บทที่ 1
บทนำ

ในบทนำนี้กล่าวถึงเรื่องราวคร่าวๆที่เกี่ยวกับการค้นคว้าแบบอิสระนี้ เช่น ที่มาและหลักการต่างๆ

1.1 ที่มาและความสำคัญของปัญหาที่นำไปสู่การค้นคว้า

เกมคือการสั่งงานให้คอมพิวเตอร์สร้างภาพกราฟิกขึ้นมาแล้วรอให้ผู้เล่นมาเล่นตอบโต้กับภาพกราฟิกที่สร้างขึ้นมา ภาพกราฟิกตอบโต้ไม่ได้ การทำให้ภาพกราฟิกตอบโต้ได้นี้เป็นส่วนที่ผู้พัฒนาเกมต้องทำให้ได้ เป็นงานที่มีมากมาย งานที่มากนี้ทำให้การเขียนเกมเป็นเรื่องที่ยาก

เกมทั่วไปสร้างโดยการวนลูป ทำการรอรับอินพุตจากผู้เล่น ในขั้นตอนการวนลูป เกิดงานการรับอินพุต (Update Input) เมื่อกดคีย์บอร์ด ลูกศรขึ้นหน้า ถอยหลัง เลี้ยวซ้าย เลี้ยวขวา ต้องมีการปรับค่าของตำแหน่งผู้เล่นที่เกิดจากการรับอินพุต โดยให้ตำแหน่งของกล้องแทนตำแหน่งของผู้เล่น เกมมีลักษณะเหมือนผู้เล่นเดินด้วยตัวเอง เหมือนกล้องเป็นตำแหน่งการยืนของผู้เล่น ขอเรียกกล้องแทนผู้เล่น เมื่อผู้เล่นเปลี่ยนตำแหน่งก็คือการขยับตำแหน่งของกล้องนั่นเอง ตำแหน่งการมองของผู้เล่นก็คือตำแหน่งการมองของกล้อง เมื่อเกิดการเดินที่มีผลมาจากการรับอินพุตจึงเกิดงานการปรับเปลี่ยนตำแหน่งของกล้องและตำแหน่งการมองของกล้อง (Update Camera) การคำนวณตำแหน่งเหล่านี้จะใช้ฟังก์ชันตรีโกณมิติในการคำนวณ

ในแต่ละเกมมีวัตถุมากมาย ตัวละครก็ถือว่าเป็นวัตถุด้วย ทุกๆอย่างต้องมีพฤติกรรมในแบบที่เหมาะสม เช่น กำแพง เวลาที่ผู้เล่นเดินชนควรจะไม่มีการเดินผ่านไปได้ ความจริงแล้วทุกอย่างในเกมเป็นเพียงภาพวาดที่สร้างขึ้นมา ไม่ใช่วัตถุจริงๆ ถ้าไม่ได้ทำการทดสอบว่าชนกับกำแพงหรือไม่ ผู้เล่นก็จะเดินผ่านทะลุกำแพงไปได้ ทำให้เกิดงานการตรวจจับการชน (Collision Detection) วัตถุต่างๆในเกมก็ต้องมีการตอบสนองเพื่อความสมจริง

ตัวละครในเกมที่เป็นสิ่งมีชีวิตก็ควรมีปฏิกิริยาคล้ายสิ่งมีชีวิตจริง เช่น เมื่อตำแหน่งของกล้องอยู่ด้านหน้าของวัตถุ วัตถุนั้นๆ ถ้าเป็นสิ่งมีชีวิตต้องมีการมองเห็นกล้อง ผู้พัฒนาเกมต้องรู้ให้ได้ว่าด้านไหนเป็นด้านหน้าของวัตถุ และต้องรู้ว่าระยะห่างระหว่างกล้องกับวัตถุใกล้ไกลพอที่จะมองเห็นหรือไม่ ถ้าควรที่จะมองเห็นก็ต้องรู้ว่าวัตถุมองเห็นและทำการใส่ปฏิกิริยาให้กับวัตถุต่อไป การรู้ว่าด้านไหนเป็นด้านหน้า ด้านหลังต้องใช้เวกเตอร์และสูตรที่เกี่ยวข้องมาคำนวณ ส่วนการได้

ยินอาจไม่ต้องดูด้านหน้า ด้านหลังแต่ต้องการระยะทางว่าอยู่ในรัศมีการได้ยินหรือไม่ เมื่อตำแหน่งของกล้องอยู่ในรัศมีการได้ยินวัตถุนั้นๆ ก็ต้องมีการได้ยิน และใส่ปฏิกิริยาของการได้ยินต่อไป ทำให้เกิดงานการสร้างส่วนที่ทำให้เหมือนวัตถุในเกมมีชีวิตชีวา เรียกว่าปัญญาประดิษฐ์ของเกม (Artificial Intelligence)

สิ่งที่ผู้เล่นเกมเห็นบนหน้าจอทั้งหมดนี้เป็นเพียงภาพวาด เมื่อมีการรับอินพุตแล้วก็จะลบรูปที่วาดไว้ที่เฟรมที่แล้วและทำการวาดรูปวัตถุตามตำแหน่งของกล้องออกมาให้ผู้ใช้ได้เห็นทางหน้าจออีกครั้ง ทำให้เกิดงานการวาดรูปกราฟิก (Render) โดยใช้ฟังก์ชันในไดเรกเอกซ์ (Direct X) วาดรูป ไดเรกเอกซ์ มีฟังก์ชันมากมายแต่ละฟังก์ชันมีพารามิเตอร์มากและยากต่อใช้งานและทำความเข้าใจ

ทุกๆ เกมนั้นจะมีวัตถุมากมายการจะตรวจจับการชนกับวัตถุหรือวาดรูปวัตถุทั้งหมดที่มีในเกมทุกๆ ครั้งที่เปลี่ยนตำแหน่งของกล้องทำให้เครื่องคอมพิวเตอร์ทำงานหนัก ถ้าตรวจจับการชนกับวัตถุและวาดรูปวัตถุเฉพาะบางวัตถุหรือเฉพาะบางส่วนของเกมได้จะทำให้ลดงานของเครื่องคอมพิวเตอร์ลงได้มาก ทำให้เกิดงานการจัดการวัตถุในเกม งานนี้จะจำกัดการตรวจจับการชนและการจำกัดการวาดรูปวัตถุ โดยการทำให้เป็นตารางว่าวัตถุไหนอยู่ช่องไหนแล้วรับค่าว่าตำแหน่งของกล้องมาคำนวณว่าอยู่ช่องไหนแล้วตรวจจับการชนเฉพาะช่องนั้นๆ เป็นการจำกัดการตรวจจับการชน วาดรูปวัตถุเฉพาะช่องนั้นๆ และช่องโดยรอบเพื่อลดการวาดรูปวัตถุในเกมเป็นการจำกัดการวาดรูปวัตถุ เรียกงานนี้ว่าการจัดการกับวัตถุในเกม (Game Object Management)

เกมที่เล่นแบบหลายคน (Multi-Players) ต้องมีการส่งข้อมูลเชื่อมต่อกันจึงต้องมีงานที่เกี่ยวกับเน็ตเวิร์คเรียกงานนี้ว่า การติดต่อสื่อสาร (Network Communication) การติดต่อสื่อสาร สิ่งที่ต้องทำคือการเตรียมฟังก์ชันไว้สำหรับให้ผู้พัฒนาเกมเรียกใช้ในการติดต่อสื่อสาร โดยจะทำการสร้างเป็นคลาสขึ้นมาแล้วแต่ละฟังก์ชันในคลาสจะทำการเรียกใช้งานฟังก์ชันจากซ็อกเกตเอพีไอ (socket API) เช่นฟังก์ชัน bind() listen() accept() connect() send() และ recv() เพื่อให้ผู้พัฒนาเกมสะดวกในการนำไปใช้ในการส่งข้อมูลติดต่อสื่อสารระหว่างเครื่องคอมพิวเตอร์ต่างๆ ที่เข้ามาเล่นเกมร่วมกันผ่านทางเครือข่าย

ทั้งหมดนี้เป็นงานเพียงบางส่วนของการพัฒนาเกมที่ผู้พัฒนาเกมต้องทำ การพัฒนา และยังมีงานอื่นๆ อีกมากมาย ปัญหาของการสร้างเกมโดยไม่ใช้เฟรมเวิร์คเกมคือ ต้องทำงานเองทุกอย่างที่กล่าวมาข้างต้นแล้วและยังมีงานอื่นๆ อีกที่ยังไม่ได้กล่าวถึง ทำให้ผู้พัฒนาเกมต้องทำงานหลายอย่างมากในการสร้างเกมๆ หนึ่ง การนำเอางานต่างๆ ที่ต้องทำในการสร้างเกมมาพัฒนาเป็นเฟรมเวิร์คช่วยให้เกิดความสะดวกในการพัฒนาเกม ช่วยลดภาระของผู้พัฒนาเกมได้ ช่วยลดความซับซ้อนในการพัฒนาเกมและสามารถทำให้การพัฒนาเกมเกิดความรวดเร็วมากขึ้น

การศึกษาค้นคว้าอิสระนี้สนใจในเรื่องที่กล่าวมาข้างต้นคือการรับอินพุต (Update Input) การเปลี่ยนตำแหน่งการมองของกล้อง (Update Camera) การตรวจจับการชน (Collision Detection) ปัญญาประดิษฐ์ของเกม (Artificial Intelligence) การจัดการกับวัตถุในเกม (Game Object Management) การวาดรูปกราฟิก (Render) และการติดต่อสื่อสาร (Network Communication) ดังนั้นจึงนำเอาเรื่องเหล่านี้มาพัฒนาเป็นเฟรมเวิร์คของเกม

1.2 สาระสำคัญของเอกสารที่เกี่ยวข้อง

เนื้อหาที่จะกล่าวในส่วนนี้ประมวลสรุปมาจากเว็บไซต์วิกีพีเดีย http://en.wikipedia.org/wiki/Video_game

1.2.1 แพลทฟอร์ม

เกมในที่นี้ก็คือ วีดีโอเกม (Video Game) ซึ่งก็คือ เกมที่ใช้อุปกรณ์อิเล็กทรอนิกส์เป็นเครื่องเล่น (Electronic Game) เป็นการใช้อุปกรณ์ทางอิเล็กทรอนิกส์ตอบโต้กับผู้เล่น ระบบอิเล็กทรอนิกส์ที่ใช้เล่นเกมนี้เรียกว่า แพลทฟอร์ม มีอยู่หลายรูปแบบเช่น คอมพิวเตอร์ส่วนบุคคล (Personal Computer) หรือเครื่องคอนโซล (PC Video Game Consoles) เช่น เครื่องเพลย์สเตชัน(PlayStation) เครื่องเอกซ์บอกซ์ (Xbox)

คำว่าแพลทฟอร์มอาจหมายถึงอิเล็กทรอนิกส์หรือคอมพิวเตอร์ฮาร์ดแวร์ ซึ่งเชื่อมต่อกับซอฟต์แวร์เพื่อรับคำสั่งในการทำงาน

โดยทั่วไปแล้วเรียกเกมพีซี (PC Game) เป็นการอ้างอิงถึงเกมที่เล่นบนเครื่องคอมพิวเตอร์ส่วนบุคคลและเกมคอนโซล (Console Game) เป็นการอ้างอิงว่าเป็นเกมที่เล่นบนอุปกรณ์อิเล็กทรอนิกส์เฉพาะที่เชื่อมต่อกับเครื่องรับโทรทัศน์ ซึ่งเกมคอนโซล (Console Game) จะมีอุปกรณ์ในการรับอินพุตติดมาด้วย และยังมีอุปกรณ์อื่นๆ ที่สามารถที่จะเล่นเกมได้ แต่ไม่ได้ถูกออกแบบมาให้ใช้เล่นเกมโดยเฉพาะ เช่น คอมพิวเตอร์พกพา โทรศัพท์มือถือ

อุปกรณ์รับข้อมูล (Input Device) คืออุปกรณ์รับข้อมูล โดยทั่วไปถูกเรียกว่าตัวควบคุมเกม(Game Controller) ซึ่งหลากหลายไปตามแพลทฟอร์ม เช่น เครื่องเพลย์สเตชัน (PlayStation) อุปกรณ์รับข้อมูลจะมีการรับข้อมูลเพียงไม่กี่ปุ่ม เครื่องคอมพิวเตอร์ส่วนบุคคลอาจรับข้อมูลทางคีย์บอร์ดหรือเมาส์

1.2.2 รูปแบบของเกม

เกมสามารถแบ่งออกเป็นหลายรูปแบบตามลักษณะต่างๆ เช่น วิธีการเล่น เป้าหมายของการเล่นหรืออื่นๆ ส่วนใหญ่แล้วจะแบ่งตามเนื้อหาของการเล่น เกมอาจถูกแบ่งเป็นรูปแบบใหม่ๆ ได้อีก

ถ้ามีเกมรูปแบบใหม่ๆ ออกมา เพราะทุกๆ ปีเกมจะมีการสร้างให้มีการมองเห็นสิ่งต่างๆ มากขึ้นและมีความลึกในเนื้อหามากขึ้นเสมอ ในอุตสาหกรรมเกมก็มีความพยายามทำให้เกมมีความเหมือนกับชีวิตจริงมากขึ้น ส่งผลต่อความซับซ้อนของเกม ซึ่งทำให้การจัดแบ่งรูปแบบของเกมขยายออกมากขึ้นเรื่อยๆ โดยรวมแล้วแบ่งเป็น 8 รูปแบบใหญ่คือ

1. การต่อสู้ (Action)
2. การต่อสู้ในการผจญภัย (Action-Adventure)
3. การผจญภัย (Adventure)
4. การสร้างสิ่งก่อสร้าง (Construction and management simulation)
5. การจำลองชีวิตคน (Life simulation)
6. การเล่นเหมือนกับมีชีวิตในเกม (Role-playing)
7. การวางแผน (Strategy)
8. การจำลองอุปกรณ์เช่นการขับเครื่องบิน (Vehicle simulation)

1.2.3 การพัฒนาเกม

ผู้พัฒนาเกมก็คือพนักงานในอุตสาหกรรมเกม ในช่วงแรกๆ ประกอบด้วยโปรแกรมเมอร์ (Programmer) และ ผู้ออกแบบกราฟิก (Graphic Designer) ต่อมาได้ขยายออกตามความถนัด เช่น ผู้ออกแบบเสียง (Sound Designer) นักดนตรี และทางเทคนิคอื่นๆ ที่เกี่ยวข้อง ในปัจจุบันก็อาจมีการทำหน้าที่ทุกอย่างโดยไม่ได้แบ่งหน้าที่ก็มีแต่ส่วนมากเป็นระบบของแพลทฟอร์มเล็กๆ ที่ไม่ได้ออกแบบมาเพื่อการเล่นเกมโดยเฉพาะ เช่น โทรศัพท์มือถือ

ในการพัฒนาให้มีขนาดของทีมการพัฒนาเกมที่ใหญ่ขึ้นนี้ เป็นปัญหาในด้านค่าใช้จ่ายแต่ละที่ก็ต้องพยายามที่จะจ่ายเงินค่าตอบแทนในการทำงานให้มากพอ เพื่อรักษาผู้พัฒนาเกมที่มีความสามารถเอาไว้กับตนเอง และต้องพยายามลดต้นทุนค่าใช้จ่ายเพื่อเพิ่มกำไรด้วย โดยทั่วไป เกมคอนโซล (Console Game) จะมีพนักงานในทีมตั้งแต่ 5 ถึง 50 คน หรือบางทีอาจมีคนถึง 100 คน การมีทีมพัฒนาที่ใหญ่ก็เพิ่มความกดดันในการสร้างเกมที่สมบูรณ์ออกสู่ตลาด และความกดดันในการได้รับกำไรจากการขาย บางทีนำไปสู่การทำงานที่ไม่ทันต่อเวลา

1.2.4 เฟรมเวิร์คเกม

เฟรมเวิร์คเกมคือ ซอฟต์แวร์ หรือ ฟังก์ชัน ที่อำนวยความสะดวกในการเขียนเกม สามารถที่จะออกแบบมาใช้กับระบบปฏิบัติการหลากหลาย เช่น ลินุกซ์ (linux) แมคโอเอส (mac OS) ไมโครซอฟท์ วินโดวส์ (Microsoft Windows) ฟังก์ชันหลักๆ ของเฟรมเวิร์คเกมเป็นเรื่องการวาดรูป (Render) การตรวจจับการชน (Collision Detection) เสียง เน็ตเวิร์ค การจัดการหน่วยความจำ (Memory Management) และอื่นๆ

นอกจากการออกแบบมาให้ใช้งานในแต่ละระบบปฏิบัติการแล้ว ยังมีการออกแบบมาเพื่อเหมาะกับเครื่องมือการพัฒนา (Development Tools) ด้วย เช่น ไมโครซอฟท์ วิชวลสตูดิโอ (Microsoft Visual Studio) วิชวลซีพลัสพลัส (Visual c++) วิชวลซีชาร์ป (Visual c#) หรือแม้แต่ออกแบบมารวมๆ เพื่อให้ใช้กับภาษาใดภาษาหนึ่งก็มี เช่น ออกแบบสำหรับ ซีพลัสพลัส (c++) ซีชาร์ป (c#) จาวา (java)

การสร้างภาพทางหน้าจออาศัยฮาร์ดแวร์ (Hardware) ในการสร้าง ก่อนที่จะมีการ์ดแสดงผล (Graphic Card) ที่ทำงานได้เองเช่นหน่วยประมวลผลกราฟิก (GPU Graphic Processor Unit) ต้องอาศัยหน่วยประมวลผล (CPU Central Processor Unit) เป็นตัวสร้างภาพ เกมส่วนมากต้องการภาพที่สวยงาม การสร้างภาพที่สวยงามทำให้ต้องใช้รายละเอียดมากในแต่ละภาพ การมีรายละเอียดที่มากนี้ทำให้ฮาร์ดแวร์ ทำงานหนักและผู้ที่จะเขียนโปรแกรมให้วาดภาพได้นั้นต้องมีความรู้ในการเข้าถึงฮาร์ดแวร์ ซึ่งมีอยู่มากมาย เฟรมเวิร์คเกมก็เช่นเดียวกัน ในส่วนของการวาดภาพก็ต้องสามารถเข้าถึงฮาร์ดแวร์ ด้วย ความหลากหลายของฮาร์ดแวร์ จึงเป็นปัญหาในการเข้าถึง งานต่างๆ ที่เกี่ยวกับการวาดภาพในเกม เฟรมเวิร์คเกมส่วนใหญ่จะใช้ฟังก์ชันทางกราฟิก (Graphics API) เช่น ไดเรกเอกซ์ หรือ โอเพนจีแอล (OpenGL) ซึ่งจัดหาการเชื่อมต่อสำหรับหน่วยประมวลผลกราฟิก (GPU Graphic Processor Unit) หรือ การ์ดแสดงผล (Graphic Card)

ไลบรารี (Library) ระดับล่างของไดเรกเอกซ์ ก็สามารถเข้าถึงฮาร์ดแวร์อื่นๆได้ เช่น อุปกรณ์รับข้อมูล (Input Device) เช่น คีย์บอร์ด เมาส์ การ์ดเน็ตเวิร์ก(Network Card) การ์ดเสียง (Sound Card) และอื่นๆ

แนวโน้มในปัจจุบัน เทคโนโลยีก้าวหน้ามากขึ้น เฟรมเวิร์คเกมอาจทำในเรื่องอื่นๆ มากขึ้น เพิ่มขอบเขตการทำงานมากขึ้น เช่น การมุ่งเน้นในการสร้างภาพให้สมจริงมากขึ้น ใช้ในการสร้างโปรแกรมเกี่ยวกับการสอน เช่น การฝึกหัดขับขี่เครื่องบิน โปรแกรมเกี่ยวกับการแพทย์เช่นการจำลองการทำงานเกี่ยวกับอวัยวะต่างๆ โปรแกรมการจำลองสถานการณ์ทางทหาร

1.3 หลักการ ทฤษฎี เหตุผล และ/หรือสมมุติฐาน

ในหัวข้อนี้กล่าวถึงหลักการหรือวิธีการ ในการนำความรู้ที่ศึกษามาทำการพัฒนาเฟรมเวิร์คเกม การพัฒนาเฟรมเวิร์คแบ่งเป็น 7 ส่วนได้แก่ 1) การรับอินพุตของผู้เล่น (Update Input) 2) การปรับตำแหน่งของกล้อง (Update Camera) 3) การจัดการกับวัตถุในเกม (Game Object Management) 4) การตรวจจับการชน (Collision Detection) 5) ปัญญาประดิษฐ์ของเกม (Artificial Intelligence) 6) การวาดภาพกราฟิก (Render) และ 7) การติดต่อสื่อสาร (Network Communication) แต่ละส่วนมีหลักการดังนี้

1.3.1 การรับอินพุตของผู้เล่น (Update Input)

เนื่องจากในการค้นคว้าอิสระนี้ใช้ฟังก์ชันของไดเรกอินพุต (Direct Input) ในการสร้างฟังก์ชันสำหรับใช้รับข้อมูลของผู้เล่น จึงต้องใช้ความรู้ที่เกี่ยวกับฟังก์ชันของไดเรกอินพุต (Direct Input) มีฟังก์ชันที่สำคัญ เช่น

DirectInput8Create() ใช้สร้างตัวแปร *input*

GetDeviceState() รับค่าการกด

หลักการสำหรับสร้างเฟรมเวิร์คในส่วนนี้คือ การเตรียมสิ่งที่เกี่ยวกับฟังก์ชันที่ใช้ในการรับข้อมูลให้ผู้พัฒนาเกมเรียกใช้ การทำงานส่วนนี้ใช้วิธีการสร้างคลาสขึ้นมาให้มีฟังก์ชันสำหรับเรียกรับค่าการกดคีย์บอร์ดที่ผู้เล่นส่งเข้ามา การรับค่าการกดคีย์บอร์ดปุ่มต่างๆ จะทำโดยการเรียกใช้ฟังก์ชันจาก ไดเรกอินพุต (Direct Input) ซึ่งเป็นส่วนหนึ่งของไดเรกเอกซ์ (Direct X) และจะประกาศค่าคงที่ไว้สำหรับเรียกเทียบค่ารีเทิร์นมาจากการเรียกใช้ฟังก์ชัน โดยฟังก์ชันทั้งหมดจะทำให้มีการส่งพารามิเตอร์ (Parameter) ให้น้อยที่สุด เพื่อให้เกิดการเรียกใช้ที่ง่ายกว่าศึกษาและเรียกใช้โดยตรงจาก ไดเรกอินพุต (Direct Input) เหมือนกับการสร้างฟังก์ชันครอบไดเรกอินพุตไว้อีกที แล้วรวบรวมฟังก์ชันเหล่านี้ไว้ในคลาส *input* เมื่อผู้พัฒนาเกมเรียกใช้ทำได้โดยการประกาศออบเจ็กต์ และเรียกใช้จากฟังก์ชันของออบเจ็กต์นั้น

1.3.2 การปรับตำแหน่งของกล้อง (Update Camera)

ในการปรับตำแหน่งกล้องใช้สูตรของฟังก์ชันตรีโกณมิติในการหาตำแหน่งจุด (x, y, z) ของกล้อง สูตรที่ใช้คือสูตรของไซน์ (sine) และ โคไซน์ (cosine) สูตรคือ

ไซน์ (sine) = ความยาวด้านตรงข้ามมุม / ความยาวด้านตรงข้ามมุมฉาก

โคไซน์ (cosine) = ความยาวด้านประชิดมุม / ความยาวด้านตรงข้ามมุมฉาก

ในระบบ 3 มิติมีอยู่ 3 ระนาบคือ xy xz และ yz การศึกษาค้นคว้าอิสระนี้สนใจการเดินในระนาบ xz ซึ่งเหมือนการเดินบนพื้นดินของมนุษย์ การคำนวณหาตำแหน่งของกล้องในระนาบ xz ต้องมีมุมที่กระทำกับแกนวาย (y axis) หรือมุมรอบแกนวาย (y axis) ซึ่งแกนวาย (y axis) ก็คือแนวตั้งในระบบ 3 มิติ แต่การนับมุมอาจต่างออกไปคือเริ่มนับมุมที่ 90 องศา เท่ากับแทนมุม 90 องศาให้เป็น 0 องศา สูตรของการหาตำแหน่งของกล้องคือ

<u>สูตรการเดินหน้า</u>

x = x - ระยะทางที่เดินได้ * sinของมุมรอบแกน y

z = z - ระยะทางที่เดินได้ * cosของมุมรอบแกน y

<u>สูตรการถอยหลัง</u>

x = x + ระยะทางที่เดินได้ * sinของมุมรอบแกน y

z = z + ระยะทางที่เดินได้ * cosของมุมรอบแกน y

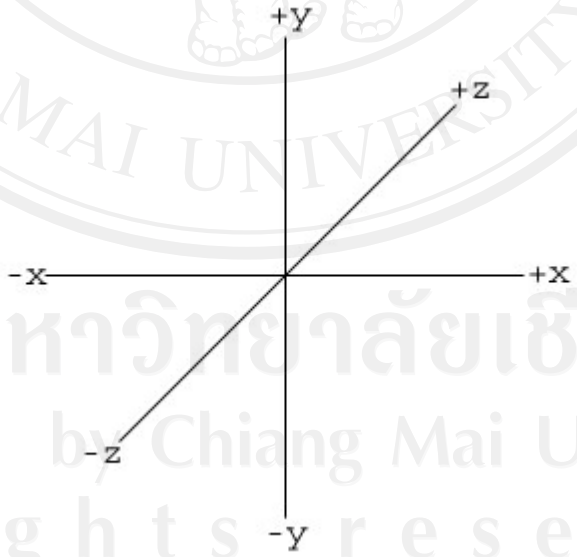
การหันซ้ายและขวาไม่มีการเปลี่ยนตำแหน่งของกล้องแต่มีการปรับ แต่จะปรับมุมที่หมุนรอบแกนวาย (y axis) โดยหันซ้ายมุมบวกเพิ่มเมื่อถึง 360 องศาให้ปรับเป็นศูนย์ใหม่ หันขวามุมลดลงเมื่อถึงติดลบให้ปรับเป็น 360 ตำแหน่งการมองให้ใช้สูตรเดียวกันหมดทั้ง เดินหน้า ถอยหลัง หันซ้าย หันขวา

<u>สูตรการหาตำแหน่งการมอง</u>

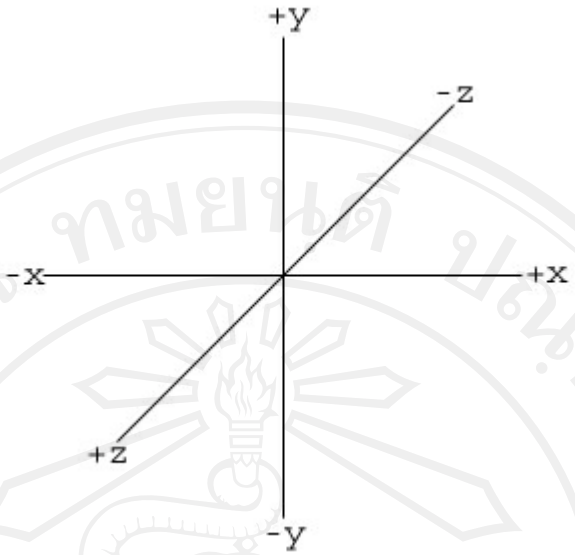
ตำแหน่งการมอง x = ตำแหน่ง x ของกล้อง - ระยะสายตา * sinของมุมรอบแกน y

ตำแหน่งการมอง z = ตำแหน่ง z ของกล้อง + ระยะสายตา * cosของมุมรอบแกน y

การศึกษาค้นคว้านี้สนใจเรื่องเกี่ยวกับการหาตำแหน่งในระนาบ xz ของระบบ 3 มิติ การปรับตำแหน่งของกล้อง คือ การเปลี่ยนตำแหน่งของกล้องไปตามที่ผู้เล่นกดคีย์บอร์ด และตำแหน่งของกล้องนี้จะแทนตำแหน่งของผู้เล่นด้วย ตำแหน่งการมองของกล้องจะแทนตำแหน่งการมองของผู้เล่นด้วยเช่นกัน ในระบบ 3 มิติ จะมีอยู่ 3 แกนคือ แกนเอกซ์ (x axis) แกนวาย (y axis) แกนแซด (z axis) ดังรูปที่ 1.1 และรูปที่ 1.2



รูปที่ 1.1 รูประบบมือซ้าย



รูปที่ 1.2 รูประบบมือขวา

จะมีอยู่ 2 ระบบ คือ ระบบมือซ้าย (รูปที่ 1.1) กับระบบมือขวา (รูปที่ 1.2) แกนวายเป็นแกนตั้งชี้ขึ้นข้างบนเป็นบวกชี้ลงล่างเป็นลบ แกนเอกซ์ เป็นแนวนอน ชี้ไปทางซ้ายเป็นลบ ชี้ไปทางขวาเป็นบวก เป็นแบบนี้ทั้งสองระบบ แต่ในแกนแซด ต่างกันคือระบบมือขวายิ่งไกลหน้าจอค่ายิ่งเป็นลบ ระบบที่ใช้เช่น โอเพนจีแอล ส่วนระบบมือซ้ายไกลหน้าจอค่าแกนแซดยิ่งเป็นบวก ระบบที่ใช้เช่น ไดเรกเอกซ์ ในการค้นคว้าแบบอิสระจะใช้ระบบมือซ้าย

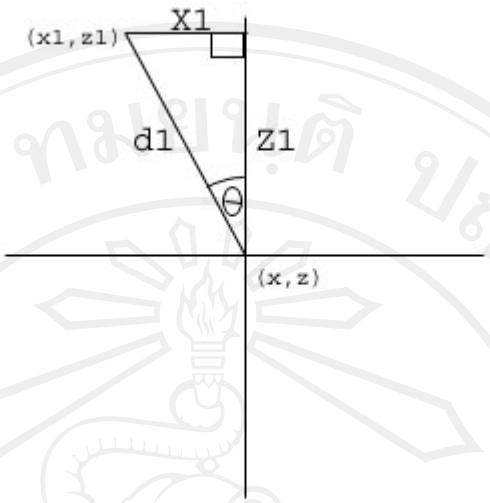
เมื่อมีการรับอินพุตจากผู้เล่น กล้องก็ต้องขยับเขยื้อนไปในทิศทางที่ตอบสนองต่อปุ่มนั้นๆ ตามที่ผู้พัฒนาเกมได้ตั้งไว้ การขยับกล้องให้ไปตำแหน่งที่ตอบสนองกับปุ่มนั้น ในระบบ 3 มิติจะใช้ความรู้ทางฟังก์ชันตรีโกณมิติมาคำนวณหาตำแหน่งของกล้อง
สูตรก็คือ

ไซน์ (sine)= ความยาวด้านตรงข้ามมุม / ความยาวด้านตรงข้ามมุมฉาก

โคไซน์ (cosine)= ความยาวด้านประชิดมุม / ความยาวด้านตรงข้ามมุมฉาก

เมื่อเดินไปข้างหน้า ข้างหลัง หรือหมุนซ้าย ขวา ข้อมูลที่ต้องการคือ มุมที่ทำกับแกนวาย (y axis) มีหน่วยเป็นเรเดียนโดยจะเก็บข้อมูลเป็นองศา เวลาใช้จะใช้ฟังก์ชันปรับเป็นเรเดียน
จะคำนวณบนระนาบ xz โดยไม่สนใจแกนวาย (y axis)

1) การเดินไปข้างหน้า



รูปที่ 1.3 การเดินไปข้างหน้า

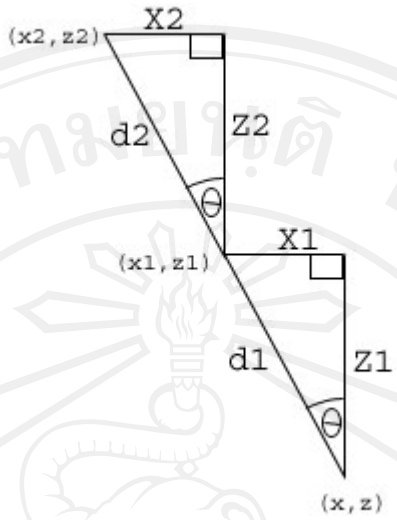
จากรูปที่ 1.3 ตำแหน่งปัจจุบันคือ (x, z) ตำแหน่งเมื่อเกิดการเดินหน้าคือ (x1, z1)
d1 คือ ระยะทางที่เดินได้
X1 คือ ระยะทางที่ต้องเปลี่ยนแปลงในแนวนอนหรือแกน x
Z1 คือ ระยะทางที่ต้องเปลี่ยนแปลงในแนวตั้งหรือแกน z ความจริงแล้วเป็นแนวลึกแต่ไม่พูดถึงแนวตั้งตามรูปที่ 1.3 แกน z จึงเป็นแกนตั้งแทน
ɵ คือ มุมที่กระทำกับแกน y หรือมุมที่หมุนรอบแกน y

จากสูตรฟังก์ชันตรีโกณมิติ
$\sin ɵ = X1/d1$
ดังนั้น $X1 = d1 \sin ɵ$

$\cos ɵ = Z1/d1$
ดังนั้น $Z1 = d1 \cos ɵ$

ดังนั้นตำแหน่ง (x1, z1) ซึ่งต้องเป็นตำแหน่งการยืนของกล้องในระนาบ xz คือ
$x1 = x - X1$
y = y คงที่เพราะเป็นการเดินบนระนาบ xz จึงไม่สนใจแนวตั้ง
$z = z + Z1$

เมื่อตำแหน่งของกล้องอยู่ที่ (x1, z1) แล้ว ต้องการหาตำแหน่ง (x2, z2) ที่กล้องมองไป



รูปที่ 1.4 ตำแหน่งการมองเมื่อเดินไปข้างหน้า

จากรูปที่ 1.4 ตำแหน่งการมองคือ (x2, z2)

d2 คือ ระยะสายตา

X2 คือ ระยะทางที่ต้องเปลี่ยนแปลงในแนวนอนหรือแกน x จากตำแหน่งของกล้อง

Z2 คือ ระยะทางที่ต้องเปลี่ยนแปลงในแนวตั้งหรือแกน z จากตำแหน่งของกล้อง

ɵ คือ มุมที่กระทำกับแกน y หรือมุมที่หมุนรอบแกน y

จากสูตรฟังก์ชันตรีโกณมิติ

$\sin ɵ = X2/d2$

ดังนั้น $X2 = d2 \sin ɵ$

$\cos ɵ = Z2/d2$

ดังนั้น $Z2 = d2 \cos ɵ$

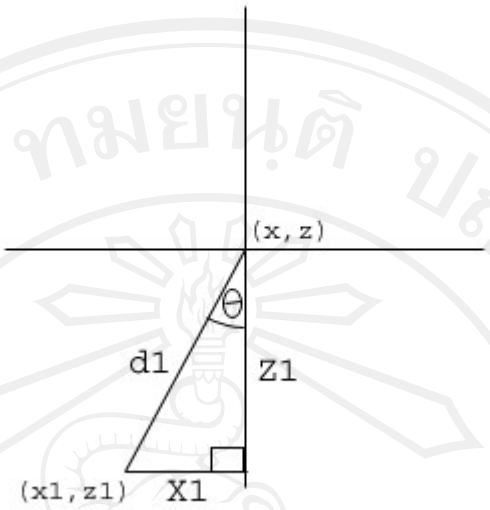
ดังนั้นตำแหน่ง (x2, z2) ซึ่งต้องเป็นตำแหน่งการมองของกล้องในระนาบ xz คือ

$x2 = x1 - X2$

$y = y$

$z2 = z1 + Z2$

2) การเดินถอยหลัง



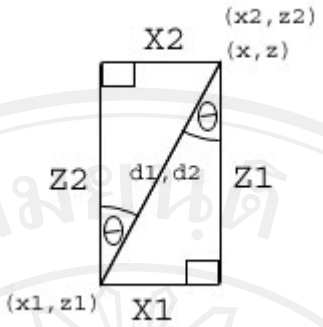
รูปที่ 1.5 การเดินไปข้างหลัง

จากรูปที่ 1.5 ตำแหน่งปัจจุบันคือ (x, z) ตำแหน่งที่เกิดการเดินถอยหลังคือ (x1, z1)
d1 คือ ระยะทางที่เดินได้
X1 คือ ระยะทางที่ต้องเปลี่ยนแปลงในแนวนอนหรือแกน x
Z1 คือ ระยะทางที่ต้องเปลี่ยนแปลงในแนวตั้งหรือแกน z
ɵ คือ มุมที่กระทำกับแกน y หรือมุมที่หมุนรอบแกน y

จากสูตรฟังก์ชันตรีโกณมิติ
$\sin ɵ = X1/d1$
ดังนั้น $X1 = d1 \sin ɵ$

$\cos ɵ = Z1/d1$
ดังนั้น $Z1 = d1 \cos ɵ$

ดังนั้นตำแหน่ง (x1, z1) ซึ่งต้องเป็นตำแหน่งการยืนของกล้องในระนาบ xz คือ
$x1 = x + X1$
$y = y$
$z = z - Z1$
เมื่อตำแหน่งของกล้องอยู่ที่ (x1, z1) แล้ว ต้องการหาตำแหน่ง (x2, z2) ที่กล้องมองไป



รูปที่ 1.6 ตำแหน่งการมองเมื่อเดินไปข้างหลัง

จากรูปที่ 1.6 ตำแหน่งการมองคือ (x2, z2)
d2 คือ ระยะสายตา
X2 คือ ระยะทางที่ต้องเปลี่ยนแปลงในแนวนอนหรือแกน x จากตำแหน่งของกล้อง
Z2 คือ ระยะทางที่ต้องเปลี่ยนแปลงในแนวตั้งหรือแกน z จากตำแหน่งของกล้อง
ɵ คือ มุมที่กระทำกับแกน y หรือมุมที่หมุนรอบแกน y

จากสูตรฟังก์ชันตรีโกณมิติ
$\sin ɵ = X2/d2$
ดังนั้น $X2 = d2 \sin ɵ$

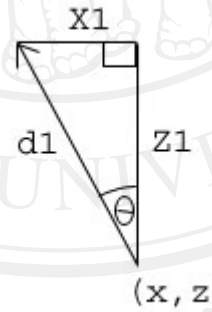
$\cos ɵ = Z2/d2$
ดังนั้น $Z2 = d2 \cos ɵ$

ดังนั้นตำแหน่ง (x2, z2) ซึ่งต้องเป็นตำแหน่งการมองของกล้องในระนาบ xz คือ
$x2 = x1 – X2$
$y = y$
$z2 = z1 + Z2$

3) การหันไปทางซ้าย

จากรูปที่ 1.7 กล้องอยู่ที่ (x, z) มองไปที่ (x1, z1)

รูปที่ 1.7 ตำแหน่งกล้องก่อนการหันไปทางซ้าย



รูปที่ 1.8 ตำแหน่งการมองเมื่อหันไปทางซ้าย

เมื่อหันไปทางซ้าย จากรูปที่ 1.8

d1 คือ ระยะสายตา

X1 คือ ระยะทางที่ต้องเปลี่ยนแปลงในแนวนอนหรือแกน x จากตำแหน่งของกล้อง

Z1 คือ ระยะทางที่ต้องเปลี่ยนแปลงในแนวตั้งหรือแกน z จากตำแหน่งของกล้อง

ɵ คือ มุมที่กระทำกับแกน y หรือมุมที่หมุนรอบแกน y

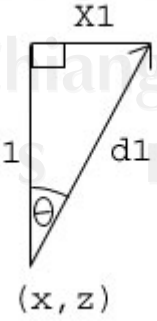
จากสูตรฟังก์ชันตรีโกณมิติ
$\sin \theta = X1/d1$
ดังนั้น $X1 = d1 \sin \theta$
$\cos \theta = Z1/d1$
ดังนั้น $Z1 = d1 \cos \theta$

ดังนั้นตำแหน่งของกล้องอยู่ที่เดิมคือ (x, z) แต่ตำแหน่งการมองจะเป็น (x1, z1) คือ
$x1 = x - X1$
$y = y$
$z = z + Z1$

4) การหันไปทางขวา

จากรูปที่ 1.9 กล้องอยู่ที่ (x, z) มองไปที่ (x1, z1)

รูปที่ 1.9 ตำแหน่งกล้องก่อนการหันไปทางขวา



รูปที่ 1.10 ตำแหน่งการมองเมื่อหันไปทางขวา

เมื่อหันไปทางขวา จากรูปที่ 1.10

d1 คือ ระยะสายตา

X1 คือ ระยะทางที่ต้องเปลี่ยนแปลงในแนวนอนหรือแกน x จากตำแหน่งของกล้อง

Z1 คือ ระยะทางที่ต้องเปลี่ยนแปลงในแนวตั้งหรือแกน z จากตำแหน่งของกล้อง

ɵ คือ มุมที่กระทำกับแกน y หรือมุมที่หมุนรอบแกน y

จากสูตรฟังก์ชันตรีโกณมิติ

$\sin ɵ = X1/d1$

ดังนั้น $X1 = d1 \sin ɵ$

$\cos ɵ = Z1/d1$

ดังนั้น $Z1 = d1 \cos ɵ$

ดังนั้นตำแหน่งของกล้องอยู่ที่เดิมคือ (x, z) ตำแหน่งการมองจะเป็น (x1, z1) คือ

$x1 = x - X1$

$y = y$

$z = z + Z1$

คลาส camera จะใช้สูตรจากข้อ 1) ถึงข้อ 4) สำหรับปรับตำแหน่งของกล้องและตำแหน่งการมองของกล้อง สูตรนำมาจากหนังสือ Developing Games in Java ในหน้า 407

1.3.3 การตรวจจับการชน (Collision Detection)

การตรวจจับการชนในการศึกษานี้ ใช้การหาระยะทางระหว่างจุดสองจุด

ให้จุดที่1อยู่ที่ตำแหน่ง (x1, y1, z1) จุดที่2อยู่ที่ตำแหน่ง (x2, y2, z2)

ระยะทางระหว่างจุดสองจุด = รากที่สองของ ($(x1-x2)^2 + (y1-y2)^2 + (z1-z2)^2$)

ถ้าเป็นแบบ 2 มิติก็ใช้สูตรเดียวกับแต่คำนวณเพียง 2 ค่า

ให้จุดที่1อยู่ที่ตำแหน่ง (x1, y1) จุดที่2อยู่ที่ตำแหน่ง (x2, y2)

ระยะทางระหว่างจุดสองจุด = รากที่สองของ ($(x1-x2)^2 + (y1-y2)^2$)

การตรวจจับการชน คือการหาว่า วัตถุ (Object) ในเกมมีการชนกันหรือไม่ มีวิธีหลักๆ คือ ตรวจจับการชน โดยสร้างอาณาเขตทรงกลม (Bounding Spheres) หลักการสำคัญก็คือสร้างรัศมี

จากจุดศูนย์กลางของวัตถุ การทดสอบระหว่างสองวัตถุจะใช้การคำนวณว่าวัตถุทั้งสองเข้าใกล้กันเกินกว่าระยะรัศมีของสองวัตถุรวมกันหรือไม่

วัตถุหนึ่งวัตถุในเกมจะมีจุดเวอร์เท็กซ์ (Vertex) มากมายที่ใช้อ้างอิงในการวาดรูป จุดต่างๆ เหล่านี้จะนำมาหาจุดมากสุดและน้อยสุดของแกนเอกซ์ (x axis) จุดมากสุดและน้อยสุดของแกนวาย (y axis) จุดมากสุดและน้อยสุดของแกนแซด (z axis) เมื่อได้แล้วจะหาจุดศูนย์กลางของวัตถุ จาก

centerX = (maxX – minX) / 2
centerY = (maxY – minY) / 2
centerZ = (maxZ – minZ) / 2
จุดศูนย์กลางของวัตถุ(x, y, z) = centerX , centerY , centerZ

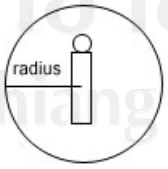
เมื่อได้จุดศูนย์กลางของวัตถุแล้วจะหารัศมีจากการลบกันระหว่างจุดน้อยสุดกับจุดศูนย์กลางในแต่ละแกน เพื่อนำมาคำนวณหาการชน
จากสูตรของระยะทางระหว่างจุดสองจุด

$$\text{ระยะทางระหว่างจุดสองจุด} = \text{รากที่สองของ} \left((x1-x2)^2 + (y1-y2)^2 + (z1-z2)^2 \right)$$

ดังนั้น

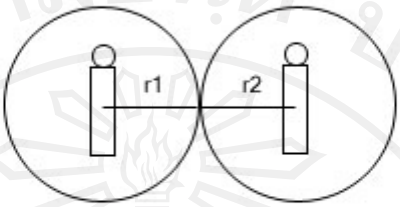
$$\text{รัศมี} = \text{รากที่สองของ} \left((centerX-minX)^2 + (centerY-minY)^2 + (centerZ-minZ)^2 \right)$$

จะได้ดังรูปที่ 1.11



รูปที่ 1.11 การสร้างอาณาเขตทรงกลมให้กับวัตถุ

รัศมีของวัตถุหนึ่งมาบวกกับรัศมีของวัตถุอีกวัตถุหนึ่งได้เป็นระยะทางที่ใกล้ที่สุดที่ไม่มีการชน



รูปที่ 1.12 ระยะทางที่ใกล้ที่สุดที่ไม่มีการชน

ดังนั้น

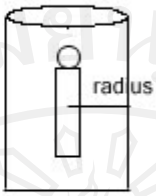
ระยะทางที่ใกล้ที่สุดที่ไม่มีการชน = รัศมีของวัตถุที่1 + รัศมีของวัตถุที่2

ถ้าดูจากรูปที่ 1.12 ก็คือ **ระยะทางที่ใกล้ที่สุดที่ไม่มีการชน = r1 + r2**

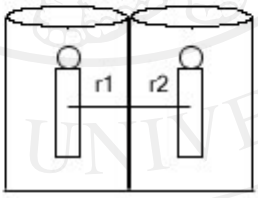
การตรวจจับการชนระหว่างวัตถุกับวัตถุจะทำได้โดย เมื่อวัตถุที่1 กับวัตถุที่2 เข้าใกล้กันเกินระยะทางที่ใกล้ที่สุดที่ไม่มีการชนถือว่ามีการชน การคำนวณหาระยะทางระหว่างวัตถุสองวัตถุคำนวณจากสูตรการหาระยะทางระหว่างจุดสองจุดเหมือนเดิม

ถ้าหากเป็นการตรวจจับการชนระหว่างกล้องกับวัตถุก็เปลี่ยนเป็นรัศมีของกล้องบวกกับรัศมีของวัตถุแทน

การตรวจจับการชนอีกแบบคือการตรวจจับการชนแบบทรงกระบอก วิธีเหมือนการตรวจจับการชนแบบทรงกลมแต่ใช้ค่า 2 มิติคือแกนเอกซ์ (x axis) และ แกนแซด (z axis) เท่านั้น สูตรการหาระยะทางจึงเป็นแบบ 2 มิติ การใช้สูตรแบบ 2 มิติเฉพาะค่า x และ z ทำให้การตรวจจับการชนเหมือนเป็นการสร้างวงกลมไว้ที่พื้น จึงเหมือนเป็นการทำทรงกระบอกรอบๆ วัตถุตามรูปที่ 1.13 และ 1.14



รูปที่ 1.13 การสร้างอาณาเขตทรงกระบอกให้กับวัตถุ



รูปที่ 1.14 ระยะทางที่ใกล้ที่สุดที่ไม่มีการชนของทรงกระบอก

1.3.4 ปัญญาประดิษฐ์ของเกม (Artificial Intelligence)

ปัญญาประดิษฐ์ในเกมจะช่วยให้วัตถุดูมีชีวิตขึ้นมา ความรู้ที่เกี่ยวข้องของเรื่องนี้ เป็นเรื่องเกี่ยวกับเวกเตอร์และสูตรการหาระยะทางระหว่างจุด 2 จุดเหมือนกับเรื่องตรวจจับการชน

สรุปสูตรดังนี้

สูตรการหานอร์มอลเวกเตอร์ (Normal Vector)

นอร์มอลเวกเตอร์ (Normal Vector) = (p1-p2) x (p3-p2)

p1 , p2 , p3 คือ เวอร์เท็กซ์ (Vertex) ตำแหน่งที่ 1 2 และ 3 ของ พื้นผิว (Polygon)

สูตรการหาค่า k

k = p1 * นอร์มอลเวกเตอร์ (Normal Vector)

p1 คือ เวอร์เท็กซ์ (Vertex) ตำแหน่งที่ 1 ของพื้นผิว (Polygon)

สูตรการหาด้านหน้า ด้านหลังระนาบ

Side = จุดใดๆ * นอร์มอลเวกเตอร์ (Normal Vector)

ถ้า

Side = k อยู่บนระนาบเดียวกัน

Side > k อยู่ด้านหน้า

Side < k อยู่ด้านหลัง

สูตรเกี่ยวกับการหาระยะทางระหว่างจุดสองจุดสูตรคือ

ระยะทางระหว่างจุดสองจุด = รากที่สองของ ($(x1-x2)^2 + (y1-y2)^2 + (z1-z2)^2$)

ปัญญาประดิษฐ์ของเกม ก็คือการพยายามทำให้เหมือนกับว่าตัวละครในเกมสามารถคิดเองได้ มีพฤติกรรมของตนเอง เพื่อให้ผู้เล่นมีความรู้สึกว่ามีปฏิกิริยาเกิดขึ้นกับการกระทำของผู้เล่น โดยจะพยายามสร้างเหมือนกับว่า ตัวละคร มีการมองเห็น ใช้การสร้างขอบเขตการมองเห็นเรียกว่า ระยะทางการมองเห็น (Vision Ray) และใช้ การตรวจจับการชน ส่วนการได้ยิน ใช้การสร้างขอบเขตการได้ยิน เรียกว่า รัศมีการได้ยิน (Hearing Radar) โดยกำหนดรัศมีขอบเขตการได้ยิน เมื่อผู้เล่นสร้างเสียงขึ้นมา ในขอบเขตของรัศมีการได้ยินก็ถือว่าได้ยิน ความจริงไม่ได้สร้างเป็นขอบเขตจริงๆ แต่การใช้รัศมีออกจากตำแหน่งของวัตถุไม่ว่าเสียงจะเกิดจากทิศทางไหน ถ้าเข้ามาในระยะรัศมีก็เหมือนกับเข้ามาในอาณาเขตการได้ยิน เหมือนกับว่าเป็นวงกลมขอบเขตการได้ยิน

การมองเห็น

การมองเห็นจะหาพื้นผิว (Polygon)ที่เป็นตัวแทนของวัตถุ โดยพื้นผิว (Polygon) จะมีเวอร์เท็กซ์ (Vertex) เป็นแบบทวนเข็มนาฬิกา โดยมีจำนวน 3 เวอร์เท็กซ์ (Vertex) คือ

เวอร์เท็กซ์ (Vertex) ที่ 1. (x, y, z) = (minX, minY, minZ)

เวอร์เท็กซ์ (Vertex) ที่ 2. (x, y, z) = (maxX, minY, minZ)

เวอร์เท็กซ์ (Vertex) ที่ 3. (x, y, z) = (minX, maxY, minZ)

เมื่อได้แล้วจะนำพื้นผิว (Polygon) ตัวแทนนี้มาคำนวณหานอร์มอลเวกเตอร์ (Normal Vector) ซึ่งเป็นเวกเตอร์ที่ตั้งฉากกับพื้นผิว (Polygon) นี้ แล้วจะถือว่านอร์มอลเวกเตอร์ (Normal Vector) นี้เป็นด้านหน้าของวัตถุ ดังรูปที่ 1.15

รูปที่ 1.15 เวกเตอร์ที่ตั้งฉากกับพื้นผิว (Polygon)

เมื่อได้ นอร์มอลเวกเตอร์ (Normal Vector) มาแล้วคำนวณหาค่า k จากสูตร

k = p1 * นอร์มอลเวกเตอร์ (Normal Vector)

p1 คือ เวอร์เท็กซ์ (Vertex) ที่ 1

กล้องจะมีตำแหน่ง แทนตำแหน่งของกล้องเข้าจุดใดๆ ตามสูตร

Side = จุดใดๆ * นอร์มอลเวกเตอร์ (Normal Vector)

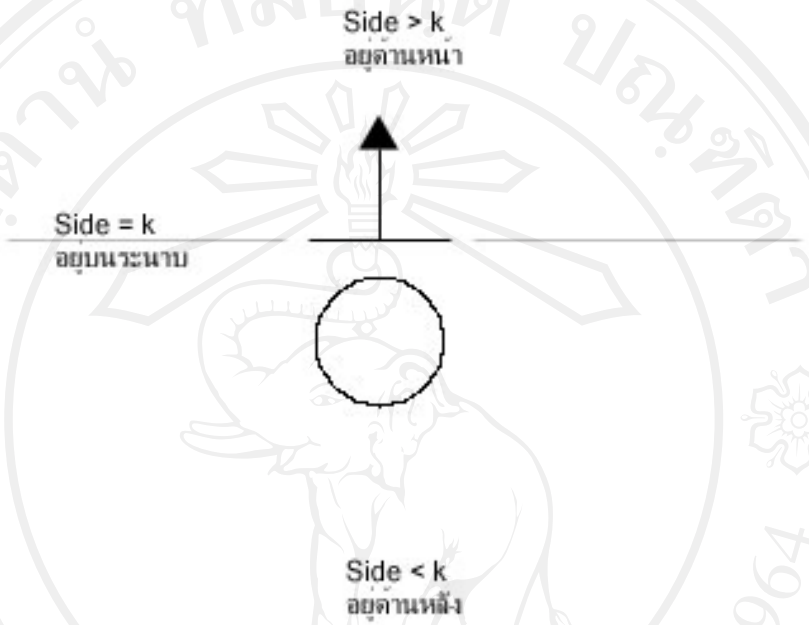
เทียบค่า k กับ ค่า Side

Side = k อยู่บนระนาบเดียวกัน

Side > k อยู่ด้านหน้า

Side < k อยู่ด้านหลัง

จะเป็นดังรูปที่ 1.16



รูปที่ 1.16 การวัดด้านหน้า ด้านหลัง

ระยะทางการมองเห็น (Vision ray) จะใช้สูตรหาระยะทางระหว่างจุดสองจุด

ระยะทางระหว่างจุดสองจุด = รากที่สองของ ($(x1-x2)^2 + (y1-y2)^2 + (z1-z2)^2$)

ระยะทางที่ไกลที่สุดที่มองเห็นจะกำหนดขึ้นมาเอง

การมองเห็นกล้องจากวัตถุกำหนดโดย ถ้ากล้องอยู่ด้านหน้าและระยะทางไม่ไกลกว่าระยะทางที่กำหนดถือว่ามองเห็น

การได้ยิน

การได้ยิน จะทำโดย หาระยะทางระหว่างกล้องกับวัตถุ ถ้าตำแหน่งของกล้องใกล้เกินรัศมีการได้ยินของวัตถุถือว่าได้ยินเสียง รูปที่ 1.17 เป็นรูปการไม่ได้ยิน ตำแหน่งของกล้องอยู่นอกอาณาเขตการได้ยินคือระยะทางระหว่างกล้องไกลเกินกว่ารัศมีการได้ยิน และรูปที่ 1.18 เป็นรูปการได้ยิน ตำแหน่งของกล้องอยู่ในอาณาเขตการได้ยินคือระยะทางระหว่างกล้องไม่เกินกว่ารัศมีการได้ยิน

รูปที่ 1.17 การไม่ได้ยิน

รูปที่ 1.18 การได้ยิน

1.3.5 การจัดการกับวัตถุในเกม (Game Object Management)

การจัดการกับวัตถุในเกม เป็นการตีกรอบหาพื้นที่ของเกม แล้วนำมาทำเป็นตารางและคำนวณว่าวัตถุอยู่ที่ช่องตารางใด วิธีการคำนวณที่เกี่ยวข้องคือ

ขนาดของตารางแต่ละช่องและจำนวนช่องตาราง

ขนาดของตารางแต่ละช่อง = 1<<9 = 512

จำนวนช่องด้านกว้าง = (ขนาดด้านกว้างของพื้นที่ในเกม>>9)+1

จำนวนช่องด้านสูง = (ขนาดด้านสูงของพื้นที่ในเกม>>9)+1

การจองพื้นที่หน่วยความจำ เป็นอาเรย์ 1 มิติมีขนาดเท่ากับ จำนวนช่องด้านกว้าง*จำนวนช่องด้านสูง

หลักการเรื่องการจัดการกับวัตถุในเกมจะใช้วิธีตีตารางเป็นช่องใน 2 มิติโดยไม่สนใจแนวตั้ง แกนวาย (y axis) โดยนำวัตถุทั้งหมดในเกมมาคำนวณหาค่า x และค่า z ที่น้อยที่สุดจากทุกวัตถุในเกมแล้วกำหนดให้เป็นจุดเริ่มต้น ต่อมาหาค่า x และ z ที่มีค่ามากที่สุดจากวัตถุทั้งหมด แล้วนำค่ามากสุด น้อยสุดเหล่านี้มาคำนวณ ได้ว่า

ความกว้าง (width) = ค่ามากสุดของ x – ค่าน้อยสุดของ x

ความยาว (height) = ค่ามากสุดของ z – ค่าน้อยสุดของ z

ดังรูปที่ 1.19

ข้อมูลค่ามากสุด น้อยสุดของ x และ z การตีตารางจะอยู่ในคลาส gridGame

รูปที่ 1.19 พื้นที่ทั้งหมดของเกม

เมื่อได้ค่าต่างๆเหล่านี้ก็จะได้ค่าเป็นสี่เหลี่ยมผืนใหญ่ที่สุดเหมือนเป็นแผนที่ ทุกวัตถุจะอยู่ในพื้นที่นี้แล้วแบ่งพื้นที่นี้เป็นช่องสี่เหลี่ยม สูตรที่ใช้ในการคำนวณจำนวนช่องคือ

จำนวนช่องด้านกว้าง(width) = (width >> 9) + 1

จำนวนช่องด้านสูง(height) = (height >> 9) + 1

ดังนั้น

จำนวนช่องทั้งหมด = จำนวนช่องด้านกว้าง * จำนวนช่องด้านสูง

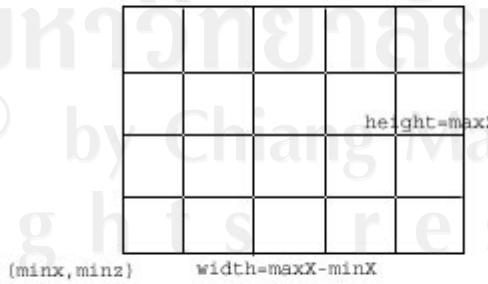
เนื่องจากใช้อาเรย์ 1 มิติ ในการเก็บช่องเหล่านี้การอ้างอิงช่องจึงต้องคำนวณโดยใช้สูตร

ช่องในด้านกว้าง + (ช่องในด้านสูง * จำนวนช่องด้านกว้าง)

ทุกด้านเริ่มต้นที่ 0 ตัวอย่างการคำนวณ เช่นจากรูปที่ 1.20 ช่องด้านล่างขวาคือ 4+(0*5) เท่ากับ 4 ช่องด้านซ้ายบนคือ 0+(3*5) เท่ากับ 15

รูปที่ 1.20 การนับช่อง

และในแต่ละช่องจะมีข้อมูลว่ามีวัตถุใดอยู่ในช่องบ้าง วัตถุจะอยู่ช่องไหนใช้วิธีวัดจากจุดศูนย์กลางของวัตถุนั้นๆ ตกอยู่ช่องไหนก็จะมีข้อมูลอยู่ช่องนั้น เมื่อมีการตีตารางแล้วจะเป็นเหมือนดังรูปที่ 1.21

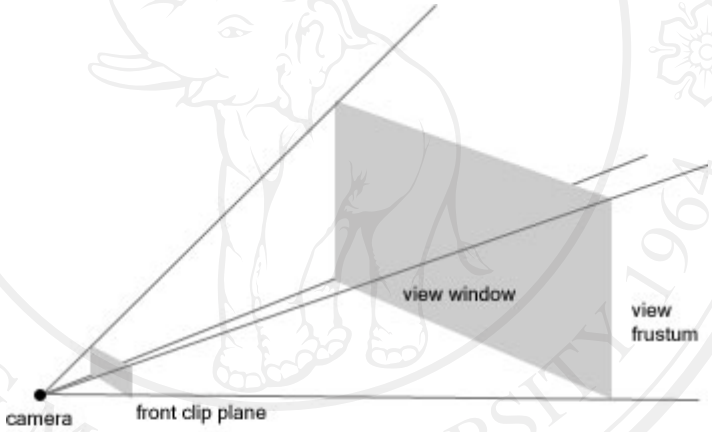


รูปที่ 1.21 พื้นที่ของเกมเมื่อนำมาตีเป็นตาราง

คลาสที่แทนช่องต่างๆในเรื่องนี้คือคลาส *cell* ใน *cell* จะมีอาเรย์ของวัตถุ อาเรย์จะเก็บข้อมูลว่ามีวัตถุอะไรอยู่ในช่องเหล่านี้บ้าง คลาส *gridGame* ก็มีอาเรย์ของ cell เหมือนกับว่า *gridGame* เป็นสี่เหลี่ยมผืนผ้า *cell* เป็นแต่ละช่องในสี่เหลี่ยมผืนผ้า การจัดการวัตถุในเกมนี้จะนำไปใช้ประโยชน์ในการจำกัดการตรวจจับการชน ให้ตรวจจับการชนเฉพาะบางช่องของ *gridGame* หรือตรวจจับเฉพาะบาง *cell* เฉพาะช่องที่ผู้เล่นอยู่หรือเฉพาะช่องที่กล้องอยู่นั่นเอง เท่ากับตรวจจับการชนเฉพาะบางวัตถุโดยไม่ต้องตรวจจับการชนกับทุกวัตถุในเกม รวมถึงการทดสอบการมองเห็น ได้ยินก็ทำเฉพาะช่องที่เป็นเป้าหมายได้เช่นกัน

1.3.6 การวาดภาพกราฟิก (Render)

การวาดภาพ 3 มิติต้องเกี่ยวข้องกับ จอภาพ การ์ดแสดงผลและอื่นๆ ขอกล่าวถึงหลักการทางคอมพิวเตอร์กราฟิกก่อนคือ



รูปที่ 1.22 รูประบบการมอง 3 มิติ

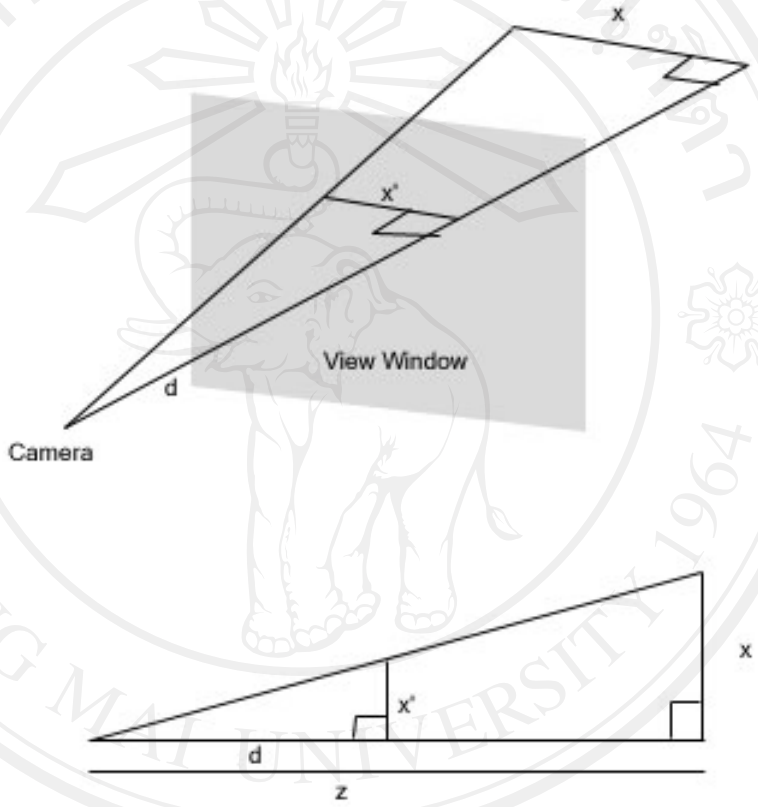
จากรูปที่ 1.22

camera คือจุดที่ยืนอยู่ของผู้เล่นหรือเป็นจุดที่มองนั่นเอง

front clip plane คือ หน้าต่างที่อยู่ใกล้ camera ที่สุดที่สามารถมองเห็นได้ บริเวณที่ใกล้กว่านี้ไม่สามารถมองเห็นได้

view window คือ หน้าต่างในโลก 3 มิติ จะมีรูปแบบเป็นแผ่น 2 มิติเหมือนหน้าจอคอมพิวเตอร์

ตามรูปที่ 1.22 ตำแหน่งจุดกำเนิดที่ (0, 0, 0) จะอยู่ตรงกลาง view window พื้นที่หลัง view window จะเรียกว่า view frustum การสร้างภาพ 3 มิติจะสร้างเป็นรูปร่าง 3 มิติบริเวณ view frustum การปรับรูป 3 มิติให้แสดงออกที่ view window เรียกว่า projection เนื่องจาก view window มีรูปแบบ 2 มิติ การ projection ออกมาที่ view window จึงต้องมีสูตรในการแปลงเป็น 2 มิติ



รูปที่ 1.23 รูปการคำนวณเพื่อแปลงเป็น view window

ตามรูปที่ 1.23

จะได้สูตรว่า

x' = dx/z

y' = dy/z

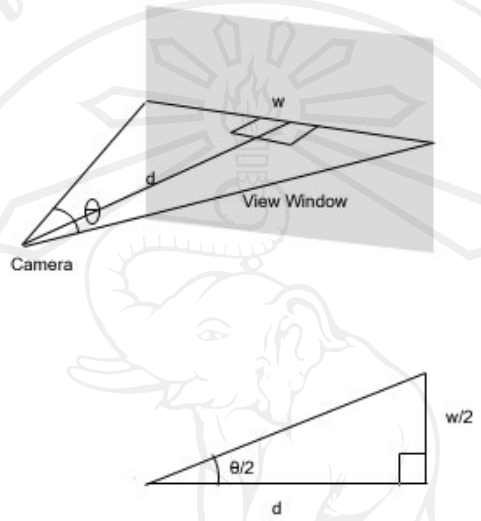
x' คือ ระยะของ x ในบริเวณ view frustum

y' คือ ระยะของ y ในบริเวณ view frustum

d คือ view distance เป็นระยะทางจาก camera ถึง view window

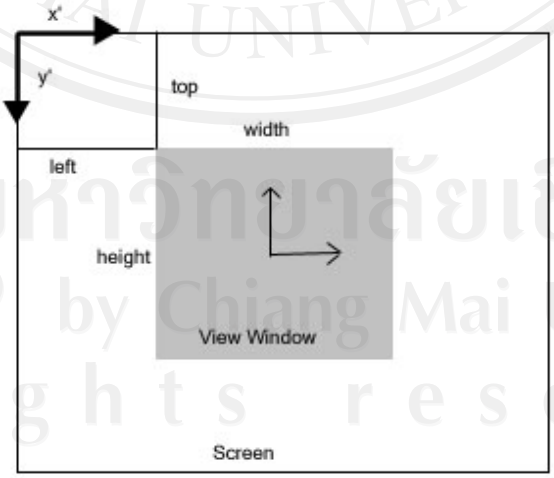
z คือระยะทางที่ลึกเข้าไปในหน้าจอ

view distance ใกล้ camera ก็สามารถมองเห็นได้ในมุมที่กว้าง แต่ถ้าไกลก็สามารถมองเห็นได้ในมุมที่แคบ ระยะทาง view distance นี้จะขึ้นอยู่กับมุม fov และ ความกว้างของview window ดังนั้น fov ก็คือมุมกว้างของ camera ที่ camera สามารถจับภาพได้ ย่อมาจาก Field of View ตามรูปที่ 1.24



รูปที่ 1.24 รูปการคำนวณ FOV

เมื่อได้ข้อมูล view window แล้วจะสามารถนำมาคำนวณด้วยสูตรออกทางหน้าจอคอมพิวเตอร์ได้จากรูปที่ 1.25



รูปที่ 1.25 รูปการปรับสู่หน้าจอ

ได้สูตรว่า

$$x' = left + width/2 + x$$
$$y' = top + height/2 + y$$

x’ คือ แนวนอนบนหน้าจอคอมพิวเตอร์

y’ คือ แนวตั้งบนหน้าจอคอมพิวเตอร์

left คือ ความต่างด้านแนวนอนที่หดลงของ view window เมื่อเทียบกับหน้าจอคอมพิวเตอร์

top คือ ความต่างด้านแนวตั้งที่หดลงของ view window เมื่อเทียบกับหน้าจอคอมพิวเตอร์

width คือ ความกว้างของ view window

height คือ ความสูงของ view window

ในการศึกษาค้นคว้าอิสระนี้ใช้ไดเรกเอกซ์ (DirectX) ช่วยในเรื่องข้างต้น ฟังก์ชันที่สำคัญคือ

Direct3DCreate8() ใช้สร้างตัวแปรไดเรกสามดีขึ้นมา

GetAdapterDisplayMode() ใช้รับรายละเอียดของจอภาพและการ์ดจอ

CreateDevice() ใช้สร้าง Direct 3D Device

createVertexBuffer() ใช้สร้าง Vertex Buffer

นอกจากนี้ยังมีการติดต่อกราฟิกของวินโดส์ (GDI Graphics Device Interface) โดยในการวาดข้อความออกทางหน้าจอ

ฟังก์ชันที่สำคัญคือ

SetTextColor() การกำหนดสีของข้อความ

SetBkColor() การกำหนดสีพื้นของข้อความ

SetBkMode() การกำหนดลักษณะพื้นหลังเช่นโปร่งแสง ทึบแสง

TextOut() การวาดข้อความ

หลักการที่ใช้ในเรื่องการวาดรูป ของการศึกษาค้นคว้าอิสระนี้ คือสร้างคลาสเพื่อเก็บข้อมูลในการวาดและทำการสร้างคลาสสำหรับวาดรูป โดยคลาสที่ใช้วาดรูปจะเรียกใช้ฟังก์ชันของไดเรกสามดี (Direct 3D) ทำการวาดรูปอีกที

ในเกมจะเกิดจากการใช้ เวอร์เท็กซ์ (Vertex) แทนจุดในโลก 3 มิติแต่ละเวอร์เท็กซ์ (Vertex) แทนหนึ่งจุด ในการศึกษาค้นคว้าอิสระนี้สร้างคลาส vector3D เพื่อแทนเวอร์เท็กซ์ (Vertex) เก็บข้อมูล x y z สี

นำแต่ละเวอร์เท็กซ์ (Vertex) มาประกอบรวมกันเป็นหนึ่งพื้นผิว (Polygon) การศึกษาค้นคว้าอิสระนี้สร้างคลาส polygon เพื่อเก็บข้อมูลพื้นผิว (Polygon) ในเกม ข้อมูลหลักคือ จำนวนเวอร์เท็กซ์ (Vertex) และข้อมูลของแต่ละเวอร์เท็กซ์ (Vertex)

แต่ละ polygon ก็จะประกอบกันเป็นหนึ่งวัตถุในเกม จุดและเส้นตรงก็ถือเป็นวัตถุซึ่งไม่ถือเป็น polygon แต่วัตถุอื่นๆ หนึ่งวัตถุต้องมีอย่างน้อย 1 พื้นผิว (Polygon) จุดและเส้นตรงก็สามารถใส่ไว้ในคลาส polygon ได้นั่นคือพื้นผิว (Polygon) ที่มี 1 เวอร์เท็กซ์ (Vertex) ก็คือจุด และ 2 เวอร์เท็กซ์ (Vertex) ก็คือเส้นตรง การศึกษาค้นคว้าอิสระนี้สร้างคลาส gameObject เพื่อเก็บข้อมูล gameObject ในเกม ข้อมูลหลักคือ จำนวนพื้นผิว (Polygon) และข้อมูลของแต่ละพื้นผิว (Polygon)

แล้วนำข้อมูลของวัตถุต่างๆ ซึ่งแต่ละวัตถุก็ประกอบด้วยหลายพื้นผิว (Polygon) นี้ มาวาดรูปออกทางหน้าจอ ในการศึกษาค้นคว้าอิสระนี้ใช้ไดเรกสามดี (Direct 3D) ช่วยวาดรูป โดยสร้างคลาส render มีฟังก์ชันอยู่ภายใน สำหรับใช้เรียกวาดรูป ฟังก์ชันเหล่านี้เรียกใช้ฟังก์ชันไดเรกสามดี (Direct 3D)

การวาดรูปของฟังก์ชันไดเรกสามดี (Direct3D) ต้องรับข้อมูลของเวอร์เท็กซ์ (Vertex) เข้ามาเป็น array ในแต่ละเวอร์เท็กซ์ (Vertex) จะสามารถเก็บข้อมูลได้หลายอย่าง สามารถที่จะบอกฟังก์ชันของไดเรกสามดี (Direct3D) แล้ววาดออกมาตามข้อมูลของเวอร์เท็กซ์ (Vertex) การวาดรูปของฟังก์ชันไดเรกสามดี (Direct3D) มีหลายแบบในฟังก์ชันเกี่ยวกับการวาดรูปสามารถระบุได้ ในการศึกษาค้นคว้าอิสระนี้จะทำฟังก์ชันให้วาดรูปได้ 3 แบบคือ 1) point lists 2) line lists และ 3) triangle lists

point lists จะรับข้อมูลเป็นเวอร์เท็กซ์ (Vertex) 1 จุด เพื่อวาดจุดหนึ่งจุด

line lists จะรับข้อมูลเป็นเวอร์เท็กซ์ (Vertex) 2 จุด เพื่อวาดเส้นตรงหนึ่งเส้น

triangle lists จะรับข้อมูลเป็นเวอร์เท็กซ์ (Vertex) 3 จุด เพื่อวาดรูปสามเหลี่ยมหนึ่งรูป

ฟังก์ชันที่ใช้ในการวาดรูปของไดเรกเอกซ์ (DirectX) จะวาดรูปลงไปในหน่วยความจำของการ์ดแสดงผลก่อน เมื่อทำการเรียกฟังก์ชันให้ทำการแสดงผลออกหน้าจอ จึงจะแสดงออกหน้าจอจริงๆ

ในการวาดข้อความ ใช้ดีไวซ์คอนเท็กซ์ ซึ่งเป็นส่วนการสร้างกราฟิกของระบบปฏิบัติการวินโดส์ (GDI Graphics Device Interface) โดยทำเป็นฟังก์ชัน ฟังก์ชันที่เกี่ยวกับการวาดรูปแสดงข้อความอยู่ในคลาส render

1.3.7 การติดต่อสื่อสาร (Network Communication)

การติดต่อสื่อสารทำได้ด้วยการใช้ฟังก์ชันทางเน็ตเวิร์คของซ็อกเกตเอพีไอ (socket API) ได้แก่ฟังก์ชัน bind() listen() accept() connect() send() recv() และเรียกใช้ฟังก์ชันของ winsock เพื่อสร้างการส่งแมสเสจกลับมาที่โปรแกรมที่ใช้ฟังก์ชันโดยการเรียกใช้ฟังก์ชันที่สามารถส่งแมสเสจมาให้โปรแกรมได้

ฟังก์ชันที่สำคัญต่างๆ มีรูปแบบเริ่มต้น(prototype) ดังนี้

int bind(SOCKET s, const struct sockaddr FAR* name, int namelen); ฟังก์ชันการสร้างพอร์ต (Port) ให้เซิร์ฟเวอร์ (Server)

int listen(SOCKET s, int backlog); ฟังก์ชันรอการเชื่อมต่อ

SOCKET accept(SOCKET s, struct sockaddr FAR* addr, int FAR* addrlen); ฟังก์ชันตอบรับการเชื่อมต่อ

int connect(SOCKET s, const struct sockaddr FAR* name, int namelen); ฟังก์ชันทำการเชื่อมต่อ

int send(SOCKET s, const char FAR *buf, int len, int flags); ฟังก์ชันทำการส่งข้อความ

int recv(SOCKET s, char FAR *buf, int len, int flags); ฟังก์ชันทำการรับข้อความ

WSAAsyncSelect (SOCKET s, HWND hWnd, unsigned int wMsg, long lEvent); ฟังก์ชันจาก winsock ทำการขอ event จากระบบ

หลักการในการสร้างเรื่องการติดต่อสื่อสาร คือทำการสร้างคลาส socketObject ขึ้นมา ในคลาสมีฟังก์ชันอยู่ 3 ส่วนคือฟังก์ชันส่วนของ เซิร์ฟเวอร์ ฟังก์ชันส่วนของ ไคลเอนท์ และฟังก์ชันส่วนที่ใช้งานร่วมกันระหว่าง เซิร์ฟเวอร์ กับ ไคลเอนท์

-ฟังก์ชันส่วนของ เซิร์ฟเวอร์

ฟังก์ชัน Bind() ทำการจัดตั้งพอร์ต ขึ้นมา

ฟังก์ชัน Listen() ทำหน้าที่คอยรอรับการเชื่อมต่อระหว่าง เซิร์ฟเวอร์ กับ ไคลเอนท์

ฟังก์ชัน Accept() ทำการรับการ connect ที่มาจาก ไคลเอนท์

-ฟังก์ชันส่วนของ ไคลเอนท์

ฟังก์ชัน Connect() ทำการ connect ไปที่ เซิร์ฟเวอร์

-ฟังก์ชันที่ใช้งานร่วมกันระหว่าง เซิร์ฟเวอร์ กับ ไคลเอนท์

ฟังก์ชัน Send() ทำหน้าที่ส่งข้อมูล

ฟังก์ชัน Recv() ทำหน้าที่รับข้อมูล

หลักการการทำงานของโปรแกรมทางเน็ตเวิร์คโดยทั่วไปเริ่มต้นจาก เซิร์ฟเวอร์ เรียกฟังก์ชัน bind() เพื่อตั้งพอร์ต ขึ้นมาก่อนต่อมาเรียก ฟังก์ชัน listen เพื่อรอฟังการ connect จากไคลเอนท์ เมื่อ ไคลเอนท์ เรียกฟังก์ชัน connect() ทำให้เกิดการเชื่อมต่อ เมื่อมีการเชื่อมต่อเข้ามาเซิร์ฟเวอร์ จะเรียกฟังก์ชัน accept() เพื่อรับการเชื่อมต่อนี้ เมื่อเสร็จสิ้นการเชื่อมต่อ เซิร์ฟเวอร์ กับไคลเอนท์ จะสามารถส่งและรับข้อมูลกันได้โดยเรียกฟังก์ชัน send กับ recv

การสร้างคลาสนี้ฟังก์ชันทั้งสามส่วนในคลาส สร้างโดยการเรียกใช้งานฟังก์ชันที่เกี่ยวกับเรื่องนั้นๆ จาก socket API คือ

Bind() ของคลาส เรียกใช้ bind() ของ socket API

Listen() ของคลาส เรียกใช้ listen() ของ socket API

Accept() ของคลาส เรียกใช้ accept() ของ socket API

Connect() ของคลาส เรียกใช้ connect() ของ socket API

Send() ของคลาส เรียกใช้ send() ของ socket API

Recv() ของคลาส เรียกใช้ recv() ของ socket API

1.4 วัตถุประสงค์ของการศึกษา

เพื่อพัฒนาเฟรมเวิร์คของเกมเตรียมไว้สำหรับเรียกใช้เมื่อพัฒนาเกมในด้าน

1) การรับอินพุตของผู้เล่น (Update input)

2) การปรับตำแหน่งของกล้อง (Update Camera)

3) การตรวจจับการชน (Collision Detection)

4) ปัญญาประดิษฐ์ของเกม (Artificial Intelligence)

5) การวาดรูปกราฟิก (Render)

6) การจัดการกับวัตถุในเกม (Game Object Management)

7) การติดต่อสื่อสาร (Network Communication)

1.5 ขอบเขตของการศึกษาค้นคว้า

1) การศึกษาวิธีการเขียนเกมเพื่อนำมาพัฒนาเฟรมเวิร์ค

ศึกษาไดเรกเอกซ์ ซึ่งประกอบด้วยฟังก์ชันหลายส่วนเช่น Direct3D DirectDraw DirectMusic DirectPlay DirectSound การศึกษาค้นคว้าอิสระนี้ศึกษาเฉพาะในส่วนของไดเรกสามดี (Direct3D) เพื่อนำมาทำเป็นคลาสโดยมีฟังก์ชันอยู่ภายในฟังก์ชันของคลาสจะเรียกใช้ฟังก์ชันในไดเรกสามดีเพื่อใช้ในการวาดรูปอีกที ศึกษาในส่วนของไดเรกอินพุต (Direct Input) เพื่อนำมาทำเป็นคลาสโดยมีฟังก์ชันอยู่ภายในฟังก์ชันของคลาสจะเรียกใช้ฟังก์ชันในไดเรกอินพุต

เพื่อใช้ในการรับอินพุตอีกที โดยศึกษาจากการดูตัวอย่างในหนังสือต่างๆ การใช้ฟังก์ชันในตัวอย่างต่างๆ ตัวอย่างการเขียนเกมที่ใช้ฟังก์ชันเหล่านี้

ศึกษาการนำสูตรทางคณิตศาสตร์มาใช้ในส่วนต่างๆ ของเกมโดยดูตัวอย่างจากหนังสือที่เกี่ยวกับการสอนเขียนเกมต่างๆ นำมาเขียนรวบรวมอยู่ในคลาสเดียวกันและมีฟังก์ชันที่เหมาะสมในแต่ละเรื่อง

ศึกษาการเขียนโปรแกรมด้วยฟังก์ชันทางเน็ตเวิร์คของ socket API ในเรื่องที่เกี่ยวกับการเขียนโปรแกรมได้แก่การเริ่มต้นสร้างพอร์ตด้วยฟังก์ชัน bind() การรอคอยการเชื่อมต่อด้วย listen() การเชื่อมต่อด้วย connect() การรับการเชื่อมต่อด้วย accept() การส่งข้อมูลด้วย send() การรับข้อมูลด้วย recv() และศึกษาการใช้ฟังก์ชันของ winsock ในเรื่องการขอให้ส่งแมสเสจมายังโปรแกรมที่เรียกใช้งาน จากตัวอย่างที่เกี่ยวกับการเขียนโปรแกรมทางเน็ตเวิร์ค เพื่อนำมาเขียนรวบรวมเป็นคลาส และวิธีการพัฒนาเกมแบบต่างๆ เพื่อนำมาปรับใช้ในการพัฒนาเฟรมเวิร์คของเกม

2) ทดสอบการทำงาน ขอบเขตอยู่ในเรื่อง ทดสอบการวาดภาพออกทางหน้าจอ ทดสอบการรับค่าอินพุตจากคีย์บอร์ด การทดสอบการเดินจากการรับค่าคีย์บอร์ด ทดสอบการชน ทดสอบการมองเห็น ทดสอบการได้ยิน จากการเรียกใช้คลาสและฟังก์ชันที่ทำขึ้นมา ขอบเขตการทดสอบเรื่องการติดต่อสื่อสารทำการ ทดสอบเรื่องการใช้คลาสที่สร้างขึ้นมาส่งข้อความระหว่างโปรแกรมบนเครื่องคอมพิวเตอร์เครื่องเดียวกัน การทดสอบทั้งหมดเพื่อดูว่าฟังก์ชันทำงานเป็นอย่างไรบ้าง

3) ปรับปรุงเฟรมเวิร์คของเกม เช่น ทำการโอเวอร์โหลดฟังก์ชัน เพื่อให้ฟังก์ชันรับค่าพารามิเตอร์ได้หลากหลายขึ้น ใช้งานได้สะดวกขึ้น

1.6 นิยามศัพท์

ในการดำเนินการค้นคว้าอิสระนี้ มีศัพท์ต่างที่เกี่ยวข้องและนิยามได้ดังนี้

-กริด (grid) การจัดการวัตถุในเกมแบบแบ่งเป็นช่องๆ

-อาณาเขตทรงกลม (Bounding Spheres) หมายถึง การสร้างขอบเขตเป็นทรงกลมสำหรับทดสอบการชน

-การตรวจจับการชน (Collision Detection) หมายถึง การสร้างอาณาเขตทรงกลมหรือทรงกระบอกขึ้นมาแล้วทดสอบว่ามีการซ้อนทับกันหรือไม่เพื่อแทนการชน

-ปัญญาประดิษฐ์ของเกม (Artificial Intelligence) หมายถึง การสร้างการมองเห็น การได้ยินให้กับวัตถุในเกม

-แกนเอกซ์ (x axis) หมายถึง แกน x ในระบบ3มิติ

-แกนวาย (y axis) หมายถึง แกน y ในระบบ3มิติ

-แกนแซด (z axis) หมายถึง แกน z ในระบบ3มิติ

-ตัวสร้าง หมายถึง constructor ของคลาส

-ตัวทำลาย หมายถึง deconstructor ของคลาส

-แมสเสจ หมายถึง Message ของวินโดส์ที่จะส่งมาแตกต่างกันออกไปตามแต่ event ที่ต่างกัน

-พอร์ต หมายถึง หมายเลขที่ใช้ในการติดต่อสื่อสาร (port)

-คีย์บอร์ด หมายถึง keyboard แป้นพิมพ์คอมพิวเตอร์

-โอเวอร์โหลด หมายถึง การสร้างฟังก์ชันที่มีชื่อเดียวกันแต่มีพารามิเตอร์ที่แตกต่างกันเพื่อสะดวกในการเรียกใช้งาน (Overload)

-รูปแบบเริ่มต้นของฟังก์ชัน หมายถึง prototype ของฟังก์ชัน

-พารามิเตอร์ หมายถึง ตัวแปรที่ส่งให้กับฟังก์ชัน

-ออบเจ็กต์ หมายถึง ตัวแปรที่มีชนิดเป็นคลาสต่างๆ

-ระยะทางการมองเห็น (Vision Ray) หมายถึง ระยะทางที่กำหนดขึ้นเพื่อวัดการมองเห็น

-รัศมีการได้ยิน (Hearing Radar) หมายถึง ระยะทางที่กำหนดขึ้นเพื่อวัดการได้ยิน

-หน่วยความจำของการ์ดแสดงผล หมายถึง พื้นที่ memory ของ graphics card

-ตัวแปรโครงสร้าง หมายถึง ตัวแปรชนิด struct ในภาษา c++

-ดีไวซ์คอนเท็กซ์ หมายถึง ส่วนการสร้างกราฟิกของระบบปฏิบัติการวินโดส์

1.7 ระเบียบวิธีการวิจัย

1) ออกแบบและพัฒนาเฟรมเวิร์คของเกม

ศึกษาไดเรกเอกซ์ สูตรทางคณิตศาสตร์และวิธีการพัฒนาเกมแบบต่างๆ

ออกแบบให้มี 11 คลาส คือ คลาส *vector3D, polygon, gameObject, camera, input, cell, gridGame, render, rectangles, socketObject, convert* แบ่งงานออกเป็นด้านต่างๆ คือ

-ด้านการรับอินพุตของผู้เล่น (Update Input) มีคลาส *input* เป็นคลาสหลักให้เคลาสนี้ทำหน้าที่ในการรับอินพุตจากการกดคีย์บอร์ดของผู้เล่น

-ด้านการปรับตำแหน่งของกล้อง (Update Camera) มีคลาส *camera* เป็นคลาสหลักให้คลาสนี้แทนตำแหน่งของผู้เล่นและใช้คำนวณหาตำแหน่งการมองของผู้เล่น คลาสนี้เก็บข้อมูลที่อยู่ของกล้อง ตำแหน่งการมองของกล้อง มุมที่หมุนรอบแกน y

-ด้านการจัดการกับวัตถุในเกม (Game Object Management) มีคลาส *gridGame, cell* เป็นคลาสหลัก ทำหน้าที่ในการจัดการวัตถุในเกม

-ด้านการตรวจจับการชน (Collision Detection) มีฟังก์ชันในคลาส *camera* ทำหน้าที่ในการทดสอบการชนกับกล้อง

-ด้านปัญญาประดิษฐ์ของเกม (Artificial Intelligence) มีคลาส *gameObject* เป็นคลาสหลัก ทำหน้าที่ในการแทนวัตถุในเกมและมีฟังก์ชันในการทดสอบการมองเห็น การได้ยินของวัตถุกับกล้อง

-ด้านการวาดรูป (Render) มีคลาส *render* เป็นคลาสหลัก ทำหน้าที่ในการรับข้อมูลมาวาดรูปในเกมและกำหนดตำแหน่งของกล้อง

-ด้านการติดต่อสื่อสาร (Network Communication) มีคลาส *socketObject* เป็นคลาสหลัก ทำหน้าที่ในการติดต่อสื่อสารทางเน็ตเวิร์ค

2) ทดสอบการทำงาน

ทดสอบการรับอินพุต

ทดสอบการเดินว่ากล้องจะเปลี่ยนตำแหน่งตามที่ต้องการหรือไม่

ทดลองเขียนรูปขึ้นมาแล้วทดสอบว่าฟังก์ชันทดสอบการชนจะรู้ได้ว่ามีการชนหรือไม่

ทดลองเขียนรูปขึ้นมาแล้วทดสอบว่าฟังก์ชันการเห็น การได้ยิน สามารถรู้ได้หรือไม่ว่าเห็น ได้ยิน

ทดสอบการวาดรูปโดยการเรียกใช้ส่วนที่ติดต่อไดเรกเอกซ์ ว่าวาดรูปตามที่ต้องการหรือไม่

ทดสอบการติดต่อสื่อสารในการส่งข้อความระหว่างโปรแกรมบนเครื่องคอมพิวเตอร์เครื่องเดียว

ทดสอบการจัดการกับวัตถุในเกม โดยสร้างวัตถุขึ้นมาว่าจะสามารถตีเป็นตารางได้ตามที่ต้องการหรือไม่

3) ปรับปรุงการทำงานฟังก์ชันต่างๆ ตามข้อผิดพลาดที่เกิดขึ้น

1.8 ความต้องการของเฟรมเวิร์คเกม

ระบบปฏิบัติการ ไมโครซอฟท์ วินโดวส์ (Microsoft Windows)

ฟังก์ชันกราฟิก ไดเรกเอกซ์ (Direct X) เวอร์ชัน 8

คอมไพเลอร์ ไมโครซอฟท์ วิชวลซีพลัสพลัส (Microsoft Visual c++)

1.9 ประเภทของเกมที่เป็นเป้าหมายของเฟรมเวิร์คเกม

เกม 3 มิติที่เหมือนกับผู้เล่นเดินเข้าไปในฉากแต่ละฉากเองหรือเกมประเภท First Person Shooter

1.10 ตัวอย่างงานวิจัยอื่นๆ

SS GameEngine เป็นเกมเอนจิ้นที่ใช้สร้างเกม 3 มิติโดยในเฟรมเวิร์คใช้ ไมโครซอฟท์เอกซ์เอ็นเอ (Microsoft XNA) ข้อมูลจากเอกสารแนะนำซอฟต์แวร์ SS GameEngine ของบริษัท จิมมี่ซอฟต์แวร์

1.10.1 ความต้องการของเฟรมเวิร์ค (Framework Requirements)

เฟรมเวิร์คจะใช้ภายใต้สภาพแวดล้อมต่อไปนี้

ระบบปฏิบัติการไมโครซอฟท์ วินโดวส์ (Microsoft Windows)

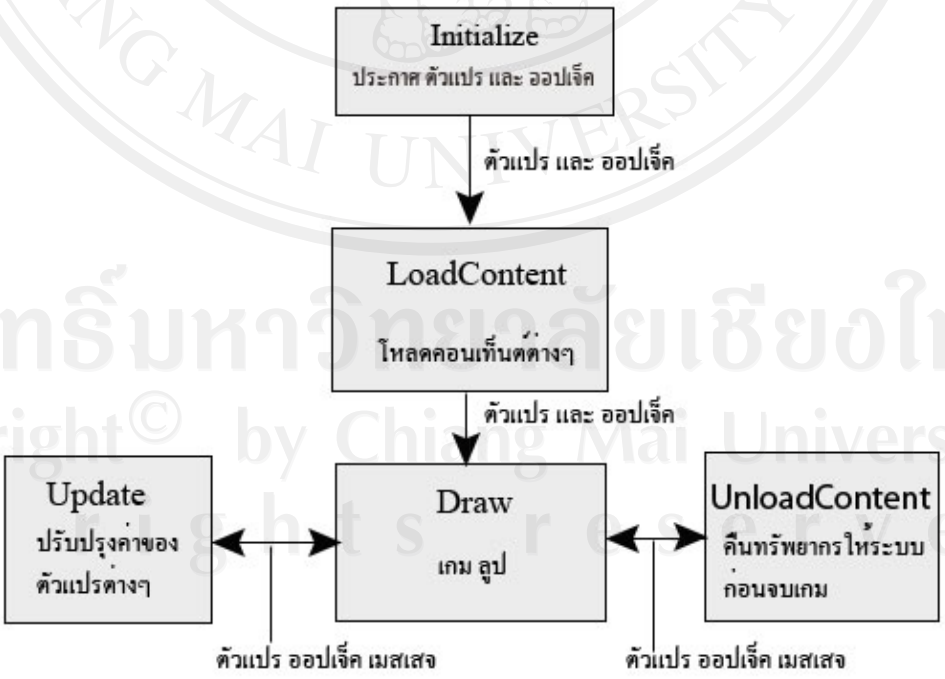
ไมโครซอฟท์ ดอตเน็ตเฟรมเวิร์ค เวอร์ชัน 2 (Microsoft .NET Framework 2.0)

ไดเรกเอกซ์ เวอร์ชัน 9 (Direct X 9.0)

ไมโครซอฟท์เอกซ์เอ็นเอเฟรมเวิร์ค เวอร์ชัน 2 (MicrosoftXNA Framework Redistributable 2.0)

ไมโครซอฟท์ วิชวลสตูดิโอ 2005 (Microsoft Visual Studio 2005)

1.10.2 การสร้างเกมด้วย SS GameEngine



รูปที่ 1.26 การทำงานของ SS GameEngine

ขั้นตอนการสร้างเกมที่ดำเนินการด้วยเกมเอนจินนี้แสดงดังรูปที่ 1.26 ซึ่งแต่ละขั้นตอนดำเนินการดังนี้

1) Initialize ใช้ในการกำหนดค่าเริ่มต้นให้กับ ออปเจ็กต์ หรือตัวแปรต่างๆ ในเกม

2) LoadContent ใช้โหลดคอนเท็นต์ต่างๆ

3) UnloadContent ใช้ลบคอนเท็นต์ต่างๆที่เคยโหลดไว้ใน LoadContent และคืนหน่วยความจำให้แก่ระบบ ถูกเรียกตอนออกจากเกม

4) Update ใช้ในการปรับปรุงค่าของตัวแปรต่างๆ ถูกเรียกใช้ทุกครั้งตลอดเวลาที่เกมกำลังรันอยู่

5) Draw ใช้ในการวาดภาพ (Render) ถูกเรียกใช้ทุกครั้งตลอดเวลาที่เกมกำลังรันอยู่