การพัฒนาเกมสามมิติบนเครือข่าย

3D GAME DEVELOPMENT (ONLINE)

นายก้องเกียรติ แสงเดช
นายกิติกรณ์ วิริยะประเสริฐกุล
นายทรงยศ จิตรยั่งยืน

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต
ภาควิชาวิศวกรรมคอมพิวเตอร์
คณะวิศวกรรมศาสตร์
สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีการศึกษา 2547

การพัฒนาเกมสามมิติบนเครือข่าย

3D GAME DEVELOPMENT (ONLINE)

นายก้องเกียรติ แสงเดช

นายกิติกรณ์ วิริยะประเสริฐกุล

นายทรงยศ จิตรยั่งยืน

อาจารย์ที่ปรึกษา

ดร.วรวัฒน์ ลิ้มโภคา

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรปริญญาวิศวกรรมศาสตร์บัณฑิต

ภาควิชาวิศวกรรมคอมพิวเตอร์

คณะวิศวกรรมศาสตร์

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ปีการศึกษา 2547

ปริญญานิพนธ์ปีการศึกษา 2547

ภาควิชา วิศวกรรมคอมพิวเตอร์

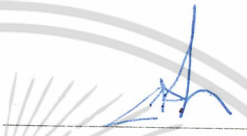
คณะวิศวกรรมศาสตร์ สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

เรื่อง การพัฒนาเกม 3 มิติบนเครือข่าย

3D GAME DEVELOPMENT (ONLINE)

ผู้จัดทำ

1. นาย ก้องเกียรติ	แสงเดช	45015353
2. นาย กิติกรณ์	วิริยะประเสริฐกุล	45015354
3. นาย ทรงยศ	จิตรยั่งยืน	45015368



อาจารย์ที่ปรึกษา

(ดร.วรวัฒน์ ลิ้มโภคา)

การพัฒนาเกมสามมิติบนเครือข่าย

1. นาย ก้องเกียรติ	แสงเดช	45015353
2. นาย กิติกรณ์	วิริยะประเสริฐกุล	45015354
3. นาย ทรงยศ	จิตรยั่งยืน	45015368
ดร.วรวัฒน์	ลิ้มโภคา	อาจารย์ที่ปรึกษา

บทคัดย่อ

ปัญหาใหญ่ปัญหาหนึ่งของเกมออนไลน์ (Online Game) คือการเกิดความหน่วงเวลาระหว่างการเล่นเกม (Lag) เป็นปัญหาที่มาจากหลายสาเหตุด้วยกัน สาเหตุหลักๆที่พบคือมาจากมีผู้เล่นจำนวนมาก ที่เชื่อมต่อเข้ามาเล่นในเวลาเดียวกัน ทำให้เครื่องเซิร์ฟเวอร์ (Server) ที่มีประสิทธิภาพสูงอยู่แล้วก็อาจจะรองรับได้ลำบาก

วิทยานิพนธ์ฉบับนี้เสนอแนวทางในการลดภาระในการประมวลผลจากการใช้เซิร์ฟเวอร์หลักตัวเดียวด้วยการใช้คลัสเตอร์ (Cluster) มาช่วยในการประมวลผล และจากการตั้งข้อสังเกตในการเล่นเกมออนไลน์ จะเห็นได้ว่าจะมีการเล่นอยู่ที่แผนที่เดียวกันเป็นจำนวนมาก ส่วนแผนที่ที่ไม่ค่อยมีคนนิยมไปก็จะเล่นได้ราบรื่นเป็นปกติ คณะผู้จัดทำจึงได้มุ่งประเด็น ไปยังการแบ่งการประมวลผล ที่ยึดการแบ่งโดยการแยกแยะตามแผนที่

3D GAME DEVELOPMENT (ONLINE)

Kongkiat Sangdej
Kitikorn Viriyaprasertkul
Songyote Jityungyuen
Dr. Voravat Limpoca Advisor

ABSTRACT

The majority problem of online game is lag during play because of many reasons, but the main one is a lot of player connected to play in the same time. That can crash down the server, although the server has a high performance.

This thesis suggestion the other way to reduce the load of processor when use the only one server to handle the whole game. Using the cluster can separate the load of online game in to many computers to complete the tasks. The most noticeable problem is lag in certain areas, so we lead the distributed processing algorithm to be distinguished by the maps of game.

กิตติกรรมประกาศ

ปริญญานิพนธ์ฉบับนี้คงไม่อาจเสร็จได้ด้วยดี หากไม่ได้รับความช่วยเหลือและร่วมมือจากหลายๆฝ่ายด้วยกัน บุคคลแรกที่ต้องกล่าวถึงเพราะเป็นส่วนสำคัญที่ทำให้วิทยานิพนธ์นี้เสร็จลงได้ก็คืออาจารย์ วรวัฒน์ ลิ้มโภคา อาจารย์ที่ปรึกษาปริญญานิพนธ์ที่ให้ความเอาใจใส่แนะนำและช่วยเหลือเสมอมาซึ่งต้องขอขอบพระคุณเป็นอย่างมาก

ขอขอบคุณภาควิชาวิศวกรรมคอมพิวเตอร์ที่เอื้อเฟื้อสถานที่ และเครือข่ายความเร็วสูงซึ่งทำให้การวิจัยอยู่ในสภาพแวดล้อมที่สะดวกสบาย ขอขอบคุณรุ่นพี่ที่ได้สำเนาเอกสารและข้อมูลที่เกี่ยวข้องไว้ให้ ทำให้การวิจัยง่ายขึ้นในบางส่วน ขอบคุณเพื่อนๆน้องๆ ในห้องวิจัย OLALA ที่ช่วยแนะนำเอกสารประกอบและเอื้อเฟื้อเครื่องคอมพิวเตอร์เพื่อใช้ในการทดลอง ขอขอบคุณสมาชิกในกลุ่มที่พยายามช่วยกันทำปริญญานิพนธ์ฉบับนี้ให้สำเร็จลุล่วงด้วยความขยันอดทน

นาย ก้องเกียรติ แสงเดช
นาย กิติกรณ์ วิริยะประเสริฐกุล
นาย ทรงยศ จิตรยั่งยืน

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญ(ต่อ)

สารบัญรูปภาพ

สารบัญรูปภาพ(ต่อ)

สารบัญตาราง

บทที่ 1
บทนำ

1.1 ความสำคัญและที่มา

เกมในปัจจุบันมีแนวโน้มรูปแบบของเกมที่มีลักษณะการเล่นที่สามารถรองรับผู้เล่นได้หลายๆคน หรือเรียกกันส่วนใหญ่ว่าเกมออนไลน์เพิ่มมากขึ้นผ่านทางระบบเครือข่ายขนาดใหญ่ โดยผู้เล่นแต่ละคนจะต้องทำการเชื่อมต่อเข้าไปยังเซิร์ฟเวอร์เสียก่อนจึงจะสามารถเล่นได้ ซึ่งโดยส่วนใหญ่แล้วในเซิร์ฟเวอร์หนึ่งๆจะรองรับผู้เล่นได้ประมาณ 2000- 3000 คน ดังนั้นเกมในปัจจุบันส่วนใหญ่จึงแบ่งแยกเซิร์ฟเวอร์มีจำนวนหลายตัวให้ผู้เล่นเลือกเซิร์ฟเวอร์ที่จะเล่นได้เพื่อรองรับผู้เล่นจำนวนที่มีมากขึ้น การศึกษาแนวทางในการพัฒนาของเกมออนไลน์นี้เป็นแนวทางหนึ่งที่น่าสนใจสำหรับผู้ที่สนใจจะพัฒนาเกมเพื่อความบันเทิงเป็นอย่างมาก และเนื่องจากระบบจะต้องสามารถรองรับผู้เล่นได้จำนวนมากทางกลุ่มผู้จัดทำโครงงานจึงเล็งเห็นจุดที่น่าสนใจการพัฒนาเกมในลักษณะนี้ว่าควรจะมีทิศทางและจุดเด่นอย่างไรโดยได้เน้นในส่วนของความสามารถทางด้านการกระจายการประมวลผลที่ฝั่งเซิร์ฟเวอร์เป็นหลัก

1.2 วัตถุประสงค์ของงานวิจัย

1.2.1 เพื่อศึกษาการทำงานของเกมสามมิติออนไลน์

1.2.2 เพื่อศึกษาแนวทางในการกระจายการประมวลผลต่างๆ

1.2.3 เพื่อทำการออกแบบและสร้างเกมที่สามารถรองรับผู้เล่นได้จำนวนมากโดยอาศัยหลักการกระจายการประมวลผลต่างๆที่ได้ทำการศึกษามา

1.3 ขอบเขตงานวิจัย

งานวิจัยนี้เป็นการพัฒนาเกม 3 มิติด้วยไดเร็กเอ็กซ์ (DirectX) ซึ่งเกมที่จะทำการพัฒนาขึ้นนั้นนอกจากจะมีลักษณะมุมมองภายในเกมที่มีลักษณะเป็นสามมิติแล้ว เกมที่พัฒนาขึ้นยังสามารถเล่นได้หลายคนผ่านระบบเครือข่ายได้ โดยผู้เล่นแต่ละคนจะต้องทำการล็อกอินที่เครื่องฝั่งไคลเอนต์เชื่อมต่อไปยังเซิร์ฟเวอร์กลางเพื่อทำการเล่นเกมร่วมกัน เกมที่ฝั่งของไคลเอนต์จะไม่ต้องทำการเก็บเสถานะใดๆที่เครื่องของตนเองแต่สถานะต่างๆของผู้เล่นแต่ละคนจะถูกเก็บไว้ที่ภายในเครื่องเซิร์ฟเวอร์ และเมื่อมีผู้เล่นจำนวนมากได้ทำการเชื่อมต่อไปที่เครื่องเซิร์ฟเวอร์แล้วเกิดภาระการทำงานที่หนัก เซิร์ฟเวอร์จะสามารถแบ่งกระจายการประมวลผลไปให้กับคลัสเตอร์เพื่อที่จะแบ่งเบาภาระของเซิร์ฟเวอร์เพียงตัวเดียวได้

โครงงานวิจัยนี้ได้เน้นในเรื่องของการกระจายการประมวลผลเป็นหลักและยังได้ทำการทดสอบการทำงานของระบบการกระจายการประมวลผลโดยรวมแล้วนำมาวิเคราะห์ถึงจุดเด่นจุดด้อยต่างๆของการทำงานเพื่อที่จะศึกษาความเป็นไปได้ในการนำต้นแบบไปพัฒนาใช้งานภายในอนาคต

1.4 วิธีการดำเนินงาน

งานวิจัยโครงงานนี้เริ่มต้นจากการศึกษาทฤษฎีพื้นฐานต่างๆที่จำเป็นเช่น ไดเร็กเอ็กซ์ กราฟิกสามมิติ การพัฒนาเกมสามมิติ และ ระบบผู้เล่นหลายคน ซึ่งมีรายละเอียดในบทที่ 2, 3, 4 และ 5 จากนั้นนำเอาความรู้ที่ได้มาทำการออกแบบเกมให้มีลักษณะเชื่อมต่อเล่นได้หลายคนติดต่อสื่อสารแบบไคลเอนต์เซิร์ฟเวอร์ รวมไปการถึงออกแบบวิธีในการกระจายการประมวลผลในแบบต่างๆซึ่งมีรายละเอียดในที่ 6 จากนั้นก็จะดำเนินการพัฒนาชิ้นงานตามโครงสร้างการออกแบบ เมื่อพัฒนาชิ้นงานเสร็จสิ้นก็จะนำชิ้นงานนั้นมาทำการทดสอบระบบซึ่งมีรายละเอียดในบทที่ 7 ในบทนี้ยังมีรายละเอียดของการใช้งานโปรแกรมอีกด้วย และบทที่ 8 ซึ่งเป็นบทสุดท้ายจะทำการสรุปผลงานวิจัย รวมทั้งเสนอแนะแนวทางในการวิจัยต่อและการนำงานวิจัยนี้ไปประยุกต์ใช้

บทที่ 2
DirectX

2.1 ความรู้เบื้องต้นเกี่ยวกับ DirectX

DirectX เป็น API ของไมโครซอฟต์ที่พัฒนาขึ้นเพื่อใช้จัดเตรียมอินเตอร์เฟซสำหรับควบคุมฮาร์ดแวร์มัลติมีเดียบนระบบ Microsoft Windows ได้อย่างมีประสิทธิภาพ และเป็นเครื่องมือให้โปรแกรมเมอร์ทำงานทำงานกับคำสั่งและโครงสร้างข้อมูลในระดับใกล้ชิดฮาร์ดแวร์โดยไม่ต้องสร้างโค้ดติดต่อในระดับล่างซึ่งวิธีติดต่อจะแตกต่างกันไปตามประเภทของอุปกรณ์ การเขียนโค้ดที่เป็นอิสระจากอุปกรณ์ในลักษณะนี้ ช่วยให้โปรแกรมเมอร์สามารถสร้างซอฟต์แวร์เพื่อทำงานดังกล่าวได้เป็นอย่างดี แม้ผู้ใช้จะปรับเปลี่ยนอุปกรณ์ตัวใหม่และเพิ่มการ์ดเร่งความเร็วแบบสามมิติ, เสียง, อุปกรณ์อินพุต และอื่นๆก็ตาม

โค้ดที่เขียนด้วย DirectX สามารถทำงานได้อย่างรวดเร็วด้วย API ของDirectX ที่สร้างขึ้นมาบนรากฐานของ HAL (Hardware Abstraction Layer) สามารถซ่อนลักษณะเฉพาะของดีไวซ์ (Device) ที่เกี่ยวพันอยู่กับฮาร์ดแวร์ และเนื่องจาก DirectX ได้รับการออกแบบให้มีความสามารถในการเสริมขยายได้ในอนาคต ดังนั้นมันจึงรองรับความสามารถของฮาร์ดแวร์เร่งความเร็วต่างๆมากมายที่ปัจจุบันยังไม่มี ด้วยความสามารถจำลองการทำงานผ่าน HEL (Hardware Emulation Layer)

2.2 API ของDirectX

DirectX ประกอบไปด้วย API มากมายที่ได้รับการออกแบบเพื่อใช้พัฒนาเกมและการจำลองแบบแบบสามมิติ DirectX มีไลบรารีที่เก็บฟังก์ชันที่ใช้ในการเรนเดอร์แบบสองมิติและสามมิติ, สร้างเสียงแบบปกติและแบบสามมิติ, ดนตรี, ติดต่อแป้นพิมพ์, จอยสติ๊ก และอุปกรณ์อินพุตชนิดต่างๆ รวมทั้งฮาร์ดแวร์ที่มีความสามารถสร้างปฏิกิริยาสะท้อนกลับ (เช่น จอยสติ๊กแบบสั่น) และการเล่นเกมผ่านเครือข่าย อินเตอร์เฟซที่ DirectX เตรียมไว้ให้เต็มไปได้ความสามารถ โดยเราสามารถใช้ไลบรารีของคำสั่งต่างๆ ที่รวบรวมเข้ามาเพื่อสร้างเกมและทำซิมมูเลชั่นที่งดงามและยิ่งใหญ่อลังการ

ชุด API ที่มีอยู่ใน DirectX คือ DirectDraw, Direct3D, DirectMusic, DirectSound, DirectPlay, DirrectInput และ DirectSetup

2.2.1 DirectDraw

DirectDraw เป็นชุด API สำหรับจัดการอุปกรณ์แสดงผล ควบคุมข้อมูลบิตแมป หน่วยความจำที่ออกนอกพื้นที่สกรีนและสร้างการติดต่อที่รวดเร็วให้กับฟีเจอร์ของฮาร์ดแวร์สามารถใช้ส่วน DirectDraw ทำการสร้างเกมสองมิติได้

2.2.2 Direct3D

ไดเรกสามมิติในยุคแรกมี API อยู่สองโหมดคือ โหมด Immediate (IM) และโหมด Retained (RM) ในโหมด IM เป็นโหมดที่ใช้งานยากแต่มีความยืดหยุ่นสูง เป็น API ในระดับล่างสำหรับใช้เขียนเกมที่ทำงานได้เร็วและมีประสิทธิภาพเท่าที่จะเป็นไปได้บนระบบ ในโหมด RM เป็นโหมดที่สร้างขึ้นมาเป็นเลเยอร์ที่อยู่บนสุดของโหมด IM โดยโหมดนี้จะจัดเตรียมบริการต่างๆเช่น การจัดการ Texture, การโหลด Object File, การจัดลำดับเฟรม และการจัดทำออบเจ็กต์เคลื่อนไหว การศึกษาและใช้งานโหมด RM นั้นง่ายกว่าเมื่อเทียบกับโหมด IM แต่โปรแกรมเมอร์คนใดที่ต้องการประสิทธิภาพและความยืดหยุ่นก็มักจะเลือกใช้การทำงานในโหมด IM ดังนั้นการพัฒนาการทำงานในโหมด RM จึงได้หยุดลงใน DirectX เวอร์ชัน 6 และมุ่งพัฒนาการทำงานในโหมด IM ให้มีความสามารถและใช้งานง่ายในเวอร์ชันต่อมา ด้วยเหตุนี้การทำงานในโหมด RM จึงไม่สนับสนุนเทคโนโลยีใหม่ เช่น Multitexturing, Bump Mapping, Hardware Transformation และ Lighting

2.2.3 DirectMusic

เป็นชุด API ที่ทำงานร่วมกับข้อมูลประเภท Message-based Musical Data ซึ่งเป็นข้อมูลที่แปลงมาจาก Wave Sample ด้วยซินธีไซเซอร์(Synthesizer)ทั้งแบบฮาร์ดแวร์หรือซอฟต์แวร์การใช้เสียงดนตรีที่ได้จากซินธีไซเซอร์จะเป็นไปตามมาตรฐาน DLS (Down loadable Sound) นอกจากนี้ DirectMusic ยังมีกลไกที่ใช้สร้างเพลงได้ตามที่เรากำหนด

2.2.4 DirectSound

API ชุดนี้เป็นเครื่องมือที่ใช้จัดการเสียงแบบสเตริโอและแบบสามมิติอย่างมีประสิทธิภาพ ประกอบด้วยความสามารถในการจัดการหน่วยความจำและการผสมเสียงของฮาร์ดแวร์ DirectSound ได้รับการออกแบบมาเพื่อดึงความสามารถของฮาร์ดแวร์บนระบบ การรวมเสียงแบบสามมิติเข้าไปในเกมหรือการซิมมูเลชัน ทำให้แอพพลิเคชันสามารถให้เสียงได้สมจริงสมจัง การได้ยินเสียงจากทางซ้าย ทางขวา หรือทางด้านบนของผู้เล่นหรือผู้ชม โดยเฉพาะอย่างยิ่งเสียงที่มาจากรอบๆตัวจะช่วยสร้างสภาพแวดล้อมที่น่าตื่นตาตื่นใจ

2.2.5 DirectPlay

DirectPlay ทำให้เกมสามารถเล่นได้หลายคนสนับสนุนการเล่นเกมผ่านเครือข่ายและทำให้สามารถเชื่อมต่อแบบ Transport-independent รวมทั้งให้บริการ Message Service ด้วย

2.2.6 DirectInput

เป็น API ของ DirectX ที่สนับสนุนอินพุตความเร็วต่ำ (Low-latency Input) ซึ่งเป็นอุปกรณ์อินพุตส่วนใหญ่ที่มีในปัจจุบัน สนับสนุนอุปกรณ์อินพุตทุกชนิดที่มีความสามารถแสดงปฏิกิริยาสะท้อนกลับ เช่น จอยสติ๊ก หรือพวงมาลัยแบบสั่นได้อุปกรณ์อินพุตแบบนี้สามารถจำลองสถานการณ์บางส่วนให้

สมจริง เช่น จำลองสถานการณ์เมื่อขับรถชนบนถนน ที่อุปกรณ์อินพุต(พวงมาลัย)จะส่งแรงสั่นสะเทือนให้ผู้เล่นรู้สึกตามความรุนแรงและความเสียหายของรถ แรงถีบของปืนเมื่อผู้เล่นยิง แรงปะทะลมเมื่อบิน และคลื่นที่กระทบเรือ เป็นต้น

2.2.7 DirectSetup

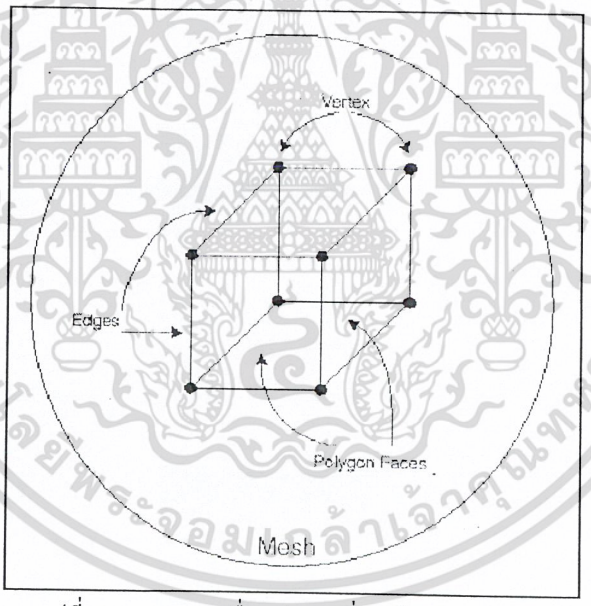
API ชุดนี้ของ DirectX มีหน้าที่ติดตั้งคอมโพเนนต์ DirectX Runtime ให้กับระบบโดยอัตโนมัติถ้าหากระบบดังกล่าวยังไม่ดีรับการติดตั้ง แม้ในปัจจุบันมักจะมีการติดตั้งคอมโพเนนต์มากับระบบปฏิบัติการแล้วก็ตาม

บทที่ 3
กราฟิกสามมิติ

ในการพัฒนาเกมสามมิติที่มีกราฟิกต่างๆ มีความจำเป็นอย่างยิ่งที่จะต้องมีความเข้าใจทฤษฎีพื้นฐานที่จำเป็นสำหรับนำไปประยุกต์ใช้ โดยเฉพาะทฤษฎีในด้านกราฟิกซึ่งเป็นพื้นฐานสำคัญสำหรับการพัฒนาเกมสามมิติ

3.1 กราฟิกสามมิติทั่วไป

โพลีกอน(Polygon) คือ รูปทรงสามเหลี่ยมทั่วไปที่ประกอบไปด้วยจุดสามจุด เวอร์เท็กซ์(Vertex) คือ ส่วนย่อยที่เล็กที่สุดในระบบสามมิติ ซึ่งก็คือจุดที่ตั้งอยู่ภายในสเปซสองมิติหรือสามมิติ เราสามารถสร้างเส้นได้โดยการรวมสองเวอร์เท็กซ์เข้าด้วยกัน เส้นสามเส้นจะรวมกันเป็นโพลีกอน ความสัมพันธ์ของจุดเวอร์เท็กซ์ที่สามารถวาดเส้นเชื่อมต่อกันเป็นรูปร่าง คือ พื้นผิวโพลีกอน (Polygon face) และรูปที่เราได้มาทั้งหมดถูกเรียกว่า mesh หรือว่าโมเดล



รูปที่ 3.1 แสดงการเชื่อมต่อจุดเพื่อสร้างวัตถุสามมิติ

เพื่อเป็นการสร้างความดูสมจริงให้กับวัตถุเราทำการกำหนดสีให้กับโพลีกอนเรียกว่า material และเรายังสามารถนำภาพบิตแมป (Bitmap) ไปแปะลงบนพื้นผิวของโพลีกอนเมื่อนำโพลีกอนไปทำการเรนเดอร์ซึ่งเรียกว่าเท็กซ์เจอร์(Texture)

3.2 ระบบพิกัด (Coordinate System)

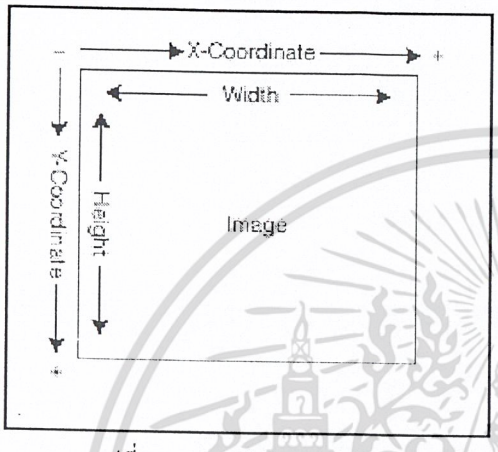
ระบบพิกัดที่ใช้ในคอมพิวเตอร์กราฟิกส่วนใหญ่แบ่งออกเป็นสองพิกัดด้วยกันหลักๆคือ

3.2.1 ระบบพิกัดสองมิติ

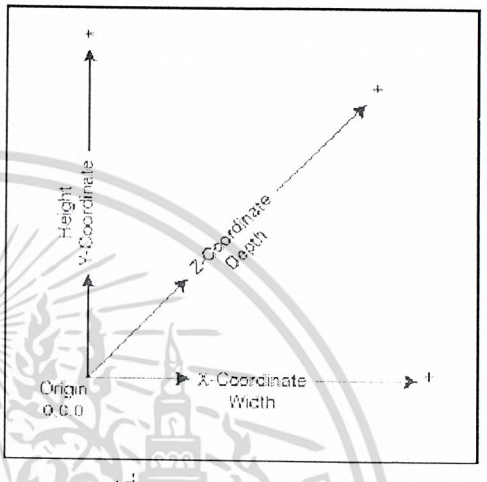
เป็นพิกัดที่ใช้ในการแสดงภาพที่มีเฉพาะค่าของความกว้างและความสูง ซึ่งถูกกำหนดโดยค่า X และค่า Y

3.2.2 ระบบพิกัดสามมิติ

เป็นระบบพิกัดที่มีแกน Z เพิ่มเข้ามาเพื่อแสดงถึงความลึกของภาพ



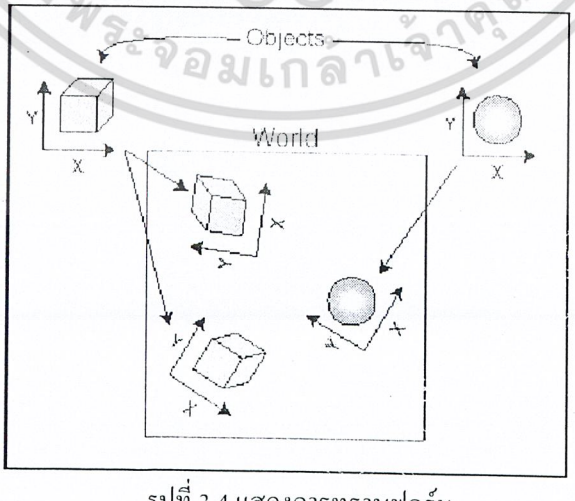
รูปที่ 3.2 แสดงพิกัดสองมิติ



รูปที่ 3.3 แสดงพิกัดสามมิติ

3.3 การทรานฟอร์ม (Transformation)

หลังจากเราได้ทำการสร้างโมเดลเสร็จแล้ว เราก็พร้อมที่จะโมเดลวางลงบน World ตามตำแหน่งที่ต้องการ จากรูปที่ 3.4 แสดงการวางโมเดลในรูปแบบต่างๆลงบน World แบบสามมิติ ซึ่งเราสามารถเคลื่อนย้าย(Translate) ปรับขนาด(Scale) หรือหมุน(Rotate)วัตถุได้โดยวัตถุที่แสดงออกมาจะมีทิศทางที่แตกต่างกันออกไป ซึ่งการกระทำต่างๆเหล่านี้เราจะเรียกว่าการทรานฟอร์มโดยแสดงตามรูป 3.4



รูปที่ 3.4 แสดงการทรานฟอร์ม

3.4 Lighting

โมเดลของแสงในไดเร็กสามมิติจะคล้ายคลึงกับแสงที่เกิดขึ้นในโลกความเป็นจริง โดยโมเดลของแสงในไดเร็กสามมิติจะอาศัยการคำนวณการตกกระทบของสีบนพื้นผิว ผลจากการคำนวณจะเป็นสีที่เกิดขึ้นบนพื้นผิวเมื่อนำมาเรนเดอร์บนสกรีน ในความเป็นจริงแล้วแสงจะสะท้อนจากพื้นผิวก่อนจึงจะเข้ากระทบตาของเรา สำหรับไดเร็กสามมิติจะใช้การให้แสงที่สมจริงด้วยเหตุผลด้านประสิทธิภาพด้วยการกำหนดองค์ประกอบให้กับแสงซึ่งแสงในไดเร็กสามมิติจะมี4ชนิดได้แก่แสงแวดล้อม(Ambient Light), แสงแบบจุด(Point Light),แสงสปอร์ตไลต์(Spotlight)และแสงแบบมีทิศทาง(Directional Light) แสงแวดล้อม(Ambient Light)คือแสงที่ไม่มีทิศทางตายตัวและไม่มีแหล่งกำเนิดแสงเพราะเป็นแสงที่กระจัดกระจายไปทั่ว แสงแวดล้อมนี้สร้างความสว่างที่มีความเข้มต่ำทุกๆแห่งในซีนและประกอบไปด้วยสีและความเข้มของแสงเท่านั้น นอกจากนี้แสงแวดล้อมยังเป็นแสงที่อิสระต่ออ็อบเจ็กต์ให้กำเนิดแสง แสงอีก3ชนิดที่เหลือจะแสดงตามรูป 3.5

3.4.1 แสงแบบจุด (Point Light)

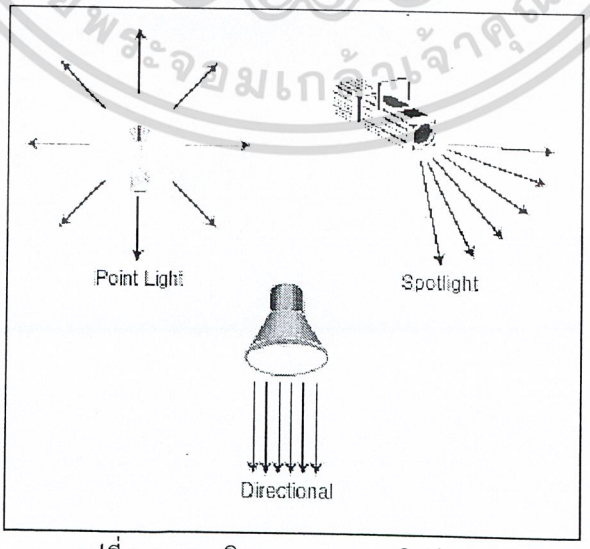
แสงแบบจุดมีแหล่งกำเนิดแสงเป็นจุดมีตำแหน่งอยู่ในที่ใดที่หนึ่งในระบบพิกัดและจะกระจายแสงออกทุกทิศทุกทางอย่างสม่ำเสมอจากจุดกำเนิด

3.4.2 แสงสปอร์ตไลต์ (Spotlight)

แสงที่เกิดจากสปอร์ตไลต์บางครั้งอาจดูเหมือนแสงแบบจุดแต่ต่างกันตรงที่แสงแบบสปอร์ตไลต์จะกระจากแสงเป็นรูปกรวยแหลม

3.4.3 แสงแบบทิศทางตรง (Directional Light)

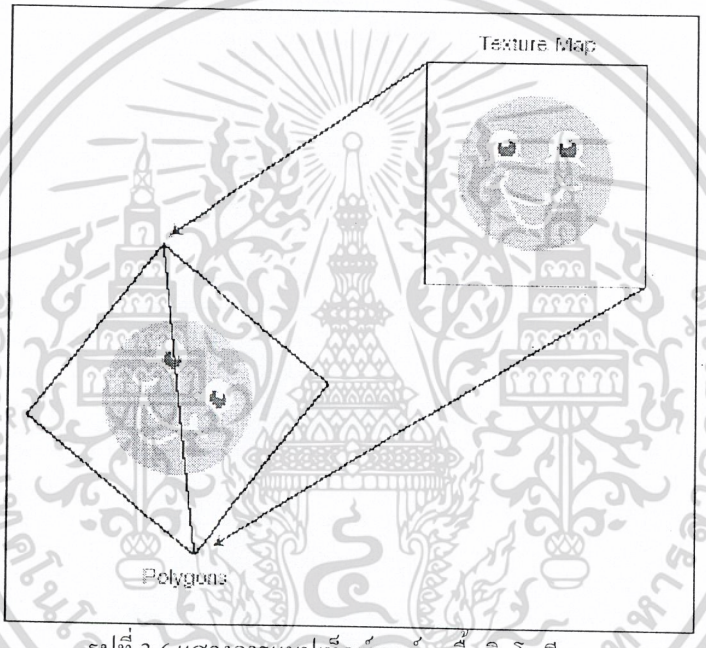
แสงแบบนี้บ่งบอกถึงแหล่งกำเนิดแสงที่มีระยะทางไม่สิ้นสุด แสงชนิดนี้ให้แสงแบบขนานดังนั้นแสงชนิดนี้จึงเคลื่อนไปในทิศทางเดียวกันภายในซีน ความเจือจางของแสงและระยะทางไม่มีผลกระทบใดๆกับแสงชนิดนี้เพราะไดเร็กสามมิติจะใช้เฉพาะสีและทิศทางที่กำหนดสำหรับการคำนวณสีของเวอร์เทกซ์เท่านั้น



รูปที่ 3.5 แสดงทิศทางของแสงชนิดต่างๆ

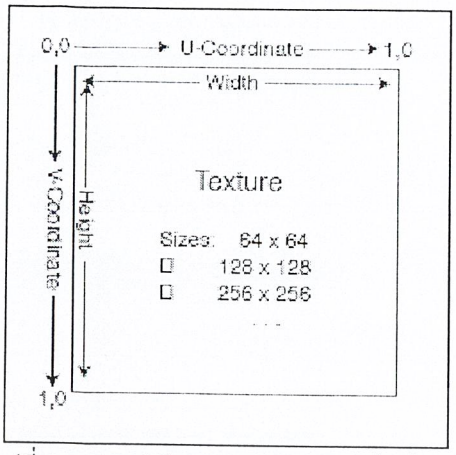
3.5 Mapping Texture

ไดเร็กสามมิติมีความสามารถในการใช้เท็กซ์เจอร์แมปลงบนพื้นผิวชิ้นงานกราฟิกเพื่อให้การเรนเดอร์ซีนทำได้อย่างสมจริง เท็กซ์เจอร์แมปปิ้ง(Texture-Mapping)เป็นเทคนิคที่ใช้การวาดรูปลงบนพื้นผิวของโพลีกอน(Polygon)ซึ่งแสดงตามรูปที่ 3.6 โดยทั่วไปเท็กซ์เจอร์เป็นภาพบิตแมปแต่สามารถใช้กับชิ้นงานกราฟิกสามมิติเพื่อให้ดูเสมือนวัตถุในโลกความเป็นจริงได้ เท็กซ์เจอร์มีลักษณะเป็นเม็ดสีรวมกันเป็นภาพสองมิติสามารถนำมาแมปลงบนพื้นผิวอ็อบเจ็กต์สามมิติได้ ซึ่งจริงๆแล้วเท็กซ์เจอร์เป็นเสมือนสีของอ็อบเจ็กต์ รูปร่าง และคุณลักษณะที่สัมผัสได้แต่ในไดเร็กสามมิติแล้วมันเป็นแม่แบบสีของอ็อบเจ็กต์ซึ่งไม่มีผลต่อการเปลี่ยนรูปทรงทางเรขาคณิตของอ็อบเจ็กต์ซึ่งแม่แบบสีนี้ทำให้อ็อบเจ็กต์เผยรูปร่างลายลักษณ์ของพื้นผิวออกมา



รูปที่ 3.6 แสดงการแมปเท็กซ์เจอร์ลงพื้นผิวโพลีกอน

ในการแมปเท็กซ์เจอร์ลงบนพื้นผิวโพลีกอนนั้นไดเร็กสามมิติจะทำการแมปพิกัดของเท็กซ์เซลไปสู่พิกัดของอ็อบเจ็กต์แล้วจึงแมปไปสู่พิกัดของสกรีนที่เป็นตำแหน่งพิกเซล โดยพิกัดของเท็กซ์เซลจะอยู่ในช่วง0.0 ถึง 1.0 ซึ่งตำแหน่งในเท็กซ์เจอร์จะกำหนดด้วยพิกัด u(คอลัมน์)และพิกัด v(แถว) เทียบได้กับพิกัด x(width)และพิกัด y(Height)ในสกรีน ซึ่งแสดงตามรูปที่ 3.7



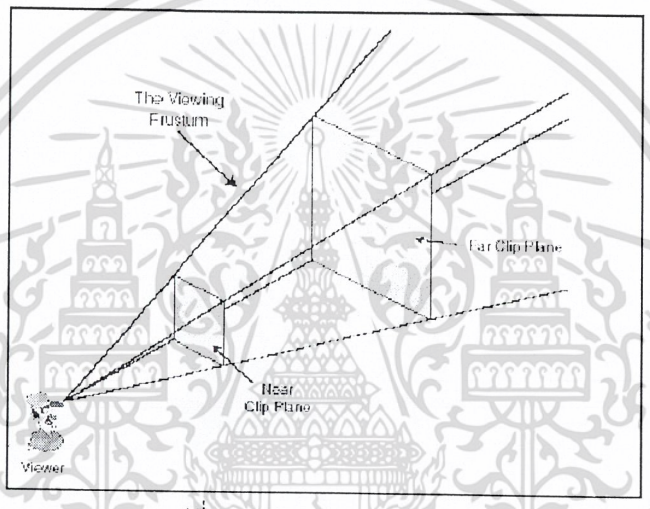
รูปที่ 3.7 แสดงพิกัด Uและพิกัด V ในเท็กซ์เจอร์

ส่วนเท็กซ์เซลคืออาร์เรย์สองมิติที่เก็บค่าสีต่างๆ เท็กซ์เซลจะมีตำแหน่งที่ไม่ซ้ำกันอยู่ในพื้นที่ของเท็กซ์เจอร์ โดยอ้างอิงตำแหน่งเป็นพิกัด Uและพิกัด V เรียกตำแหน่งพิกัดของเท็กซ์เซลเหล่านี้ว่าพิกัดในเท็กซ์เจอร์

บทที่ 4
การพัฒนาเกมสามมิติ

4.1 Viewing Frustum

Viewing frustum เป็นการรวมกันของเพลน 6 เพลนซึ่งขยายออกมาจากวิวพอร์ตใช้ในการกำหนดการมองเห็นของโพลีกอนจากผู้มอง viewing frustum จะเป็นประโยชน์อย่างมากในการออปติไมซ์การประมวลผลทางด้านกราฟิก frustum จะมีลักษณะคล้ายกับปิรามิดที่ขยายออกไปจากตัวเรา ปิรามิดจะแสดง field of view (FOV) ทุกๆสิ่งที่มองเห็นจะอยู่ภายในปิรามิดและทุกๆสิ่งที่อยู่ภายนอกปิรามิดจะมองไม่เห็น viewing frustum จะมีด้านอยู่ 6 ด้านด้วยกัน (หน้า, หลัง, ซ้าย, ขวา, บน, และล่าง)

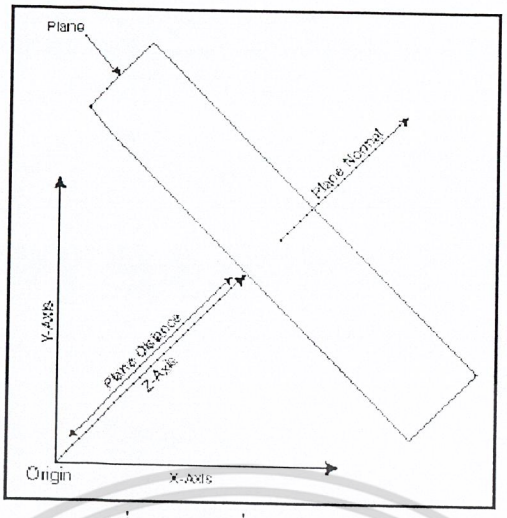


รูปที่ 4.1 แสดง viewing frustum

ในการคำนวณทางคณิตศาสตร์เราสามารถกำหนดเวอร์เท็กซ์ที่อยู่ภายใน frustum และเวอร์เท็กซ์ที่อยู่ภายนอก frustum โดยที่เวอร์เท็กซ์ที่อยู่ภายในจะถูกเรนเดอร์และเวอร์เท็กซ์ที่อยู่ภายนอกจะไม่ถูกเรนเดอร์

4.2 Plane and Clipping

ด้านทั้ง 6 ด้านของ viewing frustum เรียกว่า clipping plane เราสามารถกำหนดเพลนโดยการชี้มันไปยังทิศทางที่ระบุแล้วทำการย้ายเพลนไปยังตำแหน่งที่ห่างออกไปจากจุดกำเนิด ตามรูปที่ 4.2 แสดงภาพเพลนที่ถูกจัดไว้ในพื้นที่สามมิติ



รูปที่ 4.2 เพลนที่ถูกจัดวางไว้แล้ว

4.2.1 การตรวจสอบการมองเห็นด้วยเพลน

เมื่อเรามีเพลนแล้วจะใช้เพลนในการเช็คจุดต่างๆว่าอยู่ข้างหน้าหรือข้างหลังของเพลน โดยการคำนวณ dot product ซึ่ง dot product เป็นเวกเตอร์พิเศษโดยส่วนใหญ่แล้วจะใช้ในการคำนวณมุมระหว่างสองเวกเตอร์เมื่อเราทำการตรวจสอบจุดที่อยู่บนเพลน dot product จะบอกระยะทางจากจุดไปยังเพลนหากค่าเป็นบวกแสดงว่าจุดปรากฏอยู่ข้างหน้าเพลน หากค่าเป็นลบแสดงว่าจุดปรากฏอยู่ข้างหลังของเพลน

สามารถตรวจสอบออบเจ็กต์ว่าอยู่ภายใน frustum หรือไม่โดยการทดสอบตามลักษณะของออบเจ็กต์เช่น สี่เหลี่ยม ทรงลูกบาศก์ และทรงกลม สำหรับสี่เหลี่ยม หรือทรงลูกบาศก์เราสามารถตรวจสอบมุมทั้งหมดว่ามีอย่างน้อย 1 จุดหรือทั้งหมดอยู่ภายใน Frustum plane แต่ถ้าเป็นทรงกลมจะทำการตรวจสอบดูระยะทางจากแต่ละเพลนว่าเท่ากันหรือมากกว่าค่ารัศมีของทรงกลมหรือไม่ถ้าใช่ทรงกลมจึงจะสามารถมองเห็นได้ ในการที่จะทำการตรวจสอบจำนวนของจุดเราควรทำการตรวจสอบแต่ละจุดเป็นอิสระออกจากกัน ต้องให้แน่ใจว่ามีอย่างน้อย 1 จุดตกอยู่บนด้านหน้าของเพลนทั้งหมด

4.3 การพัฒนาเกมโดยใช้ 3D Engine ขั้นสูง

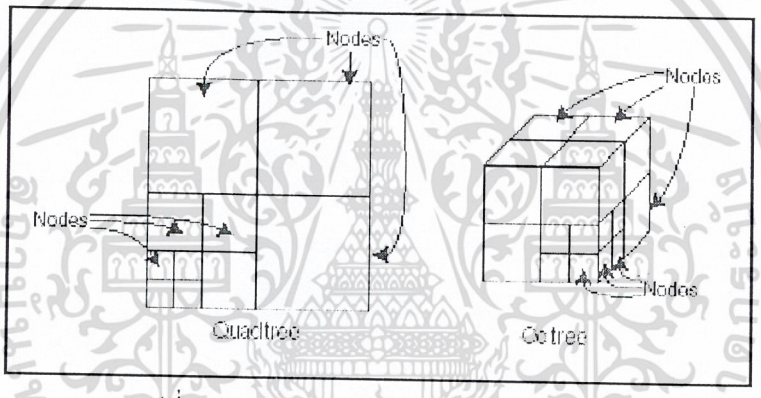
การเรนเดอร์ฉากของเกมที่มีลักษณะเพียง mesh เดียวนั้นจะกระทำได้ค่อนข้างง่ายใช้เวลาที่น้อย แต่ถ้าหากภายในฉากมีลายละเอียดเล็กๆน้อยๆที่มากขึ้น เช่น สิ่งก่อสร้าง, ต้นไม้, ถ้ำ และกราฟิกของเกมอื่นๆอีกที่จำเป็น เราสามารถใช้โปรแกรม 3Dโมเดลลิ่ง เช่น 3D Studio Max หรือ Milk Shape 3-D ทำการออกแบบฉาก ปัญหาของการทำเช่นนี้คือขนาดของโพลีกอนซึ่งจะมีขนาดใหญ่ การวาดโพลีกอนเหล่านี้ทั้งหมดในแต่ละเฟรมจะเป็นวิธีการที่ปราศจากประสิทธิภาพ ในการที่จะทำให้ได้ความเร็วที่เพิ่มขึ้นเราต้องทำการเรนเดอร์เฉพาะโพลีกอนที่อยู่ในวิว และเพื่อที่จะทำให้ได้การประมวลผลที่เร็วกว่าจะหลีกเลี่ยงการสแกนโพลีกอนแต่ละโพลีกอนทั้งหมดในฉากเพื่อที่จะกำหนดโพลีกอนที่สามารถมองเห็นได้

เราสามารถหลีกเลี่ยงการสแกนทั้งหมดภายในเฟรมได้โดยการแบ่งโมเดลสามมิติ (ฉากของเกม) ออกเป็นชิ้นเล็กๆ (หรือเรียกว่าโหนด) ซึ่งทำการบรรจุโพลิกอนเพียงเล็กน้อย จากนั้นทำการเรียงโหนดใน

โครงสร้างพิเศษ (หรือทรี) ซึ่งจะทำให้เราสามารถสแกนได้อย่างรวดเร็วเพื่อที่จะกำหนดหาโหนดที่สามารถมองเห็นได้ จากนั้นจึงทำการเรนเดอร์โหนดที่สามารถมองเห็นได้ เราสามารถค้นหาโหนดที่สามารถมองเห็นได้โดยการใช้ viewing frustum โดยแทนที่เราจะใช้การสแกนโพลีกอนนับล้านทั้งหมด เราสามารถแสกนเป็นชุดเล็กๆของโหนดเพื่อตัดสินใจว่าจะวาดหรือไม่ ซึ่งจะใช้เอนจิ้น NodeTree ในการเอาโหนดและตัดโหนดทิ้งเพื่อนำมาเรนเดอร์ได้อย่างรวดเร็ว

4.3.1 NodeTree Engine

NodeTree สามารถทำงานได้สองโหมดคือ โหมด Quadtree และโหมด Octree ในโหมด Quadtree นี้จะทำการแบ่งย่อย world (และโหนดย่อยๆ) ออกเป็น 4 โหนดในเวลาเดียวกันซึ่งโหมดนี้เหมาะสำหรับฉากที่มีแกน Y ไม่มากนัก (ความสูงของวิวพอร์ตไม่สูงมากนัก) ในโหมด Octreeจะแบ่งแยก world (และโหนดย่อยๆ) ออกเป็น 8 โหนดในเวลาเดียวกันซึ่งโหมดนี้เหมาะสำหรับ mesh สามมิติที่มีขนาดใหญ่ โดยวิวพอร์ตสามารถเคลื่อนย้ายไปที่ใดก็ได้ภายใน world

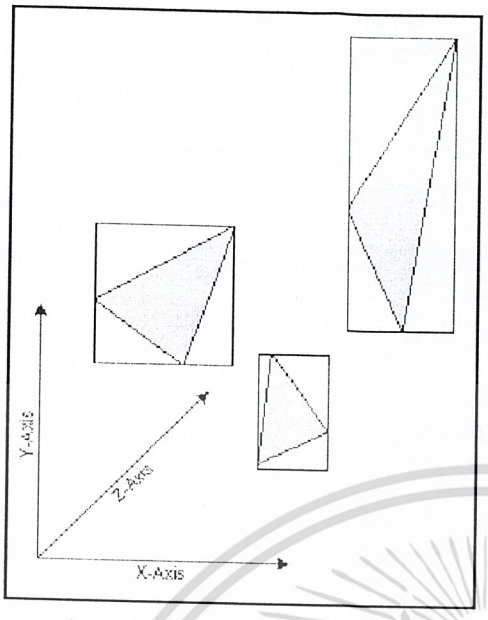


รูปที่ 4.3 แสดงโหมด Quadtree และโหมด Octree

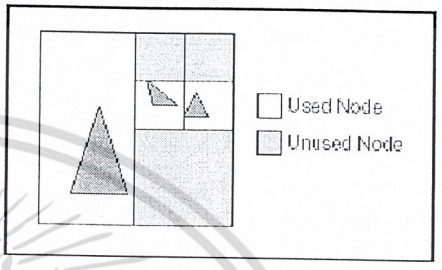
World สามารถแบ่งย่อยออกเป็นโหนดเล็กๆที่มีขนาดที่เท่ากัน Quadtree สามารถแบ่งย่อยโหนดในพื้นที่สองมิติ (ใช้แกน X และแกน Y) และ Octree สามารถแบ่งย่อยโหนดในพื้นที่สามมิติ (ใช้ทุกๆแกน)

โหนดจะแสดงกลุ่มของโพลีกอนที่อยู่ภายในในระหว่างที่เรากำลังแสดงพื้นที่สามมิติ แต่ละโหนดสามารถเชื่อมโยงไปยังโหนดอื่นๆในความสัมพันธ์แบบแม่กับลูกซึ่งหมายความว่าโหนดสมารถบรรจุโหนดอื่นๆภายในแต่ละส่วนย่อยซึ่งเป็นส่วนเล็กๆกว่าโหนดแม่ โดยปกติแล้วโลกของสามมิติจะถูกพิจารณาโดยรูทโหนด (root node)

แต่ละโพลีกอนใน mesh จะถูกปิดล้อมอยู่ภายในกล่องซึ่งเรียกว่า bounding box แสดงในรูปที่ 4.4 กล่องนี้จะแสดงขอบเขตของโพลีกอนในทุกๆทิศทาง bounding box ของโพลีกอนจะถูกหุ้มอยู่ภายในโหนดในพื้นที่สามมิติด้วยเช่นกันดังนั้นโพลีกอนจึงเชื่อมโยงอยู่ภายในโหนดโพลีกอนสามารถที่จะเชื่อมโยงกับหลายๆโหนดได้เพราะว่าขอบเขตของโพลีกอนสามารถผ่านไปยังหลายๆโหนดได้



รูปที่ 4.4 bounding box ที่ล้อมรอบโพลีกอน



รูปที่ 4.5 โพลีกอนที่ถูกรวมกลุ่มอยู่ในโหนด

เราสามารถทำการแบ่งย่อยโหนดย่อยๆลงไปได้มากกว่า 1 โหนดแล้วทำการสแกนโพลีกอนทั้งหมดอีกครั้งโดยทำไปเรื่อยๆได้จนกว่าโพลีกอนทั้งหมดในกลุ่มจะมีขนาดที่เล็กและจนกว่าแต่ละโหนดจะบรรจุโพลีกอนที่เล็กเพียงพอ ในรูปที่ 4.5 แสดงโพลีกอนจำนวนน้อยซึ่งถูกห้อมล้อมโดยสี่เหลี่ยม ซึ่งสี่เหลี่ยมแสดงถึงการแบ่งย่อยเป็นโหนดที่เล็กกว่า ในแต่ละโหนดจะพิจารณาถึงการถูกใช้ (โหนดที่บรรจุโพลีกอน) หรือไม่ถูกใช้ (โหนดไม่ได้บรรจุโพลีกอน) จากรูปที่ 4.5 เราแบ่งย่อยรูทโหนดออกเป็นโหนดที่เล็กกว่า (โหมด Quadtree) เราสามารถข้ามโหนดที่ว่างเปล่าไปได้เพื่อความเร็วที่เพิ่มขึ้น

4.3.2 การสแกน

เมื่อเราสร้างโครงสร้างทรีเสร็จก็พร้อมที่จะนำไปเรนเดอร์ โดยเริ่มต้นจากรูทโหนดเราจำเป็นต้องใช้ viewing frustum สำหรับทำการตรวจสอบบนโหนด โหนดจะถูกพิจารณาภายในวิวหากว่าหนึ่งใน 8 แปดจุด (คิดเป็นแต่ละมุมภายในทรงลูกบาศก์) อยู่ภายใน frustum หลังจากเราทำการตัดสินใจเลือกโหนดที่มองเห็นได้ก็จะทำวิธีการเดียวกันในโหนดย่อยๆ(ถ้าหากมี) หลังจากโพลีกอนที่อยู่ภายในโหนดถูกแสกนแล้วก็จะทำการทำเครื่องหมายไว้ ซึ่งการทำเช่นนี้จะทำให้เราสามารถกำจัดโพลีกอนจำนวนหลายล้านโพลีกอนออกจากกระบวนการเรนเดอร์และเป็นการประหยัดเวลา

เมื่อเราทำงานกับไดเร็กสามมิติและโครงสร้างทรีเราต้องหา mesh ที่ประกอบด้วยหลายๆเท็กซ์เจอร์ (Texture) ซึ่งการสลับเปลี่ยนเท็กซ์เจอร์เป็นการดำเนินงานที่ค่อนข้างเปลืองทรัพยากรดังนั้นเราจึงจำเป็นต้องระมัดระวังในจุดนี้ ซึ่งถ้าหากเราทำการวาดโพลีกอนที่มองเห็นได้โดยปราศจากการสลับสับเปลี่ยนเท็กซ์เจอร์ไปมา (เท็กซ์เจอร์เดิมถูกเรียกสลับไปมาหลายครั้ง) ซึ่งจะได้ประสิทธิภาพที่ดีกว่า

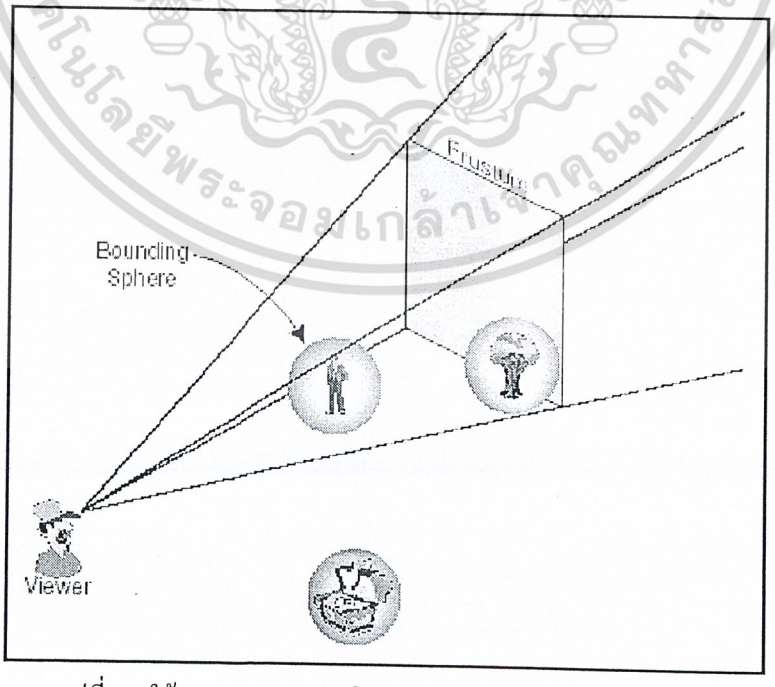
4.4 การทำงานกับกลุ่มของเท็กซ์เจอร์

เท็กซ์เจอร์กรุ๊ป(Texture Group) คือชุดของโพลีกอนซึ่งถูกจัดกลุ่มรวมเข้าไว้ด้วยกันบนพื้นฐานของเท็กซ์เจอร์ที่ถูกกำหนดเพราะว่า mesh นั้นสามารถประกอบไปด้วยหลายๆเท็กซ์เจอร์จะเป็นการดีกว่าถ้าหากว่าได้ทำการเรนเดอร์เฉพาะกลุ่มของโพลีกอนที่ขึ้นอยู่กับเท็กซ์เจอร์ที่ทำการกำหนดลงไปเดียวกัน ณ เวลาที่ให้มา ในกรณีนี้เราจะทำการกำหนดเท็กซ์เจอร์เพียงครั้งเดียวแล้วเรนเดอร์โพลีกอนที่ใช้เท็กซ์เจอร์นี้ทั้งหมด แล้วจึงทำการเรนเดอร์ถัดๆไปที่มีอยู่

เราจะทำการจัดกลุ่มของโพลีกอนทั้งหมดเข้าด้วยกันโดยอิงกับเท็กซ์เจอร์ เราต้องทำการแสกนโครงสร้างทรีและสร้างลิสต์ของโพลีกอนที่จำเป็นต้องถูกเรนเดอร์ จากนั้นทำการตรวจสอบลิสต์ของโพลีกอนว่าขึ้นอยู่กับเท็กซ์เจอร์ใดแต่ละตัวแล้วกำหนดการทำงานให้กับโพลีกอนนั้นๆ

4.5 การเพิ่มอ็อบเจ็กต์สามมิติลงในฉาก

ในฉากที่เป็น 3 มิติจะไม่สมบูรณ์ถ้าขาดออบเจ็กต์ แต่ถ้าเราทำการวาดออบเจ็กต์หลายพันตัวโดยปราศจากคลิปปิ้งใดๆจะทำให้เสียเวลาอย่างมากในไปป์ไลน์ของการเรนเดอร์ ดังนั้นเราจึงจำเป็นต้องทำการใช้ viewing frustum เพื่อที่จะทำการกำหนดออบเจ็กต์ว่าอยู่ในวิวหรือไม่ด้วยความรวดเร็ว ในขั้นตอนของการกำหนดออบเจ็กต์ที่มองเห็นได้ต้องทำการห่อหุ้มแต่ละออบเจ็กต์สามมิติด้วย bounding sphere ซึ่งเป็นวัตถุทรงกลมที่มองไม่เห็นถูกใช้ในการทำงานร่วมกับ viewing frustum ในรูปที่ 4.6 แสดงถึงฉากที่ประกอบไปด้วยออบเจ็กต์ 3 อันและ viewing frustum แต่ละ mesh จะมี bounding sphere ที่มองไม่เห็นล้อมรอบอยู่ เราใช้ทรงกลมในการพิจารณาการมองเห็น ถ้าหากว่ามันอยู่บนด้านหน้าของเพลนทั้ง 6 เพลนที่เราสร้างขึ้นมาเป็น viewing frustum



รูปที่ 4.6 ใช้ viewing frustum ในการทดสอบการมองเห็นของวัตถุ

ในรูปที่ 4.6 จะมีเพียงสองออบเจ็กต์ที่สามารถมองเห็นได้ใน viewing frustum มีออบเจ็กต์อยู่ 1 อันที่อยู่ภายนอก frustum โดนสิ้นเชิงเราต้องทำการวาดเฉพาะออบเจ็กต์ที่สามารถมองเห็นได้และกระโดดข้ามออบเจ็กต์ที่อยู่ภายนอก frustum ซึ่งการจะทำเช่นนี้ได้เราต้องทำการคำนวณแต่ละ bounding sphere ของออบเจ็กต์และจากนั้นจึงทำการทดสอบว่าวออบเจ็กต์ภายในทรงกลมนั้นสามารถมองเห็นได้ถ้าหากว่าพวกมันอยู่ใน frustum

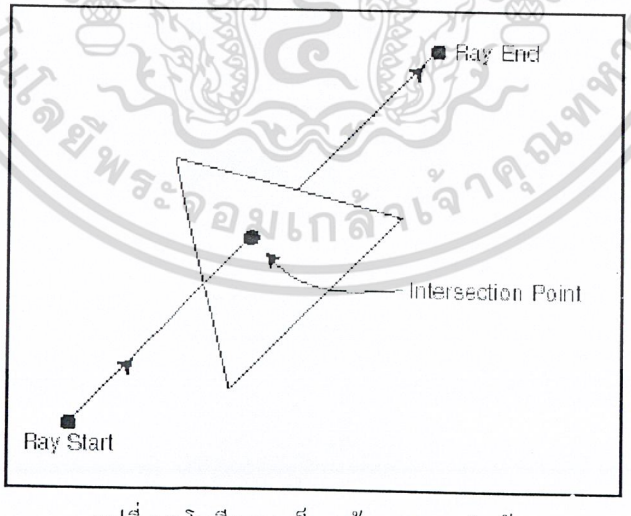
4.6 Collision Detection with Meshes

เป็นการตรวจสอบการชนกัน เมื่อออบเจ็กต์สามมิติชนกันกับออบเจ็กต์บนฉากหรือเมื่อผู้ใช้คลิก mesh ด้วยเมาส์ซึ่งจะทำให้เราทราบได้ว่าตัวละครของเราที่กำลังเดินไปรอบๆไปชนกับกำแพงหรือไปชนกับออบเจ็กต์เช่นตัวละครอื่นๆเป็นต้น ดังนั้นเกมเอนจิ้นจะต้องทราบว่าจะต้องบล็อกทางเดินของตัวละครถ้าหากเกิดการชนของออบเจ็กต์

4.6.1 Casting a Ray

ในลำดับของการตรวจสอบการบล็อคเส้นทางจากจุดหนึ่งไปยังจุดอื่นๆใช้การคาสต์ลำเส้นระหว่างจุด 2 จุดด้วยกันแล้วทำการตรวจสอบการตัดกันกันกับเพลน โดยการสร้างเพลนที่ใช้แสดงโพลีกอนเราสามารถใช้พีชคณิตในการหาจุดตัดแสดง (อ้างอิงรูปที่ 4.6) เราจะใช้การ dot product และการคำนวณหาระยะห่างระหว่างจุดกับเพลนในการหาจุดตัดระหว่างเพลนกับจุด ตามจุดที่ย้ายไปอยู่ในพื้นที่สามมิติ จุดจะมีการเคลื่อนย้าย ทำการสร้างเส้นที่แสดงเส้นทางของออบเจ็กต์ที่ใช้ระหว่างการเคลื่อนย้าย

รูปที่ 4.7 จะเห็นโพลีกอนและเส้น เส้นจะแสดงถึงลำเส้นที่คาสต์จากจุดเริ่มต้นไปยังจุดสิ้นสุด ลำเส้นจะไปตัดกับโพลีกอนที่ครึ่งทางด้านล่างของลำเส้น



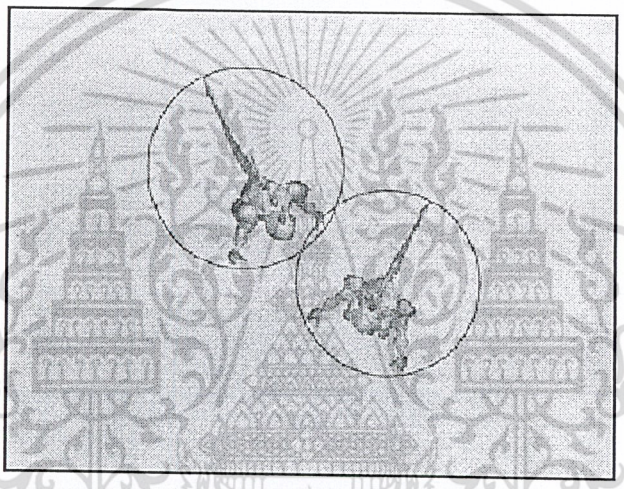
รูปที่ 4.7 โพลีกอนบล็อคเส้นทางของลำเส้น

ลำเส้นจะตัดกับเพลนอยู่เสมอจะไม่ขนานไปกันเพลน เราต้องสามารถบอกว่าเกิดการตัดกันภายในด้านใดของโพลีกอน ในส่วนนี้จะมีส่วนช่วยเหลือของ D3DX คือฟังก์ชัน D3DXIntersect สามารถทดสอบการตัด

กัน. มั่นใจได้ว่าจุดของลำเส้นที่ตัดไปยังเพลนอยู่ภายในโพลีกอน. ได้รับค่าพิกัดที่แน่นอนของจุดตัด และแสดงระยะทางจากจุดเริ่มต้นของลำเส้นไปยังจุดตัด

4.6.2 การชนกันของ Mesh

การตรวจสอบการชนกันระหว่างอ็อบเจ็กต์ (object-to-object collision detect) เราจะใช้ bounding sphere แต่ก่อนที่จะใช้เราต้องทำความเข้าใจบางอย่างเกี่ยวกับการใช้ bounding sphereโดยตามรูปที่ 4.8 แสดงมอนสเตอร์สองตัวซึ่งพวกมันมีหางที่ค่อนข้างยาวหางเหล่านี้มีผลกระทบต่อขนาดทั้งหมด bounding sphere สำหรับ mesh ทรงกลมจะมีขนาดที่ใหญ่มากห้อมล้อม mesh ข้างใน (ซึ่งรวมถึงหางด้วย) หากเราทำการเคลื่อนย้ายมอนสเตอร์สองตัวเข้าหากันดังรูปที่ 4.8 เราจะเห็นได้ว่า bounding sphere สองอันเกิดการตัดกันทั้งๆที่มอนสเตอร์ทั้งสองตัวยังไม่ได้อยู่ติดกัน



รูปที่ 4.8 bounding sphere มีขนาดใหญ่มากเกินความจำเป็นในระหว่างการตรวจสอบการชนกัน

แทนที่เราจะใช้ bounding sphere ของ mesh เราจะทำการคำนวณหารัศมีของ bounding sphere ของเราเองสำหรับในแต่ละ mesh เราสามารถทำการปรับการครอบคลุมของจำนวนพื้นที่ที่ล้อมรอบ mesh ได้อย่างรวดเร็ว เมื่อแก้ปัญหาแล้วเราสามารถกลับมาใช้ฟังก์ชันที่ใช้ในการตรวจสอบการตัดกันของสอง bounding sphere วิธีที่เราจะทราบได้ว่าทรงกลม 2 ลูกตัดกันหรือไม่โดยการคำนวณระยะทางระหว่างจากจุดศูนย์กลางของทรงกลมถ้าหากระยะทางน้อยกว่าหรือเท่ากันกับรัศมีสองรัศมีรวมกันแสดงว่าทรงกลมมีการตัดกัน

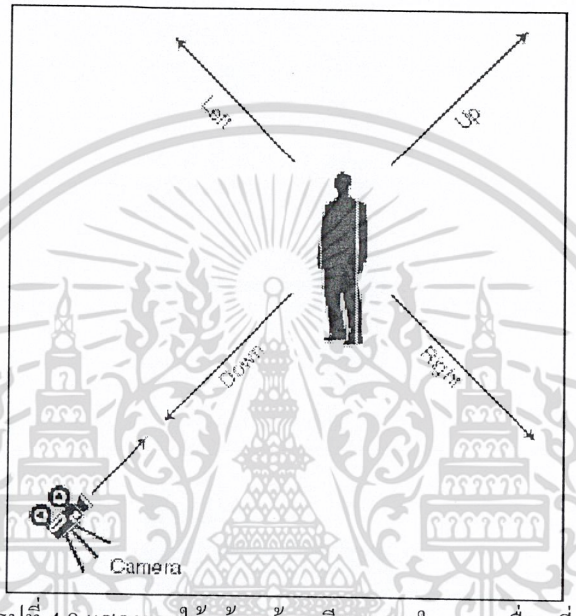
4.7 การควบคุมผู้เล่น(Controlling Player Characters)

ในเกมผู้เล่นจะมีบทบาทที่สำคัญฉะนั้นเราต้องควบคุมตัวผู้เล่นให้เป็นไปตามความต้องการของเรา โดยทั่วไปผู้เล่นในเกมก็จะการควบคุมทิศทางในการเคลื่อนที่ซึ่งทิศทางการเคลื่อนที่นั้นเราสามารถควบคุมการเคลื่อนที่ด้วยมือได้ไม่ว่าจะเป็นการเคลื่อนที่ไปข้างหน้า การเคลื่อนที่ไปด้านหลัง การเคลื่อนที่ไปซ้ายและการเคลื่อนที่ไปทางขวาซึ่งการเคลื่อนที่นี้สามารถไปได้ทุกหนทุกแห่งที่อยู่ในแผนที่(MAP) ใน

การควบคุมตัวผู้เล่นจะมีวิธีการควบคุม 2 วิธีคือ การควบคุมทิศทางการเคลื่อนที่(Directional Control) และการควบคุมการหมุนของตัวผู้เล่น (Rotational Control)

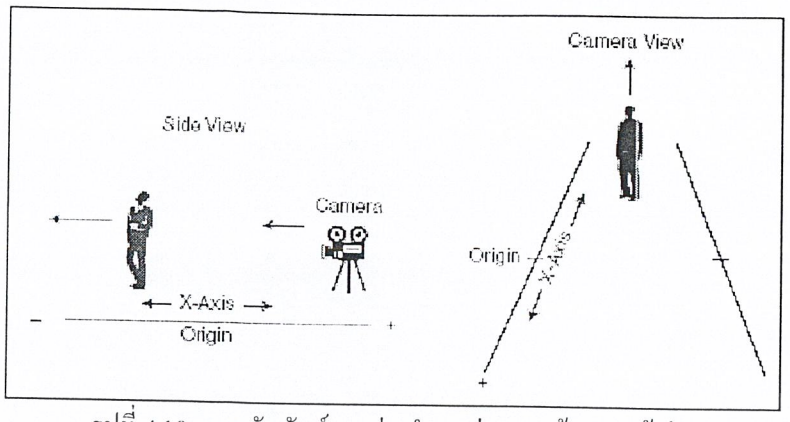
4.7.1 การควบคุมทิศทางการเคลื่อนที่(Directional Control)

การควบคุมทิศทางการเคลื่อนที่นั้นจะใช้ลูกศรบนคีย์บอร์ดหรือจอยสติ๊ก(joystick)ควบคุการเคลื่อนที่ ในการควบคุมทิศทางการเคลื่อนที่ๆดีที่สุดนั้นเราจะใช้กล้องเข้ามาช่วยเพื่อให้เกิดมุมมองของผู้เล่นโดยผู้เล่นสามารถมองจากด้านบนและมองจากด้านข้างได้ดูรูปที่ 4.9ประกอบ



รูปที่ 4.9 แสดงการใช้กล้องเข้ามามีบทบาทในการเคลื่อนที่

จากรูปที่ 4.9 เมื่อผู้เล่นเดินเคลื่อนที่ภายใต้การควบคุมของการกดคีย์บอร์ดซึ่งมีกล้องตั้งอยู่บนสกรีนฉะนั้นเมื่อเรากดคีย์ลูกศรบนผู้เล่นก็จะเคลื่อนที่ไปข้างหน้าตามมุมมองของกล้องซึ่งจะทำให้ผู้เล่นนั้นห่างจากกล้องไป ถ้าเรากดคีย์ลูกศร้ายผู้เล่นก็จะเคลื่อนที่ไปด้านซ้ายหรือถ้าเรากดคีย์ลูกศรขวาผู้เล่นก็จะเดินไปทางขวาตามมุมมองของกล้อง วิธีการที่มีการนำกล้องเข้ามาช่วยควบคุมการเคลื่อนที่นั้นจะต้องพิจารณาตำแหน่งของกล้องเป็นหลักเมื่อผู้เล่นเกิดการเคลื่อนที่ ดังนั้นเมื่อเรากดคีย์ลูกศรบนผู้เล่นก็จะเคลื่อนที่ไกลออกไป ในการเคลื่อนที่ของผู้เล่นจะมีการควบคุมทิศทางการเคลื่อนที่ ซึ่งเราจะต้องรู้มุมของกล้อง ตำแหน่งผู้เล่นและทิศทางที่ผู้เล่นต้องการเคลื่อนที่ไปเช่นเรากำหนดให้ 0.0เป็นการเดินไปข้างหน้าหรือห่างจากกล้องไป 1.57เป็นการเดินไปทางขวา 3.14เป็นการเดินถอยหลัง และ4.17เป็นการเดินไปทางซ้าย



รูปที่ 4.10 ความสัมพันธ์ระหว่างตำแหน่งของกล้องและผู้เล่น

รูปที่ 4.10 แสดงให้เห็นความสัมพันธ์ระหว่างตำแหน่งของกล้องและตำแหน่งของผู้เล่นซึ่งผู้เล่นต้องการเดินไปข้างหน้าโดยตำแหน่งของผู้เล่นจะมีค่า X=-4 ,Z=0

4.7.2 การควบคุมการหมุน(Rotational Control)

ในการควบคุมการหมุนผู้เล่นสามารถหมุนได้โดยใช้การควบคุมจากคีย์บอร์ดคือลูกศรซ้าย ลูกศรขวาและในการเดินหน้าและการเดินถอยหลังจะใช้คีย์ลูกศรบนและคีย์ลูกศรล่าง การควบคุมการหมุนจะดีกว่าการควบคุมทิศทางในบางลักษณะเพราะว่าในการเคลื่อนที่จะมีการคำนวณที่มากกว่าผู้เล่นแต่ละคนจะเก็บค่าทิศทางการเคลื่อนที่ในขณะนั้น อย่างไรก็ตามทิศทางการเคลื่อนที่สามารถแสดงด้วยค่าดังต่อไปนี้ 0.0 เป็นจุดที่อยู่ตามแกน positive Z, 1.57 เป็นจุดที่อยู่ตามแกน positive X, 3.14 เป็นจุดที่อยู่ตามแกน negative Z และ 4.71 เป็นจุดที่อยู่ตามแกน negative X ณ.จุดใดๆเราสามารถกดคีย์ลูกศรซ้ายแล้วให้เกิดการหมุนของผู้เล่นไปทางซ้าย(negative rotation)และกดคีย์ลูกศรขวาแล้วให้เกิดการหมุนของตัวผู้เล่นไปทางขวา(positive rotation)

4.8 การควบคุมตัวละครที่ไม่ใช่ผู้เล่น(Controlling Non-Player Characters)

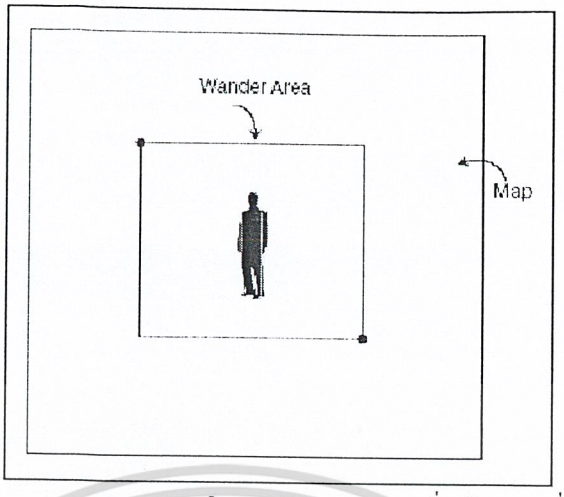
ในการควบคุมตัวละครที่ไม่ใช่ผู้เล่นนั้นเราสามารถคาดเดาได้ว่ายุ่งยากสลับซับซ้อนกว่าการควบคุมผู้เล่นในหัวข้อที่ผ่านมา ซึ่งในหัวข้อนี้จะแสดงถึงวิธีการอันหลากหลายในการนำทางให้กับตัวละครที่ไม่ใช่ผู้เล่น(Non-Player Characters หรือ NPC) ใช้เดินในเกมโดยทั่วไปการเดินหรือการเคลื่อนที่ของ NPC นั้นจะมี 5 ชนิดดังนี้

4.8.1 Standing Still

ตัวละครที่เป็น NPC จะยืนอยู่นิ่งๆโดยไม่มีการเคลื่อนที่ไปไหนภายในบริเวณพื้นที่ๆกำหนด

4.8.2 Wandering an Area

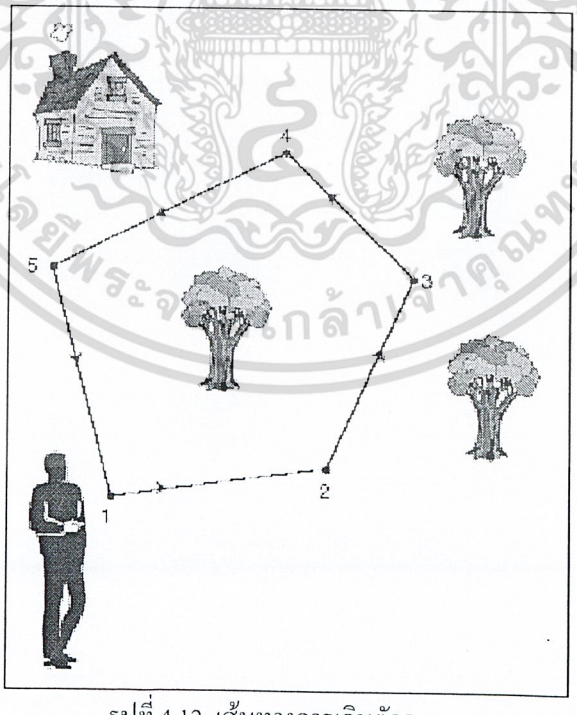
ในเกมออนไลน์อย่างเช่นเกม Ultima Online จะยอมให้ตัวละคร NPC เดินภายในพื้นที่ๆกำหนดไว้ซึ่งการดำเนินการจะไม่ซับซ้อนนัก เราสามารถที่จะระบุหรือกำหนดขอบเขตพื้นที่ให้ตัวละคร NPC เดินภายในขอบเขตพิกัดที่กำหนดได้ดูตามรูปที่ 4.11



รูปที่ 4.11 แสดงขอบเขตการเดินของตัวละคร NPC ที่อยู่ในแผนที่

4.8.3 Walking a Route

ในการเดินของตัวละครนั้นมันไม่ฉลาดพอที่จะเดินไปตามเส้นทางที่อยู่รอบๆ ได้ ฉะนั้นเราจะต้องทำการกำหนดเส้นทางการเดินให้กับตัวละคร NPC เส้นทางที่พูดถึงนี้จะรวมไปถึงการกำหนดพิกัดเพื่อให้สามารถเดินไปตามพิกัดที่เรากำหนดขึ้นมาได้ ในการสร้างเส้นทางเราจะต้องมีการเลือกจุดที่ต้องการให้ตัวละคร NPC เดินและเราจะสร้างอาร์เรย์ที่ชื่อ sRoutePoint ที่เป็นโครงสร้างข้อมูลเพื่อที่จะเก็บจุดพิกัดที่เรากำหนดได้ ดูรูปที่ 4.12

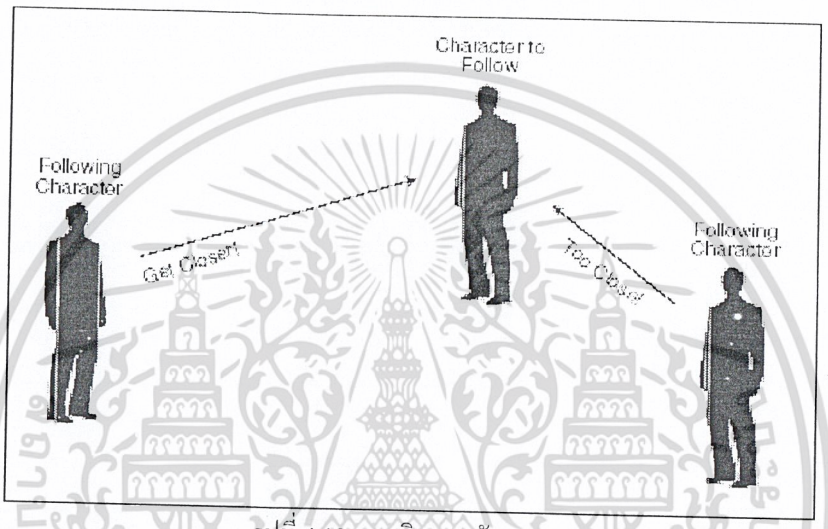


รูปที่ 4.12 เส้นทางการเดินตัวละคร

รูปที่ 4.12 แสดงจุดพิกัดที่ให้ตัวละคร NPC เดินซึ่งมีจุดเริ่มต้นในการเดินที่หมายเลข1และเดินต่อไปที่จุดหมายเลข2 3 4 และ5ซึ่งเป็นจุดสุดท้ายและจะกลับมาเริ่มต้นเส้นทางที่หมายเลข1อีกครั้ง

4.8.4 Following Another Character

การเดินของตัวละคร NPC ที่เดินติดตามกันไปเป็นวิธีการที่ไม่ยุ่งยากเพราะว่าตัวละคร NPC ตัวแรกที่เดินไปตามเส้นทางก่อนนั้นจะรู้พิกัดเส้นทางของตัวเองและตัวละคร NPC ตัวที่สองก็จะเดินไปตามเส้นทางที่ตัวแรกเดินไป ข้อแตกต่างอย่างเดียวของตัวละคร NPC ตัวอื่นๆอยู่ที่ระยะทางที่ระบุจากตัวละคร NPC ตัวอื่นๆไปหาตัวละคร NPC ตัวแรกซึ่งแสดงในรูปที่ 4.13



รูปที่ 4.13 การเดินตามตัวละคร

รูปที่ 4.13 ตัวละครตัวแรกคือตัวที่อยู่กลางจะเดินไปตัวละครอื่นๆก็จะเดินตามมา ถ้าตัวละครที่เดินตามมาอยู่ใกล้ตัวแรกมากกว่าระยะทางที่กำหนดตัวละครที่เดินตามมาก็จะหยุดนิ่ง ถ้าจุดที่ตัวละครยืนอยู่ห่างจากตัวแรกเกินค่าระยะทางที่ระบุตัวละครตัวนั้นก็จะเดินตามตัวแรกมา

4.8.5 Evading Another Character

เป็นวิธีการเดินที่หลบหลีกหมายความว่าตัวละคร NPC จะเดินห่างจากผู้เล่นอื่นๆ ถ้าตัวละคร NPC อยู่ใกล้กับผู้เล่นเกินกว่าค่าระยะทางต่ำสุดที่เซตไว้ ตัวละคร NPC จะหลบหลีกโดยเคลื่อนที่ไปในทิศทางตรงกันข้าม โดยจะทำผ่านการใช้ฟังก์ชั่น CalculateEvadeMovement

บทที่ 5
ระบบผู้เล่นหลายคน

5.1 Networking with DirectPlay

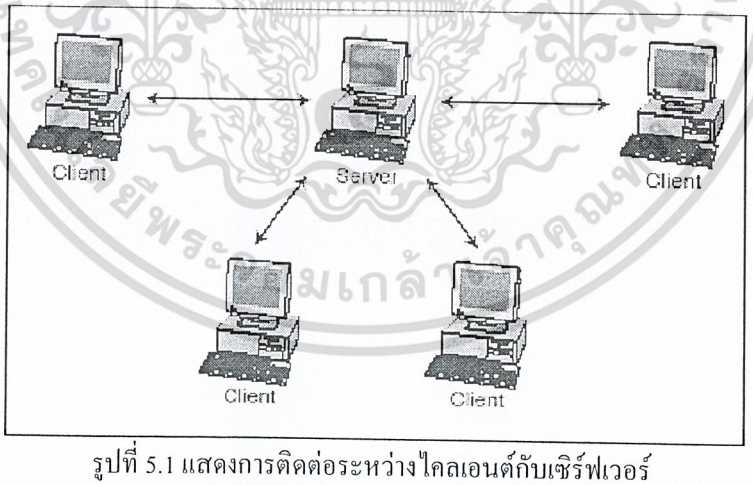
DirectPlay จะยอมให้ผู้เล่นติดต่อกับเกมที่เล่นเป็นกลุ่มได้หลายวิธี ไม่ว่าจะเป็นการติดต่อผ่านทางโมเด็มหรือการเชื่อมต่อผ่านระบบเครือข่ายการเชื่อมต่อแต่ละประเภทนั้นจะเรียกว่าเป็น Connection Type โดยส่วนใหญ่Connection Typeที่เป็นมาตรฐานที่DirectPlayมีให้นั่นก็คือ TCP/IPและIPX เป็นต้น การใช้ DirectPlayจะต้องมีการอ้างอิงรูปแบบการเข้าถึงเครือข่ายด้วยว่าจะเข้าถึงเครือข่ายแบบใดเช่น อ้างถึงเครือข่ายแบบไคลเอนต์/เซิร์ฟเวอร์หรือเครือข่ายแบบPeer-to-Peer ซึ่งแต่ละแบบก็จะมีอินเตอร์เฟซเป็นของตัวเอง รูปแบบของเครือข่ายพื้นฐานมีดังนี้คือ

-Server

-Client

-Peer-to-Peer

แต่ละรูปแบบจะมีการต่อคอมพิวเตอร์เข้าด้วยกันเพื่อที่จะแชร์ข้อมูลต่างๆ ซึ่งแต่ละโมเดลก็จะขึ้นอยู่กับความต้องการของแอพพลิเคชั่นของเราและต้องดูข้อดีข้อเสียของแต่ละโมเดลที่จะใช้ด้วยในส่วนโมเดลของไคลเอนต์/เซิร์ฟเวอร์ เราจะใช้ เซิร์ฟเวอร์เพื่อเป็นศูนย์กลางของระบบเครือข่ายและคอมพิวเตอร์เครื่องอื่นๆที่ต่อเข้ากับเครื่องเซิร์ฟเวอร์จะเรียกว่าไคลเอนต์และเมื่อเกิดเชื่อมต่อกันระหว่างไคลเอนต์/เซิร์ฟเวอร์ก็จะเริ่มมีการส่งและรับข้อมูล



รูปที่ 5.1 แสดงการติดต่อระหว่างไคลเอนต์กับเซิร์ฟเวอร์

รูปแบบของเครือข่ายที่ดีที่สุดนั้นจะขึ้นอยู่กับแอพพลิเคชั่นที่ใช้และโมเดลแบบไคลเอนต์/เซิร์ฟเวอร์นั้นจะเหมาะสำหรับเครือข่ายที่มีมากกว่า 4 เครื่อง ส่วนโมเดลแบบPeer-to-Peerนั้นจะเหมาะสำหรับเครือข่ายขนาดเล็กและเป็นการติดต่อโดยตรงระหว่างเครื่อง

สำหรับเกมบางเกมนั้นจะใช้โมเดลแบบPeer-to-Peerเพราะว่ามียูสเซอร์ที่เล่นเกมแค่4-8คนเช่นเกม Diabo แต่สำหรับเกมออนไลน์ที่มียูสเซอร์เป็นพันๆคนนั้นก็ต้องใช้โมเดลแบบไคลเอนต์/เซิร์ฟเวอร์มา

จัดการ ในการเล่นเกมออนไลน์นั้นก็จะมียูสเซอร์หลายคนมาเล่นและเราจะหายูสเซอร์ใหม่ที่เข้ามาเล่นเกมออนไลน์ได้โดยใช้ล็อบบี้เซิร์ฟเวอร์ซึ่งล็อบบี้เซิร์ฟเวอร์นั้นจะเป็นเหมือนห้องโถงสำหรับคนเล่นเกม โดยล็อบบี้จะให้ผู้เล่นทุกคนสามารถล็อกอินเข้ามาทำการติดต่อกันและเล่นเกมร่วมกันได้โดยล็อบบี้เป็นที่ๆผู้เล่นใช้เป็นจุดเริ่มต้นใช้งานแอพพลิเคชั่นและใช้แลกเปลี่ยนข้อมูลข่าวสารระหว่างผู้เล่น

5.2 Latency Time and Lag

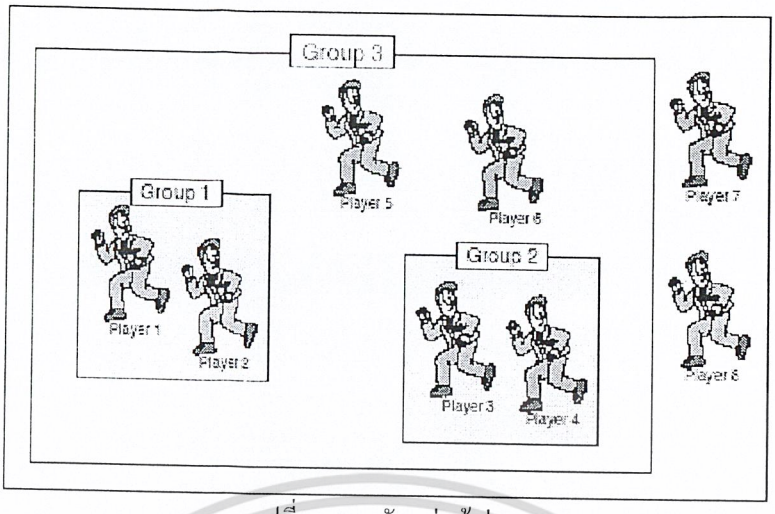
Latency จะเป็นเวลาที่เครื่องไคลเอนต์ส่งข้อมูลไปที่เซิร์ฟเวอร์เพื่อทำการประมวลผลและLag เป็นเวลาหน่วงในการติดต่อสื่อสารระหว่างเครื่องไคลเอนต์กับเซิร์ฟเวอร์ ซึ่งจะเป็นเวลาที่ส่งข้อมูลไปประมวลผลและเวลาที่ได้รับการตอบสนองกลับมา

5.3 Communication Protocols

เครือข่ายแต่ละเครือข่ายสามารถติดต่อกันได้หลายเครือข่ายแต่ว่าเพื่อให้เกิดความเข้าใจระหว่างเครือข่ายแล้วจะต้องใช้โปรโตคอลที่เหมือนกันในการติดต่อสื่อสาร ซึ่งในปัจจุบันโปรโตคอลที่นิยมใช้กันมากก็คือโปรโตคอลTCP/IP(Transfer Control Protocol/Internet Protocol)

5.4 Networking with Players

ใน DirectPlay ผู้เล่น(player)ก็จะมีการเชื่อมต่อเป็นหนึ่งการเชื่อมต่อ(ผู้ที่เล่นเกม)ซึ่งไม่ว่าจะเชื่อมต่อกับเครื่องคอมพิวเตอร์ใดในระบบเครือข่ายก็จะเป็นหนึ่งการเชื่อมต่อ คอมพิวเตอร์หนึ่งเครื่องสามารถที่จะมีผู้เล่นหลายๆคนได้แต่ทั่วไปแล้วจะมีได้แค่คนเดียวเท่านั้นที่เล่นเกมได้ ในความเป็นจริงแล้วเซิร์ฟเวอร์จะกำหนดหมายเลขไอดีให้ผู้เล่นแต่ละคน ผู้เล่นแต่ละคนก็จะได้รับหมายเลขประจำตัว(Player ID)ซึ่งระบบจะใช้การส่งแมสเสจโดยตรงไปที่ผู้เล่นนั้นๆ สำหรับเกมที่มีขนาดใหญ่ๆคือสามารถมีผู้เล่นเข้ามาเล่นได้เป็นพันๆคนนั้นจะต้องมีการจัดการกับผู้เล่นทุกคนในเกมซึ่งผู้เล่นบางคนที่เล่นเกมหรือทุกคนจะต้องมีการกำหนดเป็นกรุ๊ป การใช้การกรุ๊ปนั้น จะเป็นการจัดการผู้เล่นที่มากๆให้เกิดความสะดวกหรือง่ายกว่าที่จะต้องจัดการกับผู้เล่นแต่ละคนทุกคนในเกม โดยเราสามารถรวมกลุ่มของผู้เล่นในAreaเดียวกันและการส่งข้อมูลก็จะส่งข้อมูลของกรุ๊ปนั้นๆไปทั้งหมดเพียงครั้งเดียวดีกว่าที่จะส่งข้อมูลไปที่ผู้เล่นแต่ละความสัมพันธ์ของผู้เล่นและกรุ๊ปนั้นสามารถดูได้ตามรูปที่ 5.2 ซึ่งกรุ๊ปแต่ละกรุ๊ปจะไม่มีการจำกัดจำนวนของผู้เล่นและไม่มีการจำกัดจำนวนของกรุ๊ปซึ่งเราสามารถสร้างได้



รูปที่ 5.2 การจัดกลุ่มผู้เล่น

รูปที่ 5.2 มีผู้เล่น8คนที่เชื่อมต่อเข้ามาในเกมผู้เล่น6คนจะรวมเป็นกลุ่มและผู้เล่นอีก2คนจะแยกจากกลุ่ม

5.5 Networking with Messages

แมสเสจเป็นประเภทแพ็กเก็จข้อมูลซึ่งแต่ละแมสเสจก็จะมีความหมายเฉพาะเป็นของตัวมันเองจะกำหนดไว้ในแมโครและการใช้งานแมสเสจนั้นจะขึ้นอยู่กับโมเดลของแต่ละเครือข่ายที่ใช้ด้วยเช่น ถ้าเราใช้โมเดลแบบClient เราจะไม่ได้รับแมสเสจของโมเดลแบบPeer-to-Peer ดังนั้นเพื่อที่จะรับแมสเสจได้โมเดลของเครือข่ายจะต้องกำหนดให้มีการเรียกฟังก์ชั่นรับแมสเสจทุกครั้งเมื่อมีแมสเสจเข้ามาDirectPlayจะมีความสามารถในการรับ-ส่งแมสเสจคือ การรับ-ส่งแมสเสจแบบอะซิงโครนัส(Asynchronous) ความหมายก็คือระบบจะReturn การควบคุมมาที่ยูสเซอร์หลังจากที่เราให้คำสั่งในการส่งข้อมูลหรือการรับ-ส่งแมสเสจแบบซิงโครนัส(Synchronous)ก็คือจะรอจนกว่าข้อมูลที่ส่งไปนั้นจะเสร็จสมบูรณ์(synchronous) การรับ-ส่งแมสเสจแบบอะซิงโครนัสเป็นวิธีการที่ส่วนมากต้องการใช้ซึ่งวิธีการรับ-ส่งแมสเสจแบบนี้จะไม่สนับสนุนระบบที่ใช้วิธีการรับ-ส่งแมสเสจแบบซิงโครนัสตัวอย่างเช่น ถ้าเราส่งแมสเสจที่มีจำนวนมากๆและพยายามที่จะเล่นเกมไปด้วยในเวลาเดียวกัน เราก็ต้องการที่จะไม่ให้เกิดเหตุการณ์ใดๆในระหว่างที่เครือข่ายพยายามส่งข้อมูลทั้งหมดนั้น ขณะเดียวกัน DirectPlay ก็จะจัดการกับข้อมูลในเครือข่ายนั้นเพื่อให้ยูสเซอร์สามารถเล่นเกมต่อไปได้ซึ่งในDirectPlayก็จะมีแมสเสจที่เป็นมาตรฐานดังแสดงในรูปที่ 5.3 ในส่วนของการรับประกัน(Guaranteed Delivery)ในการส่งข้อมูลนั้นจะต้องการันตีว่าแมสเสจที่ส่งไปนั้นมีการส่งไปถึงปลายทาง ซึ่งDirectPlayจะการันตีการส่งถึงของแมสเสจด้วย(ทำการส่งแมสเสจจนกระทั่งการส่งนั้นสำเร็จลุล่วง) ในส่วนของควบคุมการไหลของข้อมูล(Throtting) ณ.เวลาหนึ่งระบบจะกลายเป็นโอเวอร์โหลด(Overload) ฉะนั้นจะต้องพยายามจัดการกับแมสเสจที่เข้ามาอย่างไรก็ตาม DirectPlayมีระบบการควบคุมการไหลของแมสเสจซึ่งจะทำการdiscardแมสเสจที่มีลำดับความสำคัญต่ำที่ส่งมาจากคิว(queue)

Message Macro	Description
DPN_MSGID_ADD_PLAYER_TO_GROUP	ผู้เล่นแอดเข้ามาในกรุ๊ปที่มีอยู่
DPN_MSGID_APPLICATION_DESC	แอพพลิเคชั่นร้องขอข้อมูล
DPN_MSGID_ASYNC_OP_COMPLETE	การส่งข้อมูลแบบอะซิงโครนัสแบบสมบูรณ์
DPN_MSGID_CLIENT_INFO	ไคลเอนต์ร้องขอข้อมูล
DPN_MSGID_CONNECT_CONPLETE	การเชื่อมต่อสำเร็จ
DPN_MSGID_SERVER_INFO	เซิร์ฟเวอร์ร้องขอข้อมูล
DPN_MSGID_CREATE_GROUP	ใช้สร้างกรุ๊ป
DPN_MSGID_CREATE_PLAYER	ใช้ในการสร้างผู้เล่น
DPN_MSGID_DESTROY_GROUP	ใช้ในการที่กรุ๊ปถูกลบ
DPN_MSGID_DESTROY_PLAYER	ใช้เมื่อผู้เล่น destroy
DPN_MSGID_RECEIVE	ใช้ในการรับข้อมูล
DPN_MSGID_TERMINATE_SESSION	ใช้ในการปิดการเชื่อต่อกับเครือข่าย

รูปที่ 5.3 แสดงแมสเสจมาตรฐานของ DirectPlay

DirectPlay จะแนะนำให้รู้จักวิธีการ Unicode ที่เป็นมาตรฐานสากลซึ่งอาจจะมีความแตกต่างของโปรแกรมและคอมพิวเตอร์ที่ทำการแชร์ information กันเพราะว่าเครือข่ายต่างๆมีการเชื่อมต่อกันกับเครือข่ายอื่นๆทั่วโลกเช่นเดียวกันเกมของเราก็อาจจะอยู่ในประเทศที่ต่างกันและใช้ภาษาต่างกันเพราะว่าในแต่ละประเทศจะมีการทำงานของWindowsที่ต่างกันดังนั้นผู้ใช้ระบบก็จะต้องมีการปรับแต่งการใช้งานถึงการใช้Unicode Character เช่นภายในระบบของDirectPlay จะมีการสร้างการใช้ Unicode Character (Single Characterใช้16บิตแทนที่8บิตในการเก็บข้อมูล)มีการจัดหาภาษาให้ใช้ได้ซึ่งจะใช้ Character มากกว่า 256 Character

5.6 Identifying Application with GUIDs

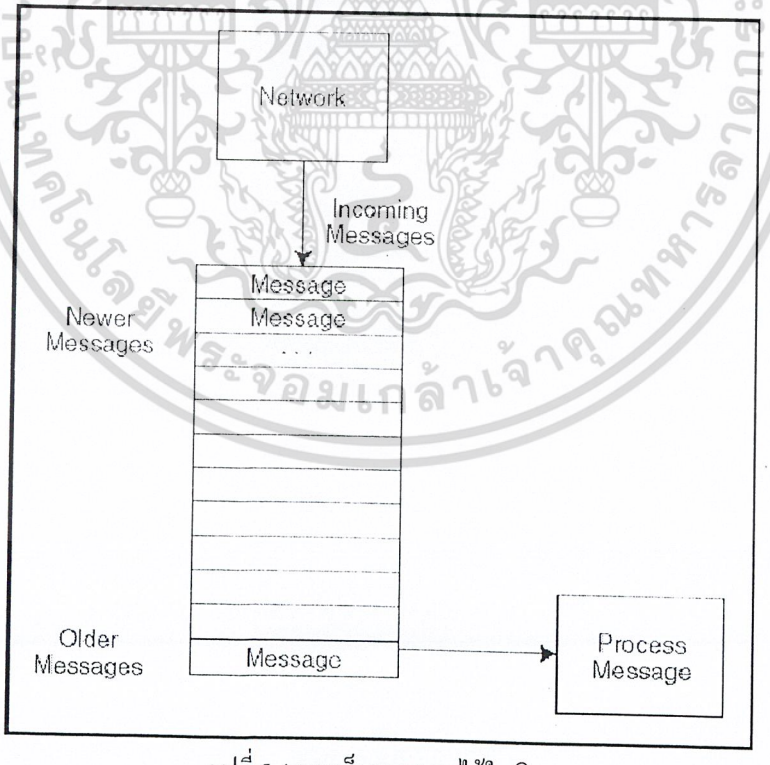
การติดต่อระหว่างเครื่องในเครือข่ายนั้นเราจำเป็นต้องรู้ GUID(Global Unique ID) เพราะว่ามีแอพพลิเคชั่นทางด้านเครือข่ายจำนวนมากที่ใช้ติดต่อสื่อสารกันฉะนั้นเราจะต้องทำการกำหนดแอพพลิเคชั่นของเราให้เป็น Unique Number ผ่านอินเตอร์เฟสของ DirectPlay และยอมให้มีแอพพลิเคชั่นเพียงตัวเดียวที่ใช้ Number เหมือนกันเพื่อเชื่อมต่อกับแอพพลิเคชั่นที่ต้องการ Number พิเศษนี้จะเป็น GUID(Global Unique ID) เมื่อเราสร้างแอพพลิเคชั่นขึ้นมาเพื่อติดต่อกันขณะเดียวกันจะต้องมีการกำหนด GUID ที่ไม่ซ้ำกันและมั่นใจว่าแอพพลิเคชั่นที่ต้องการจะเชื่อมต่อนั้นจะต้องมีการใช้ GUID ที่เหมือนกัน

5.7 การทำงานที่ฝั่งเซิร์ฟเวอร์(Working with Servers)

ขั้นแรกจะต้องทำการสร้างเซิร์ฟเวอร์ขึ้นมาก่อน โดยเซิฟร์เวอร์จะทำหน้าที่เป็นซีพียูประมวณผลเกมบนเครือข่ายผู้เล่นทุกคนจะทำการติดต่อกับเซิฟร์เวอร์ผ่านไคลเอนต์แอพพลิเคชั่นและเริ่มรับ-ส่งข้อมูล เซิร์ฟเวอร์จะทำการเก็บข้อมูลต่างๆของเกมและมีการซิงโครไนซ์(Synchronized)ข้อมูลระหว่างเซิร์ฟเวอร์กับไคลเอนต์

เซิร์ฟเวอร์ของเกมคือส่วนของซอฟท์แวร์พิเศษที่ไม่ต้องการภาพกราฟิกสวยงาม ไม่ต้องตรวจสอบอินพุทฟังค์ชั่น เซิร์ฟเวอร์เพียงแค่ต้องการที่จะประมวลผลการกระทำที่รับมาจากผู้เล่นที่เชื่อมต่อเข้ามา และทำการส่งอัพเดทไปยังไคลเอนต์ เมื่อเซิร์ฟเวอร์เริ่มรันจะเริ่มเข้าสู่ลูปทำการประมวลผลแมสเสจที่เข้ามาจากเครือข่าย อัพเดทผู้เล่นทุกคนที่เชื่อมต่อโดยอ้างอิงจากสถานะล่าสุดที่จำไว้แล้วทำการส่งแมสเสจในการอัพเดทกลับไปที่ไคลเอนต์

เมื่อรับแมสเสจเข้ามาแมสเสจจะถูกเก็บเข้าไปในแมสเสจคิว(Message queue)การใช้แมสเสจคิวเพิ่มความเร็วให้กับการจัดการเครือข่าย และปล่อยให้งานส่วนใหญ่อยู่ที่แอพพลิเคชั่นหลัก เซิร์ฟเวอร์จะจัดการ แมสเสจคิว (a stack of message) ซึ่งบรรจุแมสเสจทั้งหมด โดยเซิร์ฟเวอร์จะดึงเอาแมสเสจที่เก่าที่สุดออกมาแล้วเอาไปประมวลผล เซิร์ฟเวอร์จัดการกับ Connection ของผู้เล่นโดยขั้นแรกทำการตรวจสอบว่ามีช่องเก็บผู้เล่นเหลือว่างอยู่หรือไม่ ถ้ามีก็จะเอาข้อมูลรายละเอียดต่างๆของผู้เล่นเข้ามาเก็บไว้ในโครงสร้างข้อมูลภายในของเซิร์ฟเวอร์ เมื่อมีผู้เล่นยกเลิกการเชื่อมต่อก็จะทำการล้างช่องเก็บผู้เล่นนั้นๆ



รูปที่ 5.4 การเก็บแมสเสจไว้ในคิว

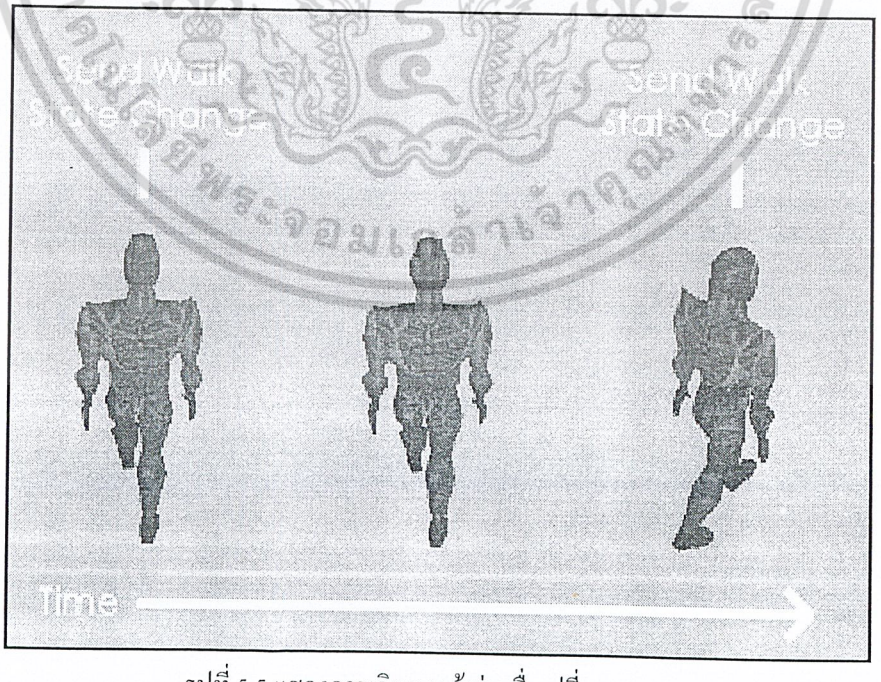
รูปที่ 5.4 แสดงแมสเสจจากเครือข่ายถูกใส่เข้าไปในคิวตามลำดับเพื่อให้แน่ใจว่าแมสเสจที่เก่าที่สุดจะเป็นอันต่อไปที่จะได้รับการประมวลผล

เซิร์ฟเวอร์สามารถทำ Collision detection เพื่อบอก client ได้ว่าไม่สามารถเดินทะลุกำแพงได้ เพราะเซิร์ฟเวอร์รู้พิกัดล่าสุดของตัวละครที่กำลังเคลื่อนที่แล้วตรวจสอบว่าไม่สามารถเดินต่อไปได้ ก็จะส่งแมสเสจไปบอกว่าทางที่กำลังจะเดินไปนั้นเป็นสิ่งกีดขวางด้วยการอนุญาตให้เฉพาะเซิร์ฟเวอร์เท่านั้นที่สามารถอัพเดทเกมทั้งหมด สามารถหลีกเลี่ยงผู้ไม่ประสงค์ดีที่มักจะส่งข้อมูลปลอมไปยังเซิร์ฟเวอร์ด้วยความมุ่งหวังว่าจะสามารถเคลื่อนย้ายตัวละครไปยังทางที่เป็นไปไม่ได้

5.8 การทำงานที่ฝั่งไคลเอนต์(Working with Clients)

การทำงานของไคลเอนต์จะไม่ซับซ้อนเท่าเซิร์ฟเวอร์โดยไคลเอนต์จะมีการใช้แมสเสจเพียง2 แมสเสจได้แก่แมสเสจreceive_data และแมสเสจ session_terminate และต้องทำการเชื่อมต่อและรักษาเซสชั่นของการเชื่อมต่อ หลักการของไคลเอนต์แอพพลิเคชั่นจะต้องมีการระบุผู้เล่นที่เครื่องโฮสให้สามารถรับข้อมูลได้

เมื่อเริ่มการเชื่อมต่อไคลเอนต์ต้องทำเพียงเก็บรายละเอียดการควบคุมตัวละครภายในเครื่องตัวเองแล้วส่งไปที่เซิร์ฟเวอร์ เมื่อรับการอัพเดทจากเซิร์ฟเวอร์ ไคลเอนต์จะทำการคาดเดาว่าจะจัดการสิ่งใดๆ เกี่ยวกับตัวละคร ฉาก และการแสดงผล โดยอ้างอิงจากสถานะล่าสุด โดยในการเคลื่อนที่นั้นไคลเอนต์จะส่งแมสเสจไปบอกเซิร์ฟเวอร์เพียงแค่ทิศทางในการเดินเท่านั้นและก็ทำการเคลื่อนที่ไปในทิศทางที่ผู้เล่นได้ควบคุม จนกว่าที่เซิร์ฟเวอร์จะส่งอัพเดทกลับมาว่าให้หยุดดังนั้นเพื่อลดการส่งแมสเสจลงก็จะทำการตรวจสอบการเปลี่ยนแปลงทิศทางการเดินเท่านั้น



รูปที่ 5.5 แสดงการเดินของผู้เล่นเมื่อเปลี่ยนสถานะ

รูปที่ 5.5 แสดงการเดินเมื่อไคลเอนต์ส่งการเปลี่ยนสถานะเฉพาะทิศทางการเดินที่แตกต่างจากทิศทางการเดินล่าสุด ซึ่งจะทำให้ประหยัดเวลา เพราะการเดินไปในทางเดิมไม่จำเป็นต้องบอกเซิร์ฟเวอร์ซ้ำๆ การอัพเดทผู้เล่นควรทำแค่ทุกๆ 33 ms. การจำกัดเวลาในการอัพเดทเพื่อป้องกันการฟลัด(flood) แมสเสจไปยังเซิร์ฟเวอร์มากเกินไป

5.9 การทำงานร่วมกันระหว่างไคลเอนเซิร์ฟเวอร์

5.9.1 เซิร์ฟเวอร์กับไคลเอนทำงานร่วมกันอย่างไรในมุมมองของเกม

เมื่อผู้เล่นสร้างความเคลื่อนไหวหรือการกระทำใดๆในเกม สิ่งที่ทำนั้นจะถูกส่งออกไปยังเซิร์ฟเวอร์ในรูปของแมสเสจเพื่อตรวจสอบ เซิร์ฟเวอร์มีหน้าที่ในการทำให้สิ่งที่ดำเนินไปในเกม มีความพร้อมกัน(Synchronization) รับการกระทำที่ผู้เล่นส่งมา อัพเดทโลกในเกมและส่งการอัพเดทให้กับไคลเอนต์ ด้วยประการนี้เซิร์ฟเวอร์จึงรองรับค้ำจุนเกมทั้งหมด ในขณะที่ฝั่งไคลเอนต์เพียงแค่ทำหน้าที่รวบรวมว่าผู้เล่นทำอะไรบ้างในเกมและแสดงผลเหตุการณ์ต่างๆ ออกมาที่จอภาพของผู้เล่น

5.9.2 ชนิดของแมสเสจ

5.9.2.1 Connection request เป็นการติดต่อกับเซิร์ฟเวอร์คือผู้เล่นเข้าร่วมเล่นเกม

5.9.2.2 Navigation ผู้เล่นสามารถเคลื่อนที่ไปรอบๆแผนที่

5.9.2.3 Combat การต่อสู้แบบต่างๆของผู้เล่นในเกม

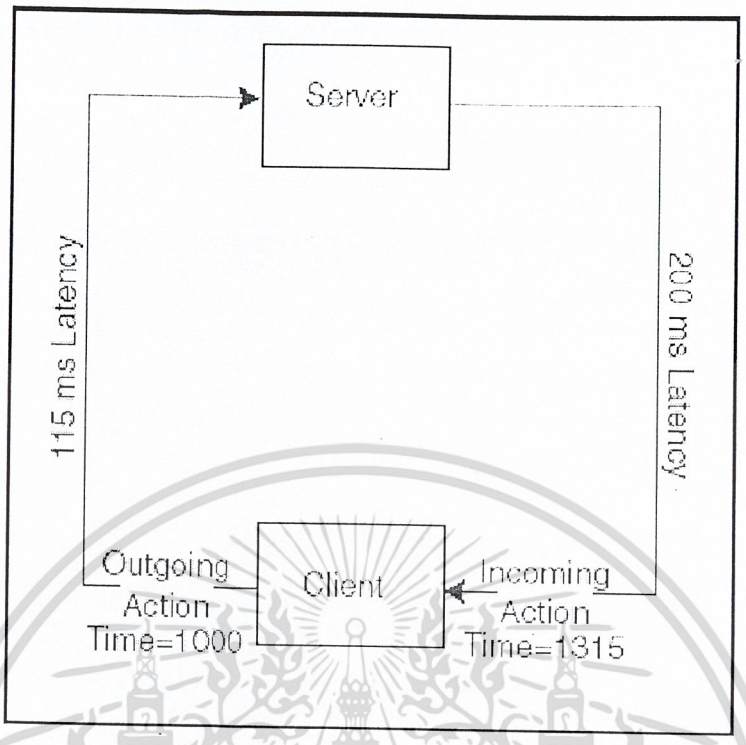
5.9.2.4 Resource management ในโลกที่เต็มไปด้วยสิ่งของ (item) ผู้เล่นสามารถที่จะซื้อขายแลกเปลี่ยน หรือใช้ สิ่งของต่างๆ การจัดการกับสิ่งของ มาจากฝั่งไคลเอน และ request ในการจัดทำนั้นจะถูกส่งไปยังเซิร์ฟเวอร์เพื่อทำการอัพเดท

5.9.2.5 Conversation ผู้เล่นสามารถพูดคุยกันได้

5.9.2.6 Game update เซิร์ฟเวอร์จำเป็นต้องทำให้ไคลเอนทั้งหมดรู้เป็นระยะๆ ว่าสถานะปัจจุบันของเกมเป็นอย่างไร โดยมักจะรวมตำแหน่งของผู้เล่นทั้งหมดในเกม รวมถึงข้อมูลเกี่ยวกับไอเท็ม(Item) และทรัพยากรในโลกของเกม

เซิร์ฟเวอร์เป็นระบบเดียวที่รับผิดชอบ การรองรับสถานะของเกม ไคลเอนทุกเครื่องที่เชื่อมต่อต้องรอการอัพเดทเพื่อให้เกมดำเนินไป ข้อมูลที่ส่งไปจากไคลเอนไปยังเซิร์ฟเวอร์และกลับมาจะถูกหน่วงเวลา ความหน่วงนี้ในการสื่อสารข้อมูลเรียกว่า Latency ซึ่งมีหน่วยวัดเป็น milliseconds

เนื่องจากเซิร์ฟเวอร์เป็นระบบเดียวที่อนุญาต ให้เกิดความเปลี่ยนแปลงในโลกของเกม เซิร์ฟเวอร์ต้องทำการตรวจสอบสิ่งที่ผู้เล่นทำให้ถูกต้องก่อนที่จะยอมให้มีการกระทำของผู้เล่นเกิดขึ้นในเกมจึงทำให้เกิดการหน่วงขึ้นที่เรียกกันว่า lag ดูรูปที่ 5.6 ประกอบ



รูปที่ 5.6 แสดงเวลาการส่งแมสเสจระหว่างไคลเอนต์กับเซิร์ฟเวอร์

จากรูปที่ 5.6 แสดงให้เห็นว่าไคลเอนต์ส่งแมสเสจไปที่เซิร์ฟเวอร์และไคลเอนต์ก็จะรอ การตอบรับจากเซิร์ฟเวอร์ทำให้เกิดการหน่วงเวลา

เพื่อที่จะทำให้สิ่งต่างๆดำเนินไปอย่างราบเรียบ และช่วยแบ่งเบาสิ่งที่เกิดขึ้นมาจาก Latency และ lag จะมีการอนุญาตให้ไคลเอนต์สามารถเปลี่ยนแปลงตนเองได้บางส่วนในช่วงที่เซิร์ฟเวอร์อัพเดทนั่นคือการเคลื่อนที่ เมื่อเป็นเช่นนั้นไคลเอนต์ไม่จำเป็นต้องรอการอัพเดทของเซิร์ฟเวอร์ในการเคลื่อนที่ตัวละคร ไคลเอนต์สามารถคาดเดาการเคลื่อนที่ของตนเองและตัวละครของผู้เล่นคนอื่น ได้จากสถานะล่าสุด (Last know state)ของผู้เล่นแต่ละคน การคาดเดาในลักษณะนี้เรียกว่า ***Dead Reckoning*** ซึ่งโดยทั่วไปใช้ในการทำทางด้าน Network Programming

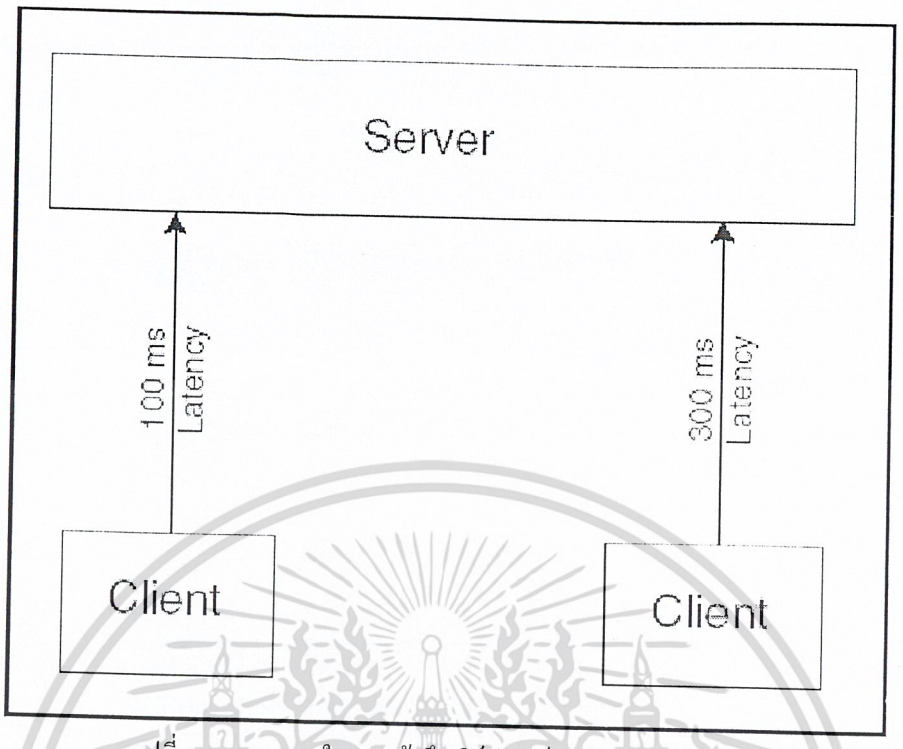
รูปที่ 5.7 แสดงการใช้ Dead Reckoning

จากรูปที่ 5.7 แสดงให้เห็นการใช้ Dead Reckoning ตัวละครที่เดินอยู่สามารถเดินต่อไปเรื่อยจนกว่าเซิร์ฟเวอร์จะบอกให้หยุด

เมื่อมีการกระทำใดๆอย่างอื่นนอกจากการเคลื่อนที่ เช่นการต่อสู้ การเข้าเล่นเกม การใช้ dead reckoning นั้นจะไม่เป็นที่ยอมรับ เมื่อระบบต้องการรู้ว่าใครโจมตีใครและได้รับความเสียหายแค่ไหน ระบบต้องคิวรี(query) จากเซิร์ฟเวอร์เพื่อดูความตั้งใจ

เครื่องคอมพิวเตอร์แต่ละเครื่องที่อยู่ในระบบเน็ตเวิร์คมี Latency ที่ต่างกันบางเครื่องใช้เวลามากในการส่งแมสเสจไปยังเซิร์ฟเวอร์แล้วรับกลับมา ไคลเอนต์ที่มีการเชื่อมต่อที่ความเร็วสูงกว่าก็จะได้เปรียบ

แมสเสจทั้งหมดที่รับมาถูกบันทึกเวลาที่รับตัวมันมา เซิร์ฟเวอร์ใช้เวลาดังกล่าวในการตัดสินใจว่าจะอัพเดทผู้เล่นอย่างไรเช่น หากแมสเสจที่เซิร์ฟเวอร์รับมาไม่ได้รับการประมวลผลภายในเวลา 100 ms เซิร์ฟเวอร์ทำการชดเชยสำหรับจำนวนเวลาดังกล่าวระหว่างการอัพเดทและชดเชยไปยังไคลเอนต์ด้วย ช่วงเวลาดังกล่าวจะชดเชยการเคลื่อนที่ที่เหมาะสมให้กับการทำ Dead Reckoning



รูปที่ 5.8 แสดงเวลาในการเข้าถึงเซิร์ฟเวอร์จากไคลเอนต์

จากรูปที่ 5.8 ไคลเอนต์ฝั่งซ้ายมี latency time น้อยกว่าไคลเอนต์ฝั่งขวา ฉะนั้นไคลเอนต์ฝั่งซ้ายจึงได้รับการประมวลผลที่เซิร์ฟเวอร์ก่อน

5.10 Handling Player

เมื่อเซิร์ฟเวอร์เริ่มทำงานแมสเสจแรกที่ได้รับเป็นแมสเสจสำหรับสร้างผู้เล่นผู้เล่นแรกจะทำการสร้างเป็น Host Player หลังจากนั้นผู้เล่นคนอื่น ๆก็จะเริ่มเข้ามาเล่นเกม โดยที่เซสชั่นของโฮสต้องคงอยู่

5.10.1 การจัดการกับแมสเสจ Create-Player

แมสเสจที่ใช้ในการสร้างผู้เล่นจะกำหนดโดยแมสเสจที่ชื่อว่าDPN_MSGID_CREATE_PLAYER และในการสร้างผู้เล่นนี้จะมีบัฟเฟอร์คอยเก็บจำนวนผู้เล่นที่เล่นอยู่ในเกมโดยการใช้การกำหนดโครงสร้าง (Structure) ในโครงสร้างข้อมูลก็จะมีการกำหนดให้ผู้เล่นแต่ละคนมีหมายเลขเป็นของตัวเองที่ไม่ซ้ำกันเพื่อใช้ในการอ้างถึงผู้เล่นคนนั้นๆ และจะมี ContextDataPtr ที่เป็นพอยเตอร์ที่ใช้เก็บข้อมูลของผู้เล่น มีการใช้งานร่วมกับโครงสร้างข้อมูลจากอาร์เรย์ของ Structure ที่บรรจุข้อมูลของผู้เล่นซึ่งเป็นข้อมูลที่ใช้กำหนดผู้เล่นในแอพพลิเคชั่น ข้อมูลที่เป็นพอยเตอร์สามารถชี้ไปยัง โครงสร้าง(Structure) ,คลาส(class) ,ข้อมูลใน buffer ที่ใช้เก็บ ชื่อผู้เล่น อายุ อาวุธที่ใช้ปัจจุบัน เกราะ และอื่น ๆ

5.10.2 Destroying Player

เมื่อผู้เล่นหยุดการเชื่อมต่อหรือเกิดการ Destroy จะได้รับแมสเสจที่ชื่อ DPNMSG_DESTROY_PLAYER ซึ่งจะเก็บอยู่ในรูปแบบโครงสร้างข้อมูลโดยภายในโครงสร้างข้อมูลก็จะมีการเก็บหมายเลขของผู้เล่นที่ทำการ Destroy ไปและมีพอยเตอร์สำหรับชี้ไปยังผู้เล่นที่ถูก Destroy การหยุดการเชื่อมต่อของผู้เล่น

มีได้หลายสาเหตุไม่ว่าจะเป็นการสูญเสียการเชื่อมต่อ(Connect loss)หรือการที่เซสชั่นถูกตัดขาด(Session Terminate)จากเซิร์ฟเวอร์ เป็นต้น

5.11 Receiving Data

ในการรับข้อมูลของเกมจะอยู่ในรูปแบบของแอพพลิเคชั่นที่ทำการระบุแมสเสจ แต่จะถูกห่อหุ้มด้วยแมสเสจที่ชื่อ DPN_MSGID_RECEIVE ซึ่งแมสเสจนี้จะกำหนดเป็นโครงสร้างข้อมูลเพื่อทำการเก็บค่าต่างๆไม่ว่าจะเป็นหมายเลขของผู้เล่นที่ส่งข้อมูล หรือเก็บขนาดของข้อมูลที่รับเข้ามา โดยโครงสร้างข้อมูลจะกำหนดด้วยชื่อ DPNMSG_RECEIVE

5.12 Sending Server Message

เซิร์ฟเวอร์จะส่งข้อมูลไปยังไคลเอนต์โดยมีการใช้ฟังก์ชั่น SendTo ซึ่งจะทำการส่งข้อมูลไปยังผู้เล่นคนหนึ่งหรือผู้เล่นทั้งหมดในเกม โดยในฟังก์ชั่นจะต้องพิจารณาถึงความปลอดภัย และวิธีการส่งข้อมูล โดยในการส่งข้อมูลแต่ละครั้งจะต้องมีขนาดของข้อมูลและพอยเตอร์ที่ชี้ไปยังข้อมูล

5.13 Ending the Host Session

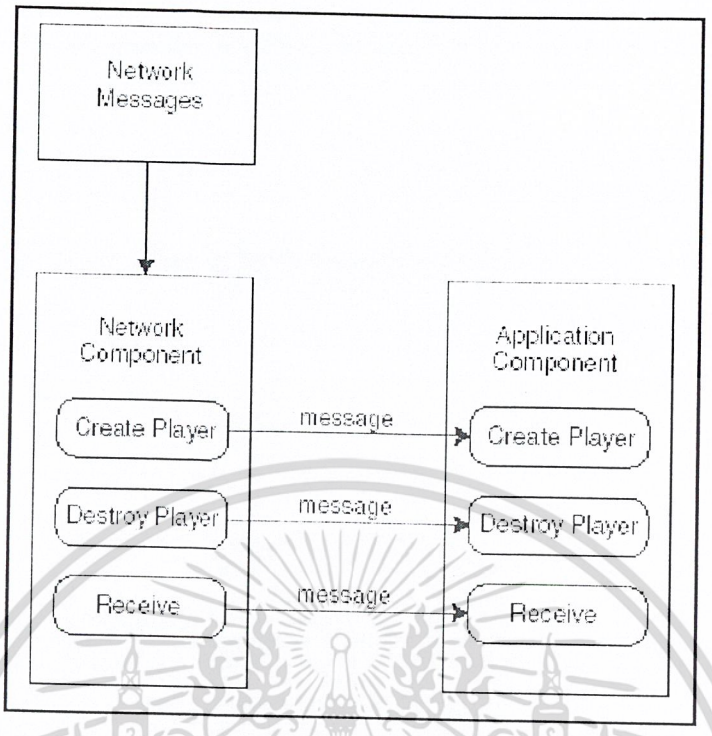
เมื่อเซิร์ฟเวอร์สิ้นสุดการทำงานจะมีการหยุดก็จะเกิดการหยุดเซสชั่น ซึ่งจะทำการของเซิร์ฟเวอร์หยุดการส่งของผู้เล่นทุกคนเพราะว่าฟังก์ชั่นมีการทำงานแบบซิงโครนัส(Synchronous) คือมันจะไม่รีเทิร์นค่าจนกว่าการส่งข้อมูลทั้งหมดจะสมบูรณ์และทุกการเชื่อมต่อจะปิดลง

5.14 Terminating the Client Session

เมื่อไคลเอนต์มีการตัดการเชื่อมต่อจากเซสชั่น DirectPlay จะมีขั้นตอนการทำงานที่ปิดการเชื่อมต่อได้อย่างถูกต้อง โดยในDirectPlay นั้นจะใช้ฟังก์ชั่น IDirectPlay8Client::Close จัดการ

5.15 แมสเสจภายในเกม

เซิร์ฟเวอร์จำเป็นที่จะทำการแปลงแมสเสจของไคเร็กเพลย์ ให้เป็นแมสเสจที่ใช้ภายในเกม ตามรูปที่ 5.9 เน็ตเวิร์กคอมโพเนนต์ต้องทำการส่งถ่ายแมสเสจที่รับเข้ามาลงไปในแอพพลิเคชั่นคอมโพเนนต์ โดยทำการสร้างฟังก์ชั่นสามฟังก์ชั่นให้เท่ากันกับฟังก์ชั่นของเน็ตเวิร์คคลาส แมสเสจที่รับเข้ามาภายในแอพพลิเคชั่นคอมโพเนนต์จะถูกนำเข้าไปเก็บไว้ในคิวของเซิร์ฟเวอร์



รูปที่ 5.9 เน็ตเวิร์คคอมโพเนนต์ส่งแมสเสจที่รับเข้ามาให้กับแอพพลิเคชั่นคอมโพเนนต์

แมสเสจต่างๆที่ใช้ในการส่งข้อมูลจากไคลเอนต์มายังเซิร์ฟเวอร์จะถูกนำไปเก็บไว้ภายในคิวของเซิร์ฟเวอร์ดังแสดงดังรูปที่ 5.10 ในการที่จะจัดการการเข้าและออกของแมสเสจภายในคิวนั้นจะอาศัยตัวแปรพอยน์เตอร์ 2 ตัวในการจัดการคือ m_MsgHead ใช้ในการชี้ตำแหน่งถัดไปของคิวที่จะแทรกแมสเสจถัดไปเพิ่มเข้าไปคิว และ m_MsgTail ใช้ในการชี้ตำแหน่งของแมสเสจที่จะถูกนำออกไปประมวลผลลำดับถัดไป

แมสเสจที่ใช้ภายในเกมแบ่งออกเป็น 2 ฝั่งคือ

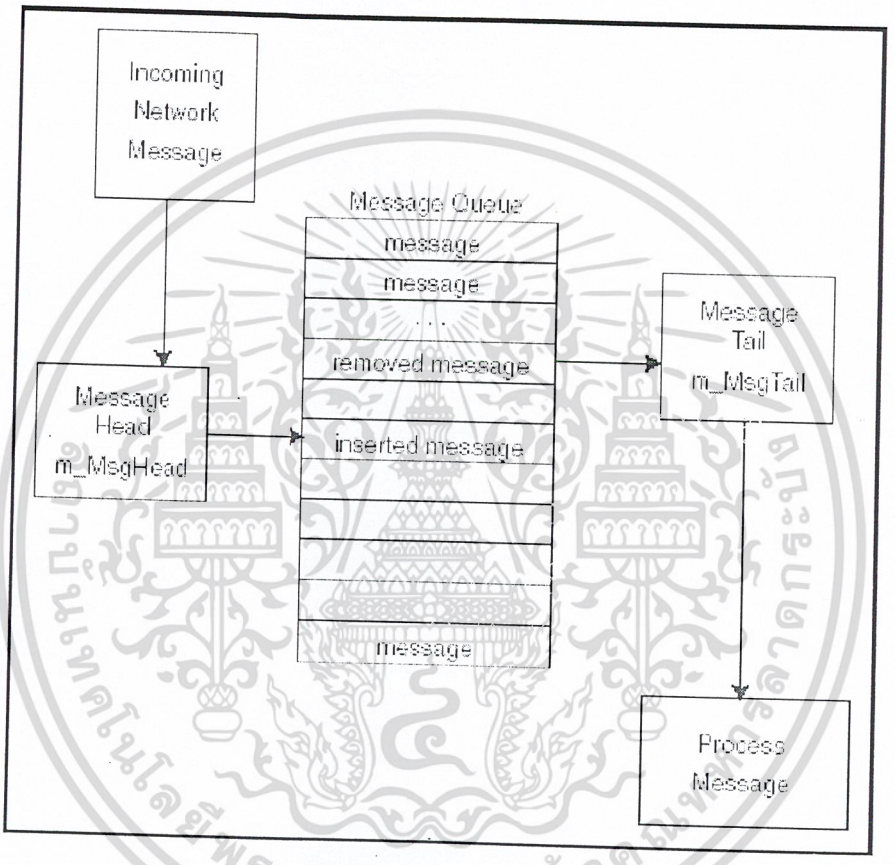
1 ฝั่งไคลเอนต์ส่งให้กับเซิร์ฟเวอร์

- แมสเสจในการขอหมายเลขไอดีให้กับผู้เล่นโดยที่ไอดีของผู้เล่นจะไม่ซ้ำกันแสดงดังรูปที่ 5.11
- แมสเสจขอข้อมูลของผู้เล่น
- แมสเสจเพิ่มผู้เล่นลงในเซิร์ฟเวอร์
- แมสเสจลบผู้ลบเล่นออกจากเซิร์ฟเวอร์
- แมสเสจการเปลี่ยนแปลงสถานะของผู้เล่น (รับมาจากอินพุตของฝั่งไคลเอนต์)

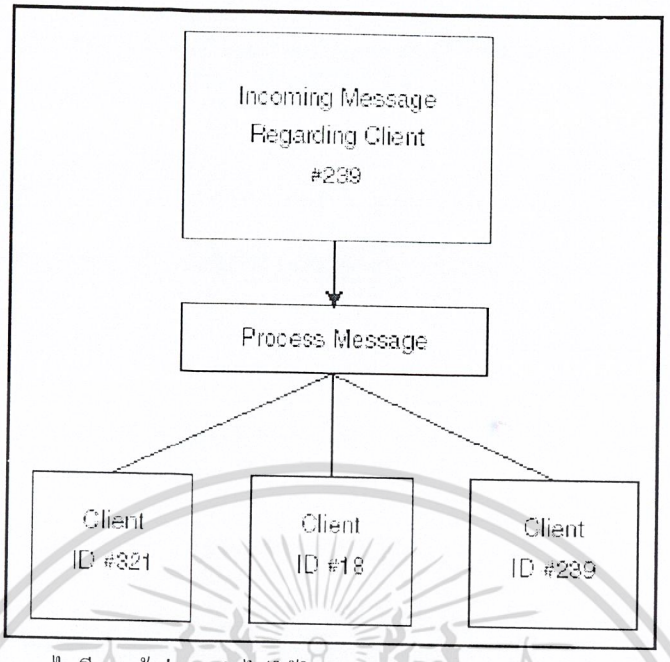
2 ฝั่งเซิร์ฟเวอร์ส่งให้กับไคลเอนต์

- แมสเสจในการกำหนดหมายเลขไอดีให้กับผู้เล่น
- แมสเสจเพิ่มข้อมูลของผู้เล่นให้กับไคลเอนต์
- แมสเสจส่งข้อมูลของผู้เล่นให้กับไคลเอนต์
- แมสเสจการเปลี่ยนแปลงสถานะของผู้เล่นที่อัพเดตมาจากเซิร์ฟเวอร์

- แมสเสจการเปลี่ยนสถานะของตัวมอนสเตอร์และตัวละครที่ผู้เล่นไม่ได้บังคับ (NPC) โดยอัพเดตมาจากเซิร์ฟเวอร์
- แมสเสจการเปลี่ยนแผนที่
- แมสเสจกำหนดตัวเลขเพื่อแสดงในฝั่งไคลเอนต์ (ใช้ในกรณีฟันมอนสเตอร์)
- แมสเสจลดพลังชีวิตของผู้เล่น
- แมสเสจเพิ่มพลังชีวิตให้กับผู้เล่น (ในกรณีที่ตัวละครของผู้เล่นตายแล้วเกิดใหม่)

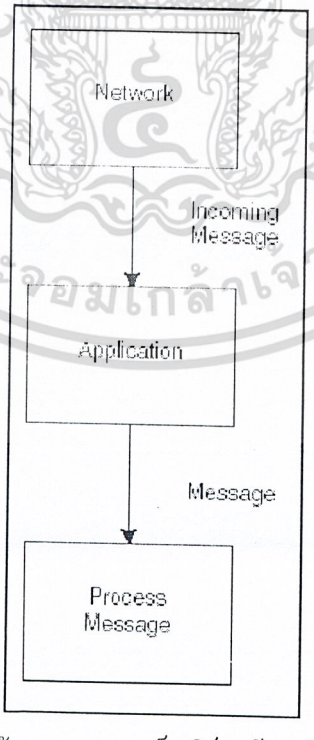


รูปที่ 5.10 คิวภายในเซิร์ฟเวอร์



รูปที่ 5.11 ไอดีของผู้เล่นจะนำไปใช้ในการแยกแมสเสจว่าเป็นของผู้เล่นคนใด

การจัดการแมสเสจทางด้านฝั่งไคลเอนต์ไม่จำเป็นต้องมีคิวเพื่อคอยจัดการแมสเสจไคลเอนต์เพียงเปลี่ยนแมสเสจที่รับมาจากเน็ตเวิร์คคอมโพเนนต์ให้เป็นแมสเสจที่เกี่ยวกับเกมแล้วสามารถนำไปประมวลผลได้ทันทีดังแสดงในรูปที่ 5.12



รูปที่ 5.12 ไคลเอนต์รับแมสเสจจากเน็ตเวิร์คแล้วนำไปประมวลผลทันที

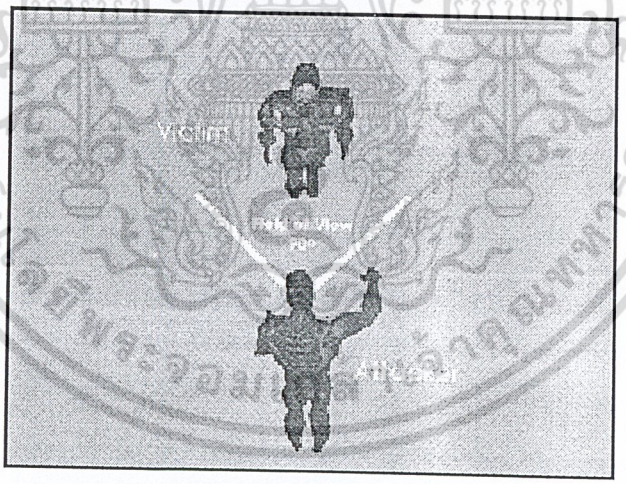
5.16 สถานะของผู้เล่น

สถานะของตัวละครภายในเกมแบ่งออกเป็นสถานะต่างๆดังนี้คือ

- สถานะเดิน (Move) ใช้ในการควบคุมการเดินของผู้เล่นและมอนสเตอร์
- สถานะว่าง (Idle) ใช้ในการควบคุมให้ผู้เล่นและมอนสเตอร์หยุดการเคลื่อนไหวใดๆ
- สถานะโจมตี (Swing) ใช้ในการควบคุมการโจมตีของผู้เล่นและมอนสเตอร์
- สถานะบาดเจ็บ (Hurt) ใช้ในการกำหนดให้ผู้เล่นอยู่ในสภาพบาดเจ็บขณะโดนโจมตี
- สถานะตาย (Die) ใช้ในการกำหนดให้ผู้เล่นหรือมอนสเตอร์ตายหากค่าพลังชีวิตหมด

ในการควบคุมต่างๆจะต้องอาศัยค่าทิศทาง (Direction) ของตัวละครมาทำการคำนวณด้วยทุกครั้ง เช่นในการเดินเพื่อให้ทราบว่าตัวละครนั้นเดินไปในทิศทางใดรวมไปถึงการโจมตีด้วย โดยที่ค่าทิศทางของตัวละครนั้นจะต้องมีความสอดคล้องกันทั้งบนฝั่งเซิร์ฟเวอร์และไคลเอนต์ เนื่องจากสถานะต่างๆของตัวละครภายในเกมต้องมีความสอดคล้องกันนี้เองเซิร์ฟเวอร์จึงจำเป็นที่จะต้องอัพเดตข้อมูลสถานะให้กับไคลเอนต์ทุกตัวเป็นระยะๆอยู่ตลอดเวลาเฉลี่ย 100 มิลลิวินาทีต่อหนึ่งครั้ง

นอกจากจะต้องทำการอัพเดตข้อมูลสถานะตัวละครจากเซิร์ฟเวอร์ให้กับไคลเอนต์ทุกตัวทุกๆ 100 มิลลิวินาทีแล้ว ทั้งไคลเอนต์และเซิร์ฟเวอร์จะต้องทำการอัพเดตสถานะของตัวละครภายในตัวเองทุกๆ 33 มิลลิวินาทีอีกด้วย



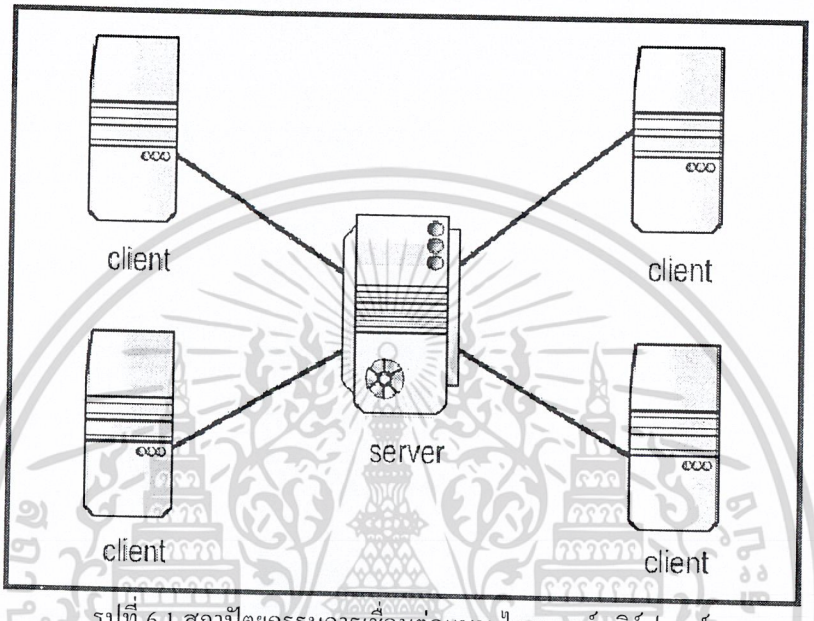
รูปที่ 5.13 การโจมตีต้องอาศัยค่าทิศทางมาทำการคำนวณหาค่ามุมการมองเห็น

สถานะต่างๆที่ใช้ภายในเกมยังนำไปใช้ในอ้างอิงการแสดงภาพเคลื่อนไหวประกอบในฝั่งของไคลเอนต์อีกด้วย เช่นสถานะโจมตีฝั่งไคลเอนต์จะแสดงภาพเคลื่อนไหวของการฟันดาบ สถานะเดินไคลเอนต์จะแสดงภาพเคลื่อนไหวของการเดินเป็นต้น โดยที่ช่วงระยะเวลาของการแสดงภาพเคลื่อนไหวเหล่านี้เซิร์ฟเวอร์จะเป็นตัวกำหนดไคลเอนต์เพียงทำหน้าที่แสดงผลตามเท่านั้น

บทที่6
การกระจายการประมวลผล

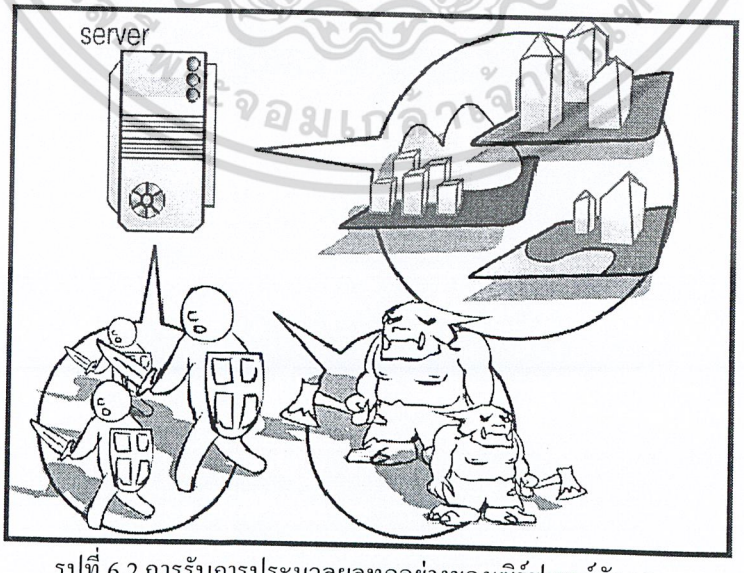
6.1 การกระจายการประมวลผล

จากสถาปัตยกรรมการเชื่อมต่อแบบ ไคลเอนต์-เซิร์ฟเวอร์ (Client-Server)



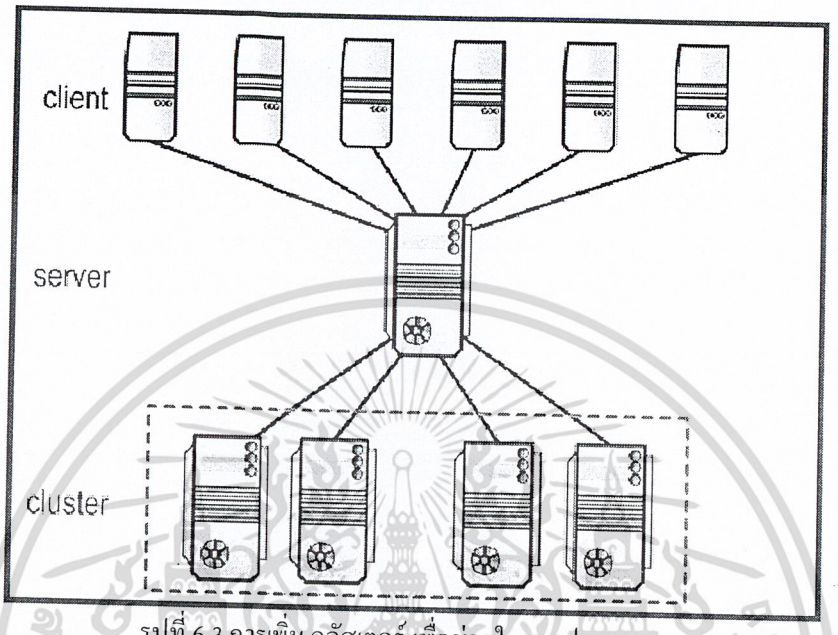
รูปที่ 6.1 สถาปัตยกรรมการเชื่อมต่อแบบ ไคลเอนต์-เซิร์ฟเวอร์

เซิร์ฟเวอร์ตัวกลางต้องรับภาระการประมวลผลทุกอย่าง ไม่ว่าจะเป็นฉากแผนที่ ผู้เล่นที่เชื่อมต่อเข้ามา รวมถึง NPC (non-player character) ไม่ว่าจะเป็น monster หรือตัวดำเนินเรื่องของเกม



รูปที่ 6.2 การรับการประมวลผลทุกอย่างของเซิร์ฟเวอร์ตัวกลาง

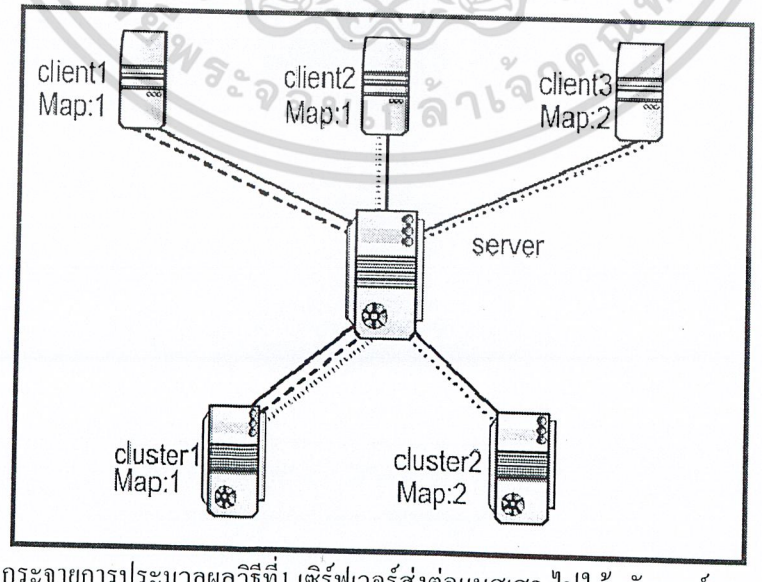
จึงได้มีการเพิ่ม คลัสเตอร์(Cluster) เข้าไปเพื่อช่วยในการประมวลผล โดยเซิร์ฟเวอร์กลางจะโอนภาระการประมวลผลไปให้กับคลัสเตอร์ ที่ต่อเชื่อมอยู่ในรูปแบบของแมสเสจ โดยเซิร์ฟเวอร์ตัวกลางจะส่งผ่าน Massage ที่รับมาจากผู้เล่นฝั่ง ไคลเอนต์ ให้กับ คลัสเตอร์



รูปที่ 6.3 การเพิ่ม คลัสเตอร์ เพื่อช่วยในการประมวลผล

แนวทางในการกระจายการประมวลผล

วิธีที่1. แบ่งแผนที่แต่ละแผนที่ของเกมไว้ใน คลัสเตอร์แต่ละตัว วิธีนี้จะแบ่งแยกการประมวลผลได้ชัดเจนโดย แมสเสจ จาก ไคลเอนต์ ที่กำลังเล่นเกมอยู่ในแผนที่เดียวกันจะถูกส่งไปที่ คลัสเตอร์ ตัวเดียวกันหมด

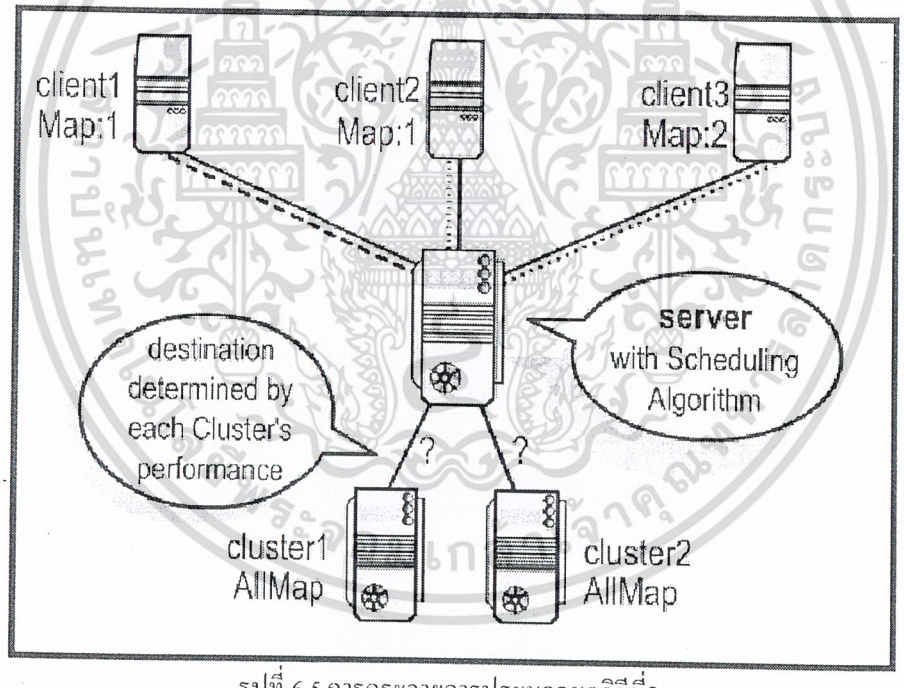


รูปที่ 6.4 การกระจายการประมวลผลวิธีที่1 เซิร์ฟเวอร์ส่งต่อแมสเสจ ไปให้คลัสเตอร์ตามแผนที่ (Map)

จากรูปที่ 6.4 ไคลเอนต์ ที่เป็นผู้เล่นเครื่องที่ 1 และ 2 กำลังเล่นเกมอยู่ในแผนที่ที่หนึ่ง ส่วน ไคลเอนต์ คนที่สามกำลังเล่นอยู่ในแผนที่ที่สอง ส่วน คลัสเตอร์เครื่องที่ 1 และ 2 รับผิดชอบการประมวลผลกับแมสเสจ ที่จะมาจากผู้เล่นที่เล่นเกมอยู่ในแผนที่ที่ หนึ่งและสอง ตามลำดับ ดังนั้น เซิร์ฟเวอร์ตัวกลางจึงส่งต่อแมสเสจ จาก ไคลเอนต์ 1 และ ไคลเอนต์ 2 มาให้กับ คลัสเตอร์ 1 และส่งต่อแมสเสจ จาก ไคลเอนต์ 3 ไปประมวลผลที่ คลัสเตอร์ 2

ในกรณีนี้ คลัสเตอร์ แต่ละตัวจะเก็บข้อมูลแผนที่ที่ตัวเองรับผิดชอบ โดยไม่จำเป็นต้องรับรู้ข้อมูลของแผนที่อื่นๆเนื่องจากไม่ได้นำมาวิเคราะห์ประมวลผล เช่น การตรวจสอบการชนในแผนที่ใดๆก็ไม่จำเป็นต้องตรวจสอบว่าจะชนกับสิ่งกีดขวางที่อยู่ในแผนที่ที่เราไม่ได้เล่นหรือผู้เล่นคนอื่นอยู่คนละแผนที่กันเป็นต้น

วิธีที่2. การแบ่งให้ทุก คลัสเตอร์ มีแผนที่ทุกแผนที่ วิธีนี้ คลัสเตอร์ทุกตัวจำเป็นต้องรู้ข้อมูลทุกอย่าง และทุกแผนที่ของเกม การกระจายการประมวลผลจะเฉลี่ยแมสเสจ ไปให้ทุก คลัสเตอร์ โดยใช้ Scheduling Algorithms ต่างๆ

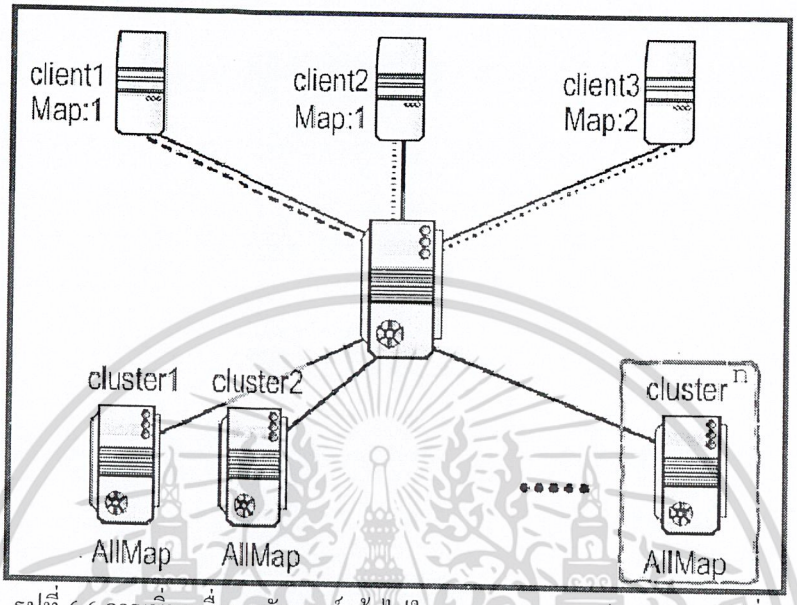


รูปที่ 6.5 การกระจายการประมวลผลวิธีที่2

จากรูปที่ 6.5 เซิร์ฟเวอร์ตัวกลางต้องมี Scheduling Algorithms เพื่อที่จะรู้ว่าจะส่งต่อแมสเสจ จากผู้เล่นไปยัง คลัสเตอร์ เครื่องใด Scheduling Algorithms มีหลายแบบ เช่น Weighted Round-Robin Scheduling, Weighted Least-Connections เป็นต้น

การเก็บข้อมูลของ คลัสเตอร์ ทุกตัวต้องเก็บข้อมูลเหมือนกันหมด เพราะว่ามีทุกแผนที่ที่ต้องคอยจัดการอยู่ในตัวนั่นเอง วิธีนี้จะเสีย Overhead มากในการ อัพเดต สถานะต่างๆของตัวละครในเกมให้กับ

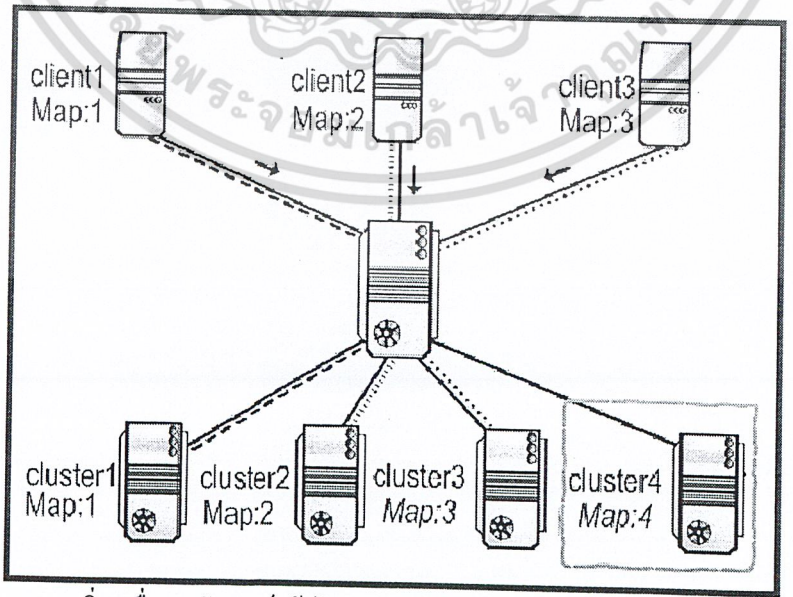
คลัสเตอร์ ทุกตัวรู้เหมือนกัน ในการเพิ่มเครื่องเพื่อให้ทำงานให้เร็วขึ้นสามารถเพิ่มเข้าไปได้ โดยผู้ติดตั้งต้องบอกให้เซิร์ฟเวอร์ตัวกลางรู้ถึง ประสิทธิภาพของ คลัสเตอร์ แต่ละเครื่องในกรณีที่ใช้ Scheduling Algorithms แบบที่มีการพิจารณา Weight



รูปที่ 6.6 การเพิ่มเครื่อง คลัสเตอร์ เข้าไปในการกระจายการประมวลผลวิธีที่2

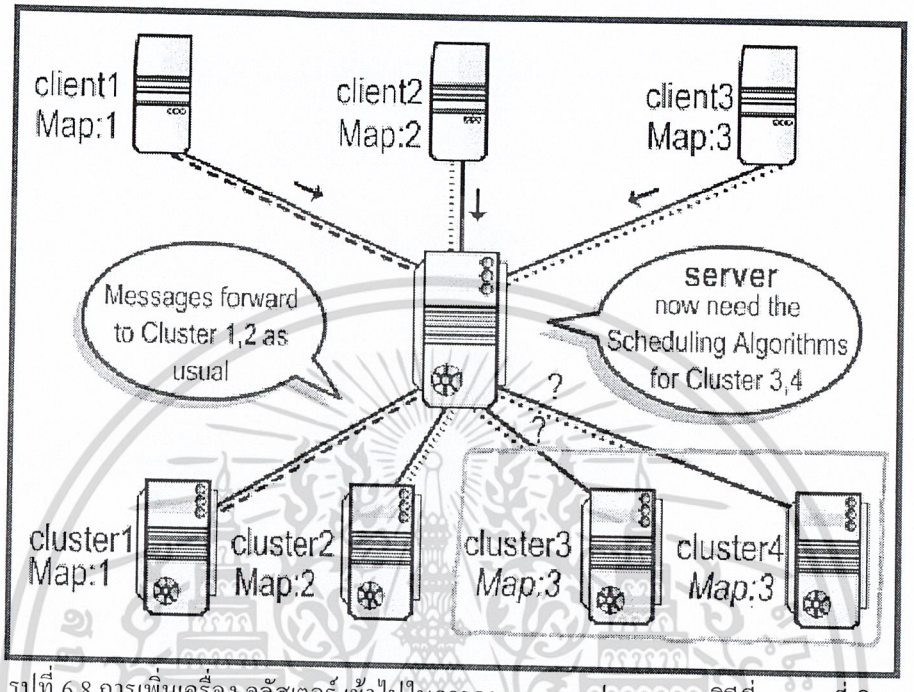
การเพิ่มเครื่องเข้าไปช่วยประมวลผลสามารถทำได้ง่ายกว่าวิธีที่1 แต่ทุกเครื่องจำเป็นต้องมีข้อมูลแผนที่ให้ครบ และการเสีย Overhead ในการ อัพเดต ข้อมูลก็ยิ่งมากตามไปด้วย

แต่ในการเพิ่มเครื่อง คลัสเตอร์ เข้าไปในการกระจายแบบวิธีที่1 หากเป็นการเพิ่มในแง่ของการเพิ่มแผนที่ใหม่ก็สามารถเพิ่มเข้าไปได้เลย



รูปที่ 6.7 การเพิ่มเครื่อง คลัสเตอร์ เข้าไปในการกระจายการประมวลผลวิธีที่1 (แผนที่ใหม่)

จากรูปที่ 6.7 เป็นการเพิ่มแผนที่ที่4เข้าไปในเกมตามโครงสร้างของวิธีที่1 แต่หากเป็นการเพิ่มแผนที่เดียวกันเป็นแผนที่แผนที่ไปเซิร์ฟเวอร์ตัวหลักก็ต้องใช้ Scheduling Algorithms เหมือนกับว่าเอาวิธีการกระจายวิธีที่2 มาใส่ไว้ในวิธีที่1 อีกที



รูปที่ 6.8 การเพิ่มเครื่อง คลัสเตอร์ เข้าไปในการกระจายการประมวลผลวิธีที่1 (แผนที่เดิม)

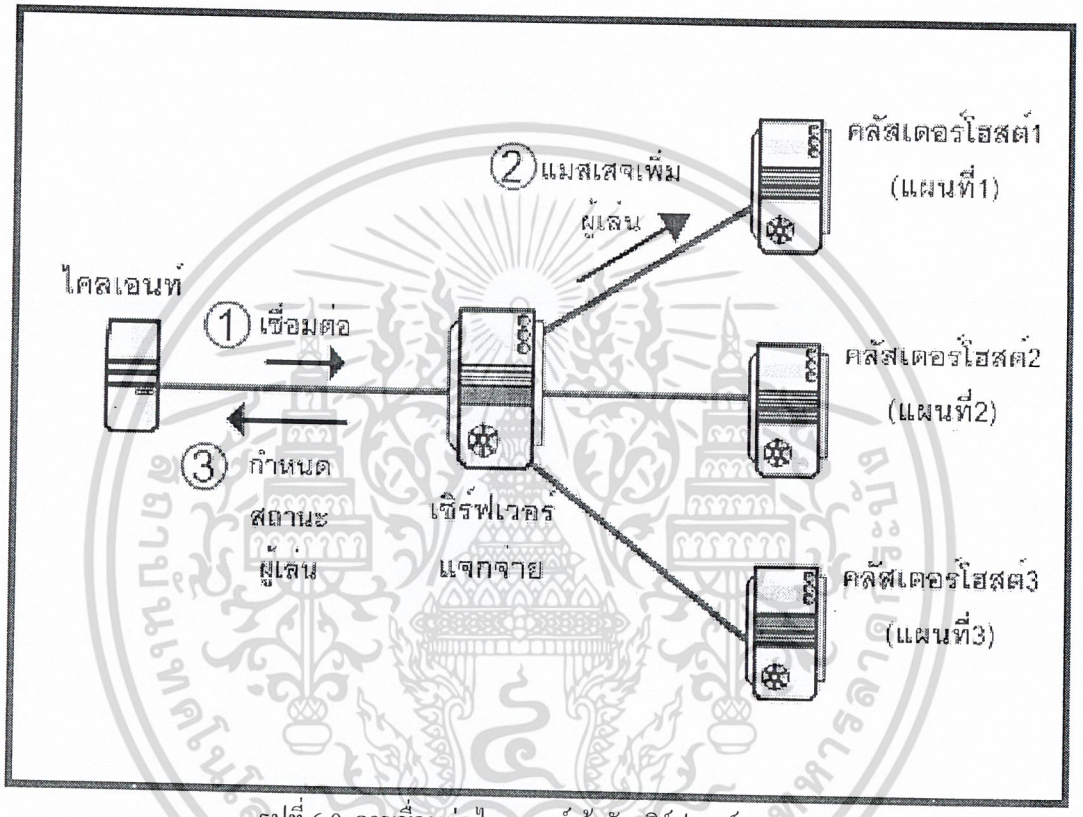
จากรูปที่ 6.8 เป็นการเพิ่ม คลัสเตอร์ อีกตัวเข้าไปแต่เป็นแผนที่ที่3 การกระจาย แมสเสจ มาที่ คลัสเตอร์ 1 และ 2 ยังคงเป็นแบบเดิมเพราะมี1แผนที่ ต่อ 1 คลัสเตอร์ ส่วนแผนที่ที่3 เมื่อตอนนี้มีอยู่ 2 คลัสเตอร์ เซิร์ฟเวอร์ตัวกลางจะส่งต่อ แมสเสจ ให้ใคร คำตอบคือต้องมี Scheduling Algorithms ย่อยๆขึ้นมาใช้โดยเฉพาะกับ คลัสเตอร์ ที่เป็นแผนที่เดียวกัน วิธีนี้อาจดูเหมือนว่าเป็นการใช้วิธีที่สองแต่ การเสีย Overhead จะน้อยกว่า เนื่องจากเป็นแผนที่เดียวกันหมดและไม่ต้อง ส่ง อัพเดต ให้กับทุกเครื่องนอกจากเครื่องที่เป็นแผนที่เดียวกันนั่นเอง

6.2 การสื่อสารของระบบกระจายการประมวลผลแบบแบ่งแผนที่

เนื่องจากการสื่อสารของเกมระหว่างไคลเอนต์กับคลัสเตอร์โฮสต์ที่ทำหน้าที่ประมวลผลเกมไม่ได้ทำการเชื่อมต่อกันโดยตรงซึ่งการสื่อสารจะผ่านตัวเซิร์ฟเวอร์แจกจ่ายดังนั้นแมสเสจของการเกมจึงต้องถูกจัดการ โดยตัวเซิร์ฟเวอร์แจกจ่ายเป็นหลัก หน้าที่ของเซิร์ฟเวอร์แจกจ่ายจะทำหน้าที่เพียงส่งผ่านแมสเสจของเกมระหว่างไคลเอนต์กับคลัสเตอร์โฮสต์และเก็บข้อมูลสถานะของผู้เล่นบางส่วนเท่านั้น การจัดลำดับการสื่อสารของเกมแบบกระจายการประมวลผลมีดังนี้คือ

6.2.1 การเชื่อมต่อไคลเอนต์เข้ากับเซิร์ฟเวอร์

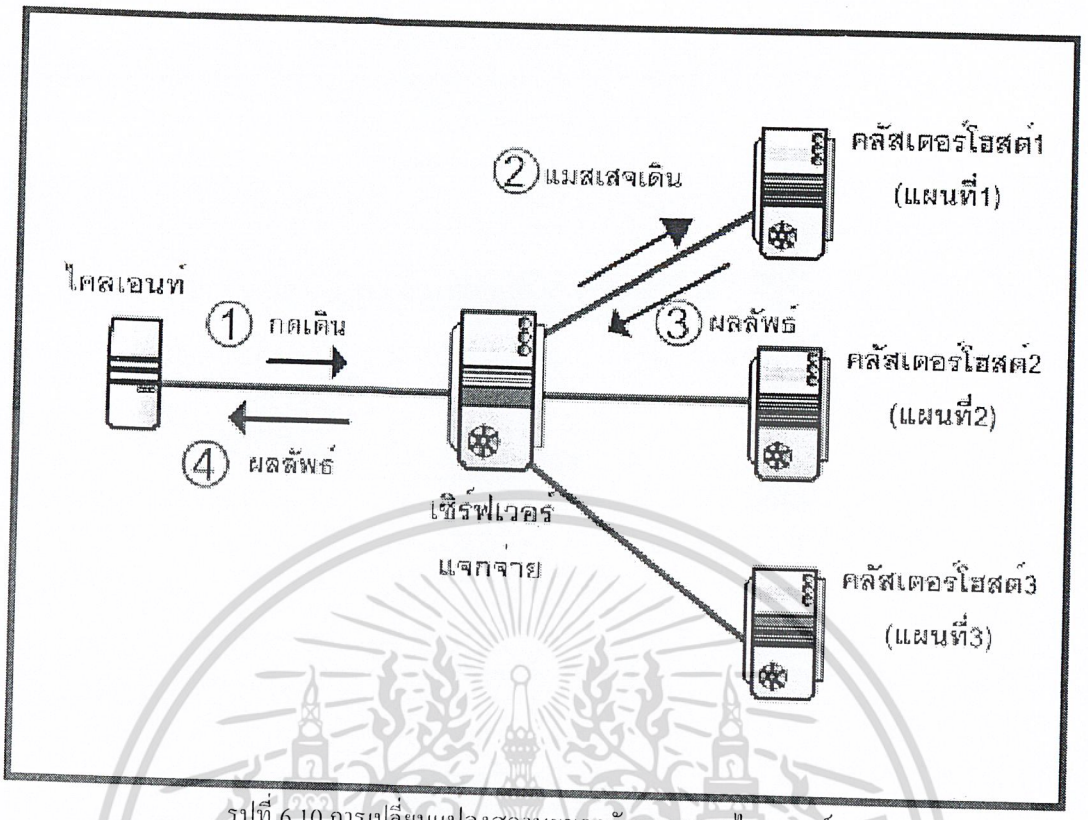
ไคลเอนต์ทำการเชื่อมต่อกับเซิร์ฟเวอร์กับเซิร์ฟเวอร์แจกจ่าย เมื่อเซิร์ฟเวอร์แจกจ่ายได้รับจะทำการกำหนดไอดีให้กับผู้เล่นและกำหนดสถานะของตัวละครแล้วก็จะทำการส่งแมสเสจสร้างผู้เล่นให้กับคลัสเตอร์โฮสต์ตามแผนที่ผู้เล่นอยู่ โดยปกติแล้วผู้เล่นจะเริ่มต้นที่แผนที่ 1 ดังนั้นจึงต้องส่งแมสเสจไปยังคลัสเตอร์โฮสต์ตัวที่ 1 จากนั้นเซิร์ฟเวอร์แจกจ่ายจึงส่งแมสเสจกำหนดสถานะตัวละครกลับมาให้กับตัวไคลเอนต์และผู้เล่นคนอื่นๆที่อยู่แผนที่เดียวกัน



รูปที่ 6.9 การเชื่อมต่อไคลเอนต์เข้ากับเซิร์ฟเวอร์

6.2.2 การเปลี่ยนแปลงสถานะของตัวละคร

เมื่อไคลเอนต์ทำการบังคับควบคุมตัวละครผ่านทางอุปกรณ์อินพุตไคลเอนต์จะส่งแมสเสจการเปลี่ยนสถานะไปยังเซิร์ฟเวอร์แจกจ่าย ซึ่งแมสเสจจะถูกเก็บไว้ภายในคิวและเมื่อแมสเสจนั้นๆได้รับการประมวลผลตัวเซิร์ฟเวอร์แจกจ่ายจะทำการตรวจสอบดูว่าผู้เล่นฝั่งไคลเอนต์นั้นกำลังอยู่แผนที่ใด จากนั้นก็จะส่งแมสเสจไปให้กับคลัสเตอร์โฮสต์ของแผนที่นั้นๆ เมื่อคลัสเตอร์โฮสต์ทำการประมวลผลเสร็จแล้วก็จะส่งแมสเสจการอัพเดตสถานะมาให้กับเซิร์ฟเวอร์แจกจ่ายและเซิร์ฟเวอร์แจกจ่ายก็จะทำการตรวจสอบไอดีว่าแมสเสจนั้นเป็นของผู้เล่นคนไหนเพื่อส่งต่อกลับไปให้กับผู้เล่นนั้นๆและผู้เล่นคนอื่นๆที่อยู่ในแผนที่เดียวกัน



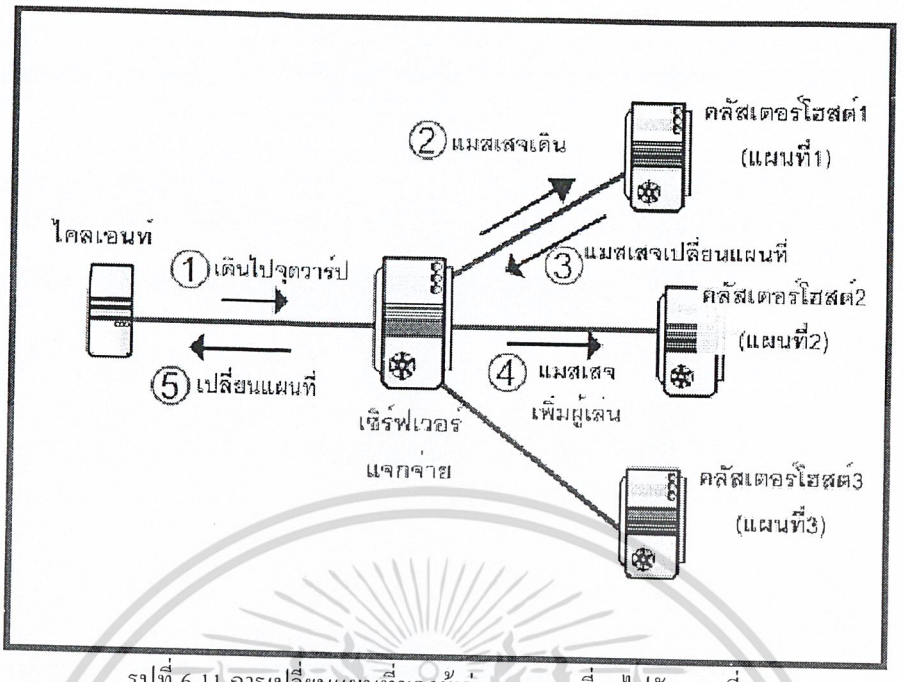
รูปที่ 6.10 การเปลี่ยนแปลงสถานะของตัวละครจากไคลเอนต์

6.2.3 การอัพเดตสถานะจากคลัสเตอร์โฮสต์

คลัสเตอร์โฮสต์แต่ละตัวจะต้องทำการอัพเดตสถานะของตัวละครผู้เล่นและมอนสเตอร์ให้กับไคลเอนต์ทุกๆ 100 มิลลิวินาทีเพื่อทั้งฝั่งเซิร์ฟเวอร์และไคลเอนต์มีสถานะของตัวละครที่ตรงกัน แต่ว่าคลัสเตอร์โฮสต์ไม่ได้ทำการเชื่อมต่อกับไคลเอนต์โดยตรงดังนั้นจึงต้องทำการส่งแมสเสจอัพเดตสถานะผ่านไปยังเซิร์ฟเวอร์แจกจ่ายแล้วส่งต่อไปให้กับไคลเอนต์อีกทีหนึ่ง

6.2.4 การเปลี่ยนแผนที่ภายในเกม

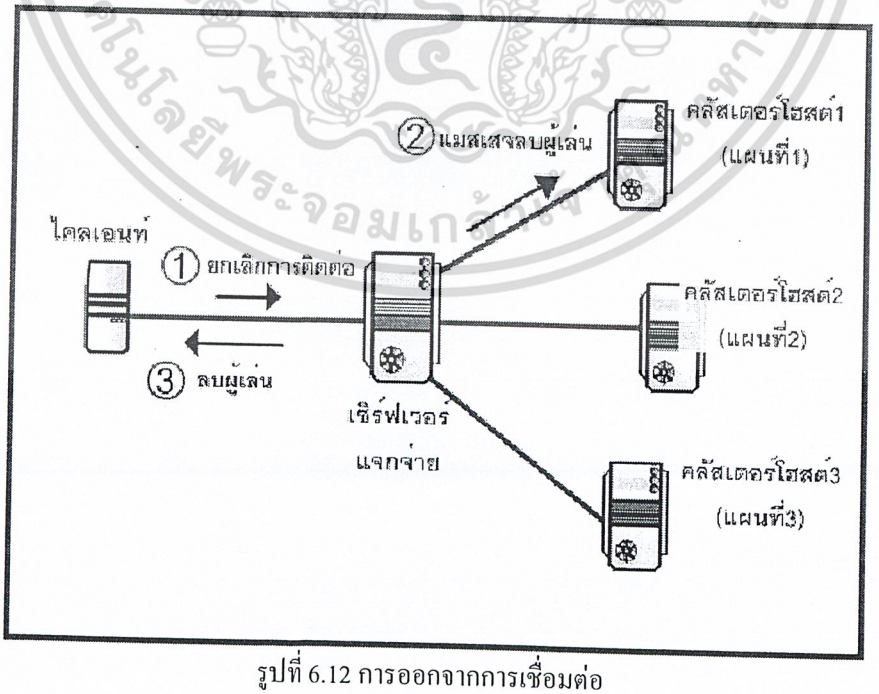
ผู้เล่นสามารถที่จะเคลื่อนที่ข้ามไปมาระหว่างแผนที่ได้ดังนั้นคลัสเตอร์โฮสต์จะต้องสามารถทำงานประสานงานร่วมกันได้ การทำงานร่วมกันนี้จะอาศัยตัวเซิร์ฟเวอร์แจกจ่ายเป็นตัวสับเปลี่ยนการเชื่อมต่อระหว่างผู้เล่นไปยังคลัสเตอร์โฮสต์ตัวอื่นๆ ดังรูปที่ 6.11 สมมุติว่าถ้าผู้เล่นบังคับตัวละครให้เดินทางไปยังจุดวาร์บเพื่อย้ายตัวละครจากแผนที่ 1 ไปยังแผนที่ 2 คลัสเตอร์โฮสต์ตัวที่ 1 (จัดการแผนที่ 1) จะทำการส่งแมสเสจการเปลี่ยนแผนที่ของตัวละครไปให้กับเซิร์ฟเวอร์แจกจ่ายแล้วจากนั้นจะลบข้อมูลสถานะตัวละครของผู้เล่นนั้นๆออกจากคลัสเตอร์โฮสต์ เมื่อเซิร์ฟเวอร์ได้รับแมสเสจการเปลี่ยนแผนที่ก็จะส่งแมสเสจการเปลี่ยนแผนที่ไปให้กับไคลเอนต์ของผู้เล่นและส่งแมสเสจสร้างการผู้เล่นไปยังคลัสเตอร์โฮสต์ตัวที่ 2 (จัดการแผนที่ 2) หลังจากนั้นการประมวลผลของผู้เล่นจะถูกส่งไปประมวลผลที่คลัสเตอร์โฮสต์ตัวที่ 2 ต่อไป



รูปที่ 6.11 การเปลี่ยนแผนที่ของผู้เล่นจากแผนที่ 1 ไปยังแผนที่ 2

6.2.5 การออกจากการเชื่อมต่อของไคลเอนต์

เมื่อไคลเอนต์ออกจากเกมจะส่งแมสเสจยกเลิกการเชื่อมต่อไปยังเซิร์ฟเวอร์แจกจ่าย แล้วเซิร์ฟเวอร์แจกจ่ายจะส่งแมสเสจการลบข้อมูลสถานะของผู้เล่นนั้นๆไปให้กับคลัสเตอร์โฮสต์ที่ประมวลผลผู้เล่นนั้น และจะทำการลบข้อมูลของผู้เล่นที่เครื่องเซิร์ฟเวอร์แจกจ่ายเองออกด้วยเช่นกันรวมทั้งส่งแมสเสจให้กับผู้เล่นคนอื่นๆที่อยู่แผนที่เดียวกันให้ทราบด้วย



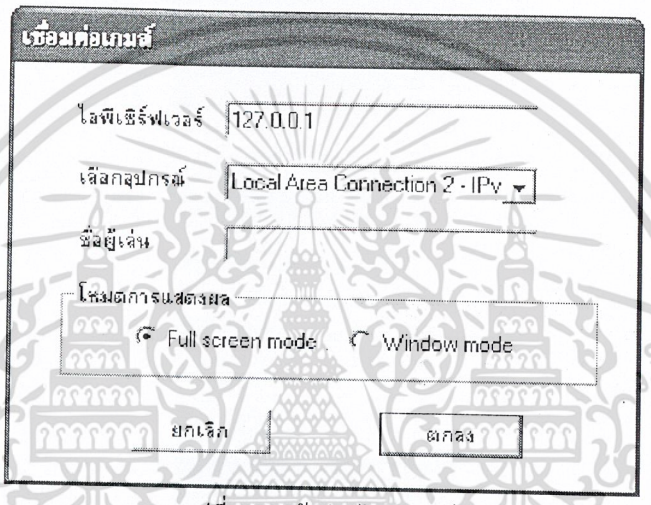
รูปที่ 6.12 การออกจากการเชื่อมต่อ

บทที่ 7
การใช้งานและการทดสอบ

ตัวโปรแกรมแบ่งการทำงานออกเป็นสองส่วนคือ ส่วนที่เป็นเซิร์ฟเวอร์และไคลเอนต์ โดยทั้งสองส่วนจะทำหน้าที่แตกต่างแยกออกจากกัน มีรายละเอียดในการใช้งานเพื่อทำการทดลองแต่ละส่วนดังนี้

7.1 การใช้งานไคลเอนต์

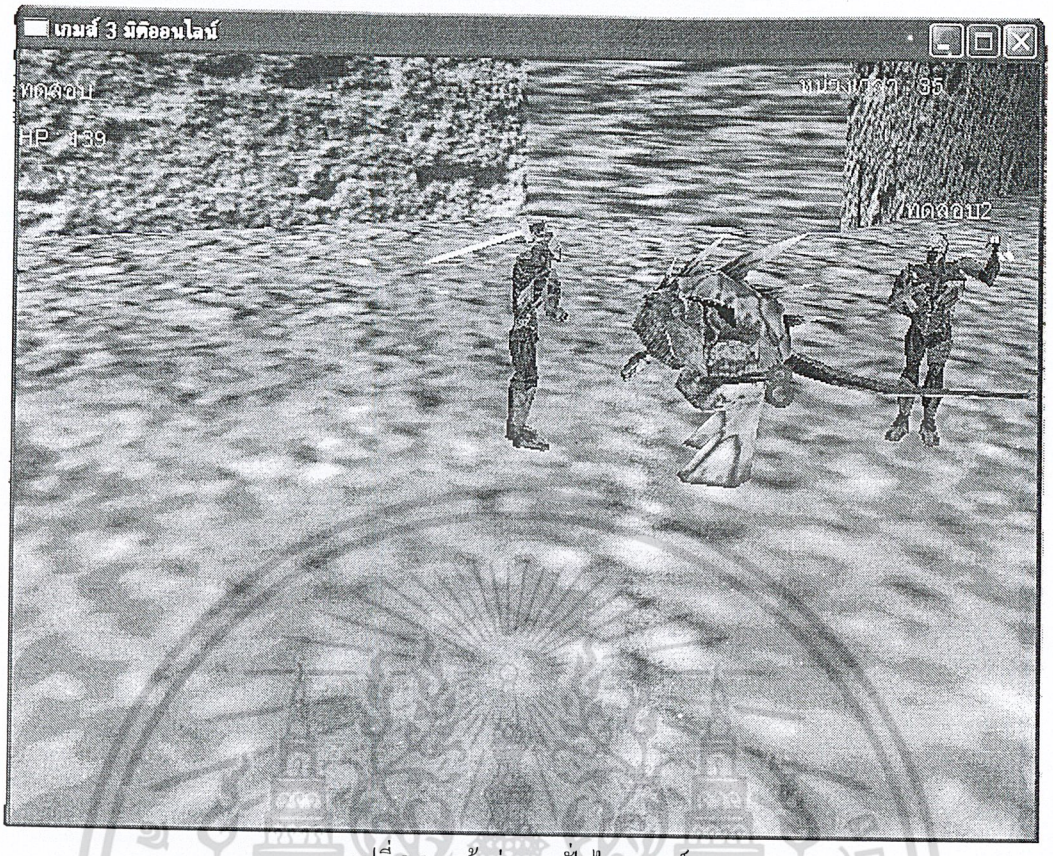
ส่วนของไคลเอนต์ทำหน้าที่ในการเชื่อมต่อเกมเข้ากับส่วนของเซิร์ฟเวอร์มีการใช้งานดังนี้



รูปที่ 7.1 หน้าต่างไคลเอนต์

- ไอพีเซิร์ฟเวอร์ ใช้กรอกไอพีแอดเดรสของเครื่องเซิร์ฟเวอร์ที่ต้องการจะติดต่อ
- เลือกอุปกรณ์ ใช้ในการเลือกอุปกรณ์การ์ดเครือข่ายที่ต้องการใช้ในการติดต่อสื่อสาร
- ชื่อผู้เล่น ใช้ในการป้อนชื่อผู้เล่น
- โหมดการแสดงผล สามารถเลือกโหมดการแสดงผลได้ 2 โหมดคือ โหมดเต็มจอภาพ (Full screen mode) และโหมดวินโดวส์ (Window mode)

หลังจากการเชื่อมต่อแล้วจะปรากฏหน้าต่างเกมขึ้นดังรูปที่ 7.2 ภายในหน้าต่างจะแสดงกราฟิกภายในเกม รวมไปถึงมีชื่อผู้เล่น ค่าพลังชีวิตของผู้เล่น (HP) ค่าหน่วงเวลาหรือค่าเวลาตอบสนองผู้เล่น (Response time)

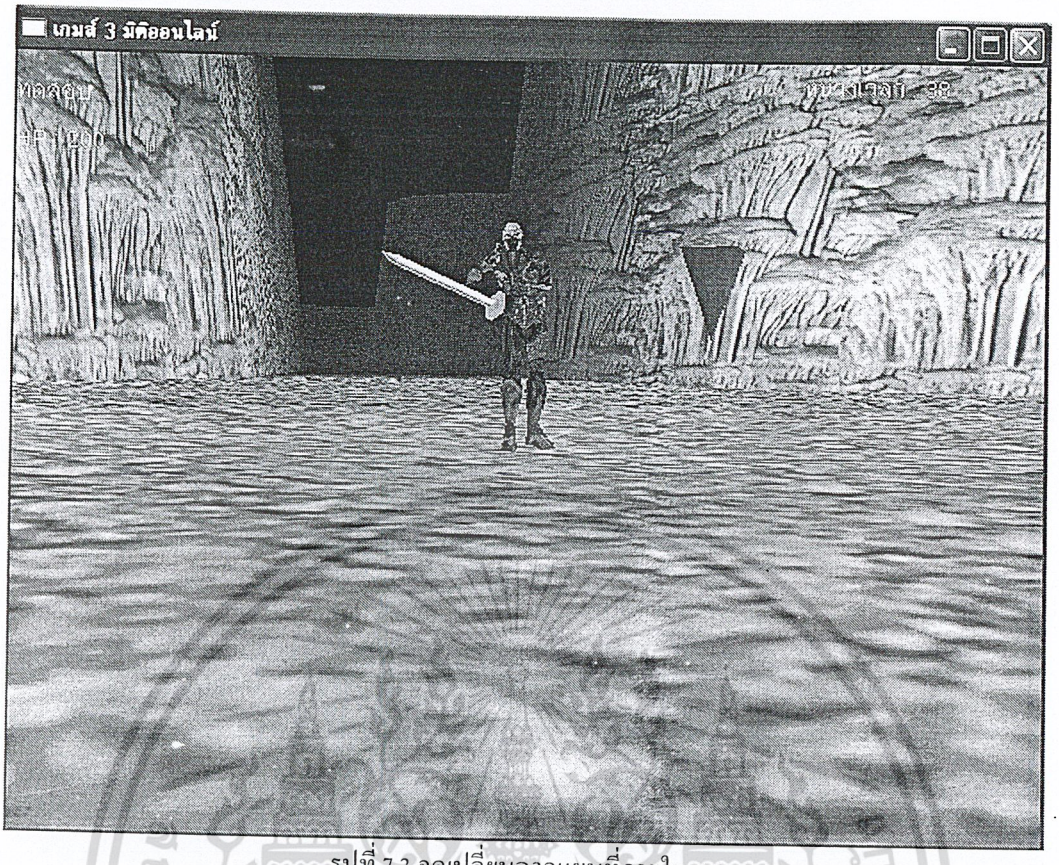


รูปที่ 7.2 หน้าต่างเกมฝั่งไคลเอนต์

ปุ่มควบคุมเกมมีดังนี้คือ

- เดินหน้า กดปุ่มลูกศรบน
- เดินซ้าย กดปุ่มลูกศรซ้าย
- เดินขวา กดปุ่มลูกศรขวา
- เดินลง กดปุ่มลูกศรล่าง
- ปรับมุมกล้อง คลิกเมาส์ขวาค้างไว้แล้วเลื่อนเมาส์
- ฟันดาบ คลิกเมาส์ซ้าย หรือกดปุ่ม Space bar

การเปลี่ยนแผนที่ภายในเกมให้บังคับตัวละครของผู้เล่นไปในจุดที่มีสัญลักษณ์สามเหลี่ยมสีแดงที่กำลังหมุนอยู่ดังรูปที่ 7.3 ฉากแผนที่ภายในเกมจะมีทั้งหมด 3 ฉากด้วยกัน



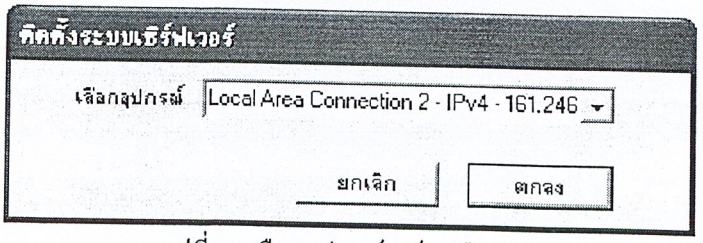
รูปที่ 7.3 จุดเปลี่ยนฉากแผนที่ภายในเกม

7.2 การใช้งานเซิร์ฟเวอร์

ส่วนของเซิร์ฟเวอร์จะทำหน้าที่จัดการการประมวลผลของเกมทั้งหมด ซึ่งส่วนของเซิร์ฟเวอร์ที่จะนำมาทดลองนั้นมีอยู่ 2 แบบด้วยกันคือ แบบไม่มีการกระจายการประมวลผลซึ่งจะใช้เซิร์ฟเวอร์ตัวเดียวประมวลผลเกมทุกๆฉาก และส่วนของเซิร์ฟเวอร์ที่สามารถกระจายการประมวลผลโดยอาศัยตัว คลัสเตอร์โฮสต์เป็นตัวประมวลผลเกมในแต่ละฉากและตัวเซิร์ฟเวอร์กลางที่ใช้ในการจ่ายแมสเสจของเกมให้กับคลัสเตอร์โฮสต์แต่ละตัว โดยแยกการใช้งานเป็นดังนี้

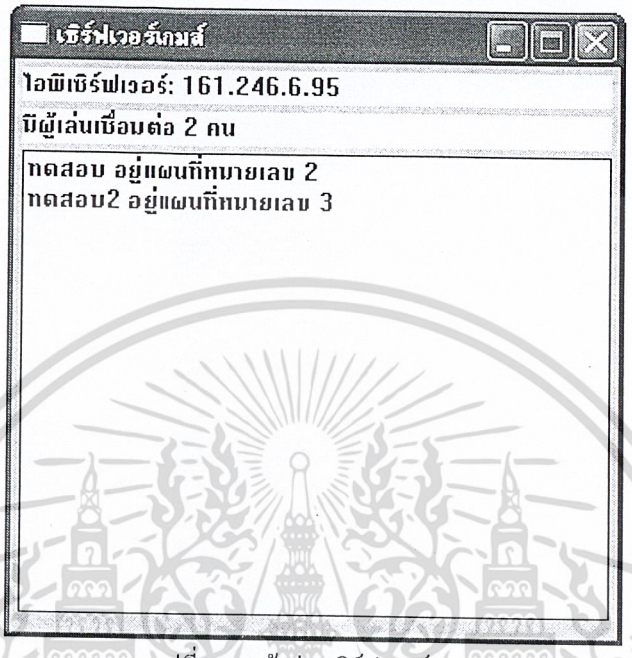
7.2.1 แบบไม่มีการกระจายการประมวลผล

ส่วนนี้จะใช้เซิร์ฟเวอร์เพียงตัวเดียวดังนั้นจึงสามารถทำการติดตั้งระบบได้โดยใช้เครื่องเซิร์ฟเวอร์เพียงเครื่องเดียวเท่านั้น โดยการเลือกอุปกรณ์การ์ดเครือข่ายที่จะใช้ในการติดต่อกับไคลเอนต์ดังรูปที่ 7.4



รูปที่ 7.4 เลือกอุปกรณ์การ์ดเครือข่าย

หลังจากเลือกการ์ดเครือข่ายแล้วระบบจะทำการติดตั้งเองโดยอัตโนมัติ ซึ่งเมื่อทำการติดตั้งเสร็จแล้วจะมีหน้าต่างเซิร์ฟเวอร์เกมเกิดขึ้นดังรูปที่ 7.5 ซึ่งในหน้าต่างนี้จะบอกรายละเอียดของไอพีแอดเดรสของเครื่องที่ทำหน้าที่เป็นเซิร์ฟเวอร์ จำนวนผู้เชื่อมต่อ รายชื่อผู้เชื่อมต่อ และแผนที่ที่ผู้เล่นอยู่

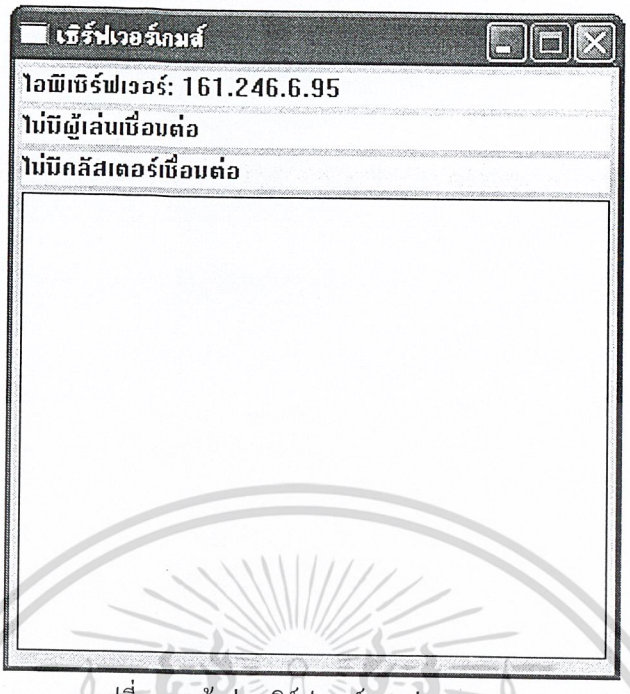


รูปที่ 7.5 หน้าต่างเซิร์ฟเวอร์เกม

7.2.2 แบบมีการกระจายการประมวลผล

การติดตั้งเซิร์ฟเวอร์แบบนี้จะต้องใช้เซิร์ฟเวอร์ 1 ตัวเพื่อทำการแจกจ่ายแมสเสจของเกมให้กับคลัสเตอร์โฮสต์แต่ละตัว ในการทดสอบนี้จะใช้คลัสเตอร์โฮสต์ 3 ตัวทำการประมวลผลเกมที่มีแผนที่อยู่ 3 ฉาก ซึ่งมีขั้นตอนในการติดตั้งระบบดังนี้

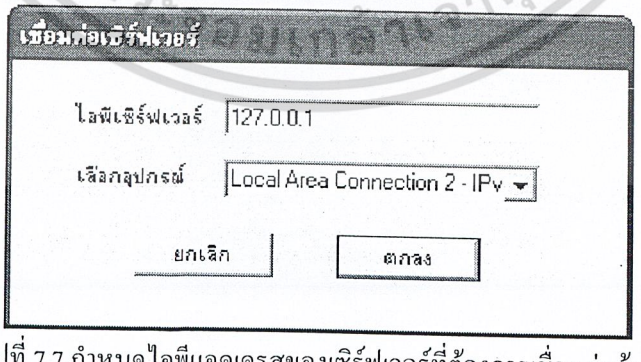
- ติดตั้งระบบของเซิร์ฟเวอร์ที่ทำหน้าที่แจกจ่ายแมสเสจ โดยการเลือกอุปกรณ์การ์ดเครือข่ายที่ต้องการใช้ดังรูปที่ 7.4 จากนั้นระบบจะทำการติดตั้งเองโดยอัตโนมัติเมื่อสำเร็จแล้วจะปรากฏหน้าต่างเซิร์ฟเวอร์เกมเกิดขึ้นเช่นกัน แต่จะแตกต่างจากแบบไม่มีการกระจายการประมวลผลตรงที่หน้าต่างจะปรากฏจำนวนของคลัสเตอร์โฮสต์ที่เชื่อมต่ออยู่ด้วยดังรูปที่ 7.6



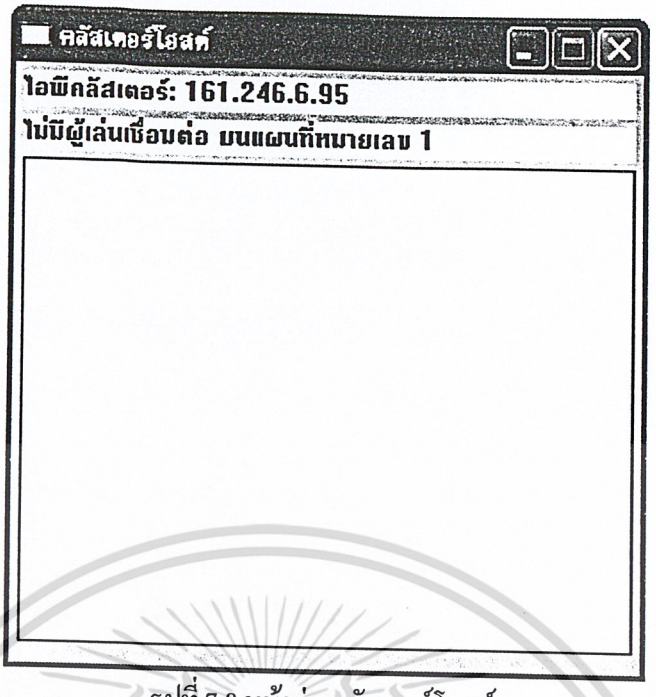
รูปที่ 7.6 หน้าต่างเซิร์ฟเวอร์แจกจ่ายแมสเสจ

-ติดตั้งระบบของคลัสเตอร์โฮสต์ จะใช้คลัสเตอร์โฮสต์ 1 ตัวต่อแผนที่ 1 ฉาก ซึ่งในการติดตั้งตัวคลัสเตอร์โฮสต์จะต้องทำการเชื่อมต่อกับเซิร์ฟเวอร์แจกจ่ายแมสเสจโดยจะต้องกำหนดไอพีแอดเดรสของเซิร์ฟเวอร์ลงในช่องไอพีเซิร์ฟเวอร์ และเลือกอุปกรณ์การ์ดเครือข่ายที่จะใช้ในการติดต่อ ดังรูปที่ 7.7 จากนั้นระบบจะทำการติดตั้งเองโดยอัตโนมัติ โดยที่ระบบจะกำหนดหมายเลขของแผนที่ตามลำดับของการเชื่อมต่อกับเซิร์ฟเวอร์ ซึ่งตัวที่เชื่อมต่อตัวแรกจะเป็นแผนที่ 1 ตัวที่เชื่อมต่อหลังจากนั้นจะเป็นแผนที่ 2 และ 3 ตามลำดับ ซึ่งในการทดสอบครั้งนี้จะทำการทดสอบเพียง 3 ฉากเท่านั้น

เมื่อติดตั้งสำเร็จจะปรากฏหน้าต่างคลัสเตอร์โฮสต์ขึ้น ซึ่งภายในจะมีรายละเอียดของไอพีแอดเดรสที่ทำหน้าที่คลัสเตอร์ จำนวนผู้เล่นที่เชื่อมต่อ และรายชื่อผู้เล่นที่เชื่อมต่อ ดังรูปที่ 7.8



รูปที่ 7.7 กำหนดไอพีแอดเดรสของเซิร์ฟเวอร์ที่ต้องการเชื่อมต่อด้วย



รูปที่ 7.8 หน้าต่างคลัสเตอร์โฮสต์

7.3 การทดสอบประสิทธิภาพของระบบ

เนื่องจากเซิร์ฟเวอร์ที่ใช้ในการประมวลผลเกมนั้นมีอยู่ 2 แบบด้วยกัน ดังนั้นวิธีในการทดสอบประสิทธิภาพของระบบนั้นจะทำการทดสอบระบบทั้ง 2 แบบ เพื่อเปรียบเทียบกันให้เห็นถึงข้อดีและข้อเสียของแต่ละระบบ ลักษณะรูปแบบของการทดสอบนั้นจะอาศัยจำนวนของตัวละครที่เป็นมอนสเตอร์แทนโหลดของระบบ การทดลองในแต่ละครั้งจะมีจำนวนของมอนสเตอร์ในแต่ละแผนที่ต่างๆเท่ากัน ดังแสดงในตารางที่ 7.1

จากนั้นจะทำการเชื่อมต่อผู้เล่นฝั่งไคลเอนต์เข้ากับเซิร์ฟเวอร์เพียงตัวเดียวเพื่อทำการหาค่าเวลาตอบสนองระหว่างไคลเอนต์กับเซิร์ฟเวอร์ในแต่ละครั้งแล้วนำค่านั้นมาเปรียบเทียบกันในแต่ละแบบ

การทดลอง	แผนที่ 1	แผนที่ 2	แผนที่3	จำนวนรวม
1	10	10	10	30
2	20	20	20	60
3	30	30	30	90
4	40	40	40	120

ตารางที่ 7.1 จำนวนของมอนสเตอร์ในแต่ละแผนที่

คอมพิวเตอร์ที่ใช้ในการทดสอบนี้มีความเร็วของหน่วยประมวลผล(CPU) ของเครื่องที่แตกต่างกันดังแสดงในตารางที่ 7.2

รูปแบบการทำงาน	ไคลเอนต์	เซิร์ฟเวอร์	คลัสเตอร์ โฮสต์ 1	คลัสเตอร์ โฮสต์ 2	คลัสเตอร์ โฮสต์3
ไม่กระจายการประมวลผล	1.3 GHz	1.47 GHz	-	-	-
กระจายการประมวลผล	1.3 GHz	1.47 GHz	1.6 GHz	1.6 GHz	1.6 GHz

ตารางที่ 7.2 ความเร็วของหน่วยประมวลผล

7.3.1 การทดสอบระบบแบบไม่มีการกระจายการประมวลผล

การทดสอบมีขั้นตอนดังนี้คือ

- ทำการติดตั้งระบบเซิร์ฟเวอร์
- เชื่อมต่อไคลเอนต์เข้ากับเซิร์ฟเวอร์
- ป้อนอินพุตที่ฝั่งไคลเอนต์ เช่นการเดิน การฟันมอนสเตอร์
- บันทึกค่าเวลาตอบสนอง

หลักการคำนวณค่าเวลาตอบสนองมีดังนี้คือ

- ป้อนอินพุตที่ฝั่งไคลเอนต์ เช่นการเดิน การฟันมอนสเตอร์
- ไคลเอนต์ทำการบันทึกค่าเวลาปัจจุบันที่ส่ง
- ส่งแมสเสจของเกมส่งไปยังเซิร์ฟเวอร์แจกจ่ายและส่งต่อไปยังคลัสเตอร์โฮสต์
- คลัสเตอร์โฮสต์ส่งแมสเสจอัพเดตสถานะมายังเซิร์ฟเวอร์แจกจ่ายและส่งกลับมายังไคลเอนต์
- ไคลเอนต์ทำการบันทึกค่าเวลาปัจจุบันที่รับ
- นำค่าเวลาปัจจุบันที่รับมาลบกับค่าเวลาปัจจุบันที่ส่งจะได้ค่าเวลาตอบสนอง

ค่าเวลาที่ทำการบันทึกมีดังนี้

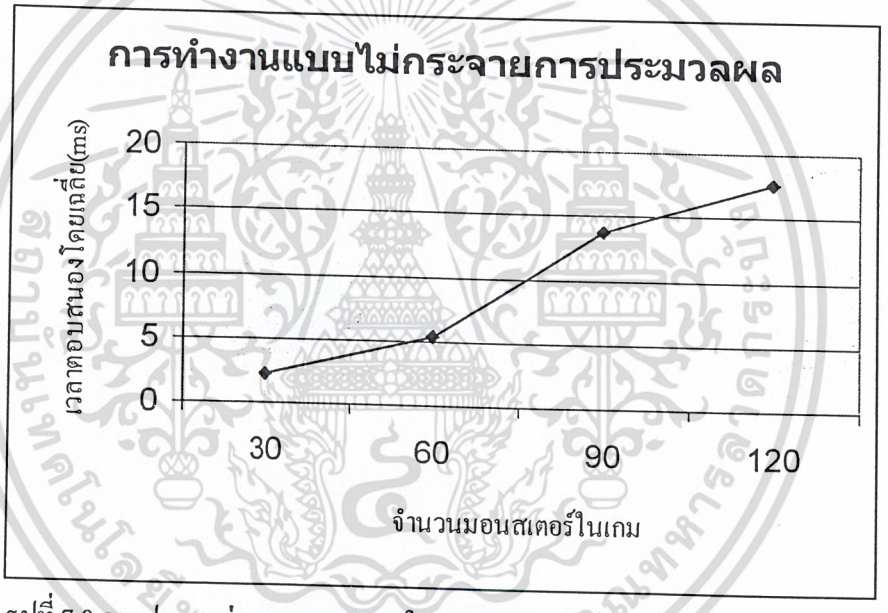
- ค่าเวลาสูงสุดคือค่าเวลาตอบสนองสูงสุดที่ทำการบันทึกได้
- ค่าเวลาต่ำสุดคือค่าเวลาตอบสนองต่ำสุดที่ทำการบันทึกได้
- ค่าเวลาเฉลี่ยคือการหาผลรวมของค่าเวลาตอบสนองทั้งหมดที่ได้ทำการบันทึกนำมาหารกับจำนวนครั้งที่ได้ทำการบันทึก

ผลการทดสอบสรุปได้ดังตารางที่ 7.3 โดยที่คอลัมน์การทดลองอ้างอิงกับค่าในตารางที่ 7.1 ค่าเวลาสูงสุด ต่ำสุด และค่าเวลาเฉลี่ยเป็นค่าของเวลาการตอบสนองที่ได้ทำการบันทึกเป็นสถิติจากการทดลองหลายๆครั้งในช่วงเวลาที่แตกต่างกัน โดยมีหน่วยเป็นมิลลิวินาที (ms)

การทดลอง	ค่าเวลาสูงสุด	ค่าเวลาต่ำสุด	ค่าเวลาเฉลี่ย
1	16 ms	1 ms	2.4 ms
2	31 ms	1 ms	5.4 ms
3	58 ms	1 ms	13.75 ms
4	64 ms	1 ms	17.6 ms

ตารางที่ 7.3 ผลการทดสอบการทำงานแบบไม่กระจายการประมวลผล

จากผลการทดสอบที่ได้สังเกตเห็นได้ว่ายิ่งมีจำนวนโหลดที่เป็นมอนสเตอร์มากขึ้นจะทำให้ค่าเวลาตอบสนองเฉลี่ยโดยรวมเพิ่มขึ้นตามไปด้วย และค่าเวลาตอบสนองสูงสุดที่วัดได้ก็มีแนวโน้มสูงขึ้นเรื่อยๆ รูปที่ 7.9 แสดงให้เห็นว่ายิ่งโหลดเพิ่มมากขึ้นเท่าใดค่าเวลาตอบสนองก็จะทวีเพิ่มมากขึ้นตามไปด้วย

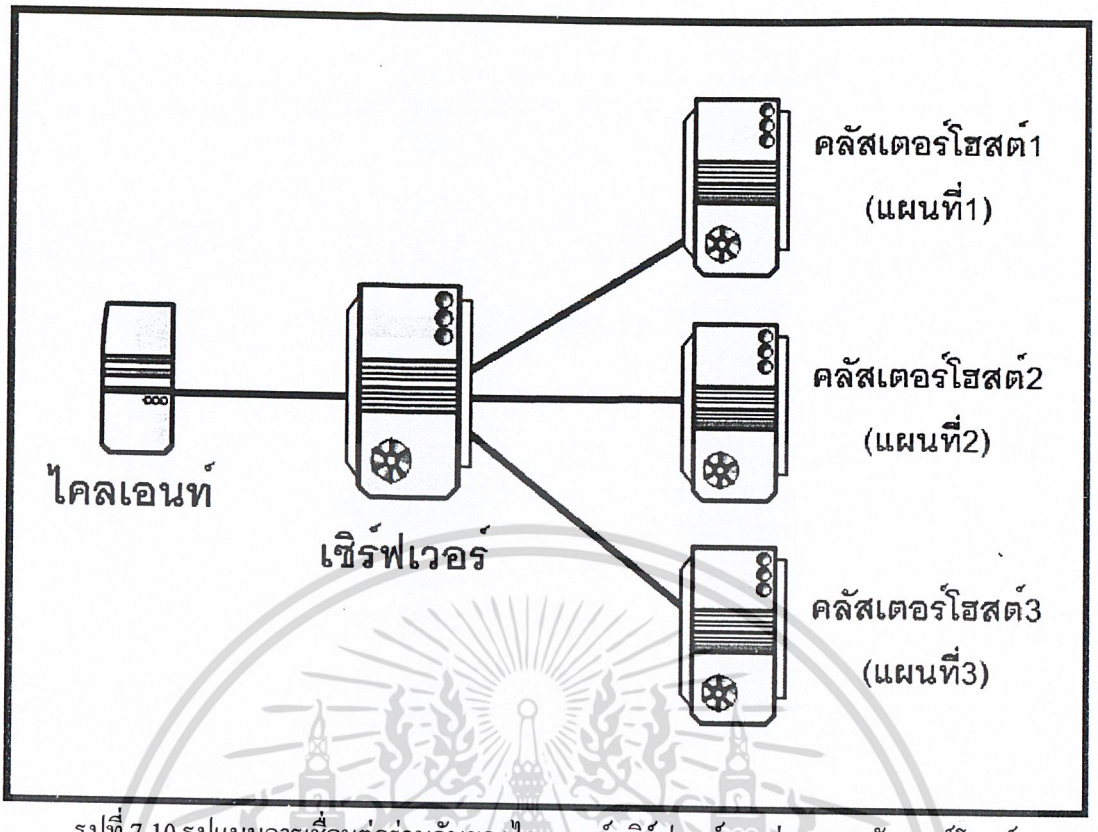


รูปที่ 7.9 กราฟแสดงค่าเวลาตอบสนองในการทำงานแบบไม่กระจายการประมวลผล

7.3.2 การทดสอบระบบแบบมีการกระจายการประมวลผล

การทดสอบมีขั้นตอนดังนี้คือ

- ทำการติดตั้งระบบเซิร์ฟเวอร์แจกจ่าย
- ทำการเชื่อมต่อคลัสเตอร์โฮสต์ทั้ง 3 ตัวเข้ากับเซิร์ฟเวอร์แจกจ่าย
- เชื่อมต่อไคลเอนต์เข้ากับเซิร์ฟเวอร์แจกจ่าย
- ป้อนอินพุตที่ฝั่งไคลเอนต์ เช่นการเดิน การฟันมอนสเตอร์
- บันทึกค่าเวลาตอบสนอง

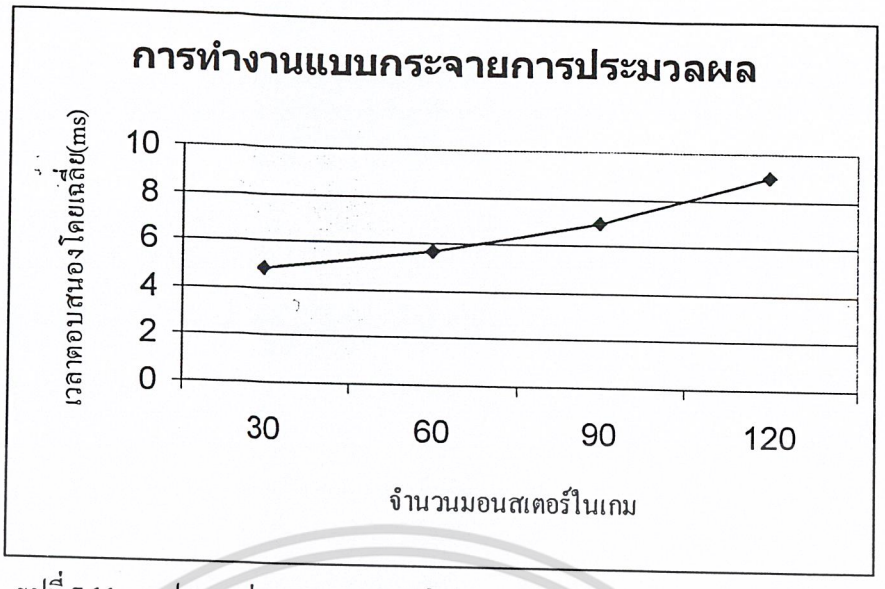


รูปที่ 7.10 รูปแบบการเชื่อมต่อร่วมกันของไคลเอนต์ เซิร์ฟเวอร์แจกจ่ายและคลัสเตอร์โฮสต์

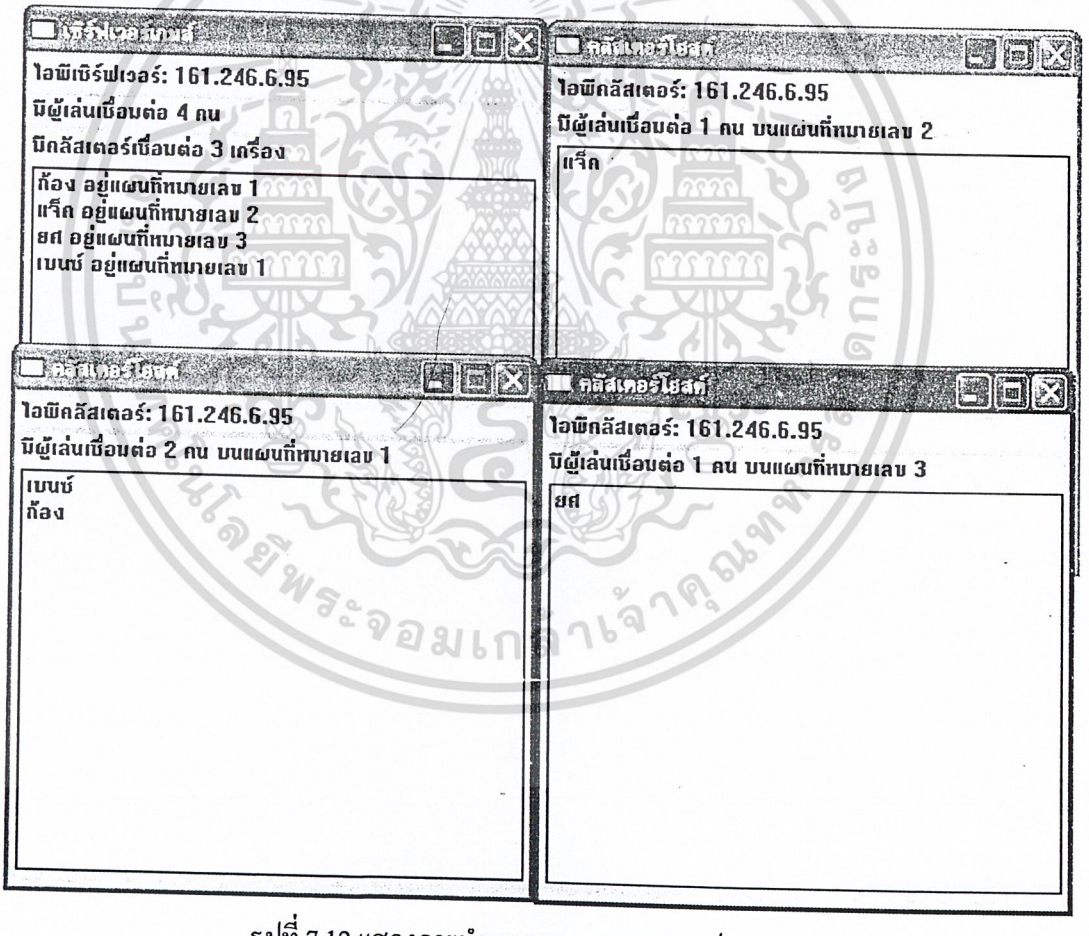
ผลการทดสอบสรุปได้ดังตารางที่ 7.4 และรูปที่ 7.11 สามารถสังเกตเห็นได้ว่าค่าเวลาตอบสนองจะเพิ่มขึ้นน้อยกว่าแบบไม่กระจายการประมวลผล ถึงแม้ช่วงที่มีโหลดจำนวนน้อยจะได้ค่าเวลาตอบสนองที่สูงกว่าเล็กน้อย แต่ถ้าโหลดมีจำนวนที่มากขึ้นเรื่อยๆค่าเวลาตอบสนองจะมีค่าน้อยกว่าแบบไม่กระจายการประมวลผล

การทดลอง	ค่าเวลาสูงสุด	ค่าเวลาต่ำสุด	ค่าเวลาเฉลี่ย
1	17 ms	1 ms	4.85 ms
2	39 ms	1 ms	5.7 ms
3	22 ms	1 ms	6.95 ms
4	53 ms	1 ms	8.95 ms

ตารางที่ 7.4 ผลการทดสอบการทำงานแบบกระจายการประมวลผล



รูปที่ 7.11 กราฟแสดงค่าเวลาตอบสนองในการทำงานแบบกระจายการประมวลผล



รูปที่ 7.12 แสดงการทำงานแบบกระจายการประมวลผล

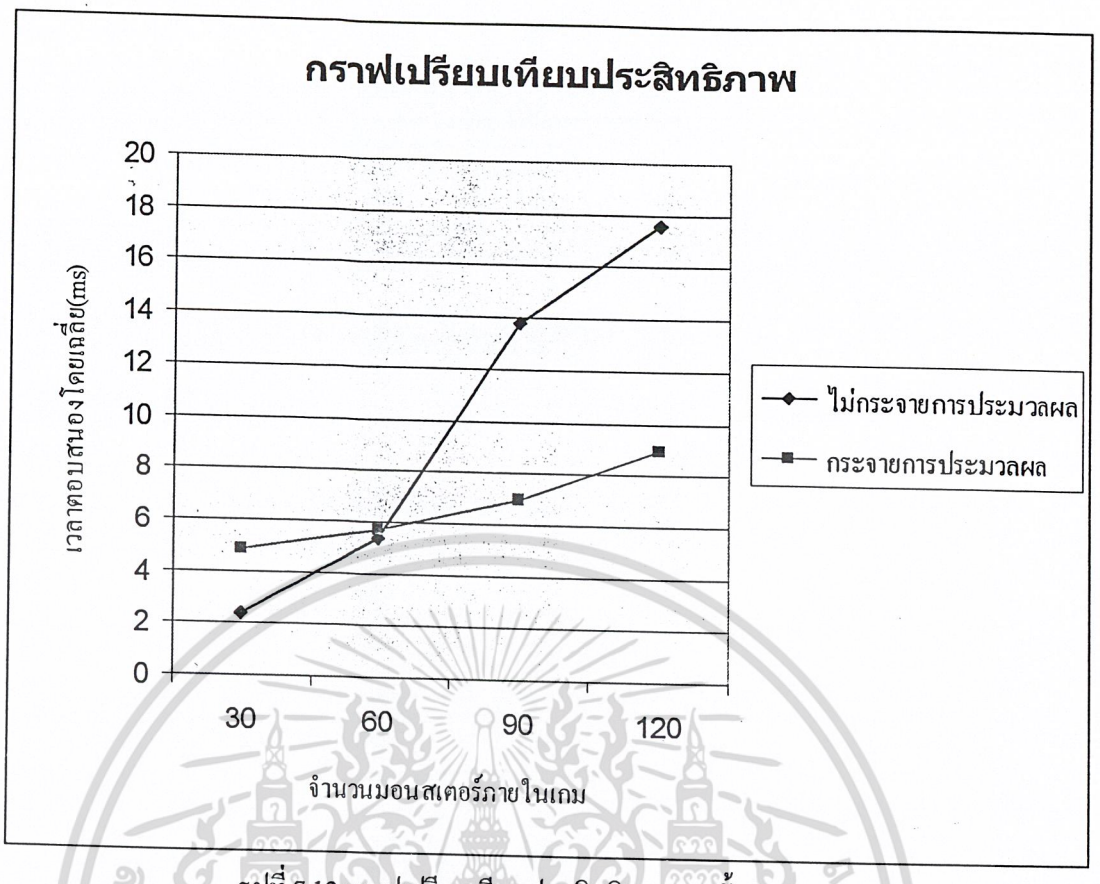
รูปที่ 7.12 แสดงหน้าต่างของเซิร์ฟเวอร์แจกจ่าย และหน้าต่างคลัสเตอร์โฮสต์ขณะทำงานแบบมีการกระจายการประมวลผลที่มีผู้เล่นเชื่อมด้วยกัน 4 คน และมีคลัสเตอร์โฮสต์เชื่อมต่ออยู่ 3 ตัว โดยที่มีผู้เล่นอยู่ฉากแผนที่แตกต่างกันดังนี้

- แผนที่ 1 มีผู้เล่นชื่อ ก้องและเบนซ์กำลังเล่นอยู่
- แผนที่ 2 มีผู้เล่นชื่อ แจ็คกำลังเล่นอยู่
- แผนที่ 3 มีผู้เล่นชื่อ ยศกำลังเล่นอยู่

จากดังรูปจะสังเกตได้ว่ามีรายชื่อผู้เล่นกระจายไปตามหน้าต่างของคลัสเตอร์โฮสต์ต่างๆ ตามแผนที่ที่ตัวเองกำลังเล่นอยู่ซึ่งทำให้สามารถทราบได้ว่าขณะนี้ผู้เล่นกำลังถูกกระจายการประมวลผลไปที่คลัสเตอร์โฮสต์ตัวใดอยู่ และที่หน้าต่างของเซิร์ฟเวอร์แจกจ่ายก็จะแสดงรายชื่อผู้เล่นทั้งหมดที่ทำการเชื่อมต่อรวมทั้งแผนที่ที่ผู้เล่นอยู่

7.4 กราฟสรุปประสิทธิภาพของระบบ

จากการทดลองที่ผ่านมานั้นสามารถพล็อตเป็นกราฟเปรียบเทียบประสิทธิภาพของระบบได้ดังรูปที่ 7.13 โดยสังเกตได้ว่าในขณะที่มีจำนวนโหลดน้อยๆการทำงานแบบกระจายการประมวลผลจะใช้เวลามากกว่าเนื่องจากต้องทำการส่งจ่ายแมสเสจข้อมูลไปมาระหว่างคลัสเตอร์โฮสต์กับเซิร์ฟเวอร์ ในขณะที่แบบไม่กระจายการประมวลผลไม่ต้องทำการส่งจ่ายแมสเสจระหว่างเซิร์ฟเวอร์จึงทำให้ได้เวลาตอบสนองที่น้อยกว่า แต่ถ้าหากในขณะนั้นมีโหลดจำนวนมาก แบบไม่กระจายการประมวลผลจะใช้เวลาค่อนข้างมากเนื่องจากต้องรับภาระในการประมวลผลผู้เล่นทุกคนที่เข้ามาเชื่อมต่อเล่นเกม รวมทั้งปัจจัยอื่นๆเช่นตัวละครที่ผู้เล่นไม่ได้บังคับทุกตัวเช่น มอนสเตอร์ ตัวละครประกอบอื่นๆเป็นต้น แต่ถ้าเป็นแบบกระจายการประมวลผลค่าเวลาตอบสนองที่ได้จะน้อยกว่ามากเนื่องจากจะมีการแบ่งการประมวลผลไปให้กับเครื่องคลัสเตอร์โฮสต์ต่างๆจึงทำให้แบบนี้สามารถรองรับผู้เล่นได้จำนวนมาก



รูปที่ 7.13 กราฟเปรียบเทียบประสิทธิภาพของทั้งสองระบบ

บทที่ 8
บทวิจารณ์และสรุป

8.1 สรุป

จากการศึกษาและทดลองการทำงานของเกมสามมิติออนไลน์แบบกระจายการประมวลผล สามารถที่จะสรุปถึงจุดเด่นได้ดังนี้คือ สามารถที่จะช่วยแบ่งเบาภาระการประมวลผลบนเซิร์ฟเวอร์เพียงตัวเดียวได้เป็นอย่างมากโดยเฉพาะในกรณีที่มีจำนวนโหลดมาก ซึ่งเซิร์ฟเวอร์แบบที่ไม่มีการกระจายการประมวลผลจะต้องทำงานอย่างหนักเพื่อประมวลผลทุกๆอย่างไม่ว่าจะผู้เล่นที่เชื่อมต่อหรือตัวละครและข้อมูลอื่นๆภายในเกมทำให้ค่าเวลาตอบสนองในกรณีที่มีโหลดจำนวนมากมีค่าสูง แต่ถ้าเป็นแบบมีการกระจายการประมวลผลแล้วโหลดต่างๆจะถูกส่งไปประมวลผลยังที่คลัสเตอร์โฮสต์แต่ละตัวทำให้ระบบสามารถรองรับจำนวนโหลดที่เพิ่มมากขึ้นได้

การทำงานแบบการกระจายโหลดที่ใช้ในงานวิจัยนี้ยังมีจุดด้อยตรงที่การทำงานของระบบยังมีโอเวอร์เฮดที่ยังค่อนข้างสูงอยู่เนื่องจากการติดต่อสื่อสารระหว่างไคลเอนต์กับส่วนคลัสเตอร์โฮสต์ที่ทำหน้าที่ประมวลผลต้องผ่านตัวเซิร์ฟเวอร์แจกจ่ายอยู่ตลอดเวลาทำให้การเวลาตอบสนองในขณะที่มีจำนวนโหลดไม่มากนักมีค่าสูงกว่าแบบที่ไม่มีการกระจายการประมวลผลอยู่เล็กน้อย ถ้าสามารถจัดการลดโอเวอร์เฮดลงได้ระบบการทำงานของการกระจายการประมวลผลจะมีความสมบูรณ์แบบอยู่มาก

8.2 แนวทางในการพัฒนาต่อ

1 เพิ่มเติมความสามารถให้คลัสเตอร์โฮสต์สามารถรายงานค่าเวลาตอบสนองเฉลี่ยบนตัวเองได้เพื่อที่จะได้ทราบสภาวะการทำงานของคลัสเตอร์โฮสต์นั้นๆ

2 พัฒนาการกระจายการประมวลผลแบบแบ่งแผนที่ให้สามารถกระจายแผนที่เดียวกันออกเป็นหลายๆคลัสเตอร์โฮสต์ได้เผื่อในกรณีที่มีโหลดจำนวนมากอยู่แผนที่ใดแผนที่หนึ่งโดยเฉพาะ

3 พัฒนาความสามารถในการกระจายการประมวลผลให้ทำงานแบบโหลดบาลานซ์ (Load balancing) ซึ่งจะทำให้การกระจายการประมวลผลมีประสิทธิภาพมากขึ้น

8.3 การนำไปประยุกต์ใช้งาน

งานวิจัยนี้สามารถนำไปประยุกต์ใช้งานได้ในเกมออนไลน์ทั่วๆไปที่ต้องการสร้างระบบให้สามารถรองรับผู้เล่นได้จำนวนมากโดยแบ่งให้แผนที่ที่มีผู้เล่นจำนวนมากกำหนดให้ใช้เครื่องคลัสเตอร์โฮสต์ที่มีประสิทธิภาพสูงในการประมวลผล ในขณะแผนที่ทั่วๆไปที่มีจำนวนผู้เล่นไม่มากนักอาจรวมอยู่ภายในคลัสเตอร์โฮสต์เดียวกันเป็นต้น

บรรณานุกรม

[1] Jim Adams: "Programming Role Playing Games with DirectX®", Premier Press, 2002

[2] Microsoft Corporation: "Windows 2000 Network Load Balancing Technical Overview", 2000

[3] http://www.redhat.com/docs/manuals/enterprise/RHEL-AS-2.1-Manual/install-guide/s1-lvs-scheduling.html: "LVS Scheduling Overview", Redhat Documentation

[4] Peter J. Kovach: "Inside Direct3D", Microsoft Press, 2000

[5] Omar A. Abdelwahed: "Distributed_gaming", http://www.gamedev.net/reference/articles/article1948.asp,2003