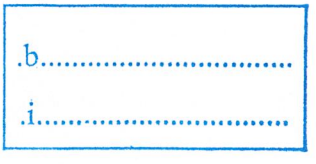
แขนกลจำลองบนคอมพิวเตอร์

ROBOTIC ARM SIMULATION

โดย

นางสาวพรีสา อารีกุล

นายเอกรัตน์ พนมฤทธิ์



ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมระบบควบคุม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

แขนกลจำลองบนคอมพิวเตอร์

ROBOTIC ARM SIMULATION

โดย

นางสาวพรีสา อารีกุล

นายเอกรัตน์ พนมฤทธิ์

อาจารย์ที่ปรึกษา

อาจารย์สุมิตร พนาอุดมทรัพย์

ปริญญานิพนธ์ฉบับนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิศวกรรมระบบควบคุม

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2544

ปริญญานิพนธ์ปีการศึกษา 2544

ภาควิชา วิศวกรรมระบบควบคุม

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง แขนกลจำลองบนคอมพิวเตอร์ (Robotic Arm Simulation)

ผู้จัดทำ

1. นางสาวพรีสา อารีกุล รหัส 41014296
2. นายเอกรัตน์ พนมฤทธิ์ รหัส 41014550

.....อาจารย์ที่ปรึกษา

(อาจารย์สุมิตร พนาอุดมทรัพย์)

แขนกลจำลองบนคอมพิวเตอร์

พรีสา อารีกุล
เอกรัตน์ พนมฤทธิ์
อาจารย์สุมิตร พนาอุดมทรัพย์
ปีการศึกษา 2544

บทคัดย่อ

ปริญญานิพนธ์ฉบับนี้ ถูกเรียบเรียงขึ้นจากผลงานในการเขียนโปรแกรมจำลองภาพและการทำงานของแขนกล 3 มิติที่มีการสั่งงานและแสดงผลทางคอมพิวเตอร์ ซึ่งเป็นผลงานทางด้านซอฟต์แวร์ (Software) เพียงอย่างเดียว โดยทำการสร้างภาพแขนกลจำลอง 3 มิติด้วยโปรแกรมมายา เวอร์ชั่น 3.0 (Maya version 3.0) แล้วรับอินพุท (Input) ข้อมูลจากผู้ใช้ ซึ่งจะอยู่ในรูปของมุมหรือระยะทางที่ข้อของแขนกลเคลื่อนที่ไป แล้วแสดงผลของเอาท์พุท (Output)ในรูปของตำแหน่งปลายมือแขนกลและภาพการเคลื่อนที่ของแขนกลไปตามข้อมูลอินพุทที่ผู้ใช้ป้อนเข้าไป ซึ่งการทำงานแบบนี้เราเรียกว่าเป็นแบบไดเร็กไคเนมาติกซ์ (Direct Kinematics) โดยโปรแกรมที่ใช้เขียนด้วยวิชวลซีพลัสพลัส (Visual C++) ร่วมกับไดเร็กเอ็กซ์ 8 เอสดีเค(Direct X 8 SDK) ซึ่งเป็นโปรแกรมที่ใช้จัดการกับภาพ 3 มิติโดยเฉพาะ เนื่องจากโปรแกรม C++ เพียงอย่างเดียวไม่สามารถรองรับการเรนเดอร์ (Render) ภาพ 3 มิติที่ซับซ้อนได้ โปรแกรมที่สร้างขึ้นนี้ยังไม่สามารถตอบสนองความต้องการของผู้ที่จะนำไปใช้งานจริงได้ เช่น หากต้องการนำโปรแกรมไปทดสอบโดยการต่อเข้ากับแขนกลจริงโดยมีการสั่งงานผ่านคอมพิวเตอร์ ซึ่งยังคงต้องมีการพัฒนาต่อ ๆ ไปเป็นลำดับ

Robotic Arm Simulation

Phreesa Areekul

Akarat Panomrit

Sumit Panaudomsap Advisor

2001

Abstract

This paper proposes the application of simulating 3 dimensions robotic arm models on the computer. It is a software application which uses Maya version 3.0 to build the 3-D robotic arms and Visual C++ programming language with Direct X 8.0 SDK to show the movement of those models. We can pass the angle of revolute joint or distance of prismatic joint as the input and the model will move correspond to these the values. The position of the end effector in 3-D Coordinate is also shown.

This application is a basic guide line of developing robotic arm software simulation to carry practical work in real life.

สารบัญ

สารบัญรูปภาพ

หน้า

หน้า

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความเป็นมา

" หุ่นยนต์ " หรือโรบอท (Robot) เป็นสิ่งประดิษฐ์หรือเครื่องจักรกลที่สามารถทำงานต่าง ๆ แทนมนุษย์ได้ " แขนกล " (Robotic arm) ก็เป็นอีกรูปแบบหนึ่งของโรบอท ที่มีการทำงานได้เฉพาะอย่าง แต่จะมีลักษณะการทำงานที่เลียนแบบแขนและมือของมนุษย์ เรามักจะไม่พบเห็นแขนกลที่ว่านี้ในชีวิตประจำวันนัก แต่ในทางอุตสาหกรรมแล้วเราสามารถพบเห็นมันได้บ่อย เพราะมันมีความสามารถหลายอย่างที่เหนือกว่ามนุษย์ จากการใช้หลักการทางเศรษฐศาสตร์ในการคิดความคุ้มค่าหรือต้นทุนในการใช้ robot แทนแรงงานคน พบว่ามีหลายอุตสาหกรรมซึ่งเหมาะที่จะใช้ robot เป็นตัวดำเนินการในกระบวนการผลิตมากกว่า โดยเฉพาะอุตสาหกรรมใหญ่ ๆ ที่ต้องการกำลังการผลิตมาก ๆ แขนกลในมโนภาพของบางคนอาจจะคิดว่าความสามารถของมันทำได้แค่เพียงการหยิบจับหรือยกเคลื่อนย้ายสิ่งของเท่านั้น แต่ความเป็นจริงแล้วมันสามารถทำได้มากกว่านั้นขึ้นอยู่กับโครงสร้างลักษณะทางกายภาพ การโปรแกรมการทำงานและจุดประสงค์การใช้งาน ไม่ว่าจะเป็นการเชื่อมโลหะ, การเจาะ, การพ่นสี (Spray painting). การขันสกรู-คลายสกรู. การตัดชิ้นงานด้วยแสงเลเซอร์ (Laser), การซีล (Seal), การประกอบบอร์ดอิเล็กทรอนิกส์ (Electronics boards) เป็นต้น เหล่านี้เป็นเหตุผลที่ทำให้การนำหุ่นยนต์หรือแขนกลไปใช้งานอย่างกว้างขวาง ฉะนั้นแนวโน้มในการผลิตและพัฒนาหุ่นยนต์หรือแขนกลก็ย่อมมีเพิ่มขึ้นตามไปด้วย

จากที่กล่าวมาทั้งหมดเป็นเหตุผลที่ทำให้เกิดโครงงานนี้ขึ้น โดยได้เล็งเห็นแล้วว่าการสร้างและคิดค้นแขนกลเพื่อนำไปใช้งานจริงนั้นมีความลำบากและความซับซ้อนอยู่มาก เพราะต้องใช้ความรู้ทั้งทางด้านวิศวกรรมอิเล็กทรอนิกส์, วิศวกรรมเครื่องกล, วิศวกรรมอุตสาหการ, วิทยาศาสตร์คอมพิวเตอร์ (Computer science), คณิตศาสตร์ และ เศรษฐศาสตร์ ดังนั้นถ้าหากเราจะมีอุปกรณ์ช่วยในการทดลองหรือทดสอบแขนกล เพื่อนำไปใช้งานหรือในระหว่างการวิจัยพัฒนาก็น่าจะเป็นสิ่งดีและมีประโยชน์ไม่น้อย จึงมีแนวคิดที่จะสร้างโปรแกรมจำลองแขนกลบนคอมพิวเตอร์ขึ้น

1.2 จุดประสงค์

1. ใช้โปรแกรม Visual C++ ร่วมกับ Direct X ในการสร้างโปรแกรมให้ภาพแขนกลจำลอง 3 มิติเคลื่อนที่ไปตามค่าอินพุท (Input) ที่ป้อนได้

2. เขียนโปรแกรมพื้นฐานในการพัฒนาแขนกล ด้วยการจำลองรูปแบบแขนกล 3 มิติลงบนคอมพิวเตอร์ โดยที่ไม่ต้องทดสอบกับแขนกลจริง

3. เพื่อการประหยัดเวลาและทรัพยากรในการทดลอง การใช้งานแขนกลตัวหนึ่ง ๆ ได้

2.3 ขอบเขต

สามารถสร้างโปรแกรมแสดงการเคลื่อนที่ของแขนกลเมื่อเรากำหนดค่ามุมหรือระยะการเคลื่อนที่ของข้อแขนกลซึ่งแสดงด้วยภาพ 3 มิติที่สร้างไว้ตายตัวเป็นอินพุท แล้วแสดงผลเป็นค่าตำแหน่งของจุดปลายมือได้อย่างถูกต้องในพิกัดตำแหน่ง (Coordinate) ที่แขนกลนั้นตั้งอยู่รวมถึงแสดงภาพการเคลื่อนที่ของแขนกลให้เป็นไปตามค่าอินพุทที่ต้องการ ซึ่งการแก้ปัญหาในรูปแบบนี้เรียกว่าการแก้ปัญหาแบบไดเร็กไคเนมาติกส์ (Direct Kinematics) และเมื่อสามารถแก้ปัญหาแบบไดเร็กไคเนมาติกส์ได้แล้ว ก็จะทำการแก้ปัญหาแบบอินเวอร์สไคเนมาติกส์ (Inverse Kinematics) ต่อไป โดยจะทำการรับค่าอินพุทเป็นตำแหน่งปลายมือของแขนและค่าเอาท์พุทที่ได้จะอยู่ในรูปของมุมที่แต่ละข้อของแขนกลจำลองเคลื่อนที่ไป รวมทั้งมีการแสดงภาพการเคลื่อนไหวไปยังตำแหน่งที่กำหนด ตัวโปรแกรมเขียนด้วย Visual C++ version 6.0 และ Direct X 8 SDK ส่วนภาพ 3 มิติสร้างด้วยโปรแกรม Maya Version 3.0

2.4 วิธีการพัฒนา

ผลงานชิ้นนี้อาจเป็นพื้นฐานหรือแนวคิดริเริ่มที่จะพัฒนาโปรแกรมที่ใช้สำหรับทดลองแขนกลที่นำไปใช้งานได้จริงต่อไปในอนาคต โดยในส่วนโปรแกรมสามารถที่จะเพิ่มรูปแบบของแขนกลประเภทต่าง ๆ หรือสามารถกำหนดรูปแบบแขนกลให้เป็นไปในแบบที่เราต้องการได้โดยไม่มีการกำหนดไว้ล่วงหน้าตายตัวหรือมีการเชื่อมต่อกับแขนกลของจริงแล้วสามารถสั่งการผ่านคอมพิวเตอร์โดยแสดงผลทั้งในรูปการเคลื่อนที่ของแขนกลจริงและแขนกลจำลองเหล่านี้เป็นต้น ซึ่งคงต้องอาศัยทักษะและความชำนาญทางด้านการเขียนโปรแกรมหรืออาจจะใช้โปรแกรมอื่นร่วมด้วยรวมไปถึงการเพิ่มอุปกรณ์ทางด้านฮาร์ดแวร์ (Hardware) บางอย่างที่จำเป็น เช่นอุปกรณ์อนาลอกทูดิจิตอล (Analog to digital) หรือ ดิจิตอลทูอนาลอก (Digital to analog) เป็นต้น

บทที่ 2
ทฤษฎีพื้นฐานในการวิเคราะห์หุ่นยนต์ (Fundamental of Robotic Analysis)

หุ่นยนต์ (Robot) หมายถึง เครื่องที่ควบคุมหรือใช้งานด้วยมือที่สามารถนำมาใส่โปรแกรมการทำงานในรูปแบบ Multifunction ได้ตามจุดประสงค์ที่เราต้องการ โดยจะถูกออกแบบมาเพื่อเคลื่อนย้ายวัตถุ เครื่องมือ หรืออุปกรณ์พิเศษไปตามการเคลื่อนที่ที่ได้โปรแกรมไว้สำหรับการทำงานในรูปแบบต่าง ๆ

จะเห็นได้ว่าคำจำกัดความของคำว่า "หุ่นยนต์" ที่กล่าวมานั้นจะหมายถึง แขนกล"Robotic Arm" หรือ "Manipulator" เนื่องจากคำว่า Robot ที่เราพูดถึงเป็น Industrial Robot หรือหุ่นยนต์ที่ใช้งานในอุตสาหกรรม

2.1. การแบ่งประเภทของหุ่นยนต์ (Robot Classification)

2.1.1 แบ่งตามเทคโนโลยีการขับเคลื่อน (Drive Technology)

พิจารณาจากต้นกำลังที่ใช้ในการขับเคลื่อนข้อต่อ (joint) ของหุ่นยนต์ ซึ่งวิธีที่นิยมมี 2 แบบด้วยกันคือการขับเคลื่อนด้วยไฟฟ้าและการขับเคลื่อนด้วยไฮดรอลิก (Hydrolic) แขนกลส่วนใหญ่ในปัจจุบันเป็นแบบที่ขับเคลื่อนด้วยไฟฟ้า โดยใช้ดีซี เซอร์โว มอเตอร์ (DC Servo Motor) หรือ ดีซี สเตปปิ้ง มอเตอร์ (DC Stepping Motor) แต่อย่างไรก็ตามในการใช้แขนกลเคลื่อนย้ายวัตถุที่หนักมาก ๆ และต้องการความเร็วสูง มักจะใช้การขับเคลื่อนด้วยไฮดรอลิก

2.1.2 แบ่งตามขอบเขตการทำงาน (Work Envelope Geometries)

ขอบเขตการทำงานสุทธิ (Gross Work Envelope) หมายถึง ขอบเขตในระนาบสามมิติที่ข้อมือ (wrist) ของแขนกลสามารถเคลื่อนที่ไปถึงได้โดยเราจะเรียกแกนของข้อต่อ 3 ข้อแรกว่าเป็นแกนหลัก (Major axe) ซึ่งจะใช้ในการหาตำแหน่งของข้อมือ ส่วนแกนของข้อต่อที่เหลือเรียกว่าแกนรอง (Minor axe) ซึ่งจะใช้ในการหาทิศทางการหมุนของปลายมือ (Tool) ดังนั้นรูปทรงขอบเขตการทำงานพิจารณาได้จากลำดับและชนิดของข้อต่อที่ใช้ในแกนหลัก ชนิดของข้อต่อที่ใช้ใน Industrial Robot มีอยู่ 2 แบบด้วยกัน ดังตาราง

ชนิด	เครื่องหมาย	สัญลักษณ์	ลักษณะการเคลื่อนที่
ข้อต่อแบบหมุน (Revolute)	R		หมุนรอบแกน
ข้อต่อแบบเลื่อน (Prismatic)	P		เชิงเส้นตามแนวแกน

ตารางที่ 2.1 แสดงชนิดข้อต่อของหุ่นยนต์

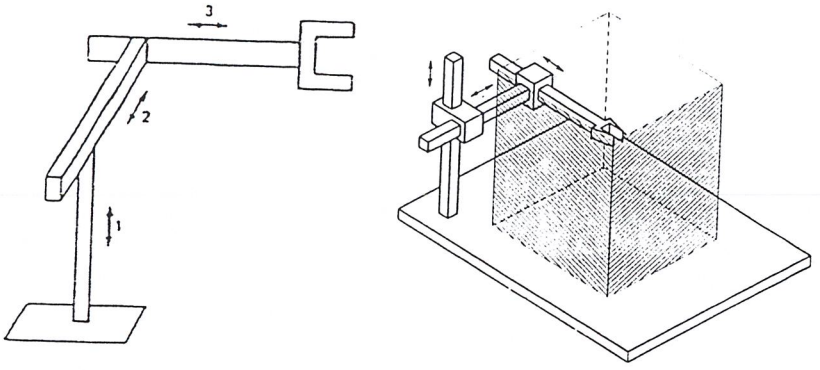
การต่อกันอย่างเป็นลำดับของข้อต่อแขนกลชนิดต่าง ๆ ทำให้เกิดลักษณะรูปทรงขอบเขตการทำงานแตกต่างกันไป 5 รูปแบบด้วยกัน ในการวิเคราะห์การเคลื่อนที่ข้อต่อแบบหมุน (R) จะวิเคราะห์ยากกว่าข้อต่อแบบเลื่อน (P) ฉะนั้นถ้ายิ่งมีข้อต่อแบบหมุนมากแขนกลก็จะยิ่งซับซ้อนมากขึ้น

Robot	Axis 1	Axis 2	Axis 3	Total revolute
Cartesian	P	P	P	0
Cylindrical	R	P	P	1
Spherical	R	R	P	2
SCARA	R	R	P	2
Articulated	R	R	R	3

P = Prismatic , R = Revolute

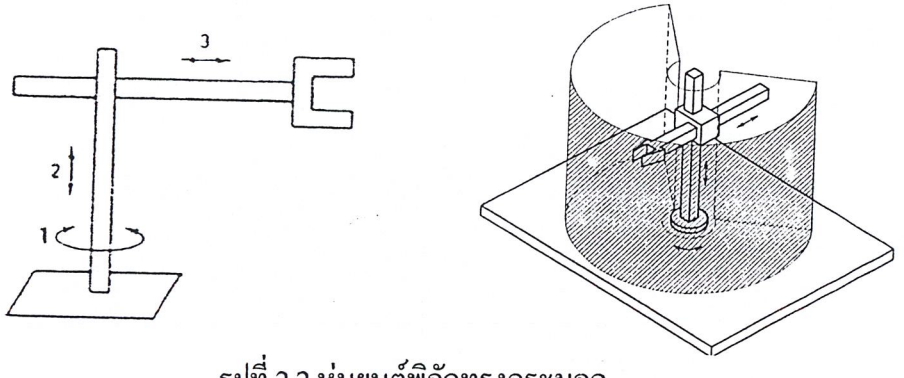
ตารางที่ 2.2 แสดงขอบเขตการทำงานของหุ่นยนต์ซึ่งขึ้นกับลักษณะข้อต่อของแกนหลัก

ก. หุ่นยนต์พิกัดคาร์ทีเซียน (Cartesian-coordinate robot หรือ Rectangular-coordinate robot) สัญลักษณ์ **PPP** ดังรูปที่ 2.1 โดยในแกนที่ 1,2 และ 3 จะเคลื่อนที่ขึ้น-ลง, เข้า-ออก และเดินหน้า-ถอยหลัง ตามลำดับ ซึ่งจะทำให้พื้นที่การทำงานมีลักษณะเป็นกล่องสี่เหลี่ยมมุมฉาก



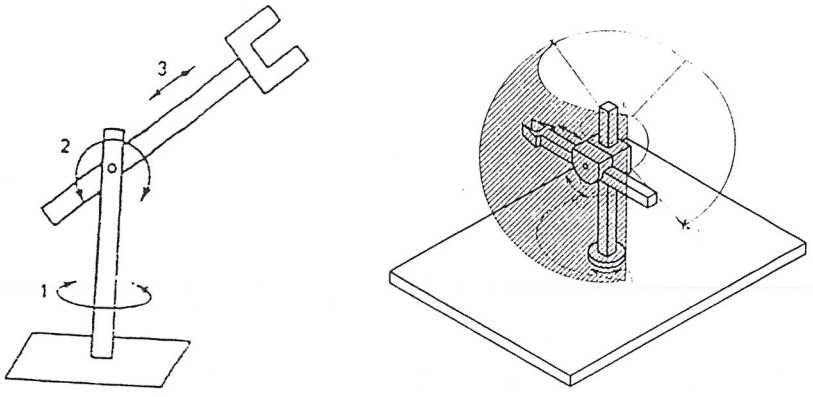
รูปที่ 2.1 หุ่นยนต์พิกัดคาร์ทีเซียน

ข. หุ่นยนต์พิกัดทรงกระบอก (Cylindrical-coordinate robot) สัญลักษณ์ **RPP** ดังรูปที่ 2.2 โดยในแกนที่ 1,2 และ 3 จะทำการหมุนไปข้างหน้าหรือถอยหลัง, เคลื่อนที่ขึ้น-ลง และเข้า-ออก ตามลำดับ



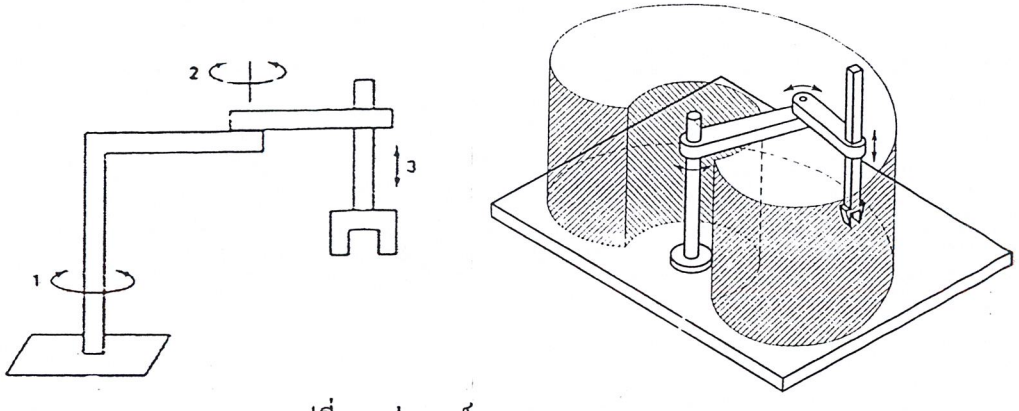
รูปที่ 2.2 หุ่นยนต์พิกัดทรงกระบอก

ค. หุ่นยนต์พิกัดทรงกลม (Spherical-coordinate robot) สัญลักษณ์ **RRP** แสดงดังรูปที่ 2.3 ลักษณะการเคลื่อนที่ของแขนในข้อที่ 2 จะยกขึ้นและลงในแนวดิ่งได้โดยทำมุมกับแกนนอนของฐาน สามารถหมุนได้รอบแกนที่ตั้งฉากกับฐานด้วยข้อต่อที่ 1 และเคลื่อนที่เข้าออกได้ด้วยแกนที่ 3



รูปที่ 2.3 หุ่นยนต์พิกัดทรงกลม

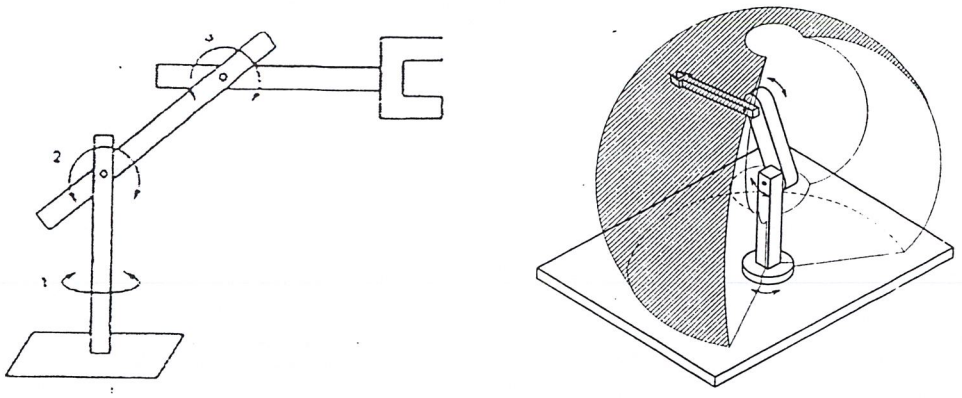
ง. หุ่นยนต์ SCARA (Selective Compliance Assembly Robot Arm) แสดงได้ดังรูปที่ 2.4 ซึ่งจะมีลักษณะคล้ายกับหุ่นยนต์พิกัดทรงกลม แต่จะแตกต่างกันตรงข้อต่อที่ 2 ซึ่งจะหมุนรอบแกนในแนวตั้ง และการเคลื่อนที่ของแกนที่ 3 จะเป็นแนวดิ่งคือมีการเคลื่อนที่ขึ้น-ลง สัญลักษณ์ที่ใช้แสดงแทนคือ **RRP** ดังนั้นข้อต่อที่ 1 และ 2 จึงเป็นตัวควบคุมการเคลื่อนที่ในแนวราบ ส่วนการเคลื่อนที่ในแนวดิ่งจะถูกกำหนดโดยข้อต่อที่ 3



รูปที่ 2.4 หุ่นยนต์ SCARA

จ. หุ่นยนต์ข้อต่อหมุน (Articulated-coordinate robot หรือ Revolute robot)

สัญลักษณ์ **RRR** ทั้ง 3 ข้อเป็นแบบข้อต่อหมุน แขนกลแบบนี้มีลักษณะเหมือนกับแขนของมนุษย์มากที่สุด คือมีข้อต่อต่าง ๆ เหมือนกับหัวไหล่, ข้อศอก และข้อมือ ดังนั้นพื้นที่การทำงานจึงสามารถทำงานได้ในทุกตำแหน่งในระยะความยาวของแขน



รูปที่ 2.5 หุ่นยนต์ข้อต่อหมุน

หุ่นยนต์แต่ละชนิดก็มีข้อดีข้อเสียแตกต่างกันออกไป เพราะลักษณะทางกายภาพที่แตกต่างกัน แต่ถ้าหากเรามองในแง่ของการทำงานที่เป็นแบบซ้ำ ๆ ที่จุดเดิมตลอด หุ่นยนต์พิกัดคาร์ทีเซียนจะสามารถทำงานได้ดีกว่า คือสามารถเคลื่อนที่ไปหาเป้าหมายโดยมีความผิดพลาดน้อยที่สุด แต่ถ้ามองในแง่ของการเข้าถึงวัตถุ ชนิดพิกัดทรงกลมและชนิดข้อต่อหมุนจะสามารถเข้าถึงวัตถุได้ดีกว่าชนิดอื่น ชนิดพิกัดทรงกระบอกมีข้อดีตรงที่สามารถยกวัตถุได้มากกว่า ในงานทั่วไปแล้วใช้จะแบบพิกัดทรงกระบอกและพิกัดทรงกลม เพราะทั้ง 2 ชนิดสามารถที่จะทำงานเป็นแบบ Load และ Unload โดยมีการเคลื่อนที่ของแขนออกไปในด้านข้างได้ดีกว่าชนิดอื่น ๆ

2.1.3 แบ่งตามลักษณะการควบคุมการเคลื่อนไหว

การแบ่งชนิดของหุ่นยนต์อีกวิธีหนึ่งคือแบ่งตามลักษณะการควบคุมการเคลื่อนไหวซึ่งแสดงได้โดยตารางที่ 3 โดยชนิดแรกคือ**การเคลื่อนที่แบบจุดต่อจุด (Point-to-Point Motion)** ซึ่งปลายมือ (Tool) จะเคลื่อนที่เป็นลำดับของจุดซึ่งไม่ต่อเนื่องในพื้นที่การทำงาน (Work Space) คือไม่มีการควบคุมปลายมือบนเส้นทางระหว่างจุดนั่นเอง การเคลื่อนที่แบบจุดต่อจุดมีประสิทธิภาพในการใช้งานที่จำกัด เช่นงานประเภทการเชื่อมโลหะตามจุดต่าง ๆ

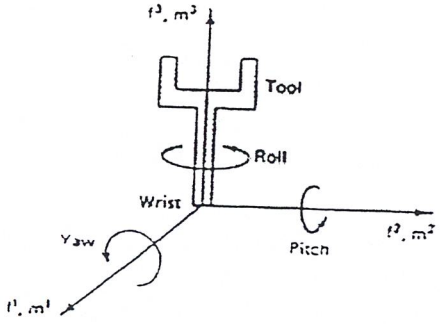
ส่วนการเคลื่อนที่อีกแบบหนึ่งคือ**การเคลื่อนที่แบบเป็นเส้นทางต่อเนื่อง (Continuous-Path)** คือเส้นทางในการเคลื่อนที่ทั้งหมดของปลายมือจะถูกกำหนดและความเร็วในการเคลื่อนที่ไปตามเส้นทางอาจจะไม่คงที่ ฉะนั้นหุ่นยนต์แบบนี้จึงต้องการการแก้ปัญหาในการควบคุมมากกว่าหุ่นยนต์ที่มีการเคลื่อนที่แบบจุดต่อจุด ตัวอย่างการนำไปใช้งานของหุ่นยนต์ประเภทนี้ ได้แก่ การพ่นสีชิ้นงาน (Spray Painting) การเชื่อมประสาน และการ Seal

วิธีการควบคุม	การนำไปใช้งาน
แบบจุดต่อจุด (Point to point)	การเชื่อมแบบเป็นจุด
	การหยิบจับชิ้นงานแล้วเคลื่อนย้าย
	การยกและวางชิ้นงาน
แบบเป็นเส้นทางต่อเนื่อง (Continuous path)	การพ่นสีชิ้นงาน
	การเชื่อมประสาน
	การ Seal และ Glue

ตารางที่ 2.2 การแบ่งชนิดของหุ่นยนต์ตามการควบคุมการเคลื่อนที่

2.2 การหมุนของปลายมือ (Tool Orientation)

ดังที่กล่าวมาแล้วว่าเราใช้ 3 แกนหลักในการพิจารณารูปทรงของพื้นที่การทำงาน แกนที่เหลือจึงใช้ในการพิจารณาการหมุนของปลายมือ (Tool หรือ End effector) ถ้ามีแกนรองที่เป็นอิสระ 3 แกนทำให้ทูลสามารถหมุนได้อย่างไม่จำกัดในปริภูมิ 3 มิติ การหมุนของทูลสามารถแสดงได้ด้วยระบบยอว์-พิช-โรล (yaw-pitch-roll) หรือ RYP ดังรูปที่



รูปที่ 2.6 แสดงยอว์-พิช-โรล (yaw-pitch-roll) ของทูล(Tool)

ในการบ่งบอกทิศทางการหมุนของทูลโครงพิกัด M = { m^1,m^2 ,m^3 } จะถูกติดไว้ที่ Tool และโครงพิกัด (แกน 3 แกน ของ M) จะเคลื่อนที่ไปพร้อม ๆ กับ Tool จากรูปที่ 7 แกน m^3 อยู่ในแนวเดียวกันกับแกนของ Tool หรือแกน Roll และชี้ออกจากมือ แกน m^2 ขนานกับทิศทางที่ปลายมือเลื่อนปิด-เปิด (หักข้อมือขึ้น-ลง) ส่วนแกน m^1 เป็นไปตามกฎมือขวาของโครงพิกัด M

การแสดงการเคลื่อนที่แบบยอว์, พิชและโรล จะเป็นไปตามแกนที่อยู่นิ่งอยู่กับที่ เมื่อเริ่มต้นโครงพิกัดการเคลื่อนที่ของ Tool M จะทับอยู่กับโครงพิกัด F = { f^1 ,f^2 ,f^3 } ที่หยุดนิ่งอยู่กับที่ โดยจะติดกับส่วนปลายของปลายแขน (Forearm) yaw คือการหมุน Tool รอบแกนข้อมือ f^1 , pitch คือการหมุน Tool รอบแกนข้อมือ f^2, roll คือการหมุน Tool รอบแกนข้อมือ f^1 ดังแสดงได้จากตารางที่

2.3 ในทุกกรณีที่มีการหมุนมุมบวกจะเกิดขึ้นก็ต่อเมื่อมีการหมุนไปในทิศทางทวนเข็มนาฬิกาโดยมองจากปลายแขน (แกนที่หมุน) เข้าหาจุดกำเนิด

ลำดับของการหมุนยอว์-พิช-โรลมีความสำคัญมาก เพราะมีผลต่อลักษณะการวางตัวสุดท้ายของทูล ตัวอย่างเช่น หมุนยอว์ไป 90° ตามด้วยพิช 90° จะมีลักษณะการวางตัวสุดท้ายแตกต่างกับการหมุนแกนพิชก่อนเป็นมุม 90° แล้วตามด้วยยอว์ 90° เป็นต้น

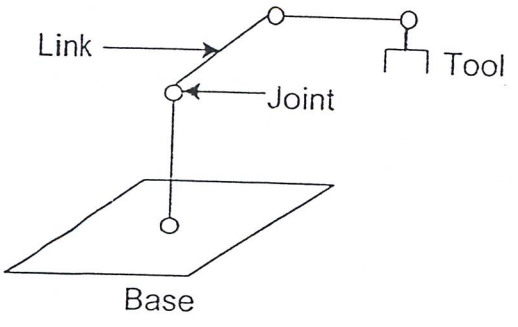
ขั้นตอน	การอธิบาย	แกน
1	Yaw	f^1
2	Pitch	f^2
3	Roll	f^3

ตารางที่ 2.3 การเคลื่อนที่แบบ ยอว์-พิชและโรล

2.3 สมการแขนกล

แขนกลมีลักษณะเป็นวัตถุเกร็ง (Rigid body) ซึ่งประกอบด้วย link หลาย ๆ อันนำมาเชื่อมต่อกันด้วยข้อต่อ (Joint) ดังรูปที่ 8 ปลายริมสุดข้างหนึ่งของลิงค์ (link) ที่นำมาต่อกันจะติดแน่นอยู่กับฐาน ส่วนปลายริมสุดอีกด้านหนึ่งสามารถเคลื่อนที่ได้อย่างอิสระ ปลายข้างที่เคลื่อนที่ได้นี้จะต่อเข้ากับเครื่องมือ (Tool or End-effector) ซึ่งทูล หรือเอนด์ เอฟเฟ็กเตอร์ นี้ก็เปรียบเสมือนปลายมือของแขนกลนั่นเองซึ่งใช้สำหรับทำงานที่เฉพาะเจาะจงลงไป โดยสามารถอยู่ในรูปของมือจับ, เครื่องเจาะ, เครื่องเชื่อมหรืออาจจะเป็นค้อนก็ได้

ข้อต่อของหุ่นยนต์ที่ใช้เชื่อมต่อลิงค์ มี 2 ชนิดดังที่ได้กล่าวมาแล้วคือ ข้อต่อหมุน (Revolute joint) และข้อต่อเลื่อน (Prismatic joint)



รูปที่ 2.7 แสดงการเชื่อมต่อลิงค์ (link) ของแขนกล

การแก้ปัญหาในสมการการเคลื่อนที่ของแขนกลเราจะแสดงในรูปของ **เมตริกซ์(Matrices)** โดยเราจะแบ่งปัญหาออกเป็น 2 รูปแบบด้วยกันคือ

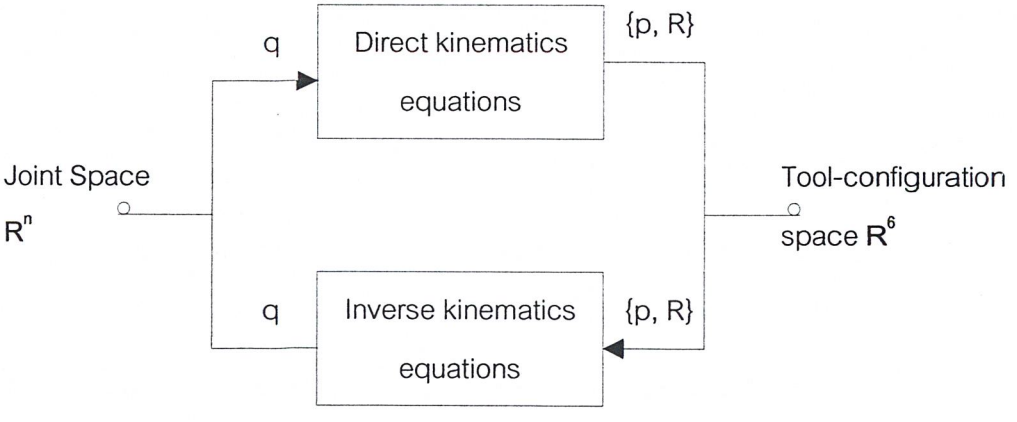
2.3.1 ไดเร็กไคเนมาติกส์ (Direct Kinematics) คือการกำหนดมุมหมุนและตำแหน่งของแต่ละข้อต่อของแขนกล เพื่อโปรแกรมให้เครื่องมือเคลื่อนที่ไปตามเส้นทางที่กำหนดไว้และทำการเคลื่อนย้ายวัตถุในพื้นที่ทำงาน โดยการโปรแกรมการเคลื่อนที่ของเครื่องมือนี้ จำเป็นจะต้องคำนวณความสัมพันธ์ระหว่างตัวแปรข้อต่อกับตำแหน่งการหมุนของอุปกรณ์หรือปลายมือ ในรูปของสมการเมตริกซ์

ปัญหาไดเร็กไคเนมาติกส์ (Direct Kinematics) คือการกำหนดเวกเตอร์ของตัวแปรข้อต่อ (Joint Variables) ของแขนกลโดยแยกตามลักษณะข้อต่อเป็น

- ข้อต่อหมุน ตัวแปรได้แก่ ค่าองศาการหมุนรอบแกนข้อต่อ
- ข้อต่อเลื่อน ตัวแปรได้แก่ ระยะการเคลื่อนที่ไปตามแนวแกนข้อต่อ

แล้วทำการคำนวณหาตำแหน่ง (Position) และการหมุน (Orientation) ของเครื่องมือเทียบกับโครงพิกัดที่ติดอยู่กับฐานของแขนกล

2.3.2 อินเวอร์สไคเนมาติกส์ (Inverse Kinematics) เป็นการพิจารณาย้อนกลับกับปัญหาแบบไดเร็กไคเนมาติกส์ คือการกำหนดตำแหน่งและการหมุนของปลายมือมาให้ แต่จะต้องทำการแก้สมการเมตริกซ์หาค่ามุมหมุนตามข้อต่อต่าง ๆ ปัญหาอินเวอร์สไคเนมาติกส์ มีความสำคัญและนำไปใช้งานจริงในแขนกลมากกว่าปัญหาแบบไดเร็กไคเนมาติกส์ แต่วิธีการแก้ปัญหาจะยุ่งยากและซับซ้อนมากกว่าเพราะไม่มีขั้นตอนการแก้ปัญหาที่แน่นอนและสามารถมีได้หลายคำตอบในการแก้ปัญหาหนึ่ง ๆ

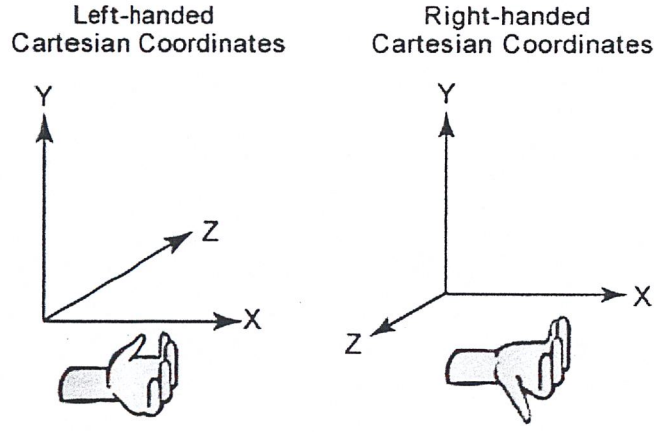


รูปที่ 2.8 แสดงความสัมพันธ์ระหว่างสมการแบบไดเร็กและอินเวอร์สไคเนมาติกส์ (Direct and inverse kinematics)

บทที่ 3
คณิตศาสตร์พื้นฐานสำหรับการพัฒนาโปรแกรม 3 มิติ

3.1 ระบบคู่อันดับในปริภูมิ 3 มิติ (3D Coordinate System)

ระบบคู่อันดับที่ใช้ในโปรแกรม 3 มิติ ส่วนใหญ่เป็นระบบคู่อันดับคาร์ทีเชียน (Cartesian) ซึ่งมี 2 ระบบด้วยกันคือระบบคู่อันดับมือซ้ายและระบบคู่อันดับมือขวา ระบบคู่อันดับทั้งสองนั้น แกนบวก X จะชี้ไปทางด้านขวามือและแกนบวก Y จะมีทิศชี้ขึ้นบนเสมอ ส่วนทิศทางของแกน Z มีทิศขึ้นอยู่กับระบบคู่อันดับว่าเป็นระบบคู่อันดับมือซ้ายหรือขวา ซึ่งสามารถหาทิศทางของแกนบวก Z ของระบบคู่อันดับมือซ้ายได้จากการกวาดนิ้วมือซ้าย (ยกเว้นนิ้วหัวแม่มือ) จากแกน X ไปหาแกนY ทิศทางของนิ้วหัวแม่มือที่พุ่งไปคือทิศทางของแกนบวก Z ที่เราต้องการทราบ ส่วนระบบคู่อันดับมือขวาก็มีวิธีการหาทิศทางของแกนบวก Z เช่นเดียวกันแต่เปลี่ยนจากการกวาดมือซ้ายมาเป็นมือขวาแทน ซึ่งสามารถแสดงได้ดังรูป



รูปที่ 3.1 แสดงระบบคู่อันดับมือซ้าย (Left-handed Cartesian Coordinates)และระบบคู่อันดับมือขวา (Right-handed Cartesian Coordinates)

3.2 จุดหรือเวกเตอร์ (Point or Vector)

จุด (Point) หรือ (Vector) เป็นข้อมูลที่มีความสำคัญมากในกราฟฟิกส์ 3 มิติ เนื่องจากข้อมูลต่าง ๆ ไม่ว่าจะเป็นเวอร์เท็กซ์ (Vertex) ซึ่งเป็นข้อมูลในการแสดงโมเดล (Model), ตำแหน่งและทิศทางของวัตถุ รวมทั้งความเร็วและแรง (Force) ทั้งหมดสามารถแทนค่าในรูปของจุดหรือเวกเตอร์

ในระนาบ 3 มิติได้ทั้งหมด โดยในปริภูมิ 3 มิติวัตถุประกอบด้วยความกว้าง (Width) คือค่าในแกน X, ความยาว (Height) คือค่าในแกน Y และความลึก (Depth) คือค่าในแกน Z

3.2.1การแสดงค่าของจุดหรือเวกเตอร์

3.2.1.1 การแสดงค่าของจุดหรือเวกเตอร์ในรูปคู่อันดับ

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

3.2.1.2 การแสดงค่าของจุดหรือเวกเตอร์ในรูปเมตริกซ์

$$v = [x \quad y \quad z]$$

เนื่องจากคำว่าจุดและเวกเตอร์มีความหมายไปในทางเดียวกัน คือ จุด หมายถึง ตำแหน่งในปริภูมิ 3 มิติ ส่วน เวกเตอร์ หมายถึง เส้นที่ลากจากตำแหน่งอ้างอิงหรือจุดกำเนิด (Origin) มายังตำแหน่งในปริภูมิ 3 มิติ เวกเตอร์จึงมีคุณสมบัติมากกว่าจุดคือ สามารถบอกทิศทางได้ ในที่นี้จึงสามารถใช้งานสลับกันได้ แต่ถ้าใช้ในการคำนวณจะกล่าวถึงในรูปของเวกเตอร์

3.2.2 การหาความยาวของเวกเตอร์

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

3.2.3 เวกเตอร์หนึ่งหน่วย (Unit vector)

เวกเตอร์หนึ่งหน่วย คือเวกเตอร์ที่มีขนาด 1 หน่วยเสมอไม่ว่าจะมีทิศทางใดก็ตาม การหาค่าเวกเตอร์หนึ่งหน่วยเป็นไปตามสมการด้านล่าง

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

มีเวกเตอร์หนึ่งหน่วยที่ใช้แสดงทิศทางในแนวแกนต่าง ๆ ดังนี้

แกน x : i = (1, 0, 0)

แกน y : j = (0, 1, 0)

แกน z : k = (0, 0, 1)

เนื่องจากเวกเตอร์มีการนำไปใช้มากในทางฟิสิกส์ซึ่งมักจะเขียนเวกเตอร์ในรูปของค่าเวกเตอร์หนึ่งหน่วย i, j, k ทำให้เราสามารถแสดงค่าจุดในปริภูมิ 3 มิติในรูปของเวกเตอร์ได้ โดยนำค่าในแต่ละแกนมาคูณกับเวกเตอร์หนึ่งหน่วยเหล่านี้ เช่น ถ้าต้องการแสดงจุด a = (3, 5, 2) ในรูปของเวกเตอร์จะได้

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

ซึ่งวิธีการแสดงค่าเวกเตอร์ในรูปเวกเตอร์หนึ่งหน่วยจะมีความสำคัญในการคำนวณค่ากับเมตริกซ์

3.2.4 ผลคูณเชิงสเกลาร์ (Scalar product or Dot product)

คือผลคูณของเวกเตอร์ 2 เวกเตอร์ที่ได้ผลลัพธ์เป็นสเกลาร์ ซึ่งค่าสเกลาร์ที่ได้แสดงค่าความยาวของเงาที่ฉายอยู่บนอีกเวกเตอร์หนึ่ง จะเห็นว่าผลคูณที่ได้จะบ่งบอกถึงขนาดแต่เพียงอย่างเดียวแต่ไม่ได้ระบุทิศทาง

3.2.4.1 ผลคูณเชิงสเกลาร์ในระนาบ 2 มิติ

$$u \bullet v = u_x v_x + u_y v_y$$

$$u \bullet v = \|u\| \bullet \|v\| \cos\theta$$

3.2.4.2 ผลคูณเชิงสเกลาร์ในระนาบ 3 มิติ

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

3.2.5 ผลคูณเชิงเวกเตอร์ของเวกเตอร์

คือผลคูณของเวกเตอร์ 2 เวกเตอร์ที่ได้ผลลัพธ์เป็นเวกเตอร์ ซึ่งเวกเตอร์ที่ได้จะตั้งฉากกับเวกเตอร์ทั้งสองเสมอ

$$u \times v = \left(u_y v_z \quad u_z v_y, u_z v_x \quad u_x v_z, u_x v_y \quad u_y v_x\right)$$

การคูณเชิงเวกเตอร์ไม่มีคุณสมบัติในการสลับที่ดังนั้น $u \times v \neq v \times u$ และถ้า 2 เวกเตอร์ขนานกันจะหาค่าผลคูณไม่ได้

3.3 ระนาบ (Plane)

เป็นข้อมูลพื้นฐานที่สำคัญอีกอย่างหนึ่ง ระนาบในปริภูมิ 3 มิติก็มีความหมายนัยเดียวกับเส้นตรงที่มีความยาวอนันต์ในปริภูมิ 2 มิติ หรือก็คือสิ่งที่อยู่ในมิติ n-1 นั่นเอง (เมื่อ n คือเลขจำนวนเต็มบวกแสดงค่ามิติ) ซึ่งจะแบ่งปริภูมิเป็น 2 ส่วนแยกจากกันโดยเด็ดขาด สามารถนำไปใช้ในการคำนวณงานหลาย ๆ อย่างใน 3 มิติ เช่น การหาว่าจุดหรือโพลีกอน (Polygon) อยู่ในระนาบเดียวกัน

หรือไม่, ใช้ในการแบ่งโพลีกอนที่อยู่ในบริเวณที่จะแแสดงผล (Clipping) และใช้ในการคำนวณหา BSP Tree เป็นต้น

สมการระนาบในปริภูมิ 3 มิติ คือ

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

นอกจากนั้นเรายังสามารถหาค่าอื่น ๆ ที่เกี่ยวข้องกับระนาบนี้ได้ ดังนี้

1. ค่า (a, b, c) คือเวกเตอร์ที่ตั้งฉากกับระนาบ
2. ค่า d คือระยะทางตั้งฉากจากระนาบกับจุดกำเนิด
3. เมื่อนำค่าจุดใด ๆ มาแทนค่าในตัวแปร x, y, z ของสมการระนาบปริภูมิสามมิติใด ๆ สามารถหาว่าจุดนั้นอยู่ในระนาบนั้น ๆ หรือไม่ โดยสมการดังกล่าวจะเป็นจริงถ้าจุดอยู่ในระนาบ

3.4 เมตริกซ์ (Matrix)

เมตริกซ์ช่วยให้เราสามารถแสดงสมการหลาย ๆ สมการได้พร้อม ๆ กัน เช่น ถ้าเราต้องการจะแก้สมการที่มีตัวแปร x, y, z ดังนี้

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

เมื่อนำค่าสัมประสิทธิ์ของสมการมาใส่ในกรอบขนาด m × n ที่เรียกว่า "เมตริกซ์" เมื่อ n คือจำนวนของคอลัมน์ (Vertical dimension) และ m คือจำนวนของแถว (Horizontal dimension) จะได้

$$\begin{aligned} 3x - 8y + 12z &= 0 \\ 15x + 14y - 2z &= 0 \\ 32x + 0.5y - z &= 0 \end{aligned} \quad \Rightarrow \quad \begin{bmatrix} 3 & -8 & 12 \\ 15 & 14 & -2 \\ 32 & 1.5 & 1 \end{bmatrix}$$

สัญลักษณ์ที่ใช้แทนเมตริกซ์ขนาด m × n เป็นดังนี้

$$A_{m\times n} = \lfloor a_{ij} \rfloor_{m\times n} = \begin{bmatrix} a_{11} & a_{12} & \cdot & \cdot & \cdot & a_{1n} \\ a_{21} & a_{22} & \cdot & \cdot & \cdot & a_{2n} \\ \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & & & & \cdot \\ \cdot & \cdot & & & & \cdot \\ a_{m1} & a_{m2} & \cdot & \cdot & \cdot & a_{mn} \end{bmatrix}$$

3.4.1 การบวกเมตริกซ์

นิยาม : ถ้า $A=[a_{ij}]_{m\times n}$ และ $B=[b_{ij}]_{m\times n}$ แล้ว $A+B=[a_{ij}+b_{ij}]_{m\times n}$ (โดย i คือตำแหน่งแถว และ j คือตำแหน่งหลัก) ตัวอย่างเช่น

$$\begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}+\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix}=\begin{bmatrix} a_{11}+b_{11} & a_{12}+b_{12} \\ a_{21}+b2 & a_{22}+b_{22} \end{bmatrix}$$

3.4.2 การคูณเมตริกซ์

นิยาม : ถ้า $A=[a_{ij}]_{m\times n}$ และ $B=[b_{ij}]_{m\times n}$ แล้วผลคูณคือ $A\times B$ คือเมตริกซ์ $C=[c_{ij}]_{m\times n}$ โดยที่ $c_{ij}=a_{il}b_{1j}+a_{il}b_{2j}+\ldots+a_{in}b_{nj}$ ตัวอย่างเช่น

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

การคูณเมตริกซ์ไม่มีคุณสมบัติในการสลับที่ คือ $A\times B\neq B\times A$ แต่มีคุณสมบัติในการเปลี่ยนกลุ่มได้ คือ $(A\times B)\times C=A\times(B\times C)$ ดังนั้น จึงต้องระวังในเรื่องการคูณเมตริกซ์

ตัวอย่างการคูณค่าของจุดกับเมตริกซ์

$$vA'=v'$$

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

เมตริกซ์มีการนำไปใช้งานในสาขาคอมพิวเตอร์กราฟฟิกส์อย่างมาก เนื่องจากการใช้เมตริกซ์ทำให้เราสามารถย้ายตำแหน่งของจุดหมุน หรือย่อขยายได้อย่างรวดเร็วด้วยการนำจุดมาคูณหรือบวกกับเมตริกซ์เท่านั้น แต่การใช้เมตริกซ์ 3×3 จะทำให้ได้ค่าที่ไม่ถูกต้องเนื่องจากความไม่เข้ากันของการคูณและการบวกในเมตริกซ์ ดังนั้นเพื่อแก้ปัญหานี้จึงต้องเพิ่มสมาชิกในเมตริกซ์ให้เป็นเมตริกซ์ขนาด 4×4 โดยสมาชิกที่เพิ่มเข้ามา เรียกว่าโฮโมจีเนียส โคออดิเนต (Homogenous coordinate) ซึ่งแทนด้วยอักษร w ตัวอย่างการเพิ่ม homogenous coordinate ให้กับค่าจุดในระบบคู่อันดับคาร์ทีเซียน

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

โดย b จะเป็นค่าใด ๆ ก็ได้ที่ไม่เท่ากับ 0 (ปกติจะใช้ b = 1)

3.4.3 การย้ายตำแหน่งจุด (Translation)

ให้ $p = [p_x \quad p_y \quad p_z]$ เป็นเวกเตอร์แทนระยะทางที่ต้องการย้ายตำแหน่งจุดไป จะได้เมตริกซ์สำหรับการย้ายจุด (Translation matix) 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

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

3.4.4 การหมุนตำแหน่งจุดรอบแกน (Rotation)

การหมุนจะต้องทำโดยอ้างอิงกับแกนหมุน ซึ่งก็คือ แกน x, แกน y, แกน z

- เมตริกซ์ที่ใช้ในการหมุน (Rotation matrix) รอบแกน x

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

- เมตริกซ์ที่ใช้ในการหมุนรอบแกน y

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

- เมตริกซ์ที่ใช้ในการหมุนรอบแกน z

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

โดยค่า θ คือมุมที่ต้องการหมุนจากแกนนั้น ๆ

3.4.4 การย่อขยายกลุ่มของจุด (Scale)

สามารถแสดงด้วยเมตริกซ์ ดังนี้

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

นอกจากการใช้เมตริกซ์เพื่อการย้ายตำแหน่งของจุด, การหมุนตำแหน่งของจุด และการย่อขยายตำแหน่งของจุดแล้วยังสามารถใช้ข้อมูลในเมตริกซ์สำหรับหาค่าอื่น ๆ ได้อีกดังนี้ ถ้าหากเรามีเมตริกซ์

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

โดย เวกเตอร์ p แสดงความสัมพันธ์ของวัตถุกับจุดกำเนิด

เวกเตอร์ n, o และ a แสดงทิศทางในแต่ละแกนหรือในรูปเวกเตอร์หนึ่งหน่วย

3.5 ควอเตอร์เนียน (Quaternion)

ควอเตอร์เนียน คือตัวกำหนดแกน ในปริภูมิ 3 มิติและเป็นตัวกำหนดการหมุนรอบแกนนั้น ๆ โดยควอเตอร์เนียนจะประกอบด้วยสมาชิก 4 ตัวซึ่งมาจาการแพิ่มสมาชิกอีก 1 ตัวเข้าไปในค่าของเวกเตอร์ $[x, y, z]$

ควอเตอร์เนียนช่วยอำนวยความสะดวกในกระบวนการหมุนตำแหน่งของจุด และมีความเร็วมากกว่าการคำนวณโดยใช้เมตริกซ์ ประกอบด้วยองค์ประกอบ 2 ส่วน คือ ค่าเวกเตอร์และค่าสเกลาร์ ควอเตอร์เนียนสามารถเขียนในรูปต่าง ๆ ได้ดังนี้

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

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

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

$$q = [sv]$$

เมื่อ x, y, z เป็นค่าสัมประสิทธิ์ของเวกเตอร์หนึ่งหน่วย

w เป็นค่าสเกลาร์

$s = w$ และ $v = [x \quad y \quad z]$

บทที่ 4
พื้นฐานกราฟฟิกส์ 3 มิติ

เมื่อต้องนำรูปภาพ 3 มิติที่สร้างขึ้นมาใช้งานร่วมกับโปรแกรม 3 มิติ ก่อนอื่นต้องมีความรู้เกี่ยวกับพื้นฐานทางด้าน 3 มิติก่อน เพราะคำศัพท์บางคำที่ใช้อยู่ในโปรแกรม 3 มิติ จะถูกนำมาใช้ร่วมกับการเขียนโปรแกรมเชื่อมต่อกับไดเร็กเอ็กซ์ด้วย ซึ่งถ้าหากไม่มีความรู้พื้นฐานเหล่านี้อาจจะเกิดปัญหาในการเขียนโปรแกรมที่ดีได้ โดยพื้นฐานทางด้าน 3 มิติที่ควรทราบมีดังต่อไปนี้

4.1 เวอร์เท็กซ์ (Vertex)

เวอร์เท็กซ์ คือ จุด (Point) ซึ่งแต่ละเวอร์เท็กซ์จะเชื่อมต่อกันด้วยเส้น (Line) และเมื่อทุก ๆ เวอร์เท็กซ์ มีการเชื่อมต่อกันหมดแล้วจะทำให้เกิดรูปทรง (Shape) ขึ้น

รูปที่ 4.1 แสดงลักษณะของเวอร์เท็กซ์ (Vertex)

4.2 เฟซ (Face)

เฟซ คือเซ็ท (Set) ของเวอร์เท็กซ์ที่มีตั้งแต่ 3 จุดขึ้นไป มาเชื่อมต่อกันในลักษณะของรูปทรงต่าง ๆในแนวระนาบ เวอร์เท็กซ์เป็นตัวกำหนดมุมของเฟซ ซึ่งทำให้ทุก ๆ เวอร์เท็กซ์ในเฟซจะต้องถูกกำหนดให้อยู่ในแนวระนาบ

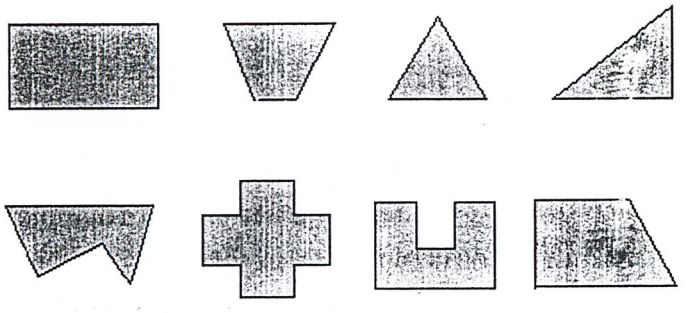
4.3 เมช (Mesh)

เมช เป็นการรวมกันของเฟซที่เชื่อมต่อกัน ซึ่ง 1 เมชสามารถมีเฟซได้ตั้งแต่ 1 เฟซขึ้นไป การรวมของเฟซนี้ทำให้ง่ายต่อการจัดการวัตถุในการทำแอนนิเมชั่น (Animation), แมตทีเรียล (Material) และเท็กซ์เจอร์ (Texture)

4.4 โพลีกอน (Polygon)

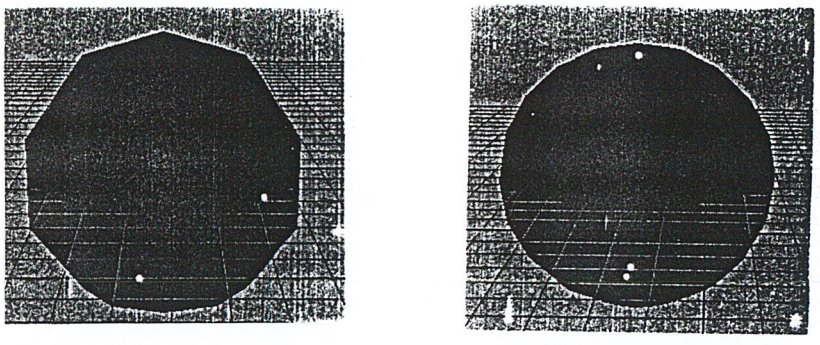
โพลีกอน คือรูปทรงที่สร้างขึ้นโดยการวาดเส้นเชื่อมระหว่างเซตของเวอร์เท็กซ์ โดยที่ 1 โพลีกอน ต้องมีด้านประกอบอย่างน้อย 3 ด้าน

Direct3D ใช้โพลีกอนในการสร้างภาพของวัตถุ ซึ่งโพลีกอน ก็คือพื้นผิวที่เกิดจากเส้นรอบรูปที่มีลักษณะเป็นวงปิด (โพลีกอนสนใจเฉพาะพื้นผิวไม่สนใจปริมาตรของวัตถุ) ซึ่งเส้นรอบรูปของโพลีกอนเป็นรูปทรงเรขาคณิต เช่น สามเหลี่ยม, สี่เหลี่ยม เป็นต้น หลักการในการสร้างรูปทรงของวัตถุโดยใช้โพลีกอนก็คือ การใช้พื้นผิวพื้นฐานที่สุด (Primitive) ประกอบกันในการสร้างรูปทรงต่างตั้งแต่อย่างง่ายไปจนถึงอย่างที่ซับซ้อนมาก ฉะนั้นพื้นผิววงปิดที่พื้นฐานที่สุดก็คือ รูปสามเหลี่ยมนั่นเอง ดังนั้นโดยอาศัยหลักการของโพลีกอน เราสามารถสร้างพื้นผิวของวัตถุที่มีรูปร่างใด ๆ ก็ได้โดยไม่มีข้อจำกัด



รูปที่ 4.2 แสดงลักษณะของโพลีกอน

ในการวาดรูปที่มีส่วนโค้งเราไม่นิยมที่จะวาดส่วนโค้งนั้นโดยตรงแต่เรานิยมที่จะนำโพลีกอนหลาย ๆ โพลีกอน มาเรียงต่อกันเพื่อให้ได้ส่วนโค้ง เนื่องจาคอมพิวเตอร์ถนัดที่จะคำนวณเส้นตรงมากกว่าเส้นโค้ง การคำนวณเส้นโค้งมันจะทำได้ช้าและอาจเกิดข้อผิดพลาดได้ง่าย จึงนิยมใช้โพลีกอนมาเรียงกันมากกว่า และยิ่งใช้โพลีกอนจำนวนมากเท่าใดยิ่งทำให้ส่วนโค้งนั้นดูสมจริงมากเท่านั้น



รูปที่ 4.3 ลักษณะวัตถุทรงกลมที่มีจำนวนโพลีกอนแตกต่างกัน

4.5 เชดดิ้ง โหมด (Shading mode)

เชดดิ้งโหมดใช้สำหรับกำหนดลักษณะภาพของวัตถุที่ปรากฏเมื่อทำการเรนเดอร์ เชดดิ้งโหมดแบ่งออกเป็น 2 โหมดด้วยกันโดยจะมีความแตกต่างกันตรงที่ความเข้มของสีและการให้แสงที่จุดใด ๆ ก็ตามบนพื้นผิวของโพลีกอน

4.5.1 แฟลต เชดดิ้ง (Flat shading) เป็นโหมดในการเรนเดอร์โพลีกอน โดยใช้สีของเวอร์เท็กซ์จุดแรกที่สะท้อนแสงไฟในโพลีกอนใดโพลีกอนหนึ่งเป็นสีของทุก ๆ โพลีกอนบนวัตถุ

4.5.2 กอรอด เชดดิ้ง (Gouraud shading) จะทำการเรนเดอร์ภาพโดยคำนวณสีของแต่ละเวอร์เท็กซ์โดยใช้เวอร์เท็กซ์ นอมอลกับตัวแปรแสง จากนั้นจะทำการคำนวณแทรกสอด (Interpolate) ค่าสีระหว่างเฟซของแต่ละโพลีกอน

Flat shading Gouraud shading

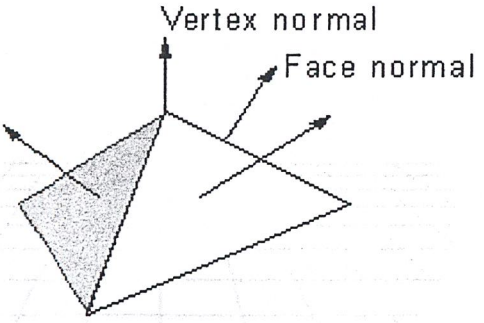
รูปที่ 4.4 เปรียบเทียบลักษณะภาพในโหมดการเรนเดอร์ที่แตกต่างกัน

4.6 นอมอล เวกเตอร์ (Normal vector)

เป็นชนิดของยูนิตเวกเตอร์ใช้ในการคำนวณแสงต่าง ๆ ของเฟซและเมช โดยนอมอลมี 2 ชนิดคือ

4.6.1 เฟซ นอมอล (Face normal) คือยูนิตเวกเตอร์ของเฟซ โดยจะตั้งฉากกับเฟซและมีทิศทางชี้ออกมาจากด้านหน้าของเฟซ โดยด้านหน้าของเฟซเท่านั้นที่เราสามารถมองเห็นได้ ซึ่งด้านหน้าของเฟซจะขึ้นอยู่กับเวอร์เท็กซ์ที่ประกอบกันขึ้นเป็นเฟซและระบบคู่อันดับในปริภูมิ 3 มิติว่าเป็นระบบคู่อันดับมือซ้ายหรือมือขวา และเฟซ นอมอลจะถูกใช้ในโหมดแบบแฟลท เชดดิ้ง

4.6.2 เวอร์เท็กซ์ นอมอล (Vertex normal) คือยูนิตเวกเตอร์ที่เชื่อมกับเวอร์เท็กซ์และสามารถถูกคำนวณได้โดยหาค่าเฉลี่ยของเฟซ นอมอลกับเวอร์เท็กซ์ เวอร์เท็กซ์ นอมอลมักใช้ในการคำนวณหาค่าสีของเวอร์เท็กซ์ Direct3D ใช้เวอร์เท็กซ์ นอมอลในการเรนเดอร์แบบกอรอด เชดดิ้ง และยังนำไปประมวลผลเพื่อสร้างภาพที่มีความสมจริงยิ่งขึ้นได้ เช่น นำไปคำนวณมุมที่แสงกระทำต่อวัตถุทำให้ได้วัตถุที่มีสีสันเหมือนจริงยิ่งขึ้น



รูปที่ 4.5 แสดงเฟซ นอมอล (Face normal) และเวอร์เท็กซ์ นอมอล (Vertex normal) ของวัตถุ

บทที่ 5
โปรแกรม Microsoft Visual C ++ การเขียนโปรแกรมบนระบบปฏิบัติการวินโดว์ และการใช้งาน STL

5.1 โปรแกรม Microsoft Visual C++

OOP เป็นวิธีการเขียนโปรแกรมโดยอาศัยแนวความคิดแบบโครงสร้าง (Structure) การเขียนโปรแกรมแบบ OOP มีความสามารถในการปกป้องข้อมูลหรือซ่อนข้อมูล (Data hiding), มีคุณสมบัติในการรวมข้อมูลเข้ากับฟังก์ชั่นหรือเอนเคปซูเลชัน (Encapsulation) และคุณสมบัติการสืบทอด (Inheritance)

5.2 การเขียนโปรแกรมบนระบบปฏิบัติการวินโดวส์

โปรแกรมในวินโดวส์กับโปรแกรมในดอส (DOS) มีความแตกต่างกันหลายประการคือ ในวินโดวส์เราสามารถรันได้หลายโปรแกรมพร้อมกันโดยการใช้รีซอร์ส (Resource -- ส่วนประกอบทั้งหมดที่มีอยู่ในคอมพิวเตอร์เครื่องนั้น) ร่วมกัน แต่โปรแกรมในดอสจะรัน (Run) ได้เพียงครั้งละ 1 โปรแกรม โปแกรมในวินโดวส์สื่อสารกับผู้ใช้โปรแกรมแบบกราฟฟิกส์ แต่โปรแกรมในดอสสื่อสารกับผู้ใช้โปรแกรมแบบเท็กซ์ (Text Mode) ในวินโดวส์จะติดต่อกับอุปกรณ์ชนิดต่าง ๆ ของคอมพิวเตอร์ได้โดยอัตโนมัติ เช่นทำการตรวจสอบอุปกรณ์ที่มีอยู่ในเครื่องเป็นต้น ส่วนโปรแกรมในดอสจะติดต่อได้แต่เฉพาะกับชนิดของอุปกรณ์ตามที่กำหนดไว้เท่านั้น ดังนั้นการเขียนโปรแกรมบนระบบปฏิบัติการดอสจะมีความยากลำบากมาก เนื่องจากอุปกรณ์คอมพิวเตอร์มีจำนวนมากมาย ฉะนั้นการเขียนโปรแกรมในดอสจะมีการทำงานที่สลับซับซ้อนกว่าในวินโดวส์มาก ซึ่งการเขียนโปรแกรมในที่นี้จะเป็นการเขียนโปรแกรมแบบ MFC โดยใช้ MFC AppWizard (.EXE version) ซึ่ง AppWizard จะเป็นตัวสร้างซอร์สโค๊ดต้นแบบขึ้นมา เพื่อให้การพัฒนาโปรแกรมนั้นสามารถทำได้ง่ายและรวดเร็วยิ่งขึ้น โดยจะเรียกใช้คลาสจาก MFC เป็นหลัก

หน้าต่างแอพพลิเคชั่นที่สร้างขึ้นในโปรแกรมนี้ (User Interface) อยู่ในรูปแบบของ SDI (Single Document Interface) หรือมีลักษณะคล้ายโน๊ตแพด (Notepad) และกำหนดรูปแบบคลาส View เป็นแบบ CFormView ซึ่งมีลักษณะการอินเตอร์เฟสเป็นแบบเฟรมวินโดวส์ที่มีทูลบาร์มีสเตตัสบาร์และสามารถสร้างปุ่มกดหรือบรรจุคอนโทรลต่าง ๆ ลงไปได้

ส่วนรายละเอียดปลีกย่อยของฟังก์ชันและการแม็ปเมสเสจ (Massage map) ที่ใช้เขียนโปรแกรมในส่วนของยูสเซอร์ อินเตอร์เฟส (User Interface) ผู้จัดทำจะไม่ขอกล่าวในที่นี้เนื่องจากเป็นคำสั่งทั่วไปที่มีการเรียกใช้งานบ่อยอยู่แล้ว

5.3 Standard Template Library (STL)

Standard Template Library คือ กลุ่มของคลาส (Set of class) และฟังก์ชัน (Function) ที่ช่วยในการเก็บข้อมูล (Collective of data) เช่น dynamic arrays (vector) และช่วยในการทำอัลกอริทึม (Algorithm) ง่าย ๆ เช่น การเรียงลำดับ (sorting) ซึ่งมีการกำหนดเป็นมาตรฐานโดย ANSI/ISO C++ Standards Committee ในปี 1994

STL ถูกออกแบบมาโดยคำนึงถึงความเร็วในการทำงานเป็นอันดับแรกจึงไม่ต้องกังวลเลยว่าการนำเอา STL มาใช้จะทำให้การทำงานของโปรแกรมที่เขียนขึ้นล่าช้าลง

STL มีจำนวนมากและมีบางส่วนที่มีความซับซ้อน การใช้ STL ให้ได้อย่างมีประสิทธิภาพจึงจำเป็นต้องเข้าใจองค์ประกอบพื้นฐานและวิธีการใช้งานองค์ประกอบเหล่านี้เสียก่อน

5.3.1 องค์ประกอบพื้นฐานของ STL (Basic components of STL)

5.3.1.1 คอนเทนเนอร์ (Containers) คือ STL template class ซึ่งจัดการในการเก็บข้อมูล ข้อมูลที่เก็บจะเป็นข้อมูลชนิดใด ๆ ก็ได้ที่มีดีฟอลท์ คอนสตรัคเตอร์ (default constructor), ดีสตรัคเตอร์ (destructor) และแอสซายน์เมนท์ โอเปอเรเตอร์ (assignment operator) ซึ่งแน่นอนว่าจะต้องมีความแตกต่างกันในการใช้งานคอนเทนเนอร์ในแต่ละชนิด

5.3.1.2 ไอเทอเรเตอร์ (Iterators) เปรียบเสมือนพอยท์เตอร์ (Pointer) ที่ชี้ไปยังข้อมูลที่อยู่ในคอนเทนเนอร์ นอกจากนั้นแล้วใน STL ยังใช้สัญลักษณ์เดียวกับที่ใช้ในพอยท์เตอร์อีกด้วย เช่น ในโอเปอเรเตอร์ (Operator) + - ในการย้ายไอเทอเรเตอร์ ไปยังข้อมูลตัวถัดไปในคอนเทนเนอร์ เรียกว่า อินครีเมนท์ (Increment) นอกจากนั้นการเข้าถึงข้อมูลก็ใช้สัญลักษณ์ "*" เช่นเดียวกับพอยท์เตอร์ ซึ่งเรียกการเข้าถึงข้อมูลชนิดนี้ว่าดีเรฟเฟอเรนท์ (Dereference)

5.3.1.3 อัลกอริทึม (Algorithms) ที่ทำงานกับ STL ไม่ได้อยู่ในรูปของเมมเบอร์ฟังก์ชัน (Member function) ของคอนเทนเนอร์ คลาส (Container class) แต่อยู่ในรูปของสแตนด์อโลน ฟังก์ชัน (Stand-alone function) ซึ่งทำงานอยู่บนไอเทอเรเตอร์ การแยกส่วนที่ทำหน้าที่ในการเก็บข้อมูล (Containers) ออกจากอัลกอริทึมทำให้จำนวนของอัลกอริทึมที่ต้องเขียนขึ้นเฉพาะสำหรับแต่ละคอนเทนเนอร์ลดลงได้ เนื่องจากคอนเทนเนอร์ใดที่มีไอเทอเรเตอร์คล้ายคลึงกันสามารถใช้อัลกอริทึมร่วมกันได้

5.3.2 แนวคิดของ STL (STL Concepts)

หลักการต่อไปนี้เป็นสิ่งสำคัญในการใช้งาน STL สิ่งแรกที่ต้องรู้ก็คือฟังก์ชันที่ให้หาช่วงของข้อมูล (จุดที่เริ่มเก็บข้อมูลถึงจุดสุดท้ายที่มีการเก็บข้อมูล) เมื่อใช้งานคอนเทนเนอร์ การ

หาช่วงข้อมูลมี 2 วิธีการ (Method) ที่สำคัญ คือ begin() และ end() ด้วยการใช้ 2 วิธีการนี้สามารถหาช่วงข้อมูลทั้งหมดที่มีอยู่ในคอนเทนเนอร์ได้โดย begin() และจะคืนค่า (return) ค่าตำแหน่งของข้อมูลตัวแรกในคอนเทนเนอร์และ end() จะคืนค่าตำแหน่งสุดท้ายของข้อมูลที่สามารถใช้งาน ข้อดีของการจัดการช่วงของข้อมูลในลักษณะนี้คือ ไม่ต้องเขียนโค๊ด (Code) มาจัดการกับช่วงของข้อมูลที่ไม่ได้ใช้งาน และสามารถนำไปใช้กับไอเทอเรเตอร์ที่ทำงานกับทั้งคอนเทนเนอร์ได้โดยการกำหนดให้ทำงานจนถึงค่าที่รีเทอร์น (Return) มาจาก method end() ส่วนข้อเสียก็คือ การทำงานที่ไม่เป็นไปตามลำดับการทำงานของไอเทอเรเตอร์ ต้องใช้ตัวแปรหรือไอเทอเรเตอร์พิเศษเข้าช่วย

นอกจากนั้นยังมีสิ่งที่ต้องคำนึงถึงอีกอย่างในการใช้งาน STL ก็คือ STL container ซึ่งถูกออกแบบมาให้ส่งค่าอาร์กิวเมนท์แบบบายแวลู (Passed argument by value) ไม่ใช่แบบบายเรฟเฟอร์เรนท์ (Passed argument by reference) ดังนั้นถ้าใช้งาน STL กับข้อมูลขนาดเล็กก็จะไม่มีปัญหาแต่ถ้าจะนำไปใช้กับข้อมูลขนาดใหญ่ก็ควรใช้กับพอยท์เตอร์ของคลาส (Class) หรือสตรัค (Struct) จะดีกว่า

5.3.3 คอนเทนเนอร์พื้นฐานใน STL ที่ใช้งานบ่อย

5.3.3.1 เวกเตอร์ (Vector)

เวกเตอร์ใน STL หมายถึง อาร์เรย์ (Array) ที่สามารถเปลี่ยนแปลงขนาดได้ (Resizable array) ซึ่งแตกต่างจาก C++ อาร์เรย์ที่ไม่สามารถเปลี่ยนแปลงขนาดได้ อย่างไรก็ตามมีความจำเป็นอย่างยิ่งที่จะต้องทำความเข้าใจวิธีการในการเปลี่ยนแปลงขนาดของเวกเตอร์ด้วย

เวกเตอร์สร้างขึ้นโดยใช้อาร์เรย์ ดังนั้นในการเปลี่ยนแปลงขนาดจึงมีความจำเป็นที่จะต้องจองหน่วยความจำใหม่ (Reallocate memory) แล้วจึงย้ายข้อมูลไปยังอาร์เรย์ใหม่ที่สร้างขึ้น หลังจากนั้นจึงปล่อย (Release) หน่วยความจำเดิม แต่เราก็สามารถจองพื้นที่หน่วยความจำให้มีขนาดใหญ่กว่าที่ต้องการใช้งานได้ เพื่อที่จะเหลือพื้นที่สำหรับในกรณีที่ต้องการขยายขนาดซึ่งจะทำให้การทำงานเร็วขึ้นเนื่องจากไม่ต้องจองหน่วยความจำและย้ายข้อมูลบ่อย ๆ

5.3.3.2 ลิสต์ (List)

ลิสต์ใน STL สร้างขึ้นในแบบดับบลี ลิงค์ ลิสต์ (Doubly linked list) ดังนั้นการแทรกหรือลบข้อมูลที่ใด ๆ ก็ตามจึงสามารถทำได้ในเวลาอันรวดเร็ว แต่จะสูญเสียความสามารถในการเข้าถึงข้อมูลแบบสุ่ม (Random access) ที่มีในเวกเตอร์และดีคิว (Deque)

5.3.3.3 ดีคิว (Deque)

ดีคิวหรือดับเบิล-เอ็นด์ คิว (Double-ended queue) ได้รับการออกแบบมาสำหรับใช้ในกรณีที่ต้องการความสามารถในการแทรกหรือดึงข้อมูลออกได้จากปลายทั้งสองด้าน

ของคอนเทนเนอร์ (ไม่มีความต้องการที่จะแทรกหรือดึงข้อมูลที่อยู่ตรงกลางออกแต่จริง ๆ แล้วก็สามารถทำได้) เช่นเดียวกับเวกเตอร์ที่สามารถเข้าถึงข้อมูลในดีคิวแบบสุ่มได้ แต่ประสิทธิภาพจะต่ำกว่าการเข้าถึงข้อมูลในเวกเตอร์ เนื่องจากความซับซ้อนของข้อมูลภายในของดีคิว ซึ่งเกิดจากการเรียงลำดับการเชื่อมต่อของเมมโมรี่ บล็อก (Memory block)

5.3.3.4 แม็ป (Maps)

เป็นแวลู-แพริง คอนเทนเนอร์ (Value-pairing container) หมายความว่ามีข้อมูล 2 ชนิดที่จะต้องใช้คู่กันอยู่เสมอโดยมีโครงสร้างเป็นคีย์/แวลู (Key/value) แล้วจึงแทรกเข้าไปในคอนเทนเนอร์ สามารถหาค่าในคอนเทนเนอร์ได้โดยหาจากค่าของคีย์ (Key) ซึ่งจะหาได้ในเวลา O (Log n) ถึงแม้ว่าจะมีประสิทธิภาพไม่ดีเท่ากับการใช้แฮ็ช เทเบิล (Hash table) แต่มีข้อดีเนื่องจากมีความสามารถในการเรียงลำดับข้อมูลระหว่างการแทรกข้อมูลอันเป็นผลมาจากวิธีการที่ใช้ในการเก็บข้อมูล (Balanced binary tree หรือ red-black tree)

5.3.3.5 สแต็ค (Stacks)

สแต็คมีฟังก์ชันสำคัญ 3 ฟังก์ชัน คือ push(), pop() และ top() สำหรับใช้ในการเพิ่มและลบข้อมูลออกจากคอนเทนเนอร์ โดยค่าเริ่มต้นแล้วสแต็คสร้างขึ้นโดยใช้ดีคิว แต่เราสามารถเปลี่ยนชนิดของคอนเทนเนอร์ที่ใช้สร้างสแต็คได้โดยการกำหนดค่าในคอนสตรัคเตอร์ (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;
```

การใช้เวกเตอร์แทนการใช้ดีคิวสามารถทำได้เนื่องจากสแต็คสามารถสร้างขึ้นจากคอนเทนเนอร์ใด ๆ ก็ตามที่มีฟังก์ชัน push_back(), pop_back() และ back() เนื่องจากฟังก์ชัน push(), pop() และ top()ของสแต็คถูกจับเข้าคู่ (Map) กับฟังก์ชันเหล่านี้

เช่นเดียวกับสแต็ค คืออย่าลืมว่า STL container ส่วนใหญ่สร้างขึ้นโดยคำนึงถึงความเร็วเป็นอันดับแรก ดังนั้นก่อนจะใช้คำสั่ง pop() และ top() ต้องแน่ใจว่ามีข้อมูลอยู่ในสแต็ค โดยใช้ฟังก์ชัน size() หรือ empty() ในการตรวจสอบว่าสแต็คมีข้อมูลอยู่หรือไม่

5.3.3.6 คิว (Queues)

คิวทำงานคล้ายกับสแต็ค ยกเว้นข้อมูลจะถูกใส่จากด้านท้ายและดึงออกได้จากด้านหน้า มีฟังก์ชันที่สำคัญได้แก่ push(), pop(), front() และ back() โดย back() ใช้อ้างอิงถึงข้อมูลที่เพิ่งเพิ่มเข้าไปในคิวและ front() ใช้อ้างอิงถึงข้อมูลที่อยู่ด้านหน้าสุดของคิวซึ่งจะถูกดึงออก

เช่นเดียวกับสแต็คนั่นคือจะต้องใช้ฟังก์ชัน size() และ empty() ในการควบคุมขนาดของคิว นอกจากนั้นยังสามารถกำหนดชนิดของคอนเทนเนอร์ที่ใช้สร้างได้เช่นเดียวกับสแต็คด้วย แต่การใช้เวกเตอร์ในการสร้างคิวนั้นจะให้ประสิทธิภาพไม่ดีเนื่องจากการแทรกข้อมูลจากด้านหน้าของเวกเตอร์สไม่มีประสิทธิภาพนัก

5.3.3.7 ไพรออริตี้ คิว (Priority_queue)

ทำงานคล้ายกับคิว แตกต่างกันตรงที่ข้อมูลที่แทรกเข้ามาจะได้รับการเรียงลำดับจากมากไปหาน้อย โดยใช้ เครื่องหมาย " < " (Less-than operator) แต่เนื่องจากฟังก์ชันในการเรียงลำดับสามารถกำหนดได้โดยใช้คอนสรัคเตอร์ (เป็นพารามิเตอร์ตัวที่สาม) เช่นกัน เราจึงสามารถกำหนดรูปแบบการเรียงลำดับตามต้องการได้

5.3.4 การใช้งาน STL

Standard C++ library ทุกตัวถูกประกาศ (Declare) หรือนิยาม (Define) ไว้ในสแตนดาร์ด เฮดเดอร์ (Standard header) การใช้งานไลบรารีเหล่านี้จึงต้องทำการอินคลูด (Include) สแตนดาร์ด เฮดเดอร์ของสแตนดาร์ด ไลบรารีที่ต้องการใช้งานด้วย ซึ่ง STL ที่ใช้งานในโปรแกรมจำลองแขนกลบนคอมพิวเตอร์จะอยู่ในสแตนดาร์ด เฮดเดอร์ ดังนี้ <algorithm>, <list>, <map>, <queue>, <stack>, <string> และ <vector>

ตัวอย่างการอินคลูด สแตนดาร์ด เฮดเดอร์ (Include standard header)

```
// include vector facilities
#include <vector>
```

จะเห็นว่าการอินคลูด สแตนดาร์ด เฮดเดอร์ จะแตกต่างจากการอินคลูดไฟล์เฮดเดอร์ทั่วไปเนื่องจากไม่มี ".h" นอกจากนั้นการใช้งาน STL ควรจะประกาศเนมสเปซ (Namespace) ด้วย เนื่องจากชื่อทุกชื่อใน Standard C++ header นิยามโดยใช้ std namespace ซึ่งถ้าไม่ประกาศเนมสเปซแล้วการใช้งาน STL จะต้องเขียนโค๊ดในรูปของ std::vector แต่ถ้าประกาศแล้วเขียนเพียงเวกเตอร์ก็สามารถใช้งานได้แล้ว ซึ่งการประกาศใช้เนมสเปซทำได้ดังนี้

```
using namespace std;
```

บทที่ 6

Direct X

6.1 แนะนำไดเร็กเอ็กซ์

ไมโครซอฟไดเร็กเอ็กซ์ (Microsoft DirectX) คือชุดของแอพพลิเคชั่นโปรแกรมมิ่งอินเทอร์เฟส (Application Programming Interface : API) ในระดับต่ำ (low-level) ที่ได้รับการออกแบบมาสำหรับการสร้างเกมและแอพพลิเคชั่นทางด้านมัลติมีเดีย (Multimedia) ประสิทธิภาพสูงอื่น ๆ สนับสนุนการทำงานกราฟฟิกส์แบบ 2 มิติ (2-Dimension : 2-D), 3 มิติ (3-Dimension : 3-D), เสียงเอฟเฟ็กต์ (Sound effect), อุปกรณ์รับข้อมูลและสนับสนุนการใช้งานแอพพลิเคชั่นบนเน็ตเวิร์ค (Network) เช่นเกมประเภทผู้เล่นหลายคน (Multiplayer game)

ดังนั้นถ้ากล่าวให้ละเอียดขึ้นไปอีกไดเร็กเอ็กซ์ จะหมายถึง ไลบรารี คำสั่ง (Run Time Library) ที่ช่วยงานด้านมัลติมีเดียกราฟฟิกส์ โดยตัวไดเร็กเอ็กซ์ฟาวเดชัน (Direct X Foundation) จะมีส่วนประกอบที่เรียกว่า HAL (Hardware Abstraction Layer) ซึ่งจะใช้ซอฟแวร์ในการตรวจสอบความสามารถของฮาร์ดแวร์ที่อยู่ในเครื่องคอมพิวเตอร์นั้นอย่างอัตโนมัติ แล้วนำมากำหนดค่าพารามิเตอร์ของแอพพลิเคชั่นให้ตรงตามความเหมาะสมระหว่างเกมคอมพิวเตอร์ หรือโปรแกรมมัลติมีเดียกับไดรเวอร์ (Driver) ของฮาร์ดแวร์ที่มีส่วนเกี่ยวข้องกับเกมหรือโปรแกรมนั้น ทำให้การประมวลผลโปรแกรมทำได้เร็วขึ้น เพราะไม่ต้องอาศัยตัวกลางเช่น GDI (Graphic Device Interface) ทำให้เราเขียนโปรแกรมสื่อสารเฉพาะกับ Direct X ก็เพียงพอแล้ว

นอกจากนี้ Direct X Foundation ยังมีส่วนประกอบที่เรียกว่า HEL (Hardware Emulation Layer) ทำให้เราสามารถใช้โปรแกรมมัลติมีเดีย หรือโปรแกรมที่ข้องกับ 3D บนฮาร์ดแวร์ที่ไม่สนับสนุนการใช้งานทางด้าน 3 มิติได้ โดยจะทำการจำลองความสามารถบางอย่างที่ฮาร์ดแวร์ตัวนั้นไม่มีให้สามารถใช้งานได้กับโปรแกรมที่ต้องการ แม้จะมีข้อเสียตรงที่อาจทำให้การทำงานล่าช้าลงบ้างก็ตาม

6.2 ความจำเป็นของไดเร็กเอ็กซ์

เนื่องจากปัจจุบันระบบปฏิบัติการวินโดวส์มีการใช้งานกันมากที่สุดในเครื่องคอมพิวเตอร์ประเภทส่วนบุคคลหรือ PC (Personal Computer) และเป็นตลาดของผลิตภัณฑ์เพื่อความบันเทิงที่มีขนาดใหญ่ แต่กลับเกิดข้อจำกัดขึ้นเนื่องจากความสามารถในการพัฒนาอุปกรณ์แตกต่างกันถึงแม้ว่าเป็นอุปกรณ์ที่ทำหน้าที่อย่างเดียวกัน นั่นคือไม่มีมาตรฐานในการพัฒนาอุปกรณ์ เพื่อแก้ปัญหานี้ไดเร็กเอ็กซ์จึงได้รับการออกแบบมาเพื่อให้เป็นมาตรฐานสำหรับการติดต่อกับอุปกรณ์ด้วยซอฟแวร์

(Standard software interface) ทำให้ผู้พัฒนาซอฟแวร์สามารถทำงานได้ง่ายขึ้น เนื่องจากไม่ต้องทำการทดสอบกับอุปกรณ์ทุกชนิดที่มีอยู่ในท้องตลาด เพียงแต่ทดสอบให้สามารถทำงานกับไดเร็กเอ็กซ์ได้ก็เพียงพอแล้ว ส่วนผู้ผลิตฮาร์ดแวร์ก็มีหน้าที่จะต้องสร้างไดรเวอร์ของอุปกรณ์ (Device driver) ให้รองรับกับไดเร็กเอ็กซ์

6.3 ชุดพัฒนาซอฟแวร์สำหรับไดเร็กเอ็กซ์ (DirectX Software Development Kit : DirectX SDK)

ชุดพัฒนาซอฟแวร์สำหรับไดเร็กเอ็กซ์ คือ ชุดของไฟล์เฮดเดอร์ (Header file), ไลบรารี่ (Library), เอกสาร (Document) และตัวอย่างซอสโค๊ด (Source Code) ต่าง ๆ ที่ช่วยในการพัฒนาซอฟแวร์ด้วยการใช้ไดเร็กเอ็กซ์ ในปัจจุบันมีการพัฒนามาถึงเวอร์ชั่น 8 (Version 8.0)แล้ว ซึ่งในเวอร์ชั่นนี้ประกอบด้วยอินเทอร์เฟส (Interface) ต่าง ๆ ที่ช่วยในการทำงานเกี่ยวกับมัลติมีเดีย ดังนี้

6.3.1 DirectX Graphics

เป็นการรวมกันระหว่างไดเร็กดรอว์ (DirectDraw) และ Direct3D ซึ่งเป็นฟังก์ชั่นส่วนประกอบของไดเร็กเอ็กซ์เวอร์ชั่นก่อนหน้านี้ให้มารวมอยู่ในแอพพลิเคชั่นเดียวซึ่งสามารถใช้งานได้กับงานกราฟฟิกส์ทุกชนิดทั้งแบบ 2 มิติและ 3 มิติ อินเทอร์เฟสในกลุ่มนี้ช่วยจัดการการใช้อุปกรณ์แสดงผล 3 มิติที่มีอยู่ในเครื่องคอมพิวเตอร์ (3-D video-display hardware) ในรูปแบบที่ไม่ขึ้นกับชนิดของอุปกรณ์ที่มีอยู่ (Device-independent) DirectX Graphics มีจุดเด่นอยู่ตรงการประมวลผลแบบขนานและสามารถจัดการกับกราฟฟิกส์ให้มีความสมจริงกว่าเวอร์ชั่นที่ผ่านมา

6.3.2 DirectX Audio

เป็นการรวมกันระหว่างไดเร็กซาวน์ (DirectSound) และไดเร็กมิวสิค (DirectMusic) ซึ่งเป็นฟังก์ชั่นส่วนประกอบของไดเร็กเอ็กซ์เวอร์ชั่นก่อนหน้านี้ให้มารวมอยู่ในแอพพลิเคชั่นเดียว ซึ่งสามารถทำงานด้านเสียงซึ่งนอกจากการเล่นไฟล์เสียงแล้ว ยังสนับสนุนการใช้งานฮาร์ดแวร์เร่งความเร็ว (Hardware acceleration) ในการสร้างเสียงแบบไดนามิก (Dynamic soundtrack) และเอฟเฟ็กต์เสียง 3 มิติขั้นสูง (Advanced 3-D position effect)

6.3.3 DirectInput

ได้รับการออกแบบมาเพื่อทำงานกับอุปกรณ์อินพุทเพื่อให้เข้าถึงข้อมูลได้ด้วยการติดต่อโดยตรงกับฮาร์ดแวร์ด้วยฮาร์ดแวร์ไดรเวอร์ (Hardware Driver) ซึ่งเร็วกว่าการติดต่อด้วยเมสเสจในวินโดวส์ นอกจากนั้นยังสนับสนุนอุปกรณ์แบบแรงตอบสนอง (Force feedback) อีกด้วย

6.3.4 DirectPlay

ได้รับการออกแบบมาเพื่อทำงานด้านเน็ตเวิร์ค (Network) ช่วยให้สามารถเล่นเกมได้ครั้งละหลาย ๆ คน โดยการเชื่อมต่อผ่านทางโมเด็มหรือระบบเน็ตเวิร์ค

6.3.5 DirectShow

ออกแบบมาสำหรับการทำงานเกี่ยวกับมีเดียสตรีมมิ่ง (Media-streaming) เช่น การเล่นไฟล์วีดิโอคุณภาพสูง (High-quality video)

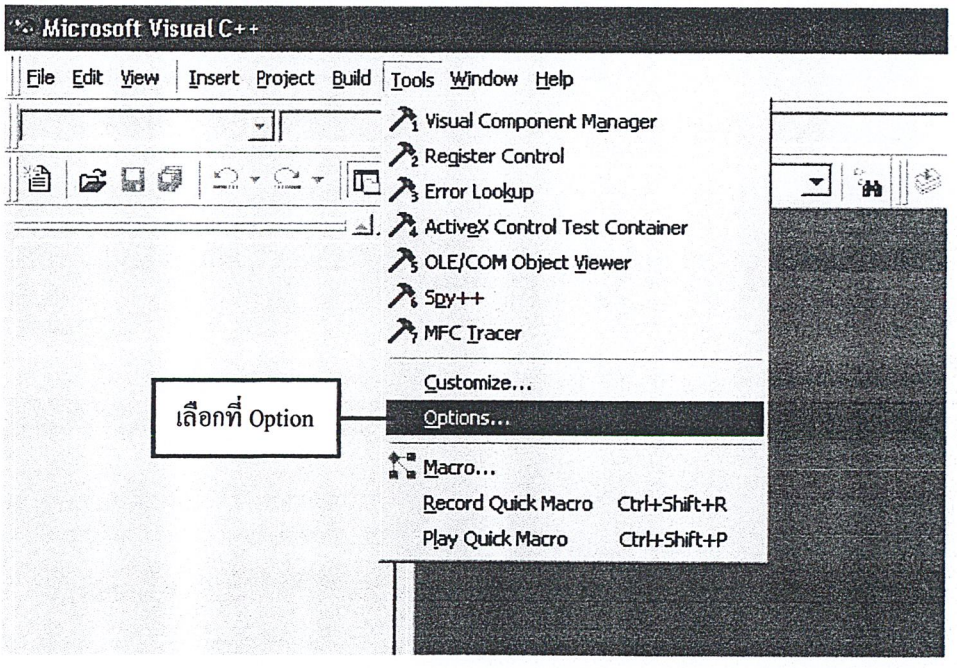
6.3.6 DirectSetup

เป็น API ที่จัดการติดตั้ง (Install) องค์ประกอบที่จำเป็นในการใช้งานไดเร็กเอ็กซ์บนเครื่องคอมพิวเตอร์ด้วยการเรียกใช้เพื่อคำสั่งเดียว เนื่องจากไดเร็กเอ็กซ์เป็นผลิตภัณฑ์ที่มีความซับซ้อนสูงรวมทั้งขั้นตอนการติดตั้งด้วยจึงไม่ควรที่จะติดตั้งไดเร็กเอ็กซ์ด้วยตัวเอง

6.4 การเชื่อมต่อ Direct X 8.0 SDK กับ Visual C++

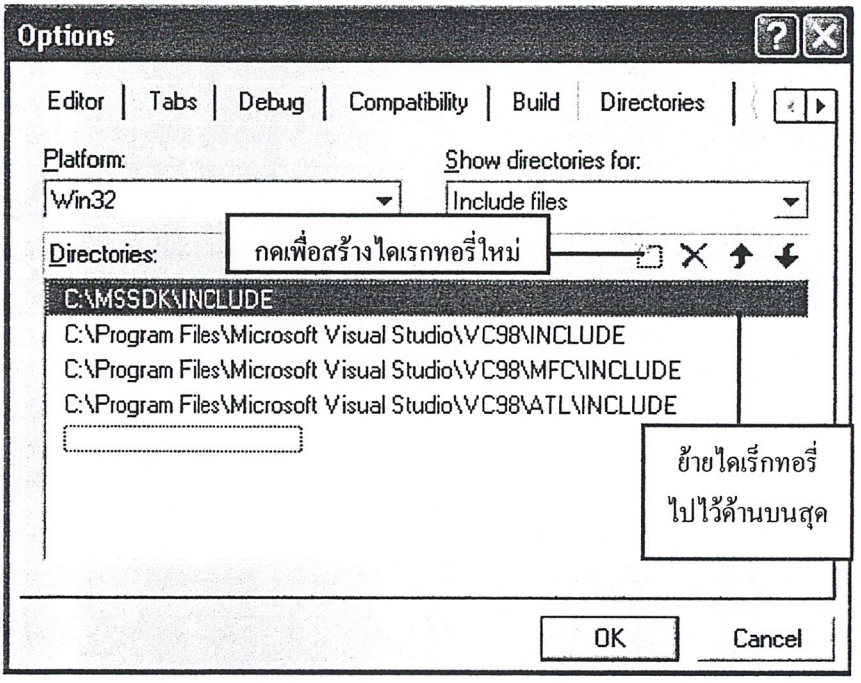
การใช้งาน Direct X SDK ร่วมกับคอมไพเลอร์ Microsoft Visual C++ นั้นจะต้องกำหนดไดเรกทอรี (Directory) ที่เก็บไฟล์เฮดเดอร์ (Header file) และไฟล์ไลบรารี่ (Library) ก่อนเพื่อให้โปรแกรมสามารถทำงานทางด้านกราฟฟิกส์ต่าง ๆ ได้ โดย

6.4.1 เลือก Option จากเมนู Tool



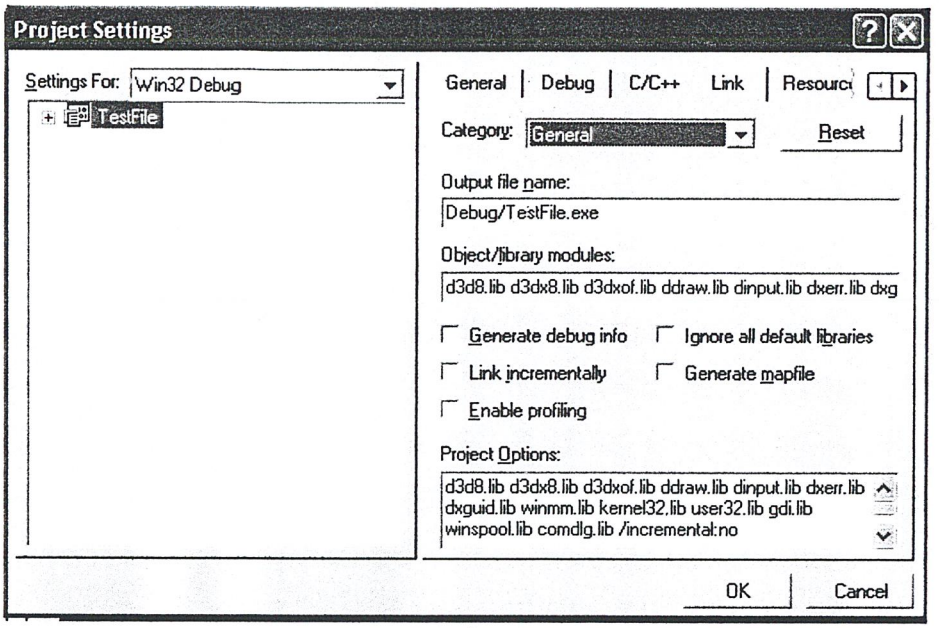
รูปที่ 6.1 แสดงการเลือก Option จากเมนู Tools เพื่อสร้างไดเร็กทอรี่ที่ต้องการ

6.4.2 ปรากฏไดอะลอกบอกซ์ของ Options เลือกแทป Directories แล้วเลือก include file ที่ช่อง "Show directories for" จากนั้นกดปุ่ม new เพื่อเลือกไดเร็กทอรี่ที่เก็บไฟล์อินคลูด (Include file) ของ Direct X SDK ทำการเลือกไดเร็กทอรี่ที่ต้องการ โดยไดเร็กทอรี่ที่ได้จะอยู่บรรทัดล่างสุด จึงต้องทำการเลื่อนไดเร็กทอรี่นั้น ๆ ไปไว้ที่บรรทัดบนสุด ซึ่งในที่นี้คือ C:\MSSDK\IN\CLUDE ดังรูป



รูปที่ 6.2 แสดงการเพิ่มไดเร็กทอรี่ต้องการและการเรียงต้องนำไปไว้ด้านบนสุด

6.4.3 นอกจากนี้ยังต้องกำหนดไลบรารีที่จะให้คอมไพเลอร์ลิงค์ (Link) อีกด้วย โดยทำการเลือก Setting จากเมนู Project แล้วจึงกำหนดค่าไลบรารีที่จะใช้เพิ่มเข้าไปในช่อง "Object/library modules: " ดังรูป



รูปที่ 6.3 แสดงการกำหนดค่าไลบรารี่ที่ต้องการใช้งาน

6.5 Direct3D

6.5.1 ไดเร็กเอ็กซ์กราฟฟิกส์ (DirectX graphics)

เนื่องจากไดเรกเอ็กซ์กราฟฟิกส์ เป็นการรวมกันระหว่างไดเร็กดรอว์ (DirectDraw) และ Direct3D จึงจะขอกล่าวถึงความหมายของอินเทอร์เฟสทั้งสองดังต่อไปนี้

1) DirectDraw คือส่วนประกอบที่สำคัญที่สุดของไดเร็กเอ็กซ์ ซึ่งช่วยให้ผู้พัฒนาเกมคอมพิวเตอร์สามารถติดต่อโดยตรงกับหน่วยความจำของการ์ดแสดงผล ทำให้เพิ่มประสิทธิภาพในการวาดภาพ 2 มิติได้เร็วยิ่งขึ้น โดยวิธีการทำงานของ DirectDraw แบ่งออกเป็น 2 วิธีหลัก ๆ คือ

1.1) บลิทติ้ง (Blitting) คือการปะภาพที่มีอยู่แล้วลงในพิกัดที่กำหนด และอาศัย คัลเลอร์คีย์ (Color Key) ในการกำหนดสีที่ไม่ต้องการแสดงผล

1.2) เพจ ฟลิปปิ้ง (Page Flipping) คือวิธีการทำให้เกิดภาพเคลื่อนไหวโดยการสลับกันไปมาระหว่างฟรอนท์ บัฟเฟอร์ (Front buffer)กับแบค บัฟเฟอร์ (Back buffer)

2) Direct3D จะสนับสนุนงานทางด้าน 3 มิติ ซึ่งประกอบด้วยการทำงานหลัก 2 แบบ คือแบบรีเทน (Retain) ซึ่งเป็น API ระดับสูง และแบบอิมมีดิเอต (Immediate) ซึ่งเป็น API ระดับต่ำ โดย Direct3D จะทำการวาดภาพวัตถุ 3 มิติลงบนแบค บัฟเฟอร์ ของ DirectDraw เพื่อจะทำการเพจ ฟลิปปิ้ง แสดงผลทางหน้าจอต่อไป

Direct3D คือ ซอฟแวร์อินเทอร์เฟส (Software interface) ซึ่งจัดการการเข้าถึงอุปกรณ์แสดงผล (Display device) โดยยังรักษาความเข้ากันได้กับกราฟฟิกส์ดีไวซ์อินเทอร์เฟส (Graphics device interface : GDI) และจัดการให้มีการใช้งานในแบบที่ไม่ขึ้นกับชนิดของอุปกรณ์ (Device independent) และมีความสามารถในการใช้คุณสมบัติพิเศษที่อยู่ในฮาร์ดแวร์ได้

6.5.2 ความก้าวหน้าทางด้านโครงสร้างและคุณสมบัติของ Direct3D

- สนับสนุนการใช้งานบัฟเฟอร์ความลึก (z-buffer หรือ w-buffer)
- สามารถเรนเดอร์ (Render) ภาพได้ทั้งในแบบแฟลต เชดดิ้ง (Flat shading) และแบบกอรอด (Gouraud shading)
- สามารถใช้แหล่งแสงได้หลายแหล่งและหลายชนิด
- สนับสนุนการใช้แมตทีเรียล (Material) และเท็กซ์เจอร์ (Texture) แบบเต็มรูปแบบ รวมไปถึงการทำมิปแมป (Mipmap)
- มีซอฟแวร์อีมูเลชั่นไดรเวอร์ (Software emulation driver) ที่มีประสิทธิภาพสูง
- ไม่ขึ้นอยู่กับชนิดของฮาร์ดแวร์ (Hardware independent)
- สนับสนุนการตัด (Clipping) ในโปรแกรมที่ทำงานในวินโดวส์และแบบเต็มจอภาพ
- สามารถใช้งาน Image-stretching hardware ได้
- สามารถใช้งานเมมโมรี (Memory) ของอุปกรณ์แสดงผลทั้งส่วนที่เป็นมาตรฐานและส่วนที่ขยายเพิ่มเติมได้
- คุณสมบัติอื่น ๆ เช่น การใช้งานฮาร์ดแวร์แต่เพียงผู้เดียว (Exclusive hardware access) และการเปลี่ยนความละเอียดในการแสดงผล (Resolution switching)

ด้วยโครงสร้างและคุณสมบัติเหล่านี้การสร้างโปรแกรมด้วย Direct3D จึงมีประสิทธิภาพสูงกว่าการใช้ GDI ของวินโดวส์และการเขียนโปรแกรมบนระบบปฏิบัติการดอส (DOS)

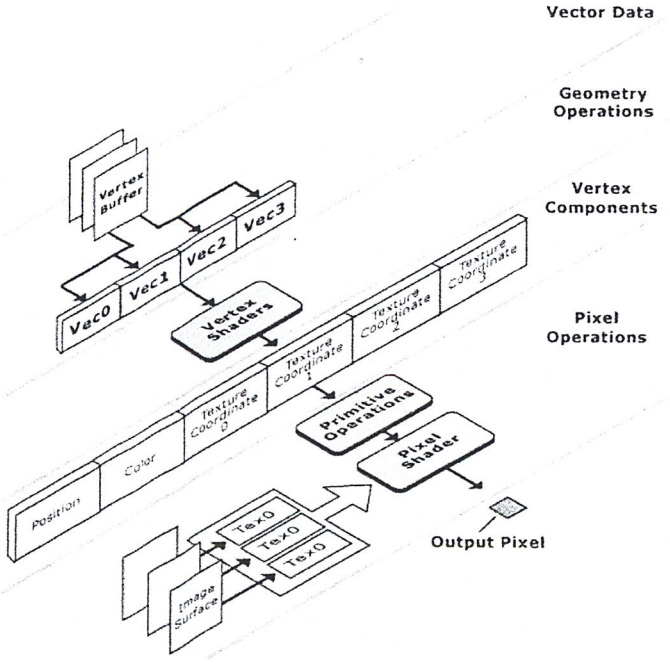
การจัดการสิ่งต่าง ๆ ใน Direct3D มีพื้นฐานอยู่บนทฤษฎีของเวอร์เท็กซ์ (Vertex), โพลีกอน (Polygon) และคำสั่งต่าง ๆ ที่ควบคุมข้อมูลเหล่านี้ ซึ่งสามารถเข้าถึงไปป์ไลน์ (Pipeline) หรือกระบวนการในการเรนเดอร์ภาพ 3 มิติได้ทันทีไม่ว่าจะเป็นการเปลี่ยนแปลงตำแหน่ง (Transformation), การให้แสง (Lighting) และการลงจุดสี (Rasterization) ซึ่งถ้าฮาร์แวร์ที่มีอยู่ในเครื่องไม่สนับสนุนการเร่งความเร็วในส่วนใด ๆ ของไปป์ไลน์ก็ตาม Direct3D จะใช้

ซอฟแวร์ที่มีประสิทธิภาพสูงทำงานในส่วนนั้นแทน แต่อย่างไรก็ตามซอฟแวร์ก็ยังมีประสิทธิภาพต่ำกว่าฮาร์ดแวร์อยู่ดี

Direct3D ได้จัดเตรียมวิธีการที่ง่ายและไม่ซับซ้อนในการเซ็ทอัพ (Set up)และเรนเดอร์ภาพ 3 มิติ กุญแจสำคัญในวิธีการเรนเดอร์คือวิธีดรอว์ไพรมิทีฟ (DrawPrimitive) ซึ่งมันสามารถทำการเรนเดอร์วัตถุ (Object) 1 อย่างหรือมากกว่านั้นในซีน (Scene) เดียวกัน

6.6 สถาปัตยกรรมของ Direct3D (Direct3D Architecture)

ใน Direct3D มีอยู่ 2 ส่วนที่สามารถโปรแกรมได้ก็คือ เวอร์เท็กซ์เชดเดอร์ (Vertex shader) และพิกเซลเชดเดอร์ (Pixel shader) ซึ่งเวอร์เท็กซ์เชดเดอร์เป็นตัวที่จัดการการสร้างและกระบวนการต่าง ๆ บนเวอร์เท็กซ์ ส่วนพิกเซลเชดเดอร์เป็นกระบวนการหลังจากมีการประมวลผลข้อมูลออกมาเป็นตำแหน่งของพิกเซลและค่าสีของพิกเซลนั้น ๆ แล้วดังแผนภาพ



รูปที่ 6.4 แสดงสถาปัตยกรรมของ Direct3D

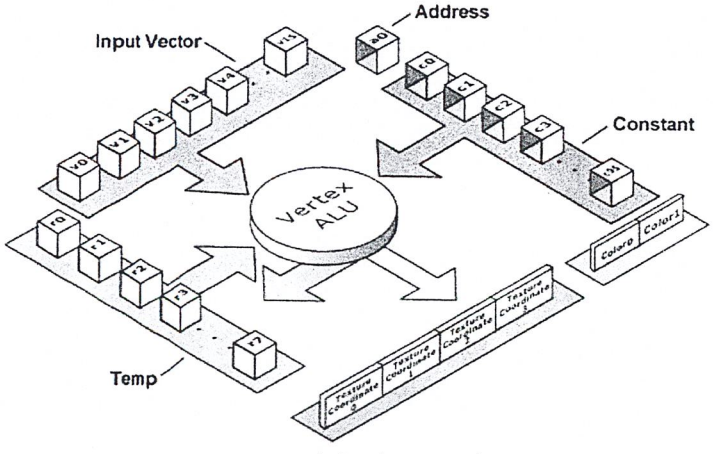
จะเห็นว่าการทำงานของเวอร์เท็กซ์เชดเดอร์จะรับข้อมูลมาจากบัฟเฟอร์ (Buffer) ซึ่งโดยปกติแล้วจะเป็นเวอร์เท็กซ์บัฟเฟอร์ (Vertex buffer) ที่เก็บข้อมูลเวอร์เท็กซ์ และนำมาผ่านกระบวนการต่าง ๆ ซึ่งสามารถกำหนดกระบวนการที่จะทำได้โดยการใช้ Direct3D vertex shader assembler

หลังจากที่ข้อมูลผ่านการประมวลผลในเวอร์เท็กซ์เชดเดอร์แล้วจะต้องเรียกฟังก์ชันในการวาดข้อมูลเหล่านี้ เช่น DrawPrimitive เป็นต้น ซึ่งเมื่อเรียกฟังก์ชันเหล่านี้แล้วจุดแต่ละจุดที่ได้จากฟังก์ชันก็จะส่งเข้าสู่พิกเซลเชดเดอร์ต่อไป โดยข้อมูลที่ส่งให้พิกเซลเชดเดอร์ประกอบด้วยตำแหน่งบนจอภาพ, สี, เท็กซ์เจอร์และคู่อันดับแสดงตำแหน่งของเท็กซ์เจอร์ (Texture coordinate) พิกเซลเชดเดอร์จะประมวลผลตามคำสั่งที่กำหนดโดยใช้ Direct3D pixel assembly

นอกจากข้อมูลที่ได้รับจากฟังก์ชันในการวาดข้อมูล เช่น DrawPrimitive แล้วพิกเซลเชดเดอร์ยังต้องการข้อมูลของเท็กซ์เจอร์ในรูปของเท็กซ์เจอร์เซอร์เฟส (Texture surface) อีกด้วย ซึ่งจะใช้ร่วมกับข้อมูลคู่อันดับแสดงตำแหน่งของเท็กซ์เจอร์ในการกำหนดสีที่จะปรากฏ แล้วจึงส่งค่าพิกเซลไปยังเฟรมบัฟเฟอร์ (Frame buffer) เพื่อแสดงผลออกจอภาพต่อไป

6.6.1 สถาปัตยกรรมของเวอร์เท็กซ์เชดเดอร์ (Vertex Shader Architecture)

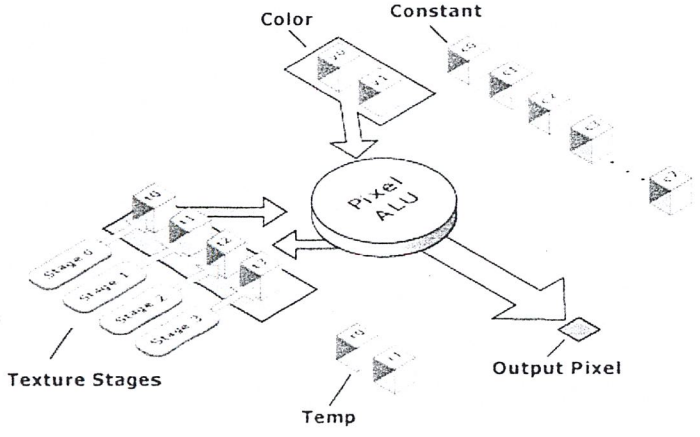
เวอร์เท็กซ์เชดเดอร์รับข้อมูลมาจากเวอร์เท็กซ์ดาต้าสตรีม (Vertex data stream) และมีข้อมูลค่าคงที่ที่มีความจำเป็นในการคำนวณ ซึ่งสามารถอ้านเข้ามาได้โดยใช้เวอร์เท็กซ์เชดเดอร์ดีแคลเรเตอร์ (Vertex shader declarator) ดังแผนภาพ



รูปที่ 6.5 แสดงสถาปัตยกรรมของเวอร์เท็กซ์เชดเดอร์ (Vertex Shader Architecture)

6.6.2 สถาปัตยกรรมของพิกเซลเชดเดอร์ (Pixel Shader Architecture)

ข้อมูลที่เข้ามาในพิกเซลเชดเดอร์ต้องเป็นข้อมูลแบบคลิปสเปซเวอร์เท็กซ์ (Clip space vertex) ในโอโมจีเนียสโคออดิเนต Homogenous coordinate ค่าของสีทั้งในแบบดิฟฟิวส์ (Diffuse) และสเปคคูลาร์ (Specular) มาจากค่าในรีจิสเตอร์สี (Color register) v0 และ v1 ค่าคู่อันดับแสดงตำแหน่งของเท็กซ์เจอร์ (Texture coordinate) และเท็กซ์เจอร์ที่ใช้มาจากค่าในรีจิสเตอร์เท็กซ์เจอร์ (Texture register) t0, t1, t2 และ t3 จากข้อมูลในขั้นตอนการกำหนดเท็กซ์เจอร์ (Texture setup stage) และข้อมูลในเวอร์เท็กซ์ ดังแผนภาพ



รูปที่ 6.6 แสดงสถาปัตยกรรมของพิกเซลเชดเดอร์ (Pixel Shader Architecture)

6.7 เรนเดอร์ไปป์ไลน์ใน Direct3D (Direct3D Rendering Pipeline)

เรนเดอร์ไปป์ไลน์ (Rendering pipeline) คือชุดของขั้นตอนในการประมวลซึ่งกำหนดการกระทำของไปป์ไลน์ในการเปลี่ยนแปลงตำแหน่งและให้แสงของเวอร์เท็กซ์ (Vertex transform and lighting pipeline) และไปป์ไลน์ในการเฉลี่ยค่าของพิกเซลและเท็กซ์เจอร์ (Pixel and texture blending pipline)

ใน Direct3D มีเรนเดอร์ไปป์ไลน์ 2 แบบ คือ การประมวลผลฟังก์ชันแบบตายตัว (Fixed function vertex and pixel processing) และการประมวลผลเวอร์เท็กซ์และพิกเซลแบบโปรแกรมได้ (Programmable vertex and pixel processing) ซึ่งมีรายละเอียดดังต่อไปนี้

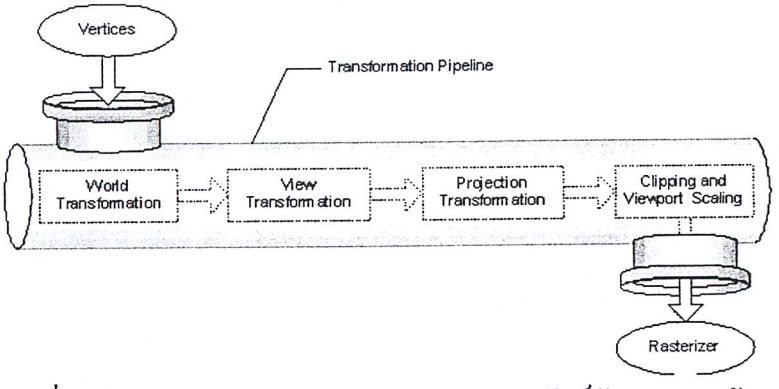
6.7.1 การประมวลผลฟังก์ชันแบบตายตัว (Fixed Function Vertex and Pixel Processing)

ใน Direct3D การประมวลผลฟังก์ชันแบบตายตัวจะใช้ไปป์ไลน์ที่เรียกว่า ทรานส์ฟอร์มเมชัน ไปป์ไลน์ (Transformation pipeline) หรือจีโอเมทรี ไปป์ไลน์(Geometry pipeline) ซึ่งมีส่วนประกอบต่าง ๆดังนี้

6.7.1.1 ทรานส์ฟอร์มเมชัน เอนจิ้น (Transformation engine)

ทำหน้าที่ในการกำหนดตำแหน่งของโมเดลและผู้ดู (Viewer) ในโลกของ Direct3D, ทำการฉายเวอร์เท็กซ์ (Project vertex) สำหรับการแสดงผลบนจอภาพ และตัด(Clip) เวอร์เท็กซ์ส่วนที่มองไม่เห็นทิ้ง นอกจากนั้นยังทำงานในส่วนของแสงอีกด้วย โดยการคำนวณค่าของดิฟฟิวส์และสเปคคูลาร์ในแต่ละเวอร์เท็กซ์

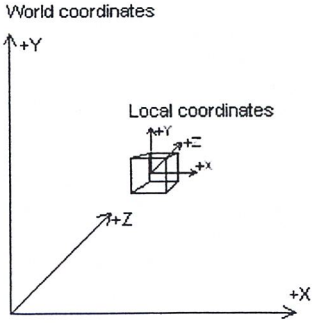
ทรานส์ฟอร์มเมชัน ไปป์ไลน์ (Transformation pipeline) เมื่อรับข้อมูลเวอร์เท็กซ์เข้ามาแล้วจะประมวลผลโดยใช้ เวิลด์, วิวและโปรเจ็กต์ชัน ทรานส์ฟอร์มเมชัน ซึ่งอยู่ในรูปของเมตริกซ์ หลังจากนั้นข้อมูลเวอร์เท็กซ์จะถูกตัดให้เหลือแต่ส่วนที่มองเห็น แล้งจึงส่งไปยังส่วนของการลงจุดสี (Rasterization) เพื่อทำการเรนเดอร์ต่อไป ดังแผนภาพ



รูปที่ 6.7 แสดงกระบวนการของการประมวลผลฟังก์ชันแบบตายตัว

6.7.1.2 เวิลด์ ทรานส์ฟอร์มเมชัน (World transformation)

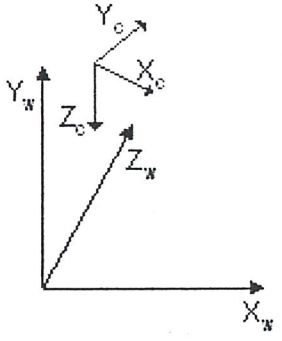
คือการเปลี่ยนแปลงตำแหน่งอ้างอิงของเวอร์เท็กซ์บนวัตถุจากเดิมที่มีความสัมพันธ์หรืออ้างอิงกับแกนของตัวเอง (Model space) เท่านั้น มาอ้างอิงกับตำแหน่งใหม่ซึ่งใช้อ้างอิงกับวัตถุทั้งหมดที่ปรากฏอยู่ในฉาก (World space) ซึ่งโครงพิกัดตำแหน่งที่ใช้จะเรียกว่า เวิลด์ โคออดิเนต



รูปที่ 6.8 แสดงภาพความสัมพันธ์ระหว่าง Model coordinate และ World coordinate

6.7.1.3 วิว ทรานส์ฟอร์มเมชัน (View transformation)

เป็นการกำหนดตำแหน่งของผู้ดู (Viewer) หรือกล้อง (Camera) ในเวิลด์สเปซ (World space)



รูปที่ 6.9 แสดงโครงพิกัดตำแหน่งของกล้องกับ World coordinate

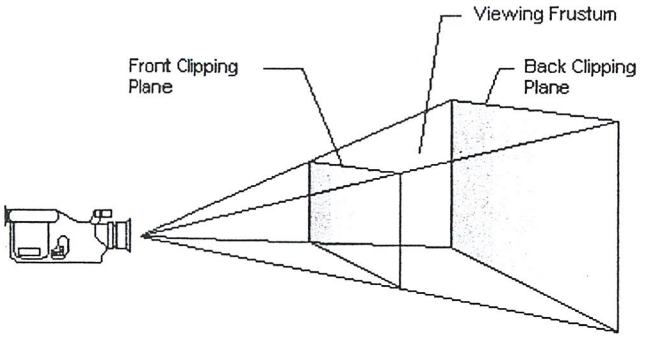
6.7.1.4 โปรเจ็กต์ชัน ทรานส์ฟอร์มเมชัน (Projection transformation)

เป็นการกำหนดค่าภายในของกล้องในทำนองเดียวกับการเลือกเลนส์ที่จะใช้ในกล้องถ่ายภาพ ซึ่งเป็นส่วนที่มีความซับซ้อนที่สุดในการทรานส์ฟอร์มเมชันทั้งสามแบบ ประกอบด้วยหัวข้อต่อไปนี้

ก. วิว ฟรัสทรัม (View Frustrum)

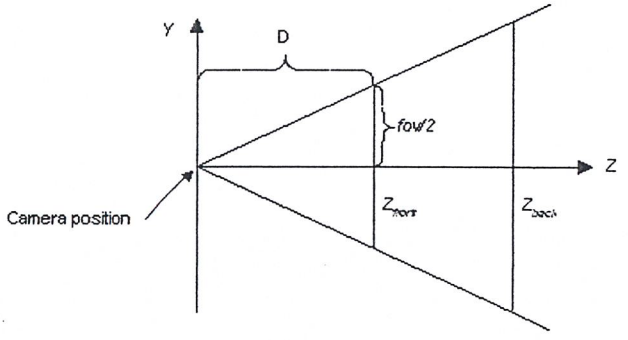
วิว ฟรัสทรัม คือบริเวณ 3 มิติที่จะปรากฏ โดยมีความสัมพันธ์กับตำแหน่งและมุมมองของกล้อง (Viewpot's camera) รูปร่างของบริเวณ 3 มิติที่เกิดขึ้นจะเป็นตัวกำหนดว่าโมเดลจะโปรเจ็กต์ (Project) บนจอภาพในลักษณะใด ซึ่งลักษณะการโปรเจ็กต์ที่ใช้กันทั่วไปคือแบบเพอร์สเปกทีฟ (Perspective) โดยการโปรเจ็กต์ลักษณะนี้จะทำให้วัตถุมีความลึกสมจริง และวัตถุที่อยู่ใกล้จะดูมีขนาดใหญ่กว่าวัตถุที่อยู่ห่างออกไป

วิว ฟรัสทรัม มีรูปร่างเหมือนปิรามิดซึ่งมีตำแหน่งของกล้องเป็นจุดยอด บริเวณ 3 มิติจะเกิดขึ้นจากการตัดปิรามิดนี้ด้วยระนาบตัดด้านหน้า (Front clipping plane) และระนาบตัดด้านหลัง (Back clipping plane) วัตถุจะถูกมองเห็นก็ต่อเมื่อมันอยู่ในบริเวณนี้เท่านั้น ดังรูป



รูปที่ 6.10 แสดงการ โปรเจ็กต์ (Project) ภาพแบบเพอร์สเปกทีฟ (Perspective)

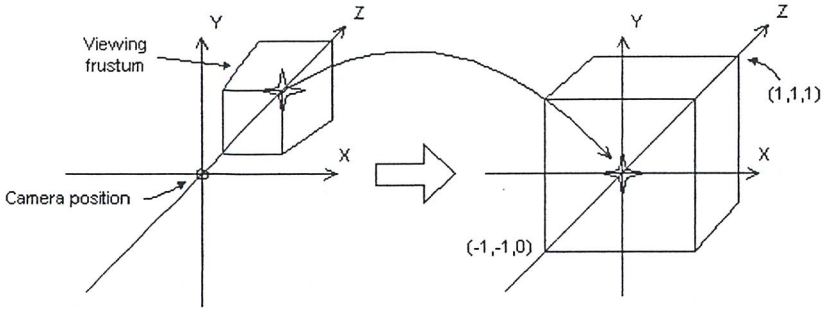
พารามิเตอร์สำคัญที่เป็นตัวกำหนด วิว ฟรัสทรัม ก็คือ fov (Field of view) และค่าระยะทางของระนาบที่มาตัดทั้งด้านหน้าและด้านหลัง ซึ่งกำหนดอยู่ในแกน z ดังรูป



รูปที่ 6.11 แสดง fov และระยะ D ซึ่งเป็นพารามิเตอร์ที่สำคัญของตัววิว ฟรัสทรัม (View frustrum)

ข. โปรเจ็กต์ชันเมตริกซ์ (Projection matrix)

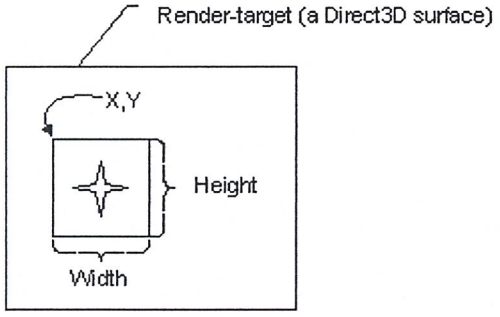
โปรเจ็กต์ชั่นเมตริกซ์ใช้ในการปรับขนาดของวิว ฟรัสทรัม เนื่องจากบริเวณที่วัตถุสามารถมองเห็นได้จะต้องถูกปรับให้เป็นรูปสี่เหลี่ยมลูกบาศก์ จากรูปจะต้องมีการเลื่อนจากมุมบนขวาของฉาก (Scene) มายังเซ็นเตอร์ของระบบพิกัด จึงจะสามารถประมวลผลต่อไปได้ ดังรูป



รูปที่ 6.12 แสดงการใช้โปรเจ็กต์ชัน ทรานส์ฟอร์มเมชัน (Projection transformation) เปลี่ยนวิว ฟรัสทรัม ไปอยู่ในระบบพิกัดตำแหน่งใหม่

ค. วิวพอร์ท (Viewport) และ คลิปปลิ้ง (Clipping)

วิวพอร์ท คือพื้นที่สี่เหลี่ยมที่ถูกโปรเจกต์มาจากฉาก 3 มิติ ใน Direct3D พื้นที่สี่เหลี่ยมดังกล่าวจะอยู่ในโคออดิเนตของพื้นผิว Direct3D ซึ่งระบบจะใช้เป็นบริเวณที่เรนเดอร์ภาพ (Render-target) ดังรูป



รูปที่ 6.13 แสดงพื้นที่สี่เหลี่ยมของวิวพอร์ท (Viewport) หรือพื้นที่ของเรนเดอร์-ทาร์เก็ท (Render-target)

ง. การลงจุดสี (Resterization)

หลังจากผ่านทรานส์ฟอร์มเมชัน ไปป์ไลน์แล้ว ค่าเวอร์เท็กซ์จะถูกแปลง, ตัด และย่อขยายจนมีขนาดพอดีกับวิวพอร์ทบนเซอร์เฟสที่ต้องการเรนเดอร์ (Render target) คือพร้อมที่จะส่งไปยังส่วนของการลงจุดสี (Rasterization) เพื่อแสดงผลบนจอภาพ แต่เนื่องจากในขณะนี้ข้อมูลเวอร์เท็กซ์ยังคงเป็นแบบโฮโมจีเนียส (Homogenous) อยู่ และตัวลงจุดสียังต้องการข้อมูลของ RHW (Reciprocal-of-homogenous-w) เพิ่มเติมอีกด้วย ซึ่ง Direct3D จะทำการเปลี่ยนแปลง

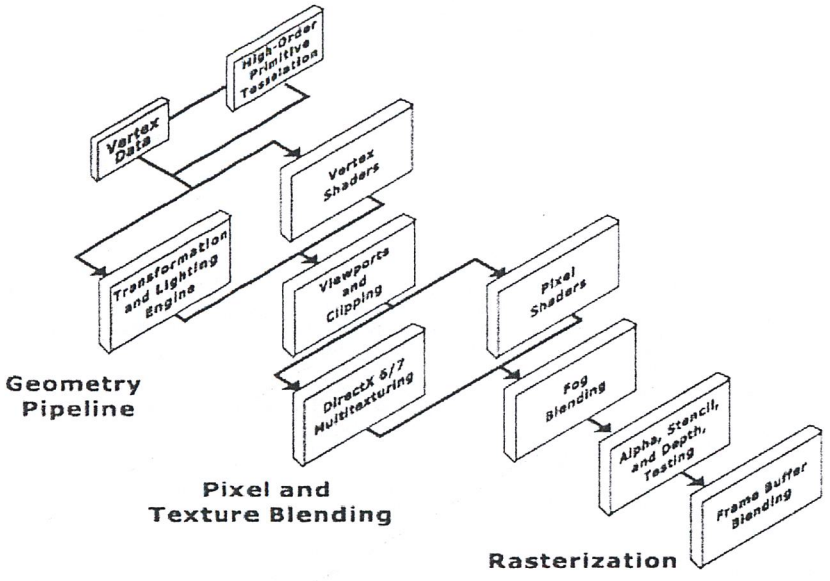
ค่าตำแหน่งในแกน x, แกน y และแกน z ไปเป็นข้อมูลที่ไม่โฮโมจีเนียสโดยการหารด้วยค่า w และหาค่า RHW จากส่วนกลับของ w ตามสมการ

$$X_s = x/w$$
$$Y_s = y/w$$
$$Z_s = z/w$$
$$RHW = 1/w$$

เมื่อได้ค่าที่ต้องการแล้วจึงส่งเข้าสู่ตัวลงจุดสีซึ่งจะใช้ค่าในแกน x และแกน y ในการกำหนดว่าเป็นจุดใดบนจอภาพ และใช้ข้อมูลในแกน z ในการวัดความลึกเพื่อเลือกว่าจุดสีจุดใดที่จะได้แสดงผลโดยใช้ แซท-บัฟเฟอร์ (Z-Buffer) ส่วนค่า RHW ใช้ในการคำนวณหมอก (Fog), การปะเท็กซ์เจอร์ (Texture mapping) และใช้ในการวัดความลึกเพื่อเลือกว่าจุดสีใดที่จะได้แสดงผล เราจะใช้ ดับบลิว-บัฟเฟอร์ (W-buffer) เป็นตัวจัดการ

6.7.2 การประมวลผลเวอร์เท็กซ์และพิกเซลแบบโปรแกรมได้ (Programmable Vertex and Processing)

นอกจากไปป์ไลน์ในการประมวลผลเดิมที่มีอยู่ในเวอร์ชันเก่าๆ แล้วในDirect3D เวอร์ชัน 8.0 ยังได้เพิ่มไปป์ไลน์ที่สามารถโปรแกรมได้อีกด้วย โดยส่วนแรกที่เพิ่มเข้ามาจากกระบวนการการประมวลผลฟังก์ชันแบบตายตัว คือ High order primitive module ซึ่งทำหน้าที่ในการเทสเซลเลท (Tessellate)โดยแบ่ง High order primitive ออกเป็นพื้นที่สี่เหลี่ยมส่วนย่อย ๆ เพื่อที่จะสามารถเข้าสู่ขั้นตอนต่อไปของกระบวนการได้ ส่วนเวอร์เท็กซ์เชดเดอร์เป็นโมดูลที่สามารถโปรแกรมได้ โดยจะใช้ในการประมวลผลทางจีโอเมตริกซ์ (Geometric) บนข้อมูลชนิดเวอร์เท็กซ์ เวอร์เท็กซ์เชดเดอร์สามารถรองรับฟังก์ชันในการทำงานได้หลากหลายรวมไปถึงการทรานส์ฟอร์มเมชันและการให้แสงแบบมาตรฐานได้ ส่วนพิกเซลเชดเดอร์ก็เป็นโมดูลที่สามารถโปรแกรมเช่นเดียวกัน มีหน้าที่ในการควบคุมสีและการผสมสีในส่วนของความโปร่งใส (Alpha channel) และในกระบวนการที่เกี่ยวข้องกับคู่อันดับกำหนดตำแหน่งของเท็กซ์เจอร์ (Texture coordinate) กระบวนการทั้งหมดสามารถแสดงได้ด้วยแผนภาพดังนี้



รูปที่ 6.14 แสดงส่วนประกอบหลักและการไหลของข้อมูล (Data flow) ในการประมวลผลเวอร์เท็กซ์และพิกเซลแบบโปรแกรมได้

6.8 Direct3D Object

Direct3D ถูกสร้างขึ้นโดยใช้หลักการของ COM Object และอินเตอร์เฟส (Interface) ดังนั้นโปรแกรมที่เขียนขึ้นในภาษา C และ C++ จึงสามารถใช้งานอินเทอร์เฟสและอ็อบเจ็กต์ (Object) ได้โดยตรง ต่างจากการใช้ภาษาเบสิก (Basic) หรือภาษาอื่น ๆ ที่ต้องทำงานผ่านชั้นของโค๊ดที่ทำหน้าที่ในการเปลี่ยนแปลงข้อมูลในการติดต่อกับไดเร็กเอ็กซ์รันไทม์ (DirectX runtime) อีกทอดหนึ่ง

Direct3D Object เป็นอ็อบเจ็กต์แรกที่ต้องสร้างขึ้นถ้าต้องการจะใช้งาน Direct3D และเป็นอ็อบเจ็กต์สุดท้ายที่จะต้องปลดปล่อย (Release) หลังจากการใช้งาน เราสามารถตรวจสอบความสามารถของฮาร์ดแวร์และอุปกรณ์อื่น ๆ ที่มีอยู่ได้โดยการสั่งงานผ่านทาง Direct3D Object ซึ่งทำให้เราสามารถตรวจสอบอุปกรณ์ได้โดยไม่ต้องสร้างมันขึ้นมา

6.9 การใช้งาน Direct3D

ขั้นตอนแรกคือ การรับค่าพอยเตอร์ที่ชี้ไปยัง Direct3D interface เพื่อใช้ในการเข้าถึงการทำงานต่างๆ ของ Direct3D โดยสามารถทำได้ดังนี้

```
LPDIRECT3D8 g_pD3D = NULL;
```

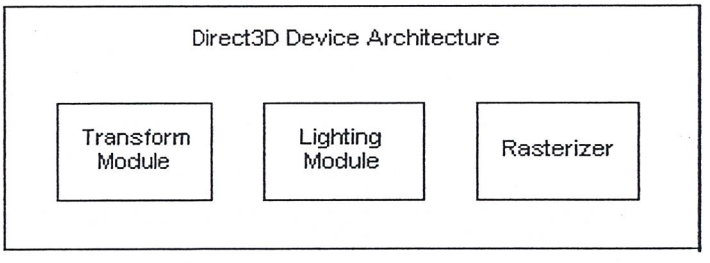
```
if( NULL == (g_pD3D = Direct3DCreate8(D3D_SDK_VERSION)))
    return E_FAIL;
```

เมื่อสร้างดีไวซ์ขึ้นมาแล้วสามารถใช้คำสั่งในการหาพอยเตอร์ที่ชี้ไปยัง Direct3D interface ที่ใช้สร้างดีไวซ์ได้โดยใช้เมทธอด (method) Idirect3Ddevice8::GetDirect3D

6.10 Direct3D Device

ดีไวซ์ใน Direct3D คือ องค์ประกอบที่ใช้ในการเรนเดอร์ (render) ซึ่งจะเก็บรักษาสถานะในการเรนเดอร์ (render state) นอกจานั้นยังทำหน้าที่ในกระบวนการเปลี่ยนแปลงตำแหน่งและให้แสง (transformation and lighting operation) และการลงจุด (rasterize) ของภาพบนเซอร์เฟส (surface) อีกด้วย

โครงสร้างของดีไวซ์ตามสถาปัตยกรรม ดังแผนภาพ



รูปที่ 6.15 แสดงสถาปัตยกรรมของดีไวซ์

6.10.1 ชนิดของดีไวซ์ แบ่งได้ ดังต่อไปนี้

ก. HAL Device

ย่อมาจาก Hardware Abstraction Layer คือ ฮาร์ดแวร์ซึ่งมีความสามารถในการเร่งความเร็วในการลงจุดสี (Rasterization) นอกจากนี้ยังมีความสามารถในการเร่งความเร็วในการประมวลผลเวอร์เท็กซ์ (vertex processing) อีกด้วย นั่นคือ HAL device มีการสร้างโมดูลในบางส่วนหรือทั้งหมดของเรนเดอร์ไปป์ไลน์ไว้ในตัวฮาร์ดแวร์ จึงมีประสิทธิภาพสูง แต่ HAL device ไม่สามารถเรนเดอร์บนเซอร์เฟสแบบ 8 บิตได้

การสร้างดีไวซ์แบบ HAL สามารถทำได้โดยการเรียกเมทธอด Idirect3D::CreateDevice และกำหนดค่าของD3DDEVTYPE_HAL เป็นชนิดของดีไวซ์

ข. Reference Device

สนับสนุนการทำงานของ Direct3D ในทุกรูปแบบ เนื่องจากมันได้รับการออกแบบให้ทำงานอย่างถูกต้องมากกว่าต้องการความเร็วในการทำงาน ผลลัพธ์ที่ได้จึงทำให้การทำงานช้ามาก จึงเหมาะสำหรับใช้ตรวจสอบความถูกต้องในการทำงานของโปรแกรมเท่านั้น

ค. Pluggable Software Device

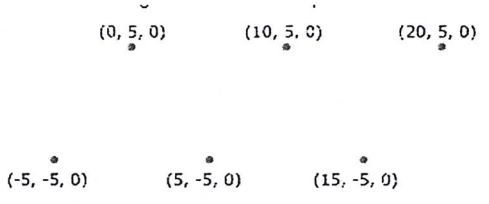
ในบางกรณีที่อุปกรณ์คอมพิวเตอร์ไม่สามารถในการเร่งความเร็วในการประมวลผลบางอย่างได้ Direct3D จะจำลอง 3- D ฮาร์ดแวร์ขึ้นมาโดยใช้ซอฟแวร์ แต่จะทำให้การทำงานช้าลงไป แม้ว่าสามารถที่จะใช้คำสั่งพิเศษที่มีอยู่ในตัวซีพียูได้ก็ตาม

6.10.2 ชนิดของไพรมิทีฟที่สามารถทำงานได้ (Device-Support Primitive Type)

ก. Point List

คือ กลุ่มของเวอร์เท็กซ์ที่จะได้รับการเรนเดอร์เป็นจุดเดี่ยวๆ สามารถนำไปประยุกต์ใช้ในการทำกลุ่มของดวงดาว และเส้นประบนพื้นผิวของโพลีกอนได้

ตัวอย่าง จุดที่ได้รับการเรนเดอร์



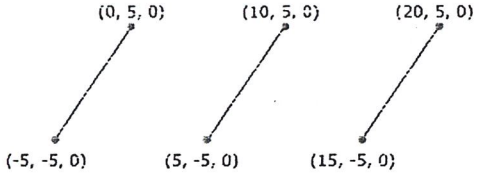
รูปที่ 6.16 แสดง Point LIst

จุดสามารถมีแมตทีเรียล (material) และเท็กซ์เจอร์ (texture) ได้เช่นกัน โดยสีของแมตทีเรียลหรือเท็กซ์เจอร์จะปรากฏเป็นสีของจุดนั้นๆ

ข. Line List

คือ กลุ่มของเวอร์เท็กซ์ที่ได้รับการเรนเดอร์เป็นเส้นตรงเดี่ยวๆ ซึ่งสามารถนำไปประยุกต์ทำเป็นฝนที่กำลังตกได้

ตัวอย่าง เส้นตรงที่ได้รับการเรนเดอร์



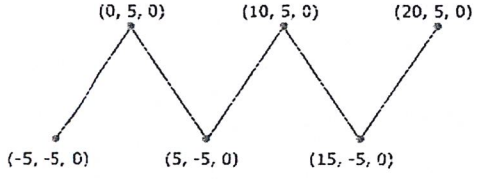
รูปที่ 6.17 แสดงลักษณะ Line List

สามารถกำหนดค่าแมตทีเรียล และเท็กซ์เจอร์ให้ตรงเส้นตรงได้เช่นกัน โดยสีจะปรากฏไปตามความยาวของเส้นตรง

ค. Line Strips

คือ เส้นตรงที่มีการเชื่อมต่อปลายเข้าด้วยกัน

ตัวอย่าง การเรนเดอร์ดังรูป

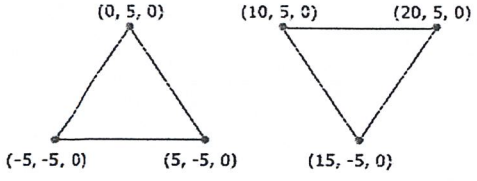


รูปที่ 6.18 แสดงลักษณะของ Line Strips

ง.Triangle List

คือ จุดสามจุดที่รวมกันเพื่อสร้างรูปสามเหลี่ยม 1 รูป โดยไม่เกี่ยวข้องกับรูปสามเหลี่ยมอื่นๆ เนื่องจากไพรมิทีฟทุกตัวจะต้องเป็นรูปสามเหลี่ยม ดังนั้น จำนวนจุดที่ส่งไปเรนเดอร์ไปป์ไลน์จะต้องหารด้วย 3 ลงตัว

ตัวอย่าง สามเหลี่ยมที่ได้จากการเรนเดอร์ ดังรูป

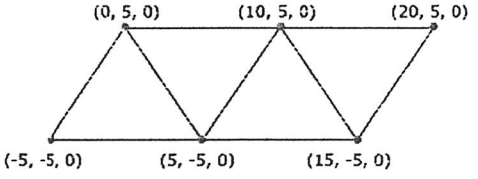


รูปที่ 6.19 แสดงลักษณะของ Triangle List

จ.Triangle Strips

คือ กลุ่มสามเหลี่ยมที่เชื่อมต่อกันไป เนื่องจากมีการใช้บางจุดร่วมกันจึงทำให้สามารถประหยัดจำนวนข้อมูลได้ จึงทำให้ประสิทธิภาพในการประมวลผลดีขึ้น แต่มีโมเดลจำนวนน้อยที่สามารถทำให้อยู่ในรูปของข้อมูลลักษณะนี้ได้

ตัวอย่าง สามเหลี่ยมที่ได้จากการเรนเดอร์ ดังรูป

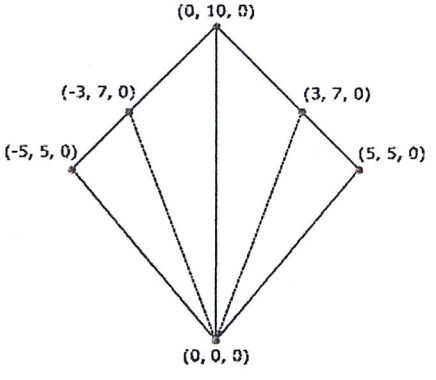


รูปที่ 6.20 แสดงลักษณะของ Triangle List

ฉ.Triangle Fans

มีลักษณะคล้ายกับ Triangle Strip แต่จะใช้จุดปลายร่วมกัน จึงมีการนำไปใช้งานได้จำกัด

ตัวอย่าง สามเหลี่ยมที่ได้จากการเรนเดอร์ ดังรูป



รูปที่ 6.21 แสดงลักษณะของ Triangle Fan

6.10.3 การใช้งานดีไวซ์ (Using Device)

<u>6.10.3.1 การตรวจสอบฮาร์ดแวร์</u>

สามารถใช้เมดทอดต่อไปนี้ ในการตรวจสอบคุณสมบัติที่มีอยู่ในฮาร์ดแวร์ได้

IDirect3D8::CheckDeviceFormat

ใช้ในการตรวจสอบว่าฮาร์ดแวร์มีความสามารถในการเรนเดอร์และเก็บเท็กซ์เจอร์บนเซอร์เฟสชนิดใดบ้าง นอกจากนั้นยังใช้ตรวจสอบรูปแบบของสแตนซิลบัฟเฟอร์ (Depth-stencil buffer) และเดฟบัฟเฟอร์ (Depth buffer) ที่สามารถใช้งานได้

IDirect3D8::CheckDeviceType

ใช้ในการตรวจสอบว่าฮาร์ดแวร์มีความสามารถในการเร่งความเร็วในการประมวลผลด้านใดบ้าง ความสามารถของดีไวซ์ในการทำสวอพเชน (Swap Chain) ในการแสดงผล และตรวจสอบว่าดีไวซ์มีความสามรถในการเรนเดอร์ในรูปแบบการแสดงผลที่ใช้งานอยู่หรือไม่ ซึ่งก่อนจะใช้เมทธอดควรจะตรวจสอบก่อนว่าฮาร์ดแวร์มีรูปแบบของบัฟเฟอร์แต่ละชนิดอย่างไรบ้าง

6.10.3.2 การสร้างดีไวซ์

ในการสร้างดีไวซ์ ถ้าเขียนโปรแกรมโดยใช้ภาษา C/C++ จะต้องสร้าง Direct object ขึ้นมาก่อน จากนั้นจึงใส่ค่าซึ่งกำหนดคุณลักษณะของดีไวซ์ที่ต้องการใช้ข้อมูลโครงสร้าง D3DPRESENT_PARAMETER

หลังจากนั้นจึงเรียกเมทธอด IDirect3D::CreateDevice สำหรับสร้างดีไวซ์ ในที่นี้กำหนดให้ใช้ดีฟอลต์อแดปเตอร์ (defult adapter) สร้างดีไวซ์แบบ HAL และใช้การประมวลผลเวอร์เท็กซ์ด้วยซอฟท์แวร์ (software processing)

6.10.3.3 การกำหนดค่าของเมตริกซ์ทรานฟอร์เมชัน (setting trasformation)

ในภาษา c/c++ สามารถเรียกเมทธอด IDirect3DDevice::SetTransForm โดยส่งค่าเมตริกซ์ที่ต้องการกำหนดและชนิดของทรานส์ฟอร์มเมชัน ดังตัวอย่าง เป็นการกำหนดค่าของวิวทรานซ์ฟอร์มเมชัน

ในการกำหนดค่าทรานส์ฟอร์มเมชันในชนิดอื่นๆ สามารถทำได้โดยกำหนดค่า D3DTS_WORLD, DTDTS_VIEW และ DTDTS_PROJECTION ซึ่งค่าตัวแปรที่สามารถใช้ได้มีการกำหนดเอาไว้ใน D3DTRANSFORMSTATETYPE enumerated type นอกจากนี้ยังประกอบด้วยตัวแปรที่ใช้ในการทรานส์ฟอร์มเมชันเกี่ยวกับการทำแอนนิเมชัน (Animation) และกำหนดคู่อันดับของเท็กซ์เจอร์ด้วย ซึ่งในปริญญานิพนธ์ชิ้นนี้ในส่วนของการกำหนดแอนนิเมชันจะไม่มี

6.11 การประมวลผลเวอร์เท็กซ์ (vertex processing)

ใน Direct3Ddevice interface สนับสนุนการประมวลผลเวอร์เท็กซ์ทั้งในแบบซอฟแวร์และฮาร์ดแวร์ ซึ่งโดยปกติแล้วความสามารถของฮาร์ดแวร์ในเครื่องคอมพิวเตอร์แต่ละเครื่องจะแตกต่างกันไป แต่ความสามารถในการประมวลผลของซอฟแวร์จะเหมือนกันเสมอ

ค่าต่อไปนี้ใช้ในการควบคุมวิธีการในการประมวลผลเวอร์เท็กซ์สำหรับ HAL และ reference Device

- D3DCREATE_SOFTWARE_VERTEXPROCESSING
- D3DCREATE_HARDWARE_VERTEXPROCESSING
- D3DCREATE_MIXED_VERTEXPROCESSING

D3DCREATE_MIXED_VERTEXPROCESSING อนุญาติให้ดีไวซ์สามารถทำการประมวลผลเวอร์เท็กซ์ทั้งฮาร์ดแวร์และซอฟแวร์

6.12 การเรนเดอร์ (Rendering)

โปรแกรมใช้เมทธอดในกลุ่มของ DrawPrimitive ในการเรนเดอร์ภาพสามมิติ ซึ่งต่อไปนี้จะกล่าวถึงขั้นตอนการทำงานในแต่ละส่วน

6.12.1 การล้างเซอร์เฟส (Clearing Surface)

ก่อนที่จะเรนเดอร์ฉากในขณะหนึ่งๆ จะต้องล้างเซอร์เฟสที่อยู่ที่วิวพอร์ต (view port) ที่เราต้องการเสียก่อน เนื่องจากข้อมูลที่มีอยู่เดิมอาจทำให้ภาพที่ปรากฏออกมามีความผิดพลาดได้ นอกจากนี้การล้างเซอร์เฟสยังเป็นการเปลี่ยนสถานะของเดฟบัฟเฟอร์ให้อยู่ในสถานะที่เราต้องการอีกด้วย ซึ่งเราสามารถกำหนดได้ว่าให้ล้างเซอร์เฟสด้วยค่าสีหรือเท็กซ์เจอร์ใดๆ

เมทธอดที่ใช้ในการล้างเซอร์เฟส คือ IDirect3Ddevice8::Clear

6.12.2 การเริ่มต้นและจบฉาก (Beginning and Ending a Scene)

ในภาษา C/C++ ก่อนที่จะทำการเรนเดอร์จะต้องเรียกใช้เมทธอด IDirect3Ddevice8::BeginScene ซึ่ง BeginScene เป็นการสั่งให้ระบบทำการตรวจสอบข้อมูลภายในและตรวจสอบว่าเซอร์เฟสที่ต้องการเรนเดอร์มีอยู่จริงหรือไม่ นอกจากนั้นยังสามารถกำหนดค่าต่างๆ ภายใน (Internal flag) เพื่อกำหนดว่าจะมีการเริ่มประมวลผล หลังจากเรียกเมทธอดนี้แล้วจึงสามารถใช้คำสั่งต่างๆ ในการเรนเดอร์ได้ แต่การสั่งให้เรนเดอร์โดยไม่เรียกเมทธอด BeginScene ก่อนจะทำให้การเรนเดอร์ล้มเหลว

หลังจากเรนเดอร์เรียบร้อยแล้วจะต้องเรียกเมทธอดIDirect3Ddevice8::EndScene ซึ่งมีหน้าที่ล้างข้องมูลที่อยู่ในแคช (Cache) ตรวจสอบความถูกต้องของเซอร์เฟสที่ทำการเรนเดอร์และทำการล้างค่าต่างๆ ภายใน (Internal flag)

นั่นคือ คำสั่งในการเรนเดอร์จะต้องอยู่ระหว่างเมทธอด BeginScene และ EndScene เทานั้น

ตัวอย่างแสดงการใช้เมทธอด

```
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;
            }
```

6.12.3 การแสดงผลภาพ (Presenting a Scene)

เมทธอดเหล่านี้ทำหน้าที่ในการควบคุมสถานะของดีไวซ์ที่ใช้ในการแสดงผลบนจอมอนิเตอร์ ประกอบด้วย

- IDirect3DDevice8::Present
- IDirect3DDevice8::Reset
- IDirect3DDevice8::GetGammaRamp
- IDirect3DDevice8::SetGammRamp
- IDirect3DDevice8::GetRasterStatus

ซึ่งมีคำหลายคำที่เกี่ยวข้องกับการใช้งานและจำเป็นต้องทราบความหมายและการใช้ดังต่อไปนี้

1). Front Buffer คือ ส่วนของเมมโมรีที่จะได้รับการแปลงโดยกราฟิกส์อแดปเตอร์ (Graphics Adapter) แล้วแสดงผลออกทางหน้าจอคอมพิวเตอร์หรืออุปกรณ์อื่น

2). Back Buffer คือ เซอร์เฟสที่ข้อมูลจะถูกเลื่อนขั้นไปเป็น Front Buffer

3). Swap Chain คือ กลุ่มของ Back Buffer ที่เรียงลำดับเพื่อจะกลายเป็น Front Buffer ซึ่งใช้ในการแสดงผลแบบเต็มจอภาพ (Full Screen) จะมีการแสดงในรูปของ Flipping DDI (Device Driver Interface) ส่วนการแสดงผลแบบวินโดว์จะเป็น blitting DDI

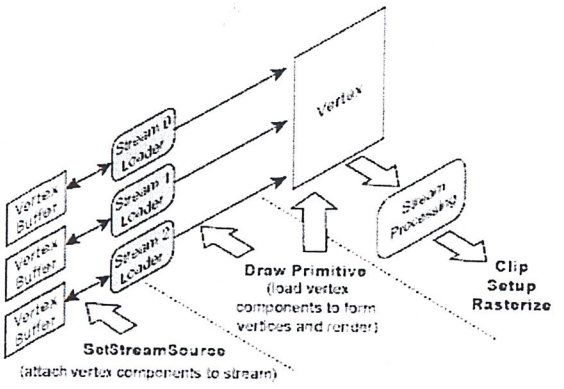
เนื่องจากสวอพเชนเป็นคุณสมบัติของ Direct3D ดังนั้น ดีไวซ์ทุกชนิดจะต้องมีคุณสมบัติในการทำสวอพเชนอย่างน้อย 1 สวอพเชนเสมอ

6.12.4 การเรนเดอร์ไพรมิทีฟ (Rendering Primitives)

- เวอร์เท็กซ์ดาต้าสตรีม (Vertex Data Stream)

Direct3D มี API ที่จัดการในส่วนของการเรนเดอร์ไพรมิทีฟ ซึ่งไพรมิทีฟที่จะเรนเดอร์ประกอบขึ้นจากข้อมูลของเวอร์เท็กซ์ซึ่งเก็บอยู่ในดาต้าบัฟเฟอร์ ข้อมูลของเวอร์เท็กซ์อาจจะประกอบด้วยตำแหน่ง สี ค่าแกนตั้งฉาก (Normal) เป็นต้น

องค์ประกอบของเวอร์เท็กซ์อาจจะประกอบขึ้นจากข้อมูลในเมเโมรีบัฟเฟอร์อันเดียวหรือหลายอันก็ได้ นั่นคือ องค์ประกอบของบัฟเฟอร์หนึ่งๆ สามารถแบ่งเก็บในโมรีบัฟเฟอร์มากกว่า 1 บัฟเฟอร์ได้ ดังนั้นในการเรนเดอร์ไพรมิทีฟจะต้องอ่านข้อมูลแล้วนำข้อมูลเหล่านี้มาประกอบกันจนเป็นเวอร์เท็กซ์ที่พร้อมจะประมวลผล ดังแผนภาพ



รูปที่ 6.22 แสดงการทำงานของบัฟเฟอร์

การเรนเดอร์ไพรมิทีฟประกอบด้วย 2 ขั้นตอน ดังนี้

1. การกำหนดค่าให้กับองค์ประกอบในสตรีม
2. การเรียกเมทธอดในกลุ่ม DrawPrimitive เพื่อเรนเดอร์สตรีมเหล่านั้น

- การกำหนดแหล่งของสตรีม (Setting the Stream Source)

เมทธอด IDirect3Ddevice8::SetStreamSource ใช้ในการกำหนดเวอร์เท็กซ์บัฟเฟอร์ให้กับสตรีมของข้อมูลของดีไวซ์ (device data stream) แต่การสร้างความสัมพันธ์ระหว่างสตีมข้อมูลเหล่านี้จะยังไม่เกิดขึ้นจนกว่าจะมีการเรียกเมทธอดในกลุ่มของ DrawPrimitive

สตรีม (Stream) คือ อาร์เรย์ของข้อมูลที่มีองค์ประกอบต่างๆ ซึ่งแต่ละองค์ประกอบอาจประกอบด้วยข้อมูลที่ย่อยลงไปอีกได้ ซึ่งใช้ในการแสดงตำแหน่ง แกนตั้งฉาก สี เป็นต้น ค่าพารามิเตอร์ Stride ใช้ในการกำหนดขนาดขององค์ประกอบเหล่านี้ โดยมีหน่วยเป็นไบต์ (byte)

- การเรนเดอร์โดยใช้เวอร์เท็กซ์และอินเด็กซ์บัฟเฟอร์ (Rendering form Vertex and Index Buffers)

ในการใช้เมทธอดที่เกี่ยวกับการวาดไพรมิทีฟ สามารถแบ่งออกเป็น 2 กลุ่ม คือ แบบที่มีอินเด็กซ์ และแบบไม่มีอินเด็กซ์ ซึ่งการอินเด็กซ์ข้อมูลในการประกอบไพรมิทีฟขึ้นมาเป็นรูปร่าง ใน Direct3D จะเก็บอินเด็กซ์ไว้ในอินเด็กซ์บัฟเฟอร์ โดยสามารถกำหนดอินเด็กซ์บัฟเฟอร์ที่จะใช้งานได้โดยเรียกเมทธอด Idirect3Ddevice8::SetIndies

เมทธอดที่ใช้ในการวาดไพรมิทีฟ

-IDirect3DDevice8::DrawPrimitive

-IDirect3DDevice8::DrawIndexPrimitive

6.13 สถานะของดีไวซ์ (Device States)

ใน Direct3D ดีไวซ์ทำงานโดยการคำนวณค่าสถานะ โดยโปรแกรมจะกำหนดค่าสถานะเหล่านี้ทั้งสถานะที่เกี่ยวข้องกับการคำนวณแสง การเรนเดอร์ และการควบคุมโมดูลส่วนทรานส์ฟอร์มเมชัน (transformation module) เป็นต้น

ซึ่งสถานะของดีไวซ์สามารถแบ่งได้เป็น 2 แบบ ดังนี้

6.13.1 Render States

การควบคุมคุณสมบัติในส่วนของการลงจุดสี (Rasterization module) ซึ่งสามารถใช้ในกาiเปลี่ยนแปลงคุณสมบัติการเรนเดอร์ กำหนดเชดดิ้งที่จะใช้งาน กำหนดคุณสมบัติของหมอกและกระบวนการในการลงจุดสีอื่นๆ

การกำหนดคุณสมบัติเหล่านี้โปรแกรมที่เขียนในภาษา c/c++ สามารถทำได้โดยการเรียกเมทธอด Idirect3Ddevice8::SetRendState ซึ่งได้มีการกำหนดค่าสถานะของดีไวซ์ไว้ในชนิดข้อมูล D3DRENDERSTATETYPE enumerated type

การประมวลผลเวอร์เท็กซ์แบบฟังก์ชันตายตัว (Fix Function Vertex Processing) สามารถกำหนดคุณสมบัติการทำงานได้โดยการกำหนดค่าสถานะเรนเดอร์ แต่ถ้าใช้การประมวลเวอร์เท็กซ์แบบโปรแกรมได้ (Programmable Vertex Processing) คุณสมบัติหลายๆ อย่างจะไม่มีผล

นอกจากเมทธอด SetRenderState แล้วยังมีเมทธอดอื่นๆ ที่เกี่ยวข้องกับการกำหนดการทำงานของไปป์ไลน์ประมวลผลเวอร์เท็กซ์แบบฟังก์ชันตายตัว (Fix Function Vertex Processing Pipeline) ซึ่งเกี่ยวข้องกับการกำหนดค่าเมตริกซ์ทรานส์ฟอร์มเมชัน (transformation) การกำหนดแมตทีเรียล และกำหนดแสงดังนี้

- IDirect3DDevice8::SetTransform
- IDirect3DDevice8::SetMaterial
- IDirect3DDevice8::SetLight
- IDirect3Ddevice8::LigthEnable

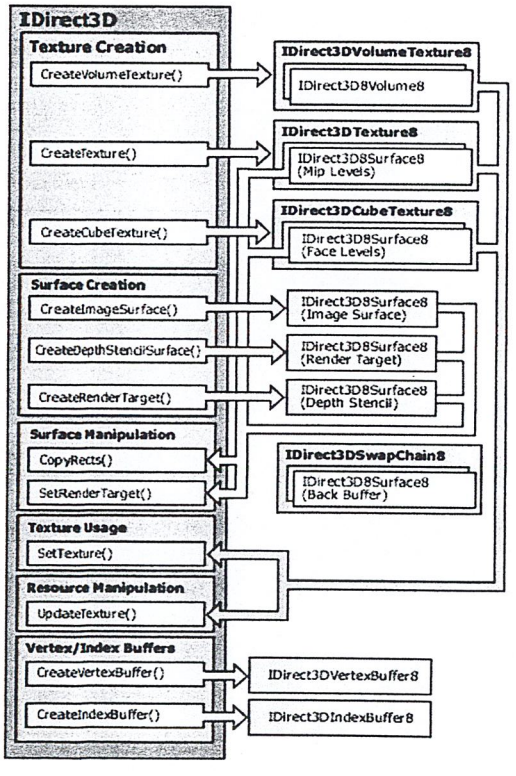
6.13.2 สถานะของเท็กซ์เจอร์ (Texture Stage State)

ใช้ในการควบคุมรูปแบบของการทำเท็กซ์เจอร์และการกำหนดฟิลเตอร์ (filter) ที่จะใช้งาน โปรแกรมที่เขียนในภาษา c/c++ สามารถกำหนดคุณสมบัติเหล่านี้ได้โดยการเรียกเมทธอด IDirect3DDevice8::SetTextureStageState และชนิดข้อมูล D3DTEXTURESTAGETYPE enumerated type ได้กำหนดค่าสถานะของเท็กซ์เจอร์ที่สามารถกำหนดค่าได้ไว้ให้แล้ว โดยการส่งค่าตัวแปร D3DTEXTURESTAGETYPE enumerated type เป็นพารามิเตอร์แรกในเมทธอด SetTextureStageState

นอกจากนั้น การกำหนดเท็กซ์เจอร์เซอร์เฟสที่จะใช้งานสามารถทำได้โดยการเรียกเมทธอด IDirect3DDevice::SetTexture

6.14 รีซอร์สใน Direct3D (Direct3D Resources)

คือ เท็กซ์เจอร์และบัฟเฟอร์ที่ใช้ในการเรนเดอร์ โปรแกรมมีความจำเป็นต้องสร้าง โหลด คัดลอก และใช้รีซอร์สเหล่านี้ ซึ่งประกอบด้วย เท็กซ์เจอร์ เวอร์เท็กซ์บัฟเฟอร์ และอินเด็กซ์บัฟเฟอร์ ความสัมพันธ์ของรีซอร์ส (Resource Relationships) เป็นดังแผนภาพดังต่อไปนี้



รูปที่ 6.23 แสดงการทำงานของรีซอร์ส

ลูกศรวิ่งจากซ้ายไปขวา แสดงว่า เมทธอดสร้างรีซอร์สชนิดนั้นได้ ลูกศรวิ่งจากขวาไปซ้าย แสดงว่า รีซอร์สชนิดนั้นสามารถใช้เป็นอาร์กิวเมนต์ (argument) ในเมทธอดนั้นได้

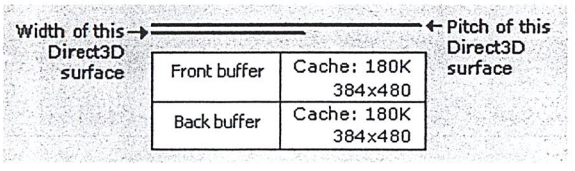
6.15 เซอร์เฟส (Surface)

คือ พื้นที่ในเมมโมรี่ส่วนที่ใช้ในการแสดงผล (display memory) ซึ่งโดยทั่วไปจะอยู่ในเมมโมรี่ที่อยู่บนการ์ดแสดงผล (Display Card) แต่ก็สามารถอยู่ในเมมโมรี่ระบบ (system memory) ได้เช่นกัน

เซอร์เฟสจะถูกบรรจุอยู่ใน IDirect3DSurface8 Interface

6.15.1 ความกว้างและค่าพิทช์ (Width and Pitch)

ความกว้างและพิทช์ มีความหมายต่างกัน ความกว้างในทาง Direct3D หมายถึงมิติของเซสเฟสในทางลอจิก (Logical) ความกว้างมีหน่วยเป็นพิกเซล (Pixel) ดังนั้นจึงมีค่าเท่ากันเสมอไม่ว่าพิกเซลจะมีขนาดกี่บิตก็ตาม แต่พิทช์ มีความหมายถึง ระยะของข้อมูลเป็นจำนวนไบต์ โดยมักรวมเมมโมรี่ในส่วนที่เป็นแคช (Cache) เข้าไปด้วย ดังรูป



รูปที่ 6.24 แสดงลักษณะของความกว้างและพิทช์

ค่าพิทช์มีความจำเป็นในกรณีที่ต้องการเข้าถึงเมมโมรี่โดยตรง ซึ่งต้องคำนึงถึงขนาดเมมโมรี่โดยห้ามไปเขียนในส่วนของแคชที่สงวนเอาไว้

6.15.2 รูปแบบของเซอร์เฟส (surface format)

รูปแบบของเซอร์เฟสเป็นตัวกำหนดชนิดของข้อมูลในแต่ละพิกเซลซึ่งเก็บอยู่ในเซอร์เฟสเมมโมรี ใน Direct3D ใช้ D3DFORMAT ซึ่งเป็นสมาชิกของ D3DSURFACE_DESC ในการอธิบายรูปแบบของเซอร์เฟส โดยสามารถหาค่ารูปแบบของเซอร์เฟสใดๆ ได้โดยเรียกเมทธอด IDirect3DSurface8::GetDesc

6.16 แสง (Light)

เมื่อมีการกำหนดให้มีการคำนวณแสงใน Direct3D ในกระบวนการลงสี (Rasterization) ซึ่งเป็นขั้นตอนสุดท้ายในการเรนเดอร์ไปป์ไลน์ จะมีการคำนวณค่าสีของพิกเซลด้วยผลรวม

ระหว่างค่าแมตทีเรียล เทกเซล ซึ่งได้มาจากการปะเท็กซ์เจอร์ (Texture mapping) และความเข้มข้นของแสงและสีที่ปรากฏอยู่ในฉากซึ่งเกิดจากแหล่งแสงชนิดต่างๆ (Light source) หรือระดับของแสงแอมเบียน (Ambient ligth level) เมื่อผู้ใช้กำหนดให้ใช้ Direct3D ในส่วนของแสงและแมตทีเรียล นั่นหมายความว่า ต้องการให้ Direct3D จัดการรายละเอียดทั้งหมดให้ แต่สำหรับผู้ที่มีความชำนาญมากๆ ก็สามารถคำนวณค่าเกี่ยวกับแสงได้เองเช่นกัน

การใช้แสงและแมตทีเรียลจะทำให้ภาพที่ได้มีความสวยงามและสมจริงขึ้นอย่างมาก ค่าของแมตทีเรียลเป็นการกำหนดคุณสมบัติในการสะท้อนแสงของพื้นผิว ส่วนค่าของแสงเป็นการกำหนดแสงที่ส่งออกเพื่อให้สะท้อน ดังนั้น ถ้าต้องการจะใช้แสงเพื่อเพิ่มความสมจริงและสวยงามให้วัตถุจำเป็นที่ต้องกำหนดค่าแมตทีเรียลให้วัตถุด้วยเสมอ (มิฉะนั้นวัตถุจะอยู่ในสภาพที่มองไม่เห็น)

ทั้ง Direct Ligth และ Ambient Ligth ต่างเป็นวัตถุที่ส่องแสงออกมาได้ แต่วัตถุทั้งสองไม่ขึ้นแก่กันและกัน และให้ผลลัพธ์ในการทำงานที่แตกต่างกัน

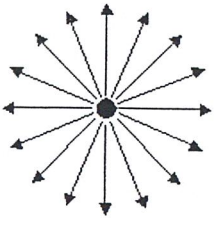
Direct Light จะต้องมีทิศทางและสีเสมอ ค่าเหล่านี้ถูกใช้คำนวณใน Shading algorithm เช่นกอรอดเชดดิ้ง (Gouraud shading) ด้วย แสงที่มีชนิดต่างกันจะมีวิธีในการเปล่งแสงแตกต่างกันไป ซึ่งจะทำให้เกิดเอฟเฟ็กต์ต่างๆ กัน แสงชนิดนี้สามารถกำหนดได้ด้วยการเรียกใช้เมทธอด IDirect3DDevice::SetRenderState โดยใส่ค่า D3DRS_AMBIENT ในพารามิเตอร์ที่กำหนดชนิดของสถานะ (State parameter) และค่าสีที่ต้องการในพารามิเตอร์มูลค่า (Valve parameter)

6.16.1 ชนิดของแสง (Light Object)

ใน Direct3D มีแสงอยู่ 3 ชนิด คือ จุดแสง (point light) สปอตไลท์ (Spotligth) และแสงทิศทาง (directional light) ซึ่งแสงแต่ละชนิดมีรายละเอียดดังนี้

6.16.1.1 จุดแสง

จุดแสงมีสีและตำแหน่งในฉาก แต่ไม่มีทิศทางเดียว มันจะให้แสงเท่าๆ กันทุกทิศทุกทาง ดังรูป

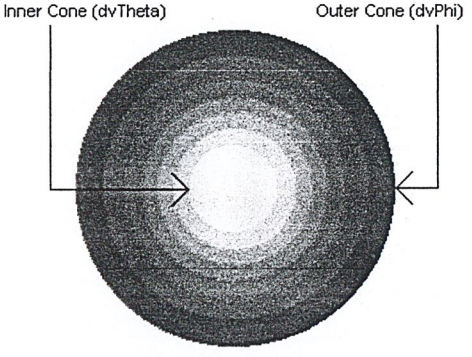


รูปที่ 6.25 แสดงลักษณะของจุดแสง

หลอดไฟฟ้าแบบหลอดไส้เป็นตัวอย่างของจุดแสง ค่าของแสงที่ออกมาจากจุดแสงถูกกำหนดด้วยพารามิเตอร์ Attenuation ซึ่งกำหนดลักษณะการจางลงของความเข้มแสง และระยะทางที่แสงส่องไปถึง (Range) ใช้ตำแหน่งของจุดแสง ระยะทางที่แสงเดินทาง และค่าตั้งฉากของเวอร์เท็กซ์ในการคำนวณความสว่างของพื้นผิว

6.16.1.2 สปอตไลท์ (Spotlight)

สปอตไลท์มีทั้งสี ตำแหน่ง และทิศทาง แสงที่เปล่งออกมาจากสปอตไลท์มีลักษณะเป็นกรวยซึ่งส่วนที่อยู่ในสุดจะมีความสว่างมากที่สุดและค่อยๆ ลดลงเมื่อห่างออกไปเรื่อยๆ ความต่างของความเข้มแสงแสดงดังรูป



รูปที่ 6.26 แสดงลักษณะของ สปอตไลท์

ไฟฉายเป็นตัวอย่างของสปอตไลท์ ความเข้มของแสงที่เปล่งออกจากสปอตไลท์ถูกกำหนดโดยค่า Falloff attenuation และระยะทางที่แสงเดินทางไปถึง (range) เนื่องจากมีพารามิเตอร์จำนวนมากการคำนวณแสงจากสปอตไลท์จึงมีความซับซ้อนมากที่สุดและเปลืองเวลาในการประมวลผลที่สุด

6.16.1.3 แสงทิศทาง (Directional light)

มีแต่ค่าของสี และทิศทาง ไม่มีค่าซึ่งแสดงตำแหน่ง เนื่องจากเปล่งแสงในลักษณะคู่ขนาน นั่นคือ เป็นแสงที่ตกกระทบวัตถุทั้งหมดในฉากด้วยทิศทางเดียวกัน เช่นเดียวกับแสงจากดวงอาทิตย์ (เนื่องจากเดินทางมาเป็นระยะทางที่ไกลมากจึงเสมือนว่าเดินทางขนานกันมา) แสงทิศทางไม่มีค่า Attenuation หรือระยะทาง ดังนั้น การคำนวณจึงมีความซับซ้อนน้อยที่สุดและเปลืองการประมวลผลน้อยที่สุดเช่นกัน

6.16.2 คุณสมบัติของแสง (Light properties)

คุณสมบัติของแสงอธิบายชนิดของแหล่งแสง (light source) และสี นอกจากนั้นยังกำหนดค่าที่มีความจำเป็นสำหรับแสงในแต่ละชนิด (แสงแต่ละชนิดใช้ค่าพารามิเตอร์ไม่เหมือนกัน) ใน Direct3D ใช้โครงสร้างข้อมูล D3DLIGHT8 ในการเก็บข้อมูลเกี่ยวกับแสงทั้งหมด ซึ่งประกอบด้วยข้อมูลที่แบ่งเป็นกลุ่มต่างๆ ได้ดังนี้

- ชนิดของแสง (Light Type)
- สีของแสง (Light color)
- ค่าแมตทีเรียลของวัตถุ (Color vertices)
- ตำแหน่ง ระยะทาง และ attenuation ของแสง
- ทิศทางของแสง (Light direction)
- คุณสมบัติของสปอตไลท์

6.16.3 การใช้แสง (Using Lights)

6.16.3.1 การกำหนดค่าพารามิเตอร์ของแสง (setting light properties)

ในภาษา C/C++ สร้างและกำหนดค่าของแสงได้ด้วยการกำหนดค่าพารามิเตอร์ต่างๆ ที่มีอยู่ในโครงสร้างข้อมูล D3DLIGHT8 แล้วเรียกเมทธอด Idirect3Ddevice::SetLight ซึ่งต้องระบุค่าอินเด็กซ์ของแสงที่จะกำหนดด้วย เนื่องจากใน Direct3D สามารถกำหนดแหล่งแสงได้พร้อมกันสูงสุด 8 แหล่งเท่านั้น

ตัวอย่างการกำหนดค่าของแหล่งแสง

```
/*
* For the purposes of this example, the d3dDevice variable
* is a valid pointer to an IDirect3DDevice8 interface.
 */
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;


// Position it high in the scene and behind the user.
```

```
// Remember, these coordinates are in world space, so
// the user could be anywhere in world space, too.
// For the purposes of this example, assume the user
// is at the origin of world space.
d3dLight.Position.x = 0.0f;
d3dLight.Position.y = 1000.0f;
3dLight.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 ไปแล้วสามารถเปลี่ยนแปลงค่าของแสงได้ด้วยการเรียกเมทธอด SetLight ด้วยค่าของแสงที่ต้องการอีกครั้ง

6.16.3.2 การสั่งให้มีการคำนวณแสง (Enabling and Disabling Lights)

เมื่อกำหนดค่าของแหล่งแสงในอินเด็กซ์ใดๆ แล้ว การที่จะให้ Direct3D นำค่าพารามิเตอร์เหล่านั้นไปใช้ในการคำนวณทำได้โดยการเรียกใช้เมทธอด IDirect3Ddevice::LightEnable ซึ่งโดยปกติแหล่งแสงจะไม่มีการนำไปใช้คำนวณ (Disable) ไว้ เมทธอด LightEnable ต้องการอาร์กิวเมนต์ 2 ตัว ค่าแรกคือ อินเด็กซ์ของแหล่งแสง ส่วนค่าที่สองคือ พารามิเตอร์ที่บอกว่าให้นำค่าไปคำนวณหรือไม่ (TRUE ถ้าต้องการให้คำนวณ และ FALSE ถ้าไม่ต้องการ)

ตัวอย่างการกำหนดให้ Direct3D 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.
}
```

แต่ถ้ามีการสั่งให้นำค่าในอินเด็กซ์ที่ไม่มีการกำหนดค่าไว้ไปคำนวณ Direct3D จะนำค่าในตารางต่อไปนี้ไปใช้ในการคำนวณแทน

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 ตารางที่ใช้ในการคำนวณเมื่อไม่มีการกำหนดค่า

6.16.3.3 การหาค่าของแสงที่กำหนดไว้แล้ว (Retrieving Light Properties)

สามารถหาค่าคุณสมบัติของแสงที่มีอยู่แล้วได้ด้วยการเรียกเมทธอด IDirect3DDevice::GetLight แล้วส่งค่าอาร์กิวเมนต์ตัวแรกเป็นอินเด็กซ์ของแสงที่ต้องการดังตัวอย่าง

```
/*
 * For the purposes of this example, the pd3dDevice variable
 * is a valid pointer to an IDirect3DDevice8 interface.
 */
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.
}
```

ถ้าเรากำหนดค่าอินเด็กซ์ของแสงที่ไม่มีอยู่ใน Direct3D (มีเฉพาะค่า 0-7 เท่านั้น เนื่องจากมีแสงได้สูงสุดเพียง 8 แหล่งแสง) เมทธอด GetLight จะส่งค่า D3DERR_INVALIDCALL คืนมา

6.17 แมตทีเรียล (Materials)

แมตทีเรียลเป็นค่าที่อธิบายวิธีการสะท้อนของโพลีกอนเนื่องจากแหล่งแสงที่มีอยู่ในฉาก มีความสำคัญในการคำนวณค่าดังต่อไปนี้

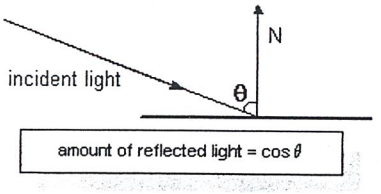
- วิธีการที่พื้นผิวสะท้อนแสงดิฟฟิวส์ (Diffuse light) และแสงแอมเบียน
- ลักษณะความมันวาว (Specular highlight)
- โพลีกอนมีคุณสมบัติในการเปล่งแสงหรือไม่อย่างไร (Emissive)

6.17.1 คุณสมบัติของแมตทีเรียล (Material properties)

มีรายละเอียดดังต่อไปนี้

วิธีการสะท้อนแสงดิฟฟิวส์และแสงแอมเบียน (Diffuse and Ambient Reflection)

ค่า Diffuse และ Ambient เป็นสมาชิกในโครงสร้างข้อมูล D3DMATERIAL8 ซึ่งอธิบายการสะท้อนของแสงทั้งสองชนิด แต่โดยปกติแล้วแสงดิฟฟิวส์จะส่งผลมากกว่าแสงแอมเบียน คุณสมบัติในการสะท้อนแสงดิฟฟิวส์จึงเป็นคุณสมบัติที่ปรากฏให้เห็นเด่นชัดกว่า เนื่องจากแหล่งแสงส่วนมากจะมีทิศทาง ดังนั้นการสะท้อนแสงดิฟฟิวส์จะสะท้อนได้ดียิ่งขึ้นถ้าทิศทางของแหล่งแสงขนานกับทิศทางตั้งฉากของเวอร์เท็กซ์ แต่ถ้ามีมุมระหว่างแหล่งแสงกับทิศทางตั้งฉากของเวอร์เท็กซ์เกิดขึ้นมากเท่าไหร่ก็ยิ่งทำให้ความเข้มแสงอ่อนลงเท่านั้น ปริมาณความเข้มแสงที่กระทบผิวคือ ค่าโคไซน์ (Cosine) ของมุมระหว่างทิศทางของแหล่งแสงและทิศทางตั้งฉากของเวอร์เท็กซ์ ดังรูป



รูปที่ 6.27 แสดงมุมระหว่างทิศทางของแหล่งแสงและทิศตั้งฉากของเวอร์เท็กซ์

การสะท้อนแสงแอมเบียนก็เหมือนกับแหล่งแสงแอมเบียน คือ ไม่มีทิศทาง การสะท้อนของแสงแอมเบียนมีผลต่อวัตถุน้อย แต่มีผลต่อทุกวัตถุในฉากและส่งผลถึงภาพที่ปรากฏเหมือนกัน ดังนั้นจึงมีความจำเป็นอย่างยิ่งที่จะต้องใช้คุณสมบัติการสะท้อนของแสงทั้งในแสงดิฟฟิวส์และแสงแอม

เบียนในการกำหนดสีของวัตถุที่อยู่ในฉาก และโดยปกติแล้วจะใช้ค่าเดียวกัน เช่น ถ้าต้องการกำหนดสีให้ผลึกคริสตัลเป็นสีฟ้า ก็จะกำหนดให้คุณสมบัติในการสะท้อนแสงสีฟ้าทั้งในค่าของดิฟฟิวส์และแอมเบียน ดังนั้น ถ้าในฉากมีแหล่งแสงสีขาว ก็มองเห็นคริสตัลเป็นสีฟ้า แต่ถ้าในฉากมีแต่แสงสีแดงก็จะเห็นคริสตัลเป็นสีดำ เป็นต้น

การเปล่งแสง (Emission)

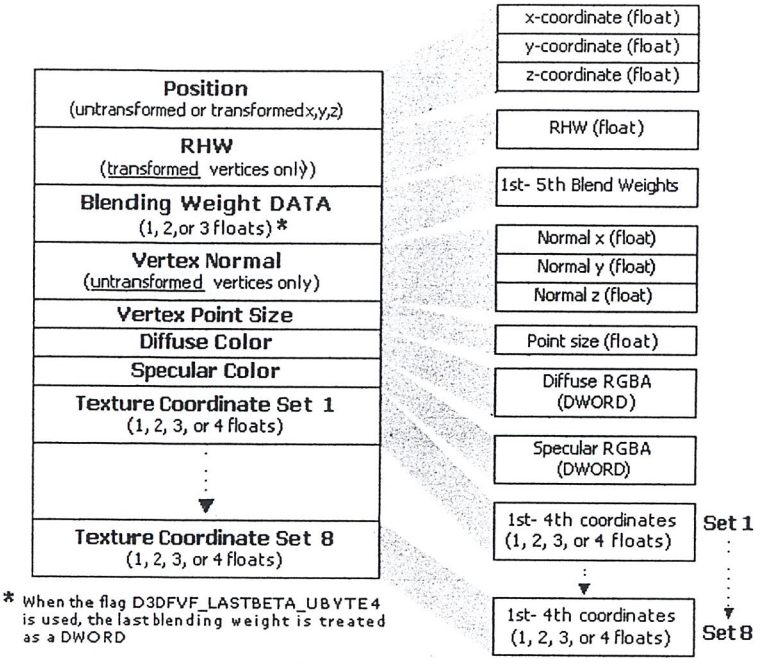
ใช้ในการทำให้วัตถุเหมือนว่าเปล่งแสงได้ ค่าอีมิสซีฟ (emissive) ซึ่งเป็นสมาชิกในโครงสร้างข้อมูล D3DMATERIAL8 ใช้ในการกำหนดสีและความโปร่งใสของแสงที่เปล่งออกมาจากวัตถุ เราใช้ค่าการเปล่งแสงของวัตถุในกรณีที่ต้องการทำให้วัตถุเหมือนกับเปล่งแสงได้โดยไม่ต้องการให้ประมวลผลแหล่งแสงเพิ่มเนื่องจากจะใช้การคำนวณมากกว่า แต่สิ่งที่แตกต่างจากการใช้แหล่งแสงคือ แสงที่เปล่งออกมาจะไม่มีผลต่อวัตถุตัวอื่น ๆ ในฉากเลย

ความมันวาว (Specular Reflection)

เป็นค่าที่กำหนดลักษณะความมันวาวบนวัตถุ ซึ่งทำให้วัตถุมีความสว่างขึ้น ในโครงสร้างข้อมูล D3DMATERIAL8 ใช้สมาชิก 2 ตัวในการกำหนดลักษณะความมันวาวของวัตถุ คือสเปคคูลาร์ใช้กำหนดสีของความมันวาวที่ปรากฏ และพาวเวอร์ (Power) ใช้กำหนดขนาดของความมันวาว ดังภาพอย่างด้านซ้ายมือ กำหนดค่าพาวเวอร์เท่ากับ 10 และค่าสเปคคูลาร์เป็นสีขาว ส่วนด้านขวามือไม่ได้กำหนดค่าความมันวาวให้ (กำหนดให้แต่ค่าดิฟฟิวส์ และค่าแอมเบียน)

6.18 รูปแบบของเวอร์เท็กซ์ (Vertex Formats)

Flexible Vertex Format (FVF) ใช้อธิบายว่าในข้อมูลเวอร์เท็กซ์มีการเก็บข้อมูลอะไรไว้บ้างซึ่งอยู่ในดาต้าสตรีมเดียวกัน (Data stream) ซึ่งปกติแล้วจะใช้ในการกำหนดข้อมูลสำหรับไปป์ไลน์ที่ใช้ประมวลผลเวอร์เท็กซ์แบบฟังก์ชันตายตัว (Fixed Function Processing Pipeline) แผนภาพต่อไปนี้แสดงลำดับของข้อมูลที่สามารถเลือกใช้งานได้ โดยการเก็บข้อมูลจะต้องเก็บเรียงตามลำดับการเก็บในแผนภาพนี้ด้วย



รูปที่ 6.28 แสดงลำดับการเก็บข้อมูล

ค่าคู่อันดับแสดงตำแหน่งของเท็กซ์เจอร์ (texture coordinate) สามารถประกาศใช้ชนิดข้อมูลแบบอื่นก็ได้โดยมีค่าได้ตั้งแต่ 1 ถึง 4 ค่า

การเขียนโปรแกรมโดยปกติแล้วจะไม่ได้ใช้องค์ประกอบที่มีอยู่ทั้งหมดเนื่องจากข้อมูลบางค่าไม่สามารถใช้พร้อมกันได้ เช่น D3DFVF_XYZ และ D3DFVF_XYZRHW

การใช้อินเด็กซ์เวอร์เท็กซ์เบลนดิ้ง (Indexed vertex blending) จะต้องประกาศค่า D3DFVF_LASTBETA_UBYTE4 ในการกำหนดข้อมูลสำหรับการทำแอนนิเมชันด้วย

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
};
```

ตัวอย่างการกำหนดชนิดข้อมูลในเวอร์เท็กซ์โดยใช้ D3DFVF_LAST_UBYTflag

```
#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
};
```

6.19 เท็กซ์เจอร์(Texture)

เป็นสิ่งที่ช่วยสร้างความสวยงามและสมจริงให้กับวัตถุ 3 มิติ ดังนั้น Direct3D จึงสนับสนุนคุณสมบัติในการจัดการเท็กซ์เจอร์ไว้จำนวนมาก เพื่อให้ผู้พัฒนาสามารถนำคุณสมบัติเหล่านี้ไปใช้ในการประมวลผลเท็กซ์เจอร์ในระดับสูงได้

เท็กซ์เจอร์ คือ ภาพบิตแมพที่ปะลงบนไพรมิทีฟ ซึ่งทำให้ภาพที่ได้มีลักษณะที่เหมือนจริงยิ่งขึ้น เช่น ถ้าเราต้องการจำลองต้นไม้ ก็นำภาพบิตแมพที่เป็นลายต้นไม้มาปะในส่วนที่เป็นลำต้น และนำค่าบิตแมพของใบไม้มาแปะที่ใบไม้ก็จะทำให้ได้วัตถุที่ดูเหมือนต้นไม้ยิ่งขึ้น

ใน Direct3D เราสามารถกำหนดเท็กซ์เจอร์ปะบนไพรมิทีฟได้ด้วยการเรียกใช้เมทธอด IDirect3DDevice::SetTexture

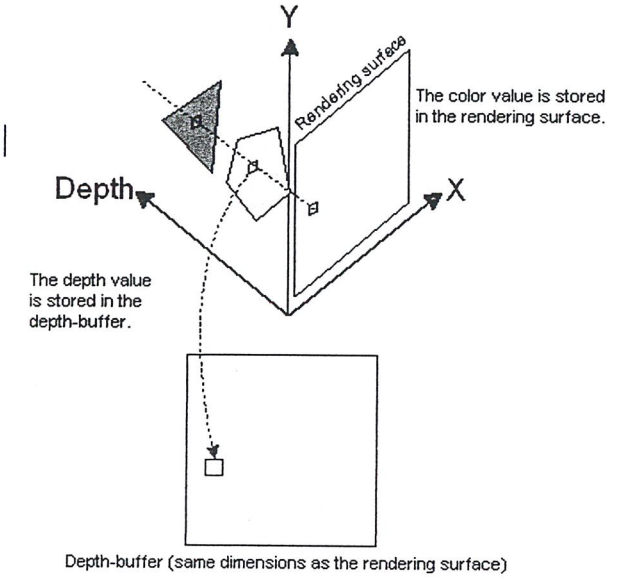
เนื่องจากคุณสมบัติในการประมวลผลเท็กซ์เจอร์มีจำนวนมากและซับซ้อนอย่างยิ่ง การใช้งานคุณสมบัติเหล่านี้ในส่วนของการทำแขนกลจำลองจึงขอละไว้

6.20 เดฟบัฟเฟอร์ (Depth buffer)

หรือที่รู้จักในชื่อของ Z-Buffer และ W-Buffer คือ คุณสมบัติของฮาร์ดแวร์ในการเก็บข้อมูลเกี่ยวกับความลึกของพิกเซล ซึ่งจะใช้ในการคำนวณรวมกับข้อมูลที่อยู่ในเรนเดอร์เซอร์เฟสที่ใช้เรนเดอร์อยู่ในขณะนั้น (render-target surface) ในการตัดสินว่าจุดบนวัตถุใดจะเป็นจุดที่สามารถมองเห็นได้ในกรณีที่วัตถุซ้อนทับกัน

ในตอนเริ่มต้น ข้อมูลที่เก็บอยู่ในเดฟบัฟเฟอร์จะเป็นค่ามากที่สุดที่เป็นไปได้ และค่าของสีที่กำหนดก็จะเป็นค่าที่กำหนดไว้แล้วตั้งแต่เริ่มแรก (Defult) หรือค่าสีที่ใช้เป็นฉากหลัง หรือค่าสีของเท็กซ์เจอร์ที่ใช้เป็นฉากหลังของจุดนั้นๆ เมื่อมีการวาดโพลีกอนด้วยคำสั่งใด ๆ ก็ตาม

ทุกๆโพลีกอนจะต้องนำมาหาจุดตัดในแกน x,y เพื่อนำค่าสีนั้นมาเป็นค่าสีของจุดตัดนั้นและเก็บค่า z ซึ่งเป็นค่าความลึกของจุดไว้ด้วย ในกรณีที่ใช้ Z-Buffer จุดสีที่มีความลึกน้อยกว่าจะได้รับการบันทึกค่าของสีและความลึกของภาพไว้ ดังภาพ



รูปที่ 6.29 แสดงการทำงานของเดฟบัฟเฟอร์

อุปกรณ์ที่มีขายทั่วไปตามท้องตลาดส่วนใหญ่สนับสนุนการใช้งานบัฟเฟอร์ที่เป็น Z-Buffer มากกว่าการใช้งานที่เป็น W-Buffer

6.21 เวอร์เทกบัฟเฟอร์ (Vertex buffer)

คือ เมมโมรี่ที่ใช้เก็บข้อมูลของเวอร์เท็กซ์ และสามารถนำข้อมูลที่มีอยู่ในบัฟเฟอร์นี้ไปเรนเดอร์โดยเรียกใช้เมทธอดในกลุ่มของการวาดไพรมิทีฟ โดยเวอร์เท็กซ์บัฟเฟอร์บรรจุอยู่ใน IDirectBuffer8Interface

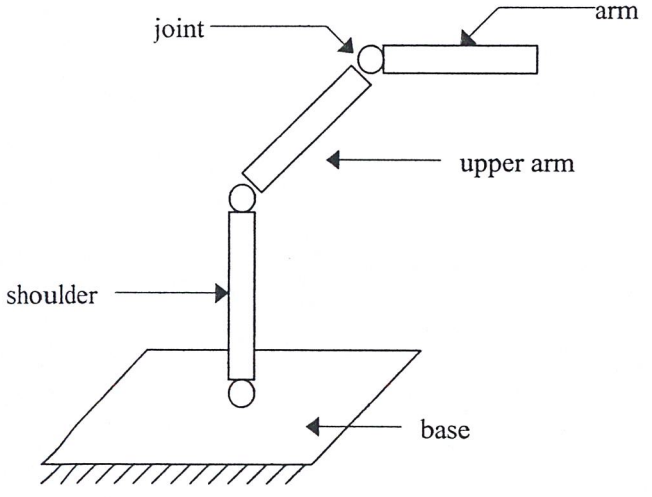
6.22 อินเด็กซ์บัฟเฟอร์ (Index buffer)

คือ เมมโมรี่ที่ใช้เก็บข้อมูลอินเด็กซ์ ซึ่งใช้ในการกำหนดค่าออฟเซต (Offset) ในเวอร์เท็กซ์บัฟเฟอร์และใช้ในการเรนเดอร์ไพรมิทีฟ โดยอินเด็กซ์บัฟเฟอร์บรรจุอยู่ใน IDirect3DIndexBuffer8 Interface

บทที่ 7
การสร้างภาพ 3 มิติและขั้นตอนการทำงานของโปรแกรม

7.1 การสร้างภาพ 3 มิติจากโปรแกรมมายา (Maya version 3.0)

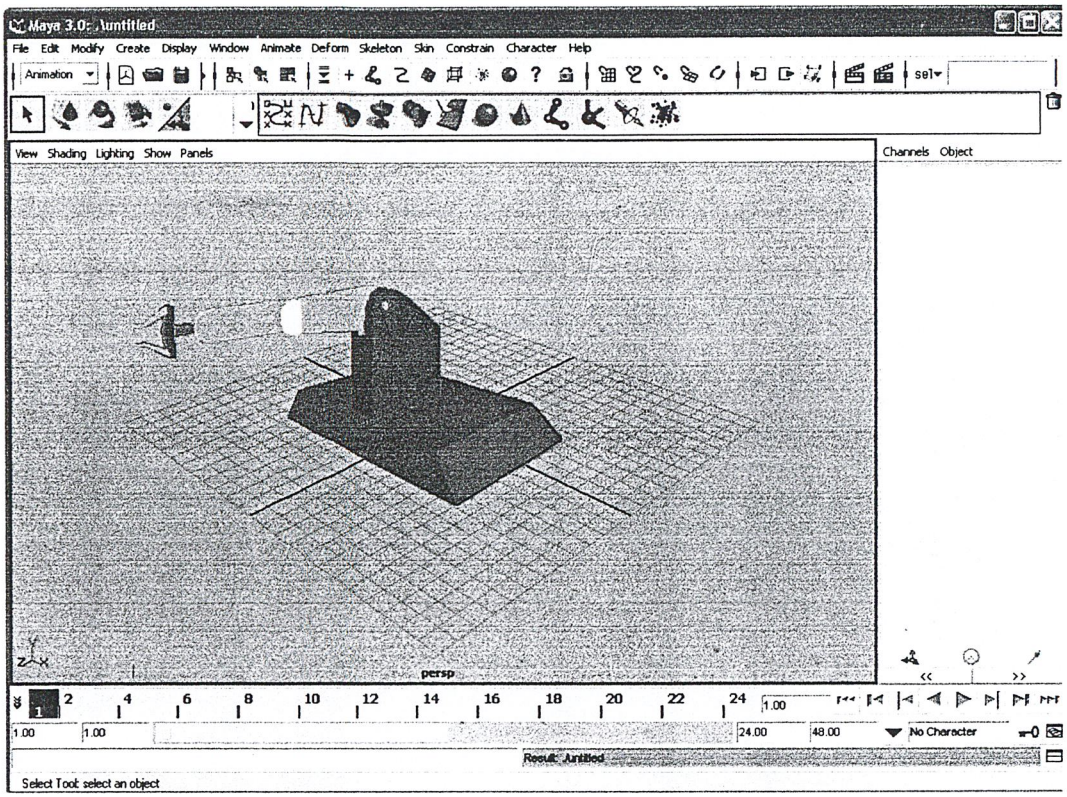
เนื่องจากภาพแขนกลที่สร้างขึ้นมีลักษณะเป็นข้อต่อ (Joint) ซึ่งสามารถหมุนได้และเชื่อมต่อกันด้วยลิงค์ (Link) ประกอบกันขึ้นเป็นรูปแขน ปลายสุดของแขนจะอยู่ในลักษณะที่มือจับ (Gripper) หรือเครื่องมืออื่น ๆ เช่น เครื่องเจาะหรือเครื่องตัด ซึ่งเราเรียกว่าทูล (Tool) ฉะนั้นเวลาเราสร้างโมเดล (Model) ขึ้นมาในลักษณะที่มีข้อต่อตามต้องการแล้ว เราจะต้องทำการเชื่อมต่อกันระหว่างแขนกลแต่ละส่วนตั้งแต่ฐาน (Base) ไปจนถึงปลายมือหรือทูล ซึ่งอยู่ในรูปไฮราคี (Hierarchy) หรือแบบสืบทอดกันไปเรื่อย ๆ คือส่วนของโมเดลที่มีการเคลื่อนที่แล้วมีส่วนอื่น ๆ ซึ่งอยู่ติดกันเคลื่อนที่ตามไปด้วย ส่วนที่เคลื่อนที่ตามจะถูกเรียกว่าลูก (Child) และส่วนที่เคลื่อนที่ในตอนต้นเรียกว่าแม่ (Parent)



รูปที่ 7.1 ภาพจำลองแสดงการเชื่อมต่อส่วนต่าง ๆ ที่ประกอบกันขึ้นเป็นรูปแขนกล

จากภาพ เบส (Base) จะเป็นแพเรนท์ของโชลเดอร์ (Shoulder), อัพเพอร์ อาร์ม (Upper arm) และอาร์ม (Arm)

ส่วนโชลเดอร์ (Shoulder) จะเป็นแพเรนท์ของอัพเพอร์ อาร์ม (Upper arm) และอาร์ม (Arm) ส่วนอัพเพอร์ อาร์ม (Upper arm) จะมีลูกเพียงส่วนเดียวคืออาร์ม (Arm) เท่านั้น ซึ่งในโปรแกรมมายาเราสามารถกำหนดความเป็นไฮราคีนี้ได้ด้วยคำสั่ง parent จากเมนู edit



รูปที่ 7.2 แสดงภาพโมเดล 3 มิติที่สร้างขึ้นจากโปรแกรมมายา เวอร์ชัน 3.0

7.2 การเอ็กซ์พอร์ท (Export) โมเดลเป็นไฟล์ .x

เนื่องจากโมเดลที่ถูกสร้างขึ้นจากโปรแกรมมายาเป็นไฟล์ .bm หรือมายาไบนารี่ไฟล์ (Maya binary file) ซึ่งเป็นรูปแบบที่ยังนำใช้งานในการเขียนโปรแกรมไม่ได้ โดยเราจะต้องทำการเอ็กซ์พอร์ทโมเดลออกมาเป็นไฟล์ .x เสียก่อน แต่เนื่องจากในมายาเวอร์ชัน 3.0 นี้ไม่รองรับกับการเปลี่ยนนามสกุลเป็นไฟล์ .x ตามที่เราต้องการ จึงต้องทำการนำปลั๊กอิน (Plug-in) มาลงในโปรแกรมเสียก่อน จึงจะสามารถเอ็กซ์พอร์ทออกมาได้

<u>หมายเหตุ</u> ปลั๊กอินที่ทำให้โปรแกรมมายา สามารถเอ็กซ์พอร์ทไฟล์ออกมาเป็นไฟล์ .x ดาวน์โหลด (Download) ได้ที่

http://therabbithole.redback.inficad.com/tricks/directx8/Extras_Dirct3D/Tools/Maya30/

7.3 ขั้นตอนการทำงานของโปรแกรม

เมื่อเราได้โมเดลในรูปแบบไฟล์ .x ออกมาแล้ว ข้อมูลภายในนั้นเราสามารถเปิดดูได้ด้วยโน๊ตแพด (Notepad) ซึ่งเป็นข้อมูลของจุดที่เขียนอยู่ในรูปเมตริกซ์หลาย ๆ ส่วนและจะถูกนำเข้าสู่กระบวนการเรนเดอร์

7.3.1 การนำข้อมูลในไฟล์ .x ไปใช้งาน

ตัวอย่างข้อมูลไฟล์ .x ดูได้ที่ภาคผนวก ก. เมตริกซ์ต่าง ๆ ที่ปรากฏจะปิดล้อมด้วยเครื่องหมาย "{ }" หากดูที่ตอนต้นของไฟล์จะพบทรานฟอร์มเมชันเมตริกซ์ (Tramsformation matrix) ของส่วนต่าง ๆ ของโมเดล เช่น ราก (Root), ฐาน (Base), ไหล่ (Shoulder), ข้อศอก (Elbow), พิช (Pitch), เครื่องมือ (Tool) เป็นต้น โดยจะอยู่ในรูปแบบดังตังอย่าง

```
Frame armBase                                   .....(1)
{
FrameTransformMatrix                            .....(2)
{0.000000,1.000000,0.000000,0.000000,-0.997564,0.000000,0.069756,0.000000,0.069756,
-0.000000,0.997564,0.000000,7.500000,4.000000,0.000000,1.000000;;}          .....(3)
```

โดย จาก (1) armBase หมายถึงชื่อเฟรมหรือส่วนประกอบต่าง ๆ ของโมเดล

(2) FrameTransformMatrix คือเมตริกซ์ที่ถูกใช้ในการคำนวณเมื่อมีการเคลื่อนที่ในส่วนของโมเดลนั้น ๆ

(3) ข้อมูลในทรานฟอร์มเมชันเมตริกซ์ อยู่ในรูปแบบของเมตริกซ์ 4×4

ข้อมูลในส่วนลำดับต่อมาจะเป็นข้อมูลที่อยู่ในรูปเมตริกซ์เช่นกันซึ่งมีชื่อต่าง ๆ ได้แก่ Mesh, MeshNormals, MeshTextureCoords และ MeshMaterialList เป็นต้น

ข้อมูลที่นำไปใช้ในกระบวนการเรนเดอร์ภาพมีอยู่ 4 ข้อมูลที่สำคัญด้วยกัน คือ

1). FrameTransformMatrix ทุก ๆ ส่วนของโมเดลหรือเฟรม ถูกเรียกใช้เมื่อส่วนของโมเดลมีการเคลื่อนที่

2). Mesh
3). MeshNormal

} ข้อมูล 2 ตัวนี้จะถูกนำไปเก็บในเวอร์เท็กซ์บัฟเฟอร์ (Vertex Buffer) โดยจะเก็บเป็นอาร์เรย์ (Array) ของข้อมูลที่เป็นโพสิชั่น (Position) และนอมอล (Normal) เป็นชุด ๆไป

4). Index จะถูกนำไปใส่ในอินเด็กซ์บัฟเฟอร์ (Index Buffer) ซึ่งจะเป็นตัวบอกลำดับในการวาดว่าจะเริ่มวาดที่จุดไหนก่อน ข้อมูลอินเด็กซ์เป็นข้อมูลที่สำคัญเพราะหากลำดับในการลงจุดผิดพลาดอาจทำให้โพลีกอนนั้น ๆ มองไม่เห็นไปเลย เพราะในการเรนเดอร์เราจะมองเห็นเฉพาะด้านที่เป็นฟรอนท์เฟซ (Front face) เท่านั้น

ข้อมูลภายในเวอร์เท็กซ์บัฟเฟอร์และอินเด็กซ์บัฟเฟอร์ เป็นข้อมูลที่มีการเซ็ทค่าเพียงครั้งเดียว และจะยังคงค่าข้อมูลเดิมอยู่ตลอดเวลา ไม่ว่าโมเดลจะมีการเปลี่ยนแปลงอย่างไร หลังจากที่เราทำการจัดเรียงข้อมูลลงในเวอร์เท็กซ์บัฟเฟอร์และอินเด็กซ์บัฟเฟอร์แล้วก็จะทำการเซ็ทค่าเวอร์เท็กซ์บัฟเฟอร์ด้วยเมทธอด SetStreamSource เพื่อเป็นการบอกว่าใช้ข้อมูลชุดนี้ในการเรนเดอร์ และใช้เมทธอด SetVertexShader เพื่อเป็นการเซ็ทข้อมูลในการเรนเดอร์และบอกวิธีการเรนเดอร์ ส่วนการเซ็ทอินเด็กซ์บัฟเฟอร์ทำได้ด้วยเมทธอด SetIndices เพื่อบอกว่าจะใช้อินเด็กซ์อันนี้ในการเรนเดอร์

7.3.2 การหาค่าเมตริกซ์เวิลด์ (World matrix)

การหาค่าเมตริกซ์เวิลด์เป็นเมตริกซ์ที่มีการเปลี่ยนแปลงค่า เมื่อส่วนใดส่วนหนึ่งของโมเดลมีการเปลี่ยนแปลง โดยในโปรแกรมเมตริกซ์ที่คูณกันแล้วได้ผลลัพธ์เป็นเวิลด์ ได้แก่

1). เมตริกซ์ m_matAnim เมตริกซ์นี้จะเกิดขึ้นก็ต่อเมื่อมีการเคลื่อนที่ของเฟรม ซึ่งจะทำการหาค่าได้จากเมตริกซ์โรเทชั่น (Rotation matrix) คูณกับเมตริกซ์ทรานส์ฟอร์มเมชั่นของเฟรมนั้น ถ้าหากเฟรมใด ๆ ไม่มีการเคลื่อนที่เมตริกซ์ m_matAnim จะเท่ากับเมตริกซ์ m_matTran

2). เมตริกซ์ทรานส์ฟอร์มหรือ FrameTransformMatrix ของเฟรมในโมเดลที่มีการเคลื่อนที่ โดยจะเก็บอยู่ในตัวแปร m_matTran

3). เมตริกซ์ m_pParent เป็นเมตริกซ์ของเฟรมที่เป็นแพเรนท์ (Parent) ของส่วนที่มีการเคลื่อนที่โดยจะเกิดจากเมตริกซ์ทรานส์เลชันของเฟรมที่เป็นแพเรนท์ทั้งหมดมาคูณกัน

ท้ายที่สุดเมื่อนำเมตริกซ์ทั้ง 3 มาคูณกันก็จะได้เมตริกซ์เวิลด์ (World matrix) ออกมา โดยในโปรแกรมจะใช้ชื่อว่า m_matRender

7.3.3 การเรนเดอร์ภาพ

เมื่อได้ข้อมูลในเมตริกซ์เวิลด์และจัดการเรียงข้อมูลในเวอร์เท็กซ์บัฟเฟอร์เรียบร้อยแล้ว เมตริกซ์เวิลด์ที่ได้จะนำมาคูณกับวิวและโปรเจ็กต์ชันเมตริกซ์ (View and Projection matrix) เพื่อที่จะนำข้อมูลที่ได้ไปคูณกับเวอร์เท็กซ์บัฟเฟอร์ ซึ่งมีการส่งข้อมูลแบบสตรีมดาต้า (Stream data) คือจะส่งข้อมูลทีละค่าออกมาทีละค่า ผลลัพธ์ที่ได้จะเป็นค่าโคออดิเนต ที่จะทำการลงจุดสีบนจอภาพ โดยจะใช้เมทธอด DrawIndexedPrimitive ในการวาดลงบนแบ็คบัฟเฟอร์ (Back buffer) ก่อนแล้วจึงทำการฟลิป (flip) โดยใช้คำสั่ง Present() เพื่อสั่งให้คัดลอกลงในฟร้อนท์บัฟเฟอร์ (Front buffer) เนื่องจากฟร้อนท์บัฟเฟอร์เท่านั้นที่สามารถมองเห็นได้

นอกจากนี้ยังต้องทำการเรียกใช้ฟังก์ชันเพื่อเซ็ทค่าในการเรนเดอร์ ที่สำคัญมีดังนี้

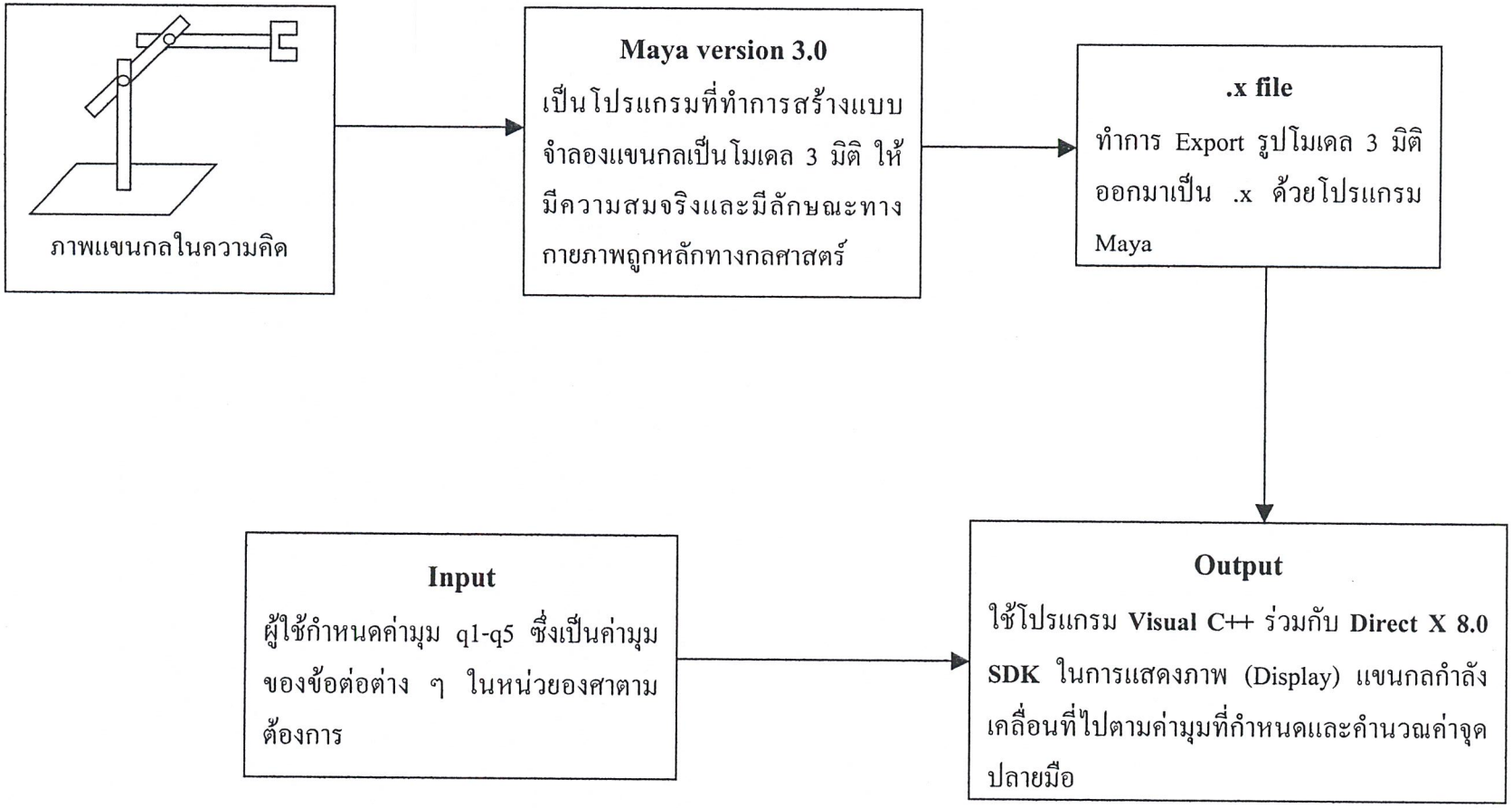
- LightEnable ใช้ในการคำนวณแสงเงา (Shade) เพื่อให้โมเดลมีความสมจริงยิ่งขึ้น
- ในการเรนเดอร์ต้องมีการประกาศ BeginScene และ EndScene เพื่อบ่งบอกว่าจะมีการเรนเดอร์เกิดขึ้น

7.3.4 ค่าเอาท์พุทปลายมือ

เนื่องจากใน DirectX การคำนวณเพื่อเรนเดอร์ภาพกระทำในรูปเมตริกซ์ตำแหน่งและเมตริกซ์ทรานส์ฟอร์มเมชันอยู่แล้ว ดังนั้นตำแหน่งปลายมือที่เราต้องการนำมาแสดงเป็นค่าเอาท์พุทสามารถดึงมาใช้ได้เลย โดยนำค่าข้อมูลในแถวสุดท้าย คอลัมน์ที่ 1 ถึง 3 ของเมตริกซ์ m_matRender หรือเมตริกซ์เวิลด์มาใช้งาน

7.3.5 หลักการทำงานของโปรแกรมโดยรวม

จากที่กล่าวมาทั้งหมดอาจจะยังไม่มองมไม่เห็นภาพรวมการทำงานของโปรแกรมมากนักจึงทำการอธิบายหลักการทำงานของโปรแกรมโดยรวม เพื่อให้ง่ายแก่การทำความเข้าใจสามารถพิจารณาขั้นตอนและกระบวนการคร่าว ๆ ทั้งหมดได้ดังแผนภาพ



รูปที่ 7.3 แสดงแผนภาพการทำงานโดยรวมของโปรแกรม

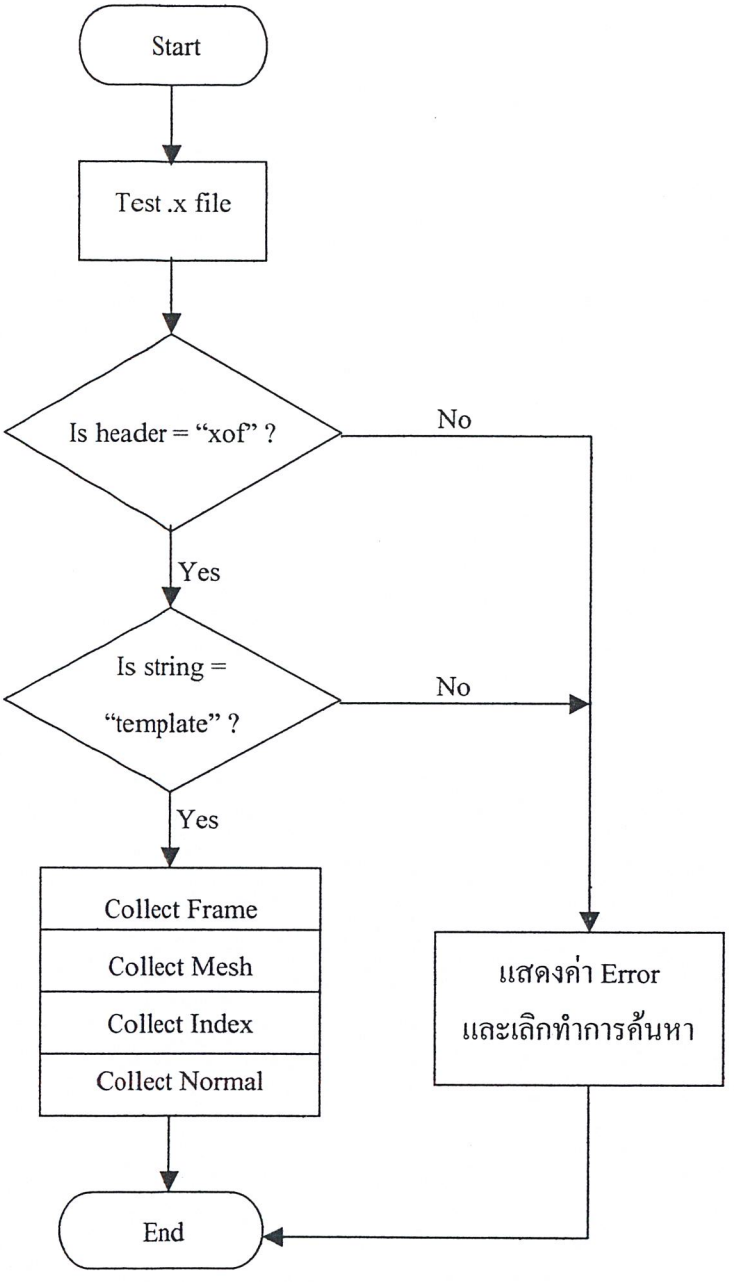
7.3.5 โมดูล (Module) ที่สำคัญในโปรแกรม

เนื่องจากไฟล์และโมดูลในโปรแกรมมีเป็นจำนวนมาก ดังนั้นจึงขออธิบายเฉพาะไฟล์ที่สำคัญเท่านั้น ซึ่งได้แก่

- **Ngfile.cpp** เป็นไฟล์ที่จัดการข้อมูลที่อยู่ในไฟล์ .x ซึ่งจะมีฟังก์ชันย่อยของคลาส NGFILE ในการตรวจสอบไฟล์ ว่ามีไฟล์ที่ต้องการอยู่จริง โดยจะตรวจสอบตรงส่วนที่เป็นเฮดเดอร์ (Header) หากเป็นไฟล์ .x ก็ให้ตรวจสอบสตริง (String) บรรทัดแรกว่าเป็นค่า "xof" หรือไม่ ถ้าใช่ก็ตรวจสอบต่อไปว่ามีสตริงว่าเป็น "template" ถ้าเป็นให้ตัดตรงส่วนนั้นทิ้ง จากนั้นทำการเก็บค่าที่เป็น Frame, Mesh, Index และ normal เอาไว้ในรูปของอาร์เรย์

ฟังก์ชั่นใน MFC ที่สำคัญในไฟล์ Ngfile.cpp และถูกเรียกใช้ได้แก่

fopen	ฟังก์ชันในการเปิดไฟล์
fread	ใช้ในการอ่านข้อมูลเข้ามาจากดาต้าสตรีม (Data Stream)
fwrite	ใช้ในการเขียนข้อมูลออกมาเป็นสตรีม

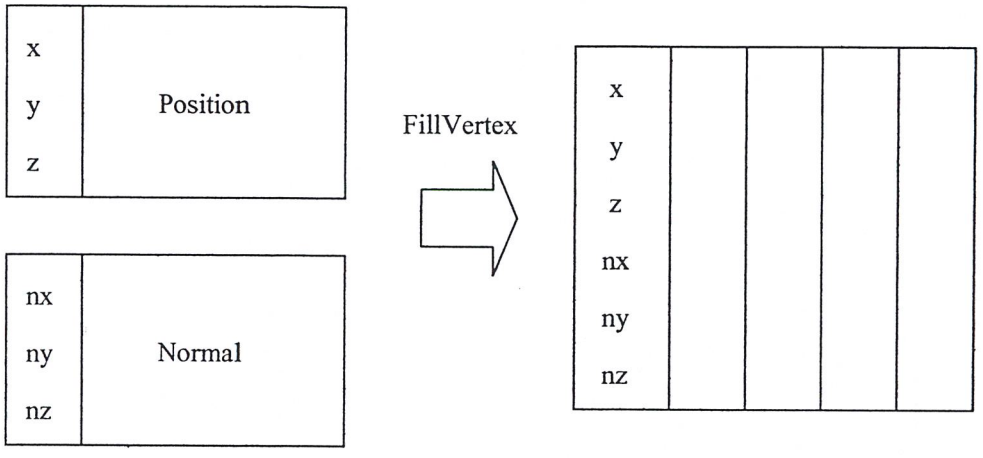


รูปที่ 7.4 แสดงผังการทำงานของโปรแกรมจัดเก็บข้อมูลจากไฟล์ .x (Ngfile.cpp)

- **Ngmodel.cpp** จะมีฟังก์ชันสำคัญที่เขียนขึ้นอยู่ 4 ฟังก์ชัน คือ

1. ฟังก์ชัน Prepare() เป็นฟังก์ชันที่ใช้ตรวจสอบว่าข้อมูล Vertex เป็นชนิดไหน ซึ่งได้แก่ Position และ Normal นอกจากนี้ยังทำการตรวจสอบข้อมูลชนิด Index ด้วย

2. ฟังก์ชัน FillVertex() เป็นฟังก์ชันที่ใช้ในการจัดเรียงข้อมูลให้อยู่ในรูปแบบที่สามารถนำไปใช้งานต่อได้เสียก่อน โดยสามารถอธิบายได้ดังภาพ



รูปที่ 7.5 แสดงการจัดเรียงข้อมูลก่อนที่นำไปเก็บในเวอร์เท็กซ์บัฟเฟอร์

3. ฟังก์ชัน CreateBuffer() เป็นฟังก์ชันที่ใช้ในการสร้างและส่งข้อมูลให้แก่ Vertex buffer รวมทั้ง Index buffer ด้วย

4. ฟังก์ชัน Render() เป็นฟังก์ชันที่ใช้เซ็ทค่าต่าง ๆ ก่อนที่จะเข้าสู่กระบวนการเรนเดอร์หรือเรนเดอร์ไปป์ไลน์

5. ฟังก์ชัน UpdateMatrix () มีการคูณเมตริกซ์ m_matAnim กับเมตริกซ์ทรานส์ฟอร์มเมชันของเฟรมที่เป็นแพเรนท์ ซึ่งจะกระทำการคูณกันเมื่อเฟรมใดเฟรมหนึ่งมีการเคลื่อนที่

เมดธอดใน Direct3D ที่นำมาใช้ในไฟล์ Ngmodel.cpp ได้แก่

- GetDevice() รีเทอร์นค่าดีไวซ์ที่สร้างขึ้นเพื่อนำไปใช้งาน
- GetDeviceCaps() ตรวจสอบความสามารถของดีไวซ์
- CreateVertexShader() กำหนดวิธีการในการเรนเดอร์แก่ฮาร์ดแวร์ เนื่องจากฮาร์ดแวร์สามารถเรนเดอร์ค่าได้หลายแบบ
- CreateVertexBuffer() ทำการสร้างเวอร์เท็กซ์บัฟเฟอร์
- CreateIndexBuffer() ทำการสร้างอินเด็กซ์บัฟเฟอร์

- Lock() และ Unlock() เป็นฟังก์ชันในคลาส CcritSec ซึ่งฟังก์ชัน Lock จะทำการล็อคช่วงของเมมโมรี่หรือจองพื้นที่หน่วยความจำและประกาศพอยน์เตอร์ที่ชี้ไปยังเมมโมรี่ที่เก็บค่าเวอร์เท็กซ์บัฟเฟอร์ จากนั้นก็จะทำการเขียนข้อมูลลงในหน่วยความจำท้ายสุดต้องทำการปลดปล่อยข้อมูลเวอร์เท็กซ์ด้วยคำสั่ง Unlock()
- SetVertexShader เซ็ทข้อมูลเวอร์เท็กซ์ให้พร้อมในการเรนเดอร์
- SetStreamSource เซ็ทว่าเวอร์เท็กซ์บัฟเฟอร์อันไหนที่เราจะใช้
- SetMaterial ส่งค่าให้ดีไวซ์มาทำการเซ็ทค่าแมตทีเรียล (Material)
- SetIndices เซ็ทอินเด็กซ์บัฟเฟอร์ที่จะใช้หรือเซ็ทว่าเป็นข้อมูลอินเด็กซ์ชุดนี้ที่จะใช้
- DrawIndexedPrimitive ทำการวาดลงในแบ็คบัฟเฟอร์โดยใช้ข้อมูลจากอินเด็กซ์บัฟเฟอร์เป็นลำดับในการลงเวอร์เท็กซ์
- SetRenderState เซ็ทรูปแบบการเรนเดอร์ซึ่งสามารถดูรูปแบบการเรนเดอร์ได้จาก D3DRENDERSTATETYPE ตัวอย่างการเรียกใช้เมทธอด SetRenderState

```
ng3d->GetDevice()->SetRenderState(D3DRS_SOFTWAREVERTEXPROCESSING,FALSE);
```

โดย D3DRS_ SOFTWAREVERTEXPROCESSING คือ กระบวนการเรนเดอร์เวอร์เท็กซ์แบบผสม (Mixed) ที่จะทำการรีเซ็ทค่าข้อมูลในอินเด็กซ์และเวอร์เท็กซ์เชดเดอร์เป็นค่าดีฟอลท์ และค่า FALSE คือค่าที่ถูกเซ็ทใหม่ให้แก่ดีไวซ์ของเรนเดอร์เสตท

- SetTransform เป็นการเซ็ททรานฟอร์มเมชันเมตริกซ์ ซึ่งจะมีอยู่ 3 เมตริกซ์ด้วยกัน ได้แก่ เมตริกซ์ World, View และ Projection ตัวอย่างการเซ็ทค่า World matrix

```
ng3d->GetDevice()->SetTransform(D3DTS_WORLDMATRIX(i),&m_Bone.at(i).pBone->
                                m_matRender);
```

ค่าพารามิเตอร์ตัวแรก D3DTS_WORLDMATRIX(i) คือต้องการเซ็ทให้เป็นเมตริกซ์ชนิดใดในที่นี้คือ World matrix ส่วนพารามิเตอร์ตัวที่สองคือพอยน์เตอร์ที่ชี้ไปยัง World matrix

- MutiplyTransform ทำการคูณเมตริกซ์หลักที่ใช้ในกระบวนการเรนเดอร์ทั้ง 3 เมตริกซ์เข้าด้วยกัน คือเมตริกซ์ World, View และ Projection ก่อนที่จะเข้าสู่กระบวนการคลิปปิ้ง (Clipping) ต่อไป

-**NGCAMERA.cpp** จะมีฟังก์ชันสำคัญที่เขียนขึ้นเพียงฟังก์ชันเดียว คือ

ฟังก์ชัน Active() เป็นฟังก์ชันที่ใช้ในการเซ็ทค่าต่าง ๆ ให้แก่ View matrix มีเมทธอดที่สำคัญดังต่อไปนี้

- D3DmatrixLookAtLH เป็นตัวสร้าง Left-hand, look at matrix ตัวอย่างการใช้งาน

```
D3DXMatrixLookAtLH(&m_MatView,&m_vCamera,&m_vTarget,&D3DXVECTOR3
(0.0f,1.0f,0.0f));
```

โดยพารามิเตอร์สำคัญได้แก่ &m_MatView คือพอยเตอร์ที่ชี้ไปยังเมตริกซ์ผลลัพธ์

&m_vCamera คือพอยเตอร์ที่ชี้ไปยังเวกเตอร์ในระบบ 3 แกนที่เก็บค่าตำแหน่งของตาของผู้ดูหรือกล้องในซีนหรือฉาก

&m_vTarget คือพอยเตอร์ที่ชี้ไปยังเวกเตอร์ในระบบ 3 แกนที่เก็บค่าตำแหน่งของเป้าหมายที่กล้องมองไป

ส่วนพารามิเตอร์ตัวสุดท้ายเป็นเวกเตอร์ในระบบ 3 แกน ส่วนใหญ่มักจะตั้งค่าเป็น (0,1,0)

- D3DXMatrixPerspectiveFovLH เป็นตัวสร้าง Left-hand perspective matrix มีตัวอย่างการใช้งานดังนี้

```
D3DXMatrixPerspectiveFovLH(&m_MatProj,m_fFOV,m_fRatio,m_fFront,m_fRear);
```

ค่าพารามิเตอร์สำคัญได้แก่ &m_MatProj คือพอยเตอร์ที่ชี้ไปยังเมตริกซ์ผลลัพธ์

m_fFOV คือ Field of view หน่วยเป็นเรเดียน

m_fRatio คืออัตราส่วนของจอที่ต้องการใช้งาน

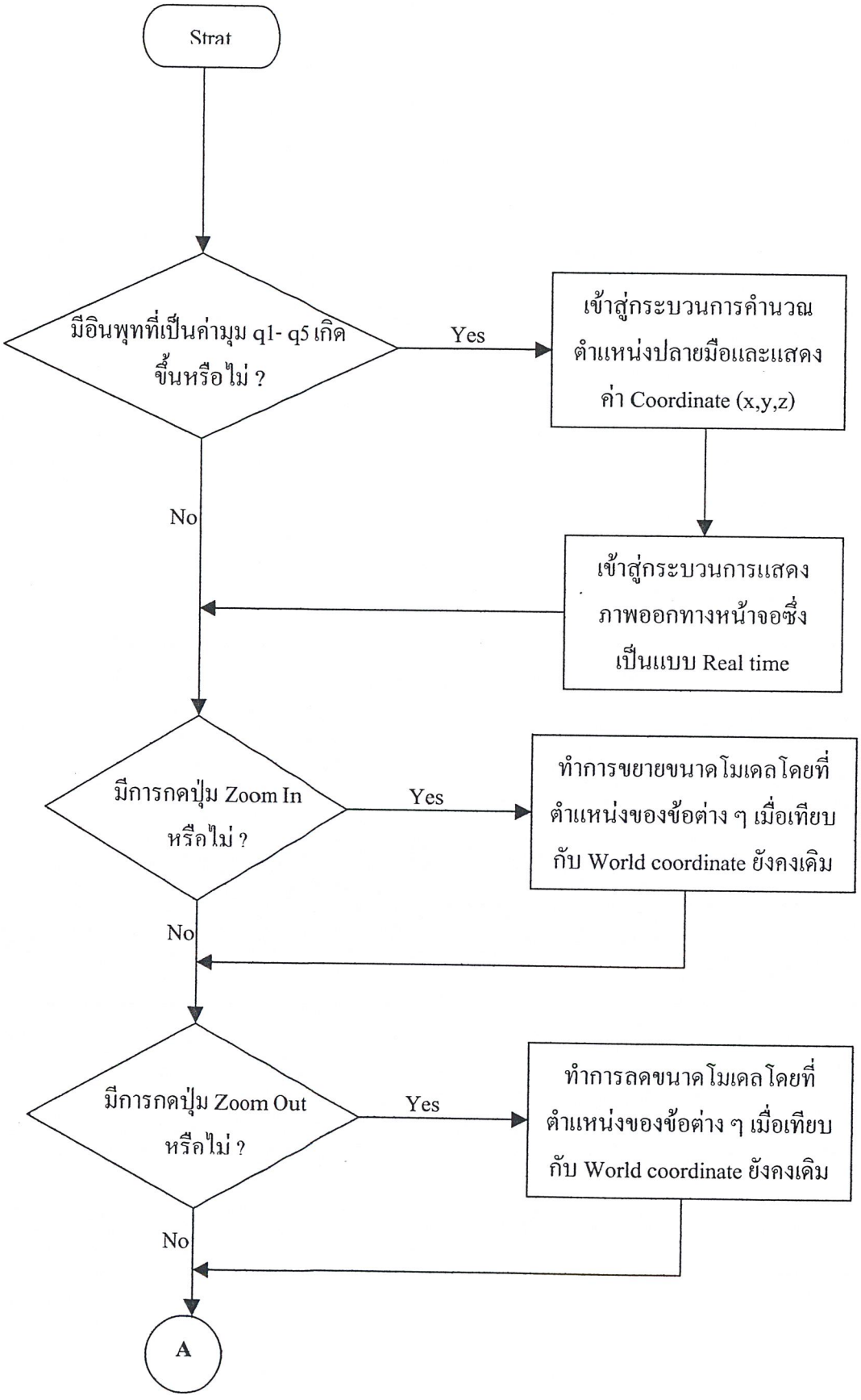
m_fFront คือระยะความลึกของระนาบด้านหน้า

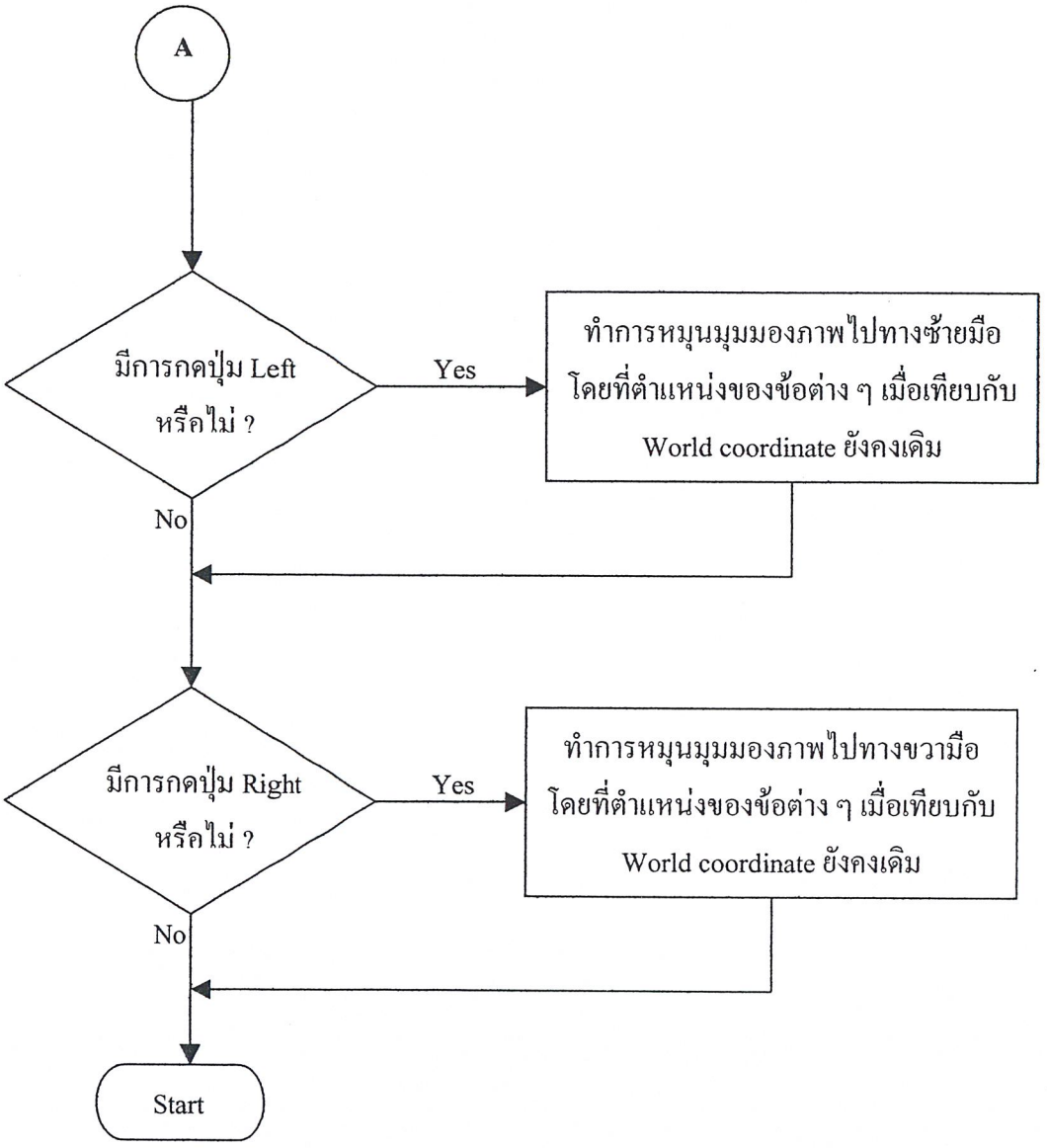
m_fRear คือระยะความลึกของระนาบด้านหลัง

- SetTransform ซึ่งในที่นี้จะเป็นการเซ็ทค่าของ View matrix และ Projection matrix ดังตัวอย่าง

```
m_pDevice->SetTransform(D3DTS_VIEW,&m_MatView);
m_pDevice->SetTransform(D3DTS_PROJECTION,&m_MatProj);
```

-**DlgProp.cpp** เป็นไฟล์ที่ใช้ในการรับข้อมูลเมื่อมีการเลื่อนสไลด์เดอร์บาร์ (Slider bar) หรือเปลี่ยนแปลงตำแหน่งของเฟรมของโมเดล โดยจะมีการ Map message NM_RELEASEDCAPTURE คือเมื่อมีการเปลี่ยนแปลงตำแหน่งของสไลด์เดอร์บาร์หรือค่าใน Edit box ทำให้เกิดเมตริกซ์การหมุน mat ขึ้นแล้วจะนำค่าที่ได้ไปคูณกับ m_matTran หรือเมตริกซ์ทรานส์เลชัน กลายเป็นเมตริกซ์ m_matAnim ขึ้น สามารถเขียนโฟลวชาร์ตแสดงการทำงานของโปรแกรมในส่วนกราฟฟิกส์ยูสเซอร์อินเทอร์เฟส (Graphics User Interface) ร่วมกับฟังก์ชันการคำนวณตำแหน่งปลายมือและฟังก์ชันเรนเดอร์ภาพ รวมทั้งคอนโทรลต่าง ๆ เมื่อมีการแม็ป แมสเสจเกิดขึ้น


Strat
มีอินพุทที่เป็นค่ามุม q1- q5 เกิดขึ้นหรือไม่ ?
Yes
เข้าสู่กระบวนการคำนวณตำแหน่งปลายมือและแสดงค่า Coordinate (x,y,z)
No
เข้าสู่กระบวนการแสดงภาพออกทางหน้าจอซึ่งเป็นแบบ Real time
มีการกดปุ่ม Zoom In หรือไม่ ?
Yes
ทำการขยายขนาดโมเดลโดยที่ตำแหน่งของข้อต่าง ๆ เมื่อเทียบกับ World coordinate ยังคงเดิม
No
มีการกดปุ่ม Zoom Out หรือไม่ ?
Yes
ทำการลดขนาดโมเดลโดยที่ตำแหน่งของข้อต่าง ๆ เมื่อเทียบกับ World coordinate ยังคงเดิม
No
A



รูปที่ 7.6 โฟลวชาร์ตแสดงการทำงานของโปรแกรมในส่วน User Interface ร่วมกับฟังก์ชันอื่นในโปรแกรม

-**Robot3View** จะมีฟังก์ชันสำคัญที่ถูกเรียกใช้ ดังนี้

ฟังก์ชัน Render() เป็นฟังก์ชันย่อยของคลาส CRobot3View ซึ่งมีเมทธอดของไดเร็กท์เอ็กซ์ที่เรียกใช้ ได้แก่

- Clear() เป็นการล้างเซอร์เฟซที่อยู่ในวิวพอร์ท (Viewport)
- BeginScene() เป็นฟังก์ชันที่บ่งบอกว่าจะมีการเรนเดอร์เกิดขึ้นซึ่งจะต้องประกาศไว้ก่อนกระบวนการเรนเดอร์ทั้งหมด

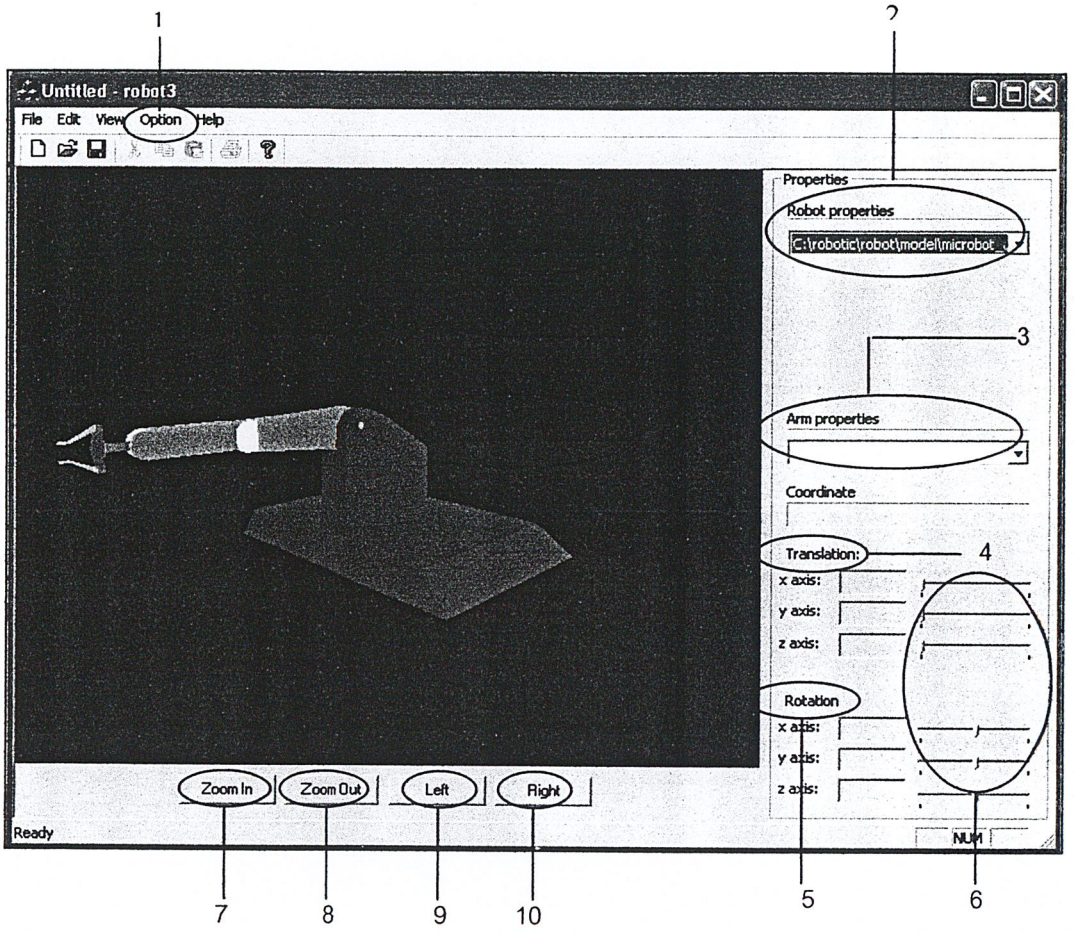
- EndScene() เป็นฟังก์ชันที่บ่งบอกว่าสิ้นสุดการเรนเดอร์แล้ว

ฟังก์ชันสำคัญทางคณิตศาสตร์ของซอฟท์แวร์ DirectX ที่ถูกเรียกใช้ภายในโปรแกรม

D3DXVECTOR3	เป็นการประกาศเวกเตอร์ในระบบ 3 แกน
D3DXVECTOR2	เป็นการประกาศเวกเตอร์ในระบบ 2 มิติ
D3DXMATRIX	เป็นการประกาศเมตริกซ์แบบ 4×4
D3DXMatrixMultiply	เป็นฟังก์ชันที่ใช้ในการคูณเมตริกซ์ด้วยกัน
D3DXMatrixIdentity	เป็นการสร้างเมตริกซ์หนึ่งหน่วย
D3DXMatrixScaling	เป็นการคูณเมตริกซ์ด้วยค่าคงที่ตามแนวแกน x,y,z
D3DXRotationQuaternion	เป็นการสร้างเมตริกซ์จากควอเตอร์เนียน (Quaternion)
D3DxMatrixTranslation	เป็นการสร้างทรานส์เลชันเมตริกซ์คือเมตริกซ์ที่มีการเลื่อนเฉพาะตำแหน่งแต่ไม่มีการหมุนของแกน
D3DXVec3TransformCoord	เป็นการคูณเวกเตอร์ 3 มิติไปกับเมตริกซ์
D3DXMatrixRotationYawPitchRoll	เป็นการสร้างเมตริกซ์จากการกำหนดค่า Yaw, Pitch และ Roll หรือค่า y , x และ z ตามลำดับ

บทที่ 8

อธิบายการใช้งานโปรแกรม Robotic Arm Simulation



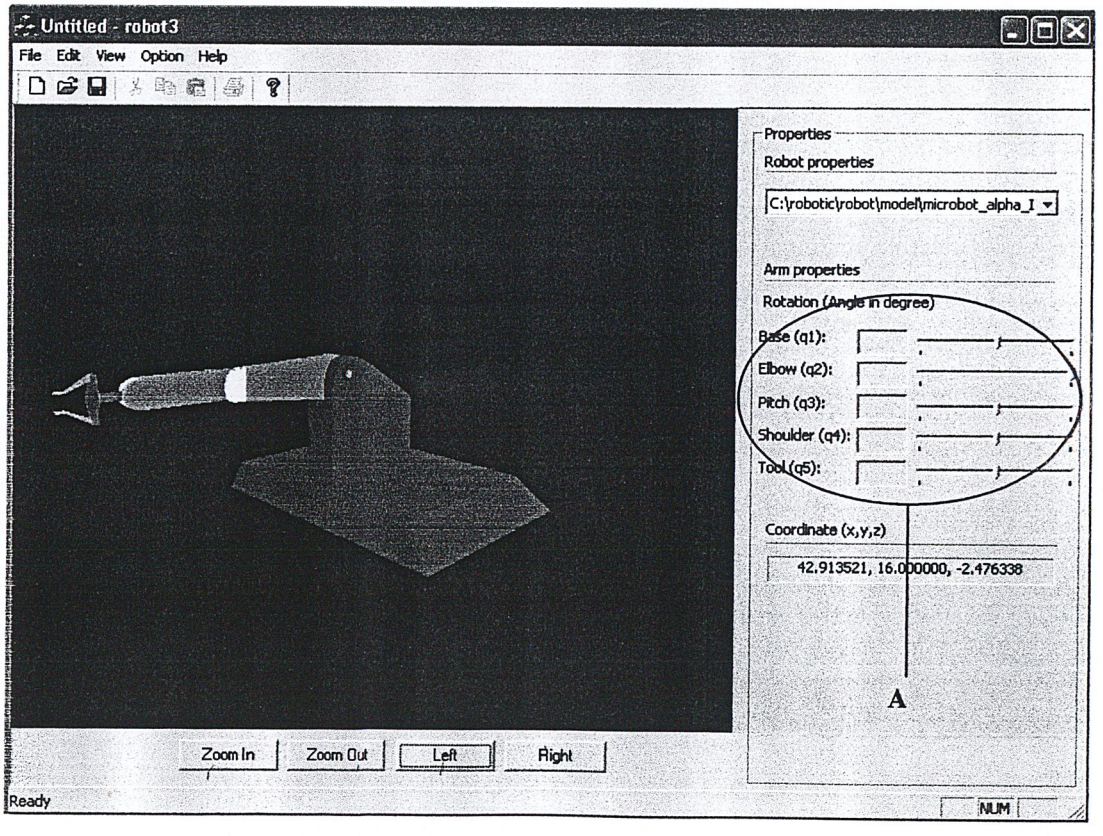
รูปที่ 8.1 ภาพหน้าจอเมื่อมีการรันโปรแกรม

8.1 ส่วนต่าง ๆ ภายในหน้าต่างโปรแกรม Robotic Arm Simulation

1. Option — คลิกเพื่อเลือกว่าต้องการซิมูเลท (Simulate) แขนกลแบบใด โดยเลือกที่เมนูย่อย Add Model
2. Robot properties — บอกชื่อไฟล์ของโมเดลที่ทำการซิมูเลทอยู่
3. Arm Properties — เป็นการเลือกว่าจะมีการสั่งให้แขนกลส่วนใดทำการเลื่อนหรือหมุนตรงโดยคลิกเลือกในคอมโบบอกซ์ (Combo Box)

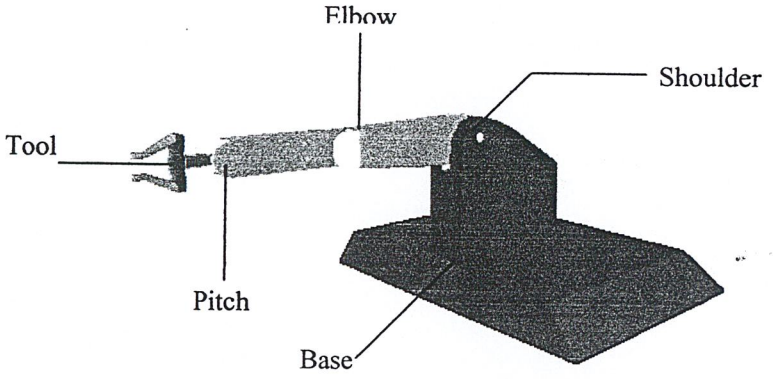
4. Translation	ทำการเลื่อน (Translate) ข้อต่อเลื่อนของแขนกลโดยการป้อนค่าเป็นระยะแนวตามแกน X,Y และ Z
5. Rotate	ทำการหมุน (Rotate) ข้อต่อหมุนของแขนกลโดยการป้อนค่าเป็นมุม(องศา) รอบแกนแกน X,Y และ Z
6. Slider bar	เป็นคอนโทรลที่ใช้ในการปรับค่าอินพุท
7. Zoom In	เป็นการปรับภาพให้ขยายใหญ่ขึ้น
8. Zoom Out	เป็นการย่อขนาดภาพลง
9. Left	เป็นการเลื่อนมุมมองไปทางซ้าย
10. Right	เป็นการเลื่อนมุมมองไปทางขวา

เนื่องจากโมเดลที่สร้างขึ้นมีเพียงรูปแบบเดียวซึ่งเป็นแขนกลแบบข้อต่อหมุน (Articulate) จึงทำการปรับปรุงรูปแบบหน้าต่างใช้งานเพื่อให้สะดวกต่อการใช้งานยิ่งขึ้น เพื่อว่าผู้ที่ไม่มีความรู้ทางด้านหุ่นยนต์หรือแขนกลก็สามารถใช้งานได้



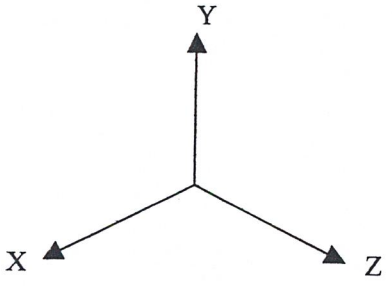
รูปที่ 8.2 ภาพหน้าจอเมื่อมีการปรับเปลี่ยนให้ใช้งานได้ง่ายยิ่งขึ้น

จากภาพจะเห็นว่ามีการปรับเปลี่ยนในส่วนที่ทำการป้อนค่าอินพุทคือส่วน A ซึ่งจะมีสไลเดอร์ บาร์เท่ากับจำนวนข้อของโมเดลแขนกลที่สร้างขึ้น ซึ่งค่าที่รับเข้ามาสามารถป้อนได้โดยใช้สไลเดอร์ บาร์โดยรับค่ามุมเป็นองศามีความละเอียดเป็นทศนิยม 2 ตำแหน่ง และสามารถแสดงชื่อของข้อต่าง ๆ ในโมเดลที่สร้างขึ้น ดังภาพ



รูปที่ 8.3 แสดงชื่อข้อต่อต่าง ๆ ของโมเดล

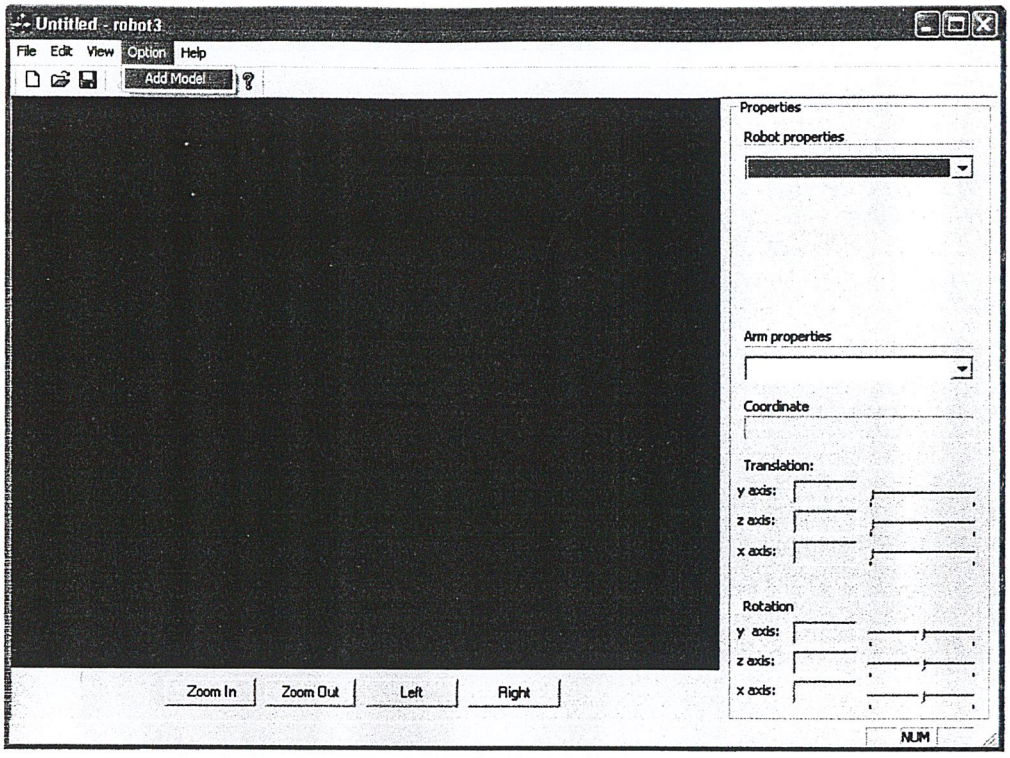
เมื่อทำการเลื่อนสไลเดอร์ บาร์ของข้อต่อใด ข้อต่อนั้นก็จะทำการเลื่อนไปเป็นมุมจำนวนเท่านั้นซึ่งจะเป็นแบบเรียลไทม์ (Real time) โดยแกนหมุนทุก ๆ ข้อและตำแหน่งปลายมือที่เป็นเอาท์พุทสามารถอ้างอิงได้จากเวิลด์โคออดิเนต (World Coordinate) ซึ่งมีพิกัดดังภาพ



รูปที่ 8.4 แสดงเวิลด์โคออดิเนตของซีน (Scene)

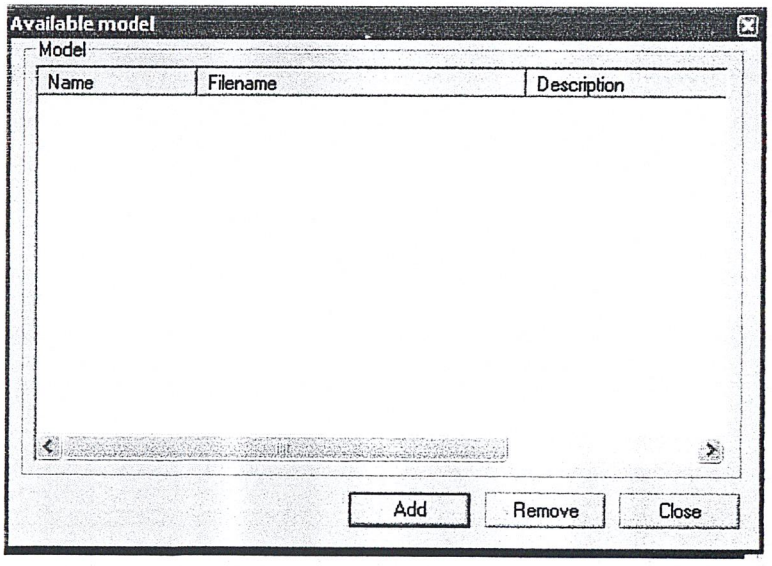
8.2 การใช้งานโปรแกรม

8.2.1 ทำการโหลดโมเดลโดยเลือกเมนู Option แล้วเลือก Add Model ดังภาพ



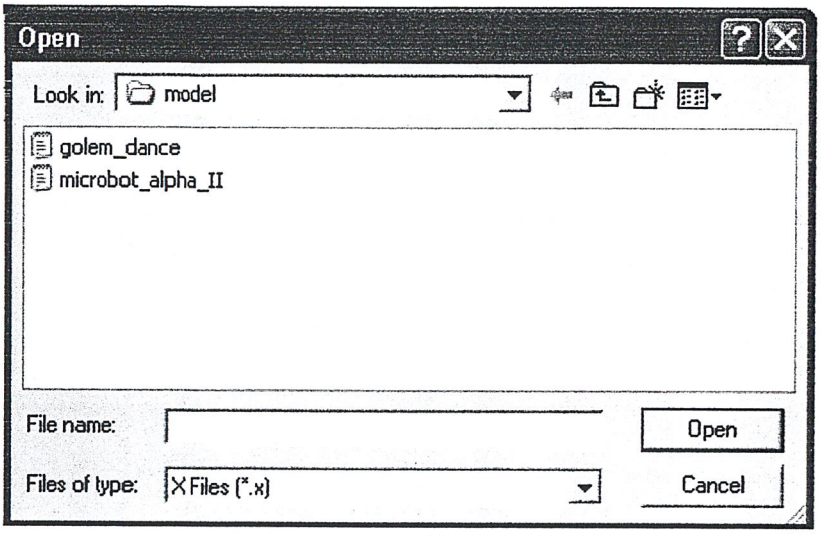
รูปที่ 8.5 ภาพหน้าจอขณะเลือกเมนู Option

8.2.2 เมื่อเลือก Add Model แล้วจะปรากฏไดอะลอกบอกซ์ของลิสต์ (List) ไฟล์ที่สามารถเพิ่มหรือเอาออกได้ ดังภาพ



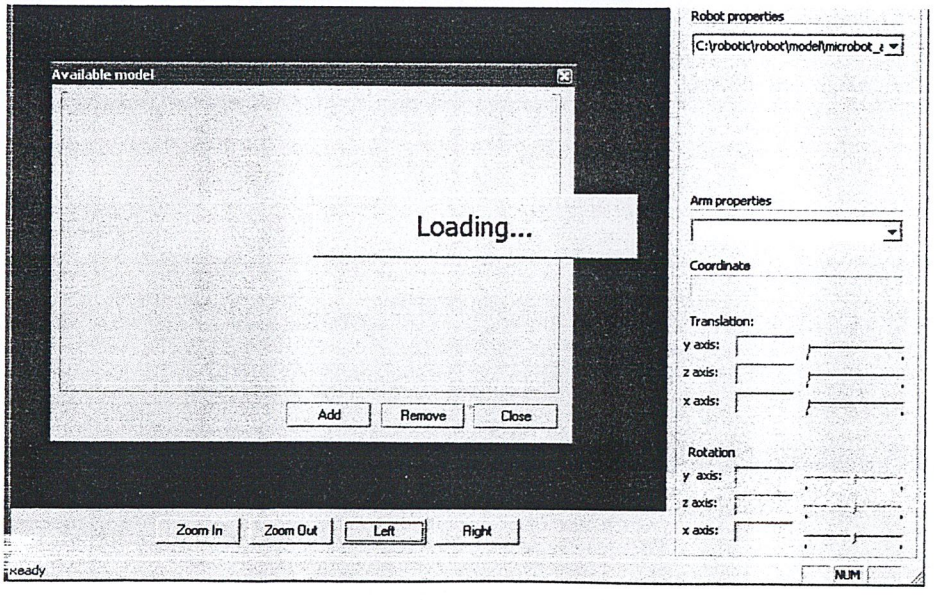
รูปที่ 8.6 แสดงไดอะลอกบอกซ์แสดงไฟล์ที่มีในลิสต์

8.2.3 หากไม่มีชื่อไฟล์ที่ต้องการในลิสต์ เราสามารถคลิกที่ปุ่ม Add แล้วทำการเลือกไฟล์ .x ในไดเร็กทอรี่หรือโฟลเดอร์ที่แสดงอยู่



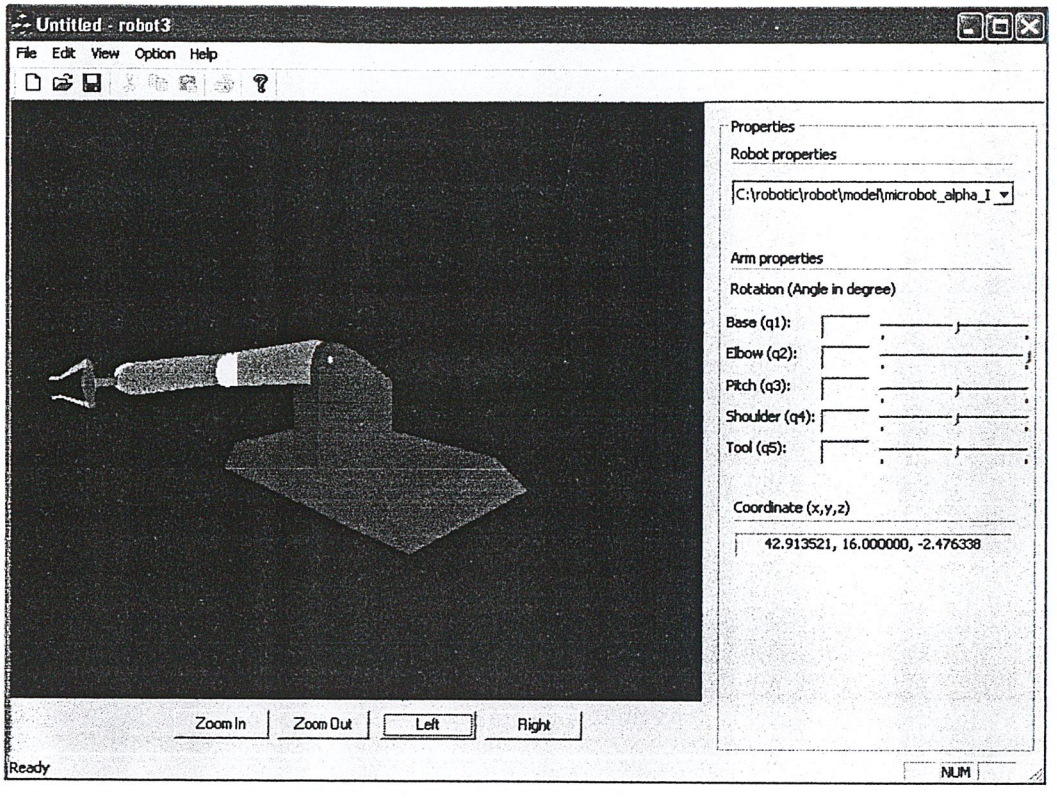
รูปที่ 8.7 แสดงไดอะล็อกบอกซ์ที่แสดงไฟล์ .x ในไดเร็กทอรี่ที่มีไฟล์นามสกุลนี้อยู่

8.2.4 เมื่อทำการเลือกไฟล์แล้วคลิกที่ปุ่ม Open แล้วก็จะปรากฏหน้าจอแสดงการโหลดดังภาพ



รูปที่ 8.8 ภาพแสดงว่าโปรแกรมกำลังทำการโหลดไฟล์ .x ที่ต้องการอยู่

8.2.5 เมื่อทำการโหลดโมเดลขึ้นมาได้แล้ว ก็จะได้หน้าจอดังภาพ



รูปที่ 8.9 แสดงหน้าจอของโปรแกรมเมื่อมีการโหลดโมเดลขึ้นมาเสร็จเรียบร้อยแล้ว

8.2.6 สามารถทำการซิมูเลทโมเดลที่โหลดขึ้นมาได้ด้วยคอนโทรลต่าง ๆ ที่อธิบายไว้แล้วในหัวข้อที่ 8.1

บทที่ 9
บทสรุปผลงานและวิจารณ์

9.1 บทสรุปผลงาน

ผลงานที่ได้จากการทำปริญญานิพนธ์ครั้งนี้เป็นซอฟแวร์ ซิมูเลชั่น (Software Simulation) ซึ่งเป็นการจำลองภาพแขนกล 3 มิติลงบนคอมพิวเตอร์ โดยจะทำการซิมูเลท (Simulate) ดูการเคลื่อนที่ (Movement) ของแขนกลนั้น ๆ ด้วยการป้อนค่าอินพุทเป็นมุมหมุนและแนวแกนที่หมุนของข้อต่าง ๆ ตามต้องการ จากนั้นแขนกลจำลองก็จะเคลื่อนที่ไปตามค่าที่กำหนด และสามารถบอกตำแหน่งของปลายมือแขนกลหรือทูล (Tool) ได้

แขนกล 3 มิติสร้างขึ้นจากโปรแกรมมายาเวอร์ชัน 3.0 (Maya version 3.0) ในที่นี้มีการสร้างแขนกลรูปแบบตายตัว (Fixed) 1 โมเดล ส่วนภาคการคำนวณและการแสดงผลภาพเขียนโปรแกรมโดยใช้ Visual C++ และ Direct X 8.0 SDK

9.2 บทวิจารณ์

จากผลงานโปรแกรมที่ได้จะมีเฉพาะการแก้ปัญหาแบบไดเร็กไคเนมาติกส์ (Direct kinematics) แต่สามารถนำไปพัฒนาต่อให้มีการแก้แก้ปัญหาแบบอินเวอร์สไคเนมาติกส์ (Inverse kinematics) รวมอยู่ในโปรแกรมเดียวกัน ก็จะทำให้แอพลิเคชั่นซิมูเลทแขนกลบนคอมพิวเตอร์มีความสมบูรณ์มากยิ่งขึ้น และโปรแกรมที่เขียนขึ้นสามารถรองรับได้เฉพาะโมเดลแขนกลชนิดเดียว คือ ชนิดแขนกลข้อต่อหมุน โดยสามารถสร้างโมเดลชนิดข้อต่อหมุนเพิ่มขึ้นเพื่อดูการซิมูเลทได้อีกตามต้องการ

ถึงแม้ว่าโปรแกรมจำลองแขนกลบนคอมพิวเตอร์นี้ยังไม่สามารถนำไปใช้งานจริงได้ แต่ก็สามารถนำไปใช้เพื่อการศึกษาสำหรับนักเรียน นักศึกษาได้เป็นอย่างดี แต่หากต้องการนำไปพัฒนาจนถึงขั้นใช้งานจริง ก็สามารถทำได้โดยการปรับปรุงทางด้านโปรแกรมและการเชื่อมต่อระหว่างฮาร์ดแวร์อีกพอสมควร

ภาคผนวก

ตัวอย่างข้อมูลในไฟล์ .x

ตัวอย่างข้อมูลในไฟล์ microbot_alpha_II . x ซึ่งนำมาแสดงเฉพาะข้อมูลเมตริกซ์ชุดแรกเพียงชุดเดียวเท่านั้น

```
xof 0303txt 0032
template XSkinMeshHeader {
 <3cf169ce-ff7c-44ab-93c0-f78f62d172e2>
 WORD nMaxSkinWeightsPerVertex;
 WORD nMaxSkinWeightsPerFace;
 WORD nBones;
}

template VertexDuplicationIndices {
 <b8d65549-d7c9-4995-89cf-53a9a8b031e3>
 DWORD nIndices;
 DWORD nOriginalVertices;
 array DWORD indices[nIndices];
}

template SkinWeights {
 <6f0d123b-bad2-4167-a0d0-80224f25fabb>
 STRING transformNodeName;
 DWORD nWeights;
 array DWORD vertexIndices[nWeights];
 array FLOAT weights[nWeights];
 Matrix4x4 matrixOffset;
}


Frame SCENE_ROOT {


 FrameTransformMatrix {

1.000000,0.000000,0.000000,0.000000,0.000000,1.000000,0.000000,0.000000,0.000000,0.00
0000,1.000000,0.000000,0.000000,0.000000,0.000000,1.000000;;
 }


 Frame armBase {


  FrameTransformMatrix {
   0.000000,1.000000,0.000000,0.000000,-
0.997564,0.000000,0.069756,0.000000,0.069756,-
0.000000,0.997564,0.000000,7.500000,4.000000,0.000000,1.000000;;
  }

  Frame armShoulder {


   FrameTransformMatrix {
    0.000000,-1.000000,0.000000,0.000000,-1.000000,-0.000000,0.000000,0.000000,-
0.000000,-0.000000,-1.000000,0.000000,12.000000,0.000000,0.000000,1.000000;;
   }

   Frame armElbow {


    FrameTransformMatrix {
     1.000000,0.000000,0.000000,0.000000,0.000000,-
1.000000,0.000000,0.000000,0.000000,-0.000000,-
1.000000,0.000000,15.500000,0.000000,0.000000,1.000000;;
    }

    Frame armPitch {


     FrameTransformMatrix {

1.000000,0.000000,0.000000,0.000000,0.000000,1.000000,0.000000,0.000000,0.000000,0.00
0000,1.000000,0.000000,12.000000,0.000000,0.000000,1.000000;;
```

```
    }

    Frame armTool {


     FrameTransformMatrix {

1.000000,0.000000,0.000000,0.000000,0.000000,1.000000,0.000000,0.000000,0.000000,0.00
0000,1.000000,0.000000,3.500000,0.000000,0.000000,1.000000;;
      }

     Frame polySurface13 {


      FrameTransformMatrix {

1.000000,0.000000,0.000000,0.000000,0.000000,1.000000,0.000000,0.000000,0.000000,0.00
0000,1.000000,0.000000,-39.000000,-16.000000,1.000000,1.000000;;
      }

      Mesh {
       86;
       39.000000;18.000000;0.000000;,
       39.000000;14.000000;0.000000;,
       39.500000;14.000000;-0.500000;,
       40.000000;16.000000;-0.500000;,
       39.500000;18.000000;-0.500000;,
       39.000000;18.000000;-2.000000;,
       39.000000;14.000000;-2.000000;,
       39.500000;14.000000;-1.500000;,
       40.000000;16.000000;-1.500000;,
       39.500000;18.000000;-1.500000;,
       39.625000;17.500000;-0.500000;,
       39.625000;17.500000;-1.500000;,
       39.625000;14.500000;-0.500000;,
       39.625000;14.500000;-1.500000;,
       42.000000;15.127005;-0.778997;,
       42.000000;15.345253;-0.778997;,
       42.000000;15.345253;-1.221003;,
       42.000000;15.127005;-1.221003;,
       42.000000;16.654747;-0.778997;,
       42.000000;16.872995;-0.778997;,
       42.000000;16.872995;-1.221003;,
       42.000000;16.654747;-1.221003;,
       42.955639;15.127005;-0.778997;,
       42.955639;15.345253;-0.778997;,
       42.955639;15.345253;-1.221003;,
       42.955639;15.127005;-1.221003;,
       42.955639;16.654747;-0.778997;,
       42.955639;16.872995;-0.778997;,
       42.955639;16.872995;-1.221003;,
       42.955639;16.654747;-1.221003;,
       39.000000;18.000000;0.000000;,
       39.000000;14.000000;0.000000;,
       39.000000;18.000000;-2.000000;,
       39.000000;14.000000;-2.000000;,
       39.000000;14.000000;0.000000;,
       39.500000;14.000000;-0.500000;,
       39.000000;14.000000;-2.000000;,
       39.500000;14.000000;-1.500000;,
       40.000000;16.000000;-0.500000;,
       39.625000;17.500000;-0.500000;,
       40.000000;16.000000;-1.500000;,
       39.625000;17.500000;-1.500000;,
       39.000000;18.000000;0.000000;,
       39.000000;18.000000;-2.000000;,
       39.500000;18.000000;-0.500000;,
       39.500000;18.000000;-1.500000;,
       39.625000;14.500000;-0.500000;,
       40.000000;16.000000;-0.500000;,
       39.625000;14.500000;-1.500000;,
       40.000000;16.000000;-1.500000;,
       39.500000;14.000000;-0.500000;,
       39.625000;14.500000;-0.500000;,
       39.625000;14.500000;-0.500000;,
       39.625000;14.500000;-1.500000;,
       42.000000;15.345253;-0.778997;,
       39.500000;14.000000;-1.500000;,
       39.625000;14.500000;-1.500000;,
       42.000000;15.345253;-1.221003;,
       42.000000;15.127005;-1.221003;,
       42.000000;15.127005;-0.778997;,
```

```
39.500000;18.000000;-0.500000;,
39.625000;17.500000;-0.500000;,
42.000000;16.872995;-0.778997;,
39.500000;18.000000;-1.500000;,
39.625000;17.500000;-1.500000;,
42.000000;16.872995;-1.221003;,
39.625000;17.500000;-1.500000;,
39.625000;17.500000;-0.500000;,
42.000000;16.654747;-1.221003;,
42.000000;16.654747;-0.778997;,
42.955639;15.127005;-0.778997;,
42.955639;15.345253;-0.778997;,
42.955639;15.345253;-0.778997;,
42.955639;15.345253;-1.221003;,
42.955639;15.345253;-1.221003;,
42.955639;15.127005;-1.221003;,
42.955639;15.127005;-1.221003;,
42.955639;15.127005;-0.778997;,
42.955639;16.654747;-0.778997;,
42.955639;16.872995;-0.778997;,
42.955639;16.872995;-0.778997;,
42.955639;16.872995;-1.221003;,
42.955639;16.872995;-1.221003;,
42.955639;16.654747;-1.221003;,
42.955639;16.654747;-1.221003;,
42.955639;16.654747;-0.778997;;
56;
3;5,6,9;,
3;9,6,11;,
3;11,6,8;,
3;8,6,13;,
3;6,7,13;,
3;0,4,1;,
3;4,10,1;,
3;10,3,1;,
3;2,1,12;,
3;3,12,1;,
3;30,31,32;,
3;31,33,32;,
3;34,35,36;,
3;35,37,36;,
3;22,23,25;,
3;23,24,25;,
3;38,39,40;,
3;39,41,40;,
3;42,43,44;,
3;44,43,45;,
3;26,27,29;,
3;27,28,29;,
3;46,47,48;,
3;47,49,48;,
3;50,51,14;,
3;51,15,14;,
3;52,53,54;,
3;53,16,54;,
3;55,17,56;,
3;56,17,57;,
3;37,35,58;,
3;35,59,58;,
3;60,19,61;,
3;61,19,18;,
3;44,45,62;,
3;45,20,62;,
3;63,64,65;,
3;64,21,65;,
3;66,67,68;,
3;67,69,68;,
3;14,15,70;,
3;15,71,70;,
3;54,16,72;,
3;16,73,72;,
3;57,17,74;,
3;17,75,74;,
3;58,59,76;,
3;59,77,76;,
3;18,19,78;,
3;19,79,78;,
3;62,20,80;,
3;20,81,80;,
3;65,21,82;,
3;21,83,82;,
3;68,69,84;,
```

```
3;69,85,84;;

MeshNormals {
 86;
 -0.707107;0.000000;-0.707107;,
 -0.564162;0.039371;-0.824725;,
 -0.696311;0.174078;-0.696311;,
 -0.523860;0.049336;-0.850375;,
 -0.525612;-0.040518;-0.849759;,
 -0.707107;0.000000;0.707107;,
 -0.564162;0.039371;0.824725;,
 -0.696311;0.174078;0.696311;,
 -0.523860;0.049336;0.850375;,
 -0.525612;-0.040518;0.849759;,
 -0.315244;-0.078811;-0.945732;,
 -0.315244;-0.078811;0.945732;,
 -0.696311;0.174078;-0.696311;,
 -0.696311;0.174078;0.696311;,
 -0.080607;0.010412;-0.996692;,
 -0.038949;0.000000;-0.999241;,
 0.113231;-0.993569;0.000000;,
 -0.060466;0.007811;0.998140;,
 -0.058435;0.000000;-0.998291;,
 -0.060466;-0.007811;-0.998140;,
 -0.139762;-0.990185;0.000000;,
 -0.038949;0.000000;0.999241;,
 -1.000000;0.000000;0.000000;,
 -1.000000;0.000000;0.000000;,
 -1.000000;0.000000;0.000000;,
 -1.000000;0.000000;0.000000;,
 -1.000000;0.000000;0.000000;,
 -1.000000;0.000000;0.000000;,
 -1.000000;0.000000;0.000000;,
 -1.000000;0.000000;0.000000;,
 1.000000;0.000000;0.000000;,
 1.000000;0.000000;0.000000;,
 1.000000;0.000000;0.000000;,
 1.000000;0.000000;0.000000;,
 0.000000;1.000000;0.000000;,
 -0.210181;0.977662;0.000000;,
 0.000000;1.000000;0.000000;,
 -0.210181;0.977662;0.000000;,
 -0.970142;-0.242536;0.000000;,
 -0.970142;-0.242536;0.000000;,
 -0.970142;-0.242536;0.000000;,
 -0.970142;-0.242536;0.000000;,
 0.000000;-1.000000;0.000000;,
 0.000000;-1.000000;0.000000;,
 -0.139762;-0.990185;0.000000;,
 -0.279525;-0.960138;0.000000;,
 -0.970142;0.242536;0.000000;,
 -0.970142;0.242536;0.000000;,
 -0.970142;0.242536;0.000000;,
 -0.970142;0.242536;0.000000;,
 -0.124730;0.031182;-0.991701;,
 -0.120716;0.015593;-0.992565;,
 0.335294;-0.942113;0.000000;,
 0.335294;-0.942113;0.000000;,
 0.226462;-0.974020;0.000000;,
 -0.124730;0.031182;0.991701;,
 -0.120716;0.015593;0.992565;,
 -0.058435;0.000000;0.998291;,
 -0.279525;0.960138;0.000000;,
 -0.139762;0.990185;0.000000;,
 -0.124730;-0.031182;-0.991701;,
 -0.120716;-0.015593;-0.992565;,
 -0.279525;-0.960138;0.000000;,
 -0.124730;-0.031182;0.991701;,
 -0.120716;-0.015593;0.992565;,
 -0.080607;-0.010412;0.996692;,
 0.335294;0.942113;0.000000;,
 0.335294;0.942113;0.000000;,
 0.226462;0.974020;0.000000;,
 0.113231;0.993569;0.000000;,
 0.000000;0.000000;-1.000000;,
 0.000000;0.000000;-1.000000;,
 0.000000;-1.000000;0.000000;,
 0.000000;-1.000000;0.000000;,
 0.000000;0.000000;1.000000;,
 0.000000;0.000000;1.000000;,
 0.000000;1.000000;0.000000;,
 0.000000;1.000000;0.000000;,
```

```
 0.000000;0.000000;-1.000000;,
 0.000000;0.000000;-1.000000;,
 0.000000;-1.000000;0.000000;,
 0.000000;-1.000000;0.000000;,
 0.000000;0.000000;1.000000;,
 0.000000;0.000000;1.000000;,
 0.000000;1.000000;0.000000;,
 0.000000;1.000000;0.000000;;
 56;
 3;5,6,9;,
 3;9,6,11;,
 3;11,6,8;,
 3;8,6,13;,
 3;6,7,13;,
 3;0,4,1;,
 3;4,10,1;,
 3;10,3,1;,
 3;2,1,12;,
 3;3,12,1;,
 3;30,31,32;,
 3;31,33,32;,
 3;34,35,36;,
 3;35,37,36;,
 3;22,23,25;,
 3;23,24,25;,
 3;38,39,40;,
 3;39,41,40;,
 3;42,43,44;,
 3;44,43,45;,
 3;26,27,29;,
 3;27,28,29;,
 3;46,47,48;,
 3;47,49,48;,
 3;50,51,14;,
 3;51,15,14;,
 3;52,53,54;,
 3;53,16,54;,
 3;55,17,56;,
 3;56,17,57;,
 3;37,35,58;,
 3;35,59,58;,
 3;60,19,61;,
 3;61,19,18;,
 3;44,45,62;,
 3;45,20,62;,
 3;63,64,65;,
 3;64,21,65;,
 3;66,67,68;,
 3;67,69,68;,
 3;14,15,70;,
 3;15,71,70;,
 3;54,16,72;,
 3;16,73,72;,
 3;57,17,74;,
 3;17,75,74;,
 3;58,59,76;,
 3;59,77,76;,
 3;18,19,78;,
 3;19,79,78;,
 3;62,20,80;,
 3;20,81,80;,
 3;65,21,82;,
 3;21,83,82;,
 3;68,69,84;,
 3;69,85,84;;
}

MeshTextureCoords {
 86;
 0.000000;-0.000000;,
 1.000000;0.000000;,
 1.000000;-0.125000;,
 0.500000;-0.250000;,
 0.000000;-0.125000;,
 0.000000;-0.000000;,
 1.000000;0.000000;,
 1.000000;-0.125000;,
 0.500000;-0.250000;,
 0.000000;-0.125000;,
 0.125000;-0.156250;,
 0.125000;-0.156250;,
 0.875000;-0.156250;,
```

```
 0.875000;-0.156250;,
 1.000000;-0.125000;,
 0.875000;-0.156250;,
 0.875000;-0.156250;,
 1.000000;-0.125000;,
 0.125000;-0.156250;,
 0.000000;-0.125000;,
 0.000000;-0.125000;,
 0.125000;-0.156250;,
 1.000000;-0.125000;,
 0.875000;-0.156250;,
 0.875000;-0.156250;,
 1.000000;-0.125000;,
 0.125000;-0.156250;,
 0.000000;-0.125000;,
 0.000000;-0.125000;,
 0.125000;-0.156250;,
 0.000000;-0.000000;,
 1.000000;0.000000;,
 0.000000;-0.000000;,
 1.000000;0.000000;,
 1.000000;0.000000;,
 1.000000;-0.125000;,
 1.000000;0.000000;,
 1.000000;-0.125000;,
 0.500000;-0.250000;,
 0.125000;-0.156250;,
 0.500000;-0.250000;,
 0.125000;-0.156250;,
 0.000000;-0.000000;,
 0.000000;-0.000000;,
 0.000000;-0.125000;,
 0.000000;-0.125000;,
 0.875000;-0.156250;,
 0.500000;-0.250000;,
 0.875000;-0.156250;,
 0.500000;-0.250000;,
 1.000000;-0.125000;,
 0.875000;-0.156250;,
 0.875000;-0.156250;,
 0.875000;-0.156250;,
 0.875000;-0.156250;,
 1.000000;-0.125000;,
 0.875000;-0.156250;,
 0.875000;-0.156250;,
 1.000000;-0.125000;,
 1.000000;-0.125000;,
 0.000000;-0.125000;,
 0.125000;-0.156250;,
 0.000000;-0.125000;,
 0.000000;-0.125000;,
 0.125000;-0.156250;,
 0.000000;-0.125000;,
 0.125000;-0.156250;,
 0.125000;-0.156250;,
 0.125000;-0.156250;,
 0.125000;-0.156250;,
 1.000000;-0.125000;,
 0.875000;-0.156250;,
 0.875000;-0.156250;,
 0.875000;-0.156250;,
 0.875000;-0.156250;,
 1.000000;-0.125000;,
 1.000000;-0.125000;,
 1.000000;-0.125000;,
 0.125000;-0.156250;,
 0.000000;-0.125000;,
 0.000000;-0.125000;,
 0.000000;-0.125000;,
 0.000000;-0.125000;,
 0.125000;-0.156250;,
 0.125000;-0.156250;,
 0.125000;-0.156250;;
}

MeshMaterialList {
 2;
 56;
 0,
 0,
 0,
 0,
```

```
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 0,
 1,
 1,
 0,
 0,
 0,
 0,
 1,
 1,
 0,
 0,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1,
 1;

 Material lambert3SG {
  0.320000;0.320000;0.320000;1.000000;;
  0.000000;
  0.320000;0.320000;0.320000;;
  0.000000;0.000000;0.000000;;
 }

 Material lambert4SG {
  0.640000;0.640000;0.640000;1.000000;;
  0.000000;
  0.640000;0.640000;0.640000;;
  0.000000;0.000000;0.000000;;
 }
}

VertexDuplicationIndices {
 86;
 30;
 0,
 1,
 2,
 3,
 4,
 5,
 6,
 7,
 8,
 9,
 10,
```

```
      11,
      12,
      13,
      14,
      15,
      16,
      17,
      18,
      19,
      20,
      21,
      22,
      23,
      24,
      25,
      26,
      27,
      28,
      29,
      0,
      1,
      5,
      6,
      1,
      2,
      6,
      7,
      3,
      10,
      8,
      11,
      0,
      5,
      4,
      9,
      12,
      3,
      13,
      8,
      2,
      12,
      12,
      13,
      15,
      7,
      13,
      16,
      17,
      14,
      4,
      10,
      19,
      9,
      11,
      20,
      11,
      10,
      21,
      18,
      22,
      23,
      23,
      24,
      24,
      25,
      25,
      22,
      26,
      27,
      27,
      28,
      28,
      29,
      29,
      26;
    }
  }
}
```

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จลุล่วงได้อย่างดีด้วยคำแนะนำ คำปรึกษาเกี่ยวกับแนวคิดโดยรวมในการเขียนโปรแกรม และแหล่งข้อมูลความรู้เรื่องไดเร็กเอ็กซ์ (DirectX 8.0) ทั้งในรูปแบบหนังสือและแผ่นซีดีจากอาจารย์สุมิตร พนาอุดมทรัพย์ ซึ่งเป็นอาจารย์ที่ปรึกษาในการทำปริญญานิพนธ์ ผู้จัดทำรู้สึกซาบซึ้งในความอนุเคราะห์จากอาจารย์และขอกราบขอบพระคุณเป็นอย่างสูง

ขอขอบพระคุณอาจารย์สุเธียร เกียรติสุนทร ที่เอื้อเฟื้อให้แผ่นโปรแกรมตัวอย่างมาเป็นแนวทางในการทำชิ้นงานครั้งนี้ รวมทั้งขอขอบพระคุณอาจารย์ทุกท่านในภาควิชาวิศวกรรมระบบควบคุมที่เห็นความสำคัญและประโยชน์ของงานชิ้นนี้ จึงได้มีการอนุมัติให้เป็นหัวข้อปริญญานิพนธ์ขึ้น

ขอขอบพระคุณบิดาและมารดาของพวกข้าพเจ้าที่ให้ความช่วยเหลือและเป็นกำลังใจในการเรียนและการจัดทำปริญญานิพนธ์ฉบับนี้ให้ประสบความสำเร็จได้ตลอดรอดฝั่ง

ขอขอบคุณเพื่อน ๆ ภาควิชาระบบควบคุมและภาควิชาคอมพิวเตอร์ที่ให้คำแนะนำและข้อมูลต่าง ๆ อันเป็นประโยชน์ทำให้งานชิ้นนี้สำเร็จราบรื่นไปได้ด้วยดี

ขอขอบคุณรุ่นพี่ที่ช่วยตอบคำถาม ข้อสงสัยและให้คำแนะนำที่สามารถนำมาประยุกต์ใช้ในการทำงานครั้งนี้

คุณค่าและประโยชน์อันพึงมีจากปริญญานิพนธ์ฉบับนี้ ผู้จัดทำขอมอบแด่ผู้มีพระคุณทุกท่าน

นางสาวพรีสา อารีกุล

นายเอกรัตน์ พนมฤทธิ์

เอกสารอ้างอิง

1. ชัยวัฒน์ คำรัตน์, "ก้าวแรกกับไดเร็กเอ็กซ์ 8 SDK ด้วย Microsoft visual C++", บริษัท คอมพิวเตอร์ เอจ เทคโนโลยี จำกัด, หน้า 14-21 และ 114-119, 2544
2. นิรุธ อำนวยศิลป์, "คู่มือการเขียนโปรแกรม Microsoft Visual C++ Version 6.0", 498 หน้า, บริษัท ซัคเซส มีเดีย จำกัด, 2544
3. ยุทธนา ลีลาศวัฒนกุล, "คู่มือการเขียนโปรแกรมและใช้งาน Visual C++ 6.0 ฉบับโปรแกรมเมอร์", อินโฟเพรส , หน้า 3-34 และหน้า 37-175, 2544
4. ชัยวัฒน์ คำรัตน์, "3D Game Programming with DirectX เล่ม 2", บริษัท คอมพิวเตอร์ เอจ เทคโนโลยี จำกัด, หน้า 160-220, 2542
5. Legal Information DirectX 8.0 SDK Programmer's Reference, "Introducing DirectX 8.0"; "DirectX Graphics", © 1995-2000 Microsoft Corporation, 2000