

計算機圖學 專題報告

海浪粒子系統模擬

學號: 310511060 姓名 : 呂和軒

學號: 310553023 姓名 : 簡嘉均

學號: 310553026 姓名 : 王家均

● Abstract

我們使用 OpenGL 模擬了一個包含些許特效的海島場景，其中包含了使用預先建好的島嶼模型、使用統計模型產生的海浪模擬以及使用了多個粒子系統去模擬火焰、雪花、龍捲風、旋風、尾流以及煙火效果，並且使用 GPU 來加速粒子系統的運算，讓場景可以順暢的顯示。

● Method

■ Ocean Simulation

我們的專題是使用 Jerry Tessendorf 的統計模型來模擬海洋的高度 (Height Field)，也就是海浪的波形，根據 Tessendorf 的論文，由一個以位置(position)和時間(time)為變數的隨機函數 $h(x, t)$ 來代表高度場，其中 x 為水平座標 (x, z) ，此函數可以寫成 sinusoids 的 summation：

$$h(x, t) = \sum_k \tilde{h}(k, t) \exp(ik \cdot x)$$

其中 t 是時間， $k = (k_x, k_z)$ 為 x 與 z 方向上的波長， $\tilde{h}(k, t)$ 為海浪高度場在時間 t 的 Fourier amplitudes。藉由 Inverse Fourier Transform，我們可以得到座標 x 在時間 t 的高度，而該座標的斜率可以寫成：

$$\epsilon(x, t) = \nabla h(x, t) = \sum_k ik \tilde{h}(k, t) \exp(ik \cdot x)$$

法向量(normal)亦可以用上述的結果計算。

$\tilde{h}(k, t)$ 是經由 Phillips spectrum 計算出來的。

$$P_k(k) = A \frac{\exp(-1/(kL))^2}{k^4} |\hat{k} \cdot \hat{\omega}|^2$$

其中 $\hat{\omega}$ 為風向， $L = V^2/g$ 由風速 V 和重力常數 g 決定。

詳細的參數細節都可以從 Tessendorf 的論文找到。

根據 Tessendorf 的演算法， k 和 x 的式子為：

$$k = \left(\frac{2\pi n}{L_x} + \frac{2\pi m}{L_z} \right)$$

$$x = \left(\frac{n_x L_x}{N} + \frac{m_x L_z}{M} \right)$$

其中 $-N/2 \leq n \leq N/2, -M/2 \leq m \leq M/2$ ，在實作中，我們是用 array 去儲存每個位置的 height，所以將公式改用 array 的 index n', m' 來表示， n, m 可以表示成：

$$n = n' - \frac{N}{2}, \quad m = m' - \frac{M}{2}$$

原本的 $h(x, t)$ 改成：

$$h(x, t) = \sum_{n'=0}^N \sum_{m'=0}^M \hat{h}(n', m', t) \exp \left(i2\pi \left(\frac{n' n_x}{N} + \frac{m' m_x}{M} \right) - i\pi(n_x + m_x) \right)$$

實作中是對 $h'(x, t)$ 做 Inverse Fourier Transform 再做 translation 算出 $h(x, t)$ ：

$$h'(x, t) = \sum_{n'=0}^N \sum_{m'=0}^M \hat{h}(n', m', t) \exp \left(i2\pi \left(\frac{n' n_x}{N} + \frac{m' m_x}{M} \right) \right)$$

所以我們會計算兩個 $N \times M$ 大小的 array，儲存著 $N \times M$ 個 vertex 的資訊，這兩個 array 分別是 normal field 和 height field，normal field 儲存的是 vertex 在 x 和 z 方向的斜率，height field 儲存的是 vertex 在 x 和 z 方向的 displacement 以及 y 方向的高度，計算的方法就是使用 FFTW 的 library 照著上述的公式計算，接著將 normal field 和 height field 傳入 shader 計算 lighting。

在 lighting 的部分，我們是使用 Blinn-Phong-shading，海的顏色是由 3 個步驟去計算：

1. Height color

從表面的波形高度決定海面的基礎顏色，把海面高度映射到 $[-1, 1]$ ，在深藍色(0.02, 0.05, 0.10)與淺綠色(0, 0.64, 0.68)之間作插值。

2. Reflect color

這部分是海洋反射天空的部分，計算的方法是：

$$reflCoeff = n \cdot v$$

$$reflectColor = skyColor \cdot (1 - reflCoeff^c)$$

n 是 normal of fragment， v 是 view direction， c 是 constant，類似於 shininess，它的值我們是設定 0.3

3. Specular color

Specular color 的算法和 Blinn-Phong-shading 一樣，只是在計算最後的顏色方式不太一樣，方式如下：

$$combinedColor = ambient + HeightColor + reflectColor$$

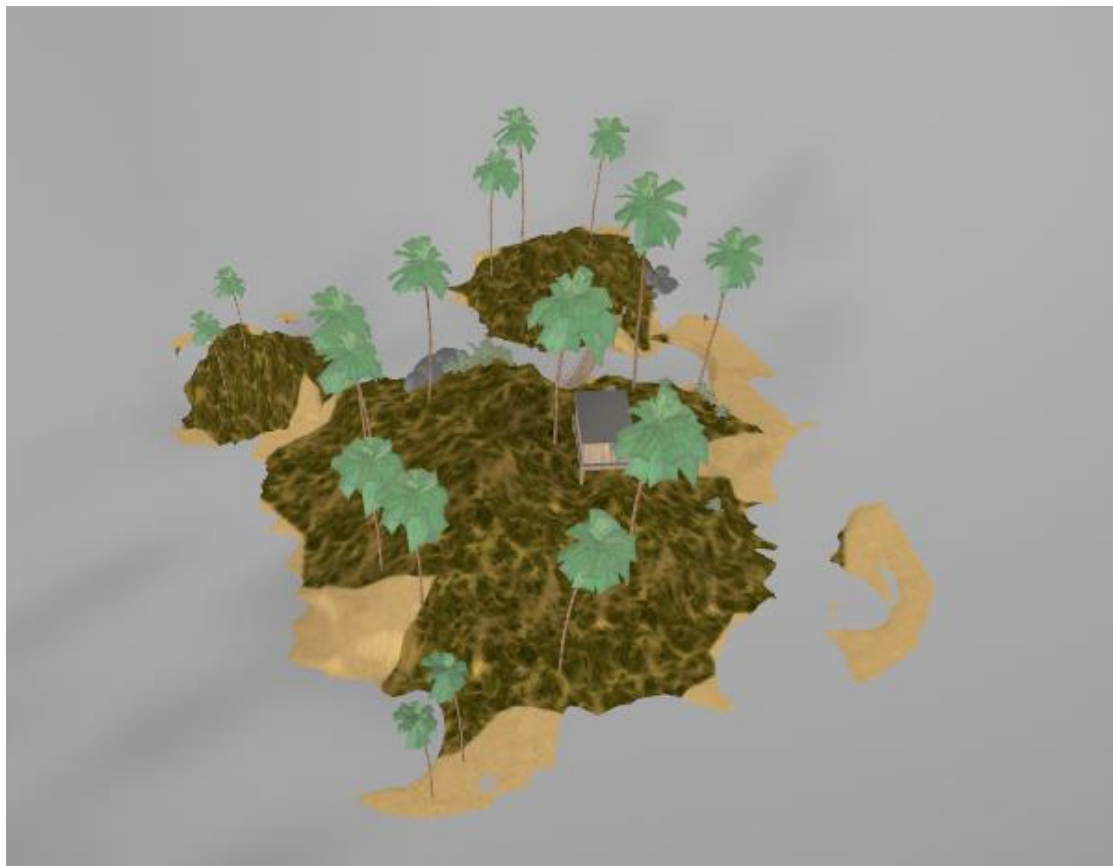
$$finalColor = (1 - specCoeff) \cdot combinedColor + SpecularColor$$

實作中需要設定的參數有 N 、 M 、 L_x 、 L_z 、 $\hat{\omega}$ 、 V 、 A 、 C 、skyColor、height color 做插值的基礎顏色。

■ Complex model

要使用複雜的模型時，首先我們需要有模型，所以我們在 FREE3D 網站先找好了一個要使用島嶼模型，島嶼模型裡包含著所不需要的平面海浪，所以先透過文字編輯器修改了海浪的四個頂點，讓其不會顯示出來；接著我們需要將模型讀入 OpenGL，所以使用了在 learnopengl.com 上有教學的 model loading library，此 library 為 assimp，全名 Open Asset Import Library，此 Library 可以幫助我們處理多種模型檔案的格式，並轉成內建的資料結構，我們就可以不用處理讀檔案的部分了；接下來要將 Assimp 的資料結構轉成 OpenGL 可以使用的格式，所以需要創建一個 model 的 class 並將需要的功能都寫好，再將所需的資料從上述的格式一一輸入進 model 裡面，將全部物件的頂點、法向量、材質、材質貼圖、面等等都處理完畢後，最後即可靠著 draw 將模型給繪製出來。

在 3D 檢視器中可見的模型(已移除海平面)如下：



■ Particle System

我們的目標是使用粒子系統模擬出火焰特效，因此我們需要建構出粒子物件來描述粒子參數，讓粒子能根據我們所設想的行為動作，並透過粒子管理物件來對所有粒子進行更新。

渲染粒子特效的方法分為許多種，一開始我們將每個粒子都做為一個獨立的 3D 物件，單一粒子透過多個頂點來描述粒子的形狀，並且由 CPU 負責管理更新，GPU 則負責渲染繪製，儘管此方法對於粒子系統的建造較為直觀簡潔，但很快地我們發現了此方法的瓶頸，由於每次更新完粒子後，都需要將新的粒子參數重新與 GPU 記憶體作交換，並且繪製一個粒子便要重新傳入參數，造成記憶體傳輸上的時間消耗，另一個問題為 CPU 並不適合大量的粒子更新運算，透過迴圈一一對粒子更新是很大的時間開銷，因此要建立起多個粒子特效必定無法使用此方法來完成，而由於粒子與粒子間具有獨立性，因此可以透過 GPU 來對粒子做平行運算，一次更新大量粒子可以大幅度的減少運算時間，因此我們決定移植整個粒子架構至 GPU，由 GPU 來做粒子管理及繪製。

為了移植粒子系統至 GPU 上運算，我們依靠 Transform feedback 以及 Geometry shader 這兩個核心技術來實現，透過 Transform feedback 我們可以將 GPU 運算完的資料輸出及保存起來，在下個時間幀使用，減少資料在記憶體方面的移動開銷，透過 Geometry shader 我們則可以在 GPU Pipeline 中生成粒子，並在繪製時描述粒子形狀，利用這兩個技術我們便可以將粒子系統從 CPU 端移植到 GPU 端運算，大幅度的改善整體運算渲染時間，保持畫面的流暢性，以下針對 Transform feedback、Geometry shader 以及實作粒子繪製流程做介紹。

Transform Feedback

簡單的概念就是可以將 GPU 計算完的輸出給抽取出來存放到 buff 中，作為下一幀的輸入使用，因此稱作為 feedback，透過這個概念我們就可以利用 shader 來做粒子參數的更新，將粒子資訊存入 transform feedback buffer，在下一幀讀取繼續做更新和繪製，就不需要透過 CPU 來重新更新粒子和重新綁定 buff 給 GPU，因此透過這個技術，可以將粒子的更新移植到 GPU 上完成，減少了更新大量粒子所耗費的時間，也減少了記憶體寫入和搬移的開銷，讓我們可以即時的在同個場景中渲染大量的粒子及不同的粒子特效。

值得提到的一點是，由於同個 buffer 不能同時作為輸入和輸出，通常會有兩個 Transform feedback buffer 來做交替使用，一個做為輸入暫存 那麼另一個則做為輸出暫存，在下一幀做交換，輪替的來做更新和繪製粒子。

Geometry Shader



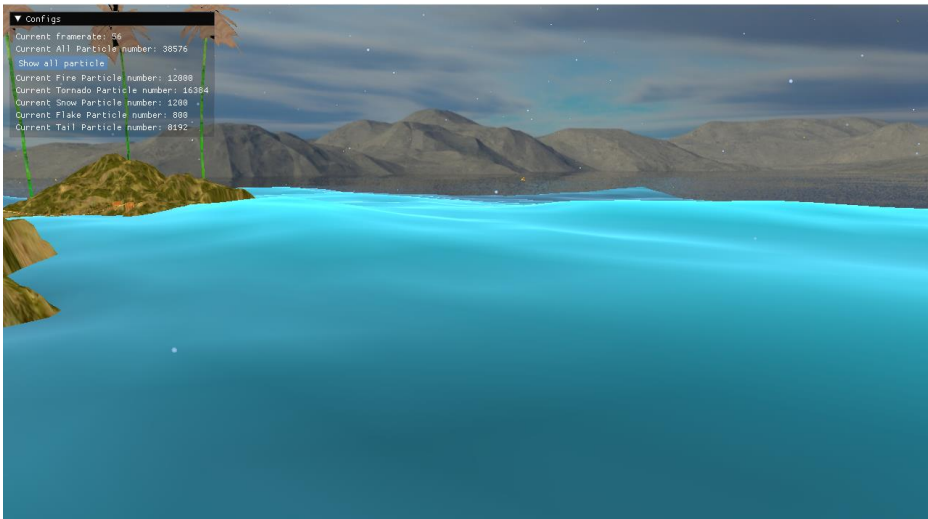
Geometry Shader 的主要功能是將輸入的頂點進行刪減或增加，藉此改變原本的幾何形狀，在粒子系統中可以用來生成粒子，我們只要輸入一個或多個基礎的粒子，就可以經過 geometry shader 生成出其他粒子點，第二個功能是在繪製粒子時可以給粒子賦予形狀，而最常使用在形狀生成是 billboard 法，他可以生成一個不管在任何角度都朝向相機視角的幾何平面，做法是將粒子位置投射到相機座標系中，由 geometry shader 生成對應而且面向相機的 4 個平面點，然後輸出到 fragment shader 繪製，以這樣的方法可以讓粒子在任何角度都顯示得出來，不需要使用到 3D 立體形狀去描述粒子，減少繪製開銷。

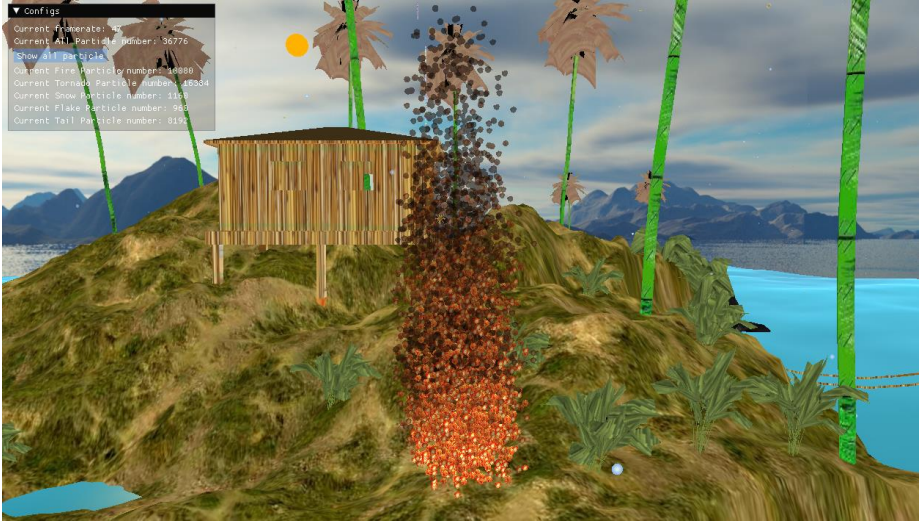
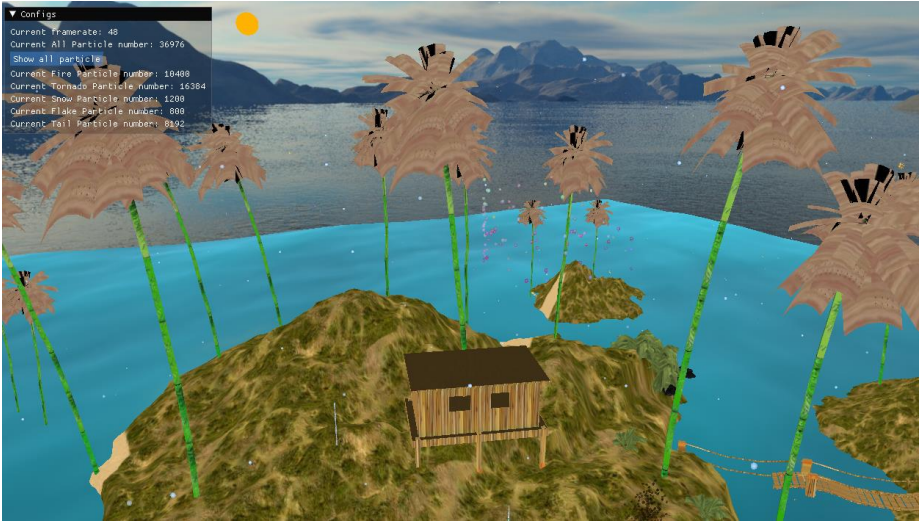
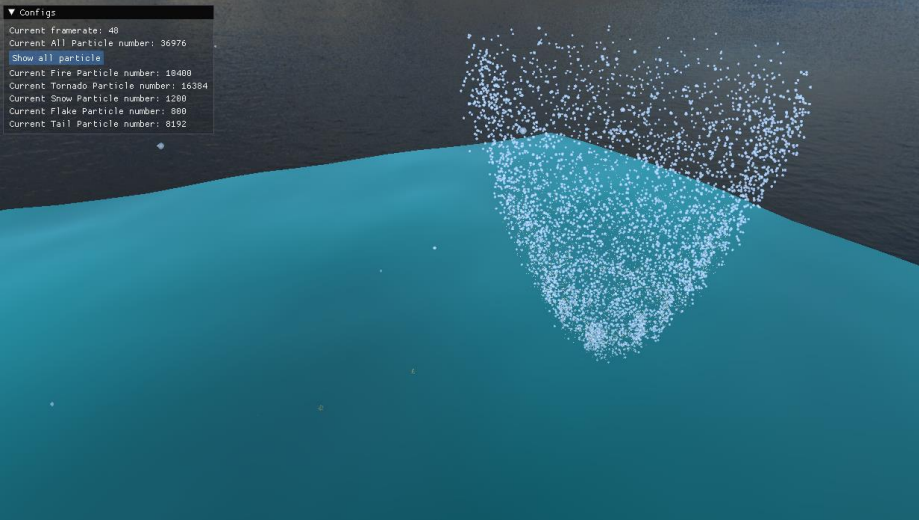
粒子系統整體流程可分為發射(emit)、更新(update)及繪製(draw)三部分，在發射步驟時，會輸入一個或多個基底粒子到發射 shader 中，再透過 geometry shdaer 去生成出更多的粒子，通常會由 1 個基本粒子生成出 80 個粒子，也可以自行控制，其中根據不同特效設定初始的粒子參數，並透過採樣隨機貼圖來加入粒子的隨機性，讓他們活動得更加自然，最後利用 Transform Feedback Buffer 將新生成的粒子資訊輸出。

在更新粒子部分，會將新的和舊的粒子一同做更新，主要的流程就是隨著時間更新他們的位置 速度 顏色及壽命等等資訊，並且透過 geometry shader 去判斷粒子壽命，如果壽命超出範圍則該粒子不輸出，若粒子還活著那就輸出到 TF buffer 中，藉此來管理粒子存亡和數量，這邊會是主要設計粒子特效的部分，根據不同的運動規則和時間變化，可以設計出不同樣式的特效。

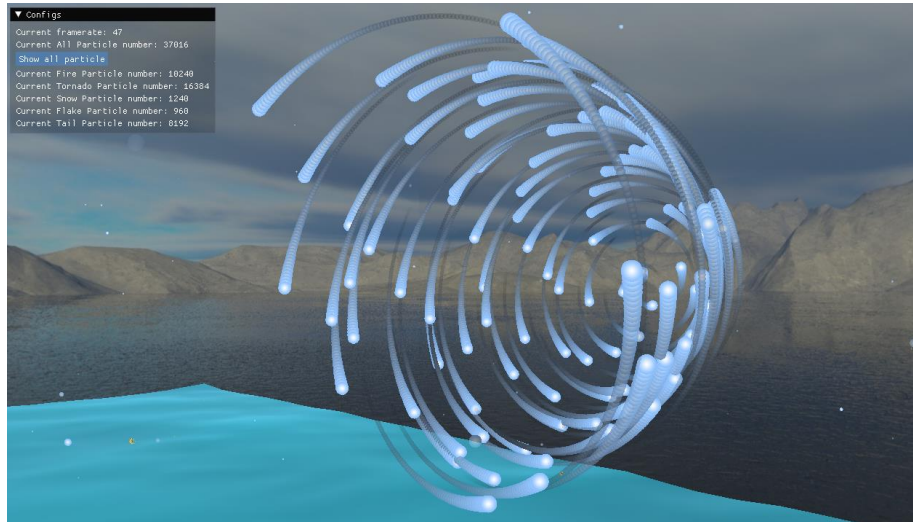
繪製部分則是將上個步驟更新完的粒子做渲染，從 TF buffer 存取粒子資訊進行繪製，到目前為止都還沒有替粒子給出實際的形狀，僅包含粒子資訊而已，因此我們需要透過 geometry shader 插入更多頂點來描述粒子形狀，較常使用的方法為 billboard，使得繪製出的粒子都朝向視角，並且控制粒子的大小變化，最後輸出到 fragment shader，去控制頂點的顏色、材質與透明度，完成粒子的繪製。

● Result

Overview	 <p>▼ Configs</p> <ul style="list-style-type: none">Current framerate: 47Current All Particle number: 38976Show all particleCurrent Fire Particle number: 19249Current Tornado Particle number: 15384Current Snow Particle number: 1298Current Flake Particle number: 968Current Tail Particle number: 8192
Island Model	 <p>▼ Configs</p> <ul style="list-style-type: none">Current framerate: 48Current All Particle number: 37056Show all particleCurrent Fire Particle number: 19328Current Tornado Particle number: 16384Current Snow Particle number: 1289Current Flake Particle number: 968Current Tail Particle number: 8192
Ocean Simulation	 <p>▼ Configs</p> <ul style="list-style-type: none">Current framerate: 56Current All Particle number: 39576Show all particleCurrent Fire Particle number: 12088Current Tornado Particle number: 16384Current Snow Particle number: 1289Current Flake Particle number: 888Current Tail Particle number: 8192

<p>Fire effect</p>	 <p>▼ Configs Current framerate: 46 Current All Particle number: 36976 Show all particles Current Fire Particle number: 10488 Current Tornado Particle number: 16384 Current Snow Particle number: 1208 Current Flake Particle number: 888 Current Tail Particle number: 8192</p>
<p>Snow and Firework</p>	 <p>▼ Configs Current framerate: 46 Current All Particle number: 36976 Show all particles Current Fire Particle number: 10488 Current Tornado Particle number: 16384 Current Snow Particle number: 1208 Current Flake Particle number: 888 Current Tail Particle number: 8192</p>
<p>Tornado</p>	 <p>▼ Configs Current framerate: 46 Current All Particle number: 36976 Show all particles Current Fire Particle number: 10488 Current Tornado Particle number: 16384 Current Snow Particle number: 1208 Current Flake Particle number: 888 Current Tail Particle number: 8192</p>

Tail effect



專題的實際成果除了提案時設立的目標如島嶼模型、海浪模擬以及模擬粒子火焰外，額外實現了全域隨機雪花、龍捲風、尾流粒子以及煙火等粒子特效，左上角可以看到當前粒子數量以及每秒幀數，在三萬多個粒子渲染的情況下，我們仍然可以達到將近每秒 50 幀的更新速度，測試環境的顯示卡型號為 RTX2070。

● Problem

■ Ocean Part

1. 在 Tessendorf 的論文中，計算 displacement 的公式為：

$$\mathbf{D}(\mathbf{x}, t) = \sum_{\mathbf{k}} -i \frac{\mathbf{k}}{k} \tilde{h}(\mathbf{k}, t) \exp(i\mathbf{k} \cdot \mathbf{x})$$

我不確定，但公式中的負號加不加似乎不會有任何影響

2. 在對 $h'(\mathbf{x}, t)$ 做 translation 算出 $h(\mathbf{x}, t)$ 時，由於 $h(\mathbf{x}, t)$ 的公式中的 $i\pi(n_x + m_x)$ 在 $n_x + m_x$ 為偶數時會把結果加一個負號，所以在 $n_x + m_x$ 為偶數時，要多加一個負號這樣 $h(\mathbf{x}, t)$ 的計算結果才會正確。
3. 由於 Fourier transform 的計算都是在 CPU 計算，當 N 、 M 太大時，FPS 沒有辦法達到 real time performance，但 N 、 M 不夠大又沒辦法模擬出真實逼真的海浪效果，所以必須把 Fourier transform 的計算移至 GPU，這部分我沒完成。

■ Model Part

1. 看過多個範例程式，只有一個有處理到 .obj 檔中關於材質的部分(.mtl)，許多程式只有單純一張材質貼圖，並不能 load 由很多部分所組成的 model。

2.範例程式中的 model 寫法關於 shader 的使用跟作業中使用 shader 的方式不一樣，所以必須統一程某一種，不然無法使用。

■ Particle Part

1. 起初使用 Transform feedback 時沒將暫存器綁定好，使得無法將 GPU 內部的粒子資料讀出來進行繪製和更新，但 Shader 端及 Visual studio 端都沒有報錯，上網來回查詢綁定對照流程也沒有問題，但就是綁定不上去，後來保險起見在多個地方額外加入 Buffer 綁定就解決了。

2. GPU 內部獲取亂數問題，通常會使用預先生成的隨機貼圖來存取亂數，但起初都使用同一個亂數貼圖，因此發現有的粒子會因為亂數不夠隨機而容易產生相同的規律性，使得粒子特效不自然，後來將亂數貼圖分別在生成更新和繪製時都重新採樣，解決亂數不夠隨機的問題。

● Conclusion

經過這次的 project，我們學習到如何使用海洋頻譜與 FFTW library 的 Inverse Fourier Transform 來計算 vertex 的 displacement vector、height field 與 normal field，藉此模擬海面的波形，以及使用 Assimp library 讀取並畫出預先建好的島嶼模型，除此之外，我們還熟悉了如何建構一套粒子系統，並利用 Transform feedback 以及 Geometry shader 進行粒子的生成、更新、管理與繪製，也學習到如何利用粒子系統去設計並展示初步的多項特效，如：火焰、雪花、龍捲風、旋風以及煙火等等，此外，我們利用 GPU 進行粒子系統的管理與繪製，能夠在 real-time frame 底下模擬上萬數量的粒子。

● Distribution of Work

- 310511060 呂和軒 50% --- 粒子系統、專題整合、粒子系統部分報告
- 310553026 王家均 25% --- 處理 model loading、部分報告
- 310553023 簡嘉均 25% --- Theory of wave simulation、海浪部分報告

● Reference

- <https://github.com/JoeyDeVries/LearnOpenGL>
- [Small Tropical Island 免费的 3D 模型 - .obj - Free3D](#)

- https://github.com/asylum2010/Asylum_Tutorials
- [Jerry Tessendorf 的 paper : Simulating Ocean Water](#)
- <https://github.com/jiasli/OceanSurface>