# การจำลองคลื่นแบบมีปฏิสัมพันธ์โดยใช้ OpenGL

# INTERACTIVE WAVE SIMULATION USING OPENGL



ทศพล กาญจนพรชัย

TOSSAPON KANCHANAPORNCHAI



ปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต

ภาควิชาคณิตศาสตร์และวิทยาการคอมพิวเตอร์

คณะวิทยาศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2553

# INTERACTIVE WAVE SIMULATION USING OPENGL

**TOSSAPON KANCHANAPORNCHAI**

**A SPECIAL PROJECT SUBMITTED IN PARTIAL FULFILLMENT**

**OF THE REQUIREMENT FOR DEGREE OF BACHELOR OF SCIENCE**

**DEPARTMENT OF MATHEMATICS AND COMPUTER SCIENCE**

**FACULTY OF SCIENCE**

**KING MONGKUT'S INSTITUDE OF TECHNOLOGY LADKRABANG**

**ACADEMIC YEAR 2010**

**หัวข้อปัญหาพิเศษ** การจำลองคลื่นแบบมีปฏิสัมพันธ์โดยใช้ OpenGL

Interactive Wave Simulation Using OpenGL

**ชื่อนักศึกษา** นายทศพล กาญจนพรชัย

**รหัสประจำตัว** 50050138

**ปริญญา** วิทยาศาสตรบัณฑิต

**สาขาวิชา** วิทยาการคอมพิวเตอร์

**อาจารย์ที่ปรึกษา** ผศ.ดร.กรกช ประชุมรักษ์

คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง อนุมัติให้นับปัญหาพิเศษนี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิทยาศาสตรบัณฑิต สาขาวิชาวิทยาศาสตร์ วิทยาการคอมพิวเตอร์ ประจำปีการศึกษา 2553

| คณะกรรมการสอบ | ลายมือชื่อ |
|---|---|
| ผศ.ดร.นันทิกา เบญจเทพานันท์ | |
| ดร.วรางคณา กิ้มปาน | |
| ผศ.ดร.กรกช ประชุมรักษ์ | |

| | |
|---|---|
| **หัวข้อปริญญานิพนธ์** | การจำลองคลื่นแบบมีปฏิสัมพันธ์โดยใช้ OpenGL |
| **นักศึกษา** | นายทศพล กาญจนพรชัย |
| **รหัสประจำตัว** | 50050138 |
| **ปริญญา** | วิทยาศาสตรบัณฑิต |
| **สาขาวิชา** | วิทยาการคอมพิวเตอร์ |
| **พ.ศ.** | 2553 |
| **อาจารย์ผู้ควบคุมปริญญานิพนธ์** | ผศ. ดร. กรกช ประชุมรักษ์ |

# บทคัดย่อ

โครงงานนี้จัดทำขึ้นเพื่อการจำลองปรากฏการณ์ธรรมชาติอย่างหนึ่งซึ่งก็คือคลื่นน้ำ โดยใช้สมการและทฤษฎีจากงานวิจัยของนักวิจัยหลายๆท่าน ที่คิดค้นสมการที่สามารถจำลองคลื่นน้ำ ในโครงงานนี้จะเน้นความสำคัญที่การประมวลผลที่รวดเร็วและให้ผู้ใช้ได้มีปฏิสัมพันธ์กับแบบจำลอง ซึ่งคลื่นจะเป็นคลื่นแบบสงบราบเรียบเป็นหลักและเป็นคลื่นที่เกิดขึ้นเนื่องจากอิทธิพลของลมในมหาสมุทร เป้าหมายหลักคือโครงงานนี้เน้นความรวดเร็วในการประมวลผลจึงได้ทำการดัดแปลงสมการเดิมเพื่อให้การประมวลผลรวดเร็วโดยได้นำวิธี Multithreading เข้ามาใช้เพื่อให้สามารถประมวลผลได้รวดเร็วขึ้นอีกบน CPU รุ่นใหม่ๆที่มีการแบ่งหน่วยประมวลผลเป็นหลายส่วน และจะใช้เทคนิคของ OpenGL ทำ effect ของการสะท้อนแสงของน้ำให้สวยงามยิ่งขึ้น

| | |
|---|---|
| **Thesis Title** | Interactive Wave Simulation Using OpenGL |
| **Student** | Mr. Tossapon Kanchanapornchai |
| **Student ID.** | 50050138 |
| **Degree** | Bachelor of Science |
| **Program** | Computer Science |
| **Year** | 2010 |
| **Thesis Advisor** | Asst. Prof. Dr. Korakot Prachumrak |

# ABSTRACT

This thesis simulates one of natural phenomenon which is the ocean wave by using equation and theory from various researchers. These equation can be used to simulate behaviors of the ocean wave. In this thesis, our primary goal is to be able to simulate the ocean wave at real-time rate, And water surface can interact with objects. Wave in this thesis is calm and smooth occurred by force of wind above the ocean surface. Because our goal is to make it as fast as possible, some of the equations are modified and multithreading technique is used. In new generation CPUs, most of them have multi cores which make multithreading very useful as this technique can utilize more than one core at a time. Furthermore we have used an OpenGL technique called Cube Mapping (or Environment Mapping) to make water surface reflective and look more realistic.

# กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้สำเร็จได้ด้วยดีด้วยคำแนะนำ และคำปรึกษาต่างๆจาก ผศ. ดร. กรกช ประชุมรักษ์ ซึ่งเป็นอาจารย์ผู้ควบคุมและเป็นอาจารย์ที่ปรึกษาปริญญานิพนธ์ ได้ให้คำปรึกษา แนวคิด คำแนะนำ ความคิดเห็น และข้อเสนอแนะต่างๆที่เป็นประโยชน์อย่างมากในการทำปริญญานิพนธ์เล่มนี้ ข้าพเจ้ารู้สึกซาบซึ้งในความอนุเคราะห์จากอาจารย์และขอขอบพระคุณเป็นอย่างสูง

ข้าพเจ้าขอกราบขอบคุณ ผศ. ดร. นันทิกา เบญจเทพานันท์ และดร. วรางคณา กิ้มปาน ที่ได้มาเป็นกรรมการในการสอบให้สำเร็จลุล่วงไปได้ด้วยดี

ขอขอบคุณเหล่าบรรดาอาจารย์ พี่ๆ เพื่อนๆ น้องๆ และบุคคลอื่นๆที่เกี่ยวข้องในสาขาวิทยาการคอมพิวเตอร์ คณะวิทยาศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง ที่คอยให้คำแนะนำต่างๆและได้ให้กำลังใจเสมอมา

สุดท้ายนี้ ข้าพเจ้าขอกราบขอบคุณ บิดา มารดา และครอบครัวของข้าพเจ้าที่เป็นกำลังใจ ให้คำแนะนำ ให้การสนับสนุนในเรื่องต่างๆหลายๆเรื่อง ที่ทำให้ข้าพเจ้าสามารถทำปริญญานิพนธ์เล่มนี้สำเร็จลุล่วงไปได้ด้วยดี

คุณค่าและประโยชน์ในปริญญานิพนธ์เล่มนี้ ข้าพเจ้าขอมอบให้แด่ผู้มีพระคุณทุกท่าน และทุกๆท่านที่สนใจในปริญญานิพนธ์เล่มนี้

ทศพล กาญจนพรชัย

# สารบัญ

# สารบัญ (ต่อ)

# สารบัญ (ต่อ)

# สารบัญตาราง

ตารางที่

# สารบัญรูป

รูปที่

# บทที่ 1
# บทนำ

## 1.1 ความสำคัญและที่มาของโครงงาน

หนึ่งในปรากฏการณ์ธรรมชาติที่ซับซ้อนที่สุดอย่างหนึ่งของโลกก็คือพฤติกรรมของของเหลว หนึ่งในนั้นก็คือการแตกตัวของคลื่นน้ำ ซึ่งจะสามารถสังเกตเห็นการเคลื่อนที่และการประพฤติต่างๆ ของคลื่นได้หลายระดับ ตั้งแต่ระดับที่ละเอียดมากซึ่งก็คือละอองน้ำหรือฟองที่เกิดขึ้น ไปถึงระดับที่ระเอียดปานกลาง ซึ่งก็คือการกระเพื่อมของพื้นผิวน้ำ และในระดับที่หยาบที่สุดคือลูกคลื่นลูกใหญ่ๆซึ่งคว่ำลงและแตกตัวออก แต่ในขณะนี้ก็ยังไม่มีผู้ที่สามารถคิดค้นสมการ วิธีการ หรือทฤษฎีที่สามารถอธิบายพฤติกรรมของคลื่นได้อย่างแท้จริง ซึ่งโครงงานนี้จะใช้ทฤษฎีจาก Ocean Wave Simulation under Wind Change Effect [1] ในการจำลองคลื่นในระดับที่สามารถสังเกตได้อย่างชัดเจนโดยการสังเกตด้วยการมองของมนุษย์ เป้าหมายของงานนี้ต้องการที่จะควบคุมและจำลองพฤติกรรมต่างๆของคลื่น โดยที่ยังดูเสมือนจริงและขึ้นกับกฎต่างๆของหลักฟิสิกส์

โดยในโครงงานนี้จะนำเทคนิค Multithreading และเทคนิคต่างๆของ OpenGL เข้ามาช่วยเพื่อให้การแสดงผลสามารถทำได้อย่างสวยงามและรวดเร็ว เนื่องจาก CPU (Central Processing Unit) รุ่นใหม่ๆมีการแบ่งหน่วยการประมวลผลเป็นหลายๆ core การทำ Multithreading นั้นจะทำให้สามารถดึงสมรรถภาพของ CPU ออกมาได้อย่างเต็มที่ และการใช้เทคนิค Vertex Buffer Object ก็เป็นการใช้ GPU (Graphic Processing Unit) ในการแสดงผลซึ่งสามารถทำได้รวดเร็วกว่าใช้ CPU มาก

## 1.2 วัตถุประสงค์ของโครงงาน

1. ศึกษาการสร้าง 3D application ด้วย OpenGL
2. Application สามารถทำงานได้บนหลาย Platform หรือ OS
3. จำลองธรรมชาติของคลื่นน้ำที่เกิดจากอิทธิพลของลมบนคอมพิวเตอร์
4. สามารถทำงานได้รวดเร็วและประหยัดทรัพยากรของเครื่อง
5. สามารถนำมาพัฒนาต่อเป็นเกมหรือ Application แบบ real-time ได้
6. ใช้ OpenGL ทำ Visual Effect ของน้ำให้ดูเสมือนจริงหรือใกล้เคียง
7. Application สามารถทำงานแบบขนานได้ในส่วนที่ต้องใช้เวลามากในการคำนวณ
8. นำเทคนิคขั้นสูงต่างๆของ OpenGL มาใช้เพื่อช่วยในการแสดงผล

### 1.3 ขอบเขตของโครงงาน

1. สร้าง Application ที่สามารถจำลองคลื่นน้ำบน OpenGL
2. จำลองพฤติกรรมของคลื่นในระดับที่สังเกตเห็นได้
3. การจำลองไม่ขัดแย้งกับหลักของฟิสิกส์ของไหล
4. มี Interaction ระหว่างน้ำกับวัตถุ
5. นำงานวิจัยที่เกี่ยวกับคลื่นมาสร้างเป็น algorithm ซึ่งทำงานร่วมกับ OpenGL ได้
6. ใช้ Phong Shading ในการคำนวณ Normal Vector เพื่อจำลองความโค้งมนของลูกคลื่น
7. ใช้เทคนิค Multithreading เพื่อประมวลผลแบบขนานที่ทำงานได้ดีบน CPU ที่มีหลาย core
8. ใช้เทคนิค Vertex Buffer Object ของ OpenGL มาใช้เพื่อความรวดเร็วในการแสดงผล
9. ใช้กฎของ Archimedes ในการคำนวณการลอยตัวของวัตถุในน้ำหรือบนผิวน้ำ
10. ใช้ OpenGL ทำการจำลองการสะท้อนของผิวน้ำเพื่อให้มีความสวยงามโดยใช้วิธี Cube Mapping

### 1.4 ขั้นตอนการดำเนินงาน

1. ศึกษาการสร้าง 3D Application ด้วย OpenGL
2. ศึกษา Programming library ต่างๆที่จำเป็นในการสร้าง Application
3. ศึกษาขั้นตอนและวิธีการจำลองคลื่นแบบต่างๆจากผลงานวิจัยต่างๆ
4. ประเมินความเป็นไปได้ของขอบเขตการทำงานของ Application
5. ออกแบบระบบต่างๆ
6. คิดขั้นตอนวิธีที่จะใช้ใน Application
7. ลงมือสร้าง Application โดยใช้ข้อมูลที่ออกแบบไว้
8. ทดสอบและแก้ไขข้อผิดพลาดของระบบ
9. สรุปผลการทำงานและจัดทำรายงาน

### 1.5 ประโยชน์ที่คาดว่าจะได้รับ

1. เรียนรู้เกี่ยวกับการสร้าง Application ด้วย OpenGL ขั้นสูง
2. ฝึกความเข้าใจในการสร้าง Algorithm จากสมการคณิตศาสตร์ต่างๆ
3. สามารถนำไปประยุกต์ใช้สำหรับการประกอบอาชีพในอนาคต
4. สามารถนำ Application ไปพัฒนาต่อได้

# บทที่ 2
# ทฤษฎีและงานวิจัยที่เกี่ยวข้อง

## 2.1 OpenGL (Open Graphic Library)

OpenGL นั้นเป็น library ที่ทำให้สามารถสร้างโปรแกรมกราฟิกที่สามารถนำไปใช้ได้บนหลายๆ ระบบ ซึ่งสามารถทำได้ทั้งในรูปแบบสองมิติและสามมิติ โดย OpenGL จะมี function ให้ใช้มากกว่า 250 functions ซึ่งสามารถนำมาใช้สร้างฉากที่มีความซับซ้อนมากๆจากรูปทรงพื้นฐานได้ OpenGL นั้นได้ถูกสร้างขึ้นโดยบริษัท Silicon Graphics Inc. (SGI) ในปี 1992 และเป็นที่นิยมในการใช้งานด้านงานออกแบบโดยใช้คอมพิวเตอร์ช่วย การทำโลกเสมือน การจำลองทางวิทยาศาสตร์ การแสดงผลของข้อมูล และการจำลองการบินและยังมีการทำไปใช้ทำเกมอีกด้วยซึ่งในด้านการทำเกมนั้นได้มีคู่แข่งรายสำคัญอย่าง DirectX ของบริษัท Microsoft โดย OpenGL นั้นได้ถูกสร้างขึ้นโดยองค์กรที่ไม่หวังผลตอบแทนชื่อว่า consortium Khronos Group

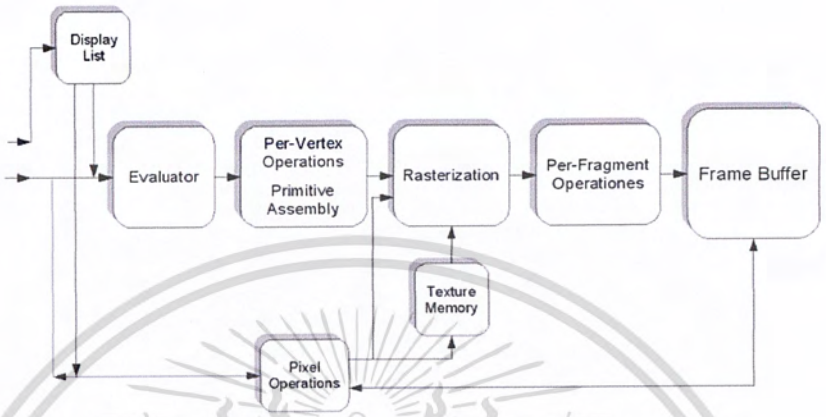
### 2.1.1 การออกแบบ

OpenGL นั้นถูกสร้างขึ้นมาด้วยเหตุผลหลักๆคือ

- เพื่อซ่อนความซับซ้อนของการใช้ 3D accelerator โดยทำให้รูปแบบการเรียกใช้นั้นเรียบง่ายและเป็นมาตรฐาน
- เพื่อซ่อนความแตกต่างทางด้านความสามารถต่างๆของแต่ละ platform โดยทำให้ใช้ OpenGL ทั้งหมดในการทำ implementation (ใช้การจำลองโดยใช้ software emulation หากจำเป็น)

OpenGL นั้นมีการดำเนินการแบบพื้นฐานเพื่อใช้ทำงานกับรูปแบบพื้นฐานต่างๆ เช่น จุด เส้น และ รูปร่างหรือรูปทรง และแปลงสิ่งเหล่านี้ให้เป็นข้อมูล pixel ซึ่งสามารถทำโดยใช้ graphic pipeline ซึ่งรู้จักกันในชื่อของ OpenGL state machine โดยคำสั่งของ OpenGL ส่วนใหญ่นั้นถ้าไม่เป็นการมอบหมายรูปร่างให้ graphic pipeline ประมวลผลก็เป็นการปรับปรุงค่าต่างๆและบอกวิธีที่จะประมวลผลรูปร่างต่างๆให้กับ graphic pipeline ก่อนที่จะมี OpenGL 2.0 นั้นแต่ละขั้นตอนของ pipeline ได้มีขั้นตอนแบบ

ตายตัวและสามารถปรับแต่งได้น้อยมาก ซึ่งใน OpenGL 2.0 นั้นได้มีขั้นตอนเพิ่มเติมมากขึ้นที่สามารถปรับแต่งได้อย่างสะดวกโดยใช้ GLSL




รูป 2.1 OpenGL graphic pipeline

2.1.2 Library ต่างๆที่เกี่ยวข้อง

ใน OpenGL นั้นมี library หลายแบบที่สามารถนำมาใช้เพื่อทำให้มีรูปแบบการใช้งานที่ OpenGL เดิมนั้นไม่มีให้ใช้ ซึ่งมีดังนี้

- GLU
- GLUT
- GLUI
- SDL
- FLTK
- GLEW
- GLEXT
- GLEE
- และอื่นๆ

โดยในที่นี้เราได้ใช้ GLUT ในการพัฒนาโปรแกรม

2.1.3 GLUT (OpenGL Utility Toolkit)

OpenGL Utility Toolkit (GLUT) เป็น library ของ OpenGL ซึ่งโดยหลักแล้วจะทำงานในระดับ system-I/O กับ Operating System (OS) หรือระบบปฏิบัติการณ์ของเครื่องที่ใช้ โดยจะมีคำสั่งที่ใช้กับการสร้างหน้าต่างต่างๆ เช่นการกำหนดชื่อและค่าต่างๆของหน้าต่าง การควบคุมหน้าต่าง การตรวจดูสถานะของ mouse และ keyboard และมีรูปแบบพื้นฐานใหม่เพิ่มเข้ามา (primitive) เช่น ลูกบาศก์ ทรงกลม และ กาน้ำ GLUT ยังมีความสามารถในการสร้างหน้าต่าง pop-up อีกด้วย แต่ก็ทำได้แค่ในวงที่จำกัด

GLUT นั้นถูกสร้างขึ้นโดย Mark J. Kilgard ซึ่งเป็นผู้แต่งหนังสือ OpenGL Programming for the X Window System และ The Cg Tutorial: The Definitive Guide to Programmable Real-Time Graphics และได้ทำงานอยู่ที่บริษัท Silicon Graphic Inc.

GLUT เล็งไปที่สองสิ่งคือ ทำให้สามารถสร้าง code ที่สามารถ port ไปใช้บน OS อื่นๆ ได้ และทำให้การเรียนรู้การใช้ OpenGL นั้นทำได้ง่าย ซึ่งการเริ่มต้นเขียนโปรแกรมด้วย GLUT นั้นไม่ต้องการความรู้ทางด้านโปรแกรมมากและไม่ต้องการความรู้เกี่ยวกับ API ของสร้างหน้าต่างของแต่ละ OS

2.1.3.1 ข้อจำกัดของ GLUT

ในบางกรณีนั้นผู้พัฒนาจะไม่สามารถทำสิ่งที่ต้องการได้อย่างง่ายดายใน GLUT เนื่องจากผลของการตัดสินใจขณะทำการสร้าง GLUT ทำให้เกิดข้อจำกัดนี้ ซึ่งมีผู้ที่พยายามทำการแก้ไขข้อจำกัดนี้ของ GLUT ไว้ โดยข้อจำกัดบางอย่างของ GLUT มีดังนี้

- GLUT library นั้นต้องการการเรียกใช้ glutMainLoop() ซึ่งเป็น function ที่ไม่มีการคืนค่ากลับจึงทำให้เป็นการยากที่จะสามารถนำ GLUT ไปใช้ใน library ซึ่งต้องการการควบคุม event loop ของตัวเอง
- glutMainLoop() ไม่มีการคืนค่ากลับซึ่งหมายความว่าโปรแกรมจะไม่สามารถออกจาก event loop นี้ได้

- GLUT ทำลาย process ก็ต่อเมื่อหน้าต่างนั้นได้ถูกปิด ในบางโปรแกรมอาจจะไม่เป็นที่ต้องการเนื่องจากอาจจะต้องการ callback เพิ่มเติม

## 2.2 การจำลองคลื่น (Wave Simulation)

คลื่นที่อยู่ในหมวดนี้นั้นจะเป็นคลื่นที่มีลักษณะราบเรียบและไม่มีการกฎการณ์ที่ซับซ้อนเช่น การคว่ำตัวลงของคลื่น ซึ่งจะเป็นผลให้การจำลองคลื่นในลักษณะนี้ประมวลผลได้รวดเร็วและเหมาะกับงานที่ต้องการความรวดเร็วหรือโปรแกรมที่มีการตอบสนองกับผู้ใช้ โดยในงานนี้จะเป็นคลื่นมหาสมุทรที่มีความลึกซึ่งจะไม่ต้องคำนวณค่าที่เกี่ยวข้องกับความลึกและคลื่นจะเป็นลักษณะแบบราบเรียบ ส่งผลให้การประมวลผลมีความซับซ้อนน้อยและมีความเหมาะสมกับโปรแกรมเช่น เกม ที่ต้องการการแสดงผลแบบ real-time

### 2.2.1 Ocean Wave Simulation Overview

การจำลองคลื่นมหาสมุทรนั้นได้เป็นหัวข้อสำคัญในวงการของ computer graphic และ virtual reality มาตลอดเวลา จากการที่คลื่นมหาสมุทรนั้นเป็นปรากฏการณ์ธรรมชาติที่มีความซับซ้อนซึ่งยากที่สามารถที่จะจำลองโดยใช้วิธีแบบทั่วๆไปได้ จนถึงบัดนี้วิธีการที่จะจำลองนั้นมีอยู่ 3 ชนิดหลักๆ ชนิดแรกคือวิธีที่ใช้ parametric function ซึ่งวิธีนี้นั้นจะสามารถใช้จำลองการเคลื่อนตัวของคลื่นได้เป็นอย่างดี แต่ในกรณีของพื้นผิวน้ำที่ถูกกระตุ้นจะไม่สามารถให้ผลลัพธ์ได้ดีนัก และในอีกชนิดหนึ่งจะเป็นการใช้โมเดลกายภาพโดยใช้สมการการเคลื่อนที่ของของไหล โดยจะให้ผลลัพธ์ได้อย่างยอดเยี่ยมและสมจริงแต่การประมวลผลนั้นจะทำได้ช้ามาก และในชนิดสุดท้ายซึ่งที่จะใช้ในงานนี้ซึ่งเกี่ยวข้องกับ Random Ocean Wave Theory โดยใช้ spectrum ของความถี่และ spectrum แบบมีทิศทางของคลื่นมหาสมุทร ซึ่งเรียกว่า ocean wave spectrum method ในวิธีนี้จะให้ความสมจริงสำหรับการจำลองคลื่นในมหาสมุทรที่มีระดับน้ำลึกและการประมวลผลจะไม่ช้าเมื่อเทียบกับวิธีที่กล่าวมาข้างต้น

ในระดับน้ำที่ลึกของมหาสมุทรคลื่นจะเคลื่อนที่และมีการรบกวนเกิดขึ้นซึ่งตามหลักแล้วจะเกิดจากอิทธิพลของลมซึ่งจะไม่ถูกสนใจในงานที่ผ่านๆมาของการจำลองคลื่นทะเล เป้าหมายหลักของงานนี้คือจะทำให้คลื่นที่จำลองจะเปลี่ยนไปเมื่อมีการเปลี่ยนแปลงของลม ใน

การที่เราจะจำลองนั้นเราจะต้องสร้างโมเดลทั้งลมและคลื่น โดยโมเดลที่มีอยู่แล้วจะขึ้นอยู่กับ spectrum ของมหาสมุทรที่สามารถจำลองความผันผวนของคลื่นภายใต้ความเร็วและทิศทางของลมที่คงตัว

การเปลี่ยนแปลงของลมนั้นเป็นขั้นตอนที่ซับซ้อนโดยที่การสร้างโมเดลที่แทนลมที่ตรงกับความเป็นจริงนั้นแทบจะเป็นไปไม่ได้ แต่ก็จำไว้ว่าเป้าหมายของเราคือเล็งไปที่ความสมจริงของทางด้านการมองเห็นไม่ใช่ความสมจริงทางด้านกายภาพซึ่งต้องการใช้ในการศึกษาในวิทยาศาสตร์ ดังนั้นในงานนี้จึงได้นำเสนอเกี่ยวกับโมเดลของลมในแบบที่เรียบง่ายซึ่งแทนการเปลี่ยนแปลงความเร็วและทิศทางของลมและได้นำเข้าไปใช้ร่วมกับโมเดลคลื่น

### 2.2.2 การสร้างคลื่นมหาสมุทร (Ocean Wave Generation)

#### 2.2.2.1 แบบจำลองคลื่นมหาสมุทร (Ocean Wave Model)

มีแบบจำลองหลายแบบซึ่งได้ถูกนำเสนอสำหรับคลื่นมหาสมุทรเช่น แบบจำลองของ Longuet-Higgins ซึ่งได้อธิบายถึงความปั่นป่วนของน้ำบนจุดที่ตายตัว โดยการใช้ cosine wave superimposition ดังต่อไปนี้

$$\eta(x,y,t) = \sum_{i=1}^{\infty}\sum_{j=1}^{\infty} a_{ij}\cos(k_i(x\cos(\theta_p+\theta_j)+y\sin(\theta_p+\theta_j)) - \omega_i t - \varepsilon_{ij}) \quad (2.1)$$

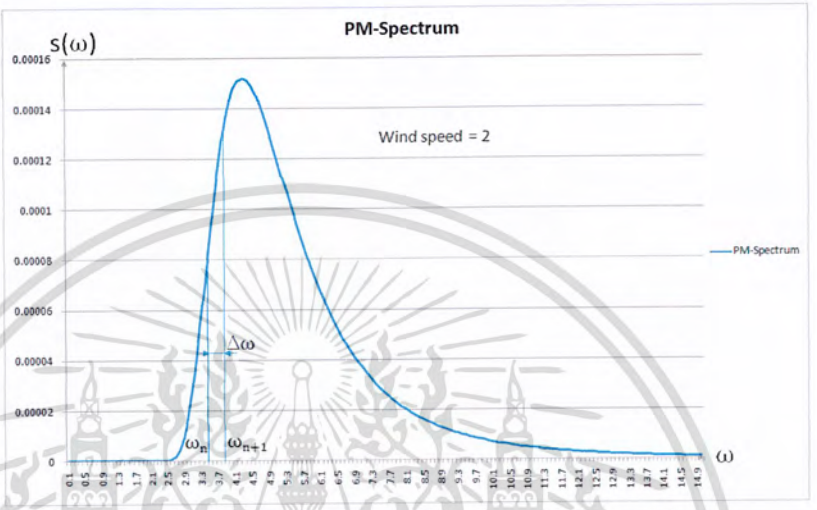
โดยที่ $a_{ij}$ $\omega_{ij}$ $k_i$ $\varepsilon_{ij}$ แทนค่า amplitude ความถี่ของวงกลม หมายเลขของคลื่นและเฟสเริ่มต้นของคลื่น $\theta_p$ แทนทิศทางลม $\theta_j$ แทนทิศทางที่ต่างกันของคลื่นและลม (x, y) คือจุดบนพื้นผิวของคลื่นและ $\eta(x,y,t)$ แทนความสูงของจุด

จากผลการสังเกตได้แสดงให้เห็นว่าคลื่นมหาสมุทรเป็นกระบวนการสุ่มแบบปกติ ซึ่ง spectrum ของความถี่ของคลื่นนั้นอธิบายถึงการกระจายตัวกันของพลังงานของคลื่น เราสามารถหาได้ว่าพื้นที่ภายใต้ส่วนโค้งของคลื่นจะเท่ากับพลังงานของคลื่นทั้งหมด โดยที่ค่าเฉลี่ยของพลังงานของส่วนประกอบของคลื่นทั้งหมดที่มีความถี่ในช่วง $\omega \sim (\omega + d\omega)$ สามารถกำหนดได้ว่า

(2.2)

$$S(\omega) = \frac{1}{\Delta\omega} \sum_{\omega}^{\omega+\Delta\omega} \frac{1}{2} a_n^2$$

โดยที่ $S(\omega)$ หมายถึง frequency spectrum function ของคลื่นมหาสมุทร




รูป 2.2 กราฟแสดง spectrum ของการกระจายตัวของพลังงานในคลื่น

ที่ในความเป็นจริงแล้วการกระจายตัวกันของพลังงานคลื่นไม่ได้มีความสัมพันธ์แค่กับความถี่แต่ยังมีความสัมพันธ์กับทิศทางของคลื่นด้วย ซึ่งความสัมพันธ์นี้สามารถอธิบายได้โดย spectrum ของคลื่นแบบมีทิศทาง โดยที่ function สามารถนำมาแสดงได้เป็นลักษณะดังนี้ (2.3)

$$S(\omega,\theta) = S(\omega) \cdot G(\omega,\theta)$$

โดย $G(\omega,\theta)$ แทน function ของการกระจายตัวแบบมีทิศทาง $\theta$ อาจเป็นค่าตั้งแต่ $0 \sim 2\pi$ ในทางทฤษฎี แต่ขอบเขตจริงๆก็คือ $-\pi/2 \sim \pi/2$ ซึ่งมาจากการที่การกระจายตัวของพลังงานคลื่นนั้นแท้จริงแล้วได้ถูกแบ่งออกทั้งสองทิศทางจากทิศทางของลมหลักตั้งแต่ $-\pi/2$ ถึง $\pi/2$ และในกรณีเดียวกันเราได้ให้พลังงานเฉลี่ยของทุกส่วนของคลื่นที่มีความถี่ $\omega \sim (\omega + d\omega)$ และทิศทางใน $-\pi \sim \pi$ ดังนี้

$$S(\omega,\theta) = \frac{1}{\Delta\omega \cdot \Delta\theta} \sum_{\omega}^{\omega+d\omega} \sum_{\theta}^{\theta+d\theta} \frac{1}{2} a_n^2 \qquad (2.4)$$

ปัญหาสำคัญก็คือการที่จะกำหนดค่าของ amplitude ความถี่ เฟส และทิศทาง ซึ่งค่าเหล่านี้ควรจะอยู่ในภาคของทฤษฎีคลื่นมหาสมุทร

### 2.2.2.2 การคำนวณค่าตัวแปรต่างๆในแบบจำลองคลื่น (Calculation of Parameters in Wave Model)

#### 2.2.2.2.1 Amplitude

ในที่นี้เราได้ใช้ spectrum ความถี่ของ Pierson-Moscowitz

$$S(\omega) = \frac{\alpha g^2}{\omega^5} \exp\left[-\beta\left(\frac{g}{U\omega}\right)^4\right] \tag{2.5}$$

ซึ่ง $\alpha = 8.1 \times 10^{-3}$ , $\beta = 0.74$ g คือค่าคงตัวของแรงโน้มถ่วง U คือค่าของความเร็วลมที่ความสูง 19.5 เมตรเหนือระดับน้ำทะเล

เราได้ใช้สมการของ spectrum แบบมีทิศทางดังนี้

$$G(\omega, \theta) = \frac{8}{3\pi}\cos^4(\theta) \tag{2.6}$$

และ amplitude จาก

$$a_{ij} = \sqrt{2S(\omega_i)G(\omega_i, \theta_i)\Delta\omega\Delta\theta} \tag{2.7}$$

โดย $\omega_i$ คือตัวแทนของความถี่ $\theta_i$ คือตัวแทนองศาของทิศทาง $\Delta\omega$ คือช่วงความถี่ $\Delta\theta$ คือช่วงองศาของทิศทาง

#### 2.2.2.2.2 ช่วงของความถี่และหมายเลขที่ถูกแบ่งส่วน

ก่อนอื่นเราจะต้องทำให้แน่ใจว่าค่าในช่วงความถี่ที่เราจะใช้นั้นเหมาะสมโดยดูจากกราฟการกระจายตัวของพลังงานซึ่งค่าในส่วนที่มีความถี่น้อยมากหรือสูงมากโดยการไม่สนใจในส่วนที่มีพลังงานต่ำมาก โดยเราได้ค่าตัวแทนความถี่ $\omega_1, \omega_2, \ldots, \omega_n$ ซึ่งมีค่าของช่วง $\Delta\omega$

$$\Delta\omega = \frac{\omega_{max} - \omega_{min}}{n} \quad (2.8)$$

โดยค่า n หมายถึงจำนวนของเลขทั้งหมดในช่วงของความถี่

$$\omega_i = \omega_{min} + (i-1)\Delta\omega + \frac{\Delta\omega}{2} \quad (1 \le i \le n) \quad (2.9)$$

**2.2.2.2.3 จำนวนคลื่น**

สำหรับคลื่นมหาสมุทรมีสมการความสัมพันธ์ของการแพร่กระจายระหว่างความถี่และจำนวนคลื่น ในกรณีของน้ำลึกซึ่งก้นทะเลสามารถที่จะไม่นำมาคิดได้โดยความสัมพันธ์ได้ดังสมการ

$$\omega^2 = k \cdot g \quad (2.10)$$

**2.2.2.2.4 ตัวแทนขององศาทิศทาง**

จากการกระจายตัวของพลังงานคลื่นขอบเขตของ $\theta_i$ คือ $\left(\theta_P - \frac{\pi}{2}\right) \sim \left(\theta_P + \frac{\pi}{2}\right)$ เราได้ตัวแทนองศาทิศทางที่มีช่วง $\Delta\theta$

$$\Delta\theta = \frac{\pi}{m} \quad (2.11)$$

$$\theta_j = -\frac{\pi}{2} + \Delta\theta(j-1) \quad (1 \le j \le m) \quad (2.12)$$

เราได้แยกสถานะของคลื่นเป็นหลายๆระดับขึ้นอยู่กับความเร็วลมตามการศึกษาเกี่ยวกับมหาสมุทร ตามตารางซึ่งแสดงให้เห็นถึงขอบเขตความถี่ ช่วงของความถี่ซึ่งจะใช้ในการจำลอง

| Parameter / level | Wind speed | Frequency range | Frequency interval | Divide number |
|---|---|---|---|---|
| Gentle wave | <5.5 | 1.2~6.0 | 0.4 | 12 |
| Small wave | 0.55~8.0 | 0.6~3.0 | 0.2 | 12 |
| Middle wave | 8.0~11.5 | 0.4~2.4 | | 10 |
| High wave | 11.5~14.0 | 0.4~2.0 | | 8 |
| Large wave | 14.0~17.0 | 0.3~1.4 | 0.1 | 11 |
| Crazy wave | >17.0 | 0.2~1.2 | | 10 |

ตาราง 2.1 แสดงการปรับค่าในการจำลองคลื่นแบบต่างๆ

### 2.2.2.3 แบบจำลองลม (Wind Model)

การจำลองคลื่นมหาสมุทรโดยให้ลมมีความเร็วคงตัวเสมอนั้นจะไม่สามารถจำลองได้อย่างสมบูรณ์ เนื่องจากว่าไม่สามารถแก้ปัญหาว่าพื้นผิวของน้ำที่ถูกกระตุ้นโดยลมซึ่งเกิดการเปลี่ยนแปลงเมื่อความเร็วลมเปลี่ยนนั้นเกิดขึ้นได้อย่างไรและเป็นอย่างไร ซึ่งสามารถแก้ไขด้วยการสร้างแบบจำลองลมที่มีการเปลี่ยนแปลงความเร็วและนำมารวมกับวิธีที่กล่าวมาข้างต้น

#### 2.2.2.3.1 ฟังก์ชันของความเร็วลม (Wind Speed Function)

แบบจำลองลมของเราสามารถกล่าวได้ดังต่อไปนี้

$$U(t) = \begin{cases} U_1 & , t_0 \le t < t_1 \\ f(t) & , t_1 \le t \le t_2 \\ U_2 & , t_2 < t \le t_3 \end{cases} \qquad (2.13)$$

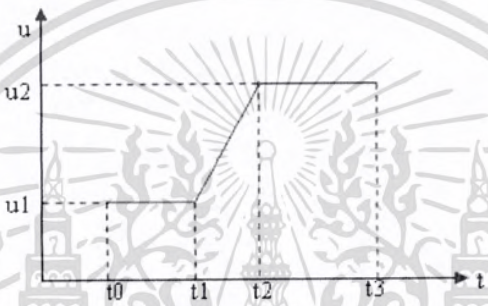
ในขณะที่ U(t) แสดงถึงความเร็วลมที่ 19.5 เมตรเหนือระดับน้ำทะเลที่เวลา t โดย $t_0$ $t_1$ $t_2$ $t_3$ แสดงถึงเวลาใดๆในการจำลองซึ่ง $t_0 < t_1 < t_2 < t_3$ ความเร็วลม $U_1$ ในเวลา $t_0$~$t_1$ และ $U_2$ ในช่วง $t_2$~$t_3$ ความเร็วลมได้เปลี่ยนใน

เวลา $t_1$~$t_2$ ซึ่งทำให้คลื่นในมหาสมุทรเปลี่ยนแปลง โดยเราได้กำหนด f(t) ใน เวลา $t_1$~$t_2$ เป็น abstract function ซึ่งผ่านคุณสมบัติ $f(t_1) = U_1$ และ $f(t_2) = U_2$

ขณะที่ $U_1 < U_2$ f(t) เป็นฟังก์ชันแบบ single-increasing และ

ขณะที่ $U_1 > U_2$ f(t) เป็นฟังก์ชันแบบ single-decreasing

รูปคร่าวๆของแบบจำลองสามารถดูได้จากในรูป 2.3




รูป 2.3 กราฟความเร็วลมตามช่วงเวลา

**2.2.2.3.2 กระบวนการเมื่อเกิดการเปลี่ยนแปลงของลม**

จากกระบวนการของการหาการกระจายตัวของพลังงานคลื่นนั้น สามารถแสดงได้แค่ในกรณีที่ความเร็วลมคงตัว ซึ่งลมนั้นเป็นตัวการที่ทำให้เกิดคลื่นมหาสมุทร และถ้าลมเกิดการเปลี่ยนแปลงนั้นจะทำให้การกระจายพลังงานคลื่นนั้นเกิดการเปลี่ยนแปลงด้วย นั่นหมายความว่าในเวลาที่แตกต่างกัน spectrum ของพลังงานจะแตกต่างกันด้วย

สมมุติว่า spectrum ของความถี่คือ $S(\omega)_1$ ขอบเขตของความถี่คือ $\omega_{1min}$~$\omega_{1max}$ และทิศทางลม $\theta_{1p}$ เมื่อความเร็วลมคือ $U_1$ และ spectrum ของความถี่คือ $S(\omega)_2$ ขอบเขตของความถี่จะเป็น $\omega_{2min}$~$\omega_{2max}$ และทิศทางลม $\theta_{2p}$ เมื่อความเร็วลมคือ $U_2$ ปัญหาก็คือเราจะทำอย่างไรเพื่อทำให้ขอบเขตของความถี่และทิศทางลมคงตัวในขณะที่เวลาการจำลองอยู่ในช่วง $t_1$~$t_2$

เมื่อความเร็วลมเปลี่ยนจาก $U_1$ ไปเป็น $U_2$ ขอบเขตของความถี่ควรจะเปลี่ยนจาก $\omega_{1min}$~$\omega_{1max}$ เป็น $\omega_{2min}$~$\omega_{2max}$ และทิศทางลมควรเปลี่ยนจาก $\theta_{1p}$ เป็น $\theta_{2p}$ เราได้ใช้วิธีการเปลี่ยนแปลงแบบเป็นเส้นตรงเพื่อที่จะได้ขอบเขตความถี่ระหว่างเวลา $\omega_{min}$~$\omega_{max}$ และทิศทางลม $\theta_p$ ของส่วนประกอบต่างๆของคลื่นซึ่งสัมพันธ์กับความเร็วลม f(t) ใน $t_1$~$t_2$ สามารถคำนวณได้โดยสูตรต่อไปนี้

$$\omega_{max} = \omega_{1max} - \frac{f(t)}{U_2 - U_1}|\omega_{2max} - \omega_{1max}| \tag{2.14}$$

$$\omega_{min} = \omega_{1min} - \frac{f(t)}{U_2 - U_1}|\omega_{1min} - \omega_{2min}| \tag{2.15}$$

$$\theta_p = \theta_{1p} + \frac{t}{t_3 - t_2}(\theta_{2p} - \theta_{1p}) \tag{2.16}$$

Spectrum ของความถี่คือ

$$S(\omega) = \frac{ag^2}{\omega^5}\exp\left[-\beta\left(\frac{g}{f(t)\omega}\right)^4\right] \tag{2.17}$$

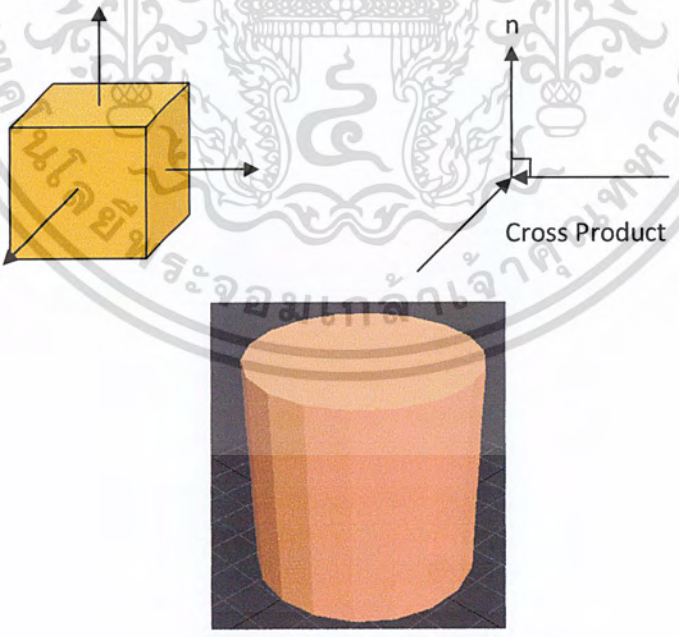
## 2.3 Normal Vector

Normal vector หรือ Surface normal หรือเรียกสั้นๆว่า normal เป็น vector ที่ตั้งฉากกับระนาบของพื้นผิว หาได้โดยการนำ vector ที่แสดงขอบของ Polygon มา Cross product กันจะได้ vector ที่ตั้งฉากกับระนาบของพื้นผิวนั้นๆ โดย Normal vector นั้นสามารถนำมาใช้ในงาน Computer Graphic ได้ในด้านของการคำนวณแสงที่ตกกระทบพื้นผิวว่าจะทำให้พื้นผิวมีความสว่างมากน้อยเพียงใด ขึ้นอยู่กับทิศทางของ Normal vector และทิศทางของแสง

ในงาน Computer Graphic นั้นจะมีการใช้ Normal vector ด้วยกันสองอย่างใหญ่ๆก็คือ Flat Shading และ Smooth Shading(Phong Shading)

### 2.3.1 Flat Shading

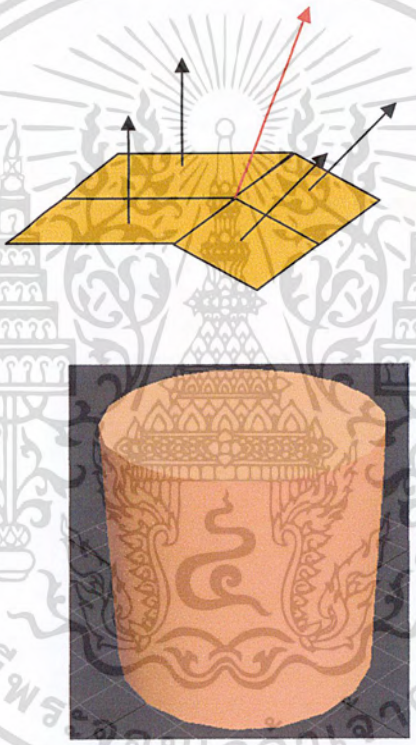
Flat Shading เป็นการคำนวณค่า Normal vector ของพื้นผิวโดยตรงโดยการนำ vector ที่แทนขอบของ Polygon มา Cross Product กันจะได้ Normal vector ของพื้นผิวนั้นๆออกมาโดยตรง เช่นดังในรูปตัวอย่าง




รูป 2.4 แสดงการใช้ Flat Shading

### 2.3.2 Smooth Shading

Smooth Shading จะเป็นการหา Normal vector เช่นเดียวกับ Flat Shading แต่วิธีการนั้นจะแตกต่างออกไปโดยที่ Smooth Shading นั้นจะเป็นการเลียนแบบพื้นผิวที่มีความโค้งและขั้นตอนการหา Normal vector นั้นจะทำโดยการหา Normal vector ของ Polygon ที่ใช้งานจุดยอดร่วมกันแล้วนำค่าที่ได้มาหาค่าเฉลี่ยเพื่อให้เกิดการเลียนแบบพื้นผิวที่มีความโค้งมนและดูสวยงามกว่าแบบ Flat Shading



รูป 2.5 แสดงการใช้ Smooth Shading

## 2.4 Multithreading

Multithreading เป็นวิธีการแบ่งการทำงานของ Process ที่ทำงานบนคอมพิวเตอร์ออกเป็นหลายๆส่วนที่สามารถทำงานได้พร้อมๆกันหรือทำงานแบบขนานกันได้ โดย CPU สมัยใหม่นั้นจะเป็นแบบ Multi-core และในแต่ละ core สามารถประมวลผลไปพร้อมๆกันได้ ซึ่งเพิ่มความสามารถการทำงานแบบ Multitasking ให้กับระบบปฏิบัติการ เทคนิคของการทำ Multithreading นั้นจะสามารถทำให้ Process หนึ่งๆสามารถทำงานได้บน CPU หลายๆ core พร้อมๆกันโดยที่ถ้าเป็นแบบ Single-threaded Process จะไม่สามารถทำงานบน core หลายๆ core พร้อมกันได้ และถ้าผู้พัฒนาได้แบ่งงานให้แต่ละ thread ทำได้เท่าๆกันจะสามารถเพิ่มความเร็วในการประมวลผลได้เป็นอย่างมากโดยอาศัยหลักการของการทำงานแบบขนาน

ถึงแม้ว่าการทำ Multithreading นั้นจะสามารถเพิ่มความเร็วการประมวลผลได้มาก แต่ก็มีข้อควรระวังหลายข้อ ซึ่งทำให้การพัฒนาระบบแบบ Multithreading นั้นทำได้ยากลำบาก ปัญหาที่พบได้มากที่สุดและต้องระวังที่สุดคือปัญหาที่เรียกว่า Racing Condition ซึ่งมีสาเหตุมาจากในแต่ละ thread นั้น OS ไม่ได้จัดสรรทรัพยากรให้เท่ากัน ซึ่งส่งผลให้การทำงานมีความเร็วแตกต่างกันไปในแต่ละ thread โดย Racing Condition นั้นจะทำให้ผลลัพธ์ที่เราต้องการนั้นออกมาผิด ตัวอย่างเช่น ถ้าในแต่ละ thread ใช้หน่วยความจำร่วมกันและต้องทำงานกับตัวแปรที่ใช้ร่วมกันอาจทำให้ผลลัพธ์ออกมาผิดพลาดได้ เช่น ในขณะที่ thread มาดึงค่าตัวแปร n ซึ่งมีค่าเท่ากับ 5 ไปคูณ 3 ประมวลผลอีก thread หนึ่งมาแก้ไขค่าตัวแปร n เป็น 7 แล้ว thread แรกประมวลผลเสร็จ เก็บผลลัพธ์ลงตัวแปรทำให้ผลลัพธ์กลายเป็น 15 ซึ่งผลลัพธ์ที่ถูกต้องควรจะเป็น 21 เพราะกาทำงานไม่ประสานกัน จึงต้องใช้วิธีการทำ Synchronization หรือการจัดจังหวะของ thread เพื่อให้เข้าใช้งานส่วนที่ใช้ร่วมกันได้อย่างสอดคล้อง เช่นในกรณีที่กล่าวมาข้างต้น thread ที่ทำหน้าที่ในการคูณจะต้องรอให้ thread ที่ทำการบวกค่าทำงานเสร็จก่อนถึงจะไปนำค่าจากตัวแปรมาประมวลผลจึงจะได้ค่าที่ถูกต้อง แต่การจัดจังหวะของ thread ให้รอกันนั้นถ้าทำไม่ดีอาจเกิดเหตุการณ์ที่เรียกว่า Deadlock ได้ ซึ่งหากเกิด Deadlock นั้นจะทำให้ Process ไม่สามารถทำงานต่อได้ เนื่องจาก thread ที่ทำหน้าที่ทำงานนั้นต่างก็รออีก thread หนึ่งให้เสร็จงาน แต่ในแต่ละ thread นั้นต้องการใช้ทรัพยากรสองส่วน ซึ่งแต่ละ thread นั้นได้จองทรัพยากรไป 1 ส่วน แล้วแต่อีกส่วนหนึ่งถูกอีก thread หนึ่งจองไว้เช่นกัน จึงต้องรอให้อีก thread หนึ่งปล่อยทรัพยากรให้

แต่ก็ไม่สามารถปล่อยได้เนื่องจากก็รอทรัพยากรอยู่เหมือนกัน ทำให้เกิดเหตุการณ์ที่ต่าง thread ต่างรอให้อีก thread ทำงานเสร็จ ซึ่งก็ไม่มีทางที่จะทำงานเสร็จได้เลย จึงเกิดการรอที่ไม่สิ้นสุดขึ้น ซึ่งเรียกว่า Deadlock ดังที่ได้กล่าวมา

### 2.4.1 ข้อดีของการทำ Multithreading

- หาก thread เกิด cache miss มากๆ thread อื่นๆจะสามารถใช้ทรัพยากรที่เหลืออยู่ของระบบมาประมวลผลได้ ซึ่งส่งผลให้ความเร็วในการประมวลผลโดยรวมเพิ่มขึ้น

- ถ้า thread ไม่สามารถทำงานได้เนื่องจากต้องรอผลลัพธ์ของอีก thread หนึ่ง การแบ่งงานให้ thread ทำงานแบบขนานทำให้ thread ที่ต้องรอไม่เกิดการรอนาน ทำให้ประมวลผลได้เร็วขึ้น

- ถ้าในหลายๆ thread ทำงานบนเซตของข้อมูลชุดเดียวกันจะสามารถใช้งาน cache ร่วมกันได้

### 2.4.2 ข้อเสียของการทำ Multithreading

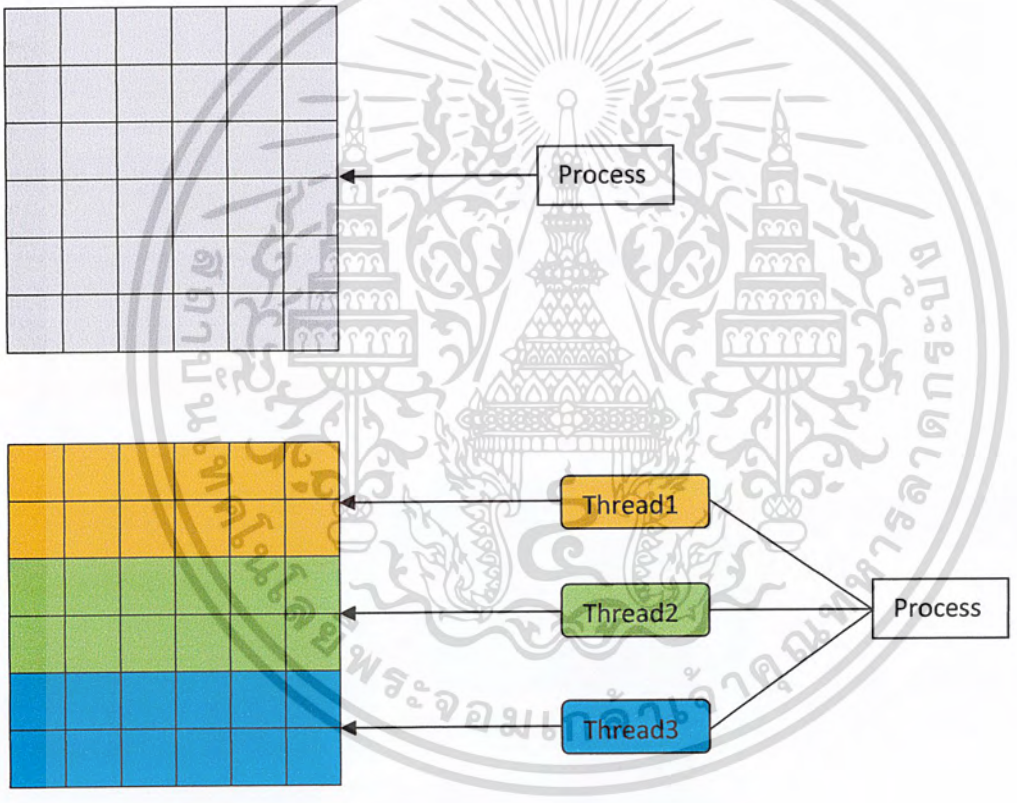
- หาก thread ใช้ทรัพยากรร่วมกันอาจมีการรบกวนกันได้เช่นการใช้ cache ร่วมกันจึงต้องมีการทำ Synchronization เพื่อหลีกเลี่ยงปัญหา

- ความเร็วในการประมวลผลในแต่ละ thread จะไม่เพิ่มขึ้นและอาจลดลงเนื่องจากต้องมีการทำ thread switching บ่อยและมีขั้นตอน pipeline เพิ่มขึ้นในระดับ Hardware

- อาจต้องมีการเปลี่ยนแปลงในส่วนของโปรแกรมและระบบปฏิบัติการมาก

### 2.4.3 ตัวอย่างการใช้ Multithreading

ตัวอย่างการทำ Multithreading โดยการประมวลผลสามารถทำได้พร้อมๆกันโดยไม่ขึ้นต่อกัน ในรูปบนจะเป็นการใช้ Process แบบ Single thread ซึ่งจะต้องทำงานกับข้อมูลทั้งหมดโดยทำงานได้แค่ครั้งละส่วน แต่ในรูปล่างจะเป็นการแยก Process เป็น 3 thread ซึ่งสามารถทำงานได้ขนานกันและประมวลผลได้ครั้งละ 3 ส่วนพร้อมๆกันโดยไม่ต้องรอให้ thread อื่นทำงานเสร็จ




รูป 2.6 เปรียบเทียบการประมวลผลแบบ Multithreading

## 2.5 Vertex Array

Vertex Array เป็นอีกวิธีหนึ่งในการแสดงผลของ OpenGL ซึ่งเรียกว่า non-immediate-mode จะต่างกับ immediate-mode ตรงที่ไม่ต้องเรียกใช้ฟังก์ชันทุกๆครั้งที่มีการกำหนดจุดยอด แต่จะเก็บจุดยอดไว้ใน array ซึ่งจะถูกนำไปเรียกใช้โดยการเรียกฟังก์ชันเพียงครั้งเดียว ตัวอย่างเช่นถ้าจะวาดกล่องสี่เหลี่ยมขึ้นมา 1 กล่อง ในแบบ immediate-mode จะต้องเรียกใช้ฟังก์ชันถึง 24 ครั้ง แต่แบบ vertex array จะเรียกใช้เพียงครั้งเดียวและการใช้ vertex array สามารถนำจุดยอดที่มีการใช้ร่วมกันในหลายๆ polygon มาใช้ซ้ำได้ เป็นการกดความซ้ำซ้อนของการใช้ยุดยอดร่วม โดยการนำ index array มาใช้บอกลำดับของการเชื่อมกันของ polygon

### 2.5.1 ฟังก์ชันที่ใช้ใน Vertex Array

glEnableClientState(GLenum target)

glDisableClientState(GLenum target)

ใช้ในการกำหนดว่าจะใช้หรือหยุดใช้ array แบบไหนโดยค่า target สามารถเป็นได้ดังนี้

- GL_VERTEX_ARRAY
- GL_NORMAL_ARRAY
- GL_COLOR_ARRAY
- GL_INDEX_ARRAY
- GL_TEXTURE_COORD_ARRAY
- GL_EDGE_FLAG_ARRAY

glVertexPointer()

glNormalPointer()

glIndexPointer()

glColorPointer()

glTexCoordPointer()

glEdgeFlagPointer()

ฟังก์ชันทั้ง 6 ที่กล่าวมานี้เป็นฟังก์ชันที่ใช้กำหนดตำแหน่งของ pointer ให้กับ OpenGL เพื่อให้สามารถไปดึงข้อมูลมาประมวลผลได้

glDrawArrays()

ฟังก์ชันนี้จะไม่สามารถเลือกให้กระโดดไปตามจุดใน vertex array ได้แต่เป็นการวาดตาม array ตั้งแต่หัวจนถึงท้ายไปตามลำดับ ซึ่งจะไม่สามารถใช้ index array ช่วยลดการเก็บข้อมูลจุดยอดที่ใช้ซ้ำได้

glDrawElements()

glDrawRangeElements()

ฟังก์ชันนี้จะต่างจาก glDrawArrays() ตรงที่สามารถกระโดดไปใช้จุดต่างๆใน vertex array ได้ตามลำดับที่กำหนดไว้ใน index array ทำให้ลดการกำหนดจุดยอดซ้ำซ้อนได้ และยังมีฟังก์ชันชื่อglDrawRangeElements() ที่จะต่างออกไปอีกตรงที่สามารถกำหนดจุดเริ่มต้นหรือจุดจบของส่วนที่ต้องการวาดได้

### 2.5.2 การใช้งาน Vertex Array

กำหนดตัวแปรที่ใช้เก็บค่าของจุดยอดเป็นอันดับแรก

```
GLfloat vertices[] = {…};          // จุดยอด 24 จุดของกล่องสี่เหลี่ยม
```

เปิดการใช้งาน Vertex Array โดยใช้คำสั่ง

```
glEnableClientState(GL_VERTEX_ARRAY);
```

กำหนด Pointer ของ Vertex Array ให้กับ OpenGL โดยค่าที่รับจะเป็นจำนวนข้อมูลต่อจุดยอดหนึ่งจุด ชนิดของจุดยอด การกระโดดข้ามตำแหน่งใน Array และตัวแปรที่เก็บจุดยอดไว้ตามลำดับ

```
glVertexPointer(3, GL_FLOAT, 0, vertices);
```

ใช้คำสั่งหลังจากที่ได้กำหนดค่าต่างๆเรียบร้อยแล้วเพื่อทำการวาดลงบนฉาก

```
glDrawArray(GL_QUADS, 0, 24);
```

ปิดการใช้งาน Vertex Array หลังจากใช้เสร็จ

```
glDisableClientState(GL_VERTEX_ARRAY);
```

## 2.6 Vertex Buffer Object (VBO)

Vertex Buffer Object (VBO) คือส่วนเสริมของ OpenGL ที่สร้างขึ้นมาเพื่ออัพโหลดข้อมูลต่างๆที่ใช้ในการแสดงผลภาพไปยังหน่วยแสดงภาพ (video device) โดยตรงเพื่อใช้ในการแสดงผลภาพแบบ non-immediate-mode ซึ่ง VBO นั้นสามารถทำให้ความเร็วในการแสดงผลเพิ่มขึ้นเมื่อเทียบกับแบบ immediate-mode เพราะ VBO จะเก็บข้อมูลเช่น จุดยอด สี หรือนอมอลเวคเตอร์ไว้ในหน่วยความจำของหน่วยแสดงผล (video memory) และยังไม่มีการเรียกฟังก์ชันซ้ำซ้อนแบบ immediate-mode ทำให้สามารถแสดงผลได้โดยตรงจากหน่วยแสดงผลและทำได้รวดเร็วกว่า

### 2.6.1 ฟังก์ชันที่ใช้ใน VBO

void glGenBuffersARB(GLsizei n, GLuint *ids)

ใช้ในการสร้างบัฟเฟอร์ออบเจคต์ (Buffer object) โดยรับค่า 2 ค่าคือ จำนวนบัฟเฟอร์ที่ต้องการสร้างและตัวแปรที่ใช้เก็บ id ของบัฟเฟอร์ตามลำดับ

void glBindBufferARB(GLenum target, GLuint id)

ใช้ในการบอกว่าบัฟเฟอร์นั้นเป็นชนิดไหนและบอกว่าขณะนี้กำลังทำงานกับบัฟเฟอร์ไหนโดยรับค่า 2 ค่าคือ ชนิดของบัฟเฟอร์และ id ของบัฟเฟอร์ ตามลำดับ และชนิดของบัฟเฟอร์นั้นจะมีได้ 2 แบบคือ GL_ARRAY_BUFFER_ARB และ GL_ELEMENT_ARRAY_BUFFER_ARB ซึ่งจะใช้เก็บค่าต่างๆของจุดยอด (vertex attributes) และเก็บดัชนี (indices) ตามลำดับ

void glBufferDataARB(GLenum target, GLsizei size, const void *data, GLenum usage)

ใช้ในการคัดลอกข้อมูลจากหน่วยความจำลงในบัฟเฟอร์ ค่าที่รับคือชนิดของบัฟเฟอร์ ขนาดของข้อมูล pointer ที่ชี้ตำแหน่งข้อมูล และข้อมูลการใช้ (static, dynamic, stream และ copy หรือ draw) ตามลำดับ

void glBufferSubDataARB(GLenum target, GLuint offset, GLsizei size, void *data)

ฟังก์ชันนี้จะทำงานคล้ายกับ glBufferDataARB() ต่างกันเพียงแค่การคัดลอกข้อมูลไปแทนที่นั้นจะคัดลอกเพียงบางส่วนเท่านั้น เริ่มต้นจากตำแหน่ง offset จนถึง offset+size โดยนำข้อมูลจาก *data ไปแทนที่ตามค่า size

void glDeleteBuffersARB(GLsizei n, GLuint *ids)

เป็นฟังก์ชันที่ใช้ลบบัฟเฟอร์ที่ไม่ใช้แล้วออกเพื่อคืนหน่วยความจำให้กับระบบโดยรับค่า 2 ค่าคือจำนวนที่จะลบและ id ของบัฟเฟอร์ที่จะทำการลบ

### 2.6.2 การใช้งาน VBO

สร้าง VBO ขึ้นมาเพื่อใช้เก็บค่าจุดยอดลงในหน่วยความจำของหน่วยแสดงผล

```
GLuint vboid = 0;
glGenBuffersARB(1, &vboid);
```

กำหนดชนิดของ VBO และ Id ของ VBO ที่ต้องการจะใช้งาน

```
glBindBufferARB(GL_ARRAY_BUFFER_ARB, vboid);
```

นำข้อมูลจาก Vertex Array เก็บลงในบัฟเฟอร์ (สมมุติว่ามี array ที่เก็บจุดยอดอยู่แล้วชื่อ vertexarray) ขนาดของ array หาได้โดยการใช้คำสั่ง sizeof(ชื่อตัวแปร) กำหนดตัวแปร และชนิดการใช้งานของบัฟเฟอร์

```
glBufferDataARB(GL_ARRAY_BUFFER_ARB, sizeof(vertexarray), vertexarray,
GL_STATIC_DRAW_ARB);
```

เปิดการใช้งาน Vertex Array

```
glEnableClientState(GL_VERTEX_ARRAY);
```

กำหนด pointer แต่ใน VBO จะต่างจาก Vertex Array ตรงที่ค่าสุดท้ายจะเป็น offset ในบัฟเฟอร์

```
glVertexPointer(3, GL_FLOAT, 0, 0);
```

ปิดการใช้งาน Vertex Array

```
glDisableClientState(GL_VERTEX_ARRAY);
```

ปิดการใช้งาน VBO โดยให้ค่าสุดท้ายเป็น 0

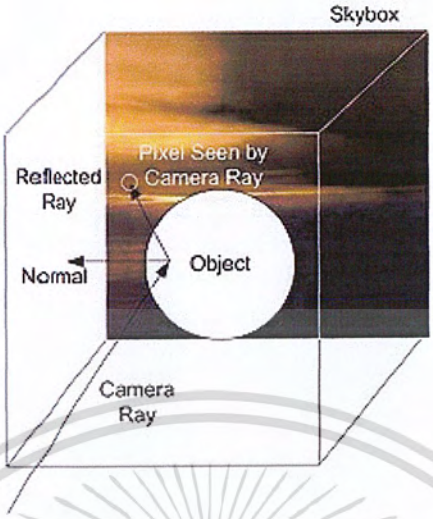
```
glBindBufferARB(GL_ARRAY_BUFFER_ARB, 0);
```

## 2.7 Environment Mapping

ใน Computer Graphic นั้น Environment Mapping หรือ Reflection Mapping เป็นเทคนิคการจำลองพื้นผิวที่มีความมันเงาและสะท้อนแสงโดยการใช้พื้นผิวที่ได้ทำการประมวลผลไว้ล่วงหน้าซึ่งก็คือภาพสิ่งแวดล้อมที่ล้อมรอบวัตถุนั้นๆ

เทคนิคของการเก็บภาพพื้นผิวที่ล้อมรอบนั้นมีหลายวิธี โดยวิธีแรกเรียกว่า Sphere Mapping ซึ่งเป็นการใช้ไฟล์พื้นผิวหรือรูปพื้นผิวเพียงพื้นผิวเดียวเช่นเดียวกับที่สะท้อนมาจากวัตถุทรงกลม แต่วิธีนี้ให้ผลลัพธ์ไม่ดีนักและสิ้นเปลืองทรัพยากรและได้ถูกแทนด้วยวิธี Cube Mapping ซึ่งให้ผลดีกว่าและประมวลผลได้เร็ว โดยในวิธีนี้จะเป็นการใช้พื้นผิวทั้งหมด 6 ด้านที่ถูกฉายจากด้านต่างๆของกล่อง ซึ่งในงานนี้ได้นำเทคนิค Cube Mapping มาใช้จำลองการสะท้อน

การใช้เทคนิคนี้เป็นวิธีที่มีประสิทธิภาพและสามารถทำได้รวดเร็วกว่าวิธีดั้งเดิมซึ่งใช้ Ray Tracing ในการหาการสะท้อนจากการดูการตกกระทบของลำแสงตามจุดต่างๆ โดยการใช้วิธีนี้จะได้ผลลัพธ์ที่ใกล้เคียงกับการใช้ Ray Tracing แต่จะใช้การประมวลผลน้อยกว่ามากเพราะค่าความสว่างจะได้มาจากการคำนวณองศาการตกกระทบและการสะท้อนตามด้วยการหาจุดของพื้นผิวที่เกี่ยวข้อง ต่างจาก Ray Tracing ตรงไม่ต้องตรวจจับลำแสงที่ตกกระทบกับพื้นผิวและคำนวณค่าความสว่างของลำแสงทำให้ GPU (Graphic Processing Unit) ทำงานน้อยลง แต่วิธีนี้เพียงแค่จำลองการสะท้อนเท่านั้นและจะไม่มีการสะท้อนของตัววัตถุเองเช่นในกรณีที่วัตถุมีลักษณะเว้า ถึงอย่างนั้นวิธีนี้ก็เป็นวิธีที่สามารถคำนวณและแสดงผลการสะท้อนที่ทำได้รวดเร็วที่สุดและเหมาะกับการนำมาใช้ในงานที่ต้องการความเร็วเช่นเกมเป็นต้น




รูป 2.7 กระบวนการของขั้นตอนการทำ Cube Mapping

รูป 2.8 เมื่อนำ Cube Mapping ไปใช้กับวัตถุที่มีความมันเงา

## 2.8 กฎเกี่ยวกับแรงลอยตัวของ Archimedes (Archimedes' Principle of Buoyancy)

ในหลักของฟิสิกส์ แรงลอยตัวคือแรงกระทำต่อวัตถุในทิศทางขึ้นด้านบนซึ่งเป็นแรงที่เกิดจากของไหลที่ต้านทานน้ำหนักของวัตถุ ในของไหลนั้นโดยปกติแล้วความดันจะเพิ่มขึ้นตามความลึกที่เพิ่มขึ้นซึ่งเป็นผลมาจากน้ำหนักของของไหลที่อยู่ด้านบนกระทำต่อด้านล่าง จึงทำให้วัตถุที่อยู่ในระดับความลึกที่มากกว่าได้รับแรงดันที่มากกว่าวัตถุที่ลอยอยู่ด้านบน จากการที่ในระดับความลึกแตกต่างกันมีแรงดันไม่เท่ากันจึงทำให้วัตถุซึ่งมีความหนาแน่นแตกต่างกันจะลอยหรือจมอยู่ในระดับความลึกที่แตกต่างกัน ซึ่งแรงที่กระทำต่อวัตถุนั้นจะเท่ากับน้ำหนักของของไหลที่ถูกแทนที่ด้วยวัตถุนั้นๆจึงจะเห็นได้ว่าวัตถุที่มีความหนาแน่นมากกว่าของไหลจะจมลง หรือถ้าวัตถุมีความหนาแน่นน้อยกว่าของไหลหรือมีรูปร่างพิเศษเช่น เรือ จะสามารถลอยอยู่บนพื้นผิวของของไหลได้ ในหลักของฟิสิกส์ของไหลได้กล่าวไว้ว่า แรงลอยตัวที่กระทำต่อวัตถุจะเท่ากับน้ำหนักของของไหลที่มีปริมาตรเท่ากับปริมาตรของวัตถุที่มาแทนที่ซึ่งตรงกับคำพูดที่ Archimedes ได้กล่าวไว้ว่า

> "Any object, wholly or partially immersed in a fluid, is buoyed up by a force equal to the weight of the fluid displaced by the object."
>
> — *Archimedes of Syracuse*

แต่หลักการของ Archimedes นั้นจะไม่คำนึงถึงแรงตึงผิวของของไหล หลักการของ Archimedes นั้นจะสามารถเขียนเป็นสมการได้ดังนี้

แรงที่เกิดจากน้ำหนักของวัตถุ

$$F_o = m_o g \tag{2.18}$$

โดยที่ $m_o$ = มวลของวัตถุ (kg)

$g$ = ความเร่งเนื่องจากแรงดึงดูด (9.8 m/s)

โดย m สามารถหาได้จากปริมาตรของวัตถุคูณกับความหนาแน่นของวัตถุดังนี้

$$m_o = \rho_o V \tag{2.19}$$

โดยที่ $\rho_o$ = ความหนาแน่นของวัตถุ

V = ปริมาตรของวัตถุ

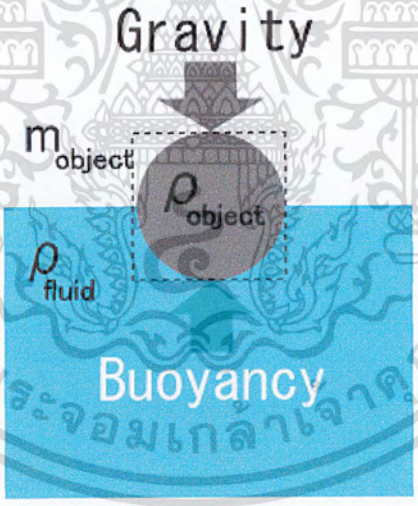
แรงลอยตัวที่กระทำต่อวัตถุที่เป็นผลมาจากของไหลสามารถหาได้ดังนี้

$$F_b = \rho_f V_{disp} g \tag{2.20}$$

โดยที่ $\rho_f$ = ความหนาแน่นของของไหล

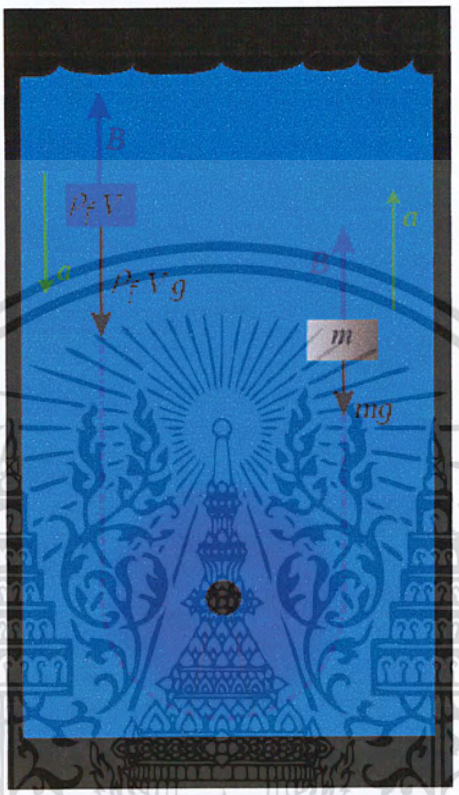
$V_{disp}$ = ปริมาตรของของไหลที่ถูกแทนที่โดยวัตถุ

g = ความเร่งเนื่องจากแรงดึงดูด




รูป 2.9 แสดงแรงที่กระทำต่อวัตถุที่เกิดจากของไหล

จากในรูป 2.10 กล่องทางซ้ายแทนของไหลที่ถูกแทนที่โดยวัตถุ กล่องทางขวาแทนวัตถุที่อยู่ในของไหล และวงกลมสีฟ้าด้านล่างคือรอกไร้แรงเสียดทานที่สมมุติขึ้นมาเพื่อให้อธิบายได้ดีขึ้น




รูป 2.10 แรงที่กระทำต่อวัตถุในของไหล

เราสามารถหาความเร่งของวัตถุที่ลอยหรือจมได้จาก

$$a = g \cdot (m_o - m_f)/(m_o + m_f) \tag{2.21}$$

โดยที่ g = ความเร่งเนื่องจากแรงดึงดูด

$m_o$ = มวลของวัตถุ

$m_f$ = มวลของของไหลที่ถูกแทนที่

เราสามารถหา $m_f$ ได้จาก

$$m_f = \rho_f V_{disp} \tag{2.22}$$

โดยที่ $\rho_f$ = ความหนาแน่นของของไหล

$V_{disp}$ = ปริมาตรของของไหลที่ถูกแทนที่โดยวัตถุ

# บทที่ 3

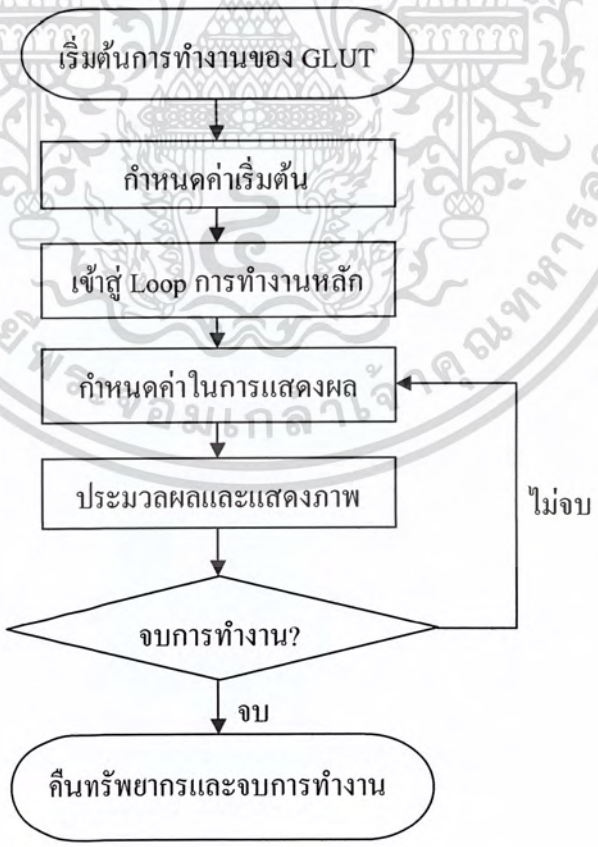
# ขั้นตอนการออกแบบและพัฒนาซอฟต์แวร์

จากในบทที่แล้วที่ได้กล่าวถึงทฤษฎีต่างๆที่ใช้ในการจำลองคลื่นและทฤษฎีต่างๆที่นำมาใช้ประกอบกันเพื่อให้การจำลองมีความสวยงามและรวดเร็วไปแล้ว ในบทนี้จำเป็นการนำเสนอวิธีการนำทฤษฎีต่างๆมาใช้ในการจำลองคลื่นแบบมีปฏิสัมพันธ์ให้ได้ออกมาเป็นผลลัพธ์จริง

## 3.1 การสร้างหน้าต่างแสดงผลสามมิติโดยใช้ GLUT (OpenGL Utility Toolkit)

ในโครงงานนี้ได้นำเสนอการจำลองคลื่นเป็นแบบสามมิติและได้นำ OpenGL มาใช้โดยผ่าน GLUT ซึ่งทำให้ง่ายในการสร้างหน้าต่างแสดงผล การรับข้อมูลจากแป้นพิมพ์หรือเมาส์ ในการจะใช้ GLUT นั้นมีขั้นตอนดังนี้ และในที่นี้จะใช้ภาษา C++ ในการพัฒนาโปรแกรม

### 3.1.1 Flow Chart ของการสร้างหน้าต่างแสดงผลด้วย GLUT



### 3.1.2 Code ที่ใช้ในการสร้างหน้าต่างแสดงผลด้วย GLUT

ขั้นตอนแรกให้ทำการ include ไฟล์ที่ชื่อ glut.h เข้ามาในไฟล์ main.cpp โดยใช้คำสั่ง

```
#include <gl\glut.h>
```

แล้วสร้างฟังก์ชันชื่อ main ซึ่งมีชนิดของการคืนค่าเป็น int โดยภายในฟังก์ชันจะเป็นการกำหนดค่าเริ่มต้นของ OpenGL ต่างๆ เมื่อกำหนดค่าเสร็จแล้วจึงทำการสร้างหน้าต่างที่ใช้ในการแสดงผลด้วยฟังก์ชัน glutCreateWindow() ดังตัวอย่างข้างล่างนี้

```
int main(int argc, char **argv)
{
    glutInit(&argc, argv); // ฟังก์ชันเริ่มต้น GLUT
    glutInitDisplayMode( GLUT_RGBA | GLUT_DOUBLE | GLUT_DEPTH ); // ปรับโหมดการวาด
    glutInitWindowSize(swidth, sheight); // ขนาดหน้าจอ
    glutInitWindowPosition( 200, 200 ); // กำหนดตำแหน่งของหน้าจอ
    glutCreateWindow( PROGRAM_TITLE ); // เมื่อกำหนดค่าเสร็จสร้างหน้าต่างขึ้นมา
    glutDisplayFunc(&renderScene); // กำหนด pointer ไปยังฟังก์ชันที่สร้างฉาก
    glutIdleFunc(&idleControl); // pointer ของฟังก์ชันควบคุมขณะว่าง
    glutReshapeFunc(&resize); // pointer ไปยังฟังก์ชันที่ใช้ย่อขยายหน้าต่าง
    glutKeyboardFunc(&keyboard); // pointer ไปยังฟังก์ชันที่ตรวจจับแป้นพิมพ์
    glutPassiveMotionFunc(&mousePassive); // pointer ไปยังฟังก์ชันตรวจจับเมาส์
    initGL(swidth, sheight); // ฟังก์ชันกำหนดค่าเริ่มต้นของ OpenGL ที่ผู้ใช้สร้างเอง
    glutMainLoop(); // เข้าสู่การวนโปรแกรมหลักของ GLUT
    return 0;
}
```

```
void initGL(int Width,int Height)
{
        glClearColor(0.1f, 0.1f, 0.1f, 0.0f); // กำหนดสีที่ใช้เมื่อทำการล้างบัฟเฟอร์
        glShadeModel(GL_SMOOTH); // กำหนด Shading Model
        glPolygonMode( GL_FRONT_AND_BACK, GL_FILL); // กำหนดวิธีการวาดรูปทรง
        resize(Width,Height); // เซ็ตแมทริกซ์ให้ตรงกับขนาดของหน้าต่าง
}
```

ส่วนในส่วนของการวาดฉากต่างๆนั้นสามารถนำคำสั่งต่างๆที่ต้องการใช้วาดไปใส่ไว้ในฟังก์ชัน renderScene ที่ได้กำหนดไว้ในคำสั่งในฟังก์ชัน main() ดังนี้

```
void renderScene()
{
        glutSolidTeapot(1); // วาดกาน้ำชาลงบนฉาก
}
```

### 3.2 การควบคุมมุมมองในโปรแกรม

ในโครงงานนี้เราได้คิดค้นวิธีที่สามารถทำให้กล้องสามารถมองได้อย่างอิสระซึ่งจะมีลักษณะคล้ายกับกล้องในเกม First-person Shooter (FPS) ซึ่งการควบคุมนั้นจะใช้เมาส์ในการเปลี่ยนมุมมองและใช้แป้นพิมพ์ในการเปลี่ยนตำแหน่งของกล้องเพื่อให้สะดวกในการดู

ขั้นตอนวิธีที่ใช้ในการสร้างกล้องก็คือเราได้กำหนดตำแหน่งของกล้องด้วยค่า x y z ตามแกนในโลกสามมิติและกำหนดมุมการหันของกล้องเป็น yaw และมุมเงยเป็น pitch ซึ่งจะนำไปใช้คำนวณหาทิศทางหรือจุดที่กล้องมองอยู่เพื่อนำไปใช้ในการควบคุมการเลื่อนตำแหน่งของกล้องเมื่อกดปุ่มบน keyboard และในมุมเงยของกล้อง (pitch) นั้นจะต้องกำกับไม่ให้ค่านั้นไปถึง 90 หรือ -90 ด้วยเนื่องจากจะทำให้ภาพที่ได้ออกมานั้นเกิดความผิดพลาดเนื่องจาก cos 90 = 0

Source code ของการหาตำแหน่งที่กล้องมองอยู่

```
GLfloat lookat_x;
GLfloat lookat_y;
GLfloat lookat_z;
lookat_x = camera_x + cosd(_yaw) * cosd(_pitch) * lookdistance;
lookat_y = camera_y + sind(_pitch) * lookdistance;
lookat_z = camera_z + sind(_yaw) * cosd(_pitch) * lookdistance;
```

ซึ่ง cosd(angle) ก็คือฟังก์ชัน cos ในตรีโกณมิติซึ่งรับค่าเป็นองศา

sind(angle) ก็คือฟังก์ชัน sin ในตรีโกณมิติซึ่งรับค่าเป็นองศา

การปรับฉากให้เข้ากับตำแหน่งและมุมมองของกล้องสามารถทำได้โดยใช้ function ดังนี้

```
gluLookAt(camera_x, camera_y, camera_z, lookat_x, lookat_y, lookat_z, 0, 1, 0);
```

## 3.3 ฟังก์ชันทั่วไปต่างๆที่ใช้ในโปรแกรม

### 3.3.1 ฟังก์ชันที่ใช้วาดตัวอักษรลงบนหน้าจอ

```
void drawText(int x, int y,void *font,char *str)
{
    glRasterPos2i(x, y);
    while(*str)glutBitmapCharacter(font, *str++);
}
```

เป็นฟังก์ชันที่ใช้ในการวาดตัวอักษรลงบนหน้าจอโดยกำหนดพิกัด x,y ซึ่งจะเริ่มจากมุมซ้ายล่างของหน้าจอ และสามารถกำหนด font ที่ต้องการได้

### 3.3.2 ฟังก์ชันที่ใช้โหลดพื้นผิวจากไฟล์

```
GLuint loadBMPTexture(char *filename, GLuint num_texture)
{
        unsigned char *texture;
        int i;
        int j = 0;
        FILE *file;
        BITMAPFILEHEADER fileheader;
        BITMAPINFOHEADER infoheader;
        RGBTRIPLE rgb;
        num_texture++;
        if((file = fopen(filename,"rb")) == NULL)return -1;
        fread(&fileheader,sizeof(fileheader),1,file);
        fseek(file,sizeof(fileheader),SEEK_SET);
        fread(&infoheader,sizeof(infoheader),1,file);
        texture = (byte *) malloc(infoheader.biWidth*infoheader.biHeight*4);
        memset(texture,0,infoheader.biWidth*infoheader.biHeight*4);
        for(i=0;i<infoheader.biHeight*infoheader.biWidth;i++)
        {
                fread(&rgb,sizeof(rgb),1,file);
                texture[j+0] = rgb.rgbtRed;
                texture[j+1] = rgb.rgbtGreen;
                texture[j+2] = rgb.rgbtBlue;
                texture[j+3] = 255;
                j+=4;
        }
        fclose(file);
        glBindTexture(GL_TEXTURE_2D,num_texture);
        glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_S,GL_CLAMP);
        glTexParameterf(GL_TEXTURE_2D,GL_TEXTURE_WRAP_T,GL_CLAMP);
       glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
        glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
        glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
        glTexImage2D(GL_TEXTURE_2D, 0, 4, infoheader.biWidth, infoheader.biHeight, 0,
        GL_RGBA, GL_UNSIGNED_BYTE, texture);
        gluBuild2DMipmaps(GL_TEXTURE_2D, 4, infoheader.biWidth, infoheader.biHeight,
        GL_RGBA, GL_UNSIGNED_BYTE, texture);
        free(texture);
        return num_texture;
}
```

ฟังก์ชันนี้ทำหน้าที่โหลดพื้นผิวเข้าไปยังหน่วยความจำของ OpenGL แล้วคืนค่ากลับเป็น ID ของพื้นผิวซึ่งจะนำไปใช้กับฟังก์ชัน glBindTexture() เวลาที่จะใช้กับวัตถุต่างๆ โดยรับค่าสองค่าคือชื่อไฟล์และ ID ของพื้นผิวล่าสุด

### 3.3.3 ฟังก์ชันที่ใช้ในการทำ Cube Mapping

```
void initCubeMap()
{
        for(int i = 0; i < 6; i++)loadBMPTexture(target[i], cubefilename[i], pixeldata[i]);
        glTexParameteri(GL_TEXTURE_CUBE_MAP_EXT, GL_TEXTURE_MIN_FILTER,
        GL_NEAREST_MIPMAP_LINEAR);
        glTexParameteri(GL_TEXTURE_CUBE_MAP_EXT, GL_TEXTURE_MAG_FILTER,
        GL_NEAREST);

        glEnable(GL_TEXTURE_CUBE_MAP_EXT);

       assert(mode == GL_NORMAL_MAP_EXT || mode == GL_REFLECTION_MAP_EXT);
        glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, mode);
        glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, mode);
        glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, mode);

        glTexParameteri(GL_TEXTURE_CUBE_MAP_EXT, GL_TEXTURE_WRAP_S,
GL_CLAMP);
        glTexParameteri(GL_TEXTURE_CUBE_MAP_EXT, GL_TEXTURE_WRAP_T,
GL_CLAMP);

        glEnable(GL_TEXTURE_GEN_S);
        glEnable(GL_TEXTURE_GEN_T);
        glEnable(GL_TEXTURE_GEN_R);
}
```

ฟังก์ชันนี้เป็นการใช้เทคนิค Cube Mapping เพื่อจำลองการสะท้อนของผิวน้ำโดยใช้พื้นผิว 6 ด้านซึ่งประกอบกันเป็นสี่เหลี่ยมลูกเต๋าและเปิดการใช้งานการสร้างการสะท้อนแบบต่างๆของ OpenGL

## 3.4 การแสดงผลของพื้นผิวน้ำ

Vertex Array Class ได้ถูกสร้างขึ้นเพื่อทำหน้าที่ควบคุมการคำนวณ การจัดเก็บข้อมูล และการแสดงผลของพื้นผิวน้ำโดยใช้วิธีของ Vertex Array และใช้เทคนิค Vertex Buffer Object ในการเก็บข้อมูลต่างๆที่จำเป็นในการแสดงผลเพื่อให้สามารถแสดงผลได้อย่างรวดเร็ว ซึ่งภายในคลาสจะเก็บข้อมูลตัวแปรต่างๆที่จำเป็นและมีฟังก์ชันต่างๆที่ใช้ในการคำนวณพื้นผิวน้ำ นอกจากนั้นยังมีส่วนของการทำ Multithreading เพื่อให้สามารถประมวลผลได้รวดเร็วขึ้นในกรณีที่ CPU เป็นแบบที่แบ่งเป็นหลายส่วน (Multicore CPU)

### 3.4.1 ขั้นตอนและฟังก์ชันต่างๆที่ใช้ในการแสดงผลพื้นผิวน้ำ

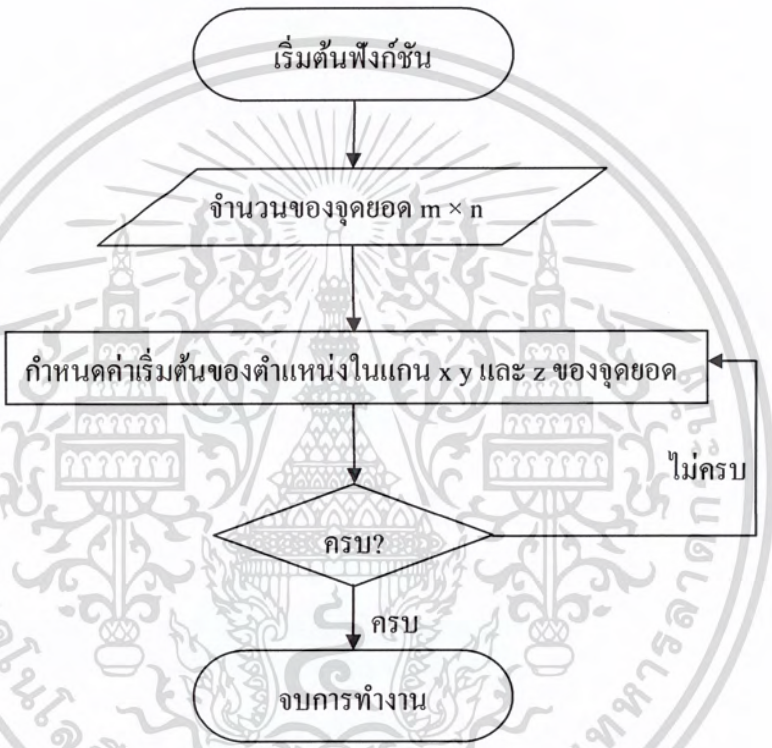
ในอันดับแรกจะต้องสร้าง object ซึ่งเป็น Instance ของคลาสขึ้นมาใน main ก่อนเพื่อใช้เป็นตัวกำหนดควบคุมพฤติกรรมต่างๆของคลาส เมื่อกำหนดตัวแปรของคลาสเรียบร้อยแล้ว โปรแกรมจะเรียกฟังก์ชัน VArray::VArray() โดยอัตโนมัติซึ่งเป็น Constructor ของคลาสทำหน้าที่กำหนดค่าตัวแปรเริ่มต้น จองพื้นที่หน่วยความจำ สร้าง thread ที่ใช้ประมวลผล และกำหนดการแบ่งหน้าที่ของแต่ละ thread


รูป 3.1 Flowchat ของฟังก์ชัน VArray::VArray()

หลังจากที่เราได้สร้างตัวแปรที่ชี้ไปยัง Instance ของคลาสและกำหนดค่าเริ่มต้นเรียบร้อยแล้ว จะตรวจว่าระบบรองรับ VBO Extension หรือไม่ และถ้าหากระบบรองรับจะเรียกฟังก์ชัน VArray::initVBO() ซึ่งเป็นฟังก์ชันที่กำหนดค่าเริ่มต้นของ Vertex Buffer Object ที่นำมาใช้ในการแสดงผล ซึ่งถ้าหากต้องการใช้ VBO จะต้องทำการเรียกฟังก์ชันนี้ก่อนเพื่อทำการจองพื้นที่และกำหนดค่าเริ่มต้นที่จำเป็นในการใช้เก็บข้อมูลต่างๆของจุดยอด

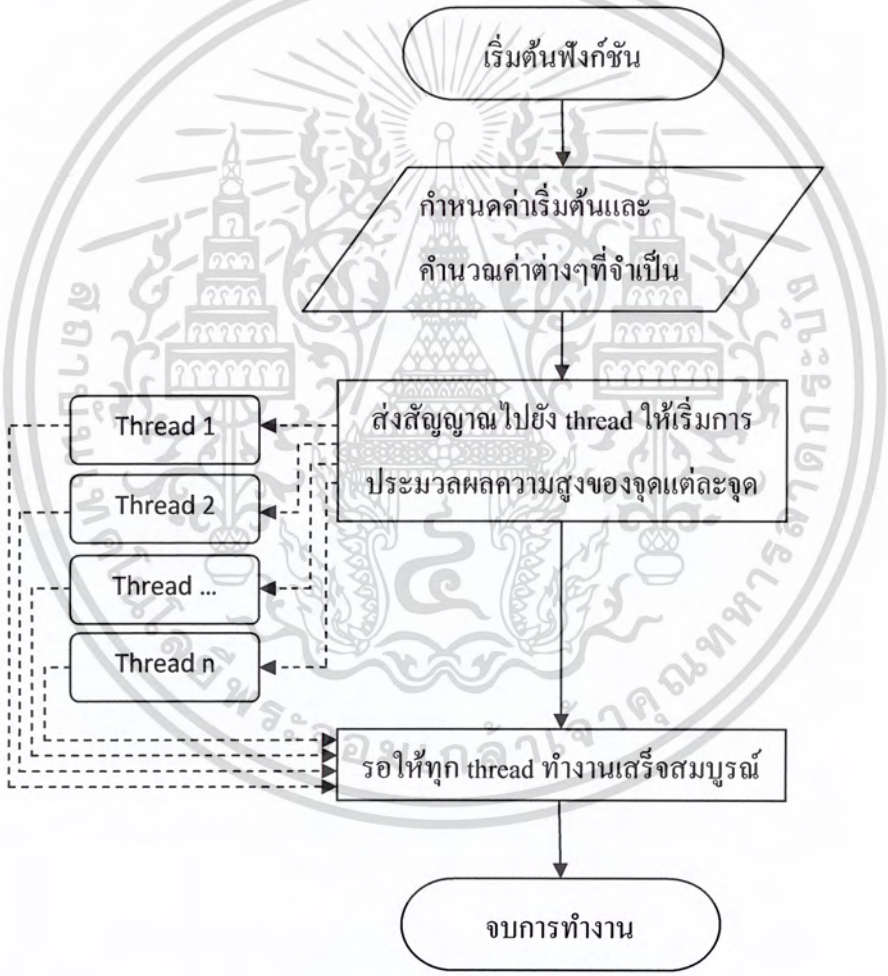

รูป 3.2 Flowchat ของฟังก์ชัน void VArray::initVBO()

เมื่อกำหนดค่าเริ่มต้นของ VBO แล้วก็จะมาถึงส่วนที่ต้องเริ่มการประมวลผล แต่ก่อนอื่นนั้นเราจะต้องปรับฉากที่จะวาดให้เหมาะสมกับมุมกล้องก่อน โดยการเรียกฟังก์ชัน Camera::updateCamera() และเรียกฟังก์ชัน makeCubeMap() เพื่อกำหนดการวาดการสะท้อนของพื้นผิวน้ำ แล้วจึงเรียกฟังก์ชัน Varray::generateVertexArray() เพื่อสร้างจุดยอดเริ่มต้นที่เรียงกันเป็นตารางขึ้นมาเพื่อนำไปปรับความสูงของแต่ละจุดให้ออกมาเป็นคลื่นในภายหลัง



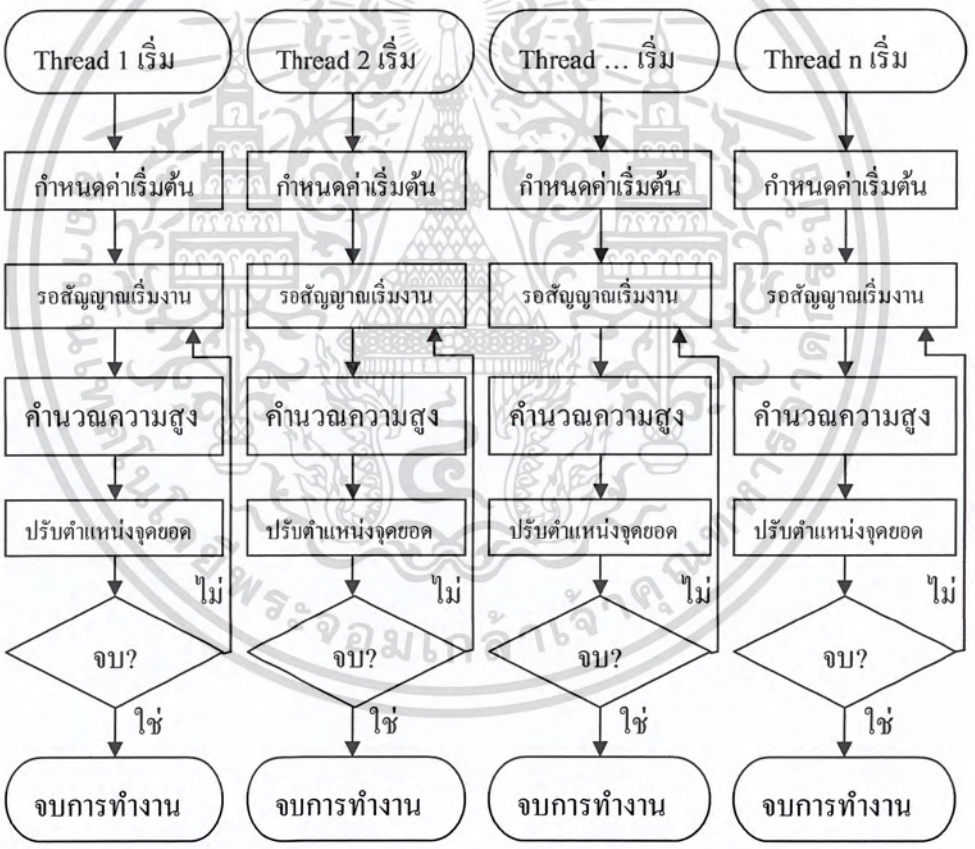

รูป 3.3 Flowchat ของฟังก์ชัน void VArray::generateVertexArray()

เมื่อได้กำหนดค่าเริ่มต้นของจุดยอดแต่ละจุดเสร็จสิ้นแล้วแล้ว จะต้องเรียกฟังก์ชัน Varray::generateOceanWave() ซึ่งใช้สำหรับปรับพื้นผิวที่สร้างขึ้นมาให้กลายเป็นพื้นผิวที่มีคลื่นที่สร้างขึ้นจากสมการคณิตศาสตร์เพื่อใช้จำลองคลื่นทะเลโดยคลื่นจะเปลี่ยนแปลงตามค่าเวลา t และค่าทิศทางลม winddir ที่ได้ป้อนให้กับฟังก์ชัน ซึ่งในการคำนวณในส่วนนี้จะใช้เวลานานจึงต้องนำเทคนิค Multithread มาใช้เพื่อให้สามารถคำนวณได้เร็วขึ้นใน CPU ที่มีหลาย core



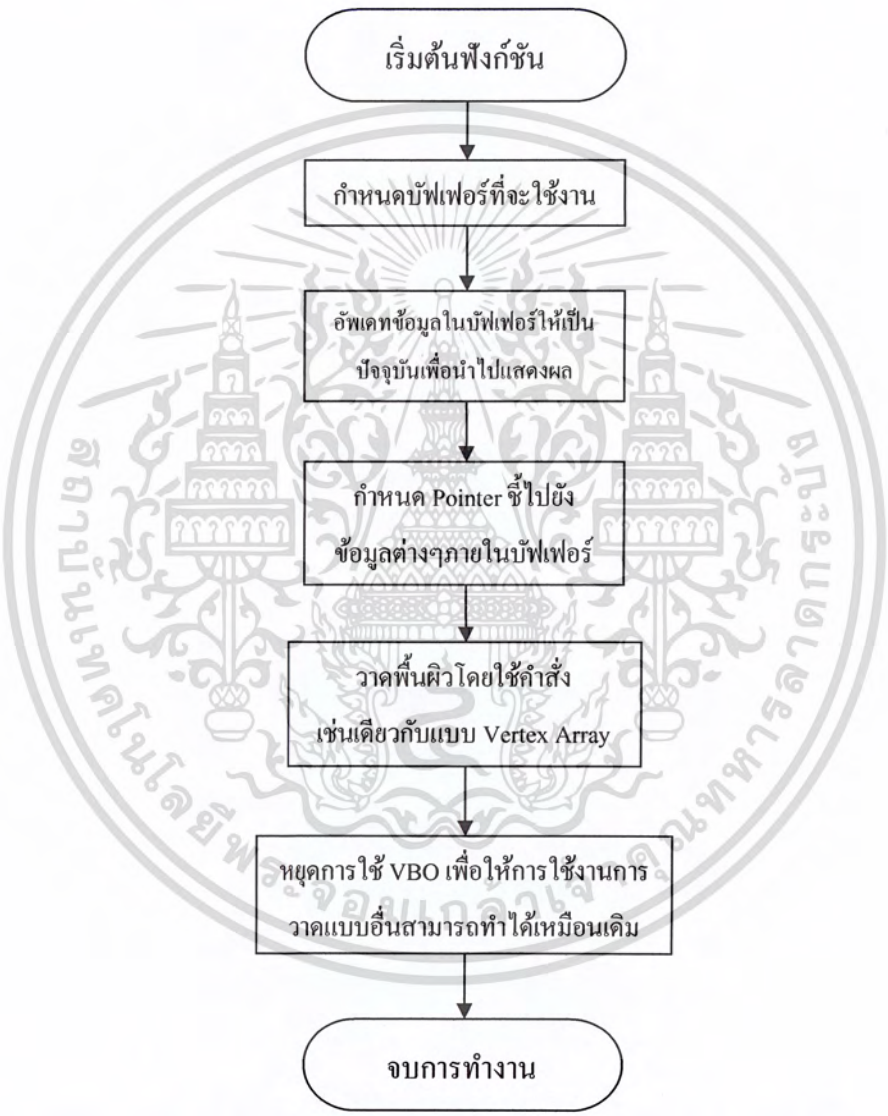

รูปที่ 3.4 Flowchat ของฟังก์ชัน void VArray::generateOceanWave(long t, float winddir)

Thread แต่ละ thread นั้นเป็น thread function ที่ชื่อ heightCalc() ที่จะถูกใช้งานขณะที่ฟังก์ชัน generateOceanWave() ทำงาน ซึ่งจะส่งสัญญาณไปให้ thread ทั้งหมดเริ่มคำนวณความสูงในแต่ละจุด และใน main จะรอจนกว่า thread ทั้งหมดจะทำงานเสร็จจึงจะทำงานต่อ thread แต่ละ thread จะทำหน้าที่คำนวณความสูงของจุดยอดแต่ละจุดที่นำมาใช้แทนผิวน้ำ โดยในกระบวนการคำนวณความสูงของจุดยอดนั้นต้องใช้เวลามากในกรณีที่มีจุดยอดมากจึงได้ทำการแบ่งเป็นหลายๆ thread เพื่อที่จะให้ประมวลผลแบบขนานได้ซึ่งจะทำได้รวดเร็วกว่าในกรณีที่ CPU แบ่งเป็นหลายหน่วยประมวลผล



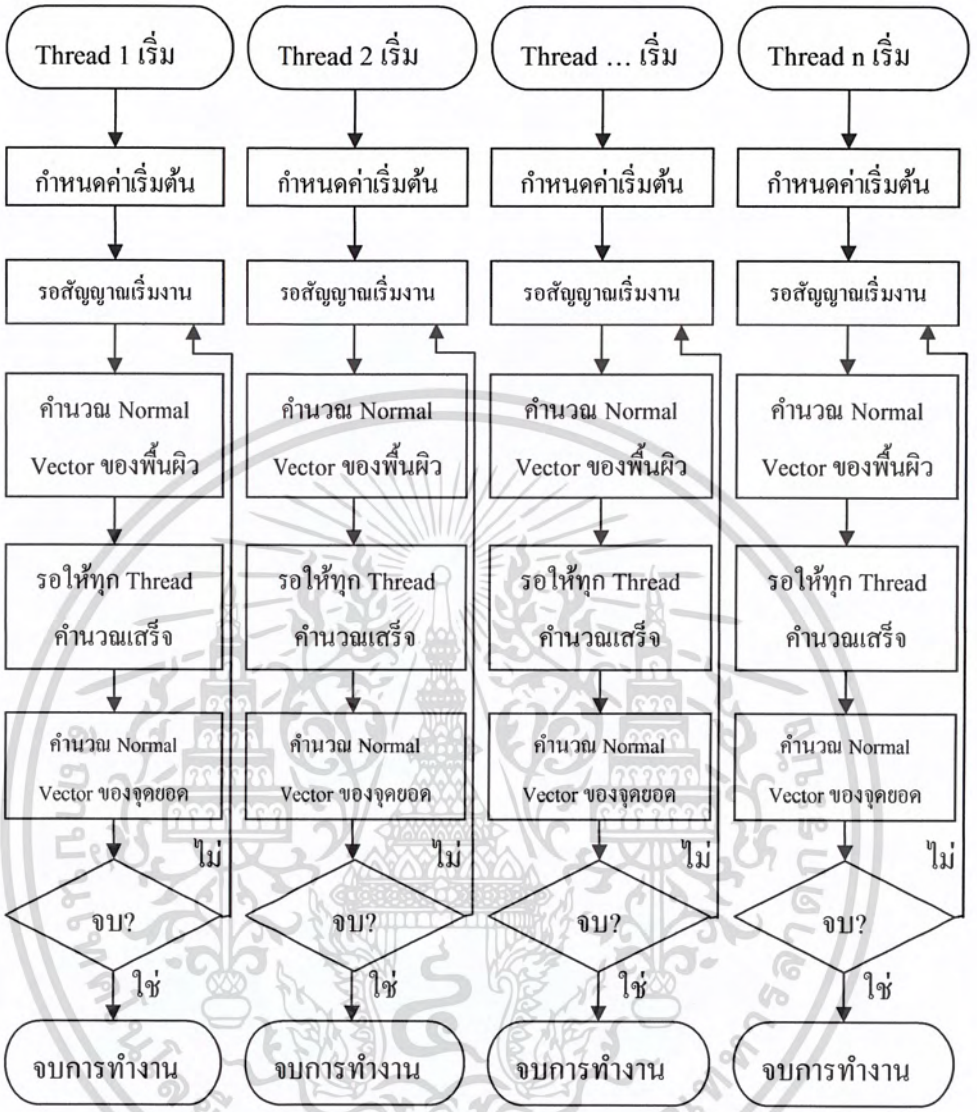

รูปที่ 3.5 Flowchat ของฟังก์ชัน UINT heightCalc(LPVOID p)

หลังจากที่คำนวณความสูงเสร็จแล้วก็ถึงเวลาที่จะนำผลลัพธ์ออกมาแสดงโดยการใช้ฟังก์ชัน Varray::renderVertexArray() ที่ใช้ในการแสดงผลลัพธ์ที่ประมวลผลแล้วออกมาเป็นภาพสามมิติใน OpenGL โดยจะเลือกใช้งาน VBO หรือไม่ก็ได้ ถ้าหากต้องการเลือกใช้ให้ประกาศ #define VBO_USED ไว้ที่หัวไฟล์ func.h



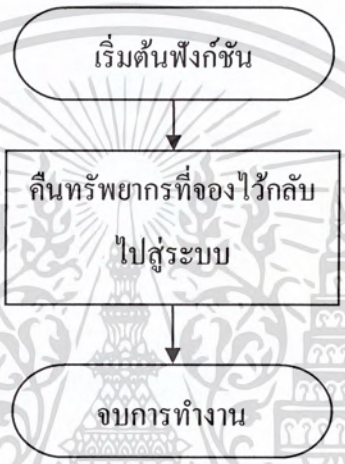

รูปที่ 3.6 Flowchat ของฟังก์ชัน void VArray::renderVertexArray()

ในฟังก์ชัน Varray::renderVertexArray() นั้นจะมีการคำนวณ Normal map เพื่อใช้ในการคำนวณแสงก่อนที่จะแสดงผล ซึ่งจะใช้ฟังก์ชันชื่อ normalCalc() ที่เป็น thread function ที่ใช้ในการคำนวณ Normal Vector ของแต่ละจุด โดยในงานนี้ได้ใช้การคำนวณแบบ Phong Shading เพื่อให้พื้นผิวดูมีความโค้งมนเสมือนจริงเมื่อใช้งานกับแหล่งกำเนิดแสงซึ่งได้ใช้เทคนิค Multithreading เข้ามาช่วยเช่นกันเพื่อให้ประมวลผลได้รวดเร็ว ในขั้นตอนนี้จะแบ่งการทำงานของ thread เป็น 2 ขั้นตอน โดยขั้นแรกจะคำนวณ Normal Vector ของพื้นผิวก่อน แต่ Normal Vector ที่ได้ในตอนนี้ถ้านำไปใช้จะได้ภาพแบบ Flat Shading เพื่อที่จะทำให้ออกมาเป็นแบบ Phong Shading นั้นเราจะต้องคำนวณหา Normal Vector ของจุดยอดแต่ละจุดโดยนำ Normal Vector ของพื้นผิวทุกด้านที่ใช้จุดยอดนี้ในการวาดมาหาค่าเฉลี่ย




รูปที่ 3.7 Flowchat ของฟังก์ชัน UINT normalCalc(LPVOID p)

เมื่อโปรแกรมทำงานจบ ก่อนที่จะปิดโปรแกรมนั้นเราจะต้องสร้างฟังก์ชันที่คืนหน่วยความจำให้กับระบบ ซึ่งในกรณีนี้เราได้ใช้ Destructor ที่จะถูกเรียกใช้โดยอัตโนมัติเมื่อโปรแกรมเริ่มปิดตัวลงโดยฟังก์ชันนี้จะชื่อเหมือน Constructor แต่จะมีเครื่องหมาย ~ นำหน้าฟังก์ชัน บ่งบอกว่าเป็น Destructor ของคลาส VArray ทำหน้าที่คืนหน่วยความจำและทรัพยากรต่างๆของระบบที่จองไว้เพื่อใช้ในการแสดงผลพื้นผิวน้ำเมื่อโปรแกรมจบการทำงาน




รูปที่ 3.8 Flowchat ของฟังก์ชัน VArray::~VArray()

# บทที่ 4
# ผลลัพธ์ของการจำลองคลื่น

จากการที่นำทฤษฎีและวิธีการทั้งหมดที่กล่าวมานั้นมาประกอบรวมเข้าด้วยกันเพื่อให้สามารถจำลองคลื่นบนคอมพิวเตอร์โดยใช้ OpenGL ได้แล้ว ได้พบกับปัญหาใหญ่ปัญหาหนึ่งก็คือความเร็วในการแสดงผลโดยนำมาทดสอบบนเครื่องที่มีอุปกรณ์ดังนี้

CPU : Intel Core i7 720QM (1.6 – 2.8 GHz, 8 Cores)

GPU : NVIDIA GeForce GTX 260M

Memory : DDR3 (1066 MHz) 4GB

เนื่องจากการคำนวณความสูงของคลื่นนั้นใช้พลังการประมวลผลของ CPU มาก จึงทำให้ประมวลผลได้ช้ามากในช่วงแรกของการเขียนโปรแกรมซึ่งมีเป้าหมายเพียงแค่ให้สามารถประมวลผลได้เท่านั้น จึงทำให้ประมวลผลได้ช้ามากถึง 1 ภาพต่อวินาทีโดยใช้การแบ่งช่วงความถี่และช่วงองศาแค่ 5 ส่วน สาเหตุมาจากแบ่งการประมวลผลในแต่ละส่วนของสมการเป็นแบบแยกฟังก์ชันเพื่อให้สามารถเข้าใจได้ง่าย แต่ก็มีข้อเสียมากคือการเรียกใช้ฟังก์ชันนั้นจะต้องการพลังการประมวลผลเพิ่มเติม และเนื่องจากในงานนี้ต้องประมวลผลจุดยอดถึง 10,000 จุดทำให้มีการเรียกฟังก์ชันมากกว่า 300,000 ครั้งในการประมวลผลเพียง 1 ภาพซึ่งหลังจากนั้นได้ทำการรวมฟังก์ชันให้เป็นฟังก์ชันเดียวก็สามารถลดการเรียกฟังก์ชันลงได้เป็นจำนวนมาก ทำให้ความเร็วการประมวลผลเพิ่มเป็น 5 ภาพต่อวินาที และได้ทำการปรับฟังก์ชันเพิ่มเติมอีกโดยแยกส่วนที่ไม่จำเป็นต้องประมวลผลหลายครั้งออกมานอก Loop ซึ่งสามารถเพิ่มความเร็วได้เป็น 15 ภาพต่อวินาที ซึ่งหลังจากนั้นได้ทำการปรับฟังก์ชันเพิ่มเติมและได้แบ่งการประมวลผลออกเป็น thread ทำให้ความเร็วเพิ่มขึ้น โดยแบ่งช่วงความถี่และช่วงองศาเป็น 9 ช่วงก็ยังสามารถประมวลผลได้ประมาณ 22 ภาพต่อวินาทีและเมื่อนำเทคนิค VBO (Vertex Buffer Object) ของ OpenGL มาใช้ซึ่งทำให้การวาดฉากทำได้เร็วขึ้นก็สามารถแสดงผลได้ประมาณ 24 ภาพต่อวินาที และในขั้นสุดท้ายได้แบ่งการประมวลผลในส่วนที่ใช้เวลามากออกเป็นหลาย thread เพื่อให้ประมวลผลแบบขนานได้และใช้งาน CPU แบบหลาย core ได้อย่างเต็มประสิทธิภาพ

## 4.1 การวัดประสิทธิภาพในการประมวลผลแบบ Multithread

ตาราง 4.1 แสดงการเปรียบเทียบความเร็วในการประมวลผล

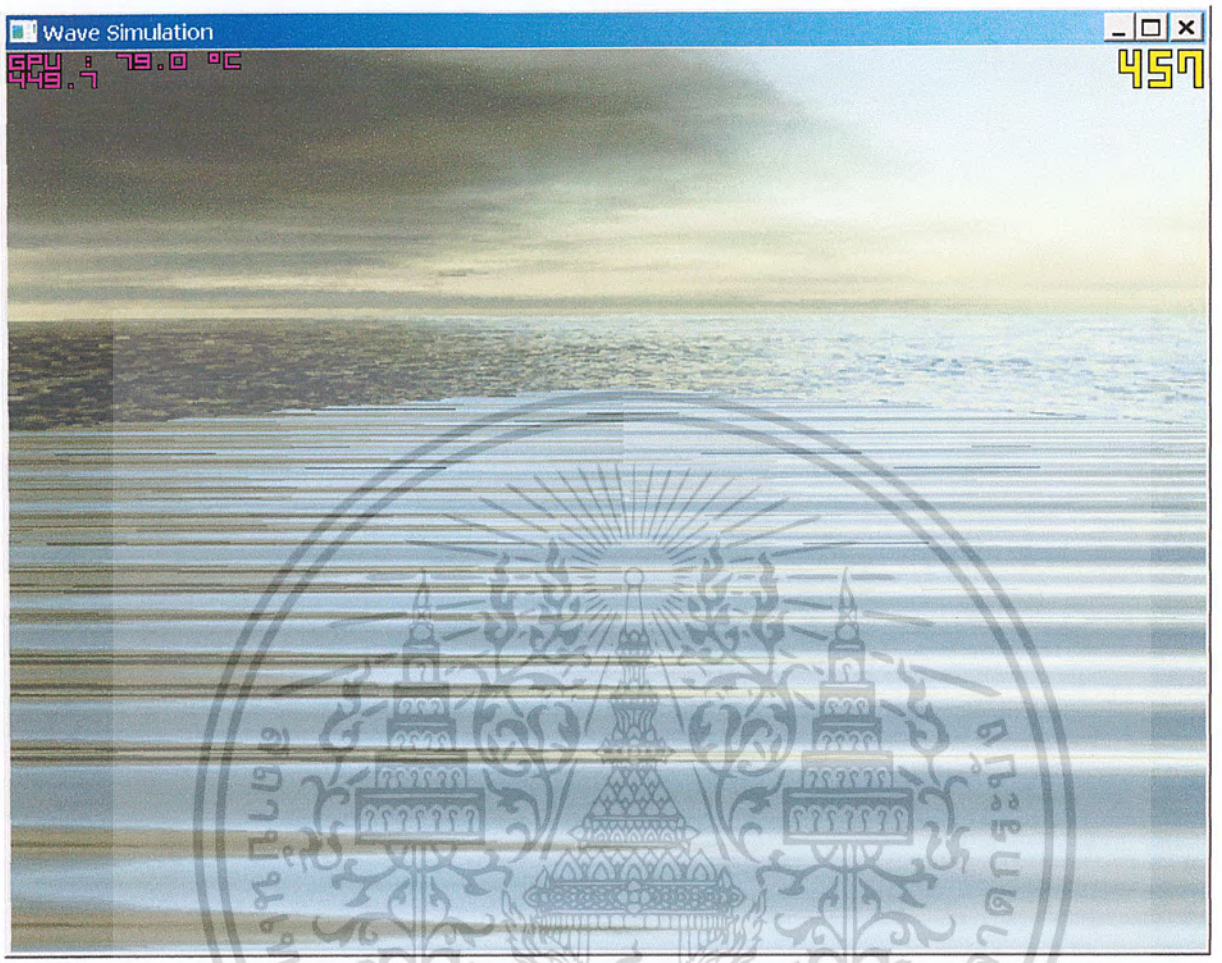
| จำนวน thread ที่ประมวลผล Height map | จำนวน thread ที่ประมวลผล Normal map | ปริมาณการใช้ CPU (%) | ความเร็วในการประมวลผล (ภาพ/วินาที) |
|---|---|---|---|
| 1 | 1 | 11 - 13 | 24 |
| 2 | 1 | 16 - 21 | 32 |
| 3 | 1 | 28 - 36 | 42 |
| 4 | 1 | 36 - 45 | 50 |
| 5 | 1 | 43 - 52 | 60 |
| 6 | 1 | 48 – 58 | 65 |
| 7 | 1 | 51 – 70 | 72 |
| 8 | 1 | 42 - 64 | 63 |
| 1 | 2 | 12 - 13 | 24 |
| 1 | 3 | 12 – 14 | 24 |
| 1 | 4 | 12 – 15 | 25 |
| 1 | 5 | 13 – 15 | 25 |
| 1 | 6 | 13 – 15 | 25 |
| 1 | 7 | 13 – 15 | 25 |
| 1 | 8 | 11 – 14 | 22 |
| 2 | 2 | 18 – 22 | 33 |
| 4 | 4 | 40 – 49 | 56 |
| 7 | 7 | 62 – 78 | 81 |

จากตารางจะเห็นว่า เมื่อแยกการประมวลผลในส่วนของ Height Map มากขึ้นจะเห็นว่าสามารถประมวลผลได้เร็วขึ้นมาก เนื่องจากในส่วนของการประมวลผลความสูงของจุดนั้นต้องทำถึง 10,201 จุดในโครงงานนี้ เมื่อให้โปรแกรมสามารถประมวลผลแบบขนานได้ ประสิทธิภาพในการประมวลผลจึงเพิ่มขึ้นมาก แต่ทาง Normal Map นั้นจะเห็นผลน้อย เนื่องจาก Normal Map นั้นประมวลผลไม่นานมากนัก แต่ก็ยังได้รับประโยชน์จากการแยกประมวลผล และจะเห็นได้ว่า หากแยกเป็น 8 ส่วนการประมวลผลจะช้าลง เนื่องจากการประมวลผลแบบ Multithread นั้น OS จะต้องทำ thread switching ซึ่งในการควบคุม thread switching นั้นก็ต้องใช้ CPU เช่นกัน เพราะการแบ่งเป็น 8 นั้นจะทำให้ทุก core ของ CPU (8 cores) มี thread ไปทำงานอยู่ ซึ่งจะแย่งการใช้ CPU กับส่วนควบคุมของ OS ทำให้การทำ thread switching นั้นช้าลง ส่งผลให้การประมวลผลช้าลงไปด้วย

## 4.2 การจำลองในช่วงความถี่และช่วงองศาต่างๆ

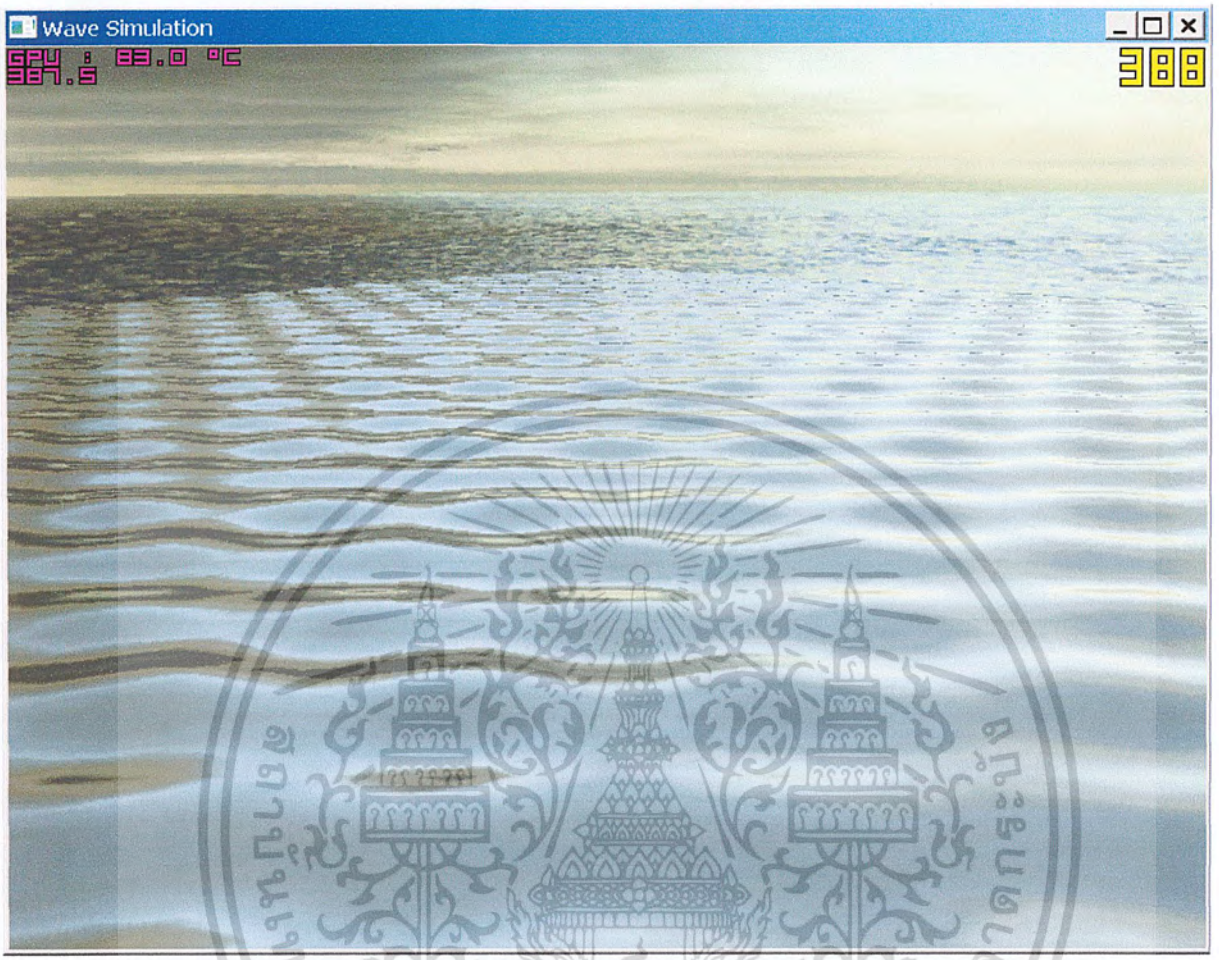
ในหัวข้อนี้จะเป็นรูปเปรียบเทียบการจำลองในการใช้ช่วงความถี่และช่วงองศาที่แตกต่างกันไป โดยจะมีตัวเลขบอกความเร็วประมวลผลอยู่ที่มุมขวาบนของภาพ และการประมวลผลในส่วนนี้ทั้งหมดจะแบ่ง thread การประมวลผลเป็น 7 thread ทั้ง Normal และ Height map ในการจำลองจะใช้ปริมาณช่องของสี่เหลี่ยมเป็นจำนวน 100 x 100 ช่อง ซึ่งเท่ากับ 101 x 101 จุดยอด หรือ 10,201 จุด ซึ่งรูปจะอยู่ในหน้าที่ 48 - 54


รูป 4.1 การจำลองโดยใช้ช่วงความถี่ 2 และช่วงองศา 2



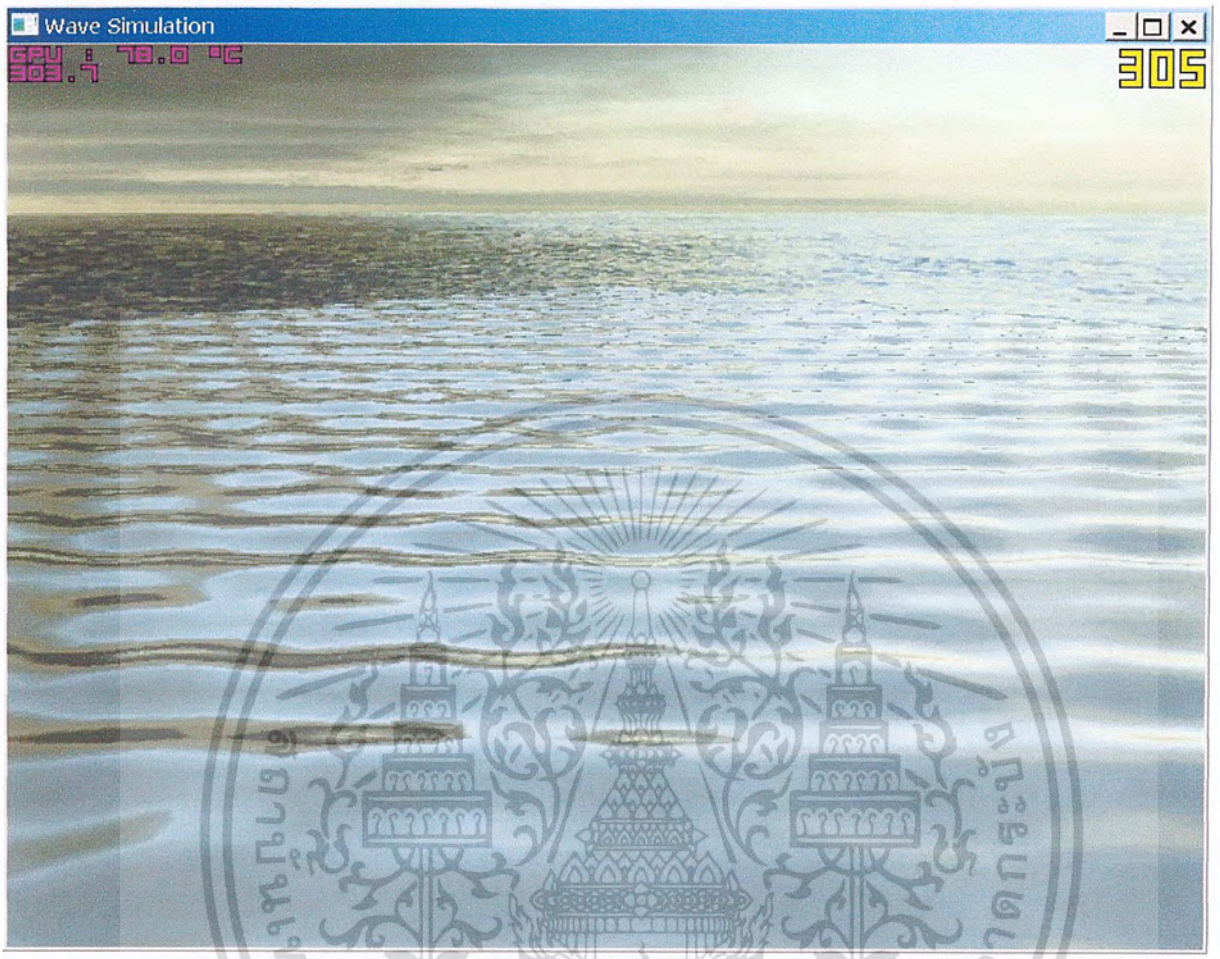

รูป 4.2 การจำลองโดยใช้ช่วงความถี่ 4 และช่วงองศา 2


รูป 4.3 การจำลองโดยใช้ช่วงความถี่ 9 และช่วงองศา 2




รูป 4.4 การจำลองโดยใช้ช่วงความถี่ 2 และช่วงองศา 4


รูป 4.5 การจำลองโดยใช้ช่วงความถี่ 2 และช่วงองศา 9




รูป 4.6 การจำลองโดยใช้ช่วงความถี่ 4 และช่วงองศา 4


รูป 4.7 การจำลองโดยใช้ช่วงความถี่ 9 และช่วงองศา 9

## 4.3 ผลลัพธ์ที่ได้จากการจำลองคลื่น

จากผลลัพธ์ที่ได้เราได้เลือกใช้แบบแบ่งช่วงความถี่และช่วงองศาเป็นอย่างละ 9 ซึ่งจะเห็นได้ว่าการแบ่งช่วงมากขึ้นจะทำให้ผลลัพธ์ที่ได้ออกมาสวยงามสมจริงยิ่งขึ้น แต่ก็ต้องแลกด้วยระยะเวลาที่ใช้ในการประมวลผลที่มากขึ้น หลังจากนั้นเราได้เพิ่มกล่องไม้ที่ลอยบนผิวน้ำเข้าไปโดยใช้หลักฟิสิกส์ของ Archimedes ในการคำนวณการลอยตัวของวัตถุ โดยจำนวนกล่องสามารถปรับได้ตอนเริ่มโปรแกรม แต่กล่องที่ใช้จำลองนั้นสามารถซ้อนทับกันเองได้ เนื่องจากการชนกันและการมีปฏิสัมพันธ์ของกล่องแต่ละกล่องนั้นอยู่นอกขอบเขตของโครงงานนี้


รูป 4.8 การจำลองคลื่นโดยมีปฏิสัมพันธ์กับวัตถุ

# บทที่ 5
# บทสรุปและข้อเสนอแนะ

## 5.1 บทสรุป

จากการจำลองคลื่นแบบมีปฏิสัมพันธ์ที่ใช้ในโครงงานนี้ มีวัตถุประสงค์เพื่อที่จะนำสมการจากงานวิจัยมาใช้จำลองคลื่นบนคอมพิวเตอร์ได้และมีความสวยงามสมจริง แต่ยังสามารถประมวลผลได้อย่างรวดเร็ว ซึ่งในงานวิจัยอื่นๆที่ได้ศึกษามาจะเห็นได้ว่าการจำลองคลื่นหลายๆงานนั้นทำได้ช้า และไม่เหมาะสมกับการนำมาพัฒนาต่อ โดยในงานนี้ได้พยายามที่จะพัฒนาการจำลองให้สามารถทำได้รวดเร็วที่สุด และได้นำเทคนิค Multithreading มาใช้ เพื่อที่จะสามารถดึงสมรรถภาพของ CPU ที่มีหลาย core ออกมาได้อย่างเต็มที่ นอกจากนั้นยังได้ใช้เทคนิคของ OpenGL เพื่อจำลองการสะท้อนของพื้นผิวน้ำเพื่อให้เกิดความสวยงามสมจริง

1) การจำลองคลื่นจากวิธีที่ใช้ในงานนี้เป็นการคำนวณความสูงของจุดแต่ละจุดบนพื้นผิวน้ำ โดยใช้วิธีการ cosine wave superimposition หรือการคำนวณการแทรกสอดของคลื่นเป็นจำนวนมากบนจุดๆหนึ่งให้ออกมาเป็นค่าความสูง
2) ในงานวิจัยทางด้านการจำลองคลื่นทั่วๆไปนั้นไม่ได้ใช้การคำนวณแบบ Multithreading ซึ่งใน CPU รุ่นใหม่ๆนั้นเกือบทั้งหมดจะเป็นแบบหลาย core จึงทำให้การนำ Multithreading มาใช้มีประโยชน์ในการเพิ่มความเร็วในการประมวลผลได้มาก
3) นอกจากนั้นในงานนี้ยังได้เพิ่มการมีปฏิสัมพันธ์กับวัตถุเข้าไปโดยการใช้หลักการลอยตัวของ Archimedes ในการคำนวณแรงต่างๆที่มากระทำต่อวัตถุในของไหล ซึ่งจะไม่ขัดกับหลักฟิสิกส์ แต่ถึงกระนั้นหลักของ Archimedes ก็ไม่ได้คำนึงถึงปัจจัยหลายๆอย่างเช่น แรงตึงผิว และความหนืดของของไหล
4) ในการนำเทคนิค Cube Mapping ของ OpenGL มาใช้นั้น สามารถทำให้พื้นผิวน้ำมีการสะท้อนสภาพแวดล้อมโดยรอบได้ โดยแทบไม่มีการใช้พลังการประมวลผลหรือทรัพยากรเพิ่ม เป็นวิธีที่เหมาะแก่การนำไปประยุกต์เป็นโปรแกรมอย่างอื่นได้ เช่นใช้ในเกม ที่ไม่ต้องการให้เหมือนจริงตรงตามหลักความจริงทุกอย่าง แต่เน้นความสวยงามที่เหมาะสมและการประมวลผลที่รวดเร็ว

## 5.2 ข้อจำกัด

1) ในการแบ่งการประมวลผลเป็นหลายๆส่วนนั้น การแบ่งให้มากขึ้นไม่ใช่ว่าจะเป็นผลดีเสมอไป เช่น ในกรณีของเครื่องที่ใช้ในการทดสอบในงานนี้มี CPU ที่มี 8 cores ถ้าหากแบ่งการประมวลผลเป็น 8 ส่วนจะทำให้โปรแกรมไปแย่งทรัพยากรกับ OS ซึ่งทำหน้าที่ควบคุมการแบ่งทรัพยากร ทำให้การแบ่งทำได้ไม่ดีและการทำงานช้าลง หรือถ้าหากำแบ่งเกินจำนวน core ของ CPU ก็ไม่เกิดผลดีเพิ่ม เนื่องจากใน core หนึ่งๆจะมี thread ที่มาใช้งานมากกว่าหนึ่ง ทำให้ OS ต้องแบกรับภาระการทำ thread switching เพิ่มขึ้นแก ส่งผลให้การทำงานช้าลงไปอีก แต่ถึงกระนั้นก็มีปัจจัยทาง Hardware หลายๆอย่างที่ทำให้จำนวนการแบ่งที่เหมาะสมในแต่ละเครื่องแตกต่างกันไป

2) ในงานนี้ได้ใช้ OpenGL เพื่อให้มีความสามารถนำไปรันบน OS อื่นๆนอกจาก Windows ได้ แต่ว่าในงานนี้ได้ใช้ส่วนควบคุมการทำ Multithreading ของ Windows จึงทำให้การนำไปรันบน OS อื่นต้องทำการปรับตรงส่วนนี้ให้เข้ากับระบบการจัดการ thread ของ OS นั้นๆ หรือใช้ Library ที่รองรับการจัดการ thread ที่สามารถใช้ได้หลาย OS

## 5.3 ข้อเสนอแนะ

1) ในการเลือกจำนวน thread ที่ต้องการจะแบ่งนั้น ควรจะต้องทำการทดสอบว่าแบ่งเท่าไหร่ถึงจะดีที่สุด เนื่องจากในเครื่องแต่ละเครื่องนั้นมี Hardware ที่แตกต่างกัน จึงไม่สามารถกำหนดการแบ่งที่แน่ชัดได้

2) หากจะนำโปรแกรมไปดัดแปลงให้สามารถใช้งานบน OS อื่นๆนอกจาก Windows ได้ จะต้องปรับโครงสร้างบางส่วนที่ยังใช้ Library ของ Windows อยู่เช่นในส่วนของการจัดการ thread อาจจะใช้ pthread แทน ซึ่งเป็น Library ที่สามารถทำงานได้บนหลาย OS

# เอกสารอ้างอิง

[1] Xin Z.; Fengxia L.; Shouyi Z.; Zhisheng L., **"Ocean Wave Simulation under Wind Change Effect"** *Innovative Computing, Information and Control, 2006. ICICIC '06. First International Conference on* , vol.3, no., pp.26-29, Aug. 30 2006-Sept. 1 2006.

[2] **OpenGL**, http://en.wikipedia.org/wiki/Opengl.

[3] **GLUT**, http://en.wikipedia.org/wiki/OpenGL_Utility_Toolkit.

[4] **Smooth Shading**, http://en.wikipedia.org/wiki/Phong_shading.

[5] **Vertex Array**, http://www.songho.ca/opengl/gl_vertexarray.html.

[6] **Vertex Buffer Object**, http://www.songho.ca/opengl/gl_vbo.html.

[7] **Environment Mapping**, http://en.wikipedia.org/wiki/Environment_mapping.

[8] **Archimedes' Principle of Buoyancy**, http://en.wikipedia.org/wiki/Buoyancy.