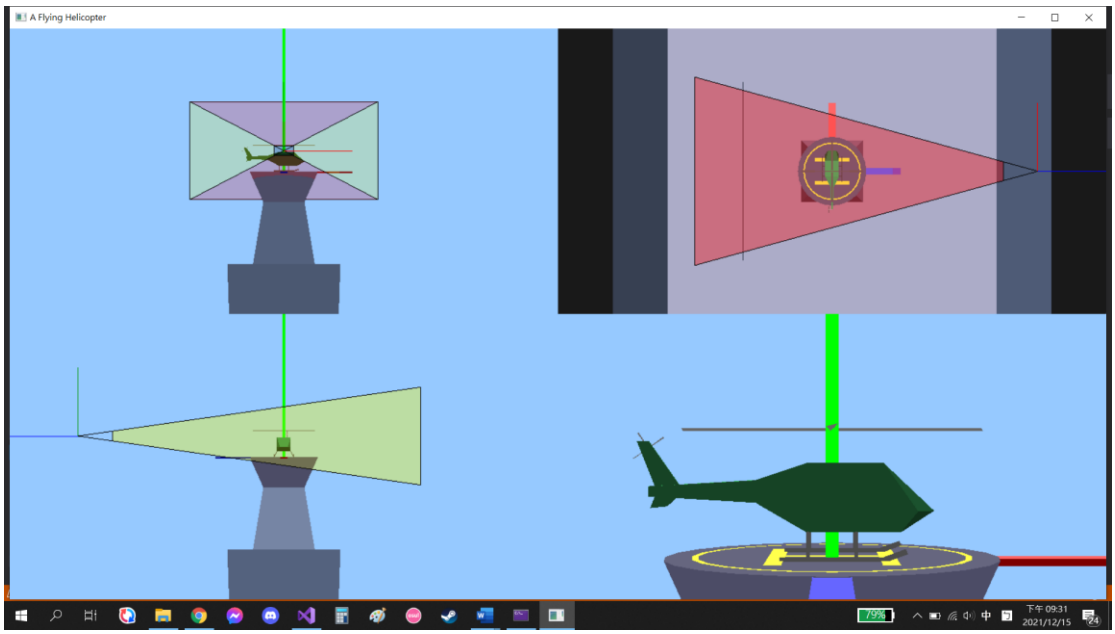
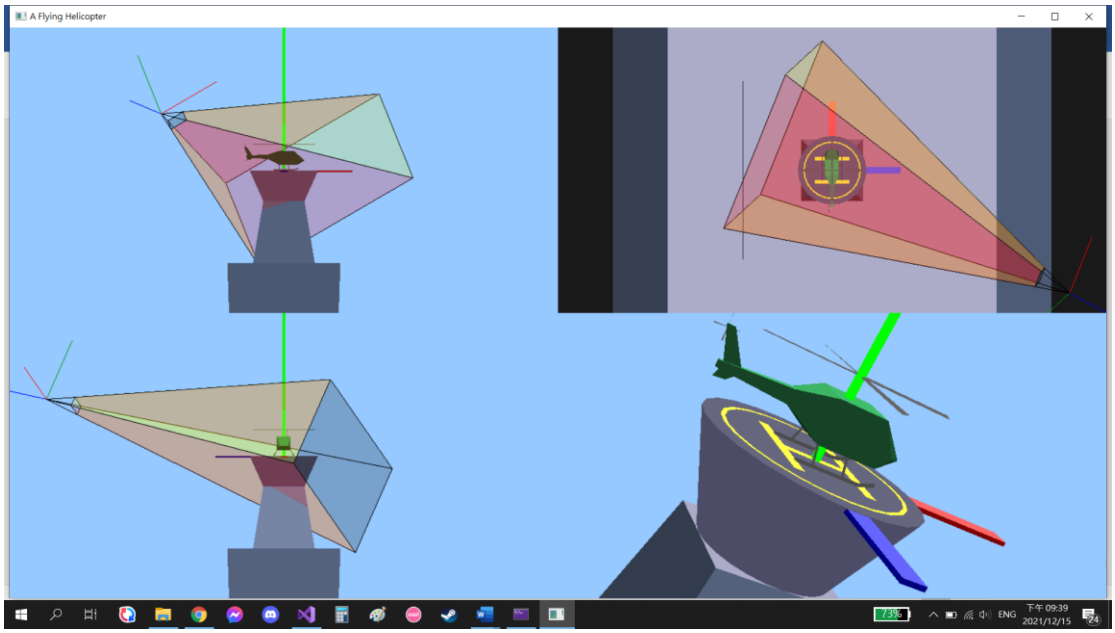
執行畫面

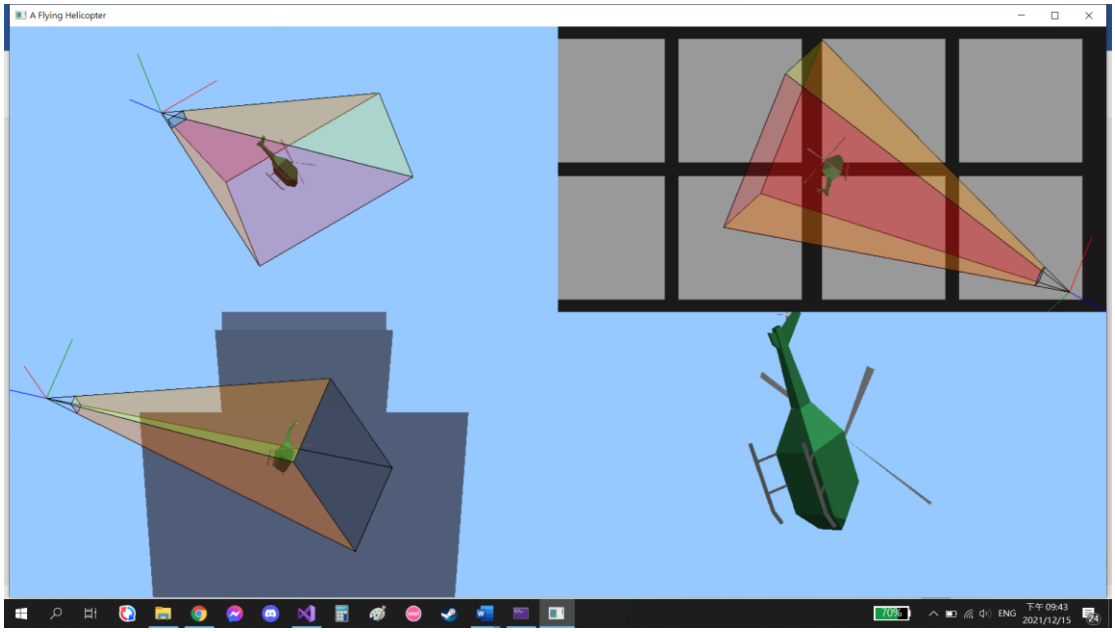


紅色線為 X 軸，綠色為 Y 軸，藍色為 Z 軸

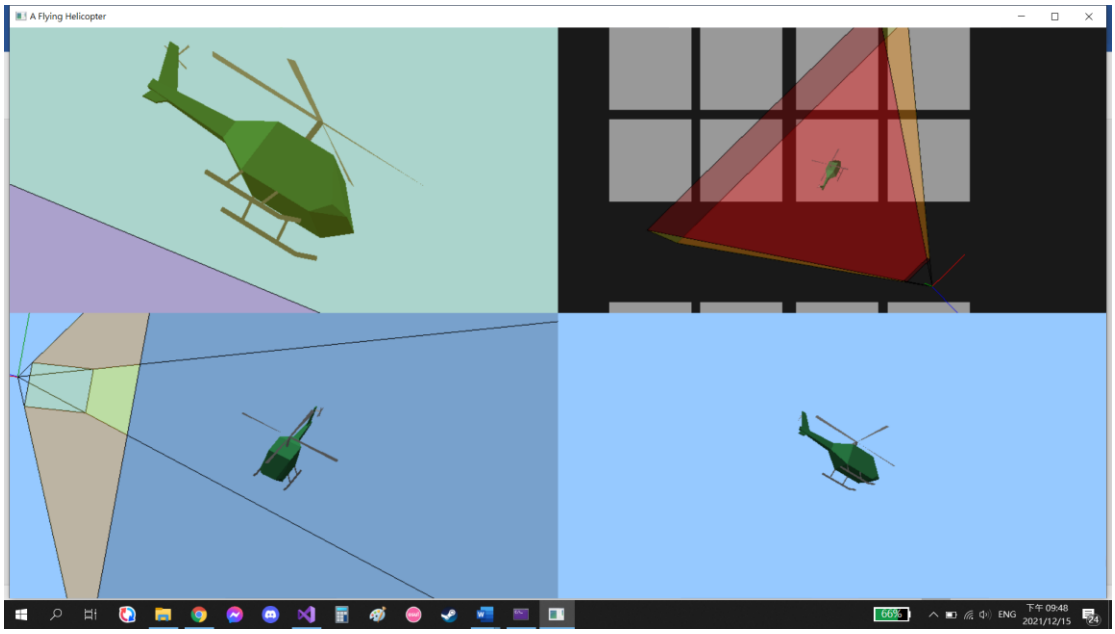


灰色面為 near 與 far，紅色面為視線上下，黃色面為視線左右

左上角是往外看，會被大腦騙，左下角是往內看



鏡頭可以選擇跟著直升機



可以對不同畫面縮放，或對主畫面 zoom in/out

上面圖中的 blend 顏色有些錯，我已修正好了，但懶得重新截圖

使用者手冊

使用按鍵

按鍵作用 (不分大小寫)

按鍵	效果	定義
Double- L-Ctrl	關閉引擎	使不按 ctrl 或 space 時升力趨於 0
Double- Space	啟動引擎	使不按 ctrl 或 space 時升力趨於與重力平衡
w	前進(加速)	自動拉動傾斜主旋翼，使機身趨於 往前傾斜相對天頂 25/40/70 度
s	後退(減速)	自動拉動傾斜主旋翼，使機身趨於 往後傾斜相對天頂 25/40/40 度
a	左飛、左轉	自動拉動傾斜主旋翼，使機身趨於 往左傾斜相對天頂 25/40/40 度
d	右飛、右轉	自動拉動傾斜主旋翼，使機身趨於 往右傾斜相對天頂 25/40/40 度
w、s+a、d	以上傾斜組合 共八方位	自動拉動傾斜主旋翼，使機身趨於 往專設方向傾斜至相對天頂專設度數
q、z	左旋、左轉	加強尾旋翼推力
e、c	右旋、右轉	減弱尾旋翼推力
L-Ctrl	垂降	減弱主旋翼推力
Space	爬升	加強主旋翼推力
L-Shift	全速模式	使傾斜角上限提升至第二段
Double- W	俯衝模式	使傾斜角上限提升至第三段
w、a、s、d **都不按**	水平回正 巡航模式	自動拉動傾斜主旋翼，使機身趨於水平 當傾斜小於 25 度則切回第一段(巡航模式)
L-Ctrl、space **都不按**	高度平衡	在引擎啟動時隨時調整主旋翼推力 使垂直分立與重力趨向抵銷

f、滑鼠中鍵	**鎖定視角**	鏡頭跟隨直升機
m	**切換監視器**	切換 分割鏡頭/perspective/x/y/z 方向視角
r	**切換鏡頭操控模式**	切換 平移模式/旋轉模式
up	視點上升	在平移模式時
down	視點下降	在平移模式時
left	視點左移	在平移模式時
right	視點右移	在平移模式時
R-Ctrl	視點後退	在平移模式時
R-Shift	視點前進	在平移模式時
up	視角上升	在旋轉模式時
down	視角下降	在旋轉模式時
left	視角左旋	在旋轉模式時
right	視角右旋	在旋轉模式時
R-Ctrl	視角逆旋	在旋轉模式時
R-Shift	視角順旋	在旋轉模式時
滾輪上滑	放大視角	對所指畫面放大
滾輪下滑	縮小視角	對所指畫面縮小

特別實作與演算法

(上一版本太多演算法，程式根本截不完就沒貼)

都是基本需求，沒什麼特別好講的

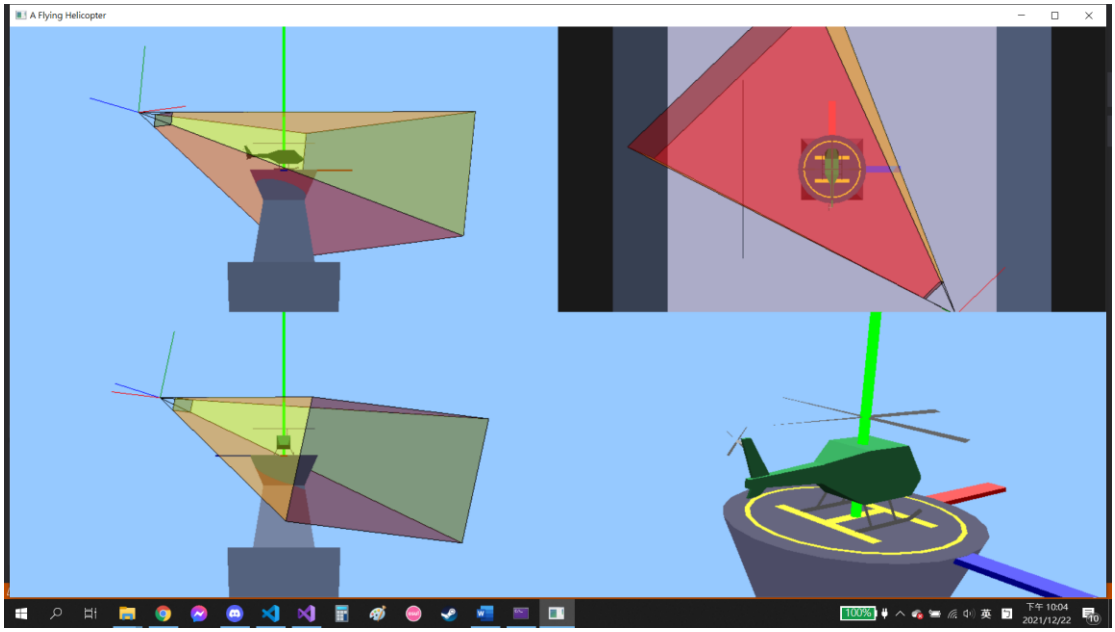
只有這個比較有一點技術

View volume 的 Blend 排序方法

```
131   vector<int> frontSet, backSet;
132   Coordinate normal[6];
133   float innProdValue[6];
134   normal[0] = { 0, 0, 1 };
135   normal[1] = { 0, 0, -1 };
136   for (int i = 2; i < 6; i++)
137       normal[i] = outerProduct(volumeVertex[volumeFace[i].order[1]] - volumeVertex[volumeFace[i].order[0]],
138                                volumeVertex[volumeFace[i].order[2]] - volumeVertex[volumeFace[i].order[1]]);
139   if (m == 1)
140       for (int i = 0; i < 6; i++)
141           innProdValue[i] = normal[i].x * eyeAxisX.x + normal[i].y * eyeAxisY.x + normal[i].z * eyeAxisZ.x;
142   else if(m==2)
143       for (int i = 0; i < 6; i++)
144           innProdValue[i] = normal[i].x * eyeAxisX.y + normal[i].y * eyeAxisY.y + normal[i].z * eyeAxisZ.y;
145   else if(m==3)
146       for (int i = 0; i < 6; i++)
147           innProdValue[i] = normal[i].x * eyeAxisX.z + normal[i].y * eyeAxisY.z + normal[i].z * eyeAxisZ.z;
148   for (int i = 0; i < 6; i++) {
149       if(innProdValue[i] > 0) frontSet.push_back(i);
150       else backSet.push_back(i);
151   }
152   for (auto order : backSet) volumeFace[order].drawAsRGB(volumeVertex);
153   for (auto order : frontSet) volumeFace[order].drawAsRGB(volumeVertex);
```

因為 view volume 的六個面從任一個視角來看，一個面不是蓋住其他面，就是被其他面蓋住。故將其分為兩堆，先畫出被蓋住的面再畫出蓋住別人的面，就能呈現正確的透明顏色。而分堆的依據就是 該面法向量 內積 視角方向:

$$n \cdot v = n_u \cdot v + n_v \cdot v + n_w \cdot v$$
$$= (n_x * axisX) \cdot v + (n_y * axisY) \cdot v + (n_z * axisZ) \cdot v$$
$$= n_x * axisX_x + n_y * axisY_x + n_z * axisZ_x$$
$$suppose\ v = (1,0,0)$$



這才是正確版

心得

這次的製作量就比較少了，但由於前一版我是用經緯座標計算視角的，和這次的作業要求的三軸座標不同，等於是完全新的工作。如果要統一計算必須放棄先前的經緯系，改寫為三軸系來計算，還要統一一堆參數超麻煩，就沒時間做了。

在範例與課堂上領悟到有效率得旋轉座標軸的設計方法之原理：事先將三角函數值設為常數、將旋轉拆解為對 xyz 其中一軸旋轉，即簡化為多次在平面上旋轉兩軸，以極小的誤差換取更高的效率。但因為我運動都是以物理量計算，故都是用一般旋轉公式，還真不知道怎麼應用平面旋轉，只有視角操作這種基本旋轉可以用上。(幸好有教平面旋轉，差點沒忘了還有這可以用，要不然我會走火入魔繼續用一般旋轉來處理視角操作...)