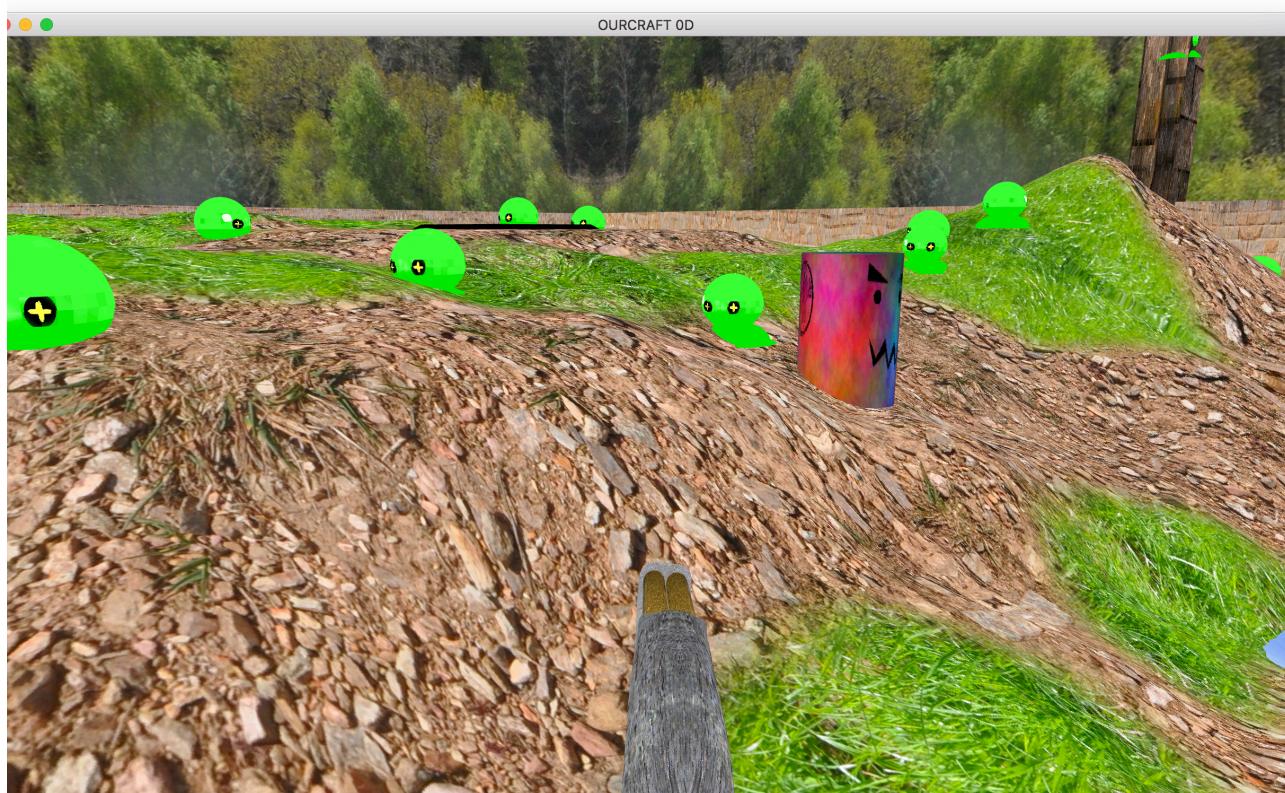


ASM Final Project 2017

group 35

104303040 張元 - 104502039 莊秉叡

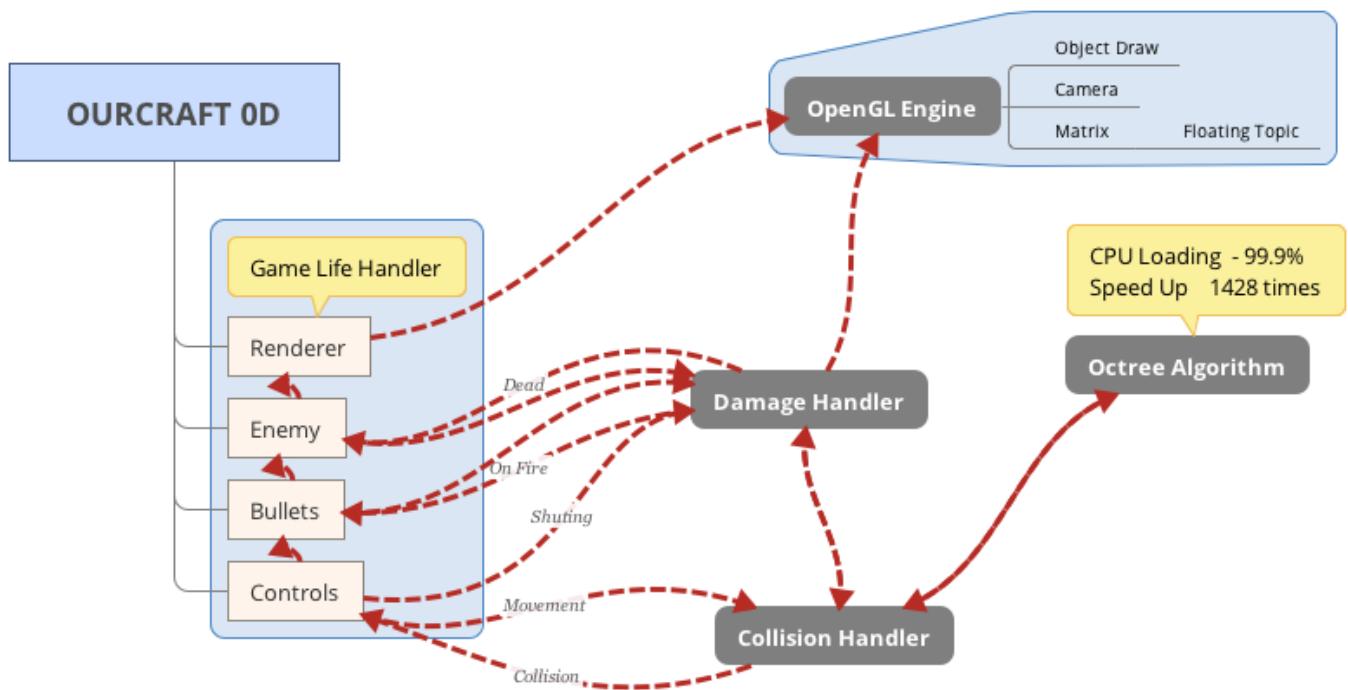


研究動機



當初構想時就打算做3D遊戲，也知道3D遊戲有許多困難的地方，即使用高階語言製作，如矩陣運算、碰撞演算法等。後來以SDL框架先做了2D的遊戲，由於迅速完成，我一樣決定做3D遊戲，溝通後我們就捨棄SDL繼續原本的目標，3D engine Library使用Open Source的OpenGL，負責處理3D顯示的部分，而遊戲主體，碰撞handler，怪物系統，子彈扣血系統，都是100%由組語打造，以及八元樹演法應用於碰撞優化，我們的環境是在macOS上，並使用NASM語法與Compiler作為我們在mac上的asm開發，NASM與MASM大同小異，幾乎可說只差在環境。

程式架構



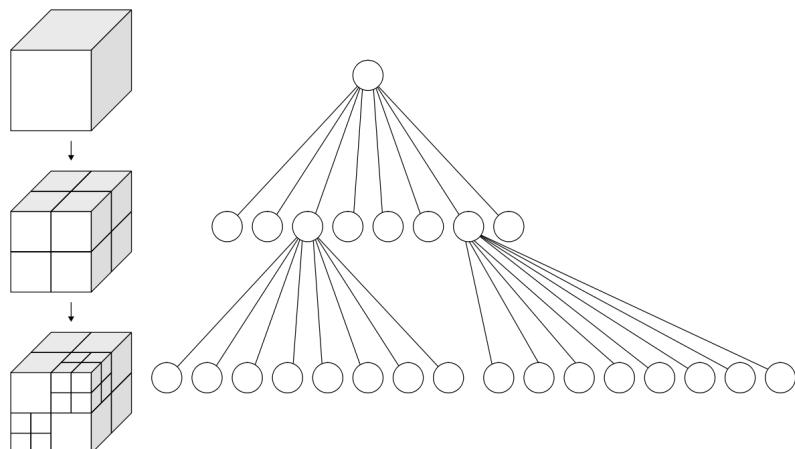
在**main.asm**這份1200行組語中，主要處理遊戲Render的動作以及遊戲生命週期，而每一次Render都會處理好每個遊戲系統環節，以下各個介紹，control input 在listen event後會將玩家三維坐標交由Collision Handler（撰寫於**legacy.asm**與**Object.asm**中），會回饋給Controls，碰撞，及無碰撞合理地形高度y值，與無碰撞也無觸地，三種回饋，碰撞邊界與無碰撞也無觸地，會導致一樣的情形即終止行動，但無碰撞也無觸地會回傳修補遺失的地形高度。

而Controls收到發射訊號，會先在自身生成子彈，on fire狀態的子彈會被Renderer處理，當有on fire狀態的子彈，會觸發Damage Handler，Damage Handler會與Enemy以及Bullets溝通（撰寫於**Enemy.asm**與**Weapon.asm**中），Damage Handler的碰撞部分會請求Collision Handler協助，判斷on fire子彈與所有enemy的碰撞，Damage Handler會依據情況處理該enemy扣血或死亡，同時處理子彈的生命週期，轉移on fire狀態，這樣大致說明了程式的運作流程。

在退出遊戲生命時，會釋放所有記憶體，包括OpenGL視窗，3D物件的載入，建的八元樹，以及跟GPU註冊的行程。

Collision Handler with OcTree Optimization

碰撞判斷，是3D遊戲的精髓之一，電腦對上億個點並無空間感，即使將點的最小單位依據物件主體設計，隨著開發，很快的頂點數量會急速上升，若用地毯式掃法，CPU幾乎會飆至99% loading，再不改變碰撞演算法的情狀下可以使用將obj拆散的機制，減少區域頂點運算量，但這純粹屬於投機取巧減少量的方式加速，成本是需一直載入釋放3D obj檔案。而OcTree是利用資料結構與基本演算法來優化，結構是簡單的八元樹，而走訪演算法，也只是基本的對三維中三的座標軸進行同時二分搜，達到八元樹的traversal，OcTree中也有很多種優化方法跟變形，我們是運用在空間中不會移動的地形物件上，於遊戲一開始會建立一個籠罩全空間的八元樹，並將所有地形節點insert build建枝，值得提醒的是此舉並不會增加使用記憶體的量，因為是將原本就會載入遊戲的obj建入OcTree，所以佔用記憶體是一模一樣的，而效能方面，倘若是 3000*3000*3000的空間，總共有270000000000，約270億個頂點空間，而12層八元樹，能在12次判斷中判斷任兩點是否存在至 $1*1*1$ 精度的空間中，效能極為驚人，而本遊戲即為此大小的空間，而我們是建5層，所以能將每次要算5000000個頂點降至小於7000個，這也是正是加速1428倍的原因。



inc

我們有寫一些include檔，主要是data或浮點數的放置，與macro的撰寫，如x64，有先寫好x64傳參的macro，方便撰寫，吃str的macro自動放置binary中當format string，這樣可以動態產生不用一直加到data section，等等，以及OpenGL的symbols。

結論

當初做3D，組員認為在開玩笑，我想藉這次機會玩玩看高階語言在機器碼的呈現，不但可以了解想了解的事物來龍去脈，也可以了解程式語言的特性，如物件導向在底層的呈現，高階語言編譯器的些許內涵，重點是好玩，組員同樣也是想玩OpenGL，我們就開工了，反正都是想自己玩的東西，還可以當一次課上報告，呈現給些許人看，都是附加價值。

