高等計算機圖學 HW4 Report &操作說明

ntu\_r06922063 邱政凱

(對於複雜的模型(bunny)，我的程式可以達成約300倍的加速)

使用教學:

可以直接執行hw4.exe。它會自動讀取同個路徑下名為input.txt的檔案當作輸入。若要自行指定檔名，可使用cmd

./hw4 “檔名”

的指令，以參數的形式把要使用的input檔名傳入。(預設input.txt是複製rabbit.txt的內容)

(老師提供的兩個input我分別命名為monkey.txt和rabbit.txt)

程式執行完會輸出一個名為"colorOutput.ppm"的ppm圖片檔，請使用infraview來檢視。

專案中有一個codeblocks的專案檔hw4.cbp，可以使用此專案檔配合code blocks來開起來編譯。

Hw4.cpp : 這次作業的程式本體

algebra3.cpp: 這次作業使用老師提供的algebra3函式庫

imageio.cpp : ppm檔案相關的讀寫操作。

並且，我有稍微更改了input的格式

在每個"單一"物件的所有triangle mesh的敘述出現之後之後加上一個字元"O"的line來代表以上的所有MESH代表一個單一物件，需要有自己的KD-TREE。這樣做的目的是為了避免bounding box差距太大的兩個物體共用同一個kd-tree，會讓kd-tree的加速效果退化非常多。

例如老師的範例場景中，動物模型再靠右側，但是地板平面幾乎橫跨畫面的x方向，如果兩者共用同一個KD-tree，會變成動物模型任何一個kd-node中只要有包含到地板平面的mesh，他的bounding box幾乎就會橫跨整個畫面的x軸，造成很容易打中某些Bounding box，但是實際上該bounding box的mesh只有兩個三角形。

我把老師的monkey和bunny模型的模型本體跟地板的plane分別弄成兩個不同的物件

要測試請直接使用資料夾中的monkey.txt和rabbit.txt，裡面為我已經加上額外兩行"O"的物件描述的版本

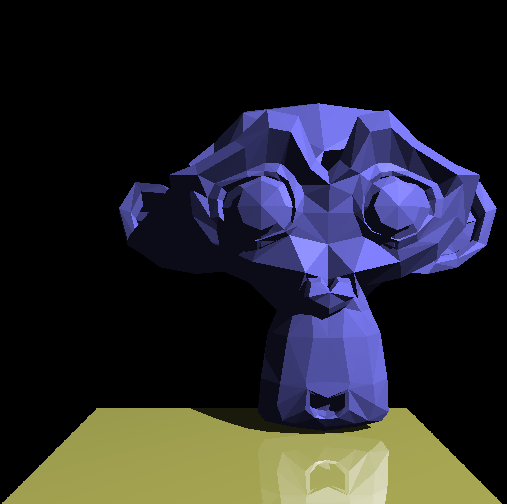
這次的渲染加速作業，我實作了KD-Tree，將場景動態依照每個物件個別化分成不同的kd-tree，然後建構自己的tree(在範例input中，底下的plane跟動物模型兩者就是不同的object，所以兩個檔案各有兩個kd-tree)，詳細步驟與心得在結果下面。

結果圖: (原尺寸，使用Left-hand rule)

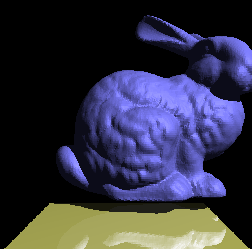
Monkey 256\*256



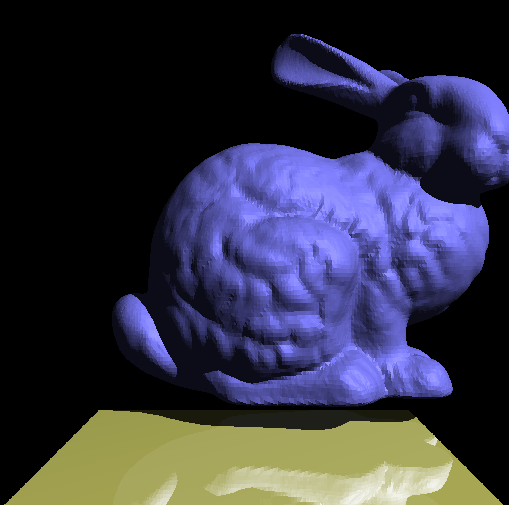
Monkey 512 \* 512



Bunny 256 \* 256



Bunny 512 \* 512



執行時間 & 相關數據： (原本的速度請參照HW3 Report，量測時間只計算render time，不包含建構kd-tree的pre processing時間)

設備: Interl core i5-7200U 2.5GHz

Monkey 256 \* 256

render time: 2.137 seconds (原本: 14.293 SEC)

total kd-tree node count: 138

max depth of kd-tree subdivision: 9

average ray-mesh Intersection check count: 133.332

Monkey 512 \* 512

render time: 7.833 seconds (原本: 73.435 SEC)

total kd-tree node count: 138

max depth of kd-tree subdivision: 9

average ray-mesh Intersection check count: 133.237

Bunny 256\*256

render time: 5.576 seconds (原本: 1627.135 SEC)

total kd-tree node count: 18094

max depth of kd-tree subdivision: 17

average ray-mesh Intersection check count: 128.707

Bunny 512 \* 512

render time: 21.031 seconds (原本: 5780.73 SEC)

total kd-tree node count: 18094

max depth of kd-tree subdivision: 17

average ray-mesh Intersection check count: 128.655

從以上的數據可以看到，在小型的模型(monkey)裡，KD-tree的深度只有9，總共建構了138個kd-node，並且對於每次要進行ray-triangle intersection時，平均需要檢查的數量從原本的980個(與所有mesh進行檢查)變成只需要檢查133.33個。相較於原本的速度大約快了7~10倍。

在大型的模型(bunny)中，KD-Tree的深度是17，總共建構了18094格kd-node，並且對於每次要進行ray-triangle intersection時，平均需要檢查的數量從原本的70000個MESH變成只需要128.655個mesh，因此速度提升了大約300倍！

儘管對於較小的模型，可能因為平均ray-triangle intersection檢查數量減少所提升的效能不足以完全彌補檢查bounding box與ray的intersection花的時間(大約平均提升7~10倍的速度)，但是實際上真正的效能瓶頸一般還是大型、複雜的物件，我的實作確實有效提升了對於大型複雜物件的渲染速度到達約300倍的加速。

實作方法&心得:

我實作KD-Tree的方法主要是先把單一物件中的所有mesh放進root kd-node中，接著從這些mesh中估計出一個可以把所有mesh包覆在內的bounding box，接著依照此bounding box中最長的axis來當作劃分左右子樹的依據，找出此kd-node中所有Mesh對應axis的值，進行排序，並且取出中位數當作左右子樹的臨界值 (例如x軸最長，則從所有Mesh中找出他們的x值的中位數，把所有mesh(只要有任何一個頂點)中x值小於此值的mesh丟到左子樹，大於此值的mesh丟到右子樹)，並且遞迴地建構kd tree，直到某一層他劃分出來的左右子樹中的mesh的重疊比例高過一定的threshold(根據模型大小計算)就停止繼續建構左右子樹。(因為triangle mesh有三個頂點，所有有時單一mesh會同時被丟到左右子樹中，當同時存在左右子樹裡的mesh的比例跟總mesh樹的筆直高到一定程度，我們可以想見再繼續建構子樹也沒有太大的意義，故停止)。這個比值的threshold是根據模型大小來計算的，取法為 “同時存在於左右子樹中的mesh數” / ( 父節點mesh數 / log10(total mesh number) )。

以上是建構kd-tree的部分，而實際在進行hit test的方法，主要就是每次要判斷某個ray會打到場上哪些物件時，就對於所有的object的root KDnode做hit test，(hit test演算法參照的是:

<https://www.scratchapixel.com/lessons/3d-basic-rendering/minimal-ray-tracer-rendering-simple-shapes/ray-box-intersection>

這個網址的實作方式)，若有打中root node，則進一步檢查是否有打中左右子樹的bounding box，若有，則往該方向的子樹進一步進行以上的步驟，遞迴地直到檢查到leaf節點為止。到達leaf節點時，則對於節點中包含的所有mesh進行ray-triangle intersection test。藉由這樣的方法，我們可以有效的減少需要檢查triangle intersection test的mesh數量。