Part1. Bézier curve

實作目的：

在 part1 中，我們需要實作在不同 Levels of detail(LoD)和 sampling rate 下的 Bézier curve，並試圖將得出的結果圖利用 nearest neighbor interpolation 來放大，最後再將 sampling points 擴增為四倍，來得到最後的計算結果。

實作細節：

首先，我們要了解 LoD 的意思，在 Low LoD 中，t = {0, 0.2, 0.4,…, 1}，sample rate 為 0.2，包含 0 和 1 共 1/0.2 + 1 共 6 個點；而 High LoD 的部分，t = {0, 0.01, 0.02,…, 1}，sample rate 為 0.01，包含 0 和 1 共 1/0.01 + 1 共 101 個點，在 matlab 中可以用

```
LoD_low = 0:0.2:1;
LoD_high = 0:0.01:1;
```

來表示。

接下來，就是根據講義帶入 Bézier curve 的 equation：

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

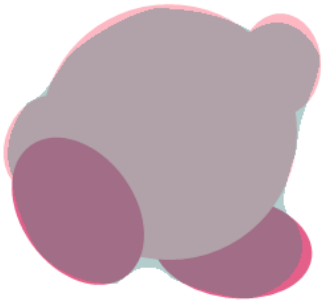
在這裡，我們利用 for 迴圈來設定 p0 到 p3 的值，首先，先從助教寫好的 ctrlPointList 中取出 p0 到 p3，根據 curve 的定義，每一個 group 的終點將會是下一個 group 的起點，因此在實作時要特別注意。在這裡，我使用 for 迴圈中每次重複迴圈時加 3 的方式，來表示前一次最後一個點是下次第一個點的概念，用 code 來說的話，是以 mod(i,x)來實作，當 i = x 時，代表 x 是最後的終點(ending end point)。當設定好 p0、p1、p2、p3 後，只要依據公式帶入對應值，並將最後結果 assign 給 outlineVertexList，就可以完成 Bézier curve 的實作。

```
for i = 0: 3: x

    p0 = ctrlPointList(mod(i, x) + 1,:);
    p1 = ctrlPointList(mod(i+1, x) + 1,:);
    p2 = ctrlPointList(mod(i+2, x) + 1,:);
    p3 = ctrlPointList(mod(i+3, x) + 1,:);

    pt_low(:,1) = ((1-LoD_low).^3) * p0(1) + 3 * LoD_low.*(1-LoD_low).^2 * p1(1) + 3 * (LoD_low.^2).*(1-LoD_low) * p2(1) + LoD_low.^3 * p3(1);
    pt_high(:,1) = ((1-LoD_high).^3) * p0(1) + 3 * LoD_high.*(1-LoD_high).^2 * p1(1) + 3 * (LoD_high.^2).*(1-LoD_high) * p2(1) + LoD_high.^3 * p3(1)
    pt_low(:,2) = ((1-LoD_low).^3) * p0(2) + 3 * LoD_low.*(1-LoD_low).^2 * p1(2) + 3 * (LoD_low.^2).*(1-LoD_low) * p2(2) + LoD_low.^3 * p3(2);
    pt_high(:,2) = ((1-LoD_high).^3) * p0(2) + 3 * LoD_high.*(1-LoD_high).^2 * p1(2) + 3 * (LoD_high.^2).*(1-LoD_high) * p2(2) + LoD_high.^3 * p3(2)

    out_low = [out_low; pt_low];
    out_high = [out_high; pt_high];

    pt_low = zeros(length(LoD_low), 2);
    pt_high = zeros(length(LoD_high), 2);

end
```

實作結果：

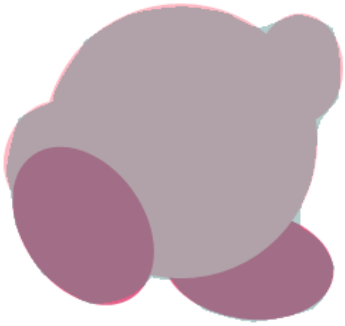
1. Low sampling rate of 36 points with low LoD

2. Low sampling rate of 36 points with high LoD

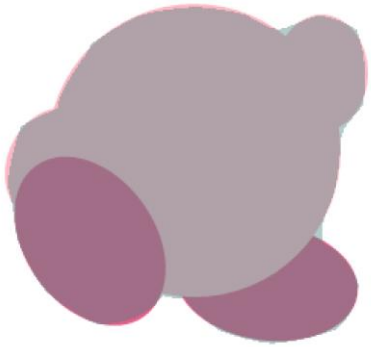


3. High sampling rate of 72 points with low LoD

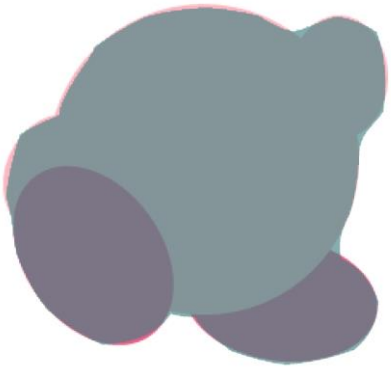
4. High sampling rate of 72 points with high LoD



5. Nearest-neighbor interpolation to scale by 4 times



6. Scale the sampled control points by 4 times



實作結果分析：

1. Discuss the results between different sampling rates and different levels of detail

以 sampling rate 來說，首先，我們要了解關於實作 Bézier curve 中重要的一點，在之前提供的計算公式中的 p0 到 p3 四個點，會有兩點分別為 starting end point 與 ending end point，而剩下的兩點則擔任 direction point 的角色。direction point 共同負責控制此曲線的方向，因此當 sampling rate 較高時，因為取樣的點較多，在每一個 group 中，點之間的距離會比較小，會比較貼近原本圖像中的曲線，讓最後形成的圖較接近原圖，因此在相同 LoD 下，sampling rate 高時會有較好的結果（實作結果的 1,3 比較與 2,4 比較）。
以 LoD 來說，LoD 會將前一步驟選出的 4 個點(p0~p3)再細分為 n 等分，例如像這次的實作中，分別是 low LoD : t = {0, 0.2, 0.4,…, 1}(n = 6)和 high LoD : t = {0, 0.01, 0.02,…, 1}(n = 101)，當在 0 和 1 之間細分的點愈多時，代表我們在這四個點中用更多的點去逼近，因此實作出的曲線會越平滑(階梯狀比較不明顯)、變化程度更加細緻。

2. Compare results in (b) and discuss it.

使用 nearest neighbor interpolation 的方式來做 scaling 時，因為它取點的方式是簡單的內插(如同我們 HW1 中實作)，因此放大就是直接取最接近的 index 作為圖中放大後的影像，在實作出的結果圖片中可以發現，這樣的作法 pixilation 非常明顯，放大效果會有誤差，容易產生鋸齒狀，因為它就真的只是單純的把一張圖片直接放大而已，這樣 bitmap 式的放大方法，畫質的改變不在其考慮範圍之中。

反之，使用 sampling point 座標來放大時，呈現出來的成果就比第一種方式來的好很多，因為他的計算方式是把原始作曲線的點位置校正，放到放大後的位置再重新計算一次 Bézier curve。這樣向量化(vector)的放大，就比較沒有 pixilation 的問題了。

Part2. 3D Models

實作目的：

在 part2 中，主要的目的是希望可以利用 matlab 劃出簡單的 3D 模型(例如題目要求的六角柱與立方體)，並學習如何將不同模型共同顯示在相同 3D 空間中，並練習不同光線(direction light、position light)的實作和了解光線參數(ambient strength ka、diffuse strength kd、specular strength ks)的調整對 3D 空間中模型呈現的影響。

實作細節：

在 a 小題中，我們要想辦法將 cube 的面補起來，而實作的方法參考了以下概念：每一個面的正方形，都可以視為由兩個三角形組成，因此只要把原本存在的四面體側面，bind 上缺漏的面(將缺的這些三角形的頂點找出來，加進 verts 矩陣裡)，最後將他們加進新的 face 就可以完成。

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

接下來的部分，我們便照著 spec 中的步驟逐步完成實作。

首先，我們要把 object 的 center 移動到(0,0,0)這個位置，因此我們必須得知的是目前 center 的位置，接下來只要把模型中的每個點減掉 center 座標(此時的新 center = center - center = (0,0,0))就可以將整個模型以 center 為(0,0,0)完成平移。

```
center = [(max(vertex(:, 1))+min(vertex(:, 1)))/2, (max(vertex(:, 2))+min(vertex(:, 2)))/2, (max(vertex(:, 3))+min(vertex(:, 3)))/2];
vertex = [vertex(:,1) - center(1), vertex(:,2) - center(2), vertex(:,3) - center(3)];
```

而接下來，我們要利用 triangular surface approximation，在 x-z 平面上劃出一個邊長和高都是 1 的 HSV color hexagonal prism，因為上下兩個六角形面是由六個三角形所組成，因此每個面會有最重要的七個點(六角形六個頂點+中心點)，透過這七個點，我們就可以完成六角形上下兩面的實作。

在設定 vector 的 index 這部分，我使用 for 迴圈跑過所有 index 來填入對應結果；設定 vertex 位置這部分，也是使用 for 迴圈來填入對應值；至於 color 值的設定則是使用 matlab 內建的 hsv2rgb 這個 function 來計算出顏色。

根據 HSV 的設定，H 為顏色、S 為飽和度、V 為亮度，這三個變數範圍皆在 0 到 1 之間，因此在實作上，把每個點的 H 設為 vertsPolarAngle(i)/2*pi，也就是依據該點在六角形上的相對位置，來形成不同角度不同的顏色差異，S 的部分，上下兩個六角形都設為 1，V 則是該點在六角柱的高度，上六角形設為 1、下六角形設為 0，再透過 hsv2rgb 就可以呈現顏色的表現。

當 vertex 和 color 設定完成後，就是建立六角柱的每個面，上下兩面作法已經在上面的說明中提及，至於側面的作法，與 a 小題非常相似，就是透過將六角柱每個側面的矩形切為兩個三角形來實作。

```
for i = 1:numofVert
    top_colors = [top_colors; hsv2rgb([vertsPolarAngle(i) / (2 * pi), 1, 1])];
    bottom_colors = [bottom_colors; hsv2rgb([vertsPolarAngle(i) / (2 * pi), 1, 0])];
end

myColor = hsv2rgb([0, 0, 1]);
top_colors = [top_colors; myColor];
myColor1 = hsv2rgb([0, 0, 0]);
bottom_colors = [bottom_colors; myColor1];
vertColors = [top_colors; bottom_colors];
```

將 center 移動到(0,-1.4,0)的方法與前面移動到原點的方式相同，只要將其 y 值減掉 1.4 即可完成。而透過 bind 兩個模型的 faces、vertex、colors 設定，就可以將兩個模型共同顯示在一個 world space 中。

```
faces = faces + size(verts,1);
face = [faces_hex; faces];
vert = [v; vertex];
color = [vertColors; colors];
```

在 lighting 的部分，實作 directional light 的方式是給予 light()這個 function 的 style 設為 infinite，也就是平行光無限制延伸的概念。

```
light('Style', 'infinite');
lighting phong;
```

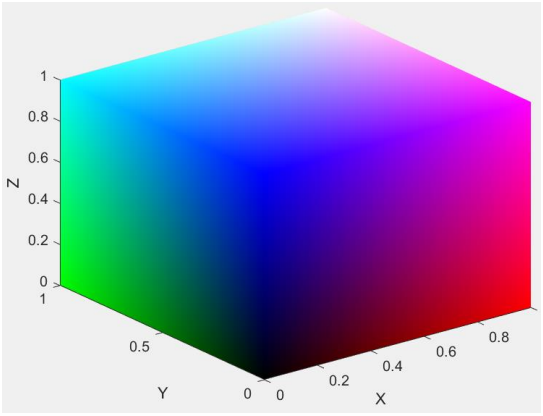
而 point light 的部分，就是將 style 設為 local，並設定點光源位置即可。

```
light('Position',[0.0,0.0,5.0], 'Style', 'local');
lighting phong;
```

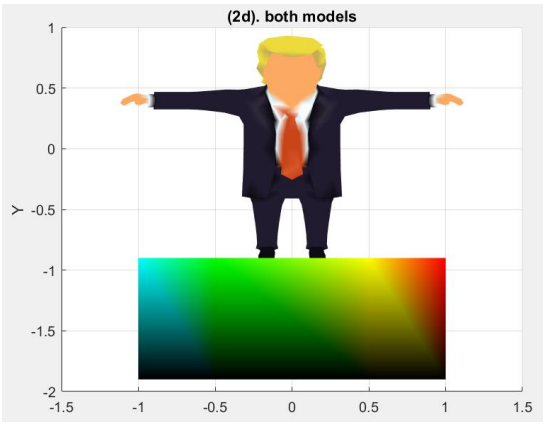
在最後的光線參數調整中，用 material()這個 function 來設定 ambient strength ka、diffuse strength kd、specular strength ks 即可。

實作結果：

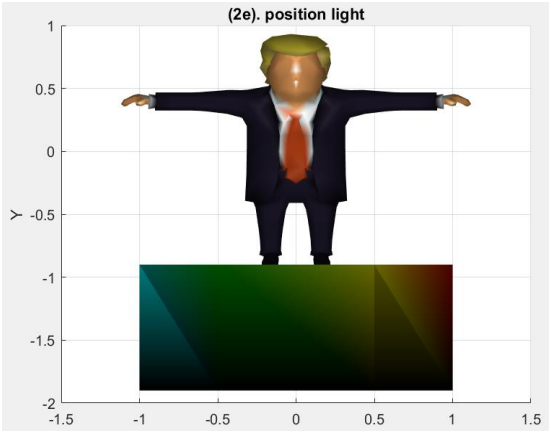
(a). cube



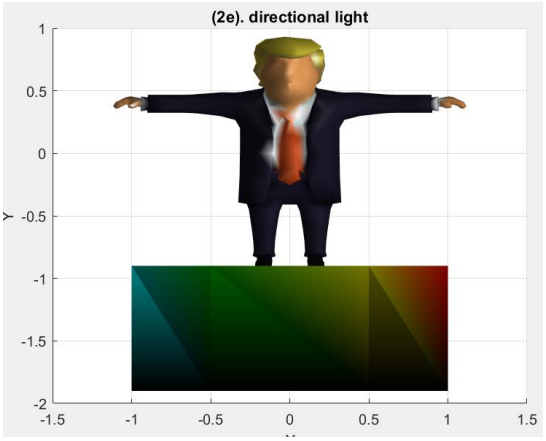

(d). Both models in same world space



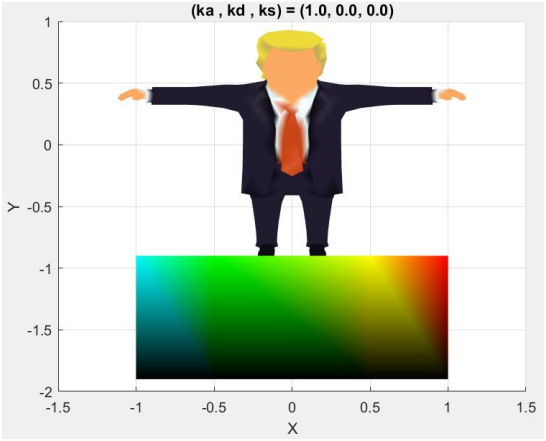

(e). Positional light



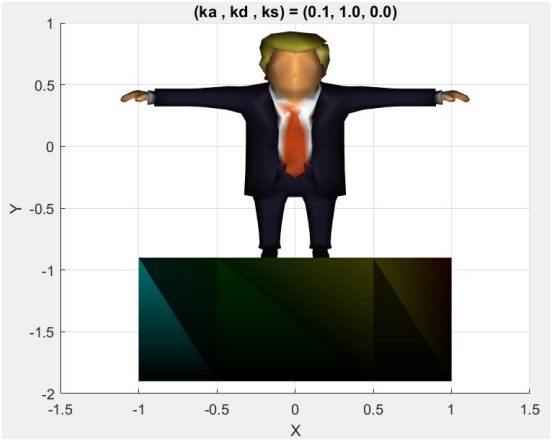

(e). Directional light



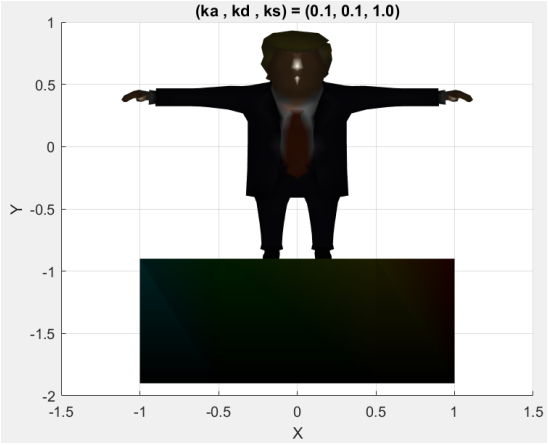

(f). (ka, kd, ks) = (1.0, 0.0, 0.0)

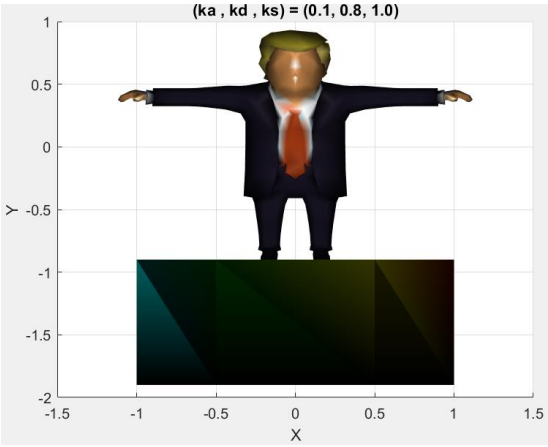



(f). (ka, kd, ks) = (0.1, 1.0, 0.0)



(f). (ka, kd, ks) = (0.1, 0.1, 1.0)　(f). (ka, kd, ks) = (0.1, 0.8, 1.0)




實作結果分析：

1. Differences of different kind of light in (e)

因為我將點光源(position light)的位置設定為(0,0,5)，從 x-y 平面上來看的話，光是以垂直的角度照射到川普模型的正面上，所以可以看到他的臉部中間會有光照的效果出現；在 direction light 照射下的結果圖片中，川普的臉呈現一邊亮一邊暗的明顯，亮側就是受 direction light 的受光面，暗側則是背光面。

2. Differences from the results in (f)

在比較結果之前，我們應該要先了解這三個參數的物理意義：ambient 為環境光強度，是一種沒有方向性的光強度；diffuse 代表漫射光強度，是光線向量與法向量的 dot 值，specular 為反射強度，計算的是視線向量與反射向量的 dot 值。

第一組參數將 ambient 設為 1，其他兩項設為 0，環境光參數 ambient 的出現代表模型將會全部被環境光照射，因此整張圖呈現均勻的亮色，沒有什麼陰影。

第二組參數將 diffuse 值調高為 1，其他數值偏低，而漫射光參數 diffuse 會影響到光的陰影效果，所以陰影會很明顯，可以較明顯的看出物體的凹凸起伏與皺褶的程度。

第三組參數將 specular 值調高為 1，其他數值偏低，以圖片效果來說，整張圖片亮度偏暗，但是在特定反射區域亮度非常的高，會產生發亮光澤的效果，這和光源位置和所在視角有關。

第四組參數因為三者數值都很相近，因此結果會跟一般反射效果最相似，不論是凹凸程度或者光澤都很明顯。