

# Day1

1. What is VTK for?
2. VTK pipe line
3. Show the first Object
4. Property of the Object

# What is Visualization Toolkit for?

VTK is an open-source software system for **image processing, 3D graphics, volume rendering and visualization**. VTK includes many advanced algorithms (e.g., surface reconstruction, implicit modelling, decimation) and rendering techniques (e.g., hardware-accelerated volume rendering, LOD control).

VTK is used by academicians for teaching and research; by government research institutions such as Los Alamos National Lab in the US or CINECA in Italy; and by many commercial firms who use VTK to build or extend products.

The origin of VTK is with the textbook "The Visualization Toolkit, an Object-Oriented Approach to 3D Graphics" originally published by Prentice Hall and now published by Kitware, Inc. (Third Edition ISBN 1-930934-07-6). VTK has grown (since its initial release in 1994) to a world-wide user base in the commercial, academic, and research communities. A more recent PDF version of the textbook is available from the link below.

# VTK VS. PCL

| Feature              | VTK (Visualization Toolkit)                                                                         | PCL (Point Cloud Library)                                                                                         |
|----------------------|-----------------------------------------------------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------|
| Primary Focus        | General 3D visualization, scientific data rendering, and volume rendering.                          | 3D geometry processing and <b>point cloud manipulation</b> .                                                      |
| Core Data Structures | Structured/unstructured grids, polygons, volumes, and meshes.                                       | 2D/3D point clouds ( <code>pcl::PointCloud&lt;T&gt;</code> ), octrees, and Kd-trees.                              |
| Key Functionalities  | Rendering pipelines, isosurface extraction, volume rendering, interactive widgets, multi-viewports. | Point cloud registration (ICP), segmentation, filtering, surface reconstruction, feature extraction (e.g., FPFH). |
| Hardware Integration | Focuses on GPUs for display rendering (OpenGL/VTK rendering pipeline).                              | Interfaces heavily with 3D sensors (e.g., LiDAR, RGB-D cameras, OpenNI).                                          |
| PCL Integration      | VTK serves as the underlying rendering engine for <code>pcl::visualization</code> .                 | Uses VTK internally to power visualizers like <code>pcl::visualization::PCLVisualizer</code> .                    |
| Best Use Case        | Medical imaging (CT/MRI), CFD/engineering visualization, CAD model rendering.                       | Autonomous driving, robotics perception, 3D scanning, SLAM, and spatial analysis.                                 |

# First example

```
#pragma once
#define _SILENCE_CXX17_ITERATOR_BASE_CLASS_DEPRECATION_WARNING
#include <vtkAutoInit.h>
VTK_MODULE_INIT(vtkRenderingOpenGL2);
VTK_MODULE_INIT(vtkInteractionStyle);

#include <vtkActor.h>
#include <vtkCamera.h>
#include <vtkConeSource.h>
#include <vtkNamedColors.h>
#include <vtkNew.h>
#include <vtkPolyDataMapper.h>
#include <vtkProperty.h>
#include <vtkRenderWindow.h>
#include <vtkRenderer.h>
```

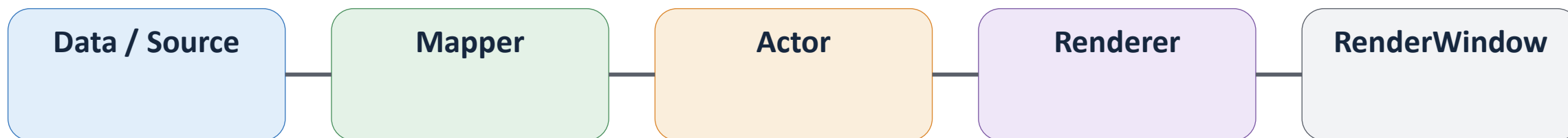
```
void createCone() {  
    vtkNew<vtkNamedColors> colors;  
    vtkNew<vtkConeSource> cone;  
    cone->SetHeight(3.0);  
    cone->SetRadius(1.0);  
    cone->SetResolution(10);  
  
    vtkNew<vtkPolyDataMapper> coneMapper;  
    coneMapper->SetInputConnection(cone->GetOutputPort());  
  
    vtkNew<vtkActor> coneActor;  
    coneActor->SetMapper(coneMapper);  
    coneActor->GetProperty()->SetColor(colors->GetColor3d("MistyRose").GetData());  
  
    vtkNew<vtkRenderer> ren1;  
    ren1->AddActor(coneActor);  
    ren1->SetBackground(colors->GetColor3d("MidnightBlue").GetData());  
  
    vtkNew<vtkRenderWindow> renWin;  
    renWin->AddRenderer(ren1);  
    renWin->SetSize(800, 600);  
    renWin->SetWindowName("Tutorial_Step1");  
  
    for (int i = 0; i < 1000; ++i)  
    {  
        renWin->Render();  
        ren1->GetActiveCamera()->Azimuth(1);  
    }  
}
```

# vtkPolyData

1. 概念比喻：它就是 3D 世界裡的「外皮 (Skin / Surface)」
  - 想像一下 3D 模型。vtkImageData 像是 3D 的像素方塊（如 Minecraft），vtkUnstructuredGrid 像是內部充滿結構的實心實體（如 CT 掃描的器官內部）。而 vtkPolyData 就是物體的最外層皮膚！無論是 3D 遊戲角色、CAD 車殼，還是點雲，都是用 vtkPolyData 組合出來的空心外皮。
2. 結構拆解：vtkPolyData 的三大核心元素
  - 一個 vtkPolyData 內部其實只由三樣東西組成：
    1. Points（點）：vtkPoints3D 空間中的 (x, y, z) 座標陣列。這是所有幾何的基底。
    2. Cells（單元 / 拓撲關係）：vtkCellArray 告訴 VTK「這些點要怎麼連起來」。例如：點 0 和點 1 連成線，或者點 0, 1, 2 連成三角形。
    3. Point / Cell Data（屬性數據）：vtkPointData / vtkCellData 附加在點或面上的資訊，例如顏色、法線向量（Normals）、溫度、壓力、UV 貼圖座標等。

# VTK Visualization Pipeline

從 vtkConeSource 到螢幕：理解 VTK 每個物件的角色與關係



核心觀念：資料怎麼產生 → 怎麼轉成可繪製內容 → 怎麼放進場景 → 怎麼顯示  
VTK 的強項不是單一繪圖函式，而是 Pipeline + Rendering 架構

# 1. 完整 Pipeline：你的程式到底在做什麼？



程式中的 `SetInputConnection()`：把前一個 Pipeline object 的輸出接到下一個 object。

程式中的 `SetMapper()` / `AddActor()` / `AddRenderer()`：則是在建立 Rendering Scene 的關係。

可以把它想成：工廠流水線 + 最後的展示場。

## 2. vtkConeSource：資料的來源 Source

對應程式：`vtkNew<vtkConeSource> cone;`



`SetHeight(3.0)`：幾何體高度

`SetRadius(1.0)`：底面半徑

`SetResolution(10)`：圓周方向的離散程度

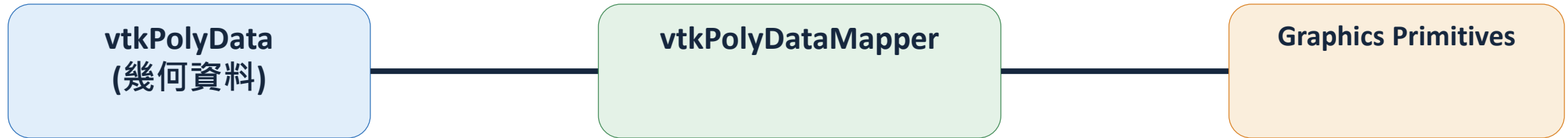
`GetOutputPort()`：取得 Source 的輸出端口，供下一個 Pipeline object 使用

**Source 的任務：產生資料  
不是負責把東西顯示在螢幕上**

**可替換的 Source**  
**Sphere / Cube / Plane / OBJ Reader / Point Cloud**

### 3. vtkPolyDataMapper：資料 → 可繪製的幾何

對應程式：`coneMapper->SetInputConnection(cone->GetOutputPort());`



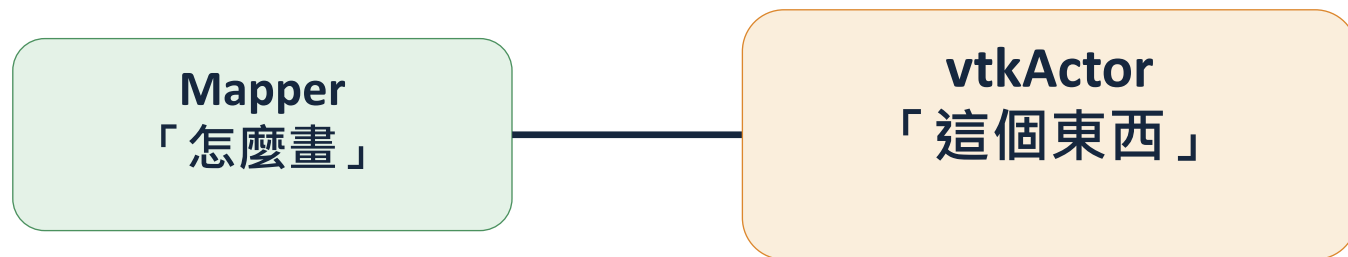
Mapper 可以理解成「翻譯器」：把 VTK Data 轉成 Rendering 可以使用的形式。

它通常不決定物件在世界中的位置，也不負責背景、Camera。

Pipeline 可以插入 Filter：Source → Filter → Mapper。

## 4. vtkActor：場景中的「物件」

對應程式：`coneActor->SetMapper(coneMapper);`



`SetMapper()`：指定 Actor 要使用哪個 Mapper。

`GetProperty()`：控制顏色、透明度、點大小、線寬、Shading 等。

Actor 也有自己的 Transformation：  
Position / Rotation / Scale。

同一個 Mapper 可以被多個 Actor 使用，讓同一份幾何資料出現在不同位置。

**Mapper ≠ Actor**  
Mapper 偏向「繪製資料」；Actor 偏向「場景中的實體與外觀/姿態」

## 5. vtkProperty : 外觀 Appearance

對應程式：`coneActor->GetProperty()->SetColor(...);`

**Color**

**Opacity**

**Point Size**

**Line Width**

**Representation**

**Shading**

Representation : Points / Wireframe / Surface

Shading : Flat / Gouraud / Phong 等

Opacity 可以做透明 Mesh ; Point Size / Line Width 對 Point Cloud 和線段很常用。

NamedColors 只是方便取得顏色，不是 Rendering Pipeline 的核心。

## 6. vtkRenderer : 一個 3D 場景 / Viewport

**vtkRenderer**

AddActor(coneActor) : 把物件加入場景。

SetBackground() : 設定背景。

GetActiveCamera() : 取得目前 Camera。

可以有多個 Renderer，組成多個 Viewport。

**Front**

**Top**

**Side**

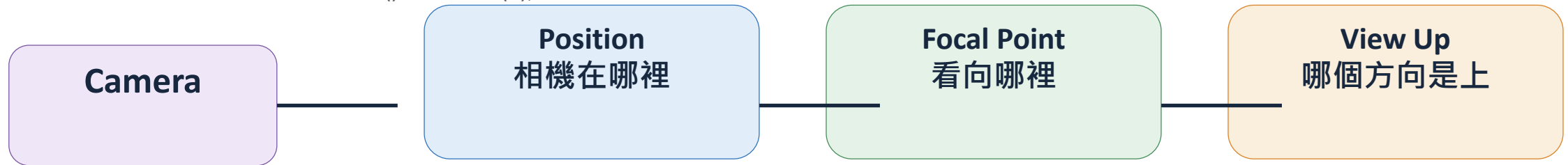
**3D**

Renderer 是「場景管理者」，不是整個 OS 視窗。

RenderWindow 可以包含一個或多個 Renderer。

## 7. vtkCamera：決定「你從哪裡看」

你的程式使用：`ren1->GetActiveCamera()->Azimuth(1);`



`Azimuth()`：繞場景的垂直方向旋轉 Camera。

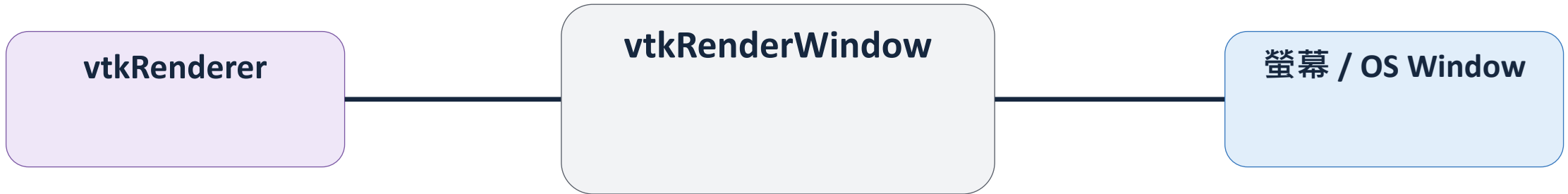
`Elevation()`：上下改變視角。

`Zoom()` / `Dolly()`：改變觀看尺度或相機位置。

`ResetCamera()`：讓 `Renderer` 自動調整 Camera，使物件進入視野。

重要觀念：旋轉 Camera  $\neq$  旋轉 Cone  
你目前的程式是在轉「觀看角度」，不是轉 Actor。

## 8. vtkRenderWindow : 真正的顯示視窗



`AddRenderer(ren1)` : 把 `Renderer` 放進 `RenderWindow` 。

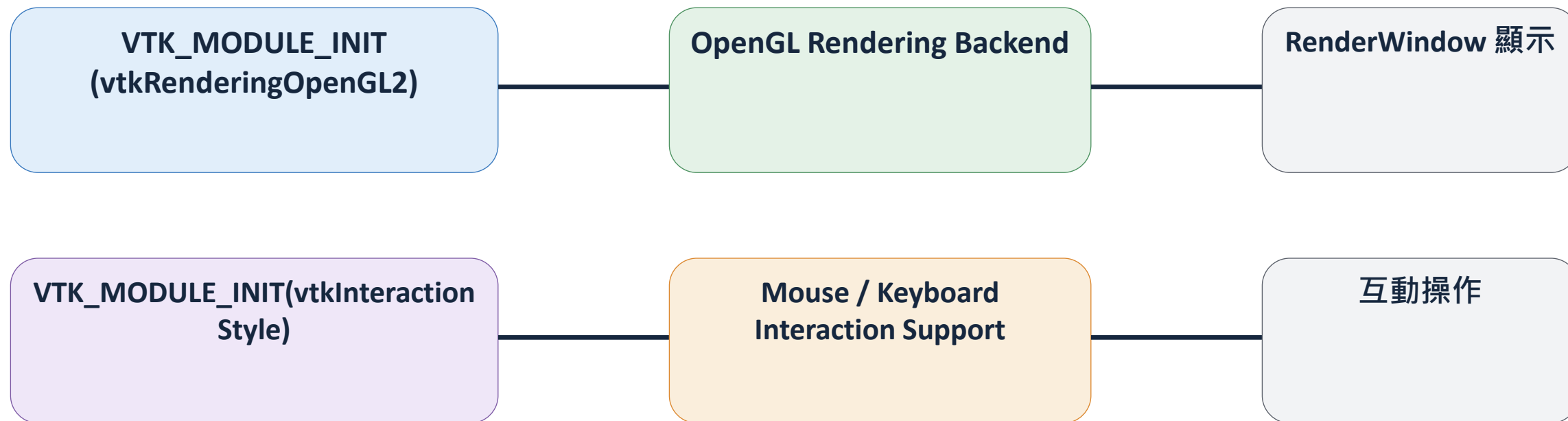
`SetSize(800, 600)` : 設定視窗尺寸。

`SetWindowName()` : 設定視窗名稱。

`Render()` : 要求 VTK 進行一次渲染。

你的程式目前用 `RenderWindow + Render()`，並沒有顯式建立 `Interactor`。  
`VTK_MODULE_INIT(...)` 負責初始化 `RenderingOpenGL2 / InteractionStyle` 模組。

## 9. VTK\_MODULE\_INIT：為什麼你加了它之後就能正常跑？



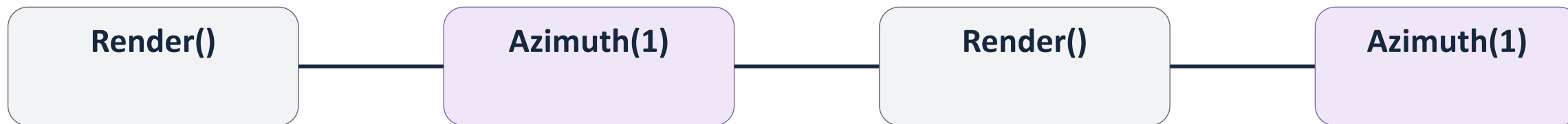
這兩行不是「建立一個 VTK 物件」，而是在舊式 / 模組化 VTK 建置中顯式初始化模組。

RenderingOpenGL2：告訴 VTK 使用 OpenGL rendering backend。

InteractionStyle：提供互動風格相關模組。

不同 VTK 版本 / CMake 設定可能需要不同的模組初始化方式。

## 10. 你的 for 迴圈：其實是在做 Camera Animation



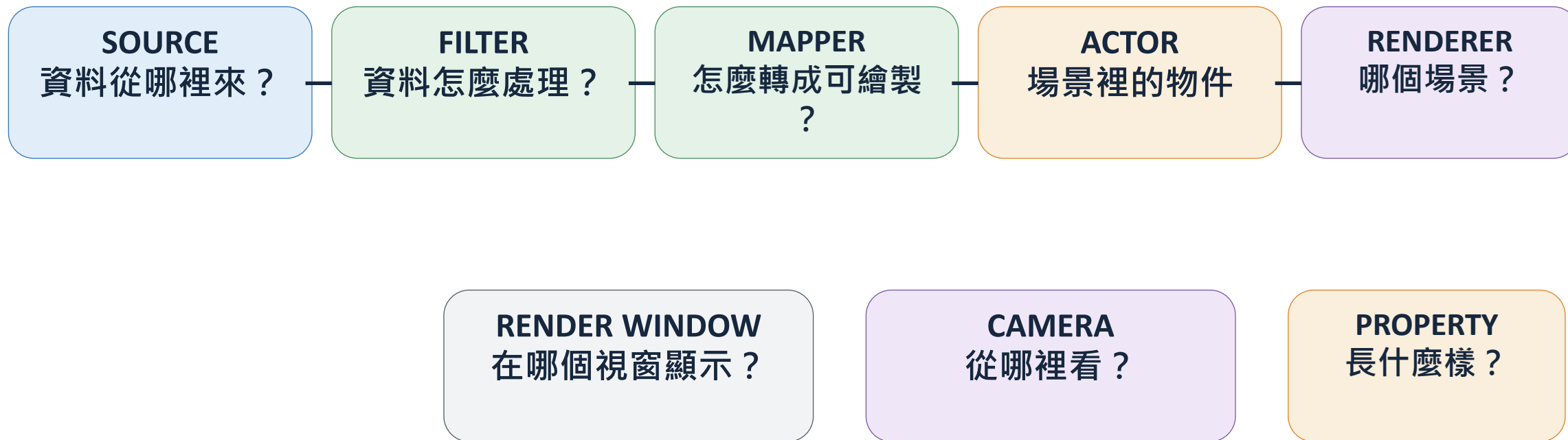
每一次迴圈：先畫一次，再把 Active Camera 水平旋轉  $1^\circ$ 。

10000 次迴圈理論上會旋轉很多圈，但程式執行很快，使用者不一定看得到平滑動畫。

如果要做真正的互動式 / 即時動畫，通常會加入 `RenderWindowInteractor + Timer`。

這也是從「靜態 Visualization」進入「Real-time Visualization」的入口。

## 12. 建議學生記住的 VTK 心智模型



一句話：VTK = 用 Pipeline 把「資料」一路送到「3D 場景與螢幕」。

Source / Filter 偏向資料；Mapper / Actor / Renderer / Camera 偏向 Visualization。

Property 是外觀；RenderWindow 是容器；Interactor 則負責使用者互動。

掌握這個架構後，再學其他 vtkXXX class 會容易很多。

# 牛刀小試1

```
#define  
_SILENCE_CXX17_ITERATOR_BASE_CLASS_DEPRECATION_WARNING  
#include <vtkAutoInit.h>  
  
VTK_MODULE_INIT(vtkRenderingOpenGL2);  
VTK_MODULE_INIT(vtkInteractionStyle);  
  
#include <vtkActor.h>  
#include <vtkCamera.h>  
#include <vtkConeSource.h>  
#include <vtkSphereSource.h>  
#include <vtkCubeSource.h>  
#include <vtkNamedColors.h>  
#include <vtkNew.h>  
#include <vtkPolyDataMapper.h>  
#include <vtkProperty.h>  
#include <vtkRenderWindow.h>
```

```
#include <vtkRenderer.h>  
#include <vtkRenderWindowInteractor.h>  
  
enum class ObjectType {  
    Cone,  
    Sphere,  
    Cube  
};  
  
// According to different object type to show  
different object  
void CreateObject(ObjectType objectType){  
    ...  
}
```



| 功能                           | VTK 函式                                 | 常用        |
|------------------------------|----------------------------------------|-----------|
| 顏色                           | SetColor()                             | ★ ★ ★ ★ ★ |
| 透明度                          | SetOpacity()                           | ★ ★ ★ ★ ★ |
| Surface / Wireframe / Points | SetRepresentation...()                 | ★ ★ ★ ★ ★ |
| Point Size                   | SetPointSize()                         | ★ ★ ★ ★ ★ |
| Line Width                   | SetLineWidth()                         | ★ ★ ★ ★ ★ |
| Edge 顯示                      | SetEdgeVisibility()                    | ★ ★ ★ ★   |
| Edge Color                   | SetEdgeColor()                         | ★ ★ ★ ★   |
| Vertex 顯示                    | SetVertexVisibility()                  | ★ ★ ★     |
| Shading                      | SetInterpolationToFlat/Gouraud/Phong() | ★ ★ ★ ★   |
| Lighting                     | SetLighting()                          | ★ ★ ★     |
| Ambient                      | SetAmbient()                           | ★ ★ ★     |
| Diffuse                      | SetDiffuse()                           | ★ ★ ★     |
| Specular                     | SetSpecular()                          | ★ ★ ★     |
| Specular Power               | SetSpecularPower()                     | ★ ★ ★     |
| PBR                          | SetInterpolationToPBR()                | ★ ★       |
| Metallic                     | SetMetallic()                          | ★ ★       |
| Roughness                    | SetRoughness()                         | ★ ★       |
| Backface Culling             | SetBackfaceCulling()                   | ★ ★       |

# Change the actor color

| HTML name       | Decimal code |     |     |
|-----------------|--------------|-----|-----|
|                 | R            | G   | B   |
| Red colors      |              |     |     |
| IndianRed       | 205          | 92  | 92  |
| LightCoral      | 240          | 128 | 128 |
| Salmon          | 250          | 128 | 114 |
| DarkSalmon      | 233          | 150 | 122 |
| LightSalmon     | 255          | 160 | 122 |
| Red             | 255          | 0   | 0   |
| Crimson         | 220          | 20  | 60  |
| FireBrick       | 178          | 34  | 34  |
| DarkRed         | 139          | 0   | 0   |
| Pink colors     |              |     |     |
| Pink            | 255          | 192 | 203 |
| LightPink       | 255          | 182 | 193 |
| HotPink         | 255          | 105 | 180 |
| DeepPink        | 255          | 20  | 147 |
| MediumVioletRed | 199          | 21  | 133 |
| PaleVioletRed   | 219          | 112 | 147 |
| Orange colors   |              |     |     |

| HTML name         | Decimal code |     |     |
|-------------------|--------------|-----|-----|
|                   | R            | G   | B   |
| Green colors      |              |     |     |
| GreenYellow       | 173          | 255 | 47  |
| Chartreuse        | 127          | 255 | 0   |
| LawnGreen         | 124          | 252 | 0   |
| Lime              | 0            | 255 | 0   |
| LimeGreen         | 50           | 205 | 50  |
| PaleGreen         | 152          | 251 | 152 |
| LightGreen        | 144          | 238 | 144 |
| MediumSpringGreen | 0            | 250 | 154 |
| SpringGreen       | 0            | 255 | 127 |
| MediumSeaGreen    | 60           | 179 | 113 |
| SeaGreen          | 46           | 139 | 87  |
| ForestGreen       | 34           | 139 | 34  |
| Green             | 0            | 128 | 0   |
| DarkGreen         | 0            | 100 | 0   |
| YellowGreen       | 154          | 205 | 50  |
| OliveDrab         | 107          | 142 | 35  |
| Olive             | 128          | 128 | 0   |

| HTML name      | Decimal code |     |     |
|----------------|--------------|-----|-----|
|                | R            | G   | B   |
| Brown colors   |              |     |     |
| Cornsilk       | 255          | 248 | 220 |
| BlanchedAlmond | 255          | 235 | 205 |
| Bisque         | 255          | 228 | 196 |
| NavajoWhite    | 255          | 222 | 173 |
| Wheat          | 245          | 222 | 179 |
| BurlyWood      | 222          | 184 | 135 |
| Tan            | 210          | 180 | 140 |
| RosyBrown      | 188          | 143 | 143 |
| SandyBrown     | 244          | 164 | 96  |
| Goldenrod      | 218          | 165 | 32  |
| DarkGoldenrod  | 184          | 134 | 11  |
| Peru           | 205          | 133 | 63  |
| Chocolate      | 210          | 105 | 30  |
| SaddleBrown    | 139          | 69  | 19  |
| Sienna         | 160          | 82  | 45  |
| Brown          | 165          | 42  | 42  |
| Maroon         | 128          | 0   | 0   |

<https://examples.vtk.org/site/ColorNamesSeries/ColorNamePatches.html>

# Change the actor color, capacity, and position

```
void createCone(double R, double G, double B) {  
    vtkNew<vtkNamedColors> colors;  
  
    vtkNew<vtkConeSource> cone;  
    cone->SetHeight(3.0);  
    cone->SetRadius(1.0);  
    cone->SetResolution(10);  
  
    vtkNew<vtkPolyDataMapper> coneMapper;  
    coneMapper->SetInputConnection(cone->GetOutputPort());  
  
    vtkNew<vtkActor> coneActor;  
    coneActor->SetMapper(coneMapper);  
    //coneActor->GetProperty()->SetColor(colors->GetColor3d("MistyRose").GetData());  
    coneActor->GetProperty()->SetColor(R, G, B); // 0.0 ~ 1.0  
    coneActor->GetProperty()->SetOpacity(0.5); // Set the opacity to 50%  
    coneActor->SetPosition(2,0,0);  
    ...  
}
```

# Representation

```
// Set the representation to surface
```

```
coneActor->GetProperty()-> SetRepresentationToSurface();
```

```
// Set the representation to Points
```

```
coneActor->GetProperty()-> SetRepresentationToPoints();
```

```
coneActor->GetProperty()->SetPointSize(5); // Set the point size
```

```
// Set the representation to wireframe
```

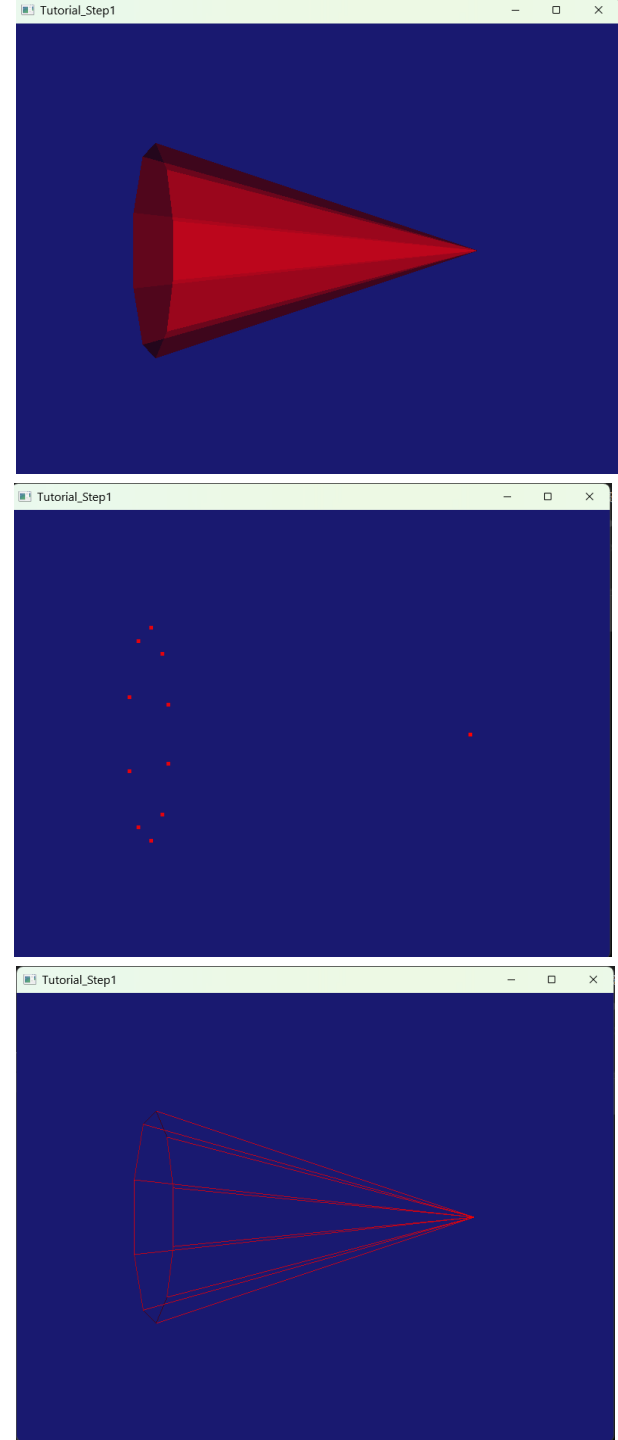
```
coneActor->GetProperty()-> SetRepresentationToWireframe();
```

```
coneActor->GetProperty()->SetLineWidth(4.0);
```

OR

```
coneActor->GetProperty()-> SetEdgeVisibility(true);
```

```
coneActor->GetProperty()-> SetEdgeColor(1.0, 1.0, 1.0);
```

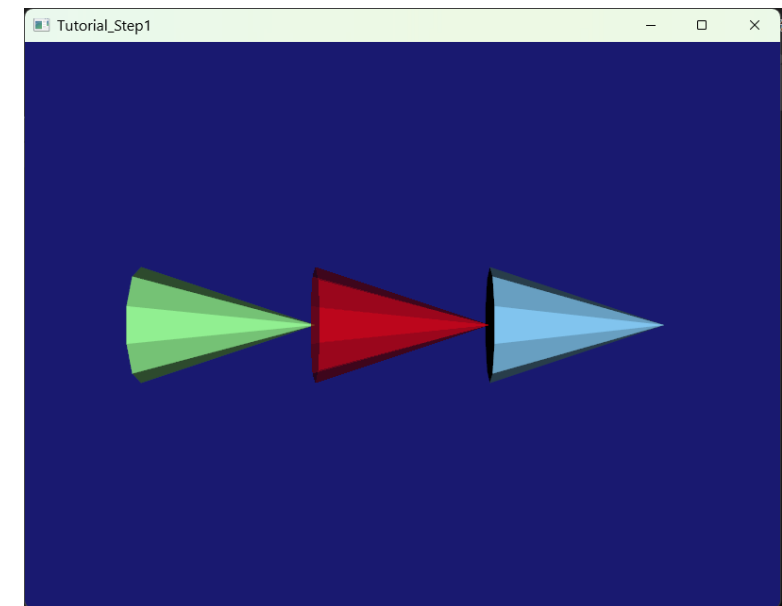


# Practice2 : Multi Actor

```
// create three actor and set their mapper  
// change the property using the method we just learned
```

```
void CreateObject(vtkActor* actor, vtkRenderer* render, ObjectType objectType, double X, double Y,  
double Z, double R, double G, double B, double radius);
```

```
Void multiActor(){  
    vtkNew<vtkActor> coneActor, cA2, cA3;  
    vtkNew<vtkRender> renderer;  
    CreateObject(...);  
    CreateObject(...);  
    CreateObject(...);  
    ...  
}
```



# vtkNew & smartPointer

在 VTK（Visualization Toolkit）中，**vtkNew** 與 **vtkSmartPointer** 都是用來自動管理 VTK 物件生命週期（記憶體管理）的智慧型指標，目的是避免記憶體洩漏（Memory Leak）與手動呼叫 **Delete()** / **UnRegister()**。兩者最大的差異在於「所有權與語意」以及「物件何時被建立」。

| 特性           | <b>vtkNew&lt;T&gt;</b>                                    | <b>vtkSmartPointer&lt;T&gt;</b>                                        |
|--------------|-----------------------------------------------------------|------------------------------------------------------------------------|
| 主要定位         | 區域變數、RAII 生命週期（類似 C++ <code>std::unique_ptr</code> 或物件實例） | 共享所有權、傳入/傳出函式（類似 C++ <code>std::shared_ptr</code> ）                    |
| 物件建立時間       | 宣告時自動建立物件                                                 | 可設為 <code>nullptr</code> ，需透過 <code>.New()</code> 或由其他地方賦值             |
| 指標遞減 (Reset) | ✗ 不能重定向指向別的物件，也不能賦值為 <code>nullptr</code>                 | ○ 可以隨時指向新物件或設為 <code>nullptr</code>                                    |
| 成員存取         | 使用 <code>-&gt;</code> 或 <code>Get()</code>                | 使用 <code>-&gt;</code> 或 <code>Get()</code> / <code>GetPointer()</code> |
| 作為函式傳回值      | ✗ 不適合（會移轉或毀損 RAII 語意）                                     | ○ 非常適合，用於跨範疇傳輸物件                                                       |

# Finished the function CreateObject()

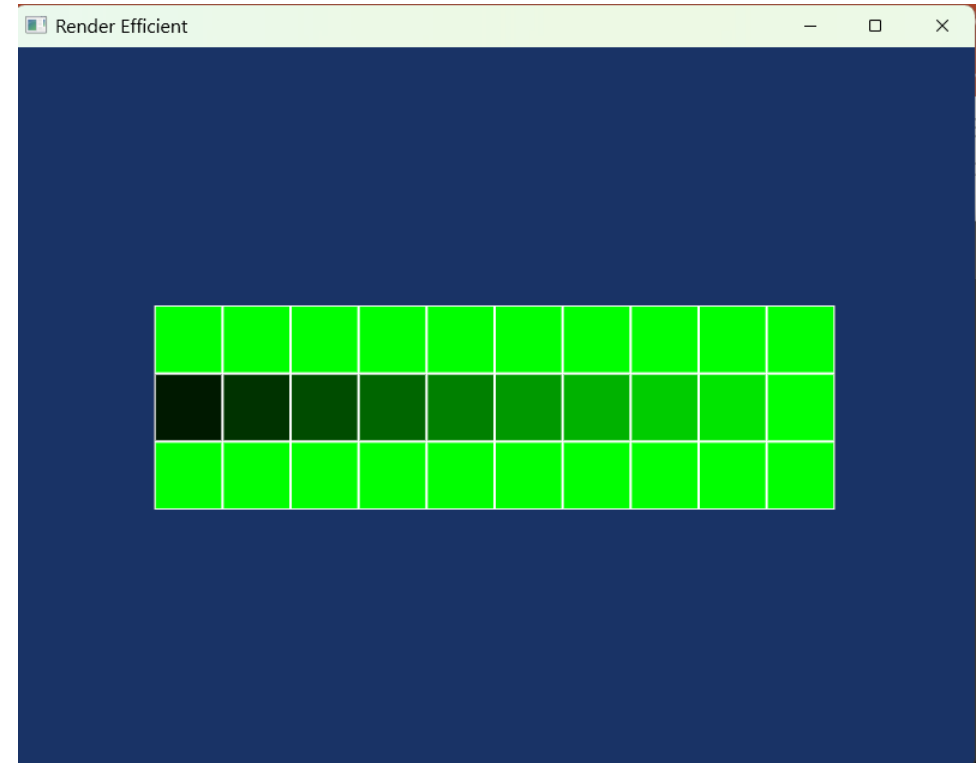
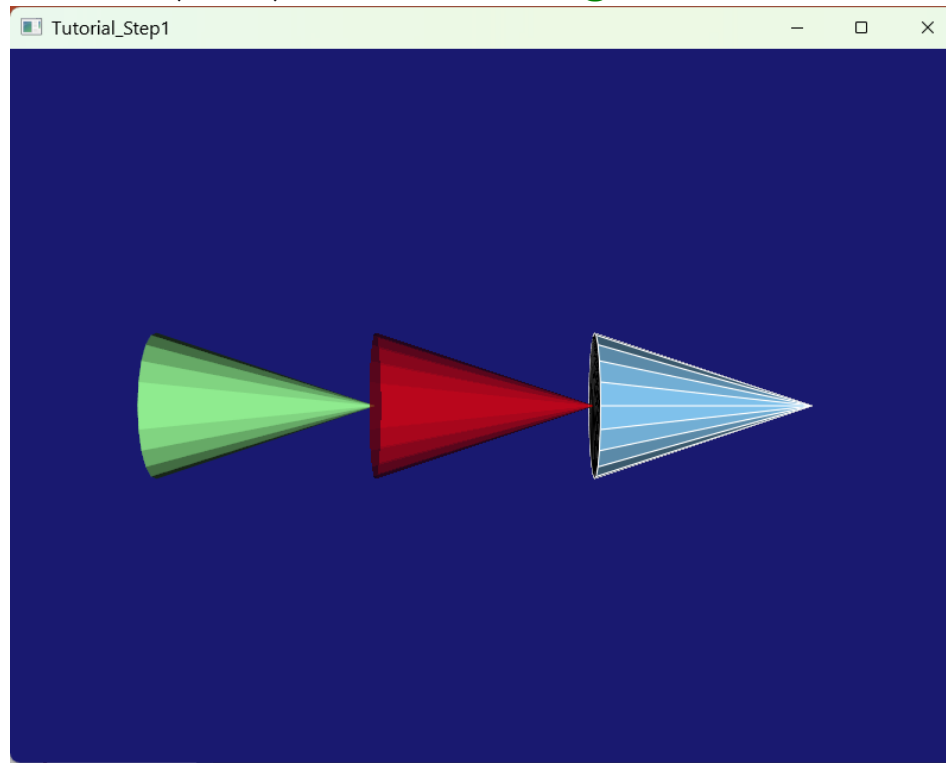
```
void RenderEfficient() {  
  
    vtkNew<vtkRenderer> ren1;  
    ren1->SetBackground(0.1, 0.2, 0.4); // Dark    blue background  
  
    vtkSmartPointer<vtkActor> actor =    vtkSmartPointer<vtkActor>::New();  
    CreateObject(actor, ren1, ObjectType::Cube, i * 2.0, 2.0 * c, 0.0, 0.0, 1.0, 0.0);  
  
    vtkNew<vtkRenderWindow> renWin;  
    renWin->AddRenderer(ren1);  
    renWin->SetSize(800, 600);  
    renWin->SetWindowName("Render Efficient");  
    for (int i = 0; i < 1000; ++i) {  
        renWin->Render();  
    }  
}
```

# Finished the function CreateObject()

```
void RenderEfficient() {  
  
    vtkNew<vtkRenderer> ren1;  
    ren1->SetBackground(0.1, 0.2, 0.4); // Dark blue background  
  
    std::vector<vtkSmartPointer<vtkActor>> actors;  
    int numObjects = 10;  
    int column = 3;  
    for (int c = 0; c < column; ++c) {  
        for (int i = 0; i < numObjects; ++i) {  
            vtkSmartPointer<vtkActor> actor = vtkSmartPointer<vtkActor>::New();  
            CreateObject(actor, ren1, ObjectType::Shpere, i * 2.0, 2.0 * c, 0.0, 0.0, 1.0, 0.0);  
            actors.pushback(actor);  
        }  
    }  
  
    vtkNew<vtkRenderWindow> renWin;  
    renWin->AddRenderer(ren1);  
    renWin->SetSize(800, 600);  
    renWin->SetWindowName("Render Efficient");  
    for (int i = 0; i < 1000; ++i) {  
        renWin->Render();  
    }  
}
```

# The edge

```
vtkProperty* property = actor->GetProperty();  
property->SetEdgeVisibility(true);  
property->SetEdgeColor(1.0, 1.0, 1.0); // White edges  
property->SetLineWidth(2.0); // Set edge line width
```



# 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)

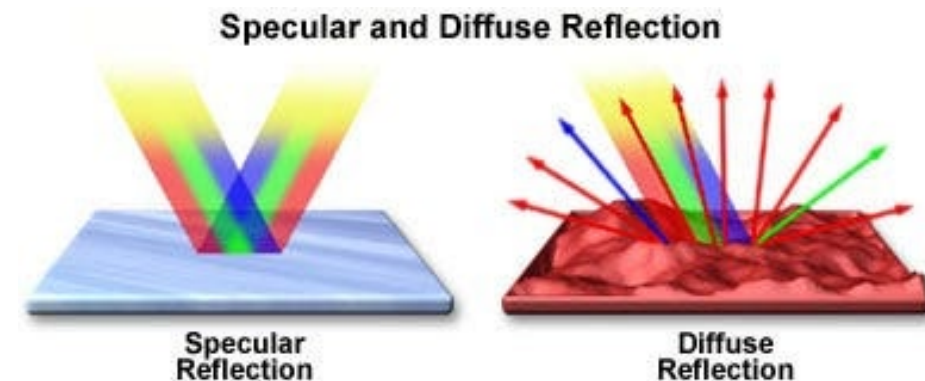


Figure 3

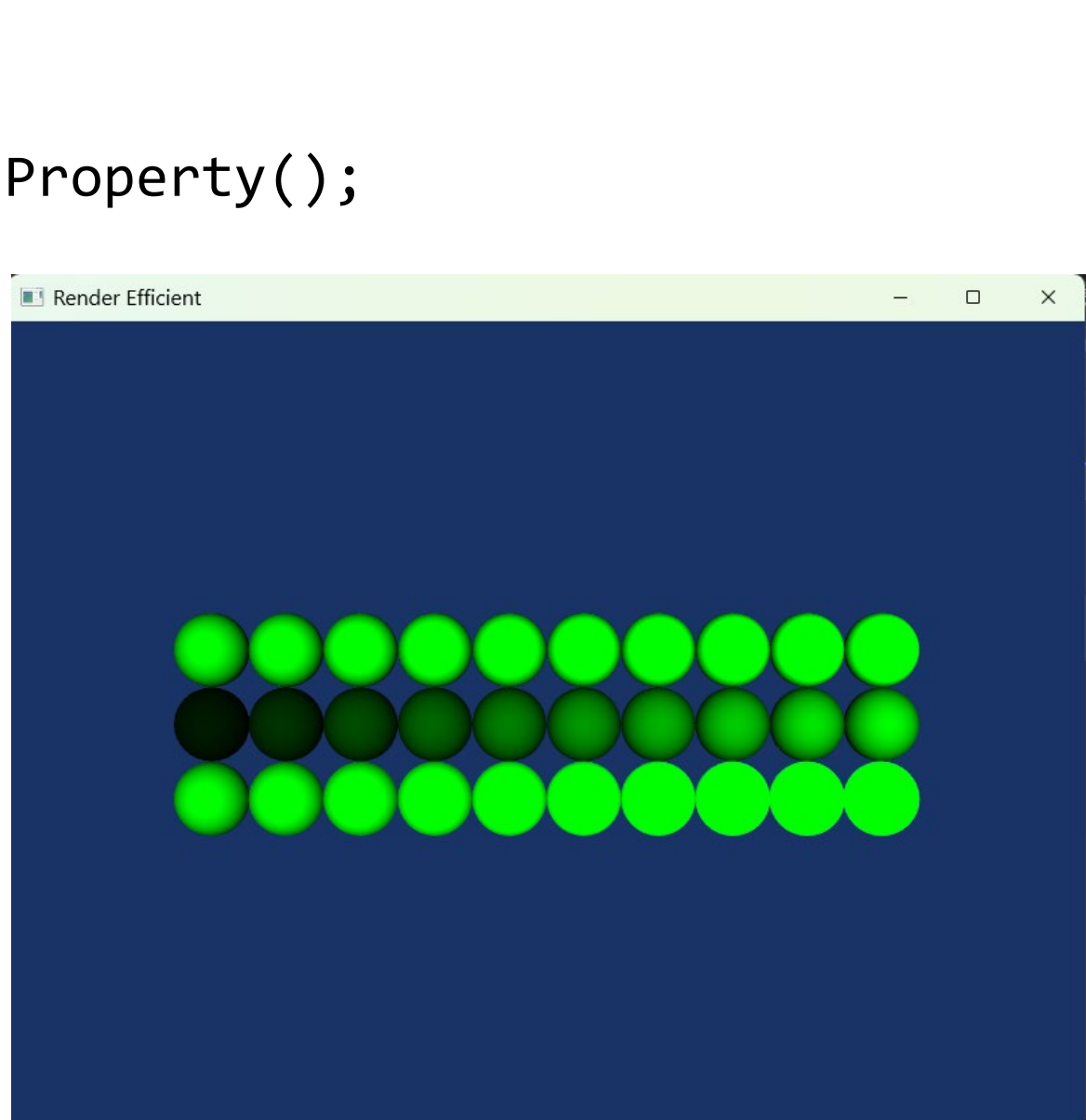
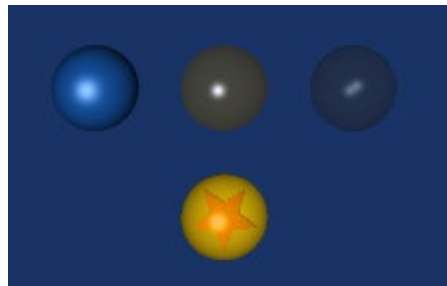
```
vtkProperty* property = actor->GetProperty();
```

```
Row1: property->SetAmbient(scale);
```

```
Row2: property->SetDiffuse(scale);
```

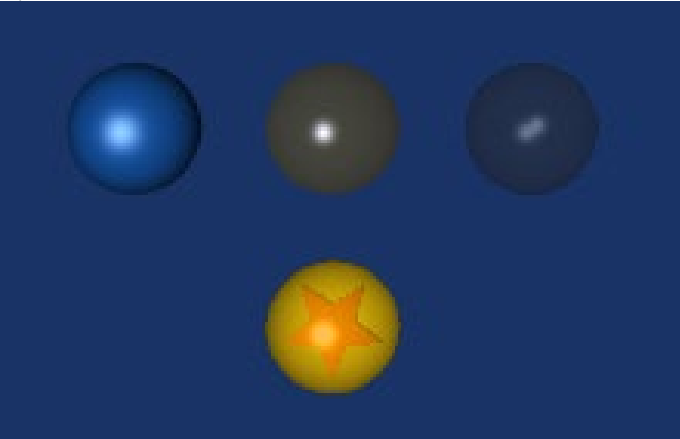
```
Row3: property->SetSpecular(scale)
```

```
//scale: 0.0 ~ 1.0
```



# HW Day1 材質測試

| 材質  | Color (底色) | Diffuse (漫反射) | Specular (高光) | SpecularPower (高光大小) | Opacity (不透明度) | 特點說明                     |
|-----|------------|---------------|---------------|----------------------|----------------|--------------------------|
| 塑膠  | 任意色        | 0.7           | 0.5 (純白)      | 30                   | 1.0            | 亮點適中，高光恆為白色。             |
| 鐵球  | 鐵灰色        | 0.1           | 0.9 (金屬色)     | 100                  | 1.0            | 漫反射低，高光極強且帶金屬色。          |
| 玻璃  | 淡藍/白色      | 0.1           | 1.0 (純白)      | 120                  | 0.3            | 開啟透明度，表面有強烈微小亮點。         |
| 七龍珠 | 橘黃色        | 0.6           | 0.8 (暖白色)     | 80                   | 0.75           | 高 Ambient 營造晶瑩發光感 + 半透明。 |



```
vtkSmartPointer<vtkPolyData> Create3DStarData(double  
outerRadius, double innerRadius, double thickness) {
```

```
    vtkNew<vtkPoints> points;  
    vtkNew<vtkCellArray> polys;
```

```
    double pi = 3.14159265358979323846;
```

```
    // -----  
    // 1. 建立頂點 (Points)  
    // -----
```

```
    // 正面中心點 (Index 0) 與 背面中心點 (Index 1)
```

```
    points->InsertNextPoint(0.0, 0.0, thickness / 2.0);  
    // 0: Front Center
```

```
    points->InsertNextPoint(0.0, 0.0, -thickness / 2.0);  
    // 1: Back Center
```

```
    // 正面 10 個邊緣頂點 (Index 2 ~ 11)
```

```
    for (int i = 0; i < 10; ++i) {
```

```
        double angle = i * pi / 5.0 - pi / 2.0; // 正上方  
        // 為第一個頂角
```

```
        double r = (i % 2 == 0) ? outerRadius :  
innerRadius;
```

```
        points->InsertNextPoint(r * cos(angle), r *  
sin(angle), thickness / 2.0);  
    }
```

```
    // 背面 10 個邊緣頂點 (Index 12 ~ 21)
```

```
    for (int i = 0; i < 10; ++i) {
```

```
        double angle = i * pi / 5.0 - pi / 2.0;
```

```
        double r = (i % 2 == 0) ? outerRadius :  
innerRadius;
```

```
        points->InsertNextPoint(r * cos(angle), r *
```

```
sin(angle), -thickness / 2.0);  
    }
```

```
    // -----  
    // 2. 建立正面與背面的三角形面 (使用 Centroid 放射狀分割)  
    // -----
```

```
    for (int i = 0; i < 10; ++i) {  
        int next = (i + 1) % 10;
```

```
        // -----  
        // A. 正面三角形 (點順序逆時針，法線朝外 +Z)
```

```
        vtkNew<vtkTriangle> frontTri;  
        frontTri->GetPointIds()->SetId(0, 0);  
        // 正面中心  
        frontTri->GetPointIds()->SetId(1, 2 + i);  
        // 當前點  
        frontTri->GetPointIds()->SetId(2, 2 + next);  
        // 下一點  
        polys->InsertNextCell(frontTri);
```

```
        // B. 背面三角形 (點順序順時針，法線朝外 -Z)  
        vtkNew<vtkTriangle> backTri;
```

```
        backTri->GetPointIds()->SetId(0, 1);  
        // 背面中心  
        backTri->GetPointIds()->SetId(1, 12 + next);  
        // 下一點  
        backTri->GetPointIds()->SetId(2, 12 + i);  
        // 當前點  
        polys->InsertNextCell(backTri);
```

```
        // C. 側邊牆面 (使用 4 點的 Quad 連接前後)  
        vtkNew<vtkQuad> sideQuad;
```

```
        sideQuad->GetPointIds()->SetId(0, 2 + i);  
        // 正面當前點  
        sideQuad->GetPointIds()->SetId(1, 12 + i);  
        // 背面當前點  
        sideQuad->GetPointIds()->SetId(2, 12 + next);  
        // 背面下一點  
        sideQuad->GetPointIds()->SetId(3, 2 + next);  
        polys->InsertNextCell(sideQuad);  
    }
```

```
    vtkSmartPointer<vtkPolyData> starPolyData =  
    vtkSmartPointer<vtkPolyData>::New();  
    starPolyData->SetPoints(points);  
    starPolyData->SetPolys(polys);  
    // -----  
    // 3. 強制自動計算正確法線 (防止光影交界處黑掉)  
    // -----  
    vtkNew<vtkPolyDataNormals> normals;  
    normals->SetInputData(starPolyData);  
    normals->ComputeCellNormalsOn();  
    normals->ComputePointNormalsOn();  
    normals->Update();  
    return normals->GetOutput(); // 回傳重新計算過法線的乾淨 PolyData  
}
```

```
    sideQuad->GetPointIds()->SetId(0, 2 + i);  
    // 正面當前點  
    sideQuad->GetPointIds()->SetId(1, 12 + i);  
    // 背面當前點  
    sideQuad->GetPointIds()->SetId(2, 12 + next);  
    // 背面下一點  
    sideQuad->GetPointIds()->SetId(3, 2 + next);  
    polys->InsertNextCell(sideQuad);  
    }
```

```
    vtkSmartPointer<vtkPolyData> starPolyData =  
    vtkSmartPointer<vtkPolyData>::New();  
    starPolyData->SetPoints(points);  
    starPolyData->SetPolys(polys);
```

```
    // -----  
    // 3. 強制自動計算正確法線 (防止光影交界處黑掉)  
    // -----  
    vtkNew<vtkPolyDataNormals> normals;
```

```
    normals->SetInputData(starPolyData);  
    normals->ComputeCellNormalsOn();  
    normals->ComputePointNormalsOn();  
    normals->Update();
```

```
    return normals->GetOutput(); // 回傳重新計算過法線的乾淨 PolyData  
}
```

# Create the Star

# Include for star

```
#include <vtkRegularPolygonSource.h>
```

```
#include <vtkPolygon.h>
```