การพัฒนา Collaborative Networked Game โดยใช้อุปกรณ์ตอบสนองต่อแรง

Development of Collaborative Networked Game Using Haptic Devices

โดย

นายอุรเดน กิจพยัคฆ์

รหัส 43067037

H001917

อาจารย์ที่ปรึกษา

ผศ.ดร.นพพร โชติกกำธร

รายงานนี้เป็นส่วนหนึ่งของวิชาโครงการพัฒนาระบบงาน

หลักสูตรวิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ

ภาคเรียนที่ 1 ปีการศึกษา 2545

คณะเทคโนโลยีสารสนเทศ

สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง

ชื่อหัวข้อ	การพัฒนา Collaborative Networked Game โดยใช้อุปกรณ์ตอบสนองต่อแรง
นักศึกษา	นายอุราเคน กิจพยัคฆ์
อาจารย์ที่ปรึกษา	ผศ.ดร.นพพร โชติกกำเธร
ระดับการศึกษา	วิทยาศาสตรมหาบัณฑิต สาขาวิชาเทคโนโลยีสารสนเทศ
แขนงวิชา	วิทยาการสารสนเทศ
ปีการศึกษา	2545

บทคัดย่อ

โครงการนี้ได้ทำการศึกษาถึงการออกแบบและพัฒนาเกมคอมพิวเตอร์ โดยการนำเอาอุปกรณ์ตอบสนองต่อแรงสัมผัส (Haptic Device) มาใช้เป็นอุปกรณ์รับข้อมูลและสร้างแรงตอบสนองต่อแรงสัมผัสที่สระทำ โดยลักษณะเกมส์เป็นการแสดงถึงแรงตอบสนองและภาพที่สัมพันธ์กันของการจำลองการบังคับห่วงกลมไม่ให้สัมผัสกับเส้นลวดคัดโค้ง ซึ่งบังคับร่วมกันโดยผู้เล่นสองคน ผ่านเครือข่ายคอมพิวเตอร์ โดยได้ทำการศึกษาและประยุกต์ใช้ GHOST SDK ในการทำ Haptic Simulation และใช้ OpenGL ในการแสดงภาพทางกราฟิก

Title	Development of Collaborative Networked Game Using Haptic Devices
Student	Mr. Uraken Kitpayak
Advisor	Asst.Prof.Dr. Nopporn Chotikakamthorn
Level of Study	Master of Science in Information Technology
Major	Information Science
Academic Year	2002

ABSTRACT

This project is a study and development about computer game that using force feedback device (Haptic Device) for input information and for feedback a force. For this game, it shows force-interaction and relative graphic of a control-ring simulation game, that its rule is prevent form touching a ring with any bend-coil by collaborative control of two players that join across computer network. For implementation, we study and apply GHOST SDK for haptic simulation and OpenGL for graphic representation.

กิตติกรรมประกาศ

ผู้เขียนขอขอบพระคุณเป็นอย่างยิ่งสำหรับ ผศ.ดร. นพพร โชติกกำธร ที่ได้ให้โอกาสและความเมตตา ขอบคุณคณาจารย์ทุกท่านที่ได้ประสิทธิ์ประสาทวิชา และสุดท้ายขอขอบพระคุณคุณพ่อ คุณแม่ และพี่สาว สำหรับการสนับสนุนในทุก ๆ ด้าน

อุรานคน กิจพยัคฆ์

สารบัญ

สารบัญ(ต่อ)

สารบัญ(ต่อ)

หน้า

สารบัญตาราง

หน้า

ตารางที่

สารบัญตาราง(ต่อ)

หน้า

ตารางที่

สารบัญภาพ

หน้า

รูปที่

บทที่ 1

บทนำ

1.1 โครงการพัฒนาระบบงาน

ปัจจุบันคอมพิวเตอร์และเทคโนโลยีสารสนเทศ มีความสำคัญในชีวิตประจำวันในสัดส่วนที่มากขึ้นอย่างมาก เนื่องจากคอมพิวเตอร์มีราคาถูกลง แต่มีประสิทธิภาพที่มากขึ้น ทำให้เกิดการใช้งานอย่างแพร่หลาย ทำให้เกิดเทคโนโลยีต่อเนื่องตามมามากมาย ทั้งในแง่เพื่อการศึกษา, วิทยาศาสตร์, อุตสาหกรรม, หรือความบันเทิง แต่ผลท้ายสุดคือเพื่อตอบสนองความต้องการของมนุษย์ ในส่วนของเทคโนโลยีเสมือนจริง (Virtual Reality) ที่โครงการนี้ได้ทำการศึกษาและพัฒนาก็เป็นความต้องการอันหนึ่งของมนุษย์ที่จะจำลองการตอบสนองหรือให้ความรู้สึกที่เป็นธรรมชาติหรือเสมือนจริง ระหว่างมนุษย์กับคอมพิวเตอร์และอุปกรณ์ต่อพ่วง ซึ่งมีประโยชน์ทั้งในแง่ของความบันเทิงและในแง่การควบคุมการทำงาน

โครงการนี้เป็นการศึกษา, พัฒนา, และประยุกต์ใช้งานอุปกรณ์ประเภทตอบสนองต่อแรง (Force Feedback Device) โดยพัฒนาเป็นเกมคอมพิวเตอร์ที่ร่วมเล่นกันสองคนโดยผ่านเครือข่ายคอมพิวเตอร์ ลักษณะของเกมซึ่งต่อไปขอเรียกว่าเกม RealLife เป็นการควบคุมห่วงวงกลมเสมือนร่วมกันของผู้เล่นสองคนเพื่อให้ร้อยผ่านไปตามแนวลวดดัดโค้งเสมือน โดยผู้เล่นแต่ละคนก็มีอุปกรณ์ตอบสนองต่อแรงเพื่อใช้ในการควบคุมการเคลื่อนที่ของห่วง ถ้าหากผู้เล่นคนหนึ่งมีการเคลื่อนที่ก็จะมีแรงส่งผลไปยังอีกผู้เล่นหนึ่งหากตำแหน่งของทั้งสองไม่สัมพันธ์กัน (เนื่องจากควบคุมห่วงอันเดียวกัน) ผู้เล่นทั้งสองจึงต้องประสานร่วมกันทั้งดึงและดันเพื่อพาห่วงให้ร้อยผ่านลวดเสมือนที่ดัดโค้งไปให้ได้จนพ้น ถ้าห่วงชนกับลวดก็จะเสียคะแนนและมีแรงปฏิกิริยาตอบสนอง ซึ่งในการพัฒนาจะใช้การสื่อสารข้อมูลผ่านเครือข่ายคอมพิวเตอร์ ในการแลกเปลี่ยนข้อมูลระหว่างสองผู้เล่น

1.2 ประโยชน์ของโครงการ

โครงการนี้เป็นทั้งการศึกษาและการพัฒนาในหลาย ๆ ด้าน ซึ่งเป็นประโยชน์ต่อผู้ศึกษาเองและผู้พัฒนาต่อในภายหลัง ประกอบด้วยการศึกษาและพัฒนาการสร้างภาพกราฟิกให้สัมพันธ์กับการตอบสนองด้วยแรงของอุปกรณ์จำพวก Force Feedback Device, การศึกษาและพัฒนาการติดต่อสื่อสารผ่านเครือข่ายคอมพิวเตอร์ด้วย TCP และ Multicast (บน UDP/IP) โดยใช้ Windows Sockets สำหรับพัฒนาเกมที่มีลักษณะ Collaborative Networked Interactive Real-time Applications, และการศึกษาและพัฒนาการตรวจสอบการชนและสร้างแรงตอบสนองของอุปกรณ์ Force Feedback Device สำหรับการจำลองการตอบสนองที่ไม่ใช่ Point-based (โดยพัฒนาเป็นวงแหวน).

1.3 สภาพแวดล้อมในการพัฒนา

เครื่องมือหรืออุปกรณ์ที่ใช้ภายในโครงการนี้ประกอบด้วยอุปกรณ์จำพวก Force Feedback Device คือ PHANToM Premium 1.0 with Encoder gimbal/Finger stylus 2 ชุด ที่ต่ออยู่กับเครื่องคอมพิวเตอร์แต่ละเครื่องและเชื่อมต่อกันผ่านเครือข่าย Ethernet LAN ในการพัฒนาโปรแกรมได้พัฒนาบนระบบปฏิบัติการ Windows NT 4.0 โดยใช้ Visual C++ Version 6 ในการพัฒนา ส่วน Library ที่ใช้คือ GHOST SDK Version 2.1 สำหรับติดต่อกับ PHANToM ส่วน OpenGL สำหรับแสดงภาพกราฟิกสามมิติ และ Windows Socket สำหรับการติดต่อสื่อสารผ่านเครือข่ายคอมพิวเตอร์

1.4 แนวทางการพัฒนา

ในการพัฒนาโครงการนี้ ต้องใช้ความรู้ในหลายด้านและมีความเกี่ยวข้องกับเทคโนโลยีหลายแขนง อีกทั้งยังมีข้อจำกัดต่าง ๆ ที่ต้องคำนึง โดยสรุปเป็นประเด็นสำคัญ คือ

1) การพัฒนาการตอบสนองต่อแรงและภาพที่สัมพันธ์กัน

2) การพัฒนาโปรแกรมที่เป็นลักษณะ Interactive real-time application

3) การพัฒนาโปรแกรมที่มีหลายผู้เล่นผ่านเครือข่ายคอมพิวเตอร์

1.4.1 การพัฒนาการตอบสนองต่อแรงและภาพที่สัมพันธ์กัน

เนื่องจากการตอบสนองต่อแรงเพียงอย่างเดียวโดยไม่เห็นภาพ ยังให้ข้อมูลที่เป็นประโยชน์ไม่เพียงพอในการติดต่อหรือใช้งานโดยมนุษย์ จึงต้องมีการพัฒนาการแสดงภาพที่สัมพันธ์กันด้วย โดยอุปกรณ์ตอบสนองต่อแรงที่ใช้คือ PHANToM มีลักษณะ 6 dof (Degree-Of-Freedom) คือ รับข้อมูลในแบบ Cartesian 3 แกน และในแบบแกนหมุน 3 แกน รวมเป็น 6 dof แต่แสดงข้อมูลออก

คือ แรง เพียง 3 dof คือในแบบ Cartesian ตามแนวแกน x, y, z โดยยังไม่สามารถแสดงแรงหมุนได้ PHANToM มี Software Library ที่ช่วยในการพัฒนาและใช้งานคือ GHOST SDK สิ่งที่ต้องทำคือการศึกษาการใช้งาน GHOST SDK เพื่อรับข้อมูลการบังคับและเพื่อแสดงแรงตอบสนองทั้งในแบบสนามของแรง และการจำลองวัตถุเสมือน ต่อจากนั้นจึงศึกษาและนำข้อมูลที่ได้มาสร้างเป็นภาพกราฟิก 3 มิติ ที่สัมพันธ์กับวัตถุเสมือนนั้น โดยใช้ Library ทางด้านกราฟิกคือ OpenGL แล้วจึงทำการศึกษาในส่วนการตรวจสอบขอบเขตการชน เพื่อใช้ในการจำลองวัตถุเสมือนหรือการแสดงสนามของแรงในแบบต่าง ๆ ที่ไม่มีใน GHOST SDK เพื่อใช้ในการพัฒนาเกมต่อไป

1.4.2 การพัฒนาโปรแกรมที่เป็นลักษณะ Interactive real-time application

เกม RealLife มีลักษณะเป็น Interactive และ Real-time คือ มีความต้องการการตอบสนองที่ถูกต้องและรวดเร็วพอจนเสมือนว่าเกิดขึ้นจริงและในเวลาจริง โดยโครงการนี้มอง Interactive ในแง่ของความถูกต้องในการตอบสนองระหว่างสองผู้เล่น ซึ่งต้องศึกษาการออกแบบเกมเพื่อให้เกิดความเป็นธรรมและถูกต้องตรงกันระหว่างสองผู้เล่น โดยโครงการนี้พยายามใช้หลักการ Bucket Synchronization เข้ามาประยุกต์ใช้ ในส่วนของ Real-time มองในแง่ของการตอบสนองให้ตรงเวลาจริงมากที่สุด สำหรับกรณีเกมนี้คือ ให้รวดเร็วที่สุด ซึ่งต้องศึกษาถึงข้อจำกัดต่าง ๆ ทั้งในด้าน Software และ Hardware เพื่อให้ประสิทธิภาพโดยรวมดีที่สุด

1.4.3 การพัฒนาโปรแกรมที่มีหลายผู้เล่นผ่านเครือข่ายคอมพิวเตอร์

ถึงแม้ว่าเกม RealLife จะเป็นการเล่นระหว่างผู้เล่นสองคนผ่านเครือข่ายคอมพิวเตอร์ แต่โครงการนี้ก็ได้ศึกษาและพัฒนาในแนวทางที่จะรองรับการขยายจำนวนผู้เล่นหรือผู้สังเกตการณ์ที่อาจเพิ่มในอนาคตได้ โดยใช้การ Multicast ในการส่งข้อมูลภายในกลุ่ม และใช้การสื่อสารแบบ Connection-oriented ในลักษณะ Client-Server ช่วยในการควบคุมการติดต่อสื่อสารระหว่างผู้เล่นอีกชั้นหนึ่ง โดยใช้ Windows Socket ในการพัฒนาโปรแกรม

1.5 โครงสร้างเอกสาร

เอกสารโครงการนี้ประกอบด้วย 7 บท และภาคผนวก 2 บท คือ

บทที่ 1) บทนำ (Introduction)

บทที่ 2) การออกแบบ (Design)

บทที่ 3) ระบบย่อยกราฟิก (Graphics Subsystem)

บทที่ 4) ระบบย่อยสื่อสาร (Communications Subsystem)

บทที่ 5) ระบบย่อยการตอบสนองต่อแรง (Haptics Subsystem)

บทที่ 6) โปรแกรมเกม (RealLife Game)

บทที่ 7) บทสรุป (Conclusion)

ภาคผนวก ก. การติดตั้งโปรแกรมเกม RealLife

ภาคผนวก ข. รหัสโปรแกรมเกม RealLife

โดยบทที่ 2 กล่าวถึงการออกแบบโดยรวมของเกม RealLife, Logic ของเกม, ทฤษฎีและเหตุผลของการออกแบบดังกล่าว ส่วนบทที่ 3 กล่าวถึงการพัฒนาส่วนกราฟิกเพื่อรองรับการแสดงผลโดยใช้ OpenGL ส่วนบทที่ 4 กล่าวถึงการพัฒนาส่วนติดต่อสื่อสารโดยใช้ Windows Socket ในแบบ TCP, การทำ Multicast (บน UDP/IP) ส่วนบทที่ 5 กล่าวถึงส่วนการแสดงการตอบสนองด้วยแรง รวมถึงทฤษฎีที่เกี่ยวข้อง เช่น กระบวนการและวิธีการในการจำลองการตอบสนองของแรงและการสร้างภาพที่สัมพันธ์กัน หลักการตรวจสอบขอบเขตการชนที่ใช้ การพัฒนาเพิ่มเติมในส่วนของเกมที่ต้องใช้ ส่วนบทที่ 6 กล่าวถึงการรวมกันของระบบย่อยต่าง ๆ เพื่อพัฒนาเป็นเกม โครงสร้างโดยรวมทั้ง User Interface และ Logic ของโปรแกรม, การออกแบบ Application Protocol ของเกม และการออกแบบกระบวนการในการประสานงาน และการพัฒนาโปรแกรมเกม และบทที่ 7 เป็นการสรุปผลที่ได้, ปัญหา, และแนวทางในการพัฒนาต่อ

บทที่ 2

การออกแบบ

2.1 เกม RealLife

เกม RealLife เป็นเกมคอมพิวเตอร์ที่ร่วมเล่นกันสองฝ่ายผ่านเครือข่ายคอมพิวเตอร์ โดยแต่ละฝ่ายใช้อุปกรณ์ประเภทตอบสนองต่อแรง (Force Feedback Device) ในการควบคุมห่วงวงกลมเสมือนร่วมกันเพื่อให้ร้อยผ่านไปตามแนวลวดดัดโค้ง ถ้าหากผู้เล่นฝ่ายหนึ่งมีการเคลื่อนที่ก็จะมีแรงส่งผลไปยังผู้เล่นอีกฝ่ายหนึ่งหากตำแหน่งของทั้งสองไม่สัมพันธ์กัน ถ้าห่วงชนกับลวดก็จะเสียคะแนนและมีแรงปฏิกิริยาตอบสนอง โดยผู้เล่นแต่ละฝ่ายจะเห็นเส้นลวดเสมือนและตำแหน่งห่วงที่ตรงกัน.

2.2 แนวคิด

เกม RealLife เป็นเกม Interactive real-time ที่มีผู้เล่นมากกว่าหนึ่งฝ่ายและติดต่อสื่อสารผ่านเครือข่ายคอมพิวเตอร์ ดังนั้นการสื่อสารที่ใช้แลกเปลี่ยนตำแหน่งของห่วงต้องมีความรวดเร็วเพียงพอที่จะรักษาความเป็น Interactive และจังหวะที่เหมาะสมที่ยังคงรักษาความเป็น Real-time จากลักษณะของ Real-time ที่ใช้ข้อมูลเป็นจังหวะที่เหมาะสม สำหรับเกม RealLife ซึ่งข้อมูลที่สื่อสารเป็นตำแหน่งของห่วง และข้อมูลนี้นำไปซึ่งการตอบสนองด้วยแรง และสำหรับอุปกรณ์ Force feedback ที่ใช้คือ PHANToM ที่ใช้ร่วมกับ GHOST SDK นั้นถูกออกแบบให้ตอบสนองในระดับ 1 kHz ดังนั้นจังหวะที่ยอมรับได้คือ 1 kHz ดังนั้นในการคำนวณในลูปของการตอบสนองต่อแรงต้องน้อยกว่า 1 ms ลงมา ซึ่งก็คือขอบเขตบนของความเป็น Interactive ของการตอบสนองต่อแรง และจากการที่ข้อมูลที่ส่งเป็นตำแหน่ง และการประมวลผลก็ใช้เฉพาะตำแหน่ง จึงเป็นการส่งที่สมบูรณ์ในตัว คือไม่ขึ้นกับข้อมูลก่อนหน้า และขอบเขตการสื่อสารอยู่ภายวง LAN เดียวกัน ดังนั้นการใช้การสื่อสารแบบ Unreliable จึงเป็นตัวเลือกที่เป็นไปได้และดีที่สุดสำหรับกรณีนี้ สำหรับปัญหาของการสื่อสารแบบ Unreliable ก็คือ ไม่ประกันลำดับและการส่งถึง ในส่วนลำดับไม่มีปัญหาเพราะสื่อสารภายในวง LAN เดียวกัน ส่วนการไม่ประกันการส่งถึงก็แก้ได้โดยใช้ Forward error control ซึ่งความจริงเกิดน้อยมากสำหรับการสื่อสารภายในวง LAN เดียวกันในปัจจุบัน

ดังนั้นโครงการเกม RealLife จึงใช้การ Multicast บน UDP/IP ในการสื่อสารตำแหน่งของวงแหวน ส่วนการสื่อสารข้อมูลที่ต้องการ Reliable เช่น ชุดเส้นลวดที่ใช้, พารามิเตอร์ที่ใช้ร่วมกัน หรือสถานะของเกม ก็จะใช้ TCP ในการสื่อสาร ซึ่งเป็นในลักษณะ Client-Server ซึ่ง Server เป็นฝ่ายกำหนดสถานะบางส่วนของเกม เช่น การเลือกชุดเส้นลวด การเริ่มเกม การหยุดเกม การคำนวณคะแนน เป็นต้น.

ข้อมูลที่ส่งเป็นตำแหน่งของวงแหวนเสมือน คือจุดศูนย์กลางแทนที่จะเป็นตำแหน่งของอุปกรณ์ เหตุผลเพราะลักษณะเกมเป็นแบบทำงานร่วมกัน (Collaborative) ที่ไม่มีฝ่ายใดเป็น Master ดังนั้นการคำนวณจึงเป็นแบบกระจายโดยสมบูรณ์ คือในแบบ Global state/Local view (Kurose. 1999) ฉะนั้นการส่งข้อมูลจึงเป็นแบบ Local view ของ Global state ซึ่งก็คือจุดศูนย์กลางของวงแหวนในมุมมองของแต่ละฝ่าย การคำนวณของแต่ละฝ่ายก็คือ การหาตำแหน่งของวงแหวนในมุมมองของฝ่ายตนจากข้อมูลมุมมองตำแหน่งของวงแหวนของฝ่ายอื่น ๆ

สำหรับการแสดงเส้นลวดคัดโค้งจะใช้ Curve แบบ Bezier เพราะสามารถอยู่ในรูปของ Parametric form ได้ ซึ่งจะทำให้หาตำแหน่งเส้นลวดที่จุดต่าง ๆ ได้สะดวกสำหรับการตรวจสอบการชนของห่วง ส่วนการตรวจสอบการชนทำอย่างง่าย ๆ คือการคำนวณหาว่า ศูนย์กลางของห่วงห่างจากเส้นลวด ณ ตำแหน่งที่สัมพันธ์กันมากกว่ารัศมีของห่วงหรือไม่ ถ้ามากกว่าแสดงว่าห่วงชนเส้นลวด และจะแสดงแรงตอบสนองเป็นสัดส่วนตามปริมาณที่เกินนั้น

ส่วนการแสดงกราฟิกจะใช้ตำแหน่งห่วงที่คำนวณได้แสดงห่วงวงกลม ส่วนเส้นลวดใช้การต่อกันของเส้นย่อยแต่ละช่วงที่คำนวณได้จาก Parametric form ของ Bezier.

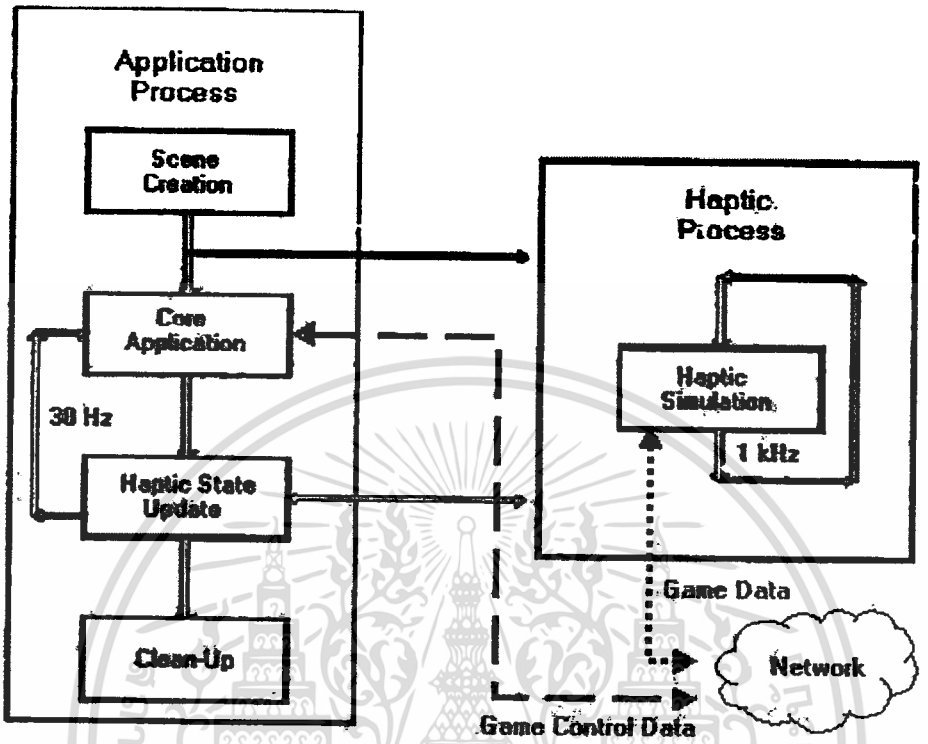
2.3 โครงสร้างโปรแกรม

โครงสร้างของเกม RealLife จากที่ได้กล่าวมาจะประกอบด้วย 2 Process คือ 1) Application Process และ 2) Haptic Process ดังรูปที่ 2.1 คือ

2.3.1 Application Process

Application Process หรือ Application Loop ทำหน้าที่หลัก ๆ คือ

- แสดง GUI และ Function ต่าง ๆ ของ Application
- สร้าง/จัดการ Scene Graph ของ Haptics เพื่อสร้าง Haptic Environment
- เริ่ม/หยุด/จัดการ Haptic Simulation Process
- แสดงกราฟิกจากข้อมูลที่ได้จาก Haptic Process
- สื่อสารข้อมูลควบคุมเกม



รูปที่ 2.1 โครงสร้างของโปรแกรมเกม RealLife.

2.3.2 Haptic Process

Haptic Process หรือ Servo Loop ซึ่งเมื่อสร้าง Scene Node (แต่ละ Application จะมีเพียง 1 Node) ที่มีข้อมูลของ Scene Graph ที่ได้สร้างขึ้นมาแล้ว เมื่อเริ่มการ Simulation จะเป็นการวนที่เรียกว่า Servo Loop ที่อัตรา 1 kHz ซึ่งจะมีการทำงานคือ

- ปรับปรุงสถานะของ Phantom Node ใน Scene Graph ที่แทนตัว Haptic Device.
- ปรับปรุงสถานะแบบ Dynamic ของกลุ่ม Dynamic Nodes
- ตรวจสอบการชนของ Phantom Node กับ Geometry Nodes
- ส่งผลของแรงไปยัง Phantom Node
- ถ้า Application มีการกำหนด Graphic Callback หรือ Event Callback สำหรับ Node ใด ก็จะมีการเตรียมข้อมูลสำหรับ Callback นั้น เมื่อ Node นั้นๆ มีสถานะใหม่
- สื่อสารข้อมูลเกม

สำหรับกรณีเกม RealLife ได้เพิ่มหน้าที่การสื่อสารข้อมูลเกมเข้าไปใน Haptic Process และหน้าที่การสื่อสารข้อมูลควบคุมเกมเข้าไปใน Application Process ซึ่งแตกต่างจากโครงสร้างโปรแกรม

ของ GHOST Application ธรรมดา (SensAble Technologies. 1998) ที่ไม่มีส่วนของการสื่อสารผ่านเครือข่าย

2.4 การออกแบบโปรแกรม

จากลักษณะของเกมดังกล่าวจะพบว่าแยกเป็นระบบย่อยที่สำคัญได้ 3 ระบบคือ 1) ส่วนกราฟิก 2) ส่วนสื่อสาร 3) ส่วนการตอบสนองต่อแรง ที่จะประกอบร่วมกันเป็นเกม โดยใช้ Frame work SDI ของ MFC ในการสร้างโปรแกรม สำหรับรูปแบบสัญลักษณ์ในการออกแบบจะใช้ 1) CRC :Classes/Responsibilities/Collaborators 2) Collaboration Graphs. (Douglas. 1995)

โดย CRC เป็นการออกแบบ Classes หรือ Objects ที่มีในระบบว่าเป็น Class อะไร, สืบทอดมาจาก Class อะไร, มีหน้าที่อะไรบ้าง, และในหน้าที่นั้น ๆ ต้องทำงานร่วมกับ Classes หรือ Objects ใดบ้าง ซึ่งชื่อ Class จะอยู่ด้านบนซ้ายของตาราง ตามด้วยเครื่องหมาย Colon และชื่อ Class ที่สืบทอดมา (ถ้ามีการสืบทอด) ส่วนด้านบนขวาแสดงประเภทของ Class ว่าเป็น Abstract Class หรือ Concrete Class และเป็น Subsystem หรือไม่ โดยภายในตารางแยกเป็นด้านซ้ายแสดงหน้าที่ของ Class นั้นซึ่งถูกกำกับด้วยหมายเลข ส่วนด้านขวาเป็น Classes ที่ทำงานร่วมเพื่อให้หน้าที่ในข้อนั้น ๆ สำเร็จ (ถ้ามี)

ส่วน Collaboration Graphs เป็นการสร้าง Diagrams เพื่อให้เห็นความสัมพันธ์ของ Class ที่มีในระบบให้ชัดเจนขึ้น โดยแสดงเป็นกล่องสี่เหลี่ยมที่แสดงชื่อ Class, ที่ขอบของกล่องสี่เหลี่ยมแสดงหมายเลขซึ่งตรงกับหมายเลขหน้าที่ใน CRC ของ Class นั้น ๆ แล้วจะเชื่อมโยงด้วยลูกศรไปยังหมายเลขหน้าที่ของ Class ที่ทำงานร่วมกันเพื่อให้หน้าที่นั้นสำเร็จ โดยทิศของหัวลูกศรหมายถึงการใช้บริการจากหน้าที่ข้อที่ชี้ของ Class ที่ถูกชี้อยู่

2.4.1 CRC

ตารางที่ 2.1 CRC ของ CGameApp

CGameApp:CwinApp	Concrete
1.Create SDI Framework	
2.Peek Messages	
3.Route Commands	CMainFrame, CGameDoc, CGameView
4.Render OpenGL When No Message	CGameGL

จาก CRC, CGameApp มีหน้าที่ 1) สร้าง SDI Framework คือ ตัว Mainframe Window (CMainFrame), ตัว Document (CGameDoc), และตัว View Window (CGameView) ตามโครงสร้างของ MFC SDI Framework 2) ทำหน้าที่อ่านและจัดการ Windows Messages 3) ทำการกำหนดเส้นทาง Commands และส่ง Commands ภายในตัว Framework (คือ CGameApp, CMainFrame, CGameDoc, CGameView) 4) เรียกให้ CGameGL แสดงกราฟิกเมื่อไม่มี Windows Messages ให้ Process (Idle).

ตารางที่ 2.2 CRC ของ CMainFrame

CMainFrame:CframeWnd	Concrete
1.Manage View, Menu, Toolbar, Status bar 2.Handle Commands	

จาก CRC, CMainFrame มีหน้าที่ 1) สร้างและจัดการ Menu bar, Toolbar, Status bar และจัดการ View 2) ทำหน้าที่ต่าง ๆ ตาม Commands หรือ Windows Messages.

ตารางที่ 2.3 CRC ของ CGameDoc

CGameDoc:Cdocument	Concrete
1.Load/Save Game Setting 2.Handle Command	CGameGL, CGameHaptic, CGameNet

จาก CRC, CGameDoc มีหน้าที่ 1) อ่านหรือบันทึกข้อมูลของเกมจากระบบย่อยต่าง ๆ จากไฟล์หรือลงไฟล์ 2) ทำหน้าที่ต่าง ๆ ตาม Commands หรือ Windows Messages.

ตารางที่ 2.4 CRC ของ CGameView

CGameView:Cview	Concrete
1.Communicate Other Peer	CGameNet
2.Select Curve Set	CGameHaptic
3.Start/Stop Game	CGameHaptic
4.Load/Save Setting	CGameDoc
5.Handle Command	
6.Render OpenGL on View Area	CGameGL
7.Resize View	CGameGL

จาก CRC, CGameView มีหน้าที่ 1) เป็น Graphic User Interface (GUI) เพื่อใช้ในการติดต่อกับผู้เล่นฝ่ายอื่น โดยใช้บริการของ CGameNet Subsystem (ในส่วนของบริการ TCP) ในการสื่อสารข้อมูลควบคุมเกม เช่น การเริ่มหรือหยุดเล่นเกม 2) เป็น GUI ในการเลือกชุดของเส้นลวดดัดโค้งเสมือนที่จะ Simulate โดย CGameHaptic Subsystem 3) เป็น GUI ในการเริ่มหรือหยุดเกมซึ่งจะไปสั่งให้ CGameHaptic ซึ่งควบคุม Haptic Process ทำการเริ่มหรือหยุดการ Simulations ด้วย 4) เป็น GUI ในการกำหนดข้อมูลเพื่อบันทึก หรือเลือกไฟล์ข้อมูลเพื่ออ่าน ซึ่งใช้บริการ CGameDoc ในการอ่านหรือบันทึกข้อมูลจากไฟล์ 5) ทำหน้าที่ต่าง ๆ ตาม Commands หรือ Windows Messages 6) เป็นที่แสดงกราฟิก โดยใช้บริการของ CGameGL ในการเตรียมให้ Window ของตัวเอง (CGameView) สามารถ Render กราฟิกโดยฟังก์ชันของ OpenGL ได้ 7) เมื่อมีการเปลี่ยนแปลงขนาดของ Window จะบอกไปยัง CGameGL ด้วยเพื่อคำนวณหรือจัดกราฟิกให้สัมพันธ์กันกับขนาดที่เปลี่ยนแปลง

ตารางที่ 2.5 CRC ของ CGameGL

CGameGL:Cobject	Concrete Subsystem
1.Make Window Support OpenGL	CGameView
2.Set OpenGL State	
3.Render OpenGL	CGameHaptic
4.Load/Save Setting	CGameDoc

จาก CRC, CGameGL Subsystem มีหน้าที่ 1) ทำให้ CGameView สามารถ Render ฟังก์ชันของ OpenGL ได้ 2) กำหนดค่าต่าง ๆ ที่ใช้ในการแสดงกราฟิก 3) เป็นคนสั่งให้ Render OpenGL 4) อ่านหรือบันทึกข้อมูลของระบบย่อยโดยใช้บริการจาก CGameDoc.

ตารางที่ 2.6 CRC ของ CGameNet

CGameNet:Cobject	Concrete Subsystem
1. Support TCP Communication	
2. Support Multicast Communication	
3.Load/Save Setting	CGameDoc

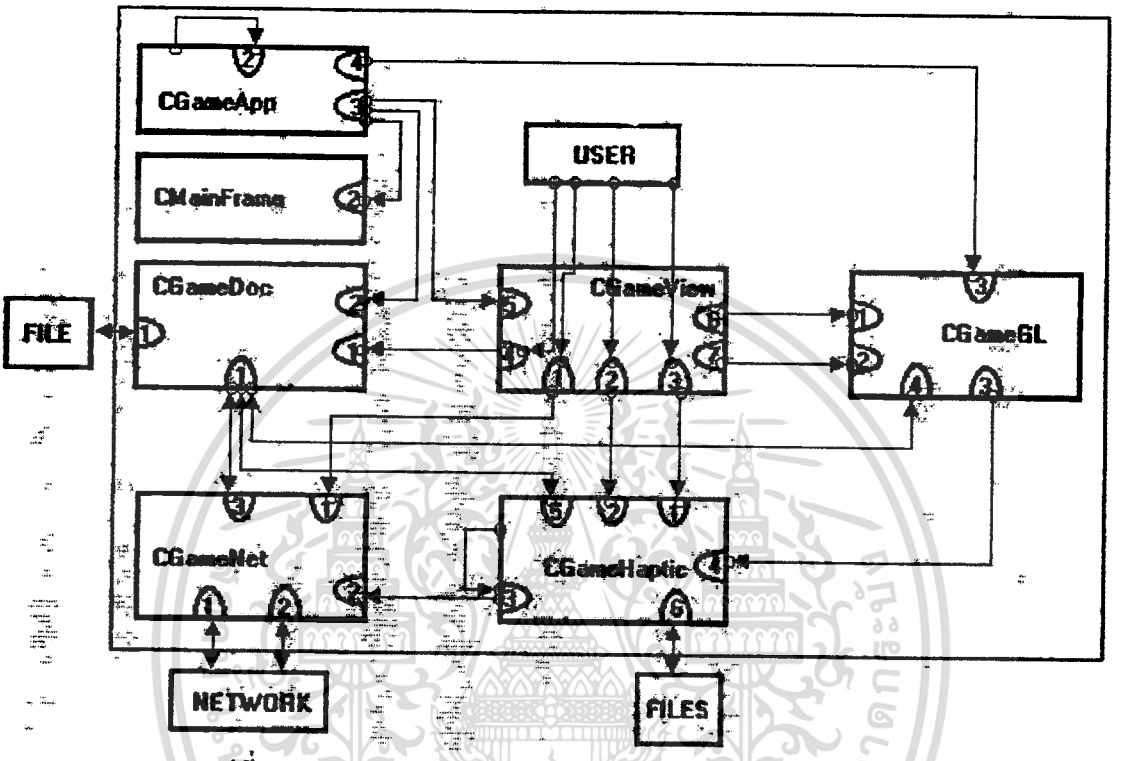
จาก CRC, CGameNet Subsystem มีหน้าที่ 1) ให้บริการการสื่อสารแบบ TCP 2) ให้บริการการสื่อสารแบบ Multicast 3) อ่านหรือบันทึกข้อมูลของระบบย่อยโดยใช้บริการจาก CGameDoc.

ตารางที่ 2.7 CRC ของ CGameHaptic

CGameHaptic:Cobject	Concrete Subsystem
1.Control Haptic Process	
2.Manage Haptic Scene Graph	
3.Simulate Ring-Curve Effect	CGameNet
4.Handle Haptic Scene Graph Graphics	
5.Load/Save Setting	CGameDoc
6.Load Curve Set	

จาก CRC, CGameHaptic Subsystem มีหน้าที่ 1) ควบคุม Haptic Process 2) สร้างและจัดการ Haptic Scene Graphs 3) เป็นส่วนที่ใช้ในการ Simulate การควบคุมห่วงกลมเสมือนร่วมกันของสองฝ่ายผ่านการสื่อสารแบบ Multicast และการจำลองการตอบสนองต่อแรงของเส้นลวดดัดโค้งเสมือนที่มีต่อห่วงกลมเสมือน 4) แสดงกราฟิกในส่วนของ Haptic Scene Graph 5) อ่านหรือบันทึกข้อมูลของระบบย่อยโดยใช้บริการจาก CGameDoc 6) อ่านข้อมูลของ Bezier Control Points จากไฟล์ที่จะจำลองเป็นเส้นลวดดัดโค้งเสมือน

2.4.2 Collaboration Graphs



รูปที่ 2.2 Collaboration Graphs ของโปรแกรมเกม RealLife.

จาก Collaboration Graphs แสดงถึงความสัมพันธ์ของ Classes ต่าง ๆ หรือระบบย่อยต่าง ๆ ภายในโปรแกรมเกม RealLife โดยโปรแกรมใช้ MFC SDI Framework เป็นพื้นฐาน ซึ่งประกอบด้วย CGameApp (Application), CMainFrame (MainFrame Window), CGameDoc (Document), CGameView (View Window) และแบ่งเป็นระบบย่อยตามหน้าที่คือ CGameGL (ระบบย่อยกราฟิก) ทำหน้าที่สนับสนุนทางด้านการแสดงกราฟิกด้วย OpenGL, CGameNet (ระบบย่อยสื่อสาร) ทำหน้าที่สนับสนุนในการสื่อสารด้วย TCP และ Multicast (บน UDP/IP), CGameHaptic (ระบบย่อยการตอบสนองต่อแรง) ทำหน้าที่ในการควบคุม Haptic Process และการจำลองการบังคับห่วงกลมเสมือนร่วมกันของสองฝ่าย และการจำลองการตอบสนองต่อแรงของเส้นลวดคัดโค้งเสมือนที่มีต่อห่วงกลมเสมือน โดยความสัมพันธ์ตาม Diagram แสดงถึงการให้บริการในหน้าที่และการใช้บริการในหน้าที่ต่าง ๆ ของแต่ละ Class เพื่อให้หน้าที่นั้น ๆ สำเร็จลง ตามที่ได้อธิบายแล้วใน CRC ของแต่ละ Class.

2.5 สรุป

จากการออกแบบในเบื้องต้นด้วย CRC และ Collaboration Graphs ทำให้เห็นโครงสร้างของเกม RealLife ในเบื้องต้น ซึ่งประกอบด้วยส่วนจาก MFC SDI Framework คือ CGameApp, CMainFrame, CGameDoc, และ CGameView ในส่วนที่เหลือแบ่งงานออกเป็นทางด้านกราฟิก, การตอบสนองต่อแรง, และการสื่อสารผ่านเครือข่าย คือ CGameGL, CGameHaptic, และ CGameNet ตามลำดับ โดยทำเป็นระบบย่อย ซึ่งจะอธิบายรายละเอียดในแต่ละด้านในแต่ละบทต่อไป

บทที่ 3

ระบบย่อยกราฟิก

3.1 ระบบย่อยกราฟิก

GHOST SDK ที่ใช้ในการ Simulation การตอบสนองต่อแรงที่จะนำมาพัฒนาเกม RealLife นั้น ไม่มีฟังก์ชันที่ช่วยในการแสดงภาพกราฟิก แต่ก็มีกลไก Callback ในการ Update ข้อมูลเพื่อให้ Application Process นำข้อมูลนั้นไปแสดงหรือคำนวณเพื่อตอบสนองต่อไป เช่นการแสดงภาพกราฟิกที่สัมพันธ์กัน เป็นต้น ดังนั้นระบบย่อยกราฟิกจึงทำหน้าที่รองรับการแสดงภาพกราฟิกสำหรับเกม RealLife ซึ่งใช้ Graphics Library คือ OpenGL ให้ได้กับ Application Framework ที่ใช้ คือ MFC SDI โดยหน้าที่หลักคือการสร้าง OpenGL Rendering Context จาก CGameView เพื่อแสดงกราฟิกที่ Render โดยฟังก์ชันของ OpenGL, รองรับการย่อขยายพื้นที่ที่แสดงกราฟิก, รองรับการจัดและเปลี่ยนสำหรับมุมมองและแสง เป็นต้น

3.2 แนวคิดระบบย่อยกราฟิก

จาก CRC และ Collaboration Graphs ของระบบรวมทำให้เห็นหน้าที่ของระบบย่อยกราฟิกที่มีและความสัมพันธ์กับส่วนอื่น ๆ คือ 1) การทำให้ Device Context ของ CGameView สามารถ Render กราฟิกจากฟังก์ชันของ OpenGL 2) การกำหนดค่าทางกราฟิกต่าง ๆ ของ OpenGL เช่น มุมมอง, แสง เป็นต้น 3) เป็นคนเรียกหรือควบคุมการ Render จากส่วนอื่น ๆ ที่ต้องการ ในกรณีเกม RealLife คือ ระบบย่อยตอบสนองต่อแรงที่ต้องการ Render กราฟิกของห่วงกลมเสมือนและลวดดัดโค้งเสมือน

จากหน้าที่หลักของระบบย่อยที่กล่าวมาแล้ว ทำให้พบนามที่น่าจะเป็น Object ได้ (Douglas. 1995) คือ Rendering Context (CGDC) ทำหน้าที่สร้าง OpenGL Rendering Context และควบคุมการ Render ใน Context นี้ โดย State ต่าง ๆ ของ OpenGL กำหนดโดยตัวของระบบย่อย คือ CGameGL แต่ถ้าเป็น State ที่ยุ่งยากหรือใช้บ่อย ๆ ก็จะกำหนดผ่าน Graphic Objects ซึ่งในที่นี้คือ Camera (CGCam) และ Light (CGLight) เช่น CGCam อาจจะมีการหมุน, เปลี่ยนมุมมอง ซึ่งมีความซับซ้อน หรือ CGLight ที่อาจจะช่วยในการจัดแสง เป็นต้น

ส่วนตัวระบบย่อยคือ CGameGL ทำหน้าที่ประสานงาน เช่น เมื่อ CGameView มีการเปลี่ยนขนาด Window, CGameGL ก็จะประสานให้ CGCam มีค่าที่เหมาะสมเพื่อให้ตรงกับขนาดของ Window จริงเป็นต้น และ CGameGL ยังทำหน้าที่คอยเรียกกลับไปยังส่วนอื่น ๆ ที่ต้องการ Render กราฟิกโดยฟังก์ชันของ OpenGL ด้วย โดยลูปในการ Render อาจเลือกได้ 2 ประเภท คือ การใช้ Window Timer หรือการใช้ OnIdle เมื่อไม่มีการ Process Window Message. แต่สำหรับเกม RealLife ใช้กรณีที่ 2 โดยเกี่ยวไว้ (Hook) ใน Message Loop ของ CGameApp.

3.3 CRC

ตารางที่ 3.1 CRC ของ CGameGL

CGameGL:Cobject	Concrete Subsystem
1.Make Window Support OpenGL 2.Set OpenGL State 3.Render OpenGL 4.Load/Save Setting	CGameView CGameHaptic CGameDoc

จาก CRC, CGameGL Subsystem มีหน้าที่ 1) ทำให้ CGameView สามารถ Render ฟังก์ชันของ OpenGL ได้ 2) กำหนดค่าต่าง ๆ ที่ใช้ในการแสดงกราฟิก 3) เป็นคนสั่งให้ Render OpenGL 4) อ่านหรือบันทึกข้อมูลของระบบย่อยโดยใช้บริการจาก CGameDoc.

ตารางที่ 3.2 CRC ของ CGDC

CGDC	Concrete
1.Make OpenGL Rendering Context 2.Initialize Graphic Objects 3.Render on Rendering Context	CGCam, CGLight

จาก CRC, CGDC มีหน้าที่ 1) สร้าง OpenGL Rendering Context 2) กำหนดค่าเริ่มต้นของ Graphic Objects ในที่นี้คือ CGCam และ CGLight 3) Render OpenGL บน OpenGL Rendering Context ที่สร้าง.

ตารางที่ 3.3 CRC ของ CGCam

CGCam	Concrete
1.Set Camera Position 2.Cnange, Slide Camera Posiuon 3.Row, Pitch, Yaw Camera	

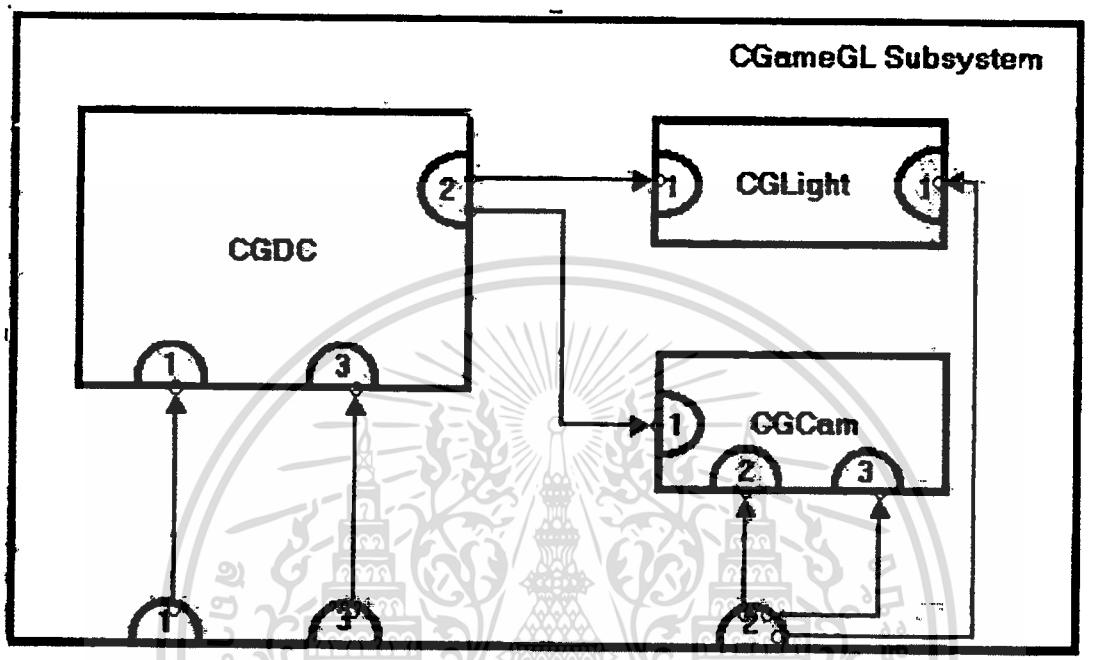
จาก CRC, CGCam มีหน้าที่ 1) กำหนดตำแหน่งและการจัดวางของมุมมอง (กล้อง) 2) ช่วยในการเปลี่ยนตำแหน่งกล้อง 3) ช่วยในการหมุนกล้องในแบบ Row, Pitch, และYaw.

ตารางที่ 3.4 CRC ของ CGLight

CGLight	Concrete
1.Set Light Parameters	

จาก CRC, CGLight มีหน้าที่ 1) ช่วยในการกำหนดพารามิเตอร์ของแสง.

3.4 Collaboration Graphs



รูปที่ 3.1 Collaboration Graphs ของ CGameGL Subsystem

จาก Collaboration Graphs ของระบบย่อยกราฟิก แสดงถึงความสัมพันธ์ภายในของระบบย่อย ซึ่งประกอบด้วย Classes ต่าง ๆ คือ CGDC, CGCam, และ CGLight ซึ่งจาก Diagram และ CRC จะเห็นว่าระบบย่อยกราฟิกทำ 4 หน้าที่ คือ

ในข้อ 1 คือ ทำให้ Window มีความสามารถในการ Render OpenGL ซึ่งทำได้โดยไปใช้บริการของ CGDC ในข้อ 1 คือ สร้าง OpenGL Rendering Context และ CGDC ก็ยังทำหน้าที่ในการกำหนดค่าเริ่มต้นของ Graphic Objects ซึ่งในที่นี้คือ CGCam และ CGLight ด้วย

ในข้อ 2 คือ กำหนดค่าต่าง ๆ ที่ใช้ในการแสดงกราฟิก ซึ่งจะไปใช้บริการในข้อ 2 หรือ 3 ของ CGCam ในการกำหนดค่าต่าง ๆ ที่เกี่ยวกับมุมมอง(กล้อง) และใช้บริการในข้อ 1 ของ CGLight ในการกำหนดค่าพารามิเตอร์ทางแสง

ในข้อ 3 คือ Render OpenGL ซึ่งจะไปใช้บริการในข้อ 3 ของ CGDC คือ) Render OpenGL บน OpenGL Rendering Context ที่สร้าง

ในข้อ 4 คือ อ่านหรือบันทึกข้อมูลของระบบย่อย โดยทำงานร่วมกับ CGameDoc ซึ่งอยู่ภายนอกของระบบย่อยกราฟิกและหน้าที่นี้ไม่มีการทำงานร่วมกับ Class ใดในระบบย่อยกราฟิกจึงไม่มีการเขียนหมายเลขและความสัมพันธ์ใน Diagram.

3.5 การแสดงภาพกราฟิกด้วย OpenGL

OpenGL เริ่มแรกพัฒนาโดย Silicon Graphics Incorporated. (SGI) เป็นระบบกราฟิกที่มีประสิทธิภาพสูง, ไม่ขึ้นกับระบบ Window ใด เหมาะสำหรับการแสดงภาพกราฟิก 3 มิติ และยังสามารถใช้สำหรับภาพ 2 มิติ และรูปภาพได้ดีอีกด้วย ปัจจุบันอยู่ภายใต้การควบคุมของ OpenGL Architectural Review Board (ARB) ซึ่งประกอบด้วยบริษัทชั้นนำทางด้านคอมพิวเตอร์กราฟิกหลายแห่ง

3.5.1 สถาปัตยกรรมของ OpenGL

OpenGL State

OpenGL ถูกออกแบบมาให้เป็นลักษณะ State ที่คอยควบคุมการ Render ความหมายของ State ในที่นี้คือ เมื่อเรากำหนดรูปทรงที่จะวาดเข้าไป (ในรูปของ Vertices และ Image) ผลที่ออกมาขึ้นกับว่าเราได้กำหนด State นั้นมาก่อนอย่างไร ซึ่งก็คือ การกำหนดสีอะไร, Model ของแสงเป็นอย่างไร หรือ Perspective เป็นอย่างไรมาก่อนเป็นต้น โดยฟังก์ชันที่ใช้จะเป็นฟังก์ชันง่าย ๆ ไม่มีพารามิเตอร์มากนักแต่เมื่อรวมกันกับ State ที่กำหนดมาก่อนก็ทำให้เกิดเป็นภาพที่ซับซ้อนขึ้น

OpenGL Data Types

OpenGL ใช้ Data Types ของตัวเองคือขึ้นต้นด้วย GL ตามด้วยชนิดของ Type นั้น ซึ่งมีดังนี้

ตารางที่ 3.5 ชนิดข้อมูลที่ใช้ใน OpenGL

Suffix	Data type	Typical C or C++ type	OpenGL type name
b	8-bit integer	signed char	GLbyte
s	16-bit integer	short	GLshort
i	32-bit integer	int or long	GLint, GLsizei
f	32-bit floating point	float	GLfloat, GLclampf
d	64-bit floating point	double	GLdouble, GLclampd
ub	8-bit unsigned number	unsigned char	GLubyte, GLboolean
us	16-bit unsigned number	unsigned short	GLushort
ui	32-bit unsigned number	unsigned int or unsigned long	GLuint, GLenum, GLbitfield

สาเหตุเพราะแต่ละระบบกำหนดขนาดตัวแปรในภาษา C และ C++ ไม่เหมือนกัน เช่นบางระบบกำหนด int เป็น 16-bit ส่วนอีกระบบกำหนดเป็น 32-bit เป็นต้น จึงแก้ปัญหาโดยใช้ Data type ของ OpenGL ที่กำหนดขึ้นมาเอง และเพื่อให้สามารถ port ไปใช้ได้ในทุกระบบ

3.5.2 ฟังก์ชันในการ Render ของ OpenGL

OpenGL เป็น Library ระดับต่ำเมื่อเทียบกับพวก Open Inventor ดังนั้นการสร้างกราฟิกโดย OpenGL จะเกิดจากการสร้างรูปทรงเรขาคณิตและภาพง่าย ๆ โดยฟังก์ชันพื้นฐาน เพื่อประกอบเป็นภาพที่ซับซ้อนขึ้น และเมื่อแสดงผลผ่าน Mode ต่าง ๆ ก็จะเกิดเป็นภาพตามต้องการ โดยฟังก์ชันหลัก ๆ คือ

<u>การสร้าง Geometric Primitives</u>

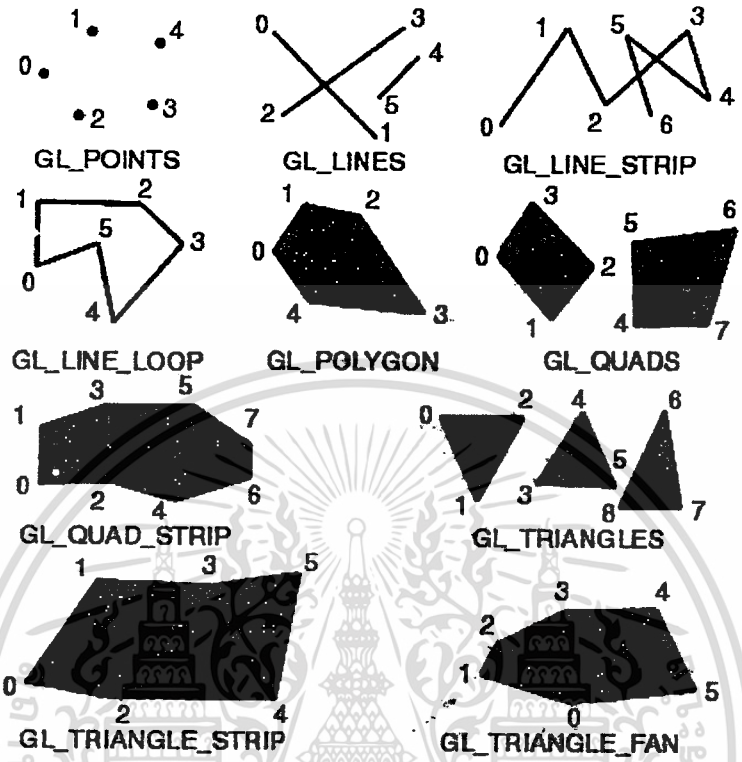
Geometric Primitives เป็นรูปร่าง ๆ ง่าย เช่น จุด, เส้น, สามเหลี่ยม เป็นต้น สร้างได้จากการกำหนด Vertices (จุดยอด) และลำดับการเรียงของ Vertices ซึ่ง Vertices กำหนดโดยฟังก์ชัน เช่น glVertex3f() ซึ่งโดยปกติฟังก์ชันใน OpenGL จะออกแบบไว้รองรับพารามิเตอร์ได้หลายตัวแปรและหลายประเภทของตัวแปร ตัวอย่างเช่น glVertex3f(...) สิ่งที่สังเกตได้คือ ตัวเลข 3 หมายถึง 3 มีการรับพารามิเตอร์ 3 ตัว ส่วนตัวอักษรท้ายสุดเป็นการบอกว่าพารามิเตอร์นั้นเป็นตัวแปรแบบใด เช่น f คือ GLfloat อีกตัวอย่างหนึ่งคือ glVertex2i(...) เป็นการบอกว่ารับตัวแปรประเภท GLint จำนวน 2 ตัวเป็นต้น ซึ่งประเภทของตัวแปรเป็นไปตามตาราง 3.1

เรากำหนดเซตของ Vertices เพื่อประกอบเป็น Primitive ต่าง ๆ โดยเริ่มที่กำหนดประเภทของ Primitive ซึ่งเป็นพารามิเตอร์ของ glBegin(...) ตามด้วยชุดของ Vertices แล้วปิดท้ายด้วย glEnd()

ตัวอย่างเช่น การสร้าง Primitive ชนิด Polygon โดยใช้ Vertices 3 มิติ ประเภท Float รวมทั้งหมด 4 จุด เพื่อสร้างรูปสี่เหลี่ยมสามารถเขียนได้ดังนี้

```
glBegin (GL_POLYGON);
glVertex3f (1.0, 1.0, 1.0);
glVertex3f (1.0, -1.0, 1.0);
glVertex3f (1.0, -1.0, -1.0);
glVertex3f (1.0, 1.0, -1.0);
glEnd ();
```

โดย Primitive ที่มีใน OpenGL แสดงดังรูปที่ 3.2



รูปที่ 3.2 รูปทรงพื้นฐานของ OpenGL

<u>การกำหนด State</u>

ส่วนใหญ่จะใช้ glEnable() และ glDisable() ในการ Enable และ Disable Mode ต่าง ๆ เช่น glEnable(GL_LIGHTING) หมายถึงการ Enable Lighting Model เป็นต้น

ส่วนการกำหนดค่าของ State ก็เรียกใช้ตามฟังก์ชันนั้น ๆ เช่น การกำหนดสีใช้ glColor3f (...) หรือการกำหนด Clear Color ก็ใช้ glClearColor(...) เป็นต้น

3.6 สรุป

จาก CRC และ Collaboration Graphs ทำให้มองเห็นภาพรวมและหน้าที่ของระบบย่อยกราฟิกได้ชัดเจนขึ้น และจากหลักการคร่าว ๆ ของการแสดงกราฟิกด้วย OpenGL ที่ได้เสนอไป ก็น่าจะเพียงพอในระดับหนึ่ง สำหรับรายละเอียดในการ Implement ในบางส่วน เช่น การทำให้ Device Context ของ Window สามารถ Render OpenGL ดูได้จาก MSDN หรือเทคนิคการจัดการกล้องรวมทั้งทฤษฎีทางกราฟิกดูได้จาก (Hill. 2001).

บทที่ 4

ระบบย่อยสื่อสาร

4.1 ระบบย่อยสื่อสาร

จากแนวคิดและการออกแบบในบทการออกแบบ จะพบว่าข้อมูลที่ใช้ในการสื่อสารภายในเกม RealLife มี 2 ประเภทคือ 1) ข้อมูลเกม 2) ข้อมูลควบคุมเกม และแต่ละประเภทก็มีลักษณะที่ต้องการในการสื่อสารที่ต่างกัน คือ ข้อมูลเกมหรือ Local View of Global State (Kurose. 1999) ของจุดศูนย์กลางวงแหวนเสมือนที่มีการส่งแลกเปลี่ยนไปยังผู้เล่นอื่น ๆ เพื่อให้ผู้เล่นอื่น ๆ นำไปคำนวณ Global State ของตัวเองนั้น ตามแนวคิดและจากการที่ Haptic Process มีลูปในการคำนวณการตอบสนองต่อแรงที่ 1 kHz และความจริงที่ว่าการเข้าถึงข้อมูลภายใน Haptic Process ที่ไม่ผ่านลูปคือ ไม่ผ่านการเรียกโดยตัวลูปเองหรือ Callback โดยตัวลูปนั้นไม่รับรองว่าข้อมูลนั้นเป็นของลูปนั้น หรือ ไม่รับรองว่าถูกต้อง (Sensable Technologies. 1998) ทำให้การเข้าถึงข้อมูลภายใน Haptic Process เท่ากับ 1 kHz หรือทุก ๆ 1 ms. นับเป็นอุปสรรคสำคัญในการที่จะทำให้เกมตอบสนองอย่างราบเรียบ เพราะการตอบสนองต่อแรงอยู่ที่ระดับ 1 kHz เท่ากับอัตราในการเข้าถึงข้อมูลของ Haptic Process เพื่อสื่อสารไปยังผู้เล่นอื่น ๆ ทำให้การ Implement ตามแนวคิด Transmission Control Mechanism ของ (Kurose. 1999) เป็นไปได้เพียงบางส่วนเท่านั้น.

ข้อมูลเกมที่สื่อสาร คือ Local View of Global State ของจุดศูนย์กลางวงแหวนเสมือนมีลักษณะในการส่งและคำนวณที่สมบูรณ์ในตัว คือไม่ขึ้นกับข้อมูลก่อนหน้า และจากการที่มีการนำไปคำนวณแบบ Real-time ทำให้มีความต้องการในการส่งที่รวดเร็วที่สุด, Overhead น้อยที่สุดเพราะมีการส่งถี่, และข้อมูลที่มาถึงช้ากว่า Play-out ของตัวมันจะไม่มีประโยชน์ ดังนั้นการสื่อสารแบบ Unreliable เช่น UDP ก็น่าจะเหมาะสมกว่า และจากการออกแบบโดยใช้หลักการคำนวณแบบกระจายโดยสมบูรณ์ซึ่งแลกเปลี่ยนข้อมูลแบบ Local View of Global State ทำให้การส่งในลักษณะ Multicast น่าจะเป็นตัวเลือกที่เหมาะสมมากกว่า Peer-to-peer ถึงแม้จะมีเพียง 2 ผู้เล่น เพราะ Implement ง่ายกว่าและยังสามารถขยายจำนวนผู้เล่นหรืออาจจะเพิ่มบุคคลที่สาม เพื่อเฝ้าดูหรือวิเคราะห์การสื่อสารเพื่อพัฒนาต่อไป เป็นต้น

ส่วนข้อมูลควบคุมเกม เช่นในที่นี้คือ ข้อมูลควบคุมการเริ่มหยุดเกม, ชุดของเส้นลวดเสมือนที่ใช้, ข้อมูลคะแนน เป็นต้น มีความต้องการการสื่อสารที่ Reliable สูงกว่าข้อมูลเกมธรรมดา และมีลักษณะคำนวณที่จุดใดจุดหนึ่งมากกว่าการกระจายกันคำนวณ จึงใช้ TCP ในการสื่อสารตามปกติ

4.2 แนวคิดระบบย่อยสื่อสาร

จากการออกแบบ ระบบย่อยสื่อสารทำ 2 หน้าที่ที่สำคัญ คือ

1) สนับสนุนการสื่อสารแบบ TCP
2) สนับสนุนการสื่อสารแบบ Multicast (บน UDP/IP)

4.2.1 การสนับสนุนการสื่อสารแบบ TCP

ในการ Implement การสื่อสารแบบ TCP ก็คือต้อง Implement ทั้งฝ่าย Client และ Server ถ้าแยก Implement เป็นคนละโปรแกรม จะทำให้ยุ่งยากและซับซ้อน ดังนั้นจึงรวมทั้ง Client และ Server ให้ใช้ Code เดียวกันให้มากที่สุด และการที่ใช้ Windows Socket ในการ Implement จะมี Socket 3 ประเภทที่เกี่ยวข้อง คือ Listening Socket, Server Socket, และ Client Socket โดยมีลำดับการทำงานตามตาราง

ตารางที่ 4.1 ลำดับการทำงานของ Window Socket แบบ Client-Server

Server	Client
1. Create Listening Socket	1. Create Client Socket
2. Listening Socket Wait for Connections	
	3. Client Socket Request Connection
4. If not Accept this Client go to Step 2 If Accept this Client so Create New Server Socket for this Client and then go to Step 2	
5. Server Socket Communicate with Client Socket	5. Client Socket Communicate with Server Socket

โดยในการออกแบบจะให้ Object เดียวทำหน้าที่ได้ทั้ง 3 แบบดังกล่าว แล้วอาศัยการสร้างแบบ Dynamics ในส่วน Application Protocol จะสนับสนุนจนถึงเฉพาะการแยก Message ของ Client

Socket กับ Server Socket และการแยกประเภท Message เท่านั้น ส่วนการตอบสนองตาม Application Protocol จะให้เป็นหน้าที่ของ CGameView เพื่อเป็นการแบ่งหน้าที่ที่ชัดเจน และเพราะ CGameView มีส่วนในการควบคุมเกมมากที่สุด ในการแยกประเภท Message จะใช้การเข้า Code ในส่วนหน้าของ Message ในการแยกประเภท และสร้างเป็น Message Object ตามที่กำหนดไว้ใน Message Factory แล้วส่ง Message Objects ไปยัง CGameView โดยใช้ User Window Message. นี่คือแนวคิดโดยสรุปของการสนับสนุนการสื่อสารแบบ TCP

4.2.2 การสนับสนุนการสื่อสารแบบ Multicast (บน UDP/IP)

ในส่วนการสนับสนุนการสื่อสารแบบ Multicast ก็ใช้ Window Socket แต่ต่างจาก TCP ตรงที่ไม่มีการ Connect กันก่อน และใช้ Group Address แทน โดยมี Socket หนึ่งเป็นตัว Join Group และคอยรับ Multicast Message. ส่วนอีก Socket หนึ่งทำหน้าที่ส่ง Message ไปยัง Group Address นั้นเมื่อต้องการ

4.3 CRC

ตารางที่ 4.2 CRC ของ CGameNet

CGameNet:Cobject	Concrete Subsystem
1.Support TCP Communication 2.Support Multicast Communication 3.Load/Save Setting	CGameDoc

จาก CRC, CGameNet Subsystem มีหน้าที่ 1) ช่วยสนับสนุนในการสื่อสารแบบ TCP 2) ช่วยสนับสนุนในการสื่อสารแบบ Multicast 3) อ่านหรือบันทึกข้อมูลของระบบย่อยโดยใช้บริการจาก CGameDoc.

4.3.1 CRC ของส่วน TCP

เนื่องจากในส่วนของ TCP ตามที่อธิบายมาข้างต้น ส่วนใหญ่มีการทำงานแบบ Dynamics ซึ่งการใช้ CRC และ Collaboration Graphs ตาม (Douglas. 1995) นั้นไม่สามารถอธิบายได้ทั้งหมด ดังนั้นจึงใช้การเขียน CRC และ Collaboration Graphs เพียงบางส่วนที่พอจะทำได้ ร่วมกับการอธิบายประกอบ เพราะยังไม่มีสัญลักษณ์ในการออกแบบใดที่อธิบายการทำงานแบบ Dynamics ได้

ตารางที่ 4.3 CRC ของ CCommSocket

CcommSocket:Csocket	Concrete
1.Connect/Disconnect Server	
2.Wait for Connection from Client	
3.Communicate with associate Socket	CCommMsg
4.Separate and Create Message Object	CCommMsgFactory
5.Send Message Object to Application Protocol Handlers	CCommMsg

จาก CRC, CCommSocket ซึ่งก็คือ Class ที่ทำหน้าที่แทน Socket ทั้ง 3 ประเภทที่กล่าวมาในตอนต้น มีหน้าที่ 1) ทำหน้าที่ Server/Client Socket คือ Connect/Disconnect 2) ทำหน้าที่ Listening Socket ในการคอย Connection 3) ติดต่อสื่อสารระหว่างกัน (Client Socket-Server Socket) โดย Encode เป็น Message Object (คือ Class ที่สืบทอดจาก CCommMsg) ในการส่งข้อมูล 4) ในกรณีที่รับ Message Packet มาให้แยกและสร้างเป็น Message Object ขึ้นมา โดยใช้ Message Factory คือ CCommMsgFactory ในการสร้าง Message Object ตามที่ Encode มาใน Message Packet 5) ส่ง Message Object ไปให้ส่วนที่รับผิดชอบ Application Protocol จัดการโดยใช้ Window Message.

ตารางที่ 4.4 CRC ของ CCommMsg

CcommMsg:Cobject	Concrete
1.Set Message Object Code	
2.Serialize Data	

จาก CRC, CCommMsg ซึ่งเป็น Base Class ของ Message Object มีหน้าที่ 1) กำหนดรหัสของ Message Object แต่ละชนิด 2) อ่านเขียนข้อมูลของตัวเอง.

ตารางที่ 4.5 CRC ของ CCommMsgFactory

CcommMsgFactory	Concrete
1.Register Message Object Type Suite	
2.Create Message Object from Type	

จาก CRC, CCommMsgFactory ซึ่งเป็น Message Factory มีหน้าที่ 1) กำหนดประเภทของ Message Object ที่ใช้ 2) สร้าง Message Object ตามประเภท.

4.3.2 CRC ของส่วน Multicast (บน UDP/IP)

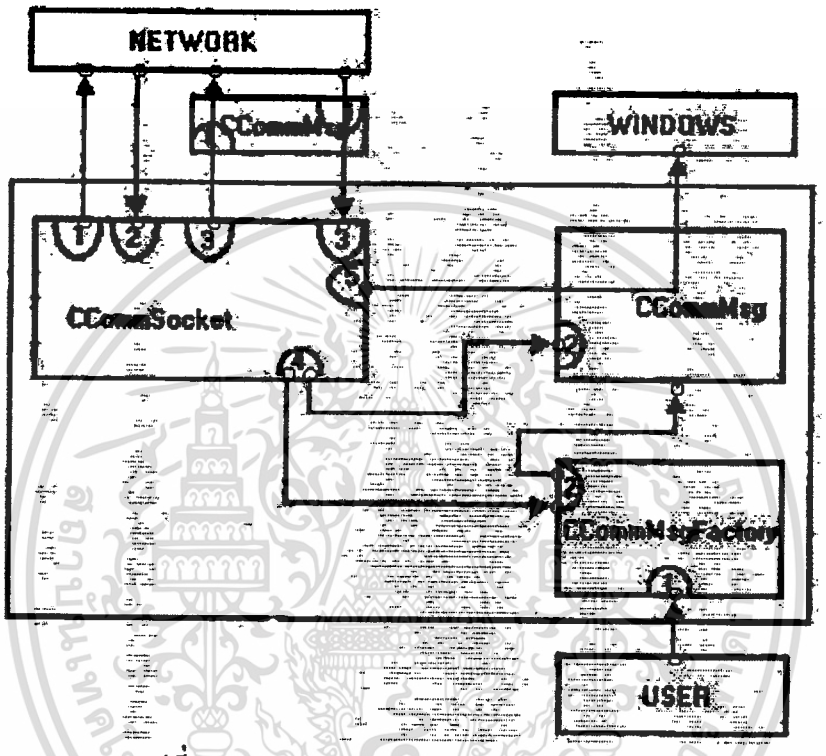
ตารางที่ 4.6 CRC ของ CMcastSocket

CmcastSocket: CAsyncSocket	Concrete
1.Join/Leave Group	
2.Receive Message from Group	
3.Create CasyncSocket Instance for Send Message to Group	
4.Send Message to Group	CAsyncSocket

จาก CRC, CMcastSocket ซึ่งให้บริการในส่วนของ Multicast มีหน้าที่ 1) เข้าร่วมหรือออกจากกลุ่มของ Multicast 2) คอยรับข้อมูลจากกลุ่ม 3) สร้าง Socket สำหรับส่งข้อมูลไปยังกลุ่ม 4) ให้บริการส่งข้อมูลไปยังกลุ่ม.

4.4 Collaboration Graphs

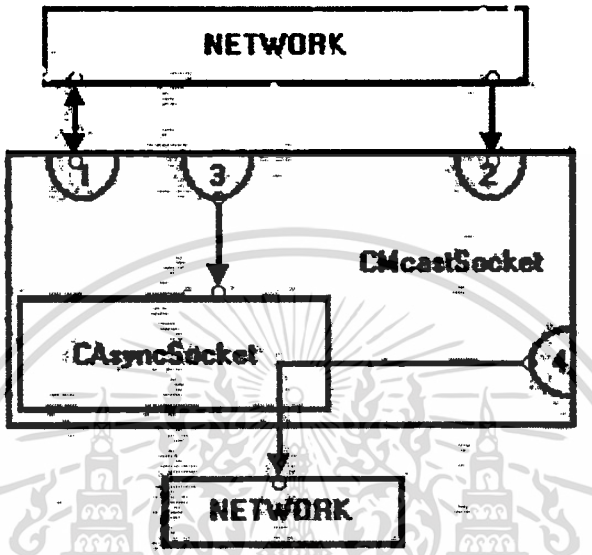
4.4.1 Collaboration Graphs ของส่วน TCP



รูปที่ 4.1 Collaboration Graphs ของส่วน TCP

ความหมายของ Collaboration Graphs ของส่วน TCP คือ เมื่อดูประกอบจากคำอธิบายในเบื้องต้นของลำดับการทำงานของ Window Socket แบบ Client-Server ว่ามี Socket 3 ประเภท แล้วจากการออกแบบรวมให้เป็น Socket Class เดียวโดยทำหน้าที่ทั้ง 3 ประเภทเหมือนเดิม จุดที่น่าสนใจคือ เมื่อติดต่อเสร็จแล้ว Client Socket และ Server Socket ทำงานกับส่วนอื่นอย่างไร จากรูปจะเห็นว่ามีการส่ง Messages ระหว่างกันโดย Encode ในรูป Message Type ต่าง ๆ ดังนั้นเวลารับข้อมูลเข้ามาก็ต้อง Decode ก่อนโดยใช้ Message Factory ที่เรากำหนด, Factory นี้ก็จะสร้าง Message Object ชนิดนั้นขึ้นมา แล้วจึงนำเอา Message Object นั้นไป Serialize ข้อมูล(ในส่วนที่ไม่ใช่ส่วนข้อมูล Encode Type), เมื่อได้ข้อมูลครบของ Message นั้น ก็สามารถส่งต่อไปยัง Handler ที่ทำหน้าที่ Application Protocol เพื่อตอบสนองต่อไป โดยส่งในลักษณะ User Messages ของระบบ Windows ที่มีพารามิเตอร์เป็น Message Object นั้น กับ Pointer ของ Socket นั้นเพื่อให้สามารถแยกออกได้ว่า เป็น Server หรือ Client.

4.4.2 Collaboration Graphs ของส่วน Multicast (บน UDP/IP)



รูปที่ 4.2 Collaboration Graphs ของส่วน Multicast (บน UDP/IP)

จาก Collaboration Graphsในส่วน Multicast อธิบายได้ว่า CMcastSocket ทำหน้าที่ Join/Leave Group และคอยรับ Message แต่ถ้าจะส่ง Message ต้องใช้ CAsyncSocket ที่สร้างตอนเริ่ม Join Group ในการส่ง

4.5 สรุป

จากแนวคิดและการออกแบบที่ได้กล่าวมา ก็จะเห็นภาพอย่างคร่าว ๆ ในหน้าที่ของระบบนี้ ซึ่งได้แก่ การสนับสนุนการสื่อสารผ่านเครือข่ายทั้งในแบบ TCP และ Multicast (บน UDP/IP) สำหรับสัญลักษณ์ที่ใช้ในการอธิบายการออกแบบนี้ อาจจะดูสับสน เพราะในปัจจุบันยังไม่มีสัญลักษณ์ที่ใช้อธิบายระบบที่เป็นแบบ Dynamics ได้และเป็นที่ยอมรับโดยทั่วไป ดังนั้นจึงอาศัยรูปแบบของ CRC และ Collaboration Graphs ที่ดัดแปลงผสมกับคำอธิบาย ในการอธิบายการออกแบบระบบย่อยนี้

บทที่ 5
ระบบย่อยการตอบสนองต่อแรง

ระบบย่อยการตอบสนองต่อแรงตามที่ได้ออกแบบในบทการออกแบบ มีหน้าที่หลักคือ การควบคุม Haptic Process, การสร้างและจัดการ Haptic Scene Graphs, การแสดงกราฟิกในส่วนของ Haptic Scene Graphs, การจำลองการควบคุมห่วงกลมเสมือนร่วมกันของสองฝ่ายผ่านการสื่อสารแบบ Multicast, การจำลองเส้นลวดดัดโค้งเสมือนและการจำลองแรงตอบสนองของเส้นลวดดัดโค้งเสมือนที่มีต่อห่วงกลมเสมือน ซึ่งภายในบทนี้จะอธิบายถึงทฤษฎีที่เกี่ยวข้องและตัวระบบย่อยการตอบสนองต่อแรง ในลำดับดังนี้

5.1 ทฤษฎีที่เกี่ยวข้อง

ทฤษฎีที่เกี่ยวข้องสำหรับระบบย่อยการตอบสนองต่อแรงมี 3 ส่วนที่สำคัญ คือ 1) การเขียนโปรแกรมด้วย GHOST SDK เพื่อใช้ในการจำลองการตอบสนองต่อแรง 2) การ Extend Functions ของ GHOST SDK เพื่อการจำลองการตอบสนองต่อแรงเพิ่มจาก GHOST SDK เช่น การจำลองการตอบสนองต่อแรงของเส้นลวดดัดโค้งเสมือนที่มีต่อห่วงกลมเสมือนในระบบย่อยนี้ 3) Bezier Curves ที่ใช้ในการแสดงกราฟิกและใช้ช่วยในการจำลองแรงตอบสนองของเส้นลวดดัดโค้งเสมือน

5.1.1 การเขียนโปรแกรมด้วย GHOST SDK

GHOST SDK เป็น Software Library ที่ใช้ร่วมกับ PHANToM ซึ่งเป็นอุปกรณ์ Force Feedback Device ที่ผลิตและพัฒนาโดยบริษัท Sensable Technologies, Inc. ซึ่งใช้ในการ Implement เกม RealLife นี้ โดยทฤษฎีการเขียนโปรแกรมด้วย GHOST SDK ในบทนี้จะกล่าวถึง

5.1.1.1 GHOST SDK

5.1.1.2 โครงสร้างของ GHOST SDK Applications

5.1.1.3 โครงสร้างการโปรแกรม

5.1.1.4 การสร้าง Haptic Scene Graphs

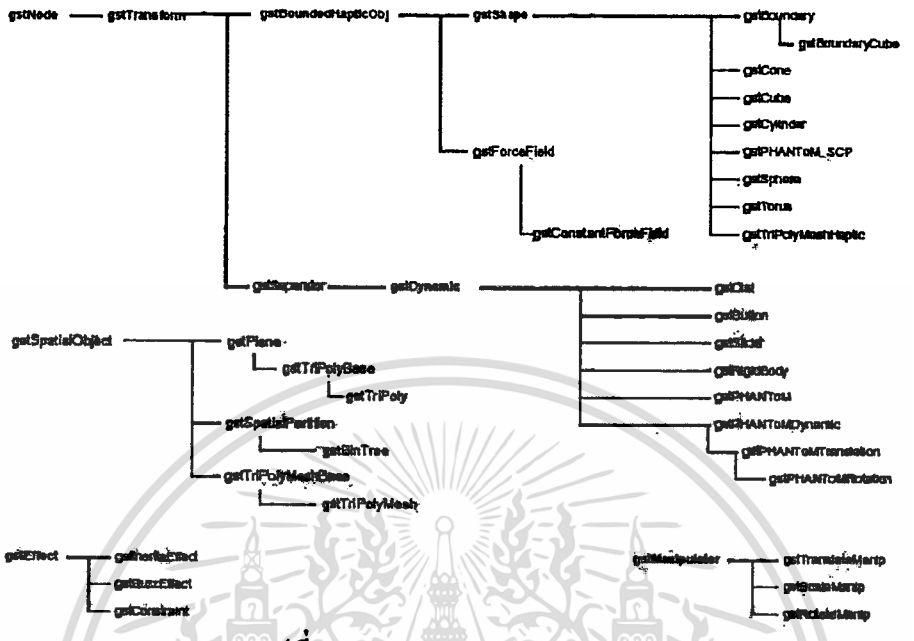
5.1.1.5 การควบคุมการ Simulations

5.1.1.1 GHOST SDK

GHOST (General Haptics Open Software Toolkit) เป็น API เชิงวัตถุภาษา C++ ที่ใช้ในการสร้าง จัดการ และควบคุมการ Simulation Haptic Environment ของ PHANToM (ซึ่งเป็น Force Feedback Device ที่พัฒนาโดย Sensable Technologies, Inc.) ทั้งบนระบบ Windows NT และ IRIX โดย GHOST SDK ใช้โครงสร้างแบบ Scene Graph (คือการนำ Object ของ Class ต่างๆ ซึ่งต่อไปจะเรียกว่า Node มาสัมพันธ์กันแบบ Tree) เพื่อกำหนดลักษณะของ Haptic Environment.

โดยส่วนปลายของ Tree จะเป็น Node ที่แทนรูปทรง (Geometry Nodes) หรือแทนลักษณะของแรงในรูปแบบต่าง ๆ และ Node ที่แทนตัว PHANToM Device (Phantom Nodes) ส่วนที่ไม่ใช่ปลายของ Tree จะเป็น Node ที่ใช้ในการรวมกลุ่ม (Hierarchical Nodes) เพื่อกำหนดลักษณะร่วมกัน ซึ่งอาจจะมีการรวมกลุ่มและทำให้ทั้งกลุ่มมีพฤติกรรมร่วมกันแบบ Dynamics หรือแบบ Static

ในการควบคุมการ Simulations จะใช้ Scene Node ควบคุม คือใช้ในการกำหนด Haptic Scene Graphs ที่ต้องการจำลองการตอบสนองต่อแรง และใช้ในการเริ่ม, หยุด, หรือจัดการการ Simulations โดย Class ต่าง ๆ ภายใน GHOST SDK เป็นดังรูปที่ 5.1



รูปที่ 5.1 GHOST SDK Class Trees.

5.1.1.2 โครงสร้างของ GHOST SDK Applications

โครงสร้างของ Haptic Applications ที่สร้างโดยใช้ GHOST SDK จะประกอบด้วย 2 Process คือ 1) Application Process และ 2) Haptic Process ดังรูปที่ 5.2 คือ

Application Process

Application Process หรือ Application Loop ทำหน้าที่หลัก ๆ คือ

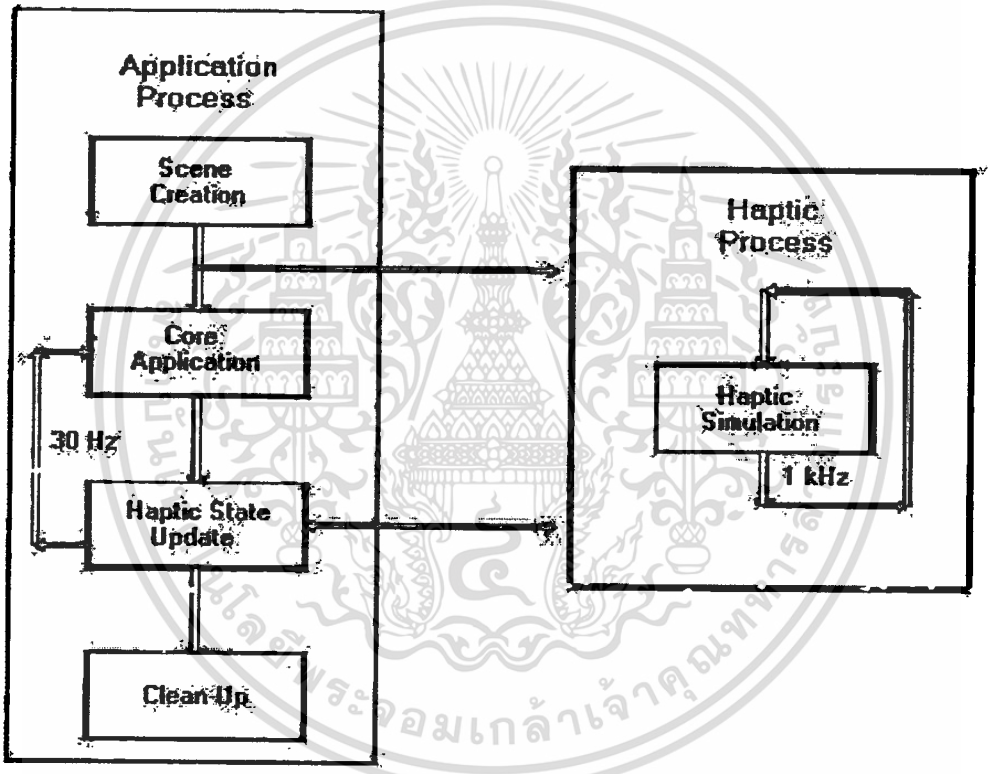
- แสดง Graphic User Interfaces และ Functions ต่าง ๆ ของ Application
- สร้าง/จัดการ Haptic Scene Graph เพื่อสร้าง Haptic Environment
- เริ่ม/หยุด/จัดการ Haptic Simulation Process ผ่าน Scene Node
- แสดงกราฟิกจากข้อมูลที่ได้จาก Haptic Process

Haptic Process

Haptic Process หรือ Servo Loop ทำหน้าที่ Simulation Haptic Environment ซึ่งกำหนดโดย Haptic Scene Graph และ Scene Node ที่ได้สร้างและกำหนดใน Application Process โดยการ Simulation จะเป็นการวนลูปการคำนวณเพื่อกำหนดแรงตอบสนองที่จะแสดงผ่าน PHANToM Device ที่อัตรา 1 kHz ซึ่งภายในลูปการคำนวณจะมีการทำงาน คือ

- ปรับปรุงสถานะของ Phantom Node ใน Haptic Scene Graph

- ปรับปรุงสถานะแบบ Dynamic ของกลุ่ม Dynamic Nodes
- ตรวจสอบการชนของ Phantom Node กับ Geometry Nodes
- คำนวณแรงสุทธิที่มีต่อ Phantom Node
- ถ้าใน Application Process มีการกำหนด Graphic Callback หรือ Event Callback สำหรับ Node ใด ก็จะมีการเตรียมข้อมูลสำหรับ Callback ของ Node นั้นๆ เมื่อ Node นั้น ๆ มีสถานะใหม่



รูปที่ 5.2 โครงสร้างของ GHOST SDK Applications

5.1.1.3 โครงสร้างการโปรแกรม

ในการสร้าง Haptic Application ด้วย GHOST SDK มีโครงสร้างการโปรแกรม คือ

ตารางที่ 5.1 โครงสร้างการโปรแกรมด้วย GHOST SDK

1. สร้าง Application ตามปกติ
2. สร้าง Haptic Scene Graphs และกำหนด Graphic Callbacks หรือ Event Callbacks
3. สร้าง Scene Node และกำหนด Haptic Scene Graphs ที่ใช้
4. ควบคุม Haptic Process ผ่าน Scene Node

ตารางที่ 5.1 โครงสร้างการโปรแกรมด้วย GHOST SDK (ต่อ)

5. เมื่อต้องการข้อมูลของ Haptic Process ให้เรียกผ่าน Scene Node เพื่อให้ Haptic Process เตรียมข้อมูลให้พร้อมก่อน และจะเรียกกลับผ่าน Callback ที่กำหนดใน Haptic Scene Graphs
6. เมื่อเลิกใช้ ทำการยกเลิกการจองทรัพยากรต่าง ๆ

ตัวอย่าง เช่น

ตารางที่ 5.2 ตัวอย่างการโปรแกรมด้วย GHOST SDK

ขั้นตอนที่	ตัวอย่าง
1	// Make application ...

2	// Create Root Node of Haptic Scene Graphs
	gstSeparator* root = new gstSeparator;
	// Create PHANToM Node
	gstPHANToM* phantom = new gstPHANToM("phantom.ini");
	// Create Geometry Nodes
	gstSphere* sphere = new gstSphere; // Create sphere
	sphere->setGraphicsCallback(callbackfuntion); // Set Callback functions
	sphere->serRadius(20); // Set radius to 2 millimeters
	// Build Haptic Scene Graphs
	root->addChild(phantom);
	root->addChild(sphere);

3	// Create the only one Scene Node for the Application
	gstScene* scene = new gstScene;
	// Set Haptic Scene Graphs
	scene->setRoot(root);

4	// Controls Haptic Process through Scene Node
	scene->startServoLoop();
	...
	scene->stopServoLoop();

5	// Use callback mechanism through Scene Node when want to know Haptic
	// Process Data
	scene->updateGraphics(); // For Graphics callback
	scene->updateEvents(); // For Events callback
	// The Haptic Process will callback to Callback Function
	// when it prepare data ready
	...

6	// Clean up delete scene; delete root; ...

5.1.1.4 การสร้าง Haptic Scene Graphs

GHOST SDK ใช้การสร้าง Haptic Scene Graphs เพื่อกำหนด Haptic Environment โดยการนำ Node ต่าง ๆ มาสัมพันธ์กับแบบ Tree โดยมี Node บนสุดทำหน้าที่เป็น Root ของ Haptic Scene Graph มี Node ภายในที่ทำหน้าที่กำหนดกลุ่มหรือเพื่อสืบทอดลักษณะ, พฤติกรรมต่าง ๆ เป็นลำดับชั้น มี Node ปลายสุดที่ทำหน้าที่แทนตัววัตถุ หรือการตอบสนองต่อแรงในรูปแบบต่าง ๆ ที่ต้องการ Simulation

ประเภทของ Node แบ่งตามประเภทของ Class ซึ่งมีลักษณะหน้าที่ต่าง ๆ กัน คือ

Geometry Node Class

ประกอบด้วย gstCube, gstCone, gstCylinder, gstSphere, gstTorus และ gstTripolyMeshHaptic ทำหน้าที่แทนรูปทรงตามชื่อ และกำหนดลักษณะต่าง ๆ ของรูปทรงนั้น ตัวอย่าง คือ

ตารางที่ 5.3 ตัวอย่างการโปรแกรม Geometry Node

```
gstSphere* sphere = new gstSphere;
sphere->setGraphicsCallback(sphereCB);
sphere->setTranslate(0,50,0);
sphere->setRadius(20);
sphere->setSurfaceKspring(0.3);
sphere->setSurfaceFdynamic(0);
sphere->setSurfaceFstatic(0);
...
gstCube* cube = new gstCube;
cube->setGraphicsCallback(cubeCB);
...
```

เป็นการสร้างทรงกลมรัศมี 20 mm. ที่ตำแหน่ง 0,50,0 ที่ลักษณะผิวมีความนุ่มและไม่ฝืด และสร้างลูกบาศก์

Hierarchical Node Class

คือ gstSeparator ทำหน้าที่รวมกลุ่ม Node เพื่อการจัดการที่เป็นกลุ่มเดียวกัน เช่น เคลื่อนย้าย, หมุน, ปรับขนาด ตัวอย่าง คือ

ตารางที่ 5.4 ตัวอย่างการโปรแกรม Hierarchical Node

```
gstSeparator *root = new gstSeparator;
gstSeparator *rootPhantom = new gstSeparator;
gstSeparator *rootHaptic = new gstSeparator;
root->addChild(rootPhantom);
root->addChild(rootHaptic);
rootHaptic->addChild(sphere);
rootHaptic->addChild(cube);
rootHaptic->setRotate(gstVector(0.0,0.0,1.0),M_PI/2.0);
```

เป็นการสร้าง Root ของ Haptic Scene Graphs คือ root และสร้าง Tree ภายใต้ root โดยแยกเป็น rootPhantom และ rootHaptic จากนั้นรวมกลุ่ม sphere และ cube ภายใต้ rootHaptic แล้วหมุนทั้งกลุ่มรอบแกน z ทิศทวนเข็มนาฬิกา 90 องศา

Haptic Interface Device Node Class

คือ gstPHANToM และ gstPHANToM_SCP ทำหน้าที่แทนตัว Device เพื่อให้ข้อมูลตำแหน่งและสถานะของตำแหน่งจริงและตำแหน่งสัมผัสพื้นผิว Surface Contact Point (SCP) (SCP เกิดเมื่อสัมผัสกับรูปทรง) ตามลำดับ ตัวอย่าง คือ

ตารางที่ 5.5 ตัวอย่างการโปรแกรม Phantom Node

```
gstPHANToM* phantom = new gstPHANToM("phantom.ini");
gstPHANToM_SCP *scp = new gstPHANToM_SCP;
phantom->setSCPNode(scp);
rootPhantom->addChild(phantom);
rootPhantom->addChild(scp);
// If use SCP Node for Graphics
scp->setGraphicsCallback(scpCB);
```

เป็นการสร้างทั้ง Phantom Node และ SCP Node โดย SCP node ต้องมีความสัมพันธ์กับ Phantom Node ที่สร้างขึ้นมาจริง ในลักษณะการใช้งานอาจจะใช้เฉพาะ Phantom Node ก็ได้ แต่เมื่อมีการสัมผัสจนจมเข้าไปในพื้นผิวจะทำให้ภาพของเครื่องมือจมไปด้วยจากข้อมูลจริง ถ้าต้องการให้เห็นตลอดก็ใช้ข้อมูลของ SCP Node ในการแสดงภาพบนผิวแทน เมื่อตัดสินใจเลือกใช้ข้อมูลจาก Node ใดก็กำหนด Callback Function ให้กับ Node นั้นเพียง Node เดียว.

Addition Haptic Interface Device Node Class

คือ gstBoundaryCube ทำหน้าที่กำหนดขอบเขตให้กับ Device ตัวอย่าง คือ

ตารางที่ 5.6 ตัวอย่างการโปรแกรม Workspace Node

```
gstBoundaryCube* wrkspc = new gstBoundaryCube();
wrkspc->setGraphicsCallback(wrkspcCB);
wrkspc->setWidth(WRKSPC_WIDTH);
wrkspc->setLength(WRKSPC_LENGTH);
wrkspc->setHeight(WRKSPC_HEIGHT);
wrkspc->setPosition(X,Y,Z);
phantom->setBoundaryObj(wrkspc);
rootPhantom->addChild(wrkspc);
```

เป็นการกำหนดขนาดและขอบเขตของ Device เพื่อกันการออกนอกพื้นที่

และ GHOST SDK ยังมี Node Classes ต่าง ๆ อีกที่ช่วยในการแสดงแรงตอบสนอง หรือลักษณะต่าง ๆ เช่น Effect Node Class, Manipulator Node Class, Dynamic Property Node Class, PHANToM Node Class (Sensable Technologies, 1998)

5.1.1.5 การควบคุมการ Simulations

ในการควบคุมการ Simulation ของ Haptic Process จะควบคุมผ่าน Scene Node ซึ่งสร้างจาก gstScene Class ทำหน้าที่กำหนด Haptic Scene Graph ที่จะ Simulate, สั่งเริ่มหยุด Haptic Process สั่งปรับปรุงข้อมูลของ Haptic เมื่อ Application Process ต้องการใช้ และจะมีเพียง Node เดียวในหนึ่ง Application ตัวอย่าง คือ

ตารางที่ 5.7 ตัวอย่างการโปรแกรม Scene Node

```
gstScene* scene = new gstScene;
scene->setRoot(root);
scene->startServoLoop();
...
scene->updateGraphics();
scene->updateEvents();
...
scene->stopServoLoop();
```

เป็นการสร้าง Scene Node โดยกำหนด root ให้เป็น Haptic Scene Graph ที่จะ Simulate, สั่งเริ่ม Haptic Process สั่งปรับปรุงข้อมูล Haptic สำหรับ Callback ที่ได้ลงทะเบียนไว้ และสั่งหยุด Haptic Process เมื่อต้องการ

5.1.2 การ Extend Functions ของ GHOST SDK

เมื่อพิจารณา GHOST SDK เราสามารถแยกความสามารถหลัก ๆ ออกได้เป็น 3 กลุ่ม คือ

1. การจำลองการตอบสนองในลักษณะวัตถุมีรูปทรง (Geometry) เช่น ทรงกลม, ทรงกระบอก, ทรงกรวย
2. การจำลองการตอบสนองในลักษณะสนามแรง (Force Fields)
3. การจำลองการตอบสนองแบบพิเศษ Effect และ Manipulator (Effect and Manipulator) เช่น การสั่น, แรงเฉื่อย, การจำกัดตำแหน่ง

และแต่ละกลุ่มก็สามารถเพิ่มเติมความสามารถโดยอาศัยการสืบทอดเพื่อเพิ่มฟังก์ชันต่าง ๆ เข้าไป แต่ในที่นี้ สนใจเฉพาะการเพิ่มเติมความสามารถของ Effect ดังที่กล่าวมาแล้วในบทการออกแบบ ดังนั้นจึงขอกล่าวถึงการ Extend Functions ในส่วนเฉพาะ gstEffect.

gstEffect

gstEffect ใช้ในการส่งแรงเพิ่มถึง gstPHANToM หรือ gstPHANToMDynamic โดยตรง ซึ่งนอกเหนือจากแบบแรงจากการชนกับผิววัตถุ โดยมี Derived Classes คือ gstBuzzEffect เป็น Effect ในการสั่น, gstConstraintEffect เป็น Effect ในการจำกัดตำแหน่งของ PHANToM, และ gstInertiaEffect เป็น Effect ในการสร้างแรงเฉื่อย

ในการใช้ gstEffect จะต้องสร้าง Instance และกำหนดให้กับ Instance ของ gstPHANToM หรือ gstPHANToMDynamic ผ่าน Method setEffect ซึ่งจะเริ่มแสดง Effect โดยใช้ Method startEffect และจะหยุดเมื่อถึงเวลาที่กำหนดหรือผ่าน Method stopEffect ขึ้นกับแต่ละ Class ที่ Implement มา ดังตัวอย่าง

ตารางที่ 5.8 ตัวอย่างการโปรแกรม gstEffect Class

gstBuzzEffect* buzzEffect = new gstBuzzEffect;
...
phantom->setEffect(buzzEffect);
phantom->startEffect();
...
phantom->stopEffect();

เป็นการใช้ gstBuzzEffect ซึ่งเป็น Effect ในการสั่น โดยการใช้ Effect จะใช้ได้เพียงครั้งละ 1 Effect เท่านั้น

การเพิ่มฟังก์ชันทาง gstEffect

gstEffect เป็น Abstract Base Class ในการส่งแรงเพิ่มไปยัง PHANToM นอกเหนือจากการคำนวณแรงจากการชนกับผิววัตถุของ Class พวกมีรูปทรง โดยตัวอย่าง คือ

ตารางที่ 5.9 ตัวอย่างการเพิ่มฟังก์ชันทาง gstEffect

```
double time;
gstBoolean active;
virtual gstBoolean start()
{
  active = TRUE;
  time = 0.0;
  return TRUE;
}
virtual void stop()
{
  active = FALSE;
}
virtual gstVector calcEffectForce(void* PHANToM)
{
  gstPHANToM* phantom = (gstPHANToM*)PHANToM;
  // Read current state from phantom
  ...
  gstVector force;
  // Calculate force
  ...
  return force;
}
```

โดย ตัวแปร time เป็น Elapsed Time ตั้งแต่เริ่ม Effect

ตัวแปร active เป็นตัวบอกถึงสถานะการเริ่มหรือหยุดของ Effect

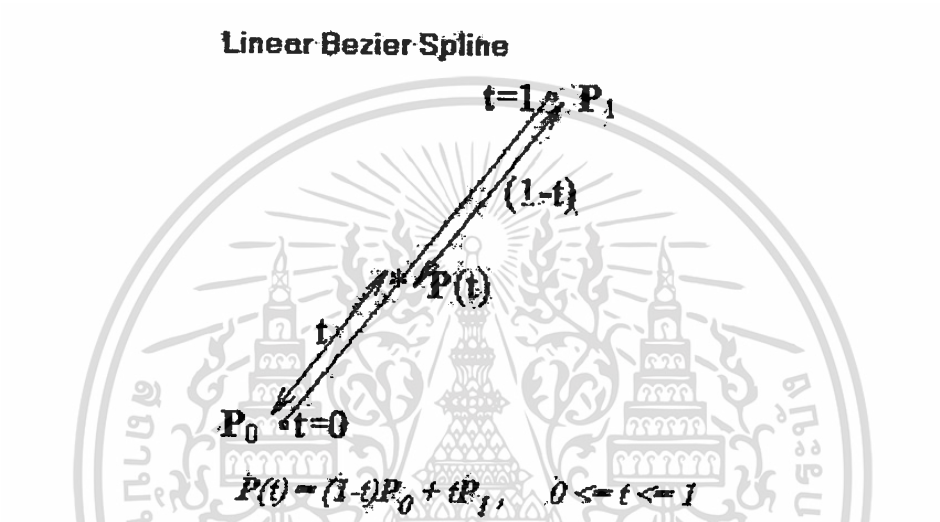
Method start และ stop เป็นส่วน Implement ของเราในการเริ่มหรือหยุด Effect

Method calcEffect เป็นส่วน Implement ของเราในการคำนวณแรงที่จะส่งเพิ่มไปยัง PHANToM ซึ่งจะผ่าน Pointer ของ gstPHANToM เข้ามาเพื่อให้รู้สถานะของ PHANToM

5.1.3 Bezier Curves

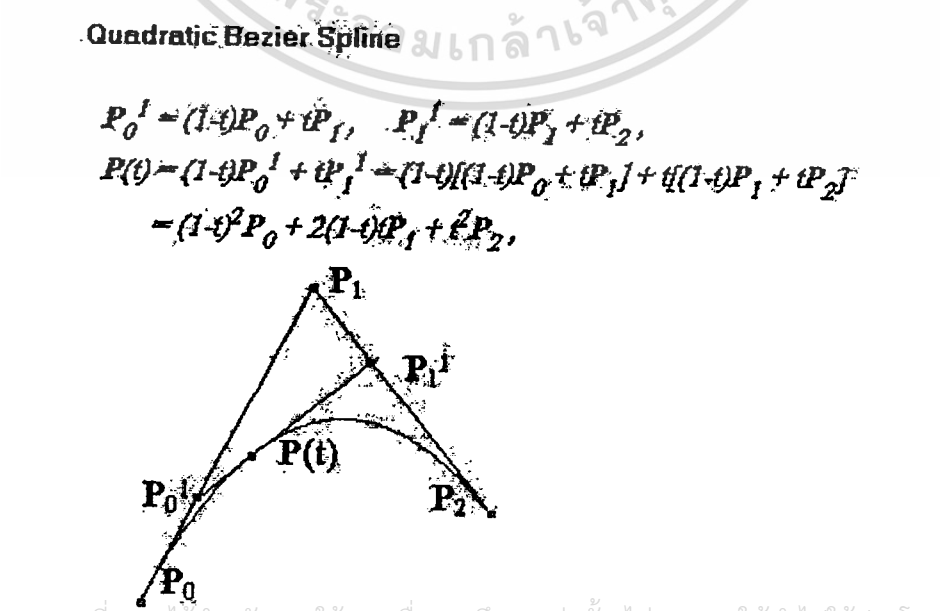
Bezier Curves เป็น Curve ที่ได้จากการทำซ้ำ Affine Combination Control Points ในรูปแบบ t กับ (1-t) เช่น

<u>กรณี Linear Bezier</u>



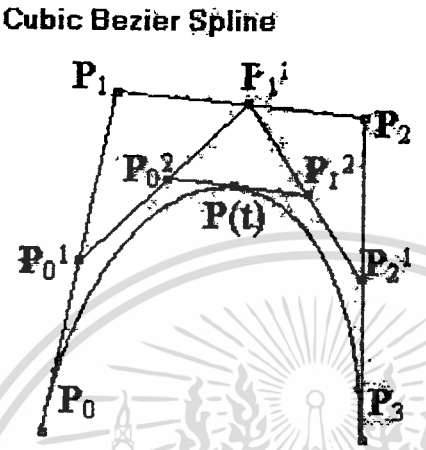
รูปที่ 5.3 Linear Bezier Spline

<u>กรณี Quadratic Bezier</u>



รูปที่ 5.4 Quadratic Bezier Spline

<u>กรณี Cubic Bezier</u>



$$P(t)=(1-t)^3P_1+3t(1-t)^2P_2+3t^2(1-t)P_3+t^3P_4$$

Points $P_0, P_0^1, P_0^2, P(t)$ and $P(t), P_1^2, P_2^1, P_3$ are control points of new small splines again.

รูปที่ 5.5 Cubic Bezier Spline

โดย t มีค่าตั้งแต่ 0 ถึง 1 ซึ่งเมื่อ Run t ผ่านฟังก์ชันของ Bezier Curve P(t) ก็จะได้เป็น Curve ซึ่งลากผ่านจุดที่ t ต่าง ๆ ตั้งแต่ 0 ถึง 1 โดย Control Point เป็นจุดที่เรากำหนดขึ้นมาเพื่อใช้ในการออกแบบ Curve ตามที่เราต้องการ

5.2 ระบบย่อยการตอบสนองต่อแรง

GHOST SDK ที่ใช้ในการ Simulation การตอบสนองต่อแรงที่จะนำมาพัฒนาเกม RealLife เป็น Object-Oriented API ที่ใช้การสร้าง Scene Graph ของ Haptics เพื่อสร้าง Haptics Environment ระบบย่อยการตอบสนองต่อแรงจึงทำหน้าที่สร้าง Scene Graph ของเกม และควบคุมการดำเนินของเกมในส่วนการตอบสนองต่อแรงโดยผ่าน Scene Node ที่อธิบายมาแล้วในส่วนของ GHOST SDK และสนับสนุนการแสดงกราฟิกของแต่ละวัตถุที่จำลองขึ้นมา ซึ่งจะถูกเรียกไปแสดงในเวลาที่เหมาะสมโดยระบบย่อยกราฟิก เช่นในที่นี้คือ รูปทรงพื้นฐานต่าง ๆ , ห่วงกลมเสมือน และเส้นลวดดัดโค้งเสมือน เป็นต้น จะเห็นได้ว่าระบบย่อยนี้เพียงแต่ทำหน้าที่สร้าง Scene Graphs ขึ้นมา แล้วก็ควบคุมโดยผ่าน Scene Node ไม่มีอะไรซับซ้อน แต่ส่วนสำคัญอยู่ที่ Effect Node ที่เรา Implement ขึ้นมาเพื่อจำลองการควบคุมห่วงกลมเสมือน และแสดงแรงตอบสนองของเส้นลวดดัดโค้งเสมือนที่มีต่อห่วงกลมเสมือน ซึ่งจะได้อธิบายต่อไป

5.2.1 แนวคิด

จากแนวคิดและการออกแบบในบทการออกแบบ จะเห็นว่านอกจากการสร้าง Haptic Scene Graph และการควบคุม Haptic Process แล้ว ระบบย่อยการตอบสนองต่อแรงมีภาระกิจที่เป็นส่วนหลักของเกม 2 ประการ คือ การบังคับห่วงร่วมกันของ 2 ผู้เล่นผ่านเครือข่ายคอมพิวเตอร์ ซึ่งเราได้กล่าวไว้ในตอนแนวคิดว่าจะใช้การสื่อสารแบบ Multicast ในการแลกเปลี่ยน Local View of Global State ซึ่งในที่นี้ก็คือ จุดศูนย์กลางของห่วงกลม โดยพ่วงระยะการแลกเปลี่ยนข้อมูลไปกับ Haptic Loop คือทุก ๆ 1 ms. ประการที่สองคือ การตรวจสอบว่าห่วงชนเส้นลวดดัดโค้งเสมือนหรือไม่ โดยในการออกแบบเส้นลวดดัดโค้งเสมือนเราใช้ Cubic Bezier Curve และใช้การคำนวณง่าย ๆ ในการตรวจสอบการชนระหว่างห่วงกลมเสมือนกับเส้นลวดดัดโค้งเสมือน คือ ถ้าระยะห่างของจุดศูนย์กลางห่วงกับตำแหน่ง Projection ของจุดศูนย์กลางห่วงตามระนาบที่ตั้งฉากกับแกน X บนเส้นลวดดัดโค้งเสมือนมากกว่ารัศมีของห่วง แสดงว่าห่วงชนเส้นลวดที่จุดนั้น และจะตอบสนองด้วยแรงที่เป็นอัตราส่วนตามที่เกินนั้น

5.2.2 การออกแบบ

5.2.2.1 CRC

สำหรับระบบย่อยการตอบสนองต่อแรง จะออกแบบและอธิบายเฉพาะ Class ที่สำคัญของเกมเท่านั้น คือ ตัวระบบย่อย(CGameHaptic), Ring-Curve Effect(HRingEffect), และ Bezier Curve (CBezier3d) เท่านั้น ส่วน Class อื่น ๆ ในระบบก็คือ Class ใน GHOST SDK หรือ Derived Class จาก GHOST SDK ที่เพิ่มฟังก์ชันทางกราฟิกเข้าไป ขอไม่อธิบายเพราะดูได้จากเอกสารของ GHOST SDK เอง และมีการ Implement เพิ่มเติมไม่มากนัก

ตารางที่ 5.10 CRC ของ CGameHaptic

CGameHaptic:CObject	Concrete Subsystem
1.Control Haptic Process 2.Manage Haptic Scene Graph 3.Simulate Ring-Curve Effect 4.Handle Haptic Scene Graph Graphics	CGameNet
5.Load/Save Setting 6.Load Curve Set	CGameDoc

จาก CRC, CGameHaptic Subsystem มีหน้าที่ 1) ควบคุม Haptic Process 2) สร้างและจัดการ Haptic Scene Graphs 3) เป็นส่วนที่ใช้ในการ Simulate การควบคุมห่วงกลมเสมือนร่วมกันของสองฝ่ายผ่านการสื่อสารแบบ Multicast และการจำลองการตอบสนองต่อแรงของเส้นลวดคัดโค้งเสมือนที่มีต่อห่วงกลมเสมือน 4) แสดงกราฟิกในส่วนของ Haptic Scene Graph 5) อ่านหรือบันทึกข้อมูลของระบบย่อยโดยใช้บริการจาก CGameDoc 6) อ่านข้อมูลของ Bezier Control Points จากไฟล์ที่จะจำลองเป็นเส้นลวดคัดโค้งเสมือน

ตารางที่ 5.11 CRC ของ HRingEffect

HRingEffect:gstConstraintEffect	Concrete
1.Exchange Data for Collaborative Controlled-Ring	CGameNet
2.Simulate Ring-Curve Effect	CBezier3d
3.Render Ring-Curve Graphics	CBezier3d

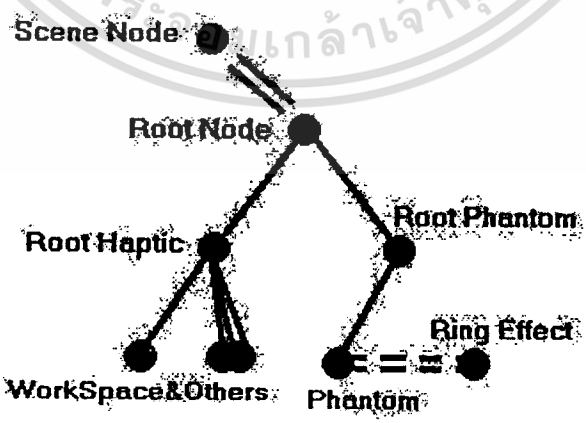
จาก CRC, HRingEffect มีหน้าที่ 1) แลกเปลี่ยนข้อมูลที่ใช้ในการควบคุมห่วงระหว่างสองฝ่ายผ่าน Network โดยทำงานร่วมกับ CGameNet ในส่วนของบริการ Multicast 2) เป็นส่วนที่ใช้ในการ Simulate การควบคุมห่วงกลมเสมือนร่วมกันของสองฝ่าย และการจำลองการตอบสนองต่อแรงของเส้นลวดดัดโค้งเสมือนที่มีต่อห่วงกลมเสมือนโดยทำงานร่วมกับ CBezier3d ในการคำนวณการชน 3) แสดงกราฟิกในส่วนของ ห่วงกลมเสมือนและเส้นลวดดัดโค้งเสมือน

ตารางที่ 5.12 CRC ของ CBezier3d

CBezier3d:CObject	Concrete
1.Load Bezier Control Points 2.Render Bezier Curve Graphics 3.Calculate t from x	

จาก CRC, CBezier3d มีหน้าที่ 1) อ่าน Control Points จากไฟล์ 2) แสดงกราฟิกของ Bezier Curves 3) คำนวณค่า t จากค่า x ของจุดบน Curve

5.2.2.2 Haptic Scene Graphs



รูปที่ 5.6 Haptic Scene Graphs ของเกม RealLife

จากรูป Haptic Scene Graph ความหมายคือ เรามี Root Node เป็น Root แล้วแบ่งกลุ่มเป็น Root Haptic เพื่อจัดกลุ่มที่เป็น Geometry Objects และ Root Phantom เพื่อเป็น Parent ของ

Phantom Node ส่วนในการ Implement เราจะใช้ Ring Effect Node เป็น Effect และ Scene Node เป็นตัวควบคุม Haptic Process

5.2.3 การ Implement

5.2.3.1 การสร้าง Haptic Scene Graphs

ตารางที่ 5.13 การสร้าง Haptic Scene Graphs

```
// WorkSpace
m_WorkSpace = new HBoundaryCube;
m_WorkSpace->setWidth(m_Xmax-m_Xmin);
m_WorkSpace->setHeight(m_Ymax-m_Ymin);
m_WorkSpace->setLength(m_Zmax-m_Zmin);
m_WorkSpace->setPosition((m_Xmax+m_Xmin)/2,
                         (m_Ymax+m_Ymin)/2,
                         (m_Zmax+m_Zmin)/2);
// Phantom
m_Phantom = new HPhantom(PHANTOM_NAME);
m_Phantom->setBoundaryObj(m_WorkSpace);
// Game
m_RingEff = new HRingEffect;
// RootHaptic
m_ RootHaptic = new gstSeparator;
m_RootHaptic->addChild(m_WorkSpace);
// RootPhantom
m_RootPhantom = new gstSeparator;
m_RootPhantom->addChild(m_Phantom);
// Root
m_Root = new gstSeparator;
m_Root->addChild(m_RootPhantom);
m_Root->addChild(m_RootHaptic);
// Scene
m_Scene = new gstScene;
m_Scene->setRoot(m_Root);
```

นี่คือ Code ในส่วนการสร้าง Haptic Scene Graph ของเกม (ซึ่งอาจจะมีการเปลี่ยนแปลงบ้าง ขอให้ดูที่ภาคผนวกเป็นหลักสำหรับการ Implement ครั้งสุดท้าย) เป็นการสร้างตามรูปที่ 5.7 Haptic Scene Graphs ของเกม RealLife

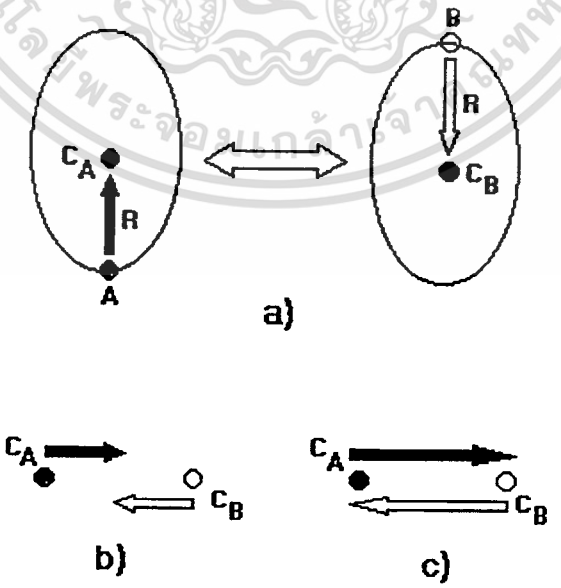
5.2.3.2 การ Implement HRingEffect

<u>การบังคับห่วงร่วมกัน</u>

ในการ Implement จากการศึกษาการใช้งาน GHOST SDK เราสามารถ Implement ได้โดยใช้ Node ประเภทต่าง ๆ ได้หลายแบบในการสืบทอด แต่ Node ที่น่าสนใจคือ gstConstraintEffect เพราะคุณสมบัติที่เป็น Effect Node คือ Set เข้าออกได้, หยุดเริ่ม Effect ได้ และคุณสมบัติของตัว Constraint เองคือ สามารถกำหนด Constraint Point ซึ่งจะส่งแรงเพื่อดึง Phantom ไปยังจุดนั้น ด้วย Model Spring-Damp ซึ่งในการ Implement จริงเราอาจจะใช้ฟังก์ชันนี้หรือคำนวณเองก็ได้ เพื่อใช้ในการบังคับห่วงให้อยู่ในตำแหน่งที่ต้องการ

จากนั้นในลูปของ Haptic Process ที่เรียกมายัง Class ที่เรา Subclass มาจาก gstConstraintEffect ซึ่งต่อไปขอเรียกว่า HRingEffect นั้นจะมีการเรียกใช้ฟังก์ชันที่เราต้อง Implement ขึ้นมาเพื่อแสดงแรงที่เราคำนวณได้หรือที่ต้องการ Simulation. นั่นคือ ฟังก์ชัน gstVector HRingEffect::calcEffectForce(void *phantom){...}
ซึ่งให้ Pointer ที่ชี้ไปยัง Phantom Node เป็นพารามิเตอร์เพื่อใช้อ่านข้อมูลต่าง ๆ ของ Phantom และนำข้อมูลมาคำนวณแรงตอบสนองต่อไป

โดยการ Implement เป็นไปตามแนวคิดตามรูป



รูปที่ 5.7 แนวคิดการบังคับห่วงร่วมกัน

จากแนวคิดตามรูป สมมติมีผู้เล่น A และผู้เล่น B ควบคุมคนละด้านของห่วงตามรูป และสื่อสารกันเพื่อบังคับเสมือนหนึ่งเป็นห่วงเดียวกันผ่านเครือข่าย
ภายในฟังก์ชัน calcEffectForce ก็จะมีการ Implement ตามลำดับดังนี้

ตารางที่ 5.14 ลำดับการทำงานภายใน calcEffectForce ของ HRingEffect

1. อ่านค่าตำแหน่งของ Phantom
2. คำนวณ Local View of Global State คือ จุดศูนย์กลางห่วงในมุมของฝ่ายตน ตามรูป 5.1 a)
3. แลกเปลี่ยน Local View of Global State
4. นำ Global State ในความคิดของฝ่ายอื่นมาร่วมคำนวณเพื่อเป็น Global State ในความคิดของฝ่ายตน
5. นำ Global State ที่ได้ไปคำนวณกลับเป็นแรงที่ควรจะแสดงกับ Phantom ในฝ่ายตน

จะสังเกตได้ว่าในลำดับที่ 4 สามารถ Implement ได้ในหลายแบบซึ่งจะมี Effect ต่าง ๆ กัน ทั้งที่เกิดจากการที่ Play-out ไม่ตรงกัน หรือการ Implement Error Control ที่ต่างกัน หรือแม้กระทั่งการคำนวณที่เข้าข้างฝ่ายตัวเอง เป็นต้น ดังนั้นจึงขอไม่สรุปว่าวิธีใดดีที่สุด ทั้งนี้ควรขึ้นอยู่กับผลการทดลองมากกว่า ในการตัดสินเพราะมีพารามิเตอร์ที่เกี่ยวข้องมากมายทั้งทาง Network และทาง Haptic

เช่นในรูป 5.8 b) และ รูป 5.8 c) ที่มีความคิดในการคำนวณหา Global State ที่ต่างกัน เช่น กรณี b ในความเป็นธรรมกับทุกฝ่ายคือควรจะมาพบกันที่ตรงกลาง แต่อาจจะทำให้เกิดการยืดไปเรื่อย ๆ ส่วนในกรณี c ที่ดูเหมือนพยายามทำให้ Global State ของแต่ละฝ่ายปรับเข้ามาหากันให้เร็วที่สุด แต่ก็มี Effect การสวิงตัวจนเกิดแรงสะสมจนเกินที่ Phantom รับได้

<u>การตอบสนองด้วยแรงเมื่อชนถวดคัดโค้งเสมือน</u>

ในการออกแบบเราใช้ Bezier Curve เพราะสร้าง Curve ได้หลากหลายกว่า การใช้การคำนวณทางคณิตศาสตร์แต่เพียงอย่างเดียว เพราะสามารถออกแบบได้โดยกำหนดจุด Control Point สำหรับทฤษฎีหรือวิธีการคำนวณดูได้จาก (Hill. 2001) สำหรับเกม RealLife ใช้ 4 Control Point หรือเรียกว่า Cubic Bezier Spline ในการโปรแกรม

ในการหา Projection ของจุดศูนย์กลางห่วง เราใช้ค่า x ในการคำนวณย้อนกลับหาค่า t แล้วนำ t หาตำแหน่งบน Curve แล้วคำนวณหาว่าชนหรือไม่และแสดงผลแรงตามที่อธิบายในตอนต้น

5.3 สรุป

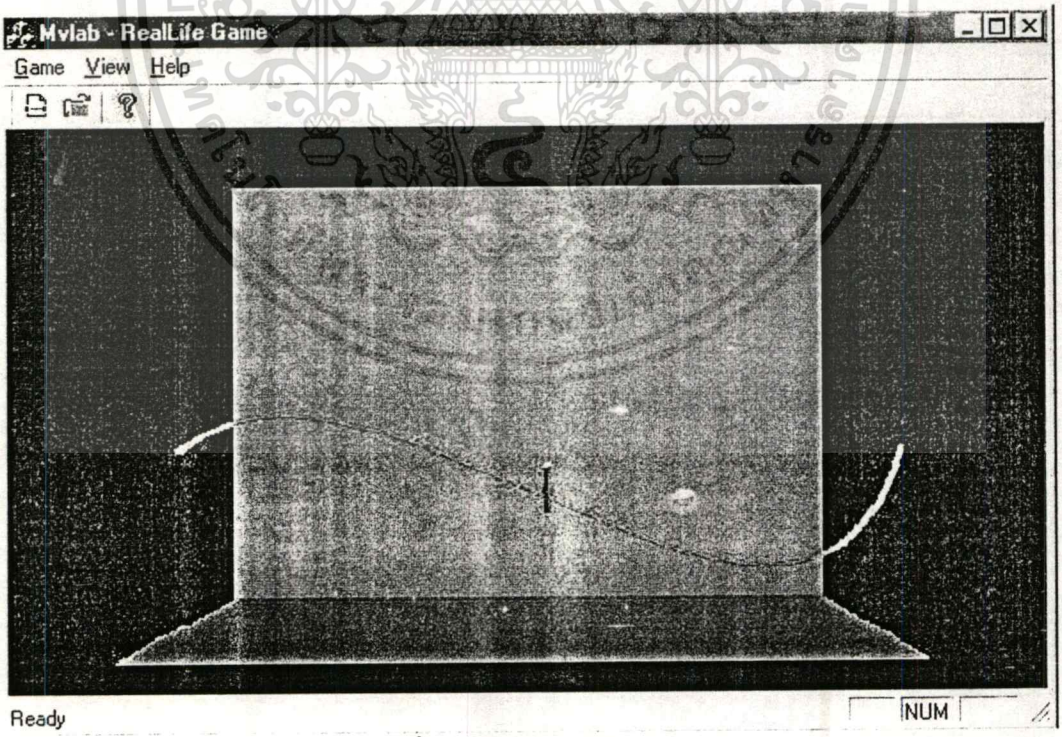
สำหรับระบบย่อยการตอบสนองต่อแรง มีหน้าที่หลัก คือ การสร้าง Haptic Scene Graphs เพื่อใช้ในการ Simulate Haptic Environment ซึ่งสำหรับเกมนี้ก็คือ การสร้าง Effect ของการควบคุมห่วงร่วมกันของสองฝ่ายผ่านการสื่อสารแบบ Multicast และตรวจสอบการชนกับเส้นลวดคัดโค้งที่ใช้ Bezier Curve ในการออกแบบ

บทที่ 6

โปรแกรมเกม

6.1 เกม RealLife

เกม RealLife เป็นเกมคอมพิวเตอร์ที่ร่วมเล่นกันสองฝ่ายผ่านเครือข่ายคอมพิวเตอร์ โดยแต่ละฝ่ายใช้อุปกรณ์ประเภทตอบสนองต่อแรง (Force Feedback Device) ในการควบคุมห่วงวงกลมเสมือนร่วมกันเพื่อให้ร้อยผ่านไปตามแนวลวดดัดโค้ง ถ้าหากผู้เล่นฝ่ายหนึ่งมีการเคลื่อนที่ก็จะมีแรงส่งผลไปยังผู้เล่นอีกฝ่ายหนึ่งหากตำแหน่งของทั้งสองไม่สัมพันธ์กัน ถ้าห่วงชนกับลวดก็จะมีแรงปฏิกิริยาตอบสนอง โดยผู้เล่นแต่ละฝ่ายจะเห็นเส้นลวดเสมือนและตำแหน่งห่วงที่ตรงกัน ดังรูป



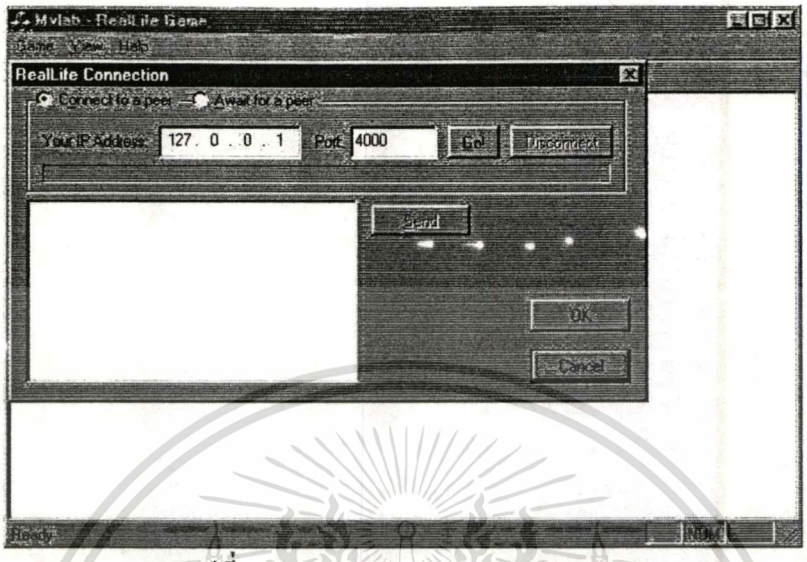
รูปที่ 6.1 ลักษณะหน้าตาเกม

6.2 ขั้นตอนการเล่นเกม

1) เมื่อเปิดโปรแกรมแล้วจะเป็นดังรูป ซึ่งมีเมนู Game, View, และ Help โดยเมนู Game มีเมนูย่อย Game->Play... และ Game->Options...โดยเมนู Play ใช้ในการเริ่มเล่นเกม และเมนู Options ใช้ในการเซตค่าต่าง ๆ ของโปรแกรม ซึ่งจะกล่าวต่อไป โดยการเรียกใช้อาจจะใช้ Toolbar คือ 2 ปุ่มทางซ้ายของ Toolbar หรือใช้ Short Cut Key คือ Ctrl+P และ Ctrl+O ตามลำดับ

รูปที่ 6.2 ลักษณะหน้าตาเกมตอนเริ่มต้น

2) เมื่อต้องการเริ่มเล่นหลังจากเซตค่าต่าง ๆ เรียบร้อยแล้ว (ซึ่งจะอธิบายการเซตค่าภายหลัง) ให้เลือกเมนู Game->Play... ซึ่งจะปรากฏ Dialog ในการเชื่อมต่อการสื่อสาร ดังรูป โดยผู้เล่นต้องเลือกว่าจะเป็นฝ่าย Server หรือ Client ฝ่ายใดฝ่ายหนึ่ง โดยเลือกจาก Radio button ระหว่าง Connect to a peer ซึ่งหมายถึงเป็น Client กับ Await for a peer ซึ่งหมายถึงเป็นฝ่าย Server โดยฝ่าย Server ต้องทำการเปิดรอก่อน Client ถึงจะติดต่อได้

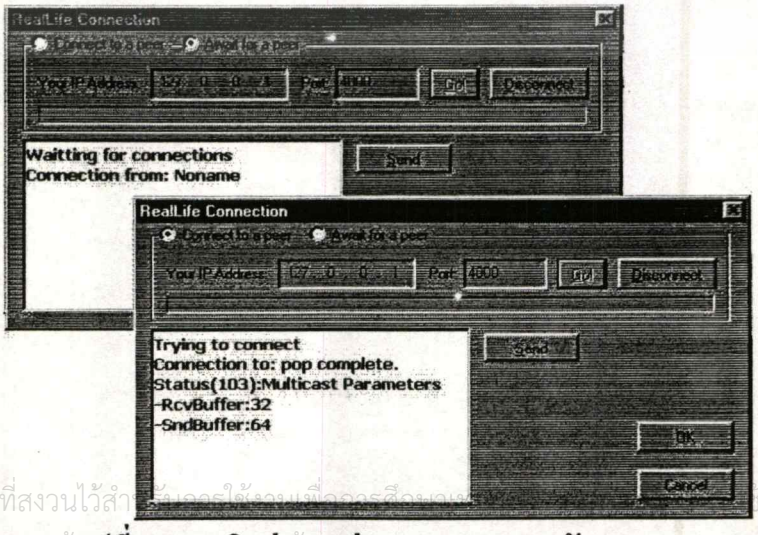


รูปที่ 6.3 RealLife Connection Dialog

โดยฝ่าย Server เลือก Await for a peer จากนั้น กำหนด Port ที่ต้องการใช้ แล้วกดปุ่ม Go! ซึ่งสถานะต่าง ๆ ก็จะปรากฏใน Edit Box ด้านล่างซ้ายว่ากำลังรอการติดต่อจาก Client

ถ้าเป็นฝ่าย Client ให้เลือก Connect to a peer แล้วระบุ IP Address และ Port Number ของ Server ที่ต้องการติดต่อ จากนั้นจึงกดปุ่ม Go! สถานะต่างก็จะปรากฏใน Edit Box ด้านล่างซ้ายเช่นเดียวกัน แต่ถ้าติดต่อ Server ได้ ก็จะบอกว่า Connect กับ Server ได้ เมื่อติดต่อได้แล้วให้กดปุ่ม OK แล้วรอการควบคุมจาก Server

เช่นดังรูปที่ 6.4 Dialog บนเป็นของฝ่าย Server และ Dialog ล่างเป็นของฝ่าย Client จะเห็นว่ามีการบอกสถานะว่า Server ส่งค่าอะไรมาบ้าง



รูปที่ 6.4 การติดต่อระหว่าง Game Server กับ Client

เมื่อฝ่าย Server ได้รับการติดต่อจาก Client แล้วซึ่งทราบจาก Edit Box ด้านล่างซ้ายที่คอยบอกสถานะ ให้กดปุ่ม OK ซึ่งจะปรากฏ Dialog บอกให้เซตตำแหน่งของ PHANToM ให้อยู่ในตำแหน่ง Neutral Point ซึ่งขึ้นกับรุ่นของ PHANToM ให้ดูคู่มือของรุ่นนั้นเป็นหลัก หรือในบางรุ่นอาจจะเซตอัตโนมัติ จากนั้นให้กด OK จากนั้นโปรแกรมในฝ่าย Server จะทำการส่งข้อมูลไปบอกฝ่าย Client ว่าจะเริ่มเล่นแล้ว ซึ่งก็จะปรากฏ Dialog ในฝ่าย Client เช่นเดียวกันว่าให้เซตตำแหน่งของ PHANToM เมื่อ Client กด OK โปรแกรมก็จะสื่อสารกันในแบบ Multicast และเริ่มเล่นได้ดังรูปที่ 6.1

3) ผู้เล่นทั้งสองฝ่ายต้องช่วยกันควบคุมห่วงไม่ให้ชนเส้นลวดดัดโค้งเสมือน ถ้าเกิดการชนจะมีแรงตอบสนองเหมือนว่าเป็นเส้นลวดจริง ๆ และลวดก็จะเปลี่ยนจากสีฟ้าเป็นสีแดง

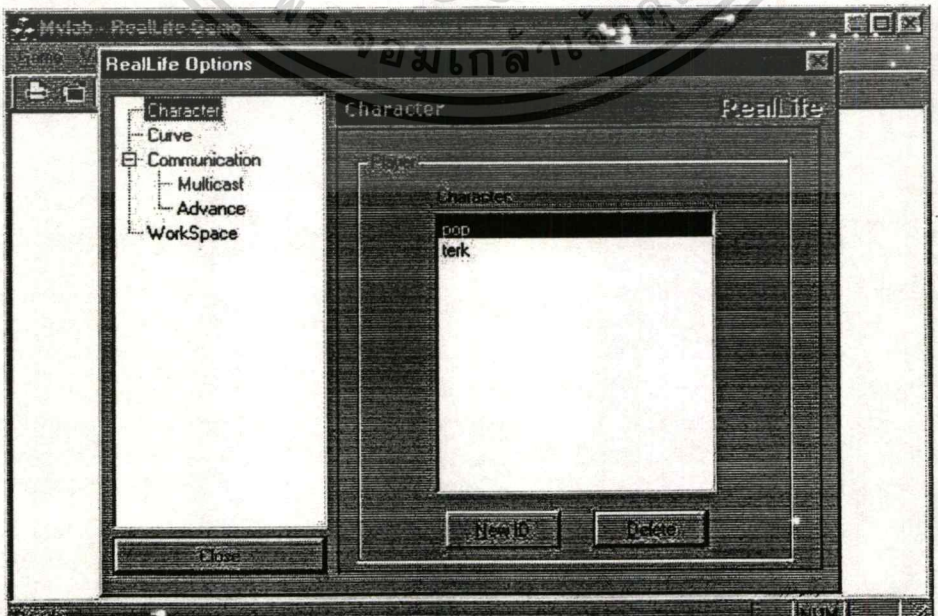
4) เมื่อต้องการเลิกเล่นให้ปิดโปรแกรม โดยเลือก Game->Exit หรือปุ่ม Close

6.3 การกำหนดค่าของโปรแกรม

ในการกำหนดค่าต่าง ๆ ของโปรแกรมให้เลือก Game->Options... หรือกด Ctrl+O ซึ่งจะปรากฏ Dialog ในการเซตค่าดังรูป ซึ่งประกอบด้วย Character, Curve, Communication, Multicast, Advance, และ Workspace Dialog

6.3.1 การตั้งชื่อ

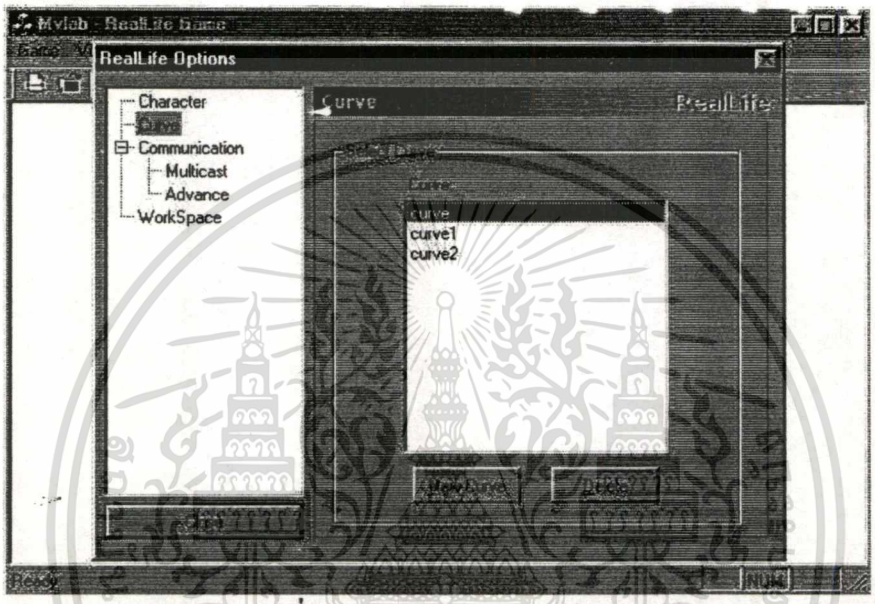
ใช้ในการเลือกชื่อ, ตั้งชื่อ, และลบชื่อของผู้เล่น



รูปที่ 6.5 การตั้งชื่อ

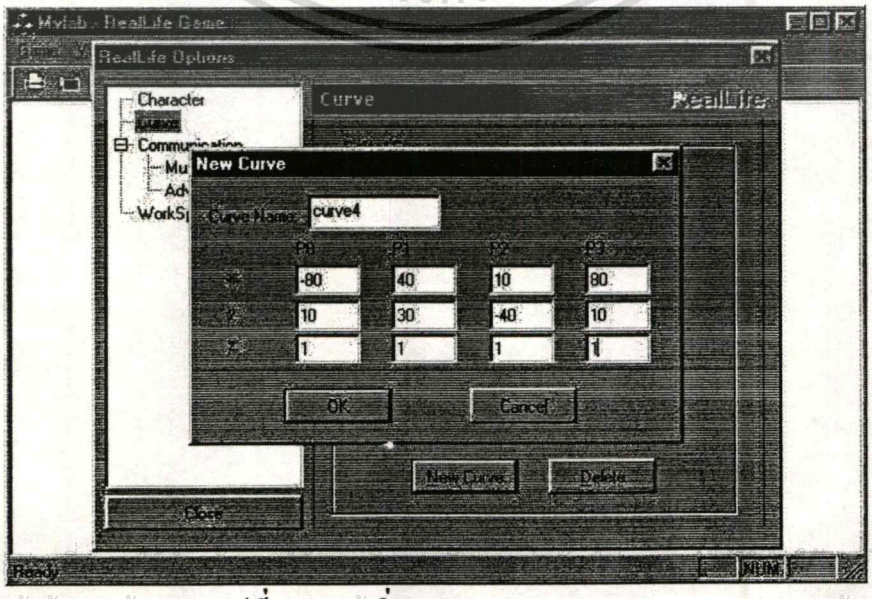
6.3.2 การเลือกชุดของ Curve

ใช้ในการเลือก Curve, สร้าง Curve, และลบ Curve โดย Curve ที่ใช้คือ Cubic Bezier Curve ซึ่งต้องกำหนด Control Point 4 จุดดังรูปที่ 6.7



รูปที่ 6.6 การเลือกชุดของ Curve

ในการเพิ่มชุดของเส้นลวดคัดโค้งเสมือน ซึ่งใช้ Cubic Bezier Curve ในการออกแบบต้องกำหนด Control Point 4 จุด คือ P0, P1, P2, P3 และกำหนดชื่อ Curve ในช่อง Curve Name.

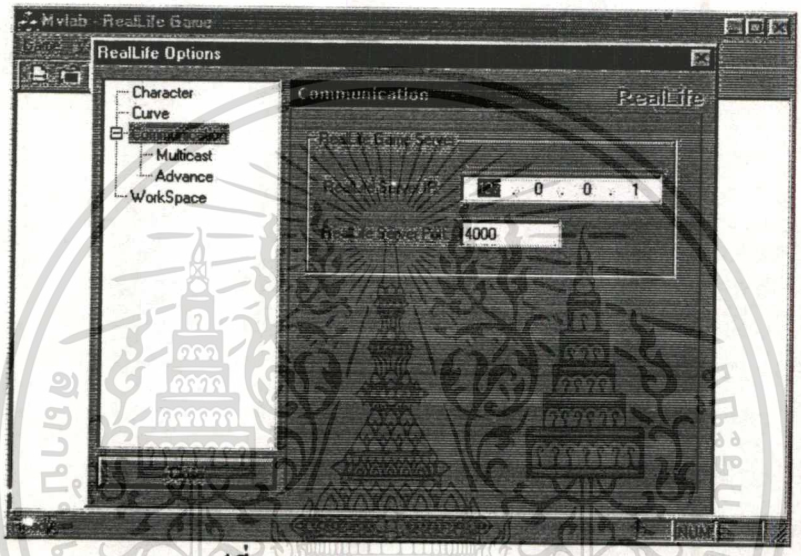


รูปที่ 6.7 การเพิ่ม Cubic Bezier Curve

6.3.3 การกำหนดค่าที่ใช้ในการสื่อสาร

6.3.3.1 การกำหนด Game Server

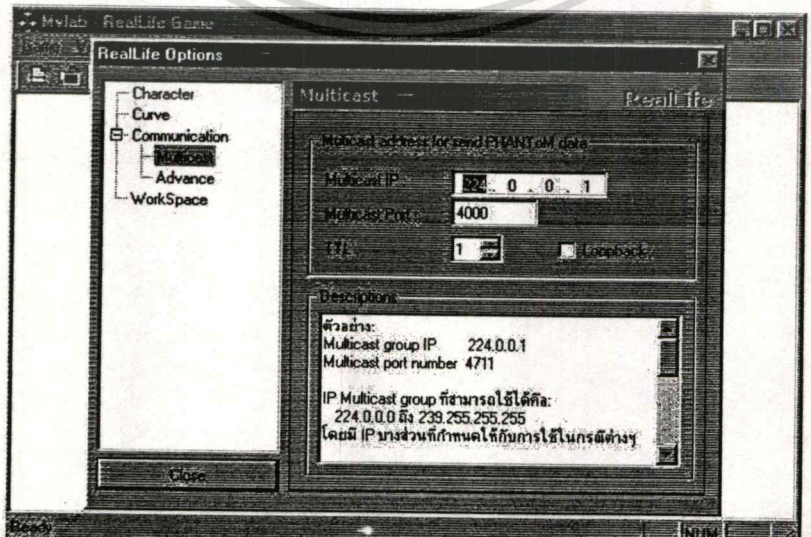
เป็นการกำหนด IP Address และ Port Number ของ Game Server เพื่อที่เวลาจะเล่นจะได้ไม่ต้องป้อนค่าทุกครั้งใน RealLife Connection Dialog ที่กล่าวมาในขั้นตอนการเล่นเกม



รูปที่ 6.8 การกำหนด Game Server

6.3.3.2 การกำหนด Multicast

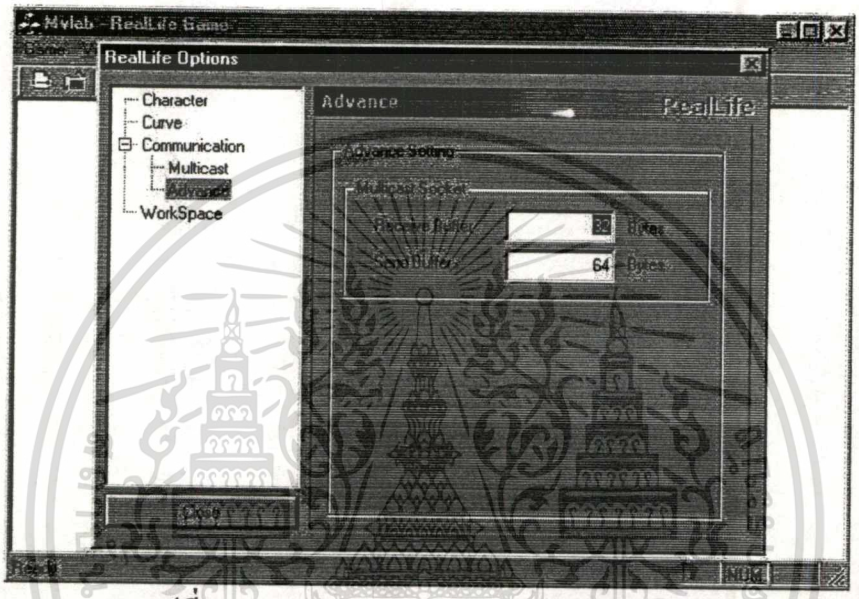
ใช้ในการกำหนด Multicast Group Address และ Group Port ที่จะใช้ในการสื่อสาร



รูปที่ 6.9 การกำหนด Multicast

6.3.3.3 การกำหนด Buffer ของส่วน Multicast

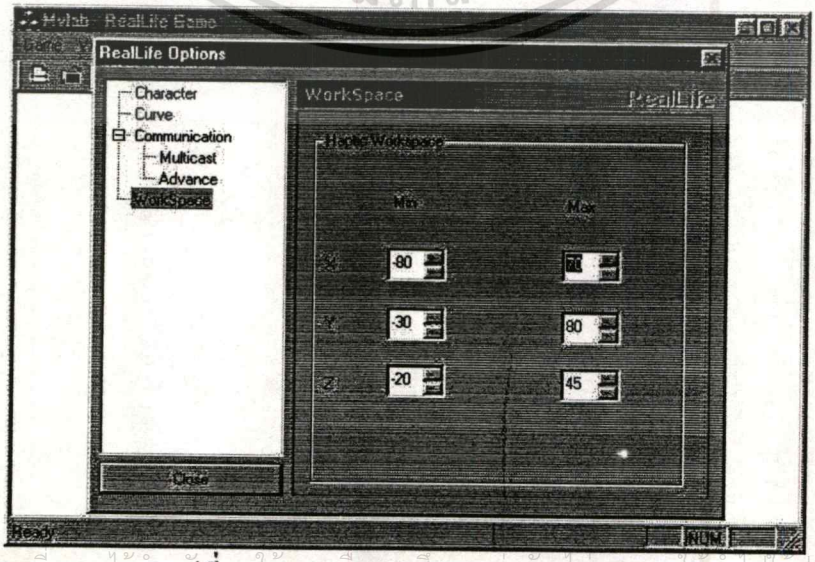
เนื่องจากการกำหนดขนาดของ Buffer ในการส่งและรับในส่วนของ Multicast มีผลต่อประสิทธิภาพของโปรแกรม จึงมีการเพิ่มการเซตค่านี้เพื่อการทดลองเท่านั้น



รูปที่ 6.10 การกำหนด Buffer ของส่วน Multicast

6.3.4 การกำหนด Workspace ของเกม

เป็นการกำหนด Haptic Workspace ของเกมว่ามีขนาดเท่าใด ซึ่งขึ้นกับรุ่นของ PHANToM



รูปที่ 6.11 การกำหนด Haptic Workspace

6.4 สรุป

สำหรับเกม RealLife ที่ได้พัฒนาขึ้นมา วิธีการเล่นก็เป็นดังที่กล่าวมาแล้ว โดยมีการสื่อสารแบบ TCP ในการแลกเปลี่ยนข้อมูลควบคุมเกม เช่น การกำหนดค่า Buffer ในส่วนของ Multicast, การควบคุมการเริ่มเล่นเกม เป็นต้น และมีการสื่อสารแบบ Multicast ในการแลกเปลี่ยนข้อมูลในการบังคับห่วงดังที่กล่าวมาในบทก่อน ๆ โดยขั้นแรกสื่อสารกันแบบ TCP เพื่อให้ Server ซึ่งเป็นฝ่ายกำหนดสถานะต่าง ๆ ส่งข้อมูลไปยัง Client เพื่อให้เข้าใจตรงกัน และเป็นฝ่ายควบคุมการเริ่มเล่นเกม จากนั้นจึงเป็นหน้าที่ของการสื่อสารแบบ Multicast ในการแลกเปลี่ยนข้อมูลเพื่อการบังคับห่วงกลมเสมือนร่วมกันต่อไป

บทที่ 7
บทสรุป

7.1 สรุป

การพัฒนาโครงการเกมนี้ เป็นการศึกษาและพัฒนาใน 3 ด้าน คือ 1) ด้านกราฟิก 2) ด้านการสื่อสารผ่านเครือข่าย และ 3) ด้านการตอบสนองต่อแรง โดยพัฒนาบนระบบ Windows และใช้ Visual C++ เป็น Tool ในการพัฒนา โดยใช้ MFC SDI เป็น Framework หลัก ซึ่งการพัฒนาทางด้านกราฟิกได้ใช้ OpenGL เป็น Graphics Library, ส่วนการพัฒนาทางด้านการสื่อสารผ่านเครือข่ายใช้ Windows Socket และ MFC เป็นหลัก, และส่วนด้านการตอบสนองต่อแรงใช้ GHOST SDK ร่วมกับ PHANToM ในการพัฒนา

ในด้านกราฟิก สิ่งที่พัฒนาหรือ Implement ขึ้นมาก็คือ การเพิ่มฟังก์ชันทางกราฟิกของ GHOST SDK เพื่อให้ Haptic Object ก็เป็น Graphics Object ในตัวด้วย

ในด้านการสื่อสารผ่านเครือข่าย สิ่งที่พัฒนาหรือ Implement ขึ้นมาก็คือ การสื่อสารในแบบ TCP และการสื่อสารในแบบ Multicast

ในด้านการตอบสนองต่อแรง สิ่งที่พัฒนาหรือ Implement ขึ้นมาก็คือ การจำลองการควบคุมห่วงกลมเสมือนร่วมกับผ่านเครือข่าย และการจำลองการตอบสนองต่อแรงของห่วงกลมเสมือนกับเส้นถวดดัดโค้งเสมือน ซึ่งใช้ Bezier Curves ในการออกแบบ

จากการพัฒนาที่ผ่านมาก็ถือว่าสำเร็จในระดับหนึ่ง เมื่อเทียบกับเวลาที่ใช้และทักษะที่มีอยู่ ถึงแม้ว่าประสิทธิภาพของเกม อาจจะดูไม่ดีนัก

7.2 ข้อจำกัดและปัญหาของโครงการ

สำหรับข้อจำกัดของโครงการนี้ ก็คือ 1) เรื่องเวลา ซึ่งก็ต้องเข้าใจว่าชีวิตมนุษย์แปรผกผันกับเวลา ดังนั้นโครงการนี้เมื่อใช้เวลาพอสมควรแล้วก็คงต้องยุติ ถึงแม้ว่าโครงการจะไม่สำเร็จดีเยี่ยมตามต้องการ 2) เรื่องทักษะ ซึ่งก็คือทักษะในการเขียนโปรแกรม การออกแบบระบบ และการทำเอกสาร ซึ่งผู้พัฒนาก็พยายามพัฒนาอยู่ 3) เรื่องทฤษฎี ซึ่งสำหรับโครงการนี้ก็พยายามหาที่อ้างอิงบ้าง คิดเองบ้าง ดังนั้นโครงการนี้จึงเป็นในลักษณะ Pilot Project มากกว่าเป็น Project ธรรมดา.

สำหรับปัญหาของโครงการนี้ ก็คือ 1) แนวทางการพัฒนา ซึ่งตามปกติก็คือ สัมมนา 1, สัมมนา 2, และโครงการพัฒนาระบบงาน รวมเป็น 3 เทอมในการพัฒนา ซึ่งก็คือการศึกษาในเรื่องที่จะพัฒนาต่อเนื่องกันมา แต่โครงการนี้มีการเปลี่ยนแปลงในระหว่างการพัฒนาคือ เปลี่ยนจากการพัฒนาบน IRIX มาพัฒนาบน Windows แทน ซึ่งทำให้โครงการเสียเวลาไปพอสมควร แต่ก็ทำให้ได้เทคนิคบางอย่าง และความสะดวกในการพัฒนาบน Windows มาทดแทน

ปัญหาเรื่องที่ 2) คือ ปัญหาด้านการออกแบบระบบ ในการออกแบบจะเห็นได้ว่า ถ้าเป็นโครงการทางด้านฐานข้อมูล ก็จะมีรูปแบบการออกแบบฐานข้อมูล เช่น ERD, Data Flow Diagrams, Context Diagrams เป็นต้น และก็รูปแบบการออกแบบโปรแกรมคือส่วน User Interfaces เท่านั้น เป็นส่วนใหญ่ แต่โครงการนี้เป็นโครงการทางด้านโปรแกรม ดังนั้นการออกแบบก็จะต้องเน้นในรูปแบบการออกแบบโปรแกรมมากกว่า ซึ่งถ้าเป็น Structure Programming ก็จะใช้การอธิบาย Algorithms และการเขียน Flow Chart แต่ในระบบที่ใหญ่และซับซ้อนในเชิงการโปรแกรมเช่นโครงการนี้ และระบบการโปรแกรมในปัจจุบันจะใช้ Object-Oriented Programming ในแก้ปัญหาเป็นหลัก ซึ่งรูปแบบในการออกแบบยังไม่มีอันใดที่เป็นที่ยอมรับกันโดยทั่วไป ถึงแม้จะมีการรวมรูปแบบต่าง ๆ และพัฒนาให้เป็นรูปแบบมาตราฐานเช่น UML แต่โครงการนี้ก็ขอใช้เพียง CRC และ Collaboration Graphs เท่านั้นในการอธิบายการออกแบบ ซึ่ง CRC และ Collaboration Graphs ที่ได้แสดงในเอกสารนี้ก็อาจจะผิดเพี้ยนไปจากต้นแบบเดิมไปบ้าง เพื่อความเหมาะสม

ปัญหาเรื่องที่ 3) คือการ Implement ของ Application ที่มีลักษณะ Haptic-Interactive, Real-time, Networked-Collaborative ดังเช่นโครงการนี้ ซึ่งแต่ละคำก็แสดงปัญหาของตัวเอง และมีปัญหาในการ Implement พอสมควร และเมื่อมารวมกันเป็น Application เดียวกันอีก ปัญหาและข้อจำกัดก็ต้องมีมากเป็นธรรมดา

7.3 ข้อเสนอแนะ

สำหรับโครงการพัฒนาระบบงานนี้ยังมีส่วนที่น่าจะพัฒนาได้ดีกว่านี้ เช่น การสื่อสารแบบ Multicast (บน UDP/IP) ที่ใช้ Windows Sockets ในแบบ Asynchronous ควรจะใช้เป็นแบบ Synchronous กับ Thread น่าจะทำให้การรับและส่งข้อมูลได้รวดเร็วกว่านี้, Transmission Control ในส่วน Networked-Collaborative น่าจะเป็นแบบ Bucket Synchronization และควรพัฒนา Error Control ที่เหมาะสมที่จะใช้กับลักษณะ Haptic-Interactive(Force Feedback) ถ้าเป็นไปได้.

บรรณานุกรม

Douglas, Young A. 1995. **Object Oriented Programming with C++ and OSF Motif.** 2nd ed. New Jersey, Prentice-Hall.

Hill, F. S. Jr. 2001. **Computer Graphics Using OpenGL.** 2nd ed. New Jersey, Prentice-Hall.

Kurose, J. et al. 1999. "End-to-end Transmission Control Mechanisms for Multiparty Interactive Applications on the Internet." **Proc. IEEE Infocom'99.**

SensAble Technologies. 1998. **GHOST SDK Programmer's Guide Version 2.** Cambridge, SensAble Technologies.

ภาคผนวก

ภาคผนวก ก

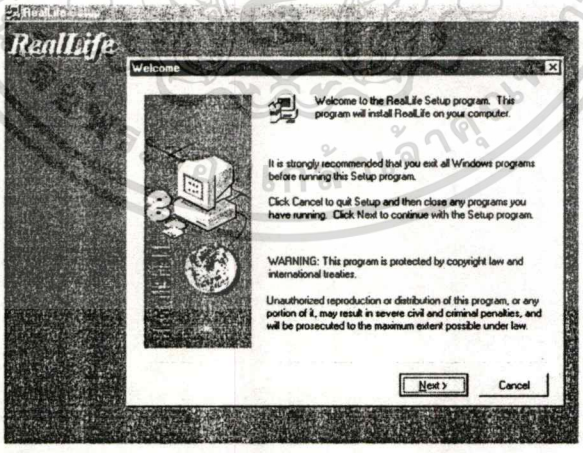
การติดตั้งโปรแกรมเกม RealLife

ก.1 ความต้องการระบบขั้นต่ำ

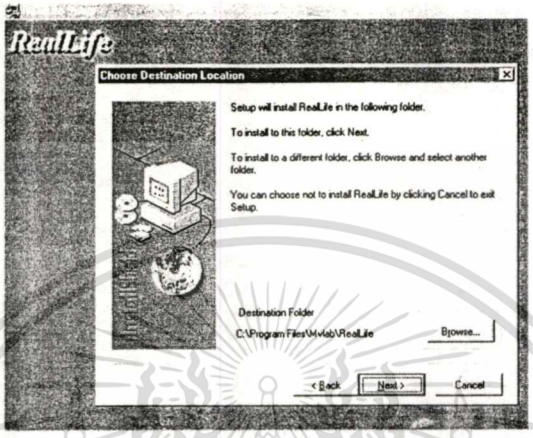
1. Pentium-class PC 166 MHz at minimum
2. Windows NT 4.0 + Service Pack 6 at minimum
3. PHANToM
4. PHANToM Device Driver Version 2.1 at minimum
5. Ethernet LAN 10/100 Mbits/sec

ก.2 ขั้นตอนการติดตั้ง

1) เปิดโปรแกรม setup.exe กด Next



2) เลือก Directory ที่จะติดตั้ง แล้วกด Next



3) กด Finish จบขั้นตอนการติดตั้ง

การเรียกใช้โปรแกรมคือ Start->Programs->Mvlab->RealLife

ภาคผนวก ข

รหัสโปรแกรมเกม RealLife

สำหรับรหัสโปรแกรมเกม RealLife ในภาคผนวกนี้ เลือกเฉพาะโปรแกรมบางส่วนที่สำคัญและเป็นหัวใจหลักเท่านั้น เพราะรหัสโปรแกรมทั้งหมดมีจำนวนมากจึงไม่สามารถพิมพ์ทั้งหมดได้

```
//////////////////////////////////////////////////////////////////
// HPhantom.h:
//////////////////////////////////////////////////////////////////
#ifndef __HPHANTOM_H__
#define __HPHANTOM_H__

#include <gstPHANToM.h>
#include "HGraphics.h"

class HPhantom :public gstPHANToM ,public HGraphics
{
public:
        HPhantom(char* PHANTOMName,int resetEncoders = TRUE);
        virtual ~HPhantom();
        void SetPhantomSize(float PhantomSize=1.f);
        virtual void PreDraw();
        virtual void HandleBuildList();
        virtual void OnGraphicCB(gstTransform *node, void *callData);
protected:
        float m_PhantomSize;
        GLUquadricObj* quadObj;
private:
        static void HandleGraphicCB(gstTransform* node,void* callData,void* clientData);

//////////////////// GHOST Standard ////////////////////
public:
        virtual gstType     getTypeId() const;
        virtual gstBoolean isOfType(gstType type) const;
        static  gstType     getClassTypeId();
        static  gstBoolean  staticIsOfType(gstType type);
protected:
        virtual void putInSceneGraph();
        virtual void removeFromSceneGraph();
private:
        static gstType HPhantomClassTypeId;
//////////////////// GHOST Standard ////////////////////
};

#endif // __HPHANTOM_H__
//////////////////////////////////////////////////////////////////
// HPhantom.cpp:
//////////////////////////////////////////////////////////////////
#include "stdafx.h"
#include "HPhantom.h"

HPhantom::HPhantom(char* PHANTOMName,int
resetEncoders):gstPHANToM(PHANTOMName,resetEncoders)
{
        m_Material.SetAmbient(1.0, 0.2, 0.4, 1.0);
        m_Material.SetDiffuse(1.0, 0.2, 0.4, 1.0);
        m_PhantomSize = 1.f;
   quadObj = gluNewQuadric();
        setGraphicsCallback(&HPhantom::HandleGraphicCB,this);
}

HPhantom::~HPhantom()
{
        setGraphicsCallback(NULL,NULL);
   gluDeleteQuadric(quadObj);
        removeFromSceneGraph();
```

```
}

void HPhantom::SetPhantomSize(float PhantomSize/*=1.f*/)
{
        m_PhantomSize = PhantomSize;
        m_Transform.SetScale(&CVector3d(m_PhantomSize,m_PhantomSize,m_PhantomSize));
        Draw();
}

void HPhantom::PreDraw()
{
        glScalef(m_PhantomSize,m_PhantomSize,m_PhantomSize);
}

void HPhantom::HandleBuildList()
{
   gluQuadricDrawStyle(quadObj,GLU_FILL);
   gluQuadricNormals(quadObj,GLU_SMOOTH);
        glEnable(GL_CULL_FACE);
   glEnable(GL_DEPTH_TEST);
   glDepthFunc(GL_LEQUAL);
        gluSphere(quadObj,m_PhantomSize,16,16);
}

void HPhantom::OnGraphicCB(gstTransform *node, void *callData)
{
   gstPoint pt;
        getPosition_WC(pt);
        m_Transform.SetTranslation(&CVector3d(pt.x(),pt.y(),pt.z()));
        getScaleFactor(pt);
        m_Transform.SetScale(&CVector3d(pt.x(),pt.y(),pt.z()));
        getCumulativeTransform().getRotationAngles(pt);
        m_Transform.SetRotation(&CVector3d(pt.x()*180/M_PI,pt.y()*180/M_PI,pt.z()*180/M_PI));
        Draw();
}

void HPhantom::HandleGraphicCB(gstTransform *node, void *callData,void *clientData)
{
        ((HPhantom*)clientData)->OnGraphicCB(node,callData);
}

//////////////////// GHOST Standard ////////////////////
gstType HPhantom::HPhantomClassTypeId;

gstType HPhantom::getTypeId() const
{
        return getClassTypeId();
}

gstBoolean HPhantom::isOfType(gstType type) const
{
        return staticIsOfType(type);
}

gstType HPhantom::getClassTypeId()
{
        gstAssignUniqueId(HPhantomClassTypeId);
        return HPhantomClassTypeId;
}
```

```
gstBoolean HPhantom::staticIsOfType(gstType type)
{
        if(type==getClassTypeId())return TRUE;
        else return(gstPHANToM::staticIsOfType(type));
}

void HPhantom::putInSceneGraph()
{
        gstPHANToM::putInSceneGraph();
        // Your Code
}

void HPhantom::removeFromSceneGraph()
{
        // Your Code
        gstPHANToM::removeFromSceneGraph();
}
////////////////////// GHOST Standard //////////////////////


// HRingEffect.h: interface for the HRingEffect class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_HRINGEFFECT_H__271E1F73_D436_11D6_AF4A_00036D1FDFF6__INCLUDED_)
#define AFX_HRINGEFFECT_H__271E1F73_D436_11D6_AF4A_00036D1FDFF6__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

#include <gstConstraintEffect.h>
#include <gstPHANToM.h>
#include "..\Comm\McastSocket.h"
#include "..\Lib3d\Material3d.h"
#include "..\Lib3d\Vector3d.h"
#include "..\Lib3d\Bezier3d.h"

class HRingEffect :public gstConstraintEffect
{
public:
        HRingEffect();
        virtual ~HRingEffect();
public:
        void SetMcast(CMcastSocket* pMcast){ m_pMcast=pMcast; }
        void DrawGL();
        CBezier3d                   m_bezier;
public:
        virtual gstBoolean start();
        virtual void                stop();
        virtual gstVector  calcEffectForce(void* phantom);
        virtual gstVector  calcEffectForce(void* phantom,gstVector& torques);
        gstVector                   calcEffectForceV2(void* phantom);
        gstVector                   calcRingForce(gstPoint pt);
public:
        BOOL LoadCurve(LPCTSTR lpszFileName);
        gstPoint m_ptMyGlobe,m_ptMyGlobeRing;
        gstPoint m_ptOtherGlobe[2]; // Future 0:Sum,1..:Individual
        gstPoint m_ptCgRing;
        gstPoint m_ptPhantom;
```

```
        float m_LocalWeight;
        float m_GlobalWeight;
        int m_nHandle;
        float m_RadiusRing;
        BOOL m_bCollide;
private:
        gstPoint        m_ptRecv;
        CMcastSocket*   m_pMcast;
private: // OpenGL
        GLUquadricObj*  m_quadObj;
        CMaterial3d             m_mat[3];
        gstPoint        m_ptCenterRing;
        void doughnut(GLfloat r,GLfloat R,GLint nsides,GLint rings);
};

#endif // !defined(AFX_HRINGEFFECT_H__271E1F73_D436_11D6_AF4A_00036D1FDFF6__INCLUDED_)
// HRingEffect.cpp: implementation of the HRingEffect class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "HRingEffect.h"
#include <math.h>

#ifdef _DEBUG
#undef THIS_FILE
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////

HRingEffect::HRingEffect()
:m_pMcast(NULL)
{
        m_nHandle = 1;
        m_GlobalWeight = 0.5;
        m_LocalWeight = 1;
        m_bCollide = FALSE;
        // OpenGL
        m_RadiusRing = 8;
   m_quadObj = gluNewQuadric();
        gluQuadricDrawStyle(m_quadObj,GLU_FILL);
   gluQuadricNormals(m_quadObj,GLU_SMOOTH);
        m_mat[0].SetAmbient(0.4f, 0.2f, 1.0f, 1.0f); // MyGlobe
        m_mat[0].SetDiffuse(0.4f, 0.2f, 1.0f, 1.0f);
        m_mat[1].SetAmbient(0.4f, 1.0f, 0.2f, 1.0f); // OtherGlobe[1]
        m_mat[1].SetDiffuse(0.4f, 1.0f, 0.2f, 1.0f);
}

HRingEffect::~HRingEffect()
{
        m_pMcast = NULL;
}

gstBoolean HRingEffect::start()
{
        return gstConstraintEffect::start();
```

```
}

void HRingEffect::stop()
{
	gstConstraintEffect::stop();
}

gstVector HRingEffect::calcEffectForce(void *phantom,gstVector& torques)
{
	torques = gstVector(0,0,0);
	return calcEffectForce(phantom);
}

gstVector HRingEffect::calcEffectForce(void *phantom)
{
	if(!active){
		return gstVector(0,0,0);
	}
	return calcEffectForceV2(phantom);
}

gstVector HRingEffect::calcEffectForceV2(void *phantom)
{
	static BOOL bFirst = FALSE;
	static CString msg;
	static float fx,fy,fz;
	static gstPoint tmp,diff;
	static gstPoint ptRcv;
	static gstVector force;
	force.init(0,0,0);
	gstPHANToM* p = (gstPHANToM*)phantom;
	p->getPosition_WC(m_ptPhantom);
	// m_ptCgRing <-- m_ptPhantom
	m_ptCgRing = m_ptPhantom;
	m_ptCgRing.sety(m_ptCgRing.y()+m_RadiusRing*m_nHandle);
	if(m_pMcast){ // Send Local View of Global State
		msg.Format("%f %f %f",m_ptCgRing.x(),m_ptCgRing.y(),m_ptCgRing.z());
		if(!m_pMcast->SendTo(msg,msg.GetLength()))
			TRACE("[HRingEffect::calcEffectForceV1] SendTo Fail\n");
	}
	if(m_pMcast){
		if(m_pMcast->bDataReceived){
			m_pMcast->bDataReceived = FALSE;
			sscanf(m_pMcast->m_strBuffer,"%f %f %f",&fx,&fy,&fz);
			ptRcv.init(fx,fy,fz);
			// Separate
			m_ptOtherGlobe[1] = ptRcv; // 2 Joints
			bFirst = TRUE;
		}else{
		////////////////////// Error Control
			if(bFirst){
				//m_ptOtherGlobe[1] = ?
			}else{
				m_ptOtherGlobe[1].init(0,0,0);
			}
		}
		////////////////////// Calc Global State
		// Calc of all other Global
		m_ptOtherGlobe[0] = m_ptOtherGlobe[1];
		m_GlobalWeight = 0.51;
```

```
                diff = m_ptOtherGlobe[0] - m_ptCgRing;
                tmp = m_ptCgRing + diff*m_GlobalWeight;
                m_ptMyGlobe = tmp;
        }
////////////////// Play out //////////////////
        // Calc Ring
        m_ptCenterRing = m_ptMyGlobe;
        // m_ptMyGlobeRing <-- m_ptMyGlobe
        m_ptMyGlobeRing = m_ptMyGlobe;
        m_ptMyGlobeRing.sety(m_ptMyGlobeRing.y()-m_RadiusRing*m_nHandle);

        pointConstraint = m_ptMyGlobeRing;
        force = gstConstraintEffect::calcEffectForce(phantom);
        if(m_pMcast){
                force = force + calcRingForce(m_ptMyGlobe)/2;
        }
        return force;
}

gstVector HRingEffect::calcRingForce(gstPoint pt)
{
        // Calc Collision between Ring and Bezier Curve
        static gstVector force;       // Force Output
        force.init(0,0,0);
        static gstPoint ptMe;         // Center of Ring
        ptMe = pt;                                  // pt is Center of Ring
        double t;                           // t in P(t)
        static gstVector vtDiff;
        static gstPoint  ptOnCurve;
        static CVector3d vrTmp;
        // Detect in Range
        if(ptMe.x()>m_bezier.m_P[0].x && ptMe.x()<m_bezier.m_P[3].x){
                // Find x-plane projection on Bezier Curve
                // Solve t from x in cubic polynomial
                t = m_bezier.GetTfromX(ptMe.x()); // t > 0 and t = real number
                // Find point on bezier curve from t
                vrTmp = m_bezier.PointOnCurve(t);
                ptOnCurve.init(0,vrTmp.y,vrTmp.z);

                // Detect collision
                ptMe.setx(0);
                vtDiff = ptMe - ptOnCurve;
                float over = vtDiff.norm()-m_RadiusRing;
                if(over>0){ // Collision
                        force = over*(-1)*vtDiff.normalize();
                        m_bCollide = TRUE;
                }else{                  // Non collision
                        m_bCollide = FALSE;
                }
        }
        return force;
}

void HRingEffect::DrawGL()
{
        gstPoint* pt;
        // Draw Bezier Spline Curve
        if(m_bCollide){
                CMaterial3d::SetMaterial(CMaterial3d::RED_PLASTIC);
        }
```

```
        m_bezier.DrawGL();
        // Other Peer
        gstPoint tmp;
        tmp = m_ptOtherGlobe[1];
        tmp.sety(tmp.y()+m_RadiusRing*m_nHandle);
        pt = &tmp;
        glPushMatrix();
        m_mat[1].Show();
        glTranslatef(pt->x(),pt->y(),pt->z());
        gluSphere(m_quadObj,1,10,10);
        glPopMatrix();
        // Draw Ring
        pt = &m_ptCenterRing;
        glPushMatrix();
        glTranslatef(pt->x(),pt->y(),pt->z());
        glRotatef(90,0.f,1.f,0.f);                          // Fix Orientation
        doughnut(0.5,m_RadiusRing,20,20);
        glPopMatrix();
}

void HRingEffect::doughnut(GLfloat r, GLfloat R, GLint nsides, GLint rings)
{
        int i, j;
        GLfloat theta, phi, theta1;
        GLfloat cosTheta, sinTheta;
        GLfloat cosTheta1, sinTheta1;
        GLfloat ringDelta, sideDelta;

        ringDelta = 2.0 * M_PI / rings;
        sideDelta = 2.0 * M_PI / nsides;

        theta = 0.0;
        cosTheta = 1.0;
        sinTheta = 0.0;
        for(i=rings-1;i>=0;i--){
                theta1 = theta + ringDelta;
                cosTheta1 = cos(theta1);
                sinTheta1 = sin(theta1);
                glBegin(GL_QUAD_STRIP);
                phi = 0.0;
                for(j=nsides;j>=0;j--){
                        GLfloat cosPhi, sinPhi, dist;

                        phi += sideDelta;
                        cosPhi = cos(phi);
                        sinPhi = sin(phi);
                        dist = R + r * cosPhi;

                        glNormal3f(cosTheta1 * cosPhi, -sinTheta1 * cosPhi, sinPhi);
                        glVertex3f(cosTheta1 * dist, -sinTheta1 * dist, r * sinPhi);
                        glNormal3f(cosTheta * cosPhi, -sinTheta * cosPhi, sinPhi);
                        glVertex3f(cosTheta * dist, -sinTheta * dist,  r * sinPhi);
                }
                glEnd();
                theta = theta1;
                cosTheta = cosTheta1;
                sinTheta = sinTheta1;
        }
}
```

```
BOOL HRingEffect::LoadCurve(LPCTSTR lpszFileName)
{
	CString sName = lpszFileName;
	CFile file;
	if(!file.Open(sName,CFile::modeRead)){
		AfxMessageBox("Load Curve Fail");
		return FALSE;
	}else{
		CArchive ar(&file,CArchive::load);
		m_bezier.Serialize(ar);
		ar.Close();
		file.Close();
	}
	return TRUE;
}
// Bezier3d.h: interface for the CBezier3d class.
//
//////////////////////////////////////////////////////////////////////

#if !defined(AFX_BEZIER3D_H__EA6B3D28_DD5A_11D6_BDCA_8C57C83DCD6B__INCLUDED_)
#define AFX_BEZIER3D_H__EA6B3D28_DD5A_11D6_BDCA_8C57C83DCD6B__INCLUDED_

#include "Object3d.h"

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000

class CBezier3d : public CObject3d
{
	DECLARE_SERIAL(CBezier3d)
public:
	CBezier3d();
	virtual ~CBezier3d();
	const CBezier3d& operator=(const CBezier3d&);
	void Serialize(CArchive& ar);
	CVector3d m_P[4];
public:
	double GetTfromX(double x);
	BOOL BuildListGL();
	CVector3d PointOnCurve(double t);
private:
	void DrawBezier();
	void SolveCubic(double a,double b,double c,double d,int* solutions,double* x);
	double m_StepT;
	BOOL m_bShowCylinders;
	GLUquadricObj* m_quad;
};

#endif // !defined(AFX_BEZIER3D_H__EA6B3D28_DD5A_11D6_BDCA_8C57C83DCD6B__INCLUDED_)
// Bezier3d.cpp: implementation of the CBezier3d class.
//
//////////////////////////////////////////////////////////////////////

#include "stdafx.h"
#include "Bezier3d.h"
#include <math.h>

#ifdef _DEBUG
#undef THIS_FILE
```

```
static char THIS_FILE[]=__FILE__;
#define new DEBUG_NEW
#endif

//////////////////////////////////////////////////////////////////////
// Construction/Destruction
//////////////////////////////////////////////////////////////////////
#define M_PI 3.141592654
IMPLEMENT_SERIAL( CBezier3d, CObject3d, 1 )

CBezier3d::CBezier3d()
{
	m_P[0].Set(-85.0,10.0,1.0);
	m_P[1].Set(-40.0, 50.0, 1.0);
	m_P[2].Set( 60.0, -70.0, 1.0);
	m_P[3].Set(80.0,  10.0,  1.0);
	m_StepT = 0.005;
	m_bShowCylinders = TRUE;
	m_quad = gluNewQuadric();
}

CBezier3d::~CBezier3d()
{
	gluDeleteQuadric(m_quad);
}

const CBezier3d& CBezier3d::operator=(const CBezier3d& v)
{
	if(this==&v) return *this;
	for(int i=0;i<4;i++){
		m_P[i] = v.m_P[i];
	}
	m_StepT = v.m_StepT;
	m_bShowCylinders = v.m_bShowCylinders;
	m_bModified = TRUE;
	return *this;
}

void CBezier3d::Serialize( CArchive& ar )
{
   CObject3d::Serialize(ar);
   if(ar.IsStoring()){
		for(int i=0;i<4;i++){
			for(int j=0;j<3;j++){
				ar << m_P[i].m_pt[j];
			}
		}
   }else{
		for(int i=0;i<4;i++){
			for(int j=0;j<3;j++){
				ar >> m_P[i].m_pt[j];
			}
		}
		m_bModified = TRUE;
	}
}

BOOL CBezier3d::BuildListGL()
{
	if(!m_bModified && m_bListDone) return FALSE;
```

```
        // Erase last list
        glDeleteLists(m_nListOpenGL,1);
        // Search for a new list
        m_nListOpenGL = glGenLists(1);
        if(m_nListOpenGL==0) return FALSE;
        // Start list
        glNewList(m_nListOpenGL,GL_COMPILE_AND_EXECUTE);
        //glPushMatrix();
        DrawBezier();
        //glPopMatrix();
        glEndList();
        // List is done now
        m_bListDone = TRUE;
        m_bModified = FALSE;
        return TRUE;
}

void CBezier3d::DrawBezier()
{
        for(double t=0.0; t<1.0; t+=m_StepT ){
                CVector3d vPoint = PointOnCurve(t);
                if(m_bShowCylinders){
                        glPushMatrix();
                        glTranslatef(vPoint.x, vPoint.y, vPoint.z);
                        glRotatef(90,0,1,0);
                        gluCylinder(m_quad,0.5f,0.5f,1.0f,5,1);
                        glPopMatrix();
                }
        }
}

CVector3d CBezier3d::PointOnCurve(double t)
{
        double var1, var2, var3;
   CVector3d vPoint;
        vPoint.Set(0.0f, 0.0f, 0.0f);
        // Bezier curve:
        // B(t) = P1 * ( 1 - t )^3 + P2 * 3 * t * ( 1 - t )^2 + P3 * 3 * t^2 * ( 1 - t ) + P4 * t^3
   var1 = 1 - t;                                // (1 - t)
   var2 = var1 * var1 * var1;        // (1 - t)^3
   var3 = t * t * t;                            // t^3
        // Calc
   vPoint.x = var2*m_P[0].x + 3*t*var1*var1*m_P[1].x + 3*t*t*var1*m_P[2].x + var3*m_P[3].x;
   vPoint.y = var2*m_P[0].y + 3*t*var1*var1*m_P[1].y + 3*t*t*var1*m_P[2].y + var3*m_P[3].y;
   vPoint.z = var2*m_P[0].z + 3*t*var1*var1*m_P[1].z + 3*t*t*var1*m_P[2].z + var3*m_P[3].z;
   return(vPoint);
}

void CBezier3d::SolveCubic(double a,double b,double c,double d,int* solutions,double* x)
{
   long double a1 = b/a, a2 = c/a, a3 = d/a;
   long double Q = (a1*a1 - 3.0*a2)/9.0;
   long double R = (2.0*a1*a1*a1 - 9.0*a1*a2 + 27.0*a3)/54.0;
   double R2_Q3 = R*R - Q*Q*Q;
   double theta;
   if (R2_Q3 <= 0){
      *solutions = 3;
      theta = acos(R/sqrt(Q*Q*Q));
      x[0] = -2.0*sqrt(Q)*cos(theta/3.0) - a1/3.0;
      x[1] = -2.0*sqrt(Q)*cos((theta+2.0*M_PI)/3.0) - a1/3.0;
```

```
        x[2] = -2.0*sqrt(Q)*cos((theta+4.0*M_PI)/3.0) - a1/3.0;
    }else{
        *solutions = 1;
        x[0] = pow(sqrt(R2_Q3)+fabs(R), 1/3.0);
        x[0] += Q/x[0];
        x[0] *= (R < 0.0) ? 1 : -1;
        x[0] -= a1/3.0;
    }
}

double CBezier3d::GetTfromX(double x)
{
        double x0 = m_P[0].x;
        double x1 = m_P[1].x ;
        double x2 = m_P[2].x ;
        double x3 = m_P[3].x;
        double cx = 3*(x1-x0);
        double bx = 3*(x2-x1)-cx;
        double ax = x3-x0-cx-bx;
        double dx = x0-x;
        int sol;
        double time[3];
        SolveCubic(ax,bx,cx,dx,&sol,time);
        return time[2];
}
```

ประวัติผู้เขียน

ชื่อผู้เขียน	นาย อุราเคน กิจพยัคฆ์
วันเดือนปีเกิด	8 กันยายน 2520
สถานที่เกิด	กรุงเทพมหานคร
วุฒิการศึกษาระดับปริญญาตรี	วิทยาศาสตรบัณฑิต(ฟิสิกส์ประยุกต์)
สถานที่สำเร็จการศึกษา	สถาบันเทคโนโลยีพระจอมเกล้าเจ้าคุณทหารลาดกระบัง
ปีที่สำเร็จการศึกษา	2542