การพัฒนาเกมโดยใช้ไดเร็กเอ็กซ์

GAME DEVELOPMENT USING DIRECTX

โดย

นางสาวกนกกานต์ ยันตะกนก

นางสาวกฤติกา สุขทรัพย์วศิน

อาจารย์ที่ปรึกษา

ดร. วรวัฒน์ ลิ้มโภคา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2542

ปริญญานิพนธ์ปีการศึกษา 2542

ภาควิชา วิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาเกมโดยใช้ไดเร็กเอ็กซ์

GAME DEVELOPMENT USING DIRECTX

ผู้จัดทำ

1. นางสาวกนกกานต์ ยันตะกนก รหัสประจำตัว 39014001
2. นางสาวกฤติกา สุขทรัพย์วศิน รหัสประจำตัว 39014010

อาจารย์ที่ปรึกษา

(ดร. วรวัฒน์ ลิ้มโภคา)

การพัฒนาเกมโดยใช้ไดเร็กเอ็กซ์

นางสาวกนกกานต์ ยันตะกนก 39014001
นางสาวกฤติกา สุขทรัพย์วศิน 39014010
ดร. วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษา
ปีการศึกษา 2542

บทคัดย่อ

การพัฒนาเกมโดยใช้ไดเร็กเอ็กซ์เป็นโครงงานที่กล่าวถึงการเขียนโปรแกรมเกม 3 มิติ โดยใช้ไดเร็กเอ็กซ์เป็นเครื่องมือช่วยในการพัฒนาโปรแกรม ไดเร็กเอกซ์เป็นเทคโนโลยีที่ถูกสร้างขึ้นมาเพื่อช่วยให้ผู้พัฒนาซอฟต์แวร์ ดึงความสามารถของอุปกรณ์ฮาร์ดแวร์ต่างๆ มาใช้อย่างมีประสิทธิภาพและให้เกิดประโยชน์สูงสุด

ขอบเขตของโครงงานนี้ คือการพัฒนาเกมคอมพิวเตอร์ 3 มิติ โดยมีไดเร็กเอ็กซ์เวอร์ชัน 6 เป็นเครื่องมือในการช่วยเขียนโปรแกรม เริ่มจากการสร้างโมเดลจำลองฉากภายในเกมด้วยโปรแกรม สามดีสตูดีโอ แม็กซ์ แล้วจึงใช้ไดเร็กสามดี ในโหมดรีเทนควบคุมวัตถุ 3 มิติต่างๆ ภายในฉาก รวมถึงการควบคุมลำดับเหตุการณ์ที่เกิดขึ้นภายในเกม ส่วนการแสดงผลทางจอภาพและอุปกรณ์ฮาร์ดแวร์ต่างๆ สามารถควบคุมโดยผ่านไดเร็กดรอว์ การควบคุมดนตรีประกอบเกมโดยไดเร็กซาวด์ และการตอบสนองอินพุตจากผู้เล่นโดยใช้ไดเร็กอินพุต

Game Development Using DirectX

Kanokkan Yantakanok
Krittika Suksubwasin
Dr. Worawat Limpoka Advisor

ABSTRACT

Game Development Using DirectX is the project that describes 3D game programming that uses DirectX SDK (Software Development Kit). DirectX technology is produced to provide service for software developer to get the hardware optimize performance.

This thesis is concerned with 3D computer game development using DirectX 6 as a programming tool. First, build a scene using 3D Studio MAX. Then control 3D objects in the scene and game event through Direct3D Retained Mode. DirectDraw is applied for displaying images on the monitor and hardware device management . Make game soundtrack and sound effect by DirectSound. Finally response user input via through DirectInput.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือ และร่วมมือจากหลาย ๆ ฝ่ายด้วยกัน บุคคลแรกที่ต้องกล่าวถึงเพราะเป็นส่วนสำคัญที่ทำให้วิทยานิพนธ์นี้เสร็จลงได้ก็คือ อาจารย์ วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษาวิทยานิพนธ์ ที่ให้ความเอาใจใส่ แนะนำ และช่วยเหลือเสมอมา ซึ่งต้องขอขอบพระคุณเป็นอย่างมาก

และต้องขอขอบพระคุณบุคคลสำคัญที่สุดที่ทำให้ข้าพเจ้ามีวันนี้ ก็คือ บิดา มารดา อันเป็นที่เคารพรักยิ่ง ซึ่งได้เลี้ยงดูผู้เขียนมาเป็นอย่างดี พร้อมทั้งให้โอกาสในการศึกษาอย่างเต็มที่ และยังให้กำลังใจ เอาใจใส่เสมอมา ในทุก ๆ ด้านอันหาที่เปรียบมิได้ ข้าพเจ้าขอระลึกในพระคุณอันสุดประมาณ และขอกราบขอบพระคุณมา ณ ที่นี้

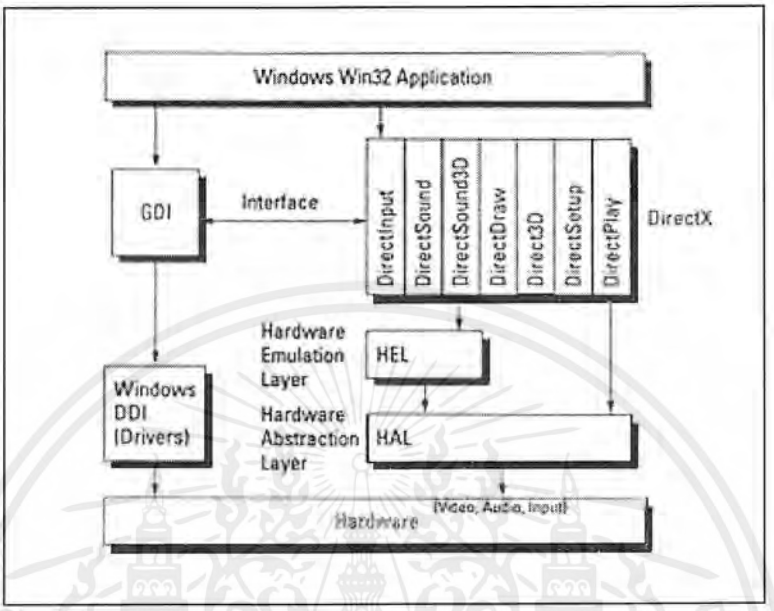
กนกกานต์ ยันตะกนก
กฤติกา สุขทรัพย์วศิน

สารบัญ

หน้าที่

สารบัญภาพ

คอมโพเนนท์ต่างๆ ของ DirectX จะครอบคลุมการออกแบบวิดีโอเกมในแต่ละด้าน กล่าวคือ กราฟิก เสียง อินพุต ภาพ 3 มิติ และระบบเครือข่าย รูปข้างล่างแสดงถึงคอมโพเนนท์ภายใน DirectX และความสัมพันธ์กับ Win32 GDI และฮาร์ดแวร์



รูปที่ 1-1 คอมโพเนนท์ภายใน DirectX และความสัมพันธ์กับ Win32 GDI และฮาร์ดแวร์

คอมโพเนนท์ของ DirectX เวอร์ชัน 6 ที่สำคัญมีดังนี้

DirectDraw เป็นส่วนประกอบหลัก เนื่องจากมีหน้าที่จัดการกับการแสดงผลทางจอภาพ และให้ความเร็วในการประมวลผลสูงกว่าเกมบนดอสและการใช้ GDI (Graphics Device Interface) ของวินโดวส์ เนื่องจากการส่งข้อมูลสามารถทำได้โดยตรง อีกทั้งยังประกอบไปด้วยฟังก์ชันต่าง ๆ ที่คอยสนับสนุนเกม 2 มิติที่มีประสิทธิภาพ

Direct3D มีประสิทธิภาพสูงมาก ใช้ในการสร้างเกม 3 มิติบนวินโดวส์ ทำให้ได้รับประโยชน์จากการใช้งานร่วมกับการ์ดเร่งความเร็ว 3 มิติอย่างมีประสิทธิภาพ เพราะมีฟังก์ชันสนับสนุนอยู่ มีส่วนประกอบอยู่ 2 ส่วน คือ Direct3D Retained Mode API และ Direct3D Immediate Mode API

DirectInput ทำให้การสร้างเกมเพื่อรองรับระบบการสั่งงานจากผู้เล่นโดยใช้จอยสติก เมาส์ หรือคีย์บอร์ดทำได้ง่ายและมีประสิทธิภาพเป็นอย่างยิ่ง อีกทั้งยังมีฟังก์ชันที่สนับสนุนการเพิ่มอุปกรณ์การเล่นที่จะมีขึ้นมาใหม่ในอนาคต

DirectSound	มีฟังก์ชันที่ช่วยให้โปรแกรมเมอร์ทำงานได้ง่ายขึ้น ในด้านการใส่เสียงประกอบและเสียงเอฟเฟ็กต์ต่าง ๆ เข้าไปในเกม โดยถ้าผู้เขียนเกมทำความเข้าใจกับหลักสถาปัตยกรรมของ DirectSound แล้วจะสามารถทำงานได้อย่างมีประสิทธิภาพ
DirectSound3D	มีการทำงานพื้นฐานเหมือนกับ DirectSound แต่เพิ่มความสามารถในการทำเสียง 3 มิติ ทฤษฎีที่ใช้ทำระบบเสียงแบบ 3 มิติ มีแนวคิดว่าจะทำให้วัตถุทุกชนิดสามารถให้กำเนิดเสียงได้ไม่ว่าจะอยู่ในตำแหน่งใดก็ตาม โดยควบคุมการแสดงสัญญาณเสียงเข้าสู่หูด้านซ้ายและขวา
DirectMusic	มีลักษณะคล้ายกับ DirectSound แต่จะมีฟังก์ชันที่คอยสนับสนุนการใช้งานไฟล์รูปแบบ MIDI
DirectPlay	ทำให้การสร้างระบบเกมที่มีผู้เล่นหลายคนทำได้ง่ายยิ่งขึ้น และอำนวยสะดวกในการใช้งานร่วมกับโมเด็มและระบบเครือข่ายคอมพิวเตอร์
DierctAnimation	ทำหน้าที่ในการนำสื่อพื้นฐานต่างๆ เช่น ภาพและเสียง มาประกอบรวมกันเป็นภาพเคลื่อนไหว
DirectShow	ใช้ในการองค์ประกอบต่างๆ มาแสดงผลในรูปของสื่อมัลติมีเดีย

1.2 วัตถุประสงค์ของโครงงาน

1.2.1 ศึกษาสถาปัตยกรรมของ DirectX และการทำงานของแอปพลิเคชั่นเมื่อใช้ DirectX โดย DirectX จะมีส่วนช่วยผู้เขียนโปรแกรมให้สามารถติดต่อกับฮาร์ดแวร์บนเครื่องคอมพิวเตอร์ได้อย่างไร เพื่อเป็นแนวทางในการนำ DirectX มาใช้ในการเขียนโปรแกรมมัลติมีเดียหรือเกม

1.2.2 ศึกษาขั้นตอนในการเขียนโปรแกรมเกมบนระบบปฏิบัติการวินโดวส์ และศึกษาเครื่องมือทางกราฟิกเพื่อใช้ในการจำลองวัตถุเป็นโมเดล 3 มิติ

1.2.3 ทำการพัฒนาเกม 3 มิติบนระบบปฏิบัติการวินโดวส์โดยเลือกใช้คอมโพเนนท์ของ DirectX ที่ช่วยในการติดต่อกับฮาร์ดแวร์ เพื่อให้ได้เกม 3 มิติที่มีประสิทธิภาพ

1.3 ขอบเขตของโครงงาน

เกมคอมพิวเตอร์ 3 มิติ ที่พัฒนาขึ้นในโครงงานนี้ ใช้ DirectX SDK เวอร์ชัน 6 โดยใช้ Direct3D Retained Mode ในการควบคุมวัตถุ 3 ต่างๆ ในเกม เช่น การสร้างวัตถุ 3 มิติในฉาก การกำหนดตำแหน่งและขนาด รวมถึงการเคลื่อนไหวของวัตถุนั้น ส่วนการแสดงผลออกทางจอภาพและอุปกรณ์ฮาร์ดแวร์ต่างๆ สามารถควบคุมโดย DirectDraw หรือ Direct3D Immediate Mode แต่เกมที่พัฒนาขึ้นในโครงงานนี้ใช้เพียง DirectDraw เท่านั้น การควบคุมดนตรีประกอบโดย DirectSound และการตอบสนองอินพุตจากผู้เล่นเกมโดยใช้ DirectInput

ปัจจัยหลักที่นำมาใช้ในการตัดสินใจว่าจะเลือกใช้เครื่องมือใดเป็นหลักในการเขียนโปรแกรม คือ เครื่องมือนั้นสามารถทำงานร่วมกับ DirectX Software Development Kit (SDK) ได้ดีเพียงใด DirectX สามารถทำให้การเคลื่อนไหวที่มีประสิทธิภาพบนวินโดวส์เกิดขึ้นได้ รวมถึงการเล่นเสียง การตอบสนองอินพุตจากผู้เล่นเกม และการควบคุมเกมที่มีผู้เล่นหลายคน Microsoft Visual C++ เป็นเครื่องมือที่ดูเหมือนจะสามารถใช้งานร่วมกับ DirectX ได้ดีที่สุด (นับตั้งแต่เวอร์ชัน 4.1 ซึ่งมี DirectX SDK มาพร้อมกันด้วย) อย่างไรก็ตามเครื่องมืออื่นๆ เช่น Delphi ก็สามารถใช้ได้เช่นกัน ส่วนเครื่องมือในการสร้างวัตถุ 3 มิติ ในที่นี้ใช้โปรแกรม 3D Studio MAX และใช้โปรแกรม Adobe PhotoShop ในการตกแต่งภาพพื้นผิวของวัตถุบางชนิด

1.4 วิธีการดำเนินงาน

โครงงานนี้เริ่มต้นจากการศึกษาทฤษฎีพื้นฐานต่างๆ ที่ต้องใช้ในการเขียนโปรแกรมบนระบบปฏิบัติการวินโดวส์ ศึกษาสถาปัตยกรรมของ DirectX เพื่อเลือกคอมโพเนนท์ของ DirectX มาใช้ในการพัฒนาโปรแกรมเกมดังกล่าวถึงในหัวข้อความสำคัญและที่มาในบทนี้ ต่อมาจึงได้ออกแบบและคิดรูปแบบเกมที่ต้องการจะพัฒนา ในที่นี้ต้องการพัฒนาเกม 3 มิติที่จำลองการขับยานชมบริเวณคณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีเจ้าคุณทหารลาดกระบัง แล้วจึงเริ่มจำลองวัตถุต่างๆ ที่ต้องการให้มีในฉากซึ่งก็คือตึกต่างๆ ภายในคณะวิศวกรรมศาสตร์เป็นโมเดล 3 มิติโดยใช้โปรแกรม 3D Studio MAX ในการสร้างโมเดล 3 มิติซึ่งมีรายละเอียดดังในบทที่ 2

หลังจากนั้นจึงเริ่มเขียนโปรแกรมโดยใช้คอมโพเนนท์ต่างๆ ของ DirectX ซึ่งในโครงงานนี้จะใช้ Direct3D Retained Mode ในการควบคุมวัตถุ 3 มิติต่างๆ ในเกม เช่น การสร้างวัตถุ 3 มิติในฉาก การกำหนดตำแหน่งและขนาด รวมถึงการเคลื่อนไหวของวัตถุนั้น ส่วนการแสดงผลออกทางจอภาพและอุปกรณ์ฮาร์ดแวร์ต่างๆ สามารถควบคุมโดย DirectDraw หรือ Direct3D Immediate Mode แต่เกมที่พัฒนาขึ้นในโครงงานนี้ใช้เพียง DirectDraw เท่านั้น การควบคุมดนตรีประกอบโดย DirectSound และการตอบสนองอินพุตจากผู้เล่นเกมโดยใช้ DirectInput โดยในบทที่ 3,4,5 และ 6 จะกล่าวถึงรายละเอียดทางด้านทฤษฎีและการใช้งาน DirectDraw, Direct3D, DirectSound และ DirectInput ตามลำดับ

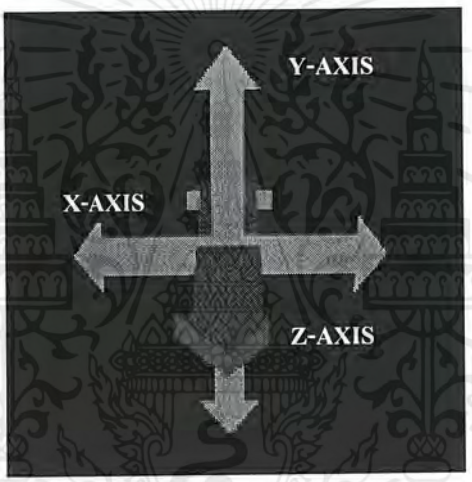
สำหรับบทที่ 7 เป็นการแสดงผลลัพธ์ที่ได้จากโครงงาน ซึ่งก็คือเกม 3 มิติที่ได้ทำการพัฒนาขึ้นมา และบทที่ 8 ซึ่งเป็นบทสุดท้ายจะเป็นการสรุปการทำงาน ผลที่ได้รับจากการทำโครงงานนี้ และแนวทางในการวิจัยเพิ่มเติม

บทที่ 2

3D Studio MAX

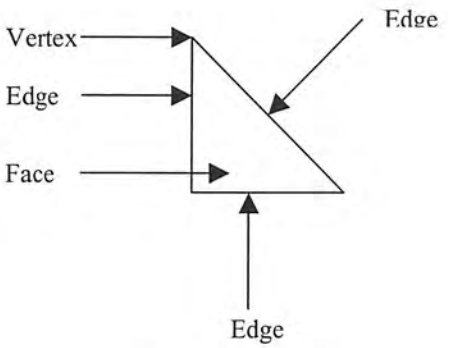
2.1 หลักการสร้างกราฟิก 3 มิติด้วยโปรแกรม 3D Studio Max

ในการทำงานบนโปรแกรม 3D Studio Max นั้นจะต้องทำความเข้าใจก่อนว่า กำลังติดต่อกับโลกเสมือนที่ถูกสร้างขึ้นโดยคอมพิวเตอร์ ดังนั้นจึงต้องเข้าใจวิธีการแสดงและเก็บวัตถุต่างๆ ภายในโลกเสมือนนี้ด้วย คือ วัตถุทุกๆ ชิ้นมีโคออร์ดิเนตของตัวมันเอง และถูกสร้างขึ้นจากรูปทรงทางคณิตศาสตร์ หรือกล่าวได้ว่า วัตถุมีลักษณะเป็น 3D space บน cyberspace (สร้างและอยู่เฉพาะบนซอฟต์แวร์เท่านั้น) และทุกๆ จุดภายใน cyberspace มี 3 โคออร์ดิเนต คือ การแสดงความสูง ความกว้าง และความลึกของจุด ซึ่งแต่ละ coordinate จะเรียงตัวไปตามแกนเฉพาะหนึ่งแกน (axis) คือ ความกว้างถูกแสดงด้วยแกน x ความลึกแสดงด้วยแกน y ความสูงแสดงด้วยแกน z



รูปที่ 2-1 ระบบพิกัด 3 มิติ

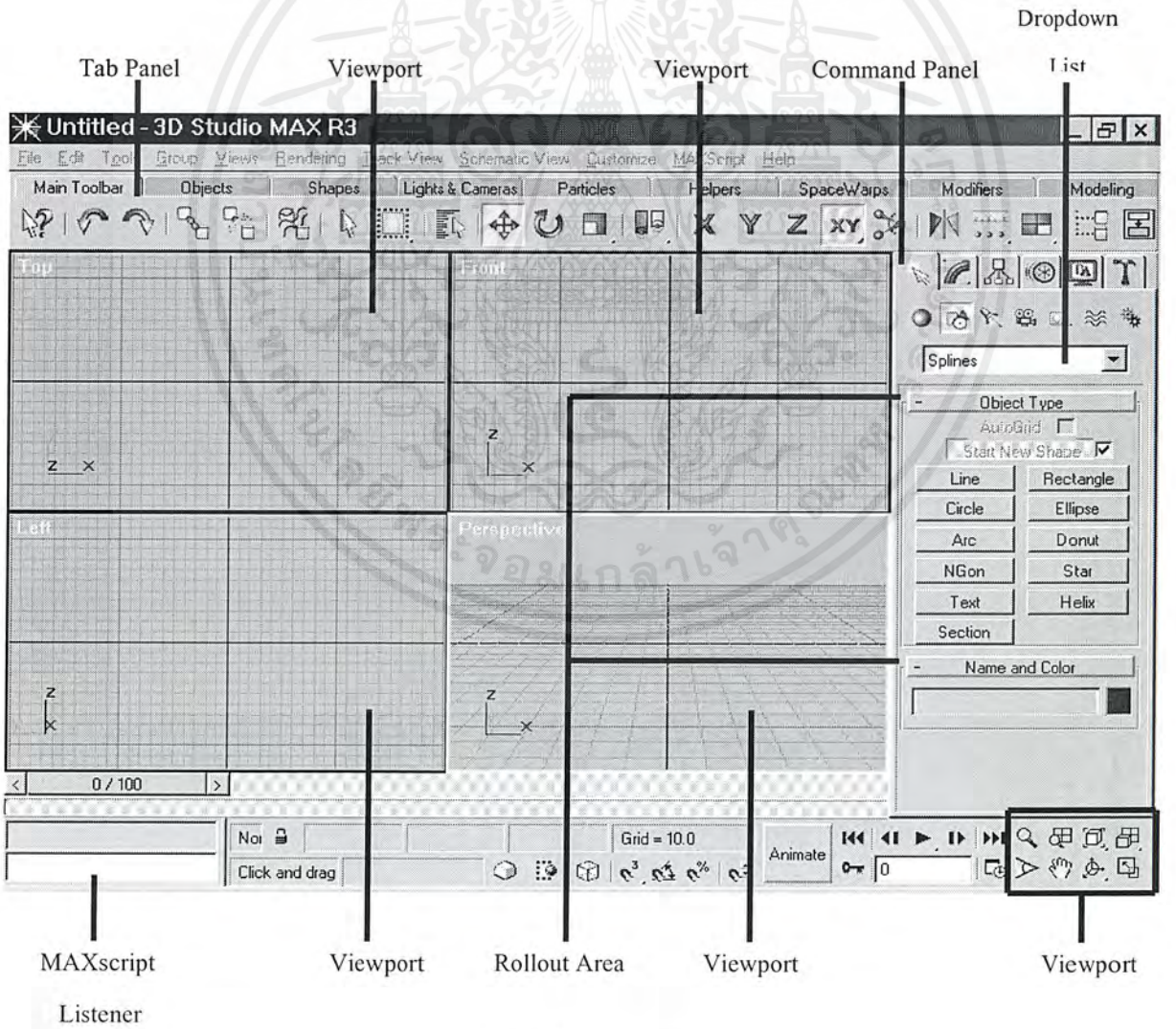
- Line : เส้นที่เกิดจากการเชื่อมจุด 2 จุด
- Polyline : เส้นหลายเส้นที่ทำให้เกิดเป็นด้าน (segment)
- Spline : ใน 3D Studio Max หมายถึง line และ polyline
- Closed shape : เกิดจากการสร้าง polyline และเชื่อมจุดเริ่มต้นกับจุดสุดท้าย และ closed shape อย่างง่ายที่สุด คือ โพลีกอน (polygon) ที่มี 3 ด้าน หรือสามเหลี่ยม หรือเรียกว่า face ซึ่ง face เป็นวัตถุพื้นฐานในระบบ และการสร้างวัตถุที่มีความซับซ้อนมากขึ้นใน 3D Studio Max ล้วนแต่สร้างมาจากโพลีกอนทั้ง 3 ด้านและ 4 ด้าน (สี่เหลี่ยม)



รูปที่ 2-2 Face

2.2 อินเทอร์เฟซภายในโปรแกรม 3D Studio Max

2.2.1 Viewport



รูปที่ 2-3Viewport

ในหน้าจอหลัก 3D Studio Max สามารถเลือกที่จะแสดง viewportได้พร้อมกัน 4 viewport หรือแสดงเพียง 1 viewport เท่านั้นได้เช่นกัน viewport ใน 3D Studio Max ได้แก่ Top Bottom Left Right Perspective และ camera (ถ้ามี) และคลิกเปลี่ยนให้แสดง viewport ใดๆ ได้โดยกดปุ่มตัวอักษรบนคีย์บอร์ด เช่น Top กด T Bottom กด B Left กด L Right กด R Front กด F Back กด K Perspective กด P Camera กด C และ Spotlight กด $ หรือคลิกเมาส์ขวาที่มุมบนซ้ายของ viewport ใดก็ได้ แสดงเมนู pop-up เลือก View และเลือก viewport ที่ต้องการ

Command Panel : เป็นอินเทอร์เฟซหลักที่ติดต่อกับผู้ใช้ ประกอบด้วย 6 commands คือ Create/Modify/ Hierarchy/Motion/Display/Utility

สำหรับโมเดลที่สร้างในโครงงานนี้จะใช้เพียง Create และ Modify command เท่านั้น

- Create command : เป็น command ที่ใช้ในการสร้างวัตถุทั้งหมดใน 3D Studio Max และส่วนบนเป็นปุ่มคำสั่ง 7 ปุ่ม คือ Geometry/Shapes/Lights/Cameras/Helpers/Space Warps/Systems และภายในแต่ละปุ่มประกอบด้วย dropdown list ซึ่งแบ่งรูปทรงออกเป็นประเภทหลัก และส่วน rollout ซึ่งรวมคำสั่งย่อยที่ช่วยในการสร้างวัตถุ
- Modify command : เป็น command ที่ใช้แก้ไขวัตถุที่สร้างขึ้นจาก create command

การเรียกใช้งานส่วน rollout สามารถทำได้ 2 วิธี คือ

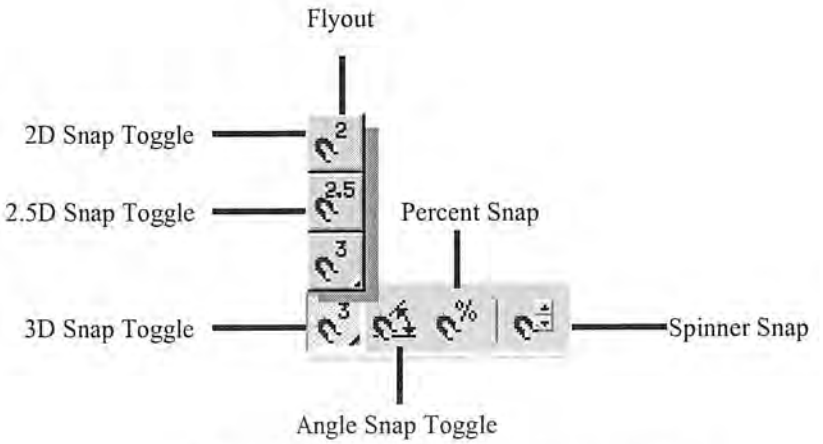
- คลิกเปิดปิดเพื่อขยายส่วน rollout
- คลิกเมาส์ขวาบริเวณ rollout area แสดงเมนู pop-up เพื่อเลือกเปิดและปิด rollout ทั้งหมดที่อยู่ภายใต้ปุ่มคำสั่งต่างๆ ได้

Tab Panel : เป็นอินเทอร์เฟซที่ติดต่อกับผู้ใช้อย่างหนึ่งที่เพิ่มขึ้นใน 3D Studio Max R3.0 ประกอบด้วย 11 tab หลัก คือ Main Toolbar/Objects/Shapes/Lights&Cameras/Participles/Helpers/Spacewarps/Modifiers /Modeling/Rendering และการคลิกเมาส์ขวาบริเวณ tab panel แสดงเมนู pop-up เพื่อเพิ่มเติม ลบหรือเปลี่ยนชื่อ tab สามารถทำได้โดย

2.2.2 เครื่องมือช่วยในการวาดอื่นๆ

เครื่องมือช่วยในการวาดอื่นๆ ที่ใช้ในโครงงานนี้ประกอบด้วย

- การกำหนดหน่วย : เลือก Customize (ในเมนู pull-down) >Unit Setup ซึ่งช่วยในการกำหนดประเภทของหน่วยที่ใช้ในการสร้างวัตถุ
- การกำหนด snap : เลือกการเลื่อนของเคอร์เซอร์ในแบบต่างๆ ซึ่งส่วนใหญ่จะใช้แบบ vertex คือ เคลื่อนเคอร์เซอร์ทีละหนึ่งช่อง (grid) ทำได้โดยเลือก Customize>Grid and Snap Settings>Snaps Tab>Vertex check box หรือเลือกปุ่ม snap เพื่อกำหนด/ยกเลิกการกำหนด snap และถ้าคลิกเมาส์ค้างไว้จะแสดงการกำหนด snap ทีละ Vertex แบบอื่นๆ ให้เลือกในลักษณะ flyout



รูปที่ 2-4 อินเทอร์เฟซของการกำหนด snap แบบต่างๆ

การกำหนด snap แบบอื่นๆ ได้แก่

- Angle Snap Toggle : การปรับมุมทีละ 5 องศา เมื่อมีการหมุนวัตถุ
- Percent Snap : การปรับขนาดทีละ 10 เปอร์เซ็นต์
- Spinner Snap : การปรับค่าใน spinner (เป็นช่องว่างสำหรับเติมค่าใดๆ หรือคลิกเมาส์ที่ปุ่มเพิ่มค่า/ลดค่า) ทีละ 1

<u>เพิ่มเติม</u> : ซึ่งสามารถปรับเปลี่ยนค่า snap เหล่านี้ให้เพิ่ม/ลดทีละเท่าใดได้โดยเลือก Customize>Grid and Snap Settings>Options Tab ส่วนการกำหนด snap แบบ Vertex ให้เพิ่ม/ลดเท่าใด คือ การกำหนดขนาด grid

- การกำหนดขนาดช่อง (grid) ใน 3D Studio Max : เลือก Customize>Grid and Snap Settings>Home Grid Tab ซึ่งสามารมเติมค่าขนาดช่องใน Grid Spacing spinner (ค่าเริ่มต้นที่กำหนดไว้ใน 3D Studio Max เป็น 10) และเติมจำนวนเส้น/ช่องเล็กๆ ในหนึ่งช่องใหญ่ใน Major Lines every Nth spinner (เริ่มต้นเป็น 10)
- การรวมวัตถุให้เป็นก้อนเดียวกัน : เลือกวัตถุที่ต้องการรวมกลุ่มทั้งหมดก่อน จากนั้นเลือก Group (ในเมนู pull-down) >Group และตั้งชื่อกลุ่มวัตถุ
- การเลือกวัตถุ : มี 5 แบบ คือ
  - การเลือกโดยใช้เมาส์คลิกเลือกวัตถุที่ต้องการ (สังเกตได้จากวัตถุที่ยังไม่ถูกเลือกเมาส์เคอร์เซอร์จะเป็นเครื่องหมายบวก) และสามารถเลือกวัตถุได้หลายๆ ชิ้นโดยการกดปุ่ม Ctrl บนคีย์บอร์ดค้างไว้แล้วคลิกเมาส์เลือกวัตถุอื่นต่อไป
  - การเลือกแบบ windowed ทำได้โดยการลากเมาส์ซ้ายเป็นกรอบสี่เหลี่ยม ซึ่งแบ่งเป็น 2 ประเภทย่อย คือ windowed เลือกวัตถุนอกกรอบ และ crossing เลือกวัตถุที่อยู่ภายใน และสัมผัสกรอบ โดยเลือก Edit (เมนู pull-down) >Region>windowed/crossing ซึ่งการเลือกวัตถุแบบนี้ควรเลือกประเภทวัตถุใน Selection Filter drop-down list ก่อน

- การเลือกจากชื่อวัตถุ โดยคลิกปุ่ม Select by Name ใน Main Toolbar และเลือกวัตถุจากรายการ หรือเลือก Edit (เมนู pull-down) >Select by>Name
- การเลือกจากสีวัตถุ โดยเลือก Edit (เมนู pull-down) >Select by>Color
- การเลือกจากกลุ่มวัตถุ โดยเลือก Edit (เมนู pull-down) >Edit Named Selections

2.2.3 การ Merge ไฟล์

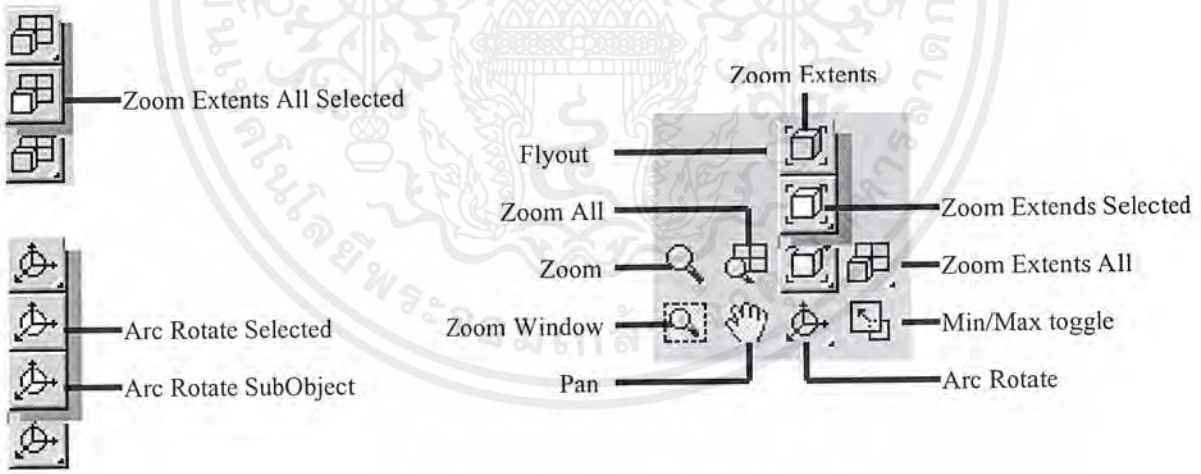
การโหลดไฟล์และรวมเข้ากับไฟล์ที่แสดงอยู่ หรือเลือกรวมเฉพาะวัตถุได้เช่นกัน โดยเลือก File (เมนู pull-down) >Merge และ browse ไฟล์ที่ต้องการ แล้วเลือกวัตถุที่ต้องการจากรายการ

2.2.4 การ Import/Export ไฟล์

สามารถโหลไฟล์ในรูปแบบอื่นได้ ซึ่งใน 3D Studio Max R3.0 สามารถ import และรองรับการแสดงไฟล์ประเภท 3D Studio Max R4.0 ASE DXF STL DWG และ VRML และ export เป็นไฟล์ประเภท 3D Studio Max R4.0 (3DS PRJ และ SHP)

2.2.5 การใช้งาน Viewport Controls

เครื่องมือ หรือคำสั่งควบคุม viewport เป็นอินเทอร์เฟซอยู่ที่มุมล่าง-ขวาของหน้าจอ 3D Studio Max และอินเทอร์เฟซควบคุม camera viewport มีความแตกต่างจาก viewport อื่นๆ ทั้งหมด



รูปที่ 2-5 อินเทอร์เฟซของ Viewport Controls

Zoom : เป็นคำสั่งย่อ-ขยายภาพบน viewport ที่ active อยู่ คลิกเมาส์ลากขึ้น คือ ขยายภาพ (Zoom-In) ลากลง คือ ย่อภาพ (Zoom-out)

Zoom All : คำสั่งนี้เหมือนกับคำสั่งที่แล้ว แต่มีผลกับทุก viewport

Zoom Extents : เป็นของคำสั่งย่อ-ขยายภาพ แต่จะทำจนกว่าจะมองเห็นวัตถุทั้งหมดใน viewport และมีคำสั่ง Zoom Extents Selected เป็น flyout สำหรับเลือกเฉพาะวัตถุที่ต้องการ การเลือกคำสั่งให้คลิกเมาส์ค้าง

ไว้ที่ปุ่มจะปรากฏ flyout ขึ้นมาให้เลือก (ให้สังเกตว่า หากที่มุมล่าง-ขวาของปุ่มใดมีรูปสามเหลี่ยมเล็กๆ อยู่ แสดงว่า มีปุ่ม flyout สำหรับเลือกคำสั่งเพิ่มเติม)

Zoom Extents All : คำสั่งนี้เหมือนกับคำสั่งที่แล้ว แต่มีผลกับทุก viewport ในเวลาเดียวกัน และมีคำสั่ง Zoom Extents All Selected เป็น flyout สำหรับเลือกเฉพาะวัตถุที่ต้องการ

Zoom Window : คำสั่งนี้คล้ายกับคำสั่ง Zoom แต่จะเป็นการปรับค่าพื้นที่ในการมองเห็นของ viewport เท่านั้น ซึ่งถ้าใช้คำสั่งบน Planar viewport เช่น Top Right หรือ Front จะถูกแทนที่ด้วย Region Zoom หรือถ้าใช้คำสั่งบน Perspective viewport จะถูกแทนที่ด้วย Field of View

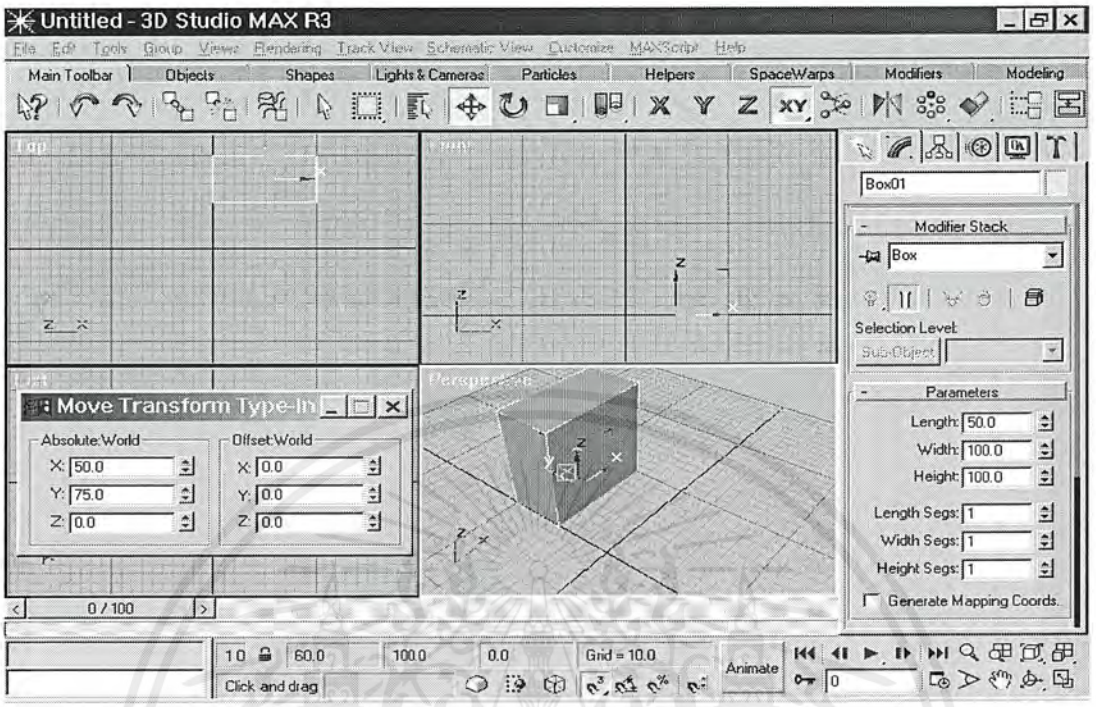
รูปที่ 2-6 Zoom Window

Pan : คำสั่งให้เลื่อน viewport โดยไม่มีผลต่อการย่อ-ขยายภาพ

Arc Rotate : คำสั่งนี้ใช้หมุน viewport ไปรอบๆ โดยใช้จุดศูนย์กลางของ viewport เป็นแกนในการหมุน การใช้งานเมื่อคลิกเลือกคำสั่งนี้ จะปรากฏไอคอนสีเขียวบน viewport คลิกเมาส์ค้างไว้แล้วลากเมาส์ไปรอบๆ ทำให้ viewport เปลี่ยนไปตามเมาส์ และมี Arc Rotate Selected เป็น flyout สำหรับเลือกเฉพาะวัตถุที่ต้องการเป็นแกนหมุน และ Arc Rotate SubObject เป็น flyout สำหรับเลือก material ที่กำหนดลงบนวัตถุหรือวัตถุย่อย (Sub-Object) เป็นแกนหมุน

2.3 ตัวอย่างการใช้งานโปรแกรม 3D Studio MAX

2.3.1 ตัวอย่างการสร้างและปรับคุณสมบัติวัตถุในระดับพื้นฐาน



รูปที่ 2-7 ตัวอย่างการสร้างและปรับคุณสมบัติวัตถุในระดับพื้นฐาน

- Active ที่ Top viewport เลือก Create Command>ปุ่ม Geometry>Object Type rollout>ปุ่ม Box สร้าง box01
- Active ที่ Perspective viewport เลือก Modify Command>Parameters rollout>ใส่ค่า/ปรับค่าใน Length Width และ Height spinner (50, 100, 100)
- ปรับโคออร์ดิเนต โดยกดปุ่ม Select and Move (Main Toolbar Tab)>Tool (เมนู pull-down)>Move Transform Type-In dialogue>ใส่ค่า/ปรับค่าใน X Y และ Z spinner (50, 75, 0)

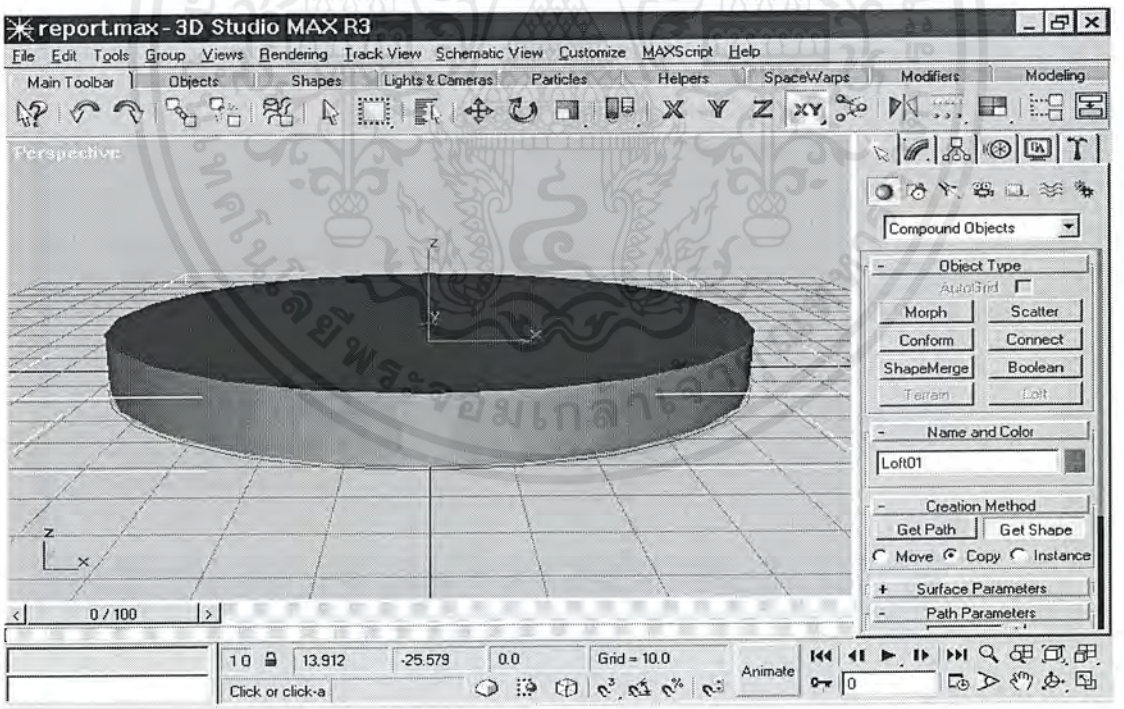
2.3.2 ตัวอย่างการสร้างวัตถุระดับสูงแบบ Loft

เริ่มสร้างวัตถุตัวอย่าง (วงกลม) โดยเลือก Create Command>ปุ่ม Shapes>Object Type rollout>ปุ่ม Circle จากนั้นปรับค่ารัศมี โดยคลิกขยาย Parameters rollout และใส่/ปรับค่ารัศมีใน Radius spinner (50) และสร้างเส้นตรง (ให้ตั้งฉากกับวงกลมที่สร้างไว้) คลิกเมาส์ active Top viewport และเลือก Create Command>ปุ่ม Shapes>

- ขยาย Creation Method rollout>Initial Type>Corner option
- ขยาย Keyboard Entry rollout>ใส่ค่า 0 ใน X spinner Y spinner และ Z spinner>คลิกปุ่ม Add Point และใส่ค่า 10 (ความสูงเส้นตรงที่ต้องการ)ใน z spinner คลิกปุ่ม Add Point

ได้เส้นตรงตั้งฉากกับวงกลม ซึ่งมีโคออร์ดิเนต (0, 0, 0) และเราจะนำวงกลมกับเส้นตรงมาสร้างเป็นวัตถุทรงกระบอกได้ โดยการสร้างวัตถุ loft ซึ่งมีขั้นตอนดังนี้

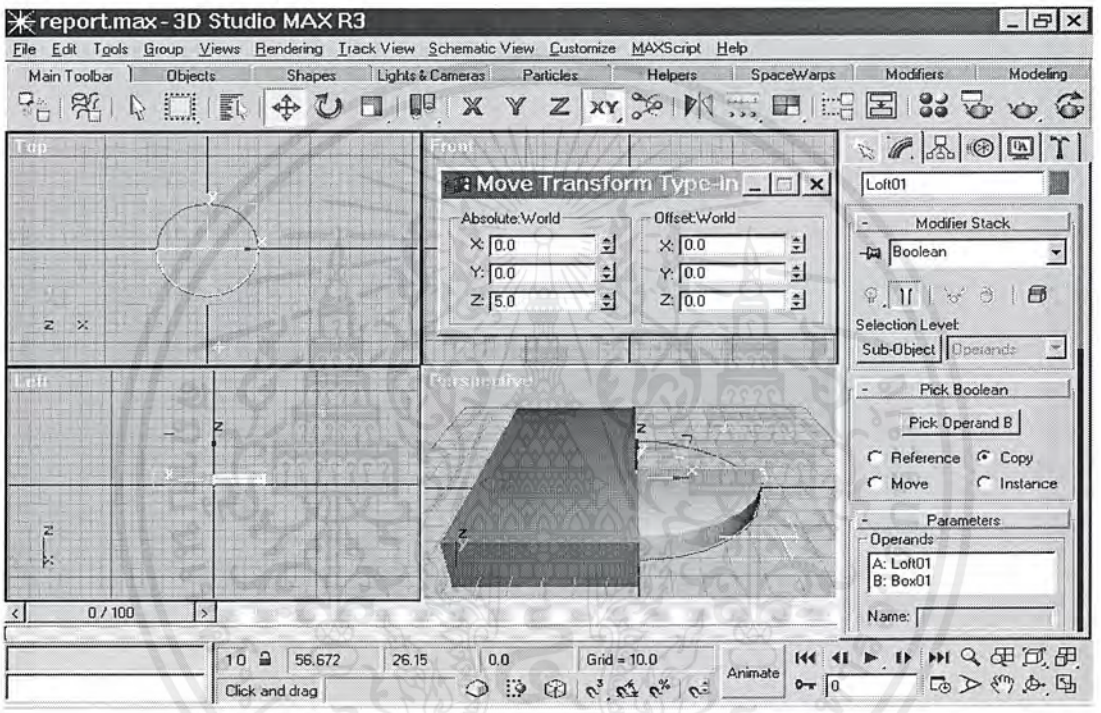
- Main Toolbar Tab>Select and Move>เลือกเส้นตรงเพื่อสร้างวัตถุ loft (ได้ทรงกระบอกตัน ถ้าเลือกวงกลม จะได้ทรงกระบอกกลวง)
- Create command>ปุ่ม Geometry>Compound Objects ใน drop-down list>Object Type rollout>ปุ่ม Loft
  - Creation Method rollout>ปุ่ม Get Shape>เลือก Copy option กรณีสร้างวัตถุ loft เสร็จจะคัดลอกวงกลมเก็บ แต่วงกลมจะมีการเปลี่ยนขนาดอย่างไรไม่ทำให้วัตถุ loft เปลี่ยนขนาดตาม



รูปที่ 2-8 ตัวอย่างการสร้างวัตถุระดับสูงแบบ Loft

2.3.3 ตัวอย่างการสร้างวัตถุระดับสูงแบบ Boolean

- สร้าง box ซึ่งมีขนาด 60 ×120 ×20 และย้ายไปที่โคออร์ดิเนต (0, 30, -5)
- คลิกปุ่ม Select and Move (ใน Main Toolbar Tab)>คลิกเมาส์เลือกวัตถุ loft>Create Command>Object Type rollout>คลิกปุ่ม Boolean
  - Operation>Subtraction (A-B) option ซึ่งวัตถุ A คือ วัตถุ loft
  - Pick Boolean rollout>คลิกปุ่ม Pick Operand B และคลิกเมาส์เลือก box ที่สร้างไว้ คือ ให้วัตถุ B เป็น box

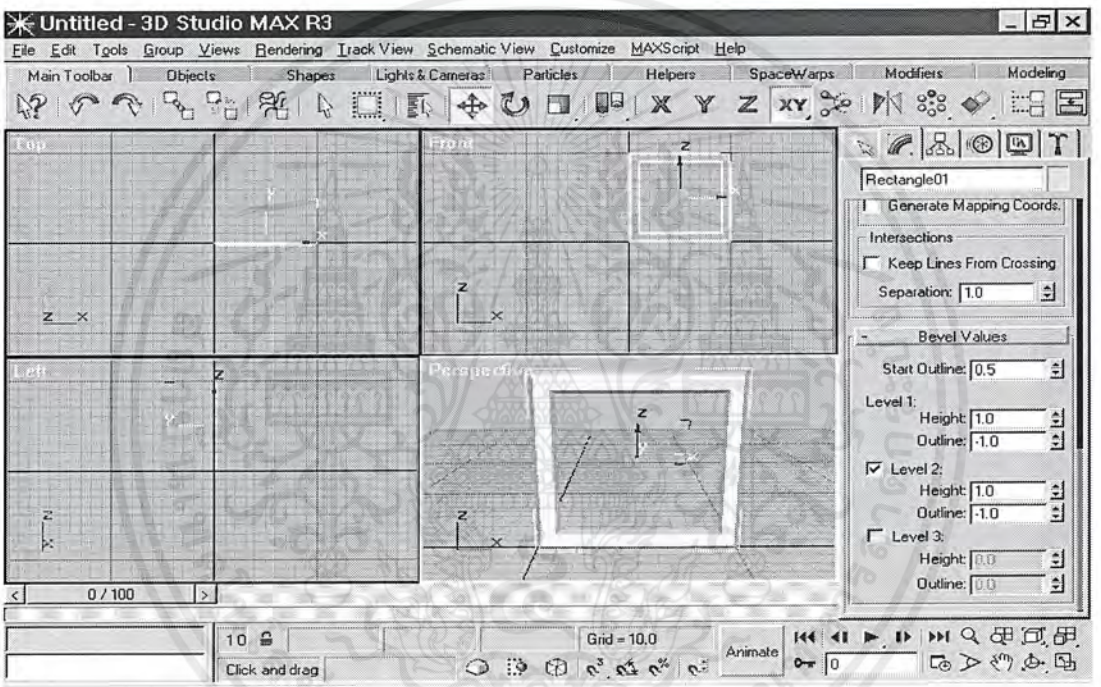


รูปที่ 2-9 ตัวอย่างการสร้างวัตถุระดับสูงแบบ Boolean

สังเกตเห็นว่าวัตถุ loft ถูกตัดไป ½ ส่วน หรือเป็นวัตถุ boolean ทรงกระบอกที่สร้างขึ้นเพียง ½ ส่วน

2.3.4 ตัวอย่างการสร้างวัตถุระดับสูงแบบ Edit Spline และ Bevel

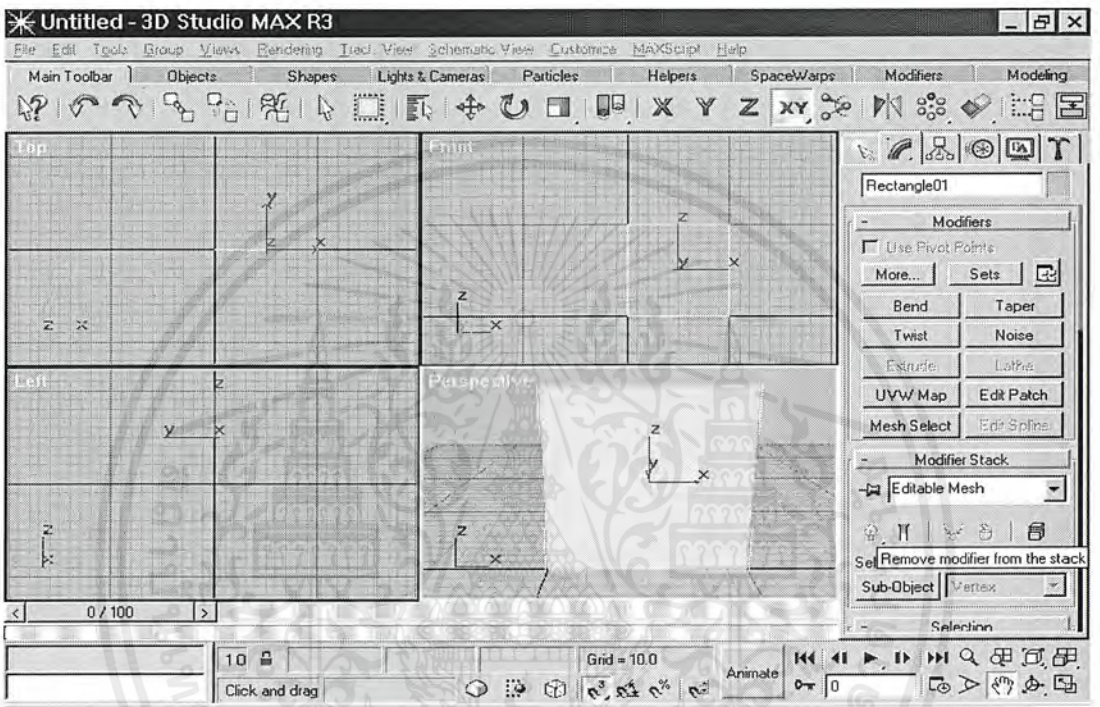
- สร้างรูปสี่เหลี่ยม 2 รูปซ้อนกันให้เป็นวัตถุชิ้นเดียวกัน โดย active ที่ Top viewport เลือก Create Command>ปุ่ม Shapes คลิกปลด check box ด้านขวาของปุ่ม Start New Shape ออก คลิกปุ่ม Rantangle สร้างรูปสี่เหลี่ยมซ้อนกัน 2 รูปเป็น Rantangle01
- แปลง Rantangle01 เป็น Editable Mesh โดย Modify Command>Modifier Stack rollout>ปุ่ม Edit Spline สังเกตว่าค่าใน drop-down list เป็น Edit Spline
- สร้าง Rantangle01 เป็น Bevel โดย Modify Command>Modifier rollout>ปุ่ม More คลิกเลือก Bevel ขยาย Bevel Values rollout ใส่ค่า/ปรับค่าใน Start Outline spinner (0.5) Level1 Height spinner (1.0) Level1 Outline spinner (-1.0) Level2 Height spinner (1.0) Level2 Outline spinner (-1.0)



รูปที่ 2-10 ตัวอย่างการสร้างวัตถุระดับสูงแบบ Edit Spline และ Bevel

2.3.5 ตัวอย่างการสร้างวัตถุระดับสูงแบบ Editable Mesh

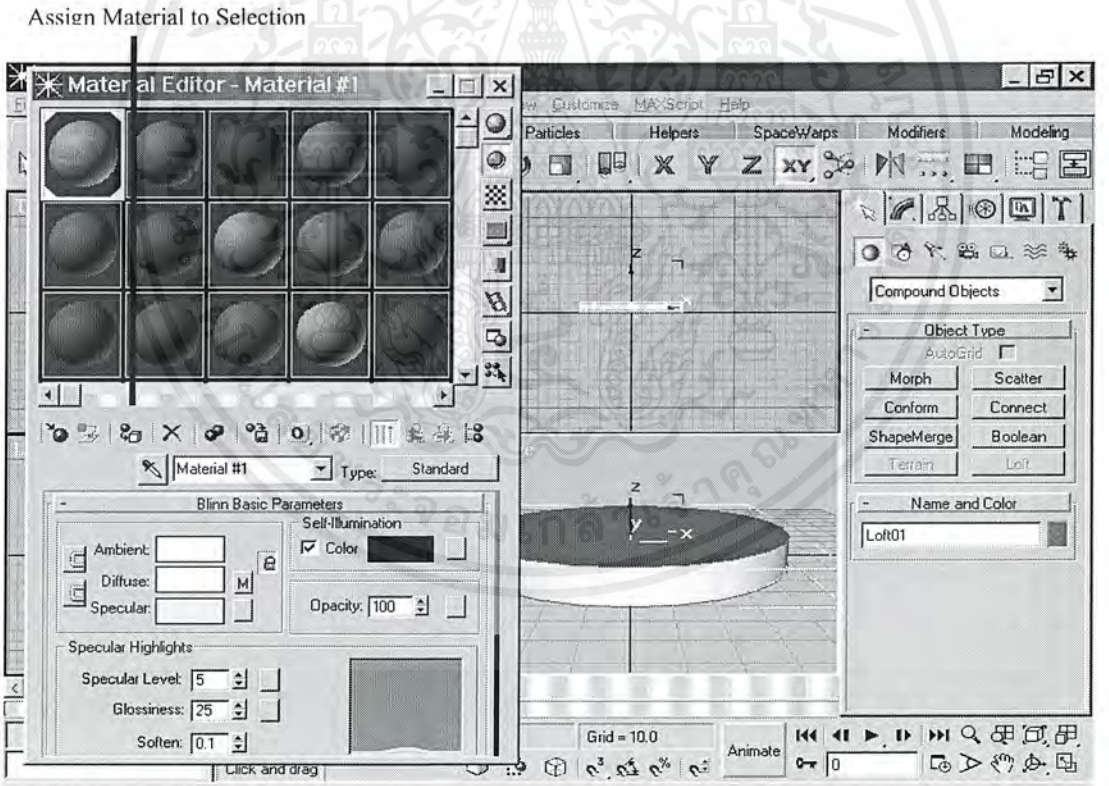
- active ที่ Top viewport> Create Tab>ปุ่ม Shapes>Object Type rollout>ปุ่ม Rantangle และสร้างรูปสี่เหลี่ยม หรือ Rantangle01
- Modify Tab>Modifiers rollout>ปุ่ม Edit Stack แปลงเป็น Editable Mesh หมายถึง แปลงวัตถุจากรูปสี่เหลี่ยมที่สร้างไว้เป็นแผ่นสี่เหลี่ยม หรือ editable mesh ซึ่งเป็นกฎของ 3D Studio Max ที่สามารถมองเห็นพื้นผิวได้เพียงด้านเดียว แสดงผลดังรูป



รูปที่ 2-11ตัวอย่างการสร้างวัตถุระดับสูงแบบ Editable Mesh

2.3.6 ตัวอย่างการกำหนด Material ให้กับวัตถุ

- Main Toolbar Tab>ปุ่ม Select and Move>เลือกวัตถุ loft
- เลือก Tool (เมนู pull-down) >Material Editor>เลือก Material จากช่องแสดง Material สังเกตว่า Material ที่เลือกมีกรอบสีขาวล้อมรอบ
- Binn Basic Parameters rollout>คลิกเมาส์ที่ช่อง Ambient Color แสดง Color Selector dialogue เพื่อปรับสีตามต้องการ แต่ในที่นี้จะนำไฟล์รูปแบบอื่น (bitmap) จากภายนอก 3D Studio Max มาเป็น Material จึ่งต้องปรับ Ambient Color ให้เป็นสีขาว และปรับสีที่ช่อง Diffuse Color ให้เป็นสีขาวด้วย
- คลิกเมาส์ที่ปุ่มด้านขวา Diffuse Color เพื่อ browse ไฟล์ Material แสดง Material/Map Browser dialogue คลิกเลือก bitmap เพื่อเลือกรูปแบบไฟล์ที่ต้องการ เนื่องจากโครงงานนี้ต้องนำโมเดลไปพัฒนาเกมโดยใช้ DirectX บน Windows ซึ่งจะรองรับไฟล์รูปภาพประเภท bitmap มีขนาดเป็น $2^n \times 2^n$ (เป็นสี่เหลี่ยมจัตุรัส) และคลิกปุ่ม OK เพื่อแสดง Select Bitmap Image File dialogue คลิกเมาส์เลือกไฟล์ที่ต้องการ และคลิกปุ่ม Open และจะกลับมาที่ Material/Map Browser dialogue และคลิกปุ่ม Assign Material to Selection



รูปที่ 2-12 ตัวอย่างการกำหนด Material ให้กับวัตถุ

2.3.6.1 ตัวอย่างการใช้เทคนิคบน PhotoShop สร้าง Material เพื่อเพิ่มความสมจริงให้กับวัตถุ

รูปที่ 2-13 ตัวอย่างการใช้เทคนิคบน PhotoShop สร้าง Material เพื่อเพิ่มความสมจริงให้กับวัตถุ

สำหรับพื้นผิวัตถุที่เป็นกระจก ในการแต่ง Bitmap บน PhotoShop จะนำ Linear Gradient Tool มาใช้ในการแสดงสีในความเข้มไม่เท่ากัน เพื่อให้หน้าต่างที่เป็นกระจกมีเงา

- ปรับโหมดของ Bitmap ให้เป็น RGB Color ก่อน
- ใช้ Rectangular Maquee Tool เลือกส่วนที่ต้องการทำ Gradient
- ใช้ Linear Gradient Tool คลิกเมาส์ค้างและลากบนส่วนที่ต้องการ โดยจุดเริ่มต้นลากเมาส์เป็นสีในช่อง foreground และจุดสุดท้ายเป็นสีในช่อง Background และไล่ความเข้มสีในปริมาณที่เหมาะสมกับระยะที่ลากเมาส์ค้างตั้งแต่จุดแรกถึงจุดสุดท้าย
- ปรับโหมดของ Bitmap ให้เป็น Indexed Color

2.3.7 ตัวอย่างการ Map Material

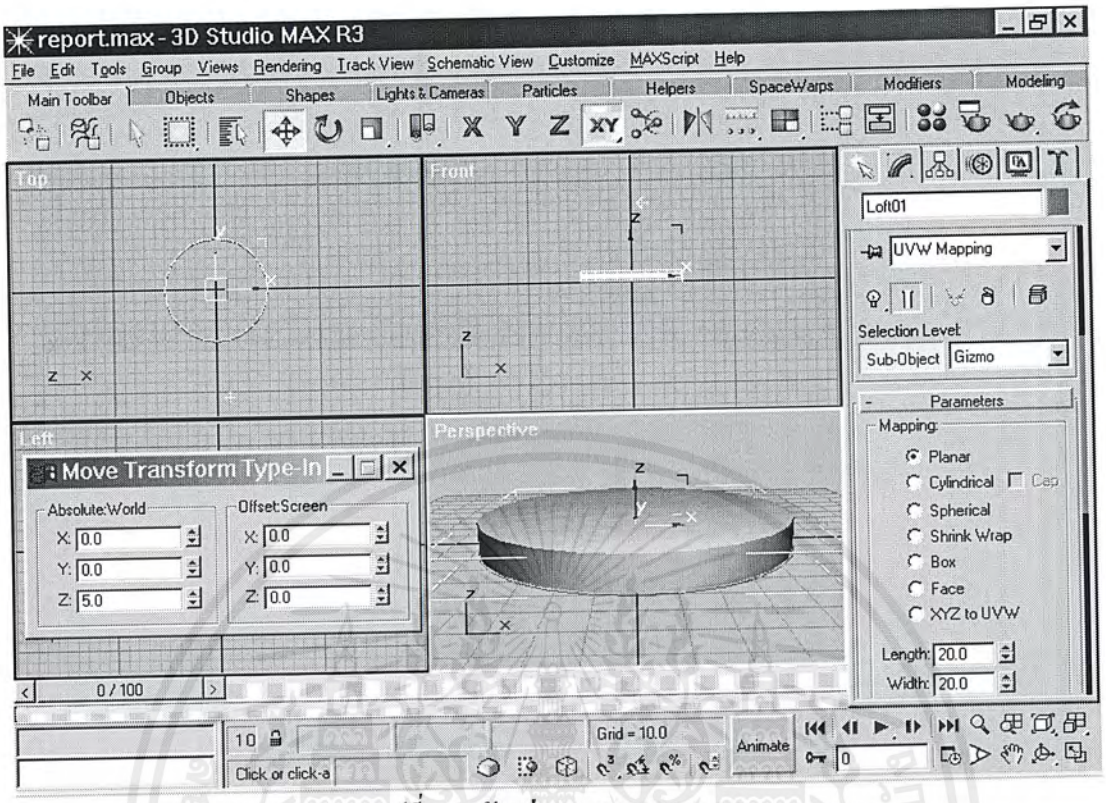
ขั้นตอนนี้สำคัญมาก คือถ้าทำเพียงการกำหนด material ให้กับวัตถุ สามารถเรนเดอร์โดยที่เห็นลาย material ได้เฉพาะใน 3D Studio Max เท่านั้น และลาย material บนวัตถุจะไม่เป็นระเบียบตามที่ต้องการ

- เริ่มต้นเลือก Modify Command>Modifier rollout>ปุ่ม UVW Map แสดง Parameters rollout
- เลือก Planar option (ให้ map วัตถุด้วย material ที่เลือกไว้ เพียง 2 ด้าน คือ หน้า-หลัง/ซ้าย-ขวา/บน-ล่าง) และใส่ค่าใน Length spinner และ Width spinner ซึ่งเป็นความลึก กว้างของ material ที่จัดเรียงลงบนวัตถุ และใน Alignment ให้เลือก X/Y/Z option (เป็นการเลือกให้ material วางตัวตามแกน X/Y/Z)

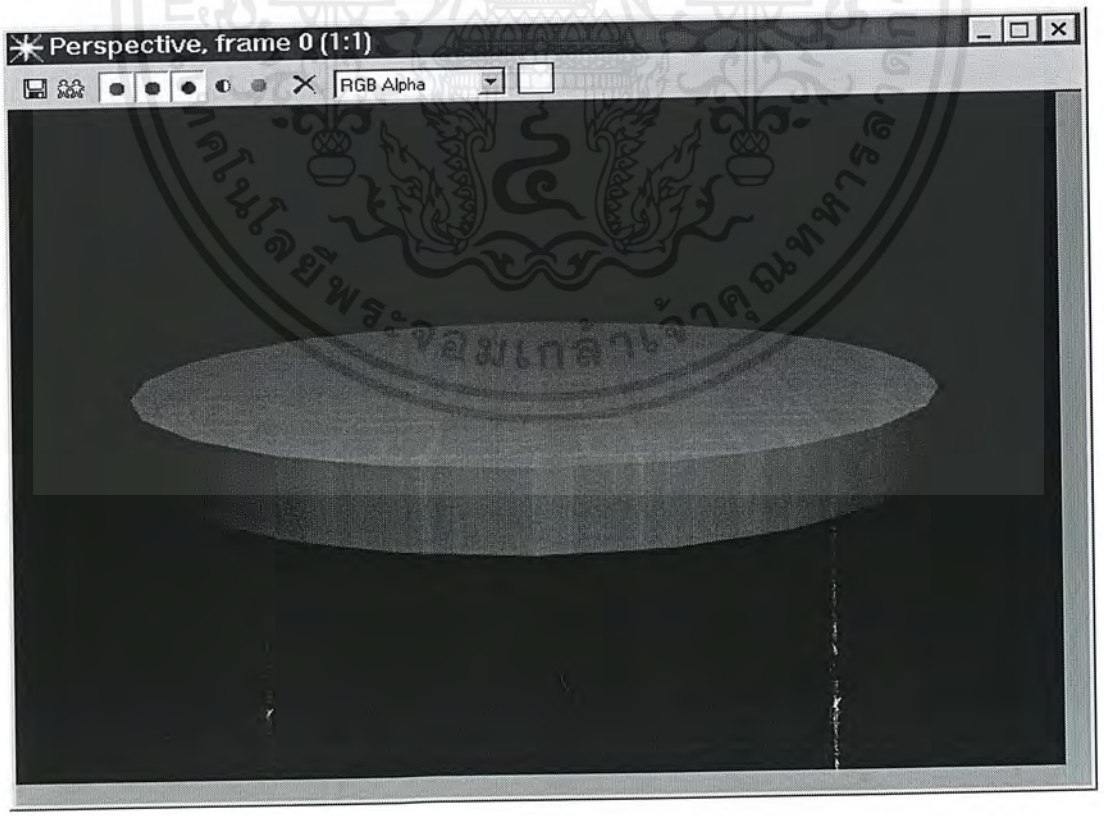
การจัดเรียง material บนวัตถุ การเพิ่มขนาด material และการหมุน material สามารถทำได้ โดยมอง material เป็นวัตถุย่อยๆ (Sub-Object) ซึ่งมีขั้นตอนดังนี้

- Modify Command>Modifier Stack rollout>เลือก UVW Mapping และคลิกปุ่ม Sub-Object

- การเคลื่อน การหมุน การปรับขนาด material (วัตถุย่อย) ใช้หลักการเช่นเดียวกับวัตถุ

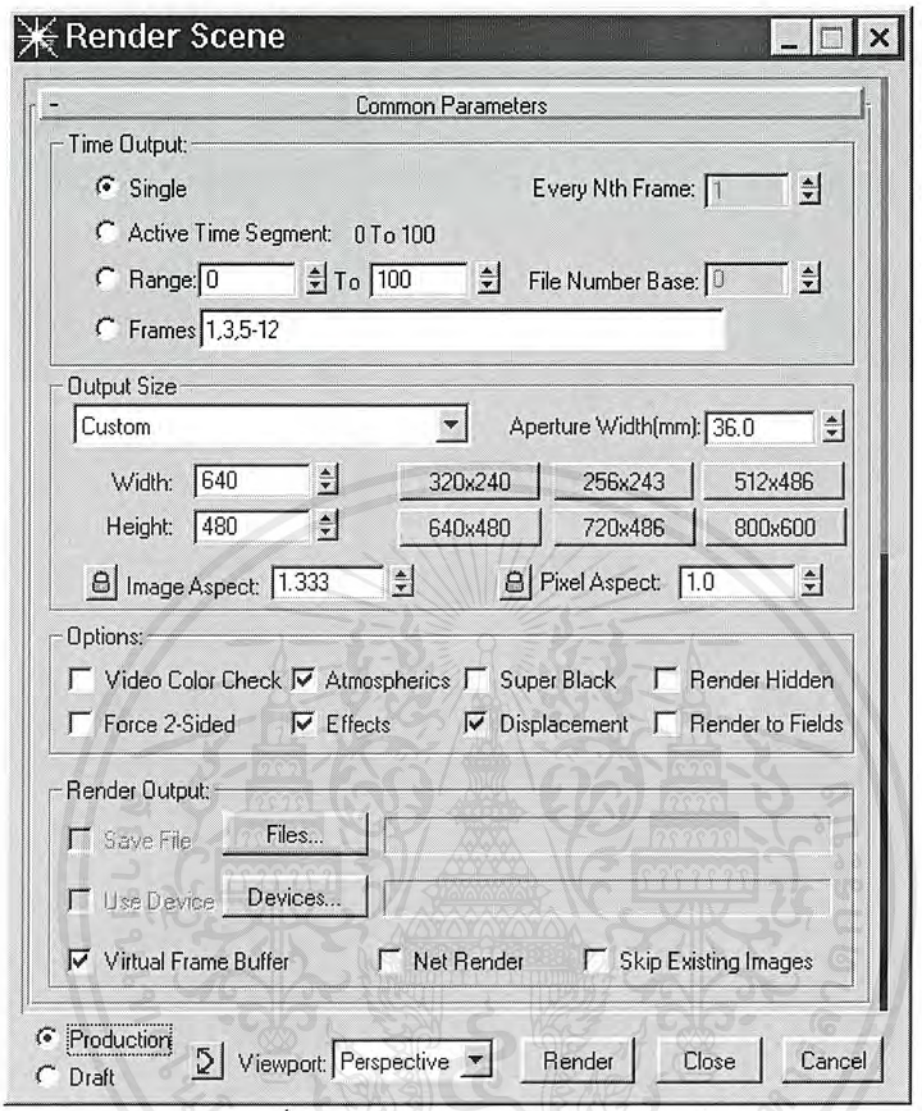


รูปที่ 2-14 ตัวอย่างการ Map Material



รูปที่ 2-15 ผลการเรนเดอร์ซึ่งได้จากการ map material

2.3.8 การเรนเดอร์วัตถุ



รูปที่ 2-16 การกำหนดคุณสมบัติในการเรนเดอร์วัตถุ

การ map ลาย material ลงบนวัตถุ จำเป็นต้องทำการเรนเดอร์ เพื่อแสดงผล ซึ่งมี 3 วิธี คือ

- เลือก Render (เมนู pull-down) >Render แสดง Render Scene dialogue
  - Time Output rollout>single option
  - Output Size rollout>คลิกเลือกปุ่มขนาดหน้าจอที่ต้องการให้เรนเดอร์ (ในที่นี้เลือก 640 ×480) ซึ่งหมายถึง ค่าใน Length spinner และ Height spinner ตามลำดับ
  - Render Output>คลิกปุ่ม Files.. ถ้าต้องการ capture ผลจากการเรนเดอร์เก็บเป็นไฟล์รูปภาพ
  - Viewport spinner หมายถึง เลือกเรนเดอร์จาก viewport ใด
  - ปุ่ม Render
- เลือก viewport ที่ต้องการเรนเดอร์>ปุ่ม Quick Render (ใน Main Toolbar Tab)
- กดปุ่ม Shift และ Q พร้อมกัน

2.4 ปัญหาในการสร้างโมเดล

- การจัดแสงและกล้องใน 3D Studio Max จะไม่แสดงผล เมื่อนำมาโปรแกรมเกมด้วย DirectX
- การสร้างเกมในโครงงานนี้ประกอบด้วยวัตถุจำนวนมาก เพราะต้องคำนึงถึง concept หลักของการสร้างโมเดลที่ต้องการความเหมือนจริงให้ได้มากที่สุด ทำให้การรันเกม (การเรนเดอร์วัตถุในเกม) ช้าลงไปมาก
- การเรนเดอร์โมเดลในขณะรันเกม หรือ real-time ด้วย DirectX รองรับการแสดงผลในรูปแบบ file.x เท่านั้น ดังนั้นไม่สามารถเรนเดอร์ file.max ได้โดยตรง โดยต้องเอ็กซ์พอร์ตเป็น file.3ds และใช้ไฟล์ conv3ds.exe แปลง file.3ds เป็น file.x ซึ่งผลลัพธ์ที่แสดงใน 3D Studio Max บางครั้งไม่เป็นไปในในลักษณะการแสดงผล file.x แบบ real-time และบางครั้งไม่สวยงามอย่างที่ได้เรนเดอร์ใน 3D Studio Max และถ้าไฟล์หนึ่งไฟล์มีวัตถุมากเกินไปเมื่อแปลงเป็น file.x จะแสดงผลไม่ถูกต้อง ดังนั้นต้องแบ่งวัตถุออกเป็นหลายไฟล์ โดยแต่ละไฟล์เก็บวัตถุในปริมาณที่เรนเดอร์แบบ real-time แล้วไม่เกิดความผิดเพี้ยน

บทที่ 3

DirectDraw

3.1 รู้จักกับ DirectDraw

DirectDraw เป็นคอมโพเนนท์ของ DirectX ที่ทำให้ผู้เขียนโปรแกรมสามารถจัดการหน่วยความจำแสดงผล และฮาร์ดแวร์ blitter (ส่วนประกอบฮาร์ดแวร์ที่อยู่ภายในการ์ดแสดงผล ทำหน้าที่ในการ blit หรือการถ่ายโอนบล็อกข้อมูล) ได้โดยตรง และสนับสนุนการทำ flipping โดยจะกล่าวรายละเอียดต่อไปในบทนี้ นอกจากนี้ DirectDraw ยังคงสภาพการทำงานของแอปพลิเคชั่นบนระบบวินโดวส์ให้เข้ากันได้กับไดรเวอร์ของอุปกรณ์ต่างๆ ด้วย

DirectDraw เป็นอินเทอร์เฟซที่ทำให้สามารถติดต่อกับอุปกรณ์แสดงผลได้โดยตรง ในขณะเดียวกันยังรักษาการทำงานให้เข้ากับอินเทอร์เฟซหลัก (GDI) ซึ่งใช้ในการติดต่ออุปกรณ์กราฟิกบนระบบวินโดวส์ แต่ GDI ไม่ใช่ API สำหับกราฟิกในระดับสูง DirectDraw จะทำให้เกมและซอฟต์แวร์ในระบบย่อยของวินโดวส์ เช่น แพ็กเกจกราฟิก 3 มิติทั้งหลาย เครื่องแปลงสัญญาณวีดีโอดิจิตอล มีอิสระไม่ขึ้นกับอุปกรณ์บนเครื่องคอมพิวเตอร์

DirectDraw ทำงานท่ามกลางความหลากหลายของฮาร์ดแวร์แสดงผล ที่เป็นได้ตั้งแต่จอภาพ SVGA แบบธรรมดาจนถึงฮาร์ดแวร์ระดับสูงที่สนับสนุนการทำ clipping การทำ stretching (การ blit รูปไปยังปลายทางซึ่งมีมิติต่างออกไป) และรูปแบบสีที่ไม่ใช่ RGB อินเทอร์เฟซถูกออกแบบมาทำให้แอปพลิเคชันสามารถประเมินความสามารถของฮาร์ดแวร์แล้วจึงนำคุณลักษณะที่ได้รับการสนับสนุนจากฮาร์ดแวร์ตัวเร่งมาใช้ ส่วนคุณลักษณะที่ไม่ฮาร์ดแวร์ไม่สามารถทำได้จะถูกจำลองขึ้นมาโดย DirectX

DirectDraw ทำให้มีอิสระในการเข้าถึงหน่วยความจำแสดงผลโดยวิธีที่ไม่ขึ้นกับอุปกรณ์ ที่สำคัญเราใช้ DirectDraw ในการจัดการหน่วยความจำแสดงผล โดยที่แอปพลิเคชันต้องรู้จักเพียงข้อจำกัดของอุปกรณ์พื้นฐานบางชนิดที่เป็นมาตรฐานในการใช้งานฮาร์ดแวร์ เช่น รูปแบบสี RGB และ YUV และระยะห่างระหว่างแอดเดรสเริ่มต้นที่แสดงถึงเส้นบนจอภาพแต่ละเส้น โดยที่ผู้เขียนโปรแกรมไม่จำเป็นต้องเรียกใช้โพรซีเดอร์พิเศษเพื่อจัดการกับ blitter หรือรีจิสเตอร์พาเลตสี เมื่อใช้ DirectDraw จะทำให้สามารถใช้หน่วยความจำแสดงผลได้อย่างง่าย รวมทั้งยังใช้ประโยชน์จากการทำ blitting และความสามารถในการแปลงสีให้เข้ากับฮาร์ดแวร์แสดงผล โดยไม่ขึ้นกับฮาร์ดแวร์ส่วนใดส่วนหนึ่งโดยเฉพาะ

DirectDraw ทำให้เกิดเกมกราฟิกชั้นนำบนเครื่องคอมพิวเตอร์ระบบวินโดวส์ 95 และวินโดวส์รุ่นหลัง รวมทั้งวินโดวส์ NT เวอร์ชัน 4.0 หรือวินโดวส์ 2000

3.2 เหตุผลในการนำ DirectDraw มาใช้ในการพัฒนาเกม

DirectDraw เป็นคอมโพเนนท์ที่ทำให้การพัฒนาโปรแกรมกราฟิกบนวินโดวส์ทำได้อย่างมีประสิทธิภาพ

- มีอินเทอร์เฟซสำหรับใช้งานฮาร์ดแวร์แสดงผลได้โดยตรง ส่งผลให้แอปพลิเคชั่นแสดงผลได้ดีที่สุด เมื่อโปรแกรมทำงานในระดับ Hardware Abstraction Layer หรือ HAL ของ DirectDraw
- DirectDraw สามารถประเมินความสามารถของฮาร์ดแวร์และใช้คุณลักษณะพิเศษต่างๆ ของฮาร์ดแวร์เมื่อทำได้ ตัวอย่างเช่น หากการ์ดจอบนเครื่องสนับสนุนฮาร์ดแวร์ที่ทำ blitting DirectDraw ก็จะมอบหน้าที่การ blit ให้กับการ์ดจอ ซึ่งจะทำให้การแสดงผลมีประสิทธิภาพดียิ่งขึ้น นอกจากนี้ DirectDraw ยังมีระดับการทำงาน Hardware Emulation Layer ซึ่งจะสนับสนุนคุณลักษณะต่างๆ เมื่อฮาร์ดแวร์ไม่สามารถทำได้
- DirectDraw ทำงานภายใต้ระบบวินโดวส์ จึงได้รับประโยชน์จากการอ้างอิงแอดแดรสของหน่วยความจำขนาด 32 บิต และโมเดลของหน่วยความจำแบบแฟลตซึ่งระบบปฏิบัติการเตรียมให้ DirectDraw จะแสดงหน่วยความจำแสดงผลและหน่วยความจำของระบบเป็นแหล่งเก็บข้อมูลรูปแบบบล็อกขนาดใหญ่ ไม่ใช่เซ็กเมนต์ขนาดเล็กๆ คนที่เคยใช้การอ้างอิงแอดแดรสแบบเซ็กเมนต์หรือแบบอ็อฟเซ็ต จะรู้คุณค่าของโมเดลหน่วยความจำแบบแฟลตนี้ได้อย่างรวดเร็ว
- DirectDraw ทำให้การสลับหน้าจอหรือที่เรียกว่า page flipping ที่มีบัฟเฟอร์สำรองหรือ back buffer หลายอันในแอปพลิเคชั่นโหมดเต็มจอสามารถทำได้โดยง่าย ดูรายละเอียดเพิ่มเติมได้จากหัวข้อ Page Flipping และ Back buffering
- สนับสนุนการทำ clipping ในแอปพลิเคชั่นทั้งในโหมดวินโดว์และโหมดเต็มจอ
- สนับสนุน Z-buffer สามมิติ
- สนับสนุนโอเวอร์เลย์ของฮาร์ดแวร์ตัวช่วยโดยการจัดลำดับระดับความลึกหรือ z-ordering
- สามารถเข้าถึงฮาร์ดแวร์ที่ทำหน้าที่ image-stretching
- เข้าถึงพื้นที่หน่วยความจำของอุปกรณ์แสดงผลในระดับมาตรฐานและระดับที่สูงขึ้นได้ในเวลาเดียวกัน
- คุณลักษณะอื่นๆ รวมถึงพาเลตแบบไดนามิคและแบบคัสตอม การเข้าถึงฮาร์ดแวร์เฉพาะอย่าง และการสับเปลี่ยนโหมดความละเอียดหรือรีโซลูชั่น

เมื่อรวมคุณลักษณะเหล่านี้ไว้ด้วยกัน จะเห็นได้ว่าเราสามารถสร้างแอปพลิเคชั่นที่มีประสิทธิภาพดียิ่งกว่าแอปพลิเคชั่นมาตรฐานที่มีรากฐานมาจาก GDI บนระบบวินโดวส์ทั่วไป หรือแม้แต่แอปพลิเคชั่นบนระบบดอสก็ตาม

3.3 ความเข้าใจพื้นฐานทางด้านกราฟิก

เนื้อหาในส่วนนี้จะอธิบายความรู้พื้นฐานที่จำเป็นต้องใช้ในการเขียนโปรแกรมด้านกราฟิกโดยใช้ DirectDraw ซึ่งมีดังต่อไปนี้

- Device-Independent Bitmaps
- Drawing Surfaces

- Blitting
- Page Flipping และ Back Buffering
- รู้จักกับ Rectangles

3.3.1 Device-Independent Bitmaps

ระบบปฏิบัติการวินโดวส์ รวมถึง DirectX ใช้ Device-Independent Bitmap หรือ DIB เป็นรูปแบบไฟล์กราฟิกของตนเอง DIB เป็นไฟล์ซึ่งบรรจุข้อมูลที่อธิบาย ขนาดของรูปภาพ จำนวนสีที่ใช้ ค่าที่แสดงถึงสีเหล่านั้น และข้อมูลของแต่ละ pixel นอกจากนี้ DIB ยังประกอบด้วยค่าตัวแปรที่ใช้ไม่บ่อยนัก เช่นข้อมูลเกี่ยวกับการบีบอัดไฟล์ สีที่สำคัญบางสี (หากไม่ได้ใช้ทั้งหมดทุกสี) และขนาดจริงของรูปภาพ (ในกรณีที่ต้องการพิมพ์ภาพนั้น) โดยปกติไฟล์ DIB จะอยู่ในรูป .bmp หรือบางครั้งอาจใช้ .dib

เนื่องจาก DIB ใช้กันอย่างแพร่หลายในการเขียนโปรแกรมบนวินโดวส์ Platform SDK จึงบรรจุฟังก์ชันมากมายไว้เรียบร้อยแล้ว เราสามารถนำมาใช้กับ DirectX ได้เลย ตัวอย่างเช่น การเขียนเกมในโครงงานนี้ก็ใช้ไฟล์รูป .bmp มาใช้ในการตกแต่งพื้นผิวของวัตถุ 3 มิติภายในเกมเช่นกัน

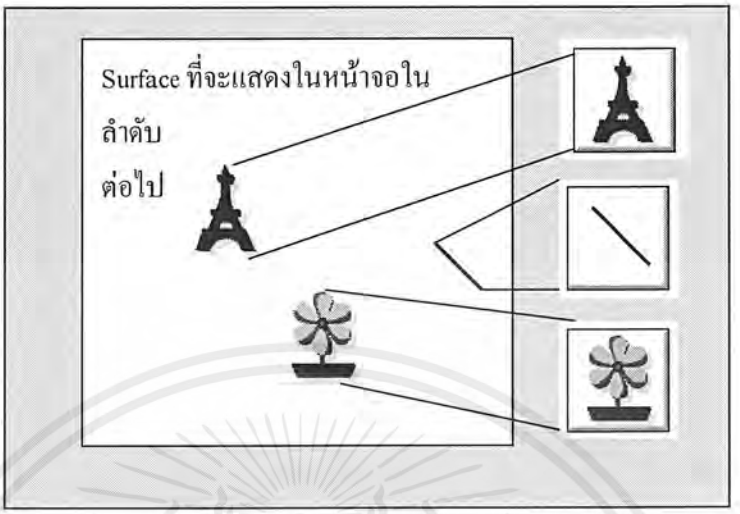
3.3.2 Drawing Surface

Drawing Surface รับข้อมูลวีดีโอและถูกแสดงออกทางหน้าจอเป็นรูปภาพในขั้นสุดท้าย ในโปรแกรมวินโดวส์ส่วนใหญ่มักจะเข้าถึง Drawing Surface โดยใช้ฟังก์ชันของ Win32 เช่น *GetDC* ซึ่งย่อมาจาก Get Device Context หลังจากนั้นจึงจะสามารถวาดภาพลงบนหน้าจอได้ อย่างไรก็ตาม ส่วนต่างๆ ภายในระบบสามารถใช้ฟังก์ชันกราฟิกของ Win32 โดยผ่านทางอินเทอร์เฟซของอุปกรณ์กราฟิกที่เรียกว่า GDI หรือ Graphics Device Interface ซึ่งเป็นคอมโพเนนท์หนึ่งของระบบที่ทำให้แอปพลิเคชั่นต่างๆ บนวินโดวส์มาตรฐานสามารถแสดงผลทางหน้าจอได้

ข้อเสียของ GDI ก็คือไม่ได้ออกแบบมาสำหรับซอฟต์แวร์มัลติมีเดียที่มีประสิทธิภาพสูง แต่ออกแบบมาสำหรับแอปพลิเคชั่นทางธุรกิจ เช่น เวิร์ดโปรเซสเซอร์ และแอปพลิเคชั่นสเปรดชีทต่างๆ GDI ยังสนับสนุนเพียงการเข้าถึงวีดีโอบัฟเฟอร์ภายในหน่วยความจำของระบบ ไม่ใช่หน่วยความจำวีดีโอ และยังไม่สามารถดึงลักษณะเด่นที่มีในการ์ดวีดีโอบางชนิดมาใช้ให้เกิดประโยชน์ได้ โดยสรุป GDI เหมาะสำหรับแอปพลิเคชั่นทางธุรกิจมากที่สุด แต่อาจมีการแสดงผลไม่เร็วพอสำหรับซอฟต์แวร์มัลติมีเดียหรือเกมต่างๆ

ในขณะเดียวกัน DirectDraw นั้นจะทำให้เราสามารถวาดพื้นผิวหรือ Surface ซึ่งแสดงถึงข้อมูลภายในหน่วยความจำวีดีโอ จึงหมายความว่าเมื่อเราสามารถใช้ DirectDraw เราสามารถเขียนข้อมูลลงบนหน่วยความจำบนการ์ดวีดีโอได้โดยตรง ส่งผลทำให้กระบวนการทางกราฟิกมีความเร็วสูงขึ้นมาก โดย Surface เหล่านี้จะถูกแสดงในรูปของชุดของบล็อกข้อมูลภายในหน่วยความจำ ซึ่งการเข้าถึงแอดเดรสของข้อมูลเหล่านี้สามารถทำได้โดยง่าย

Surface เป็นส่วนประกอบที่สำคัญมากของ DirectDraw มีหน้าที่ในการจัดการกับรูปภาพและการประมวลผลต่างๆ



รูปที่ 3-1 ตัวอย่างของ Surface

จากรูปจะเห็นได้ว่าภายใน 1 Surface นั้นจะบรรจุทั้งรูปภาพและส่วนประกอบของกราฟิก ซึ่งเราจะเตรียมไว้สำหรับนำไปแสดงในลำดับต่อไปในกรณีที่ Surface นี้เป็น Surface สำรอง

3.3.3 Blitting

คำว่า blit เป็นคำย่อจาก "bit block transfer" ซึ่งเป็นกระบวนการในการถ่ายโอนบล็อกข้อมูลจากตำแหน่งหนึ่งไปยังอีกตำแหน่งหนึ่งภายในหน่วยความจำ การ blit มักจะใช้ในการแสดงการเคลื่อนไหวช้าๆ

3.3.4 Page Flipping และ Back Buffering

การสลับภาพบนหน้าจอหรือการทำ Page Flipping ถือเป็นกุญแจสำคัญในการทำมัลติมีเดีย ภาพเคลื่อนไหว และซอฟต์แวร์เกม การทำ page flipping โดยซอฟต์แวร์เปรียบเหมือนการทำภาพเคลื่อนไหวจากแผ่นกระดาษ โดยกระดาษแต่ละหน้าจะมีรูปภาพที่ค่อยๆ เปลี่ยนแปลงไปทีละเล็กน้อย ดังนั้นเมื่อเราพลิกแผ่นกระดาษเหล่านั้นอย่างรวดเร็วรูปที่เห็นจะเหมือนมีการเคลื่อนไหวด้วย

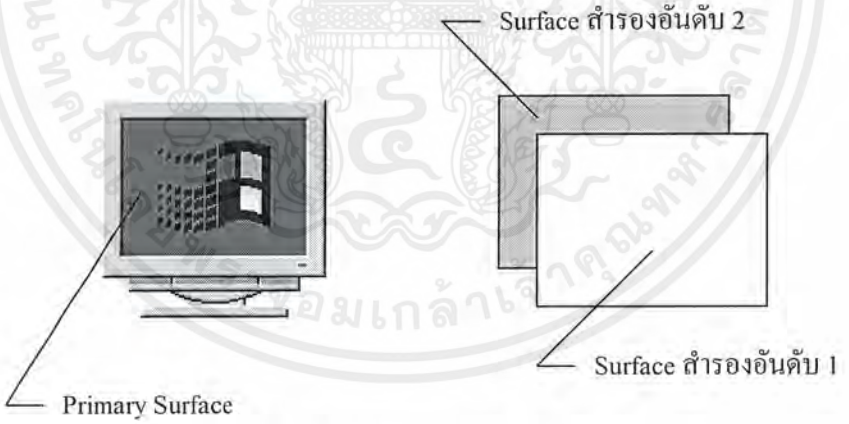
การใช้ซอฟต์แวร์ในการทำ page flipping จะคล้ายกับกระบวนการเช่นนี้มาก กล่าวคือ เริ่มต้นเราจะจัดเรียง DirectDraw surface ไว้ชุดหนึ่ง โดยออกแบบจะมีลำดับการแสดงบนหน้าจอเช่นเดียวกับชุดรูปภาพบนแผ่นกระดาษที่จะถูกพลิกเป็นลำดับถัดๆ กันไป พื้นผิวหรือ surface แรกเรียกว่า surface หลักหรือ primary

surface และ surface ลำดับต่อๆ มาเรียกว่า back buffer แอปพลิเคชั่นของเราจะเขียนหรือเปลี่ยนแปลงข้อมูลลงใน back buffer แล้วจึงทำการสลับกับ primary surface ทำให้ภาพใน back buffer เดิมแสดงออกสู่หน้าจอ ในขณะที่ระบบกำลังแสดงรูปภาพออกทางหน้าจอ ซอฟต์แวร์ก็จะทำการเขียนลงบน back buffer อีกครั้ง กระบวนการเช่นนี้จะดำเนินไปเรื่อยๆ ในขณะที่ภาพกำลังเคลื่อนไหว ส่งผลให้เราสามารถทำภาพเคลื่อนไหวได้อย่างรวดเร็วและมีประสิทธิภาพ

DirectDraw นั้นให้เราสามารถกำหนดกรรมวิธีในการทำ page flipping ได้โดยง่าย โดยสามารถกำหนดให้เป็นได้ตั้งแต่แบบพื้นฐานซึ่งใช้ 2 บัฟเฟอร์ ทำงานเกี่ยวเนื่องกัน (primary buffer กับอีก 1 back buffer) ไปจนถึงแบบที่มีจำนวน back buffer เพิ่มมากขึ้น ซึ่งจะซับซ้อนมากยิ่งขึ้นเช่นกัน

โดยในการสร้าง Surface นั้น เราจะสร้าง Surface ได้หลายแบบด้วยกัน แต่ที่ใช้กันบ่อยๆ ก็จะสร้างไว้ 2 Surface ดังนี้

1. Primary Surface (Surface หลัก) เป็น Surface ที่กำลังแสดงอยู่บนหน้าจอปัจจุบัน
2. Other Surface หรือ Back Surface (Surface สำรอง) เป็น Surface ที่เตรียมไว้สำหรับแสดงต่อไป โดยจะเลื่อนขึ้นไปอีก 1 ขั้น คือ ถ้าเป็น Surface สำรองอันดับ 1 ก็จะเลื่อนขึ้นไปเป็น Surface หลักในลำดับต่อไป ส่วน Surface สำรองอันดับ 2 ก็จะเลื่อนไปเป็น Surface สำรองอันดับ 1 และ Surface หลักก็จะเลื่อนกลับมาเป็น Surface สำรองอันดับที่ 2 ในกรณีที่เราสร้าง Surface ไว้ทั้งหมดด้วยกัน 3 Surface ดังแสดงในรูป

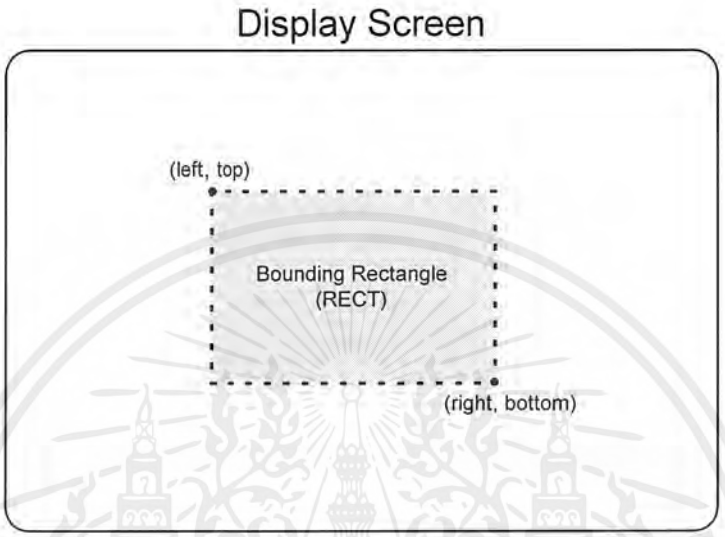


รูปที่ 3-2 Complex Surface

3.3.5 รู้จักกับ Rectangle

จากแนวคิดของ DirectDraw และการเขียนโปรแกรมบนวินโดวส์ วัตถุต่างๆ บนหน้าจอจะถูกอ้างอิงในรูปกรอบสี่เหลี่ยมผืนผ้า โดยขอบของกรอบสี่เหลี่ยมผืนผ้ามักจะขนานกับขอบของหน้าจอ ดังนั้นสี่เหลี่ยม

ผืนผ้าดังกล่าวสามารถอธิบายได้โดยจุด 2 จุด คือมุมบน-ซ้าย และมุมล่าง-ขวา โดยแอปพลิเคชั่นส่วนใหญ่จะใช้โครงสร้างข้อมูล RECT ในการเก็บข้อมูลเกี่ยวกับกรอบสี่เหลี่ยมนี้ไว้ใช้เมื่อมีการ blitting บนหน้าจอหรือการตรวจสอบการชนกันของวัตถุต่างๆ



รูปที่ 3-3 ตัวอย่างกรอบสี่เหลี่ยมผืนผ้า

DirectDraw มีฟังก์ชัน blitting ที่ใช้กรอบสี่เหลี่ยมในการอ้างอิงวัตถุต่างๆ เนื่องจากเหตุผลทางด้านประสิทธิภาพ เสถียรภาพ และความง่ายในการใช้งาน อย่างไรก็ตาม เราสามารถสร้างกระบวนการ blit ในรูปแบบอื่นที่ไม่ใช่กรอบสี่เหลี่ยมได้เช่นกัน

3.4 สถาปัตยกรรมของ DirectDraw

เนื้อหาในส่วนนี้ประกอบด้วย ข้อมูลทั่วไปเกี่ยวกับความสัมพันธ์ระหว่างคอมโพเนนท์ DirectDraw กับคอมโพเนนท์อื่นๆ ของ DirectX ระบบปฏิบัติการ และฮาร์ดแวร์ภายในระบบ

3.4.1 โครงสร้างโดยทั่วไปของ DirectDraw

ซอฟต์แวร์ด้านมัลติมีเดียต้องการการแสดงผลกราฟิกในระดับสูง เมื่อเปรียบเทียบกับการใช้ GDI แล้ว นับได้ว่า DirectDraw ทำให้แอปพลิเคชั่นด้านกราฟิกบนวินโดวส์มีประสิทธิภาพและความเร็วในระดับที่สูงขึ้นกว่าเดิมมาก ในขณะเดียวกันยังคงความเป็นอิสระไม่ขึ้นกับอุปกรณ์ไว้ด้วย DirectDraw มีเครื่องมือที่ช่วยในการแสดงงาหลักทางด้านกราฟิกต่างๆ ซึ่งได้แก่

- การใช้ surface แสดงผลหลายๆ surface
- การเข้าถึงหน่วยความจำวีดีโอได้โดยตรง
- การทำ page flipping
- การจัดการพาเลต
- การทำ clipping

นอกจากนี้ DirectDraw ยังทำให้เราสามารถดึงความสามารถของฮาร์ดแวร์แสดงผลมาใช้ในขณะ run time และนำเสนอการแสดงผลที่ดีที่สุดเท่าที่จะเป็นไปได้จากความสามารถของฮาร์ดแวร์ที่อยู่บนเครื่องคอมพิวเตอร์ของผู้ใช้ซอฟต์แวร์

ในขณะที่ทำงานร่วมกับคอมโพเนนท์อื่นๆ DirectDraw จะใช้ฮาร์ดแวร์ให้เกิดประโยชน์สูงสุดเท่าที่จะเป็นไปได้ และสามารถทำการจำลองทางซอฟต์แวร์สำหรับคุณลักษณะที่ดีที่สุดในกรณีที่ฮาร์ดแวร์ไม่สามารถทำได้ ความอิสระไม่ขึ้นกับอุปกรณ์เกิดขึ้นได้เมื่อใช้การทำงานระดับ Hardware Abstraction Layer หรือ HAL ซึ่งจะกล่าวรายระเอียดต่อไป

คอมโพเนนท์ DirectDraw จัดหาบริการผ่านทางอินเทอร์เฟซที่มีแนวคิดมาจาก COM ขณะที่กำลังทำโครงงานชิ้นนี้ DirectDraw มีอินเทอร์เฟซดังนี้ IDirectDraw4 IDirectDrawSurface4 IDirectDrawPalette IdirectDrawClipper และ IDirectDrawVideoPort

1. DirectDraw objects มีหน้าที่หลักในการจัดการกับระบบการแสดงผล และใช้ในการติดต่อกับส่วนอื่นๆ ในลำดับถัดไป
2. IDirectDrawSurface objects ใช้ในการจัดการกับรูปภาพหรือข้อความที่จะทำการแสดงที่หน้าจอ โดยมีฟังก์ชันและเมธอดต่างๆ สนับสนุน
3. IDirectDrawPalette objects จะช่วยในการจัดการกับพาเลต เพื่อที่จะใช้ในการแสดงโหมดสีของรูปภาพที่จะทำการแสดงบนจอภาพ
4. IDirectDrawClipper objects ช่วยในการป้องกันไม่ให้รูปที่จะแสดง ออกจากขอบเขตที่กำหนดไว้ให้ ซึ่งจะใช้ในการสร้างเกมในโหมดวินโดว์
5. IDirectDrawVideoPort มีเมธอดที่ใช้ในการส่งผ่านข้อมูลวีดีโอสดๆ จากพอร์ตวีดีโอของฮาร์ดแวร์ไปยัง surface ของ DirectDraw

3.4.2 Hardware Abstraction Layer (HAL)

DirectDraw ทำให้แอปพลิเคชั่นที่สร้างขึ้นมีอิสระไม่ขึ้นกับฮาร์ดแวร์โดยการทำงานในระดับ Hardware Abstraction Layer หรือ HAL ซึ่งเป็นอินเทอร์เฟซโดยเฉพาะของอุปกรณ์ ที่ได้จากผู้ผลิตอุปกรณ์แต่ละชนิด โดย DirectDraw จะนำมาใช้ในการทำงานกับฮาร์ดแวร์แสดงผลโดยตรง แอปพลิเคชั่นไม่ได้ติดต่อกับ HAL แต่จะติดต่อกับโครงสร้างในระดับต่ำกว่าที่ HAL นำเสนอให้ หรือกล่าวได้ว่า DirectDraw เป็นผู้เปิดทางให้แอปพลิเคชั่นได้ใช้ชุดของอินเทอร์เฟซและเมธอดที่เสถียรภาพในการแสดงผลกราฟิก ผู้ผลิตอุปกรณ์สร้าง

HAL เป็นโค้ดในรูปแบบ 16 บิตรวมกับรูปแบบ 32 บิตสำหรับระบบวินโดวส์ แต่สำหรับวินโดวส์ NT นั้น HAL จะถูกใช้ในรูปแบบโค้ด 32 บิต ในที่นี้ HAL อาจเป็นส่วนหนึ่งในไดรเวอร์แสดงผล หรือ DLL ซึ่งจะติดต่อกับไดรเวอร์แสดงผลผ่านอินเทอร์เฟซที่สร้างขึ้นมาเป็นพิเศษโดยผู้ผลิตไดรเวอร์

HAL นั้นถูกสร้างขึ้นมาโดยผู้ผลิตชิป ผู้สร้างบอร์ด หรือผู้ผลิตอุปกรณ์นั้นๆ โดย HAL เป็นเพียงโค้ดที่สร้างขึ้นเพื่อทำให้เกิดความอิสระจากอุปกรณ์ และไม่ได้แสดงผลการจำลองใดๆ หากมีฟังก์ชันการทำงานใดที่ไม่สามารถแสดงได้โดยฮาร์ดแวร์ HAL ก็จะไม่รายงานฟังก์ชันนั้นเป็นความสามารถของฮาร์ดแวร์

3.4.3 Software Emulation

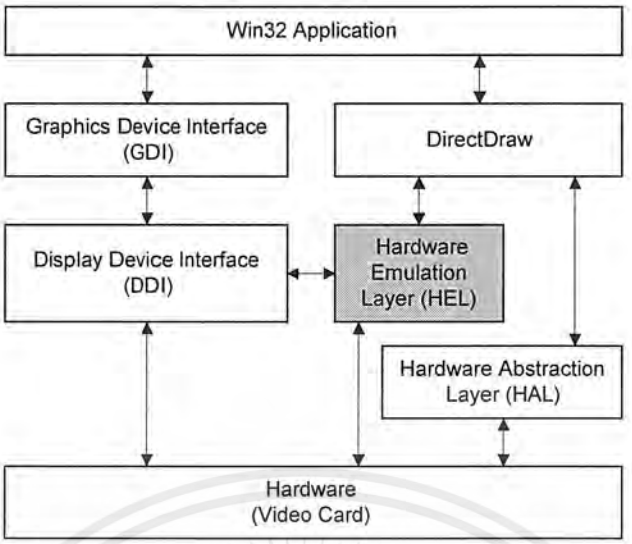
เมื่อฮาร์ดแวร์ไม่สนับสนุนคุณลักษณะผ่านทาง Hardware Abstraction Layer หรือ HAL DirectDraw จะทำการจำลองคุณลักษณะนั้นขึ้นมา โดยฟังก์ชันที่ทำการจำลองนี้ทำได้ผ่านทาง Hardware Emulation Layer หรือ HEL โดย HEL มีการทำงานเช่นเดียวกับ HAL คือจะเสนอความสามารถของตนให้ DirectDraw และแอปพลิเคชั่นก็จะไม่ทำงานกับ HEL โดยตรง ผลลัพธ์ที่ได้จึงเหมือนกับแอปพลิเคชั่นได้รับการสนับสนุนสำหรับคุณลักษณะที่สำคัญๆ เกือบทุกชนิด ไม่ว่าคุณสมบัติดังกล่าวจะจะสนับสนุนโดยฮาร์ดแวร์หรือผ่านทาง HEL ก็ตาม

แน่นอนว่าการจำลองโดยซอฟต์แวร์ไม่สามารถเทียบเท่าการแสดงผลที่ได้จากคุณลักษณะต่างๆ ของฮาร์ดแวร์โดยตรงได้ เราสามารถร้องขอคุณลักษณะต่างๆ ที่ฮาร์ดแวร์สนับสนุนได้โดยใช้เมธอดของ DirectDraw (IDirectDraw ::GetCaps) ซึ่งจะตรวจสอบพิจารณาคุณลักษณะเหล่านี้ระหว่างช่วงกำหนดค่าเริ่มต้นของแอปพลิเคชั่น เราสามารถปรับเปลี่ยนค่าตัวแปรต่างๆ ที่จะส่งผลให้การแสดงผลเป็นไปอย่างเหมาะสมที่สุดภายใต้ฮาร์ดแวร์ที่มีประสิทธิภาพแตกต่างกันไป

ส่วนใหญ่แล้ว เมื่อใช้คุณลักษณะที่สนับสนุนโดยฮาร์ดแวร์ร่วมกับการจำลองสามารถส่งผลให้แสดงผลได้ช้ากว่าการจำลองโดยซอฟต์แวร์เพียงอย่างเดียว ตัวอย่างเช่น ถ้าหากไดรเวอร์อุปกรณ์แสดงผลสนับสนุน DirectDraw แต่ไม่สนับสนุนการทำ Stretch blitting เราสามารถสังเกตเห็นถึงประสิทธิภาพที่สูญเสียไปเมื่อมีการ stretching blitting จาก surface บนหน่วยความจำวีดีโอ ที่เป็นเช่นนี้เพราะว่าหน่วยความจำวีดีโอมักจะช้ากว่าหน่วยความจำของระบบ ทำให้ CPU ต้องรอเมื่อมีการเข้าถึง surface บนหน่วยความจำวีดีโอ ดังนั้นบางครั้งการสร้าง surface บนหน่วยความจำของระบบอาจเป็นสิ่งที่ดีที่สุดก็ได้

3.4.4 การทำงานรวมกันภายในระบบ

ไดอะแกรมข้างล่างแสดงถึงความสัมพันธ์ระหว่าง DirectDraw, อินเทอร์เฟซของอุปกรณ์กราฟิกหรือ GDI, Hardware Abstraction Layer หรือ HAL, Hardware Emulation Layer หรือ HEL และฮาร์ดแวร์



รูปที่ 3-4 ความสัมพันธ์ระหว่าง DirectDraw, GDI, HAL, HEL และฮาร์ดแวร์ในระบบ

จากรูปดังกล่าวจะเห็นว่าออบเจ็กต์ DirectDraw อยู่ระดับเดียวกับ GDI และทั้งคู่มีการเข้าถึงฮาร์ดแวร์โดยตรงผ่านเลเยอร์หนึ่งซึ่งเป็นอิสระไม่ขึ้นกับอุปกรณ์ แต่ DirectDraw ต่างจาก GDI คือจะใช้ประโยชน์จากคุณลักษณะพิเศษของฮาร์ดแวร์เมื่อใดก็ตามที่เป็นไปได้ หากฮาร์ดแวร์ไม่สนับสนุนคุณลักษณะใด DirectDraw จะพยายามจำลองคุณลักษณะนั้นขึ้นมาโดยใช้ HEL นอกจากนี้ DirectDraw สามารถนำเสนอหน่วยความจำของ surface ในรูปแบบ device context ซึ่งจะทำให้เราสามารถใช้ฟังก์ชันต่างๆ ของ GDI ในการทำงานกับออบเจ็กต์ surface ได้

บทที่ 4

Direct3D

4.1 รู้จักกับ Direct3D

Direct3D ถูกออบแบบมาสำหรับใช้ในการพัฒนาเกมและแอปพลิเคชั่นด้านกราฟิก 3 มิติบนคอมพิวเตอร์ที่ใช้ระบบวินโดวส์ของไมโครซอฟต์ หน้าที่หลักของ Direct3D คือการนำเสนอรูปแบบการเข้าถึงฮาร์ดแวร์วีดีโอแสดงผลทางด้าน 3 มิติโดยเป็นอิสระไม่ขึ้นกับอุปกรณ์ หรืออาจกล่าวได้ว่า Direct3D เป็นอินเทอร์เฟซในการวาดภาพของฮาร์ดแวร์ 3 มิติ

เราสามารถใช้ Direct3D ได้ทั้ง 2 โหมด คือโหมด Immediate หรือโหมด Retained โดยโหมด Retained จะเป็นอินเทอร์เฟซในการเขียนโปรแกรมแอปพลิเคชั่น (API) 3 มิติในระดับสูงที่มีไว้สำหรับโปรแกรมเมอร์ที่ต้องการทำการพัฒนาอย่างรวดเร็ว หรือผู้ที่ต้องการส่วนช่วยเหลือหรือ help ใน built-in ของโหมด Retained ที่สนับสนุนการสืบทอดเป็นลำดับชั้น (hierarchies) และการทำภาพเคลื่อนไหว

บริษัทไมโครซอฟต์ได้พัฒนาโหมด Immediate ของ Direct3D เป็น API 3 มิติในระดับล่าง ซึ่งถือเป็นเครื่องมือชั้นเลิศให้กับผู้พัฒนาที่ต้องการเชื่อมโยงเกมหรือแอปพลิเคชั่นทางด้านมัลติมีเดียที่มีประสิทธิภาพสูงอื่นๆ เข้ากับระบบปฏิบัติการวินโดวส์ของไมโครซอฟต์ โหมด Immediate เป็นวิธีสำหรับแอปพลิเคชั่นในการติดต่อกับฮาร์ดแวร์ตัวเร่งในระดับล่างโดยเป็นอิสระไม่ขึ้นกับอุปกรณ์ ดังนั้นหากมองเป็นภาพโดยรวมจะเห็นว่าโหมด Retained ของ Direct3D ถูกสร้างขึ้นมาอยู่ในระดับเหนือจากโหมด Immediate อีกทีหนึ่ง

ต่อไปนี้เป็นคุณลักษณะเด่นของ Direct3D

- สามารถสลับสับเปลี่ยนระดับการทำ buffering ได้ (ใช้ z-buffers หรือ w-buffers)
- การทำเงารูปแบบ Flat และ Gouraud
- จัดให้มีแหล่งกำเนิดแสงได้หลายที่และแสงหลายชนิดได้
- สนับสนุน material และ texture รวมทั้งการทำ mipmappping
- มีไดรเวอร์การจำลองโดยซอฟต์แวร์ที่มั่นคงแข็งแรง
- สามารถทำการเปลี่ยนแปลงหรือ Transformation และ clipping ได้
- มีความอิสระไม่ขึ้นกับอุปกรณ์
- สนับสนุนบนระบบ NT
- สนับสนุนสถาปัตยกรรม MMX ของ Intel

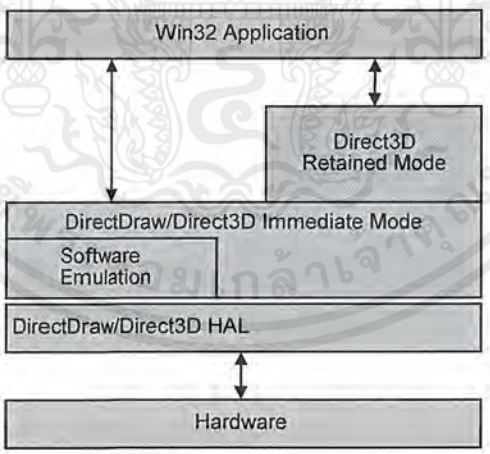
โดยทั่วไปผู้พัฒนาที่เลือกใช้โหมด Immediate แทนการใช้โหมด Retained จะต้องมีประสบการณ์ด้านการเขียนโปรแกรมในระดับสูงเสียก่อน และอาจต้องมีประสบการณ์ด้านกราฟิก 3 มิติด้วย

4.2 Direct3D Immediate Mode

โครงสร้างการจัดการของโหมด Immediate นั้นมีรากฐานมาจาก vertices, polygons และคำสั่งที่จะใช้ควบคุม จึงทำให้สามารถเข้าถึงการเปลี่ยนแปลง การจัดแสง และการปรับเปลี่ยน pipeline ของกราฟิก 3 มิติได้อย่างใกล้ชิด หากฮาร์ดแวร์ไม่สามารถเร่งการเรนเดอร์ได้ Direct3D ก็จะนำเสนอทางเลือกให้กับแอปพลิเคชั่นว่าจะใช้ไดรเวอร์การจำลองทางซอฟต์แวร์แทนก็ได้ โหมด Immediate เหมาะสำหรับผู้พัฒนาที่มีแอปพลิเคชั่น 3 มิติอยู่แล้วและต้องการประสิทธิภาพสูงสุดโดยการรักษาระดับเลเยอร์ระหว่างแอปพลิเคชั่นกับฮาร์ดแวร์ให้บางที่สุดเท่าที่จะทำได้

มีวิธีอยู่ 2 วิธีในการใช้โหมด Immediate คือเราสามารถใช้เมธอด DrawPrimitive หรือเราจะทำงานกับ execute buffer ก็ได้ ผู้พัฒนาส่วนใหญ่ที่ได้เขียนโค้ดที่ใช้ execute buffer ไปแล้วก็อาจจะทำเช่นนั้นต่อไป ไม่มีเทคนิคไหนที่เร็วกว่าแน่นอน เราควรเลือกโดยดูจากความต้องการของแอปพลิเคชั่นและสไตล์การเขียนโปรแกรมของเรา ในที่นี้จะไม่ขอกล่าวในรายละเอียดในการใช้ 2 เทคนิคนี้เนื่องจากโครงงานนี้ไม่ได้ใช้โหมด Immediate

โครงสร้างสถาปัตยกรรมของ Direct3D สามารถดูได้จากรูปข้างล่าง แอปพลิเคชั่น Dirct3D จะติดต่อกับฮาร์ดแวร์กราฟิกโดยวิธีคล้ายกัน ไม่ว่าจะใช้โหมด Retained หรือโหมด Immediate ก็ตาม และแอปพลิเคชั่นทั้งหลายสามารถเลือกว่าจะใช้ประโยชน์จากการจำลองโดยซอฟต์แวร์หรือไม่ก็ได้ ก่อนที่จะติดต่อกับ HAL เนื่องจาก Direct3D เป็นอินเทอร์เฟซของออบเจ็กต์ DirectDraw ดังนั้น HAL ในที่นี้จะหมายถึง DirectDraw/Direct3D HAL



รูปที่4-1 การติดต่อระหว่างแอปพลิเคชั่นกับฮาร์ดแวร์เมื่อใช้ Direct3D

Direct3D นั้นรวมอยู่อย่างใกล้ชิดกับคอมโพเนนท์ DirectDraw ของ DirectX ดังนั้น surface ทั้งหลายของ DirectDraw จะถูกใช้เป็นเป้าหมายในการเรนเดอร์ (surface หน้าและหลัง) และใช้เป็น z-buffers แท้จริงแล้วอินเทอร์เฟซ COM ของ Direct3D เป็นอินเทอร์เฟซของออบเจ็กต์ DirectDraw อีกชั้นหนึ่งนั่นเอง

4.3 ความรู้เบื้องต้นเกี่ยวกับกราฟิก 3 มิติ

เนื้อหาในส่วนนี้จะอธิบายถึงแนวคิดด้านเทคนิคบางประการที่เราจำเป็นต้องเข้าใจก่อนที่จะเขียนโปรแกรมด้านกราฟิก 3 มิติได้ โดยจะกล่าวถึงเพียงพื้นฐานทั่วไปเกี่ยวกับระบบพิกัดและการเปลี่ยนแปลงต่างๆ ไม่รวมถึงด้านรายละเอียดถึงด้านสถาปัตยกรรม เช่น การจัดวางโมเดล การจัดแสง และค่าตัวแปรต่างๆ ซึ่งสามารถดูได้จากหัวข้อ สถาปัตยกรรมของ Direct3D Retained- Mode

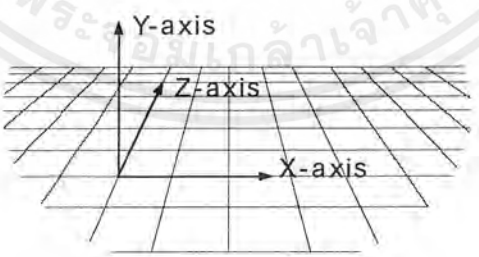
4.3.1 ระบบพิกัด 3 มิติ

ระบบพิกัดคาร์ทีเซียนในกราฟิก 3 มิติมีด้วยกัน 2 ประเภท คือระบบมือซ้าย และระบบมือขวา ในระบบพิกัดทั้งสองประเภท จะมีแกน x บวกชี้ไปทางด้านขวา และแกน y บวกชี้ขึ้นข้างบน มีวิธีในการจำว่าแกน z บวกนั้นจะชี้ไปทิศทางใด โดยชี้นิ้วไม่ว่าจะเป็นมือซ้ายหรือขวาก็ได้ไปในทิศของแกน x บวกแล้วงอนิ้วให้ชี้ไปในทิศของแกน y บวก หากนิ้วโป้งของเราชี้ไปในทิศทางใด ไม่ว่าจะชี้เข้าหรือชี้ออกจากตัวเรา จะเป็นทิศทางของแกน z บวกของระบบพิกัด

ในส่วนนี้จะอธิบายถึงระบบพิกัดของ Direct3D และระบบพิกัดประเภทต่างๆ ที่จะใช้ในแอปพลิเคชั่น

4.3.1.1 ระบบพิกัดของ Direct3D

Direct3D ใช้ระบบพิกัดแบบซ้ายมือ จึงหมายความว่าแกน z บวกจะชี้ไปในทิศทางออกจากตัวผู้มองดังที่แสดงในรูป



รูปที่ 4-2 ระบบพิกัดของ Direct3D

ในระบบพิกัดแบบซ้ายมือ การหมุนจะเป็นแบบตามเข็มนาฬิการอบแกนใดก็ตามที่ชี้มายังผู้มอง หากเราต้องการจะทำงานในระบบพิกัดขวามือ เราสามารถทำได้โดยเปลี่ยนแปลง 2 สิ่งกับข้อมูลที่จะผ่านเข้ามาใน Direct3D

- Flip ลำดับของ vertices รูปสามเหลี่ยม เพื่อทำให้ระบบหมุนกลับด้านตามเข็มนาฬิกาจากด้านหน้า หรือกล่าวอีกอย่างหนึ่งว่า vertices คือ v0 v1 v2 จะถูกผ่านค่าสู่ Direct3D เป็น v0 v2 v1
- กำหนดจุดบนระบบพิกัดลงในเมตริกซ์ตามทิศทางในแกน z ซึ่งจะเป็นโครงสร้างแบบ D3DMATRIX

4.3.1.2 ระบบพิกัดแบบ U และแบบ V

Direct3D ใช้พิกัด textureซึ่งพิกัดเหล่านี้ (แบบ U และแบบ V) จะใช้เมื่อต้องการกำหนด texture ลงบนวัตถุ เวคเตอร์ V ใช้อธิบายตำแหน่งและทิศทางการหันเหของ texture และวางตัวตามแกน z ส่วนเวคเตอร์ U (หรือเรียกว่าเวคเตอร์ชี้ขึ้นหรือ up vector) โดยทั่วไปจะวางตัวตามแกน y ด้วยจุดกำเนิด [0,0,0]

4.3.2 การเปลี่ยนแปลงทาง 3 มิติ

ในโปรแกรมที่ทำงานกับกราฟิก 3 มิติ เราสามารถใช้การเปลี่ยนแปลงทางเรขาคณิตเพื่อ

- แสดงตำแหน่งของวัตถุสัมพันธ์กับวัตถุอื่นๆ
- หมุน ตัด และกำหนดขนาดของวัตถุ
- เปลี่ยนตำแหน่ง ทิศทาง และสัดส่วนหรือ perspective ของมุมมอง

เราสามารถแปลงจุดใดๆ ไปเป็นอีกจุดหนึ่งโดยการใช้เมตริกซ์ขนาด 4×4 ตัวอย่างต่อไปนี้เป็นการใช้เมตริกซ์ในการแสดงจุด (x, y, z) และสร้างจุด (x', y', z') ขึ้นมาใหม่

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

เรากระทำปฏิบัติการต่อไปนี้กับ (x, y, z) และเมตริกซ์เพื่อสร้างจุด (x', y', z'):

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

การเปลี่ยนแปลงที่พบบ่อยๆ มักจะเป็นการเปลี่ยนตำแหน่ง การหมุน และการเปลี่ยนขนาด เราสามารถแสดงการกระทำเหล่านี้โดยใช้เพียงเมตริกซ์เดียว และทำการคำนวณการเปลี่ยนแปลงหลายๆอย่างได้ในครั้งเดียว

จำนวนในเมตริกซ์จะถูกเจาะจงตามลำดับแถว ตัวอย่างเช่น เมตริกซ์ต่อไปนี้สามารถแสดงแทนได้ด้วยอาเรย์

$$\begin{bmatrix} s & 0 & 0 & 0 \\ 0 & s & t & 0 \\ 0 & 0 & s & v \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

โดยอาเรย์สำหรับเมตริกซ์นี้จะมีลักษณะดังนี้

```
D3DMATRIX scale = {
  D3DVAL(s),  0,          0,          0,
  0,          D3DVAL(s),  D3DVAL(t),  0,
  0,          0,          D3DVAL(s),  D3DVAL(v),
  0,          0,          0,          D3DVAL(1)
};
```

4.3.2.1 การเปลี่ยนตำแหน่ง

การเปลี่ยนแปลงต่อไปนี้เป็นการเปลี่ยนตำแหน่งจากจุด (x, y, z) ไปยังจุดใหม่ (x', y', z')

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

4.3.2.2 การหมุน

การเปลี่ยนแปลงที่จะอธิบายในส่วนนี้ใช้สำหรับระบบพิกัดแบบซ้ายมือ โดยการเปลี่ยนแปลงนี้จะทำการหมุนจุด (x, y, z) รอบแกน x จึงเกิดจุดใหม่ (x', y', z')

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

การเปลี่ยนแปลงนี้จะทำการหมุนจุดรอบแกน y

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

การเปลี่ยนแปลงนี้จะทำการหมุนจุดรอบแกน z

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

สังเกตว่าในตัวอย่างเหล่านี้ ตัวอักษรกรีก เธต้า หรือ theta แสดงถึงมุมของการหมุน โดยใช้รูปแบบเรเดียน มุมต่างๆ จะวัดตามเข็มนาฬิกา เมื่อมองจากปลายแกนที่ทำการหมุนรอบไปยังจุดกำเนิด

4.3.2.3 การเปลี่ยนขนาด

การเปลี่ยนแปลงต่อไปนี้เป็นการเปลี่ยนขนาดจุด (x, y, z) ตามค่าต่างๆ ในทิศทาง x, y และ z ไปเป็นจุดใหม่ (x', y', z')

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

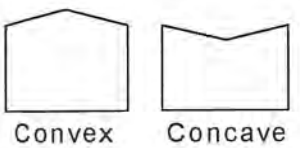
4.3.3 โพลีกอน (Polygon)

วัตถุ 3 มิติใน Direct3D ถูกสร้างขึ้นมาจาก mesh จำนวนมาก mesh เป็นชุดของ faces ซึ่งแต่ละอันจะอธิบายโดยโพลีกอน ชนิดของโพลีกอนพื้นฐานเป็นแบบสามเหลี่ยม แม้ว่าแอปพลิเคชั่นแบบโหมด Retained จะสามารถกำหนดให้โพลีกอนประกอบด้วย vertices มากกว่า 3 อันได้ แต่ระบบจะต้องแปลงเป็นสามเหลี่ยมก่อนที่จะทำการเรนเดอร์วัตถุ ส่วนแอปพลิเคชั่น ในโหมด Immediate จะต้องใช้แบบสามเหลี่ยมเท่านั้น

เนื้อหาในส่วนนี้จะอธิบายว่าแอปพลิเคชั่นของเราสามารถใช้โพลีกอนทั้งหลายใน Direct3D ได้อย่างไร

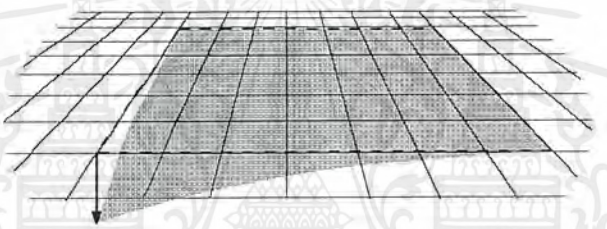
4.3.3.1 ความต้องการด้านเรขาคณิต

โพลีกอนมักจะเป็นรูปแบบสามเหลี่ยม เนื่องจากสามเหลี่ยมจะเป็นส่วนที่นูนออก (convex) และจะเป็นระนาบเสมอ มีเงื่อนไขอยู่ 2 อย่างที่ต้องการเมื่อโพลีกอนจะถูกเรนเดอร์ คือ โพลีกอนจะต้องเป็นส่วนที่นูนออกถ้าหากเส้นถูกวาดระหว่างจุดใดๆ 2 จุดของโพลีกอน เส้นนั้นจะต้องอยู่ภายในโพลีกอนนั้นด้วย



รูปที่ 4-3 พื้นผิวแบบนูนออก (convex) และแบบเว้าเข้า (concave)

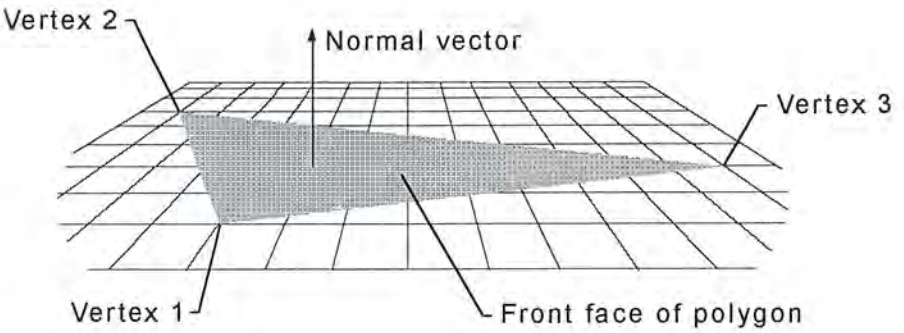
vertices ทั้งสามของสามเหลี่ยมมักจะถูกอธิบายด้วยระนาบ แต่การสร้างโพลีกอนที่ไม่เป็นระนาบโดยไม่ได้ตั้งใจมักจะเกิดได้เสมอหากทำการเพิ่ม vertex ลงไปอีกอันหนึ่ง



รูปที่ 4-4 ตัวอย่างโพลีกอนที่ไม่เป็นระนาบ

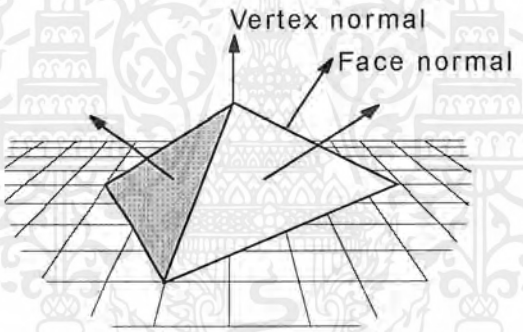
4.3.3.2 Face Normal และ Vertex Normal

แต่ละ face ภายใน mesh จะมีเวคเตอร์เรียกว่า normal ซึ่งตั้งฉากกับ face โดยทิศทางของเวคเตอร์จะขึ้นอยู่กับการนิยาม vertices ทั้งหลายและระบบพิกัดที่ใช้ว่าเป็นแบบขวามือหรือแบบซ้ายมือด้วย หากเวคเตอร์ normal ของ face มีทิศทางหันเหเข้าสู่ผู้มอง แสดงว่าด้านนั้นเป็นด้านหน้าของ face ใน Direct3D เฉพาะด้านหน้าเท่านั้นที่จะถูกมองเห็น และด้านหน้าเป็นด้านที่ vertices ถูกนิยามตามลำดับตามเข็มนาฬิกา



รูปที่ 4-5 เวคเตอร์ normal และด้านหน้าของ face

แอปพลิเคชั่น Direct3D ไม่จำเป็นต้องกำหนด face normal เอง ระบบจะทำการคำนวณให้โดยอัตโนมัติเมื่อใดก็ตามที่จำเป็นต้องใช้ ระบบจะใช้ face normal ในโหมดแสงเงาแบบ flat สำหรับโหมดแสงเงาแบบ Phong และ Gouraud และการควบคุมแสงและการกระทำต่างๆ เกี่ยวกับ texture ระบบจะใช้ vertext normal



รูปที่ 4-6 Face Normal และ Vertex Normal

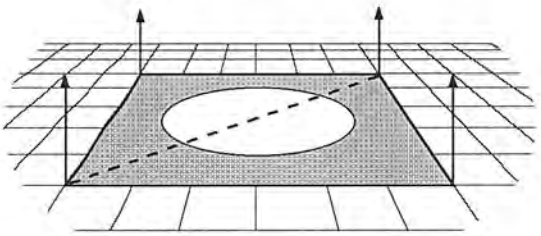
4.3.3.3 โหมดแสงเงา (Shade Modes)

ในโหมดแสงเงาแบบ flat ระบบจะทำการจำลองสีของ vertex หนึ่งให้กับ face อื่นอย่างหยาบๆ แต่ในโหมดแสงเงาแบบ Gouraud และแบบ Phong ระบบจะใช้ vertex normal ในการคำนวณเพื่อให้วัตถุที่เป็นโพลีกอนมองดูเรียบเนียนขึ้น ในการทำแสงเงาแบบ Gouraud สีและความหนาแน่นของ vertices ที่อยู่ใกล้เคียงจะถูกนำมาใช้ในการคำนวณและสอดแทรกลงบนพื้นที่ระหว่าง vertices เหล่านั้น ส่วนการให้แสงเงาแบบ Phong ระบบจะคำนวณค่าแสงเงาที่เหมาะสมให้แต่ละ pixel บนพื้นผิวของวัตถุ

ข้อสังเกต : การจัดแสงเงาแบบ Phong ยังไม่ได้รับการสนับสนุนในปัจจุบัน

แอปพลิเคชั่นส่วนใหญ่จะใช้การจัดแสงเงาแบบ Gouraud เพราะว่าทำให้วัตถุปรากฏอย่างเรียบเนียนและถูกคำนวณอย่างประสิทธิภาพ การทำแสงเงาแบบ Gouraud อาจขาดรายละเอียดบางอย่างที่การทำแสงเงา

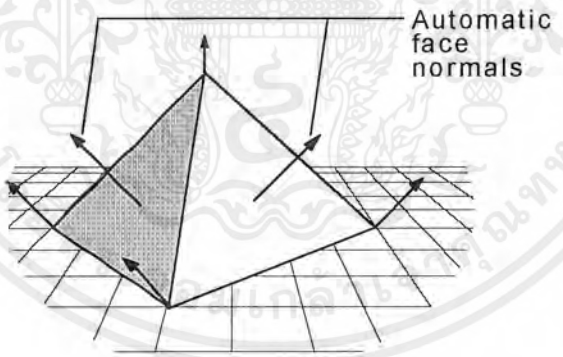
แบบ Phong สามารถทำได้ ตัวอย่างเช่น การทำแสงเงาแบบ Gouraud และ Phong จะสร้างผลลัพธ์ที่แตกต่างกันไป ดังตัวอย่างที่แสดงในรูปข้างล่าง ซึ่งสปอตไลต์ ถูกส่องไปบน face



รูปที่ 4-7 แสงสปอตไลต์ส่องไปบน face

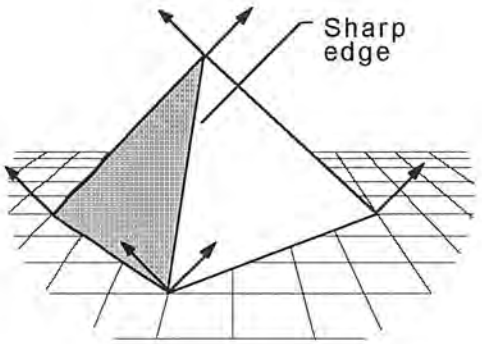
ในกรณีนี้ โหมดแสงเงาแบบ Phong จะคำนวณค่าของแต่ละ pixel และแสดงแสงสปอตไลต์ ส่วนโหมดแสงเงาแบบ Gouraud จะสอดแทรกแสงลงไประหว่าง vertices ต่างๆ อาจจะทำให้ดูกลมกลืนจนเรนเดอร์ออกมาเหมือนกับว่าไม่มีแสงสปอตไลต์อยู่

ในโหมดแสงเงาแบบ flat ปิรามิดข้างล่างจะถูกแสดงโดยมีขอบที่แหลมคมระหว่างพื้นผิวแต่ละด้านที่มาต่อกัน ระบบจะสร้าง face normal โดยอัตโนมัติ อย่างไรก็ตามในโหมดแสงเงาแบบ Gouraud หือแบบ Phong ค่าต่างๆ ในการทำแสงเงาจะถูกสอดแทรกลงไประหว่างขอบ จนทำให้ผลที่ปรากฏออกมาเป็นพื้นผิวที่โค้งสวยงามขึ้น



รูปที่ 4-8 ปิรามิดที่มีขอบแหลมคม

หากเราต้องการใช้โหมดแสงเงาแบบ Gouraud หรือแบบ Phong ในการแสดงผลพื้นผิวที่โค้งมน และเราต้องการจะเพิ่มออบเจ็กต์บางอันที่มีขอบแหลมคมด้วยเช่นกัน แอปพลิเคชั่นจะต้องจำลอง vertex normal ที่บริเวณจุดตัดของด้านใดๆ ซึ่งเราต้องการให้เป็นขอบแหลมคม ดังแสดงในรูปข้างล่าง



รูปที่ 4-9 vertex normal ที่บริเวณจุดตัดของด้านซึ่งเป็นขอบแหลมคม

นอกจากนี้การทำให้ออบเจ็กต์หนึ่งมีได้ทั้งพื้นผิวที่โค้งมนและเรียบแบน การให้แสงลงบนพื้นผิวราบเรียบโดยใช้โหมดแสงเงาแบบ Gouraud นั้นทำได้สมจริงมากกว่าโหมด flat ด้านที่อยู่ในโหมดแสงเงาแบบ flat จะเป็นสีเดียวกัน แต่ในโหมด Gouraud จะสามารถทำให้แสงตกกระทบบนด้านอย่างถูกต้อง โดยการทำเช่นนี้จะส่งผลแตกต่างอย่างเห็นได้ชัดหากด้านดังกล่าวอยู่ใกล้กับแหล่งกำเนิดแสง

4.4 สถาปัตยกรรมของ Direct3D Retained Mode

การเข้าถึงโหมด Retained ของ Direct3D สามารถทำได้ผ่านทางออบเจ็กต์กลุ่มเล็กๆ ซึ่งมีรายละเอียดดังนี้

ออบเจ็กต์	*คำอธิบาย*
Direct3DRMAnimation2	ใช้ในการกำหนดว่าจะให้มีการเปลี่ยนแปลงเกิดขึ้นอย่างไร มักจะอ้างอิงกับออบเจ็กต์Direct3DRMFrame3เราสามารถกำหนดตำแหน่ง ทิศทางการหันเหและขนาดภายในการเคลื่อนไหวของออบเจ็กต์ Direct3DRMVisual, Direct3DRMLight และ Direct3DRMViewport2 ได้ โดยมีหลักการเหมือนกับ Direct3DRMAnimation แต่เพิ่มความสามารถในการเข้าถึงกุญแจสำคัญและเฟรมภายในการเคลื่อนไหวได้
Direct3DRMAnimationArray	แก้ไขการเคลื่อนไหวต่างๆ ในอาเรย์และมีอินเทอร์เฟซในการร้องขอการเคลื่อนไหว
Direct3DRMAnimationSet2	อนุญาตให้ออบเจ็กต์ Direct3DRMAnimation2 สามารถรวมกลุ่มเข้าด้วยกัน มีหลักการเช่นเดียวกับ Direct3DRMAnimationSet แต่เพิ่มความสามารถในการเข้าถึงองค์ประกอบภายในการเคลื่อนไหวได้

Direct3DRMClippedVisual	อนุญาตให้ clipping ปรากฏบนพื้นผิวใดๆ ก่อนที่จะทำการเรนเดอร์
Direct3DRMDevice3	แทนจุดหมายที่ต้องการแสดงผลให้เห็นสำหรับผู้เรนเดอร์ มีคุณสมบัติเพิ่มเติมจากออบเจ็กต์ Direct3DRMDevice2 คือเราสามารถจัดและแก้ไข material ทั้งหลายได้และมีฟังก์ชันบางอย่างที่มีการควบคุมในระดับที่สูงขึ้น
Direct3DRMFace2	แสดงถึงแต่ละโพลีกอนภายใน mesh
Direct3DRMFrame3	จัดตำแหน่งของออบเจ็กต์ภายในฉากและกำหนดตำแหน่งและการหันเหของออบเจ็กต์ที่มองเห็น มีส่วนที่เพิ่มเติมจากออบเจ็กต์ Direct3DRMFrame คือสามารถเข้าถึงแกนต่างๆ ของเฟรม ขอบเขตรูปลูกบาศก์ และ material ได้ และขยายจาก Direct3DRMFrame2 โดยเพิ่มความสามารถในการเปลี่ยนแปลง การเปลี่ยนในแนวขวาง และการจัดหมอก
Direct3DRMInterpolator	จัดเก็บการกระทำหรือ action ต่างๆ และใช้การกระทำเหล่านั้นกับออบเจ็กต์โดยคำนวณค่าต่างๆ ให้โดยอัตโนมัติ
Direct3DRMLight	กำหนดแหล่งกำเนิดแสงซึ่งมีอยู่ด้วยกัน 5 ชนิดเพื่อให้แสงส่องไปยังวัตถุที่มองเห็นในฉาก
Direct3DRMMaterial2	กำหนดว่าพื้นผิวจะสะท้อนแสงอย่างไร มีหลักการเช่นเดียวกับออบเจ็กต์ Direct3DRMMaterial แต่สามารถควบคุมคุณสมบัติของ material ได้ดียิ่งขึ้น
Direct3DRMMesh	ประกอบไปด้วยชุดของพื้นผิวหรือ face เราสามารถใช้ออบเจ็กต์นี้ในการจัดการกับกลุ่มของ faces และ vertices ทั้งหลาย
Direct3DRMMeshBuilder3	ทำให้เราสามารถทำงานกับแต่ละ face และ vertices ภายใน mesh ได้ เป็นส่วนขยายมาจากออบเจ็กต์ Direct3DRMMeshBuilder โดยเพิ่มการควบคุม normal ได้มากขึ้นและสามารถแก้ไข face บน mesh ได้ และยังมีส่วนที่

	เพิ่มเติมจากออบเจ็กต์ Direct3DRMMeshBuilder2 คือ การใช้อินเทอร์เฟซ DrawPrimitive ของโหมด Immediate เมื่อทำการสร้างโดยอินเทอร์เฟซ IDirect3DRM3
Direct3DRMObject	ใช้เป็นคลาสพื้นฐานสำหรับออบเจ็กต์อื่นๆ ในโหมด Retained ของ Direct3D ทั้งหมด โดยจะมีคุณลักษณะพื้นฐานร่วมกันของออบเจ็กต์ทั้งหมด
Direct3DRMObject2	เหมือนกับ Direct3DRMObject แต่พิเศษกว่าตรงที่อนุญาตให้ข้อมูลที่ใช้กับออบเจ็กต์มีความยืดหยุ่นได้มากขึ้น
Direct3DRMPickedArray	แสดงออบเจ็กต์ที่มองเห็นให้สอดคล้องกับจุดในระบบพิกัด 2 มิติที่กำหนดให้
Direct3DRMPicked2Array	แสดงออบเจ็กต์ที่มองเห็นให้สอดคล้องกับจุดตัดของรัศมีที่กำหนดให้
Direct3DRMProgressiveMesh	ประกอบด้วย mesh ที่มีลักษณะยาบรวมกันกับบันทึกที่อธิบายว่าจะเพิ่มเติม mesh ที่กำหนดไว้แล้วได้อย่างไร ซึ่งจะอนุญาตให้สามารถจัด mesh ในระดับทั่วไปได้ เช่นเดียวกับการดาวน์โหลด mesh จากแหล่งกำเนิดอื่นที่อยู่ไกลออกไปได้อย่างเป็นลำดับต่อเนื่องกัน
Direct3DRMShadow2	นิยามแสงเงา เช่นเดียวกับ Direct3DRMShadow ยกเว้นคุณสมบัติของแสงเงาสามารถติดตั้งและแก้ไขได้หลังจากทำการสร้างแล้ว
Direct3DRMTexture3	ประกอบด้วยอาเรย์รูปสี่เหลี่ยมผืนผ้าของสีในแต่ละ pixel เหมือนกับออบเจ็กต์ Direct3DRMTexture ยกเว้นทรัพยากรต่างๆ สามารถโหลดได้จากไฟอื่นนอกเหนือจากไฟล์ที่กำลังปฏิบัติการในขณะนั้น texture สามารถสร้างจากรูปภาพในหน่วยความจำ และเราสามารถทำการ mipmap (ลำดับของ texture ที่มีความละเอียดต่ำลงตามลำดับอย่างต่อเนื่อง เตรียมไว้สำหรับแสดงรูปภาพเดียวกัน รูปที่มีความละเอียดสูงใช้เมื่อออบเจ็กต์ที่มองเห็นอยู่ใกล้กับผู้มอง เมื่อออบเจ็กต์เคลื่อนที่ไกลออกไปและดูเหมือนมีขนาดเล็กลง ก็จะใช้รูปภาพที่มีความละเอียดต่ำลงตามลำดับ) ออบเจ็กต์นี้

	แตกต่างจากออบเจ็กต์ Direct3DRMTexture2 ตรงที่สามารถควบคุมการจัดการ texture ได้อย่างดียิ่งขึ้น
Direct3DRMUserVisual	จัดหาฟังก์ชันการทำงานที่ไม่มีในระบบ ซึ่งนิยามโดยแอปพลิเคชั่น
Direct3DRMViewport2	กำหนดว่าฉาก 3 มิติจะถูกเรนเดอร์ลงบนวินโดว์ 2 มิติได้อย่างไร เช่นเดียวกับออบเจ็กต์ Direct3DRMViewport แต่จะอนุญาตให้เจาะจงได้ว่าจะเคลียร์อะไรบ้าง
Direct3DRMVisual	ประกอบด้วยสิ่งต่างๆ ที่สามารถเรนเดอร์ลงบนฉาก ออบเจ็กต์นี้ไม่จำเป็นต้องมองเห็นก็ได้ เช่น เราสามารถเพิ่มเฟรมลงไปในฉาก แต่ไม่สามารถมองเห็นเฟรมนั้นได้
Direct3DRMWrap	คำนวณตำแหน่งบนระบบพิกัดของ texture สำหรับ face หรือ mesh

4.5 การนำ Direct3D Retained Mode มาใช้ในการพัฒนาเกม

4.5.1 การจัดแสง

การเปลี่ยนแปลงต่างๆ ของวัตถุไม่สามารถมองเห็นได้หากวัตถุนั้นอยู่ในความมืด การมองเห็นวัตถุจะเกิดขึ้นได้ก็ต่อเมื่อมีแหล่งกำเนิดแสงเท่านั้น ขณะที่เราเห็นการแสดงผลออกทางหน้าจอ วัตถุในฉากจะถูกเรนเดอร์ตามคุณภาพของแหล่งกำเนิดแสงที่อยู่ภายในฉาก

แหล่งกำเนิดแสงทุกชนิดมีคุณสมบัติพื้นฐานอย่างหนึ่งที่เหมือนกัน นั่นก็คือสีของแสงนั่นเอง โดยทั่วไปแหล่งกำเนิดแสงจะให้แสงสีขาว ซึ่งหมายถึงค่าความเข้มข้นของแสงสีต่างๆ (แสงสีแดง เขียว และน้ำเงิน) สูงสุด อย่างไรก็ตามเราสามารถกำหนดสีของแสงได้โดยการกำหนดค่าความเข้มข้นของแสงสีแดง เขียว และน้ำเงินให้กับแหล่งกำเนิดแสง โดยความเข้มข้นของแสงใน Direct3D มีค่าอยู่ในช่วง 0 จนถึง 1 เช่น แสงสีแดงมีค่าความเข้มแสง RGB เป็น 1,0,0 เราสามารถผสมแสงให้เกิดแสงสีต่างๆ เองได้ เช่น แสงสีเหลืองเกิดจากการผสมแสงสีแดงและเขียวเท่าๆ กันดังนั้นจึงมีค่าความเข้มแสง RGB เป็น 1,1,0 เป็นต้น

ชนิดของแล่งกำเนิดแสง

- แหล่งกำเนิดแสงประเภท Ambient
- แหล่งกำเนิดแสงประเภท Point
- แหล่งกำเนิดแสงประเภท Directional
- แหล่งกำเนิดแสงประเภท Spot Light

4.5.2 การเปลี่ยนแปลงส่วนสัดของรูปภาพ (Perspective Transformation)

การแปลงรูปแบบการมองเห็น 3 มิติเพื่อนำไปแสดงผลทางหน้าจอซึ่งเสมือนพื้นผิว 2 มิติ ทำได้โดยการแบ่งแยกเนื้อที่ 3 มิติเป็นรูปแบบที่สร้างเป็นภาพ 2 มิติได้ง่าย ซึ่งจะต้องพิจารณาว่าเนื้อที่ส่วนใดเป็นตำแหน่งของผู้มองหรือกล้อง และจากตำแหน่งนั้นจะมองเห็นส่วนใดในฉากบ้าง วิธีนี้ทำได้โดยการกำหนด "viewing frustum" ซึ่งเปรียบเหมือนฐานของปิรามิด โดยเปรียบผู้มองอยู่ ณ ตำแหน่งยอดของปิรามิด และมองผ่านภายในปิรามิดไปยังฐานของปิรามิด หากกำหนดขนาดของฐานปิรามิดให้ใหญ่ขึ้น ผู้มองก็จะเห็นเนื้อที่ภายในฉากกว้างมากขึ้น แต่เห็นวัตถุต่างๆ ขนาดเล็กลงกว่าเดิม ในทางกลับกันหากกำหนดฐานของปิรามิดให้มีขนาดเล็กลง วัตถุที่อยู่ในฉากจะมองดูใหญ่ขึ้น และเห็นวัตถุจำนวนน้อยลง การกำหนดขนาดของฐานปิรามิดเป็นการบ่งบอกถึงมุมมองของผู้ดู เรียกว่า FOV หรือ field-of-view

4.5.3 การทำ Z-buffering

สิ่งที่สำคัญในการแสดงกราฟิก 3 มิติ คือการพิจารณาว่าวัตถุใดอยู่ในตำแหน่งที่มองเห็นได้ และวัตถุใดถูกวัตถุอื่นบดบังอยู่ การเรนเดอร์ฉากในเกมโดยไม่คำนึงว่า surface ใดอยู่ใกล้กับผู้มองมากกว่า surface อื่น จะทำให้ภาพที่แสดงออกมามีความสับสน กรรมวิธีที่แก้ปัญหานี้เรียกว่า "Hidden Surface Removal Techniques" ซึ่ง Direct3D จะใช้เทคนิคอย่างหนึ่ง ซึ่งเรียกว่า Z-buffering

Z-buffering ใช้หน่วยความจำเป็นบัฟเฟอร์ในการเก็บข้อมูลว่าขณะที่แสดงผล พื้นใดอยู่ใกล้กับผู้มองมากที่สุด ค่า Z ที่เก็บในบัฟเฟอร์ไม่ได้แสดงความสัมพันธ์กับแกน Z แต่จะแสดงระยะห่างของ surface กับผู้มอง ในขณะที่วาดภาพ ค่า Z จะถูกเปรียบเทียบกับค่า Z ที่อยู่ใน Z-buffer หากค่าในบัฟเฟอร์บ่งบอกว่า surface ที่กำลังวาดนั้นอยู่ใกล้กับผู้มองมากกว่า surface อื่นๆ ก็จะวาด surface ใหม่นั้นทับ surface ที่มีอยู่เดิม แต่ถ้าหาก surface ใหม่ดังกล่าวอยู่ไกลกว่า surface เดิม ก็จะไม่ทำการวาด surface นั้นลงไป

การเรนเดอร์สามารถทำได้อย่างมีประสิทธิภาพโดยการจัดเรียง surface ตามลำดับหน้า-หลัง สาเหตุที่ทำให้ประสิทธิภาพเพิ่มขึ้นเนื่องจากวัตถุด้านหน้าสุดภายในฉากถูกวาดก่อน และไม่จำเป็นต้องวาด surface ที่ถูกบดบัง การที่โหมด Retained ของ Direct3D ทำ Z-buffering รวมถึงการเรียงลำดับให้ นับเป็นประโยชน์ต่อโปรแกรมเมอร์เป็นอย่างยิ่ง

Z-buffering นับเป็นเทคนิคอย่างหนึ่งใน Hidden Surface Removal Techniques ทั้งหลาย ที่มีความง่ายและความเร็วมากที่สุด และมีความละเอียดถึงขั้นระดับ pixel เลยทีเดียว (บางเทคนิคไม่ใช่) แต่ผลกระทบเมื่อใช้เทคนิคนี้ก็คือต้องการใช้หน่วยความจำในการทำ Z-buffer จำนวนมาก โดย Z-buffer ต้องมีขนาดอย่างน้อยเท่ากับขนาดของภาพผลลัพธ์ที่แสดง และสามารถมีความละเอียดได้ถึง 32 บิตเลยทีเดียว

4.5.4 การแสดงผลด้วนวิธีการเรนเดอร์แบบต่างๆ

เมื่อโมเดล 3 มิติภายในฉากถูกแปลงเป็น 2 มิติแล้วจึงถือว่าพร้อมที่จะทำการเรนเดอร์ ซึ่งเป็นขั้นตอนสุดท้ายในการสร้างภาพผลลัพธ์ที่จะแสดงให้เห็นออกทางหน้าจอ มีเทคนิคการเรนเดอร์มีอยู่หลายประเภท จะกล่าวถึงบางส่วนดังต่อไปนี้

Wireframe	วิธีนี้จะวาดเฉพาะขอบของ face ทั้งหมดที่ประกอบกันเป็นวัตถุ
Unlit	วาด face ทั้งหมดตามสีที่ถูกกำหนดไว้โดยไม่คำนึงถึงแสงและการหันเหของ face
Flat	แสดง face โดยพิจารณาตามแหล่งกำเนิดแสง โดยนำ normal มาใช้ในการคำนวณแสงที่ตกกระทบ
Gouraud	เพิ่มเงาให้กับ face โดยพิจารณาค่าความเข้มแสงเฉลี่ยที่กระทบบนพื้นผิว
Phong	ใช้ vertex normal ในการคำนวณแสงทั้งหมดที่ตกกระทบลงบนพื้นผิว
Ray-tracing	ให้การแสดงผลที่สมจริงมากที่สุด ซึ่งเทคนิคนี้จะคำนวณทั้งเงา แสงสะท้อน และการหักเหของแสงให้โดยอัตโนมัติ เทคนิคนี้ไม่เหมาะสำหรับใช้ในการแสดงผลการฟิกแบบเรียลไทม์และไม่สนับสนุนโดย Direct3D

4.5.5 การแสดงภาพเคลื่อนไหว (Animation)

การแสดงกราฟิก 3 มิติแบบเรียลไทม์จะขาดความสนุกสนานหากขาดการเคลื่อนไหว ซึ่งทำได้ 2 วิธี คือการกำหนดคุณสมบัติในการเคลื่อนไหวและการกำหนดเฟรมหลักในการเคลื่อนไหว

- การกำหนดคุณสมบัติในการเคลื่อนไหว (Motion Attributes)

เป็นวิธีที่ง่ายที่สุด ทำได้โดยการเปลี่ยนแปลงตำแหน่ง การหมุน หรือการเปลี่ยนแปลงขนาดวัตถุหรือกลุ่มของวัตถุเมื่อที่มีการ update ฉากทุกครั้ง

- การกำหนดเฟรมหลักในการเคลื่อนไหว (Key-framing)

เป็นวิธีดั้งเดิมในการแสดงภาพเคลื่อนไหว โดยการกำหนดเพียงเฟรมหลักบางส่วนในภาพเคลื่อนไหวทั้งหมด ในขณะที่เฟรมที่เหลือจะสร้างโดยคำนวณจากเส้นทางระหว่างเฟรมหลักที่ได้กำหนดไว้ เทคนิคนี้ก็คือการกำหนดตำแหน่งของวัตถุ ณ เวลาต่างๆ ในช่วงการเคลื่อนไหว แล้วให้คอมพิวเตอร์รับหน้าที่ในการจัดวางวัตถุในช่วงเวลาระหว่างนั้น

บทที่ 5

DirectSound

การโปรแกรมเสียงเป็นงานที่มีความยาก เนื่องจากต้องเข้าใจเสียง และเสียงดนตรี ซึ่งระบบเสียงที่เขียนขึ้นนั้นต้องทำงานได้ เมื่อนำไปใช้การ์ดเสียงทุกชนิด ในยุคแรกๆ นั้นนักโปรแกรมเมอร์เกมมักใช้ไลบรารีเสียงสำหรับระบบดอสและวินโดวส์จากผู้ผลิตรายที่สาม ซึ่งมีราคาสูงมาก และสำหรับระบบ DOS ไลบรารีเสียงทำงานได้ดี แต่กับระบบวินโดวส์ถึงแม้ว่าจะรองรับการเล่น (playback) เสียงและมัลติมีเดีย แต่ไม่รองรับการแสดงภาพเกม (video game) แบบ real-time ปัจจุบัน DirectSound ถูกสร้างขึ้นโดยสามารถแก้ไขปัญหาดังกล่าวได้ รวมถึงการรองรับการทำงานกับการ์ดเสียงได้มากถึงล้านตัว และการแสดงเสียงแบบ 3 มิติ

5.1 ประเภทของเสียงบนคอมพิวเตอร์

คอมพิวเตอร์สามารถสร้างเสียงได้ 2 ประเภท คือ เสียงดิจิตอล และเสียงสังเคราะห์

เสียงดิจิตอลเป็นเสียงเอฟเฟ็กต์ที่อัดได้จากการสั่นสะเทือนของเสียงต่างๆ เช่น เสียงระเบิด เสียงผู้คนสนทนากัน จากนั้นแปลงเป็นสัญญาณไฟฟ้า หรืออนาลอก และแปลงเป็นสัญญาณดิจิตอลโดยการแซมเปิล ซึ่งอัตราการแซมเปิล หมายถึง จำนวนครั้งที่แซมเปิลในช่วงเวลาหนึ่งวินาที และสามารถแซมเปิลเสียงได้จากความถี่และแอมปลิจูด การแซมเปิลด้วยความถี่จะต้องทำที่ความถี่ 2 เท่าของความถี่เสียงที่ต้องการอัด และการแซมเปิลด้วยแอมปลิจูดขนาด 8 บิตในปัจจุบันเพียงพอกับการนำไปใช้ในเกม และขนาด 16 บิตสำหรับการสร้างเสียงและดนตรีคุณภาพสูง

เสียงสังเคราะห์เป็นการโปรแกรมเสียง หรือนำเสียงมาสร้างใหม่ โดยเสียงที่ได้จะเป็นไปตามอัลกอริทึม และคุณภาพของฮาร์ดแวร์เสียง หรือกล่าวได้ว่าเสียงสังเคราะห์ใช้เฉพาะสำหรับเป็นเสียงดนตรีเท่านั้น ซึ่งเก็บในรูปแบบไฟล์ MIDI (Musical Instrument Digital Interface) ซึ่งเสียงที่ได้จะดูเรียบง่ายเกินไปถ้าเทียบกับเสียงธรรมชาติ และขาดความรู้สึกที่จะได้รับจากเสียงดิจิตอล และต้องอาศัยเทคนิคขั้นสูงในการสร้างเสียงสังเคราะห์ให้ใกล้เคียงเสียงดิจิตอล ได้แก่ wave table และ wave guide

5.2 ฮาร์ดแวร์เสียง

Creative Labs Sound Blasters เป็นการ์ดเสียงที่คุณภาพเด่นที่สุดเหนือการ์ดเสียงอื่นๆ ซึ่งมีเทคนิคการสังเคราะห์เสียงขั้นสูงทั้งสองเทคนิค คือ

- เทคนิค wave table มีชิป DSP (Digital Signal Processor) ซึ่งแซมเปิลได้เสียงดิจิตอล และนำมาเล่นอีกครั้งที่ความถี่และแอมปลิจูดที่ต้องการ ซึ่งการทำกระบวนการนี้ ต้องมีหน่วยความจำในการเก็บเสียง
- เทคนิค wave guide มีชิป DSP (Digital Signal Processor) และฮาร์ดแวร์เสียงคุณภาพดี ซึ่งสามารถสร้างโมเดลของเครื่องดนตรีเสมือนได้ และเล่นเสียง

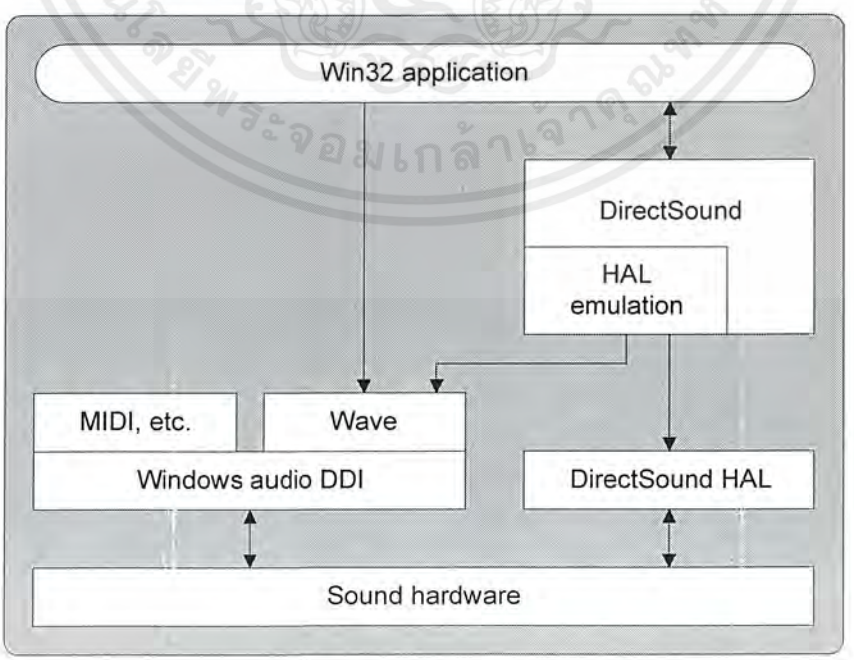
DirectSound จะเล่นเฉพาะเสียงดิจิตอล ดังนั้นจึงไม่รองรับการเล่นเสียง MIDI ทำให้ในการเขียนโปรแกรมสร้างเกมไม่นำเสียงแบบ MIDI มาใช้ เนื่องจากเมื่อใช้การ์ดเสียงต่างชนิดกันทำให้เกิดปัญหามาก และควบคุมคุณภาพของผลลัพธ์เสียงได้ยาก แต่ใน DirectX 7.0 รองรับการเล่นเสียงแบบ MIDI และในโครงงานนี้จะกล่าวถึงเฉพาะ DirectX 6.0 ซึ่งเป็นเวอร์ชันที่รองรับการเล่นเสียงแบบ wave เท่านั้น

5.3 รู้จักกับ DirectSound

DirectSound ประกอบด้วยจำนวนคอมโพเนนท์ หรืออินเทอร์เฟซเช่นเดียวกับ DirectDraw สำหรับกรณีที่มีการ์ดเสียง สามารถใช้งานผ่านไดรเวอร์ DirectSound และถ้าไม่มีฮาร์ดแวร์เสียง DirectSound สามารถทำงานได้โดย HEL หรือการจำลองฮาร์ดแวร์ และวินโดวส์ DDI (Device Driver Interface) โดยการรวมผลงานเสียงที่สร้างเข้ากับ DirectSound.DLL และการนำ DirectSound มาใช้งานต้องสร้างออบเจ็กต์ DirectSound COM (Component Object Model) และร้องขออินเทอร์เฟซจากออบเจ็กต์หลัก ซึ่งโครงงานไม่ขอกล่าวถึง DirectSound3D และอินเทอร์เฟซ DirectSoundCapture ที่เพิ่มขึ้นในระบบเสียง 3 มิติ

Win32 API บนวินโดวส์มีฟังก์ชันสร้างเสียงให้กับแอปพลิเคชั่นได้ระดับหนึ่ง แต่ในระบบของเกมที่แท้จริงต้องการ tool ที่มีประสิทธิภาพมากกว่านั้น โดย DirectSound ได้เสนอสิ่งต่างๆ เหล่านี้ได้แก่

- การเร่งความเร็ว โดยอัตโนมัติ ถ้ามีฮาร์ดแวร์เหล่านั้นอยู่
- การรวมเสียงต่างๆ เข้าด้วยกัน อย่างไม่จำกัด
- การกำหนดตำแหน่งของเสียง 3 มิติ โดย Direct3D
- การแปลงข้อมูลอินพุตของเสียง ในรูปแบบต่างๆ หลายรูปแบบ ให้ได้เอาท์พุตที่ตรงกัน
- รองรับกำหนดคุณสมบัติใหม่ๆที่มีเข้ามาของฮาร์ดแวร์ โดยไม่ต้องเปลี่ยน API
- การเล่นข้อมูลเสียงใช้ทรัพยากรต่ำ

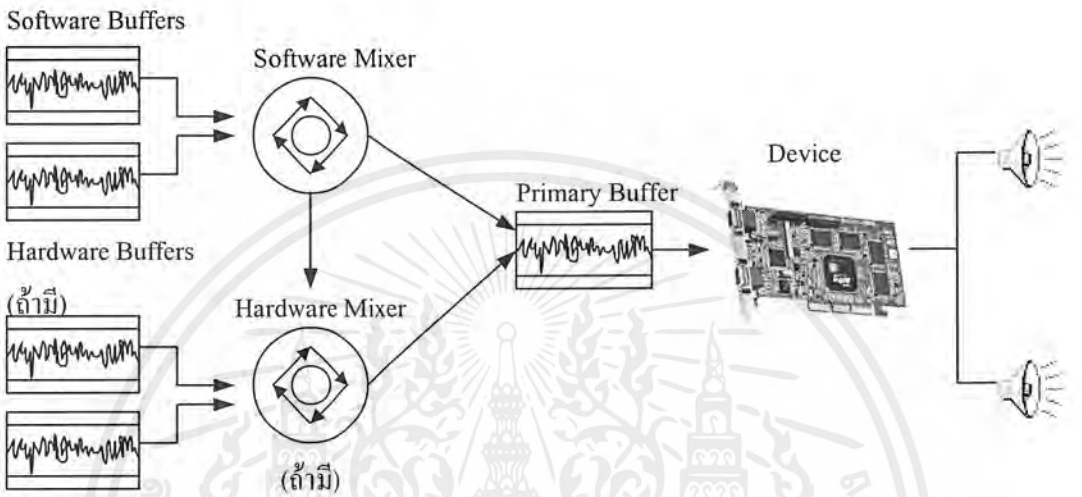


รูปที่ 5-1 สถาปัตยกรรมของ DirectSound

การเรียกใช้ DirectSound ประกอบด้วย 3 ส่วน

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

5.4 การทำงานของ DirectSound



รูปที่ 5-2 การทำงานของ DirectSound

DirectSound จะเริ่มด้วยออบเจ็กต์บัฟเฟอร์เสียง secondary ซึ่งแทนเสียงเพียงเสียงเดียว เสียงเหล่านี้อาจเป็น static sound หรือ streaming sound ก็ได้

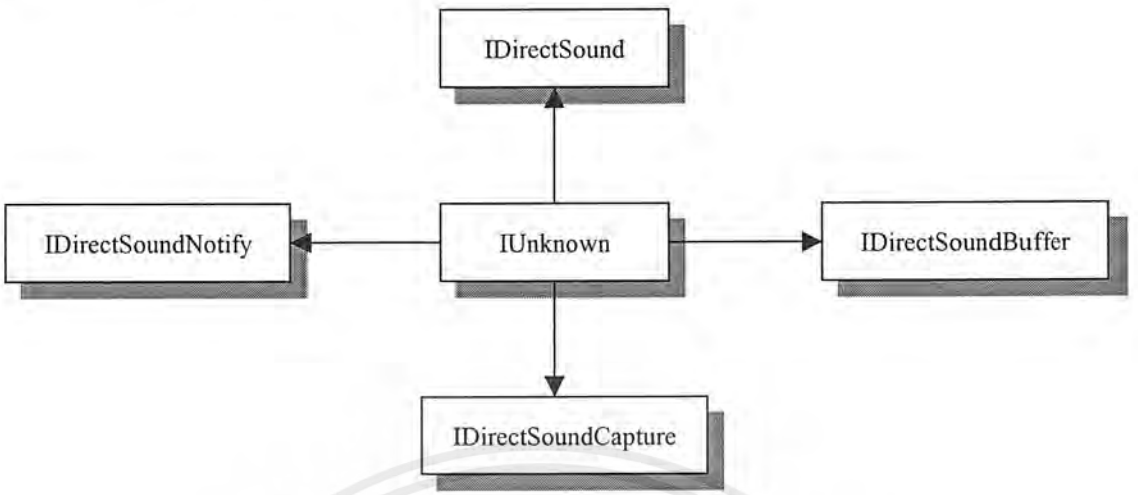
Static sound คือ เสียงสั้นๆ ที่มีขนาดข้อมูลที่พอดีกับหน่วยความจำ

Streaming sound คือ ข้อมูลเสียงส่วนหนึ่ง ที่ย้ายจากหน่วยความจำมาเก็บไว้ในบัฟเฟอร์ ช่วงขณะหนึ่ง โดยบัฟเฟอร์ทั้งหมดจะมีรูปแบบการแซมเปิลสัญญาณเสียงแบบ PCM (Pulse Code Modulate)

DirectSound จะนำข้อมูลจากบัฟเฟอร์ secondary แต่ละอันมาผสานเข้าด้วยกันในบัฟเฟอร์ primary โดยจะทำการแปลงรูปแบบของข้อมูลที่จำเป็น เช่น แปลงอัตราการแซมเปิล และประยุกต์ให้มีเสียงเอฟเฟ็กต์พิเศษเพิ่มเติม เช่น การจัดตำแหน่งของเสียงในระบบ 3 มิติ

ถ้าภายในฮาร์ดแวร์มีหน่วยความจำและตัวผสานเสียง DirectSound จะนำบัฟเฟอร์ที่สร้างขึ้นไปเก็บในหน่วยความจำของฮาร์ดแวร์โดยอัตโนมัติให้ได้มากที่สุดก่อน และถ้าไม่มีหรือมีไม่เพียงพอ ก็จะนำบัฟเฟอร์ไปเก็บในหน่วยความจำของระบบ หรือบัฟเฟอร์ซอฟต์แวร์ ซึ่ง DirectSound จะผสานเสียงโดยใช้ซอฟต์แวร์ที่จำลองตัวขึ้นเป็นตัวผสานเสียง และส่งไปยังตัวผสานเสียงบนฮาร์ดแวร์ เพื่อรวมกับเสียงบนบัฟเฟอร์ฮาร์ดแวร์ และส่งไปยังบัฟเฟอร์ primary

5.5 การติดต่อของ DirectSound



รูปที่ 5-3 การติดต่อของ DirectSound

Iunknown : คือ ออบเจ็กต์ COM หลัก

IdirectSound : คือ ออบเจ็กต์ COM หลักของ DirectSound ใช้แทนฮาร์ดแวร์เสียงของมันเอง การ์ดเสียง 1 อัน แทนออบเจ็กต์ 1 ออบเจ็กต์

IdirectSoundBuffer : คือ ฮาร์ดแวร์ผสาน (mix) เสียง และเก็บข้อมูลเสียง DirectSound มีบัฟเฟอร์อยู่ 2 ชนิด คือ บัฟเฟอร์ primary เป็นเสียงที่กำลังเล่นอยู่ และผสานเข้าด้วยกันโดยฮาร์ดแวร์(ถ้ามี) หรือซอฟต์แวร์ ต้องเก็บไว้ในหน่วยความจำของฮาร์ดแวร์เท่านั้น และบัฟเฟอร์ secondary แทนเสียงที่ถูกเก็บไว้เพื่อจะเล่น อาจเก็บไว้ในหน่วยความจำระบบ หรือใน SRAM (Sound RAM) บนการ์ดเสียง

IdirectSoundCapture : เป็นออบเจ็กต์ COM ของ DirectX ซึ่งทำการบันทึกและจับ (capture) ข้อมูลเสียงเข้ามา เช่น voice-recognition เป็นออบเจ็กต์ที่ทำงานอิสระไม่ขึ้นกับ DirectSound และมีอินเทอร์เฟซระดับบนเป็นของตัวเอง หรือ IdirectSoundCapture ซึ่งมีส่วนที่ครอบคลุม COM (COM Wrapper) อีกทีหนึ่ง เพื่อให้สามารถใช้งานฟังก์ชันของ waveform API บน Win32 ดังนั้นระบบปฏิบัติการวินโดวส์ 98 และวินโดวส์ NT จะมีไดรเวอร์ Win32 Driver Model (WDM) ซึ่งสนับสนุนการเร่งความเร็วของฮาร์ดแวร์ ดังนั้นในโครงงานนี้จะขอกล่าวถึงแต่การนำ DirectSound มาใช้ ซึ่งในทางกายภาพ DirectSoundCapture มีส่วนคล้ายกับ DirectSound แต่ในรายละเอียดนั้นแตกต่างกัน

IdirectSoundNotify : การส่ง message กลับไป ให้ DirectSound จำเป็นสำหรับระบบเสียงที่มีความซับซ้อน

5.6 รูปแบบเสียงใน DirectSound (โครงสร้าง WAVEFORMATEX)

- nChannels (WORD) : จำนวน channel (1 คือ mono และ 2 คือ stereo)
- nSamplesPerSec (DWORD) : อัตราแซมเปิลความถี่ มีหน่วยเป็นเฮิรตซ์ หรือจำนวนครั้งที่แซมเปิลในช่วงเวลา 1 วินาที สำหรับ 1 channel ดังนั้นเสียงแบบ stereo จะมีอัตราการแซมเปิลเป็น 2 เท่า ซึ่งใน

การแซมเปิลจะแทรก channel ทางซ้ายก่อน โดยทั่วไปฮาร์ดแวร์สำหรับพีซีจะรองรับอัตราการแซมเปิลขนาด 11025 22050 และ 44100 เฮิรตซ์

- wBitsPerSample (WORD) : เป็นจำนวนบิตสำหรับการแซมเปิลแอมปลิจูด มักใช้ 8/16 บิต
- wFormatTag (WORD) : เป็นตัวระบุว่าข้อมูลจะแปลงไปอย่างไร สำหรับ DirectSound จะใช้ WAVE_FORMAT_PCMtag ซึ่งเท่ากับ 1
- nBlockAlign (WORD) : เป็นจำนวนไบต์ในการแซมเปิลหนึ่งครั้งหรือ nChannel * wBitsPerSample/8
- nAvgBytesPerSec (WORD) : อัตราการส่งข้อมูลเฉลี่ย หรือ nSamplesPerSec * nBlockAlign

5.7 การเข้าถึง DirectSound

การเข้าถึงระบบ DirectSound ต้องทำตามขั้นตอนต่อไปนี้

1. รับ GUID สำหรับอุปกรณ์ด้านเสียง
2. สร้างออบเจ็กต์ของ DirectSound (จำเป็น)
3. กำหนด Cooperative level (จำเป็น)
4. กำหนดรูปแบบของบัฟเฟอร์ primary

5.7.1 การกำหนดอุปกรณ์เสียง (Enumeration)

DirectSound เป็นออบเจ็กต์ COM ของ DirectX ตัวหนึ่ง ดังนั้นจะมีฟังก์ชันที่ระบุว่าจะเลือกใช้ฮาร์ดแวร์ตัวใดจากไดรเวอร์ที่มีอยู่ และส่งข้อมูลของฮาร์ดแวร์นั้นกลับไป ซึ่งการกำหนดอุปกรณ์ไม่จำเป็นต้องทำได้ เพราะในการเริ่มต้นสร้างออบเจ็กต์ DirectSound ระบบจะเลือกใช้อุปกรณ์ด้านเสียงที่เหมาะสมมากกว่าอยู่แล้ว หรือเราสามารถส่งค่า NULL ไปให้ DirectSound เพื่อกำหนดให้อุปกรณ์ของเราเป็น default ได้เช่นกัน หรือผู้ใช้สามารถเลือกอุปกรณ์ที่เหมาะสมเองได้จากรายการที่ระบบมีให้เลือก

ก่อนที่จะสร้างออบเจ็กต์ DirectSound ขึ้น เราจะเรียกฟังก์ชัน DirectSoundEnumerate ซึ่งจะต้องประสานงานร่วมกับอุปกรณ์ที่ได้ระบุไว้ และฟังก์ชันดังกล่าวประกอบด้วยตัวแปร 2 ตัว คือ

- lpDSEnumCallback (LPDSENUMCALLBACK) : เป็นพอยทเตอร์ชี้ไปที่แอดแดสของฟังก์ชัน Callback ของอุปกรณ์ที่เลือก
- lpContex (LPVOID) : เป็นพอยทเตอร์ที่ชี้ไปยังข้อมูลขนาด 32 บิต ซึ่งต้องการส่งหรือรับจากฟังก์ชัน Callback ของอุปกรณ์ที่เลือก

หลักการ : เมื่อเรียกใช้ฟังก์ชัน DirectSoundEnumerate และ DirectSound จะสร้างรายการของอุปกรณ์เสียงที่สามารถเลือกใช้งานได้ในระบบ ส่งข้อมูลเสียงของแต่ละอุปกรณ์กลับไปยังฟังก์ชัน Callback ของแอปพลิเคชั่น และฟังก์ชัน DSEnumCallback จะต้องประกาศเป็น "BOOL CALLBACK" ซึ่งประกอบด้วยตัวแปร 4 ตัว คือ

- lpGuid (LPGUID) : พอยทเตอร์ชี้ไป GUID ของอุปกรณ์
- lpstrDescription (LPCSTR) : เป็นชื่อ Friendly ของอุปกรณ์
- lpcstrModule (LPCSTR) : เป็นชื่อโมดูล

- lpContext (LPVOID) : ข้อมูลเสียง

โดย DirectSound จะรับค่าจากตัวแปร 3 ตัวแรกจากไดรเวอร์ของอุปกรณ์ ส่วน lpContext เป็นค่าเดียวกันกับที่ต้องส่งไปให้ฟังก์ชัน DirectSoundEnumerate และฟังก์ชัน DSEnumCallback จะคืนค่าเป็น true/false เพื่อบอกให้ดำเนิน/สิ้นสุดการกำหนดอุปกรณ์ แต่อย่างไรก็ตาม สามารถเรียกใช้เมธอด GetCaps เพื่อตรวจสอบความสามารถของอุปกรณ์ (ภายในฟังก์ชัน Callback) ในขณะที่เริ่มต้นสร้างออบเจ็กต์ DirectSound ได้เช่นกัน แล้วจึงเลือกที่จะปลดอุปกรณ์และทำการกำหนดอุปกรณ์ครั้งใหม่ต่อไป (โดยการคืนค่าฟังก์ชันเป็น true) หรือเก็บอุปกรณ์ไว้และหยุดการกำหนดอุปกรณ์ (คืนค่าเป็น false)

ตัวอย่างการกำหนดอุปกรณ์เสียงที่เหมาะสมมากกว่าโดยผู้ใช้ ทำได้โดยเข้าไปใน คอนโทรลพาแนล>มัลติมีเดียแอพเพท และเลือกจากรายการ ตัวอย่างเช่น การกำหนดอุปกรณ์เสียงตัวแรก คือ ไดรเวอร์เสียง primary ลงในรายการอุปกรณ์เท่านั้น และถ้าผู้ใช้เลือกสตริงนี้ แอปพลิเคชั่นจะรับผิดชอบในการสร้างออบเจ็กต์ DirectSound สำหรับอุปกรณ์ที่ถูก default สำหรับบนระบบที่มีการ์ดเสียง DirectSoundEnumerate จะเรียกฟังก์ชัน Callback อย่างน้อย 2 ครั้ง

- การกำหนดอุปกรณ์เป็นไดรเวอร์เสียง primary
- ฟังก์ชัน Callback รับ GUID สำหรับฮาร์ดแวร์เพื่อสร้างอุปกรณ์เสียงที่ default ไว้

ฟังก์ชัน Callback ส่งค่าผ่านตัวแปร lpcstrDescription ไว้ในรายการ และเก็บ GUID สำหรับอุปกรณ์นั้นไว้ในรูปแบบรายการข้อมูลจากสตริง default เมื่อผู้ใช้เลือกอุปกรณ์ ใช้พอยทเตอร์เรียกฟังก์ชัน DirectSoundCreate ต่อไป

5.7.2 การสร้างออบเจ็กต์ DirectSound

ฟังก์ชัน DirectSoundCreate มีหน้าที่ในการสร้างออบเจ็กต์ DirectSound โดยรับค่า GUID ของอุปกรณ์ตัวอย่างในรายการ หรือค่า NULL ในกรณีที่เลือกอุปกรณ์เอง ซึ่งใช้พารามิเตอร์ LPGUID lpGuid เป็นพอยทเตอร์ไปยังค่าดังกล่าว หลังจากการกำหนดอุปกรณ์

ข้อควรระวัง: การสร้างออบเจ็กต์ DirectSound สามารถสร้างหลายๆ ออบเจ็กต์ได้ แต่ปัญหาคือ เมื่อกำหนด Cooperative level ในออบเจ็กต์ใหม่ ค่า cooperative level จะไปทับของเดิม

ตัวอย่าง:

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

เพิ่มเติม: DS_OK หรือ DirectSound OK และไม่ว่าจะทำอะไรกับออบเจ็กต์ DirectSound ต้องทำการ release ออบเจ็กต์ทุกครั้ง เพื่อลบพื้นที่บนหน่วยความจำซึ่งจองไว้สำหรับออบเจ็กต์

5.7.3 การกำหนดระดับ Cooperative

เมื่อสร้างออบเจ็กต์ DirectSound จะต้องกำหนดระดับ cooperative เพื่อให้ระบบวินโดวส์จัดการการทำงานของออบเจ็กต์ และการแบ่ง (share) อุปกรณ์เสียงให้กับแอปพลิเคชั่น และงานอื่นๆบนระบบ และกำหนดขีดความสามารถของแอปพลิเคชั่นในการควบคุมฮาร์ดแวร์เสียง

การกำหนดระดับ cooperative แบ่งเป็น 2 กลุ่มหลัก คือ การกำหนดซึ่งทำให้นักพัฒนาเกมสามารถ/ไม่สามารถควบคุมตัวผสานเสียงของฮาร์ดแวร์ หรือบัฟเฟอร์เสียง ระบบวินโดว์ส

อินเทอร์เฟซ IDirectSound ใช้เมธอด SetCooperativeLevel ( ) ในการกำหนดระดับ Cooperative ซึ่งมีพารามิเตอร์ 2 ค่า คือ

- HWND (hwnd) : เป็นส่วนจัดการวินโดวส์
- Dwlevel (DWORD) : เป็นการกำหนดระดับ cooperative ซึ่งได้แก่
    - DSSCL_NORMAL : เป็นระดับที่ดีที่สุดสำหรับแอปพลิเคชั่นทุกแอปพลิเคชั่น คือ สามารถเล่นเสียงได้ แต่ไม่ได้รับอนุญาตให้โปรแกรม และเปลี่ยนรูปแบบบัฟเฟอร์ primary ซึ่ง DirectSound จะสร้างบัฟเฟอร์ primary เป็น default ให้เลือก ตัวอย่างเช่น มีอัตราการแซมเปิลด้วยความถี่ 22 กิโลเฮิรตซ์ แอมปลิจูดขนาด 8 บิต และมี 2 channel หรือ stereo
    - DSSCL_PRIORITY : เป็นระดับที่เข้าถึงฮาร์ดแวร์เสียงได้ทุกอุปกรณ์ ดังนั้นนักพัฒนาสามารถโปรแกรม และเปลี่ยนรูปแบบตัวผสานเสียง primary ได้ และยังสามารถร้องขอให้ฮาร์ดแวร์เสียงปฏิบัติการระดับสูงบนหน่วยความจำได้ เช่น การบีบอัดข้อมูล ในกรณีที่ต้องการเปลี่ยนรูปแบบข้อมูลของบัฟเฟอร์ primary
    - DSSCL_EXCLUSIVE : เป็นระดับที่มีลักษณะเช่นเดียวกับ priority แต่จะไม่แสดงแอปพลิเคชั่นที่เป็น background หรือเมื่อ minimize แอปพลิเคชั่น
    - DSSCL_WRITEPRIMARY : เป็นระดับ priority สูงที่สุด ซึ่งเข้าถึงและโปรแกรมตัวผสานเสียง หรือบัฟเฟอร์ primary ดังนั้นต้องมีไดรเวอร์ DirectSound ซึ่งการกำหนดระดับนี้จะไม่สร้างและใช้บัฟเฟอร์ secondary และแอปพลิเคชั่นอื่นๆ

ตัวอย่าง:

```
// การกำหนดระดับ cooperation
if (lpds->SetCooperativeLevel(main_window_handle,DSSCL_NORMAL)!=DS_OK)
  { /* error setting cooperative level */ }
```

เพิ่มเติม : ถ้าเกิดความผิดพลาดใดๆ แอปพลิเคชั่นอื่นจะได้ใช้และควบคุมการ์ดเสียงแทน

5.7.4 การกำหนดรูปแบบบัฟเฟอร์ pimary

บัฟเฟอร์ primary เป็นฮาร์ดแวร์ผสานเสียง (ตัวผสานเสียงจำลอง) บนการ์ดเสียง และทำงานตลอดการเล่นเสียง จาก default DirectSound จะสร้างบัฟเฟอร์ primary ที่มีอัตราการแซมเปิลด้วยความถี่ 22,050 กิโลเฮิรตซ์ แอมปลิจูดขนาด 8 บิต และ stereo (2 channel) หรือการกำหนดระดับ cooperative เป็น normal และถ้าต้องการเปลี่ยนรูปแบบบัฟเฟอร์ primary เพื่อให้แอปพลิเคชั่นเล่นเสียงได้อย่างมีคุณภาพ

มากขึ้น ตัวอย่างเช่น ให้มีอัตราการแซมเปิลแอมปลิจูดขนาด 16 บิต (เป็น default ซึ่ง DirectSound ทำงานได้ดีที่สุด) สามารถทำได้เมื่อกำหนดระดับ cooperative เป็น priority สูงที่สุด หรือระดับ DSSCL_PRIORITY เป็นต้นไป และรูปแบบบัฟเฟอร์ primary ต้องเข้ากันได้กับบัฟเฟอร์ secondary

5.7.5 เปลี่ยนการจัดวางและคุณสมบัติของลำโพง (Speaker)

ถ้าแอปพลิเคชั่นใช้ระบบเสียง 3 มิติ นักพัฒนาอาจต้องเปลี่ยนการจัดวางและคุณสมบัติของลำโพง เพื่อให้เข้ากับอุปกรณ์จริง เมื่อนำแอปพลิเคชั่นไปใช้งาน ซึ่งจะทำให้ลำโพงแสดงเสียงเอฟเฟ็กต์ในระบบ 3 มิติได้ดีที่สุด และการทำเช่นนี้เป็นหน้าที่ของเมธอด GetSpeakerConfig ซึ่งประกอบด้วยพารามิเตอร์ 1 ค่าคือ dwSpeakerConfig (DWORD) ซึ่งเป็นค่าคุณสมบัติของลำโพง ได้แก่ DSSPEAKER_HEADPHONE หรือหูฟัง DSSPEAKER_MONO หรือ mono DSSPEAKER_QUAD DSSPEAKER_SURROUND DSSPEAKER_STEREO หรือ stereo และถ้าเป็นลำโพงคู่ ต้องกำหนดมุมในการจัดวางลำโพง เช่น

- DSSPEAKER_GEOMETRY_MIN : (5 องศา) หมายถึง การวางลำโพงให้อยู่ห่างจากตำแหน่งหูผู้ฟัง 95 องศา หรือห่างจากตำแหน่งที่เป็นค่า minimum (90 องศาห่างจากตำแหน่งหูผู้ฟัง) 5 องศา
- DSSPEAKER_GEOMETRY_NARROW : (10 องศา)
- DSSPEAKER_GEOMETRY_WIDE : (20 องศา)
- DSSPEAKER_GEOMETRY_MAX : (180 องศา)

และเมธอดคืนค่าที่เป็น default หรือค่าสุดท้ายที่ได้กำหนดไว้ ดังนั้นเป็นหน้าที่ของนักพัฒนาที่จะต้องทราบสิ่งที่ผู้ใช้ชอบเมื่อนำแอปพลิเคชั่นที่พัฒนาไปใช้งานจริง เพื่อนำมากำหนดคุณสมบัติและการจัดวางลำโพง

ตัวอย่าง:

```
lpds -> SetSpeakerConfig (DSS_SPEAKER_COMBINED(
                    DSS_SPEAKER_STEREO, DSS_GEOMETRY_WIDE));
                    /*ห่างจากตำแหน่งที่เป็นค่า minimum (90 องศาห่างจากตำแหน่งหูผู้
                    ฟัง) 20 องศา*/
```

5.7.6 การประเมินความสามารถของฮาร์ดแวร์

เป็นลักษณะเฉพาะของออบเจ็กต์ COM ของ DirectX ดังนั้น DirectSound มีเมธอด GetCaps ซึ่งใช้ในการค้นหาทรัพยากรฮาร์ดแวร์ที่แอปพลิเคชั่นสามารถดึงมาใช้งานได้ ทำให้นักพัฒนาสามารถจัดการกับบัฟเฟอร์เสียงได้ดีที่สุด ซึ่งเหมาะสำหรับกรณีที่แอปพลิเคชั่นมีการเล่นเสียงโดยใช้เนื้อที่บนหน่วยความจำมาก และใช้ซีพียูทำกระบวนการทางกราฟิกมากถึงขีดสุด นักพัฒนาควรมีการจัดการกับบัฟเฟอร์เสียงอย่างระมัดระวังให้พอเหมาะกับฮาร์ดแวร์ที่มีให้ใช้งาน

5.8 การกำหนดคุณสมบัติ (property)

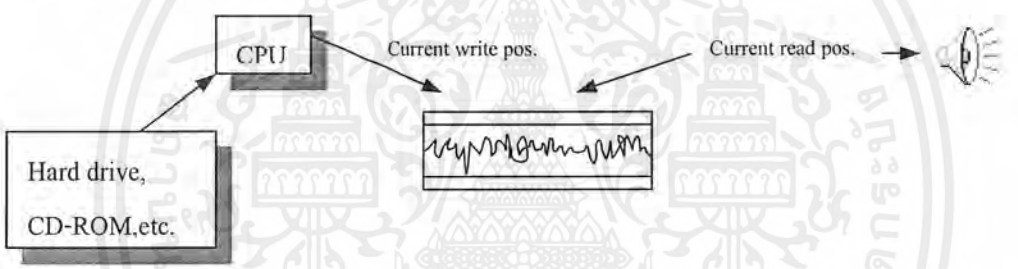
มีไว้เพื่อรองรับคุณลักษณะ (feature) ใหม่ๆ ของการ์ดเสียง โดย GUID จะรวมคุณสมบัติเหล่านี้ไว้ และแต่ละคุณสมบัติจะมีหมายเลข index ของมันเอง ผู้ผลิตก็จะออกไฟล์ส่วนหัวที่มีค่าเหล่านี้ออกมา เมื่อต้องการใช้ก็ไปร้องขอมา

5.9 การเล่น DirectSound

ในส่วนต่อไปนี้กล่าวถึงการสร้างบัฟเฟอร์ secondary การอ่านข้อมูลมาใส่ในบัฟเฟอร์ รวมถึงการเล่นเสียงในบัฟเฟอร์

จากที่ผ่านมา กล่าวถึงการควบคุมออบเจ็กต์เสียงขั้นพื้นฐาน หรือบัฟเฟอร์ โดยใช้อินเทอร์เฟซ IdirectSoundBuffer และการนำอินเทอร์เฟซมาใช้เปลี่ยนรูปแบบเอาท์พุต หรือทำการผสานเสียงในบัฟเฟอร์ primary แต่ส่วนนี้กล่าวถึงหน่วยความจำของระบบซึ่งนำมาใช้เป็นบัฟเฟอร์ Secondary

5.9.1 บัฟเฟอร์ secondary

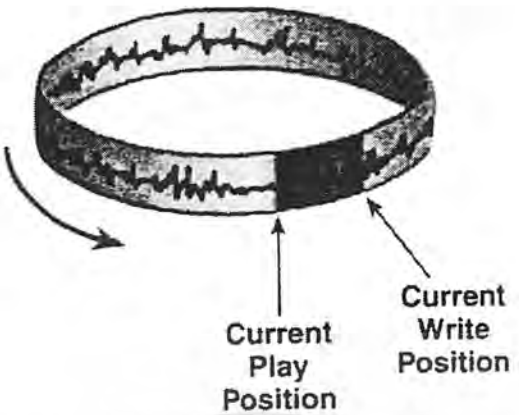


รูปที่ 5-4 Streaming audio data

เป็นบัฟเฟอร์ที่เก็บเสียงที่จะใช้เล่น โดยส่วนมากเก็บข้อมูลเสียงบนหน่วยความจำบนการ์ดเสียง เนื่องจากการเพิ่มหน่วยความจำบนระบบมีราคาสูง และการเก็บข้อมูลเสียงบนการ์ดใช้ไฟในกระบวนการเล่นเสียงน้อยกว่ามาก ซึ่งบัฟเฟอร์ 2 ประเภท คือ

- บัฟเฟอร์เสียงแบบ static : มีขนาดเท่าใด สามารถเก็บข้อมูลเสียงได้เท่านั้น และเล่นเสียงแบบวนไปเรื่อยๆ
- บัฟเฟอร์เสียงแบบ streaming : เหมาะกับการเล่นเสียงจากซีดีรอม ซึ่งขนาดบัฟเฟอร์ไม่เพียงพอต่อการเก็บข้อมูลเสียงที่มีขนาดใหญ่ หลักการคือ อ่านข้อมูลเสียงไปไว้ใน chunk (ขนาดของข้อมูลที่อ่านได้ต่อครั้ง) และ stream ออก

5.9.1.1 ตำแหน่งของการเขียน และอ่านข้อมูลบนบัฟเฟอร์ streaming



รูปที่ 5-5 Secondary buffer with current play and write positions.

บัฟเฟอร์ streaming เรียกอีกอย่างหนึ่งว่า บัฟเฟอร์ circular หลักการ คือ การหมุนอาร์เรย์ที่เก็บข้อมูลเสียงในขณะเล่นเสียงอยู่ด้วยการบันทึกข้อมูลเสียงใหม่ทับส่วนที่เล่นแล้วไปด้วย ดังนั้นจึงจำเป็นที่จะต้องมีพอยทเตอร์ไว้ 2 ตัว DirectSound จะเก็บพอยทเตอร์ของบัฟเฟอร์เสียงทั้งหมดไว้ คือ ตำแหน่งที่เล่น (current play position) และ ตำแหน่งที่อ่าน (current write position)

<u>เพิ่มเติม</u> :

- Current play position คือ ค่า offset ในบัฟเฟอร์เพื่อให้ DirectSound นำข้อมูลตั้งแต่นั้น ไปเป็นข้อมูลในการผสานเสียง เพื่อเล่นเสียง
- Current write position คือ ค่า offset ในบัฟเฟอร์เพื่อให้ DirectSound นำข้อมูลตั้งแต่นั้น ไปเป็นข้อมูลในการบันทึกเสียงลงไป ตั้งแต่ตำแหน่งนี้เป็นต้นไป เป็นตำแหน่งที่ปลอดภัยที่จะเขียนข้อมูลลงบัฟเฟอร์ ซึ่งในโครงงานไม่ได้นำบัฟเฟอร์ประเภทนี้มาใช้งาน ดังนั้นไม่ขอกล่าวในรายละเอียด

5.9.2 การทำให้บัฟเฟอร์ที่ใช้ให้มีประสิทธิภาพ

5.9.2.1 บัฟเฟอร์ secondary แบบ static และ streaming

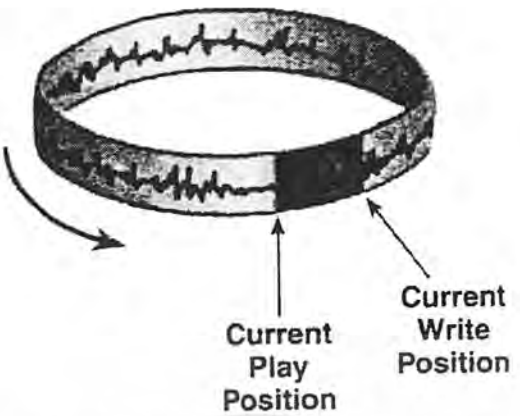
บัฟเฟอร์ static ควรจะเป็นเสียงที่มีความยาวสั้นๆ เสียงเดียว ที่ต้องเล่นซ้ำไปมา ส่วนบัฟเฟอร์ streaming จะเหมาะกับข้อมูลเสียงที่ขนาดไม่พอกับบัฟเฟอร์ เนื่องจากขนาดใหญ่ไปต้อง คัดลอกเป็นส่วนๆ ไปเก็บไว้ในบัฟเฟอร์ในขณะที่เล่นเสียงไปด้วย

โดย default เมธอด CreateSoundBuffer จะสร้างบัฟเฟอร์ streaming แต่ถ้าต้องการสร้างบัฟเฟอร์ static ต้องกำหนด flag เป็น DSBCAPS_STATIC

5.9.2.2 ฮาร์ดแวร์และซอฟต์แวร์

การ์ดเสียงบางรุ่นมีความสามารถในการผสานเสียงอยู่บนการ์ดแล้ว และ DirectSound สามารถสร้างบัฟเฟอร์ secondary บนฮาร์ดแวร์ หรือสามารถใช้หน่วยความจำจากการ์ดเสียงได้ แต่ถ้าไม่มีก็จะใช้

5.9.1.1 ตำแหน่งของการเขียน และอ่านข้อมูลบนบัฟเฟอร์ streaming



รูปที่ 5-5 ตำแหน่งการเล่นและเขียนข้อมูลเสียงบนบัฟเฟอร์ secondary

บัฟเฟอร์ streaming เรียกอีกอย่างหนึ่งว่า บัฟเฟอร์ circular หลักการ คือ การหมุนอาร์เรย์ที่เก็บข้อมูลเสียงในขณะเล่นเสียงอยู่ด้วยการบันทึกข้อมูลเสียงใหม่ทับส่วนที่เล่นแล้วไปด้วย ดังนั้นจึงจำเป็นที่จะต้องมีพอยทเตอร์ไว้ 2 ตัว DirectSound จะเก็บพอยทเตอร์ของบัฟเฟอร์เสียงทั้งหมดไว้ คือ ตำแหน่งที่เล่น (current play position) และ ตำแหน่งที่อ่าน (current write position)

<u>เพิ่มเติม</u> :

- Current play position คือ ค่า offset ในบัฟเฟอร์เพื่อให้ DirectSound นำข้อมูลตั้งแต่นั้น ไปเป็นข้อมูลในการผสานเสียง เพื่อเล่นเสียง
- Current write position คือ ค่า offset ในบัฟเฟอร์เพื่อให้ DirectSound นำข้อมูลตั้งแต่นั้น ไปเป็นข้อมูลในการบันทึกเสียงลงไป ตั้งแต่ตำแหน่งนี้เป็นต้นไป เป็นตำแหน่งที่ปลอดภัยที่จะเขียนข้อมูลลงบัฟเฟอร์ ซึ่งในโครงงานไม่ได้นำบัฟเฟอร์ประเภทนี้มาใช้งาน ดังนั้นไม่ขอกล่าวในรายละเอียด

5.9.2 การทำให้บัฟเฟอร์ที่ใช้ให้มีประสิทธิภาพ

5.9.2.1 บัฟเฟอร์ secondary แบบ static และ streaming

บัฟเฟอร์ static ควรจะเป็นเสียงที่มีความยาวสั้นๆ เสียงเดียว ที่ต้องเล่นซ้ำไปมา ส่วนบัฟเฟอร์ streaming จะเหมาะกับข้อมูลเสียงที่ขนาดไม่พอกับบัฟเฟอร์ เนื่องจากขนาดใหญ่ไปต้อง คัดลอกเป็นส่วนๆ ไปเก็บไว้ในบัฟเฟอร์ในขณะที่เล่นเสียงไปด้วย

โดย default เมธอด CreateSoundBuffer จะสร้างบัฟเฟอร์ streaming แต่ถ้าต้องการสร้างบัฟเฟอร์ static ต้องกำหนด flag เป็น DSBCAPS_STATIC

5.9.2.2 ฮาร์ดแวร์และซอฟต์แวร์

การ์ดเสียงบางรุ่นมีความสามารถในการผสานเสียงอยู่บนการ์ดแล้ว และ DirectSound สามารถสร้างบัฟเฟอร์ secondary บนฮาร์ดแวร์ หรือสามารถใช้หน่วยความจำจากการ์ดเสียงได้ แต่ถ้าไม่มีก็จะใช้

หน่วยความจำระบบแทน ถ้าเราต้องการสร้างบัฟเฟอร์จากการ์ดเสียงอย่างเดียวกำหนด flag ในโครงสร้าง DSBUFFERDESC เป็น DSBCAPS_LOCHARDWARE ในกรณีนี้ ถ้าไม่มีหน่วยความจำในการ์ดเสียงจะ fail หรือ ถ้าต้องการบัฟเฟอร์จากหน่วยความจำบนระบบอย่างเดียวก็กำหนด flag เป็น DSBCAPS_LOCSOFTWARE ซึ่งเหมาะกับบัฟเฟอร์ streaming หรือในกรณีที่เล่นเสียงเดียวกันพร้อมกัน (Duplicated sound) จะต้องคัดลอกบัฟเฟอร์ static เพิ่มไว้บนระบบด้วย (จากบัฟเฟอร์ที่มีอยู่บนการ์ดเสียง) ซึ่งไม่ต้องใช้ทรัพยากรจากบนฮาร์ดแวร์อีก เนื่องจากการเรียก DuplicateSoundBuffer จะ fail ถ้ามีทรัพยากรที่จำเป็น commit บนฮาร์ดแวร์ หรือซอฟต์แวร์ที่ใดที่หนึ่งอยู่แล้ว

ค่า flag อื่นๆ ที่น่าสนใจในโครงสร้าง DSBUFFERDESC ได้แก่

- ไม่ได้ใส่ flag : การพยายามสร้างบัฟเฟอร์ streaming บนฮาร์ดแวร์ หรือการ์ด และถ้าทรัพยากรหรือเนื้อที่หน่วยความจำของการ์ดไม่พอ จำเป็นต้องสร้างบัฟเฟอร์บนซอฟต์แวร์ หรือหน่วยความจำระบบ
- DSBCAPS_STATIC : การพยายามสร้างบัฟเฟอร์ static บนฮาร์ดแวร์ และถ้าเนื้อที่หน่วยความจำของการ์ดไม่พอ จำเป็นต้องสร้างบัฟเฟอร์บนซอฟต์แวร์
- DSBCAPS_ LOCHARDWARE : การสร้างบัฟเฟอร์ streaming บนฮาร์ดแวร์ และถ้าทรัพยากรของการ์ดไม่พอ การเรียกเกิด fail
- DSBCAPS_ LOCHARDWARE | DSBCAPS_STATIC : การสร้างบัฟเฟอร์ static บนฮาร์ดแวร์ และถ้าทรัพยากรของการ์ดไม่พอ การเรียกเกิด fail
- DSBCAPS_ LOCSOFTWARE : การสร้างบัฟเฟอร์ streaming บนซอฟต์แวร์
- DSBCAPS_ LOCSOFTWARE | DSBCAPS_STATIC : การสร้างบัฟเฟอร์บนซอฟต์แวร์

5.9.2.3 การผสานเสียงด้วยฮาร์ดแวร์/ซอฟต์แวร์

โดยปกติการผสานเสียงจะเป็นหน้าที่หลักของการ์ดเสียง แต่ถ้าการ์ดเสียงไม่รองรับ DirectX DirectSound จะผสานเสียงโดยใช้หน่วยความจำระบบแทน ซึ่งสามารถผสานเสียงจากหลายๆ บัฟเฟอร์บนระบบได้ และ stream เอาท์พุตเสียงไปยังตัวผสานเสียงฮาร์ดแวร์ เพื่อผสมกับเสียงจากบัฟเฟอร์บนฮาร์ดแวร์อีกที

5.9.2.4 การกำหนดบัฟเฟอร์เป็น Stick และ Global Focus

เมื่อสร้างบัฟเฟอร์ขึ้นมาจะมีการ กำหนดว่าเมื่อ minimize โปรแกรม หรือไปเรียกใช้โปรแกรมอื่นจะได้ยินเสียงที่เล่นหรือไม่ ถ้ากำหนดให้บัฟเฟอร์เป็น global focus หรือการกำหนด flag ในโครงสร้าง DSBUFFERDESC ให้เป็น DSBCAPS_GLOBALFOCUS จะได้ยิน ถ้ากำหนดให้เป็น sticky focus หรือการกำหนด flag ให้เป็น DSBCAPS_STICKYFOCUS จะไม่ได้ยิน (เป็น default)

5.9.2.5 การควบคุมบัฟเฟอร์

เมื่อสร้างบัฟเฟอร์ secondary ขึ้นมา นักพัฒนาสามารถควบคุมออบเจ็กต์ DirectSoundBuffer ให้ทำคุณลักษณะเหล่านี้ได้ โดยการกำหนด flag ในโครงสร้าง DSBUFFERDESC ซึ่งได้แก่

- คุณสมบัติบนระบบ 3 มิติ : กำหนด flag เป็น DSBCAPS_CTRL3D กรณีที่ต้องการใช้บัฟเฟอร์ผ่านอินเทอร์เฟซ IdirectSound3Dbuffer ซึ่งกำหนด flag นี้รวมกับ DSBCAPS_CTRLPAN ไม่ได้
- ความถี่ : กำหนด flag เป็น DSBCAPS_CTRLFREQUENCY
- Pan : กำหนด flag เป็น DSBCAPS_CTRLPAN ซึ่งกำหนดรวมกับ DSBCAPS_CTRL3D ไม่ได้
- ความดัง : กำหนด flag เป็น DSBCAPS_CTRLVOLUME
- Position notification : กำหนด flag เป็น DSBCAPS_CTRLPOSITIONNOTIFY

5.9.2.6 การหลีกเลี่ยง overhead ที่ไม่จำเป็น

- อย่าใช้การควบคุมบัฟเฟอร์ที่กล่าวข้างต้นถ้าไม่จำเป็น ตัวอย่างเช่น หลีกเลี่ยงการกำหนด flag ในโครงสร้าง DSBUFFERDESC ให้เป็น DSBCAPS_CTRALL (ควบคุมบัฟเฟอร์ทั้งหมด) หรือ DSBCAPS_CRLDEFAULT (ควบคุมบัฟเฟอร์เฉพาะบางอย่าง)
- เก็บบัฟเฟอร์ secondary ไว้ในรูปแบบเดียวกัน และถ้าจำเป็นก็เปลี่ยนรูปแบบให้ตรงกับบัฟเฟอร์ primary
- อย่าสร้างบัฟเฟอร์เสียง 3 มิติ ถ้าไม่จำเป็น

5.9.3 ไฟล์ประเภท WAVE หรือ waveform audio

เป็นรูปแบบข้อมูลเสียงอย่างง่ายๆ ส่วนมากบนระบบวินโดวส์ และวินโดว์สเก็บไฟล์ WAV ไว้ใน RIFF (Resource Interchange File Format) ซึ่งเก็บไฟล์ส่วนหัวและคุณสมบัติของ chuck ข้อมูล สำหรับการอ่านและเขียนไฟล์ RIFF เพียงค้นหาฟังก์ชัน wrapper ในโค้ดตัวอย่างของ DirectX SDK ควบคู่กับไฟล์ identical หรือ WAVE.C พร้อมกับรวมเวอร์ชันของไฟล์และไฟล์ส่วนหัวเพิ่มเติมด้วย ตัวอย่างเช่น การเรียกใช้ตัวอย่างโปรแกรม Dsstream จะค้นหาไฟล์ identical ที่ชื่อ DSTRWAVE.C และใช้ฟังก์ชันต่อไปนี้ในไฟล์ WAVE.C

- WaveOpenFile : เปิดไฟล์และรับข้อมูลเกี่ยวกับรูปแบบ
- WaveStartDataRead : ย้ายพอยทเตอร์ของไฟล์ไปที่จุดเริ่มต้นของ chunk ข้อมูล
- WaveReadFile : คัดลอกข้อมูลจากไฟล์ และ update พอยทเตอร์
- WaveCloseReadFile : ปิดไฟล์

5.9.3.1 การอ่านไฟล์ WAV จาก RIFF

ไฟล์เสียงสั้นๆ มักเก็บเป็นรูปแบบ resource ใน DLL (executable) มากกว่าอยู่ในรูปแบบไฟล์ WAV ซึ่งในบ้างครั้งค้นหาโค้ดตัวอย่างใน DirectX SDK ไม่พบ ยังสามารถเล่นเสียงได้โดยโหลดไฟล์ resource เข้าไปเก็บไว้ในบัฟเฟอร์

ในการเริ่มต้นสร้างบัฟเฟอร์ static ต้องรวมไฟล์ DSUTIL.H และ DSUTIL.C (จากโฟลเดอร์ \SDK\SAMPLES\MISC) และถ้าจะไม่คอมไพล์ไฟล์ดังกล่าว ต้อง include ไฟล์ WINDOWSX.H และ import ไฟล์ WAV ในโปรแกรม Microsoft Visual C++ เข้าไปเป็นไฟล์ resource ประเภท WAVE และปรับเปลี่ยนไฟล์ resource หรือเปลี่ยนชื่อไฟล์ WAV ใน DSUTIL.C (ตามหลักการตั้งชื่อไฟล์ WAV) ทำให้สามารถใช้ฟังก์ชัน FindResource ค้นหาไฟล์ resource รูปแบบ WAV ได้

ตัวอย่าง : สามารถสร้างบัฟเฟอร์ static และโหลดเสียงเข้าไปในบัฟเฟอร์ได้ โดยส่งค่าพอยทเตอร์ DirectSound และชื่อของไฟล์ resource ไปให้ฟังก์ชัน DSLoadSoundBuffer ถ้า DS_OK จะคืนค่าพอยทเตอร์ initialized ให้กับบัฟเฟอร์

```
LPDIRECTSOUND          lpds;    // สร้างออบเจ็กต์ DirectSound และกำหนดค่าเริ่มต้น
LPDIRECTSOUNDBUFFER    lpdsb;

lpdsb = DSLoadSoundBuffer(lpds, "BOUNCE");
if (lpdsb == NULL)
{
        // Failure.
}
```

เพิ่มเติม : เนื่องจาก DirectSound ไม่มีเมธอดที่ใช้อ่านและเขียนไฟล์ WAV ดังนั้นไมโครซอฟต์จึงเพิ่มตัวอย่างโปรแกรมสำหรับการเรียกไฟล์ WAV และไฟล์ resource รวมถึงการคัดลอกข้อมูลไปยังพอยทเตอร์ของหน่วยความจำ

5.9.4 การสร้างและใช้งานบัฟเฟอร์ secondary

จากข้างต้นสามารถสร้างบัฟเฟอร์ได้โดยอัตโนมัติ ต่อไปนี้เป็นขั้นตอนการกำหนดคุณสมบัติให้กับบัฟเฟอร์และเล่นเสียง

1. กำหนดค่าเริ่มต้นในโครงสร้าง WAVEFORMATEX เพื่อกำหนดรูปแบบไฟล์ WAV

   เพิ่มเติม :

   - การกำหนดรูปแบบไฟล์ WAV เป็นขั้นตอนแรกของการกำหนดคุณสมบัติของบัฟเฟอร์ ส่วนมากนักพัฒนาจะได้รับคุณสมบัติเหล่านี้จากตัวไฟล์ WAV เอง ตัวอย่างเช่น ฟังก์ชัน WaveOpenFile ในไฟล์ WAVE.C ได้ส่งค่าในฟังก์ชันผ่านพอยทเตอร์ให้กับโครงสร้าง WAVEFORMATEX
   - การกำหนดคุณสมบัติของบัฟเฟอร์ มีส่วนที่สำคัญ 3 ส่วนดังนี้
     - Flag ที่กำหนดว่าจะให้บัฟเฟอร์อยู่บนฮาร์ดแวร์/ซอฟต์แวร์ และควบคุมให้บัฟเฟอร์ทำงานแบบใด
     - ขนาดของบัฟเฟอร์ถ้าเป็น static ควรจะมีขนาดเท่ากับจำนวนไบต์ของข้อมูลเสียงที่ใหญ่ที่สุดที่ใช้เก็บบัฟเฟอร์ แต่ถ้าเป็นบัฟเฟอร์ streaming ควรมีขนาดใหญ่เพียงพอ

กับข้อมูลสำหรับเล่นเสียง 1-2 วินาที ซึ่งสามารถคำนวณได้จาก จำนวนไบต์เฉลี่ยต่อวินาที (nAvgBytesPerSec)

- ย้ายพอยทเตอร์ไปชี้ที่โครงสร้าง WAVEFORMATEX

ตัวอย่าง :

```
WAVEFORMATEX        pcmwf;        // โครงสร้าง WAVEFORMATEX

DWORD     dsbstatus;          // สถานะของบัฟเฟอร์เสียง

DWORD     audio_length_1   = 0, // ขนาดหน่วยความจำที่จะล็อคไว้สำหรับอ่าน
          audio_length_2   = 0,
          snd_buffer_length = 64000; // ขนาดบัฟเฟอร์ที่ใช้งาน

// จองพื้นที่หน่วยความจำสำหรับบัฟเฟอร์
UCHAR *snd_buffer_ptr = (UCHAR *)malloc(snd_buffer_length);
// กำหนดค่าต่างๆ ในโครงสร้างรูปแบบข้อมูลเสียงแบบ WAV
memset(&pcmwf, 0, sizeof(WAVEFORMATEX));
pcmwf.wFormatTag      = WAVE_FORMAT_PCM;
pcmwf.nChannels               = 1; // เสียงแบบ mono
pcmwf.nSamplesPerSec  = 11025; // 11 กิโลเฮิรตซ์
pcmwf.nBlockAlign     = 1;
pcmwf.nAvgBytesPerSec = pcmwf.nSamplesPerSec * pcmwf.nBlockAlign;
pcmwf.wBitsPerSample  = 8;
pcmwf.cbSize          = 0;
```

2. กำหนดค่าเริ่มต้นผ่านพารามิเตอร์ของบัฟเฟอร์ในโครงสร้างข้อมูล DSBUFFERDESC รวมถึงการย้ายพอยทเตอร์ไปชี้ที่ WAVEFORMATEX

ตัวอย่าง : การกำหนดค่าต่างๆ ในโครงสร้างข้อมูล DSBUFFERDESC

```
// สร้างบัฟเฟอร์ secondary
memset(&dsbd,0,sizeof(DSBUFFERDESC));
dsbd.dwSize   = sizeof(DSBUFFERDESC); // ขนาดโครงสร้าง
dsbd.dwFlags = DSBCAPS_CTRLDEFAULT
             | DSBCAPS_STATIC
             | DSBCAPS_LOCSOFTWARE; // กำหนดคุณสมบัติให้กับบัฟเฟอร์
dsbd.dwBufferBytes = snd_buffer_length+1; // ขนาดบัฟเฟอร์
dsbd.lpwfxFormat = &pcmwf; // เป็นพอยทเตอร์ชี้ไปยังโครงสร้าง WAVEFORMATEX
```

3. เรียกเมธอด CreateSoundBuffer เพื่อสร้างบัฟเฟอร์

ตัวอย่าง :

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

เพิ่มเติม :

- การใช้งานบัฟเฟอร์ เริ่มต้นจากการโหลดเสียงสั้นไปไว้ในบัฟเฟอร์ static เมื่อสร้างออบเจ็กต์ DirectSound โปรแกรมจะสร้างบัฟเฟอร์ static โดยอัตโนมัติทันที และโหลดไฟล์ WAV เข้าไปในบัฟเฟอร์ เล่นเสียง
- เมื่อเปิดไฟล์ครั้งแรก จะรับคุณสมบัติต่างๆ ของบัฟเฟอร์จากไฟล์ส่วนหัว และส่งไฟล์ RIFF ไว้ที่ส่วนเริ่มต้นของข้อมูลเสียง และก่อนที่จะใช้งานฟังก์ชันในไฟล์ WAVE.C ต้องกำหนดตัวแปรแบบ global 4 ตัว ได้แก่
  - * pwfx (WAVEFORMATEX) : เป็นคุณสมบัติของรูปแบบไฟล์ WAV
  - hmmio (HMMIO) : เป็นส่วนจัดการของไฟล์
  - mmckinfo (MMCKINFO) : เป็นคุณสมบัติของ chuck ข้อมูลเสียง
  - mmckinfoParent (MMCKINFO) : เป็นคุณสมบัติของ parent chunk ข้อมูลเสียง

ตัวอย่าง : การเปิดไฟล์ WAV ที่มีชื่อ lpzFileName โดยใช้พอยทเตอร์ชี้ไปที่ไฟล์ดังกล่าว โดยเราเรียกฟังก์ชัน WaveOpenFile เพื่อรับส่วนจัดการไปยังไฟล์ ตรวจสอบว่าเป็นไฟล์ RIFF หรือไม่ รับคุณสมบัติของรูปแบบไฟล์ WAV และส่งไปยังส่วนเริ่มต้นของ chuck ข้อมูล

```
if (WaveOpenFile (lpzFileName, &hmmio, &pwfx, &mmckinfoParent) != 0)
  return FALSE;
if (WaveStartDataRead (&hmmio, &mmckinfo, &mmckinfoParent) != 0)
  return FALSE;
```

4. ล็อคตำแหน่งของบัฟเฟอร์ เพื่อรับค่าแอดแดสของบัฟเฟอร์ ซึ่งใช้เมธอด Lock

ตัวอย่าง : ต้องผ่านค่า offset เริ่มต้นและจำนวนไบต์ที่จะล็อคไปยังเมธอด Lock ซึ่งได้รับพอยทเตอร์ 2 ค่าและพอยทเตอร์ที่ใช้นับ 2 ค่า

```
// คัดลอกข้อมูลเสียงใส่บัฟเฟอร์
if (lpdsbsecondary->Lock(0, //    เป็นค่า offset เริ่มต้นที่จะล็อค
                         snd_buffer_length, // เป็นจำนวนไบต์ที่จะล็อคในแต่ละครั้ง
                         &audio_ptr_1, // รับค่าแอดแดสเริ่มต้นที่จะล็อคของส่วนแรก
                         &audio_length_1, // ขนาดไบต์ในส่วนแรก ถ้าเป็นบัฟเฟอร์ circular
                         &audio_ptr_2,  // รับค่าแอดแดสเริ่มต้นที่จะล็อคของส่วนที่ 2 ถ้าเป็น
                         บัฟเฟอร์ static ให้เป็น NULL
                         &audio_length_2, // ขนาดไบต์ในส่วนที่ 2 ถ้าเป็นบัฟเฟอร์ circular ถ้า
```

เป็นบัฟเฟอร์ static ให้ เป็น NULL

DSBLOCK_FROMWRITECURSOR)!=DS_OK)

5. คัดลอกข้อมูล(จากไฟล์ WAV หรือ ข้อมูลเสียงแบบอื่น) ไปยังบัฟเฟอร์ตามตำแหน่งที่ Lock ไว้

ตัวอย่าง :

```
// คัดลอกส่วนแรกจากบัฟเฟอร์ circular
CopyMemory(audio_ptr_1, snd_buffer_ptr, audio_length_1);
// คัดลอกส่วนท้ายจากบัฟเฟอร์ circular
CopyMemory(audio_ptr_2, (snd_buffer_ptr+audio_length_1),audio_length_2);
```

6. ปลดล็อคบัฟเฟอร์ โดยใช้เมธอด Unlock

ตัวอย่าง :

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

7. สำหรับกรณีที่เล่นเสียงจากบัฟเฟอร์ static ต้องกำหนดตำแหน่งที่จะเล่น โดยใช้เมธอด SetCurrentPosition ซึ่งประกอบด้วยพารามิเตอร์ 1 ค่าคือ dwNewPosition (DWORD) เป็นค่า offset ในบัฟเฟอร์ที่เป็นตำแหน่งเริ่มต้นเล่นเสียงหรือตำแหน่งที่วนกลับมาเล่นอีกในครั้งต่อไป และกรณีที่หยุดการเล่นก่อนตำแหน่งสุดท้าย สามารถกลับไปเริ่มเล่นใหม่ในตำแหน่งที่ต้องการได้
8. เล่นเสียงจากบัฟเฟอร์ โดยใช้เมธอด Play ซึ่งประกอบด้วยพารามิเตอร์ 3 ค่าคือ
   - dwReserved1 (DWORD) : ต้องเป็นค่าเท่ากับ 0
   - dwReserved2 (DWORD) : ต้องเป็นค่าเท่ากับ 0
   - dwFlags (DWORD) : ถ้าต้องการให้เล่นเสียงรอบเดียว กำหนดให้ flag เป็น 0 แต่ถ้าต้องการให้เล่นวนไม่รู้จบ ต้องกำหนดให้ flag เป็น DSBPLAY_LOOPING

ตัวอย่าง :

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

สามารถหยุดเล่นได้โดยฟังก์ชัน Stop()

9. ถ้าเป็นบัฟเฟอร์ streaming ต้องทำขั้นตอนที่ 4-6 ซ้ำ

5.9.4.1 การเพิ่มจำนวนบัฟเฟอร์ static ที่เก็บเสียงแบบเดียวกัน

เมื่อต้องการได้ยินเสียงแบบเดียวกันหลายๆ เสียงพร้อมกัน เช่น เสียงกระสุนที่ดังพร้อมกันแบบต่อเนื่อง สามารถทำได้โดยสร้างบัฟเฟอร์ static ขึ้นมาหลายอัน แล้วใส่ข้อมูลเสียงแบบเดียวกันลงไป ในความเป็นจริงนักพัฒนาควรใช้ทรัพยากรอย่างคุ้มค่าที่สุด จึงควรสร้างเพียง 1 บัฟเฟอร์ instance ซึ่งสามารถเข้าถึงออบเจ็กต์ DirectSoundBuffer ได้หลายๆ ออบเจ็กต์ โดยใช้เมธอด DuplicateSoundBuffer ซึ่งประกอบไปด้วยพารามิเตอร์ 2 ค่าคือ

- lpDsbOriginal (LPDIRECTSOUNDBUFFER) : บัฟเฟอร์ต้นแบบ
- lplpDsbDuplicate (LPLPDIRECTSOUNDBUFFER) : แอดแดสของพอยทเตอร์ชี้ไปบัฟเฟอร์ instance

<u>เพิ่มเติม</u> :

- การสร้างบัฟเฟอร์เสียงแบบเดียวกันหลายๆ บัฟเฟอร์ ต้องคำนึงถึงการใช้ทรัพยากรอย่างประหยัด เหมือนกับว่าใช้เนื้อที่หน่วยความจำแบบ physical เดียวกันกับบัฟเฟอร์ต้นแบบ
- ไฟล์ DSUTIL.C จะ declare บัฟเฟอร์ให้เป็นประเภท SNDOBJ รวมทั้งเพิ่มฟังก์ชันต่างๆ เช่น SndObjCreate ซึ่งทำให้การจัดการกับบัฟเฟอร์มีลักษณะคล้ายหลักการโปรแกรมเชิงออบเจ็กต์มากไปกว่าครึ่งหนึ่งแล้ว และมีผลดีต่อการสร้างและเล่นเสียงจากบัฟเฟอร์หลายๆ บัฟเฟอร์พร้อมกันได้โดยไม่เกิด fail

5.9.4.2 การส่งข้อมูลให้เล่นเสียงในบัฟเฟอร์ streaming

การกำหนดคุณสมบัติของบัฟเฟอร์ streaming เหมือนกับบัฟเฟอร์ static แต่มีตัวแปร global เพิ่มเติมดังต่อไปนี้

<u>ตัวอย่าง</u> :

```
WAVEFORMATEX        * pwfx;          // เป็นคุณสมบัติของรูปแบบไฟล์ WAV
HMMIO               hmmio;           // เป็นส่วนจัดการของไฟล์
MMCKI               mmckinfo, mmckinfoParent;
// เป็นคุณสมบัติของ chuck และ parent chuck ข้อมูลเสียง
DWORD               dwMidBuffer;     // จำนวนไบต์ของบัฟเฟอร์ / 2
BOOL SetupStreamBuffer (LPSTR lpzFileName)
{
        DSBUFFERDESC dsbdesc;
        HRESULT              hr;

        // ปิดไฟล์ที่เปิดอยู่ให้หมด และทำให้บัฟเฟอร์ว่าง
        WaveCloseReadFile (&hmmio, &pwfx);
        if (WaveOpenFile (lpzFileName, &hmmio, &pwfx, &mmckinfoParent) != 0)
           return FALSE;
        if (WaveStartDataRead (&hmmio, &mmckinfo, &mmckinfoParent) != 0)
           return FALSE;
        // สร้างบัฟเฟอร์ secondary เพื่อเก็บข้อมูลเสียง 2 นาที
        memset(&dsbd,0,sizeof(DSBUFFERDESC));
        dsbd.dwSize = sizeof(DSBUFFERDESC);
        dsbd.dwFlags = DSBCAPS_GETCURRENTPOSITION2
```

```
                    | DSBCAPS_GLOBALFOCUS
                    | DSBCAPS_CTRLPAN;
        dsbd.dwBufferBytes = pwfx- >nAvgBytesPerSec * 2;
        dsbd.lpwfxFormat = &pcmwf;

        if (FAILED (hr = lpds->CreateSoundBuffer (&dsbdesc, &lpdsb, NULL) ) )
        {
            WaveCloseReadFile (&hmmio, &pwfx);
        }
        FillBufferWithSilence (lpdsb);
        hr = lpdsb->Play (0, 0, DSBPLAY_LOOPING);

        dwMidBuffer = dsbdesc.dwBufferBytes / 2;

        return TRUE;
    } // การกำหนดคุณสมบัติบัฟเฟอร์ streaming
```

จากข้างต้น การคืนค่าจากฟังก์ชันบูลีน SetupStreamBuffer เป็น true/false เพื่อแสดงว่าการกำหนดบัฟเฟอร์ถูกต้อง/ไม่ถูกต้อง และการกำหนด flag ในโครงสร้าง DSBUFFERDESC ต้องเพิ่มกรณีที่เป็น DSBCAPS_GETCURRENTPOSITION2 เพราะสามารถหาตำแหน่งที่จะเล่นเสียง

ในการส่งข้อมูลในบัฟเฟอร์ streaming จะต้องรู้ว่าเมื่อไร ที่จะถึงเวลาที่จะส่งข้อมูลอีก การส่งข้อมูลทำได้ 2 วิธี คือ

- Polling: จะเป็นการตรวจข้อมูลที่เล่นในขณะนั้น ทุกช่วงระยะเวลา (ตรวจทุกครั้งที่ผ่านลูป message) การใช้วิธีนี้วนเรียกใช้ซีพียูมากกว่าแต่ใช้ง่ายกว่า
- Notification: รอ DirectSound ให้สัญญาณเล่นเสียงได้ เมื่อพอยทเตอร์ของตำแหน่งเล่นอยู่ในตำแหน่งที่ต้องการให้เริ่มเล่น

```
    BOOL PlayBuffer (void)
    {
        HRESULT         hr;
        DWORD           dwPlay;
        DWORD           dwStartOfs;
        Static DWORD    dwLastPlayPos;
        VOID            * lpData;
        DWORD           dwBytesLocked;
        UINT            cbBytesRead

        if (lpdsb = = NULL) return FALSE;
```

```
if (FAILED (lpdsb->GetCurrentPosition (&dwPlay,   //รับค่า offset ที่เป็นตำแหน่งเล่น
                                                  ถ้าไม่ต้องการรับค่า ให้ส่งค่า NULL
                                                  NULL) ) ) //รับค่า offset ที่เป็นตำแหน่งเขียน
                                                  ถ้าไม่ต้องการรับค่า ให้ส่งค่า NULL
   return FALSE;

// ถ้าตำแหน่งเล่นไปถึงตำแหน่งแรกของส่วนครึ่งหลัง
// ในขณะเดียวกันเขียนข้อมูลในส่วนครึ่งแรกได้
if ( ( (dwPlay > = dwMidBuffer) && (dwLastPlayPos < dwMidBuffer) )
   | | (dwPlay < dwLastPlayPos) )
{
        dwStartOfs = (dwPlay > = dwMidBuffer) ? 0 : dwMidBuffer;

        hr = lpdsb->Lock (dwStartOfs,        // ค่า offset ที่จะเริ่มต้นล็อค
                dwMidBuffer,      // จำนวนไบต์ที่จะล็อคในแต่ละครั้ง
                &lpvData,         // รับค่าแอดแดสของไบต์เริ่มล็อค
                &dwBytesLocked, // จำนวนไบต์ที่ล็อค
                NULL, // แอดแดสของส่วนอีกครึ่งที่เขียนข้อมูล
                NULL, // จำนวนไบต์ของส่วนอีกครึ่งที่เขียนข้อมูล
                0, // flag
        WaveReadFile ( hmmio, // ไฟล์ส่วนจัดการ
                DwBytesLocked , // จำนวนไบต์ที่จะอ่านในแต่ละครั้ง
                ( BYTE * )lpvData, // บัฟเฟอร์ที่จะอ่านข้อมูลไปใส่
                &mmckinfo, // ไฟล์คุณสมบัติของ chuck
                &cbBytesRead); // จำนวนไบต์ที่จะอ่าน

        if (cbBytesRead < dwBytesLocked)
                // ตำแหน่งอ่านไปถึงตำแหน่งสุดท้ายของส่วนครึ่งแรก
        {
                if (WaveStartDataRead (&hmmio, &mmckinfo, &mmckinfoParent)
                        != 0)
                {
                        OutputDebugString ("Can't reset file.\n");
                }
```

```
                else
                {
                        WaveReadFile ( hmmio,
                                        DwBytesLocked – cbBytesRead,
                                        ( BYTE * )lpvData + cbBytesRead,
                                        &mmckinfo,
                                        &cbBytesRead);
                }
            }
            lpdsb->Unlock (lpvData, dwBytesLocked, NULL, 0);
        }
        dwLastPlayPos = dwPlay;
        return TRUE;
    } // เล่นเสียงจากบัฟเฟอร์
```

จากข้างต้น ถ้าบัฟเฟอร์ไม่ทำงานหรือไม่มีการเล่นเสียง ฟังก์ชันจะคืนค่าตำแหน่งเล่นออกมา หรือไม่เช่นนั้น สามารถรับค่าตำแหน่งเล่นจากการใช้ฟังก์ชัน GetCurrentPosition

จากตัวอย่างโค้ดจะเห็นว่า ตำแหน่งเขียนข้อมูลจะอยู่ล่วงหน้าตำแหน่งเล่นประมาณ 1 นาที เนื่องจากเป็นบัฟเฟอร์เล่นเสียง 2 นาที (ส่วนมากล่วงหน้าประมาณ 15 3 10^{-3} นาที) และเมื่อสร้างบัฟเฟอร์กำหนดให้ตัวแปร global ที่เป็น dwMidBuffer มีค่าเท่ากับขนาดของบัฟเฟอร์ / 2 และกำหนดตำแหน่งเล่นจะอยู่ที่ค่า offset หรือตำแหน่งกึ่งกลางบัฟเฟอร์ ดังนั้นจึงปลอดภัยที่จะเขียนข้อมูลในบัฟเฟอร์ส่วนครึ่งแรก และเมื่อเล่นเสียงจนตำแหน่งเล่นไปอยู่ที่ตำแหน่งสุดท้ายของบัฟเฟอร์ และเริ่มวนกลับไปเล่นตำแหน่งแรกของบัฟเฟอร์ จึงจะเริ่มเขียนในบัฟเฟอร์ส่วนหลัง หรือตำแหน่งเขียนอยู่ที่กึ่งกลางบัฟเฟอร์ซึ่งตำแหน่งเล่นอยู่ที่ส่วนครึ่งแรก/ครึ่งหลังของบัฟเฟอร์ จะต้องล็อคบัฟเฟอร์ส่วนดังกล่าวไว้สำหรับเล่นโดยไม่สามารถเขียนข้อมูลทับส่วนที่ล็อคได้ และปลดล็อคในส่วนที่ต้องการเขียนทับ

5.9.5 การติดต่อกับ DMA (Direct Memory Access)

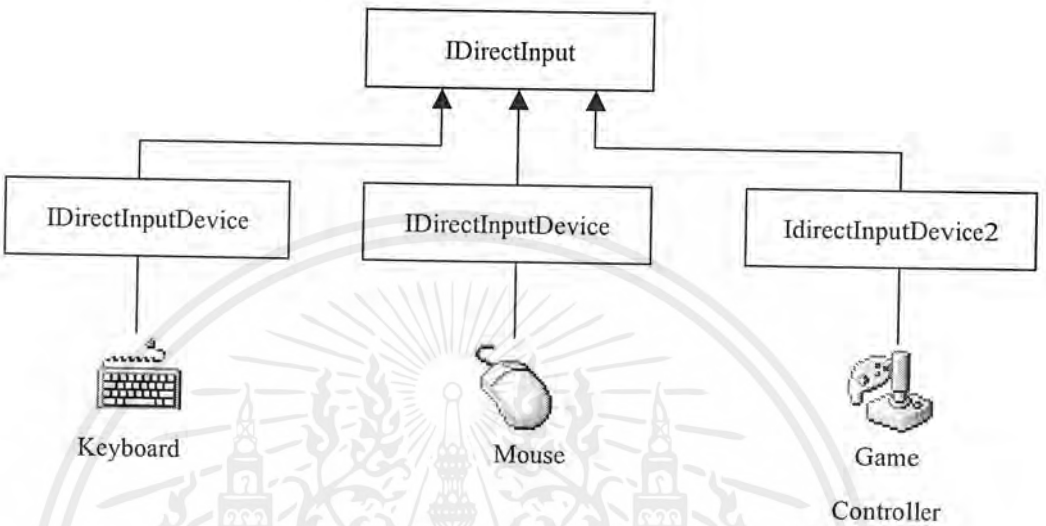
การ์ดเสียงส่วนมากในปัจจุบันมีสถาปัตยกรรมแบบ ISA (Industry Standard Architecture) ซึ่งมีคุณสมบัติตรงกันข้ามกับการ์ดแบบ PCI (Peripheral Component Interconnect) คือ มี channel ในการส่งข้อมูลเสียงเพื่อเล่นเสียงแบบเข้าถึงหน่วยความจำโดยตรง หรือ DMA ซึ่งเมื่อตัวผสานเสียงทำงาน/หยุด จะทำให้เกิด overhead จึงมีเทคนิคแก้ปัญหาดังกล่าว คือ ในขณะที่ minimize แอปพลิเคชั่น จะให้เล่นเสียงจากบัฟเฟอร์ primary/secondary อย่างเบาๆอยู่ตลอดเวลา โดยเมื่อใช้เมธอด Play เล่นเสียงจากบัฟเฟอร์ primary กำหนด flag ให้เป็น DSBPLAY_LOOPING หรือให้เล่นเสียงวนลูปอยู่ตลอดเวลา

บทที่ 6

DirectInput

6.1 รู้จักกับ DirectInput

6.1.1 โครงสร้างการติดต่อของ DirectInput



รูปที่ 6-1 การติดต่อของ DirectInput

DirectInput ได้นำเสนออินเทอร์เฟซสำหรับอุปกรณ์รับอินพุต (Input device) ต่างๆ (เช่น จอยสติก เฮดเกียร์ เมาส์ที่มีหลายปุ่ม เป็นต้น) รวมถึงอุปกรณ์รับอินพุต-แสดงเอาท์พุต ซึ่งมีแรงตอบกลับมาสู่ผู้เล่นเกม เรียกว่า force-feedback โดย DirectInput จะมีข้อได้เปรียบที่สำคัญเหนือจากฟังก์ชัน API มาตรฐานทั่วไปอยู่ 2 ประการ คือ สามารถสนับสนุนอุปกรณ์ได้หลายชนิดมากขึ้นและสามารถตอบสนองได้เร็วกว่า DirectInput สามารถทำงานภายในระบบการส่งข้อความของวินโดวส์เนื่องจากทำงานโดยตรงกับไดเวอร์ของอุปกรณ์

คุณลักษณะใหม่ของ DirectInput นั้นรวมถึงการสนับสนุนอุปกรณ์ใหม่ๆ เช่น เกมแพด คันโยก (flight yokes) และเฮดเกียร์รุ่นใหม่ๆ ที่เพิ่มความสมจริงให้กับการเล่นเกม เช่นเดียวกับอุปกรณ์ force-feedback ซึ่งสามารถทำเอฟเฟ็กต์เช่น การตอบกลับด้วยแรงสั่นสะเทือนและแรงต้าน ทำให้ผู้ใช้ได้รับความสมจริงและเกมมีความสนุกน่าเล่นมากขึ้น

สถาปัตยกรรมพื้นฐานของ DirectInput ประกอบด้วยออบเจ็กต์ DirectInput ซึ่งรองรับอินเทอร์เฟซและออบเจ็กต์ COM สำหรับอุปกรณ์รับอินพุต แต่ละตัวที่ส่งข้อมูลเข้ามา โดยแต่ละอุปกรณ์ (device) จะมีออบเจ็กต์ instance ของมันเอง ซึ่งแต่ละออบเจ็กต์จะมีการควบคุมที่แตกต่างกัน เช่น ปุ่ม แกน เป็นต้น

คำว่าออบเจ็กต์ในที่นี้ใช้แสดงสิ่งต่างๆ ที่สร้างโดย DirectInput ที่รองรับเมธอดของอินเทอร์เฟซ COM โดยเมธอดเหล่านี้ไม่ได้ถูกใช้เรียกผ่าน OOP เช่น ภาษา VC++

ในด้านความเร็ว DirectInput ทำงานโดยตรงกับอุปกรณ์ต่างๆ โดยผ่านระบบ message ของวินโดวส์

6.1.2 ปุ่มและแกน

ปุ่มเป็นสวิตช์อย่างหนึ่ง ซึ่งมีตำแหน่งเพียงกดเปิด/ปิด ซึ่ง DirectInput ไม่ได้กำหนดความแตกต่างระหว่างแต่ละปุ่ม แต่สำหรับการควบคุมเมาส์ คีย์บอร์ด จอยสติก และเกมแพด ใช้แกน ซึ่งไม่มีความแตกต่างในแต่ละปุ่ม แต่มีตำแหน่ง หรืออาจกล่าวได้ว่า แกนเป็นออบเจ็กต์ physical และทำให้เกิดค่าต่างๆ โดยเจาะจงให้เคลื่อนย้ายไปในทิศทางที่ต้องการ และมีค่าที่ควบคุมแกนอยู่ 2 ค่า คือ แกน X และแกน Y สำหรับบนจอคอมพิวเตอร์ 2 มิติ ซึ่ง throttle knob หรือ slider บนจอยสติก หรือ wheel บนเมาส์ควบคุมเพียง 1 แกน สำหรับจอยสติกมีค่าควบคุมแกนเพิ่มอีก 1 ค่า คือ แกนที่แสดงการบิด ควบคุมโดยใช้ rudder นอกจากนี้การใช้พวงมาลัย (steering wheel) คันโยก (yoke) และกันเร่งความเร็ว (pedal)

สำหรับตัวควบคุมแกน Z ของอุปกรณ์ นำมาใช้ควบคุมแอปพลิเคชันในระบบ 3 มิติ หรืออาจใช้ควบคุมการเลื่อนขึ้น-ลงของหน้าต่าง การปรับค่าความดังของลำโพง หรือแม้แต่อะไรก็ตามที่ไม่เกี่ยวกับการควบคุมระยะเลยก็สามารถใช้ตัวควบคุมดังกล่าวได้เช่นกัน อาจกล่าวได้ว่าอุปกรณ์รับอินพุตต่างๆ ใช้ตัวควบคุมบนอุปกรณ์เป็นออบเจ็กต์ physicalในการ map กับออบเจ็กต์ทาง logical หรือแกน

6.2 การใช้งาน DirectInput

การเรียกใช้ DirectSound ประกอบด้วย 3 ส่วน

1. ขณะ run-time .DLL(Device Driver Interface) จะถูกโหลดเข้าไปเมื่อใช้ DirectInput
2. ขณะ compile-time ใช้ library DINPUT.LIB
3. เรียกไฟล์ส่วนหัว ที่ชื่อ DINPUT.H

6.2.1 ขั้นตอนการสร้าง DirectInput

1. สร้าง DirectInput ขึ้นมา โดยเรียกฟังก์ชัน DirectInputCreate โดยการคืนค่าพอยทเตอร์ไปอินเทอร์เฟซ IDirectInput เพื่อสร้างออบเจ็กต์ ซึ่งฟังก์ชันนี้ประกอบด้วยพารามิเตอร์ 4 ค่าคือ
   - hinst (HINSTANCE) : ส่งส่วนจัดการ Instance ไปแอปพลิเคชัน หรือ DLL ซึ่งสร้างออบเจ็กต์ DirectInput
   - dwVersion (DWORD) : เป็นเลขเวอร์ชันของ DirectInput ซึ่งใช้ในการออกแบบแอปพลิเคชัน โดยปรกติเป็นค่า DIRECTINPUT_VERSION เนื่องจากนักพัฒนามีอิสระในการออกแบบแอปพลิเคชันด้วย DirectX เวอร์ชันเก่าได้ ตัวอย่างเช่น การส่งค่า 0x0300 และสามารถรันเกมบนระบบที่ติดตั้งไฟล์ run-time ของ DirectX 3.0 แต่ระบบต้องมีโครงสร้างที่ compatible กับ DirectX 3.0 และส่งค่า DIDEVCAPS_DX3 กับระบบที่มี DirectX 5.0 ซึ่งวิธีที่ปลอดภัยและง่ายที่สุดในการกำหนดค่า DIRECTINPUT_VERSION 0x0300 ก่อนการ includeไฟล์ส่วนหัว

ตัวอย่าง :

```
#define DIRECTINPUT_VERSION 0x0300
#include <dinput.h>
```

- * lplpDirectInput (LPDIRECTINPUT) : เป็นแอดแดสของพอยทเตอร์ชี้ไปออบเจ็กต์ DirectInput อันใหม่
- pUnkOuter (LPUNKNOWN) : เป็นค่า NULL

2. การกำหนดอุปกรณ์ (Enumerating Devices) คือ การร้องขอ DirectInput ให้หาอุปกรณ์รับอินพุตที่ต่ออยู่

เพิ่มเติม :

- เมาส์และคีย์บอร์ด อาจไม่ต้องใช้ แต่สำหรับเกมที่ต้องมีผู้เล่นมากกว่า 1 คน ต้องมีการกำหนดตัวแปรเริ่มต้น คือ GUID_SysMouse และ GUID_SysKeyboard ในการเรียกเมธอด CreateDevice ในอินเทอร์เฟซ IdirectInput
- จอยสติก จำเป็นเพราะจอยสติกมีหลายแบบ ทำให้ GUID ไม่ว่างพอเหมือนเมาส์และคีย์บอร์ด เราต้องร้องขอไปว่า จอยสติกต่ออยู่ แล้วรับ GUID มาแล้วจึงใช้ ID เหล่านี้
- การกำหนดอุปกรณ์ใดๆ จะต้องกำหนดที่ฟังก์ชัน callback ก่อนแล้วจึงเรียกเมธอด EnumDevice ซึ่งประกอบด้วยพารามิเตอร์ 3 ค่าได้แก่
    - dwDevType (DWORD) : เป็นค่า DIDEVTYPE_*value เพื่อระบุประเภทอุปกรณ์ที่ต้องการ หรือให้ค่าเป็น 0 สำหรับอุปกรณ์ทุกประเภท
    - pvRef (LPVOID) : ส่วนมากเป็นพอยทเตอร์ไปข้อมูลที่จะเลือกและนำไปใช้ในฟังก์ชัน callback ซึ่งเป็นค่าพารามิเตอร์ตัวที่ 2 ที่ส่งไปยังฟังก์ชัน callback
    - dwflags (DWORD) :

```
HRESULT EnumDevices( DWORD dwDevType,
                     // ค่า DIDEVTYPE_*value หรือชนิดของอุปกรณ์ที่หา เป็นค่า 0
                     สำหรับอุปกรณ์ทุกชนิด
                     LPDIENUMCALLBACK lpCallback,
                     // แอดแดสของฟังก์ชัน callback
                     LPVOID pvRef,
                     // ค่าของข้อมูล 32-bit ที่ใช้ส่งกลับไปฟังก์ชัน callback
                     DWORD dwFlags ); // flag ว่าจะกำหนดอุปกรณ์แบบไหน
```

3. การกำหนดอุปกรณ์

- การสร้างอุปกรณ์ : ต้องมี GUID สำหรับแต่ละอุปกรณ์ โดยได้รับจากการกำหนดอุปกรณ์ ในสร้าง GUID instance ในโครงสร้าง DIDEVICEINSTANCE และDirectInput จะส่งค่าไปยัง

ฟังก์ชัน callback หรือถ้าไม่ได้รับ GUID ให้กำหนดตัวแปรเริ่มต้น เช่น GUID_SysMouse หรือ GUID_SysKeyboard และใช้เมธอด CreateDevice สร้างอุปกรณ์ ซึ่งมีพารามิเตอร์ 3 ค่าคือ

- rguid (REFGUID) : เป็น GUID instance สำหรับอุปกรณ์
- lplpDirectInputDevice (LPDIRECTINPUTDEVICE) : แอดแดสของพอยทเตอร์ไปยังอินเทอร์เฟซ IDirectInputDevice อันใหม่
- pUnkOuter (LPUNKNOWN) : ต้องเป็นค่า NULL

- การกำหนดรูปแบบข้อมูล : ต้องรู้โครงสร้างของข้อมูลที่มาจากอุปกรณ์ต่างๆ เพราะถึงแม้ว่าอุปกรณ์ต่างๆ มีขนาดข้อมูลเท่ากัน แต่มีรูปแบบต่างกัน โดยใช้เมธอด SetDataFormat ซึ่งมีพารามิเตอร์ คือ lpdf (LPCDIDATAFORMAT) หรือค่าแอดแดสของโครงสร้าง DIDATAFORMAT ซึ่งแสดงคุณสมบัติของอาร์เรย์ในโครงสร้าง DIOBJECTDATAFORMAT และโครงสร้างนี้จะกำหนดการจัดการข้อมูลของแต่ละปุ่มและแกนบนอุปกรณ์

  เพิ่มเติม : DirectSound ยอมให้กำหนดอุปกรณ์อินพุตที่ยังไม่ได้ถูกสร้างเลย นักพัฒนาต้องกำหนดรูปแบบข้อมูลก่อนทำการสร้างออบเจ็กต์ DirectInput ในโครงสร้างข้อมูลสำหรับอุปกรณ์มาตราฐาน ซึ่งเป็นตัวแปร global และจะได้รับการคืนค่าเป็นโครงสร้างมาตราฐานดังต่อไปนี้

  - c_dfDIMouse c_dfDIkeyboard และ c_dfDIJoystick : คืนค่า DIMOUSESTATE DIKEYBOARD และ DIJOYSTATE ซึ่งเป็น char[256]
  - c_dfDIJoystick2 : คืนค่า DIJOYSTATE2 สำหรับเกมคอนโทรลเลอร์ที่ซับซ้อนขึ้น

- รับรูปแบบข้อมูลเกี่ยวกับอุปกรณ์ : ซึ่งมีเมธอดที่รับค่าคุณสมบัติต่างๆ เกี่ยวกับอุปกรณ์ และออบเจ็กต์ instance ของอุปกรณ์ เพื่อใช้ในการกำหนดอุปกรณ์ ได้แก่

  - GetDeviceStatus : ส่งค่า GUID instance ของอุปกรณ์ และถ้าต่ออุปกรณ์ดังกล่าวเข้าระบบแล้ว เมธอดคืนค่า DI_OK โดยไม่ต้องสร้างอุปกรณ์ดังกล่าวเป็นอุปกรณ์ DirectInput
  - EnumObjects : การกำหนดปุ่ม แกน และ point-of-view hat บนอุปกรณ์
  - GetCapabilities : กรณีที่ต่ออุปกรณ์เข้าระบบ และอุปกรณ์รองรับตัวควบคุม force-feedback จำนวนปุ่ม แกน และ point-of-view hat แสดงคุณสมบัติต่างๆ ในโครงสร้าง DIDEVCAPS
  - GetDeviceData : รับข้อมูลอินพุตจากบัฟเฟอร์บนอุปกรณ์
  - GetDeviceState : รับสถานะในขณะนั้นจากปุ่ม แกน และ hat
  - GetObjectInfo : รับค่าคุณสมบัติจากปุ่ม หรือแกนพิเศษในโครงสร้าง DIDEVICEINSTANCE ผลตอบรับจากอุปกรณ์ และ GUID instance ประเภทของอุปกรณ์ เป็นต้น
  - GetProperty : รับค่าคุณสมบัติของอุปกรณ์ และสามารถใช้ SetProperty ในการเปลี่ยนลักษณะของตัวควบคุม

  เพิ่มเติม : เมธอดที่รับรูปแบบข้อมูล ได้แก่

- EnumCreatedEffectObjects : ถ้าอุปกรณ์มีตัวควบคุมเป็น force-feedback จะต้องกำหนดเอฟเฟ็กต์
- EnumEffects : เหมือนเมธอดที่แล้ว แต่เป็นเอฟเฟ็กต์ที่อุปกรณ์รองรับ
- GetEffectInfo : คืนค่าต่างๆ ในโครงสร้าง DIEFFECTINFO ที่เกี่ยวกับเอฟเฟ็กต์ของอุปกรณ์ที่มี force-feedback รองรับ
- GetForceFeedbackState : เหมือนเมธอดที่แล้ว แต่รับค่าสถานะหรือแม้แต่แสดงว่าอุปกรณ์เปิดอยู่หรือไม่

เพิ่มเติม : การกำหนดรูปแบบข้อมูล โดยใช้เมธอด EnumObjects ซึ่งบางครั้งไม่จำเป็น เนื่องจากข้อจำกัดทางคุณสมบัติของอุปกรณ์ด้วย และจากนั้นคืนค่าข้อมูลไปที่โครงสร้าง DIDEVICEINSTANCE และใช้ฟังก์ชัน CountJoyBottons ส่งค่าไปส่วนจัดการวินโดวส์ของ dialog box และแสดงชื่อ friendly ของปุ่มและจอยสติกใน drop-down list ของ dialog box ดังกล่าว ขณะเดียวกันจะเก็บค่า offset ของอุปกรณ์ ไว้สำหรับแสดงตัวควบคุมของอุปกรณ์

- การแสดงตัวควบคุมของอุปกรณ์ : มี 2 แบบ คือ โดย Offset กับโดย ID
- กำหนดและรับค่าคุณสมบัติของอุปกรณ์ : คุณสมบัติอุปกรณ์ ตัวอย่างเช่น วิธีการแสดงข้อมูลของแกนเป็น relative (มีขอบเขตของแกน)/absolute (ไม่มีขอบเขตของแกน) ขนาดบัฟเฟอร์สำหรับเก็บข้อมูลอินพุต และความสัมพันธ์ของระยะระหว่างแกนทาง physical และข้อมูลที่แสดงออกมาเป็นต้น และสามารถใช้เมธอด SetProperty เป็นตัวกำหนด และ GetProperty เป็นตัวรับค่า ซึ่งเมธอดเหล่านี้ประกอบด้วยพารามิเตอร์ 2 ค่าคือ
  - rguidProp (REFGUID) : เป็นส่วนระบุคุณสมบัติของอุปกรณ์ ซึ่งมักกำหนดไว้ในส่วนเริ่มต้นแล้ว (DIPROP_*value) จากไฟล์ส่วนหัว DINPUT.H หรือจาก GUID ของอุปกรณ์
  - pdiph (LPCDIPROPHEADER) : แอดเดสของโครงสร้าง DIPROPHEADER และโครงสร้างข้อมูลหลัก ได้แก่ DIPROPDWORD สำหรับเก็บค่าประเภท DWORD หรือ DIPROPRANGE สำหรับเก็บค่าประเภท LONG
- กำหนด Cooperative level: เป็นการกำหนด flag ว่า ทำโหมด exclusive หรือไม่ และแอปพลิเคชันเข้าถึงข้อมูลได้ตลอดเวลาหรือไม่ foreground(เข้าถึงข้อมูลเฉพาะเมื่อแอปพลิเคชันใช้งาน) /background (ตลอดเวลา) ในโครงงานนี้จึงกำหนดเป็น DISCL_NONEXCLUSIVE | DISCL_BACKGROUND

4. Acquire การอนุญาตให้ใช้อุปกรณ์และแจ้ง DirectInput ว่าต้องการข้อมูลจากอุปกรณ์ตามรูปแบบของข้อมูลที่กำหนดไว้
5. Unacquire การยกเลิกการใช้อุปกรณ์จาก DirectInput
6. Release การยกเลิกการใช้ DirectInput

เพิ่มเติม : Poll สำหรับจอยสติกโดย DirectInput จะตรวจสอบว่าอุปกรณ์ต้อง polling หรือไม่ ถ้าต้องการจะเก็บสถานะที่เปลี่ยนไป(โดยอินเทอร์รัปจากฮาร์ดแวร์) ทันที

บทที่ 7
ผลลัพธ์ที่ได้จากโครงงาน

7.1 เกม 3 มิติ

เกมคอมพิวเตอร์3 มิติที่ได้ทำการพัฒนาขึ้นมา มีรูปแบบเป็นการจำลองการขับยานในคณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง กติกาคือให้ขับยานวนรอบคณะให้ครบ 3 รอบและพยายามทำเวลาในแต่ละรอบให้ดีที่สุด เมื่อเกมจบจะเปรียบเทียบเวลาในการขับยานแต่ละรอบและแสดงเวลาในรอบที่ดีที่สุด เมื่อผู้เล่นเริ่มเล่นเกมหน้าจอแรกจะปรากฏเมนูให้เลือกว่าจะเริ่มเล่นเกม (NEW GAME) หรือออกจากเกม (EXIT)

รูปที่ 7-1 หน้าจอเริ่มแรกของเกม ให้ผู้เล่นเลือกว่าจะเริ่มเล่นหรือออกจากเกม

เมื่อขับยานครบรอบ เกมจะเริ่มจับเวลาของรอบถัดไป มุมด้านล่างขวาของจอแสดงจำนวนรอบ และมุมด้านล่างซ้ายแสดงเวลาที่ใช้ในรอบของนั้น

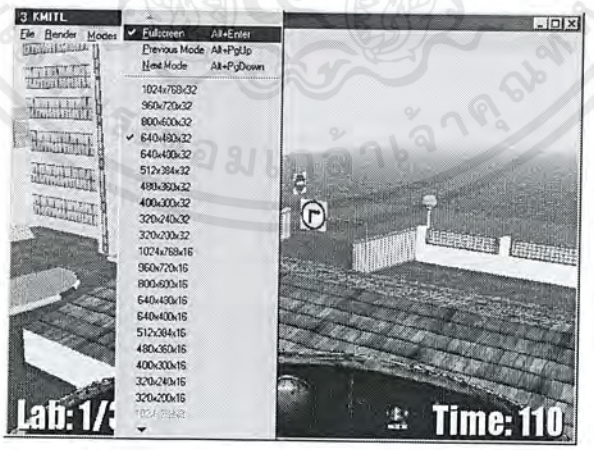
รูปที่ 7-2 หน้าจอของเกม เมื่อขับยานครบรอบ

ในระหว่างที่เล่นเกม ผู้เล่นสามารถหยุดเล่นเกมชั่วคราวโดยกดปุ่ม Pause Break และเลือกได้ว่าจะเริ่มเล่นเกมใหม่อีกครั้งหรือออกจากเกมได้ หากต้องการกลับไปเล่นเกมต่อให้กดปุ่ม Pause Break อีกครั้ง

รูปที่ 7-3 หน้าจอของเกม เมื่อกดปุ่ม Pause Break

7.2 การเลือกโหมดแสดงผล

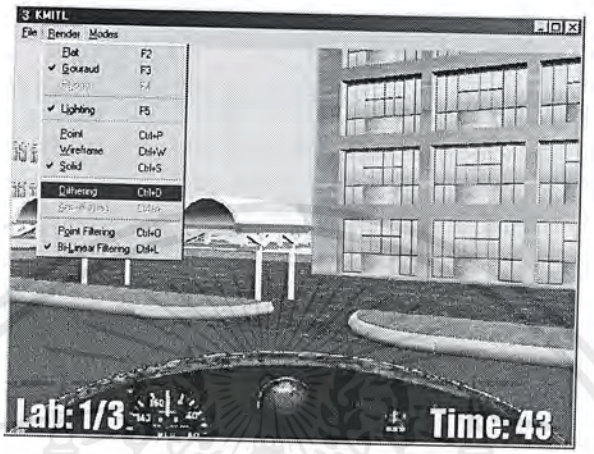
ขณะเล่นเกม เราสามารถปรับเปลี่ยนโหมดแสดงผลได้ โดยกด Alt เลื่อนไปที่แถบเมนู Modes แล้วเลือกโหมดความละเอียดตามความต้องการ สังเกตว่าขณะที่เล่นเกมจะมีข้อความปรากฏด้านบนของจอภาพ ซึ่งจะแสดงว่าขณะนั้นเกมมีการแสดงผลด้วยความเร็วเท่าไร (เฟรมต่อวินาที) ดังนั้นผู้เล่นควรเลือกโหมดการแสดงผลที่ไม่ทำให้เกมช้าลงมากนัก โดยพิจารณาจากความสามารถของฮาร์ดแวร์ของเครื่องคอมพิวเตอร์ที่ใช้เล่นเกม



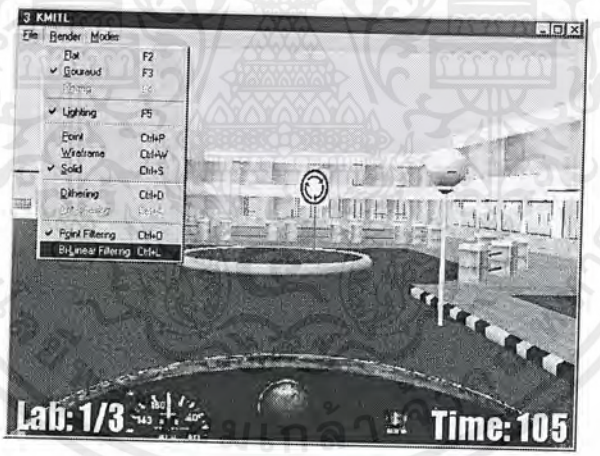
รูปที่ 7-4 การเลือกโหมดแสดงผล

7.3 การปรับเปลี่ยนค่าการแสดงผล

ในเมนูการเรนเดอร์ มีฟังก์ชันการทำงานให้เลือกใช้มากมาย หากต้องการให้เกมแสดงผลได้ดียิ่งขึ้น ควรปรับให้มีการทำ Dithering ซึ่งเป็นการจำลองให้สีมีจำนวนมากขึ้น โดยนำสีมี่มีอยู่มาวางจัดเรียงใกล้ๆ กัน ทำให้เกิดสีใหม่ขึ้น เหมือนกับหลักการผสมสี และควรปรับให้เกมมีการทำงานแบบ Bi-Linear Filtering ซึ่งจะทำการคำนวณให้ texture เนียนขึ้น



รูปที่ 7-5 กำหนดให้เกมมีการแสดงผลแบบ Dithering



รูปที่ 7-6 กำหนดให้เกมมีการแสดงผลแบบ Bi-Linear Filtering

เกมที่พัฒนาขึ้นมานี้ สามารถทำงานบนเครื่องคอมพิวเตอร์ที่มีฮาร์ดแวร์แตกต่างกันไปได้ เช่น หากนำเกมไปเล่นบนเครื่องคอมพิวเตอร์ที่มีการ์ดแสดงผลที่มีประสิทธิภาพสูง จะส่งผลทำให้เกมมีการแสดงผลที่ดีมากขึ้นตามไปด้วย คือ รูปภาพที่แสดงออกทางหน้าจอมีความสวยงาม สมจริง แสดงรายละเอียดของโมเดลที่เราสร้างขึ้นมาได้ครบถ้วนและเกมมีความเร็วในระดับพอสมควร แต่หากนำเกมนี้ไปเล่นบนเครื่องคอมพิวเตอร์ที่ฮาร์ดแวร์มีประสิทธิภาพไม่สูงนัก เกมก็ยังสามารถทำงานได้แต่อาจมีการแสดงผลในระดับที่ต่ำลง เช่น ความเร็วของเกมลดลง ความสวยงามและรายละเอียดของภาพที่แสดงออกมาจะมีประสิทธิภาพต่ำลงด้วย

8.3 แนวทางในการพัฒนางานวิจัยเพิ่มเติม

เกมที่พัฒนาขึ้นมานี้ ใช้เพียง Direct3D Retained Mode ของ DirectX เวอร์ชัน 6 ซึ่งเป็นโหมดการทำงานที่เหมาะสำหรับโปรแกรมเมอร์ที่ต้องการพัฒนาแอปพลิเคชั่นโดยรวดเร็ว แต่เกมที่อยู่ในท้องตลาดในปัจจุบันมักจะใช้ Direct3D Immediate Mode ซึ่งเป็นเครื่องมือชั้นเลิศให้กับผู้พัฒนาที่ต้องการเชื่อมโยงเกมหรือแอปพลิเคชั่นทางด้านมัลติมีเดียที่มีประสิทธิภาพสูงอื่นๆ เข้ากับระบบปฏิบัติการวินโดวส์ ดังนั้นหากต้องการพัฒนาเกมบนวินโดวส์จึงควรจะศึกษาและใช้เทคโนโลยีของ Immediate Mode จะดีกว่า ดังเห็นได้จาก DirectX SDK เวอร์ชันใหม่ๆ จะมุ่งเน้นที่การพัฒนาความสามารถของ Direct3D Immediate Mode มากขึ้น จึงมีฟังก์ชันการทำงานที่สนับสนุนการเขียนโปรแกรมทางด้านกราฟิกเพิ่มขึ้นมาอีกมากมาย

อีกแนวทางหนึ่งในการพัฒนาเกม สามารถทำได้โดยการศึกษา Game Engine ของบริษัทต่างๆ ซึ่งจะทำให้เราสามารถสร้างเกมได้ง่ายขึ้น เนื่องจาก Game Engine ในระดับประยุกต์เหล่านี้มีการออกแบบที่ครอบคลุมรายละเอียด เช่นการใช้ DirectX ไว้อยู่แล้ว การศึกษาทำความเข้าใจเครื่องมือต่างๆ เช่น เครื่องมือการสร้างฉาก การสร้างตัวละคร และการเขียนโปรแกรมเพื่อควบคุมรูปแบบเกม อาจมีความซับซ้อนและใช้เวลาสักหน่อย แต่เมื่อเราใช้ Game Engine ในการพัฒนาเกมจะทำให้ได้รับผลตอบแทนที่คุ้มค่า กล่าวคือเกมที่ได้มีประสิทธิภาพสูง หากเปรียบเทียบกับการพัฒนาเกมในโครงงานนี้ซึ่งจะต้องเขียนโปรแกรมควบคุมเกมโดยใช้คอมโพเนนท์ของ DirectX บางตัว จะต้องศึกษาและทำความเข้าใจสถาปัตยกรรมและการใช้งานเองทั้งหมด รวมถึงความแตกต่างของเครื่องมือต่างๆ ที่ใช้เช่น การสร้างโมเดลโดยโปรแกรม 3D Studio MAX แล้วจึงแปลงเป็นไฟล์ในรูปแบบที่นำมาใช้กับ DirectX ได้ อาจทำให้สูญเสียการเข้าถึงรายละเอียดของวัตถุหรือโมเดลบางอย่างไป เช่นเราจะรู้คุณสมบัติบางอย่างเท่านั้น เช่น ตำแหน่งหรือขนาดของออบเจ็กต์ แต่ถ้าหากใช้ Game Engine มักจะมีเครื่องมือในการสร้างฉากและตัวละคร รวมถึงเครื่องมือในการเขียนโปรแกรมเป็นของตนเอง ทำให้สามารถควบคุมเกมได้อย่างมีประสิทธิภาพยิ่งขึ้น

Game Engine ซึ่งเป็นเครื่องมือที่ถูกสร้างและพัฒนาจากการประมวลผลขั้นพื้นฐานที่มีความเกี่ยวข้องกันมาบรรจุไว้ด้วยกัน ซึ่งทำให้ง่ายและประหยัดเวลาในการใช้งาน ยกตัวอย่างเช่น Game Engine ระดับพื้นฐานทั้งสาม อันได้แก่ Microsoft DirectX SDK, OpenGL และ Glide ส่วน Game Engine ในระดับประยุกต์ ได้แก่ DGC, CDX, NUKE, FxEngine, Genesis3D และ Crystal Space เป็นต้น หากผู้ใดมีความสนใจต้องการใช้ Game Engine เหล่านี้มาใช้ในการพัฒนาสามารถหาข้อมูลได้จากเวปไซต์ของแต่ละบริษัท หรือเลือกดูรายชื่อ Game Engine ที่มีในปัจจุบันได้จาก http://www.cs.tu-berlin.de/~ki/engines.htm#eng_doom

บรรณานุกรม

[1] Andre LaMothe (1998) : "*Windows Game Programming for Dummies*", IDG Books Worldwide, Inc. , 1998.

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

[3] Stan Trujillo (1996) : " *Cutting Edge Direct3D Programming*" Coriolis Group Book,1996.

[4] ชัยวัฒน์ คำรัตน์ (1999) : "*3D Game Programming with DirectX*", Computer Age Technology Co,. Ltd. , 1999.