

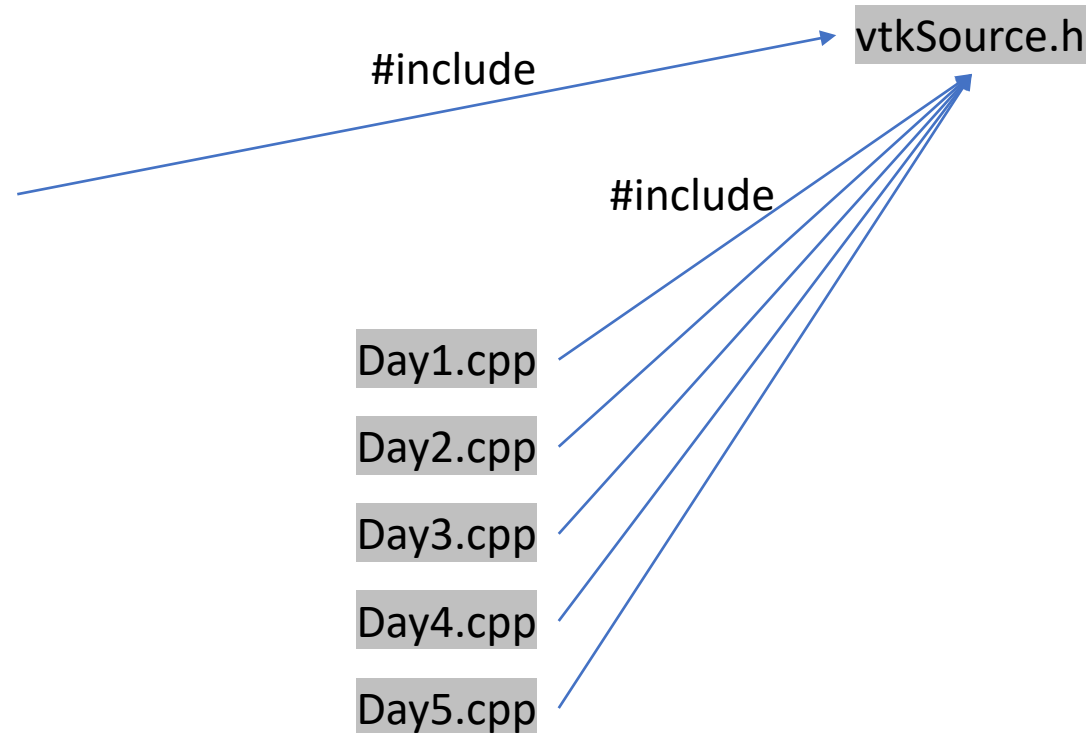
Day3

1. Camera
2. Light
3. Animation: Earth & Moon

How I prepare my VTK project?

Main.cpp

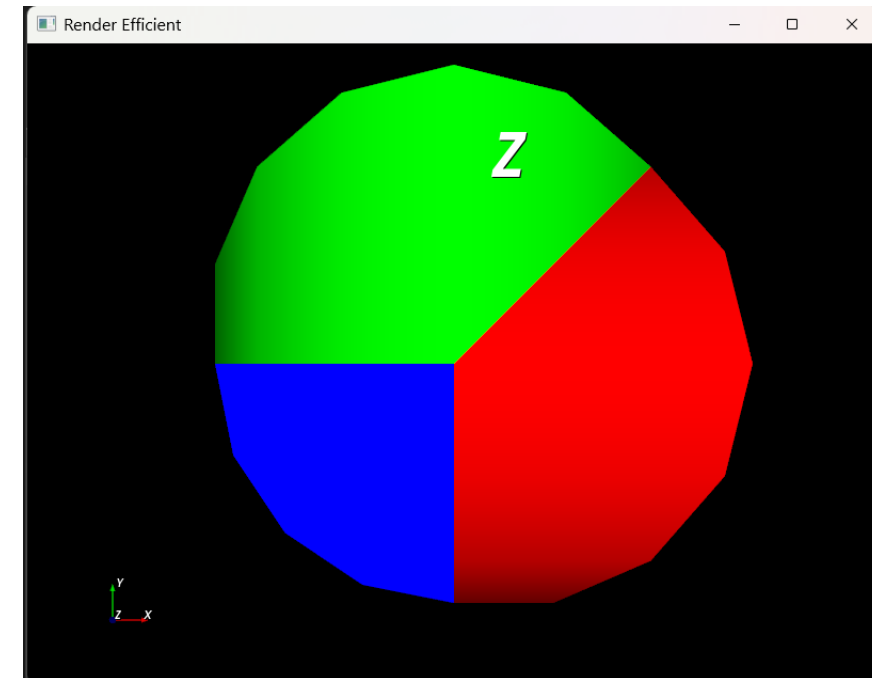
```
int main(){  
    Day1function();  
    Day2function();  
    Day3function();  
    Day4function();  
    Day5function();  
}
```



Camera position problem

```
Void RotateTest2(){  
...  
    // Adding the orientation axes  
    AddAxes(ren1);  
  
    vtkNew<vtkRenderWindow> renWin;  
    renWin->AddRenderer(ren1);  
    renWin->SetSize(800, 600);  
    renWin->SetWindowName("Render Efficient");  
  
    vtkNew<vtkRenderWindowInteractor> iren;  
    iren->SetRenderWindow(renWin);  
  
    // Setting the camera position to view the objects from a better angle  
    ren1->ResetCamera();  
  
    // =====  
    // 設定左下角的 Orientation Marker Widget  
    // =====  
    vtkSmartPointer<vtkAxesActor> cornerAxes = CreateAxes();  
  
    vtkNew<vtkOrientationMarkerWidget> widget;  
    widget->SetOrientationMarker(cornerAxes);    // 設定要在角落顯示的 Actor  
    widget->SetInteractor(iren);                // 綁定 Interactor  
    widget->SetViewport(0.0, 0.0, 0.2, 0.2);    // 設定角落視埠位置 (xmin, ymin, xmax, ymax) -> 這裡設為左下角 20% 區域  
    widget->SetEnabled(1);                      // 啟用 Widget  
    widget->InteractiveOn();  
  
    iren->Initialize();  
}
```

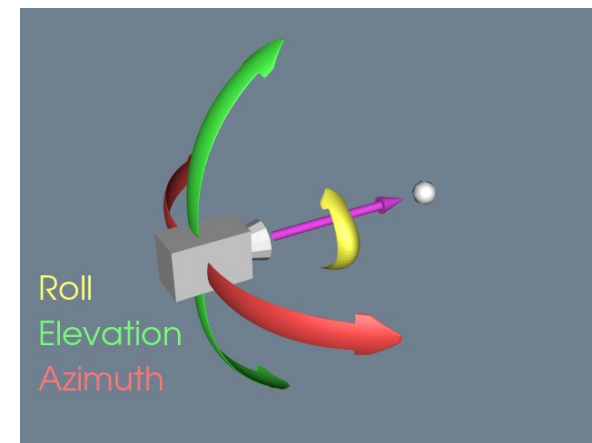
This function help you view the result easily.
Renderer->ResetCamera();



The Camera Frame

這是最基礎但也最重要的概念，相機在 3D 空間中的姿態完全由這三個向量/座標決定：

- Position (相機位置)：相機放在 3D 空間中的哪裡。
 - `camera->SetPosition(x, y, z);`
- Focal Point (焦點/目標點)：相機正在鏡頭瞄準哪裡（通常是場景中心或感興趣的物件）。
 - `camera->SetFocalPoint(x, y, z);`
- View Up (頭頂方向向量)：決定相機的「上下方向」（類似相機拿正還是拿斜）。
 - `camera->SetViewUp(0, 1, 0);`（預設 Y 軸朝上）
- `camera->Azimuth(angle);`：繞著 View Up 軸水平旋轉（方位角）
- `camera->Elevation(angle);`：俯仰旋轉（仰角/俯角）
- `camera->Dolly(value);`：前後推鏡頭（放大/縮小）
- `camera->Zoom(factor);`：調整焦距。



Phong 光照模型 (Phong Lighting Model)

vtkProperty 將光照拆解為三個要素：

- 環境光 (Ambient Light)：沒有明確光源方向的全域基礎光，防止陰影處完全變黑。property->SetAmbient(0.2); (範圍 0.0 ~ 1.0)
- 漫反射光 (Diffuse Light)：主要的散射光，決定物體在主光源照射下的本體顏色與明暗。property->SetDiffuse(0.8);
- 高光/鏡面光 (Specular Light & Power)：受光源與觀察者角度影響的反光點，用來呈現材質感（金屬、塑膠）。property->SetSpecular(0.5);property->SetSpecularPower(30.0); (數值越高，反光點越小越銳利)

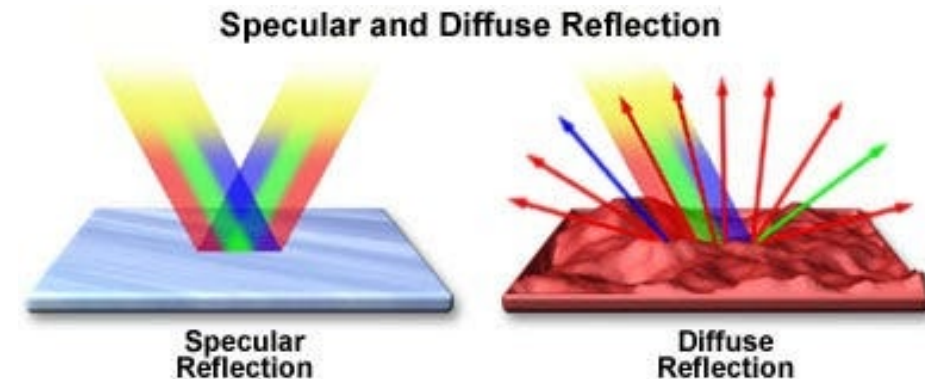


Figure 3

光源類型

vtkLight 的幾種模式 VTK 的光源主要透過 vtkLight 來管理，主要可以教以下三種常見類型：

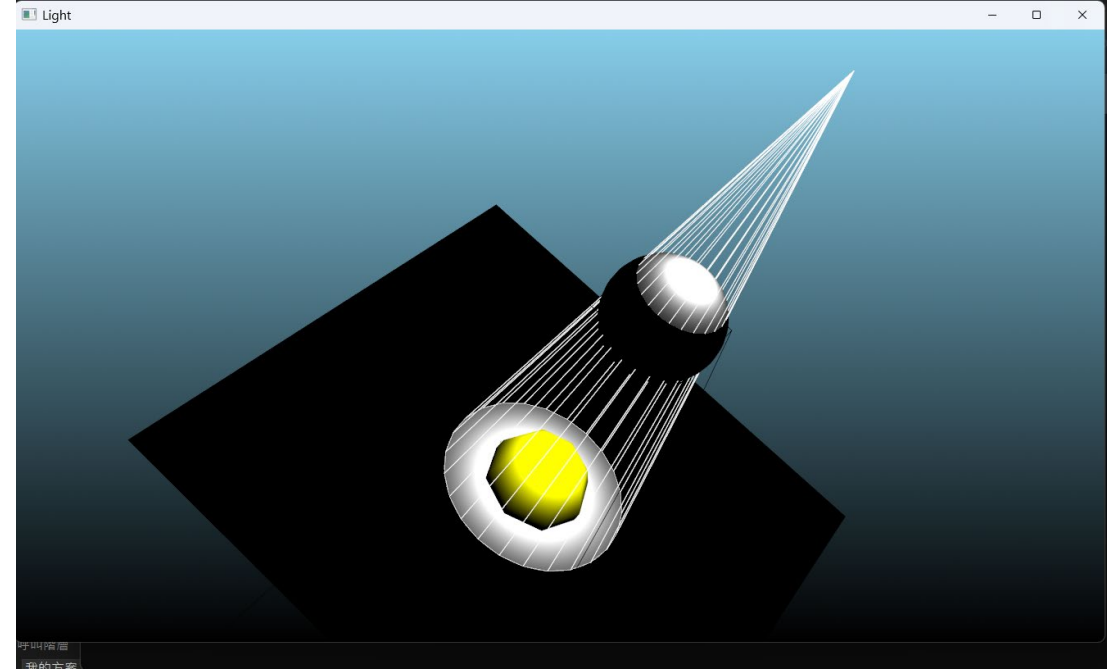
- 頭燈模式 (Headlight / Camera Light) :
 - 預設行為：VTK 的 vtkRenderer 預設就會帶有一盞跟隨相機移動的 Headlight。
 - 特點：永遠從觀測者的角度照向物體，沒有陰影死角。
- 位置光源 / 點光源 (Positional Light) : 設定為空間中的某個特定座標（例如 (x, y, z) ），向四周發散或朝向特定目標（Focal Point）。
 - 啟用方式：light->SetPositional(true);
- 聚光燈 (Spotlight) : 當位置光源開啟後，設定 Cone Angle 就會變成聚光燈。
 - 關鍵設定：light->SetConeAngle(30.0); (錐形開口角度) 與 light->SetExponent(10.0); (衰減係數)。

Turn on the light

```
Void LightTest(){  
    ...  
    renderer->AutomaticLightCreationOff();  
    // On: The headlight is automatically created and added to the renderer.  
    // Off: No light is created.  
  
    double lightPosition[3] = { 0, 0, 1 };  
  
    // Create a light  
    double lightFocalPoint[3] = { 0, 0, 0 };  
  
    vtkNew<vtkLight> light;  
    light->SetLightTypeToSceneLight();  
    light->SetPosition(lightPosition[0], lightPosition[1], lightPosition[2]);  
    light->SetPositional(true); // required for vtkLightActor below  
    light->SetConeAngle(10);  
    light->SetFocalPoint(lightFocalPoint[0], lightFocalPoint[1],  
        lightFocalPoint[2]);  
    light->SetExponent(100.0); // 光的集中度  
    light->SetIntensity(1.0); // 燈光強度  
    // Set the white light color.  
    light->SetColor(colors->GetColor3d("White").GetData());  
    ...  
}
```

```
#include <vtkLight.h>  
#include <vtkLightActor.h>  
#include <vtkLightCollection.h>  
#include <vtkPlaneSource.h>
```

```
Void LightTest(){  
    ...  
    // Display where the light is  
    vtkNew<vtkLightActor> lightActor;  
    lightActor->SetLight(light);  
    renderer->AddViewProp(lightActor);  
  
    // Display where the light is focused  
    vtkNew<vtkSphereSource> lightFocalPointSphere;  
    lightFocalPointSphere->SetCenter(lightFocalPoint);  
    lightFocalPointSphere->SetRadius(0.1);  
    lightFocalPointSphere->Update();  
  
    vtkNew<vtkPolyDataMapper> lightFocalPointMapper;  
    lightFocalPointMapper->SetInputConnection(  
        lightFocalPointSphere->GetOutputPort());  
  
    vtkNew<vtkActor> lightFocalPointActor;  
    lightFocalPointActor->SetMapper(lightFocalPointMapper);  
    lightFocalPointActor->GetProperty()->SetColor(  
        colors->GetColor3d("Yellow").GetData());  
    renderer->AddViewProp(lightFocalPointActor); //加入燈光輔助線  
  
    renderer->AddLight(light);  
  
    ...  
}
```



```
auto camera = renderer->GetActiveCamera();  
camera->SetPosition(-2.17199, -2.50774, 2.18);  
camera->SetFocalPoint(-0.144661, -0.146372, 0.180482);  
camera->SetViewUp(0.0157883, 0.638203, 0.769706);  
camera->SetDistance(3.69921); // 沿著視角方向，將相機與焦點的距離調整成3.69921  
camera->SetClippingRange(1.76133, 6.14753); //距離小於 near 與 距離大於6.14753切掉  
  
renderWindowInteractor->Start();  
}
```

```

void SpotLightTest() {

    // create a rendering window and renderer
    vtkNew<vtkRenderer> renderer;
    vtkNew<vtkRenderWindow> renderWindow;

    renderWindow->SetSize(640, 480);

    renderWindow->AddRenderer(renderer);

    renderWindow->SetWindowName("SpotLights");


    // create a renderwindowinteractor
    vtkNew<vtkRenderWindowInteractor> iren;

    iren->SetRenderWindow(renderWindow);


    // create a plane and add it to the renderer
    vtkNew<vtkPlaneSource> source;

    source->SetOrigin(-5.0, 0.0, 5.0);

    source->SetPoint1(5.0, 0.0, 5.0);

    source->SetPoint2(-5.0, 0.0, -5.0);

    source->SetResolution(100, 100);
    vtkNew<vtkPolyDataMapper> mapper;

    mapper->SetInputConnection(source->GetOutputPort());


    vtkNew<vtkNamedColors> colors;


    vtkNew<vtkActor> actor;

    actor->SetMapper(mapper);

    renderer->AddActor(actor);


    // set camera view

    renderer->ResetCamera();

    renderer->GetActiveCamera()->Azimuth(40.0);

    renderer->GetActiveCamera()->Elevation(30.0);


    // create a green light

```

```

    vtkNew<vtkLight> greenLight;

    greenLight->PositionalOn();

    greenLight->SetPosition(-4.0, 4.0, -1.0);

    greenLight->SetColor(colors->GetColor3d("Lawn_green").GetData());

    greenLight->SetIntensity(0.5);


    // create a pink light
    vtkNew<vtkLight> pinkLight;

    pinkLight->PositionalOn();

    pinkLight->SetPosition(4.0, 5.0, 1.0);

    pinkLight->SetColor(colors->GetColor3d("DeepPink").GetData());

    pinkLight->SetIntensity(0.6);


    // add lights to the renderer
    renderer->AddLight(pinkLight);

    renderer->AddLight(greenLight);


    // add the position of the light
    vtkNew<vtkActor> lightOne, lightTwo;

    CreateObject(lightOne, renderer, ObjectType::Sphere, -4.0, 4.0, -1.0, 1, 0, 0, 0.1);

    CreateObject(lightTwo, renderer, ObjectType::Sphere, 4.0, 5.0, 1.0, 1, 0, 0, 0.1);


    // define a background for better visibility
    renderer->SetBackground(colors->GetColor3d("RoyalBlue").GetData());

    renderer->SetBackground2(colors->GetColor3d("SkyBlue").GetData());

    renderer->GradientBackgroundOn();


    // enable interactor

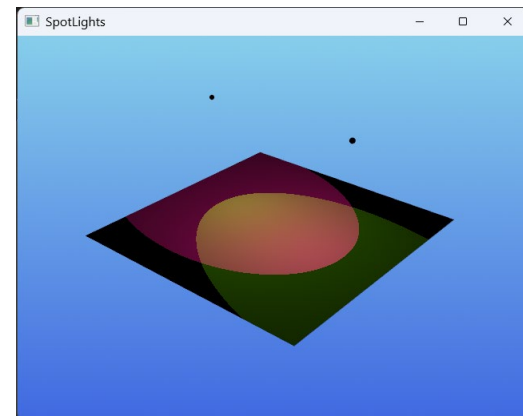
    iren->Initialize();

    renderWindow->Render();

    iren->Start();

}

```



Animation

...

```
vtkNew<vtkRenderWindowInteractor> interactor;  
interactor->SetRenderWindow(renderWindow);
```

```
vtkNew<AnimationCallback> cb;  
cb->earthActor = earthActor;  
cb->moonActor = moonActor;  
cb->moonDistance = 8.0;
```

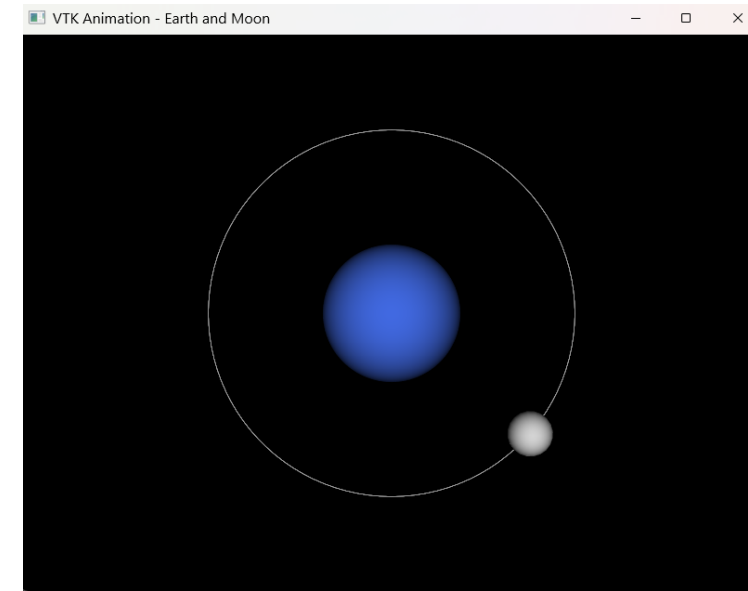
```
// 綁定 Interactor 的 TimerEvent 事件  
interactor->AddObserver(vtkCommand::TimerEvent, cb);
```

```
// 初始化 Interactor 並開啟定時器 (每 20 毫秒觸發一次，約 50 FPS)  
interactor->Initialize();  
interactor->CreateRepeatingTimer(20);
```

```
interactor->Start();
```

```
}
```

```
#include <vtkTransform.h>  
#include <vtkTexture.h>  
#include <vtkRegularPolygonSource.h>  
#include <vtkTubeFilter.h>  
#include <vtkImageReader2Factory.h>  
#include <vtkImageReader2.h>  
#include <vtkTransformTextureCoords.h>  
#include <vtkTexturedSphereSource.h>
```



Animation callback

```
class AnimationCallback : public vtkCommand {

public:

    static AnimationCallback* New() { return new AnimationCallback; }

    // 需綁定的動態物件與參數

    vtkActor* earthActor = nullptr;

    vtkActor* moonActor = nullptr;

    double moonDistance = 10.0; // 月球距離地球的公轉半徑

    double orbitAngle = 0.0;    // 公轉角度

    double earthSpin = 0.0;      // 地球自轉角度


    // 當 Timer 時間到時，VTK 會自動呼叫 Execute

    void Execute(vtkObject* caller, unsigned long eventId, void* callData) override {

        if (eventId == vtkCommand::TimerEvent) {

            // A. 更新角度（模擬時間流逝）

            orbitAngle += 1.0; // 月球公轉速度

            earthSpin += 2.0; // 地球自轉速度（比月球快）


            // B. 地球自轉

            earthActor->SetOrientation(0.0, earthSpin, 0.0);


            // C. 月球公轉 + 自轉矩陣運算 (Transform)

            vtkNew<vtkTransform> moonTransform;

            moonTransform->PostMultiply(); // 確保矩陣相乘順序由左至右（先自轉/位移，再繞原點公轉）


            // 1) 月球自轉

            moonTransform->RotateY(orbitAngle);

            // 2) 推離原點至軌道半徑（放置於 X 軸）

            moonTransform->Translate(moonDistance, 0.0, 0.0);

            // 3) 繞原點（地球）進行 Y 軸公轉

            moonTransform->RotateY(orbitAngle);


            // 將運算後的 Transform 套用至月球 Actor

            moonActor->SetUserTransform(moonTransform);


            // D. 要求視窗重新渲染

            auto interactor = dynamic_cast<vtkRenderWindowInteractor*>(caller);

            if (interactor && interactor->GetRenderWindow()) {
```

動畫是怎麼產生的？

- 視覺暫留原理：動畫本質上是連續播放的靜態圖片（Frames）。
- VTK 中的挑戰：如果用 `while(true) { Rotate(); }`，程式會卡死在迴圈，視窗（GUI）會完全凍結無法互動。
- 解法：事件驅動模型 (Event-Driven)
 - 不要自己寫無限迴圈，而是請「計時器（Timer）」定期叫醒程式。
 - 每隔幾毫秒更新一次物件狀態 → 重新繪製 → 將控制權還給滑鼠。

撰寫你的專屬 Callback

- 繼承 `vtkCommand` :
 - 必須覆寫 `Execute` 函式，這是每次 `Timer` 響起時會執行的內容。
- `Execute` 內的黃金三步驟：
 - 更新數據：改變角度、座標或顏色（例如 `angle += 1`）。
 - 套用變更：呼叫 `actor->RotateY()` 或 `SetUserTransform()`。
 - 強制重繪：呼叫 `renderWindow->Render()`，這是畫面更新的最關鍵步驟！如果忘記寫這行，背景有計算出要移動的位置，但畫面依然不動。

使用 Timer 驅動動畫

- 關鍵物件：`vtkRenderWindowInteractor`
 - 負責監聽使用者的滑鼠、鍵盤操作。
 - 也可以發出定時的「時間訊號」。
- 啟動定時器：
 - `interactor->CreateRepeatingTimer(20);`
 - 意義：每 20 毫秒發出一個 `TimerEvent`，相當於每秒 50 幀（50 FPS）。
- 如何接聽訊號？
 - 使用 Observer Pattern（觀察者模式）。
 - 當 Interactor 發出 `TimerEvent` 時，我們準備好的 `vtkCommand`（Callback）就會被觸發。

HW: Do some animation to the light

