การพัฒนาโปรแกรมเพื่อการสร้างภาพวิถีการเคลื่อนที่ของทรงกลมแบบสามมิติจากภาพถ่ายวีดีโอ

Development of a Program for 3D Spherical Object Animation Using Video Images

โดย

ปรียาภา จินดาเลิศอุดมดี

รหัส 46066206

อาจารย์ที่ปรึกษา

รศ. ดร. นพพร โชติกกำธร

รายงานนี้เป็นส่วนหนึ่งของวิชาโครงการพัฒนาระบบงาน

หลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ

ภาคฤดูร้อน ปีการศึกษา 2547

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ชื่อหัวข้อ	การพัฒนาโปรแกรมเพื่อการสร้างภาพวิถีการเคลื่อนที่ของทรงกลมแบบสามมิติจากภาพถ่ายวีดีโอ
นักศึกษา	นางสาวปรียาภา จินดาเลิศอุดมดี
อาจารย์ที่ปรึกษา	รศ.ดร.นพพร โชติกกำธร
ระดับการศึกษา	วิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
แขนงวิชา	วิทยาการสารสนเทศ
ปีการศึกษา	2547

บทคัดย่อ

รายงานฉบับนี้นำเสนอผลการพัฒนาโปรแกรมสร้างภาพวิถีการเคลื่อนที่ของวัตถุทรงกลมแบบสามมิติในหลากหลายมุมมอง โดยอาศัยข้อมูลที่ได้จากกล้องวีดีโอเพียงตัวเดียว ในการพัฒนานั้นใช้หลักการและวิธีการทางด้านการประมวลผลภาพมาทำการวิเคราะห์ข้อมูลจากรูปภาพ โดยข้อมูลที่ได้มานั้นจะถูกสร้างเป็นภาพเคลื่อนไหวโดยอาศัยกระบวนการสร้างภาพด้วยคอมพิวเตอร์กราฟฟิค

ขั้นตอนการทำงานของโปรแกรมประกอบไปด้วย 3 ขั้นตอนหลัก ขั้นตอนแรกเป็นการค้นหาตำแหน่งของ Marker และนำข้อมูลมาสร้าง Homography Matrix ขั้นตอนที่สองทำการวิเคราะห์ภาพเพื่อค้นหาวัตถุทรงกลม และ จุดศูนย์กลางของทรงกลมที่อยู่ในภาพ โดยข้อมูลตำแหน่งของจุดศูนย์กลางทรงกลมนั้นจะถูกนำมาทำการประมาณตำแหน่งจุดศูนย์กลางของทรงกลมใน World Coordinate โดยอาศัย Homography Matrix ที่ได้จากขั้นตอนแรก ขั้นตอนที่สามเป็นการนำค่าตำแหน่งที่ได้คำนวณมา ผ่านกระบวนการ Cosine Interpolation เพื่อประมาณเส้นทางการเคลื่อนที่ของทรงกลม

โปรแกรมที่นำเสนอนั้นถูกพัฒนาด้วยคอมไพเลอร์ Microsoft Visual C++ 6.0 การพัฒนาส่วนของการประมวลผลภาพจะทำงานร่วมกับ TLIB และสร้างภาพเคลื่อนไหวด้วย OpenGL ไลบรารี โดยทำการทดสอบโปรแกรมภายใต้ระบบปฏิบัติการ Microsoft Windows XP

Title	Development of a Program for 3D Spherical Object Animation Using Video Images
Student	Miss Preeyapa Jindalertudomdee
Advisor	Assoc. Prof. Dr. Nopporn Chotikakamthorn
Level of Study	Master of Science in Information Technology
Major	Information Science
Academic Year	2004

ABSTRACT

This report presents the result of development program for 3D spherical object animation with various view points using information from only one video camera. The system applies image-processing theories and methods to analyze the image data. The estimated object position is used to create the animation using computer graphic techniques.

The program is divided into three major parts. In the first, the program finds the positions of makers, and then creates the homography matrix from this information. In the second part, the program analyzes video image to search for a spherical and finds the centroid of the object. Then, the centroid is used to estimate the center of the sphere in the world-coordinate system using the homograhy matrix as obtained from the first step. Finally, the object position are interpolated using the cosine-interpolation techniques.

This program was developed using Microsoft Visual C++ 6.0 compiler. The TLIB library was used in the image analysis part, and OpenGL library was used for graphic rendering. The program was tested on Microsoft Windows XP.

กิตติกรรมประกาศ

ขอกราบขอบพระคุณ รศ. ดร. นพพร โชติกกำธร อาจารย์ปรึกษาโครงงาน ที่กรุณาให้ความรู้ คำปรึกษา และคำแนะนำต่างๆ ในการพัฒนาโครงงานนี้ให้สำเร็จลุล่วงไปด้วยดี

ขอกราบขอบพระคุณคณาจารย์ทุกท่านในคณะเทคโนโลยีสารสนเทศที่กรุณาถ่ายทอดความรู้ และเทคนิคต่างๆ ในการศึกษา ตลอดระยะเวลาที่ศึกษาในสถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ขอขอบคุณพี่ๆ และเพื่อนๆ ในห้องปฏิบัติการวิจัยสื่อประสมและระบบเสมือน และคณะเทคโนโลยีสารสนเทศทุกคนที่คอยให้คำปรึกษา คำแนะนำ และกำลังใจในยามที่มีปัญหา

สุดท้ายขอกราบขอบพระคุณคุณพ่อ คุณแม่ และบุคคลในครอบครัว ที่ให้กำลังใจและให้การอุปการะมาโดยตลอด ขอกราบขอบพระคุณเป็นอย่างสูง

คุณความดีและประโยชน์ที่ได้รับจากโครงงานนี้ ผู้เขียนขอมอบแด่ผู้มีพระคุณทุกท่าน

ปรียาภา จินดาเลิศอุดมดี

พฤษภาคม 2548

สารบัญ

หน้า

สารบัญ (ต่อ)

หน้า

สารบัญตาราง

หน้า

ตารางที่

สารบัญภาพ

หน้า

ภาพที่

สารบัญแผนภาพ

หน้า

แผนภาพที่

บทที่ 1
บทนำ

1.1 ความเป็นมาของปัญหา

ปัจจุบันเทคโนโลยีทางด้าน image processing และ computer vision นั้นมีวิวัฒนาการและเข้ามามีบทบาทในชีวิตประจำวันของมนุษย์เรามากยิ่งขึ้น โดยได้รับการพัฒนาไปในหลากหลายรูปแบบของการใช้งานไม่ว่าจะเป็นทางด้านการศึกษาซึ่งจะเน้นไปทางด้านการศึกษาทางวิทยาศาสตร์และการแพทย์ อย่างเช่นในการตรวจสอบ DNA และด้านอื่นๆ อีกมากมายเพื่อเพิ่มความประสิทธิภาพในการประกอบการทำงานในด้านดังกล่าว โดนการทำงานด้านนี้ยังได้รับการพัฒนาอย่างไม่หยุดนิ่งและมีการพัฒนาอยู่ตลอดเวลา โดยได้ทำการผนวกความก้าวหน้าทางเทคโนโลยีและเทคนิคทางด้าน computer graphic ไปรวมเข้าไว้ด้วยกันเพื่อตอบสนองความต้องการที่เพิ่มมากขึ้น อย่างเช่น เกมส์คอมพิวเตอร์ซึ่งได้นำกล้องวีดีโอถ่ายการเคลื่อนไหวของมนุษย์ซึ่งจะมีการติดตั้งอุปกรณ์บางชนิด ตัวอย่างเช่น ถุงมือ (Cyber Glove) แว่นตา (Cyber Eyes) และอุปกรณ์จำลองแรงสัมผัส (Force Feedback Device) เป็นต้น เพื่อที่จะทำให้สามารถรับรู้ได้ว่าเคลื่อนไหวไปในทิศทางใดและนำข้อมูลที่ได้มาทำการสร้างเป็นภาพสามมิติซึ่งเลียนแบบการเคลื่อนไหวของมนุษย์ ได้อย่างถูกต้องและเป็นธรรมชาติ

สำหรับในอุตสาหกรรมด้านอื่นนั้นก็มีการนำวิวัฒนาการทางด้านนี้มาก่อให้เกิดประโยชน์เช่นกัน โดยมีการนำเทคโนโลยีไปใช้ในโรงงานเพื่อการผลิตที่ว่องไว รวมไปถึงการนำไปใช้ในอุตสาหกรรมการนำเสนอในรูปแบบต่างๆ ตัวอย่างเช่น การนำเสนอให้งานทางด้านกราฟฟิคมีความน่าสนใจมากยิ่งขึ้นและยังคงความถูกต้องของข้อมูลไว้อย่างครบถ้วน นอกจากนี้กระบวนการทางด้านนี้นั้นถูกใช้อย่างแพร่หลายในงานวิจัยอีกมากมาย

จากแนวความคิดข้างต้น ทำให้เกิดแนวความคิดในการพัฒนาการสร้างภาพวิถีการเคลื่อนที่ของทรงกลมแบบสามมิติจากภาพถ่ายวีดีโอซึ่งในโครงการนี้จะกล่าวถึงส่วนของการคำนวณค่าที่จำเป็นต่างๆ จากข้อมูลที่ได้จากกล้องวีดีโอ และนำข้อมูลไปสร้างเป็นงานทางด้านกราฟฟิคโดยเลือกการสร้างโปรแกรมที่จำลองกีฬาโบว์ลิ่ง สำหรับรายละเอียดในการออกแบบและโครงสร้างของโปรแกรมจะได้กล่าวในบทต่อไป

1.2 วัตถุประสงค์

1. โครงการนี้ได้ถูกพัฒนาขึ้นเพื่อศึกษาวิธีการพัฒนาโครงการอย่างมีระบบ ตามหลักการการพัฒนาระบบงาน

2. นำความรู้ที่ได้ศึกษามาประยุกต์ใช้และแก้ไขปัญหาในระบบงานที่พัฒนา

3. จำลองกีฬาการเล่นโบว์ลิ่ง เพื่อนำไปเป็นพื้นฐานสำหรับการพัฒนาระบบงาน

4. สร้างงานทางด้านกราฟฟิคด้วยข้อมูลที่ถูกต้องและน่าเชื่อถือได้ และมีความน่าสนใจในตัวชิ้นงานนั้นๆเพิ่มมากขึ้น

1.3 แนวทางการศึกษา

1. ศึกษาทฤษฎีเกี่ยวกับ Image Processing และ computer vision รวมไปถึง computer graphic เพื่อนำไปพัฒนาโปรแกรมที่สร้างขึ้น

2. ศึกษาอัลกอริทึมในการนำข้อมูลออกจากภาพที่มีและนำไปสร้างใหม่

2. วิเคราะห์และออกแบบโครงสร้างข้อมูล

5. พัฒนาโปรแกรม

6. ทดสอบโปรแกรม

6. สรุปผลและจัดทำเอกสารประกอบโครงการ

1.4 สภาพแวดล้อมในการดำเนินโครงการ

1. ประมวลผลบนซีพียู 2.4 GHz Pentium 4

2. หน่วยความจำขนาด 256 MB

3. ระบบปฏิบัติการ Windows XP

4. การพัฒนาและทดสอบอัลกอริทึมบนโปรแกรม MatLab 6.5 , Visual C++ 6.0

5. ทำการติดตั้งไลบรารี TLIB สำหรับทำการประมวลผลเกี่ยวกับไฟล์วีดีโอ และ OpenGL เพื่อทำการแสดงผลทางด้านกราฟฟิคสามมิติ

6. พัฒนาโครงงานด้วยภาษา C++

1.5 โครงสร้างของระบบโดยย่อทางด้านกายภาพ

1. ลูกโบว์ลิ่งมีขนาดเส้นรอบวง 26.704-27.002 นิ้ว และมีเส้นผ่าศูนย์กลาง 8.5-8.595 นิ้ว

2. พื้นสนามหรือเลน มีความกว้าง 1.07 เมตร ความยาว 14.17 เมตร

3. กล้องวีดีโอ 1 ตัวโดยจะต้องติดตั้งให้อยู่ในลักษณะเป็น Perspective และมีความสูงอยู่ในระดับที่เพียงพอเพื่อที่จะสามารถมองเห็นการเคลื่อนที่ของลูกบนเลนตั้งแต่เริ่มจนกระทั้งกระทบกับพิณหรือออกไปนอกฉากที่จะทำการศึกษา

4. อุปกรณ์ในการ Capture เพื่อที่จะเปลี่ยนค่าทางสัญญาณภาพให้สามารถนำมาแสดงผลและคำนวณในคอมพิวเตอร์ได้ ซึ่งในที่นี้อาจจะเป็น Card Capture , Firewire หรือ USB

1.6 ขอบเขตของโครงงาน

ในการพัฒนาโครงการและทดสอบอัลกอริทึมนั้นจะทำการพิจารณาและศึกษาทรงกลมที่ปรากฏในฉากเพียง 1 ลูก โดยไฟล์วีดีโอที่นำมาทำการศึกษานั้นสร้างด้วยโปรแกรม Maya โดยในการจำลองนั้นจะคำนึงถึงโครงสร้างทางกายภาพแท้จริงของกีฬาโบว์ลิ่งด้วย ซึ่งจะประกอบไปด้วยเลนส์เพียง 1 เลนส์และตำแหน่งของ Marker จำนวน 4 ตำแหน่ง เพื่อขั้นตอนในการ Calibration ซึ่งได้กำหนดให้เป็นสีขาว โดยจะยังไม่ได้คำนึงถึงองค์ประกอบอื่นๆที่จะมีผลกระทบต่อการพิจารณาทรงกลมโดยพัฒนาภายใต้ระบบปฏิบัติการ Microsoft Windows XP Service PackII

1.7 ประโยชน์ที่คาดว่าจะได้รับ

1. สามารถนำโปรแกรมไปประยุกต์ใช้ร่วมกับกีฬาโบว์ลิ่ง เพื่อทำให้การแข่งขั้นมีความน่าสนใจมากยิ่ง

2. นำไปประยุกต์ใช้กับงานทางด้านการนำเสนอทางด้านอื่นได้มากยิ่งขึ้น

3. นำไปพัฒนาซอฟต์แวร์และงานทางด้านกราฟิกให้เสมือนจริง และเป็นธรรมชาติ ด้วยระบบการควบคุมที่ง่ายและสะดวก

บทที่ 2
ทฤษฎีที่ใช้สร้างโปรแกรมจำลองการเคลื่อนที่ของทรงกลมบนพื้นราบ

โครงการนี้เป็นระบบการสร้างโปรแกรมจำลองการเคลื่อนที่ของทรงกลมบนพื้นราบจากกล้องวีดีโอ ในการพัฒนานั้นจะต้องมีความเข้าใจพื้นฐานทางด้าน Image Processing และ Computer Vision เพื่อทำการวิเคราะห์ข้อมูลที่ได้จากไฟล์ภาพ นอกจากนี้จะต้องมีความเข้าใจทางด้านคอมพิวเตอร์กราฟฟิคเพื่อนำข้อมูลที่ได้มาสร้างในคอมพิวเตอร์ให้มีความเหมือนจริงที่สุด ซึ่งจะกล่าวต่อไป

2.1 Image Analysis

เป็นกระบวนการพื้นฐานที่มีบทบาทสำคัญอย่างยิ่งในงานด้านการอธิบายเกี่ยวกับส่วนประกอบในรูปภาพเพื่อที่จะวิเคราะห์ค่าและแบ่งประเภทของข้อมูลที่ได้จากรูปภาพออกเป็นส่วนตามวัตถุประสงค์ที่สนใจศึกษา อย่างเช่น ถ้าศึกษาตัวอักษรที่ปรากฏอยู่บนภาพ ข้อมูลที่ได้นั้นจะประกอบไปด้วย ขนาด ทิศทาง และ เส้นรอบตัวอักษร เป็นต้น ในส่วนของโครงงานนี้ทำการแบ่งกระบวนการทำงานได้ดังนี้

2.1.1 Image Segmentation

เป็นขั้นตอนเริ่มต้น ของการทำ Image Processing จุดมุ่งหมายของการทำ Image Segmentation นั้นคือการแยกวัตถุที่สนใจศึกษาออกจากพื้นหลังโดยพยายามที่จะไม่ให้วัตถุที่ทำการแยกออกมานั้นสูญเสียรายละเอียดของตัววัตถุไปหรือพยายามที่จะทำให้สูญเสียน้อยที่สุด ซึ่งปกติประกอบด้วยขั้นตอนหลักอยู่สามขั้นตอนประกอบไปด้วยการขจัด Noise ต่างๆ ซึ่งติดมากับรูป เนื่องจากการที่ภาพนั้นมี Noise นั้นเป็นสาเหตุที่จะทำให้การแยกแยะวัตถุผิดพลาดโดยปกติ Noise นั้นจะถูกกำจัดด้วย วิธีพื้นฐานทางด้าน Image processing คือการทำ Opening ซึ่งเป็น Operation ในการปิดช่องว่างที่เกิดขึ้นและ Closing คือการลบขอบที่เกิดจากการมี Noise ภายในภาพออกไป วิธีการนี้เรียกว่าเรียกว่า Image Pre-Processing ในกระบวนการนี้จะรวมไปถึงกระบวนทำ Background Subtraction ซึ่งเป็นกระบวนที่เปรียบเทียบภาพสองภาพ โดยที่ภาพหนึ่งนั้นจะเป็นภาพที่ได้รับการบันทึกไว้ก่อนหน้า(Reference Image) เพื่อเป็นภาพที่จะนำมาเปรียบเทียบกับภาพปัจจุบันที่ได้รับ(Current Image)เป็นวิธีการที่ได้ใช้อย่างแพร่หลายซึ่งในโปรแกรมนี้จะนำภาพมาใช้

ทั้งหมด 5 ภาพมาหาค่าเฉลี่ยเพื่อกำหนดเป็น Reference Image และนำกระบวนการดังกล่าวมาใช้ในการค้นหาตำแหน่งของทรงกลมเมื่อเริ่มเข้าสู่ Scene ที่สนใจศึกษา

ขั้นต่อมาคือ Initial Object Discrimination ซึ่งเป็นการแยกวัตถุออกจากรูปภาพแล้วจัดแบ่งให้เป็นหมวดหมู่ในที่นี้คือทำการแยกวัตถุจากภาพที่สนใจศึกษาออกมา ในการดำเนินงานจะมุ่งความสำคัญไปที่ ขอบของวัตถุซึ่งจะใช้กระบวนการวิเคราะห์รูปที่เรียกว่า Image Morphological ในส่วนของการทำ Dilation และ Erosion ซึ่งสามารถเขียนในรูปของสมการได้ดังนี้

สมการ Dilation คือ $$I \oplus S = \bigcup_{s \text{ in } S} I_s \qquad (1)$$

สมการ Erosion คือ $$I \ominus S = \{i \mid i + s \text{ in } I, \forall s \text{ in } S\} \qquad (2)$$

ขั้นตอนสุดท้ายคือการลดขนาดวัตถุลงให้เป็น 1 Pixel เพื่อนำมาเปรียบเทียบกับรูปแบบของวัตถุที่มีการจัดเก็บไว้ในฐานข้อมูลซึ่งจะนำมาใช้ในกระบวนการต่อไป

รูปที่ 2.1 แสดงรูปหลังกระบวนการ Background Subtraction

2.1.2 Marker

ตำแหน่งของ Marker คือจุดที่ถูกกำหนดไว้ในระบบพิกัดสามมิติ ซึ่งในที่นี้ได้กำหนดค่าความสว่างในบริเวณที่เป็น Marker ให้มีค่าเท่ากับ 255 (สีขาว) โดยที่จุดดังกล่าวจะทราบตำแหน่งที่แท้จริงในพิกัดสามมิติ ขั้นตอนแรกนั้นจะต้องทำการหาจุดที่เป็น Marker ที่ปรากฏขึ้นในภาพ ซึ่งจะมีการกำหนดค่าที่เรียกว่า Threshold ในที่นี้ได้กำหนดค่าความสว่างของจุดไว้ที่มากกว่าหรือเท่ากับค่าใดค่าหนึ่งแต่ไม่จำเป็นว่าค่าที่ทำการกำหนดไปนั้นจะเท่ากับค่าความสว่างของจุดเริ่มต้น เพราะ

ภาพที่ได้อาจจะมีค่าความสว่างลดลงได้ เมื่อจุดใดบนภาพที่มีค่าเกินกว่าค่า Threshold จะถูกกำหนดให้เป็นตำแหน่งของ Marker ซึ่งจุดที่จะต้องทำการค้นหามีทั้งหมด 4 ตำแหน่ง

2.1.3 Feature Extraction

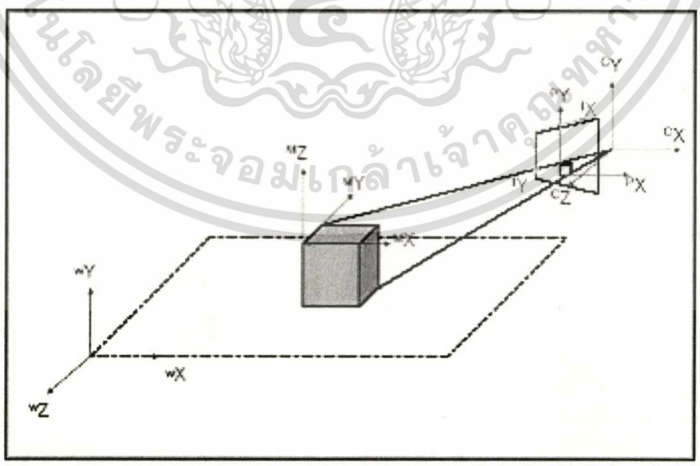
ขั้นตอนเป็นการตรวจสอบว่าวัตถุที่สนใจศึกษา ในที่นี้คือทรงกลมอยู่ที่ตำแหน่งใดบนภาพ โดยอาศัยการคำนวณพื้นฐาน อย่างเช่น คุณลักษณะที่อธิบายถึงโครงสร้างทางกายภาพของวัตถุ ซึ่งประกอบไปด้วย เส้นรอบรูป พื้นที่ของวัตถุ Major Axis , Minor Axis แต่ในที่นี้จะมุ่งความสนใจไปที่การหาจุดศูนย์กลางของทรงกลม(Center Of Mass)ที่ปรากฏอยู่ในรูปภาพ ตามสมการด้านล่างนี้

Center Of Mass x = Sum of object's x-pixel coordinate/Number of Pixel in Object (3)

Center Of Mass y = Sum of object's y-pixel coordinate/Number of Pixel in Object (4)

2.2 Coordinate Systems

ระบบพิกัดต่างๆนั้นจะประกอบไปด้วย อย่างน้อย 4 พิกัด ประกอบไปด้วย World Coordinate , Camera Coordinate , Image Plane Coordinate และ Image Pixel Coordinate ซึ่งจะมีความสัมพันธ์กันตามภาพ 2.2



รูปที่ 2.2 แสดงรูปแบบของตำแหน่งในระบบพิกัด

ในการเปลี่ยนระบบพิกัดใน 3 มิติระหว่างพิกัดทั้งสี่พิกัดข้างต้นนั้นมีการใช้เมตริกซ์ในการย้ายแกนต่างๆดังต่อไปนี้

$$\text{Translation Matrix} \begin{bmatrix} P_2^x \\ P_2^y \\ P_2^z \\ 1 \end{bmatrix} = \begin{bmatrix} 1,0,0,t_x \\ 0,1,0,t_y \\ 0,0,1,t_z \\ 0,0,0,1 \end{bmatrix} \begin{bmatrix} P_1^x \\ P_2^y \\ P_3^z \\ 1 \end{bmatrix} \quad (5)$$

$$\text{Scaling Matrix} \begin{bmatrix} P_2^x \\ P_2^y \\ P_2^z \\ 1 \end{bmatrix} = \begin{bmatrix} S_x,0,0,0 \\ 0,S_y,0,0 \\ 0,0,S_z,0 \\ 0,0,0,1 \end{bmatrix} \begin{bmatrix} P_1^x \\ P_2^y \\ P_3^z \\ 1 \end{bmatrix} \quad (6)$$

$$\text{Rotation Matrix} \begin{bmatrix} P_2^x \\ P_2^y \\ P_2^z \\ 1 \end{bmatrix} = \begin{bmatrix} r_{11},r_{12},r_{13},0 \\ r_{21},r_{22},r_{23},0 \\ r_{31},r_{32},r_{33},0 \\ 0,0,0,1 \end{bmatrix} \begin{bmatrix} P_1^x \\ P_2^y \\ P_3^z \\ 1 \end{bmatrix} \quad (7)$$

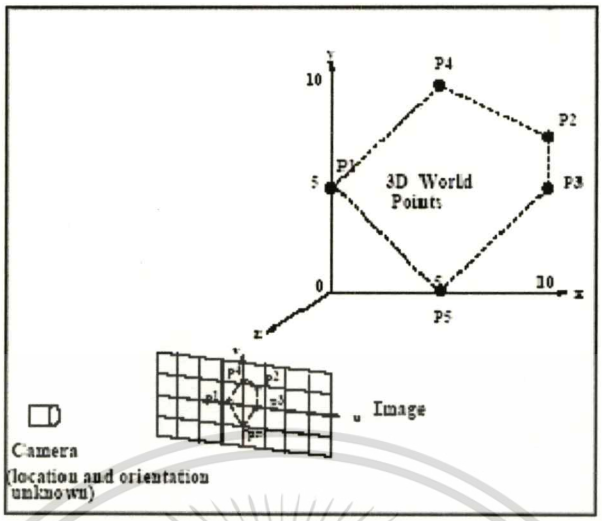
เมื่อ P_1 คือจุดก่อนที่จะทำการย้ายตำแหน่ง

P_2 คือจุดที่ทำการย้ายตำแหน่งไป

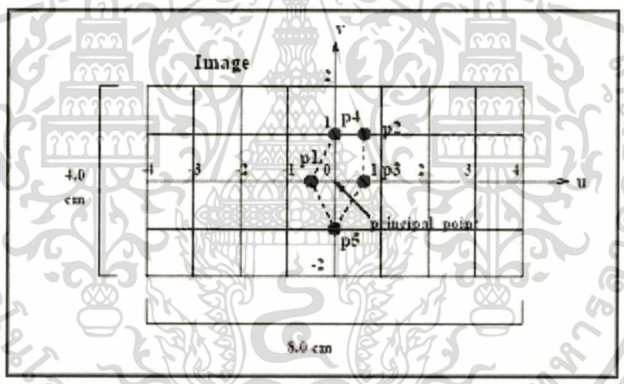
จากเมตริกซ์พื้นฐานทั้งสามนั้นสามารถนำมาคำนวณหา Homography Matrix ที่จะได้กล่าวในหัวข้อถัดไป

2.3 Homography Matrix

Homography Matrix เป็นเมตริกซ์ที่จะถูกนำมาใช้ในกระบวนการ Mapping เพื่อเปลี่ยนแปลงจุดใน World Coordinate ไปเป็น Image Coordinate ดังภาพ



รูปที่ 2.3 แสดงรูปแบบของตำแหน่งระหว่าง Image plane และ World plane



รูปที่ 2.4 แสดง Image plane

สามารถเขียนอยู่ในรูปแบบของสมการได้คือ

$$^{I}P = {}^{I}_{W}C \cdot {}^{W}P \qquad (8)$$

เมื่อ $^{I}_{W}C$ คือ Homography Matrix ของกล้อง

^{W}P คือ ตำแหน่งใน World Coordinate

^{I}P คือ ตำแหน่งใน Image Coordinate

สามารถเขียนอยู่ในรูปแบบของ Matrix คือ

$$\begin{bmatrix} s^I P_r \\ s^I P_c \\ s \end{bmatrix} = {}^I{}_W\, C \begin{bmatrix} {}^W P_x \\ {}^W P_y \\ {}^W P_z \\ 1 \end{bmatrix} \text{ จะได้ } \begin{bmatrix} s^I P_r \\ s^I P_c \\ s \end{bmatrix} = \begin{bmatrix} c_{11}, c_{12}, c_{13}, c_{14} \\ c_{21}, c_{22}, c_{23}, c_{24} \\ c_{31}, c_{32}, c_{33}, 1 \end{bmatrix} \begin{bmatrix} {}^W P_x \\ {}^W P_y \\ {}^W P_z \\ 1 \end{bmatrix} \qquad (9)$$

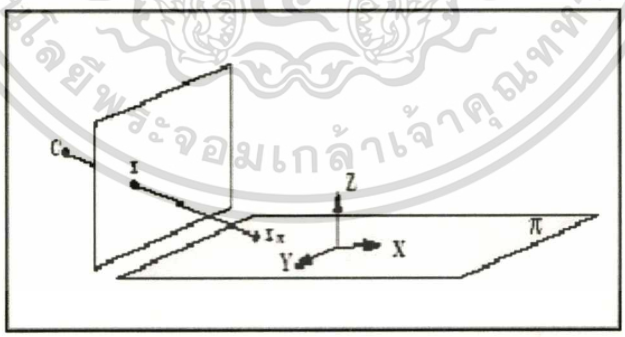
เนื่องจากการเคลื่อนที่จะอยู่บนระนาบดังนั้นสามารถที่จะกำหนดให้แกน z เป็น 0 ได้ซึ่งจะได้เป็น Matrix ขนาด 3×3 แทนดังนี้

$$\begin{bmatrix} s^I P_r \\ s^I P_c \\ s \end{bmatrix} = \begin{bmatrix} c_{11}, c_{12}, c_{13}, c_{14} \\ c_{21}, c_{22}, c_{23}, c_{24} \\ c_{31}, c_{32}, c_{33}, 1 \end{bmatrix} \begin{bmatrix} {}^W P_x \\ {}^W P_y \\ 0 \\ 1 \end{bmatrix} \text{ จะได้ } \begin{bmatrix} s^I P_r \\ s^I P_c \\ s \end{bmatrix} = \begin{bmatrix} c_{11}, c_{12}, c_{14} \\ c_{21}, c_{22}, c_{24} \\ c_{31}, c_{32}, 1 \end{bmatrix} \begin{bmatrix} {}^W P_x \\ {}^W P_y \\ 1 \end{bmatrix} \qquad (10)$$

ดังนั้น

$${}^I P_r = \frac{\left[c_{11} c_{12} c_{14}\right] \cdot \left[{}^W P_x \; {}^W P_y 1\right]}{\left[c_{31} c_{32} 1\right] \cdot \left[{}^W P_x \; {}^W P_y 1\right]} \qquad (11)$$

$${}^I P_c = \frac{\left[c_{21} c_{22} c_{24}\right] \cdot \left[{}^W P_x \; {}^W P_y 1\right]}{\left[c_{31} c_{32} 1\right] \cdot \left[{}^W P_x \; {}^W P_y 1\right]} \qquad (12)$$



รูปที่ 2.5 แสดง World Coordinate บนแนวแกน z = 0

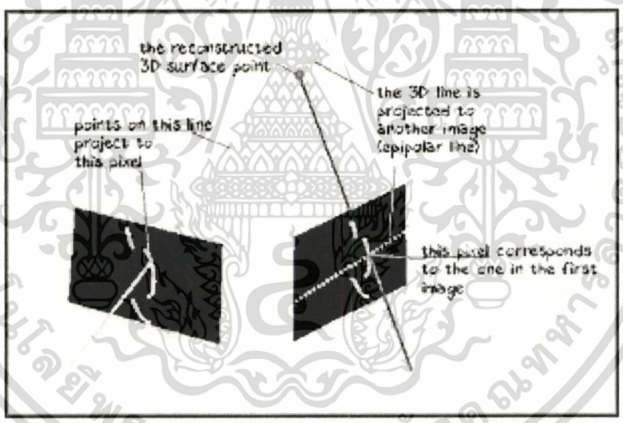
แต่การจะได้มาซึ่ง Homography Matrix นั้นจะคำนวณได้โดยง่ายตามอัลกอริทึม ซึ่งจะอาศัยจุดทั้งหมด 4 จุดในการสร้างเมตริกซ์ โดยที่จุดทั้ง 4 นั้นจะต้องระบุได้ว่าอยู่ที่ตำแหน่งใดใน World Coordinate เพื่อนำมาสร้างความสัมพันธ์กับตำแหน่งที่ปรากฏอยู่บน Image Coordinate

2.4 Stereo Vision

ในการทำโครงงานนั้นจะมีการจำลองกล้อง Virtual Camera ขึ้นเพื่อนำมาใช้ในการสร้างและคำนวณหาตำแหน่งของทรงกลมที่มีความสัมพันธ์กันอย่างถูกต้อง โดยจะใช้บทบาทของ Stereo Vision ซึ่งเป็นทฤษฎีที่ใช้ในการกู้คืนตำแหน่งของวัตถุในสามมิติ

2.4.1 Point Correspondence

คุณสมบัติพื้นฐานของจุดที่ถูกเรียกว่า Point Correspondence นั้นจะประกอบไปด้วยสองคุณสมบัติข้อแรกข้อ จุดที่อยู่ใน Image Coordinate ของกล้องตัวที่ 1จะสามารถกำหนดที่เกี่ยวเนื่องกันใน Image Coordinate ของกล้องอีกตัวได้ ข้อที่สองจุดนั้นจะเป็นจุดเดียวกันใน World Coordinate ในที่นี้คือจุดที่เป็น Marker และจุดศูนย์กลางของทรงกลมและมีความเกี่ยวเนื่องกันซึ่งจะนำไปสู่ Epiporlar Geometry



รูปที่ 2.6 แสดงตำแหน่งของจุดที่เป็น Point Correspondence

2.4.2 Epipolar Geometry

จากรูป 2.6 นั้นจะพบว่าจุดที่เป็น Point Correspondence ในทั้ง Image Plane ทางด้านซ้ายและด้านขวาทำให้เกิดระนาบที่เรียกว่า Epipolar Plane ซึ่งตัดผ่านทั้งจุดที่เป็นตำแหน่งใน World Coordinate และจุดที่ปรากฏอยู่บน Image Plane ดังนั้นจึงอาศัยความสัมพันธ์ดังกล่าวมาแก้ระบบสมการเพื่อที่จะค้นหาตำแหน่งจุดศูนย์กลางของทรงกลมใน World Coordinate จากทฤษฎีเบื้องต้นนั้นจะสามารถคำนวณหา Homography Matrix ดังนั้นจึงสามารถทำการแก้สมการได้ดังต่อไปนี้

$$ {}^{I}P_1 = \lambda_1 \cdot {}^{I}{}_{W}C_1 \cdot {}^{W}P \qquad (13) $$

$$ {}^{I}P_2 = \lambda_2 \cdot {}^{I}{}_{W}C_2 \cdot {}^{W}P \qquad (14) $$

$$ P = \lambda^{-1}{}_1 \cdot {}^{I}{}_{W}C_1^{-1} \cdot {}^{I}P_1 \qquad (15) $$

ดังนั้นจะได้ $$ {}^{I}P_2 = \lambda^{-1}{}_3 \cdot {}^{I}{}_{W}C_2 \cdot {}^{I}{}_{W}C_1^{-1} \cdot {}^{I}P_1 \qquad (16) $$

เมื่อ ${}^{I}{}_{W}C_1$ และ ${}^{I}{}_{W}C_2$ คือ Homography Matrix ของกล้องที่ 1 และ 2 ตามลำดับ

${}^{W}P$ คือ ตำแหน่งใน World Coordinate

${}^{I}P_1$ และ ${}^{I}P_2$ คือ ตำแหน่งใน Image Coordinate ของกล้องที่ 1 และ 2 ตามลำดับ

λ_i คือ ค่าคงที่จำนวนหนึ่งเมื่อ $i = 1,2,3$

2.5 Image Programming

TLIB เป็นไลบารีที่ใช้เพื่อทำการเก็บค่าข้อมูลที่ต้องการจากไฟล์วีดีโอ สามารถใช้งานได้ง่ายและมีประสิทธิภาพค่อนข้างสูง ซึ่งได้รับการพัฒนาโดย Grange sebastien

2.5.1 คุณสมบัติของ TLIB

1. ถูกสร้างขึ้นเพื่อใช้ในการโปรแกรมงานด้าน Real-Time Object Tracking โดยตัวไลบารีเองนั้นจะถูกเรียกใช้ผ่านภาษา C/C++ เพื่อเพิ่มประสิทธิภาพการโปรแกรมด้าน Image Processing

2. คุณลักษณะเด่นของ TLIB นั้นจะรองรับ Image Format ได้หลากหลายรูปแบบซึ่งรูปแบบที่ ไลบารี รองรับนั้นจะถูกระบุอยู่ที่ tl_format ซึ่งจะเก็บทั้งรูปแบบภาพที่รองรับและ ค่าของสีที่สามารถใช้งานและอ่านค่าได้ โดยแต่ละจุดในรูปนั้นจะถูกระบุ tlPixelArray ตัวอย่างเช่น การจัดเก็บค่าของ RGB นั้นจะถูกจัดเก็บเป็นลำดับแบบ Array ตามลำดับแถว ดังนี้ RGBRGBRGBRGBRGB...

3. ไลบารีนั้นถูกออกแบบให้สามารถติดต่อกับอุปกรณ์แสดงผลรวมถึงอุปกรณ์จัดเก็บค่าได้หลากหลาย ดังนั้นจึงสามารถใช้งานได้ตั้งแต่คอมพิวเตอร์ส่วนบุคคลไปถึงระดับซูปเปอร์คอมพิวเตอร์

4. สามารถใช้งานได้ง่ายมีคำสั่งในการทำงานที่ได้รับการอธิบายการใช้งานได้เป็นอย่างดี รวมถึงมีโครงสร้างที่ไม่สลับซับซ้อน

2.5.2 สถาปัตยกรรม

TLIB นั้นถูกสร้างเพื่อให้เขียนโปรแกรมได้ดีทั้งในภาษา C และ C++ เพื่อประโยชน์ในการประมวลผลได้ทั้งการเขียนโปรแกรมที่มีโครงสร้างแบบวัตถุ เพื่อสะดวกต่อผู้พัฒนา

2.6 Graphic

ในการสร้างงานของ Graphic นั้นได้เลือกใช้ OpenGL ในการนำเสนอ ซึ่งในการใช้งานนั้นจะทำได้ง่าย เพราะในตัวไลบารีของ OpenGL นั้นจะมีความสามารถที่คอยควบคุมการแสดงผลภาพได้อย่างมีประสิทธิภาพ อีกทั้งตัวไลบารีมีเสถียรภาพมากเพราะได้รับการพัฒนากันมาอย่างยาวนาน รวมถึงได้รับการทดสอบอย่างแพร่หลาย อีกทั้ง OpenGL มีคำสั่งต่างๆมากมายซึ่งได้รับการจัดโครงสร้างมาเป็นอย่างดี

2.6.1 สถาปัตยกรรม

OpenGL ถูกออกแบบในลักษณะของสถานะ คือเมื่อกำหนดรูปทรง อย่างเช่น ทรงกลม ที่จะวาดแล้วผลลัพธ์นั้นจะขึ้นอยู่กับการกำหนดสถานะก่อนหน้านั้นด้วย เช่น สี โครงสร้าง ตำแหน่ง เพื่อแสดงผลได้อย่างถูกต้องและสวยงาม

2.6.2 การเรียกใช้ฟังก์ชันของ OpenGL

OpenGL มีแบบแผนในการตั้งชื่อเพื่อให้ผู้พัฒนาทราบที่มาของฟังก์ชัน รวมถึงจำนวนอาร์กิวเมนต์ที่ใช้ติดต่อ ซึ่งรูปแบบโครงสร้างมีดังนี้

<Library Type><Root command><Number of arguments><Data Type>

Library Type	บอกว่าใช้ไลบารี่ไหน
Root Command	เป็นรากของคำสั่ง
Number of arguments	จำนวนของพารามิเตอร์
Data Type	ชนิดของตัวแปร

2.6.3 ชนิดของข้อมูลใน OpenGL

OpenGL นั้นเพื่อการที่สามารถนำเอาไปโปรแกรมในระบบปฏิบัติการที่หลากหลายจึงได้ทำการกำหนดชนิดของข้อมูลขึ้นเอง แต่จะมีความคล้ายคลึงกับภาษร C ซึ่งสามารถเทียบเคียงกันได้ประกอบไปด้วย

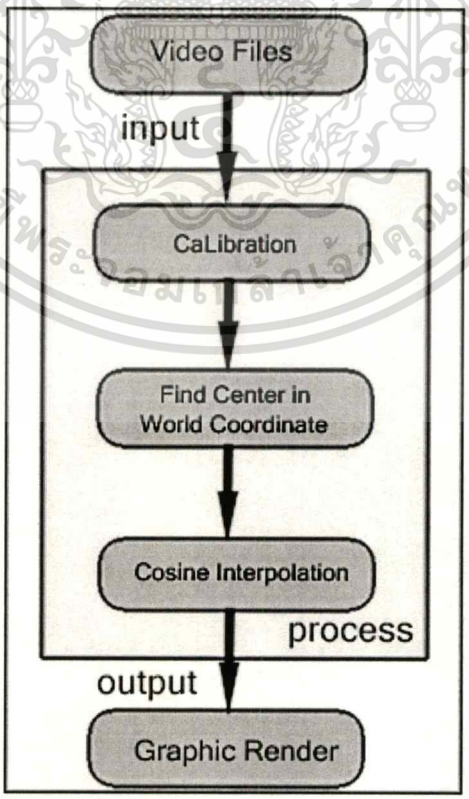
Suffix	Data Type	Typical C type	OpenGL Type Name
B	8-bit integer	Signed char	GLbuyte
S	16-bit integer	Short	GLshort
I	32-bit integer	Int or long	GLint ,GLsizel
F	32-bit floating point	Float	GLfloat , GLclampf
D	64-bit floating point	Double	GLdouble , GLclampd
Ub	8-bit unsigned number	Insigned char	GLubyte , GLboolean
Us	16-bit unsigned number	Unsigned short	GLushort
Ui	32-bit unsigned number	Unsigned int or unsigned long	GLuint ,Glenum , GLbitfield

ตารางที่ 2.1 ตารางแสดงชนิดข้อมูลของ OpenGL

บทที่ 3
การออกแบบและอัลกอริทึมโปรแกรมจำลองการเคลื่อนที่ของทรงกลมบนพื้นราบ

3.1 โปรแกรมจำลองการเคลื่อนที่ของทรงกลมบนพื้นราบ

ในการพัฒนาโครงงานนั้น ข้อมูลที่ปรากฏในรูปภาพนั้นมีความสำคัญเพื่อที่จะนำมาทำการกู้คืนข้อมูลสามมิติจากกล้องวีดีโอซึ่งมีเพียงกล้องเดียว โดยเบื้องต้นได้ทำการจำลองการเคลื่อนที่ของทรงกลมบนพื้นด้วยโปรแกรม Maya เพื่อสามารถตรวจความถูกต้องของอัลกอริทึม โดยโปรแกรมทั้งหมดของโครงการนั้นจะพัฒนาด้วย Microsoft Visual C++ ซึ่งมีขั้นตอนหลักจะประกอบไปด้วยสองส่วนหลัก คือ ส่วนแรกคือ การวิเคราะห์เฟรมภาพ ด้วยไลบารี TLIB ส่วนที่สองคือการแสดงผลด้าน Graphic ด้วย OpenGL การทำงานของโปรแกรมจะเริ่มจากการนำไฟล์วีดีโอซึ่งเป็น Input ของโปรแกรมเข้าสู่การคำนวณค่าตัวแปรในขั้นตอนต่างๆก่อนจะถูกนำเสนอเป็น Output ในรูปแบบของกราฟฟิคที่สามารถเสนอมุมมองได้หลากหลายมากขึ้น



รูปที่ 3.1 แสดงภาพรวมการทำงานของโปรแกรม Input , Process และ Output

3.1.1 การวิเคราะห์เฟรมภาพ

จะแยกเป็นสองขั้นตอนย่อย คือขั้นตอนแรกคือการ Calibration ซึ่งจะเป็นขั้นตอนที่จะค้นหาค่าตำแหน่งของ Maker ในเฟรมของวีดีโอแล้วนำค่ามาคำนวณสร้างเป็น Homography Matrix ขั้นตอนที่สองคือ ขั้นตอนการหาตำแหน่งในสามมิติของวัตถุทรงกลม ซึ่งจะประกอบด้วยขั้นตอนย่อยๆ หลายขั้นตอนโดยจะเริ่มจากการทำ Background Subtraction แล้วนำเฟรมที่ผ่านกระบวนการดังกล่าวมาทำการประมาณตำแหน่งของทรงกลมก่อนที่จะหาค่าจุดศูนย์กลางที่ปรากฏในภาพ เพื่อที่จะเข้าสู่การดำเนินการขั้นถัดไปคือทำการประมาณค่าตำแหน่งของจุดศูนย์กลางทรงกลมใน World Coordinate อีกครั้ง เมื่อได้ข้อมูลครบถ้วนจะนำข้อมูลดังกล่าวมาแสดงผลต่อไป

3.1.2 การแสดงผลด้าน Graphic

นำตำแหน่งจุดศูนย์กลางของทรงกลมที่ได้จากการคำนวณทั้งหมดมาทำการจำลองเส้นทางการเคลื่อนที่ในทุกๆเฟรมด้วยฟังก์ชัน Cosine Interpolation แล้วแสดงการวิถีการเคลื่อนที่ของทรงกลมด้วย OpenGL

3.2 การออกแบบและอัลกอริทึม

ในส่วนนี้นั้นจะพูดถึงการออกแบบโปรแกรมรวมถึงอัลกอริทึมในขั้นตอนการทำงานต่างๆ โดยจะประกอบไปด้วยขั้นตอนการ Calibration ขั้นตอนการหาวัตถุทรงกลม และขั้นตอนการแสดงผลด้วย OpenGL

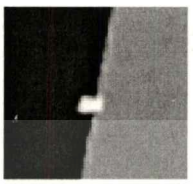
3.2.1 ขั้นตอนการ Calibration

ในการทำงานนั้นโปรแกรมจะทำการ Capture ภาพออกเป็นเฟรมๆ เพื่อทำการค้นหาข้อมูลในแต่เฟรม ซึ่งในที่นี้จะไม่ได้ทำการค้นหาข้อมูลในทุกเฟรมและในส่วนของขั้นตอนการ Calibration นั้นจะถูกประมวลผลแค่เพียงเฟรมแรกที่ทำการศึกษาเพียงเฟรมเดียวเท่านั้น โดยเริ่มจาก

3.2.1.1 เก็บตำแหน่ง Marker

ตำแหน่งของ Marker ที่จะทำการเก็บค่านั้นจะถูกกำหนดในมองเห็นอยู่ในแนวระนาบของเลนส์โบว์ลิ่งด้วยโดยที่ตำแหน่งของ Marker นั้นจะถูกกำหนดไว้ที่มุมแต่ละมุมของเลนส์โบว์ลิ่ง เนื่องจากจะเป็นจุดที่ถูกกำหนดให้รู้ค่าในตำแหน่งของ World Coordinate เพื่อนำมาสร้าง Homography Matrix ในขั้นต่อมา ในเบื้องต้นได้กำหนดจุดที่ต้องการให้เป็น Marker ไว้ด้วยจุดที่เป็นสีขาวซึ่งมีค่าความสว่างอยู่ที่ 255 ดังนั้นในการพัฒนาโปรแกรมจึงได้กำหนดให้ค่า Threshold

อยู่ที่ 200 หากว่าค่าความสว่าง ณ จุดใดผ่านค่า Threshold จะถูกกำหนดให้เป็นตำแหน่งที่ Marker ปรากฏอยู่ ซึ่งในการพัฒนานั้นได้กำหนดให้ทำการหา Marker ทั้งหมดสี่ตำแหน่งที่มุมของสนามในแต่ละมุม



รูปที่ 3.2 แสดงตัวอย่างของตำแหน่งที่ใช้เป็น Marker

อัลกอริทึมในการค้นหาตำแหน่ง Marker โดยในที่นี้จะแสดงเพียง 1 จุด ที่ทำการค้นหาค่า

```
void CTlibboDlg::findwhitepoint()
{
    tlColor *P;
    P = new tlColor();
    c =0;

    //top left
    for(x = 116; x <160 ; x++)
        {
            for(y = 10; y <120 ; y++)
            {
                bgImage->getColor(x,y,P);
                if ((P->intensity)>200)
                {
                        Col[c]=x;
                        Row[c]=y;
                        c++;
                }
                if (c>0) break;
            }

        }
```

3.2.1.2 การคำนวณหา Homography Matrix

ขั้นตอนการคำนวณหา Homography Matrix เป็นขั้นตอนต่อเนื่องหลังจากที่ได้ค่าตำแหน่งของ Marker ที่ปรากฏอยู่ใน Pixel Coordination แล้วทั้งหมด 4 จุด โดยที่จุดทั้ง 4 นั้นจะเป็นจุดที่ทราบตำแหน่งใน World Coordinate ซึ่งโปรแกรมจะนำค่าจุดทั้งหมดนั้นมาสร้างเป็น Matrix ดังนี้

$$\begin{pmatrix} x_1 & y_1 & 1 & 0 & 0 & 0 & -x'_1x_1 & -x'_1y_1 \\ 0 & 0 & 0 & x_1 & y_1 & 1 & -y'_1x_1 & -y'_1y_1 \\ x_2 & y_2 & 1 & 0 & 0 & 0 & -x'_2x_2 & -x'_2y_2 \\ 0 & 0 & 0 & x_2 & y_2 & 1 & -y'_2x_2 & -y'_2y_2 \\ x_3 & y_3 & 1 & 0 & 0 & 0 & -x'_3x_3 & -x'_3y_3 \\ 0 & 0 & 0 & x_3 & y_3 & 1 & -y'_3x_3 & -y'_3y_3 \\ x_4 & y_4 & 1 & 0 & 0 & 0 & -x'_4x_4 & -x'_4y_4 \\ 0 & 0 & 0 & x_4 & y_4 & 1 & -y'_4x_4 & -y'_4y_4 \end{pmatrix} \begin{pmatrix} h_{11} \\ h_{12} \\ h_{13} \\ h_{21} \\ h_{22} \\ h_{23} \\ h_{31} \\ h_{32} \end{pmatrix} = \begin{pmatrix} x'_1 \\ y'_1 \\ x'_2 \\ y'_2 \\ x'_3 \\ y'_3 \\ x'_4 \\ y'_4 \end{pmatrix} \quad (17)$$

เมื่อ (x_i, y_i) คือ ตำแหน่งของ Maker ใน World Coordinate

(x'_i, y'_i) คือ ตำแหน่งของ Maker ใน Pixel Coordinate

$h_{i,j}$ คือค่า Homography Matrix

โดยเงื่อนไขในการแก้สมการนั้นคือที่ตำแหน่ง $h_{3,3}$ จะมีการกำหนดค่าเท่ากับ 1 เมื่อทำการสร้างเงื่อนไขให้อยู่ในรูปแบบ Matrix ได้แล้วจะทำการแก้สมการหาค่า Homography Matrix หรืออาจจะถูกเรียกว่าเป็นการหา Plane Projective Tranformation ด้วยวิธีการแก้สมการ Matrix ที่เรียกว่า Gaussian Elimination with Backsubstitution และทำการคำนวณหาค่า Inverse Matrix ของ Homography Matrix เพื่อนำมาคำนวณในขั้นตอนต่อไปเป็นการสิ้นสุดขั้นตอน Calibration

อัลกอริทึมในการคำนวณหา Homography Matrix ด้วยวิธี Gaussian Elimination with Backsubstitution

```
int CTlibboDlg::gelimd(float **a,float *b,float *X,int n)
{
    float tmp,pvt,*t;
    int k;

    for (i=0;i<n;i++) {             // outer loop on rows
        pvt = a[i][i];              // get pivot value
        if (fabs(pvt) < EPS) {
            for (j=i+1;j<n;j++) {
                if(fabs(pvt = a[j][i]) >= EPS) break;
            }
            if (fabs(pvt) < EPS) return 1;     // nowhere to run!
            t=a[j];                 // swap matrix rows...
            a[j]=a[i];
            a[i]=t;
            tmp=b[j];               // ...and result vector
            b[j]=b[i];
            b[i]=tmp;
        }
// (virtual) Gaussian elimination of column
        for (k=i+1;k<n;k++) {       // alt: for (k=n-1;k>i;k--)
            tmp = a[k][i]/pvt;
            for (j=i+1;j<n;j++) {   // alt: for (j=n-1;j>i;j--)
                a[k][j] -= tmp*a[i][j];
            }
            b[k] -= tmp*b[i];
        }
    }
// Do back substitution

    for (i=n-1;i>=0;i--) {
        X[i]=b[i];
        for (j=n-1;j>i;j--) {
            X[i] -= a[i][j]*X[j];
        }
        X[i] /= a[i][i];
    }
    return 0;
}
```

อัลกอริทึมในการคำนวณหา Inverse Homography Matrix

```
int CTlibboDlg::matinv(float **invM,int n)
{
float **inv,tmp;
int i,j,k,retval;
    retval = 0;
    inv = new float *[n];
    for (i=0;i<n;i++) {
        inv[i] = new float [n];
    }
    for (i=0;i<n;i++) { // Initialize identity matrix
        for (j=0;j<n;j++) {
            inv[i][j] = 0.0;
        }
        inv[i][i] = 1.000;
    }
    for (k=0;k<n;k++) {
        tmp = invM[k][k];
        if (tmp == 0) {
            retval = 1;
        }
        for (j=0;j<n;j++) {
            if (j>k) invM[k][j] /= tmp;     // Don't bother with previous entries
            inv[k][j] /= tmp;
        }
        for (i=0;i<n;i++) {                 // Loop over rows
            if (i == k) continue;
            tmp = invM[i][k];
            for (j=0;j<n;j++) {
                if (j>k) invM[i][j] -= invM[k][j]*tmp;
                inv[i][j] -= inv[k][j]*tmp;
            }
        }
    }
    for (i=0;i<n;i++) { // Copy inverse to source matrix
        for (j=0;j<n;j++) {
            invM[i][j] = inv[i][j];
        }
    }
    return retval;
}
```

3.2.2 ขั้นตอนการหาวัตถุทรงกลม

ขั้นตอนในการค้นหานั้นจะเริ่มจากกระบวนการ Background Subtraction โดยที่โปรแกรมนั้นจะทำการ Capture เฟรมภาพจากไฟล์วีดีโอ โดยการ Capture นั้นจะทำการเลือกเฟรมที่ 1- 10 เป็นภาพที่จะนำมาสร้างเป็น Reference Image เพื่อที่จะลบออกจากภาพปัจจุบัน Current Image ซึ่งการเลือก Capture เฟรมภาพนั้นจะทำทุกๆ 10 เฟรมจะไม่ทำการ Capture ทุกเฟรม ในเฟรมที่ได้ทำการ Capture มานั้นจะถูกนำมาเพื่อคำนวณหาว่ามีวัตถุทรงกลมได้เข้ามาใน Scene ที่จะทำการศึกษาหรือไม่ โดยทำการตั้งค่า Threshold ไว้เพื่อเป็นตัวชี้วัดว่ามีทรงกลมเข้ามาใน Scene และเพื่อเป็นการขจัด Noise ที่อาจจะเกิดขึ้นได้ และเมื่อผ่านกระบวนการนั้นแล้วจะเข้าสู่กระบวนการหาค่าจุดศูนย์กลางของทรงกลม ซึ่งจะนำค่าความสว่างของแต่ละ Pixel มาคำนวณ โดยสมมุติฐานจะถูก

ตั้งอยู่ที่ว่า จุดศูนย์กลางนั้นจะมีค่าความสว่างเฉลี่ยมากที่สุด เพราะภาพที่นำมาใช้นั้นจะถูกปรับให้อยู่ใน Gray Scale เพื่อง่ายต่อการคำนวณค่าความสว่างในแต่ละ Pixel ซึ่งอัลกอริทึมในการคำนวณหาจุดศูนย์กลางนั้นจะถูกแบ่งเป็นขั้นตอนย่อยๆ ส่วนแรกจะเป็นการระบุตำแหน่งที่ทรงกลมอยู่ภายในภาพ ส่วนที่สองจะเป็นการระบุตำแหน่งจุดศูนย์กลางของทรงกลม และกระบวนการสุดท้ายจะเป็นการคำนวณเพื่อการคืนค่าตำแหน่งของทรงกลมในสามมิติ

3.2.2.1 การคำนวณหาค่า Profile ตามแนว Column และ Row

การคำนวณหา Profile นั้นจะทำเพื่อการระบุบริเวณที่ทรงกลมอยู่ภายในภาพ หลังจากที่ได้เฟรมภาพที่ต้องการทำการค้นหาข้อมูลผ่านกระบวนการ Background Subtraction แล้วเฟรมภาพดังกล่าวจะถูกนำเข้าสู่กระบวนการหาค่า Profile ซึ่งจะเป็นการคำนวณหาค่าความสว่างเฉลี่ยในภาพทั้งหมดตามแนว Column และ Row ซึ่งในที่นี้จำนวน Column ที่ใช้ทำการคำนวณคือ 320 และจำนวน Row คือ 240 โดยจะคำนวณในแนว Column ของภาพก่อนซึ่งจะทำการรวมค่าความสว่างในทุกจุดที่ปรากฏในแต่ละ Column แล้วทำการหารด้วยจำนวน Row เพื่อที่จะระบุตำแหน่งความสว่างว่าปรากฏอยู่บริเวณ Column ใดภายในภาพทำการวนซ้ำจนครบทุก Column ก่อนจะทำการคำนวณแบบเดียวกันที่ Row ทุก Row เช่นกันเหมือนผ่านกระบวนการหาบริเวณที่ทรงกลมปรากฏแล้วนั้นจึงทำการเรียกฟังก์ชั่นการคำนวณหาจุดศูนย์กลางวงกลมต่อไป

อัลกอริทึมในการหาค่า Profile ตามแนว Column และ Row

```
void CTlibboDlg::profilexy()
{
    int tempx=0;
    int tempy=0;
    int i,j;
    thres=50;
    tlColor *P;
    P = new tlColor();
    profilex = new float[320];
    profiley = new float[240];
    for(i=0;i<320;i++)
        profilex[i]=0;
    for(i=0;i<240;i++)
        profiley[i]=0;
    //profilex
    for (i=0;i<320;i++)
    {
        for(j=0;j<240;j++)
        {
            tempImage->getColor(i,j,P);
            tempx=(P->intensity)+tempx;
        }
    profilex[i]=tempx/240;
    tempx=0;
    }
    //profiley
    for (j=0;j<240;j++)
    {
        for(i=0;i<320;i++)
        {
            tempImage->getColor(i,j,P);
            tempy=(P->intensity)+tempy;
        }
    profiley[j]=tempy/320;
    tempy=0;
    }
    cx=findcenter(profilex,320,thres);
    cy=findcenter(profiley,240,thres);
}
```

3.2.2.2 การคำนวณหาค่าจุดศูนย์กลางทรงกลม

เป็นฟังก์ชันที่จะถูกเรียกหลังจากผ่านกระบวนการหา Profile ตามแนว Column และ Row เพื่อระบุบริเวณของทรงกลมภายในเฟรมภาพที่ศึกษาแล้ว ซึ่งฟังก์ชั่นนี้จะทำการคืนค่าตำแหน่งของจุดศูนย์กลางของทรงกลมซึ่งจะเป็นค่าเฉลี่ยว่าปรากฏอยู่ใน Pixel ที่เท่าใดภายในภาพ ขั้นตอนของกระบวนการจะประกอบไปด้วย การหา Pixel ที่มีความสว่างสูงสุดในแนวแกนที่ได้ส่งค่าเข้าสู่ฟังก์ชั่น และขั้นตอนต่อมาจะทำการตรวจสอบขนาดของทรงกลม ก่อนที่จะเข้าสู่ขั้นตอนการประมาณตำแหน่งศูนย์กลางของทรงกลมว่าจะปรากฏอยู่ที่ Pixel ใดในภาพด้วยวิธีการ Feature Extraction โดยจะทำการรวมค่าความสว่างตามแนวแกนที่ระบุก่อนทำการหารด้วยผลรวมของค่า

ความสว่างในแต่ละ Pixel คูณกับตำแหน่งในแนวแกนดังกล่าวหลังสิ้นสุดกระบวนการจะได้มาซึ่งตำแหน่งของศูนย์กลางวงกลมที่อยู่ในภาพ

อัลกอริทึมในการคำนวณหาค่าจุดศูนย์กลางทรงกลม

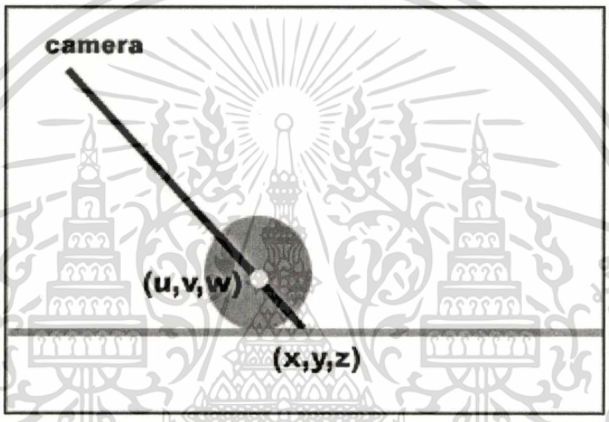
```
float CTlibboDlg::findcenter(float *profilex,int ii,float thres)
{
    float maxvalue=0;
    int ix=0;
    int n,lx,rx;
    float s,mox,sumtemp;
    // find max value
    for(j=0;j<ii;j++)
    {
        if(profilex[j]>maxvalue){
            maxvalue=profilex[j];
            ix=j;
        }
    }
    //check left right
    thres=(thres*maxvalue)/100;
    //go to left
    n=ix;
        while((profilex[n]>thres)&&(n>1))
        {
            n=n-1;
        }
    lx=n;
    //go to right
     n=ix;
        while((profilex[n]>thres)&&(n<ii))
        {
            n=n+1;
        }
     rx = n;
     mox =0;
     sumtemp=0;
    for(i=lx;i<rx;i++)
    {
        mox=mox+(i*profilex[i]);
        sumtemp=sumtemp+profilex[i];
    }
    s =mox/sumtemp;
return s;
}
```

3.2.2.3 การคำนวณเพื่อการคืนค่าตำแหน่งของทรงกลมในสามมิติ

เนื่องจากการคำนวณจาก Homography นั้นจะเป็นเพียงค่าประมาณ อีกทั้งข้อมูลในระบบสามมิติจากกล้องถ่ายภาพเพียงตัวเดียวนั้นจะสูญเสียค่าของแนวแกนในทางด้านระดับความลึกไป

ซึ่งในที่นี้ได้กำหนดให้เป็นแนวแกน y โดยจะต้องทำการคำนวณเพื่อทำการคืนค่าตำแหน่งที่ให้ได้ใกล้เคียงกับตำแหน่งที่แท้จริงให้ได้มากที่สุด

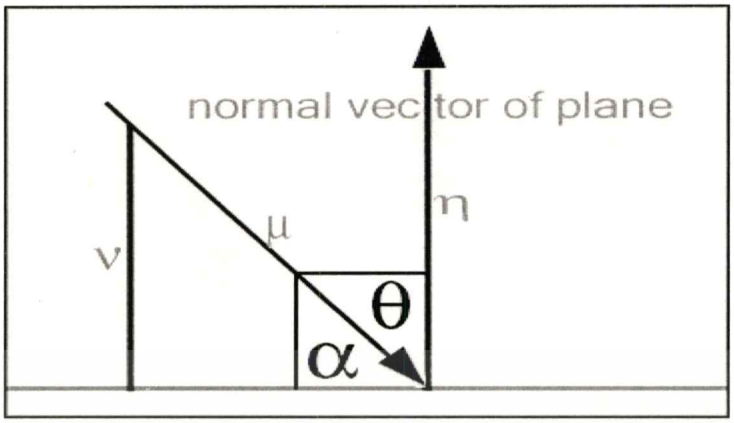
อัลกอริทึมที่ใช้ในการค้นหานั้นจะมีแนวทางดังต่อไปนี้ คือ ทำการกำหนดตำแหน่งกล้องที่จะใช้เทียบเคียงกับกล้องจริง ซึ่งในที่นี้คือตำแหน่งของ Camera ที่แสดงในรูปที่ 3.3 ขั้นตอนต่อมาคือนำจุดศูนย์กลางซึ่งปรากฏอยู่ในเฟรมภาพที่ได้จากการคำนวณนั้นมาคูณกับ Inverse Matrix ของ Homography เพื่อจะคืนค่าตำแหน่ง แต่ค่าตำแหน่งที่ได้นั้นเป็นเพียงตำแหน่งที่ตัดกับระนาบในแนวแกน z เท่ากับศูนย์ ณ ตำแหน่ง (x, y, z) ดังรูปที่ 3.3



รูปที่ 3.3 ภาพแสดงการตัดขวางของระนาบและเส้นตรงที่จากตำแหน่งกล้องผ่านจุดศูนย์กลางทรงกลมและตัดกับระนาบ

โดยที่จุดศูนย์กลางที่แท้จริงคือจุด (u, v, w) ดังนั้นจะทำการสร้างเวกเตอร์ที่จะลากจากตำแหน่งของ Camera ลากผ่านจุด (u, v, w) และตัดกับระนาบที่ตำแหน่ง (x, y, z) ซึ่งในแนวของเวกเตอร์นั้นจะมีหลายจุดบนที่จะถูกระบุเป็นจุดศูนย์กลางได้แต่จะมีเพียงจุดเดียวที่จะมีขนาดเท่ากับรัศมีของทรงกลม

เบื้องต้นโปรแกรมจะสามารถระบุเพียงตำแหน่งของกล้องและค่าตำแหน่งของจุด (x, y, z) ที่ได้จากการคำนวณ จึงจะนำค่าทั้งสองมาสร้างเป็นเวกเตอร์ที่ลากผ่านจุดทั้งสอง เมื่อได้เวกเตอร์ดังกล่าวมาแล้วจะนำมาทำให้อยู่ในรูปของเวกเตอร์ 1 หน่วย (Unit Vector) ดังรูปที่ 3.4



รูปที่ 3.4 ภาพแสดงการทำมุมระหว่างเวกเตอร์ $\bar{n}$ เวกเตอร์ $\bar{u}$ และ เวกเตอร์ $\bar{v}$

ในที่นี้จะเรียกว่า $\bar{u}$ ก่อนที่จะมาทำการ Dot Product กับ Normal Vector $\bar{n}$ ของระนาบแนวแกน z ซึ่งมีค่ามุมเท่ากับ θ โดยที่ θ เป็นมุมระหว่างเวกเตอร์ $\bar{n}$ กับเวกเตอร์ $\bar{u}$ และนำมาประมาณมุม α ซึ่งเป็นมุมระหว่างระนาบกับ $\bar{u}$ โดยที่

$$\left|\bar{u}\right| = \left|\bar{v}\right| \bullet \sin\alpha \qquad (18)$$

เมื่อ $\left|\bar{v}\right|$ คือค่าคงที่ในที่นี้คือค่าประมาณรัศมีของทรงกลม จากการแก้สมการข้างต้นจะได้มาซึ่งขนาดของเวกเตอร์ $\bar{u}$ ขั้นตอนต่อไปจะทำการคำนวณเพื่อหาค่าตำแหน่ง (u,v,w) ดังสมการต่อไปนี้

$$(u,v,w) = (x,y,z) - \left|\bar{u}\right| \bullet \bar{u} \qquad (19)$$

อัลกอริทึมในการคืนค่าตำแหน่งของทรงกลมในสามมิติ

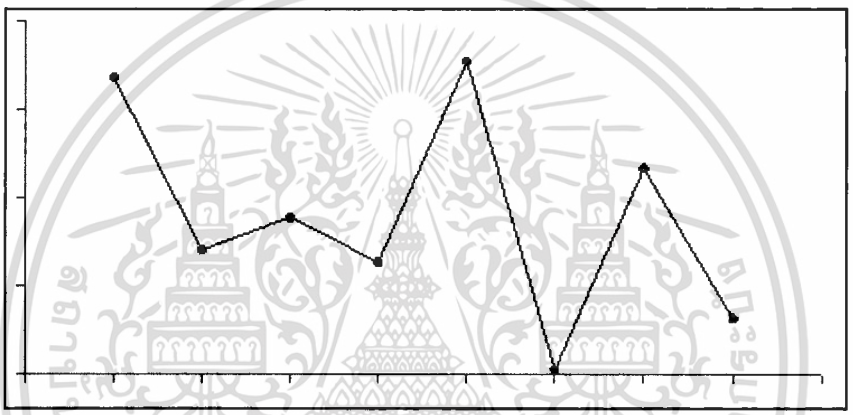
```
wCenter = new double[3];

//Mul Matirx
wCenter[0]= ((imgCenter[0]*invM[0][0])+(imgCenter[1]*invM[0][1])+(imgCenter[2]*invM[0][2]))
wCenter[1]= ((imgCenter[0]*invM[1][0])+(imgCenter[1]*invM[1][1])+(imgCenter[2]*invM[1][2]))
wCenter[2]= ((imgCenter[0]*invM[2][0])+(imgCenter[1]*invM[2][1])+(imgCenter[2]*invM[2][2]))

//center of world
temp = (wCenter[0]/wCenter[2]);
wCenter[0]=temp;
temp = (wCenter[1]/wCenter[2]);
wCenter[1]=temp;
wCenter[2]=1.00;

//find vector between Image Plane & Camera
vecU[0]=wCenter[0]-cameraCo[0];
vecU[1]=wCenter[1]-cameraCo[1];
vecU[2]=wCenter[2]-cameraCo[2];

temp=sqrt((vecU[0]*vecU[0])+(vecU[1]*vecU[1])+(vecU[2]*vecU[2]));
norVec[0]=vecU[0]/temp;
norVec[1]=vecU[1]/temp;
norVec[2]=vecU[2]/temp;

//dot product with z-axis;
cAlfa=(norVec[0]*normalVec[0])+(norVec[1]*normalVec[1])+(norVec[2]*normalVec[2]);

float arcCosAlfa=acos(cAlfa);
float alfaAxis=arcCosAlfa*(180/pi);
float zetaAxis=180-alfaAxis;
float zeta=zetaAxis*(pi/180);
float fa =((1)/sin(zeta));

norVec[0]=(fa)*norVec[0];
norVec[1]=(fa)*norVec[1];
norVec[2]=(fa)*norVec[2];

wCenter[0]=wCenter[0]-norVec[0];
wCenter[1]=wCenter[1]-norVec[1];
```

3.2.3 ขั้นตอนการแสดงผลสามมิติด้วย OpenGL

ในการแสดงผลของ OpenGL นั้นระบบในแนวแกนของการแสดงผลนั้นจะไม่เหมือนกับระบบในการคำนวณเบื้องต้นดังนั้นก่อนการแสดงผลจริงโปรแกรมจะทำการปรับแนวแกนจากเดิมการคำนวณนั้นจะระบุให้ระนาบแนวแกน z มีค่าเป็นศูนย์ แต่ในการแสดงผลด้วย OpenGL นั้นจะปรับให้เป็นแนวแกน y เป็นค่าศูนย์ หลังจากโปรแกรมจะทำการเก็บค่าตำแหน่งที่เป็นไปได้ทั้งหมดแล้วนั้นก็พร้อมที่จะแสดงผลในรูปแบบของงานถ้าด้านกราฟฟิค แต่ถ้าโปรแกรมนำค่าตำแหน่งที่ได้มาแสดงผลทันทีนั้น รูปแบบของการแสดงผลจะไม่ต่อเนื่องขาดความนุ่มนวลและสมจริงจึงได้นำการ Interpolation เข้ามาเพื่อช่วยให้การแสดงผลนั้นสมจริงมากยิ่งขึ้นโดยที่ได้ทำการเลือกวิธีการที่เรียกว่า Cosine Interpolation ทำการประมวลผลค่าตำแหน่งเพิ่มในโปรแกรม

3.2.3.1 Cosine Interpolation Methods

ฟังก์ชันในการทำ Interpolation Methods นั้นจะเป็นการคำนวณพื้นฐานของงานทางด้านกราฟฟิคเพื่อ Smooth ภาพที่จะนำมาแสดงผล ซึ่งการทำ Interpolation นั้นเป็นคุณลักษณะการประมานเส้นโค้งที่ลากผ่านจุดโดยจะประมานค่าจากจุดสองจุดที่ถูกระบุค่าไว้ ซึ่งวิธีที่จะทำการ Interpolation นั้นสามารถทำได้หลากหลายรูปแบบ อนึ่งในการทำโปรแกรมนี้นั้นได้เลือกวิธี Cosine Interpolation ซึ่งเมื่อนำมาเปรียบเทียบกับวิธีอื่นจะคำนวณได้ง่ายและมีความโค้งและค่าตำแหน่งที่เพียงพอต่อการแสดงผลในงานชิ้นนี้แล้ว

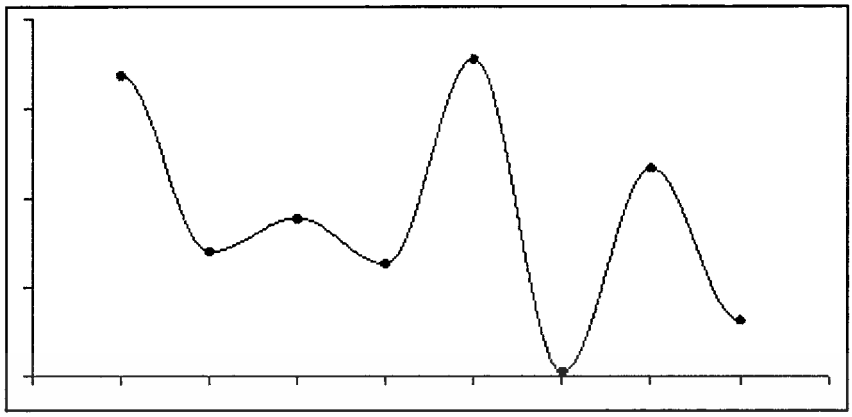


แผนภาพที่ 3.1 แสดงการ interpolation ด้วยวิธี Linear Interpolation

จากภาพเป็นการแสดงการการแสดงผลภาพแบบ Linear Interpolation ซึ่งจากกราฟจะสามารถวิเคราะห์ได้ว่าการแสดงผลในงานทางด้านกราฟฟิคเมื่อถึงจุดเชื่อมต่อจะเกิดการกระตุกหรือว่าเปลี่ยนแนวตำแหน่งไปอย่างรวดเร็วทำให้ขาดความสมจริง ในขณะที่ถ้าเลือกใช้ Cosine Interpolation นั้นจะได้ผลการคำนวณดังรูปที่ 3.2 และมีรูปแบบสมการดังต่อไปนี้

$$f(x) = \left[y_1 + (y_2 - y_1)\right] \bullet \left[\left[1 - \cos(\pi \bullet x)\right] \div 2\right] \qquad (20)$$

เมื่อค่า x คือค่าคงที่โดยจะคิดจากอัตราความเร็วของภาพที่จะใช้นำเสนอ (Frame Rate)



แผนภาพที่ 3.2 แสดงการ interpolation ด้วยวิธี Cosine Interpolation

ซึ่งจะมีความโค้งของเส้นที่มีความสมจริงมากกว่าและแสดงผลในงานทางด้านกราฟฟิค ได้ดีกว่าแบบ Linear Interpolation ฟังก์ชันในการทำ interpolation

```
double CosineInterpolate(
   double y1,double y2,
   double mu)
{
   double mu2;

   mu2 = (1-cos(mu*PI))/2;
   return(y1*(1-mu2)+y2*mu2);
}
```

บทที่ 4
ผลการทดสอบ

ในการทดสอบความถูกต้องและประสิทธิภาพของขั้นตอนการสร้างโปรแกรมจำลองการเคลื่อนที่ของทรงกลมบนพื้นราบนั้น แต่ละขั้นตอนจะมีกระบวนการเปรียบเทียบที่แตกต่างกันเพื่อทดสอบอัลกอริทึมโดยที่จะมีการกระทำการเปรียบเทียบกับค่าตำแหน่งในระบบ World Coordinate เนื่องจากการทำการจำลองแบบการเคลื่อนที่ด้วยโปรแกรม Maya ซึ่งสามารถอ่านค่าตำแหน่งของจุดศูนย์กลางทรงกลมได้ในแต่ละเฟรมของไฟล์วีดีโอที่ตัวโปรแกรม Maya ทำการ Render ขึ้น รวมทั้งสามารถทำการทดสอบการคำนวณหาค่าเมตริกซ์ต่างๆ ที่ได้จากโปรแกรมนำมาเปรียบเทียบเพื่อตรวจสอบความถูกต้องของการคำนวณกับค่าของเมตริกซ์ต่างๆที่คำนวณได้จากโปรแกรม Matlab ซึ่งเป็นโปรแกรมที่ถูกใช้เพื่อการคำนวณทางด้าน Image Processing อย่างแพร่หลาย โดยมีผลการดำเนินการทดลองดังนี้

4.1 การทดลองหาค่า Pixel ที่มีความสว่างมากกว่า 200

เป็นการทดลองเพื่อตรวจสอบการทำงานขั้นตอนการ Calibration ของโปรแกรมในส่วนของการหาตำแหน่ง Marker ซึ่งถูกกำหนดด้วยโปรแกรม Maya ให้เป็นสีขาวดังนั้นในการทดสอบจะทำการค้นเฟรมภาพที่ทำการ Capture มานั้นที่ตำแหน่ง Pixel ใดภายในเฟรมภาพมีความสว่างเกินกว่า 200 ซึ่งโปรแกรมจะทำการหาทั้งหมด 4 จุด ซึ่งได้พบทั้งหมดดังต่อไปนี้

หลัก(Column)	139	188	120	210
แถว(Row)	108	108	222	222

ตารางที่ 4.1 แสดงจุด Maker ที่โปรแกรมอ่านค่าได้

โดยจะทำการตรวจสอบของ Pixel ที่อ่านค่าได้กับตำแหน่งไฟล์ภาพที่เฟรมเดียวกันแต่ไฟล์นั้นจะถูกอ่านค่าด้วยโปรแกรม Matlab โดยจากการทดสอบปรากฏว่า ณ ที่ตำแหน่งซึ่งเก็บค่าได้ดังที่ตารางที่ 4.1 นั้นทุก Pixel มีค่าความสว่างเกิน 200 ซึ่ง Pixel ดังกล่าวโปรแกรมจะกำหนดให้เป็นจุดที่จะถูกนำมาเทียบเคียงกับค่าตำแหน่งที่กำหนดไว้ใน World Coordinate ดังตารางที่ 4.2

แกน X (Column)	0.071	3.925	0.047	3.941
แกน Y (Row)	13.915	13.916	1.094	1.095

ตารางที่ 4.2 แสดงค่า World Coordinate ที่กำหนดขึ้น

4.2 การคำนวณหา Homography Matrix และ Inverse Matrix

จากขั้นตอนการพัฒนาโปรแกรม ที่ได้กล่าวไปแล้ว จะทำการนำจุดที่ได้กำหนดเป็น Marker นำมาคำนวณหา Homograghy Matrix และ Inverse Matrix ซึ่งจะได้ค่า Homography Matrix ดังต่อไปนี้

ค่า Homography Matrix

24.8145	11.0787	115.715
-0.028	-2.152	241.009
-0.001	0.068	1

ตารางที่ 4.3 แสดงค่าของ Homography Matrix

การทดสอบค่าของ Homography Matrix ที่คำนวณได้นั้นจะทำทดสอบด้วยโปรแกรม Matlab โดยการนำ Homography Matrix ไปคูณกับตำแหน่งใน World Coordinate ที่ถูกกำหนดขึ้น ผลลัพธ์ของการคูณนั้นคือ Pixel ที่ถูกกำหนดให้เป็น Marker โดยได้ผลลัพธ์ดังตารางที่ 4.4

แกน X	แกน Y	ค่าหลักที่คำนวณได้	ค่าแถวที่คำนวณได้
0.071	13.915	139.5801	108.4538
3.925	13.916	189.0873	108.6037
0.047	1.094	120.0795	222.1478
3.941	1.095	210.7697	222.8325

ตารางที่ 4.4 แสดงการทดลองนำจุดใน World Coordinate มาประมาณค่าจุดที่ควรปรากฏในภาพ

โดยผลลัพธ์ที่ได้จากโปรแกรม Matlab ตรวจสอบได้ว่าจะเกิดค่าความผิดพลาดประมาณ 12.5%

ทำการนำ Homography Matrix มาทำการหาค่า Inverse ด้วยฟังก์ชันภายในโปรแกรมได้ผลลัพธ์ดังตารางที่ 4.5

0.004	0.006	-6.292
0.0001	-0.053	12.8996
0.0001	0.003	0.114

ตารางที่ 4.5 แสดงค่าของค่า Inverse Matrix ของ Homography

และนำมาเทียบเคียงกับ Inverse Matrix ที่คำนวณด้วยโปรแกรม Matlab

0.0040053	0.0069346	-6.306
0.00046016	-0.053856	12.927
0.0001	0.0036691	0.11469

ตารางที่ 4.6 แสดงค่าของค่า Inverse Matrix ของ Homography ที่ได้จากโปรแกรม Matlab

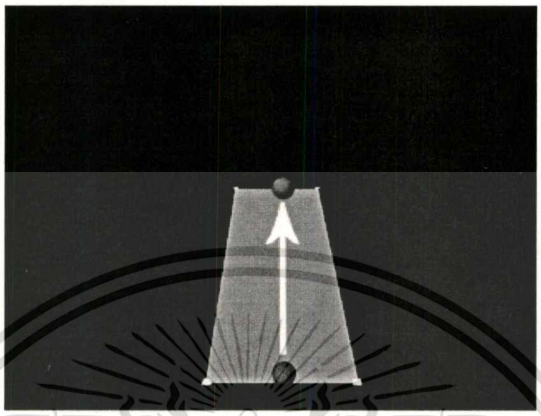
ซึ่งผลลัพธ์ที่ได้จากโปรแกรมนั้นจะพบค่าความผิดพลาด 11.11 %

4.3 การทดสอบอัลกอริทึมการหาตำแหน่งในระนาบสามมิติ

เมื่อผ่านกระบวนการขั้นตอนการ Calibration รวมถึงการหาวัตถุทรงกลมทั้งหมดแล้วนั้น จะนำจุดที่ได้จากโปรแกรมซึ่งเป็นจุดที่นำไปสร้างงานทางด้าน Graphic นำมาเปรียบเทียบกับจุดที่ถูกสร้างและอ่านค่าด้วยตำแหน่งด้วยโปรแกรม Maya เพื่อประมาณค่าความผิดพลาดที่เกิดขึ้น

4.3.1 การทดสอบครั้งที่ 1

การทดสอบเบื้องต้นโดยการเคลื่อนที่ของทรงกลมในแนวแกน x นั้นไม่มีการเบี่ยงเบนดังภาพ โดยลูกศรสีขาวแสดงแนวทางการเคลื่อนที่ของทรงกลม



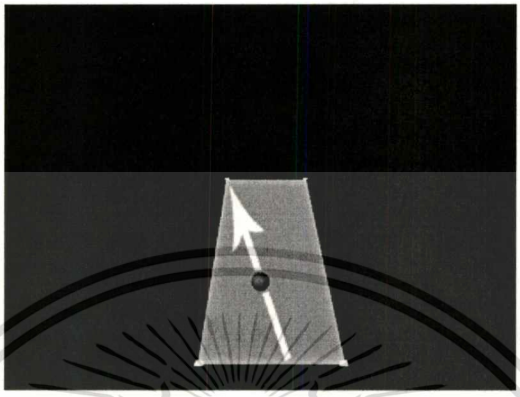
รูปที่ 4.1 ภาพแสดงแนวการเคลื่อนที่ของทรงกลมในการทดลองที่ 1 แสดงเฟรมที่ 20

เฟรมภาพที่	แกนx	แกนY	แกน X ที่ได้จากโปรแกรม	แกน Yที่ได้จากโปรแกรม
20	2.04	0.849	2.0843	0.8736
30	2.04	1.783	2.1	1.963
40	2.04	2.605	2.115	2.9455
50	2.04	3.51	2.08	4.239
60	2.04	4.45	2.096	5.3907
70	2.04	5.462	2.111	6.368
80	2.04	6.476	2.029	8.0785
90	2.04	7.408	2.075	9.0288
100	2.04	8.338	1.98	-0.326
110	2.04	9.342	1.98	-0.326
120	2.04	10.347	1.98	-0.326
130	2.04	11.264	1.98	-0.326

ตารางที่ 4.7 แสดงผลการทดสอบเบื้องต้นโดยการเคลื่อนที่ของทรงกลมในแนวแกน x นั้นไม่มีการเบี่ยงเบน

4.3.2 การทดสอบครั้งที่ 2

การทดสอบโดยการเคลื่อนที่ของทรงกลมในแนวแกน x นั้นมีการเบี่ยงเบนไปทางด้านซ้าย ดังภาพ



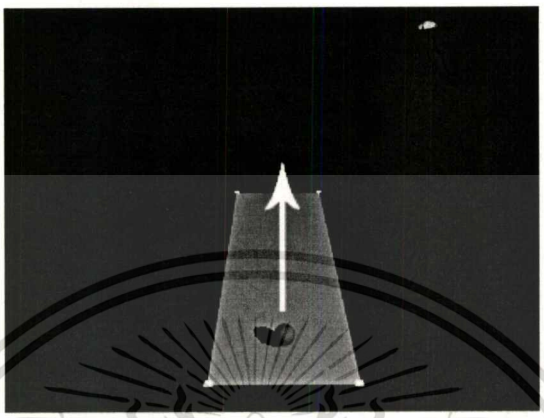
รูปที่ 4.2 ภาพแสดงการเคลื่อนที่ของทรงกลมในการทดลองที่ 2 แสดงเฟรมที่ 60

เฟรมภาพที่	แกน X	แกน Y	แกน X ที่ได้จากโปรแกรม	แกน Yที่ได้จากโปรแกรม
20	1.931	0.244	1.997	0.345
30	1.874	1.307	1.92	1.65914
40	1.817	2.369	1.877	2.9456
50	1.76	4.494	1.822	4.2398
60	1.703	5.557	1.769	5.50266
70	1.645	6.619	1.645	6.93661
80	1.588	7.682	1.57075	8.1473
90	1.531	8.744	1.4402	9.052
100	1.474	9.807	1.55	-0.72
110	1.417	10.869	1.55	-0.72
120	1.359	11.932	1.55	-0.72
130	1.302	12.994	1.55	-0.72

ตารางที่ 4.8 แสดงผลการทดสอบเบื้องต้นโดยการเคลื่อนที่ของทรงกลมในแนวแกน x นั้นมีการเบี่ยงเบนไปทางด้านซ้าย

4.3.3 การทดสอบครั้งที่ 3

การทดสอบเบื้องต้นโดยการเคลื่อนที่ของทรงกลมแนวแกน x นั้นไม่มีการเบี่ยงเบนและทำการใส่ค่าความสว่างเพื่อให้ทรงกลมเกิดเงาดังภาพ



รูปที่ 4.3 ภาพแสดงการเคลื่อนที่ของทรงกลมและเงาที่เกิดขึ้นในการทดลองที่ 3 แสดงเฟรมที่ 40

เฟรมภาพที่	แกนx	แกนY	แกน X ที่ได้จากโปรแกรม	แกน Yที่ได้จากโปรแกรม
20	2.04	0.849	1.940	0.86
30	2.04	1.783	1.900	1.940
40	2.04	2.605	1.95	2.92
50	2.04	3.51	2.01	4.28
60	2.04	4.45	2.096	5.46
70	2.04	5.462	2.19	6.648
80	2.04	6.476	2.297	7.699
90	2.04	7.408	2.32	8.99
100	2.04	8.338	2.35	10.28
110	2.04	9.342	2.45	11.53
120	2.04	10.347	1.98	-0.326
130	2.04	11.264	1.98	-0.326

ตารางที่ 4.9 แสดงผลการทดสอบเบื้องต้นโดยการเคลื่อนที่ของทรงกลมแนวแกน x นั้นไม่มีการเบี่ยงเบนและทำการใส่ค่าความสว่างเพื่อให้ทรงกลมเกิดเงา

4.4 สรุปผลการทดลองกรณีข้างต้น

จากจุดที่ได้ผลการทดลองระบุได้ว่าค่าประมาณของ Homography นั้นสามารถแสดงผลให้ได้พอประมาณ และเมื่อวัตถุที่สนใจศึกษาเกิดเงา ซึ่งในที่นี้ได้ใช้โปรแกรมช่วยสร้างในจะทำให้การวิเคราะห์การประเมินตำแหน่งของจุดศูนย์กลางมีความผิดพลาดมากขึ้น ซึ่งความผิดพลาดที่เกิดนั้นจะเกิดในระนาบแนวแกน X แต่ในระนาบแนวแกน Y นั้นจะวิเคราะห์ได้ละเอียดมากขึ้น อีกทั้งข้อผิดพลาดอีกประกายที่จะเกิดเพิ่มขึ้นเมื่อวัตถุนั้นออกห่างจากกล้องเพิ่มขึ้น รวมถึงกรณีที่เกิดภาพเสมือนว่าลูกทรงกลมได้วิ่งออกนอกระนาบที่เป็นตัวกำหนดตำแหน่งไปแล้วนั้นจะประมาณค่าแล้วผิดพลาดด้วย

4.5 การทดสอบการแสดงผลของโปรแกรม

หลังจากการคำนวณค่าแล้วโปรแกรมการจำลองวิถีการเคลื่อนที่ของทรงกลมจะถูกนำเสนอด้วย OpenGL ดังภาพ

รูปที่ 4.4 ภาพแสดงการนำเสนอโปรแกรม

ในการทดสอบการแสดงผลนั้นได้ทำการทดลองเก็บค่าจากไฟล์วีดีโอและทำการนำเสนอด้วย OpenGL โดยสามารถทำการเปลี่ยนมุมกล้องจากปุ่มคอนโทรลของโปรแกรมได้อย่างถูกต้องและไม่เกิด Run Time Error

บทที่ 5
สรุปและวิจารณ์ผลโครงการ

โปรแกรมเพื่อการตรวจวัดวิถีการเคลื่อนที่ของทรงกลมบนพื้นราบนั้น พัฒนาขึ้นเพื่อนำโปรแกรมที่ได้ไปประยุกต์ใช้ในการนำเสนองานทางด้านกราฟฟิค ทำให้การทำงานได้ง่ายขึ้น และมีความเสมือนจริงมากที่สุด ที่ถูกต้องและมีความน่าสนใจมากยิ่งขึ้น ซึ่งจากขั้นตอนการศึกษาปัญหา การศึกษาความเป็นไปได้ การออกแบบ และการพัฒนาโครงการสามารถสรุปและวิจารณ์ผลการดำเนินโครงงานได้ดังนี้

5.1 สรุปผลการดำเนินโครงการ

โปรแกรมเพื่อการตรวจวัดวิถีการเคลื่อนที่ของทรงกลมบนพื้นราบที่พัฒนาขึ้นสามารถสร้างให้มีความเหมือนจริง โดยใช้หลักการทางการประมวลผลภาพมาทำการวิเคราะห์ข้อมูลเพื่อที่จะตรวจวัดตำแหน่งรวมถึงแนวการเคลื่อนที่ของทรงกลม สำหรับกระบวนการทางด้านคอมพิวเตอร์กราฟฟิคจะนำทฤษฎีที่เกี่ยวข้องกับการนำเสนอในอย่างเช่นการใช้วิธีการที่เรียกว่า Cosine Interpolation ในการประมาณวิถีการเคลื่อนที่ทั้งหมดของทรงกลมเพื่อความเสมือนจริงและราบรื่น จากผลการทดลองในบทที่ 4 สามารถสรุปปัจจัยที่มีผลต่อการดำเนินโครงการได้ ดังนี้

5.1.1 ภาพ

1. ขนาดจุดของสัญญาณรบกวนบนที่เกิดขึ้นบนภาพ ถ้าขนาดของจุดมีขนาดใหญ่จะทำให้ผลของการประเมินตำแหน่งของวัตถุผิดพลาด

2. ขนาดของภาพต้นแบบของวัตถุ ถ้าทรงกลมที่สนใจศึกษานั้นมีขนาดเล็กลงมากขึ้นเท่าไหร่ความผิดพลาดของตำแหน่งของการประเมินค่าจะมีค่ามากขึ้นไปด้วย เพราะโปรแกรมอาจจะพิจารณาว่าสิ่งที่เกิดขึ้นเป็นสัญญาณรบกวนไม่ใช่วัตถุที่จะทำการศึกษา

3. ค่าความสว่างแสงและเงาจะมีผลต่อภาพที่จะนำมาประมวลผล ยิ่งเกิดเงามากขึ้นเท่าไหร่ความผิดพลาดของการประมาณจุดศูนย์กลางของทรงกลมก็จะผิดพลาดมากเพิ่มไปด้วย

5.1.2 การคำนวณค่าทางคณิตศาสตร์

การคำนวณหาค่าทางคณิตศาสตร์ที่เกิดขึ้นในโปรแกรมไม่ว่าจะเป็นค่าเมตริกซ์ การหาค่า Homography หรือค่า Inverse Matrix รวมไปถึงการประมาณค่ามุมที่เกิดขึ้นระหว่างระนาบที่สนใจศึกษากับสมการเส้นตรงที่จะลากผ่านกล้องตัดผ่านจุดศูนย์กลางนั้นจะเป็นการประมาณค่าซึ่งจะใช้ได้ผลดีในระดับหนึ่งเท่านั้นค่าที่คำนวณได้จะเกิดข้อผิดพลาดมากขึ้นถ้าข้อมูลที่ได้มาจากกระบวนการทางด้าน Image Processing ไม่เพียงพอและเป็นความจำกัดของกล้องเพียงตัวเดียวที่การกู้คืนค่าในแนวแกนลึกจะทำได้ไม่ครบถ้วน

5.1.3 ปัจจัยที่มีผลต่อการนำแสดงผลภาพ

1. ตำแหน่งที่คำนวณได้มีความผิดพลาดมากน้อยแค่ไหน
2. จำนวนอัตราการแสดงภาพที่นำเสนอระหว่างกล้องและการแสดงผลภาพจะต้องทำการปรับให้มีความใกล้เคียงกัน
3. ค่าความสว่างแสงของแสงมีผลกระทบต่อการทำงาน
4. ประสิทธิภาพของฮาร์ดแวร์

5.2 ข้อเสนอแนะ

สำหรับโปรแกรมนี้สามารถนำไปปรับปรุงและพัฒนาเพื่อประยุกต์ใช้ได้ จึงจะเสนอแนวทางในการพัฒนาดังนี้

1. การพัฒนาอัลกอริทึมควรจะมีประสิทธิภาพมากกว่านี้เพราะค่าตำแหน่งที่ได้มานั้นยังพบความผิดพลาดอยู่มาก
2. ส่วนของส่วนติดต่อกับผู้ใช้นั้นควรจะทำให้ใช้งานได้ง่ายกว่านี้
3. ในเบื้องต้นนั้นยังทำการทดสอบกับไฟล์วีดีโอที่สร้างจากโปรแกรม Maya ขั้นตอนการพัฒนาต่อควรจะทำการติดต่อกับกล้องจริงและทำงานแบบ Real-Time ด้วย

บรรณานุกรม

Bob Fisher. Computing Plane Projective Transformations. [Online]. Available : http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/EPSRC_SSAZ/epsrc_ssaz.html. 1997.

Edward Angel. 2003 **Interactive Computer Graphics A Top-Down Approach with OpenGL Third Edition.** New York : Addison Wesley.

Gregory, A Baxes. 1994. **Digital Image Processing Princibles And Applications.** New York : John Wiley and Sons

Linda, G and George, C. 2001 **Computer Vision** New York : Prentice-Hall

Marc Pollefeys. **Visual 3D Modeling from Images.** [Online]. Available : http://www.cs.unc.edu/~marc/tutorial. 2001.

Paul Bourke. **Interpolation methods.** [Online]. Available : http://astronomy.swin.edu.au/~pbourke/analysis/interpolation. 1999.

Peter Kovesi. **MATLAB Functions for Computer Vision and Image Analysis.** [Online]. Available: http://www.csse.uwa.edu.au/~pk. 2001.

Ramesh, J et al. 1995 **Machine Vision** New York : McGraw-Hill

Richard, I Hartley. VOL. 19, NO. 6, June 1997. "In Defense of the Eight-Point Algorithm" In Transactions On Pattern Analysis And Machine Intelligence **IEEE Computer Society**

Richard, H and Andrew, Z. June 1999. "Multiple view Geometry" In Computer Vision and Pattern Recognition **IEEE Computer Society.**

Richard, S and Michael, S. 1999. **OpenGL SuperBible, Second Edition.** Indiana : Waite Group Press

Sebastien Grange. **TLIB 1.1 documentation.** [Online]. Available : http://imtsg14.epfl.ch/tlib/index.html. 2005

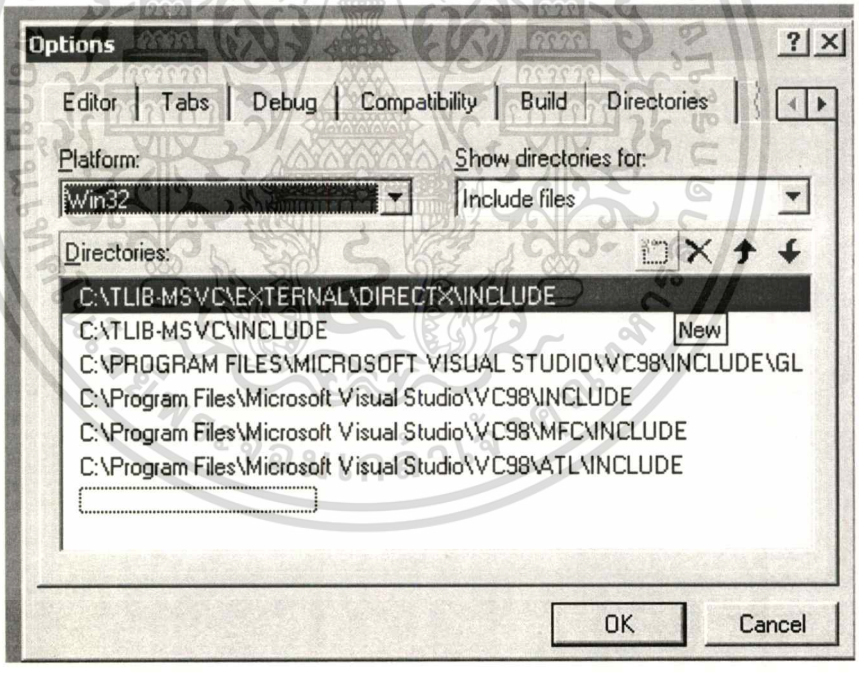
William, H Press. **Numerical Recipes in C++: The Art of Scientific Computing.** [Online]. Available : http://www.library.cornell.edu/nr/bookcpdf.html. 2001

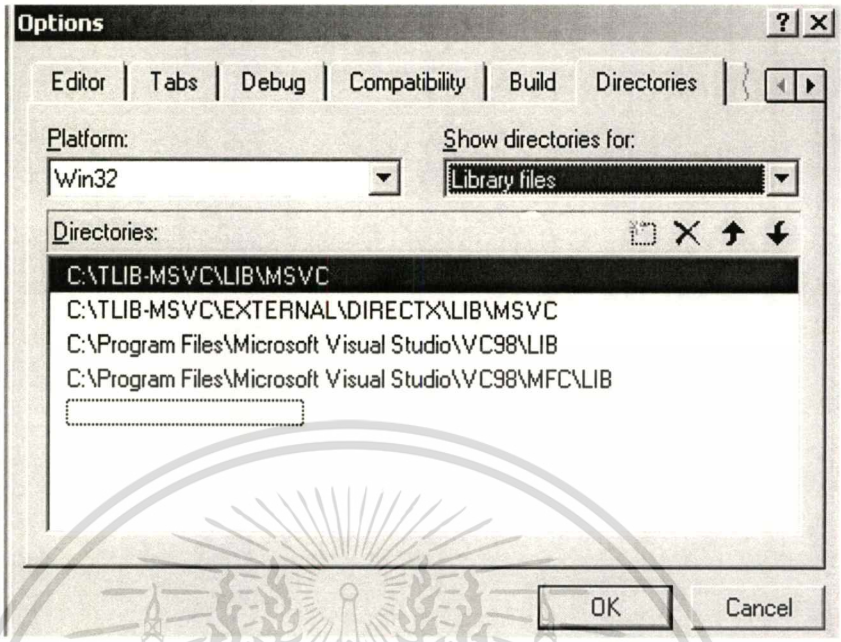
ภาคผนวก

ภาคผนวก ก
การใช้งานไลบรารี TLIB

ไลบรารี TLIB เป็นไลบรารีที่ใช้ในการช่วยโปรแกรมทางด้านการประมวลผลภาพ Image Processing ในการเรียกใช้ไลบรารีนั้นจะต้องทำการติดตั้งเพิ่มลงไปบน Microsoft Visual C++ แล้วจึงนำไปพัฒนาต่อ ซึ่งขั้นตอนการติดตั้งประกอบไปด้วย

1. การแตกไฟล์จัดเก็บไว้ที่ c:\tlib-msvc
2. ติดตั้งเพิ่มลงบน Microsoft Visual C++ ดังขั้นตอนต่อไปนี้ คลิกเลือก Tools>Options>Directories ทำการเพิ่มลงใน Include files และใน Library file ดังภาพ





การสร้างโปรเจคใน Microsoft Visual C++ ในฟังก์ชัน Main ของ C++ รวมทั้ง Header file และเพิ่มไลบรารีที่เกี่ยวข้อง

```
// tlibboDlg.h : header file
//

//tlib to include
#include "tlVision.h"
#include "tlDisplay.h"
#include "tlSourceVideoFile.h"

//standard & user library
#include "math.h"
#include "OpenGLControl.h"
#include <stdlib.h>
#include <windows.h>
#include <string.h>

//OpenGL Library
#include <gl\gl.h>
#include <gl\glut.h>

#define EPS 1e-10
#define pi 3.1416
```

ภาคผนวก ข
คู่มือโครงสร้างของโปรแกรม

โครงสร้างของ TLIB

ในการพัฒนาโปรแกรมนั้นจะทำการ Include Header File จาก TLIB ไลบราลีเข้าสู่ Microsoft Visual C++ 6.0 ที่จำเป็นเข้าสู่โปรแกรมประกอบไปด้วย tlVision.h , tlDisplay.h และ tlSourceVideoFile.h ตามภาคผนวก ก.

tlVision.h

ไม่ได้ถูกจัดให้อยู่ในรูปแบบของ Class แต่จะเป็นส่วนที่จะทำการจัดเก็บ Main TLIB header file เกือบทั้งหมดของไลบารีเพื่อสะดวกและง่ายต่อการเรียกใช้ Hearder ไฟล์ตัวอื่น ซึ่งในโปรแกรมได้ทำการเรียกใช้ tlImage.h ซึ่งเป็น Class ที่จะใช้ในเข้าถึงไฟล์ภาพเพื่อจัดการข้อมูลต่างๆที่อยู่ในภาพในระดับ Pixel เช่น drawing functions, file read/write เป็นต้น และ tlColor.h ซึ่งเป็น Class ที่ใช้ในการเข้าถึงค่าสีรวมทั้งถูกใช้เพื่อจัดการการจัดเก็บค่าสีในรูปแบบต่างๆที่ TLIB รองรับ ผ่านการนำเข้าของ tlVision.h

Constructor Documentation

tlImage::tlImage()

เป็น Constructor Class ที่ถูกสร้างเพื่อมารองรับการจัดเก็บประเภทไฟล์รูปภาพด้วย TLIB

tlColor::tlColor()

เป็นการสร้างค่าเริ่มต้นของตัวแปรและกำหนดค่าสีในตัวแปรให้เป็น RGB

Member Function

Int tlPixelArray::toGray(tlPixelArray * dest)

เนื่องจาก tlImage นั้นเป็น Class ลูกของ tlPixelArray ซึ่งได้รับการถ่ายทอดคุณสมบัตินี้มา ซึ่งฟังก์ชันนี้จะทำการเข้าถึงในแต่ละ Pixel ของรูปภาพแล้วทำการเปลี่ยนให้อยู่ในรูปของ Gray

Scale พารามิเตอร์ dest นั้นจะเป็นที่จัดเก็บไฟล์ภาพหลังจากการเปลี่ยนเป็น Gray Image แล้ว โดยฟังก์ชั่นจะคืนค่า -1 เมื่อเกิดข้อผิดพลาด

Int tlPixelArray::sub(tlPixelArray * pixelArray , tlPixelArray * dest)

เป็นฟังก์ชั่นในการนำค่าตำแหน่งในไฟล์ภาพมาทำการลบกันโดยส่งผ่านพารามิเตอร์ pixelArray เมื่อ pixelArray คือไฟล์ภาพที่จะนำมาทำการลบกับ Current Image และ dest คือที่จัดเก็บไฟล์ภาพหลังจากผ่านฟังก์ชั่นการ sub แล้ว

Int tlPixelArray::getColor(int x,int y ,tlColor * color)

เป็นฟังก์ชันที่ใช้ในการดึงค่าความสว่างใน Pixel ที่กำหนด โดยส่งผ่านพารามิเตอร์ x , y และทำการเก็บค่าความสว่างไว้ที่พารามิเตอร์ color ที่ได้รับการประกาศเป็นตัวแปร tlColor โดยที่ x,y คือตำแหน่งในไฟล์ภาพที่ต้องการอ่านค่าความสว่างโดยสามารถเรียกดูค่าความสว่างผ่านตัวแปร Public ของ Class tlColor ที่ตัวแปร intensity

tlDisplay.h

เป็น Class ที่ใช้เพื่อจัดการแสดงผลรูปภาพและ Event ต่างๆ เช่นหน้าจอการแสดงผลการ Render ไฟล์ภาพรวมถึงควบคุมการทำงานของคีย์บอร์ดและเม้าส์

Constructor Documentation

tlDisplay:: tlDisplay(char * name)

ถูกใช้เพื่อการรองรับ Class tlDisplay ที่จะทำการสร้างหน้าต่างเพื่อแสดงผลโดยพารามิเตอร์ที่ส่งค่ามานั้นคือ Window Name ที่จะถูกแสดงผล

Member Function

Int tlDisplay::show()

เป็นฟังก์ชันที่ใช้เรียกเพื่อสร้าง Windows ที่จะใช้แสดงผล

Int tlDisplay::display(tlImage * image)

ใช้เพื่อแสดงผลของภาพลงบน Windows Area ที่ถูกสร้างจากฟังก์ชัน Show ก่อนหน้า

Int tlDisplay::close ()

เป็นฟังก์ชันที่ใช้เรียกเพื่อทำลาย Windows ที่จะใช้แสดงผล

tlSourceVideoFile.h

เป็น Class ที่ได้รับการถ่ายทอดคุณสมบัติมาจาก Class tlSource ถูกเรียกใช้เพื่อ Implement กับงานประเภทวีดีโอไฟล์

Constructor Documentation

tlSourceVideoFile:: tlSourceVideoFile(char * filename)

ถูกสร้างเพื่อใช้ในการส่งผ่านค่าวีดีโอโดยทำการส่งชื่อไฟล์ที่ต้องการทำการเปิด

Member Function

Int tlSourceVideoFile::open(char * filename)

ฟังก์ชันนี้ใช้ในการเปิดไฟล์วีดีโอที่ต้องการจะทำการแสดงผลและเก็บข้อมูลโดยจะคืนค่า -1 เมื่อมีการผิดพลาด

tlSourceVideoFile::getLength()

การทำงานในส่วนนี้จะใช้เพื่อนำค่าความยาวเฟรมทั้งหมดที่มีในวีดีโอออกมาแสดงผลหรือจัดเก็บโดยฟังก์ชันจะทำการคืนค่าความยาวในรูปแบบของตัวเลขเป็น Integer

tlSourceVideoFile::getIndex()

เป็นฟังก์ชันที่จะถูกเรียกเมื่อต้องการทราบค่าตำแหน่งในเฟรมปัจจุบันของไฟล์วีดีโอที่ทำการประมวลผล โดยจะทำการคืนค่าเป็นค่าตำแหน่งของ Current Frame

tlSourceVideoFile::grab(tlImage * image)

เป็น Virtual Function ที่จะได้มาซึ่งเฟรมของรูปภาพที่กำลังทำการแสดงผล (Current Frame) โดยที่ค่าที่สามารถ grab มาได้นั้นจะจัดอยู่ในรูปแบบของไฟล์รูปภาพที่จะจัดเก็บด้วย tlImage

โครงสร้างของ OpenGL

ในการพัฒนาโปรแกรมนั้นจะใช้รูปทรงพื้นฐานที่มีอยู่ใน OpenGL ซึ่งรูปทรงพื้นฐานนั้นสร้างได้จากการกำหนดจุดและการเรียงลำดับของจุด ซึ่งกำหนดด้วยฟังก์ชัน glVertex3f( ) การกำหนดเซตของจุดเพื่อกำหนดรูปทรงนั้น โดยจะเริ่มจากกำหนดรูปทรงที่ต้องการ ซึ่งเป็นส่วนหนึ่งของพารามิเตอร์ของฟังก์ชัน glBegin() ตามด้วยชุดของจุดยอด และปิดด้วย glEnd()

ซึ่งในที่นี้จะใช้คำสั่งนี้ในการสร้างระนาบเพื่อเป็นพื้นของสนามโบว์ลิ่ง และมีจุดที่ต้องทำการกำหนดทั้งหมด 4 จุดเพื่อสร้างรูปทรงสี่เหลี่ยมดังต่อไปนี้

```
glBegin(GL_QUADS);                          // Begin Drawing A Quad
glColor3f(0.5960f,0.501f,0.301f);
    glVertex3f(0.0, 0.0, -3.0);             // Bottom Left Corner Of Floor
    glVertex3f(0.0, 0.0,-22.0);             // Top Left Corner Of Floor
    glVertex3f( 4.0, 0.0,-22.0);            // Top Right Corner Of Floor
    glVertex3f( 4.0, 0.0, -3.0);            // Bottom Right Corner Of Floor
glEnd();
```

โดยในชุดคำสั่งที่แสดงนั้นจะมีการใช้คำสั่ง glColor3f() ซึ่งใช้ในการกำหนดค่าสีของระนาบ ส่วนลูกทรงกลมที่สร้างด้วย OpenGL นั้นจะใช้คำสั่ง glutSolidSphere() ในการสร้างและทำการกำหนดให้แสดงผลเป็นสีแดงด้วยคำสั่ง glColor3f( )

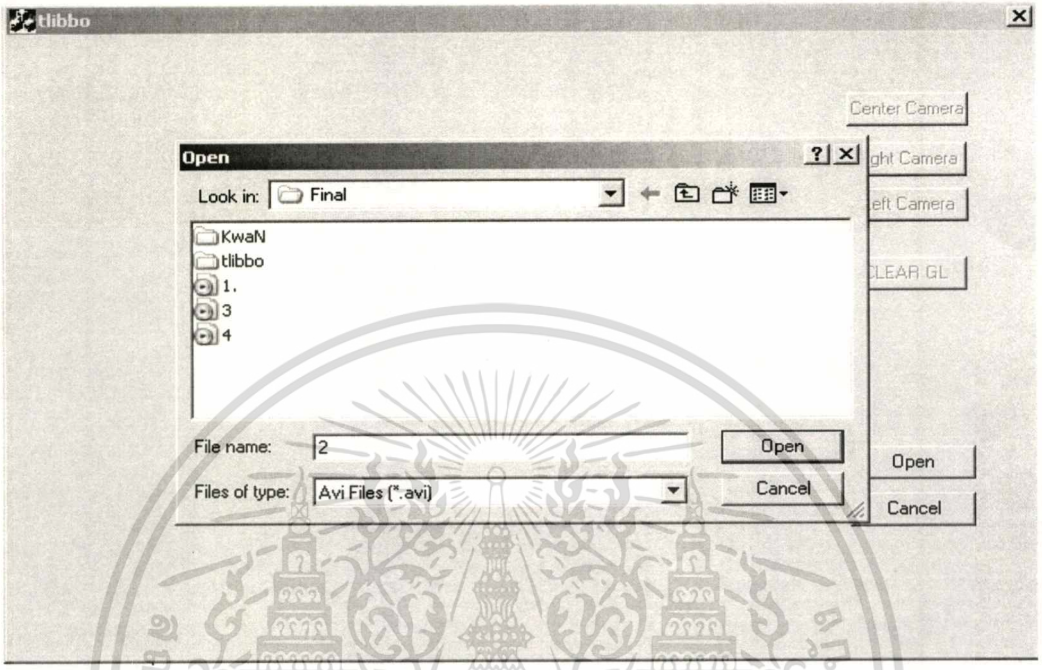
นอกจากนี้ OpenGL นั้นยังสามารถกำหนดสถานะเพิ่มเติมได้อื่นได้ เช่นการยอมให้แสดงผลหรือไม่แสดงผล ฟังก์ชันที่ใช้เพื่อให้แสดงผลคือ glEnable() ส่วนฟังก์ชันที่ไม่ยอมให้แสดงผลนั้นคือ glDisable () ตัวอย่างเช่น glEnable(GL_DEPTH_TEST) เป็นการยินยอมให้มีการเรียกฟังก์ชัน z-buffer algorithm อีกทั้งยังมีคำสั่งที่ถูกเรียกใช้ในการ Translation ซึ่งใช้ในการกำหนดตำแหน่งกล้องและทรงกลมรวมถึงทำการเลื่อนทรงกลมให้เคลื่อนที่ ฟังก์ชันในการ Translation คือ glTranslatef( )

ภาคผนวก ค
คู่มือการใช้โปรแกรม

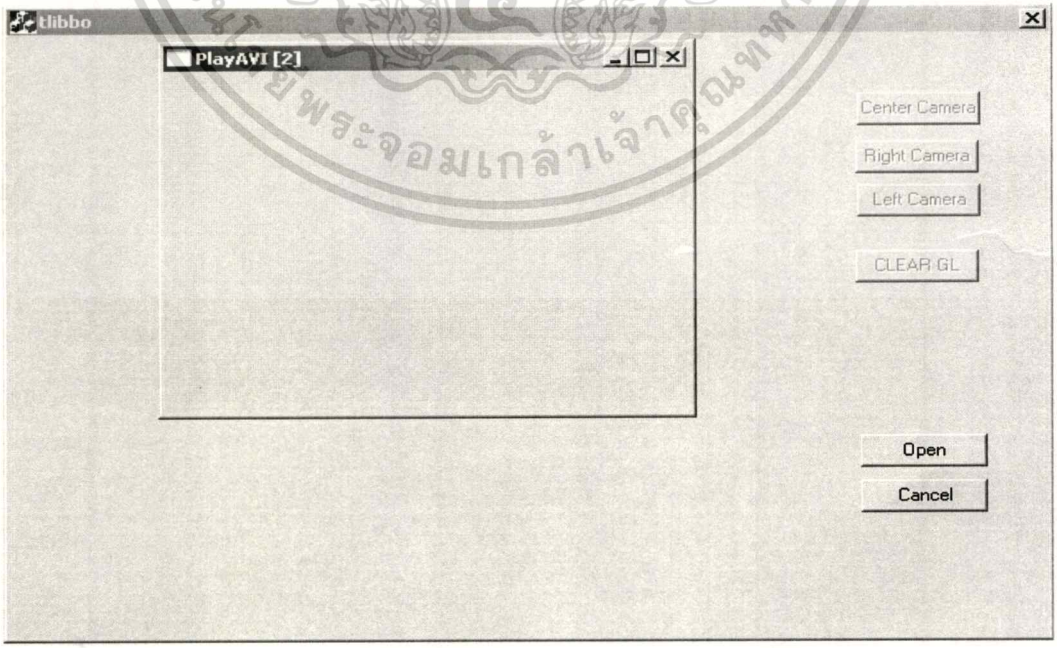
การใช้งานโปรแกรม

1. เปิดโปรแกรมที่ Start -> Programs -> tlibbo จะปรากฏโปรแกรมดังภาพ

2. ทำการเปิดไฟล์วีดีโอที่ต้องการโดยคลิก Open



3. เมื่อไฟล์วีดีโอนั้นได้ทำการเล่นจนจบผู้ใช้ทำการปิดหน้าต่างแสดงผล โปรแกรมจะ Enable ปุ่มเลือกมุมกล้องทางด้านขวาซึ่งจะประกอบไปด้วยทั้งหมดสามมุมกล้อง



4. เมื่อแสดงผลครบถ้วนผู้ใช้จะทำการล้างค่าที่แสดงผลเพื่อให้แสดงผลที่มุมกล้องอื่นได้ อีกโดนการเลือกไปที่ Clear GL ก่อนที่จะทำการเลือกมุมมองอื่น

5. ถ้าต้องการเก็บค่าจากไฟล์วีดีโอใหม่ให้เลือก Open อีกครั้ง
6. ถ้าต้องการออกจากโปรแกรมให้คลิก Cancel

ประวัติผู้เขียน

ชื่อผู้เขียน	นางสาวปรียาภา จินดาเลิศอุดมดี
วันเดือนปีเกิด	30 กรกฎาคม 2524
สถานที่เกิด	กรุงเทพมหานคร
ที่อยู่ปัจจุบัน	1300/57 ซอย สุขุมวิท 50 แขวง พระโขนง เขตคลองเตย จังหวัด กรุงเทพมหานคร
ประวัติการศึกษา	ระดับประถมศึกษา โรงเรียนประเสริฐธรรมวิทยา จังหวัดกรุงเทพมหานคร ระดับมัธยมศึกษา โรงเรียนสตรีมหาพฤฒาราม จังหวัดกรุงเทพมหานคร ระดับปริญญาตรี ภาควิชาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ มหาวิทยาลัยศรีนครินทรวิโรฒ ประสานมิตร ระดับปริญญาโท แขนงวิชาวิทยาการสารสนเทศ คณะเทคโนโลยีสารสนเทศ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง