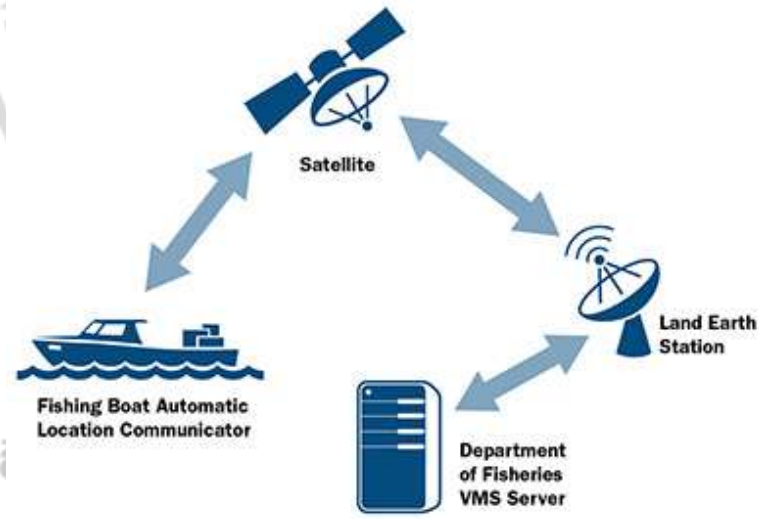
บทที่ 3
วิธีการดำเนินงานวิจัย

โครงการวิจัยนี้เป็นงานวิจัยเชิงพัฒนาเพื่อพัฒนาระบบที่ช่วยในการระบุตำแหน่งเรือประมงโดยใช้สมาร์ทโฟนและเทคโนโลยีตวามเป็นจริงเสริมเป็นตัวช่วยในการแสดงผลกราฟิกส์ ผู้วิจัยได้สรุปวิธีการดำเนินงานที่สำคัญได้ทั้งหมดสองส่วน ประกอบด้วย การออกแบบระบบส่วนที่ทำหน้าที่ติดตามและบันทึกข้อมูลตำแหน่งเรือ และส่วนที่สองคือการออกแบบส่วนแสดงผลตวามเป็นจริงเสริม ดังรายละเอียดต่อไปนี้

3.1 การออกแบบระบบติดตามและบันทึกตำแหน่งเรือ

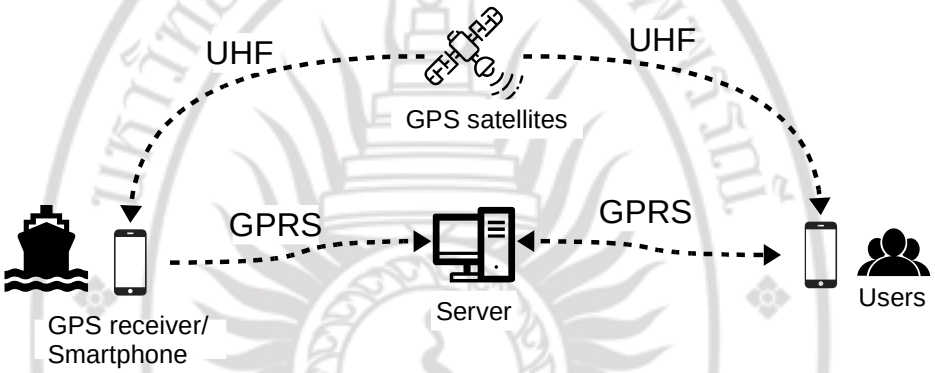
โดยปกติระบบการสื่อสารที่เป็นที่นิยมสำหรับระบบติดตามเรือได้แก่การใช้การสื่อสารผ่านดาวเทียมดังแสดงในภาพที่ 3.1 เนื่องจากระบบดังกล่าวจะครอบคลุมพื้นที่ได้กว้างกว่าระบบอื่น และมีประสิทธิภาพเหมาะสมสำหรับการประยุกต์กับการติดตามตามเรือเดินทะเล อย่างไรก็ตามระบบผ่านดาวเทียมยังมีข้อจำกัดสำคัญคือเรื่องของต้นทุน นอกจากนั้นระบบบางส่วนยังเป็นระบบเชิงพาณิชย์ซึ่งโปรแกรมส่วนใหญ่จะเป็นระบบปิด ผู้ใช้ไม่สามารถเข้าถึงหรือนำข้อมูลจากระบบไปพัฒนาต่อยอดได้โดยง่าย ข้อจำกัดเหล่านี้จึงอาจจะเป็นอุปสรรคสำหรับการพัฒนาและนำไปใช้จริงในระบบประมงชายฝั่งของประเทศไทย



ภาพที่ 3.1 ระบบติดตามเรือผ่านดาวเทียม
ที่มา : Government of Western Australia, 2018

ดังนั้นในการพัฒนาระบบตรวจสอบเรือประมงโดยใช้ตวามเป็นจริงเสริมนี้จึงใช้อีกทางเลือกหนึ่งซึ่งเหมาะสำหรับการประมงชายฝั่ง คือการใช้สมาร์ทโฟนและการสื่อสารผ่านเครือข่ายโทรศัพท์ เนื่องจากสมาร์ทโฟนเป็นอุปกรณ์ที่มีการใช้งานแพร่หลาย ส่วนใหญ่จะมีเครื่องรับสัญญาณจีพีเอส และ

สามารถเชื่อมต่ออินเทอร์เน็ตได้ ดังนั้นผู้วิจัยจึงพัฒนาระบบโดยมีกรอบแนวคิดดังแสดงในภาพที่ 3.2 ซึ่งเรือประมงจะทราบตำแหน่งของตัวเองได้จากเครื่องรับสัญญาณจีพีเอสในสมาร์ทโฟน และการติดต่อสื่อสารระหว่างเรือที่อยู่กลางทะเลกับเครื่องคอมพิวเตอร์แม่ข่ายที่อยู่บนฝั่งนั้นจะทำด้วยการส่งข้อมูลจะถูกส่งผ่านจีพีอาร์เอส (General Packet Radio Service : GPRS) มายังหน่วยรับบนฝั่งผ่านเครือข่ายจีเอสเอ็ม (global service mobile : GSM) ซึ่งข้อมูลที่ส่งมานั้นจะถูกจัดเก็บอยู่ที่คอมพิวเตอร์ให้บริการ ผู้ใช้อื่น ๆ เช่น เจ้าของเรือหรือเจ้าหน้าที่ที่เกี่ยวข้องด้านการติดตามเรือสามารถใช้สมาร์ทโฟนเพื่อแสดงตำแหน่งและชื่อเรือแบบตวามเป็นจริงเสริมได้ผ่านบริการจีพีอาร์เอสเช่นเดียวกัน



ภาพที่ 3.2 แนวคิดการทำงานของระบบที่เสนอ

3.1.1 ความต้องการของระบบ

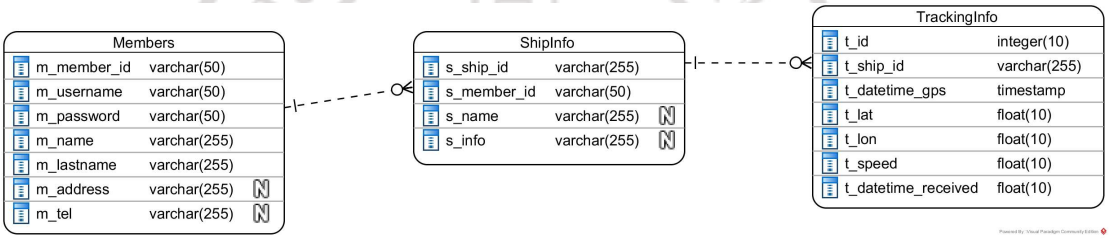
ระบบส่วนที่ทำหน้าหน้าที่จัดการข้อมูลเรือนั้นเป็นเว็บแอปพลิเคชัน มีระบบการตรวจสอบสิทธิผู้ใช้หรือระบบยืนยันตัวตน (authentication) ซึ่งผู้วิจัยได้แบ่งสิทธิ์การเข้าใช้งานเป็นสองส่วนหลัก คือ ไม่ต้องผ่านกระบวนการตรวจสอบสิทธิถือเป็นผู้ใช้ทั่วไป (general user) ซึ่งสามารถดูข้อมูลข่าวสารผ่านหน้าเว็บไซต์ได้แต่ไม่สามารถเข้าถึงตำแหน่งของเรือต่าง ๆ ได้ และผู้ใช้อีกกลุ่มซึ่งสามารถดูตำแหน่งของเรือได้ผ่านการสมัครสมาชิกเพื่อขอสิทธิเป็นเจ้าของเรือ (owner) โดยสิทธิดังกล่าวจะต้องผ่านหน้าเพื่อตรวจสอบสิทธิในการเข้าใช้งาน เช่นเดียวกันกับระดับผู้ใช้ซึ่งทำหน้าที่ผู้ดูแลระบบ (administrator) ดังรายละเอียดในตารางที่ 3.1

ตารางที่ 3.1 สิทธิ์ผู้ใช้ในระบบ

ประเภท	รายละเอียด
1) ผู้ใช้งานทั่วไป (user)	- ข้อมูลเกี่ยวกับโครงการ - ข่าวสารทั่วไป
2) เจ้าของเรือ (owner)	- สมัครสมาชิก - แสดงข้อมูลเรือของตน - แสดงตำแหน่งเรือของตน

3.1.2 ฐานข้อมูล

ผู้วิจัยได้รวบรวมข้อมูลต่าง ๆ พบว่าข้อมูลสำคัญที่จำเป็นต่อการทำงานของระบบคือข้อมูลตำแหน่งเรือซึ่งจะได้มาจากสมาร์ทโฟนบนเรือ และข้อมูลเกี่ยวกับเรือ เช่น ชื่อเรือและชื่อเจ้าของเรือ เป็นต้น ซึ่งผู้วิจัยนำข้อมูลเหล่านี้มาวิเคราะห์เพื่อสร้างเป็นฐานข้อมูลโดยใช้แบบจำลองฐานข้อมูลเชิงสัมพันธ์ (entity-relationship model : ER) ได้ผลเป็นแผนภาพอีอาร์ (ER-diagram) ดังแสดงในภาพที่ 3.3 ซึ่งมีตารางข้อมูลหลักสามตาราง คือ ตารางสมาชิก (Members) ตารางข้อมูลเรือ (ShipInfo) และตารางข้อมูลการติดตามเรือ (TrackingInfo) โดยมีพจนานุกรมข้อมูลของแต่ละตารางดังแสดงในตารางที่ 3.2 ถึงตารางที่ 3.4 โดยมีรายละเอียดของตารางต่าง ๆ ดังนี้



ภาพที่ 3.3 แผนภาพอีอาร์ของระบบ

1) ตารางสมาชิก

เป็นตารางที่ใช้สำหรับบันทึกข้อมูลผู้ใช้งานในระบบ โดยจะงานวิจัยนี้เป็นการเสนอระบบต้นแบบดังนั้นจึงบันทึกเพียงข้อมูลเบื้องต้นของผู้ใช้งาน เช่น ชื่อ นามสกุล ชื่อผู้ใช้ และรหัสผ่าน เป็นต้น ดังพจนานุกรมข้อมูลตารางสมาชิกในตารางที่ 3.2

ตารางที่ 3.2 ตารางสมาชิก

ชื่อฟิลด์	ความหมาย	ชนิดข้อมูล	คีย์หลัก/คีย์นอก
m_member_id	รหัสสมาชิก	varchar(50)	PK
m_username	ชื่อผู้ใช้	varchar(50)	
m_password	รหัสผ่าน	varchar(50)	
m_name	ชื่อ	varchar(255)	
m_lastname	นามสกุล	varchar(255)	
m_address	ที่อยู่	varchar(255)	
m_tel	เบอร์โทรศัพท์	varchar(255)	

2) ตารางข้อมูลเรือ

เป็นตารางที่ใช้สำหรับบันทึกข้อมูลเบื้องต้นเกี่ยวกับเรือแต่ละลำ โดยจะประกอบด้วยข้อมูลเกี่ยวกับเรือเช่น ชื่อเรือ รายละเอียด และรหัสสมาชิกที่เป็นเจ้าของเรือ เป็นต้น ดังพจนานุกรมข้อมูลตารางสมาชิกในตารางที่ 3.3

ตารางที่ 3.3 ตารางข้อมูลเรือ

ชื่อฟิลด์	ความหมาย	ชนิดข้อมูล	คีย์หลัก/คีย์นอก
s_id	รหัสเรือ	varchar(20)	PK
s_member_id	รหัสเจ้าของเรือ	varchar(50)	FK
s_name	ชื่อเรือ	varchar(255)	
s_info	รายละเอียดเพิ่มเติม	varchar(255)	

3) ตารางการติดตามเรือ

ตารางนี้เป็นตารางหลักสำหรับบันทึกพิกัดเรือเพื่อการแสดงผลกราฟิกส์ โดยในงานวิจัยนี้ออกแบบให้เก็บข้อมูลพิกัดเรือในระบบพิกัดภูมิศาสตร์เป็นค่าละติจูดและลองจิจูด รวมทั้งมีการบันทึกวันที่และเวลาด้วย โดยมีรายละเอียดเกี่ยวกับโครงสร้างตารางดังแสดงในตารางที่ 3.4

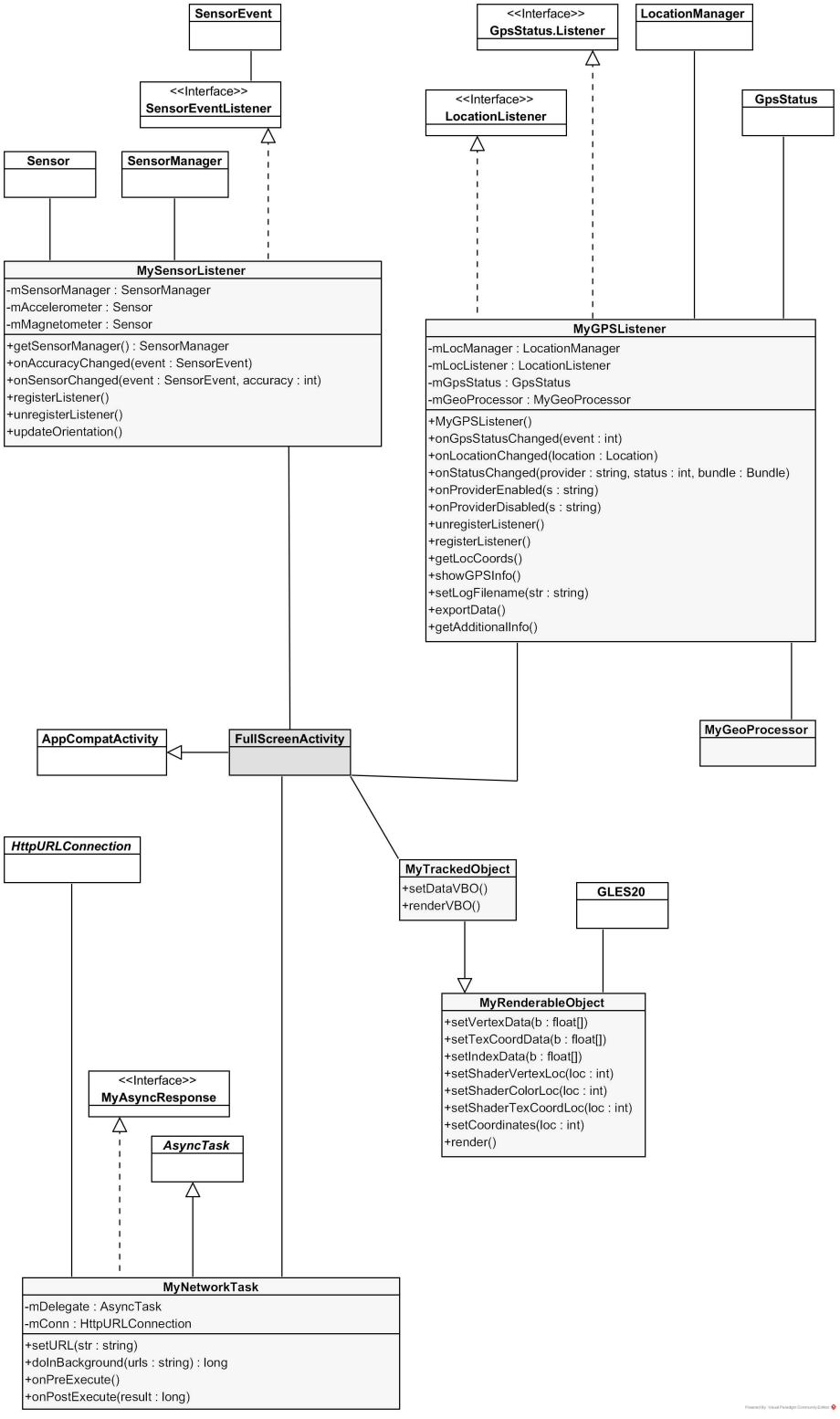
ตารางที่ 3.4 ตารางข้อมูลการติดตามตำแหน่งเรือ

ชื่อฟิลด์	ความหมาย	ชนิดข้อมูล	คีย์หลัก/ คีย์นอก
tr_id	รหัสข้อมูล	int (auto)	PK
t_ship_id	รหัสเรือ	varchar(20)	FK
t_datetime_gps	เวลาจากเครื่องรับสัญญาณจีพีเอส	timestamp	
t_latitude	ละติจูด	float(10,6)	
t_longitude	ลองจิจูด	float(10,6)	
t_speed	ความเร็ว	float(10,4)	
t_datetime_received	เวลารับสัญญาณ	timestamp	

3.2 ระบบการแสดงผลแบบตวามเป็นจริงเสริม

การแสดงกราฟิกส์แบบตวามเป็นจริงเสริมนั้นจำเป็นที่จะต้องทราบตำแหน่งและทิศทางของอุปกรณ์ที่ใช้งาน ณ ขณะนั้น ซึ่งปัจจุบันเราสามารถทราบข้อมูลเหล่านี้ได้เซ็นเซอร์ที่มีอยู่ในสมาร์ทโฟนเกือบทุกรุ่น ดังนั้นเมื่อทราบข้อมูลตำแหน่งและทิศทางของสมาร์ทโฟนแล้วเราจึงสามารถทำการแปลงภาพกราฟิกส์ของข้อมูลส่วนที่ต้องการนำมาเสริมเข้ากับภาพจริงให้มีการตำแหน่งและการเอียงตัวที่สอดคล้องกับอุปกรณ์ได้ ซึ่งเมื่อวิเคราะห์ความต้องการต่าง ๆ ในส่วนของการแสดงผลตวามเป็นจริงเสริมนั้นพบว่าจำเป็นต้องมีการอ่านค่าจากเซ็นเซอร์ การอ่านค่าพิกัดของอุปกรณ์จากเครื่องรับสัญญาณจีพีเอส การอ่านค่าพิกัดเรือจากเครื่องให้บริการ และการซ้อนข้อมูลตำแหน่งเรือกับภาพถ่ายจากกล้องของสมาร์ทโฟน ซึ่งผู้วิจัยนำข้อมูลเหล่านี้ไปวิเคราะห์ด้วยเทคนิคการวิเคราะห์และออกแบบระบบเชิงวัตถุ (object-oriented analysis and design : OOAD) และยูเอ็มแอล (unified

modeling language : UML) ได้ผลลัพธ์เป็นคลาสไดอะแกรม (class diagram) ดังแสดงในภาพที่ 3.4 โดยแสดงคลาสที่สำคัญด้วยพื้นหลังสีเทา



ภาพที่ 3.4 คลาสไดอะแกรม

คลาสที่สำคัญในการศึกษาครั้งนี้ประกอบด้วยคลาส MySensorListener สำหรับการจัดการข้อมูลจากเซ็นเซอร์วัดทิศทางของสมาร์ทโฟน, คลาส MyGPSListener สำหรับการจัดการข้อมูลจากเครื่องรับสัญญาณจีพีเอส, คลาส MyNetworkTask สำหรับการอ่านข้อมูลจากอินเทอร์เน็ตและคลาส FullScreenActivity สำหรับการประสานงานระหว่างคลาสต่าง ๆ รวมทั้งคลาสที่เกี่ยวข้องการกับแสดงผลกราฟิกส์ด้วย โดยมีรายละเอียดการออกแบบคลาสที่สำคัญแต่ละส่วนดังนี้

3.2.1 การระบุตำแหน่งอุปกรณ์

งานวิจัยนี้ระบุตำแหน่งปัจจุบันของผู้ใช้ด้วยการใช้ข้อมูลจากเครื่องรับสัญญาณจีพีเอสในสมาร์ทโฟนที่ทำงานบนระบบปฏิบัติการแอนดรอยด์ โดยการเข้าถึงและการใช้งานเครื่องรับสัญญาณจีพีเอสนั้นทำได้ด้วยการเรียกใช้บริการ LocationManager ของระบบปฏิบัติการ โดยมีรายละเอียดขั้นตอนการทำงานอ้างอิงตาม Google (2017) ซึ่งขั้นตอนแรกคือการขออนุญาตเข้าถึงบริการ LocationManager ซึ่งทำได้ด้วยการกำหนดค่าการอนุญาตสองค่า ประกอบด้วย ACCESS_COARSE_LOCATION และ ACCESS_FINE_LOCATION เพื่อระบุระดับความถูกต้องของตำแหน่งที่ระบบจะคืนค่ากลับมา การเปิดการขออนุญาตทำได้ด้วยการกำหนดค่าใน เมนิเฟสต์ ดังแสดงในภาพที่ 3.5

```
<uses-permission
 android:name="android.permission.ACCESS_COARSE_LOCATION"/>
```

ภาพที่ 3.5 การขออนุญาตเข้าถึงบริการระบุตำแหน่งของแอนดรอยด์

อย่างไรก็ตามในกูเกิลได้มีการเปลี่ยนแปลงเงื่อนไขการขออนุญาตการเข้าถึงบริการนี้ โดยตั้งแต่แอนดรอยด์เอสดีเคเวอร์ชัน 23 (Android SDK 23) เป็นต้นมาได้มีการกำหนดว่าโปรแกรมที่จะใช้งาน LocationManager นั้นนอกจากการกำหนดในเมนิเฟสต์แล้ว โปรแกรมจะต้องขออนุญาตในขณะที่โปรแกรมกำลังทำงาน โดยมีแนวทางการขออนุญาตด้วยการกำหนดให้โปรแกรมมีการตรวจสอบสถานะการอนุญาต ณ ปัจจุบันว่าระบบอนุญาตให้ใช้งานหรือยังด้วยการเรียกเมท็อด ActivityCompat.checkSelfPermission โดยส่งแอคทีวิตีปัจจุบันและชื่อของบริการที่ต้องการตรวจสอบ หากผู้ใช้ยังไม่มีการอนุญาตก็สามารถขออนุญาตได้ด้วยการเรียกเมท็อด ActivityCompat.requestPermissions พร้อมกับส่งค่าต่าง ๆ ดังแสดงในภาพที่ 3.6


```
if (ActivityCompat.checkSelfPermission(
   this, Manifest.permission.ACCESS_FINE_LOCATION)
   != PackageManager.PERMISSION_GRANTED) {
  ActivityCompat.requestPermissions(this,
    new String[]
        {Manifest.permission.ACCESS_FINE_LOCATION},
    REQUEST_LOCATION);
} else {...}
```

ภาพที่ 3.6 ตัวอย่างชุดคำสั่งสำหรับการขออนุญาตเข้าถึงบริการระบุตำแหน่งขณะทำงาน

เมื่อขออนุญาตเสร็จเรียบร้อยแล้ว ลำดับต่อมาจึงเป็นการสร้างวัตถุของคลาส LocationManager เพื่อใช้สำหรับการติดต่อกับตัวจัดการข้อมูลจากเครื่องรับสัญญาณจีพีเอส ซึ่งในงานวิจัยนี้จะทำการสร้างวัตถุนี้ในเหตุการณ์ onCreate ของคลาสหลักของโปรแกรม ดังตัวอย่างชุดคำสั่งในภาพที่ 3.7

```
@Override
protected void onCreate(Bundle savedInstanceState) {
  ...
mFusedLocationClient =
LocationServices.getFusedLocationProviderClient(this);
  ...
}
```

ภาพที่ 3.7 การสร้างวัตถุของคลาส LocationManager

โดยในงานวิจัยนี้ออกแบบคลาส MyGPSListener เพื่อเป็นคลาสหลักสำหรับอ่านข้อมูลพิกัดของอุปกรณ์ โดยออกแบบให้เป็นคลาสที่สืบทอดต่อจาก LocationListener และคลาส GpsStatus.Listener ด้วย โดยมีเมท็อดที่สำคัญดังนี้

1) เมท็อด onGpsStatusChanged

เมท็อดนี้จะถูกเรียกให้ทำงานเมื่อสถานะของเครื่องรับสัญญาณจีพีเอสมีการเปลี่ยนแปลง โดยระบบปฏิบัติการจะคืนค่าสถานะกลับมาเป็นค่าคงที่ มีทั้งหมดสี่ค่า ดังแสดงในตารางที่ 3.5

ตารางที่ 3.5 สถานะของดาวเทียมจากเมท็อด onGpsStatusChanged
ที่มา : Google, 2017

ค่า	ความหมาย
GPS_EVENT_STARTED	เริ่มต้นทำงาน
GPS_EVENT_STOPPED	หยุดการทำงาน
GPS_EVENT_FIRST_FIX	พิกัดพร้อมใช้งานเป็นครั้งแรก
GPS_EVENT_SATELLITE_STATUS	มีข้อมูลสถานะจากดาวเทียม

ซึ่งในกรณีสุดท้ายนี้ระบบปฏิบัติการจะส่งค่าแบบเป็นคาบและสามารถเรียกอ่านค่าสถานะของดาวเทียมแต่ละดวงได้ด้วยเมธอด getSatellites

2) เมท็อด onLocationChanged

เมท็อดนี้จะถูกเรียกให้ทำงานเมื่อพิกัดจากเครื่องรับสัญญาณจีพีเอสเปลี่ยนแปลง โดยระบบจะส่งค่าคืนกลับมาเป็นวัตถุของคลาส Location ซึ่งมีโครงสร้างประกอบด้วยค่าพิกัดภูมิศาสตร์ที่เครื่องรับสัญญาณจีพีเอสรับค่าได้ ซึ่งมีค่าละติจูด ลองจิจูด และเวลา รวมถึง

ข้อมูลอื่น เมื่อโปรแกรมสกัดค่าพิกัดได้จะบันทึกค่าที่ได้รับไว้ในตัวแปรสมาชิก `mLocCoords` จากนั้นจึงส่งข้อมูลนี้ต่อไปให้แอคทีวิตีเพื่อปรับปรุงการแสดงผลต่อไป

3) เมท็อด `onStatusChanged`

เมท็อดนี้จะถูกเรียกให้ทำงานเมื่อสถานะของผู้ให้บริการ (provider) เปลี่ยนแปลง เช่น ระบบไม่สามารถอ่านค่าตำแหน่งได้ หรือเมื่อระบบสามารถอ่านค่าตำแหน่งได้อีกครั้งหลังจากไม่สามารถอ่านค่าได้มาช่วงระยะเวลาหนึ่ง

4) เมท็อด `onProviderEnabled`

เมท็อดนี้จะถูกเรียกให้ทำงานเมื่อผู้ใช้กำหนดใช้งานผู้ให้บริการ โดยจะมีค่าเป็นสายอักขระของชื่อของผู้ใช้บริการที่เป็นผู้เรียกเหตุการณ์นี้

5) เมท็อด `onProviderDisabled`

เมท็อดนี้จะถูกเรียกให้ทำงานเมื่อผู้ใช้ยกเลิกการใช้งานผู้ให้บริการ การเรียกเมท็อดนี้จะทำให้การอ่านค่าพิกัดจากเครื่องรับสัญญาณจีพีเอสหยุดลงทันที

6) เมท็อด `registerListener` และ `unregisterListener`

เมท็อดนี้เป็นเมท็อดสำหรับการลงทะเบียน (register) บริการและยกเลิกการลงทะเบียน (unregister) บริการระบุตำแหน่งจากเครื่องรับสัญญาณจีพีเอส โดยงานวิจัยนี้ออกแบบให้บริการหยุดทำงานชั่วคราวเมื่อโปรแกรมไม่ได้ทำงาน เช่น ปิดโปรแกรม หรือ สลับการทำงานไปยังโปรแกรมอื่น เป็นต้น เพื่อประหยัดทรัพยากรของระบบและเพื่อลดความร้อนของอุปกรณ์ โดยงานวิจัยนี้กำหนดการลงทะเบียนใช้บริการระบุตำแหน่งพร้อมกับการเพิ่มตัวฟังเหตุการณ์การเปลี่ยนสถานะดาวเทียมจีพีเอส โดยมีตัวแปรสมาชิก `mIsRegistered` สำหรับการตรวจสอบว่าโปรแกรมสามารถเข้าถึงและใช้งานบริการระบุตำแหน่งได้หรือไม่ ดังชุดคำสั่งในภาพที่ 3.8

```
/**
 * Register GPS Listener
 */
public void registerListener() {
  this.mIsRegistered = false;
  try {
    this.mLocManager.requestLocationUpdates(
        LocationManager.GPS_PROVIDER,
        0, 0,
        this);
    // Add GPSStatus listener.
    this.mLocManager.addGpsStatusListener(this);
    this.mIsRegistered = true;
  } catch(SecurityException e) {
    ...
  }
}
```

ภาพที่ 3.8 การลงทะเบียนใช้บริการระบุตำแหน่ง `requestLocationUpdates`

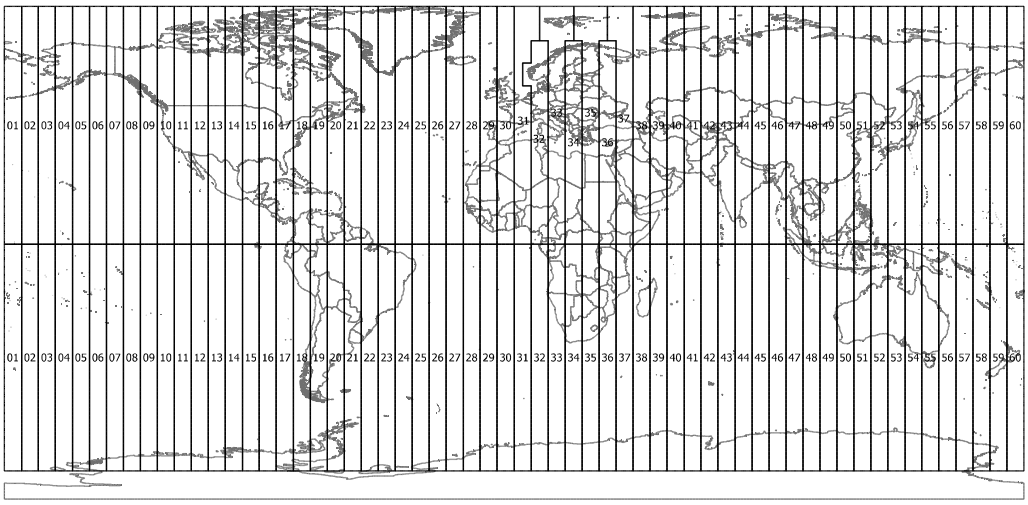
7) เมท็อด `showGPSInfo`

เป็นเมท็อดที่ผู้วิจัยออกแบบไว้สำหรับอ่านค่าและแปลงค่าพิกัดที่ได้จากเครื่องรับสัญญาณจีพีเอสให้พร้อมใช้งานในส่วนอื่น ๆ ต่อไปได้ โดยเมท็อดนี้จะอ่านค่าสถานะของดาวเทียมที่สำคัญ เช่น จำนวนที่รับสัญญาณได้และจำนวนดาวเทียมที่ใช้ในการคำนวณตำแหน่ง เพื่อให้ผู้ใช้ตรวจสอบว่าสามารถเชื่อถือค่าพิกัดที่อ่านค่าได้มากน้อยเพียงใด หลังจากนั้นจึงอ่านค่าพิกัดจากบริการซึ่งจะคืนค่ามาเป็นระบบพิกัดภูมิศาสตร์ ซึ่งโดยทั่วไปแล้วพิกัดที่อ่านค่าได้จะเป็นพิกัดที่มีพื้นหลักฐาน (datum) เป็นดับเบิลยูจีเอสแปดสิบสี่ (World Geodetic System 1984 : WGS84)

3.2.2 การแปลงระบบพิกัด

อย่างไรก็ตามในการแสดงผลบนหน้าจอคอมพิวเตอร์โดยทั่วไปนั้นมักจะใช้ระบบพิกัดฉาก ดังนั้นจึงจำเป็นต้องแปลงพิกัดจากระบบพิกัดภูมิศาสตร์ให้เป็นเป็นระบบพิกัดฉาก โดยงานวิจัยนี้เลือกใช้เส้นโครงแผนที่ (map projection) เป็นแบบระบบพิกัดยูทีเอ็ม (universal transverse mercator : UTM) โซนสี่สิบเจ็ดเอ็นพี (47NP) และใช้พื้นหลักฐานเป็นดับเบิลยูจีเอสแปดสิบสี่

โดยผู้วิจัยได้แยกการคำนวณการแปลงพิกัดนี้ไว้ในคลาส `MyGeoProcessor` เป็นใช้เป็นคลาสอรรถประโยชน์ (utility) สำหรับการแปลงพิกัดในระบบพิกัดภูมิศาสตร์เป็นยูทีเอ็ม ภาพที่ 3.8 แสดงพื้นที่โซนต่าง ๆ ของระบบยูทีเอ็ม ซึ่งงานวิจัยนี้ได้ออกแบบให้รองรับการแปลงเฉพาะโซนสี่สิบเจ็ดเอ็น (47N) เท่านั้น



ภาพที่ 3.9 โซนของระบบยูทีเอ็ม
ที่มา : National Geospatial-Intelligence Agency, (2018)

อย่างไรก็ตามแอนดรอย์ไม่มีคลังโปรแกรมสำหรับคำนวณการแปลงระบบพิกัดดังกล่าว ดังนั้นผู้วิจัยจึงคำนวณการแปลงระบบพิกัดภูมิศาสตร์เป็นยูทีเอ็มโดยใช้เทคนิคการแปลงระบบพิกัดทรงกลมของ Palacios (2006) และ Martinez (2012) ซึ่งคำนวณได้จากสมการที่ 3.1 ถึงสมการที่ 3.24

$$\phi = \phi_{deg}(\pi /180) \quad 3.1$$
$$\lambda = \lambda_{deg}(\pi /180) \quad 3.2$$
$$e_2 = ((C_1^2 - C_2^2)^{0.5})/C_2 \quad 3.3$$
$$e_3 = e_2^2 \quad 3.4$$
$$c = C_1^2/C_2 \quad 3.5$$
$$H = \text{fix}((\lambda_{deg}/6) + 31) \quad 3.6$$
$$S = ((6H) - 183) \quad 3.7$$
$$\delta_s = \lambda - (S(\pi /180)) \quad 3.8$$
$$a = \cos(\phi)\sin(\delta_s) \quad 3.9$$
$$\varepsilon = 0.5\log((1 + a)/(1 - a)) \quad 3.10$$
$$v = \tan^{-1}(\tan(\phi)/\cos(\delta_s)) - \phi \quad 3.11$$
$$V = (c/((1 + (e_3(\cos(\phi))^2)))^{0.5})C_3 \quad 3.12$$
$$t_a = (e_3/2)\varepsilon^2(\cos(\phi))^2 \quad 3.13$$
$$a_1 = \sin(2\phi) \quad 3.14$$
$$a_2 = a_1(\cos(\phi))^2 \quad 3.15$$
$$j_2 = \phi + (a_1/2) \quad 3.16$$
$$j_4 = ((3j_2) + a_2)/4 \quad 3.17$$
$$j_6 = ((5j_4) + (a_2(\cos(\phi))^2))/3 \quad 3.18$$
$$\alpha = (3/4)e_3 \quad 3.19$$
$$\beta = (5/3)\alpha^2 \quad 3.20$$
$$\gamma = (35/27)\alpha^3 \quad 3.21$$
$$B_m = cC_3(\phi - \alpha j_2 + \beta j_4 - \gamma j_6) \quad 3.22$$
$$E = \varepsilon V(1 + (t_a/3)) + C_4 \quad 3.23$$
$$N = vV(1 + t_a) + B_m \quad 3.24$$

เมื่อ (ϕ_{deg}, λ_{deg}) คือพิกัดละติจูดและลองจิจูดอินพุต (องศา), (E, N) คือพิกัดในระบบยูทีเอ็ม (เมตร) ระบุตำแหน่งตามแนวแกน **X** และ **Y** ตามลำดับ และ C_1, C_2, C_3, และ C_4 คือพารามิเตอร์ของการแปลงพิกัดในระบบยูทีเอ็ม ซึ่งในงานวิจัยกำหนดใช้ค่าดังแสดงในตารางที่ 3.8

ตารางที่ 3.6 พารามิเตอร์การแปลงพิกัดยูทีเอ็มพื้นหลักฐานดับเบิลยูจีเอสแปดสิบสี่
ที่มา : Google, 2017

พารามิเตอร์	ความหมาย	ค่า
C_1	ความยาวครึ่งแกนหลัก	6378137.000000 เมตร
C_2	ความยาวครึ่งแกนรอง	6356752.314245 เมตร
C_3	ค่าปรับสเกล	0.9996
C_4	ระยะเลื่อนจุดกำเนิดแกน **X**	500000 เมตร

3.2.3 การระบุการทิศทางของสมาร์ทโฟน

ค่าจากเครื่องรับสัญญาณจีพีเอสนั้นเป็นพิกัดปัจจุบันของผู้ใช้ ยังไม่สามารถนำมาใช้ในการแสดงผลแบบตวามเป็นจริงเสริมได้ทันที เราจะเป็นที่จะต้องทราบค่าการหมุนหรือทิศทางของสมาร์ทโฟนด้วย ดังนั้นงานวิจัยนี้จึงอ่านค่าทิศทางของสมาร์ทโฟนโดยใช้บริการ SensorManager ของระบบปฏิบัติการแอนดรอยด์ซึ่งทำให้โปรแกรมสามารถเข้าถึงรายการและข้อมูลของเซ็นเซอร์ที่ติดตั้งในสมาร์ทโฟนได้ ซึ่งเซ็นเซอร์ในระบบปฏิบัติการแอนดรอยด์นั้นจะเป็นได้ทั้งอุปกรณ์ที่เป็นฮาร์ดแวร์และซอฟต์แวร์ (Google, 2017)

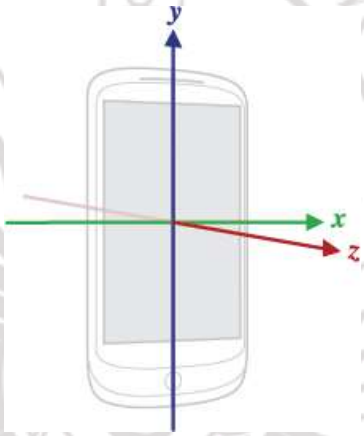
ตารางที่ 3.7 ตัวอย่างเซ็นเซอร์ที่มีในระบบปฏิบัติการแอนดรอยด์
ที่มา : Google, 2017

เซ็นเซอร์	ประเภท	รายละเอียด
TYPE_ACCELEROMETER	ฮาร์ดแวร์	วัดแรงที่เกิดจากความเร่งที่กระทำกับอุปกรณ์ รวมทั้งแรงโน้มถ่วงด้วย (เมตรต่อวินาทีกำลังสอง)
TYPE_GYROSCOPE	ฮาร์ดแวร์	วัดอัตราการหมุนของอุปกรณ์ (เรเดียนต่อวินาที)
TYPE_LINEAR_ACCELERATION	ฮาร์ดแวร์	วัดแรงที่เกิดจากความเร่งที่กระทำกับอุปกรณ์ รวมไม่รวมแรงโน้มถ่วง (เมตรต่อวินาทีกำลังสอง)
TYPE_MAGNETIC_FIELD	ฮาร์ดแวร์	วัดความเข้มสนามแม่เหล็กโลก (ไมโครเทสล่า)
TYPE_ORIENTATION	ซอฟต์แวร์	วัดมุมของการหมุนของอุปกรณ์รอบแกนหลัก (องศา)
TYPE_ROTATION_VECTOR	ซอฟต์แวร์หรือฮาร์ดแวร์	วัดทิศทางของอุปกรณ์จากเวกเตอร์จากการหมุนของอุปกรณ์

โดยบริการ SensorManager นี้แบ่งเซ็นเซอร์ออกเป็นสามกลุ่ม ประกอบด้วยเซ็นเซอร์ความเคลื่อนไหว (motion sensors) เซ็นเซอร์สภาพแวดล้อม (environmental sensors) และเซ็นเซอร์ตำแหน่ง (position sensors) โดยกรณีของเซ็นเซอร์ความเคลื่อนไหวจะเป็นกลุ่มของเซ็นเซอร์ที่ทำหน้าที่วัดความเคลื่อนไหวของอุปกรณ์ด้วยการวัดความเร่งและความเร็วเชิงมุม ตัวอย่างของอุปกรณ์กลุ่มนี้ได้แก่ เครื่องวัดความเร่ง (accelerometers), เครื่องวัดแรงโน้มถ่วง (gravity sensors) และ เครื่องวัดอัตราการหมุน (ความเร็วเชิงมุม) หรือไจโรสโคป (gyroscopes) สำหรับเซ็นเซอร์กลุ่มที่สองนั้นจะทำหน้าที่ตรวจวัดสภาพแวดล้อมรอบ ๆ สมาร์ทโฟน เช่น ความเข้มแสงแวดล้อม (ambient light) อุณหภูมิ และความชื้นเป็นต้น ตัวอย่างขอเซ็นเซอร์ในกลุ่มนี้ได้แก่

บาโรมิเตอร์ ฟโตมิเตอร์ และเทอร์โมมิเตอร์ สำหรับเซ็นเซอร์กลุ่มสุดท้ายจะทำหน้าที่วัดตำแหน่งของอุปกรณ์ ตัวอย่างเช่นเซ็นเซอร์วัดการเอียงตัวและเครื่องวัดความเข้มสนามแม่เหล็กโลก (magnetometer) เป็นต้น ตารางที่ 3.7 แสดงตัวอย่างของเซ็นเซอร์บางส่วนที่ปรากฏในระบบปฏิบัติการแอนดรอยด์ที่เกี่ยวข้องกับการวัดการเอียงตัว

ทั้งนี้ข้อมูลที่ส่งมาจากเซ็นเซอร์ความเคลื่อนไหวนั้นจะเป็นวัตถุของคลาส `SensorEvent` ซึ่งระบบพิกัดที่ใช้ในบริการต่าง ๆ ของแอนดรอยด์จะกำหนดให้ระนาบหน้าจอของสมาร์ทโฟนเป็นระนาบ **X-Y** โดยมีจุดกำเนิดที่กึ่งกลางหน้าจอ มีแกน **X+** ชี้ไปทางขวามือและ **Y+** ชี้ขึ้นด้านบน ในขณะที่แกน **Z+** ชี้ออกจากระนาบ พุ่งเข้าหาผู้ใช้ ดังภาพที่ 3.10



ภาพที่ 3.10 กรอบอ้างอิงของพิกัดของระบบปฏิบัติการแอนดรอยด์
ที่มา : Google, 2017

3.2.4 การจัดการข้อมูลตำแหน่งและทิศทางของสมาร์ทโฟน

โดยปกติแล้วสิ่งที่ต้องทำเมื่อต้องการใช้งานเซ็นเซอร์จะมีรูปแบบคล้ายกับการใช้บริการระบุตำแหน่ง นั่นคือการตรวจสอบรายการและคุณสมบัติของเซ็นเซอร์ที่มีในอุปกรณ์ จากนั้นจึงลงทะเบียนการใช้เซ็นเซอร์และจัดการกับข้อมูลรวมถึงเหตุการณ์ต่าง ๆ ที่ต้องการ รวมทั้งต้องกำหนดรายละเอียดการทำงานเมื่อมีเหตุการณ์การหยุด `onPause` หรือทำงานต่อ `onResume` ดังนั้นงานวิจัยนี้จึงออกแบบคลาส `MySensorListener` เพื่อทำหน้าที่จัดการการใช้งานเซ็นเซอร์ต่าง ๆ ที่ต้องการไว้ในคลาสเดียว โดยผู้วิจัยได้ออกแบบให้คลาสนี้เป็นคลาสที่สืบทอดต่อจากคลาส `SensorEventListener` โดยมีรายละเอียดที่สำคัญดังนี้

1) การตรวจสอบรายการเซ็นเซอร์

การตรวจสอบรายการเซ็นเซอร์ (sensors list) ทำได้ด้วยการเรียกเมท็อด `getSensorList` ของคลาส `SensorManager` ซึ่งระบบจะคืนค่าเป็นข้อมูลต่าง ๆ เกี่ยวกับเซ็นเซอร์ เช่น ชื่อเซ็นเซอร์และข้อมูลผู้ผลิต เป็นต้น โดยในกรณีนี้จะตรวจสอบว่ามีเซ็นเซอร์ที่ต้องการหรือไม่ ถ้ามีครบตามที่ต้องการโปแกรมจะสร้างวัตถุของเซ็นเซอร์ขึ้นไว้ในตัวแปรสมาชิก `mAccelerometer`, `mMagnetometer`, และ `mOrientationSensor` หากไม่พบจะจบการทำงาน

2) การลงทะเบียนใช้งาน

การลงทะเบียนใช้งานทำด้วยการเรียกเมท็อด `registerListener` ของคลาส `SensorManager` ซึ่งจะทำให้ระบบส่งข้อมูลจากเซ็นเซอร์ที่ระบุกลับมาให้ผู้ใช้ทันทีตามค่าการหน่วงเวลาที่ผู้ใช้กำหนด โดยระบบปฏิบัติการมีค่าการหน่วงเวลาสี่แบบดังแสดงในตารางที่ 3.8 โดยสามารถระบุได้ตั้งแต่แบบปกติที่มีค่าการหน่วงเวลาประมาณ 200,000 ไมโครวินาที ซึ่งจะทำให้โปรแกรมมีอัตราการปรับปรุงข้อมูลประมาณ 5 ข้อมูลต่อวินาทีหรือสามารถกำหนดให้ไม่มีการหน่วงเวลาเลยก็เป็นได้

ตารางที่ 3.8 ค่าการหน่วงเวลาสำหรับการอ่านข้อมูลจากเซ็นเซอร์
ที่มา : Google, 2017

ค่าคงที่	ความหมาย	ค่าการหน่วง (ไมโครวินาที)
`SENSOR_DELAY_FASTEST`	อ่านค่าจากเซ็นเซอร์ให้เร็วที่สุดเท่าที่จะทำได้	0
`SENSOR_DELAY_GAME`	อ่านค่าด้วยอัตราที่เหมาะสมสำหรับการใช้งานกับเกม	20,000
`SENSOR_DELAY_UI`	อ่านค่าด้วยอัตราที่เหมาะสมสำหรับการใช้ร่วมกับส่วนติดต่อผู้ใช้	60,000
`SENSOR_DELAY_NORMAL`	อ่านค่าด้วยอัตราที่เหมาะสมสำหรับการคำนวณการเอียงตัวของหน้าจอ	200,000

ตารางที่ 3.9 พารามิเตอร์ของเหตุการณ์ `onSensorChanged`
ที่มา : Google, 2017

ค่า	ความหมาย
`public int accuracy`	ความถูกต้องของค่าของเหตุการณ์นี้
`public Sensor sensor`	เซ็นเซอร์ที่เป็นตัวสร้างเหตุการณ์นี้
`public long timestamp`	เวลา (นาโนวินาที) ณ ขณะที่เหตุการณ์นี้เกิดขึ้น
`public final float[] values`	แถวลำดับของค่าจากเซ็นเซอร์

3) การอ่านค่าจากเซ็นเซอร์

โปรแกรมจะอ่านค่าจากเซ็นเซอร์ได้สองแนวทางคือเมื่อความถูกต้องของเซ็นเซอร์มีการเปลี่ยนแปลง และอีกกรณีหนึ่งคือเมื่อค่าที่วัดได้มีการเปลี่ยนแปลง โดยในกรณีแรกจะทำให้เกิดเหตุการณ์ `onAccuracyChanged` ซึ่งในกรณีของงานวิจัยนี้ไม่ได้ดำเนินการใด ๆ กับเหตุการณ์นี้ สำหรับกรณีที่สองคือเหตุการณ์ `onSensorChanged` ซึ่งจะถูกเรียกให้ทำงานเมื่อมี

เหตุการณ์เซ็นเซอร์ใหม่เข้ามาในระบบ อย่างไรก็ตามค่าที่ได้รับอาจจะไม่ใช่ค่าใหม่เสมอไป อาจจะเป็นค่าเดิมแต่ค่าเวลา (timestamp) มีการเปลี่ยนแปลงก็เป็นได้ (Google, 2017) ซึ่งเมท็อดนี้จะส่งค่าที่โปรแกรมต้องการมาเป็นอาร์กิวเมนต์ซึ่งเป็นวัตถุของคลาส `SensorEvent` ที่ภายในจะบันทึกค่าเกี่ยวกับเซ็นเซอร์ เวลา ความถูกต้องและข้อมูลที่วัดได้ ดังแสดงในตารางที่ 3.9

ระบบปฏิบัติการจะส่งค่าที่วัดได้มาในตัวแปร `values` ซึ่งข้อมูลในตัวแปรนี้จะมีค่าต่างกันไปขึ้นอยู่กับเซ็นเซอร์ที่เป็นผู้เรียกเหตุการณ์นี้ให้ทำงาน เช่น หากเป็น `TYPE_ACCELEROMETER` ค่าภายในตัวแปร `values` จะค่าความเร่งที่กระทำกับเซ็นเซอร์ที่วัดได้ตามแนวแกน **X**, **Y**, และ **Z** ตามลำดับ โดยมีรายละเอียดของแต่ละเซ็นเซอร์ดังแสดงในตารางที่ 3.10

ตารางที่ 3.10 ตารางความหมายข้อมูลที่อ่านได้จาก `onSensorChanged`
ที่มา : Google, 2017

เซ็นเซอร์	ค่าในแถวลำดับ
`TYPE_ACCELEROMETER`	ความเร่ง (เมตรต่อวินาทีกำลังสอง) `values[0]`: ความเร่งตามแนวแกน **X** `values[1]`: ความเร่งตามแนวแกน **Y** `values[2]`: ความเร่งตามแนวแกน **Z**
`TYPE_MAGNETIC_FIELD`	ความเข้มสนามแม่เหล็ก (ไมโครเทสล่า) `values[0]`: ความเข้มสนามแม่เหล็กตามแนวแกน **X** `values[1]`: ความเข้มสนามแม่เหล็กตามแนวแกน **Y** `values[2]`: ความเข้มสนามแม่เหล็กตามแนวแกน **Z**
`TYPE_GYROSCOPE`	อัตราการหมุน (rate of rotation) รอบแกนต่าง ๆ ของอุปกรณ์หรือความเร็วเชิงมุม (angular speed) มีค่าเป็นบวกหากเป็นการหมุนตามเข็มนาฬิกา (เรเดียนต่อวินาที) `values[0]`: อัตราการหมุนรอบแกน **X** `values[1]`: อัตราการหมุนรอบแกน **Y** `values[2]`: อัตราการหมุนรอบแกน **Z**


```
public void onSensorChanged(SensorEvent event)
{
  ...
  linear_acceleration[0] = event.values[0] - g[0];
  linear_acceleration[1] = event.values[1] - g[1];
  linear_acceleration[2] = event.values[2] - g[2];
  ...
}
```

ภาพที่ 3.11 การจัดการข้อมูลความเร่ง
ที่มา : Google, 2018

อย่างไรก็ตามในกรณีของค่าที่วัดได้จากเครื่องวัดความเร่งนั้นจำเป็นที่จะต้องแยกหรือลบผลจากแรงโน้มถ่วงที่กระทำกับเซ็นเซอร์ออกไปเสียก่อน รวมทั้งอาจจะจำเป็นต้องลดสัญญาณรบกวนให้น้อยลงด้วย ซึ่งอาจจะทำได้ดังแสดงตัวอย่างในภาพที่ 3.11

4) การยกเลิกการใช้งาน

การอ่านค่าจากเซ็นเซอร์นั้นต้องใช้การคำนวณมากกว่าการทำงานปกติ ซึ่งจะทำให้ระบบปฏิบัติการใช้พลังงานของหน่วยประมวลผลมาก ส่งผลให้อุปกรณ์ร้อนและเปลืองพลังงาน ดังนั้นงานวิจัยนี้ออกแบบให้โปรแกรมหยุดอ่านค่าจากเซ็นเซอร์เมื่อไม่ได้จำเป็นต้องใช้งาน โดยหากโปรแกรมตรวจพบว่าผู้ใช้มีการพักหน้าจอหรือหยุดการทำงาน ก็จะยกเลิกการลงทะเบียนเซ็นเซอร์ โดยงานวิจัยนี้ออกแบบให้การดำเนินการนี้ทำในเหตุการณ์ `onPause`

3.2.5 การคำนวณทิศทาง

งานวิจัยนี้เลือกใช้เซ็นเซอร์สามตัว ประกอบด้วยเครื่องวัดความเร่ง เครื่องวัดทิศทาง และเครื่องวัดความเร็วเชิงมุม โดยนำข้อมูลจากเซ็นเซอร์เหล่านี้มารวมกันเพื่อคำนวณทิศทางของอุปกรณ์ ดังรายละเอียดการทำงานต่อไปนี้

1) การขจัดสัญญาณรบกวน

ดังที่ได้กล่าวมาแล้วว่าข้อมูลที่วัดได้โดยตรงจากเซ็นเซอร์ของสมาร์ทโฟนมักจะมีสัญญาณรบกวนมาก การนำค่าที่วัดได้ไปใช้ทันทีอาจจะทำให้เกิดความคลาดเคลื่อนได้ ในการคำนวณการเอียงตัวของอุปกรณ์นั้นอาจจะทำได้ด้วยการใช้ตัวกรองเติมเต็มหรือคอมพลิเมนทารีฟิลเตอร์ (complementary filter) ซึ่งใช้ข้อมูลความเร่งร่วมกับข้อมูลความเร็วเชิงมุมโดยอาจจะคำนวณได้ตามเทคนิคของ W3C (2017) ซึ่งนำค่าจากเครื่องวัดความเร็วเชิงมุมและเครื่องวัดความเร่งมาใช้ในการคำนวณได้เป็นสมการ

$$\theta = \alpha(\theta_p + \omega\Delta t) + (1-\alpha)\eta \qquad 3.25$$

เมื่อ α เป็นถ่วงน้ำหนักซึ่งเป็นเลขจำนวนจริงในช่วง $[0, 1]$, ω เป็นค่าจากเครื่องวัดความเร็วเชิงมุม, η เป็นค่าจากเครื่องวัดความเร่ง, θ_p เป็นค่ามุมที่คำนวณได้ในรอบก่อนหน้า และ Δt เป็นช่วงเวลาระหว่างการคำนวณรอบก่อนหน้ากับรอบปัจจุบัน

การกรองวิธีนี้ให้ผลที่ค่อนข้างดีแต่มีข้อจำกัดประการหนึ่งคืออุปกรณ์จะต้องมีเซ็นเซอร์วัดความเร็วเชิงมุม ซึ่งมักจะพบในสมาร์ทโฟนรุ่นที่มีราคาค่อนข้างสูงเป็นส่วนใหญ่ ดังนั้นงานวิจัยนี้จึงเลือกเทคนิคการกรองสัญญาณที่สามารถทำงานได้ด้วยการใช้ข้อมูลจากเครื่องวัดความเร่งเท่านั้น โดยผู้วิจัยอาศัยค่าที่ได้จากบริการเซ็นเซอร์แบบ `TYPE_ORIENTATION` ของแอนดรอยด์ ปัญหาอีกประการหนึ่งคือเซ็นเซอร์ในสมาร์ทโฟนมักจะมีสัญญาณรบกวนมาก อันอาจจะเนื่องมาจากเป็นเซอร์ราคาประหยัด ดังนั้นผู้วิจัยจึงลดสัญญาณรบกวนด้วยเทคนิคตัวกรองความถี่ต่ำผ่าน (low-pass filter) โดยมีสมการการคำนวณตามเทคนิคของ W3C (2017) เป็น

$$\theta = (1-\alpha)(\theta_n) + (\alpha)(\theta_p) \qquad 3.26$$

เมื่อ θ_n คือค่าที่วัดได้ในรอบปัจจุบันและ θ_p คือค่าที่วัดได้จากก่อนหน้านี้ โดยงานวิจัยได้กำหนดค่าขีดแบ่ง α เป็นค่าคงที่ `LOW_PASS_THRESHOLD_HARD` ในคลาส `MyUtil`

2) การคำนวณเมทริกซ์การหมุน

เมื่อมีข้อมูลจากเซ็นเซอร์ส่งเข้ามา โปรแกรมจะเรียกเมท็อด `updateOrientation` เพื่อคำนวณทิศทางของอุปกรณ์ ซึ่งเมื่อเมท็อดนี้เริ่มทำงานจะมีการคำนวณเมทริกซ์การหมุนโดยการเรียกใช้เมท็อด `getRotationMatrix` ของระบบปฏิบัติการ โดยส่งอาร์กิวเม้นต์เป็นค่าเวกเตอร์ความเร่งที่วัดได้จากเครื่องวัดความเร่งและค่าความเข้มสนามแม่เหล็กโลก ดังรายละเอียดในตารางที่ 3.11

ตารางที่ 3.11 ตัวแปรที่ใช้ในการคำนวณการหมุน

ตัวแปร	ความหมาย
`R`	เมทริกซ์การหมุน
`I`	เมทริกซ์ความเอียง (inclination matrix)
`gravity`	เวกเตอร์ความเร่งที่วัดได้จากเครื่องวัดความเร่ง
`geomagnetic`	ค่าความเข้มสนามแม่เหล็กโลก

เมทริกซ์ **R** และ **I** ที่คำนวณได้นั้นจะอยู่ในกรอบอ้างอิงที่มีแกน **X** ที่ได้จากการคำนวณผลคูณไขว้ระหว่างแกน **Y** และ **Z** ซึ่งจะเป็นเวกเตอร์ที่ตั้งฉากกับระนาบพื้นและชี้ไปทางทิศตะวันออกโดยประมาณ ดังนั้นขั้นตอนต่อไปคือการแปลงเมทริกซ์เมทริกซ์ **R** และ **I** ให้อยู่ในกรอบอ้างอิงที่ต้องการด้วยการเรียกใช้งานเมท็อด `remapCoordinateSystem` ของระบบปฏิบัติการแปลงระบบพิกัดใหม่โดยจะคำนวณเมทริกซ์การหมุนใหม่ตามแกน **XY** ที่ให้มา ตารางที่ 3.12 แสดงพารามิเตอร์ของเมท็อด `remapCoordinateSystem`

ตารางที่ 3.12 พารามิเตอร์ของเมท็อด `remapCoordinateSystem`

ตัวแปร	ความหมาย
`inR`	เมทริกซ์การหมุนอินพุต
`X`	แกน **X+**
`Y`	แกน **Y+**
`outR`	เมทริกซ์การหมุนเอ๊าต์พุต

หลังจากนั้นจึงนำเมทริกซ์การหมุนที่ได้มาแปลงให้เป็นเวกเตอร์การหมุนรอบแกนโดยใช้เมท็อด `getOrientation` ของคลาส `SensorManager` ซึ่งมีพารามิเตอร์สองค่าคือเมทริกซ์การหมุนอินพุตและแถวลำดับของค่าผลลัพธ์ โดยผลลัพธ์ที่คำนวณได้นั้นจะเป็นมุมรอบแกน **Z**, **X**, และ **Y** ตามลำดับ ดังรายละเอียดในตารางที่ 3.13 โดยผลลัพธ์จะได้เป็นมุมของทิศทางของอุปกรณ์ตามที่ต้องการ

ตารางที่ 3.13 พารามิเตอร์ของเมท็อด `getOrientation`

ตัวแปร	ความหมาย
`float[] R`	เมทริกซ์การหมุนอินพุต
`float[] values`	แถวลำดับของมุมทิศทาง: `values[0]`: มุมของการหมุนรอบแกน **X** `values[1]`: มุมของการหมุนรอบแกน **Y** `values[2]`: มุมของการหมุนรอบแกน **Z**

3.3 การแสดงภาพตวามเป็นจริงเสริม

งานวิจัยนี้ออกแบบให้ระบบอ่านค่าพิกัดเรือทั้งหมดที่อยู่ในระบบจากเครื่องแม่ข่ายโดยใช้การสื่อสารผ่านอินเทอร์เน็ตด้วย โดยใช้โพรโทคอลเอชทีทีพี (Hypertext Transfer Protocol : HTTP) และทีซีพี/ไอพี (Transmission Control Protocol/Internet Protocol : TCP/IP) เหมือนเว็บทั่วไป โปรแกรมบนสมาร์ทโฟนจะร้องขอข้อมูลตำแหน่งเรือทั้งหมดตามช่วงการหน่วงเวลาที่กำหนดไว้ โดยมีรายละเอียดการออกแบบโปรแกรมส่วนการแสดงภาพกราฟิกส์ตวามเป็นจริงเสริมดังนี้

3.3.1 การอ่านพิกัดเรือจากอินเทอร์เน็ต

ผู้วิจัยออกแบบคลาส `MyNetworkTask` เพื่อเป็นคลาสหลักของการรับส่งข้อมูลผ่านอินเทอร์เน็ต และเนื่องจากการสื่อสารในลักษณะนี้จำเป็นต้องทำงานหลายเธรดและทำงานแบบไม่ประสานจังหวะ (asynchronous) ดังนั้นผู้วิจัยจึงออกแบบให้คลาสนี้สืบทอดมาจากคลาส `AsyncTask` ซึ่งในการออกแบบผู้ใช้จะต้องโอเวอร์ไรด์ (override) เมท็อดอย่างน้อยสองเมท็อดคือ `doInBackground` และ `onPostExecute`

ตารางที่ 3.14 เมท็อดที่สำคัญของคลาส `MyNetworkTask`

เมท็อด	รายละเอียด
`onPreExecute`	จะถูกเรียกให้ทำงานก่อนการทำงานหลัง ใช้เพื่อเตรียมหรือสร้างวัตถุต่าง ๆ ที่จำเป็นในการใช้งานในรอบนั้น
`doInBackground`	เธรดพื้นหลังนี้จะทำงานทันทีเมื่อเมท็อด `onPreExecute` เสร็จการทำงาน ซึ่งการทำงานของฟังก์ชันพื้นหลังนี้อาจจะใช้เวลานานก็เป็นได้ พารามิเตอร์ที่ถูกส่งมาจะถูกนำไปใช้ในขั้นตอนนี้ และผลการทำงานจะคืนกลับมาให้เมื่อทำงานเสร็จ
`onProgressUpdate`	ทำงานเมื่อมีการปรับปรุงการทำงาน
`onPostExecute`	ทำงานหลังจากเมท็อด `doInBackground` จบการทำงาน

โดยเมื่อคลาสนี้เริ่มทำงานจะมีเหตุการณ์เกิดขึ้นสี่ขั้นตอน แต่ละขั้นตอนจะถูกเรียกโดยฟังก์ชั่นเรียกกลับ `onPreExecute`, `doInBackground`, `onProgressUpdate` และ `onPostExecute` ตามลำดับ ดังรายละเอียดในตารางที่ 3.14 โดยเมท็อด `doInBackground` จะเป็นเมท็อดหลักสำหรับการรับส่งข้อมูล ซึ่งจะใช้ยูอาร์แอล (Uniform Resource Locator : URL) และการเชื่อมต่อผ่านโพรโตคอลเอชทีทีพี สำหรับการดึงข้อมูลจากเครื่องแม่ข่าย โดยเตรียมค่าสำหรับการรับส่งข้อมูลนี้จะประกอบด้วยพารามิเตอร์ต่าง ๆ ดังแสดงในตารางที่ 3.15

ตารางที่ 3.15 พารามิเตอร์สำหรับการร้องขอข้อมูลผ่านอินเทอร์เน็ต

เมท็อด	รายละเอียด
`setReadTimeout`	ค่าหมดเวลารอ (timeout) หรือไทม์เอ๊าต์ สำหรับการอ่านข้อมูล ค่าโดยปริยายมีค่าเท่ากับ 3000 มิลลิวินาที
`setConnectTimeout`	ไทม์เอ๊าต์ของการเชื่อมต่อ ค่าโดยปริยายเท่ากับ 3000 มิลลิวินาที
`setRequestMethod`	รูปแบบของการร้องขอของโพรโทคอล GET
`setRequestProperty`	กำหนดคุณสมบัติของการร้องขอ
`setDoInput`	การร้องขอนี้มีการส่งข้อมูลอินพุตหรือไม่
`setUseCaches`	มีการใช้ข้อมูลแคชหรือไม่
`setDoOutput`	การร้องขอนี้มีข้อมูลตอบรับหรือไม่

หลังจากนั้นจึงเชื่อมต่อแล้วรอการตอบรับ (response) โดยจะมีการเรียกเมท็อด `onPostExecute` ให้ทำงานหลังจากการรับส่งข้อมูลเสร็จสิ้นแล้ว โดยจะตรวจสอบว่ามีข้อมูล `mDelegate` หรือไม่ ถ้ามีจะเรียกโปรแกรมหลักให้ทำงานเพื่อนำข้อมูลที่ได้รับไปปรับปรุงการแสดงผลต่อไป

3.3.2 การอ่านภาพจากกล้องของสมาร์ทโฟน

เมื่อทราบพิกัดของเรือแล้ว ขั้นตอนต่อไปคือการแสดงกราฟิกส์ของพิกัดเรือซ้อนบนภาพจริง ในกรณีของงานวิจัยนี้จะใช้ภาพจริงจากกล้องถ่ายภาพของสมาร์ทโฟน ซึ่งผู้วิจัยออกแบบให้โปรแกรมเรียกใช้งานกลุ่มคำสั่งจากคลังโปรแกรมคาเมราเอพีไอ (Camera API) เพื่ออ่านและแสดงภาพจากกล้องของสมาร์ทโฟน โดยข้อกำหนดของระบบปฏิบัติแอนดรอยด์กำหนดให้ผู้ใช้จะต้องขออนุญาตเพื่อการเข้าถึงกล้องและใช้งานกล้อง ซึ่งสามารถทำได้ด้วยการกำหนดค่าใน เมนิเฟสต์ดังภาพที่ 3.12

```
<uses-permission android:name="android.permission.CAMERA" />
<uses-feature android:name="android.hardware.camera" />
```

ภาพที่ 3.12 การขออนุญาตเข้าถึงกล้องของแอนดรอยด์

ตารางที่ 3.16 เมท็อดที่สำคัญของคลาส `CameraView`

เมท็อด	รายละเอียด
`surfaceCreated`	สร้าง `Surface` เสร็จแล้ว
`surfaceDestroyed`	`Surface` ถูกทำลาย
`surfaceChanged`	มีการเปลี่ยนแปลง `Surface`

ผู้วิจัยออกแบบคลาส `CameraView` เพื่อใช้เป็นคลาสหลักสำหรับการจัดการกล้อง โดยมีการออกแบบภายในตามแนวทางที่ Google (2017) แนะนำ มีเมท็อดที่สำคัญดังแสดงในตารางที่ 3.16 โดยการใช้งานกล้องในระบบปฏิบัติการแอนดรอยด์นั้นจะมีขั้นตอนหลักที่สำคัญดังนี้

1) การตรวจและเข้าถึงกล้อง

ขั้นแรกคือการตรวจสอบว่ามีกล้องพร้อมทำงานหรือไม่ด้วยการทดสอบการเรียกเมท็อด `open` ของคลาส `Camera` ซึ่งหากพบกล้องจะดำเนินการขั้นต่อไป หากไม่พบจะสิ้นสุดการทำงาน

2) การสร้างคลาสสำหรับการพรีวิว

โดยทั่วไปสมาร์ทโฟนอาจจะมีกล้องได้อย่างน้อยสองตัว คือกล้องหลักซึ่งทำหน้าที่ถ่ายภาพจากด้านหลังกล้อง และกล้องที่สองซึ่งทำหน้าที่ถ่ายภาพจากด้านหน้าของกล้อง งานวิจัยนี้ใช้กล้องด้านหลังในการทำงาน ซึ่งหากโปรแกรมตรวจสอบแล้วว่ากล้องอยู่ในสถานะว่างและพร้อมใช้งาน ก็จะทำการสร้างวัตถุของคลาส `Camera` จากนั้นสั่งเปิดกล้องแล้วส่งวัตถุกลับไปให้คลาสหลักนำไปใช้งานเป็นวิว (View) สำหรับนำภาพที่ได้จากกล้องมาแสดงผลต่อไป

3) การสร้างเลย์เอ้าต์สำหรับการพรีวิว

เมื่อมีการเชื่อมต่อกับกล้องเรียบร้อยแล้ว ลำดับต่อไปคือการเตรียมคอนโทรลสำหรับการนำภาพมาแสดง และเนื่องจากคลาส `CameraView` สืบทอดมาจากคลาส `SurfaceView` ดังนั้นจึงสามารถกำหนดใช้วัตถุของคลาสนี้เป็นเลย์เอาต์ (Layout) สำหรับการแสดงผลในโปรแกรมได้ทันที อย่างไรก็ตามผู้ใช้อาจจะมีการถือสมาร์ทโฟนในทิศทางต่าง ๆ และเพื่อให้การแสดงผลเห็นภาพได้ในมุมกว้าง งานวิจัยนี้จึงกำหนดให้โปรแกรมแสดงภาพจากกล้องโดยกำหนด ทิศทางการใช้งานไว้คงที่ที่ 90 องศา ซึ่งหมายถึงการทำให้สมาร์ทโฟนอยู่ในโหมดแนวนอน (landscape) ตลอดเวลา

4) การสร้างเมท็อดสำหรับการจัดการเหตุการณ์จากกล้อง

เมท็อดสำหรับการฟังเหตุการณ์จากกล้องที่จำเป็นเมท็อดแรกคือ `surfaceCreated` ที่จะมีการเตรียมพารามิเตอร์ต่าง ๆ ที่เกี่ยวข้องกับการแสดงผลให้พร้อมเสร็จแล้วจึงเริ่มการแสดงภาพทันที เมท็อดที่จะถูกเรียกเมื่อมีการเปลี่ยนแปลงคือ `surfaceChanged` ในกรณีนี้โปรแกรมจะพักการแสดงผลชั่วคราวเพื่อปรับปรุงพารามิเตอร์ให้เรียบร้อยแล้วจึงเริ่มการแสดงภาพใหม่ และเมท็อดสุดท้ายคือ `surfaceDestroyed` ซึ่งจะทำงานเมื่อมีสัญญาณสิ้นสุดโปรแกรม เมท็อดนี้จะยุติการแสดงภาพจากกล้อง ปิดการเชื่อมต่อ และล้างตัวแปรออกจากหน่วยความจำ

3.3.3 การแสดงกราฟิกส์สามมิติ

งานวิจัยนี้มีคลาสด้านการแสดงผลกราฟิกส์ที่สำคัญสำหรับการใช้งานสองคลาส ประกอบด้วยคลาส GLSurfaceView และคลาส GLSurfaceFView.Renderer ซึ่งคลาส GLSurfaceView เป็นคลาสสำหรับการแสดงผลหรือวิวเพื่อให้ผู้ใช้วาดกราฟิกส์ที่ต้องการด้วยกลุ่มคำสั่งที่คล้ายกับการใช้งาน SurfaceView โดยมีการขออนุญาตใช้งานดังแสดงในภาพที่ 3.16

```
<uses-feature android:glEsVersion="0x00020000"
android:required="true" />
```

ภาพที่ 3.13 การขออนุญาตเข้าถึงโอเพนจีเอล

ตารางที่ 3.17 เมท็อดที่สำคัญของคลาส MyOverlayCanvasRenderer

เมท็อด	รายละเอียด
MyOverlayCanvasRenderer	ตัวสร้าง
onSurfaceCreated	สร้าง Surface เสร็จ
setGLparams	เตรียมโอเพนจีเอล
initWorld	เตรียมข้อมูลวัตถุต่าง ๆ ในโปรแกรม
initShaders	เตรียมข้อมูลเฉดเดอร์
onDrawFrame	วาดภาพลงบนคอนโทรล
onSurfaceChanged	มีการเปลี่ยนแปลงบนคอนโทรล
renderWorld	เรนเดอร์ฉากหลัง
renderTrackedObjects	เรนเดอร์เรือแบบสองมิติ
updateTrackedObjects	ปรับปรุงข้อมูลเรือ
render3DModel	เรนเดอร์เรือแบบสามมิติ

สำหรับคลาส GLSurfaceFView.Renderer จะเป็นคลาสที่รวบรวมกลุ่มคำสั่งสำหรับการวาดกราฟิกส์ ซึ่งผู้ใช้จะต้องอิมพลิเมนต์เมท็อดหลักสามเมท็อด เมท็อดแรกคือ onSurfaceCreated เมท็อดนี้จะทำงานเมื่อมีการสร้าง surface เกิดขึ้นและใช้เมท็อดนี้สำหรับการเตรียมการต่าง ๆ ที่เกิดขึ้นก่อนการแสดงผลจริง เช่น การเตรียมตัวแปรสภาพแวดล้อมต่าง ๆ ของโอเพนจีเอล เป็นต้น เมท็อดที่สองคือ onDrawFrame ระบบปฏิบัติการจะเรียกเมท็อดนี้เมื่อมีการวาดหน้าจอเกิดขึ้น ใช้เมท็อดนี้สำหรับการและการวาดใหม่ (re-draw) เพื่อปรับภาพให้เป็นปัจจุบันตามความต้องการของผู้ใช้ และเมท็อดที่สามคือ onSurfaceChanged ซึ่งจะถูกเรียกให้ทำงานเมื่อ Surface มีการเปลี่ยนแปลง เช่น การที่ผู้ใช้ปรับเปลี่ยนการเอียงตัวของสมาร์ทโฟนจากแนวตั้งเป็นแนวนอน เป็นต้น

งานวิจัยนี้ได้รวบรวมความต้องการด้านกราฟิกส์สามมิติเหล่านี้มาสร้างเป็นคลาส MyOverlayCanvasRenderer โดยผู้วิจัยได้ออกแบบให้คลาสนี้เป็นคลาสที่สืบทอดต่อจากคลาส GLSurfaceView.Renderer โดยเน้นการวาดกราฟิกส์ด้วยการใช้เอพีไอจากคลาส GLES20 เป็นหลัก โดยมีการออกแบบเมท็อดที่สำคัญดังแสดงในตารางที่ 3.17

1) เมท็อด setGLparams

เมท็อดนี้จะใช้สำหรับการเตรียมตัวแปรต่าง ๆ ให้พร้อมใช้งาน เช่นการเตรียมเฉดเดอร์ (shader) และรายการเวอร์เท็กซ์สำหรับการวาดภาพพื้นหลังต่าง ๆ ในโปรแกรม โดยได้แยกการเตรียมการใช้งานโอเพนจีเอลไว้ในเมท็อด setGLparams ซึ่งจะเป็นการเตรียมข้อมูลการล้างหน้าจอ การเปิด/ปิดงานฟังก์ชั่นต่าง ๆ ของโอเพนจีเอล ดังตัวอย่างในภาพที่ 3.14

```
private void setGLparams() {
  ...
  GLES20.glClearColor(0.00f, 0.00f, 0.00f, 0.00f);
  GLES20.glEnable(GLES20.GL_BLEND);
  GLES20.glBlendFunc(GLES20.GL_SRC_ALPHA,
                     GLES20.GL_ONE_MINUS_SRC_ALPHA);
  GLES20.glFrontFace(GLES20.GL_CCW);
  ...
}
```

ภาพที่ 3.14 เมท็อด setGLparams

2) เมท็อด initWorld

เป็นเมธอดสำหรับเตรียมเวอร์เท็กซ์บัฟเฟอร์ (vertex buffer) สำหรับเตรียมข้อมูลสามมิติของวัตถุอื่น ๆ ที่จะถูกวาดประกอบในโปรแกรม เช่น เส้นกริดบนท้องฟ้าซึ่งจะใช้แสดงทิศทางให้ผู้ใช้ทราบและเส้นกริดแสดงระนาบพื้น (ground plane) รวมไปถึงการเตรียมบัฟเฟอร์สำหรับการวาดตำแหน่งเรือ และแบบจำลองเรือแบบสามมิติด้วย

3) เมท็อด initShaders

เมท็อดนี้ใช้สำหรับการเตรียมเฉดเดอร์ที่จะใช้ในการแสดงผล โดยมีตัวแปร mShaderProgram เป็นตัวแปรสำหรับการใช้งานเฉดเดอร์ และเนื่องจากโอเพนจีเอลมีเฉดเดอร์ให้เลือกปรับแต่งได้หลายแบบ ซึ่งบางแบบอาจจะไม่จำเป็นสำหรับงานวิจัยนี้ ดังนั้นผู้วิจัยนี้เตรียมเฉพาะเวอร์เท็กซ์เฉดเดอร์และเฟล็กเม้นต์เฉดเดอร์ โดยเก็บข้อมูลเฉดเดอร์ทั้งสองไว้ในตัวแปร mVertexShader และ mFragmentShader ตามลำดับ ในโปรแกรมหลักจะมีตัวแปรที่ใช้สำหรับการสื่อสารกับเฉดเดอร์คือตัวแปร MVPMatrix ซึ่งจะใช้บันทึกข้อมูลเมทริกซ์การแปลงของสมาร์ทโฟน, a_pos สำหรับเก็บค่าพิกัด และ a_color สำหรับบันทึกค่าสี

ภาพที่ 3.15 แสดงตัวอย่างการเตรียมการใช้งาน โดยเริ่มต้นโปรแกรมจะสร้างเฉดเดอร์โปรแกรม ด้วยการเรียกเมท็อด glCreateProgram จากคลาส GLES20 ตามด้วยการสร้างเฉดเดอร์ทั้งสองด้วยการเรียกเมท็อด create_shader เสร็จแล้วจึงนำเฉดเดอร์ที่สร้างได้กำหนดเข้าเฉดเดอร์โปรแกรม ทำการลิงค์ และกำหนดการใช้งาน ขั้นตอนต่อไปคือการเตรียมการเชื่อมโยง

ตัวแปรที่จะใช้สื่อสารระหว่างโปรแกรมหลักกับเฉดเดอร์ โดยการเรียกใช้เมท็อด glGetUniformLocation และ glGetAttribLocation

```
private void initShaders() {
  ...
  this.mShaderProgram = GLES20.glCreateProgram();
  this.mVertexShader =  this.create_shader(
                        GLES20.GL_VERTEX_SHADER,
                        this.vertex_shader_src);
  this.mFragmentShader = this.create_shader(
                        GLES20.GL_FRAGMENT_SHADER,
                        this.fragment_shader_src);
  GLES20.glAttachShader(this.mShaderProgram,
                        this.mVertexShader);
  GLES20.glAttachShader(this.mShaderProgram,
                        this.mFragmentShader);
  GLES20.glLinkProgram(this.mShaderProgram);
  GLES20.glUseProgram(this.mShaderProgram);
  ...
  this.gl_mat_mvp_loc = GLES20.glGetUniformLocation(
                        this.mShaderProgram,
                        "u_MVPMatrix");
  this.gl_vert_loc = GLES20.glGetAttribLocation(
                        this.mShaderProgram,
                        "a_pos");
  this.gl_vert_color_loc = GLES20.glGetAttribLocation(
                        this.mShaderProgram,
                        "a_color");
  this.gl_tex_coord_loc = GLES20.glGetAttribLocation(
                        this.mShaderProgram,
                        "a_texCoordinate");
  this.gl_tex_loc = GLES20.glGetUniformLocation(
                    this.mShaderProgram,
                    "u_texture");
  ...
}
```

ภาพที่ 3.15 การเตรียมการใช้งานเฉดเดอร์

4) เมท็อด onDrawFrame

เมื่อจำเป็นต้องวาดภาพบนหน้าต่าง โปรแกรมหลักจะส่งสัญญาณมาที่เมท็อด onDrawFrame ซึ่งเมธอดจะทำตามขั้นตอนหลักของโอเพนจีเอล คือ เริ่มด้วยการล้างภาพเดิมด้วยการลบข้อมูลในบัฟเฟอร์แสดงผล ซึ่งในที่นี้จะทำการลบทั้งบัฟเฟอร์สีและบัฟเฟอร์ความลึก หลังจากนั้นจึงตรวจสอบรูปแบบการแสดงผลว่าโปรแกรมจำเป็นต้องการวาดภาพแบบสองมิติหรือสามมิติแล้วจึงเรียกเมท็อดที่เกี่ยวข้องต่อไป

5) เมท็อด onSurfaceChanged

เมื่อขนาดหน้าต่างโปรแกรมมีการเปลี่ยนแปลง โปรแกรมหลักจะส่งสัญญาณเรียกกลับมายังเมท็อด onSurfaceChanged ซึ่งเมท็อดนี้จะนำความกว้างและความสูงของหน้าจอมาคำนวณอัตราส่วนของหน้าจอ จากนั้นจึงกำหนดกรอบการแสดงผลของโอเพนจีเอล ด้วย glViewport เสร็จแล้วจึงเตรียมเมทริกซ์กล้องด้วยการสั่งงานเมท็อด set3DProjection

6) เมท็อด renderWorld

การแสดงกราฟิกส์ประกอบเพื่อให้ผู้ใช้รับรู้ถึงทิศทางของกล้อง โดยแสดงเป็นกริดบนภาพฉากหลัง ซึ่งโปรแกรมจะทำด้วยการกำหนดเมทริกซ์ที่เกี่ยวข้องทั้งหมดให้เป็นเมทริกซ์เอกลักษณ์ เสร็จแล้วจึงเตรียมเมทริกซ์กล้องสำหรับการแสดงผลแบบสามมิติ ขั้นต่อไปคือการอ่านข้อมูลการเอียงตัวของกล้องจาก mSensors ซึ่งคืนผลเป็นแถวลำดับของการหมุนรอบแกน และเนื่องจากโอเพนจีเอลอีเอสมีสายท่อการทำงานแบบสมัยใหม่และไม่มีฟังก์ชันอรรถประโยชน์สำหรับการกำหนดกล้องให้เรียกใช้ งานวิจัยนี้จึงนำแถวลำดับเหล่านี้มาสร้างเป็นเมทริกซ์การหมุนรอบแกนที่เกี่ยวข้องได้เป็นเมทริกซ์ R_X, R_Y, R_Z ตามลำดับ แล้วจึงคำนวณเมทริกซ์การหมุน $R_X \times R_Y \times R_Z$ ด้วยการใช้ Array.copyof() เป็นตัวช่วยในการสำรองข้อมูลระหว่างการคูณเมทริกซ์ไว้ในตัวแปร gl_mat_temp เพื่อใช้เป็นที่พักข้อมูลการดำเนินการ $R_X \times R_Y$ แล้วจึงเก็บผลลัพธ์การแปลงขั้นสุดท้ายในตัวแปร mGLMatMV

หลังจากได้เมทริกซ์การแปลงแล้วจึงปรับปรุงเมทริกซ์การแสดงผลด้วยการคำนวณการคูณระหว่าง mGLMatProj กับ mGLMatMV เก็บในตัวแปร gl_mat_mvp แล้วส่งตัวแปรนี้ไปให้เฉดเดอร์ด้วยการเรียกเมท็อด glUniformMatrix4fv เมื่อจัดการเมทริกซ์การแปลงแล้วจึงวาดภาพเส้นกริดด้วยการเรียกเมท็อด render() จึงสิ้นสุดการทำงาน

7) เมท็อด renderTrackedObjects

เมท็อด renderTrackedObjects จะมีการเตรียมเมทริกซ์การแสดงผลสามมิติเช่นเดียวกับที่กล่าวมาแล้วในบทที่ 3 หลังจากนั้นจึงเป็นการเตรียมตัวแปรสำหรับเก็บพิกัดแบบสองมิติและสามมิติ โปรแกรมจะนำพิกัดสามมิติของวัตถุจาก getWorldCoorrds มาเก็บในตัวแปร $\mathbf{X}$ แล้วจึงแปลงเป็นพิกัดสองมิติบนหน้าจอด้วยการคำนวณพิกัดสองมิติ $\mathbf{x}$ จาก

$$\mathbf{x} = \mathbf{M}_{proj} \times \mathbf{M}_{mv} \times \mathbf{X} \qquad 3.27$$

เมื่อ $\mathbf{X}$ คือ พิกัดสามมิติ, คือ $\mathbf{M}_{proj}$ เมทริกซ์การฉาย, คือ $\mathbf{M}_{mv}$ เมทริกซ์การแปลง และ คือ $\mathbf{x}$ พิกัดสองมิติ ทั้งนี้โอเพนจีเอลมีการจัดการเมทริกซ์แบบคอลัมน์เป็นหลัก (column-major) ดังนั้นการคำนวณจริงจึงต้องมีการสลับลำดับการคูณให้เหมาะสม

หลังจากได้พิกัดสองมิติเบื้องต้นแล้วจึงแปลงเป็นพิกัดเอกพันธ์ (homogeneous) โดยทำให้ค่า (x, y, w) มีค่าเป็น $(x/w, y/w, 1.0)$ และเลื่อนพิกัดให้มีกรอบอ้างอิงตามระบบพิกัดหน้าจอแบบนอร์มัลไลซ์ (normalized screen coordinate) ให้เป็นระบบพิกัดปกติที่มีจุดกำเนิดที่กลางจอภาพและมีขอบเขตการแสดงผลเท่ากับความกว้างและความสูงของจอภาพ

8) เมท็อด `updateTrackedObjects`

เมท็อดนี้ใช้เพื่อการปรับปรุงข้อมูลพิกัดของเรือ เมท็อดนี้จะถูกเรียกให้ทำงานเมื่อคลาส `MyNetworkTask` ได้รับข้อมูลใหม่ โดยข้อมูลที่ส่งมาจะเป็นแถวลำดับของตัวแปรของคลาส `MyShipInfo`

เมท็อด `updateTrackedObjects` จะนำข้อมูลที่ได้รับมาสกัดและปรับปรุงพิกัดการแสดงผลของเรือแต่ละลำ ด้วยการอ่านค่า $\mathbf{X}_{user}$ ซึ่งเป็นตำแหน่งปัจจุบันของสมาร์ทโฟน แปลงให้เป็นพิกัดในระบบ UTM แล้วจึงคำนวณพิกัดบนหน้าจอของเรือแต่ละลำโดยนำพิกัด $\mathbf{X}_{ship}$ มาคำนวณพิกัดแบบสัมพัทธ์ (relative) $\mathbf{X}_{rel}$ จากตำแหน่งของกล้องจากสูตร

$$\mathbf{X}_{rel} = \mathbf{X}_{ship} - \mathbf{X}_{user} \quad 3.28$$

จากนั้นเปรียบเทียบระยะทางระหว่างผู้ใช้กับพิกัดเรือซึ่งในที่นี้ทำได้ด้วยการคำนวณขนาด (magnitude) ของเวกเตอร์ $\mathbf{X}_{rel}$ แล้วเปรียบเทียบระยะทางนี้กับค่าขีดแบ่งที่กำหนดไว้ล่างหน้า หากระยะทางนี้สูงกว่าขีดแบ่งแสดงว่าวัตถุอยู่ไกลจากกล้องมาก โปรแกรมจะไม่นำเรือลำนี้มาวาดบนหน้าจอ

หากพบว่าเรืออยู่ในระยะที่พอจะแสดงผลได้ โปรแกรมจะใช้พิกัด $\mathbf{X}_{rel}$ นี้ในการแสดงผลสามมิติ นอกจากนั้นยังจะเตรียมข้อมูลป้าย (label) เพื่อแสดงข้อมูลเรือ โดยโปรแกรมจะแสดงป้ายด้วยการใช้เทคนิคบิลบอร์ด (billboard) ซึ่งเป็นเทคนิคการแสดงวัตถุสามมิติให้มีด้านใดด้านหนึ่งหันหน้าเข้าหากล้องเสมอหรือหมายถึงการทำให้ระนาบของรูปสามเหลี่ยมนั้นขนานกับระนาบรับภาพของกล้อง งานวิจัยนี้แสดงตวามเป็นจริงเสริมของตำแหน่งเรือด้วยป้ายที่เป็นรูปสามเหลี่ยมที่มีมุมด้านหนึ่งอยู่บนระนาบพื้นและหันระนาบของรูปสามเหลี่ยมเข้าหากล้องเสมอ

โดยถ้ากำหนดให้มุมของรูปสามเหลี่ยมที่อยู่บนระนาบพื้น มุมซ้ายบนและมุมขวาบนมีค่าเป็น $\mathbf{X}_1 = (X_1, Y_1, Z_1)$, $\mathbf{X}_2 = (X_2, Y_2, Z_2)$ และ $\mathbf{X}_3 = (X_3, Y_3, Z_3)$ ตามลำดับ ถ้าผู้ใช้หรืออุปกรณ์มีตำแหน่งปัจจุบันเป็น $\mathbf{X}_{user}$ และมีมุม θ เป็นค่ามุมอะซิมุธหรือการหมุนของสมาร์ทโฟนรอบแกน $\mathbf{Y}$ ของโอเพนจีเอลหรือแกน $\mathbf{Z}$ ในโลกจริงแล้ว เราสามารถหมุนรูปสามเหลี่ยมนี้ให้หันเข้าหากล้อง และคำนวณพิกัดที่หมุนแล้ว $\tilde{\mathbf{X}}_1$, $\tilde{\mathbf{X}}_2$ และ $\tilde{\mathbf{X}}_3$ จากสมการที่ 3.27 ถึงสมการที่ 3.27

$$\ddot{X}_2 = -w \quad 3.29$$

$$\ddot{Y}_2 = 0.0 \quad 3.30$$

$$\ddot{X}_3 = w \quad 3.31$$

$$\ddot{Y}_3 = 0.0 \quad 3.32$$

$$X_2 = ((\ddot{X}_2 \cos(\theta)) + (\ddot{Y}_2 \sin(\theta))) + X \quad 3.33$$

$$Y_2 = ((-\ddot{X}_2 \sin(\theta)) + (\ddot{Y}_2 \cos(\theta))) + (-Y) \quad 3.34$$

$$X_3 = ((\ddot{X}_3 \cos(\theta)) + (\ddot{Y}_3 \sin(\theta))) + X \quad 3.35$$

$$Y_3 = ((-\ddot{X}_3 \sin(\theta)) + (\ddot{Y}_3 \cos(\theta))) + (-Y) \quad 3.36$$

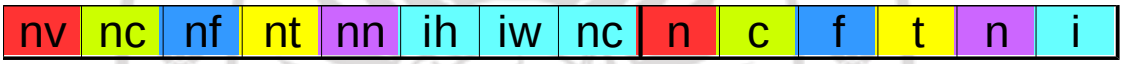
เมื่อจุดมุม $\tilde{\mathbf{X}}_1 = (X_1, 0.0, Z_1)$

3.3.4 ข้อมูลแบบจำลองเรือสามมิติ

หัวข้อที่ผ่านมาเป็นรายละเอียดเกี่ยวกับสร้างภาพตวามเป็นจริงเสริมด้วยการใช้การซ้อนข้อมูลเรือหรือป้ายชื่อของเรือลงบนภาพจริง ซึ่งเมื่อผู้ใช้แตะที่ป้ายชื่อเรือแล้วโปรแกรมจะแสดงแบบจำลองสามมิติของเรือให้ผู้ใช้เห็นบนหน้าจอ อย่างไรก็ตามรูปแบบแฟ้ม (file format) ของแบบ

จำลองสามมิติมีหลายแบบ มีทั้งที่บันทึกเป็นแฟ้มข้อความและแฟ้มไบนารี รูปแบบแฟ้มสำหรับแบบจำลองสามมิติที่นิยมใช้อย่างหนึ่งคือแฟ้มโอบีเจ (OBJ) ซึ่งเป็นรูปแบบแฟ้มที่พัฒนาโดยเวฟฟร้อนท์เทคโนโลยี (Wavefront Technologies) (Wolfram, 2018) มีรูปแบบเป็นแฟ้มข้อความซึ่งสามารถใช้โปรแกรมแก้ไขข้อความเปิดดูโครงสร้างภายในได้ง่าย ข้อมูลภายในถูกจัดเป็นกลุ่ม เช่น กลุ่มของพิกัดหรือเวอร์เท็กซ์ พิกัดภาพลายผิว เวกเตอร์ปกติ และข้อมูลเวอร์เท็กซ์ของรูปสามเหลี่ยม

โครงสร้างข้อมูลของแฟ้มโอบีเจเข้าใจได้ง่าย เขียนโปรแกรมเพื่ออ่านข้อมูลได้สะดวก อย่างไรก็ตามยังมีข้อมูลส่วนเกินที่ไม่จำเป็นกับงานวิจัยนี้ นอกจากนั้นการบันทึกข้อมูลเป็นแฟ้มข้อความนั้นอาจจะทำให้ประสิทธิภาพการทำงานในขั้นตอนการเตรียมแบบจำลองลดลงได้ ดังนั้นผู้วิจัยนี้จึงออกแบบรูปแบบแฟ้มขึ้นมาใหม่เพื่อให้เหมาะสมกับการใช้งานในครั้งนี้ โดยแปลงข้อมูลจากแฟ้มโอบีเจมาเป็นรูปแบบแฟ้มใหม่ซึ่งกำหนดส่วนขยายของแฟ้มเป็นเอ็มโอดี (MOD) ซึ่งการนำอักขระสามตัวแรกของคำว่าโมเดล (model) มาตั้งเป็นชื่อ โดยแฟ้มนี้จะแฟ้มแบบไบนารี่ที่มีพื้นที่ส่วนต้นของแฟ้มเอ็มโอดีจะเป็นข้อมูลส่วนหัว (header) บรรจุข้อมูลเกี่ยวกับขนาดหรือจำนวนของข้อมูลต่าง ๆ ที่จะอยู่ถัดจากข้อมูลส่วนหัว ภาพที่ 3.16 แสดงตัวอย่างข้อมูลภายในแฟ้มเอ็มโอดี



ภาพที่ 3.16 โครงสร้างข้อมูลแฟ้มเอ็มโอดี

1) ข้อมูลส่วนหัว

ข้อมูลส่วนหัวของแฟ้มเอ็มโอดีประกอบด้วยฟิลด์ทั้งหมดแปดฟิลด์ (ข้อมูลแปดชุดแรกในภาพที่ 3.16) ประกอบด้วยจำนวนเวอร์เท็กซ์ จำนวนสี จำนวนดัชนีเวอร์เท็กซ์สำหรับการวาดรูปสามเหลี่ยม จำนวนพิกัดภาพลายผิว จำนวนเวกเตอร์ปกติ ความกว้างของภาพลายผิว ความสูงของภาพลายผิว และจำนวนแบนด์ (band) ของภาพลายผิว ตามรายละเอียดในภาพที่ 3.18

ตารางที่ 3.18 ข้อมูลส่วนหัวของแฟ้มเอ็มโอดี

ชื่อฟิลด์	ความหมาย	ประเภท	ความยาว (ไบต์)
nv	จำนวนเวอร์เท็กซ์	int32	4
nc	จำนวนสี	int32	4
nf	จำนวนเวอร์เท็กซ์ของรูปสามเหลี่ยม	int32	4
nt	จำนวนพิกัดลายผิว	int32	4
nn	จำนวนเวกเตอร์ปกติ	int32	4
ih	ความสูงภาพลายผิว	int32	4
iw	ความกว้างภาพลายผิว	int32	4
nc	จำนวนแบนด์ภาพลายผิว	int32	4

2) ข้อมูลแบบจำลอง

ข้อมูลถัดจากข้อมูลส่วนหัวคือข้อมูลรายละเอียดของแบบจำลอง โดยแฟ้มเอ็มโอดีจะมีการเรียงลำดับข้อมูลเหล่านี้สัมพันธ์กับลำดับของข้อมูลในส่วนหัว นั่นคือจะเริ่มจากเวอร์เท็กซ์ ต่อด้วยข้อมูลสี ตามลำดับ หลังจากสิ้นสุดข้อมูลเวกเตอร์ปกติแล้วจะเป็นข้อมูลภาพลายผิว ซึ่งงานวิจัยนี้เลือกบันทึกข้อมูลเป็นแบบบีไอพี (band interleaved by pixel : BIP) (Japan Association on Remote Sensing, 1974) มีความละเอียดเชิงรังสี (radiometric resolution) เท่ากับ 32 บิตต่อพิกเซล นั่นคือข้อมูลสี่ไบต์แรกจะเป็นข้อมูลพิกเซลแรกของแบนด์สีแดง, สีเขียว, สีน้ำเงิน และอัลฟา (alpha) ตามลำดับ การจัดเรียงข้อมูลพิกเซลลักษณะนี้มีข้อดีคือสามารถนำไปใช้ต่อในโอเพนจีเอลได้ทันที ตารางที่ 3.19 แสดงรายละเอียดเพิ่มเติมเกี่ยวกับข้อมูลส่วนนี้

ตารางที่ 3.19 ข้อมูลส่วนท้ายของแฟ้ม `mod`

ชื่อฟิลด์	ความหมาย	ประเภท	ความยาว (ไบต์)
v	พิกัดเวอร์เท็กซ์	float32	sizeof(float32)*nv*3
c	สี (แดง, เขียว, น้ำเงิน)	float32	sizeof(float32)*nv*3
f	ดัชนีของเวอร์เท็กซ์ของสามเหลี่ยม	int32	sizeof(int32)*nv*3
t	พิกัดภาพลายผิว	float32	sizeof(float32)*nv*3
n	เวกเตอร์ปกติ	float32	sizeof(float32)*nv*3
i	ภาพลายผิว	uint8	sizeof(uint8)*iw*ih*nc

```
...
for(i = 0; i < (nv*3); i+=3) {
  _vertices[i+0] = 5.0f * fb.get(idx+0);
  _vertices[i+2] = 5.0f * fb.get(idx+1);
  _vertices[i+1] = 5.0f * fb.get(idx+2);

  if(_vertices[i+2] < minz) { minz = _vertices[i+2]; }

  // move to next point
  idx += 3;
}
...
```


ภาพที่ 3.17 การเตรียมบัฟเฟอร์ในเมท็อด `loadFromFile`

3) การวาดแบบจำลอง

งานวิจัยนี้บันทึกข้อมูลแบบจำลองทั้งหมดไว้ในโปรแกรม โดยใช้ตัวจัดการสินทรัพย์ (asset manager) ของแอนดรอยด์เป็นตัวช่วยในการจัดการ โดยผู้วิจัยออกแบบคลาส `ARModel` ซึ่งมีเมท็อด `loadFromFile` สำหรับการโหลดแบบจำลองเข้าสู่หน่วยความจำ โดยเมื่อเมท็อดเริ่มต้น

ทำงานจะตรวจสอบและโหลดแบบจำลองจากตัวจัดการสินทรัพย์ พร้อมทั้งเตรียมการอ่านแฟ้มแบบไบนารี หลังจากเปิดไฟล์แล้วจึงอ่านข้อมูลส่วนหัว เตรียมบัฟเฟอร์ แล้วจึงอ่านข้อมูลมาพักในบัฟเฟอร์ เสร็จแล้วจึงปิดไฟล์ ภาพที่ 3.17 แสดงตัวอย่างการอ่านข้อมูลไบนารีโดยใช้ `ByteBuffer` อ่านข้อมูล โดยงานวิจัยนี้กำหนดรูปแบบการเรียงไบต์แบบลิตเติลแอนเดียน (little endian) โดยมีการจำลองการอ่านข้อมูลเสมือนเป็นบัฟเฟอร์ของข้อมูลแบบ `float` จากนั้นจึงสกัดพิกัดสามมิติ (X, Y, Z) จากดัชนีลำดับที่ 0, 1 และ 2 ตามลำดับ เมื่อได้ข้อมูลมาแล้วจึงแปลงข้อมูลทั้งหมดให้ไปอยู่ในบัฟเฟอร์ของโอเพนจีเอลด้วยการเรียกใช้เมท็อด `buildFloatBuffer` จากคลาส `RenderUtils` ดังตัวอย่างในภาพที่ 3.18

```
...
FloatBuffer vb = RenderUtils.buildFloatBuffer(_vertices);
FloatBuffer cb = RenderUtils.buildFloatBuffer(_colors);
ShortBuffer ib = RenderUtils.buildShortBuffer(_indices);
FloatBuffer tcb = RenderUtils.buildFloatBuffer(_texCoords);
FloatBuffer nb = RenderUtils.buildFloatBuffer(_normals);
...
```

ภาพที่ 3.18 การแปลงข้อมูลเป็นบัฟเฟอร์ของโอเพนจีเอล