# บทที่ 4
# ผลการวิจัย

เนื้อหาในบทนี้จะเป็นการแสดงรายละเอียดผลการทดลอง โดยในส่วนของการพัฒนาโปรแกรมต่าง ๆ นั้นจะแสดงรายละเอียดของระบบที่พัฒนาได้ เช่น การอิมพลิเมนต์ชุดคำสั่งที่สำคัญ และส่วนติดต่อผู้ใช้แบบกราฟิกส์ เป็นต้น ดังรายละเอียดต่อไปนี้

## 4.1 การเก็บข้อมูล

ผู้วิจัยนำผลการวิเคราะห์และออกแบบระบบที่ได้กล่าวในบทที่ 3 มาอิมพลิเมนต์ให้ทำงานได้จริง โดยใช้เครื่องมือที่หลากหลายและสามารถทำงานประสานกันได้ โดยมีรายละเอียดเกี่ยวกับการอิมพลิเมนต์และการเก็บข้อมูลเพื่อการทดสอบการทำงานของระบบดังนี้

### 4.1.1 เครื่องมือที่ใช้ในการพัฒนาโปรแกรม

เครื่องมือที่ใช้สำหรับการพัฒนาแบ่งได้เป็นสองกลุ่มหลักตามระบบปฏิบัติการที่ใช้ในการทำงาน นั่นคือโปรแกรมบนเว็บและโปรแกรมบนสมาร์ทโฟน โดยแต่ละแบบมีการใช้เครื่องมือที่ต่างกัน ดังนี้

1) การพัฒนาโปรแกรมบนเว็บ

โปรแกรมบนเว็บแบ่งเป็นสองส่วนหลักคือส่วนที่เกี่ยวข้องกับการจัดการข้อมูลเรือและส่วนของการบริการข้อมูลตำแหน่งเรือให้สมาร์ทโฟน ซึ่งทั้งสองส่วนนี้ผู้วิจัยใช้ภาษาพีเอชพี 5 (PHP 5) เป็นภาษาหลักในการพัฒนาโปรแกรมและใช้มายเอสคิวแอล (MySQL) เป็นระบบจัดการฐานข้อมูล ในส่วนของการจัดการข้อมูลตำแหน่งและการแสดงเรือบนแผนที่นั้น ผู้วิจัยได้พัฒนาเว็บแอปพลิเคชันสำหรับการแสดงแผนที่โดยการใช้คลังโปรแกรมกูเกิลแมพเอพีไอ (GoogleMap API) ของกูเกิลและใช้ภาษาจาวาสคริปต์ (JavaScript) เป็นตัวจัดการและประสานงานโปรแกรมระหว่างผู้ใช้กับเครื่องให้บริการโดยใช้เทคนิคเอแจ็กซ์ (Asynchronous JavaScript And XML : AJAX) ในการทำงาน

ในขั้นตอนการทดสอบการทำงานทำในคอมพิวเตอร์เดลวอสโตร 3460 (DELL VOSTRO 3460) ใช้หน่วยประมวลผลกลางอินเทลคอร์ไอเซเว่น (Intel Core I7) รุ่น 3632QM มีหน่วยความจำหลัก 8 กิกะไบต์ และใช้ระบบปฏิบัติการไมโครซอฟต์วินโดวส์ 10 (Microsoft Windows 10) สำหรับการใช้งานจริงนั้นผู้วิจัยได้นำชุดคำสั่งไปติดตั้งในเครื่องคอมพิวเตอร์ให้บริการของมหาวิทยาลัยราชภัฏรำไพพรรณีใช้ระบบปฏิบัติการลินุกซ์ (Linux)

2) การพัฒนาโปรแกรมบนแอนดรอยด์

ในส่วนของการแสดงผลความจริงเสริมบนสมาร์ทโฟนนั้น ผู้วิจัยได้นำการออกแบบที่ได้กล่าวมาแล้วในบทที่ 3 มาพัฒนาเป็นโปรแกรมประยุกต์ที่ทำงานบนระบบปฏิบัติการแอนดรอยด์ด้วยการเขียนโปรแกรมภาษาจาวา (Java) และใช้โปรแกรมประยุกต์แอนดรอยด์สตูดิโอ (Android Studio) เวอร์ชัน 2.1.1 เป็นเครื่องมือหลักในการพัฒนา โดยกำหนดใช้ชุดพัฒนาซอฟต์แวร์ขั้นต่ำเป็น Android API 15 และกำหนดค่าอื่น ๆ ในการสร้างโปรแกรมดังแสดงในตารางที่ 4.1

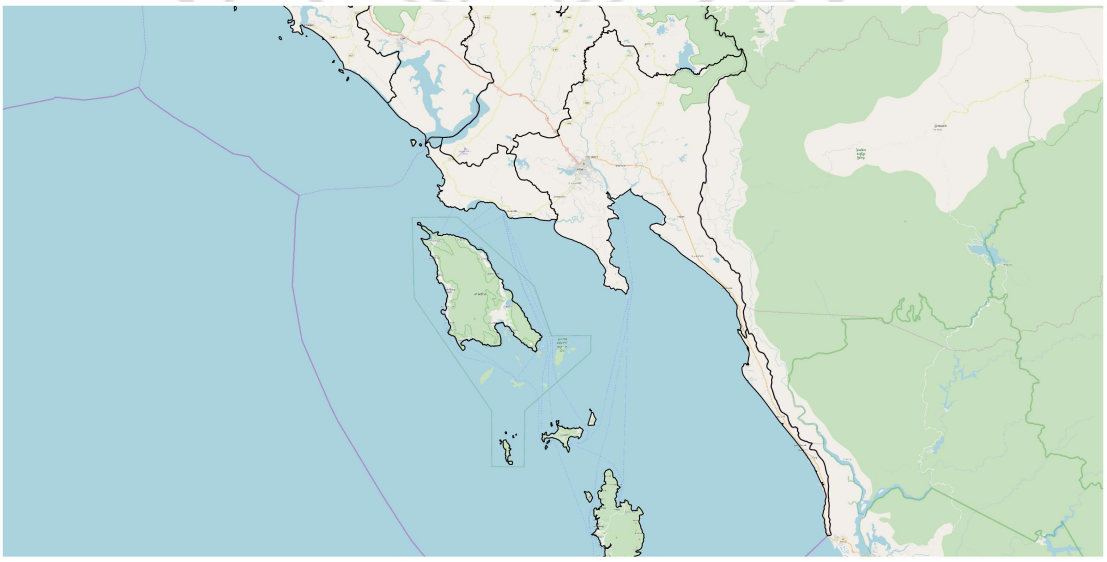
**ตารางที่ 4.1** เครื่องมือที่ใช้ในการพัฒนาโปรแกรมบนระบบปฏิบัติการแอนดรอยด์

| ข้อมูลการพัฒนา | ค่าที่กำหนด |
|---|---|
| Android Studio | 2.1.1 |
| Min SDK version | API 15 |
| Compile SDK version | API 23 (Marshmallow) |
| Target SDK version | API 23 |
| Building tools version | 23.0.3 |

ในส่วนของการแสดงกราฟิกส์บนสมาร์ทโฟนนั้น ผู้วิจัยใช้คลังโปรแกรมจีแอลอีเอสทู (GLES 2.0) ซึ่งเป็นคลังโปรแกรมต่อขยายของโอเพนจีเอลบนสมาร์ทโฟน และทดสอบการทำงานบนสมาร์ทโฟนซัมซุงกาแล็กซีเอสทรี (Samsung Galaxy S3), ซัมซุงกาแล็กซีอีไฟว์ (Samsung Galaxy E5) และซัมซุงกาแล็กซีเอเซเว่น (Samsung Galaxy A7)

### 4.1.2 พื้นที่ศึกษา

ผู้วิจัยได้เลือกพื้นที่สำหรับการเก็บข้อมูลที่มีลักษณะคล้ายบริเวณเขตเศรษฐกิจพิเศษตราด และเนื่องจากรูปแบบเรือประมงที่ใช้ในจังหวัดตราดไม่แตกต่างจากพื้นที่อื่น ๆ มากนัก ดังนั้นผู้วิจัยจึงเลือกทดสอบการทำงานของระบบบริเวณท่าเรือประมงหมู่บ้านปากน้ำแขมหนู ตำบลตะกาดเง้า อำเภอท่าใหม่ จังหวัดจันทบุรี ซึ่งแม้ว่าสภาพทางภูมิศาสตร์บนบกที่แตกต่างกัน แต่การทดสอบการทำงานนี้ทำในทะเลห่างจากชายฝั่งไปประมาณ 10 - 20 กิโลเมตร ดังนั้นความแตกต่างนี้จึงไม่ส่งผลกระทบใด ๆ ต่อการทดสอบการทำงานของระบบ ดังภาพที่ 4.1 และภาพที่ 4.2

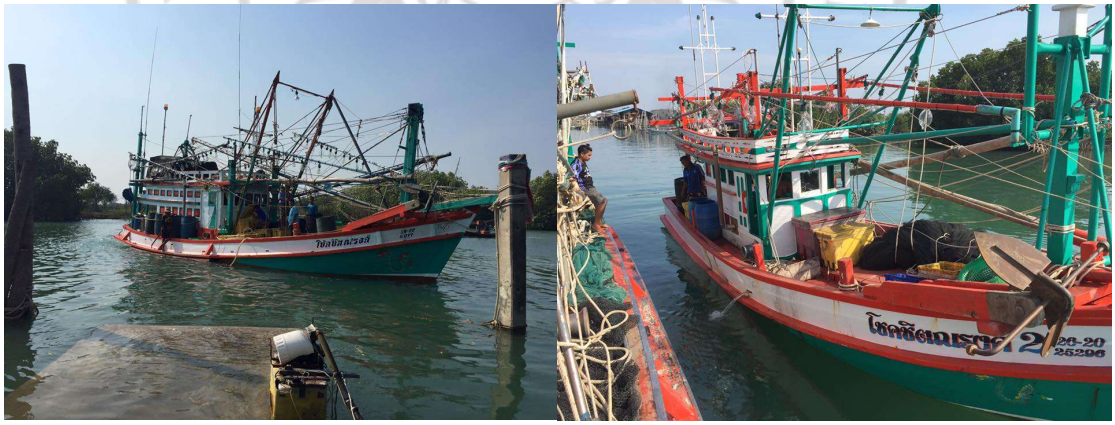


**ภาพที่ 4.1** พื้นที่ศึกษา
**ที่มา :** OpenStreetMap, 2018

**ภาพที่ 4.2** ท่าเรือที่ใช้ในการศึกษา

### 4.1.3 วิธีการเก็บข้อมูล

ผู้วิจัยติดตั้งโปรแกรมประยุกต์สำหรับการบันทึกและติดตามตำแหน่งเรือประมงลงบนสมาร์ทโฟนทั้งสามรุ่นดังที่ได้กล่าวมาแล้ว จากนั้นให้ผู้ช่วยผู้วิจัยนำสมาร์ทโฟนขึ้นเรือประมงและเคลื่อนที่ออกจากฝั่งไปยังแหล่งหาปลาเป้าหมายซึ่งอยู่ห่างจากฝั่งไปประมาณ 20 กิโลเมตร โดยระหว่างการเดินทางนั้นผู้ช่วยผู้วิจัยได้เปิดโปรแกรมเพื่อส่งข้อมูลกลับมาที่เครื่องให้บริการตลอดเวลา ภาพที่ 4.3 แสดงภาพเรือประมงโชคชิตณรงค์ 2 ซึ่งเป็นเรือที่ใช้ในการทดลอง



**ภาพที่ 4.3** เรือโชคชิตณรงค์ 2



ภาพที่ 4.4 แสดงการบันทึกข้อมูลภายในเรือประมงที่ใช้ในการทดลอง โดยมีนายชิตณรงค์ ชาวไชย นักศึกษาสาขาวิชาภูมิสารสนเทศ ซึ่งในขณะนั้นอยู่ชั้นปีที่ 4 เป็นผู้ช่วยวิจัย ทำหน้าที่เก็บข้อมูลภาคสนาม ในช่วงเดือนธันวาคม พ.ศ. 2559 ถึงมกราคม พ.ศ. 2560 การดำเนินการส่วนนี้เป็นการจำลองการส่งข้อมูลจากเรือประมงกลับมายังเครื่องให้บริการ ซึ่งในงานวิจัยนี้กำหนดให้ใช้เครื่องให้บริการเป็นเสมือนศูนย์ควบคุมการติดตามเรือ ในอีกด้านหนึ่งจะมีผู้ช่วยผู้วิจัยใช้โปรแกรมประยุกต์เพื่อการแสดงความจริงเสริมที่ได้ติดตั้งลงบนสมาร์ทโฟนเช่นเดียวกัน ระหว่างทดสอบได้มีการเปิดการเชื่อมต่ออินเทอร์เน็ตผ่านทางเครือข่ายจีพีอาร์เอสไว้ตลอดเวลา และผู้ช่วยผู้วิจัยคนที่สองนี้จะ

ตรวจสอบความถูกต้องของตำแหน่งของเรือประมงด้วยสายตา ภาพที่ 4.4 (ก) แสดงหน้าต่างโปรแกรมหลักซึ่งในภาพแสดงข้อมูลต่าง ๆ ที่อ่านได้จากเซ็นเซอร์ในสมาร์ทโฟน โดยมีปรากฏดาวเทียมทั้งหมด 32 ดวงแต่นำมาใช้ในการคำนวณตำแหน่งได้เพียง 4 ดวง นอกจากนั้นยังมีการแสดงค่าที่อ่านได้จากเครื่องวัดความเร่ง (ข้อมูล AX, AY และ AZ) และค่าที่วัดได้จากเครื่องวัดสนามแม่เหล็กโลก (ข้อมูล MX, MY และ MZ) นั้นไม่ได้นำมาใช้ในการทดลองครั้งนี้จึงกำหนดค่าเป็น 0.0 ทั้งหมด สำหรับภาพที่ 4.4 (ข) เปรียบเทียบสมาร์ทโฟนที่ใช้ในการทดลองกับระบบนำร่องภายในเรือ


(ก) ทดสอบภายนอกเรือ (ข) ภายในเรือ

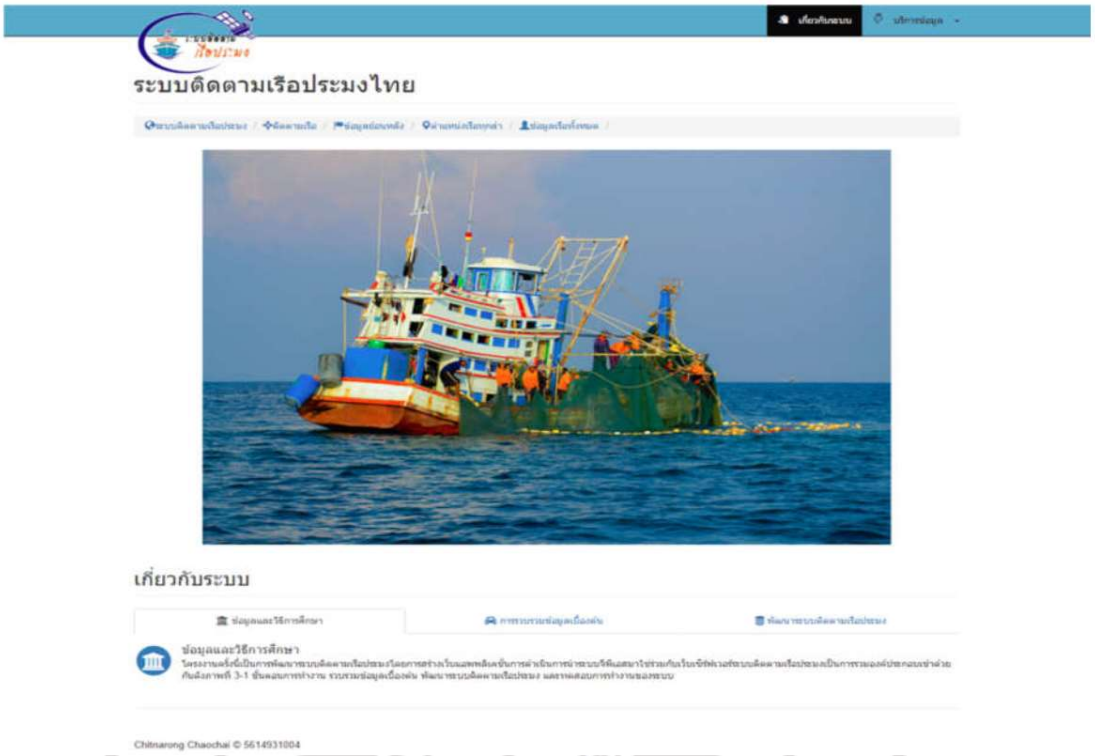
**ภาพที่ 4.4** การเก็บข้อมูลในทะเล

## 4.2 ผลการทดลอง

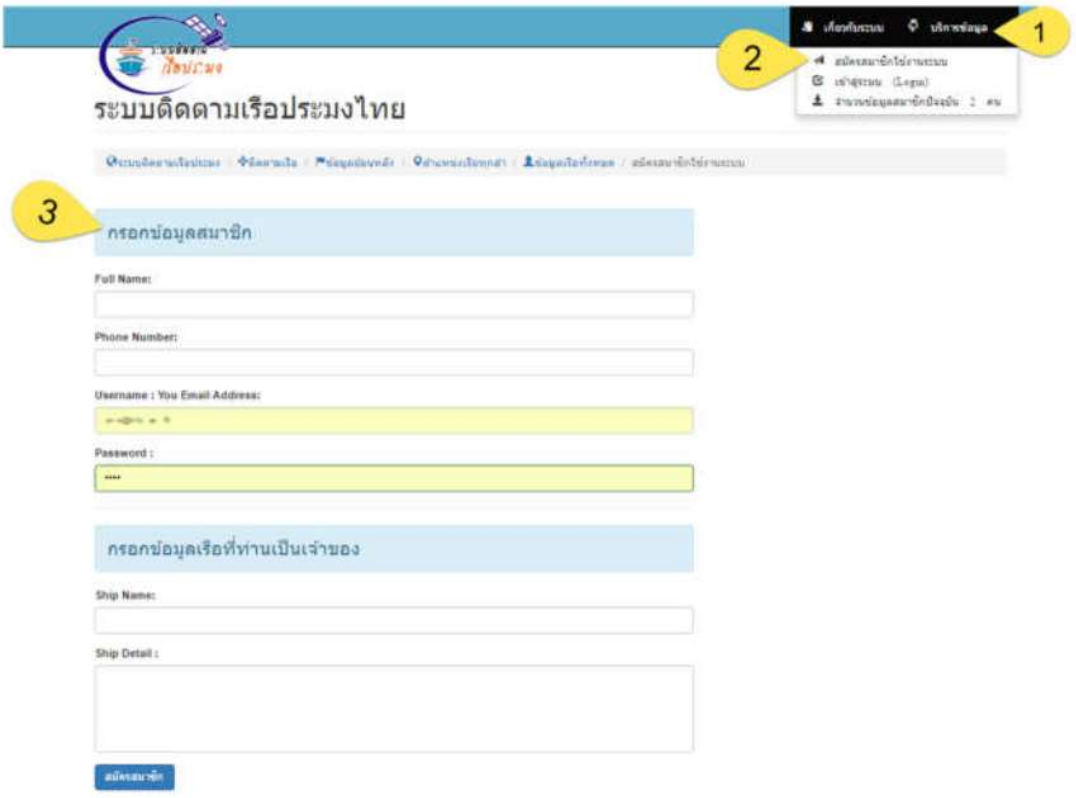
รายงานส่วนของผลการทดลองแบ่งได้เป็นสองส่วน ส่วนแรกเป็นผลการอิมพลิเมนต์และการทดสอบการทำงานของระบบจัดการผู้ใช้ และส่วนที่สองคือการแสดงความจริงเสริมบนสมาร์ทโฟน โดยมีรายละเอียดของแต่ละส่วนดังนี้

### 4.2.1 ระบบจัดการผู้ใช้

ภาพที่ 4.5 แสดงตัวอย่างหน้าหลักของส่วนผู้ดูแลระบบ โดยผู้ดูแลระบบสามารถเพิ่มข้อมูลสมาชิกได้โดยการเลือกเมนูบริการข้อมูล (หมายเลข 1) ตามด้วยการเลือกเมนูสมัครสมาชิกใช้งานระบบ (หมายเลข 2) จากนั้นระบบจะแสดงหน้าจอสำหรับการป้อนข้อมูลสมาชิกและกรอกข้อมูลเรือที่สมาชิกนั้นเป็นเจ้าของ (หมายเลข 3) โดยเมื่อป้อนข้อมูลและบันทึกข้อมูลเรียบร้อยแล้วระบบจะแจ้งผลกาารบันทึกข้อมูลให้ทราบ ดังตัวอย่างในภาพที่ 4.6

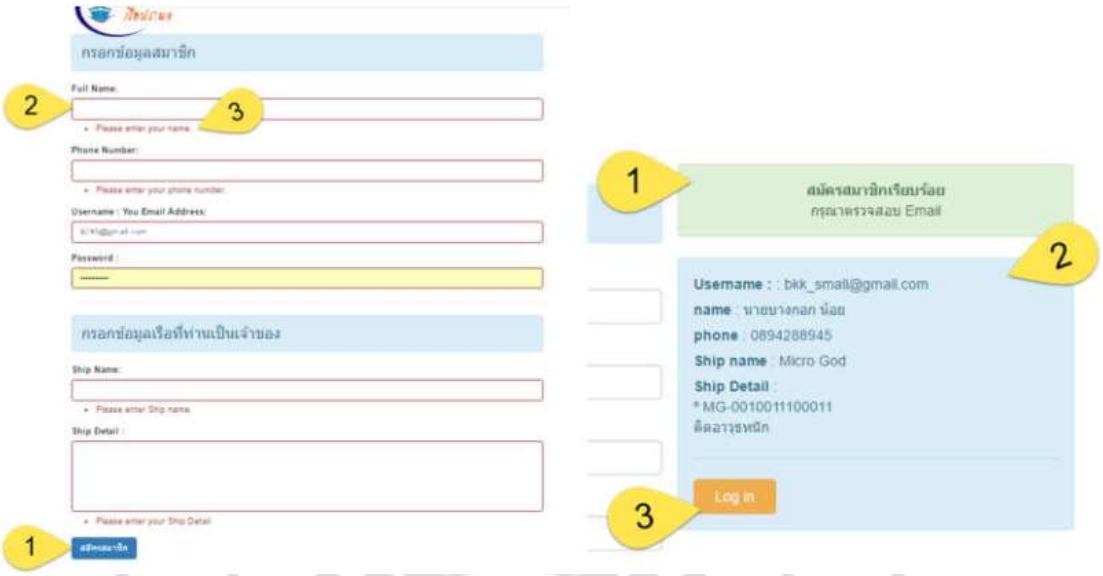



**ภาพที่ 4.5** หน้าแรก




**ภาพที่ 4.6** การสมัครสมาชิกใหม่

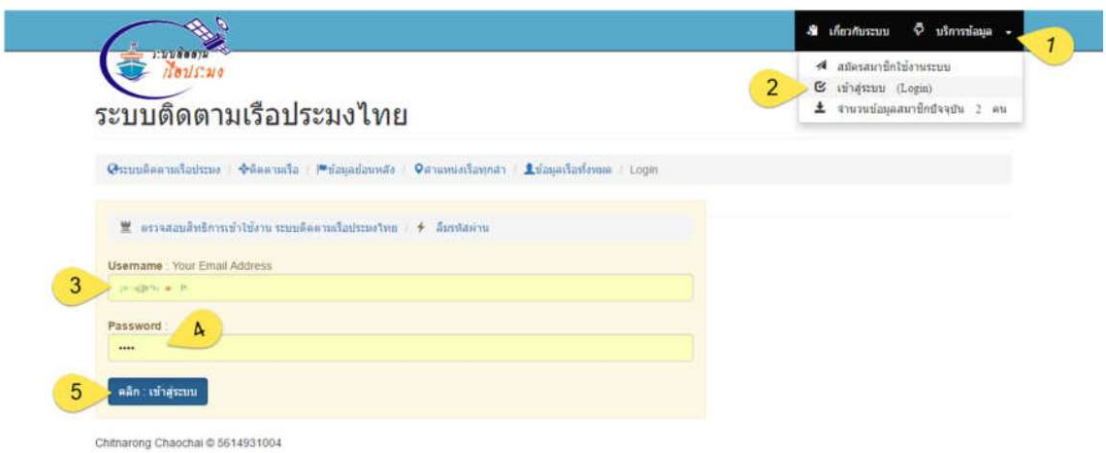
ภาพที่ 4.7 แสดงหน้าต่างแจ้งเตือนในกรณีที่การสมัครสมาชิกมีปัญหา




**ภาพที่ 4.7** การแจ้งเตือนในกรณีที่การสมัครสมาชิกมีปัญหา

1) การเข้าสู่ระบบ

ภาพที่ 4.8 แสดงหน้าต่างการเข้าสู่ระบบ โดยสมาชิกที่มีข้อมูลในระบบแล้ว สามารถเข้าสู่ระบบได้ด้วยการเลือกเมนูบริการข้อมูล (หมายเลข 1) ตามด้วยเมนูเข้าสู่ระบบ (หมายเลข 2) โดยระบบจะแสดงหน้าจอให้ผู้ใช้ป้อนชื่อผู้ใช้ (หมายเลข 3) และรหัสผ่าน (หมายเลข 4) หลังจากป้อนข้อมูลเสร็จเมื่อคลิกปุ่มเข้าสู่ระบบ (หมายเลข 5) แล้วระบบก็จะดำเนินการตรวจสอบสิทธิการเข้าใช้งาน



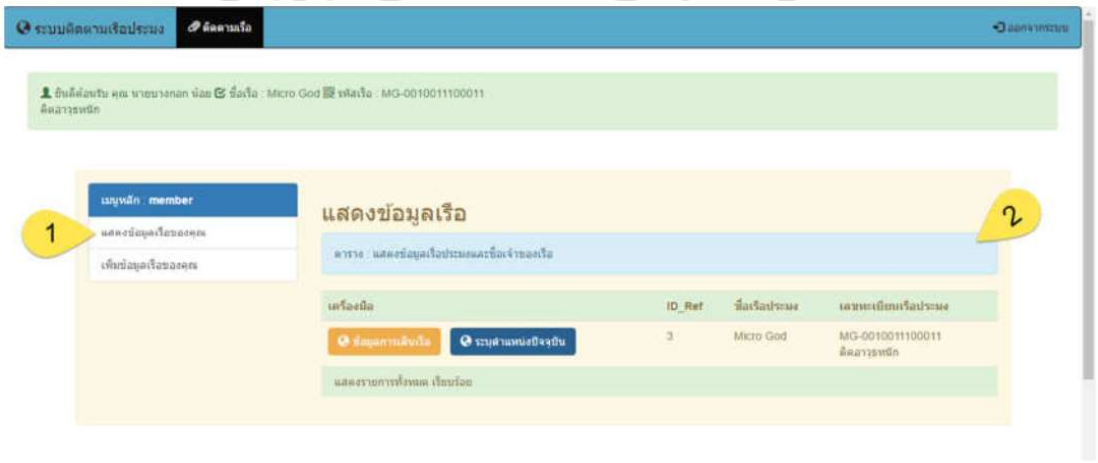

**ภาพที่ 4.8** การเข้าสู่ระบบ

หากเข้าสู่ระบบได้สำเร็จระบบจะแสดงหน้าต่างสำหรับจัดการข้อมูลต่างๆ ดังแสดงในภาพที่ 4.9 โดยหน้าต่างนี้ จะประกอบด้วยเมนูสำหรับการส่วนจัดการข้อมูลในการติดตามเรือ (หมายเลข 1) แถบสถานะแสดงข้อมูลข้อมูลผู้ใช้งานที่เข้าระบบพร้อมบอกข้อมูลเรือที่เป็นเจ้าของ (หมายเลข 2) และเมนูสำหรับจัดการข้อมูลและสถานะสิทธิการใช้งาน (หมายเลข 3) ผู้ใช้สามารถออกจากการใช้งานได้ด้วยการเลือกเมนูออกจากระบบ (หมายเลข 4)

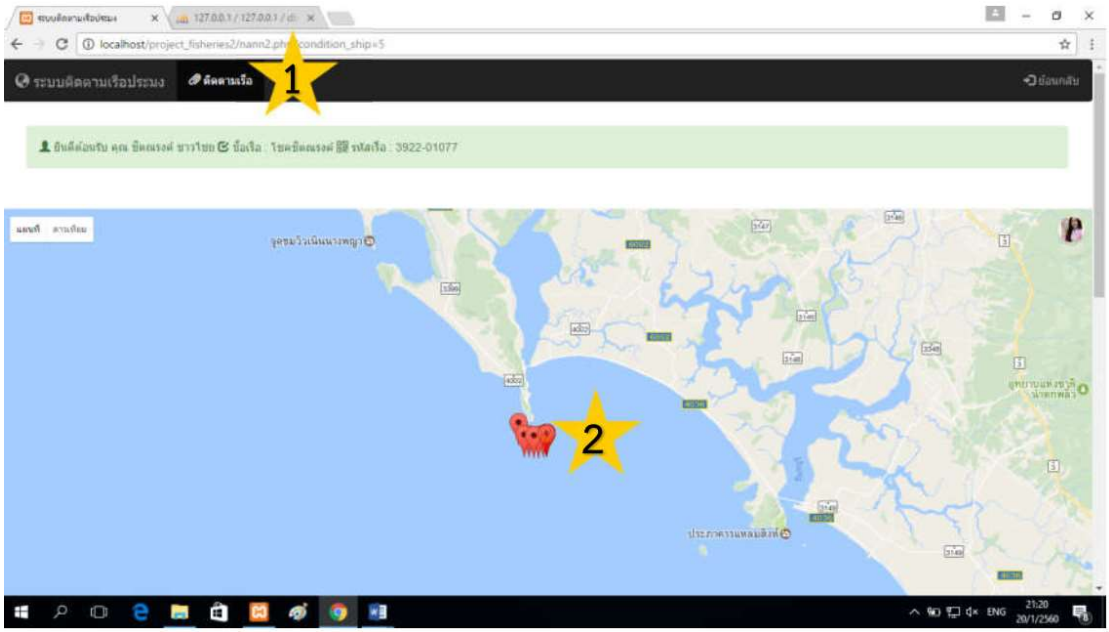

**ภาพที่ 4.9** หน้าหลักสำหรับผู้ใช้ที่เข้าสู่ระบบสำเร็จ

2) การจัดข้อมูลเรือ

ภาพที่ 4.10 แสดงหน้าต่างการจัดการข้อมูลเรือ ผู้ดูแลระบบสามารถจัดการข้อมูลสมาชิกได้ด้วยการเมนูเรือของคุณ (หมายเลข 1) โดยระบบจะแสดงข้อมูลเรือพร้อมทั้งแสดงรายละเอียดข้อมูลของเรือที่สมาชิกผู้ใช้เป็นเจ้าของให้ทราบ (หมายเลข 2) เมื่อผู้ใช้คลิกที่เมนูติดตามเรือ (หมายเลข 1) จะแสดงแผนที่การติดตามตำแหน่งเรือให้ผู้ใช้ทราบ โดยใช้สัญลักษณ์หมุดสีแดงแทนตำแหน่งเรือ ดังแสดงในภาพที่ 4.11




**ภาพที่ 4.10** การจัดการข้อมูลเรือ




**ภาพที่ 4.11** ตัวอย่างการติดตามตำแหน่งเรือ

### 4.2.2 ระบบส่งข้อมูลพิกัดเรือจากสมาร์ทโฟน

ในระหว่างการพัฒนาโปรแกรมเพื่อการส่งค่าพิกัดจากสมาร์ทโฟน ผู้วิจัยได้ทดสอบการทำงานของระบบส่วนนี้บนสมาร์ทโฟนซัมซุงกาแล็กซีเอสทรีที่ติดตั้งระบบปฏิบัติการ Android 4.0 โดยตารางที่ 4.11 แสดงตัวอย่างข้อมูลเซ็นเซอร์ที่มีในสมาร์ทโฟนรุ่นดังกล่าว

**ตารางที่ 4.2** รายการเซ็นเซอร์ที่ใช้ระหว่างการทดสอบการทำงาน

| ชื่อ | ผู้ผลิต | ประเภท | พิสัยสูงสุด | ความละเอียด | กำลัง (มิลลิแอมป์) | ค่าหน่วงเวลาน้อยสุด (ไมโครวินาที) |
|---|---|---|---|---|---|---|
| LSM330DLC 3-axis Accelerometer | STMicroelectronics | 1 | 19.6133 | 0.009576807 | 0.23 | 10000 |
| AK8975C 3-axis Magnetic field sensor | Asahi Kasei Microdevices | 2 | 2000.0 | 0.06 | 6.8 | 10000 |
| iNemoEngine Orientation sensor | STMicroelectronics | 3 | 360.0 | 0.015625 | 7.8 | 10000 |
| LSM330DLC Gyroscope sensor | STMicroelectronics | 4 | 8.726646 | 3.0543262E-4 | 6.1 | 5000 |

ตารางที่ 4.11 แสดงตัวอย่างข้อมูลบางส่วนที่บันทึกได้จากสมาร์ทโฟน (ในตารางมีการแปลงข้อมูลเวลาจากรูปแบบ `timestamp` ให้เป็น `datetime` เพื่อความสะดวกในการอ่านของผู้อ่าน) นอกจากนั้นระบบจะส่งข้อมูลจำนวนดาวเทียมที่ปรากฏและใช้ในการคำนวณไปให้ผู้ใช้ด้วย ซึ่งในงานวิจัยนี้จะยังไม่ได้นำค่านี้ไปใช้งาน แต่ได้มีการออกแบบข้อมูลไว้เพื่อการนำค่าเหล่านี้ไปวิเคราะห์เพื่อหาพื้นที่อับสัญญาณได้

**ตารางที่ 4.3** ข้อมูลตัวอย่างที่ได้จากสมาร์ทโฟน

| รหัส | ลองจิจูด | ละติจูด | วันและเวลา | จำนวนดาวเทียม | |
|---|---|---|---|---|---|
| | | | | ปรากฏ | ใช้งาน |
| 1036 | 12.21081433 | 101.95438507 | 2017-01-15 19:57:08 | 12 | 4 |
| 1086 | 12.21081432 | 101.95438507 | 2017-01-15 19:57:09 | 10 | 6 |
| 1137 | 12.21081428 | 101.95438511 | 2017-01-15 19:57:10 | 10 | 6 |
| 1190 | 12.21081406 | 101.95438523 | 2017-01-15 19:57:11 | 8 | 6 |
| 1289 | 12.21081407 | 101.95438520 | 2017-01-15 19:57:13 | 9 | 6 |
| 1340 | 12.21081505 | 101.95438480 | 2017-01-15 19:57:14 | 9 | 6 |
| 1391 | 12.21081505 | 101.95438485 | 2017-01-15 19:57:15 | 9 | 5 |
| 1441 | 12.21081478 | 101.95438480 | 2017-01-15 19:57:16 | 9 | 6 |
| 1492 | 12.21081471 | 101.95438503 | 2017-01-15 19:57:17 | 9 | 7 |
| 1543 | 12.21081470 | 101.95438499 | 2017-01-15 19:57:18 | 9 | 7 |

### 4.2.3 ระบบการแสดงความเป็นจริงเสมือนบนสมาร์ทโฟน

1) ข้อมูลจากเซ็นเซอร์

การรวมและกรองสัญญาณรบกวนด้วยตัวกรองความถี่ต่ำผ่านนั้นทำได้ตามแนวคิดที่ได้กล่าวมาแล้วในบทที่ 3 ซึ่งในงานวิจัยนี้จะมีการตรวจสอบให้แน่ใจว่าเซ็นเซอร์ที่เรียกเหตุการณ์นี้เป็นเซ็นเซอร์วัดความเร่งหรือไม่ ถ้าถูกต้องก็จะอ่านค่ามาบันทึกไว้แล้วจึงส่งค่าที่วัดได้ใหม่และค่าเดิมไปให้ตัวกรองด้วยเมท็อด `lowPassFilter3` ซึ่งผู้วิจัยได้แยกการอิมพลิเมนต์ออกไปไว้ในคลาส `MyUtil` และนำผลการคำนวณกลับมาบันทึก เสร็จแล้วจึงปรับปรุงเมทริกซ์การหมุนและเวกเตอร์การหมุนด้วยการเรียกเมธอด `updateOrientation` ดังแสดงในภาพที่ 4.12

การปรับค่าเมทริกซ์การหมุนและเวกเตอร์การหมุนให้เป็นปัจจุบันนั้นทำด้วยการส่งค่าความเร่งที่วัดได้ล่าสุดที่กรองสัญญาณรบกวนแล้ว ไปให้ระบบคำนวณเมทริกซ์การหมุน หากตรวจสอบพบว่าการคำนวณทำได้อย่างถูกต้องก็จะดำเนินการแปลงกรอบอ้างอิงให้เหมาะสมกับการใช้งานร่วมกับโอเพนจีเอล ในกรณีของงานวิจัยนี้กำหนดค่าพารามิเตอร์ค่าที่สองและสามของเมท็อด `remapCoordinateSystem` เป็น `AXIS_Z` และ `AXIS_MINUS_X` ตามลำดับ เสร็จแล้วจึงแปลงเป็นเวกเตอร์การหมุน `mTheta` ในหน่วยองศา ดังรายละเอียดในภาพที่ 4.13

```
@Override
public void onSensorChanged(SensorEvent event) {
  int et = event.sensor.getType();
  ...
  if (et == Sensor.TYPE_ACCELEROMETER) {
    // Record filtered data.
    this.mAccValue = MyUtil.lowPassFilter3(
            MyUtil.LOW_PASS_THRESHOLD_HARD,
            event.values,
            this.mAccValue);
  }
  if (et == Sensor.TYPE_MAGNETIC_FIELD) {
    // Record filtered data.
    this.mMagValue = MyUtil.lowPassFilter3(
            MyUtil.LOW_PASS_THRESHOLD_HARD,
            event.values,
            this.mMagValue);
  }
  // Update device orientation information.
  this.updateOrientation();
  ...
}
```

**ภาพที่ 4.12** การรวมและการกรองสัญญาณจากเซ็นเซอร์

```
private void updateOrientation() {
  ...
  // Calculate device orientation
  // using acceleration and magnetic field.
  boolean b = SensorManager.getRotationMatrix(
          this.mRp,       this.mI,
          this.mAccValue, this.mMagValue);

  // If calculation can be performed
  if(b == true) {
    // Transform coordinate reference frame
    SensorManager.remapCoordinateSystem(
        this.mRp, SensorManager.AXIS_Z,
        SensorManager.AXIS_MINUS_X, this.mR);

   // Update orientation angles
    SensorManager.getOrientation(this.mR, this.mTheta);
    this.mTheta[0] = (float)Math.toDegrees(this.mTheta[0]);
    this.mTheta[1] = (float)Math.toDegrees(this.mTheta[1]);
    this.mTheta[2] = (float)Math.toDegrees(this.mTheta[2]);
    ...
  }
  ...
}
```

**ภาพที่ 4.13** การปรับปรุงการคำนวณทิศทางการเอียงตัวของสมาร์ทโฟน

2) ภาพจากกล้อง

ภาพที่ 4.14 แสดงรายละเอียดชุดคำสั่งในเมท็อด surfaceChanged ของคลาส CameraView โดยโปรแกรมจะมีการตรวจสอบเบื้องต้นว่ามี surface สำหรับการแสดงผลถูกต้องหรือไม่ หากไม่มีจะสิ้นสุดการทำงาน จากนั้นจึงสั่งหยุดการแสดงผลชั่วคราวเพื่อนำค่าที่ส่งมามาปรับปรุงค่าของโปรแกรม โดยจะสั่งให้มีการคำนวณองศารับภาพ, กำหนดใช้ mHolder ที่ส่งมาเป็นตัวจัดการการแสดงภาพ, กำหนดทิศทางการเอียงตัวของอุปกรณ์ เสร็จแล้วจึงเริ่มการพรีวิวภาพทันที โดยรายละเอียดเกี่ยวกับข้อกำหนดและข้อจำกัดการแสดงภาพพรีวิวจากกล้องนั้น ผู้อ่านสามารถอ่านเพิ่มเติมได้จากเอกสาร Camera API (Google, 2018)

```
/**
 *
 */
@Override
public void surfaceChanged(SurfaceHolder surfaceHolder, int
format, int width, int height) {
  if(this.mHolder.getSurface() == null) { return; }

  try{ this.mCamera.stopPreview(); }
  catch (Exception e){
    ...
  }

  //now, recreate the camera preview
  try{
    // Update FOV
    this.updateFOV();

    // Set camera preview
    this.mCamera.setPreviewDisplay(this.mHolder);
    this.mCamera.setDisplayOrientation(0);
    this.mCamera.startPreview();
  } catch (IOException e) {
    ...
  }
}
```

**ภาพที่ 4.14** การปรับปรุงภาพจากกล้อง



3) การเตรียมการใช้งานโอเพนจีเอล

การเตรียมการใช้งาน นั้นทำตามขั้นตอนในหัวข้อ 2.3.3 (1) ซึ่งมีการกำหนดพารามิเตอร์ต่าง ๆ เพื่อให้เหมาะสมกับการแสดงภาพความจริงเสริมดังรายละเอียดในตารางที่ 4.4

**ตารางที่ 4.4** พารามิเตอร์ของการเตรียมการใช้งานโอเพนจีเอล

| ฟังก์ชัน | ความหมาย | ค่าที่กำหนดใช้ |
|---|---|---|
| glClearColor | สีสำหรับการล้างบัฟเฟอร์ | (0,0, 0,0, 0,0, 1.0) |
| glEnable | การเปิดการใช้งานคุณสมบัติของโอเพนจีเอล | GL_BLEND |
| glBlendFunc | ฟังก์ชันสำหรับการคำนวณการผสมสี | GL_SRC_ALPHA, GL_ONE_MINUS_SRC_ALPHA |
| glFrontFace | รูปแบบการคำนวณด้านหน้า (front face) ของรูปสามเหลี่ยม | GL_CCW |
| glDisable | การปิดการใช้งานคุณสมบัติของโอเพนจีเอล | GL_DEPTH_TEST |
| glLineWidth | ความกว้างของเส้นที่จะถูกวาด | 3.0 |

```
private final String vertex_shader_src =
    "attribute vec4 a_pos;  \n" +
    "attribute vec4 a_color;  \n" +
    "attribute vec2 a_texCoordinate; \n" +
    "varying vec4 v_color; \n" +
    "varying vec2 v_texCoordinate; \n" +
    "uniform mat4 u_MVPMatrix; \n" +
        "void main() { \n" +
        "  gl_Position = u_MVPMatrix * a_pos; \n" +
        "  v_color = a_color; \n" +
        "  v_texCoordinate = a_texCoordinate; \n" +
        "  gl_PointSize = 10.0; \n" +
        "}";
```

**ภาพที่ 4.15** เวอร์เท็กซ์เฉดเดอร์

ข้อมูลเวอร์เท็กซ์เฉดเดอร์ถูกเก็บอยู่ในตัวแปรสมาชิก vertex_shader_src และเฟรกเมนต์เฉดเดอร์เก็บอยู่ในตัวแปรสมาชิก fragment_shader_src ของคลาส MyOverlayCanvasRenderer โดยตัวแปรทั้งสองถูกกำหนดให้มีรูปแบบเป็น final เพื่อป้องกันการเปลี่ยนโดยไม่ได้ตั้งใจ ดังรายละเอียดในภาพที่ 4.15 และ ภาพที่ 4.16



```
private final String fragment_shader_src =
    "precision mediump float; \n" +
        "varying vec4 v_color; \n" +
        "uniform sampler2D u_Texture;  \n" +
        "varying vec2 v_texCoordinate; \n" +
        "void main() { \n" +
        "  gl_FragColor = v_color; \n" +
        "}";
```

**ภาพที่ 4.16** เฟรกเมนต์เฉดเดอร์

เนื่องจากงานวิจัยนี้ไม่ได้มีการใช้งานความสามารถทางกราฟิกส์ที่ซับซ้อนมาก ดังนั้น ผู้วิจัยจึงออกแบบให้เฉดเดอร์ทั้งสองทำหน้าที่เป็นเฉดเดอร์ส่งผ่าน (pass-through shader) เท่านั้น เฉดเดอร์ทั้งสองเฉดเดอร์ไม่ได้มีการคำนวณเพื่อเปลี่ยนแปลงเวอร์เท็กซ์หรือเฟรกเมนต์ใด ๆ ยกเว้นแต่ในส่วนของเวอร์เท็กซ์เฉดเดอร์ที่มีการกำหนดค่า `gl_PointSize` ให้เป็น 10.0 เนื่องจากโอเพนจีเอลอีเอสรุ่นที่ใช้ในการพัฒนายังไม่รอบรับการกำหนดขนาดของจุดจากโปรแกรมได้โดยตรง

4) การแสดงรายละเอียด

เมื่อจำเป็นต้องวาดหน้าจอ เช่น มีข้อมูลส่งมาจากเครือข่ายเพื่อปรับปรุงพิกัดเรือหรือมีการเปลี่ยนแปลงภาพจากกล้องนั้น โปรแกรมจะล้างข้อมูลในบัฟเฟอร์ด้วยเมท็อด `glClear` พร้อมกับส่งค่า `GL_COLOR_BUFFER_BIT` และ `GL_DEPTH_BUFFER_BIT` เพื่อกำหนดให้ล้างทั้งข้อมูลในบัฟเฟอร์สีและบัฟเฟอร์ความลึก หลังจากนั้นจึงตรวจสอบสถานะการแสดงผลในตัวแปร `mRenderMode` ว่าต้องการแสดงผลแบบใด โดยในสถานะปกติจะมีค่าเป็น `RENDER_MODE_AR` ซึ่งจะทำให้โปรแกรมซ้อนทับข้อมูลเรือด้วยการป้ายตำแหน่งเรือ แต่หากมีการแตะที่ป้ายชื่อจะทำให้ตัวแปร `mRenderMode` เปลี่ยนค่าเป็น `RENDER_MODE_ MODEL` ซึ่งหมายถึงการแสดงผลเป็นแบบจำลองเรือสามมิติ ดังรายละเอียดในภาพที่ 4.17

```
/**
 * Main rendering process.
 *
 * @param unused
 */
public void onDrawFrame(GL10 unused) {
  // Redraw background color
  GLES20.glClear(GLES20.GL_COLOR_BUFFER_BIT |
                 GLES20.GL_DEPTH_BUFFER_BIT);

  // Start drawing elements.
  switch(this.mRenderMode) {
    case MyOverlayCanvasRenderer.RENDER_MODE_AR:
      this.renderAR();
      break;
    case MyOverlayCanvasRenderer.RENDER_MODE_MODEL:
      this.render3DModel();
      break;
  }
}
```



**ภาพที่ 4.17** การอิมพลิเมนต์เมท็อด `onDrawFrame`

ในส่วนของการเรนเดอร์ภาพพื้นหลังในเมท็อด `renderWorld` นั้น โปรแกรมจะเริ่มต้นด้วยการรีเซตเมทริกซ์ที่เกี่ยวข้องให้เป็นเมทริกซ์เอกลักษณ์ด้วยการใช้ฟังก์ชัน `setIdentity` จากคลาส `Matrix` แล้วจึงกำหนดเมทริกซ์การแปลงให้เหมาะสมกับข้อมูลปัจจุบันด้วยการเรียกเมท็อด `set3DProjection` โดยส่งค่าองศารับภาพ อัตราส่วนหน้าจอ และระยะระนาบใกล้/ไกลไปให้คำนวณ หลังจากนั้นจึงอ่านค่าทิศทางจากตัวจัดการเซ็นเซอร์มาเก็บใน

ตัวแปร theta แล้วคำนวณเมทริกซ์การแปลงตามรายละเอียดในหัวข้อ 2.3.3 (6) ซึ่งในการคำนวณเมทริกซ์การหมุนและการคูณเมทริกซ์นั้นทำด้วยเมท็อด rotateM และ multiplyMM ของคลาส Matrix ดังรายละเอียดในภาพที่ 4.18

```
1    private void renderWorld() {
2      // Reset OpenGL matrices
3      Matrix.setIdentityM(this.gl_mat_mvp, 0);
4      ...
5
6      // Setup perspective projection matrix
7      this.set3DProjection(
8          this.mCameraFOV,   this.mScreenAspect,
9          this.mCameraZNear, this.mCameraZFar);
10
11     // Get camera orientation
12     float[] theta = this.mSensors.getOrientationDeg();
13
14     // Setup camera
15     Matrix.rotateM(this.gl_mat_Rx,0,
16                    theta[1], 1.0f,0.0f,0.0f);
17     Matrix.rotateM(this.gl_mat_Ry,0,
18                    theta[0],0.0f,1.0f,0.0f);
19     Matrix.rotateM(this.gl_mat_Rz,0,
20                    theta[2],0.0f,0.0f,1.0f);
21
22     Matrix.multiplyMM(this.mGLMatMV, 0,
23                       this.gl_mat_Rx, 0,
24                       this.gl_mat_Ry, 0);
25
26     this.gl_mat_temp = Arrays.copyOf(this.mGLMatMV,
27                                      this.mGLMatMV.length);
28     Matrix.multiplyMM(this.mGLMatMV, 0,
29                       this.gl_mat_Rz, 0,
30                       this.gl_mat_temp, 0);
31
32     // M = M_proj * M_modelview
33     Matrix.multiplyMM(this.gl_mat_mvp, 0,
34                       this.gl_mat_proj, 0,
35                       this.mGLMatMV, 0);
36    // Pass the matrix to shader
37     GLES20.glUniformMatrix4fv(this.gl_mat_mvp_loc, 1,
38                               false, this.gl_mat_mvp, 0);
39
40     // Render world objects
41     for(MyRenderableObject obj: this.mWorldObjects) {
42       obj.render();
43     }
44   }
```

**ภาพที่ 4.18** การอิมพลิเมนต์เมท็อด renderWorld

5) การคำนวณบิลบอร์ด

ชุดคำสั่งในภาพที่ 4.19 แสดงการอิมพลิเมนต์การแสดงป้ายตำแหน่งเรือ โดยบรรทัดที่ 5 - 12 เป็นการสร้างตัวแปรสำหรับการคำนวณ จากนั้นจึงเป็นการอ่านค่าทิศทางปัจจุบันของสมาร์ทโฟน ในบรรทัดที่ 18 - 19 เป็นการคำนวณเวกเตอร์ทิศทางกล้องบนระนาบพื้นซึ่งทำด้วยการแปลงค่ามุมในตัวแปร **a** ให้เป็นเวกเตอร์ทิศทาง `cam_dir` ต่อมาคำสั่งในบรรทัดที่ 21 - 29 จึงเป็นการปรับเมทริกซ์ต่าง ๆ ให้เป็นเมทริกซ์เอกลักษณ์แล้วหมุนรอบแกน **X**, **Y** และ **Z** ด้วยค่ามุม $-a_Y$, $a_X$ และ $-a_Z$ ตามลำดับ การคำนวณเมทริกซ์ดังกล่าวทำด้วยด้วยเมท็อด `rotateM` และ `multiplyMM` ของคลาส `Matrix` เช่นเดียวกับก่อนหน้านี้

การวนซ้ำหลักเพื่อคำนวณพิกัดของมุมของบิลบอร์ดเกินขึ้นระหว่างบรรทัดที่ 42 - 87 โดยจะมีการสกัดพิกัดปัจจุบันของเรือแต่ละลำ เสร็จแล้วคำนวณมุมระหว่างเวกเตอร์ทิศทาง `cam_dir` ของกล้องกับตำแหน่งเรือจาก

$$\mathbf{t}(\mathbf{c}, \mathbf{s}) = \cos^{-1}(\text{dot}(\mathbf{c}, \mathbf{s})/(\|\mathbf{c}\|\times\|\mathbf{s}\|)) \qquad 4.1$$

เมื่อ **s** คือพิกัดเรือแบบสัมพัทธ์, **c** คือเวกเตอร์ทิศทางของสมาร์ทโฟน และ **t**(**c**, **s**) คือมุมระหว่างเวกเตอร์ทั้งสอง

ถ้ามุมระหว่าง **c** และ **s** มีค่าเกินค่าขีดแบ่ง $\theta_{FOV}$ แสดงว่าเรือลำนั้นอยู่นอกองศารับภาพที่กำหนดไว้ โปรแกรมก็จะข้ามไปการคำนวณยังเรือลำอื่น แต่ถ้าหากพิกัดเรืออยู่ในองศารับภาพแล้วก็จะทำการฉายพิกัดของเรือลงบนระนาบภาพด้วยชุดคำสั่งระหว่างบรรทัดที่ 57 - 76 โดยในชุดคำสั่งได้มีการกระจายการคูณเมทริกซ์การแปลงกับพิกัดของวัตถุเพื่อความสะดวกในการสลับค่าพิกัดระหว่างกรอบอ้างอิงปกติและกรอบอ้างอิงของโอเพนจีเอล เสร็จแล้วจึงเรียกให้มีการปรับพิกัดของป้ายชื่อเรือและเรียกเมท็อด `render` ของวัตถุเรือแต่ละลำเพื่อดำเนินการวาดภาพต่อไป

```
1    /**
2     *
3     */
4    private synchronized void renderTrackedObjects() {
5      float[] tmp = new float[16];
6      float[] Rx = new float[16];
7      float[] Ry = new float[16];
8      float[] Rz = new float[16];
9      float[] M = new float[16];
10     float[] MP = new float[16];
11     float[] cam_dir = new float[2];
12     float[] p = new float[2];
13
14     // Get camera orientation
15     float[] a = this.mSensors.getOrientationDeg();
16
17     // Camera direction vector
18     cam_dir[0] = (float)Math.cos(a[0]*MyUtil.DEG_TO_RAD);
19     cam_dir[1] = (float)Math.sin(a[0]*MyUtil.DEG_TO_RAD);
20
21     Matrix.setIdentityM(M,  0);
22     Matrix.setIdentityM(MP, 0);
23     Matrix.setIdentityM(Rx, 0);
```

```
24      Matrix.setIdentityM(Ry, 0);
25      Matrix.setIdentityM(Rz, 0);
26
27      Matrix.rotateM(Rx, 0, -a[1], 1.0f, 0.0f, 0.0f);
28      Matrix.rotateM(Ry, 0,  a[0], 0.0f, 1.0f, 0.0f);
29      Matrix.rotateM(Rz, 0, -a[2], 0.0f, 0.0f, 1.0f);
30
31      Matrix.multiplyMM(M, 0, Rx, 0, Ry, 0);
32      this.gl_mat_temp = Arrays.copyOf(M, M.length);
33      Matrix.multiplyMM(M, 0, Rz, 0, this.gl_mat_temp, 0);
34      Matrix.multiplyMM( MP, 0, this.gl_mat_proj, 0, M, 0);
35
36      float[] X = new float[4];
37      float[] Xp = new float[4];
38
39      // Render each tracked object
40      float t;
41      int i = 0;
42      for(MyTrackedObject obj: this.mTrackedObjects) {
43        // Get coordinates
44        X[0] = obj.getWorldCoords()[0];
45        X[1] = obj.getWorldCoords()[1];
46        X[2] = obj.getWorldCoords()[2];
47        X[3] = 1.0f;
48
49        // Position
50        p[0] = X[0];
51        p[1] = X[2];
52        t = MyUtil.angleTwoVector(cam_dir, p);
53
54        if(t > this.mCameraFOV) {
55          ...
56        } else {
57          // x = MX
58          Xp[0] = (X[0]*MP[ 0]) + (X[1]*MP[ 4])
59                + (X[2]*MP[ 8]) + (X[3]*MP[12]);
60          Xp[1] = (X[0]*MP[ 1]) + (X[1]*MP[ 5])
61                + (X[2]*MP[ 9]) + (X[3]*MP[13]);
62          Xp[2] = (X[0]*MP[ 2]) + (X[1]*MP[ 6])
63                + (X[2]*MP[10]) + (X[3]*MP[14]);
64          Xp[3] = (X[0]*MP[ 3]) + (X[1]*MP[ 7])
65                + (X[2]*MP[11]) + (X[3]*MP[15]);
66
67          // Normalize
68          Xp[0] /= Xp[3];     Xp[1] /= Xp[3];
69          Xp[2] /= Xp[3];     Xp[3] /= Xp[3];
70
71          // Convert to screen coordinates
72          Xp[0] = (Xp[0]+1.0f)/2.0f;
73          Xp[1] = (Xp[1]+1.0f)/2.0f;
74          Xp[0] = Xp[0] * this.mScreenWidth;
75          Xp[1] = Xp[1] * this.mScreenHeight;
76          Xp[1] = Xp[1] - 130.0f;
```

```
77        }
78
79        ((FullscreenActivity)this.mContext).updateLabelPos(
80            i, Xp[0], Xp[1]);
81
82        // Draw object
83        obj.render();
84
85        i++;
86      }
87    }
```

**ภาพที่ 4.19** การอิมพลิเมนต์เมท็อด renderTrackedObjects

เมื่อมีการอ่านข้อมูลเรือมาได้จะมีการเรียกเมท็อด updateTrackedObjects ให้ทำงาน โดยโปรแกรมหลักจะส่งรายการข้อมูลเรือมาเป็นอาร์กิวเมนต์ ภายในเมท็อดนี้จะอ่านค่าตำแหน่งและทิศทางปัจจุบันของสมาร์ทโฟนมาเก็บไว้ เสร็จแล้วจึงเริ่มต้นปรับปรุงข้อมูลเรือแต่ละลำ โดยการคำนวณพิกัดสัมพัทธ์และระยะทางของเรือเทียบกับสมาร์ทโฟน เมื่อได้ข้อมูลเบื้องต้นแล้วจึงเตรียมบัฟเฟอร์สำหรับบันทึกค่าพิกัด vert, ค่าสี colors และค่าดัชนี ind ที่จำเป็นสำหรับการแสดงผลด้วยโอเพนจีเอล เสร็จแล้วจึงสร้างวัตถุของคลาส MyTrackedObject ขึ้นมา เมื่อกำหนดค่าของวัตถุใหม่นี้เรียบร้อยแล้วจึงเพิ่มวัตถุนี้เข้าไปในรายการเรือที่ต้องการต่อไป ดังรายละเอียดในภาพที่ 4.20

```
1   /**
2    *
3    */
4   public synchronized void
5   updateTrackedObjects(ArrayList<MyShipInfo> s) {
6     ...
7
8     // Get current camera orientation
9     float[] theta = this.mSensors.getOrientationDeg();
10
11    if(theta[0] > 0.0f) { t=theta[0] * MyUtil.DEG_TO_RAD; }
12    else { t = -theta[0] * MyUtil.DEG_TO_RAD; }
13
14    ct = (float)Math.cos(t);
15    st = (float)Math.sin(t);
16
17    try {
18      if (this.mTrackedObjects.size() > 0) {
19        this.mTrackedObjects.clear();
20      }
21
22      // Current GPS coordinates
23      float x0 = (float)this.mGPSPosUTM[0];
24      float z0 = (float)this.mGPSPosUTM[1];
25
```

```
26        // Calculate vertices coordinates
27        int i;
28        for (i = 0; i < s.size(); i++) {
29          MyShipInfo si = s.get(i);
30          // Calculate relative position to X0
31          ax = si.mCoords[0] - x0;
32          az = si.mCoords[1] - z0;
33          // Distance to camera
34          d = (float)Math.sqrt(ax*ax + az*az);
35          sf = d/10000.0f;
36
37          if(sf <= 1.0f) { alpha = sf; }
38          else { alpha = 1.0f; }
39          alpha = 1.0f - alpha;
40          // Billboard
41          x1 = -sw * sf;   z1 = 0.0f;
42          x2 =  sw * sf;   z2 = 0.0f;
43
44          // x' = Rx + t
45          x1p = ((x1 *  ct) + (z1 * st)) +   ax;
46          z1p = ((x1 * -st) + (z1 * ct)) + (-az);
47          x2p = ((x2 *  ct) + (z2 * st)) +   ax;
48          z2p = ((x2 * -st) + (z2 * ct)) + (-az);
49
50          float[] vert = new float[]{
51              ax,   0.0f, -az,
52              x1p, sw*sf,  z1p,
53              x2p, sw*sf,  z2p};
54          short[] ind = new short[]{ 0, 1, 2 };
55          float[] colors = new float[] {
56              1.0f, 0.0f, 0.0f, 0.8f,
57              1.0f, 0.0f, 0.0f, 0.8f,
58              1.0f, 0.0f, 0.0f, 0.8f,
59          };
60          MyTrackedObject mto = new MyTrackedObject();
61          mto.setShaderVertexLoc(this.gl_vert_loc);
62          mto.setShaderDefaultVertColorLoc(
63              this.gl_vert_color_loc);
64          mto.setPrimitiveType(GLES20.GL_TRIANGLE_STRIP);
65          mto.setVertexData( vert );
66          mto.setIndexData( ind );
67          mto.setColorData( colors );
68          mto.setCoordinates(ax, 0.0f, -az);
69          mto.setDefaultColor(1.0f, 0.0f, 0.0f, alpha);
70          this.mTrackedObjects.add(mto);
71        }
72      } catch ( Exception e ) {
73        ...
74      }
75    }
```

**ภาพที่ 4.20** การอิมพลิเมนต์เมท็อด `updateTrackedObjects`

การอิมพลิเมนต์การเรนเดอร์เครื่องหมายชี้ตำแหน่งเรือนั้นทำด้วยเมท็อด render โดยเริ่มต้นจะมีการเปิดใช้งานเวอร์เท็กซ์อาเรย์ด้วยฟังก์ชัน glEnableVertexAttribArray แล้วจึงกำหนดตัวชี้ไปยังข้อมูลพิกัดด้วยฟังก์ชัน glVertexAttribPointer โดยส่งข้อมูลที่อยู่ของบัฟเฟอร์ข้อมูลเวอร์เท็กซ์ จำนวนข้อมูลต่อหนึ่งเวอร์เท็กซ์, ชนิดตัวแปรและขนาดข้อมูลทั้งหมดตามลำดับ โดยการกำหนดบัฟเฟอร์ค่าสีทำด้วยแนวทางเดียวกัน หลังจากนั้นจึงกำหนดตัวชี้ของข้อมูลดัชนี เสร็จแล้วจึงเริ่มการวาดด้วยการเรียกใช้เมท็อด glDrawElements โดยส่งข้อมูลประเภทรูปทรงที่ต้องการวาดที่อยู่ในตัวแปร mPrimitiveType (ในที่นี้คือ GL_TRIANGLES) ความยาวข้อมูล ชนิดข้อมูลมีค่าเป็น GL_UNSIGNED_SHORT ซึ่งหมายถึงข้อมูลแบบ unsigned int แบบสองไบต์ และบัฟเฟอร์ที่เก็บข้อมูลที่ต้องการนำไปเรนเดอร์ เสร็จแล้วจึงปิดการใช้งานเวอร์เท็กซอาเรย์ด้วยฟังก์ชัน glDisableVertexAttribArray ดังรายละเอียดในภาพที่ 4.20

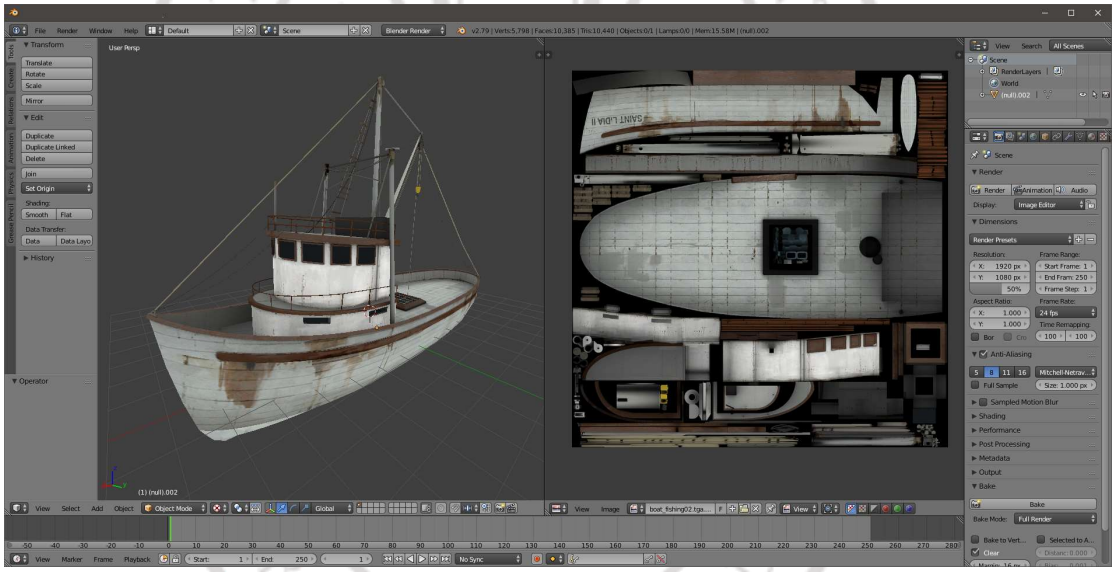
```
1    /**
2     *
3     */
4    public void render() {
5      // Enable vertex attribute array
6      GLES20.glEnableVertexAttribArray(this.mGLVertexLoc);
7      GLES20.glVertexAttribPointer(
8          this.mGLVertexLoc, MyUtil.NUM_COORDS_PER_VERTEX,
9          GLES20.GL_FLOAT, false,
10         MyUtil.NUM_COORDS_PER_VERTEX*MyUtil.SIZE_OF_FLOAT,
11         this.mVertexBuffer);
12
13     // Set vertex color
14     GLES20.glVertexAttribPointer(
15         this.mGLColorLoc, MyUtil.NUM_COLORS_PER_VERTEX,
16         GLES20.GL_FLOAT, false, 0, this.mColorBuffer);
17     GLES20.glEnableVertexAttribArray(this.mGLColorLoc);
18
19     // Set index position
20     this.mIndexBufffer.position(this.mIndexPosition);
21
22     // Draw elements
23     GLES20.glDrawElements(
24         this.mPrimitiveType, this.mIndices.length,
25         GLES20.GL_UNSIGNED_SHORT, this.mIndexBufffer);
26
27     // Disable vertex array
28     GLES20.glDisableVertexAttribArray(this.mGLVertexLoc);
29   }
```



**ภาพที่ 4.21** การอิมพลิเมนต์เมท็อด renderWorld

### 4.2.4 การแสดงแบบจำลองสามมิติ

ภาพที่ 4.21 แสดงตัวอย่างแบบจำลองเรือสามมิติ ซึ่งผู้วิจัยใช้ข้อมูลแบบจำลองเรือสามมิติจากเว็บไซต์ให้บริการแบบจำลองสามมิติแคดนาฟ (CadNav)[1] เปิดให้ผู้ใช้ดาวน์โหลดได้ฟรี โดยในตัวอย่างนี้ใช้แบบจำลองหมายเลข 41490 ซึ่งไม่ปรากฏรายละเอียดเกี่ยวกับผู้สร้างแบบจำลอง แฟ้มนี้เป็นแบบจำลองเรือประมงมีข้อมูลต้นฉบับเป็นแฟ้มโอบีเจ มีจำนวนเวอร์เท็กซ์ 31,320 จุด และมีจำนวนรูปสามเหลี่ยมทั้งหมด 10,440 รูป และมีแฟ้มภาพลายผิวต้นฉบับเป็นทีจีเอ (Truevision TGA หรือ TARGA) แบบ 24 บิตต่อพิกเซล โดยแบบจำลองนี้มีขนาดแฟ้มโอบีเจ, ทีจีเอ และเอ็มทีแอล (MTL) เท่ากับ 989280 ไบต์, 4194854 ไบต์ และ 243 ไบต์ ตามลำดับ ซึ่งรวมแล้วแบบจำลองนี้มีขนาดเท่ากับ 5184637 ไบต์ ภาพที่ 4.23 แสดงแบบจำลองต้นฉบับเมื่อเปิดดูในโปรแกรมประยุกต์ด้านคอมพิวเตอร์กราฟิกส์สามมิติเบลนด์เดอร์ (Blender)[2]



**ภาพที่ 4.22** แบบจำลองเรือประมงเปิดในโปรแกรมเบลนด์เดอร์

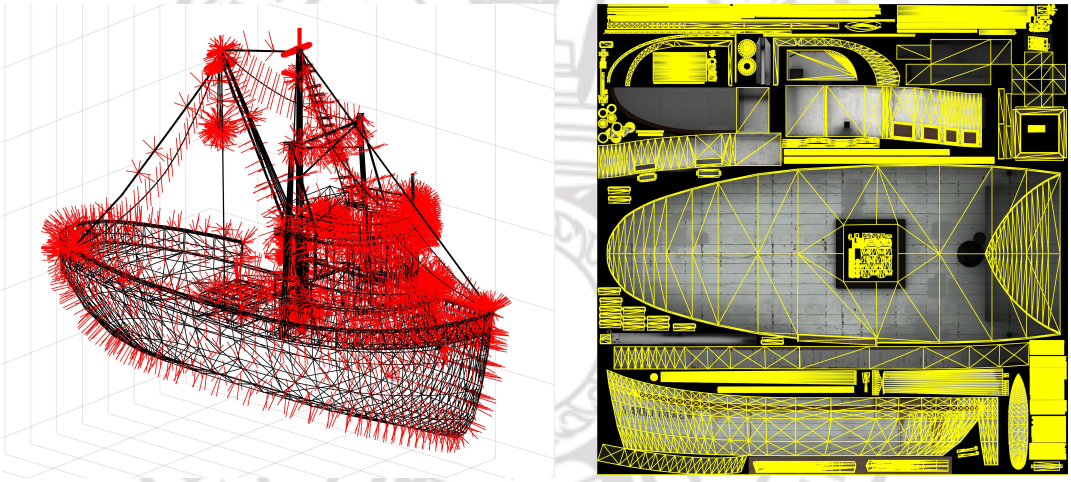
ผู้วิจัยพัฒนาคลังโปรแกรมเพื่อแปลงแบบจำลองให้เป็นแฟ้มเอ็มโอดีตามรายละเอียดในบทที่ 3 แบบจำลองที่แปลงได้มีจำนวนเวอร์เท็กซ์เท่ากับ 31320 จุด และมีจำนวนรูปสามเหลี่ยมทั้งหมด 10440 รูป ผลการแปลงได้แฟ้มเอ็มโอดีจำนวนหนึ่งแฟ้มซึ่งรวมข้อมูลที่จำเป็นทุกอย่างแล้วมีขนาดเท่ากับ 5024960 ไบต์ ซึ่งจะเห็นว่าขนาดของแฟ้มเอ็มโอดีเทียบกับแฟ้มโอบีเจนั้นไม่ค่อยมีความแตกต่างกันนัก อาจจะเนื่องมาจากจำนวนเวอร์เท็กซ์หรือจำนวนรูปสามเหลี่ยมของแบบจำลองนั้นมีจำนวนไม่สูงมาก แต่ถ้าตามหากแบบจำลองมีรายละเอียดที่สูงมากกว่านี้ก็อาจจะทำให้ขนาดแฟ้มเอ็มโอดีมีขนาดใหญ่มากกว่าก็เป็นได้ เนื่องจากผู้วิจัยออกแบบแฟ้มเอ็มโอดีโดยมีการกระจายข้อมูลเวอร์เท็กซ์ของรูปสามเหลี่ยมแต่ละรูปออกมาจากกัน ต่างกับแฟ้มโอบีเจที่ใช้แนวคิดการเชื่อมโยงเวอร์เท็กซ์ทำให้ลดขนาดของแฟ้มได้ดีกว่า อย่างไรก็ตามการแสดงความจริงเสริมนั้นมักจะใช้

1 http://www.cadnav.com/

2 https://www.blender.org/

แบบจำลองที่มีรายละเอียดปานกลางหรือรายละเอียดต่ำที่เรียกว่าโลว์โพลีกอนโมเดล (low-polygon model) เพื่อลดรายละเอียดที่ไม่จำเป็นออกไป จึงทำให้ความแตกต่างนี้ไม่มีผลการทำงาน

ภาพที่ 4.23 (ก) แสดงตัวอย่างข้อมูลแบบจำลองสามมิติ ซึ่งแสดงรูปสามเหลี่ยมแต่ละรูปด้วยเส้นสีดำและแสดงเวกเตอร์ปกติด้วยเส้นสีแดง และภาพที่ 4.23 (ข) แสดงตัวอย่างการกำหนดพิกัดภาพลายผิวที่ใช้ เส้นสีเหลืองในภาพแทนตำแหน่งของรูปสามเหลี่ยมแต่ละรูป ซ้อนทับบนภาพลายผิวที่ใช้ เมื่อผู้ใช้แตะที่ป้ายชื่อของเรือ โปรแกรมจะตรวจสอบและแสดงผลแบบจำลองสามมิติโดยงานวิจัยนี้กำหนดให้จำลองหมุนรอบแกน $\mathbf{Y}$ อย่างช้า ๆ เพื่อให้ผู้ใช้ได้เห็นข้อมูลของเรือประมงได้รอบลำ เมื่อผู้ใช้แตะที่หน้าจอจะทำให้โปรแกรมกลับไปทำงานในโหมดแสดงผลแบบความจริงเสริมตามปกติ



(ก) (ข)

**ภาพที่ 4.23** ข้อมูลเวอร์เท็กซ์และเวกเตอร์ปกติของแบบจำลองเรือ