Programming)

1.4 วิธีดำเนินงาน

โครงงานนี้เริ่มต้นด้วยการศึกษาถึงประโยชน์ และ ส่วนประกอบต่างๆของ DirectX libraries และการพัฒนาแอปพลิเคชั่นโดยใช้ DirectX libraries กับคอมไพเลอร์ Visual C++ ดังในบทที่ 2

จากนั้นได้ศึกษา DirectDraw ซึ่งเป็นส่วนประกอบของ DirectX ที่ทำหน้าที่เกี่ยวกับการแสดงผลทางด้านกราฟิกแบบ 2 มิติ DirectInput เป็นส่วนประกอบที่จัดการกับอินพุต และ DirectSound จัดการเกี่ยวกับเสียง ดังในบทที่ 3, 4, และ 5 ตามลำดับ ต่อมาได้ศึกษาระบบกราฟฟิกแบบ 3 มิติ และ Direct3D Retained-Mode ซึ่งเป็นส่วนที่สำคัญที่สุดของโครงงาน ดังในบทที่ 6 จากนั้นได้ศึกษาเรื่องความสัมพันธ์ระหว่าง DirectX, การ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ, และ CPU ว่ามีการทำงานร่วมกันอย่างไร ดังในบทที่ 7 และศึกษาในเรื่องของ DirectSetup ดังในบทที่ 8

ต่อมาได้ศึกษาถึงการสร้างโมเดล 3 มิติ และการนำโมเดล 3 มิติ มาใช้ในแอปพลิเคชั่นร่วมกับ Direct3D Retained-Mode ดังในบทที่ 9

บทที่ 10 จะเป็นการนำความรู้ที่ได้ศึกษามาทั้งหมดนั้นมาออกแบบแอปพลิเคชั่นทางด้านเกม ซึ่งการออกแบบและพัฒนาจะทำในรูปแบบเชิงวัตถุ ซึ่งทำให้การพัฒนาแอปพลิเคชั่นเป็นสัดส่วนและมีลำดับขั้นตอน

บทที่ 11 เป็นผลลัพธ์ของโครงงาน คือ รูปแบบของเกมที่ได้ อธิบายส่วนประกอบต่างๆภายในเกม วิธีเล่นเกม และฟังก์ชันการทำงานต่างๆภายในเกม

สุดท้าย บทที่ 12 เป็นการสรุปผล และแนวทางในการพัฒนาโครงงานต่อ

บทที่ 2
แนะนำไดเร็กเอ็กซ์

2.1 ไดเร็กเอ็กซ์คืออะไร

ไมโครซอฟต์ไดเร็กเอ็กซ์ (Microsoft DirectX) คือ กลุ่มของแอปพลิเคชั่นโปรแกรมมิ่งอินเตอร์เฟส (Application Programming Interfaces) หรือ เอพีไอ (APIs) ที่ช่วยสนับสนุนการสร้างแอปพลิเคชั่นแบบเรียลไทม์ (Real-Time Applications) ด้าน เกม และ มัลติมีเดียที่ต้องการประสิทธิภาพการทำงานสูง

2.2 จุดประสงค์ของไดเร็กเอ็กซ์

ไมโครซอฟต์พัฒนาไดเร็กเอ็กซ์เพื่อ เพิ่มประสิทธิภาพการทำงานด้านมัลติมีเดียของแอปพลิเคชั่นที่ทำงานอยู่บนระบบปฏิบัติการไมโครซอฟต์วินโดวส์ (Microsoft Windows) ให้มีความสามารถเทียบเท่าหรือสูงกว่าประสิทธิภาพของแอปพลิเคชั่นที่ทำงานอยู่บนระบบปฏิบัติการ MS-DOS หรือเครื่องเกมประเภทคอนโซล (Game Consoles)

2.3 ประโยชน์ของการพัฒนาแอปพลิเคชั่นโดยใช้ไดเร็กเอ็กซ์

ไมโครซอฟต์พัฒนาไดเร็กเอ็กซ์ขึ้นมาโดยมีจุดประสงค์หลักคือเพื่อสนับสนุนการพัฒนาแอปพลิเคชั่นทางด้านเกมบนระบบปฏิบัติการไมโครซอฟต์วินโดวส์ ก่อนที่จะมีไดเร็กเอ็กซ์นั้นการพัฒนาแอปพลิเคชั่นทางด้านเกมสำหรับเครื่องคอมพิวเตอร์ส่วนบุคคลนั้นจะกระทำบนระบบปฏิบัติการ MS-DOS ซึ่งผู้ที่ทำการพัฒนาแอปพลิเคชั่นเกมต้องทำการพัฒนาแอปพลิเคชั่นให้สามารถใช้ได้กับอุปกรณ์ยี่ห้อต่างๆที่อยู่มากมายในท้องตลาด ส่วนการพัฒนาแอปพลิเคชั่นโดยใช้ไดเร็กเอ็กซ์นั้นผู้พัฒนาจะได้รับประโยชน์จากดีไวซ์อินดีเพนเดนท์ (device-independent) แต่ยังคงคุณสมบัติไดเร็กแอ็คเซส (Direct Access) ของอุปกรณ์ไว้ ซึ่งหมายความว่าเราสามารถพัฒนาแอปพลิเคชั่นโดยได้ไม่ต้องคำนึงถึงว่าจะมีอุปกรณ์อยู่มากมายหลายยี่ห้อในท้องตลาด เพราะแอปพลิเคชั่นของเราจะสามารถทำงานได้กับอุปกรณ์ทุกยี่ห้อที่สนับสนุนไดเร็กเอ็กซ์ และจะสามารถเข้าถึงการทำงานในระดับต่ำ (low-level) ของอุปกรณ์นั้นๆได้ อีกทั้งยังสามารถใช้ความสามารถจากการเร่งความเร็วโดยใช้ฮาร์ดแวร์ที่มีอยู่บนอุปกรณ์ต่างๆ เช่น การ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ และ ชุดคำสั่งพิเศษของไมโครโพรเซสเซอร์ เช่น ชุดคำสั่ง MMX, 3D NOW!, SSE ได้อีกด้วย

ไดเร็กเอ็กซ์จะให้ประโยชน์ใน 2 ด้านคือ

1. ประโยชน์ในด้านการพัฒนา DirectX Windows Application
2. เป็นมาตรฐานสำหรับการพัฒนาฮาร์ดแวร์

2.4 ส่วนประกอบของ DirectX

ในส่วนนี้จะให้รายละเอียดของส่วนประกอบต่างๆของ DirectX อย่างคร่าวๆ ส่วนประกอบของ DirectX ได้แก่

- **DirectDraw** จะเป็นเทคนิคการเร่ง ฮาร์ดแวร์ และซอฟต์แวร์แอนิเมชัน โดยการเข้าถึงรูปภาพ bitmap ในหน่วยความจำ(ไม่ต้องผ่านระบบปฏิบัติการ Windows) ทำให้แอปพลิเคชั่นของเราต้องการเพียงการจัดการฮาร์ดแวร์พื้นฐานง่ายๆ เราไม่ต้องเรียก procedure ที่เจาะจงสำหรับจัดการกับสีต่างๆ รวมทั้งรองรับการจัดการกับ palettes, clipping และ animation การใช้ DirectDraw ทำให้จัดการกับหน่วยความจำได้ง่ายขึ้น

  DirectDraw จะช่วยผู้เขียนโปรแกรมดังนี้

  - Hardware abstraction layer (HAL) ของ DirectDraw จะติดต่อโดยรงกับฮาร์ดแวร์โดยตรง ทำให้ฮาร์ดแวร์สามารถแสดงผลที่สูงสุดได้
  - DirectDraw จะประเมินความสามารถของฮาร์ดแวร์ของเรา และใช้งานให้มีประสิทธิภาพสูงสุด นอกจากนั้น DirectDraw ยังมี Hardware emulation layer (HEL) ที่รองรับการทำงานเมื่อไม่มีฮาร์ดแวร์มารองรับ
  - DirectDraw อยู่ภายใต้ระบบปฏิบัติการ Windows ดังนั้นจะได้รับข้อดีในการอ้างอิงหน่วยความจำ 32-bit และ flat memory model ที่ระบบปฏิบัติการจัดหาให้
    DirectDraw แสดงหน่วยความจำวีดีโอ (video memory) และหน่วยความจำระบบ (system memory) เป็นบล็อกขนาดใหญ่ ไม่ใช่เป็น segment เล็กๆ ที่ใช้อ้างอิงหน่วยความจำเป็น segment:offset เหมือนสมัยก่อน
  - รองรับแอปพลิเคชั่นทั้งแบบเต็มหน้าจอ (full-screen mode) และไม่เต็มหน้าจอ (windowed mode)
  - รองรับ 3D Z-buffers
  - รองรับการช่วยเหลือจากฮาร์ดแวร์ด้วย Z-Ordering
  - เข้าถึงหน่วยความจำมาตรฐาน และหน่วยความจำของส่วนแสดงผลได้พร้อมกัน

- **DirectSound** ทำให้ใช้ฮาร์ดแวร์ และซอฟต์แวร์ทางด้านเสียง(การ mix และเล่น)ได้ และเข้าถึงฮาร์ดแวร์ด้านเสียงได้โดยตรง DirectSound จะทำให้การทำงานของ drivers ต่างๆ สามารถเข้ากันได้
  DirectSound สามารถรับข้อมูลเสียง และเล่นได้ DirectSound ยังรองรับคุณสมบัติต่างๆที่จะทำให้ผู้พัฒนาโปรแกรมนำข้อดีต่างๆที่เพิ่มขึ้นมาจากการ์ดเสียง และ drivers ต่างๆที่เกี่ยวข้อง

  DirectSound ยังช่วยให้ทำงานง่ายขึ้นเช่น

  - สามารถตอบฮาร์ดแวร์ขณะที่ทำงานอยู่ เพื่อตัดสินหาวิธีการตั้งค่าต่างๆ ที่ดีที่สุดสำหรับเครื่องคอมพิวเตอร์ส่วนบุคคลแต่ละเครื่อง
  - สามารถใช้คุณสมบัติต่างๆ ของฮาร์ดแวร์ใหม่ แม้ว่ามันจะไม่ได้รองรับ DirectSound โดยตรง

- จัดการกับเสียงในระบบ 3 มิติได้
- รับข้อมูลเสียงได้

- **DirectMusic** เป็นอุปกรณ์ทางด้านเสียงเพลงของ DirectX ไม่เหมือนกับ DirectSound ตรงที่ DirectMusic จะเล่นข้อมูลที่เป็นเพลง

  DirectMusic รองรับมาตรฐาน Musical Instrument Digital Interface (MIDI) และ downloadable sounds (DLS), DirectMusic จะมี tools สำหรับแต่ง และเล่นเพลง

- **DirectPlay** ทำให้การสร้างเกมระบบผู้เล่นหลายๆ คนง่ายขึ้น และสะดวกในการใช้งานร่วมกับโมเด็ม DirectPlay จะจัดหาการติดต่อสื่อสารที่มีประสิทธิภาพให้กับแอปพลิเคชั่นทำให้ผู้พัฒนาแอปพลิเคชั่นไม่ต้องมาพะวงกับการเชื่อมต่อโพรโตคอลต่างๆ เข้าด้วยกัน ทำให้พัฒนาแอปพลิเคชั่นของตนได้ง่ายขึ้น

- **Direct3D** ใช้ในการสร้างเกมในรูปแบบ 3 มิติ บนระบบปฏิบัติการ Windows ซึ่งได้เปรียบจากการที่สามารถใช้งานร่วมกับการ์ดเร่งความเร็วกราฟิกแบบ 3 มิติ ได้อย่างมีประสิทธิภาพเพราะมีฟังก์ชันคอยสนับสนุนนั่นเอง นอกจากนั้นสามารถรองรับการทำงานบนระบบปฏิบัติการ Windows NT ได้ และใช้เทคโนโลยี MMX ได้โดยตรง Direct3D มี 2 ส่วนประกอบ คือ Direct3D Retained-Mode API และ Direct3D Immediate Mode API

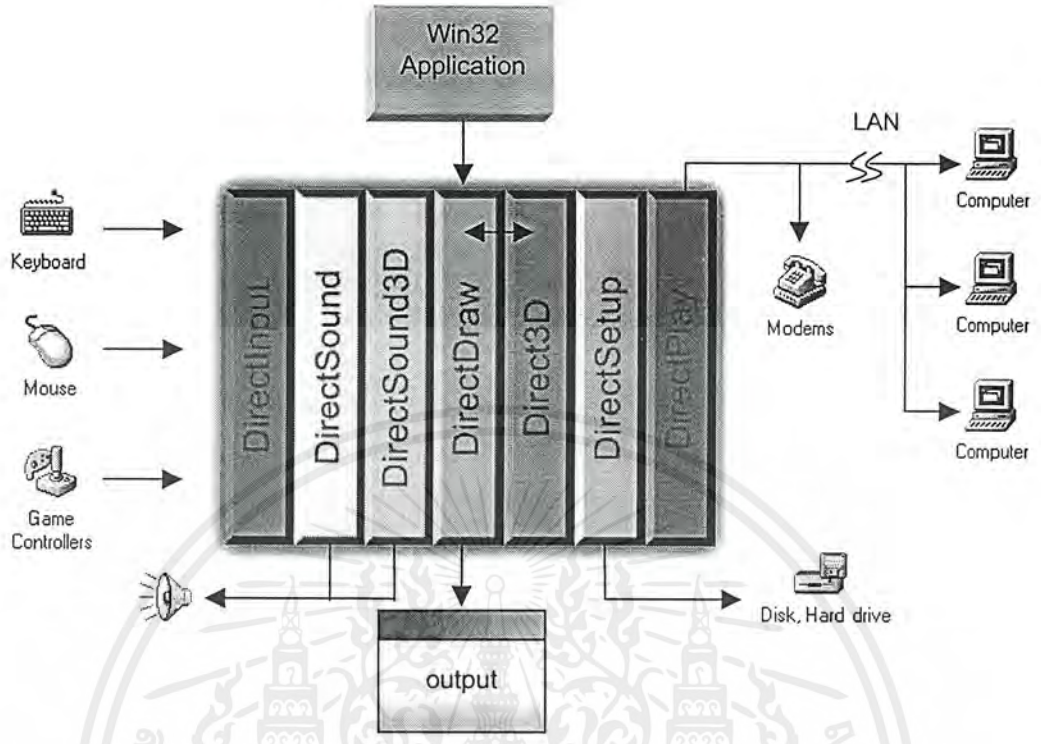
  ผู้พัฒนาส่วนมากจะใช้ Immediate Mode แทน Retained-Mode สำหรับโปรแกรมที่ต้องการความละเอียดทางกราฟิกสูง

- **DirectInput** ทำให้การสร้างเกมเพื่อรองรับระบบการสั่งงานจากผู้เล่นไม่ว่าจะเป็นจอยสติ๊ก เมาส์ หรือคีย์บอร์ด ทำได้ง่ายและมีประสิทธิภาพเป็นอย่างยิ่ง อีกทั้งยังมีฟังก์ชันที่สนับสนุนการเพิ่มเติมอุปกรณ์การเล่นที่จะมี ใหม่ในอนาคต

  DirectInput จะทำให้สามารถเข้าถึงข้อมูลอินพุตได้เร็วกว่าเดิม เพราะมันไม่ต้องพึ่งพาการทำงานของระบบปฏิบัติการ Windows แต่จะสื่อสารโดยตรงกับฮาร์ดแวร์

- **DirectSetup** ช่วยในการติดตั้งส่วนประกอบที่จำเป็นของ DirectX ให้กับผู้ใช้อย่างอัตโนมัติ ทำให้สะดวกมาก DirectSetup จะจัดหาวิธีการอัตโนมัติที่จะติดตั้งแอปพลิเคชั่นให้เหมาะสมกับระบบปฏิบัติการ Windows

- **AutoPlay** เป็นส่วนประกอบสำคัญในระบบปฏิบัติการ Windows ทำให้แอปพลิเคชั่นสามารถที่จะทำการติดตั้งลงไปในฮาร์ดิสก์ได้อย่างอัตโนมัติ โดยแอปพลิเคชั่นจะเริ่มทำงานเมื่อเรานำแผ่นโปรแกรมเกมใส่เข้าไปในไดรฟ์ซีดีรอม AutoPlay เป็นส่วนของ Microsoft Win32 API ใน SDK และไม่เหมือนกับใน DirectX



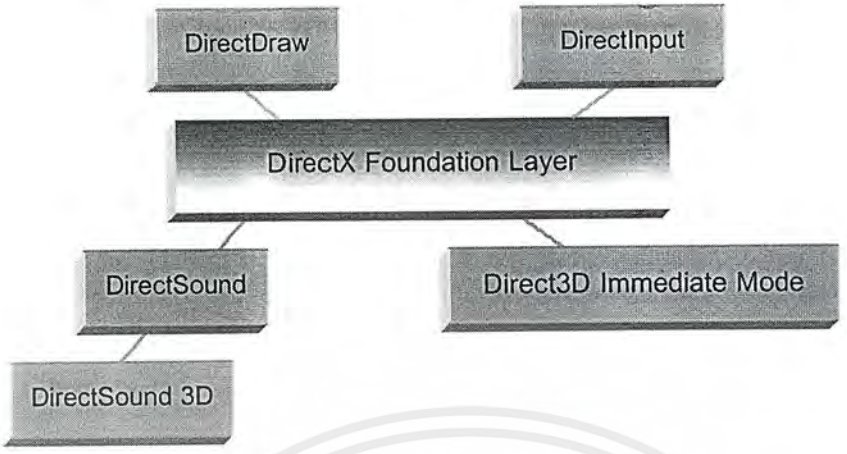
รูปที่ 2-1 ส่วนประกอบของ DirectX

โดยที่ในโครงงานนี้จะพัฒนาแอปพลิเคชั่นเกมแบบ 3 มิติ ในมุมมองของบุคคลที่หนึ่ง ดังนั้นส่วนประกอบของไดเร็กเอ็กซ์ที่จะนำมาใช้ คือ DirectDraw, Direct3D Retained-Mode, DirectInput, DirectSound โดยที่

- DirectDraw ใช้เกี่ยวกับการจัดการการแสดงผลของกราฟิกการ์ด
- Direct3D Retained-Mode ใช้จัดการเกี่ยวกับการแสดงผลแบบ 3 มิติ ของเกม
- DirectInput ใช้จัดการเกี่ยวกับการรับอินพุต ทั้งจาก คีย์บอร์ด, เมาส์, จอยสติ๊ก
- DirectSound ใช้จัดการเกี่ยวกับเสียง และ เพลงประกอบภายในเกม

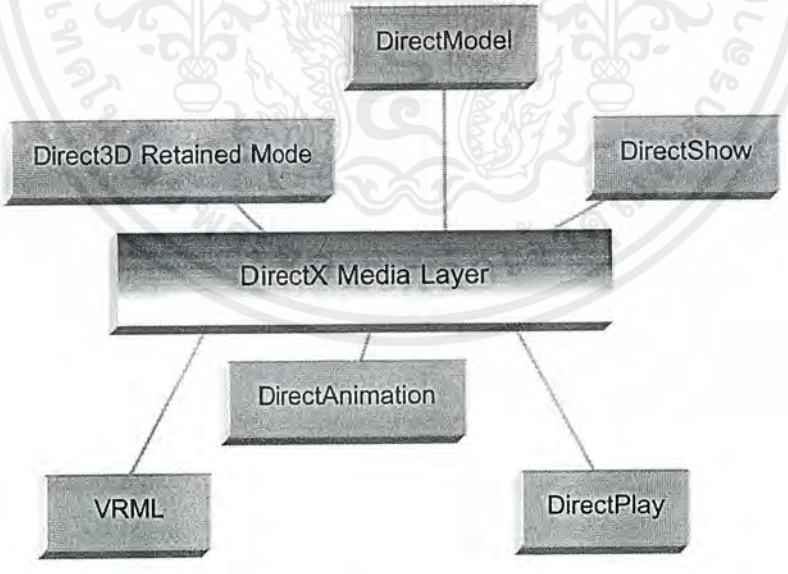
2.5 การแบ่ง Layers ของ DirectX components

DirectX นั้นแบ่งออกได้เป็น 2 ส่วนหลักๆ คือ DirectX Foundation Layer จะทำหน้าที่เป็นตัวกลางในการติดต่อระหว่างแอปพลิเคชั่นกับฮาร์ดแวร์ ผู้ผลิตซอฟต์แวร์ก็เพียงแต่เขียนโปรแกรมให้สนับสนุนมาตรฐานทั้ง 4 ชนิดของ DirectX Foundation Layer โดยไม่จำเป็นต้องคำนึงถึงฮาร์ดแวร์ที่ต้องทำงานด้วย เพราะผู้ผลิตฮาร์ดแวร์ต่างๆที่จะใช้ทำงานร่วมกับระบบปฏิบัติการ Windows จะต้องผลิตฮาร์ดแวร์ให้สนับสนุนมาตรฐานของ DirectX Foundation Layer นี้เช่นกัน DirectX Foundation Layer แสดงในรูปที่ 2-2



รูปที่ 2-2 Components ของ DirectX Foundation Layer

อีกส่วนได้แก่ DirectX Media Layer เป็นส่วนที่นำฟีเจอร์ต่างๆของ DirectX Foundation Layer มาประสานงานใช้ร่วมกันทำให้เกิดฟีเจอร์ใหม่ๆ ซึ่ง DirectX Foundation Layer จะสั่งงานฮาร์ดแวร์โดยรับคำสั่งจาก DirectX Media Layer ซึ่งรับคำสั่งจากซอฟต์แวร์อีกทีหนึ่ง DirectX Media Layer แสดงดังรูปที่ 2-3



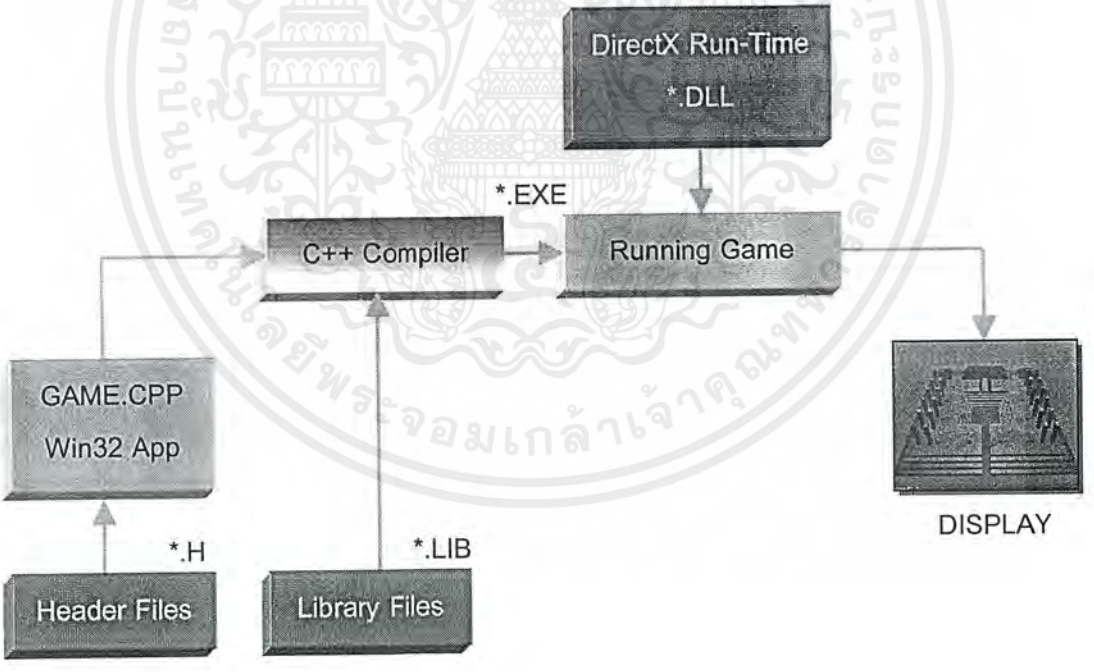
รูปที่ 2-3 Components ของ DirectX Media Layer

2.6 ประเภทของ DirectX library

DirectX Library นั้นมีอยู่ 2 รูปแบบด้วยกัน คือ

1. **DirectX Run-Time** เป็น driver สำหรับผู้ใช้ (end-user) แอปพลิเคชั่นที่ถูกพัฒนาโดยใช้ DirectX โดยที่ DirectX Run-Time นั้นจะอยู่ในรูปแบบของไฟล์ *.DLL ซึ่งจะถูกเรียกใช้โดยแอปพลิเคชั่นเมื่อแอปพลิเคชั่นทำงาน โดยทั่วไปแล้วผู้พัฒนาแอปพลิเคชั่นที่ใช้ DirectX นั้นจะให้ DirectX Run-Time มาพร้อมกับแอปพลิเคชั่นอยู่แล้ว

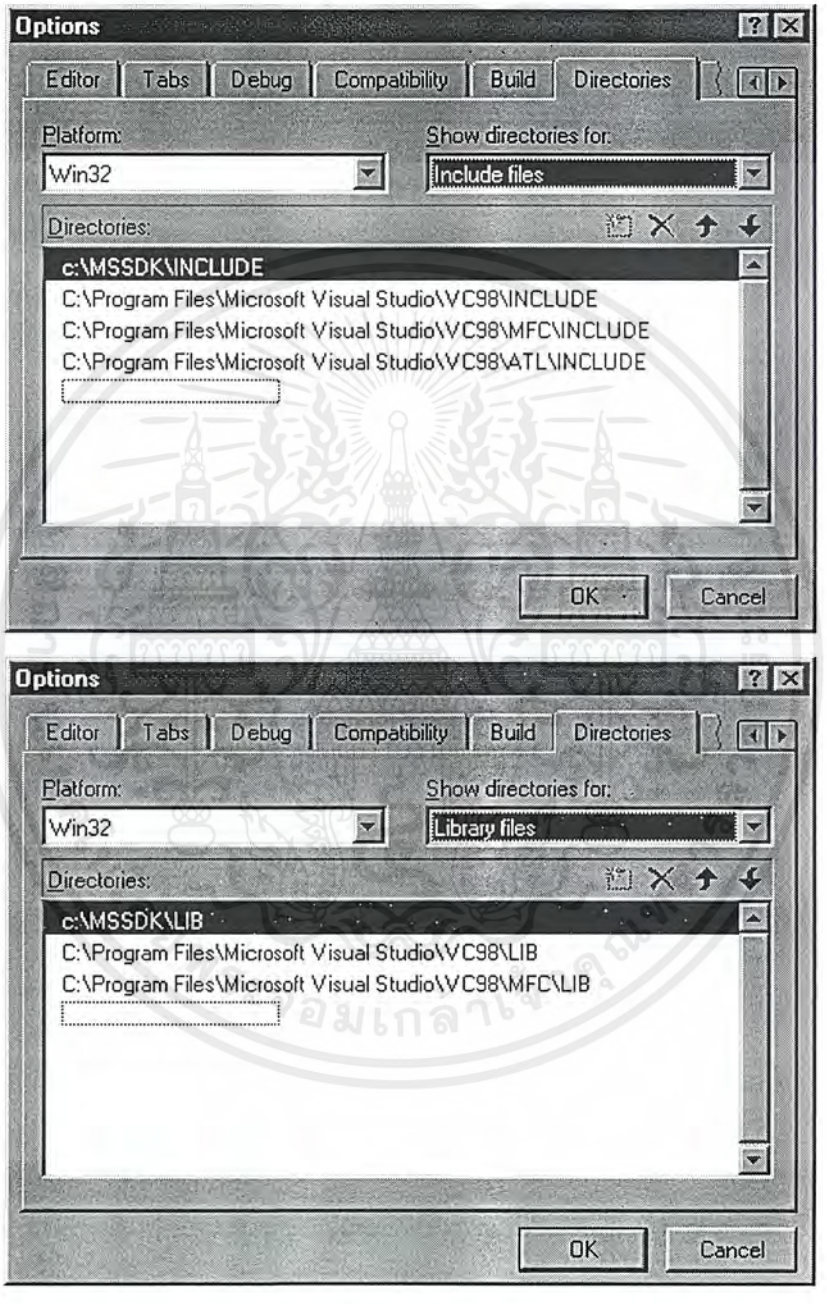
2. **DirectX SDK** เป็นส่วนที่มีไว้สำหรับผู้พัฒนาแอปพลิเคชั่นโดยที่ DirectX SDK นี้จะประกอบด้วยไฟล์ *.LIB และ ไฟล์ *.H สำหรับการพัฒนาแอปพลิเคชั่นโดยใช้คอมไพเลอร์ภาษาซีพลัสพลัส (C++ Compiler) มีเอกสารสำหรับช่วยเหลือในการพัฒนาแอปพลิเคชั่น มีตัวอย่างแอปพลิเคชั่นที่แสดงถึงฟังก์ชันการทำงานต่างๆของ DirectX พร้อม source code โดยที่เราสามารถดาวน์โลด DirectX SDK ได้จาก Web site ของไมโครซอฟต์



รูปที่ 2-4 โครงสร้างของแอปพลิเคชั่นและ DirectX Library

2.7 การใช้ Visual C++ กับ DirectX SDK

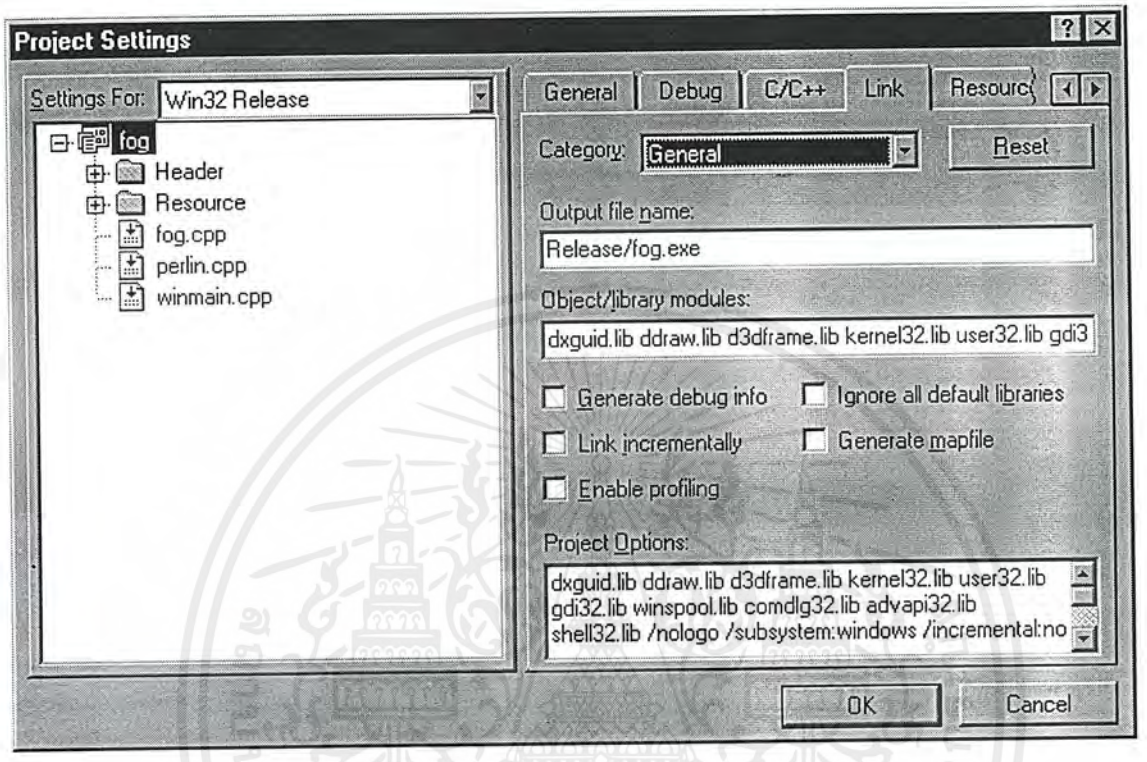
การใช้ DirectX SDK ร่วมกับคอมไพเลอร์ Visual C++ นั้นเราบอกให้คอมไพเลอร์ทราบว่า directory ของ Header files และ Library files นั้นอยู่ที่ใด โดยเราสามารถทำได้โดยเลือก "Options" จาก เมนู "Tool" ของคอมไพเลอร์ Visual C++ จะปรากฏ Options dialog ดังรูปที่ 2-5



รูปที่ 2-5 Option dialog

เราต้องเลื่อน directory ที่เราเพิ่มเข้าไปใหม่นั้นให้อยู่บนสุดมิเช่นนั้น Visual C++ จะใช้ DirectX Library ที่ถูกติดตั้งมาพร้อมกับตัวคอมไพเลอร์เองซึ่งอาจเป็นเวอร์ชันที่เก่ากว่าเวอร์ชันที่เราต้องการใช้งาน

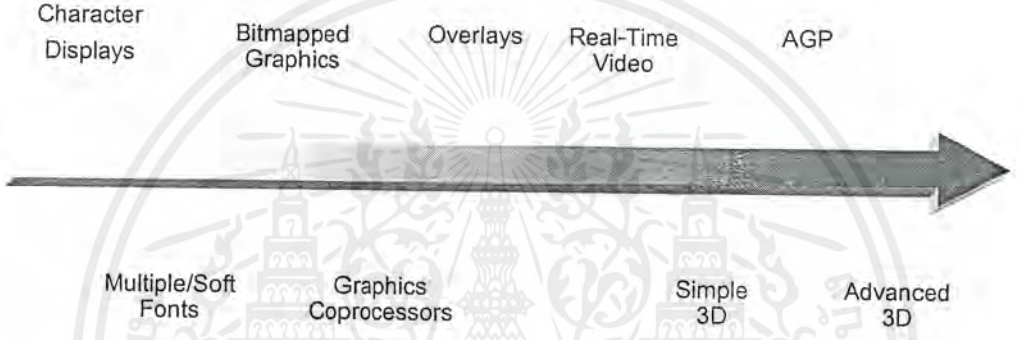
นอกจากนี้เราต้องเซต Project link libraries เพื่อให้คอมไพเลอร์ลิงค์ DirectX Library เข้ากับแอปพลิเคชั่นของเรา โดยเลือก "Settings" จาก เมนู "Project" ของคอมไพเลอร์ Visual C++ จะปรากฏ Project Settings dialog ดังรูปที่ 2-6



รูปที่ 2-6 Project Settings dialog

บทที่ 3
DirectDraw

3.1 เทคโนโลยีการแสดงผล



รูปที่ 3-1 วิวัฒนาการของ display card

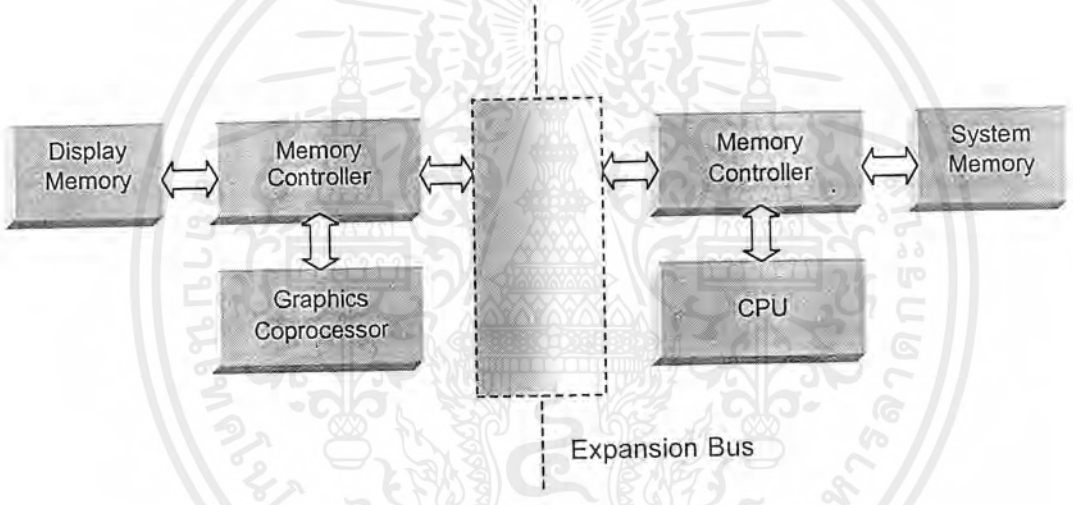
จากรูปที่ 3-1 เราจะเห็นได้ว่าเทคโนโลยีของการแสดงผลนั้นได้พัฒนามาไกลมาก นับตั้งแต่ที่ระบบปฏิบัติการ Microsoft Windows และระบบปฏิบัติการอื่นๆที่เป็นแบบ GUI (Graphical User Interface) ได้รับความนิยม ทำให้เกิดความต้องการ์ดแสดงผล (display card) ที่มีประสิทธิภาพมากขึ้น ซึ่งระบบปฏิบัติการที่เหล่านี้จะช้าเมื่อเปรียบเทียบกับระบบปฏิบัติการแบบ character-based สาเหตุเนื่องมาจากการแสดงผลทุกส่วนจะอยู่ในรูป pixels (รวมถึงตัวอักษรด้วย) ซึ่งต้องการการ update อยู่ตลอดเวลา โดยที่ในระบบปฏิบัติการที่ใช้ระบบ character-based ตัวอักษรหนึ่งตัวจะถูกแทนด้วยข้อมูลขนาด 1 byte ในหน่วยความจำของ display card ด้วยเหตุนี้ตัวอักษรที่เคยถูกแสดงด้วยข้อมูลขนาด 1 byte กลับต้องใช้ข้อมูลขนาดหลาย bytes

การจัดการกับหน่วยความจำทำให้ CPU ต้องรับภาระหนัก โดยต้องแบ่งช่วงเวลาหนึ่งเพื่อจัดการข้อมูลของหน้าจอแสดงผล ด้วยเหตุนี้จึงทำให้เกิด *graphics coprocessor* ขึ้นมา ซึ่งจะช่วยรับภาระเกี่ยวกับการแสดงผลแทน CPU โดยที่จะทำงานขนานกับ CPU ทำให้ CPU มีเวลาไปทำงานอย่างอื่นได้มากขึ้น โดยทั่วไปงานที่เกี่ยวกับกราฟิกจะเป็นงานประเภทการเคลื่อนย้ายข้อมูลในหน่วยความจำจากตำแหน่งหนึ่งไปยังอีกตำแหน่งหนึ่ง ซึ่ง graphics coprocessor ที่เกิดขึ้นมาตัวแรกก็ไม่ได้เป็นอะไรที่มากไปกว่า *blitter* ชื่อนี้ได้มากจากการทำงานพื้นฐานคือ bit block transfer (BLT)

เมื่อมี blitter แล้ว CPU ก็ไม่ต้องทำอะไรนอกจากการ setup blitter โดยใช้คำสั่งในระดับต่ำ (low-level instructions) นอกจากหน้าที่พื้นฐานของ graphics coprocessor ที่ทำการย้ายข้อมูลจากหน่วยความจำที่ตำแหน่งหนึ่งไปอีกตำแหน่งหนึ่งแล้ว มันยังสามารถทำ shrinking, stretching, format conversions, และ ฟังก์ชันอื่นๆได้อีก โดยทั่วไป graphics coprocessor ถูกออกแบบมาเป็นพิเศษเพื่อช่วยในการทำงาน Windows operations ทั่วๆไป การทำงานพื้นฐานคือ BLT operation ซึ่งจะมีความสามารถอื่นๆเพิ่มเข้ามา และความสามารถในการเคลื่อนย้ายข้อมูลได้อย่างรวดเร็ว

3.2 การจัดการหน่วยความจำ

ความต้องการความเร็วในการเคลื่อนย้ายข้อมูลในหน่วยความจำ ทำในเกิด *display memory* ซึ่งเป็น memory ชนิดพิเศษที่อยู่บนการ์ดแสดงผล ทำให้เกิดสถาปัตยกรรมดังในรูปที่ 3-2



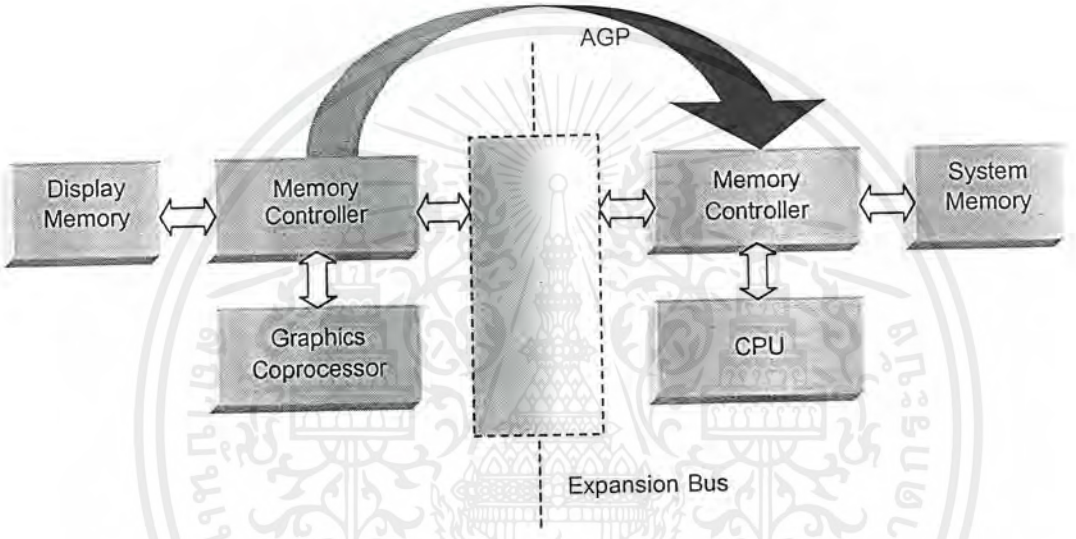
รูปที่ 3-2 สถาปัตยกรรมของการแสดงผลทั่วไป

ด้านหนึ่งของ expansion bus มี CPU กับ system memory ที่ความจุสูง อีกด้านหนึ่งมี graphics coprocessor กับ display memory ที่มีความจุต่ำลงมา แต่ก็เป็นหน่วยความจำที่มีความสำคัญ

System memory เป็นหน่วยความจำที่อยู่บน motherboard ของเครื่องคอมพิวเตอร์ ซึ่ง CPU จะใช้เก็บข้อมูลทุกๆประเภทของแอปพลิเคชั่น system memory เป็นหน่วยความจำที่ CPU เข้าถึงได้ง่าย แต่ไม่สามารถเข้าถึงได้โดย graphics coprocessor นั่นหมายความว่า CPU จะจัดการกับการเคลื่อนย้ายข้อมูลทั้งหมดระหว่างระหว่าง system memory กับ display memory ซึ่งการเคลื่อนย้ายข้อมูลนี้ต้องทำผ่าน expansion bus ที่มีประสิทธิภาพที่จำกัด

เนื่องจาก display memory นั้นอยู่บน display card ทำให้ graphics coprocessor สามารถเข้าถึงได้โดยไม่ต้องผ่าน expansion bus ดังนั้น graphics coprocessor จึงสามารถเข้าถึงหน่วยความจำส่วนนี้ได้รวดเร็วกว่า CPU

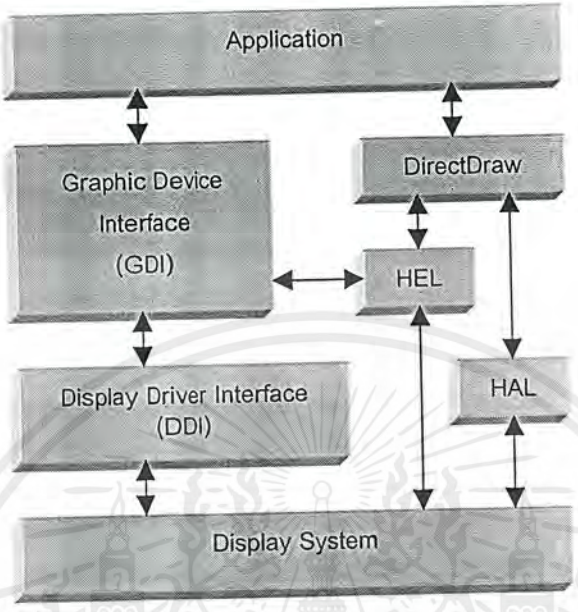
วิวัฒนาการของการออกแบบฮาร์ดแวร์ทำให้เกิดหน่วยความจำสองส่วนที่เป็นอิสระจากกันคือ display memory สำหรับ graphics coprocessor และ system memory สำหรับ CPU ซึ่งถูกแยกออกจากกันโดย expansion bus ที่มีความเร็วต่ำ graphics coprocessor นั้นไม่สามารถเข้าถึง system memory ได้ ส่วน CPU สามารถเข้าถึง display memory ได้โดยผ่าน expansion bus เท่านั้น ระบบการแสดงผลนั้นต้องการทั้งความเร็วในการเคลื่อนย้ายข้อมูลและจำนวนหน่วยความจำที่มาก ซึ่งมันจะดูไม่ยืดหยุ่นที่มีหน่วยความจำจำนวนมากแต่สามารถใช้ได้เฉพาะทาง ด้วยเหตุนี้ทำให้เกิดการออกแบบ Accelerated Graphics Port (AGP) ขึ้นมาดังแสดงในรูปที่ 3-3



รูปที่ 3-3 สถาปัตยกรรมของ Accelerated Graphics Port

AGP สามารถแก้ได้ทั้งสองปัญหาคือ เรื่องความแออัดของ expansion bus และ เรื่องความไม่ยืดหยุ่นของหน่วยความจำ โดยการย้ายระบบการแสดงผลออกไปจาก expansion bus แล้วหันไปใช้ AGP พอร์ตแทน ระบบการแสดงผลสามารถใช้พอร์ตนี้เข้าถึง system memory (ในส่วนที่ถูกออกแบบให้ใช้โดยระบบการแสดงผล) ได้อย่างรวดเร็ว และข้อมูลจำนวนมากสามารถถูกส่งผ่านพอร์ตนี้ได้เร็วกว่าทาง expansion bus เราเรียกหน่วยความจำประเภทนี้ว่า *nonlocal display memory*

3.3 DirectDraw กับ Display Hardware



รูปที่ 3-4 DirectDraw components

ก่อนที่จะมี DirectDraw การเขียนโปรแกรมบน Windows นั้นเราต้องทำการติดต่อกับ *Graphic Device Interface (GDI)* ซึ่ง GDI นั้นสนับสนุนฟังก์ชันการทำงานเกี่ยวกับกราฟฟิกที่เป็นแบบ device-independent จำนวนมากตั้งแต่ font ถึง bitmap โดยที่ GDI นั้นใช้ driver layer ที่เรียกว่า *Display Driver Interface (DDI)* เพื่อติดต่อกับฮาร์ดแวร์ด้านกราฟฟิก ฟังก์ชันการทำงาน DDI layer นั้นจะถูกสนับสนุนโดยผู้ผลิตฮาร์ดแวร์ ซึ่งจะพัฒนา display driver ให้กับผลิตภัณฑ์ของตนเอง ฟังก์ชันการทำงานพิเศษของฮาร์ดแวร์นั้นจะถูกแยกออกไปอยู่ใน display driver และ การ์ดแสดงผลที่ดีนั้นต้องใช้การเร่งความเร็วโดยใช้ฮาร์ดแวร์ (hardware acceleration) เพื่อสนับสนุน Windows operations ให้มากที่สุด

การจักการแบบนี้จะดีสำหรับแอปพลิเคชั่นทั่วๆไป แต่จะไม่ดีสำหรับแอปพลิเคชั่นที่ต้องการเข้าถึงในการทำงานในระดับต่ำ (low-level) และ ใช้ความสามารถพิเศษของฮาร์ดแวร์ โดยปกติแล้ว GDI จะสนับสนุน device-independent และ high level of abstraction ดังนั้นฟังก์ชันการทำงานในระดับต่ำ เช่นการเข้าถึง display memory โดยตรงจึงไม่ถูกสนับสนุน และ ฟังก์ชันการทำงานในระดับสูง (high-level) อย่างเช่น graphic overlays ก็ไม่ได้ถูกสนับสนุนเช่นกัน

DirectDraw ทำการแก้ปัญหานี้โดยใช้สถาปัตยกรรม DirectDraw's HAL/HEL

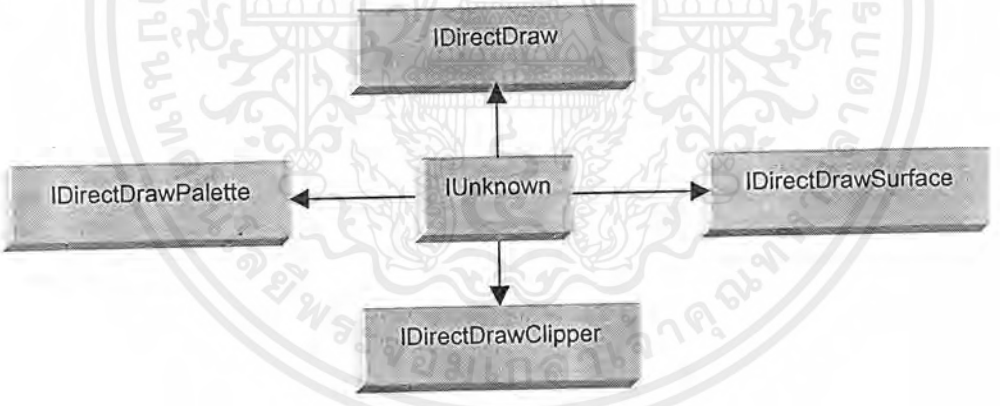
1. **Hardware Abstraction Layer (HAL)** DirectDraw สนับสนุน device-independent ผ่านทาง hardware abstraction layer (HAL) ซึ่งเป็น device-specific interface ที่ถูกสนับสนุนโดยผู้ผลิตฮาร์ดแวร์ ซึ่ง DirectDraw ใช้เพื่อทำงานโดยตรงกับ display hardware

2. **Hardware Emulation Layer (HEL)** เมื่อฮาร์ดแวร์ไม่สนับสนุนการทำงานบางอย่างผ่านทาง hardware abstraction layer (HAL) DirectDraw จะพยายามจำลองการทำงานนั้นผ่านทาง hardware emulation layer (HEL) ซึ่งประสิทธิภาพการทำงานนั้นจะต่ำกว่า hardware abstraction layer (HAL)

เมื่อแอปพลิเคชั่นของเราทำการปลุก DirectDraw ขึ้นมา DirectDraw จะพิจารณาความสามารถที่ถูกรายงานมาจากแต่ละ component ว่าจะตอบสนองการร้องของโดยใช้ HAL หรือ HEL (ความสามารถของ HEL จะเหมือนกันในทุกเวอร์ชันของ DirectDraw ในขณะที่ความสามารถของ HAL แตกต่างกันไปตามการ์ดแสดงผล) โดยปกติแล้ว DirectDraw จะเลือกใช้ HAL ซึ่งจะประมวลผลการร้องขอโดยใช้ฮาร์ดแวร์ ถ้าการร้องขอนั้นถูกสนับสนุนโดย HAL แต่ถ้าการร้องขอนั้นไม่ได้ถูกสนับสนุนโดย HAL แล้ว HEL จะถูกใช้แทนโดยที่ HEL อาจเรียกใช้ GDI เพื่อให้ทำงานบางอย่างให้ หรือ HEL อาจเข้าถึงฮาร์ดแวร์โดยตรงในงานปกติบางอย่างที่เป็นแบบ nonhardware-specific เช่น การเข้าถึงหน่วยความจำ

3.4 DirectDraw Interface

ทุกๆ COM component นั้นจะมี interfaces ของตัวเอง และ DirectDraw ก็เช่นเดียวกัน เราต้องติดต่อกับ component ผ่าน interfaces เหล่านี้ ดังรูปที่ 3-5



รูปที่ 3-5 DirectDraw Interfaces

1. **IUnknow** เป็น interface class ที่ทุกๆ interfaces นั้นสืบทอดมา
2. **IDirectDraw** แสดงถึง video card เราใช้ interface นี้เพื่อเลือก video modes และทำการ set system-cooperation level ทั้งหมด และเป็น interface หลักของ COM object ที่เราสร้างขึ้นมา จากนั้นเราสามารถเราสามารถสร้าง interface อื่นๆได้โดยทำการร้องขอจาก interface หลัก
3. **IDirectDrawSurface** แสดงถึง video memory หรือ surface ที่เราใช้ในการแสดงผล อย่างไรก็ตาม DirectDraw surface นั้นสามารถูกสร้างใน system memory ได้ ซึ่งเราสามารถย้ายข้อมูลจาก system memory ไปสู่ video memory ได้โดยอาศัยฮาร์ดแวร์

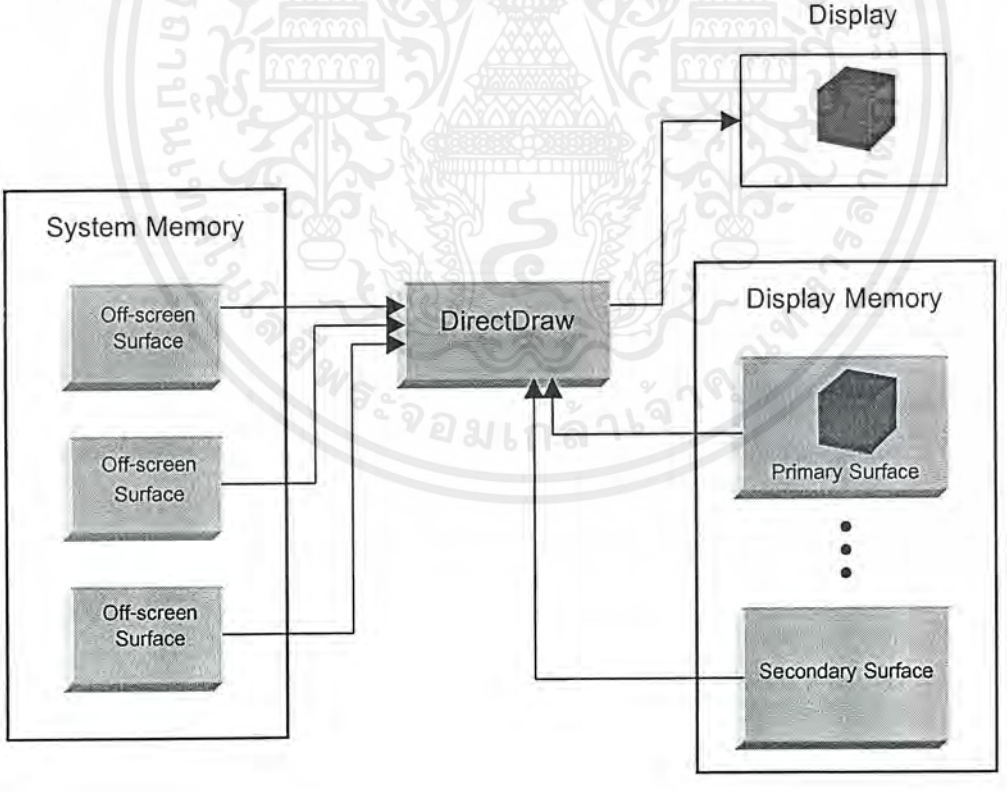
4. **IDirectDrawPalette** แสดงถึง color palette ที่เชื่อมโยงอยู่กับ surface แต่ interface นี้ จะมีความจำเป็นใน 256-color modes กับ Windows modes ที่มี palettes เท่านั้น ในรูปแบบการแสดงผลแบบ Hi-Color หรือ True Color นั้นจะไม่มีความจำเป็น
5. **IDirectDrawClipper** แสดงถึง DirectDraw clipper ซึ่งเป็นชุดของสี่เหลี่ยมที่กำหนดของเขตการแสดงผลที่ DirectDraw สามารถใช้ได้ ซึ่งมีความสำคัญมากกับแอปพลิเคชั่นที่ทำงานใน Windowed modes

3.5 สถาปัตยกรรมของ DirectDraw

แอปพลิเคชั่นทางด้านมัลติมีเดียนั้นต้องการประสิทธิภาพการแสดงผลที่สูง DirectDraw นั้นจึงถูกออกแบบมาให้มีประสิทธิภาพการทำงานที่สูงกว่า GDI และยังคงคุณสมบัติ device-independence DirectDraw นั้นมี tools ต่างๆ ดังนี้

3.5.1 DirectDraw Surfaces

การอ้างถึงหน่วยความจำที่ใช้ในการแสดงผลของ DirectDraw นั้นจะเรียกว่า *Surfaces* ซึ่งเป็นที่เก็บรูปภาพไม่ว่าจะเป็น 2D หรือ 3D ดังรูปที่ 3-6 แสดงแนวความคิดของ surfaces



รูปที่ 3-6 แนวความคิดของ DirectDraw Surfaces

Surfaces สามารถถูกแบ่งออกได้เป็นประเภทต่างๆดังนี้

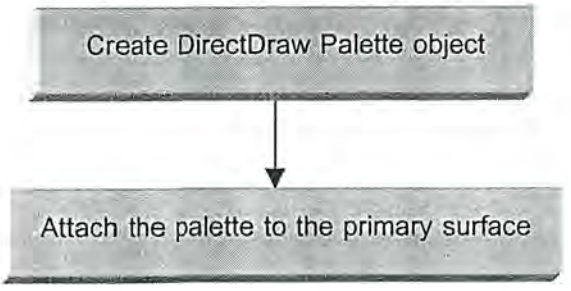
1. **Primary surface** หรือ *Frontbuffer* เป็น surface ที่ถูก mapped โดยตรงไปยัง video memory บน video card ซึ่งจะถูกแสดงผลออกที่จอมอนิเตอร์
2. **Secondary surface** หรือ *Backbuffer* เป็น surface ที่มีโครงสร้างเช่นเดียวกันกับ Primary surface แต่เป็น surface ที่ไม่สามารถมองเห็นได้ มีประโยชน์สำหรับใช้เตรียมการแสดงผล
3. **Offscreen surfaces** เป็น surface ที่ใช้สำหรับเก็บภาพ bitmap ที่ใช้ในแอปพลิเคชั่น

คุณสมบัติของ surface

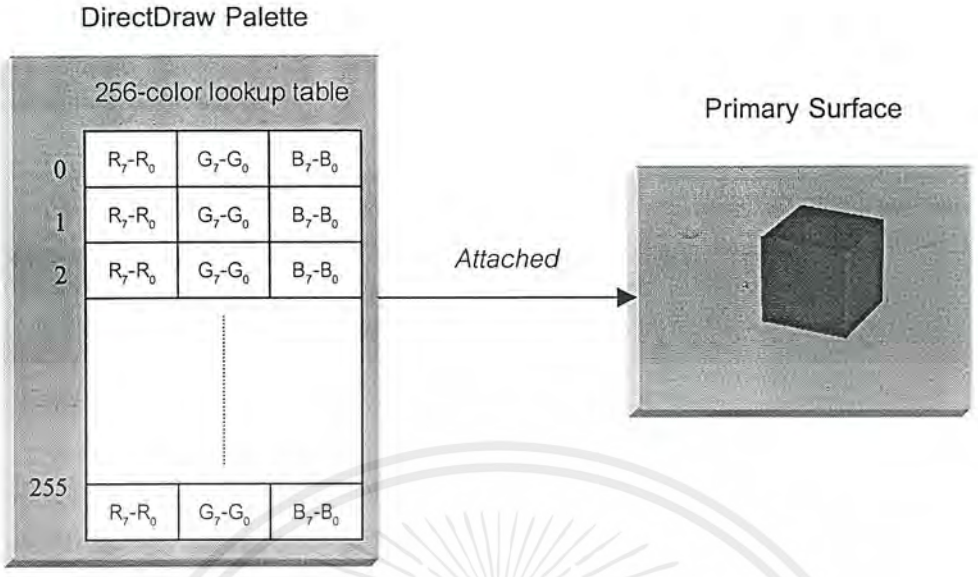
1. Surface นั้นสามารถจะมีขนาดเท่าใดก็ได้ แต่ Primary และ Secondary surface นั้นต้องมีขนาดเท่ากับ resolution ปัจจุบัน
2. เราสามารถสร้าง surface ใน video memory หรือ system memory ก็ได้ มีการทำงานบางอย่างที่สามารถทำได้เร็วกว่าถ้า surfaces นั้นอยู่ใน video memory แต่ถ้า video memory มีพื้นที่ไม่เพียงพอเราก็สามารถสร้าง surfaces ใน system memory ได้ การทำ memory operation จะกระทำได้เร็วที่สุดเมื่อเทียบกับกรณีอื่นๆ เพราะ memory สามารถจะถูกเคลื่อนย้ายได้โดยไม่ต้องผ่าน system bus ที่มีความเร็วต่ำ
3. ทุกๆ surfaces จะต้องมีคุณสมบัติเดียวกัน เช่น bit-depth และ color space เมื่อเราสร้าง surfaces ขึ้นมามันจะมีคุณสมบัติเทียบเท่ากับ primary surface ถ้าเราสร้างมาจาก DirectDraw object ตัวเดียวกัน เราจึงสามารถทำการคัดลอกข้อมูลจาก surface หนึ่งไปยังอีก surface หนึ่งได้ มิเช่นนั้น DirectDraw จะสับสน เช่น เราสั่งให้ DirectDraw คัดลอกข้อมูลจาก surface มี 8-bit color ไปสู่ surface ที่มี 24-bit color

3.5.2 การจัดการกับ Palette

ใน 256-color mode นั้น palette จะเก็บค่า RGB สำหรับแต่ละสี เมื่อเราวาดลงไปบน display memory แต่ละ byte จะเป็นแสดงถึง color index เมื่อเราจะทำงานที่เกี่ยวกับ palette เราต้องสร้าง DirectDraw Palette ขั้นตอนการทำงานเกี่ยวกับ palette แสดงในรูปที่ 3-7 และรูปที่ 3-8



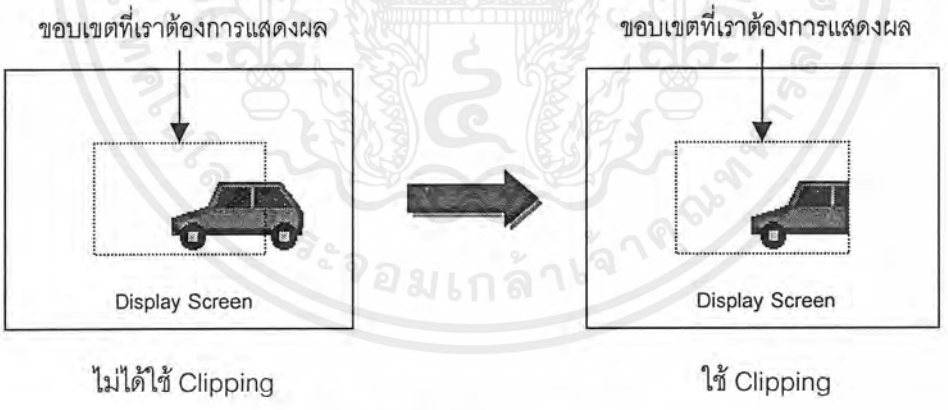
รูปที่ 3-7 ขั้นตอนการจัดการเกี่ยวกับ palette



รูปที่ 3-8 DirectDraw Palette กับ DirectDraw Surface

3.5.3 DirectDraw Clipper

เป็นการกำหนดขอบเขตการแสดงผลโดยตัดส่วนที่อยู่นอกขอบเขตการแสดงผลที่เราต้องการออกไป ดังแสดงในรูปที่ 3-9

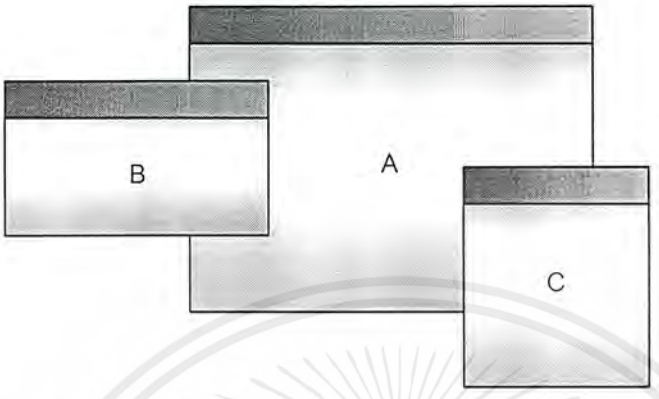


รูปที่ 3-9 การ Clipping

clipping มีประโยชน์มีประโยชน์มากมาย เช่น ในกรณีที่แอปพลิเคชั่นของเราทำงานบน windowed-mode เมื่อแอปพลิเคชั่นของเราทำงานบน full-screen mode เราไม่จำเป็นต้องสร้าง clipper ก็ได้

ใน windowed mode แอปพลิเคชั่นสามารถที่ถูกผู้ใช้ปรับขนาดและเลื่อนตำแหน่งของ window ได้โดยอิสระ ดังรูปที่ 3-10 ดังนั้นถ้าแอปพลิเคชั่น A ซึ่งเป็น DirectDraw application นั้นโดยไม่ได้สนใจในเรื่องนี้ แอปพลิเคชั่น A ก็จะเขียนทับลงไปบน window ของแอปพลิเคชั่นอื่นๆ

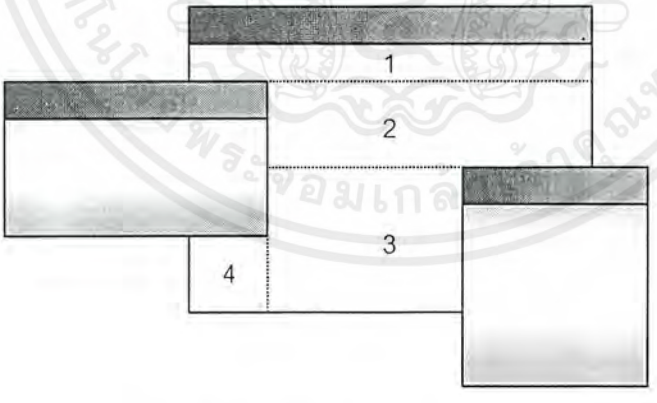
ดังนั้นเราสามารถแก้ไขปัญหานี้ได้โดยใช้ DirectDraw Clipper ซึ่ง clipper จะเก็บ list ของพื้นที่การแสดงผลที่ถูกต้องไว้ใน clip list ซึ่งจะถูกใช้เพื่อควบคุมการแสดงผลของแอปพลิเคชั่น ดังรูปที่ 3-11



รูปที่ 3-10 แอปพลิเคชั่นของเราสามารถที่จะถูกซ้อนทับโดยแอปพลิเคชั่นอื่นๆได้

การใช้ clipper กับ DirectDraw แอปพลิเคชั่น มี 2 แบบ แบบแรกใช้ clipper กับ แอปพลิเคชั่นที่ทำงานใน full-screen mode เราจะทำการเชื่อมต่อ clipper เข้ากับ surface แบบที่ 2 ใช้ clipper กับ แอปพลิเคชั่นที่ทำงานใน windowed-mode เราจะเชื่อมต่อ clipper เข้ากับ window ของแอปพลิเคชั่นนั้น

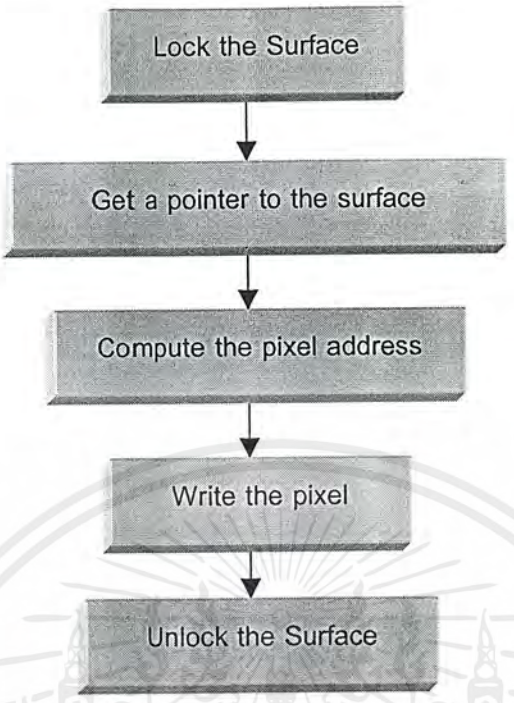
เมื่อเรากำหนด DirectDraw Clipper object ให้กับ window ของแอปพลิเคชั่นแล้ว clip list จะถูก update โดยอัตโนมัติ โดยระบบปฏิบัติการ Windows



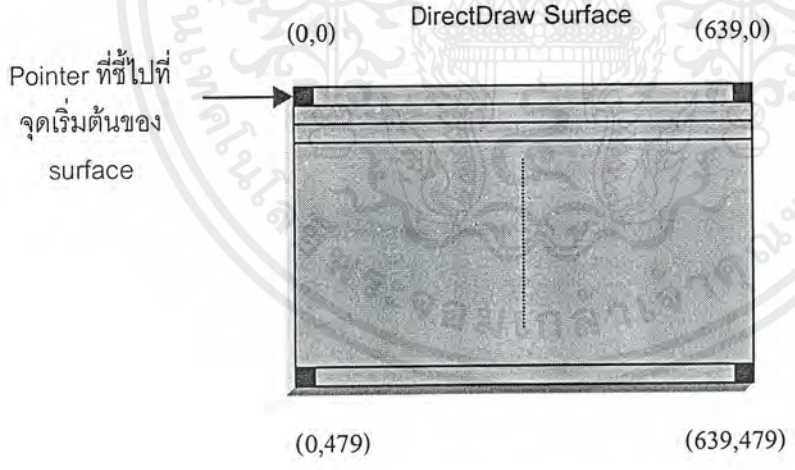
รูปที่ 3-11 clip list ใน window ของแอปพลิเคชั่น

3.5.4 การเข้าถึง Display Memory โดยตรง

การวาดใน DirectDraw นั้นมีความแตกต่างจาก GDI เนื่องจาก GDI มีฟังก์ชันที่ทำหน้าที่จัดการเกี่ยวกับรูปภาพและสีจำนวนมากมาย ในขณะที่ DirectDraw นั้นมีไม่มากมายเท่าแต่สามารถเข้าถึง display memory ได้โดยตรงโดยมีขั้นตอนดังรูปที่ 3-12 และ 3-13



รูปที่ 3-12 ถึงการเข้าถึง Display Memory โดยตรง

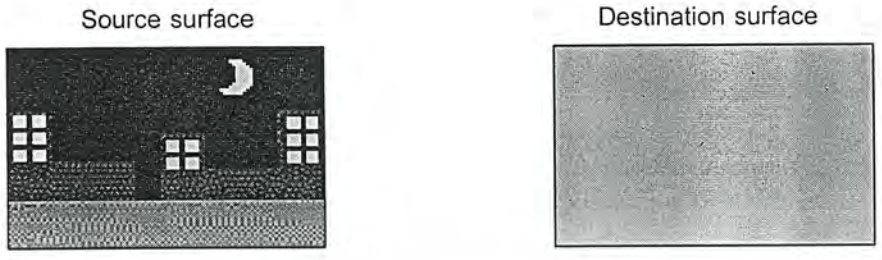


*รูปที่ 3-13 pointer ที่ได้จากการ lock surface ขนาด 640*480*

3.5.5 Blitting

Blit เป็นการคัดลอกข้อมูลที่หน่วยความจำตำแหน่งหนึ่งไปอีกตำแหน่งหนึ่ง ใน DirectDraw จุดเริ่มต้นและเป้าหมายในการคัดลอกต้องเป็น DirectDraw Surface objects โดยที่ surfaces เหล่านี้สามารถจะอยู่ใน system memory หรือ display memory ก็ได้ ดังที่แสดงในรูปที่ 3-14 โดยทั่วไปการ blit ที่ทำใน display memory จะมีประสิทธิภาพสูงกว่าใน system memory รูปที่ 3-15 แสดงโครงสร้างของ DirectDraw

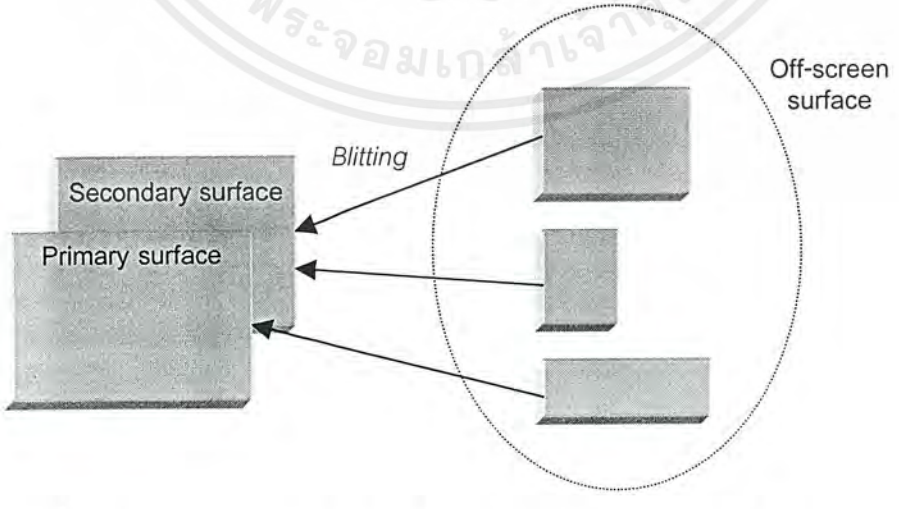
แอปพลิเคชั่นโดยทั่วไป off-screen surfaces จะใช้เก็บภาพเพื่อที่จะทำการ Blit ไปสู่ primary surface หรือ secondary surface



(a) ก่อนทำการ blit

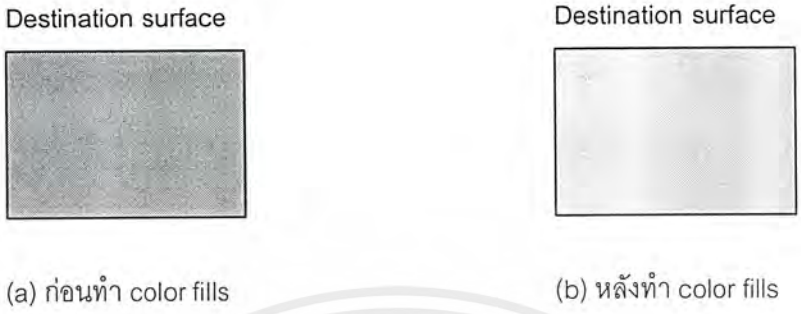
(b) หลังจากทำการ blit

รูปที่ 3-14 ผลหลังการ blit ระหว่าง surfaces

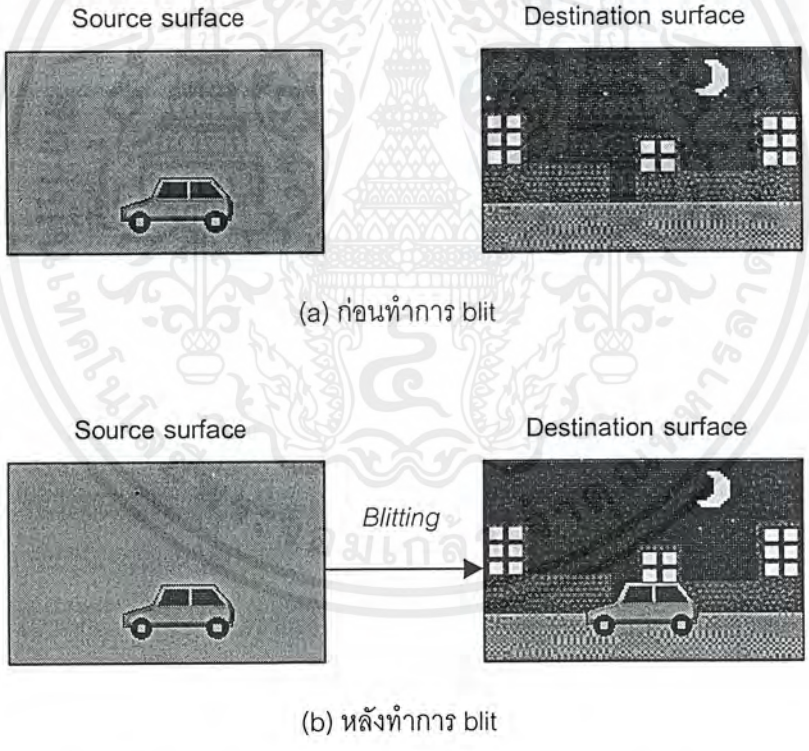


รูปที่ 3-15 การ blit จาก off-screen surface ไปสู่ primary surface และ secondary surface

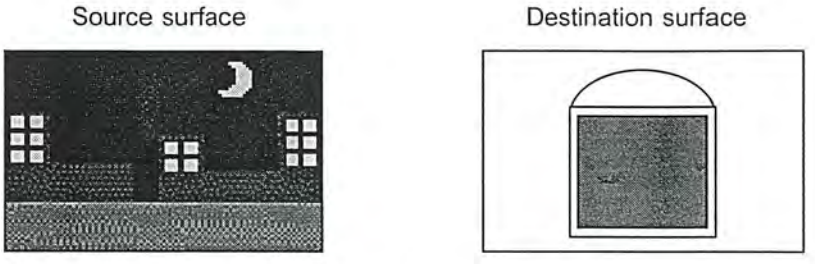
นอกจากการ Blit ตามปกติแล้ว DirectDraw ยังสนับสนุนการใช้เทคนิคพิเศษ ดังแสดงในรูปที่ 3-16, 3-17, 3-18,



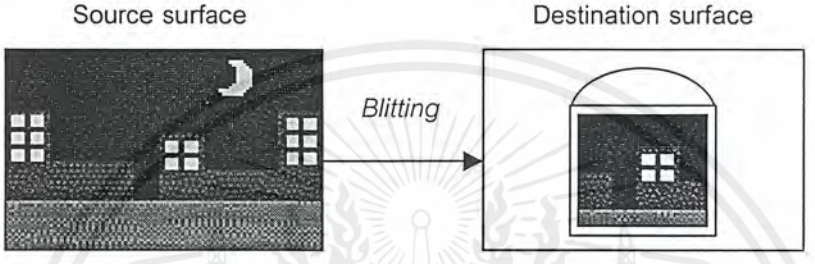
รูปที่ 13-16 การใช้เทคนิค color fills ด้วยสีเหลือง



รูปที่ 13-17 การใช้เทคนิค transparency โดยใช้ source color key เป็นสีเขียว



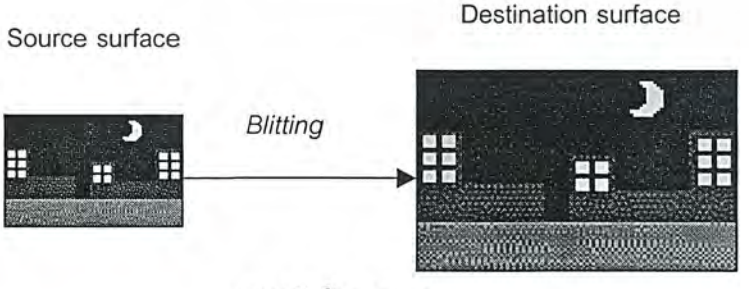
(a) ก่อนทำการ blit



(b) หลังทำการ blit

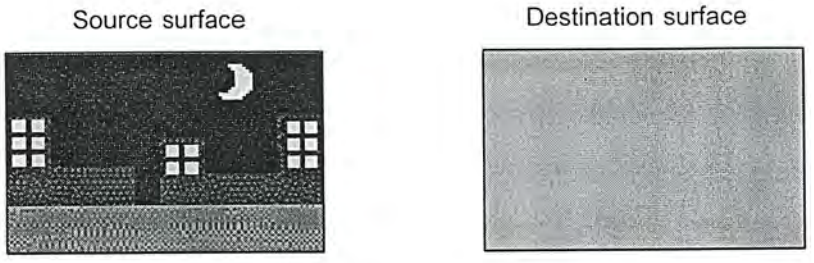
รูปที่ 3-18 การใช้เทคนิค transparency โดยใช้ destination color key เป็นสีเขียว

(a) ก่อนทำการ blit

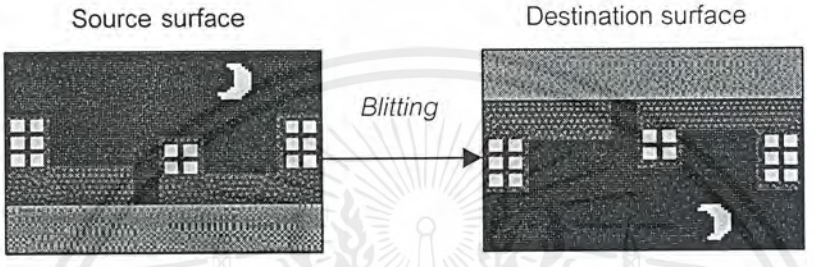


(b) หลังทำการ blit

รูปที่ 3-19 การใช้เทคนิค scaling



(a) ก่อนทำการ blit

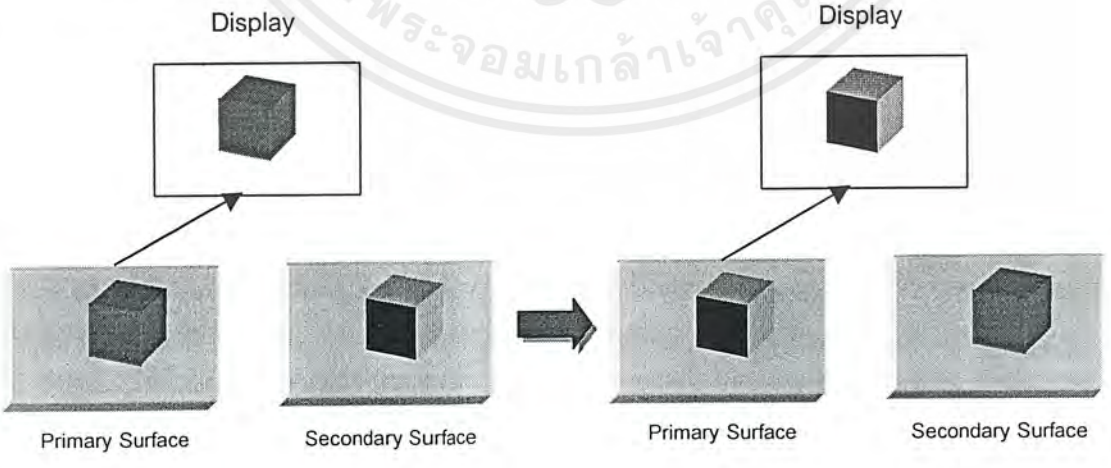


(b) หลังทำการ blit

รูปที่ 3-20 การใช้เทคนิค mirroring ในแนวตั้ง

3.5.6 Page Flipping

เป็นเทคนิคที่ใช้ในการแสดงผลเพื่อให้ได้ภาพเคลื่อนไหวที่มีความนุ่มนวล ไม่เกิดการกระพริบ มีขั้นตอนการทำดังรูปที่ 3-21 โดยที่เราจะวาดภาพที่ต้องการแสดงผลใน frame ต่อลงไปบน secondary surface ก่อน เมื่อวาดภาพทั้งหมดเสร็จเรียบร้อยแล้วเราจึงสั่งให้ DirectDraw ทำการ flip surfaces เพื่อนำภาพที่เราได้เตรียมไว้แสดงผลออกที่จอมอนิเตอร์



(a) ก่อนทำการ Flip

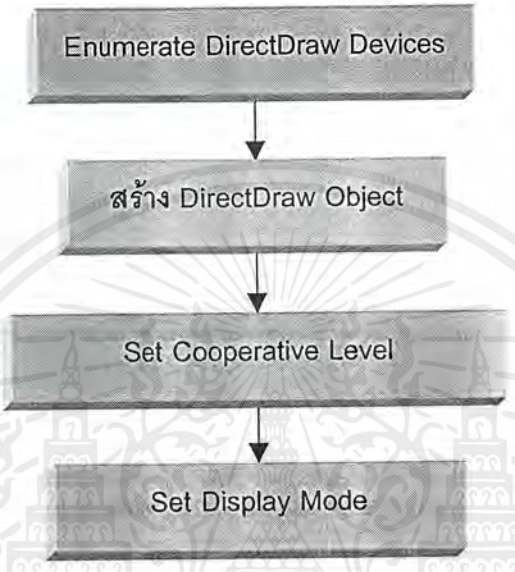
(b) หลังทำการ Flip

รูปที่ 3-21 การทำ Page Flipping

เมื่อ surface ถูก flip จะมีเพียงหน่วยความจำเท่านั้นที่ถูกสับเปลี่ยน ค่าต่างๆของ surface ยังคงเหมือนเดิม palettes และ objects อื่นๆที่เกี่ยวข้องกับ surface นั้นจะไม่ได้รับผลกระทบจากการ flip

3.6 การใช้ DirectDraw

โดยทั่วไปแอปพลิเคชั่นที่ใช้ DirectDraw จะเริ่มด้วยขั้นตอนดังรูปที่ 3-22



รูปที่ 3-22 ขั้นตอนเริ่มต้นของแอปพลิเคชั่นที่ใช้ DirectDraw

3.6.1 DirectDraw object

เมื่อเราทำงานกับ DirectDraw สิ่งแรกที่เราต้องสร้างคือ DirectDraw object ซึ่งจะมี methods ที่เกี่ยวกับการสร้าง และ จัดการกับ child object ของมัน เช่น surfaces และ palettes โดยที่ child objects เหล่านี้จะมี methods ที่สามารถเข้าถึงความสามารถต่างๆของ DirectDraw ได้ DirectDraw object จะสนับสนุน method ที่ใช้ในการตรวจสอบประสิทธิภาพของเครื่องคอมพิวเตอร์ที่แอปพลิเคชั่นทำงานอยู่ เช่น การตรวจสอบจำนวน display memory ที่สามารถใช้ได้ และการจัดการเกี่ยวกับการแสดงผลโดยทั่วๆไป

เราสามารถสร้างได้หลาย DirectDraw object ได้ในแอปพลิเคชั่นเดียวกัน ซึ่งแต่ละ DirectDraw object จะเป็นอิสระต่อกัน Surfaces และ child object อื่นๆนั้นไม่สามารถที่จะแบ่งกันใช้ได้ นอกจาก clippers ซึ่งเป็นอิสระต่อกันและสามารถจะใช้ร่วมกันได้ถ้าจำเป็น

3.6.2 DirectDraw Devices

เมื่อเราสร้าง DirectDraw object เราสร้างมันสำหรับแต่ละ DirectDraw device ที่ถูกเจาะจง โดยจะสอดคล้องกับ HAL ที่ถูกติดตั้งอยู่บนเครื่องคอมพิวเตอร์ เป็นไปได้ที่เครื่องคอมพิวเตอร์ของผู้ใช้จะมีมากกว่าหนึ่ง DirectDraw HAL

3.6.3 Enumerating Devices

เพื่อให้แน่ใจว่าผู้ใช้มีอิสระในการเลือก DirectDraw Devices ที่มีอยู่บนเครื่องคอมพิวเตอร์ของผู้ใช้แต่ละคน เราควรใช้ DirectDraw Enumerating function ดังตารางที่ 3-1

HRESULT DirectDrawEnumerate (LPDDENUMCALLBACK lpCallback, LPVOID lpContext);	
lpCallback	: address ของ callback function ที่จะถูก called กับร่วมกับ description ของแต่ละ device
lpContext	: address ของ context variable ที่ผู้ใช้กำหนด ซึ่งจะถูกส่งไปให้กับ callback function

ตารางที่ 3-1 DirectDrawEnumerate function

Enumeration functions จะทำการระบุสมาชิกของเซตโดยการปลุก callback function สำหรับสมาชิกแต่ละตัวในเซต เมื่อเราเรียกใช้ enumeration function เราให้ pointer ที่ชี้ไปที่ callback function กับมัน callback function ของเราอาจถูกเรียกใช้เพียงหนึ่งครั้ง, หลายครั้ง, หรือไม่ถูกเรียกใช้เลย ขึ้นอยู่จำนวนสมาชิกภายในเซต บาง enumeration functions ให้โอกาสเรายกเลิกการทำ enumeration โดยการการคืนค่าใดค่าหนึ่งที่ถูกกำหนดจาก callback function บาง enumeration function อนุญาตให้เราส่ง context variable ไปให้ enumeration function ซึ่งจะถูกส่งผ่านไปยัง callback function ด้วยความยืดหยุ่นนี้ทำให้เราสามารถสร้าง callback ที่ประพฤติตัวต่างกันขึ้นอยู่กับ context variable แต่ละ enumeration function จะกำหนดการประกาศ callback function ในรูปแบบต่างๆกัน ในกรณีของ DirectDrawEnumerate นั้น callback function จะถูกกำหนดดังตารางที่ 3-2

เมื่อเราเรียกใช้ DirectDrawEnumerate function โดยการผ่านค่าซึ่งเป็นชื่อของ callback function ที่เรากำหนดได้เข้าไป callback function ที่เราได้ประกาศไว้จะถูกเรียกเป็นจำนวนครั้งเท่ากับจำนวน DirectDraw devices ที่มีอยู่บนเครื่องคอมพิวเตอร์ที่แอปพลิเคชั่นทำงานอยู่ callback function ถูกประกาศให้ส่งคืนค่า BOOL ค่านี้ควรถูกเซตให้เป็น DDENUMRET_OK เพื่อทำการ enumeration ต่อจนกว่าจะครบตามจำนวน driver ที่มีอยู่ หรือ DDENUMRET_CANCLE เพื่อหยุดการทำ enumeration เมื่อใดก็ได้ที่เราต้องการ

ชื่อของ callback function นั้นเราสามารถตั้งได้ตามอิสระเพียงแต่ function ต้องมีรูปแบบตามที่ประกาศไว้ใน ตารางที่ 3-2 ทุกๆครั้งที่ callback function ถูกเรียกนั้น callback function จะได้รับค่า pointer ที่ชี้ไปยัง GUID (Global Unique Identifier) ซึ่ง GUID นี้จะเป็นตัวชี้ไปที่ DirectDraw devices ที่มีอยู่บนเครื่องคอมพิวเตอร์ที่แอปพลิเคชั่นทำงานอยู่ ดังนั้นใน callback function เราจะทำการคัดลอกค่า GUID เหล่านี้ไว้เพื่อใช้ในการสร้าง DirectDraw object ต่อไป

```
BOOL DDEnumCallback (GUID FAR *lpGUID,
                     LPSTR lpDriverDescription,
                     LPSTR lpDriverName,
                     LPVOID lpContext);
```

lpGUID	: address ของ unique identifier สำหรับ device นี้
lpDriverDescription	: address ของ description ของ device
lpDrivername	: address ของชื่อ device
lpContext	: address ของ context variable ที่ผู้ใช้กำหนด ซึ่งถูกส่งให้ DirectDrawEnumerate

ตารางที่ 3-2 การกำหนด DDEnumCallback function

3.6.4 การสร้าง DirectDraw Object

ดังตารางที่ 3-3 เราสามารถกำหนดให้ lpGUID เป็น NULL ได้เพื่อเลือก primary device หรือกำหนดให้เป็นตำแหน่งของ GUID ที่เราต้องการ โดยค่าของ GUID นั้นได้มาจากการทำ enumeration

```
HRESULT DirectDrawCreate (GUID FAR *lpGUID,
                          LPDIRECTDRAW FAR *lplpDD,
                          Iunknow FAR *pUnkOuter);
```

lpGUID	: address ของ unique identifier ของ DirectDraw device ที่ถูกกำหนด หรือค่าใดค่าหนึ่งในตารางที่ 3-4
lplpDD	: address ของ pointer ที่จะถูกให้ชี้ไปที่ IDirectDraw interface
pUnkOuter	: ยังไม่ได้ถูกใช้ในปัจจุบัน

ตารางที่ 3-3 DirectDrawCreate function

NULL	: ใช้ primary device
DDCREATE_EMULATIONONLY	: ใช้ HEL เท่านั้น สำหรับการทดสอบและการทำ debugging
DDCREATE_HARDWAREONLY	: ใช้ HAL เท่านั้น กับ default device สำหรับการทดสอบและการทำ debugging

ตารางที่ 3-4 ค่าสำหรับ lpGUID parameter ของ DirectDrawCreate

ตัวอย่างการใช้งาน

```
LPDIRECTDRAW        lpdd;                                   // pointer ที่ชี้ไปที่ DirectDraw interface
if (DirectDrawCreate(NULL, &lpdd, NULL) != DD_OK)    // สร้าง DirectDraw object และตรวจสอบ
ความผิดพลาด
  {/* error */}
```

3.6.5 การตรวจสอบเวอร์ชันของ DirectDraw

โดยทั่วไปแอปพลิเคชั่นที่ใช้ DirectX จะทำการติดตั้งบางเวอร์ชันของ DirectX drivers และ libraries โดยใช้ built-in installation functions ของ DirectX ในกรณีนี้ แอปพลิเคชั่นนั้นอาจจะทำงานกับ DirectX เวอร์ชันที่แอปพลิเคชั่นนั้นทำการติดตั้งลงไป หรือ DirectX เวอร์ชันใหม่กว่าที่ถูกติดตั้งโดยระบบปฏิบัติการ หรือ แอปพลิเคชั่นอื่นๆที่ใช้ DirectX ในบางครั้งผู้พัฒนาแอปพลิเคชั่นต้องการเผยแพร่แอปพลิเคชั่นของตนเองผ่านทางอินเตอร์เนตจึงต้องการลดขนาดของแอปพลิเคชั่นให้เล็กที่สุดจึงไม่ได้รวม DirectX เวอร์ชันที่ใช้พัฒนาแอปพลิเคชั่นนั้นไว้กับแอปพลิเคชั่นด้วย จึงไม่อาจแน่ใจได้ว่า DirectX เวอร์ชันที่ถูกติดตั้งอยู่บนระบบของผู้ใช้นั้นตรงกับความต้องการของแอปพลิเคชั่นหรือไม่

ผู้พัฒนาแอปพลิเคชั่นบางรายพยายามเขียนโปรแกรมเพื่อตรวจสอบระบบของผู้ใช้ เช่น การตรวจสอบไฟล์ต่างๆ โดยทั่วไปวิธีการนี้เป็นวิธีการที่ไม่ถูกต้อง เพราะจะทำให้แอปพลิเคชั่นที่ถูกสร้างขึ้นมาต้องทำงานกับ DirectX เวอร์ชันที่ถูกกำหนดไว้เท่านั้น ทำให้ไม่สามารถใช้กับ DirectX เวอร์ชันใหม่ๆได้ วิธีแก้ปัญหาที่ดีที่สุดคือ ไม่ควรเขียนโปรแกรมเพื่อตรวจสอบเวอร์ชันของ DirectX เราควรจะสนใจกับส่วน interface ที่สนับสนุนฟังก์ชันที่เราต้องการ ในส่วนของ DirectX เวอร์ชันใหม่ๆนั้นจะมี interfaces ใหม่ๆเพิ่มขึ้นมา และมี interfaces ของเวอร์ชันก่อนหน้านั้นด้วย ด้วยเหตุนี้เราสามารถใช้ฟังก์ชัน **QueryInterface** เพื่อทำการร้องขอ interface ที่เราต้องการ ถ้าไม่เกิด error ขึ้นเราก็จะสามารถใช้ interfaces ที่เราต้องการได้ แต่ถ้าเกิด error ขึ้นเราสามารถที่จะตัดสินใจได้ว่าจะไม่อนุญาตให้แอปพลิเคชั่นของเราทำงาน หรือจะให้แอปพลิเคชั่นของเราทำงานในประสิทธิภาพที่ต่ำลงได้

3.6.6 การตรวจสอบประสิทธิภาพของ DirectDraw

ในระบบที่มีฮาร์ดแวร์สนับสนุนการทำงานของ DirectDraw จะทำให้แอปพลิเคชั่นของเราทำงานได้เร็วขึ้น แต่ในระบบที่ไม่มีฮาร์ดแวร์มาสนับสนุนนั้นจะทำให้แอปพลิเคชั่นของเราทำงานช้าลง ดังนั้นเป็นการดีถ้าแอปพลิเคชั่นของเราสามารถปรับตัวเองให้เข้ากับระบบที่ทำงานอยู่ได้ เช่นในระบบที่ไม่มีฮาร์ดแวร์มาสนับสนุนแอปพลิเคชั่นก็ยังทำงานได้โดยจะลดความความซับซ้อนของภาพลงเพื่อให้แอปพลิเคชั่นทำงานได้เร็วขึ้น เราสามารถใช้ GetCaps method เพื่อตรวจสอบประสิทธิภาพของระบบ ดังตารางที่ 3-5

HRESULT IDirectDraw::GetCaps (LPDCAPS lpDDDriverCaps, LPDDCAPS lpDDHELCaps);	
LpDDDriverCaps	: address ของ DDCAPS structure ที่จะถูกเติมด้วย hardware capabilities ของ device
lpDDHELCaps	: address ของ DDCAPS structure ที่จะถูกเติมด้วย capabilities ของ HEL

ตารางที่ 3-5 GetCaps method

ตัวอย่างการใช้งาน

```
DDCAPS      DriverCaps;
DDCAPS      HELCaps;

// Initial โครงสร้าง DDCAPS
memset (&DriverCaps, 0, sizeof (DDCAPS));
DriverCaps.dwSize = sizeof (DDCAPS);
memset (&HELCaps, 0, sizeof (DDCAPS));
HELCaps.dwSize = sizeof (DDCAPS);

// Get DirectDraw capabilities
if (lpdd->GetCaps (&DriverCaps, &HELCaps) != DD_OK)
   {/* error */}
```

3.6.7 Set Cooperative Level

เป็นการกำหนดว่าแอปพลิเคชั่นของเราจะทำงานใน windowed mode หรือ full-screen mode เราจะเปลี่ยน display mode หรือจะจัดการกับ palette หรือไม่ เราจะอนุญาตให้ผู้ใช้ทำการ reboot ขณะที่ใช้แอปพลิเคชั่นของเราหรือไม่ เราสามารถกำหนดคุณสมบัติเหล่านี้ได้โดยอาศัย SetCooperativeLevel method ดังตารางที่ 3-6

```
HRESULT IDirectDraw::SetCooperativeLevel (HWND hWnd,
                                          DWORD dwFlags);

hWnd        : window handle ของแอปพลิเคชั่น สามารถใส่ค่า NULL
              ได้ถ้าเราใช้ DDSCL_NORMAL
dwFlags     : control flags ดังตารางที่ 3-7
```

ตารางที่ 3-6 SetCooperativeLevel method

DDSCL_ALLOWMODEX	: อนุญาตให้ใช้ ModeX ต้องใช้ร่วมกับ DDSCL_EXCLUSIVE และ DDSCL_FULLSCREEN
DDSCL_ALLOWREBOOT	: อนุญาตให้ reboot โดยกด Ctrl+Alt+Delete เมื่ออยู่ใน full-screen mode ใช้ร่วมกับ DDSCL_EXCLUSIVE และ DDSCL_FULLSCREEN
DDSCL_EXCLUSIVE	: ใช้ร่วมกับ DDSCL_FULLSCREEN เพื่อใช้ exclusive level
DDSCL_FULLSCREEN	: DirectDraw จะ Update video display ทั้งหน้าจอ, ใช้ร่วมกับ DDSCL_EXCLUSIVE
DDSCL_NORMAL	: แอปพลิเคชั่นทำงานใน windowed mode
DDSCL_NOWINDOWCHANGES	: DirectDraw แอปพลิเคชั่นไม่สามารถ minimize หรือ restore ได้

ตารางที่ 3-7 Cooperative level flags

ตัวอย่างการใช้งาน

```
// Set cooperative level to exclusive full-screen mode
lpdd -> SetCooperativeLevel (hwnd,
                             DDSCL_ALLOWMODEX |
                             DDSCL_ALLOWREBOOT |
                             DDSCL_EXCLUSIVE |
                             DDSCL_FULLSCREEN);
```

3.6.8 Enumerating Display Modes

DirectDraw object สนับสนุน methods ที่ช่วยจัดการกับ display modes อย่างสมบูรณ์ เราสามารถทำการ enumerate display modes ได้แบบเดียวกับที่เรา enumerate devices โดยใช้ IDirectDraw::EnumDisplayModes ดังตารางที่ 3-8

```
HRESULT IDirectDraw::EnumDisplayModes ( DWORD dwFlags,
                                        LPDDSURFACEDESC lpDDSurfaceDesc,
                                        LPVOID lpContext,
                                        LPDDENUMMODESCALLBACK
                                            lpEnumModesCallback);
```

dwFlags	: DDEDM_REFRESHRATES แบ่งกลุ่ม modes ตาม refresh rate DDEDM_STANDARDVGAMODES ทำการ enumerate mode 13H ด้วย
lpDDSurfaceDesc	: DDSURFACEDESC structure ใช้สำหรับ filter modes ที่เราต้องการ ให้ค่า NULL เพื่อ enumerate modes ทั้งหมด
lpContext	: ค่า context ที่ถูกส่งไปให้ callback function
lpEnumModesCallback	: address ของ callback function

ตารางที่ 3-8 EnumDisplayModes methods

```
BOOL WINAPI IDirectDraw::EnumModesCallback (LPDDSURFACEDESC lpDDSurfaceDesc,
                                            LPVOID lpContext);

lpDDSurfaceDesc        : surface description ที่อธิบาย display mode
lpContext              : ค่า context ที่รับมาจาก enumeration method
```

ตารางที่ 3-9 EnumerateModesCallback function

3.6.9 Setting Display Mode

```
HRESULT IDirectDraw::SetDisplayMode( DWORD dwWidth,
                                     DWORD dwHeight,
                                     DWORD dwBPP,
                                     DWORD dwRefreshRate,     // กำหนดให้เป็น 0
                                     DWORD dwFlags);          // กำหนดให้เป็น 0
```

ตารางที่ 3-10 SetDisplayMode method

ตัวอย่างการใช้งาน

```
if (lpdd -> SetDisplayMode(640, 480, 8) != DD_OK)
   {/* error */}
```

3.6.10 การสร้าง Primary Surface

```
HRESULT IDirectDraw::CreateSurface (LPDDSURFACEDESC lpDDSurfaceDesc,
                                    LPDIRECTDRAWSURFACE FAR *lplpDDSurface,
                                    IUnknow Far *pUnkOuter); // กำหนดให้เป็น NULL
```

lpDDSurfaceDesc	: ชี้ไปที่ *DirectDraw surface description structure*
lplpDDSurface	: ชี้ไปที่ DirectDraw surface interface
pUnkOuter	: เป็น NULL เสมอ

ตารางที่ 3-11 CreateSurface method

```
typedef struct _DDSURFACEDESC {
    DWORD dwSize;                   // ขนาดของโครงสร้างนี้เป็น bytes
                                    //เราต้อง set เอง
    DWORD dwFlags;                  // flags field
    DWORD dwHeight;                 // Height ของ surface
    DWORD dwWidth;                  // Width ของ surface
    union {
        LONG lPitch;                // จำนวน byte ใน 1 แถว
        DWORD dwLinearSize;         // ไม่ใช้
       };
    DWORD dwBackBufferCount;        // จำนวน BackBuffers
    union {
        DWORD dwMipMapCount;        // จำนวนของ mip levels
        DWORD dwZBufferBitDepth;    // ขนาดของ Z buffer
        DWORD dwRefreshRate;        // refresh rate
       };
    DWORD dwAlphaBitDepth;          // จำนวนของ bit สำหรับ alpha
    DWORD dwReserved;
    DWORD lpSurface;                // pointer ชี้ไปที่ surface ของ memory
```

ต่อ...

```
    DDCOLORKEY ddckCKDestOverlay;        // destination overlay
    DDCOLORKEY ddckCKDesBlt;             // destination
    DDCOLORKEY ddckCKSrcOverlay;         // source overlay
    DDCOLORKEY ddckCKSrcBlt;             // source blit
    DDPIXELFORMAT ddpfPixelFormat;       // pixel format of surface
    DDSCAPS ddsCaps;                     // capabilities structure

} DDSURFACEDESC;
```

ตารางที่ 3-12 structure ของ DDSURFACEDESC

dwFlags	: DDSD_ALL
	: DDSD_ALPHABITDEPTH
	: DDSD_BACKBUFFERCOUNT
	: DDSD_CAPS
	: DDSD_CKDESTBLT
	: DDSD_CKDESTOVERLAY
	: DDSD_CKSRCBLT
	: DDSD_CKSRCOVERLAY
	: DDSD_HEIGHT
	: DDSD_LINEARSIZE
	: DDSD_LPSURFACE
	: DDSD_MIPMAPCOUNT
	: DDSD_PITCH
	: DDSD_PIXELFORMAT
	: DDSD_REFRESHRATE
	: DDSD_WIDTH
	: DDSD_ZBUFFERBITDEPTH
dwSize	: Size ของ DDSURFACEDESC
dwHieght	: height ของ surface
dwWidth	: width ของ surface

lpSurface : pointer ชี้ไปที่ VRAM ของ surface เมื่อ surface ถูก LOCK

dwBackBufferCount : จำนวน offscreen buffer

lPitch : *memory pitch* ของ surface

ddsCaps : ความสามารถอื่นๆของ surface

: ความจริงเป็น DWORD Flags

```
typedef struct _DDSCAPS
{
        DWORD dwCaps;        // ความสามารถของ surface
} DDSCAPS, *LPDDSCAPS;
```

ตารางที่ 3-13 structure ของ DDSCAPS

dwCaps	: DDSCAPS_BACKBUFFER	surface เป็น backbuffer ขณะทำการ flipping chain
	: DDSCAPS_COMPLEX	มีมากกว่า primary surface
	: DDSCAPS_FLIP	surface สามารถ flip ได้
	: DDSCAPS_FRONTBUFFER	surface เป็น frontbuffer ใน flipping structure
	: DDSCAPS_MODEX	surface เป็น 320 *200 หรือ 320*240 เป็น ModeX surface
	: DDSCAPS_OFFSCREENPLAIN	Offscreen surface
	: DDSCAPS_OWNDC	surface มีความเกี่ยวข้องกับ Windows device context ในช่วงเวลาที่นาน
	: DDSCAPS_PRIMARYSURFACE	surface เป็น primary surface สามารถมองเห็นได้, และถูกแสดงออกหน้าจอ
	: DDSCAPS_STANDARDVGAMODE	surface เป็น standard VGA mode ไม่ใช่ modeX
	: DDSCAPS_SYSTEMMEMORY	memory ในส่วนของ surface ถูก allocate ใน system memory

ตัวอย่างการใช้งาน

```
// pointer ชี้ไปที่ DirectDraw object
LPDIRECDRAW             lpdd;
// เก็บ DirectDraw surface description
DDSURFACEDESC       ddsd;
// pointer ชี้ไปที่ interface เมื่อ surface ถูกสร้างขึ้น
LPDIRECTDRAWSURFACE       lpddsprimary;
// สร้าง DirectDraw object
DirectDrawCreate (NULL, &lpdd, NULL);
// set cooperation level
lpdd -> SetCooperationLevel (hwnd, DDSCL_ALLOWREBOOT |
                                   DDSCL_ALLOWMODEX |
                                   DDSCL_FULLSCREEN |
                                   DDSCL_EXCLUSIVE);
// set display mode 640*480 256 colors
lpdd -> SetDisplayMode (640, 480, 8);

// สร้าง primary surface
// เซตขนาดของ ddsd
 ddsd.dwSize = sizeof (ddsd);
// บอกว่าเราต้องการสร้าง primary surface
ddsd.dwFlags = DDSD_CAPS;
ddsd.ddCaps.dwFlags = DDSCAPS_PRIMARYSURFACE;
// สร้าง surface
if (lpdd -> CreateSurfacce (&ddsd, &lpddsprimary, NULL) != DD_OK)
    {/* error */}
```

3.6.11 การ Release Surface

```
// Check ก่อนว่า surface ที่เราจะ Release นั้นไม่เป็น NULL
if (lpddsprimary)
    lpddsprimary -> Release();
// Release DirectDraw object
if (lpdd)    lpdd -> Release();
```

3.6.12 การ Lock Surface

```
HRESULT IDirectDrawSurface::Lock (LPRECT lpDestRect,
                                  LPDDSURFACEDESC lpDDSD,
                                  DWORD dwFlags,
                                  HANDLE hEvent);
```

lpDestRect	: rectangle ที่เราจะทำการ Lock
lpDDSD	: คืน properties ของ surface ที่ถูก Lock
dwFlags	: control flags
hEvent	: ไม่ใช้ set ให้เป็น NULL

ตารางที่ 3-14 Lock method

dwFlags	: DDLOCK_READONLY	surface ที่ถูก Locked อ่านได้อย่างเดียว
	: DDLOCK_SURFACEMEMMORYPTR	surface ที่ถูก Locked จะ return pointer ชี้ไปที่ surface memory โดยตัวแปร lpSurface ใน DDSURFACEDESC structure
	: DDLOCK_WAIT	รอจนกว่าจะ Lock ได้
	: DDLOCK_WRITEONLY	surface ที่ถูก Locked สามารถจะเขียนได้อย่างเดียว

3.6.13 การ Unlock surface

```
HRESULT IDirectDrawSurface::Unlock (LPVOID lpSurfaceData);
                                    // ชี้ไปที่ Surface
                                    // เราสามารถใช้ NULL ได้ถ้าต้องการจองทั้ง
                                    surface
```

ตารางที่ 3-15 Unlock method

ตัวอย่างการใช้งาน

```
// pointer to DirectDraw object
LPDIRECTDRAW          lpdd;
// เก็บ DirectDraw surface description
DDSURFACEDESC         ddsd;

// pointer ชี้ไปที่ interface เมื่อ surface ถูกสร้างขึ้น
LPDIRECTDRAWSURFACE       lpddsprimary;
// access surface memory
UCHAR video_buffer = NULL;

// สร้าง DirectDraw object
DirectDrawCreate (NULL, &lpdd, NULL);
// set cooperation level
lpdd -> SetCoperationLevel (hwnd, DDSCL_ALLOWMODEX |
                                  DDSCL_ALLOWREBOOT |
                                  DDSCL_FULLSCREEN |
                                  DDSCL_EXCLUSIVE);
// set display mode
lpdd -> SetDisplayMode (640, 480, 8);

// set data structure เพื่อสร้าง primary surface
ddsd.dwSize = sizeof (ddsd);
ddsd.dwFlags= DDSD_CAPS;
ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE;

// สร้าง primary surface
lpdd -> CreateSurface (&ddsd, &lpddsprimary, NULL);

// set surface description เพื่อทำการ lock surface
memset (&ddsd, 0, sizeof (ddsd));
ddsd.dwSize = sizeof (ddsd);
```

```
// Lock primary surface
lpddsprimary -> Lock (NULL, &ddsd,
                      DDLOCK_SURFACEMEMORYPTR |
                      DDLOCK_WAIT,
                      NULL);
// get video pointer
video_buffer = (UCHAR *) ddsd.lpSurface;

// ใช้ pointer นี้สำหรับ access surface
video_buffer[x + y*ddsd.lPitch] = col;

// Unlock surface
lpddsprimary -> Unlock (dsdsd.lpSurface);      // หรือจะใช้
lpddsprimary -> Unlock (videobuffer);          // หรือจะใช้
lpddsprimary -> Unlock (NULL);                 // ใช้ได้เหมือนกันหมด
```

3.6.14 การสร้าง Secondary Surface

หลังจากสร้าง primary surface, เราทำการ query สำหรับ secondary surfaceฟังก์ชันที่เพิ่มขึ้นมาคือ GetAttachedSurface();

ตัวอย่างการใช้งาน

```
// DirectDraw surface description
DDSURFACEDESC       ddsd;
// device capabilities structure, ถูกใช้สำหรับร้องขอ Secondary Backbuffer
DDSCAPS             ddscaps;
LPDIRECTDRAWSURFACE     lpddsprimary,   // Primary surface
                        lpddsback;      // Secondary backbuffer surface

// เตรียมสร้าง Primary surface ที่มี Backbuffer
memset (&ddsd, 0, sizeof (ddsd);
ddsd.dwSize = sizeof (ddsd);
```

```
// set Flags เพื่อยืนยันความถูกต้องของ capabilities field และ backbuffer count field
ddsd.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
// เราต้องการให้ DirectDraw รู้ว่าเราต้องการ complex flippable surface structure
ddsd.ddCaps.dwCaps = DDCAPS_PRIMARYSURFACE |
                     DDCAPS_FLIP |
                     DDCAPS_COMPLEX;
// set Backbuffer count ให้ = 1
ddsd.dwBackBufferCount = 1;

// สร้าง Primary surface
lpdd -> CreateSurface (&ddsd, &lpddsprimary, NULL);

// ร้องขอ Secondary backbuffer  surface โดยใช้ ddcaps
ddcaps = DDCAPS_BACKBUFFER;
lpddsprimary -> GetAttachedSurface (&ddcaps, &lpddsback);
```

3.6.15 Flipping

```
HRESULT IDirectDrawSurface::Flip (LPDIRECTDRAWSURFACE3 lpDDSurfaceOverride,
                                  // เป็น NULL เสมอ
                                  DWORD dwFlags);   // เป็น DDFLIP_WAIT เสมอ
```

ตารางที่ 3-16 Flip method

การเรียกใช้ Flip() เราต้องเรียกใช้จาก Primary surface interface เสมอ

ตัวอย่างการใช้งาน

```
// flip Primary surface และ Secondary surface
while (lpddsprimary -> Flip (NULL, DDFLIP_WAIT) != DD_OK);
```

3.6.16 การสร้าง Offscreen Surface

การสร้าง offscreen surfaces นั้นสามารถสร้างได้ทั้งใน video memory และ system memory

ตัวอย่างการใช้งาน

```
DDSURFACEDESC           ddsd;
LPDIRECTDRAWSURFACE     lpwork;

memtset (&ddsd, 0, sizeof (ddsd));
ddsd.dwSize = sizeof (ddsd);

// set flags
ddsd.dwFlags = DDSD_CAPS | DDSD_WIDTH | DDSD_HEIGHT;

// กำหนดขนาดของ surface ที่จะสร้าง
ddsd.dwWidth = 200;
ddsd.dwHeight = 100;

// surface ที่จะสร้างเป็น surface ประเภทใด
ddsd.ddsCaps.dwCaps = DDSCAPS_OFFSCREENPLAIN;

// ทำการสร้าง surface และตรวจสอบ error
if (lpdd -> CreateSurface (&ddsd, &lpwork, NULL) != DD_OK)
   {/* error */}
```

3.6.17 การสร้าง DirectDraw Palette

```
typedef struct tagPALETTEENTRY { BYTE peRed;      // red component 8-bits
                                 BYTE peGreen;    // green component 8-bits
                                 BYTE peBlue;     // Blue component 8-bits
                                 BYTE peFlags;    // control flags
                                                  ให้เท่ากับ PC_NOCOLLAPSE
                                 } PALETTEENTRY;
```

ตารางที่ 3-17 PALETTEENTRY structure

```
HRESULT IDirectDraw::CreatePalette (DWORD dwFlags,
                                    LPPALETTEENTRY lpColorTable,
                                    LPDIRECTDRAWPALETTE FAR *lplpDDPal,
                                    IUnknow FAR *pUnkOuter);
```

dwFlags	: control flags
lpColorTable	: ไม่ใช้ set ให้เป็น NULL
lplpDDPal	: pointer ที่ชี้ไปที่ palette
pUnkOuter	: ไม่ใช้ set ให้เป็น NULL

ตารางที่ 3-18 CreatePalette method

dwFlags	: ให้เท่ากับ DDPCAPS_8BIT | DDPCAPS_INITIALIZE | DDPCAPS_ALLOW256

ตัวอย่างการใช้งาน

```
// create array of palette entry
PALETTEENTRY            palette[256];
// the palette object
LPDIRECTDRAWPALETTE     lpddpal;

// first clear out all the entries
memset (palette, 0, 256*sizeof (PALETTEENTRY));

// create a R, G, B gradient palette
for (int index = 0; index < 256; index++)
    {
    if (index < 64)
        palette[index].peRed = index*4;
    else
    if (index >= 64 && index < 128)
```

```
        palette[index].peGreen = (index – 64)*4;
    else
    if (index >= 128 && index < 192)
        palette[index].peBlue = (infdex – 128)*4;
    else
    if (index >= 192 && index < 256)
        palette[index].peRed = palette[index].peGreen =
        palette[index].peBlue = (index – 192)*4;
    // set falags
    palette[index].peFlags = PC_NOCOLLAPSE;
    }
// สร้าง palette
if ((lpdd -> CreatePalette( DDPCAPS_8BIT | DDPCAPS_INITIALIZE |
                            DDPCAPS_ALLOW256,
                            palette, &lpddpal, NULL)) != DD_OK)
    {/* erreo */}
// หลังจากสร้าง palette object แล้วต้องกำหนด palette ให้กับ surface
lpddsprimary -> SetPalete (lpddpal);
```

3.6.18 Set Palette Entries

```
HRESULT IDirectDrawPalette::SetEntries (DWORD dwFlags,
                                        DWORD dwStartingEntry,
                                        DWORD dwCount,
                                        LPPALETTEENTRY lpEntries);
```

dwFlags	: control flags เป็น 0 เสมอ
dwStartingEntry	: starting index to change
dwCount	: number of colors to change
lpEntries	: pointer to data storage

ตารางที่ 3-19 แSetEntries method

ตัวอย่างการใช้งาน

```
// create single color
static PALETTEENTRY color = {10, 0, 20, PC_NOCOLLAPS};
// change the palette entry at location 'index'
lpddpal -> SetEntries (0, index, 1, color);

// define and initialize the new palette
PALETTEENTRY new_palette[256];

// change the entrie palette
lpddpal -> SetEntries (0, 0, 256, new_palette);
```

3.6.19 Querying Palette Entries

```
HRESULT IDirectDrawPalette::GetEntries (DWORD dwFlags,
                                        DWORD dwStartingEntry,
                                        DWORD dwCount,
                                        LPPALETTEENTRY lpEntries);

dwFlags          : control flags เป็น 0 เสมอ
dwStartingEntry  : starting index to get
dwCount          : number of colors to retrieve
lpEntries        : pointer to data storage
```

ตารางที่ 3-20 GetEntries method

ตัวอย่างการใช้งาน

```
// data storage for palette here
PALETTEENTRY save_palette[256];

// save the palette
lpddpal -> GetEntries (0, 0, 256, save_palette);
```

3.6.20 การใช้ Blit method เพื่อ fill surface

```
HRESULT IDirectDrawSurface::Blt (LPRECT lpDestRect,
                                 LPDIRECTDRAWSURFACE3 lpDDSrcSurf,
                                 LPRECT lpSrcRect,
                                 DWORD dwFlags,
                                 LPDDBLTFX lpDDBltFx);
```

lpDestRect	: destination rectangle ถ้าเป็น NULL Destination surface ทั้งหมดจะถูกใช้
lpDDSrcSurf	: source surface
lpSrcRect	: source rectangle ถ้าเป็น NULL Source surface ทั้งหมดจะถูกใช้
dwFlags	: control flags
lpDDBltFx	: special fx structure

ตารางที่ 3-21 Blit method

dwFlags	: DDBLT_COLORFILL	ใช้ dwFillColor ใน DDBLTFX เป็น RGB color เพื่อ fill destination rectangle
	: DDBLT_DDFX	
	: DDBLT_DDROPS	
	: DDBLT_KEYDEST	ใช้ color key ที่สัมพันธ์กับ destination surface
	: DDBLT_KEYSRC	ใช้ color key ที่สัมพันธ์กับ source surface
	: DDBLT_ROP	
	: DDBLT_ROTATIONANGLE	
	: DDBT_WAIT	

```
typedef struct _DDBLTFX {
        DWORD dwSize;                   // the size of the structure in bytes
        DWORD dwDDFX;                   // type of blitter fx
        DWORD dwROP;                    // Win32 raster ops that are supported
        DWORD dwDDROP;                  // DirectDraw raster ops that are supported
        DWORD dwRotationalAngle;        // angle for rotations
        DWORD dwZBufferOpCode;          // z – buffer fieldsused for 3D
```

ต่อ...

```
DWORD dwZBufferLow;                    // advanced parameter
DWORD dwZBufferHigh;                   // advanced parameter
DWORD dwZBufferBaseDest;               // advanced
union
{
    DWORD dwZDestConst;                // advanced
    LPDIRECTDRAWSURFACE lpDDSZBufferDest;        // advanced...
};
DWORD dwZSrcConstBitDepth;             // advanced...
union
{
    DWORD dwZSrcConst;                 // advanced...
    LPDIRECTDRAWSURFACE lpDDSZBufferSrc;// advanced...
};
DWORD dwAlphadEdgeBlendBlitDepth;      // alpha stuff (advanced)
DWORD dwAlphaEdgeBlend;                // advanced...
DWORD dwReserved;                      // advanced...
DWORD dwAlphaDestConstBitDepth;        // advanced...
union
{
    DWORD dwAlphaDestConst;            // advanced...
    LPDIRECTDRAWSURFACE lpDDSAlpaDest; // advanced...
};
DWORD dwAlphaSrcConstBitDepth;         // advanced...
union
{
    DWORD dwAlphaSrcConst;             // advanced...
    LPDIRECTDRAWSURFACE lpDDSAlphaSrc; // advanced...
};
```

ต่อ...

```
        union
        {
            DWORD dwFillColor;          // color word used for fill
            DWORD dwFillDepth;          // z filling (advanced)
            DWORD dwFillPixel;          // color fill word for RGB(alpha) fills
            LPDIRECTDRAWSURFACE lpDDSPattern;
        };
        // the following aer very important
        DDCOLORKEY ddckDestcolorkey;    // destination color key
        DDCOLORKEY ddckSrcColorkey;     // Source color key
    } DDBLTFX, FAR* LPDDBLTFX;
```

ตารางที่ 3-22 DDBLTFX structure

ตัวอย่างการใช้งาน

```
DDBLTFX     ddbltfx;        // this contain the DDBLTFX structure
RECT        fill_area;      // this contain destination rectangle

// clear out the structure and set the size field
memset (&ddbltfx, 0, sizeof (DDBLTFX));
ddbltfx.dwSize = sizeof (DDBLTFX);

// set the dwfillcolor field to the desired color
ddbltfx.dwFillColor = color;    // data 8, 16, 24

// fill in the destination-rectangle data
fill_area.top = top;
fill_area.left = left;
fill_area.buttom = buttom;
fill_area.right = right;
```

```
// ready to blit surface;
lpddsprimary -> Blt(&fill_area,          // point to destination rectangle
                    NULL,                // point to source surface, NA
                    NULL,                // point to source rectangle, NA
                    DDBLT_COLORFILL | DDBLT_WAIT, // fill and wait
                    &ddbltfx);           // point to DDBLTFX structure
```

3.6.21 Transparency

```
HRESULT IDirectDrawSurface::SetColorKey (DWORD dwFlags,          // what kind of key
                                         LPDDCOLORKEY lpDDColorKey);
                                                                 // range of key
```

ตารางที่ 3-23 SetColorKey method

dwFlags : DDCKEY_SRCBLT
: DDCKEY_DESTBLT

```
typedef struct _DDCOLORKEY {
                DWORD dwColorSpaceLowValue;     // starting color (inclusive)
                DWORD dwColorSpaceHighValue;    // ending color (inclusive)
            } DDCOLORKEY, FAR *LPDDCOLORKEY;
```

ตารางที่ 3-24 DDCOLORKEY structure

ตัวอย่างการใช้งาน

```
DDCOLORKEY      key;            // color key
// set transparent color range to 0
key.dwColorSpaceLowValue = 0;
key.dwColorSpaceHighValue = 0;
```

```
// set color key now on the source surface, which is usually the backbuffer
lpddsback -> SetColorKey ( DDCKEY_SRCBLT, &key);
// perform the blit from backbuffer to primary buffer
lpddsprimary -> Blt (&dest_rect, lpddsback, &source_rect,
                        DDBLT_KEYSRC | DDBLT_WAIT, NULL);
```

ตัวอย่างการใช้งาน

```
DDCOLORKEY          key;            // color key
// set writable values from 0 to 249
key.dwColorSpaceLowValue = 0;
key.dwColorSpaceHighValue = 249;
// set color key now on the destination surface
lpddsprimary -> SetColorKey (DDCKEY_DESTBLT, &key);
// perform the blit from backbuffer to primary buffer
lpddsprimary -> Blt (...);
```

3.6.22 DirectDraw Clipper

```
HRESULT IDirectDraw::CreateClipper (DWORD dwFlags,
                                    LPDIRECTDRAWCLIPPER FAR *lplpDDClipper,
                                    IUnknow FAR *pUnkOuter);
```

dwFlags	: control flags เป็น 0 เสมอ
lplpDDClipper	: pointer ที่ชี้ไปที่ DirectDrawClipper interface
pUnkOuter	: ไม่ใช้ set ให้เป็น NULL

ตารางที่ 3-25 CreateClipper method

ตัวอย่างการใช้งาน

```
LPDIRECTDRAWCLIPPER     lpDDClipper;        // clipper
if ((lpdd -> CreateClipper (0, &lpddclipper, NULL)) != DD_OK)
    return (NULL);
```

```
typedef struct _RGNDATA {                    // rgnd
        RGNDATAHEADER rdh;                   // the header
        char Buffer[1];                      // a list of RECTs defining clipping
    } RGNDATA;
```

ตารางที่ 3-26 RGNDATA structure

```
typedef struct _RGNDATAHEADER {              // rgndh
        DWORD dwSize;                        // size of this header
        DWORD iType;                         // must be RDH_RECTANGLES
        DWORD nCount;                        // number of rectangles in buffer
        DWORD nRgnSize;                      // size of the buffer
        RECT rcBound;                        // a bounding box around all the rects
    } RGNDATAHEADER;
```

ตารางที่ 3-27 RGNDATAHEADER structure

```
HRESULT IDirectDrawClipper::SetClipList (LPRGNDATA lpClipList,
                                         DWORD dwFlags);

lpClipList          : pointer ที่ชี้ไปที่ RGNDATA
dwFlags             : control flags เป็น 0 เสมอ
```

ตารางที่ 3-28 SetClipList method

ตัวอย่างการใช้งาน

```
RGNDATA region_data;                    // holds the RECTs and data header
// fill in region_data ...
// set clipping list
if ((lpddclipper -> SetClipList (region_data, 0)) != DD_OK)
    {/* error */}
```

HRESULT IDirectDrawSurface::SetClipper (LPDIRECTDRAWCLIPPER lpDDClipper);	
lpDDClipper	: pointer ชี้ไปที่ DirectDrawClipper

ตารางที่ 3-29 SetClipper method

ตัวอย่างการใช้งาน

```
if ((lpdds -> SetClipper (lpddclipper)) != DD_OK)
    {/* error */}
```

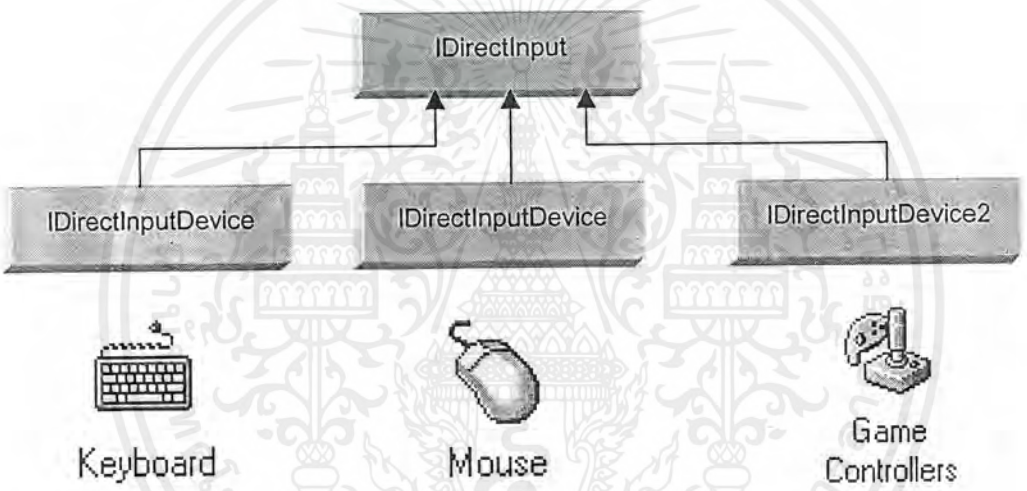
บทที่ 4

DirectInput

สถาปัตยกรรมพื้นฐานของ DirectInput ประกอบด้วย DirectInput object ซึ่งรองรับ COM interface และ object สำหรับ input device แต่ละตัวที่ส่งข้อมูลเข้ามา โดย device แต่ละอันจะมี "object instances" ของมันเอง ซึ่งแต่ละตัวจะมีการควบคุม ที่แตกต่างกัน เช่น ปุ่ม, แกน เป็นต้น

คำว่า "object" ในที่นี้ใช้แสดงสิ่งต่างๆ ที่สร้างโดย ระบบ DirectInput ที่รองรับ method ของ COM interface โดย method เหล่านี้ไม่ได้ถูกใช้เรียกผ่าน OOP เช่น ภาษา VC++

ในด้านความเร็ว DirectInput ทำงานโดยตรงกับ อุปกรณ์ต่างๆ โดยผ่านระบบ message ของ Windows



รูปที่ 4-1 การติดต่อของ DirectInput

4.1 ขั้นตอนการสร้าง Direct Input

1. **สร้าง Direct Input** ขึ้นมา โดยเรียกฟังก์ชัน DirectInputCreate โดยการ return pointer ไป IDirectInput Interface เพื่อสร้าง object
2. **Enumerating Device** คือการร้องขอ Direct Input ให้หา input device ที่ต่ออยู่

Note: 1. Mouse และ keyboard อาจไม่ต้องใช้

2. Joystick จำเป็นเพราะ joystick มีหลายชนิด ทำให้ GUID ไม่ว่างพอเหมือน mouse และ keyboard เราต้อง query ว่า joystick ที่ต่ออยู่ แล้วรับ GUID มาแล้วจึงใช้ ID เหล่านี้

การ Enumerate device ใดๆ จะเรียก callback function ก่อนแล้วจึงเรียก EnumDevice method

```
HRESULT EnumDevices( DWORD dwDevType,                    //ชนิดของ device ที่หา
                     LPDIENUMCALLBACK lpCallback,        //address ของ callback function
                     LPVOID pvRef,                       //ค่า 32-bit ที่ใช้ส่งกลับ
                     DWORD dwFlags );                    //flag ว่าจะ enumerate แบบไหน
```

3. **Setting up Device**
   - Create Device: สร้าง input device ต้องมี GUID สำหรับแต่ละ device โดยอาจจะร้องขอจาก Direct Input Enumerate

     Note: ถ้าไม่ได้ enumerate จาก guidInstance ของ DIDEVICEINSTANCE structure ที่ DirectInput ส่งมายัง callback function ของเรา เราจะต้องกำหนดเองก่อน เช่น GUID_SysMouse or GUID_SysKeyboard
   - Setting Data Format: ต้องรู้โครงสร้างของข้อมูลที่มาจาก device ต่างๆ เพราะ device ต่างๆ มีขนาดข้อมูลไม่เท่ากัน
   - Getting Data Format about a device
   - Identify Device Things: มี 2 แบบ คือ โดย Offset กับโดย ID
   - Set Property of device
   - Set Cooperative level: set เป็น DISCL_NONEXCLUSIVE | DISCL_BACKGROUND

4. **Acquire** การอนุญาตให้ใช้ device และแจ้ง Direct Input ว่าต้องการข้อมูลจาก device ตามรูปแบบของข้อมูลที่กำหนดไว้
5. **Unacquire** การยกเลิกการใช้ Device จาก Direct Input
6. **Release** การยกเลิกการใช้ Direct Input

Note: Poll สำหรับ Joystick โดย Direct Input จะตรวจสอบว่า device ต้อง polling หรือไม่ ถ้าต้องการจะเก็บ state ที่เปลี่ยนไป(โดย Hardware Interrupt) ทันที

4.2 แนวคิดในการโปรแกรม DirectInput แบบ OOP

DInput มี 5 classes

CInput

CDevice

CKeyboard, CMouse, CJoystick } Inherit มาจาก CDevice

4.2.1 CInput

Method

- HRESULT CreateInput(void *hinst); /*จะสร้าง Direct Input Object ขึ้น*/
- LPDIRECTINPUT GetDI(void) {return lpdi;} /*จะ return ค่า lpdi ออกมา*/

Attribute

- LPDIRECTINPUT lpdi;

4.2.2 CDevice

Method

- HRESULT Create(CInput*, REFGUID); /* สร้าง Device Object ขึ้นมา*/
- HRESULT SetDataFormat(LPCDIDATAFORMAT); /*set รูปแบบของข้อมูล ของ device*/
- HRESULT SetCooperativeLevel(void*, DWORD); /*set cooperative level*/
- HRESULT RunControlPanel(void*); /* run windows control panel เพื่ออนุญาตให้ผู้ใช้ติดตั้ง input device ตัวใหม่ หรือเพื่อทำการแก้ไข*/
- HRESULT Acquire(void); /* acquire device*/
- HRESULT Unacquire(void); /* unacquire device*/
- HRESULT SetRelative(void); /* set ข้อมูล device ที่เป็น relative*/
- HRESULT SetAbsolute(void); /* set ข้อมูล device ที่เป็น absolute*/

Attribute

- LPDIRECTINPUTDEVICE2 lpdi2;
- BOOL m_bActive; /*เก็บ boolean ว่า method ของ Direct Input ว่าเป็น true,false*/

4.2.3 CKeyboard

Method

- HRESULT CreateKeyboard(CInput*, void*); /*สร้าง Keyboard device*/
- void GetState(void); /*รับ ข้อมูลจาก Keyboard*/

Attribute

- BYTE keyboard_state[256]; /*ที่เก็บ state ของ Keyboard*/

4.2.4 CMouse

Method

- HRESULT CreateMouse(CInput*, void*); /*สร้าง mouse device*/
- void GetState(void); /*รับ ข้อมูลจาก mouse*/
- LONG GetX(void) {return m_X;} //return x-position mouse
- LONG GetY(void) {return m_Y;} //return y-positon mouse
- LONG GetZ(void) {return m_Z;} //returnn z-position mouse(wheel)
- BYTE GetLB(void) {return m_Buttons[0];} //return left button
- BYTE GetRB(void) {return m_Buttons[1];} //retrun right button
- BYTE GetMB(void) {return m_Buttons[2];} //return middle button

Attribute

- LONG m_X,m_Y,m_Z;
- BYTE m_Buttons[4];

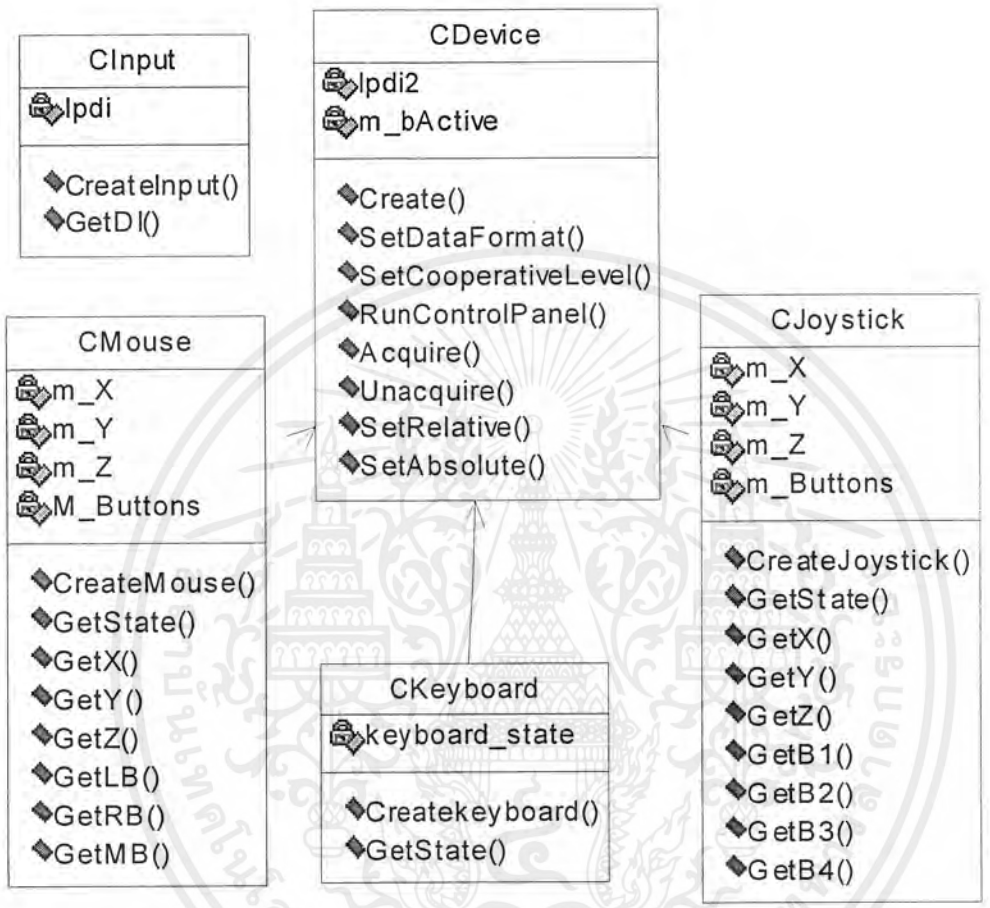
4.2.5 CJoystick

Method

- HRESULT CreateJoystick(CInput*, void*); /*สร้าง mouse device*/
- void GetState(void); /*รับ ข้อมูลจาก mouse*/
- LONG GetX(void) {return m_X;} //return x-position joy
- LONG GetY(void) {return m_Y;} //reyurn y-position joy
- LONG GetZ(void) {return m_Z;} //return z-position joy
- BYTE GetB1(void) {return m_Buttons[0];} //return button1
- BYTE GetB2(void) {return m_Buttons[1];} //return button2
- BYTE GetB3(void) {return m_Buttons[2];} //return button3
- BYTE GetB4(void) {return m_Buttons[3];} //return button4

Attribute

- LONG m_X,m_Y,m_Z;
- BYTE m_Buttons[32];



รูปที่ 4-2 class diagram ของ DirectInput

วิธีการเรียกใช้ Object ที่สร้างสำหรับ Direct Input

Declare pointer to Class

```
CInput* input_t;
CDevice* device_t;
CKeyboard* keyboard_t;
CMouse* mouse_t;
CJoystick* joystick_t;
```

} Pointer ไปยัง class ต่างๆ

Allocate memory for pointer

```
input_t = new CInput;           //allocate object CInput
device_t = new Cdevice;         //allocate object Cdevice
keyboard_t = new CKeyboard;     //allocate object CKeyboard
mouse_t = new CMouse;           //allocate object CMouse
joystick_t = new CJoystick;     //allocate object Cjoystick
```

Access method

Keyboard:

keyboard_t -> CreateKeyboard(input_t,&hwnd); //สร้าง object keyboard และ set ค่าต่างๆ ที่เกี่ยวกับ keyboard ทั้งหมด <เรียกใช้ใน Game_Init();>

keyboard_t -> GetState(); //เป็นการรับข้อมูลจาก Keyboard แล้วเก็บไว้ใน keyboard_state <เรียกใช้ใน Game_Main();>

Mouse:

Mouse_t -> CreateMouse(input_t,&hwnd); //สร้าง object mouse และ set ค่าต่างๆ ที่เกี่ยวกับ mouse ทั้งหมด <เรียกใช้ใน Game_Init();>

```
mouse_t -> GetState();  //เป็นการรับข้อมูลจาก mouse
mouse_t -> GetX(); //ส่ง ค่า mouse ในแกน X มา โดยตัวแปร m_X
mouse_t -> GetY(); //ส่ง ค่า mouse ในแกน Y มา โดยตัวแปร m_Y
mouse_t -> GetZ(); //ส่ง ค่า mouse ในแกน Z <wheel>มา โดยตัวแปร m_Z
mouse_t -> GetLB(); //ส่งค่าในปุ่มซ้าย โดยตัวแปร m_Buttons[0]
mouse_t -> GetRB(); //ส่งค่าในปุ่มขวา โดยตัวแปร m_Buttons[1]
mouse_t -> GetMB(); //ส่งค่าในปุ่มกลาง โดยตัวแปร m_Buttons[2]
```

} เรียกใช้ใน Game_Main();

Joystick:

Joystick_t -> CreateJoystick(input_t,&hwnd); //สร้าง object joystick และ set ค่าต่างๆ ที่เกี่ยวกับ joystick ทั้งหมด <เรียกใช้ใน Game_Init();>

joystick_t -> GetState(); //เป็นการรับข้อมูลจาก joystick

joystick_t -> GetX(); //ส่ง ค่า joystick ในแกน X มา โดยตัวแปร m_X

joystick_t -> GetY(); //ส่ง ค่า joystick ในแกน Y มา โดยตัวแปร m_Y

joystick_t -> GetZ(); //ส่ง ค่า joystick ในแกน Z มา โดยตัวแปร m_Z

joystick_t -> GetB1(); //ส่งค่าในปุ่มซ้าย โดยตัวแปร m_Buttons[0]

joystick_t -> GetB2(); //ส่งค่าในปุ่มขวา โดยตัวแปร m_Buttons[1]

joystick_t -> GetB3(); //ส่งค่าในปุ่มกลาง โดยตัวแปร m_Buttons[2]

joystick_t -> GetB4(); //ส่งค่าในปุ่มกลาง โดยตัวแปร m_Buttons[3]

} เรียกใช้ใน Game_Main();

Check Button

Keyboard:

If (keyboard_t -> keyboard_state[DIK_RIGHT]) { /*move left*/ }

If (keyboard_t -> keyboard_state[DIK_SPACE]) { /*open door*/ }

If (keyboard_t -> keyboard_state[DIK_RCONTROL]) { /*fire*/ }

Mouse:

If (mouse_t -> m_X > 0) { /*move right*/ }

If (mouse_t -> m_X < 0) { /*move left*/ }

If (mouse_t -> m_Y > 0) { /*move up*/ }

If (mouse_t -> m_Y < 0) { /*move down*/ }

If (mouse_t -> m_Buttons[0]) { /* fire */ }

Joystick:

If (joystick_t -> m_X > 0) { /*move right*/ }

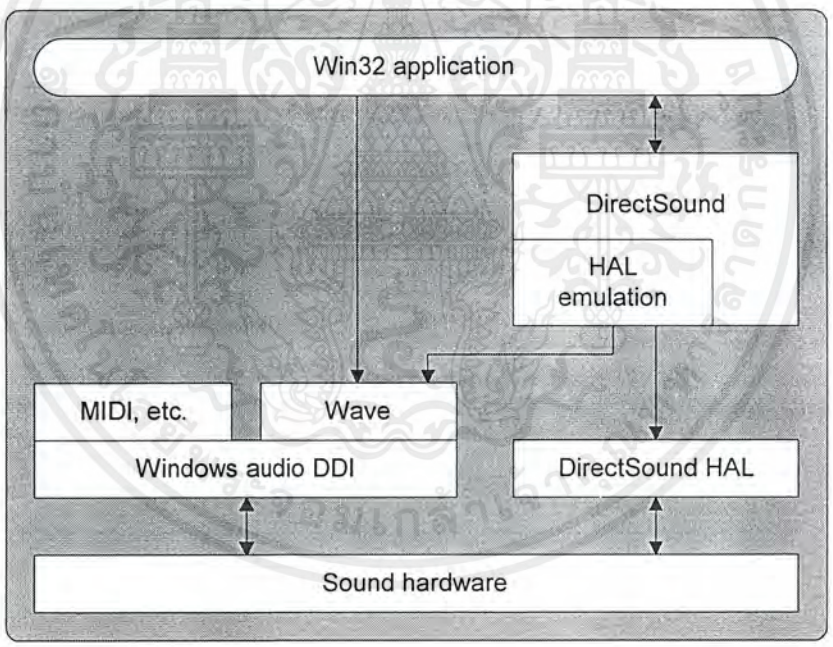
If (mouse_t -> m_Buttons[0]) { /*fire*/ }

บทที่ 5

DirectSound

ถ้าเราต้องการเพียงเสียง Background และมีเสียงอื่นๆ เข้ามา เราสามารถใช้คำสั่ง PlaySound ของ Microsoft ได้ แต่ในระบบของเกมที่แท้จริงยังต้องการ tools ที่มีประสิทธิภาพมากกว่านั้น โดย DirectSound ได้เสนอสิ่งต่างๆ เหล่านี้ได้แก่

- การเร่งความเร็ว โดยอัตโนมัติ ถ้ามีฮาร์ดแวร์เหล่านั้นอยู่
- การรวมเสียงต่างๆ เข้าด้วยกัน อย่างไม่จำกัด
- การกำหนดตำแหน่งของเสียง 3 มิติ โดย Direct3D
- การแปลงข้อมูลอินพุตของเสียง ในรูปแบบต่างๆ หลายรูปแบบ ให้ได้เอาท์พุตที่ตรงกัน
- รองรับคุณสมบัติ(Property set) ใหม่ๆที่มีเข้ามาของฮาร์ดแวร์ โดยไม่ต้องเปลี่ยน API
- การเล่นข้อมูลเสียงใช้ทรัพยากร(resouce) ต่ำ

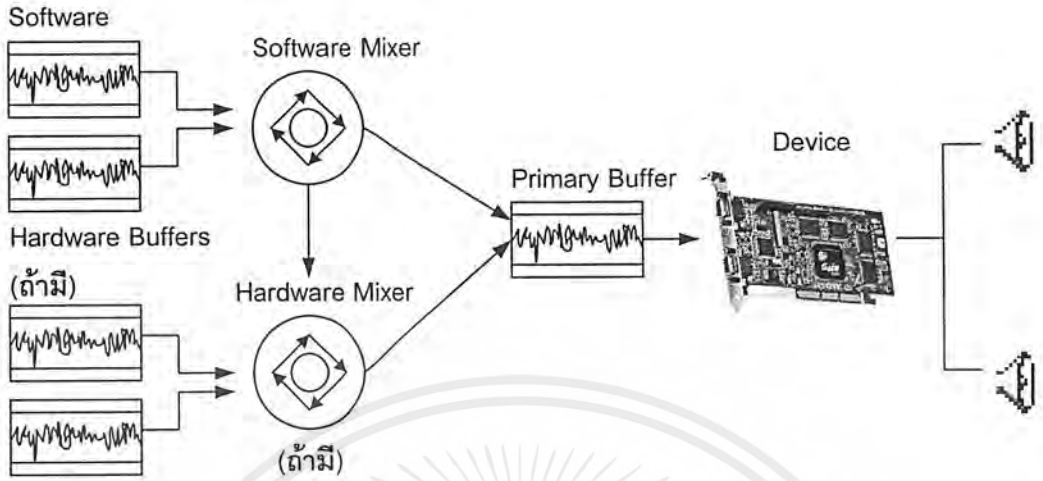


รูปที่ 5-1 สถาปัตยกรรมของ DirectSound

การเรียกใช้ DirectSound ประกอบด้วย 3 ส่วน

1. ขณะ run-time .DLL(Device Driver Interface) จะถูกโหลดเข้าไปเมื่อใช้ DirectSound
2. ขณะ compile-time ใช้ library DSOUND.LIB
3. เรียก HEADER ที่ชื่อ DSOUND.H

5.1 การทำงานของ DirectSound



รูปที่ 5-2 การทำงานของ DirectSound

DirectSound จะเริ่มด้วย *secondary sound buffer* object ซึ่งแทนเสียงเพียงเสียงเดียว เสียงเหล่านี้อาจเป็น static sound หรือ streaming sound ก็ได้

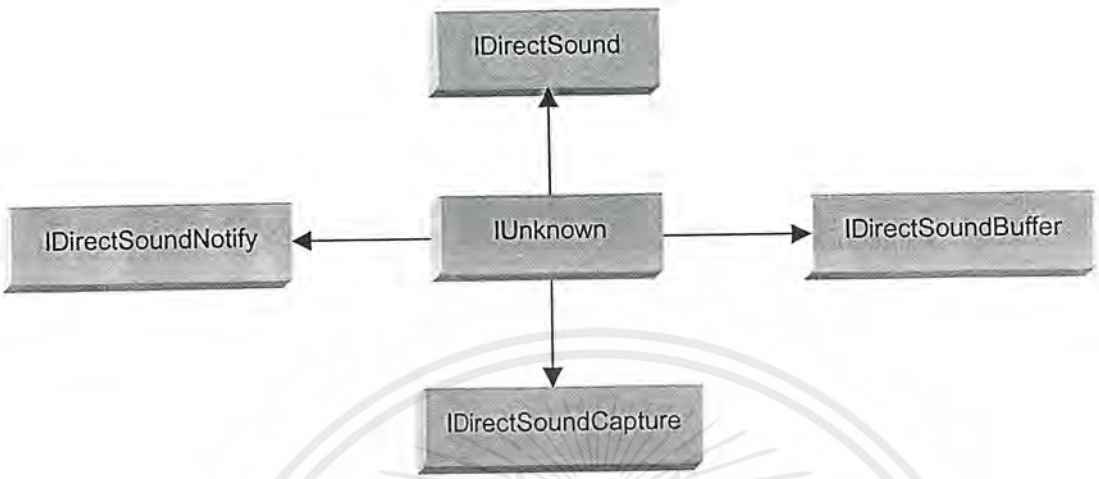
Static sound คือ เสียงสั้นๆ ที่มีขนาดข้อมูลที่พอดีกับหน่วยความจำ

Streaming sound คือ ข้อมูลเสียงส่วนหนึ่ง ที่ย้ายมาจากหน่วยความจำมาเก็บไว้ใน buffer โดย buffers ทั้งหมดจะถูกเก็บข้อมูลเสียงในรูปแบบของ pulse code modulation (PCM)

เมื่อเริ่มเล่น Secondary buffer, DirectSound จะใช้ข้อมูลจาก buffer แต่ละอัน แล้วรวม(mix) เข้าด้วยกันใน *primary buffer* โดยตัวมันจะแปลงรูปแบบของข้อมูลเท่าที่จำเป็น ตัวอย่างเช่น แปลง sampling rate จาก 44Hz เป็น 22Hz และเพิ่ม effect ต่างๆ เข้าไป เช่น ตำแหน่งของเสียงในทาง 3 มิติ

DirectSound จะแทนที่ buffers ต่างๆ ในหน่วยความจำหลักด้วย Hardware buffers และ Hardware mixing (ถ้ามี) โดยอัตโนมัติ

5.2 การติดต่อของ DirectSound



รูปที่ 5-3 การติดต่อของ DirectSound

IUnknown: คือ COM object หลักของทุก COM objects

IDirectSound: คือ COM object หลักของ DirectSound ใช้แทน ฮาร์ดแวร์ ทางด้านเสียงของมันเอง การ์ดเสียง 1 อัน แทน 1 object

IDirectSoundBuffer: คือ ฮาร์ดแวร์ที่รวมข้อมูล ด้านเสียงเข้าด้วยกัน DirectSound มี buffer อยู่ 2 ชนิดด้วยกัน: *primary buffer* คือ เสียงที่กำลังเล่นอยู่ และรวมเข้าด้วยกันโดย ฮาร์ดแวร์(ถ้ามี) หรือ ซอฟต์แวร์ *secondary buffer* แทนเสียงที่ถูกเก็บไว้เพื่อจะเล่น (playback) อาจเก็บไว้ในหน่วยความจำระบบ หรือใน SRAM บนการ์ดเสียง

IDirectSoundCapture: การรับข้อมูลเสียงเข้ามา เช่น การรู้จำเสียง(voice-recognition)

IDirectSoundNotify: การส่ง message กลับไป ให้ DirectSound

รูปแบบข้อมูลเสียง

เราสามารถกำหนดรูปแบบของข้อมูลเสียงได้ใน โครงสร้างของ WAVEFORMATEX ดังนี้

- จำนวน Channels (1 คือ mono, 2 คือ stereo)
- sampling rate ในรูปของหน่วย Hz ต่อ 1 channel ดังนั้นถ้าข้อมูลเสียงของเราเป็นแบบ stereo ค่า sample rate ต้องเป็น 2 เท่าจากเดิม
- จำนวนบิทต่อวินาที(bits per sample) โดยปกติจะเป็น 8 หรือ 16 บิท
- Format tag ซึ่งจะเป็นตัวระบุว่าข้อมูลจะแปลงไปอย่างไร สำหรับ DirectSound จะใช้ WAVE_FORMAT_PCM tag

5.3 การเข้าถึง DirectSound

การเข้าถึงระบบ DirectSound ต้องทำตามขั้นตอนต่อไปนี้

1. รับ GUID สำหรับอุปกรณ์ด้านเสียง
2. สร้าง object ของ DirectSound (จำเป็น)
3. กำหนด Cooperative level (จำเป็น)
4. กำหนดรูปแบบของ primary buffer

- Enumerating Output Devices เป็นการหาว่า ฮาร์ดแวร์ว่าเป็นอะไร แล้วส่งข้อมูลของฮาร์ดแวร์นั้นไป (ไม่จำเป็น) เราสามารถส่งค่า NULL ไปให้ DirectSound เพื่อกำหนดให้อุปการณ์ของเราเป็น default

- การสร้าง DirectSound object

  ตัวอย่าง

  ```
  LPDIRECTSOUND lpds;
  if (DirectSoundCreate(NULL, &lpds, NULL)!= DS_OK) {/* error */}
  ```

  Note: ในทางปฏิบัติเป็นไปได้ที่จะสร้าง DirectSound objects ไว้หลายๆ อัน แต่เมื่อเรากำหนด Cooperative level ใหม่ใน object ใหม่ ค่า Cooperative level จะไปทับของเดิม

- กำหนด Cooperative level

  Normal cooperation เล่นเสียงได้แต่เขียน primary buffer ไม่ได้

  Priority cooperation เข้าถึง ฮาร์ดแวร์ได้หมด เขียน primary buffer ได้

  Exclusive cooperation เหมือน priority แต่ application จะไม่ทำงานเมื่อ application minimize

  Write_Primary cooperation ทำได้ทุกอย่าง

  ตัวอย่าง

  ```
  // set cooperation level
  if (lpds->SetCooperativeLevel(main_window_handle,DSSCL_NORMAL)!=DS_OK)
  ```

- กำหนดรูปแบบของ Primary

  ทุกๆ เสียงจะต้องเล่นภายใน Primary buffer เราจะต้องแน่ใจว่ารูปแบบของ Primary buffer จะเข้ากันได้กับ Secondary buffer โดย DirectSound จะทำงานได้ดีที่สุดที่ 16-bits per sample

  ถ้าต้องกรเปลี่ยนรูปแบบ Primary buffer อย่างน้อยต้องกำหนด Cooperative level เป็น DSSCL_PROIORITY

- การเปลี่ยนตำแหน่งของลำโพง

  เราสามารถกำหนดรูปแบบการตั้งลำโพงได้ โดยใช้ GetSpeakerConfig method

  ตัวอย่าง

  lpds -> SetSpeakerConfig (DSS_SPEAKER_COMBINED(

  DSS_SPEAKER_STEREO, DSS_GEOMETRY_WIDE));

  /*20 degrees from MINIMUM(MINIMUM is 90 degrees from ear)*/

- การประเมินความสามารถของฮาร์ดแวร์

  เราสารมารถรู้ถึง ความสามารถของฮาร์ดแวร์ได้โดยใช้ GetCaps method ซึ่งถ้าเรามี ฮาร์ดแวร์ที่ดีอยู่แล้ว การทำงานส่วนนี้สามารทำได้โดยฮาร์ดแวร์

- Property Sets

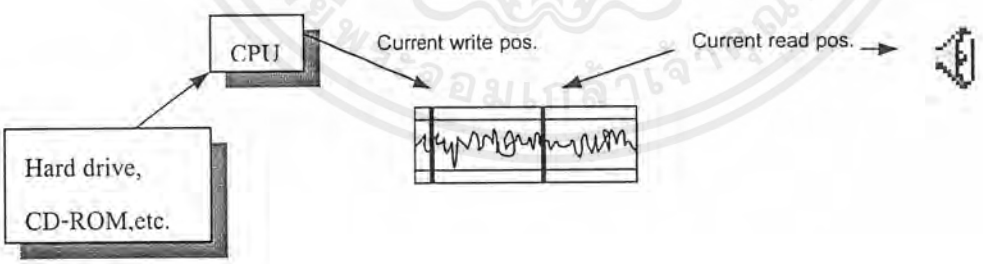
  มีไว้เพื่อรองรับ featuresใหม่ๆ ของการ์ดเสียง โดยคุณสมบัติ(properties) เหล่านี้ จะถูกรวมไว้โดย GUID และแต่ละ property จะมี หมายเลข index ของมันเอง ผู้ผลิตก็จะออก header files ที่มีการค่าเหล่านี้ออกมา เมื่อต้องการใช้ก็ไป query มา

5.4 ตำแหน่งของการเขียน และอ่านข้อมูลบน Buffer

Buffer ทางด้านเสียง แบ่งเป็น

Static sound buffer: เสียงทั้งหมดจะนำมาเก็บไว้ใน buffer เหมาะกับ SRAM

Streaming sound buffer: ข้อมูลของเสียงชนิดนี้มีความยาวมาก streaming sound อาจเป็นเสียงที่เล่นจาก CD ทำให้ไม่มีหน่วยความจำพอ ต้องอ่านมาทีละบล็อก(block) ของหน่วยความจำ เรียกว่า chunk ลักษณะของ buffer จะเป็น buffer circular, ซึ่งเหมือนกับเทปเพลง นั่นคือถ้า เล่นเสียงแบบต่อเนื่องบน buffer เมื่อเล่นจนถึงท้าย buffer มันจะย้อนกลับมาเล่นใหม่ ดังนั้นจึงจำเป็นที่จะต้องมี pointer ไว้ 2 ตัว



รูปที่ 5-4 Streaming audio data

DirectSound จะเก็บ pointers ของทุก sound buffer ไว้ คือ ตำแหน่งที่เล่น (current play position) และตำแหน่งที่อ่าน (current write position)

Note: Current play position: คือ ค่า offset ใน buffer เพื่อให้ DirectSound นำข้อมูลตั้งแต่นั้น ไปเป็นข้อมูลในการ mix เพื่อเล่นเสียง

Current write position: คือ ค่า offset ใน buffer เพื่อให้ DirectSound นำข้อมูลตั้งแต่นั้น ไปเป็นข้อมูลในการเขียนข้อมูลลงไป ตั้งแต่ตำแหน่งนี้เป็นต้นไป เป็นตำแหน่งที่ปลอดภัยที่จะเขียนข้อมูลลง buffer

5.5 การทำให้ Buffer ที่ใช้ให้มีประสิทธิภาพ

- Static buffer sound และ Streaming buffer sound

  Static secondary buffer ควรจะเป็นเสียงที่มีความยาวสั้นๆ เสียงเดียว ที่ต้องเล่นซ้ำไปมา ส่วน streaming buffer จะเหมาะกับข้อมูลเสียงที่ขนาดไม่พอกับ buffer เนื่องจากขนาดใหญ่ไปต้อง copy เป็นส่วนๆ ไปเก็บไว้ใน buffer

  CreateSoundBuffer method จะกำหนด default สร้าง streaming buffer ไว้แล้ว และถ้าต้องการสร้าง static buffer ก็ต้องกำหนด flag เป็น DSBCAPS_STATIC

- Hardware และ Software

  ในการ์ดเสียงบางรุ่น จะมีความสามารถในการ mix เสียงในตัวมันเอง buffer จึงสามารถใช้หน่วยความจำจากฮาร์ดแวร์ได้ แต่ถ้าไม่มีก็จะใช้หน่วยความหลักของระบบแทน ถ้าเราต้องการ buffer จากการ์ดเสียงอย่างเดียวกำหนด flag เป็น DSBCAPS_LOCHARDWARE ในกรณีนี้ ถ้าไม่มีหน่วยความจำในการ์ดเสียงจะ fail หรือ ถ้าต้องการ buffer จากหน่วยความจำหลักอย่างเดียวก็กำหนด flag เป็น DSBCAPS_LOCSOFTWARE ซึ่งเหมาะกับ streaming buffer และ การเล่นเสียงเดียวกันพร้อมกัน (Duplicate sound)

- Hardware และ Software Mixing

  โดยปกติการ mix เสียงจะเป็นหน้าที่หลักของ การ์ดเสียง แต่ถ้าการ์ดเสียงไม่รองรับDirectX DirectSound จะ mix เสียงโดยใช้หน่วยความจำหลักแทน

- Stick และ Global Focus

  เมื่อสร้าง buffer ขึ้นมาจะมีการ กำหนดว่าเมื่อ minimize โปรแกรม หรือไปเรียกใช้โปรแกรมอื่นจะได้ยินเสียงที่เล่นหรือไม่ ถ้า set buffer เป็น global focus จะได้ยิน ถ้า set เป็น sticky focus จะไม่ได้ยิน (default)

- Control

  เมื่อสร้าง secondary buffer ขึ้นมาเราสามารถทำ feature เหล่านี้ได้

  - 3D Properties
  - ความถี่ (SetFrequency)
  - Pan (SetPan)
  - ความดัง (SetVolume)
  - Position notification

- การหลีกเลี่ยง overhead ที่ไม่จำเป็น

- อย่าใช้ Control ด้านบนถ้าไม่จำเป็น หลีกเลี่ยง flag DSBCAPS_CTRALL หรือ DSBCAPS_CRLDEFAULT
- เก็บ secondary buffer ไว้ในรูปแบบเดียวกัน และถ้าจำเป็นก็เปลี่ยนรูปแบบให้ตรงกับ primary buffer
- อย่าสร้าง 3D SoundBuffer ถ้าไม่จำเป็น

5.6 การสร้าง Buffer

ขั้นตอนของการสร้าง buffer และเล่นเสียงเป็นดังนี้

1. กำหนดค่าเริ่มต้น ในโครงสร้างข้อมูล WAVEFORMATEX เพื่อกำหนดรายละเอียดข้อมูลเสียง

ตัวอย่าง

```
WAVEFORMATEX        pcmwf;              // generic waveformat structure

DWORD    dsbstatus;                     // status of sound buffer

DWORD    audio_length_1   = 0,          // length of locked memory
         audio_length_2   = 0,
         snd_buffer_length = 64000;     // working buffer

// allocate memory for buffer
UCHAR *snd_buffer_ptr = (UCHAR *)malloc(snd_buffer_length);
// set up the format data structure
memset(&pcmwf, 0, sizeof(WAVEFORMATEX));

pcmwf.wFormatTag  = WAVE_FORMAT_PCM;
pcmwf.nChannels = 1;                    //mono sound
pcmwf.nSamplesPerSec  = 11025;          //11 KHz
pcmwf.nBlockAlign = 1;
pcmwf.nAvgBytesPerSec = pcmwf.nSamplesPerSec * pcmwf.nBlockAlign;
pcmwf.wBitsPerSample = 8;
pcmwf.cbSize = 0;
```

2. กำหนดค่าเริ่มต้น ในโครงสร้างข้อมูล DSBUFFERDESC ด้วยพารามิเตอร์ของ buffer รวมทั้ง pointer ไปที่ WAVEFORMATEX

ตัวอย่าง

```
// create the secondary buffer (no need for a primary)
```

```
memset(&dsbd,0,sizeof(DSBUFFERDESC));
dsbd.dwSize = sizeof(DSBUFFERDESC);
dsbd.dwFlag = DSBCAPS_CTRLDEFAULT | DSBCAPS_STATIC | DSBCAPS_LOCSOFTWARE;
dsbd.dwBufferBytes = snd_buffer_length+1;
dsbd.lpwfxFormat = &pcmwf;
```

3. เรียก CreateSoundBuffer method เพื่อสร้าง buffer

   ตัวอย่าง

```
if (lpds->CreateSoundBuffer(&dsbd,&lpdsbsecondary,NULL)!=DS_OK)
```

4. Lock ตำแหน่งของ buffer

   ตัวอย่าง

```
// copy data into sound buffer
if (lpdsbsecondary->Lock(0,
                         snd_buffer_length,
                         &audio_ptr_1,
                         &audio_length_1,
                         &audio_ptr_2,
                         &audio_length_2,
                         DSBLOCK_FROMWRITECURSOR)!=DS_OK)
```

5. Copy ข้อมูล(จาก .wav files หรือ ข้อมูลเสียงแบบอื่น) ไปยัง buffer ตามตำแหน่งที่ Lock ไว้

   ตัวอย่าง

```
// copy first section of circular buffer
CopyMemory(audio_ptr_1, snd_buffer_ptr, audio_length_1);
// copy last section of circular buffer
CopyMemory(audio_ptr_2, (snd_buffer_ptr+audio_length_1),audio_length_2);
```

6. Unlock buffer

   ตัวอย่าง

```
if (lpdsbsecondary->Unlock(audio_ptr_1,
                           audio_length_1,
                           audio_ptr_2,
                           audio_length_2)!=DS_OK)
```

7. กำหนดตำแหน่งที่จะเล่น (Set play position) ถ้าจำเป็น
8. เล่นเสียง (Play) ถ้าต้องการให้เล่นเสียงรอบเดียว set flag =0 แต่ถ้าต้องการให้เล่นวนไม่รู้จบ set flag = DSBPLAY_LOOPING

ตัวอย่าง

if (lpdsbsecondary->Play(0,0,DSBPLAY_LOOPING)!=DS_OK)

สามารถหยุดเล่นได้โดยฟังก์ชั่น Stop()

9. ถ้าเป็น Streaming buffer ก็ทำขั้นตอนที่ 4-6 ซ้ำ

5.7 การกำหนด buffer

การกำหนด buffer มีส่วนที่สำคัญ 3 ส่วนดังนี้

- Flag ที่กำหนดว่าจะให้ buffer อยู่ที่ส่วนใด และทำงานแบบใด

  DSBCAPS_CTRL3D: ใช้เมื่อต้องการจะจัดการ buffer ด้วย IDirectSound3Dbuffer interface ใช้กับ DSBCAPS_CTRLPAN ไม่ได้

  DSBCAPS_CTRLALL: ใช้ควบคุม buffer ทั้งหมด

  DSBCAPS_CTRLDEFAULT: จะควบคุม buffer เฉพาะที่ต้องการ
- ขนาดของ buffer ถ้าเป็น static buffer ควรจะมีขนาดเท่ากับจำนวน bytes ของข้อมูลเสียงที่ใหญ่ที่สุด ที่ใช้เก็บ buffer แต่ถ้าเป็นstreaming buffer ควรมีขนาดใหญ่เพียงพอกับข้อมูลสำหรับเล่นเสียง 1-2 วินาที ซึ่งสามารถคำนวณได้จาก จำนวน bytes เฉลี่ยต่อวินาที (nAvgBytesPerSec)
- pointer ไปที่ WAVEFORMATEX

5.8 การ Duplicate ใน static buffer

เมื่อต้องการได้ยินเสียง หลายเสียงในเวลาเดียวกัน เช่น เสียงกระสุนที่ดังพร้อมกันแบบต่อเนื่อง สามารถทำได้โดยสร้าง Static buffers ขึ้นมาหลายอัน แล้วใส่ข้อมูลเสียงลงไป

5.9 การส่งข้อมูลให้เล่นเสียงใน Streaming buffer

ในการส่งข้อมูลใน Streaming buffer จะต้องรู้ว่าเมื่อไร ที่จะถึงเวลาที่จะส่งข้อมูลอีก การส่งข้อมูลทำได้ 2 ลักษณะ คือ

- Polling: จะเป็นการตรวจข้อมูลที่เล่นในขณะนั้น ทุกช่วงระยะเวลา (ตรวจทุกครั้งที่ผ่าน message loop) การใช้วิธีนี้จะใช้ cycle มากกว่าแต่ง่ายกว่าในการใช้
- Notification: รอ DirectSound ให้สัญญาณ ให้เล่นได้

5.10 แนวคิดในการโปรแกรม DirectSound แบบ OOP

DSound มี 3 classes

CSound

CWaveFile

CBuffer

5.10.1 CSound

Method

- HRESULT CreateSound(void hwnd); /*สร้าง DirectSound object*/
- LPDIRECTSOUND GetDS(void) {return lpds;} /*return lpds*/
- LPDIRECTSOUNDBUFFER GetPrimary(void) {return lpdsprimary;} /*return lpdsprimary*/

Attribute

- LPDIRECTSOUND lpds; //DirectSound interface pointer
- LPDIRECTSOUNDBUFFER lpdsprimary; //Long pointer ไปที่ DirectSound buffer สำหรับ primary buffer

5.10.2 CwaveFile

Method

- BOOL Open(LPSTR pszFilename); //เปิด file
- BOOL Cue(void);
- UINT Read(BYTE* pbDest, UINT cbSize); //อ่าน file
- UINT GetNumBytesRemaining(void) { return (m_nDataSize - m_nBytesPlayed);} //หาจำนวน bytes ที่เหลืออยู่
- UINT GetAvgDataRate(void) { return (m_nAvgDataRate); } //หาจำนวนข้อมูลเฉลี่ย
- UINT GetDataSize(void) { return (m_nDataSize); } //ขนาดข้อมูล
- UINT GetNumBytesPlayed(void) { return (m_nBytesPlayed); } //จำนวน bytes ที่เล่น
- UINT GetDuration(void) { return (m_nDuration); } //หาช่วงเวลา
- BYTE GetSilenceData(void);

Attribute

- WAVEFORMATEX* m_pFormat;
- HMMIO m_hmmio;
- MMRESULT m_mmr;
- MMCKINFO m_mmckiRiff;
- MMCKINFO m_mmckiFmt;

- MMCKINFO m_mmckiData;
- UINT m_nDuration;
- UINT m_nBlockAlign;
- UINT m_nAvgDataRate;
- UINT m_nDataSize;
- UINT m_nBytesPlayed;

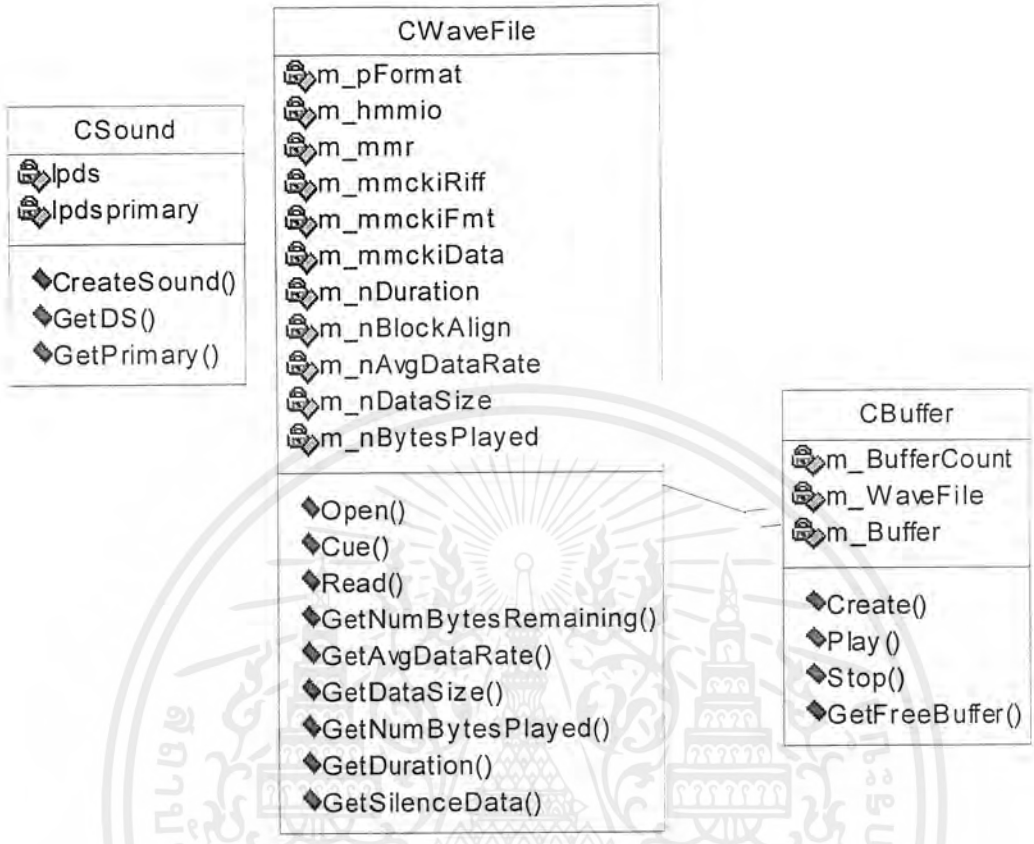
5.10.3 CBuffer

Method

- virtual HRESULT Create(CSound* sound, char* filename, int count); //สร้าง secondary buffer ขึ้นมา
- HRESULT Play(DWORD flags); //เล่นเสียง flag=0 คือเล่นรอบเดียว, flag= DSBPLAY_LOOPING คือเล่นวน
- HRESULT Stop(void); //หยุดเล่น
- LPDIRECTSOUNDBUFFER GetFreeBuffer(void);

Attribute

- int m_BufferCount; //จำนวน buffer
- CWaveFile* m_WaveFile; //pointer ไปยัง CWaveFile
- LPDIRECTSOUNDBUFFER *m_Buffer; //pointer ไปยัง DirectSound buffer



รูปที่ 5-5 class diagram ของ DirectSound

5.11 ตัวอย่าง วิธีการเรียกใช้ Object ที่สร้างสำหรับ DirectSound

Declare pointer to Class

```
CSound* sound_t;
CWaveFile* wavefile_t;
CBuffer* buffer_t;
CBuffer* buffer_t2;
```

Pointer ไปยัง class ต่างๆ

Allocate memory for pointer and Access Method

sound_t = new CSound; //allocate pointer สำหรับ CSound

sound_t -> CreateSound(main_window_handle); // สร้าง sound Object และ Set Cooperative Level และ สร้าง PrimaryBuffer

```
//สร้างเสียง Intro ของเกม อยู่ในส่วน Game_Init(); ////////////////////////////////////////
wavefile_t = new CWaveFile; //allocate pointer สำหรับ CWaveFile
buffer_t = new CBuffer; //allocate pointer สำหรับ CBuffer

buffer_t -> Create(sound_t, "1125sc1.wav", 1); // กำหนดรูปแบบข้อมูล และ buffer, สร้าง buffer, Lock,
คัดลอกไฟล์ 1125sc1.wav มาใส่ buffer, Unlock, ส่วนค่าหลังสุดเป็นจำนวน buffer ที่ต้องการสร้าง
buffer_t -> Play(0); //เล่นเสียง ชื่อไฟล์ 1125sc1.wav รอบเดียว <flag =0>
Sleep(10000); //หน่วงเวลา
buffer_t -> Create(sound_t, "03-Track.wav", 1);

//Game_Main(); ////////////////////////////////////////
buffer_t -> Play(DSBPLAY_LOOPING); //Theme Sound, Play looping

// ตัวอย่าง event sound
keyboard_t -> GetState();// จาก DirectInput

if (keyboard_t -> keyboard_state[DIK_1]) // ถ้ากด 1 จะเล่นเสียง slowdown.wav
{
        buffer_t2 = new CBuffer; //allocate Cbuffer สำหรับ เสียง event ต่างๆ
        buffer_t2 -> Create(sound_t, "slowdown.wav", 1);
        buffer_t2 -> Play(0);
        delete buffer_t2;
}
// ถ้ากด s Theme sound จะหยุดเล่น กด s อีกครั้ง Theme sound ก็จะเล่นต่อ
else
if (keyboard_t -> keyboard_state[DIK_S])
{
        if (Check_theme_sound)
        {
                buffer_t -> Stop();//stop sound
                Check_theme_sound = FALSE;
        }
```

```
MessageBox(main_window_handle, "Continue", "Stop Sound", MB_OK);
if (!Check_theme_sound)
{
        buffer_t -> Play(DSBPLAY_LOOPING);
        Check_theme_sound = TRUE;
}
```

Deallocate pointer

```
//release buffer
delete buffer_t;

//release sound
delete sound_t;
```

บทที่ 6
Direct3D

6.1 Direct3D Immediate Mode และ Direct3D Retained-Mode

Direct3D ถูกออกแบบมาให้สนับสนุนทั้งการแสดงผลโดยใช้การคำนวณทั้งจากซอฟต์แวร์และฮาร์ดแวร์ Direct3D Retained-Mode ซึ่งเป็น API ในระดับสูง (high-level) นั้นถูกพัฒนาขึ้นมาเพื่อควบคุมวัตถุ 3 มิติ (3D objects) และจัดการกับฉาก 3 มิติ (3D scenes) ส่วน Direct3D Immediate Mode ซึ่งเป็น API ในระดับต่ำ (low-level) นั้น ถูกออกแบบมาเพื่อผู้พัฒนาซอฟต์แวร์ที่ต้องการย้ายซอฟต์แวร์ที่ได้พัฒนาไว้แล้วบน MS-DOS ให้มาทำงานบน Windows โดยที่ไม่ต้องละทิ้ง 3D engines ที่ได้พัฒนาไว้แล้วบน MS-DOS

6.1.1 Direct3D Immediate Mode

Diret3D Immediate Mode เป็น API ในระดับต่ำ ผู้พัฒนาแอปพลิเคชั่นสามารถจะควบคุมได้ในระดับ polygons และ vertices โดยสนับสนุนการเข้าถึงความสามารถของฮาร์ดแวร์ในแบบ device independent ไม่เหมือนกับ Retained-Mode API ใน Immediate Mode API นั้นไม่มี geometry engine ให้ดังนั้นแอปพลิเคชั่นของเราจึงต้องจัดการในส่วนนี้เอง ผู้พัฒนาแอปพลิเคชั่นสามารถย้ายแอปพลิเคชั่นที่ได้พัฒนาไว้แล้วบน MS-DOS มาทำงานบน Windows ได้ทำให้สามารถใช้ระบบการแสดงผล ระบบการจัดการ วัตถุ 3 มิติ และ ระบบการจัดการฉาก 3 มิติ ที่ได้พัฒนาไว้แล้วมาปรับใช้ได้โดยไม่ต้องพัฒนาขึ้นมาใหม่ และสามารถที่จะใช้ความสามารถของฮาร์ดแวร์ที่ช่วยการประมวลผลกราฟฟิก 3 มิติได้ในแบบ device-independent

6.1.2 Direct3D Retained-Mode

Direct3D Retained-Mode อนุญาตให้เราใช้งานได้โดยที่เราไม่ต้องพัฒนา 3D geometry engines, 3D databases, object format ขึ้นมาเอง โดยที่ผู้ใช้สามารถ load วัตถุ 3 มิติเข้ามาใช้ในแอปพลิเคชั่นได้ด้วยคำสั่งเพียงคำสั่งเดียว ผู้ใช้สามารถที่จะใช้คำสั่งเพื่อจะควบคุมให้วัตถุ เลื่อนตำแหน่ง (translate), หมุน (rotate), เปลี่ยนขนาด (scale) ได้โดยง่าย โดยไม่ต้องอาศัยความรู้เกี่ยวกับโครงสร้างภายในของวัตถุ 3 มิติ

ดังนั้น Direct3D Immediate Mode นั้นจึงเหมาะกับกลุ่มผู้พัฒนาแอปพลิเคชั่นทางด้าน 3 มิติ ที่ได้เคยพัฒนาแอปพลิเคชั่นไว้แล้วบน MS-DOS ซึ่งผู้พัฒนาเหล่านี้ได้พัฒนาระบบ 3D engines ไว้ และมีโปรแกรมช่วย (utilities) ต่างๆที่ทำงานร่วมกันได้อย่างมีประสิทธิภาพ จึงไม่อยากที่จะละทิ้งสิ่งเหล่านี้ให้สูญเปล่า ส่วน Dirct3D Retained-Mode เหมาะกับผู้พัฒนาที่ต้องการความรวดเร็วในการพัฒนาแอปพลิเคชั่น โดยไม่ต้องการประสิทธิภาพการทำงานที่สูงมากนัก หรือผู้ที่ไม่มีความคุ้นเคยกับการพัฒนาแอปพลิเคชั่นแบบในรูปแบบนี้ ดังนั้น ในโครงงานนี้จึงศึกษาเกี่ยวกับ Direct3D Retained-Mode

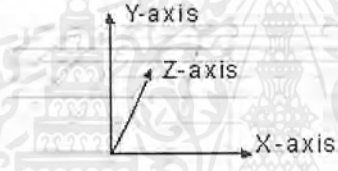
6.2 พื้นฐานคอมพิวเตอร์กราฟฟิกแบบ 3 มิติ

6.2.1 3D Coordinate Systems

ในระบบคอมพิวเตอร์กราฟฟิกแบบ 3 มิติ นั้น มีระบบ coordinate อยู่ 2 แบบ คือ ตามกฎมือขวาและตามกฎมือซ้าย ในทั้งทั้ง 2 แบบ ด้านบวกของแกน x จะชี้ไปทางขวา ส่วนด้านบวกของแกน y จะชี้ขึ้นข้างบน ตามกฎมือขวาด้านบวกของแกน z จะชี้เข้าหาตัวเรา ตามกฎมือซ้ายด้านบวกของแกน z จะชี้ออกจากตัวเรา

6.2.1.1 Direct3D Coordinate System

ระบบ coordinate ใน Direct3D จะเป็นไปตามกฎมือซ้าย คือ ด้านบวกของแกน z จะชี้ออกจากตัวเรา ในระบบ coordinate ตามกฎมือซ้ายนี้ การหมุนตามเข็มนาฬิกาถือว่าเป็นการหมุนในทิศทางบวก ดังรูปที่ 6-1



รูปที่ 6-1 ระบบ coordinate ใน Direct3D

6.2.1.2 U- และ V- Coordinate

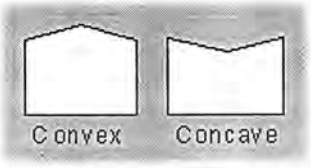
เป็น texture coordinates ใน Direct3D ถูกใช้เมื่อมีการ map textures ลงบนวัตถุ 3 มิติ เวกเตอร์ v แสดงถึงทิศทางหรือการวางตัวของ texture ในแนวแกน z ส่วน เวกเตอร์ u (เป็นเวกเตอร์ที่ชี้ในแนวตั้ง) แสดงถึงทิศทางหรือการวางตัวของแกน y

6.2.2 Polygons

วัตถุ 3 มิติใน Direct3D นั้นอยู่ในรูปแบบของ meshes ซึ่ง mesh นั้นประกอบด้วย faces ซึ่งถูกอธิบายโดย polygon ในรูปแบบง่ายๆ polygon ที่พื้นฐานที่สุดคือรูปสามเหลี่ยม ใน Direct3D Retained-Mode สามารถกำหนดให้ polygons มี ได้มากกว่า 3 vertices โดยที่ Direct3D จะทำการแปลงในอยู่ในรูปแบบที่เป็นรูปสามเหลี่ยมก่อนวัตถุนั้นจะเข้าสู่ขั้นตอนการแสดงผล ส่วนใน Direct3D Immediate Mode นั้น polygons ต้องอยู่ในรูปแบบของรูปสามเหลี่ยมเท่านั้น

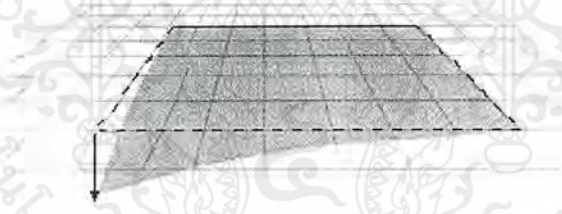
6.2.2.1 Geometry Requirements

รูปสามเหลี่ยมเป็น polygon ในรูปแบบที่ดีที่สุด เพราะมีคุณสมบัติทั้ง convex และ planar การที่ polygon มีคุณสมบัติ convex นั้นหมายความ เส้นระหว่าง 2 จุดใดๆบนขอบของ polygon จะอยู่ภายใน polygon เสมอดังรูปที่ 6-2



รูปที่ 6-2 polygon ที่มีคุณสมบัติ Convex และ Concave

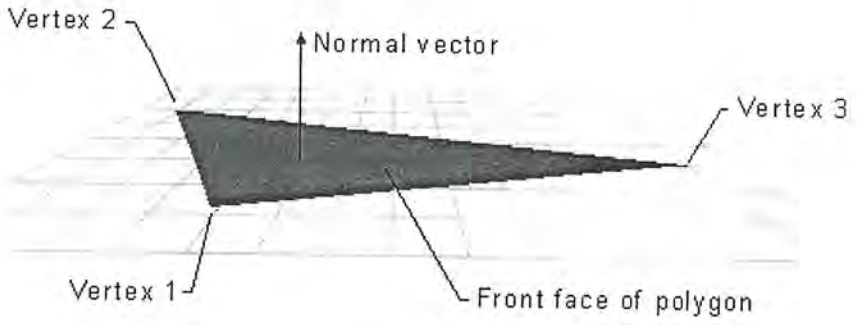
polygons ที่เป็นรูปสามเหลี่ยมนั้นจะมีคุณสมบัติ planar เสมอ ถ้ารูป polygons มี vertices มากกว่า 3 จะมีโอกาสเกิด nonplanar polygon ได้ง่าย ดังรูปที่ 6-3



รูปที่ 6-3 nonplanar polygon

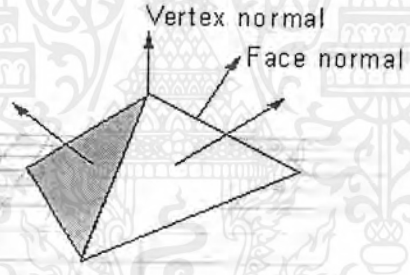
6.2.2.2 Face และ Vertex Normals

แต่ละ face ใน mesh จะมีเวกเตอร์ตั้งฉาก (normal vector) ทิศทางของเวกเตอร์ที่ตั้งฉากกับ face (face normal) นี้จะพิจารณาได้จากลำดับของ vertices ที่ถูกกำหนดใน polygon และขึ้นอยู่กับว่าเรายึดตามกฎมือขวา หรือว่ากฎมือซ้าย เราสามารถพิจารณาด้านหน้าของ face ได้โดยดูว่าเวกเตอร์ที่ตั้งฉากกับ face นั้นชี้ออกไปทางใด ทางด้านนั้นถือว่าเป็นด้านหน้าของ face ใน Direct3D ทางด้านหน้าของ face เท่านั้นที่เราสามารถมองเห็นได้ ดังรูปที่ 6-4



รูปที่ 6-4 face และ normal vector

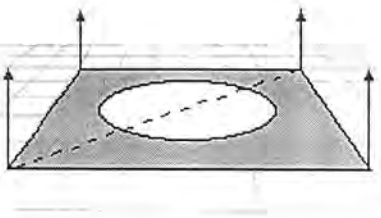
การทำงานกับ Direct3D นั้นเราไม่จำเป็นต้องกำหนดเวกเตอร์ที่ตั้งฉากกับ face ให้แต่ละ face ด้วยตนเองเพราะ Direct3D จะทำการคำนวณให้เราโดยอัตโนมัติ Direct3D จะใช้เวกเตอร์ที่ตั้งฉากกับ face ใน flat shade mode ส่วนใน Phong และ Gouraud shade modes นั้น Direct3D จะใช้ เวกเตอร์ที่ตั้งฉากกับ vertex ในการคำนวณเกี่ยวกับแสง และ textures ดังรูปที่ 6-5



รูปที่ 6-5 face normal และ vertex normal

6.2.2.3 Shade Modes

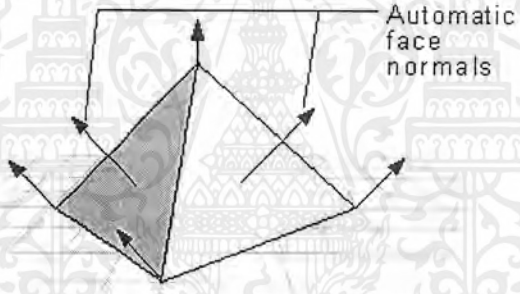
ใน flat shade mode นั้น ระบบจะ ใช้สีที่คำนวณได้จาก vertex แรกของ face กำหนดให้พื้นผิวของทั้ง face นั้น ใน Gouraud และ Phong shade modes เวกเตอร์ตั้งฉากกับ vertex จะถูกใช้คำนวณเพื่อให้เกิดพื้นผิวที่มีความนุ่มนวล ใน Gouraud shading จะคำนวณสีที่แต่ละ vertex และทำการ interpolate ค่าสีในพื้นระหว่าง vertices เหล่านั้น ส่วนใน Phong shading นั้นจะคำนวณ ค่า shade ที่เหมาะสมให้กับแต่ละ pixel บน face แอปพลิเคชั่นโดยทั่วไปจะใช้ Gouraud shading เพราะพื้นผิวของวัตถุ 3 มิติ ที่ได้ออกมามีความนุ่มนวล และ ใช้เวลาในการประมวลผลที่น้อยกว่า Phong shading แต่ Phong shading มีข้อดีกว่า Gouraud shading เพราะมีบางกรณีที่ Gouraud shading จะให้ผลลัพธ์ออกมาผิดจากความเป็นจริง ยกตัวอย่างดังรูปที่ 6-6



รูปที่ 6-6 แสงที่ส่องอยู่ภายใน face

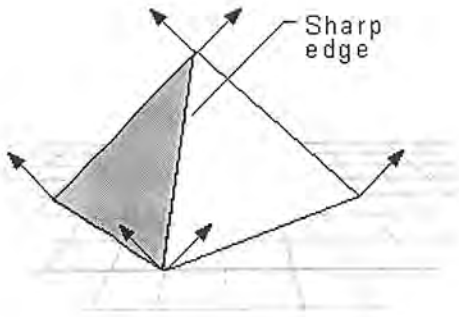
ในกรณีนี้ Phog shade mode จะทำการคำนวณคำนวณค่าที่แต่ละ pixel และสามารถจะแสดงแสงไฟที่ส่องอยู่ตรงกลาง face ได้ ส่วนใน Gouraud shade mode นั้นจะคำนวณค่าเฉพาะที่แต่ละ vertex และทำการ interpolate พื้นที่ที่อยู่ระหว่าง vertices ดังนั้นเมื่อแสดงผลออกมาเราจะไม่เห็นแสงไฟที่ส่องอยู่ตรงกลาง face

ดังรูปที่ 6-7 รูปปีรามิดจะถูกแสดงผลโดยมีขอบที่คมใน flat shade mode ส่วนใน Gouraud และ Phog shade modes เนื่องจากการ interpolate ระหว่างขอบของ faces ขอบจึงดูนุ่มนวล



รูปที่ 6-7 face normal ของรูปปีรามิด

ถ้าเราต้องการใช้ Gouraud หรือ Phong shade mode เพื่อแสดงผลพื้นผิวของวัตถุ 3 มิติ ให้มีความนุ่มนวล แต่ต้องการให้ขอบของวัตถุ 3 มิติ เป็นขอบที่คม เราต้องเพิ่ม เวกเตอร์ตั้งฉากที่แต่ละ vertex ของ face ที่ตัดกัน ที่เราต้องการให้ขอบแลดูคม ดังรูปที่ 6-8



รูปที่ 6-8 วัตถุ 3 มิติ ใน Gouraud Shade ที่มีขอบคม

ดังนั้นในวัตถุ 3 มิติ หนึ่งชิ้นสามารถที่จะมีได้ทั้งขอบที่ดูนุ่มนวลและขอบที่แลดูคม Gouraud shading เหมาะที่จะใช้กับ Direct3D แอปพลิเคชั่นมากที่สุด เพราะทำให้วัตถุแลดูเสมือนจริงและใช้เวลาในการประมวลผลที่ไม่มากนัก ส่วน Phong shading นั้นยังไม่ได้รับการสนับสนุนใน DirectX เวอร์ชันปัจจุบัน (เวอร์ชัน 6.0)

6.3 สถาปัตยกรรมของ Direct3D Retained-Mode

6.3.1 DIRECT3DRM

Direct3D ประกอบด้วยหลายๆ COM interfaces และ COM interfaces จะขึ้นอยู่กับ Object หลัก คือ Direct3DRM (RM หมายถึง Retained-Mode) เราสามารถสร้าง interface นี้ได้โดย

```
LPDIRECT3DRM d3drm;                  // Pointer ที่ชี้ไปที่ Direct3DRM interface
Direct3DRMCreate (&d3drm);           // สร้าง Object และทำการ initialize Pointer ที่ชี้ไปที่ interface
```

6.3.1.1 การสร้าง Direct3DRM Objects

จุดมุ่งหมายหลักของการสร้าง Direct3DRM interface นั้นก็เพื่อสร้าง Direct3D objects อื่นๆ เช่น

- CreateAnimation ( )
- CreateAnimationSet ( )
- CreateDeviceFromClipper ( )
- CreateDeviceFromDirect3D ( )
- CreateDeviceFromSurface ( )
- CreateFace ( )
- CreateFrame ( )
- CreateLight ( )

- CreateLightRGB ( )
- CreateMaterial ( )
- CreateMesh ( )
- CreateMeshBuilder ( )
- CreateObject ( )
- CreateUserVisual ( )
- CreateShadow ( )
- CreateTexture ( )
- CreateTextureFromSurface ( )
- LoadTexture ( )
- CreateViewport ( )
- CreateWrap ( )

เราไม่จำเป็นต้องใช้ฟังก์ชันเหล่านี้ทั้งหมดเพราะบางฟังก์ชันจะทำหน้าที่เดียวกัน เช่น CreateLight ( ) กับ CreateLightRGB ( ) นั้นเหมือนกันเพียงแต่จะต่างกันในรูปของการกำหนดค่าสี

6.3.1.2 การเปลี่ยนแปลง Search Path

Direct3DRM interfaces นั้นสามารถให้เราปรับ Search Path เพื่อให้ Direct3D ค้นหาไฟล์ต่างๆที่แอปพลิเคชั่นต้องการใช้ เราสามารถปรับแต่ง Search Path ได้โดยใช้ฟังก์ชันเหล่านี้

- AddSearchPath ( )
- GetSearchPath ( )
- SetSearchPath ( )

6.3.1.3 การควบคุมการ Update การแสดงผล

Direct3DRM Object มีฟังก์ชัน **Tick ( )** ซึ่งควบคุมความเร็วในการแสดงผลของโปรแกรม แต่ละครั้งที่ Tick ( ) ฟังก์ชันนั้นถูกเรียก Direct3D จะทำการ update ส่วนของแอนิเมชันใน scene ถ้าเราเรียกใช้ Tick ( ) บ่อยนั้นจะทำให้แอนิเมชันใน scene ของเรามีการเปลี่ยนแปลงที่รวดเร็วมากขึ้น และ Tick ( ) นั้นรับพารามิเตอร์ หนึ่งตัว ซึ่งปกติจะใส่เป็น 1.0 ถ้าพารามิเตอร์นี้น้อยกว่า 1.0 นั้นแอนิเมชันภายใน scene ของเราจะมีการเปลี่ยนแปลงที่ช้ากว่าปกติ แต่ถ้ามากกว่า 1.0 นั้นแอนิชันภายใน scene ของเราจะมีการเปลี่ยนแปลงที่เร็วมากกว่าปกติ

6.3.2 DIRECT3DRMDEVICE

Direct3D devices นั้นเป็น object ที่สร้าง output ซึ่งเป็นภาพกราฟฟิกแบบ 3 มิติ Direct3D นั้นสนับสนุน Devices หลายรูปแบบ แอปพลิเคชั่นของเราสามารถที่จะเลือก devices ได้อย่างอิสระ หรือจะให้ Direct3D เลือก device ให้เราโดยอัตโนมัติ โดยที่ devices นั้นมีอยู่ 2 ประเภทหลักๆคือ Software Devices และ Hardware Devices ซึ่ง Software Devices นั้นใช้สำหรับเครื่องคอมพิวเตอร์ที่ไม่ได้ติดตั้งการ์ดเร่งความเร็วแบบ 3 มิติ ส่วน Hardware Devices นั้นจะสามารถใช้ได้บนเครื่องคอมพิวเตอร์ที่ติดตั้งการ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ ดังนั้นแอปพลิเคชั่นของเราควรจะสามารถตรวจสอบเครื่องคอมพิวเตอร์ของผู้ใช้ได้ว่าติดตั้งการ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ หรือไม่ ถ้าติดตั้งอยู่แอปพลิเคชั่นของเราควรเลือกใช้ Hardware Devices เพื่อให้แอปพลิเคชั่นของเราใช้ความสามารถจากการ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ ที่ติดตั้งบนเครื่องคอมพิวเตอร์ของผู้ใช้ได้ แต่วิธีที่ดีที่สุดนั้นคือ แอปพลิเคชั่นของเราควรจะตรวจสอบว่าบนเครื่องคอมพิวเตอร์ที่แอปพลิเคชั่นของเราที่ทำงานอยู่นั้นสามารถจะสร้าง Devices ประเภทใดได้บ้างและให้ผู้ใช้นั้นสามารถเลือก Device ที่ผู้ใช้ต้องการได้ โดยแอปพลิเคชั่นของเรานั้นจะเลือก Device ที่มีประสิทธิภาพมากที่สุดเป็น default

Devices นั้นถูกแสดงโดย Direct3DRMDevice interface และสามารถสร้างได้โดยฟังก์ชันของ Direct3DRM object เราสามารถสร้าง Device ได้ 3 วิธี

1. สร้าง DirectDraw "clipper" object และสร้าง Device จาก **CreateDeviceFromClipper ( )** ซึ่งเป็นวิธีการที่ง่ายที่สุด
2. สร้าง DirectDraw primary surface และ Back-Buffer (เพื่อใช้ความสามารถของ Page Flipping) และสร้าง device จาก **CreateDeviceFromSurface ( )**
3. Init ค่าของ Direct3D Immediate Mode และสร้าง device จาก **CreateDeviceFromD3D ( )**

6.3.2.1 Color Models

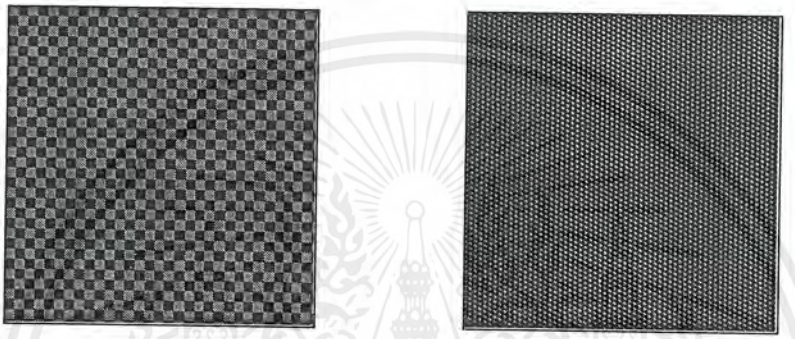
ทั้ง Software และ Hardware Devices มีอยู่ 2 รูปแบบด้วยกัน คือ "RBG" และ "Ramp" RGB color model นั้นสนับสนุนแสงที่มีสี ซึ่ง Ramp color model นั้นไม่สนับสนุนซึ่งจะประมวลผลได้รวดเร็วกว่า RGB color model อย่างมาก Ramp devices มักจะถูกเรียกว่า "mono" devices เพราะ เป็นแสงแบบ "monochromatic" ซึ่งเป็นแสงในช่วงสีดำถึงขาว

6.3.2.2 Rendering Options

Devices นั้นสามารถที่จะใช้ปรับ rendering options ในรูปแบบต่างๆได้โดยใช้ฟังก์ชัน **GetQuality ( )** และ **SetQuality ( )** เช่นเราสามารถกำหนด rendering mode ได้เช่น Gouraud และ Flat ซึ่ง device ใช้สำหรับแสดงผล scene

ฟังก์ชัน **GetShades ( )** และ **SetShades ( )** ทำให้เราสามารถกำหนดจำนวน shades ของสี ที่ device ใช้แสดงผล scene การเซตค่านี้ขึ้นอยู่กับ ค่า bit depth ของ โหมดการแสดงผลปัจจุบัน

ฟังก์ชัน **GetDither ( )** และ **SetDither ( )** ทำให้เราสามารถกำหนดได้ค่าจะใช้เทคนิค Dithering ในการแสดงผลหรือไม่ โดยที่ Dithering เป็นเทคนิคการจำลองสีให้มีจำนวนสีมากกว่าจำนวนสีที่มีอยู่โดยจะมีความจำเป็นกับโหมดการแสดงผลที่มีจำนวนสีน้อยๆ โดยมีวิธีการเช่นเรามีสีอยู่แค่ 8 สี คือ ขาว, ดำ, แดง, น้ำเงิน, เขียว, เหลือง, Cyan (เขียวผสมกับฟ้า), Magenta (ม่วงแดง) แต่ถ้าเราต้องการสี Purple (ม่วงน้ำเงิน) เราสามารถจำลองสีม่วงน้ำเงินได้โดยเราจะใส่จุดสีแดงและน้ำเงินสลับกันในรูปแบบที่ถูกกำหนดไว้ เมื่อเรามองจะกลายเป็นสีม่วงน้ำเงิน ดังรูปที่ 6-9 แสดง Dithering pattern



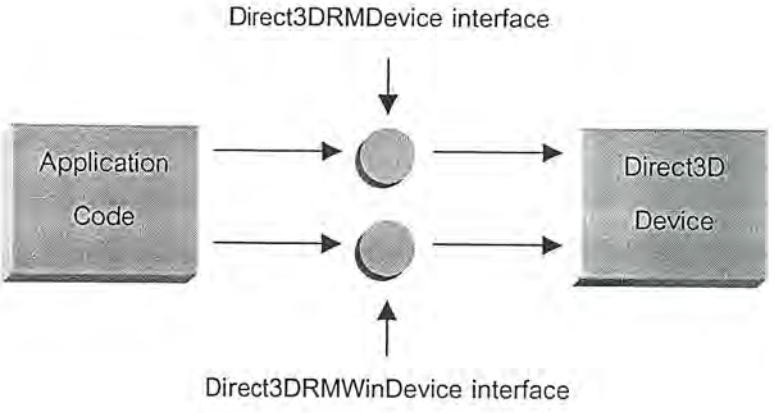
(a) (b)

รูปที่ 6-9 Dithering pattern

รูปที่ 6-9 (a) นั้นเป็นการนำสีแดงกับน้ำเงินมาเรียงกันตาม Dithering pattern รูปที่ 6-9 (b) เมื่อมองไกลๆแล้วจะกลายเป็นสีม่วงน้ำเงิน เราสามารถทำให้สีนี้ดูแดงขึ้นได้โดยการเพิ่มจุดสีแดงเข้าไป หรือต้องการให้เป็นสีน้ำเงินเข้มมากขึ้นก็เพิ่มจุดสีน้ำเงินเข้าไป

6.3.3 DIRECT3DWINDEVICE

Objects ของ Direct3D device นั้นสนับสนุนหลาย interfaces ส่วน Direct3DwinDevice interface นั้นแสดงถึง devices ที่ถูกสนับสนุนโดย Windows ดังรูปที่ 6-10 ซึ่งจำเป็นสำหรับแอปพลิเคชั่นแบบ Windowed-Mode



รูปที่ 6-10 Device object และ interfaces

เราสามารถเรียกใช้ Direct3DRMWinDevice interface โดยการใช้ **QueryInterface ( )** จาก Direct3DRMDevice interface ดังตัวอย่าง

```
LPDIRECT3DRMWINDEVICE windev;     // Pointer ที่ชี้ไปที่ Direct3DRMWinDevice interface
device->QueryInterface (IID_IDirect3DRMWinDevice, (void**)&windev);
```

device เป็น pointer ที่ถูก Initialize แล้วซึ่งจะชี้ไปที่ Direct3DRMDevice interface ส่วน **QueryInterface ( )** ฟังก์ชันนั้นใช้สำหรับรับค่า pointer ที่ชี้ไปที่ Direct3DRMWinDevice interface และ IID_IDirect3DRMWinDevice คือ GUID (Globally Unique Identifier) สำหรับ interface ที่ถูกร้องขอ

WinDevice นั้นสนับสนุนสองฟังก์ชัน คือ HandleActivate ( ) และ HandlePaint ( ) นั้นถูกใช้เพื่อตอบสนอง WM_ACTIVATE และ WM_PANIT ที่ถูกส่งเข้ามาที่ Message Handler ตามลำดับ สำหรับแอปพลิเคชั่นที่ทำงานใน Windowed-Mode

6.3.4 DIRECT3DRMVIEWPORT

Viewport ใน Direct3D นั้นก็คือกล้อง (Camera) นั่นเอง ซึ่ง viewport จะเป็นตัวที่อธิบายตำแหน่ง (location) และ การวางตัว (orientation) ของตำภาพที่จะถูกแสดงออกมา viewport นั้นสามารถที่จะปรับ "field-of-view", "front and back clipping", "perspective transformations"

Viewport ถูกแสดงโดย Direct3DRMViewport interface และสามารถสร้างได้โดย Direct3DRM **CreateViewport ( )** ฟังก์ชัน ดังนี้

```
d3drm->CreateViewport (device, camera,
                       0, 0, device->GetWidth ( ), device->GetHeight ( ),
                       &viewport);
```

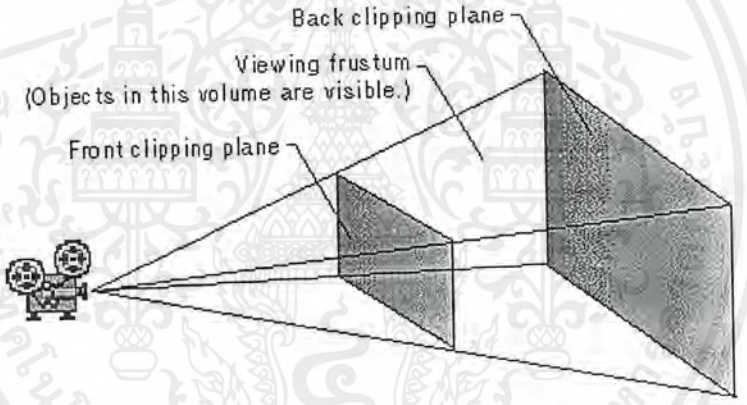
d3drm คือ pointer ที่ชี้ไปที่ Direct3DRM interface ส่วน device เป็น pointer ที่ชี้ไปที่ Direct3DRMDevice interface และ camera จะเป็น frame ที่จะอธิบายถึงตำแหน่งและการวางตัวของ viewport

6.3.4.1 Field of View

Field of View (FOV) สามารถปรับได้โดยใช้ฟังก์ชัน **SetField ( )** ค่าของ FOV ปกติคือ 0.5 ถ้าเราเซตค่าให้น้อยกว่านี้จะได้ภาพขยายเหมือนมองจากกล้องส่องทางไกล แต่ถ้าใช้ค่ามากกว่านี้จะทำให้ได้ภาพที่มีมุมมองของภาพกว้าง

6.3.4.2 Clipping

Viewport สามารถจะใช้ควบคุม Front และ Back 3D clipping โดยใช้ฟังก์ชัน SetFront ( ) และ SetBack ( ) กำหนดพื้นด้านหน้าของกล้องที่วัตถุต่างๆจะถูกแสดง โดยที่วัตถุที่อยู่นอกบริเวณนี้จะไม่ถูกแสดงออกมาตามรูปที่ 6-11



รูปที่ 6-11 Clipping plane

6.3.5 DIRECT3DRMFRAME

Direct3D Retained-Mode นั้นใช้ frame อย่างมาก ซึ่ง frame ในที่นี้หมายถึง "frame of reference" ซึ่งเป็นตำแหน่งอ้างอิงของวัตถุต่างๆ เช่น meshes, faces, cameras, light sources ซึ่งวัตถุเหล่านี้ตัวของมันเองจะไม่อธิบายถึงตำแหน่งและการวางตัว โดยวัตถุเหล่านี้จะถูกเชื่อมเข้ากับ frame และจะใช้ตำแหน่งและการวางตัวของ frame ที่วัตถุเหล่านี้เชื่อมต่ออยู่

6.3.5.1 Frame Hierarchy

Scene ของ Direct3D นั้นถูกกำหนดเป็นแบบ hierarchy ที่มี root frame และ child frame จำนวนเท่าใดก็ได้ที่เชื่อมต่ออยู่กับ root frame แต่ละ child frame สามารถจะมี frame ลูกของตัวเองได้ โดยที่ frame นั้นถูกแสดงโดย Direct3DRMFrame interface และถูกสร้างด้วยฟังก์ชัน **CreateFrame ( )** ดังนี้

```
LPDIRECT3DRMFRAME newframe;
d3drm->CreateFrame (parentframe, &newframe);
```

newframe เป็น pointer ที่ชี้ไปที่ frame ที่เราต้องการสร้างขึ้นมาใหม่ และทำการ initialize ด้วยฟังก์ชัน **CreateFrame ( )** โดยที่ parentframe เป็น pointer ที่ชี้ไปที่ parent frame ของ newframe ถ้ากำหนดให้เป็น NULL จะเป็นการสร้าง root fame ของ scene

Child frame นั้นจะรับคุณสมบัติต่างๆมาจาก parent frame คุณสมบัติอย่างหนึ่งที่ได้รับมาคือ "frame of reference" คือเมื่อ child frame นั้นถูกเชื่อมต่อเข้ากับ parent frameแล้ว เมื่อ parent frame เคลื่อนที่ child frame นั้นจะเคลื่อนที่ตาม parent frame ไปด้วย

frame นั้นไม่ได้ถูกจำกัดอยู่แค่ว่าเราจะต้องใช้ parent frame เป็น frame of reference เสมอ หลายๆ ฟังก์ชันที่เกี่ยวกับ frame สามารถให้เรากำหนดว่าเราจะใช้ frame ใดใน scene เป็น frame of reference ได้อย่างอิสระ ด้วยความสามารถของ frame hierarchy นี้เองทำให้ Direct3D Retained-Mode นี้มีประสิทธิภาพในการจัดการเรื่องโครงสร้างของ scene เป็นอย่างมาก

6.3.5.2 Frame Positioning

Direct3DRMFrame interface นั้นมีหลายๆฟังก์ชันที่ใช้จัดการเกี่ยวกับการวางตำแหน่งของ frame ดังนี้

- AddRotation ( )
- AddScale ( )
- AddTranslation ( )
- GetOrientation ( )
- GetPosition ( )
- LookAt ( )
- SetOrientation ( )
- SetPosition ( )

ฟังก์ชัน **GetPosition ( )** และ **SetPosition ( )** อนุญาตให้เราตรวจสอบและกำหนดตำแหน่งของ frame ได้ **GetOrientation ( )** และ **SetOrientation ( )** อนุญาตให้เราตรวจสอบและกำหนดการวางตัวของ frame ได้ ส่วนฟังก์ชัน **LookAt ( )** เหมาะสำหรับกรณีที่เราต้องการให้ camera และ light source ตามวัตถุใดๆไป

SetPosition ( ), SetOrientation ( ), และ **LookAt ( )** นั้นเป็นฟังก์ชันที่ทำการกำหนดค่าให้กับ frame โดยที่ไม่สนใจค่าเก่าที่ถูกกำหนดไว้ ส่วนฟังก์ชัน AddTranslation ( ), AddRotation ( ), และ AddScale ( ) นั้นจะทำการปรับค่าของ frame โดยพิจารณาจากค่าเก่าที่ได้ถูกกำหนดไว้ก่อนหน้าด้วย

6.3.5.3 Frame Movement

Frame interface นั้นสนับสนุนฟังก์ชันที่จะเปลี่ยนแปลงค่าต่างๆที่เกี่ยวกับการเคลื่อนไหวของ frame ในทุกๆรอบที่ scene ถูก update และแสดงผล

GetRotation ( ) และ **SetRotation ( )** เป็นฟังก์ชันที่จัดการเกี่ยวกับค่าการหมุนของ frame โดยที่ **SetRotation ( )** จะมีผลต่อทุกๆครั้งที่ scene ถูก update

6.3.5.4 Move Callbacks

ด้วยความสามารถของ frame ที่เราสามารถกำหนดค่าเกี่ยวกับการเคลื่อนไหวให้มันได้นั้นเป็นสิ่งที่ดี แต่ ในแอปพลิเคชั่นแบบ interactive นั้น วัตถุนั้นต้องสามารถตอบสนองต่อวัตถุอื่นๆ หรือ ตอบสนองต่ออินพุตที่ผู้ใช้ป้อนเข้ามาได้ ดังนั้น frame interface จึงสนับสนุนการปรับเปลี่ยนค่าต่างที่เกี่ยวกับการเคลื่อนไหวในขณะ run-time โดยการใช้ "callbacks" ซึ่ง callbacks นี้คือฟังก์ชันที่เราเขียนขึ้นมาและกำหนดให้กับ Direct3D หลังจากนั้น เมื่อ Direct3D จะเรียกการทำงานที่อยู่ภายในฟังก์ชันนี้ทุกๆรอบก่อนที่ scene จะถูกแสดง ทำให้เราสามารถปรับค่าต่างๆที่เกี่ยวกับการเคลื่อนไหวของ frame ภายใน scene ได้

Callbacks นั้นสามรถติดตั้งได้โดยใช้ฟังก์ชัน **AddMoveCallback ( )** โดยที่ frame หนึ่งๆนั้นสามารถจะมีได้หลายๆ Callbacks โดยการทำงานจะเป็นไปตามลำดับที่มันถูกติดตั้ง และเราสามารถยกเลิกการใช้ callbacks ได้โดยการเรียกใช้ฟังก์ชัน **DeleteMoveCallback ( )**

6.3.6 DIRECT3DRMMESHBUILDER

Direct3DRMMeshBuilder interface นั้นเป็น interface ระดับสูง (high-level) ที่มีฟังก์ชันต่างๆ ช่วยอำนวยความสะดวกในการจัดการเกี่ยวกับวัตถุ 3 มิติ ในรูปแบบของ mesh ซึ่งรวมถึงการสร้าง mesh และการปรับปรุง mesh โดยที่ MeshBuilder นั้นตัวมันเองไม่ใช่ mesh แต่สามารถจะใช้แทน mesh ได้ MeshBuilder นั้นสามารถที่จะถูกเพิ่มเข้าไปใน scene ได้ในรูปแบบที่เป็นส่วนประกอบที่สามารถมองเห็นได้ (visual elements) เมื่อ MeshBuilder นั้นถูกเพิ่มเข้าไปใน scene มันจะใช้ mesh ที่อยู่ภายใน MeshBuilder นั้นแสดงใน scene เราสามารถสร้าง MeshBuilder ได้โดยใช้ฟังก์ชัน **CreateMeshBuilder ( )** ของ Direct3DRM ดังนี้

```
LPDIRECT3DRMMESHBUILDER meshbuilder;
d3drm->CreateMeshBuilder (&meshbuilder);
```

6.3.6.1 Load and Saving

ฟังก์ชัน **Load ( )** ทำให้ MeshBuilder นั้นสามารถ load mesh จากดิสก์, program resources, หรือ memory โดยที่ฟังก์ชัน Load ( ) จะ load textures ที่มากับ mesh เมื่อ textures นั้นอยู่ใน directory ที่มีอยู่ใน search path ของ Direct3D

Load ( ) นั้นสามารถจะใช้ callback function ได้เมื่อทำการ load texture เป็นการ override texture ปกติที่มากับ mesh ทำให้เราสามารถ load texture ได้จาก directory ที่กำหนดและ resource ของโปรแกรม

ฟังก์ชัน Save ( ) นั้นทำให้เราสามารถบันทึก mesh ได้ทั้งในรูปแบบ “Text file” , “Binary file”, หรือ “Compressed file”

6.3.6.2 Rendering Options

MeshBuilder นั้นอนุญาตให้เราจัดการกับคุณภาพของ mesh เช่น color, texture, และ rendering options ฟังก์ชันของ MeshBuilder ที่ใช้สำหรับจัดการ rendering options ดังนี้

- GetPerspective ( )
- GetQuality ( )
- GetTextureCoordinate ( )
- SetColor ( )
- SetColorRBG ( )
- SetPerspective ( )
- SetQuality ( )
- SetTexture ( )
- SetTextureCoordinate ( )
- SetTextureTopology ( )

ฟังก์ชัน **SetColor ( )** และ **SetColorRGB ( )** สามารถกำหนดสีให้กับ faces ของ mesh ซึ่งที่ไม่มี **GetColor ( )** ฟังก์ชัน นั้นไม่มีเพราะเราไม่สามารถแน่ใจได้ว่าทุกๆ faces ใน mesh นั้นจะมีสีเดียวกันหมด

ฟังก์ชัน **GetQuality ( )** และ **SetQuality ( )** นั้นสามารถใช้เพื่อกำหนด rendering mode ได้โดยที่ rendering mode นี้รวมถึง *shade mode* (flat, gouraud) *fill mode* (point, wireframe, solid) และ *dithering* ในส่วนของ rendering mode นั้นเราสามารถกำหนดได้ 2 ที่ คือ กำหนดที่ MeshBuilder เอง และกำหนดที่ device ซึ่งการกำหนดที่ MeshBuilder นั้นค่า rendering mode นั้นจะแตกต่างกันไปตามแต่ละ MeshBuilder ส่วนการกำหนดที่ device นั้น rendering mode ที่กำหนดจะถูกกำหนดให้กับทุกๆ MeshBuilders ภายใน scene แต่มีข้อจำกัดคือ rendering mode ที่ device นั้นจะเป็นขีดจำกัดของ rendering mode ใน scene ทั้งหมด เช่น เราเซตให้ MeshBuilder มี rendering mode แบบ gouraud แต่ที่ device นั้นเราเซตให้มี rendering mode แบบ flat ดังนั้น MeshBuilder ที่มี rendering mode แบบ gouraud จะถูกแสดงในรูป flat

ฟังก์ชัน **SetTexture ( )** กำหนด textures ให้กับ MeshBuilder ส่วน **GetTextureCoordinates ( ), SetTextureCoordinates ( )** และ **SetTextureTopology ( )** ทำให้เราสามารถกำหนดค่าต่างๆของ textures ได้

ฟังก์ชัน **GetPerspective ( )** และ **SetPerspective ( )** สามารถใช้เพื่อกำหนดให้เรากำหนดว่าจะใช้คุณสมบัติของ “texture perspective correction” ได้ ซึ่งคุณสมบัตินี้จะทำการจัดการกับการแสดงผล texture ให้ถูกต้องตามตำแหน่งและการวางตัวจาก viewer แต่จะใช้เวลาในการประมวลผลที่มากขึ้น ซึ่งเทคนิคนี้จะมีประโยชน์ในกรณีที่ texture นั้นถูกกำหนดให้กับ face ที่มีขนาดใหญ่และอยู่ใกล้ viewer

6.3.6.3 Face Access

MeshBuilder interface จัดการเกี่ยวกับการปรับปรุงและเพิ่ม faces ให้กับ mesh ตัวอย่างเช่น List ของ faces ที่ประกอบขึ้นเป็น mesh อยู่ภายใน MeshBuilder นั้นสามารถรับมาได้จากฟังก์ชัน **GetFaces ( )** จำนวนของ faces ใน mesh นั้นสามารถได้มาจากฟังก์ชัน **GetFaceCount ( )** โดยที่ faces นั้นสามารถจะถูกสร้างขึ้นและเพิ่มเข้าไปให้กับ MeshBuilder ได้โดยใช้ฟังก์ชัน **CreateFace ( ), AddFace ( ),** และ **AddFaces ( )**

6.3.6.4 Vertex Access

MeshBuilder นั้นอนุญาตให้เราเข้าถึง vertices ที่ถูกใช้ใน mesh ได้ จำนวนของ vertices ภายใน mesh นั้นสามารถจะได้มาจากฟังก์ชัน **GetVertexCount ( )** โดยที่ vertices ทั้งหมดนั้นสามารถจะรับมาและปรับปรุงได้จากฟังก์ชัน **GetVertices ( )** และ **SetVertices ( )** ตามลำดับ ฟังก์ชันที่จัดการเกี่ยวกับ vertices นั้นมีประสิทธิภาพมากโดยที่ ทั้ง shape และ mesh นั้นสามารถที่จะถูกปรับปรุงได้โดยใช้ฟังก์ชันที่จัดการเกี่ยวกับ vertices ได้

6.3.6.5 Translating and Scaling

ตำแหน่งของ mesh นั้นจะขึ้นอยู่กับ frame ที่มันเชื่อมต่ออยู่ mesh นั้นสามารถจะถูกกำหนดได้ว่ามันจะอยู่ตรงไหนเมื่อเปรียบเทียบกัน frame ที่มันเชื่อมต่ออยู่ได้ ฟังก์ชัน **Translate ( )** ทำให้เราสามารถกำหนดค่าของช่วงระยะที่แยกออกไป (offset) ได้ทำให้เราสามารถเชื่อมต่อ mesh หลายๆอันเข้ากับ frame อันเดียวกันได้โดยไม่ไปซ้อนอยู่ที่ตำแหน่งเดียวกันหมด

ฟังก์ชัน **Scale ( )** จะทำการย่อหรือขยายขนาดของ mesh ตามค่า scale factor ที่เรากำหนดให้ ซึ่งการย่อหรือขยายนั้นจะยึด local axis ของวัตถุนั้นๆ เป็นหลัก scale factor ที่มีค่าต่างๆกันสามารถจะใช้ได้กับ axis ต่างๆของวัตถุเดียวกันทำให้เราสามารถทำเทคนิค “stretch” หรือ “reduce” ได้

6.3.6.6 Performance

ด้วยประสิทธิภาพและความง่ายในการใช้ MeshBuilder ซึ่งมีฟังก์ชันต่างๆที่ช่วยเราจัดการเกี่ยวกับ mesh มากมาย แต่ในด้านประสิทธิภาพและความรวดเร็วในการประมวลผลนั้น MeshBuilder นั้นมีข้อด้วยอยู่ ถ้าเราใช้ MeshBuilder ทำการ load mesh เข้ามาแล้วปรับแต่งค่าต่างๆของ mesh นั้นจนเป็นที่พอใจ แล้วเรานำ MeshBuilder นั้นไปใช้ใน scene โดยไม่ทำการเปลี่ยนแปลงคุณสมบัติต่างๆของมันนั้น การใช้ MeshBuilder ก็จะไม่ทำให้เราต้องเสียเวลาในการประมวลผลที่มากขึ้น แต่ถ้าเรามีการเปลี่ยนแปลงคุณสมบัติต่างๆ (color, texture, vertex, position, อื่นๆ) ของ MeshBuilder บ่อยๆขณะที่แอปพลิเคชั่นของเรากำลังทำการประมวลผลจะทำให้ประสิทธิภาพการทำงานของแอปพลิเคชั่นเราลดลง

6.3.7 DIRECT3DRMMESH

Direct3DRMMesh interface ถูกออกแบบมาโดยเน้นด้านการประมวลผลที่รวดเร็วกว่า MeshBuilder แต่ว่า Mesh นั้น ใช้ยากกว่า MeshBuilder เพราะมีฟังก์ชันที่ช่วยในการทำงานด้านต่างๆน้อยกว่า MeshBuilder

เราสามารถสร้าง Mesh interface ได้จากฟังก์ชัน **CreateMesh ( )** ของ Direct3DRM และ ฟังก์ชัน **CreateMesh ( )** ของ Direct3DRMMeshBuilder

6.3.7.1 Mesh Groups

Mesh interface นั้นจะเน้นการทำงานส่วนใหญ่ไปที่การจัดการกับ "groups" ซึ่ง group นั้นคือกลุ่มของ faces ภายใน mesh ซึ่งถูกมองอยู่ในรูปแบบ "single entity" การจัดการเกี่ยวกับ group นั้นมีฟังก์ชันดังนี้

- AddGroup ( )
- GetGroup ( )
- GetGroupColor ( )
- GetGroupCount ( )
- GetGroupMapping ( )
- GetGroupMaterial ( )
- GetGroupQuality ( )
- GetGroupTexture ( )
- SetGroupColor ( )
- SetGroupColorRGB ( )
- SetGroupMapping ( )
- SetGroupMaterial ( )
- SetGroupQuality ( )
- SetGroupTexture ( )

เราสามารถสร้าง group ได้โดยใช้ฟังก์ชัน **AddGroup ( )** ซึ่งเราต้องให้ค่ากับฟังก์ชัน AddGroup ( ) นี้ ในระดับของ vertex นั้นหมายความว่าเราต้องให้ค่าของ ตำแหน่งของ vertex, normal vectors, และ texture coordinate เมื่อ group ถูกสร้างขึ้นมาแล้วคุณสมบัติต่างๆของ group นั้นมาสามารถจะถูกปรับเปลี่ยนได้โดยสะดวก

6.3.7.2 Vertex Access

Direct3DRMMesh interface นั้นสนับสนุนฟังก์ชัน **GetVertices ( )** และ **setVertices ( )** สำหรับจัดการกับตำแหน่งของ vertices GetVertices ( ) นี้ใช้สำหรับรับค่าคุณสมบัติของ vertex นั้น ต่อจากนั้นเราสามารถปรับเปลี่ยนค่าของคุณสมบัติเหล่านี้ได้โดยใช้ฟังก์ชัน SetVertices ( ) ซึ่งมักจะใช้ในเทคนิคที่มีการเปลี่ยนแปลงรูปร่างของ mesh เช่น การทำ "vertex animation" และ "morphing"

Direct3DRMMesh interface นั้นมีฟังก์ชัน **Scale ( )** และ **Translate ( )** เหมือนกับ Direct3DRMMeshBuilder interface

6.3.7.3 การสร้าง Mesh จาก MeshBuilder

MeshBuilder นั้นสามารถจะใช้สร้าง Mesh ได้ โดยทั่วไปแล้ว MeshBuilder นั้นจะใช้สำหรับ load วัตถุ 3 มิติ ซึ่งอยู่ในรูปแบบของ mesh มาจากดิสก์ แล้วทำการกำหนด colors, textures, faces, vertices, normals แล้วใช้ฟังก์ชัน **CreateMesh ( )** ของ Direct3DRMMeshBuilder เพื่อทำการสร้าง Direct3DRMMesh object เทคนิคนี้เป็นเทคนิคที่สะดวกมาก แต่ทุกๆ faces ของ mesh นั้นจะรวมกันเป็น group เดียว ซึ่ง mesh ที่มีหลาย groups นั้นต้องสร้างได้โดยใช้ฟังก์ชัน **AddGroup ( )**

6.3.8 DIRECT3DRMFACE

ที่จริงแล้ว meshes นั้นคือ กลุ่มของ faces การสร้าง mesh ก็หมายถึงการสร้าง faces โดยมากเราจะทำการเปลี่ยนแปลง faces ที่มีอยู่แทนที่จะสร้างมันขึ้นมาใหม่ MeshBuilder interface นั้นอนุญาตให้เราเข้าถึง faces ที่มีอยู่โดยใช้ฟังก์ชัน GetFaces ( ) อย่างไรก็ตามเราก็ยังสามารถสร้าง faces ขึ้นมาใหม่เองและเพิ่มมันเข้าไปให้กับ MeshBuilder และ คุณสมบัติต่างๆของ faces เช่น color, texture, และ vertices นั้นสามารถจะถูกเปลี่ยนแปลงได้ โดยที่ faces นั้นถูกแสดงอยู่ในรูปของ Direct3DRMFace interface

Dirct3DRMFace interface นั้นสนับสนุนฟังก์ชัน **GetColor ( )** และ **SetColor ( )** เพื่อตรวจสอบและเปลี่ยนแปลงสีของ faces ตามลำดับ

6.3.8.1 Face Textures

เราสามารถกำหนด textures ให้กับ faces ได้โดยใช้ฟังก์ชัน เหล่านี้

- GetTexture ( )
- GetTextureCoordinateIndex ( )
- GetTextureCoordinates ( )
- GetTextureTopology ( )
- SetTexture ( )
- SetTextureCoordinates ( )
- SetTextureTopology ( )

6.3.8.2 Face Materials

Materials คือ การกำหนดคุณสมบัติด้านการสะท้อนแสงให้กับพื้นผิวของวัตถุ 3 มิติ ว่าจะให้มันมีความมันวาว, ด้าน, หรือ มีลักษณะเหมือนโลหะ เราสามารถตรวจสอบ และ ปรับเปลี่ยนแปลงค่าของ Material ให้กับ faces ได้โดยใช้ฟังก์ชัน **GetMaterial ( )** และ **SetMaterial ( )** ตามลำดับ

6.3.8.3 Face Vertices

เราสามารถจัดการกับ vertices ของ face ได้โดยใช้ฟังก์ชันเหล่านี้

- AddVertex ( )
- GetVertex ( )
- GetVertexCount ( )
- GetVertexIndex ( )
- GetVertices ( )

ความสามารถในการเพิ่ม vertices ให้กับ face นั้นสามารถจะทำให้เกิด "concave" faces ได้ ซึ่งในส่วนการแสดงผลของ Direct3D นั้นไม่สามารถประมวลผล concave faces ได้ แต่ว่าในส่วนของ Direct3D Retained-Mode นั้นจะไม่มีปัญหาเพราะ Direct3D Retained-Mode นั้นจะทำการแบ่ง faces ให้อยู่ในรูป triangles ให้โดยอัตโนมัติ

6.3.9 DIRECT3DRMTEXTURE

ใน Direct3D นั้น เราสามารถกำหนด textures ให้กับ faces, meshes หรือ ใช้เป็นภาพพื้นหลัง (background) ให้กับ scene ของเราได้โดยตรงในรูปแบบของ "background" หรือ "decal" Textures นั้นสามารถ load ได้จากไฟล์ภาพ bitmap (*.BMP) หรือ *.PPM, จาก program resource, หรือ จาก memory โดยที่ Texture นั้นถูกแสดงอยู่ในรูปของ Direct3DRMTexture interface

6.3.9.1 การสร้าง Textures

วิธีที่ง่ายที่สุดในการสร้าง Textures คือ ใช้ฟังก์ชัน **Direct3DRMLoadTexture ( )** ดังนี้

```
LPDIRECT3DRM texture;
d3drm->LoadTexture ("texture.bmp", &texture);
```

ฟังก์ชัน LoadTexture ( ) นั้นรับชื่อของไฟล์ *.BMP และ *.PPM เพื่อใช้สร้างเป็น texture และเรายังสามารถ load texture ได้จาก program resource โดยใช้ฟังก์ชัน **LoadTextureFromResource ( )** ดังนี้

```
LPDIRCT3DRMTEXTURE texture;
HRSRC id = FindResource (NULL, MAKEINTRESOURCE (IDR_SAMPLETEXTURE), "TEXTURE");
d3drm->LoadTextureFromResource (id, texture);
```

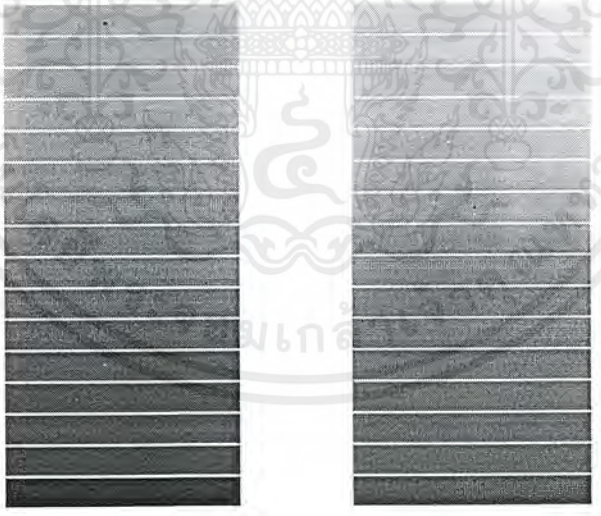
ฟังก์ชัน **LoadTextureFromResource ( )** นั้นรับค่า ID ของ resource และเราจะสร้าง texture จากค่า id ที่ได้มา

ซึ่งโดยการทำงานภายในแล้ว Direct3D นั้นจะใช้ DirectDraw surface ในการแสดง textures ซึ่ง เราสามารถสร้าง texture ได้โดยใช้ฟังก์ชัน **LoadTextureFromSurface ( )** เช่นกัน

6.3.9.2 Texture Colors

เมื่อ Direct3DRMTexture interface นั้นถูกสร้างขึ้นมา เราสามารถใช้ฟังก์ชัน **GetColors ( )** และ **SetColors ( )** เพื่อกำหนดจำนวนสีที่ Direct3D ใช้เพื่อแสดง texture และ ฟังก์ชัน **GetShades ( )** และ **SetShades ( )** เพื่อกำหนดว่าจะให้ texture ถูกแสดงออกมาอย่างไร

จำนวนของสีใน texture (จัดการโดยใช้ฟังก์ชัน GetColors ( ) และ SetColor ( )) หมายถึงจำนวนสีที่แตกต่างกันที่แสดงอยู่ใน texture ส่วนฟังก์ชัน GetShades ( ) และ SetShades ( ) จะใช้เพื่อควบคุมจำนวน shades ของแต่ละสีที่มีอยู่ใน texture ยกตัวอย่างเช่น ถ้า texture มีสี 2 สี ฟังก์ชัน GetColors ( ) นั้นจะส่งค่ากลับมาเท่ากับ 2 และ ฟังก์ชัน GetShades ( ) นั้นจะส่งค่ากลับมาเท่ากับ 16 (เพราะ 16 เป็นค่า default ของจำนวน shades) ในตัวอย่างนี้แต่ละสีจะมีจำนวน shades เท่ากับ 16 คือแต่ละสีจะมีระดับจากสีอ่อนไปหาแก่จำนวน 16 ระดับสีดังรูปที่ 6-12



รูปที่ 6-12 Shades ของสี จำนวน 16 shades

การลดจำนวนสีหรือ shades ของ texture นั้นจะทำให้คุณภาพของภาพที่ได้นั้นลดลง ตัวอย่างเช่น texture ที่ใช้สีจำนวนมากแต่ texture นั้นไม่ได้เป็นจุดสนใจภายใน scene เราสามารถจะลดจำนวนสีและจำนวน shades ของ texture นั้นได้ ส่วน texture ที่เป็นจุดสนใจภายใน scene นั้นควรกำหนดจำนวนสี และจำนวน shades ให้เหมาะสมเพื่อให้ผลลัพธ์ออกมาตรงกับความต้องการของเรา

6.3.9.3 Decals

Decal นั้นคือ texture ที่เรากำหนดให้กับ scene โดยที่ไม่ได้กำหนดให้กับ faces หรือว่า meshes ซึ่ง decal จะถูกแสดงใน scene โดยยึดตามตำแหน่งของของมัน และมันจะถูกแสดงเป็นภาพใหญ่ขึ้นหรือเล็กลงตามระยะใกล้ไกลจาก viewer เมื่อ decal อยู่ข้างหลังวัตถุ มันจะถูกปิดบังโดยวัตถุนั้น

Decal นั้นสามารถจะมองได้มุมเดียวคือหันเข้าหา viewer เสมอ และไม่สามารถแสดงจากมุมต่างๆได้ อย่างไรก็ตาม decal นั้นมีประโยชน์มากสำหรับการนำภาพ 2 มิติ มาแสดงภายใน scene ซึ่งเป็นเทคนิคที่ช่วยลดการคำนวณเกี่ยวกับ meshes และ faces ได้ เช่นใช้จำลองภาพการระเบิดของวัตถุ

Direct3DRMTexture interface มีหลายฟังก์ชันที่ใช้จัดการเกี่ยวกับ decal ดังนี้

- GetDecalOrigin ( )
- GetDecalScale ( )
- GetDecalSize ( )
- GetDecalTransparency ( )
- GetDecalTransparencyColor ( )
- SetDecalOrigin ( )
- SetDecalScale ( )
- SetDecalSize ( )
- SetDecalTransparency ( )
- SetDecalTransparencyColor ( )

ฟังก์ชันเกี่ยวกับ decal ส่วนมากจะใช้สำหรับ decal เท่านั้น แต่สำหรับฟังก์ชัน transparency นั้นสามารถใช้ได้สำหรับทั้ง decals และ textures ที่ถูกกำหนดให้กับ meshes หรือ faces

6.3.10 DIRECT3DRMTEXTUREWRAP

Texture wrap นั้นเป็นวัตถุที่อธิบายวิธีการที่ texture ถูกกำหนดให้ mesh ซึ่ง Texture wrap นั้นควบคุมรูปแบบการ wrap (flat, cylindrical, และ spherical), การวางตัวของ wrap, และคุณสมบัติของ texture เช่น scale, และ origin

Texture wrap นั้นถูกสร้างและกำหนดให้กับ MeshBuilder หรือ mesh โดยที่ texture wrap นั้นอธิบายว่า texture นั้นจะถูกกำหนดให้แต่ละ face ในรูปแบบใด ซึ่งช่วยประหยัดเวลาได้มากเพราะ mesh นั้นอาจมี faces ได้มากมายนับไม่ถ้วน

Texture wrap นั้น ถูกอธิบายโดย Direct3DRMTextureWrap interface และสามารถสร้างได้โดยใช้ฟังก์ชัน **CreateWrap ( )**

6.3.11 DIRECT3DRMMATERIAL

Materials จะเป็นตัวกำหนดพฤติกรรมของแสงที่กระทำต่อ faces และ meshes วัตถุสามารถที่จะมีลักษณะมันวาว หรือ ด้าน ได้

Direct3DRMMaterial interface สามารถสร้างได้โดยฟังก์ชัน CreateMaterial ( ) ซึ่ง Material interface นี้ อนุญาตให้เราเซตค่าได้ 3 ค่า คือ specular light power, specular light color, และ emissive light color โดยที่ material นี้สามารถจะกำหนดให้กับ faces, meshbuilders, และ meshes ได้

6.3.11.1 Specular Light Power

Specular light เป็นแสงซึ่งสะท้อนออกมาจากวัตถุและสร้าง "specular highlights" พฤติกรรมของ specular highlights จะมีผลกระทบกับลักษณะของวัตถุ ถ้าเรากำหนดให้มี highlight น้อยจะทำให้วัตถุเป็นมันวาว ถ้ากำหนดให้มี highlight มากจะทำให้วัตถุมีลักษณะพื้นผิวเหมือนพลาสติก ถ้ากำหนดให้มี highlight น้อยมากๆ หรือไม่มีเลย จะทำให้วัตถุมีลักษณะพื้นผิวด้าน

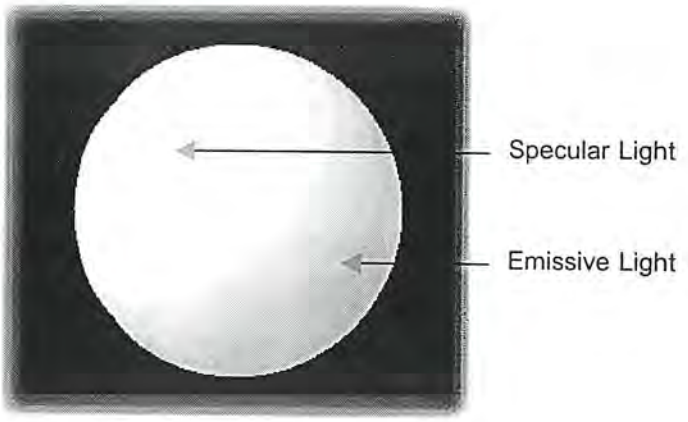
6.3.11.2 Specular Light Color

สีของ specular highlight นั้นสามารถจะถูกกำหนดได้โดยอิสระ ปกติจะเป็นสีขาว เราสามารถปรับสีของ highlight ได้โดยฟังก์ชัน **GetSpecular ( )** และ **SetSpecular ( )**

6.3.11.3 Emissive Light Color

Emissive light นั้นเป็นแสงที่ส่องออกมาจากตัววัตถุเอง มีประโยชน์ในกรณีที่เราต้องการจำลองแหล่งกำเนิดแสงเช่น โคมไฟ หรือ หลอดไฟนีออน ตามปกติแล้ววัตถุจะไม่มี emissive light (หรือ emissive light เป็นสีดำ) เราสามารถควบคุมค่า emissive light ได้โดยใช้ฟังก์ชัน **GetEmissive ( )** หรือ **SetEmissive ( )**

Ramp Color Model นั้นไม่สนับสนุนแสงที่มีสี แต่ emissive light นั้นไม่ได้ขึ้นอยู่กับแหล่งกำเนิดแสง ดังนั้นวัตถุสามารถจะมี emissive light ที่มีสีได้ทั้งใน Ramp และ RGB color models



รูปที่ 6-13 Specular และ Emissive light ของวัตถุ

6.3.12 DIRECT3DRMLIGHT

Direct3D สนับสนุน แหล่งกำเนิดแสง 5 ประเภท คือ ambient, point, directional, parallel, และ spotlight คุณสมบัติที่แหล่งกำเนิดแสงทุกๆชนิดมีเหมือนกันคือ color โดยที่แหล่งกำเนิดแสงทุกๆชนิดนั้นสามารถกำหนดตำแหน่งและการวางตัวได้ แต่แหล่งกำเนิดแสงบางประเภทนั้นก็ไม่ต้องการตำแหน่งและการวางตัว

แหล่งกำเนิดแสงนั้นถูกแสดงโดย Direct3DRMLigth interface และสามารถสร้างได้โดยฟังก์ชัน **CreateLight ( )** และ **CreateLithtRGB ( )** ทั้งสองฟังก์ชันนี้เราต้องกำหนดสี และ ชนิดของแหล่งกำเนิดแสงและคุณสมบัติต่างๆของแสงนั้นเราสามารถจะเปลี่ยนได้ตลอดเวลาโดยอิสระ

แหล่งกำเนิดแสงทุกชนิดจะอ้างอิงตำแหน่งและการวางตัวกับ frame ที่มันเชื่อมต่ออยู่ โดยที่เมื่อเราสร้างแหล่งกำเนิดแสงขึ้นมาแล้วเราต้องกำหนดมันให้กับ frame ก่อนนำไปใช้ใน scene โดยใช้ฟังก์ชัน **AddLight ( )** ของ Direct3DRMFrame interface

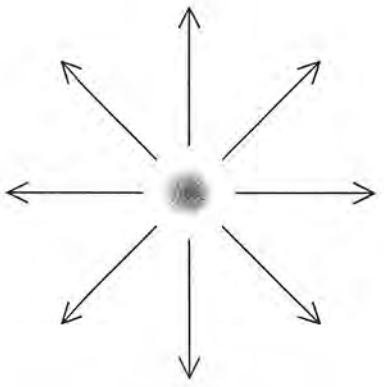
6.3.12.1 Ambient Lights

Ambient light นั้นเป็นแหล่งกำเนิดแสงที่ใช้ง่ายที่สุด เพราะเรากำหนดแค่ค่าของสีให้กับมัน ambient light นั้นจะต้องถูกกำหนดให้กับ frame ก่อนที่มันจะเป็นส่วนหนึ่งของ scene ซึ่งมันจะไม่สนใจตำแหน่งและการวางตัวของ frame ดังนั้น ambient light สามารถจะเชื่อมต่ออยู่กับ frame ใดก็ได้ภายใน scene

ใน scene นั้นสามารถจะมี ambient light เป็นจำนวนเท่าใดก็ได้ ผลลัพธ์ที่ได้จะเป็นการรวมค่าของทุกๆ ambient lights ภายใน scene ตัวอย่างเช่น เมื่อเรากำหนด ambient lights จำนวนสามดวงให้กับ scene มีสีแดง, เขียว, และ น้ำเงิน จะได้ผลเหมือนกับเรากำหนด ambient light ที่มีสีขาวให้กับ scene

6.3.12.2 Point Lights

Point light เป็นแหล่งกำเนิดแสงที่ส่องออกไปทุกทิศทาง เพราะฉะนั้นเราจะสนใจแค่ตำแหน่งของมันโดยไม่สนใจการวางตัวของมัน ดังรูปที่ 6-14

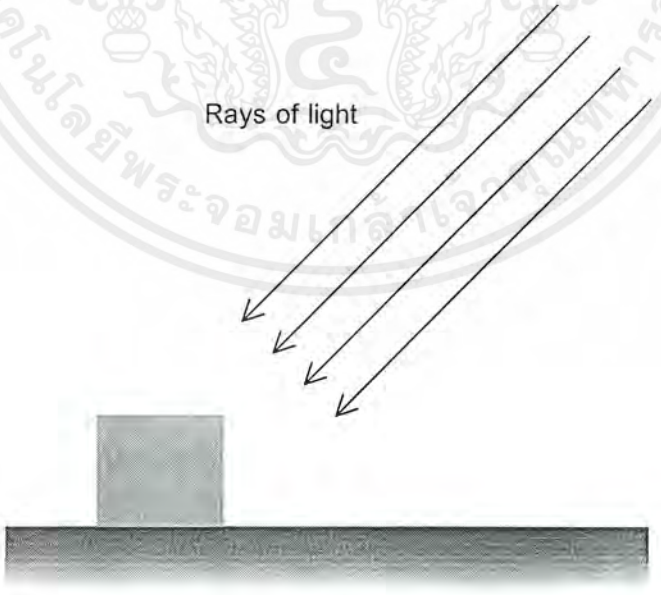


รูปที่ 6-14 แหล่งกำเนิดแสงแบบ Point Light

6.3.12.3 Directional Lights

Directional light นั้นตรงกันข้ามกับ point light นั้นมีแต่การวางตัวแต่ไม่มีตำแหน่ง แหล่งกำเนิดแสงแบบ directional light นั้นสร้างรังสีของแสงแบบขนาน (parallel light rays) ซึ่งมันจะไม่มีตำแหน่งของแหล่งกำเนิดแสง ทิศทางของแสงที่ส่องมานั้นจะยึดตาม frame ที่แหล่งกำเนิดแสงนั้นเชื่อมต่ออยู่

การคำนวณของแหล่งกำเนิดแสงแบบ directional light นั้นจะใช้เวลาในการประมวลผลน้อยกว่า point light และมีประโยชน์สำหรับการจำลองแหล่งกำเนิดแสงที่ส่องมาจากระทางที่ไกลมากๆเช่นแสงจากดวงอาทิตย์ ดังรูปที่ 6-15

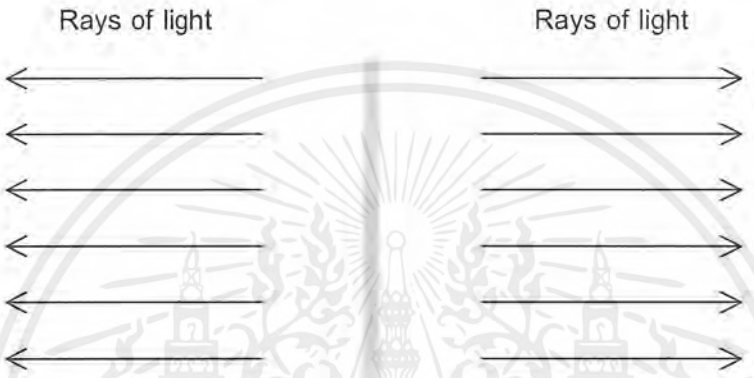


รูปที่ 6-15 แหล่งกำเนิดแสงแบบ Directional Light

6.3.12.4 Parallel Lights

แหล่งกำเนิดแสงแบบ parallel light เป็นประเภทหนึ่งของ แหล่งกำเนิดแสงแบบ directional lights โดย parallel lights นั้นจะสร้าง รังสีของแสงแบบขนาน (parallel light rays) แต่แทนที่จะส่องแสงออกมาในทิศทางเดียว ซึ่ง parallel light นั้นจะส่องแสงในสองทิศทางที่ตรงกันข้ามกัน ดังรูปที่ 6-16

Parallel light นั้น ต้องการทั้งตำแหน่งและการวางตัว โดยที่การวางตัวนั้นจะแสดงถึงทิศทางที่แสงจะส่องไป ส่วนตำแหน่งนั้นจะแสดงถึงระนาบที่แสงถูกสร้างขึ้น



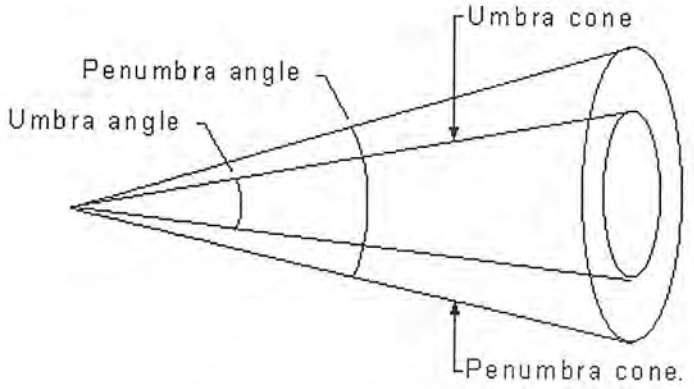
รูปที่ 6-16 แหล่งกำเนิดแสงแบบ Parallel Light

6.3.12.5 Spot Lights

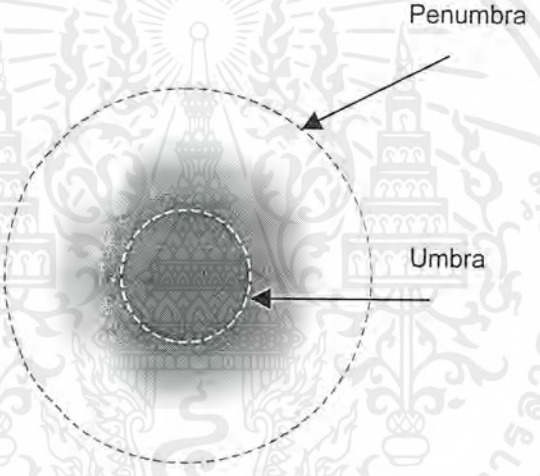
แหล่งกำเนิดแสงแบบ spot lights นั้นจะสร้างแสงในรูปแบบทรงกรวย (cone) โดยที่ spot light ใช้ตำแหน่งของ frame เพื่อแสดงถึงแหล่งกำเนิดของแสง (ยอดของทรงกรวย) และการวางตัวของ frame เป็นตัวแสดงทิศทางของแสงที่ส่องออกมา

แสงที่ถูกสร้างโดย spot light นั้นจะถูกอธิบายโดยมุมสองมุม คือ "Umbra" และ "Penumbra" มุมของ umbra นั้นแสดงถึงทรงกรวยที่แสงจะมีความเข้มเต็มที่ และ penumbra นั้นเป็นทรงกรวยที่ใหญ่กว่าซึ่งอธิบายจุดสิ้นสุดของแสง ดังรูปที่ 6-17 ระดับของแสงในช่วงระหว่างทรงกรวยของ umbra และ penumbra นั้นจะเริ่มจากเข้มกลายเป็นอ่อนและหายไปในที่สุดตามลำดับ ดังรูปที่ 6-18

มุมของ umbra และ penumbra สามารถถูกเปลี่ยนแปลงได้โดยฟังก์ชัน **SetUmbra ( )** และ **SetPenumbrea ( )** ของ Direct3DRMLight interface



รูปที่ 6-17 แหล่งกำเนิดแสงแบบ Spotlight



รูปที่ 6-18 ขอบเขตของ umbra และ penumbra

6.3.13 DIRECT3DRMSHADOW

Dirct3D นั้นถูกออกแบบมาโดยเน้นด้านประสิทธิภารของการทำงาน ดังนั้นต้องละทิ้งความถูกต้องและความเสมือนจริงบางอย่างไปบ้าง โดยการแสดงผลด้านเงา (shadow) เป็นตัวอย่างของความจริงนี้เพราะ Direct3D นั้นไม่สนับสนุนการคำนวณเงา โดยที่เงานั้นจะอยู่ในรูปแบบของวัตถุที่เราต้องเพิ่มเข้าไปเองภายใน scene และจำลองว่าเป็นเงาของวัตถุ ซึ่งการจำลองนี้เป็นการจำลองอย่างหยาบๆและมีข้อจำกัด ดังนี้

การสร้างเงานั้นต้องขึ้นอยู่กับวัตถุเหล่านี้

- วัตถุที่จะฉายเงา (casting shadow)
- แหล่งกำเนิดแสงที่จะใช้เพื่อคำนวณลักษณะของเงา
- ระนาบ (plane) ที่เงาจะไปปรากฏ

ข้อจำกัดแรกคือเราต้องกำหนดแหล่งกำเนิดแสงที่จะฉายเงา และแหล่งกำเนิดแสงแต่ละอันที่จะฉายเงานั้นต้องมี shadow object ของตัวเอง ซึ่งทำให้การใช้เทคนิคนี้ไม่ได้ทำได้ง่ายๆ

ข้อจำกัดอีกข้อคือตำแหน่งของเงานั้นถูกกำหนดโดยระนาบ ซึ่งหมายความว่าเราจะใช้เทคนิคนี้ได้บนวัตถุที่แบนราบเท่านั้น

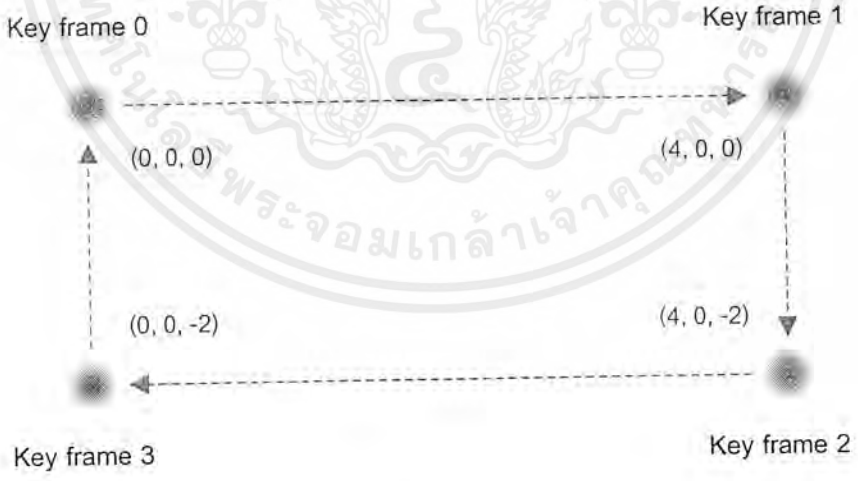
เราสามารถจัดการกับเงาได้โดยใช้ Direct3DRMShadow interface และสร้างเงาได้โดยใช้ฟังก์ชัน **CreateShadow ( )**

6.3.14 DIRECT3DRMANIMATION

ใน Direct3D นั้นคำว่าแอนิเมชัน (animation) นั้นมีอยู่สองความหมายคือ แอนิเมชันคือการเคลื่อนไหวของวัตถุภายใน scene แต่แอนิเมชันก็เป็นชื่อของ interface ที่สนับสนุนการสร้างและการแสดงผลแบบวนกลับไปกลับมา (playback) ของแอนิเมชันแบบ "key-framed"

6.3.14.1 การสร้าง Keys

Direct3DRMAnimation interface อนุญาตให้เราสร้างลำดับของแอนิเมชัน (animation sequence) โดยใช้ keys แต่ละ key คือ ค่าของการเลื่อนตำแหน่ง (translation), การหมุน (rotation), หรือ การย่อขยาย (scale) ที่กำหนดให้กับวัตถุที่จุดใดใดในแอนิเมชัน โดยที่ Animation interface จะสร้างแอนิเมชันทั้งหมดโดยพิจารณาจาก keys ที่เรากำหนดให้และคำนวณตำแหน่งของวัตถุระหว่าง key frames ให้โดยอัตโนมัติ ดังรูปที่ 6-19



รูปที่ 6-19 แอนิเมชันแบบ Key-framed

จากรูปที่ 6-19 แสดงให้เห็นว่าเมื่อเราต้องการสร้างแอนิเมชันให้วัตถุเคลื่อนที่เป็นรูปสี่เหลี่ยมตามเข็มนาฬิกาเรากำหนด keys ให้อยู่บริเวณมุมของรูปสี่เหลี่ยมเท่านั้นส่วน แอนิเมชันที่อยู่ระหว่าง key frames นั้น Direct3D จะทำการคำนวณให้เราเองว่าในช่วงเวลาใดวัตถุควรจะอยู่ที่ตำแหน่งใดระหว่าง key frames

Animation interface นั้นสนับสนุนการสร้าง key frames โดยฟังก์ชันเหล่านี้

- AddPositionKey ( )
- AddRotationKey ( )
- AddScaleKey ( )

แต่ละฟังก์ชันเหล่านี้ต้องการ "time" และ "translation" ซึ่ง timeจะเป็นตัวกำหนดตำแหน่งใน แอนิเมชันที่ key จะมีผลต่อแอนิเมชัน ส่วน translation นั้นคือ position, rotation, และ scale ที่จะใช้ที่ key frame ใดๆ ความยาวของแอนิเมชันนั้นจะถูกกำหนดโดย key frame ที่มีค่า time มากที่สุด เราสามารถยกเลิกการใช้ keys ใดๆ ได้โดยใช้ฟังก์ชัน **DeleteKey ( )**

6.3.14.2 การเซตค่า time ในแอนิเมชัน

เมื่อแอนิเมชันถูกสร้างขึ้นมาแล้วนั้น ฟังก์ชัน **SetTime ( )** นั้นสามารถถูกใช้เพื่อกำหนดตำแหน่งภายในแอนิเมชันได้ ค่าที่ส่งไปให้ฟังก์ชัน SetTime ( ) นั้นจะขึ้นอยู่กับค่า time ของ key frames ในแอนิเมชัน

ค่าของ time นั้นเป็นเลขทศนิยม ดังนั้นเราสามารถจะกำหนด keys ไว้ที่ใดก็ได้ภายในแอนิเมชัน แม้ว่าเราจะกำหนดให้มีความยาวเท่ากับ 1 แต่เราสามารถกำหนดให้มีการเปลี่ยนแปลงที่รวดเร็วหรือช้าได้ขึ้นกับค่าที่เพิ่มเข้าไปให้กับฟังก์ชัน SetTime ( ) ว่าจะมีค่ามากหรือน้อย หรือว่าเราจะให้แอนิเมชันเล่นแบบย้อยกลับก็ได้

เราสามารถสร้างแอนิเมชันได้ดังนี้

```
LPDIRECT3DRMANIMATION2 animation;        // pointer ที่ชี้ไปที่ Animation interface
pD3DRM->CreateAnimation (&animation);

// กำหนด keys เพื่อให้ได้แอนิเมชันดังรูปที่ 6-19
animation->AddPositionKey (0, 0, 0, 0);
animation->AddPositionKey (1, 4, 0, 0);
animation->AddPositionKey (2, 4, 0, -2);
animation->AddPositionKey (3, 0, 0, -2);
animation->AddPositionKey (0, 0, 0, 0);
```

จากตัวอย่างข้างบนฟังก์ชัน AddPositionKey ( ) ค่าที่ส่งให้กับฟังก์ชันนี้คือ time, position.x, position.y, และ position.z ตามลำดับ

เมื่อเราต้องการให้แอนิเมชันมีการเคลื่อนไหวนั้นเราจะทำการ update ค่า time ของแอนิเมชันในทุกๆ รอบที่แอปพลิเคชั่นของเราทำการประมวลผล ดังนี้

```
static currtime;            // เก็บค่า time ในการประมวลผลในรอบก่อนหน้าไว้
currtime += speed;
animation->SetTime (currtime);
```

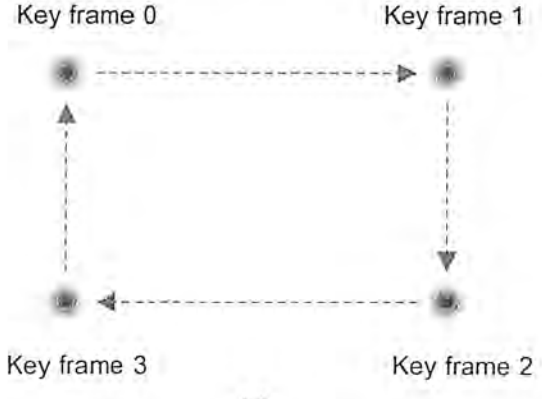
จากตัวอย่างข้างต้นจะเห็นได้ว่าความเร็วของแอนิเมชันนั้นอยู่ที่ค่า "speed" ที่เราทำการเพิ่มเข้าไปให้กับค่า "currtime" ซึ่งเป็นตัวบอกตำแหน่งปัจจุบันของแอนิเมชันในการประมวลผลรอบก่อนหน้านี้ การจองตัวแปร currtime เป็นแบบ static เพื่อให้ค่าของตัวแปร currtime ยังคงถูกเก็บรักษาไว้แม้ว่าจะออกจากฟังก์ชันไปแล้วเพื่อจะได้เก็บค่าไว้เพื่อทำการประมวลผลครั้งหน้า ดังค่า speed ที่เราเพิ่มเข้าไปเราต้องพิจารณาจากค่า time มากที่สุดของแอนิเมชันที่เราได้ทำการกำหนดไว้ด้วย และถ้าเรากำหนดค่า speed ให้เป็นลบ แอนิเมชันจะเล่นแบบย้อนกลับ

6.3.14.3 Animation Options

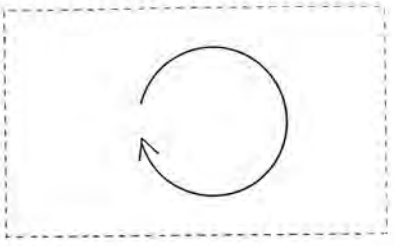
ฟังก์ชัน GetOptions ( ) และ SetOptions ( ) ถูกใช้เพื่อจัดการเกี่ยวกับพฤติกรรมของแอนิเมชัน ดังนี้

- แอนิเมชันแบบ linear หรือ spline
- แอนิเมชันแบบ open หรือ closed
- พิจารณา position
- พิจารณา scale และ rotation

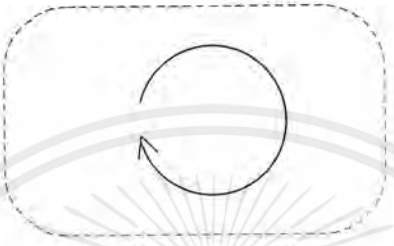
แอนิเมชันแบบ "linear" นั้นหมายความว่า animation interface จะทำการเคลื่อนที่วัตถุระหว่าง frames โดยคำนวณระยะทางที่สั้นที่สุดระหว่าง keys ส่วน แอนิเมชันในแบบ "spline-based" จะใช้เส้นโค้ง (curves) เพื่อ คำนวณตำแหน่งของวัตถุ ดังนั้นการเคลื่อนไหวจะเป็นไปในลักษณะที่โค้งแทนที่จะหักมุม ดังรูปที่ 6-20



(a)



(b)



(c)

รูปที่ 6-20 แอนิเมชันแบบ linear และ แบบ spline

จากรูปที่ 6-20 (a) คือ รูปแบบของ key frames ที่เรากำหนดให้กับแอนิเมชัน รูปที่ 6-20 (b) คือ รูปแบบการเคลื่อนไหวของแอนิเมชันแบบ "linear" และ รูปที่ 6-20 (c) คือรูปแบบการเคลื่อนไหวของแอนิเมชันแบบ "spline"

ส่วน "open" และ "close" นั้นเป็น options ที่กำหนดว่า animation object นั้นจะจัดการกับค่า time ที่ถูกกำหนดโดยฟังก์ชัน SetTime ( ) อย่างไรเมื่อค่า time นั้นมีค่ามากกว่าค่า time มากที่สุดของแอนิเมชัน (out-of-range) โดยที่แอนิเมชันในแบบ close นั้น เมื่อค่า time ที่เรากำหนดให้นั้นมากกว่าค่า time มากที่สุดของแอนิเมชัน ค่า time ที่เรากำหนดให้นั้นจะถูกนำมาลบด้วยค่า time ที่มากที่สุดของแอนิเมชัน ดังนั้นแอนิเมชันแบบ close นั้นจะเล่นวนไปเรื่อยแม้ว่าค่า time ที่เรากำหนดให้นั้นจะมากขึ้นเรื่อยๆ จนเกินกว่าค่า time ที่มากที่สุดของแอนิเมชัน ส่วนแอนิเมชันแบบ open นั้นเมื่อค่า time ที่เรากำหนดให้นั้นมากกว่าค่า time ที่มากที่สุดของแอนิเมชัน แล้วแอนิเมชันจะหยุดอยู่ที่ตำแหน่งที่มีค่าเวลามากที่สุดของแอนิเมชัน ซึ่งเราสามารถเริ่มต้นแอนิเมชันได้ใหม่โดยการกำหนดค่า time ให้เท่ากับค่า time เริ่มต้นของแอนิเมชันนั้น

"Use position" เป็น option ที่จะอนุญาตให้ใช้หรือไม่ให้ใช้ position keys ถ้าเราทำการปิด options นี้ position keys ที่เรากำหนดให้กับแอนิเมชันนั้นจะไม่มีผล ซึ่งเทคนิคนี้มีประโยชน์ในกรณีที่เราต้องให้ animation object ควบคุมแต่ scale และ rotation โดยไม่สนใจ position

"Use scale and rotation" เป็น option ที่อนุญาตให้ใช้หรือไม่ให้ใช้ scale และ rotation keys เมื่อเราปิด options นี้ scale และ rotation keys ที่เรากำหนดให้กับแอนิเมชันจะไม่มีผล ใช้เมื่อเราต้องการควบคุม scale และ rotation ของวัตถุโดยตรงและให้ animation object ทำการควบคุม position

6.3.15 DIRECT3DRMANIMATIONSET

Animation set นั้นเป็นการรวมหลายๆ animation object ไว้ด้วยกัน เหมาะสำหรับแสดงอนิเมชันของ scene ทั้งหมด โดยทั่วไป animation set นั้นจะถูกสร้างโดยการนำ scene ที่ถูกกำหนดแอนิเมชันแล้วจากโปรแกรมที่สามารถจัดการด้านแอนิเมชันได้เช่น "3D Studio" วัตถุแต่ละชิ้นภายใน scene นั้นจะถูกแสดงโดย animation object และ animation objects ทั้งหมดจะถูกรวมเป็น animation set

Direct3DRMAnimationSet interface นั้นถูกใช้เพื่อควบคุม animation sets โดยถูกสร้างจากฟังก์ชัน **CreateAnimationSet ( )** ของ Direct3DRM

6.3.15.1 Loading Animation Sets

Animation sets นั้นสามารถจะทำการ load เข้ามาใช้ได้โดยใช้ฟังก์ชัน **Load ( )** ของ Direct3DRMAnimationSet ชื่อของไฟล์ที่ถูกกำหนดให้กับฟังก์ชัน Load ( ) ต้องเป็นไฟล์ที่รวบรวมแอนิเมชันทั้งหมดใน scene เหมือนกับฟังก์ชัน load อื่นๆใน Direct3D ฟังก์ชัน load นั้นสามารถจะ load ได้จากไฟล์, program resources, หรือ memory

แต่ละ animation object นั้นสามารถที่จะถูกเพิ่มหรือลบออกจาก AnimationSet object โดยใช้ฟังก์ชัน **AddAnimation ( )** และ **DeleteAnimation ( )**

6.3.15.2 การเซตค่า time ใน animation sets

โดยที่ Direct3DRMAnimationSet interface นั้นสนับสนุนฟังก์ชัน **SetTime ( )** ซึ่งในกรณีของ AnimationSet ฟังก์ชัน SetTime ( ) จะทำการเซตค่า time สำหรับแต่ละ animation ที่ถูกรวมอยู่ใน animation set

6.3.16 ชนิดของข้อมูลใน Direct3D

ใน Direct3D นั้นมีชนิดของข้อมูลอยู่หลายแบบ ดังนี้

6.3.16.1 D3DVALUE

D3DVALUE นั้นเป็นชนิดของข้อมูลที่เป็นพื้นฐานที่สุดใน Direct3D โดยที่ D3DVALUE นั้นถูกประกาศเป็นตัวแปรในรูปแบบ **float** ดังรูปที่ 6-21 และใช้ใน Direct3D เพื่อแสดงถึง vertex coordinates, light intensities, rotation speeds และ อื่นๆอีก

ในระบบการพัฒนาแอปพลิเคชั่นแบบ 32-bit โดยใช้คอมไพเลอร์ C++ เช่น Visual C++ ถ้าเราไม่กำหนดชนิดของข้อมูลอย่างเจาะจงให้กับคอมไพเลอร์ จะเลือกชนิดของข้อมูลแบบ **int** (สำหรับเลขจำนวนเต็ม ซึ่งมีขนาด 2 bytes) หรือ **double** (สำหรับเลขทศนิยม ซึ่งมีขนาด 8 bytes) โดยที่เราไม่ได้กำหนดชนิดของข้อมูลเลขทศนิยมคอมไพเลอร์จะใช้ชนิดของข้อมูลขนาด 8 bytes (double) แทนที่จะใช้ 4 bytes (float)

```
typedef float D3DVALUE, *LPD3DVALUE;
typedef DWORD D3DCOLOR, D3DCOLOR, *LPD3DCOLOR;
```

รูปที่ 6-21 การกำหนดชนิดของข้อมูล D3DVALUE และ D3DCOLOR

ด้วยเหตุนี้การใช้ข้อมูลชนิด D3DVALUE โดยตรงนั้นจะทำให้เราสามารถย้าย platform ของแอปพลิเคชั่นของเราได้ง่าย แต่ถ้าเราแน่ใจว่าไม่มีโอกาสที่แอปพลิเคชั่นของเราต้องย้าย platform และเราไม่ต้องการใช้ชนิดของข้อมูลแบบ D3DVALUE แล้วเราสามารถเราสามารถใช้คำเสริมท้าย 'f' ได้เพื่อบอกกับคอมไพเลอร์ว่าข้อมูลของเรามีชนิดเป็น float ดังตัวอย่างต่อไปนี้

```
d3dfunction ( D3DVALUE ( 3 ) );              // ถูกต้อง
d3dfunction ( D3DVALUE ( 3.0 ) );            // ถูกต้อง
d3dfunction ( 3 );      // คอมไพเลอร์จะพิจารณาเป็น int และแจ้งข้อความเตือนเพราะ Direct3D นั้น
                           ต้องการข้อมูลในรูปแบบ float
d3dfunction ( 3.0 );    // คอมไพเลอร์จะพิจารณาเป็น double และแจ้งข้อความเตือนเพราะ Direct3D
                           นั้นต้องการข้อมูลในรูปแบบ float
d3dfunction ( 3.0f );   // ถูกต้องแต่อาจมีปัญหาตอนย้าย platform
d3dfunction ( 3f );     // ผิดเพราะต้องใช้เลขทศนิยมเท่านั้น
```

6.3.16.2 D3DVECTOR

D3DVECTOR ถูกกำหนดโครงสร้างดังนี้

```
typedef struct _D3DVECTOR {
    union {
        D3DVALUE x;
        D3DVALUE dvX;
    };
    union {
        D3DVALUE y;
        D3DVALUE dvY;
    };
```

```
    union {
        D3DVALUE z;
        D3DVALUE dvZ;
    };
} D3DVECTOR, *LPD3DVECTOR;
```

การใช้ "union" เพื่อให้เราเลือกใช้อย่างใดอย่างหนึ่งระหว่าง x หรือ dvX (เป็นตัวอย่าง) โดยที่ไม่ต้องจองหน่วยความจำทั้งสำหรับ x และ dvX โดยที่ D3DVECTOR นั้นจะถูกใช้ใน Direct3D เพื่อแสดงถึงทั้ง vectors และ points

6.3.16.3 D3DCOLOR

Direct3D นั้นใช้ข้อมูลในรูปแบบ D3DCOLOR เพื่อแสดงถึงสี (colors) ซึ่งข้อมูลของสีนี้จะประกอบไปด้วย red, green, blue, และ alpha ซึ่งค่าเหล่านี้สามารถจะเป็นได้ตั้งแต่ 0 ถึง 1 โดยที่ 0 หมายถึงไม่มีหรือปิด ส่วน 1 หมายถึงค่าเต็มที่หรือเปิด

อันที่จริงแล้วชนิดของข้อมูล D3DCOLOR ตรงกับ ชนิดข้อมูลแบบ **DWORD** ดังรูปที่ 6-21 แต่ไม่สามารถเก็บค่าเลขทศนิยมได้

การจัดเก็บค่า D3DCOLOR จะทำโดย คูณค่าสีแต่ละค่า (red, green, blue, alpha) ด้วย 255 และทำการ "Shift" เข้าไปใน DWORD โดยที่ Direct3D นั้นมี macros สำหรับจัดการเรื่องเหล่านี้ เราสามารถกำหนดค่าให้กับตัวแปรที่เป็นข้อมูลในรูปแบบ D3DCOLOR โดยใช้ macro **D3DRGB** หรือ **D3DRGBA** ได้ดังนี้

```
D3DCOLOR color = D3DRGB (1, 1, 1);          // สร้างสีขาว
```

หรือ

```
D3DCOLOR color = D3DRGBA (1, 1, 1, 0);      // สร้างสีขาวที่มีค่า alpha เท่ากับ 0
```

ค่าข้อมูลที่ส่งให้ macros D3DRBG และ D3DRGBA นั้นต้องมีค่าอยู่ระหว่าง 0 ถึง 1 โดยที่ macros นั้นจะจัดการเกี่ยวกับการคูณและแปลงค่าให้เอง เราไม่จำเป็นต้องใส่ค่าในรูปแบบของ D3DVALUE เพราะ macros จะจัดการให้อยู่แล้ว

เราสามารถดูค่าส่วนประกอบของสี (red, green, blue, alpha) ได้จาก D3DCOLOR โดยใช้ฟังก์ชัน **D3DRMColorGetRed ( )**, **D3DRMColorGetGreen ( )**, **D3DRMColorGetBlue ( )**, และ **D3DRMColorGetAlpha ( )**

6.3.16.4 D3DRMVERTEX

Direct3D ใช้โครงสร้างข้อมูล D3DRMVERTEX เพื่ออธิบาย vertices ที่อยู่ภายใน mesh ดังนี้

```
typedef struct _D3DRMVERTEX {
        D3DVECTOR position;
        D3DVECTOR normal;
        D3DVALUE tu, tv,
        D3DCOLOR color;
} D3DRMVERTEX;
```

ตัวแปร **position** นั้นมีโครงสร้างข้อมูลแบบ D3DVECTOR ซึ่งอธิบายตำแหน่งของ vertex ตัวแปร **normal** เป็นใช้อธิบาย normal vector ของ vertex (ใช้ใน Gouraud shading) หรือ เป็น normal vector ของ face (ใช้ใน Flat shading) ตัวแปร **tu** และ **tv** ใช้สำหรับอธิบาย coordinate ที่อยู่ภายใน texture ที่นำมา "map" กับ vertex ส่วน ตัวแปร **color** นั้นแสดงถึง ค่าสีของ vertex

6.3.16.5 D3DRMQUANTERNION

Quaternions นั้นใช้อธิบายเกี่ยวกับการหมุน (rotation) ซึ่งโดยปกตินั้นเราการสามารถอธิบายการหมุนได้โดยใช้ vector และ ค่าการหมุน (rotation value) เพื่ออธิบาย ซึ่ง vector นั้นจะอธิบายแกนที่จะทำการหมุน ส่วน ค่าการหมุนนั้นจะเป็นตัวบอกว่าหมุนไปเท่าใด Quaternions นั้นได้รวมทั้ง vector และ ค่าการหมุนไว้ในโครงสร้างเดียวกัน ดังนี้

```
typedef struct _D3DRMQUATERNION {
        D3DVALUE s;
        D3DVECTOR v;
} D3DRMQUATERNION;
typedef D3DRMQUATERNION, *LPD3DRMQUATERNION;
```

เราสามารถกำหนดค่าเริ่มต้นให้กับข้อมูลในรูปแบบ Quaternions ได้โดยใช้ฟังก์ชัน D3DRMQuaternionFromRotation ( ) โดยฟังก์ชันนี้จะรับค่าของ vector ที่ใช้เป็นแกนในการหมุน และค่าการหมุน และสร้างข้อมูลในรูปแบบ quaternion ขึ้นมา

Quaternions นั้นมีประโยชน์สำหรับการคำนวณค่ากึ่งกลางระหว่างสองค่าการหมุน (rotation values) โดยฟังก์ชัน D3DRMQuaternionSlerp ( ) นั้นจะรับค่าของ quaternions สองค่า และค่า "Slerp value" ฟังก์ชันจะทำการคำนวณ quaternion ที่อยู่ระหว่าง vectors ทั้งสอง ส่วน slerp value นั้นจะเป็นตัวกำหนดว่าค่า quaternion ใหม่ที่ได้จะอยู่ใกล้ ด้านใดมากกว่าระหว่าง quaternions ทั้งสองที่เรากำหนดให้กับฟังก์ชัน ถ้าเราใช้

slerp value เท่ากับ 0.5 แล้ว ผลลัพธ์จะได้ quaternion ที่อยู่ตรงกึ่งกลางระหว่างสอง quaternions ที่เรากำหนดให้กับฟังก์ชันพอดี

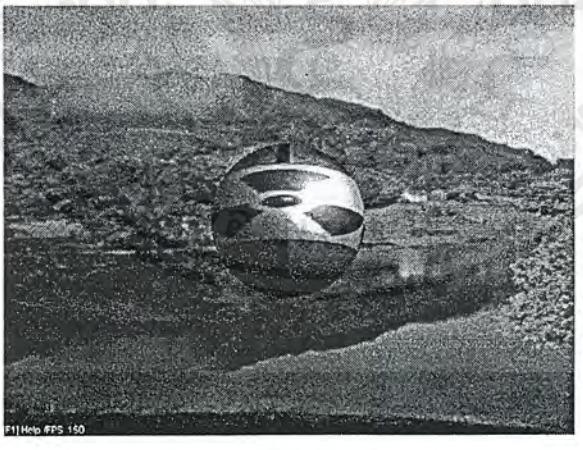
6.3.16.6 HRESULT

จุดสำคัญของฟังก์ชันทั้งหลายใน DirectX คือการส่งค่า HRESULT กลับ ซึ่งแสดงถึงความผิดพลาด (error) ที่เกิดขึ้น โดยที่ HRESULT นั้นเป็นค่าข้อมูลขนาด 32-bit ซึ่งบอกถึงผลของฟังก์ชันที่เราเรียกใช้งาน Direct3D นั้นมีค่าคงที่ (constant) มากมายที่จะใช้ทดสอบและเปรียบเทียบกับค่า HRESULT ที่ถูกส่งคืนมาหลังจากที่เราเรียกใช้ฟังก์ชันต่างๆ ในทางอุดมคติแล้ว ฟังก์ชันใน Direct3D Retained-Mode ที่ส่งคืนค่า D3DRM_OK กลับนั้นแสดงว่าการทำงานนั้นสำเร็จ ถ้าฟังก์ชันทำงานไม่สำเร็จค่าความผิดพลาดที่เป็นสาเหตุที่ทำให้ฟังก์ชันทำงานไม่สำเร็จจะถูกส่งกลับออกมา

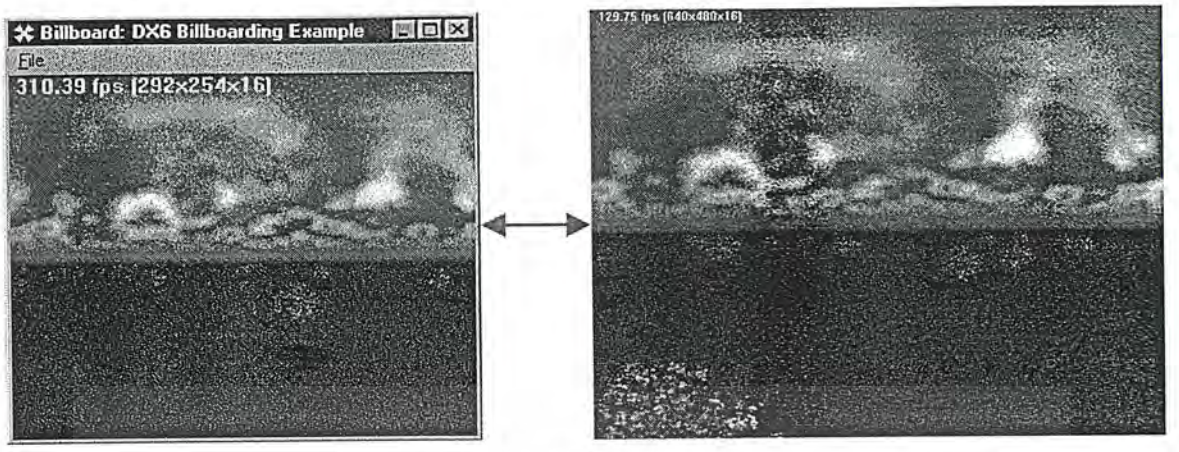
6.4 รูปแบบของแอปพลิเคชั่นที่ใช้ Direct3D Retained-Mode

แอปพลิเคชั่นที่ใช้ Direct3D ทั้ง Immediate Mode และ Retained-Mode นั้นสามารถจะทำงานได้ทั้งใน Full-screen mode และ windowed mode หรือว่าจะกำหนดให้แอปพลิเคชั่นของเราทำงานได้ทั้งใน Full-screen mode และ windowed mode โดยอนุญาตให้ผู้ใช้เลือกได้ว่าต้องการใช้งานใน mode ใด ดังรูปที่ 6-22

(a)



(b)



(c)

รูปที่ 6-22 รูปแบบของแอปพลิเคชั่นที่ใช้ Direct3D

จากรูปที่ 6-22 (a) นั้นแสดงแอปพลิเคชั่นที่ใช้ Direct3D ใน windowed mode รูปที่ 6-22 (b) แสดงแอปพลิเคชั่นที่ใช้ Direct3D ใน full-screen mode และรูปที่ 6-22 (c) แสดงแอปพลิเคชั่นที่ใช้ Direct3D ในรูปแบบที่ผู้ใช้สามารถเลือกได้ว่าจะใช้งานใน mode ใดและสามารถเปลี่ยน mode การใช้งานได้ตลอดเวลา

6.5 โครงสร้างของแอปพลิเคชั่นที่ใช้ Direct3D Retained-Mode

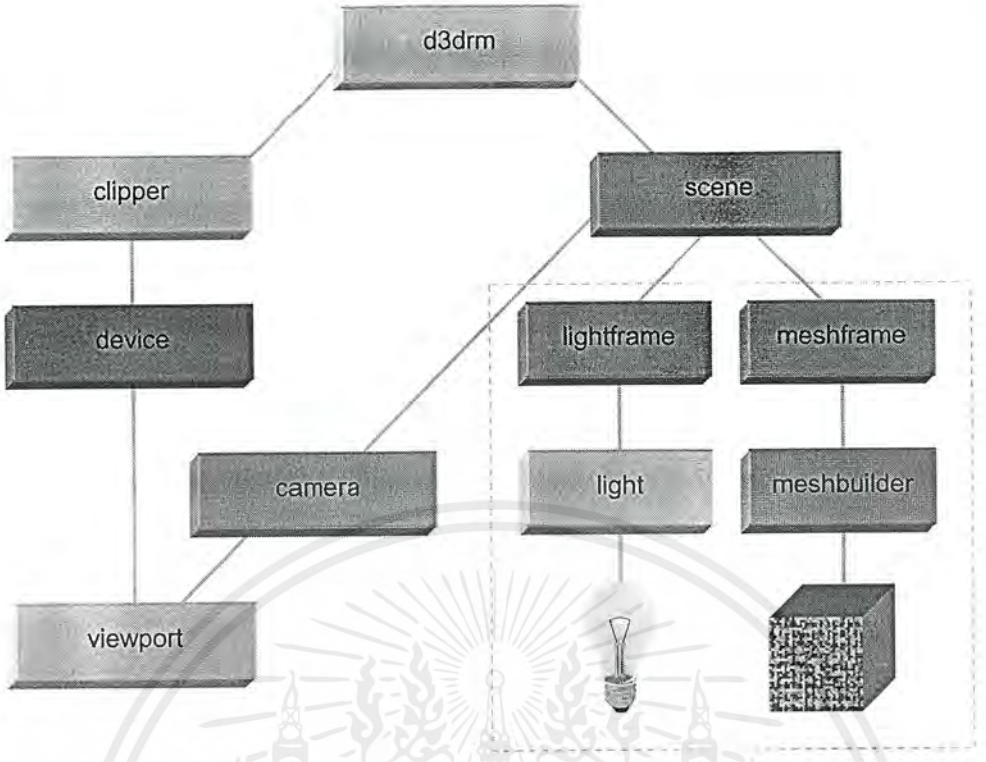
แอปพลิเคชั่นที่ใช้ Direct3D Retained-Mode นั้นจะมีโครงสร้างอยู่ 3 แบบ ซึ่ง การเลือกว่าเราจะใช้โครงสร้างแบบใดนั้นเราจะพิจารณาว่าเหมาะสมกับลักษณะของแอปพลิเคชันของเราหรือไม่เช่นจะให้แอปพลิเคชั่นของเราทำงานงานในแบบ full-screen mode, windowed mode หรือ ทั้งสองแบบ ซึ่งโครงสร้างทั้ง 3 แบบนี้จะมีผลต่อลักษณะของแอปพลิเคชั่นของเราโดยตรง

ส่วนที่สำคัญที่สุดที่จะเป็นตัวแบ่งแยกประเภทของโครงสร้างข้างต้นนั้นอยู่ที่ การสร้าง “device” ซึ่งเป็นตัวสร้างผลลัพธ์ทางด้านภาพจาก scene ของเรา โดยการสร้าง device นั้นมีอยู่ 3 แบบ ดังนี้

- CreateDeviceFromClipper ( )
- CreateDeviceFromSurface ( )
- CreateDeviceFromD3D ( )

6.5.1 โครงสร้างของแอปพลิเคชั่นที่ใช้ฟังก์ชัน CreateDeviceFromClipper ( )

โครงสร้างนี้เป็นโครงสร้างที่ง่ายที่สุด และ แอปพลิเคชั่นที่ใช้โครงสร้างนี้นั้นสามารถจะทำงานได้ใน windowed mode เท่านั้นและไม่สามารถใช้ DirectDrawSurface ร่วมกับการแสดงผลในแอปพลิเคชั่นได้ นั่นหมายความว่าเราไม่สามารถเขียนตัวหนังสือหรือนำภาพจาก bitmap ไฟล์มาใช้ในแอปพลิเคชั่นของเราได้ โดยโครงสร้างแสดงในรูปที่ 6-23



รูปที่ 6-23 โครงสร้างของแอปพลิเคชั่น แบบที่ 1

จากรูปที่ 6-23 เป็นโครงสร้างที่แสดงถึงลำดับในการสร้างองค์ประกอบต่างๆ เราจะเห็นได้ว่าเราต้องสร้าง clipper ก่อนแล้วจึงสร้าง device โดยที่ device นั้นจะถูกสร้างโดยอ้างอิงจาก clipper อีกที ความสัมพันธ์ของโครงสร้างโดยละเอียดมีดังนี้ (ขออ้างอิงกับ DirectX SDK เวอร์ชัน 6.0)

d3drm นั้นคือ Direct3DRM เราสามารถสร้างได้ฟังก์ชัน ดังตารางที่ 6-1

HRESULT Direct3DRMCreate (LPDIRECT3DRM FAR * lplpD3DRM);
lplpD3DRM : address ของ pointer ที่ชี้ไปที่ Direct3DRM interface

ตารางที่ 6-1 ฟังก์ชัน Direct3DRMCreate

ตัวอย่างการใช้งาน

```
LPDIRECT3DRM pD3DRM;          // pointer ที่ชี้ไปที่ Direct3DRM interface
Direct3DRMCreate (&pD3DRM);   // ทำการ initialize ให้กับ pointer
```

หลังจากนั้นเราจะทำการร้องขอ Direct3DRM3 interface โดยใช้ฟังก์ชัน QueryInterface ( ) ซึ่งมีตัวอย่างการใช้งานดังนี้

ตัวอย่างการใช้งาน

```
LPDIRECT3DRM3 pD3DRM3;              // pointer ที่ชี้ไปที่ Direct3DRM3 interface
pD3DRM->QueryInterface (IID_IDirect3DRM3, (LPVOID *)&pD3DRM3);
                                    // ร้องขอ pointer ที่ชี้ไปที่ Direct3DRM3_interface
```

clipper นั้นคือ DirectDrawClipper ซึ่งสามารถสร้างได้จากฟังก์ชัน ดังตารางที่ 6-2

```
HRESULT WINAPI DirectDrawCreateClipper      ( DWORD dwFlags,
                                              LPDIRECTDRAWCLIPPER FAR *lplpDDClipper,
                                              IUnknown FAR *pUnkOuter );

dwFlags          : ไม่ใช้ กำหนดเป็น 0
lplpDDClipper    : address ของ pointer ที่ชี้ไปที่ DirectDrawClipper interface
pUnkOuter        : ไม่ใช้ กำหนดเป็น NULL
```

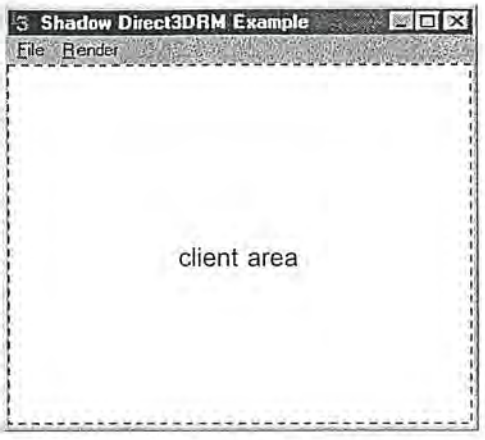
ตารางที่ 6-2 ฟังก์ชัน DirectDrawCreateClipper

ตัวอย่างการใช้งาน

```
LPDIRECTDRAWCLIPPER clipper;                // pointer ที่ชี้ไปที่ DirectDrawClipper interface
DirectDrawCreateClipper (0, &clipper, NULL);    // ทำการ initialize ให้กับ pointer
```

นอกจากการสร้าง clipper โดยใช้ฟังก์ชัน DirectDrawCreateClipper ( ) แล้วเรายังสามารถสร้าง clipper ได้โดยใช้ฟังก์ชัน **CreateClipper ( )** ของ DirectDraw interface ได้เช่นเดียวกัน

เมื่อสร้าง clipper แล้วเราต้องใช้ฟังก์ชัน **SetHwnd ( )** ดังตารางที่ 6-3 เพื่อกำหนด clipper ที่เราสร้างขึ้นมาให้กับ window ของแอปพลิเคชั่นของเราซึ่ง clipper จะเป็นตัวกำหนดขอบเขตการแสดงผลให้อยู่ในช่วง "client area" ใน window ของแอปพลิเคชั่นของเรา ดังรูปที่ 6-24



รูปที่ 6-24 client area ของ window

HRESULT SetHWnd(	DWORD dwFlags,
	HWND hWnd);
dwFlags	: ไม่ได้ใช้ กำหนดเป็น 0
hWnd	: ค่า handle ของ window ของแอปพลิเคชั่น

ตารางที่ 6-3 ฟังก์ชัน SetHwnd

ตัวอย่างการใช้งาน

```
clipper->SetHwnd (0, hWnd);     // hWnd คือ ค่า handle ของ window ของแอปพลิเคชั่น
```

device นั้นคือ Direct3DRMDevice3 ซึ่งเราใช้ฟังก์ชัน CreateDeviceFromClipper ( ) เพื่อสร้าง ดังตารางที่ 6-4

การสร้าง device นั้นเราต้องให้ค่า GUID (globally unique identifier) ซึ่งเป็นค่าที่ชี้ไปที่ driver ของ Dirct3D ที่เราต้องการใช้ โดยทั่วๆไปแล้วจะมีอยู่ 3 ชนิดด้วยกัน คือ **Ramp**, **RGB**, และ **HAL** ซึ่ง Ramp และ RGB นั้นเป็น driver แบบ "Software" และ HAL นั้นเป็น driver แบบ "Hardware" ซึ่งจะมีอยู่บนเครื่องคอมพิวเตอร์ที่ติดตั้งการ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ ไว้เท่านั้น เราสามารถกำหนดให้ค่าเท่ากับ NULL ได้ เพื่อให้สร้าง device จาก driver ที่เป็น default ซึ่งโดยปกติจะเป็น driver ในรูปแบบของ Ramp

เนื่องจากฟังก์ชัน CreateDeviceFromClipper ( ) นั้นต้องการให้เรากำหนดความกว้างและความสูงของ device ที่เราจะสร้าง ดังนั้นเราจะกำหนดให้เท่ากับความกว้างและความสูงของ client area ของ window ของแอปพลิเคชั่น เราสามารถรู้ความกว้างและความสูงของ client area ได้โดยใช้ฟังก์ชัน **GetClientRect ( )**

```
HRESULT CreateDeviceFromClipper(LPDIRECTDRAWCLIPPER lpDDClipper,
                                LPGUID lpGUID,
                                int width,
                                int height,
                                LPDIRECT3DRMDEVICE3 * lplpD3DRMDevice);
```

lpDDClipper	: pointer ที่ชี้ไปที่ DirectDrawClipper interface
lpGUID	: pointer ที่ชี้ไปที่ globally unique identifier
width	: ความกว้างของ device ที่จะสร้าง
height	: ความสูงของ device ที่จะสร้าง
lplpD3DRMDevice	: address ของ pointer ที่ชี้ไปที่ Direct3DRMDevice interface

ตารางที่ 6-4 ฟังก์ชัน CreateDeviceFromClipper

ตัวอย่างการใช้งาน

```
RECT rect;
GetClientRect (hWnd, &rect);          // หาขนาดของ client area ขณะนั้น
LPDIRECT3DRMDEVICE3 device;
pD3DRM3->CreateDeviceFromClipper (clipper, NULL, rect.right, rect.bottom, &device);
```

scene นั้นคือ Direct3DRMFrame3 ซึ่ง scene นี้ถูกกำหนดให้เป็น "root frame" ของ scene ทั้งหมด ซึ่งเราสามารถสร้าง frame ใดๆ ได้โดยใช้ฟังก์ชัน **CreateFrame ( )** ดังตารางที่ 6-5

```
HRESULT CreateFrame( LPDIRECT3DRMFRAME3 lpD3DRMFrame,
                     LPDIRECT3DRMFRAME3* lplpD3DRMFrame);
```

lpD3DRMFrame	: pointer ที่ชี้ไปที่ parent frame
lplpD3DRMFrame	: address ของ pointer ที่ชี้ไปยัง Direct3DRMFrame3 interface

ตารางที่ 6-5 ฟังก์ชัน CreateFrame ( )

ตัวอย่างการใช้งาน

```
LPDIRECT3DRMFRAME3 scene;
pD3DRM3->CreateFrame (NULL, &scene);          // กำหนดให้ parent frame เป็น
                                              NULL เพราะ scene เป็น root frame
```

camera นั้นคือ Direct3DRMFrame3 ซึ่งเป็นตัวอธิบายว่าเราจะแสดง scene ของเราจากตำแหน่งใด และในรูปแบบการวางตัวอย่างไร เราสามารถสร้าง camera โดยยึด scene เป็น parent frame ได้ดังนี้

ตัวอย่างการใช้งาน

```
LPDIRECT3DRMFRAME3 camera;
pD3DRM3->CreateFrame (scene, &camera);        // ใช้ scene เป็น parent frame
```

viewport นั้นคือ Direct3DRMViewport2 ซึ่งจะทำหน้าที่เป็นเสมือนกล้องถ่ายภาพจริงๆที่สามารถปรับ "focus" และค่าอื่นๆได้ เช่น field-of-view, front-clipping, back-clipping ฯลฯ ซึ่งอย่าสับสนกับ camera เพราะอันที่จริง camera นั้นเป็น frame ซึ่งแสดงถึงตำแหน่งและการวางตัวของกล้องถ่ายภาพเท่านั้น เราสามารถจะเปลี่ยนชื่อของ camera เป็นอย่างอื่นที่สื่อความหมายได้มากกว่านี้มากกว่านี้ก็ได้เช่น "camera_position"

การสร้าง viewport นั้นเราต้องสร้างโดยอ้างอิงจาก camera และ device โดยใช้ฟังก์ชัน **CreateViewport ( )** ดังตารางที่ 6-6 โดยที่ การสร้าง viewport นั้นเราต้องให้ตำแหน่งเริ่มต้นของ viewport ซึ่งเราจะกำหนดให้เป็น 0 และ ความกว้างและความสูงของ viewport นั้นเราจะพิจารณาจากค่าความกว้างและความสูงของ device ที่เราสร้างขึ้นก่อนหน้านี้ โดยเราจะสามารถทราบความกว้าง และความสูงได้จากฟังก์ชัน **GetWidth ( )** และ **GetHeight ( )** ของ device ตามลำดับ

ตัวอย่างการใช้งาน

```
LPDIRECT3DRMVIEWPORT2 viewport;
pD3DRM3->CreateViewport (device,
                         camera,
                         0, 0,
                         device->GetWidth ( ), device->GetHeight ( ),
                         &viewport);
```

```
HRESULT CreateViewport(LPDIRECT3DRMDEVICE3 lpDev,
                       LPDIRECT3DRMFRAME3 lpCamera,
                       DWORD dwXPos,
                       DWORD dwYPos,
                       DWORD dwWidth,
                       DWORD dwHeight,
                       LPDIRECT3DRMVIEWPORT2* lplpD3DRMViewport);
```

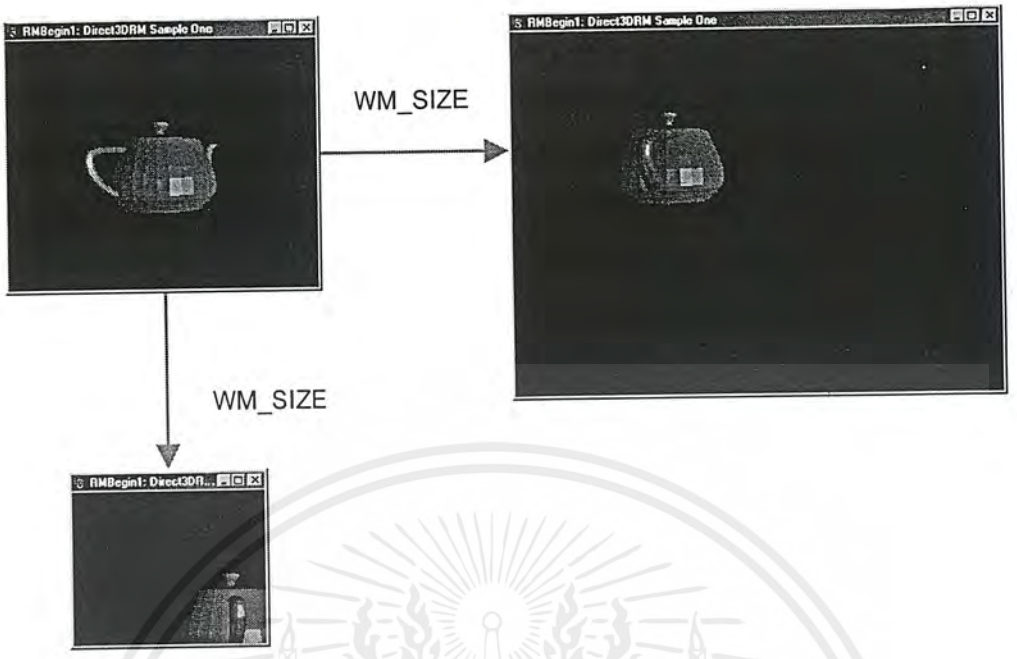
lpDev	: pointer ที่ชี้ไปที่ Direct3DRMDevice3 interface
lpCamera	: pointer ที่ชี้ไปที่ camera frame
dwXPos	: ตำแหน่ง X เริ่มต้นของ viewport เมื่อเทียบกับ device
dwYPos	: ตำแหน่ง Y เริ่มต้นของ viewport เมื่อเทียบกับ device
dwWidth	: ความกว้างของ viewport
dwHeight	: ความสูงของ viewport
lplpD3DRMViewport	: address ของ pointer ที่ชี้ไปที่ Direct3DRMViewport2 interface

ตารางที่ 6-6 ฟังก์ชัน CreateViewport

เมื่อมาถึงขั้นตอนนี้แอปพลิเคชั่นของนั้นก็พร้อมที่จะแสดงผลแล้วซึ่งแอปพลิเคชั่นของเราจะทำการแสดงผลวัตถุต่างๆที่เรากำหนดให้กับ scene เช่น meshes, lights, textures, animations ฯลฯ ซึ่งงานในส่วนนี้จะขึ้นอยู่กับแต่ละแอปพลิเคชั่นว่าจะมีรูปแบบของ scene เป็นอย่างไรซึ่งในส่วนนี้จะอยู่ในกรอบเส้นประ ของรูปที่ 6-23

ในแอปพลิเคชั่นของ Direct3D Retained-Mode ที่ทำงานในแบบ windowed mode นั้น เราต้องตอบสองกับ "windows messages" ที่ส่งเข้ามายังแอปพลิเคชั่นของเราหลักๆอยู่ 3 แบบ คือ

WM_SIZE ซึ่งจะถูกส่งเข้ามายังแอปพลิเคชั่นของเราเมื่อ window ของแอปพลิเคชั่นของเรามีการเปลี่ยนแปลงขนาด ถ้าเราต้องการใช้ภาพที่ได้ออกมานั้นขยายและย่อขนาดตามขนาดของ window หมายความว่า ขนาดของภาพนั้นจะอยู่เต็มภายใน client area เสมอ ดังนั้นเราต้องมาพิจารณา WM_SIZE กรณีของ WM_SIZE ที่เข้ามานั้นมีอยู่ 3 กรณี คือ windows มีขนาดเท่าเดิม, ย่อ, หรือ ขยาย ซึ่งถ้า window มีขนาดเท่าเดิมนั้นเราก็ไม่ต้องทำการเปลี่ยนแปลงอะไร ส่วน ถ้า window ถูกย่อขนาดให้เล็กลงนั้น เราจะต้องทำการสร้าง viewport ใหม่ให้มีขนาดเท่ากับ window ที่ถูกย่อลงโดยที่เราสามารถให้ device เดิมได้ ส่วนในกรณีของ window ที่ถูกขยายให้ใหญ่ขึ้นนั้นเราต้องทำการสร้างทั้ง device และ viewport ใหม่ ตามลำดับ



รูปที่ 6-25 ลักษณะของแอปพลิเคชั่นที่ไม่ตอบสองต่อ WM_SIZE

จากรูปที่ 6-25 นั้นแสดง แอปพลิเคชั่นที่ไม่ตอบสนองต่อ WM_SIZE ซึ่งเมื่อผู้ใช้ทำการย่อหรือขยายขนาดของ window แล้วสัดส่วนของภาพนั้นจะเท่าเดิมซึ่งไม่สัมพันธ์กับขนาดของ client area

WM_ACTIVATE จะถูกส่งเข้ามาเมื่อ window ถูก "activated" หรือ ถูก "deactivated" เช่นเมื่อเราทำการ minimize, restore หรือ ใช้เมาส์คลิกบริเวณ window เพื่อให้ window กลับมาอยู่ในสถานะ active อีก ซึ่งเราสามารถตอบสนอง WM_ACTIVETE ได้โดยใช้ฟังก์ชัน **HandleActivate ( )** ของ Direct3DRMWinDevice interface ดังนี้

ตัวอย่างการใช้งาน

```
case WM_ACTIVATE:
{
        LPDIRECT3DRMWINDEVICE windev;
        device->QueryInterface (IID_IDirect3DRMWinDevice, (void **)&windev);
        windev->HandleActivete (wparam);
        windev->Release ( );
}
```

WM_PAINT นั้นจะถูกส่งเข้ามาเมื่อ window ของแอปพลิเคชั่นต้องการ repaint การแสดง ตัวอย่าง เช่น เมื่อ ถูก window ของแอปพลิเคชั่นอื่นมาทับและเมื่อเลื่อน window ที่มาทับนั้นออกไป WM_PAINT จะถูกส่งเข้ามาที่แอปพลิเคชั่นของเรา เราใช้ฟังก์ชัน **HandlePaint ( )** เพื่อจัดการกับ WM_PAINT ที่ถูกส่งเข้ามาเพื่อป้องกันการ repaint ในพื้นที่บน device ที่อยู่นอก viewport เพราะในกรณีที่ window ของแอปพลิเคชั่นถูกย่อให้เล็กลงนั้นเราไม่ได้ทำการสร้าง device ใหม่ เราสร้าง viewport ใหม่เท่านั้น ดังนั้นขนาดของ device จึงมีขนาดใหญ่กว่าขนาดของ viewport แต่ถ้าในกรณีที่ window ของแอปพลิเคชั่นนั้นถูกขยายให้มีขนาดใหญ่ขึ้น เราจะทำการสร้างทั้ง device และ viewport ใหม่ ซึ่งทั้ง device และ viewport จะมีขนาดที่เท่ากัน

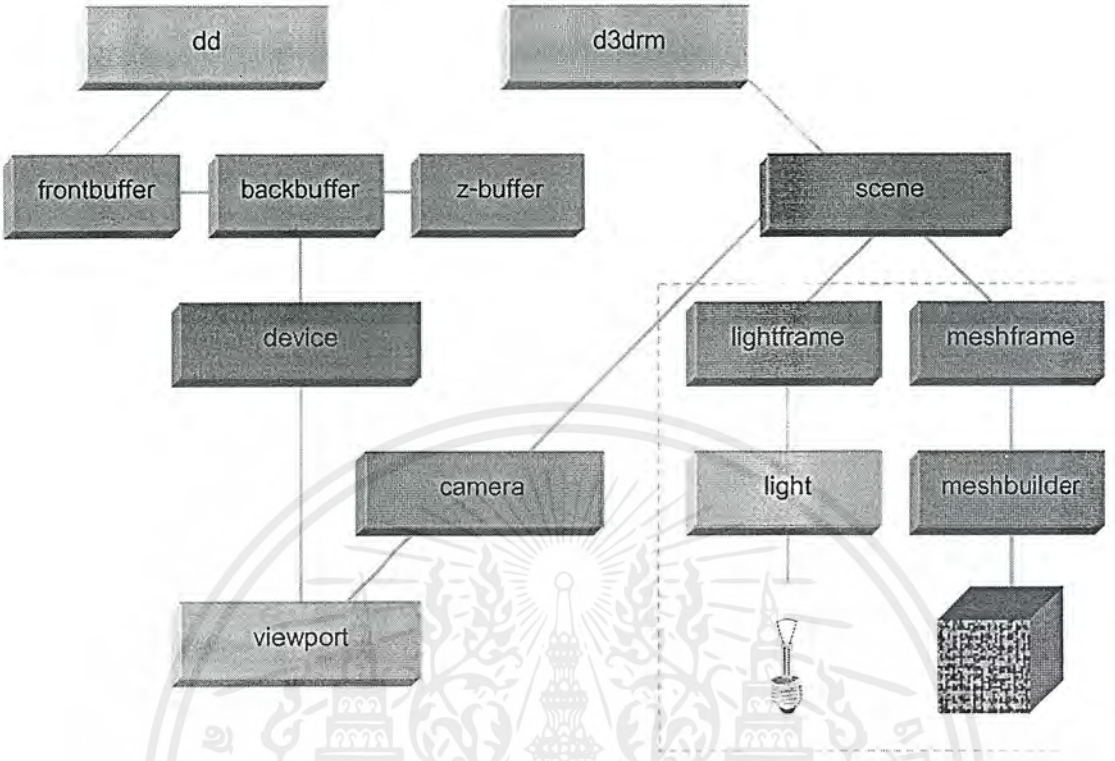
ตัวอย่างการใช้งาน

```
case WM_PAINT:
{
        PAINTSTRUCT ps;
        LPDIRECT3DRMWINDEVICE windev;
        BeginPaint (hWnd, &ps);
        device->QueryInterface (IID_IDirect3DRMWinDevice, (void **)&windev);
        windev->HandlePaint (ps.hdc);
        windev->Release ( );
        EndPaint (hWnd, &ps);
}
```

6.5.2 โครงสร้างของแอปพลิเคชั่นที่ใช้ฟังก์ชัน CreateDeviceFromSurface ( )

จากโครงสร้างข้างต้นเราจะเห็นได้ว่าเป็นโครงสร้างที่มีความเกี่ยวข้องกับ Direct3D Retained-Mode ล้วนๆ ซึ่งเป็นโครงสร้างที่มีความซับซ้อนไม่มากนัก แต่จะสามารถใช้ได้เฉพาะใน windowed mode เท่านั้น และไม่สามารถใช้ร่วมกับ DirectDrawSurface ได้ ซึ่งโครงสร้างที่ใช้ฟังก์ชัน CreateDeviceFromSurface ( ) นี้ เราจะสามารถใช้ได้ทั้ง windowed mode และ full-screen mode และยังสามารถใช้ร่วมกับ DirectDrawSurface ได้ ทำให้เราสามารถเขียนตัวหนังสือ หรือว่านำภาพที่จากไฟล์ภาพ bitmap มาใช้ในแอปพลิเคชั่นของเราได้ โดยโครงสร้างนี้จะมีความซับซ้อนมากขึ้น ซึ่งจะมีความเกี่ยวข้องกับ DirectDraw ด้วย ดังรูปที่ 6-26

โดยที่โครงสร้างของแอปพลิเคชั่นแบบนี้เราจะต้องสร้าง DirectDrawSurfaces ทั้งหมด 3 อันด้วยกัน คือ **frontbuffer** (primary surface), **backbuffer** (secondary surface), และ **z-buffer** โดยที่การสร้าง surfaces เหล่านี้เราต้องพิจารณาร่วมกับค่า GUID ที่เราใช้กำหนด driver ที่จะใช้สร้าง device ด้วยซึ่งตรงส่วนนี้นั้นมีความสำคัญมาก ซึ่งมีรายละเอียด ดังตารางที่ 6-7



รูปที่ 6-26 โครงสร้างของแอปพลิเคชั่น แบบที่ 2

	Software Driver (HEL)	Hardware Driver (HAL)
backbuffer	System or Video memory	Video memory
z-buffer	System memory	Video memory

ตารางที่ 6-7 ความสัมพันธ์ระหว่าง surfaces กับประเภทของ driver

จากตารางที่ 6-7 นั้นเราสามารถสรุปได้ว่าการสร้าง backbuffer นั้นถ้าเราต้องการใช้ software driver สำหรับสร้าง device นั้นเราสามารถที่จะกำหนดให้ backbuffer ที่เราสร้างขึ้นนั้นอยู่ใน System memory หรือ Video memory ก็ได้ ส่วน ถ้าเราต้องการใช้ hardware driver สำหรับการสร้าง device นั้นเราต้อง กำหนดให้ backbuffer อยู่ใน Video memory เท่านั้น

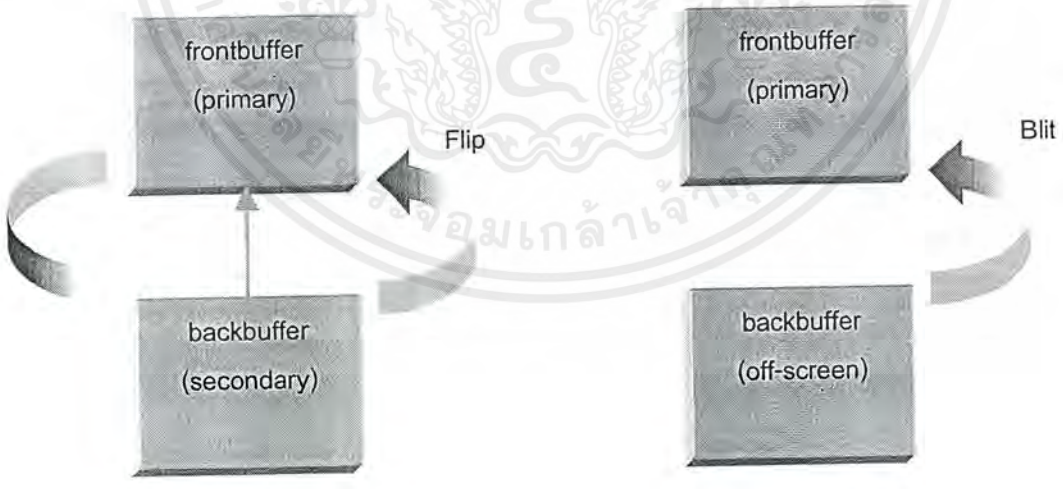
ส่วน z-buffer นั้น เราต้องการใช้ software driver ในการสร้าง device เราต้องกำหนดให้ z-buffer อยู่ใน System memory เท่านั้น ส่วนถ้าเราต้องการใช้ hardware driver ในการสร้าง device เราต้องกำหนดให้ z-buffer อยู่ใน Video memory เท่านั้น

และจากที่ได้กล่าวไปแล้วว่าแอปพลิเคชั่นที่ใช้โครงสร้างนี้นั้นสามารถที่จะทำงานได้ทั้งในรูปแบบ windowed mode และ full-screen mode ซึ่งการที่เราจะกำหนดให้แอปพลิเคชั่นทำงานในรูปแบบใดนั้นอยู่ตรงที่การเรียกใช้ฟังก์ชัน **SetCooperativeLevel ( )** ของ DirectDraw ซึ่งมีตัวอย่างดังนี้

ตัวอย่างการใช้งาน

```
// ใช้ full-screen mode
pDD->SetCooperativeLevel (hWnd,  DDSCL_EXCLUSIVE | DDSCL_FULLSCREEN);

// ใช้ windowed mode
pDD->SetCooperativeLevel (hWnd, DDSCL_NORMAL);
```

การสร้าง surfaces นั้นนอกจากจะต้องพิจารณาถึงประเภทของ device ที่เราจะสร้างขึ้นแล้วนั้นเรายังต้องพิจารณาอีกด้วยว่าแอปพลิเคชั่นของเรานั้นจะทำงานในรูปแบบใดระหว่าง full-screen mode กับ windowed mode ซึ่งโครงสร้างของ frontbuffer และ backbuffer นั้นจะแตกต่างกัน โดยที่ใน full-screen mode นั้น เราสร้าง frontbuffer และ backbuffer ในรูปแบบของ “flipping surfaces” เพื่อให้สามารถใช้เทคนิค page flipping ได้ ส่วนใน windowed mode นั้นไม่สามารถใช้เทคนิค page flipping ได้ ดังนั้นเราจะสร้าง backbuffer โดยกำหนดให้เป็น surface แบบ “off-screen” เมื่อเราทำการ update backbuffer เรียบร้อยแล้ว เราจะแสดงผลโดยใช้คำสั่ง **Blt ( )** ของ DirectDrawSurface จาก backbuffer ไปสู่ frontbuffer ดังรูปที่ 6-27



รูปที่ 6-27 โครงสร้าง surfaces

จากรูปที่ 6-27 (a) เราจะเห็นว่าแอปพลิเคชั่นในรูปแบบ full-screen mode นั้นมีการสร้าง frontbuffer และ backbuffer ที่เป็นแบบ primary และ secondary คือ backbuffer ถูกเชื่อมต่ออยู่กับ frontbuffer ทำให้เราสามารถใช้เทคนิค flipping ได้ แต่ถ้าเป็นแอปพลิเคชั่นในรูปแบบ full-screen mode แล้วเราต้องใช้เทคนิคที่เรียกว่า "double buffers" แทน ซึ่ง frontbuffer จะไม่ได้เชื่อมต่ออยู่กับ backbuffer ดังรูปที่ 6-27 (b) ซึ่งการสร้าง surfaces ต่างๆนั้นมีตัวอย่างดังนี้

ตัวอย่างการใช้งาน

```
// ในกรณีของ full-screen mode
DDSURFACEDESC ddsd;
DDSCAPS ddscaps;
LPDIRECTDRAWSURFACE frontbuffer;
LPDIRECTDRAWSURFACE backbuffer;

memset (&ddsd, 0, sizeof (DDSURFACEDESC));
ddsd.dwSize = sizeof (ddsd);
ddsd.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE | DDSCAPS_FLIP |
                      DDSCAPS_3DDEVICE | DDSCAPS_COMPLEX;
ddsd.dwBackBufferCount = 1;

if ( // ใช้ hardware driver // )
        ddsd.ddsCaps.dwCaps |= DDSCAPS_VIDEOMEMORY;

// สร้าง frontbuffer และ backbuffer โดย initialize ค่า pointer ที่ชี้ไปยัง frontbuffer
pDD->CreateSurface (&ddsd, &frontbuffer, NULL);


// ขณะนี้ทั้ง frontbuffer และ backbuffer นั้นได้ถูกสร้างขึ้นมาแล้ว โดย backbuffer จะเชื่อมต่ออยู่กับ
// frontbuffer ซึ่งเราสามารถ initialize ค่า pointer ที่ชี้ไปยัง backbuffer ได้โดย
ddscaps.dwCaps = DDSCAPS_BACKBUFFER;
frontbuffer->GetAttachedSurface (&ddscaps, &backbuffer);
```

```
// ในกรณีของ windowed mode
DDSURFACEDESC ddsd;
LPDIRECTDRAWSURFACE frontbuffer;
LPDIRECTDRAWSURFACE backbuffer;

memset (&ddsd, 0, sizeof (DDSURFACEDESC));
ddsd.dwSize = sizeof (ddsd);
ddsd.dwFlags = DDSD_CAPS;
ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE;

// สร้าง frontbuffer
pDD->CreateSurface (&ddsd, &frontbuffer, NULL);

// เตรียมสร้าง backbuffer
ddsd.dwFlags = DDSD_WIDTH | DDSD_HEIGHT | DDSD_CAPS;
ddsd.dwWidth = w;
ddsd.dwHeight = h;
ddsd.ddsCaps.dwCaps = DDSCAPS_OFFSCREENPLAIN | DDSCAPS_3DDEVICE;

if ( // ใช้ hardware driver // )
        ddsd.ddsCaps.dwCaps |= DDSCAPS_VIDEOMEMORY;
else
        ddsd.ddsCaps.dwCaps |= DDSCAPS_SYSTEMMEMORY;

// สร้าง backbuffer ที่เป็นแบบ off-screen
pDD->CreateSurface (&ddsd, &backbuffer, NULL);

// สร้าง DirectDraw Clipper เชื่อมต่อเข้ากับ window ของแอปพลิเคชั่น และ frontbuffer
LPDIRECTDRAWCLIPPER clipper;

pDD->CreateClipper (0, &clipper, NULL);
clipper->SetHWnd (0, hWnd);
frontbuffer->SetClipper (clipper);
```

ในส่วนของ **z-buffer** ทั้งในแอปพลิเคชั่นแบบ full-screen mode และ windowed mode นั้นมีรูปแบบการสร้างที่เหมือนกัน เมื่อเราสร้าง z-buffer แล้วเราต้องนำ z-buffer ที่เราสร้างขึ้น ไปเชื่อมต่อเข้ากับ backbuffer ซึ่ง z-buffer นั้นจะมีหน้าที่เกี่ยวกับการพิจารณาเกี่ยวกับระยะของวัตถุในแกน Z (Z-axis) ถ้าปราศจาก z-buffer แล้ววัตถุที่อยู่ไกลกว่าอาจถูกนำมาแสดงทับลงบนวัตถุที่อยู่ใกล้กว่า ซึ่งจะทำให้เกิดการแสดงผลที่ผิดพลาดจากความเป็นจริง เราสามารถสร้าง z-buffer และ เชื่อมต่อมันเข้ากับ backbuffer ได้ดังนี้

ตัวอย่างการใช้งาน

```
DDSURFACEDESC ddsd;
LPDIRECTDRAWSURFACE zbuff;

// เตรียมสร้าง z-buffer
memset (&ddsd, 0, sizeof (DDSURFACEDESC));
ddsd.dwSize = sizeof (DDSURFACEDESC);
ddsd.dwFlags = DDSD_WIDTH | DDSD_HEIGHT | DDSD_CAPS | DDSD_ZBUFFERBITDEPTH;
ddsd.dwWidth = w;
ddsd.dwHeight = h;
ddsd.ddsCaps.dwCaps = DDSCAPS_ZBUFFER;
ddsd.dwZBufferBitDepth = 16;

if ( // ใช้ hardware driver // )
        ddsd.ddsCaps.dwCaps |= DDSCAPS_VIDEOMEMORY;
else
        ddsd.ddsCaps.dwCaps |= DDSCAPS_SYSTEMMEMORY;

// สร้าง z-buffer
pDD->CreateSurface (&ddsd, &zbuff, NULL);

// เชื่อมต่อ z-buffer เข้ากับ backbuffer
backbuffer->AddAttachedSurface (zbuff);
```

สุดท้ายเราจะทำการสร้าง device โดยใช้ฟังก์ชัน CreateDeviceFromSurface ( ) ซึ่งมีรูปแบบดัง ตารางที่ 6-8

```
HRESULT CreateDeviceFromSurface( LPGUID lpGUID,
                                 LPDIRECTDRAW lpDD,
                                 LPDIRECTDRAWSURFACE lpDDSBack,
                                 DWORD dwFlags,
                                 LPDIRECT3DRMDEVICE3* lplpD3DRMDevice);
```

lpGUID	: pointer ที่ชี้ไปที่ globally unique identifier
lpDD	: pointer ที่ชี้ไปที่ DirectDraw
lpDDSBack	: pointer ที่ชี้ไปที่ backbuffer
dwFlags	: กำหนดให้เท่ากับ D3DRMDEVICE_NOZBUFFER
lplpD3DRMDevice	: address ของ pointer ที่ชี้ไปที่ device

ตารางที่ 6-8 ฟังก์ชัน CreateDeviceFromSurface ( )

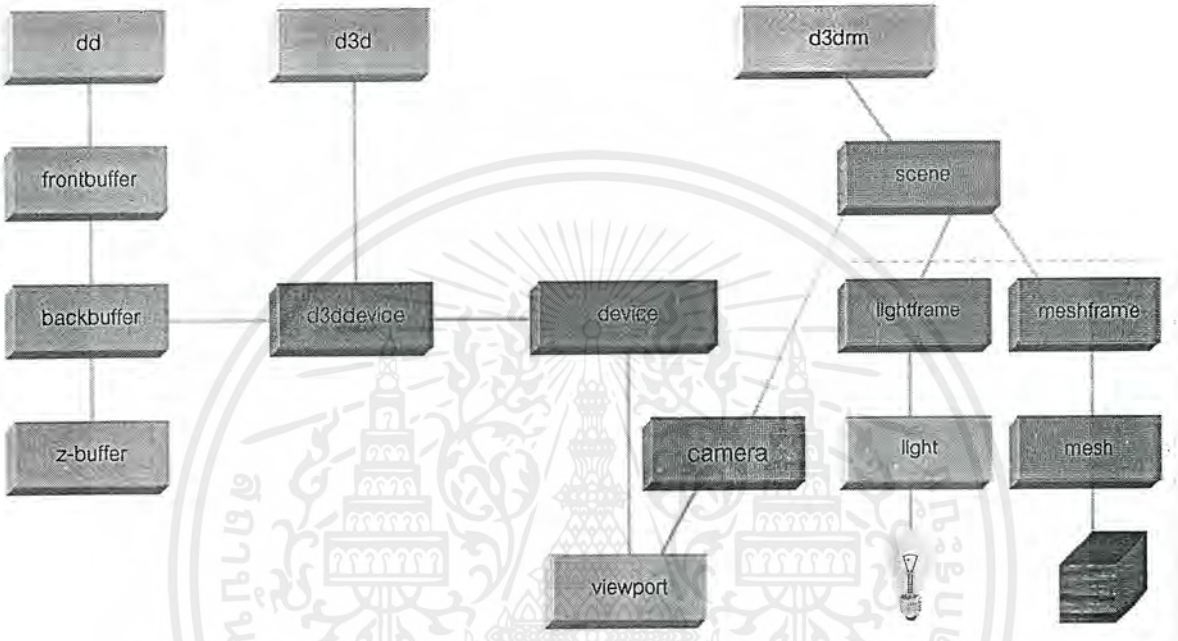
ตัวอย่างการใช้งาน

```
LPDIRECT3DRMDEVICE3 device;

pD3DRM->CreateDeviceFromSurface (NULL, pDD, backbuffer,
                                 D3DRMDEVICE_NOZBUFFER,
                                 &device);
```

จากตัวอย่างข้างต้นนั้นเราจะเห็นว่า แม้เราจะใช้ DirectX SDK เวอร์ชัน 6.0 ในการพัฒนาแอปพลิเคชั่น แต่เรายังสามารถจะใช้ได้แค่ DirectDraw เวอร์ชัน 1.0 ในแอปพลิเคชั่นของเราเพราะ device นั้นจะถูกสร้างโดยอาศัย DirectSurface เวอร์ชัน 1.0 เท่านั้น โดยเราสังเกตได้จากรูปแบบของฟังก์ชัน ในตารางที่ 6-8 ที่ใช้ "LPDIRECTDRAWSURFACE" แทนที่จะใช้ "LPDIRECTDRAWSURFACE4" ซึ่งมีอยู่ใน DirectX เวอร์ชัน 6.0 และจากฟังก์ชันข้างต้นเราพบว่าเราไม่ต้องกำหนดขนาดให้กับ device ที่เราจะสร้างเหมือนฟังก์ชัน CreateDeviceFromClipper ( ) เพราะว่า Direct3D จะสร้าง device โดยยึดจากขนาดของ backbuffer นั่นเอง

6.5.3 โครงสร้างของแอปพลิเคชั่นที่ใช้ฟังก์ชัน CreateDeviceFromD3D ( )

แอปพลิเคชั่นที่ใช้ฟังก์ชัน CreateDeviceFromD3D นั้นจะมีความสามารถเท่ากับ แอปพลิเคชั่นที่ใช้ฟังก์ชัน CreateDeviceFromSurface ( ) คือสามารถใช้ได้ทั้งใน full-screen mode และ windowed mode และสามารถใช้ DirectDrawSurface ได้ด้วย แต่โครงสร้างสุดท้ายนี้จะมีความซับซ้อนมากที่สุดเพราะมีส่วนประกอบทั้งของ DirectDraw, Direct3D, และ Direct3DRM ดังรูปที่ 6-28



รูปที่ 6-28 โครงสร้างของแอปพลิเคชั่นแบบที่ 3

รากรูปที่ 6-28 เราจะเห็นว่าโครงสร้างของแอปพลิเคชั่นที่ใช้ฟังก์ชัน CreateDeviceFromD3D ( ) นั้นมีความคล้ายคลึงกับแอปพลิเคชั่นที่ใช้ฟังก์ชัน CreateDeviceFromSurface ( ) มาก จะต่างกันก็ตรงที่ เราต้องสร้าง **d3d** คือ Direct3D object ขึ้นมาก่อนแล้วสร้าง **d3ddevice** ซึ่งเป็น device ของ Direct3D Immediate Mode แล้วเราจึงสร้าง **device** ซึ่งเป็นเป็น device ของ Direct3D Retained-Mode อีกที เราสามารถสร้าง d3d ได้ดังนี้

ตัวอย่างการใช้งาน

```
LPDIRECT3D2 pD3D;
pDD->QueryInterface (IID_IDirect3D2, (void **)&pD3D);
```

จากตัวอย่างเราสามารถสร้าง Direct3D object ได้จากฟังก์ชัน QueryInterface ( ) ของ DirectDraw เพราะว่า Direct3D นั้นถูกพัฒนาโดยตรงมาจาก DirectDraw เราสามารถสร้าง Direct3DDevice ได้โดยใช้ฟังก์ชันดังตารางที่ 6-9 และเราสามารถสร้าง Direct3DRMDevice ได้โดยใช้ฟังก์ชันดังตารางที่ 6-10

```
HRESULT CreateDevice( LPGUID lpGUID,
                      LPDIRECTDRAWSURFACE lpDDS,
                      LPDIRECT3DDEVICE2 * lplpD3DDevice );
```

lpGUID	: pointer ที่ชี้ไปที่ globally unique identifier
lpDDS	: pointer ที่ชี้ไปที่ backbuffer
lplpD3DDevice	: address ของ pointer ที่ชี้ไปที่ device

ตารางที่ 6-9 ฟังก์ชัน CreateDevice ( )

ตัวอย่างการใช้งาน

```
LPDIRECT3DDEVICE2 d3ddevice;
pD3D->CreateDevice (NULL, backbuffer, &d3ddevice);
```

```
HRESULT CreateDeviceFromD3D( LPDIRECT3D2 lpD3D,
                             LPDIRECT3DDEVICE2 lpD3DDev,
                             LPDIRECT3DRMDEVICE3 * lplpD3DRMDevice );
```

lpD3D	: pointer ที่ชี้ไปยัง Direct3D
lpD3DDev	: pointer ที่ชี้ไปยัง Direct3DDevice
lplpD3DRMDevice	: address ของ pointer ที่ชี้ไปยัง Direct3DRMDevice

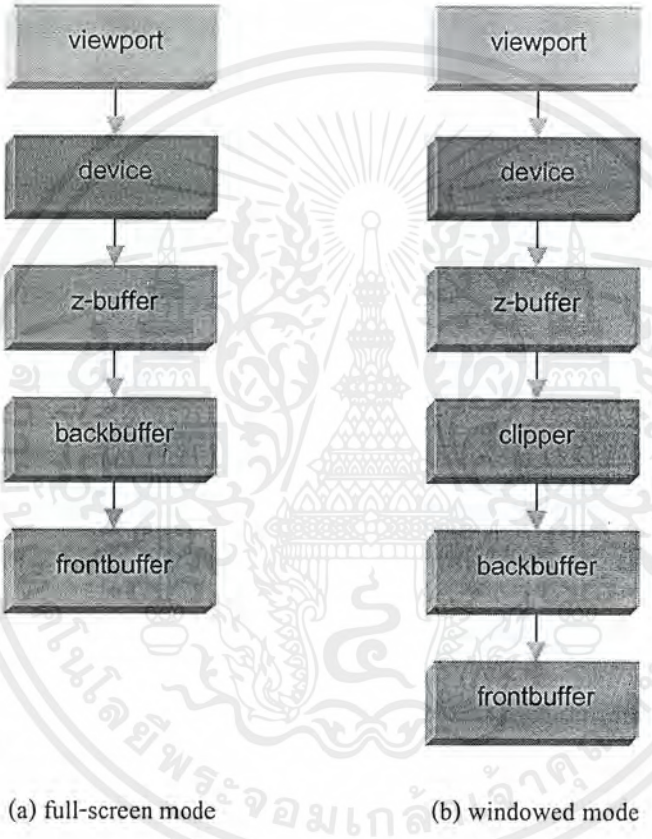
ตารางที่ 6-10 ฟังก์ชัน CreateDeviceFromD3D ( )

ตัวอย่างการใช้งาน

```
LPDIRECT3DRMDEVICE3 device;
pD3DRM->CreateDevice (pD3D, d3ddevice, &device);
```

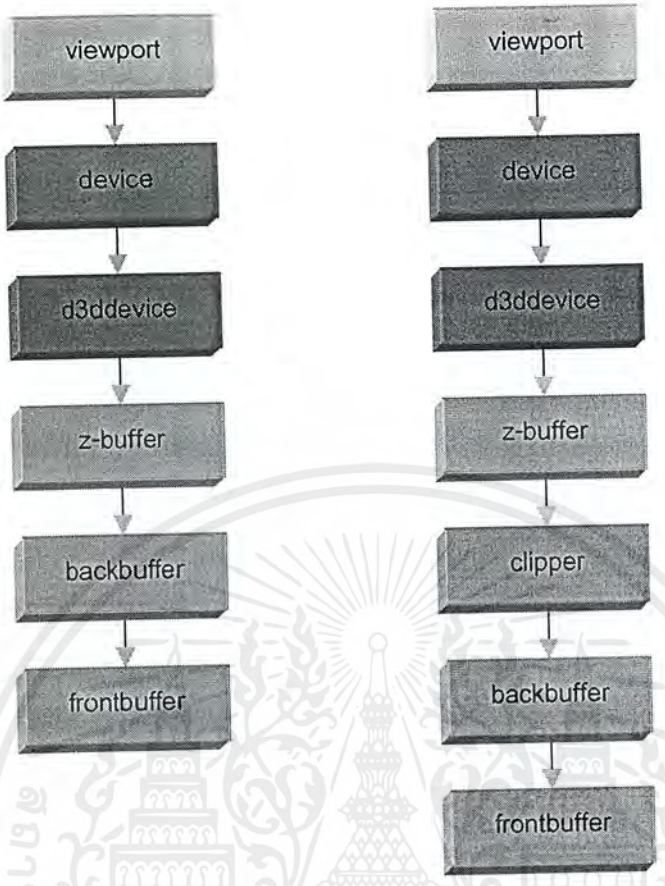
6.5.4 การเปลี่ยน Direct3D driver และ Display mode ขณะใช้งานแอปพลิเคชั่น

แอปพลิเคชั่นที่ใช้ Direct3D โดยทั่วไปนั้นสามารถจะให้ผู้ใช้เปลี่ยน Direct3D driver และ Display mode ได้ขณะใช้งานตามความต้องการของผู้ใช้ ซึ่งทั้งในโครงสร้างแบบที่ 2 และ แบบที่ 3 นั้นมีรูปแบบการทำงานตรงส่วนนี้คล้ายกัน ในโครงสร้างแบบที่ 2 ดังรูปที่ 6-26 นั้น เมื่อต้องการเปลี่ยน driver หรือ display mode เราต้องสร้างสร้าง frontbuffer, backbuffer, z-buffer และ viewport ใหม่ โดยก่อนจะสร้างนั้นเราต้อง **Release ( )** ทั้งหมดก่อนตามลำดับย้อนหลัง (สร้างทีหลังต้องถูก release ก่อน) ดังรูปที่ 6-29 จากนั้นจึงสร้างใหม่ตามขั้นตอนที่ได้ทำมาในการสร้างครั้งแรกโดยใช้ driver ของ Direct3D ที่ผู้ใช้เลือก เพื่อการสร้าง device



รูปที่ 6-29 ขั้นตอนการ Release ของโครงสร้างแอปพลิเคชั่นแบบที่ 2

ส่วนโครงสร้างของแอปพลิเคชั่นแบบที่ 3 นั้น เราต้องทำการ Release ( ) ทั้ง frontbuffer, backbuffer, z-buffer, d3ddevice, device, viewport ในลำดับย้อนหลังเช่นกัน ดังรูปที่ 6-30



(a) full-screen mode (b) windowed mode

รูปที่ 6-30 ขั้นตอนการ Release ของโครงสร้างแอปพลิเคชั่นแบบที่ 3

ส่วนใน โครงสร้างแอปพลิเคชั่นแบบที่ 2 และ 3 ใน windowed mode เมื่อมี WM_SIZE เข้ามานั้นเราจะมีการพิจารณาดังนี้คือ ถ้า window ของแอปพลิเคชั่นของเรามีขนาดใหญ่ขึ้นเมื่อเทียบกับขนาดของ device เราต้องทำการ Release ดังขั้นตอนข้างต้นแล้วทำการสร้างส่วนประกอบต่างๆใหม่

ในกรณีที่ WM_SIZE เข้าแล้วบอกให้เราทราบว่า window ของแอปพลิเคชั่นของเรามีขนาดเล็กลง (ไม่ต่างจากเดิมมากนัก) เราแค่ทำการ Release และสร้าง viewport ใหม่เท่านั้น แต่ในกรณีที่ window ของแอปพลิเคชั่นของเรามีขนาดเล็กลงกว่าเดิมมากเช่นลดลงจากเดิมครึ่งหนึ่งของขนาดเดิม เราควรทำการ Release และสร้างส่วนต่างๆใหม่ตามขั้นตอนก่อนหน้านี้

บทที่ 7
ความสัมพันธ์ระหว่าง DirectX, Graphic Card และ CPU

7.1 CPU กับการ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ

ตามปกติแล้วสำหรับเกมทั่วๆไปมีงานหลายๆส่วนที่จำเป็นต้องใช้ CPU ในการประมวลผล เช่น การจัดการเกี่ยวกับกราฟฟิก การจัดการเกี่ยวกับเสียง การจัดการเกี่ยวกับอินพุตไม่ว่าจะเป็นจากคีย์บอร์ด, เมาส์, หรือ จอยสติ๊ก และการประมวลผลภายในส่วนของเกมเอง ซึ่งงานที่ต้องใช้ CPU ในการประมวลผลมากที่สุดก็คืองานด้านกราฟฟิก ยิ่งในเกมยุคหลังๆนั้นเกมต่างๆก็ใช้ระบบกราฟิกแบบ 3 มิติด้วยแล้ว ทำให้ CPU ต้องรับภาระการประมวลผลเกี่ยวกับกราฟฟิกมากขึ้น ทางในการแก้ไขปัญหามีอยู่ 2 วิธีคือ

1. ลดคุณภาพของกราฟฟิกลงหรือลดประสิทธิภาพโดยรวมของเกมในส่วนอื่นๆลง เพื่อให้ CPU สามารถประมวลผลได้ทัน
2. หาอุปกรณ์มาช่วยแบ่งเบาภาระบางส่วนของ CPU โดยที่งานส่วนมากนั้นจะเกี่ยวกับกราฟฟิก ด้วยเหตุนี้จึงเป็นที่มาของการ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ

นอกเหนือจากการช่วยแบ่งเบาภาระของ CPU แล้ว การ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติจะมีความสามารถพิเศษที่เพิ่มเติมเข้ามา เช่นการเพิ่มหมอกหรือควันเพื่อทำให้เกิดบรรยากาศที่สมจริง

7.2 การทำงานของการ์ดเร่งความเร็วแบบ 3 มิติ

ในปัจจุบันเกมส่วนใหญ่ได้หันมาใช้ระบบกราฟฟิก 3 มิติกันเป็นส่วนใหญ่ ซึ่งในระบบกราฟฟิก 3 มิตินั้น วัตถุต่างๆภายในเกมไม่ว่าจะเป็นส่วนของฉากหรือตัวละครจะอยู่ในรูปแบบที่เรียกว่า *polygon* ดังรูปที่ 7-1 โดยที่ polygon นั้นเกิดจากการประกอบตัวของรูปทรงสี่เหลี่ยมที่มีด้านต่างๆไม่เท่ากัน ซึ่งรูปทรงสี่เหลี่ยมแต่ละรูปจะมีการลากเส้นในแนวทะแยงมุมจากมุมบนลงมามุมล่าง จะเกิดหน้าตัดเป็นรูปสามเหลี่ยมเล็กๆที่เรียกว่า face ซึ่ง face แต่ละชิ้นนี้เองจะถูกนำไปใช้คำนวณในการสร้างเทคนิคต่างๆ เช่นการสร้างแสงเงา ด้วยเหตุนี้การประมวลผลทั้งหมดของการ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติก็จะทำกับรูปทรงสามเหลี่ยม (Triangle) ที่เรียกว่า face ยิ่งวัตถุมีรูปทรงซับซ้อนมากเท่าไรก็ยิ่งมี face มากขึ้นเท่านั้น และต้องใช้เวลาในการประมวลผลที่มากขึ้นตามจำนวนของ face ที่ประกอบขึ้นเป็นวัตถุ

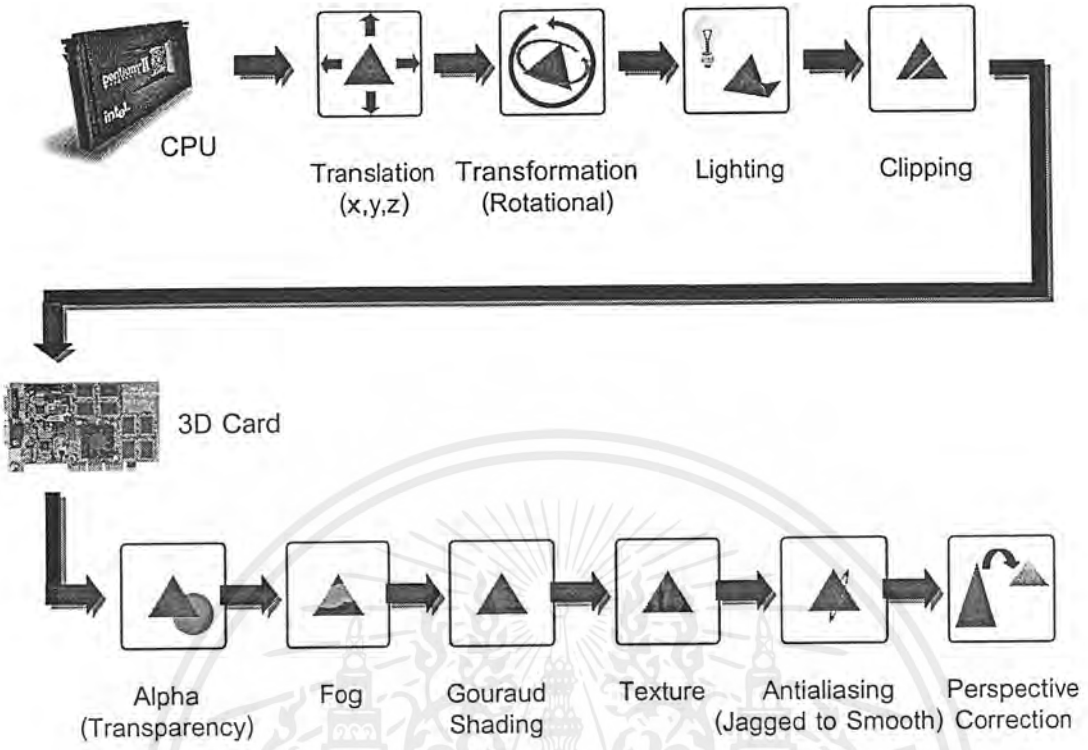
ถึงแม้ว่าการ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติจะเข้ามาเพื่อแบ่งเบาภาระของ CPU โดยตรงก็ตาม แต่ในขั้นตอนการเตรียมข้อมูลเพื่อส่งต่อให้การ์ดเร่งความเร็วแบบ 3 มิตินั้นยังต้องอาศัยการประมวลผลจาก CPU อยู่

รูปที่ 7-1 วัตถุในระบบกราฟฟิก 3 มิตินั้นอยู่ในรูปแบบของ polygon

เราสามารถแบ่งขั้นตอนการประมวลผลกราฟฟิกแบบ 3 มิติออกได้เป็น 2 ส่วนได้แก่

1. **การเตรียมการ** ขั้นตอนนี้เป็นหน้าที่ของ CPU ในการคำนวณตำแหน่งของวัตถุเพื่อให้ได้ตำแหน่งที่แน่นอนของ face แต่ละชิ้น กำหนดทิศทางของแสงและแหล่งกำเนิดแสงรวมทั้งสีของแสง ตำแหน่งของเงาที่จะเกิดขึ้น และส่วนสุดท้ายคือตัดรูป polygon บางส่วนของวัตถุที่จะไม่ได้ถูกแสดงบนจอภาพเนื่องจากถูกบดบังหรืออยู่นอกของเขตที่ต้องการแสดงผล
2. **การแสดงผล (Render)** ในขั้นตอนนี้เองที่การ์ดเร่งความเร็วแบบ 3 มิติเข้ามามีบทบาทแทนที่ของ CPU การ์ดเร่งความเร็วแบบ 3 มิติจะนำเอาชิ้นส่วนของ face แต่ละชิ้นที่เตรียมการไว้แล้วมาผ่านการตกแต่งด้วยเทคนิคต่างๆเช่น Transparency, Fog, Gouraud Shading, Texture, Antialiasing, และ Perspective Correction

จากรูปที่ 7-2 เราจะเห็นได้ว่า CPU ยังคงต้องรับภาระในการประมวลผลบางส่วนอยู่ ดังนั้น ใน DirectX เวอร์ชัน 7 จึงมีการสนับสนุนความสามารถ *hardware transform and lighting* ซึ่งแอปพลิเคชั่นที่ถูกพัฒนาโดย DirectX เวอร์ชัน 7 ขึ้นไป ที่ทำงานอยู่บนเครื่องคอมพิวเตอร์ที่ติดตั้งการ์ดเร่งความเร็วแบบ 3 มิติที่สนับสนุน DirectX เวอร์ชัน 7 ขึ้นไป เช่น การ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติที่ใช้ chip GForce ของบริษัท nVidia จะสามารถใช้ความสามารถนี้ได้ และจะช่วยแบ่งเบาภาระการประมวลผลของ CPU ในขั้นตอนการเตรียมการได้



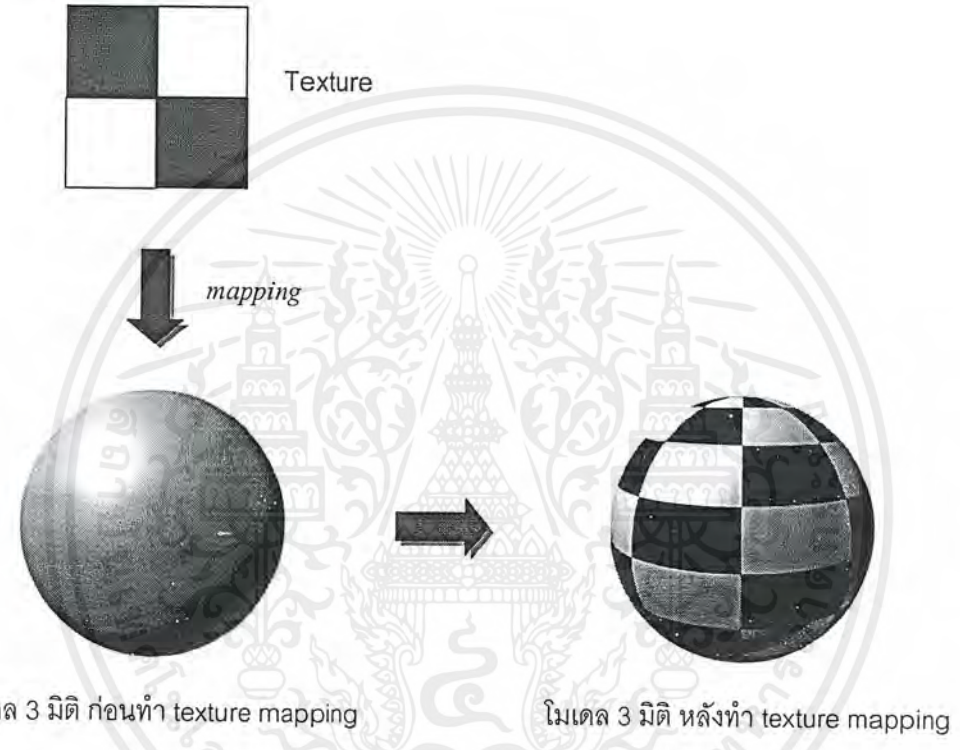
รูปที่ 7-2 3D Graphics Pipeline

7.3 3D Features

การทำงานของการ์ดเร่งความเร็วแบบ 3 มิตินั้นไม่ใช่เพียงช่วยลดภาระของ CPU เท่านั้น เทคนิคพิเศษหลายอย่างที่ไม่เคยปรากฏในโลกของเกมมาก่อนก็สามารถจะทำได้เพราะประสิทธิภาพของการ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ ซึ่งฟีเจอร์มาตรฐานที่ต้องมี หรือกำลังจะมีในอนาคตก็มีดังนี้

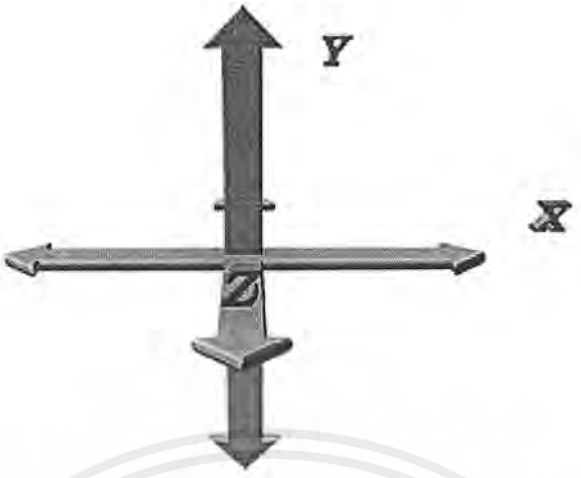
1. **Triangle Rate** ความเร็วในการสร้างและเขียนรูป polygon อัตราความเร็วนี้จะนับเป็น รูปต่อวินาที (polygons/sec)
2. **Fill Rate** อัตราความเร็วในการ render ภาพไปแสดงผลบนจอ อัตราความเร็วนี้จะนับเป็น จุดภาพต่อวินาที (pixels/sec)
3. **Hardware Triangle Setup** เป็นฟีเจอร์ของการ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติสมัยใหม่ ที่ใช้ในการแปลงระบบข้อมูลของ polygon จากเกมให้อยู่ในรูปแบบข้อมูลที่การ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ สามารถเข้าใจได้ทันที โดยกรรมวิธีทั้งหมดจะอยู่ภายในตัวการ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ ทำให้สามารถประมวลผลได้เร็วขึ้น

4. **Frame Buffers and Texture Buffers** สำหรับการ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ จะมีการแบ่งหน่วยความจำสำหรับใช้งานออกเป็น 2 ส่วนคือ Frame Buffer และ Texture Buffer ซึ่งโดยทั่วไปมีขนาดตั้งแต่ 2 MB ขึ้นไป สำหรับหน่วยความจำประเภท Frame Buffer นั้นจะเป็นหน่วยความจำที่ใช้สำหรับเก็บภาพที่จะถูกนำไปแสดงผลบนจอภาพ ซึ่งเป็นหน่วยความจำที่ display card ทุกชนิดต้องมี ส่วนหน่วยความจำประเภท Texture Buffer นั้นเป็นหน่วยความจำที่ใช้สำหรับเก็บรูปภาพที่จะใช้เป็น texture ให้กับวัตถุต่างๆภายในเกม



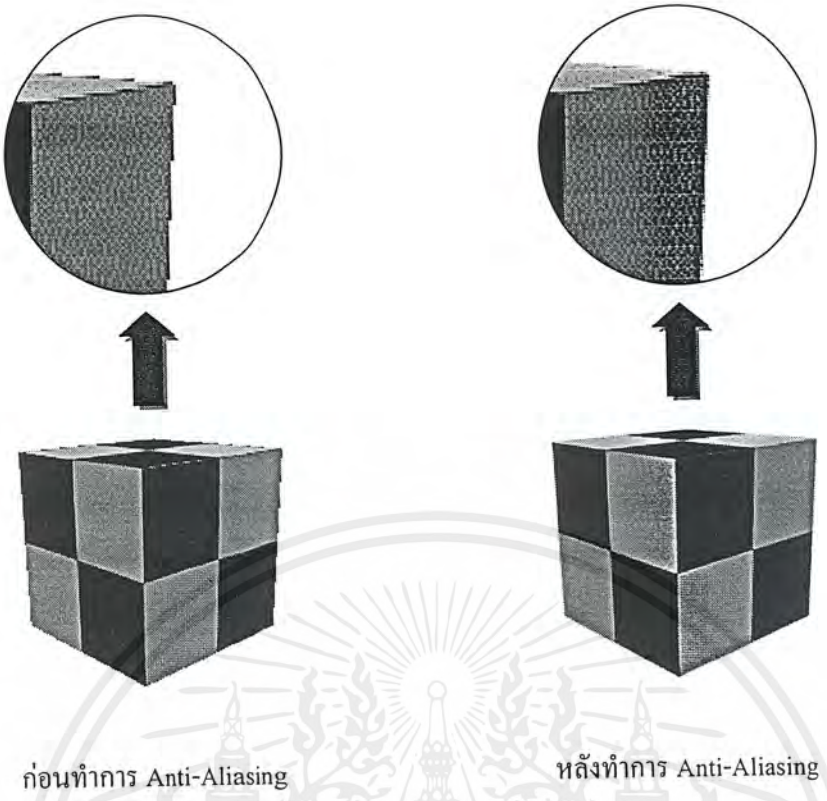
รูปที่ 7-3 การใช้ texture mapping

5. **Z-Buffer** ในระบบกราฟฟิกแบบ 3 มิตินั้นวัตถุต่างๆจะตั้งอยู่บนแกน 3 คือแกนตั้ง (Y Axis), แกนนอน (X Axis), และแกนลึก (Z Axis) ดังรูปที่ 7-4 โดยที่ Z-Buffer จะทำหน้าที่เก็บค่าแกน Z ของแต่ละจุดภาพ (Pixel) แล้วนำมาเปรียบเทียบกับจุดอื่นที่อยู่ในบริเวณเดียวกัน จากนั้นก็จะนำจุดเหล่านี้มาแสดงในระยะที่แตกต่างกันตามค่าที่เปรียบเทียบได้ ระยะที่แตกต่างกันนี้จะมากหรือน้อยก็ขึ้นอยู่กับขนาดของ Z-Buffer ซึ่งจะมีขนาดต่างๆกันไปเช่น 16 Bit, 24 Bit, และ 32 Bit

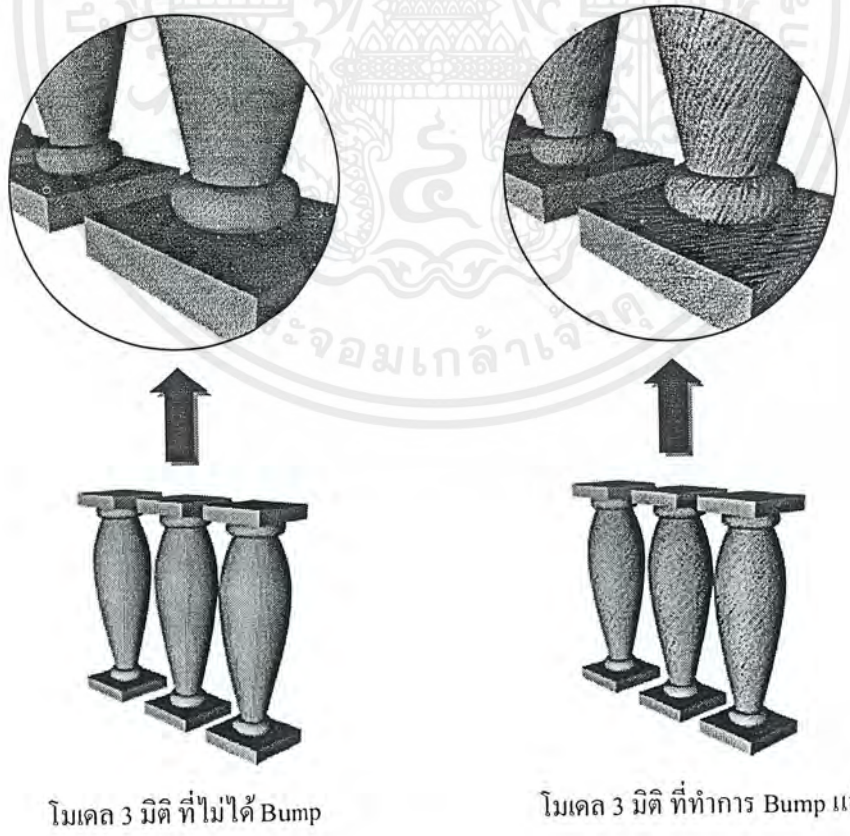


รูปที่ 7-4 Axis ในระบบกราฟฟิกแบบ 3 มิติ

6. **Single Pass Multitexturing** การประมวลผลทุก 1 รอบสัญญาณนาฬิกาสามารถสามารถทำงานกับ texture ได้มากกว่า 1 จุดภาพ
7. **True color Rendering** การแสดงผลที่มากกว่า 16 Bit ขึ้นไปตั้งแต่ 24 Bit ถึง 32 Bit
8. **Edge Anti-Aliasing** เทคนิค Anti-Aliasing เป็นเทคนิคที่ใช้กำจัดรอยหยักภายในฉาก โดยปกติภาพที่ปรากฏขึ้นบนจอภาพจะประกอบขึ้นจากการต่อกันของจุดจำนวนมาก รวมทั้ง face ของ polygon ด้วย เทคนิค Edge Anti-Aliasing จะใช้สำหรับขอบที่จะเกิดขึ้นกับ face ในฉากโดยเฉพาะ
9. **Full Screen Anti-Aliasing** เทคนิคนี้บางครั้งเรียกว่า Order Independent เป็นเทคนิคที่ใช้ลบรอยหยัก โดยมีการทำงานส่วนใหญ่ใน Frame Buffer ใช้วิธีการขยายภาพให้ใหญ่กว่าความละเอียดจริงๆของจอภาพ จากนั้นจึงทำการลบขอบแล้วค่อยลดขนาดของภาพลงมาสู่ความละเอียดปกติเพื่อให้ได้ภาพที่มีความเนียนมากขึ้น
10. **Trilinear MIP Mapping** ในระบบกราฟฟิกแบบ 3 มิตินั้นจะมีการกำหนดความละเอียดของภาพที่จะนำมาใช้เป็น texture ของวัตถุ ให้มีขนาดแตกต่างกันไป ซึ่งโดยปกติจะเป็น 2 ระดับ (Bilinear) แต่การ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ รุ่นใหม่ๆจะสนับสนุนความละเอียดของภาพที่ใช้เป็น 3 ระดับ (Trilinear MIP Mapping) ซึ่งความละเอียดของภาพที่จะนำไปใช้เป็น texture ให้กับวัตถุนั้นจะขึ้นอยู่กับระยะใกล้ไกลของวัตถุที่จะเห็นภายในฉาก
11. **Perspective Correction** นอกจากการให้ระยะใกล้ไกลของวัตถุที่จะเห็นในฉากแล้ว วัตถุแต่ละชิ้นที่อยู่ในฉากจะต้องมีการแสดงพื้นผิวให้ถูกต้องตามระยะใกล้ไกลด้วย ซึ่งเทคนิคดังกล่าวเรียกว่า Perspective Correction
12. **Bump Mapping** เป็นรูปแบบของ texture แบบหนึ่งซึ่งไม่ได้ใช้ในการแสดงรูปภาพและสีของวัตถุ แต่จะใช้ในการสร้างลักษณะของพื้นผิว เช่นความหยาบ, ความขรุขระ



รูปที่ 7-5 การทำ Anti-Aliasing

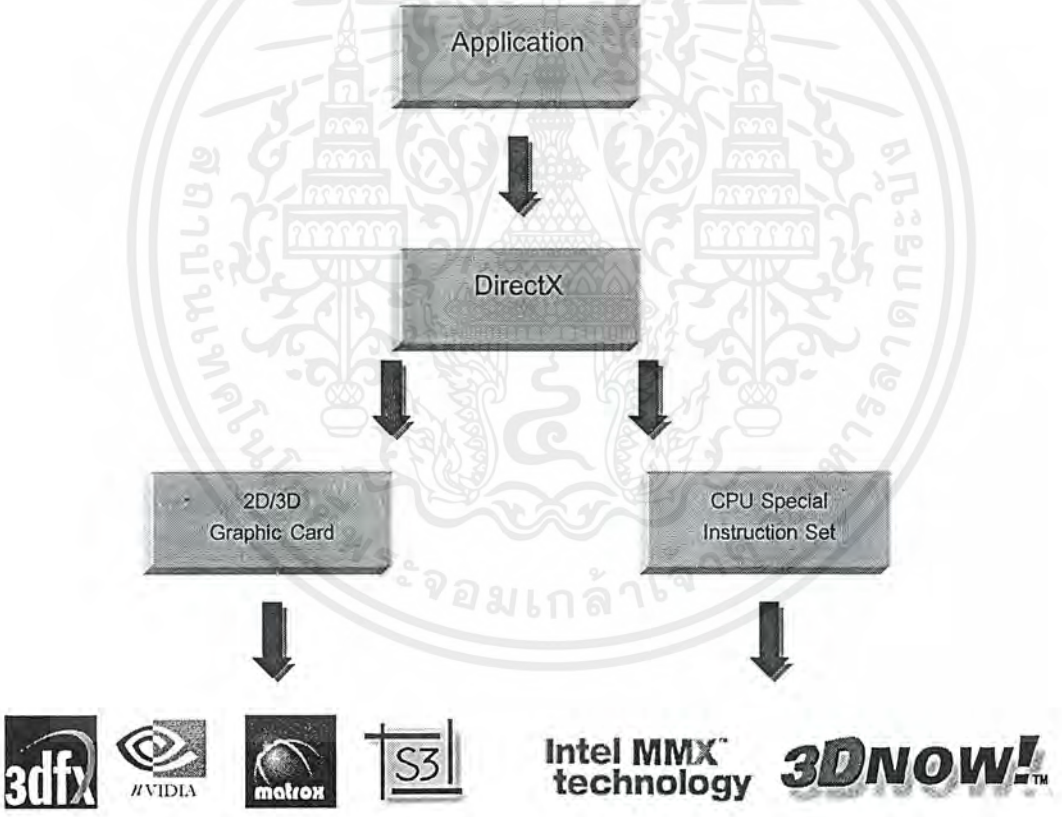


รูปที่ 7-6 Bump Mapping

13. **Texture Compression** เป็นเทคนิคในการบีบข้อมูลของรูปที่จะนำมาใช้เป็น texture ให้มีขนาดเล็กลง ทำให้สามารถใช้รูปภาพที่มีขนาดใหญ่มากขึ้นมาใช้เป็น texture ในขณะที่ใช้ Textures Buffers ที่มีขนาดเท่าเดิมหรือเล็กลงได้

7.4 DirectX, การ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ, และ ชุดคำสั่งพิเศษของ CPU

ขั้นตอนการประมวลผลกราฟฟิกนั้นถูกแบ่งออกเป็น 2 ส่วน คือขั้นตอนการเตรียมการ เป็นหน้าที่ของ CPU และขั้นตอนการแสดงผล ซึ่งเป็นหน้าที่ของการ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ แต่ใช่ว่าเกมต่างๆจะสามารถแบ่งแยกงานและส่งไปให้ CPU และ การ์ดเร่งความเร็ว 3 มิติได้ด้วยตนเอง การทำงานทั้งหมดนี้จะผ่านส่วนที่เรียกว่า API (Application Programming Interface) ทำหน้าที่เป็นตัวกลางในการเชื่อมความเข้าใจในการทำงานระหว่าง ซอฟต์แวร์เกม และฮาร์ดแวร์อีกทีหนึ่ง สำหรับ API ในระบบ 3 มิติ ที่เป็นมาตรฐานนิยมใช้กันอยู่ในปัจจุบันจะมีอยู่ 3 ชนิดด้วยกันได้แก่ OpenGL, Glide, Direct3D (เป็นส่วนหนึ่งของ DirectX)



รูปที่ 7-7 ความสัมพันธ์ระหว่าง DirectX, การ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ, และคำสั่งพิเศษของ CPU

7.5 DirectX กับ เทคโนโลยี MMX และ 3D Now!

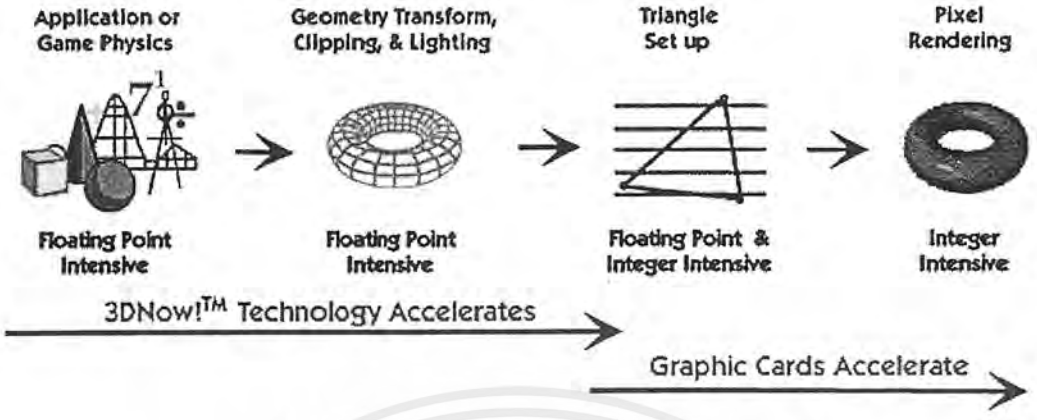
MMX และ 3D Now! เป็นชุดคำสั่งพิเศษของ CPU เพื่อเพิ่มประสิทธิภาพในการประมวลผลด้าน Multimedia และ 3D-Processing ซึ่ง Direct3D สนับสนุนชุดคำสั่งทั้ง 2 แบบนี้ MMX และ 3D Now! นั้นไม่ใช่ *hardware-accelerated devices* โดยที่จะมีบางขั้นตอนบางขั้นตอนในการประมวลผลกราฟฟิกแบบ 3 มิติ ที่ต้องทำโดยกระบวนการทางด้านซอฟต์แวร์ ซึ่งชุดคำสั่ง MMX และ 3D Now! นี้เองจะช่วยเพิ่มประสิทธิภาพในการประมวลผลให้กับ CPU ชุดคำสั่ง MMX ถูกสนับสนุนโดย DirectX version 3.0 ขึ้นไป ส่วนชุดคำสั่ง 3D Now! นั้นถูกสนับสนุนโดย DirectX version 6.0 ขึ้นไป

7.6 เทคโนโลยี 3D Now!

3DNow เป็นเทคโนโลยีที่ถูกปรับปรุงขึ้นบนชุดคำสั่งของไมโครโพรเซสเซอร์ x86 เพื่อให้ผู้ใช้คอมพิวเตอร์พีซีได้รับประโยชน์มากขึ้น และเพิ่มส่วนที่มีความชำนาญในการประมวลผลภาพ 3 มิติขึ้น และข้อดีของ 3DNow คือ เป็นซอฟต์แวร์ขนาดเล็กที่สุดที่สามารถใช้งานได้ และได้รับการ support จาก DirectX API (Application programming interface) ซึ่งเป็นซอฟต์แวร์ที่ทำงานภายใต้ CPU ตระกูล x86 อยู่แล้ว

เนื่องจากในปัจจุบันความต้องการที่จะสร้างแอปพลิเคชั่น 3 มิติให้กับเครื่องพีซีนั้น ต้องมีความเหมือนจริงเพิ่มมากขึ้นในอีกระดับหนึ่ง นอกจากนี้ตัวเร่งความเร็วในงานกราฟิก 3 มิติมีการพัฒนาและขยายตัวอย่างรวดเร็วมากขึ้นเรื่อยๆ และความสามารถของโพรเซสเซอร์ยังตามไม่ทัน จึงทำให้เกิด bottleneck ในการทำงานกราฟิกมากขึ้นตาม ส่งผลให้เครื่องพีซีทุกเครื่องจะมีการทำงานกราฟิก 3 มิติเป็นลักษณะมาตราฐาน เพื่อให้สามารถทำงานในระดับสูงขึ้น และสาเหตุที่ทำให้เกิด bottleneck คือ ความสามารถในการทำ floating point ของโพรเซสเซอร์ที่เป็นอยู่ในปัจจุบัน และเทคโนโลยี 3DNow จะแก้ bottleneck ซึ่งเกิดจากกระบวนการ pipeline ในการสร้างงานกราฟิก โดยเพิ่มความสามารถในการทำ floating-point ให้กับ CPU, เร่งความเร็วให้กับงานมัลติมีเดีย และงานกราฟิก 3 มิติ

3DNow ปรับปรุงความสามารถของโพรเซสเซอร์ในการคำนวณเพื่อทำ floating point และ 3Dnow ขจัดช่องว่างที่จะเกิดมากขึ้นเรื่อยๆ ระหว่างการทำงานของโพรเซสเซอร์และตัวเร่งความเร็วในงานกราฟิก ทำให้ปัญหา bottleneck หมดไป และเพื่อให้เห็นภาพมากขึ้น เราต้องเข้าใจว่ากระบวนการ pipeline ในงานกราฟิกเกิดขึ้นอย่างไร ซึ่งจากรูปที่ 7-8 จะแสดงทั้ง 4 สถานะของกระบวนการ pipeline ในงานด้านกราฟิก



รูปที่ 7-8 สถานะของกระบวนการ pipeline ในงานกราฟฟิก

Physics : CPU จะทำการคำนวณลักษณะทางกายภาพของเหตุการณ์จริงและวัตถุต่างๆ ที่อยู่ภายใน ด้วยวิธี floating-point-intensive

Geometry : จากนั้น CPU จะเปลี่ยนวัตถุจากรูปทรงทางคณิตศาสตร์ (รูปทรงเรขาคณิต) ไปเป็นรูปทรง 3 มิติ โดยใช้วิธี floating-point-intensive 3D geometry

Setup : CPU เริ่มกระบวนการสร้างภาพที่เห็นสัดส่วนได้ตามความเป็นจริง ซึ่งเป็นสิ่งที่สำคัญมากในการมองเห็นภาพได้ทั้ง 3 มิติ และทำให้ภาพที่ได้สมบูรณ์แบบ โดยอาศัยตัวเร่งความเร็วสำหรับกราฟิก

Rendering : ตัวเร่งความเร็วสำหรับงานกราฟิกทำการกำหนดความละเอียดให้กับวัตถุที่จะสร้างให้มีความเหมือนจริง โดยใช้การคำนวณตำแหน่ง, ปริมาณสีและเงาในแต่ละ pixel

สำหรับช่วงแรกกระบวนการ pipeline ของภาพ 3 มิติจะเป็นการรวมรูปทรงเรขาคณิตที่สร้างโดยวิธี floating-point-intensive ซึ่งการสร้างโปรแกรมสำเร็จรูปที่เป็นโมเดลแบบ 3 มิติหรือโมเดล เกม หมายรวมถึงการสร้างโมเดลที่ได้จากคำนวณทางคณิตศาสตร์ของวัตถุ จากรูปร่างทางกายภาพ และหลังจากนั้นจะเพิ่มรูปที่ต้องการสร้างให้กับ CPU สำหรับใช้ในการ floating-point ถ้าปริมาณรูปที่จะสร้างมีมาก จะส่งผลทำให้ frame rate ช้าลง และภาพที่ได้มีประสิทธิภาพต่ำ

เทคโนโลยี 3DNow เป็นวิวัฒนาการของ AMD ที่ก้าวหน้ามาก มีประสิทธิภาพในการขจัดปัญหา bottleneck ที่เกิดจากการทำ floating-point โดยจะกำหนดให้ทำ floating-point หนึ่งชุดคำสั่งในหนึ่งรอบสัญญาณนาฬิกาของ CPU (AMD-K6-2 และ AMD-K6-III) เท่านั้น แต่สามารถทำหลายๆ ชุดคำสั่งได้ในเวลาเดียวกัน ซึ่งเรียกเทคนิคเช่นนี้ว่า SIMD (Single Instruction Multiple Data) ซึ่งแตกต่างจากก่อนหน้าที่จะมีเทคโนโลยี 3DNow กระบวนการ pipeline ของงานกราฟิกจะขึ้นกับ bandwidth ของ CPU ในการทำ floating-point เพื่อแสดงรูปร่างทางกายภาพและรูปทรงเรขาคณิตเท่านั้น

7.6.1 รายละเอียดทางเทคนิค

instruction set

- มี 21 คำสั่ง
- support การทำ floating-point และ integer intensive ในลักษณะ SIMD
- โดยเฉพาะคำสั่ง SIMD integer จะเพิ่มประสิทธิภาพในการทำ MPEG decoding
- มีคำสั่ง PREFETCH เพิ่มขึ้นมา
- มีคำสั่ง FEMMS (Fast Entry/Exit Multimedia State) เพื่อลดเวลาที่ใช้ในการ switch ไปมาระหว่างโหมดคำสั่ง MMX และ x87
- support มาตรฐาน IEEE 754

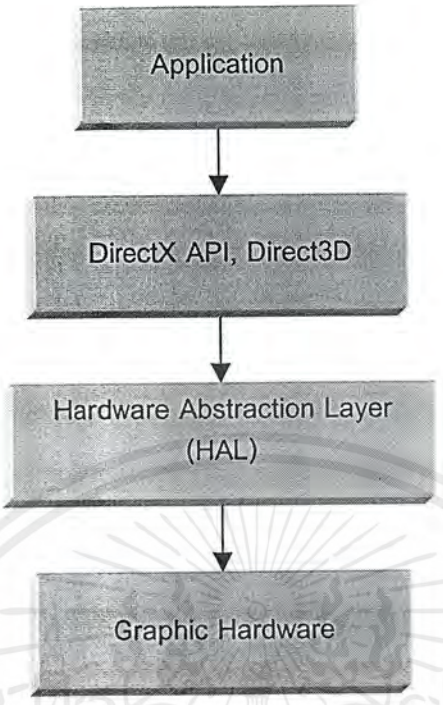
Processor Microarchitecture

- สามารถทำ pipelined execution แบบ dual ได้อย่างสมบูรณ์
- เก็บจำนวน floating-point ใน memory ได้โดยไม่จำกัด
- สามารถ execute ได้มากถึง 2 คำสั่ง (3Dnow) ใน 1 clock
- สามารถทำการคำนวณ floating-point ได้ทั้ง บวก ลบ คูณ หาร ได้ภายใน 1 clock (มีperformance ได้สูงสุดถึง 1.8 Gigaflops ที่ 450 MHz)
- เป็น floating-point stack ที่ใช้ทำ task switching ระหว่าง 3Dnow และ MMX

7.6.2 DirectX 6.0 กับ 3D Now!

DirectX 6.x เป็น API รุ่นล่าสุดของ Microsoft ซึ่งสร้างความมั่นใจและความน่าเชื่อถือให้กับผู้พัฒนามากที่สุดใน API ทุกรุ่น โดยลักษณะเด่นที่เพิ่มขึ้นใหม่จาก DirectX 5.0 ประกอบด้วยความสามารถทางมัลติมีเดียในด้าน leading-edge และส่วนที่เพิ่มขึ้นรวมถึงลักษณะ performance ที่เร็วขึ้น และลักษณะเด่นที่เพิ่มขึ้นใน Direct3D API ตัวอย่างเช่น single-pass multitexturing, bump mapping, vertex buffers, stencil planes, and texture compression

เนื่องจาก DirectX 6.x ประกอบด้วย Direct3D ทำให้เป็นประโยชน์อย่างมากกับเทคโนโลยี 3Dnow! เพราะทำให้แอปพลิเคชั่นมีประสิทธิภาพในงานกราฟฟิก 3 มิติ เพิ่มขึ้น คือ มีฟังก์ชันในการสร้างรูปทรงเรขาคณิตแบบ floating-point-intensive ภายใน Direct3D และ 3DNow มีบทบาทสำคัญที่ทำให้ performance ในงาน 3 มิติเร็วขึ้น โดยทำงานร่วมกับ Direct3D เพื่อเพิ่มความเร็วให้กับฟังก์ชันการทำ pipeline สำหรับงานกราฟิก ในการดัดแปลงรูป, การเพิ่มแสง และการตัดต่อรูป และรูปที่ 7-9 แสดงการใช้ DirectX API



รูปที่ 7-9 การทำงานระหว่าง DirectX กับ CPU

7.7 การ์ดเร่งความเร็วกราฟฟิกแบบ 3 มิติ ที่น่าสนใจในท้องตลาด (กลางปี 2542)

7.7.1 Vooodoo3 3000

คุณสมบัติ

- 16MB of SDRAM
- 7 million triangles/sec.
- 333MegaTexel/sec. Fill rate
- 166MHz Core Clock speed
- 100 billion operations/sec.
- สนับสนุนโหมดความละเอียดถึง 2048×1536
- Full 128-bit 20 accelerator
- 350 MHz RAMDAC
- DVD Hardware Assist
- สนับสนุน DirectX, Glide และ OpenGL
- Alpha-Blending
- Single Pass, Single Cycle Bump Mapping
- Single Pass, Single Cycle Trilinear MIP-Mapping
- Patented Multi-Texturing Programmable Fog Tables

- Sub-pixel และ Sub-Texel Correction
- Sub-pixel และ Sub-Texel Correction ถึง 0.4×0.4 resolution
- Per-pixel atmospheric fog ด้วย programmable fog zones
- Floating point Z buffer(W buffer)
- True per-pixel, LOD MIP mapping ด้วย biasing and champing
- Highly accurate LDC calculations
- สนับสนุนรูปแบบ Texture map 14รูปแบบ
- 8-bit palletized textures with full bilinear filtering
- Texture compression ผ่าน narrow-channel YAB format
- สนับสนุน multi-triangle strips และ fans
- Gouraud Shading

เมื่อดูจากรายละเอียดของคุณสมบัติ Voodoo3 3000 ก็คือความเร็วที่เพิ่มขึ้นเหนือกว่าการ์ดแสดงผลรุ่นก่อนอย่าง Banshee หรือ Voodoo2 นอกจากนี้ยังมีจุดเด่นอีกอย่างที่ความเร็วของ RAMDAC ซึ่งสูงถึง 350MHz ช่วยความเร็วในการทำงานแบบ 2D ให้มากยิ่งขึ้น นอกจากนี้ Voodoo3 3000 ยังสนับสนุนอัตรารีเฟรชสูงได้ทุกความละเอียด

แต่จุดอ่อนของ Voodoo3 3000 ก็จะอยู่ที่การไม่สนับสนุน render แบบ 32 บิต ในการทำงานกับเกม 3D ต่างๆ ทำให้ภาพที่ได้ไม่สวยงามเท่ากับการ์ดที่สนับสนุน render แบบ 32 บิต

สรุป

ราคาของ Voodoo3 3000 นั้นอยู่ที่ 8,900 บาท ซึ่งถือเป็นการ์ดที่แพงที่สุดซึ่งเป็นทางเลือกสำหรับผู้ที่ต้องการความเร็วสูงในการเล่นเกม อย่างไรก็ตามปัญหาของ Voodoo3 3000 ก็จะอยู่ที่การไม่สนับสนุนโหมด 32 บิต

หากต้องการการ์ดที่เล่นได้กับเกมทุกเกม Voodoo3 3000 เป็นตัวเลือกที่ดีที่สุดเพราะสนับสนุน API ครบทั้ง Glide, Direct3D และ OpenGL โดยมีประสิทธิภาพที่ดีทั้งสามแบบ

นอกจากนั้นผู้ที่มีเครื่องที่มีความเร็วไม่สูงนัก อาจจะสนใจ Voodoo3 30000 มากกว่าการ์ดรุ่นอื่นๆ เพราะหากเครื่องมีความเร็วไม่สูงนัก การเล่นเกมในโหมด 32 บิตนั้นภาพจะกระตุกมากทำให้ต้องลดโหมดลงมาที่ 16 บิตซึ่งเมื่อเป็นเช่นนี้ ก็จะไม่มีปัญหากับ Voodoo3 3000

7.7.2 3D Blaster Riva TNT2

คุณสมบัติ

- ความเร็ว 300 million bilinear filtered fill rate, multi-textured pixels ต่อวินาที
- 9 million triangles/sec.
- แบนด์วิธหน่วยความจำ 2.9GB/sec.

- 100% hardware triangle setup engine
- TwiN-Texel(TNT) dual 32-bit 3D rendering pipeline
- 2 texture mapped, lit pixels ต่อรอบสัญญาณนาฬิกา
- สนับสนุน Single pass multitexturing(DirectX6.X และ OpenGL ICD)

3D Features

- Hardware triangle setup engine
- 2 texture-mapped, lit pixels ต่อรอบสัญญาณนาฬิกา
- สนับสนุน Single pass multi-texturing สนับสนุนคุณสมบัติ เช่น bump mapping, reflection mapping, texture modulation และ procedural textures
- สนับสนุน Per-pixel perspective correct texture mapping
- สนับสนุน Full scene, order independent anti-aliasing

2D Features

- เร่งความเร็วด้วยฮาร์ดแวร์สำหรับ Windows GDI
- สนับสนุน 32-bit VGA/SVGA
- สนับสนุน multi-buffering (สูงถึง quad buffering) สำหรับการเล่นกับวิดีโอ และอนิเมชั่นแบบต่อเนื่อง

Video Support

- สนับสนุน DirectDraw
- สนับสนุน color space conversion(YUV 4:2:2 และ 4:2:0) ด้วยฮาร์ดแวร์
- สนับสนุน multi-tap X และ Y buffering

Card Specification

- สนับสนุน AGP 2X with sideband
- สนับสนุน Analog VGA
- สนับสนุน TV Output
- compatible with VESA DDC2B

จุดเด่นของ 3D Blaster Riva TNT2 Ultra ก็คือมีหน่วยความจำบนตัวการ์ดถึง 32MB พร้อม RAMDAC ความเร็วสูงขนาด 300MHz การสนับสนุนการ render ในโหมด 32 บิต และการสนับสนุน TV-Out ทั้งแบบ S-Video และ Composite และที่สำคัญที่สุดคือเป็นการ์ด TNT2 รุ่นเดียวที่รันที่ 128MHz ทำให้ 3D Blaster Riva TNT2 Ultra เป็นการ์ดที่น่าสนใจมากสำหรับการ์ดที่ใช้ชิพ TNT2

สรุป

ระดับราคาของ 3D Blaster Riva TNT2 Ultra จะมีระดับราคาประมาณ 7000 บาทซึ่งถูกกว่าการ์ดหลายรุ่นที่ใช้การ์ด TNT ธรรมดา

ประสิทธิภาพของ 3D Blaster Riva TNT2 Ultra อยู่ในระดับแนวหน้าพร้อมยังมีไดรเวอร์(Unified Driver) ที่สมบูรณ์ที่สุดในบรรดาการ์ด TNT2 ทั้งหมด เพราะ Unified Driver ที่มีมาให้นั้นสนับสนุน Glide ได้

7.7.3 WinFast 3D S230V

คุณสมบัติ

- ใช้ชิพเซต Nvidia Riva Vanta
- หน่วยความจำ 8MB
- 250MHz RaMDAC
- Fast 32-bit VGA/SVGA
- High Performance 128-bit 2D/GDI/DirectDraw Acceleration
- 64-bit wide frame buffer interface สนับสนุนถึง 8MB SDRAM
- Video Acceleration สำหรับ DirectShow, MPEG-1/2 และ Indeo
- ปรับแต่งสำหรับ Direct3D พร้อมสนับสนุน DirectX 5.0 และ 6.0
- 128-bit wide graphics engine และ frame buffer
- TwiN-Texel Architecture
- สนับสนุน AGP 4X/2X interface(AGP 2.0 และ AGP 1.0)
- สนับสนุน 32-bit RGB rendering พร้อม Destination Alpha
- สนับสนุน 24-bit Z-buffer, 8-bit stencil buffer
- สนับสนุน Anisotropic filtering(ดีกว่า Tri-Linear MIP-mapping)
- สนับสนุน Triangle setup engine ผ่านฮาร์ดแวร์
- สนับสนุน Bus Mastering DMA 2X 66MHz AGP interface พร้อม sideband support
- สนับสนุน Windows 9X, Windows 2000 และ Windows NT 4.0 เต็มรูปแบบ
- สนับสนุน DirectX 6.0 และ OpenGL
- Planar YUV12(4:2:0) to/from packed(4:2:2) conversion สำหรับ MPEG Acceleration ด้วยซอฟต์แวร์และแอพพลิเคชั่นวิดีโอคอนเฟอร์เรนซ์ H.621
- สนับสนุน VESA DDC2B+, DPMS, VBE 2.0/3.0

สรุป

ชิพที่ใช้เป็นชิพ ตัวใหม่ที่ชื่อว่า Nvidia Riva Vanta (ซึ่งก็ คือ ชิพ Riva TNT ในเวอร์ชันถูกนั่นเอง) WinFast S320V นั้นจะมีหน่วยความจำขนาด 8MB โดยมีคุณสมบัติที่น่าสนใจคือการสนับสนุนสถาปัตยกรรม TwiN-Texel ซึ่งเป็นสถาปัตยกรรมเดียวกับที่ใช้ใน Riva TNT

WinFast S230V สนับสนุนการเร่งความเร็วสามมิติได้ตั้งแต่ 640×480 จนถึง 1024×768 ในแบบ 16 บิต และ 640×480 และ 800×600 ในโหมด 32 บิต ซึ่งแม้จะด้อยกว่าการ์ดรุ่นอื่นๆ แต่เมื่อพิจารณาที่ราคาเพียง 2900 บาท จะพบว่าโหมดต่างๆที่ให้การสนับสนุนนั้นคุ้มกับราคา

WinFast S320V มี RAMDAC ความเร็ว 250MHz ช่วยให้สามารถปรับอัตรารีเฟรชในแบบ Flicker free ได้ในทุกโหมดความละเอียด

แม้จะมีความเร็วต่ำกว่าการ์ดอื่นๆในท้องตลาด แต่ WinFast S230V ยังสนับสนุนการแสดงผลในโหมด 32 บิตได้ ซึ่งขนาด Voodoo3 3000 ยังไม่มีความสามารถนี้เลย

แม้ว่าจะมีคุณภาพที่ต่ำกว่าการ์ด 3D อื่นๆ อย่าง Vodoo3 หรือ TNT2 แต่เมื่อใช้เล่นเกมบนจอมอนิเตอร์ขนาด 14-15 นิ้ว ก็จะมีความสามารถใกล้เคียงกัน WinFast S320V จึงเป็นอีกทางเลือกเมื่อเทียบราคากับการ์ดรุ่นอื่นๆ

บทที่ 8

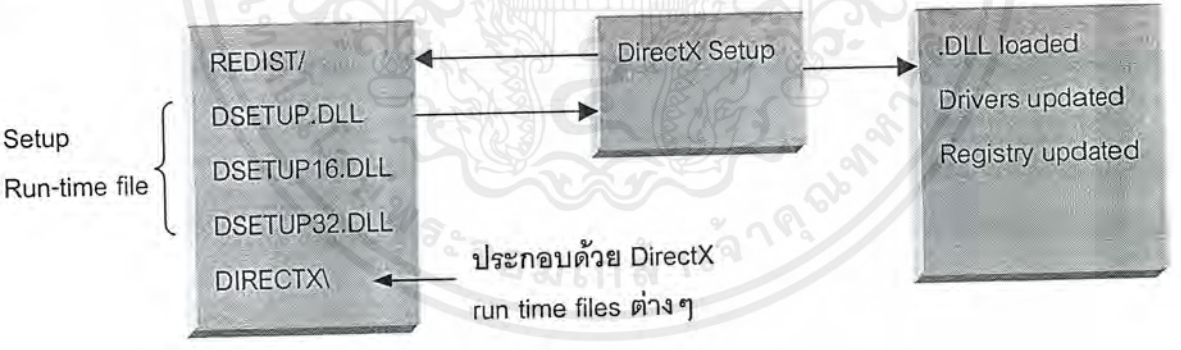
DirectSetup

แม้ว่า DirectX จะพัฒนาอย่างแพร่หลาย แต่ก็มีความเป็นไปได้ว่าเครื่องคอมพิวเตอร์ส่วนบุคคลบางเครื่องจะไม่ได้ติดตั้ง DirectX ไว้ สำหรับแอปพลิเคชั่นที่รองรับ DirectX จะต้องมี DirectX run-time อยู่ เช่นเดียวกับ driver ของ DirectX โดย ไมโครซอฟต์จะให้ DirectX ไว้ใช้ และ DirectSetup จะเป็นตัวติดตั้ง DirectX ให้โดยอัตโนมัติ

ไมโครซอฟต์ได้จัดหา DirectX run-time ให้เรียกว่า *redistributable* ทำให้เราสามารถ ติดตั้ง DirectX ด้วยโปรแกรมของเราเองได้ *redistributable* นี้ จะอยู่ใน SDK ใน directory /REDIST

ใน *redistributable* จะรองรับภาษาต่างๆ 15 ภาษา เช่น เยอรมัน ญี่ปุ่น โดยมันจะมี message และ dialog box ของภาษานั้นเอง ซึ่ง DirectSetup จะหาภาษาที่ตรงกับระบบปฏิบัติการนั้นๆ ยกเว้นภาษาอังกฤษ ที่สามารถติดตั้งได้ในทุกภาษา

DirectSetupไม่สามารถใช้ DirectX COM objects ใดๆ ได้ เพราะมันไม่ได้ถูกติดตั้งไว้ในเครื่องของผู้ใช้ (เวลาเราใช้งาน DirectSetup นั่นหมายความว่าเราต้องการติดตั้ง DirectX ลงไปจึงเสมือนว่าเราไม่มี DirectX อยู่ในเครื่องของเรา) ดังนั้นจึงต้องมี *.DLLs ไว้เพื่อให้ทำงานได้ถูกต้อง ได้แก่ DSETUP.DLL, DSETUP16.DLL, DSETUP32.DLL และในการโปรแกรมต้องรวมไฟล์ DSETUP.H และ DSETUP.LIB ไว้ด้วย



รูปที่ 8-1 สิ่งที่ DirectX ต้องการขณะทำงาน

8.1 การใช้ DirectXSetup

ระบบ DirectX ทั้งหมดสามารถถูกติดตั้งได้โดย คำสั่งเพียงคำสั่งเดียว นั่นคือ DirectXSetup() โดยมีรูปแบบดังนี้

```
int WINAPI DirectXSetup(HWND hWnd,          //handle ของ window
                        LPSTR lpszRootPath, //ตำแหน่ง directory ที่ REDIST ตั้งอยู่
                        DWORD dFlags);      //flag ที่ควบคุม
```

ฟังก์ชันนี้ จะมีการส่งค่ากลับที่แสดงถึงสิ่งที่เกิดขึ้นขณะติดตั้ง ส่วน lpszRootPath คือที่ตั้งที่เก็บ DirectX run-time files อยู่ เช่นถ้ามันอยู่ใน CD เราก็เรียก "CDROM:\\REDIST" แล้วถ้าเราติดตั้ง DirectX ลงใน Directory ที่ชื่อว่า Check\ จะเป็นดังนี้

Check\
REDIST/
DSETUP.DLL
DSETUP16.DLL
DSETUP32.DLL
DIRECTX\

ส่วน Flag จะมีดังนี้

DSETUP_DIRECTX: ติดตั้ง DirectX run-time ให้เหมาะสมกับ drivers
DSETUP_DDRAWDRV: ติดตั้ง Driver แสดงผลด้านภาพที่แสดงโดยไมโครซอฟต์
DSETUP_DSOUNDDRV: ติดตั้ง Driver แสดงผลด้านเสียงที่แสดงโดยไมโครซอฟต์
DSETUP_DXCORE: ติดตั้ง DirectX run-time แต่ไม่ได้ติดตั้ง DirectX run-time ให้เหมาะสมตาม drivers
DSETUP_TESTINSTALL: แสดงการทดสอบการติดตั้ง แต่ไม่ได้ติดตั้งจริง

8.2 การดูผลการทำงานขณะติดตั้ง

ขณะที่เราติดตั้ง DirectX อยู่เราสามารถ ดู สถานะต่างๆ ที่เกิดขึ้นได้ เพื่อใช้ในการตัดสินใจว่าจะเลือกให้ทำงานอย่างไรโดยใช้ callback function

ฟังก์ชันที่ใช้ในการเรียก callback function นี้เรียกว่า DirectXSetupSetCallback() โดยมีรูปแบบ ดังนี้

```
int WINAPI DirectXSetupSetCallback(DSETUP_CALLBACK Callback);
//Callback      : pointer ไปที่ callback function
```

ส่วน callback function จะมีรูปแบบดังนี้

```
DWORD DirectXSetupCallbackFunction(
        DWORD Reason,       //เหตุผลที่ callback
        DWORD MsgType,      //เหมือนกับพารามิเตอร์ของ MessageBox
        char *szMessage,    //message string
        char *szname,       //ขึ้นกับเหตุผล
        void *pInfo);       //upgrade ข้อมูล
```

8.3 ตัวอย่างการสร้าง callback function

```
DWORD WINAPI DSetupCallback(DWORD Reason,
                DWORD MsgType,
                char *szMessage,
                char *szName,
                void *pUpgradeInfo)
{
// this is the most generic callback function you can have
// it simply returns IDOK for status calls, otherwise
// it pops up a messagebox and allows you to make the
// decision

if (MsgType==0)
   return IDOK;

// call the messagebox function and return its value
// remember, DirectSetup is designed to respond to the
// return values of MessageBox

return(MessageBox(main_window_handle, szMessage,
          "DirectX Setup Demo -- Running", MsgType));

} // end DsetupCallback
```

8.4 การเรียกใช้ทำได้โดยเรียก DirectXSetupSetCallback(DSetupCallback)

เมื่อเราทำการเรียก DirectXSetup( ) จากนั้น DsetupCallback( ) จะถูกเรียกเมื่อไรก็ตามที่มี message หรือ เกิดปัญหาขึ้น ขณะที่กำลังติดตั้ง DirectX

นอจากนั้นเราสามารถ ตรวจดูได้ว่า DirectX เป็น version ใดได้โดย DirectXSetupGetVersion() โดยมันจะรับหมายเลข version และ version ใหม่ของ DirectX โดยแปลงเป็นเลข 32-bit มันมี prototype ดังนี้

```
INT WINAPI DirecXSetupGetVersion(
    DWORD *pdwVersion     //รับ version ในรูปแบบ 16.16
    DWORD *pdwRevision); //รับ version ใหม่
```

8.5 AutoPlay

AutoPlay เป็นสิ่งที่สร้างขึ้นใน Windows 95 เพื่อให้สามารถสร้างการทำงาน โดยอัตโนมัติขณะที่เล่นแผ่น CD-ROM โดยการทำงานของ AutoPlay ก็จะอ่านไฟล์ที่ชื่อ AUTORUN.INF จาก directory หลักของ CD ถ้ามี ไฟล์นี้ อยู่ มันก็จะโหลด และแปลงเป็น AutoPlay ไฟล์

ใน AUTORUN.INF จะมีคำสั่งต่างๆ มากมาย แต่คำสั่งที่จำเป็นคือ "open" และ "icon" โดยคำสั่ง open จะเป็นการเปิดไฟล์นั้นโดยอัตโนมัติ ส่วนมากจะเป็นไฟล์ที่ใช้ในการติดตั้ง ส่วนคำสั่ง icon คือตัวที่แสดงรูป icon ที่เรา open

ตัวอย่างการใช้งาน

ถ้าไฟล์ติดตั้งชื่อ Setup.exe และไฟล์รูป icon ชื่อ smile.ico ใน AUTORUN.INF จะเป็นดังนี้

```
[autorun]
open=Setup.exe
icon=smile.ico
```

จากนั้นเราก็คัดลอกไฟล์นี้ ลงไปใน CD โปรแกรมของเรา

บทที่ 9
การสร้างโมเดล 3 มิติ

9.1 โมเดล 3 มิติ

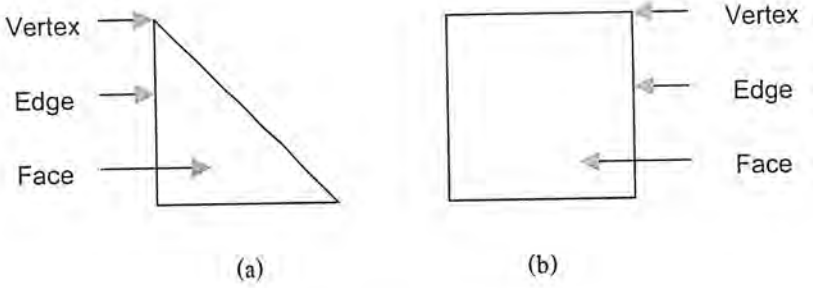
โมเดล 3 มิติ ถูกใช้ในระบบคอมพิวเตอร์กราฟฟิกแบบ 3 มิติ เพื่ออธิบายรูปทรงของวัตถุต่างๆนั้น โดยทั่วไปจะอยู่ในรูปแบบของ "mesh" ดังรูปที่ 9-1 ซึ่ง mesh นั้นจะประกอบด้วย "faces" ซึ่ง face นี้จะถูกอธิบายโดยใช้ "polygon" ซึ่งเป็นรูปทรงหลายเหลี่ยม ดังรูปที่ 9-2

รูปที่ 9-1 โมเดล 3 มิติในรูปแบบของ mesh

รูปที่ 9-2 Polygons ในรูปแบบต่างๆ

Polygon เป็นรูปทรงหลายเหลี่ยมปิดซึ่งสามารถจะมีด้านได้ไม่จำกัด โดยที่ polygon ที่พื้นฐานที่สุดนั้นคือ polygon ที่เป็นรูปสามเหลี่ยม ดังรูปที่ 9-2 (a) ซึ่งถูกเรียกว่า "Triangle" ส่วนในรูปที่ 9-2 (b) เรียกว่า "Quad"

ถ้าเรามองให้ลึกลงไปถึงส่วนประกอบของ polygon แล้วเราสามารถแยกส่วนประกอบต่างๆได้ดังรูปที่ 9-3 คือ "vertices", "edges", และ "faces" โดยที่ **vertex** นั้นคือจุดที่เส้นตรงมาเชื่อมต่อกัน **edge** คือเส้นตรงที่เราสร้างขึ้นเพื่อกำหนดขอบเขตของ polygon ส่วน **face** นั้นคือพื้นที่อยู่ภายใน polygon ที่ถูกล้อมรอบด้วย edges ดังรูปที่ 9-3



รูปที่ 9-3 ส่วนประกอบของ faces

โดยทั่วไปแอปพลิเคชั่นที่ทำงานเกี่ยวกับกราฟฟิกแบบ 3 มิติ นั้น จะใช้โมเดล 3 มิติ ซึ่งเกิดจากการรวมกันของหลายๆ polygons ในรูปแบบที่เราต้องการ ซึ่งวัตถุที่มีรูปแบบไม่ซับซ้อนนั้นสามารถจะถูกอธิบายได้โดยใช้ polygons จำนวนไม่มากเช่น รูปกล่องสี่เหลี่ยม แต่วัตถุที่มีความซับซ้อนนั้นจำเป็นต้องใช้ polygons จำนวนมากเพื่อใช้การอธิบายรูปทรงของวัตถุนั้น

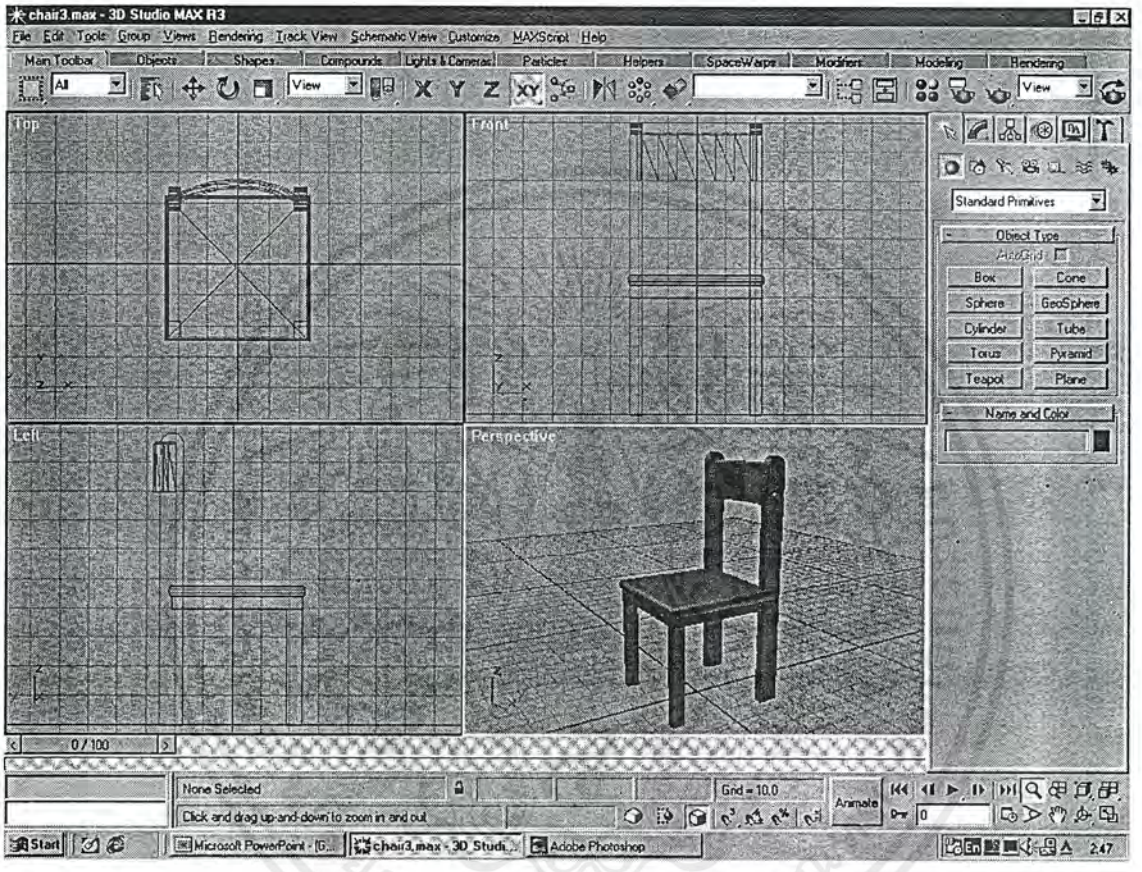
รูปที่ 9-4 (a) รูปกล่องสี่เหลี่ยม นั้นประกอบด้วย 8 vertices, 12 edges, และ 6 faces โดย faces นั้นถูกอธิบายโดยใช้ polygons ในรูปแบบ quad แต่ถ้า faces นั้นถูกอธิบายด้วย polygons ในรูปแบบ triangle แล้วจะมี 8 vertices, 24 edges, 12 faces ดังรูปที่ 9-4 (b)

รูปที่ 9-4 การใช้ polygons ในรูปแบบที่ต่างกันเพื่ออธิบาย faces

9.2 โปรแกรมที่ใช้ในการสร้างโมเดล 3 มิติ

การสร้างวัตถุ 3 มิติเพื่อที่จะนำไปใช้ในแอปพลิเคชั่นที่ใช้ Direct3D ซึ่งรวมทั้ง Direct3D Immediate Mode และ Direct3D Retained-Mode นั้นในความเป็นจริงแล้วเราสามารถจะใช้โปรแกรมทางด้านกราฟฟิก 3 มิติ ตัวใดก็ได้ เพราะว่าโดยส่วนประกอบที่ย่อยที่สุดของโมเดล 3 มิติ ที่ใช้ใน Direct3D นั้น จะประกอบด้วย vertices และ triangles ซึ่งเราสามารถจะสร้าง triangles ได้จากความสัมพันธ์ของ vertices เมื่อเป็นดังนี้ ถ้าเรารู้รูปแบบ (format) ของไฟล์ที่ได้จากโปรแกรมทางด้านกราฟฟิก 3 มิติ ที่เราใช้เพื่อสร้างโมเดล 3 มิติ แล้วเราก็

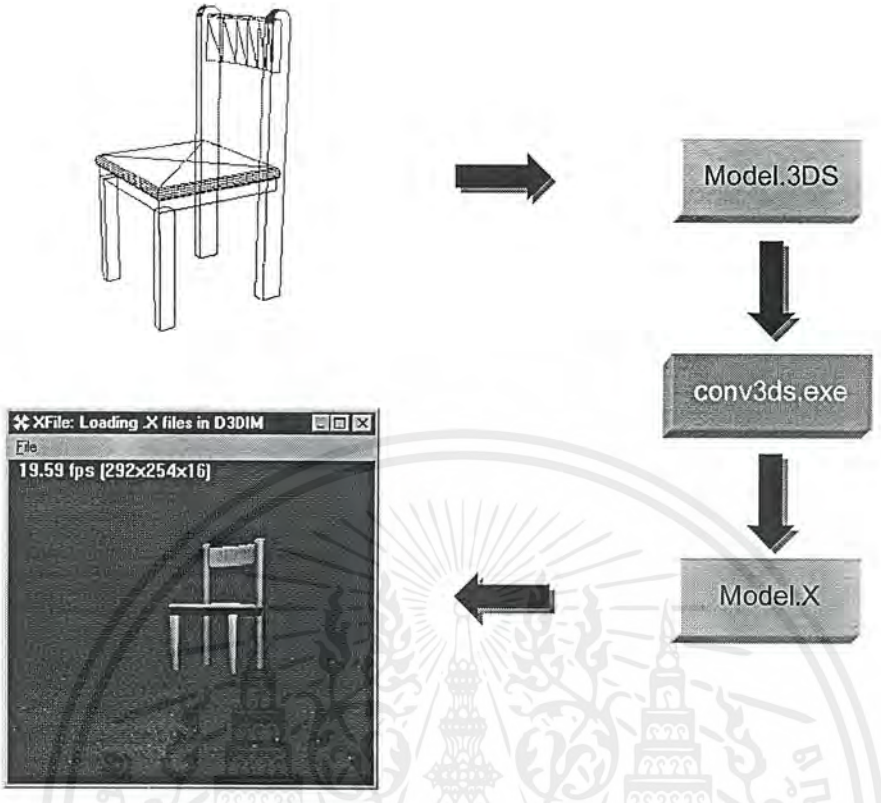
สามารถที่จะนำไฟล์นั้นมาใช้ในแอปพลิเคชั่นของเราได้ โดยเราต้องสร้างฟังก์ชันเพื่อทำการ load ไฟล์เพื่อมาใช้งานในแอปพลิเคชั่นของเราขึ้นมาเอง โปรแกรมกราฟฟิกทางด้าน 3 มิติ ที่นิยมใช้กันอย่างแพร่หลายนั้น ได้แก่ trueSpace, 3D Studio MAX, LightWave, SoftImage ฯลฯ ซึ่งในโครงงานนี้จะใช้ โปรแกรม 3D Studio Max เพราะเป็นโปรแกรมที่หามาใช้ได้ง่าย และมีหนังสำหรับเรียนรู้การใช้งานอยู่มากมายตามท้องตลาด ดังรูปที่ 9-5 แสดง interface ของโปรแกรม 3D Studio MAX 3.0



รูปที่ 9-5 Interface ของโปรแกรม 3D Studio MAX 3.0

9.3 โปรแกรม conv3ds

ใน Direct3D Retained-Mode นั้นเราไม่จำเป็นต้องสร้างฟังก์ชันเพื่อทำการ load ไฟล์ของโมเดล 3 มิติขึ้นมาเองเพราะเราสามารถใช้ MeshBuilder เพื่อ load ไฟล์ของโมเดล 3 มิติ ได้ซึ่งช่วยประหยัดเวลาในการพัฒนาแอปพลิเคชั่นได้มาก โดยไฟล์ของโมเดล 3 มิติ ที่นำมาใช้ได้นั้นเป็นไฟล์ในรูปแบบ ***.3DS** ซึ่งสร้างได้จากโปรแกรม 3D Studio MAX หรือ โปรแกรมอื่นซึ่งสามารถบันทึกเป็นไฟล์ในรูปแบบนี้ได้ แต่ MeshBuilder นั้นไม่สามารถ load ไฟล์ในรูปแบบ *.3DS ได้โดยตรง โดยที่ DirectX SDK นั้นมีโปรแกรมชื่อ **conv3ds** เพื่อช่วยแปลงไฟล์ *.3DS ให้อยู่ในรูปแบบ ***.X** ซึ่ง MeshBuilder นั้นสามารถ load ได้ ซึ่งมีขั้นตอนดังรูปที่ 9-6



รูปที่ 9-6 ขั้นตอนการนำโมเดล 3 มิติ มาใช้งาน

ไฟล์ *.X นั้น นอกจากจะมีข้อมูลของโมเดล 3 มิติ แล้วยังสามารถเก็บข้อมูลเกี่ยวกับแอนิเมชันได้อีกด้วย ซึ่งรูปแบบคำสั่งที่ใช้ในการแปลงไฟล์ *.3DS เป็น ไฟล์ในรูปแบบ *.X มีดังนี้

ตัวอย่างการใช้งาน

```
conv3ds –m Model.3DS        // วัตถุทุกๆชิ้นในไฟล์ *.3DS จะถูกรวมเป็น mesh เดียวกัน
conv3ds –A Model.3DS        // นำข้อมูลเกี่ยวกับแอนิเมชันที่มีอยู่ในไฟล์ *.3DS บักทึกไว้ในไฟล์ *.X
```

ที่จริงแล้วไฟล์ในรูปแบบ *.3DS นั้นเป็นไฟล์ของโปรแกรม 3D Studio Release 4 ซึ่งเป็นเวอร์ชันก่อนหน้าโปรแกรม 3D Studio MAX ส่วนไฟล์ของโปรแกรม 3D Studio MAX นั้นจะอยู่ในรูปแบบ *.MAX ซึ่งสาเหตุที่โปรแกรม conv3ds นั้นใช้ไฟล์ในรูปแบบ *.3DS ซึ่งเก่ากว่าไฟล์ในรูปแบบ *.MAX เพราะไฟล์ในรูปแบบ *.3DS นั้นการบันทึกเกี่ยวกับวัตถุ 3 มิติ นั้นบันทึกในรูปแบบของ mesh ล้วนๆ ซึ่งไฟล์ในรูปแบบ *.MAX นั้นไม่ได้ทำการบันทึกวัตถุ 3 มิติ ในรูปแบบของ mesh เพียงอย่างเดียว เพราะใน 3D Studio MAX มีรูปแบบการสร้างโมเดลแบบ parametric (parametric modeling) ซึ่งทำให้การสร้างและเปลี่ยนแปลงโมเดลมีความยืดหยุ่นมากขึ้น ดังนั้นจึงทำให้ไฟล์ในรูปแบบ *.MAX มีความซับซ้อนกว่าไฟล์ในรูปแบบ *.3DS

เราสามารถบันทึกไฟล์ในรูปแบบ *.3DS โดยใช้โปรแกรม 3D Studio MAX ได้โดยเลือกคำสั่ง "Export" จาก เมนู "File"

9.4 รายละเอียดของโมเดล 3 มิติ

รายละเอียดของวัตถุ 3 มิตินั้นหมายถึงจำนวน triangles ทั้งหมดในวัตถุนั้น ซึ่งโมเดลที่นำมาใช้ใน Direct3D นั้นควรมีรายละเอียดน้อยที่สุดเท่าที่จะเป็นไปได้เพราะถ้ามีรายละเอียดมากนั่นหมายถึงเวลาในการประมวลผลที่มากขึ้นนั่นเอง

(a) (b)

รูปที่ 9-7 โมเดล 3 มิติ ที่เหมือนกันแต่มีความละเอียดต่างกัน

จากรูปที่ 9-7 จะเห็นว่าโมเดล 3 มิติทั้ง (a) และ (b) นั้นให้ผลลัพธ์ออกมาที่เหมือนกันคือเป็นรูปกล่องสี่เหลี่ยม แต่ในรูปที่ 9-7 (a) นั้นมีจำนวน faces มากกว่าในรูปที่ 9-7 (b) ซึ่งเป็น faces ที่ไม่เกิดประโยชน์เพราะระนาบสี่เหลี่ยมพื้นผ้านั้นแม้จะมีจำนวน faces ต่างกันแต่ผลลัพธ์สุดท้ายจะออกมาเหมือนกัน

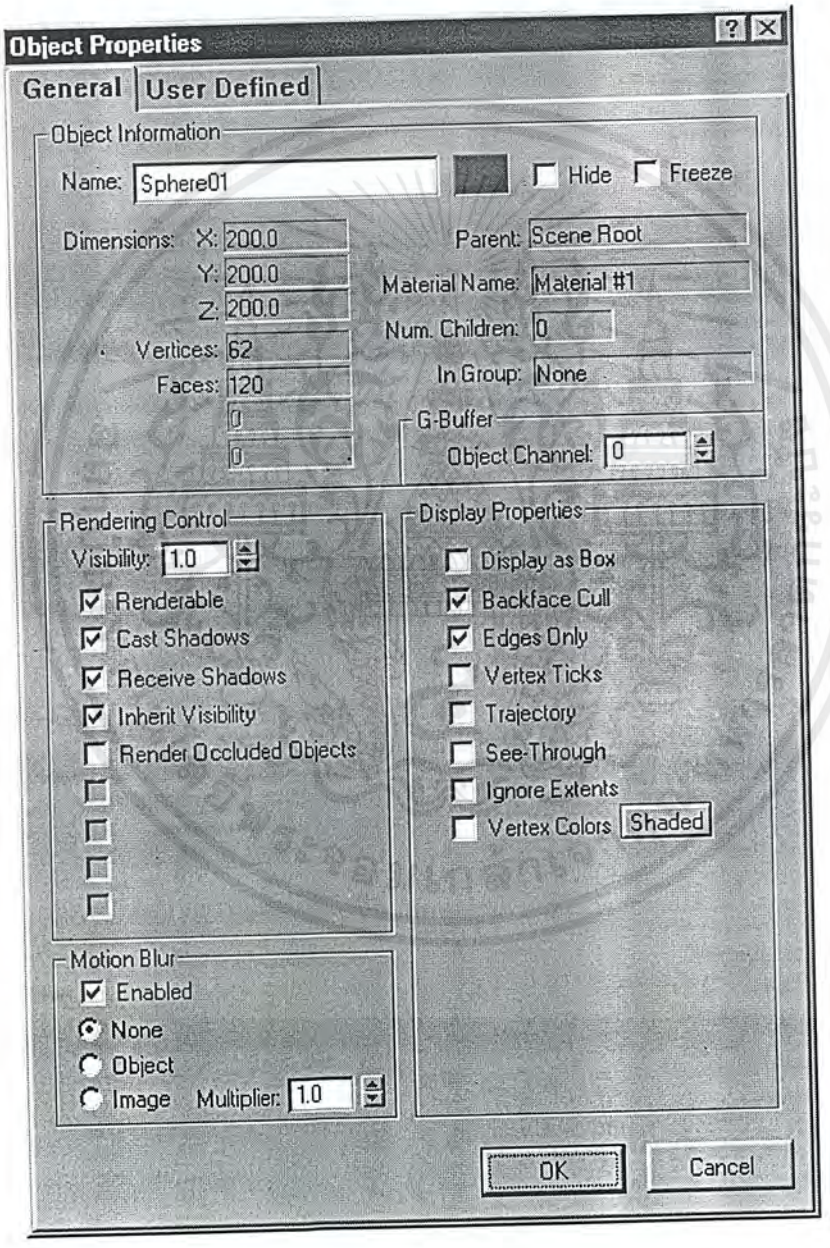
(a) (b)

รูปที่ 9-8 โมเดล 3 มิติ ที่มีคุณภาพต่างกันเพราะมีความละเอียดต่างกัน

รูปที่ 9-8 นั้นแสดงรูปทรงกลมที่มีจำนวน faces ต่างกัน ในกรณีนี้จำนวน faces จะมีผลโดยตรงต่อคุณภาพของโมเดล 3 มิติ ดังนั้นเราต้องพิจารณาว่าโมเดล 3 มิติ ที่จะนำไปใช้ในแอปพลิเคชั่นนั้นควรมีความละเอียดในระดับใดเพื่อให้ได้โมเดล 3 มิติ ที่มีคุณภาพในระดับที่น่าพอใจและทำให้แอปพลิเคชั่นของเราทำงาน

ได้ในระดับความเร็วที่เราต้องการ ซึ่งเราอาจกำหนดให้โมเดล 3 มิติ ที่ไม่เป็นจุดสนใจมีความละเอียดต่ำ ส่วนโมเดล 3 มิติที่เป็นจุดสนใจนั้นเรากำหนดให้มีความละเอียดสูงกว่าเพื่อที่ว่าจำนวน faces ทั้งหมดจะได้ไม่มากเกินไป

ในโปรแกรม 3D Studio MAX เราสามารถจะตรวจสอบจำนวน triangles ของวัตถุ 3 มิติ ได้โดยการคลิกเมาส์ปุ่มขวามือบริเวณวัตถุที่เราสนใจ จะปรากฏ popup menu ให้เราเลือกคำสั่ง Properties... จะปรากฏ Object Properties dialog ดังรูปที่ 9-9



รูปที่ 9-9 Object Properties dialog

จากรูปที่ 9-9 ในส่วนของ Object Information หัวข้อ "Faces:" แสดงถึงจำนวน triangles ทั้งหมดของวัตถุ ถ้าเราต้องการรู้จำนวน faces ทั้งหมดของ scene เราสามารถทราบได้โดยเลือกคำสั่ง "Summary Info..." จากเมนู File จะปรากฏ Summary Info dialog และดูได้จากส่วน Mesh Totals

บทที่ 10
แนวความคิดและการออกแบบ

10.1 แนวความคิดเบื้องต้น

ในโครงงานนี้จะใช้โครงสร้างแอปพลิเคชั่นของ Direct3D Retained-Mode ในรูปแบบที่ 3 คือการสร้าง device นั้นสร้างโดยใช้ฟังก์ชัน CreateDeviceFromD3D ( ) ซึ่งจะกำหนดให้แอปพลิเคชั่นทำงานใน full-screen mode เท่านั้น จากบทที่ 6 ซึ่งได้อธิบายเกี่ยวกับโครงสร้างของแอปพลิเคชั่นที่ใช้ Direct3D Retained-Mode นั้นเราจะเห็นว่าก่อนที่เราจะจัดการเกี่ยวกับเรื่องของ แสง, วัตถุ 3 มิติ, หรือ แอนิเมชัน ซึ่งเป็นส่วนที่อยู่ในรูปแบบ "application independent" โดยส่วนนี้จะขึ้นอยู่กับแต่ละแอปพลิเคชั่นว่าจะกำหนดให้เป็นอย่างไร เราต้องจัดการกับส่วน "application dependent" ก่อน ซึ่งเป็นส่วนที่ทุกๆแอปพลิเคชั่นในรูปแบบนี้ต้องมีเหมือนกันคือ ส่วนของ dd, frontbuffer, backbuffer, z-buffer, d3d, d3ddevice, d3drm, scene, camera, device, และ viewport ทางคณะผู้จัดทำจึงมีแนวความคิดที่จะนำส่วนประกอบเหล่านี้มารวมไว้ด้วยกันในรูปแบบของคลาส (class) ซึ่งเป็นรูปแบบการเขียนโปรแกรมแบบเชิงวัตถุหรือ OOP (Object-Oriented Programming) โดยจะมี method ของคลาสเพื่อที่จะทำการสร้างโครงสร้างในส่วนของ application dependent ทั้งหมด

10.2 คลาส DX_Screen

เป็นคลาสที่พัฒนาขึ้นมาเป็นคลาสแรก โดยจะรวมส่วนประกอบในส่วน application dependent ไว้ด้วยกัน โดยมีรายละเอียดดังนี้

```
#define MAX_D3DDRIVERS      5       // จำนวน drivers ของ Direct3D ที่รองรับ
#define MAX_DDDRIVERS       5       // จำนวน drivers ของ DirectDraw ที่รองรับ
#define MAX_MODES           40      // จำนวน display modes ที่รองรับ
#define MAX_MODEWIDTH       1024    // width ของ display mode ที่สูงที่สุด
#define MAX_MODEHEIGHT      768     // height ของ display mode ที่สูงที่สุด
#define MIN_BPP             16      // bit per pixel ต่ำสุด
#define DEFAULT_WIDTH       640     // width ของ display mode มาตรฐาน
#define DEFAULT_HEIGHT      480     // height ของ display mode มาตรฐาน
#define DEFAULT_BPP         16      // bit per pixel ของ display mode มาตรฐาน
#define VALUE_ERROR         -10     // ค่าข้อมูลที่แสดงความผิดพลาด
#define VALUE_YOUDECIDE     -20     // ค่าข้อมูลพิเศษ
```

ตารางที่ 10-1 ค่าคงที่ของคลาส DX_Screens

จากตารางที่ 10-1 แสดงค่าคงที่ที่ใช้กำหนดคุณสมบัติของคลาส DX_Screen เราสามารถปรับเปลี่ยนค่าเหล่านี้ได้เพื่อให้ DX_Screen มีคุณสมบัติตามที่เราต้องการ

```
typedef struct tagDDDRIVER
{
        char            Name[30];       // ชื่อของ driver
        DDCAPS          HWCaps;         // ความสามารถทางด้านฮาร์ดแวร์
        GUID            Guid            // ค่า globally unique identifier ที่ชี้ไปที่ driver
        BOOL            bIsPrimary;     // driver นี้เป็น primary driver หรือไม่
} DDDRIVER, * LPDDDRIVER;
```

ตารางที่ 10-2 โครงสร้างข้อมูล DDDRIVER

จากตารางที่ 10-2 แสดงโครงสร้างข้อมูลของ DDDRIVER ซึ่งจะเก็บข้อมูล driver ของ DirectDraw ที่คลาส DX_Screen สามารถตรวจสอบได้

```
typedef struct tagD3DDRIVER
{
        char                    Name[30];       // ชื่อของ driver
        char                    About[50];      // รายละเอียดของ driver
        D3DDEVICEDESC           Desc;           // ความสามารถของ driver
        GUID                    Guid;           // ค่า globally unique identifier ที่ชี้ไปที่
driver
        BOOL                    bIsHardware;    // เป็น hardware driver หรือไม่
        BOOL                    bDoesTextures;  // จัดการเกี่ยวกับ textures หรือไม่
        BOOL                    bDoesZBuffer;   // จัดการเกี่ยวกับ z-buffer หรือไม่
        BOOL                    bCanDoWindow;   // ทำงานใน windowed mode ที่ bpp
ปัจจุบันได้หรือไม่
} D3DDRIVER, * LPD3DDRIVER;
```

ตารางที่ 10-3 โครงสร้างข้อมูล D3DDRIVER

จากตารางที่ 10-3 แสดงโครงสร้างข้อมูล driver ของ Direct3D ที่คลาส DX_Screen สามารถตรวจสอบได้

```
typedef struct tagMODE
{
        int             w;                  // width ของ display mode
        int             h;                  // height ของ display mode
        int             bpp;                // bit per pixel ของ display mode
        BOOL            bThisDriverCanDo;   // Direct3D driver ปัจจุบันสามารถใช้ mode นี้ได้
หรือไม่
} MODE, * LPMODE;
```

ตารางที่ 10-4 โครงสร้างข้อมูล MODE

จากตารางที่ 10-4 แสดงโครงสร้างข้อมูลของ display mode ที่คลาส DX_Screen สามารถตรวจสอบได้

```
typedef struct tagDDDRIVERINFO
{
        int             NumDrivers;     // จำนวน DirectDraw drivers ทั้งหมดที่ตรวจสอบได้
        int             CurrDriver;     // DirectDraw driver ปัจจุบัน
        DDDRIVER        DDDriver[MAX_DDDRIVERS];
                                        // ข้อมูลของ DirectDraw drivers ทั้งหมด
} DDDRIVERINFO, * LPDDDRIVERINFO;
```

ตารางที่ 10-5 โครงสร้างข้อมูล DDDRIVERINFO

จากตารางที่ 10-5 แสดงโครงสร้างข้อมูล DDDRIVERINFO ซึ่งจะเก็บข้อมูลของ DirectDraw drivers ทั้งหมดที่คลาส DX_Screen สามารถตรวจสอบได้ และแสดงจำนวน DirectDraw driver ทั้งหมด และ DirectDraw driver ปัจจุบัน ซึ่งจำนวน DirectDraw drivers ทั้งหมดที่รองรับได้จะขึ้นอยู่กับ ค่า MAX_DDDRIVERS ซึ่งเราสามารถกำหนดค่าให้เหมาะสมกับแอปพลิเคชั่นของเราได้โดยอิสระ

```
typedef struct tagD3DDRIVERINFO
{
        int             NumDrivers;     // จำนวน Direct3D drivers ทั้งหมดที่ตรวจสอบได้
        int             CurrDriver;     // Direct3D driver ปัจจุบัน
        D3DDRIVER       D3DDriver[MAX_D3DDRIVERS];
                                        // ข้อมูลของ Direct3D drivers ทั้งหมด
} D3DDRIVERINFO, * LPD3DDRIVERINFO;
```

ตารางที่ 10-6 โครงสร้างข้อมูล D3DDRIVERINFO

```
typedef struct tagMODEINFO
{
        int             NumModes;       // จำนวน display modes ทั้งหมดที่ตรวจสอบได้
        int             CurrMode;       // display mode ปัจจุบัน
        MODE            Mode[MAX_MODES];
                                        // ข้อมูลของ display modes ทั้งหมด
} MODEINFO, * LPMODEINFO;
```

ตารางที่ 10-7 โครงสร้างข้อมูล MODEINFO

10.2.1 Attributes ของคลาส DX_Screen

```
HWND                    m_hWnd;                 // ค่า handle window ของแอปพลิเคชั่น
MODE                    m_WindowsDisplay;       // display mode ปัจจุบันของ Windows

LPDIRECT3DRM3           m_D3DRM;                // Direct3DRM
LPDIRECT3DRMDEVICE3     m_dev;                  // Direct3DRMDevice
LPDIRECT3DRMVIEWPORT2   m_view;                 // Direct3DRMViewport
LPDIRECT3DRMFRAME3      m_scene;                // Scene
LPDIRECT3DRMFRAME3      m_camera;               // Camera
D3DRMRENDERQUALITY      m_RenderQuality;        // คุณภาพของการแสดงผล
D3DRMTEXTUREQUALITY     m_TextureQuality;       // คุณภาพของ textures
```

```
BOOL                m_bDithering;       // เทคนิค Dithering
BOOL                m_bAntialiasing;    // เทคนิค Anti-aliasing

LPDIRECT3D2         m_D3D;              // Direct3D
LPDIRECT3DDEVICE2   m _D3DDevice;       // Direct3DDevice
D3DDRIVERINFO       m_D3DDriverInfo;    // Direct3D drivers ทั้งหมดที่ตรวจสอบได้
D3DDRIVER           m_ThisD3DDriver;    // ข้อมูลของ Direct3D driver ปัจจุบัน
D3DRENDERSTATE      m_D3DRenderState;   // คุณภาพการแสดงผล

LPDIRECTDRAW        m_DD;               // DirectDraw
BOOL                m_bIsPrimary;
                    // DirectDraw driver ปัจจุบันเป็น primary driver หรือไม่
DX_Surface*         m_FrontBuffer;
DX_Surface*         m_BackBuffer;
DX_Surface*         m_ZBuffer;
DX_Clipper*         m_Clipper;

BOOL                m_bBackBufferInVideo; // backbuffer อยู่ใน video memory หรือไม่
BOOL                m_bZBufferInVideo;    // z-buffer อยู่ใน video memory หรือไม่

DDDRIVERINFO        m_DDDriverInfo;     // ข้อมูลของ DirectDraw drivers ทั้งหมด
DDDRIVER            m_ThisDDDriver;     // ข้อมูลของ DirectDraw driver ปัจจุบัน

MODEINFO            m_ModeInfo;         // ข้อมูลของ display mode ทั้งหมด
MODE                m_ThisMode;         // ข้อมูลของ display mode ปัจจุบัน
```

10.2.2 Methods ของคลาส DX_Screen

```
DX_Screen (void);           // Constructor
~DX_Screen (void);          // Destructor
void Initialise (void);
void Finalise (void);
```

```
static BOOL WINAPI DDEnumCallback (GUID FAR* lpGUID,
                                   LPSTR lpDriverDesc,
                                   LPSTR lpDriverName,
                                   LPVOID lpContext);
static HRESULT WINAPI EnumDisplayModesCallback (LPDDSURFACEDESC pddsd,
                                                LPVOID lpContext);
static HRESULT WINAPI EnumDeviceFunc (LPGUID lpGuid,
                                      LPSTR lpDeviceDescription,
                                      LPSTR lpDeviceName,
                                      LPD3DDEVICEDESC lpHWDesc,
                                      LPD3DDEVICEDESC lpHELDesc,
                                      LPVOID lpContext);
static int CompareModes (const void* arg1, const void* arg2);

BOOL EnumerateDDDevices (LPDDDRIVERINFO lpinfo);
HRESULT CreateD3DRM (void);
HRESULT CreateDD (LPGUID lpGuid);
HRESULT RememberWindowsMode (LPMODE lpmode);
BOOL EnumerateDisplayModes (LPMODEINFO lpinfo);
HRESULT CreateD3D (void);
BOOL EnumerateDevices (LPD3DDRIVERINFO lpinfo);
BOOL VerifyDriverAndMode (int* lpdriver, int* lpmode);
BOOL PickDriver (int* driver, DWORD depths);
BOOL PickDisplayMode (int* mode, DWORD depths);
BOOL FilterDisplayModes (int driver);
DWORD BBPToDDBD (int bpp);
HRESULT SetCoopLevel (HWND hWnd);
HRESULT SetDisplayMode (int w, int h, int bpp);
HRESULT CreateBuffers (BOOL bIsHardware);
HRESULT CreateZBuffer (int w, int h, int driver);
HRESULT CreateClipper (DX_Surface* dest);
HRESULT CreateDevice (int driver);
HRESULT CreateDevAndView (void);
```

```
HRESULT D3DSetRenderState (void);
HRESULT SetRenderState (void);
void DefaultD3DRenderState (LPD3DRENDERSTATE lpstate);
BOOL ChangeDriver (int driver);
BOOL ChangeMode (int mode);
BOOL Create (HWND hWnd);
HRESULT Render (void);
HRESULT Restore (void);
HRESULT Flip (void);

LPDIRECT3DRM3           GetD3DRM (void);
LPDIRECT3DRMDEVICE3     GetDevice (void);
LPDIRECT3DRMVIEWPORT2   GetViewport (void);
LPDIRECT3DRMFRAME3      GetScene (void);
LPDIRECT3DRMFRAME3      GetCamera (void);
LPDIRECT3D2             GetD3D (void);
LPDIRECT3DDEVICE2       GetD3DDevice (void);
LPDIRECTDRAW            GetDD (void);
DX_Surface*             GetFront (void);
DX_Surface*             GetBack (void);
DX_Surface*             GetDepth (void);
DX_Clipper*             GetClipper (void);
MODE                    GetThisMode (void);
int                     GetWidth (void);
int                     GetHeight (void);
int                     GetBpp (void);
```

10.3 คลาส DX_Surface

คลาส DX_Screen นั้นประกอบด้วยคลาสสองคลาส ซึ่งหนึ่งในนั้นคือคลาส DX_Surface ซึ่งเป็นจะรวบรวมการทำงานต่างๆของ DirectDraw surface ไว้ด้วยกัน

10.3.1 Attributes ของคลาส DX_Surface

```
LPDIRECTDRAWSURFACE     m_Surface;
DDSURFACEDESC           m_Desc;
DDSCAPS                 m_Caps;
int                     m_Width;        // ความกว้างของ surface
int                     m_Height;       // ความยาวของ surface
RECT                    m_SrcRect;      // ขนาดของ Dest Rect
RECT                    m_DestRect;     // ขนาดของ Src Rect
HFONT                   m_Font;         // font handle
const char*             m_Filename;     // ชื่อของ bmp file ที่ทำการ load มา
```

10.3.2 Methods ของคลาส DX_Surface

```
DX_Surface (void);                                  // Constructor
~DX_Surface (void);                                 // Descstructor
void Initialise (void);
void Finalise (void);

HRESULT Create (DX_Screen*);                        // สร้าง primary surface ตาม m_Desc
HRESULT Create (DX_Screen*, int, int);              // สร้าง off-screen surface เปล่าๆ
HRESULT LoadBitmap (DX_Screen*, const char*);       // สร้าง off-screen จาก bitmap
HRESULT CopyBitmap (HBITMAP, int , int, int, int);
HRESULT Restore (void);
HRESULT Lock (void);                                // Lock ค่า surface ไปใส่ใน m_Desc
HRESULT UnLock (void);
HRESULT Fill (DWORD);                               // fill ค่าสีที่กำหนดทั้ง surface
HRESULT SetColorKey (void);                         // เซต color key เป็นสีดำ
HRESULT SetFont (const char*, int , int, int Attributes = FW_NORMAL);   // set font
HRESULT TextXY (int X, int Y, COLOREF, const char*);                    // draw text

virtual HRESULT Draw (DX_Surface* Dest, DWORD flags);                   // Blt
virtual HRESULT DrawFast (int X, int Y, DX_Surface* Dest, DWORD flags); // BltFast
void SetSrc (int x, int y, int width = 0, int height = 0);
void SetDest (int x, int y, int width = 0, int height = 0);
```

```
int GetWidth (void);
int GetHeight (void);
LPDIRECTDRAWSURFACE      GetSurface (void);
```

10.4 คลาส DX_Clipper

และส่วนประกอบอีกส่วนของคลาส DX_Screen คือ คลาส DX_Clipper ซึ่งเราจะใช้เป็นตัวกำหนดขอบเขตการแสดงผลที่ backbuffer

10.4.1 Attribute ของคลาส DX_Clipper

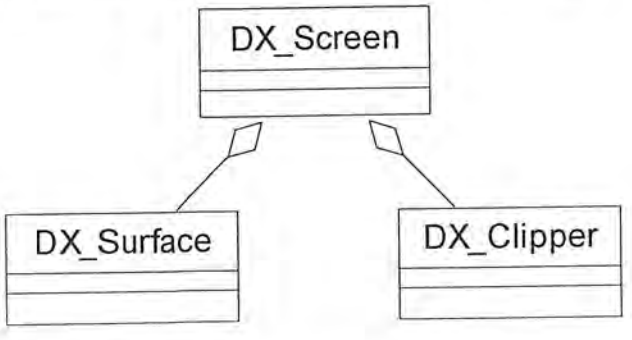
```
LPDIRECTDRAWCLIPPER      m_Clipper;
```

10.4.2 Methods ของคลาส DX_Clipper

```
DX_Clipper (void);
~DX_Clipper (void);
void Initialise (void);
void Finalise (void);

HRESULT AttachClipper (DX_Screen*, DX_Surface*, int num_rect = 1, LPRECT clip_list = NULL);
LPDIRECTDRAWCLIPPER GetClipper (void);
```

10.5 ความสัมพันธ์ระหว่าง DX_Screen, DX_Surface, และ DX_Clipper

คลาส DX_Screen, DX_Surface, และ DX_Clipper นั้นมีความสัมพันธ์กันโดยที่ DX_Screen นั้นจะเป็นคลาสหลักที่รวมส่วนประกอบต่างๆเข้าไว้ด้วยกัน ส่วนคลาส DX_Surface นั้นจะใช้เป็นส่วนประกอบของคลาส DX_Screen ในส่วนของ surfaces ทั้งหลาย ได้แก่ frontbuffer, backbuffer, และ zbuffer ส่วนคลาส DX_Clipper นั้นจะใช้เป็นส่วนที่กำหนดขอบเขตการแสดงผลที่ backbuffer ซึ่งความสัมพันธ์ระหว่างคลาส DX_Screen กับคลาส DX_Surface และ DX_Clipper นั้นจะเป็นไปในรูปแบบ "aggregation" ดังรูปที่ 10-1

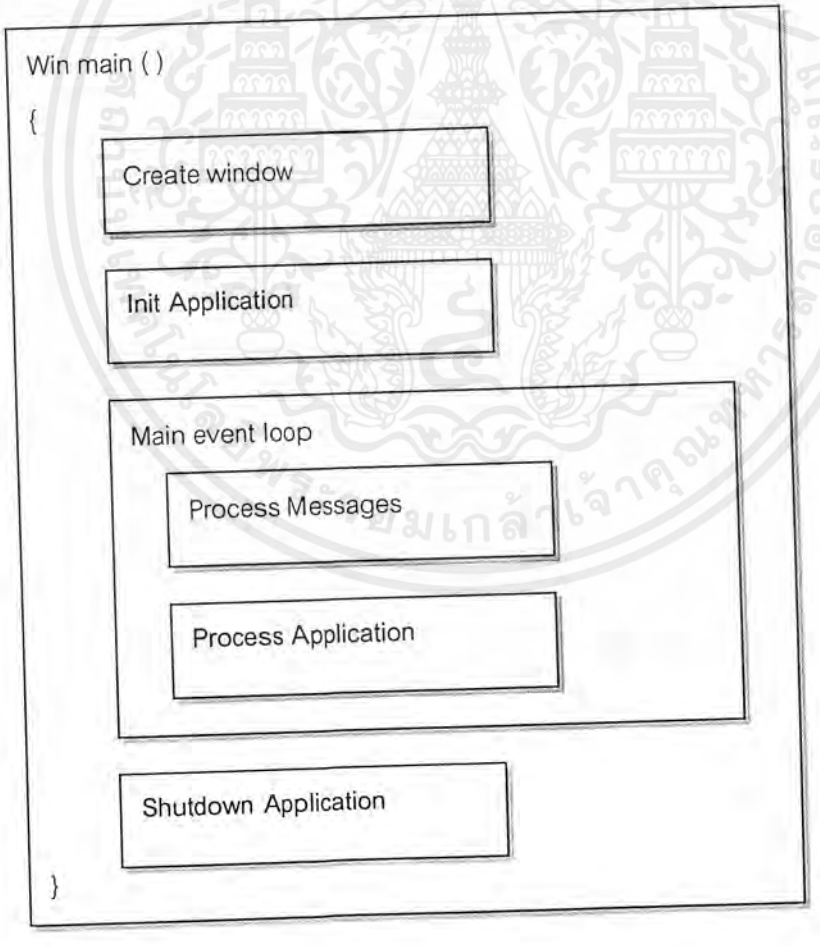


รูปที่ 10-1 ความสัมพันธ์ระหว่างคลาส DX_Screen, DX_Surface, และ DX_Clipper

10.6 รูปแบบการใช้งานคลาส DX_Screen

ความสามารถของคลาส DX_Screen นั้นเราสามารถจะสร้างโครงสร้างของแอปพลิเคชั่นในส่วนของ application dependent ได้โดยใช้คำสั่งเดียวคือเรียกใช้ method **DX_Screen::Create ( )** ซึ่งจะทำการสร้างทุกอย่างให้เราโดยเริ่มจากการตรวจสอบ DirectDraw drivers ที่มีอยู่บนเครื่องคอมพิวเตอร์ที่แอปพลิเคชั่นทำงานอยู่ และเลือก DirectDraw driver ที่มีประสิทธิภาพที่สุดเป็น default, สร้างส่วนประกอบของ Direct3D Retained-Mode, สร้าง DirectDraw จาก driver ที่ได้เลือกไว้, ทำการตรวจสอบว่าการ์ดแสดงผลนั้นสนับสนุน display modes ใดบ้างและทำการเลือกใช้ display mode ที่ถูกกำหนด, สร้าง Direct3D, ตรวจสอบ Direct3D drivers ที่มีอยู่ และเลือก driver ที่มีประสิทธิภาพมากที่สุด, สร้าง buffers, สร้าง Direct3D device, สร้าง Direct3DRM device และ viewport, สร้าง clipper และนำไปเชื่อมต่อเข้ากับ backbuffer ส่วน method **DX_Screen::Render ( )** นั้นจะทำการ update scene ไปยัง backbuffer และ methods **DX_Screen::Flip ( )** นั้นจะทำการ flip ระหว่าง frontbuffer กับ backbuffer เพื่อนำภาพที่ update แล้วแสดงออกหน้าจอ

ดังนั้นการใช้คลาส DX_Screen นั้นจึงมี methods หลักๆอยู่ 3 methods โดยการนำคลาส DX_Screen นี้ไปใช้ร่วมกับแอปพลิเคชั่นในรูปแบบ Win32 นั้นแสดงดังรูปที่ 10-2



รูปที่ 10-2 รูปแบบของแอปพลิเคชั่นแบบ Win32

จากรูปที่ 10-2 นั้น Win main ( ) จะเป็นฟังก์ชันหลักของแอปพลิเคชั่นในรูปแบบ Win32 ซึ่งจะประกอบด้วยส่วนที่เราจะใช้งานอยู่ 3 ส่วนด้วยกันคือ Init Application, Process Application, และ Shutdown Application ซึ่งมีการใช้งานดังนี้

ตัวอย่างการใช้งาน

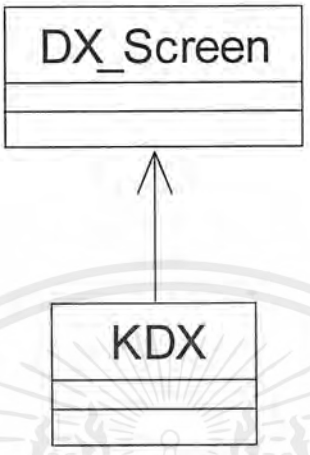
```
DX_Screen*     g_Screen;          // ประกาศ pointer ที่ชี้ไปที่ DX_Screen ในตอนต้นโปรแกรม

// ส่วน Init Application
g_Screen = new DX_Screen;         // สร้าง instance ของคลาส DX_Screen
g_Screen->Create ( );             // สร้างโครงสร้างในส่วนของ application dependent
BuildScene ( );                   // สร้างโครงสร้างในส่วนของ application independent

// ส่วน Process Application
g_Screen->Render ( );             // ทำการ update scene ไปยัง backbuffer
g_Screen->Flip ( );               // ทำการ flip ระหว่าง frontbuffer และ backbuffer

// ส่วน Shutdown Application
delete g_Screen;                  // ยกเลิกการใช้งาน instance ของคลาส DX_Screen
```

จากตัวอย่างข้างต้นในส่วนของ Init Application นั้นเป็นส่วนที่อยู่ก่อน Main event loop ซึ่งเป็นส่วนที่จะถูกเรียกเพียงครั้งเดียวในตอนเริ่มต้นการทำงานของแอปพลิเคชั่น ดังนั้นเราจึงใช้ส่วนนี้ในการสร้างโครงสร้างแบบ applicaton dependent โดยใช้ method DX_Screen::Create ( ) และสร้างส่วน application independent โดยในตัวอย่างจะใช้ฟังก์ชัน BuildScene ( ) ในส่วน Process Application นั้นจะถูกเรียกตลอดเวลาเมื่อแอปพลิเคชั่นของเราไม่ได้จัดการเกี่ยวกับ messages ดังนั้นเราจึงใช้ส่วนนี้ในการ update scene และทำการแสดงผล ส่วน Shutdown Application นั้นเป็นส่วนที่จะถูกเรียกเมื่อแอปพลิเคชั่นจะจบการทำงาน ดังนั้นเราจึงใช้ส่วนนี้ในการคืนหน่วยความจำที่ใช้สำหรับ instance ของคลาส DX_Screen

10.7 คลาส KDX

แอปพลิเคชั่นที่ใช้ Direct3D โดยทั่วไปแล้วนั้นสามารถที่จะเปลี่ยน Direct3D driver, display mode, หรือ คุณภาพของการแสดงผลในรูปแบบต่างได้ตลอดเวลา ซึ่งจะทำให้ผู้ใช้แอปพลิเคชั่นนั้นมีความอิสระในการใช้แอปพลิเคชั่นมากขึ้น เช่น ผู้ใช้ที่มีการ์ดเร่งความเร็วกราฟิกแบบ 3 มิติ ก็สามารถที่จะเลือกใช้ hardware driver ได้ ส่วนถ้าการ์ดแสดงผลของผู้ใช้นั้นมีหน่วยความจำนวนมากก็จะสามารถใช้งานแอปพลิเคชั่นใน display mode ที่มีความละเอียดสูงได้ ทำให้แอปพลิเคชั่นของเรามีความยืดหยุ่นต่อเครื่องคอมพิวเตอร์ของผู้ใช้ซึ่งมีประสิทธิภาพแตกต่างกัน

ทางคณะผู้จัดทำจึงได้มีแนวความคิดที่จะนำความสามารถเหล่านี้มาใช้ ดังนั้นจึงได้สร้างคลาสขึ้นมาชื่อว่า KDX (KOMODO DirectX) โดยพัฒนาต่อมาจากคลาส DX_Screen โดยจะใช้คุณสมบัติ "inheritance" ซึ่งเป็นเทคนิคในการพัฒนาโปรแกรมแบบเชิงวัตถุ ดังรูปที่ 10-3



รูปที่ 10-3 ความสัมพันธ์ระหว่างคลาส DX_Screen กับ KDX

คลาส DX_Screen นั้นมีความสามารถในการตรวจสอบ DirectDraw drivers, Direct3D drivers, display modes และเก็บบันทึกไว้ เพียงแต่จะเลือกส่วนที่ดีที่สุดเพื่อนำมาใช้ เช่น ถ้าพบว่ามี hardware driver ก็จะนำ driver นั้นมาใช้ ส่วน display modes นั้นจะเลือกความค่าที่ได้กำหนดไว้ซึ่งกำหนดให้เป็น 640*480*16 โดยคลาส KDX นั้น inherit มาจากคลาส DX_Screen ดังนั้นจึงรวมความสามารถเหล่านี้ไว้ด้วย โดยความสามารถของคลาส KDX ที่เพิ่มขึ้นมาจากคลาส DX_Screen นั้นคือการนำข้อมูลต่างๆที่มีอยู่ในคลาส DX_Screen มาแสดงต่อผู้ใช้ในรูปแบบของกราฟฟิก ซึ่งผู้ใช้สามารถจะตรวจสอบและเลือกใช้ได้ตามความต้องการทุกเวลาในขณะใช้แอปพลิเคชั่นอยู่

10.7.1 Frame rate

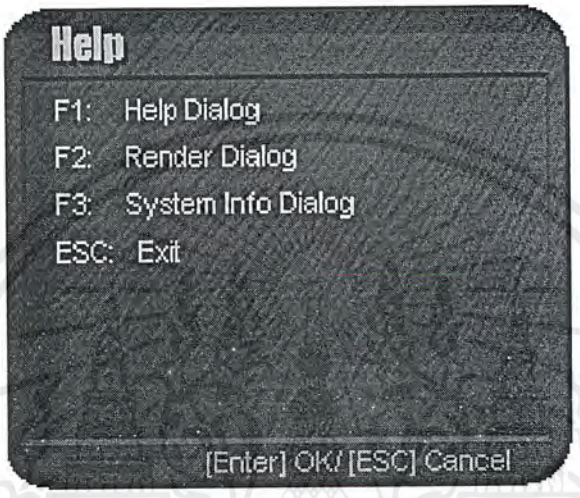
ความสามารถอย่างหนึ่งของคลาส KDX นั้นคือแสดง frame rate ซึ่งวัดเป็นหน่วย ภาพต่อวินาที ซึ่งแสดงว่าใน 1 วินาทีนั้นแอปพลิเคชั่นของเราสามารถจะวาดภาพไปยัง backbuffer และแสดงออกหน้าจอได้เป็นจำนวนเท่าใด ดังแสดงในรูปที่ 10-4

รูปที่ 10-4 Frame rate

จากรูปที่ 10-4 แสดงการภาพการรายงาน frame rate ซึ่ง “FPS: 150” นั้นหมายความว่า แอปพลิเคชั่นมี frame fate เท่ากับ 150 (FPS ย่อมาจาก Frame Per Second) ส่วน “[F1] Help” หมายความว่าเราสามารถขอความช่วยเหลือได้โดยการกดคีย์ F1

10.7.2 Help dialog

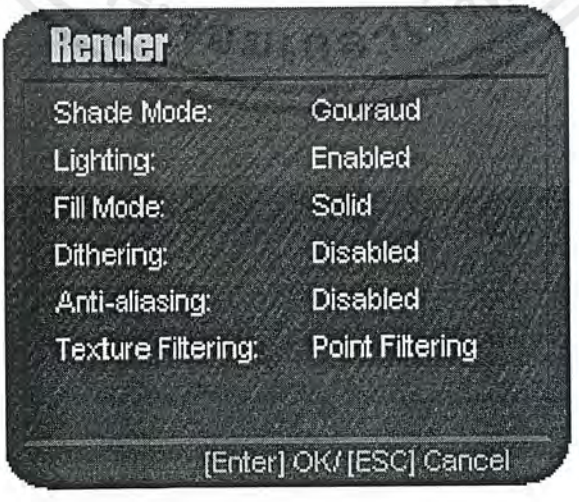
คลาส KDX นั้นมี dialog เพื่อบอกถึงวิธีการใช้งาน ซึ่งจะถูกแสดงอยู่ใน Help dialog ดังรูปที่ 10-5



รูปที่ 10-5 Help dialog

10.7.3 Render dialog

คลาส KDX นั้นมีความสามารถในการแสดงคุณภาพการแสดงผลที่ใช้อยู่ โดยจะถูกแสดงใน Render dialog ดังรูปที่ 10-6

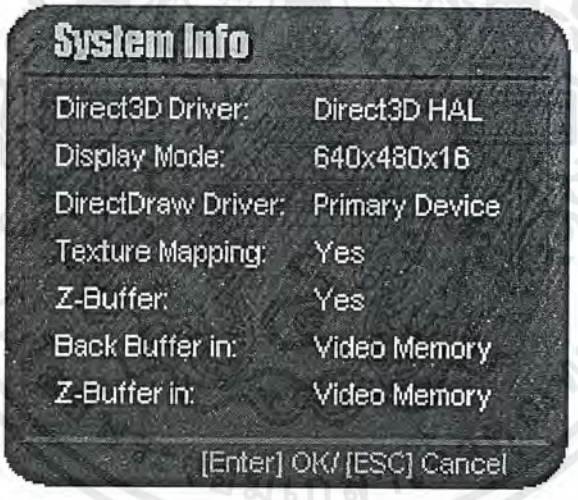


รูปที่ 10-6 Render dialog

จากรูปที่ 10-6 แสดง Render dialog ซึ่งแสดงถึงคุณภาพการแสดงผลในรูปแบบต่าง หัวข้อ **Shade Mode:** นั้นมีตัวเลือก "Flat", "Gouraud", และ "Phong" ซึ่งยังไม่ถูกสนับสนุนโดย DirectX เวอร์ชัน 6.0 ดังนั้นในที่นี้ Phong จะมีคุณภาพในระดับเดียวกับ Gouraud หัวข้อ **Lighting:** เป็นการกำหนดว่าจะให้ใช้การคำนวณเกี่ยวกับแสงหรือไม่ โดยจะมีตัวเลือก "Enabled" กับ "Disabled" หัวข้อ **Fill Mode:** มีตัวเลือก "Point", "Wireframe", และ "Solid" หัวข้อ **Dithering:** เป็นการกำหนดว่าจะใช้เทคนิค dithering หรือไม่ ซึ่งมีตัวเลือก "Enabled" กับ "Disabled" หัวข้อ **Anti-aliasing:** นั้นเป็นการกำหนดว่าจะใช้เทคนิค Anti-alias หรือไม่ มีตัวเลือกคือ "Enabled" กับ "Disabled" โดยในคลาส KDX นี้ยังไม่สนับสนุนคุณสมบัตินี้ ดังนั้นหัวข้อนี้จะเป็น "Disabled" เสมอ สุดท้ายคือหัวข้อ **Texture Filtering:** ซึ่งมีตัวเลือกคือ "Point Filtering" กับ "Bi-Linear Filtering"

10.7.4 System Info dialog

คลาส KDX มีความสามารถในการแสดงรายละเอียดเกี่ยวกับ DirectDraw driver, Direct3D drivers, display modes ซึ่งถูกแสดงอยู่ใน System Info dialog ดังรูปที่ 10-7



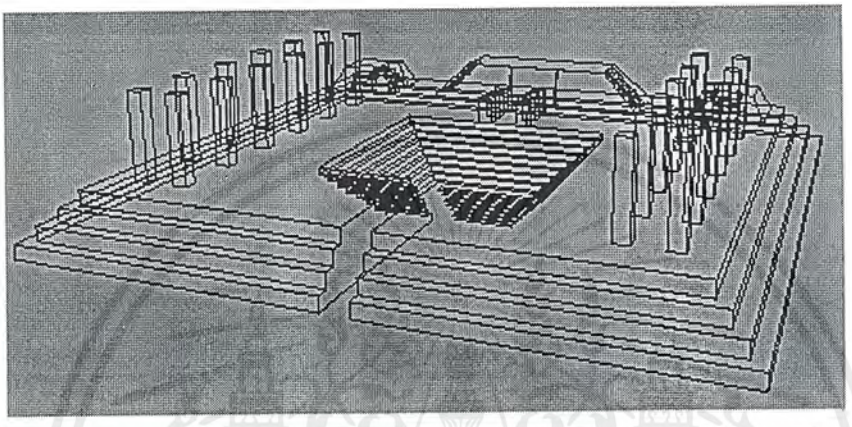
รูปที่ 10-7 System Info dialog

จากรูปที่ 10-7 แสดง System Info dialog หัวข้อ **DirectD3D Driver:** นั้นแสดง Direct3D driver ปัจจุบัน โดยเราสามารถเลือก drivers อื่นๆ ได้โดยการกดคีย์ทิศทางซ้ายขวาเพื่อเลือกและกดคีย์ enter เพื่อตกลง หัวข้อ **Display Mode:** แสดง display mode ปัจจุบัน ซึ่งเราสามารถเปลี่ยน display mode ได้โดยใช้วิธีเดียวกับการเปลี่ยน Direct3D driver ส่วนหัวข้อถัดๆมานั้นไม่สามารถเปลี่ยนตัวเลือกได้ หัวข้อ **DirectDraw Driver:** แสดงชื่อของ DirectDraw Driver ที่ใช้ หัวข้อ **Texture Mapping:** รายงานว่า Direct3D driver ปัจจุบันนั้นมีความสามารถในการจัดการกับ textures หรือไม่ หัวข้อ **Z-Buffer:** รายงานว่า Direct3D driver ปัจจุบันนั้นมี

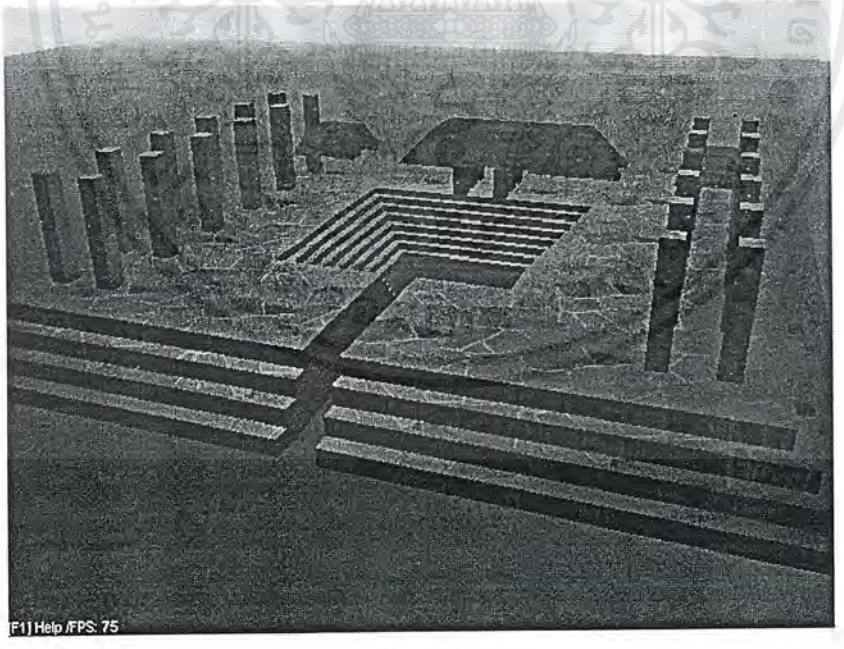
ความสามารถในการจัดการกับ z-buffer หรือไม่ ส่วน หัวข้อ **BackBuffer in:** กับ **Z-Buffer in:** นั้นรายงานว่า backbuffer กับ z-buffer นั้นอยู่ที่ใดระหว่าง system memory กับ video memory

10.8 โปรแกรม Mapper

หลังจากพัฒนาคลาส KDX เรียบร้อยแล้ว ต่อมาทางคณะผู้จัดทำโครงงานจึงได้ทดลองสร้างโมเดล 3 มิติ ที่เป็นฉากทดสอบและได้ลอง นำไปใช้กับคลาส KDX นั้นปรากฏว่าได้ผลดังรูปที่ 10-8



(a)

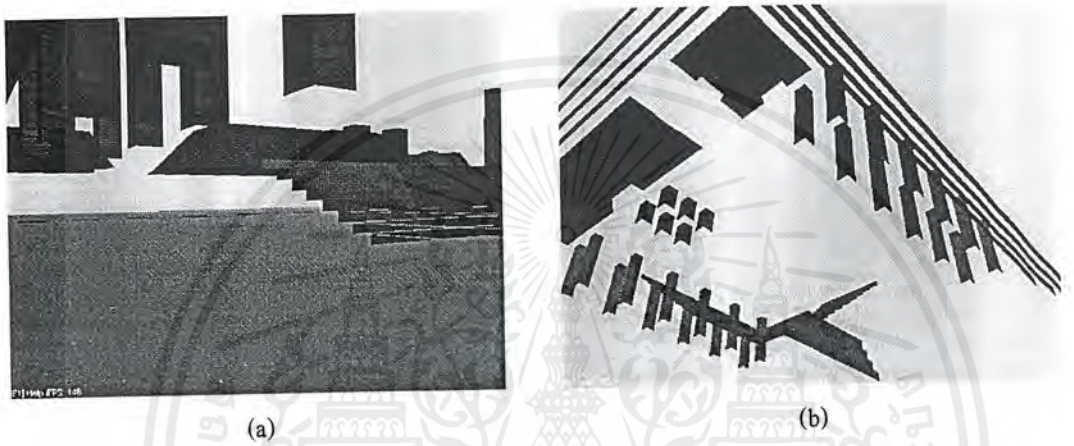


(b)

รูปที่ 10-8 โมเดลที่ทดลองใช้ร่วมกับคลาส KDX

จากรูปที่ 10-8 (a) แสดงโมเดลที่ออกถูกออกแบบโดยใช้โปรแกรม 3D Studio MAX ส่วน รูปที่ 10-8 (b) แสดงโมเดลที่นำมาใช้กับคลาส KDX

เนื่องจากจุดประสงค์ของโครงงานนี้ต้องการสร้างเกม 3 มิติ ในมุมมองของบุคคลที่หนึ่ง (first-person-perspective 3D game) โดยจะเหมือนกับว่าเราอยู่ในโลกเสมือนจริง โดยภาพที่เห็นจากเกมจะเป็นภาพที่แทนสายตาของผู้เล่น ดังนั้นทางคณะผู้จัดทำโครงงานจึงทดสอบการรับอินพุตจากทั้ง คีย์บอร์ด เมาส์ และ จอยสติ๊ก ซึ่งจะพิจารณาว่าถ้าอินพุตที่เข้ามาตรงกับที่กำหนดไว้ก็เคลื่อนย้ายส่วนของ camera ตาม อินพุตที่ถูกป้อนเข้ามา ซึ่งได้ผลออกมาดังรูปที่ 10-9

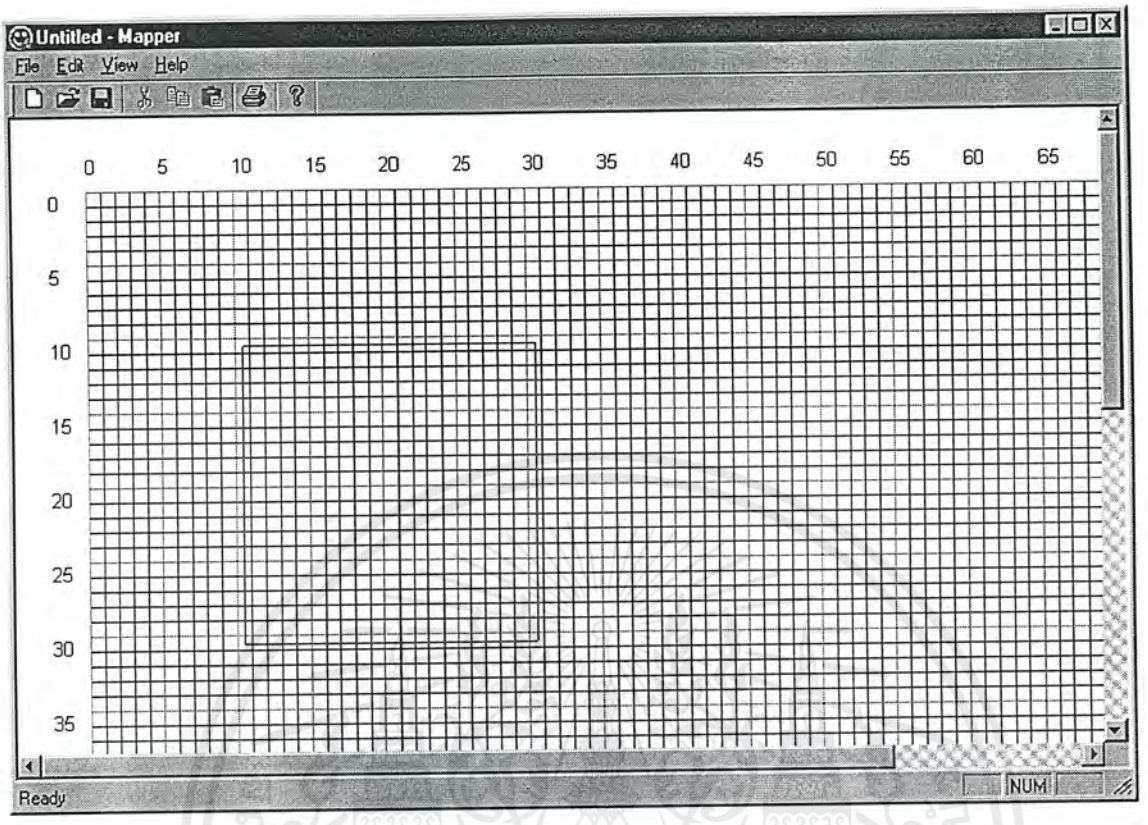


(a) (b)

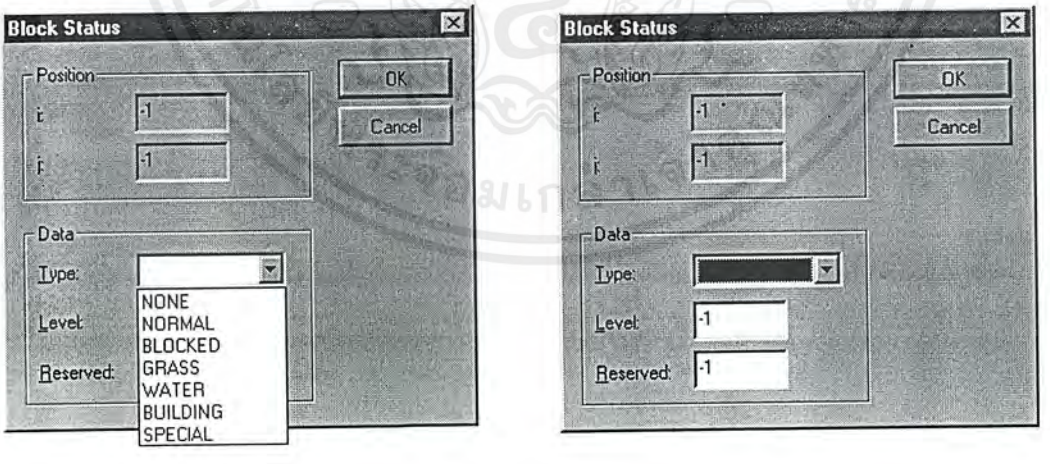
รูปที่ 10-9 การย้ายตำแหน่ง camera ตามอินพุตที่รับเข้ามา

จากรูปที่ 10-9 ทั้ง (a) และ (b) นั้นจะเห็นได้ว่าเราสามารถย้ายตำแหน่ง camera ได้อย่างอิสระซึ่ง ผิดจากความเป็นจริงเพราะเราสามารถจะทะลุเข้าไปอยู่ในวัตถุได้ ดังนั้นทางคณะผู้จัดทำโครงงานจึงได้มีแนวความคิดที่จะสร้างโปรแกรมเพื่อช่วยออกแบบแผนที่เพื่อนำแผนที่นี้ไปใช้กำหนดบริเวณที่ camera นั้นสามารถจะเคลื่อนที่ไปได้เพื่อไม่ให้เกิดปัญหาข้างต้น ดังนั้นทางคณะผู้จัดทำโครงงานจึงพัฒนาโปรแกรมเพื่อทำการออกแบบแผนที่ขึ้นมาโดยใช้ชื่อว่า "Mapper" ดังรูปที่ 10-10

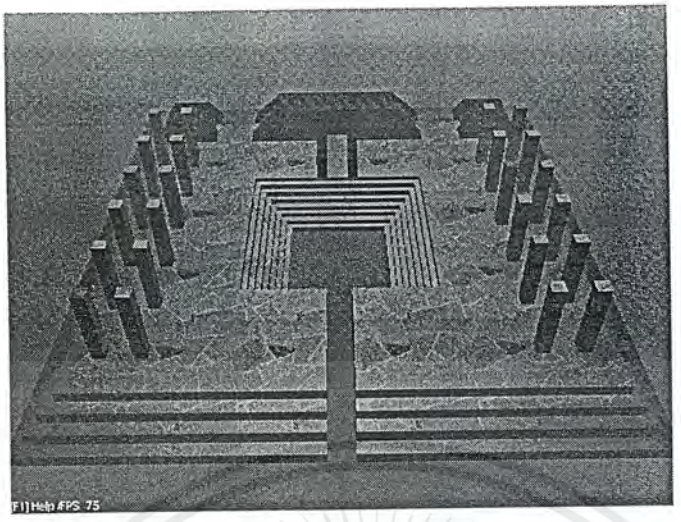
การใช้งานโปรแกรม Mapper นั้นทำได้โดยการคลิกเมาส์ค้างไว้แล้วลากรอบๆบริเวณที่เราต้องการแล้วจึงปล่อยเมาส์ จะปรากฏ Block Status dialog ดังรูปที่ 10-11 ซึ่งเราสามารถกำหนดได้ว่าบริเวณที่เราเลือกนั้น จะให้พื้นที่มีประเภทเป็นอะไรในหัวข้อ **Type:** เช่น NONE, NORMAL, BLOCKED, GRASS, WATER, BUILDING, SPECIAL และสามารถกำหนดระดับของพื้นผิวตรงส่วนนั้นได้ว่าจะให้มีความสูงเท่าใดในหัวข้อ **Level:** เพื่อว่าเราสามารถจะย้ายตำแหน่งในแนวดิ่งของ camera ให้สัมพันธ์กับระดับความสูงของแผน ณ จุดนั้นได้ซึ่งจะทำให้เกิดความเสมือนจริง เช่นการเดินขึ้นลงบันได ส่วนหัวข้อสุดท้ายคือ **Reserved:** นั้นจะเก็บข้อมูลพิเศษตามแต่จะกำหนด ซึ่งทั้ง Type และ Level นั้นจะมีผลต่อสีของ blocks ซึ่ง blocks ที่มีชนิดต่างกันจะมีสีที่ต่างกัน ส่วน blocks ที่มีสีเดียวกันแต่มี Level ต่างกันนั้นจะถูกแสดงในระดับสีที่ต่างกัน ดังรูปที่ 10-12



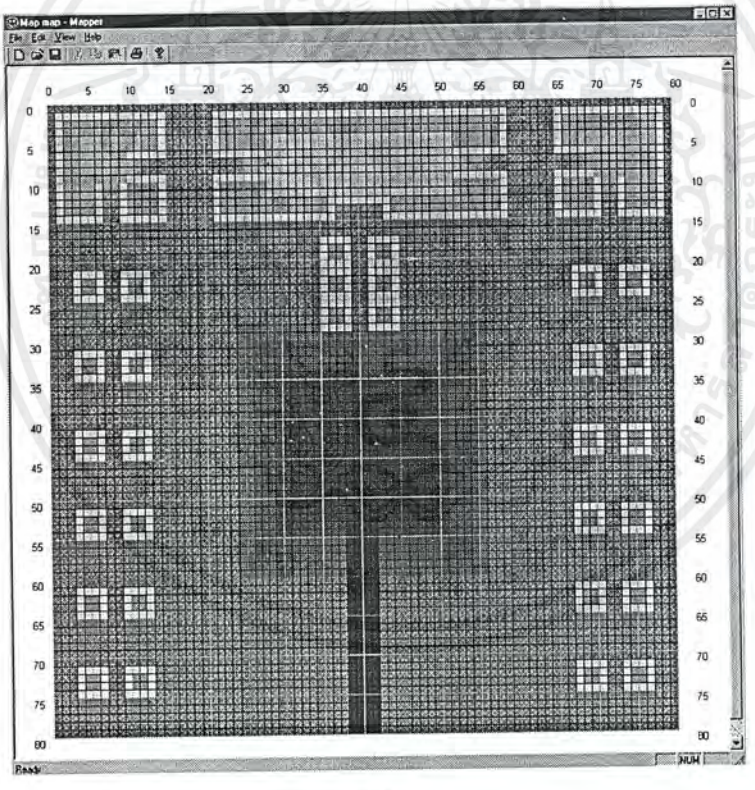
รูปที่ 10-10 Interface ของโปรแกรม Mapper



รูปที่ 10-11 Block Status dialog



(a)



(b)

รูปที่ 10-12 โมเดล 3 มิติ เปรียบเทียบกับแผนที่

จากรูปที่ 10-12 แสดง โมเดล 3 มิติ ที่นำมาใช้เป็นฉากเปรียบเทียบกับแผนที่ ซึ่งเราจะเห็นได้ว่าที่ระดับความสูงของโมเดลต่างๆกันจะถูกแสดงด้วยระดับสีที่ต่างกัน ส่วนที่เห็นได้ชัดคือส่วนของบันไดที่อยู่กลางแผนที่

10.9 คลาส KDX_Map

หลังจากพัฒนาโปรแกรม Mapper ซึ่งสามารถออกแบบแผนที่และบันทึกเก็บไว้เป็นไฟล์แล้ว ทางคณะผู้จัดทำโครงงานมีแนวความคิดที่จะนำแผนที่มาใช้ในแอปพลิเคชั่นโดยสร้างคลาสขึ้นมาเพื่อทำหน้าที่เกี่ยวกับการ load ไฟล์แผนและการตรวจสอบค่าต่างๆในแผนที่ โดยให้ชื่อคลาสนี้ว่า KDX_Map โดยโครงสร้างข้อมูลที่เก็บรายละเอียดของแต่ละ block แสดงในตารางที่ 10-8

```
typedef struct tagBLOCKSTATUS
{
        int             type;                   // ประเภทของ block
        int             level;                  // ระดับความสูงของ block
        int             reserved;               // ข้อมูลพิเศษ
} BLOCKSTATUS, * LPBLOCKSTATUS;
```

ตารางที่ 10-8 แสดง โครงสร้างข้อมูล BLOCKSTATUS

10.9.1 Attributes ของคลาส KDX_Map

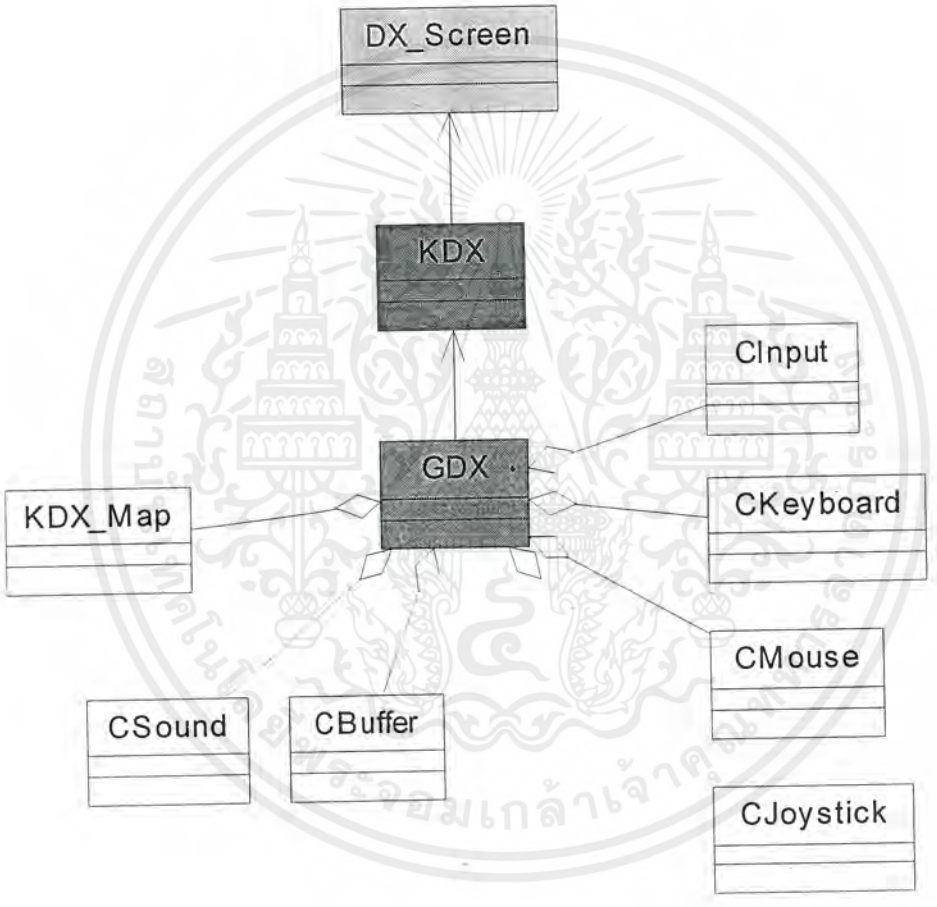
```
BLOCKSTATUS     m_Map[MAX_BLOCK][MAX_BLOCK];    // เก็บข้อมูลของ blocks ทั้งหมด
```

10.9.2 Methods ของคลาส KDX_Map

```
KDX_Map (void);                 // Constructor
~KDX_Map (void);                // Destructor
void Initialise (void);
void Finalise (void);

BOOL LoadMap (char* filename);                          // load แผนที่จากไฟล์
void FillMap (int i1, int j1, int i2, int j2, int type, int level, int reserved);
                                                        // กำหนดค่าให้แผนที่โดยตรง
BLOCKSTATUS GetBlockStatus (int i, int j);              // ตรวจสอบค่า block ที่กำหนด
int GetBlockType (int i, int j);                        // ตรวจสอบค่า Type
int GetBlockLevel (int i, int j);                       // ตรวจสอบค่า Level
int GetBlockReserved (int i, int j);                    // ตรวจสอบค่า Reserved
```

10.10 คลาส GDX

ขณะนี้เรามีคลาส KDX ซึ่งเป็นส่วนจัดการโครงสร้าง application dependent ซึ่งสามารถจะให้ผู้ใช้เลือก Direct3D drivers, display modes, และคุณภาพการแสดงผลในแบบต่างๆได้ จากบทที่ 4 และ บทที่ 5 เรามีคลาสที่ทำหน้าจัดการเกี่ยวกับ DirectInput และ DirectSound และสุดท้าย คลาส KDX_Map ที่จัดการเกี่ยวกับแผนที่ เพราะฉะนั้นในขณะนี้เราจึงมีส่วนประกอบที่จำเป็นสำหรับใช้พัฒนาแอปพลิเคชั่นทางด้านเกมแล้ว ซึ่งทางคณะผู้จัดทำโครงงานจึงมีแนวความคิดที่จะนำส่วนประกอบเหล่านี้มารวมไว้เป็นคลาสซึ่งเรียกว่า GDX โดยคลาส GDX นี้จะ inherit มาจากคลาส KDX และมีคลาสอื่นๆเป็นส่วนประกอบ ดังรูปที่ 10-13



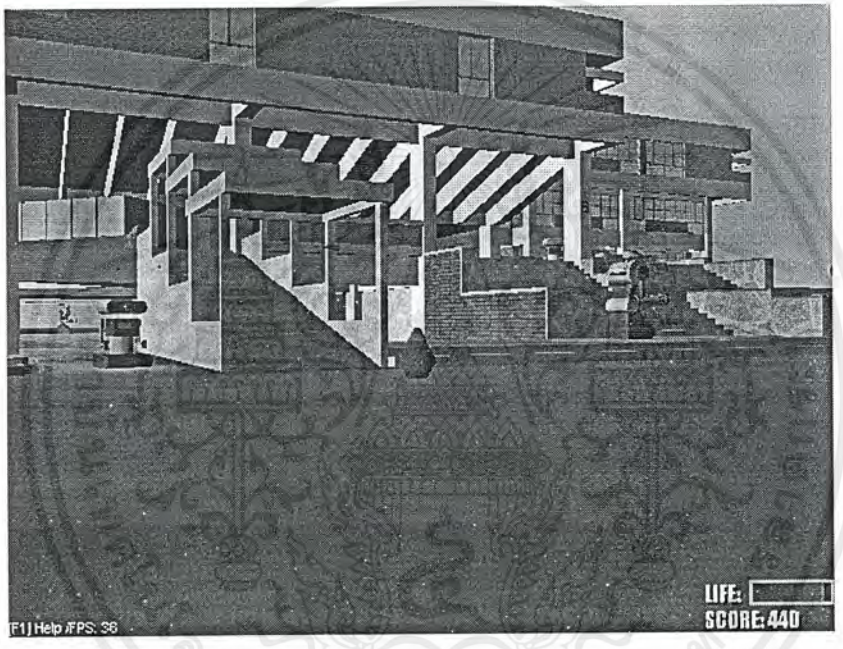
รูปที่ 10-13 ความสัมพันธ์ของคลาสทั้งหมด

จากรูปที่ 10-13 เราจะเห็นได้ว่าคลาส GDX นั้น inherit มาจากคลาส KDX และมีคลาส KDX_Map เป็นส่วนประกอบเพื่อจัดการเกี่ยวกับแผนที่ มีคลาส CSound และ CBuffer เป็นส่วนประกอบเพื่อจัดการเกี่ยวกับเสียงเพลงประกอบภายในเกม มีคลาส CInput, CKeyboard, CMouse, CJoystick เป็นส่วนประกอบเพื่อจัดการเกี่ยวกับอินพุต ซึ่งคลาส GDX จะทำหน้าที่เกี่ยวกับการควบคุมผู้เล่น, ศัตรู, และ จัดการเกี่ยวกับการนับคะแนน และระบบทั้งหมดภายในเกม

บทที่ 11
ผลลัพธ์ของโครงงาน

11.1 รูปแบบของเกม

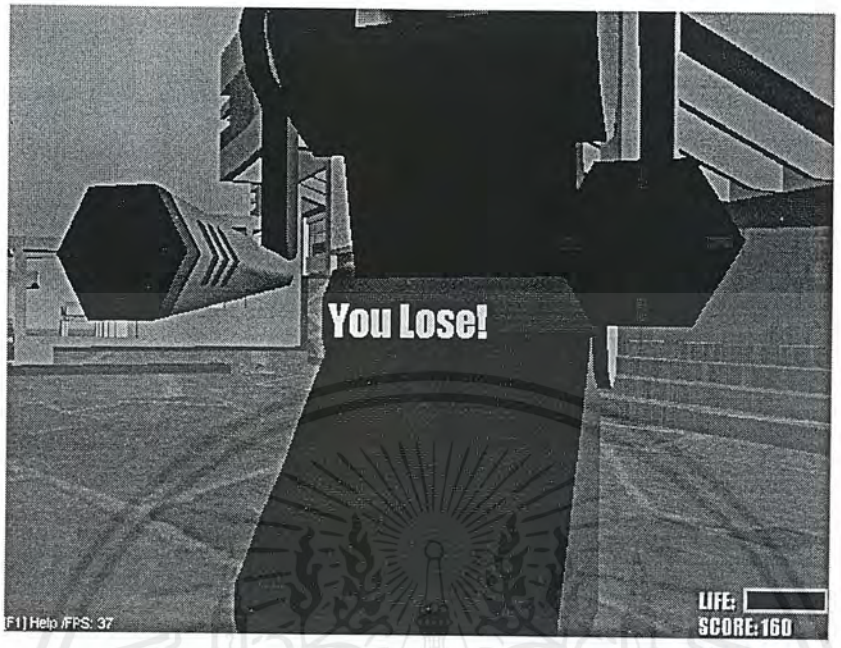
ผลลัพธ์ของโครงงานนี้ได้ออกมาในรูปแบบเกม 3 มิติ ในมุมมองของบุคคลที่หนึ่ง (first-person-perspective 3D game) โดยฉากในเกมนั้นจะใช้ฉากของตึก B ซึ่งเป็นที่ตั้งของภาควิชาวิศวกรรมคอมพิวเตอร์ ดังรูปที่ 11- 1



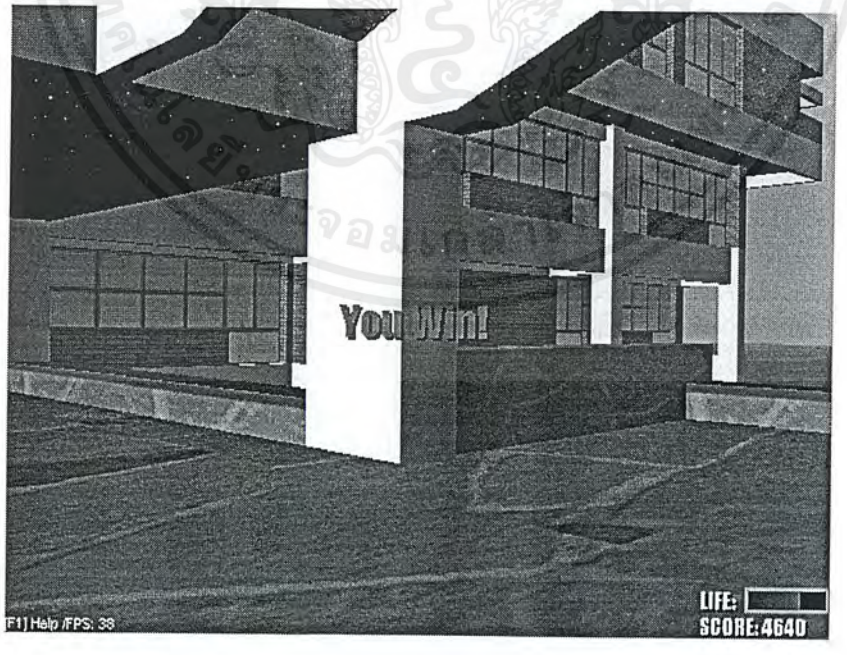
รูปที่ 11-1 ภาพภายในเกม

โดยจะมีศัตรูเป็นหุ่นยนต์ซึ่งเราสมารถทำลายได้โดยการยิงกระสุน และต้องคอยหลบกระสุนของศัตรู และไม่เข้าไปใกล้ตัวศัตรูเพราะจะทำให้เราเสียพลัง เมื่อพลังของเราหมดเราจะตายและจะปรากฏข้อความ "You Lose!" ดังรูปที่ 11-2 แต่ถ้าเรากำจัดศัตรูได้หมดจะปรากฏข้อความว่า "You Win!" ดังรูปที่ 11-3

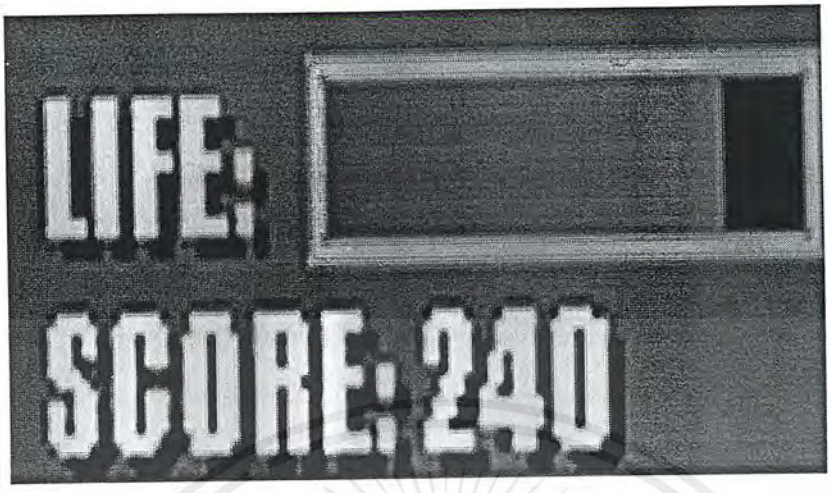
เมื่อเรายิงถูกศัตรูเราจะได้คะแนนเพิ่ม และเราสามารถดูคะแนนและพลังชีวิตของเราได้ที่บริเวณมุมล่างขวาของจอภาพ ดังรูปที่ 11-4



รูปที่ 11-2 ข้อความเมื่อผู้เล่นแพ้



รูปที่ 11-3 ข้อความเมื่อผู้เล่นชนะ



รูปที่ 11-4 พลังชีวิต และ คะแนน

11.2 การควบคุมเกม

เมื่อเริ่มเกมผู้เล่นจะพบกับฉากเริ่มต้นของเกมซึ่งจะแสดงชื่อของเกม และมีตัวเลือกให้ผู้เล่นสามารถเลือกได้คือ "New Game" เริ่มเล่นเกม "Option" ให้ผู้เล่นเลือกระดับความยากง่ายของเกม และ "Quit" ให้ผู้เล่นออกจากเกม ดังรูปที่ 11-5

รูปที่ 11-5 ฉากเริ่มต้นของเกม

เมื่อผู้เล่นเลือก "New Game" การเล่นเกมจะเริ่มขึ้น โดยที่ผู้เล่นสามารถจะบังคับเกมได้โดยใช้ ทั้งคีย์บอร์ด, เมาส์,และ จอยสติ๊ก พร้อมๆกัน

11.2.1 การควบคุมเกมโดยใช้คีย์บอร์ด

- **Up** : เดินไปข้างหน้า
- **Down** : เดินถอยหลัง
- **Left** : หันซ้าย
- **Right** : หันขวา
- **Page Up** : เงยขึ้นข้างบน
- **Page Down** : ก้มหน้าลง
- **Home** : กลับมามองในแนวระนาบ
- **Spacebar** : ยิงกระสุน

ระหว่างเล่นเกมถ้าผู้เล่นกดคีย์ Escape เกมจะหยุด และแสดงตัวเลือกให้ผู้เล่นเลือกว่าจะกลับไปเล่นเกมต่อ หรือ จะกลับไปที่ฉากเริ่มต้นของเกม ดังรูปที่ 11-6 ถ้าเราเลือก "Resume Game" ก็จะกลับไปเล่นเกมต่อ ส่วนถ้าเราเลือก "Return to Title" ก็จะกลับไปที่ฉากเริ่มต้นของเกม ซึ่งเราสามารถที่จะเลือกได้ว่าเราจะเริ่มเล่นเกมใหม่ หรือ ว่าจะออกจากเกม

รูปที่ 11-6 การหยุดเกม

11.2.2 การควบคุมเกมโดยใช้เมาส์

การควบคุมเกมโดยใช้เมาส์นั้นครอบคุมไม่ครบทุกการทำงาน โดยมีการทำงานที่ควบคุมได้ ดังนี้

- **Move Left** : หันไปทางซ้าย
- **Move Right** : หันไปทางขวา
- **Move Up** : เงยขึ้นข้างบน
- **Move Down** : ก้มหน้าลง
- **Right Button** : กลับมามองในแนวระนาบ
- **Left Button** : ยิงกระสุน

11.2.3 การควบคุมเกมโดยใช้จอยสติ๊ก

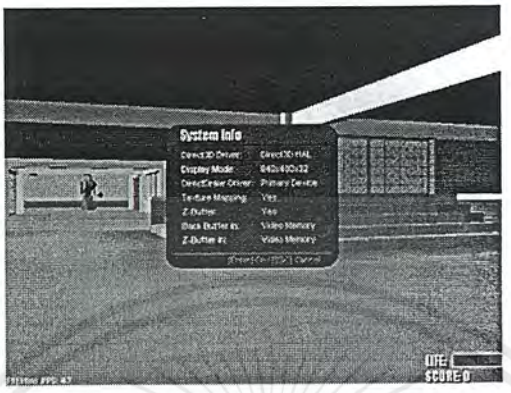
- **Up** : เดินไปข้างหน้า
- **Down** : เดินถอยหลัง
- **Left** : หันซ้าย
- **Right** : หันขวา
- **Button1** : เงยขึ้นข้างบน
- **Button2** : ก้มหน้าลง
- **Button3** : กลับมามองในแนวระนาบ
- **Button4** : ยิงกระสุน

11.3 การควบคุมการแสดงผลภายในเกม

ภายในเกมผู้เล่นสามารถเลือกรูปแบบการแสดงผลแบบต่างๆได้ในทุกขณะที่เล่นเกมอยู่ เมื่อกดปุ่ม F1 จะแสดง Help dialog ที่แสดงถึงคีย์ที่ใช้ในการเลือก dialogs ต่างๆ กด F2 จะปรากฏ Render dialog ที่ใช้ในการเลือกคุณภาพในการแสดงผล ดังรูปที่ 11-7

รูปที่ 11-7 Render dialog ขณะเล่นเกม

เมื่อกด F3 จะปรากฏ System Info dialog ซึ่งผู้ใช้สามารถจะเลือก Direct3D drivers และ display modes ได้ ดังรูปที่ 11-8



รูปที่ 11-8 System Info dialog ขณะเล่นเกม

บทที่ 12
บทวิจารณ์และสรุป

12.1 ประเมินผล

จากผลลัพธ์ของโครงงานที่ได้ออกมานั้นเมื่อเทียบกับวัตถุประสงค์ที่ได้ถูกกำหนดไว้ในตอนแรกแล้ว สรุปว่าโครงงานนี้บรรลุตามวัตถุประสงค์ โดยผลลัพธ์ของโครงนั้นได้ออกมาในรูปแบบของ เกม 3 มิติ ในมุมมองของบุคคลที่หนึ่งซึ่งการพัฒนานั้นไม่ได้เน้นในด้านของความสนุกของเกมแต่จะเน้นในด้านการนำความสามารถต่างๆของ DirectX มาทดลองใช้งาน

12.2 แนวทางการพัฒนาต่อ

ตามความเป็นจริงแล้วแอปพลิเคชั่นทางด้านเกมที่มีวางขายอยู่ในท้องตลาดนั้นเกือบทั้งหมดถูกพัฒนาโดยใช้ Direct3D Immediate Mode ซึ่งโครงงานนี้ได้ถูกพัฒนาโดยใช้ Direct3D Retained-Mode ซึ่งแม้ว่า Direct3D Retained-Mode นั้นจะใช้งานง่ายกว่า Direct3D Immediate Mode เป็นอย่างมากแต่ก็มีความยืดหยุ่นและประสิทธิภาพการทำงานที่ด้อยกว่า การพัฒนาโดยใช้ Direct3D Immediate Mode เราต้องจัดการทุกๆอย่างในระดับต่ำ (low-level) โดยการจะสร้างเกมที่เล่นแล้วสนุกนั้นหมายถึง เกมที่มีฉากจำนวนมากมีความและสวยงาม มีศัตรูมากมายหลายประเภท มีการเคลื่อนไหวของตัวละครที่เสมือนจริง ซึ่งสิ่งเหล่านี้นั้นไม่สามารถจะทำได้หรือทำได้ยากโดยไม่อาศัยเครื่องมือช่วย (utilities) เช่นการสร้างฉากนั้นในโครงงานนี้จะใช้โปรแกรม 3D Studio Max ซึ่งหลังจากสร้างโมเดล 3 มิติ ที่จะนำมาใช้เป็นฉากภายในเกมแล้ว เราต้องใช้โปรแกรม Mapper เพื่อสร้างแผนที่ให้ตรงกับโมเดล 3 มิติ ซึ่งโปรแกรม Mapper นั้นมีข้อจำกัดมากมายเช่นไม่สามารถใช้กับฉากขนาดใหญ่ได้ โดยในการพัฒนาเกมของต่างประเทศนั้นได้มีการสร้างโปรแกรมสำหรับช่วยสร้างโมเดล 3 มิติที่จะนำมาใช้เป็นฉากและสามารถแปลงให้อยู่ในรูปไฟล์แผนที่ได้โดยอัตโนมัติ ทำให้มีความสะดวกสบายและความยืดหยุ่นสูง และการออกแบบการเคลื่อนไหวของตัวละครหรือส่วนของแอนิเมชันนั้น ใน Direct3D Immediate Mode ไม่สนับสนุนการทำงานด้านแอนิเมชันเหมือนใน Direct3D Retained-Mode ดังนั้นเราจึงต้องมาจัดการในเรื่องเหล่านี้เอง ซึ่งก็ต้องการเครื่องมือช่วยอีกเช่นกัน และการสร้างเครื่องมือช่วยเหล่านี้นั้นต้องอาศัยความรู้ในด้านคอมพิวเตอร์กราฟิกแบบ 3 มิติ อย่างมาก โดยที่เราไม่สามารถจะพัฒนาเครื่องมือช่วยเหล่านี้ได้ในระยะเวลาสั้นๆ

เนื่องด้วยตอนต้นของการพัฒนาโครงงานนี้ DirectX SDK นั้นออกมาล่าสุดในเวอร์ชัน 6.0 ที่มี Direct3D Immediate Mode และ Direct3D Retained-Mode ให้เลือกใช้ซึ่งทางผู้จัดทำโครงงานนั้นได้เลือกใช้ Direct3D Retained-Mode เพราะทางคณะผู้จัดทำโครงงานยังไม่มีความคุ้นเคยกับการพัฒนาแอปพลิเคชั่นทางด้านเกมในรูปแบบ 3 มิติ แต่เมื่อมาถึงขณะนี้ไมโครซอฟต์ได้พัฒนา DirectX SDK เวอร์ชัน 7.0 ออกมา ซึ่งใน DirectX SDK เวอร์ชัน 7.0 นี้ไมโครซอฟต์เน้นการพัฒนาไปที่ Direct3D Immediate Mode เท่านั้น แต่ผู้พัฒนาที่ยังต้องการใช้ Direct3D Retained-Mode นั้นก็ยังคงสามารถใช้ได้อยู่แต่จะอยู่ในรูปแบบและความสามารถที่

เท่ากับ Direct3D Retained-Mode ที่มีอยู่ใน DirectX SDK เวอร์ชัน 6.0 สรุปคือทางไมโครซอฟต์หยุดพัฒนา Direc3D Retained-Mode แล้วนั่นเอง

จากเหตุผลข้างต้นถ้ามีการพัฒนาโครงงานนี้ต่อไป ไม่ควรจะใช้ Direct3D Retained-Mode เพราะเป็นส่วนที่ล้าสมัยไปแล้ว แต่การพัฒนาแอปพลิเคชั่นทางด้านเกมโดยใช้ Direct3D Immediate Mode นั้นมีความจำเป็นที่จะต้องอาศัยเครื่องมือช่วย โดยที่เครื่องมือช่วยประเภทนี้สามารถจะหาได้จากในอินเตอร์เนต โดยที่เราสามารถนำเครื่องมือช่วยเหล่านี้มาทดลองและเรียนรู้วิธีใช้งาน ซึ่งจะทำให้การพัฒนาแอปพลิเคชั่นทางด้านเกมนั้นมีความง่ายดายขึ้นมาก ตัวอย่างของเครื่องมือเหล่านี้ได้แก่ Genesis3D โดยสามารถดาวน์โลดเครื่องมือนี้ได้จาก www.genesis3d.com และมีเครื่องมือตัวอื่นๆอีกหลายตัวด้วยกันแต่ Genesis3D นี้จะเป็นตัวที่มีความโดดเด่นมากที่สุดเพราะมีโปรแกรมสำหรับออกแบบฉากโดยเฉพาะ และมีโปรแกรมที่สามารถทำให้เรานำไฟล์ของโมเดล 3 มิติ จากโปรแกรม 3D Studio MAX มาใช้ได้

บรรณานุกรม

[1] Andre LaMothe : "*Windows GAME PROGRAMMING for DUMMIES*", IDG Books Worldwide, Inc. 1998.

[2] Bradley Bargen, Peter Donnelly : "*Inside DirectX*", Microsoft Press. 1998.

[3] Stan Trujillo : "*CUTTING-EDGE Direct3D Programming*", The Coriolis Group, Inc. 1996.

[4] Peter J. Kovach : "*The Awesome Power of Direct3D/DIRECTX*", Manning Publications Co., 1998.

[5] Rob Glidden : "*Graphics Programming with Direct3D Techniques and Concepts*", Addison – Wesley Developers Press, 1997.

[6] ชัยวัฒน์ คำรัตน์ : "*3D GAME PROGRAMMING with DIRECTX เล่ม 1*", บริษัท คอมพิวเตอร์ เอจ เทคโนโลยี จำกัด, พ.ศ. 2542

[7] ชัยวัฒน์ คำรัตน์ : "*3D GAME PROGRAMMING with DIRECTX เล่ม 2*", บริษัท คอมพิวเตอร์ เอจ เทคโนโลยี จำกัด, พ.ศ. 2542

[8] ชัยวัฒน์ คำรัตน์ : "*Game Engine เล่ม 1*", บริษัท คอมพิวเตอร์ เอจ เทคโนโลยี จำกัด, พ.ศ. 2542

[9] David S. Linthicum, Larry Klein : "*Using Turbo C++*", Que Corporation, 1993.

[10] Ori Gurewich, Nathan Gurewich : "*teach yourself Visual C++ 4*", Sams Publishing, 1996.

[11] Jeff Prosise : "*Programming Windows with MFC Second Edition*", Microsoft Press, 1999.

[12] นิรุธ อำนวยศิลป์ : "*คู่มือการเขียนโปรแกรม Microsoft Visual C++ Version 6.0*", บริษัท ซัคเซส มีเดีย จำกัด

[13] "*นิตยสาร QuickPC*", เล่มที่ 31 ปักษ์แรก กันยายน 2541, หน้า 24-25

[14] "*นิตยสาร QuickPC*", เล่มที่ 38 ปักษ์หลัง ธันวาคม 2541, หน้า 62-78

[15] "*นิตยสาร QuickPC*", เล่มที่ 43 ปักษ์แรก มีนาคม 2542, หน้า 80-91

[16] Mark Giambruno : "*3D GRAPHICS & ANIMATION From Starting Up to Standing Out*", New Riders Publishing, 1997.

[17] Steven Elliott, Phillip Miller : "*Inside 3D STUDIO MAX*", New Riders Publishing, 1996.

[18] Michael Todd Peterson : "*3D Studio MAX 2 FUNDAMENTALS*", New Riders Publishing, 1998.