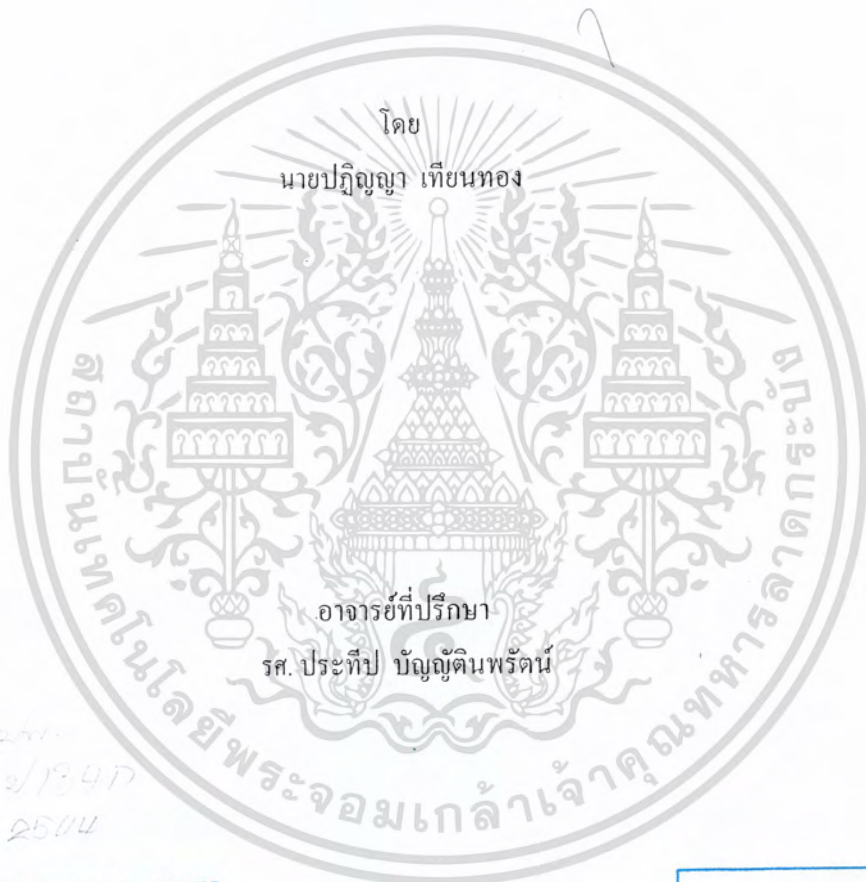


การพัฒนาซอฟต์แวร์เกม 3 มิติด้วยไคเร็กเอ็กซ์
3D GAME DEVELOPMENT USING DIRECTX



เลขหมู่.....
เลขทะเบียน... 46158
วัน, เดือน, ปี 20 ส.ค. 2546

b.....
i.....

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2544

ปริญญานิพนธ์ปีการศึกษา 2544

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาซอฟต์แวร์เกมคอมพิวเตอร์ 3 มิติด้วยโคเร็กเอ็กซ์

3D Game Development Using DirectX

ผู้จัดทำ

1. นาย ปริญญา เทียนทอง รหัสประจำตัว 41014239

 อาจารย์ที่ปรึกษา
(รศ. ประทีป บัญญัตินพรัตน์)



การพัฒนาซอฟต์แวร์เกม 3 มิติด้วยโคเร็กเอ็กซ์

นายปฏิญา เทียนทอง รหัส 41014239
 รศ. ประทีป บัญญัตินพรัตน์ อาจารย์ที่ปรึกษา
 ปีการศึกษา 2544

บทคัดย่อ

ปัจจุบันเทคโนโลยีทางด้านคอมพิวเตอร์ได้พัฒนาไปอย่างมาก ประสิทธิภาพของคอมพิวเตอร์ที่เพิ่มขึ้นนี้ทำให้เกิดการพัฒนาเกมที่แสดงผลด้วยกราฟิก 3 มิติ และกำลังเป็นที่นิยมอย่างมาก

อย่างไรก็ตาม, การพัฒนาซอฟต์แวร์เกม 3 มิติ เป็นงานที่ต้องใช้ความรู้หลากหลายด้านทั้งความรู้พื้นฐานที่เกี่ยวข้องกับคณิตศาสตร์ทั้ง 3 มิติและ 2 มิติ, ความรู้ทางด้านศิลปะในเรื่องของทฤษฎีเกี่ยวกับสี, แสง และวิธีการสร้างวัตถุที่จะนำมาใช้ และความรู้เกี่ยวกับเทคโนโลยีสำหรับการพัฒนาซอฟต์แวร์ ซึ่งความรู้ทั้งหมดนี้จะต้องประสานกันอย่างลงตัวเพื่อให้ได้ซอฟต์แวร์เกมที่ตรงกับความต้องการ

วิทยานิพนธ์ฉบับนี้จึงได้เสนอแนะแนวทางในการพัฒนาซอฟต์แวร์เกม 3 มิติ โดยรวบรวมเอาทฤษฎีทางคณิตศาสตร์ที่เกี่ยวข้องกับการพัฒนาซอฟต์แวร์เกม, วิธีการในการจัดการวัตถุต่างๆ ให้เหมาะสมกับการใช้งาน, เทคโนโลยีคอมพิวเตอร์ต่างๆ ที่จะช่วยให้สามารถพัฒนาซอฟต์แวร์เกมประสบความสำเร็จ เช่น โคเร็กเอ็กซ์, การพัฒนาซอฟต์แวร์เชิงวัตถุ

นอกจากนั้น ยังเสนอแนะเทคนิคต่างๆ ที่มักจะใช้ในการพัฒนาซอฟต์แวร์เกม เช่น การทำแอนิเมชัน, การใช้สร้างเงาของวัตถุ และการตรวจสอบการชนของวัตถุ

3D Game Development Using DirectX

Patinya Tiantong

41014239

Assoc. Prof. Pratreep Bunyuttinopparut Advisor

ABSTACT

Recently, computer technology is highly development. An increasing of computer's performance bring a development of game that use 3D graphics and it's currently most popular.

However, A development of 3D game is a task that use various of knowledge such as primary knowledge about mathematics in 2D and 3D, Art theorem that describe about color, light and how to making of resources to be used, and knowledge about software development technology that must cooperated to achieve a goal; a successful 3D game development.

This thesis is concerned with how to develop a 3D game software, by gathering knowledge of mathematics that relative to a development of game software, method to manage resources and how to prepare this resources for a further using, and computer technology that lead to a successful software development such as DirectX and Object Oriented Programming.

Moreover, Techniques that use frequently in development 3D game will be guided such as how to animate, how to make a shadow and how to detect an object collision.

กิตติกรรมประกาศ

วิทยานิพนธ์ฉบับนี้สำเร็จลงได้ด้วยความกรุณาของบุคคลหลายฝ่ายซึ่งให้ความรู้, ข้อมูลและกำลังใจทั้งจากเพื่อนๆ และคณาจารย์ โดยเฉพาะ อาจารย์ประทีป บัญญัติสินพรัตน์ เป็นที่ปรึกษาและคอยตรวจสอบผลงานในการพัฒนาของข้าพเจ้า ต้องขอขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณบิดาและมารดาอันเป็นที่เคารพของข้าพเจ้า ที่ให้กำลังใจช่วยเหลือและสนับสนุนข้าพเจ้าตลอดมา ผู้จัดทำระลึกในพระคุณ และขอกราบขอบพระคุณเป็นอย่างสูงมา ณ ที่นี้

ปริญญญา เทียนทอง



สารบัญ

	หน้าที่
บทคัดย่อภาษาไทย	I
บทคัดย่อภาษาอังกฤษ	II
กิตติกรรมประกาศ	III
สารบัญ	IV
สารบัญตาราง	VIII
สารบัญภาพ	IX
บทที่ 1 บทนำ	1
1.1 ความสำคัญและที่มา	1
1.2 จุดประสงค์ของโครงการ	2
1.3 ขอบเขตของโครงการ	2
1.4 ประโยชน์ที่คาดว่าจะได้รับ	2
1.5 วิธีดำเนินงาน	2
บทที่ 2 อุตสาหกรรมซอฟต์แวร์เกม	3
2.1 แนะนำอิเล็กทรอนิกส์เกม	3
2.2 ประวัติการพัฒนาของเกมคอมพิวเตอร์	3
2.3 นวัตกรรมในการพัฒนาเกมคอมพิวเตอร์	4
2.4 แพลตฟอร์มในการพัฒนาเกม	6
บทที่ 3 คณิตศาสตร์สำหรับการพัฒนาโปรแกรม 3 มิติ	8
3.1 ระบบพิกัดในปริภูมิ 3 มิติ	8
3.2 จุดหรือเวกเตอร์	8
3.3 โพลีกอน	10
3.4 ระนาบ	11
3.5 เมตริกซ์	12
3.6 ควอดเรอร์เนียน	15
บทที่ 4 การเขียนโปรแกรมบนระบบปฏิบัติการวินโดวส์และการใช้งานไลบรารีต้นแบบมาตรฐาน	17
4.1 การเขียนโปรแกรมบนระบบปฏิบัติการวินโดวส์	17
4.1.1 ชนิดของข้อมูลใน Win32 API	17
4.1.2 การกำหนดค่าแบบฮกกาเรียน	19
4.1.3 การเขียนโปรแกรมด้วย Win32 API	20
4.1.4 วินโดวส์เมสเสจ (Message) ที่มีการใช้งานบ่อยๆ	25
4.1.5 การเลือกใช้ไทม์เมอร์	26

สารบัญ (ต่อ)

	หน้าที่
4.2 Standard Template Library	27
4.2.1 องค์ประกอบพื้นฐานของ STL	27
4.2.2 แนวความคิดของ STL	27
4.2.3 คอนเทนเนอร์พื้นฐานใน STL ที่ใช้งานบ่อย	28
4.2.4 การใช้งาน STL	29
บทที่ 5 DirectX	30
5.1 แนะนำไคเร็กเอ็กซ์	30
5.1.1 ชุดพัฒนาซอฟต์แวร์สำหรับไคเร็กเอ็กซ์	30
5.2 การเตรียม Microsoft Visual C++ ให้พร้อมใช้งานกับ DirectX	31
บทที่ 6 ไคเร็กกราฟิก (DirectX Graphics)	33
6.1 แนะนำไคเร็กทรีดี	33
6.2 คุณสมบัติของไคเร็กทรีดี	33
6.3 สถาปัตยกรรมของไคเร็กทรีดี	34
6.3.1 สถาปัตยกรรมของเวอร์เท็กซ์เซตเตอร์	34
6.3.2 สถาปัตยกรรมของพิกเซลเซตเตอร์	35
6.4 เรนเดอร์ไปป์ไลน์ใน ไคเร็กทรีดี	36
6.4.1 การประมวลผลฟังก์ชันแบบตายตัว	36
6.4.2 การประมวลผลเวอร์เท็กซ์และพิกเซลแบบโปรแกรมได้	40
6.5 วัตถุไคเร็กทรีดี	40
6.6 ดีไวซ์ในไคเร็กทรีดี	41
6.7 การประมวลผลเวอร์เท็กซ์	46
6.8 การเรนเดอร์	46
6.9 สถานะของดีไวซ์	49
6.10 รีซอร์สใน ไคเร็กทรีดี	51
6.11 เซอร์เฟส	51
6.12 แสง	52
6.13 แมตทรีเรียล	58
6.14 รูปแบบของเวอร์เท็กซ์	61
6.15 เทกเจอร์	63
6.16 เดฟบัฟเฟอร์	63
6.17 เวอร์เท็กซ์บัฟเฟอร์	64

สารบัญ (ต่อ)

	หน้าที่
6.18 อินเด็กซ์บัพเฟอร์	67
6.19 D3DX Library	70
บทที่ 7 ไคเรกอินพุต (DirectInput)	71
7.1 แนะนำไคเรกอินพุต	71
7.1.1 ประสิทธิภาพของไคเรกอินพุต	71
7.1.2 เริ่มต้นกับไคเรกอินพุต	71
7.2 ทำความเข้าใจไคเรกอินพุต	72
7.3 การติดต่อกับวินโดวส์	73
7.4 การใช้งานไคเรกอินพุต	73
7.5 ไคเรกอินพุตดีไวซ์	75
7.6 ระดับการทำงานร่วมกัน	77
7.7 การระบุอุปกรณ์	78
7.8 รูปแบบข้อมูลของอุปกรณ์	79
7.9 ข้อมูลจากอุปกรณ์ไคเรกอินพุต	79
7.10 การสำรวจข้อมูล	80
บทที่ 8 ไคเรกออติโอ	81
8.1 ไคเรกเซาว์	81
8.2 อุปกรณ์ในไคเรกเซาว์	81
8.3 การระบุอุปกรณ์ในไคเรกเซาว์	81
8.4 การสร้างไคเรกเซาว์ออปเจกต์	82
8.5 ระดับการทำงานร่วมกัน	83
8.6 ความสามารถของอุปกรณ์	84
8.7 ไคเรกเซาว์บัพเฟอร์	84
8.8 การใช้งานข้อมูล Wave	86
บทที่ 9 ไคเรกโชว์ (DirectShow)	87
9.1 แนะนำไคเรกโชว์	87
9.2 เริ่มต้นการเขียนโปรแกรมด้วยไคเรกโชว์	87
9.3 การเขียนโปรแกรมโดยใช้ไคเรกโชว์	90
9.4 การเล่นไฟล์ (Play a File)	90
บทที่ 10 เทคนิคในการพัฒนาเกม 3 มิติ	92
10.1 เอนิเมชัน	92

สารบัญ (ต่อ)

	หน้าที่
10.1.1 การทำแอนิเมชันโดยใช้เทคนิคสปีเฟรมอินเทอร์โพเลชัน	92
10.1.2 การทำแอนิเมชันโดยใช้เทคนิคสกินนิ่ง	94
10.2 การตรวจสอบการชนของวัตถุ	98
10.2.1 วัตถุตรวจสอบการชนพื้นฐาน	98
10.2.2 การตรวจสอบการชนกับวัตถุแบบพื้น	98
10.2.3 การตรวจสอบการชนกับวัตถุแบบกำแพง	99
10.2.4 การตรวจสอบการชนกับวัตถุแบบทรงกระบอก	99
10.2.5 การตรวจสอบการชนกับวัตถุแบบทรงกลม	100
10.5 Light Map (Dark Map)	102
บทที่ 11 แนวความคิดและการออกแบบ	103
11.1 ความต้องการ	103
11.2 การออกแบบ	106
11.2.1 การพัฒนาแอปพลิเคชันที่จัดการกับข้อมูลจากแอปพลิเคชันสร้างโมเดล 3 มิติ	106
11.2.2 การพัฒนาเกมเอนจิน	108
11.2.2.1 ส่วนของการแสดงผลกราฟิก	108
11.2.2.2 เกมเอนจินในส่วนของการอินพุต	118
11.2.2.3 เกมเอนจินในส่วนของการเสียงและวีดีโอ	118
บทที่ 12 บทวิจารณ์และสรุป	120
12.1 สรุปผลการวิจัย	120
12.2 แนวทางการพัฒนาต่อ	120
ภาคผนวก	
ภาคผนวก ก. การติดตั้งและใช้งาน Plug-in	121
บรรณานุกรม	125

สารบัญตาราง

	หน้าที่
ตารางที่ 6-1 ตารางแสดงค่าโดยปริยายของแสง	57
ตารางที่ 6-2 ค่าโดยปริยายที่กำหนดให้กับเมตที่เรียล	60
ตารางที่ 6-3 ขนาดของข้อมูลที่เป็นองค์ประกอบของเวอร์เท็กซ์	66
ตารางที่ 7-1 แสดงค่าแสดงสถานะที่เป็นไปได้	78
ตารางที่ 11-1 วัตถุที่ได้จากการแยกแยะโดยใช้ข้อมูลที่เก็บสำหรับส่วนแสดงผล	110
ตารางที่ 11-2 ข้อมูลในโครงสร้างข้อมูล NGCREATEINFO	111
ตารางที่ 11-3 โครงสร้างในคลาส NG3D	112
ตารางที่ 11-4 การสร้างวัตถุของคลาส NG3D	113
ตารางที่ 11-5 ข้อมูลที่เก็บในโครงสร้างข้อมูล NGADAPTERINFO	114
ตารางที่ 11-6 โครงสร้างคลาส NGRenderer	115
ตารางที่ 11-7 ข้อมูลที่เก็บรักษาในโครงสร้างข้อมูล NGRENDERDESC	116
ตารางที่ 11-8 วัตถุที่ได้จากการแยกแยะโดยใช้ข้อมูลที่เก็บในส่วนของอินพุต	118
ตารางที่ 11-9 วัตถุที่ได้จากการแยกแยะโดยใช้ข้อมูลที่เก็บในส่วนของเสียงเอฟเฟ็ค	119
ตารางที่ 11-10 วัตถุที่ได้จากการแยกแยะโดยใช้ข้อมูลที่เก็บเสียงเพลง	119
ตารางที่ 11-11 วัตถุที่ได้จากการแยกแยะโดยใช้ข้อมูลที่เก็บในส่วนของวีดีโอ	119

สารบัญภาพ

หน้าที่

รูปที่ 3-1 ภาพแสดงระบบแกนในปริภูมิ 3 มิติ	8
รูปที่ 3-2 รูปแสดงความแตกต่างระหว่างจำนวนของรูปสามเหลี่ยมที่ใช้ในการสุ่มค่าจากวัตถุทรงกลม	10
รูปที่ 3-3 โครงสร้างข้อมูลที่ประกอบขึ้นมาเป็นพื้นผิว	10
รูปที่ 3-4 เวกเตอร์ที่ตั้งฉากกับเวอร์เท็กซ์และเวกเตอร์ที่ตั้งฉากกับพื้นผิว	11
รูปที่ 4-1 รูปแสดงขั้นตอนการทำงานในการเขียนเมสเสจรูปแบบทำงานตลอดเวลา	23
รูปที่ 6-1 โครงสร้างการทำงานของไดเรกกราฟิก	34
รูปที่ 6-2 สถาปัตยกรรมของเวอร์เท็กซ์เชดเดอร์	35
รูปที่ 6-3 สถาปัตยกรรมของพิกเซลเชดเดอร์	35
รูปที่ 6-4 เรนเดอร์ไปป์ไลน์ในไดเรกทรีดี	36
รูปที่ 6-5 รูปแสดงความสัมพันธ์ระหว่างตำแหน่งของโมเดลใน world coordinates	37
รูปที่ 6-6 รูปแสดงความสัมพันธ์ระหว่างตำแหน่งของโมเดลใน view coordinates	37
รูปที่ 6-7 รูปแสดง view frustum	38
รูปที่ 6-8 รูปแสดงการคำนวณค่า fov	38
รูปที่ 6-9 รูปแสดงวิธีการ projection	38
รูปที่ 6-10 รูปแสดงวิวพอร์ตและวิธีการตัด	39
รูปที่ 6-11 รูปแสดงขั้นตอนในการประมวลผลเวอร์เท็กซ์และพิกเซล	40
รูปที่ 6-12 รูปแสดงสถาปัตยกรรมการทำงานของดีไวซ์	41
รูปที่ 6-13 รูปแสดง Point List	42
รูปที่ 6-14 รูปแสดง Line List	43
รูปที่ 6-15 รูปแสดง Line Strip	43
รูปที่ 6-16 รูปแสดง Triangle List	43
รูปที่ 6-17 รูปแสดง Triangle Strip	44
รูปที่ 6-18 รูปแสดง Triangle Fan	44
รูปที่ 6-19 รูปแสดงเวอร์เท็กซ์ดาด้าสตรีม	48
รูปที่ 6-20 รูปแสดงความสัมพันธ์ของรีซอร์สในไดเรกทรีดี	51
รูปที่ 6-21 รูปแสดงความกว้างและค่าพิทช์	52
รูปที่ 6-22 รูปแสดงจุดแสง	53
รูปที่ 6-23 รูปแสดงสปอตไลต์	54
รูปที่ 6-24 รูปแสดงวิธีการคำนวณทางแสง	58
รูปที่ 6-25 รูปแสดงสเปกคิวลาร์	59
รูปที่ 6-26 รูปแสดงการเรียงลำดับการเก็บข้อมูลในเวอร์เท็กซ์	61

สารบัญภาพ (ต่อ)

หน้าที่

รูปที่ 6-27 รูปแสดงวิธีการคำนวณเดฟฟเฟอร์	63
รูปที่ 9-1 แสดงการเชื่อมต่อไฟล์เตอร์เพื่อเล่นไฟล์ AVI	88
รูปที่ 9-2 ขั้นตอนการสร้างไคเร็กโซว์	90
รูปที่ 10-1 รูปแสดงภาพเคลื่อนไหวที่เกิดจากเทคนิคคีย์เฟรมอินเทอร์โพลชัน	92
รูปที่ 10-2 รูปแสดงเส้นโค้ง Spline	93
รูปที่ 10-3 รูปแสดงโบนและจอยท์	94
รูปที่ 10-4 รูปแสดงการเปลี่ยนค่าที่เกิดจากการใช้เทคนิคสกินนิง	94
รูปที่ 10-5 รูปแสดงการอ้างอิงข้อมูลในเวอร์เท็กซ์ซึ่งชี้ไปยังเมตริกซ์	95
รูปที่ 10-6 รูปแสดงการอ้างอิงข้อมูลจากพื้นผิวไปยังเมตริกซ์	95
รูปที่ 10-7 รูปแสดงวัตถุตรวจสอบการชนแบบพื้นที่ได้จากฉาก	98
รูปที่ 10-8 รูปแสดงวัตถุตรวจสอบการชนแบบก้ำกึ่งที่ได้จากฉาก	99
รูปที่ 10-9 รูปแสดงวัตถุตรวจสอบการชนแบบทรงกระบอกรอบตัวละคร	100
รูปที่ 10-10 รูปแสดงวัตถุตรวจสอบการชนแบบทรงกลมที่สร้างขึ้นในส่วนที่ต้องการบนตัวละคร	101
รูปที่ 10-11 รูปแสดงการใช้ light map เพื่อเพิ่มแสงเงาให้กับฉาก	102
รูปที่ 11-2 ขั้นตอนการสร้างรีซอร์สที่ใช้ในการพัฒนาเกม	105
รูปที่ 11-3 ลำดับการเก็บข้อมูลในไฟล์รูปแบบ MD2	107
รูปที่ 11-4 คลาสไดอะแกรมในส่วนของการฝึก	110
รูปที่ 11-5 แสดงความสัมพันธ์ของคลาส NGModel และ NGClone	117
รูปที่ ก-1 แสดงไฟล์ที่จำเป็นในการติดตั้ง Plug-in สำหรับ Maya4.0	121
รูปที่ ก-2 แสดงเมนู “Window>Settings/Preferences>Plug-in Manager”	121
รูปที่ ก-3 แสดงไดอะล็อกบ็อกโหลด Plug-in	122
รูปที่ ก-4 แสดงไดอะล็อกบ็อกเขียนไฟล์ .X	122
รูปที่ ก-5 แสดงไฟล์ที่จำเป็นในการติดตั้ง Plug-in สำหรับ PhotoShop	123
รูปที่ ก-6 แสดงเมนู “File>Save As”	123
รูปที่ ก-7 แสดงไดอะล็อกบ็อกเขียนไฟล์ .DDS	124

บทที่ 1

บทนำ

1.1 ความสำคัญและที่มา

เกมหรือการละเล่นเป็นกิจกรรมที่อยู่คู่กับวิถีชีวิตของมนุษย์มาช้านาน แม้แต่ชนชาติไทยของเราก็มีเกมที่ใช้เล่นกันเพื่อความบันเทิงหรือในยามว่างเว้นจากการทำงานที่รู้จักกันดีก็เช่น หมากรุกไทย เป็นต้น แต่เมื่อวันเวลาผ่านไปความก้าวหน้าทางเทคโนโลยีมีมากขึ้น วิถีชีวิตของมนุษย์ก็เปลี่ยนแปลงไปตามกาลเวลาและเทคโนโลยีที่เปลี่ยนแปลงไปด้วยเช่นกัน จากยุคของเกมที่เป็นการเล่นพื้นบ้านมาสู่ยุคของอิเล็กทรอนิกส์เกม

เทคโนโลยีของคอมพิวเตอร์มีการพัฒนาก้าวหน้าไปอย่างรวดเร็วนับตั้งแต่ยุคของหลอดสุญญากาศ, ทรานซิสเตอร์ มาสู่ยุคของวงจรรวม (Integrated circuit) ในวงจรของการพัฒนาของคอมพิวเตอร์นี้เองทำให้เกิดเทคโนโลยีและอุตสาหกรรมต่างๆ ตามมามากมายไม่ว่าจะเป็นการพัฒนาทางด้านโทรคมนาคม, ด้านข้อมูลข่าวสาร ในส่วนของคอมพิวเตอร์เองก็มีการนำไปประยุกต์ในงานในด้านต่างๆ มากมาย ศาสตร์หลายๆ อย่างก็พึ่งถือกำเนิดเมื่อมีการพัฒนาเทคโนโลยีคอมพิวเตอร์นี้เอง เช่น การประมวลผลภาพด้วยคอมพิวเตอร์ (Image processing)

อิเล็กทรอนิกส์เกมเป็นการประยุกต์ใช้เทคโนโลยีทางด้านคอมพิวเตอร์มาใช้เพื่อความบันเทิง นับตั้งแต่การถือกำเนิดของเครื่องวีดีโอเกมคอนโซล Atari 2600 ซึ่งถือเป็นจุดกำเนิดของความสำเร็จในวงการเกมคอมพิวเตอร์จนถึงปัจจุบันกับความสำเร็จของ Sony PlayStation™ เทคโนโลยีเกมคอมพิวเตอร์ได้มีการพัฒนาขึ้นมามากมาย จนมีผู้ให้คำจำกัดความของกลุ่มอุตสาหกรรมนี้ใหม่ว่าเป็น “Computer Entertainment” ซึ่งไม่ได้มีความหมายครอบคลุมเฉพาะเกมคอมพิวเตอร์เท่านั้น แต่หมายถึงงานทางด้านเพลง (Music), ภาพยนตร์ (Movie) และเกมซึ่งถูกรวมเข้าด้วยกันอย่างพอเหมาะ ดังจะเห็นได้จากซอฟต์แวร์เกมคอมพิวเตอร์ในปัจจุบันนี้ไม่ได้มีเฉพาะส่วนของเกมที่ให้ผู้เล่นได้สนุกสนานเท่านั้น ยังมีส่วนของภาพยนตร์และเพลงที่ทำให้ผู้เล่นเกิดความประทับใจและเข้าใจเนื้อหาที่เกมต้องการจะสื่อออกมาอีกด้วย

Sony PlayStation™ และ Sega Saturn™ เป็นจุดเปลี่ยนแปลงที่มีผลกระทบต่อวงการเกมคอมพิวเตอร์เป็นอย่างมาก เนื่องจากเครื่องเกมคอนโซลทั้งสองเครื่องมีความสามารถในการประมวลผลภาพ 3 มิติ ซึ่งแตกต่างจากเครื่องคอมพิวเตอร์และเครื่องเกมคอนโซลอื่นๆ ในขณะนั้นซึ่งมีความสามารถในการประมวลผลภาพ 2 มิติเท่านั้น และความนิยมในเกม 3 มิติ ก็มีมากขึ้นเรื่อยๆ จนมีแนวโน้มว่าจะมาแทนเกมที่ใช้กราฟิกแบบ 2 มิติได้เกือบทั้งหมด เนื่องจากความสวยงามและสมจริงของภาพรวมทั้งความสามารถในการจำลองคุณสมบัติทางฟิสิกส์ของวัตถุ ซึ่งแนวโน้มนี้เองเป็นแรงผลักดันให้เกิดการพัฒนาเทคโนโลยีในการประมวลผลภาพ 3 มิติอย่างรวดเร็วทั้งในกลุ่มของเครื่องเกมคอนโซลและในเครื่องคอมพิวเตอร์ส่วนบุคคล

1.2 จุดประสงค์ของโครงการ

1. เพื่อรวบรวมและเสนอแนะแนวทาง, ทฤษฎี และเทคโนโลยีต่างๆ ที่ใช้ในการพัฒนาซอฟต์แวร์เกม 3 มิติ
2. พัฒนาซอฟต์แวร์เกม 3 มิติ โดยใช้วิธีการและเทคโนโลยีตามที่ได้ศึกษามา

1.3 ขอบเขตของโครงการ

โครงการนี้เป็นโครงการที่เสนอแนะวิธีการ, ขั้นตอน, ทฤษฎีและเทคโนโลยีต่างๆ โดยเฉพาะได้เร็กเอ็กซ์ เพื่อใช้ในการพัฒนาซอฟต์แวร์เกม 3 มิติ และแสดงตัวอย่างโดยการสร้างซอฟต์แวร์เกม 3 มิติขึ้นสำหรับทดสอบวิธีการที่ได้เสนอแนะ

1.4 ประโยชน์ที่คาดว่าจะได้รับ

- 1.4.1 มีความรู้เกี่ยวกับการประมวลผลภาพสามมิติ และเทคนิคอื่นๆที่เกี่ยวข้อง เช่น การสร้างภาพเคลื่อนไหว, การตรวจสอบการชนของวัตถุ เป็นต้น
- 1.4.2 มีความสามารถในการเขียนโปรแกรมด้วยไคเร็กเอ็กซ์และใช้งานไลบรารีอื่นๆ ที่เกี่ยวข้อง
- 1.4.3 มีความเข้าใจในขั้นตอน, วิธีการ และอุปสรรค รวมทั้งวิธีการแก้ไขปัญหาในการพัฒนาซอฟต์แวร์

1.5 วิธีดำเนินงาน

โครงการนี้เริ่มต้นด้วยการศึกษาเทคโนโลยีที่มีอยู่ในการทำงานด้านต่างๆ เช่น วิธีการในการตรวจสอบการชนของวัตถุว่ามีอะไรบ้าง แล้วเลือกเอาวิธีที่คิดว่าดีที่สุดมาสร้างเป็นองค์ประกอบหนึ่งในซอฟต์แวร์

โดยเริ่มต้นจากศึกษาข้อมูลของอุตสาหกรรมการผลิตซอฟต์แวร์เกมในปัจจุบัน แล้วจึงแนะนำฟังก์ชันทางคณิตศาสตร์สำคัญๆ ที่ใช้ในการประมวลผลภาพกราฟิก 3 มิติ และจึงกล่าวถึงวิธีการในการเขียนโปรแกรมบนระบบปฏิบัติการวินโดวส์เบื้องต้นรวมทั้งการใช้งาน STL ซึ่งช่วยในการเขียนโปรแกรมได้อย่างมาก จากนั้นจึงกล่าวถึงองค์ประกอบของ DirectX ในรายละเอียด แล้วจึงกล่าวถึงวิธีการใช้การใช้เทคนิคขั้นที่สูงขึ้นไป คือ การทำแอนิเมชันและการตรวจสอบการชนของวัตถุ

บทต่อมากล่าวถึงวิธีการสร้างรีซอร์สที่นำมาใช้ในโปรแกรม เช่น การสร้างข้อมูลโมเดล 3 มิติ และการสร้างข้อมูลรูปภาพ 2 มิติ

ในที่สุดท้ายจะเป็นรายละเอียดของโปรแกรมที่ได้พัฒนาไปแล้ว

บทที่ 2

อุตสาหกรรมซอฟต์แวร์เกม

2.1 แนะนำอิเล็กทรอนิกส์เกม

อิเล็กทรอนิกส์เกม (Electronic Games) คือ ซอฟต์แวร์ที่เล่นเพื่อความบันเทิง, ทำความ สามารถ หรือเพื่อใช้ในการศึกษา มี 2 ชนิดใหญ่ด้วยกันคือ วิดีโอเกม (Video game) คือ อุปกรณ์ที่ได้รับการออกแบบมาเพื่อเป็นระบบสำหรับการเล่นเกมโดยเฉพาะ อาจอยู่ในรูปของอุปกรณ์มือถือ (handheld device), เครื่องเล่นเกมที่ต่อเข้ากับโทรทัศน์ (video game console) หรือ เครื่องเล่นเกมสแน็คชอตเหรียญ (coin-operated arcade console) และอีกประเภทหนึ่งคือ เกมคอมพิวเตอร์ ซึ่งเล่นโดยใช้เครื่องคอมพิวเตอร์ส่วนบุคคล (personel computer)

อิเล็กทรอนิกส์เกมมีหลากหลายชนิดด้วยกัน เช่น เกมวางแผนการรบ (Strategy Game), เกมผจญภัย (Adventure/Exploration game), เกมจำลองการบิน (Flying simulation) และเกมภาษา (Role Playing Game) เป็นต้น นอกจากนั้นในปัจจุบันเกมยังมีการนำไปใช้ในการเรียนการสอนมากขึ้น เช่น การสอนการเขียน, การอ่าน, การแก้ไขปัญหา (problem-solving) และความสามารถพื้นฐานอื่นๆ ที่จำเป็นในการศึกษา

อิเล็กทรอนิกส์เกมได้รับการออกแบบมาเพื่อให้ใช้ความสามารถที่แตกต่างกันในการเล่น หลายๆ เกม เช่น Tetris และ Pac-Man ใช้ความสามารถในการทำงานประสานกันระหว่างมือและตาในการกำหนดตำแหน่งของวัตถุ ในเกมประเภทนี้ความท้าทายก็คือ การเล่นให้ได้ยาวนานที่สุดเท่าที่จะทำได้ในขณะที่เกมมีความยากหรือความเร็วมากขึ้น เกมอื่นๆ เช่น Super Mario Bros. มีความซับซ้อนยิ่งขึ้นไปอีก ผู้เล่นจะต้องใช้ความสามารถในการทำงานประสานกันระหว่างมือและตาในการกำหนดตำแหน่งของตัวละคร โดยโต้ตอบการกระทำต่างๆ ที่เกิดขึ้นบนจอภาพ นอกจากนั้น บางเกมยังการตัดสินใจที่รวดเร็วและถูกต้อง เช่น เกมผจญภัยซึ่งต้องแก้ไขปริศนาที่จะทำให้เกมดำเนินต่อไปได้ และเกมวางแผนการรบซึ่งผู้เล่นจะต้องใช้ความสามารถในการตัดสินใจที่ซับซ้อนยิ่งขึ้นไปอีกโดยมีเงื่อนไขของเวลาและตัวแปรที่มีความซับซ้อนยิ่งขึ้น บางเกมยังสามารถให้ผู้เล่นสามารถเล่นเกมร่วมกันได้โดยผ่านระบบเน็ตเวิร์ค เช่น LAN หรือ Internet แต่ก็มีเกมจำนวนมากที่คอมพิวเตอร์จะรับบทบาทเป็นผู้เล่นอีกฝ่ายหนึ่ง

2.2 ประวัติการพัฒนากองเกมคอมพิวเตอร์

เกมบนคอมพิวเตอร์ได้รับการพัฒนาขึ้นครั้งแรกในช่วงกลางทศวรรษ 1970 เริ่มต้นด้วยในรูปแบบของตัวอักษร (text adventure) ซึ่งเล่นผ่านระบบเน็ตเวิร์คในสถาบันวิจัยและสถาบันการศึกษา เกมเหล่านี้มันท้าทายให้ผู้เล่นในการไปถึงจุดหมายหรือการทำกิจกรรมบางอย่าง เช่น การหาไอเทมเวทมนต์ เป็นต้น ซึ่งในช่วงนั้นแรกจะอธิบายสถานการณ์ต่างๆ แล้วให้ผู้เล่นตอบสนองด้วยการพิมพ์คำสั่งต่างๆ คำสั่งที่พิมพ์ลงไปจะเป็นการนำไปสู่สถานการณ์อื่นๆ ต่อไป ตัวอย่างของเกมประเภทนี้ เช่น Zork

ในช่วงปลายของทศวรรษ 1970 Zork และเกมอื่นๆ ที่คล้ายคลึงกันได้เปลี่ยนมาเล่นกันบนเครื่องคอมพิวเตอร์ส่วนบุคคล ซึ่งในช่วงนั้นก็เป็นที่นิยมอย่างมาก เมื่อเทคโนโลยีมีการพัฒนามากขึ้น โปรแกรมเมอร์สามารถใช้กราฟิกต่างๆ ในเกมได้มากขึ้น แต่ตลาดของเกมคอมพิวเตอร์ก็ยังเติบโตช้าอยู่

เนื่องจากมีคนจำนวนไม่มากนักที่มีคอมพิวเตอร์ส่วนบุคคล ในช่วงกลางทศวรรษ 1980 มีการพัฒนาเกมขับเคลื่อนล้อรถที่ชื่อ Choplifter ซึ่งทำให้ตลาดเกมคอมพิวเตอร์ขยายตัวมากขึ้น หลังจากนั้นในปี 1982 บริษัทไมโครซอฟท์จึงผลิต Fight Simulator ซึ่งให้ผู้เล่นได้จำลองการบินในเครื่องบินด้วยเครื่องคอมพิวเตอร์ส่วนบุคคล

ในช่วงปีคริสต์ศักราช 1980 เครื่องคอมพิวเตอร์ส่วนบุคคลมีประสิทธิภาพมากขึ้น จึงมีเกมที่เริ่มมีความซับซ้อนผลิออกมาจำหน่าย เช่น เกมจากบริษัท Sierra On-Line, Electronics Arts และ Strategic Simulations, Inc. (SSI) นอกจากนี้ ยังมีเกมในซีรีส์ที่เรียกว่า Sim Game ออกวางจำหน่ายโดยบริษัท Maxis ซึ่งให้ผู้เล่นได้บริหารจัดการเมือง (SimCity), จัดการระบบนิเวศวิทยา (SimEarth) เป็นต้น

ในช่วงคริสต์ศักราช 1990 เครื่องคอมพิวเตอร์ส่วนบุคคลมีประสิทธิภาพที่สูงขึ้นมาก นอกจากนั้นยังมีการประดิษฐ์ CD-ROM ขึ้นมา ซึ่งทำให้เกมในขณะนั้นสามารถเก็บข้อมูลภาพและเสียงได้มากขึ้น ซึ่งเกมที่ได้ความนิยมมากในช่วงนั้น ได้แก่ Doom (ID Software) และ Myst (Broderbund)

หลังจากการพัฒนาระบบอินเทอร์เน็ตในช่วงกลางทศวรรษ 1990 เกมประเภทมัลติเพลเยอร์ก็เริ่มเป็นที่นิยมมากขึ้น เกมเหล่านี้มักจะเล่นกันบนระบบอินเทอร์เน็ตซึ่งมีผู้เล่นพร้อมๆ กันจำนวนมากเป็นร้อยเป็นพันคนโดยผู้เล่นทุกคนเล่นเกมในเวลาเดียวกัน ผู้เล่นจะต้องเข้าไปอยู่โลกที่ไม่ได้มีเพียงแต่ตัวละครที่สร้างขึ้นโดยคอมพิวเตอร์เท่านั้น ยังมีตัวละครที่ควบคุมด้วยผู้เล่นด้วยกันอีกด้วย

ช่วงปลายของศตวรรษที่ 1990 ดูเหมือนว่าคอมพิวเตอร์เกมจะไม่มีข้อจำกัดใดๆ เสียแล้ว

2.3 บุคลากรในการพัฒนาเกมคอมพิวเตอร์

ในการพัฒนาเกมคอมพิวเตอร์จะต้องใช้บุคลากรในด้านต่างๆ จำนวนมากซึ่งสามารถจำแนกออกเป็นประเภท ตามหน้าที่ได้ 5 กลุ่ม ด้วยกันดังนี้

1. วางแผนและกำกับ (Planning and Direction)

รับผิดชอบด้านการวางแผนโครงการ, สรรหาบุคลากร, ดูแลเรื่องเงินทุน, แก้ไขปัญหาต่างๆ ที่เกิดขึ้น และควบคุมการดำเนินงานให้เป็นไปตามแผนงานที่กำหนดเอาไว้ ประกอบด้วยงานในตำแหน่งของ

- ผู้ผลิต (Producer)
- ผู้กำกับ (Director)
- นักวางแผนหรือนักออกแบบเกม (Planner or Game Designer)
- นักเขียน (Writer or Scenario Writer)

2. ออกแบบกราฟิก (Graphics Design)

รับผิดชอบงานด้านกราฟิกต่างๆ เช่น การออกแบบตัวละครหรือฉาก, การสร้างรูปภาพที่จะนำมาทำเป็นฉากหลัง (background) หรือ ภาพที่ใช้เป็นเทกเจอร์ (texture) รวมทั้งการสร้างโมเดล 3 มิติและสร้างข้อมูลแอนิเมชัน (animation) ของโมเดล 3 มิติ ประกอบด้วยงานในตำแหน่งของ

- นักออกแบบฉากหลัง (Back Ground Designer)
- นักออกแบบตัวละคร (Character Designer)

- นักออกแบบโมเดล 3 มิติ (3D Modeling Designer)
- นักออกแบบเทกเจอร์ (Texture Designer)
- นักออกแบบการเคลื่อนไหว (Motion Designer or Animator)
- นักออกแบบเอฟเฟกต์ (Effect Designer)
- นักออกแบบภาพยนตร์ (Movie Designer)
- นักออกแบบสื่อสิ่งพิมพ์ (Publicity Designer)

3. โปรแกรม (Program)

รับผิดชอบในการเขียนโปรแกรมเป็นส่วนใหญ่ เช่น การเขียนโปรแกรมเพื่อช่วยในการทำงาน (tool), การเขียนโปรแกรมในส่วนแสดงภาพ 3 มิติของเกม เป็นต้น ประกอบด้วยงานในหน้าที่ของ

- วิศวกรระบบ (System Engineer)
- หัวหน้าโปรแกรมเมอร์ (Chief Programmer)
- ผู้ช่วยโปรแกรมเมอร์ (Assistant Programmer)
- โปรแกรมเมอร์ด้านเสียง (Sound Programmer)
- โปรแกรมเมอร์ระบบเครือข่าย (Network Programmer)
- โปรแกรมเมอร์สำหรับโปรแกรมช่วยเหลือต่างๆ (Tool Programmer)

4. เสียง (Sound)

รับผิดชอบงานที่เกี่ยวข้องกับเสียงที่ปรากฏในเกมทั้งหมด ไม่ว่าจะเป็นเสียงประกอบที่เป็นเสียงธรรมชาติ เช่น เสียงลม, เสียงฝนตก, เสียงแมลง เป็นต้น หรือเสียงของมนุษย์ เช่น เสียงเพลง, เสียงพูดต่างๆ รวมทั้งในส่วนเสียงเอฟเฟกต์ (sound effect) เช่น เสียงยิงปืน, เสียงระเบิด เป็นต้น ประกอบด้วยงานในหน้าที่ของ

- นักแต่งเพลง (Music Composer)
- นักออกแบบเสียงเอฟเฟกต์ (Sound Effect Designer)
- วิศวกรเสียง (Sound Engineer)

5. หน้าที่อื่นๆ (Other Tasks)

รับผิดชอบงานในส่วนอื่นๆ ซึ่งจะช่วยให้เกมมีความสมบูรณ์ยิ่งขึ้น หรือช่วยส่งเสริมในแง่ของการตลาด ประกอบด้วยงานในหน้าที่ของ

- ผู้ทดสอบเกม (Test Player or Debugger)
- ผู้โปรโมทผลงาน (Product Promoter)
- นักแสดงสำหรับบันทึกการเคลื่อนไหว (Actor/Actress for motion capture)
- นักพากย์ (Voice talent)
- นักออกแบบเว็บไซต์ (Web Designer)

2.4 แพลตฟอร์มในการพัฒนาเกม (Game Platform Development)

แพลตฟอร์ม (Platform) หมายถึง ฮาร์ดแวร์และระบบปฏิบัติการที่ใช้ในการทำงานของอุปกรณ์คอมพิวเตอร์นั้นๆ ซึ่งในการพัฒนาเกมสิ่งแรกที่ต้องคำนึงถึง คือ ฮาร์ดแวร์ที่จะพัฒนาเกมสำหรับ ซึ่งสามารถแบ่งได้ 2 กลุ่ม คือ

2.4.1 การพัฒนาเกมสำหรับวิดีโอเกม (Video Game)

ในกลุ่มของเครื่องวิดีโอเกมมีการพัฒนาเกม 3 มิติจำนวนมากในกลุ่มของเครื่องวิดีโอที่ต่อเข้ากับโทรทัศน์หรือที่เรียกว่า เครื่องวิดีโอเกมคอนโซล (Console) ซึ่งในปัจจุบันมีผลิตภัณฑ์ในกลุ่มนี้ที่ออกวางจำหน่ายแล้วและมีโครงการที่จะออกวางจำหน่าย ได้แก่ Sony PlayStation2, Nintendo GameCube และ Microsoft XBox

ตลาดของเครื่องวิดีโอเกมคอนโซลเป็นตลาดที่มีขนาดใหญ่สังเกตได้จากจำนวนของอุปกรณ์ที่ขายได้ เช่น Sony PlayStation2 ที่สามารถขายได้ประมาณ 25 ล้านเครื่อง ซึ่งอาจจะน้อยกว่าจำนวนของเครื่องคอมพิวเตอร์ทั่วโลกมากนักแต่ต้องไม่ลืมว่าเครื่องวิดีโอเกมคอนโซลทุกเครื่องมีคุณสมบัติเหมือนกัน ดังนั้น เกมที่พัฒนาให้จึงสามารถเล่นได้ทุกเครื่อง ในขณะที่ตลาดเกมของเครื่องคอมพิวเตอร์ส่วนบุคคลมีกลุ่มของเครื่องคอมพิวเตอร์จำนวนไม่มากนักที่มีความสามารถสูงพอที่จะเล่นเกมคอมพิวเตอร์ 3 มิติได้อย่างมีประสิทธิภาพ นอกจากนั้นยังมีความแตกต่างในแง่ของประเภทเกมที่เป็นที่นิยม และการพัฒนาโปรแกรมเกมอีกด้วย

การพัฒนาเกมสำหรับเครื่องวิดีโอเกมคอนโซลจำเป็นต้องใช้อุปกรณ์ฮาร์ดแวร์พิเศษในการพัฒนา เช่น การพัฒนาเกมให้เครื่องวิดีโอเกมคอนโซล Sony PlayStation2 จะต้องใช้ผลิตภัณฑ์ที่มีชื่อว่า PlayStation2 Development Tool DTL-T10000 ซึ่งมีราคาขายในประเทศญี่ปุ่น 2,000,000 เยน หรือเป็นเงินบาทประมาณ 740,000 บาท ซึ่งมีราคาแพงมากนอกจากนั้น การขายอุปกรณ์ยังขายในรูปแบบของบริษัทต่อบริษัทเนื่องจากจะต้องมีข้อสัญญาผูกมัดในการจ่ายค่าธรรมเนียม (royalty fee) เพื่อจำหน่ายเกมสำหรับอุปกรณ์นั้นๆ นอกจากนั้นวิธีการและขั้นตอนในการพัฒนายังมีการเก็บรักษาเป็นความลับเฉพาะบริษัทอีกด้วย

2.4.2 การพัฒนาสำหรับเครื่องคอมพิวเตอร์ส่วนบุคคล (Personnel Computer)

ในขณะที่การพัฒนาเกมบนเครื่องวิดีโอเกมคอนโซลมีข้อจำกัดมากมาย การพัฒนาเกมในเครื่องคอมพิวเตอร์ส่วนบุคคลก็ทำได้ง่ายและมีข้อจำกัดน้อยกว่า ในโครงการนี้จึงตัดสินใจพัฒนาเกมในกลุ่มของเครื่องคอมพิวเตอร์ส่วนบุคคล ซึ่งมีสิ่งที่จะต้องคำนึงถึงคือ ประสิทธิภาพในการทำงานส่วนใหญ่ขึ้นอยู่กับประสิทธิภาพของอุปกรณ์เร่งความเร็วในการแสดงผล 3 มิติ

2.4.3 ระบบปฏิบัติการ (Operationg System)

ระบบปฏิบัติการที่มีการใช้งานแพร่หลายที่สุดในโลก คือ ระบบปฏิบัติการวินโดวส์ ซึ่งผลิตขึ้นโดยบริษัทไมโครซอฟท์ ดังนั้น การพัฒนาเกมให้กับระบบปฏิบัติการนี้จึงเป็นทางเลือกที่ดีที่สุด ทั้งในแง่จำนวนกลุ่มผู้เล่นเกมที่เป็นเป้าหมายและข้อมูลในการพัฒนาที่มีจำนวนมากกว่าการพัฒนาเกมในระบบปฏิบัติการอื่นๆ เช่น ระบบปฏิบัติการลินุกซ์ซึ่งนิยมใช้งานด้านเน็ตเวิร์คเท่านั้น

2.4.4 Application Programming Interface

ในระบบปฏิบัติการวินโดวส์ API ที่ใช้ในการพัฒนาเกมมี 2 API ที่สำคัญ คือ ไคเร็กเอ็กซ์ (DirectX) ที่พัฒนาขึ้นโดยบริษัทไมโครซอฟท์ และโอเพนจีแอล (OpenGL) ซึ่งพัฒนาขึ้นโดยบริษัทซิลิกอนกราฟิกส์ (Silicon Graphics Inc. : SGI) แต่ไคเร็กเอ็กซ์ มีคุณสมบัติที่โดดเด่นกว่าโอเพนจีแอลในหลายๆ ด้าน คือ สนับสนุนการทำงานด้านเสียง, อุปกรณ์อินพุต และด้านเน็ตเวิร์ค เป็นต้น ในขณะที่โอเพนจีแอลสนับสนุนการทำงานเฉพาะด้านกราฟิกส์



บทที่ 3

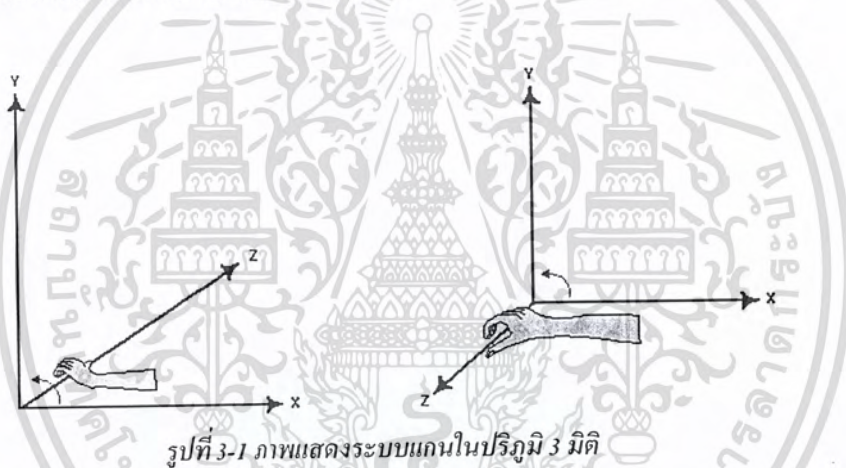
คณิตศาสตร์สำหรับการพัฒนาโปรแกรม 3 มิติ

3.1 ระบบคู่อันดับในปริภูมิ 3 มิติ (3D Coordinate Systems)

ระบบคู่อันดับที่ใช้ในโปรแกรม 3 มิติส่วนใหญ่เป็นระบบคู่อันดับคาร์ทีเซียน (Cartesian) ซึ่งมี 2 ระบบ คือ ระบบมือขวาและระบบมือซ้าย ระบบคู่อันดับทั้ง 2 ระบบนั้นแกน x ในส่วนที่มีค่าบวกจะชี้ไปทางขวามือ และแกน y ในส่วนที่มีค่าเป็นบวกก็จะชี้ขึ้นด้านบนเสมอ ส่วนทิศทางของแกน z สามารถหาได้โดยการใช้หลักมือซ้ายและมือขวาเช่นเดียวกับชื่อของระบบคู่อันดับแต่ละชนิด โดยถ้าเป็นระบบมือซ้ายก็ให้ใช้มือซ้ายและถ้าเป็นระบบมือขวาก็ให้ใช้มือขวา

วิธีหาทิศทางของแกน z

เริ่มต้นแบ่มือออกแล้วชี้ปลายนิ้วมือไปในทิศทางของแกน x ด้านที่มีค่าเป็นบวก หลังจากนั้นให้งอนิ้วมือจนอยู่ในทิศทางของแกน y ด้านที่มีค่าเป็นบวก ทิศทางของหัวมือคือทิศทางของแกน z ในด้านที่มีค่าเป็นบวกซึ่งจะมีทิศทางชี้เข้าหรือออกจากตัวเราเสมอ



รูปที่ 3-1 ภาพแสดงระบบแกนในปริภูมิ 3 มิติ

3.2 จุดหรือเวกเตอร์ (Point or Vector)

จุดหรือเวกเตอร์เป็นข้อมูลที่มีความสำคัญมากในกราฟิก 3 มิติ เนื่องจากข้อมูลต่างๆ ไม่ว่าจะเป็นเวอร์เท็กซ์ (Vertex) ซึ่งใช้เป็นข้อมูลในแสดงโมเดล, ตำแหน่งและทิศทางของวัตถุ และความเร็ว/แรง ทั้งหมดสามารถแทนค่าในรูปของจุดในระนาบ 3 มิติหรือเวกเตอร์ได้ทั้งหมด ในปริภูมิ 3 มิติวัตถุประกอบด้วยความกว้าง (width) คือ ค่าในแกน x, ความยาว (height) คือ ค่าในแกน y และความลึก (depth) คือ ค่าในแกน z

การแสดงค่าของจุดหรือเวกเตอร์

$$V = (x, y, z)$$

การแสดงค่าของจุดหรือเวกเตอร์ในรูปเมตริกซ์

$$V = \begin{bmatrix} x & y & z \end{bmatrix}$$

เนื่องจากคำว่าจุดและเวกเตอร์มีความหมายไปในทางเดียวกัน คือ

จุด หมายถึง ตำแหน่งในปริภูมิ 3 มิติ เวกเตอร์ หมายถึง เส้นที่ลากจากตำแหน่งอ้างอิง (จุดกำเนิด) มายังตำแหน่งในปริภูมิ 3 มิติ เวกเตอร์มีคุณสมบัติมากกว่าจุดคือ สามารถบอกทิศทางได้ ในที่นี้จึงใช้สลับกันได้ แต่ถ้าใช้ในการคำนวณจะกล่าวถึงในรูปของเวกเตอร์

การหาความยาวของเวกเตอร์

$$\|v\| = \sqrt{x^2 + y^2 + z^2}$$

การหาเวกเตอร์หนึ่งหน่วย (unit vector)

เวกเตอร์หนึ่ง คือ เวกเตอร์ที่มีขนาดหนึ่งหน่วยเสมอ ไม่ว่าจะอยู่ในทิศทางใดก็ตาม

$$u = \frac{v}{\|v\|}$$

มีเวกเตอร์หนึ่งหน่วย ที่ใช้แสดงทิศทางในแนวแกนต่างๆ ดังนี้

$$\text{แกน } x: i = (1, 0, 0)$$

$$\text{แกน } y: j = (0, 1, 0)$$

$$\text{แกน } z: k = (0, 0, 1)$$

เนื่องจากเวกเตอร์มีการนำไปใช้มากในทางฟิสิกส์ซึ่งมักจะเขียนเวกเตอร์ในรูปของค่าเวกเตอร์หนึ่งหน่วย i, j, k ทำให้เราสามารถแสดงค่าจุดในปริภูมิ 3 มิติในรูปของเวกเตอร์ได้โดยนำค่าในแต่ละแกนมาคูณกับเวกเตอร์หนึ่งหน่วยเหล่านี้ เช่น ถ้าต้องการแสดงจุด $a = (3, 5, 2)$ ในรูปของเวกเตอร์ จะได้

$$a = 3i + 5j + 2k$$

วิธีการแสดงค่าเวกเตอร์ในรูปเวกเตอร์หนึ่งหน่วยจะมีความสำคัญในการคำนวณค่ากับเมตริกซ์

ผลคูณเชิงสเกลาร์ของเวกเตอร์ (Scalar product or Dot product)

คือ ผลคูณของเวกเตอร์ 2 เวกเตอร์ที่ได้ผลลัพธ์เป็นสเกลาร์ ซึ่งค่าสเกลาร์ที่ได้แสดงค่าความยาวของเงาที่ฉายอยู่บนอีกเวกเตอร์หนึ่ง

ในระนาบ 2 มิติ

$$u \cdot v = u_x \times v_x + u_y \times v_y$$

$$u \cdot v = \|u\| \times \|v\| \times \cos(\theta)$$

ในระนาบ 3 มิติ

$$u \cdot v = u_x \times v_x + u_y \times v_y + u_z \times v_z$$

ผลคูณเชิงเวกเตอร์ของเวกเตอร์ (Cross Product)

คือ ผลคูณของเวกเตอร์ 2 เวกเตอร์ที่ได้ผลลัพธ์เป็นเวกเตอร์ ซึ่งเวกเตอร์ที่ได้จะตั้งฉากกับเวกเตอร์ทั้งสองเสมอ

$$u \times v = (u_y v_z - u_z v_y, u_z v_x - u_x v_z, u_x v_y - u_y v_x)$$

การคูณเชิงเวกเตอร์ของ 2 เวกเตอร์ไม่มีคุณสมบัติในการสลับที่ ดังนั้น $u \times v$ ไม่เท่ากับ $v \times u$ และถ้า 2 เวกเตอร์ขนานกันจะหาค่าผลคูณไม่ได้ (อาจจะได้เวกเตอร์ที่มีค่าอนันต์ (infinity) ซึ่งตั้งฉากกับเวกเตอร์หนึ่ง)

3.3 โพลีกอน (Polygon)

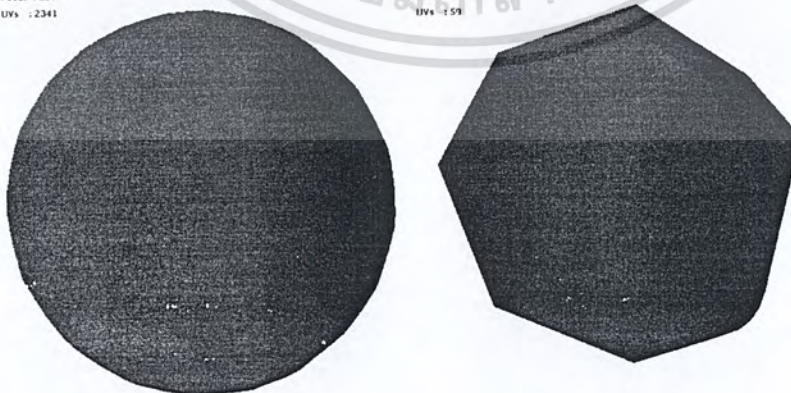
โพลีกอนเป็นพื้นฐานในสาขาวิชาคอมพิวเตอร์กราฟิก ถึงแม้ว่าในปัจจุบันจะมีเทคโนโลยีอย่างอื่น เช่น NURBS เป็นต้น แต่อุปกรณ์ที่ใช้กับเครื่องคอมพิวเตอร์ส่วนบุคคลทุกวันนี้ส่วนใหญ่ยังคงใช้หลักการวาดรูปด้วยโพลีกอน

ไคเร็กทรีดใช้โพลีกอนในการสร้างภาพของวัตถุ ซึ่งโพลีกอน ก็คือ พื้นผิวที่เกิดจากเส้นรอบรูปที่มีลักษณะเป็นวงปิด (โพลีกอนสนใจเฉพาะพื้นผิวไม่สนใจปริมาตรของวัตถุ) ซึ่งเส้นรอบรูปของโพลีกอนจะมีรูปร่างเป็นรูปทรงเรขาคณิต เช่น สามเหลี่ยม, สี่เหลี่ยม เป็นต้น หลักการในการสร้างรูปทรงของวัตถุโดยใช้โพลีกอนก็คือ การใช้พื้นผิวพื้นฐานที่สุด (primitive) ประกอบกันในการสร้างรูปทรงที่มีความซับซ้อนมากขึ้น และรูปทรงที่พื้นฐานที่สุดก็คือ รูปสามเหลี่ยม นั่นเอง ดังนั้นโดยอาศัยหลักการของโพลีกอนเราสามารถสร้างพื้นผิวของวัตถุที่มีรูปร่างใดๆ ก็ได้โดยไม่มีข้อจำกัด

อย่างไรก็ตาม รูปทรงที่สร้างขึ้นจากโพลีกอนจะเป็นการประมาณค่าพื้นผิวของวัตถุโดยใช้รูปสามเหลี่ยมสามเหลี่ยมมุมค่าจุดทั้งสามขึ้นมาจากพื้นผิวจริง ซึ่งถ้าจำนวนครั้งที่เรากลุ่มน้อยเกินไป (มีรูปสามเหลี่ยมน้อย) ก็อาจจะทำให้วัตถุที่มีความซับซ้อนสูงๆ มีการคาดเคลื่อนไป เช่น ถ้าเราใช้รูปสามเหลี่ยมจำนวนน้อยในการสร้างรูปทรงกลมจะทำให้ได้รูปหลายเหลี่ยมแทน

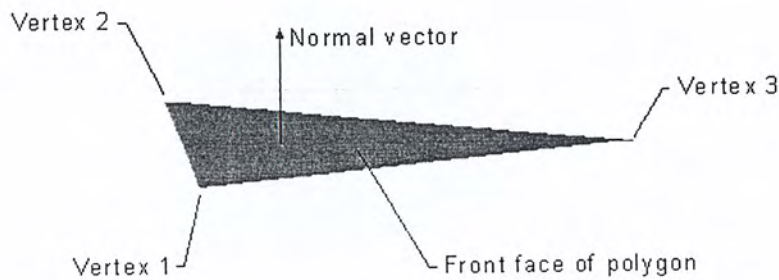
Verts : 2282
Edges : 4560
Faces : 2280
UVs : 2341

Verts : 40
Edges : 114
Faces : 76
UVs : 59



รูปที่ 3-2 รูปแสดงความแตกต่างระหว่างจำนวนของรูปสามเหลี่ยมที่ใช้ในการกลุ่มค่าจากวัตถุทรงกลม

โครงสร้างข้อมูลของ polygon



รูปที่ 3-3 โครงสร้างข้อมูลที่ประกอบขึ้นมาเป็นพื้นผิว

ในการสร้างรูปสามเหลี่ยม 1 รูป จะต้องใช้จุดสามจุดในการกำหนดตำแหน่งความยาวของด้านแต่ละด้านของโพลีกอน ซึ่งจุดบนโพลีกอนมีชื่อเรียกว่า เวอร์เท็กซ์เนื่องจากมันไม่ได้เก็บเฉพาะข้อมูลเกี่ยวกับตำแหน่งเท่านั้น แต่ยังประกอบด้วยข้อมูลอื่นๆ ที่ใช้กำหนดคุณลักษณะของพื้นผิวที่จะปรากฏอีกด้วย

ข้อมูลเหล่านั้นใคร่ครวญสามารถนำไปประมวลผลเพื่อสร้างภาพที่มีความสมจริงยิ่งขึ้นได้ เช่น ค่าเวกเตอร์ตั้งฉากของเวอร์เท็กซ์ (normal vector) สามารถนำไปใช้คำนวณมุมที่แสงกระทำต่อวัตถุทำให้ได้วัตถุที่มีสีส้มเหมือนจริงยิ่งขึ้น, ข้อมูลน้ำหนักในการเบลนด์ (blend weight data) ใช้ในการทำแอนิเมชันให้กับวัตถุ ซึ่งทำให้ได้วัตถุที่มีการเคลื่อนไหวอย่างเป็นธรรมชาติ และคู่อันดับของเทกเจอร์ (texture coordinate) ใช้กำหนดคู่อันดับของภาพที่จะปะบนวัตถุทำให้ได้วัตถุที่มีสีส้มของพื้นผิวเหมือนจริงและเป็นธรรมชาติมากขึ้น เป็นต้น



รูปที่ 3-4 เวกเตอร์ที่ตั้งฉากกับเวอร์เท็กซ์และเวกเตอร์ที่ตั้งฉากกับพื้นผิว

3.4 ระนาบ (Plane)

เป็นข้อมูลพื้นฐานที่สำคัญอีกอย่างหนึ่ง ระนาบในทาง 3 มิติก็มีความหมายเดียวกับเส้นตรงที่มีความยาวอนันต์ใน 2 มิติ หรือก็คือ สิ่งที่อยู่ในมิติ n-1 นั่นเอง ซึ่งจะแบ่งปริภูมิหรือระนาบเป็น 2 ส่วนแยกจากกันโดยเด็ดขาด สามารถนำไปใช้ในการคำนวณหลายๆ อย่างใน 3 มิติ เช่น การหาว่าจุดหรือโพลีกอนอยู่ในระนาบเดียวกันหรือไม่, ใช้ในการแบ่งโพลีกอนที่อยู่ในบริเวณที่จะแสดงผล (clipping) และใช้ในการคำนวณหา BSP Tree เป็นต้น

สมการระนาบในปริภูมิ 3 มิติ คือ

$$ax + by + cz + d = 0$$

นอกจากนั้น เรายังสามารถหาค่าอื่นๆ ที่เกี่ยวข้องกับระนาบได้ ดังนี้

1. ค่า (a, b, c) คือเวกเตอร์ที่ตั้งฉากกับระนาบ
2. ค่า d คือระยะทางตั้งฉากจากระนาบกับจุดกำเนิด
3. เมื่อนำค่าจุดใดๆ มาแทนค่าใน x, y, z สามารถหาว่าจุดนั้นอยู่ในระนาบหรือไม่ ถ้าอยู่ในระนาบจะได้ค่าผลลัพธ์เป็น 0

3.5 เมตริกซ์ (Matrix)

เมตริกซ์ช่วยให้เราสามารถแสดงสมการหลายๆ สมการได้พร้อมๆ กัน เช่น ถ้าเราต้องการจะแก้สมการที่มีตัวแปร x, y, z ดังนี้

$$3x - 8y + 12z = 0$$

$$15x + 14y - 2z = 0$$

$$32x + 0.5y - z = 0$$

เมื่อนำค่าสัมประสิทธิ์ของสมการมาใส่ในกรอบขนาด n คูณ m ที่เรียกว่า เมตริกซ์ เมื่อ n คือจำนวนของคอลัมน์ (vertical dimension) และ m คือ จำนวนของแถว (horizontal dimension) จะได้

$$\begin{aligned} 3x - 8y + 12z &= 0 \\ 15x + 14y - 2z &= 0 \\ 32x + 0.5y - z &= 0 \end{aligned} \Rightarrow \begin{bmatrix} 3 & -8 & 12 \\ 15 & 14 & -2 \\ 32 & 0.5 & 1 \end{bmatrix}$$

สัญลักษณ์ที่ใช้แทนเมตริกซ์ขนาด $m \times n$

$$A_{m \times n} = [a_{ij}]_{m \times n} = \begin{bmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \dots & a_{mn} \end{bmatrix}$$

การบวกเมตริกซ์

นิยาม

ถ้า $A = [a_{ij}]_{m \times n}$ และ $B = [b_{ij}]_{m \times n}$ แล้ว $A + B = [a_{ij} + b_{ij}]_{m \times n}$

ตัวอย่าง เช่น

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11} + b_{11} & a_{12} + b_{12} \\ a_{21} + b_{21} & a_{22} + b_{22} \end{bmatrix}$$

การคูณเมตริกซ์

นิยาม

ถ้า $A = [a_{ij}]_{m \times n}$ และ $B = [b_{ij}]_{n \times r}$ แล้ว ผลคูณ $A \times B$ คือ เมตริกซ์ $C = [c_{ij}]_{m \times r}$ โดยที่

$$c_{ij} = a_{i1}b_{1j} + a_{i2}b_{2j} + \dots + a_{in}b_{nj}$$

ตัวอย่าง เช่น

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} + \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix} = \begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

การคูณเมตริกซ์ไม่มีคุณสมบัติในการสลับที่ คือ $A \times B \neq B \times A$ ดังนั้น ต้องระวังในการคูณเมตริกซ์

ตัวอย่าง การคูณค่าของจุดกับเมตริกซ์

$$vA = v'$$

$$\begin{bmatrix} x & y & z \end{bmatrix} \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix} = \begin{bmatrix} x' & y' & z' \end{bmatrix}$$

เมตริกซ์มีการนำไปใช้ในสาขาคอมพิวเตอร์กราฟิกอย่างมาก เนื่องจากการใช้เมตริกซ์ทำให้เราสามารถทำการย้ายตำแหน่งของจุด, หมุน หรือย่อขยายได้อย่างรวดเร็วด้วยการนำจุดมาคูณหรือบวกกับเมตริกซ์เท่านั้น แต่การใช้เมตริกซ์ 3×3 จะทำให้ได้ค่าที่ไม่ถูกต้อง เนื่องจากความไม่เข้ากันของการคูณและการบวกในเมตริกซ์ ดังนั้นเพื่อแก้ปัญหานี้จึงต้องเพิ่มองค์ประกอบในเมตริกซ์ให้เป็นเมตริกซ์ขนาด 4×4 โดยองค์ประกอบที่เพิ่มขึ้นมา เรียกว่า homogenous coordinate ซึ่งแทนด้วยตัวอักษร w

ตัวอย่างการเพิ่ม homogenous coordinate ให้กับค่าจุดในระบบพิกัดคาร์ทีเซียน

$$\begin{bmatrix} x & y & z \end{bmatrix} \Rightarrow \begin{bmatrix} bx & by & bz & b \end{bmatrix}$$

โดย b จะเป็นค่าใดๆ ก็ได้ที่ไม่เท่ากับ 0 (ปกติจะใช้ $b = 1$)

การย้ายตำแหน่งจุด (translation)

ให้ $p = [p_x \ p_y \ p_z]$ เป็นเวกเตอร์แทนระยะทางที่ต้องการย้ายจุดไป จะได้เมตริกซ์สำหรับการย้ายจุด (translation matrix) ดังนี้

$$T(p) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p_x & p_y & p_z & 1 \end{bmatrix}$$

นั่นคือ จะต้องนำจุดที่ต้องการจะย้ายตำแหน่งมาคูณกับเมตริกซ์สำหรับการย้ายจุด ดังนี้

$$\begin{bmatrix} x & y & z & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ p_x & p_y & p_z & 1 \end{bmatrix} = \begin{bmatrix} x + p_x & y + p_y & z + p_z & 1 \end{bmatrix}$$

การหมุนตำแหน่งจุดรอบแกน (rotation)

การหมุนจะต้องทำโดยอ้างอิงกันแกนหมุน ซึ่งก็คือ แกน x, แกน y และแกน z

เมตริกซ์ที่ใช้ในการหมุน (rotation matrix) รอบแกน x

$$R_x(\theta) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\theta) & \sin(\theta) & 0 \\ 0 & -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

เมตริกซ์ที่ใช้ในการหมุน (rotation matrix) รอบแกน y

$$R_y(\theta) = \begin{bmatrix} \cos(\theta) & 0 & -\sin(\theta) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\theta) & 0 & \cos(\theta) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

เมตริกซ์ที่ใช้ในการหมุน (rotation matrix) รอบแกน z

$$R_z(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 & 0 \\ -\sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

โดย ค่า θ คือ มุมที่ต้องการหมุนจากแกนนั้นๆ

การย่อขยายกลุ่มของจุด (scale)

$$S = \begin{bmatrix} S_x & 0 & 0 & 0 \\ 0 & S_y & 0 & 0 \\ 0 & 0 & S_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

นอกจากการใช้เมตริกซ์เพื่อการย้ายตำแหน่งของจุด, การหมุนตำแหน่งของจุดและการย่อขยายตำแหน่งของจุด แล้วยังสามารถใช้ข้อมูลในเมตริกซ์ในการหาค่าอื่นๆ ได้อีก ดังนี้

เวกเตอร์ p แสดงความสัมพันธ์ของวัตถุกับจุดกำเนิด

เวกเตอร์ n, o และ a แสดงทิศทางในแต่ละแกนหรือในรูปเวกเตอร์หนึ่งหน่วย i, j และ k

ซึ่งค่าเวกเตอร์เหล่านี้สามารถหาได้จากเมตริกซ์ในตำแหน่งต่างๆ ดังนี้

$$\begin{bmatrix} n_x & n_y & n_z & 0 \\ o_x & o_y & o_z & 0 \\ a_x & a_y & a_z & 0 \\ p_x & p_y & p_z & 1 \end{bmatrix}$$

3.6 ควอเทอร์เนียน (Quaternion)

ควอเทอร์เนียนช่วยอำนวยความสะดวกในกระบวนการหมุนตำแหน่งของจุด และมีความเร็วมากกว่าการคำนวณโดยใช้เมทริกซ์ ประกอบด้วยองค์ประกอบ 2 ส่วน คือ ค่าเวกเตอร์และค่าสเกลาร์ ควอเทอร์เนียนสามารถเขียนในรูปต่างๆ ได้ ดังนี้

$$q = (x, y, z, w)$$

$$q = xi + yj + zk + w$$

$$q = [x \ y \ z \ w]$$

$$q = [sv]$$

เมื่อ x, y, z เป็นค่าสัมประสิทธิ์ของเวกเตอร์หนึ่งหน่วย i, j, k
 w เป็นค่าสเกลาร์

$$s = w \text{ และ } v = [x \ y \ z]$$

การคูณควอเทอร์เนียน (quaternion multiplication)

เนื่องจากควอเทอร์เนียนที่เราใช้เป็นสับเซตของควอเทอร์เนียนทั่วไป (General quaternion) ซึ่งเป็นการขยายความสามารถของจำนวนเชิงซ้อน (extension of complex number) ดังนั้น การคูณควอเทอร์เนียน จะต้องนำคุณสมบัติการคูณกันของจำนวนเชิงซ้อนมาใช้ด้วย ดังนี้

$$i \times i = -1$$

$$j \times j = -1$$

$$k \times k = -1$$

$$i \times j = -j \times i = k$$

$$j \times k = -k \times j = i$$

$$k \times i = -i \times k = j$$

เนื่องจากในระบบจำนวนจริง (real number) ค่า i, j และ k จะเป็นแกนในแนวต่างๆ นั้นเอง ดังนั้น การคูณควอเทอร์เนียนจึงอยู่ในรูปของการคูณจำนวนเชิงซ้อน ดังนี้

$$\begin{aligned} q_1 q_2 &= (w_1 + x_1 i + y_1 j + z_1 k)(w_2 + x_2 + y_2 + z_2) \\ &= (w_1 w_2 - x_1 x_2 - y_1 y_2 - z_1 z_2) + (y_1 z_2 - y_1 z_2 + w_1 x_2 + w_1 x_2) i + \\ &\quad (z_1 x_2 - z_1 x_2 + w_1 y_2 + w_1 y_2) j + (x_1 y_2 - x_1 y_2 + w_1 z_2 + w_1 z_2) k \end{aligned}$$

การหมุนตำแหน่งของจุดโดยใช้ควอเทอร์เนียน

การหมุนเวกเตอร์ $P = [x \ y \ z]$ ทำได้โดยการใช้ควอเทอร์เนียนบริสุทธิ์ (pure quaternion) p และคอนจูเกต (conjugate) ของ q

ควอเตอร์เนียนบริสุทธิ์ของ p

$$p = xi + yj + zk$$

ค่าของควอเตอร์เนียนที่ใช้ในการหมุนรอบแกน $A(x, y, z)$ ด้วยมุม θ คือ

$$q = (sx, sy, sz, c)$$

เมื่อ

$$s = \sin(\theta/2)$$

$$c = \cos(\theta/2)$$

คอนจูเกตของควอเตอร์เนียน

$$q^* = [-x \ -y \ -z \ w]$$

การหมุนตำแหน่งจุดโดยใช้ควอเตอร์เนียน

$$Rot_q(P) = qpq^*$$



บทที่ 4

การเขียนโปรแกรมบนระบบปฏิบัติการวินโดวส์และการใช้งานไลบรารีต้นแบบมาตรฐาน

4.1 การเขียนโปรแกรมบนระบบปฏิบัติการวินโดวส์

การเขียนโปรแกรมให้ทำงานบนระบบปฏิบัติการวินโดวส์แตกต่างจากการเขียนโปรแกรมบนระบบปฏิบัติการดอส (DOS) อย่างมาก ในระบบปฏิบัติการดอสมีการทำงานของโปรแกรมได้เพียงโปรแกรมเดียวในขณะหนึ่ง หรือสามารถใช้เวลาการทำงานของโปรเซสเซอร์ (processing time) ทั้งหมดได้ และสามารถควบคุมอุปกรณ์ในเครื่องคอมพิวเตอร์ได้ทั้งหมด แต่ในขณะเดียวกันผู้เขียนโปรแกรมก็ต้องตรวจสอบอุปกรณ์ต่างๆ ที่มีอยู่ในเครื่องคอมพิวเตอร์เองด้วย ซึ่งเป็นงานที่ยากลำบากมากเนื่องจากอุปกรณ์คอมพิวเตอร์มีจำนวนมากมาย ในขณะที่การเขียนโปรแกรมบนระบบปฏิบัติการวินโดวส์จะต้องแบ่งการใช้งานโปรเซสเซอร์กับโปรแกรมอื่นๆ และไม่สามารถเข้าไปควบคุมการทำงานของอุปกรณ์ที่มีอยู่ในเครื่องได้โดยตรง แต่ระบบปฏิบัติการวินโดวส์จะช่วยให้เราเขียนโปรแกรมได้สะดวกขึ้นเนื่องจากระบบปฏิบัติการได้จัดการงานที่สลับซับซ้อนให้เราทั้งหมดแล้ว เช่น การตรวจสอบอุปกรณ์ที่มีอยู่ในเครื่อง เป็นต้น นอกจากนั้น การทำงานหลายๆ อย่างก็สะดวกขึ้น เช่น ถ้าเราต้องการเล่นเพลงก็ใช้คำสั่งเพียง PlaySound ในขณะที่การเขียนโปรแกรมบนระบบปฏิบัติการดอสจะต้องทำงานที่สลับซับซ้อนกว่านี้มาก โดยการเขียนโปรแกรมบนวินโดวส์จะต้องใช้ Win32 API (Application Programming Interface)

Win32 API คือ ชุดคำสั่งในภาษาซีที่เราเขียนโปรแกรมให้ทำงานบนระบบปฏิบัติการวินโดวส์ได้ ซึ่งได้จัดการงานที่มีสลับซับซ้อนหลายอย่างให้กับเราแล้ว เช่น การทำงานแบบมัลติทาสกิ้ง (multi-tasking), การป้องกันหน่วยความจำ (memory protection) เป็นต้น นอกจากนั้น ยังมีส่วนของอินเทอร์เฟซ (interface) ที่ช่วยในการแสดงผลแบบ GUI (Graphics User Interface) เช่น ไดอะล็อกบ็อกซ์ (dialog box), เมนู (menu) และงานเกี่ยวกับมัลติมีเดีย (multimedia) ซึ่งชุดคำสั่งเหล่านี้จะทำให้เราทำงานได้ง่ายขึ้นมาก

API ในระบบปฏิบัติการวินโดวส์มีหลายกลุ่มด้วยกัน และสำหรับงานแต่ละชนิดก็มีการทำงานแตกต่างกันในที่นี้จะนำเสนอแต่เฉพาะในส่วนที่มีความจำเป็นในการใช้งานเท่านั้น

4.1.1 ชนิดของข้อมูลใน Win32 API (Win32 Data Types)

นอกจากชนิดของข้อมูลที่มีอยู่ในภาษา C และ C++ แล้วใน Win32 API ยังได้กำหนดชนิดของข้อมูลขึ้นมาอีกจำนวนหนึ่ง ซึ่งสามารถนำไปใช้ในการกำหนดค่าส่งคืนของฟังก์ชัน (function return value), พารามิเตอร์ของฟังก์ชันและเมสเสจ (function and message parameter) และเป็นสมาชิกในข้อมูลอื่น (structure member) ซึ่งต่อไปนี้เป็น ชื่อและขนาดของข้อมูลเหล่านี้บางส่วนที่นำมาใช้งาน

BOOL	ค่าตัวแปรบูลีน (มีค่าเป็น TRUE หรือ FALSE)
BOOLEAN	ค่าตัวแปรบูลีน (มีค่าเป็น TRUE หรือ FALSE)
BYTE	ไบต์ (byte) มีขนาด 8 บิต (bit)
CALLBACK	ระเบียบในการเรียกใช้สำหรับฟังก์ชันเรียกกลับ (calling convention for callback function)

CHAR	ตัวอักษรขนาด 8 บิต ในระบบปฏิบัติการวินโดวส์ ในตัวอักษรในระบบ ANSI
COLORREF	ค่าสีเขียว, ค่าสีฟ้า และค่าสีแดง (RGB) มีขนาด 32 บิต
DLGPROC	ค่าพอยเตอร์ (พอยเตอร์) ที่ชี้ไปยังโคออดิเนตบ็อกซ์ที่กำหนดไว้ในฟังก์ชันเรียกกลับ
DWORD	จำนวนเต็มแบบไม่คิดเครื่องหมายขนาด 32 บิต (32 bit unsigned integer)
FLOAT	จำนวนทศนิยมขนาด 32 บิต
HANDLE	ค่าแฮนเดิล (handle) ของวัตถุ
HBITMAP	ค่าแฮนเดิล (handle) ของบิตแมพ (bitmap)
HDC	ค่าแฮนเดิล (handle) ของคันทรีบริคบริค (device context : DC)
HFILE	ค่าแฮนเดิล (handle) ของไฟล์
HFONT	ค่าแฮนเดิล (handle) ของฟอนต์ (font)
HICON	ค่าแฮนเดิล (handle) ของไอคอน (icon)
HINSTANCE	ค่าแฮนเดิล (handle) ของอินสแตนซ์ (instance)
HMENU	ค่าแฮนเดิล (handle) ของเมนู (menu)
HTREEITEM	ค่าแฮนเดิล (handle) ของไอเทม (item) ในทรีวิวคอนโทรล (tree-view control)
HWND	ค่าแฮนเดิล (handle) ของวินโดวส์
LONG	ค่าจำนวนเต็มแบบคิดเครื่องหมายขนาด 32 บิต (32-bit signed integer)
LONGLONG	ค่าจำนวนเต็มแบบคิดเครื่องหมายขนาด 64 บิต (64-bit signed integer)
LPARAM	ค่าพารามิเตอร์ของเมสเสจ (message parameter)
LPBOOL	ค่าพอยเตอร์ที่ชี้ไปยังข้อมูลชนิด BOOL
LPBYTE	ค่าพอยเตอร์ที่ชี้ไปยังข้อมูลชนิด BYTE
LPCTSTR	ค่าพอยเตอร์ที่ชี้ไปยังสตริงที่จบด้วยค่าศูนย์ (null-terminated string) ขนาด 8 บิต ตัวอักษรใช้แบบ ANSI
LPDWORD	ค่าพอยเตอร์ที่ชี้ไปยังข้อมูลชนิด DWORD
LPINT	ค่าพอยเตอร์ที่ชี้ไปยังข้อมูลชนิด INT
PLONG	ค่าพอยเตอร์ที่ชี้ไปยังข้อมูลชนิด LONG
LPSTR	ค่าพอยเตอร์ที่ชี้ไปยังสตริงที่จบด้วยค่าศูนย์ (null-terminated string) ขนาด 8 บิต ตัวอักษรใช้แบบ ANSI
LPVOID	ค่าพอยเตอร์ที่ชี้ไปยังข้อมูลชนิดใดๆ
LPWORD	ค่าพอยเตอร์ที่ชี้ไปยังข้อมูลชนิด WORD
LRESULT	ผลลัพธ์ที่คิดเครื่องหมายสำหรับการประมวลผลเมสเสจ (signed result of message processing)
MSG	โครงสร้างของเมสเสจ

RECT	โครงสร้างของพื้นที่สี่เหลี่ยม
TCHAR	จะเป็น WCHAR ถ้ากำหนดว่าใช้ UNICODE ถ้าไม่ได้กำหนดจะเป็น CHAR
UINT	INT แบบไม่คิดเครื่องหมาย
ULONG	LONG แบบไม่คิดเครื่องหมาย
ULONGLONG	จำนวนเต็มแบบไม่คิดเครื่องหมายขนาด 64 บิต (64-bit unsigned integer)
VOID	ชนิดของข้อมูลใดๆ
WINAPI	ระเบียบในการเรียกใช้สำหรับ Win32 API (calling convention for Win32 API)
WNDCLASSEX	โครงสร้างของวินโดว์คลาส (Window class)
WNDPROC	ค่าพอยเตอร์ที่ชี้ไปยังวินโดว์โปรซีเจอร์ (Window procedure) ที่กำหนดไว้โดยโปรแกรม)
WORD	จำนวนเต็มแบบไม่คิดเครื่องหมายขนาด 16 บิต (16-bit unsigned integer)
LPARAM	ค่าพารามิเตอร์ของเมสเสจขนาด 32 บิต (32-bit message parameter)

4.1.2 การกำหนดค่าแบบฮังการี (Hungarian Notation)

ในการตั้งชื่อตัวแปรเพื่อที่จะให้ผู้ใช้้ง่ายต่อการจดจำชนิดข้อมูลของตัวแปรนั้นๆ มีรูปแบบในการกำหนดชื่อหลายแบบ แต่ที่นิยมกันในหมู่โปรแกรมเมอร์ที่เขียนโปรแกรมบนระบบปฏิบัติการวินโดว์ คือการกำหนดแบบฮังการี ซึ่งก็คือการกำหนดชื่อตัวแปรโดยใช้ตัวอักษรย่อของชนิดข้อมูลนำหน้าชื่อตัวแปร โดยมีการกำหนดดังตัวอย่าง

คำแนะนำ	ชนิดของข้อมูล
c	char, CHAR
b	BYTE
n	SHORT
i	int, INT
x, y	int, INT (x, y - coordinate)
cx,cy	int, INT (c - count)
b	BOOL
w	WORD
l	LONG
dw	DWORD
fn	function
str	string (STL)
sz	string terminated by 0 byte
h	HANDLE

p	พอยเตอร์
f	FLOAT
d3d	Direct3D Interface
lp	long พอยเตอร์

4.1.3 การเขียนโปรแกรมด้วย Win32 API

4.1.3.1 ฟังก์ชันจุดเริ่มต้น (The Entry-Point Function)

ในโปรแกรมที่มีพื้นฐานอยู่บน Win32 จะต้องมียุติจุดเริ่มต้น ซึ่งโดยทั่วไปจะมีชื่อว่า ฟังก์ชัน WinMain ซึ่งภายในฟังก์ชันนี้จะต้องประกอบด้วยขั้นตอนการทำงานดังนี้

4.1.3.2 การกำหนดค่าวินโดว์คลาส (Registering the window class)

ทุกๆ วินโดว์จะต้องมีวินโดว์คลาส (window class) ซึ่งเป็นตัวกำหนดคุณสมบัติ (attribute) ของวินโดว์ เช่น รูปแบบ (style), ไอคอน (icon), ชื่อของเมนู และชื่อของวินโดว์โพรซีเจอร์ (window procedure)

ขั้นตอนแรก คือ การใส่ค่าต่างๆ ในโครงสร้างข้อมูล WNDCLASS หรือ WNDCLASSEX ด้วยข้อมูลที่ต้องการ หลังจากนั้นจึงส่งค่าโครงสร้างข้อมูลที่ใส่ค่าเรียบร้อยแล้วให้ฟังก์ชัน RegisterClass ในกรณีที่ใช้โครงสร้างข้อมูล WNDCLASS และฟังก์ชัน RegisterClassEx ในกรณีที่ใช้โครงสร้างข้อมูล WNDCLASSEX เช่น

```

WNDCLASSEX wcex;
wcex.cbSize      = sizeof(WNDCLASSEX);
wcex.style       = 0;
wcex.lpfnWndProc  = (WNDPROC)WndProc;
wcex.cbClsExtra   = 0;
wcex.cbWndExtra   = 0;
wcex.hInstance   = hInstance;
wcex.hIcon       = LoadIcon(hInstance, (LPCTSTR)IDI_MAIN);
wcex.hCursor     = LoadCursor(NULL, IDC_ARROW);
wcex.hbrBackground = (HBRUSH)(COLOR_BTNFACE + 1);
wcex.lpszMenuName = 0;
wcex.lpszClassName = strClass;
wcex.hIconSm     = LoadIcon(wcex.hInstance, (LPCTSTR)IDI_SMALL);
RegisterClassEx(&wcex);

```

4.1.3.3 การสร้างวินโดว์หลัก (main window)

สามารถสร้างวินโดว์หลักได้ด้วยการเรียกฟังก์ชัน `CreateWindow` หรือ ฟังก์ชัน `CreateWindowEx` ดังนี้

```
hWnd = CreateWindowEx(    WS_EX_TOPMOST,
                          strClass, strName,
                          WS_OVERLAPPEDWINDOW,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          CW_USEDEFAULT, CW_USEDEFAULT,
                          NULL, NULL,
                          hInstance, NULL);
```

พารามิเตอร์ตัวที่สอง `strClass` คือ ชื่อของวินโดว์คลาสที่เราได้กำหนดเอาไว้แล้ว พารามิเตอร์อื่นๆ เป็นตัวกำหนดคุณสมบัติของวินโดว์ที่จะแสดงออกทางหน้าจอ แต่วินโดว์จะยังไม่แสดงออกหน้าจอจนกว่าจะเรียกฟังก์ชัน `ShowWindow` ดังนี้ `ShowWindow(m_hWnd, SW_NORMAL);`

4.1.3.4 การเข้าสู่เมสเสจลูป (Entering The Message Loop)

เมื่อสร้างวินโดว์หลักและสั่งให้แสดงผลออกทางหน้าจอแล้ว ฟังก์ชัน `WinMain` ก็จะเริ่มต้นทำหน้าที่หลักของมันคือ การออกเมสเสจจากแอปพลิเคชันคิว (application queue) และส่ง (dispatch) ให้กับวินโดว์ที่เหมาะสม

ระบบปฏิบัติการวินโดว์ไม่ได้ส่งข้อมูลให้กับโปรแกรมโดยตรง แต่จะส่งข้อมูลจากผู้ใช้ไว้ในเมสเสจคิว (message queue) เช่นเดียวกับเมสเสจที่ถูกส่งมาจากวินโดว์หรือโปรแกรมอื่นๆ โปรแกรมมีหน้าที่ที่จะต้องอ่านข้อมูลในเมสเสจคิว, รับเมสเสจเหล่านั้นไป และส่งให้กับวินโดว์โปรซีเดอร์ที่จะประมวลผลเมสเสจเหล่านั้นต่อไป

การเขียนฟังก์ชันที่ทำงานในเมสเสจลูปแบบทั่วไป

```
while(GetMessage(&msg, NULL, 0, 0))
```

```
{
    TranslateMessage(&msg);
    DispatchMessage(&msg);
}
```

`GetMessage` เป็นฟังก์ชันที่รับเมสเสจมาจากคิวและจะรอจนกว่าจะมีเมสเสจเข้ามาจึงจะส่งค่ากลับคืน (return)

`DispatchMessage` เป็นฟังก์ชันที่ส่งแต่ละเมสเสจไปยังวินโดว์โปรซีเดอร์ที่เหมาะสม

`TranslateMessage` เป็นฟังก์ชันที่แปลงเวอร์ชวลคีย์เมสเสจ (virtual-key message) ไปเป็นเมสเสจตัวอักษร ซึ่งมีความจำเป็นในการสร้างคีย์ในการเรียกเมนู (menu access key)

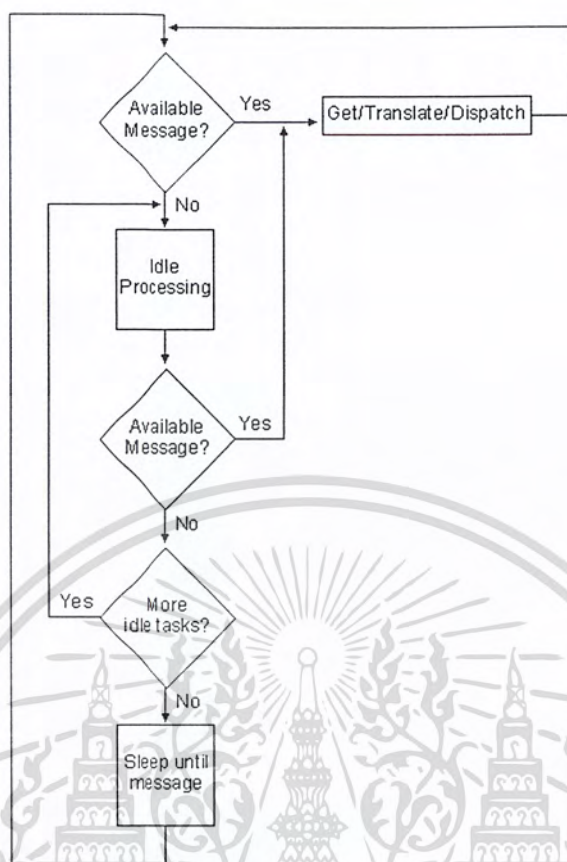
แต่โปรแกรมเกมโดยปกติแล้วจะทำงานตลอดเวลาเนื่องจากการเรนเดอร์ (render) จะต้องทำตลอดเวลาที่โปรแกรมทำงานอยู่แม้แต่ในเวลาที่ไม่มีเมสเสจของโปรแกรมเข้ามาด้วย ดังนั้น จึงไม่สามารถใช้ฟังก์ชัน `GetMessage` ได้ ต้องเขียนอีกริธีหนึ่งดังนี้

4.1.3.5 การเขียนเมสเสจรูปแบบที่สามารถทำงานได้ตลอดเวลาสำหรับการเรนเดอร์

`PeekMessage(&msg, NULL, 0U, 0U, PM_NOREMOVE);`

```
while(WM_QUIT != msg.message)
{
    // Use PeekMessage() if the app is active, so we can use idle time to
    // render the scene. Else, use GetMessage() to avoid eating CPU time.
    if (bActive)
        bGotMsg = PeekMessage(&msg, NULL, 0U, 0U, PM_REMOVE);
    else
        bGotMsg = GetMessage(&msg, NULL, 0U, 0U);
    if (bGotMsg)
    {
        TranslateMessage(&msg);
        DispatchMessage(&msg);
    }
    else
    {
        // Render a frame during idle time (no messages are waiting)
    }
}
```

`PeakMessage` เป็นฟังก์ชันที่รับเมสเสจมาจากคิวแต่แตกต่างจากฟังก์ชัน `GetMessage` ก็คือจะส่งค่ากลับคืน (return) มาทันทีโดยไม่รอว่ามีเมสเสจเข้ามาหรือไม่



รูปที่ 4-1 รูปแสดงขั้นตอนการทำงานในการเขียนเมสเสจรูปแบบทำงานตลอดเวลา

4.1.3.6 วินโดว์โปรซีเจอร์ (Window Procedure)

ทุกวินโดว์ (โปรแกรมหนึ่งๆ สามารถมีได้หลายวินโดว์) จะต้องมีวินโดว์โปรซีเจอร์ ซึ่งชื่อของวินโดว์โปรซีเจอร์สามารถตั้งได้ตามต้องการ (user-defined) ตัวอย่างของวินโดว์โปรซีเจอร์

```
LRESULT CALLBACK WndProc(HWND, UINT, WPARAM, LPARAM);
```

มีการใช้ตัวแปลง (modifier) CALLBACK เนื่องจากวินโดว์โปรซีเจอร์ต้องประกาศ (declare) โดยเรียกการเรียกอำนวยความสะดวก (calling convention) เสมอ

วินโดว์โปรซีเจอร์รับเมสเสจจากระบบปฏิบัติการวินโดว์ ซึ่งเมสเสจเหล่านั้นอาจจะเป็นข้อมูลที่รับเข้ามาจากผู้ใช้งานทางคีย์บอร์ดหรือเมาส์ หรืออาจจะเป็นสัญญาณควบคุมวินโดว์ (window-management message) ที่ส่งมาจากระบบปฏิบัติการวินโดว์ก็ได้ เราสามารถจัดการกับเมสเสจในวินโดว์โปรซีเจอร์ด้วยการเขียนโปรแกรมจัดการเองหรือส่งเมสเสจให้กับระบบปฏิบัติการวินโดว์เพื่อให้ประมวลผลแบบดีฟอลต์ (default processing) ได้ด้วยการเรียกฟังก์ชัน DefWindowProc ดังตัวอย่าง มีกำหนดวิธีประมวลผลเมื่อได้รับเมสเสจ WM_CREATE, WM_ACTIVE, และ WM_DESTROY โดยการใช้คำสั่ง switch ดังนี้


```

switch (uMsg)
{
    case WM_CREATE:
        // TO DO: handle create window message
        ShowWindow(hWnd,SW_NORMAL);
    break;

    case WM_ACTIVATE:
        // TO DO: handle activate window message
        dwActive = LOWORD(wParam);
        switch (dwActive)
        {
            case WA_ACTIVE:
                ShowWindow(hWnd,SW_RESTORE);
                bActive = TRUE;
                break;
            case WA_CLICKACTIVE:
                ShowWindow(hWnd,SW_RESTORE);
                bActive = TRUE;
                break;
            case WA_INACTIVE:
                ShowWindow(hWnd,SW_MINIMIZE);
                bActive = FALSE;
                break;
        }
    break;

    case WM_DESTROY:
        // TO DO: handle destroy window message
        PostQuitMessage(0);
    break;

    default:
        return DefWindowProc(hWnd, uMsg, wParam, lParam);
}

```

4.1.4 วินโดว์เมสเสจ (Message) ที่มีการใช้งานบ่อยๆ

เมสเสจที่ส่งมาจากคีย์บอร์ด (Keyboard Message)

WM_ACTIVATE	วินโดว์ถูกกระตุ้น
WM_KEYDOWN	กดแป้นคีย์บอร์ด
WM_KEYUP	ปล่อยแป้นคีย์บอร์ด
WM_CHAR	พิมพ์ตัวอักษร

เมสเสจที่ส่งมาจากระบบปฏิบัติการวินโดว์ (Windows Message)

WM_CREATE	สร้างวินโดว์
WM_COMMAND	คอมมานด์จากคอนโทรลต่างๆ เช่น เมนู เป็นต้น
WM_QUIT	ออกจากโปรแกรม
WM_CLOSE	ปิดโปรแกรม
WM_DESTROY	ทำลายวินโดว์
WM_SETTEXT	กำหนดตัวอักษรบนไทเทิลของวินโดว์
WM_MOVE	เคลื่อนย้ายวินโดว์
WM_SIZE	ย่อขยายขนาดของวินโดว์
WM_TIMER	ไทเมอร์ส่งค่ามาตามที่กำหนดไว้

เมสเสจที่ส่งมาจากเมาส์ (Mouse Message)

WM_MOUSEMOVE	เคลื่อนย้ายเมาส์เคอร์เซอร์
WM_MOUSEWHEEL	หมุนล้อที่อยู่บนเมาส์
WM_LBUTTONDOWN	กดเมาส์ปุ่มซ้าย
WM_LBUTTONUP	ปล่อยเมาส์ปุ่มซ้าย
WM_RBUTTONDOWN	กดเมาส์ปุ่มขวา
WM_RBUTTONUP	ปล่อยเมาส์ปุ่มขวา

4.1.5 การเลือกใช้ไทม์เมอร์ (Timer)

การใช้งานไทม์เมอร์เมสเสจ WM_TIMER มีประสิทธิภาพไม่เพียงพอที่จะนำมาใช้ในโปรแกรมเกม เนื่องจากมีโอเวอร์เฮด (overhead) ที่เกิดจากการทำงานในลักษณะของเมสเสจ นอกจากนั้น WM_TIMER ยังถูกกำหนดให้มีระดับความสำคัญ (priority) ต่ำกว่าวินโดว์เมสเสจอื่นๆ จำนวนมาก

นอกจาก การวัดค่าเวลาจากเมสเสจไทมเมอร์แล้ว เราสามารถเรียกฟังก์ชันเพื่ออ่านค่าเวลาในระหว่างแต่ละเมสเสจลูปได้ ซึ่งจะเป็นวิธีการอ่านค่าโดยตรงจากไทมเมอร์จึงไม่มีโอเวอร์เฮดเกิดขึ้น ไทมเมอร์ที่ใช้งานได้มี 2 ชนิด ดังนี้

1. ไทมเมอร์ความละเอียดสูง (high-resolution timer) สามารถอ่านค่าเวลาได้ด้วยการเรียกฟังก์ชัน `timeGetTime` ซึ่งไทมเมอร์ตัวนี้จะให้ความละเอียดในระดับ 1 มิลลิวินาที (1/1000 ของวินาที) ค่าที่ส่งคืนมาจะอยู่ในรูป `DWORD` ซึ่งไม่สามารถคำนวณหาช่วงเวลาจากค่านี้ได้ แต่สามารถนำไปใช้หาค่าเวลาที่เปลี่ยนแปลงไปได้โดยเก็บค่าเวลาเดิมไว้แล้วนำมาลบจากค่าที่อ่านได้
2. เคาท์เตอร์ประสิทธิภาพสูง (performance counter) ในกรณีที่ต้องการค่าละเอียดมากยิ่งขึ้น เคาท์เตอร์ประสิทธิภาพสูงซึ่งมีอยู่ในซีพียูรุ่นใหม่ๆ (ไม่ได้มีอยู่ในซีพียูทุกรุ่น) มีความละเอียดในระดับ 1 ไมโครวินาที (1/1000000 ของวินาที) ซึ่งโดยปกติแล้วเคาท์เตอร์ในซีพียูจะทำงานที่ความถี่ 3.16 MHz แต่เนื่องจากไม่ได้มีอยู่ในซีพียูทุกรุ่นจึงต้องทำการตรวจสอบก่อนว่ามีอยู่หรือไม่ โดยใช้ฟังก์ชัน `QueryPerformanceFrequency` ซึ่งโดยปกติจะคืนค่าความถี่ของเคาท์เตอร์แต่ในกรณีที่ไม่มีเคาท์เตอร์จะส่งคืนค่า 0 และใช้ฟังก์ชัน `QueryPerformanceCount` ในการอ่านค่าจากเคาท์เตอร์

4.2 Standard Template Library (STL)

Standard Template Library คือ ชุดของคลาสและฟังก์ชัน (function) ที่ช่วยในการเก็บข้อมูล (collective of data) เช่น dynamic arrays (vector) และช่วยในการทำอัลกอริทึมง่าย เช่น การเรียงลำดับ (sorting) ซึ่งมีการกำหนดเป็นมาตรฐานโดย ANSI/ISO C++ Standards Committee ในปี 1994

STL ออกแบบมาโดยคำนึงถึงมาเร็วในการทำงานเป็นอันดับแรกจึงไม่ตรงกึ่งวลเลยว่า การนำเอา STL มาใช้จะทำให้การทำงานของโปรแกรมที่เขียนขึ้นช้าลง

STL มีจำนวนมากและมีบางส่วนที่มีความซับซ้อน การใช้ STL ให้ได้อย่างมีประสิทธิภาพจึงต้องเข้าใจองค์ประกอบพื้นฐานและวิธีการใช้งานก่อนองค์ประกอบเหล่านี้ก่อน

4.2.1 องค์ประกอบพื้นฐานของ STL (Basic components of STL)

คอนเทนเนอร์ (Containers)

คอนเทนเนอร์ คือ STL template class ซึ่งจัดการลำดับในการเก็บข้อมูล ข้อมูลที่เก็บจะเป็นข้อมูลชนิดใดๆ ก็ได้ที่มี default constructor, destructor และ assignment operator ซึ่งแน่นอนว่าจะต้องมีความแตกต่างกันในการใช้งานคอนเทนเนอร์ในแต่ละชนิด

ไอเทอเรเตอร์ (Iterators)

ไอเทอเรเตอร์เปรียบเสมือนพอยเตอร์ที่ชี้ยังข้อมูลที่อยู่ในคอนเทนเนอร์ นอกจากนั้นแล้วใน STL ยังใช้สัญลักษณ์เดียวกับที่ใช้ในพอยเตอร์อีกด้วย เช่น ใน operator ++ ในการย้ายไอเทอเรเตอร์ไปยังข้อมูลตัวต่อไปที่อยู่คอนเทนเนอร์การย้ายตำแหน่งที่ชี้ข้อมูลของไอเทอเรเตอร์ไปยังข้อมูลตัวถัดไปในคอนเทนเนอร์

เรียกว่า increment นอกจากนั้น การเข้าถึงข้อมูลก็ใช้สัญลักษณ์ * เช่นเดียวกับพอยเตอร์ซึ่งเรียกการเข้าถึงข้อมูลนี้ว่า dereference

อัลกอริทึม (Algorithms)

อัลกอริทึม ที่ทำงานกับ STL ไม่ได้อยู่ในรูปของฟังก์ชันสมาชิกของคลาสคอนเทนเนอร์ แต่อยู่ในรูปของ stand-alone function ซึ่งทำงานอยู่บนไอเทอเรเตอร์ การแยกส่วนที่ทำหน้าที่ในการเก็บข้อมูลออกจากอัลกอริทึม ทำให้จำนวนของอัลกอริทึมที่ต้องเขียนขึ้นเฉพาะสำหรับแต่ละคอนเทนเนอร์ลงได้ เนื่องจากคอนเทนเนอร์ใดที่มีไอเทอเรเตอร์คล้ายคลึงกันก็สามารถใช้อัลกอริทึมร่วมกันได้

4.2.2 แนวความคิดของ STL (STL concepts)

หลักการต่อไปนี้เป็นสิ่งสำคัญในการใช้งาน STL สิ่งแรกก็คือ ต้องรู้ฟังก์ชันที่ให้หาช่วงของข้อมูล (จุดที่เริ่มเก็บข้อมูลถึงจุดสุดท้ายที่มีการเก็บข้อมูล) เมื่อใช้งานคอนเทนเนอร์การหาช่วงของข้อมูลมี 2 เมทอด (Method) ที่สำคัญ คือ begin() และ end() ด้วยการให้ 2 เมทอดนี้สามารถหาช่วงของข้อมูลทั้งหมดที่มีอยู่ใน คอนเทนเนอร์ได้ โดย begin() จะส่งค่าตำแหน่งของข้อมูลตัวแรกในคอนเทนเนอร์และ end() จะส่งค่าตำแหน่งสุดท้ายของข้อมูลที่สามารถใช้งาน ข้อดีของการจัดการช่วงของข้อมูลในลักษณะนี้ คือ ไม่ต้องเขียนโค้ดมาจัดการกับช่วงของข้อมูลที่ไม่ได้ใช้งาน และสามารถนำไปใช้กับไอเทอเรเตอร์ที่ทำงานกับทั้งคอนเทนเนอร์ได้โดยการกำหนดให้ทำงานจนถึงค่าที่ส่งมาจากเมทอด end() ส่วนข้อเสีย ก็คือ การทำงานที่ไม่เป็นไปตามลำดับการทำงานของไอเทอเรเตอร์ต้องใช้ตัวแปรหรือไอเทอเรเตอร์พิเศษเข้าช่วย

นอกจากนั้นยังมีสิ่งที่จะต้องคำนึงถึงอีกอย่างในการใช้งาน STL ก็คือ STL ออกแบบมาให้ส่งข้อมูลแบบ by value ไม่ใช่ by reference ดังนั้น ถ้าใช้งาน STL กับข้อมูลขนาดเล็กก็จะไม่มีปัญหาแต่ถ้าจะนำไปใช้กับข้อมูลขนาดใหญ่ก็ควรใช้กับพอยเตอร์ของคลาสหรือ โครงสร้างข้อมูล (struct) จะดีกว่า

4.2.3 คอนเทนเนอร์พื้นฐานใน STL ที่ใช้งานบ่อย

เวกเตอร์ (Vectors)

เวกเตอร์ใน STL หมายถึง อาร์เรย์ที่สามารถเปลี่ยนแปลงขนาดได้ (resizable array) ซึ่งแตกต่างจาก C++ อาร์เรย์ที่ไม่สามารถเปลี่ยนแปลงขนาดได้ อย่างไรก็ตาม มีความจำเป็นอย่างยิ่งที่จะต้องทำความเข้าใจวิธีการในการเปลี่ยนแปลงขนาดของเวกเตอร์ด้วย

เวกเตอร์สร้างขึ้นโดยใช้อาร์เรย์ (array) ดังนั้นในการเปลี่ยนแปลงขนาดจึงมีความจำเป็นที่จะต้องจองหน่วยความจำใหม่ (reallocate memory) แล้วจึงย้ายข้อมูลไปยังอาร์เรย์ใหม่ที่สร้างขึ้น หลังจากนั้นจึงปล่อยหน่วยความจำเดิม แต่เราก็สามารถจองพื้นที่หน่วยความจำให้มีขนาดใหญ่กว่าที่ต้องการใช้งานได้ เพื่อที่จะเหลือพื้นที่สำหรับในกรณีที่ต้องการขยายขนาดซึ่งจะทำให้การทำงานเร็วขึ้นเนื่องจากไม่ต้องจองหน่วยความจำและย้ายข้อมูลบ่อยๆ

ลิสต์ (Lists)

ลิสต์ใน STL สร้างขึ้นในแบบ doubly linked list ดังนั้น การแทรกหรือลบข้อมูลที่ใดๆ ก็ตามจึงสามารถทำได้ในเวลาอันรวดเร็ว แต่จะสูญเสียความสามารถในการเข้าถึงข้อมูลแบบสุ่ม (random access) ที่มีในเวกเตอร์และเดค (deque)

เดค (Deque)

เดค หรือ double-ended queue ได้รับการออกแบบมาสำหรับใช้ในกรณีที่ต้องการความสามารถในการแทรกหรือดึงข้อมูลออกจากปลายทั้งสองด้านของคอนเทนเนอร์ (ไม่มีความต้องการที่จะแทรกหรือดึงข้อมูลที่อยู่ตรงกลางออกแต่ก็สามารถทำได้) เช่นเดียวกับเวกเตอร์สามารถเข้าถึงข้อมูลในเดคแบบสุ่มได้ แต่ประสิทธิภาพจะต่ำกว่าการเข้าถึงข้อมูลในเวกเตอร์เนื่องจากความซับซ้อนของข้อมูลภายในของเดค ซึ่งเกิดจากการเรียงลำดับการเชื่อมต่อของ memory block

แมพ (Maps)

เป็น value-pairing container หมายความว่าข้อมูล 2 ชนิดที่จะต้องใช้คู่กันเสมอ โดยมีโครงสร้างเป็น key/value แล้วจึงแทรกเข้าไปในคอนเทนเนอร์สามารถหาค่าในคอนเทนเนอร์คอนเทนเนอร์ได้โดยหาค่าของ key ซึ่งจะหาได้ในเวลา $O(\log n)$ ถึงแม้ว่าจะมีประสิทธิภาพไม่ดีเท่ากับการใช้ hash table แต่มีข้อดี เนื่องจากมีความสามารถในการเรียงลำดับข้อมูลระหว่างการแทรกข้อมูลอันเป็นผลมาจากวิธีการที่ใช้ในการเก็บข้อมูล (balanced binary tree หรือ red-black tree)

สแต็ก (Stacks)

สแต็กมีฟังก์ชันที่สำคัญ 3 ฟังก์ชัน คือ push(), pop() และ top() สำหรับใช้ในการเพิ่มและลบข้อมูลออกจากคอนเทนเนอร์โดยค่าเริ่มต้นแล้วสแต็กสร้างขึ้นโดยใช้เดคแต่เราสามารถเปลี่ยนชนิดของคอนเทนเนอร์ที่ใช้สร้างสแต็กได้โดยการกำหนดค่าคอนสตรัคเตอร์ (constructor) ดังนี้

```
// Implements a stack with deque as the underlying container type
```

```
stack<int> c;
```

```
// Implements a stack with a vector as the underlying container type
```

```
stack< int,vector<int> > c;
```

การใช้เวกเตอร์แทนการใช้เดคสามารถทำได้เนื่องจากสแต็กสามารถสร้างขึ้นจากคอนเทนเนอร์ใดๆ ก็ตามที่มีฟังก์ชัน push_back(), pop_back() และ back() เนื่องจากฟังก์ชัน push(), pop() และ top() ของสแต็กจะถูกจับเข้าคู่ (map) กับฟังก์ชันเหล่านี้ และข้อสำคัญอย่างลึกลับที่ช่องว่างระหว่างเครื่องหมาย > มิเช่นนั้นคอมไพเลอร์จะมองเป็น single stream operator >>

สิ่งสำคัญในการใช้ stack คือ อย่างลึกลับว่า STL คอนเทนเนอร์ส่วนใหญ่สร้างขึ้นโดยคำนึงถึงความเร็วเป็นอันดับแรก ดังนั้น ก่อนจะใช้คำสั่ง pop() และ top() ต้องแน่ใจว่ามีข้อมูลอยู่ในสแต็กโดยใช้ฟังก์ชัน size() หรือ empty() ในการตรวจสอบว่าสแต็กมีข้อมูลอยู่หรือไม่

คิว (Queues)

คิวทำงานคล้ายกับสแต็กยกเว้นข้อมูลจะถูกใส่จากด้านหลังและดึงออกได้จากด้านหน้า มีฟังก์ชันที่สำคัญดังนี้ `push()`, `pop()`, `front()` และ `back()` โดย `back()` ใช้อ้างอิงถึงข้อมูลที่เพิ่งเพิ่มเข้าไปในคิวและ `front()` ใช้อ้างอิงถึงข้อมูลที่อยู่ด้านหน้าสุดของคิวซึ่งจะถูกดึงออก

เช่นเดียวกับสแต็กนั้นก็ต้องใช้ฟังก์ชัน `size()` และ `empty()` ในการควบคุมขนาดของคิวนอกจากนั้นยังสามารถกำหนดชนิดของคอนเทนเนอร์ที่ใช้สร้างได้เช่นเดียวกับสแต็กด้วยแต่การใช้เวกเตอร์ในการสร้างคิวนั้นให้ประสิทธิภาพไม่ดีเนื่องจากการแทรกข้อมูลจากด้านหน้าของเวกเตอร์ไม่มีประสิทธิภาพนัก

4.2.4 การใช้งาน STL

Standard C++ library ทุกตัวถูกประกาศ (declare) หรือนิยาม (define) ไว้ใน standard header การใช้งานไลบรารี (library) เหล่านี้จึงต้อง include standard header ของ standard library ที่ต้องการใช้งานด้วย ซึ่ง STL ที่ใช้งานในโครงการนี้จะอยู่ใน standard header ดังนี้ `<algorithm>`, `<list>`, `<map>`, `<queue>`, `<stack>`, `<string>` และ `<vector>`

ตัวอย่างการ include standard header

```
// include vector facilities
```

```
#include <vector>
```

จะเห็นว่าการ include standard header จะแตกต่างจากการ include ไฟล์เฮดเดอร์ทั่วไปเนื่องจากไม่มี .h นอกจากนั้นการใช้งาน STL ควรจะประกาศ namespace ด้วย เนื่องจากชื่อทุกชื่อใน standard C++ header นิยามโดยใช้ std namespace ซึ่งถ้าไม่ประกาศ namespace แล้วการใช้งาน STL จะต้องเขียนโค้ดในรูปของ `std::vector` แต่ถ้าประกาศแล้วเขียนเพียงเวกเตอร์ก็สามารถใช้งานได้แล้ว ซึ่งการประกาศใช้ namespace ทำได้ดังนี้

```
using namespace std;
```


บทที่ 5

DirectX

5.1 แนะนำไดเร็กเอ็กซ์

ไมโครซอฟต์ไดเร็กเอ็กซ์ (Microsoft® DirectX®) คือ ชุดของแอปพลิเคชันโปรแกรมมิ่งอินเทอร์เฟซ (Application Programming Interface : API) ในระดับล่าง ที่ได้รับการออกแบบมาสำหรับการสร้างเกมและแอปพลิเคชันทางด้านมัลติมีเดียประสิทธิภาพสูงอื่นๆ สนับสนุนการทำงานกราฟิกแบบ 2 มิติ (2D), แบบ 3 มิติ (3D), เสียงเอฟเฟ็กต์ (sound effect) และเพลง (music), อุปกรณ์รับข้อมูล และสนับสนุนการใช้งานแอปพลิเคชันบนเน็ตเวิร์ค เช่น เกมประเภทผู้เล่นหลายคน (multiplayer game)

เนื่องจากระบบปฏิบัติการวินโดวส์มีการใช้งานมากที่สุดในเครื่องคอมพิวเตอร์ที่ใช้งานในครัวเรือน และเป็นตลาดของผลิตภัณฑ์เพื่อความบันเทิงที่มีขนาดใหญ่ แต่กับเกิดข้อจำกัดขึ้นเนื่องจากการพัฒนาอุปกรณ์ที่มีความสามารถแตกต่างกันแม้จะเป็นอุปกรณ์ที่ทำหน้าที่อย่างเดียวกัน นั่นคือไม่มีมาตรฐานในการพัฒนาอุปกรณ์ เพื่อแก้ปัญหานี้ไดเร็กเอ็กซ์จึงได้รับการออกแบบมาเพื่อให้เป็นมาตรฐานสำหรับการติดต่อกับอุปกรณ์ด้วยซอฟต์แวร์ (standard software interface) ทำให้ผู้พัฒนาซอฟต์แวร์สามารถทำงานได้ง่ายขึ้น เนื่องจากไม่ต้องทดสอบกับอุปกรณ์ทุกชนิดที่มีอยู่ในท้องตลาด เพียงแค่ทดสอบให้สามารถทำงานได้กับไดเร็กเอ็กซ์ก็เพียงพอแล้ว ส่วนผู้ผลิตอุปกรณ์ฮาร์ดแวร์มีหน้าที่จะต้องสร้างไดรเวอร์ของอุปกรณ์ (device driver) สำหรับการทำงานกับไดเร็กเอ็กซ์ได้

5.1.1 ชุดพัฒนาซอฟต์แวร์สำหรับไดเร็กเอ็กซ์ (DirectX Software Development Kit : DirectX SDK)

DirectX SDK คือ ชุดของไฟล์เฮดเดอร์, ไลบรารี, เอกสาร และตัวอย่างซอร์สโค้ด (source code) ต่างๆ ที่ช่วยในการพัฒนาซอฟต์แวร์ด้วยการใช้ไดเร็กเอ็กซ์ ในปัจจุบันได้รับการพัฒนามาถึงเวอร์ชัน 8 แล้ว ซึ่งในเวอร์ชันนี้ประกอบด้วยอินเทอร์เฟซ (interface) ต่างๆ ที่ช่วยในการทำงานเกี่ยวกับมัลติมีเดีย ดังนี้

DirectX Graphics

ได้รับการออกแบบมาเพื่อทำงานด้านกราฟิกทั้ง 2 มิติและ 3 มิติ อินเทอร์เฟซในกลุ่มนี้ช่วยจัดการการใช้อุปกรณ์แสดงผล 3 มิติที่มีอยู่ในเครื่องคอมพิวเตอร์ (3-D video-display hardware) ในรูปแบบที่ไม่ขึ้นกับชนิดของอุปกรณ์ที่มีอยู่ (device-independent)

DirectX Audio

ได้รับการออกแบบมาเพื่อทำงานด้านเสียง ซึ่งนอกจากการเล่นไฟล์เสียงแล้ว ยังสนับสนุนการใช้งานฮาร์ดแวร์เร่งความเร็ว (hardware acceleration) ในการทำสร้างเสียงแบบไดนามิก (dynamic soundtrack) และเอฟเฟ็กต์เสียง 3 มิติขั้นสูง (advanced 3-D position effect)

DirectInput

ได้รับการออกแบบมาเพื่อทำงานกับอุปกรณ์อินพุตเพื่อให้เข้าถึงข้อมูลได้ด้วยการติดต่อโดยตรงกับฮาร์ดแวร์ด้วยฮาร์ดแวร์ไดรเวอร์ (hardware driver) ซึ่งเร็วกว่าการติดต่อด้วยเมสเสจในวินโดวส์ นอกจากนี้ยังสนับสนุนอุปกรณ์แบบแรงตอบสนอง (force feedback) อีกด้วย

DirectPlay

ได้รับการออกแบบมาเพื่อการทำงานด้านเน็ตเวิร์ก ในการพัฒนาโปรแกรมแบบมัลติเพลเยอร์ (multiplayer) ทั้งในแบบเพียร์ทูเพียร์ (Peer-to-Peer) และแบบไคลเอนต์/เซิร์ฟเวอร์ Client/Server

DirectShow

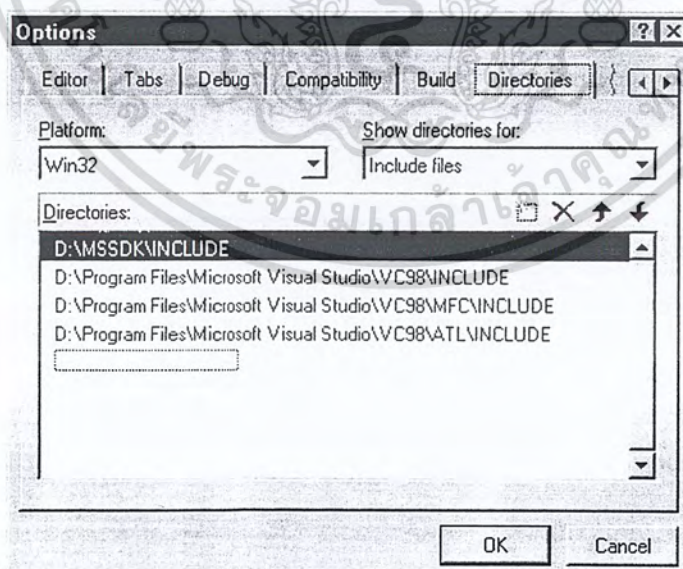
ออกแบบมาสำหรับการทำงานเกี่ยวกับมีเดียสตรีมมิ่ง (media-streaming) เช่น การเล่นไฟล์วีดีโอคุณภาพสูง (high-quality video)

DirectSetup

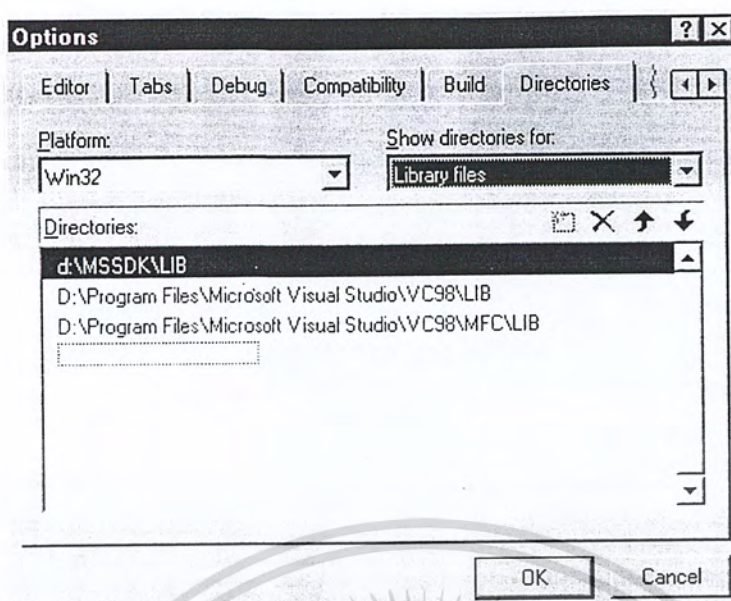
เป็น API ที่จัดการการติดตั้ง (install) องค์ประกอบที่จำเป็นในการใช้งานไคลเอนต์บนเครื่องคอมพิวเตอร์ด้วยการเรียกใช้เพื่อคำสั่งเดียว เนื่องจากไคลเอนต์เป็นผลิตภัณฑ์ที่มีความซับซ้อนสูงรวมทั้งในขั้นตอนการติดตั้งด้วยจึงไม่ควรที่จะติดตั้งไคลเอนต์ด้วยตัวเอง

5.2 การเตรียม Microsoft Visual C++ ให้พร้อมใช้งานกับ DirectX

การใช้งาน DirectX SDK ร่วมกับคอมไพเลอร์ Microsoft Visual C++ นั้นจะต้องกำหนดไคลเอนท์ทอรีที่เก็บไฟล์เฮดเดอร์ (header file) และไฟล์ไลบรารีก่อน โดยเลือก Option จากเมนู Tool แล้วเลือกแท็บ Directories ซึ่งจะปรากฏไคโอะลอกบ็อกแล้วกำหนดค่าดังรูป



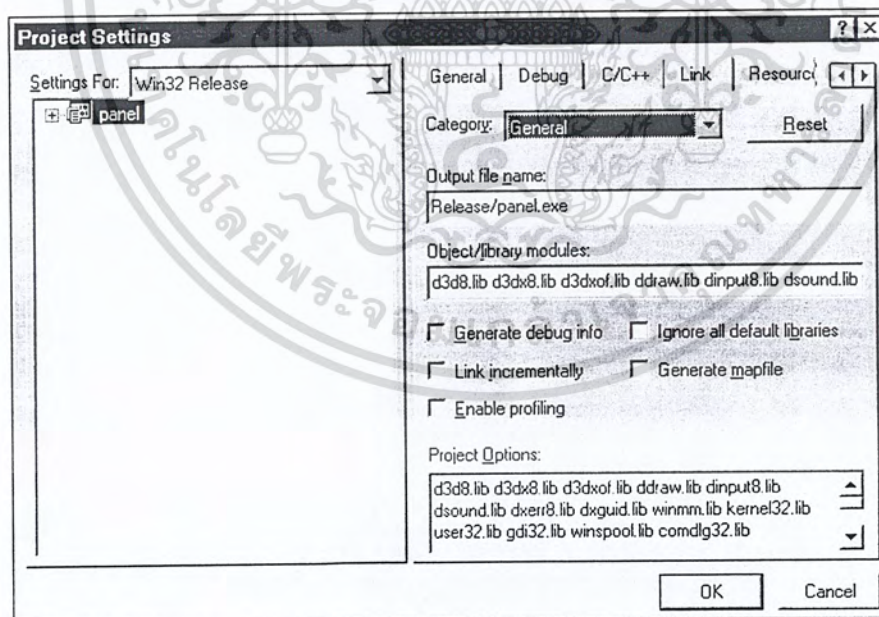
รูปที่ 5-1 ไคโอะลอกบ็อกสำหรับการกำหนดไคลเอนท์ทอรีของเฮดเดอร์ไฟล์ในไคลเอนท์



รูปที่ 5-2 ไดอะล็อกบ็อกสำหรับการกำหนดไคลเรททอรีของไลบรารีไฟล์ในไคลเรกเอ็กซ์

และมีความจำเป็นที่จะต้องเลื่อนตำแหน่งไคลเรททอรีไปอยู่ด้านบนสุดเนื่องจากไลบรารีที่มากับตัวคอมไพเลอร์อาจจะเก่ากว่าเวอร์ชันที่ต้องการใช้งาน

นอกจากนี้ยังต้องกำหนดไลบรารีที่จะให้คอมไพเลอร์ลิงก์อีกด้วยโดยเลือก Setting จากเมนู Project จะปรากฏไดอะล็อกบ็อกแล้วจึงกำหนดค่าไลบรารีที่จะใช้เพิ่มเข้าไป ดังรูป



รูปที่ 5-3 ไดอะล็อกบ็อกสำหรับการกำหนดไลบรารีที่จะนำมาลิงก์ในขณะคอมไพล์

บทที่ 6

ไคเร็กเอ็กกราฟิก (DirectX Graphics)

6.1 แนะนำไคเรทรีดี (Introduction to Direct3D)

ไคเรทรีดี คือ แอปพลิเคชันโปรแกรมมิ่งอินเทอร์เฟซระดับล่าง สำหรับนักพัฒนาโปรแกรมที่ต้องการสร้างเกมหรือโปรแกรมมัลติมีเดียประสิทธิภาพสูงบนระบบปฏิบัติการวินโดวส์ ซึ่งการใช้ไคเรทรีดีทำให้โปรแกรมสามารถติดต่อกับฮาร์ดแวร์เร่งความเร็วได้ในระดับล่าง (โดยตรง)

ไคเรทรีดี คือ ซอฟต์แวร์อินเทอร์เฟซ (software interface) ซึ่งจัดการการเข้าซึ่งอุปกรณ์แสดงผล (display device) โดยยังคงรักษาความเข้ากันได้กับกราฟิกดีไวส์อินเทอร์เฟซ (Graphics Device Interface : GDI) และจัดการให้มีการใช้งานในแบบที่ไม่ขึ้นกับชนิดของอุปกรณ์ (device independent) และมีความสามารถในการใช้คุณสมบัติพิเศษที่อยู่ในฮาร์ดแวร์ได้

6.2 คุณสมบัติของ ไคเรทรีดี

- สนับสนุนการใช้งาน depth buffer (z-buffer หรือ w-buffer)
- สามารถเรนเดอร์ภาพได้ทั้งในแบบ Flat shading และ Gouraud shading
- สามารถใช้แหล่งแสงได้หลายแหล่งและหลายชนิด
- สนับสนุนการใช้แมตทีเรียล (material) และเทกเจอร์ (texture) รวมทั้ง การทำมิพแมพ (mipmap)
- มีซอฟต์แวร์อีมูเลชันไดเวอร์ (software emulation driver) ที่มีประสิทธิภาพสูง
- ไม่ขึ้นอยู่กับชนิดของฮาร์ดแวร์
- สนับสนุนคำสั่งพิเศษที่มีอยู่ในซีพียู เช่น MMX, SSE และ 3DNow!
- สนับสนุน Hardware Abstraction Layer (HAL)
- สนับสนุนการทำ page flipping ด้วย back buffer จำนวนมากในโปรแกรมที่แสดงผลแบบเต็มจอภาพ (fullscreen)
- สนับสนุนการตัด (clipping) ในโปรแกรมที่ทำงานในวินโดวส์และแบบเต็มจอภาพ
- สามารถใช้งาน image-stretching hardware ได้
- สามารถใช้งานเมมโมรีของอุปกรณ์แสดงผลทั้งส่วนที่เป็นมาตรฐานและส่วนที่ขยายเพิ่มเติมได้
- คุณสมบัติอื่นๆ เช่น การใช้งานฮาร์ดแวร์แต่เพียงผู้เดียว (exclusive hardware access) และการเปลี่ยนความละเอียดในการแสดงผล (resolution switching)

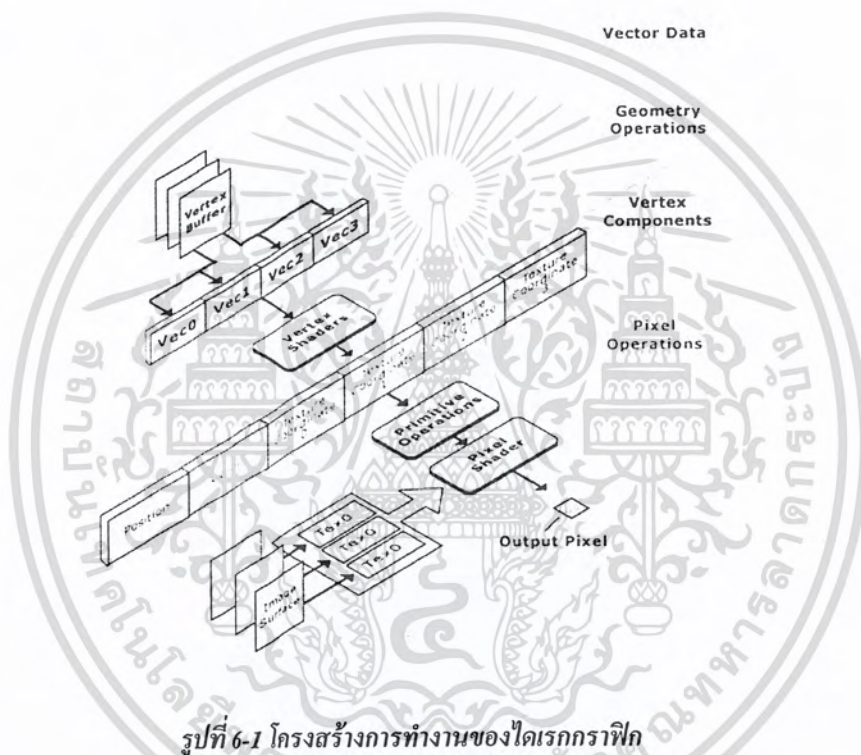
ด้วยคุณสมบัติเหล่านี้การสร้างโปรแกรมด้วยไคเรทรีดีจึงมีประสิทธิภาพสูงกว่าการใช้ GDI ของวินโดวส์และการเขียนโปรแกรมบนระบบปฏิบัติการคอส

การจัดการสิ่งต่างๆ ในไคเรทรีดีมีพื้นฐานอยู่บนทฤษฎีของเวอร์เท็กซ์, โพลีกอนและคำสั่งต่างๆ ที่ควบคุมข้อมูลเหล่านี้ ซึ่งสามารถเข้าถึงไปป์ไลน์ของกราฟิก 3 มิติในส่วนของ การเปลี่ยนแปลงตำแหน่ง (transformation), การให้แสง (lighting) และ การลงจุดเพื่อสร้างภาพ (rasterization) ได้ทันที ซึ่งถ้าฮาร์ดแวร์ที่มีอยู่ในเครื่องไม่สนับสนุนการเร่งความเร็วในส่วนใดของไปป์ไลน์ก็ตาม ไคเรทรีดีจะใช้ซอฟต์แวร์ที่

ประสิทธิภาพสูงในการทำงานในส่วนนั้นแทน (แต่การทำงานของซอฟต์แวร์ก็ยังมีประสิทธิภาพต่ำกว่าการใช้ฮาร์ดแวร์อยู่ดี)

6.3 สถาปัตยกรรมของไดเรกทรีดี (Direct3D Architecture)

ในไดเรกทรีดีมี 2 ส่วนที่สามารถโปรแกรมได้คือ เวอร์เท็กซ์เชดเดอร์ (vertex shader) และ พิกเซลเชดเดอร์ (pixel shader) ซึ่งเวอร์เท็กซ์เชดเดอร์เป็นส่วนที่จัดการการสร้างและกระบวนการต่างๆ บนเวอร์เท็กซ์ ส่วนพิกเซลเชดเดอร์ทำงานหลังจากมีการประมวลผลข้อมูล ออกมาเป็นตำแหน่งของพิกเซลและค่าสีของพิกเซลนั้นๆ แล้ว ดังภาพ



รูปที่ 6-1 โครงสร้างการทำงานของไดเรกทรีดี

จะเห็นว่าการทำงานของเวอร์เท็กซ์เชดเดอร์จะรับข้อมูลจากบัฟเฟอร์ ซึ่งโดยปกติแล้วจะเป็นเวอร์เท็กซ์บัฟเฟอร์ (vertex buffer) ที่เก็บข้อมูลเวอร์เท็กซ์ และนำมาผลกระบวนการต่างๆ ซึ่งสามารถกำหนดกระบวนการที่จะทำได้โดยใช้ DirectX vertex shader assembler

หลังจากการข้อมูลผ่านการประมวลผลในเวอร์เท็กซ์เชดเดอร์แล้วจะต้องเรียกฟังก์ชันในการวาดข้อมูลเหล่านี้ เช่น DrawPrimitive เป็นต้น ซึ่งเมื่อเรียกฟังก์ชันเหล่านี้แล้วจุดแต่ละจุดที่ได้จากฟังก์ชันก็จะส่งเข้าสู่พิกเซลเชดเดอร์ ข้อมูลที่ส่งให้กับพิกเซลเชดเดอร์ประกอบด้วยตำแหน่งบนจอภาพ, สี, เทกเจอร์และ ค่าอันดับแสดงตำแหน่งของเทกเจอร์ พิกเซลเชดเดอร์จะประมวลผลตามคำสั่งที่กำหนด โดยใช้ DirectX pixel assembler

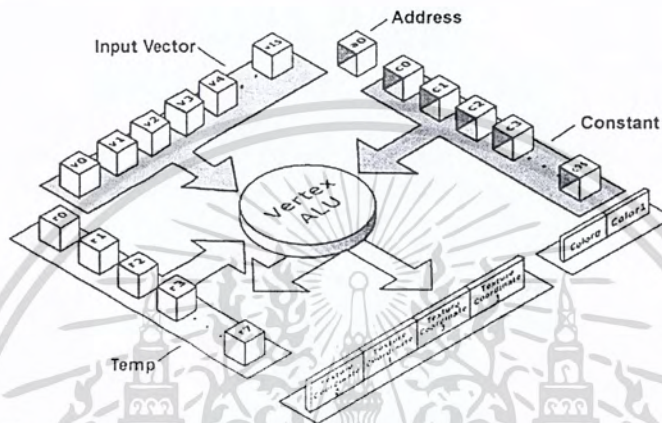
นอกจาก ข้อมูลที่ได้รับจากฟังก์ชันในการวาดข้อมูล เช่น DrawPrimitive แล้ว พิกเซลเชดเดอร์ยังต้องการข้อมูลของเทกเจอร์ในรูปของเทกเจอร์เซอร์เฟส (texture surface) อีกด้วยซึ่งจะให้ร่วมกับข้อมูลอันดับ

แสดงตำแหน่งของเทกเจอร์ ในการกำหนดสีที่จะปรากฏ แล้วจึงส่งค่าพิกเซล (pixel) ไปยังเฟรมบัฟเฟอร์ (frame buffer) เพื่อแสดงผลออกจอภาพต่อไป

6.3.1 สถาปัตยกรรมของเวอร์เท็กซ์เชดเดอร์ (Vertex Shader Architecture)

เวอร์เท็กซ์เชดเดอร์รับข้อมูลมาจากเวอร์เท็กซ์ดาต้าสตรีม (vertex data stream) และมีข้อมูลค่าคงที่ที่มี

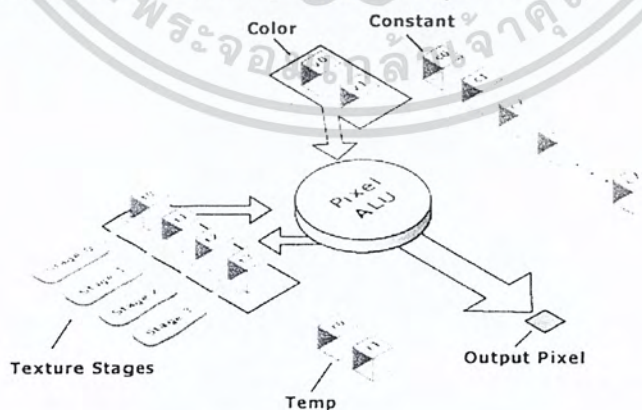
- * ความจำเป็นในการคำนวณ ซึ่งสามารถอ่านเข้ามาได้โดยใช้เวอร์เท็กซ์เชดเดอร์ดีคลอเรเตอร์ (vertex shader declarator) ดังแผนภาพ



รูปที่ 6-2 สถาปัตยกรรมของเวอร์เท็กซ์เชดเดอร์

6.3.2 สถาปัตยกรรมของพิกเซลเชดเดอร์ (Pixel Shader Architecture)

พิกเซลเชดเดอร์จะต้องอ่านข้อมูลที่เป็น clip space vertex (homogeneous coordinate) ค่าของสีดีฟฟิวส์ (diffuse) และสเปกคิวลาร์ (specular) มาจากค่าสีในรีจิสเตอร์สี (color register) v0 และ v1 ค่าคู่อันดับแสดงตำแหน่งของเทกเจอร์และเทกเจอร์ที่ใช้มาจากค่าในรีจิสเตอร์เทกเจอร์ (texture register) t0, t1, t2 และ t3 จากข้อมูลในขั้นตอนการกำหนดเทกเจอร์ (texture setup stage) และข้อมูลในเวอร์เท็กซ์ ดังภาพ



รูปที่ 6-3 สถาปัตยกรรมของพิกเซลเชดเดอร์

6.4 เรนเดอร์ไปป์ไลน์ใน ไคเรกทรีดี (Direct3D Rendering Pipeline)

เรนเดอร์ไปป์ไลน์ (rendering pipeline) คือ ชุดของขั้นตอนในการประมวลผลซึ่งกำหนดการกระทำของไปป์ไลน์ในการเปลี่ยนแปลงตำแหน่งและให้แสงของเวอร์เท็กซ์ (vertex transform and lighting pipeline) และ ไปป์ไลน์ในการเฉลี่ยค่าของพิกเซลและเทกเจอร์ (pixel and texture blending pipeline)

ในไคเรกทรีดีมีเรนเดอร์ไปป์ไลน์ 2 แบบ คือ การประมวลผลฟังก์ชันแบบตายตัว (fix function vertex and pixel processing) และการประมวลผลเวอร์เท็กซ์และพิกเซลแบบโปรแกรมได้ (Programmable Vertex and Pixel Processing) ซึ่งมีรายละเอียด ดังนี้

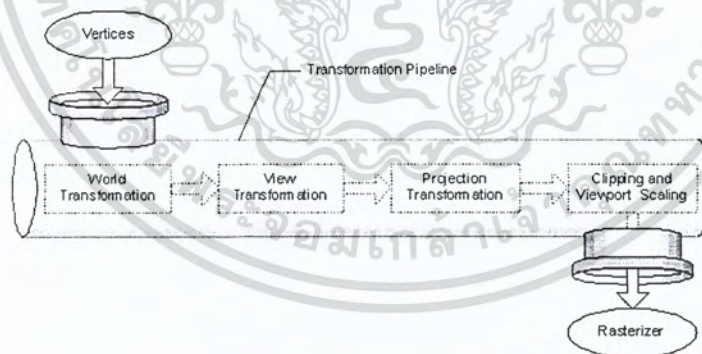
6.4.1 การประมวลผลฟังก์ชันแบบตายตัว (Fix Function Vertex and Pixel Processing)

ในไคเรกทรีดีการประมวลผลฟังก์ชันแบบตายตัวจะใช้ไปป์ไลน์ที่เรียกว่า Transformation Pipeline หรือ Geometry Pipeline ซึ่งมีส่วนประกอบ ดังนี้

Transformation engine

ทำหน้าที่ในการกำหนดตำแหน่งของโมเดลและผู้ดู (viewer) ในโลกของไคเรกทรีดี, ฉายเวอร์เท็กซ์ (project vertex) สำหรับการแสดงผลบนจอภาพ และตัด (clip) เวอร์เท็กซ์ส่วนที่มองไม่เห็นทิ้ง นอกจากนั้นยังทำงานในส่วนของการแสดงด้วยโดยการคำนวณค่าของดีฟิฟัส และสเปกคิวลาร์ในแต่ละเวอร์เท็กซ์

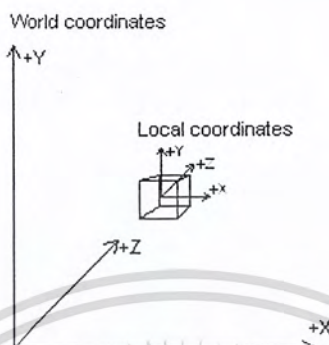
Transformation pipeline รับเวอร์เท็กซ์เข้ามาแล้วประมวลผลโดยใช้ world, view และ projection transformation ซึ่งอยู่ในรูปของเมตริกซ์ หลังจากนั้นข้อมูลเวอร์เท็กซ์จะถูกตัดให้เหลือแต่ส่วนที่มองเห็น แล้วจึงส่งไปยังส่วนของการลงจุดสีต่อไป คังภาพ



รูปที่ 6-4 เรนเดอร์ไปป์ไลน์ในไคเรกทรีดี

World transformation

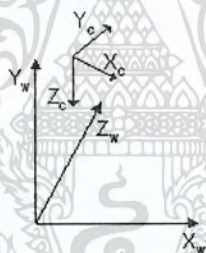
คือ การเปลี่ยนแปลงตำแหน่งของเวอร์เท็กซ์จากเดิมที่มีความสัมพันธ์กับตัวเองเท่านั้น (model space) ให้เปลี่ยนมาอ้างอิงตำแหน่งใหม่ซึ่งใช้อ้างอิงกับวัตถุทั้งหมดที่จะปรากฏในฉาก (world space) ดังภาพ



รูปที่ 6-5 รูปแสดงความสัมพันธ์ระหว่างตำแหน่งของโมเดลใน world coordinates

View transformation

เป็นการกำหนดตำแหน่งของผู้ดู (viewer) หรือกล้อง (camera) ใน world space



รูปที่ 6-6 รูปแสดงความสัมพันธ์ระหว่างตำแหน่งของโมเดลใน view coordinates

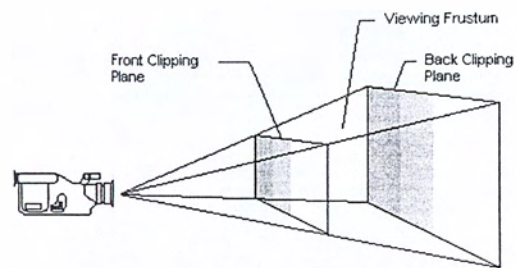
Projection transformation

เป็นการกำหนดค่าภายในของกล้องในทำนองเดียวกับการเลือกเลนส์ที่จะใช้ในกล้องถ่ายภาพ ซึ่งเป็นส่วนที่มีความซับซ้อนมากที่สุดใน การ transformation ทั้งสามแบบ ประกอบด้วยหัวข้อต่อไปนี้

View Frustum

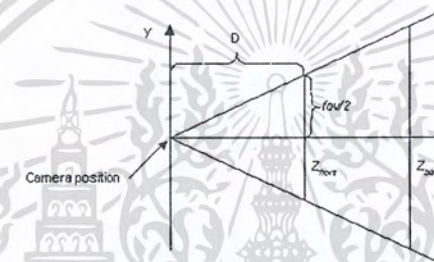
View frustum คือ บริเวณ 3 มิติที่จะปรากฏโดยมีความสัมพันธ์กับตำแหน่งและมุมมองของกล้อง (viewport's camera) รูปร่างของบริเวณ 3 มิติที่เกิดขึ้นจะเป็นตัวกำหนดว่าโมเดลจะโปรเจกต์บนจอภาพในลักษณะใด ซึ่งลักษณะการโปรเจกต์ที่ใช้ทั่วไป คือ แบบเพอร์สเปกทิฟ (perspective) ซึ่งวัตถุที่อยู่ใกล้จะดูมีขนาดใหญ่กว่าวัตถุที่อยู่ห่างออกไป

View frustum มีรูปร่างเหมือนปิระมิดซึ่งมีตำแหน่งของกล้องเป็นจุดยอด บริเวณ 3 มิติจะเกิดขึ้นจากการตัดปิระมิดนี้ด้วย ระนาบตัดด้านหน้า (front clipping plane) และระนาบตัดด้านหลัง (back clipping plane) และวัตถุจะมองเห็นก็ต่อเมื่อมันอยู่ในบริเวณนี้เท่านั้น ดังแผนภาพ



รูปที่ 6-7 รูปแสดง view frustum

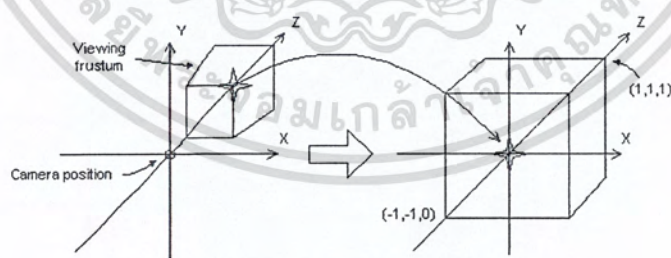
view Frustum มีพารามิเตอร์ที่สำคัญ คือ fov (field of view) และค่าระยะทางของระนาบตัดทั้งด้านหน้าและด้านหลัง ซึ่งกำหนดอยู่ในแกน z ดังแผนภาพ



รูปที่ 6-8 รูปแสดงการคำนวณค่า fov

Projection matrix

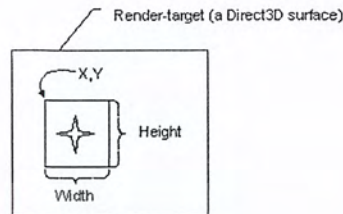
Projection matrix ใช้ในการปรับขนาดของ view frustum เนื่องจากบริเวณที่วัตถุสามารถมองเห็นได้จะต้องถูกปรับให้เป็นรูปสี่เหลี่ยมลูกบาศก์จึงจะสามารถประมวลผลต่อไปได้ ดังภาพ



รูปที่ 6-9 รูปแสดงวิธีการ projection

Viewport and clipping

Viewport คือ สี่เหลี่ยมที่โปรเจกต์ซึ่งฉากใน 3 มิติ ซึ่งใน ไคเรกทรีดี จะอยู่ในรูปของคู่อันดับใน Direct3D Surface ซึ่งระบบใช้เป็นบริเวณที่เรนเดอร์ภาพ ดังภาพ



รูปที่ 6-10 รูปแสดงวิวพอร์ตและวิธีการตัด

การลงจุด (Rasterization)

หลังจากผ่าน transformation pipeline แล้วค่าพิกัดจะถูกแปลง, ตัด และย่อขยายจนมีขนาดพอดีกับวิวพอร์ต (viewport) บนเซอร์เฟสที่ต้องการเรนเดอร์ (render-target) คือ พร้อมที่จะส่งไปยังส่วนของการลงจุดสีเพื่อแสดงผลบนจอภาพ แต่เนื่องจากในขณะนี้ข้อมูลพิกัดยังคงอยู่แบบโฮโมจีเนียส (homogeneous) อยู่ และตัวลงจุดสียังต้องการข้อมูลของ reciprocal-of-homogeneous-w (RHW) เพิ่มเติมอีกด้วย ซึ่ง ไคเรกทรีดี จะทำการแปลงค่าตำแหน่งในแกน x, แกน y และแกน z ไปเป็นข้อมูลที่ไม่โฮโมจีเนียสโดยหารด้วยค่า w และหาค่า RHW จากส่วนกลับของค่า w ดังสมการ

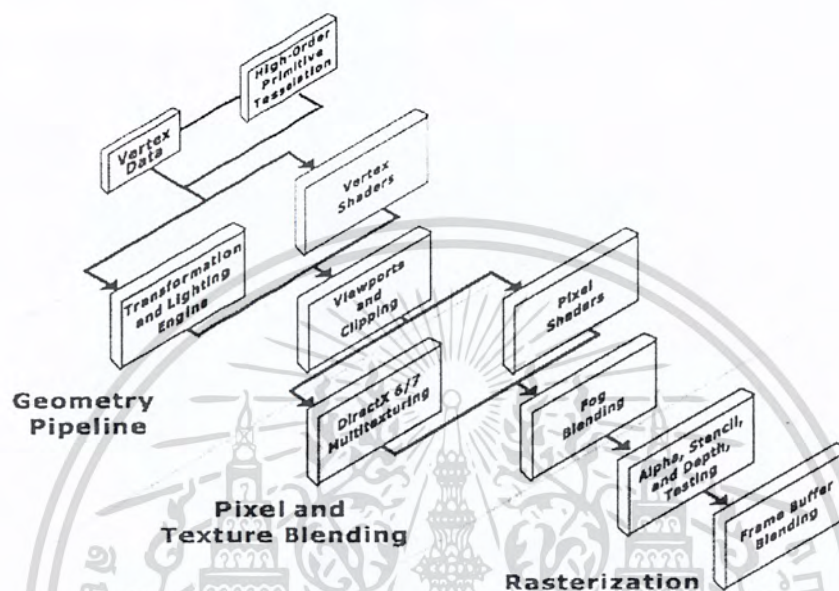
$$\begin{aligned}X_s &= x/w \\Y_s &= y/w \\Z_s &= z/w \\RHW &= 1/w\end{aligned}$$

เมื่อได้ค่าที่ต้องการแล้วจึงส่งเข้าสู่ตัวลงจุดสี (rasterizer) ซึ่งจะใช้ค่าในแกน x และแกน y ในการกำหนดว่าเป็นจุดใดบนจอภาพ, และใช้ข้อมูลในแกน z เพื่อใช้วัดความลึกเพื่อเลือกว่าจุดสีใดที่จะได้แสดงผล โดยใช้ Z-Buffer ส่วนค่า RHW ใช้ในการคำนวณหมอก (fog), การทำปะเทกเจอร์ (texture mapping) และ ใช้ในการวัดความลึกเพื่อเลือกว่าจุดสีใดที่จะได้แสดงผลโดยใช้ W-Buffer

6.4.2 การประมวลผลเวอร์เท็กซ์และพิกเซลแบบโปรแกรมได้

(Programmable Vertex and Pixel Processing)

นอกจากไปป์ไลน์ในการประมวลผลเดิมที่มีอยู่ในเวอร์ชันเก่าๆ แล้วใน ไดรেকทรีดี เวอร์ชัน 8 ยังได้เพิ่มไปป์ไลน์ที่สามารถโปรแกรมได้อีกด้วย โดยมีส่วนที่เพิ่มเข้ามามีดังนี้



รูปที่ 6-11 รูปแสดงขั้นตอนในการประมวลผลเวอร์เท็กซ์และพิกเซล

ส่วนแรกที่เพิ่มเข้ามาคือ high-order primitive module ซึ่งทำหน้าที่ในการเทสเซลเลต (tessellate) high-order primitive เช่น bezier และ B-spline ส่วนเวอร์เท็กซ์เชดเดอร์ (vertex shadder) เป็นโมดูลที่สามารถโปรแกรมได้ใช้ในการประมวลผลทางจีโอเมตริก (geometric) กับข้อมูลเวอร์เท็กซ์ ซึ่งมีความสามารถในการประมวลผลได้หลากหลายรวมทั้งมีความสามารถในการประมวลผลในแบบมาตรฐานเดิมได้ ส่วนพิกเซลเชดเดอร์ (pixel shadder) เป็นโมดูลที่สามารถโปรแกรมได้เช่นเดียวกัน มีหน้าที่ในการควบคุมสีและการผสมสีในส่วนของความโปร่งใส (alpha channel) และในกระบวนการเกี่ยวกับคูลัมน์กำหนดตำแหน่งของเทกเจอร์ (texture coordinate)

6.5 วัตถุไดเร็กทรีดี (Direct3D Object)

ไดเร็กทรีดี ถูกสร้างขึ้นโดยใช้หลักการของ COM object และอินเทอร์เฟซ ดังนั้น โปรแกรมที่เขียนขึ้นในภาษาซีและซีพลัสพลัส (C/C++) จึงสามารถใช้งานอินเทอร์เฟซและออปเจกต์ได้โดยตรง ต่างจากการใช้ภาษาเบสิก (basic) หรือภาษาอื่นๆ ที่ต้องทำงานผ่านชั้นของโค้ดที่ทำหน้าที่ในการแปลงข้อมูลในการติดต่อกับ DirectX runtime อีกทอดหนึ่ง

วัตถุไคเรทรีดีเป็นวัตถุแรกที่ต้องสร้างขึ้นถ้าต้องการจะใช้งานไคเรทรีดี และเป็นวัตถุสุดท้ายที่จะต้องปลดปล่อย (release) หลังจากการใช้งาน เราสามารถตรวจสอบความสามารถของฮาร์ดแวร์และอุปกรณ์อื่นๆ ที่มีอยู่ได้โดยการส่งงานผ่านทางวัตถุไคเรทรีดี ซึ่งจะทำให้เราสามารถตรวจสอบอุปกรณ์ได้โดยไม่ต้องสร้างมันขึ้นมา

การสร้างวัตถุไคเรทรีดี

ขั้นตอนแรก คือการรับค่าพอยเตอร์ที่ชี้ไปยังไคเรทรีดีอินเตอร์เฟส เพื่อใช้ในการเข้าถึงการทำงานต่างๆ ของไคเรทรีดี โดยสามารถทำได้ดังนี้

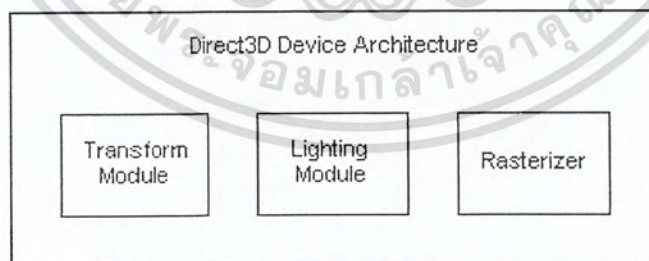
```
LPDIRECT3D8      g_pD3D = NULL;
if ( NULL == (g_pD3D = Direct3DCreate8(D3D_SDK_VERSION)))
    return E_FAIL;
```

เมื่อสร้างดีไวซ์ขึ้นมาแล้วสามารถใช้คำสั่งในการหาพอยเตอร์ที่ชี้ไปยัง ไคเรทรีดีอินเตอร์เฟสที่ใช้สร้างดีไวซ์ได้โดยใช้เมทอด (method) IDirect3DDevice8::GetDirect3D

6.6 ดีไวซ์ในไคเรทรีดี (Direct3D Device)

ดีไวซ์ในไคเรทรีดี คือ องค์ประกอบที่ใช้ในการเรนเดอร์ ซึ่งได้เก็บรักษาสถานะในการเรนเดอร์เอาไว้ (render state) นอกจากนั้นยังทำหน้าที่ในกระบวนการเปลี่ยนแปลงตำแหน่งและให้แสง (transformation and lighting operation) และการลงจุด (rasterize) ของภาพ อีกด้วย

โครงสร้างของดีไวซ์ ดังภาพ



รูปที่ 6-12 รูปแสดงสถาปัตยกรรมการทำงานของดีไวซ์

ชนิดของดีไวซ์

HAL Device

Hardware Abstraction Layer (HAL) device คือ ฮาร์ดแวร์ซึ่งมีความสามารถในการเร่งความเร็วในการลงจุดสี นอกจากนั้นยังอาจมีความสามารถในการเร่งความเร็วในการประมวลผลเวอร์เท็กซ์ (vertex processing) อีกด้วย นั่นคือ HAL device มีการสร้างโมดูลในบางส่วนหรือทั้งหมดของเรนเดอร์ไปป์ไลน์ไว้ในตัวฮาร์ดแวร์ จึงมีประสิทธิภาพในการทำงานสูง แต่ HAL device ไม่สามารถเรนเดอร์บนเซอร์เฟสแบบ 8 บิตได้

การสร้างดีไวซ์แบบ HAL สามารถทำได้โดยเรียกเมทอด IDirect3D8::CreateDevice และกำหนดค่า D3DDEVTYYPE_HAL เป็นชนิดของดีไวซ์

Reference Device

Reference device สนับสนุนการทำงานของ ไดรেকทรีดี ในทุกรูปแบบ เนื่องจากมันได้รับการออกแบบให้ทำงานอย่างถูกต้องมากกว่าต้องการความเร็วในการทำงาน ผลลัพธ์ที่ได้จึงทำให้การทำงานช้ามาก ใช้สำหรับตรวจสอบความถูกต้องในการทำงานของโปรแกรมเท่านั้น

Pluggable Software Device

ในบางกรณีที่อุปกรณ์ในเครื่องคอมพิวเตอร์ไม่สามารถเร่งความเร็วในการประมวลผลบางอย่างได้ ไดรอกทรีดี จะจำลอง 3D ฮาร์ดแวร์ขึ้นมาโดยใช้ซอฟต์แวร์จะทำให้การทำงานช้าลงไป แม้ว่าจะสามารถใช้คำสั่งพิเศษที่มีอยู่ในซีพียูได้ก็ตาม

ชนิดของไพรมิทีฟที่สามารถใช้งานได้ (Device-Supported Primitive Types)

ใน ไดรอกทรีดี ดีไวซ์สามารถสร้างและควบคุมไพรมิทีฟ ดังต่อไปนี้ได้

Point List

คือ กลุ่มของเวอร์เท็กซ์ที่จะได้รับการเรนเดอร์เป็นจุดเดี่ยวๆ สามารถนำไปประยุกต์ใช้ในการทำกลุ่มของดวงดาว และเส้นประบนผิวของโพลีกอนได้

ตัวอย่าง จุดที่ได้รับการเรนเดอร์

$(0, 5, 0)$ $(10, 5, 0)$ $(20, 5, 0)$

$(-5, -5, 0)$ $(5, -5, 0)$ $(15, -5, 0)$

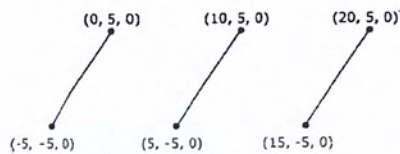
รูปที่ 6-13 รูปแสดง Point List

จุดสามารถมีเมตริกเรียล และเทกเจอร์ได้เช่นกัน โดยสีของเมตริกเรียลหรือเทกเจอร์จะปรากฏเป็นสีของจุดนั้นๆ

Line List

คือ กลุ่มของเวอร์เท็กซ์ที่จะได้รับการเรนเดอร์เป็นเส้นตรงเดี่ยวๆ ซึ่งสามารถนำไปประยุกต์ทำเป็นแผ่นที่กำลังตกได้

ตัวอย่าง เส้นตรงที่ได้รับการเรนเดอร์



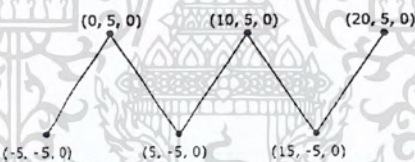
รูปที่ 6-14 รูปแสดง Line List

สามารถกำหนดค่าแมตทรีเรียล และเทกเจอร์ ให้กับเส้นตรงได้เช่นกันโดยสีจะปรากฏไปตามความยาวของเส้นตรง

Line Strips

คือ เส้นตรงที่มีการเชื่อมต่อปลายเข้าด้วยกัน

ตัวอย่าง การเรนเดอร์ดังรูป

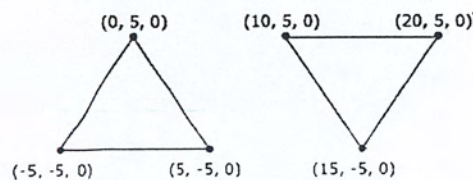


รูปที่ 6-15 รูปแสดง Line Strip

Triangle Lists

คือ จุด 3 จุดที่รวมกันเพื่อสร้างรูป 3 เหลี่ยม 1 รูป โดยไม่เกี่ยวข้องกับรูป 3 เหลี่ยมอื่นๆ เนื่องจากทุกๆ ไพรามีที่พอจะต้องเป็นรูป 3 เหลี่ยม ดังนั้น จำนวนจุดที่ส่งให้กับเรนเดอร์ไปป์ไลน์จะต้องหารด้วย 3 ลงตัว

ตัวอย่าง สามเหลี่ยมที่ได้จากการเรนเดอร์ ดังรูป

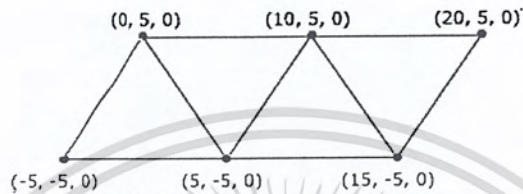


รูปที่ 6-16 รูปแสดง Triangle List

Triangle Strips

คือ กลุ่มของสามเหลี่ยมที่เชื่อมต่อกันไป เนื่องจากการใช้บางจุดร่วมกันจึงทำให้สามารถประหยัดจำนวนข้อมูลได้ จึงทำให้ประสิทธิภาพในการประมวลผลดีขึ้น แต่มีโมเดลจำนวนน้อยที่สามารถทำให้อยู่ในรูปของข้อมูลลักษณะนี้ได้

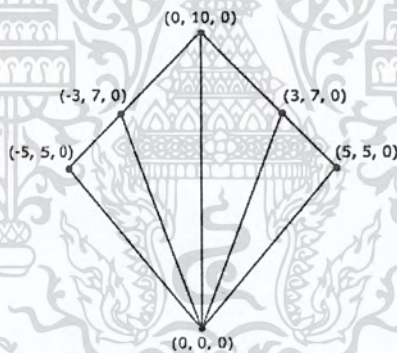
ตัวอย่าง สามเหลี่ยมที่ได้จากการเรนเดอร์ ดังรูป



รูปที่ 6-17 รูปแสดง Triangle Strip

Triangle Fans

มีลักษณะคล้ายกับ Triangle strip แต่จะใช้จุดปลายร่วมกัน จึงมีการนำไปใช้งานได้จำกัด ตัวอย่าง สามเหลี่ยมที่ได้จากการเรนเดอร์ ดังรูป



รูปที่ 6-18 รูปแสดง Triangle Fan

การใช้งานดีไวซ์ (Using Device)

การตรวจสอบฮาร์ดแวร์

ในการตรวจสอบคุณสมบัติที่มีอยู่ในฮาร์ดแวร์ จะต้องใช้เมทอดดังต่อไปนี้

IDirect3D8::CheckDeviceFormat ใช้ในการตรวจสอบว่าฮาร์ดแวร์มีความสามารถในการเรนเดอร์และการเก็บเทกเจอร์บนเซอร์เฟสชนิดใดบ้าง นอกจากนั้นยังใช้ตรวจสอบรูปแบบของสแตนซิลบัฟเฟอร์ (depth-stencil buffer) และเดฟบัฟเฟอร์ (depth buffer) ที่สามารถใช้งานได้

IDirect3D8::CheckDeviceType ใช้ในการตรวจสอบว่าฮาร์ดแวร์มีความสามารถในการเร่งความเร็วในการประมวลผลด้านใดบ้าง, ความสามารถของดีไวซ์ในการทำสวอปเชน (swap chain) ในการแสดงผล และ

ตรวจสอบว่าดีไวซ์มีความสามารถในการเรนเดอร์ในรูปแบบการแสดงผลที่ใช้งานอยู่หรือไม่ (current display format) (จำเป็นในการสร้างโปรแกรมที่ทำงานในวินโดว)

IDirect3D8::CheckDepthStencilMatch ใช้ตรวจสอบว่ารูปแบบของสแตนซิลบัฟเฟอร์ (depth-stencil buffer format) สามารถใช้งานร่วมกับรูปแบบที่ต้องการจะใช้เรนเดอร์ (render-target format) หรือไม่ ซึ่งก่อนจะใช้เมทอดนี้ควรจะตรวจสอบก่อนว่าฮาร์ดแวร์มีรูปแบบของบัฟเฟอร์แต่ละชนิดอย่างไรบ้าง

การสร้างดีไวซ์

ในการสร้างดีไวซ์ ถ้าเขียนโปรแกรมโดยใช้ภาษา C/C++ จะต้องสร้าง วัตถุใดเรกทรีดีขึ้นมาก่อน หลังจากนั้น จึงใส่ค่าซึ่งกำหนดคุณลักษณะของดีไวซ์ที่ต้องการใช้ข้อมูลโครงสร้าง D3DPRESENT_PARAMETER ดังตัวอย่าง

```
LPDIRECT3DDEVICE8 d3dDevice = NULL;
D3DPRESENT_PARAMETERS d3dpp;
ZeroMemory( &d3dpp, sizeof(d3dpp) );
d3dpp.Windowed = TRUE;
d3dpp.SwapEffect = D3DSWAPEFFECT_COPY_VSYNC;
```

หลังจากนั้นจึงเรียกเมทอด IDirect3D::CreateDevice สำหรับสร้างดีไวซ์ในที่นี้กำหนดให้ใช้เดปเตอร์โดยปริยาย (default adapter), สร้างดีไวซ์แบบ HAL และใช้การประมวลผลเวอร์เท็กซ์ด้วยซอฟต์แวร์ (software vertex processing) ดังตัวอย่าง

```
if ( FAILED( g_pD3D->CreateDevice( D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hWnd,
    D3DCREATE_SOFTWARE_VERTEXPROCESSING, &d3dpp, &d3dDevice ) ) )
    return E_FAIL;
```

การกำหนดค่าของเมตริกซ์ทรานสฟอร์มเมชัน (Setting transformation)

ในภาษา C/C++ สามารถเรียกเมทอด IDirect3DDevice::SetTransform โดยส่งค่าเมตริกซ์ที่ต้องการกำหนดและชนิดของทรานสฟอร์มเมชัน ดังตัวอย่างการกำหนดค่าของ view transformation

```
HRESULT hr;
D3DMATRIX view;
// Fill in the view matrix.
if ( FAILED(hr = pDev->SetTransform(D3DTS_VIEW, &view)))
{
    // Code to handle the error goes here;
}
```


ในการกำหนดค่าทรานสฟอร์มเมชันในชนิดอื่นๆ สามารถทำได้โดยกำหนดค่า D3DTS_WORLD, D3DTS_VIEW และ D3DTS_PROJECTION ซึ่งค่าตัวแปรที่สามารถใช้ได้มีการกำหนดเอาไว้ใน D3DTRANSFORMSTATETYPE enumerated type นอกจากนั้นยังประกอบด้วยตัวแปรที่ใช้ในการทรานสฟอร์มเมชันเกี่ยวกับการทำแอนิเมชัน (Animation) และตำแหน่งคู่อันดับของเทกเจอร์ด้วย

6.7 การประมวลผลเวอร์เท็กซ์ (Vertex Processing)

ใน IDirect3DDevice8 อินเทอร์เฟซสนับสนุนการประมวลผลเวอร์เท็กซ์ทั้งในแบบซอฟต์แวร์และฮาร์ดแวร์ ซึ่งโดยปกติแล้วความสามารถของฮาร์ดแวร์ในเครื่องคอมพิวเตอร์แต่ละเครื่องจะแตกต่างกันไป แต่ความสามารถในการประมวลผลของซอฟต์แวร์จะเหมือนกันเสมอ ค่าต่อไปนี้ ใช้ในการควบคุมวิธีการในการประมวลผลเวอร์เท็กซ์สำหรับ Hardware Abstraction Layer (HAL) และ Reference Device

- D3DCREATE_SOFTWARE_VERTEXPROCESSING

- D3DCREATE_HARDWARE_VERTEXPROCESSING

- D3DCREATE_MIXED_VERTEXPROCESSING

D3DCREATE_MIXED_VERTEXPROCESSING คือ การอนุญาตให้ไดรเวอร์สามารถประมวลผลเวอร์เท็กซ์ได้ทั้งในฮาร์ดแวร์และซอฟต์แวร์

ถ้าใช้ D3DCREATE_HARDWARE_VERTEXPROCESSING จะต้องกำหนดด้วยว่าเป็นการสร้างดีไวซ์บริสุทธิ์ (pure device) ด้วยการเพิ่มค่า D3DCREATE_PUREDEVICE เข้าไปด้วย (ใช้โอเพอร์เรเตอร์ |)

6.8 การเรนเดอร์ (Rendering)

เมทอดในกลุ่มของ DrawPrimitive ใช้ในการเรนเดอร์ภาพจาก 3 มิติ ซึ่งต่อไปนี้จะกล่าวถึงขั้นตอนในการทำงานเพื่อที่จะเรนเดอร์ภาพในแต่ละขั้นตอน

1. การล้างเซอร์เฟส (Clearing Surface)

ก่อนที่จะเรนเดอร์ฉากในขณะหนึ่งๆ จะต้องล้างเซอร์เฟสที่อยู่ในวิวพอร์ต (viewport) ที่เราต้องการเสียก่อนเนื่องจากข้อมูลที่มีอยู่เดิมอาจจะทำให้ภาพที่ปรากฏออกมาไม่มีความผิดพลาดได้ การล้างเซอร์เฟสนอกจากจะเป็นการสั่งให้ล้างข้อมูลบนเซอร์เฟสในบริเวณที่เราต้องการล้างแล้วยังเป็นการเปลี่ยนสถานะของเดฟบัฟเฟอร์ให้อยู่ในสถานะที่เราต้องการอีกด้วย ซึ่งเราสามารถกำหนดได้ว่าให้ล้างเซอร์เฟสด้วยค่าสีหรือเทกเจอร์ใดๆ

เมทอดที่ใช้ในการล้างเซอร์เฟส คือ IDirect3DDevice8::Clear

หมายเหตุ ในกรณีที่ทราบแน่นอนแล้วว่าจะมีการเรนเดอร์ภาพใหม่ทั้งจอภาพก็ไม่มี ความจำเป็นต้องล้างเซอร์เฟสในส่วนของคุณค่าสี RGB ซึ่งจะทำให้ได้อัตราเฟรมเพิ่มขึ้นประมาณ 10 เฟรมต่อวินาที

2. การเริ่มต้นและจบฉาก (Beginning and Ending a Scene)

ในภาษา C/C++ ก่อนที่จะทำการเรนเดอร์จะต้องเรียกเมทอด `IDirect3DDevice8::BeginScene` ซึ่ง `BeginScene` เป็นการสั่งให้ระบบทำการตรวจสอบข้อมูลภายในและตรวจสอบว่าเซอร์เฟสที่ต้องการเรนเดอร์มีอยู่จริงหรือไม่ นอกจากนั้นยังกำหนดค่าต่างๆ ภายใน (internal flag) เพื่อกำหนดว่าจะมีการเริ่มการประมวลผลหลักจากเรียกเมทอดนี้แล้วจึงสามารถใช้คำสั่งต่างๆ ในการเรนเดอร์ได้ และการสั่งให้เรนเดอร์โดยไม่เรียกเมทอด `BeginScene` ก่อนจะทำให้การเรนเดอร์ล้มเหลว

หลังจากเรนเดอร์เรียบร้อยแล้วจะต้องเรียกเมทอด `IDirect3DDevice8::EndScene` ซึ่งมีหน้าที่ล้างข้อมูลที่อยู่ในแคช (cache), ตรวจสอบความถูกต้องของเซอร์เฟสที่ทำการเรนเดอร์และทำการล้างค่าต่างๆ ภายใน

นั่นคือ คำสั่งในการเรนเดอร์ทั้งหมดจะต้องอยู่ระหว่างเมทอด `BeginScene` และ `EndScene` เท่านั้น ดังตัวอย่างแสดงการใช้เมทอด

```
HRESULT hr;
```

```
if(SUCCEEDED(pDevice->BeginScene()))
```

```
{
```

```
    // Render primitives only if the scene
```

```
    // starts successfully.
```

```
    // Close the scene.
```

```
    hr = pDevice->EndScene();
```

```
    if(FAILED(hr))
```

```
        return hr;
```

```
}
```

การนำภาพออกมาแสดงผล (Presenting a Scene)

เมทอดเหล่านี้ที่ทำหน้าที่ในการควบคุมสถานะของดีไวส์ในการแสดงผลบนจอคอมพิวเตอร์ ประกอบด้วย

- `IDirect3DDevice8::Present`

- `IDirect3DDevice8::Reset`

- `IDirect3DDevice8::GetGammaRamp`

- `IDirect3DDevice8::SetGammaRamp`

- `IDirect3DDevice8::GetRasterStatus`

ซึ่งมีคำหลายคำที่มีความเกี่ยวข้องกับการใช้งานเมทอดเหล่านี้ซึ่งจำเป็นต้องทราบ ดังนี้

Front Buffer คือ ส่วนของเมมโมรีที่จะได้รับการแปลงโดยกราฟิกอแดปเตอร์ (Graphics Adapter) แล้วแสดงผลออกจากหน้าจอคอมพิวเตอร์หรืออุปกรณ์อื่นๆ

Back Buffer คือ เซอร์เฟสที่ข้อมูลจะถูกเลื่อนขึ้นไปเป็น front buffer

Swap Chain คือ ชุดของ Back Buffer ที่เรียงลำดับเพื่อจะกลายเป็น Front Buffer ซึ่งในการแสดงผลแบบเต็มจอภาพ (full screen) จะมีการแสดงอยู่ในรูปของ flipping DDI (Device Driver Interface) ส่วนการแสดงผลแบบวินโดวจะเป็น blitting DDI

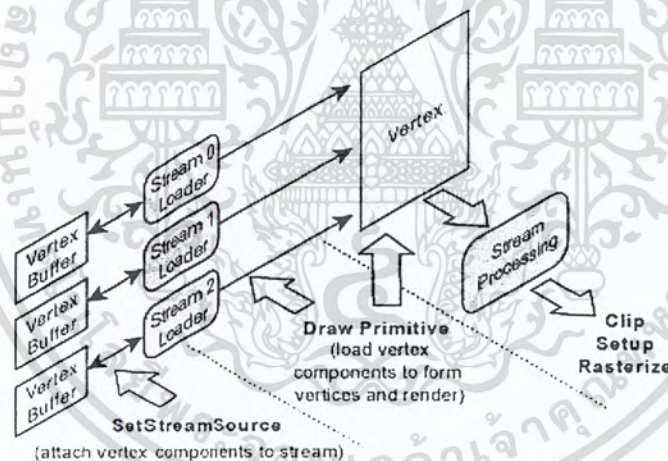
เนื่องจาก Swap Chain เป็นคุณสมบัติของไคลเรทรีดี ดังนั้น ดีไวซ์ทุกชนิดจะต้องมีคุณสมบัติในการทำ Swap Chain อย่างน้อย 1 Swap Chain เสมอ นั่นคือ จะต้องมีการมี Back Buffer อย่างน้อย 1 เซอร์เฟส

การเรนเดอร์ไพรมีทีฟ (Rendering Primitives)

- เวอร์เท็กซ์ดาต้าสตรีม (Vertex Data Stream)

ไคลเรทรีดี มี API ที่จัดการในส่วนของการเรนเดอร์ไพรมีทีฟ ซึ่งไพรมีทีฟที่จะเรนเดอร์ประกอบขึ้นจากข้อมูลของเวอร์เท็กซ์ซึ่งเก็บอยู่ในดาต้าบัฟเฟอร์ ข้อมูลของเวอร์เท็กซ์อาจจะประกอบด้วยตำแหน่ง, สี, ค่าแกนตั้งฉาก (normal) เป็นต้น

องค์ประกอบของเวอร์เท็กซ์อาจจะประกอบขึ้นจากข้อมูลในเมมโมรีบัฟเฟอร์อันเดียวหรือหลายอันก็ได้ นั่นคือ องค์ประกอบของเวอร์เท็กซ์หนึ่งๆ สามารถแบ่งเก็บในเมมโมรีบัฟเฟอร์มากกว่า 1 บัฟเฟอร์ได้ ดังนั้นในการเรนเดอร์ไพรมีทีฟจะต้องอ่านข้อมูลแล้วนำข้อมูลเหล่านั้นมาประกอบกันจนได้เป็นเวอร์เท็กซ์ที่พร้อมจะประมวลผล ดังแผนภาพ



รูปที่ 6-19 รูปแสดงเวอร์เท็กซ์ดาต้าสตรีม

การเรนเดอร์ไพรมีทีฟประกอบด้วย 2 ขั้นตอน คือ

1. การกำหนดค่าให้กับองค์ประกอบในสตรีม
2. การเรียกเมทอดในกลุ่ม DrawPrimitive เพื่อเรนเดอร์สตรีมเหล่านั้น

การกำหนดแหล่งของสตรีม (Setting the Stream Source)

เมทอด `IDirect3DDevice8::SetStreamSource` ใช้ในการกำหนดเวอร์เท็กซ์บัฟเฟอร์ให้กับสตรีมข้อมูลของดีไวซ์ (device data stream) แต่การสร้างความสัมพันธ์ระหว่างสตรีมข้อมูลเหล่านี้จะยังไม่เกิดขึ้นจนกว่าจะมีการเรียกเมทอดในกลุ่มของ `DrawPrimitive`

สตรีม (stream) คือ อาร์เรย์ของข้อมูลที่เป็นองค์ประกอบต่างๆ แต่ละองค์ประกอบอาจจะประกอบด้วยข้อมูลที่ย่อยลงไปอีกได้ ซึ่งใช้ในการแสดงตำแหน่ง, แกนตั้งฉาก, สี เป็นต้น ค่าพารามิเตอร์ `Stride` ใช้ในการกำหนดขนาดขององค์ประกอบเหล่านี้ มีหน่วยเป็นไบต์ (byte)

การเรนเดอร์โดยใช้เวอร์เท็กซ์และอินเด็กซ์บัฟเฟอร์ (Rendering from Vertex and Index Buffers)

ในการเมทอดที่เกี่ยวข้องกับการวาดไพรมีทีฟ สามารถแบ่งออกได้ 2 กลุ่ม คือ แบบที่มีอินเด็กซ์ และแบบที่ไม่มีอินเด็กซ์ ซึ่งการอินเด็กซ์ใช้เป็นข้อมูลในการประกอบไพรมีทีฟขึ้นมาให้เป็นรูปร่าง ใน ไคเรทรีดีจะเก็บอินเด็กซ์ไว้ในอินเด็กซ์บัฟเฟอร์ โดยสามารถกำหนดอินเด็กซ์บัฟเฟอร์ที่จะใช้งานได้โดยการเรียกเมทอด `IDirect3DDevice8::SetIndices`

เมทอดที่ใช้ในการวาดไพรมีทีฟ

- `IDirect3DDevice8::DrawPrimitive` ใช้ในการวาดไพรมีทีฟที่ไม่มีค่าอินเด็กซ์อยู่ในอินเด็กซ์บัฟเฟอร์
- `IDirect3DDevice8::DrawIndexPrimitive` ใช้ในการวาดไพรมีทีฟที่มีค่าอินเด็กซ์อยู่ในอินเด็กซ์บัฟเฟอร์

6.9 สถานะของดีไวซ์ (Device States)

ในไคเรทรีดี ดีไวซ์ทำงานโดยใช้ค่าสถานะในการคำนวณต่างๆ โดยโปรแกรมจะกำหนดค่าสถานะเหล่านี้ ทั้งสถานะที่เกี่ยวข้องกับการคำนวณแสง, การเรนเดอร์ และการควบคุมโมดูลส่วนทรานสฟอร์มเมชัน เป็นต้น สถานะของดีไวซ์สามารถแบ่งออกได้ 2 แบบ ดังนี้

สถานะในการเรนเดอร์ (Render States)

ควบคุมคุณสมบัติในการในส่วนของการลงจุดสี (rasterization module) ซึ่งสามารถใช้ในการเปลี่ยนแปลงคุณสมบัติในการเรนเดอร์, กำหนด shading ที่จะใช้งาน, กำหนดคุณสมบัติของหมอกและกระบวนการในการลงจุดสีอื่นๆ

คุณสมบัติเหล่านี้ในภาษา C/C++ กำหนดได้ด้วยการเรียกเมทอด `IDirect3DDevice8::SetRenderState` ซึ่งค่าสถานะของดีไวซ์ได้ถูกกำหนดเอาไว้แล้วในชนิดข้อมูล `D3DRENDERSTATETYPE`

การประมวลผลเวอร์เท็กซ์แบบฟังก์ชันตายตัว สามารถกำหนดคุณสมบัติการทำงานได้โดยการกำหนดค่าสถานะเรนเดอร์ แต่ถ้าใช้การประมวลผลเวอร์เท็กซ์แบบโปรแกรมได้ (Programmable Vertex Processing) คุณสมบัติหลายๆ อย่างที่กำหนดโดยใช้สถานะในการเรนเดอร์จะไม่มีผล

นอกจากเมทอด `SetRenderState` แล้วยังมีเมทอดอื่นๆ ที่เกี่ยวข้องกับการกำหนดการทำงานของไปป์ไลน์ประมวลผลเวอร์เท็กซ์แบบฟังก์ชันตายตัว ซึ่งเกี่ยวข้องกับการกำหนดค่าเมตริกซ์ทรานสฟอร์มเมชัน, การกำหนดเมตทีเรียล (material) และกำหนดแสง ดังนี้

- `IDirect3DDevice8::SetTransform`
- `IDirect3DDevice8::SetMaterial`
- `IDirect3DDevice8::SetLight`
- `IDirect3DDevice8::LightEnable`

ค่าสถานะของชั้นเทกเจอร์ (Texture Stage States)

สถานะของชั้นเทกเจอร์ ใช้ในการควบคุมรูปแบบของการประมวลผลเทกเจอร์และการกำหนดฟิลเตอร์ (filter) ที่จะใช้งาน

โปรแกรมที่เขียนในภาษา C/C++ สามารถกำหนดคุณสมบัติของเทกเจอร์เหล่านี้ได้โดยการเรียกเมทอด `IDirect3DDevice8::SetTextureStageState` ซึ่งค่าสถานะของชั้นเทกเจอร์ได้ถูกกำหนดเอาไว้แล้วในชนิดข้อมูล `D3DTEXTURESTAGESTATETYPE` และจะต้องใส่เป็นพารามิเตอร์ตัวแรกในเมทอด `SetTextureStageState`

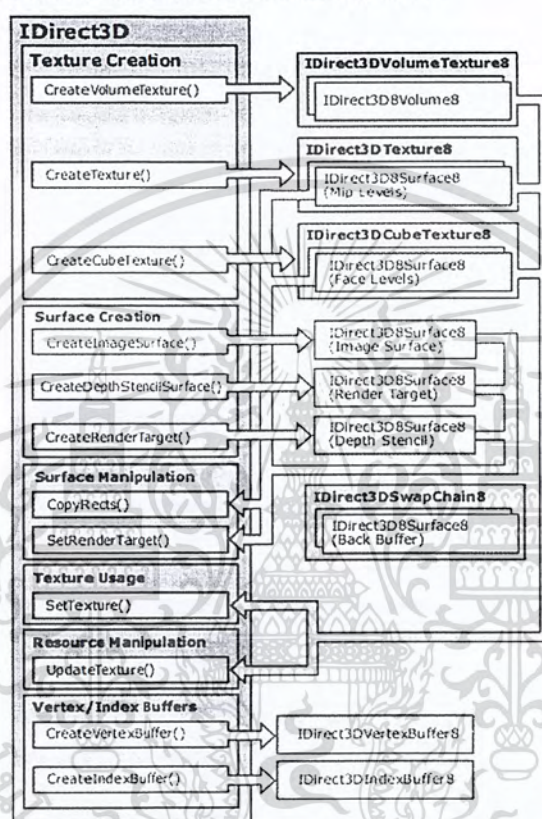
นอกจากนั้น การกำหนดเทกเจอร์เซอร์เฟสที่จะใช้งานสามารถทำได้โดยการเรียกเมทอด `IDirect3DDevice::SetTexture`

6.10 รีซอร์สใน ไดรเรกทรีดี (Direct3D Resources)

ในไดเรกทรีดี รีซอร์ส (resource) คือ เทกเจอร์และบัฟเฟอร์ที่ใช้เก็บข้อมูลที่จะถูกใช้ในการเรนเดอร์ โปรแกรมมีความจำเป็นจะต้องสร้าง, โหลด, คัดลอก และใช้งานรีซอร์สเหล่านี้ ซึ่งประกอบด้วย เทกเจอร์ (texture), เวอร์เท็กซ์บัฟเฟอร์ (Vertex Buffer) และอินเด็กซ์บัฟเฟอร์ (Index Buffer)

ความสัมพันธ์ของรีซอร์ส (Resource Relationships)

แผนภาพต่อไปนี้ แสดงความสัมพันธ์ของเมทอด กับรีซอร์สแต่ละชนิด



รูปที่ 6-20 รูปแสดงความสัมพันธ์ของรีซอร์สในไดเรกทรีดี

ลูกศรที่วิ่งจากซ้ายไปขวาแสดงว่าเมทอดสามารถสร้างรีซอร์สชนิดนั้นได้ ลูกศรที่วิ่งจากทางขวาไปทางซ้ายแสดงว่ารีซอร์สชนิดนั้นสามารถใช้เป็นอาร์กิวเมนต์ (argument) ในเมทอดนั้นได้

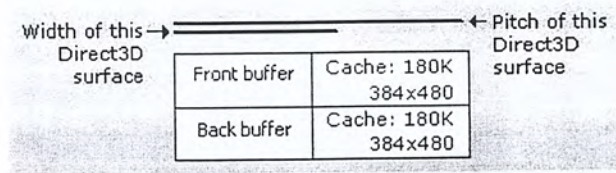
6.11 เซอร์เฟส (Surfaces)

เซอร์เฟส คือ พื้นที่ในเมมโมรีส่วนที่ใช้ในการแสดงผล (display memory) ซึ่งทุกๆ ไปจะอยู่ในเมมโมรีที่อยู่บนการ์ดแสดงผล (display card) แต่ก็สามารถอยู่ในเมมโมรีระบบ (system memory) ได้เช่นกัน เซอร์เฟสถูกบรรจุอยู่ใน IDirect3DSurface8 interface

ความกว้างและค่าพิทช์ (Width and Pitch)

คำว่าความกว้าง (width) และพิทช์ (pitch) มีความหมายแตกต่างกัน ความกว้างในไดเรกทรีดีที่เกี่ยวข้องกับเซอร์เฟสจะหมายถึง มิติของเซอร์เฟสในทางลอจิก (logical) ในส่วนของความกว้าง มีหน่วยเป็นพิกเซล

(pixel) ดังนั้นจึงมีค่าเท่ากันเสมอไม่ว่าพิกเซลจะมีขนาดกี่บิตก็ตาม ส่วนค่าพิทช์ มีความหมายถึงระยะของข้อมูลเป็นจำนวนไบต์ และมักจะมีการรวมแรมโมรีในส่วนที่เป็นแคช (cache) เข้าไปด้วย ดังรูป



รูปที่ 6-21 รูปแสดงความกว้างและค่าพิทช์

ค่าพิทช์มีความจำเป็นในกรณีที่ต้องการเข้าถึงแรมโมรีโดยตรง ซึ่งต้องคำนึงถึงขนาดของแรมโมรีโดยห้ามไปเขียนในส่วนของแคชที่สงวนเอาไว้

รูปแบบของเซอร์เฟส (Surface Format)

รูปแบบของเซอร์เฟสเป็นตัวกำหนดชนิดของข้อมูลในแต่ละพิกเซลซึ่งเก็บอยู่ในเซอร์เฟสแรมโมรี ในโคเรกทรีตี ใช้ชนิดข้อมูล D3DFORMAT ซึ่งเป็นสมาชิกของ D3DSURFACE_DESC ในการอธิบายรูปแบบของเซอร์เฟส โดยสามารถหาค่ารูปแบบของเซอร์เฟสใดๆ ได้โดยเรียกเมทอด IDirect3DSurface8:: GetDesc

6.12 แสง (Lights)

เมื่อมีการกำหนดให้มีการคำนวณแสงในโคเรกทรีตีแล้วในกระบวนการลงจุดสี ซึ่งเป็นขั้นตอนสุดท้ายในเรนเดอร์ไปป์ไลน์ จะมีการคำนวณค่าสีของพิกเซลด้วยผลรวมระหว่างค่าเมตทีเรียล (material), เทกเซล (texel) ซึ่งได้มาจากการปะเทกเจอร์ และความเข้มของแสงและสีของแสงที่ปรากฏอยู่ในฉากซึ่งเกิดจากแหล่งแสงชนิดต่างๆ (light source) หรือระดับของแสงแอมเบียนท์ (ambient light level) เมื่อมีการกำหนดค่าในสถานะการเรนเดอร์ ซึ่งการกำหนดค่าเหล่านี้หมายความว่าต้องการให้ โคเรกทรีตี จัดการรายละเอียดทั้งหมดให้ แต่สำหรับผู้ที่มีความชำนาญมากๆ ก็สามารถคำนวณค่าเกี่ยวกับแสงได้เองเช่นกัน

การใช้แสงและเมตทีเรียลจะทำให้ภาพที่ได้มีความสวยงามและสมจริงขึ้นอย่างมาก ค่าของเมตทีเรียลเป็นการกำหนดคุณสมบัติในการสะท้อนแสงของพื้นผิว ส่วนค่าของแสงเป็นการกำหนดแสงที่ส่งออกไปเพื่อให้สะท้อน ดังนั้น ถ้าต้องการจะใช้แสงเพื่อเพิ่มความสมจริงและสวยงามให้วัตถุจำเป็นต้องกำหนดค่าเมตทีเรียลให้วัตถุด้วยเสมอ (มิฉะนั้น วัตถุจะอยู่ในสภาพที่มองไม่เห็นหรือเป็นสีดำ)

Direct Light และ Ambient Light

ทั้ง Direct Light และ Ambient Light ต่างก็เป็นวัตถุที่ส่องแสงออกมาได้ แต่วัตถุทั้งสองไม่ขึ้นแก่กันและกัน และให้ผลลัพธ์ในการทำงานที่แตกต่างกัน

Direct Light จะต้องมิติศทางและสีเสมอ ค่าเหล่านี้ถูกใช้ในการคำนวณใน shading algorithm เช่น Gouraud shading ด้วย แสงที่มีชนิดต่างกันจะมีวิธีการเปล่งแสงแตกต่างกันไป ซึ่งจะทำให้เกิดเอฟเฟกต์ต่างๆ กัน แสงชนิดนี้สามารถกำหนดได้ด้วยการเรียกเมทอด `IDirect3DDevice::SetLight`

Ambient Light ส่งผลต่อวัตถุทั้งหมดในฉากไม่ว่าอยู่จุดไหนก็ตาม นั่นก็คือ เป็นระดับของแสงที่จะใส่ให้กับวัตถุที่มีอยู่ในฉากทั้งหมด ดังนั้น มันจึงไม่มีทิศทางมีแต่ค่าที่แสดงสีและความเข้มของแสงเท่านั้น สามารถกำหนดค่าได้ด้วยการเรียกเมทอด `IDirect3DDevice::SetRenderState` โดยใส่ค่า `D3DRS_AMBIENT` ในพารามิเตอร์ที่กำหนดชนิดของสถานะ และค่าสีที่ต้องการในพารามิเตอร์ตัวที่สอง

ชนิดของแสง (Light Object)

ในไดเรกทรีมีแสงอยู่ 3 ชนิดคือ จุดแสง (point light), สปอตไลท์ (spotlight) และแสงทิศทาง (Directional light) ซึ่งแสงแต่ละชนิดมีรายละเอียดดังนี้

จุดแสง (Point Light)

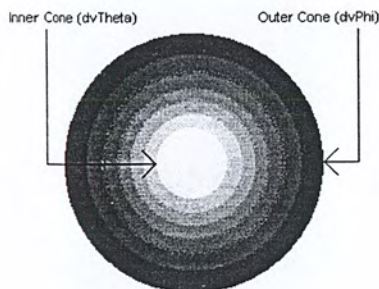
จุดแสงมีสีและตำแหน่งในฉากแต่ไม่มีทิศทางเดียว มันจะให้แสงเท่าๆ กันในทุกทิศทาง ดังรูป

รูปที่ 6-22 รูปแสดงจุดแสง

หลอดไฟฟ้าแบบหลอดไส้เป็นตัวอย่างของจุดแสง ค่าของแสงที่ออกมาจากจุดแสงถูกกำหนดด้วยพารามิเตอร์ attenuation ซึ่งกำหนดลักษณะการจางลงของความเข้มแสง และระยะทางที่แสงส่องไปถึง (range) ไดเรกทรีดี ใช้ตำแหน่งของจุดแสง, ระยะทางที่แสงเดินทาง และค่าตั้งฉากของเวกเตอร์ที่กซ์ในการคำนวณความสว่างของพื้นผิว

สปอตไลท์ (Spotlight)

สปอตไลท์ มีทั้งสี, ตำแหน่ง และทิศทาง แสงที่เปล่งออกมาจากสปอตไลท์มีลักษณะเป็นกรวยซึ่งส่วนที่อยู่ในสุดจะมีความสว่างมากที่สุดและค่อยๆ ลดลงเมื่อออกห่างไปเรื่อยๆ ความต่างของความเข้มแสงแสดงดังรูป



รูปที่ 6-23 รูปแสดงสปอตไลท์

ไฟฉายเป็นตัวอย่างของสปอตไลท์ ความเข้มของแสงที่เปล่งออกจากสปอตไลท์ถูกกำหนดด้วยค่า falloff, attenuation และระยะทางที่แสงเดินทางไปถึง (range) เนื่องจากมีพารามิเตอร์จำนวนมากการคำนวณแสงจากสปอตไลท์จึงมีความซับซ้อนมากที่สุดและเปลืองเวลาในการประมวลผลที่สุด

แสงทิศทาง (Directional Light)

มีแต่ค่าของสี และทิศทาง ไม่มีค่าซึ่งแสดงตำแหน่ง เนื่องจากเปล่งแสงในลักษณะคู่ขนาน นั่นคือ เป็นแสงที่ตกกระทบวัตถุทั้งหมดในฉากด้วยทิศทางเดียวกัน เช่นเดียวกับแสงจากดวงอาทิตย์ (เนื่องจากเดินทางมาเป็นระยะทางที่ไกลมากจึงเสมือนว่าเดินทางขนานกันมา) แสงทิศทางไม่มีค่า attenuation หรือระยะทาง ดังนั้นการคำนวณจึงมีความซับซ้อนน้อยที่สุดและเปลืองการประมวลผลน้อยที่สุดเช่นกัน

คุณสมบัติของแสง (Light properties)

คุณสมบัติของแสงอธิบายชนิดของแหล่งแสง (light source) และสี นอกจากนั้นยังกำหนดค่าที่มีความจำเป็นสำหรับแสงในแต่ละชนิด (แสงแต่ละชนิดใช้ค่าพารามิเตอร์ไม่เหมือนกัน) ในโคเรกทรีดีใช้โครงสร้างข้อมูล D3DLIGHT8 ในการเก็บข้อมูลเกี่ยวกับแสงทั้งหมด ซึ่งประกอบด้วยข้อมูลที่แบ่งเป็นกลุ่มต่างๆ ได้ดังนี้

- ชนิดของแสง (Light Type)
- สีของแสง (Light Color)
- ค่าเมตริกซ์เรียวของวัตถุ (Color vertices)
- ตำแหน่ง, ระยะทาง และ attenuation ของแสง
- ทิศทางของแสง (Light direction)
- คุณสมบัติของสปอตไลท์ (spotlight properties)

การใช้แสง (Using Lights)

การกำหนดค่าพารามิเตอร์ของแสง (Setting Light Properties)

ในภาษา C/C++ สร้างกำหนดค่าของแสงได้ด้วยการกำหนดค่าพารามิเตอร์ต่างๆ ที่มีอยู่ในโครงสร้างข้อมูล D3DLIGHT8 แล้วเรียกเมทอด IDirect3DDevice::SetLight ซึ่งจะต้องระบุค่าอินเด็กซ์ของแสงที่จะกำหนดด้วย เนื่องจากในไดเรกทรีดีสามารถกำหนดแหล่งแสงได้พร้อมกันสูงสุด 8 แหล่งเท่านั้น

ตัวอย่างการกำหนดค่าของแหล่งแสง

```
D3DLIGHT8 d3dLight;
HRESULT hr;
// Initialize the structure.
ZeroMemory(&d3dLight, sizeof(D3DLIGHT8));
// Set up a white point light.
d3dLight.Type = D3DLIGHT_POINT;
d3dLight.Diffuse.r = 1.0f;
d3dLight.Diffuse.g = 1.0f;
d3dLight.Diffuse.b = 1.0f;
d3dLight.Ambient.r = 1.0f;
d3dLight.Ambient.g = 1.0f;
d3dLight.Ambient.b = 1.0f;
d3dLight.Specular.r = 1.0f;
d3dLight.Specular.g = 1.0f;
d3dLight.Specular.b = 1.0f;
d3dLight.Position.x = 0.0f;
d3dLight.Position.y = 1000.0f;
d3dLight.Position.z = -100.0f;
```

```
// Don't attenuate.
```

```
d3dLight.Attenuation0 = 1.0f;
d3dLight.dvRange = 1000.0f;
```



```
// Set the property information for the first light.
```

```
hr = d3dDevice->SetLight(0, &d3dLight);
```

```
if (FAILED(hr))
```

```
{
```

```
    // Code to handle the error goes here.
```

```
}
```

เมื่อกำหนดค่าของแสงในอินเด็กซ์ใดๆ ไปแล้วสามารถเปลี่ยนแปลงค่าของแสงได้ด้วยการเรียกเมทอด SetLight ด้วยค่าของแสงที่ต้องการอีกครั้ง

การสั่งให้มีการคำนวณแสง (Enabling and Disabling Lights)

เมื่อกำหนดค่าของแหล่งแสงในอินเด็กซ์ใดๆ แล้วการจะให้ไคเรกทรีดี้นำค่าพารามิเตอร์เหล่านั้นไปใช้ในการคำนวณด้วยจะต้องเรียกเมทอด IDirect3DDevice::LightEnable ซึ่งโดยปกติแหล่งแสงจะไม่มีการนำไปใช้คำนวณ (disable) ไว้ เมทอด LightEnable ต้องการอาร์กิวเมนต์ 2 ตัว ค่าแรกคือ อินเด็กซ์ของแหล่งแสง ส่วนค่าที่สองคือ พารามิเตอร์ที่บอกว่าให้นำค่าไปคำนวณหรือไม่ (TRUE ถ้าต้องการให้คำนวณและ FALSE ถ้าไม่ต้องการ)

ตัวอย่างการกำหนดให้ ไคเรกทรีดี นำค่าของแหล่งแสงไปใช้ในการคำนวณ

```
/*
```

```
* For the purposes of this example, the d3dDevice variable
```

```
* is a valid pointer to an IDirect3DDevice8 interface.
```

```
*/
```

```
HRESULT hr;
```

```
hr = pd3dDevice->LightEnable(0, TRUE);
```

```
if (FAILED(hr))
```

```
{
```

```
    // Code to handle the error goes here.
```

```
}
```

แต่ถ้ามีการสั่งให้นำค่าในอินเด็กซ์ที่ไม่มีการกำหนดค่าไว้ไปคำนวณไคเรกทรีดีจะนำค่าในตารางต่อไปนี้ไปใช้ในการคำนวณแทน

Member	Default
Type	D3DLIGHT_DIRECTIONAL
Diffuse	(R:1, G:1, B:1, A:0)
Specular	(R:0, G:0, B:0, A:0)
Ambient	(R:0, G:0, B:0, A:0)
Position	(0, 0, 0)
Direction	(0, 0, 1)
Range	0
Falloff	0
Attenuation0	0
Attenuation1	0
Attenuation2	0
Theta	0
Phi	0

ตารางที่ 6-1 ตารางแสดงค่าโดยปริยายของแสง

การหาค่าของแสงที่กำหนดไว้แล้ว (Retrieving Light Properties)

สามารถหาค่าคุณสมบัติของแสงที่มีอยู่แล้วได้ด้วยการเรียกเมทอด IDirect3DDevice::GetLight แล้วส่งค่าอาร์กิวเมนต์แรกเป็นอินเด็กซ์ของแสงที่ต้องการ ดังตัวอย่าง

```
HRESULT hr;
D3DLIGHT8 light;
// Get the property information for the first light.
hr = pd3dDevice->GetLight(0, &light);
if (FAILED(hr))
{
    // Code to handle the error goes here.
}
```

ถ้าเรากำหนดค่าอินเด็กซ์ของแสงที่ไม่มีอยู่ในใดเรกทรีดี (มีเฉพาะค่า 0-7 เท่านั้นเนื่องจากมีแสงได้สูงสุดเพียง 8 แหล่งแสง) เมทอด GetLight จะส่งค่า D3DERR_INVALIDCALL คืนมา

6.13 แมตทีเรียล (Materials)

แมตทีเรียลเป็นค่าที่อธิบายวิธีการสะท้อนแสงของโพลีกอนเนื่องจากแหล่งแสงที่มีอยู่ในฉาก มีความสำคัญในการคำนวณค่าดังต่อไปนี้

- วิธีการที่พื้นผิวสะท้อนแสงดิฟฟิวส์ (diffuse light) และแสงแอมเบียน (ambient light)
- ลักษณะความมันวาว (specular highlight)
- โพลีกอนมีคุณสมบัติในการเปล่งแสงหรือไม่ อย่างไร (emissive)

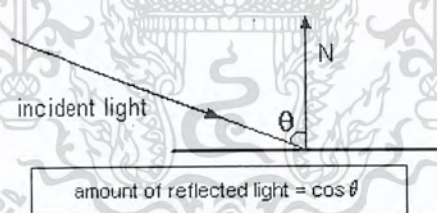
ในภาษา C/C++ คุณสมบัติของแสงใช้โครงสร้างข้อมูล D3DMATERIAL8 ในการเก็บข้อมูลเกี่ยวกับแมตทีเรียล

คุณสมบัติของแมตทีเรียล (Material Properties)

คุณสมบัติของแมตทีเรียลให้รายละเอียดเกี่ยวกับคุณสมบัติต่างๆ ดังต่อไปนี้

1. วิธีการสะท้อนแสงดิฟฟิวส์และแสงแอมเบียน (Diffuse and Ambient Reflection)

ค่า Diffuse และ Ambient เป็นสมาชิกในโครงสร้างข้อมูล D3DMATERIAL8 ซึ่งใช้อธิบายวิธีการสะท้อนแสงทั้งสองชนิด แต่เนื่องจากแสงดิฟฟิวส์โดยปกติแล้วจะส่งผลมากกว่าแสงแอมเบียน คุณสมบัติในการสะท้อนแสงดิฟฟิวส์จึงเป็นคุณสมบัติที่ปรากฏให้เห็นเด่นชัดกว่า เนื่องจากแหล่งแสงส่วนมากจะมีทิศทาง ดังนั้นการสะท้อนแสงดิฟฟิวส์จะสะท้อนได้ดียิ่งขึ้นถ้าทิศทางของแหล่งแสงขนานกับทิศทางตั้งฉากของเวอร์เท็กซ์ แต่ถ้ามุมระหว่างแหล่งแสงกับทิศทางตั้งฉากของเวอร์เท็กซ์เกิดขึ้นมากเท่าไรก็ยิ่งทำให้ความเข้มของแสงอ่อนลงเท่านั้น ปริมาณความเข้มของแสงที่กระทบพื้นผิวคือค่า cosine ของมุมระหว่างทิศทางของแหล่งแสงและทิศทางตั้งฉากของเวอร์เท็กซ์ ดังรูป



รูปที่ 6-24 รูปแสดงวิธีการคำนวณทางแสง

การสะท้อนแสงแอมเบียนก็เหมือนกับแหล่งแสงแอมเบียน คือ ไม่มีทิศทาง การสะท้อนแสงแอมเบียนมีผลต่อวัตถุน้อย แต่มีผลต่อวัตถุทุกวัตถุที่มีอยู่ในฉากและถึงแม้จะมีผลน้อยแต่การเปลี่ยนแปลงของแสงแอมเบียนและค่าการสะท้อนแสงแอมเบียนก็ส่งผลกระทบต่อภาพที่ปรากฏได้เช่นเดียวกัน

ดังนั้น จึงมีความจำเป็นอย่างยิ่งที่จะต้องใช้คุณสมบัติการสะท้อนแสงทั้งแสงดิฟฟิวส์และแสงแอมเบียนในการกำหนดสีของวัตถุที่อยู่ในฉาก และโดยปกติแล้วจะใช้ค่าเดียวกัน เช่น ถ้าต้องการกำหนดสีให้ผลึกคริสตัลเป็นสีฟ้า ก็จะกำหนดค่าให้คุณสมบัติในการสะท้อนแสงสีฟ้าทั้งในค่าของดิฟฟิวส์และแอมเบียน ดังนั้น ถ้าในฉากมีแหล่งแสงสีขาวก็จะมองเห็นคริสตัลเป็นสีฟ้า แต่ถ้าในฉากมีแต่แหล่งแสงสีแดงก็จะมองเห็นคริสตัลเป็นสีแดงเป็นต้น

การเปล่งแสง (Emission)

ใช้ในการทำให้วัตถุดูเหมือนว่าเปล่งแสงได้ ค่า Emissive ซึ่งเป็นสมาชิกในโครงสร้างข้อมูล D3DMATERIAL8 ใช้ในการกำหนดสีและความโปร่งใสของแสงที่เปล่งออกมาจากวัตถุ เราใช้ค่าการเปล่งแสงของวัตถุในกรณีที่ต้องการทำให้วัตถุดูเหมือนกับเปล่งแสงได้โดยไม่ต้องการให้ประมวลผลแหล่งแสงเพิ่มเนื่องจากจะใช้การคำนวณมากกว่า แต่สิ่งที่แตกต่างจากการใช้แหล่งแสงคือ แสงที่เปล่งออกมาจะไม่มีผลต่อวัตถุอื่นๆ ในฉากเลย (จึงใช้การคำนวณน้อยกว่า)

ความมันวาว (Specular Reflection)

เป็นค่าที่กำหนดลักษณะความมันวาวบนวัตถุ ซึ่งทำให้วัตถุมีความสว่างยิ่งขึ้น ในโครงสร้างข้อมูล D3DMATERIAL8 ใช้สมาชิก 2 ตัวในการกำหนดลักษณะความมันวาวของวัตถุ คือ Specular ใช้กำหนดสีของความมันวาวที่จะปรากฏ และ Power ใช้กำหนดขนาดของความมันวาว ดังภาพตัวอย่างด้านซ้ายมือ กำหนดค่า Power เท่ากับ 10 และค่า Specular เป็นสีขาว ส่วนด้านขวามือไม่ได้กำหนดค่าความมันวาวให้ (กำหนดให้แต่ค่า Diffuse และค่า Ambient)



รูปที่ 6-25 รูปแสดงสเปกคิวลาร์

การใช้งานเมตทีเรียล (Using Materials)

การกำหนดคุณสมบัติของเมตทีเรียล (Setting Material Properties)

ในภาษา C/C++ การกำหนดคุณสมบัติของเมตทีเรียลสามารถทำได้โดยการกำหนดค่าต่างๆ ในโครงสร้างข้อมูล D3DMATERIAL8 แล้วส่งเป็นอาร์กิวเมนต์ให้กับเมทอด IDirect3DDevice8:: SetMaterial

```
D3DMATERIAL8 mat;
```

```
// Set the RGBA for diffuse reflection.
```

```
mat.Diffuse.r = 0.5f;
```

```
mat.Diffuse.g = 0.0f;
```

```
mat.Diffuse.b = 0.5f;
```

```
mat.Diffuse.a = 1.0f;
```



```
// Set the RGBA for ambient reflection.
```

```
mat.Ambient.r = 0.5f;
```

```
mat.Ambient.g = 0.0f;
```

```
mat.Ambient.b = 0.5f;
```

```
mat.Ambient.a = 1.0f;
```

```
// Set the color and sharpness of specular highlights.
```

```
mat.Specular.r = 1.0f;
```

```
mat.Specular.g = 1.0f;
```

```
mat.Specular.b = 1.0f;
```

```
mat.Specular.a = 1.0f;
```

```
mat.Power = 50.0f;
```

หลังจากกำหนดค่าให้กับโครงสร้างข้อมูลแล้วจึงเรียกเมทอด `IDirect3DDevice::SetMaterial` เพื่อ
กำหนดค่าให้กับดีไวซ์ที่จะทำการเรนเดอร์ไพรมีทีฟต่อไป ดังตัวอย่าง

```
HRESULT hr;
```

```
hr = pd3dDev->SetMaterial(&mat);
```

```
if(FAILED(hr))
```

```
{
```

```
    // Code to handle the error goes here.
```

```
}
```

เมื่อมีการสร้างไคลเรทรีดีดีไวซ์ค่าของแมตทีเรียลจะมีการกำหนดโดยอัตโนมัติเป็นค่าดังในตารางต่อไปนี้

Member	Value
Diffuse	(R:1, G:1, B:1, A:0)
Specular	(R:0, G:0, B:0, A:0)
Ambient	(R:0, G:0, B:0, A:0)
Emissive	(R:0, G:0, B:0, A:0)
Power	(0.0)

ตารางที่ 6-2 ค่าโดยปริยายที่กำหนดให้กับแมตทีเรียล

การหาค่าของคุณสมบัติของแมตทีเรียล (Retrieving Material Properties)

สามารถหาค่าคุณสมบัติของแมตทีเรียลที่ใช้งานอยู่ได้โดยการเรียกเมทอด IDirect3DDevice8::

GetMaterial ดังตัวอย่าง

```
HRESULT hr;
```

```
D3DMATERIAL8 mat;
```

```
hr = pd3dDev->GetMaterial(&mat);
```

```
if(FAILED(hr))
```

```
{
```

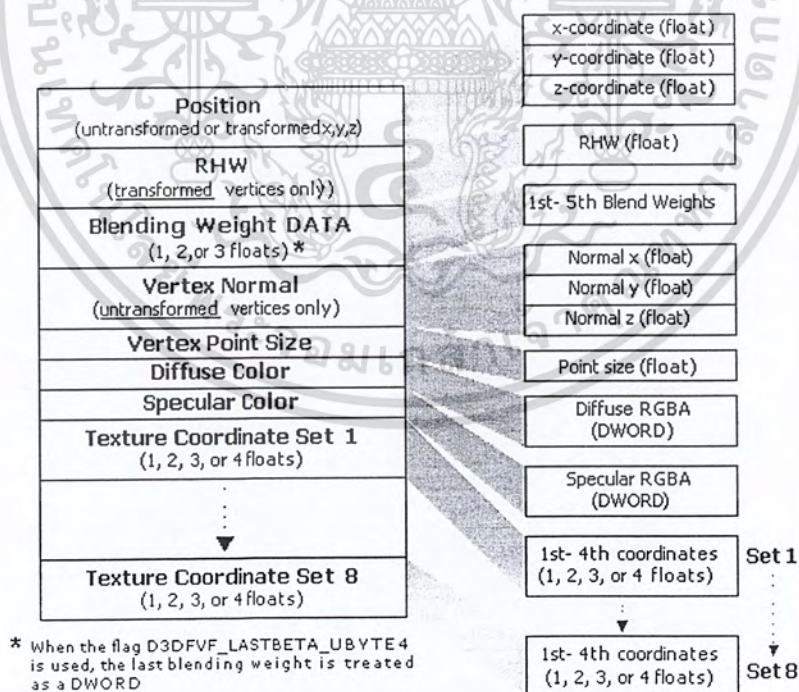
```
    // Code to handle the error goes here.
```

```
}
```

6.14 รูปแบบของเวอร์เท็กซ์ (Vertex Formats)

Flexible Vertex Format (FVF) ใช้ในการอธิบายว่าในข้อมูลเวอร์เท็กซ์มีการเก็บข้อมูลอะไรไว้บ้างซึ่งอยู่ในคำสั่งสตรีมเดียวกัน (data stream) ซึ่งโดยปกติแล้วจะใช้ในการกำหนดข้อมูลสำหรับไปป์ไลน์ที่ใช้ประมวลผลเวอร์เท็กซ์แบบฟังก์ชันตายตัว

แผนภาพต่อไปนี้แสดงลำดับของข้อมูลที่สามารถเลือกใช้งานได้ โดยการเก็บข้อมูลจะต้องเก็บเรียงตามลำดับการเก็บในแผนภาพนี้ด้วย



รูปที่ 6-26 รูปแสดงการเรียงลำดับการเก็บข้อมูลในเวอร์เท็กซ์

ค่าคู่อันดับแสดงตำแหน่งของเทกเจอร์ สามารถประกาศใช้ชนิดข้อมูลแบบอื่นก็ได้โดยสามารถมีได้ตั้งแต่ 1 ค่าถึง 4 ค่า

การเขียนโปรแกรมโดยปกติแล้วจะไม่ได้ใช้องค์ประกอบที่มีอยู่ทั้งหมดเนื่องจากบางข้อมูลบ้างค่าไม่สามารถใช้พร้อมกันได้ เช่น D3DFVF_XYZ และ D3DFVF_XYZRHW

การใช้ indexed vertex blending จะต้องประกาศค่า D3DFVF_LASTBETA_UBYTE4 ในการกำหนดข้อมูลสำหรับการทำแอนิเมชัน (Animation) ด้วย

ตัวอย่างการกำหนดชนิดข้อมูลในเวอร์เท็กซ์

```
#define D3DFVF_BLENDVERTEX (D3DFVF_XYZB3|D3DFVF_NORMAL|D3DFVF_TEX1)
```

```
struct BLENDVERTEX
```

```
{
```

```
    D3DXVECTOR3 v;           // Referenced as v0 in the vertex shader
    FLOAT    blend1;         // Referenced as v1.x in the vertex shader
    FLOAT    blend2;         // Referenced as v1.y in the vertex shader
    FLOAT    blend3;         // Referenced as v1.z in the vertex shader
                             // v1.w = 1.0 - (v1.x + v1.y + v1.z)
```

```
    D3DXVECTOR3 n;           // Referenced as v3 in the vertex shader
```

```
    FLOAT    tu, tv;         // Referenced as v7 in the vertex shader
```

```
};
```

ตัวอย่างการกำหนดชนิดของข้อมูลในเวอร์เท็กซ์โดยใช้ D3DFVF_LAST_UBYTE4 flag

```
#define D3DFVF_BLENDVERTEX (D3DFVF_XYZB4 | D3DFVF_LASTBETA_UBYTE4
|D3DFVF_NORMAL|D3DFVF_TEX1)
```

```
struct BLENDVERTEX
```

```
{
```

```
    D3DXVECTOR3 v;           // Referenced as v0 in the vertex shader
    FLOAT    blend1;         // Referenced as v1.x in the vertex shader
    FLOAT    blend2;         // Referenced as v1.y in the vertex shader
    FLOAT    blend3;         // Referenced as v1.z in the vertex shader
                             // v1.w = 1.0 - (v1.x + v1.y + v1.z)
```

```
    DWORD    indices;        // Referenced as v2.xyzw in the vertex shader
```

```
    D3DXVECTOR3 n;           // Referenced as v3 in the vertex shader
```

```
    FLOAT    tu, tv;         // Referenced as v7 in the vertex shader
```

```
};
```

6.15 เทกเจอร์ (Textures)

เทกเจอร์เป็นสิ่งที่ช่วยสร้างความสวยงามและสมจริงให้กับวัตถุใน 3 มิติได้อย่างมาก ดังนั้น ไคเรกทรีดี จึงสนับสนุนคุณสมบัติในการจัดการเทกเจอร์ไว้อย่างมาก เพื่อให้ผู้ที่พัฒนาสามารถนำคุณสมบัติเหล่านี้ไปใช้ในการประมวลผลเทกเจอร์ในระดับสูงได้

เทกเจอร์ คือ ภาพบิตแมพที่ปะลงบนโพรเจกทีฟใน ไคเรกทรีดี ซึ่งทำให้ภาพที่ได้มีลักษณะที่เหมือนจริงยิ่งขึ้น เช่น ถ้าเราต้องการจำลองต้นไม้ก็นำภาพบิตแมพที่เป็นลายต้นไม้มาปะในส่วนที่เป็นลำต้น และนำภาพบิตแมพของใบไม้มาปะบริเวณที่เป็นใบไม้ก็จะได้ว่าวัตถุที่ดูเหมือนต้นไม้ยิ่งขึ้น

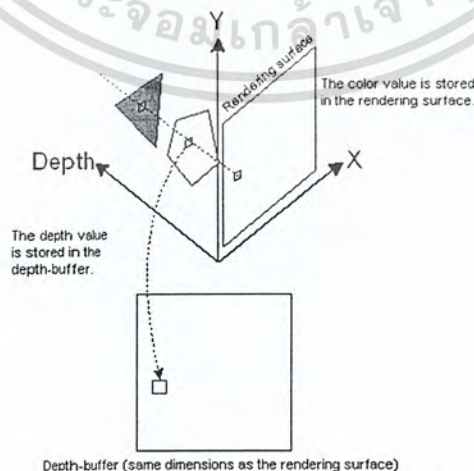
ใน ไคเรกทรีดี เราสามารถกำหนดเทกเจอร์ที่จะใช้ปะบนโพรเจกทีฟได้ด้วยการเรียกเมทอด `IDirect3DDevice::SetTexture`

เนื่องจากคุณสมบัติในการประมวลผลเทกเจอร์ใน ไคเรกทรีดี มีจำนวนมากและเป็นคุณสมบัติที่มีความซับซ้อนอย่างยิ่ง การใช้งานคุณสมบัติเหล่านั้นจะกล่าวถึงในรายละเอียดเมื่อมีการพัฒนาโปรแกรมในขั้นต่อไป

6.16 เดฟบัฟเฟอร์ (Depth Buffers)

เดฟบัฟเฟอร์ หรือที่รู้จักในชื่อของ Z-Buffer และ W-Buffer คือ คุณสมบัติของฮาร์ดแวร์ในการเก็บข้อมูลเกี่ยวกับความลึกของพิกเซล (pixel) ใน ไคเรกทรีดี ซึ่งจะใช้ในการคำนวณรวมกับข้อมูลที่อยู่ในเรนเดอร์เซอร์เฟสที่ใช้เรนเดอร์ในขณะนั้น (render-target surface) ในการตัดสินใจว่าจุดบนวัตถุใดจะเป็นจุดที่สามารถมองเห็นได้ ในกรณีที่วัตถุมีการซ้อนทับกัน

ในตอนเริ่มต้น ข้อมูลที่เก็บอยู่ในเดฟบัฟเฟอร์จะเป็นค่าที่มากที่สุดที่สามารถเป็นไปได้ และค่าของสีที่กำหนดก็เป็นค่าสีที่กำหนดไว้ตั้งแต่เริ่มต้น (by default) หรือค่าสีที่ใช้เป็นฉากหลัง (background) หรือค่าสีของเทกเจอร์ที่ใช้เป็นฉากหลังในจุดนั้นๆ เมื่อมีการส่งให้วาดพอลีกอนด้วยคำสั่งใดๆ ก็ตามทุกๆ พอลีกอนจะต้องนำความมาหาจุดตัดในแกน x, y เพื่อนำค่าสีนั้นมาเป็นค่าสีของจุดตัดนั้นและเก็บค่า z ซึ่งแสดงค่าความลึกของจุดไว้ด้วย ในกรณี ที่ใช้ Z-Buffer จุดสีที่มีค่าความลึกน้อยกว่าจะได้รับการบันทึกค่าของสีและความลึกเอาไว้ ดังภาพ



รูปที่ 6-27 รูปแสดงวิธีการคำนวณเดฟบัฟเฟอร์

อุปกรณ์ที่มีขายทั่วไปในท้องตลาดส่วนใหญ่จะสนับสนุนการใช้งานเคพพ์เฟอร์ที่เป็น Z-Buffer มากกว่าการใช้งานเคพพ์เฟอร์ที่เป็น W-Buffer ซึ่งเคพพ์เฟอร์ทั้งสองชนิดในบางครั้งก็แสดงภาพที่มีความผิดพลาดได้เช่นกัน

การใช้งานเคพพ์เฟอร์ (Using a Depth Buffer)

เคพพ์เฟอร์เป็นพารามิเตอร์ที่ต้องกำหนดค่าในขณะที่สร้างดีไวซ์ขึ้นมา โดยการกำหนดค่าในโครงสร้างข้อมูล D3DPRESENT_PARAMETERS ดังตัวอย่าง

```
D3DPRESENT_PARAMETERS d3dpp;
ZeroMemory( &d3dpp, sizeof(d3dpp) );
d3dpp.Windowed          = TRUE;
d3dpp.SwapEffect         = D3DSWAPEFFECT_COPY_VSYNC;
d3dpp.EnableAutoDepthStencil = TRUE;
d3dpp.AutoDepthStencilFormat = D3DFMT_D16;
```

หลังจากนั้น จึงเรียกเมทอด IDirect3D::CreateDevice ในการสร้างดีไวซ์และสร้างเคพพ์เฟอร์ ดังตัวอย่าง

```
if ( FAILED( g_pD3D->CreateDevice( D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hWnd,
                                D3DCREATE_SOFTWARE_VERTEXPROCESSING, &d3dpp, &d3dDevice ) ) )
    return E_FAIL;
```

การสั่งให้มีการใช้งานเคพพ์เฟอร์ (Enabling Depth Buffering)

หลังจากสร้างเคพพ์เฟอร์แล้วจะต้องสั่งให้มีการใช้งานเคพพ์เฟอร์ด้วยการเรียกเมทอด IDirect3DDevice8::SetRenderState แล้วใส่ค่าอาร์กิวเมนต์ตัวแรกเป็น D3DRS_ZENABLE และใส่ค่าอาร์กิวเมนต์ตัวที่สองเป็น TRUE หรือ D3DZB_TRUE ในกรณีที่ต้องการใช้งาน หรือใส่ค่าเป็น FALSE หรือ D3DZB_FALSE ในกรณีที่ไม่ต้องการใช้งาน (D3DZB_TRUE และ D3DZB_FALSE เป็นสมาชิกในชนิดข้อมูล D3DZBUFFERTYPE)

6.17 เวอร์เท็กซ์บัฟเฟอร์ (Vertex Buffers)

เวอร์เท็กซ์บัฟเฟอร์ คือ เมมโมรีบัฟเฟอร์ที่ใช้เก็บข้อมูลของเวอร์เท็กซ์ และสามารถนำข้อมูลที่มีอยู่ในบัฟเฟอร์นี้ไปเรนเดอร์ได้โดยเรียกเมทอดในกลุ่มของการวาดไพรมิทีฟ (DrawPrimitive) โดยเวอร์เท็กซ์บัฟเฟอร์บรรจุอยู่ใน IDirect3DVertexBuffer8 interface

การสร้างเวอร์เท็กซ์บัฟเฟอร์ (Creating a Vertex Buffer)

เวอร์เท็กซ์บัฟเฟอร์สามารถสร้างขึ้นมาด้วยการเรียกเมทอด `IDirect3DDevice8::CreateVertexBuffer` ซึ่งต้องการพารามิเตอร์ต่างๆ ในการกำหนดคุณสมบัติของเวอร์เท็กซ์บัฟเฟอร์ ดังนี้

- พารามิเตอร์ตัวที่ 1 คือ ขนาดของเวอร์เท็กซ์บัฟเฟอร์ในหน่วยไบต์ซึ่งสามารถคำนวณได้จากการใช้ฟังก์ชัน `sizeof` กับชนิดของข้อมูลที่เก็บข้อมูลเวอร์เท็กซ์
- พารามิเตอร์ตัวที่ 2 คือ ข้อมูลที่ใช้ในการควบคุมการทำงานบางอย่าง เช่น การตัด (clipping) ข้อมูลเวอร์เท็กซ์ที่ไม่ได้อยู่ในวิวพอร์ตหรือไม่ เป็นต้น
- พารามิเตอร์ตัวที่ 3 คือ ค่าที่อธิบายรูปแบบการเก็บข้อมูลในแต่ละเวอร์เท็กซ์ (component of vertex) ซึ่งจะอธิบายในรูปของ FVF (Flexible Vertex Format) โดยการใช้ค่ารวมของ FVF แต่ละชนิด (combination of FVF) หรือใส่ค่าเป็น 0 ในกรณีที่ไม่ใช่ FVF
- พารามิเตอร์ตัวที่ 4 อธิบายบริเวณและคุณสมบัติของเมมโมรี่ที่ใช้เก็บเวอร์เท็กซ์บัฟเฟอร์
- พารามิเตอร์ตัวที่ 5 คือ ตัวแปรที่จะใช้เก็บค่าพอยเตอร์ที่ชี้ไปยังเวอร์เท็กซ์บัฟเฟอร์ที่สร้างขึ้นมา

ตัวอย่างการสร้างเวอร์เท็กซ์บัฟเฟอร์

```
// The custom vertex type
```

```
struct CUSTOMVERTEX {
```

```
    FLOAT x, y, z;
```

```
    FLOAT rhw;
```

```
    DWORD color;
```

```
    FLOAT tu, tv; // The texture coordinates
```

```
};
```

```
#define D3DFVF_CUSTOMVERTEX (D3DFVF_XYZRHW | D3DFVF_DIFFUSE | D3DFVF_TEX1)
```

```
// Create a clipping-capable vertex buffer. Allocate enough memory in the default memory pool to hold three
```

```
// CUSTOMVERTEX structures.
```

```
if ( FAILED( d3dDevice->CreateVertexBuffer(3*sizeof(CUSTOMVERTEX), 0,
```

```
D3DFVF_CUSTOMVERTEX, D3DPOOL_DEFAULT, &g_pVB)))
```

```
    return E_FAIL;
```

การเข้าถึงข้อมูลที่อยู่ในเวอร์เท็กซ์บัฟเฟอร์ (Accessing the Contents of a Vertex Buffer)

เวอร์เท็กซ์บัฟเฟอร์ให้เราสามารถเข้าถึงเมมโมรี่ที่จองไว้สำหรับข้อมูลเวอร์เท็กซ์ได้โดยตรง ด้วยการใช้อเมทอด `IDirect3DVertexBuffer8::Lock` ในการรับค่าพอยเตอร์ที่ชี้ไปยังเวอร์เท็กซ์บัฟเฟอร์ที่ต้องการจะเข้าถึงข้อมูล ซึ่งการเข้าถึงเมมโมรี่ก็เพื่อที่จะใส่ข้อมูลให้กับเวอร์เท็กซ์บัฟเฟอร์นั่นเองเนื่องจากในขั้นตอนการสร้างยังไม่

ได้มีการกำหนดค่าของข้อมูลภายในเวอร์เท็กซ์บัฟเฟอร์เลย หลังจากที่เราได้กำหนดค่าของข้อมูลในแต่ละเวอร์เท็กซ์ในเวอร์เท็กซ์บัฟเฟอร์แล้ว ต้องเรียกเมทอด IDirect3DVertexBuffer8::Unlock จึงจะนำเวอร์เท็กซ์บัฟเฟอร์ไปใช้ในการเรนเดอร์ได้ ดังตัวอย่าง

```
// To fill the vertex buffer, you need to lock the buffer to
// gain access to the vertices. This mechanism is required because
// vertex buffers may be in device memory.

VOID* pVertices;

if ( FAILED( g_pVB->Lock(      0,                          // Fill from the start of the buffer.
                          sizeof(g_Vertices),              // Size of the data to load.
                          (BYTE**)&pVertices,              // Returned vertex data.
                          0 )))                             // Send default flags to the lock.

    return E_FAIL;

memcpy( pVertices, g_Vertices, sizeof(g_Vertices) );
g_pVB->Unlock();
```

เนื่องจากการใช้ภาษา C/C++ ทำให้สามารถเข้าถึงเมมโมรีได้โดยตรง ซึ่งอาจจะเกิดปัญหาขึ้นในกรณีที่เขียนข้อมูลลงเมมโมรีไม่ถูกต้องซึ่งจะทำให้ข้อมูลที่อยู่ในเวอร์เท็กซ์บัฟเฟอร์ผิดพลาด ดังนั้น ในการเข้าถึงเมมโมรีในแต่ละครั้งจะต้องแน่ใจว่าข้อมูลที่จะเขียนอยู่ในลำดับและมีขนาดที่ถูกต้อง ถ้าใช้ FVF ข้อมูลที่เป็นองค์ประกอบของเวอร์เท็กซ์จะมีขนาดดังในตารางต่อไปนี้

Vertex Format Flag	Size
D3DFVF_DIFFUSE	sizeof(DWORD)
D3DFVF_NORMAL	sizeof(float) × 3
D3DFVF_SPECULAR	sizeof(DWORD)
D3DFVF_TEXn	sizeof(float) × n × t
D3DFVF_XYZ	sizeof(float) × 3
D3DFVF_XYZRHW	sizeof(float) × 4

ตารางที่ 6-3 ขนาดของข้อมูลที่เป็นองค์ประกอบของเวอร์เท็กซ์

การเรนเดอร์ข้อมูลที่อยู่ในเวอร์เท็กซ์บัฟเฟอร์ (Rendering from a Vertex Buffer)

มีขั้นตอนในการทำงานดังนี้

1. การกำหนดสตรีมที่ใช้เป็นแหล่งของข้อมูลของเวอร์เท็กซ์ สามารถกำหนดได้โดยเรียกเมทอด `l` ได้
เรกทรีดี `Device8::SetStreamSource` ดังตัวอย่าง

```
lpd3dDevice->SetStreamSource( 0, g_pVB, sizeof(CUSTOMVERTEX) );
```

- พารามิเตอร์ตัวที่ 1 เป็นการบอกว่าเป็นตัวบอกลำดับของสตรีมมีค่าตั้งแต่ 0 ถึงจำนวนของสตรีมที่มากที่สุด - 1
- พารามิเตอร์ตัวที่ 2 คือ เวอร์เท็กซ์บัฟเฟอร์ที่จะใช้งาน
- พารามิเตอร์ตัวที่ 3 คือ ขนาดของเวอร์เท็กซ์แต่ละเวอร์เท็กซ์

2. กำหนดเวอร์เท็กซ์เชดเดอร์ (vertex shader) ที่จะใช้งาน สามารถกำหนดได้ด้วยการเรียกเมทอด `l` ได้
เรกทรีดี `Device::SetVertexShader` ดังตัวอย่าง

```
d3dDevice->SetVertexShader( D3DFVF_CUSTOMVERTEX );
```

3. การวาดไพรมีทีฟ (DrawPrimitive) โดยใช้เวอร์เท็กซ์บัฟเฟอร์ สามารถวาดไพรมีทีฟได้โดยการเรียกเมทอด `l` ได้
เรกทรีดี `Device8::DrawPrimitive` ดังตัวอย่าง

```
d3dDevice->DrawPrimitive( D3DPT_TRIANGLELIST, 0, 1 );
```

- พารามิเตอร์ตัวที่ 1 คือ ชนิดของไพรมีทีฟที่จะวาด
- พารามิเตอร์ตัวที่ 2 คือ ตำแหน่งเริ่มต้นของข้อมูลในเวอร์เท็กซ์บัฟเฟอร์ที่จะนำมาใช้วาด
- พารามิเตอร์ตัวที่ 3 คือ จำนวนของไพรมีทีฟที่จะวาด

6.18 อินเด็กซ์บัฟเฟอร์ (Index Buffers)

อินเด็กซ์บัฟเฟอร์ คือ เมมโมรีบัฟเฟอร์ที่ใช้ในการเก็บข้อมูลอินเด็กซ์ (Index data) ซึ่งใช้ในการกำหนดค่าออฟเซต (offset) ในเวอร์เท็กซ์บัฟเฟอร์และใช้ในการเรนเดอร์ไพรมีทีฟ โดยอินเด็กซ์บัฟเฟอร์บรรจุอยู่ใน `IDirect3DIndexBuffer8` interface

การสร้างอินเด็กซ์บัฟเฟอร์ (Creating an Index Buffer)

สามารถสร้างอินเด็กซ์บัฟเฟอร์ได้ด้วยการเรียกเมทอด `IDirect3DDevice8::CreateIndexBuffer` ซึ่งต้องการพารามิเตอร์ดังต่อไปนี้

พารามิเตอร์ตัวที่ 1 ระบุขนาดของอินเด็กซ์บัฟเฟอร์มีหน่วยเป็นไบต์

พารามิเตอร์ตัวที่ 2 ข้อมูลที่ใช้ในการควบคุมการทำงานบางอย่าง เช่น การตัด (clipping) ข้อมูลอินเด็กซ์ที่ไม่ได้อยู่ในวิวพอร์ตหรือไม่ เป็นต้น

พารามิเตอร์ตัวที่ 3 ใช้ระบุขนาดของชนิดข้อมูลของอินเด็กซ์ ซึ่งมี 2 แบบ คือ แบบ 16 บิต จะต้องใช้ค่า `D3DFMT_INDEX16` และแบบ 32 บิตจะต้องใช้ค่า `D3DFMT_INDEX32` ซึ่งเป็นสมาชิกของชนิดข้อมูล `D3DFORMAT` enumerated type

พารามิเตอร์ตัวที่ 4 อธิบายบริเวณและคุณสมบัติของเมมโมรีที่ใช้เก็บเวอร์เท็กซ์บัฟเฟอร์

พารามิเตอร์ตัวที่ 5 คือ ตัวแปรที่จะใช้เก็บค่าพอยเตอร์ที่ชี้ไปยังอินเด็กซ์บัฟเฟอร์ที่สร้างขึ้นมา

ตัวอย่างการสร้างอินเด็กซ์เท็กซ์บัฟเฟอร์

```
if( FAILED( d3dDevice->CreateIndexBuffer( 16384 *sizeof(WORD), D3DUSAGE_WRITEONLY,
                                         D3DFMT_INDEX16, D3DPOOL_DEFAULT, &g_IB ) ) )
    return E_FAIL;
```

การเข้าถึงข้อมูลที่อยู่ในอินเด็กซ์บัฟเฟอร์ (Accessing the Contents of a Index Buffer)

อินเด็กซ์บัฟเฟอร์ให้เราสามารถเข้าถึงเมมโมรีที่จองไว้สำหรับข้อมูลอินเด็กซ์ได้โดยตรง ด้วยการ使用方法 `IDirect3DIndexBuffer8::Lock` ในการรับค่าพอยเตอร์ที่ชี้ไปยังอินเด็กซ์บัฟเฟอร์ที่ต้องการจะเข้าถึงข้อมูล ซึ่งการเข้าถึงเมมโมรีก็เพื่อที่จะใส่ข้อมูลให้กับอินเด็กซ์บัฟเฟอร์นั่นเองเนื่องจากในขั้นตอนการสร้างยังไม่ได้มีการกำหนดค่าของข้อมูลภายในอินเด็กซ์บัฟเฟอร์เลย หลังจากที่เราได้กำหนดค่าของข้อมูลในแต่ละอินเด็กซ์ในอินเด็กซ์บัฟเฟอร์แล้ว จะต้องเรียกเมทอด `IDirect3DIndexBuffer8::Unlock` จึงจะนำอินเด็กซ์บัฟเฟอร์ไปใช้ในการเรนเดอร์ได้ ดังตัวอย่าง

```
VOID* pIndices;

if( FAILED( IB->Lock( 0, // Fill from start of the buffer.
                    sizeof(g_Indices), // Size of the data to load.
                    (BYTE**)&pIndices, // Returned index data.
                    0 ) ) ) // Send default flags to the lock.
    return E_FAIL;

memcpy( pIndices, g_Indices, sizeof(g_Vertices) );

IB->Unlock();
```

เช่นเดียวกับ ในกรณีของเวอร์เท็กซ์บัฟเฟอร์ การเข้าถึงเมมโมรีโดยตรงจะต้องระวังเรื่องขนาดของข้อมูลให้ดีเนื่องจากการกำหนดขนาดของข้อมูลผิดจะทำให้ภาพที่แสดงผลออกมาผิดพลาดได้ ซึ่งข้อมูลอินเด็กซ์มี 2 ชนิดเท่านั้น คือ ขนาด 16 บิต และขนาด 32 บิต โดยขนาดของอินเด็กซ์ที่สามารถใช้งานได้สามารถตรวจสอบได้ด้วยการเรียกเมทอด IDirect3DDevice::GetDeviceCaps ซึ่งจะให้รายละเอียดความสามารถของฮาร์ดแวร์แก่เรา ในโครงสร้างข้อมูล D3DCAPS8 ซึ่งถ้าสมาชิกที่ชื่อ MaxVertexIndex มากกว่าค่า 0x0000FFFF ก็จะใช้ข้อมูลอินเด็กซ์ขนาด 32 บิตได้

การเรนเดอร์จากอินเด็กซ์บัฟเฟอร์ (Rendering from an Index Buffer)

มีขั้นตอนในการทำงานดังนี้

1. การกำหนดสตรีมที่ใช้เป็นแหล่งของข้อมูลของเวอร์เท็กซ์ สามารถกำหนดได้โดยเรียกเมทอด

IDirect3DDevice8::SetStreamSource ดังตัวอย่าง

```
d3dDevice->SetStreamSource( 0, g_pVB, sizeof(CUSTOMVERTEX) );
```

พารามิเตอร์ตัวที่ 1 เป็นการบอกว่าเป็นตัวบอกลำดับของสตรีมมีค่าตั้งแต่ 0 ถึงจำนวนของสตรีมที่มากที่สุด - 1

พารามิเตอร์ตัวที่ 2 คือ เวอร์เท็กซ์บัฟเฟอร์ที่จะใช้งาน

พารามิเตอร์ตัวที่ 3 คือ ขนาดของเวอร์เท็กซ์แต่ละเวอร์เท็กซ์

2. กำหนดอินเด็กซ์บัฟเฟอร์ที่จะใช้งานด้วยการเรียกเมทอด IDirect3DDevice8::SetIndices ดังตัวอย่าง

```
d3dDevice->SetIndices(IB, 0);
```

พารามิเตอร์ตัวที่ 1 คือ อินเด็กซ์บัฟเฟอร์ที่จะใช้งาน

พารามิเตอร์ตัวที่ 2 คือ ตำแหน่งของข้อมูลอินเด็กซ์ที่เริ่มใช้งาน

3. การวาดไพรมิทีฟ (DrawPrimitive) โดยใช้อินเด็กซ์บัฟเฟอร์ สามารถวาดไพรมิทีฟได้โดยการเรียกเมทอด

IDirect3D::DrawIndexPrimitive ดังตัวอย่าง

```
d3dDevice->DrawIndexedPrimitive(D3DPT_TRIANGLELIST, 0, dwVertices, 0, dwIndices / 3);
```

พารามิเตอร์ตัวที่ 1 คือ ชนิดของไพรมิทีฟที่จะวาด

พารามิเตอร์ตัวที่ 2 คือ ค่าต่ำสุดของอินเด็กซ์ที่ใช้ในการเรนเดอร์

- พารามิเตอร์ตัวที่ 3 คือ จำนวนของอินเด็กซ์ทั้งหมดที่จะใช้ในการเรนเดอร์ โดยเริ่มนับจากค่า BaseVertexIndex (จากเมทอด SetIndices) + MinIndex (พารามิเตอร์ตัวที่ 2)
- พารามิเตอร์ตัวที่ 4 คือ ตำแหน่งของอินเด็กซ์ฟเฟอร์
- พารามิเตอร์ตัวที่ 5 คือ จำนวนของโพรมิทฟที่จะเรนเดอร์

6.19 D3DX Library

นอกจากไลบรารีในส่วนของโคเรกทรีตี แล้วไมโครซอฟท์ยังได้พัฒนาไลบรารีเพื่อช่วยในการทำงานเบื้องต้นบางอย่างให้แก่ นักพัฒนาอีกด้วย ซึ่งส่วนที่มีการนำมาใช้ในโปรเจกต์นี้คือ ฟังก์ชันที่ทำหน้าที่ในการคำนวณค่าทางคณิตศาสตร์ต่างๆ ซึ่งประกอบด้วยกลุ่มของฟังก์ชัน ดังนี้

ระนาบ (Plane)

ควอเตอร์เนียน (Quaternion)

เวกเตอร์ 2 มิติ (2-D Vector)

เวกเตอร์ 3 มิติ (3-D Vector)

เวกเตอร์ 4 มิติ (4-D Vector)



บทที่ 7

ไคเร็กอินพุต (DirectInput)

7.1 แนะนำไคเร็กอินพุต

ไคเร็กอินพุต คือ แอปพลิเคชันโปรแกรมมิ่งอินเทอร์เฟซ (API) สำหรับการใช้งานอุปกรณ์รับข้อมูล เช่น เมาส์, คีย์บอร์ด, จอยสติค และอุปกรณ์ควบคุมเกมอื่นๆ และอุปกรณ์ที่รับและส่งข้อมูลได้ เช่น อุปกรณ์ประเภทแรงส่งกลับ (force-feedback device)

7.1.1 ประสิทธิภาพของไคเร็กอินพุต (The Power of DirectInput)

นอกจากจะสนับสนุนการใช้งานบางอย่างที่ไม่สามารถใช้งานโดยผ่าน Win32 API ได้แล้ว ไคเร็กอินพุตยังเข้าถึงข้อมูลได้เร็วกว่า เนื่องจากการติดต่อโดยตรงกับฮาร์ดแวร์ไคเร็กอินพุตซึ่งไม่ต้องเสียเวลากับการทำให้เป็นเมสเสจด้วยระบบปฏิบัติการวินโดวส์

ไคเร็กอินพุตให้โปรแกรมสามารถรับข้อมูลจากอุปกรณ์รับข้อมูลแม้ว่ากำลังทำงานอยู่ในแบ็กกราวนด์ นอกจากนี้ ยังสนับสนุนการใช้งานอุปกรณ์ทุกชนิดรวมทั้งอุปกรณ์ประเภทฟอร์สฟีดแบ็กด้วย

โปรแกรมสามารถรับข้อมูลจากอุปกรณ์รับข้อมูลได้โดยไม่มีความจำเป็นจะต้องรู้ว่ากำลังติดต่อกับอุปกรณ์ชนิดไหนที่สร้างข้อมูลนั้นขึ้นมา โดยใช้วิธีการแมปการกระทำ (action mapping)

ความสามารถและประสิทธิภาพที่มีมากกว่า Win32API ทำให้ไคเร็กอินพุตมีความสำคัญอย่างยิ่งในการพัฒนาโปรแกรมประเภทเกม, การจำลองสถานการณ์ และโปรแกรมที่อินเทอร์เอกทีฟกับผู้ใช้ในรูปแบบเรียลไทม์

ถ้าแอปพลิเคชันใช้คีย์บอร์ดในการรับตัวอักษรและเมาส์ในการเคลื่อนที่ไปยังบริเวณต่างๆ การใช้ไคเร็กอินพุตไม่มีประโยชน์ใดๆ เพิ่มขึ้น

7.1.2 เริ่มต้นกับไคเร็กอินพุต (Getting Started with DirectInput)

ในการที่จะเข้าใจไคเร็กอินพุตจะต้องเข้าใจในสิ่งเหล่านี้ก่อน

- วัตถุไคเร็กอินพุต (DirectInput object)
- อุปกรณ์ (Device : keyboard, mouse, joystick)
- วัตถุ DirectInputDevice
- วัตถุ Device

ต่อไปนี้เป็นขั้นตอนในการใช้งานไคเร็กอินพุต

1. สร้างวัตถุไคเร็กอินพุต

เนื่องจากจะต้องใช้เมทอดของออปเจกต์นี้ในการตรวจสอบและสร้าง DirectInput device objects

2. ตรวจสอบ Device

ถ้าต้องการจะใช้งานเพียงแค่เมาส์และคีย์บอร์ดซึ่งมีอยู่ในเครื่องคอมพิวเตอร์ทุกเครื่องก็ไม่จำเป็นต้องตรวจสอบ แต่ในกรณีที่ต้องการใช้อุปกรณ์อื่นๆ ที่อาจจะมิอยู่ในเครื่องคอมพิวเตอร์ก็ต้องการตรวจสอบก่อน ถ้า DirectInput พบอุปกรณ์ที่ตรงกับที่เรากำหนดมันจะให้เราสามารถตรวจสอบคุณสมบัติของอุปกรณ์ก่อนได้ และจะให้ค่าเฉพาะสำหรับอุปกรณ์นั้นไว้เพื่อใช้ในการสร้าง DirectInput device object เพียงแทนอุปกรณ์นั้นๆ

3 สร้าง DirectInputDevice object สำหรับแต่ละอุปกรณ์ที่ต้องการ

ที่ค่าเฉพาะสำหรับอุปกรณ์ที่ได้รับมาระหว่างการตรวจสอบจะถูกใช้ในการสร้าง แต่ถ้าใช้คีย์บอร์ดและเมาส์สามารถใช้ค่า GUID มาตรฐานแทนได้

4. เซตอัพอุปกรณ์ (Set up the device)

สำหรับแต่ละอุปกรณ์ ในขั้นตอนแรกจะต้องกำหนดระดับในการใช้งานอุปกรณ์ (cooperative level) ซึ่งเป็นตัวกำหนดวิธีการที่อุปกรณ์จะถูกใช้งานร่วมกับแอปพลิเคชันอื่นๆ อย่างไร นอกจากนั้นจะต้องกำหนดรูปแบบของข้อมูลที่จะใช้ในการจำแนก device object เช่น button, axe ใน data packet หรือถ้าต้องการในรูปของ buffer data ก็ต้องกำหนดขนาดของบัฟเฟอร์ และยังสามารถหาข้อมูลเกี่ยวกับอุปกรณ์และกำหนดคุณสมบัติต่างๆ เช่น ระยะของค่าที่ส่งมาจากจอยสติค เป็นต้น

5. การได้รับอุปกรณ์ (Acquiring device) ในขั้นตอนนี้เป็นการบอก DirectInput พร้อมที่จะรับข้อมูลจากอุปกรณ์แล้ว

6. การรับข้อมูล (Retrieve data)

เป็นการรับค่าสถานะของอุปกรณ์ในขณะนั้นหรือรับบันทึกเหตุการณ์ที่เกิดขึ้นในช่วงเวลาหลังจากการรับค่าครั้งล่าสุด หรือ จะให้ DirectInput เตือนขึ้นมาเองก็ได้

7. การประมวลผลข้อมูล (Act on the data)

แอปพลิเคชันนำข้อมูลไปประมวลผล

8. ปิดไดเรกอินพุต (Close DirectInput)

แต่นี้ก็ไม่ใช่วิธีการเดียวในการใช้งาน DirectInput เรายังสามารถใช้ action mapping ได้อีกเช่นกัน ด้วยการกำหนด action map ให้กับอุปกรณ์ ก็สามารถนำ device object ไปใช้สำหรับแต่ละค่าในแอปพลิเคชันได้

7.2 ทำความเข้าใจไดเรกอินพุต (Understanding DirectInput)

DirectInput Object

เฉพาะอุปกรณ์รับข้อมูลที่ได้รับข้อมูลได้เพียงอย่างเดียวใน DirectInput สร้างขึ้นโดยประกอบด้วย DirectInput object ซึ่งอยู่ใน IDirectInput8 COM Interface และ DirectInputDevice object สำหรับแต่ละอุปกรณ์ที่จะให้ข้อมูล ซึ่งทุกตัวจะต้องมี device object ซึ่งแทน key, button เป็นต้น (device object instance)

DirectInputDevice object อยู่ใน IDirectInputDevice8 interface ซึ่งจำนวนและชนิดของอุปกรณ์สามารถตรวจสอบได้โดยเรียกเมทอด IDirectInputDevice8::EnumObjects ค่าที่อธิบายอุปกรณ์แต่ละตัวจะเก็บอยู่ใน DIDEVICEOBJECTINSTANCE

Force-feedback effect อยู่ใน IDirectInputEffect interface ซึ่งมีเมทอดที่ใช้ในการสร้าง, เปลี่ยนแปลง, เริ่ม และหยุด การทำเอฟเฟ็ค

7.3 การติดต่อกับวินโดว์ (Interaction with Windows)

เนื่องจาก DirectInput เป็นการทำงานโดยตรงกับ device driver ดังนั้นในบางครั้งมันอาจจะไม่สนใจ mouse และ keyboard message และ ไม่สนใจค่ากำหนดของ mouse และ keyboard ที่กำหนดไว้ใน control panel ด้วย

DirectInput ไม่มีการเห็นค่ากำหนดในการเข้าตัวอักษรสำหรับคีย์บอร์ด เมื่อใช้ buffer data DirectInput จะแปลสัญญาณเป็นการกดหรือการปล่อย 1 เหตุการณ์ไม่มีการซ้ำ เมื่อใช้ immediate data DirectInput จะสนใจเฉพาะค่าสถานะปัจจุบันของคีย์เท่านั้น นั่นคือ เหตุการณ์ของคีย์บอร์ดจะ ไม่มีการประมวลผลโดยผ่านวินโดว์

DirectInput ไม่ได้ทำการแปลงหรือเปลี่ยนค่าของตัวอักษรใดๆ ทั้งสิ้น เช่น ปุ่ม SHIFT จะได้รับการปฏิบัติเช่นเดียวกับปุ่มอื่นๆ ไม่ใช้การเปลี่ยนค่าของข้อมูลที่ได้รับจากปุ่มอื่นๆ นอกจากนั้น ค่าที่ส่งกลับมามีเหมือนกันไม่ว่าจะมีการกำหนดคีย์ภาษาเป็นเช่นไร

ใน Windows 2000 การรับค่าของคีย์บอร์ดในแบบ exclusive จะป้องกัน แอปพลิเคชัน อื่นที่จะทำงานขึ้นมาภายหลังเมื่อได้รับข้อมูลจากคีย์บอร์ดได้

เนื่องจาก directinput ทำงานโดยตรงกับ mouse driver นั่นคือ ข้ามระบบย่อยของวินโดว์ที่ทำหน้าที่ในการแปลข้อมูลของเมาส์ให้กับแอปพลิเคชัน ดังนั้น ถ้าแอปพลิเคชันยังคงต้องการใช้ตัวชี้ของวินโดว์ในการใช้งานควรใช้ standard windows mouse messages และ Microsoft ® Win32® function

เมื่อใช้ระบบเมาส์ในแบบ exclusive mode DirectInput จะเข้าไปบังระบบ mouse message และ ทำให้วินโดว์ไม่สามารถที่จะแสดงตัวชี้มาตรฐานได้

DirectInput ไม่สนใจค่าที่กำหนดใน control panel เช่น ค่าความเร่ง และการแปลงค่าปุ่ม อย่างไรก็ตาม DirectInput จะยังคงเห็นค่าที่กำหนดไว้ใน Driver เอง เช่น ถ้ามีเมาส์แบบ 3 ปุ่ม และ มีการใช้งาน driver utility software เพื่อทำให้ปุ่มกลางกลายเป็น double click DirectInput จะบอกว่ามีการกดปุ่มกลางเป็นการคลิกปุ่มเมาส์ 2 ครั้ง

7.4 การใช้งาน DirectInput

1. CreateDirectInput

ขั้นตอนแรกในการใช้งาน DirectInput คือการรับค่าของ IDirectInput 8 interface ซึ่งสามารถทำได้โดยเรียกฟังก์ชัน DirectInput8Create

ในการใช้งานควรสร้าง DirectInput object เพียงตัวเดียวและปล่อยค่าเมื่อจบการทำงาน

2. DirectInput Device Enumeration

DirectInput สามารถถามระบบถึงอุปกรณ์อินพุตที่มีอยู่ทั้งหมดได้ และสามารถตรวจสอบได้ว่า อุปกรณ์เชื่อมต่ออยู่หรือไม่ และส่งค่าข้อมูลของอุปกรณ์เหล่านั้นกลับคืนมา กระบวนการนี้เรียกว่า การ ระบุ (enumeration)

ถ้าแอปพลิเคชันต้องการใช้งานเฉพาะอุปกรณ์มาตรฐาน เช่น เมาส์ หรือ คีย์บอร์ด ก็ไม่จำเป็นที่จะต้องทำการระบุอุปกรณ์ เนื่องจากสามารถใช้ค่าตัวแปรที่กำหนดไว้แล้ว (predefined global variable) เมื่อ เรียกเมทอด `Idirectinput8::CreateDevice`

อุปกรณ์ชนิดอื่นๆ และระบบที่มีคีย์บอร์ดหรือเมาส์หลายตัว จะต้องเรียก

`IdirectInput8::EnumDevices` หรือ `IdirectInput8::EnumDevicesBySemantics` ในการระบุอุปกรณ์ที่มีอยู่ และหาอุปกรณ์ตัวใดตัวหนึ่งที่เหมาะสมที่สุด หรือจะใช้ข้อมูลที่ได้มาเป็นตัวเลือกก็ได้ ตัวอย่าง code

```
GUID KeyboardGUID = GUID_SysKeyboard;
```

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

พารามิเตอร์ตัวแรก คือ ชนิดของอุปกรณ์ที่จะทำการระบุ สามารถใส่ค่าเป็น NULL ได้ถ้าต้องการที่จะ ระบุอุปกรณ์ทุกชนิด หรือจะกำหนดค่าเจาะจงโดยชนิดของอุปกรณ์ก็ได้ โดยใช้ค่า `DIDEVTYPE_*` ตัวใดตัวหนึ่งซึ่งค่าที่เป็นองค์ประกอบได้กำหนดไว้แล้วใน `DIDeviceInstance`

พารามิเตอร์ตัวที่สอง คือ ค่าพอยเตอร์ที่ชี้ไปยัง callback function ที่จะถูกเรียกเมื่อมีการระบุอุปกรณ์ แต่ละตัว ฟังก์ชันตัวนี้สามารถตั้งชื่ออะไรก็ได้

พารามิเตอร์ตัวที่สาม คือ ค่า 32 บิต ซึ่งต้องการส่งให้กับ callback function เช่น ส่งค่าพอยเตอร์ที่ชี้ไปยังตัวแปรของ GUID เพื่อที่จะกำหนดค่า keyboard instance identifier ใน กับตัวแปรเหล่านี้

พารามิเตอร์ตัวที่สี่ คือ ค่าสถานะที่บอกว่าจะให้ระบุอุปกรณ์ทั้งหมดหรือเฉพาะที่เชื่อมต่ออยู่เท่านั้น (`DIEDFL_ALLDEVICES` หรือ `DIEDFL_ATTACHEDONLY`)

ถ้าต้องการที่จะใช้อุปกรณ์หลายชนิด callback function สามารถในการกำหนดค่าเริ่มต้นให้กับ อุปกรณ์แต่ละชนิดได้ และจะได้รับค่า GUID ของอุปกรณ์จาก callback นอกจากนั้นยังสามารถประมวลผลอย่างอื่นได้อีกเช่น การค้นหาชนิดของอุปกรณ์บางชนิด หรือ การเพิ่มชื่อของอุปกรณ์ที่รู้จักเข้าในบัญชีรายชื่อ

ตัวอย่าง

```

BOOL hasEnhanced;

BOOL CALLBACK DEnumKbdCallback(LPCDDIDEVICEINSTANCE lpddi, LPVOID pvRef)
{
    *(GUID*) pvRef = lpddi->guidInstance;

    if (GET_DIDEVICE_SUBTYPE(lpddi->dwDevType) == DIDEVTYPEKEYBOARD_PCENH)
    {
        hasEnhanced = TRUE;

        return DIENUM_STOP;
    }

    return DIENUM_CONTINUE;
} // End of callback

```

7.5 ไดรฟ์อินพุตดีไวซ์ (DirectInput Device)

การติดตั้งดีไวซ์ (Device Setup)

แอปพลิเคชันจะต้องรับค่า DirectInput device object สำหรับแต่ละอุปกรณ์ที่ต้องการอินพุตจากมัน นอกจากนั้น จะต้องเตรียมอุปกรณ์แต่ละตัวให้พร้อมที่จะใช้งานอย่างที่ต้องการ เช่น การกำหนดค่ารูปแบบข้อมูล รวมถึงการหาข้อมูลของอุปกรณ์แต่ละตัวและการเปลี่ยนแปลงคุณสมบัติของอุปกรณ์นั้นๆ

ขั้นตอนต่อไปนี้เป็นขั้นตอนในกระบวนการติดตั้ง ซึ่งจะต้องทำตามขั้นตอนเสมอ แต่สามารถเพื่อเติมขั้นตอนเข้าไปได้ถ้าต้องการข้อมูลจากอุปกรณ์เพิ่มเติม

1. สร้าง DirectInput device (จำเป็น)
2. รับค่าความสามารถของอุปกรณ์ (device capability) (ตัวเลือก)
3. ระบุเป็นอักษร, ปุ่ม หรือแกน ของอุปกรณ์ (ตัวเลือก)
4. กำหนดค่าระดับการทำงานร่วมกัน (cooperative level) (แนะนำว่าควรทำ)
5. กำหนดค่ารูปแบบของข้อมูล (จำเป็น)
6. กำหนดค่าคุณสมบัติของอุปกรณ์ (ถ้าใช้บัฟเฟอร์จะต้องกำหนดขนาดของมัน)
7. เมื่อใดที่พร้อมที่จะรับข้อมูล ให้รับอุปกรณ์มา (acquire device) (จำเป็น)

การสร้างอุปกรณ์ไดเรกอินพุต (Creating a DirectInput Device)

เพื่อที่จะรับข้อมูลจากอุปกรณ์ ชิ้นแรกจะต้องสร้างออปเจกต์ (object) ที่แทนอุปกรณ์นั้นก่อน
 IDirectInput8::CreateDevice เป็นเมทอดที่ใช้ในการรับค่าพอยเตอร์ที่ชี้ไปยังอินเทอร์เฟซ (interface)
 IDirectInputDevice8 เมทอดของอินเทอร์เฟซนี้ยังใช้ในการควบคุมและรับข้อมูลจากอุปกรณ์อีกด้วย
 ตัวอย่าง

```
LPDIRECTINPUTDEVICE8 lpdiKeyboard;  

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

พารามิเตอร์ตัวแรก คือ อินสแตนซ์ (instance) ของ GUID ที่ใช้จำแนกอินสแตนซ์ของอุปกรณ์
 สำหรับอินเทอร์เฟซที่จะสร้าง ใน DirectInput มี GUID 2 ตัวที่ได้รับการ
 กำหนดค่าไว้ล่วงหน้าแล้ว คือ GUID_SysMouse และ GUID_SysKeyboard
 ซึ่งแทนเมาส์และคีย์บอร์ดของระบบ ค่า GUID 2 ตัวนี้สามารถส่งผ่านให้กับเม
 ทอด CreateDevice ได้เลย สำหรับอุปกรณ์ชนิดอื่นๆ นอกจากเมาส์และ
 คีย์บอร์ดแล้ว ค่าอินสแตนซ์ GUID จะได้มาจากเมทอด EnumDevices หรือ
 EnumDevicesBySemantics

ความสามารถของอุปกรณ์

การที่จะรับข้อมูลจากอุปกรณ์จำเป็นต้องรู้ความสามารถบางอย่างที่อุปกรณ์มี เช่น เมาส์เชื่อม
 ต่ออยู่กับเครื่องคอมพิวเตอร์อยู่หรือไม่ เป็นต้น ค่าเหล่านี้หาได้จากการเรียกเมทอด IDirectInputDevice8::

GetCapabilities ซึ่งจะส่งข้อมูล โครงสร้าง DIDEVCAPS กลับคืนมา

ตัวอย่าง

```
DIDEVCAPS DIMouseCaps;  

HRESULT hr;  

BOOLEAN WheelAvailable;
```

```
DIMouseCaps.dwSize = sizeof(DIDEVCAPS);  

hr = lpdiMouse->GetCapabilities(&DIMouseCaps);  

WheelAvailable = ((DIMouseCaps.dwFlags & DIDC_ATTACHED)  

  && (DIMouseCaps.dwAxes > 2));
```

นอกจากนั้นยังมีอีกวิธีการหนึ่งในการตรวจสอบปุ่มหรือแกน โดยการเรียก
 IDirectInputDevice8::GetObjectInfo จากออปเจกต์นั้น ถ้าเมทอดส่งค่ากลับคืนเป็น
 DIERR_OBJECTNOTFOUND แสดงว่าไม่มีออปเจกต์นั้นอยู่
 ตัวอย่าง การตรวจสอบว่ามีแกน z อยู่หรือไม่

```
DIDeviceObjectInstance didoi;
```

```
didoi.dwSize = sizeof(DIDeviceObjectInstance);
```

```
hr = lpdiMouse->GetObjectInfo(&didoi, DIMOFS_Z, DIPH_BYOFFSET);
```

```
WheelAvailable = SUCCEEDED(hr);
```

7.6 ระดับการทำงานร่วมกัน (Cooperative Levels)

ระดับการทำงานร่วมกันของอุปกรณ์ใช้ในการกำหนดว่าจะมีการใช้งานอินพุตร่วมกันได้หรือไม่
 ระหว่างแอปพลิเคชันและกับระบบปฏิบัติการวินโดวส์ ระดับการทำงานร่วมกันสามารถกำหนดได้โดยการ
 เรียกเมทอด IDirectInputDevice8::SetCooperativeLevel

ตัวอย่าง

```
/* hwnd is the top-level window handle. */
```

```
lpdiDevice->SetCooperativeLevel(hwnd, DISCL_NONEXCLUSIVE | DISCL_FOREGROUND)
```

ค่าพารามิเตอร์ตัวแรก คือ ค่าแฮนเดิลของวินโดวส์ที่อยู่บนสุด (top-level window) ซึ่งจะได้รับการติด
 ต่อกับอุปกรณ์

ค่าพารามิเตอร์ตัวที่สอง คือ ค่าสถานะที่จะกำหนดให้กับระดับการใช้งานร่วมกัน

แม้ว่า DirectInput จะได้กำหนดค่าโดยปริยายให้กับระดับการใช้งานร่วมกันของอุปกรณ์ไว้แล้ว
 ก็จะต้องกำหนดค่านี้ใหม่เนื่องจากเมทอดนี้เป็นเมทอดเดียวที่จะให้ค่าแฮนเดิลของวินโดวส์แก่ไคเรกอินพุต
 เนื่องจากถ้าไคเรกอินพุตไม่มีค่าแฮนเดิลจะไม่สามารถตอบสนองต่อเหตุการณ์ที่มีวินโดวส์เมซเซจเข้ามา
 เกี่ยวข้องได้

ค่าแสดงสถานะ	หมายเหตุ
DISCL_NONEXCLUSIVE DISCL_BACKGROUND	ค่ากำหนดโดยปริยาย
DISCL_NONEXCLUSIVE DISCL_FOREGROUND	
DISCL_EXCLUSIVE DISCL_FOREGROUND	
DISCL_EXCLUSIVE DISCL_BACKGROUND	ไม่สามารถใช้กับคีย์บอร์ดและเมาส์ได้

ตารางที่ 7-1 แสดงค่าแสดงสถานะที่เป็นไปได้

ในกรณีของคีย์บอร์ด สามารถเพิ่ม DISCL_NOWINKEY ร่วมกับ DISCL_NONEXCLUSIVE ได้ โดยค่าแสดงสถานะนี้จะทำให้เป็นคีย์โลโก้วินโดวส์ใช้การไม่ได้ ใน exclusive mode เป็นคีย์โลโก้วินโดวส์จะใช้การไม่ได้เสมอ

ค่าแสดงสถานะของระดับการใช้งานร่วมกัน

Foreground และ Background

Foreground คือ อุปกรณ์อินพุตจะใช้งานได้ก็ต่อเมื่อแอปพลิเคชันทำงานอยู่ในเบื้องหน้า หรือเรียกอีกอย่างหนึ่งว่าได้รับ โฟกัส ถ้าแอปพลิเคชันไปทำงานเบื้องหลังเมื่อใดอุปกรณ์อินพุตจะไม่สามารถใช้งานจากแอปพลิเคชันได้

Background คือ แอปพลิเคชันสามารถใช้อุปกรณ์อินพุตได้ตลอดเวลาไม่ว่าจะอยู่เบื้องหน้าหรือเบื้องหลัง

Exclusive และ Nonexclusive

ในความเป็นจริงแล้วแม้ว่าแอปพลิเคชันจะใช้งานอุปกรณ์แบบแต่เพียงผู้เดียวไม่ได้หมายความว่าไม่มีแอปพลิเคชันอื่นที่สามารถรับอินพุตได้ นั่นคือ ไม่มีแอปพลิเคชันใดที่สามารถใช้อุปกรณ์อินพุตแต่เพียงผู้เดียวได้โดยแท้จริง ตัวอย่างเช่น การกดแป้นพิมพ์ ALT+TAB

7.7 การระบุอ็อบเจกต์อุปกรณ์ (Device Object Enumeration)

เป็นสิ่งจำเป็นเนื่องแอปพลิเคชันจำเป็นต้องรู้ว่ามียูนหรือคันทริกใดที่มีอยู่บนอุปกรณ์ การระบุอ็อบเจกต์อุปกรณ์มีวิธีการคล้ายคลึงกับการระบุอุปกรณ์ โดยสามารถใช้เมทอด IDirectInputDevice8::EnumObjects และ IDirectInputDevice8::GetCapabilities แทน

ตัวอย่าง

```
char szName[MAX_PATH];

BOOL CALLBACK DIEnumDeviceObjectsCallback(
    LPCDIDeviceObjectInstance lpddoi, LPVOID pvRef)
{
    lstrcpy(szName, lpddoi->tszName);

    // Now add szName to a list or array.

    return DIENUM_CONTINUE;
}
```

7.8 รูปแบบข้อมูลของอุปกรณ์ (Device Data Formats)

ถ้าไม่มีการใช้ Action Mapping การกำหนดรูปแบบข้อมูลสำหรับอุปกรณ์เป็นขั้นตอนที่สำคัญในการได้มาและเริ่มต้นการใช้งานอุปกรณ์ ในไคเรกอินพุตมีเมทอดจำนวนมากที่ใช้เกี่ยวข้องกับการกำหนดรูปแบบข้อมูล

เมทอด IDirectInputDevice8::SetDataFormat กำหนดว่าออปเจ็กต์อุปกรณ์ใดที่จะใช้และวิธีการที่ข้อมูลจะเรียงลำดับ โครงสร้างข้อมูล DIDATAFORMAT และ DIOBJECTDATAFORMAT แสดงวิธีการกำหนดค่าสำหรับอุปกรณ์ที่มีไม่มาตรฐาน ส่วนอุปกรณ์ที่เป็นมาตรฐาน เช่น จอยสติค, เมาส์ และคีย์บอร์ดได้มีการกำหนดค่าไว้แล้วดังนี้ c_dfDIJoystick, c_dfDIJoystick2, c_dfDIKeyboard, c_dfDIMouse และ c_dfDIMouse2 ซึ่งสามารถส่งให้กับเมทอด SetDataFormat ได้ทันที

ตัวอย่าง

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

7.9 ข้อมูลจากอุปกรณ์ไคเรกอินพุต

ข้อมูลบัฟเฟอร์และอิมมีเดียท (Buffered and Immediate data)

ในไคเรกอินพุตข้อมูลมี 2 แบบ คือ บัฟเฟอร์ และ อิมมีเดียท โดยบัฟเฟอร์จะบันทึกเหตุการณ์ที่เกิดขึ้นเก็บไว้จนกว่าแอปพลิเคชันจะมารับค่าเหล่านั้นไป (อาจเกิดบัฟเฟอร์โอเวอร์โฟลว์ (Buffer Overflow) ถ้ากำหนดขนาดบัฟเฟอร์เล็กเกินไปทำให้สูญเสียข้อมูลใหม่ๆ ไป) และอิมมีเดียทที่จะรับเฉพาะข้อมูลสถานะปัจจุบันเท่านั้น

7.10 การสำรวจข้อมูล (Polling)

การสำรวจข้อมูลจากอุปกรณ์อินพุตโดยปกติจะต้องทำให้มีเซชเชจลูป (message loop) เนื่องจากจอยสติคและอุปกรณ์เกมส์อื่นๆ ไม่ได้สร้างฮาร์ดแวร์อินเทอร์รัพ (Hardware Interrupt) หรือส่งค่าใดๆ กลับคืนมาให้ จากกว่าจะมีการเรียกเมทอด `IdirectInputDevice8::Poll` (`Poll` ในที่นี้ไม่ได้หมายความว่าไปรับข้อมูลมาแต่เป็นการทำให้มีข้อมูลไว้ใช้งาน)



บทที่ 8

ไดเรกต์เอ็ดไอโอ (DirectX Audio)

8.1 ไดเรกต์ซาว์ (DirectSound)

ไดเรกต์ซาว์ออปเจกต์เป็นตัวแทนของอุปกรณ์และถูกใช้ในการจัดการอุปกรณ์และการสร้างเซวบัฟเฟอร์ (Sound Buffer)

ไดเรกต์ซาว์บัฟเฟอร์ออปเจกต์เป็นตัวแทนของบัฟเฟอร์ที่เก็บข้อมูลเสียงเอาไว้ บัฟเฟอร์สามารถใช้ในการเล่น, หยุด หรือรักษาดำแหน่ง (pause) ได้ นอกจากนั้นยังกำหนดค่าความถี่หรือรูปแบบได้

เซวบัฟเฟอร์ปฐมภูมิ (primary sound buffer) ทำหน้าที่เก็บข้อมูลเสียงที่ผู้ฟังจะได้ยิน ส่วนเซวบัฟเฟอร์ทุติยภูมิ (secondary sound buffer) ซึ่งบัฟเฟอร์ชนิดนี้สามารถมีได้หลายตัวโดยแต่ละตัวจะเก็บข้อมูลเสียงหรือสตรีมของเสียง 1 ชุดเอาไว้ ไดเรกต์ซาว์จะสร้างบัฟเฟอร์ปฐมภูมิโดยอัตโนมัติส่วนบัฟเฟอร์ทุติยภูมิจะสร้างขึ้นโดยแอปพลิเคชัน เมื่อมีการเล่นข้อมูลเสียงในบัฟเฟอร์ทุติยภูมิไดเรกต์ซาว์จะผสมข้อมูลเสียงของแต่ละบัฟเฟอร์ทุติยภูมิเข้าด้วยกันแล้วเก็บลงในบัฟเฟอร์ปฐมภูมิแล้วส่งไปยังอุปกรณ์เอาต์พุตเจ็อนไอที่เป็นตัวกำหนดจำนวนของบัฟเฟอร์ทุติยภูมิขึ้นอยู่กับเวลาในการประมวลผลข้อมูลเท่านั้นเนื่องจากไดเรกต์ซาว์ไม่มีข้อกำหนดของจำนวนบัฟเฟอร์

เสียงที่มีความยาวไม่มากสามารถที่จะโหลดเข้าสู่บัฟเฟอร์ได้ทั้งหมดและสามารถที่จะเล่นเสียงนั้นได้ด้วยการเรียกฟังก์ชันเพียงฟังก์ชันเดียว แต่ถ้าข้อมูลมีความยาวมากเกินไปจะต้องบัฟเฟอร์จะต้องรับข้อมูลในรูปแบบของสตรีม โดยแอปพลิเคชันสามารถรู้ได้ว่าจะต้องส่งข้อมูลเป็นสตรีมข้อมูลเมื่อใดโดยการสำรวจตำแหน่งของตัวชี้ที่แสดงตำแหน่งการเล่นหรือรับการแจ้งเมื่อตัวชี้ตำแหน่งมาถึงตำแหน่งที่กำหนด

รูปแบบของข้อมูลในเซวบัฟเฟอร์ทุติยภูมิจะต้องกำหนดโดยแอปพลิเคชันเนื่องจากไดเรกต์ซาว์ไม่มีเมทอดที่ช่วยในการแปลงข้อมูลจากไฟล์ wave

8.2 อุปกรณ์ในไดเรกต์ซาว์ (DirectSound Devices)

ในการใช้งานไดเรกต์ซาว์ขั้นตอนแรกจะต้องสร้างไดเรกต์ซาว์ออปเจกต์ซึ่งในเป็นตัวแทนของอุปกรณ์เสียง เช่น การ์ดเสียง

8.3 การระบุอุปกรณ์ในไดเรกต์ซาว์ (Enumeration of Sound Devices)

อันที่จริงแล้วแอปพลิเคชันไม่จำเป็นต้องระบุอุปกรณ์ก็ได้ถ้าต้องการใช้อุปกรณ์ที่ติดตั้งไว้โดยปริยาย (default device) ด้วยการเรียกฟังก์ชัน DirectSoundCreate8

แต่ถ้าต้องการที่จะสร้างอุปกรณ์อย่างเฉพาะเจาะจงหรือต้องการที่จะให้ผู้ใช้งานสามารถเลือกอุปกรณ์ที่ต้องการจะใช้งานได้ หรือในกรณีที่จะต้องใช้งานอุปกรณ์มากกว่า 1 ตัว ก็มีความจำเป็นที่จะต้องทำการระบุอุปกรณ์ที่มีอยู่ในเครื่องคอมพิวเตอร์ก่อน

การระบุอุปกรณ์ในไดเรกทอรีมีจุดประสงค์ ดังนี้

1. เพื่อตรวจสอบว่าฮาร์ดแวร์สามารถใช้งานได้หรือไม่
2. เพื่อที่จะกำหนดค่าเฉพาะเจาะจง (Globally unique identifier GUID) ให้กับแต่ละอุปกรณ์
3. สามารถสร้างไดเรกทอรีออปเจกต์สำหรับแต่ละอุปกรณ์ที่ทำการระบุอยู่แบบชั่วคราวเพื่อใช้ในการตรวจสอบความสามารถของอุปกรณ์ได้

ในการระบุอุปกรณ์จะต้องกำหนดฟังก์ชันเรียกกลับ (callback function) ที่จะถูกเรียกทุกๆ ครั้งที่ไดเรกทอรีทำการระบุอุปกรณ์ ซึ่งในฟังก์ชันนี้สามารถที่จะทำการประมวลผลใดๆ ก็ได้ และสามารถกำหนดชื่อฟังก์ชันอย่างไรก็ได้แต่จะต้องอยู่ในรูปแบบที่กำหนดไว้ใน DSEnumCallback โดยฟังก์ชันเรียกกลับนี้จะต้องส่งค่า TRUE ถ้าต้องการให้มีการระบุอุปกรณ์ต่อไป และส่งค่า FALSE ถ้าต้องการให้จบขั้นตอนการระบุอุปกรณ์

ตัวอย่าง ฟังก์ชันเรียกกลับต่อไปนี้ทำการเพิ่มชื่อของอุปกรณ์ในบัญชีของคอมพิวเตอร์

```

BOOL CALLBACK DSEnumProc(LPGUID lpGUID, LPCTSTR lpszDesc,
                          LPCTSTR lpszDrvName, LPVOID lpContext )
{
    HWND hCombo = *(HWND *)lpContext;
    LPGUID lpTemp = NULL;
    if ( lpGUID != NULL )
    {
        if (( lpTemp = malloc( sizeof(GUID))) == NULL )
            return( TRUE );
        memcpy( lpTemp, lpGUID, sizeof(GUID));

        ComboBox_AddString( hCombo, lpszDesc );
        ComboBox_SetItemData( hCombo, ComboBox_FindString( hCombo, 0, lpszDesc ), lpTemp );
        return( TRUE );
    }
}

```

8.4 การสร้างไดเรกทอรีออปเจกต์

วิธีการสร้างไดเรกทอรีออปเจกต์ที่ง่ายที่สุดคือการเรียกฟังก์ชัน DirectSoundCreate8 ค่าพารามิเตอร์ตัวแรกคือ ค่า GUID ของอุปกรณ์ที่จะติดต่อกับออปเจกต์ ซึ่งค่านี้สามารถหาได้จากขั้นตอนการระบุอุปกรณ์ หรือจะส่งค่าที่มีการกำหนดเอาไว้แล้วดังต่อไปนี้

DSDEVID_DefaultPlayback

ใช้อุปกรณ์ที่กำหนดไว้โดยปริยายโดยระบบ ถ้าต้องการระบุค่าพารามิเตอร์นี้อาจจะส่งค่า NULL เข้าไปแทน

DSDEVID_DefaultVoicePlayback

ค่านี้คือการกำหนดให้ใช้อุปกรณ์ติดต่อเสียง (voice communication device) โดยปริยาย ซึ่งโดยปกติแล้วอุปกรณ์นี้จะเป็นหูฟังแบบ USB ที่มีไมโครโฟน

ถ้าไม่มีอุปกรณ์ที่กำหนด การเรียกฟังก์ชัน DirectSoundCreate8 จะไม่สำเร็จ ตัวอย่างการสร้างโคเร็กเซว

LPDIRECTSOUND8 lpds;

HRESULT hr = DirectSoundCreate8(NULL, &lpds, NULL);

8.5 ระดับการทำงานร่วมกัน (Cooperative Levels)

เนื่องจากในเวลาขณะหนึ่งอาจจะมีแอปพลิเคชันทำงานอยู่มากกว่า 1 แอปพลิเคชัน การกำหนดระดับการทำงานร่วมกันเป็นการกำหนดให้โคเร็กเซวควบคุมไม่ให้มีแอปพลิเคชันเข้าใช้อุปกรณ์ในทางที่ผิดหรือผิดเวลา แอปพลิเคชันแต่ละตัวจะมีระดับการทำงานร่วมกันของตัวเองที่จะเป็นตัวกำหนดจำนวนของแอปพลิเคชันอื่นๆ ที่สามารถเข้าใช้งานอุปกรณ์ได้

หลังจากการสร้างออปเจกต์โคเร็กเซวแล้วจะต้องกำหนดค่าระดับการทำงานร่วมกันโดยเรียกเมทอด IDirectSound8::SetCooperativeLevel ก่อนที่จะเล่นเสียง

HRESULT hr = lpDirectSound->SetCooperativeLevel(hwnd, DSSCL_PRIORITY);

ระดับการทำงานร่วมกันแบบปกติ (Normal Cooperative Level)

ในแบบนี้แอปพลิเคชันไม่สามารถกำหนดรูปแบบของบัพเฟอร์ปฐมภูมิได้ นั่นคือ จะต้องใช้บัพเฟอร์ปฐมภูมิที่มีความถี่ 22 kHz แบบสเตอริโอ และ 8-bit แชนแนล

ระดับการทำงานร่วมกันแบบไพอริตี (Priority Cooperative Level)

เมื่อใช้อุปกรณ์โคเร็กเซวในระดับไพอริตี แล้วแอปพลิเคชันมีสิทธิที่จะเข้าถึงรีซอร์สบนฮาร์ดแวร์ เช่น ฮาร์ดแวร์มิกซิง (Hardware Mixing), สามารถกำหนดรูปแบบของบัพเฟอร์ปฐมภูมิ

แอปพลิเคชันจำพวกเกมสควร์ใช้ระดับการทำงานร่วมกันในระดับนี้เนื่องจากสามารถกำหนดการแซมเพิลและความถี่ได้

ระดับการทำงานร่วมกันแบบเขียนบนปฐภูมิ (Write-primary Cooperative Level)

ระดับนี้เป็นระดับที่แอปพลิเคชันสามารถเข้าถึงบัฟเฟอร์ปฐภูมิได้โดยตรง นั่นคือ แอปพลิเคชันจะเขียนข้อมูลโดยตรงลงบนบัฟเฟอร์ปฐภูมิและจะไม่สามารถเล่นเสียงจากบัฟเฟอร์ทุติยภูมิได้

8.6 ความสามารถของอุปกรณ์ (Device Capabilities)

ไคเร็กเซวาร์ับข้อมูลความสามารถของอุปกรณ์เพื่อที่ใช้ข้อมูลนั้นในการปรับความคุณสมบัติต่างๆ ให้ใช้ประโยชน์จากอุปกรณ์ที่มีความสามารถในการเร่งความเร็วได้มากที่สุด หลังจากที่สร้างออปเจกต์ไคเร็กเซวาร์แล้วสามารถรับค่าความสามารถของอุปกรณ์เสียงได้ด้วยการเรียกเมทอด `IdirectSound8::`

`GetCaps`

ตัวอย่าง

`DSCAPS dscaps;`

`dscaps.dwSize = sizeof(DSCAPS);`

`HRESULT hr = lpDirectSound->GetCaps(&dscaps);`

โครงสร้างข้อมูล DSCAPS รับข้อมูลเกี่ยวกับประสิทธิภาพและรีซอร์สของอุปกรณ์ รวมทั้งขนาดของรีซอร์สทั้งหมดและขนาดของรีซอร์สที่สามารถใช้งานได้ขณะนั้น ถ้าแอปพลิเคชันต้องการจะเพิ่มส่วนการใช้งานฮาร์ดแวร์ควรจะเรียกเมทอดนี้ทุกครั้งก่อนที่จะทำ

8.7 ไคเร็กเซวาร์บัฟเฟอร์

ไคเร็กเซวาร์บัฟเฟอร์ออปเจกต์ควบคุมการส่งข้อมูลเวฟจากต้นทางไปสู่ปลายทาง ต้นทางอาจจะเป็นซินทีไซเซอร์ (synthesizer), บัฟเฟอร์ตัวอื่น, ไฟล์ หรือรีซอร์ส ปลายทางโดยปกติแล้วจะเป็นตัวผสมเสียงในบัฟเฟอร์ปฐภูมิ และจากบัฟเฟอร์ปฐภูมิข้อมูลจะถูกส่งไปยังฮาร์ดแวร์ซึ่งจะแปลงค่า PCM แซมเพิลไปเป็นคลื่นเสียง

บัฟเฟอร์

เมื่อไคเร็กเซวาร์เริ่มต้นการทำงานจะสร้างบัฟเฟอร์ปฐภูมิโดยอัตโนมัติและจัดการกับบัฟเฟอร์ปฐภูมิสำหรับการผสมเสียงและส่งข้อมูลไปยังอุปกรณ์เอาต์พุต โดยทั่วไปแล้ว บัฟเฟอร์แบบปฐภูมิจะได้รับการควบคุมจากไคเร็กเซวาร์อยู่เบื้องหลัง ดังนั้น แอปพลิเคชันจึงไม่จำเป็นต้องสนใจความเป็นไปของมัน

แอปพลิเคชันจะต้องสร้างบัฟเฟอร์ทุติยภูมิอย่างน้อย 1 บัฟเฟอร์เพื่อเก็บข้อมูลเสียงที่ต้องการจะเล่น บัฟเฟอร์แบบทุติยภูมินี้สามารถที่จะสร้างขึ้นและให้คงอยู่ตลอดไปหรือทำลายทิ้งเมื่อไม่ต้องการก็ได้ และสามารถเก็บข้อมูลที่ใช้เล่นซ้ำที่ครั้งก็ได้ หรือจะเขียนข้อมูลใหม่ทุกครั้งที่จะเล่น นอกจากนั้นยังสามารถกำหนดให้การเล่นเป็นแบบครั้งเดียวจบหรือเล่นวนไปเรื่อยๆ ได้เช่นกัน และยังสามารถใช้กับสตรีมในกรณีที่มีขนาดของไฟล์ใหญ่เกินไปได้อีกด้วย

ตำแหน่งของบัฟเฟอร์สามารถอยู่ได้ทั้งในฮาร์ดแวร์และซอฟต์แวร์ ในกรณีของฮาร์ดแวร์การผสมเสียงจะเกิดขึ้นโดยตัวประมวลผลบนการ์ดเสียง ส่วนในกรณีของซอฟต์แวร์จะผสมเสียงโดยใช้ CPU นอกจากนั้น ข้อมูลของซอฟต์แวร์จะอยู่ในเมมโมรีของระบบเสมอ ส่วนฮาร์ดแวร์สามารถอยู่ได้ทั้งในเมมโมรีของระบบและเมมโมรีบนการ์ดเสียง

การผสมเสียงสามารถทำได้โดยการสั่งให้มีการเล่นข้อมูลในบัฟเฟอร์ทุกขัณภูมิพร้อมๆ กัน ไม่มี การจำกัดจำนวนของบัฟเฟอร์ที่สามารถเล่นได้ในไดเร็กทอรี ดังนั้นจำนวนของบัฟเฟอร์ที่สามารถเล่นได้ จะขึ้นอยู่กับความสามารถในการประมวลผลเท่านั้น

การสร้างบัฟเฟอร์ทุกขัณภูมิ

การสร้างบัฟเฟอร์ทุกขัณภูมิทำได้โดยการเรียกเมทอด `IdirectSound8::CreateSoundBuffer` ซึ่งจะส่งค่าพอยเตอร์ที่ชี้ไปยังอินเทอร์เฟซ `IdirectSoundBuffer8` ซึ่งใช้ในการควบคุมและการเล่นบัฟเฟอร์ ตัวอย่าง

```

BOOL AppCreateBasicBuffer( LPDIRECTSOUND8 lpDirectSound,
    LPDIRECTSOUNDBUFFER *lplpDsb)
{
    PCMWAVEFORMAT pcmwf;
    DSBUFFERDESC dsbdesc;
    HRESULT hr;
    // Set up wave format structure.
    memset(&pcmwf, 0, sizeof(PCMWAVEFORMAT));
    pcmwf.wf.wFormatTag = WAVE_FORMAT_PCM;
    pcmwf.wf.nChannels = 2;
    pcmwf.wf.nSamplesPerSec = 22050;
    pcmwf.wf.nBlockAlign = 4;
    pcmwf.wf.nAvgBytesPerSec = pcmwf.wf.nSamplesPerSec * pcmwf.wf.nBlockAlign;
    pcmwf.wBitsPerSample = 16;
    // Set up DSBUFFERDESC structure.
    memset(&dsbdesc, 0, sizeof(DSBUFFERDESC));
    dsbdesc.dwSize = sizeof(DSBUFFERDESC);
    dsbdesc.dwFlags=
        DSBCAPS_CTRLPAN | DSBCAPS_CTRLVOLUME | DSBCAPS_CTRLFREQUENCY;
    dsbdesc.dwBufferBytes = 3 * pcmwf.wf.nAvgBytesPerSec;
    dsbdesc.lpwfxFormat = (LPWAVEFORMATEX)&pcmwf;

```



```
// Create buffer.

hr = lpDirectSound->CreateSoundBuffer(&dsbdesc, lpplpDsb, NULL);
if SUCCEEDED(hr)
{
    // IDirectSoundBuffer interface is in *lpplpDsb.

    // Use QueryInterface to obtain IDirectSoundBuffer8.

    return TRUE;
}
else
{
    // Failed.

    *lpplpDsb = NULL;

    return FALSE;
}
}
```

8.8 การใช้งานข้อมูล Wave

โคเร็กเซิร์ฟเวอร์สามารถเล่นได้เฉพาะข้อมูลเสียงในแบบเวฟ (waveform audio data) เท่านั้น ซึ่งข้อมูลจะประกอบด้วยค่าดิจิทัลแซมเพิล (Digital sample) ของเสียงในอัตราแซมเพิลที่คงที่ ซึ่งการแทนสัญญาณแบบอนาลอกด้วยตัวเลขที่เรียงเป็นลำดับเรียกว่า pulse code modulation (PCM)

ข้อมูลเวฟโดยปกติจะเก็บอยู่ในรูปของไฟล์หรือรีซอร์สใน Resource Interchange File Format (RIFF) ข้อมูลจะเก็บส่วนที่บรรยายรูปแบบของข้อมูล รวมทั้งพารามิเตอร์ต่างๆ เช่น อัตราแซมเพิลหรือจำนวนของช่องทางเอาต์พุต (output channel) รูปแบบของเสียงบางรูปแบบได้สามารถใช้โครงสร้างข้อมูล WAVEFORMATEX หรือ WAVEFORMATEXTENSIBLE ในการอธิบายได้

การอ่านข้อมูลเวฟจากไฟล์

เนื่องจากโคเร็กเซิร์ฟเวอร์ไม่ได้สร้างเมทอดสำหรับการอ่านไฟล์เอาไว้ให้ แต่ใน Dsutil.cpp ที่มาพร้อมกับ SDK ได้สร้างคลาส CwaveFile ไว้ให้แล้วซึ่งมีฟังก์ชันการทำงานดังนี้

Open เปิดไฟล์สำหรับการอ่านและรับข้อมูลรูปแบบของไฟล์หรือเปิดไฟล์สำหรับการเขียนและเขียนข้อมูลในส่วนของหัวไฟล์

GetSize หาขนาดของข้อมูล สามารถเรียกได้หลังจากการเปิดไฟล์

Read คัดลอกข้อมูลไปยังบัฟเฟอร์และเลื่อนตัวชี้ตำแหน่งการอ่านข้อมูล

Write เขียนข้อมูลจากบัฟเฟอร์และเลื่อนตัวชี้ตำแหน่งของการเขียนข้อมูล

ResetFile เลื่อนตัวชี้ทั้งการอ่านและเขียนข้อมูลกลับมาที่จุดเริ่มต้น

Close ปิดไฟล์

บทที่ 9

ไคเร็กโชว์ (DirectShow)

9.1 แนะนำไคเร็กโชว์

Microsoft® DirectShow® Application Programming Interface (API) คือ สถาปัตยกรรมของมีเดียสตรีมมิ่ง (media-streaming) สำหรับระบบปฏิบัติการวินโดวส์ การใช้ไคเร็กโชว์ (DirectShow) ช่วยให้แอปพลิเคชันสามารถเล่นไฟล์วีดีโอคุณภาพสูงและเล่นหรือบันทึกไฟล์เสียงได้

ไคเร็กโชว์สนับสนุนไฟล์รูปแบบต่างๆ จำนวนมาก เช่น Advanced Streaming Format (ASF), Motion Picture Experts Group (MPEG), Audio-Video Interleaved (AVI), MPEG Audio Layer-3 (MP3) และ Wave รวมทั้งสนับสนุนการบันทึกโดยผ่านทางอุปกรณ์ Windows Driver Model (WDM) หรืออุปกรณ์อื่นๆ สำหรับวินโดวส์ ไคเร็กโชว์ได้รวมเอาเทคโนโลยีของไคเร็กเอ็กซ์อื่นๆ มาไว้รวมกัน โดยจะตรวจสอบโดยอัตโนมัติและเลือกใช้อาร์คแวร์ในการเร่งความเร็ววีดีโอและออดิโอ (video and audio acceleration hardware) เมื่อมีอุปกรณ์นั้นอยู่ นอกจากนั้นยังสนับสนุนการใช้งานโดยไม่ผ่านฮาร์ดแวร์อีกด้วย

ไคเร็กโชว์ทำให้การเล่นไฟล์มีเดีย, การแปลงรูปแบบไฟล์ และการบันทึกเป็นเรื่องง่าย แต่ในขณะเดียวกันก็สามารถเข้าถึงสถาปัตยกรรมควบคุมสตรีมที่อยู่ข้างใต้ได้เช่นกันสำหรับแอปพลิเคชันที่ต้องการผลลัพธ์ที่ต่างออกไป นั่นคือ สามารถที่จะสร้างองค์ประกอบของไคเร็กโชว์ขึ้นมาใหม่ได้เพื่อที่จะสนับสนุนไฟล์รูปแบบใหม่ๆ หรือเอฟเฟ็กต์ที่แตกต่างออกไป

ไคเร็กโชว์อยู่บนพื้นฐานของ Component Object Model (COM) ในการเขียนแอปพลิเคชันที่ใช้งานไคเร็กโชว์จึงมีความจำเป็นที่ต้องทำความเข้าใจกับ COM client programming (การเขียนแอปพลิเคชันที่ทำงานบนเครื่องคอมพิวเตอร์เดียวโดยใช้ COM) ซึ่งส่วนมากแล้วมักจะไม่ใช่สิ่งที่จะต้องสร้าง COM object ขึ้นมาเอง เนื่องจากไคเร็กโชว์ได้สร้างองค์ประกอบที่ต้องการไว้ให้แล้ว (แต่สามารถสร้างเพิ่มเติมได้)

9.2 เริ่มต้นการเขียนโปรแกรมด้วยไคเร็กโชว์

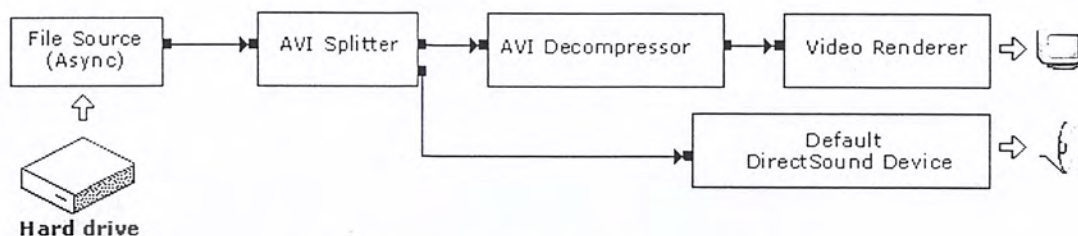
ในการเริ่มต้นเขียนโปรแกรมด้วยไคเร็กโชว์จำเป็นที่จะต้องทำความเข้าใจกับศัพท์และองค์ประกอบที่ประกอบกันในการทำงานของไคเร็กโชว์ดังนี้

ฟิลเตอร์ (Filter), ฟิลเตอร์กราฟ (Filter Graphs) และ ฟิลเตอร์กราฟเมนเจอร์ (Filter Graph Manager)

ในไคเร็กโชว์มีองค์ประกอบของซอฟต์แวร์ (Software component) ที่เรียกว่า ฟิลเตอร์ (filter) เป็นองค์ประกอบพื้นฐาน ซึ่งฟิลเตอร์จะทำกระบวนการ 1 อย่างกับมัลติมีเดียสตรีม เช่น มีไคเร็กโชว์ฟิลเตอร์ที่ทำการอ่านไฟล์, รับข้อมูลจากกล้องวีดีโอ, ถอดรหัส (decode) บางส่วนของสตรีม เช่น MPEG-1 หรือ ส่งข้อมูลไปยังกราฟิกการ์ด

ฟิลเตอร์รับข้อมูลและสร้างเอาต์พุตออกมา เช่น ถ้าฟิลเตอร์ถอดรหัส MPEG-1 video อินพุตคือสตรีมที่ผ่านการเข้ารหัสด้วย MPEG และเอาต์พุตคือ สตรีมข้อมูลที่ไม่มีการบีบอัดในแบบ RGB video ดังนั้นในการทำงาน 1 อย่าง แอปพลิเคชันจะต้องนำเอาฟิลเตอร์เหล่านี้มาต่อเข้าด้วยกันโดยนำเอาต์พุตของ

ฟิลเตอร์หนึ่งมาต่อกับอินพุตของอีกฟิลเตอร์หนึ่ง กลุ่มของการเชื่อมต่อของฟิลเตอร์นี้เรียกว่า ฟิลเตอร์กราฟ ดังรูป



รูปที่ 9-1 แสดงการเชื่อมต่อฟิลเตอร์เพื่อเล่นไฟล์ AVI

แอปพลิเคชัน ไม่จำเป็นต้องเข้ามาควบคุมแต่ละฟิลเตอร์โดยตรง เนื่องจากไคลเร็กโซว์ได้เตรียมองค์ประกอบขั้นสูง (high-level component) ในการควบคุมไว้แล้ว ที่เรียกว่า ฟิลเตอร์กราฟเมเนเจอร์ (Filter Graph Manager)

ฟิลเตอร์กราฟเมเนเจอร์ควบคุมการไหลของสตรีมข้อมูล โดยแอปพลิเคชันสามารถควบคุมได้ โดยการเรียกฟังก์ชันง่ายๆ เช่น Run (ส่งข้อมูลเข้าสู่กราฟ), Stop (หยุดการไหลของข้อมูล) ถ้าต้องการที่จะควบคุมฟิลเตอร์โดยตรงก็สามารถทำได้เช่นกันโดยผ่านทาง COM Interface นอกจากนั้น ฟิลเตอร์กราฟเมเนเจอร์ยังสามารถส่งค่าแจ้งเหตุการณ์ (event notification) ให้กับแอปพลิเคชันได้อีกด้วย ตัวอย่าง เช่น การแจ้งว่าจบสตรีมข้อมูลแล้ว เป็นต้น

นอกจากนั้น ฟิลเตอร์กราฟเมเนเจอร์ยังทำให้กระบวนการในการสร้างฟิลเตอร์กราฟมีความง่ายมากยิ่งขึ้น เช่น เพียงแค่บอกชื่อของไฟล์ที่ต้องการจะเล่น ฟิลเตอร์กราฟจะไปสร้างกราฟที่เหมาะสมสำหรับการเล่นไฟล์ให้ทันที

ฟิลเตอร์ในไคลเร็กโซว์แบ่งเป็นประเภทต่างๆ ดังนี้

Source Filter

แทนข้อมูลมัลติมีเดียที่เป็นวัตถุในกระบวนการ ซึ่งอาจจะได้มาจากไฟล์บนฮาร์ดดิสต์ หรือจากแผ่น CD, DVD ในไดรฟ์ หรืออาจจะได้มาจากรายการสดผ่านทางอุปกรณ์รับสัญญาณโทรทัศน์ หรือจากกล้องดิจิทัล เป็นต้น

Transform Filter

เป็นฟิลเตอร์ที่รับข้อมูลวัตถุหรือข้อมูลที่มีการประมวลผลแล้วบางส่วนมาเพื่อที่จะประมวลผลข้อมูลเหล่านั้น ซึ่งการประมวลผลอาจจะเป็นการแบ่งข้อมูลออกเป็นส่วนย่อยๆ , การบีบอัดและขยาย และการแปลงรูปแบบ (format converter)

Render Filter

เป็นฟิลเตอร์ที่จะรับข้อมูลที่ได้ประมวลผลแล้วมาเพื่อแสดงภาพบนจอมอนิเตอร์หรือส่งสัญญาณเสียงผ่านทางลำโพง หรืออุปกรณ์อื่นๆ รวมทั้งการเขียนลงไฟล์บนดิสก์ ที่มีชื่อเรียกเฉพาะว่า file writer

Pins

จากรูปฟิลเตอร์กราฟจะเห็นว่าข้อมูลสตรีมมิ่งจะเดินทางจากซอร์สฟิลเตอร์ผ่านฟิลเตอร์อื่นๆ และไปจบที่เรนเดอร์ฟิลเตอร์ การเชื่อมต่อระหว่างฟิลเตอร์จะต้องใช้พินในการเชื่อมต่อ พินคือ รายละเอียดในระดับล่างของข้อมูลที่จะส่งระหว่างฟิลเตอร์ พินเป็น COM object ที่สนับสนุนโดย Ipin COM interface มีทิศทาง (อินพุตและเอาต์พุต) และมีการติดต่อกับบางส่วนของฟิลเตอร์ที่มีอยู่ในกราฟ พินแสดงการเชื่อมต่อระหว่างแต่ละฟิลเตอร์ พินจะรู้ว่าชนิดของมีเดียใดที่ใช้งานได้และจะมีการเจรจาต่อรองระหว่างพินเพื่อเลือกชนิดของมีเดียที่จะใช้ในการติดต่อ เมื่อกำหนดชนิดได้แล้วจะต้องมีการเจรจาต่อไปในการกำหนดวิธีการในการส่งข้อมูลระหว่างฟิลเตอร์

มีเดียแซมเพิล (Media Samples)

หลังจากข้อมูลวัตถุดิบได้ส่งผ่านเข้าสู่กราฟไม่ว่าจะมาจากไหนก็ตาม จะต้องมีการจัดให้เป็นหน่วยที่มีความหมาย (meaningful unit) ที่เรียกว่า มีเดียแซมเพิล (Media Sample) ในโคเร็กโซว์มีเดียแซมเพิลถูกห่อหุ้ม (Wrap) ด้วย COM object ที่อิมพลีเมนต์ (Implement) ImediaSample2 ในวิดีโอคือ ข้อมูล 1 เฟรมเป็นต้น

นาฬิกา (Clock)

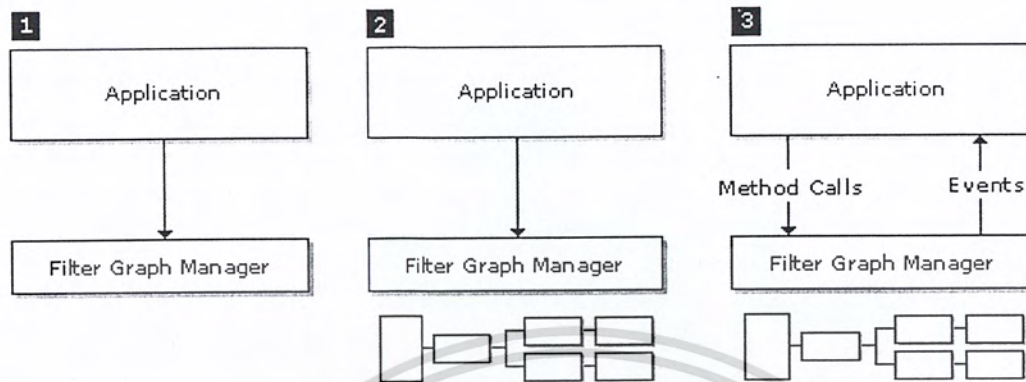
ในกระบวนการที่เกี่ยวข้องกับมัลติมีเดีย จะต้องมีการควบคุมจังหวะของการแสดงผลให้มีความถูกต้อง เช่น วิดีโอจะต้องควบคุมอัตราเฟรม (Frame rate) ให้เป็นไปตามที่กำหนด ซึ่งโคเร็กโซว์มีเดียแซมเพิลจะต้องใช้งานร่วมกันในกระบวนการต่างๆ โดยฟิลเตอร์กราฟเมเนเจอร์จะเลือกนาฬิกาขึ้นมา และส่งให้ฟิลเตอร์ทุกตัวใช้นาฬิกาตัวนั้นเป็นจังหวะเดียวกัน

ตัวจองหน่วยความจำ (Allocator)

มีฟิลเตอร์ 2 ตัวมีการเชื่อมต่อกัน พินจะต้องตกลงในรายละเอียดในการส่งข้อมูลและชนิดของข้อมูล ในการเชื่อมต่อจะต้องกำหนดขนาด, ตำแหน่ง และจำนวนของแซมเพิลที่จะใช้ ขนาดของแซมเพิลจะขึ้นอยู่กับชนิดของมีเดียและรูปแบบของมีเดีย และตำแหน่งของบัฟเฟอร์อาจอยู่บนเมมโมรีของระบบหรือที่ใดก็ได้ เช่น บนการ์ดบันทึกภาพวิดีโอ การสร้างและการจัดการสิ่งเหล่านี้ทำโดยตัวจองหน่วยความจำ

9.3 การเขียนโปรแกรมโดยใช้ไคเร็กโซว์

โดยปกติแล้วแอปพลิเคชันที่จะใช้งานไคเร็กโซว์จะมีขั้นตอนพื้นฐาน 3 ขั้นตอน ดังรูป



รูปที่ 9-2 ขั้นตอนการสร้างไคเร็กโซว์

- ขั้นที่ 1 การสร้างอินสแตนซ์ (Instance) ของฟิลเตอร์กราฟเมนเนเจอร์ โดยเรียกฟังก์ชัน CoCreateInstance
- ขั้นที่ 2 ใช้ฟิลเตอร์กราฟเมนเนเจอร์ในการสร้างฟิลเตอร์กราฟ (อาจจะต้องใช้ฟังก์ชันประกอบอื่นๆที่มีอยู่ในไคเร็กโซว์ช่วย)
- ขั้นที่ 3 ควบคุมฟิลเตอร์กราฟและตอบสนองต่อเหตุการณ์ (event)

9.4 การเล่นไฟล์ (Play a File)

การที่จะเล่นไฟล์โดยใช้ไคเร็กโซว์มีขั้นตอนดังนี้

1. สร้างอินสแตนซ์ของฟิลเตอร์กราฟเมนเนเจอร์
2. ใช้ฟิลเตอร์กราฟเมนเนเจอร์เพื่อสร้างฟิลเตอร์กราฟ
3. ใช้ฟิลเตอร์กราฟเมนเนเจอร์ในการทำงานของฟิลเตอร์กราฟ
4. รอให้การเล่น (playback) จบ

การจะทำขั้นตอนทั้ง 4 อย่างข้างต้นจะต้องใช้ COM Interface ต่อไปนี้

IQueryBuilder สร้างฟิลเตอร์กราฟ

IMediaControl จัดการมีเดียสตรีมมิ่งในฟิลเตอร์กราฟ

IMediaEvent จัดการเหตุการณ์ของฟิลเตอร์กราฟ

ซึ่งอินเทอร์เฟซข้างต้นทั้งหมดคือ ฟิลเตอร์กราฟเมนเนเจอร์

ขั้นที่ 1

เรียกฟังก์ชัน CoInitialize เพื่อตั้งค่าเริ่มต้นให้กับ COM Library แล้วเรียก CoCreateInstance เพื่อสร้างฟิลเตอร์กราฟเมเนเจอร์ ดังตัวอย่าง

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

ฟังก์ชัน CoCreateInstance จะคือค่าพอยเตอร์ที่ชี้ไปยังอินเทอร์เฟซ IGraphBuiler ในฟิลเตอร์กราฟเมเนเจอร์คืนมาให้ แล้วใช้อินเทอร์เฟซนี้ในการขออีก 2 อินเทอร์เฟซที่เหลือดังตัวอย่าง

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

ขั้นที่ 2

ใช้เมทอด IGraphBuilder::RenderFile เพื่อสร้างฟิลเตอร์กราฟที่จะใช้ในการเล่นไฟล์นั้นๆ คำที่ส่งให้เมทอดนี้คือ ชื่อของไฟล์ที่จะเล่นโดยจะต้องอยู่ในรูปแบบ wide character (2-byte) Unicode™ string โดยใช้ “L” นำหน้าชื่อไฟล์ที่ต้องการ (ทำการแปลง ASCII เป็น wide character string)

```
pGraph->RenderFile(L"C:\\Hello_World.avi", NULL);
```

ในกรณีที่ไม่มีพบไฟล์หรือไม่สามารถสร้างฟิลเตอร์กราฟได้เมทอดจะส่งค่า VFW_E_NOT_FOUND คืนมาให้

ขั้นที่ 3

หลังจากฟิลเตอร์กราฟเมเนเจอร์สร้างฟิลเตอร์กราฟแล้วจึงเริ่มกระบวนการเล่นไฟล์โดยการเรียกเมทอด IMediaControl::Run เพื่อให้กราฟเริ่มทำงาน โดยจะส่งข้อมูลเข้าสู่กราฟ

ขั้นที่ 4

เมทอด IMediaEvent::WaitForCompletion จะบล็อกจนกว่าไฟล์จะเล่นจบ เนื่องจากการเล่นไฟล์จะเกิดขึ้นในเทรด (thread) ที่สร้างขึ้นต่างหาก

บทที่ 10

เทคนิคในการพัฒนาเกม 3 มิติ

10.1 เอนิเมชัน (Animation)

เอนิเมชันหรือการทำภาพเคลื่อนไหวเป็นเทคนิคที่มีความซับซ้อนมากยิ่งขึ้น ซึ่งเทคนิคที่นิยมใช้กันมี 2 ประเภทคือ เทคนิคคีย์เฟรมอินเทอร์โพลชัน (Keyframe Interpolation Techniques) และเทคนิคสกินนิ่ง (Skinning Techniques)

10.1.1 การทำเอนิเมชันโดยใช้เทคนิคคีย์เฟรมอินเทอร์โพลชัน (Keyframe Interpolation Technique)

คีย์เฟรม (Keyframe) คือ การสร้างการกระทำจากคีย์ด้วยการกำหนดคุณสมบัติให้แตกต่างกันไปตามเวลาที่เปลี่ยนแปลงไป

การใช้เทคนิคของคีย์เฟรมอินเทอร์โพลชัน ใน Direct3D มีหลักการดังนี้

- เก็บตำแหน่งของเวอร์เท็กซ์ไว้เป็นชุดตามค่าคีย์เฟรม การทำเอนิเมชันสามารถทำได้โดยการเฉลี่ยค่าระหว่างตำแหน่งของเวอร์เท็กซ์ในแต่ละชุดโดยใช้ค่าของเวลาเป็นตัวกำหนดน้ำหนักของการเฉลี่ย (Blending)
- สามารถเฉลี่ยค่าคุณสมบัติอื่นๆ ได้ด้วยวิธีการเดียวกัน เช่น ค่าแกนตั้งฉาก (normal) หรือค่าคู่อันดับกำหนดตำแหน่งของเทกเจอร์ (texture coordinate)
- การเฉลี่ยค่าสามารถทำได้โดยฮาร์ดแวร์ได้ (ทำงาน GPU) ด้วยการใช้เวอร์เท็กซ์เชดเดอร์ (vertex shader) และสตรีมข้อมูลเวอร์เท็กซ์หลายๆ สตรีม

ตัวอย่าง ผลงานที่ได้เกิดจากการใช้เทคนิคคีย์เฟรมอินเทอร์โพลชัน



รูปที่ 10-1 รูปแสดงภาพเคลื่อนไหวที่เกิดจากเทคนิคคีย์เฟรมอินเทอร์โพลชัน

การใช้เทคนิคคีย์เฟรมอินเทอร์โพลชันมี 2 ชนิด แบ่งตามวิธีการคำนวณค่าเวอร์เท็กซ์ คือ

10.1.1.1 เทคนิคคีย์เฟรมอินเทอร์โพลชันเชิงเส้น (Linear Keyframe Interpolation)

สมการที่ใช้ในการคำนวณค่าเวอร์เท็กซ์คือ

$$V_{final} = V_0 \times (1 - t) + V_1 \times t$$

เมื่อ ค่า V_{final} คือ ค่าที่ได้จากการเฉลี่ยค่า V_0 และค่า V_1 โดยใช้เวลากำหนดน้ำหนักในการเฉลี่ยค่า

10.1.1.2 เทคนิคคีย์เฟรมอินเทอร์โพลชันด้วย Hermite Spline (Hermite Spline Keyframe Interpolation)

การใช้ Hermite Spline ช่วยให้การเฉลี่ยค่าของเวอร์เท็กซ์มีความราบรื่นยิ่งขึ้น เนื่องจาก Spline เป็นสมการที่ใช้แสดงค่าของเส้นโค้ง แต่จะทำให้ต้องประมวลผลข้อมูลเพิ่มขึ้น
สมการที่ใช้ในการคำนวณค่าเวอร์เท็กซ์คือ

$$V_{final} = h_0 \times V_i + h_1 \times (V_{i+1} - V_{i-1}) + h_2 \times (V_{i+2} - V_i) + h_3 \times V_{i+1}$$

การใช้ Hermite Spline Keyframe Interpolation ต้องใช้สตริมของข้อมูล 4 สตริม สำหรับเก็บค่าช่วงก่อน
คำนวณ 2 สตริม และหลังจากคำนวณ 2 สตริม
รูปแสดงตัวอย่างของ Spline

รูปที่ 10-2 รูปแสดงเส้นโค้ง Spline

ข้อดีของคีย์เฟรมอินเทอร์โพลชัน (Keyframe Interpolation)

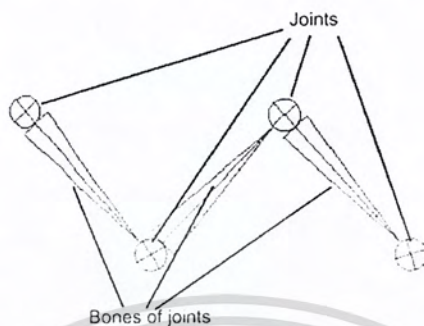
1. สามารถสร้างแอนิเมชันของเวอร์เท็กซ์ในรูปแบบใดๆ ก็ได้ โดยการกำหนดตำแหน่งของเวอร์เท็กซ์ในแต่ละเฟรม
2. การคำนวณในแบบเชิงเส้นเป็นการคำนวณง่ายๆ ที่ไม่มีความซับซ้อน

ข้อเสียของคีย์เฟรมอินเทอร์โพลชัน

1. ต้องใช้ข้อมูลจำนวนมาก เช่น ถ้าโมเดลของเราประกอบด้วยเวอร์เท็กซ์ 2000 เวอร์เท็กซ์ และในการทำท่าทาง 1 ท่า ใช้คีย์เฟรมจำนวน 10 คีย์เฟรม จะต้องเก็บข้อมูลพร้อมๆ กันทีเดียวถึง 2000×10 เท่ากับ 20000 เวอร์เท็กซ์ จึงเหมาะสำหรับวัตถุที่มีจำนวนเวอร์เท็กซ์น้อยๆ เท่านั้น

10.1.2 การทำแอนิเมชันโดยใช้เทคนิคสกินนิ่ง (Skinning Technique)

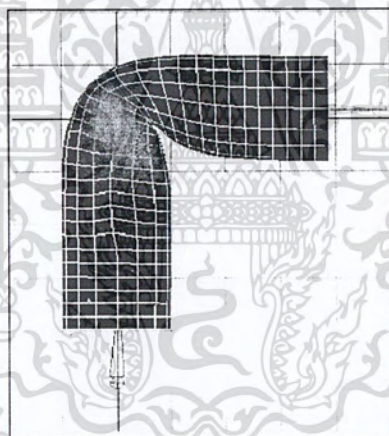
โบน (Bone) และจอยท์ (Joint) เป็นพื้นฐานสำคัญในการใช้เทคนิคสกินนิ่ง จอยท์คือ จุดที่มีการเชื่อมต่อกันของโบน ส่วนโบน คือ ความสัมพันธ์ระหว่างจอยท์ 2 จอยท์ ดังรูป



รูปที่ 10-3 รูปแสดงโบนและจอยท์

หลักการของเทคนิคสกินนิ่ง

เทคนิคสกินนิ่ง คือ การกำหนดค่าน้ำหนักให้กับแต่ละเวอร์เท็กซ์ให้มีการเคลื่อนที่สอดคล้องกับโบนที่เคลื่อนที่ไป ดังภาพ

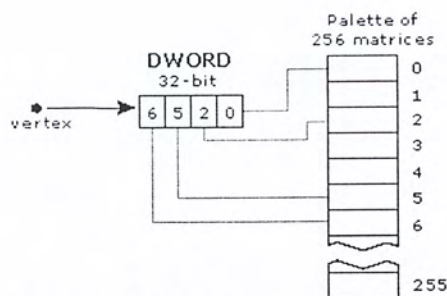


รูปที่ 10-4 รูปแสดงการเคลื่อนที่ค่าที่เกิดจากการใช้เทคนิคสกินนิ่ง

ใน Direct3D เวอร์ชัน 8 มีวิธีการที่เราจะใช้เทคนิคสกินนิ่ง 2 วิธี คือ

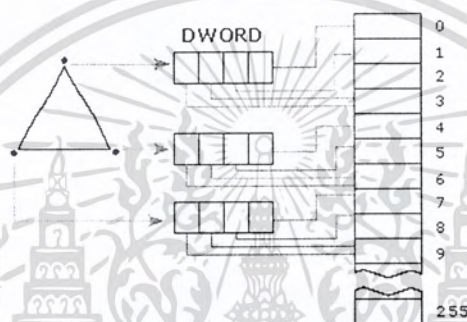
10.1.2.1 Fixed-function matrix palette skinning

หรือ Indexed Vertex Blending ในไคเร็กทรีดีมีการกำหนดค่าของโบนในรูปของเมตริกซ์ โดยการอ้างอิงโบนจะใช้ค่าอินเด็กซ์ ซึ่งสามารถกำหนดค่าอินเด็กซ์ (index) ของโบน (bone) ที่จะใช้ได้ถึง 4 ค่าต่อ 1 เวอร์เท็กซ์และพาเลตต์ (palette) ของโบนสามารถมีได้ถึง 256 เมตริกซ์ ดังรูป



รูปที่ 10-5 รูปแสดงการอ้างอิงข้อมูลในเวอร์เท็กซ์ซึ่งชี้ไปยังเมตริกซ์

หรือสามารถใช้ได้ถึง 12 ค่าต่อ 1 รูปสามเหลี่ยม ดังรูป



รูปที่ 10-6 รูปแสดงการอ้างอิงข้อมูลจากพื้นผิวไปยังเมตริกซ์

การคำนวณค่าของเวอร์เท็กซ์ สามารถทำได้ดังสมการ

$$V_{world} = V_m \times M[index0] \times b_0 + V_m \times M[index1] \times b_1 + V_m \times M[index2] \times b_2 + V_m \times M[index3] \times (1 - b_0 - b_1 - b_2)$$

V_{model} คือ ค่าตำแหน่งของเวอร์เท็กซ์ใน model space

$Index0..Index3$ คือค่าของอินเด็กซ์ของเมตริกซ์ต่อเวอร์เท็กซ์ 1 ค่าซึ่งรวมกันอยู่ในรูปของ DWORD

$M[]$ คือ อาร์เรย์ของ world เมตริกซ์ที่ใช้ค่าอินเด็กซ์เป็นตัวชี้

$b_0..b_2$ คือ ค่าน้ำหนักในการเฉลี่ยสำหรับเมตริกซ์ในแต่ละค่าอินเด็กซ์

V_{world} คือ ผลลัพธ์ที่ได้ซึ่งอยู่ใน world space

การกำหนดค่าของ Indexed Vertex Blending Transform และสถานะของการเรนเดอร์

สามารถกำหนดค่าของเมตริกซ์ได้ด้วยการเรียกเมทอด IDirect3DDevice8::SetTransform แล้วใช้มาโคร D3DTS_WORLDMATRIX ในการกำหนดค่าให้กับเมตริกซ์โดยใช้ค่าอินเด็กซ์ระหว่าง 0-255 ดังตัวอย่าง


```
D3DMATRIX matBlend1;
D3DXMatrixIdentity( &matBlend1 );
d3dDevice->SetTransform( D3DTS_WORLDMATRIX(0), &matBlend );
```

ในการใช้งาน Indexed Vertex Blending ยังต้องกำหนดสถานะการเรนเดอร์อีกด้วยโดยใช้เมทอด `IDirect3DDevice8::SetRenderState` ใส่ค่า `D3DRS_INDEXEDVERTEXBLENDENABLE` ในพารามิเตอร์ตัวแรก และค่า `TRUE` ในพารามิเตอร์ตัวที่สอง ดังตัวอย่าง

```
d3dDevice->SetRenderState( D3DRS_INDEXEDVERTEXBLENDENABLE, TRUE );
```

นอกจากนั้น ยังต้องกำหนดจำนวนของอินเด็กซ์ที่จะใช้ในการคำนวณด้วยการเรียกเมทอด `IDirect3DDevice::SetRenderState` ใส่ค่า `D3DRS_VERTEXBLEND` เป็นพารามิเตอร์แรก และใส่ค่าซึ่งเป็นสมาชิกของชนิดข้อมูล `D3DVERTEXBLEND_FLAGS` enumerated type เป็นพารามิเตอร์ตัวที่สอง ดังตัวอย่าง

```
d3dDevice->SetRenderState( D3DRS_VERTEXBLEND, D3DVBF_3WEIGHTS );
```

ข้อดีของการใช้ Fixed-function matrix palette skinning

2. ใช้งานง่าย
3. ใช้ข้อมูลจำนวนน้อยในการทำแอนิเมชัน 1 ทำ เนื่องจากเก็บเฉพาะค่าเมตริกซ์ที่เป็นคีย์เท่านั้น
4. สามารถเพิ่มเติมข้อมูลแอนิเมชันเข้าไปใหม่ได้โดยไม่ต้องแก้ไขโมเดล

ข้อเสีย

1. ไม่มีฮาร์ดแวร์ที่ช่วยเร่งความเร็วในการทำงาน และฮาร์ดแวร์ที่มีอยู่ก็ไม่สามารถใช้ได้ถึง 256 โบน
2. ซอฟต์แวร์ที่มีอยู่ใน Direct3D เวอร์ชัน 8 ยังไม่มีประสิทธิภาพในการทำงานดีพอ
3. ไม่สามารถทำงานกับ per-pixel lighting ได้ดี
4. จำกัดจำนวนโบนต่อเวอร์เท็กซ์ไม่เกิน 4 โบน

10.1.2.2 การเขียนโปรแกรมเวอร์เท็กซ์เชดเดอร์ (Vertex Shader)

ข้อดี

1. มีฮาร์ดแวร์ที่สนับสนุนการทำงาน เช่น GeForce3
2. ซอฟต์แวร์ที่มีประสิทธิภาพสูงกว่า
3. ไม่มีการจำกัดจำนวนโบนต่อเวอร์เท็กซ์
4. ทำงานได้อย่างดีกับ per-pixel lighting

ข้อเสีย

1. มีความซับซ้อนสูงมาก
2. มีความจำเป็นที่จะต้องแบ่งกลุ่มของโพรซีเจอร์ตามจำนวน โบน

ตัวอย่าง โค้ดในการทำสกินนิ่งจำนวน 2 โบนต่อ 1 เวอร์เท็กซ์ด้วยเวอร์เท็กซ์เชดเดอร์จะเห็นว่ามีความซับซ้อนมากกว่าวิธี Indexed Vertex Blending มาก

```
//load first index
```

```
mov a0.x, V_INDICES.x
```

```
//transform position by first bone, store in r0
```

```
dp4 r0.x, V_POSITION, c[a0.x + CV_BONESTART + 0]
```

```
dp4 r0.y, V_POSITION, c[a0.x + CV_BONESTART + 1]
```

```
dp4 r0.z, V_POSITION, c[a0.x + CV_BONESTART + 2]
```

```
//transform normal by first bone, store in r2
```

```
dp3 r2.x, V_NORMAL, c[a0.x + CV_BONESTART + 0]
```

```
dp3 r2.y, V_NORMAL, c[a0.x + CV_BONESTART + 1]
```

```
dp3 r2.z, V_NORMAL, c[a0.x + CV_BONESTART + 2]
```

```
//load second index
```

```
mov a0.x, V_INDICES.y
```

```
//transform position by second bone, store in r1
```

```
dp4 r1.x, V_POSITION, c[a0.x + CV_BONESTART + 0]
```

```
dp4 r1.y, V_POSITION, c[a0.x + CV_BONESTART + 1]
```

```
dp4 r1.z, V_POSITION, c[a0.x + CV_BONESTART + 2]
```

```
//transform normal by second bone, store in r3
```

```
dp3 r3.x, V_NORMAL, c[a0.x + CV_BONESTART + 0]
```

```
dp3 r3.y, V_NORMAL, c[a0.x + CV_BONESTART + 1]
```

```
dp3 r3.z, V_NORMAL, c[a0.x + CV_BONESTART + 2]
```

```
//blend between r0 and r1 -- the positions
```

```
mul r0, r0, V_WEIGHT.x
```

```
mad r1, r1, V_WEIGHT.y, r0
```

```
//blend between r2 and r3 -- the normals
```

```
mul r2, r2, V_WEIGHT.x
```

```
mad r3, r3, V_WEIGHT.y, r7
```


10.2 การตรวจสอบการชนของวัตถุ (Collision Detection)

10.2.1 วัตถุตรวจสอบการชนพื้นฐาน

ในโครงการนี้การตรวจสอบการชนของวัตถุจะใช้การตรวจสอบกับวัตถุการชนพื้นฐาน 4 แบบ ดังนี้

1. แบบพื้น (Ground type)
2. แบบกำแพง (Wall type)
3. แบบทรงกระบอก (Cylinder type)
4. แบบทรงกลม (Sphere type)

10.2.2 การตรวจสอบการชนกับวัตถุแบบพื้น

โดยปกติแล้วตัวละครที่ปรากฏอยู่ในเกมมักจะเดินอยู่บนพื้นตามแรงดึงดูดของโลก ซึ่งเราสามารถจำลองคุณสมบัตินี้ได้โดยสร้างข้อมูลที่ใช้แทนพื้นขึ้นมา ซึ่งตัวละครในเกมจะต้องเลื่อนหรือไถไปตามพื้นผิวของวัตถุตรวจสอบการชนนี้

ข้อมูลที่ใช้ในสร้างวัตถุตรวจสอบการชนแบบพื้น

- จุด 3 จุดซึ่งประกอบกันเป็นรูปสามเหลี่ยม

ข้อมูลที่มีอยู่ในวัตถุตรวจสอบการชนแบบพื้น

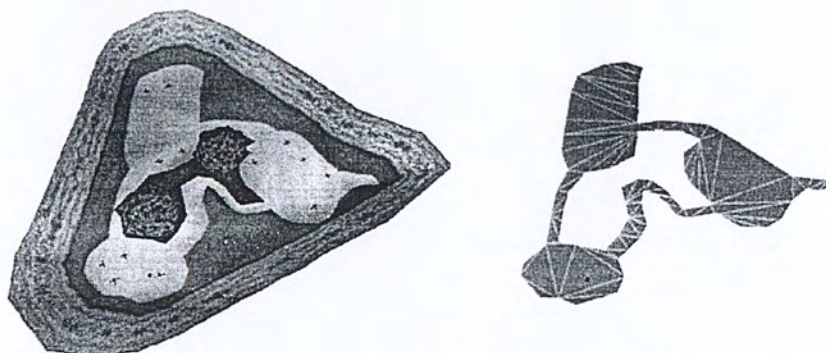
- สมการเส้นตรงซึ่งแสดงขอบเขตของรูปสามเหลี่ยม
- ค่า Boolean ซึ่งแสดงว่าผลลัพธ์ที่ได้จากการแทนค่าในสมการเส้นตรงแบบใดคือ จุดที่อยู่ในรูปสามเหลี่ยม
- สมการระนาบของรูปสามเหลี่ยมนั้น
- เมตริกซ์อ้างอิง

ข้อมูลที่ถูกใช้ในการตรวจสอบ

- ค่าตำแหน่งของวัตถุ

ผลลัพธ์ที่ได้จากการตรวจสอบการชนแบบพื้น

- ความสูงในแนวตั้งฉากกับรูปสามเหลี่ยมที่พบว่าตำแหน่งวัตถุอยู่ภายใน
- สมการระนาบของรูปสามเหลี่ยมนั้น



รูปที่ 10-7 รูปแสดงวัตถุตรวจสอบการชนแบบพื้นที่ได้จากฉาก

10.2.3 การตรวจสอบการชนกับวัตถุแบบกำแพง

ตรงข้ามกับวัตถุแบบพื้น วัตถุแบบกำแพงจะวางตัวในแนวแกนที่เราสนใจ ซึ่งโดยปกติคือ แกน y นั่นคือ วัตถุชนิดนี้จะเป็นตัวขวางกั้นคล้ายกับกำแพงในชีวิตประจำวัน

ข้อมูลที่ใช้ในสร้างวัตถุตรวจสอบการชนแบบพื้น

- จุด 3 จุดซึ่งประกอบกันเป็นรูปสามเหลี่ยม

ข้อมูลที่มีอยู่ในวัตถุตรวจสอบการชนแบบพื้น

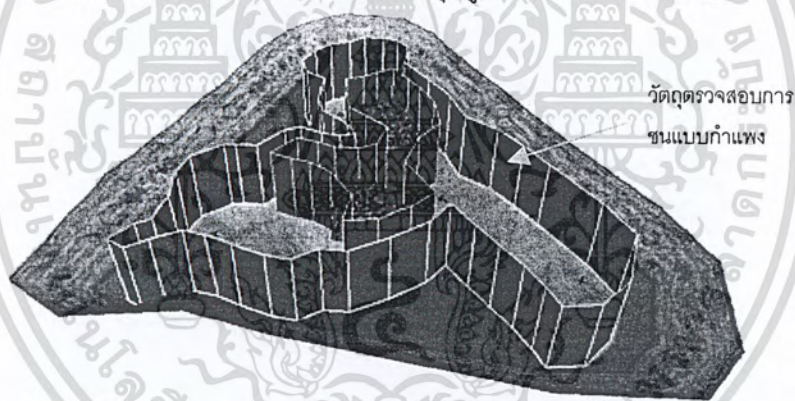
- ตำแหน่งอ้างอิง
- ความสูง
- สมการเส้นตรง 1 สมการ ซึ่งแทนเส้นตรงที่เกิดจากกำแพงเมื่อมองลงมาจากแกน y
- เมตริกซ์อ้างอิง

ข้อมูลที่ถูกใช้ในการตรวจสอบ

- วัตถุตรวจสอบการชนแบบทรงกระบอก

ผลลัพธ์ที่ได้จากการตรวจสอบการชนแบบพื้น

- เวกเตอร์ตั้งฉากกับสมการเส้นตรงที่ตัดผ่านจุดศูนย์กลางของทรงกระบอก



รูปที่ 10-8 รูปแสดงวัตถุตรวจสอบการชนแบบกำแพงที่ได้จากฉาก

10.2.4 การตรวจสอบการชนกับวัตถุแบบทรงกระบอก

วัตถุตรวจสอบการชนแบบทรงกระบอกเป็นตัวกำหนดขอบเขตของตัวละคร วัตถุตรวจสอบนี้ใช้ในการตรวจสอบเพื่อป้องกันการซ้อนกันของตัวละคร และสามารถประยุกต์ใช้งานอื่นๆ ได้อีกมากมาย

ข้อมูลที่ใช้ในสร้างวัตถุตรวจสอบการชนแบบพื้น

- ความสูง, รัศมี
- ตำแหน่งอ้างอิง

ข้อมูลที่มีอยู่ในวัตถุตรวจสอบการชนแบบพื้น

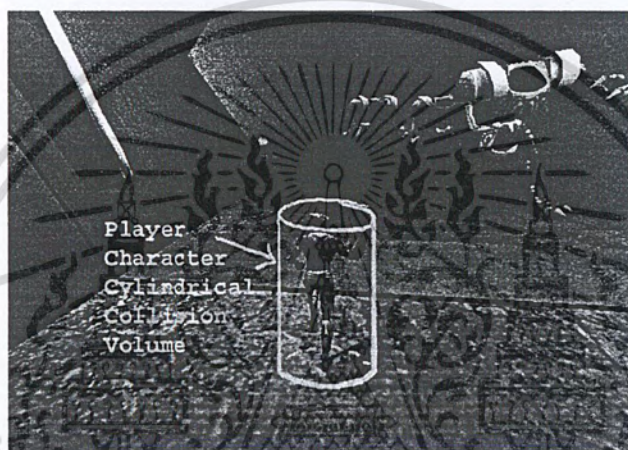
- ตำแหน่งอ้างอิง
- ความสูงและรัศมี
- เมตริกซ์อ้างอิง

ข้อมูลที่ถูกใช้ในการตรวจสอบ

- วัดตรวจสอบการชนแบบทรงกระบอก
- วัดตรวจสอบการชนแบบก้ำแพง
- วัดตรวจสอบการชนแบบทรงกลม
- ตำแหน่งอ้างอิงใช้ตรวจสอบกับวัดตรวจสอบการชนแบบพื้น

ผลลัพธ์ที่ได้จากการตรวจสอบการชนแบบพื้น

- เวกเตอร์ตั้งฉากซึ่งตัดผ่านจุดศูนย์กลางของวัดตรวจสอบการชนแบบทรงกระบอกและวัดตรวจสอบการชนแบบทรงกลม



รูปที่ 10-9 รูปแสดงวัดตรวจสอบการชนแบบทรงกระบอกรอบตัวละคร

10.2.5 การตรวจสอบการชนกับวัตถุแบบทรงกลม

ในกรณีที่ต้องการการตรวจสอบที่มีความละเอียดยิ่งขึ้น วัดตรวจสอบการชนแบบทรงกลมจะตอบสนองความต้องการในส่วนนี้ได้ เนื่องจากการใช้วัดตรวจสอบการชนชนิดนี้เหมาะสมจะได้ผลลัพธ์ที่มีความใกล้เคียงกับความเป็นจริงที่สุด

ข้อมูลที่ใช้ในสร้างวัดตรวจสอบการชนแบบพื้น

- รัศมี
- ตำแหน่งอ้างอิง

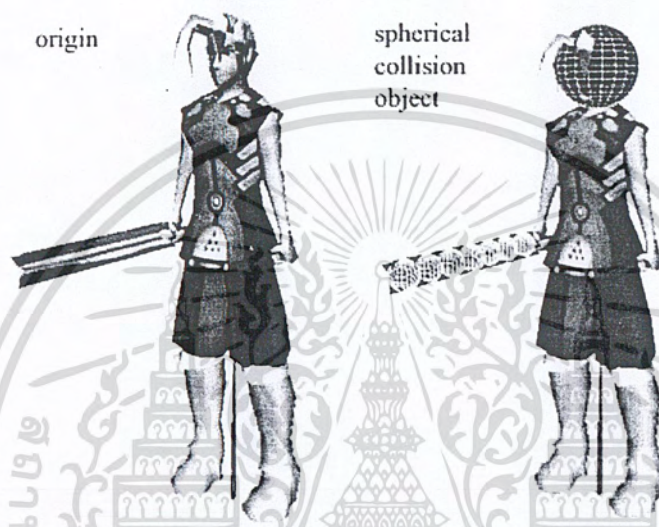
ข้อมูลที่มีอยู่ในวัดตรวจสอบการชนแบบพื้น

- ตำแหน่งอ้างอิง
- รัศมี
- เมตริกซ์อ้างอิง

ข้อมูลที่ถูกใช้ในการตรวจสอบ

- วัดตรวจสอบการชนแบบทรงกระบอก
- วัดตรวจสอบการชนแบบก้ำแพง

- วัตถุตรวจสอบการชนแบบทรงกลม
- ผลลัพธ์ที่ได้จากการตรวจสอบการชนแบบพื้น
- เวกเตอร์ตั้งฉากซึ่งตัดผ่านจุดศูนย์กลางของวัตถุตรวจสอบการชนแบบทรงกระบอกและวัตถุตรวจสอบการชนแบบทรงกลม
 - เวกเตอร์ซึ่งตั้งฉากกับวัตถุตรวจสอบการชนแบบกำแพงซึ่งตัดผ่านจุดศูนย์กลางของวัตถุตรวจสอบการชนแบบทรงกลม

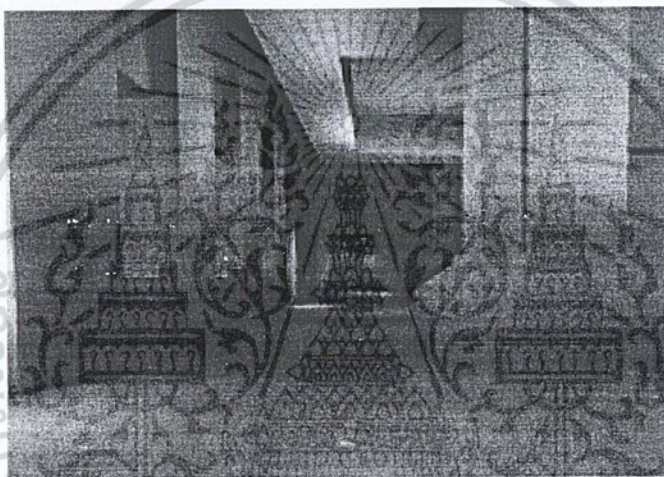


รูปที่ 10-10 รูปแสดงวัตถุตรวจสอบการชนแบบทรงกลมที่สร้างขึ้นในส่วนที่ต้องการบนตัวละคร

10.3 Light Map (Dark Map)

เป็นวิธีการที่นิยมใช้ในการเพิ่มความสมจริงให้กับฉากในเรื่องของแสงและเงา เนื่องจากใช้การประมวลผลที่น้อยกว่าและมีวิธีการที่ไม่สลับซับซ้อนมาก เนื่องจากวิธีนี้เป็นการเพิ่มเทกเจอร์เข้าไปอีก 1 ชั้น ซึ่งเทกเจอร์ชั้นที่เพิ่มเข้ามานี้จะเป็นตัวกำหนดความสว่างของวัตถุที่ปรากฏขึ้นมาด้วย ซึ่งการที่จะให้ผลลัพธ์มีความถูกต้องจะต้องกำหนดสถานะชั้นเทกเจอร์ที่เพิ่มเข้ามา ดังตัวอย่าง

```
pDevice->SetTextureStageState( 1, COLORARG1, D3DTA_TEXTURE);
pDevice->SetTextureStageState( 1, COLORARG2, D3DTA_CURRENT);
pDevice->SetTextureStageState( 1, COLOROP, D3DTOP_MODULATE);
pDevice->SetTexture(1, pddsLightMap);           // set lightmap texture to be used
```



รูปที่ 10-11 รูปแสดงการใช้ light map เพื่อเพิ่มแสงเงาให้กับฉาก

บทที่ 11

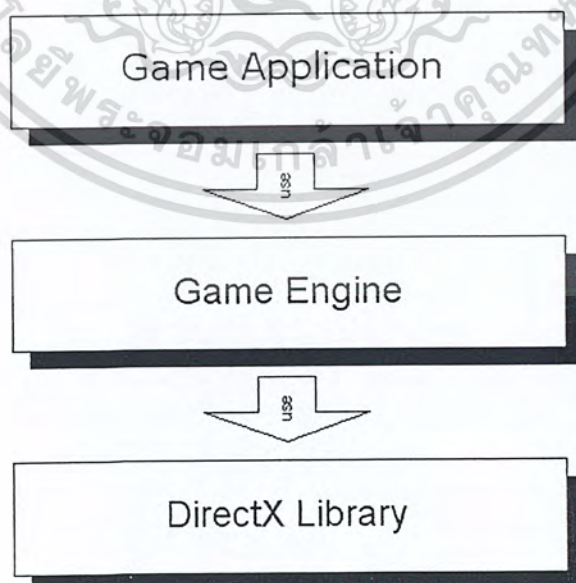
แนวความคิดและการออกแบบ

11.1 ความต้องการ (Requirement)

ในขั้นตอนแรกจะต้องกำหนดคุณลักษณะของแอปพลิเคชันที่จะสร้างขึ้น ซึ่งในโครงการนี้ได้กำหนดคุณสมบัติของแอปพลิเคชันขั้นต้นดังนี้

1. เป็นแอปพลิเคชันประเภทเกมที่แสดงผลด้วยกราฟิก 3 มิติ
2. กราฟิก 3 มิติที่แสดงเป็นตัวละครในเกมจะต้องมีความสามารถในการทำแอนิเมชัน
3. มีความสามารถในการตรวจสอบการชนของวัตถุเพื่อความจริง
4. เล่นเพลงจากไฟล์ในรูปแบบของเวฟในส่วนของเอฟเฟ็กต์และในรูปแบบ MP3 ในส่วนของเสียงเพลง
5. เล่นไฟล์วิดีโอได้
6. สามารถใช้งานอุปกรณ์อินพุตได้หลากหลายทั้งเมาส์, คีย์บอร์ด และจอยสติค

โครงการนี้ทำการพัฒนาแอปพลิเคชันประเภทเกมสับนไลบรารีของ DirectX แต่เนื่องจากโคเร็กเอ็กซ์ในขณะนี้ (เวอร์ชัน 8) ประกอบขึ้นด้วยอินเทอร์เฟซและเมทอดที่ทำงานในชั้นพื้นฐานเท่านั้น เช่นการสร้างบัฟเฟอร์สำหรับเก็บข้อมูลเวอร์เทกซ์, การคัดลอกข้อมูลไปยังฟรอนต์บัฟเฟอร์ เป็นต้น ดังนั้นเพื่อที่การพัฒนาซอฟต์แวร์เกมจะมีความสะดวกรวดเร็วและมีความผิดพลาดน้อยลงแล้วจำเป็นที่จะต้องนำเอาอินเทอร์เฟซและเมทอดที่ทำงานพื้นฐานเหล่านี้มารวมกันสร้างเป็นคลาสเพื่อทำงานอย่างใดอย่างหนึ่งและทำให้อยู่ในรูปที่ใช้งานง่ายโดยจำกัดขอบเขตการเข้าถึงอินเทอร์เฟซและเมทอดและกำหนดวิธีการใช้งานบางอย่าง ซึ่งคลาสที่สร้างขึ้นในส่วนนี้เรียกว่า เกมเอนจิน (Game Engine) นั่นเอง หลังจากนั้นจึงใช้คลาสที่สร้างขึ้นมานี้ในการพัฒนาแอปพลิเคชันเกม



รูปที่ 11-1 ลำดับชั้นในการพัฒนาเกม

นอกจากนั้น โมเดลและภาพ 2 มิติที่จะนำมาใช้ในการพัฒนาโปรแกรมก็มีความจำเป็นที่จะต้องกำหนดไว้ ซึ่งข้อมูลในไฟล์ที่ได้จากแอปพลิเคชันสร้างโมเดล 3 มิติ และไฟล์ 2 มิติที่มี มักจะไม่ตรงกับความต้องการของแอปพลิเคชันเกม

ความต้องการรูปแบบของไฟล์ที่เก็บ โมเดล 3 มิติที่ใช้ในแอปพลิเคชันเกมสำหรับโครงการนี้

1. มีข้อมูลเอ็นิเมชันและเก็บข้อมูลเอ็นิเมชันแต่ละท่วงท่าแยกจากกันโดยเด็ดขาดสามารถใช้งานได้ง่าย
2. ข้อมูลเอ็นิเมชันที่ต้องการจะต้องเป็นเอ็นิเมชันในรูปแบบที่ใช้กับอินเด็กซ์เวอร์เท็กซ์เบลนดิงได้ คือ จะต้องเป็นข้อมูลที่มีค่าน้ำหนัก, ค่าเมตริกซ์และค่าชื่อของโบนที่เกี่ยวข้อง
3. ข้อมูลเวอร์เท็กซ์ของโมเดลจะต้องอยู่ในรูปแบบที่สามารถโหลดเข้าไปใช้งานได้ทันทีโดยไม่ต้องมีการประมวลผลใดๆ เกิดขึ้นอีก
4. จะต้องเป็นไฟล์แบบไบนารีเนื่องจากไฟล์ในรูปแบบของเท็กซ์ไฟล์จะเสียสโห้ยในการแปลงจากสตริงเป็นข้อมูล
5. ข้อมูลเมตที่เรียลเป็นข้อมูลในระดับโมเดลไม่ใช่ข้อมูลในระดับเฟรม

ความต้องการรูปแบบของไฟล์ที่เก็บข้อมูล 2 มิติที่ใช้ในแอปพลิเคชันเกมสำหรับโครงการนี้

1. จะต้องประกอบด้วยบิตข้อมูลที่แสดงค่าความโปร่งใส
2. อยู่ในรูปแบบที่สามารถเขียนข้อมูลจากไฟล์ลงบัฟเฟอร์ได้อย่างรวดเร็วไม่ต้องใช้การประมวลผลใดๆ อีก (เนื่องจากไฟล์เท็กซ์เจอร์จะมีขนาดไม่ใหญ่อยู่แล้วโดยปกติจะมีขนาด 256*256 พิกเซล ขนาด 32 บิต หรือ 256 kBytes)

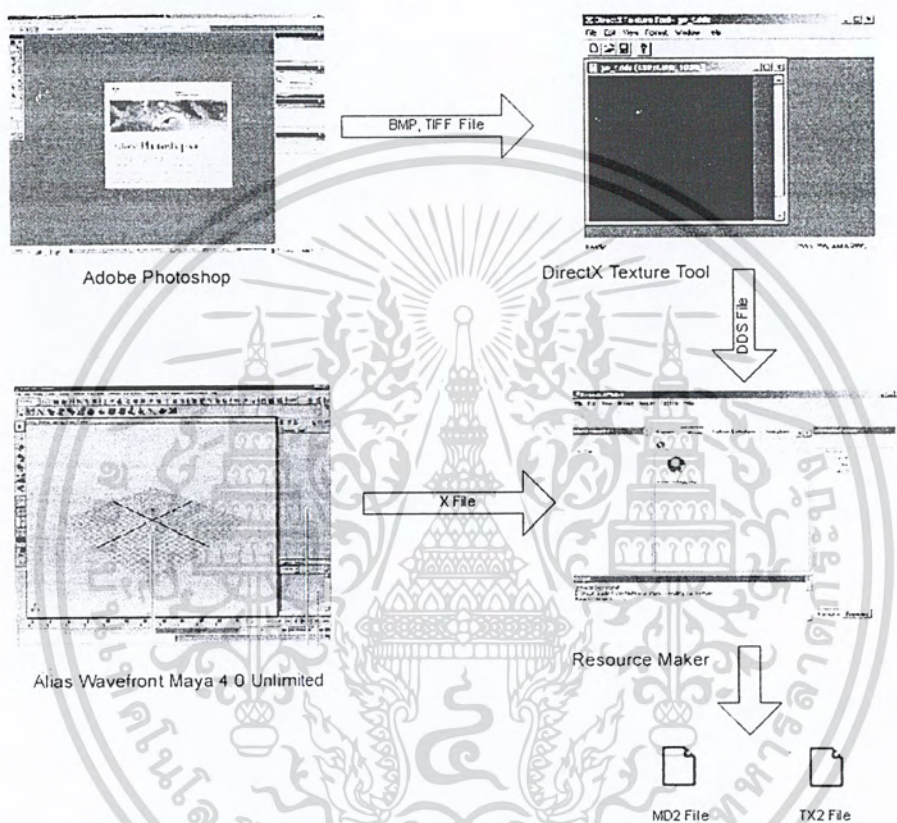
ซึ่งจากการค้นหารูปแบบไฟล์ต่างๆ ในปัจจุบันพบว่า ไฟล์โมเดล 3 มิติ ส่วนมากจะไม่มีข้อมูลเอ็นิเมชัน และบางส่วนของที่มีข้อมูลเอ็นิเมชัน แต่มักจะเป็นเอ็นิเมชันในแบบเวอร์เท็กซ์เบลนดิง ซึ่งไม่ใช่ในโครงการนี้ หลังจากตรวจสอบคุณสมบัติของรูปแบบไฟล์ชนิดต่างๆ แล้วพบว่า ไฟล์ในรูปแบบของ .X มีคุณสมบัติใกล้เคียงกับรูปแบบของไฟล์ที่ต้องการมากที่สุด แต่มีบางส่วนซึ่งไม่ตรงกันความต้องการ ถึงแม้ว่าใน SDK ของไคลร์เอ็กซ์จะมีไลบรารีในส่วนที่จัดการอ่านและเขียนข้อมูลอยู่แล้วซึ่งทำให้ง่ายในการใช้งาน แต่การเข้าถึงข้อมูลต่างๆ ที่อยู่ในเวอร์เท็กซ์บัฟเฟอร์จะถูกจำกัด และลักษณะการเก็บข้อมูลเอ็นิเมชันไม่ได้เก็บข้อมูลแต่ละท่วงท่าแยกจากกันทำให้มีปัญหาในการสร้างโมเดล 3 มิติ

ดังนั้น จากความต้องการต่างๆ ข้างต้น โครงการนี้จึงแบ่งออกเป็นองค์ประกอบย่อยๆ 3 ส่วนดังนี้

1. การพัฒนาโปรแกรมสำหรับอ่านข้อมูลจากไฟล์ในรูปแบบต่างๆ ที่ได้จากแอปพลิเคชันสร้างโมเดล 3 มิติ และเขียนข้อมูลลงในแบบที่ต้องการ
2. การพัฒนาเกมเอนจินซึ่งจะให้ป็นไลบรารีในการเขียนแอปพลิเคชันเกม
3. ใช้ไลบรารีจากเกมเอนจินในการพัฒนาแอปพลิเคชันเกม

สรุปขั้นตอนในการสร้างไฟล์โมเดลและเทกเจอร์ที่จะใช้ในการพัฒนาแอปพลิเคชันเกมในโครงการนี้

เริ่มต้นจากการขึ้นโมเดลในแอปพลิเคชันสร้างโมเดล 3 มิติ เช่น Maya, 3D Studio Max เป็นต้น แล้วส่งข้อมูลออกมาในรูปของ X ไฟล์ ส่วนภาพที่จะใช้เป็นเทกเจอร์สร้างขึ้นโดยใช้โปรแกรม Adobe® Photoshop, คาว์นโหลดจากอินเทอร์เน็ตหรือสแกนเข้ามาจากเครื่องสแกน ซึ่งภาพที่ได้มาส่วนใหญ่มักจะไม่ตรงกับความต้องการจึงต้องมาแต่งภาพใน Adobe® PhotoShop ก่อน หลังจากนั้นนำข้อมูลในรูปของไฟล์ต่างๆ ที่ได้มาแปลงให้อยู่ในรูปแบบที่จะใช้งานเก็บเอาไว้ ส่วนของแอปพลิเคชันเขียนขึ้นโดยใช้ไลบรารีในเกมเอนจินที่เขียนขึ้นมา ดังแผนภาพ



รูปที่ 11-2 ขั้นตอนการสร้างรีซอร์สที่ใช้ในการพัฒนาเกม

11.2 การออกแบบ

11.2.1 การพัฒนาแอปพลิเคชันที่จัดการกับข้อมูลจากแอปพลิเคชันสร้างโมเดล 3 มิติ

ในอุตสาหกรรมการพัฒนาซอฟต์แวร์เกม ขั้นตอนนี้คือการพัฒนาทูล (Tool) ซึ่งจะช่วยให้การพัฒนาซอฟต์แวร์เกมเป็นไปได้อย่างรวดเร็วและมีประสิทธิภาพ ซึ่งการพัฒนาทูลไม่ได้จำกัดว่าจะต้องเป็นโปรแกรมที่มีประสิทธิภาพมากมายนัก จึงควรเลือกพัฒนาด้วยภาษาโปรแกรมที่ง่ายและสามารถเขียนแอปพลิเคชันแบบ GUI ได้เป็นอย่างดี เช่น Visual Basic, Java เป็นต้น ซึ่งในโครงการนี้ได้เลือกใช้ Visual C# ซึ่งแม้จะอยู่ในขั้นเบต้าแต่ก็สามารถตอบสนองความต้องการของผู้พัฒนาโครงการได้

เหตุผลที่ใช้ Microsoft® Visual® C# ในการพัฒนามีดังนี้

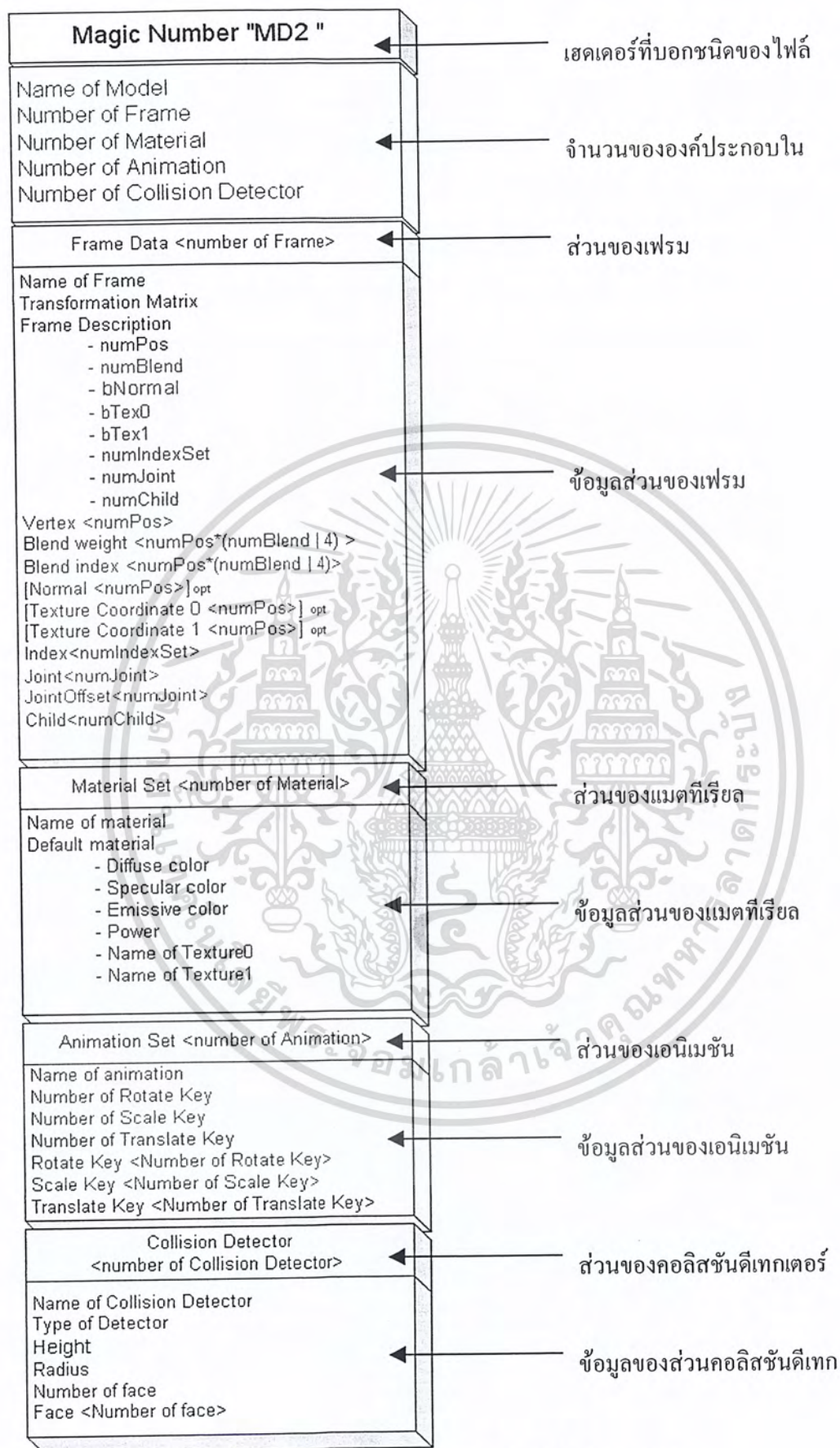
1. เป็นภาษาที่ทำงานด้าน GUI ได้เป็นอย่างดี เนื่องจากมีแพลตฟอร์มที่สนับสนุนงานด้าน GUI ที่มีประสิทธิภาพ
2. เป็นภาษาที่มีสิ่งแวดล้อมในการพัฒนา (IDE) คล้ายคลึงกับการใช้งาน Visual C++ ซึ่งใช้ในการพัฒนาเป็นหลักในโครงการนี้
3. มีโครงสร้างภาษาเป็น OOP คล้ายคลึงกับภาษาจาวา และภาษา C++ ทำให้ใช้เวลาในการเรียนรู้ น้อย สามารถใช้งานได้ทันที

หน้าที่ของแอปพลิเคชันแปลงข้อมูลจากไฟล์ในรูปแบบต่างๆ

1. อ่านข้อมูลจากไฟล์ในรูปแบบของไฟล์นามสกุล X แล้วนำข้อมูลเวอร์เท็กซ์มาจัดเรียงให้อยู่ในรูปแบบที่สามารถใช้งานได้ทันทีโดยโคเรกทรีตี้ ดังรูปที่
2. เพิ่มข้อมูลในส่วนของแต่ละตำแหน่งของเทกเจอร์อีก 1 ชั้น โดยข้อมูลจะมาจากอีกโมเดลหนึ่ง ในกรณีที่ต้องการใช้งานมัลติเทกเจอร์
3. เพิ่มข้อมูลในส่วนของเอ็นิเมชัน โดยข้อมูลเอ็นิเมชันจะนำมาจากอีกโมเดลหนึ่ง
4. ใส่ชื่อของเทกเจอร์ที่ต้องการให้กับแมตทีเรียล (ไม่ใช่ชื่อของเทกเจอร์ที่มากับไฟล์เนื่องจากไฟล์เทกเจอร์ที่ใช้ในแอปพลิเคชัน 3 มิติมีรูปแบบไม่ตรงกับที่จะใช้ในเกม)
5. เขียนข้อมูลจากไฟล์ในรูปแบบ MD2 ซึ่งกำหนดขึ้นมา

ขั้นตอนของข้อมูลจากไฟล์รูปแบบ X ที่ต้องการการประมวลผล

1. นำข้อมูลมาประกอบเป็นเวอร์เท็กซ์
2. ปรับค่าอินเด็กซ์ที่มากกว่า 3 ค่า ให้เป็น 3 ค่า และลบค่าอินเด็กซ์ที่มีน้อยกว่า 3 ค่าทิ้งไป
3. นำแมตทีเรียลมาเป็นคุณสมบัติในระดับ โมเดล (เนื่องจากใน X File แมตทีเรียลเป็นคุณสมบัติระดับเฟรม)
4. นำข้อมูลที่อยู่ในส่วนของ SkinWeights มาจัดเรียงจากมากไปน้อย เก็บชื่อของ Joint ที่เกี่ยวข้องไว้เพื่อหาค่าอินเด็กซ์ในการอ้างอิงจากระดับ โมเดล



รูปที่ 11-3 ลำดับการเก็บข้อมูลในไฟล์รูปแบบ MD2

11.2.2 การพัฒนาเกมเอนจิน

ความต้องการ (Requirement)

แอปพลิเคชันที่ใช้งานเอนจินจะต้องใช้งานฟังก์ชันและออปเจกต์ได้อย่างง่าย ๆ

กล่าวอย่างง่าย ๆ ในโครงงานนี้มีหมายถึง สามารถมองข้อมูลของโมเดลเป็นวัตถุที่สามารถแสดงผลได้ทันทีโดยแอปพลิเคชันไม่ต้องคำนึงถึงวิธีการที่จะทำให้มันแสดงผลเลย

การออกแบบ (Design)

เกมเอนจิน จะสร้างขึ้นโดยแบ่งเป็นโมดูลที่ทำหน้าที่ในด้านที่แตกต่างกันเฉพาะในส่วนที่ต้องการใช้ เช่น งานด้านแสดงผลกราฟิก, งานด้านการควบคุมเสียง, งานด้านการควบคุมการเล่นไฟล์วิดีโอ และงานด้านการจัดการอุปกรณ์อินพุต โดยคลาสที่ประกอบอยู่ในแต่ละโมดูลจะสร้างขึ้นโดยการรวบรวมอินเทอร์เฟซและเมทอดที่ทำงานพื้นฐานในไคเร็กเอ็กซ์ให้เป็นคลาสที่ทำงานอย่างใดอย่างหนึ่งและอยู่ในรูปที่สามารถใช้งานได้ง่ายที่สุด

ในโครงงานนี้ได้แบ่งส่วนของเอนจินออกเป็น โมดูล ดังนี้

11.2.2.1 ส่วนของการแสดงผลกราฟิก

ควบคุมการแสดงผลทั้งในแบบของ 2 มิติและ 3 มิติ โดยทำงานติดต่อกับการ์ดแสดงผลผ่านทางไคเร็กทรีดี แต่เนื่องจากไคเร็กทรีดีเป็น API ขั้นพื้นฐาน ซึ่งประกอบขึ้นด้วยอินเทอร์เฟซและเมทอดที่ทำงานในระดับองค์ประกอบย่อย จึงมีความจำเป็นที่จะต้องรวบรวมและจัดการกับอินเทอร์เฟซและเมทอดเหล่านี้ให้ใช้งานได้ง่ายและมีความผิดพลาดน้อยโดยการนำหลักการของ OOP เข้ามาใช้

เอนจินในส่วนของการแสดงผลจะเป็นการนำเอาอินเทอร์เฟซและเมทอดที่มีอยู่ในไคเร็กทรีดีมาสร้างเป็นคลาส เพื่อที่จะใช้ในการทำงานเฉพาะอย่าง เช่น การโหลดโมเดล, การเตรียมบัฟเฟอร์สำหรับข้อมูลที่จะใช้ในการเรนเดอร์ เป็นต้น

คุณสมบัติที่ต้องการ (Requirement)

- ด้านการทำงาน
 1. มีความสามารถในการทำภาพแอนิเมชันโมเดล 3 มิติ
 2. มีความสามารถในการตรวจสอบการชนของวัตถุ
 3. มีความสามารถในการแสดงผลภาพ 2 มิติและการทำแอนิเมชันภาพ 2 มิติและเทกเจอร์ (Sprite)
 4. มีประสิทธิภาพในการทำงานที่ดี (ไม่ต่ำกว่า 30 FPS ในสภาพแวดล้อมที่กำหนด)
- ด้านการใช้งาน

สามารถใช้งานฟังก์ชันและคลาสในการทำงานต่างๆ ได้ง่ายดาย เช่น ต้องการแสดงผลโมเดล 3 มิติ 1 ชิ้น มีขั้นตอนการทำงาน คือ โหลดข้อมูลจากไฟล์และสั่งให้แสดงผลเท่านั้น

การออกแบบ

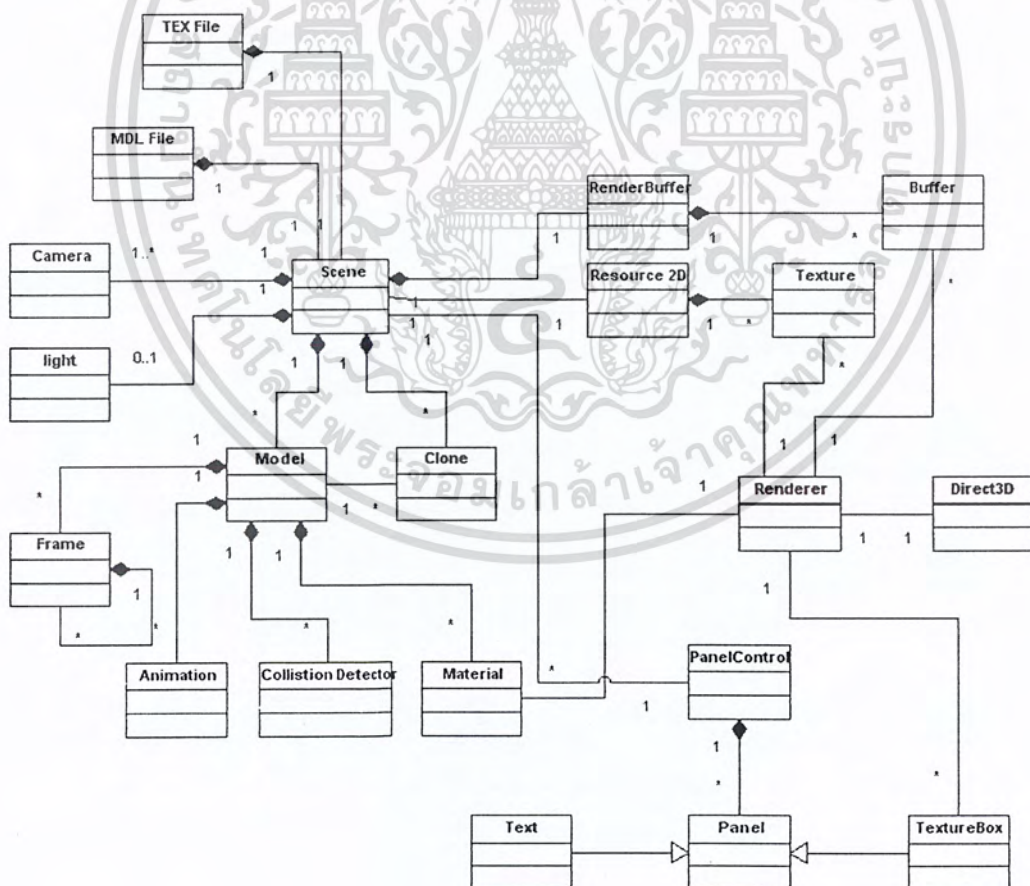
1. กำหนดวัตถุ

เนื่องจากแอปพลิเคชันของเราทำงานกับข้อมูลทั้งหมดไม่มีส่วนที่ติดต่อกับฮาร์ดแวร์โดยตรง (ได้เรียกสามมิติทำงานในส่วนที่ติดต่อกับฮาร์ดแวร์ให้แล้ว) การกำหนดวัตถุจึงใช้ข้อมูลที่เก็บเป็นตัวกำหนด

วัตถุ	ข้อมูล	ระยะเวลาในการเก็บ
เฟรม (Frame)	เก็บข้อมูลเวอร์เท็กซ์ซึ่งใช้ในการกำหนดตำแหน่งของจุด (position), ค่าของเวกเตอร์ตั้งฉาก, การเรียงตัวของจุด, ค่าตำแหน่งของเทกเจอร์ เป็นต้น	ช่วงเวลาที่มีการแสดงโมเดลนั้นในเกม
แอนิเมชัน (Animation)	เก็บข้อมูลสำหรับการกำหนดค่าการเคลื่อนไหว, การหมุน, การย่อขยาย และค่าเวลา ที่จะใช้ในการคำนวณการเคลื่อนไหวของโมเดล	ช่วงเวลาที่โมเดลนั้นในเกม
คอลลิชันดีเทกเตอร์ (Collision Detector)	เก็บข้อมูลสำหรับการตรวจสอบการชนของวัตถุ เช่น ขอบเขต, ชนิดของการตรวจสอบ เป็นต้น	ช่วงเวลาที่โมเดลนั้นในเกม
เมตทีเรียล (Material)	เก็บข้อมูลของเมตทีเรียลที่ใช้ในการกำหนดลักษณะการคำนวณทางแสงของโมเดล	ช่วงเวลาที่โมเดลนั้นในเกม
เทกเจอร์ (Texture)	เก็บข้อมูลของภาพ 2 มิติและข้อมูลที่จะใช้แสดงแอนิเมชันของภาพ 2 มิติ	ช่วงเวลาที่ต้องการแสดงภาพ 2 มิติ
ไดเรกทรีดี (Direct3D)	เก็บสถานะและคุณสมบัติต่างๆ ที่ได้กำหนดให้กับไดเรกทรีดี	ช่วงที่แอปพลิเคชันทำงาน
โมเดล (Model)	เก็บข้อมูลที่แสดงความสัมพันธ์ระหว่างแต่ละเฟรม และความสัมพันธ์ระหว่างแอนิเมชัน, เมตทีเรียล และคอลลิชันดีเทกเตอร์	ช่วงเวลาที่มีการแสดงโมเดลนั้นในเกม
โคลน (Clone)	เก็บข้อมูลที่อ้างอิงไปยังข้อมูลของ โมเดล และข้อมูลสำหรับแต่ละโมเดลที่จะถูกแสดง	เมื่อต้องการแสดงโมเดลแต่ละโมเดล
พานเนลคอนโทรล (PanelControl)	เก็บข้อมูลและควบคุมการแสดงผลภาพ 2 มิติ	ช่วงเวลาที่แอปพลิเคชันทำงาน
พานเนล (Panel)	เก็บข้อมูลที่แสดงขอบเขตพื้นที่ในการแสดงผลภาพ 2 มิติ	ช่วงเวลาที่ต้องการแสดงผลภาพ 2 มิติ
บัฟเฟอร์ (Buffer)	เก็บข้อมูลของเวอร์เท็กซ์ที่สามารถส่งไปเรนเดอร์ได้โดยไดเรกทรีดี	ช่วงเวลาที่ต้องการแสดงโมเดลนั้นในเกม
เรนเดอร์เรอร์ (Renderer)	เก็บข้อมูลและจัดการให้มีการเรนเดอร์อย่างถูกต้องและมีประสิทธิภาพ	ช่วงเวลาที่แอปพลิเคชันทำงาน

เรนเดอร์บัฟเฟอร์ (RenderBuffer)	เก็บข้อมูลที่แสดงสถานะและควบคุมการทำงานของบัฟเฟอร์	ช่วงเวลาที่มีการใช้งานบัฟเฟอร์
กล้อง (Camera)	เก็บข้อมูลสถานะของกล้อง เช่น ตำแหน่ง, โฟกัส เป็นต้น	ช่วงเวลาที่ต้องการใช้กล้อง
แสง (Light)	เก็บข้อมูลสถานะของแสง	ช่วงเวลาที่ต้องการใช้แสง
ฉาก (Scene)	เก็บข้อมูลความสัมพันธ์ของวัตถุต่างๆ ที่จะปรากฏขึ้นในการแสดงผลหนึ่งๆ เช่น โมเดล(โคลน), กล้อง, แสง และ ภาพ 2 มิติ เป็นต้น	ช่วงเวลาที่ต้องการแสดงผลภาพ
ไฟล์โมเดล (Model File)	เก็บข้อมูลและฟังก์ชันที่ใช้ในการโหลดไฟล์โมเดล	ช่วงเวลาที่ต้องการโหลดไฟล์โมเดล
ไฟล์เทกเจอร์ (Texture File)	เก็บข้อมูลและฟังก์ชันที่ใช้ในการโหลดไฟล์เทกเจอร์	ช่วงเวลาที่ต้องการโหลดไฟล์เทกเจอร์

ตารางที่ 11-1 วัตถุที่ได้จากการแยกแยะโดยใช้ข้อมูลที่เก็บสำหรับส่วนแสดงผล



รูปที่ 11-4 คลาสไดอะแกรมในส่วนของการฝึก

รายละเอียดในคลาสที่มีความสำคัญ

Direct3D (class NG3D ใน ng3d.h)

ทำหน้าที่ในส่วนที่เกี่ยวข้องทั้งหมด เช่น การระบุอุปกรณ์ที่มีอยู่ในเครื่องคอมพิวเตอร์, การตรวจสอบความสามารถของอุปกรณ์ และการสร้างอุปกรณ์สำหรับการแสดงผล ซึ่งคลาสนี้ผู้พัฒนาได้ใช้เทคนิคเพื่อให้สามารถสร้างคลาสนี้ได้เพื่อหนึ่งคลาสในหนึ่งแอปพลิเคชันเท่านั้น

โครงสร้างข้อมูล NGCREATEINFO เป็นข้อมูลที่ส่งให้กับคลาส NG3D เพื่อสร้างอุปกรณ์ในการแสดงผลที่ตามที่กำหนดซึ่งถ้าไม่สามารถหาอุปกรณ์ที่กำหนดได้ คลาสจะใช้ค่าโดยปริยายในการสร้างอุปกรณ์

โครงสร้างข้อมูล NGCREATEINFO

```
typedef struct _NGCREATEINFO
{
    HWND    hWnd;

    DWORD   dwAdapter;
    DWORD   dwWidth;
    DWORD   dwHeight;
    DWORD   dwMode;

    DWORD   dwMinDepthBits;
    DWORD   dwMinStencilBits;
    DWORD   dwNumDepthBuffer;

    BOOL    bWindow;
    BOOL    bCursor;
    BOOL    bUseDepthBuffer;
} NGCREATEINFO, *LPNGCREATEINFO;
```

ตารางที่ 11-2 ข้อมูลในโครงสร้างข้อมูล NGCREATEINFO

สมาชิก

hWnd

ค่าแฮนเดิลของวินโดวที่ต้องการจะใช้เรนเดอร์

dwAdapter

ค่าแสดงหมายเลขของอุปกรณ์ที่จะใช้ซึ่งคุณสมบัติของอุปกรณ์จะได้อาจจากการขั้นตอนการระบุอุปกรณ์ ในกรณีที่ต้องการใช้อุปกรณ์โดยปริยายให้ใส่ค่าเป็น 0

dwWidth

ค่าความกว้างของพื้นที่แสดงผลซึ่งจะมีผลเมื่อแสดงผลในแบบเต็มจอเท่านั้น

dwHeight

ค่าความสูงของพื้นที่แสดงผลซึ่งจะมีผลเมื่อแสดงผลในแบบเต็มจอเท่านั้น

dwMode

ค่าแสดงคุณลักษณะของสีที่จะแสดงผล โดยปกติจะมีค่าเป็น 16 ในกรณีที่ต้องการแสดงผลสีแบบ 16 บิต หรือ 32 ในกรณีที่ต้องการแสดงผลแบบ 32 บิต

dwMinDepthBits

ค่าแสดงบิตของเดฟบัฟเฟอร์ขั้นต่ำที่สามารถใช้งานได้โดยปกติกำหนดค่าไว้ที่ 16 หรือ 16 บิต

dwMinStencilBits

ค่าแสดงบิตของสเตนซิลบัฟเฟอร์ขั้นต่ำที่สามารถใช้งานได้โดยปกติมักจะไม่ใช่ใช้งานจึงกำหนดค่าไว้ที่ 0

dwNumDepthBuffer

ค่าแสดงจำนวนของเดฟบัฟเฟอร์ปกติกำหนดค่าไว้ที่ 1 สามารถกำหนดค่ามากกว่านี้ได้ซึ่งจะได้ภาพที่มีความนุ่มนวลขึ้นแต่จะใช้หน่วยความจำจำนวนมาก

bWindow

ให้แสดงผลในแบบวินโดวหรือไม่

bCursor

ให้แสดงฮาร์ดแวร์เคอร์เซอร์เมื่อแสดงผลแบบเต็มจอหรือไม่

bUseDepthBuffer

ให้มีการใช้งานเคฟบัฟเฟอร์หรือไม่

โครงสร้างของคลาส NG3D

```

class NG3D
{
protected:
    NGCREATEINFO          m_CreateInfo;
    RECT                  m_rcClient;
    DWORD                 m_dwNumAdapters;
    DWORD                 m_dwAdapter;
    DWORD                 m_dwBehavior;

    LPDIRECT3D8            m_lpD3d;
    LPDIRECT3DDEVICE8      m_lpDevice;

    D3DPRESENT_PARAMETERS m_d3dpp;
    D3DCAPS8              m_d3dCaps;

    LPDIRECT3DSURFACE8     m_lpBackBuffer;
    D3DSURFACE_DESC        m_d3dsdBackBuffer;

    NGADAPTERINFO          m_AdaptersInfo[8];
    NGMODEINFO*            m_pModeInfo;

    NG3D(NGCREATEINFO*);

    BOOL                  FindDepthStencilFormat(UINT,
        D3DDEVTYPE, D3DFORMAT, D3DFORMAT*);
    HRESULT                InitialGraphics();
    HRESULT                InitialDevice();
    HRESULT                CheckDevice();
    HRESULT                FindPreferDevice();

    static NG3D*          m_cGraphics;

public:
    ~NG3D();
    static HRESULT         CreateGraphics(NGCREATEINFO*);
    HRESULT                ResizeScene(RECT*);
    HRESULT                TestAdapter(NGADAPTERINFO*&, INT&);
    BOOL                   ConfirmDevice(D3DCAPS8*, DWORD, D3DFORMAT);
    virtual HRESULT        ClearResource();
    virtual HRESULT        RestoreResource();

    // Accessors
    LPDIRECT3DDEVICE8      GetDevice();
    NGADAPTERINFO*         GetAdapterInfo(UINT id);
    DWORD                  GetAdapterCount();
    D3DCAPS8*              GetDeviceCaps();
    NGCREATEINFO*          GetCreateInfo();

    static NG3D*           GetGraphics();
};

inline NG3D* Graphics()
{
    return NG3D::GetGraphics();
}

```

ตารางที่ 11-3 โครงสร้างในคลาส NG3D

ฟังก์ชันที่สำคัญ

static HRESULT CreateGraphics(NGCREATEINFO* createInfo)

ทำการสร้างอุปกรณ์สำหรับการแสดงผลตามค่าที่กำหนดใน โครงสร้างข้อมูล NGCREATEINFO
พารามิเตอร์

createInfo ค่าพอยเตอร์ที่ชี้ไปยังโครงสร้างข้อมูล NGCREATEINFO ซึ่งกำหนดลักษณะของ
อุปกรณ์แสดงผลที่จะสร้าง

หมายเหตุ คลาส NG3D มีวิธีการใช้งานต่างจากคลาสอื่น เนื่องจากคลาสนี้เป็นคลาสแบบสแตติก การใช้งาน
จึงต้องทำตามขั้นตอนดังนี้

```
#include "ng3d.h"
.
.
static NG3D*   g_NG3d = NULL;
.
.

// Create Direct3D (NG3D)
NGCREATEINFO createInfo;
NG3D::CreateGraphics(createInfo);
g_NG3d = Graphics()->GetGraphics();// Graphics() is a pointer to NG3D object
```

ตารางที่ 11-4 การสร้างวัตถุของคลาส NG3D

นอกจากนั้นแล้ว ฟังก์ชัน Graphics() สามารถนำไปใช้ที่ใดก็ได้ที่ include "ng3d.h" ไว้ ซึ่งการใช้
ฟังก์ชันนี้เมื่อกับการใช้งานพอยเตอร์ที่ชี้มายังออบเจกต์ NG3D

BOOL TestAdapter(NGADAPTERINFO*& adapterInfo, INT& numAdapter)

เป็นฟังก์ชันที่ใช้ในการทดสอบอุปกรณ์แสดงผลที่มีอยู่ในเครื่องคอมพิวเตอร์
พารามิเตอร์

adapterInfo ค่าพอยเตอร์ที่ชี้ไปยังโครงสร้างข้อมูล NGADAPTERINFO ซึ่งจะรับข้อมูลออกไป

numAdapter จำนวนของอุปกรณ์ที่ตรวจสอบพบ

ส่งค่ากลับ

ฟังก์ชันจะส่งค่า TRUE ถ้ามีการพบอุปกรณ์อย่างน้อย 1 ตัว และ FALSE ถ้าไม่พบอุปกรณ์ใดๆ
โครงสร้างข้อมูล NGADAPTERINFO

โครงสร้างข้อมูลนี้ใช้ในการเก็บข้อมูลของอุปกรณ์แสดงผลแต่ละตัวที่พบ


```
typedef struct _NGADAPTERINFO
{
    D3DADAPTER_IDENTIFIER8      d3dAdapterID;
    D3DDISPLAYMODE              d3ddmDesktop;
    D3DMULTISAMPLE_TYPE          d3dMultiSampleType;
    D3DMULTISAMPLE_TYPE          d3dWindowMultiSampleType;
    NGDEVICEINFO                 devices;
    DWORD                        dwNumModes;
    NGMODEINFO                   modes[100];
} NGADAPTERINFO, *LPNGADAPTERINFO;
```

ตารางที่ 11-5 ข้อมูลที่เก็บในโครงสร้างข้อมูล NGADAPTERINFO

สมาชิก

d3dAdapterID	เก็บข้อมูลทอธีบายที่มาของการ์ดแสดงผล เช่น ชื่ออุปกรณ์, รุ่น, ผู้ผลิต, ผู้จำหน่าย และไดรเวอร์ เป็นต้น
d3ddmDesktop	รีโซลูชันของการแสดงผลในปัจจุบัน
d3dMultiSampleType	ความสามารถในการทำ Anti-Aliasing ในการแสดงผลแบบเต็มจอ
d3dWindowMultiSampleType	ความสามารถในการทำ Anti-Aliasing ในแสดงผลแบบวินโดว์
devices	รายละเอียดของอุปกรณ์ที่มีอยู่บนการ์ด (มีให้เลือกเฉพาะในแบบฮาร์ดแวร์)
dwNumModes	จำนวนของรีโซลูชันที่สามารถแสดงผลได้
modes	เก็บข้อมูลรีโซลูชันที่สามารถแสดงผลได้

Renderer (คลาส NGRenderer ใน “ngrender.h”)

คลาส NGRenderer ทำหน้าที่ในการจัดระเบียบการเรนเดอร์ให้เป็นไปอย่างถูกต้อง โดยคลาส NGClone หรือคลาส NGTextureBox ที่ต้องการจะแสดงผลจะต้องส่งข้อมูลมาให้กับ Renderer ทุกครั้งที่ต้องการจะแสดงผล หรือส่งข้อมูลมาเก็บอย่างถาวรก็ได้ในกรณีที่ต้องการแสดงผลตลอดเวลา โดย Renderer จะเป็นผู้ไปขอสิ่งที่จำเป็นในการเรนเดอร์ภาพจะขอป๊อปเจ็คต์อื่นๆ เช่น ขอตำแหน่งจาก Frame, ขอค่าของแมตทรีเรียล, ขอค่าพอยเตอร์ของเทกเจอร์ เป็นต้น แล้วส่งให้บัฟเฟอร์ที่เก็บข้อมูลเรนเดอร์ภาพโดยส่งค่าที่จำเป็นไปให้

โครงสร้างของคลาส NGRenderer

```

class NGRenderer
{
private:
    NGRENDERDESC*      m_rdesc[1024];
    NGRENDERDESC*      m_tbdesc[1024];
    INT                 m_index;
    INT                 m_index2;
    NG3D*               m_p3d;
    LPDIRECT3DDEVICE8   m_pDevice;
    NGBuffer*           m_curBuffer;
    INT                 m_curTex0;
    INT                 m_curTex1;
    DWORD               m_dwFVF;
    NGRes2d*            m_res2d;
    INT                 m_Size2d;
    NGRenderer*         Initial(NG3D*);
    static NGRenderer*  m_this;

public:
    NGRenderer();

    BOOL Clear();
    BOOL Add(NGRENDERDESC*);
    BOOL Sort();
    BOOL Render();
    INT Length();
    NGRENDERDESC** GetRenderDesc();
    VOID SetRes2d(NGRes2d*);

    static NGRenderer* CreateRenderer(NG3D*);
    static NGRenderer* GetRenderer();
};

inline NGRenderer* Render()
{
    return NGRenderer::GetRenderer();
};

```

ตารางที่ 11-6 โครงสร้างคลาส NGRenderer

ฟังก์ชันที่สำคัญ

static NGRenderer* CreateRenderer(NG3D* p3d)

เป็นฟังก์ชันที่ใช้ในการสร้างอินสแตนซ์ของคลาสนี้

หมายเหตุ เช่นเดียวกับคลาส NG3D ออปเจ็กต์ของคลาสนี้จะมีได้เพียง 1 ออปเจ็กต์ใน 1 แอปพลิเคชัน เนื่องจากเป็นวัตถุที่จะจัดการให้มีการเรนเดอร์ภาพอย่างมีประสิทธิภาพและมีความถูกต้อง คลาสนี้มีวิธีการสร้างแบบเดียวกับคลาส NG3D แต่ใช้ Render() ในการใช้งานแทน

พารามิเตอร์

p3d อินสแตนซ์ของคลาส NG3D สร้างใส่ฟังก์ชัน Graphics() ในพารามิเตอร์นี้ได้

BOOL Add(NGRENDERDESC* renderDesc)

ฟังก์ชันนี้ทำหน้าที่เพิ่มงานในการเรนเดอร์โดยข้อมูลในโครงสร้างข้อมูล NGRENDERDESC จะใช้ในการเรียงลำดับ, นำไปใช้ข้อริชอร์สจากวัตถุอื่นๆ และใช้เป็นพารามิเตอร์ในการเรนเดอร์ภาพโดยออปเจ็กต์ของคลาส NGBuffer ที่เก็บข้อมูลเวกเท็กซ์ที่ใช้ในการเรนเดอร์ภาพ

พารามิเตอร์

renderDesc พอยเตอร์ที่ชี้ไปยังโครงสร้างข้อมูล NGRENDERDESC ซึ่งใช้ในการกำหนดการเรนเดอร์

โครงสร้างข้อมูล NGRENDERDESC

```
struct NGRENDERDESC
{
    NGBuffer*      pBuffer;
    INT            vStart;
    INT            vCount;
    INT            iBuffer;
    INT            iStart;
    INT            iCount;
    INT            iMtrl;
    INT            t0;
    INT            t1;
    INT            blend;
    D3DMATERIAL8*  mtrl;
    D3DMATRIX*     mat;
    ArrayList<NGFRAMEX*> joint;
    ArrayList<D3DXMATRIX*> jOffset;
};
```

ตารางที่ 11-7 ข้อมูลที่เก็บรักษาในโครงสร้างข้อมูล NGRENDERDESC

สมาชิก

pBuffer

พอยเตอร์ที่ชี้ไปยังวัตถุของ NGBuffer ซึ่งเก็บข้อมูลของเวอร์เท็กซ์และอินเด็กซ์ที่จะเรนเดอร์บนพื้นที่แสดงผล

vStart

ตำแหน่งเริ่มต้นที่จะเรนเดอร์ในเวอร์เท็กซ์บัฟเฟอร์

vCount

จำนวนของเวอร์เท็กซ์ในเวอร์เท็กซ์บัฟเฟอร์

iBuffer

หมายเลขของอินเด็กซ์บัฟเฟอร์ที่ใช้ในการเรนเดอร์ข้อมูลชุดนี้

iStart

ตำแหน่งเริ่มต้นของข้อมูลในอินเด็กซ์บัฟเฟอร์

iCount

จำนวนไพลม์ทีฟที่จะเรนเดอร์

iMtrl

หมายเลขของแมตทีเรียลใช้อ้างอิงในการขอข้อมูลแมตทีเรียล

t0

หมายเลขของเทกเจอร์ใช้อ้างอิงในการขอข้อมูลเทกเจอร์

t1

หมายเลขของเทกเจอร์ใช้อ้างอิงในการขอข้อมูลเทกเจอร์

blend

จำนวนของข้อมูลที่ใช้ในการทำอินเด็กซ์เวอร์เท็กซ์เบลนดิง

mtrl

พอยเตอร์ที่ชี้ไปยังข้อมูลแมตทีเรียล

mat

พอยเตอร์ที่ชี้ไปยังข้อมูลตำแหน่ง

joint

พอยเตอร์ที่ชี้ไปยังตำแหน่งของโบน

jointOffset

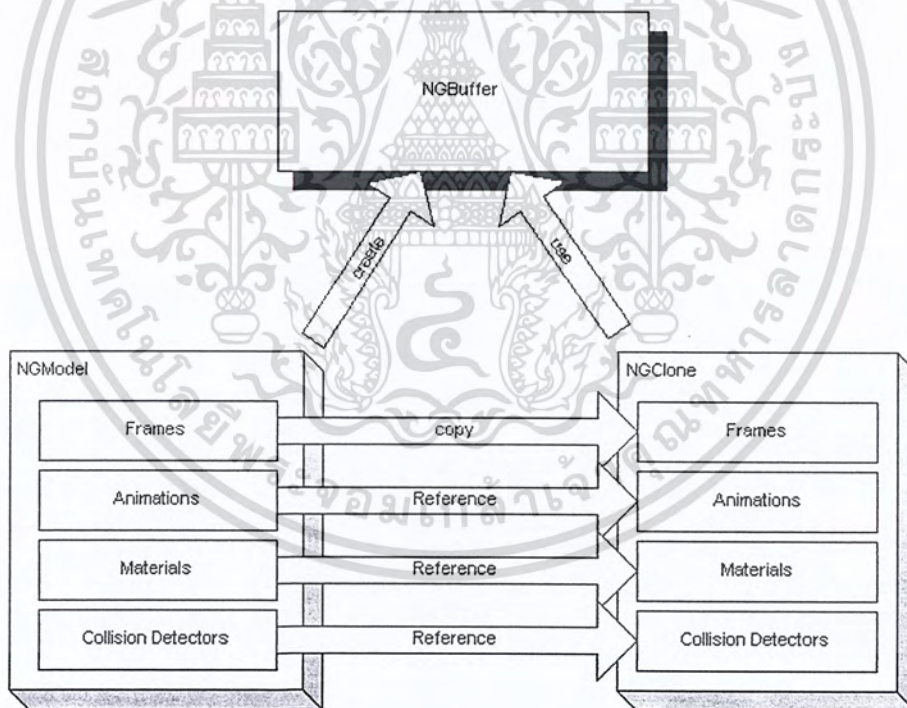
พอยเตอร์ที่ชี้ไปยังตำแหน่งที่เก็บค่าออฟเซตของโบน

Model และ Clone (คลาส NGModel และ NGClone ใน “ngmodel.h”)

โดยปกติแล้วในแอปพลิเคชันเกมจำนวนมาก โมเดลที่ปรากฏในฉากหนึ่งๆ มักมีข้อมูลของเวอร์เท็กซ์และอินเด็กซ์ที่ใช้ในการกำหนดรูปร่างของโมเดลที่เหมือนกัน ทั้งนี้เนื่องมาจากการออกแบบโมเดลให้หลากหลายเป็นงานที่ต้องใช้จินตนาการอย่างมาก, ใช้เวลานาน และเสียค่าใช้จ่ายมาก (ค่าจ้างนักออกแบบกราฟิก 3 มิติที่มีฝีมือมีราคาแพงมาก) นอกจากนั้น การมีโมเดลจำนวนมากยังสิ้นเปลืองหน่วยความจำและทำให้แอปพลิเคชันทำงานได้ไม่มีประสิทธิภาพเนื่องจากต้องติดต่อกับดิสก์ในการอ่านไฟล์จำนวนมาก ดังนั้น ในโครงการนี้จึงคิดวิธีการในการใช้งานโมเดลหนึ่งให้สามารถแสดงผลออกมาเสมือนเป็นหลายโมเดลโดยสร้างคลาส NGClone ขึ้นมา

คลาส NGClone จะทำการคัดลอกข้อมูลที่ทำให้แต่ละโมเดลมีความแตกต่างกันออกมาจากคลาส NGModel เช่น ข้อมูลที่เกี่ยวข้องกับตำแหน่งของโมเดล ส่วนข้อมูลอื่นที่ไม่ทำให้เกิดความแตกต่างจะใช้งานร่วมกัน เช่น ข้อมูลของเมตทรีเรียล, ข้อมูลของแอนิเมชัน และข้อมูลของคอลลิชันดีเทกเตอร์ เป็นต้น นั่นคือ คลาส NGModel ในโครงการนี้จะทำหน้าที่เป็นต้นแบบในการสร้างคลาส NGClone ซึ่งจะเป็นโมเดลที่ใช้ในการแสดงผลจริง

ความสัมพันธ์ระหว่างคลาส NGModel และคลาส NGClone



รูปที่ 11- 5 แสดงความสัมพันธ์ของคลาส NGModel และ NGClone

11.2.2.2 เกมเอนจินในส่วนของอุปกรณ์อินพุต

เอนจินในส่วนของการติดต่อกับอุปกรณ์อินพุตจะเป็นการนำเอาอินเทอร์เฟซและเมทอดที่มีอยู่ในไลบรารีอินพุตมาสร้างเป็นคลาส เพื่อที่จะใช้ในการทำงานในการติดต่อสำหรับอุปกรณ์ที่เราต้องการ เช่น เมาส์, คีย์บอร์ด และจอยสติค เป็นต้น เนื่องจากไลบรารีอินพุตได้จัดการเกี่ยวกับการติดต่อกับฮาร์ดแวร์ให้แล้วอุปกรณ์อินพุตจึงปรากฏให้เห็นในรูปของข้อมูลเท่านั้น

คุณสมบัติที่ต้องการ (Requirement)

1. สามารถทำให้ติดต่อรับข้อมูลกับอุปกรณ์มาตรฐาน เช่น เมาส์, คีย์บอร์ด และจอยสติค ได้
2. มีขั้นตอนในการใช้งานง่ายไม่ซับซ้อน

การออกแบบ

1. กำหนดวัตถุ

เนื่องจากแอปพลิเคชันของเราทำงานกับข้อมูลที่ได้รับจากไลบรารีอินพุตเท่านั้น ดังนั้นอุปกรณ์ต่างๆ ที่เชื่อมต่ออยู่จะปรากฏให้เราเห็นเป็นข้อมูลทั้งหมด ดังนั้นจึงให้ข้อมูลเป็นหลักในการค้นหาวัตถุ

ข้อมูล	วัตถุ	ระยะเวลาในการเก็บข้อมูล
อินพุต	ข้อมูลที่ได้รับจากการตรวจสอบอินพุตจากอุปกรณ์ที่เราเอาไว้แล้ว ไม่ว่าจะเป็นเมาส์, คีย์บอร์ด หรือจอยสติค	ตลอดเวลาที่แอปพลิเคชันทำงานอยู่

ตารางที่ 11-8 วัตถุที่ได้จากการแยกแยะโดยใช้ข้อมูลที่เก็บในส่วนของอินพุต

11.2.2.3 เกมเอนจินในส่วนของเสียงและวิดีโอ

ในส่วนของเราเรามีความต้องการที่แตกต่างกัน 2 รูปแบบ คือ

1. การเล่นเกมเอฟเฟ็ก เช่น เสียงฝีเท้า, เสียงประตู เป็นต้น ซึ่งการเล่นเสียงประเภทนี้มักจะต้องใช้ตำแหน่งในการคำนวณด้วย รวมทั้งยังต้องมีความสามารถในการผสมเสียงจากหลายๆ แหล่งเสียง เนื่องจากในความเป็นจริงแล้วเสียงสามารถเกิดขึ้นได้จากหลายๆ แหล่งเสียงพร้อมๆ กัน

2. การเล่นเกมเพลงหรือเสียงดนตรี เสียงเพลงหรือเสียงดนตรีจะเป็นสิ่งที่สร้างบรรยากาศให้เกิดขึ้นในเกมทำให้ผู้เล่นเกมมีอารมณ์ร่วมไปกับเหตุการณ์ที่ปรากฏขึ้น ซึ่งโดยปกติแล้วการเล่นเพลงจะเล่นเพียง 1 เพลงในแต่ละครั้งและต้องการขนาดของข้อมูลที่ไม่ใหญ่มากรวมทั้งใช้การประมวลผลไม่มาก ในสมัยก่อนไฟล์ในแบบ Midi ได้รับความนิยมมาก แต่ในปัจจุบันสามารถใช้ไฟล์ในรูปแบบที่มีขนาดเล็กและให้เสียงที่สมจริงมากกว่าได้ เช่น ไฟล์ ในรูปแบบ MP3 เป็นต้น

ในส่วนของการเล่นไฟล์วิดีโอก็สามารถใช้ไลบรารีโซว์ในการเล่นได้เช่นกัน เนื่องจากไลบรารีโซว์นั้นสามารถมีความสามารถในการเล่นไฟล์สตรีมทุกรูปแบบที่มีการสร้าง DMO (Direct Media Object) เอาไว้ (ทุกรูปแบบที่ Window Media Player สามารถเล่นได้)

การออกแบบ

1. กำหนดวัตถุ

- การเล่นเสียงเอฟเฟ็ค

วัตถุ	ข้อมูล	ระยะเวลาในการเก็บข้อมูล
เสียงเอฟเฟ็ค	ข้อมูลของเวลาที่อยู่ในรูปของไฟล์	ช่วงเวลาที่มีการเล่นเสียงเอฟเฟ็ค

ตารางที่ 11-9 วัตถุที่ได้จากการแยกแยะโดยใช้ข้อมูลที่เก็บในส่วนของเสียงเอฟเฟ็ค

- การเล่นเสียงดนตรีและเสียงเพลง

วัตถุ	ข้อมูล	ระยะเวลาในการเก็บข้อมูล
เสียงเพลงหรือเสียงดนตรี	ข้อมูลของเสียงที่ถูกบีบอัดด้วย MP3	ช่วงเวลาที่มีการเล่นเสียงเพลงหรือเสียงดนตรี

ตารางที่ 11-10 วัตถุที่ได้จากการแยกแยะโดยใช้ข้อมูลที่เก็บเสียงเพลง

- การเล่นวิดีโอ

วัตถุ	ข้อมูล	ระยะเวลาในการเก็บข้อมูล
วิดีโอ	ข้อมูลของวิดีโอที่ถูกบีบอัดด้วยอัลกอริทึมแบบต่างๆ เช่น AVI, DIVX และ MOVE เป็นต้น	ช่วงเวลาที่ต้องการเล่นไฟล์วิดีโอ

ตารางที่ 11-11 วัตถุที่ได้จากการแยกแยะโดยใช้ข้อมูลที่เก็บในส่วน of วิดีโอ

บทที่ 12

บทวิจารณ์และสรุป

12.1 สรุปผลการวิจัย

จากการทดสอบการทำงานของโปรแกรมที่เขียนขึ้นจากไลบรารีเกมเอนจินในโครงการนี้ พบว่า การใช้งานยังไม่ง่ายอย่างที่ได้ออกแบบไว้เนื่องจากในระหว่างพัฒนามีความจำเป็นต้องเพิ่มเติมส่วนประกอบต่างๆ เข้าไปมากมายทำให้การพัฒนาไม่ตรงกับที่ออกแบบไว้ นอกจากนั้น ยังมีปัญหาในเรื่องของประสิทธิภาพในการทำงานของโปรแกรมซึ่งทำให้ต้องกลับไปออกแบบโครงสร้างของเกมเอนจินใหม่หลายครั้ง ซึ่งทำให้การทำงานล่าช้า

นอกจากนั้น ในส่วนของรีซอร์สที่นำมาใช้ในโครงการพบว่าในบางครั้งข้อมูลที่ได้นั้นไม่ตรงกับที่สร้างในโปรแกรม 3 มิติทำให้ต้องเสียเวลาแก้ไข และเกิดความสับสนว่าเขียนแบบใดจึงจะให้ผลลัพธ์ที่ถูกต้องกันแน่

ดังนั้นสำหรับผู้ที่ต้องการจะพัฒนาเกมเอนจินในโครงการต่อไป ขอให้รวบรวมทฤษฎีและกฎต่างๆ ให้เรียบร้อยเสียก่อน ตรวจสอบความถูกต้องของรีซอร์สที่จะนำมาใช้งาน และอย่ามองข้ามเรื่องของประสิทธิภาพในการทำงานเนื่องจากเกมต่างจากโปรแกรมประเภทอื่นๆ การเขียนโปรแกรมเกมโดยไม่คำนึงถึงประสิทธิภาพอาจให้ผลลัพธ์ที่แตกต่างกันเป็นสิบเท่าอย่างที่คุณพัฒนาโครงการนี้ประสบ

12.2 แนวทางในการพัฒนาต่อ

โคเรเอกซ์ในเวอร์ชัน 8 นี้มีส่วนของไปป์ไลน์ที่สามารถโปรแกรมได้อยู่ซึ่งผู้พัฒนาไม่ได้ใช้งานในส่วนนี้เลย แต่เล็งเห็นว่าในอนาคตการใช้งานไปป์ไลน์ที่สามารถโปรแกรมได้นี้จะมีความสำคัญมากขึ้น เนื่องจากให้ผู้พัฒนาสามารถสร้างเอฟเฟ็กที่สวยงามและแตกต่างไปจากผลลัพธ์ที่เคยได้อย่างมาก และการตรวจสอบการชนของวัตถุในโครงการนี้เน้นการตรวจสอบที่ใช้งานได้จริงแต่ยังไม่มีความสมบูรณ์เพียงพอ

นอกจากนั้น ในปัจจุบันเกมแบบมัลติเพลเยอร์ที่เล่นผ่านระบบเน็ตเวิร์กเริ่มที่จะมีความนิยมมากขึ้น เกมที่จะพัฒนาในอนาคตจึงควรจะสามารถในส่วนนี้ด้วย



ภาคผนวก ก

การติดตั้งและใช้งาน Plug-in

1. Maya 4.0 X-File EXPORTER

Maya 4.0 X-File Exporter เป็น Plug-in สำหรับโปรแกรม Maya 4.0 ในการเขียนไฟล์แบบ .X ซึ่งไม่สามารถอ่านไฟล์ .X ได้ นอกจากนั้น Plug-in ของ Maya ยังไม่มีความเข้ากันได้ระหว่างเวอร์ชันอีกด้วย จึงต้องใช้ Plug-in ที่สร้างมาสำหรับเวอร์ชันนั้นเท่านั้น Plug-in สามารถดาวน์โหลดได้ที่

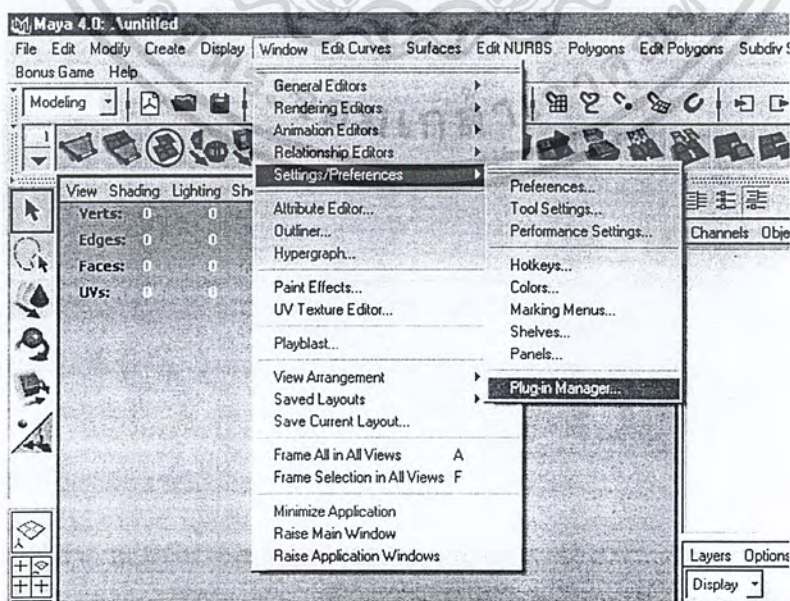
www.microsoft.com/directx ในหัวข้อ developer

1.1 การติดตั้ง



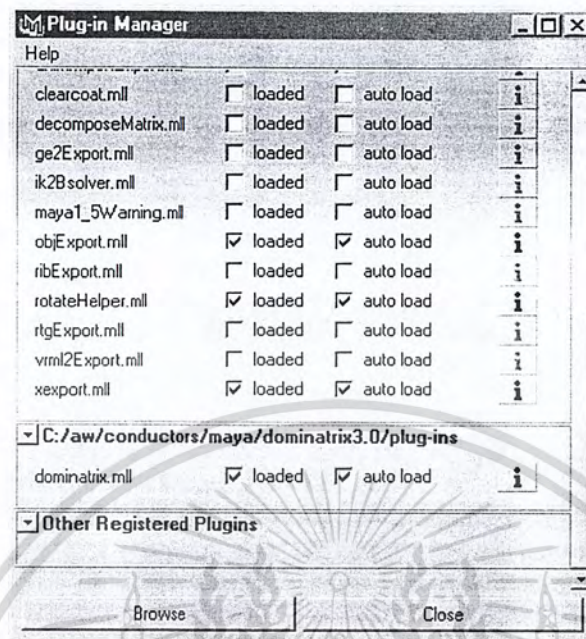
รูปที่ ก-1 แสดงไฟล์ที่จำเป็นในการติดตั้ง Plug-in สำหรับ Maya4.0

1. ปิดโปรแกรม Maya 4.0 ที่กำลังทำงานอยู่ทั้งหมด
2. คัดลอกไฟล์ที่ชื่อ "xExport.mll" ไปไว้ยังไดเรกทอรี "<mayapathname>\bin\plug-ins"
3. คัดลอกไฟล์ที่ชื่อ "xfileTranslatorOpts.mel" ไปไว้ยังไดเรกทอรี "<mayapathname>\scripts\others"
4. คัดลอกไฟล์ที่ชื่อ "bicubicBezierPatches.mel" ไปไว้ยังไดเรกทอรี "<mayapathname>\scripts\others"
5. เปิดโปรแกรม Maya 4.0
6. ไปที่เมนู "Window->Settings/Preferences ->Plug-in Manager" ดังรูป



รูปที่ ก-2 แสดงเมนู "Window>Settings/Preferences>Plug-in Manager"

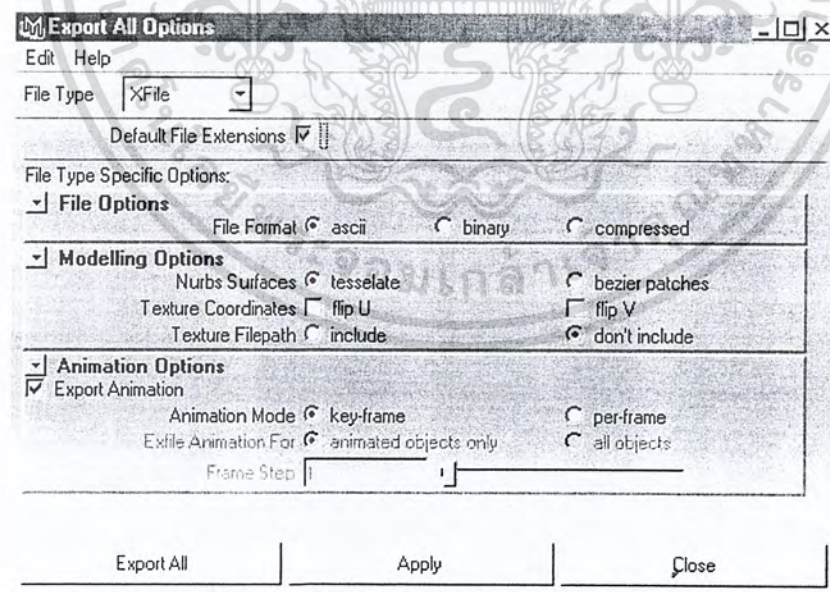
7. คลิกที่ loaded เพื่อโหลด Plug-in เพื่อใช้งาน และคลิกที่ auto load ในกรณีที่ต้องการให้โหลด Plug-in ทุกครั้งที่เปิดโปรแกรม



รูปที่ ก-3 แสดงไดอะล็อกบ็อกโหลด Plug-in

1.2 การใช้งาน

1. ไปที่เมนู "File->Export All"
2. เลือก "XFile (*.*)" จาก "File of type" drop-down menu



รูปที่ ก-4 แสดงไดอะล็อกบ็อกเขียนไฟล์ .X

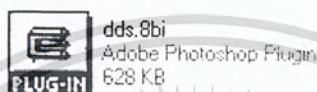
3. กด Export All เมื่อต้องการเขียนไฟล์
4. ใส่ชื่อไฟล์ที่ต้องการในไดอะล็อกบ็อกแล้วกดปุ่ม Save

2. NVIDIA DDS Format (V3.11) for PhotoShop6.0

NVIDIA DDS Format (V3.11) เป็น Plug-in ที่ทำให้เราสามารถอ่านและเขียนไฟล์แบบ DDS ได้อย่างง่ายดาย ทำให้อ่านความสะดวกได้อย่างมาก

หมายเหตุ ไฟล์เทกเจอร์จะต้องเป็นรูปสี่เหลี่ยมจัตุรัสซึ่งมีความยาวด้านมีค่าเป็น 2 ยกกำลัง n โดยผลลัพธ์ของ 2 ยกกำลัง n จะต้องน้อยกว่าขนาดความยาวเทกเจอร์ที่ยาวที่สุดที่การ์ดเร่งความเร็วสามารถประมวลผลได้ (โดยปกติมีความยาวด้านที่ยาวที่สุดประมาณ 2048 pixel)

2.1 การติดตั้ง

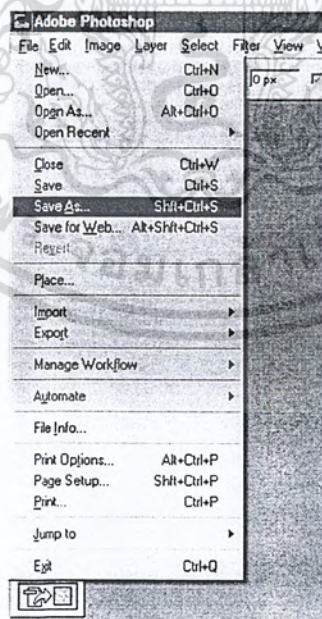


รูปที่ ก-5 แสดงไฟล์ที่จำเป็นในการติดตั้ง Plug-in สำหรับ PhotoShop

1. ปิดโปรแกรม Adobe PhotoShop 6.0 ที่กำลังทำงานอยู่ทั้งหมด
2. คัดลอกไฟล์ชื่อ “dds.8bi” ไปยังไดเรกทอรี “<Adobe PhotoShop Pathname>\Plug-ins\File Formats”
3. เปิดโปรแกรม Adobe PhotoShop 6.0 โดย Plug-in จะถูกโหลดขึ้นมาทำงานโดยอัตโนมัติ

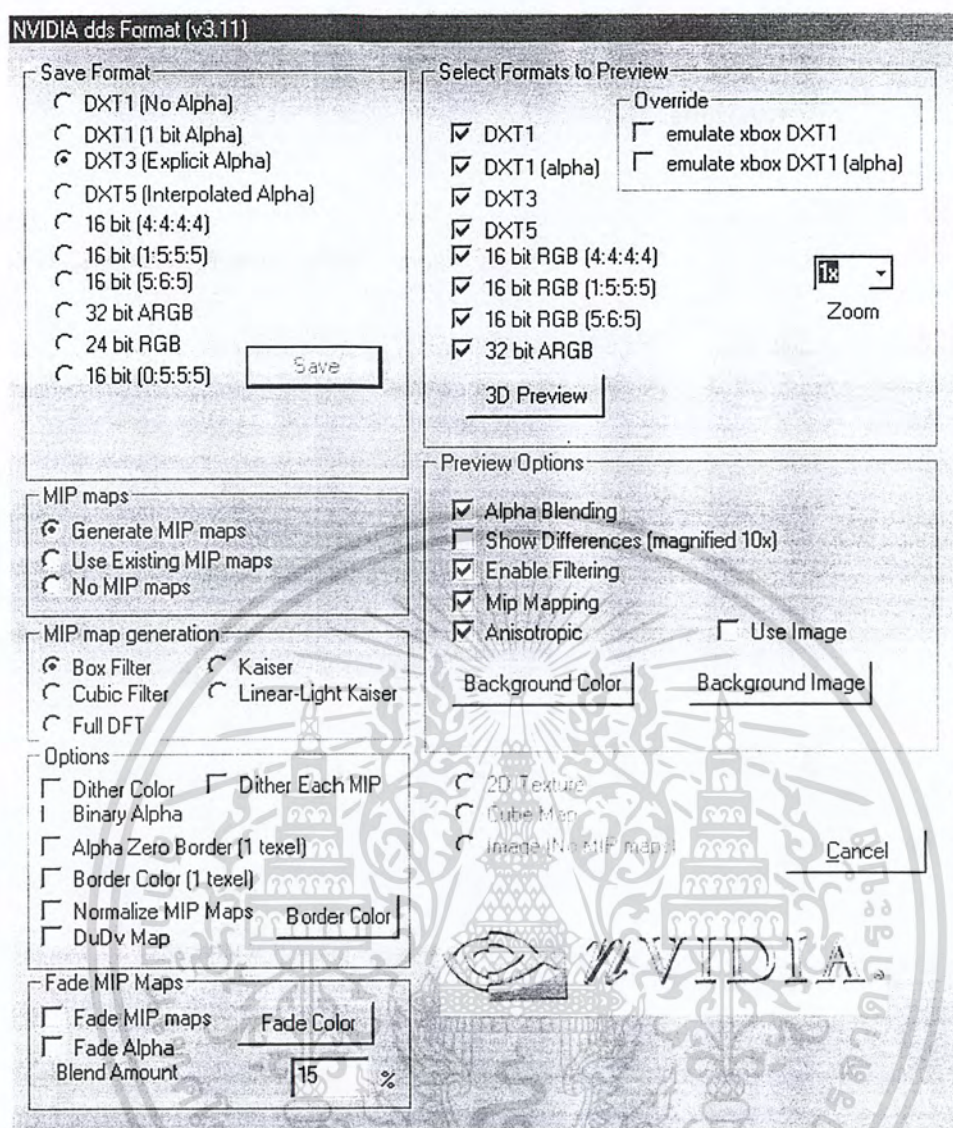
2.2 การใช้งาน

1. ไปยังเมนู “File->Save As”



รูปที่ ก-6 แสดงเมนู “File>Save As”

2. เลือกชนิดของไฟล์เป็น DDS
3. กดปุ่ม save จะปรากฏไดอะล็อกบ็อกกำหนดค่าของไฟล์ DDS



รูปที่ ก-7 แสดงไดอะล็อกบ็อกซ์เขียนไฟล์ .DDS

- เลือกรูปแบบไฟล์ที่ต้องการแล้วกดปุ่ม save ถ้าไม่สามารถกดได้แสดงว่ารูปมีปัญหา ซึ่งอาจเกิดจากขนาดของรูปหรือความลึกของสีไม่สามารถใช้เป็นเทกเจอร์ได้

บรรณานุกรม

- [1] Adrain Perez, Dan Royer: "Advance 3-D Game Programming with DirectX 7.0", Wordware Publishing Inc., 6 2000.
- [2] Mark DeLoura (2000): "Game Programming Gems", Charles River Media, 8 2000.
- [3] Andre LaMothe (1999): "Tricks of the Windows Game Programming Gurus", Sams, 9 1999.
- [4] Peter Donnelly and DirectX Team (1998): "Inside DirectX", Microsoft Press, 3 1998.
- [5] "DirectX Official Homepage," <http://www.microsoft.com/directx>
- [6] "NVIDIA Developer Homepage," <http://www.nvidia.com/developer>
- [7] "Game Developer Conferences Homepage," <http://www.gdconf.com>

