การวัดประสิทธิภาพของ Unreal Engine 5 สำหรับการพัฒนาเทคโนโลยีเสมือนจริง (VR)

Evaluate the Performance of Unreal Engine 5 for Virtual Reality (VR) Development

นายภควัฒน ใจลังกา รหัสประจำตัว 65021981

ปริญญานิพนธ์นี้เป็นส่วนหนึ่งของการศึกษาตามหลักสูตรวิศวกรรมศาสตร์บัณฑิต
สาขาวิชาวิศวกรรมคอมพิวเตอร์ คณะเทคโนโลยีสารสนเทศและการสื่อสาร
มหาวิทยาลัยพะเยา
ปีการศึกษา 2567

หัวข้อโครงงาน	การวัดประสิทธิภาพของ Unreal Engine 5 สำหรับการพัฒนาเทคโนโลยีเสมือนจริง (VR)
ผู้ดำเนินโครงงาน	นายภควัฒน ใจลังกา รหัส 65021981
อาจารย์ที่ปรึกษา	ผศ.ดร.บวรศักดิ์ ศรีสังสิทธิสันติ
สาขาวิชา	วิศวกรรมคอมพิวเตอร์
คณะ	เทคโนโลยีสารสนเทศและการสื่อสาร
ปีการศึกษา	2567

.....

บทคัดย่อ

การศึกษาวิจัยครั้งนี้มีวัตถุประสงค์เพื่อวัดและประเมินประสิทธิภาพของการพัฒนาเกมเสมือนจริง (Virtual Reality: VR) โดยใช้ Unreal Engine 5 ซึ่งเป็นเอนจินเกมที่ได้รับความนิยมและมีศักยภาพสูงในการสร้างสภาพแวดล้อมที่สมจริง การทดลองได้มุ่งเน้นไปที่การใช้เทคนิคสำคัญสำหรับการเพิ่มประสิทธิภาพ ได้แก่ Nanite, Level of Detail (LOD) และ Runtime Virtual Texture (RVT) เพื่อตรวจสอบผลกระทบต่อการประมวลผลด้านกราฟิก โดยใช้การทดสอบในฉาก VR หลายลักษณะ พร้อมบันทึกค่าการทำงาน เช่น CPU Frame Time, GPU Frame Time, อัตราเฟรมเรต (FPS) และการใช้งานหน่วยความจำ

ผลการทดลองแสดงให้เห็นว่า การนำเทคนิคข้างต้นมาใช้สามารถลดภาระการประมวลผลและเพิ่มความเสถียรของระบบได้อย่างมีนัยสำคัญ โดยค่า CPU และ GPU Frame Time ลดลงอย่างต่อเนื่อง ทำให้เฟรมเรตสูงขึ้นและคงที่มากกว่าเดิม ขณะเดียวกันยังช่วยลดการใช้หน่วยความจำกราฟิก (VRAM) ในบางกรณี ผลลัพธ์ดังกล่าวยืนยันว่า Unreal Engine 5 สามารถรองรับการพัฒนาเกม VR ที่มีความสมจริงและซับซ้อนได้ หากมีการเลือกใช้เทคนิคการเพิ่มประสิทธิภาพที่เหมาะสม งานวิจัยนี้จึงสามารถเป็นแนวทางให้แก่นักพัฒนาในการสร้างเกม VR ที่มีประสิทธิภาพสูงและมอบประสบการณ์การเล่นที่ราบรื่นแก่ผู้ใช้

Project Title	Evaluate the Performance of Unreal Engine 5 for Virtual Reality (VR) Development
Project Authors	Mr. Pakawat Jailanka ID. 65021981
Project Advisor	Assistant Professor Dr. Bowonsak Srisungsittisunti
Department	Computer Engineering
Faculty	School of Information and Communication Technology
Academic Year	2024

Abstract

This research aims to measure and evaluate the performance of virtual reality (VR) game development using Unreal Engine 5, a popular and highly capable game engine known for creating realistic environments. The study focused on utilizing key optimization techniques, namely Nanite, Level of Detail (LOD), and Runtime Virtual Texture (RVT) technology, to examine their impact on graphical processing performance. Various types of VR scenes and sample environments were tested, with performance metrics recorded such as CPU frame time, GPU frame time, frame rate (FPS), and memory usage.

The experimental results demonstrated that implementing these techniques significantly reduced processing workload and improved overall system stability. Both CPU and GPU frame times consistently decreased, leading to higher and more stable frame rates. Additionally, in some cases, the use of these techniques helped reduce graphics card memory (VRAM) consumption. These findings confirm that Unreal Engine 5 is capable of supporting the development of complex and realistic VR games, provided that appropriate optimization methods are applied. Therefore, this study can serve as a valuable reference for developers who aim to create high-performance VR games that deliver a smooth and immersive user experience.

กิตติกรรมประกาศ

ปริญญานิพนธ์เรื่อง “การวัดประสิทธิภาพของ Unreal Engine 5 สำหรับการพัฒนาเทคโนโลยีเสมือนจริง (VR)” สำเร็จลุล่วงไปได้ด้วยความกรุณาและการสนับสนุนจากหลายฝ่าย คณะผู้จัดทำขอแสดงความขอบพระคุณต่อทุกท่านที่มีส่วนเกี่ยวข้องดังนี้

ขอกราบขอบพระคุณ ผู้ช่วยศาสตราจารย์ ดร.บวรศักดิ์ ศรีสังสิทธิสันติ อาจารย์ที่ปรึกษาที่ได้กรุณาสละเวลาให้คำปรึกษา ชี้แนะแนวทาง และให้ข้อมูลที่เป็นประโยชน์ต่อโครงงานนี้เป็นอย่างมาก ทำให้คณะผู้จัดทำสามารถดำเนินงานวิจัยจนสำเร็จไปด้วยดี

นอกจากนี้คณะผู้จัดทำขอขอบพระคุณอาจารย์และคณาจารย์ประจำสาขาวิศวกรรมคอมพิวเตอร์คณะเทคโนโลยีสารสนเทศและการสื่อสารมหาวิทยาลัยพะเยาที่ได้ถ่ายทอดความรู้ คำแนะนำ และประสบการณ์อันมีคุณค่า ซึ่งเป็นรากฐานสำคัญในการดำเนินงานวิจัยครั้งนี้

คณะผู้จัดทำยังขอขอบพระคุณเพื่อนนักศึกษาและบุคคลรอบข้างที่คอยให้คำแนะนำและกำลังใจอย่างต่อเนื่อง ตลอดจนให้การสนับสนุนในด้านต่าง ๆ อันเป็นแรงผลักดันสำคัญที่ช่วยให้การดำเนินงานสำเร็จลุล่วง

สุดท้ายนี้ คณะผู้จัดทำหวังเป็นอย่างยิ่งว่าปริญญานิพนธ์เรื่อง “การวัดประสิทธิภาพของ Unreal Engine 5 สำหรับการพัฒนาเทคโนโลยีเสมือนจริง (VR)” จะเป็นประโยชน์ต่อผู้ที่สนใจศึกษาและพัฒนาด้าน Virtual Reality ตลอดจนผู้ที่ทำงานด้านการออกแบบเกมและระบบกราฟิกหากมีข้อผิดพลาดประการใดคณะผู้จัดทำขอน้อมรับคำติชมและข้อเสนอแนะทุกประการด้วยความยินดี

คณะผู้จัดทำ

สารบัญ

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญ(ต่อ)

หน้า

สารบัญภาพ

หน้า

สารบัญภาพ(ต่อ)

หน้า

บทที่ 1

บทนำ

ในยุคปัจจุบัน เทคโนโลยี Virtual Reality (VR) ได้ก้าวขึ้นมาเป็นเครื่องมือที่มีศักยภาพสูงในหลากหลายด้าน ทั้งในวงการบันเทิง การศึกษา การแพทย์ และการออกแบบ เนื่องจากสามารถสร้างประสบการณ์เสมือนจริงที่ทำให้ผู้ใช้งานรู้สึกมีส่วนร่วมและเข้าถึงได้อย่างเต็มที่ การพัฒนาเทคโนโลยีนี้ทำให้เกม VR ได้รับความสนใจเป็นอย่างมาก ทั้งจากนักพัฒนาและผู้บริโภค ด้วยลักษณะเฉพาะของการปฏิสัมพันธ์แบบสามมิติที่สร้างความตื่นเต้นและประสบการณ์ที่แตกต่างจากเกมในรูปแบบดั้งเดิม

อย่างไรก็ตาม การพัฒนาเกม VR ยังคงเผชิญกับความท้าทายหลายประการ โดยเฉพาะในด้านประสิทธิภาพของเกม ซึ่งมีผลโดยตรงต่อประสบการณ์ของผู้ใช้งาน หากเกมมีการทำงานที่ไม่เสถียร หรือเกิดปัญหาในการแสดงผล อาจก่อให้เกิดอาการเวียนศีรษะหรือคลื่นไส้ในผู้เล่น ซึ่งจะกระทบต่อความสนุกสนานและความต่อเนื่องในการเล่นเกม VR

ดังนั้น การวิจัยเพื่อวัดประสิทธิภาพของเกม VR และการพัฒนาที่มุ่งเน้นให้เกมทำงานได้อย่างมีประสิทธิภาพสูงสุดจึงมีความสำคัญอย่างยิ่ง การวัดประสิทธิภาพจะช่วยให้นักพัฒนาสามารถระบุและแก้ไขปัญหาที่อาจเกิดขึ้นได้อย่างรวดเร็วและมีประสิทธิผล นอกจากนี้ การพัฒนาเกมที่มีคุณภาพยังช่วยส่งเสริมประสบการณ์การเล่นเกมที่ราบรื่นและสมจริงยิ่งขึ้น

การศึกษานี้มีความสำคัญในการพัฒนาแนวทางและเทคนิคในการวัดประสิทธิภาพของเกม VR เพื่อให้สามารถประเมินและปรับปรุงคุณภาพของเกมได้อย่างมีประสิทธิภาพ ทั้งยังเป็นการส่งเสริมการพัฒนาเกม VR ที่มีคุณภาพสูง เพื่อรองรับความต้องการของตลาดและยกระดับประสบการณ์การเล่นเกมของผู้ใช้งานให้ดียิ่งขึ้น

1.1 วัตถุประสงค์

– เพื่อศึกษาและวิเคราะห์การใช้ Unreal Engine ในการพัฒนาเกมVRโดยมุ่งเน้นการเพิ่มประสิทธิภาพการทำงาน (Performance Optimization) ของเกม

– เพื่อวัดผลและเปรียบเทียบประสิทธิภาพของเกมVRก่อนและหลังการปรับปรุงโดยใช้ Unreal Engine เป็นฐานในการพัฒนา

– เพื่อรวบรวมข้อมูลเชิงลึกเกี่ยวกับผลกระทบของการปรับแต่งประสิทธิภาพในเกม VR ที่พัฒนาด้วย Unreal Engine เพื่อเป็นแนวทางในการพัฒนาเกมที่มีความเสถียรและมีประสิทธิภาพสูงขึ้น

– เพื่อสร้างแนวทางและคำแนะนำสำหรับนักพัฒนาเกมในการเพิ่มประสิทธิภาพของเกมVR ด้วย Unreal Engine ให้เกิดประสิทธิผลสูงสุดและมอบประสบการณ์ที่ดีที่สุดแก่ผู้เล่น

1.2 ขอบเขตการดำเนินงาน

– การพัฒนาเกม VR ด้วย Unreal Engine โครงงานนี้จะใช้ Unreal Engine เป็นเครื่องมือหลักในการพัฒนาเกม VR และจะเลือกใช้อุปกรณ์ VR ที่เข้ากันได้ เช่น Oculus Rift, HTC Vive หรือ Meta Quest สำหรับทดสอบประสิทธิภาพ

– การวัดประสิทธิภาพก่อนและหลังการปรับปรุง (Optimization)การวัดประสิทธิภาพของเกมจะทำโดยใช้ Unreal Engine ในการตรวจสอบค่าอัตราเฟรมเรต (FPS), การใช้หน่วยความจำ และอัตราการประมวลผลอื่น ๆ ก่อนและหลังการปรับปรุง เพื่อให้ได้ข้อมูลเชิงเปรียบเทียบ

– การใช้ฟีเจอร์ของ Unreal Engine สำหรับการเพิ่มประสิทธิภาพ โดยเน้นการใช้ Level of Detail (LOD), Nanite และ Runtime Virtual Texture (RVT) เพื่อทดสอบผลลัพธ์ของการปรับแต่งที่ส่งผลต่อประสิทธิภาพของเกม VR โดย LOD ช่วยลดจำนวนโพลิกอนในระยะไกล , Nanite ช่วยจัดการเรนเดอร์โมเดลที่มีความละเอียดสูงโดยไม่กระทบประสิทธิภาพ และ RVT ช่วยจัดการการแสดงผลพื้นผิวให้เหมาะสมกับระยะมองเห็นเพื่อลดภาระของ GPU

– การทดลองในสถานการณ์ที่หลากหลายโครงงานนี้จะทำการทดสอบในสภาวะและฉากต่าง ๆ ภายในเกม VR เช่น ฉากที่มีองค์ประกอบซับซ้อนและฉากที่เรียบง่าย เพื่อวิเคราะห์การเปลี่ยนแปลงของประสิทธิภาพในสภาวะที่แตกต่างกัน

– การวิเคราะห์ข้อมูลและสรุปผลลัพธ์ทำการรวบรวมและวิเคราะห์ข้อมูลที่ได้จากการทดสอบประสิทธิภาพในรูปแบบของตารางและกราฟ พร้อมกับสรุปผลลัพธ์เพื่อสร้างแนวทางที่เหมาะสมในการเพิ่มประสิทธิภาพของเกม VR ด้วย Unreal Engine

– ขอบเขตการใช้เทคโนโลยี VR การทดสอบจะจำกัดขอบเขตอยู่ที่การสร้างและปรับปรุงประสิทธิภาพเกม VR ใน Unreal Engine เท่านั้น โดยจะไม่รวมถึงการสร้างเกมจากเอนจินอื่นหรือการเปรียบเทียบกับเอนจินอื่น

1.3 ขอบเขตการศึกษา

1.3.1 ขอบเขตเนื้อหา (Content Scope)

- การใช้ Unreal Engine ในการพัฒนาเกม VR
- เทคนิคการเพิ่มประสิทธิภาพ (Performance Optimization) เช่น LOD, RVT, Nanite และการใช้เครื่องมือวิเคราะห์ใน Unreal Engine เช่น Profiler และ GPU Visualizer
- การวิเคราะห์ประสิทธิภาพในด้านอัตราเฟรมเรต (Frame Rate), การใช้ทรัพยากรระบบ (Resource Utilization) เช่น GPU และ CPU

1.3.2 ขอบเขตพื้นที่การศึกษา (Geographical Scope)

- ใช้ข้อมูลและการทดสอบในสภาพแวดล้อมของ Unreal Engine และฮาร์ดแวร์ VR ที่รองรับ

1.3.3 ขอบเขตกลุ่มเป้าหมาย (Target Group)

- นักพัฒนาเกมที่ต้องการเพิ่มประสิทธิภาพเกม VR
- ผู้ที่เกี่ยวข้องกับการพัฒนาและบริหารโครงการเกม VR

1.3.4 ขอบเขตด้านผลลัพธ์ (Result Scope)

- การวัดและเปรียบเทียบประสิทธิภาพของเกม VR ก่อนและหลังการปรับปรุง
- การนำเสนอผลการศึกษาในรูปแบบกราฟ รายงาน หรือข้อมูลเชิงวิเคราะห์ที่เข้าใจง่าย

1.4 ขั้นตอนการดำเนินงาน

ขั้นตอนการดำเนินงาน

1.4.1 การวางแผนโครงงานและกำหนดขอบเขต

– ศึกษาและกำหนดวัตถุประสงค์ ขอบเขต และขอบเขตของงาน

– วิเคราะห์ Unreal Engine และเลือกฟีเจอร์สำหรับการเพิ่มประสิทธิภาพที่ต้องการทดสอบในเกม VR

1.4.2 การพัฒนาเกมต้นแบบใน Unreal Engine

– สร้างเกม VR ต้นแบบโดยใช้ Unreal Engine ที่มีองค์ประกอบพื้นฐานสำหรับการทดสอบ เช่น ฉากที่มีรายละเอียดกราฟิกที่หลากหลาย และองค์ประกอบการเคลื่อนไหวที่ท้าทายต่อประสิทธิภาพของระบบ

– ตรวจสอบการทำงานและการแสดงผลในอุปกรณ์ VR เพื่อให้แน่ใจว่าเกมต้นแบบสามารถใช้งานได้ในสถานะเริ่มต้น

1.4.3 การวัดประสิทธิภาพของเกม VR ก่อนการปรับปรุง

– ใช้เครื่องมือใน Unreal Engine เพื่อตรวจวัดค่าอัตราเฟรมเรต (FPS), การใช้หน่วยความจำ และอัตราการประมวลผลในเกมต้นแบบ

– บันทึกข้อมูลพื้นฐานที่ได้เพื่อใช้เป็นฐานในการเปรียบเทียบหลังจากการปรับปรุง

1.4.4 การปรับปรุงประสิทธิภาพของเกม VR (Optimization)

– ใช้ฟีเจอร์ Level of Detail (LOD), Nanite และ Runtime Virtual Texture (RVT) ใน Unreal Engine เพื่อเพิ่มประสิทธิภาพการประมวลผลและการแสดงผลในเกม VR

– ปรับแต่งการตั้งค่า LOD เพื่อลดรายละเอียดของโมเดลในระยะไกล, Nanite เพื่อจัดการเรนเดอร์โมเดลที่ซับซ้อนอย่างมีประสิทธิภาพ และ RVT เพื่อปรับปรุงการแสดงผลพื้นผิว ลดภาระของระบบโดยไม่ลดคุณภาพของประสบการณ์ VR

1.4.5 การวัดประสิทธิภาพหลังการปรับปรุง

– ทำการทดสอบและวัดผลการทำงานของเกมอีกครั้งหลังจากปรับปรุง โดยตรวจสอบค่า FPS, การใช้หน่วยความจำ และอัตราการประมวลผลต่าง ๆ

– บันทึกข้อมูลที่ได้หลังการปรับปรุงเพื่อเปรียบเทียบกับข้อมูลก่อนการปรับปรุง

1.4.6 การวิเคราะห์ผลลัพธ์และสรุปข้อมูล

– วิเคราะห์ข้อมูลที่ได้จากการวัดประสิทธิภาพก่อนและหลังการปรับปรุง โดยนำเสนอในรูปแบบตารางและกราฟเพื่อแสดงการเปลี่ยนแปลง

– สรุปผลลัพธ์การเพิ่มประสิทธิภาพของเกม VR ที่ได้จากการใช้ฟีเจอร์ต่าง ๆ ใน Unreal Engine พร้อมอธิบายผลกระทบของการปรับปรุงที่มีต่อประสบการณ์การใช้งาน VR

1.4.7 การจัดทำรายงานผลการดำเนินงานและแนวทางการพัฒนาต่อไป

– จัดทำรายงานสรุปขั้นตอนการดำเนินงาน ข้อมูลที่ได้จากการวัดผล และข้อสรุปจากการวิเคราะห์

– นำเสนอข้อเสนอแนะและแนวทางในการพัฒนาเกม VR ด้วย Unreal Engine เพื่อเพิ่มประสิทธิภาพในการทำงานและมอบประสบการณ์ที่ดีแก่ผู้เล่น

การดำเนินงาน	ปีการศึกษา 2567			ปีการศึกษา 2568									
	ต.ค.	พ.ย.	ธ.ค.	ม.ค.	ก.พ.	มี.ค.	เม.ย	พ.ค.	มิ.ย.	ก.ค.	ส.ค.	ก.ย.	ต.ค.
1. วิเคราะห์ระบบและเก็บข้อมูลความต้องการของระบบ	←	→											
2. ศึกษาทฤษฎีที่เกี่ยวข้องกับการพัฒนาระบบ	←		→										
3. ออกแบบและพัฒนาระบบ		←					→						
4. ดำเนินการเขียนโปรแกรม		←						→					
5. นำเสนอหัวข้อ					↔								↔
6. ทดสอบและปรับปรุงแก้ไขข้อผิดพลาด					←					→			
7. สรุปผลการดำเนินงาน										↔		↔	
8. จัดทำเอกสาร		←		→						↔		↔	
9. นำเสนอโครงงาน					↔								↔

รูปที่ 1.1 แผนการดำเนินงานโครงงานวิศวกรรม

1.5 ประโยชน์ที่ได้รับ

– ได้แนวทางในการปรับปรุงประสิทธิภาพเกม VR ด้วย Unreal Engine

– ลดปัญหาภาพกระตุกและเพิ่มความเสถียรของเกม VR

– มีข้อมูลเชิงลึกสำหรับพัฒนาประสิทธิภาพเกม VR ในอนาคต

– ลดการใช้ทรัพยากรระบบ ทำให้เกมทำงานได้ดีขึ้นบนอุปกรณ์ VR ที่หลากหลาย

– เพิ่มความพึงพอใจของผู้เล่นและความน่าเชื่อถือในตลาด VR

1.6 ฮาร์ดแวร์และซอฟแวร์ที่ใช้งาน

1.6.1 ฮาร์ดแวร์

คอมพิวเตอร์ส่วนบุคคล:

CPU : Ryzen 5900x 12 core 24 thread

Ram : 64 gb

GPU: RTX 3080 12gb TUF OC

เครื่องคอมพิวเตอร์แบบพกพา:

CPU: Intel i5 1035G1

Ram: 20 gb

GPU: UHD Graphics

อุปกรณ์แสดงผลเสมือนจริง (VR Device):

Meta Quest 3

Meta Quest Controller

1.6.2 ซอฟแวร์

- Unreal engine 5 version 5.4.x

1.7 โครงสร้างของปริญญานิพนธ์

บทที่ 1: บทนำ

บทนำกล่าวถึงความสำคัญของการใช้ Unreal Engine ในการพัฒนาเกม Virtual Reality (VR) ซึ่งเป็นเทคโนโลยีที่กำลังเติบโตอย่างรวดเร็วในวงการเกมและความบันเทิง การพัฒนาเกม VR มีความท้าทายในด้านการจัดการประสิทธิภาพ (Performance) เพื่อให้เกมสามารถมอบประสบการณ์ที่เสถียรและราบรื่นแก่ผู้เล่น เนื้อหาในบทนี้จะระบุปัญหาหลักที่เกี่ยวข้องกับการลดลงของเฟรมเรตหรือปัญหาภาพกระตุกที่ส่งผลต่อความพึงพอใจของผู้เล่น พร้อมระบุวัตถุประสงค์ของการศึกษาที่มุ่งเน้นการเพิ่มประสิทธิภาพการแสดงผลของเกม VR โดยใช้ Unreal Engine เป็นฐานการพัฒนา และประโยชน์ที่คาดว่าจะได้รับ เช่น แนวทางการปรับปรุงประสิทธิภาพเกมที่เหมาะสม

บทที่ 2: เอกสารและงานวิจัยที่เกี่ยวข้อง

บทนี้ทบทวนแนวคิด ทฤษฎี และงานวิจัยที่เกี่ยวข้องกับ Unreal Engine, Virtual Reality และเทคนิคการเพิ่มประสิทธิภาพ เช่น LOD (Level of Detail), RVT (Runtime Virtual Texture), และ Nanite รวมถึงการวิเคราะห์เครื่องมือที่มีใน Unreal Engine เช่น Profiler และ GPU Visualizer เพื่อการตรวจสอบประสิทธิภาพของระบบ นอกจากนี้ยังรวมถึงการศึกษาเกี่ยวกับผลกระทบของเฟรมเรตและประสิทธิภาพเกม VR ต่อประสบการณ์ผู้ใช้ โดยสรุปสาระสำคัญที่เกี่ยวข้องเพื่อใช้เป็นพื้นฐานในการวิเคราะห์และดำเนินการศึกษา

บทที่ 3: วิธีดำเนินการวิจัย

บทนี้ระบุขั้นตอนการดำเนินงาน ตั้งแต่การเลือกฉากและโมเดลที่ใช้ในการทดสอบใน Unreal Engine การตั้งค่าระบบเพื่อเก็บข้อมูลประสิทธิภาพ เช่น ค่าเฟรมเรต (Frame Rate), การใช้ทรัพยากรระบบ (GPU/CPU Utilization) การปรับแต่งค่าใน LOD, RVT, และ Nanite เพื่อทดสอบผลกระทบต่อประสิทธิภาพ การเก็บรวบรวมข้อมูลโดยใช้ Profiler GPU Visualizer และวิเคราะห์ข้อมูลเพื่อเปรียบเทียบก่อนและหลังการปรับปรุง โดยใช้สถิติเชิงพรรณนาและการนำเสนอข้อมูลผ่านกราฟหรือรายงาน

บทที่ 4: ผลการศึกษา

นำเสนอผลลัพธ์ที่ได้จากการวิเคราะห์ข้อมูล เช่น การเปรียบเทียบค่าเฟรมเรตก่อนและหลังการปรับปรุงประสิทธิภาพ การแสดงข้อมูลการใช้ทรัพยากรระบบในแต่ละขั้นตอน พร้อมการอภิปรายผลเพื่อชี้ให้เห็นถึงผลกระทบของเทคนิคต่าง ๆ เช่น การปรับ LOD ที่ช่วยลดการประมวลผลโมเดลที่อยู่ไกล, RVT ที่ช่วยลดการโหลดข้อมูลของพื้นผิว, และ Nanite ที่ช่วยเพิ่มประสิทธิภาพการเรนเดอร์สำหรับโมเดลที่มีความซับซ้อน

บทที่ 5: สรุป อภิปรายผล และข้อเสนอแนะ

บทสุดท้ายสรุปผลการศึกษาเกี่ยวกับการเพิ่มประสิทธิภาพเกม VR ด้วย Unreal Engine โดยยืนยันผลลัพธ์ที่ได้ว่าสามารถช่วยเพิ่มความเสถียรและลดการใช้ทรัพยากรระบบได้อย่างมีประสิทธิภาพ อภิปรายผลลัพธ์ที่ได้เปรียบเทียบกับงานวิจัยที่เกี่ยวข้อง พร้อมทั้งเสนอข้อเสนอแนะสำหรับนักพัฒนาเกม เช่น การเลือกใช้เทคนิคที่เหมาะสมกับลักษณะของเกม หรือการปรับแต่งเพิ่มเติมในระบบ VR ในอนาคต เสนอแนวทางสำหรับการวิจัยเพิ่มเติม เช่น การศึกษาเทคโนโลยีใหม่ใน Unreal Engine หรือการพัฒนาเกม VR ที่รองรับอุปกรณ์หลากหลายมากขึ้น

บทที่ 2

ทฤษฎีที่เกี่ยวข้อง

ในบทนี้จะกล่าวถึงทฤษฎีและหลักการที่เกี่ยวข้องกับการวิจัยเพื่อให้เข้าใจถึงเทคโนโลยีและแนวทางที่ใช้ในการวัดและปรับปรุงประสิทธิภาพของ Unreal Engine 5 สำหรับการพัฒนาเทคโนโลยีเสมือนจริง (VR) โดยมีการรวบรวมข้อมูลที่ครอบคลุมในด้านต่าง ๆ เช่น เทคโนโลยีใหม่ใน Unreal Engine 5 หลักการทำงานและความสำคัญของเทคโนโลยี VR รวมถึงกระบวนการและเครื่องมือที่ใช้ในการวัดผลด้านประสิทธิภาพ นอกจากนี้ ยังมีการกล่าวถึงหลักการเพิ่มประสิทธิภาพที่สามารถนำไปใช้เพื่อปรับปรุงประสบการณ์ของผู้ใช้งานในระบบ VR

ทฤษฎีที่เกี่ยวข้อง

2.1 Nanite

Nanite เป็นเทคโนโลยีการเรนเดอร์ไมโครโพลิกอน (Micropolygon Rendering) ที่พัฒนาโดย Unreal Engine 5 เพื่อรองรับการแสดงผลของโมเดล 3D ที่มีความละเอียดสูงอย่างมีประสิทธิภาพ โดยสามารถลดภาระการสร้างระดับรายละเอียด (Level of Detail: LOD) แบบดั้งเดิม และช่วยให้นักพัฒนาสามารถใช้โมเดลที่มีจำนวนโพลิกอนสูงได้โดยไม่กระทบต่อประสิทธิภาพการทำงานของระบบ

ข้อดีของ Nanite

- รองรับการแสดงผลโมเดลความละเอียดสูง

Nanite ช่วยให้สามารถแสดงผลโมเดลที่มีจำนวนโพลิกอนสูงได้ในระดับเรียลไทม์ โดยไม่ต้องลดคุณภาพของโมเดลหรือแยกสร้าง LOD

- ลดขั้นตอนการพัฒนา

นักพัฒนาไม่จำเป็นต้องปรับลดรายละเอียดของโมเดลด้วยตนเอง เนื่องจาก Nanite สามารถจัดการรายละเอียดได้อัตโนมัติตามระยะห่างจากกล้อง

- เพิ่มประสิทธิภาพการใช้ทรัพยากร

Nanite ใช้เทคนิคการสตรีมข้อมูลโพลิกอนที่จำเป็น ช่วยลดภาระของหน่วยประมวลผลและการใช้หน่วยความจำ

- เหมาะสำหรับฉากที่มีรายละเอียดสูง

สามารถนำไปใช้กับฉากขนาดใหญ่หรือฉากที่ต้องการรายละเอียด เช่น พื้นผิวภูมิประเทศ หิน หรือโครงสร้างสถาปัตยกรรม

- การผสานเทคโนโลยีอื่นใน Unreal EngineNanite สามารถทำงานร่วมกับระบบอื่น เช่น Lumen เพื่อเพิ่มความสมจริงของการเรนเดอร์และระบบแสง

ข้อเสียของ Nanite

- ข้อจำกัดการรองรับ
  - ไม่รองรับโมเดลที่มีการเคลื่อนไหว เช่น Skinned Mesh (ตัวละคร)
  - ไม่รองรับวัสดุโปร่งแสง (Transparent Materials)
- ข้อกำหนดของฮาร์ดแวร์

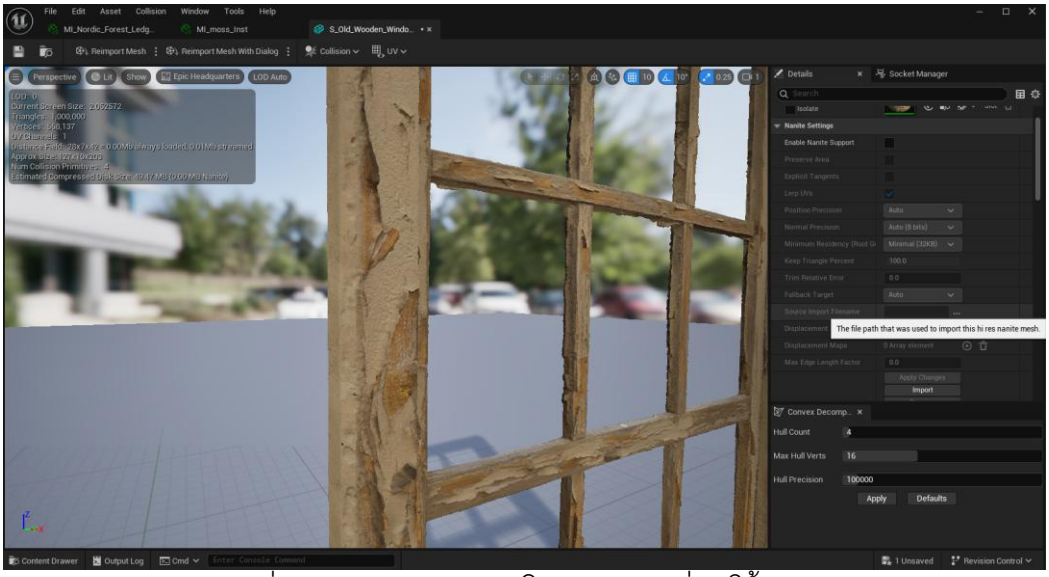
Nanite ต้องการฮาร์ดแวร์ที่ทันสมัย โดยเฉพาะการ์ดจอรุ่นใหม่ที่รองรับการประมวลผลขั้นสูง

- ขนาดไฟล์โมเดลที่ใหญ่ขึ้น

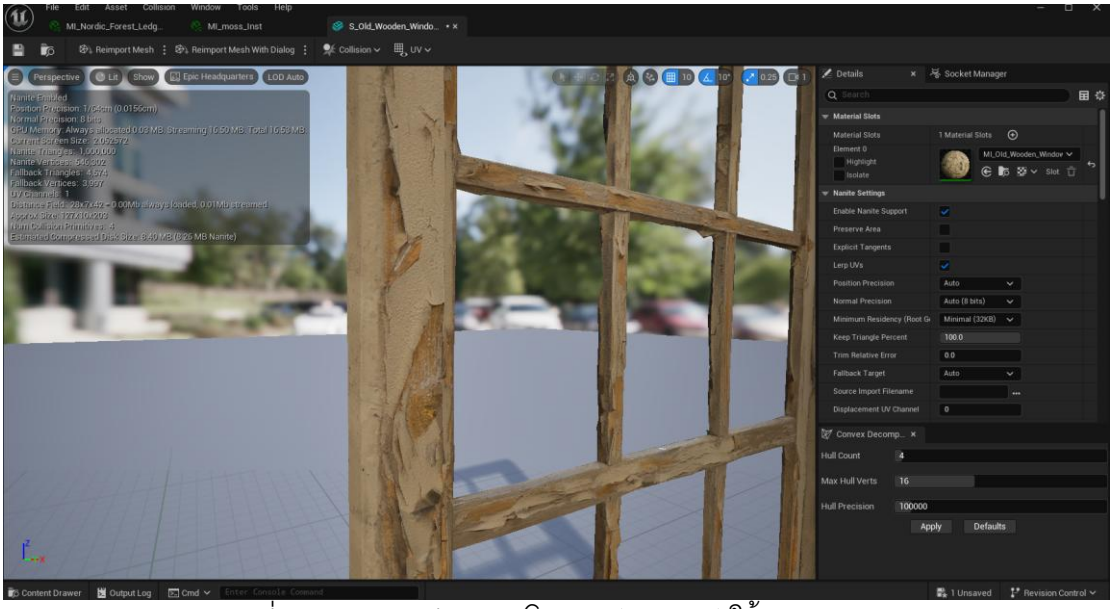
การใช้โมเดลที่รองรับ Nanite มักส่งผลให้ขนาดไฟล์เพิ่มขึ้น ซึ่งอาจกระทบต่อพื้นที่จัดเก็บและระยะเวลาในการโหลด

- กระบวนการตั้งค่าที่อาจต้องการความชำนาญ

แม้ Nanite จะทำงานอัตโนมัติในหลายส่วน แต่การปรับแต่งเพิ่มเติมในบางกรณีอาจต้องใช้ความรู้เฉพาะทาง



รูปที่ 2.1 รายละเอียดของโมเดลดังเดิมก่อนใช้งาน Nanite



รูปที่ 2.2 รายละเอียดของโมเดลดังเดิมหลังใช้งาน Nanite

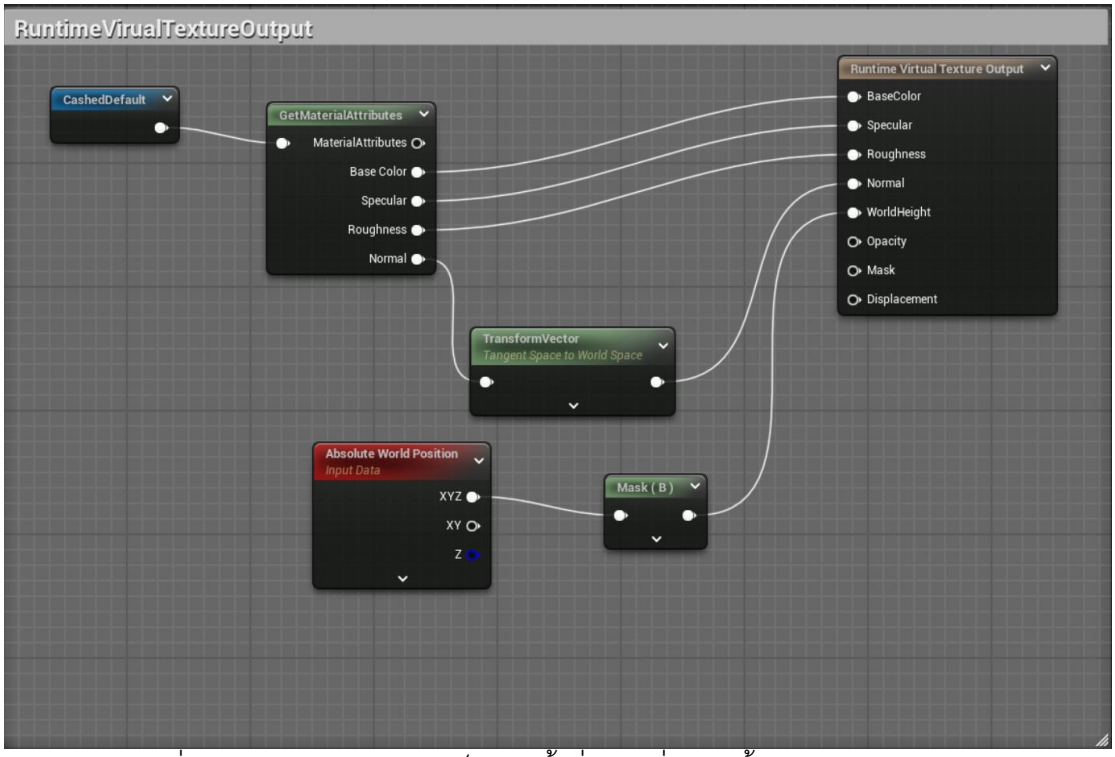
2.2 Runtime Virtual Texture (RVT)

เป็นเทคโนโลยีสำหรับการจัดการและเรนเดอร์ Virtual Texture ใน Unreal Engine ที่ถูกออกแบบมาเพื่อเพิ่มประสิทธิภาพในการแสดงผลกราฟิก โดยเฉพาะในฉากขนาดใหญ่หรือพื้นที่ที่มีรายละเอียดซับซ้อน เช่น ภูมิทัศน์ (Landscape) หรือสภาพแวดล้อมแบบ Open-world

2.2.1 หลักการทำงานของ Runtime Virtual Texture (RVT)

Runtime Virtual Texture (RVT) เป็นระบบที่ใช้สำหรับการจัดเก็บและเรียกใช้งานข้อมูลวัสดุ (Material Attributes) ในรูปแบบของ Virtual Texture ซึ่งช่วยลดภาระในการประมวลผลวัสดุแบบเรียลไทม์ และเพิ่มประสิทธิภาพการแสดงผลในฉากที่มีรายละเอียดสูง โดยระบบจะรวบรวมข้อมูล เช่น Base Color, Roughness, Normal Map และ Height Map แล้วจัดเก็บในรูปแบบ Texture Cache ที่พร้อมสำหรับการเรนเดอร์

ตัวอย่างการตั้งค่าการส่งออกข้อมูลวัสดุ (Material) ไปยัง Runtime Virtual Texture Output



รูปที่ 2.3 ระบบการทำงานเป็นการตั้งค่าการส่งออกข้อมูลวัสดุ (Material)

- Cached Default

เป็นตัวกำหนดค่าพื้นฐานของวัสดุที่ใช้ข้อมูลจาก Virtual Texture เพื่อช่วยลดการประมวลผลในส่วนที่ซ้ำซ้อน

- GetMaterialAttributes

ใช้เพื่อดึงคุณสมบัติของวัสดุ (Material Attributes) เช่น Base Color, Specular, Roughness, และ Normal เพื่อส่งต่อไปยังระบบการเรนเดอร์ Virtual Texture

- Transform Vector

ทำหน้าที่แปลงเวกเตอร์จาก Tangent Space (พื้นที่สัมพัทธ์) เป็น World Space (พื้นที่โลก) เพื่อให้ข้อมูลเวกเตอร์ถูกต้องในระบบการแสดงผล

- Absolute World Position

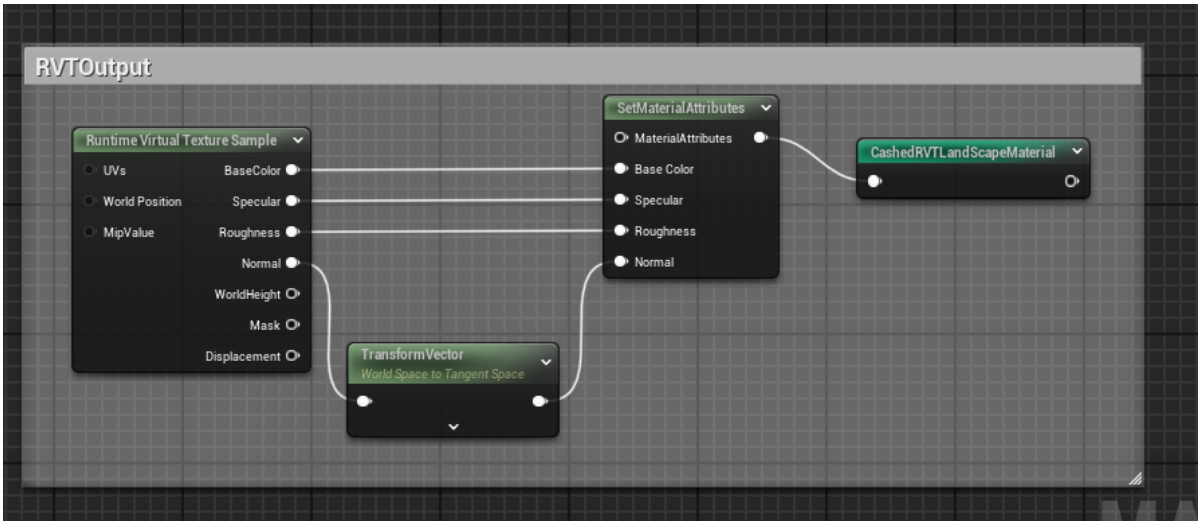
ใช้ในการดึงตำแหน่งของวัตถุใน World Space เพื่อคำนวณการเรนเดอร์หรือการแมปเท็กซ์เจอร์ให้สัมพันธ์กับตำแหน่งที่แท้จริง

- Mask (B)

ใช้เพื่อกรองหรือตัดข้อมูลบางส่วนของค่า Vector โดยทั่วไปจะใช้ในกรณีที่ต้องการแยกข้อมูลเฉพาะ Channel เช่น สีหรือความสูง

- Runtime Virtual Texture Output

เป็นปลายทางของข้อมูลทั้งหมดในกราฟนี้ โดยทำหน้าที่รวบรวมคุณสมบัติของวัสดุ (Base Color, Specular, Roughness, Normal ฯลฯ) และส่งข้อมูลเข้าสู่ Runtime Virtual Texture เพื่อการเรนเดอร์



รูปที่ 2.4 ตั้งค่ากราฟวัสดุ (Material Graph) เพื่อเชื่อมโยงข้อมูลจาก Runtime Virtual Texture และส่งข้อมูลเหล่านั้นไปยังระบบวัสดุแบบ Landscape

รูปที่ 2.5 การผสมเลเยอร์และใช้งาน Runtime Virtual Texture (RVT)รูปที่ 2.4 ตั้งค่ากราฟวัสดุ (Material Graph) เพื่อเชื่อมโยงข้อมูลจาก Runtime Virtual Texture และส่งข้อมูลเหล่านั้นไปยังระบบวัสดุแบบ Landscape

Roughness: ความหยาบของพนผว

Normal: ข้อมูลพื้นผิวปกติ

World Position: พิกัดของตำแหน่งในพื้นที่โลก

- SetMaterialAttributes

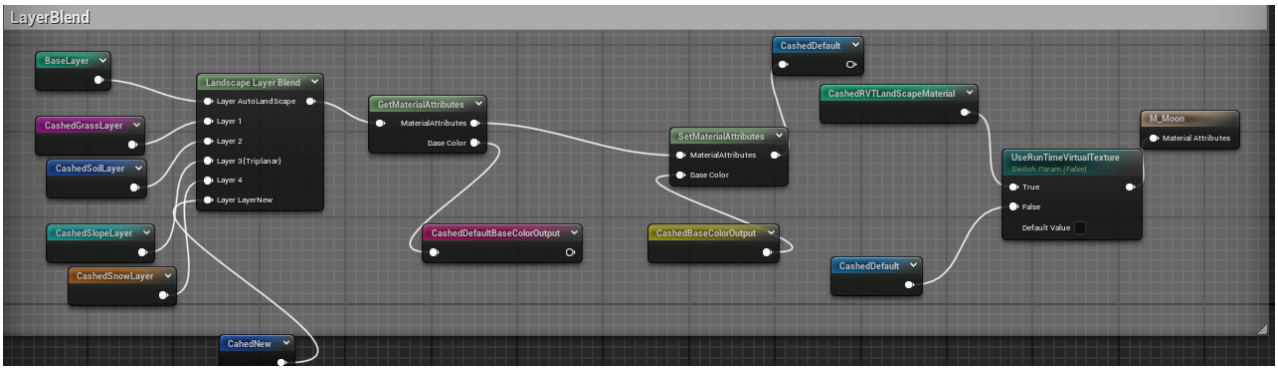
รวบรวมข้อมูลที่ได้รับจากโนด Runtime Virtual Texture Sample เพื่อจัดการคุณสมบัติของวัสดุให้เหมาะสมสำหรับการแสดงผล โดยรวบรวมข้อมูล เช่น Base Color, Specular, Roughness, และ Normal

- Transform Vector (World Space to Tangent Space)

ทำหน้าที่แปลงข้อมูลเวกเตอร์จากพื้นที่โลก (World Space) ให้เป็นพื้นที่สัมพัทธ์ (Tangent Space) และช่วยให้ข้อมูลของพื้นผิววัสดุสามารถปรับตามมุมมองและตำแหน่งของวัตถุในฉากได้อย่างถูกต้อง

- Cached RVT Landscape Material (Output)

เป็นปลายจากการจัดเก็บจาก SetMaterialAttributes และใช้ในกรณีที่วัสดุเป็นส่วนหนึ่งของภูมิทัศน์ (Landscape) หรือฉากขนาดใหญ่ที่ต้องการประสิทธิภาพสูงและการประมวลผลแบบเรียลไทม์



รูปที่ 2.5 การผสมเลเยอร์และใช้งาน Runtime Virtual Texture (RVT)

รูปที่ 2.6 หน้าต่างการตั้งค่า LOD ใน Unreal engine 5รูปที่ 2.5 การผสมเลเยอร์และใช้งาน Runtime Virtual Texture (RVT)

- Landscape Layer Blend

  ทำหน้าที่ผสมเลเยอร์วัสดุหลายเลเยอร์เข้าด้วยกันในภาพนี้ มีเลเยอร์ย่อย เช่น:

  CachedGrassLayer: วัสดุที่เป็นหญ้า

  CachedSoilLayer: วัสดุที่เป็นดิน

  CachedSlopeLayer: วัสดุที่เหมาะกับพื้นที่ลาดชัน

  CachedSnowLayer: วัสดุที่เป็นหิมะ

เลเยอร์เหล่านี้จะถูกผสมตามพารามิเตอร์ที่ตั้งไว้ เช่น ความสูง, ความลาดชัน หรือข้อมูล Mask

- GetMaterialAttributes

ดึงคุณสมบัติของวัสดุ เช่น Base Color สำหรับการปรับเปลี่ยนหรือส่งต่อในกราฟวัสดุ

- SetMaterialAttributes

  ใช้กำหนดคุณสมบัติของวัสดุหลังจากมีการปรับแต่ง เช่น การตั้งค่าข้อมูล Base Color หรืออื่น ๆ เพื่อส่งต่อไปยังระบบวัสดุปลายทาง

- CachedDefaultBaseColorOutput

  เก็บข้อมูลสีพื้นฐาน (Base Color) ที่ผ่านการปรับแต่งแล้วจากโนดก่อนหน้า

- CachedBaseColorOutput

  เป็นโนดที่ใช้เก็บข้อมูลสีพื้นฐานเพิ่มเติมสำหรับการผสมหรือเปรียบเทียบข้อมูลในภายหลัง

- CachedDefault

  โนดที่ใช้เก็บค่าที่ตั้งไว้เริ่มต้น (Default Value) สำหรับการใช้งานในกราฟวัสดุ

- CachedRVTLandscapeMaterial

  เป็นวัสดุที่เก็บข้อมูลจาก Runtime Virtual Texture เพื่อการแสดงผลภูมิทัศน์ (Landscape) ที่มีประสิทธิภาพสูง

- UseRuntimeVirtualTexture (Switch Param)

  ทำหน้าที่กำหนดเงื่อนไข (Switch) ว่าจะใช้ Runtime Virtual Texture หรือไม่ โดยสามารถกำหนดค่าได้ว่าจะแสดงผลแบบใด:

  True: ใช้ข้อมูล RVT

  False: ใช้ค่าที่ตั้งไว้ใน Default Value

- M_Moon (Material Attributes)

  เป็นวัสดุปลายทางที่เก็บข้อมูลวัสดุที่ผ่านการประมวลผลทั้งหมดในกราฟ

ข้อดีของ RVT (Runtime Virtual Texture)

- ลดการใช้หน่วยความจำ (Memory Efficiency)

  RVT โหลดเฉพาะข้อมูลเท็กซ์เจอร์ที่จำเป็นต่อการแสดงผลตามตำแหน่งของกล้อง ทำให้ประหยัดหน่วยความจำและลดการประมวลผล

- ปรับปรุงประสิทธิภาพในฉากขนาดใหญ่

  ระบบ RVT เหมาะสำหรับการจัดการข้อมูลในฉากที่มีพื้นที่ขนาดใหญ่หรือรายละเอียดมาก เช่น Landscape

- การรวมข้อมูลวัสดุที่ซับซ้อน

  รองรับการบันทึกข้อมูลหลายเลเยอร์ของวัสดุใน Texture เดียว เช่น Base Color, Normal, และ Roughness

- ลดเวลาในการพัฒนา

  RVT ช่วยลดความซับซ้อนในการตั้งค่า LOD และการจัดการเท็กซ์เจอร์ ทำให้นักพัฒนามีเวลาเน้นการสร้างสรรค์เนื้อหามากขึ้น

ข้อเสียหรือข้อจำกัดของ RVT (Runtime Virtual Texture)

- ข้อจำกัดด้านฮาร์ดแวร์

  การใช้งาน RVT ต้องการอุปกรณ์ฮาร์ดแวร์ที่มีประสิทธิภาพสูง เช่น GPU รุ่นใหม่ที่รองรับการทำงานของระบบ

- การตั้งค่าที่ซับซ้อน

การปรับตั้งค่า RVT ในโปรเจกต์อาจต้องใช้ความรู้เชิงลึกเกี่ยวกับระบบการเรนเดอร์

- การใช้งานที่จำกัด

RVT ไม่เหมาะสำหรับวัตถุที่เคลื่อนไหวหรือวัสดุที่มีลักษณะโปร่งใส (Translucent Materials)

2.3 Level of Detail (LOD)

LOD (Level of Detail) เป็นเทคนิคที่ใช้ในกราฟิกคอมพิวเตอร์และเกมเพื่อปรับรายละเอียดของโมเดลหรือ Texture ตามระยะทางระหว่างวัตถุกับกล้อง เพื่อเพิ่มประสิทธิภาพในการเรนเดอร์ โดยโมเดลหรือ Texture ที่อยู่ไกลจะมีรายละเอียดลดลง (Low Poly/Low Resolution) และโมเดลที่อยู่ใกล้จะมีรายละเอียดสูงขึ้น (High Poly/High Resolution)

2.3.1 หลักการทำงานของ Level of Detail (LOD)

เป็นเทคนิคในการลดความซับซ้อนของโมเดล 3 มิติ โดยการสร้างเวอร์ชันของโมเดลในระดับความละเอียดที่แตกต่างกัน เพื่อลดการใช้ทรัพยากรของระบบ เช่น หน่วยประมวลผล (CPU) และการ์ดจอ (GPU) โดยเฉพาะอย่างยิ่งในเกมหรือแอปพลิเคชันที่ต้องประมวลผลฉากขนาดใหญ่หรือรายละเอียดซับซ้อน

- การสร้าง LOD

  โมเดลแต่ละชิ้นจะมี หลายระดับของความละเอียด (เช่น LOD0, LOD1, LOD2)

  – LOD0: ความละเอียดสูงสุด ใช้เมื่อวัตถุอยู่ใกล้กล้อง

  – LOD1: ลดจำนวนโพลิกอน (Polygons) ลง ใช้เมื่อวัตถุอยู่ในระยะปานกลาง

  – LOD2 และลำดับถัดไป: ความละเอียดต่ำสุด ใช้เมื่อวัตถุอยู่ไกลมาก

- การตรวจสอบระยะห่าง (Distance–Based Switching):

  ระบบจะคำนวณระยะห่างระหว่างกล้อง (Camera) กับโมเดล:

  – เมื่อโมเดลอยู่ใกล้กล้อง: ใช้ LOD ที่มีรายละเอียดสูง (เช่น LOD 0)

  – เมื่อโมเดลอยู่ไกลออกไป: สลับไปใช้ LOD ที่มีรายละเอียดต่ำกว่า (เช่น LOD 1, 2, หรือ 3)

- การสลับระดับ LOD แบบเรียลไทม์ (Dynamic LOD Switching):

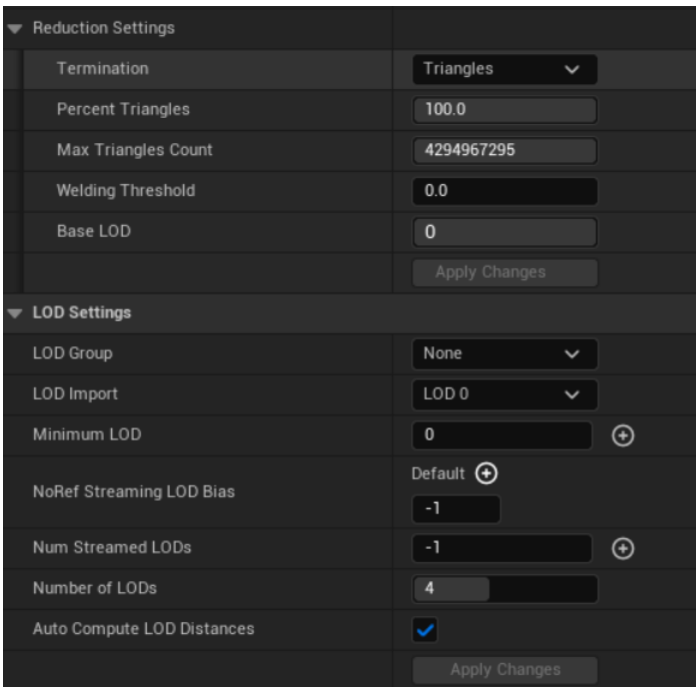
  ระบบจะสลับระหว่าง LOD ต่างๆ แบบอัตโนมัติตามระยะห่างและเงื่อนไขอื่น เช่น

  – มุมมองของผู้เล่น

– ขนาดของโมเดลที่แสดงผลบนหน้าจอ

– การประยุกต์ใช้กับ Texture (Texture LOD):
นอกจากโมเดลแล้ว เทคโนโลยี LOD ยังสามารถปรับลดคุณภาพของเท็กซ์เจอร์ได้ด้วย เช่น การใช้เท็กซ์เจอร์ความละเอียดต่ำสำหรับโมเดลที่อยู่ไกล

2.3.2 ขั้นตอนการใช้งาน Level of detail (LOD)



รูปที่ 2.6 หน้าต่างการตั้งค่า LOD ใน Unreal engine 5

– Reduction Settings

- Termination:

วิธีการกำหนดเงื่อนไขการลดจำนวนโพลิกอน (Polygons) ในโมเดล:เลือกเป็น Triangles หมายถึงการลดจำนวนโพลิกอนตามจำนวนสามเหลี่ยม (Triangles) ที่ต้องการ

- Percent Triangles:

เปอร์เซ็นต์ของจำนวนโพลิกอนที่ต้องการคงไว้ในโมเดล (ในกรณีนี้คือ 100.0% หมายถึงไม่มีการลดความละเอียด)

- Max Triangles Count:

จำนวนโพลิกอนสูงสุดที่โมเดลจะถูกลดเหลือ (ในภาพนี้ตั้งค่าเป็น 4,294,967,295 ซึ่งเป็นจำนวนสูงสุด แปลว่าไม่มีข้อจำกัด)

- Welding Threshold:

ระยะที่ใช้สำหรับการรวมจุด (Vertices) ใกล้กันเป็นจุดเดียวเพื่อลดความซับซ้อนของโมเดล (ตั้งค่าเป็น 0.0 หมายถึงไม่มีการรวมจุด)

- Base LOD:

ระดับ LOD พื้นฐานที่ใช้เป็นจุดเริ่มต้นสำหรับการลดรายละเอียด (ตั้งค่าเป็น 0 หมายถึงใช้ LOD ที่มีความละเอียดสูงสุดเป็นพื้นฐาน)

– LOD Settings

- LOD Group:

กลุ่มของการตั้งค่า LOD สำหรับประเภทโมเดล เช่น Static Meshes, Skeletal Meshes หรือ Props (ในกรณีนี้เลือก None หมายถึงไม่ได้เลือกกลุ่มเฉพาะ)

- LOD Import:

ระบุระดับ LOD ที่จะนำเข้า (ตั้งค่าเป็น LOD 0 หมายถึงนำเข้าระดับความละเอียดสูงสุด)

- Minimum LOD:

ระบุ LOD ขั้นต่ำที่จะแสดงผลในเกม (ในกรณีนี้ตั้งค่าเป็น 0)

- NoRef Streaming LOD Bias:

ตัวคูณที่ใช้กำหนดระยะการสตรีม LOD ในกรณีที่ไม่มีการอ้างอิง (ตั้งค่าเป็น –1 ซึ่งเป็นค่าเริ่มต้น)

- Num Streamed LODs:

จำนวน LOD ที่สามารถสตรีมเข้ามาได้ (ตั้งค่าเป็น –1 หมายถึงค่าเริ่มต้นให้ระบบคำนวณเอง)

- Number of LODs:

จำนวนระดับ LOD ทั้งหมดที่โมเดลนี้รองรับ (ตั้งค่าเป็น 4 หมายถึงมี 4 ระดับ LOD)

- Auto Compute LOD Distances:

ตัวเลือกให้ระบบคำนวณระยะห่างระหว่าง LOD แต่ละระดับโดยอัตโนมัติ (เลือก เปิดใช้งาน

รูปที่ 2.7 รายละเอียดของ LOD0

รูปที่ 2.8 รายละเอียดของ LOD3รูปที่ 2.7 รายละเอียดของ LOD0

รูปที่ 2.8 รายละเอียดของ LOD3

รูปที่ 3.3 บริบทการทดสอบ RVT (Runtime Virtual Texture)รูปที่ 2.8 รายละเอียดของ LOD3

- ข้อดีของ LOD
    - ช่วยเพิ่มประสิทธิภาพในการประมวลผลกราฟิก

ระบบจะใช้โมเดลที่มีจำนวนโพลิกอนต่ำสำหรับวัตถุที่อยู่ไกลจากกล้อง ซึ่งลดการประมวลผลที่จำเป็นของ GPU และ CPU โดยไม่กระทบต่อคุณภาพของภาพในระดับที่ผู้ใช้สามารถสังเกตได้ส่งผลให้ฉากที่มีวัตถุจำนวนมาก เช่น Open-World หรือ Large-Scale Environment แสดงผลได้ลื่นไหลขึ้นแม้ในอุปกรณ์ที่มีสเปกต่ำกว่า

    - ช่วยลดการใช้หน่วยความจำ

โมเดลที่มีรายละเอียดต่ำใช้หน่วยความจำน้อยกว่า เช่น จำนวนโพลิกอนและขนาดของเท็กซ์เจอร์ลดลง
ลดการโหลดข้อมูลเข้าสู่หน่วยความจำขณะเล่นเกม ทำให้การสลับระหว่างฉากหรือการโหลดใหม่ทำได้รวดเร็วขึ้น

    - เหมาะสำหรับการพัฒนา Open-World Games

เกมขนาดใหญ่ที่มีพื้นที่กว้าง เช่น Far Cry, The Legend of Zelda: Breath of the Wild ใช้ LOD เพื่อให้สามารถแสดงผลฉากที่มีรายละเอียดสูงได้โดยไม่กระทบต่อเฟรมเรต
ผู้พัฒนาสามารถจัดการโมเดลในระยะไกลที่ไม่ค่อยสำคัญได้อย่างมีประสิทธิภาพ

    - ลดอาการเฟรมเรตตก (Frame Drop)

เมื่อแสดงผลโมเดลที่อยู่ไกลด้วยรายละเอียดต่ำ เฟรมเรตจะเสถียรขึ้น เนื่องจากการประมวลผลลดลงและช่วยให้เกมหรือแอปพลิเคชันสามารถแสดงผลกราฟิกคุณภาพสูงได้แม้ในอุปกรณ์ที่ทรัพยากรจำกัด

- ข้อเสียของ LOD
    - เพิ่มเวลาและทรัพยากรในการพัฒนา

นักพัฒนาต้องสร้างโมเดลหลายเวอร์ชันสำหรับ LOD เช่น LOD 0 ถึง LOD 4 ซึ่งต้องใช้เวลาและทรัพยากรเพิ่มเติมในการออกแบบและปรับแต่งแต่ละระดับของโมเดล
หากไม่มีระบบลดโพลิกอนอัตโนมัติ (เช่น Reduction Tool) อาจต้องใช้แรงงานมนุษย์เพิ่มขึ้น

    - เกิดปัญหา "Pop-in" หรือการเปลี่ยนระดับที่ไม่ราบรื่น

หากการตั้งค่าระยะห่างของ LOD ไม่เหมาะสม (เช่น สลับ LOD ใกล้เกินไป) ผู้เล่นอาจเห็นการเปลี่ยนแปลงของโมเดลแบบฉับพลัน (Pop-in Effect) ซึ่งทำลายความสมจริงและลด

    - คุณภาพของประสบการณ์ผู้เล่น

ปัญหานี้อาจเด่นชัดในฉากที่มีการเคลื่อนไหวเร็ว เช่น การขับรถหรือบิน

    - ความซับซ้อนในการจัดการ LOD

การตั้งค่าระบบ LOD ต้องใช้การคำนวณระยะและการทดลองเพื่อหาจุดที่สมดุลระหว่าง

- คุณภาพของภาพและประสิทธิภาพการทำงาน

การผสาน LOD เข้ากับเทคโนโลยีอื่น เช่น Runtime Virtual Texture (RVT) หรือ Dynamic Shadows ต้องการความรู้เทคนิคที่สูง

- คุณภาพของภาพลดลงในบางกรณี

วัตถุในระยะไกลอาจดูไม่ชัดเจนหรือขาดรายละเอียดจนเกินไปหากลดระดับ LOD มากเกินควรส่งผลต่อประสบการณ์ผู้ใช้ในเกมที่ต้องการรายละเอียดสูง เช่น เกมที่มุ่งเน้นความสมจริง (Realism–Based Games)

- ปัญหาในการจัดการในกรณีหลายแพลตฟอร์ม

หากเกมหรือโปรเจกต์ต้องทำงานในแพลตฟอร์มที่มีความสามารถแตกต่างกัน (เช่น PC, Console, Mobile) การตั้งค่า LOD ให้เหมาะสมกับแต่ละแพลตฟอร์มอาจยุ่งยากและใช้เวลานาน

2.4 งานวิจัยที่เกี่ยวข้อง

2.4.1 Comparative analysis of Unity and Unreal Engine efficiency in creating virtual exhibitions of 3D scanned models

งานวิจัยได้ดำเนินการพัฒนาแอปพลิเคชันต้นแบบ 2 ชุดที่มีลักษณะเหมือนกัน โดยใช้ทรัพยากร (assets) เดียวกัน แต่ออกแบบและพัฒนาแยกต่างหากใน Unity และ Unreal Engine เพื่อเปรียบเทียบประสิทธิภาพในแง่ของการใช้ทรัพยากรฮาร์ดแวร์ รวมถึงขนาดของไฟล์โปรเจกต์

Computer	Processor (CPU)	Graphic card (GPU)	RAM
PC 1	AMD Ryzen 7 3700x, 3.60GHz	AMD Radeon RX580	32 GB
PC 2	Intel i5-4460, 3.20GHz	AMD Radeon R9 270x	8 GB
Laptop 1	Intel i5-8250U, 1.60GHz	NVIDIA GeForce 940MX	8 GB
Laptop 2	Intel i5-7200U, 2.50GHz	NVIDIA GeForce 940MX	8 GB

รูปที่ 2.9 รายละเอียดคอมพิวเตอร์ที่ใช้ในการทดลอง

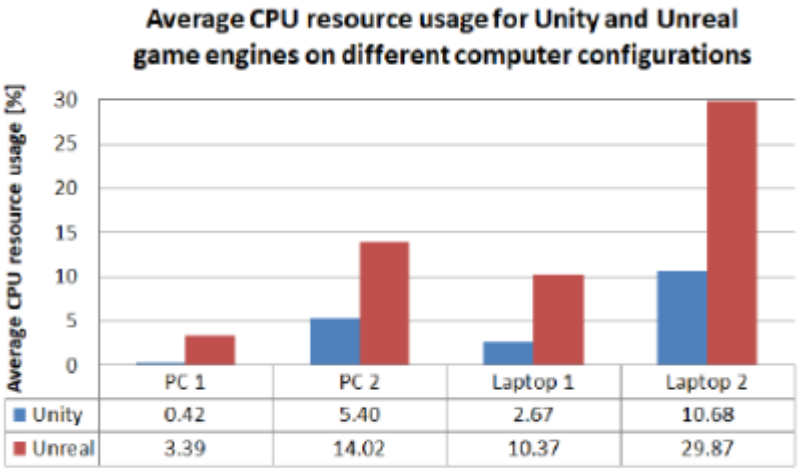
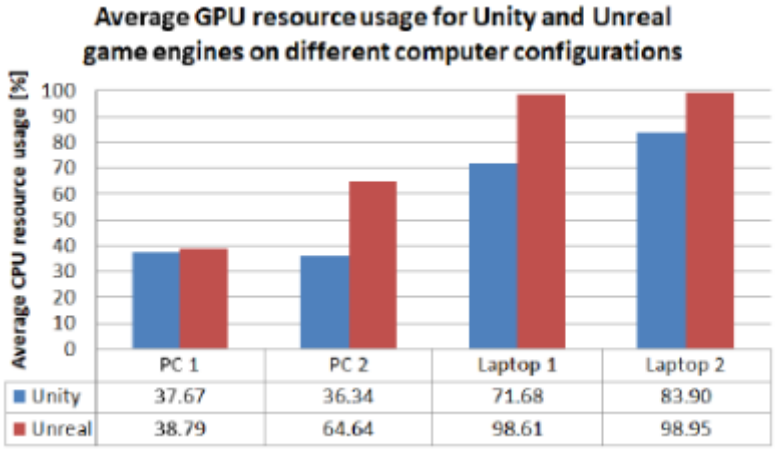


Figure 9: Average CPU usage.

รูปที่ 2.10 ค่าเฉลี่ยการใช้งานหน่วยประมวลผลหลัก



รูปที่ 2.11 ค่าเฉลี่ยการใช้งานหน่วยประมวลผลกราฟิก

- ข้อดีของงานวิจัยที่เกี่ยวข้อง
  - เปรียบเทียบเชิงปฏิบัติการ (Practical Comparison):

    งานวิจัยนี้นำเสนอการทดลองเปรียบเทียบจริงระหว่าง Unity และ Unreal Engine ในการสร้างแอปพลิเคชันนิทรรศการเสมือนจริง ทำให้ผู้อ่านสามารถเห็นภาพชัดเจนในด้านประสิทธิภาพและการใช้ทรัพยากรของทั้งสองเอนจิน
  - ให้ข้อมูลที่ใช้ตัดสินใจได้ง่าย:

    ผลลัพธ์ที่ได้ช่วยให้ผู้พัฒนา (developers) และองค์กรสามารถเลือกเอนจินที่เหมาะสมตามลักษณะงานหรือข้อจำกัดของทรัพยากรฮาร์ดแวร์

- ครอบคลุมหลายมิติของประสิทธิภาพ:

งานวิจัยวิเคราะห์ทั้งการใช้ทรัพยากร CPU, GPU, RAM, FPS และขนาดของไฟล์โปรเจกต์ ซึ่งเป็นปัจจัยสำคัญสำหรับการพัฒนาแอปพลิเคชัน

- เหมาะสมสำหรับอุตสาหกรรมเฉพาะทาง:

มีประโยชน์โดยเฉพาะในอุตสาหกรรมที่เกี่ยวข้องกับการสแกน 3 มิติและการสร้างนิทรรศการดิจิทัล เช่น พิพิธภัณฑ์หรือการอนุรักษ์มรดกวัฒนธรรม

– ข้อเสียของงานวิจัยที่เกี่ยวข้อง

- ข้อจำกัดของบริบทการทดลอง:

การทดลองใช้ตัวแปรเพียง 1 ชุดในการสร้างแอปพลิเคชันต้นแบบ (virtual exhibition) อาจไม่ครอบคลุมทุกกรณีของการใช้งาน Unity และ Unreal Engine ในลักษณะงานที่แตกต่างกัน เช่น การพัฒนาเกมหรือแอปพลิเคชัน AR/VR

- ฮาร์ดแวร์ที่ใช้ทดสอบไม่หลากหลาย:

การทดลองอาจไม่ได้แสดงผลลัพธ์ที่เหมาะสมในทุกสภาพแวดล้อมของฮาร์ดแวร์ เช่น อุปกรณ์มือถือหรือระบบที่มีทรัพยากรจำกัด

- ไม่ได้กล่าวถึงประสบการณ์ผู้ใช้ (User Experience):

แม้ว่าจะมีการวิเคราะห์ FPS และประสิทธิภาพโดยรวม แต่งานวิจัยไม่ได้กล่าวถึงความง่ายในการพัฒนา (developer experience) หรือความแตกต่างในการจัดการเนื้อหาในแต่ละเอนจิน

- ไม่มีการวิเคราะห์เชิงลึกด้านกราฟิกคุณภาพ:

แม้ Unreal Engine จะมีข้อดีด้านคุณภาพกราฟิกสูง แต่ไม่ได้รับการวิเคราะห์หรือเปรียบเทียบกับ Unity ในมุมนี้ ซึ่งอาจเป็นจุดสำคัญสำหรับผู้ที่ต้องการคุณภาพกราฟิกสูง

- ขาดการประเมินด้านต้นทุน:

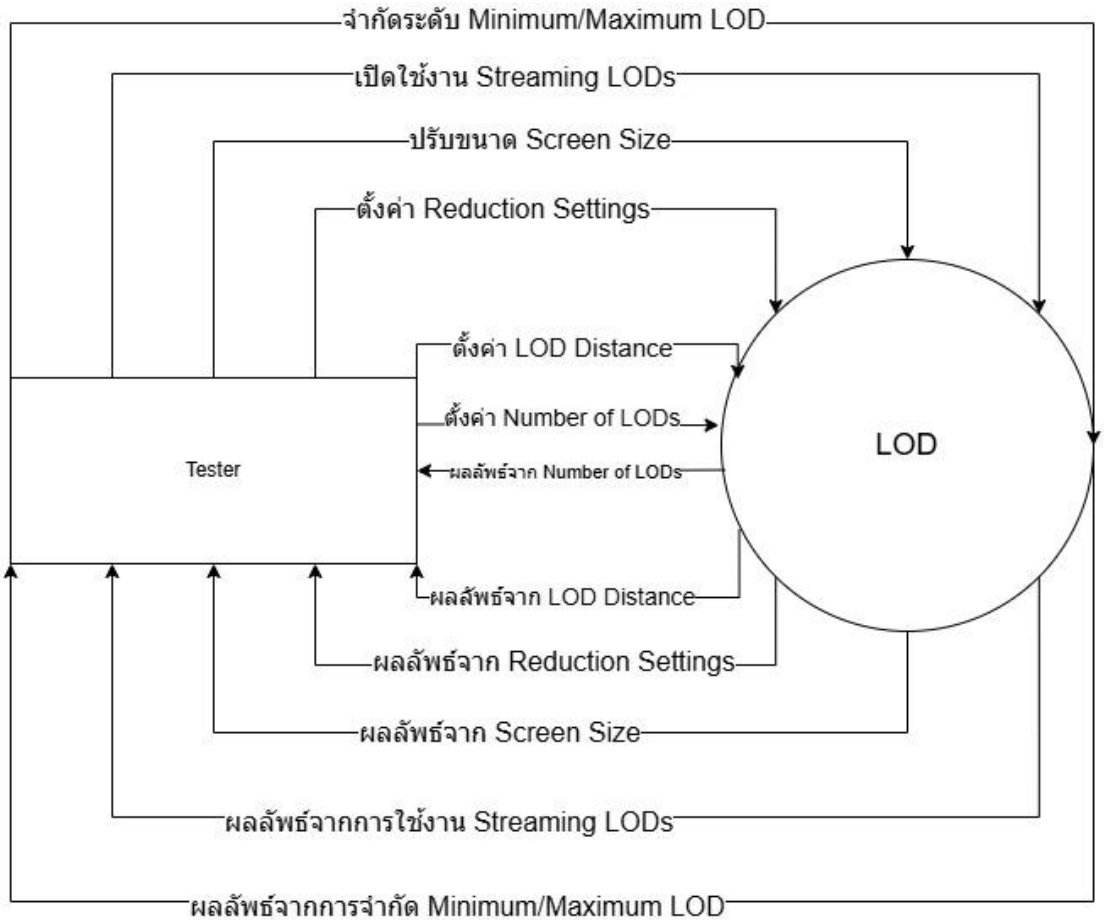
งานวิจัยไม่ได้กล่าวถึงต้นทุนการพัฒนา เช่น ค่าใช้จ่ายในการซื้อไลเซนส์หรือความพร้อมใช้งานของปลั๊กอินและเครื่องมือเสริมในทั้งสองเอนจิน

บทที่ 3
การวิเคราะห์และออกแบบระบบ

ในบทนี้ กล่าวถึงการวิเคราะห์และออกแบบระบบ ซึ่งประกอบไปด้วยโครงสร้างของระบบทั้งหมด โดยแบ่งออกเป็นสองส่วนหลัก คือ ส่วนของการออกแบบระบบเพื่อเพิ่มประสิทธิภาพของเกม VR ด้วยเทคนิค Runtime Virtual Texture (RVT), Nanite และ Level of Detail (LOD) และส่วนของการออกแบบระบบในส่วนของการเก็บและจัดการข้อมูลที่เกี่ยวข้อง

3.1 การวิเคราะห์การทำงานระบบ

3.1.1 การทดสอบด้วย LOD(Level of detail)

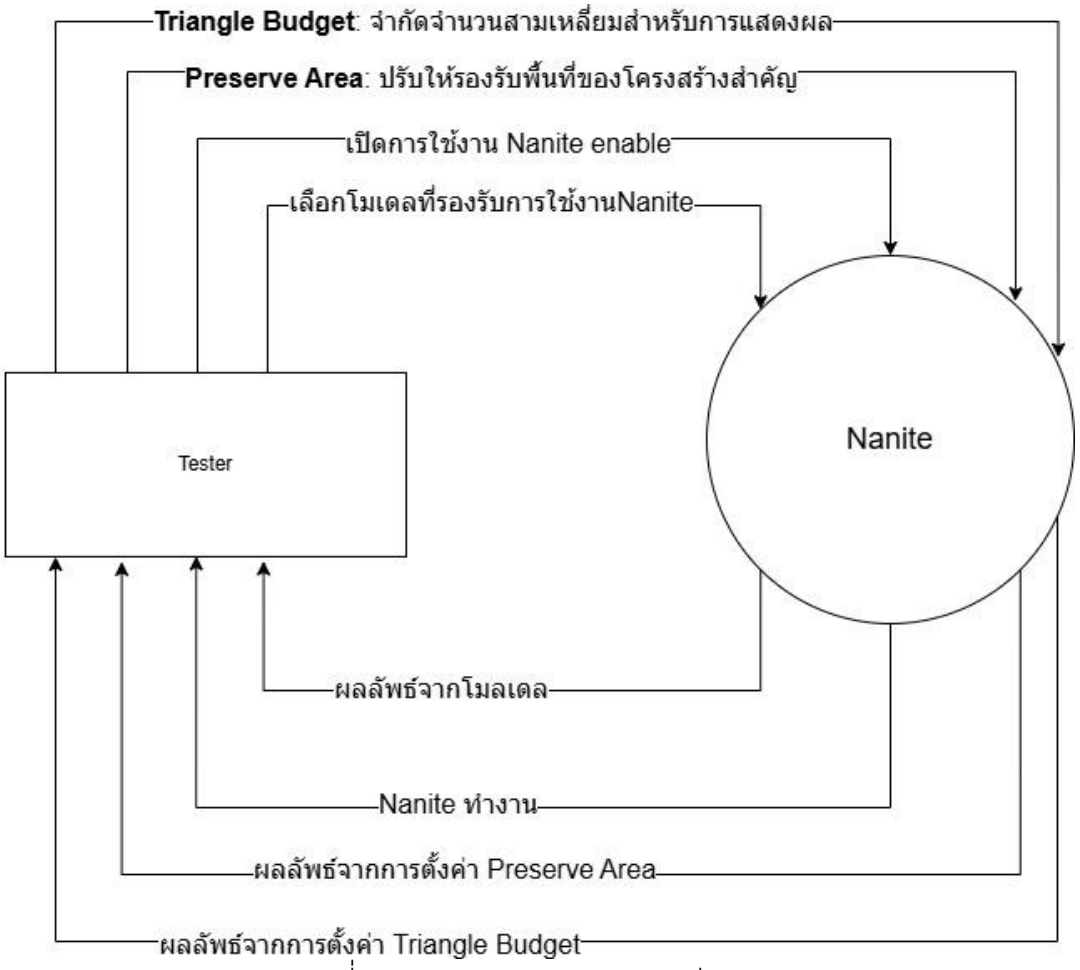


รูปที่ 3.1 บริบทการทดสอบทฤษฎี LOD (Level of detail)

- กระบวนการวิเคราะห์ LOD
  - Tester ปรับค่าพารามิเตอร์ต่าง ๆ เช่น Minimum/Maximum LOD, Streaming LODs และ Screen Size เพื่อปรับแต่งระดับรายละเอียดของโมเดล
  - การตั้งค่าระยะทาง (LOD Distance) และจำนวนของ LOD (Number of LODs) ถูกออกแบบเพื่อควบคุมการแสดงผลให้เหมาะสมกับการประมวลผล
  - กระบวนการปรับค่าลดรายละเอียด (Reduction Settings) จะช่วยลดการใช้ทรัพยากรในระยะไกล

ผลลัพธ์จากการปรับแต่งทั้งหมดนี้จะถูกส่งกลับมาให้ Tester วิเคราะห์และปรับแต่ง

3.1.2 การทดสอบด้วย Nanite

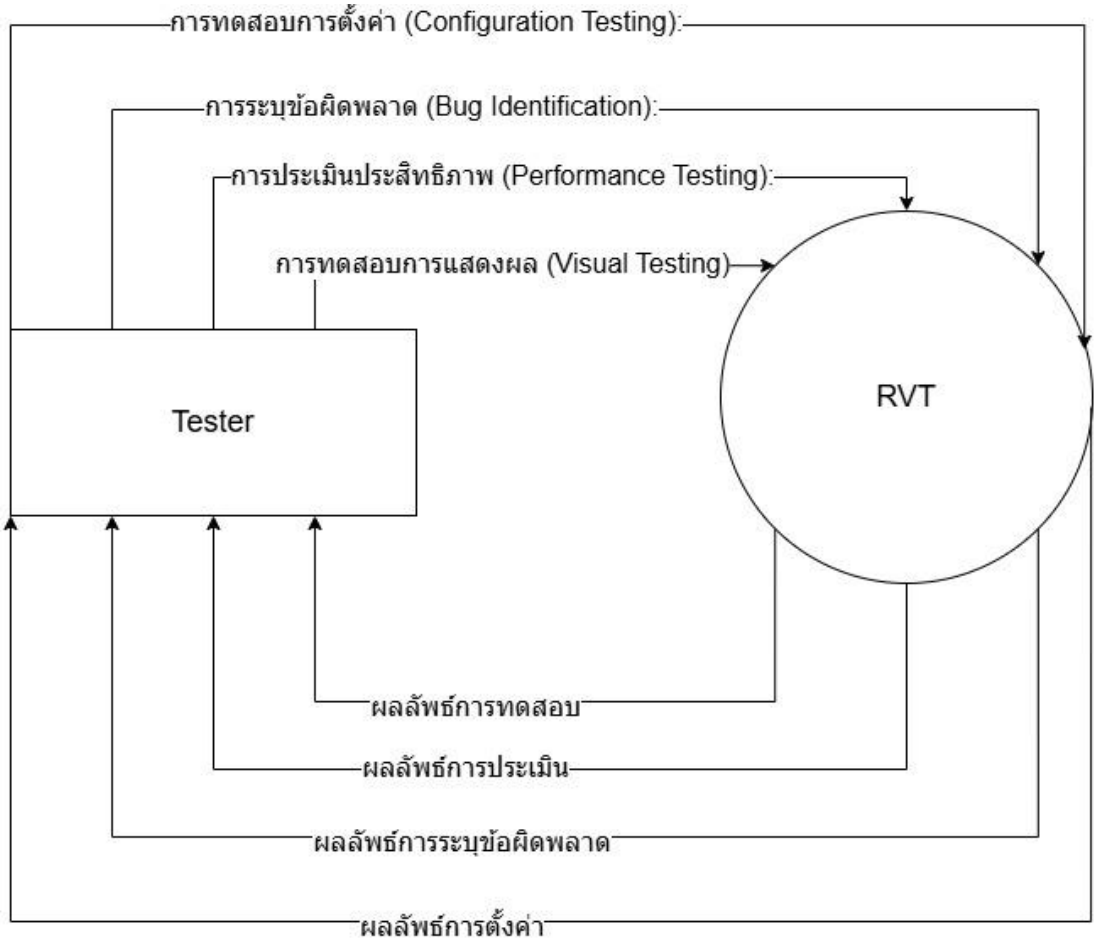


รูปที่ 3.2 บริบทการทดสอบทฤษฎี Nanite

- กระบวนการทำงาน Nanite

Tester ทำการตั้งค่าพารามิเตอร์ Nanite เช่น Triangle Budget เพื่อจำกัดจำนวนสามเหลี่ยมที่ใช้ในการสร้างโมเดล รวมถึง Preserve Area เพื่อคงพื้นที่สำคัญ

- กระบวนการเปิดใช้งาน Nanite และเลือกโมเดลที่รองรับ Nanite จะทำให้ระบบปรับปรุงโมเดลให้เหมาะสมสำหรับการแสดงผลที่มีความละเอียดสูง
- การประมวลผล Nanite ทำงานโดยจัดการทรัพยากรระบบให้มีประสิทธิภาพสูงสุด
- ผลลัพธ์ที่ได้จากโมเดล Nanite จะถูกส่งกลับมาให้ Tester ประเมินและปรับปรุงต่อ

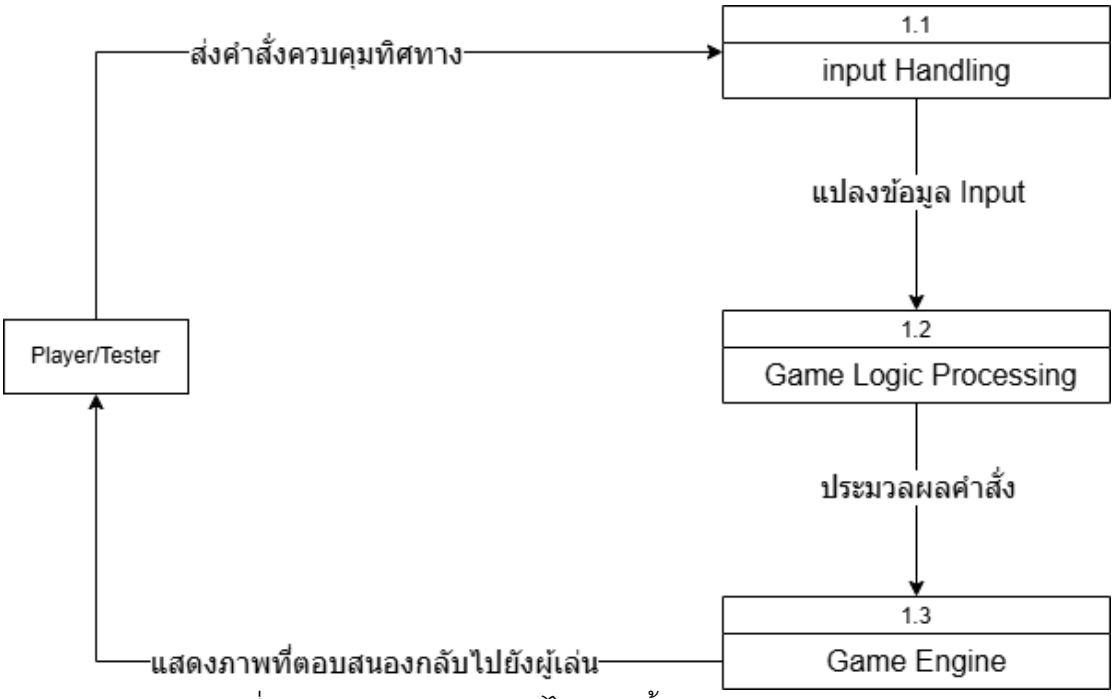


รูปที่ 3.3 บริบทการทดสอบ RVT (Runtime Virtual Texture)

รูปที่ 3.6 หน้าต่างการตั้ง LOD (Level of detail)รูปที่ 3.3 บริบทการทดสอบ RVT (Runtime Virtual Texture)

- กระบวนการตั้งค่าและทดสอบ RVT (Runtime Virtual Texture)
  Tester ทำการตั้งค่าการทดสอบ RVT โดยผ่านกระบวนการต่าง ๆ เช่น การทดสอบการตั้งค่า (Configuration Testing) เพื่อปรับแต่งพารามิเตอร์ให้เหมาะสม
- การตรวจจับข้อผิดพลาด (Bug Identification) เป็นขั้นตอนที่ใช้ในการวิเคราะห์ปัญหา เช่น ข้อผิดพลาดของพื้นผิวเสมือนจริง
- การประเมินประสิทธิภาพ (Performance Testing) ใช้ทดสอบประสิทธิภาพของ RVT ในการลดภาระการแสดงผล
- การทดสอบการแสดงผล (Visual Testing) ใช้เพื่อวิเคราะห์คุณภาพการแสดงผลภาพ ผลลัพธ์จากการทดสอบแต่ละส่วนจะถูกนำกลับมาพิจารณาเพื่อปรับปรุง RVT อย่างต่อเนื่อง

3.2 แผนภาพแสดงการไหลของข้อมูล



รูปที่ 3.4 แผนภาพแสดงการไหลของข้อมูล

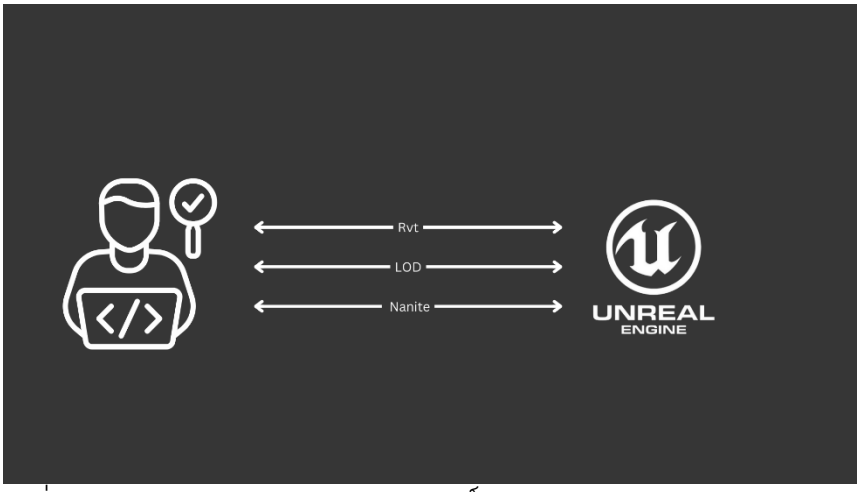
แผนภาพกระแสข้อมูลนี้แสดงกระบวนการทำงานของระบบควบคุมเกม ตั้งแต่ผู้เล่นส่งคำสั่งไปจนถึงการแสดงผลตอบกลับ โดยแบ่งออกเป็นสามกระบวนการหลัก ดังนี้:

1. การจัดการอินพุต (Input Handling – 1.1)
   - ผู้เล่นหรือผู้ทดสอบ (Player/Tester) ส่งคำสั่งควบคุมไปยังระบบ
   - ระบบทำการรับและแปลงข้อมูลอินพุตให้อยู่ในรูปแบบที่สามารถนำไปประมวลผลได้

2. การประมวลผลตรรกะของเกม (Game Logic Processing – 1.2)
   - กระบวนการนี้ทำหน้าที่ตรวจสอบและประมวลผลข้อมูลอินพุตที่ได้รับ
   - คำนวณผลลัพธ์ของคำสั่ง เช่น การเคลื่อนที่ของตัวละครหรือการโต้ตอบกับวัตถุภายในเกม

3. เอนจินเกม (Game Engine – 1.3)
   - ผลลัพธ์จากการประมวลผลจะถูกส่งไปยังเอนจินเกม
   - Game Engine ทำหน้าที่สร้างและแสดงผลภาพ รวมถึงตอบสนองกลับไปยังผู้เล่น

3.3 การออกแบบระบบ

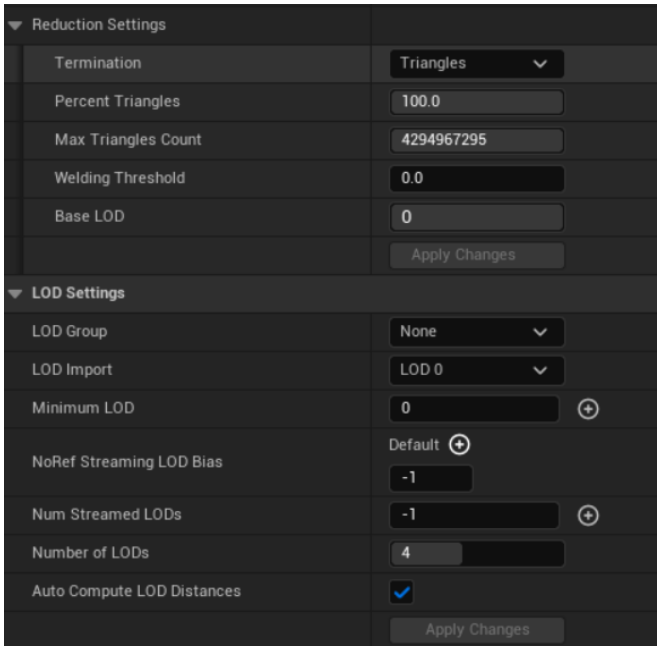


รูปที่ 3.5 แสดงบทบาทและความสัมพันธ์ของ Tester กับ Unreal engine 5

จากรูปที่ 3.5 ระบบนี้ได้รับการออกแบบให้ทำงานร่วมกันระหว่างผู้พัฒนา (Tester) และ Unreal Engine โดยมีการสื่อสารผ่านฟีเจอร์ 3 ส่วนหลัก ได้แก่ RVT, LOD และ Nanite เพื่อเพิ่มความสามารถในการจัดการทรัพยากรและเพิ่มความลื่นไหลของการแสดงผลในเกม VR

3.2.1 การออกแบบระบบ LOD (Level of deatail)

ในกระบวนการออกแบบระบบ LOD ได้รับการวางแผนเพื่อเพิ่มประสิทธิภาพในการแสดงผลของวัตถุสามมิติในโปรเจกต์ VR โดยใช้ Unreal Engine ระบบ LOD จะช่วยลดจำนวน polygon ของโมเดลเมื่ออยู่ในระยะไกลจากกล้อง เพื่อลดการใช้ทรัพยากรของระบบโดยไม่ลดคุณภาพของภาพที่สังเกตได้



รูปที่ 3.6 หน้าต่างการตั้ง LOD (Level of detail)

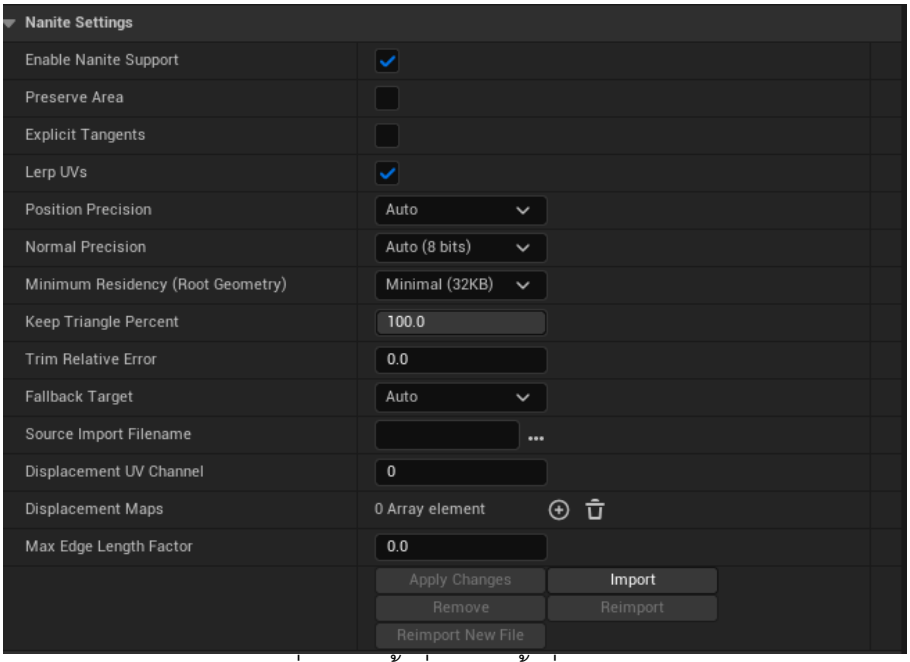
การตั้งค่า LOD เบื้องต้นใน Unreal Engine

- Termination: เลือก "Triangles" สำหรับการลด polygon
- Percent Triangles: ตั้งค่าไว้ที่ 50%–100% ขึ้นกับความละเอียดที่ต้องการ
- Number of LODs: กำหนด 3–4 ระดับ (LOD0, LOD1, LOD2, ฯลฯ)
- Auto Compute LOD Distances: เปิดใช้งานเพื่อลดการตั้งค่าด้วยตนเอง
- Minimum LOD: ตั้งค่าระดับ LOD ต่ำสุดที่แสดง

การตั้งค่านี้เหมาะสำหรับการปรับสมดุลระหว่างคุณภาพกราฟิกและประ สิทธิภาพของระบบ

3.2.2 การออกแบบระบบ Nanite

ในกระบวนการออกแบบระบบ Nanite ได้รับการวางแผนเพื่อเพิ่มประสิทธิภาพในการจัดการวัตถุสามมิติในโปรเจกต์ VR โดยใช้ Unreal Engine ระบบ Nanite ช่วยในการแสดงผลโมเดลที่มีความซับซ้อนสูงโดยอัตโนมัติ ด้วยการจัดการจำนวน polygon อย่างชาญฉลาด ทำให้สามารถแสดงผลได้อย่างมีประสิทธิภาพโดยไม่ลดคุณภาพของภาพที่มองเห็นได้ และลดภาระของระบบในการประมวลผล Geometry ที่ไม่จำเป็น ทั้งนี้ Nanite ยังช่วยลด Draw Calls และเพิ่มความลื่นไหลของเฟรมเรตในฉากที่มีโมเดลจำนวนมาก



รูปที่ 3.7 หน้าต่างการตั้งค่า Nanite

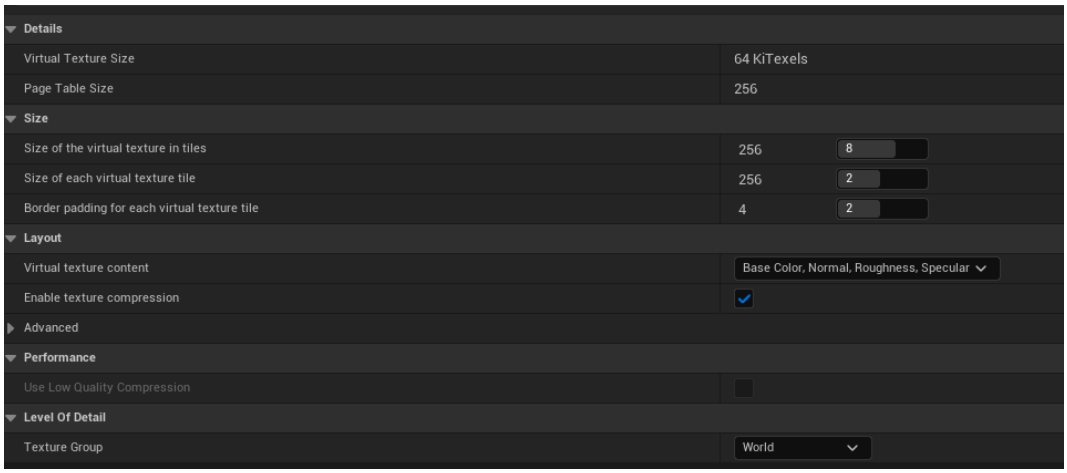
- Position Precision: ค่านี้ควบคุมความละเอียดของตำแหน่งของจุดยอด (Vertex) ยิ่งค่าสูงยิ่งละเอียด แต่ก็จะกินทรัพยากรมากขึ้น
- Normal Precision: ค่านี้ควบคุมความละเอียดของข้อมูล Normal (ข้อมูลที่บอกทิศทางของพื้นผิว) ยิ่งค่าสูงพื้นผิวก็จะดูเรียบเนียนและสมจริงมากขึ้น
- Minimum Residency: ค่านี้กำหนดปริมาณข้อมูลที่น้อยที่สุดที่ต้องเก็บไว้ในหน่วยความจำ เพื่อให้การแสดงผลเป็นไปอย่างราบรื่น
- Keep Triangle Percent: ค่านี้กำหนดเปอร์เซ็นต์ของสามเหลี่ยมที่จะถูกเก็บไว้เมื่อทำการลดรายละเอียดของโมเดล
- Trim Relative Error: ค่านี้กำหนดค่าความคลาดเคลื่อนสูงสุดที่ยอมรับได้ในการลดรายละเอียดของโมเดล

รูปที่ 3.8 แสดงค่ารายละเอียดของโมเดลที่ใช้งาน Nanite

ข้อมูลเหล่านี้บอกถึงรายละเอียดทางเทคนิคเกี่ยวกับวิธีการที่ Unreal Engine 5 ใช้ในการแสดงผลโมเดล 3 มิติ โดยเฉพาะอย่างยิ่งส่วนที่เกี่ยวข้องกับระบบ Nanite ซึ่งเป็นเทคโนโลยีที่ช่วยให้แสดงผลโมเดลที่มีรายละเอียดสูงได้อย่างมีประสิทธิภาพ

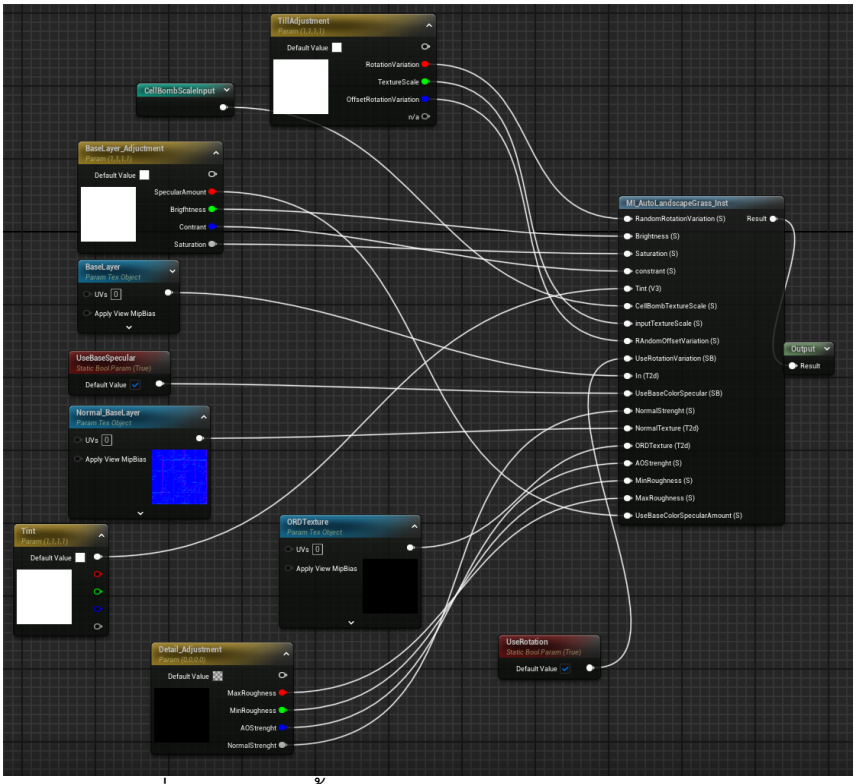
- Nanite Enabled: ระบบ Nanite เปิดใช้งานอยู่ หมายความว่าระบบกำลังทำงานเพื่อแสดงผลโมเดล 3 มิติที่มีรายละเอียดสูง
- Position Precision: ความละเอียดในการกำหนดตำแหน่งของจุดยอด (Vertex) ของโมเดล ยิ่งค่านี้สูง โมเดลก็จะมีรายละเอียดมากขึ้น แต่ก็จะกินทรัพยากรมากขึ้นด้วย
- Normal Precision: ความละเอียดของข้อมูล Normal ซึ่งบอกทิศทางของพื้นผิว ยิ่งค่านี้สูง พื้นผิวก็จะดูเรียบเนียนและสมจริงมากขึ้น
- GPU Memory: ปริมาณหน่วยความจำของการ์ดจอที่ใช้ในการประมวลผลโมเดล
- Current Screen Size: ขนาดของวัตถุบนหน้าจอเมื่อเทียบกับขนาดจริง
- Nanite Triangles, Nanite Vertices: จำนวนรูปสามเหลี่ยมและจุดยอดที่ระบบ Nanite ใช้ในการสร้างโมเดล
- Fallback Triangles, Fallback Vertices: จำนวนรูปสามเหลี่ยมและจุดยอดที่ใช้เมื่อระบบ Nanite ไม่สามารถทำงานได้
- UV Channels: จำนวนช่องที่ใช้ในการแมป Texture ลงบนโมเดล
- Distance Field: ข้อมูลที่ใช้ในการคำนวณระยะห่างระหว่างวัตถุ เพื่อใช้ในการสร้างเงาและเอฟเฟกต์ต่างๆ
- Approx Size: ขนาดโดยประมาณของวัตถุ
- Num Collision Primitives: จำนวนปริมาตรที่ใช้ในการตรวจจับการชน
- Estimated Compressed Disk Size: ขนาดไฟล์ที่ใช้ในการเก็บข้อมูลของโมเดลบนดิสก์

3.2.3 การออกแบบระบบ Runtime Virtual Texture (RVT)



รูปที่ 3.9 หน้าต่างการตั้งค่าของไฟล์ Runtime Virtual Texture (RVT)

ไฟล์ RVT อาจไม่สามารถใช้งานได้โดยตรงใน Unreal Engine 5 อย่างไรก็ตาม ยังมีวิธีการนำเข้าข้อมูลจากไฟล์ Revit เพื่อใช้ในการสร้างสภาพแวดล้อมสามมิติสำหรับเกมหรือแอปพลิเคชัน โดยการใช้ Node Graph ที่ได้รับการพัฒนาขึ้นเพื่อช่วยในการแปลงข้อมูลจากไฟล์ RVT ให้อยู่ในรูปแบบที่สามารถใช้งานร่วมกับ Unreal Engine ได้อย่างมีประสิทธิภาพ



รูปที่ 3.10 การสร้าง Material สำหรับ Autolandscape

ขั้นตอนการสร้าง Material Autolandscape

1. สร้าง Material ใหม่:

ใน Unreal Engine ให้เปิด Material Editor ขึ้นมา แล้วสร้าง Material ใหม่

ตั้งชื่อ Material ให้สื่อความหมาย เช่น LandscapeMaterial

2. กำหนดค่าพื้นฐาน:

Base Color: กำหนดสีพื้นฐานของวัสดุ เช่น สีเขียวสำหรับหญ้า สีน้ำตาลสำหรับดิน

Roughness: กำหนดความหยาบของพื้นผิว ยิ่งค่าสูง พื้นผิวก็จะยิ่งดูหยาบ

Metallic: กำหนดความเป็นโลหะของพื้นผิว

Specular: กำหนดความมันวาวของพื้นผิว

3. ใช้ Node เพื่อสร้างความหลากหลาย:

Noise Texture: สร้างความแตกต่างของสีและความหยาบของพื้นผิว

Panner: ทำให้ Texture เคลื่อนไหวได้

Mask: ใช้เพื่อควบคุมการผสมผสานระหว่าง Texture ต่างๆ

World Position Offset: ใช้เพื่อสร้างความแตกต่างของสีและความหยาบตามตำแหน่งในโลก

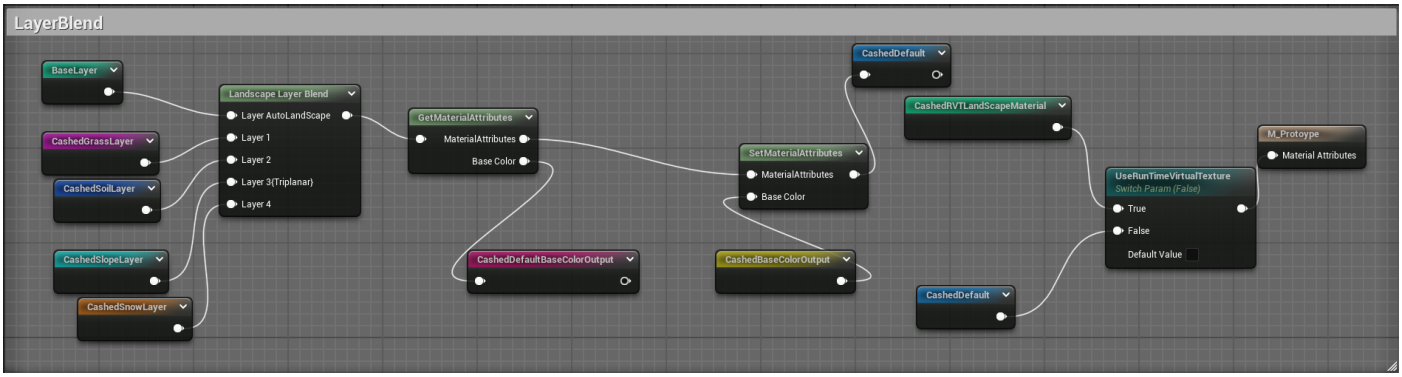
Texture Sample: ใช้เพื่อนำ Texture มาใช้ใน Material

4. เชื่อมต่อ Node:

เชื่อมต่อ Node ต่างๆ เข้าด้วยกันเพื่อสร้างเอฟเฟกต์ที่ต้องการ เช่น การผสมผสานระหว่าง Texture ที่แตกต่างกัน หรือการสร้างความแตกต่างของสีตามความสูงของพื้นผิว

5. ปรับแต่งค่า:

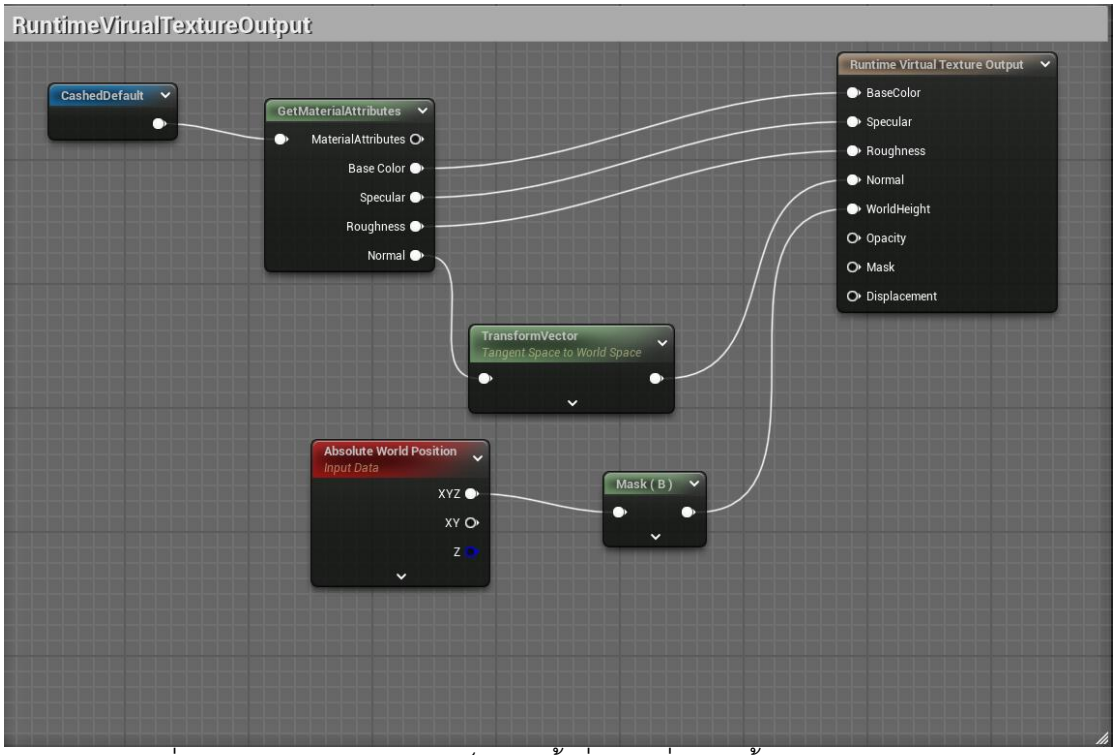
ปรับค่าต่างๆ ของ Node เพื่อให้ได้ผลลัพธ์ที่ต้องการ เช่น ความถี่ของ Noise, ความเร็วของ Panner, และค่า Mask



รูปที่ 3.11 การผสมเลเยอร์ของ Material Autolandscape

รูปที่ 3.13 ตั้งค่ากราฟวัสดุ (Material Graph) เพื่อเชื่อมโยงข้อมูลจาก Runtime Virtual Texture และส่งข้อมูลเหล่านั้นไปยังระบบวัสดุแบบ Landscapeรูปที่ 3.11 การผสมเลเยอร์ของ Material Autolandscape

2. สร้าง Layer: เพิ่ม Layer เข้าไปใน Material โดยใช้ Node ชื่อ "Layer Blend"
3. กำหนด Material ให้กับ Layer: เชื่อมต่อ Material ที่ต้องการผสมเข้ากับแต่ละ Layer
4. สร้าง Mask: สร้าง Mask โดยใช้ Node ต่างๆ เช่น Noise Texture, World Position Offset หรือ Texture Sample
5. เชื่อมต่อ Mask กับ Layer: เชื่อมต่อ Mask เข้ากับ Layer เพื่อควบคุมการปรากฏของแต่ละ Layer
6. ปรับแต่ง Blend Mode: เลือก Blend Mode ที่เหมาะสมกับเอฟเฟกต์ที่ต้องการ
7. ปรับค่าต่างๆ: ปรับค่าของ Node ต่างๆ เพื่อให้ได้ผลลัพธ์ที่ต้องการ



รูปที่ 3.12 ระบบการทำงานเป็นการตั้งค่าการส่งออกข้อมูลวัสดุ (Material)

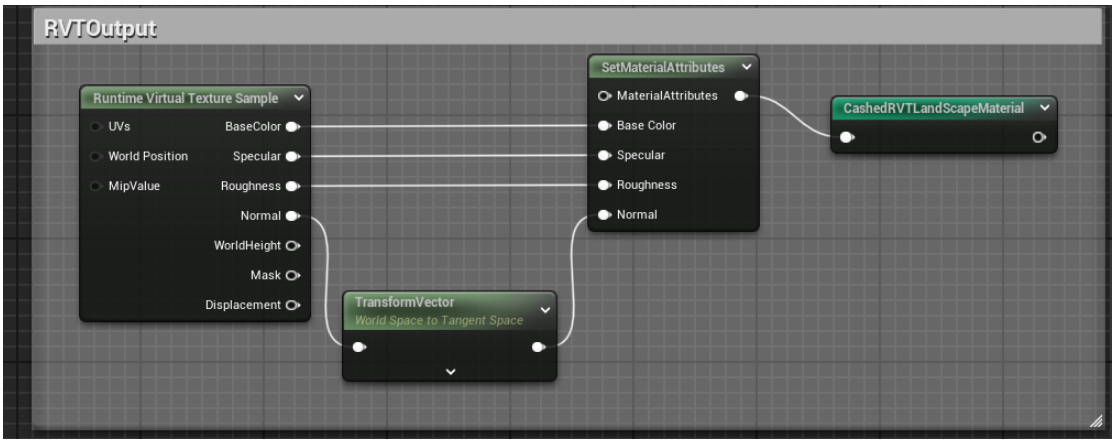
ขั้นตอนการสร้าง Material ตาม Node Graph

1. เพิ่ม Node: ลาก Node ต่างๆ ที่ต้องการมาวางใน Material Editor ตามภาพ
2. เชื่อมต่อ Node: เชื่อมต่อ Node ต่างๆ เข้าด้วยกันตามภาพ
3. ปรับค่า: ปรับค่าต่างๆ ของ Node เพื่อให้ได้ผลลัพธ์ที่ต้องการ เช่น

   ค่าสี: ปรับค่าสีใน Node GetMaterialAttributes

   ค่า Mask: ปรับค่าใน Node Mask เพื่อควบคุมการผสมของ Texture

   ค่า Transform: ปรับค่าใน Node TransformVector เพื่อควบคุมการแปลงพิกัด



รูปที่ 3.13 ตั้งค่ากราฟวัสดุ (Material Graph) เพื่อเชื่อมโยงข้อมูลจาก Runtime Virtual Texture และส่งข้อมูลเหล่านั้นไปยังระบบวัสดุแบบ Landscape

รูปที่ 3.15 ตัวอย่างฉากป่ารูปที่ 3.13 ตั้งค่ากราฟวัสดุ (Material Graph) เพื่อเชื่อมโยงข้อมูลจาก Runtime Virtual Texture และส่งข้อมูลเหล่านั้นไปยังระบบวัสดุแบบ Landscape

– เพิ่ม Runtime Virtual Texture Sample และเลือก RVT Asset ที่ต้องการในช่อง Virtual Texture.

– เชื่อมต่อค่าดังนี้:

Base Color, Specular, Roughness และ Normal เชื่อมต่อไปยัง Set Material Attributes.

– เชื่อมต่อ Normal ผ่าน Transform Vector เพื่อแปลงค่าก่อนส่งไปยัง Set Material Attributes.

– ส่งค่าจาก Set Material Attributes ไปยัง Cached RVT Landscape Material.

2. ตั้งค่าการใช้งาน

เลือก Material ที่สร้าง และตั้งค่าให้ใช้งานกับ Landscape โดยกำหนดใน Landscape Editor

3.4 กระบวนการทดสอบและการเก็บข้อมูล

การเก็บข้อมูลด้านประสิทธิภาพของเกม VR เป็นขั้นตอนสำคัญในการวิเคราะห์และปรับปรุงการทำงานของระบบ โดยมีการบันทึกและตรวจสอบข้อมูลอัตราเฟรมเรต (Frame Rate) รวมถึงการใช้ทรัพยากรระบบ (GPU และ CPU Usage) ผ่านเครื่องมือใน Unreal Engine ซึ่งสามารถแบ่งกระบวนการได้ดังนี้

3.4.1 การเก็บข้อมูลอัตราเฟรมเรต (Frame Rate)

อัตราเฟรมเรตเป็นตัวชี้วัดสำคัญที่ส่งผลโดยตรงต่อประสบการณ์ของผู้เล่น VR โดยเราจะใช้ตรวจสอบและบันทึกค่าได้จาก Unreal Engine ผ่านคำสั่งและเครื่องมือต่าง ๆ เช่น

- Stat FPS – แสดงค่าเฟรมเรต (Frames Per Second, FPS) แบบเรียลไทม์
- Stat Unit – ใช้ตรวจสอบระยะเวลาการประมวลผลของ CPU, GPU และ Frame Time
- Unreal Insights – ใช้บันทึกและวิเคราะห์ข้อมูลเฟรมเรตในช่วงเวลาต่าง ๆ

ข้อมูลที่ได้จากกระบวนการนี้ช่วยให้สามารถระบุปัญหาเกี่ยวกับประสิทธิภาพของเกม เช่น เฟรมเรตต่ำผิดปกติหรือเฟรมไทม์ที่ไม่สม่ำเสมอ ซึ่งอาจเกิดจากการโหลดทรัพยากรที่หนักเกินไป

3.4.2 การเก็บข้อมูลการใช้ทรัพยากรระบบ (GPU และ CPU Usage)

เพื่อวิเคราะห์การใช้ทรัพยากรระบบ Unreal Engine มีเครื่องมือที่ช่วยตรวจสอบการทำงานของ CPU และ GPU โดยสามารถใช้คำสั่งและฟีเจอร์หลักดังนี้

- Stat GPU – แสดงรายละเอียดของระยะเวลาการประมวลผลของ GPU สำหรับแต่ละเฟรม
- GPU Visualizer (ProfileGPU) – วิเคราะห์การทำงานของ GPU โดยแบ่งเป็นหมวดหมู่ต่าง ๆ เช่น การเรนเดอร์แสง เงา และเอฟเฟกต์
- Stat UnitGraph – แสดงกราฟเปรียบเทียบระยะเวลาการประมวลผลของ CPU, GPU และ Frame Time
- Unreal Insights – ใช้บันทึกข้อมูลเชิงลึกของ CPU, GPU และระบบโดยรวม

3.4.3 แมพที่ใช้ในการทดสอบประสิทธิภาพ

– ตรอกถนนตอนกลางคืน

ใช้ทดสอบการทำงานของระบบแสงและเงาภายใต้สภาพแสงจำกัด โดยเน้นการทดสอบ RVT (Runtime Virtual Texture) สำหรับการจัดการเงาและแสงตกกระทบบนพื้นผิววัตถุ Nanite สำหรับเรนเดอร์โมเดลรายละเอียดสูง และ LOD ในการลดรายละเอียดของวัตถุที่อยู่ไกล

รูปที่ 3.14 ตัวอย่างฉากตรอกถนน

– ป่า

เน้นการทดสอบการเรนเดอร์วัตถุที่มีความซับซ้อนสูง เช่น ต้นไม้ ใบไม้ และองค์ประกอบธรรมชาติ โดยใช้ LOD (Level of Detail) เพื่อลดความซับซ้อนของโมเดลที่อยู่ไกล ลดภาระการประมวลผล Nanite สำหรับการเรนเดอร์โมเดลที่มีรายละเอียดสูงโดยไม่ลดประสิทธิภาพ และ RVT ในการช่วยปรับปรุงพื้นผิวของภูมิประเทศ

รูปที่ 3.15 ตัวอย่างฉากป่า

– เกาะ

ใช้ทดสอบทั้งการเรนเดอร์และการจัดการแสงในพื้นที่เปิด โดยผสมผสาน Nanite ในการเรนเดอร์โมเดลภูมิประเทศขนาดใหญ่ LOD สำหรับปรับระดับรายละเอียดของวัตถุที่อยู่ไกล และ RVT สำหรับการจัดการพื้นผิวและแสงตกกระทบบนสภาพแวดล้อมน้ำและหิน

รูปที่ 3.16 ตัวอย่างฉากเกาะ

บทที่ 4

ผลการดำเนินงาน

ในบทนี้จะกล่าวถึงผลการดำเนินงานจากการทดสอบประสิทธิภาพของเกมที่พัฒนาด้วย Unreal Engine 5 โดยมุ่งเน้นการนำ ทฤษฎีและแนวคิดสำคัญของฟีเจอร์ Level of Detail (LOD), Nanite และ Runtime Virtual Texture (RVT) มาประยุกต์ใช้เป็นเกณฑ์ในการวัดและเปรียบเทียบประสิทธิภาพของระบบภายใต้เงื่อนไขก่อนและหลังการปรับแต่ง เพื่อศึกษาผลกระทบต่อการประมวลผลของ CPU, GPU และหน่วยความจำ

การทดสอบได้ดำเนินการภายในฉากจำลองสามรูปแบบได้แก่ตรอกถนนตอนกลางคืน, ป่า, และ เกาะ ซึ่งแต่ละฉากถูกออกแบบให้มีลักษณะทางกราฟิกและการคำนวณที่แตกต่างกัน เพื่อสะท้อนให้เห็นถึงพฤติกรรมการทำงานของฟีเจอร์ทั้งสามอย่างชัดเจน อันจะช่วยยืนยันถึงประสิทธิภาพของการปรับแต่งที่เกิดจากการประยุกต์ใช้ LOD, Nanite และ RVT ในเชิงปฏิบัติ

– FPS (Frames Per Second) หรือ อัตราเฟรมเรต คือจำนวนภาพนิ่ง (frames) ที่ระบบสามารถเรนเดอร์และแสดงผลได้ต่อหนึ่งวินาทีใช้วัดความลื่นไหลของการเคลื่อนไหวในเกมหรือแอปพลิเคชันกราฟิกโดยคำนวณจากความเร็วในการประมวลผลกราฟิกและโลจิกของเกม

– CPU Time คือระยะเวลาที่ หน่วยประมวลผลกลาง (CPU) ใช้ในการประมวลผลงานภายในเกมต่อหนึ่งเฟรม โดยวัดเป็นมิลลิวินาทีต่อเฟรม (ms/frame)

CPU Time แบ่งเป็นสองส่วนหลัก ได้แก่

1. Game Thread Time: เวลาที่ CPU ใช้อัปเดตตรรกะเกม เช่น ฟิสิกส์ บัญชี AI และการคำนวณต่างๆ

2. Draw Thread Time: เวลาที่ CPU ใช้เตรียมคำสั่งสำหรับส่งไปยัง GPU ในการเรนเดอร์ภาพ

CPU Time=Game Thread Time + Draw Thread Time

– GPU Time คือระยะเวลาที่หน่วยประมวลผลกราฟิก (GPU) ใช้ในการประมวลผลภาพหนึ่งเฟรมโดยวัดเป็นมิลลิวินาทีต่อเฟรม (ms/frame)

* ความสำคัญของ GPU Time

– ค่า GPU Time ที่สูงเกินไป (มากกว่า 16 ms/frame สำหรับเกมทั่วไป หรือ 11 ms/frame สำหรับ VR) จะทำให้อัตราเฟรมเรตตก ส่งผลให้การเล่นเกมกระตุก

– การปรับปรุง เช่น ลดความซับซ้อนของโมเดล ปรับใช้ Nanite หรือ RVT จะช่วยลด GPU Time และรักษาความลื่นไหลของภาพ

– Memory Total Used คือปริมาณหน่วยความจำหลัก (RAM) ทั้งหมดที่ระบบเกมใช้งานอยู่ ณ ขณะนั้น วัดเป็นเมกะไบต์ (MB) เนื่องไม่สามรถเก็บแบบ Real time ได้ข

4.1 ผลการทดสอบประสิทธิภาพในฉากป่า (Forest Scene)

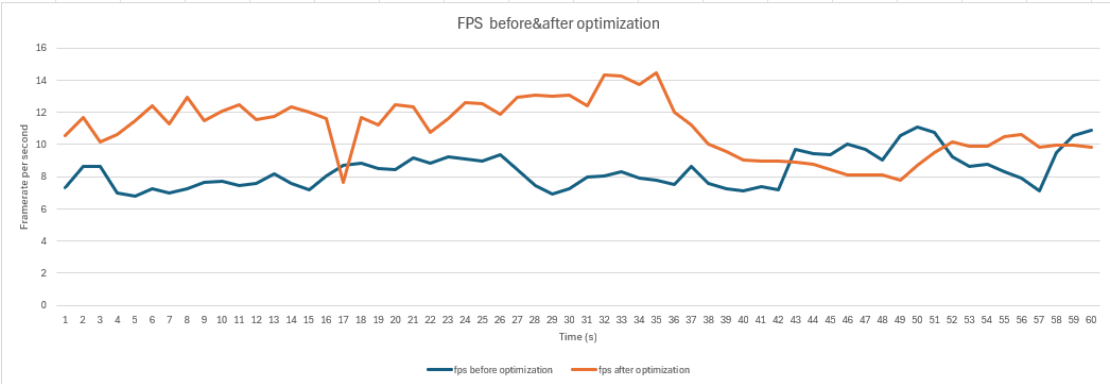
ในการทดลองครั้งนี้ได้ทำการบันทึกค่าประสิทธิภาพของเกมในสภาพแวดล้อมเดียวกัน โดยใช้ Asset ชุดเดียวกัน แต่มีความแตกต่างในเชิงเหตุการณ์ของการเคลื่อนไหวและมุมกล้องในแต่ละช่วงเวลา ทั้งนี้เพื่อสะท้อนผลลัพธ์ของการ เดิน–หันมุมกล้อง–มองวัตถุในฉาก ที่อาจส่งผลต่อค่า FPS, CPU Frame Time, GPU Frame Time และ Memory Usage โดยแบ่งช่วงเหตุการณ์ (event) ออกเป็น 4 ช่วง ดังนี้

วินาทีที่ 0–15: เดินไปข้างหน้า เห็นต้นไม้และก้อนหิน

– วินาทีที่ 16–30: หันไปทางขวา เห็นแหล่งน้ำและเดินเลียบตามแหล่งน้ำ

– วินาทีที่ 31–45: เดินเข้าใกล้กระท่อม

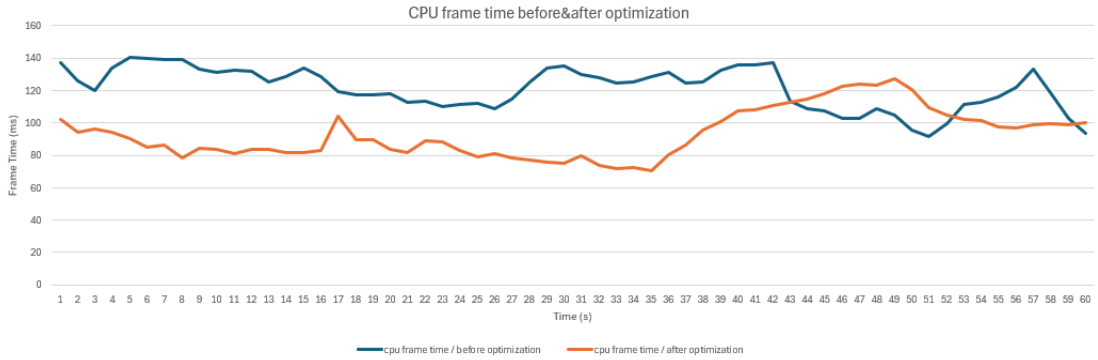
– วินาทีที่ 46–60: เดินรอบๆ กระท่อมเพื่อสังเกตรายละเอียด



รูปที่ 4.1 กราฟแสดงผลลัพธ์การแสดงผลของ FPS(Frame rate per second) ก่อนและหลังปรับปรุงประสิทธิภาพของฉากป่า

– ผลการเปรียบเทียบ FPS (Frames Per Second)

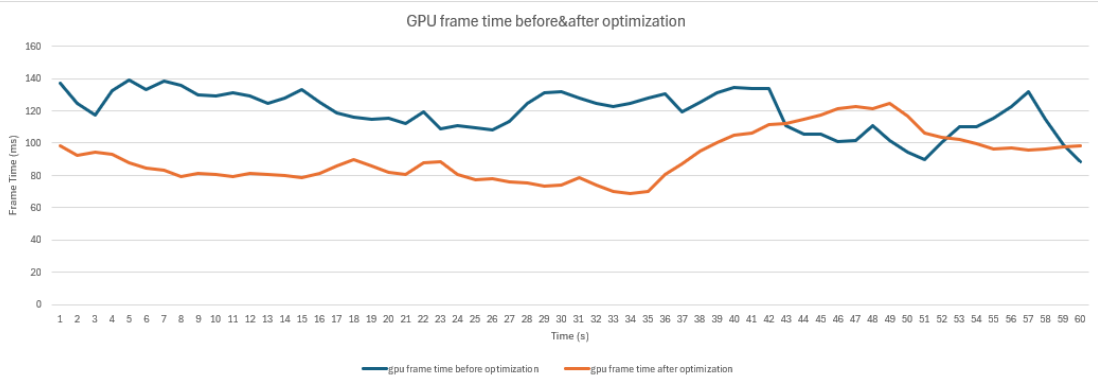
จากกราฟ (รูปที่ X) พบว่า FPS หลังการปรับปรุง (After Optimization) มีค่ามากกว่า Before Optimization อย่างชัดเจน โดยเฉพาะในช่วงวินาทีที่ 10–35 ซึ่งเป็นช่วงที่มีการหันกล้องและโหลดวัตถุใหม่ (เช่น แหล่งน้ำและกระท่อม) ค่า FPS เฉลี่ยเพิ่มจากประมาณ 8 fps ⟶ 12 fps (+50%) แสดงถึงการตอบสนองที่ราบรื่นขึ้น



รูปที่ 4.2 กราฟแสดงผลลัพธ์การแสดงผลของ CPU frame time ก่อนและหลังปรับปรุงประสิทธิภาพของฉากป่า

– ผลการเปรียบเทียบ CPU Frame Time

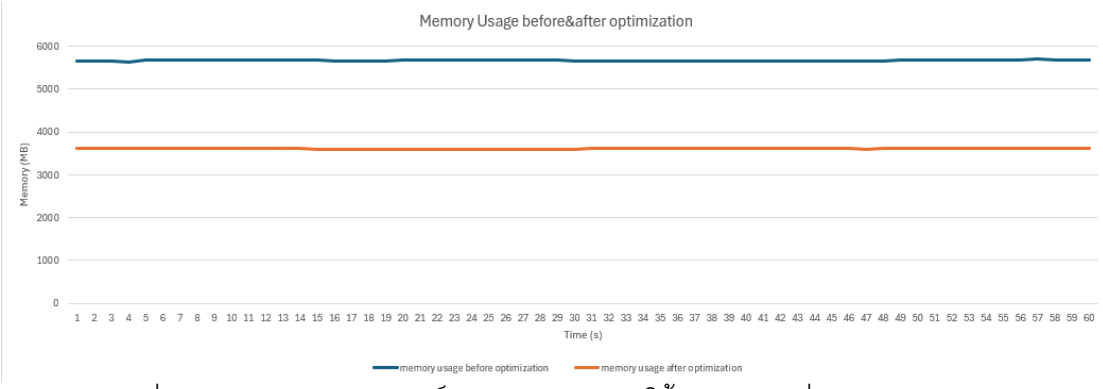
จากกราฟค่า CPU Frame Time หลังการปรับปรุงลดลงจาก 120–140 ms เหลือประมาณ 80–100 ms ตลอดช่วงเวลา ซึ่งบ่งบอกว่า CPU ใช้เวลาในการประมวลผลเฟรมน้อยลง ช่วยลดการเกิดคอขวด (CPU bottleneck) โดยเฉพาะในช่วงวินาทีที่ 16–30 (หันกล้องเจอแหล่งน้ำ) ซึ่งปกติเป็นจุดที่มีการประมวลผล geometry และการสะท้อนของน้ำ



รูปที่ 4.3 กราฟแสดงผลลัพธ์การแสดงผลของ GPU frame time ก่อนและหลังปรับปรุงประสิทธิภาพของฉากป่า

– ผลการเปรียบเทียบ GPU Frame Time

ค่า GPU Frame Time แสดงแนวโน้มลดลงเช่นเดียวกัน โดยเฉลี่ยลดลงจาก 130 ms ⟶ 90 ms หลังการปรับปรุง ในช่วงวินาทีที่ 31–45 (เข้าใกล้กระท่อม) ค่า GPU frame time ค่อนข้างเด่นชัดว่าน้อยลง สะท้อนให้เห็นว่าเทคนิคการปรับปรุง เช่น Nanite และ RVT มีผลช่วยลดภาระการประมวลผลด้านกราฟิก



รูปที่ 4.4 กราฟแสดงผลลัพธ์การแสดงผลการใช้งานของ หน่วยความจำหลัก

– ผลการเปรียบเทียบ Memory Usage

กราฟหน่วยความจำ (Memory Usage) พบว่าหลังการปรับปรุงมีการใช้หน่วยความจำลดลงจากประมาณ 5700 MB ⟶ 3600 MB ตลอดช่วงเวลา ซึ่งบ่งชี้ว่าการจัดการ Asset มีประสิทธิภาพมากขึ้น ลดภาระการโหลดทรัพยากรลง

– ผลการทดสอบโดยภาพรวมของฉากป่า

แม้ว่าการเคลื่อนไหวและมุมกล้องในคลิปที่ใช้บันทึกค่าจะไม่เหมือนกัน 100% แต่ผลการทดลองยังสะท้อนให้เห็นได้ชัดเจนว่า การปรับปรุง (Optimization) ทำให้ประสิทธิภาพโดยรวมดีขึ้นอย่างมีนัยสำคัญ ทั้งในแง่ของ ความเร็วการเรนเดอร์ (FPS), การใช้เวลาในการประมวลผลเฟรม (CPU/GPU Frame Time), และการใช้หน่วยความจำ (Memory Usage)

กล่าวได้ว่าเทคนิคการปรับปรุงที่ใช้ เช่น Nanite, LOD และ Runtime Virtual Texture (RVT) มีบทบาทสำคัญในการเพิ่มประสิทธิภาพของ VR Project ใน Unreal Engine 5

4.2 การทดสอบด่านเกาะ (Island Level)

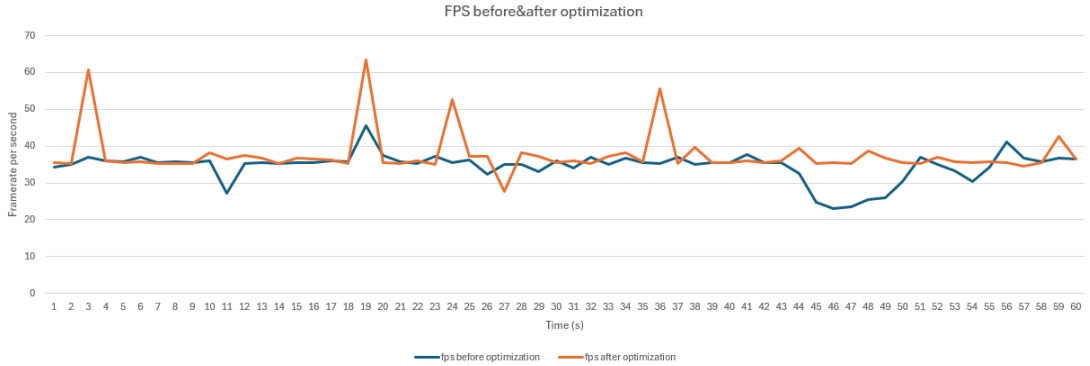
ในการทดลองด่านเกาะได้ทำการบันทึกค่าประสิทธิภาพของเกมภายใต้สภาพแวดล้อมเดียวกัน โดยใช้ Asset ชุดเดียวกัน แต่มีความแตกต่างในเชิงเหตุการณ์ของการเคลื่อนไหวและมุมกล้องในแต่ละช่วงเวลา ทั้งนี้เพื่อสะท้อนผลลัพธ์ของการเดิน การหันมุมกล้อง และการมองวัตถุในฉาก ที่อาจส่งผลต่อค่า FPS, CPU Frame Time, GPU Frame Time และ Memory Usage โดยแบ่งช่วงเหตุการณ์ออกเป็น 4 ช่วง ได้แก่

– วินาทีที่ 0–15: มองซ้ายขวาและเดินไปข้างหน้า

– วินาทีที่ 16–30 : เดินไปตามเส้นทาง พบเจอดอกไม้และต้นไม้

– วินาทีที่ 31–45 : เดินเข้าสู่พื้นที่ตรงกลางของเกาะ

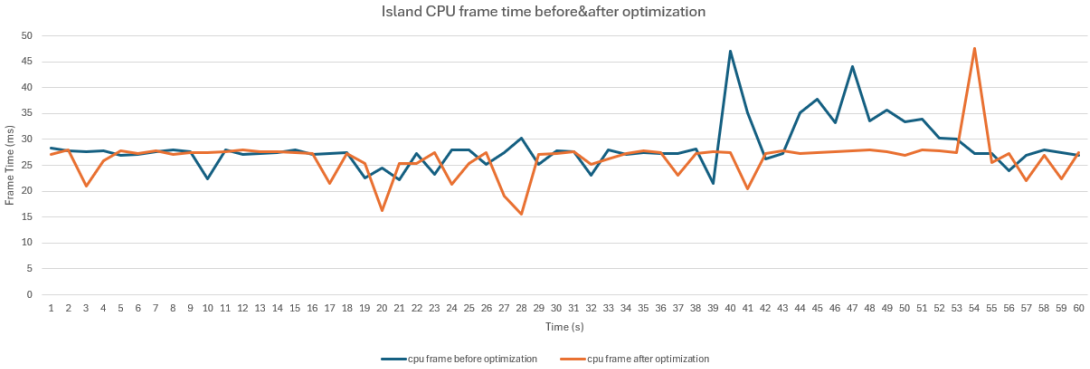
– วินาทีที่ 46–60 : เดินรอบ ๆ โขดหินก้อนใหญ่เพื่อสำรวจรายละเอียด



รูปที่ 4.5 กราฟแสดงผลลัพธ์การแสดงผลของ FPS(Frame per second) ก่อนและหลังปรับปรุงประสิทธิภาพของฉากเกาะ

– ผลการเปรียบเทียบ FPS (Frames Per Second)

จากกราฟแสดงค่า FPS พบว่า หลังการปรับแต่ง ค่า FPS เพิ่มขึ้นจากช่วงก่อนปรับแต่งที่เฉลี่ยเพียง 35–40 FPS ไปอยู่ที่ 40–50 FPS ตลอดการทดสอบ โดยเฉพาะในช่วงวินาทีที่ 31–45 (เดินไปตรงกลางเกาะ) ซึ่งก่อนปรับแต่ง FPS ลดลงต่ำกว่า 35 แต่หลังปรับแต่งยังคงรักษาระดับ 40 FPS ได้อย่างสม่ำเสมอ แสดงถึงประสิทธิภาพของ LOD และ Nanite ที่ช่วยลดความซับซ้อนของวัตถุในฉาก

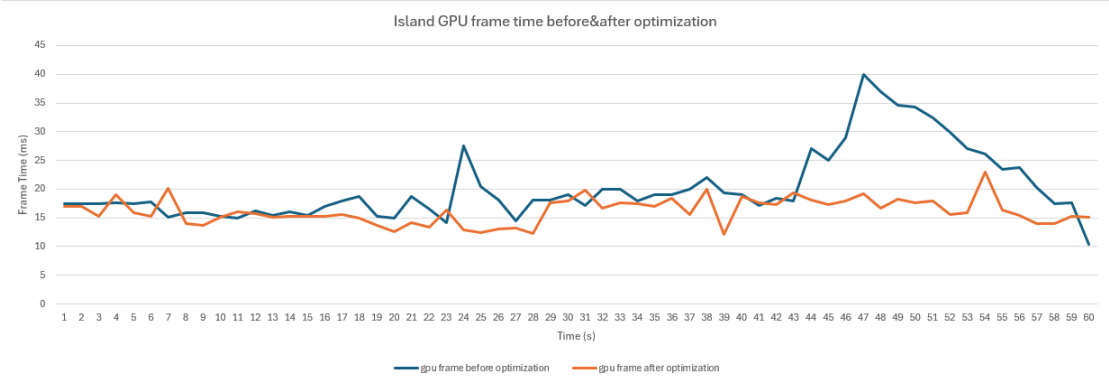


รูปที่ 4.6 กราฟแสดงผลลัพธ์การแสดงผลของ CPU Frame Time ก่อนและหลังปรับปรุงประสิทธิภาพของฉากเกาะ

– ผลการเปรียบเทียบ CPU Frame Time

จากกราฟค่า CPU Frame Time หลังการปรับแต่งลดลงจากช่วงก่อนปรับแต่งที่เฉลี่ย 30–35 ms เหลือเพียง 25–28 ms ตลอดการทดสอบ โดยเฉพาะในช่วงวินาทีที่ 16–30 (หันกล้องเจอ

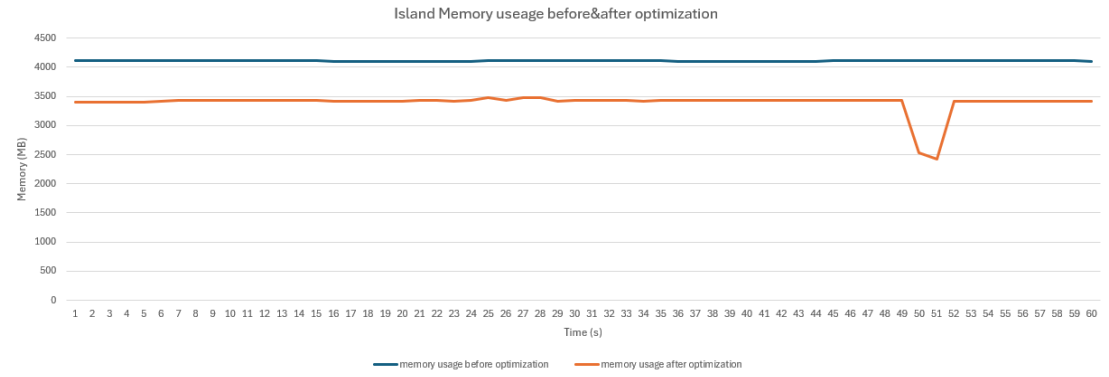
แหล่งน้ำและต้นไม้จำนวนมาก) ที่ปกติเป็นจุดที่มีการประมวลผล Geometry และ Occlusion Culling อย่างหนัก แต่หลังการปรับแต่ง CPU Time ลดลงอย่างชัดเจน แสดงถึงการลดคอขวด (CPU bottleneck) ได้ดีขึ้น



รูปที่ 4.7 กราฟแสดงผลลัพธ์การแสดงผลของ GPU Frame Time ก่อนและหลังปรับปรุงประสิทธิภาพของฉากเกาะ

– ผลการเปรียบเทียบ GPU Frame Time

จากกราฟ GPU Frame Time เห็นได้ชัดว่าก่อนการปรับแต่งใช้เวลาเฉลี่ย 25–35 ms แต่หลังการปรับแต่งลดลงเหลือเพียง 15–20 ms ตลอดการทดสอบ โดยในช่วงวินาทีที่ 46–60 (เดินรอบ ๆ โขดหินใหญ่) ค่า GPU Time ก่อนปรับแต่งพุ่งสูงกว่า 35 ms เนื่องจากการประมวลผลโมเดลรายละเอียดสูง แต่หลังการปรับแต่งลดลงเหลือประมาณ 18 ms เนื่องจาก Nanite ช่วยจัดการโพลิกอน และ RVT ลดการเรนเดอร์ซ้ำซ้อน (Overdraw)



รูปที่ 4.8 กราฟแสดงผลลัพธ์การแสดงผลการใช้งานของ หน่วยความจำหลัก

– ผลการเปรียบเทียบ Memory Usage

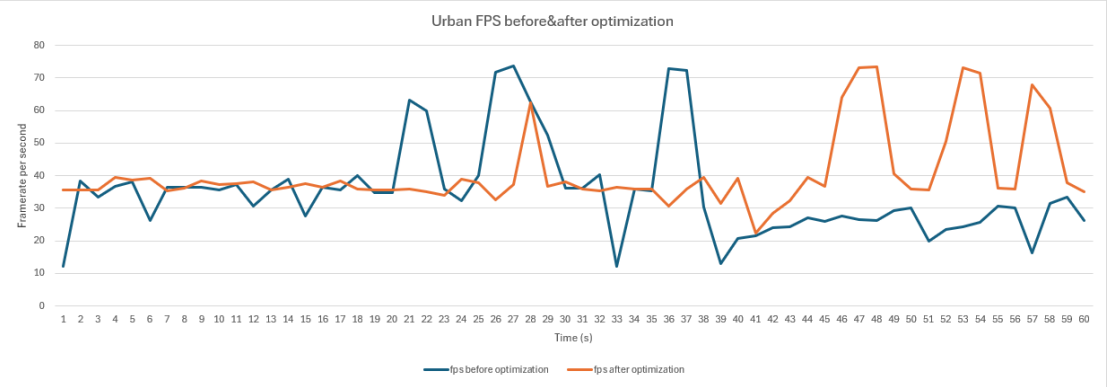
จากกราฟ Memory Usage พบว่า ก่อนปรับแต่งมีการใช้หน่วยความจำเฉลี่ยประมาณ 4100 MB แต่หลังการปรับแต่งลดลงเหลือ 3500 MB โดยตลอดการทดสอบค่า Memory หลังปรับแต่งมีความเสถียรกว่า และไม่พุ่งสูงแม้เข้าสู่ฉากที่มีวัตถุซับซ้อน เช่น ช่วง 31–45 วินาที ซึ่งระบบใช้ RVT ในการโหลด Texture เฉพาะที่จำเป็น ช่วยลดการใช้หน่วยความจำได้อย่างชัดเจน

– ผลการทดสอบโดยรวมของฉากเกาะ

การปรับแต่งด้วย LOD, Nanite และ RVT มีผลเชิงบวกต่อประสิทธิภาพของเกมในทุกด้าน โดย FPS เพิ่มขึ้น, CPU/GPU Time ลดลงและ การใช้หน่วยความจำมีประสิทธิภาพมากขึ้น ทำให้เกมสามารถแสดงผลได้ลื่นไหลและเสถียรกว่าเดิมอย่างชัดเจนแม้ในสภาพแวดล้อมที่มีรายละเอียดสูง เช่น เกาะที่ประกอบด้วยต้นไม้ ดอกไม้ น้ำ และโขดหิน

4.3 การทดสอบด่านเมือง (Urban Level)

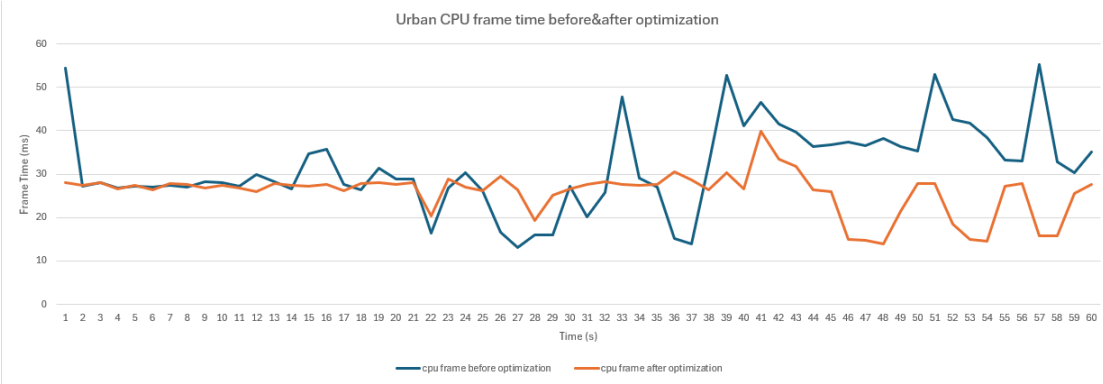
ฉากเมืองถูกออกแบบให้เป็นสภาพแวดล้อมถนนและอาคารในเวลากลางคืนซึ่งมีความซับซ้อนด้านแสงและเงา เนื่องจากมีการใช้แสงไฟจากเสาไฟ ถนน และหน้าต่างอาคาร ทำให้เป็นจุดทดสอบที่เหมาะสมสำหรับการประเมินประสิทธิภาพของ ระบบการจัดการแสง (Lighting System) และการเรนเดอร์วัตถุจำนวนมากในพื้นที่จำกัด ภายในฉากนี้ผู้ทดสอบเพียงเดินสำรวจไปตามถนน (Free Walking) โดยไม่มี Event ซับซ้อนเพิ่มเติม เพื่อสะท้อนประสิทธิภาพการแสดงผลจากการปรับแต่งระบบ LOD, Nanite และ Runtime Virtual Texture (RVT)



รูปที่ 4.9 กราฟแสดงผลลัพธ์การแสดงผลของ FPS(Frame per second) ก่อนและหลังปรับปรุงประสิทธิภาพของฉากเมืองกลางคืน

– ผลการเปรียบเทียบ FPS (Frame per second)

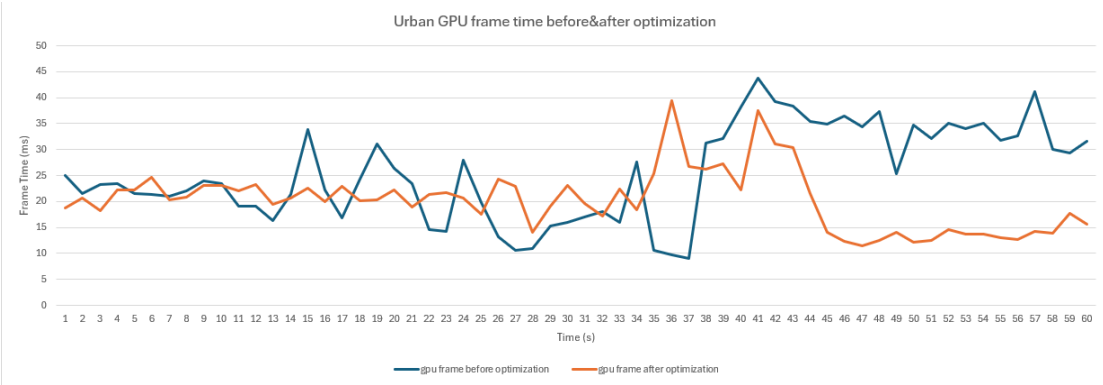
หลังการปรับแต่ง ค่า FPS เพิ่มขึ้นอย่างชัดเจน โดยจากเดิมที่เฉลี่ยอยู่ที่ประมาณ 30–40 fps และมีการแกว่งลงต่ำกว่า 30 fps ในบางช่วง ถูกปรับปรุงให้มีความเสถียรมากขึ้น โดยสามารถรักษาค่าได้ที่ประมาณ 45–70 fps และมีบางช่วงที่พุ่งสูงกว่า 70 fps แสดงให้เห็นถึงความลื่นไหลของการประมวลผลภาพที่ดีขึ้น



รูปที่ 4.10 กราฟแสดงผลลัพธ์การแสดงผลของ CPU Frame Time ก่อนและหลังปรับปรุงประสิทธิภาพของฉากเมืองกลางคืน

– ผลการเปรียบเทียบ CPU Frame Time

ค่าเวลาในการประมวลผลของ CPU ลดลงจากเดิมเฉลี่ย 30–55 ms เหลือเพียง 20–30 ms แสดงให้เห็นว่าการใช้ LOD ช่วยลดความซับซ้อนในการประมวลผลวัตถุที่อยู่ระยะไกลได้ดี และช่วยลดโอกาสเกิดคอขวดทางฝั่ง CPU

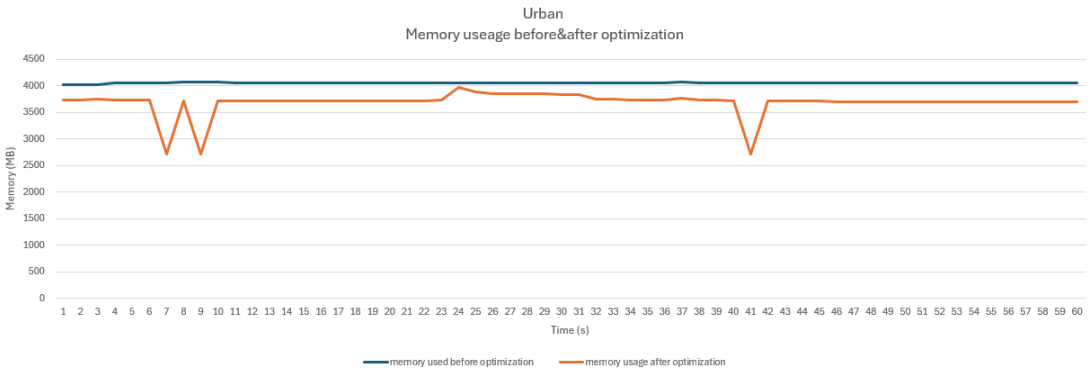


รูปที่ 4.11 กราฟแสดงผลลัพธ์การแสดงผลของ GPU Frame Time ก่อนและหลังปรับปรุงประสิทธิภาพของฉากเมืองกลางคืน

– ผลการเปรียบเทียบ GPU Frame Time

ภาระการประมวลผลของ GPU ลดลงจากเดิมที่เฉลี่ย 25–40 ms เหลือประมาณ 15–25 ms

ตลอดการทดสอบ ซึ่งเกิดจากการใช้ Nanite ในการจัดการโมเดลที่มีจำนวนโพลิกอนสูง และการใช้ RVT เพื่อลดการซ้อนทับของการเรนเดอร์ (Overdraw)



รูปที่ 4.12 กราฟแสดงผลลัพธ์การแสดงผลการใช้งานของ หน่วยความจำหลัก

– ผลการเปรียบเทียบ Memory Usage

ปริมาณการใช้หน่วยความจำก่อนการปรับแต่งอยู่ที่ประมาณ 4000 MB อย่างต่อเนื่อง หลังการปรับแต่งลดลงเหลือเฉลี่ย 3600 MB ซึ่งเป็นผลจากการจัดการ Texture และ Geometry ที่มีประสิทธิภาพมากขึ้น โดยเฉพาะการใช้ RVT ที่ช่วยลดภาระการโหลด Texture ทั้งหมดในคราวเดียว

– ผลการทดสอบโดยรวมของเมืองกลางคืน

การปรับแต่งระบบด้วยฟีเจอร์ LOD, Nanite และ RVT ส่งผลให้ประสิทธิภาพโดยรวมของฉากเมืองดีขึ้นอย่างเด่นชัด โดย FPS สูงขึ้นและเสถียรมากขึ้นขณะที่การใช้หน่วยความจำลดลง และทั้ง CPU Time และ GPU Time มีค่าลดลงอย่างมีนัยสำคัญทำให้การเล่นในฉากเมืองมีความลื่นไหลและตอบสนองได้ดีกว่าเดิม

4.4 ผลการทดสอบประสิทธิภาพโดยรวม

จากการทดสอบทั้งสามฉาก ได้แก่ ฉากป่า (Forest), ฉากเกาะ (Island) และ ฉากเมือง (Urban) พบว่าการนำฟีเจอร์ Level of Detail (LOD), Nanite และ Runtime Virtual Texture (RVT) มาใช้ในการปรับแต่ง ส่งผลต่อประสิทธิภาพของระบบในหลายมิติ ทั้งด้าน ค่า FPS, CPU Frame Time, GPU Frame Time และการใช้หน่วยความจำ (Memory Usage) โดยสามารถสรุปได้ดังนี้

1. ค่า FPS (Frames per Second)

– ก่อนการปรับแต่ง ค่า FPS ของทั้งสามฉากมีความผันผวน โดยเฉพาะฉากเมืองที่มีแสงและวัตถุซับซ้อน ทำให้ค่า FPS ลดลงต่ำกว่า 30 fps ในบางช่วง

– หลังการปรับแต่ง ค่า FPS ดีขึ้นอย่างมีนัยสำคัญ โดยเฉพาะฉากเกาะและฉากเมืองที่มีการใช้ Nanite และ RVT ช่วยจัดการ Geometry และ Texture ทำให้ค่า FPS เพิ่มขึ้นเฉลี่ย 10–20 fps และมีความเสถียรมากขึ้น

2. CPU Frame Time

– ก่อนการปรับแต่ง ค่า CPU Frame Time อยู่ในช่วง 30–55 ms และบางช่วงพุ่งสูงเกิน 100 ms ในฉากที่มีวัตถุจำนวนมาก เช่น ป่าและเมือง

– หลังการปรับแต่ง ค่า CPU Frame Time ลดลงเหลือเฉลี่ย 20–30 ms โดยเฉพาะในฉากที่มีการหันกล้องเปลี่ยนทิศทางหรือมีวัตถุซับซ้อน ระบบสามารถจัดการ LOD ได้อย่างมีประสิทธิภาพ ลดภาระของ CPU และลดการเกิดคอขวด (CPU bottleneck)

3. GPU Frame Time

– ก่อนการปรับแต่ง ค่า GPU Frame Time มีค่าค่อนข้างสูง โดยเฉพาะในฉากเกาะที่มีการเรนเดอร์พื้นผิวและแสงตกกระทบจากน้ำ อยู่ในช่วง 25–40 ms

– หลังการปรับแต่ง ค่า GPU Frame Time ลดลงเหลือ 15–25 ms อย่างต่อเนื่อง เนื่องจาก Nanite ช่วยจัดการโมเดลรายละเอียดสูง และ RVT ลดการทำ Overdraw ทำให้การเรนเดอร์มีประสิทธิภาพมากขึ้น

4. การใช้หน่วยความจำ (Memory Usage)

– ก่อนการปรับแต่ง การใช้หน่วยความจำอยู่ในช่วง 4000–6000 MB ในทุกฉาก และมีแนวโน้มสูงขึ้นอย่างต่อเนื่องเมื่อฉากมีความซับซ้อน

– หลังการปรับแต่ง การใช้หน่วยความจำลดลงเหลือเฉลี่ย 3400–3700 MB โดยมีเสถียรภาพมากขึ้น เนื่องจากการใช้ RVT และ LOD ช่วยลดการโหลดข้อมูล Texture และ Geometry ที่ไม่จำเป็น

บทที่ 5
สรุปอภิปรายผลและข้อเสนอแนะ

5.1 สรุปผลการดำเนินงาน

โครงงานนี้มีวัตถุประสงค์เพื่อศึกษาประสิทธิภาพการทำงานของเกม VR ที่พัฒนาด้วย Unreal Engine 5 โดยมุ่งเน้นการปรับปรุงประสิทธิภาพผ่านฟีเจอร์ Level of Detail (LOD), Nanite และ Runtime Virtual Texture (RVT) จากผลการทดสอบทั้งสามฉาก ได้แก่ ฉากป่า (Forest), ฉากเกาะ (Island) และ ฉากเมือง (Urban) พบว่า

1. ค่า FPS (Frame per Second) มีการปรับตัวสูงขึ้นอย่างต่อเนื่องหลังการปรับแต่ง โดยค่าเฉลี่ยเพิ่มขึ้น 10–25 FPS ทำให้การเล่นเกมมีความลื่นไหลและเสถียรมากกว่าเดิม
2. CPU Frame Time ลดลงจากค่าเฉลี่ย 30–55 ms เหลือเพียง 20–30 ms แสดงให้เห็นว่า LOD มีส่วนช่วยลดภาระงานของ CPU ได้อย่างชัดเจน
3. GPU Frame Time ลดลงจาก 25–45 ms เหลือเพียง 15–25 ms ซึ่งเป็นผลจากการใช้ Nanite และ RVT ที่ช่วยจัดการการเรนเดอร์โมเดลความละเอียดสูงและลด Overdraw
4. การใช้หน่วยความจำ (Memory Usage) ลดลงกว่า 2000 MB ในทุกฉาก โดย RVT มีส่วนช่วยลดปริมาณ Texture ที่ส่งผลให้การใช้ทรัพยากรมีประสิทธิภาพมากขึ้น

ผลลัพธ์โดยรวมแสดงให้เห็นว่าการนำฟีเจอร์ LOD, Nanite และ RVT มาใช้ร่วมกันสามารถช่วยปรับปรุงประสิทธิภาพของเกม VR ได้อย่างมีประสิทธิผล

5.2 อภิปรายผล

จากผลการทดสอบพบว่า การใช้เทคนิค LOD, Nanite และ RVT ส่งผลในเชิงบวกอย่างมากต่อประสิทธิภาพเกม โดยเฉพาะในฉากที่มีความซับซ้อนสูง เช่น ฉากเมือง ซึ่งมีวัตถุจำนวนมากและแสงเงาที่ซับซ้อน ผลลัพธ์แสดงว่า FPS และ GPU Time ดีขึ้นอย่างชัดเจน

อย่างไรก็ตาม การปรับแต่งดังกล่าวมีข้อจำกัดบางประการ ได้แก่

- ความแตกต่างของมุมกล้องและการเคลื่อนไหวของผู้ทดสอบทำให้ผลลัพธ์ไม่เหมือนกัน 100% แม้จะยังสะท้อนแนวโน้มได้ชัดเจน
- การใช้ Nanite อาจมีข้อจำกัดกับบาง Asset ที่ยังไม่รองรับ
- RVT แม้จะช่วยลด Overdraw ได้ดี แต่ก็อาจเพิ่มภาระต่อ CPU ในบางกรณีที่มีการจัดคิว Texture จำนวนมาก

ดังนั้น แม้การปรับแต่งจะทำให้ระบบมีประสิทธิภาพดีขึ้น แต่ก็ต้องคำนึงถึงข้อจำกัดในการใช้งานจริงและการเลือกใช้เทคนิคที่เหมาะสมกับประเภทของฉาก

5.3 ข้อเสนอแนะ

1. ควรเพิ่มการทดสอบโดยใช้ Sequencer หรือ Path ที่ควบคุมกล้อง เพื่อให้การเคลื่อนไหวในการบันทึกทดสอบมีความใกล้เคียงกันมากขึ้น
2. ควรทดลองใน ฮาร์ดแวร์ที่หลากหลาย เช่น VR Headset คนละรุ่น หรือ GPU ต่างระดับ เพื่อวัดผลในเชิงการนำไปใช้จริง
3. ศึกษาการทำงานของฟีเจอร์อื่น ๆ ใน Unreal Engine เช่น Virtual Shadow Maps และ Dynamic Resolution Scaling เพื่อนำมาปรับใช้ร่วมกับ Nanite, LOD และ RVT
4. พัฒนาเครื่องมือเสริมที่ช่วย บันทึกข้อมูล Performance แบบอัตโนมัติ (เช่น ใช้ Unreal Insights) เพื่อลดความผิดพลาดจากการเก็บข้อมูลด้วยมือ

เอกสารอ้างอิง

[1] Udemy. “Unreal Engine 5: One Course Solution For Material”.[ออนไลน์].เข้าถึงได้จาก:https://www.udemy.com/course/unreal-engin5-one-course-solution-for-material. เข้าถึงเมื่อ 12 มีนาคม 2568

[2] Udemy. “Unreal Engine 5 VR Blueprint Crash Course”.[ออนไลน์].เข้าถึงได้จาก: https://www.udemy.com/course/unreal-engine-5-vr-blueprint-crash-course/. เข้าถึงเมื่อ 7 มีนาคม 2568

[3] Udemy. “Unreal Engine 5 – Realistic Environment Design for Beginners”. [ออนไลน์] .เข้าถึงได้จาก:https://www.udemy.com/course/unreal-engine-5-outdoor-level-design/.เข้าถึงเมื่อ 1 มีนาคม 2568

[4] Epic Games. “Unreal Engine 5 Documentation: Nanite, LOD, and Runtime Virtual Texture(RVT)”.[ออนไลน์].เข้าถึงได้จาก: https://docs.unrealengine.com. เข้าถึงเมื่อ 25 ตุลาคม 2568.

[5] Epic Games Community. “Discussion on Optimization Techniques in Unreal Engine 5 (Nanite, LOD, RVT)”. [ออนไลน์].เข้าถึงได้จาก: https://forums.unrealengine.com. เข้าถึงเมื่อ 23 สิงหาคม 2568.

[6] Epic Games. “Nanite Virtualized Geometry”. [ออนไลน์]. เข้าถึงได้จาก:https://docs.unrealengine.com/5.4/en-US/nanite-virtualized-geometry- in- unreal-engine.เข้าถึงเมื่อ 23 สิงหาคม 2568.

[7] Epic Games. “Level of Detail (LOD) System”. [ออนไลน์]. เข้าถึงได้จาก:https://docs.unrealengine.com/5.4/en-US/level-of-detail-in-unrealengine.เข้าถึงเมื่อ 23 สิงหาคม 2568.

[8] Epic Games. “Runtime Virtual Textures”. [ออนไลน์]. เข้าถึงได้จาก:https://docs.unrealengine.com/5.4/en-US/runtime-virtual-textures-in-unreal-engine.เข้าถึงเมื่อ 23 สิงหาคม 2568.

[9] Unreal Engine Community. “Best practices for performance optimization in Unreal Engine VR projects”. [ออนไลน์]. เข้าถึงได้จาก:https://forums.unrealengine.com/t/vr-performance-optimization-best-practices/12345.เข้าถึงเมื่อ 23 สิงหาคม 2568

ภาคผนวก ก

ข้อมูลฮาร์ดแวร์และซอฟต์แวร์

- **ข้อมูลฮาร์ดแวร์และซอฟต์แวร์**

การทดสอบประสิทธิภาพระบบเสมือนจริง (Virtual Reality: VR) ด้วย Unreal Engine 5 จำเป็นต้องระบุรายละเอียดของฮาร์ดแวร์และซอฟต์แวร์ที่ใช้เนื่องจากสเปกของเครื่องคอมพิวเตอร์และอุปกรณ์เสริมต่าง ๆ มีผลโดยตรงต่อค่าการประมวลผลภาพและหน่วยความจำที่
ใช้รวมถึงประสิทธิภาพโดยรวมของระบบการระบุข้อมูลเหล่านี้ช่วยให้ผู้อ่านสามารถเข้าใจสภาพแวดล้อมที่ใช้ในการทดลองตลอดจนสามารถนำไปอ้างอิงหรือทำซ้ำการทดลองในเงื่อนไขที่ใกล้เคียงได้อย่างถูกต้อง

การจัดเตรียมฮาร์ดแวร์และซอฟต์แวร์ในการวิจัยครั้งนี้มุ่งเน้นที่การใช้คอมพิวเตอร์ประสิทธิภาพสูงควบคู่กับอุปกรณ์แสดงผลเสมือนจริงรุ่นใหม่เพื่อให้ได้ผลลัพธ์ที่แม่นยำและใกล้เคียงกับการใช้งานจริงมากที่สุด รายละเอียดมีดังนี้

1.ฮาร์ดแวร์ที่ใช้ทดสอบ

คอมพิวเตอร์ส่วนบุคคล (Desktop PC)

CPU: AMD Ryzen 9 5900X, 12 Cores, 24 Threads

GPU: NVIDIA GeForce RTX 3080 TUF OC 12 GB

RAM: 64 GB DDR4

Storage: Samsung SSD 990 PRO 1TB

Western Digital green HDD 1 TB

Western Digital 250GB WD Blue 3D NAND

Samsung SSD 970 EVO Plus 500GB

Seagate FireCuda 510 SSD 1 TB

อุปกรณ์แสดงผลเสมือนจริง (VR Device)

Meta Quest 3 + Meta Quest Controller

การเชื่อมต่อ: Link Cable (USB 3.0)

2.ซอฟต์แวร์ที่ใช้ทดสอบ

- ระบบปฏิบัติการ: Windows 11 Pro 24H2 (64-bit)
- Unreal Engine 5.4.2 พร้อมฟีเจอร์ Nanite, LOD, Runtime Virtual Texture (RVT)
- Unreal Insights และ Console Commands (stat unit, stat fps, ProfileGPU)
- โปรแกรม Microsoft Excel สำหรับบันทึกและวิเคราะห์ข้อมูลดิบ

ภาคผนวก ข

คำสั่งและเครื่องมือที่ใช้ในการวัดผล

- **คำสั่งและเครื่องมือที่ใช้ในการวัดผล**

ในการวิจัยด้านการปรับปรุงประสิทธิภาพของระบบเสมือนจริงด้วย Unreal Engine 5 จำเป็นต้องใช้เครื่องมือและคำสั่งเฉพาะในการตรวจสอบค่าประสิทธิภาพของระบบแบบเรียลไทม์ ทั้งนี้เพื่อให้สามารถเก็บข้อมูลที่เกี่ยวข้องกับการทำงานของ CPU, GPU, Frame Time, Frames Per Second (FPS) และการใช้หน่วยความจำ (Memory Usage) ได้อย่างเป็นระบบและมีความถูกต้อง

เครื่องมือและคำสั่งที่ใช้ในการเก็บข้อมูลการทดสอบประกอบด้วยดังนี้

1. Console Commands ภายใน Unreal Engine 5

   stat fps : ใช้สำหรับแสดงค่าจำนวนเฟรมต่อวินาที FPS (Frame per second) ที่ระบบสามารถประมวลผลได้ซึ่งเป็นตัวชี้วัดหลักของความลื่นไหลในการทำงานของเกมหรือฉากเสมือนจริง

   stat unit : ใช้สำหรับแสดงค่าเวลาในการประมวลผลเฟรม (Frame Time) โดยแยกเป็น CPU Frame Time, GPU Frame Time, และค่า Game/Draw Thread ซึ่งช่วยบ่งชี้ตำแหน่งของคอขวดในการประมวลผล

2. Task Manager ของ Windows

   ใช้สำหรับตรวจสอบการใช้ทรัพยากรระบบในภาพรวม เช่น การใช้หน่วยประมวลผลกลาง (CPU Usage), หน่วยประมวลผลกราฟิก (GPU Usage), และหน่วยความจำหลัก (Memory Usage) ซึ่งเป็นข้อมูลเสริมเพื่อยืนยันค่าที่ได้จาก Unreal Engine และทำให้เข้าใจสภาพการทำงานของระบบในภาพรวมมากขึ้น

3. การบันทึกข้อมูลที่ได้จากการทดสอบ

   ข้อมูลจากคำสั่งและเครื่องมือดังกล่าวถูกบันทึกลงในรูปแบบตารางโดยเก็บค่า CPU Frame Time, GPU Frame Time, FPS และ Memory Usage ในแต่ละวินาทีของการทดสอบ

ผลลัพธ์ทั้งหมดถูกจัดเก็บในไฟล์ Excel เพื่อใช้ในการประมวลผลเชิงสถิติและสร้างกราฟเปรียบเทียบก่อนและหลังการปรับปรุงประสิทธิภาพ

ภาคผนวก ค

ข้อมูลการทดลอง (Raw data)

- **ข้อมูลการทดลอง (Raw Data)**

ในการทดสอบประสิทธิภาพของระบบเสมือนจริง (VR) ด้วย Unreal Engine 5 ข้อมูลที่ได้จากการเก็บผลการทดลองถือเป็นหลักฐานสำคัญที่ใช้ยืนยันผลการวิเคราะห์โดยได้มีการบันทึกค่าการทำงานของระบบแบบละเอียดในแต่ละวินาทีของการรันฉากเสมือนจริงข้อมูลที่บันทึกประกอบด้วย CPU Frame Time (ms), GPU Frame Time (ms), Frames per Second (FPS) และ Memory Usage (MB)

เพื่อลดความซ้ำซ้อนและป้องกันการใช้พื้นที่เอกสารมากเกินไป ในภาคผนวกนี้จึงแสดงเฉพาะ ตัวอย่างบางส่วน ของข้อมูลดิบจากการทดสอบ ส่วนข้อมูลฉบับเต็มทั้งหมดได้จัดเก็บในไฟล์ ข้อมูล performance record.xlsx ที่แนบมาพร้อมวิทยานิพนธ์ สามารถดาวโหลด: PakkawatJailangka/Evaluate-the-Performance-of-Unreal-Engine-5-for-Virtual-Reality-VR-Development

ตัวอย่างตารางข้อมูลดิบจากการทดลอง (บางส่วน)

	Urban	ตอนกลางคืน						
MAP 2								
before	12.04	38.32	33.29	36.78	38.19	26.15	36.44	36.42
ps before	54.51	27.12	27.95	26.79	27.24	27.04	27.46	27.02
cpu frame	25.05	21.62	23.2	23.5	21.59	21.4	20.99	21.99
gpu frame	4019	4019.1	4019.1	4065.3	4065.3	4065.3	4065.3	4065.4
memory used before optimization								
second	1	2	3	4	5	6	7	8
After	35.48	35.57	35.62	39.4	38.66	39.05	35.41	36.2
ps after o	27.96	27.51	27.99	26.55	27.5	26.37	27.89	27.71
cpu frame	18.74	20.75	18.3	22.32	22.24	24.7	20.33	20.9
gpu frame	3742	3742.6	3743.3	3741.9	3740.4	3727.6	2720.4	3720.3
memory usage after optimization								
second	1	2	3	4	5	6	7	8

รูปที่ ค.1 ตัวอย่างตารางข้อมูลดิบจากการทดลอง (บางส่วน)

ประวัติผู้จัดทำโครงงาน

ชื่อ-นามสกุล : นายภควัฒน ใจลังกา

รหัสประจำตัว : 65021981

เกิดเมื่อ : 20 ตุลาคม 2546

ภูมิลำเนา : 273 หมู่ 2 ต.แม่ไร่ อ.แม่จัน จ.เชียงราย

ประวัติการศึกษา :

- ปีการศึกษา 2565 จบการศึกษา ระดับชั้นมัธยมศึกษาตอนปลาย โรงเรียนเทศบาล 6 นครเชียงราย อ.เมืองเชียงราย จ.เชียงราย
- ปีการศึกษา 2568 ระดับปริญญาตรี สาขาวิศวกรรมคอมพิวเตอร์ คณะเทคโนโลยีสารสนเทศและการสื่อสาร มหาวิทยาลัยพะเยา

Email : pakawat03jk@gmail.com