การพัฒนาแอปพลิเคชันกราฟิกความเร็วสูงบนวินโดว์ด้วย DirectX

High speed graphic for windows with DirectX technology

โดย

นายพิเชียร อนวัชรพันธ์ 37014288

นายสรวิชญ์ พลสิทธิ์ 37014474

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2540

ปริญญานิพนธ์ปีการศึกษา 2540

ภาควิชา วิศวกรรมศาสตร์คอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาแอปพลิเคชันกราฟิกความเร็วสูงบนวินโดว์สด้วย DirectX

(High speed graphic for windows with DirectX technology)

ผู้จัดทำ

นายพิเชียร อนวัชพันธ์ รหัสประจำตัว 37014288

นายสรวิชญ์ พลสิทธิ์ รหัสประจำตัว 37014474

.....อาจารย์ที่ปรึกษา

(.....)

การพัฒนาแอปพลิเคชันกราฟฟิกความเร็วสูงบนวินโดว์โดยใช้เทคโนโลยี DirectX

นายพิเชียร อนวัชพันธุ์
นายสรวิชญ์ พลสิทธิ์
ดร. บุญธีร์ เครือตราชู อาจารย์ที่ปรึกษา
ปีการศึกษา 2540

บทคัดย่อ

DirectX เป็นกลุ่มของเทคโนโลยีที่พัฒนาโดยไมโครซอฟท์ เพื่อให้แอปพลิเคชันทางด้านมัลติมีเดียบน PC ที่ใช้ระบบปฏิบัติการวินโดว์มีประสิทธิภาพสูงยิ่งขึ้นอย่างเช่น สามารถแสดงสีได้เหมือนจริง, เล่นวิดีโอ, แสดงอนิเมชันสามมิติ และระบบเสียงรอบทิศทาง โดยรวมเข้ากับระบบปฏิบัติการในตระกูลวินโดว์ ปริญญานิพนธ์ฉบับนี้ได้ศึกษาสมบัติและข้อดีของ DirectX เพื่อที่จะเป็นเครื่องมือช่วยในการพัฒนาแอปพลิเคชันกราฟฟิกความเร็วสูงบนวินโดว์

ไมโครซอฟท์ประสบความสำเร็จในการพัฒนา DirectX โดยประกอบไปด้วยกลุ่มของคำสั่งและคอมโปเน้น เพื่อช่วยในการบรรลุวัตถุประสงค์สองประการ ประการแรก DirectX ช่วยให้นักพัฒนาสามารถทำแอปพลิเคชันทางมัลติมีเดียที่ทำงานบนเครื่องพีซีที่ใช้วินโดว์เครื่องใดๆก็ได้ โดยไม่ขึ้นกับฮาร์ดแวร์ และในเวลาเดียวกันสามารถได้รับประสิทธิภาพจากฮาร์ดแวร์นั้นได้อย่างเต็มที่เท่าที่ฮาร์ดแวร์นั้นจะให้ได้ ประการที่สอง DirectX ทำให้ทำให้นักพัฒนาทำงานได้อย่างสะดวกมากยิ่งขึ้น โดยมีเครื่องมือช่วยในการสร้างและเล่นเกี่ยวกับมัลติมีเดีย และยังช่วยในการใช้สื่อของมัลติมีเดียมารวมเข้าด้วยกันได้โดยง่าย DirectX จะใช้ DirectMedia ช่วยในการนี้โดยเฉพาะ

DirectX สามารถช่วยให้นักพัฒนามีโอกาสที่จะสร้างสรรค์งานตามความต้องการของตัวเอง โดยไม่ต้องคำนึงถึงชนิดของการ์ดแสดงผล, การ์ดเสียง และชิปเร่งการทำงานสามมิติที่อยู่บนเครื่อง และ DirectX ได้ออกแบบมาเพื่อรองรับฮาร์ดแวร์ในอนาคต ดังนั้นรับรองได้ว่านักพัฒนาและผู้บริโภคจะได้รับประสิทธิภาพสูงสุดตามความก้าวหน้าของเทคโนโลยีในอนาคต

High Speed Graphic for Windows with DirectX

Mr.Bhichien Anawachapun

Mr.Sorawit Bholsithi

Dr. Boontee Kruatrachue Advisor

1997

Abstract

DirectX is a group of technologies designed by Microsoft to make Windows-based computers for running and displaying applications rich in multimedia elements such as full-color graphics, video, 3D animation, and surround sound. Built directly into the Windows family of operating systems. This project will educated about property and advantage of DirectX that can be the tool for develop high speed graphic application for windows

Microsoft's goal in developing DirectX was to provide developers with a common set of instructions and components that would accomplish two things. **First**, DirectX would allow developers to be confident that their multimedia applications would run on any Windows-based PC, no matter what the hardware, and at the same time ensure that their products take full advantage of high-performance hardware capabilities to achieve the best possible performance. **Second**, DirectX would make life easier for developers by giving them tools that simplify the creation and playback of multimedia content, while at the same time making it easier to integrate a wide range of multimedia elements. DirectX does this with the DirectX Media layer.

DirectX provides developers with new opportunities for creativity and innovation by allowing them to focus on building unique features for their application without having to worry about which display adapter, soundcard, or 3D accelerator chip is installed in your PC. And because DirectX was designed to support future innovations in software and hardware, developers and consumers can be confident that they will continue to get the best possible performance from their applications as technology advances.

สารบัญ

บทที่ 1
บทนำ

ขอบเขตในการทำโปรเจ็กชุดนี้เป็นการศึกษาความสามารถและการใช้งานของ DirectX ซึ่งจะใช้ DirectX Software Development Kit (SDK) (โปรเจ็กชิ้นนี้อยู่ระหว่างคาบเกี่ยวของการพัฒนา DirectX เวอร์ชัน 3 และ 5 ซึ่งขณะนี้ได้เปลี่ยนไปใช้ชุดพัฒนาของเวอร์ชัน 5 ทั้งหมดเรียบร้อยแล้ว) และ Visual C++ เวอร์ชัน 5 ในเป็นเครื่องมือหลักในการพัฒนา การศึกษาจะเน้นในการใช้งาน และแสดงความสามารถขององค์ประกอบต่างๆ ที่มีใน DirectX ซึ่งได้แก่ DirectDraw, DirectSound, DirectInput, Direct3D และ DirectPlay

เหตุผลที่เลือกศึกษา DirectX เนื่องจากเป็นเทคโนโลยีใหม่และน่าสนใจ มีแนวโน้มที่จะเป็นมาตรฐานเทคโนโลยีของเกมส์และมัลติมีเดียในอนาคต DirectX จะเปลี่ยนรูปแบบการเขียนโปรแกรมที่ต้องการความเร็วสูงอย่างเช่นด้านเกมส์ จากเดิมที่การพัฒนาอยู่บนดอสเป็นส่วนใหญ่ ให้ย้ายมาบนวินโดว์ โดยสามารถทำได้อย่างมีประสิทธิภาพและง่ายยิ่งขึ้น ดังนั้น DirectX อาจเป็นตัวกำหนดแนวโน้มของอุตสาหกรรมฮาร์ดแวร์ในอนาคตด้วย

ประโยชน์ที่ได้รับจากการพัฒนา DirectX บนวินโดว์

จุดประสงค์แรกที่ทางไมโครซอฟท์พัฒนา DirectX ออกมาเพื่อตลาดทางด้านเกมส์บนพีซี ซึ่งแต่เดิมมีฐานการผลิตหลักอยู่บนดอส เหตุที่เกมส์ทำบนดอสเพราะว่าดอสเกิดก่อนวินโดว์ และวินโดว์เองไม่เหมาะกับการพัฒนาเกมส์ การพัฒนาเกมส์บนดอสมีประโยชน์ตรงที่นักพัฒนาสามรถติดต่อฮาร์ดแวร์ของอุปกรณ์ต่างๆได้โดยตรงเพื่อความรวดเร็วของเกมส์ แต่นักพัฒนาจำเป็นจะต้องจัดการเกี่ยวกับฮาร์ดแวร์เองทั้งหมด แต่ด้วยความสามารถของ DirectX ทำให้นักพัฒนาสร้างแอปพลิเคชันโดยไม่ต้องคำนึงถึงฮาร์ดแวร์ที่ใช้ แต่ก็ไม่สูญเสียความสามารถในการเข้าถึงฮาร์ดแวร์โดยตรง DirectXในขณะนี้มีความสามารถเทียบเท่ากับที่ดอสมีในปัจจุบัน นอกจากนี้เมื่อเทียบกับแอปพลิเคชันดั้งเดิมของวินโดว์ DirectX มีความสามารถดีกว่ามาก สามารถเข้าถึงฮาร์ดแวร์แบบ real-time ทั้งที่มีอยู่ในปัจจุบันและที่จะมีในอนาคต ซึ่งจะกำหนดมาตรฐานในการอินเตอร์เฟสระหว่างฮาร์ดแวร์กับแอปพลิเคชัน ลดความซับซ้อนในการติดตั้ง การปรับค่าคอนฟิกต่างๆ ของฮาร์ดแวร์ และทำให้เกิดประโยชน์สูงสุดในการใช้งาน อินเตอร์เฟสที่ DirectX มีให้นักพัฒนาใช้ช่วยให้นักพัฒนาไม่ต้องศึกษารายละเอียดของฮาร์ดแวร์ที่มีหลากหลายอีกต่อไป

DirectX เป็นผู้กำหนดแนวทางในเทคโนโลยีฮาร์ดแวร์

DirectX จะทำหน้าที่กำหนดมาตรฐานในการอินเตอร์เฟสระหว่างฮาร์ดแวร์และซอฟท์แวร์โดยไม่ต้องคำนึงถึงผู้ผลิตฮาร์ดแวร์โดยเฉพาะเจาะจง มีฟังก์ชันที่เรียกใช้ความสามารถทางฮาร์ดแวร์ที่ล้ำยุคมากซึ่ง

บางฟังก์ชันอาจไม่มีในฮาร์ดแวร์ในปัจจุบัน แต่สามารถเรียกใช้งานได้โดยการจำลองการทำงานการทำงานของฟังก์ชันนั้น

องค์ประกอบต่างๆที่มีใน DirectX

DirectX ประกอบไปด้วยคอมโปเน้นต่างๆที่จำเป็นในการพัฒนาแอปพลิเคชันทางเกมส์และมัลติมีเดีย ซึ่งจะมีดังนี้

DirectDraw ใช้ในการเร่งความเร็วของฮาร์ดแวร์ และเทคนิคที่มีประสิทธิภาพของการทำภาพเคลื่อนไหว โดยที่สามารถเข้าถึงหน่วยความจำทั้งในและนอกส่วนแสดงผล และใช้ความสามารถสูงสุดของฮาร์ดแวร์ในการเคลื่อนย้ายข้อมูลระหว่างวิดีโอเมมโมรี และการสลับเพจของการแสดงผล

DirectInput มีความสามารถในการใช้อุปกรณ์อินพุตได้หลากหลาย โดยใช้งานผ่านไดร์เวอร์และ API ของวินโดว์ ตอนนี้สนับสนุนอุปกรณ์ในปัจจุบันที่มีอยู่เช่น คีย์บอร์ด, เมาส์, joystick และอุปกรณ์ชนิดใหม่คือ Force Feed Back

DirectSound มีความสามารถในการผสมและเล่นเสียงในหลายรูปแบบทั้งใช้ความสามารถของฮาร์ดแวร์และซอฟท์แวร์ ทั้งยังทำระบบเสียงแบบสามมิติรอบทิศทางได้

Direct3D มี API ระดับสูงในการสร้างแอปพลิเคชันที่เกี่ยวกับภาพสามมิติได้อย่างสมบูรณ์และง่าย นอกจากนี้ยังมีการทำงานในระดับต่ำเพื่อการควบคุมการเรนเดอร์ภาพให้ได้ประสิทธิภาพสูงสุด และช่วยให้การย้ายแอปพลิเคชันจากแบบเดิมมาใช้ร่วมกับ DirectX ได้ง่ายยิ่งขึ้น Direct3D ยังสามารถทำงานร่วมกับ DirectDraw ได้อีกด้วย

DirectPlay เป็น API ระดับสูงที่ช่วยให้การเขียนโปรแกรมติดต่อผ่านโมเด็มหรือเน็ตเวิร์ก โดยใช้โปรโตคอลได้หลายชนิด

DirectX 5

การพัฒนาโปรเจ็กชิ้นนี้ในตอนเริ่มต้นเริ่มจากพัฒนาบน DirectX 3 และปัจจุบันได้เปลี่ยนไปใช้เวอร์ชันล่าสุดคือเวอร์ชัน 5 เพื่อให้ทันกับการพัฒนาของ DirectX ที่พัฒนาไปอย่างรวดเร็ว ถึงแม้จะมีการเปลี่ยนเวอร์ชันของ DirectX ก็ไม่มีปัญหากับโปรแกรมที่พัฒนาอยู่เดิม เพราะเวอร์ชันใหม่สามารถทำงานกับโปรแกรมที่เขียนบนเวอร์ชันเดิมได้ เพราะ DirectX ใช้เทคโนโลยี COM ซึ่งเป็น object base system ที่มีความสามารถในการนำอินเตอร์เฟส กลับมาใช้ใหม่ได้ การเปลี่ยนไปเป็นเวอร์ชันใหม่ทำให้ได้รับประสิทธิภาพที่เพิ่มขึ้นได้ทันทีถ้าใช้อินเตอร์เฟสเหมือนเดิม แต่ถ้าความสามารถที่เพิ่มขึ้นมาต้องการอินเตอร์เฟสแบบพิเศษที่เพิ่มขึ้น ก็สามารถเปลี่ยนอินเตอร์เฟสได้ทันที โดยที่ยังคงสามารถใช้อินเตอร์เฟสอันเดิมได้อีกด้วย ความสามารถนี้เหมือนกับการสืบทอด(inherit)ของการเขียนโปรแกรมแบบ Object Oriented

ในเวอร์ชัน 5 มีความสามารถเพิ่มจากเวอร์ชัน3 (DirectX ไม่มีเวอร์ชัน 4 ทางไมโครซอฟท์เปลี่ยนจากเวอร์ชัน 3 ไปเป็นเวอร์ชัน 5 เลย)มีดังนี้

DirectDraw

DirectDraw เพิ่มความสามารถของวิดีโอพอร์ตเพื่อให้นักพัฒนาโปรแกรมสามารถควบคุมการไหลของข้อมูลจากฮาร์ดแวร์วิดีโอพอร์ตไปบน surface ที่เป็นหน่วยความจำในการแสดงผล และเพิ่มความสามารถในการสนับสนุนชุดคำสั่งของ MMX โดยไม่ต้องแก้ไขโปรแกรม

DirectDraw เวอร์ชันใหม่สามารถสร้าง surface ที่กว้างกว่า Primary Surface ได้ตามที่ต้องการถ้าฮาร์ดแวร์สนับสนุน

DirectDraw สนับสนุนสถาปัตยกรรม Advanced Graphics Port (AGP) ซึ่งช่วยในการสร้าง surface ที่เป็น non-local video memory

DirectInput

ในเวอร์ชันใหม่ได้ใช้อินเตอร์เฟสของ COM กับ joystick และยังสนับสนุนอุปกรณ์ตัวใหม่อย่าง

DirectSound

มีอินเตอร์เฟสใหม่เพิ่มขึ้นมาเพื่อสนับสนุนการบริการเสริมที่มีให้ใน soundcard และไดร์เวอร์ และมีความสามารถในการบันทึกเสียงที่เข้ามาแทนบริการเดิมของ Win32 เพื่อที่จะติดต่อกับฮาร์แวร์โดยตรง

DirectPlay

มีอินเตอร์เฟสใหม่ซึ่งจะมีฟังก์ชันการทำงานที่เพิ่มขึ้น เช่นการสนับสนุนการใช้ password สำหรับแต่ละ session ได้ดีกว่าเดิม, สามารถติดต่อกับ secure server, สามารถสร้างออปเจ็กของ DirectPlay ได้หลายออปเจ็กพร้อมๆกัน และความสามารถอื่นอีกหลายอย่าง

Direct3D

ในโหมดการทำงาน Immediate มีฟังก์ชันการทำงานที่เพิ่มขึ้นมาก ส่วนในโหมด Retain สนับสนุนการ interpolator ซึ่งมีความสามารถในการ blend color, การสร้างภาพเคลื่อนไหวระหว่างสองตำแหน่งได้ราบเรียบขึ้น, การ morph และเพิ่มการ transform ในหลายรูปแบบ และการทำ mesh แบบ progressive

Force Feed Back

DirectSetup

ได้เพิ่มความสามารถในการปรับค่าคอนฟิกของผู้ใช้งานได้ดียิ่งขึ้น โดยมี callback function ช่วยในการติดต่อกับสถานะของโปรแกรมที่ติดตั้งอยู่ในปัจจุบันก่อนที่จะมีการติดตั้ง DirectX และไดร์เวอร์ต่างๆ สำหรับโปรแกรม

บทที่ 2

DirectDraw

DirectDraw เป็นคอมโปเน้นที่ถือว่าเป็นหัวใจหลักของ DirectX ทำให้แอปพลิเคชันทางด้านกราฟฟิกสามารถติดต่อกับวิดีโอเมมโมรีและวิดีโอฮาร์ดแวร์ได้โดยตรง แต่เดิมแอปพลิเคชันบนวินโดว์ต้องการใช้ฟังก์ชันเกี่ยวกับการแสดงผลทางกราฟฟิก ต้องติดต่อผ่าน GDI ซึ่งทำงานได้ช้า เพราะไม่ได้เน้นด้านความเร็ว แต่เน้นเสถียรภาพมากกว่า เนื่องจากต้องใช้ร่วมกับหลายแอปพิลิเคชัน แต่แอปพลิเคชันบางประเภทจำพวกเกมส์จำเป็นต้องใช้ความเร็วด้านกราฟฟิกเป็นอย่างมากจึงไม่สามารถใช้บนวินโดว์ได้ วิธีที่จะแสดงกราฟิกได้อย่างรวดเร็ววิธีหนึ่งคือเขียนโปรแกรมให้ติดต่อกับวิดีโอเมมโมรีโดยตรง ซึ่งวิธีนี้เป็นวิธีที่ใช้ในเกมส์บนดอส แต่ไม่เหมาะที่จะใช้บนวินโดว์ เพราะวินโดว์ไม่ให้โปรแกรมสามารถติดต่อกับฮาร์ดแวร์โดยตรง เนื่องจากส่วนที่ทำการแสดงผลจะต้องใช้ร่วมกัน ดังนั้นเราจะต้องติดต่อผ่านไดร์เวอร์ของ DirectX

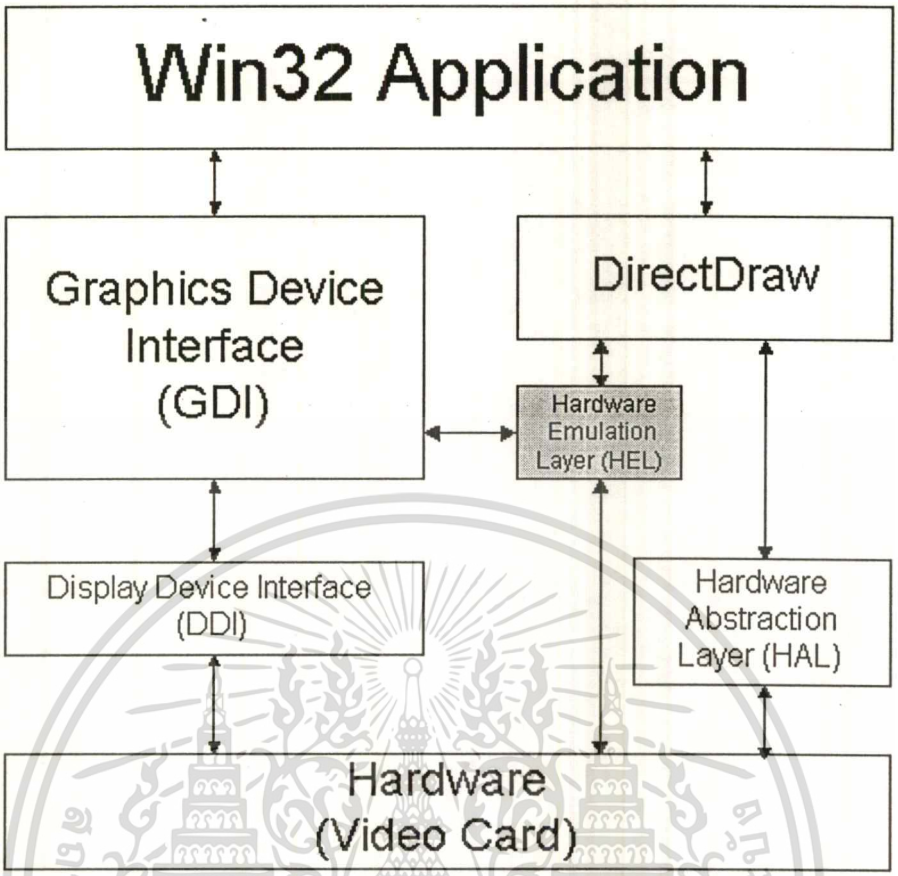
DirectDraw ไม่ใช่ API ระดับสูงสำหรับกราฟฟิกอย่างการลากเส้น เขียนรูปสี่เหลี่ยม แต่เป็น API ระดับต่ำจะติดต่อกับวิดีโอเมมโมรีผ่านโมเดลของหน่วยความจำที่เรียกว่า surface ซึ่งจะมองหน่วยความจำเป็นผืนเดียวกันทำให้จัดการง่าย นอกจากนี้ยังสามารถเข้าถึงความสามารถเฉพาะของการ์ดในแต่ละรุ่นได้ โดยไม่จำเป็นต้องเขียนโปรแกรมเฉพาะการ์ดแสดงผลแต่ละอัน ด้วยการใช้สถาปัตยกรรม HAL/HEL

Hardware Abstraction Layer (HAL) เป็นอินเตอร์เฟสที่ฮาร์ดแวร์สนับสนุนโดยตรง ซึ่งการแสดงผลจะมีประสิทธิภาพสูงสุดเมื่อใช้ฟังก์ชันการทำงานของฮาร์ดแวร์ที่สำหรับทำงานจำเพาะนั้นๆ

Hardware Emulation Layer (HEL) ความสามารถบางอย่างที่ฮาร์แวร์ไม่มี แต่โปรแกรมต้องการใช้ความสามารถนั้น HEL จะทำหน้าที่จำลองการทำงานของฟังก์ชันการทำงานนั้น

ดังนั้นแอปพลิเคชันที่เขียนโดยใช้ DirectX จะมีความเร็วด้านกราฟฟิกเทียบเท่ากับความเร็วของแอปพลิเคชันบนดอส แต่นักพัฒนาไม่จำเป็นต้องเขียนโปรแกรมให้เจาะจงกับฮาร์ดแวร์เหมือนดอส และการพัฒนาแอปพลิเคชันบนวินโดว์สามารถทำได้รวดเร็ว และสามารถใช้ประสิทธิภาพของระบบปฏิบัติการ 32 บิตได้

หลักการ



รูป 2.1 หลักการทำงานของ DirectDraw

จากภาพแสดงให้เห็นว่าการทำงานของ DirectDraw ไม่ผ่าน GDI แต่จะผ่าน HEL และ HAL ซึ่งรายละเอียดมีดังนี้

HAL เป็นอินเตอร์เฟสที่เจาะจงกับฮาร์ดแวร์ของแต่ละผู้ผลิต แต่ส่วนใหญ่เป็นฟังก์ชันมาตรฐานที่จะต้องมีอยู่แล้วในการ์ดแสดงผลทั่วไป ซึ่งเป็นหน้าที่ที่ผู้ผลิตการ์ดแสดงผลนั้นต้องเขียนไดร์เวอร์สำหรับสนุน ผู้ผลิตสามารถทำฟังก์ชันสำหรับ HAL โดยใช้โค้ด16 และ 32 บิตบนวินโดว์ 95 แต่ในวินโดว์ NT ต้องใช้โค้ด 32 บิตเพียงอย่างเดียว ซึ่ง HAL อาจเป็นส่วนหนึ่งของไดร์เวอร์หรือเป็น DLL แยกออกมาก็ได้ ฟังก์ชันกราฟฟิกของ HAL จะต้องเป็นความสามารถโดยตรงของชิปบนการ์ดแสดงผล แต่ถ้าการ์ดแสดงผลไม่สนับสนุนฟังก์ชันนั้นๆ จะมีการรายงานให้กับDirectDraw

HEL เนื่องจากไม่สามารถที่จะทำฟังก์ชันการทำงานให้มีทุกอย่างสำหรับแอปพลิเคชันต้องการได้ ก็มีทางเลือกอย่างหนึ่งก็คือ การจำลองการทำงานของฟังก์ชันนั้นด้วยซอฟท์แวร์ ซึ่งเป็นการเติมฟังก์ชันที่ขาดไป เพื่อให้หนับสนุนฟังก์ชันหลักๆของกราฟฟิกทั้งหมดที่ใช้กันอยู่

ก่อนที่จะมีการจำลองการทำงาน จะต้องมีขบวนการตรวจสอบว่าการ์ดแสดงผลนี้สนับสนุนฟังก์ชันอะไรบ้าง ถ้าพบว่าฟังก์ชันใดไม่สนับสนุนก็จะทำการจำลองการทำงาน การจำลองการทำงานนี้จะต้องใช้

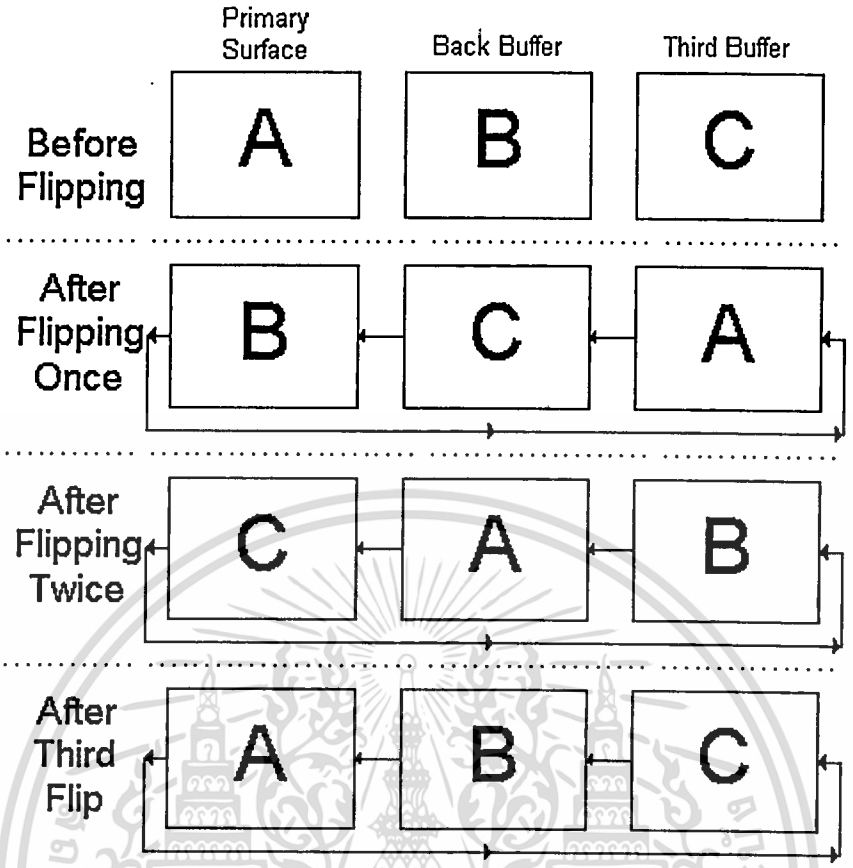
หน่วยประมวลหลัก(CPU) และใช้หน่วยความจำของระบบซึ่งจะทำให้ประสิทธิภาพลดลง และไม่เร็วเท่าการประมวลผลบนชิปและหน่วยความจำของการ์ด

Surface

surface เป็นแนวความคิดที่สำคัญอันหนึ่งบน DirectDraw ใช้หน่วยความจำซึ่งจะมองเป็นผืนเดียวกันเพื่อเก็บภาพสำหรับการแสดงผล โดยไม่ต้องสนใจว่าจะเป็นหน่วยความจำหลัก หรือหน่วยความจำบนการ์ดแสดงผล ถ้าเป็น surface ที่อยู่บนการ์ดแสดงผลจะทำการเข้าถึงได้รวดเร็วกว่าหน่วยความจำของระบบ เนื่องจากสามารถประมวลผลด้วยชิปบนการ์ดแสดงผล พร้อมกับหน่วยประมวลผลหลักทำให้ไม่เสียเวลาในการทำงาน และการย้ายข้อมูลระหว่างวิดีโอเมมโมรี ทำได้รวดเร็วกว่าการย้ายจากหน่วยความจำหลักเข้าวิดีโอเมมโมรีหลายเท่า

Primary Surface เป็นชิ้นของหน่วยความจำบนการ์ดแสดงผลที่เป็นส่วนที่จะแสดงผลออกทางหน้าจอ ซึ่ง surface ชนิดนี้จะมีอยู่อันเดียวเท่านั้น ถ้ามีการแก้ไข surface นี้ก็จะมีผลต่อภาพที่ปรากฏบนหน้าจอทันที

Back Buffer Surface เป็นชิ้นของหน่วยความจำที่มีขนาดเท่ากับ Primary Surface ใช้เป็นบัฟเฟอร์สำหรับการวาดภาพที่จะแสดงผลให้เสร็จเรียบร้อยเสียก่อนที่จะนำไปแสดงผล สามารถมีได้หลาย surface แต่ส่วนใหญ่ใช้ surface เดียวก็เพียงพอ เหตุผลที่ต้องมี Back Buffer ก็คือเพื่อทำให้การแสดงผลแสดงออกมาได้ราบเรียบไม่กระตุก เป็นเทคนิคหนึ่งที่ใช้กันแพร่หลายเรียกว่า Double Buffer ส่วนใหญ่ใช้ในแอปพลิเคชันเกมส์ และ การแสดงกราฟฟิกของมัลติมีเดีย หลังจากที่ทำการวาดลงบน Back Buffer เรียบร้อยแล้วจะทำการ flip (ทำการสลับระหว่าง Back Buffer กับ Primary Surface) การ flip นี้ส่วนใหญ่ฮาร์ดแวร์จะสนับสนุน

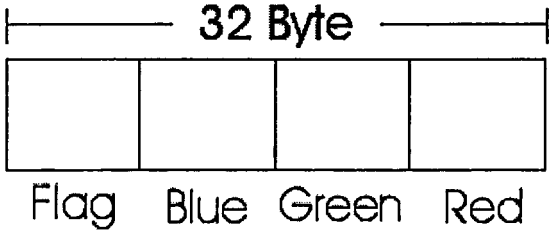


รูปที่ 2.2 การ flip Surface

จากภาพแสดงการใช้สามบัฟเฟอร์ในการ flip แต่ตัวอย่างโปรแกรมของโปรเจ็กนี้จะใช้แค่สองบัฟเฟอร์

Offscreen Surface เป็นชิ้นของหน่วยความจำ สามารถอยู่ได้ทั้งในหน่วยความจำของการ์ดและหน่วยความจำหลักของระบบ มีขนาดเท่าใดก็ได้(surfacr เวอร์ชัน 5 ขึ้นมาสามารถสร้าง surface ได้กว้างกว่า Pirmary Surface แต่ฮาร์ดแวร์ต้อง สนับสนุนด้วย) ส่วนใหญ่ใช้เพื่อการเก็บฉากหรือตัวละคร เพื่อใช้ในการวาดลงบน Back Buffer เนื่องจากมันสามารถอยู่ที่ใดของหน่วยความจำได้ ดังนั้น DirectDraw จะพยายามให้อยู่บนหน่วยความจำของการ์ดแสดงผลเพื่อความเร็ว

Palette เป็น surface ประเภทหนึ่งเหมือนกันทำหน้าที่เก็บตารางของสี ในกรณีที่ภาพที่นำมาแสดงผลเป็นแบบใช้ index ในการเก็บสี(index color) แต่ส่วนใหญ่ใช้เพียงอันเดียวสำหรับ Primary Surface ก็เพียงพอ รูปแบบการเก็บข้อมูลของ palette จะแตกต่างจากการเก็บปกติของRGB เช่น สีแดงบริสุทธิ จะใช้ค่าของ RGB คือ 0,0,255 แต่ค่าที่ปรากฏบน surface จะเป็น 0xFF0000FF



รูปที่ 2.3 การเก็บข้อมูลของ palette บน Surface

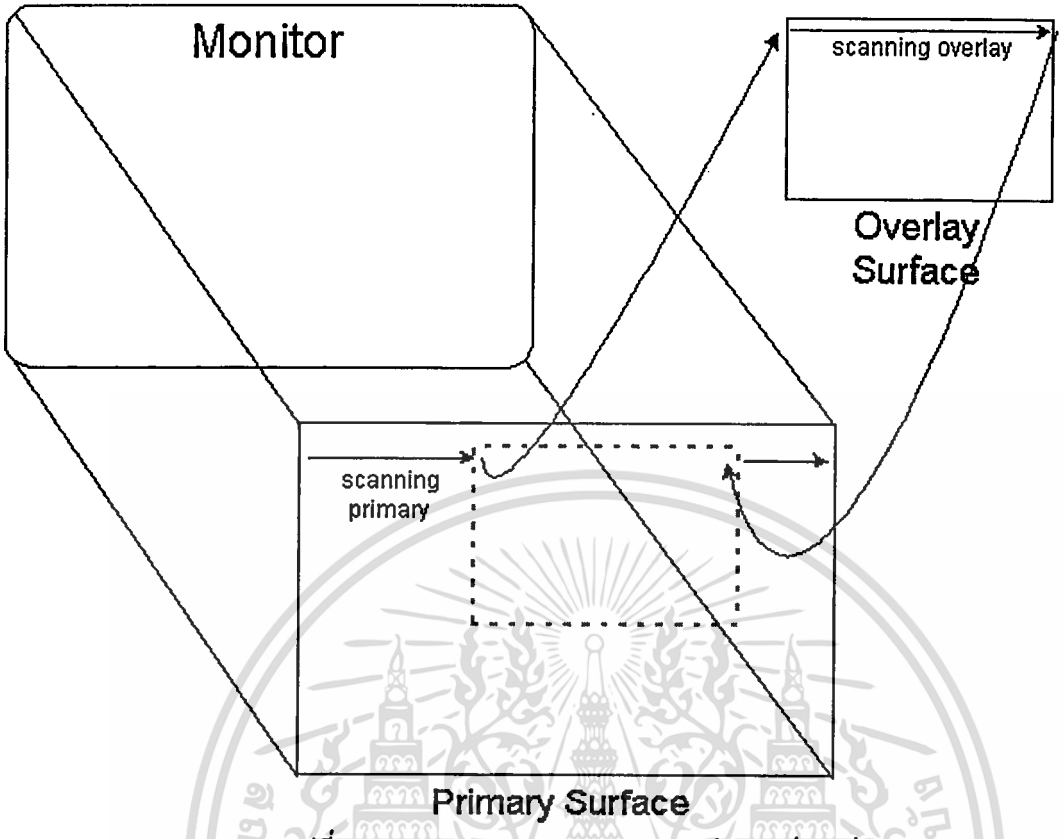
จากภาพ flag จะไม่ได้ใช้งานจะมีค่า 0xFF ตลอดเวลา

Blit

วิธีการในการทำสำเนาข้อมูลใน surface อันหนึ่งไปอีกอันหนึ่ง ซึ่งก็คือการวาดภาพลง surface อีกอันนั่นเอง การ blit สามารถทำการวาดส่วนที่ต้องการจากอีก surface หนึ่งไปยังตำแหน่งที่ต้องการของอีก surface หนึ่ง และสามารถวาดภาพที่มีบางส่วนโปร่งใสได้โดยดูจาก color key ที่กำหนดในขณะทำการ blit ค่าของ color key จะเป็นค่าของสีที่ไม่ถูกแสดงผล(โปร่งใส) เมื่อทำการวาดลงบนอีก surface หนึ่ง ซึ่งทำให้ภาพสามารถแยกออกเป็นวัตถุหรือตัวละครได้

Overlay Surface

เป็นวิธีการในการแสดง Offscreen Surface บน Primary Surface โดยไม่ต้อง blit ไปที่ Primary Surface แต่จำเป็นต้องการการสนับสนุนจากการ์ดแสดงผล DirectDraw ไม่สามารถจำลองการทำงานได้



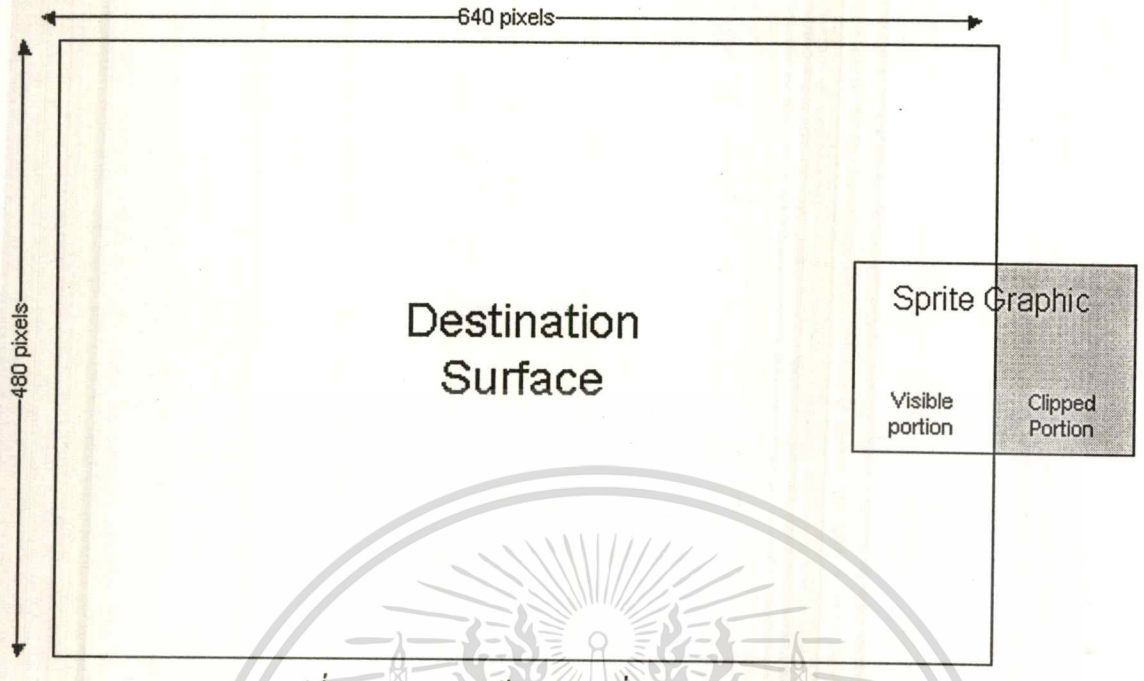
รูปที่ 2.4 แสดงการวาด Overlay Surface โดยฮาร์ดแวร์

จะคล้ายกับว่ามีพลาสติกใสอยู่บนหน้าจอ สามารถวาดภาพบนพลาสติกนั้นได้โดยไม่ทำให้ข้อมูลบนหน้าจอเสียหาย ถ้าไม่ต้องการให้ภาพนั้นปรากฏสามารถนำออกได้ทันที เหมือนกับหยิบแผ่นพลาสติกออก

นอกจากนี้ Overlay Surface ยังมีความสามารถ transparent (โปร่งใส) และ stretching (ย่อขยายในแนวต่างๆ) ได้โดยไม่ต้องเปลี่ยนแปลงข้อมูลของภาพ การประมวลทั้งหมดจะทำบนวิดีโอเมมโมรีและใช้ฮาร์ดแวร์ทำทั้งหมด ดังนั้นการทำงานจะรวดเร็วกว่า Offscreen Surface มาก แต่ต้องเน้นอีกอย่างก็คือรูปแบบการเก็บข้อมูลพิกเซลของ Overlay จะไม่เหมือนกับ surface ทั่วๆไป และจะต้องอยู่บนวิดีโอเมมโมรีเท่านั้น

Clipper

clipper หรือเรียกอีกอย่างว่า DirectDraw Clipper Objects เป็นออปเจ็กหนึ่งของ DirectDraw ที่ทำให้สามารถ blit เฉพาะบางส่วนของ surface ที่ต้องการได้ มีประโยชน์มากในการทำ sprite ส่วนใหญ่ใช้ในการกำหนดขอบเขตไม่ให้ blit เกินขอบเขตของ surface ที่กำหนด อย่างเช่นเราจะทำการวาด sprite หรือ surface ดังภาพ



รูปที่ 2.5 แสดงการใช้ Clipper เพื่อกันการวาดเกินหน้าจอ

แสดงให้เห็นว่าบางส่วนของ sprite เกินหน้าจอ หน้าจอนี้จะใช้ clipper กำหนดขอบเขตของการวาดไม่ให้เกินหน้าจอ ส่วนที่เกินจากนั้นก็จะไม่วาด ถ้ามีการวาดเกินขอบเขตของ surface ปลายทาง จะทำให้เกิด memory access violations หรืออาจจะใช้เพื่อให้วาดเฉพาะส่วนที่ต้องการ ไม่จำเป็นต้องเต็มหน้าจอก็ได้นอกจากนี้เราสามารถมีออปเจ็ก clipper ได้มากกว่าหนึ่งอันในหนึ่ง surface โดยจะใช้ clip list เป็นตัวจัดการ โดยในกลุ่มของสี่เหลี่ยมที่เป็นพื้นที่ที่สามารถมองเห็นได้ไปเก็บไว้เป็น list ซึ่งจะเป็น list ของ surface นั้นๆ

การใช้งาน

ในหัวข้อนี้จะกล่าวถึงรายละเอียดเกี่ยวกับ DirectDraw ในการเริ่มต้นสร้างและการใช้งาน

เริ่มต้นการใช้งาน DirtectDraw หลังจากการสร้างวินโดว์เรียบร้อยแล้ว

1.สร้างออปเจ็กของ DirectDraw

```
LPDIRECTDRAW        lpDD;                   //ประกาศไว้เป็น global
..
..
ddrval = DirectDrawCreate( NULL, &lpDD, NULL ); //สร้างออปเจ็ก DirectDraw
```

2.กำหนดความสามารถในการใช้งานหน้าจอ ในที่นี้จะใช้ EXCLUSIVE ซึ่งมีลำดับความสำคัญในระดับสูง สามารถเปลี่ยนความละเอียด, palette ของหน้าจอได้ และเป็นแบบเต็มหน้าจอ

```
ddrval = lpDD->SetCooperativeLevel( hwnd, DDSCL_EXCLUSIVE |
```

```
                                                    DDSCL_FULLSCREEN );
```

3.กำหนดโหมดของการแสดงผลในที่นี้โหมด 640 x 480 ที่ 8บิตสี

```
ddrval = lpDD->SetDisplayMode( 640, 480, 8);
```

4.สร้าง Primary Surface โดยมี Back Buffer ติดอยู่ด้วย การสร้าง surface ทำคล้ายๆกัน แต่ในกรณีของ Primary, BackBuffer หรือ surface ที่ซับซ้อนต้องมีการกำหนดคุณสมบัติไว้ในโครงสร้างของ ddsd และ ddscaps ซึ่งประกาศไว้ในต้นของฟังก์ชันดังนี้

```
DDSURFACEDESC      ddsd;
DDSCAPS            ddscaps;
```

หลังจากนั้นถึงทำการกำหนดคุณสมบัติพิเศษของ surface

```
/**กำหนดคุณสมบัติของ surface**/
ddsd.dwSize = sizeof( ddsd );
ddsd.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE |
                      DDSCAPS_FLIP |
                      DDSCAPS_COMPLEX;
ddsd.dwBackBufferCount = 1;                    //กำหนดให้มี Buffer หนึ่งอัน
/**สร้าง Primary Surface**/
ddrval = lpDD->CreateSurface( &ddsd, &lpDDSPrimary, NULL );
/***สร้าง Back Buffer **/
ddscaps.dwCaps = DDSCAPS_BACKBUFFER;
ddrval = lpDDSPrimary->GetAttachedSurface(&ddscaps, &lpDDSBack);
```

5.สร้าง palette จากไฟล์รูปภาพและกำหนดให้ Primary Surface ใช้ palette นั้น

```
char szBitmap[] = "all.bmp";                   //ประกาศไว้ใน global
..
..
lpDDPal = DDLoadPalette(lpDD, szBitmap);  //ฟังก์ชันนี้อยู่ใน ddutil.cpp
..
lpDDSPrimary->SetPalette(lpDDPal);
```

6.สร้าง Offscreen Surfaceที่จำเป็นต้องใช้และต้องกำหนด color key หรับ surface ที่เป็นภาพโปร่งใส

```
lpDDSOne = DDLoadBitmap(lpDD, szBitmap, 0, 0);//สร้าง offscreen surface
```

..

DDSetColorKey(lpDDSOne, RGB(0,0,0)); //กำหนด color key ให้กับSurface

การใช้งาน surface

1.การวาดลง Back Buffer โดยจะวาดที่ตำแหน่ง xpos, ypos โดยใช้ภาพจาก lpDDOne เฉพาะในกรอบสี่เหลี่ยมที่กำหนดใน rcRect และใช้ color key ของ lpDDOne

```
ddrval = lpDDSBack->BltFast( xpos, ypos, lpDDSOne,
                        &rcRect, DDBLTFAST_SRCCOLORKEY );
```

2.กรณีที่ข้อมูลใน surface หายไปอาจจะเนื่องจากโปรแกรมอื่นที่มีโหมด exclusive ทำการแก้ไข surface ดังนั้นจะเกิด Surface Lost จึงต้องทำการนำข้อมูลไปใส่ surface ใหม่อีกครั้ง

```
ddrval = lpDDSOne->Restore();
if( ddrval == DD_OK ){
        DDReLoadBitmap(lpDDSOne, szBitmap);
}
```

3.การแสดงผลทำได้โดยใช้คำสั่ง flip ซึ่งจะนำ Primary สลับกับ Back Buffer

```
ddrval = lpDDSPrimary->Flip( NULL, 0 );
```

การคืนหน่วยความจำ

1.ทุก surface ใช้คำสั่งเดียวกันคือ Release

```
lpDDSPrimary->Release();
```

2.หลังจากที่ทำการ Release ทุก surface แล้วก็ให้ทำการ Release ออปเจ็กของ DirctDraw

```
lpDD->Release();
```

บทที่ 3

DirectInput

DirectInput เป็น API สำหรับใช้ในการติดต่ออุปกรณ์ ที่เป็นอินพุตของคอมพิวเตอร์ เช่น เมาส์, คีย์บอร์ด, จอยสติค ฯลฯ ซึ่งความจริงแล้ววินโดว์ก็ทำได้เช่นกัน แต่ของวินโดว์ต่างกับ DirectInput ตรงที่ใช้ message ของวินโดว์ในการจัดการ แต่ในงานที่ต้องการการตอบสนองแบบ real-time อย่างเช่นเกมส์จำเป็นจะต้องอ่านค่าอินพุตในขณะนั้นๆได้ การทำงานของ DirectInput จะใช้โปรแกรมวนลูป polling เพื่ออ่านค่าอินพุต เหมือนกับที่เกมส์บน DOS ทำ แต่มีข้อดีกว่า DOS ก็คือการติดต่อผ่านไดร์เวอร์ของอุปกรณ์อินพุตนั้นทำให้โปรแกรมที่เขียนตามการติดต่อของ DirectInput แล้วก็สามารถใช้ได้กับอุปกรณ์อินพุตชนิดนั้นๆ ไม่ว่าจะเป็นของผู้ผลิตรายใด โดยไม่จำเป็นต้องแก้ไขโปรแกรม

DirectInput สามารถสนับสนุนอุปกรณ์อินพุตเกี่ยวกับเกมส์ที่มีอยู่ในปัจุบันเกือบทั้งหมด นอกจากนี้สามารถสนับสนุนอุปกรณ์ใหม่ๆได้โดยการเปลี่ยนแค่ไดร์เวอร์เท่านั้น อุปกรณ์ที่ DirectInput สนับสนุนมีหลายประเภทเช่น คีย์บอร์ด, จอยสติค, เฮดเกียร์, เมาส์ปกติและแบบหลายปุ่ม ฯลฯ และเทคโนโลยีล่าสุดของอุปกรณ์อินพุตเกมส์ที่สามารถเป็นได้ทั้งอินพุตและเอาต์พุตได้แก่ Force Feed Back ทำให้อุปกรณ์สามารถตอบสนองได้ เช่นในเกมส์แข่งรถเวลาเข้าทางโค้งอุปกรณ์ตัวนี้จะมีแรงต้านของแรงหนีศูนย์กลางเหมือนกับเหตการณ์จริง หรือเกิดแรงสะท้อนเมื่อเกิดการชนกัน

หลักการ

ในการเขียนโปรแกมติดต่อกับอุปกรณ์อินพุตโดยใช้ DirectInput นั้นถ้าเป็นอุปกรณ์คนละประเภทกันจะต้องจัดการต่างกัน เช่น เมาส์ กับคีย์บอร์ด จะเป็นคนละประเภทกัน แต่ถ้าเป็นอุปกรณ์ประเภทเดียวกันแต่ต่างรุ่นต่างผู้ผลิตกันก็ไม่จำเป็นต้องเขียนโปรแกมต่างกันเพราะจะมีการติดต่อมาตรฐานเหมือนกัน ส่วนความสามารถหรือลูกเล่นของอุปกรณ์ในแต่ละรุ่นเราต้องเขียนโปรแกรมจัดการเอง(บางทีไม่จำเป็นเพราะไดร์เวอร์จะจัดการให้เอง)

การจัดการอุปกรณ์ทั้งหมดบน DirectInput ทั้งหมดจะให้การ polling เพื่อใช้ในงาน real-time อุปกรณ์บางอย่างก็สามารถใช้ message ผ่านฟังก์ชันของ DirectInput ได้ แต่การ polling นั้นอาจทำให้สูญเสียข้อมูลบางอย่างได้เมื่อขณะไม่ได้ poll อินพุต แต่ก็เหมาะกับแอปพลิเคชันจำพวกเกมส์ที่ต้องการการตอบสนองอย่างรวดเร็วในขณะที่ต้องการ แต่ก็ไม่เป็นปัญหาอะไรเพราะเราสามารถใช้ทั้ง polling ร่วมกับ message ได้ อุปกรณ์ที่ใช้ในการอินพุตทั่วๆไปมีดังนี้

Keyboard (คีย์บอร์ด) เป็นอุปกรณ์มาตรฐานในการรับอินพุตของคอมพิวเตอร์ทั่วไป ปกติจะทำการอ่านอินพุตจากบัฟเฟอร์ของคีย์บอร์ดซึ่งจะเป็นการเข้าถึงข้อมูลแบบคิว ถ้าตัวอักษรตัวแรกที่เข้าก่อนยังไม่ถูกเอาออก ตัวถัดไปก็ไม่มีทางออกได้เช่นกัน ซึ่งเหมาะกับโปรแกรมจำพวก word processor แต่ข้อเสียก็คือถ้ามีการกดปุ่มค้างเอาไว้หรือกดเร็วและกดหลายตัวอักษรพร้อมๆกันเกินกว่าโปรแกรมจะอ่านทัน บัฟเฟอร์ก็จะเต็ม หรือถ้าต้องการทราบว่าคีย์ที่กด ณ เวลานั้นๆคือคีย์อะไรก็ไม่สามารถทำได้เพราะต้องรอให้เอาตัวอักษรที่อยู่หัวคิวออกก่อน ซึ่งปรากฏการณ์นี้มักเกิดกับโปรแกรมจำพวกเกมส์ต่อสู้ เพื่อให้มีการตอบสนองอย่างรวดเร็ว ดังนั้นเกมส์บนดอสจะใช้วิธีอ่านผ่านพอร์ตของคีย์บอร์ดโดยตรง ไม่ใช้อินเตอร์รัปของดอส ส่วนบนวินโดว์ปกติใช้ message ในการอ่านอินพุตของคีย์บอร์ด แต่ DirctInput สามารถอ่านอินพุตได้โดยตรงจากฮาร์ดแวร์ผ่านไดร์เวอร์ของอุปกรณ์นั้นๆ

ค่าของการอ่านคีย์จะถูกนำมาเก็บไว้ที่อาร์เรย์ของ BYTE ขนาด 256 ช่อง ซึ่งจะแมปเข้ากับค่าสแกนโค้ดซึ่งจะมีการประกาศค่าตำแหน่งของคีย์เรียบร้อยแล้วในไฟล์ dinput.h

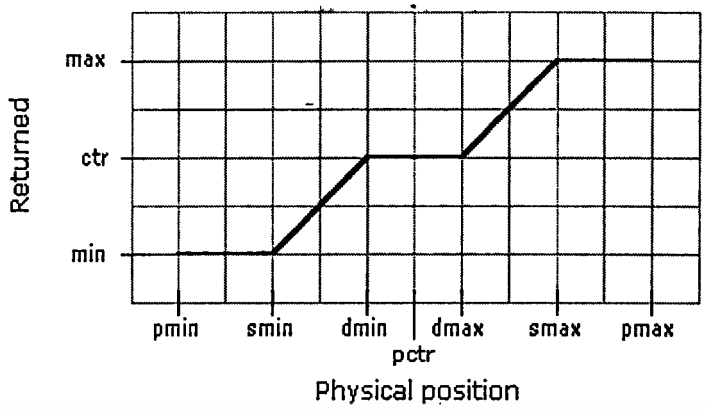
หมายเหตุ เนื่องจากการอ่านค่าจากฮาร์ดแวร์โดยตรงทำให้พฤติกรรมของคีย์บอร์ดเหมือนกับจอยแพดที่มีหลายปุ่ม

Mouse (เมาส์) เป็นอุปกรณ์ที่ใช้แพร่หลายเช่นเดียวกับคีย์บอร์ด ต้องใช้ message ของวินโดว์ เช่นกัน DirectInput ก็ใช้การอ่านค่าอินพุตจากอุปกรณ์โดยตรงเช่นกัน ถ้าใช้ DirectInput ในโหมด exclusive จะทำให้อุปกรณ์ทำงานในโหมดรับ message ทำงานไม่ได้ต้องใช้บริการจาก DirectInput เพียงอย่างเดียว

หมายเหตุ การกำหนดค่าของคีย์บอร์ดและเมาส์ใน Control Panel เช่นปรับความเร็วในการตอบสนอง หรือการสลับปุ่ม จะไม่มีผลกับการทำงานของ DirectInput แต่ถ้าเป็นการปรับค่าผ่านไดร์เวอร์ เช่น เมาส์แบบสามปุ่มกำหนดให้ปุ่มที่สามมีหน้าที่อย่างที่ต้องการ DirectInput จะทำได้เช่นกันโดยอัตโนมัติเพราะ DirectInput ติดต่อผ่านไดร์เวอร์ไม่ใช้วินโดว์ GDI

Joypad (จอยแพด) เป็นอุปกรณ์ที่ใช้ในเกมส์เป็นส่วนใหญ่ จะมีปุ่มรูปกากบาทใช้ในการเคลื่อนที่ได้ 8 ทิศทาง และจะมีปุ่มกดสี่ปุ่มหรือมากกว่านั้น การติดต่อของจอยสติคและจอยแพดจะทำเหมือนๆกัน ดังนั้นโปรแกรมส่วนที่เขียนเพื่อติดต่อกับจอยแพด ก็สามารถใช้กับจอยสติคเช่นกัน

Joystick (จอยสติค) เป็นอุปกรณ์ที่ทำหน้าที่เหมือนกับจอยแพด แต่แทนที่จะเป็นปุ่มกากบาทก็จะเป็นแท่งสำหรับโยกแทน นอกจากนี้จอยสติคยังมีความสามารถมากกว่าจอยแพดตรงที่การโยก จอยสติคสามารถส่งอินพุตแสดงถึงระดับของการโยกได้ว่าโยกมากหรือโยกน้อย เนื่องจากจอยสติคจะใช้ตัวต้านทานปรับค่าได้ต่อกับคันโยก การโยกในมุมที่ต่างๆกันจะทำให้จอยสติคส่งค่า Voltage ออกมาไม่เหมือนกัน



รูปที่ 3.1 ตำแหน่งทางกายภาพ และการส่งค่าอินพุตของ จอยสติค

การบังคับการเคลื่อนไหวสามารถทำได้หลายระดับเหมาะกับเกมส์พวกเครื่องบินหรือรถแข่ง แต่จอยสติคแบบนี้จำเป็นต้องการการประมวลผลมาก เพราะต้องคอยติดตามค่าที่เป็นอนาล็อกอยู่ตลอดเวลา และค่าแบบอนาล็อกมักไม่ค่อยแน่นอน ดังนั้นจำเป็นต้องมีการกำหนดช่วงของการเคลื่อนที่ โดยจากภาพจะต้องกำหนดค่าสูงสุดและต่ำสุดjoy เคลื่อนที่ได้ในแนวแกน x และ y (ปกติอยู่ในช่วง 0 - 10,000 ซึ่งจะทำการเลื่อนให้อยู่ในช่วง ประมาณ -5,000 ถึง 5,000) และต้องมีการกำหนดช่วงที่จะเป็นจุดศูนย์กลาง(dead zone) จากภาพอยู่ในช่วง dmin - dmax ถ้าค่าอยู่ในช่วงนี้ก็ให้ถือว่าอยู่จุดศูนย์กลาง นอกจากนี้ยังมีการกำหนดช่วงอิ่มตัวจากภาพอยู่ในช่วง pmin- smin (upper saturation zones) และ smax-pmax(upper saturation zones) ถ้าการเคลื่อนที่ทางกายภาพถึงช่วงนี้แล้วก็จะมีค่าไม่เกิน pmin หรือ pmax

Force Feed Back เป็นนวกรรมใหม่สำหรับอุปกรณ์ทางด้านเกมส์ที่สามารถเป็นอุปกรณ์เอาต์พุตได้ เพื่อเพิ่มความสนุกในการเล่นเกมส์ยิ่งขึ้น เพราะอุปกรณ์ที่ใช้บังคับสามารถตอบสนองกับผู้เล่นได้ สามารถสร้างแรงต้านได้หลายแบบ อย่างเช่นทำให้เกิดแรกต้านเหมือนเกิดแรงเสียดทานเวลาเคลื่อนไหว หรือเหมือนกับการยืดของสปริงก็ได้ หรือแรงต้านแบบคงที่ก็ได้ นักพัฒนาสามารถควบคุมการทำงานของอุปกรณ์นี้ได้ เพราะว่าเป็นอุปกรณ์เอาต์พุตด้วย ภายในโปรเจ็กนี้จะไม่ขอกล่าวถึงอุปกรณ์ตัวนี้มากนัก เพราะว่าเป็นราคาแพงหาได้ยากโปรแกรมที่ใช้งานอุปกรณ์ตัวนี้ก็ยังไม่ค่อยมี ดังนั้นจึงไม่ได้ทำการศึกษาความการใช้งานอุปกรณ์ตัวนี้

การใช้งาน

ในหัวข้อนี้จะกล่าวถึงการใช้งานของอุปกรณ์อินพุตสองตัวซึ่งใช้ในการทำโปรเจ็กชิ้นนี้ คือ คีย์บอร์ดกับจอยสติค ในส่วนของการสร้างและการเรียกใช้งาน

เริ่มการใช้งานกับคีย์บอร์ด

1.สร้างออปเจ็กของ DirectInput ซึ่งประกาศไว้ในไฟล์ dinputx.h และสร้างในไฟล์ dinputx.cpp

```
LPDIRECTINPUT       lpDI;       //DriectInput Object  //ใน dinputx.h
```

```
..
..
init_value = DirectInputCreate(_hInstance,0x0300, &lpDI, NULL) //ใน dinputx.cpp
```

ในที่นี้ 0x0300 หมายถึงเวอร์ชันของการอินเตอร์เฟสของคีย์บอร์ดจะเป็นเวอร์ชัน 3 ที่ไม่ใช้เวอร์ชัน 5 เพื่อให้โปรแกรมเข้ากับไดร์เวอร์ของเวอร์ชั่น 3 และ 5 ได้ เพราะตอนที่กำลังทดสอบทำการทดสอบในวินโดว์ NT ซึ่งขณะนั้น DirectX เวอร์ชัน 5 มีเฉพาะในวินโดว์ 95 วินโดว์ NT มีแต่เวอร์ชัน 3

2.สร้างออปเจ็กคีย์บอร์ด (lpKeyboard)

```
init_value = lpDI->CreateDevice(GUID_SysKeyboard, &lpKeyboard, NULL);
```

3.กำหนดรูปแบบการเก็บข้อมูลของคีย์บอร์ด

```
init_value = lpKeyboard->SetDataFormat(&c_dfDIKeyboard);
```

จากโค้ดจะใช้รูปแบบโครงสร้างมาตรฐานที่กำหนดไว้ใน dinput.h

4.กำหนดพฤติกรรมของคีย์บอร์ดก่อนที่จะใช้งาน

```
init_value = lpKeyboard->SetCooperativeLevel(_hwnd,
                    DISCL_NONEXCLUSIVE | DISCL_FOREGROUND);
```

ในที่นี้เป็น none exclusive และ foreground ซึ่งหมายความว่าทำงานโดยที่สามารถใช้ message ของวินโดว์ได้อยู่ และทำงานเมื่อแอปพลิเคชัน active อยู่

5.เริ่มต้นการใช้งานคีย์บอร์ด(enable)

```
lpKeyboard->Acquire();
```

การอ่านค่าจากคีย์บอร์ด

1.อ่านค่าของคีย์บอร์ดขณะนั้นไปเก็บไว้ที่อาร์เรย์ขนาด 256 ช่องโดยที่ 1 ช่องมีขนาด 1 BYTE

```
init_value = lpKeyboard->GetDeviceState(sizeof(diks), &diks);
```

2.ตรวจสอบค่าคีย์ที่กด จะใช้ฟังก์ชันเป็นตัวถามว่าคีย์ที่ต้องการทราบกดหรือเปล่า จะได้ผลลัพธ์เป็นจริงถ้าปุ่มนั้นกด ไม่เช่นนั้นจะเป็นเท็จ ต้องทำการถามหลังจากอ่านค่าด้วยวิธีในข้อ 1. แล้วเท่านั้น

```
#define KEYDOWN(name,key) (name[key] & 0x80)
..
..
BOOL keyStatus(BYTE scancode)
{
        return (KEYDOWN(diks,scancode )) ;
}
```

เริ่มการใช้งานกับจอยสติค

1.สร้างออปเจ็กของDirectInput ซึ่งประกาศไว้ในไฟล์ dinputx.h และสร้างในไฟล์ dinputx.cpp

```
LPDIRECTINPUT        lpDI;          //DriectInput Object  //ใน dinput.h
..
..
init_value  = DirectInputCreate(_hInstance,DIRECTINPUT_VERSION, &lpDI, NULL);
```

2.สร้างออปเจ็กจอยสติค (&lpJoyStick)

```
init_value  =  lpDI->CreateDevice(GUID_Joystick, &lpJoyStick, NULL);
```

3.กำหนดรูปแบบการเก็บข้อมูลของคีย์บอร์ด

```
init_value = lpJoyStick->SetDataFormat(&c_dfDIJoystick);
```

4.กำหนดพฤติกรรมของคีย์บอร์ดก่อนที่จะใช้งาน

```
init_value = lpJoyStick->SetCooperativeLevel(_hwnd,
        DISCL_NONEXCLUSIVE | DISCL_FOREGROUND);
```

5.กำหนดสมบัติของจอยสติคทั้งในแนวแกน x และ y

```
init_value = lpJoyStick->SetProperty(DIPROP_RANGE, &diprg.diph);
 init_value = lpJoyStick->SetProperty(DIPROP_DEADZONE, &dipdw.diph) ;
```

6.ทำอินเตอร์เฟสใหม่ให้กับออปเจ็กจอยสติคให้เป็นเวอร์ชัน 2 เพื่อให้มีความสามารถ polling (ในเวอร์ชันเดิมใช้ message)

```
init_value = lpJoyStick->QueryInterface(IID_IDirectInputDevice2,
            (LPVOID*)&lpJoyStickX);
```

อินเตอร์เฟสใหม่จะใช้ lpJoyStickX

7.เริ่มต้นการใช้งาน จอยสติค (enable)

```
lpJoyStickX->Acquire();
```

การอ่านค่าจากจอยสติค

1.ทำการ polling จอยสติค

```
init_value = lpJoyStickX->Poll();
```

2.ทำการนำค่าที่ polling ได้ไปเก็บไว้ในโครงสร้าง js ซึ่งเป็นค่าปุ่มที่กดของจอยสติค

```
init_value = lpJoyStickX->GetDeviceState(sizeof(DIJOYSTATE),&js);
```

การใช้งานสามารถอ่านค่าปุ่มที่กดได้โดยตรงจากstructure js

บทที่ 4

Direct3D

Direct3D เป็น API สำหรับกราฟฟิกสามมิติบนวินโดว์(ทั้งวินโดว์95 และNT) แบบ real-time ซึ่งทางไมโครซอฟท์ได้พัฒนาออกมา เพื่อเกมส์และมัลติมีเดียโดยเฉพาะ ประโยชน์ที่ได้จาก Direct3D ไม่เพียงแค่ความเร็วที่พอๆกับดอสเท่านั้น โปรแกรมสามารถดึงประสิทธิภาพของการ์ดเร่งสามมิติทั้งที่เป็นฟังก์ชันการทำงานทั่วไป(มาตรฐาน)และฟังก์ชันการทำงานเฉพาะของการ์ดแสดงผลนั้นๆ ออกมาได้โดยไม่จำเป็นต้องทำการแก้ไขโปรแกรม และ Direct3D ก็กำลังจะกลายเป็นมาตฐานของการเขียนโปรแกรม 3 มิติเพราะจะถูกผนวกรวมกับวินโดว์ 95 และ NT เวอร์ชันต่อๆไป

Direct3D สามารถทำงานร่วมกับคอมโปเน้นอื่นๆได้เป็นอย่างดี และจะทำงานใกล้ชิดกับ DirectDraw เป็นพิเศษ เพื่อทำให้การแสดงผลเป็นไปอย่างรวดเร็ว Direct3D มี 2โหมดในการใช้งาน Immediate Mode และ Retained Mode

Retain Mode จะเป็น API ในการพัฒนาแอปพลิเคชันกราฟฟิกสามมิติในระดับสูง ซึ่งจะมีความสามารถในการทำอนิเมชัน และการจัดความสัมพันธ์ของออปเจ็กสามมิติเป็นระดับชั้น(hierarchies) ได้อย่างง่าย ไม่ต้องคำนึงการทำงานในระดับต่ำมากนัก

Immediate Mode จะเป็น API ใช้สำหรับการพัฒนาแอปพลิเคชันสามมิติในระดับต่ำ เหมาะสำหรับนักพัฒนาที่ต้องการย้ายแอปพลิเคชันทางมัลติมีเดียหรือเกมส์เข้ามาสู่ระบบปฏิบัติการวินโดว์ของไมโครซอฟท์ การทำงานจะค่อนข้างซับซ้อน การทำงานหลายอย่างนักพัฒนาจำเป็นต้องจัดการเองเกือบทั้งหมด

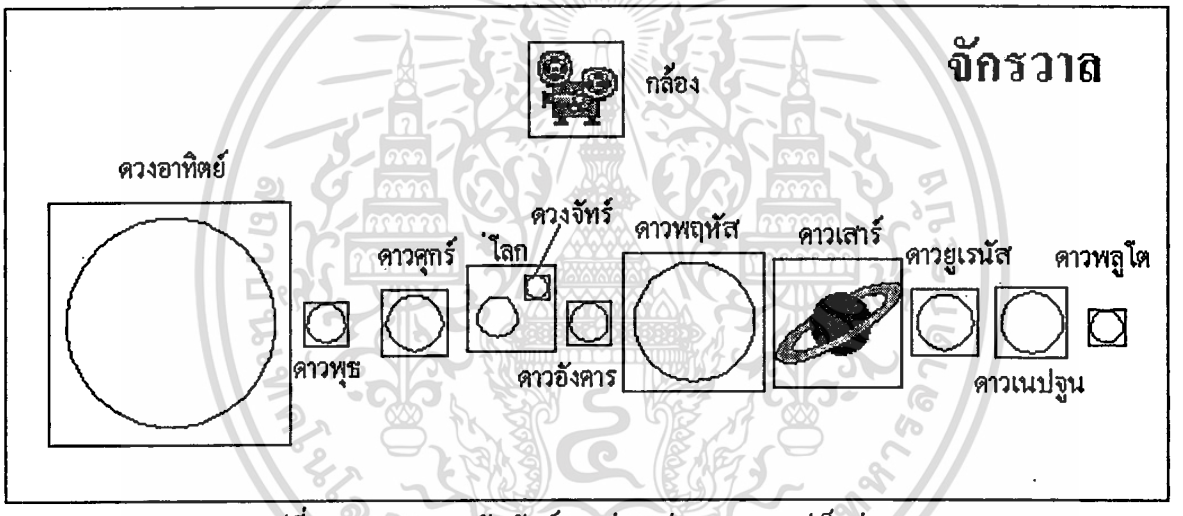
ภายในโปรเจ็กชิ้นนี้จะกล่าวถึงการทำงานในโหมด Retain เพียงอย่างเดียวเพราะในโหมด Immediate จะค่อนข้างซับซ้อนมากเกินขอบเขตของโปรเจ็ก

หลักการ

ในการเขียนโปรแกรมเกี่ยวกับงานภาพสามมิตินั้นจำเป็นต้องมีพื้นฐานทางคณิตศาสตร์ด้านเวกเตอร์ทั้งสองและสามมิติพอสมควร ความรู้พื้นฐานเกี่ยวกับการทำงานในระบบที่เป็นสามมิติสามารถดูรายละเอียดได้จากภาคผนวกภายในโปรเจ็กชิ้นนี้ ใน Direct3D จะใช้กฎมือซ้ายในการทำงานในระบบ coordinate X Y Z ออปเจ็กสามมิติมีการกระทำการทำงานพื้นฐาน คือ Translation (ย้ายตำแหน่ง), Rotation (หมุน) และ Scaling (ย่อขยายขนาด) มีแสงเพื่อที่จะทำให้เห็นออปเจ็กสามมิติได้ และการแสดงผลจะมองผ่านวิวพอร์ตออปเจ็กทุกอย่างที่มีจะต้องมีการอ้างอิงตำแหน่งโดยจะใช้เฟรมเป็นตัวเทียบ นอกจากนี้จะในแต่ละเฟรมจะมีความสัมพันธ์กันเองได้และจัดระดับชั้นของความสัมพันธ์ในรูปของแผนภาพต้นไม้

Frame เป็นสิ่งที่ใช้ในการอ้างวัตถุหรือกลุ่มของวัตถุ โดยจะมีสภาพแวดล้อมเป็นของมันเอง ตำแหน่งของวัตถุจะอ้างอิงเทียบกับเฟรมที่อยู่ เฟรมแต่ละเฟรมสามารถอ้างอิงตำแหน่งเทียบกับอีกเฟรมได้ โดยสามารถจัดเป็นระดับชั้นของความสัมพันธ์ในลักษณะของแผนภาพต้นไม้หัวกลับ (hierarchies) เช่นการจำลองระบบสุริยะจักรวาล จะต้องมีเฟรมสุริยะจักวาลเป็นเฟรมแม่ของทุกสิ่ง และระบบสุริยะจักรวาลจะต้องมีเฟรมดวงอาทิตย์ อยู่ในในเฟรมแม่ และยังมีเฟรมย่อยๆของดาวเคราะต่างได้แก่ เฟรมของระบบดาวพุธ, ระบบดาวศุกร์, ของระบบดาวโลก ฯลฯ อยู่ภายใน และภายใน เฟรมของระบบดาวโลกจะประกอบด้วยโลก และเฟรมของดวงจันทร์ ภายในเฟรมของดวงจัทร์ก็จะมีดวงจันทร์อยู่

ดังนั้นดวงจันทร์จะโคจรรอบโลกก็คือการเคลื่อนเฟรมของดวงจันทร์หมุนรอบโลก แต่จะไม่มีผลอะไรเลยกับโลก แต่ถ้าต้องการให้โลกหมุนรอบดวงอาทิตย์ก็จะต้องเคลื่อนเฟรมของระบบโลกให้หมุนรอบดวงอาทิตย์ทั้งโลก และเฟรมของดวงจันทร์เป็นเฟรมลูกของระบบโลกก็จะหมุนรอบดวงอาทิตย์ไปด้วย ซึ่งแต่ละเฟรมจะมีพิกัดและวงโครจรเป็นของตัวเอง



รูปที่ 4.1 แสดงความสัมพันธ์ระหว่าง เฟรม และ ออปเจ็กต่างๆ

จากภาพจะไม่ได้แสดงถึงดาวบริวารของเคราะห์อื่นๆนอกจากโลก

Light (แสง) ที่ใช้งานจะมีหลายประเภท

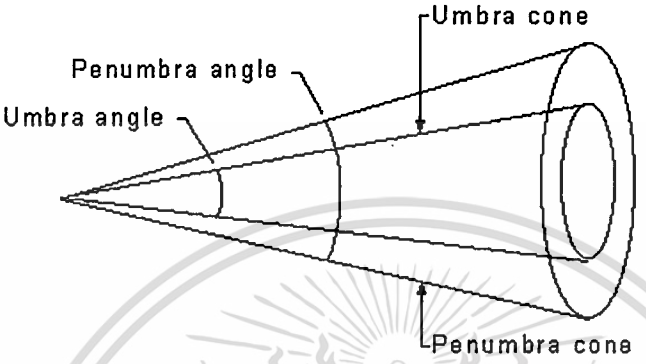
Ambient เป็นแสงที่ไม่สามารถระบุจุดกำเนิดแสดงได้ เหมือนแสงภายในอาคารที่ทุกอย่างภายในสภาพแวดล้อมนั้นสว่างเท่ากันหมด และจะไม่เกิดเงาขึ้นเพราะสว่างเท่ากัน

Directional เป็นแสงที่มีทิศทางเดียวแต่ไม่สามารถระบุระยะห่างของจุดกำเนิดได้ ให้ถือว่าแหล่งกำเนิดแสงไกลมาก เหมาะกับการเรนเดอร์ ที่ต้องการความเร็วสูง

Parallel Point มีลักษณะการส่องสว่างคล้ายแบบ Dierctional แต่จะมีตำแหน่งกำเนิดของแสง เช่นถ้าจุดกำเนิดของแสดงอยู่ระหว่างวัตถุสองอัน จะเกิดแสงขนานเช่นเดียวกับแบบ Directional แต่จะอยู่คนละทิศทาง

Point เป็นแสงที่มีจุดกำเนิดเป็นจุด เหมือนกับการส่องสว่างของหลอดไฟ ที่แสงส่องไปทุกทิศทางจากจุดกำเนิด

Spotlight เป็นแสงที่สองออกมาในลักษณะของกรวยเหมือนส่องไฟฉาย แต่จะมีความพิเศษตรงที่จะมีความสว่างสองส่วน คือบริเวณตรงกลางจะมีความเข้มสูงกว่ารอบนอก



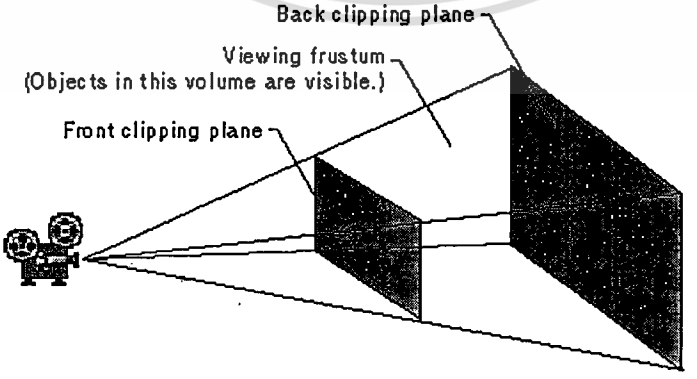
รูปที่ 4.2 แสดงการส่องแสงแบบกรวย โดยจะมีเงามืดเงามัวให้เห็น

Mesh เปรียบเสมือนตัวแทนของวัตถุสามมิติ(visual object) ซึ่งจะประกอบไปด้วยเซ็ตของ vertic(จุด) และ face(พื้นผิว) ที่จะใช้อธิบายลักษณะของวัตถุสามมิตินั้นๆ

Texture เป็นอาร์เรย์ของจุดสีที่จะนำไปแมปกับพื้นผิวของตาข่าย ส่วนใหญ่เราสร้างมาจากภาพบิตแมป

ViewPort เป็นส่วนที่ใช้ในการกำหนดว่าภาพสามมิติจะถูกเรนเดอร์ไปเป็นภาพสองมิติได้อย่างไร ซึ่งถูกใช้เป็นช่องสำหรับมองภาพ(ที่จะแสดงผล) ซึ่งวิวพอร์ตจะใช้เฟรมในการอ้างอิง และวิวพอร์ตจะทำการเรนเดอร์เฉพาะส่วนที่มันรับภาพอยู่ ซึ่งจะมองภาพตามแนวแกน Z และใช้แนวแกน Y แทนทิศชี้ขึ้นข้างบน

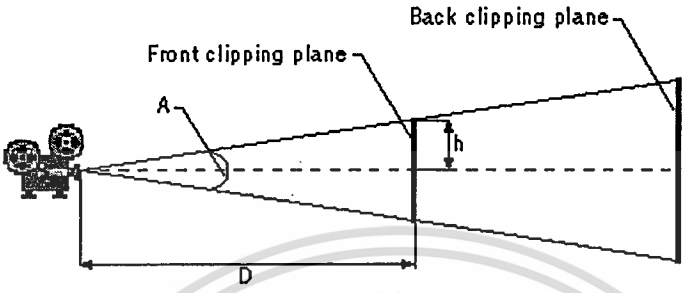
Viewing Frustum กำหนดระยะในการมองออปเจ็กสามมิติ ซึ่งพิกัดในการมองจะอยู่ในรูปของปิรามิดฐานสี่เหลี่ยม และระยะของการมองเห็นจะอยู่ในช่วง ของระนาบสองอันที่เรียกว่า clipping planes



รูปที่ 4.3 แสดงให้เห็นขอบเขตของการ เรนเดอร์

จะมีการเรนเดอร์วัตถุภายในช่วงนี้เท่านั้น และจำเป็นต้องกำหนดความกว้างของมุมมองภาพไว้ด้วยซึ่งสามารถหาความสัมพันธ์ระหว่าง Front clipping และ Back clipping ได้จากสูตร

$$A = 2\tan^{-1}\frac{h}{D}$$



รูปที่ 4.4 แสดงให้เห็นความสัมพันธ์ของ front และ back clipping

นอกจากนี้มันจะเป็นตัวกำหนดความกว้าง

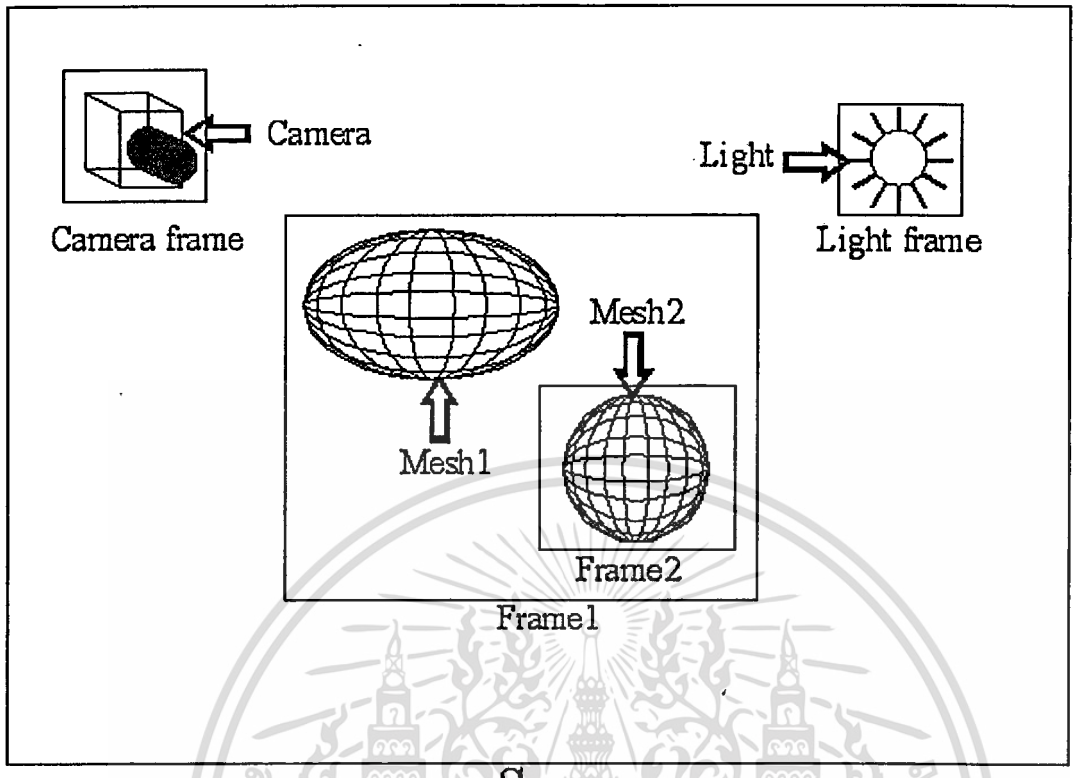
Transformations การเรนเดอร์ภาพสามมิติบนพื้นผิวสองมิติจำเป็นต้องมีการคำนวนภาพฉาย (projection) ซึ่งจะใช้พิกัด [x,y,z,w]ในการแปลงพิกัดไปเป็น [x/w y/w z/w] โดยที่ [x/w y/w] จะเป็นพิกัดที่ปรากฏบนวินโดว์ และ z/w จะเป็นความลึก ซึ่งจะมีช่วงระหว่าง 0 ถึง 1 นับจาก front clipping ถึง back clipping

$$P = \begin{bmatrix} D/hF & 0 & 0 & 0 \\ 0 & D/hF & 0 & 0 \\ 0 & 0 & F/(F-D) & 1 \\ 0 & 0 & (-F*D)/(F-D) & 0 \end{bmatrix}$$

รูปที่ 4.5 สูตรของการคำนวณภาพฉายที่จะปรากฏบนระนาบสองมิติ

จากสูตรจะเป็นสูตรของ projection matrix โดยที่ h เป็นระยะความสูงครึ่งหนึ่งของ viewing frustum, F เป็นระยะทางนับจากกล้อง และ D เป็นตำแหน่งใน Z-coordinate ของ front clipping plane

Picking เป็นความสามารถในการหา visual (สิ่งที่เราใช้แทนวัตถุสามมิติ) จาก 2-D coordinate ในวินโดว์ของวิวพอร์ตโดยจะเริ่มหาจากวัตถุที่ใกล้ที่สุด



รูปที่ 4.6 แสดงภาพโดยรวมของ Direct3D Retain Mode

การใช้งาน

เริ่มต้นการใช้งาน Direct3D (Retain Mode) ตัวอย่างตัดตอนมาจาก hello_world.cpp **ในฟังก์ชัน** InitApp( ) **จะแสดงเฉพาะโครงสร้างที่จำเป็นเท่านั้นรายละเอียดสามารถค้นหาได้จากไฟล์** hello_world.cpp

1. initial windows ทำเหมือนปกติในตัวอย่างจะเป็นโหมดวินโดว์
2. ค้นหาไดร์เวอร์สามมิติที่การ์ดมีให้
3. ทำการสร้างออปเจ็กของ Direct3D

```
LPDIRECT3DRM        lpD3DRM;        // Direct3DRM object (ประกาศไว้ที่ global)
...
...
lpD3DRM = NULL;
Direct3DRMCreate(&lpD3DRM);
```

4. สร้างเฟรมหลักที่เป็นต้นกำเนิดของทุกเฟรม (Scene) และเฟรมของกล้องที่จะใช้มองภาพเพื่อแสดงผลโดยให้เฟรมของกล้องเป็นลูกของเฟรม Scene

```
lpD3DRM->CreateFrame( NULL, &myglobs.scene);        //สร้างเฟรมหลัก
```

```
lpD3DRM->CreateFrame( myglobs.scene,&myglobs.camera); //สร้างเฟรมของกล้อง
myglobs.camera->SetPosition( myglobs.scene, D3DVAL(0.0),D3DVAL(0.0),
        D3DVAL(0.0));                    //กำหนดตำแหน่งของเฟรมกล้องในscene
```

5. สร้างคลิปเปอร์สำหรับวินโดว์ (หลักการเดียวกับ DirectDraw ดูรายละเอียดใน DirectDraw)

```
DirectDrawCreateClipper(0, &lpDDClipper, NULL);
lpDDClipper->SetHWnd( 0, win);
```

6. สร้างดีไวซ์จากไดร์เวอร์ที่หามาได้จาก EnumDrivers() ภายในฟังก์ชันนี้จะทำการสร้างส่วนของการแสดงผลแล้วจะนำไปติดกับเฟรมของกล้อง

```
GetClientRect(win, &rc);
CreateDevAndView(lpDDClipper, myglobs.CurrDriver, rc.right, rc.bottom))
```

7.สร้างเฟรมที่จะใช้ในโปรแกรม โดยให้สืบทอดจาก scene โดยทำในลักษณะเดียวกับเฟรมของกล้อง การสร้างเฟรมทำได้โดยสร้างจากออปเจ็กของ Direct3D Retain Mode โดยทำดังนี้

```
lpD3DRM->CreateFrame(lpScene, lplpFrame1);
```

จะได้ออปเจ็ก Frame1 ซึ่งเป็นลูก ของเฟรม Scene ภายในเฟรมที่สร้างนี้สามารถนำวัตถุเช่น แสง โพลิกอนสามมิติ หรือเฟรมใดๆ ไปอ้างอิงกับมันได้ในกรณีของ

แสง(light) ต้องสร้างแสงขึ้นมาก่อนโดยสร้างจากออปเจ็กของ Direct3D Retain Mode โดยทำดังนี้

```
lpD3DRM->CreateLightRGB( D3DRMLIGHT_DIRECTIONAL,
        D3DVAL(0.9), D3DVAL(0.9), D3DVAL(0.9), lplpLight1);
```

ในตัวอย่างจะใช้แสงในโหมด RGB และเป็นแบบ Directional ค่าในD3DVAL( )ทั้งสามตัวจะเป็นค่าของสีแดง เขียว และน้ำเงิน จะได้ออปเจ็กของแสงคือ lplpLight1

หลังจากสร้างเรียบร้อยแล้วต้องนำไปอ้างอิงกับเฟรมที่ต้องการในตัวอย่างจะนำแสงไปอ้างอิงกับ Frame1

```
lpLightFrame1->AddLight( *lplpLight1);
```

โพลิกอนสามมิติ จะต้องสร้าง mesh builder ขึ้นมาก่อน จะเป็นตัวแทนของโพลิกอนและค่าสมบัติต่างๆของโพลิกอน

```
lpD3DRM->CreateMeshBuilder( lplpSphere3_builder);
```

หลังจากขั้นตอนนี้จะต้องทำการโหลดโพลิกอนสามมิติ, กำหนดขนาด, กำหนดสีของพื้นผิว และจะต้องสร้าง Wrap ซึ่งเป็นตัวกำหนดวิธีในการ map texture เข้ากับโพลิกอนว่าจะใช้พิกัดระนาบ วงกลม หรือทรงกระบอกในการ mapping แล้วจะนำสมบัติที่กำหนดไปให้กับ mesh builder หลังจากนั้นจะทำการโหลด texture ให้กับ mesh builder

Control ทำการควบคุมการเคลื่อนไหวของโพลิกอนสามมิติหรือกล้อง ทำได้โดยการควบคุมผ่านเฟรมที่อ้างอิงอยู่ เช่นถ้าเคลื่อนไหวหรือหมุนเฟรม วัตถุต่างๆที่อ้างอิงกับเฟรมก็จะทำกิริยานั้นไปด้วย สามารถควบคุมเฟรมได้ในสองลักษณะใหญ่ๆก็คือ แบบอัตโนมัติ และแบบกำหนดเอง

แบบอัตโนมัติ จะใช้การเคลื่อนไหวแบบไม่ต้องการการควบคุมจากผู้ใช้งานจะทำได้โดยโปรแกรม เช่น ต้องการให้เฟรมหมุนก็ทำได้โดยกำหนดมุมในการหมุน หลังจากนั้นเฟรมจะทำการหมุนเองเมื่อถึงเวลาที่ต้องเปลี่ยนสถานะเฟรมนั้นๆ(ด้วยคำสั่ง Tick หรือ Move) หรืออาจใช้ callback function ช่วยในการทำ

แบบกำหนดเอง เหมาะกับการที่ต้องการการควบคุมจากผู้ใช้ เช่นต้องรออ่านค่าจากคีย์บอร์ด เหมาะกับการกำหนดค่าตำแหน่งหรือทิศทางของวัตถุในขณะนั้น คำสั่ง Tick หรือ Move ไม่มีผลต่อการเคลื่อนที่ เพราะไม่ได้ตั้งค่าที่เป็นแบบอัตโนมัติไว้

หมายเหตุ floating point ใน Direct3D จะใช้ขนาด 53 บิต ถ้าโปรแกรมมีการเปลี่ยนค่า precision ก็ควรจะเปลี่ยนกลับให้เหมือนเดิมด้วย

บทที่ 5

DirectSound

DirectSound ช่วยในการติดต่อกับอุปกรณ์กำเนิดเสียง (sound card) โดยไม่ต้องสนใจถึงชนิดของการ์ดเสียง มีความสามารถในการเล่นและผสมเสียงจากหลายช่องสัญญาณ (channel)โดยกินกำลังในการประมวลผลของซีพียูต่ำ ช่วยในการปรับความดังของเสียงได้อิสระ และที่สำคัญคือความล่าช้าของการเล่นเสียงมีน้อย(low-latency playback) หลังจากส่งคำสั่งไปเพื่อสั่งให้เล่นเสียงก็เล่นได้ทันที ในการทำงาน DirectSound เป็นระดับล่างสามารถติดต่อกับฮาร์ดแวร์ได้โดยตรงอย่างเช่นความสามารถในการนำข้อมูลจากไฟล์ไปไว้ในบัฟเฟอร์ก่อนการเล่นเสียง ซึ่งบัฟเฟอร์สามารถเป็นหน่วยความจำของการ์ดเองหรืออาจจะจำลองบัฟเฟอร์จากหน่วยความจำหลักก็ได้

DirectSound เป็น Application Programming Interface (API) เป็นส่วนหนึ่งของ DirectX ที่จัดการด้านเสียง ช่วยในการผสมเสียง ติดต่อกับอุปกรณ์เสียงโดยตรงและยังสามารถใช้กับไดร์เวอร์ของอุปกรณ์เสียงที่มีอยู่แล้วได้ด้วย นอกจากนี้การติดต่อกับอุปกรณ์เสียงโดยผ่าน DirectSound ช่วยให้การเขียนโปรแกรมไม่ขึ้นกับชนิดหรือผู้ผลิตอุปกรณ์เสียง

การปรับค่าคอนฟิกของอุปกรณ์สามารถทำได้โดยอัตโนมัติและมีประสิทธิภาพสูงสุด DirectSound ยังสนับสนุนความสามารถเฉพาะของการ์ดแต่ละผู้ผลิต ถึงความสามารถนั้น DirectSound จะไม่สนับสนุนก็ตามโดยให้สามารถใช้ได้โดยตรง DirectSound มีฟังก์ชันในการสร้างระบบเสียงสามมิติ ที่ใช้กันอยู่ทั่วๆ ไป เช่น ปรากฏการณ์ Dropper และระบบเสียงแบบรอบทิศ ใน DirectX เวอร์ชัน 5 ได้เพิ่มความสามารถในการบันทึกเสียงให้กับ DirectSound แต่ในเวอร์ชันปัจจุบันยังไม่สนับสนุน MIDI ที่จำเป็นมากสำหรับมัลติมีเดียและเกมส์ ซึ่งตอนนี้กำลังพัฒนาอยู่ภายใต้ชื่อ DirectMusic

ภายในโปรเจ็กนี้จะใช้ความสามารถในการเล่นเสียงในระบบสเตอริโอ โดยใช้ไฟล์ฟอร์แมตเป็น WAVE ไม่ได้ศึกษาเกี่ยวกับระบบในการสร้างเสียงแบบสามมิติเพราะมีรายละเอียดมากเกินไป ส่วนการเล่นเสียง MIDI นั้นจะเล่นผ่าน MCI (Media Control Interface) ซึ่งเป็นAPIมาตรฐานของวินโดว์

หลักการและการใช้งาน

สถาปัตยกรรมของ DirectSound

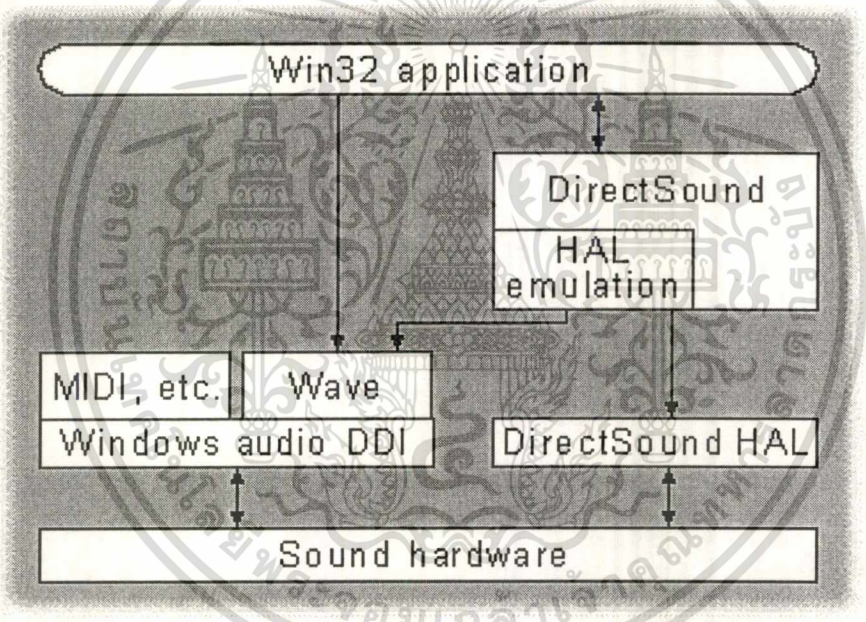
การออกแบบ DirectSound ได้ออกแบบให้มีการติดต่อแบบออปเจกต์ (component object model : COM) โดยที่ใช้งานคือ IDirectSound และ IDirectSoundBuffer

การเล่นเสียง DirectSound จะผสมเสียงจาก Secondary Sound Buffer ซึ่งข้อมูลที่อยู่ในบัฟเฟอร์นี้จะต้องอยู่ในรูปของ Pulse Code Modulation (PCM) ของแต่ละสัญญาณเสียง ถ้าข้อมูลเสียงไม่ได้อยู่ในรูปแบบที่

กำหนด ก็จะต้องเปลี่ยนรูปแบบเป็นตามที่กำหนดเมื่อนำมาใช้ในบัฟเฟอร์นี้ โดยแต่ละบัฟเฟอร์จะถูกสร้างขึ้นมาเมื่อต้องการใช้ และจะต้องคืนหน่วยความจำนั้นเมื่อเลิกใช้งาน

DirectSound จะสร้าง Primary Buffer ให้เองสำหรับเก็บเสียงหลังจากที่ผสมแล้ว และจะนำออกไปยังอุปกรณ์เสียง ถ้ามีหลายโปรแกรมใช้งานอุปกรณ์เสียงพร้อมกัน เสียงที่ออกมาจะเป็นของโปรแกรมที่ได้รับจุดสนใจในการทำงานเท่านั้น

DirectSound ติดต่อกับอุปกรณ์เสียงผ่าน Hardware Abstraction Layer (HAL) ซึ่งเป็นส่วนติดต่อของโปรแกรมควบคุมอุปกรณ์เสียง สนับสนุนการทำงานต่างๆ เช่น การจองการทำงานของอุปกรณ์เสียง, ตรวจความสามารถของอุปกรณ์เสียง, ทำงานเมื่ออุปกรณ์สนับสนุนคำสั่งนั้น, แจ้งความผิดพลาดเมื่ออุปกรณ์ไม่สนับสนุนคำสั่ง ถ้าอุปกรณ์เสียงไม่สนับสนุนคำสั่ง DirectSound จะทำการจำลองคำสั่งนั้นจากคำสั่งที่มีอยู่แล้วให้ (Hardware Emulation Layer : HEL)



รูปที่ 5.1 หลักการทำงานของ DirectSound

DirectSound Devices

การใช้งาน DirectSound จะต้องสร้าง DirectSound ออปเจคต์เพื่อเป็นส่วนติดต่อ จากนั้นจะสามารถควบคุมอุปกรณ์เสียง ความดังของเสียง ตรวจสอบความสามารถของอุปกรณ์เสียง และอื่นๆ ก่อนการสร้าง DirectSound ออปเจคต์ สามารถตรวจสอบว่ามีอุปกรณ์เสียงกี่ตัว แล้วเลือกใช้งาน DirectSound ออปเจคต์กับอุปกรณ์เสียงใดหรือจะให้ DirecSound เลือกใช้โดยอัตโนมัติ

Enumeration of Sound Devices

เป็นการตรวจดูว่ามีอุปกรณ์เสียงใดบ้างโดยใช้ฟังก์ชัน DirectSoundEnumerate แต่ถ้าต้องการเพียงเล่นเสียงตามปกติก็ไม่จำเป็นต้องหาอุปกรณ์เสียงตัวอื่นๆ

Creating the DirectSound Object

เมื่อเลือกใช้อุปกรณ์เสียงที่กำหนดไว้แล้ว (default device) โดยใช้ฟังก์ชัน DirectSoundCreate และใช้ค่า NULL ในการเจาะจงอุปกรณ์เสียงเพื่อใช้อุปกรณ์เสียงที่กำหนดไว้แล้ว ค่าที่ส่งกลับมาเมื่อมีข้อผิดพลาดแสดงว่าไม่มีอุปกรณ์เสียง หรืออุปกรณ์เสียงนั้นถูกใช้ในงานฟังก์ชันอื่น เมื่อเลิกใช้งานก็จะต้องยกเลิกการใช้งานอุปกรณ์เสียงโดยใช้ฟังก์ชัน IDirectSound::Release

การใช้งานการเริ่มการติดต่อกับอุปกรณ์เสียงด้วย DirectSound ด้วย DirectSoundCreate

```
LPDIRECTSOUND lpDirectSound;
HRESULT      hr;
hr = DirectSoundCreate(NULL, &lpDirectSound, NULL));
```

ฟังก์ชันจะส่งค่าผิดพลาดกลับมา ถ้าไม่มีอุปกรณ์เสียงหรืออุปกรณ์เสียงนั้นถูกใช้งานอยู่ แล้วอาจจะแจ้งให้ผู้ใช้ทราบ แล้วให้เลือกว่าทำงานต่อโดยไม่มีการใช้เสียงหรือให้ยกเลิกการใช้งานอุปกรณ์เสียงที่กับโปรแกรมก่อนหน้านี้

Cooperative Levels

การใช้อุปกรณ์เสียงบนวินโดวส์จะมีการใช้งานร่วมกันระหว่างโปรแกรมอื่นด้วย จึงต้องมีการกำหนดลักษณะการใช้งานกับอุปกรณ์เสียงโดยใช้ IDirectSound::SetCooperativeLevel ก่อนการใช้งานอุปกรณ์เสียง

การกำหนดลักษณะการใช้งานกับอุปกรณ์เสียง

```
HRESULT hr = lpDirectSound->lpVtbl->SetCooperativeLevel(
        lpDirectSound, hwnd, DSSCL_NORMAL);
```

ซึ่งสามารถกำหนดลักษณะการใช้งานได้ 4 แบบ คือ

Normal Cooperative Level โปรแกรมไม่สามารถกำหนดรูปแบบของบัฟเฟอร์เสียงหลัก (primary sound buffer) หรือเขียนบัฟเฟอร์หลัก ทุกโปรแกรมจะใช้รูปแบบเสียงคือ 22kHz, สเตอริโอ, 8-บิต ทำให้การสลับระหว่างโปรแกรมไม่มีการกระตุก

Priority Cooperative Level โปรแกรมสามารถควบคุมทรัพยากรต่างของอุปกรณ์เสียงได้ เช่น การผสมเสียง, เปลี่ยนรูปแบบของบัฟเฟอร์เสียงหลัก

Exclusive Cooperative Level โปรแกรมสมารถทำงานได้เหมือนกับ Priority Cooperative Level และเมื่อทำงานเบื้องหน้าจะมีเสียงจากโปรแกรมนี้เท่านั้น

Write-primary Cooperative Level โปรแกรมสามารถติดต่อกับบัฟเฟอร์เสียงหลัก และเมื่อต้องการเล่นเสียงจะต้องเขียนไปยังบัฟเฟอร์หลักโดยตรง

Device Capabilities

โปรแกรมอาจจะต้องการข้อมูลของอุปกรณ์เสียง เพื่อสามารถทำงานตามความเหมาะสมสำหรับอุปกรณ์เสียง แต่ส่วนใหญ่แล้ว DirectSound ก็ดูแลการใช้งานอยู่แล้ว สิ่งที่โปรแกรมต้องตรวจสอบก็คือ จำนวนช่องสัญญาณเสียงที่สามารถใช้งานได้พร้อมกัน หรือตรวจสอบว่าทรัพยากรเพียงพอหรือไม่ การเรียกดูข้อมูลโดยใช้ฟังก์ชัน IDirectSound::GetCaps

การใช้งานการเรียกข้อมูลเกี่ยวกับอุปกรณ์เสียง

```
DSCAPS dscaps;

dscaps.dwSize = sizeof(DSCAPS);
HRESULT hr = lpDirectSound->GetCaps(lpDirectSound,
   &dscaps);
```

ก่อนการใช้งาน DSCAPS จะต้องกำหนดค่าให้กับ dwSize ก่อน เมื่อเรียกฟังก์ชัน GetCaps แล้วข้อมูลต่างๆ จะอยู่ใน DSCAPS เกี่ยวกับความสามารถต่างๆ ทรัพยากรที่มีและมีให้ใช้แต่ละอย่างได้จำนวนเท่าไหร่

DirectSound Buffers

การเล่นเสียงจะต้องจัดการบัฟเฟอร์ก่อน เช่น การจองหน่วยความจำสำหรับบัฟเฟอร์เขียนข้อมูลเสียง การเล่นเสียง เป็นต้น

Buffer Basics

เมื่อมีการสร้าง DirectSound ออปเจคต์แล้ว DirectSound จะดูแลบัฟเฟอร์เสียงหลักให้ สำหรับการผสมเสียงและส่งไปยังอุปกรณ์เสียง โปรแกรมจะต้องจัดการกับบัฟเฟอร์รอง (secondary sound buffer) ซึ่งเก็บข้อมูลเสียงของแต่ละเสียง บัฟเฟอร์รองจะถูกจองไว้ใช้งานตลอดเวลาการทำงานหรือสามารถปล่อยเมื่อเลิกใช้งานเสียงนั้นได้ตลอดเวลา และสามารถเล่นเสียงเที่ยวเดียวหรือวนซ้ำก็ได้ ถ้าข้อมูลเสียงมีขนาดใหญ่ก็จะต้องเขียนข้อมูลต่อไปทับส่วนที่เล่นไปแล้วลงบนบัฟเฟอร์ เพื่อจะได้เล่นเสียงส่วนต่อไปได้

การผสมเสียงเกิดขึ้นเมื่อมีการเล่นเสียงจากบัฟเฟอร์เสียงรองหลายตัวพร้อมกัน การผสมเสียง DirectSound จะผสมเสียงลงบนบัฟเฟอร์เสียงหลัก ส่วนจำนวนเสียงที่สามารถเล่นได้พร้อมกันขึ้นอยู่กับความสามารถของหน่วยประมวลผล

Static and Streaming Sound Buffer

การสร้างบัฟเฟอร์เสียงรองสามารถสร้างได้ 2 แบบคือ

1. Static sound buffer สามารถเก็บข้อมูลเสียงได้ทั้งหมด

2. Streaming sound buffer สามารถเก็บข้อมูลเสียงได้เพียงบางส่วน เช่น เก็บข้อมูลเสียงได้ 3 วินาที จากทั้งหมด 15 วินาที ดังนั้นโปรแกรมจะต้องเขียนข้อมูลลงบัฟเฟอร์อยู่เป็นประจำเมื่อต้องการเล่นเสียงที่ยังไม่ได้อยู่ในบัฟเฟอร์

ถ้าอุปกรณ์เสียงมีหน่วยความจำอยู่ด้วยแล้ว DirectSound จะใช้หน่วยความจำส่วนนั้นทำเป็นบัฟเฟอร์ โดยเฉพาะ static buffer เพื่อสามารถผสมเสียงบนบอร์ดได้เลย ส่วน streaming buffer จะถูกใช้ในหน่วยความจำหลักเพื่อสามารถเขียนข้อมูลได้อย่างมีประสิทธิภาพ

โปรแกรมสามารถกำหนดได้ว่าจะต้องการสร้างบัฟเฟอร์บนบอร์ดหรือบนหน่วยความจำหลัก ในกรณีที่ต้องการสร้างบนบอร์ด แต่หน่วยความจำไม่พอหรือไม่สามารถผสมเสียงได้ การสร้างจะมีข้อผิดพลาดตอบกลับมา ถ้าอุปกรณ์เสียงไม่มีหน่วยความจำหรือความสามารถในการผสมเสียงจะไม่สามารถสร้างบัฟเฟอร์บนบอร์ดได้

Creating Secondary Buffers

การสร้างบัฟเฟอร์เสียงจะต้องกำหนดลักษณะของบัฟเฟอร์นั้นใน DSBUFFERDESC แล้วเรียกฟังก์ชัน IDirectSound::CreateSoundBuffer หลังจากสร้างบัฟเฟอร์ได้แล้วก็จะได้พอยต์เตอร์ของ IDirectSoundBuffer เพื่อใช้งานต่อไป เช่น การเขียนข้อมูล การเล่นเสียงในบัฟเฟอร์

การใช้งานการสร้างบัฟเฟอร์ข้อมูลเสียง

```
LPDIRECTSOUND lpDirectSound
LPDIRECTSOUNDBUFFER *lplpDsb

PCMWAVEFORMAT pcmwf;
DSBUFFERDESC dsbdesc;
HRESULT hr;

// Set up wave format structure.
memset(&pcmwf, 0, sizeof(PCMWAVEFORMAT));
pcmwf.wf.wFormatTag = WAVE_FORMAT_PCM;
pcmwf.wf.nChannels = 2;
pcmwf.wf.nSamplesPerSec = 22050;
pcmwf.wf.nBlockAlign = 4;
pcmwf.wf.nAvgBytesPerSec =
   pcmwf.wf.nSamplesPerSec * pcmwf.wf.nBlockAlign;
pcmwf.wBitsPerSample = 16;

// Set up DSBUFFERDESC structure.
memset(&dsbdesc, 0, sizeof(DSBUFFERDESC)); // Zero it out.
dsbdesc.dwSize = sizeof(DSBUFFERDESC);
// Need default controls (pan, volume, frequency).
dsbdesc.dwFlags = DSBCAPS_CTRLDEFAULT;
// 3-second buffer.
dsbdesc.dwBufferBytes = 3 * pcmwf.wf.nAvgBytesPerSec;
dsbdesc.lpwfxFormat = (LPWAVEFORMATEX)&pcmwf;
// Create buffer.
hr = lpDirectSound->lpVtbl->CreateSoundBuffer(lpDirectSound,
```

&dsbdesc, lplpDsb, NULL);

ลำดับการสร้างบัฟเฟอร์ควรจะสร้างบัฟเฟอร์สำหรับเสียงที่มีความสำคัญก่อน เนื่องจาก DirectSound จะใช้ทรัพยากรบนอุปกรณ์เสียงและสามารถใช้ประโยชน์จากอุปกรณ์เสียงได้ดีขึ้น หรือโปรแกรมจะกำหนดไปเลยว่าจะใช้หน่วยความจำส่วนใดโดยกำหนดค่าใน DSBUFFERDESC เป็น DSBCAPS_LOCHARDWARE สำหรับการสร้างบัฟเฟอร์บนหน่วยความจำของอุปกรณ์เสียง หรือ DSBCAPS_LOCSOFTWARE สำหรับการสร้างบัฟเฟอร์บนหน่วยความจำหลัก

เมื่อต้องการตรวจสอบว่าบัฟเฟอร์นั้นถูกจองบนหน่วยความจำส่วนใด ก็ใช้ฟังก์ชัน IDirectSoundBuffer::GetCaps หลังจากสร้างบัฟเฟอร์นั้นเสร็จ แล้วดูค่า dwFlags ของ DSBCAPS ว่าเป็น DSBCAPS_LOCHARDWARE หรือ DSBAPCS_LOCSOFTWARE

การกำหนดว่าบัฟเฟอร์นั้นเป็น Static Sound Buffer จะต้องกำหนดค่า DSBCAPS_STATIC ถ้าไม่กำหนดจะสร้างเป็นแบบ streaming sound buffer

Playing Sounds

การเล่นเสียงจะต้องเขียนข้อมูลลงบัฟเฟอร์ก่อนแล้วจึงจะสามารถเล่นเสียงได้ ดังนี้

1. ล็อคบางส่วนของบัฟเฟอร์รองโดยใช้ฟังก์ชัน IDirectSoundBuffer::Lock จะได้พอยต์เตอร์สำหรับตำแหน่งจะเขียนลงบัฟเฟอร์

2. เขียนข้อมูลเสียงในรูปแบบลงในบัฟเฟอร์

3. ปลดล็อคบัฟเฟอร์ โดยใช้ฟังก์ชัน IDirectSoundBuffer::Unlock

4. ใช้ฟังก์ชัน IDirectSoundBuffer::Play เพื่อส่งข้อมูลเสียงไปผสมลงบนบัฟเฟอร์เสียงหลัก แล้วส่งต่อไปยังอุปกรณ์เสียง

บทที่ 6

DirectPlay

DirectPlay เป็น API ในระดับสูงที่ช่วยให้การเขียนโปรแกรมติดต่อผ่านเน็ตเวิร์กง่ายขึ้น API ของ DirectPlay จะสนับสนุนการเขียนเกมส์โดยตรง มีฟังก์ชันในการจัดการ session ซึ่งเป็นช่องสัญญานแบบลอจิคอลที่ใช้ในการติดต่อสื่อสาร ถ้าใครต้องการติดต่อกับผู้ที่อยู่ใน session ก็ต้องเข้าไปใน session นั้นก่อน สิ่งพื้นฐานในการติดต่อสื่อสารในระบบเน็ตเวิร์กก็คือ ความถูกต้องของสถานะของ session นั้น ณ เวลาใดเวลาหนึ่ง เช่นการเล่นเกมส์แบบผู้เล่นหลายคน ผู้เล่นทุกคนจะมีสถานะของแต่ละคน รวมทั้งสภาพแวดล้อมของแผนที่ที่ทุกคนอยู่ ถ้าผู้เล่นคนใดคนหนึ่งเปลี่ยนสถานะเช่นเคลื่อนที่ ทุกคนภาพใน session นั้นทุกคนก็ต้องรู้ว่าผู้เล่นคนนั้นเคลื่อนที่เช่นกัน ดังนั้นผู้ที่อยู่ใน session ทุกคนจำเป็นต้องรับ/ส่ง สภาพแวดล้อมของคนอื่นใน session อยู่ตลอดเวลา ซึ่งมีอยู่หลายวิธีในการแลกเปลี่ยนข้อมูลของภายใน session แต่ทุกวิธีจำเป็นต้องใช้การส่ง message เพื่อแลกเปลี่ยนข้อมูลข่าวสาร

ปัญหาหลักของการส่ง message ก็คือเวลา เนื่องจากเน็ตเวิร์กทั่วไปไม่สามารถรับประกับความแน่นอนของเวลาที่ใช้ในการส่งได้ การส่ง message อาจมีการล่าช้าออกไปเนื่องมาจากความคับคั่งของเน็ตเวิร์ก การสูญหายของข้อมูล ฯลฯ ซึ่งจะเกิดปัญหากับบางสถานการณ์ที่ไม่สามารถทำได้พร้อมกันเช่น ตัวละครในเกมส์สองตัวกำลังจะเปิดประตูเดียวกันพร้อมๆกัน เราจำเป็นต้องแก้ปัญหาของความขัดแย้งนี้ก่อนที่จะเปลี่ยนสถานะของ session ซึ่งก็มีอีกหลายวิธีการเช่นกันอย่างเช่นกำหนดว่าเมื่อเกิดความขัดแย้งในลักษณะนี้ขึ้น จะให้ผู้เล่นที่มีสถานะสูงกว่าเป็นผู้ได้ไป

การติดต่อสื่อสารในระบบเน็ตเวิร์กมีสองแบบ

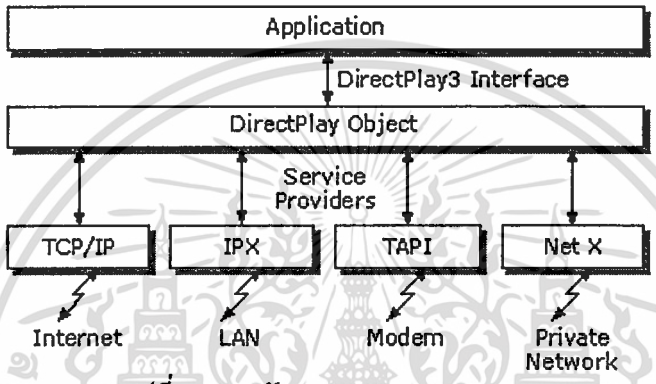
peer-to-peer เป็นการติดต่อสื่อสารในลักษณะที่เครื่องคอมพิวเตอร์ทุกเครื่องที่อยู่ใน session เดียวกันมีความเท่าเทียมกัน และเครื่องแต่ละเครื่องจะต้องเก็บสถานะทั้งหมดไว้อย่างสมบูรณ์ ซึ่งหมายความว่าทุกเครื่องต้องมีสำเนาของสภาพแวดล้อมใน session ที่เหมือนกัน เมื่อมีการเปลี่ยนแปลงสถานะของเครื่องใดเครื่องหนึ่ง เครื่องนั้นก็ต้องส่งสถานะนั้นให้กับทุกเครื่องใน session นั้นด้วย

client/server เป็นลักษณะการติดต่ออีกแบบหนึ่งที่ต้องมีเครื่องคอมพิวเตอร์เครื่องหนึ่งถูกกำหนดให้เป็นเครื่องให้บริการของ session นั้นๆซึ่งทำหน้าที่ควบคุมและจัดการแก้ไขความขัดแย้งภายใน session นั้นเพื่อให้สถานะของ session ถูกต้องอยู่เสมอ ส่วนเครื่องอื่นๆที่เหลือใน session นั้นๆก็จะถูกกำหนดให้เป็นเครื่องใช้บริการ ซึ่งจะรับข้อมูลของสถานะที่จากเครื่องให้บริการและเมื่อเครื่องใช้บริการ ได้มีการเปลี่ยนแปลงสถานะ ก็ต้องแจ้งให้เครื่องให้บริการทุกครั้ง

หลักการและการใช้งาน

สถาปัตยกรรมของ DirectPlay

DirectPlay เป็น API ด้านเน็ตเวิร์ก มีฟังก์ชันพื้นฐานให้ใช้งาน โดยไม่ขึ้นกับเน็ตเวิร์กใด เพราะ DirectPlay จะทำงานจำลองการทำงานของฟังก์ชันนั้น หากเน็ตเวิร์กชนิดนั้นไม่สนับสนุนการทำงาน เช่น การส่งข้อมูลเป็นกลุ่ม หรือการส่งที่ต้องการความแน่นอน และยังช่วยให้การเขียนโปรแกรมไม่ต้องสนใจชนิดของเน็ตเวิร์ก แต่เมื่อต้องการรายละเอียดของเน็ตเวิร์กก็ได้ เพื่อต้องการปรับสภาพความเหมาะสมของโปรแกรมกับลักษณะของเน็ตเวิร์ก เช่น ข้อมูลที่สามารถส่งได้ภายใน 1 วินาที หรือความล่าช้าในการส่งข้อมูล



รูปที่ 6.1 สถาปัตยกรรมของ DirectPlay

อย่างแรก ต้องเลือกบริการที่มีให้บนเครื่องนั่นหรือเน็ตเวิร์กที่เครื่องนั้นต่ออยู่ เช่น TCP/IP ที่ใช้ติดต่อบนอินเตอร์เนต, IPX ที่ใช้บน LAN หรือการติดต่อผ่านสายส่งข้อมูลแบบอนุกรมระหว่างเครื่อง เมื่อเลือกบริการที่จะใช้ก็สามารถกำหนดการเริ่มการติดต่อบนบริการนั้น

ฟังก์ชันที่จัดการกับใช้งานการติดต่อ

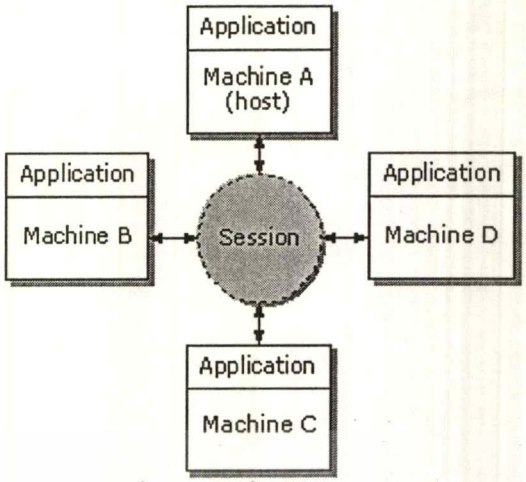
DirectPlayEnumerate เพื่อหาบริการที่มีให้ใช้งานบนเครื่องนั้น

DirectPlayCreate เพื่อใช้งานบริการที่ต้องการ

การจัดการ Session

หลังจากที่เลือกใช้งานบริการเน็ตเวิร์ก ต่อไปต้องเลือกหรือสร้าง session ภายในบริการนั้น ซึ่ง session เสมือนเป็นช่องการส่งข้อมูลภายในบริการ การสร้าง session เมื่อต้องการสร้างช่องการส่งข้อมูลใหม่ให้ผู้เล่นอื่นสามารถเข้ามาติดต่อส่งข้อมูลกันได้ หรือหา session ที่มีบนบริการนั้นแล้วเลือกว่าจะใช้ session ไหน แล้วเข้าไปใน session นั้น ต่อจากนั้นจะต้องสร้างตัวผู้เล่นเพื่อสามารถให้ส่งข้อมูลระหว่างกันได้

แต่ละ session จะมีเครื่องหนึ่งที่เป็นเจ้าของ session นั้น สามารถจัดการคุณสมบัติของ session ได้



รูปที่ 6.2 การเชื่อมต่อใน session

ฟังก์ชันที่จัดการกับ session

IDirectPlay3::EnumSessions เพื่อหา session ที่มีบนบริการที่ใช้งาน

IDirectPlay3::Open เมื่อต้องการเข้าหรือสร้าง session

IDirectPlay3::Close ออกจาก session

ฟังก์ชันเพิ่มเติมเกี่ยวกับการจัดการ session

IDirectPlay3::GetSessionDesc ต้องการข้อมูลเกี่ยวกับ session

IDirectPlay3::SetSessionDesc เปลี่ยนแปลงสมบัติของ session

การจัดการ Player

ภายใน sesssion จะต้องมีผู้เล่น (player) ซึ่งผู้เล่นจะใช้ในการติดต่อส่งและรับข้อมูลภายใน session แต่ละผู้เล่นจะใช้เพื่อระบุถึงปลายทางเท่านั้น ซึ่งผู้เล่นที่กำหนดอาจจะอยู่ที่เครื่องที่ใช้งานอยู่ หรืออาจจะอยู่อีกเครื่องหนึ่ง ดังนั้นแต่ละเครื่องจะมีผู้เล่นใน session อย่างน้อยหนึ่งคนจึงสามารถทำการส่งและรับข้อมูลได้

การส่งข้อมูลจะระบุผู้เล่นปลายทาง (ไม่ต้องระบุเครื่องปลายทาง) ถ้าผู้เล่นปลายทางอยู่บนเครื่องเดียวกัน ข้อมูลนั้นไม่จำเป็นที่จะต้องส่งไปบนเน็ตเวิร์ก ในกรณีการรับข้อมูลจะถูกระบุผู้เล่นปลายทางและจะบอกว่ามาจากผู้เล่นใดด้วย ถ้าเป็นข้อมูลของระบบจะถูกระบุเป็น DPID_SYSMSG

ฟังก์ชันพื้นฐานการจัดการเกี่ยวกับผู้เล่น

IDirectPlay3::EnumPlayers ตรวจหาผู้เล่นทั้งหมดใน session

IDirectPlay3::CreatePlayer สร้างผู้เล่น เพื่อสามารถรับและส่งข้อมูล

IDirectPlay3::DestroyPlayer ยกเลิกการใช้งานผู้เล่น

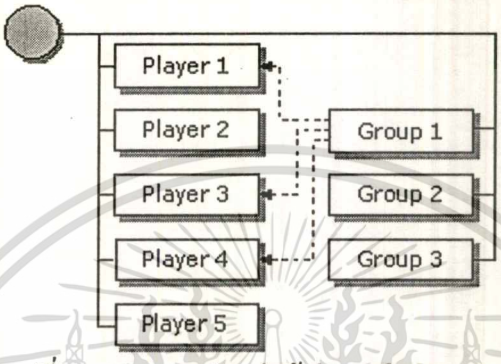
ฟังก์ชันเพิ่มเติมเกี่ยวกับผู้เล่น

IDirectPlay3::GetPlayerName เมื่อต้องการชื่อของผู้เล่น

IDirectPlay3::SetPlayerName เปลี่ยนชื่อของผู้เล่น

การจัดการ Group

DirectPlay สนันสนุนการจัดการผู้เล่นเป็นกลุ่ม โดยกลุ่มจะประกอบด้วยผู้เล่นรวมกัน เมื่อมีการส่งข้อมูลไปยังกลุ่มที่กำหนดเพียงครั้งเดียว ผู้เล่นที่อยู่ในกลุ่มจะได้รับข้อมูลทุกคน การจัดการผู้เล่นสามารถกำหนดให้ผู้เล่นอยู่ได้มากหนึ่งกลุ่ม เมื่อกำหนดกลุ่มแล้วสามารถกำหนดชื่อและสมบัติ เพื่อความสะดวกในการอ้างถึง



รูปที่ 6.3 แสดงการจัดกลุ่มผู้เล่นภายใน session

ฟังก์ชันเกี่ยวกับการจัดการกลุ่ม

IDirectPlay3::EnumGroups ตรวจหากลุ่มทั้งหมดภายใน session

IDirectPlay3::CreateGroup สร้างกลุ่มใหม่

IDirectPlay3::DestroyGroup ยกเลิกการกำหนดกลุ่ม

IDirectPlay3::EnumGroupPlayers ตรวจหาผู้เล่นภายในกลุ่ม

IDirectPlay3::AddPlayerToGroup เพิ่มผู้เล่นเข้ากลุ่ม

IDirectPlay3::DeletePlayerFromGroup ลบผู้เล่นออกจากกลุ่ม

ฟังก์ชันเพิ่มเติมเกี่ยวกับการจัดการกลุ่ม

IDirectPlay3::GetGroupName เมื่อต้องการชื่อกลุ่ม

IDirectPlay3::SetGroupName กำหนดชื่อให้กับกลุ่ม

การจัดการกับข้อมูลภายในระบบ

โปรแกรมเมื่อสร้างผู้เล่นใน session แล้ว จะสามารถรับและส่งข้อมูลระหว่างผู้เล่น ในการส่งข้อมูลของ DirectPlay ไม่ต้องมีรูปแบบของข้อมูล การใช้รูปแบบจะขึ้นอยู่กับโปรแกรมเป็นผู้กำหนดเอง การส่งข้อมูลสามารถส่งได้หลายลักษณะ คือ จงเจาะผู้รับทั้งคนเดียว เป็นกลุ่มผู้รับ หรือผู้เล่นทั้งหมดภายใน session เมื่อส่งข้อมูลจะต้องระบุว่ามาจากผู้เล่นคนใด

ข้อมูลที่โปรแกรมรับจะถูกเก็บเป็นคิว จากนั้นจะถูกนำมาประมวลผลแยกกันแต่ละข้อมูลตามความเหมาะตามชนิดของข้อมูล โปรแกรมสามารถวนรอบเพื่อรับข้อมูล หรือสร้างเทรด(thread) เพื่อรับข้อมูลโดยเฉพาะ

ประเภทข้อมูลที่ส่งกันมี 2 แบบ คือ

1. ข้อมูลระหว่างผู้เล่นด้วยกัน การส่งข้อมูลชนิดนี้ส่งภายใน session โดยส่งไปยังผู้เล่นปลายทางและมีการบอกต้นทางของข้อมูล
2. ข้อมูลของระบบ ถูกส่งไปยังผู้เล่นทุกคนใน session เพื่อบอกการโปรแกรมถึงการเปลี่ยนแปลงสถานะของ session เช่น เมื่อมีผู้เล่นเข้า/ออก

ฟังก์ชันพื้นฐานเกี่ยวกับการรับ/ส่งข้อมูล

IDirectPlay3::Send ส่งข้อมูลไปยังผู้เล่นอื่นภายใน session

IDirectPlay3::Receive รับข้อมูลจากคิวข้อมูล

ประเภทของการส่งข้อมูล

เมื่อเริ่มต้นการใช้งานการติดต่อด้วย DirectPlay จะเป็นการติดต่อแบบ peer-to-peer การติดต่อแบบนี้แต่ละเครื่องจะต้องมีสถานะของ session ครบ เช่น ผู้เล่น, กลุ่มผู้เล่น และข้อมูลของ session เอง เมื่อแต่ละเครื่องมีการเปลี่ยนแปลงสถานะของตนเองจะต้องมีการบอกไปยังทุกเครื่องภายใน session

มีการติดต่ออีกแบบ คือ แบบ client/server การส่งแบบนี้จะต้องมีเครื่องให้บริการเพื่อเก็บข้อมูลของ session ที่ครบถ้วน ส่วนเครื่องใช้บริการจะเก็บข้อมูลเพียงบางส่วนของ session หรือที่เกี่ยวข้องกับสถานะของเครื่องตนเองเท่านั้น และข้อมูลนี้จะได้รับจากเครื่องให้บริการ เมื่อมีเครื่องที่เปลี่ยนสถานะเครื่องนั้นต้องส่งข้อมูลการเปลี่ยนสถานะไปยังเครื่องให้บริการ จากนั้นเครื่องให้บริการจะทำการประมวลผลและส่งการเปลี่ยนแปลงสถานะไปยังเครื่องที่เกี่ยวข้อง ถึงการเปลี่ยนสถาะใหม่

โปรแกรมสามารถเลือกใช้รูปแบบการติดต่อส่งข้อมูลระหว่างกันได้ ไม่ว่าจะใช้แบบ peer-to-peer หรือ client/server ก็ไม่มีผลต่อการจัดการ session ของ DirectPlay

บทที่ 7
การสร้างและการออกแบบ

ในเอกสารชุดนี้ไม่ได้เน้นการออกแบบมากนักด้วยเหตุผลหลายประการ คือไม่สามารถแสดง Flow Chart ได้โดยตรงเพราะการเขียนโปรแกรมเกือบทั้งหมดเป็นรูปแบบของ Object Oriented ดังนั้นจะต้องทำการอธิบายพร้อมไปกับ source code ซึ่งจะแยกออกจากตัวเอกสารประกอบโปรเจ็กชุดนี้ออกมาอีกชุดหนึ่ง และเนื่องจากโปรแกรมส่วนใหญ่เป็นโปรแกรมเพื่อการทดสอบความสามารถของฟังก์ชันการทำงาน ซึ่งจะมีโครงสร้างขนาดเล็ก จึงเป็นการสะดวกกว่าที่คำอธิบายจะอยู่ในตัวโปรแกรม และสามารถเข้าใจได้โดยง่ายถ้าได้ให้โปรแกรมทำงานไปด้วย

บทที่ 8

ผลการทดลอง

จากการเขียนโปรแกรมเพื่อทำการทดสอบเปรียบเทียบกับแอปพลิเคชันที่มีใช้กันอยู่ทั่วไป มีผลดังนี้

DirectDraw

ความเร็วในการแสดงผล ทำได้รวดเร็วโปรแกรมที่เขียนผ่าน GDI ของวินโดว์มาก ในที่นี้จะใช้โปรแกรม ball ในการทำการทดสอบ สามารถทำอัตราของเฟรมระหว่าง 60-50 เฟรมต่อวินาทีที่ภาพลูกบอลระหว่าง 0 – 600 ลูก แต่อัตราของเฟรมจะตกอย่างรวดเร็วเหลือ 30 เฟรมต่อวินาทีที่จำนวนลูกบอลใกล้ 700 ลูก หลังจากนั้นอัตราของเฟรมไม่ลดลงมากนักเมื่อจำนวนลูกบอลมากขึ้นเรื่อยๆ (ทำได้มากกว่า 2,000 ลูก) โปรแกรมที่ใช้ฟังก์ชันกราฟฟิกผ่าน GDI ไม่สามารถทำได้ ความเร็วที่ได้น่าจะเร็วใกล้เคียงกับการเขียนโปรแกรมด้วยภาษาแอสเซมบลี เพื่อติดต่อกับการ์ดแสดงผลโดยตรง(ไม่ได้ทดสอบ) แต่การพัฒนาทำได้ง่ายกว่าบนใช้ภาษาแอสเซมบลีมาก

DirectInput

การป้อนอินพุตสามารถทำได้ดีกว่าการใช้ message ของวินโดว์ จากโปรแกรมในชุดทดสอบ donut จะแสดงให้เห็นความแตกต่างระหว่างการเคลื่อนโดนัท โดยใช้loop เพื่ออ่านอินพุตของ DirectInput และ message ซึ่งการเคลื่อนไหวของโดนัทจะกระตุกเมื่อใช้ message แต่ของ DirectInput จะเคลื่อนที่ได้ราบเรียบ เพราะไม่มีการล่าช้าของอินพุต

DirectSound

ทำโปรแกรมตัวอย่างแต่ไม่ได้ทดสอบเนื่องจากการแสดงเสียงไม่สามารถวัดได้เห็นความแตกต่างให้เห็นได้ชัดเจนและในชุดของ DirectX ยังขาดฟังก์ชันที่สำคัญในการใช้ MIDI ในโปรแกรมตัวอย่างจะใช้ MCI ของวินโดว์แทน

DirectPlay

โปรแกรมตัวอย่างที่ใช้ทดความสามารถของ DirectPlay บอกได้เพียงว่า DirectPlay ทำให้การเขียนโปรแกรมสื่อสารบนระบบเครือข่ายทำได้ง่าย โดยไม่ต้องคำนึงถึงโปรโตคอล หรือระบบการสื่อสารมากนัก แต่ไม่ได้วัดความเร็วของการติดต่อสื่อสาร เพราะอยู่นอกเหนือหัวข้อของโปรเจ็ก

Direct3D

โปรแกรมที่ใช้ทดสอบได้ดัดแปลงมาจากโปรแกรมตัวอย่าง hello_world ความเร็วในการแสดงผลทำได้ดีกว่าโปรแกรมแสดงภาพสามมิติทั่วๆไปบนวินโดว์ สามารถทำการเรนเดอร์แบบ real-time ได้ แต่จากการทำงานคาดว่ายังทำได้ไม่ดีเท่าดอส อาจเป็นเพราะฟังก์ชันที่ใช้เป็นฟังก์ชันระดับสูงเกินไปไม่สามารถเทียบความเร็วของแอปพลิเคชันบนดอสที่เขียนด้วยภาษาแอสเซมบลีได้ และการ์ดที่ใช้ในการทดสอบไม่ได้สนับสนุนกราฟฟิกสามมิติ ดังนั้นการทำงานหลายอย่างต้องมีการจำลองการทำงานด้วยฮาร์ดแวร์

หมายเหตุ

การทดสอบทำบนเครื่องรุ่นไม่ต่ำกว่า

Pentium 75

RAM 48M

การ์ดจอ Teseng ET6000 PCI RAM 2M

จอยแพด 4 ปุ่ม

ระบบปฏิบัติการวินโดว์ 95 และ NT

DirectX 3 และ 5

บทที่ 9
บทวิจารณ์และสรุป

จากการที่ได้ทดลองและทดสอบแอปพลิเคชันที่ใช้ DirectX ทั้งที่เป็นแอปพลิเคชันที่เขียนเองและที่มีอยู่ในท้องตลาด พบว่าความเร็วที่ได้รับอยู่ในระดับที่น่าพอใจ ในท้องตลาดเกมส์ที่มีสองเวอร์ชันทั้งบนดอสและวินโดว์ บนวินโดว์จะแสดงกราฟฟิกได้สวยกว่ามาก และจากการเขียนโปรแกรมทดสอบด้วยตนเองพบว่าการพัฒนาแอปพลิเคชันด้วยชุด DirectX SDK สามารถทำให้การพัฒนาทำได้ง่ายและรวดเร็ว เอกสารและฮาร์ดแวร์สนับสนุนมาก เมื่อเทียบกับการใช้ GDI ของวินโดว์ ก็พบว่าดีกว่าในทุกด้าน แต่จะด้อยกว่าตรงที่เสถียรภาพของวินโดว์จะลดลงเพราะติดต่อกับฮาร์ดแวร์โดยตรง และทำการดีบักในโหมดการทำงานแบบเต็มหน้าจอไม่ได้ ต้องทำในวินโดว์โหมดถึงจะดีบักได้ แล้วทำการแก้ไขให้เป็นแบบเต็มหน้าจอถ้าต้องการ สรุปก็คือข้อดีเป็นไปตามที่บทคัดย่อได้กล่าวไว้คือ

เร็ว การติดต่อของ DirectX จะติดต่อกับฮาร์ดแวร์โดยตรง เน้นที่การแสดงผลทำได้อย่างรวดเร็วมาก ถ้าเป็นฮาร์ดแวร์รุ่นใหม่ๆก็จะสนับสนุน DirectX โดยตรง

ง่าย การพัฒนาแอปพลิเคชันทำได้ง่ายขึ้น ไม่จำเป็นต้องเขียนภาษาแอสเซมบลี แต่ก็ไม่สูญเสียความสามารถในการติดต่อฮาร์ดแวร์โดยตรง เมื่อความสามารถของฮาร์ดแวร์มีมากยิ่งขึ้น ก็ไม่ต้องเขียนโปรแกรมใหม่ และจากการได้ทดลองเขียนโปรแกรมเองพบว่า การเขียนโปรแกรมไม่ต้องขึ้นกับรุ่นหรือผู้ผลิตฮาร์ดแวร์ชนิดนั้นๆสามารถใช้โปรแกรมร่วมกันได้เลย

จากการทดลองทำให้เห็นว่าเป็นการสมควรอย่างยิ่งที่จะใช้เทคโนโลยี DirectX พัฒนาแอปพลิเคชั่น เกมส์และมัลติมีเดียโดยเฉพาะอย่างยิ่งด้านกราฟฟิก

ภาคผนวก ก
ฟังก์ชันที่ใช้เขียนโปรแกรมบนวินโดว์

การเขียนโปรแกรมบนวินโดว์จะต้องคำนึงถึงสามส่วนที่สำคัญคือ การสร้าง, message ของวินโดว์ และ main โปรแกรมของวินโดว์ การเขียนโปรแกรมโดยใช้ DirectDraw บนวินโดว์โหมด สามารถสร้างวินโดว์ในรูปแบบใหนก็ได้ แต่ถ้าเขียนในโหมดเต็มหน้าจอ (full screen) และ exclusive เราไม่สามารถใช้รูปแบบของ WS_EX_TOOLWINDOW ได้ เพราะว่าจะกันไม่ให้วินโดว์ไปอยู่บนสุดของหน้าจอโดยแอปพลิเคชันอื่นสามารถขึ้นมาบังได้ในโหมดเต็มหน้าจอต้องใช้ WS_EX_TOPMOST ซึ่งเป็นรูปแบบที่เพิ่มขึ้นมาในวินโดว์และใช้ WS_VISIBLE เพื่อแสดงมันออกมาดังนั้นต้องการทำการกำหนดวินโดว์แบบนี้

เริ่มต้นการสร้างวินโดว์

```
#include <windows.h>
#include <windowsx.h>
..
..
//////////////////////////////////////////////////
// Register the window class, display the window, and init
// all DirectX and graphic objects.
//////////////////////////////////////////////////
BOOL WINAPI InitApp(INT nWinMode)
{
  WNDCLASSEX wcex;

  wcex.cbSize        =  sizeof(WNDCLASSEX);
  wcex.hInstance     =  g_hinst;
  wcex.lpszClassName =  g_szWinName;
  wcex.lpfnWndProc   =  WndProc;
  wcex.style         =  CS_VREDRAW|CS_HREDRAW|CS_DBLCLKS;
  wcex.hIcon         =  LoadIcon (NULL, IDI_APPLICATION);
  wcex.hIconSm       =  LoadIcon (NULL, IDI_WINLOGO);
```

```
    wcex.hCursor        =   LoadCursor (NULL, IDC_ARROW);
    wcex.lpszMenuName   =   MAKEINTRESOURCE(IDR_APPMENU);
    wcex.cbClsExtra     =   0 ;
    wcex.cbWndExtra     =   0 ;
    wcex.hbrBackground  =   GetStockObject (NULL_BRUSH);

    RegisterClassEx(&wcex);                  //ใช้ RegisterClass(&wcex); ก็ได้

    g_hwndMain = CreateWindowEx(
                WS_EX_TOPMOST,
                g_szWinName,
                g_szWinCaption,
                WS_VISIBLE|WS_POPUP,
                0,0,CX_SCREEN,CY_SCREEN,
                NULL,
                NULL,
                g_hinst,
                NULL);
    if(!g_hwndMain)
      return(FALSE);
    SetFocus(g_hwndMain);
    ShowWindow(g_hwndMain, nWinMode);
    UpdateWindow(g_hwndMain);

    return TRUE;
}
```

แต่ที่ใช้เขียนโปรแกรมจริงๆ ในโปรเจ็กจะใช้รูปแบบดังข้างล่างเพราะว่ากระทัดรัดกว่า และสามารถ ดีบักได้

```
wc.style = CS_HREDRAW | CS_VREDRAW;
wc.lpfnWndProc = WindowProc;
```

```
wc.cbClsExtra = 0;
wc.cbWndExtra = 0;
wc.hInstance = hInstance;
wc.hIcon = LoadIcon( hInstance, IDI_APPLICATION );
wc.hCursor = LoadCursor( NULL, IDC_ARROW );
wc.hbrBackground = GetStockBrush(BLACK_BRUSH);
wc.lpszMenuName = NAME;
wc.lpszClassName = NAME;
RegisterClass( &wc );

hwnd = CreateWindowEx(
        0,                          // ไม่มี WS_EX_TOPMOSTเพื่อจะได้เห็นผลจากการdebug ได้
        NAME,
        TITLE,
        WS_POPUP,
        0,
        0,
        GetSystemMetrics(SM_CXSCREEN),
        GetSystemMetrics(SM_CYSCREEN),
        NULL,
        NULL,
        hInstance,
        NULL );
if( !hwnd )
{
        return FALSE;
}
g_hwnd = hwnd;
ShowWindow( hwnd, nCmdShow );
UpdateWindow( hwnd );
```

Message ของวินโดว์

ฟังก์ชันส่วนใหญ่ของวินโดว์จะทำการติดต่อโดยใช้ message แต่ถ้าใช้ DirecrtX จะทำให้ไม่ต้องใช้ message ของวินโดว์อีกหลายตัวเช่น message เกี่ยวกับอุปกรณ์อินพุตในกรณีที่ใช้ DirectInput และการแสดงผลผ่าน GDI จะไม่ได้ใช้ถ้าใช้ DirectDraw

```
long FAR PASCAL WindowProc( HWND hWnd, UINT message,
                            WPARAM wParam, LPARAM lParam )
{
  switch( message )
  {
  case WM_ACTIVATEAPP:
        bActive = wParam;
        break;

  case WM_SETCURSOR:        //ซ่อนcursor
    SetCursor(NULL);
    return TRUE;

  case WM_KEYDOWN: //ใช้ได้ในกรณีที่ DirectInput ไม่ได้อยู่ในโหมด exclusive
        switch( wParam )
        {
              case VK_ESCAPE:
                    PostMessage(hWnd, WM_CLOSE, 0, 0); //ส่งmessage เพื่อปิดวินโดว์
                    break;
        }
        break;

  case WM_DESTROY://ได้รับmessage เพื่อให้ทำลายวินโดว์(ให้ free memory ที่ได้จองไว้ทั้งหมดตรงนี้
        finiObjects();              //ฟังก์ชันสำหรับ free ออปเจ็กของ DirectX
```

```
        PostQuitMessage( 0 );      //ส่ง message เพื่อให้หลุดจาก main โปรแกรมในส่วนที่วนรอบ
อยู่
        break;
    }
    return DefWindowProc(hWnd, message, wParam, lParam);  //ไม่มี message ที่ดักเอาไว้จะส่งให้
วินโดว์จัดการต่อ
}
```

เริ่มโปรแกรม

```
int PASCAL WinMain( HINSTANCE hInstance, HINSTANCE hPrevInstance,
                    LPSTR lpCmdLine, int nCmdShow)
{
    g_hinst = hInstance;
    MSG        msg;
    lpCmdLine = lpCmdLine;
    hPrevInstance = hPrevInstance;
    if( !doInit( hInstance, nCmdShow ) )        // initial วินโดว์และDirectX
    {
        return FALSE;
    }

    while (1)                                   //loop ของโปรแกรม
    {
    if(PeekMessage(&msg,NULL,0,0,PM_NOREMOVE))
    {
      if (!GetMessage(&msg,NULL,0,0))
        return msg.wParam;
      TranslateMessage(&msg);
      DispatchMessage(&msg);
    }
    else
```

```
        if (bActive) //ใช้บอกว่าโปรแกรมกำลังทำงานอยู่หรือไม่
        {
          /*ตรงนี้ให้ใส่โค้ดของโปรแกรมตามต้องการ*/
        }else
        {
          WaitMessage();
        }
      }
} /* WinMain */
```

ภาคผนวก ข
ฟังก์ชันที่ใช้เขียนโปรแกรมโดยใช้ DirectDraw

การติดต่อ

การใช้ไลบรารีของ DirectDraw ควรประกาศอินเตอร์เฟสของ DirectDraw ด้วย(ddraw.h) และตั้งให้

#include <ddraw.h>

ตัวแปรที่ใช้

การประกาศตัวแปรที่เป็นพอยต์เตอร์ของออปเจ็ก ส่วนใหญ่จะประกาศไว้ส่วนต้นของโปรแกรม

```
LPDIRECTDRAW            lpDD;    // พอยต์เตอร์ของออปเจ็กของ DirectDraw
LPDIRECTDRAWSURFACE     lpDDSPrimary;   // พอยต์เตอร์ของ DirectDraw primary
surface
LPDIRECTDRAWSURFACE     lpDDSBack;      // พอยต์เตอร์ของ DirectDraw back surface
LPDIRECTDRAWSURFACE     lpDDSOne;       // พอยต์เตอร์ของ Offscreen surface จะมีกี่อัน
ก็ได้
LPDIRECTDRAWPALETTE     lpDDPal; // พอยต์เตอร์ของ DirectDraw palette
```

DirectDrawCreate

การสร้างออปเจ็กของ DirectDraw ใช้ฟังก์ชัน

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

lpGUID แอดเดรสของไดร์เวอร์ที่ใช้ในตัวอย่างจะให้เป็น NULL ซึ่งหมายความว่าใช้ไดร์เวอร์ที่ทำงานอยู่

lplpDD แอดเดรสของพอยต์เตอร์ของออปเจ็ก DirectDraw

pUnkOuter เผื่อไว้ใช้ในอนาคตเพื่อความเข้ากันได้กับ COM เวอร์ชันต่อๆไป ดังนั้นขณะนี้ให้เป็น NULL เสมอ

HRESULT ผลลัพท์ของการเรียกฟังก์ชันถ้าสำเหร็จจะส่งค่ากลับมาเท่ากับ DD_OK

ตัวอย่างการใช้งาน

ddrval = DirectDrawCreate(NULL, &lpDD, NULL);

IDirectDraw2::SetCooperativeLevel

ออปเจ็กของ DirectDraw จะเป็นตัวต้นกำเนิด surface กำหนดสมบัติและโหมดของการแสดงผลซึ่งจะทำดังนี้

กำหนดคุณสมบัติการแสดงผล

HRESULT SetCooperativeLevel(**HWND** hWnd, **DWORD** dwFlags);

hWnd แฮนด์เดิลของวินโดว์

dwFlags แฟลกกำหนดสถานะการทำงานอย่างในตัวอย่างจะใช้โหมด exclusive และเต็มหน้าจอ

HRESULT ผลลัพท์ของการเรียกฟังก์ชันถ้าสำเหร็จจะ ส่งค่ากลับมาเท่ากับ DD_OK

ตัวอย่างการใช้งาน

ddrval = lpDD->SetCooperativeLevel(hwnd, DDSCL_EXCLUSIVE | DDSCL_FULLSCREEN);

IDirectDraw2::SetDisplayMode

กำหนดโหมดของการแสดงผล

HRESULT SetDisplayMode(**DWORD** dwWidth, **DWORD** dwHeight,

DWORD dwBPP);

dwWidth ความกว้างในหน่วยพิกเซลของหน้าจอ

dwHeight ความยาวในหน่วยพิกเซลของหน้าจอ

dwBPP จำนวนบิตของสีที่ใช้

HRESULT ผลลัพท์ของการเรียกฟังก์ชันถ้าสำเหร็จจะส่งค่ากลับมาเท่ากับ DD_OK

ตัวอย่างการใช้งาน

ddrval = lpDD->SetDisplayMode(640, 480, 8);

IDirectDraw2::CreateSurface

การสร้าง surface

HRESULT CreateSurface(

LPDDSURFACEDESC lpDDSurfaceDesc,

LPDIRECTDRAWSURFACE FAR *lplpDDSurface,

IUnknown FAR *pUnkOuter

);

lpDDSurfaceDesc เป็น แอดเดรส ของstructure DDSURFACEDESC ที่ใช้ของความต้องการและคุณสมบัติของ Surface

lplpDDSurface เป็น แอดเดรส ของ พอยต์เตอร์ของSurface ที่จะสร้าง

HRESULT ผลลัพท์ของการเรียกฟังก์ชันถ้าสำเหร็จจะ ส่งค่ากลับมาเท่ากับ DD_OK

IDirectDrawSurface::GetAttachedSurface

การสร้าง Surface ที่มีความซับซ้อน(complex) เช่นสามารถฟลิปได้จะต้องกำหนดโครงสร้างของ DDSURFACEDESC ให้เรียบร้อยก่อน หลังจากนั้นจะต้องทำการนำ surface ที่ซับซ้อนไปเชื่อมต่อกันโดยใช้

HRESULT GetAttachedSurface(**LPDDSCAPS** lpDDSCaps,

LPDIRECTDRAWSURFACE2 FAR * lplpDDAttachedSurface);

lpDDSCaps เป็น แอดเดรส ของstructure ของ DDSCAPS ซึ่งจะใส่ค่าความสามารถของฮาร์แวร์ที่มีต่อ Surface

lplpDDAttachedSurface เป็น แอดเดรส ของพอยต์เตอร์ของ Surface ที่จะนำไป attach

HRESULT ผลลัพท์ของการเรียกฟังก์ชันถ้าสำเหร็จจะ ส่งค่ากลับมาเท่ากับ DD_OK

ตัวอย่างการใช้งาน(การสร้าง Primary Surfaceพร้อมกับ Back Buffer และนำ Back Buffer ไปติดกับ Primary Surface)

```
ddsd.dwSize = sizeof( ddsd );
ddsd.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE |
                      DDSCAPS_FLIP |
                      DDSCAPS_COMPLEX;
ddsd.dwBackBufferCount = 1;
ddrval = lpDD->CreateSurface( &ddsd, &lpDDSPrimary, NULL );
if( ddrval != DD_OK ){
 return initFail(hwnd);
}
ddscaps.dwCaps = DDSCAPS_BACKBUFFER;
```

```
ddrval = lpDDSPrimary->GetAttachedSurface(&ddscaps, &lpDDSBack);
```

DDLoadBitmap (ddutil)

การสร้าง Offscreen Surface

ทำเหมือนกับที่ใช้สร้าง Primary Surface แต่จะใช้ฟังก์ชันใน ddutils.cpp ช่วย

```
lpDDSOffscreen = DDLoadBitmap(lpDD, szBitmap, 0, 0);
```

lpDD เป็นแอดเดรสของออปเจ็ก DirectDraw

lpDDOffscreen เป็นแอดเดรสของออปเจ็ก offscreen surface

DDLoadPalette (ddutil)

สร้างและโหลด Palette ให้กับออปเจ็ก Palette

จะใช้ฟังก์ชันที่ ddutils.cpp ให้มาเพราะถ้าใช้ฟังก์ชันของ DirectDraw โดยตรงจะซับซ้อนเกินไป

```
lpDDPal = DDLoadPalette(lpDD, szBitmap);
```

lpDDPal เป็นแอดเดรสของออปเจ็ก palette

lpDD เป็นแอดเดรสของออปเจ็ก DirectDraw

szBitmap เป็น string ที่เป็นชื่อของรูปที่เราจะนำ palette มาใช้

IDirectDrawSurface::SetPalette

กำหนดให้ใช้ palette ให้กับ surface

```
HRESULT SetPalette(LPDIRECTDRAWPALETTE lpDDPal);
```

lpDDPal เป็นแอดเดรสของออปเจ็ก palette

HRESULT ผลลัพท์ของการเรียกฟังก์ชันถ้าสำเหร็จจะส่งค่ากลับมาเท่ากับ DD_OK

ตัวอย่างการใช้งาน

```
lpDDSPrimary->SetPalette(lpDDPal); // กำหนด palette ให้กับการแสดงผล
```

DDSetColorKey(ddutils)

กำหนด color key ให้กับ surface โดยใช้ ฟังก์ชันใน ddutils.cpp ช่วย

```
DDSetColorKey(lpDDSOffscreen, RGB(0,0,0));
```

lpDDSOffscreen เป็นแอดเดรสของออปเจ็ก offscreen surface

RGB(0,0,0) เป็นค่าของสีในโหมด RGB ที่จะนำมาใช้เป็น color key ในที่นี้จะใช้มาโคร RGB()ของวินโดว์เพื่อแสดงสี

IDirectDrawSurface::BltFast

ที่กล่าวทั้งหมดเป็นการสร้างเมื่อเริ่มต้นต่อไปจะเป็นการ blit เพื่อทำการทำสำเนาส่วนของ surface ไปยังอีก surface หนึ่ง ณ ตำแหน่งที่ต้องการ

BltFast จะมีความเร็วในการ blit สูงที่สุดแต่จะไม่ค่อยมีฟังก์ชันการทำงานมากนัก

HRESULT BltFast(**DWORD** dwX, **DWORD** dwY,
LPDIRECTDRAWSURFACE2 lpDDSrcSurface, **LPRECT** lpSrcRect,
DWORD dwTrans);

dwX และ dwY ตำแหน่งในแนวแกน X และ Y ของ Surface ปลายทางที่จะ Blit

dwTrans ชนิดของการวาด

HRESULT ผลลัพท์ของการเรียกฟังก์ชันถ้าสำเหร็จจะส่งค่ากลับมาเท่ากับ DD_OK

IDirectDrawSurface::Blt

Blt จะช้ากว่า BltFast แต่สามารถทำเอฟเฟกท์หรือฟังก์ชันได้มากกว่าโดยกำหนดได้ใน lpDDBltFx

HRESULT Blt(**LPRECT** lpDestRect, **LPDIRECTDRAWSURFACE2** lpDDSrcSurface,
LPRECT lpSrcRect, **DWORD** dwFlags, **LPDDBLTFX** lpDDBltFx);

lpDestRect แอดเดรสของขอบเขตสี่เหลี่ยมที่จะวาด ณปลายทาง

lpSrcRect แอดเดรสของขอบเขตสี่เหลี่ยมที่ทำสำเนาจากต้นทาง ถ้าขอบเขตการวาดจากต้นทางและปลายทางไม่เท่ากันจะทำให้สามารถย่อขยายหรือยืดภาพได้

lpDDSrcSurface แอดเดรสของออปเจ็กต้นทางที่เราจะทำการทำสำเนาเพื่อจะวาด

dwFlags ใช้กำหนดคุณสมบัติของการ blit

lpDDBltFx แอดเดรสของโครงสร้าง DDBLTFX จะใช้กำหนดถึงเทคนิคในการ blit

HRESULT ผลลัพท์ของการเรียกฟังก์ชันถ้าสำเหร็จจะส่งค่ากลับมาเท่ากับ DD_OK

ตัวอย่างการใช้งาน

```
ddrval = lpDDSBack->BltFast( 0, 0, lpDDSOne,
        &rcRect,
        DBLTFAST_NOCOLORKEY );
```

IDirectDrawSurface::Restore & DDReLoadBitmap(ddutil)

การโหลดภาพขึ้นมาใหม่ในกรณีที่ข้อมูลใน surface สูญหายและต้องทำการโหลดขึ้นมาใหม่ในโปรเจ็กใช้ฟังก์ชันใน ddutil.cpp ช่วย

HRESULT Restore();

HRESULT ผลลัพท์ของการเรียกฟังก์ชันถ้าสำเหร็จจะส่งค่ากลับมาเท่ากับ DD_OK

ตัวอย่างการใช้งาน

```
ddrval = lpDDSOne->Restore();
if( ddrval == DD_OK )
{
    DDReLoadBitmap(lpDDSOffscreen, szBitmap);
}
```

IDirectDrawSurface::Flip

การฟลิปจาก BackBuffer ไปเป็น FrontBuffer(Primary Surface)

HRESULT Flip(
LPDIRECTDRAWSURFACE lpDDSurfaceTargetOverride,
DWORD dwFlags
);

lpDDSurfaceTargetOverride เป็นแอดเดรสของออปเจ็กที่จะทำการฟลิป โดยจะทำการลัดคิวแต่ต้องเป็นออปเจ็กที่อยู่ในคิวเท่านั้น ถ้าไม่มีการลัดคิวให้ใส่ค่าเป็น NULL

dwFlags เป็นค่ากำหนดสถานะของการฟลิป ถ้าไม่ใช้ให้เป็น 0

HRESULT ผลลัพท์ของการเรียกฟังก์ชันถ้าสำเหร็จจะส่งค่ากลับมาเท่ากับ DD_OK

ตัวอย่างการใช้งาน

```
ddrval = lpDDSPrimary->Flip( NULL, 0 );
```

ภาคผนวก ค
ฟังก์ชันที่ใช้เขียนโปรแกรมโดยใช้ DirectInput

ฟังก์ชันที่จะนำมาใช้ในส่วนนี้จะเป็นฟังก์ชันสำหรับการติดต่อคีย์บอร์ดโดยใช้ DirectInput แต่จะไม่แสดงการติดต่ออุปกรณ์ชนิดอื่นเพราะคล้ายกัน และเพื่อการกระทัดรัด สำหรับผู้สนใจสามารถหารายละเอียดได้จากเอกสารที่มากับ DirectX SDK

การติดต่อ

การใช้ไลบรารีของ DirectInput ควรประกาศอินเตอร์เฟสของ DirectInput ด้วย(dinput.h) และตั้งให้ path ของ comlier ชี้ไปที่ไฟล์ dinput.libและ dxguid.lib ด้วย

```
#include <dinput.h>
```

ตัวแปรที่ใช้

การประกาศตัวแปรที่เป็นพอยต์เตอร์ของออปเจ็ก ส่วนใหญ่จะประกาศเป็นแบบโกลบอล

```
LPDIRECTINPUT       g_pdi;
LPDIRECTINPUTDEVICE   g_pKeyboard;
```

DirectInputCreate

การสร้างออปเจ็กของ DirectInput

DirectInputCreate

```
HRESULT DirectInputCreate(HINSTANCE hinst,
        DWORD dwVersion, LPDIRECTINPUT * lplpDirectInput,
        LPUNKNOWN punkOuter);
```

hinst แฮนด์เดิลของวินโดว์

dwVersion เวอร์ชันของ DirectInput

lplpDirectInput แอดเดรสของพอยต์เตอร์ของออปเจ็กDirectInput

punkOuter เป็น แอดเดรสของออปเจ็กของ IUnKnown สำหรับ OLE ในที่นี้จะไม่ใช้(NULL)

HRESULT ผลลัพท์ของการเรียกฟังก์ชันถ้าสำเหร็จจะส่งค่ากลับมาเท่ากับ DI_OK

ตัวอย่างการใช้งาน

```
ddrval = DirectInputCreate(g_hinst,0x0300, &g_pdi, NULL);
```

IDirectInput::CreateDevice

การสร้างออปเจ็กของอุปกรณ์อินพุต

HRESULT CreateDevice(**REFGUID** rguid,

LPDIRECTINPUTDEVICE *lplpDirectInputDevice,

LPUNKNOWN pUnkOuter);

rguid IDสำหรับการอ้างอิงชนิดของอุปกรณ์

lplpDirectInputDevice แอดเดรสของพอยต์เตอร์ของออปเจ็กอุปกรณ์

punkOuter เป็น แอดเดรสของออปเจ็กของ IUnKnown สำหรับ OLE ในที่นี้จะไม่ใช้(NULL)

ตัวอย่างการใช้งาน

ddrval = g_pdi->CreateDevice(GUID_SysKeyboard, &g_pKeyboard, NULL);

IDirectInput::SetDataFormat

การกำหนดรูปแบบข้อมูลที่จะรับอินพุต

HRESULT SetDataFormat(**LPCDIDATAFORMAT** lpdf);

lpdf แอดเดรสของโครงสร้างที่รูปแบบการรับข้อมูลที่อินพุต

ตัวอย่างการใช้งาน

ddrval = g_pKeyboard->SetDataFormat(&c_dfDIKeyboard);

IDirectInputDevice::SetCooperativeLevel

กำหนดพฤติกรรมของอุปกรณ์รับอินพุต

HRESULT SetCooperativeLevel(

HWND hwnd,

DWORD dwFlags

hwnd แฮนด์เดิล ของวินโดว์

dwFlags แฟลกสำหรับกำหนดพฤติกรรมของอุปกรณ์

ตัวอย่างการใช้งาน

ddrval = g_pKeyboard->SetCooperativeLevel(hwnd,

DISCL_NONEXCLUSIVE | DISCL_FOREGROUND);

IDirectInputDevice::Acquire

อนุญาติให้ใช้อุปกรณ์อินพุตตัวนั้น

HRESULT Acquire();

ตัวอย่างการใช้งาน

g_pKeyboard->Acquire();

IDirectInputDevice::GetDeviceState

อ่านค่าอินพุตจากอุปกรณ์

HRESULT GetDeviceState(**DWORD** cbData, **LPVOID** lpvData);

cbData ขนาดของโครงสร้างที่จะเก็บค่าอินพุต

lpvData แอดเดรส ของโครงสร้างที่ใช้เก็บข้อมูล

ตัวอย่างการใช้งาน

hr = g_pKeyboard->GetDeviceState(sizeof(diks), &diks);

KEYDOWN(macroสำหรับคีย์บอร์ดเท่านั้น)

ใช้ตรวจสอบค่าอินพุต(คีย์บอร์ดเท่านั้น)ว่ามีปุ่มไหนกด

#define KEYDOWN(name,key) (name[key] & 0x80)

ตัวอย่างการใช้งาน

KEYDOWN(diks,DIK_DOWN)

ภาคผนวก ง

ฟังก์ชันที่ใช้เขียนโปรแกรมโดยใช้ Direct3D(Retain Mode)

ในส่วนนี้จะไม่ได้กล่าวถึงโหมด Immediate เพราะว่าในโปรเจ็กนี้กำหนดขอบเขตการศึกษาไว้เฉพาะโหมด Retain ดังนั้นฟังก์ชันต่างๆจะเป็นของโหมด Retain

การติดต่อ

การใช้ไลบรารีของ Direct3D ควรประกาศอินเตอร์เฟสของ Direct3D ด้วย(d3drm.h) และตั้งให้ path ของ comlier ชี้ไปที่ไฟล์ d3drm.libและ dxguid.lib ด้วย

```
#include <d3drmwin.h>
```

ตัวแปรที่ใช้

การประกาศตัวแปรที่เป็นพอยต์เตอร์ของออปเจ็ก ส่วนใหญ่จะประกาศเป็นแบบโกลบอล

```
LPDIRECT3DRM lpD3DRM;        //พอยต์เตอร์ของออปเจ็กของ Direct3DRM
LPDIRECTDRAWCLIPPER lpDDClipper;//พอยต์เตอร์ของ DirectDrawClipper ใช้ในกรณีเปโหมด
        วินโดว์
LPDIRECT3DRMDEVICE dev;    //พอยต์เตอร์ของออปเจ็กDirect3DRM device
LPDIRECT3DRMVIEWPORT view; // พอยต์เตอร์ของ viewport(ช่องสำหรับมองหรือแสดงผล)
LPDIRECT3DRMFRAME scene;   // เฟรม ต้นกำเนิดของเฟรม ทั้งหมด
LPDIRECT3DRMFRAME camera;  // เฟรม สำหรับ viewport
```

EnumDrivers(จากตัวอย่าง)

ฟังก์ชันนี้เป็นฟังก์ชันจากตัวอย่างเนื่องจากมีความซับซ้อนมากให้นำไปใช้ได้เลยรายละเอียดดูในตัวอย่าง hello_world

```
BOOL EnumDrivers(HWND win);
```

win แฮนด์เดิลของวินโดว์

BOOL ถ้าสำเหร็จจะแสดงค่าเป็น TRUE ไม่เช่นนั้น FALSE

Direct3DRMCreate

สร้างออปเจ็กของ Direct3DRM

```
HRESULT Direct3DRMCreate(LPDIRECT3DRM FAR * lplpD3DRM);
```

lplpD3DRM แอดเดรสของพอยต์เตอร์ของ Direct3DRM

ตัวอย่างการใช้งาน

Direct3DRMCreate(&lpD3DRM);

IDirect3DRM::CreateFrame

การสร้างเฟรม

HRESULT CreateFrame(

LPDIRECT3DRMFRAME lpD3DRMFrame,

LPDIRECT3DRMFRAME* lplpD3DRMFrame

);

lpD3DRMFrame พอยต์เตอร์ของเฟรมต้นให้กำเนิด ถ้าต้องการสร้างเป็นเฟรมแรกสุดในตรงนี้เป็น NULL

lplpD3DRMFrame พอยต์เตอร์ของเฟรมที่จะสร้าง

ตัวอย่างการใช้งาน

สร้างเฟรมแรกสุด

lpD3DRM->CreateFrame(NULL, &myglobs.scene);

สร้างเฟรมทั่วไป(sene เป็นผู้ให้กำเนิดและเฟรม camera ถูกสร้างขึ้น)

lpD3DRM->CreateFrame(myglobs.scene,

&myglobs.camera);

IDirect3DRM::CreateDeviceFromClipper

การสร้าง device จาก clipper

HRESULT CreateDeviceFromClipper(

LPDIRECTDRAWCLIPPER lpDDClipper,

LPGUID lpGUID,

int width, **int** height,

LPDIRECT3DRMDEVICE * lplpD3DRMDevice

);

lpDDClipper แอดเดรสของออปเจ็ก DirectDrawClipper

lpGUID แอดเดรสของ globally unique identifier (GUID)ในที่นี้ให้เป็น NULL ได้

width และ height ความกว้างและความสูงที่จะสร้าง

lplpD3DRMDevice แอดเดรสของพอยต์เตอร์ของออปเจ็กของ Direct3DRMDevice ที่จะสร้าง

ตัวอย่างการใช้งาน

```
lpD3DRM->CreateDeviceFromClipper( lpDDClipper,
    &myglobs.DriverGUID[driver], width, height, &myglobs.dev);
```

IDirect3DRM::CreateViewport

```
HRESULT CreateViewport(
        LPDIRECT3DRMDEVICE lpDev,
        LPDIRECT3DRMFRAME lpCamera,
        DWORD dwXPos, DWORD dwYPos,
        DWORD dwWidth, WORD dwHeight,
        LPDIRECT3DRMVIEWPORT* lplpD3DRMViewport
);
```

lpDev พอยต์เตอร์ของวิวพอร์ตที่จะสร้าง

lpCamera เฟรม ที่ใช้ในการอ้างอิงวิวพอร์ต

dwXPos, dwYPos, dwWidth, dwHeight ตำแหน่งและขนาดของวิวพอร์ต

lplpD3DRMViewport แอดเดรสของพอยต์เตอร์ของ Direct3DRMViewport ที่จะสร้าง

ตัวอย่างการใช้งาน

```
rval = lpD3DRM->CreateViewport( myglobs.dev,
    myglobs.camera, 0, 0, width, height, &myglobs.view);
```

IDirect3DRMViewport::SetBack

ใช้กำหนดของเขตของการเรนเดอร์ในระยะห่างที่สุด

```
HRESULT SetBack(D3DVALUE rvBack);
```

rvBack ตำแหน่งของ clipping plane ที่จะกำหนด

ตัวอย่างการใช้งาน

```
rval = myglobs.view->SetBack( D3DVAL(5000));
```

IDirect3DRMDevice::SetQuality

การกำหนดคุณภาพของการเรนเดอร์โดยค่าเริ่มแรกจะเป็นโหมด Flat

```
HRESULT SetQuality (D3DRMRENDERQUALITY rqQuality);
```

ตัวอย่างการใช้งาน

```
rval = myglobs.dev->SetQuality(
    D3DRMLIGHT_ON | D3DRMFILL_SOLID | D3DRMSHADE_GOURAUD);
```

IDirect3DRM::CreateLightRGB

การสร้างออปเจ็กแสง

```
HRESULT CreateLightRGB(
        D3DRMLIGHTTYPE ltLightType,
        D3DVALUE vRed, D3DVALUE vGreen, D3DVALUE vBlue,
        LPDIRECT3DRMLIGHT* lplpD3DRMLight
);
```

ltLightType ประเภทของแสง

vRed,vGreen,vBlue ค่าของสีของแสดงในโหมด RGB

lplpD3DRMLight แอดเดรสของพอยต์เตอร์ของออปเจ็กแสงที่จะทำการสร้าง

ตัวอย่างการใช้งาน

```
lpD3DRM->CreateLightRGB( D3DRMLIGHT_DIRECTIONAL,
    D3DVAL(0.9), D3DVAL(0.9), D3DVAL(0.9), lplpLight1);
```

IDirect3DRMFrame::AddLight

การนำออปเจ็กแสงไปอ้างอิงกับเฟรม

```
HRESULT AddLight( LPDIRECT3DRMLIGHT lpD3DRMLight);
```

lpD3DRMLight พอยต์เตอร์ของเฟรมที่เราจะนำไปอ้างอิง

ตัวอย่างการใช้งาน

```
lpLightFrame1->AddLight( *lplpLight1);
```

IDirect3DRMFrame::SetPosition

การกำหนดตำแหน่งของเฟรม โดยจะทำการอ้างอิงกับเฟรมอื่น

```
HRESULT SetPosition(
        LPDIRECT3DRMFRAME lpRef,
        D3DVALUE rvX, D3DVALUE rvY, D3DVALUE rvZ
);
```

lpRef พอยต์เตอร์ของเฟรมที่เราจะทำการอ้างอิง

rvX,rvY ,rvZ พิกัด x y z ที่เป็นตำแหน่งที่ต้องการ

ตัวอย่างการใช้งาน

```
lpWorld_frame->SetPosition( lpScene,
    D3DVAL(0.0), D3DVAL(0.0), D3DVAL(15));
```

IDirect3DRMFrame::SetOrientation

การกำหนดทิศทางของเฟรม โดยอ้างอิงจาก เฟรม อื่น แกน Z ใช้แทนทิศทางด้านหน้า แกน Y ใช้แทนทิศบน

```
HRESULT SetOrientation(
        LPDIRECT3DRMFRAME lpRef,
        D3DVALUE rvDx, D3DVALUE rvDy, D3DVALUE rvDz,
        D3DVALUE rvUx, D3DVALUE rvUy, D3DVALUE rvUz
);
```

lpRef เฟรม ที่นำมาอ้างอิง

rvDx,rvDy,rvDz เวกเตอร์ของแกน Z

rvUx,rvUy,rvUz เวกเตอร์ของแกน Y

ตัวอย่างการใช้งาน

```
lpCamera->SetOrientation( lpScene,      D3DVAL(0.0), D3DVAL(0.0), D3DVAL(1),
                          D3DVAL(0.0), D3DVAL(1), D3DVAL(0.0));
```

IDirect3DRMFrame::SetRotation

กำหนดการหมุนอัตโนมัติเมื่อใช้คำสั่ง IDirect3DRM::Tick หรือ IDirect3DRMFrame::Move เฟรม ก็จะหมุนตามค่าที่ตั้งไว้

HRESULT SetRotation(**LPDIRECT3DRMFRAME** lpRef,
D3DVALUE rvX, **D3DVALUE** rvY, **D3DVALUE** rvZ,
D3DVALUE rvTheta
);

lpRef เฟรมที่จะทำการอ้างอิงในการหมุน

rvX,rvY,rvZ แกนที่จะทำการหมุนรอบ

rvTheta มุมในการหมุนมีหน่วยเป็นเรเดียน

ตัวอย่างการใช้งาน

```
lpWorld_frame->SetRotation( lpScene,
    D3DVAL(0.0), D3DVAL(0.1), D3DVAL(0.0), D3DVAL(0.05));
```

IDirect3DRM::CreateMeshBuilder

HRESULT CreateMeshBuilder(**LPDIRECT3DRMMESHBUILDER***
lplpD3DRMMeshBuilder);

lplpD3DRMMeshBuilder แอดเดรสของพอยต์เตอร์ของออปเจ็ก Direct3DRMMeshBuilder

ตัวอย่างการใช้งาน

```
lpD3DRM->CreateMeshBuilder( lplpSphere3_builder);
```

IDirect3DRMMeshBuilder::Load

โหลดไฟล์ .x ที่เป็นโครงสร้างของ mesh

HRESULT Load(
LPVOID lpvObjSource,
LPVOID lpvObjID,
D3DRMLOADOPTIONS d3drmLOFlags,
D3DRMLOADTEXTURECALLBACK d3drmLoadTextureProc,
LPVOID lpvArg
);

lpvObjSource ไฟล์ .x ที่จะทำการโหลด

lpvObjID ชื่อหรือตำแหน่งของออปเจ็กที่จะทำการโหลดซึ่งจะอาศัยค่าจากแฟลกใน d3drmLOFlags ถ้าไม่ต้องการเจาะจงให้เป็น NULL ก็ได้

d3drmLOFlags ใช้กำหนดออปชันของการโหลด mesh

d3drmLoadTextureProc callback ฟังก์ชันสำหรับการโหลด texture แบบพิเศษถ้าใช้แบบปรกติให้เป็น NULL

lpvArg แอดเดรสของข้อมูลที่จะส่งให้ callback ฟังก์ชัน

ตัวอย่างการใช้งาน

```
lpSphere3_builder->Load("sphere3.x", NULL, D3DRMLOAD_FROMFILE, NULL, NULL);
```

IDirect3DRMMeshBuilder::Scale

การกำหนดขนาดของ mesh

HRESULT Scale(**D3DVALUE** sx , **D3DVALUE** sy , **D3DVALUE** sz);

sx,sy,sz ขนาดในแนวแกน x y z

ตัวอย่างการใช้งาน

```
lpSphere3_builder->Scale(D3DVAL(0.3), D3DVAL(0.3), D3DVAL(0.3));
```

IDirect3DRMMeshBuilder::SetColorRGB

HRESULT SetColorRGB(**D3DVALUE** red,**D3DVALUE** green,**D3DVALUE** blue);

red,green,blue สีแดง เขียวน้ำเงินในโหมด RGB

ตัวอย่างการใช้งาน

```
lpSphere3_builder->SetColorRGB(D3DVAL(1), D3DVAL(1), D3DVAL(1));
```

IDirect3DRMMesh::GetBox

การหาขนาดมากสุดและน้อยสุดที่ใช้ในการอ้างอิงของ mesh

HRESULT GetBox(**D3DRMBOX** * lpD3DRMBox);

lpD3DRMBox พอยต์เตอร์ของstructure D3DRMBOX

ตัวอย่างการใช้งาน

```
sphere3_builder->GetBox( &box);
```

IDirect3DRM::CreateWrap

```
HRESULT CreateWrap(
        D3DRMWRAPTYPE type,
        LPDIRECT3DRMFRAME lpRef,
        D3DVALUE ox, D3DVALUE oy, D3DVALUE oz,
        D3DVALUE dx, D3DVALUE dy, D3DVALUE dz,
        D3DVALUE ux, D3DVALUE uy, D3DVALUE uz,
        D3DVALUE ou, D3DVALUE ov,
        D3DVALUE su, D3DVALUE sv,
        LPDIRECT3DRMWRAP* lplpD3DRMWrap
);
```

type ชนิดของ Wrap

lpRef เฟรม ที่จะทำการอ้างอิง

ox,oy, oz จุดกำเนิดของ Wrap

dx,dy ,dz แกน Z ของ Wrap

ux,uy, uz แกน Y ของ Wrap

ou,ov จุดกำเนิดของ texture

su,sv อัตราส่วนของ texture

lplpD3DRMWrap แอดเดรสของพอยต์เตอร์ที่เราจะทำการสร้าง Wrap

ตัวอย่างการใช้งาน

```
lpD3DRM->CreateWrap
    (D3DRMWRAP_CYLINDER , NULL,
    D3DVAL(0.0), D3DVAL(0.0), D3DVAL(0.0),          //origin of wrap
    D3DVAL(0.0), D3DVAL(1.0), D3DVAL(0.0),          //z axis
    D3DVAL(0.0), D3DVAL(0.0), D3DVAL(1.0),          //y axis
    D3DVAL(0.0), D3DDivide(miny, height),           //origin in texture
    D3DVAL(1.0), D3DDivide(D3DVAL(1.0), height),//scale factor in texture
```

lpWrap);

IDirect3DRMWrap::Apply

HRESULT Apply(**LPDIRECT3DRMOBJECT** lpObject);

lpObject กำหนดรูปแบบการ Wrap ให้กับออปเจ็กของ Direct3D

ตัวอย่างการใช้งาน

(*lpWrap)->Apply((LPDIRECT3DRMOBJECT)

sphere3_builder);

IDirect3DRM::LoadTexture

โหลด textureจากไฟล์

HRESULT LoadTexture(

const char * lpFileName,

LPDIRECT3DRMTEXTURE* lplpD3DRMTexture

);

lpFileName ชื่อไฟล์ที่จะโหลด

lplpD3DRMTexture แอดเดรส ของ พอยต์เตอร์ ของออปเจ็ก D3DRMTexture ที่จะโหลด

ตัวอย่างการใช้งาน

lpD3DRM->LoadTexture("tutor.bmp", lplpTex);

IDirect3DRMFace::SetTexture

การกำหนด texture ให้กับ mesh

HRESULT SetTexture(**LPDIRECT3DRMTEXTURE** lpD3DRMTexture);

lpD3DRMTexture lplpD3DRMTexture แอดเดรส ของ พอยต์เตอร์ ของออปเจ็ก D3DRMTexture

ตัวอย่างการใช้งาน

lpSphere3_builder->SetTexture(*lplpTex);

IDirect3DRMFrame::AddVisual

นำออเจ็กสามมิติที่เกิดจาก mesh ที่มีพื้นผิว(texture)แล้วไปอ้างอิงกับเฟรม

HRESULT AddVisual(**LPDIRECT3DRMVISUAL** lpD3DRMVisual);

ตัวอย่างการใช้งาน

```
lpMoon_frame->AddVisual(
    (LPDIRECT3DRMVISUAL) lpSphere3_builder);
```

Release

การคืนหน่วยความจำที่ออปเจ็กจองไว้

ตัวอย่างการใช้งาน

```
lpWorld_frame->Release();          //relase frame
lpSphere3_builder->Release();      //relase Sphear bulider
lpLight1->Release();               //relase ออปเจ็กแสง
lpTex->Release();                  //relase ออปเจ็ก texture
lpWrap->Release();                 //relase ออปเจ็ก Wrap
```

ภาคผนวก จ
ฟังก์ชันที่ใช้เขียนโปรแกรมโดยใช้ DirectSound

โปรแกรมจะใช้ IDirectSound ในการสร้างออปเจ็ก DirectSound เพื่อการติดต่อกับอุปกรณ์เสียง การใช้ IDirectSound จะต้องใช้ dsound.h ร่วมในการเขียนโปรแกรม

DirectSoundCreate

เริ่มต้นการสร้างออปเจกต์ DirectSound เพื่อติดต่อกับอุปกรณ์เสียง

```
HRESULT DirectSoundCreate(
        LPGUID lpGuid,
        LPDIRECTSOUND * ppDS,
        IUnknown FAR * pUnkOuter
);
```

lpGuid พอยต์เตอร์ของ GUID ที่บอกว่าเป็นอุปกรณ์เสียงตัวใด ใช้ค่า NULL เมื่อต้องการใช้อุปกรณ์เสียงของระบบ

ppDS ค่าแอดเดรสของพอยต์เตอร์ของออปเจ็ก DirectSound

pUnkOuter ยังไม่มีการใช้ จะใช้ค่า NULL

ฟังก์ชันทำได้สำเร็จจะได้ค่า DS_OK

หมายเหตุ

เมื่อใช้ฟังก์ชันนี้แล้วจะต้องเรียก IDirectSound::SetCooperativeLevel เป็นฟังก์ชันต่อไป

IDirectSound::SetCooperativeLevel

กำหนดลักษณะการใช้งานของโปรแกรมกับอุปกรณ์เสียง

```
HRESULT SetCooperativeLevel( HWND hwnd, DWORD dwLevel );
```

hwnd ค่า แฮนด์เดิลวินโดวส์ของโปรแกรม

dwLevel ลักษณะการใช้งานมีค่า

DSSCL_EXCLUSIVE โปรแกรมสมารถทำงานได้เหมือนกับ Priority Cooperative Level และเมื่อทำงานเบื้องหน้าจะมีเสียงจากโปรแกรมนี้เท่านั้น

DSSCL_NORMAL โปรแกรมไม่สามารถกำหนดรูปแบบของบัฟเฟอร์เสียงหลัก (primary sound buffer) หรือเขียนบัฟเฟอร์หลัก ทุกโปรแกรมจะใช้รูปแบบเสียงคือ 22kHz, stereo, 8-bit ทำให้การสลับระหว่างโปรแกรมไม่มีการกระตุก

DSSCL_PRIORITY โปรแกรมสามารถควบคุมทรัพยากรต่างของอุปกรณ์เสียงได้ เช่น การผสมเสียง, เปลี่ยนรูปแบบของบัฟเฟอร์เสียงหลัก

DSSCL_WRITEPRIMARY โปรแกรมสามารถติดต่อกับบัฟเฟอร์เสียงหลัก และเมื่อต้องการเล่นเสียงจะต้องเขียนไปยังบัฟเฟอร์หลักโดยตรง

ฟังก์ชันทำได้สำเร็จจะได้ค่า DS_OK

หมายเหตุ

ฟังก์ชันนี้จะต้องเรียกก่อนการเล่นเสียง ควรจะใช้ DSSCL_NORMAL ถ้าเป็นการเล่นเสียงตามปกติ

IDirectSound::GetCaps

เมื่อต้องการสมบัติที่มีของอุปกรณ์เสียง หลังจากที่สร้างออปเจกต์ DirectSound ที่ติดต่อกับอุปกรณ์เสียงแล้ว

HRESULT GetCaps(**LPDSCAPS** lpDSCaps);

lpDSCaps แอดเดรสของตัวแปรแบบ DSCAPS ไว้สำหรับเก็บข้อมูล

ฟังก์ชันทำได้สำเร็จจะได้ค่า DS_OK

IDirectSound::CreateSoundBuffer

การสร้างบัฟเฟอร์สำหรับเก็บข้อมูลเสียง ก่อนที่จะสามารถเล่นเสียง

HRESULT CreateSoundBuffer(

LPCDSBUFFERDESC lpcDSBufferDesc,

LPLPDIRECTSOUNDBUFFER lplpDirectSoundBuffer,

IUnknown FAR * pUnkOuter

);

lpcDSBufferDesc แอดเดรสของตัวแปรชนิด DSBUFFERDESC ซึ่งอธิบายลักษณะของบัฟเฟอร์ที่ต้องการสร้าง เช่น รูปแบบข้อมูล, ขนาด รวมทั้งหน่วยความจำที่จะนำมาใช้เป็นบัฟเฟอร์

lplpDirectSoundBuffer แอดเดรสของพอยต์เตอร์ของออปเจกต์ DirectSoundBuffer ถ้าไม่สามารถสร้างได้จะได้ค่า NULL

pUnkOuter ยังไม่มีการใช้ จะใช้ค่า NULL

ฟังก์ชันทำได้สำเร็จจะได้ค่า DS_OK

หมายเหตุ

ก่อนเล่นเสียงจากบัฟเฟอร์ได้ ต้องมีการกำหนดลักษณะการใช้งาน DirectSound ก่อน ด้วยฟังก์ชัน IDirectSound::SetCooperativeLevel

IDirectSoundBuffer::Lock

การเรียกฟังก์ชันนี้ เพื่อต้องการตำแหน่งที่สามารถเขียนข้อมูลเสียงลงบนบัฟเฟอร์ได้

```
HRESULT Lock(
        DWORD dwWriteCursor,
        DWORD dwWriteBytes,
        LPVOID lplpvAudioPtr1,
        LPDWORD lpdwAudioBytes1,
        LPVOID lplpvAudioPtr2,
        LPDWORD lpdwAudioBytes2,
        DWORD dwFlags
);
```

dwWriteCursor ตำแหน่งที่จะเขียนข้อมูลลงบัฟเฟอร์ ถ้ากำหนด dwFlags มีค่า เป็น DSBLOCK_FROMWRITECURSOR ค่านี้จะไม่นำไปใช้

dwWriteBytes จำนวนข้อมูลที่ต้องการเขียนลงบนบัฟเฟอร์

lplpvAudioPtr1 แอดเดรสของพอยต์เตอร์ตัวแรกของบัฟเฟอร์ ที่สามารถเขียนข้อมูลได้

lpdwAudioBytes1 แอดเดรสของตัวแปรที่เก็บจำนวนข้อมูลที่สามารถเขียนได้ที่ lplpvAudioPtr1 ชี้อยู่ ถ้าค่าที่ได้มีค่าน้อยกว่า dwWriteBytes จะต้องใช้ lplpvAudioPtr2 ในการเขียนครั้งที่สองด้วย

lplpvAudioPtr2 แอดเดรสของพอยต์เตอร์ตัวที่สองของบัฟเฟอร์ ที่สามารถเขียนข้อมูลได้ ถ้ามีค่าเป็น NULL จะได้ว่า lplpvAudioPtr1 สามารถใช้ในการเขียนข้อมูลได้ทั้งหมด

lpdwAudioBytes2 แอดเดรสของตัวแปรที่เก็บจำนวนข้อมูลที่สามารถเขียนได้ที่ lplpvAudioPtr2 ชี้อยู่ ถ้า lplpvAudioPtr2 มีค่าเป็น NULL ค่านี้จะเป็น 0

dwFlags บอกสถานะการเขียน

DSBLOCK_FROMWRITECURSOR เมื่อต้องการเขียนข้อมูลต่อจากครั้งล่าสุด

DSBLOCK_ENTIREBUFFER เมื่อต้องการเขียนทั้งบัฟเฟอร์ ค่า dwWriteBytes จะไม่ต้องนำมาใช้งาน

ฟังก์ชันทำได้สำเร็จจะได้ค่า DS_OK

หมายเหตุ

เมื่อเรียกฟังก์ชันนี้แล้วจะได้ตำแหน่งและจำนวนข้อมูล ที่สามารถเขียนลงบัฟเฟอร์ได้ การที่ได้ตำแหน่งสองตัว เพราะบัฟเฟอร์จะใช้งานแบบวน เมื่อตำแหน่งแรกไม่สามารถเขียนข้อมูลได้หมด ก็ต้องใช้ตัวชี้ตัวที่สอง เพื่อเขียนข้อมูลอีกส่วนหนึ่ง

หลังจากเรียกใช้ IDirectSoundBuffer::Lock แล้วจะต้องเรียกใช้ IDirectSoundBuffer::Unlock เพื่อที่ให้ DirectSound ใช้งานบัฟเฟอร์ได้ต่อไป

IDirectSoundBuffer::Unlock

เพื่อปล่อยบัฟเฟอร์ที่กันไว้

```
HRESULT Unlock(
        LPVOID lpvAudioPtr1,
        DWORD dwAudioBytes1,
        LPVOID lpvAudioPtr2,
        DWORD dwAudioBytes2
);
```

lpvAudioPtr1 แอดเดรสของค่าที่ได้จาก lplpvAudioPtr1 ของ IDirectSoundBuffer::Lock

dwAudioBytes1 จำนวนข้อมูลที่สามารถเขียนได้ที่ใช้กับ lpvAudioPtr1

lpvAudioPtr2 แอดเดรสของค่าที่ได้จาก lplpvAudioPtr2 ของ IDirectSoundBuffer::Lock

dwAudioBytes2 จำนวนข้อมูลที่สามารถเขียนได้ที่ใช้กับ lpvAudioPtr2

ฟังก์ชันทำได้สำเร็จจะได้ค่า DS_OK

หมายเหตุ

โปรแกรมจะส่งของพอยต์เตอร์ทั้งสองตัวที่ได้จาก IDirectSoundBuffer::Lock และรวมทั้งค่าจำนวนข้อมูลที่เขียนลงบัฟเฟอร์ และจะต้องไม่กันบัฟเฟอร์ไว้นานเกินไป

IDirectSoundBuffer::Play

เล่นเสียงในบัฟเฟอร์

HRESULT Play(

DWORD dwReserved1,

DWORD dwReserved2,

DWORD dwFlags

);

dwReserved1 สงวนไว้ ต้องกำหนดให้เป็น 0

dwReserved2 สงวนไว้ ต้องกำหนดให้เป็น 0

dwFlags กำหนดการเล่นเสียง มีค่าเป็น

DSBPLAY_LOOPING เล่นเสียงแบบวน เมื่อเล่นจนสุดบัฟเฟอร์แล้วจะวนกลับมาเล่นตั้งแต่ต้นบัฟเฟอร์อีก จนกว่าจะสั่งให้หยุด

ฟังก์ชันทำได้สำเร็จจะได้ค่า DS_OK

หมายเหตุ

ถ้าบัฟเฟอร์ถูกเล่นอยู่แล้ว บัฟเฟอร์จะทำการเล่นต่อไป และจะใช้ค่าเฟลกตัวใหม่

IDirectSoundBuffer::Stop

หยุดการเล่นเสียงจากบัฟเฟอร์

HRESULT Stop();

หมายเหตุ

เมื่อสั่งหยุดการเล่นเสียง ตำแหน่งที่หยุดนี้จะเป็นตำแหน่งเริ่มต้นการเล่นเสียงในครั้งต่อไป

ภาคผนวก ฉ

ฟังก์ชันที่ใช้เขียนโปรแกรมโดยใช้ DirectPlay

การสร้างออปเจ็ก DirectPlay เพื่อสามารถติดต่อบนเน็ตเวิร์กที่บริการให้บนเครื่องนั้นๆ เราจะต้องต่อเน็ตเวิร์กเสียก่อน และจะต้องมีเครื่องอีกเครื่องหนึ่งในระบบเน็ตเวิร์กที่สามารถติดต่อกันได้ ซึ่งจะต้องทำโดยใช้ฟังก์ชันเหล่านี้

DirectPlayEnumerate

ตรวจหาบริการที่มีให้ใช้

```
HRESULT WINAPI DirectPlayEnumerate(
        LPDPENUMDPCALLBACK lpEnumCallback,
        LPVOID lpContext
);
```

lpEnumCallback พอยต์เตอร์ของฟังก์ชันที่จะถูกเรียกเมื่อมีการพบบริการที่มีให้

lpContext แอดเดรสของค่าที่จะส่งไปให้ฟังก์ชันในแต่ละครั้งที่มีการเรียก

ฟังก์ชันทำได้สำเร็จจะได้ค่า DP_OK

หมายเหตุ

ฟังก์ชันเพื่อตรวจหาบริการที่มีบนระบบทั้งหมด แม้ว่าใช้บริการบางอย่างใช้ไม่ได้ เช่น TAPI ที่มีบนระบบสำหรับใช้กับโมเด็ม แต่ว่าเครื่องไม่มีโมเด็มต่ออยู่

DirectPlayCreate

```
HRESULT WINAPI DirectPlayCreate(
        LPGUID lpGUIDSP,
        LPDIRECTPLAY FAR *lplpDP,
        IUnknown *lpUnk
);
```

lpGUIDSP แอดเดรสของตัวแปรที่เก็บ GUID ของบริการที่ต้องการ

lplpDP แอดเดรสของพอยต์เตอร์ของออปเจกต์ DirectPlay

lpUnk ยังไม่มีการใช้ จะใช้ค่า NULL

ฟังก์ชันทำได้สำเร็จจะได้ค่า DP_OK

หมายเหตุ

ควรใช้ DirectPlayEnumerate ก่อนเรียกฟังก์ชันนี้ เพื่อสามารถกำหนดบริการที่ต้องการได้

IDirectPlay::EnumSessions

ตรวจหา session ที่มีบนบริการ ถ้าอยู่ใน session อยู่แล้ว เมื่อเรียกฟังก์ชันนี้จะเกิดการผิดพลาด

```
HRESULT EnumSessions(
        LPDPSESSIONDESC lpsd,
        DWORD dwTimeout,
        LPDPENUMSESSIONSCALLBACK lpEnumSessionsCallback,
        LPVOID lpContext,
        DWORD dwFlags
);
```

lpsd พอยต์เตอร์ของตัวแปร DPSESSIONDESC เก็บข้อมูลเกี่ยวกับ session ที่ต้องการค้นหา ค่า guidApplication ที่กำหนดเป็น GUID ของโปรแกรมที่ต้องการ หรือใช้ค่า GUID_NULL เพื่อต้องการหาทุก session

dwTimeout กำหนดเวลาที่ใช้ในการค้นหา ใช้ค่า 0 เมื่อต้องการให้ DirectPlay กำหนดค่านี้ให้

lpEnumSessionsCallback พอยต์เตอร์ของฟังก์ชันที่จะถูกเรียกเมื่อพบแต่ละ session

lpContext แอดเดรสของค่าที่จะส่งไปให้ฟังก์ชันในแต่ละครั้งที่มีการเรียก

dwFlags สามารถใช้เป็น 0 จะมีค่าเท่ากับ DPENUMSESSIONS_AVAILABLE ซึ่งจะได้ session ที่สามารถมีผู้เล่นเพิ่มได้

IDirectPlay::Open

เพื่อเข้าไปใน session ที่ต้องการ โดย session นี้ได้จากฟังก์ชัน IDirectPlay::EnumSessions หรือสามารถสร้าง session ขึ้นมาใหม่และให้ผู้เล่นอื่นเข้ามาได้

```
HRESULT Open(
        LPDPSESSIONDESC lpsd,
        DWORD dwFlags
```

);

lpsd แอดเดรสของ DPSESSIONDESC ที่เก็บข้อมูลเกี่ยวกับ session ที่ต้องการสร้างหรือต้องการเข้าไป

dwFlags มีค่าได้ดังนี้

DPOPEN_CREATE สร้าง session ใหม่ แล้วจะมีสิทธิในการจัดการ session

DPOPEN_JOIN เข้าไปใน session จากนั้นสามารถสร้างผู้เล่น และรับ/ส่งข้อมูลได้

ฟังก์ชันทำได้สำเร็จจะได้ค่า DP_OK

หมายเหตุ

เมื่อเข้าไปใน session ได้แล้ว จากนั้นจะต้องสร้างผู้เล่น แล้วจึงสามารถรับ/ส่งข้อมูลกับผู้เล่นอื่นได้ การสร้างผู้เล่นโดยใช้ฟังก์ชัน IDirectPlay3::CreatePlayer.

IDirectPlay::Close

ออกจาก session ที่กำลังอยู่

HRESULT Close( );

ฟังก์ชันทำได้สำเร็จจะได้ค่า DP_OK

หมายเหตุ

ผู้เล่นที่ถูกสร้างจะถูกยกเลิกการใช้งาน และจะมี DPMSG_DELETEPLAYERFROMGROUP แล DPMSG_DESTROYPLAYERORGROUP ส่งไปยังผู้เล่นที่เหลืออยู่ใน session

IDirectPlay::EnumPlayers

ตรวจหาผู้เล่นใน session

HRESULT EnumPlayers(

LPGUID lpguidInstance,

LPDPENUMPLAYERSCALLBACK lpEnumPlayersCallback,

LPVOID lpContext,

DWORD dwFlags

);

lpguidInstance ใช้ค่า NULL เพื่อตรวจหาผู้เล่นใน session ที่กำลังอยู่ ถ้าไม่ได้เข้าไปอยู่ใน session ค่านี้จะเป็นตัวบอก session ที่ต้องการตรวจหาผู้เล่น ที่ได้จาก IDirectPlay::EnumSessions

lpEnumPlayersCallback พอยต์เตอร์ของฟังก์ชันที่จะถูกเรียกเมื่อมีการพบผู้เล่น

lpContext แอดเดรสของค่าที่จะส่งไปให้ฟังก์ชันในแต่ละครั้งที่มีการเรียก

dwFlags บอกวิธีในการค้นหาผู้เล่น ถ้าเป็น 0 หมายถึงหาผู้เล่นทุกคนภายใน session ที่โปรแกรมนี้อยู่ ซึ่งที่จำเป็นต้องใช้ คือ

DPENUMPLAYERS_ALL ค้นหาผู้เล่นทุกคนที่อยู่ใน session ปัจจุบัน

DPENUMPLAYERS_SESSION ค้นหาผู้เล่นใน session ที่กำหนดในค่า lpguidInstance ใช้ค่านี้ได้ก็ต่อเมื่อโปรแกรมไม่ได้อยู่ใน session ใด

ฟังก์ชันทำได้สำเร็จจะได้ค่า DP_OK

หมายเหตุ

ฟังก์ชันนี้ไม่สามารถเรียกใน IDirectPlay::EnumSessions ได้ ซึ่งจะต้องเรียกภายหลังเมื่อหา session ทั้งหมดได้แล้วหรือไม่มีการค้นหาต่อ นอกจากนั้นฟังก์ชันนี้จะต้องเรียกก่อนฟังก์ชัน IDirectPlay::Close

IDirectPlay::CreatePlayer

สร้างผู้เล่นภายใน session ที่อยู่

```
HRESULT CreatePlayer(
        LPDPID lpidPlayer,
        LPSTR lpFriendlyName,
        LPSTR lpFormalName,
        IUnknown FAR * pUnkOuter
);
```

lpidPlayer ค่า ID ของผู้เล่นที่ต้องการสร้าง

lpFirendlyName พอยต์เตอร์ของตัวอักษรที่เก็บชื่อไม่เป็นทางการ

lpFormalName พอยต์เตอร์ของตัวอักษรที่เก็บชื่อเป็นทางการ

pUnkOuter ยังไม่มีการใช้ จะใช้ค่า NULL

ฟังก์ชันทำได้สำเร็จจะได้ค่า DP_OK

IDirectPlay::DestroyPlayer

เมื่อต้องการลบผู้เล่นนั้นออกจาก session รวมทั้งทำให้ข้อมูลที่ส่งให้ผู้เล่นนี้ถูกลบออกจากคิวด้วย

HRESULT DestroyPlayer(DPID idPlayer);

idPlayer ตัวบอกว่าเป็นผู้เล่นไหนที่ต้องการลบออกจาก session

ฟังก์ชันทำได้สำเร็จจะได้ค่า DP_OK

หมายเหตุ

เมื่อเรียกฟังก์ชันนี้จะมีข้อความ DPMSG_DELETEPLAYERFROMGROUP ส่งไปยังแต่ละกลุ่ม หลังจากนั้นจะส่งข้อความ DPMSG_DESTROYPLAYERORGROUP การลบผู้เล่นจะทำได้เฉพาะโปรแกรมที่สร้างเท่านั้น

IDirectPlay::Send

เมื่อต้องการส่งข้อมูลไปยังผู้เล่นอื่น เป็นกลุ่ม หรือผู้เล่นทุกคนใน session

HRESULT Send(

DPID idFrom,

DPID idTo,

DWORD dwFlags,

LPVOID lpData,

DWORD dwDataSize

);

idFrom ตัวบอกว่ามาจากผู้เล่นใด

idTo ตัวบอกปลายทาง ถ้าต้องการส่งไปยังผู้เล่นก็ต้องใช้ ID ของผู้เล่น ถ้าต้องการส่งเป็นกลุ่มก็ต้องใช้ ID ของกลุ่ม ถ้าต้องการส่งทุกคนใน session จะกำหนดเป็น DPID_ALLPLAYERS ผู้เล่นไม่สามารถส่งข้อมูลให้กับตัวเองได้

dwFlags กำหนดวิธีการส่งข้อมูล ถ้าไม่กำหนดจะถูกใช้เป็นแบบไม่รับประกัน

DPSEND_ENCRYPTED ต้องการส่งข้อมูลที่เข้ารหัส สามารถทำได้เฉพาะ session ที่มีความปลอดภัย

DPSEND_GUARANTEED เมื่อต้องการแน่ใจว่าข้อมูลส่งถึงปลายทางอย่างแน่นอน

DPSEND_SIGNED ส่งข้อมูลโดยมีลายเซ็นดิจิตอล ใช้ได้เฉพาะ session ที่มีความปลอดภัย

lpData พอยต์เตอร์ของข้อมูลที่ต้องการส่ง

dwDataSize จำนวนข้อมูลที่ต้องการส่ง

ฟังก์ชันทำได้สำเร็จจะได้ค่า DP_OK

IDirectPlay::Receive

รับข้อมูลจากคิวข้อมูล

HRESULT Receive(

LPDPID lpidFrom,

LPDPID lpidTo,

DWORD dwFlags,

LPVOID lpData,

LPDWORD lpdwDataSize

);

lpidFrom ค่าจะถูกบอกว่าข้อมูลถูกส่งมาจากผู้เล่นใด หรือจะเจาะจงผู้ส่งโดยใช้ร่วมกับ

DPRECEIVE_FROMPLAYER

lpidTo ค่าจะถูกบอกว่าผู้เล่นปลายทางคือใคร เมื่อเจาะจงผู้เล่นปลายทางจะต้องใช้ร่วมกับ

DPRECEIVE_TOPLAYER

dwFlags บอกวิธีการรับข้อมูล ถ้าไม่กำหนดจะใช้ค่าเป็น 0 หมายถึงข้อมูลแรกที่มีอยู่ให้

DPRECEIVE_ALL ข้อมูลแรกที่มีอยู่

DPRECEIVE_PEEK ดูข้อมูลตามที่ระบุร่วมกับค่าอื่น จะไม่มีการนำข้อมูลออกจากคิว ถ้า lpData เป็น NULL จะต้องกำหนดค่านี้

DPRECEIVE_TOPLAYER จะได้ข้อมูลที่ถูกส่งมาจาก lpidTo.

DPRECEIVE_FROMPLAYER จะได้ข้อมูลที่ปลายทางเป็น lpidFrom.

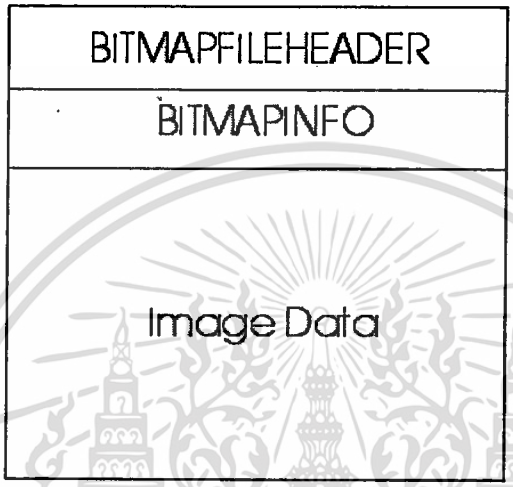
lpData พอยต์เตอร์ของบัฟเฟอร์สำหรับเก็บข้อมูลที่รับเข้ามา

lpdwDataSize พอยต์เตอร์ของค่าสำหรับเก็บความยาวของข้อมูล ต้องกำหนดบอกขนาดของบัฟเฟอร์ก่อนที่จะเรียกใช้ฟังก์ชันนี้ หลังจากเรียกแล้วค่านี้จะบอกขนาดข้อมูลที่รับมา

ฟังก์ชันทำได้สำเร็จจะได้ค่า DP_OK

ภาคผนวก ช
กราฟฟิกโดยทั่วไป

Bitmap เป็นข้อมูลที่อธิบายถึงการแสดงพิกเซลของภาพที่จะปรากกฏบนหน้าจอโดยโครงสร้างของไฟล์จะเป็นดังนี้




```
typedef  struct  tagBITMAPFILEHEADER  {
        WORD        bfType;             //สำหรับบิตแมปต้องเป็น 0x4d42 เสมอ
        DWORD       bfSize;             //ขนาดของไฟล์ทั้งหมด
        WORD        bfReserved1;        //เป็น 0 เสมอ
        WORD        bfReserved2;        //เป็น 0 เสมอ
        DWORD       bfOffBits;          //จำนวนไบต์นับตั้งแต่ BITMAPFILEHEADER
}
typedef struct tagBITMAPINFO {
        BITMAPINFOHEADER        bmiHeader;      //ดู struct tagBITMAPINFOHEADER
        RGBQUAD                 bmicolors[1];   //ตารางสี(เป็นarray ของสีRGB)
}BITMAPINFO;
typedef struct tagBITMAPINFOHEADER {
        DWORD       biSize;             //ขนาดของโครงสร้างในหน่วยของไบต์
        LONG        biWidth;            //ความกว้างของภาพเป็นพิกเซล
        LONG        biHight;            //ความสูงของภาพเป็นพิกเซล
        WORD        biPlanes;           //ไม่ได้ใช้เป็น 0 เสมอ
        WORD        biBitCount;         //จำนวนบิตต่อพิกเซล
```

```
    DWORD       biCompression;      //ชนิดของการบีบอัดภาพ
    DWORD       biSizeImage;        //ขนาดของ image data ถ้าภาพมีการบีบจะเป็น0
    LONG        biXPelsPerMeter;    //ความละเอียดในแนวนอนในหน่วยพิกเซลต่อเมตร
    LONG        biYPelsPerMeter;    //ความละเอียดในแนวตั้งในหน่วยพิกเซลต่อเมตร
    DWORD       biClrUsed;          //จำนวนสีที่ใช้ในตารางสี
    DWORD       biClrImportant;     //จำนวนของสีในตารางสีที่ขึ้นกับการแสดงผล
} BITMAPINFOHEADER;
```

Palettes รูปแบบการเก็บข้อมูลของภาพที่ใช้กันอยู่ ก็จะมีแบบ true color ซึ่งต้องเก็บข้อมูลของทุกพิกเซลของภาพในรูปแบบของ RGB ซึ่งจะต้องใช้ถึง 24 บิตในการเก็บ และอีกแบบก็คือ index color ซึ่งข้อมูลของแต่ละพิกเซลจะเก็บเป็นอินเด็กซ์ของสีซึ่งจะต้องนำไปเทียบกับตารางของสำที่เก็บค่าของสี RGB เอาไว้ ซึ่งตารางนั้นจะเรียกว่า palette ซึ่งขนาดจะมีได้ 2 , 4 ,16 และ 256 ช่องโดย 1 ช่องแทนด้วยสีขนาด 24 บิต การเก็บเฉพาะอินเด็กซ์จะมีข้อดีคือ ไฟล์จะมีขนาดเล็กเพราะว่าภาพบางภาพไม่ได้ใช้สีทั้งหมดเท่าที่RGB ทำได้ (24 บิต หรือ 16 ล้านสี) เก็บเป็นอินเด็กซ์แล้วนำไปเทียบกับ palette จะใช้เนื้อที่น้อยกว่า

นอกจากนี้ภาพที่ใช้ palette ยังสามารถเปลี่ยนสีที่ต้องการได้โดยเปลี่ยนสีใน palette ได้โดยตรง วิธีการนี้จะนำไปใช้ในเทคนิก palette animation ที่ใช้ทำเอฟเฟกของเกมส์ เช่น ลูกไฟกำลังลุก, หน้าจอสามารถเปลียนสีเช่นจากสว่างไปมืด(fade) หรือค่อยๆเปลี่ยนไปอีกสีหนึ่ง

Video memory เป็นหน่วยความจำที่มีอยู่ในการ์ดแสดงผล ซึ่งจะเป็นส่วนที่จะเก็บภาพที่จะแสดงผลออกทางหน้าจอโดยตรง จำนวนหน่วยความจำในการ์ดแสดงผลมีผลต่อโหมดการแสดงผลที่จะแสดงออกทางหน้าจอ ยิ่งมีมากยิ่งสามารถแสดงผลได้ความละเอียดสูง และแสดงสีได้มาก video memory สำหรับการแสดงผลในปัจจุบันควรไม่ต่ำกว่า 2 MB

กราฟฟิกโหมด 13

เป็นโหมดของหน้าจอโหมดหนึ่งที่เรียกตามเลขโหมดของ BIOS ซึ่งจะมีขนาด 320x200 8 บิตสี ไม่สามารถสลับเพจ(flip)การแสดงผลได้

กราฟฟิกโหมด X

เป็นโหมดการแสดงผลแบบผสมระหว่าง VGA และ โหมด 13 โหมดนี้สามารถใช้หน่วยความจำได้ถึง 256 KB (โหมด 13 ใช้ได้เพียง 64KB) โดยใช้การแสดงผลของ VGA โดยใช้ระบบ multiple vidoo ของ EGA ซึ่งทำให้มันมีหน้าจอสำหรับแสดงผลถึงสี่หน้าจอ มันสามารถแสดงผลได้ทั้ง 320x200 และ320x240

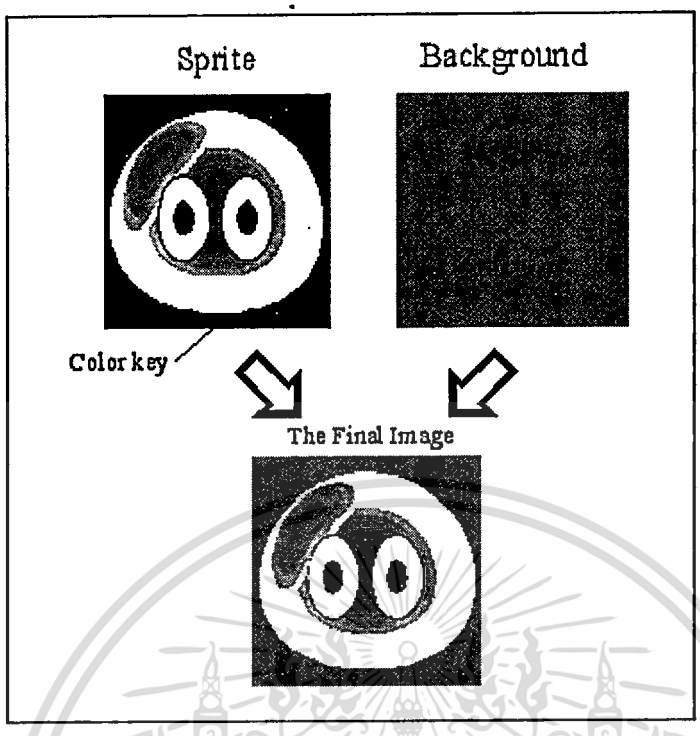
Double Buffer

เป็นเทคนิคอย่างหนึ่งในการแสดงภาพต่อเนื่องโดยใช้สองหน้าจอเพื่อสลับกันแสดงผล ถ้าดูลักษณะทางกายภาพของการแสดงผลของจอมอนิเตอร์ จอภาพจะใช้ปืนอีเล็กตรอนเพื่อทำการวาดจากซ้ายไปขวาและจากบนลงล่าง เมื่อวาดจนครบหนึ่งหน้าจอแล้ว จะมีช่วงหนึ่งที่ปืนอีเล็กตรอนไม่ทำการวาด และจะเคลื่อนที่กลับมามุมบนซ้ายเพื่อทำการวาดในครั้งต่อไปซึ่งเรียกว่า vertical bland หรือบางทีเรียก vertical retrace

ถ้าทำการวาดภาพลงไปบนหน่วยความจำสำหรับแสดงผลโดยตรงในขณะที่ปืนอีเล็กตรอนวาดไปได้ครึ่งหน้าจอ เราจะเห็นว่ามีทั้งภาพของทั้งเฟรมเก่าและเฟรมใหม่ปนกันในเวลาเดียวกัน ผู้มองภาพจากหน้าจอนั้นจะเห็นภาพฉีกขาดออกจากกัน ดังนั้นจะใช้เทคนิค Double Buffer แก้ได้โดย ให้วาดหน้าจอนั้นให้เสร็จเสียก่อนในหน่วยความจำที่ไม่ได้แสดงออกทางหน้าจอ เมื่อวาดเสร็จถึงทำการทำสำเนาส่วนนั้นออกทางส่วนแสดงผล และต้องทำในช่วงเวลาที่ปืนอิเล็กตรอนเคลื่อนที่กลับไปตำแหน่งเริ่มต้นเท่านั้น เพราะณ เวลานั้นจะไม่มีการวาดหน้าจอ

Sprite

sprite เป็นตัวแทนใช้แสดงถึงตัวละครในเกมส์ ซึ่งสามารถเคลื่อนไปบนหน้าจอได้ มีลักษณะเป็นวัตถุซึ่งจะมีลักษณะเฉพาะตัวของตัวเอง สามารถเคลื่อนไหวหรือไม่ก็ได้ ถ้าเป็น sprite ที่เคลื่อนไหวได้จะประกอบไปด้วยภาพในแต่ละเฟรมของการเคลื่อนไหวเรียงต่อกัน และจะทำการสลับภาพออกมาแสดงผลอย่างต่อเนื่องเพื่อให้เกิดภาพเคลื่อนไหว การสร้าง sprite นี้จะขึ้นกับนักพัฒนากำหนดคุณสมบัติให้กับมัน เนื่องจาก sprite เป็นตัวละครซึ่งจะมีความเป็นวัตถุมาก มีรูปร่างไม่แน่นอน ซึ่งปกติจะเก็บภาพในลักษณะที่เป็นสี่เหลี่ยมเท่านั้น ดังนั้นการที่ sprite เป็นรูปร่างที่ได้ จากมีส่วนที่ไม่ได้เป็นส่วนประกอบของวัตถุนั้นได้กำหนดให้เป็นส่วนที่โปร่งใส ซึ่งจะทำการกำหนดให้สีหนึ่งของภาพเป็นสีที่โปรงใส เรียกสีนั้นว่า color key



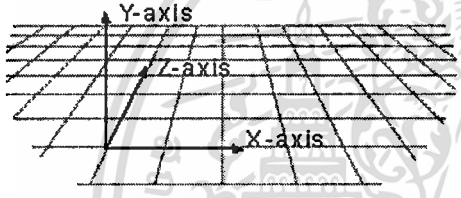
ภาพแสดง color key

ภาคผนวก ซ
กราฟฟิกสามมิติทั่วไป

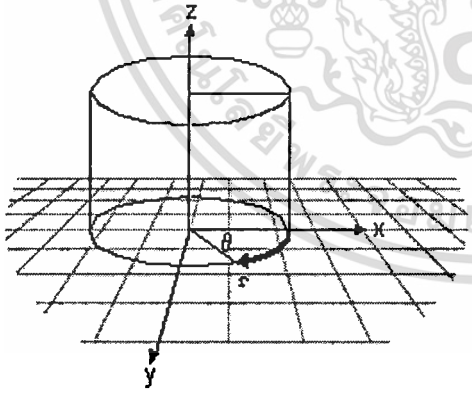
Coordinate System

การอ้างตำแหน่งในทางสามมิติทั่วๆไปจะใช้พิกัดฉากในการอ้าง โดยมีแกนสามแกน x, y และ z ในการอ้างตำแหน่ง ทุกแกนจะตั้งฉากกัน นอกจากนี้ยังมีพิกัดอื่นอีกเช่นพิกัดวงกลมที่จะใช้ค่าสามค่าเช่นกันคือ รัศมี r, มุมของวงกลม θ (Theta) และแกน Z พิกัดทรงกลม จะใช้สามค่าเช่นกัน มุมในระนาบ y x Theta (θ), มุมในระนาบที่เกิดจากแกน z กับ แกนที่อยู่บนระนาบ y x ทำมุม Theata ซึ่งเรียก Phi (ϕ), และรัศมีของทรงกลม Roe (ρ)

พิกัดฉาก

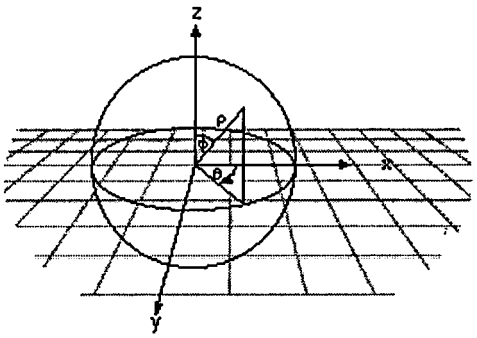


พิกัดทรงกระบอก



$x = r(\cos\theta)$ $y = r(\sin\theta)$ $z = z$

พิกกัดทรงกลม



$x = \rho(\sin\phi)(\cos\theta)$ $y = \rho(\sin\phi)(\sin\theta)$ $z = \rho(\cos\theta)$

และยังมีโคออดิเนตที่มีแค่สองตัวแปรเหมือนกับแบบสองมิติแต่เรานำมันมาใช้ในการอ้าง texture เพื่อ mapping เข้ากับวัตถุสามมิติซึ่งเรียกว่า U และ V Coordinates เวกเตอร์ V จะใช้อธิบายทิศทางและการวางตัวของ texture และจะวางตัวตามแนวแกน z ส่วนเวกเตอร์ U หรือเรียกอีกอย่างว่า up vector จะวางตัวตามแนวแกน y โดยจุดกำเนิดอยู่ที่ [0,0,0] ซึ่งการmapเข้ากับวัตถุเรียกอีกอย่างว่า wrap ซึ่งมีเทคนิคทั่วไป 4 แบบ คือ

Flat จะใช้การคำนวนจากระนาบแบนเข้าสู่วัตถุโดยพยายามแผ่ให้คลุมทั้งวัตถุ

Cylindrical จะใช้พิกัดทรงกระบอกในการคำนวน โดยทำเหมือนกับการม้วนระนาบแบนให้รอบวัตถุขณะทำการ mapping

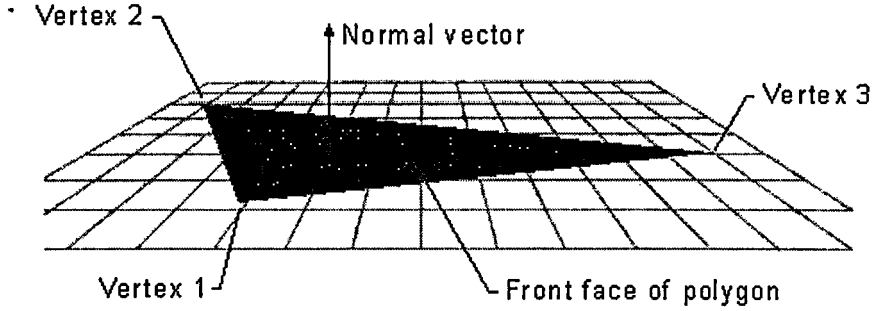
Spherical จะใช้พิกัดทรงกลมในการคำนวน โดยทำเหมือนการห่อจากระนาบแบนให้เป็นทรงกลม ซึ่งจะเกิดการสูญเสียข้อมูลของ texture บ้างเนื่องจากการห่อในแนวแกน z

Chrome จะเป็นโคออดิเนตที่คำนวนจากการสะท้อนที่ปรากฏบนวัตถุ

Unit Vector หรือเวกเตอร์หนึ่งหน่วย เป็นเวกเตอร์ที่มีขนาดหนึ่งหน่วย ส่วนใหญ่ใช้ในการแทนทิศทาง มีใช้ในทั้งพิกัดแบบสองและสามมิติ

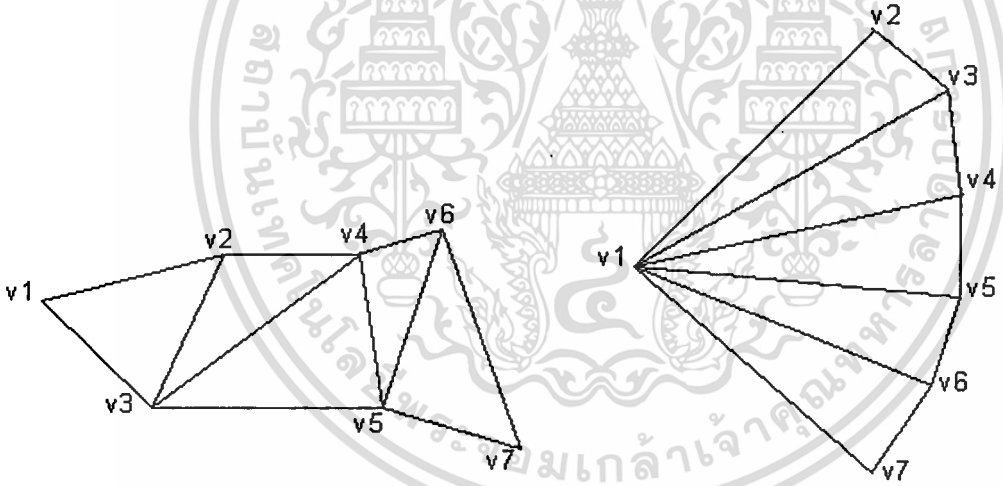
Normal Vector การที่จะแสดงระนาบใดระนาบหนึ่งในระบบสามมิติสามารใช้เวกเตอร์ในการแทนซึ่งจะเป็นเวอเตอร์หนึ่งหน่วยตั้งฉากกับพื้นผิวของระนาบนั้นๆ

Face(พื้นผิว) พื้นผิวแบบสามมิติเกิดจากเซตของจุด(Vertex)และทิศทางของพื้นผิวซึ่งทิศทางแทนด้วย normal vector เราสามารถมองเห็นพื้นผิวได้จากด้านที่ normal vector นั้นพุ่งออกมา



Mesh เป็นสิ่งที่ใช้แทนวัตถุสามมิติ ซึ่งเกิดจากเซ็ตของพื้นผิวหลายพื้นผิวประกอบกับ บางที่เราจะเรียกพื้นผิวนั้นว่า "Polygon" โดยพื้นฐานแล้วจะใช้จะใช้โพลิกอนรูปสามเหลี่ยมเป็นส่วนใหญ่ เหตุผลที่ใช้รูปสามเหลี่ยมเพราะเป็น convex เสมอ และเป็นระนาบแบนเหมาะแก่การเรนเดอร์

โพลิกอนจะเกิดจาการต่อกับของfaceรูปสามเหลี่ยมหลายรูปดังภาพ

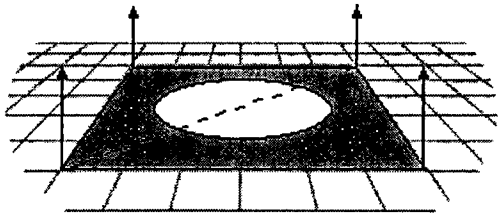


Render เนื่องจากภายใต้สภาพแวดล้อมแบบสามมิติไม่มีอุปกรณ์แสดงผลที่แสดงได้อย่างเช่น จอมอนิเตอร์ก็แสดงได้เพียงระนาบสองมิติแบน ดังนั้นเราต้องมีการสร้างภาพฉายสองมิติจากภาพสามมิติ ซึ่งเราเรียกว่า เรนเดอร์ การเรนเดอร์นั้นสามารถทำได้หลายแบบ

Flat Shade Mode เป็นโหมดที่มีทุกจุดของสีบนพื้นผิว(face) เป็นสีเดียวกันหมด รูปที่ได้จะเป็นโพลิกอนที่มองเห็นเป็นเหลี่ยมมุมมาก ตามพื้นผิวที่แท้จริง ความแตกต่างจะเห็นได้ชัดที่ขอบ(edge)

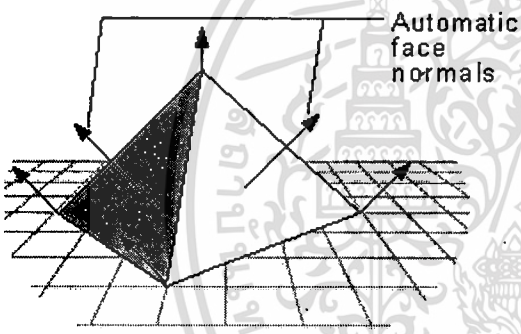
Ground Shade Mode จะต่างกับ Flat ตรงที่ ขอบของพื้นผิวที่ใช้ขอบร่วมกัน ค่าความส่องสว่างและสีจะมีการ interpolate (แทรกสอดหรือผสมแบบเฉลี่ย) ระหว่างผิวที่ใช้ขอบร่วมกัน ทำให้เห็นว่าตรงขอบมีความมนยิ่งขึ้นโดยจะเป็นแบบ linear มี

Pong Shade Mode มีการ interpolate เช่นกัน แต่จะมีการคำนวนทุกพิกเซลบนพื้นผิวด้วย ซึ่งมีผลดีก็คือจะไม่สูญเสียข้อมูลบางอย่างบนพื้นผิวไปสังเกตภาพตัวอย่าง

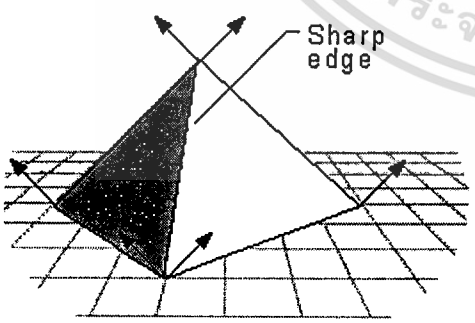


จะเห็นว่าพื้นผิวสามเหลี่ยมสองอันประกบกันโดยใช้ขอบ(ที่เป็นเส้นประ)ร่วมกัน และมีแสงลงเป็นดวงกลมระหว่างผิวทั้งสอง ถ้าเป็นแบบ Ground ผลของการเรนเดอร์จะไม่ปรากฏว่ามีแสงส่องที่บริเวณนั้นเลย แต่ถ้าเป็นแบบ Pong ถึงจะสามารถแสดงได้

เนื่องจากการ interpolate ทำให้ขอบเหลี่ยมกลายเป็นขอบมน ดังนั้นถ้าเราต้องการขอบที่เป็นเหลี่ยมในขณะที่ใช้การเรนเดอร์ในโหมด Ground หรือ Pong ต้องเพิ่ม vertex normal เข้าไปอีกตรงบริเวณที่พื้นผิวตัดกันและเราต้องการให้เป็นเหลี่ยม สังเกตภาพถัดไปเป็นภาพโพลิกอนปรกติ



หลังจากเพิ่ม vertex normal จะเป็นดังภาพถัดไป



หมายเหตุ ใน Direct3D เวอร์ชันปัจจุบันยังไม่สนับสนุนโหมดของ Pong

Raytrace เป็นโหมดการเรนเดอร์ที่กินทรัพยากรของเครื่องคอมพิวเตอร์มากที่สุด เพราะคำนวนเหมือนการตกกระทบและสะท้อนแสงแบบที่เกิดจริงในธรรมชาติ เพียงแต่เป็นการคำนวนย้อนรอยจากหน้า

จอไปกระทบวัตถุแล้วสะท้อนเข้าหาแหล่งกำเนิดแสง เพื่อจะได้ไม่ต้องคำนานส่วนที่ไม่ปรากฏบนหน้าจอ ในงานสามมิติแบบ realtime ไม่สามารถทำได้เพราะใช้ทรัพยากรมากเกินไป ไม่สามารถประมวลผลได้ทัน

3D Transformations ในสภาพแวดล้อมแบบสามมิติจะมีการ transform เพื่อการ

- เคลื่อนย้ายตำแหน่งโดยอ้างอิงกับวัตถุอื่น
- หมุน, บิด และ ย่อขยายขนาด
- เปลี่ยนตำแหน่ง ทิศทาง และทำperspective ให้กับช่องสำหรับมองภาพ

การ transform จะคำนวนจาก metrix 4x4 จากภาพต่อไปจะเป็นการ transform จากจุด (x, y, z)ไปเป็น (x', y', z')

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1]\begin{bmatrix} M_{11} & M_{12} & M_{13} & M_{14} \\ M_{21} & M_{22} & M_{23} & M_{24} \\ M_{31} & M_{32} & M_{33} & M_{34} \\ M_{41} & M_{42} & M_{43} & M_{44} \end{bmatrix}$$

ผลที่ได้จะเป็นดังนี้

$$x' = (M_{11} \times x) + (M_{21} \times y) + (M_{31} \times z) + (M_{41} \times 1)$$
$$y' = (M_{12} \times x) + (M_{22} \times y) + (M_{32} \times z) + (M_{42} \times 1)$$
$$z' = (M_{13} \times x) + (M_{23} \times y) + (M_{33} \times z) + (M_{43} \times 1)$$

จะเห็นได้ว่า metrix ที่ใช้เป็น metirx 4*4 เพราะว่าเราใช้ homogeneous coordinate system ซึ่งจะมีลักษณะสมกรดังนี้ (X/r , Y/r , Z/r , r) ซึ่งตอนนี้ใช้ r = 1 การ transform สามารถทำได้โดยใช้สมการดังนี้

Translation(เคลื่อนย้าย)

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1]\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ T_x & T_y & T_z & 1 \end{bmatrix}$$

Rotation(การหมุน)

สมการการหมุนต่อไปนี้จะยึดกฎมือซ้ายในการพิจารณา

สมการแรกจะเป็นการหมุนจุด(x y z)รอบแกน x ผลของการหมุนจะเป็น (x', y', z')

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

สมการที่สองจะเป็นการหมุนจุด(x y z)รอบแกน y ผลของการหมุนจะเป็น (x', y', z')

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

สมการสุดท้ายจะเป็นการหมุนจุด(x y z)รอบแกน z ผลของการหมุนจะเป็น (x', y', z')

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

Scaling(ย่อขยายขนาด)

จะเป็นการย่อขยาย ซึ่งทำให้จุด (x y z) มีการย้ายตำแหน่งตามทิศของ x ,y และ z ทำให้จุดอยู่ที่ตำแหน่งใหม่คือ (x', y', z')

$$[x'\ y'\ z'\ 1] = [x\ y\ z\ 1]\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

ภาคผนวก ฌ
การสื่อสารบน Network โดยทั่วไป

session เป็นช่องสัญญาณที่กำหนด เพื่อสามารถส่งข้อมูลกัน เฉพาะเครื่องที่อยู่ในช่องสัญญาณเดียวกัน

multiplayer การเล่นเกมส์ที่มีคนอย่างน้อย 2 คนเล่นด้วยกัน

message ข้อมูลที่ส่งระหว่างกันในแต่ละครั้ง ภายในระบบเน็ตเวิร์ก

peer-to-peer การติดต่อสื่อสารระหว่างเครื่องโดยตรง ไม่จำเป็นต้องผ่านเครื่องตัวกลาง

client/server การติดต่อสื่อสารระหว่างเครื่องที่ต้องส่งข้อมูลไปยังเครื่องที่ให้บริการ (server) จากนั้นเครื่องที่ให้บริการจะส่งข้อมูลต่อไปยังเครื่องปลายทาง

โปรโตคอล มาตรฐานในการส่งข้อมูลบนเน็ตเวิร์ก มีการกำหนดรูปแบบข้อมูล วิธีการส่ง, การตอบรับ, การเข้ารหัสความปลอดภัย และการตรวจสอบความถูกต้องของข้อมูล

TCP/IP เป็นโปรโตคอลที่ใช้ติดต่อกันบนอินเตอร์เนต

IPX โปรโตคอลที่ใช้บน Local Area Network (LAN)

TAPI โปรโตคอลที่ใช้กับการติดต่อสื่อสารผ่านโมเด็ม

ภาคผนวก ญ
พื้นฐานเกี่ยวกับเสียงบนคอมพิวเตอร์โดยทั่วไป

sound card (อุปกรณ์เสียง) เป็นอุปกรณ์ ที่ใช้ในการเปลี่ยนสัญญาณเสียงจากดิจิตอลเป็นสัญญาณอนาล็อก สัญญาณอนาล็อกนี้สามารถนำไปออกลำโพงได้เลย

channel (ช่องสัญญาณเสียง) แต่ละช่องสัญญาณเสียงจะเก็บข้อมูลเสียง เพื่อที่สามารถเล่นเสียง ต่างช่องสัญญาณก็สามารถเล่นเสียงได้โดยไม่ขึ้นแก่กัน คือไม่จำเป็นต้องเล่นพร้อมกัน หลังจากเล่นเสียงจะถูกผสมก่อนที่จะออกสู่ลำโพง

latency playback ช่วงเวลาที่ใช้ในตั้งแต่ส่งคำสั่งเล่นเสียงจนผู้ฟังได้ยินเสียง

buffer หน่วยความจำที่เก็บข้อมูลเสียง จะเป็นหน่วยความจำบนอุปกรณ์เสียง หรือบนหน่วยความจำหลักก็ได้

sound buffer ดู **buffer**

Musical Instrument Digital Interface (MIDI) ชนิดของข้อมูลเสียงชนิดหนึ่ง ที่อธิบายถึงตัวโน้ต และอุปกรณ์ที่เล่น การเก็บข้อมูลแบบนี้จะใช้เนื้อที่ไม่มาก จึงสามารถใช้ในการเก็บเพลงได้

WAVE เป็นรูปแบบการเก็บข้อมูลของเสียง ซึ่งเป็นมาตรฐานของไมโครซอฟท์ ใช้การเก็บข้อมูลเสียงโดยตรงซึ่งไฟล์จะมีขนาดใหญ่

secondary sound buffer ดู **buffer**

static buffer หน่วยความจำส่วนที่เก็บข้อมูลทั้งหมดของเสียง จึงเขียนข้อมูลลงหน่วยความจำเพียงครั้งเดียว

streaming buffer หน่วยความจำที่ถูกจองไว้เพื่อเก็บข้อมูลเสียงเพียงบางส่วน เนื่องจากข้อมูลเสียงมีขนาดใหญ่ จึงเป็นการเปลืองที่จะต้องจองหน่วยความจำจำนวนมากเพื่อเก็บข้อมูลเสียงทั้งหมด ดั้งนั้นจึงต้องมีการเขียนข้อมูลเสียงทับส่วนที่เล่นไปแล้ว

ภาคผนวก ฎ

ตัวอย่างโปรแกรมอย่างง่าย

โปรแกรมจะแบ่งออกเป็น 5 ส่วนหลักๆก็คือ

1.ส่วนที่ประกาศ include file และ ตัวแปรแบบ global

2.ฟังก์ชันการทำงานทั่วไป

- restoreAll(void) เนื่องจากการทำงานร่วมกับโปรแกรมอื่นบางที โปรแกรมนั้นอาจทำให้ส่วนของหน่วยความจำที่ใช้สำหรับเก็บภาพ(Surface) นำข้อมูลไปทับส่วนนั้นได้ เพราะว่าโปรแกรมอื่นจะไม่รู้จัก Surface ของ DirectDraw ดังนั้นจะต้องมีการนำข้อมูลที่หายไปกลับมา

- updateFrames(void) ใช้วาดและแสดงเฟรมถัดไป โดยจะวาดใน Back Buffer

- flipFrames(void) สลับภาพใน BackBuffer ออกมาแสดงผลที่ Primary Surface

- increasex(int x,int limit) จะส่งค่า x ที่เพิ่มออกมาโดยจะไม่เกินค่าของ limit

- int decreasex(int x,int limit) จะส่งค่า x ที่ลดออกมาโดยจะไม่เกินค่าของ limit

3. ส่วนจักการ message ของวินโดว์

- WindowProc()จะใช้สำหรับใส่ฟังก์ชันที่จะมีการตอบสนอง message ที่วินโดว์ส่งออกมา เช่น message ของคีย์บอร์ด ,การทำลายวินโดว์

4. ฟังก์ชันสำหรับใช้เริ่มทำงานโปรแกรม

- doInit(HINSTANCE hInstance, int nCmdShow) จะมีการประกาศและสร้างวินโดว์ และการสร้างออปเจ็กของ DirectDraw

5.ฟังก์ชันหลักของโปรแกรม

-WinMain () เป็นฟังก์ชันหลักสำหรับเริ่มโปรแกรม(ในดอสจะใช้ main()) โดยจะเป็น loop การทำงานไปเรื่อยๆ จะจบการทำงานเมื่อมีการจบโปรแกรม

```
/*************************** ส่วนที่1 ******************************/
#include <windows.h>
#include <windowsx.h>
#include <ddraw.h>
#include "ddutil.h"              //utility ที่ช่วยสำหรับ DirectDraw และการโหลดภาพ
LPDIRECTDRAW             lpDD;           // สร้าง object DirectDraw
LPDIRECTDRAWSURFACE      lpDDSPrimary;   //Primary surface
LPDIRECTDRAWSURFACE      lpDDSBack;      // Back Buffer surface
LPDIRECTDRAWSURFACE      lpDDSOne;       //ใช้เก็บภาพจาก ALL.BMP
LPDIRECTDRAWPALETTE      lpDDPal;        // DirectDraw palette
BOOL             bActive;  //ใช้ในการตรวจสอบว่า application active?
HINSTANCE        g_hinst;
HWND             g_hwnd    = NULL; //handle ของวินโดว์ของโปรแกรม
char szBitmap[] = "all.bmp";             //ชื่อภาพที่จะโหลด
static int    xpos = 300;                //ตำแหน่งของ donut เริ่มต้นบนหน้าจอ
static int , ypos = 200;

/*************************** ส่วนที่2 ******************************/
/*
 * restoreAll()   : restore all lost objects
 */ // ฟังก์ชันในการโหลดข้อมูลของ Surface ในกรณีที่ข้อมูลสูญหาย
HRESULT restoreAll( void )
{
    HRESULT         ddrval;
    ddrval = lpDDSPrimary->Restore();
    if( ddrval == DD_OK )
    {
        ddrval = lpDDSOne->Restore();
        if( ddrval == DD_OK )
        {
            DDReLoadBitmap(lpDDSOne,szBitmap);//ฟังก์ชันใน ddutil.cpp ช่วยในการ
โหลดภาพซ้ำ
        }
    }
    return ddrval;
} /* restoreAll */

/*
* finiObjects :release all object
*/ //ฟังก์ชันสำหรับคืนหน่วยความจำของ object ที่จองเอาไว้
static void finiObjects( void )
{
    if( lpDD != NULL )
    {
        if( lpDDSPrimary != NULL )
        {
            lpDDSPrimary->Release();
            lpDDSPrimary = NULL;
```

```
        }
        if( lpDDSOne != NULL )
        {
                lpDDSOne->Release();
                lpDDSOne = NULL;
        }
        if( lpDDPal != NULL )
        {
                lpDDPal->Release();
                lpDDPal = NULL;
        }
        lpDD->Release();
        lpDD = NULL;
    }
}
/*
 * updateFrames(void)
 */  //ฟังก์ชันในการ update สถานะของเฟรมในการแสดงผล
void updateFrames(void)
{
    RECT rcRect;
    static int currentframe = 0;
    HRESULT ddrval;
        if (currentframe<59) //ภาพ donut มีทั้งหมด 0-59 เฟรม
                currentframe++; //ทำการชี้ที่เฟรมถัดไป
                else currentframe=0;
                //bilt background
        rcRect.left = 0;
        rcRect.top = 0;
        rcRect.right = 640;
        rcRect.bottom = 480;
        while( 1 )
        {
          ddrval = lpDDSBack->BltFast( 0, 0, lpDDSOne,
            &rcRect, DDBLTFAST_NOCOLORKEY );

                if( ddrval == DD_OK )
                        break;
                if( ddrval == DDERR_SURFACELOST )
                {
                        ddrval = restoreAll();
                        if( ddrval != DD_OK )
                                return;
                }
                if( ddrval != DDERR_WASSTILLDRAWING )
                        return;

        }
        //bilt charactor
        rcRect.left   = currentframe%10*64;
```

```
    rcRect.top    = currentframe/10*64 + 480;
    rcRect.right  = currentframe%10*64 + 60;
    rcRect.bottom = currentframe/10*64 + 50 + 480;//สูตรในการคำนวนเฟรมของdonutที่จะวาด
            while( 1 )
            {
              ddrval=lpDDSBack->BltFast( xpos, ypos, lpDDSOne,
          &rcRect, DDBLTFAST_SRCCOLORKEY );  //วาดภาพ
              if( ddrval == DD_OK )
                      break;
              if( ddrval == DDERR_SURFACELOST )
              {
                      ddrval = restoreAll(); //โหลด Surface ใหม่
                      if( ddrval != DD_OK )
                              return;
              }
              if( ddrval != DDERR_WASSTILLDRAWING )//กำลังวาดอยู่
                      return;
            }
}

/*
* flipFrames() ;exchange between back and primary
*/  //สลับภาพระหว่าง BackBuffer กับ Primary เพื่อแสดงผล
void flipFrames(void)
{
    HRESULT ddrval;
    while( 1 )
    {
        ddrval = lpDDSPrimary->Flip( NULL, 0 );
        if( ddrval == DD_OK )
                break;
        if( ddrval == DDERR_SURFACELOST )
        {
                ddrval = restoreAll();
                if( ddrval != DD_OK )
                        break;
        }
        if( ddrval != DDERR_WASSTILLDRAWING )
                break;
    }
}
/*increasex
*       increase value but not more than limit
*/  //ฟังก์ชันสำหรับเพิ่มค่าไม่เกินค่าที่กำหนด(limit)
int increasex(int x,int limit)
{
    x=x+3;
    if (x > limit)
            x= limit;
    return x;
```

```
}/*increasex*/

/*decreasex
*       decrease value but not less than limit
*/ //ฟังก์ชันในการลดค่าไม่ต่ำกว่าค่าที่กำหนด
int decreasex(int x,int limit)
{
        x=x-3;
        if (x < limit)
                x = limit;
        return x;
}/*decreasex*/
/************************** ส่วนที่3 ****************************/

/*
* WindowProc()
*/ //ฟังก์ชันในการจัดการ message ของวินโดว์
long FAR PASCAL WindowProc( HWND hWnd,
        UINT message, WPARAM wParam, LPARAM lParam )
{
  switch( message )
  {
  case WM_ACTIVATEAPP:
        bActive = wParam;    //บอกสถานะว่าโปรแกรมทำงานอยู่
        break;
case WM_SETCURSOR:
        SetCursor(NULL);
        return TRUE;
case WM_CREATE:
        break;
case WM_DESTROY: //จบการทำงานของโปรแกรมจะได้รับ message นี้
        finiObjects();
        PostQuitMessage( 0 );
        break;
case WM_KEYDOWN: //จัดการ message การกดคีย์บอร์ด
        switch(wParam)        //ในส่วนนี้จะเป็นการบังคับการเคลื่อนไหวของdonut
        {
        case VK_UP: ypos = decreasex(ypos,0);
                break;
        case VK_DOWN: ypos = increasex(ypos,410);
                break;
        case VK_LEFT: xpos = decreasex(xpos,0);
                break;
        case VK_RIGHT: xpos = increasex(xpos,570);
                break;
        case VK_ESCAPE: PostMessage(hWnd,WM_CLOSE,0,0);
                break;
        }
}
return DefWindowProc(hWnd, message, wParam, lParam);
```

```
}
/************************** ส่วนที่4 ******************************/
/*
*initFail() function for manage when initial fail
*/ //ฟังก์ชันจัดการเมื่อการ initial โปรแกรมไม่ผ่าน
BOOL initFail( HWND hwnd )
{
   finiObjects(); //ฟังก์ชันเพื่อปล่อยหน่วยความจำที่จองไว้
   MessageBox(hwnd,"DirectDraw Init FAILED","Exam1",MB_OK);
   DestroyWindow( hwnd );
   return FALSE;
}

/*
doInit ()
ฟังก์ชันในการเริ่มการทำงานของโปรกแกรม
*/
static BOOL doInit( HINSTANCE hInstance, int nCmdShow )
{
   HWND                hwnd;
   WNDCLASS            wc;
   DDSURFACEDESC       ddsd;
   DDSCAPS             ddscaps;
   HRESULT             ddrval;
   /*
   * set up and register window class
   */
   wc.style = CS_HREDRAW | CS_VREDRAW;
   wc.lpfnWndProc = WindowProc;
   wc.cbClsExtra = 0;
   wc.cbWndExtra = 0;
   wc.hInstance = hInstance;
   wc.hIcon = LoadIcon( hInstance, IDI_APPLICATION );
   wc.hCursor = LoadCursor( NULL, IDC_ARROW );
   wc.hbrBackground = GetStockBrush(BLACK_BRUSH);
   wc.lpszMenuName = "donut";
   wc.lpszClassName = "donut";
   RegisterClass( &wc );
   /*
   * create a window
   */ //เริ่มสร้างวินโดว์จากคุณสมบัติที่กำหนด
   hwnd = CreateWindowEx(
     0,
          "donut",
          "Donut Project",
          WS_POPUP,
          0,
          0,
     GetSystemMetrics(SM_CXSCREEN),
     GetSystemMetrics(SM_CYSCREEN),
```

```
        NULL,
        NULL,
        hInstance,
        NULL );
if( !hwnd )  //initial วินโดว์ไม่ผ่าน
        return FALSE;
g_hwnd = hwnd;//initial วินโดว์ได้แล้ว
ShowWindow( hwnd, nCmdShow );
UpdateWindow( hwnd );

//สร้างออปเจ็กของ DirectDraw
ddrval = DirectDrawCreate( NULL, &lpDD, NULL );
if( ddrval != DD_OK )
        return initFail(hwnd);
// Get exclusive mode ในที่นี้จะเป็นแบบเต็มหน้าจอและสามารถกำหนดโหมดตามต้องการได้
ddrval = lpDD->SetCooperativeLevel( hwnd,
        DDSCL_EXCLUSIVE | DDSCL_FULLSCREEN );
if( ddrval != DD_OK )
        return initFail(hwnd);
// Set the video mode to 640x480x8
ddrval = lpDD->SetDisplayMode( 640, 480, 8);
if( ddrval != DD_OK )
        return initFail(hwnd);
// สร้าง primary surface โดยมี 1 back buffer โดยกำหนดคุณสมบัติใน ddscaps
ddsd.dwSize = sizeof( ddsd );
ddsd.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE |
        DDSCAPS_FLIP |
        DDSCAPS_COMPLEX;
ddsd.dwBackBufferCount = 1;
ddrval = lpDD->CreateSurface( &ddsd, &lpDDSPrimary, NULL );//เริ่มสร้าง
if( ddrval != DD_OK )
        return initFail(hwnd);
ddscaps.dwCaps = DDSCAPS_BACKBUFFER;
ddrval = lpDDSPrimary->GetAttachedSurface(&ddscaps,&lpDDSBack);//จะได้ค่าpointerของ
BackBuffer
if( ddrval != DD_OK )
        return initFail(hwnd);
// สร้างและกำหนดค่าของ palette
lpDDPal = DDLoadPalette(lpDD,szBitmap);          //ฟังก์ชันของ dduitl.cppช่วยในการโหลดภาพ
if (lpDDPal)
        lpDDSPrimary->SetPalette(lpDDPal);       //กำหนดให้ Primary Surface ใช้ palette เดียว
กับภาพ
lpDDSOne = DDLoadBitmap(lpDD, szBitmap, 0, 0); // สร้าง offscreen surface จากภาพ ALL.BMP.
if( lpDDSOne == NULL ) // สร้าง Offscreen Surface ไม่สำเร็จ
        return initFail(hwnd);
// กำหนดสีที่จะเป็นค่าโปรงใส
DDSetColorKey(lpDDSOne, RGB(4,0,0));
```

```
    return TRUE;        // initial ทุกอย่างสำเร็จ
} /* doInit */

/************************** ส่วนที่5 ******************************/
/*
 * WinMain ()
 */ //โปรแกรมหลัก
int PASCAL WinMain(HINSTANCE hInstance,HINSTANCE hPrevInstance,
                                LPSTR lpCmdLine, int nCmdShow)
{
    g_hinst = hInstance;
    MSG        msg;
    lpCmdLine = lpCmdLine;
    hPrevInstance = hPrevInstance;
    if( !doInit( hInstance, nCmdShow ) )
            return FALSE;  //initial โปรแกรมไม่ผ่าน

    while (1) //loop ของโปรแกรมที่ทำงานไม่รู้จบ
    {
            if(PeekMessage(&msg,NULL,0,0,PM_NOREMOVE))  //ตรวจพบว่ามี message
            {
                    if (!GetMessage(&msg,NULL,0,0))
                            return msg.wParam;
                    TranslateMessage(&msg);
                    DispatchMessage(&msg);
            }
            else
                    if (bActive)          //ถ้าโปรแกรมทำงานอยู่(Active)จึงจะมีการอัปเดตต่างๆ
                    {
                            updateFrames();      //อัปเดตค่าของโปรแกรม
                            flipFrames();        //สลับระหว่าง Primary กับ Back Buffer
                            Sleep(1000/35);      //
                    }else
                            WaitMessage();
    }
}
```

ภาคผนวก ฏ
การพัฒนาและค้นคว้า

ขั้นตอนการพัฒนา

การเขียนโปรแกรมทำบนเครื่องพีซีสองเครื่องสำหรับสองคนในการพัฒนา ซึ่งใช้

- Pentium 75 และ 100
- RAM 48M ทั้งสองเครื่อง
- Sound Blaster 32 และ 16
- จอยสติก
- ระบบปฏิบัติการวินโดว์95
- ต่อเน็ตเวิร์กทั้งสองเครื่องเพื่อแลกเปลี่ยนข้อมูลและทรัพยากรขณะทำการพัฒนา ควรจะต่ออินเตอร์เนทด้วย เพราะการค้นหาข้อมูลส่วนใหญ่มาจากแหล่งนี้

ขั้นตอนการศึกษา

ศึกษาจาก Help และโปรแกรมตัวอย่าง ที่มากับ DirectX SDK และจากเวปที่ต่างๆเกี่ยวกับ DirectX และจากหนังสือ(มีให้ไม่มากนัก)

การโปรแกรม

ทำการโปรแกรมโดยใช้ Visual C++ บนวินโดว์ 95 ที่ไม่ใช้วินโดว์ NT เนื่องจากทำงานได้ช้าและ DirectX5 ไม่สนับสนุน การสร้างโปรแกรมจะเลือกโหมดของการแสดงผลที่ 640x480 ที่ 256 สีเป็นโหมดที่ให้การแสดงภาพที่ค่อนข้างดี และไม่กินกำลังของเครื่องมากนัก ถึงตัวอย่างที่ให้มาจากในชุด SDK ส่วนใหญ่จะใช้การเขียนในรูปแบบของ C เพื่อความเข้ากันได้ในหลายคอมไพล์เลอร์ก็ตาม ในการพัฒนาจริงควรเขียนในรูปแบบ C++ โดยโปรแกรมแบบ Object Oriented ซึ่งทำให้การพัฒนาเป็นไปได้ง่ายยิ่งขึ้น

กราฟฟิก

ในการสร้างภาพส่วนใหญ่จะใช้ภาพสองมิติเป็นส่วนใหญ่ ภาพบางภาพที่เป็นสองมิติก็มักจะมาจากภาพสามมิติที่ทำการเรนเดอร์ให้เป็นสองมิติแล้ว ภาพสามมิติจะใช้เฉพาะใน Direct3D ในการสร้างภาพโดยเฉพาะเกมส์ Bubble จะสร้างจาก CorelDraw 7 เนื่องจากเป็นโปรแกรมที่ตัวของผู้พัฒนาเองถนัด หรือถ้าใครถนัดโปรแกรมใดก็ทำได้ตามสะดวก แต่ควรเก็บภาพให้อยู่ในโหมด 24 บิตสี เพื่อให้ภาพออกมาดูดีที่สุด แต่ความละเอียดควรให้พอเหมาะกับการแสดงผล และทำให้มีการเปลี่ยนความละเอียดในภายหลังเพราะจะทำให้คุณภาพของภาพลดลง ในการแสดงผลในโหมดของสีที่น้อยกว่า 24 บิตสีควรใช้ Paint Shop Pro ในการลดสีโดยเฉพาะจาก 24 บิตสีให้เหลือ 256 สีควรใช้การลดสีแบบ X Color แล้วเลือกแบบ Nearest color ไม่ควรใช้แบบอื่นเพราะจะมีความแตกต่างจากภาพต้นฉบับมาก ส่วนภาพที่ต้องการเทคนิค

พิเศษที่ CorelDraw ไม่สามารถทำได้ให้ใช้ PhotoShop ทำ เทคนิคพิเศษส่วนใหญ่ได้มาจาก plug-in ของ PhotoShop

ทั่วๆไป

สำหรับเครื่องที่ใช้ควรเลือก RAM ให้มากๆเอาไว้ก่อน เพราะขณะที่ทำการพัฒนาจำเป็นต้องทำไปด้วยพร้อมทั้งแก้ไขและค้นหาข้อมูลไปพร้อมๆกัน จำเป็นต้องใช้โปรแกรมหลายตัวเปิดพร้อมๆกัน และพยายามเลือกใช้โปรแกรมให้เหมาะกับงานจะทำให้การทำงานเป็นไปอย่างรวดเร็วขึ้น สำหรับผู้ที่พัฒนาด้านกราฟฟิกความมีความเข้าใจในพื้นฐานของกราฟฟิกโดยเฉพาะสองมิติเป็นอย่างดี เพราะมีความสำคัญในการโปรแกรมและการสร้างภาพเป็นอย่างมาก และควรมีโปรแกรมกราฟฟิกอย่างน้อยหนึ่งตัวที่สามารถใช้งานได้อย่างชำนาญ

โปรแกรมที่ใช้ในการพัฒนา

Paint Shop Pro

ใช้ในการตกแต่งภาพเล็กๆน้อย เช่นการปรับความเข้มความจางของภาพ ย่อและขยายให้ได้ขนาดตามที่ต้องการ ตัดภาพเป็นชิ้นๆ แปลงชนิดของภาพ เพิ่มและลดจำนวนสีของภาพ

ข้อดี

เป็นโปรแกรมขนาดเล็กและทำงานได้รวดเร็ว ใช้งานง่ายเหมาะกับงานเล็กๆไม่ซับซ้อน โปรแกรมนี้กินทรัพยากรต่ำมากดังนั้นสามารถเปิดพร้อมกับ Visual C++ และโปรแกรมอื่นๆได้ นอกจากนี้ความสามารถในการแปลงชนิดของภาพจาก 16 ล้านสีให้เหลือ 256 สีทำได้ดีกว่าโปรแกรมอื่นมาก

ข้อเสีย

ความสามารถของโปรแกรมไม่สูงมากนัก ไม่มีเอฟเฟกมากเหมือน PhotoShop

CorelDraw 7

ใช้ในการสร้างภาพ(ไม่ใช้ตกแต่งภาพ)ที่ซับซ้อน

ข้อดี

เหมาะกับการสร้างภาพเพราะภาพที่สร้างเป็นแบบออปเจ็กและเวกเตอร์ดังนั้นสามารถย่อขยายและแก้ไขได้โดยง่าย การย่อขยายภาพแบบเวกเตอร์มีข้อดีคือ ความละเอียดของภาพไม่เสียไปเหมือนกับภาพแบบบิตแม็ป นอกจากนี้

ข้อเสีย

โปรแกรมมีขนาดใหญ่ กินทรัพยากรสูงมาก ทำงานได้ช้า

PhotoShop 4

ใช้สำหรับแก้ไขและตกแต่งภาพที่มีความซับซ้อนมาก ใช้ทำเอฟเฟกต่างๆให้กับภาพ

ข้อดี

โปรแกรมมีความสามารถสูง สามารถตกแต่งภาพที่ซับซ้อนมากได้ และมีลูกเล่นให้ใช้มาก

ข้อเสีย

โปรแกรมนี้กินทรัพยากรสูงมาก ทำงานได้ช้า

3D Studio 4

ใช้ในการสร้างภาพสามมิติเพื่อนำมาใช้กับ Direct3D หรือใช้ในการสร้างภาพสองมิติจากการเรนเดอร์ภาพสามมิติ ซึ่งเหมาะกับการสร้างภาพสองมิติเป็นเฟรมเคลื่อนไหวในมุมต่างๆ

ข้อดี

เป็นโปรแกรมสร้างภาพสามมิติที่นิยมใช้กัน(บนดอส) ในชุดพัฒนา DirectX มีโปรแกรมสำหรับแปลงภาพจาก 3D Studio ซึ่งเป็นไฟล์นามสกุล 3DS ให้เป็นนามสกุล X

ข้อเสีย

เป็นโปรแกรมที่ใช้งานยาก

Visual C++ 5

เป็นคอมไพล์เลอร์ที่ทำงานสไตล์ Visual ของทางไมโครซอฟท์ ซึ่งนำมาใช้งานร่วมกับไลบรารีของ DirectX

ข้อดี

เนื่องจากชุดพัฒนา DirectX พัฒนาโดยไมโครซอฟท์จึงสามารถใช้ร่วมกับ Visual C++ ได้เป็นอย่างดี และใน Visual C++ 5 จะมี Help สำหรับ DirectX ในตัวด้วยทำให้การค้นหาวิธีใช้งานทำได้โดยง่าย และการใช้งานของ Visual C++ ใช้งานได้ง่าย

ข้อเสีย

เนื่องจากเป็นคอมไพล์เลอร์แบบ Visual แต่การเขียนโปรแกรมโดยใช้ DirectX แบบเต็มหน้าจอเป็นส่วนใหญ่ ดังนั้นเป็นการฟุ่มเฟือยที่จะใช้ การโปรแกรมแบบ Visual ตัวคอมไพล์เลอร์มีขนาดใหญ่มาก

DirectX SDK 3,5

เป็นชุดในการพัฒนาแอปพลิเคชัน DirectX บนวินโดว์ 95/NT สามารถใช้กับคอมไพล์เลอร์ได้หลายตัว

ข้อดี

เป็นชุดพัฒนาที่แจกฟรีหาได้จาก http://www.microsoft.com/directx มีตัวอย่างโปรแกรม, Source code, Online Help, Resource และ Utility ที่ช่วยในการพัฒนามาก

ข้อเสีย

มีขนาดใหญ่มาก(แต่สามารถเลือกลงเฉพาะที่จำเป็นได้) และปัจจุบันได้เลิกให้ Download ชุดพัฒนาแล้ว ถ้าใครต้องการต้องซื้อซีดีจากทางไมโครซอฟท์

Library CDX

เป็น Wrapper function ทำเรียกใช้งาน DirectX อีกที โดยที่ไม่จำเป็นต้องรู้เกี่ยวกับ DirectX มากนัก และใช้ในการเขียนเกมส์โดยเฉพาะ ไลบรารีแจกฟรีค้นหาข้อมูลได้จาก

http://www.maidex.demon.co.uk

ข้อดี

เป็นฟังก์ชันที่ใช้งานได้ง่ายมาก มี Source code มาให้ด้วยเหมาะแก่การศึกษา

ข้อเสีย

เป็นฟังก์ชัน Wrapper ดังนั้นเราไม่สามารถทราบรายละเอียดของการทำงานของ DirectX เหมาะกับการศึกษามากกว่า(ศึกษาจาก source code ที่แจกมาให้)

การพัฒนาให้เป็นแอปพลิเคชัน

เนื่องจากชุดพัฒนา DirectX ใช้ในการพัฒนาเกมส์และโปรแกรมประเภทมัลติมีเดียโดยเฉพาะ การพัฒนาเป็นแอปพลิเคชันทางมัลติมีเดียไม่จำเป็นต้องใช้เทคนิกมากนัก ในที่นี้จะกล่าวถึงเฉพาะการพัฒนาเป็นเกมส์เนื่องจากจะเป็นต้องใช้ทักษะมากกว่า สิ่งที่ต้องคำนึงถึงในการพัฒนาเกมส์หลักๆก็มี

1.Sprite

Sprite เป็นตัวแทนตัวละครของเกมส์ ซึ่งจะเป็นกลุ่มของภาพที่แสดงเฟรมต่างๆของตัวละครเรา สามารถเก็บเป็น array ก็ได้ แต่แนะนำว่าควรจะเขียนเป็น Object Oriented จะสะดวกกว่าการเก็บเป็น array เพราะการโปรแกรมจะไม่ซับซ้อน และเราสามารถมี method สำหรับการทำงานของออปเจ็กนั้นได้ โดยให้ในหนึ่งออปเจ็กจะแทนหนึ่ง Sprite โดยจะต้องทำการสร้าง class Sprite ขึ้นมาก่อน ใน Sprite จะประกอบไปด้วย

- ข้อมูลของภาพเป็นบิตแม็ปเฟรมต่าง
- คุณสมบัติของ sprite เช่น ชื่อ,ชนิด,ขนาด ฯลฯ
- สถานะปัจจุบันของ sprite เช่นตำแหน่ง ,ความเร็ว,ทิศทาง ฯลฯ
- method ซึ่งเป็นฟังก์ชันการทำงานของ sprite

แนะนำว่าให้ไปดูการกำหนด class ของ sprite ได้ที่ไฟล์ของ library CDX ที่มาพร้อมกับรายงานชุดนี้ เพราะเป็น class ของ sprite ที่เขียนได้ดีและ เขียนเป็นแบบ class ที่ใช้งาน sprite ทั่วๆไป

2.การตรวจสอบการชน จะดูจากขนาดหรือขอบเขตของ sprite ซึ่งแนะนำว่าการเก็บข้อมูลของขอบเขตของ sprite ควรเก็บให้เป็นรูปสี่เหลี่ยมผืนผ้าเช่นในหนึ่ง sprite จะเก็บพิกัด x,y ของบนซ้าย และล่างขวา การตรวจสอบการชนจะนำพิกัดบนซ้ายและล่างขวาของทั้งสอง sprite มาเปรียบเทียบกัน การเขียนฟังก์ชันสำหรับการตรวจสอบทำได้ไม่ยาก สามารถใช้ class CRect ของวินโดว์ก็ได้เราจะมี member function สำหรับจัดการเกี่ยวกับสี่เหลี่ยมได้ง่าย ยิ่งขึ้น การตรวจสอบการชนก็สามารถใช้ฟังก์ชันเกี่ยวกับการ intersection

3.ปัญญาประดิษฐ์(Artificial Intelligent) เป็นจุดสำคัญของเกมส์อย่างหนึ่งเลยที่เดียว โดยเฉพาะเกมส์ที่มีการแข่งขันระหว่างคนและคอมพิวเตอร์ ถ้าเกมส์ไม่มี AI ที่ดีพอแล้วก็จะเล่นได้ไม่สนุก แต่มันก็เป็นส่วนที่ยากที่สุดเหมือนกันเพราะเราจะต้องทำให้เกมส์มีความคิดใกล้เคียงกับมนุษย์ แต่ในบางเกมส์จะใช้ความเร็วเพื่อการแข่งขันกับมนุษย์มากกว่า แต่ก็ไม่สามารถใช้ได้เสมอไปกับทุกเกมส์ ในที่นี้จะไม่กล่าวถึง AI มากนักเพราะเกินหัวข้อของการทำโปรเจ็กมากเกินไป

การเคลื่อนไหวของศัตรูแบบไม่ได้ใช้ AI สามารถใช้วิธีการง่ายก็คือให้ศัตรูเคลื่อนไหวเป็นรูปแบบที่แน่นอน แต่ถ้าเป็นการไล่ตามหรือหนีจะใช้วิธีดังนี้

การไล่ตามผู้เล่น จะใช้การตรวจสอบจุดอ้างอิงระหว่างผู้เล่นและศัตรู ถ้ายังไม่อยู่จุดเดียวกันก็จะเลื่อนตำแหน่งศัตรูเข้าหา โดยจะใช้อัลกอริทึมดังนี้

```
if (enemy_x < player_x)
        enemy_x++ ;
else if (enemy_x) > player_x)
        enemy_x-- ;
if (enemy_y < player_y)
        enemy_y++ ;
else if (enemy_y) > player_y)
        enemy_y-- ;
```

หรือถ้าศัตรูวิ่งหนีก็ให้ใช้วิธีตรงกันข้าม

```
if (enemy_x < player_x)
        enemy_x-- ;
else if (enemy_x) > player_x)
        enemy_x++ ;
if (enemy_y < player_y)
        enemy_y-- ;
else if (enemy_y) > player_y)
        enemy_y++ ;
```

แต่วิธีเช่นนี้เป็นวิธีที่ค่อนข้างธรรมดาเกินผู้เล่นสามารถทำนายได้ศัตรูจะไปทางไหน เราควรมีการเปลี่ยนแปลงเล็กน้อยโดยใช้การสุ่มเข้าช่วยด้วยโดยทำการสุ่มในช่วงที่กำหนด num = ran % MAX_NUM โดยที่ num ก็คือค่าระหว่าง 0 ถึง MAX_NUM ดังนั้นอัลกอริทึมไล่ตามของศัตรูที่ปรับปรุงแล้วจะเป็นดังนี้

```
if (enemy_x < player_x)
        enemy_x += (ran % 2) +1 ;
else if (enemy_x) > player_x)
        enemy_x -= (ran % 2) +1 ;
```

```
if (enemy_y < player_y)
        enemy_y += (ran % 2) +1  ;
else if (enemy_y) > player_y)
        enemy_y -= (ran % 2) +1  ;
```

ในกรณีที่มีศัตรูหลายตัวเราสามารถใช้วิธีดังนี้

```
for( int  j = 0 ; j <  NUM_ENERMIES; j++)
{
if (enemy[j].x < player_x)
        enemy[j].x++ ;
else if (enemy[j].x) > player_x)
        enemy[j].x-- ;
if (enemy[j].y < player_y)
        enemy[j].y++ ;
else if (enemy[j].y) > player_y)
        enemy[j].y-- ;
}
```

กิริยาของศัตรูที่กระทำต่อผู่เล่นในเกมส์ส่วนใหญ่มักสามารถคาดเดาได้โดยง่ายซึ่งทำให้เกมส์เล่นได้ง่ายเกินไปดังนั้นเราจะใช้การสุ่มการกระทำของศัตรูที่กระทำต่อผู้เล่น แต่ก็ไม่ใช่ว่าจะใช้การสุ่มโดยสมบูรณ์แบบเราจะใช้ค่าของความเป็นไปได้ของการกระทำโดยจะใช้ค่าแบบถ่วงนำหนักดังตัวอย่า

```
        CHASING         0.4
        AVOIDING        0.1
        ATTACKING       0.3
        RANDOM          0.2
```

จากค่าที่ถ่วงน้ำหนักเรานำมาใส่ตารางได้ดังนี้

```
int     state_table[10] = {CHASING,CHASING,CHASING ,CHASING, AVOIDING,
ATTACKING, ATTACKING, ATTACKING, RANDOM , RANDOM}
```

และทำการสุ่มในช่วงที่กำหนด

```
int     state = state_table[  ran() % 10];
```

ภาคผนวก ฐ
การติดตั้งชุดพัฒนา DirectX

ชุดพัฒนา DirectX หรือ DirectX Software Development Kit (DirectX SDK) ที่ให้มาเป็นเวอร์ชัน 5 จาก http://www.microsoft.com ซึ่งใช้ในการพัฒนาโปรแกรมในโปรเจ็กนี้

วิธีการติดตั้งชุดพัฒนา DirectX

1.ขยายไฟล์ก่อนการติดตั้งโดยใช้โปรแกรม pkunzip ไปไว้ยังไดเรกทอรีชั่วคราวก่อน โดยใช้คำสั่ง

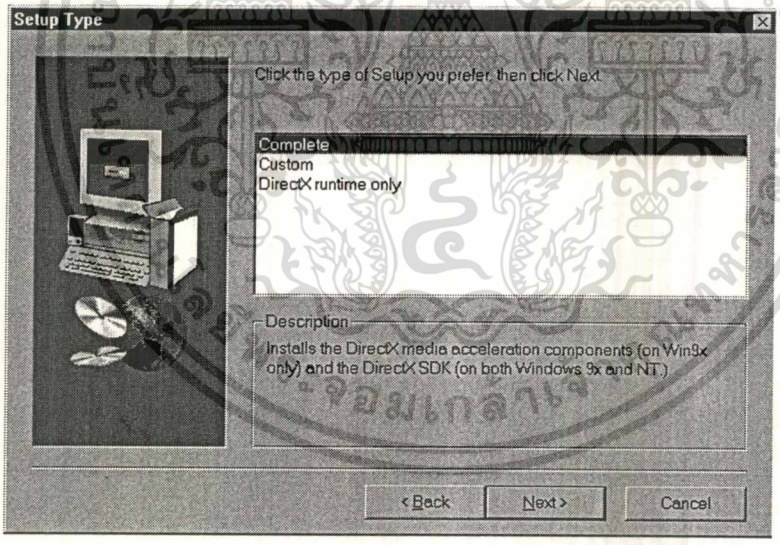
C:\TEMP>**pkunzip –d dx5sdk.zip**

2.เมื่อขยายไฟล์เสร็จแล้วจะมีโปรแกรม setup.exe แล้วเรียกโปรแกรมนี้เพื่อการติดตั้งต่อไป

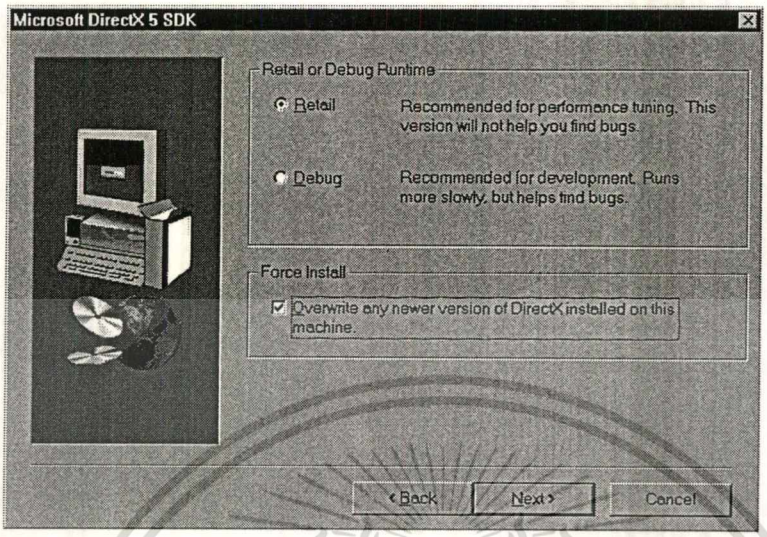
3.จากนั้นอ่านรายละเอียดต่างๆ เช่น โปรแกรมที่กำลังติดตั้งอยู่, ข้อตกลงเกี่ยวกับลิขสิทธิ์

4.เมื่อถึงหน้าต่างเลือกการติดตั้ง เพื่อความสะดวกจะเลือก Complete เพื่อติดตั้ง

- ไลบรารีของ DirectX สำหรับใช้เมื่อเขียนโปรแกรม
- DirectX Driver สำหรับติดตั้งเข้ากับระบบ



5.เลือก Retail สำหรับการพัฒนาโปรแกรมบน DirectX และเลือก Force Intsall เพื่อให้ทับส่วนที่มีอยู่แล้ว จะได้เข้ากับส่วนที่เป็นไลบรารีที่ติดตั้งมากับชุดนี้ด้วย



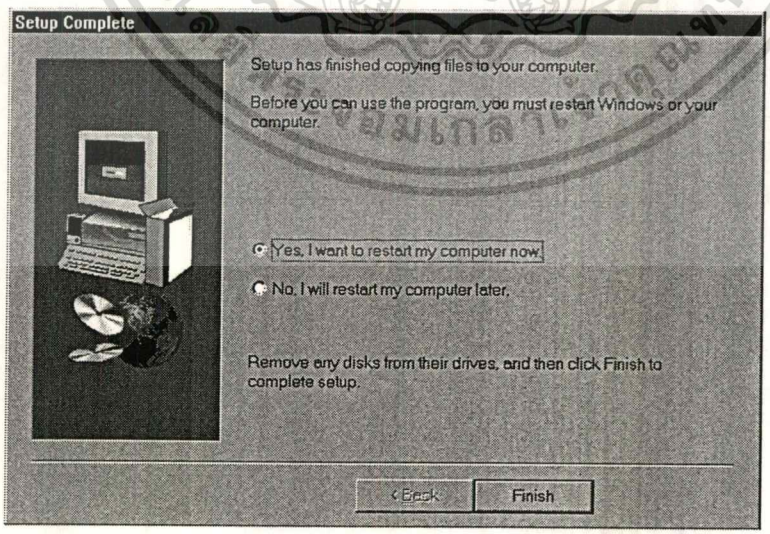
6.เมื่อถึงหน้าต่าง Optional Component ไม่ต้องเลือกคอมโปเน้นใดๆ เพราะไฟล์ที่ให้มานี้ไม่มีคอมโปเน้นเหล่านี้ด้วย สามารถเลือก Next ผ่านไปได้เลย

7.เลือกปลายทางที่จะคัดลอกไฟล์ไปไว้ ได้แก่ ไลบรารี, ตัวอย่างโปรแกรม, คู่มือการใช้งานไลบรารี และอื่นๆ

8.กำหนดกลุ่มที่จะให้เรียกใช้งาน ซึ่งกลุ่มที่กำหนดให้อยู่แล้ว คือ Microsoft DirectX 5 SDK

9.เมื่อกำหนดค่าต่างๆแล้ว โปรแกรมติดตั้งจะคัดลอกไฟล์ต่างๆ และติดตั้ง DirectX Driver ลงบนระบบ

10.หลังจากติดตั้งเสร็จแล้ว จะต้องออกจากระบบแล้วกลับเข้ามาใหม่ถึงจะใช้งานได้



กิตติกรรมประกาศ

การจัดทำปริญญานิพนธ์นี้สำเร็จลุล่วงไปได้ด้วยดี ก็ด้วยความกรุณาของ ดร.บุญธีร์ เครือตราชู ที่ให้คำแนะนำ เอื้อเฟื้อเครื่องและสถานที่ในการทำงานครั้งนี้ อีกทั้งเพื่อนๆ และน้องๆ ที่ช่วยกันทดลองโปรแกรมทดสอบที่เขียนขึ้นมา และขอขอบคุณที่ทางไมโครซอฟท์ได้ให้ Download ชุดพัฒนา DirectX โดยไม่ต้องเสียค่าใช้จ่าย นอกจากนี้ขอขอบคุณทุกคนที่เป็นกำลังใจในเรื่องการเรียนและการทำงาน

หนังสือและเอกสารข้อมูลอ้างอิง

ก.หนังสืออ้างอิง

[1] Jason Kolb: "WIN32 GAME DEVELOPER'S GUIDE WITH DirectX 3", THE WAITE GROUP, February 1, 1997, ISBN: 1571690301

[2] STEPHEN PRATA: "C++ Primer Plus", 3rd Bk&cdr Edition, THE WAITE GROUP PRESS, March 1998 , ISBN: 1571691316

ข.เอกสารอิเล็กทรอนิค

เอกสารจาก Platform SDK และ DDK จาก Visual C++ 5

Help ของ DirectX SDK

ค.แหล่งข้อมูลบนอินเตอร์เน็ต

http://www.microsoft.com/directx

http://www.maidex.demon.co.uk/

http://www.devgames.com/

http://www.geocities.com/SiliconValley/Way/3390/