CS3570 Introduction to Multimedia

Homework #4 Report

103062234 張克齊

PartI. Bezier Curve

[實作方法]

在助教提供的 code 中幫我們寫好了標點的方法，因此主要實作的就只剩 Bezier Curve 的演算法；在 spec 中有提到，我們實作的方式為一次取 4 個點 (3rd-order)，要注意的是起始點也是終點，並且每次取的最後一點也是下一次取的第一個點，因此若要取 10 個點，則需要在圖中標 9 個點，順序會是 1234, 4567, 7891，這樣我們就可以依據下圖的公式進行實作。

$$\boldsymbol{P}(t) = \boldsymbol{T} * \boldsymbol{M} * \boldsymbol{G}$$

$T = [t^3\ t^2\ t\ 1]$, ***M*** is called the ***basis matrix***. ***G*** is called the ***geometry matrix***.

$$M = \begin{bmatrix} -1 & 3 & -3 & 1 \\ 3 & -6 & 3 & 0 \\ -3 & 3 & 0 & 0 \\ 1 & 0 & 0 & 0 \end{bmatrix} \quad G = \begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix}$$

The blending functions are given by $T * M$,

$$\boldsymbol{P}(t) = (\boldsymbol{T} * \boldsymbol{M}) * \boldsymbol{G} = (1-t)^3 \boldsymbol{p}_0 + 3t(1-t)^2 p_1 + 3t^2(1-t)p_2 + t^3 p_3$$

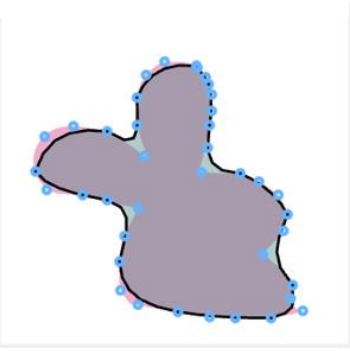
運算 Bezier 的部分我寫在 myBezier.m 中，要傳入的有標得點的 X 和 Y 座標、標的點數以及 LoD(圖中的 t)，算法如同圖中最後一行，要小心的只有當最後一段要回到起始點時的判斷，這邊因為題目是要求 39 個點以及 99 個點，也就是最後一段都會是 XXX1，因此我沒有用 mod 進行處理而是判斷最後一段，這樣就能達到 Bezier curve 的效果。

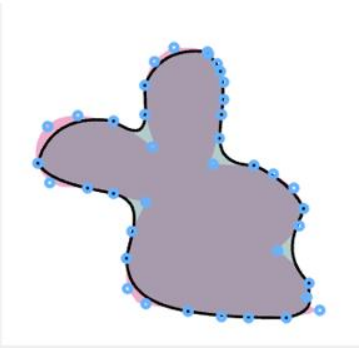
(b)小題的意思是要比較直接放大的結果跟放大每個點(x, y)再去做 Bezier 得到的結果有甚麼差異，因此可以利用 `result_big = imresize(result, 4, 'nearest');` 以 Nearest neighbor interpolation 放大四倍，並利用下圖的方式將點放大四倍後再丟入 myBezier()中，得到的結果覆蓋在放大的圖上，而最終我們可以透過 line()畫線比較兩者間的差異。

```
for i = 1 : clickedN;
    ctrlPointList(i, 1) = 4 * ctrlPointList(i, 1);
    ctrlPointList(i, 2) = 4 * ctrlPointList(i, 2);
end
```

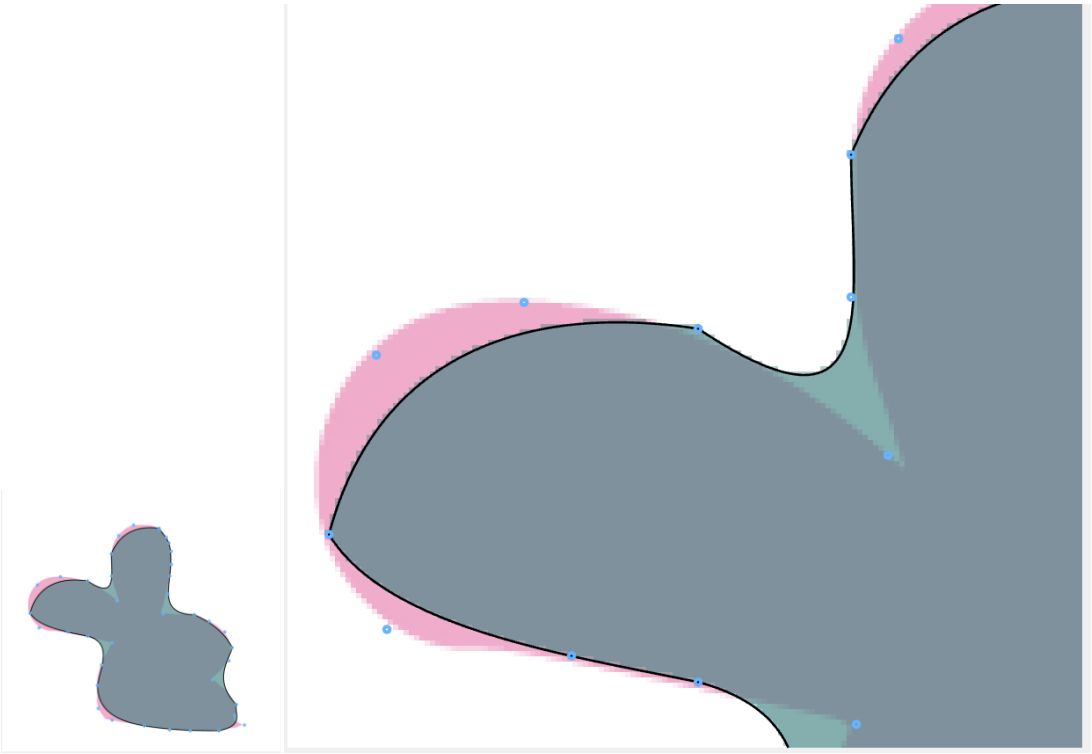
[結果圖]

40 points



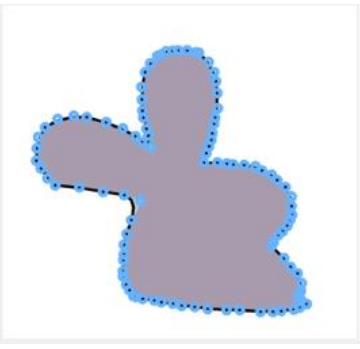


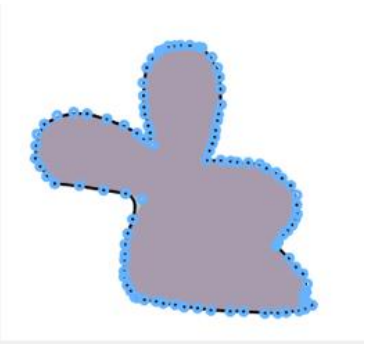
Original LoD Low LoD High



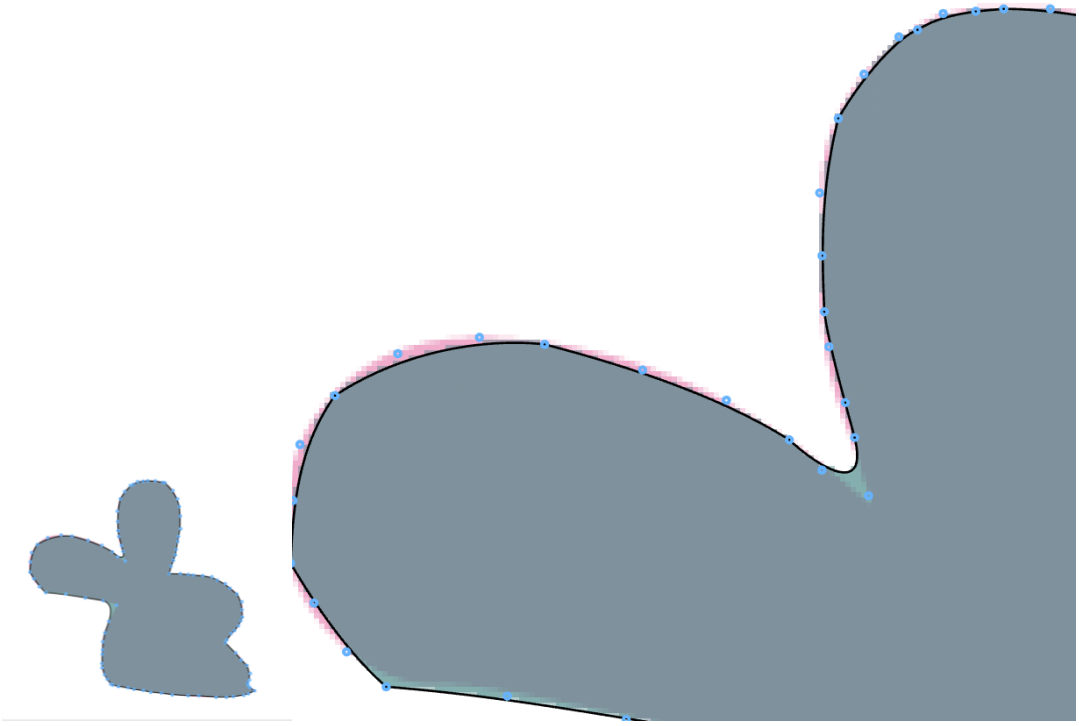
Resized by Nearest Neighbor(NN)

100 points





Original LoD Low LoD High

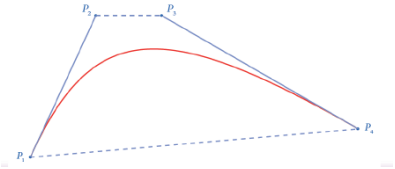


Resized by Nearest Neighbor(NN)

[比較與結論]

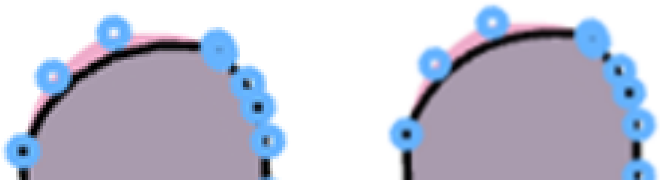
1. Sampling rate 的比較:

因為在實作 Bezier curve 中，取的四個點中只會保留起點和終點，中間兩點會被調整彎曲，如下圖所示；因此在取的點中(假設等距離)，當點數愈多代表愈密集，彎曲程度就比較小，更能貼近圖片的外圍，如上面結果圖比較可明顯發現。



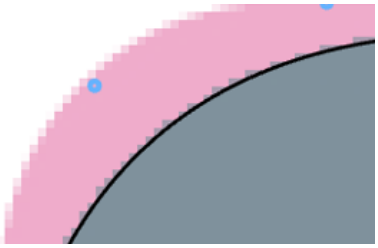
2. Levels of details(LoD)的比較:

LoD 的意義可以代表著在取的四個點中，要把他轉化成多少個點，像是 LoD = {0, 0.2, 0.4, 0.6, 0.8, 1}，就是把 4 個點轉換成 6 個等分的點，其中 0 跟 1 是起始跟終點，因此我們也可以推測出當 LoD 的間隔愈小(轉換的點愈多)，出來的曲線會更加平滑，從取 40 points 的兩張結果可以明顯發現 Low 比較不平滑。



3. (b)的比較：

從結果圖來看，我們可以發現直接放大的會有很明顯的顆粒階梯感，那是因為我們是以 pixel 為單位放大 4 倍，因此在做內插時會使得某些不在 line 上的 pixel 對應到原本圖片 line 上的 pixel，因此我們也了解到應該是要對點放大重新計算 Bezier curve 而非放大圖片才能得到比較精確的結果。

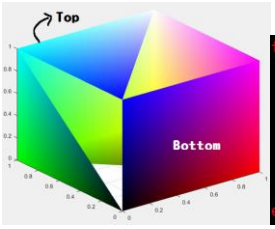


PartII. 3D Models

[實作方法]

1. RGB cube:

在執行助教附的 code 時，會發現有些三角形沒有被畫到，如下圖所示；這就讓我知道是 face 的地方寫的不完全，並且從圖中可以觀察到缺的規律是 2 個 Bottom 的 vertex 和 1 個 Top 的 vertex 圍成的三角形，所以我只要在同一個 for 迴圈加入這樣的 face 就能得到完整的 RGB cube。

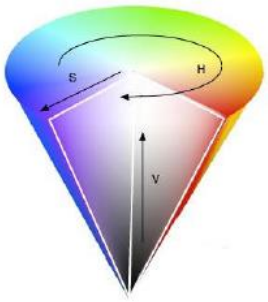



```
for vertI = 1 : 4
    faceVert1 = topVertIndex( mod(vertI,4)+1 );
    faceVert2 = topVertIndex( vertI );
    faceVert3 = botVertIndex( vertI );
    faceVert4 = botVertIndex( mod(vertI,4)+1 );
    faceVert5 = botVertIndex( vertI );
    faceVert6 = topVertIndex( mod(vertI,4)+1 );
    faces = [ faces ; faceVert1 faceVert2 faceVert3; faceVert4 faceVert5 faceVert6 ];
end
```

2. HSV cylinder:

一開始先用 supplement 教的方法做出角度及 XY 座標並存起來，這邊取的一個圓外圍的點數為 60 個，也就是每 6 度一個點；接著我先定義 Top 為上面的圓，Bottom 為下面的圓，而 vertex 的 index 規律定義為 1~60 是上面圓的外圍點，61 為上面圓的圓心，62~121 為下面圓的外圍點，122 為下面圓的圓心，這樣的設定能讓下面計算的 for 迴圈都是從 1 到 60；我使用兩個變數`top_mid = numOFVert+1;`和`bot_mid = numOFVert*2+2;`代表上下圓的圓心，並記錄每一點的座標。

再來我們要來定義 HSV 的 color，這邊可以參考第二章講義提到的 HSV color model，如下圖所示。H、S、V 的值範圍都是 0~1，圖中箭頭方向都是從 0 指向 1，因此我們可以實作出每個 vertex 對應的(H, S, V)，可對應到下面的 code。最後，因為螢幕的顯示是 RGB(0~255)，因此我們可以透過 matlab 的函式 hsv2rgb()將其轉至 RGB color model。




```
for i = 1 : numOFVert;
    topColor = [topColor; vertsPolarAngle(i)/(2*pi), 1, 1];
end
topColor = [topColor; 0, 0, 1];
for i = 1 : numOFVert;
    botColor = [botColor; vertsPolarAngle(i)/(2*pi), 1, 0];
end
botColor = [botColor; 0, 0, 0];
vertColors = [ topColor; botColor ];
vertColors = hsv2rgb(vertColors);
```

最後就是跟 RGB cube 一樣要找出每一個要畫的 face，我的順序是 Top 到 Side 到 Bottom，方式跟做 RGB cube 一樣，只要記得判斷 Top 跟 Bottom 是否繞了一圈就好，如下圖所示。這樣，就能完成 HSV cylinder。

```
% Top faces
for i = 1 : numOFVert;
    if(i == numOFVert)
        faces = [faces; i, 1, top_mid];
    else
        faces = [faces; i, i+1, top_mid];
    end
end
```

```
% Side faces
for vertI = 1 : numOFVert;
    faceVert1 = topVertIndex( mod(vertI,numOFVert)+1 );
    faceVert2 = topVertIndex( vertI );
    faceVert3 = botVertIndex( vertI );
    faces = [ faces ; faceVert1, faceVert2, faceVert3];
end
for vertI = 1 : numOFVert;
    faceVert1 = topVertIndex( mod(vertI,numOFVert)+1 );
    faceVert2 = botVertIndex( vertI );
    faceVert3 = botVertIndex( mod(vertI,numOFVert)+1 );
    faces = [ faces ; faceVert1, faceVert2, faceVert3];
end
```

3. 平移與結合:

首先考慮平移，對於 HSV cylinder 來說，因為我們本來的中心就已經是(0, 0, 0)，所以我們可以直接將所有 vertex 平移到想要的位置，如下左圖，代表將圓柱平移到中心為(0, 2.5, -2.5)；但對於讀進來的 OBJ 我們就不能直接這樣做，原因是他的中心並非是原點，所以處理時要先分別找出他的 x, y, z 軸的最大最小值相加除以 2，找的方式為跑過所有值並開變數刷新存取，這樣才可以得到我們 OBJ 當前中心位置，就能透過跟剛剛一樣的平移方式將其移至原點，如下右圖所示。

```
v(:, 1) = verts(:,1);
v(:, 2) = verts(:,2)+2.5;
v(:, 3) = verts(:,3)-2.5;
```

```
OBJmid_x = (OBJmax_x+OBJmin_x)/2;
OBJmid_y = (OBJmax_y+OBJmin_y)/2;
OBJmid_z = (OBJmax_z+OBJmin_z)/2;
```

至於結合的部分，照著助教的提示就是把圓柱和 OBJ 的 vertices, colors 以及 faces 結合在一個 matrix 中，其中要特別注意 faces 的 index 問題，因為在分開時兩者的編號都是從 1 開始，但結合後我們應該讓加在後面的 index 做平移，我是將 OBJ 加在後面，所以是 `faces = faces + size(vertexOBJ,1);`，這樣就能透過下圖的方式結合將兩個物體一起顯示。

```
BindF = [facesOBJ; faces];
BindV = [vertexOBJ; v];
BindC = [colorsOBJ; vertColors];
```

4. Lighting(每個 figure 可以給他一個新的光源)

(1) 光的種類:

這邊要實作兩種光源 position light(point light)和 directional light，方法是依據 matlab 官方文件，只要在 light()函式中加入'Style'參數，其中 local 為 position light 而 infinite 為 directional light，這樣就能打出兩種光並進行比較。

```
light('Position',[0.0,0.0,3.5], 'Style', 'local');
```

```
light('Position',[0.0,0.0,3.5], 'Style', 'infinite');
```

Style — Type of light source
'infinite' (default) | 'local'

Type of light source, specified as one of these values:

- 'infinite' — Place the light at infinity. Use the Position property to specify the direction from which the light shines in parallel rays.
- 'local' — Place the light at the location specified by the Position property. The light is a point source that radiates from the location in all directions.

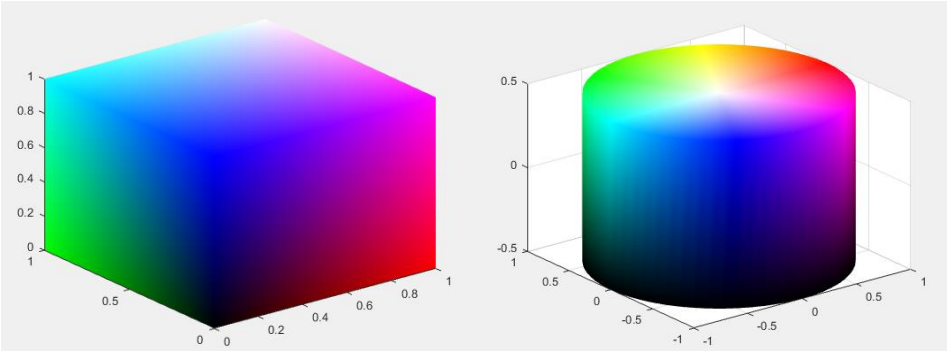
(2) 光的三元素:

這邊先討論作法，只要運用官方文件提到的 material([ka ks kd])函式，就可以給題目要求不同的值來比較結果。

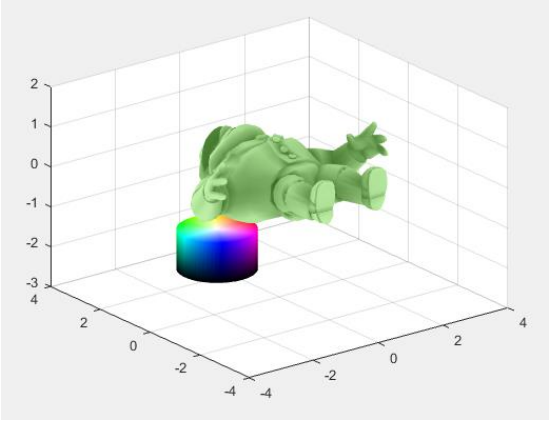
material([ka kd ks]) sets the ambient/diffuse/specular strength of the objects

```
material([1.0 0.0 0.0]);
light('Position',[0.0,0.0,3.5]);
lighting phong;
```

[結果圖]

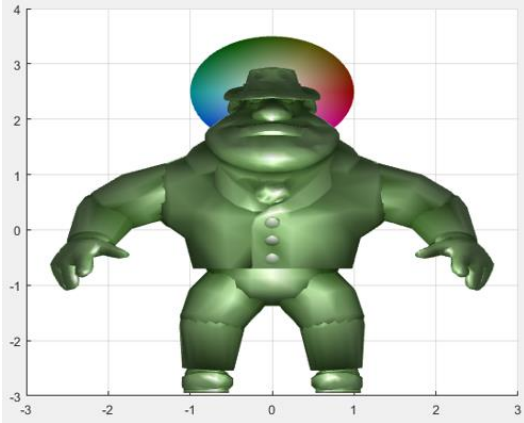


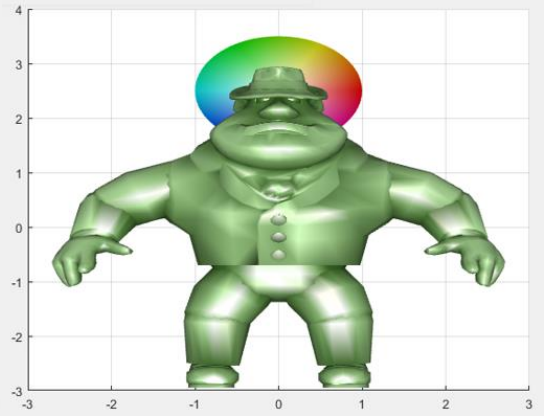
RGB Cube　　HSV Cylinder



Original combination

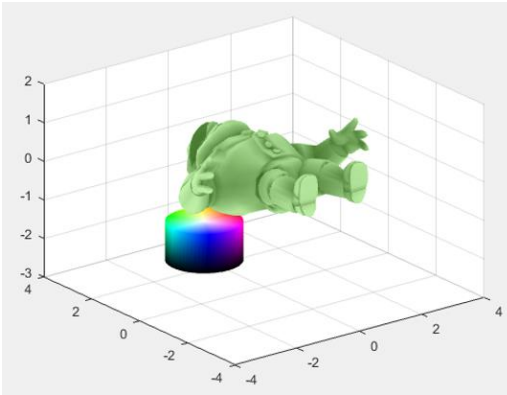
Light Source

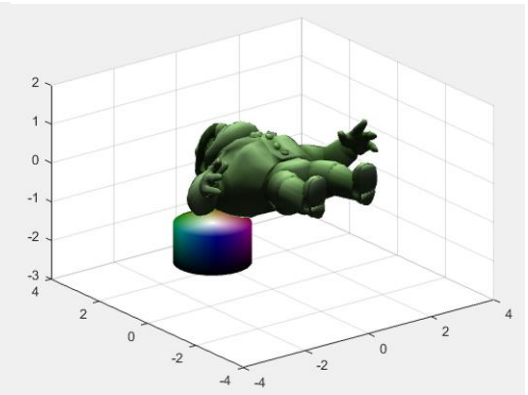




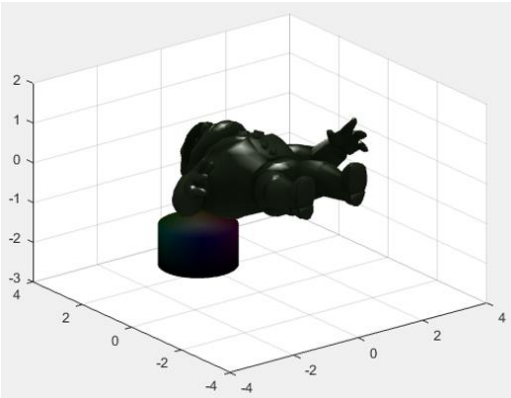
Position(Point) light　　Directional light

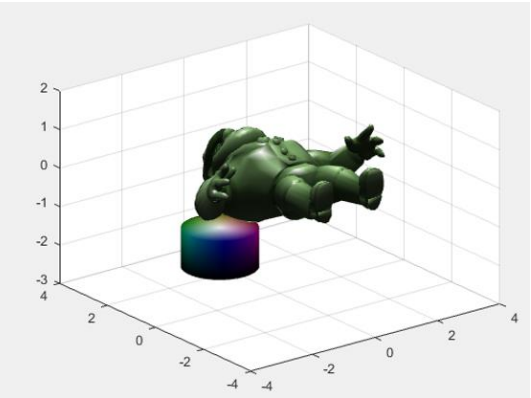
[ka, kd, ks]





[1.0, 0.0, 0.0]　　[0.1, 1.0, 0.0]





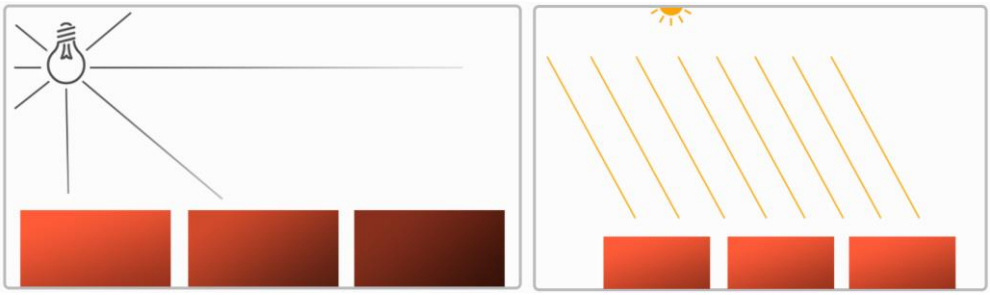
[0.1, 0.1, 1.0]　　[0.0, 0.8, 1.0]

[比較與結論]

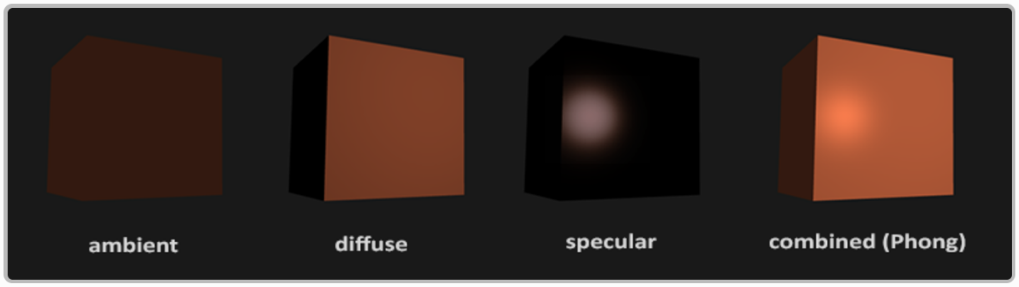
1. 討論不同種類光的結果:

Position light 又稱為 point light，代表著一個有位置的光源向所有方向發光，且光的強度會隨著距離而減低(變暗)，可以想成燈泡的效果，如下左圖；Directional light 代表著假設光源很遠，那會變成一道道平行光，並來自於同一方向，最簡單

的聯想為太陽，如下右圖。從結果圖可以顯示兩者的差異，我們可以很明顯發現若為 position light，物體會有陰影，而平行光則是每個角度都會發亮。



2. 光的三元素討論:



- 環境光照(Ambient Lighting)：即使在黑暗的情況下，世界上也仍然有一些光亮(月亮、一個來自遠處的光)，所以物體永遠不會是完全黑暗的。我們使用環境光照來模擬這種情況，也就是無論如何永遠都給物體一些顏色。
- 漫反射光照(Diffuse Lighting)：模擬一個發光物對物體的方向性影響(Directional Impact)。它是馮氏光照模型最顯著的組成部分。面向光源的一面比其他面會更亮。
- 鏡面光照(Specular Lighting)：模擬有光澤物體上面出現的亮點。鏡面光照的顏色，相比於物體的顏色更傾向於光的顏色。

Phong lighting model 主要是由上述三元素所組成，他們的概念如同參考網站上所述，這邊分析題目中給的四組職所對應的結果圖

I. (k_a , k_d , k_s) = (1.0, 0.0, 0.0):

代表只存在環境光，透過環境光可以使我們看到物體的每個地方，也就是無論光源在哪裡整個物體都有光打到。

II. (k_a , k_d , k_s) = (0.1, 1.0, 0.0):

代表主要受到反射光的影響，所以我們看到的物體會只有光源打到物體反射的部分，因此會有陰影的感覺。

III. (k_a , k_d , k_s) = (0.1, 0.1, 1.0):

代表主要為鏡面光，也就是只會顯示光源反射到我們眼睛的光，產生像是光澤的效果，其他部分若設為0則會呈現黑色。

IV. (k_a , k_d , k_s) = (0.1, 0.8, 1.0):

代表受到反射光和鏡面光的影響，因此會呈現II跟III結合的效果，也就是有陰影及光澤的結果圖。

參考網站: https://learnopengl-cn.readthedocs.io/zh/latest/02%20Lighting/02%20Basic%20Lighting/