Computer Organization and Assembly Languages

FINAL PROJECT

Ray Bomb 3D

B96203005 曾紀為
B96701225 陳昱儒
B96b02054 譚承恩
B97902014 周哲平

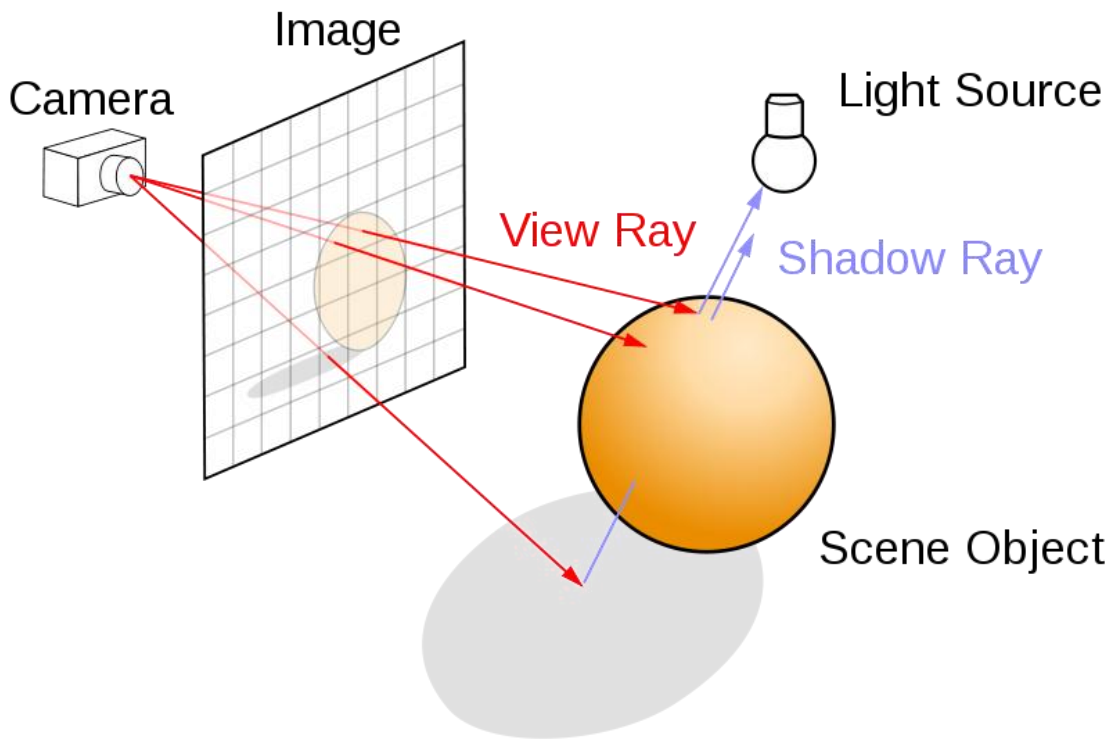
INDEX

設計理念

3D 算圖近年來重要的電腦繪圖技術，而其中的光跡追蹤演算法，擬真程度十分良好，對於各種特殊光學機制，如「折射」、「反射」等皆能有效模擬運算，唯一問題是要對畫面上的每個 **pixel** 進行一次運算，缺乏了速度上的優勢。

光跡追蹤需要對大量的點進行程序上完全相同的計算，又有運算效率上急需突破的障礙，因此讓我們想到利用組語近年來 **SIMD** 技術，來設法加速我們算圖的速度，以期達到即時 **ray tracing** 算圖的範疇。

原理說明



Ray tracing 意即光跡追蹤，其基本原理，是循著光線入射攝影機的反方向，算出光線先前與物件的交點，再於這個交點經過一次反射的逆運算，找出這道光源之反射光來自何方，如此便可確定射影機在畫面上某個點，接收到的亮度資訊為何，並換算出物體顏色，填入我們預先設好的畫布上。

對畫布上的每個點進行一次運算後得到的結果，就是我們算出的圖片，以像素點陣的資料型態存在，可供生成 **BMP** 或印於螢幕上之用。

操作說明

●●

Input:

「**input**」檔案結構如下：

<(d | s) (Use 'D'efault camera attributes or 'S'et user defined camera attributes?)

[(To set user defined camera attributes, enter attributes below)

x pos of camera y pos of camera z pos of camera

x pos of top-left corner of the projection plane y pos of top-left corner of the projection plane z pos of top-left corner of the projection plane

x pos of top-right corner of the projection plane y pos of top-right corner of the projection plane z pos of top-right corner of the projection plane

x pos of down-left corner of the projection plane y pos of down-left corner of the projection plane z pos of down-left corner of the projection plane

]

x component of light y component of light z component of light

number of triangles to render

xa ya za xb yb zb xc yc zc R G B(...of plane 0)

xa ya za xb yb zb xc yc zc R G B(...of plane 1)

xa ya za xb yb zb xc yc zc R G B(...of plane 2)

...

xa ya za xb yb zb xc yc zc R G B(...of plane n)

<(o | n) (Do some 'O'perations to camera or do 'N'o operations to camera)

[(To do some operations to camera, write the operations below)

number of operations

<op-axis angle in radius (...of op 0)

<op-axis angle in radius (...of op 1)

<op-axis angle in radius (...of op 2)

...

<op-axis angle in radius (...of op n)

] (End of file)

zip 檔中將付上簡單 **input** 範例做為參考，本程式需將 **input** 檔建立於 **D:\RayBomb3D\input.txt** 讀入。輸出則位於 **D:\RayBomb3D\data.bmp**。

關於 **input** 之詳細解釋如下。

(1) 攝影機位置**:**輸入**<d** 或**<s**、**<d** 代表攝影機的位置自動設定在圖形正上方，**<s** 則由使用者輸入攝影機所在位置的座標

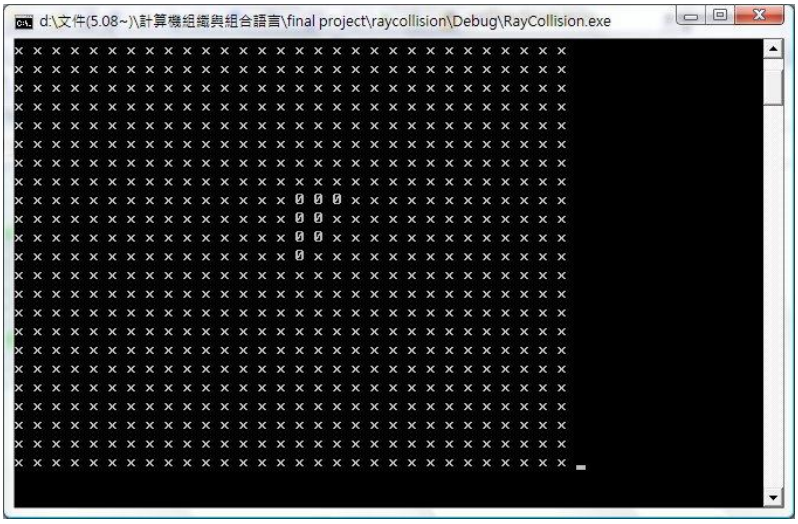
(2) 投影幕位置與大小**:**輸入三個點座標，分別為原點、**a** 端和 **b** 端，這三個點所夾的 **a,b** 向量可以定義一個螢幕所在的平面與大小。

(3) 光源方向**:**輸入光行進的方向向量，因為是假設光源來自無限遠處，所以沒有起點座標。

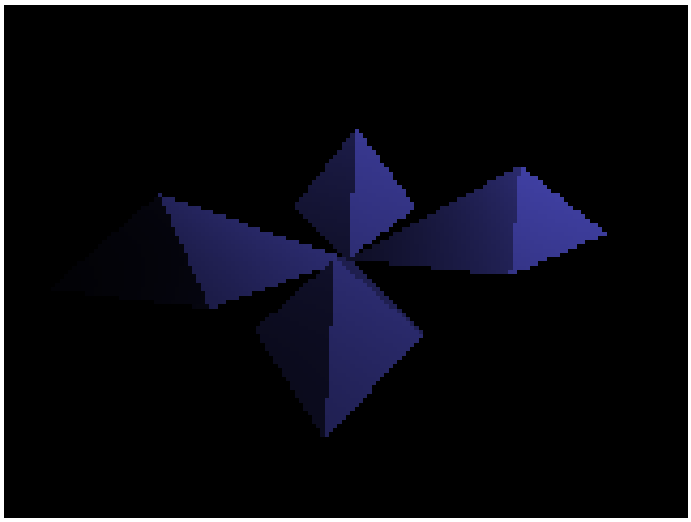
(4) 平面位置與顏色**:**先輸入預計要輸入的平面個數，再輸入平面參數，以三個點座標定義一個平面，最後輸入顏色是三個整數值，代表 **R**、**G**、**B**，範圍各由 **0~255**

(5) 調整攝影機與視平面角度**:**輸入**<n** 代表不對攝影機作旋轉，輸入**<o**，則接著輸入操作的步驟數，然後輸入要調整的軸與徑度**(**如 **<x 0.7)**

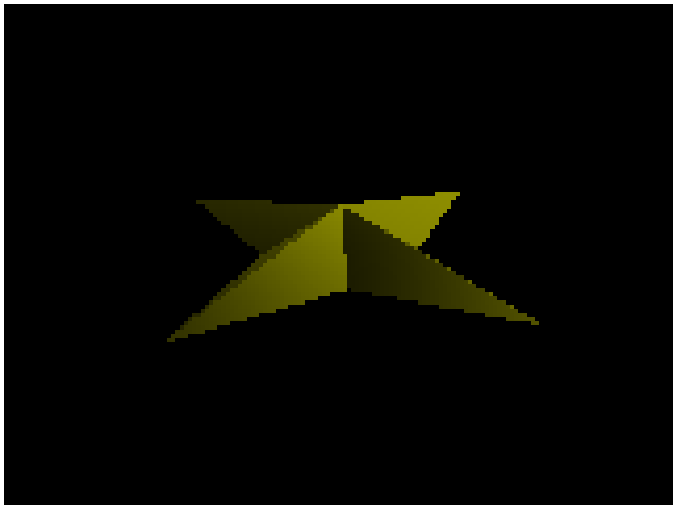
Output:



最初偵測光線與平面交點，建構出每個點的平面 **id** 之 **output** 測試。



一個較為複雜的後期 **output**。



另一個較為複雜的後期測試資料 **output**。

程式內容

- model.h

  處理程式時要讀取不同的資訊，如點，向量，平面等

- utility.h

  處理程式可能會用到的計算小工具，如求內積、外積、算距離等，

  此程式碼使用 SSE 組語加速。

- vision.h

  處理相機和投影幕的資訊，包含相機和投影幕的旋轉。

- file.h

  將測資從檔案讀取至記憶體中，供後面運作時取用。

- reflection.h

  處理反射的運算，並回傳某個點的色彩值，此程式碼大量使用 SSE

  加速，一個函式內最多可同時處理四組數據的內積。

組內分工

- 陳昱儒**:**

INPUT:一個檔案

OUTPUT: 一個 **plane array / light object / camera object** 等等。

a. 設計本程式專用的 **3D** 檔案格式。

b. 設計讀取這個檔案格式中各資料的幾種函式。

- 曾紀為**:**

INPUT: (含所有 **plane** 的**)plane array**、**camera object**。

OUTPUT: 每條 **ray** 撞擊到的 **2D plane array**。

a. 設計由 **camera** 投射視線的函式。

b. 設計偵測視線第一撞擊平面的函式。

- 周哲平**:**

INPUT: 一條視線、一個被撞擊的 **plane object**、一個 **light object**。

OUTPUT: 一個 **24-bit RGB color**。

a. 設計射線的一次反射線函式。

b. 設計由一次反射線和 **light object** 計算顏色的函式。

- 譚承恩

INPUT: 一個 **color array**。

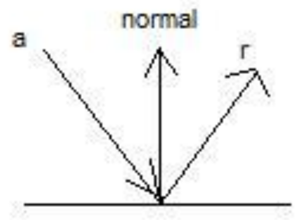
OUTPUT: 一個 **BMP** 圖檔。

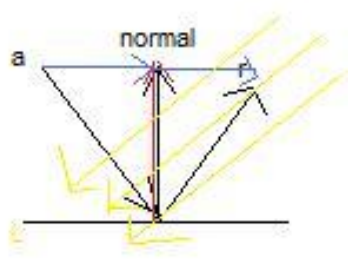
a. 設計生成 **bmp file header** 的函式。

b. 設計將 **bmp file header** 和 **color array** 填入 **bmp** 檔案中的函式。

程式設計細節

反射光線的計算(周哲平)





First, we have a(ray), normal(the normal of the plane)
Then, in order to get r(the reflection)
Step1. Find x*normal(the red line at right)

So that, the (a + x*normal)*normal is zero. (* means the inner production)

Finally, we have x = -a*b/a*a

Step2. The reflection r = a+(a + x*normal)*2 = 3*a + 2*x*normal
Second, we get the cosine of L (the light which is the yellow one) and r

Where cosine = L*r/|L|*|r|

Finally, we suppose the plane have color with RED r, GREEN g, and BLUE b.

And, when cosine is -1 the plane shows its own color, while the cosine is 1, it shows black. So we have a linearly equation

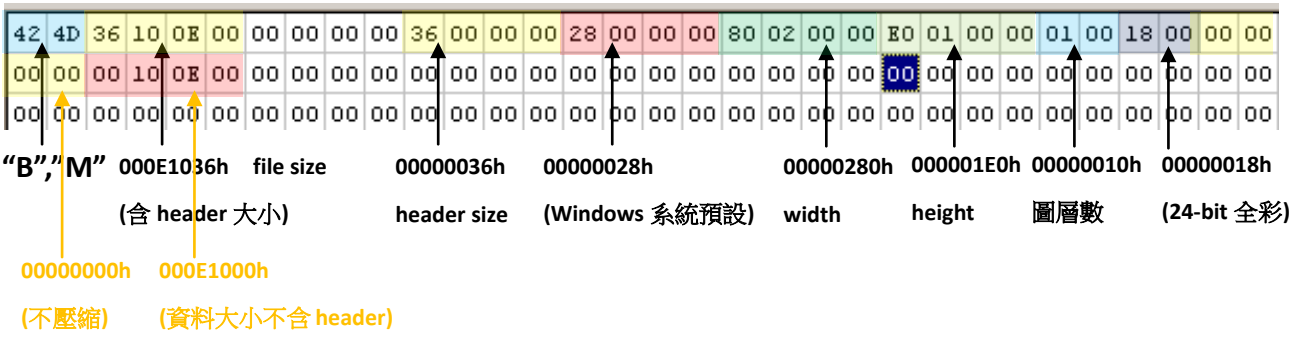
=>Show color = (1-cosine)*color/2

In order to speed up the computation, I use SSE so that I can compute four planes at a time. But, SSE only has eight registers, accordingly, I have to store and load many times and it let the efficiency go worse.

The function’s output is an address of array
Where array = {plane1RED, plane1GREEN, plane1BLUE,
plane2RED, plane2GREEN, plane2BLUE,
plane3RED, plane3GREEN, plane3BLUE,
plane4RED, plane4GREEN, plane4BLUE}

輸出(譚承恩)

檔案輸出：當每一個像素的顏色都算好，成為一個陣列的時候，就可以將其輸出成點陣圖。點陣圖的格式如下：



P.S.其它部分都可以填 0，另外這是 little indian 的排法。

EX:640*480 全彩圖的 header(藍色標示前的部分，共有 36h 個)

其中前 54(36h)個 Byte 放的是 header 資料，接下來在第 36h(藍色標示)，後面存放著每一個 pixel 的資料，3 Bytes 代表 1 pixel(G,B,R 排列)。

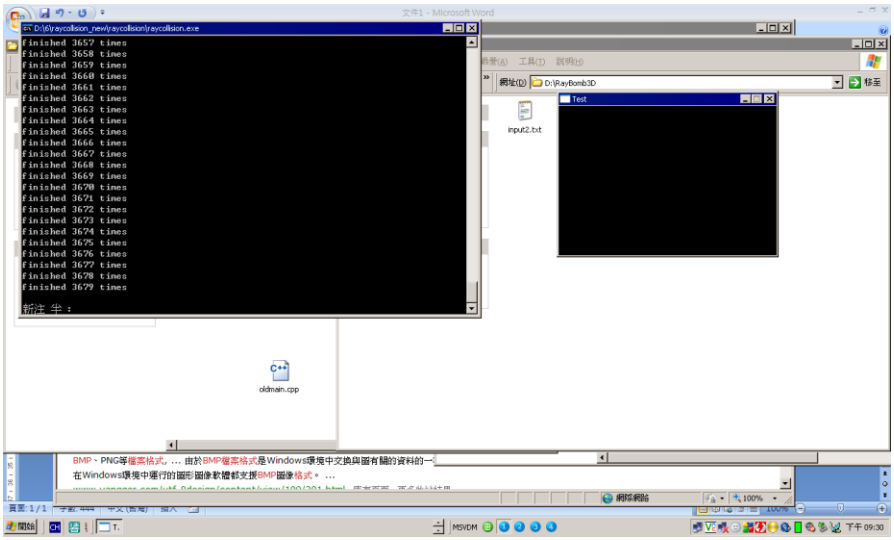
於是算出 color 陣列後只要加入 header 和資料即可完成.bmp 檔。

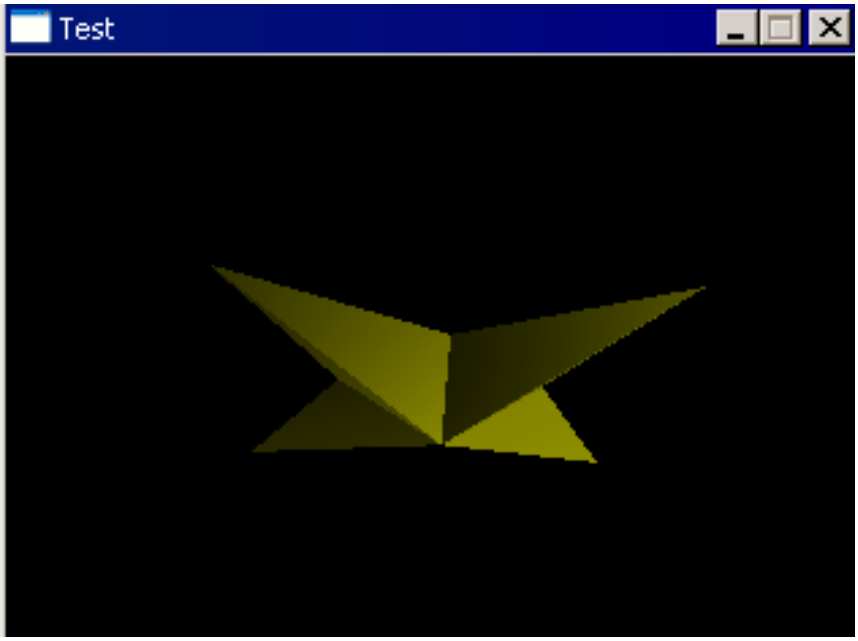
螢幕輸出(不附於本次上傳之 zip 檔中)：這裡使用的是 Windows API 的寫法，本來打算用 MFC 來寫，不過發現對視窗不夠了解，所以改用比較原始的 API 寫法，這樣子可以對每個變數做更多的控制，另一方面這種方法也比 MFC 要來得快。

一開始對視窗結構不了解，把演算的過程直接放入 CallBack Function 中，其結果就是 CallBack Function 不斷被呼叫著，導致演算法不斷地在跑，最後吃光整個系統的記憶體……到了後來，把演算法放入觸發事件(message)裡的反應中(採用 case 的架構，按右鍵有回應，按左鍵又有另一回應)。

最後演算完之後，使用 SetPixel()，把一個一個像素輸出至螢幕即可：

Ex：SetPixel(hDC,x,y,RGB(255,255,255))➔在(x,y)輸出白色





例如上圖，在左邊的視窗按下右鍵後就會執行演算，跳出左邊的 Dos 視窗。算出每一個點的資料之後可以按下左鍵，就會畫出圖形。

動畫輸出：這個部分沒有辦法在 Demo 上面跑，因為算一張圖需要十多秒，所以時間不夠製出動畫……

原先打算用 SetPixel 一個一個畫面重畫搭配倒數計時器以達成動畫效果，但其實 SetPixel 是一個一個點輸出，播起動畫來會是由上往下不斷刷新，一點都沒有動畫的感覺。

最後是用 SetDIBits 函數，將結果先算出來，不直接畫入 hDC 中，而是先畫在一個暫存區(也是 HDC 類別)，最後用 Bitblt 函式，直接把暫存區整個畫面拷到 hDC(目前視窗畫面)上，就可以完成動畫的效果了……曾經試過用全黑到全白，每個影格停留 1/16 秒，是可以展現動畫的，不過沒有套用我們算的 3D 圖上。

光線與三角面相交偵測(曾紀為)

我的做法是先利用投影公式，將代表攝影機位置的點投射到物件中由 ABC 三個點所定出面上(不論是否位於此面所代表的三角形「之內」)，成為 D 點，之後便可以利用三角形面積公式：

$$S = \frac{1}{2}\sqrt{\left(\det\begin{pmatrix} x_A & x_B & x_C \\ y_A & y_B & y_C \\ 1 & 1 & 1 \end{pmatrix}\right)^2 + \left(\det\begin{pmatrix} y_A & y_B & y_C \\ z_A & z_B & z_C \\ 1 & 1 & 1 \end{pmatrix}\right)^2 + \left(\det\begin{pmatrix} z_A & z_B & z_C \\ x_A & x_B & x_C \\ 1 & 1 & 1 \end{pmatrix}\right)^2}.$$

計算出 **ABC** 面積是否等於 **ABD+ACD+BCD** 之總和，若相等則光線與此平面有相交，反之則否。

參考資料

http://en.wikipedia.org/wiki/Ray_tracing_(graphics)

http://en.wikipedia.org/wiki/Triangle

http://msdn.microsoft.com/en-us/library/t467de55(VS.80).aspx

http://en.wikipedia.org/wiki/.bmp

Eric Lengyel, 2004, Mathematics for 3D Game Programming and Computer Graphics 2ND edition.

特別感謝

Mr. CYY 以及助教們

RayBomb3D 下載點

http://homepage.ntu.edu.tw/~b96203005/asm/RayBomb3D.exe