เครื่องเก็บข้อมูลภาพ 3 มิติ

3D SCANNER

โดย

นายจิรพงศ์ บัวเลิศ

นายทวีพล กนกเจิดกุล

นายศิริพันธุ์ ส่งชื่น

อาจารย์ที่ปรึกษา

อาจารย์เทอดศักดิ์ ลิ่วหาทอง

ปริญญานิพนธ์สำหรับปริญญาวิศวกรรมศาสตรบัณฑิต

สาขาวิชาอิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2542

ปริญญานิพนธ์ ปีการศึกษา 2542

ภาควิชา อิเล็กทรอนิกส์

คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง เครื่องเก็บข้อมูลภาพ 3 มิติ

1. นายจิรพงศ์ บัวเลิศ เลขประจำตัว 39014081

2.นายทวีพล กนกเจิดกุล เลขประจำตัว 39014194

3.นายศิริพันธุ์ ส่งชื่น เลขประจำตัว 39014524

ลงชื่อ.....อาจารย์ที่ปรึกษา

(อาจารย์เทอดศักดิ์ ลิ่วหาทอง)

เครื่องเก็บข้อมูลภาพ 3 มิติ

นายจิรพงศ์ บัวเลิศ
นายทวีพล กนกเจิดกุล
นายศิริพันธุ์ ส่งชื่น
อ.เทอดศักดิ์ ลิ่วหาทอง (อาจารย์ที่ปรึกษา)
ปีการศึกษา 2542

บทคัดย่อ

เครื่องเก็บข้อมูลภาพ 3 มิติเป็นเครื่องมือที่สามารถสแกนลักษณะโครงสร้างภายนอกของวัตถุออกมาเป็นภาพโครงสร้าง 3 มิติโดยจะทำการถ่ายภาพของวัตถุในแต่ละมุมจนครบ 1 รอบของวัตถุโดยใช้กล้องวิดีโอและแสงเลเซอร์ในการถ่ายภาพจากนั้นนำภาพทั้งหมดไปประมวลผลภาพเพื่อให้ได้โครงสร้างข้อมูลภาพ 3 มิติแล้วแสดงออกมาที่หน้าจอมอนิเตอร์ ซึ่งโครงสร้างข้อมูลภาพ 3 มิติที่ได้สามารถนำไปประยุกต์ใช้กับงานด้านต่างๆ เช่น ด้านการแพทย์ หรือ งานด้านอุตสาหกรรม

3D SCANNER

Mr. Jirapong Bualert

Mr.Taweepol kanokjerdkul

Mr.Siripan Songchun

Mr.Thurdsak Leauhatong (Advisor)

Abstract

3D SCANNER is an equipment which is able to scan structure of any objects and display to the monitor in 3D-animation figure by taking the picture of that objects in every degree until around it. This equipment requires a video , camera and a laser for capturing the pictures after that all of the picture are processed to 3D-animation figure by using " Image Processing" then they are able to show on the monitor. That image are modified to use with others branches such as medical or industrial works.

สารบัญ

	หน้า
บทคัดย่อ	I
Abstract	II
สารบัญ	III
บทที่ 1 บทนำ	1
บทที่ 2โครงสร้างของระบบโดยรวม	3
2.1 หลักการของระบบโดยรวม	6
บทที่ 3 ทฤษฎีพื้นฐานและหลักการทำงานของสเตปมอเตอร์	8
3.1 คุณลักษณะทั่วไป	8
3.2 หลักการทำงานของสเตปมอเตอร์	9
3.3 ชนิดของสเตปมอเตอร์	12
3.4 การควบคุมสเตปมอเตอร์	19
บทที่ 4 กล้อง Color Quick Cam	21
4.1 สิ่งที่จำเป็นต้องมีในการใช้งานกล้อง Color Quick Cam	21
4.2 คุณสมบัติพิเศษของกล้อง Color Quick Cam	22
4.3 ความสามารถของกล้อง Color Quick Cam	22
บทที่ 5 ชุดทดลองต้นแบบ	23
5.1 โครงสร้างและส่วนประกอบของเครื่องเก็บข้อมูลภาพ 3 มิติ	23
5.2 หลักการทำงาน	23
บทที่ 6 การ์ดควบคุมการหมุนสเตปมอเตอร์	25
6.1 ISA Slot	25
บทที่ 7 คณิตศาสตร์ในการสร้างโครงร่าง 3 มิติ	37
7.1 Two - Dimentional Coordinate Geometry	37
7.2 Three - Dimentional Coordinate Geometry	40
บทที่ 8 โปรแกรมควบคมการทำงานของเครื่องเก็บข้อมูลภาพ 3 มิติ	45
8.1 โครงสร้างการทำงานนของเครื่องเก็บข้อมูลภาพ 3 มิติ	45
8.2 โครงสร้างโปรแกรม	46
8.3 หลักการของการสร้างโครงร่าง 3 มิติ	60

	หน้า
บทที่ 9 ผลการทดลอง	68
9.1 การจัดเตรียมอุปกรณ์ที่ใช้ในการทดลอง	68
9.2 วิธีทดลอง	68
9.3 ผลการทดลอง	68
9.3.1 การทดลองโปรแกรม	68
9.3.2 การทดลองเก็บภาพวัตถุจริง	71
บทที่ 10 สรุปผลการทดลองและวิจารณ์ผลการทดลอง	82
10.1 สรุปผลการทดลองและวิจารณ์ผลการทดลอง	82
10.2 สิ่งที่ดำเนินการในภาคเรียนที่ 1	83
10.3 สิ่งที่ดำเนินการในภาคเรียนที่ 2	83
10.4 ข้อจำกัดของโครงงาน	83
10.5 แนวทางในการพัฒนาและปรับปรุงโครงงาน	84
10.5.1การพัฒนาฮาร์ดแวร์	84
10.5.2 การพัฒนาซอฟแวร์	85
ภาคผนวก	86
กิตติกรรมประกาศ	125
เอกสารอ้างอิง	126

สารบัญรูป

หน้า

บทที่ 1

รูปที่1.1 แสดงหุ่นจำลองที่ได้จากการสร้างจากเทคโนโลยี-ภาพกราฟฟิก 3 มิติ 1

บทที่ 2

รูปที่ 2.1 แสดงแสงเลเซอร์ที่ถูกส่งจากแหล่งกำเนิดไปสู่วัตถุมองจากด้านข้าง 4

รูปที่ 2.2 แสดงตำแหน่งของกล้อง ชุดกำเนิดเลเซอร์ วัตถุมองจากด้านบน 4

รูปที่ 2.3 แสดงตำแหน่งของกล้อง และชุดกำเนิดเลเซอร์ ในการใช้งานจริง 4

รูปที่ 2.4 แสดงภาพที่กล้องรับมาเมื่อมีการยิงเลเซอร์ 5

รูปที่ 2.5 แสดงโครงสร้างของระบบโดยรวม 5

รูปที่ 2.6 แสดงหลักการทำงานของระบบโดยรวม 6

บทที่ 3

รูปที่ 3.1 แสดงโครงสร้างของไฮบริดจ์สเตปมอเตอร์ที่มีจำนวนสเตปต่อรอบเท่ากับ 12 9

รูปที่ 3.2 แสดงขั้นตอนการทำงานของสเตปมอเตอร์แบบเต็มสเตป 1 เฟส 10

รูปที่ 3.3 แสดงขั้นตอนการทำงานของสเตปมอเตอร์แบบเต็มสเตป 2 เฟส 11

รูปที่ 3.4 แสดงขั้นตอนการทำงานของสเตปมอเตอร์แบบครึ่งสเตป 11

รูปที่ 3.5 แสดงภาพตัดขวางของสเตปมอเตอร์แบบ 3 เฟส 12

รูปที่ 3.6 แสดงตำแหน่งสมดุลย์เมื่อเฟสใดเฟสหนึ่งของสเตปถูกกระตุ้น 13

รูปที่ 3.7 แสดงแรงภายนอกที่มีผลต่อเส้นแรงแม่เหล็ก 13

รูปที่ 3.8 แสดงขั้นตอนการเคลื่อนที่ของโรเตอร์เมื่อสเตปมอเตอร์ถูกกระตุ้น 13

รูปที่ 3.9 แสดงขั้นตอนการเคลื่อนที่ของสเตปมอเตอร์ 14

รูปที่ 3.10 แสดงการเปรียบเทียบเส้นแรงแม่เหล็กระหว่างช่องว่างที่กว้างและแคบ 15

รูปที่ 3.11 แสดงมอเตอร์ 3 เฟส และ 4 เฟส 15

รูปที่ 3.12 แสดงโครงสร้างของสเตปมอเตอร์แบบแม่เหล็กถาวร 16

รูปที่ 3.13 แสดงการทำงานของสเตปมอเตอร์แบบแม่เหล็กถาวรขนาด 4 เฟส 17

รูปที่ 3.14 แสดงโครงสร้างของไฮบริดจ์สเตปมอเตอร์ 18

หน้า

บทที่ 4

รูปที่ 4.1 แสดงลักษณะภายนอกของกล้อง Color Quick Cam 21

รูปที่ 4.2 แสดงลักษณะภายในของกล้อง Color Quick Cam 21

บทที่ 5

รูปที่ 5.1 โครงสร้างและส่วนประกอบโดยรวมของชุดทดลองต้นแบบ(ด้านหน้า) 23

รูปที่ 5.2 โครงสร้างและส่วนประกอบโดยรวมของชุดทดลองต้นแบบ(ด้านหลัง) 24

บทที่ 6

รูปที่ 6.1 แสดงการนับขาของ Slot แบบ 62 ขา 25

รูปที่ 6.2 แสดงการนับขาของ Slot แบบ 36 ขา 26

รูปที่ 6.3 สัญญาณ Enable Data Buffer 31

รูปที่ 6.4 รูปแสดงบล็อกไดอะแกรมการทำงานของวงจรพอร์ตขนาน 33

รูปที่ 6.5 แสดงวงจรการ์ดพอร์ตขนาน 34

รูปที่ 6.6 รูปแสดงบล็อกไดอะแกรมการทำงานของวงจรขับสเตปมอเตอร์ 35

รูปที่ 6.7 แสดงวงจรขับสเตปมอเตอร์ 36

บทที่ 8

รูปที่ 8.1 บล็อกไดอะแกรมการทำงานของเครื่องเก็บข้อมูลภาพ 3 มิติ 45

รูปที่ 8.2 แสดงอัลกอริทึมของการทำ Thinning 49

รูปที่ 8.3 แผนภาพ แสดงโครงสร้างโดยรวมของโปรแกรม 52

รูปที่ 8.4 แผนภาพ แสดงการรับภาพและข้อมูลภาพจากกล้องวีดีโอ 53

รูปที่ 8.5 แผนภาพ แสดงการสร้าง Bitmap file จากภาพที่ Capture 54

รูปที่ 8.6 แผนภาพ แสดงการแปลง (Convert) ภาพจาก Color bmp
เป็น Brightness bmp 55

รูปที่ 8.7 แผนภาพ แสดงขั้นตอนการทำ Threshold 56

รูปที่ 8.8 แผนภาพ แสดงขั้นตอนการทำ Thinning 57

รูปที่ 8.9 แผนภาพ แสดงขั้นตอนการเก็บพิกัดของข้อมูลภาพจาก
Thinning bmp 58

รูปที่ 8.10 แผนภาพ แสดงการวาด (Plot) ภาพบนแกน 3 มิติ 59

รูปที่ 8.11 แสดงการย้ายจุดกำเนิดของภาพ 2 มิติ 60

หน้า

รูปที่ 8.12 แสดงตำแหน่งของ Z_E, Z_C และระนาบของภาพบนแกน 3 มิติ 62
รูปที่ 8.13 แสดงการวางตัวของระนาบกล้องและระนาบเลเซอร์ 63
รูปที่ 8.14 แสดงระนาบ xy ที่จุดกำเนิด 64
รูปที่ 8.15 แสดงการหมุนจุดบนระนาบรอบแกน y 66

บทที่ 9

รูปที่ 9.1 แสดงโครงสร้างของเครื่องเก็บข้อมูลภาพ 3 มิติและวัตถุที่ได้ทำการทดลอง 69
รูปที่ 9.2 แสดงภาพที่สร้างขึ้นเอง เพื่อนำมาทดสอบกับโปรแกรมในทุกๆสเตป 69
รูปที่ 9.3 แสดงภาพที่ได้จากการทำ Threshold 69
รูปที่ 9.4 แสดงภาพที่ได้จากการทำ Thinning 69
รูปที่ 9.5 แสดงภาพโครงร่าง 3 มิติที่ได้จากการประมวลภาพ
ที่สร้างขึ้นในสเตปที่ 1 70
รูปที่ 9.6 แสดงภาพโครงร่าง 3 มิติที่ได้จากการประมวลภาพ
ที่สร้างขึ้นในสเตปที่ 2 70
รูปที่ 9.7 แสดงภาพโครงร่าง 3 มิติที่ได้จากการประมวลภาพ
ที่สร้างขึ้นในสเตปที่ 5 70
รูปที่ 9.8 แสดงภาพโครงร่าง 3 มิติที่ได้จากการประมวลภาพ
ที่สร้างขึ้นในสเตปที่ 15 70
รูปที่ 9.9 แสดงภาพโครงร่าง 3 มิติที่ได้จากการประมวลภาพ
ที่สร้างขึ้นในสเตปที่ 30 71
รูปที่ 9.10 แสดงภาพโครงร่าง 3 มิติที่ได้จากการประมวลภาพ
ที่สร้างขึ้นในสเตปที่ 40 71
รูปที่ 9.11 แสดงภาพโครงร่าง 3 มิติที่ได้จากการประมวลภาพ
ที่สร้างขึ้นในสเตปที่ 50 71
รูปที่ 9.12 แสดงภาพที่ได้จากการประมวลผลของภาพที่รับได้จากกล้องสเตปที่ 1 72
รูปที่ 9.13 แสดงภาพที่ได้จากการประมวลผลของภาพที่รับได้จากกล้องสเตปที่ 2 73
รูปที่ 9.14 แสดงภาพที่ได้จากการประมวลผลของภาพที่รับได้จากกล้องสเตปที่ 5 74
รูปที่ 9.15 แสดงภาพที่ได้จากการประมวลผลของภาพที่รับได้จากกล้องสเตปที่ 10 75
รูปที่ 9.16 แสดงภาพที่ได้จากการประมวลผลของภาพที่รับได้จากกล้องสเตปที่ 17 76
รูปที่ 9.17 แสดงภาพที่ได้จากการประมวลผลของภาพที่รับได้จากกล้องสเตปที่ 30 77

หน้า

รูปที่ 9.18 แสดงภาพที่ได้จากการประมวลผลของภาพที่รับได้จากกล้องสเตปที่ 40 78
รูปที่ 9.19 แสดงภาพที่ได้จากการประมวลผลของภาพที่รับได้จากกล้องสเตปที่ 50 79
รูปที่ 9.20 แสดงการเปรียบเทียบภาพวัตถุจริงกับภาพจากการสแกนผลไม้ 80
รูปที่ 9.21 แสดงการเปรียบเทียบภาพวัตถุจริงกับภาพจากการสแกนถ้วยกาแฟ 80
รูปที่ 9.22 แสดงการเปรียบเทียบภาพวัตถุจริงกับภาพจากการสแกนขวดพลาสติก 80
รูปที่ 9.23 แสดงการเปรียบเทียบภาพวัตถุจริงกับภาพจากการสแกนชาม 81

บทที่ 10

รูปที่ 10.1 แสดงการเปรียบเทียบภาพวัตถุจริงกับภาพจากการสแกน 82

สารบัญตาราง

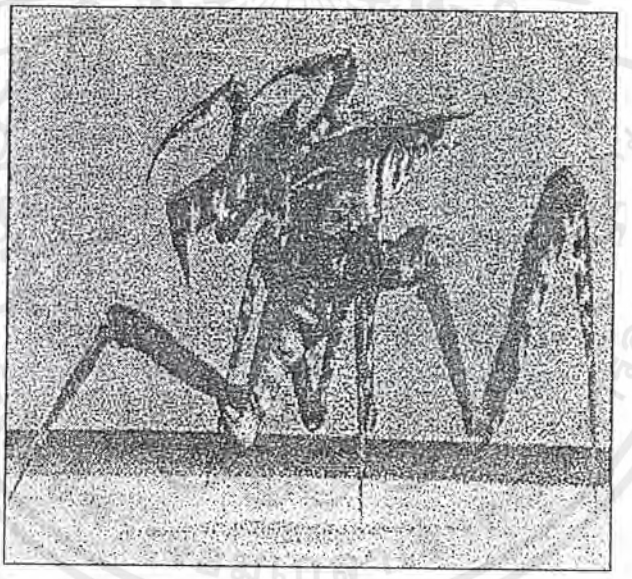
หน้า

บทที่ 6

ตารางที่ 6.1 แสดงชื่อของสัญญาณขาต่างๆ ของ Slot 27

บทที่ 1
บทนำ

เครื่องเก็บข้อมูลภาพ 3 มิติ เป็นโครงงานซึ่งสร้างขึ้นมาจากแนวความคิดของเทคโนโลยีการสร้างภาพ 3 มิติ จากลักษณะโครงสร้างภายนอกของวัตถุ ซึ่งเทคโนโลยีนี้เริ่มเป็นที่แพร่หลายมากขึ้น ซึ่งจะเห็นได้ใกล้ตัว เช่น ภาพยนตร์จากต่างประเทศที่เข้ามาฉายในประเทศไทย ไม่ว่าจะเป็น The Lost World ,Toy Story 2 ,Bug 's Life และอีกหลายเรื่องล้วนแต่ใช้เทคโนโลยีการสร้างภาพ 3 มิติทั้งสิ้น ซึ่งกว่าจะมาเป็นภาพ 3 มิติได้นั้นต้องผ่านขั้นตอนที่ยุ่งยากซับซ้อนมากคือ



รูปที่ 1.1 แสดงหุ่นจำลองที่ได้จากการสร้างจากเทคโนโลยีภาพกราฟฟิก 3 มิติ

- ขั้นตอนแรก ก็คือ ต้องทำโครงสร้างคร่าวๆ ของรูปร่างของสิ่งที่เราต้องการก่อน
- ขั้นตอนที่สอง ซึ่งเป็นขั้นตอนที่เกี่ยวข้องโดยตรงกับโครงงานนี้ คือ การใช้สแกนเนอร์สแกนไปตามโครงสร้างนั้น ซึ่งสแกนเนอร์ที่ใช้จะมีประสิทธิภาพค่อนข้างสูงมาก สามารถรู้ว่าจุดหักมุมมีขนาดกี่องศา แล้วนำข้อมูลที่สแกนได้เหล่านั้นมาประมวลผลเป็นภาพโครงสร้าง 3 มิติออกมาได้

-ขั้นตอนต่อไปก็ใช้เทคนิคทางด้านกราฟฟิกแต่งภาพโครงสร้างนั้นตามจินตนาการ แล้วนำภาพที่แต่งแล้วนั้นไปสร้างเป็นเป็นรูปทรงออกมา

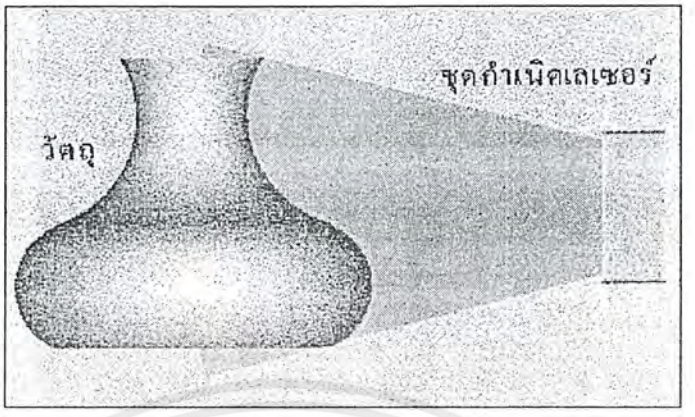
จากขั้นตอนการทำงานที่กล่าวถึงข้างต้น เป็นการใช้ประโยชน์โดยการนำภาพที่ได้จากการประมวลผลโดยเครื่องเก็บข้อมูล 3 มิติ มาสร้างภาพยนตร์ แต่ประโยชน์ของเทคโนโลยีนี้ไม่ได้มีเพียงเท่านี้ โดยสามารถนำเทคโนโลยีนี้มาใช้ประโยชน์ทางด้านการแพทย์ได้ด้วย เช่น การนำภาพ MRI (Magnetic Resonance Image) ที่ได้จากการสแกนสมองมาใช้ในการสร้างภาพ 3 มิติ เพื่อหาตำแหน่งของการผ่าตัดที่จะกระทบกระเทือนต่อเนื้อเยื่อสมองให้น้อยที่สุด

ที่กล่าวมาเป็นเพียงประโยชน์ส่วนหนึ่งเท่านั้น ยังมีเทคโนโลยีทางด้านอื่นอีกมากมายที่เอาหลักการนี้มาใช้ โดยโครงงานนี้ก็ทำขึ้นเพื่อเป็นจุดเริ่มต้นของแนวความคิดเหล่านั้น

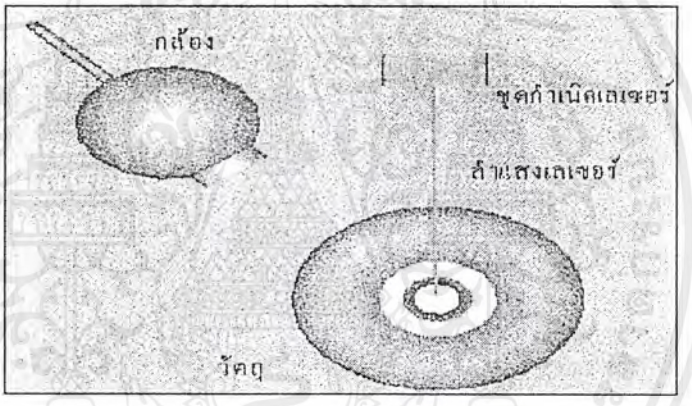
บทที่ 2
โครงสร้างของระบบโดยรวม

เครื่องเก็บข้อมูลภาพ 3 มิติ (3D-SCANNER) เป็นเครื่องมือที่ใช้ในการสแกนลักษณะโครงสร้างภายนอกของวัตถุที่เรามองเห็นกันอยู่ออกมาเป็นรูปโครงสร้าง 3 มิติแล้วแสดงออกมาทางหน้าจอมอนิเตอร์ ลักษณะการทำงานของ เครื่องเก็บข้อมูลภาพ 3 มิติ นี้ คือ การถ่ายแต่ละมุมของวัตถุโดยกล้องวิดีโอแล้วนำข้อมูลที่ได้มาทำการประมวลผลภาพ ตำแหน่งของกล้องวิดีโอจะถูกควบคุมโดยสเตปมอเตอร์ ซึ่งการควบคุมสเตปปิ้งนั้นจะถูกควบคุมผ่านทางการ์ดพอร์ตขนานที่ต่อกับISA Slot ของเครื่องคอมพิวเตอร์ด้วย โดยวงจรของการ์ด พอร์ตขนานที่ต่อเพื่อควบคุมการทำงานของสเตปมอเตอร์นั้น ได้ถูกต่อแยกอิสระจากวงจรขับสเตปมอเตอร์โดยใช้ ออปโตไอโซเลเตอร์ (Opto-Isolator) เพื่อไม่ให้เกิดการรบกวนกันและป้องกันความเสียหายที่อาจเกิดขึ้นกับเครื่องคอมพิวเตอร์ได้

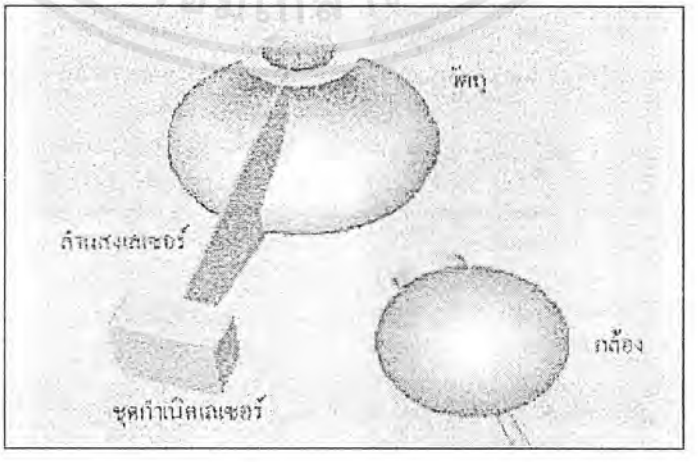
ส่วนประกอบของ เครื่องเก็บข้อมูลภาพ 3 มิติ ที่สำคัญ ได้แก่ แท่นวางวัตถุสำหรับสแกน แขนยึดกล้องวิดีโอ แขนยึดเลเซอร์ การ์ดพอร์ตขนาน สเตปมอเตอร์และวงจรขับสเตปมอเตอร์ ซึ่งรายละเอียดและการออกแบบในแต่ละส่วนจะแสดงไว้ในบทต่อๆไป



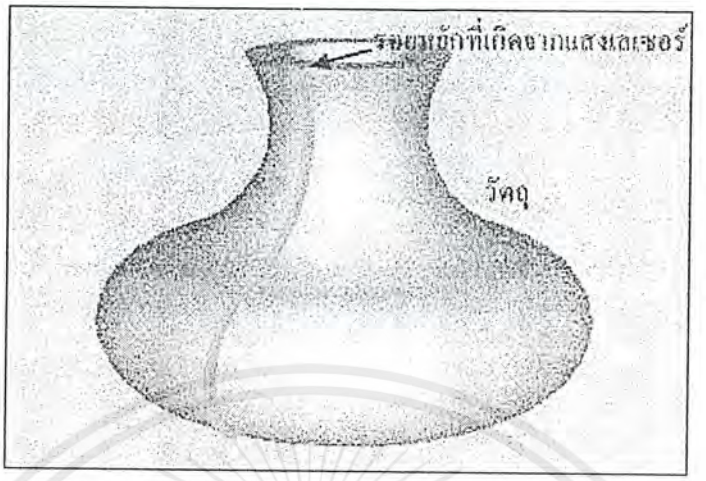
รูปที่ 2.1 แสดงแสงเลเซอร์ที่ถูกส่งจากแหล่งกำเนิดไปสู่วัตถุมองจากด้านข้าง



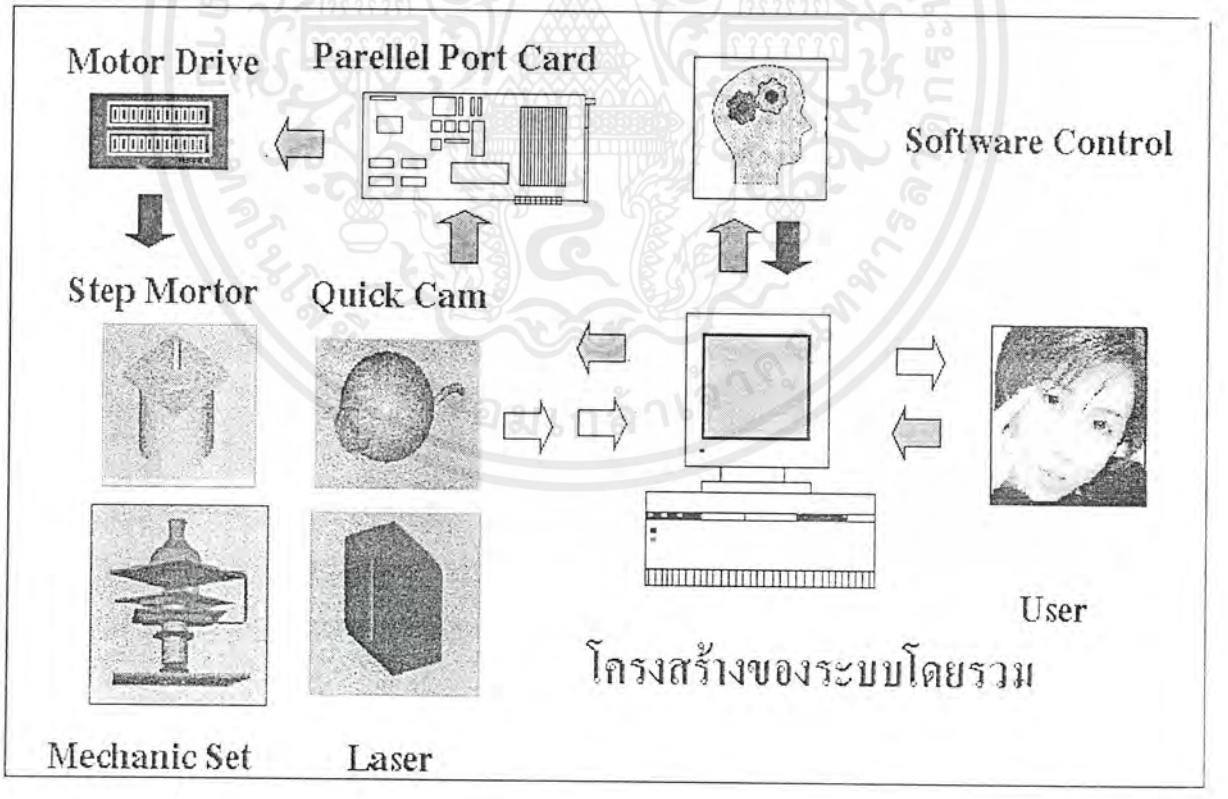
รูปที่ 2.2 แสดงตำแหน่งของกล้อง ชุดกำเนิดเลเซอร์ และวัตถุมองจากด้านบน



รูปที่ 2.3 แสดงตำแหน่งของเลเซอร์และกล้องในการใช้งานจริง

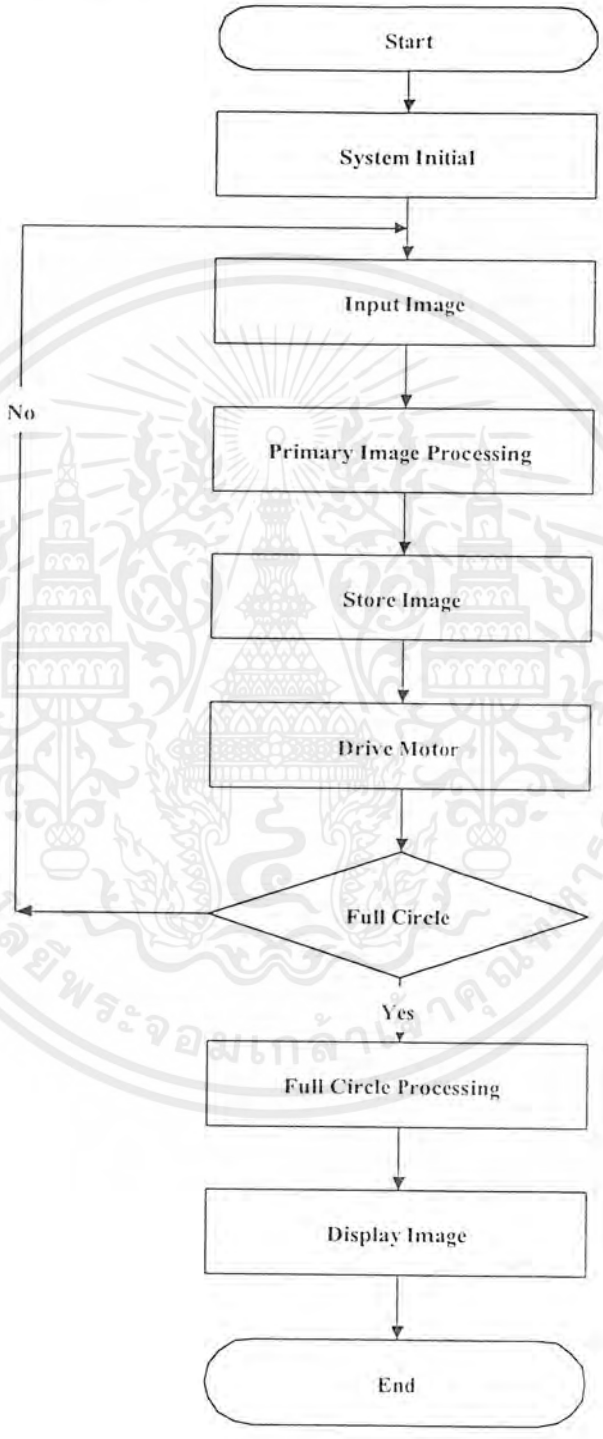


รูปที่ 2.4 แสดงภาพที่กล้องรับมาเมื่อมีการยิงเลเซอร์



รูปที่ 2.5 แสดงโครงสร้างของระบบโดยรวม

2.1 หลักการของระบบโดยรวม



รูปที่ 2.6 แสดงหลักการทำงานของระบบโดยรวม

1) System Initial

เป็นขั้นตอนทางซอฟแวร์ที่ทำหน้าที่ติดต่อกับผู้ใช้งานโดยสามารถทำงานบน วินโดว์ (Window) ได้โดยจะให้ผู้ใช้กำหนดค่าตัวแปรต่างๆ ของระบบดังนี้

- ความละเอียดของภาพ
- รูปแบบในการจัดเก็บภาพ
- การแก้ไขและเปลี่ยนแปลงภาพบางส่วน

ซึ่งข้อมูลจากส่วนนี้จะถูกเก็บไว้และนำไปใช้ในขั้นตอนต่อไป

2) Input Image

เป็นการรับภาพผ่านกล้อง Color Quick Cam ซึ่งถูกควบคุมด้วย ซอฟแวร์ ภาพที่ได้จะเป็นภาพของวัตถุที่มุมใดๆ ซึ่งจะมองเห็นเฉพาะส่วนของวัตถุที่มี เลเซอร์มากระทบเท่านั้น แล้วนำภาพที่ได้ส่งต่อไปในส่วนถัดไป

3) Primary Image Process

เป็นการนำภาพที่ได้จาก Input Image ประมวลผลเพื่อเลือกเอาส่วนที่ใช้งานไว้ (ซึ่งต้องการเฉพาะส่วนที่เป็นเส้นของเลเซอร์ไว้เท่านั้น) และกำจัดส่วนของภาพที่ไม่ต้องการออกไป เช่น สัญญาณรบกวน หรือ รายละเอียดของภาพที่ไม่ต้องการซึ่งจะใช้ Image Processing แบบ Noise Rejection และ Threshold

4) Store Image

นำภาพที่ผ่านการทำ Primary Image Process ในทุกๆมุมของการหมุนกล้องมาเก็บรวบรวมเอาไว้ด้วยกัน เพื่อรอการนำไปสร้างเป็นภาพ 3 มิติ ในขั้นต่อไป

5) Drive motor

จะสั่งงานโดยซอฟแวร์ให้หมุนมอเตอร์เพื่อหมุนกล้องไปตามองศาที่กำหนดไว้ โดยจะถูกหมุนจนครบ 1 รอบ ซึ่งจะได้ภาพ (Image) ในทุกองศาที่ต้องการ

6) Full Image Processing

เป็นของซอฟแวร์ที่จะนำข้อมูลภาพทั้งหมดที่ได้จากการถ่ายภาพวัตถุจนครบ 1 รอบแล้วมาสร้างเป็นภาพ 3 มิติ

7) Display Image

เป็นส่วนจัดเก็บภาพ 3 มิติที่ได้จากขั้นตอนที่แล้วในรูปแบบต่างๆ และแสดงภาพในแบบต่างๆออกมารวมทั้งการประยุกต์ใช้กับระบบกราฟฟิกอื่นๆ

บทที่ 3
ทฤษฎีพื้นฐานและหลักการทำงานของสเตปมอเตอร์

3.1 คุณลักษณะทั่วไป

สเตปมอเตอร์ คือ อุปกรณ์ที่เคลื่อนที่เป็นจังหวะโดยการกระตุ้นด้วยวิธีการทางแม่เหล็กไฟ-ฟ้าซึ่งเปลี่ยนสัญญาณดิจิตอลที่อินพุทซึ่งเป็นพัลซ์ไปเป็นการเคลื่อนที่แบบอนาลอกที่เอาท์พุท สเตปมอเตอร์บางครั้งถูกเรียกว่า สเตปปิ้งมอเตอร์ หรือ สเตปเปอร์มอเตอร์ เพราะว่าสเตปมอเตอร์เป็นอุปกรณ์ซึ่งเคลื่อนที่เมื่อถูกกระตุ้นโดยโวลท์เตจหรือกระแส ซึ่งโดยมากจะเป็นไฟฟ้ากระแสตรง เอาท์พุทของสเตปมอเตอร์จะมีจำนวนเท่ากับจำนวนคำสั่งอินพุทซึ่งมีลักษณะเป็นพัลซ์ โดยเมื่อป้อนกระแสให้กับสเตปมอเตอร์แล้วสเตปมอเตอร์จะมีการเคลื่อนที่เพิ่มขึ้น 1 สเตป

สเตปมอเตอร์มีมาประมาณ 40 กว่าปีมาแล้ว ในตอนนั้นการใช้งานสเตปมอเตอร์มีประสิทธิภาพต่ำกว่าเอซีมอเตอร์และดีซีมอเตอร์ แต่เมื่อไม่นานมานี้มีการนำดิจิตอลคอมพิวเตอร์เข้ามาใช้จึงมีการเปลี่ยนรูปแบบการควบคุมสเตปมอเตอร์ใหม่ โดยใช้การควบคุมด้วยไมโครโปรเซสเซอร์หรือไมโครคอนโทรลเลอร์ซึ่งช่วยให้สามารถใช้งานสเตปมอเตอร์ได้สะดวกขึ้นและมีประโยชน์ในการใช้สอยมากขึ้น ในปัจจุบันนี้สเตปมอเตอร์ได้ถูกนำมาใช้งานในอุปกรณ์ที่ใช้ในการควบคุมเชิงเลข (Numerical Control) การควบคุมกระบวนการ (Process Control) และการควบคุมอุปกรณ์ทางเครื่องกล (Machine Tool Control) เป็นต้น

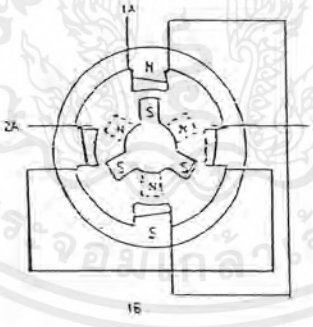
การควบคุมการทำงานของสเตปมอเตอร์โดยทั่วไปใช้วงจรขับ ซึ่งให้สเตปมอเตอร์ตอบสนองต่อสัญญาณพัลซ์ซึ่งทำให้มอเตอร์เคลื่อนที่เป็นจังหวะต่อเนื่องกันไป การเคลื่อนที่ลักษณะเช่นนี้ก่อให้เกิดการออสซิลเลทระหว่างสเตปของการเคลื่อนที่ทำให้สเตปมอเตอร์สั่นระหว่างใช้งาน การแก้ปัญหาดังกล่าวกระทำได้ด้วยวิธีการแด็มพ์ (Damping Methods) ของสเตปมอเตอร์ ซึ่งวิธีการแก้ปัญหาตามลักษณะเฉพาะของมอเตอร์แต่ละตัว

อีกแนวทางหนึ่งที่ใช้ในการแก้ปัญหาของการติดต่อระหว่างคอมพิวเตอร์กับมอเตอร์ ก็คือการใช้มอเตอร์ที่ง่ายต่อการควบคุมโดยคอมพิวเตอร์ซึ่งสเตปมอเตอร์เป็นมอเตอร์ที่เหมาะสมสำหรับงานเช่นนี้มากเพราะ มันสามารถควบคุมให้หมุนได้โดยใช้สภาวะเพียง 2 สถานะ คือ เปิด (On) และปิด (Off) ซึ่งสัญญาณที่ว่านี้เครื่องคอมพิวเตอร์สามารถสร้างขึ้นอย่างง่ายดาย

สเตปมอเตอร์มีขดลวดอยู่หลายขด โดยที่แต่ละขดถูกควบคุมการจ่ายกระแสโดยสัญญาณเปิด-ปิด ให้สามรถควบคุมตำแหน่งในการเคลื่อนที่ได้ตราบใดที่ยังไม่เกิดการเกินกำลัง (Overload) ตราบนั้นมันยังคงรักษาตำแหน่งเดิม จนกระทั่งมีการกระตุ้นโดยสัญญาณควบคุมการจ่ายกระแสอีกครั้ง และเนื่องจากสเตปมอเตอร์สามารถควบคุมตำแหน่งได้โดยการส่งสัญญาณกระตุ้นทำให้ไม่จำเป็นต้องใช้ตัววัดตำแหน่ง (Encoder) ในการทำงาน ข้อเสียที่สำคัญของ สเตปมอเตอร์ก็คือ มีข้อจำกัดด้านความเร็วเมื่อเปรียบเทียบกับ เซอร์โว มอเตอร์ (DC Servo Motor) ที่มีขนาดเท่ากันและยังมีการสั่นสะเทือน (Vibration) สูงทำให้การเคลื่อนที่เป็นไปอย่างไม่ราบเรียบ

3.2 หลักการทำงานของสเตปมอเตอร์

สเตปมอเตอร์สามารถแบ่งโครงสร้างทางกายภาพได้เป็น 2 ส่วน คือ สเตเตอร์ (Stator) และโรเตอร์ (Rotor) ตัวสเตเตอร์เป็นส่วนที่อยู่กับที่ ประกอบด้วยขดลวดทองแดงซึ่งพันอยู่รอบแกนเหล็ก เพื่อสร้างสนามแม่เหล็กเมื่อมีการจ่ายกระแสผ่านขดลวด ส่วนโรเตอร์เป็นส่วนที่เคลื่อนที่มีลักษณะเป็นแท่งเหล็กทรงกลม และที่ผิวรอบนอกมีลักษณะเป็นซีกฟันซึ่งทำจากแม่เหล็กถาวร ดังรูปที่ 3.1

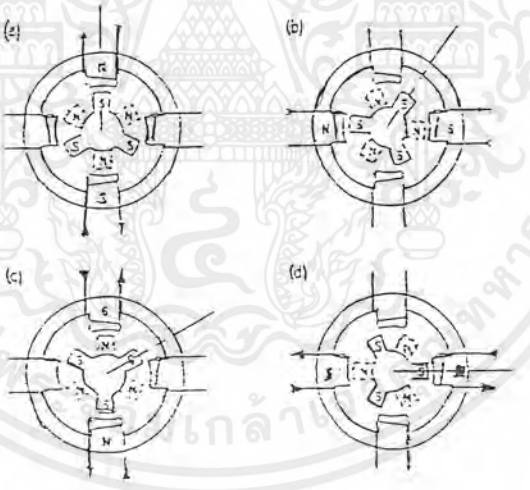


รูป 3.1 แสดงโครงสร้างของไฮบริดจ์สเตปมอเตอร์ที่มีจำนวนสเตปต่อรอบเท่ากับ 12

เมื่อยังไม่มีการจ่ายกระแสให้กับขดลวดของมอเตอร์ ซีกฟันอันใดอันหนึ่งของโรเตอร์จะอยู่ในตำแหน่งที่ตรงกันกับซีกฟันอันใดอันหนึ่งของสเตเตอร์ ทั้งนี้เป็นเพราะแม่เหล็กถาวรที่ตัวของโร-เตอร์พยามยามที่จะทำให้ค่าความต้านทานทางแม่เหล็กไฟฟ้า (Reluctance) ค่าน้อยที่สุด ซึ่ง ณ ที่จุดที่ซีกฟันของตัวโรเตอร์และสเตเตอร์ตรงกันนั้นมีค่าความต้านทานทางแม่เหล็กไฟฟ้าน้อยที่สุดทำให้

เกิดเส้นแรงแม่เหล็กไฟฟ้ามากที่สุด และจากรูปที่ 3.1 เส้นแรงแม่เหล็กไฟฟ้าจะทำให้เกิดขั้วแม่เหล็กเหนือและใต้ขึ้นมา 2 คู่ ทั้งที่ตัวสเตเตอร์และตัวโรเตอร์ ดังรูป ค่าทอร์ค (Torque) ที่ทำให้ตัวโรเตอร์สามรถยึดอยู่ในตำแหน่งดังกล่าวนี้เรียกว่า ดีเท็นทอร์ค (Detent Torque) (หมายความว่าการที่จะทำให้มอเตอร์เคลื่อนที่ในขณะที่ไม่ได้จ่ายกระแสให้กับขดลวดของแมเตอร์จะต้องออกแรงมากกว่าค่าของดีเท็นทอร์ค จึงจะทำให้โรเตอร์เริ่มเคลื่อนที่ได้) รูป 3.1 นั้นมี 12 ตำแหน่งที่สามารถเกิดดีเท็นท์ทอร์คได้

เมื่อจ่ายกระแสให้กับขดลวดที่อยู่ในสเตเตอร์คู่ใดคู่หนึ่ง ดังรูป 3.2 (a) จะทำให้เกิดขั้วแม่เหล็กเหนือและใต้ที่ซีกฟันของตัวสเตเตอร์ ซึ่งจะดึงดูดซีกฟันของตัวโรเตอร์ที่มีขั้วแม่เหล็กที่มีศักย์ต่างกันที่อยู่ใกล้ที่สุดเข้าไว้ ตำแหน่งนี้เรียกว่า สเตเบิ้ลโพสซิชั่น (Stable Position) ของโรเตอร์ จะมีจำนวนตำแหน่งเท่ากับจำนวนซีกฟันของโรเตอร์ และแรงที่จะทำให้โรเตอร์เปลี่ยนตำแหน่งไปจากตำแหน่งสเตเบิ้ลโพสซิชั่นได้นี้เรียกว่า โฮลดิ้ง ทอร์ค (Holding Torque)

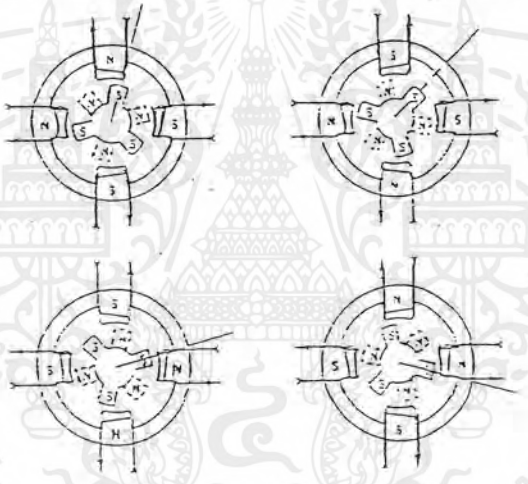


รูป 3.2 แสดงขั้นตอนการทำงานของสเตปมอเตอร์แบบ เต็มสเตปหนึ่งเฟส

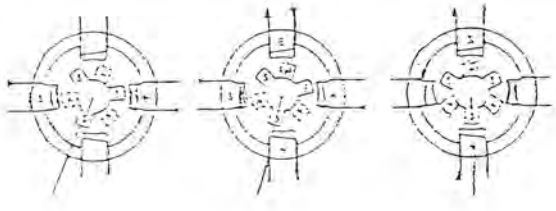
เมื่อสับเปลี่ยนการจ่ายกระแสให้แก่ขดลวด จากขดหนึ่งไปอีกขดหนึ่งเนื่องจากขดลวด วางอยู่ในตำแหน่งที่ต่างกัน 90° ก็จะทำให้ตัวสเตเตอร์ดึงดูดซีกฟันของตัวโรเตอร์อีกซีกหนึ่งที่ใกล้ที่สุดเข้าไว้ ซึ่งจะทำให้โรเตอร์เคลื่อนที่ไป 1 สเตปหรือ 30° ดังรูปที่ 3.2(b) จากนั้นก็เปลี่ยนไปจ่ายกระแสให้กับขดลวดชุดแรกโดยในคราวนี้เปลี่ยนทิศทางการไหลของกระแสให้กรงข้ามกับครั้งแรก ซึ่งจะทำให้ตัวโรเตอร์เคลื่อนที่ไปอีก 1 สเตป (เคลื่อนที่ไป 30°) ดังรูปที่ 3.2 (c) หลังจากนั้นก็ไปจ่ายกระแสให้กับขดลวดชุดที่สองโดยกลับทิศทางของกระแสที่ป้อนให้อีกเช่นกัน ทำให้โรเตอร์

หมุนไป 90° ดังรูป 3.2 (d) และถ้าหากเราป้อนกระแสให้กับมอเตอร์เหมือนที่เราป้อนในครั้งแรกแล้ว ซีกฟันซีกถัดไปของตัวโรเตอร์จะอยู่ในตำแหน่งที่เหมือนกับในรูปที่ 3.2 (a) อีกครั้งหนึ่ง ถ้าหากเราต้องการเคลื่อนที่หนึ่งรอบ เราต้องทำการกระตุ้นให้มอเตอร์เคลื่อนที่ไปจนครบ 12 สเตป และถ้าต้องการให้โรเตอร์หมุนไปอีกทิศทางหนึ่ง ก็ทำลารสลับลำดับในการจ่ายกระแส จากรูปที่ 3.2 (a), (b), (c) และ (d) ไปเป็น (a), (d), (c) และ (b) ตามลำดับ

ถ้าจ่ายกระแสให้แก่ขดลวดทั้งสองขดพร้อม ๆ กัน ซีกฟันของโรเตอร์จะอยู่ ณ.ตำแหน่งระหว่างซีกฟันของสเตเตอร์ เพราะฉะนั้นการจ่ายกระแสให้แก่มอเตอร์แบบนี้จะให้ทอร์คมากกว่าแบบที่จ่ายกระแสในเวลาขณะใดขณะหนึ่งเพียงขดเดียว ดังรูปที่ 3.3 ซึ่งแสดงถึงการป้อนกระแสให้กับขดลวดแต่ละขดเพื่อให้สเตปมอเตอร์เคลื่อนที่



รูปที่ 3.3 แสดงขั้นตอนการทำงานของสเตปมอเตอร์แบบเต็มสเตปสองเฟส



รูปที่ 3.4 แสดงขั้นตอนการทำงานของสเตปมอเตอร์แบบครึ่งสเตป

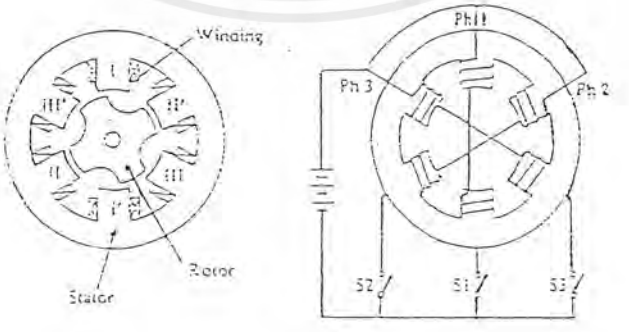
แต่ถ้าในตอนแรกจ่ายกระแสให้กับขดลวดพร้อม ๆ กันสองขด การทำอย่างนี้สลับกันไปจะทำให้โรเตอร์เคลื่อนที่ไปสเตปละ 15° ดังรูปที่ 3.4 และในการขับเคลื่อนแบบนี้จะทำให้สเตปต่อรอบ (Step/REV) เพิ่มขึ้นเป็นเท่าตัว การขับสเตปมอเตอร์แบบนี้เรียกว่า การขับแบบครึ่งสเตป (Half-Stepping) ซึ่งนิยมใช้กันในงานอุตสาหกรรมเป็นอย่างมาก แม้ว่าในบางครั้งจะให้ทอร์คน้อยกว่า แต่มีข้อดีกว่าการขับแบบเต็มสเตป (Full Stepping) ก็คือ การเคลื่อนที่เป็นไปอย่างราบเรียบ (ที่ความเร็วไม่สูงนัก) อีกทั้งมีความแม่นยำสูงและการสั่นสะเทือนน้อยกว่า

3.3 ชนิดของสเตปมอเตอร์

สเตปมอเตอร์สามารถแบ่งออกได้หลายชนิดตามลักษณะโครงสร้างและการใช้งาน ดังต่อไปนี้

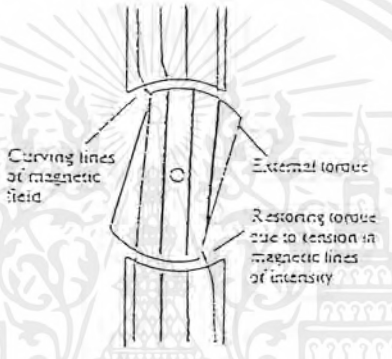
1) สเตปมอเตอร์ชนิดปรับค่ารีลัคแตนซ์ได้ (Variable Reluctance Stepping Motor)

สเตปมอเตอร์ชนิดนี้สามารถปรับค่ารีลัคแตนซ์ได้ ซึ่งรูปที่ 3.5 แสดงภาพตัดขวางของสเตปมอเตอร์แบบ 3 เฟส โดยที่สเตเตอร์มีฟันทั้งหมด 6 ซี่ ซี่ที่อยู่ตรงข้ามกันหรือทำมุม 180° ซึ่งกันและกันจะเป็นเฟสเดียวกัน ขดลวดที่พันอยู่ที่ฟันของสเตเตอร์ในแต่ละเฟสจะต่ออนุกรมหรือขนานก็ได้ จากรูปที่ 3.5 เป็นการต่อแบบอนุกรม ส่วนโรเตอร์นั้นมีฟัน 4 ซี่ ทั้งโรเตอร์และ สเตเตอร์ทำมาจากโลหะ ซิลิคอน ซึ่งมีสภาพซึมซับทางแม่เหล็กสูงและยอมให้สนามแม่เหล็กจำนวนมากไหลผ่านได้ฟันของสเตเตอร์ในเฟสเดียวกันจะมีขั้วต่างกันโดยซี่ฟัน I, II, III เป็นขั้วเหนือและซี่ I', II', III' เป็นขั้วใต้หลังจากถูกกระตุ้น

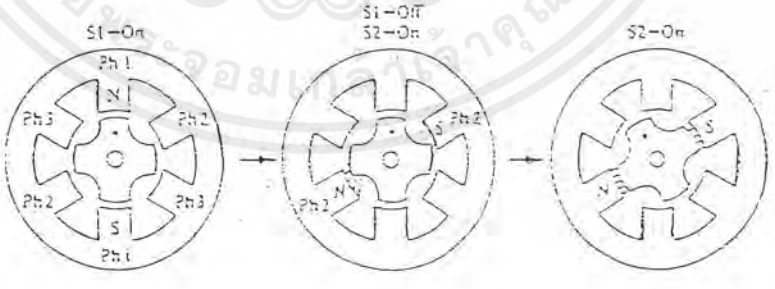


รูปที่ 3.5 แสดงภาพตัดขวางของสเตปมอเตอร์แบบ 3 เฟส

รูปที่ 3.6 แสดงตำแหน่งสมดุลย์ เมื่อเฟสใดเฟสหนึ่งของสเตปมอเตอร์ถูกกระตุ้น



รูปที่ 3.7 แสดงแรงภายนอกที่มีผลต่อเส้นแรงแม่เหล็ก



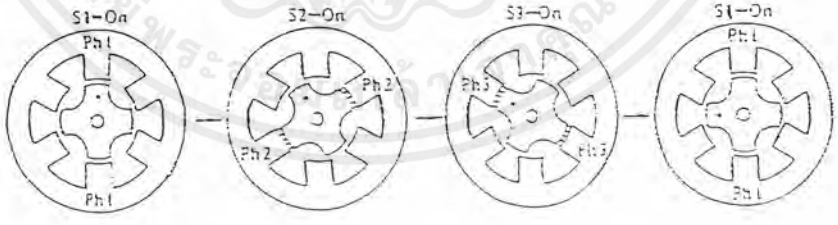
รูป 3.8 แสดงขั้นตอนการเคลื่อนที่ของโรเตอร์เมื่อสเตปมอเตอร์ถูกกระตุ้น

กระแสที่ไหลในแต่ละเฟสถูกควบคุมโดยสวิทช์ ปิด/เปิด ถ้าเฟส I ถูกกระตุ้นจะมี กระแสไหลและเกิดฟลักซ์แม่เหล็กดังแสดงในรูปที่ 3.6 แกนโรเตอร์จะอยู่ตำแหน่งเดียวกับซี่ I และ I' ทำให้ทั้งโรเตอร์และสเตเตอร์อยู่ในแนวเดียวกัน กรณีนี้จะทำให้ค่ารีลัคแตนซ์มีค่าน้อยที่สุดซึ่งเป็นตำแหน่งที่สมดุลย์ ถ้าโรเตอร์ถูกกระทำจากแรงภายนอกจะทำให้เปลี่ยนตำแหน่ง ดังรูปที่ 3.7 แรงบิดกระทำกับโรเตอร์ในทิศทางตามเข็มนาฬิกาทำให้ตำแหน่งเปลี่ยนไป มีผลทำให้เส้นแรงแม่เหล็กเคลื่อนที่จากซี่ของโรเตอร์และสเตเตอร์ เมื่อโรเตอร์และสเตเตอร์ไม่ได้อยู่ในแนวเดียวกันแล้วค่ารีลัคแตนซ์จะมีค่ามาก จากนั้นสเตปมอเตอร์จะทำตัวให้มีค่ารีลัคแตนซ์น้อยที่สุด การย้ายจากมุมที่เกิดการกระตุ้นแต่ละครั้งให้กลับไปยังตำแหน่งเดิมเรียกว่า สเตป

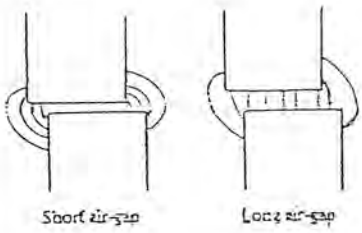
1.1 คุณสมบัติพื้นฐานของสเตปมอเตอร์แบบปรับค่ารีลัคแตนซ์

1. ช่องว่าง (Air Gap) ต้องมีขนาดเล็กที่สุด

ช่องว่างระหว่างฟันของโรเตอร์และสเตเตอร์ต้องมีขนาดเล็กที่สุดเท่าที่จะเป็นไปได้เพื่อให้ทอร์คที่เกิดขึ้นมีค่ามากและมีความแม่นยำทางตำแหน่งมากขึ้น รูปที่ 3.10 แสดงการเปรียบเทียบระหว่างช่องว่างที่กว้างและแคบในขณะที่มีแหล่งกำเนิดสนามแม่เหล็กแหล่งเดียวกันหรือมีระดับความแรงเท่ากัน ช่องว่างระหว่างฟันของโรเตอร์และสเตเตอร์ที่เล็กจะให้ทอร์คมากกว่าและทำให้การเคลื่อนที่จากจุดสมดุลย์น้อยกว่าช่องว่างขนาดใหญ่เมื่อมีแรงบิดจากภายนอกมากระทำต่อโรเตอร์



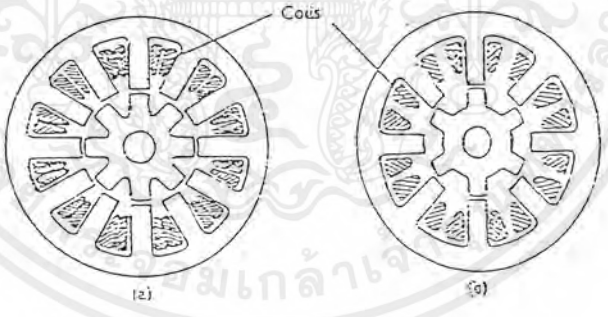
รูปที่ 3.9 แสดงขั้นตอนการเคลื่อนที่ของสเตปมอเตอร์



รูปที่ 3.10 แสดงการเปรียบเทียบเส้นแรงแม่เหล็กระหว่างช่องว่างที่กว้างและแคบ

2. มุมของการสเตปแคบ

คุณสมบัติอีกประการของสเตปมอเตอร์คือจะต้องมีมุมของการสเตปที่เล็กที่สุดเท่าที่จะเป็นไปได้ มุมที่แสดงในรูปที่ 3.6 ยังไม่ถือว่าเป็นมุมที่เล็ก แต่รูปที่ 3.11 (a) แสดงมอเตอร์ 3 เฟสซึ่งมีจำนวนฟันของโรเตอร์และสเตเตอร์เป็น 2 เท่าของรูปที่ 3.6 ส่วนรูปที่ 3.11 (b) แสดงมอเตอร์ 4 เฟส มุมของการสเตปของทั้ง 2 โครงสร้างนี้เท่ากับ 15^{o} นอกจากนี้ยังมีบางชนิดที่มีมุมของการสเตป 7.5^{o} โดยฟันของโรเตอร์และสเตเตอร์มี 12 และ 16 ซี่ตามลำดับ



รูปที่ 3.11 (a) แสดงมอเตอร์ 3 เฟส และ รูปที่ 3.11 (b) แสดงมอเตอร์ 4 เฟส

ความสัมพันธ์ของมุมของการสเตป θ_S , มุมเฟส m , จำนวนซี่ฟันของโรเตอร์ N_r , จำนวนสเตป S แสดงดังสมการ

$$S = 360/\theta_S = m N_r$$

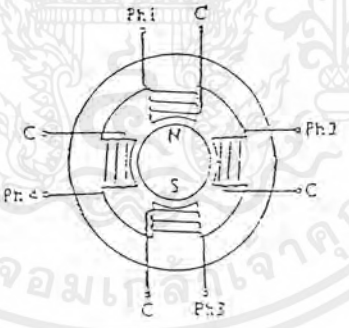
เพื่อที่จะลดขนาดของมุมที่สเตปลงต้องเพิ่มจำนวนซี่ฟันของโรเตอร์โดยที่โครงสร้างของแต่ละขั้วของเฟสใดๆ สเตเตอร์จะมีหลายซี่ฟัน แต่ก็ไม่ใช่องค์ประกอบโดยตรงที่จะกำหนดมุมของสเตปมอเตอร์

3. การสร้างสเตปมอเตอร์ให้มีโครงสร้างหลายสแตคเพื่อเพิ่มประสิทธิภาพ

โครงสร้างของสเตปมอเตอร์แบบนี้จะมี 1 เฟส โดยที่โรเตอร์และสเตเตอร์มีซี่ฟัน เหมือนกันซึ่งจะเป็นการเพิ่มประสิทธิภาพในด้านทอร์คต่อหน่วยปริมาตรของโรเตอร์

2) สเตปมอเตอร์แบบแม่เหล็กถาวร

สเตปมอเตอร์ชนิดนี้ใช้แม่เหล็กถาวรเป็นโรเตอร์ และมีซี่ฟันของสเตเตอร์ล้อมรอบ โดยซี่ฟันของสเตเตอร์ถูกพันด้วยขดลวดสำหรับสร้างสนามแม่เหล็ก เมื่อต้องการให้สเตปมอเตอร์แบบแม่เหล็กถาวรมีขนาดมุมของการสเตปเล็กลงจะต้องเพิ่มขั้วแม่เหล็กของโรเตอร์และจำนวนซี่ฟันของสเตเตอร์ แต่มีขีดจำกัดในการเพิ่มจำนวนขั้วแม่เหล็กของโรเตอร์ เนื่องจากการสร้างแม่เหล็กถาวรสร้างให้มีโครงสร้างโดยมีขั้วแม่เหล็กหลายขั้วทำได้ยาก

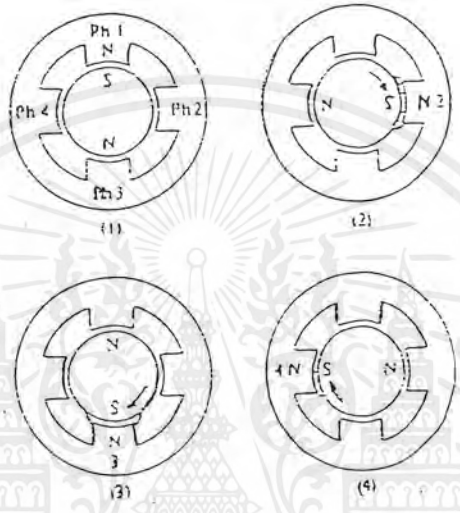


รูปที่ 3.12 แสดงโครงสร้างของสเตปมอเตอร์แบบแม่เหล็กถาวร

ตัวอย่างการทำงานของสเตปมอเตอร์แบบแม่เหล็กถาวร สมมุติว่าสเตปมอเตอร์แบบ แม่เหล็กถาวรขนาด 4 เฟส มีโรเตอร์เป็นแท่งเหล็กถาวรทรงกระบอกและสเตเตอร์มี 4 ซี่ฟันซึ่งรอบๆ พันด้วยขดลวด มีรูปแบบพื้นฐานของการทำงาน คือ เมื่อสร้างสัญญาณกระตุ้นตามลำดับเฟส โรเตอร์จะหมุนไปตามทิศทางของการกระตุ้น ดังแสดงในรูปที่ 3.13

ข้อเสียของสเตปมอเตอร์แบบแม่เหล็กถาวร คือ มีขนาดมุมสเตปใหญ่ทำให้ความละเอียดของการสเตปต่อรอบน้อยเนื่องจากโครงสร้างของโรเตอร์เป็นแม่เหล็กถาวร การสร้างแม่เหล็กถาวร

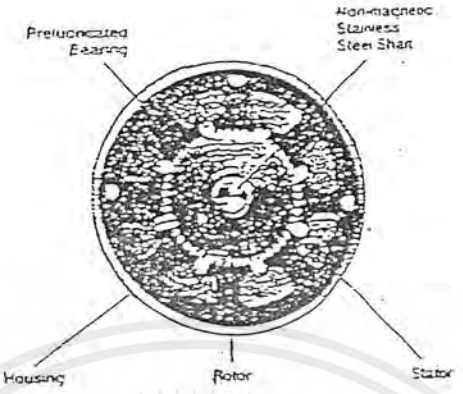
ให้มีขั้วหลายขั้วทำได้ยากทำให้ไม่สามารถสร้างสเตปเตอร์ขนาดเล็กได้ สเตปมอเตอร์แบบแม่เหล็กถาวรส่วนใหญ่จะมีโครงสร้างขนาดเล็ก ทำให้ค่าทอร์คที่ได้ต่อหน่วยต่อปริมาตรมีค่าต่ำ ถ้าต้องการปรับปรุงประสิทธิภาพในเรื่องของทอร์ค แม่เหล็กถาวรที่ใช้ต้องทำจากสารแม่เหล็กที่มีสภาพ ความเป็นแม่เหล็กสูง



รูปที่ 3.13 แสดงการทำงานของสเตปมอเตอร์แบบแม่เหล็กถาวรขนาด 4 เฟส

3) สเตปมอเตอร์แบบไฮบริดจ์

สเตปมอเตอร์ชนิดนี้มีแกนโรเตอร์เป็นแม่เหล็กถาวร โดยมีการทำงานร่วมกันของมอเตอร์แบบแม่เหล็กถาวรและมอเตอร์แบบปรับค่ารีลัคแตนซ์ได้ ไฮบริดจ์สเตปมอเตอร์นี้มีโครงสร้างของสเตเตอร์คล้ายกับโครงสร้างของสเตปมอเตอร์แบบปรับค่ารีลัคแตนซ์ได้ แต่ต่างกันที่การต่อขดลวดโดยที่แต่ละเฟสของสเตปมอเตอร์แบบปรับค่ารีลัคแตนซ์ได้จะมีขดลวด 2 ขด แต่ละขดมีขั้วต่างกันแต่ไฮบริดจ์สเตป มอเตอร์ขดลวดทั้งสองจะพันอยู่ที่ขั้วเดียวกัน เรียกว่า ไบโพล่าร์ (Bipolar) ซึ่งในการกระตุ้นแต่ละครั้งจะให้ขั้วที่แตกต่างกัน



รูปที่ 3.14 แสดงโครงสร้างของไฮบริดจ์สเตปมอเตอร์

3.1 คุณสมบัติที่สำคัญของไฮบริดจ์สเตปมอเตอร์

โครงสร้างของมอเตอร์จะมีแม่เหล็กถาวรอยู่ตรงกลางระหว่างเฟสทั้งสอง การเหนี่ยวนำสนามแม่เหล็กทำได้โดยใช้สนามแม่เหล็กซึ่งสร้างจากสเตเตอร์ซึ่งเป็นสนามแม่เหล็กแบบเฮทเทอโรโพลาร์ (Heteropolar Field) ดังนั้นทอร์คเกิดจากการทำงานร่วมกันของสนามแม่เหล็ก 2 ชนิดคือ สนามจากแม่เหล็กถาวรและสนามแม่เหล็กเหนี่ยวนำที่เกิดจากการกระตุ้นของขดลวดแต่ละขดโครงสร้างของซี่ฟันของสเตเตอร์จะใหญ่กว่าซี่ฟันของโรเตอร์เล็กน้อยเพื่อเพิ่มความถูกต้องแม่นยำทางตำแหน่งของการเคลื่อนที่

หลักกการทำงานไฮบริดจ์สเตปมอเตอร์ที่แตกต่างจากสเตปมอเตอร์แบบแปรค่ารีลัคแตนซ์ ได้คือ แรงบิดที่เกิดขึ้นจากสนามแม่เหล็กจะไม่ขึ้นอยู่กับกระแสที่ไหลผ่านขดลวดเพียงอย่างเดียว ไม่ขึ้นอยู่กับโครงสร้างของซี่ฟันด้วย ซึ่งซี่ฟันถูกออกแบบเพื่อให้ได้โครงสร้างขนาดเล็ก และใช้แม่เหล็กถาวรเป็นแกนกลางเพื่อลดผลของการออกซิเสททางแมคคานิคส์

ข้อดีของไฮบริดส์สเตปมอเตอร์คือมีขนาดสเตปขนาดเล็กมีความละเอียดของสเตปต่อรอบสูง มีค่าทอร์คสูงกว่าสเตปมอเตอร์แบบแปรค่ารีลัคแตนซ์ได้ แต่สเตปมอเตอร์แบบแปรค่ารีลัคแตนซ์ได้มีแรงเฉื่อยทางแมคคานิคส์น้อยกว่าไฮบริดจ์สเตปมอเตอร์

นอกจากสเตปมอเตอร์ทั้ง 3 ชนิดที่กล่าวมาแล้วยังมีสเตปมอเตอร์ อื่น ๆ ที่ไม่ได้กล่าวถึงอีก เช่น ลิเนียร์สเตปมอเตอร์ ซึ่งเป็นมอเตอร์ที่ได้รับการออกแบบให้มีการเคลื่อนที่แบบเป็นเชิงเส้น อิเล็คโตรดไฮดรอลิคสเตปมอเตอร์ซึ่งเป็นสเตปมอเตอร์กำลังสูงที่ใช้ในงานอุตสาหกรรม เป็นต้น

3.4 การควบคุมการทำงานของสเตปมอเตอร์

วิธีการควบคุมการทำงานของสตปมอเตอร์โดยการกระตุ้นเฟสมี 3 วิธี คือ

1) การกระตุ้นแบบซิงเกิลเฟส (Single Phase Excitation)

เป็นการกระตุ้นแบบเฟสเดี่ยวตามสัญญาณพัลส์ที่ป้อนเข้ามาที่ชุดขับสเตปมอเตอร์ ดังนี้

PHASE	PULSE							
A	1	0	0	0	1	0	0	0
B	0	1	0	0	0	1	0	0
C	0	0	1	0	0	0	1	0
D	0	0	0	1	0	0	0	1

2) การกระตุ้นแบบสองเฟส (Two Phase Excitation)

เป็นการกระตุ้นแบบทีละสองเฟสคู่พร้อมกัน ดังนี้

PHASE	PULSE							
A	1	0	0	1	1	0	0	0
B	1	1	0	0	1	1	0	0
C	0	1	1	0	0	1	1	0
D	0	0	1	1	0	0	1	1

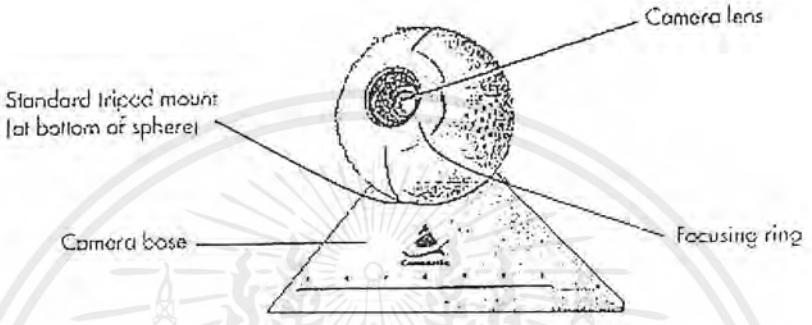
3) การกระตุ้นแบบครึ่งสเตป (Half Step Excitation)

เป็นการรวมรูปแบบการหมุนของทั้งสองแบบไว้ในแบบเดียวกัน

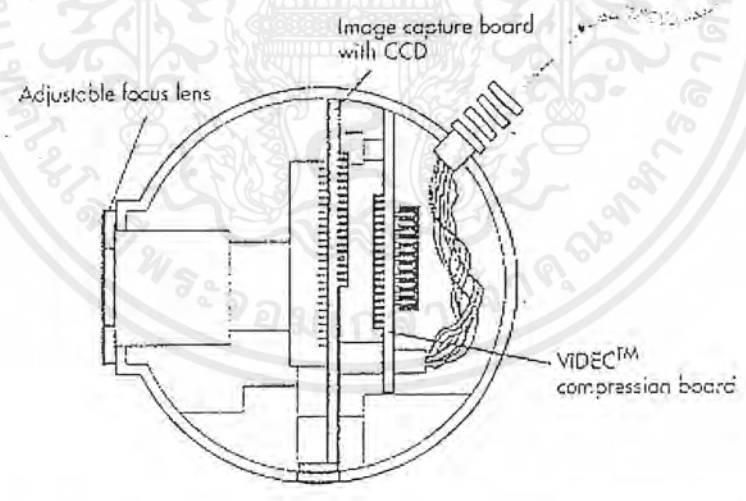
PHASE	PULSE							
A	1	1	0	0	0	0	0	1
B	0	1	1	1	0	0	0	0
C	0	0	0	1	1	1	0	0
D	0	0	0	0	0	1	1	1

วิธีการกระตุ้นเฟสทั้ง 3 แบบนี้จะมีคุณสมบัติที่แตกต่างกัน ดังนั้นการเลือกวิธีการกระตุ้นจึงจำเป็นที่จะต้องพิจารณาเพื่อให้เหมาะสมกับงานนั้น ๆ การกระตุ้นแบบเฟสเดียว จะเป็นแบบที่มีความเที่ยงตรงของตำแหน่งสูง แตจะมีแรงบิด (Torque) น้อย ส่วนการกระตุ้นแบบ 2 เฟส มีความเที่ยงตรงของตำแหน่งน้อยกว่าแบบแรก แต่มีแรงบิดสูงกว่าและการกระตุ้นแบบครึ่งสเตปเป็นแบบที่มีความเที่ยงตรงของตำแหน่งน้อยมาก แต่จะมีแรงบิดสูงมากเช่นกัน

บทที่ 4
กล้อง Color Quick Cam



รูปที่ 4.1 แสดงลักษณะภายนอกของกล้อง Color Quick Cam



รูปที่ 4.2 แสดงลักษณะภายในของกล้อง Color Quick Cam

4.1 สิ่งที่จำเป็นต้องมีในการใช้งานกล้อง Color Quick Cam

ได้แก่

- เครื่องไมโครคอมพิวเตอร์ตั้งแต่รุ่น 486 ขึ้นไป
- Microsoft Windows 3.1, Windows for Workgroups3.11 หรือ Windows 95

- หน่วยความจำ 8 MB สำหรับ Windows3.1 หรือ 12 MBสำหรับ Windows 95
- Video for Windows1.1 e หรือสูงกว่านั้น (รวมการติดตั้ง)
- พอร์ตขนาน 1 พอร์ต โดยพอร์ตเป็นแบบทิศทางเดียว (Unidirectional) หรือ 2 ทิศทาง (Bidirectional)
- หน่วยความจำในฮาร์ดดิสก์ 2 MB สำหรับซอฟแวร์ ของกล้อง Color Quick Cam
- หน่วยความจำในฮาร์ดดิสก์อย่างน้อย 5 MB สำหรับสร้างภาพเคลื่อนไหว
- หน่วยความจำในฮาร์ดดิสก์ 350 KB สำหรับภาพ 24 bit color (640×480 pixel) หรือประมาณ 80 KB สำหรับภาพ 320×240 pixelใน JPEG
- ไมโครโฟนสำหรับการบันทึกเสียงในภาพแบบ Movie
- Sound Card ซึ่ง Compatible กับ Windows

4.2 คุณสมบัติพิเศษของกล้อง Color Quick Cam

- ให้ภาพที่มีความละเอียด 640 × 480 pixel 24 bit color
- Frame Rate 30 fps (เปลี่ยนแปลงได้ตามขนาดของ Windows และ ไมโครคอมพิวเตอร์)
- ได้รับไฟเลี้ยงจากคีย์บอร์ดด้วยกำลังไฟอย่างน้อย 2 วัตต์
- ปรับโฟกัสได้จาก 1 นิ้ว ถึง Infinity (Maximum Len Speed = f/1.6)

4.3 ความสามารถของกล้อง Color Quick Cam

1)เก็บภาพเคลื่อนไหวในไมโครคอมพิวเตอร์

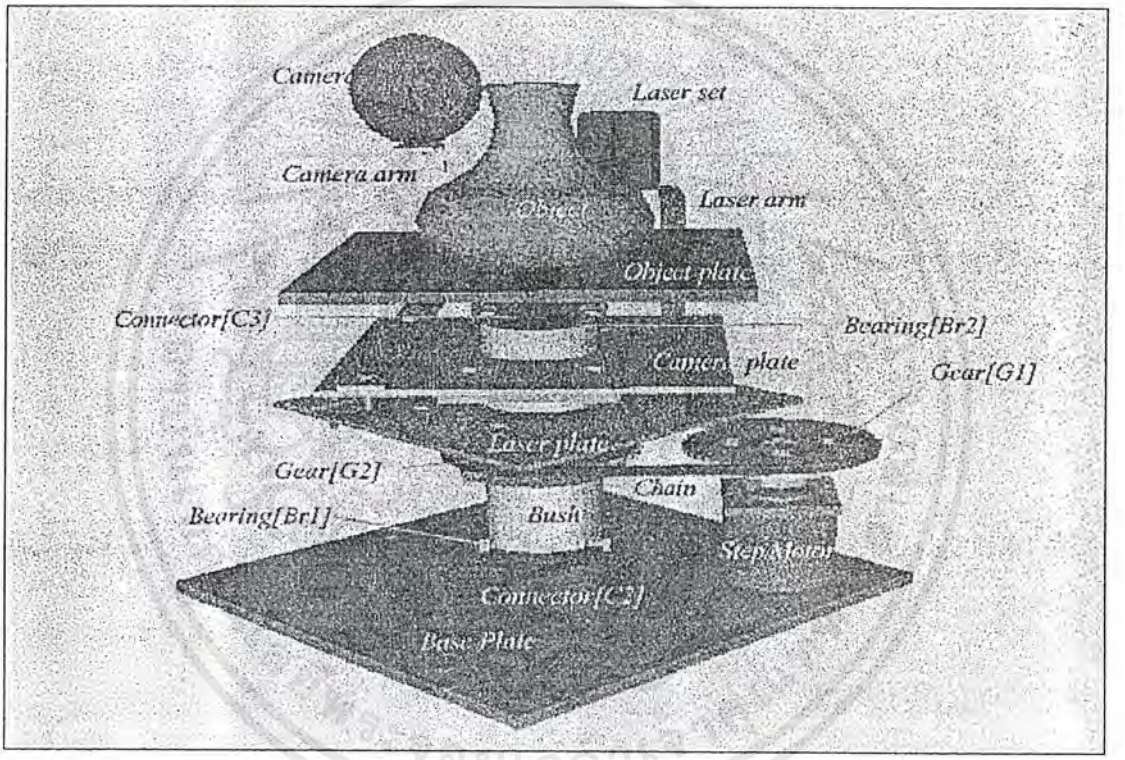
การใช้งานกล้อง Color Quick Cam นั้นเป็นวิธีที่ง่ายและประหยัดวิธีหนึ่งในการเก็บภาพเคลื่อนไหวไว้ใน ไมโครคอมพิวเตอร์ ทำให้สามารถที่จะติดต่อสื่อสารกัน โดยแสดงภาพเคลื่อนไหวได้และสามารถหลีกเลี่ยงปัญหาในขั้นตอนการติดตั้งฮาร์ดแวร์ , การซื้อ Video Board ที่แพง หรือการศึกษาถึงวิธีการใช้งาน การใช้กล้อง Color Quick Cam นั้นจะช่วยให้เราสามารถที่จะ Capture ภาพได้ง่ายและทำการดัดแปลงแก้ไขได้ด้วย

2)เก็บภาพนิ่งในไมโครคอมพิวเตอร์

กล้อง Color Quick Cam มีซอฟแวร์ที่ทำให้สามารถบันทึกภาพนิ่งให้อยู่ในรูปแบบ BMP, TIFF หรือ JPEG ซึ่งสามารถนำมาประมวลผลใน Word Processing , Page Layout , การ Present งาน หรือโปรแกรม Graphics ซึ่งภาพนิ่งเหล่านั้นสามารถนำไปใช้ในการบอกตำแหน่ง , การใช้ในงานพิมพ์ Graphics ที่ต้องการคุณภาพสูง เช่น จดหมาย หรือเอกสารสำคัญต่าง ๆ

บทที่ 5
ชุดทดลองต้นแบบ

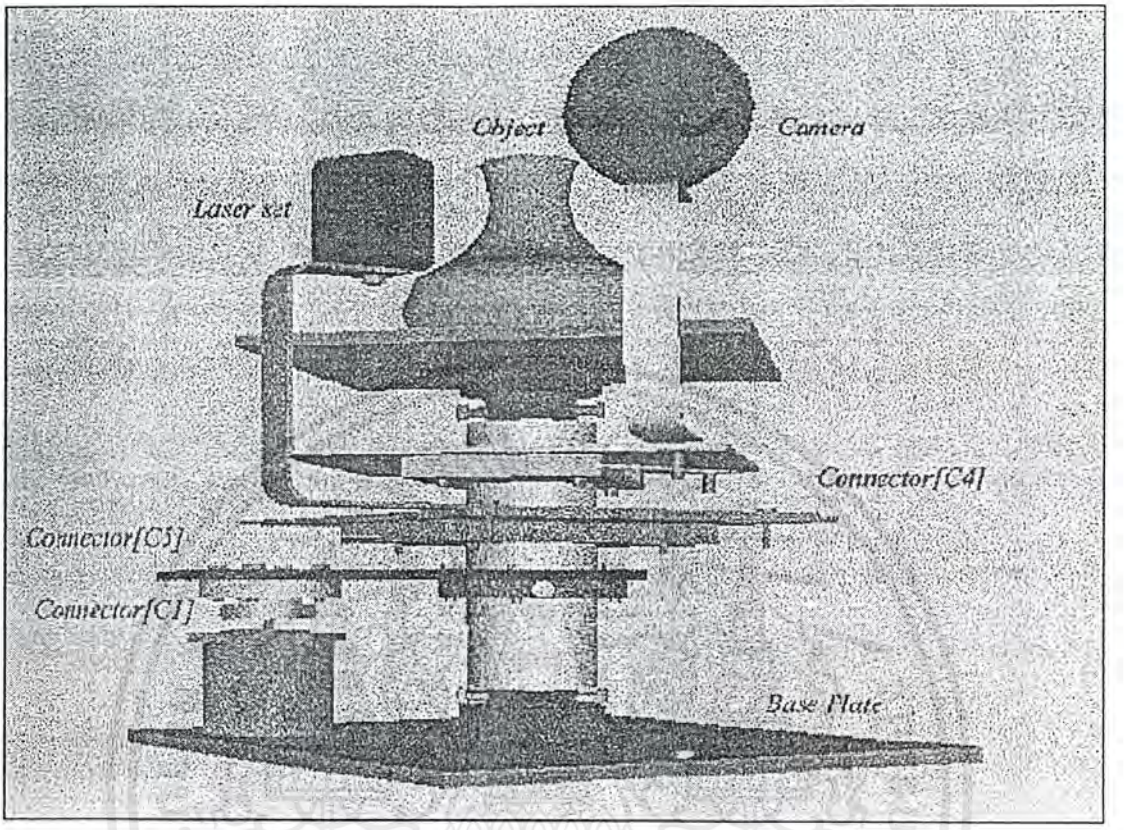
5.1 โครงสร้างและส่วนประกอบของเครื่องเก็บข้อมูลภาพ 3 มิติ



รูปที่ 5.1 โครงสร้างและส่วนประกอบโดยรวมของชุดทดลองต้นแบบ (ด้านหน้า)

5.2 หลักการทำงาน

- ต้นกำลังของระบบหมุนกล้องนี้จะเป็นมอเตอร์แบบสเตปปิ้ง ซึ่งถูกสั่งงานจากคอมพิวเตอร์
- กำลังจากมอเตอร์จะส่งผ่าน เฟือง [G1] ผ่านโซ่ ไปสู่เฟือง [G2] โดยมีอัตราทดเป็น 1:1
- ส่วนที่ไม่มีการเคลื่อนที่ คือ แผ่นรองวัตถุ (Object Plate) จะถูกยึดติดกับเพลากลาง (Shaft) โดยผ่าน [C3]



รูปที่ 5.2 โครงสร้างและส่วนประกอบโดยรวมของชุดทดลองต้นแบบ (ด้านหลัง)

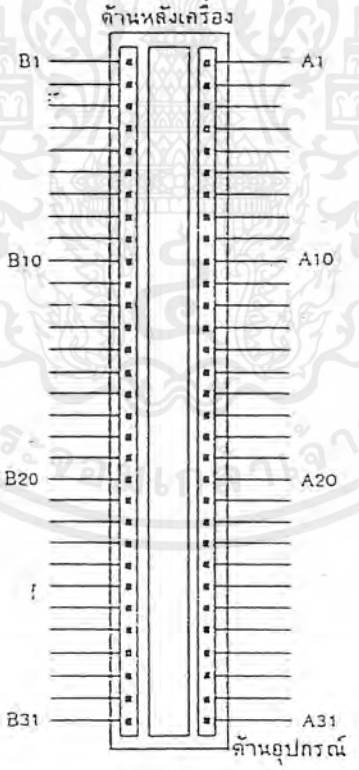
- แผ่นรองวัตถุ (Object Plate) , เพลากลาง ,ฐานรอง (Base) ,แผ่นรอง (Plate) , [C2] และ [C3] นั้นจะทำหน้าที่รองรับ น้ำหนัก ของวัตถุนอกจากนี้เพลากลาง ยังทำหน้าที่เป็นแกนกลางในการหมุนของกล้องอีกด้วย
- ปลอก (Bush) จะยึดติดกับเพลากลาง โดยที่ลูกปืน ([Br1] และ [Br2]) และสามารถหมุนได้อิสระจากตัวกลาง ปลอกจะยึดติดกับเฟือง [G2] ซึ่งจะหมุนตามการทำงานของมอเตอร์
- ชุดขายึดกล้องจะเป็นแผ่นรอง (Camera Plate) ยึดติดกับปลอกโดยผ่านตัวยึด [C4] และแผ่นรองก็จะยึดติดกับแขนยึดกล้อง (Camera Arm)
- ชุดขายึดเลเซอร์จะเป็นแผ่นรอง (Laser Plate) ยึดติดกับปลอกโดยผ่านตัวยึด [C5] และแผ่นรองก็จะยึดติดกับแขนเลเซอร์ ดังนั้น ตัวกล้องและเลเซอร์จะหมุนตามการทำงานของมอเตอร์ โดยวัตถุจะอยู่กับที่

บทที่ 6
การ์ดควบคุมการหมุน Stepping Motor

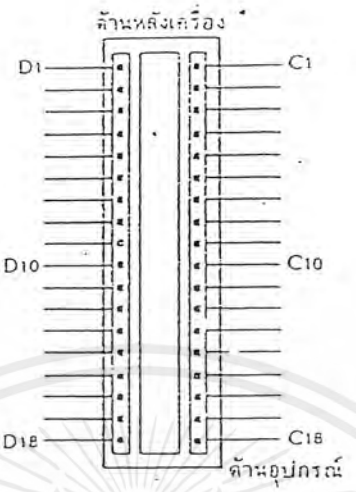
เนื่องจากพอร์ตขนานของไมโครคอมพิวเตอร์ถูกใช้ไปแล้วโดยกล้อง Color Quick Cam ดังนั้นการควบคุม สเตปมอเตอร์ จึงจำเป็นต้องใช้ Address ของพอร์ตใหม่ ดังนั้นจึงจำเป็นต้องสร้างการ์ดขึ้นมา

6.1 ISA Slot

ภายในเครื่องคอมพิวเตอร์นั้นจะมีช่องเสียบที่เรียกว่า Slot PC โดยการนำการ์ดมาเสียบที่ Slotนี้ เพื่อเป็นการเชื่อมต่อระหว่างเครื่องคอมพิวเตอร์กับอุปกรณ์ภายนอก



รูปที่ 6.1 แสดงการนับขาของ Slot แบบ 62 ขา



รูปที่ 6.2 แสดงการนับขาของ Slot แบบ 36 ขา

ตารางที่ 6.1 แสดงชื่อของสัญญาณขาต่างๆ ของSlot

ขาอินพุท	ชื่อสัญญาณ	อินพุท/เอาท์พุท
A1	-I/O CH CK	I
A2	SD7	I/O
A3	SD6	I/O
A4	SD5	I/O
A5	SD4	I/O
A6	SD3	I/O
A7	SD2	I/O
A8	SD1	I/O
A9	SD0	I/O
A10	-I/O CH RDY	I
A11	AEN	O
A12	SA19	I/O
A13	SA18	I/O
A14	SA17	I/O
A15	SA16	I/O
A16	SA15	I/O
A17	SA14	I/O
A18	SA13	I/O
A19	SA12	I/O
A20	SA11	I/O
A21	SA10	I/O
A22	SA9	I/O
A23	SA8	I/O
A24	SA7	I/O
A25	SA6	I/O
A26	SA5	I/O

ตารางที่ 6.1 ต่อ

ขาอินพุท	ชื่อสัญญาณ	อินพุท/เอาท์พุท
A27	SA4	I/O
A28	SA3	I/O
A29	SA2	I/O
A30	SA1	I/O
A31	SA0	I/O
B1	GND	กราวน์
B2	-	I
B3	+ 5 Vdc	แหล่งจ่ายไฟเลี้ยง
B4	IRQ9	I
B5	-5 Vdc	แหล่งจ่ายไฟเลี้ยง
B6	DRQ2	I
B7	-12 Vdc	แหล่งจ่ายไฟเลี้ยง
B8	IOWS	I
B9	+12 VdC	แหล่งจ่ายไฟเลี้ยง
B10	GND	กราวน์
B11	-SMEMW	O
B12	-SMEMR	O
B13	-IOW	I/O
B14	-IOR	I/O
B15	-DACK3	O
B16	DRQ3	I
B17	-DACK1	O
B18	-Refresh	I/O
B19	CLK	O
B20	IPQ7	I
B21	IRQ6	I
B22	IRQ5	I

ตารางที่ 6.1 ต่อ

ขาอินพุท	ชื่อสัญญาณ	อินพุท/เอาท์พุท
B23	IRQ5	I/O
B24	IRQ4	I/O
B25	IRQ3	I/O
B26	-DACK2	O
B27	T/C	O
B28	BALE	O
B29	+5 Vdc	แหล่งจ่ายไฟเลี้ยง
B30	OSC	O
B31	GND	กราวน์
C1	SBHE	I/O
C2	LA23	I/O
C3	LA22	I/O
C4	LA21	I/O
C5	LA20	I/O
C6	LA19	I/O
C7	LA18	I/O
C8	LA17	I/O
C9	-MEMR	I/O
C10	-MEMW	I/O
C11	SD08	I/O
C12	SD09	I/O
C13	SD10	I/O
C14	SD11	I/O
C15	SD12	I/O
C16	SD13	I/O
C17	SD14	I/O
C18	IRQ5	I/O

ตารางที่ 6.1 ต่อ

ขาอินพุท	ชื่อสัญญาณ	อินพุท/เอาท์พุท
D1	IRQ4	I
D2	IRQ3	I
D3	IRQ10	I
D4	IRQ11	I
D5	IRQ12	I
D6	IRQ13	I
D7	IRQ14	I
D8	-DACK	O
D9	DRQ0	I
D10	-DACK5	O
D11	DRQ5	I
D12	-DACK6	O
D13	DRQ6	I
D14	-DACK7	O
D15	DRQ7	I
D16	+5 Vdc	แหล่งจ่ายไฟเลี้ยง
D17	-MASTER	I
D18	GND	กราวน์

สัญญาณที่ออกจาก Slot PC ที่ใช้กับการ์ดมีดังนี้

A0-A11 : เป็นAddressของระบบที่ใช้ติดต่อกับหน่วยความจำและอุปกรณ์อินพุท/เอาท์พุท

D0-D7 : เป็นสัญญาณข้อมูลขนาด 8 บิตที่ใช้ติดต่อกับหน่วยความจำไมโครโปรเซสเซอร์

IOW : เป็นสัญญาณเขียนข้อมูลลงบนอุปกรณ์อินพุท/เอาท์พุท สัญญาณนี้ควบคุมจากไมโครโปรเซสเซอร์ Active ที่ 0

RES : สัญญาณนี้ใช้สำหรับ Reset ระบบในขณะเปิดเครื่องหรือขณะที่แหล่งจ่ายไฟเลี้ยงหยุดจ่าย

AEN : Enable Address (เป็นเอาท์พุต)

1) ประกอบของการ์ด

- ส่วนเปรียบเทียบ Address
- ส่วนรับส่งข้อมูล
- ส่วนวงจรขับสเตปมอเตอร์ (Drive Stepping Motor)

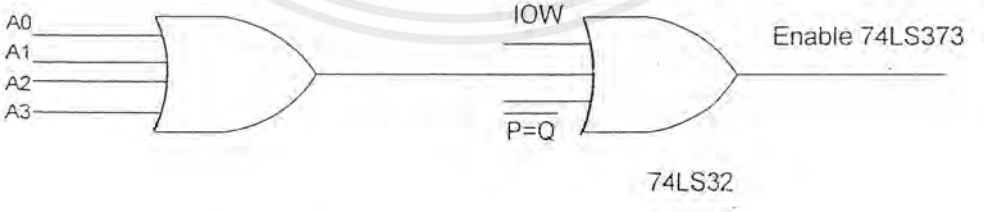
2) หลักการทำงาน

- ส่วนเปรียบเทียบ Address

จะใช้ IC 74LS688 เป็นตัวเปรียบเทียบค่า Address A4-An ของไมโครคอมพิวเตอร์กับค่า Address ที่กำหนดไว้ที่ DIP Switch ว่าตรงกันหรือเปล่า ถ้าตรงกันก็จะให้สัญญาณ Enable ออกไป ซึ่งตัว 74LS688 จะได้รับสัญญาณ Enable จากขา A_{EN} ของไมโครคอมพิวเตอร์โดยผ่าน Buffer 74LS245

- ส่วนรับส่งข้อมูล

จะใช้ IC 74F245 ซึ่งเป็น Buffer รับข้อมูลจากขาข้อมูลของไมโครคอมพิวเตอร์ แล้วส่งออกไปให้กับส่วนวงจรขับสเตปมอเตอร์ (Drive Stepping Motor) โดย 74F245 นี้จะถูกEnable โดยเอาท์พุทของ OR Gate ซึ่งการทำงานของ Gate ต่างๆ ดังรูป



รูปที่ 6.3 สัญญาณ Enable Data Buffer

โดยที่

- A0-A3, $\overline{IOW}$ เป็นสัญญาณจากไมโครคอมพิวเตอร์โดยผ่าน Buffer 74LS245
- $\overline{P=Q}$ เป็นเอาท์พุท ของส่วนเปรียบเทียบ Address มีค่าเป็น 0 เมื่อ A4-An ตรงกับค่า Address ที่

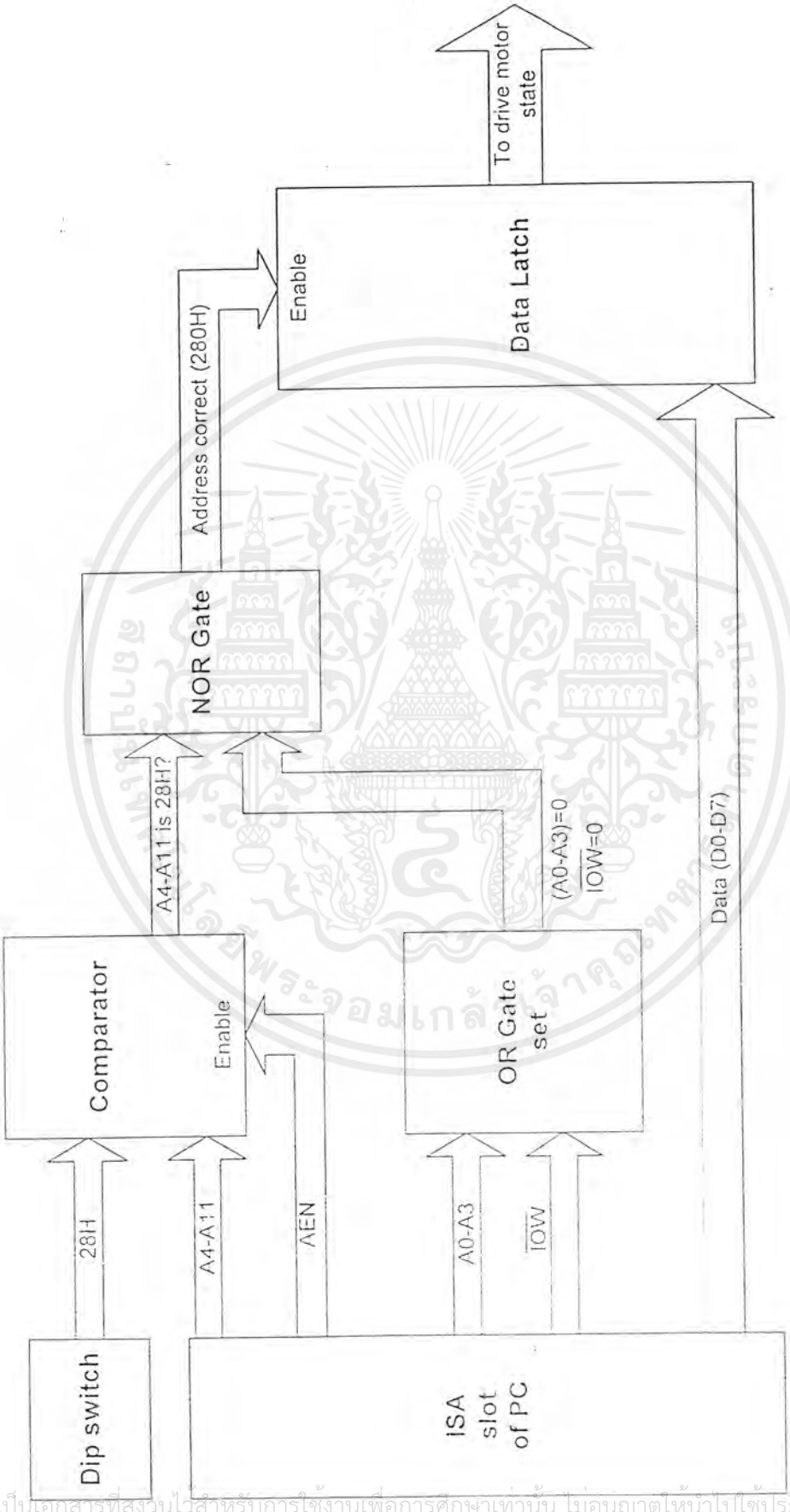
ตั้งไว้ค่า Enable จะเป็น 0 ก็ต่อเมื่อ ทั้ง 3 กรณีต่อไปนี้เป็นจริง

- $\overline{\text{IOW}}$ Active (เป็น 0)
- P=Q Active (เป็น 0)
- A0-A3 Active (เป็น 1 ทั้งหมด)

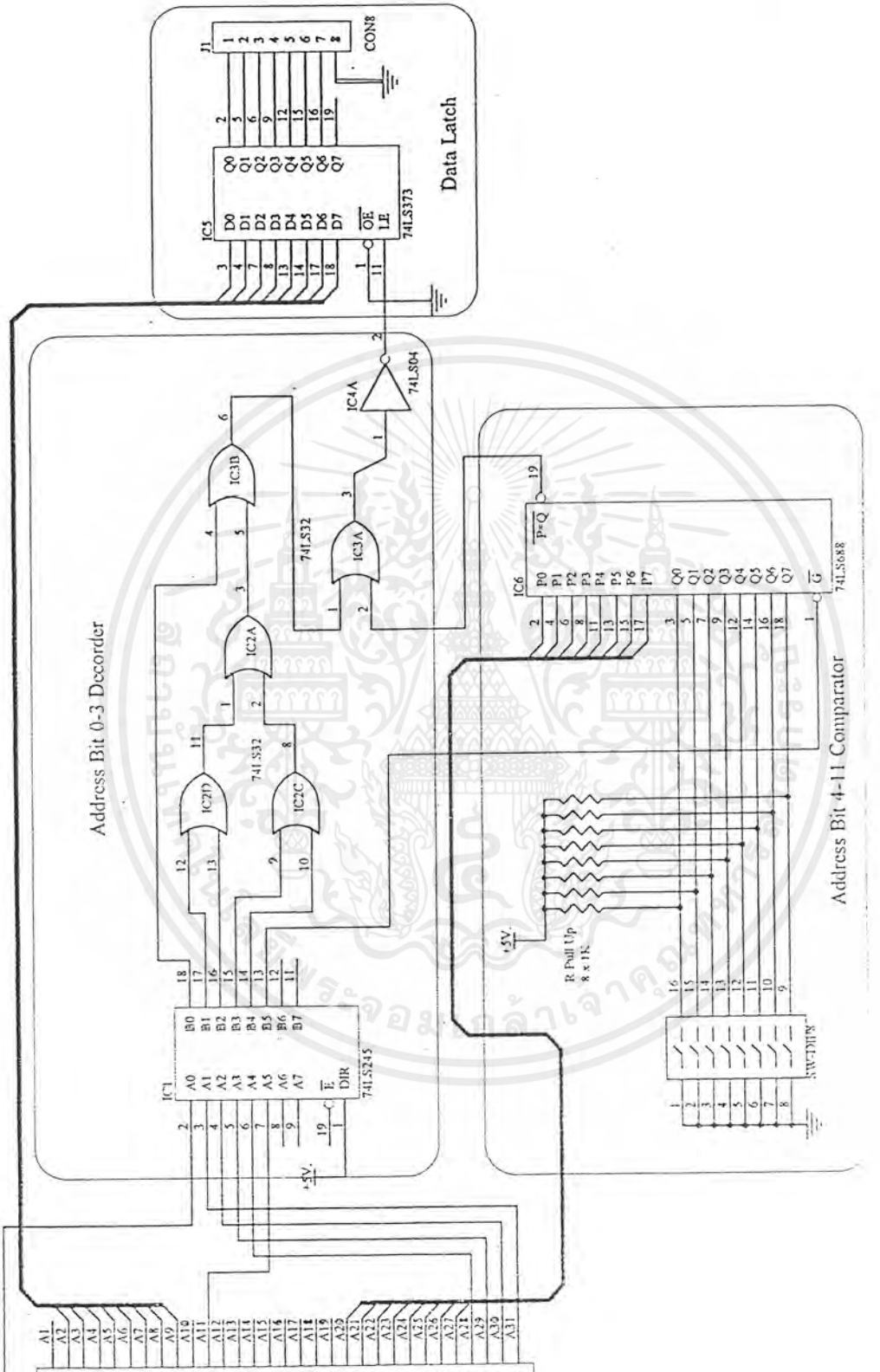
ค่า Direction ของ Buffer ทั้ง 2 ตัว คือ 74LS245 กับ 74LS288 ต่อกับไฟเลี้ยง (VCC) เพราะต้องการส่งข้อมูลทางเดียวจาก A ไป B

- ส่วนวงจรขับสเตปมอเตอร์

ข้อมูล 8 บิตจากไมโครคอมพิวเตอร์จะถูกนำมาควบคุมการหมุนของสเตปมอเตอร์ โดย Bit0 - Bit3 ต่อเข้ากับ Phase1 - Phase4 ของมอเตอร์ผ่าน Opto Isolator เพื่อแยกสัญญาณ 5 โวลต์ จากไมโครคอมพิวเตอร์กับแรงดันที่ใช้ขับสเตปมอเตอร์ ซึ่งมีค่าประมาณ 15 โวลต์ นอกจากนี้จะมี LED แสดงสถานะ On/Off ในแต่ละเฟสอีกด้วย สัญญาณที่ผ่าน Opto Isolator จะป้อนให้กับชุด Darlington ของทรานซิสเตอร์ (Transistor)เบอร์ 2222 และ Power Transistor เบอร์ 6123 เพื่อขยายกระแสให้เพียงพอที่จะนำไปขับสเตปมอเตอร์ในแต่ละเฟส โดยมีไดโอด (Diode) เป็น Free Wheeling

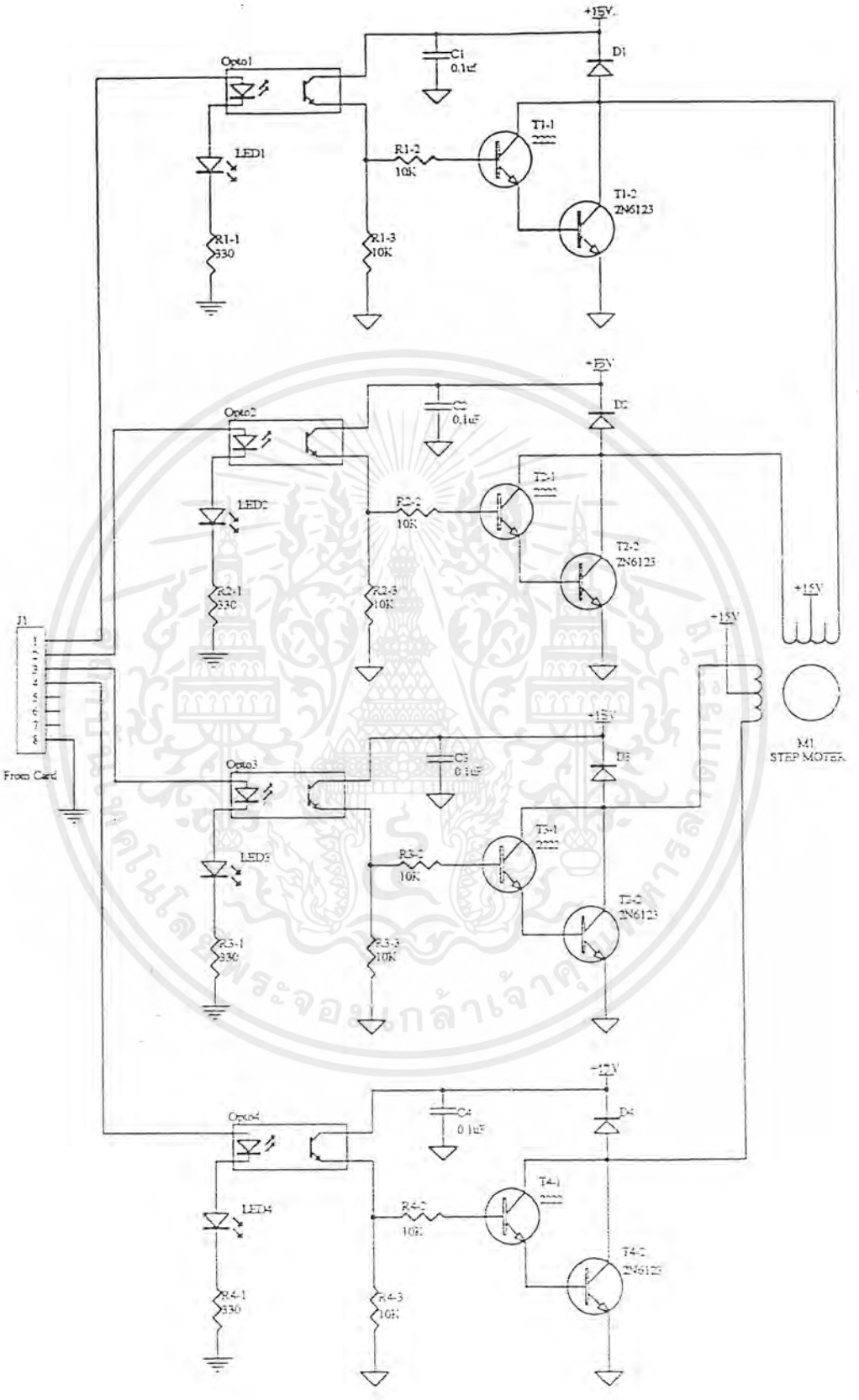


รูปที่ 6.4 รูปแสดงบล็อกไดอะแกรมการทำงานของวงจรการ์ดพอร์ทขนาน



รูปที่ 6.5 แสดงวงจรการ์ดพอร์ทขนาน

รูปที่ 6.6 รูปแสดงบล็อกไดอะแกรมการทำงานของวงจรขับสเตปมอเตอร์



รูปที่ 6.7 แสดงวงจรขับสเตปมอเตอร์

บทที่ 7
คณิตศาสตร์ในการสร้างโครงสร้าง 3 มิติ

7.1 Two- Dimensional Coordinate geometry

ในงานด้าน Computer Graphics นั้น รูปร่างและขนาดของวัตถุที่แสดงเป็น 2 มิติจะต้องใช้ระบบ Coordinate ซึ่งปกติจะใช้ x , y ในระบบ Cartesian Coordinate ดังนั้นการ Transform รูปแบบต่างๆในทางเรขาคณิต จึงได้ถูกนำไปใช้เป็นแบบจำลอง (Model) สำหรับการ Shift , Resize หรือ Reorient Coordinate เหล่านั้น

พื้นฐานของแบบจำลอง 2 มิตินี้ คือ จุด ยกตัวอย่างเช่น เส้นก็คือจุด 2 จุดต่อกัน หรือ ระนาบจะประกอบด้วยจุดหลายๆจุดที่อยู่บนระนาบ ดังนั้นในระบบ 2 มิติจะถูกกำหนดขึ้นด้วยกลุ่มของ Coordinate (x, y)หรือจุดใดๆนั่นเอง

เราจะทำการสมมุติรูป 3 เหลี่ยมในระนาบ 2 มิติซึ่งสามารถเขียนเป็นแมททริกซ์ ขนาด 3x2 ได้ดังนี้

$$[P] = \begin{bmatrix} x1 & y1 \\ x2 & y2 \\ x3 & y3 \end{bmatrix} \qquad (7.1)$$

ซึ่ง x และ y นี้เป็นเวคเตอร์ตำแหน่งที่มีความสัมพันธ์กับระบบ Coordinate แบบ Cartesian ถ้าเราใช้ระบบ Coordinate แบบ Cartesian แล้วเราจะพบกับข้อจำกัดในการ Transform ซึ่งในการ Transform นั้นจะต้องเกี่ยวข้องกับ Operation การคูณ ซึ่งมีข้อจำกัดในเรื่องขนาดของแมททริกซ์และเงื่อนไขของเวกเตอร์ ดังนั้นจึงหลีกเลี่ยงปัญหาเหล่านี้ด้วยการแปลงโดยใช้ระบบ Coordinate แบบ Homogeneous เพื่อความเข้าใจระบบ Coordinate แบบ Homogeneous แล้วเราจะสมมติจุด p(x,y) ให้อยู่ในระนาบ 3 มิติ ระยะห่างของจุด p(x,y) กับจุดกำเนิด (Origin) จะถูกแสดงโดยพารามิเตอร์ h ดังนั้นเราสามารถเขียนสมการได้คือ

$$P(x, y, z) = P(hx, hy, h) \qquad (7.2)$$

ซึ่งเราจะมองจุด p(x,y) นี้ในลักษณะของระนาบ 3 มิติโดย h ก็จะขึ้นกับค่าของ z นั่นเอง เพราะฉะนั้นจุด 2 มิติ ใดๆก็สามารถแสดงเป็นจุดในระนาบ 3 มิติได้ซึ่งเราเรียกระนาบนี้ว่า Homogeneous แต่ยกเว้นในกรณีที่จุดนั้นเป็นจุดกำเนิด (ซึ่งค่า h=0) ตัวอย่างเช่นเมื่อมีจุด P(2,4) ใน Coordinateคาร์เทเซียน จะทำให้เกิดจุด P(4,8,2) ,P(6,12,3) ,P(2,4,1) ,P(m,n,h) ใน Homogeneous Coordinate ดังนั้นถ้าเรามีจุดต่างๆ เหล่านี้จะทำให้เราสามารถหาจุดใน Cartesian Coordinate ได้คือ P(m/h ,n/h ,1)

โดยที่ $x = m/h$

$y = n/h$ (7.3)

เมื่อใช้ Homogeneous Coordinate แล้วค่าต่างๆ ในระนาบ 2 มิติ จะถูกแสดงในรูปของแมททริกซ์ ขนาด [n x 3] ซึ่ง n = จำนวนของจุดของวัตถุ ดังนั้น

$$[P] = \begin{bmatrix} x1 & y1 & 1 \\ x2 & y2 & 1 \\ x3 & y3 & 1 \end{bmatrix} \quad (7.4)$$

7.1.1) การ Scaling 2 มิติ

การทำ Scaling คือ การทำให้วัตถุมีขนาดที่เพิ่มขึ้น การScalingในทิศทาง x และ y ด้วยค่าคงที่นั้นจะทำให้เกิดการเปลี่ยนแปลงความยาวขึ้น ถ้ามีขนาดมากกว่า 1 จะเป็นการขยาย แต่ถ้าน้อยกว่า 1 จะเป็นการลดขนาด แต่ไม่ว่าจะเป็นกรณีใดจะต้องเป็นค่าบวก เพราะหากเป็นค่าลบแล้วจะเกิดการสะท้อนกลับ (Reflection) ขึ้น

การScaling จุด P(x,y) เป็น P(x*,y*) แสดงได้ดังสมการ (7.5)

$$x^* = x(Sx)$$

$$y^* = y(Sy) \quad (7.5)$$

หรือแสดงในรูปแมททริกซ์ได้ดังสมการ (7.6)

$$[x^* \; y^* \; 1] = [x \; y \; 1] \begin{bmatrix} Sx & 0 & 0 \\ 0 & Sy & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.6}$$

ในการ Scaling นี้เป็นการ Scaling ที่ใช้จุดกำเนิดเป็นจุดอ้างอิง ซึ่งแต่ละจุดบนวัตถุจะเปลี่ยนขนาดหรือทิศทางนั้นต้องขึ้นกับจุดกำเนิดเสมอ

7.1.2 การ Translation 2 มิติ

การ Translation นั้นจะทำให้วัตถุเปลี่ยนตำแหน่งไปในทิศทางตามค่าที่กำหนดไปซึ่งสามารถแสดงให้เห็นตามสมการ (7.7)

$$\begin{aligned} x^* &= x + Tx \\ Y^* &= y + Ty \end{aligned} \tag{7.7}$$

แสดงในรูปแมททริกซ์ ตามสมการ (7.8)

$$[x^* \; y^* \; 1] = [x \; y \; 1] \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ Tx & Ty & 0 \end{bmatrix} \tag{7.8}$$

7.1.3 การ Rotation 2 มิติ

การ Rotation นั้นจะใช้สำหรับการมองวัตถุในทิศทางต่างๆกัน ในการ Rotation จะต้องใช้จุดกำเนิด เป็นจุดอ้างอิงเช่นกัน ตามค่าของมุม θ ที่กำหนดไป ในการหมุน จุด P(x,y) เป็น P(x*,y*) ได้ดังสมการ (7.9)

$$\begin{aligned} x &= r\cos\alpha \\ y &= r\cos\alpha \end{aligned} \tag{7.9}$$

ซึ่งมุม α เป็นค่าพารามิเตอร์ที่แทนค่ามุมที่กวาดไปในทิศทางตามแกน x

$$\begin{aligned} x^* &= r\cos(\alpha+\theta) = r\cos\alpha\cos\theta - r\sin\alpha\sin\theta \\ y^* &= r\sin(\alpha+\theta) = r\sin\alpha\cos\theta + r\cos\alpha\sin\theta \end{aligned} \tag{7.10}$$

เมื่อแทนค่า x , y จากสมการ (7.9) จะได้สมการดังนี้

$$x^* = x\cos\theta - y\sin\theta$$
$$y^* = x\sin\theta + y\cos\theta \qquad (7.11)$$

หรือ แสดงในรูปแมททริกซ์ได้ดังสมการ (7.12)

$$[x^* \;\; y^* \;\; 1] = [x \;\; y \;\; 1] \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \qquad (7.12)$$

7.2 Three - Dimentional Coordinate Geometry

ในการออกแบบ (Design) งานทางด้านวิศวกรรมส่วนมากจะเกี่ยวข้องกับวัตถุที่เป็น 3 มิติ โดยปรกติรูปทรงหรือแบบจำนวนมากจะถูกใช้งานในการสร้างรูปทรง 3 มิติ ซึ่งสามารถแสดงออกมาบนหน้าจอมอนิเตอร์ เช่น Polygon Surface หรือ Framework ของเส้น สำหรับการใช้งานด้านอื่นๆเช่น การออกแบบเครื่องจักรกลหรือตัวโครงของเครื่องบิน เป็นการออกแบบที่พิเศษซึ่งถูกใช้เพื่อพัฒนารูปแบบของพื้นผิว 3 มิติ จนกระทั่งได้เงื่อนไขเป็นที่น่าพอใจ

เมื่อใดที่วิธีการเหล่านี้ถูกใช้งาน วัตถุ 3 มิติก็จะถูกสร้างขึ้นในระบบCoordinate3 มิติ และจะทำการ Map ไปในระบบ 2 มิติเพื่อที่จะทำการแสดงผล การสร้างภาพของวัตถุจำเป็นต้องใช้การ Transform Coordinate 3 มิติ ซึ่งแสดงได้โดยแมททริกซ์ 4x4 ดังนี้

$$[P] = \begin{bmatrix} 0 & 0 & 0 & 1 \\ x_n & 0 & 0 & 1 \\ 0 & y_c & 0 & 1 \\ 0 & 0 & z_d & 1 \end{bmatrix} \qquad (7.13)$$

ในการแปลงในระบบ 3 มิติ จะถูกใช้ด้วยวิธีการเดียวกับที่ใช้ในระบบ 2 มิติดังที่อธิบายในส่วนการแปลงในระบบ 2 มิติ ข้างต้น แต่จะเพิ่มCoordinateของ z เข้ามาด้วย ซึ่งในระบบ 2 มิติจะประกอบด้วยCoordinate x และ y เท่านั้น

ในระบบ Homogeneous Coordinate จุดต่างๆในระนาบ 3 มิติก็จะถูกแสดงเป็นแมททริกซ์ขนาด 4x4 ดังนี้

$$\begin{bmatrix} a & b & c & 0 \\ d & e & f & 0 \\ g & h & I & 0 \\ j & k & l & s \end{bmatrix} \qquad (7.14)$$

7.2.2) การ scaling 3 มิติ

ในการทำการ Scaling Transform จะทำได้โดยการใส่ค่าไปในแนวทแยงมุมของแมททริกซ์ที่ใช้ในการแปลงขนาด 4x4 ดังเช่นการ Scaling จุด P(x,y,z,1) ไปเป็น P(x*,y*,z*,1)ได้สมการดังนี้

$$[x^*,y^*,z^*,1] = [x,y,z,1] \begin{bmatrix} a & 0 & 0 & 0 \\ 0 & e & 0 & 0 \\ 0 & 0 & i & 0 \\ 0 & 0 & 0 & l \end{bmatrix} \qquad (7.15)$$

ด้วยวิธการนี้เราจะทำการ Scaling โดยใช้จุดกำเนิดเป็นจุดอ้างอิงและใช้หลักการเดียวกับการทำ Scaling ใน 2 มิติ ถ้าตัวแปรในการทำ Scaling คือ a , e , s มีค่าที่ไม่เท่ากันแล้วทำให้ภาพและขนาดของวัตถุที่ได้จะผิดเพี้ยนไปแม้ จะเกิดการเปลี่ยนแปลงขนาดเกิดขึ้นแต่จุดกำเนิดยังเหมือนเดิม

$$[x^*,y^*,z^*,1] = [x,y,z,1] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & s \end{bmatrix}$$

$$= [x,y,z,s] \qquad (7.16)$$

ด้วยวิธีการนดังกล่าว เมื่อเราใช้การ Scaling วิธีนี้แล้วจะสังเกตเห็นว่าในสดมภ์ (Column) ที่ 4 ของแมททริกซ์ตรงจุดที่ใช้การแปลงอาจจะไม่เท่ากับ 1 ก็ได้ ดังเช่นหลักการของ 2 มิติ ที่ได้กล่าวแล้วข้างต้น ซึ่งแมททริกซ์ที่ใช้นี้อาจถูกเปลี่ยนแปลงได้เพื่อให้ค่าของ x , y , z นี้เป็นค่า Coordinate ที่เป็นมาตรฐาน ซึ่งในระบบ Cartesian Coordinate ดังนั้นสมการ (7.16) จะเขียนได้ดังนี้

$$[x,y,z,s] \quad = \quad [x/s\ ,\ y/s\ ,\ z/s\ ,1] \tag{7.17}$$

เมื่อค่า s มีค่ามากกว่า 1 จะเป็นการลดขนาดของวัตถุลง แต่เมื่อต้องการที่ขยายขนาดของวัตถุต้องใช้ค่า 1/s ในแมททริกซ์ที่ทำ Scaling ซึ่งก็จะได้ค่าของ Coordinate สุดท้ายเป็น

$$[sx\ ,\ sy\ ,\ sz\ ,\ 1\ ] \tag{7.18}$$

7.2.3) การ Translation 3 มิติ

จากสมการ (7.18) แมททริกซ์ที่ใช้ในการ Transform นี้จะทำการ Translate หรือเคลื่อนย้ายจุด (x,y,z) เป็นจุดใหม่ (x*, y*, z*) โดยใช้ (j,k,l)

$$[\,x^*\ ,y^*\ ,z^*\ ,1\,] \quad = \quad [\,x\ ,y\ ,z\ ,1\,] \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ j & k & l & 1 \end{bmatrix} \tag{7.19}$$

ค่าของ j , k , l นี้ จะแสดงความสัมพันธ์ของการเคลื่อนย้ายจุดไปตามทิศทาง ของ x ,y ,z ยกตัวอย่าง เช่น การแปลงรูปลูกบาศก์ขนาด 1 หน่วย ซึ่งมีจุดยอดจุดหนึ่งอยู่ที่ตำแหน่งจุดกำเนิดไปที่ตำแหน่งใหม่ในระนาบที่พิจารณา โดยใช้แมททริกซ์ที่ใช้สำหรับการทำการแปลงดังสมการ (7.19) แต่ละจุดยอดจะถูกแปลงไปตามค่าที่กำหนดในแมททริกซ์ ดังนั้นค่า Coordinate ที่ถูกแปลงของลูกบาศก์ นี้จะมีค่าดังสมการ (7.20)

$$[P^*]_{CUBE} = \begin{bmatrix} 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 1 \\ 1 & 0 & 1 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 \\ 0 & 1 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ J & K & L & 1 \end{bmatrix} = \begin{bmatrix} 2 & 2 & 0 & 1 \\ 3 & 2 & 0 & 1 \\ 3 & 2 & 1 & 1 \\ 3 & 2 & 1 & 1 \\ 2 & 3 & 1 & 1 \\ 3 & 3 & 1 & 1 \\ 3 & 3 & 0 & 1 \\ 2 & 3 & 0 & 1 \end{bmatrix} \quad (7.20)$$

7.2.4) การ Rotation 3 มิติ

การ Rotation ในระบบ 3 มิติเป็นเรื่องที่สำคัญสำหรับการทำความเข้าใจรูปร่างของวัตถุ หรือการพิจารณางานที่ถูกออกแบบมาในมุมที่แตกต่างกันซึ่งเป็นสิ่งที่ซับซ้อนมากกว่าการ หมุนในระบบ 2 มิติ เพราะในระบบ 3 มิติต้องระบุแกนในการทำหมุนด้วยในขณะที่ระบบ 2 มิติใช้เพียงจุดในการทำหมุนเท่านั้น ในการทำการหมุนตามแกนใดๆก็ตาม เราสามารถที่จะคิดอย่างง่ายๆโดยคิดแยกเป็นการหมุนตามแกนหลักทั้ง 3 แกนคือ แกน x , y , และ z แล้วนำมาสร้างตามวิธีการเดียวกันกับที่ใช้ในการหมุน 2 มิติ ทำให้เราได้แมททริกซ์สำหรับการหมุนตามแกน z ดังสมการ (7.21)

$$[T_R]_Z^{\theta} = \begin{bmatrix} \cos\theta & \sin\theta & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7.23)$$

ซึ่งจะได้ผลลัพธ์ของการแมปปิ้ง (Mapping) ดังสมการ (7.22)

$$\begin{aligned} x^* &= x\cos\theta - y\sin\theta \\ y^* &= x\sin\theta + y\cos\theta \\ z^* &= z \end{aligned} \quad (7.22)$$

ในวิธีการเดียวกันเมื่อเราทำการหมุนเป็นมุม θ ไปตามแกน y จะได้สมการ (7.23)

$$[T_R]_Y^{\theta} = \begin{bmatrix} \cos\theta & 0 & -\sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7.23)$$

ซึ่งจะได้ผลลัพธ์ของการ Mapping ดังสมการ (7.24)

$$\begin{aligned} x^* &= x\cos\theta + z\sin\theta \\ y^* &= y \\ z^* &= -x\sin\theta + z\cos\theta \end{aligned} \quad (7.24)$$

เมื่อหมุนไปตามแกน x จะได้ผลดังสมการ (7.25)

$$[T_R]_X^{\theta} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (7.25)$$

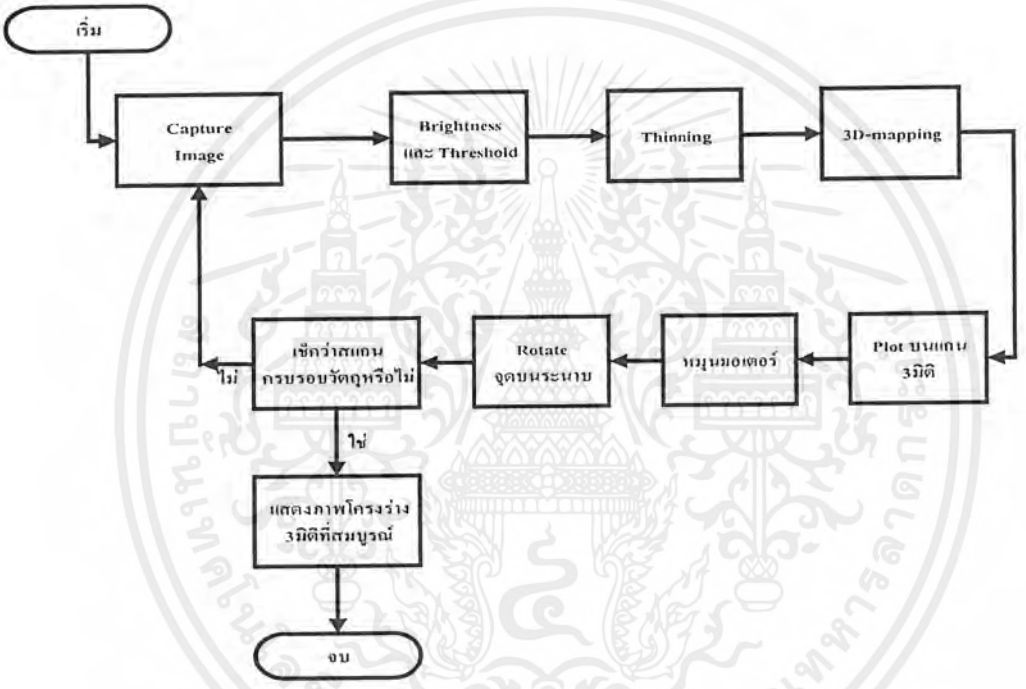
ซึ่งจะได้ผลลัพธ์ของการ Mapping ดังสมการ (7.26)

$$\begin{aligned} x^* &= x \\ y^* &= y\cos\theta - z\sin\theta \\ z^* &= y\sin\theta + z\cos\theta \end{aligned} \quad (7.26)$$

ข้อสังเกตที่สำคัญคือเครื่องหมายในเทอม Sine ในแมททริกซ์ $[T_R]_Y^{\theta}$ ซึ่งจะมีค่าตรงกันข้ามกับแมททริกซ์ $[T_R]_Z^{\theta}$, $[T_R]_X^{\theta}$ เนื่องจากผลลัพธ์ของการใช้กฎมือขวา

บทที่ 8
โปรแกรมควบคุมการทำงานของเครื่องเก็บข้อมูลภาพ 3 มิติ

8.1 โครงสร้างการทำงานของเครื่องเก็บข้อมูลภาพ 3 มิติ



รูปที่ 8.1 บล็อกไดอะแกรมแสดงการทำงานของเครื่องเก็บข้อมูลภาพ 3 มิติ

การทำงานของ เครื่องเก็บข้อมูลภาพ 3 มิติในแต่ละขั้นตอนหลักๆมีดังนี้

1) Capture Image

คือขั้นตอนการรับภาพจากกล้องวีดีโอมาเก็บไว้ในหน่วยความจำเพื่อไปทำการประมวลผลในขั้นต่อไป

2) Brightness & Threshold

เป็นการนำภาพที่ได้จากขั้นตอนแรก ซึ่งเป็นภาพสีจริงมาแปลงให้เป็นภาพที่อยู่ในรูปแบบของความสว่าง แล้วทำการ Threshold คือ เลือกเอาเฉพาะส่วนที่มีความสว่างอยู่ในช่วงที่เราต้องการ

3) Thinning

เป็นการนำภาพที่ได้จากการ Threshold มาทำให้ส่วนที่เลือกมาบางลงเพื่อให้ง่ายต่อการประมวลผลต่อไป

4) 3D-Mapping

เป็นการนำภาพที่ได้จากการ Thinning แล้วมาทำการเลือกเฉพาะจุดต่างๆที่ทำ Thinning แล้วนำ พิกัดของจุดเหล่านั้นมาทำการแปลงจากรูปแบบ 2 มิติให้เป็น 3 มิติ

5) การ Plot บนแกน 3 มิติ

คือการนำเอาจุดที่ได้จากขั้นตอน 3D-Mapping มาทำการวาด (Plot) บนแกนจำลอง 3 มิติ ที่จะแสดงบนหน้าจอมอนิเตอร์

6) หมุนมอเตอร์

เมื่อทำการประมวลผลภาพและ Plot บนแกน 3 มิติ เรียบร้อยแล้วต้องทำการหมุนมอเตอร์เพื่อไปรับภาพของวัตถุจากกล้องในสเตปต่อไป จนครบรอบ

7) Rotate จุดบนระนาบ

โดยจุดที่ Plot บนแกน 3 มิตินี้เป็นจุดที่อยู่บนระนาบอ้างอิงซึ่งกล้องยังไม่ได้หมุนไปตามมุมต่างๆ และเมื่อกล้องหมุนไปที่มุมต่างๆเพื่อรับภาพวัตถุนั้น ระนาบของภาพที่รับมาจึงต้องหมุนด้วยเพื่อเป็นระนาบจริง มิฉะนั้นภาพที่ Plot ออกมาที่มุมต่างๆ จะเกิดการซ้อนทับกันที่ตำแหน่งระนาบอ้างอิง

8) เช็คการครบรอบ

ถ้ายังไม่ครบก็วนกลับไปทำซ้ำอีกครั้งโดยที่เริ่มที่การ Capture Image ถ้าครบแล้วก็จะเข้าสู่ขั้นตอนสุดท้ายของการทำงาน

9) แสดงภาพโครงร่าง 3 มิติ ที่สมบูรณ์

เป็นขั้นตอนสุดท้ายของการทำงานโดยจะแสดงภาพที่ได้จาก Plot Coordinateทั้งหมดจากการสแกนจนครบรอบวัตถุดังนั้นภาพที่ได้ในขั้นตอนนี้จึงเป็นภาพที่สมบูรณ์ใกล้เคียงกับวัตถุจริง

8.2 โครงสร้างโปรแกรม

ในส่วนของโปรแกรมที่ควบคุมการทำงานของเครื่องเครื่องเก็บข้อมูลภาพ 3 มิตินี้จะแบ่งการควบคุมออกเป็น 2 ส่วนใหญ่ๆคือ

1. ส่วนของโปรแกรมที่ควบคุมฮาร์ดแวร์ภายนอก คือ ควบคุมการหมุนของ สเตปมอเตอร์ ซึ่งจะควบคุมผ่านทางการ์ดวงจรพอร์ทขนานที่ต่อเพิ่มเติมเข้ามา โดยมีรูปแบบและวงจรการทำงาน

ตามที่ได้กล่าวมาแล้ว รูปแบบของคำสั่งจะเป็นการส่งค่าลอจิก 1/0 ออกไปทางพอร์ตเพื่อ On / Off ทรานซิสเตอร์ที่ต่ออยู่กับแต่ละเฟสของสเตปมอเตอร์ เพื่อเหนี่ยวนำให้เกิดการหมุนตามที่ต้องการ

2. ส่วนของโปรแกรมการ Capture และประมวลผล เป็นโปรแกรมการรับภาพจากกล้อง Color Quick Cam มาแสดงผลที่หน้าจอมอนิเตอร์โดยจะทำงานสัมพันธ์กับโปรแกรมควบคุมการหมุนของสเตปมอเตอร์ คือ ในแต่ละตำแหน่งที่จะทำการ Capture จะต้องรอให้กล้องอยู่นิ่งก่อน (ซึ่งการหมุนของขากล้องถูกควบคุมจากสเตปมอเตอร์) เมื่อกล้องหยุดนิ่งแล้วก็จะทำการเก็บภาพ ณ ตำแหน่งนั้นเพื่อนำไปประมวลผลแล้วแสดงผลมาที่จอมอนิเตอร์ทันที จากนั้นก็ควบคุมให้สเตปมอเตอร์หมุนเพื่อเปลี่ยนตำแหน่งในการ Capture , ประมวลผล และแสดงผลต่อไปเรื่อยๆจนวนครบรอบวัตถุ

ภาษาที่ใช้เขียนโปรแกรมนี้จะใช้ MFC VISUAL C++ ซึ่งเป็นภาษาที่ประมวลผล (Run) บน Window ทำให้ไม่มีปัญหาเรื่องหน่วยความจำไม่เพียงพอ เนื่องจากการประมวลผลเกี่ยวกับรูปภาพเป็นกระบวนการที่ต้องใช้หน่วยความจำมากพอสมควร

โครงสร้างการทำงานของโปรแกรมควบคุมนี้จะอยู่บนพื้นฐานของ Interrupt Service Routine 2 อย่างคือ

1. Interrupt Service Routine ของเวลา ซึ่งเป็นลักษณะของการนับเวลาจนครบ เมื่อเวลาครบแล้วไม่ว่าโปรแกรมจะทำตรงไหนอยู่ก็ตามมันจะกระโดดไปทำงานที่โปรแกรมบริการอินเตอร์รัปต์ชนิดนี้ ก็คือ ส่วนของฟังก์ชั่น OnTimer

2. Interrupt Service Routine ของการรับ Frame ภาพใหม่เข้ามา ซึ่งคล้ายกับ Interrupt Service Routine ของเวลา แต่โปรแกรมจะกระโดดไปทำตรงส่วนบริการอินเตอร์รัปต์ก็ต่อเมื่อมี Frame ของภาพเข้ามาใหม่ โปรแกรมบริการอินเตอร์รัปต์ชนิดนี้ก็คือ ฟังก์ชั่น OnFrameCallback

การรับ Frame ของภาพมานั้น นอกจากจะได้ข้อมูลภาพจริงๆมาแล้ว เรายังได้รายละเอียดเกี่ยวกับภาพด้วย ตัวอย่างเช่น

-ความสูงของภาพ ในหน่วย Pixel

-ความกว้างของภาพ ในหน่วย Pixel

-ความละเอียดของภาพ ในหน่วย Bit/Pixel

ซึ่งรายละเอียดต่างๆเหล่านี้ จะทำให้สามารถสร้าง Bitmap file ขึ้นมาใหม่ได้ แล้วนำข้อมูลภาพเหล่านั้นมาทำการประมวลผลซึ่งในส่วนนี้จะเป็นขั้นตอนที่อยู่ในโปรแกรมย่อยของ OnFrameCallback ทั้งสิ้น การประมวลผลภาพในแต่ละ frame จะถูกแบ่งออกได้ตามขั้นตอนดังนี้คือ

2.1) Display Capture Bitmap เป็นการแสดงภาพนิ่งที่ได้จากการถ่ายภาพวัตถุในแต่ละมุม โดยการสร้าง Bitmap file ขึ้นมาแล้วคัดลอกข้อมูลภาพจากกล้องมาไว้ที่ไฟล์นั้น หลักการทางโปรแกรมในการรับจากกล้องวีดีโอและเก็บภาพมาแสดงบนหน้าจอเป็นไปตาม แผนภาพ ในรูปที่ 8.4 และ 8.5

2.2) Threshold Bitmap จากการออกแบบในส่วนของเครื่องเก็บข้อมูลภาพ 3 มิตินี้จะมีข้อจำกัดอยู่บางประการคือ จะต้องทำการสแกนวัตถุในที่มืด ซึ่งอันที่จริงแล้วไม่จำเป็นต้องทำเช่นนี้ก็ได้เพราะวิธีการที่จะเลือกส่วนที่ต้องการจากภาพนั้นอาจใช้วิธีการทำ Edge detection, Gradient, Laplacian และอื่นๆ อีกหลายวิธี โดยไม่ต้องทำในที่มืดก็ได้ แต่วิธีต่างๆ เหล่านั้นจะยุ่งยากมาก เพราะต้องอาศัยการพิจารณาจาก Pixel รอบๆด้วยเพื่อดูความแตกต่างดังนั้นจึงเลือกใช้วิธีการทำ Threshold เพราะเป็นการตรวจสอบทีละ Pixel เทียบกับค่าคงที่ที่เราต้องการ ซึ่งในที่นี้ก็คือค่าความสว่างที่เราต้องการนั่นเอง เพราะฉะนั้นการเขียนโปรแกรมจะง่ายขึ้นมาก ถ้าเราทำให้พื้นที่ทดลองมืด ซึ่งจะทำให้สามารถหาแนวสันเลเซอร์ได้ง่าย ไม่ซับซ้อนเพราะภาพจะมีเพียงส่วนของเส้นเลเซอร์เท่านั้นที่สว่างชัดเจน ดังนั้นจึงต้องมีข้อจำกัดตรงส่วนนี้ด้วย ขั้นตอนของการ Threshold นี้จะเป็นการตรวจหาภาพในส่วนที่เป็นแสงเลเซอร์เท่านั้น ซึ่งจะใช้การตรวจหาโดยดูจากความสว่างของภาพในแต่ละ Pixel ตามสูตร

$$Brightness = (R+G+B) / 3$$

Brightness : ความสว่างของแต่ละ Pixel

R : Red Component เป็นองค์ประกอบสีแดงในแต่ละ Pixel

G : Green Component เป็นองค์ประกอบสีเขียวในแต่ละ Pixel

B : Blue Component เป็นองค์ประกอบสีน้ำเงินในแต่ละ Pixel

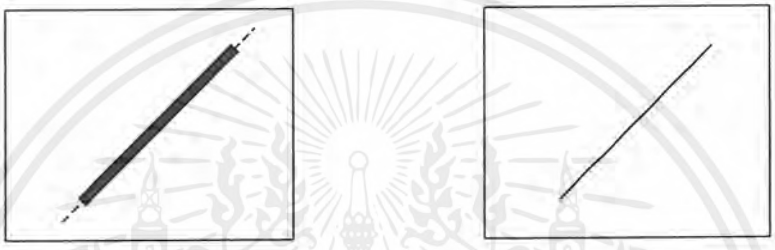
หลักการทางโปรแกรมในการเปลี่ยนภาพให้อยู่ในรูปแบบของความสว่างเป็นไปตามแผนภาพในรูปที่ 8.6

หลังจากการคำนวณความสว่างของแต่ละ Pixel มาแล้วก็เลือกตรวจสอบเฉพาะส่วนที่มีความสว่างมากซึ่งจะเป็นในส่วนของเส้นเลเซอร์ที่ต้องการ ทำให้ได้ภาพที่เป็น Threshold Bitmap ออกมาแสดงที่หน้าจอมอร์นิเตอร์ ซึ่งจะแสดงตรงส่วนที่ตรวจพบเลเซอร์ได้เป็นสีขาวและส่วนอื่นเป็นสีดำ

หลักการทางโปรแกรมในการ Threshold เอาเฉพาะส่วนที่ต้องการเป็นไปตามแผนภาพ ในรูปที่ 8.7

2.3) Thinning bitmap เป็นขั้นตอนที่ทำหลังจาก Threshold Bitmap ออกมาได้แล้วซึ่งหลักการในขั้นตอนนี้ ก็คือ จะทำให้เส้นเลเซอร์ที่ตรวจสอบได้นั้นบางลง เพื่อให้ได้เส้นที่คมชัดและแน่-

นอน โดยตามโปรแกรมนี้จะ Thinning ให้เหลือความบางของเส้นเพียง 1 Pixel นั่นคือถ้าหากเจอแถบสีขาวของเลเซอร์ในแนวนอนก็จะเลือกเอา Pixel ที่อยู่ตรงกลาง หรือถ้าเจอเพียง 1 Pixel ก็ให้เลือก Pixel นั้นเลย ทำตามขั้นตอนนี้ไปจนครบแถวในแนวนอนก็จะทำให้ได้ Thinning Bitmap ออกมาแสดงที่หน้าจอมอร์นิเตอร์ ทางด้านมุมล่างขวา โดย Pixel ที่ได้จากการ Thinning แล้วจะเป็นสีขาว และส่วนอื่นเป็นสีดำ ส่วนรูปที่ 8.2 จะแสดงให้เห็นถึง Algorithm ของการทำ Thinning อย่างชัดเจน ส่วนหลักการทางโปรแกรม Thinning เป็นไปตามแผนภาพในรูปที่ 8.8



ก) ข)

รูปที่ 8.2 แสดง Algorithm ของการทำ Thinning

ก) ก่อนการทำ Thinning ข) หลังการทำ Thinning

2.4) Mapping Coordinate เป็นส่วนของการนำจุดหรือ Pixel ที่ได้จากการ Thinning แล้วไปทำการคำนวณในเชิงกราฟฟิก 3 มิติ เนื่องจากจุดที่ได้จากภาพนั้นเป็นการรับมาจากกล้องที่ได้ตั้งในลักษณะได้ฉากกับแนวลำแสงของเลเซอร์ และรวมถึงระยะจากวัตถุมายังส่วนที่รับภาพคือกล้องนั่นเองดังนั้นจึงต้องมีการ Mapping ให้เป็นพิกัดที่ถูกต้องในการมองเห็นจริง หลักการในการ Keep Coordinate ของข้อมูลภาพจาก Thinning Bitmap เป็นไปตามแผนภาพในรูปที่ 8.8

ส่วนการ Mapping Coordinate จาก 2 มิติ ไปเป็น 3 มิติ นั้นจะเป็นขั้นตอนก่อนที่จะแสดงผลภาพบนแกนจำลอง 3 มิติ ซึ่งหลักการทางคณิตศาสตร์ในการ Mapping ได้แสดงไว้ในหัวข้อ 8.3

2.5) โครงร่างบนแกนจำลอง 3 มิติ เป็นการนำจุดที่ได้จากภาพที่ทำการ Mapping Coordinate แล้วมาวาดบนแกนจำลอง 3 มิติ แล้วแสดงผลบนจอมอร์นิเตอร์ หลักการ วาดภาพ 3 มิติ เป็นไปตามแผนภาพในรูปที่ 8.10

ทั้ง 5 ขั้นตอนนี้จะทำทุกครั้งที่มีการรับ Frame ภาพใหม่เข้ามา ดังนั้นจึงสามารถแสดงให้เห็นถึงการประมวลผลภาพทุกๆภาพได้บนหน้าจอมอร์นิเตอร์ หลังจากนั้นโปรแกรมก็จะถูกตั้งเวลาแล้วให้มีการเรียกโปรแกรมอินเตอร์รัปของเวลาอีกครั้งสลับกับอินเตอร์รัปของการรับเฟรมภาพจน

กระทั่งจนเก็บภาพจนครบรอบวัตถุแล้ว ก็จะนำผลรวมของการประมวลผลทั้งหมดมาแสดงอีกครั้งในลักษณะโครงร่างของวัตถุบนแกนจำลอง 3 มิติ

หลักการทำงานของโปรแกรมประมวลภาพ 3 มิติ

1. โปรแกรมส่วน Capture ภาพ

คือการรับข้อมูลภาพจากกล้องวีดีโอที่ทำให้นิ่งแล้วเข้ามา โดยข้อมูลภาพจะประกอบด้วยความกว้าง ,ความสูง และจำนวน Bit/Pixel และทำการจองหน่วยความจำเพื่อใช้กีบข้อมูลภาพหลังจากหยุดภาพแล้วก็นำเอาข้อมูลที่ได้จาก Capture มาเก็บลงในหน่วยความจำที่ได้จองไว้

2 โปรแกรมส่วน Threshold

คือการนำข้อมูลที่ได้จากการ Capture ซึ่งเป็นภาพสีมาทำการเปลี่ยนให้อยู่ในรูปแบบของความสว่าง โดยในขั้นตอนแรกจะทำการนำค่าองค์ประกอบของสี (Red ,Green ,Blue)ในแต่ละ Pixel มาเปลี่ยนให้อยู่ในรูปของความสว่าง (Brightness) หลังจากนั้นจะทำการ Threshold โดยเลือกช่วง Brightness ที่ต้องการแล้วทำการเปลี่ยน Pixel ให้เป็นสีขาว หรือสีดำ ตามค่าของ Brightness ที่ตั้งไว้โดยจะเป็นสีดำเมื่อค่าของความสว่างอยู่ในช่วงที่เราตั้งไว้นอกนั้นจะเป็นสีขาว

3 โปรแกรมส่วน Thinning

ขนาดของเส้นจากภาพที่ได้จากการทำ Threshold ซึ่งได้จากการตรวจจับระนาบเลเซอร์ยังมีขนาดหนาอยู่ซึ่งจะทำให้มีขนาดบางลงได้ โดยทำการเลือกจุดที่อยู่กึ่งกลางของแถบสีขาวในแนวนอน (1Pixel) ซึ่งได้จากขั้นตอนการทำ Threshold ให้คงค่าสีขาวไว้และทำการเปลี่ยน Pixel ที่อยู่ข้างๆไปเป็นสีดำ โดยเริ่มทำจากแถวบนสุดของภาพลงมาจนครบแถวในแนวนอน หลังจากนั้นจะทำการนับจำนวนจุดสีขาวที่ได้จากการทำ Thinning เพื่อที่จะได้นำไปคำนวนต่อไป

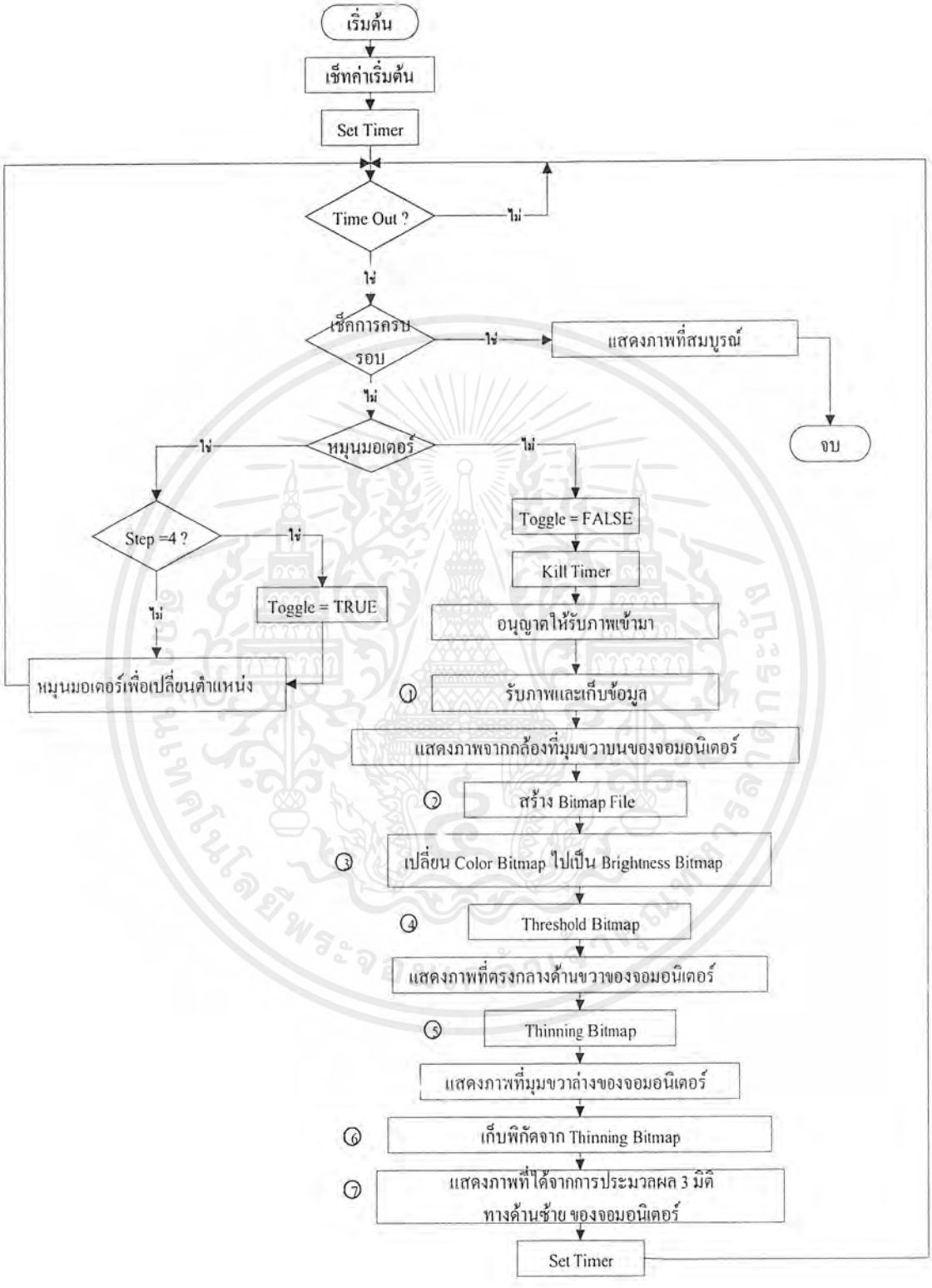
4 โปรแกรมส่วน Mapping Coordinate

โดยนำข้อมูลที่ได้จากการ Thinning จะนำมาแปลงให้อยู่ในรูปพิกัด2 มิติโดยขั้นตอนแรกทำการ Translate จุดกำเนิดที่ได้จากการ Thinning ซึ่งอยู่มุมบนซ้ายของภาพ ไปอยู่ที่กึ่งกลางของภาพ ต่อจากนั้นทำการแปลงข้อมูลแบบ 2 มิติไปเป็นข้อมูลแบบ 3 มิติโดยใช้สมการคณิตศาสตร์หาค่าพิกัดจริงของวัตถุออกมาตามหลักการการสร้างโครงร่าง 3 มิติตามหัวข้อ 8.3.1 ซึ่งจะทำให้ได้พิกัดของแนวเลเซอร์บนวัตถุออกมา แต่ค่าพิกัดที่ได้จากการคำนวณข้างต้นเป็นพิกัดในระนาบอ้างอิงซึ่งยังไม่ได้หมุนกล้อง แต่เมื่อกล้องหมุนไปเป็นมุมต่างๆเพื่อทำการรับภาพรอบวัตถุนั้น ระนาบของภาพที่รับมาจึงต้องหมุนไปตามมุมที่กล้องหมุนด้วยเพื่อเป็นระนาบจริง มิฉะนั้นภาพที่แสดงออกมาจะซ้อนทับกัน ซึ่งทำได้โดยใช้หลักการหาตำแหน่งของวัตถุในระนาบจริง ตามหัวข้อ 8.3.2

5 โปรแกรมส่วน การวาดโครงร่างบนแกน 3 มิติ

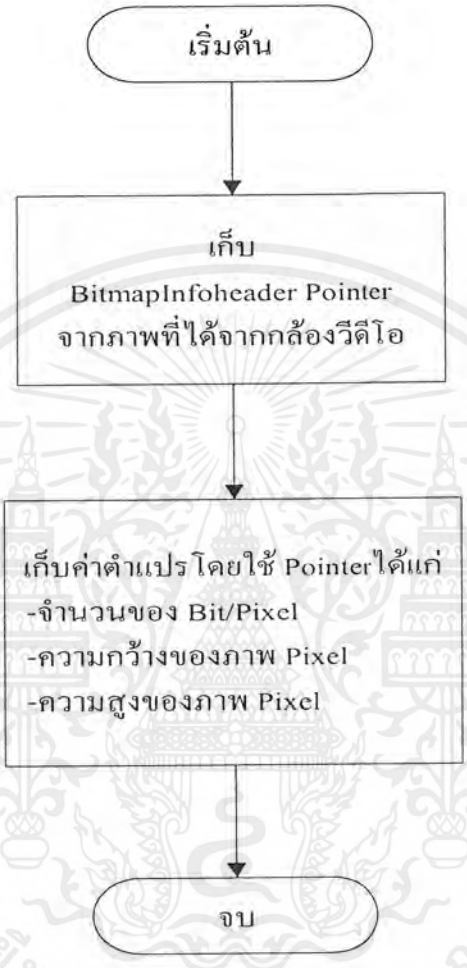
หลังจากขั้นตอนการทำ Mapping CoOrdinate แล้ว จากนั้นก็เป็นขั้นตอนการแสดงภาพซึ่งในที่นี้เราใช้ Class CDataScattering Space ในการทำ Bubble Render โดยใช้ฟังก์ชัน AddSphere เป็นฟังก์ชันที่ใช้ในการเก็บพิกัดที่จะแสดงผล

การทำงานของโปรแกรมควบคุมฮาร์ดแวร์ภายนอกและโปรแกรมการเก็บภาพและประมวลผลจะทำงานอย่างสัมพันธ์กันตามโครงสร้างโปรแกรม และแผนภาพ ดังรูปที่ 8.3



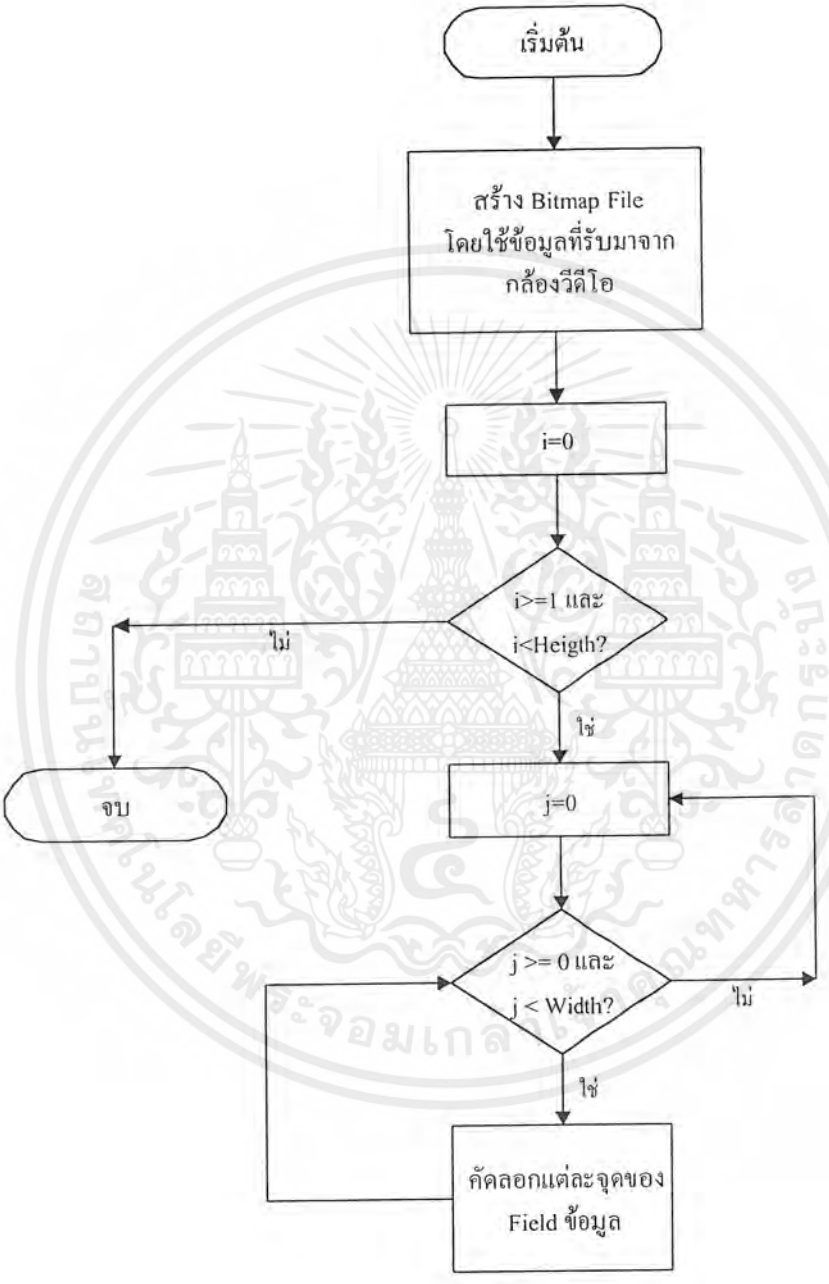
รูปที่ 8.3 แผนภาพแสดงโครงสร้างโดยรวมของโปรแกรม

1. รับภาพและเก็บข้อมูลภาพจากกล้องวีดีโอ



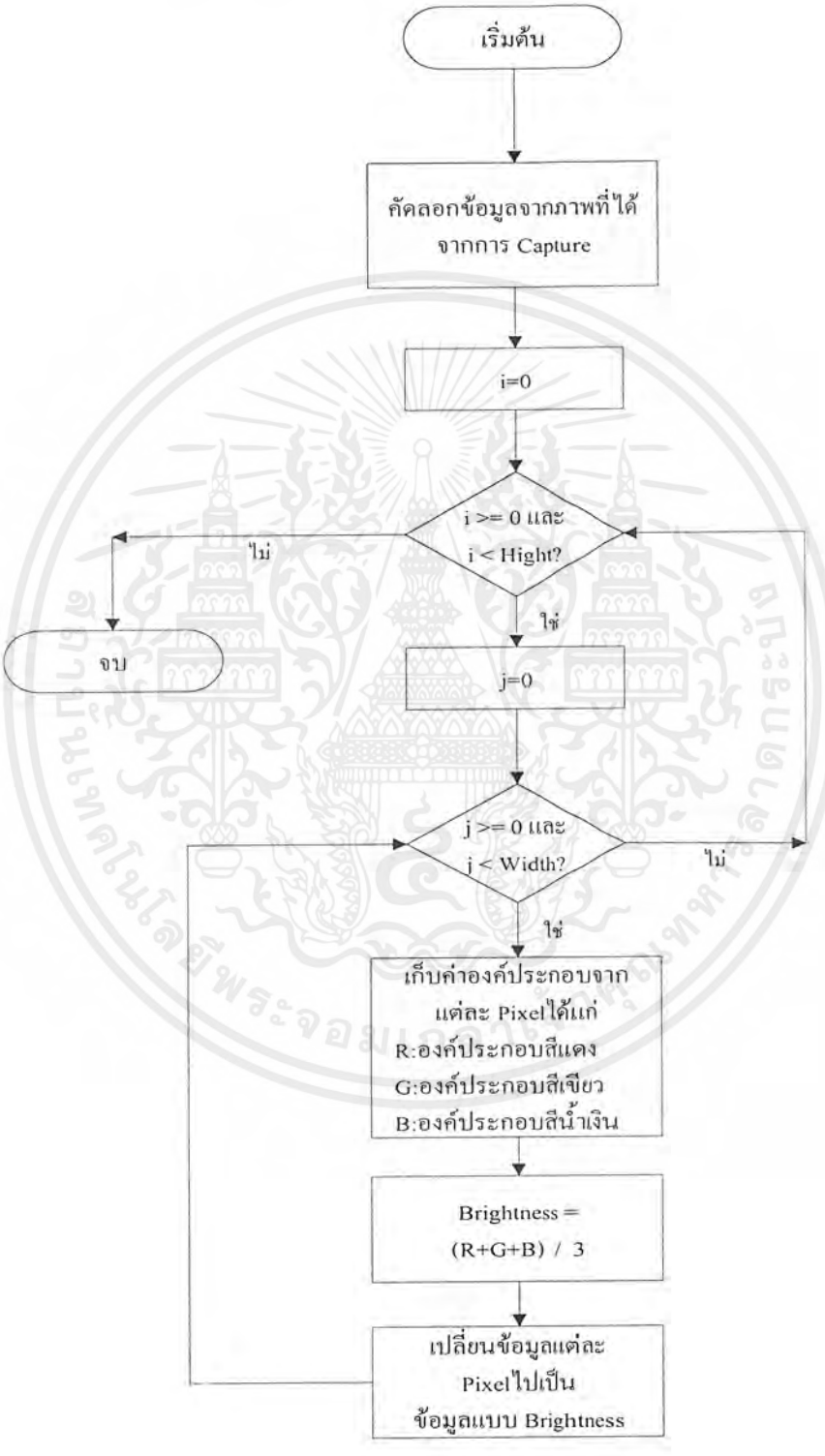
รูปที่ 8.4 แผนภาพแสดงการรับภาพและข้อมูลภาพจากกล้องวีดีโอ

2. การสร้าง Bitmap File



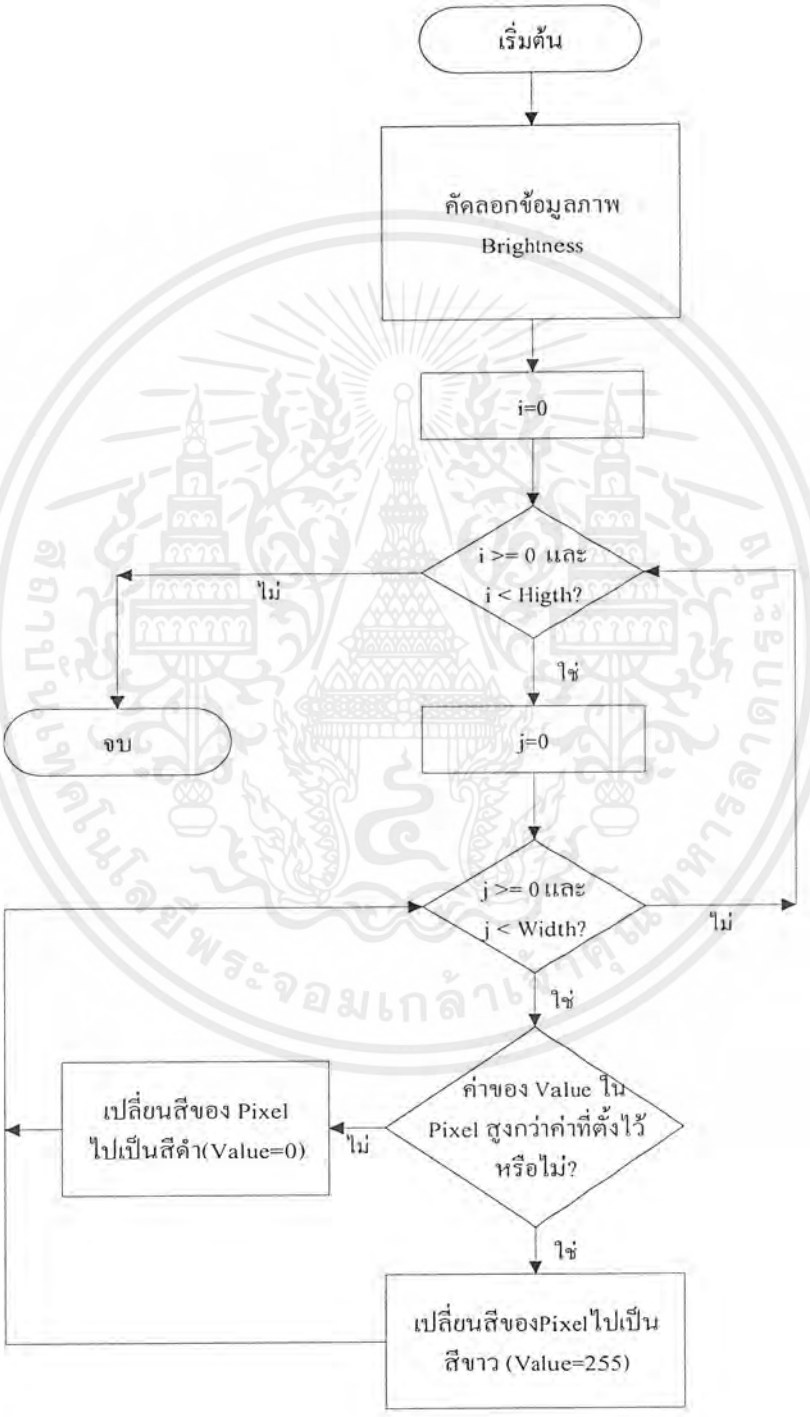
รูปที่ 8.5 แผนภาพแสดงการสร้าง Bitmap file จากภาพที่ Capture

3. เปลี่ยน Color Bitmap ไปเป็น Brightness Bitmap



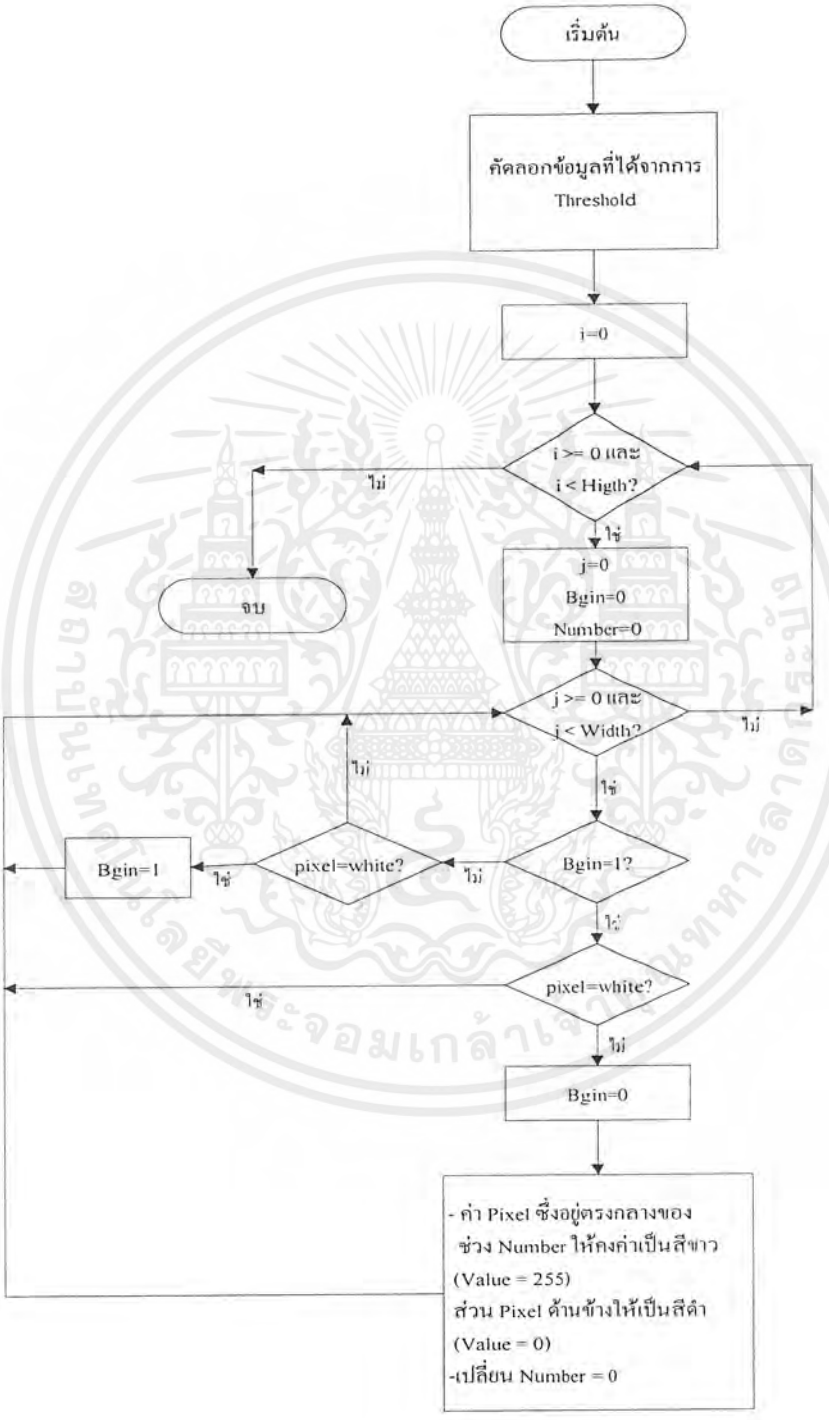
รูปที่ 8.6 แผนภาพแสดงการแปลง (Convert) ภาพจาก Color bmp เป็น Brightness bmp

4. Threshold Bitmap



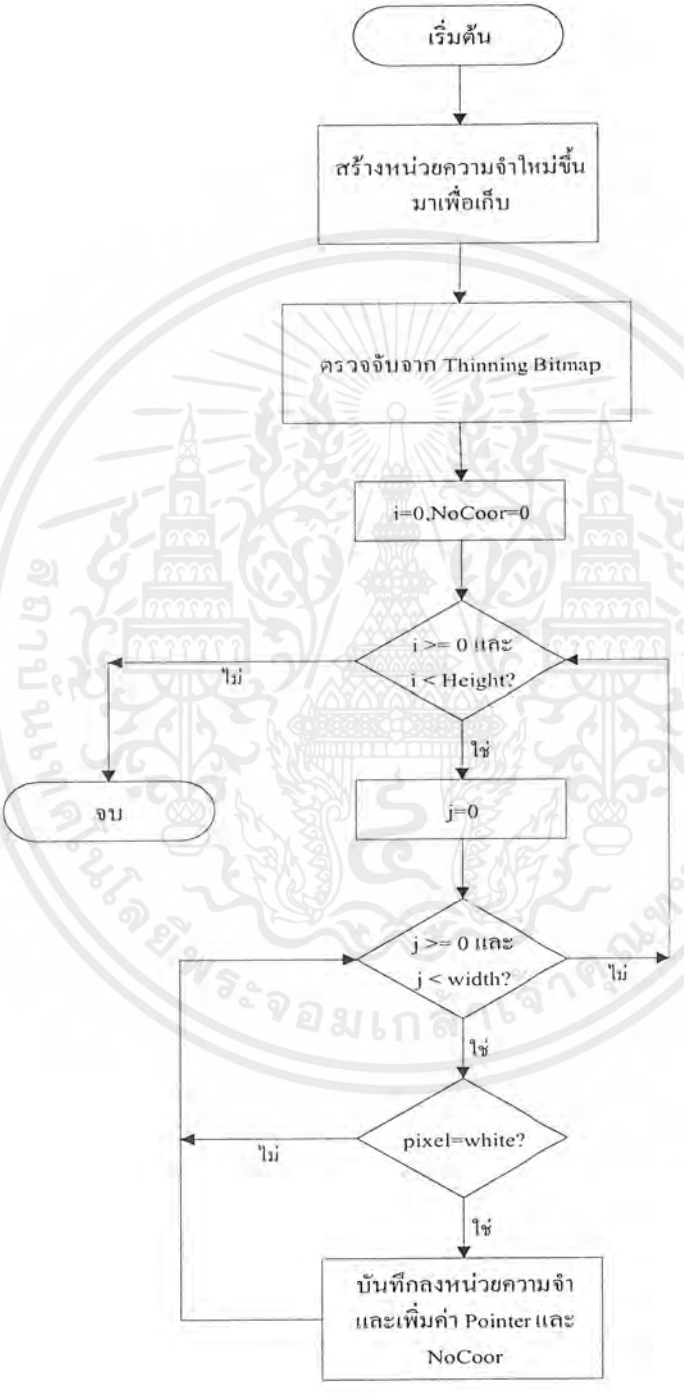
รูปที่ 8.7 แผนภาพแสดงขั้นตอนการทำ Threshold

5. Thinning Bitmap



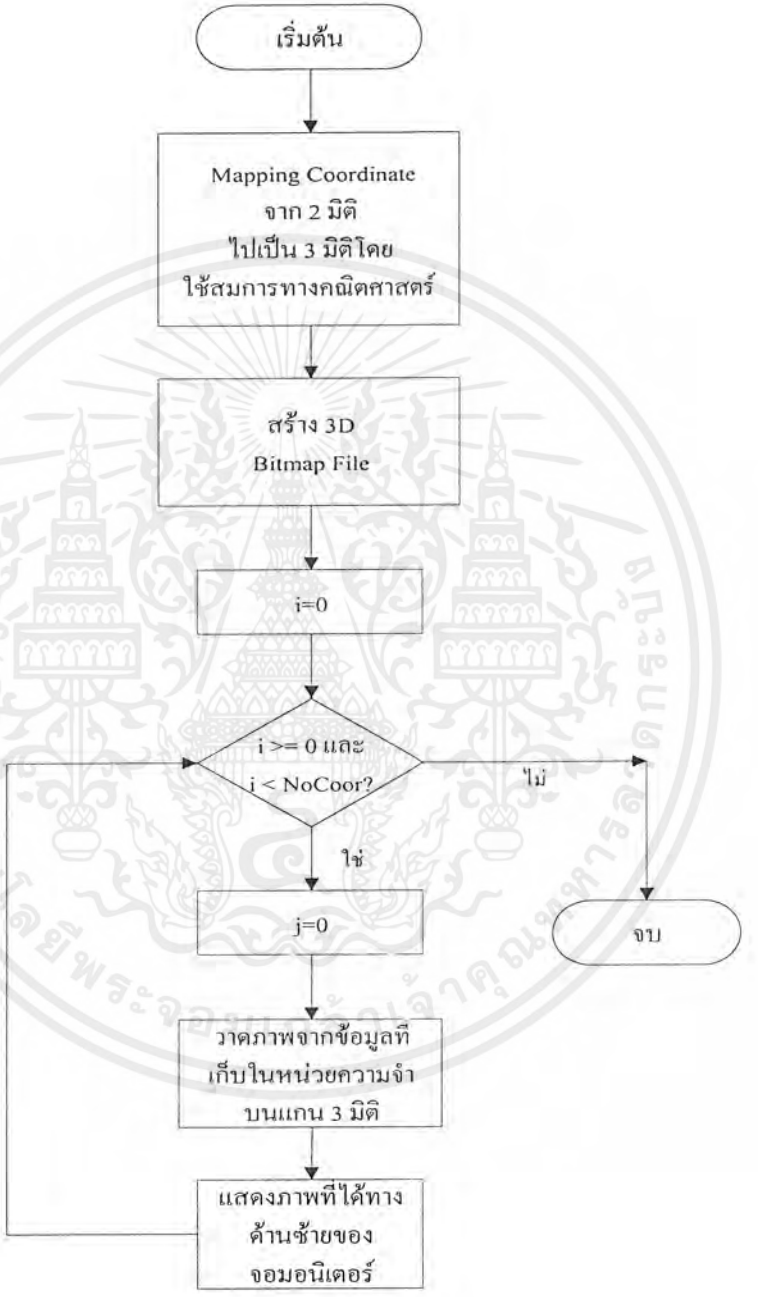
รูปที่ 8.8 แผนภาพแสดงขั้นตอนการทำ Thinning

6. เก็บพิกัดของข้อมูลจาก Thinning Bitmap



รูปที่ 8.9 แผนภาพแสดงขั้นตอนการเก็บพิกัดของข้อมูลภาพจาก Thinning Bmp

7. แสดงภาพที่ได้จากการประมวลผล 3 มิติทางจอมอนิเตอร์



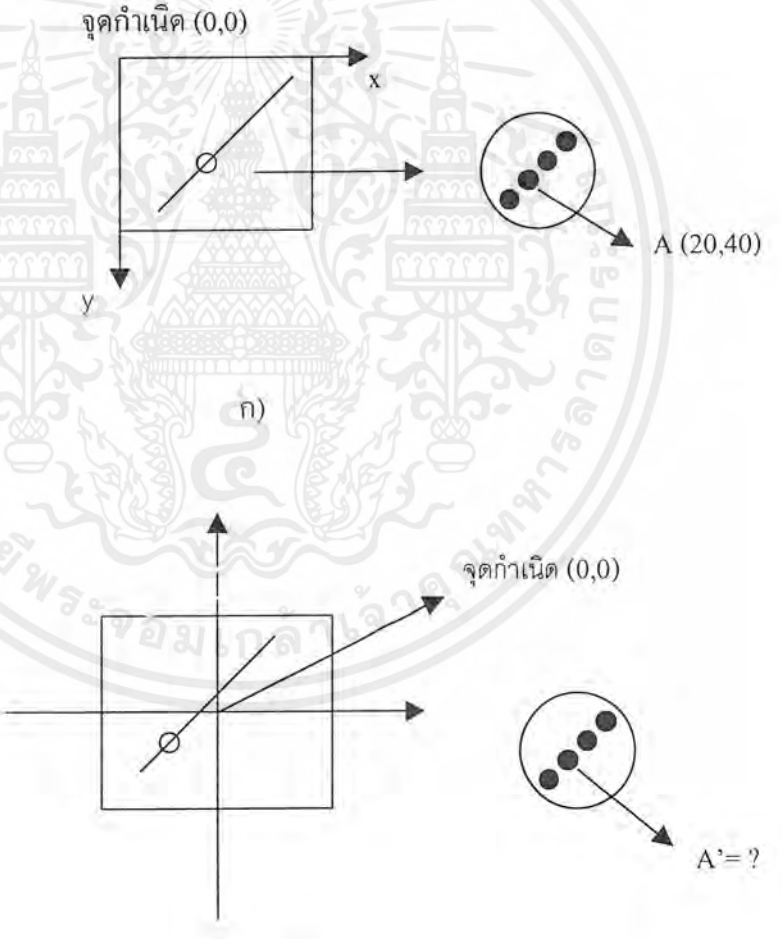
รูปที่ 8.10 แผนภาพแสดงการวาด (Plot) ภาพบนแกน 3 มิติ

8.3 หลักการของการสร้างโครงร่าง 3มิติ

8.3.1) การหาตำแหน่งของวัตถุในระนาบตั้งฉาก

โดยอาศัยหลักการทางคณิตศาสตร์ในการหาจุดตัดระหว่างระนาบของเลเซอร์กับสมการเส้นตรงซึ่งได้จากภาพที่รับจากกล้องกับจุดสังเกต (แทนจุดโฟกัสของกล้อง) ซึ่งจุดตัดนั้น ก็คือ จุดบนวัตถุที่เราต้องการนั่นเอง

1) จากขั้นตอนการทำ Thinning ทำให้ได้ภาพ 2 มิติในแต่ละมุมของวัตถุซึ่งภาพที่ได้เหล่านี้มีจุดกำเนิด (0,0) ที่มุมบนซ้ายของภาพ ดังนั้นจึงทำการ Translate จุดกำเนิดไปอยู่ที่กึ่งกลางของภาพโดยใช้หลักการทางคณิตศาสตร์คำนวณ ซึ่งแสดงได้ดังรูป 8.11



รูปที่ 8.11 แสดงการ Translate จุดกำเนิดของภาพ 2 มิติ

ก) ภาพ 2 มิติที่ได้จากกล้อง ข) ภาพ 2 มิติที่ทำการ Translate จุดกำเนิดแล้ว

2) การTranslateจุดกำเนิดในขั้นตอนที่ 1 จะใช้หลักการTranslateแบบ 2 มิติ ซึ่งได้อธิบายในหัวข้อ 7.12 โดยใช้ แมททริกซ์ ดังสมการ (8.2)

$$[x^*,y^*,z^*] = [x,y,1]\begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & 0 \\ -W/2 & H/2 & 1 \end{bmatrix} \qquad (8.2)$$

เมื่อ (x , y) เป็นจุดบนภาพที่รับจากกล้องซึ่งมีจุดกำเนิดอยู่ที่มุมบนซ้ายของภาพ

เมื่อ (x*, y*) เป็นจุดที่เกิดจากการ Translate จุด (x , y) หลังจากการ Translate จุดกำเนิด

W เป็นความกว้างของภาพตามแนวแกน x

H เป็นความสูงของภาพตามแนวแกน y

ซึ่งภาพที่รับมาจากกล้องมี W=160 Pixel และ H=120 Pixel ดังนั้นเมื่อทำการ Translate จุดกำเนิดตามสมการ (8.2) แล้ว จะได้พิกัดใหม่ของจุด A คือ A'

3) กำหนด Z_C และ Z_E บนแกน Z (เพื่อสะดวกในการคำนวณ) โดยให้

Z_C เป็นระยะทางจากจุดศูนย์กลางการหมุนของวัตถุมายังภาพ

Z_E เป็นระยะทางจากภาพมายังจุดสังเกต

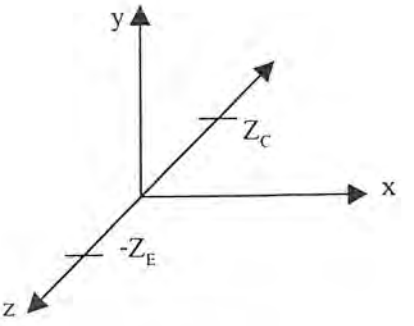
ทำให้ได้ตำแหน่งจุดกำเนิดของภาพที่รับมาซึ่งในขณะนี้อยู่ที่กึ่งกลางภาพแล้ว คือ

$$P_{OC} = (0 , 0 , Z_C)$$

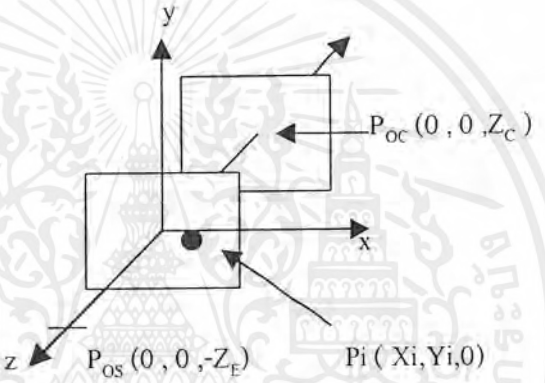
และได้ตำแหน่งของจุดสังเกต คือ

$$P_{OS} = (0 , 0 , -Z_E)$$

ดังรูปที่ 8.12



ก)



ข)

รูปที่ 8.12 รูปแสดงตำแหน่งของ Z_E , Z_C และระนาบของภาพบนแกน 3 มิติ
ก) แสดงตำแหน่งของ Z_E , Z_C ข) ระนาบของภาพ

4) การหาสมการเส้นตรงระหว่างจุดสังเกตกับจุดบนภาพใดๆ

กำหนดให้ Pi = (Xi ,Yi , 0) เป็นพิกัดของจุด A บนระนาบภาพและ

Pos = (0, 0, $-Z_E$) เป็นพิกัดของจุดสังเกต

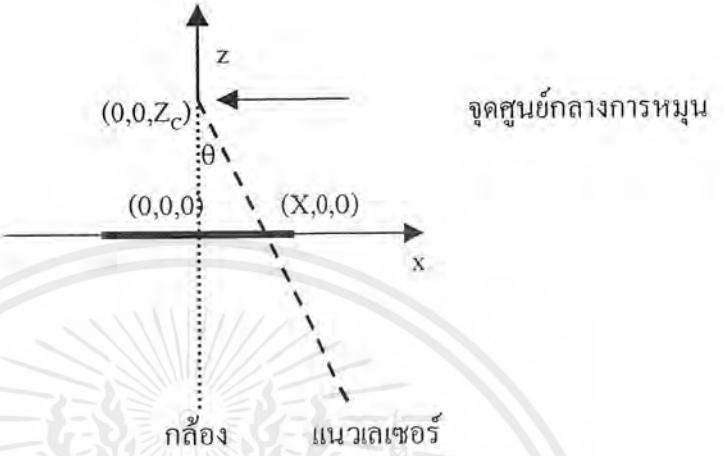
$$(X,Y,Z) = (Xi ,Yi ,0) + (0 ,0 ,Z_E) \quad (8.3)$$

ทำการ Normalize สมการ (8.3) ได้ค่าดังนี้

$$X_D = Xi \,/\, \sqrt{Xi^2 + Yi^2 + (-Z_E)^2}$$
$$Y_D = Yi \,/\, \sqrt{Xi^2 + Yi^2 + (-Z_E)^2}$$
$$Z_D = Z_E \,/\, \sqrt{Xi^2 + Yi^2 + (-Z_E)^2} \quad (8.4)$$

ซึ่ง X_D , Y_D , Z_D เป็นค่า Normalize ของ (X,Y,Z)

5) การหาระนาบของเลเซอร์



ระนาบภาพ ..... แนวกล้องไปจุดศูนย์กลางการหมุน _ _ _ ระนาบเลเซอร์

รูปที่ 8.13 แสดงการวางตัวของระนาบกล้องและระนาบเลเซอร์

เมื่อระนาบของเลเซอร์ขนานกับแกน Y ดังนั้นในการหาสมการระนาบของเลเซอร์นั้นไม่ต้องพิจารณา แกน Y ทำให้การหาสมการระนาบของเลเซอร์ทำเช่นเดียวกับการหาสมการเส้นตรง โดย ให้ m แทนความชันของเส้นตรง

$$m = \frac{Z1-Z2}{X1-X2} = \frac{0-Z_C}{X-0} = \frac{-Z_C}{X} = \frac{-1}{\tan\theta} \qquad (8.5)$$

รูปแบบของสมการเส้นตรง คือ $mX + C = Z$ โดย C เป็นค่าคงที่ (8.6)

แทนค่าจุด $(0,0,Z_C)$ ลงในสมการ (8.6) จะได้

$$C = Z_C$$

แทนค่า m และ C ลงในสมการ (8.6)

$$\frac{-x}{\tan\theta} + Z_C = Z$$

$$\frac{x}{\tan\theta} + Z = Z_C \qquad (8.7)$$

จาก (8.7) คูณ sinθ ทั้ง 2 ข้างของสมการ ได้สมการระนาบของเลเซอร์

$$X\cos\theta + Z\sin\theta = Z_C\sin\theta \qquad (8.8)$$

จากรูปแบบของสมการระนาบ

$$AX + BY + CZ = D$$

ดังนั้นจะได้

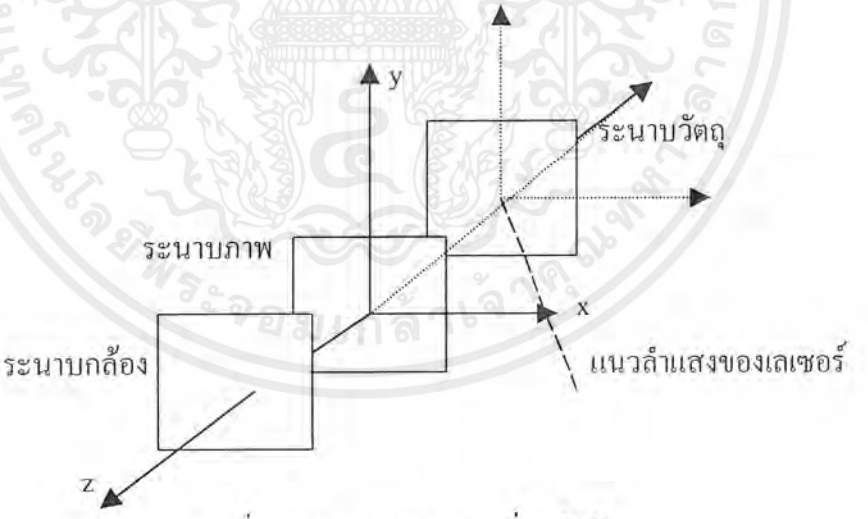
$$A = \cos\theta$$

$$B = 0$$

$$C = \sin\theta$$

$$D = Z_C \sin\theta$$

6) หาระนาบของเลเซอร์ [P_L] เนื่องจากภาพที่รับมาได้ในแต่ละมุมของวัตถุเป็นภาพที่กล้องได้ถ่ายในลักษณะที่เอียงทำมุม θ กับชุดกำเนิดเลเซอร์ ซึ่งตำแหน่งของเส้นเลเซอร์ในภาพที่ได้มาจากการเก็บภาพนั้นยังไม่ใช่ตำแหน่งที่ถูกต้อง จึงต้องหาตำแหน่งบนระนาบของเลเซอร์ที่ถูกต้อง โดยวิธีการดังนี้ระนาบ xy (ระนาบภาพ)



รูปที่ 8.14 แสดงระนาบ xy ที่จุดกำเนิด

6.1) กำหนดให้แสงเลเซอร์ตัดผ่านจุดศูนย์กลางการหมุนของวัตถุในแกน 3 มิติ

6.2) กำหนดระนาบ xy ที่จุดกำเนิด (0,0,0) ดังรูปที่ 8.14 จะได้สมการระนาบที่ Z=0 และ Normalize vector คือ (X_D,Y_D,Z_D)

6.3) หามุมระหว่างระนาบของกล้องกับระนาบของเลเซอร์ที่ได้จากการทดลอง(จากการวัด)

6.4) ทำการหมุนระนาบ xy ไปเป็นมุม θ โดยการใช้สมการ คณิตศาสตร์ที่กล่าวถึงในบทที่ 7

7) การหาสมการ เส้นตรงระหว่าง วัตถุกับจุดสังเกตใดๆ คือ

$$(X^*, Y^*, Z^*) = Ta(X_D, Y_D, Z_D) + (0, 0, -Z_E) \quad (8.9)$$

ได้

$$X^* = TaX_D$$
$$Y^* = TaY_D$$
$$Z^* = TaZ_D - Z_E \quad (8.10)$$

จุด (X^*, Y^*, Z^*) เป็นจุดที่ระนาบของเลเซอร์ ตัดกับวัตถุ ดังนั้นแทนค่าจุด(X^*, Y^*, Z^*) ลงบนระนาบเลเซอร์

$$(TaX_D)\cos\theta + (TaZ_D\sin\theta - Z_E\sin\theta) = Z_C\sin\theta$$
$$Ta(X_D\cos\theta + Z_D\sin\theta) = (Z_C + Z_E)\sin\theta$$
$$Ta = \{(Z_C + Z_E)\sin\theta\}/(X_D\cos\theta + Z_D\sin\theta) \quad (8.11)$$

นำค่า Ta ที่ได้จากสมการ (8.11) แทนในสมการ (8.10) จะได้จุดของภาพจริงที่จะนำมาแสดงในแต่ละองศาของการหมุนกล้องถ่ายภาพ

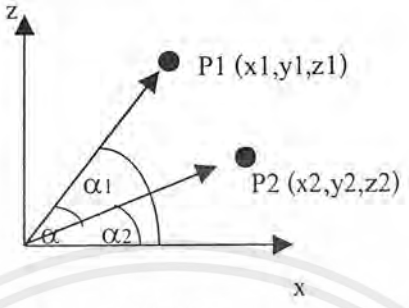
จากขั้นตอน 1-6 เราจะได้ระนาบเลเซอร์ระนาบแรกเป็นระนาบอ้างอิงซึ่งได้จากการรับภาพจากกล้องในขณะที่กล้องยังไม่ได้หมุนไปตามมุมต่างๆ และหลังจากขั้นตอนที่ 7 ทำให้ได้จุดต่างๆในระนาบเลเซอร์ระนาบแรก ซึ่งจุดเหล่านี้เป็นจุดที่แสดงตำแหน่งของเส้นเลเซอร์ที่สามารถตรวจจับได้อย่างถูกต้อง ทั้ง 7 ขั้นตอนนี้จะใช้เป็น Algorithm ในการหาระนาบเลเซอร์ในทุกๆครั้งที่กล้องรับภาพมา แต่เมื่อกล้องหมุนไปเป็นมุมต่างๆ เพื่อทำการรับภาพรอบวัตถุนั้น ระนาบเลเซอร์ของภาพที่รับมาจึงต้องหมุนไปตามมุมที่กล้องหมุนด้วยเพื่อเป็นระนาบจริง มิฉะนั้นภาพที่แสดงผลออกมาที่มุมต่างๆ จะเกิดการซ้อนทับกันที่ตำแหน่งของระนาบอ้างอิง ดังนั้นขั้นตอนต่อไปจึงต้องหาตำแหน่งของวัตถุในระนาบจริงที่มุมใดๆ

8.3.2) การหาตำแหน่งของวัตถุในระนาบจริง

โดยการหมุน จุดบนระนาบตั้งฉากซึ่งผ่านกระบวนการทั้ง 7 ขั้นตอน 8.3.1) ไปตามองศาการหมุนของกล้องในแต่ละสเตป

- การทดลองนั้นใช้การหมุนทั้งหมด 50 สเตป จึงครบรอบ (360°) ดังนั้นจะได้แต่ละสเตปเท่ากับ 360/50 = 7.2 องศา

- ให้จุด P1 เป็นจุดที่อยู่บนระนาบ y = y1 และต้องการจะหมุนรอบแกน y ตามเข็มนาฬิกาไปเป็นมุม α จะได้จุดใหม่ คือ P2



รูปที่ 8.15 แสดงการหมุนจุดบนระนาบรอบแกน y

- เนื่องจากจุด P1 และ P2 อยู่บนระนาบ y1 เหมือนกันจึงเปรียบเสมือนเป็นการหมุนในระนาบ 2 มิติ

- ให้ $\alpha 1$ เป็นมุมที่ P1 ทำกับแกน x

$$x1 = r\cos\alpha 1$$
$$z1 = r\sin\alpha 1 \qquad (8.12)$$

- ต้องการหมุนรอบแกน y ไปเป็นมุม α เพราะฉะนั้นจะได้ P2 ทำมุม $\alpha 2 = \alpha 1 - \alpha$ องศา

$$x2 = r\cos\alpha 2 = r\cos(\alpha 1 - \alpha)$$
$$z2 = r\sin\alpha 2 = r\sin(\alpha 1 - \alpha)$$

กระจายพจน์ $\cos(\alpha 1 - \alpha)$ และ $\sin(\alpha 1 - \alpha)$ ได้

$$x2 = r\cos\alpha 1\cos\alpha + r\sin\alpha 1\sin\alpha$$
$$z2 = r\sin\alpha 1\cos\alpha - r\cos\alpha 1\sin\alpha \qquad (8.13)$$

- นำค่า x1,z1 ของแต่ละจุดที่ได้จากระนาบตั้งฉากมาแทนในสมการ (8.13) จะได้จุดบนวัตถุจริงออกมา

- แต่เนื่องจากการหมุนของสมการที่ (8.13) เป็นสมการที่ใช้หมุนภาพรอบๆจุดกำเนิดหรือหมุนรอบแกน y ซึ่งผิด เพราะฉะนั้นเราจึงต้อง Translate การหมุนจากจุดกำเนิดไปหมุนรอบ จุด-ศูนย์กลางการหมุนของวัตถุ โดยทำดังนี้

Translateจาก (0,y,0) ไปยัง $(0,y,Z_C)$ หรือจุดศูนย์กลางการหมุนของวัตถุ

$$Xr = x2$$

$$Yr = y2$$

$$Zr = z2\text{-}Z_C \tag{8.14}$$

แล้วแทนสมการที่ (8.13) ลงในสมการที่ (8.14) จะได้ดังนี้

$$Xrr = (Xr)\cos\alpha + (z2\text{-}Z_C)\sin\alpha$$

$$Yrr = y2$$

$$Zrr = (z2\text{-}Z_C)\cos\alpha - (Xr)\sin\alpha \tag{8.15}$$

แล้วทำการ Translate จุดในสมการที่ (8.15) กลับมาที่จุดกำเนิดจะได้ดังนี้

$$Xf = Xrr$$

$$Yf = Yrr$$

$$Zf = Zrr + Z_C \tag{8.16}$$

- เป็นค่า α ตามสเตปการหมุนโดยเริ่มที่ 0, 7.2 , 14.4 , ... , 352.8

$$\alpha_n = 7.2\,(n\text{-}1) \tag{8.17}$$

- นำจุดบนวัตถุจริงของแต่ละสเตปมาวาดรวมกันก็จะได้โครงร่าง 3 มิติออกมา

บทที่ 9
ผลการทดลอง

9.1 การจัดเตรียมอุปกรณ์ที่ใช้ในการทดลอง

1) จัดให้มุมกล้องกับแนวลำแสงของเลเซอร์ทำมุมกันประมาณ 25 องศา
2) เชื่อมต่อสายส่งข้อมูลจากการ์ดพอร์ตขนานที่ต่ออยู่กับ ISA Slot เข้ากับชุดควบคุมการหมุนของสเตปมอเตอร์
3) จ่ายไฟเลี้ยงให้กับชุดเลเซอร์และชุดควบคุมการหมุนของสเตปมอเตอร์

จำลองพื้นที่มืดในการทดลอง โดยในที่นี้จะใช้กล่องขนาดใหญ่ครอบชุดทดลองต้นแบบทั้งหมด

9.2 วิธีทดลอง

1) นำวัตถุที่จะสแกนวางบนแท่งสแกน
2) ปิดฝาครอบกล่องที่นำมาจำลองเป็นพื้นที่มืดในการทดลอง
3) ทำการประมวลผลโปรแกรมเครื่องเก็บข้อมูลภาพ 3 มิติ
4) สังเกตผลการทดลองที่ได้ในแต่ละสเตปการหมุนบนหน้าจอมอนิเตอร์

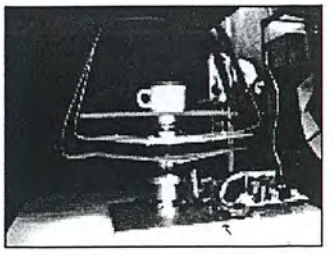
9.3 ผลการทดลอง

สำหรับการทดลองในภาคเรียนที่ 2 นี้ จะเป็นผลที่สำเร็จสมบูรณ์ทั้งหมดของโครงงานนี้ซึ่งมีเป้าหมาย คือ นำภาพที่ได้จากกล้องมาประมวลภาพแล้วนำไปวาดให้ออกมาเป็นรูปแบบของภาพ 3 มิติบนหน้าจอมอนิเตอร์ โดยในส่วนของการประมวลผลนั้นก็ได้แก่การตรวจจับแนวเส้นเลเซอร์จากภาพ (การทำ Threshold) จากนั้นก็ทำ Thinning แนวเส้นเลเซอร์นั้นๆ แล้วนำมา Plot เพื่อแสดงบนหน้าจอมอนิเตอร์ ทำเช่นนี้ทุกสเตปของการหมุนจนครบรอบวัตถุก็จะได้ภาพที่สมบูรณ์ของวัตถุนั้น

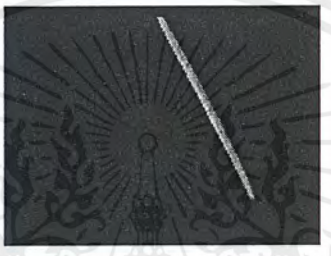
9.3.1 การทดลองโปรแกรม

เป็นการทดลองประมวลผลโปรแกรม โดยใช้ภาพที่สร้างขึ้นเอง เพื่อเป็นการทดสอบความถูกต้องของโปรแกรมในส่วนของการคำนวนจากภาพที่ได้จากกล้องไปเป็นภาพ 3 มิติโดยเวลาที่ใช้ในการทำงานโดยประมาณ คือ 70 วินาที

ภาพต่อไปนี้เป็นภาพตัวอย่างที่ได้จากการทดลองประมวลผลโปรแกรม ซึ่งจะแสดงการประมวลผลภาพทุกขั้นตอนในแต่ละสเตปของการหมุน

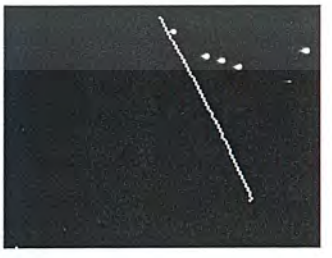


รูปที่ 9.1 แสดงโครงสร้างของเครื่องเก็บข้อมูลภาพ 3 มิติ และวัตถุที่ได้ทำการทดลอง

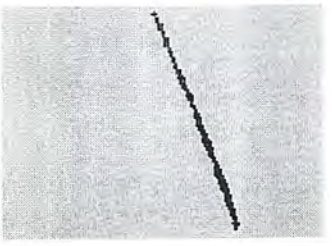


รูปที่ 9.2 แสดงภาพที่สร้างขึ้นเองเพื่อนำมาทดสอบกับโปรแกรมในทุกๆสเตป

รูปที่ 9.3 แสดงภาพที่ได้จากการทำ Threshold



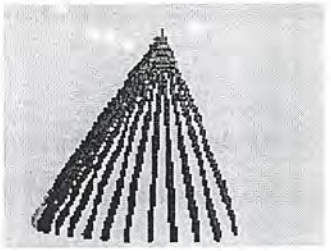
รูปที่ 9.4 แสดงภาพที่ได้จากการทำ Thinning



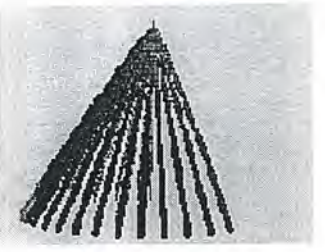
รูปที่ 9.5 แสดงภาพโครงร่าง 3 มิติที่ได้จากการประมวลผลที่สร้างขึ้นในสเตปที่ 1

รูปที่ 9.6 แสดงภาพโครงร่าง 3 มิติที่ได้จากการประมวลผลที่สร้างขึ้นในสเตปที่ 2

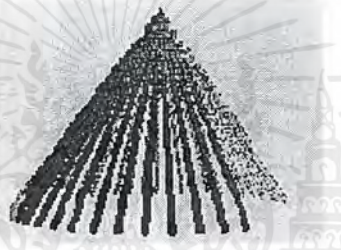
รูปที่ 9.7 แสดงภาพโครงร่าง 3 มิติที่ได้จากการประมวลผลที่สร้างขึ้นในสเตปที่ 5



รูปที่ 9.8 แสดงภาพโครงร่าง 3 มิติที่ได้จากการประมวลผลที่สร้างขึ้นในสเตปที่ 15



รูปที่ 9.9 แสดงภาพโครงร่าง 3 มิติที่ได้จากการประมวลผลที่สร้างขึ้นในสเตปที่ 30



รูปที่ 9.10 แสดงภาพโครงร่าง 3 มิติที่ได้จากการประมวลผลที่สร้างขึ้นในสเตปที่ 40

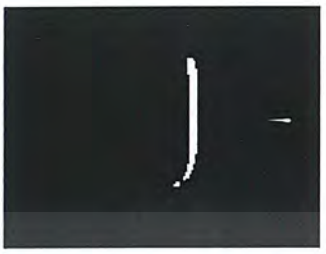
รูปที่ 9.11 แสดงภาพโครงร่าง 3 มิติที่ได้จากการประมวลผลที่สร้างขึ้นในสเตปที่ 50

9.3.2 การทดลองเก็บภาพวัตถุจริง

การทดลองเก็บภาพวัตถุจริง โดยการนำภาชนะจำพวกถ้วยกาแฟ , ชาม และผลไม้ โดยเวลาที่ใช้ในการทำงาน โดยประมาณคือ 70 วินาที

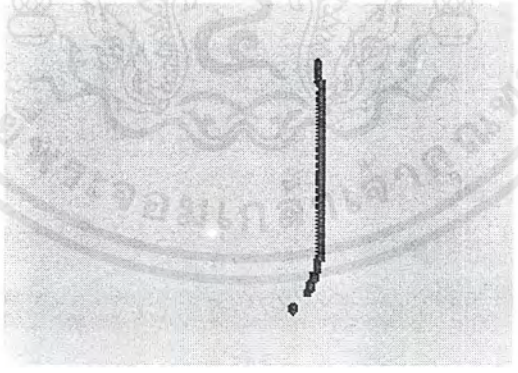
ภาพต่อไปนี้เป็นภาพตัวอย่างที่ได้จากการทดลองประมวลผลโปรแกรม ซึ่งจะแสดงการประมวลผลภาพทุกขั้นตอนในแต่ละสเตปของการหมุนกล้อง

ก)



ข)

ค)



ง)

รูปที่ 9.12 แสดงภาพที่ได้จากการประมวลผลของภาพที่รับจากกล้องสเตปที่ 1

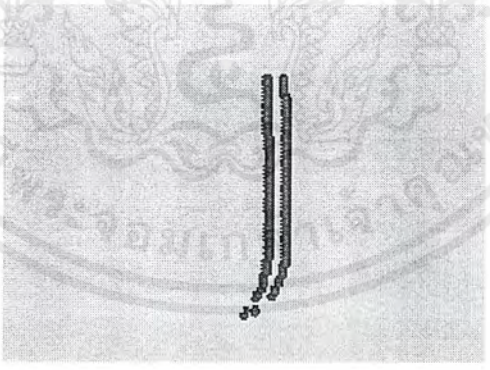
ก) ภาพที่ได้จากการ Capture ข) ภาพที่ได้จากการทำ Threshold

ค) ภาพที่ได้จากการทำ Thinning ง) ภาพโครงร่าง 3 มิติที่สร้างได้

ก)

ข)

ค)



ง)

รูปที่ 9.13 แสดงภาพที่ได้จากการประมวลผลของภาพที่รับจากกล้องสเตปที่ 2

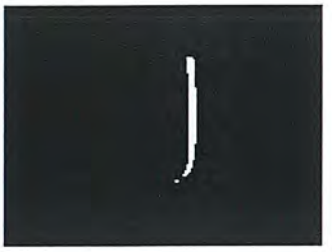
ก) ภาพที่ได้จากการ Capture

ข) ภาพที่ได้จากการทำ Threshold

ค) ภาพที่ได้จากการทำ Thinning

ง) ภาพโครงร่าง 3 มิติที่สร้างได้

ก)



ข)

ค)

ง)

รูปที่ 9.14 แสดงภาพที่ได้จากการประมวลผลของภาพที่รับจากกล้องสเตปที่ 5

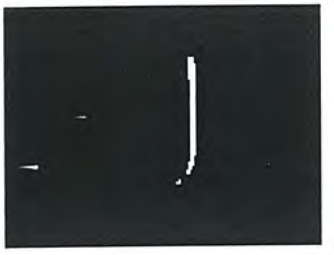
ก) ภาพที่ได้จากการ Capture

ข) ภาพที่ได้จากการทำ Threshold

ค) ภาพที่ได้จากการทำ Thinning

ง) ภาพโครงร่าง 3 มิติที่สร้างได้

ก)



ข)

ค)

ง)

รูปที่ 9.15 แสดงภาพที่ได้จากการประมวลผลของภาพที่รับจากกล้องสเตปที่ 10

ก) ภาพที่ได้จากการ Capture

ข) ภาพที่ได้จากการทำ Threshold

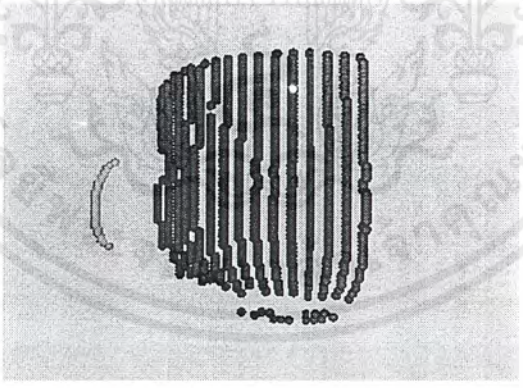
ค) ภาพที่ได้จากการทำ Thinning

ง) ภาพโครงร่าง 3 มิติที่สร้างได้

ก)

ข)

ค)



ง)

รูปที่ 9.16 แสดงภาพที่ได้จากการประมวลผลของภาพที่รับจากกล้องสเตปที่ 17

ก) ภาพที่ได้จากการ Capture

ข) ภาพที่ได้จากการทำ Threshold

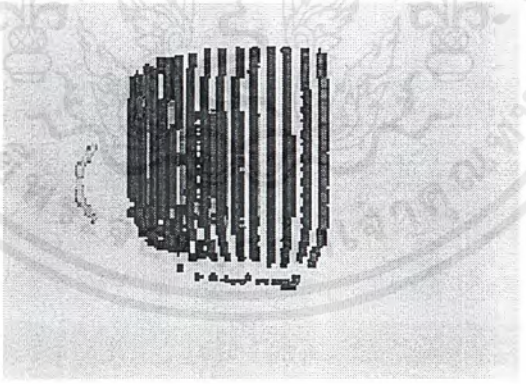
ค) ภาพที่ได้จากการทำ Thinning

ง) ภาพโครงร่าง 3 มิติที่สร้างได้

ก)

ข)

ค)



ง)

รูปที่ 9.17 แสดงภาพที่ได้จากการประมวลผลของภาพที่รับจากกล้องสเตปที่ 30

ก) ภาพที่ได้จากการ Capture

ข) ภาพที่ได้จากการทำ Threshold

ค) ภาพที่ได้จากการทำ Thinning

ง) ภาพโครงร่าง 3 มิติที่สร้างได้

ก)

ข)

ค)

ง)

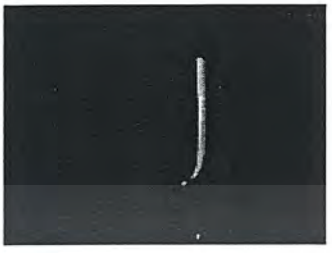
รูปที่ 9.18 แสดงภาพที่ได้จากการประมวลผลของภาพที่รับจากกล้องสเตปที่ 40

ก) ภาพที่ได้จากการ Capture

ข) ภาพที่ได้จากการทำ Threshold

ค) ภาพที่ได้จากการทำ Thinning

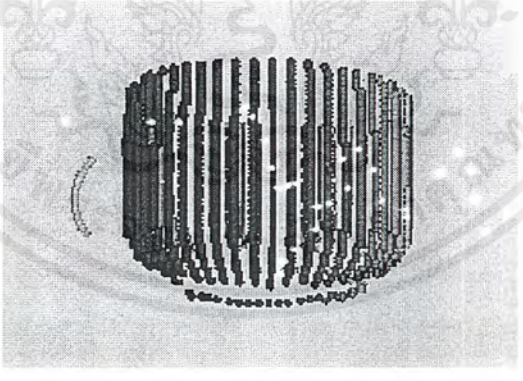
ง) ภาพโครงร่าง 3 มิติที่สร้างได้



ก)

ข)

ค)



ง)

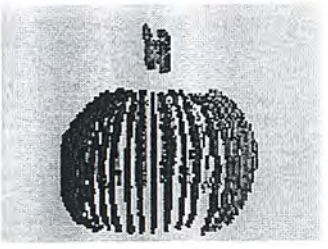
รูปที่ 9.19 แสดงภาพที่ได้จากการประมวลผลของภาพที่รับจากกล้องสเตปที่ 50

ก)ภาพที่ได้จากการ Capture

ข) ภาพที่ได้จากการทำ Threshold

ค) ภาพที่ได้จากการทำ Thinning

ง) ภาพโครงร่าง 3 มิติที่สร้างได้

ก) ข)

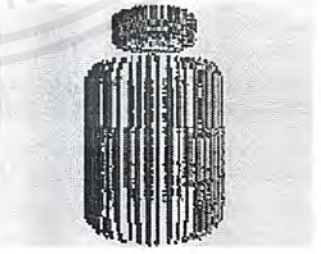
รูปที่ 9.20 แสดงการเปรียบเทียบภาพวัตถุจริงกับภาพจากการสแกนผลไม้

ก) ภาพวัตถุจริงที่รับจากกล้อง ข) ภาพจากการสแกน


ก) ข)

รูปที่ 9.21 แสดงการเปรียบเทียบภาพวัตถุจริงกับภาพจากการสแกนถ้วยกาแฟ

ก) ภาพวัตถุจริงที่รับจากกล้อง ข) ภาพจากการสแกน

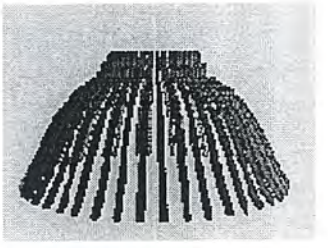



ก) ข)

รูปที่ 9.22 แสดงการเปรียบเทียบภาพวัตถุจริงกับภาพจากการสแกนขวดพลาสติก

ก) ภาพวัตถุจริงที่รับจากกล้อง ข) ภาพจากการสแกน

ก)



ข)

รูปที่ 9.23 แสดงการเปรียบเทียบภาพวัตถุจริงกับภาพจากการสแกนชาม
ก) ภาพวัตถุจริงที่รับจากกล้อง ข) ภาพจากการสแกน

บทที่ 10
สรุปผลและวิจารณ์ผลการทดลอง

10.1 สรุปผลและวิจารณ์ผลการทดลอง

จากการที่ใช้วัตถุที่นำมาสแกน คือถ้วยกาแฟ เมื่อนำมาทำตามขั้นตอนต่างๆในวิธีการทดลอง สุดท้ายจะออกมาเป็นรูปร่างของซึ่งคล้ายกับวัตถุจริงมาก ดังรูปที่ 10.1

ก) ข)

รูปที่ 10.1 แสดงการเปรียบเทียบภาพวัตถุจริงกับภาพจากการสแกน
ก) ภาพวัตถุจริงที่รับมาจากกล้อง ข) ภาพจากการสแกน

ซึ่งผลที่ได้ในภาพจากการสแกนจะพบว่ายังมีบางเส้นจะผิดเพี้ยนไป โดยเส้นบางเส้นที่ควรจะเป็นเส้นตรง ก็มีการผิดเพี้ยนไป เนื่องจากไม่สามารถควบคุมความกว้างของแนวลำแสงเลเซอร์ให้สม่ำเสมอได้ (เนื่องจากแผงยิงเลเซอร์นั้นราคาค่อนข้างสูง) ในเครื่องเก็บข้อมูลภาพ 3 มิติเครื่องนี้จึงใช้ Laser Pointer มาดัดแปลงให้ได้เป็นแนวเลเซอร์ออกมา(โดยใช้เลนส์ทรงกระบอก) จึงส่งผลให้แนวเส้นที่ได้จากการ Threshold และ Thinning นั้นไม่เป็นเส้นตรง โดยมีความผิดพลาดในตำแหน่งต่างๆของ Pixel แต่ภาพโดยรวมที่ได้จากการทดลองก็ยังคงความเป็นวัตถุจริงอยู่

ในส่วนของชุดทดลองต้นแบบนั้นยังมีปัญหาการสั่นสะเทือนบ้างเล็กน้อยในการเริ่มหมุนในสเตปแรก โดยในสเตปต่อๆไปนั้นการสั่นสะเทือนจะลดลง และการผิดเพี้ยนของภาพที่ได้จากการทดลองนั้นมีผลมาจากความไม่สมดุลของชุดทดลองต้นแบบด้วย

10.2 สิ่งที่ดำเนินการในภาคเรียนที่ 1

1) ส่วนของฮาร์ดแวร์ (Hardware)

-ชุดทดลองต้นแบบ (Mechanic) สำหรับวางวัตถุที่จะสแกน

- ชุดกล้องวีดีโอ

- การ์ดควบคุมการหมุนของสเตปมอเตอร์

2) ส่วนของซอฟแวร์ (Software)

-โปรแกรมควบคุมการหมุนของสเตปมอเตอร์

-โปรแกรมรับภาพจากกล้อง และทำการ Capture แล้วนำแสดงออกหน้าจอมอนิเตอร์

10.3 สิ่งที่ดำเนินงานในภาคเรียนที่ 2

2) 1) ส่วนของฮาร์ดแวร์ (Hardware)

- ปรับปรุงชุดทดลองต้นแบบ (Mechanic) ให้มีโครงร่างที่สมบูรณ์

- สร้างชุดเลเซอร์ (Laser)

2) ส่วนของซอฟแวร์ (Software)

-โปรแกรม Threshold เพื่อตรวจจับแนวเส้นเลเซอร์จากภาพ

-โปรแกรม Thinning ให้แนวเลเซอร์บางเท่ากัน

-โปรแกรม Keep Coordinate เพื่อเก็บพิกัดของเส้นเลเซอร์

-โปรแกรม 3D เพื่อนำพิกัดที่ได้มาวาด แล้วแสดงผลบนหน้าจอมอนิเตอร์

10.4 ข้อจำกัดของโครงงาน

1) ขนาดของวัตถุ

เนื่องจากชุดทดลองต้นแบบนี้ต้องมีโครงสร้างที่มั่นคง เพื่อแก้ปัญหาการสั่นของชุด Mechanic และปัญหาต่างๆที่จะเกิดตามมา อีกทั้งข้อจำกัดของกล้องในการรับภาพด้วย เพราะถ้าหากต้องการรับภาพวัตถุที่มีความสูงมาก จะต้องให้ตำแหน่งของกล้องอยู่ห่างจากวัตถุพอสมควร ซึ่งต้องต่อแขนของกล้องออกไปอีกเป็นผลให้โครงสร้างมีขนาดใหญ่มาก ไม่สะดวกในการทดลอง ดังนั้นรูปแบบของโครงสร้างที่ใช้ในโครงงานนี้จึงมีข้อจำกัดในด้านของความสูงของวัตถุด้วย โดยความสูงของวัตถุต้องไม่เกิน 12 เซนติเมตร จึงจะรับภาพวัตถุทั้งชิ้นได้

2) **ความมืดในการทดลอง**

เนื่องจากในโครงงานนี้ต้องการตรวจจับ แนวเลเซอร์ที่ตกกระทบบนวัตถุ ซึ่งจะทำให้ได้ง่ายและเร็ว หากมีแสงสว่างบริเวณที่ทดลองน้อยหรือไม่มีเลย เพราะจะทำให้กล้องสามารถจับภาพ

แนวเส้นเลเซอร์นั้นได้อย่างชัดเจนและประมวลผลได้ง่าย ดังนั้นในโครงงานนี้จึงได้จำลองพื้นที่ของการทดลองให้มืดโดยการนำกล่องขนาดใหญ่มาครอบไว้

3) สีของวัตถุที่นำมาสแกน

เนื่องจากการทดลองนี้อยู่ในพื้นที่มืด ดังนั้นถ้าใช้วัตถุที่มีสีดำหรือสีที่ดูดกลืนแสงได้ดีก็จะทำให้ลำแสงเลเซอร์ที่ไปตกกระทบถูกดูดกลืนไปด้วย เป็นผลให้ไม่สามารถตรวจจับแนวเส้นเลเซอร์จากภาพได้ ดังนั้นวัตถุที่จะนำมาสแกนควรจะเป็นสีขาวหรือสีอื่นๆ ที่ดูดกลืนแสงได้น้อยๆ

4) ความโค้ง-เว้าของวัตถุ

ถ้าวัตถุมีความโค้ง-เว้ามากเกินกว่ามุมของกล้องกับเลเซอร์ ก็อาจจะเป็นผลให้แนวเส้นเลเซอร์ถูกบังทำให้กล้องไม่สามารถรับภาพตรงส่วนที่ถูกบังได้ ดังนั้นจึงอาจทำให้แนวเส้นเลเซอร์ที่ได้จากการทำ Threshold และ Thinning นั้นขาดหายไปเป็นผลให้เกิดความไม่สมบูรณ์ของโครงร่าง 3 มิติได้

5) ความไม่ราบเรียบในการหมุนของโครง (ชุดทดลองต้นแบบ)

เนื่องจากในโครงงานนี้ใช้สเตปมอเตอร์เป็นตัวต้นแรงในการหมุนจึงทำให้การหมุนเมื่อทำการสแกนไม่ราบเรียบเท่าที่ควร และ ชุดทดลองต้นแบบที่ไม่สมดุลย์ ทำให้การหมุนไม่ราบเรียบและมีเสียงเกิดขึ้นเล็กน้อย

6) ภาพที่ได้ออกมาจากการประมวลผลภาพ 3 มิติ

ภาพที่ได้จาการประมวลผลภาพ 3 มิติ ที่ออกมาเป็นแนวเส้นดังรูป 10.1 (ข) ซึ่งเป็น Bubble Render ซึ่งไม่เป็นพื้นผิวเหมือนวัตถุจริงที่นำมาสแกน อีกทั้งข้อมูลภาพที่ได้จากการประมวลผลภาพยังไม่สามารถนำไปใช้ประโยชน์หรือประยุกต์ใช้งานได้

10.5 แนวทางในการพัฒนาและปรับปรุงโครงงาน

เนื่องจากในโครงงานเครื่องเก็บข้อมูลภาพ 3 มิติ นี้มีข้อจำกัดดังที่กล่าวมาแล้ว เพื่อเป็นแนวทางในการพัฒนาและปรับปรุงโครงงานนี้ เพื่อให้เกิดความสมบูรณ์ขึ้น ควรพัฒนาและปรับปรุงในด้านต่างๆ ดังนี้

10.5.1 การพัฒนาฮาร์ดแวร์

1) ชุด Mechanic

ควรปรับปรุงให้มีความแข็งแรง , มีความสมดุลย์ และไม่เกิดการสั่นเนื่องมาจากการหมุนของสเตปมอเตอร์ เพื่อให้การหมุนราบเรียบขึ้น ถ้าการหมุนไม่ราบเรียบจะส่งผลกระทบต่อการรับภาพจากกล้อง

2) ชุดเลเซอร์

ควรปรับปรุงให้แนวแสงที่ออกมาจากเลเซอร์สม่ำเสมอและมีความคมชัดมากขึ้น

10.5.2 การพัฒนาซอฟแวร์

1) พัฒนาโปรแกรมให้สามารถเก็บภาพวัตถุในสภาวะที่มีแสงได้

เนื่องจากว่าในการใช้งานจริงๆ จะมีแสงอยู่เสมอ เช่นในกองถ่ายภาพยนต์ ที่มีแสงรบกวนอยู่เสมอ จึงควรปรับปรุงให้สามารถสแกนวัตถุในสภาพที่มีแสงได้ โดยไม่จำเป็นต้องเอากล่องมาคลุมให้มืด เทคนิคที่ใช้เรียกว่า An Unbiased Detector Of Curvilinear Structures ซึ่งเป็นเทคนิคที่ใช้ในการถ่ายภาพจากดาวเทียม

2) พัฒนาโปรแกรมให้สามรถวาดออกมาเป็นพื้นผิวได้ (Render)

เนื่องจากในโครงงานนี้สามารถแสดงผลออกมาได้แค่เป็นแนวเส้น 3 มิติ (Bubble Render) ซึ่งไม่เป็นพื้นผิวเหมือนวัตถุจริงที่นำมาสแกน จึงควรปรับปรุงให้สามารถแสงผลภาพออกมาเป็นพื้นผิว(Render)ได้เพราะจะทำให้เหมือนวัตถุจริงที่นำมาสแกนมากขึ้นอีกทั้งยังทำให้เกิดรายละเอียดในการสแกนมากขึ้นด้วย

3) พัฒนาโปรแกรมให้สามารถนำไปประยุกต์ใช้งานได้

ภาพที่ได้จากการสแกน 3 มิตินี้ ควรที่จะเก็บ Coordinate เพื่อนำไปใช้ประโยชน์ในการสร้างรูปทรง 3 มิติ โดยใช้โปรแกรมกราฟฟิกทางด้าน 3 มิติอื่นๆ เข้ามาช่วยเช่น โปรแกรม 3D STUDIO MAX แล้วนำไปใช้ทำประโยชน์เช่น การหมุนภาพไปในมุมต่างๆ , ปรับแต่งสีให้เหมือนวัตถุจริง

ภาคผนวก

```
////////////////////////////////////////////////
// testView.cpp : implementation of the CTestView class
////////////////////////////////////////////////

#include "stdafx.h"
#include "test.h"
#include "math.h"
#include "3DClass.h"
#include "CDib.h"
#include "CDibTrue.h"
#include "Camera.h"
#include "DataScatteringSpace.h"
#include <conio.h>
// VVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
#include "testDoc.h"
#include "testView.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
//VVVVVVVVVVVVVVVVVVVVVVVVVVVVV
int Width;
int Height;
int i;
int j;
int icount=0;
bool Toggle=TRUE;
bool Round=FALSE;
unsigned short port=0x280;
int value=0x80;
//VVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
/////////////////////////////////////////////////////////////////////////////
// CTestView
IMPLEMENT_DYNCREATE(CTestView, CView)
BEGIN_MESSAGE_MAP(CTestView, CView)
        //{{AFX_MSG_MAP(CTestView)
        ON_COMMAND(Operate, OnOperate)
        ON_WM_TIMER()
        //}}AFX_MSG_MAP
```

```
        // Standard printing commands
        ON_COMMAND(ID_FILE_PRINT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_DIRECT, CView::OnFilePrint)
        ON_COMMAND(ID_FILE_PRINT_PREVIEW, CView::OnFilePrintPreview)
END_MESSAGE_MAP()

////////////////////////////////////////////////////////////////////////
// CTestView construction/destruction
CTestView::CTestView()
{
        m_pDSS1 = new CDataScatteringSpace();
}

CTestView::~CTestView()
{
        //delete VVV;
}

BOOL CTestView::PreCreateWindow(CREATESTRUCT& cs)
{

        return CView::PreCreateWindow(cs);
}

////////////////////////////////////////////////////////////////////////
void CTestView::OnDraw(CDC* pDC)
{
        CTestDoc* pDoc=GetDocument();
        ASSERT_VALID(pDoc);
        BMP->Display(pDC,CPoint(600,150));
        Pic1->Display(pDC,CPoint(600,300));
        m_pDSS1->GetPic()->Display(pDC,CPoint(0,0));
}
////////////////////////////////////////////////////////////////////////
// CTestView printing

BOOL CTestView::OnPreparePrinting(CPrintInfo* pInfo)
{
                return DoPreparePrinting(pInfo);
}
```

```
void CTestView::OnBeginPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
        }

void CTestView::OnEndPrinting(CDC* /*pDC*/, CPrintInfo* /*pInfo*/)
{
        }

/////////////////////////////////////////////////////////////////////////////

#ifdef _DEBUG
void CTestView::AssertValid() const
{
        CView::AssertValid();
}

void CTestView::Dump(CDumpContext& dc) const
{
        CView::Dump(dc);
}
CTestDoc* CTestView::GetDocument() // non-debug version is inline
{
        ASSERT(m_pDocument->IsKindOf(RUNTIME_CLASS(CTestDoc)));
        return (CTestDoc*)m_pDocument;
}
#endif //_DEBUG
/////////////////////////////////////////////////////////////////////////////
void CTestView::OnInitialUpdate()
{
        CView::OnInitialUpdate() ;
                        VidCap.SetView((CView*)this);
        VidCap.Create(WS_CHILD|WS_VISIBLE,CRect(600,5,0,0),this);
        VidCap.DriverConnect(0);
        VidCap.AutoSize();
        VidCap.Preview(TRUE);
        VidCap.SetCallbackOnFrame(FALSE);
        LPBITMAPINFO PicInfo;
        PicInfo=VidCap.GetVideoFormat();
        long Size;
```

```
        Width=PicInfo->bmiHeader.biWidth;
        Height=PicInfo->bmiHeader.biHeight;
        WORD BitPerPixel=PicInfo->bmiHeader.biBitCount;
        BMP=(CDib*)new CDibTrue(Width,Height);
        Pic1=(CDib*)new CDibTrue(Width,Height);
        Size=sizeof(BITMAPINFOHEADER);
}
void MyVidClass::OnFrameCallback(LPVIDEOHDR lpVidHdr)
{
        SetCallbackOnFrame(FALSE);
        Preview(FALSE);
        KillTimer(2);
        icount++;
        CDib* BMP=((CTestView*)View)->GetBitmap();
        CDib* Pic1=((CTestView*)View)->GetBitmap1();
        CDataScatteringSpace* m_pDSS1=((CTestView*)View)->GetScattering();
        unsigned char* Des=BMP->ReceivePic();
        unsigned char* Src=lpVidHdr->lpData;
        unsigned char* SrcOld=Des;
        long Size;
        Size=BMP->ReceiveBMPInfoHeader()->biSizeImage;
        //Copy Image
        unsigned char* Threshold;
        unsigned char* Original;
        Threshold=new unsigned char[Width*Height];
        Original=Threshold;
        for(i=0;i<Size;i++)
        {
                *Des=*Src;
                Des++;
                Src++;
        }
        Des=SrcOld;
        //Threshold
        BYTE Red;
        BYTE Green;
        BYTE Blue;
        unsigned char Lightness;
        Threshold=new unsigned char[(Width*Height)];
        Original=Threshold;
```

```
for(i=0;i<Height;i++)
{
for(j=0;j<Width;j++)
{
        Red=*Des;
        Des++;
        Green=*Des;
        Des++;
        Blue=*Des;
        Des++;
        Lightness=(Red+Green+Blue)/3;
        if ((Lightness>=0)&&(Lightness<70))
        {
                *Threshold=0;
        }
        else
                *Threshold=255;
        Threshold++;
}
}
Threshold=Original;
Des=SrcOld;
//Copy threshold
for(i=0;i<Height;i++)
{
for(j=0;j<Width;j++)
{
        if(*Threshold==255)
        {
                *Des=255;Des++;
                *Des=255;Des++;
                *Des=255;Des++;
        }
        else
        {
                *Des=0;Des++;
                *Des=0;Des++;
                *Des=0;Des++;
        }
        Threshold++;
```

```
}
}
Threshold=Original;
Des=SrcOld;
//Thinning Section
unsigned char* Des1=Pic1->ReceivePic();
unsigned char* SrcOld1=Des1;
unsigned char* Src1=Threshold;
  int y,x,p;
        int startx,stopx;
        int first=0;p=0;
        startx=2;stopx=10;
        for(y=0;y<Height;y++)
        {
                first=0;
                for(x=0;x<Width;x++)
                {
                        if(Threshold[p]==255)
                        {
                                first++;
                                if(first==1)
                                {
                                        startx=p;
                                }
                                else
                                {
                                        stopx=p;
                                }
                        }
                        p++;
                }
                for(x=startx;x<=stopx;x++)
                {
                        if(x==int((startx+stopx)/2))
                        {
                                Threshold[x]=255;
                        }
                        else
                        {
                                Threshold[x]=0;
```

```
                        }
                }
        }
        Threshold=Original;
        int Whitecount=0;
        for(y=0;y<Height;y++)
        {
                for(x=0;x<Width;x++)
                {
                        if(*Threshold==255)
                        {
                                Whitecount++;
                        }
                        Threshold++;
                }
        }
        Threshold=Original;
        double* X;
        double* Y;
        double* Z;
        X=new double[Whitecount];
        Y=new double[Whitecount];
        Z=new double[Whitecount];
        double *K=X;
        double *KK=Y;
        double *KKK=Z;

//Store in Array//
LPBYTE Thsorce=Threshold;
BYTE ThImage[160][120];
int H,W;
for(H=0;H<Height;H++)
{
        for(W=0;W<Width;W++)
        {
                ThImage[W][H]=*Thsorce;
                Thsorce++;
        }
}
Thsorce=Threshold;
```

```
//Keep coordinate//
for(H=0;H<Height;H++)
{
        for(W=0;W<Width;W++)
        {
                if(ThImage[W][H]==255)
                {
                        //int TrX1,TrY1;
                        //Translate origin to center of image//
                        *X=W-(Width/2);
                        X++;
                        *Y=(Height/2)-H;
                        Y++;
                }
        }
}
X=K;
Y=KK;
Z=KKK;

double* XX;
double* YY;
double* ZZ;
XX=new double[Whitecount];
YY=new double[Whitecount];
ZZ=new double[Whitecount];

double* L=XX;
double* LL=YY;
double* LLL=ZZ;

double* XXX;
double* YYY;
double* ZZZ;
XXX=new double[Whitecount];
YYY=new double[Whitecount];
ZZZ=new double[Whitecount];

double* O=XXX;
double* OO=YYY;
```

```
double* OOO=ZZZ;

int pp;
int Ze=70;
int Zc=175;
double pi= 3.14159265359;
double CLAngle=(pi/180)*25;
double StepAngle=(pi/180)*(360.0/50.0)*(icount-1);
double A;
//pp=Whitecount;
for(pp=0;pp<Whitecount;pp++)
{
        XX[pp]=(X[pp])/(sqrt(((X[pp])*(X[pp]))+((Y[pp])*(Y[pp]))+(Ze*Ze)));
        YY[pp]=(Y[pp])/(sqrt(((X[pp])*(X[pp]))+((Y[pp])*(Y[pp]))+(Ze*Ze)));
        ZZ[pp]=(Ze)/(sqrt(((X[pp])*(X[pp]))+((Y[pp])*(Y[pp]))+(Ze*Ze)));


        A=((Zc+Ze)*sin(25*(pi/180)))/(XX[pp]*cos(25*(pi/180))+ZZ[pp]*sin(25*(pi/180)));

        XXX[pp]=A*XX[pp];
        YYY[pp]=A*YY[pp];
        ZZZ[pp]=(A*ZZ[pp])-Ze;


XX=L;
YY=LL;
ZZ=LLL;

XXX=O;
YYY=OO;
ZZZ=OOO;

//////Translate Origin to center of object//////

double* Xob;
double* Yob;
double* Zob;
Xob=new double[Whitecount];
Yob=new double[Whitecount];
Zob=new double[Whitecount];
```

```
            Xob[pp]=XXX[pp];
            Yob[pp]=YYY[pp];
            Zob[pp]=ZZZ[pp]-Zc;


/////Rotate Around Center of Object/////

double* RtX;
double* RtY;
double* RtZ;
RtX=new double[Whitecount];
RtY=new double[Whitecount];
RtZ=new double[Whitecount];

            RtX[pp]=Xob[pp]*cos(StepAngle)+Zob[pp]*sin(StepAngle);
            RtY[pp]=Yob[pp];
            RtZ[pp]=Zob[pp]*cos(StepAngle)-Xob[pp]*sin(StepAngle);

//////Translate from Object to Origin/////

double* Xf;
double* Yf;
double* Zf;
Xf=new double[Whitecount];
Yf=new double[Whitecount];
Zf=new double[Whitecount];

            Xf[pp]=RtX[pp]+(125);     //Plus for move axis to right hand
            Yf[pp]=((RtY[pp]*(-1))+125);//(-1)and (+5) for move axis
            Zf[pp]=RtZ[pp]+Zc;

*X=Xf[pp];*Z=Zf[pp];*Y=Yf[pp];
m_pDSS1->AddSphere(  *X,  *Z,  *Y,3.0f);

}

//Copy thinning
for(i=0;i<Height;i++)
{
```

```
            for(j=0;j<Width;j++)
            {
                        if(*Threshold==255)
                        {
                                    *Des1=255;Des1++;
                                    *Des1=255;Des1++;
                                    *Des1=255;Des1++;
                        }
                        else
                        {
                                    *Des1=0;Des1++;
                                    *Des1=0;Des1++;
                                    *Des1=0;Des1++;
                        }
                        Threshold++;
            }
            }
            Threshold=Original;
            Des1=SrcOld1;

            ((CTestView*)View)->InvalidateRect(NULL,FALSE);
}
void CTestView::OnOperate()
{
            // TODO: Add your command handler code here
            SetTimer(2,100,NULL);
}
void CTestView::OnTimer(UINT nIDEvent)
{
            // TODO: Add your message handler code here and/or call default
            if(icount==50)
            {
                        KillTimer(2);
                        VidCap.SetCallbackOnFrame(FALSE);
                        VidCap.Preview(TRUE);
                        GetDC()->TextOut(10,20,"END TASK");
                        return;
            }
            else
            {
```

```
                if(Toggle==FALSE)
                {
                        if(Round==TRUE)
                        {
                                Toggle=TRUE;
                                Round=FALSE;
                        }
                        else
                        {
                                _outp(port,value);
                                switch(value)
                                {
                                  case(0x80): value=0x40; break;
                                        case(0x40): value=0x20; break;
                                        case(0x20): value=0x10; break;
                                        case(0x10): value=0x80; Round=TRUE;
                                }
                        }
                }
                else
                {
                        Toggle=FALSE;
                        KillTimer(2);
                        VidCap.SetCallbackOnFrame(TRUE);
                        SetTimer(2,100,NULL);
                }
        }
}

/////////////////////////////////////////////////////////////////////////
// testView.h : interface of the CTestView class
//
/////////////////////////////////////////////////////////////////////////

#if !defined(AFX_TESTVIEW_H__3429FB6E_D438_11D3_AE9C_0000216DC645__INCLUDED_)
#define AFX_TESTVIEW_H__3429FB6E_D438_11D3_AE9C_0000216DC645__INCLUDED_

#if _MSC_VER > 1000
#pragma once
```

```
#endif // _MSC_VER > 1000
//VVVVVVVVVVVVVVVVVVVVVVVVVVVV
#include "VidCap.h"
//VVVVVVVVVVVVVVVVVVVVVVVVVVVV
class MyVidClass : public CVidCap
{
public:
private:
        CView* View;
public:
        void SetView(CView* V){View=V;}
        virtual void OnFrameCallback(LPVIDEOHDR lpVidHdr);

 //     CDataScatteringSpace* GetplotData(void){return VVVV;}
//      MyVidClass();
        };
class CTestView : public CView
{
//VVVVVVVVVVVVVVVVVVVVVVVVVVVV
protected:// create from serialization only
        CDataScatteringSpace* m_pDSS1;
private:
        CDib* BMP;
        CDib* Pic1;
//VVVVVVVVVVVVVVVVVVVVVVVVVVVV
protected: // create from serialization only
//  CDataScatteringSpace* VVV;
//      CDataScatteringSpace* VVVV;
        CTestView();
        DECLARE_DYNCREATE(CTestView)
 // Attributes
public:
        //CDataScatteringSpace* m_pDSS1;
        double *X,*Y,*Z;
        MyVidClass VidCap;
        CTestDoc* GetDocument();
        //CDataScatteringSpace* Data1;
        //CView* objTr1;
        CDib* GetBitmap(void){return BMP;}
        CDib* GetBitmap1(void){return Pic1;}
```

```
        CDataScatteringSpace* GetScattering(void){return m_pDSS1;}
// Operations
public:
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CTestView)
        public:
        virtual void OnDraw(CDC* pDC);  // overridden to draw this view
        virtual BOOL PreCreateWindow(CREATESTRUCT& cs);
        virtual void OnInitialUpdate();
        protected:
        virtual BOOL OnPreparePrinting(CPrintInfo* pInfo);
        virtual void OnBeginPrinting(CDC* pDC, CPrintInfo* pInfo);
        virtual void OnEndPrinting(CDC* pDC, CPrintInfo* pInfo);

// Implementation
public:
        virtual ~CTestView();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
protected:
// Generated message map functions
protected:
                afx_msg void OnOperate();
        afx_msg void OnTimer(UINT nIDEvent);
                DECLARE_MESSAGE_MAP()
};

#ifndef _DEBUG  // debug version in testView.cpp
inline CTestDoc* CTestView::GetDocument()
   { return (CTestDoc*)m_pDocument; }
#endif

////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
#endif // !defined(AFX_TESTVIEW_H__3429FB6E_D438_11D3_AE9C_0000216DC645__INCLUDED_)
```

```
////////////////////////////////////////////////////////////////////
//  TestDoc.cpp : implementation of the CTestDoc class
////////////////////////////////////////////////////////////////////
// testDoc.cpp
#include "stdafx.h"
#include "test.h"
#include "math.h"
#include "3DClass.h"
#include "CDib.h"
#include "CDibTrue.h"
#include "Camera.h"
#include "DataScatteringSpace.h"
#include "testDoc.h"
#ifdef _DEBUG
#define new DEBUG_NEW
#undef THIS_FILE
static char THIS_FILE[] = __FILE__;
#endif
////////////////////////////////////////////////////////////////////
// CTestDoc
IMPLEMENT_DYNCREATE(CTestDoc, CDocument)
BEGIN_MESSAGE_MAP(CTestDoc, CDocument)
        //{{AFX_MSG_MAP(CTestDoc)
                // NOTE - the ClassWizard will add and remove mapping macros here.
                //    DO NOT EDIT what you see in these blocks of generated code!
        //}}AFX_MSG_MAP
END_MESSAGE_MAP()
////////////////////////////////////////////////////////////////////
// CTestDoc construction/destruction
CTestDoc::CTestDoc()
{
        m_pDSS = new CDataScatteringSpace();
        double X,Y,Z;
        double Step = 0.049087385212340519350978802863742 2f;
//VVVVVVVVVVVVVVVVVVVVVVVVVVVVVV
        X=50;Y=80;Z=120;
        m_pDSS->AddSphere(  X,  Z,  Y,3.0f);
        X=80;Y=100;Z=220;
        m_pDSS->AddSphere(X,Z,Y,4.0f);
////////////////////////
```

```
/*
        for(i=0,Rad=0.0f;i<256;i+=2,Rad+=Step)
        {
                X = (double) i;
                Z = (double) i;
                Y = (double) i;
                m_pDSS->AddSphere(  X,  Z,  Y,3.0f);
// Make Cube
                m_pDSS->AddSphere(  X,  0.0f,  0.0f,1.0f);
                m_pDSS->AddSphere(  X,  0.0f,256.0f,1.0f);
                m_pDSS->AddSphere(  X,256.0f,  0.0f,1.0f);
                m_pDSS->AddSphere(  X,256.0f,256.0f,1.0f);

                m_pDSS->AddSphere(  0.0f,  Z,  0.0f,1.0f);
                m_pDSS->AddSphere(256.0f,  Z,  0.0f,1.0f);
                m_pDSS->AddSphere(  0.0f,  Z,256.0f,1.0f);
                m_pDSS->AddSphere(256.0f,  Z,256.0f,1.0f);

                m_pDSS->AddSphere(  0.0f,  0.0f,  Y,1.0f);
                m_pDSS->AddSphere(256.0f,  0.0f,  Y,1.0f);
                m_pDSS->AddSphere(  0.0f,256.0f,  Y,1.0f);
                m_pDSS->AddSphere(256.0f,256.0f,  Y,1.0f);
        }
        for(Y = 0.0f,Rad=0.0f;Y<=255.0f;Y+=0.1f,Rad+=Step)
        {
        // Make Skew
                Radian = sin(Rad/40.0f)*128.0f;
                X = 128.0f+ (sin(Rad)*Radian);
                Z = 128.0f+ (cos(Rad)*Radian);
                m_pDSS->AddSphere(X,Z,Y,4.0f);
        }
//////////////
*/
};
CTestDoc::~CTestDoc()
{
        delete m_pDSS;
}
BOOL CTestDoc::OnNewDocument()
```

```
{
	if (!CDocument::OnNewDocument())
		return FALSE;
	// TODO: add reinitialization code here
	// (SDI documents will reuse this document)
	return TRUE;
}
/////////////////////////////////////////////////////////////////////////////
// CTestDoc serialization
void CTestDoc::Serialize(CArchive& ar)
{
	if (ar.IsStoring())
	{
		// TODO: add storing code here
	}
	else
	{
		// TODO: add loading code here
	}
}
/////////////////////////////////////////////////////////////////////////////
// CTestDoc diagnostics
#ifdef _DEBUG
void CTestDoc::AssertValid() const
{
	CDocument::AssertValid();
}
void CTestDoc::Dump(CDumpContext& dc) const
{
	CDocument::Dump(dc);
}
#endif //_DEBUG
/////////////////////////////////////////////////////////////////////////////
// CTestDoc commands
/////////////////////////////////////////////////////////////////////////////
// testDoc.h : interface of the CTestDoc class
#if !defined(AFX_TESTDOC_H__3429FB6C_D438_11D3_AE9C_0000216DC645__INCLUDED_)
#define AFX_TESTDOC_H__3429FB6C_D438_11D3_AE9C_0000216DC645__INCLUDED_
#if _MSC_VER > 1000
#pragma once
```

```
#endif // _MSC_VER > 1000
class CTestDoc : public CDocument
{
protected: // create from serialization only
        CDataScatteringSpace* m_pDSS;
                CTestDoc();
        DECLARE_DYNCREATE(CTestDoc)
// Attributes
public:
// Operations
public:
// Overrides
        // ClassWizard generated virtual function overrides
        //{{AFX_VIRTUAL(CTestDoc)
        public:
        virtual BOOL OnNewDocument();
        virtual void Serialize(CArchive& ar);
        //}}AFX_VIRTUAL
// Implementation
public:
        virtual ~CTestDoc();
#ifdef _DEBUG
        virtual void AssertValid() const;
        virtual void Dump(CDumpContext& dc) const;
#endif
protected:
// Generated message map functions
protected:
        //{{AFX_MSG(CTestDoc)
                // NOTE - the ClassWizard will add and remove member functions here.
                //    DO NOT EDIT what you see in these blocks of generated code !
        //}}AFX_MSG
        DECLARE_MESSAGE_MAP()
public:
        CDib* GetView(void) { return m_pDSS->GetPic();}
};
/////////////////////////////////////////////////////////////////////////////
//{{AFX_INSERT_LOCATION}}
// Microsoft Visual C++ will insert additional declarations immediately before the previous line.
#endif // !defined(AFX_TESTDOC_H__3429FB6C_D438_11D3_AE9C_0000216DC645__INCLUDED_)
```

```
////////////////////////////////////////////////////////
// Cdib.cpp
////////////////////////////////////////////////////////
#include "stdafx.h"
#include "math.h"
#include "3DClass.h"
#include "cdib.h"
#include <windowsx.h>  // for GlobalAllocPtr
IMPLEMENT_SERIAL(CDib, CObject, 0)
////////////////////////////////////////////////////////
CDib::CDib()
{
   m_dwLength = 0L;
   m_nBits    = 0;
   m_lpBuf    = NULL;
}

BOOL CDib::MakeDib(long x,long y,int Bits)
{
        DWORD Length;

        long LUTsize;
        long Width;
        switch(Bits)
        {
                case 24:
                        LUTsize = 0;
                        Width = x*3L;
                        break;
                case 8:
                        LUTsize = sizeof(RGBQUAD)*256L;
                        Width = x;
                        break;
                default: return FALSE;
        }
        if(Width&3) Width = (Width&0xfffffc)+4;

        Length = sizeof(BITMAPFILEHEADER) + sizeof(BITMAPINFOHEADER) + LUTsize + (Width * y);
        m_nBits = Bits;
        m_dwLength = Length;
```

```
    if(!AllocateMemory()) return FALSE;

    m_lpBMFH->bfType = 0x4d42;
    m_lpBMFH->bfSize = Length;
    m_lpBMFH->bfReserved1 = 0;
    m_lpBMFH->bfReserved2 = 0;
    m_lpBMFH->bfOffBits = sizeof(BITMAPFILEHEADER) + LUTsize +sizeof(BITMAPINFOHEADER);

    m_lpBMIH->biSize = sizeof(BITMAPINFOHEADER);
    m_lpBMIH->biWidth = x;
    m_lpBMIH->biHeight = y;
    m_lpBMIH->biPlanes = 1;
    m_lpBMIH->biBitCount = Bits;
    m_lpBMIH->biCompression = BI_RGB;
    if(Bits==24) m_lpBMIH->biSizeImage = Width*y;
    else m_lpBMIH->biSizeImage = 0;
    m_lpBMIH->biXPelsPerMeter = 0;
    m_lpBMIH->biYPelsPerMeter = 0;
    m_lpBMIH->biClrUsed = 0;
    m_lpBMIH->biClrImportant = 0;

    m_lpData = m_lpBuf+m_lpBMFH->bfOffBits;
    m_lpPic = (unsigned char*) m_lpData;
    return TRUE;
}

////////////////////////////////////////////////////////////////////
CDib::~CDib()
{
  if (m_lpBuf) {
    GlobalFreePtr(m_lpBuf); // free the DIB memory
  }
}

////////////////////////////////////////////////////////////////////
void CDib::Serialize(CArchive& ar)
{
  WORD wBM;
  CObject::Serialize(ar);
  if (ar.IsStoring()) {
```

```
    ar.GetFile()->WriteHuge(m_lpBuf, m_dwLength);
  }
  else {
                    if(m_dwLength!=0)
                                DeleteData();
    ASSERT(m_dwLength == 0L); // DIB must be empty
    ar >> wBM;
    if (wBM == (WORD) 0x4d42) {
        ar.Flush();
        ar.GetFile()->SeekToBegin();
        m_dwLength = ar.GetFile()->GetLength();
                                if(m_dwLength > 0L)
                                {
            if (!AllocateMemory()) {
            return;
          }
                                    ar.Flush();
        ar.GetFile()->ReadHuge(m_lpBuf, m_dwLength);
            m_lpData = (unsigned char*) m_lpBMFH + m_lpBMFH->bfOffBits;
                                    m_lpPic = (unsigned char*) m_lpBMFH + m_lpBMFH->bfOffBits;
          m_nBits = m_lpBMIH->biBitCount;
      }
    }
    else {
      AfxMessageBox("Invalid DIB archive");
      m_dwLength = 0L;
      m_nBits = 0;
    }
  }
}

////////////////////////////////////////////////////////////
BOOL CDib::Display(CDC* pDC, CPoint origin)
{
  // direct to device--bypass the GDI bitmap
  if (!m_lpBuf) {
    return FALSE; // nothing to display
  }
  if (!::SetDIBitsToDevice(pDC->GetSafeHdc(), origin.x, origin.y,
    (WORD) m_lpBMIH->biWidth, (WORD) m_lpBMIH->biHeight, 0, 0, 0,
```

```
    (WORD) m_lpBMIH->biHeight, m_lpData, m_lpBMI,
    DIB_RGB_COLORS)) {
     return FALSE;
   }
   return TRUE;
}

////////////////////////////////////////////////////////////
BOOL CDib::Stretch(CDC* pDC, CPoint origin, CSize size)
{
   // direct to device--bypass the GDI bitmap
   if (!m_lpBuf) {
     return FALSE; // nothing to display
   }
   if (!::StretchDIBits(pDC->GetSafeHdc(), origin.x, origin.y,
     size.cx, size.cy, 0, 0, (WORD) m_lpBMIH->biWidth,
     (WORD) m_lpBMIH->biHeight, m_lpData, m_lpBMI,
     DIB_RGB_COLORS, SRCCOPY)) {
     return FALSE;
   }
   return TRUE;
}

////////////////////////////////////////////////////////////
int CDib::GetColorBits()
{
   return m_nBits;
}

////////////////////////////////////////////////////////////
DWORD CDib::GetLength()
{
   return m_dwLength;
}

////////////////////////////////////////////////////////////
BOOL CDib::AllocateMemory(BOOL bRealloc) // bRealloc default = FALSE
{
        if (bRealloc) {
                m_lpBuf = (unsigned char*) GlobalReAllocPtr(m_lpBuf,
```

```
                m_dwLength, GHND);
        }
        else {
                m_lpBuf = (unsigned char*) GlobalAllocPtr(GHND, m_dwLength);
        }
        if (!m_lpBuf) {
                AfxMessageBox("Unable to allocate DIB memory");
                m_dwLength = 0L;
                m_nBits = 0;
                return FALSE;
        }
        m_lpBMFH = (LPBITMAPFILEHEADER) m_lpBuf;
        m_lpBMIH = (LPBITMAPINFOHEADER) (m_lpBuf + sizeof(BITMAPFILEHEADER));
        m_lpBMI = (LPBITMAPINFO) m_lpBMIH;
        return TRUE;
}

void CDib::ClearPic(void)
{
        long dwSize = m_dwLength-m_lpBMFH->bfOffBits;
        long i;
        for(i=0;i<dwSize;i++)
                m_lpPic[i]=0;
}
void CDib::WhiteBackground(void)
{
        long dwSize = m_dwLength-m_lpBMFH->bfOffBits;
        long i;
        for(i=0;i<dwSize;i++)
                m_lpPic[i]= 255;
}
void CDib::SetGrayBackground(unsigned char Level)
{
        long dwSize = m_dwLength-m_lpBMFH->bfOffBits;
        FillMemory(m_lpPic,dwSize,Level);
}

void CDib::DeleteData(void)
{
        GlobalFreePtr(m_lpBuf);
```

```
        m_lpBuf = NULL;
        m_dwLength = 0;
}

void CDib::CopyData(CDib* Src)
{
        if(m_dwLength == 0)
        {
                m_dwLength = Src->m_dwLength;
                AllocateMemory();
        }
        else
        {
                m_dwLength = Src->m_dwLength;
                AllocateMemory(TRUE);
        }
        CopyMemory(m_lpBuf,Src->m_lpBuf,m_dwLength);
}

long CDib::GetXBufferSize(void)
{
        long Width;
        switch(m_nBits)
        {
                case 24:
                        Width = (m_lpBMIH->biWidth)*3L;
                        break;
                case 8:
                        Width = m_lpBMIH->biWidth;
                        break;
        }
        if(Width&3) Width = (Width&0xfffffc)+4;
        return Width;
}
////////////////////////////////////////////////
// cdib.h
////////////////////////////////////////////////
class CDib : public CObject
{
  DECLARE_SERIAL(CDib)
```

```
protected:
        unsigned char* m_lpBuf;
  DWORD     m_dwLength; // total buffer length, including file header
  int       m_nBits;   // number of color bits per pixel
  //pointers for internal use
  LPBITMAPFILEHEADER m_lpBMFH;
  LPBITMAPINFOHEADER m_lpBMIH;
  LPBITMAPINFO      m_lpBMI;
  unsigned char* m_lpData;
        unsigned char* m_lpPic;
        MakeDib(long x,long y,int Bits);
public:
  CDib();
  ~CDib();
  virtual void Serialize(CArchive &ar);
  void SetLength(DWORD Length) { m_dwLength = Length; }
  BOOL Display(CDC*, CPoint origin);
  BOOL Stretch(CDC*, CPoint origin, CSize size);
  int GetColorBits();  // bits per pixel
        long GetXBufferSize(void);
  DWORD GetLength();
        BOOL It24Bit(void){
        if(m_nBits==24) return TRUE;
        else return FALSE;
  }
        long ReceiveWidth(void) { return m_lpBMIH->biWidth;}
        long ReceiveHeight(void) { return m_lpBMIH->biHeight;}
        unsigned char* ReceivePic(void) { return m_lpPic;}
  LPBITMAPINFOHEADER ReceiveBMPInfoHeader(void) { return m_lpBMIH;}
  LPBITMAPFILEHEADER ReceiveBMPFileHeader(void) { return m_lpBMFH;}
  LPBITMAPINFO ReceiveBMI(void) { return m_lpBMI;}
  unsigned char* ReceiveBuf(void) { return (unsigned char*) m_lpBuf;}
        void ClearPic(void);
        void WhiteBackground(void);
        void SetGrayBackground(unsigned char Level);
        void DeleteData(void);
        void CopyData(CDib* Src);
private:
  BOOL AllocateMemory(BOOL bRealloc = FALSE);
};
```

```
////////////////////////////////////////
#include "stdafx.h"
#include "math.h"
#include "3DClass.h"
C3DOperator::C3DOperator()
{
        dA11 = 1.0f; dA12 = 0.0f; dA13 = 0.0f; dA14 = 0.0f;
        dA21 = 0.0f; dA22 = 1.0f; dA23 = 0.0f; dA24 = 0.0f;
        dA31 = 0.0f; dA32 = 0.0f; dA33 = 1.0f; dA34 = 0.0f;
        dA41 = 0.0f; dA42 = 0.0f; dA43 = 0.0f; dA44 = 1.0f;
}

C3DOperator::~C3DOperator()
{
}

void C3DOperator::RotateXYPlaneRad(double dRad)
{
        double dCos = cos(dRad);
        double dSin = sin(dRad);
        double Temp = dA11;
        dA11 = (dCos*dA11) - (dSin*dA12);
        dA12 = (dSin*Temp) + (dCos*dA12);

        Temp = dA21;
        dA21 = (dCos*dA21) - (dSin*dA22);
        dA22 = (dSin*Temp) + (dCos*dA22);

        Temp = dA31;
        dA31 = (dCos*dA31) - (dSin*dA32);
        dA32 = (dSin*Temp) + (dCos*dA32);

        Temp = dA41;
        dA41 = (dCos*dA41) - (dSin*dA42);
```

```
	dA42 = (dSin*Temp) + (dCos*dA42);
}

void C3DOperator::RotateXYPlaneDeg(double dDeg)
{
	double dRad;
	dRad = dDeg*3.1415926535f/180.0f;
	double dCos = cos(dRad);
	double dSin = sin(dRad);
	double Temp = dA11;
	dA11 = (dCos*dA11) - (dSin*dA12);
	dA12 = (dSin*Temp) + (dCos*dA12);

	Temp = dA21;
	dA21 = (dCos*dA21) - (dSin*dA22);
	dA22 = (dSin*Temp) + (dCos*dA22);

	Temp = dA31;
	dA31 = (dCos*dA31) - (dSin*dA32);
	dA32 = (dSin*Temp) + (dCos*dA32);

	Temp = dA41;
	dA41 = (dCos*dA41) - (dSin*dA42);
	dA42 = (dSin*Temp) + (dCos*dA42);
}

void C3DOperator::RotateXZPlaneRad(double dRad)
{
	double dCos = cos(dRad);
	double dSin = sin(dRad);
	double Temp = dA11;
	dA11 = (dCos*dA11) + (dSin*dA13);
	dA13 = (dCos*dA13) - (dSin*Temp);
```

```
        Temp = dA21;
        dA21 = (dCos*dA21) + (dSin*dA23);
        dA23 = (dCos*dA23) - (dSin*Temp);

        Temp = dA31;
        dA31 = (dCos*dA31) + (dSin*dA33);
        dA33 = (dCos*dA33) - (dSin*Temp);

        Temp = dA41;
        dA41 = (dCos*dA41) + (dSin*dA43);
        dA43 = (dCos*dA43) - (dSin*Temp);
}

void C3DOperator::RotateXZPlaneDeg(double dDeg)
{
        double dRad;
        dRad = dDeg*3.1415926535f/180.0f;
        double dCos = cos(dRad);
        double dSin = sin(dRad);
        double Temp = dA11;
        dA11 = (dCos*dA11) + (dSin*dA13);
        dA13 = (dCos*dA13) - (dSin*Temp);

        Temp = dA21;
        dA21 = (dCos*dA21) + (dSin*dA23);
        dA23 = (dCos*dA23) - (dSin*Temp),

        Temp = dA31;
        dA31 = (dCos*dA31) + (dSin*dA33);
        dA33 = (dCos*dA33) - (dSin*Temp);

        Temp = dA41;
        dA41 = (dCos*dA41) + (dSin*dA43);
        dA43 = (dCos*dA43) - (dSin*Temp);
```

```
}

void C3DOperator::RotateYZPlaneRad(double dRad)
{
        double dCos = cos(dRad);
        double dSin = sin(dRad);
        double Temp = dA12;
        dA12 = (dCos*dA12) - (dSin*dA13);
        dA13 = (dSin*Temp) + (dCos*dA13);

        Temp = dA22;
        dA22 = (dCos*dA22) - (dSin*dA23);
        dA23 = (dSin*Temp) + (dCos*dA22);

        Temp = dA32;
        dA32 = (dCos*dA32) - (dSin*dA33);
        dA33 = (dSin*Temp) + (dCos*dA33);

        Temp = dA42;
        dA42 = (dCos*dA42) - (dSin*dA43);
        dA43 = (dSin*Temp) + (dCos*dA43);
}

void C3DOperator::RotateYZPlaneDeg(double dDeg)
{
        double dRad = dDeg*3.1415926535f/180.0f;
        double dCos = cos(dRad);
        double dSin = sin(dRad);
        double Temp = dA12;
        dA12 = (dCos*dA12) - (dSin*dA13);
        dA13 = (dSin*Temp) + (dCos*dA13);

        Temp = dA22;
        dA22 = (dCos*dA22) - (dSin*dA23);
```

```
        dA23 = (dSin*Temp) + (dCos*dA22);

        Temp = dA32;
        dA32 = (dCos*dA32) - (dSin*dA33);
        dA33 = (dSin*Temp) + (dCos*dA33);

        Temp = dA42;
        dA42 = (dCos*dA42) - (dSin*dA43);
        dA43 = (dSin*Temp) + (dCos*dA43);
}

void C3DOperator::Translator(double dX,double dY,double dZ)
{
        dA11 = dA11 + (dX * dA14);
        dA12 = dA12 + (dY * dA14);
        dA13 = dA13 + (dZ * dA14);

        dA21 = dA21 + (dX * dA24);
        dA22 = dA22 + (dY * dA24);
        dA23 = dA23 + (dZ * dA24);

        dA31 = dA31 + (dX * dA34);
        dA32 = dA32 + (dY * dA34);
        dA33 = dA33 + (dZ * dA34);

        dA41 = dA41 + (dX * dA44);
        dA42 = dA42 + (dY * dA44);
        dA43 = dA43 + (dZ * dA44);
}

void C3DOperator::Scale(double dXScale,double dYScale,double dZScale)
{
        dA11 = dA11 * dXScale;
        dA12 = dA12 * dYScale;
```

```
        dA13 = dA13 * dZScale;

        dA21 = dA21 * dXScale;
        dA22 = dA22 * dYScale;
        dA23 = dA23 * dZScale;

        dA31 = dA31 * dXScale;
        dA32 = dA32 * dYScale;
        dA33 = dA33 * dZScale;

        dA41 = dA41 * dXScale;
        dA42 = dA42 * dYScale;
        dA43 = dA43 * dZScale;
}


C3DVector::C3DVector()
{

}

C3DVector::~C3DVector()
{

}

void C3DVector::Transform(C3DOperator Operator)
{
        dX = (dX*Operator.dA11)+(dY*Operator.dA21)+(dZ*Operator.dA31)+Operator.dA41;
        dY = (dX*Operator.dA12)+(dY*Operator.dA22)+(dZ*Operator.dA32)+Operator.dA42;
        dZ = (dX*Operator.dA13)+(dY*Operator.dA23)+(dZ*Operator.dA33)+Operator.dA43;
}
```

```
//////////////////////
class C3DOperator
{
        friend class C3DVector;
private:
        double dA11, dA21, dA31, dA41;
        double dA12, dA22, dA32, dA42;
        double dA13, dA23, dA33, dA43;
        double dA14, dA24, dA34, dA44;
public:
        C3DOperator();
        ~C3DOperator();
        void RotateXYPlaneRad(double dRad);
        void RotateXYPlaneDeg(double dDeg);
        void RotateXZPlaneRad(double dRad);
        void RotateXZPlaneDeg(double dDeg);
        void RotateYZPlaneRad(double dRad);
        void RotateYZPlaneDeg(double dDeg);
        void Translator(double dX,double dY,double dZ);
        void Scale(double dXScale,double dYScale,double dZScale);
};

class C3DVector
{
private:
        double dX, dY, dZ;
public:
        C3DVector();
        C3DVector(double x,double y, double z);
        ~C3DVector();
        C3DVector operator+(const C3DVector Value) const;
        void operator+=(const C3DVector Value);
        void operator-=(const C3DVector Value);
```

```
        void operator*=(const double Mul);
        void operator/=(const double devide);
        void operator=(const C3DVector Value);
        BOOL operator!=(const C3DVector Value) const;
        BOOL operator==(const C3DVector Value) const;
        BOOL IsZeroVector(void) const;
        C3DVector operator-(const C3DVector Value) const;
        C3DVector operator*(const double Mul) const; // scale vector
        C3DVector operator*(const C3DVector Mul) const; // Cross Product
        C3DVector operator/(const double Devide) const; // devide vector
        double InnerProduct(const C3DVector Value) const;
        C3DVector CrossProduct(const C3DVector Value) const;
        void Normal(void); // return normal vector
        void ChangeToNormal(void); // change to normal vector
        double Length(void); // return length of vector
        double Length_2(void); // return length square of vector
        void XYAngleYZAngle(double& XY,double& YZ);
        void Transform(C3DOperator Operator);
        double GetX(void) { return dX;}
        double GetY(void) { return dY;}
        double GetZ(void) { return dZ;}
        void MakeVector(double x,double y,double z);
        void RotateXYPlane(double dRad);
        void RotateZXPlane(double dRad);
        void RotateYZPlane(double dRad);
};

inline void C3DVector::MakeVector(double x,double y,double z)
{
        dX = x;
        dY = y;
        dZ = z;
}
```

```
inline C3DVector::C3DVector(double x,double y, double z) :
dX(x), dY(y), dZ(z)
{
}

inline C3DVector C3DVector::operator+(const C3DVector Value) const
{
        return C3DVector(dX+Value.dX,dY+Value.dY,dZ+Value.dZ);
}

inline void C3DVector::operator=(const C3DVector Value)
{
        dX = Value.dX;
        dY = Value.dY;
        dZ = Value.dZ;
}

inline BOOL C3DVector::operator==(const C3DVector Value) const
{
        if(dX!=Value.dX) return FALSE;
        if(dY!=Value.dY) return FALSE;
        if(dZ!=Value.dZ) return FALSE;
        return TRUE;
}

inline BOOL C3DVector::operator!=(const C3DVector Value) const
{
        if(dX!=Value.dX) return TRUE;
        if(dY!=Value.dY) return TRUE;
        if(dZ!=Value.dZ) return TRUE;
        return FALSE;
}

inline void C3DVector::operator+=(const C3DVector Value)
```

```
{
        dX += Value.dX;
        dY += Value.dY;
        dZ += Value.dZ;
}

inline void C3DVector::operator-=(const C3DVector Value)
{
        dX -= Value.dX;
        dY -= Value.dY;
        dZ -= Value.dZ;
}

inline void C3DVector::operator*=(const double Mul)
{
        dX *= Mul;
        dY *= Mul;
        dZ *= Mul;
}

inline void C3DVector::operator/=(const double Devide)
{
        dX /= Devide;
        dY /= Devide;
        dZ /= Devide;
}

inline C3DVector C3DVector::operator-(const C3DVector Value) const
{
        return C3DVector(dX-Value.dX,dY-Value.dY,dZ-Value.dZ);
}

inline C3DVector C3DVector::operator*(const double Mul) const
{
```

```
	return C3DVector(dX*Mul,dY*Mul,dZ*Mul);
}

inline C3DVector C3DVector::operator*(const C3DVector Mul) const
{
	return C3DVector((dY*Mul.dZ)-(Mul.dY*dZ),(dZ*Mul.dX)-(Mul.dZ*dX),(dX*Mul.dY)-
(Mul.dX*dY));
}

inline C3DVector C3DVector::operator/(const double Devide) const
{
	return C3DVector(dX/Devide,dY/Devide,dZ/Devide);
}

inline C3DVector C3DVector::CrossProduct(const C3DVector Value) const
{
	double X = (dY*Value.dZ)-(dZ*Value.dY);
	double Y = (dZ*Value.dX)-(dX*Value.dZ);
	double Z = (dX*Value.dY)-(dY*Value.dX);
	return C3DVector(X,Y,Z);
}

inline double C3DVector::InnerProduct(const C3DVector Value) const
{
	return ((dX*Value.dX)+(dY*Value.dY)+(dZ*Value.dZ));
}

inline void C3DVector::Normal(void)
{
	double Length = sqrt((dX*dX)+(dY*dY)+(dZ*dZ));
	dX /= Length;
	dY /= Length;
	dZ /= Length;
}
```

```
inline void C3DVector::ChangeToNormal(void) // change to normal vector
{
        double Length = sqrt((dX*dX)+(dY*dY)+(dZ*dZ));
        dX /= Length;
        dY /= Length;
        dZ /= Length;
}

inline double C3DVector::Length(void) // return length of vector
{
        return sqrt((dX*dX)+(dY*dY)+(dZ*dZ));
}

inline double C3DVector::Length_2(void)
{
        return (dX*dX) + (dY*dY) + (dZ*dZ);
}

inline void C3DVector::XYAngleYZAngle(double& XY,double& YZ)
{
        YZ = atan2(dZ,dY);
        XY = atan2(sqrt((dZ*dZ)+(dY*dY)),dX);
}

inline void C3DVector::RotateXYPlane(double dRad)
{
        double dXTemp = dX;

        dX = dXTemp*cos(dRad) + dY*sin(dRad);
        dY =-dXTemp*sin(dRad) + dY*cos(dRad);
}

inline void C3DVector::RotateZXPlane(double dRad)
```

```
{
        double dZTemp = dZ;

        dZ = dZTemp*cos(dRad) + dX*sin(dRad);
        dZ =-dZTemp*sin(dRad) + dX*cos(dRad);
}

inline void C3DVector::RotateYZPlane(double dRad)
{
        double dYTemp = dY;

        dY = dYTemp*cos(dRad) + dZ*sin(dRad);
        dZ =-dYTemp*sin(dRad) + dZ*cos(dRad);
}

inline BOOL C3DVector::IsZeroVector(void) const
{
        if(dX==0.0f && dY==0.0f && dZ==0.0f)
                return TRUE;
        return FALSE;
}
```

```
////////////////////////////////////////////////
//CdibTrue.cpp
////////////////////////////////////////////////
#include "stdafx.h"
#include "math.h"
#include "3DClass.h"
#include "cdib.h"
#include "cdibtrue.h"

CDibTrue::CDibTrue(long x,long y)
{
        MakeDib(x,y,24);
}

CDibTrue::~CDibTrue()
{
}

////////////////////////////////////////////////
// CdibTrue.h
////////////////////////////////////////////////

class CDibTrue : public CDib
{
public:
        CDibTrue(long x,long y);
        ~CDibTrue();
};
```

กิตติกรรมประกาศ

การดำเนินงานทำโครงงานนี้ผู้จัดทำได้รับคำแนะนำและความช่วยเหลือจากอาจารย์ที่ปรึกษาและรุ่นพี่เป็นอย่างดี ขอขอบคุณภาคเครื่องกล ซึ่งได้เอื้อเฟื้อสถานที่และเครื่องมือในการทำงาน และขอขอบคุณผู้มีพระคุณอีกหลายท่านที่มีส่วนเกี่ยงข้องจนโครงงานนี้สำเร็จลุล่วงไปได้ด้วยดี

(นายจิรพงศ์ บัวเลิศ)

(นายทวีพล กนกเจิดกุล)

(นายศิริพันธุ์ ส่งชื่น)

ผู้จัดทำ

เอกสารอ้างอิง

1 "Color Quick Cam User Guid" ,Connectix Corporation , 1996

2 อนิรุต ลิ่วหาทอง , "การเขียนโปรแกรมบนวินโดวส์ด้วย Microsoft C ++" , ซีเอ็ดยูเคชั่น จำกัด (มหาชน) , 2539

3 Scott Stanfield with Ralph Arveson , " VISUAL C++ How-To " , Waite Group Press ™ A Division of Sams Publishing Corte Madera , CA , 1996

4 Brain W. Kerighan and Dennis M. Rithie , "The C Programming Language" , Bell Laboratories Murray Hill , New Jersey

5 David J.Kruglinsik, "Inside Visual C+" , Microsolf Press , fourth edition,1997.

6 Steve Rimmerm" WINDOWS BITMAPPED GRAPHICS" , WINDCREST/McGraw-Hill,1993

10 C.Wayne Brown and Barry J.Shepherd, "Graphics File Formats reference and guide" ,Manning Publication Co. , 1995

7 David F.Rogers & J.Alams ,"Mathematical Elements for Computer Grapics" ,Mcgraw-Hill Publishing Company , second edition ,1989

8 Keith Rule, "3D GRAPHICS FILE FORMATS " ADDISION-WESLEY DEVELOPERS PRESS,1996

9 Jeff Prosise, "Programming Windows95 with MFC ", Microsoft Press,1996