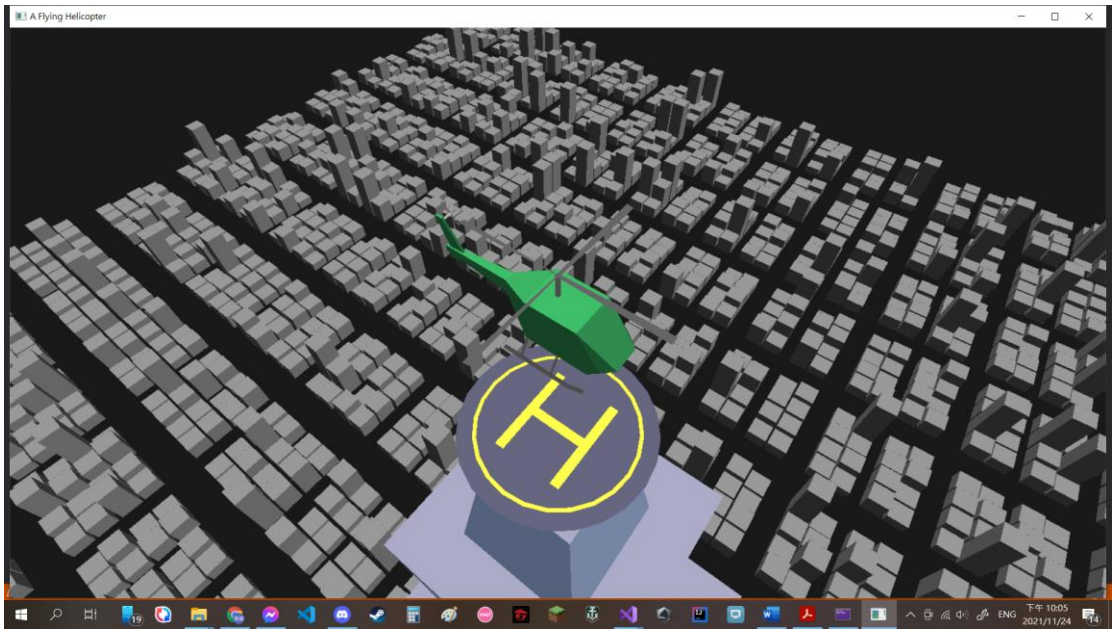
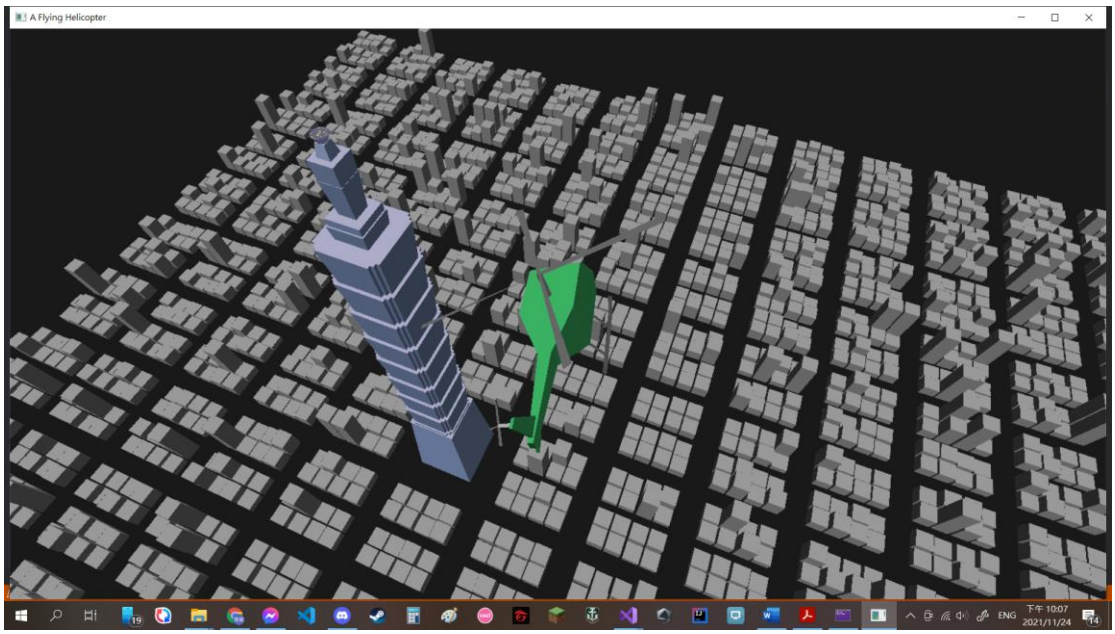
執行畫面

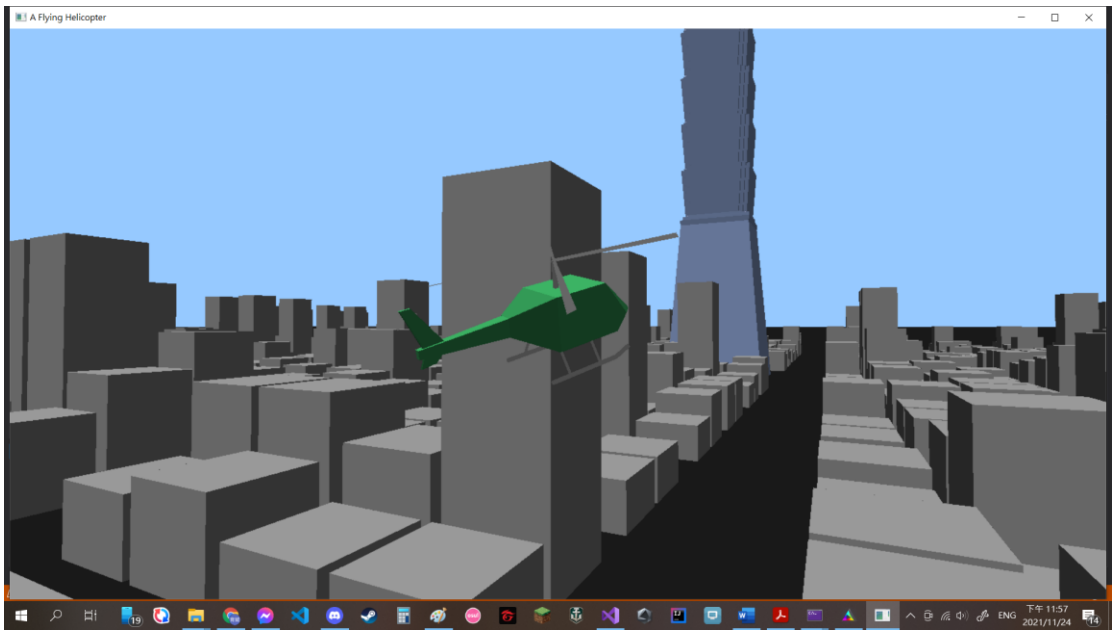
停機坪



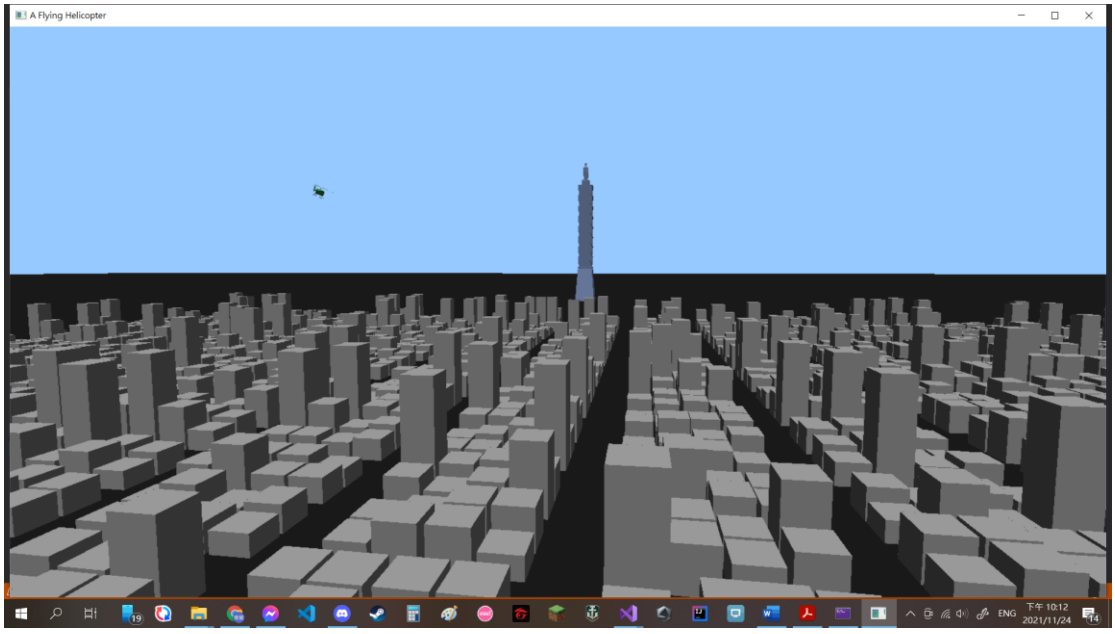
空拍街景



地面街景



切換視角



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

v	切換視角	切換為第 一/二/三 人稱視角
f*(忘了補)、*滑鼠中鍵	鎖定視角	切換 鎖定/自由 視角 鎖定直升機為視角中央/自行控制視角中央
up	視點上升	在第三人稱視角使鏡頭位置上升
down	視點下降	在第三人稱視角使鏡頭位置下降
left	視點左移	在第三人稱視角使鏡頭位置左移
right	視點右移	在第三人稱視角使鏡頭位置右移
R-Ctrl	縮小視角倍率	在第二人稱視角使視野拉廣
R-Ctrl	視點前進	在第一三人稱視角使鏡頭位置前進
R-Shift	放大視角倍率	在第二人稱視角使視野縮窄
R-Shift	視點後退	在第一三人稱視角使鏡頭位置後退
滑鼠游標	移動視線	在自由視角控制視線經緯度方向
滾輪上滑	視點前進	在第二人稱視角使鏡頭位置前進
滾輪上滑	放大視角倍率	在第一三人稱視角使視野縮窄
滾輪下滑	視點後退	在第二人稱視角使鏡頭位置後退
滾輪下滑	縮小視角倍率	在第一三人稱視角使視野拉廣
滑鼠右鍵	自由視角	在鎖定視角按住時切為自由視角

停機坪有點小，想看降落著地要點技巧，應該補地面也可停的

巡航模式: 可以邊移動邊爬升
全速模式: 在能平衡重力下，以最高速度移動，無法爬升
俯衝模式: 突破更高的速度，但高度會加速下降

特別實作與演算法

全物理控制

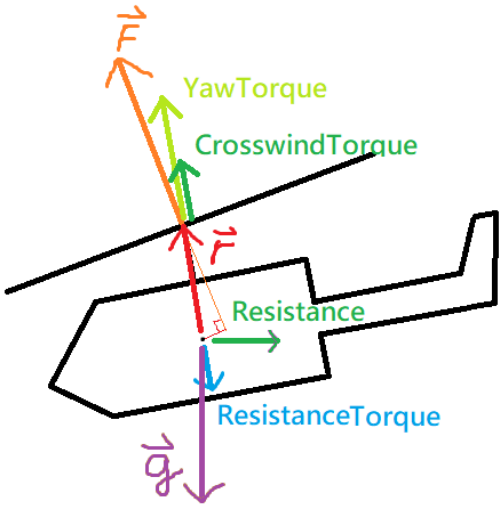
僅靠旋翼的推力與主旋翼的方向來間接對機身的控制，加上如重力、風阻力(矩)等被動力，實現作品中的所有運動

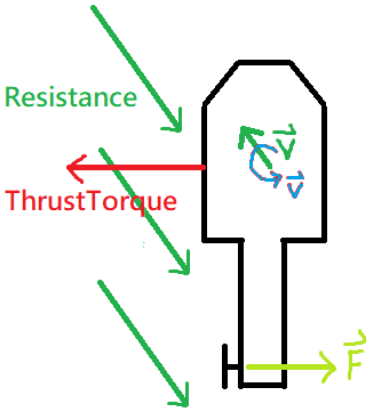
時間系統

以系統時間為標準、idleFunc 為瞬間點並演進時間(負責 display)，每次的 function 執行會計算與前一次的系統時間差(dt)，並以上一瞬間物體的運動物理量來演變出這一瞬間的世界，並更新其物理量
(在 console 印出來的那堆就是幀率倒數 dt)

運動力學

基礎原理示意圖
直升機正在往前傾斜，往左前方前進，且正在逆時針水平旋轉時





圖中名詞:

F(BladeThrust): 旋翼推力

Resistance: 與前進方向相反的阻力，以與速度成正比計算此

以上與重力三力合力決定平移運動

YawTorque: 尾旋翼推力矩與主旋翼造成的對機身內力矩 抵銷的水平旋轉力矩

ThrustTorque: 主旋翼推力矩 = r X F，使機身得以傾斜

CrossWindTorque: 阻力矩(Resistance 造成的力矩)，由於質心前方的受風面比後方的小，故前後抵銷後相當於往後延伸的力臂，如羽毛球原理，使機頭會漸漸朝向前進方向

ResistanceTorque: 旋轉阻力矩(是角速度造成的阻力)，與角速度成正比

以上四力合力矩決定旋轉運動

(至於 尾旋翼推力 與 尾旋翼對機身內力矩 就當成被主旋翼推力自動平衡掉了，不另外計算)

機體變數與運作機制

```
//位置與方向
Coordinate origin_r;
Coordinate axisX_r;
Coordinate axisY_r;
//合總物理量
Coordinate velocity;
Coordinate acceleration;
Coordinate angularVelocity; //公制弧度
Coordinate angularAcceleration;
//分(角)加速度(以下簡稱為力(矩))與其相關用值
Coordinate bladeThrust;      //主旋翼推力，亦為主旋翼旋轉方向
float limitBladeTiltAngle = 10; //bladeThrust方向相對axisY_r的最大傾角，意義即機身傾斜速度
Coordinate targetThrust;     //主旋翼目標推力
float maxThrust = 1.3 * gravity;
float minThrust = 0.7 * gravity;
float thrustAccelerate = 0.6;    //主旋翼加加速度值
Coordinate thrustTorque;     //主旋翼推力矩
Coordinate yawTorque;        //物理上包含 尾旋翼推力矩 + 主旋翼對機身內力矩
float limitYawTorque = 0.15 * PI;    //yaw省略角加加速度，會直接跳至此limit量值(順或逆都是)
//至於 尾旋翼推力 與 尾旋翼對機身內力矩 就當成被主旋翼推力自動平衡掉了，不另外計算
//重力與阻力(矩)這些被動力就不列為field
```

Coordinate 為向量座標之類別

變數解釋: (這些向量座標全以世界座標系作為基底)

origin_r	直升機質點座標
axisX_r、axisY_r	機頭(前方)與機頂(上方)的方向，Z 方向即決定
acceleration	將該瞬間所有加速度(力)的總和，並決定速度與位置 bladeThrust + gravity + resistance (接觸力另外算) 時間微分概念: a-t 圖為階梯形，v-t 圖為一次函數，x-t 圖為二次函數 故 dx 應是(v0+v)dt/2
angularAcceleration	完全同理，對應到角加速度(力矩)、角速度、角度 thrustTorque + yawTorque + resistanceTorque + crosswindTorque(接觸力矩另外算)
bladeThrust	主旋翼貢獻的加速度 (包含推力的方向與力度) 與 thrustTorque 同步 與 axisY_r 不一定相同方向，主旋翼會被拉動最多 10 度，產生力矩(thrustTorque)使機身傾斜，亦是 axisY_r 的目的地，axisX_r 當然也會同步
targetThrust	axisY_r 與 bladeThrust 操控的最終目的地 例如使用者單按 w 欲使直升機最終前傾 25 度，則 targetThrust 的方向即會設為 天頂方向對機體 z 軸旋轉 -25 度，隨後 bladeThrust 會盡量往 targetThrust 傾向，axisY_r 因此跟著傾斜，使 bladeThrust 可以更靠近 targetThrust 最大推力為 1.3g，最小為 0.7g
thrustAccelerate	扇葉轉速不能瞬間加速，bladeThrust 量值也必須如方向一樣漸漸逼近 targetThrust，類似推力的加加速率
resistance	阻力: 以正比速度計算
resistanceTorque	旋轉阻力: 以正比角速度計算
crosswindTorque	阻力矩: 畫一下圖可以知道是 resistance 外積 axisX_r

Helicopter.evolve(dt) 運動演算流程:

視按鍵決定 targetThrust -> 根據一套演算法決定 thrustTorque 與 bladeThrust -> 視按鍵決定 yawTorque -> 合總所有力(矩) -> 變動(角)速度 -> 判斷落地(若落地則再變動速度) -> 得出位置與方向

<u>坐標系旋轉</u>

就是抓三軸中的兩軸共旋轉三次的那個東東，由世界系 xy 轉到機體 xy

1. 旋轉當前 x 軸使 當前 z 軸、目標 x 軸、當前 x 軸 共平面
2. 旋轉當前 y 軸使 當前 x 軸 對到目標 x 軸

3. 旋轉 x 軸使 當前 y 與 z 軸 對到目標

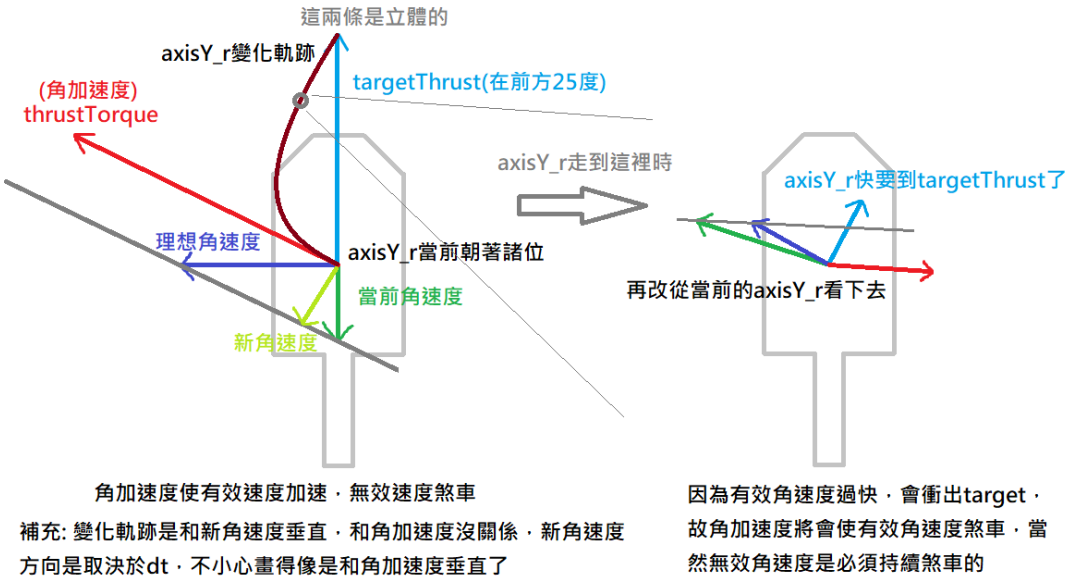
所有計算工具都在 PublicValue.h

自動傾斜演算法

不是物理計算，是演算方法，要由自己設計，是最困難的部分，我想好久

可以仔細觀察主旋翼相對機身傾斜的變化，要隨時自動調整主旋翼傾角，使機身能夠傾斜到指定傾角並使角速度靜止，故 bladeThrust 不能單只是越靠近 targetThrust 越好，否則會因慣性而來回擺動停不下來。簡單來說就是需要做個煞車，讓旋翼往反折回去，使機身的傾斜速度變慢，可以剛好在指定傾角處(targetThrust 方向)停下來。但傾斜是二維的，故無效分速度要減速，有效分速度才有加速或減速。

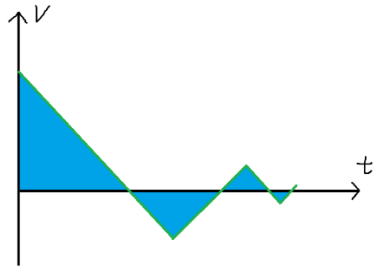
例如使用者可能按完 d 後按 a，從向右傾斜快速回到經過水平的瞬間，再改按 w 欲向前傾斜 25 度，此時示意圖如左半



圖中名詞:

以上角(加)速度都是指投影在這平面上的(畢竟傾斜和 yaw 獨立)

理想角速度: 根據剩餘角度與當前角速度推算如果接下來全速煞車的話，應該要用多少的角速度才能差不多在靜止時到達目標角度，若要精準會極其複雜，讓他超出一些也會自動修正到誤差越來越小



簡化示意圖: 它的面積即是 t0 時的剩餘角度

將第一個三角形面積控制在超出剩餘角度一些

有效角速度: 角速度投影到理想速度，扣掉有效就是無效角速度

落地接觸

因在時間最小單位下不可能計算它的受力時間，故直接以速度差計算，若該瞬間即將落地便加上其速度差，再得出位置與方向，達到反彈與摩擦效果

分為反彈力(矩)與動摩擦力(矩)，因為時間不夠我只做出一維碰撞，預期是將腳架設計成可碰撞的實體的，將腳架四角落作為接觸質點，公式都算好了，可能下次再放吧

視角計算

視線向量為經緯度座標系，滑鼠移動等距改變其經度緯度，另外有做當滑鼠移到視窗邊界會繼續延伸，因為我不知道怎麼做成像遊戲那樣游標無邊界。經緯座標系轉三維座標系很簡單就不解釋了(轉回來很麻煩也不解釋)

第二人稱: 視點為機體位置往視線反向走

鎖定視角: 要根據機體位置或方向反推經緯度回去(因為要配合右鍵點下去不要瞬移)，三人稱都要有夠麻煩

打光亮度計算

在 PublicValue.h 的 Face.drawAsHSLInc()

面的顏色以 HSL 記錄，光源為正上方，將 L 值乘上反光率，再用公式轉換為 RGB，我亂算的反射率 = 和 cos 入射角正比的直射光(為了讓背光面亮度也不同)，再簡單加上定值的漫射光

建築生成

隨著機體位置生成範圍內的建築(走到哪生到哪)，用二維陣列儲存每個 block，每個 block 又為長度 16 陣列，記錄 4*4 建築的高度，一開始即亂數決定好，隨著直升機移動而更新被我註解掉了，因為沒差。然後對直升機當前位置取餘數決定在哪個 block 上空，並把二維陣列的相對位置當作循環環面，以直升機所在 block 為中心 block 延伸，即可做出跟隨效果

上傳版的 Helicopter.h 之 bug 須修正處:

1. 少乘 5

```
thrustTorque = (5 * difV.getLength() > maxTorque ? maxTorque * difV.identity() : difV);
```

這行改成

```
thrustTorque = (5 * difV.getLength() > maxTorque ? maxTorque * difV.identity() : 5 * difV);
```

2. display 忘了轉座標系尷尬…

```
Coordinate rotateAxis = outerProduct(axisY_r, bladeThrust);
```

```
                    glTranslatef(0, 1, 0);
                    glRotatef(includedAngle(axisY_r, bladeThrust), rotateAxis.x,
rotateAxis.y, rotateAxis.z);
```

這段改成

```
Coordinate bladeThrust_t = Coordinate(
                        innerProduct(bladeThrust, axisX_r),
                        innerProduct(bladeThrust, axisY_r),
                        innerProduct(bladeThrust, outerProduct(axisX_r, axisY_r))
                    );
                    Coordinate rotateAxis = outerProduct({ 0, 1, 0 }, bladeThrust_t);
                    glTranslatef(0, 1, 0);
                    glRotatef(includedAngle({ 0, 1, 0 }, bladeThrust_t), rotateAxis.x,
rotateAxis.y, rotateAxis.z);
```

抱歉沒注意，這樣才能觀察到真正的旋翼傾斜變化

心得

第一次設計物理建模，都是自己猜想，旋轉力學的部分覺得是一大挑戰，算是一個成就。

工程規劃往往不準，動手下去寫才發現漏想了複雜的東西，結果比預期花了更多時間，又或者有時比想像的輕鬆，做了才知道。

都在玩向量計算，但程式功能上沒特別學什麼，不知會不會有點走歪了。

遇到的問題都有解決，只是計算比想像的複雜了些，但沒太大阻礙，也想過說我的計算方法可能有比較笨的地方，還請指教。

雖然有預期到畫房子吃最多效能，但其他 3D 遊戲是用什麼方法在效能允許下搞這麼多環境的，相比之下本作單純許多卻已負荷不了，是最好奇的疑問。

OS: 因為太好玩而佔用太多時間了...，之後可能會克制一下

參考資料

https://www.rapidtables.org/zh-TW/convert/color/hsl-to-rgb.html

https://zh.wikipedia.org/wiki/%E7%BD%97%E5%BE%B7%E9%87%8C%E6%A0%BC%E6%97%8B%E8%BD%AC%E5%85%AC%E5%BC%8F