
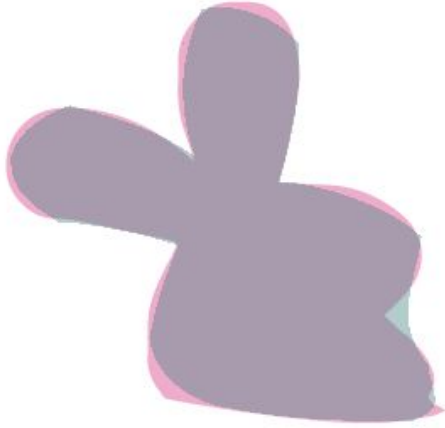
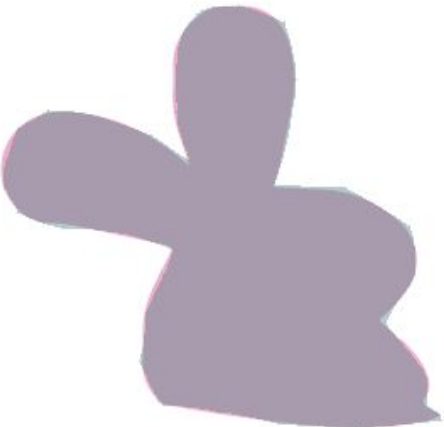
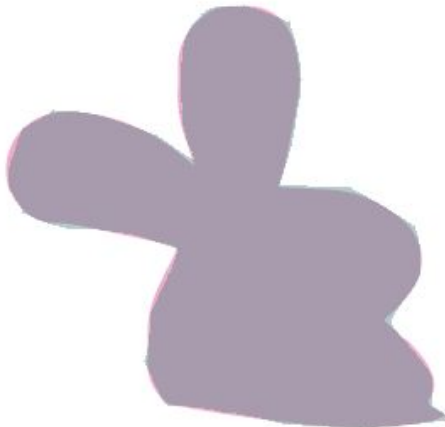


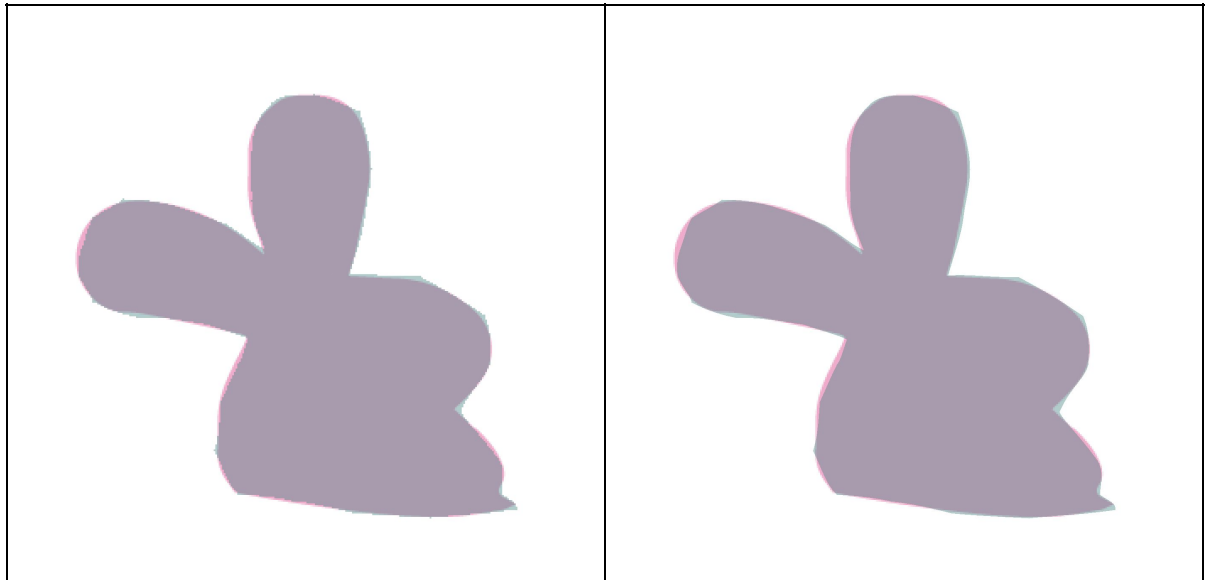
1Bézier curve

(a)result:

low sample rate&low LoD	low sample rate&high LoD
	
highsample rate&low LoD	highsample rate&high LoD
	

(b)result:

bitmap	Bézier curve
--------	--------------



I. Describe how you implement the Bézier curve.

公式

$$\begin{aligned}
 \mathbf{P}(t) &= \sum_{i=0}^3 \mathbf{P}_i B_i(t) \\
 &= (1-t)^3 \mathbf{P}_0 + 3t(1-t)^2 \mathbf{P}_1 + 3t^2(1-t) \mathbf{P}_2 + t^3 \mathbf{P}_3 \\
 &= \begin{bmatrix} (1-t)^3 & 3t(1-t)^2 & 3t^2(1-t) & t^3 \end{bmatrix} \begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{bmatrix} \\
 &= \begin{bmatrix} 1 & t & t^2 & t^3 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ -3 & 3 & 0 & 0 \\ 3 & -6 & 3 & 0 \\ -1 & 3 & -3 & 1 \end{bmatrix} \begin{bmatrix} \mathbf{P}_0 \\ \mathbf{P}_1 \\ \mathbf{P}_2 \\ \mathbf{P}_3 \end{bmatrix}
 \end{aligned}$$

$$= (T^*M)^*G$$

for $0 \leq t \leq 1$

```

79 - outlineVertexList = [];
80 - T=[];
81 - for I = 1:3:100
82 -     G = zeros(4, 2);
83 -     for J = 0:3
84 -         idx=mod(I+J,100)+1;
85 -         G(J+1, :) = ctrlPointList(idx, :);
86 -     end
87 -     for K = 1:sampleHigh
88 -         T = [t_high(K).^3, t_high(K).^2, t_high(K), 1];
89 -         outlineVertexList = [outlineVertexList; T*M*G];
90 -     end
91 - end

```

I表4個點一組共有99個點的情況(I不會跑到100所以設99 or 100沒差)

(line 83~86)其中的某一組的四個點分別存到g

(line 87~90)即在做公式 $T*M*G$

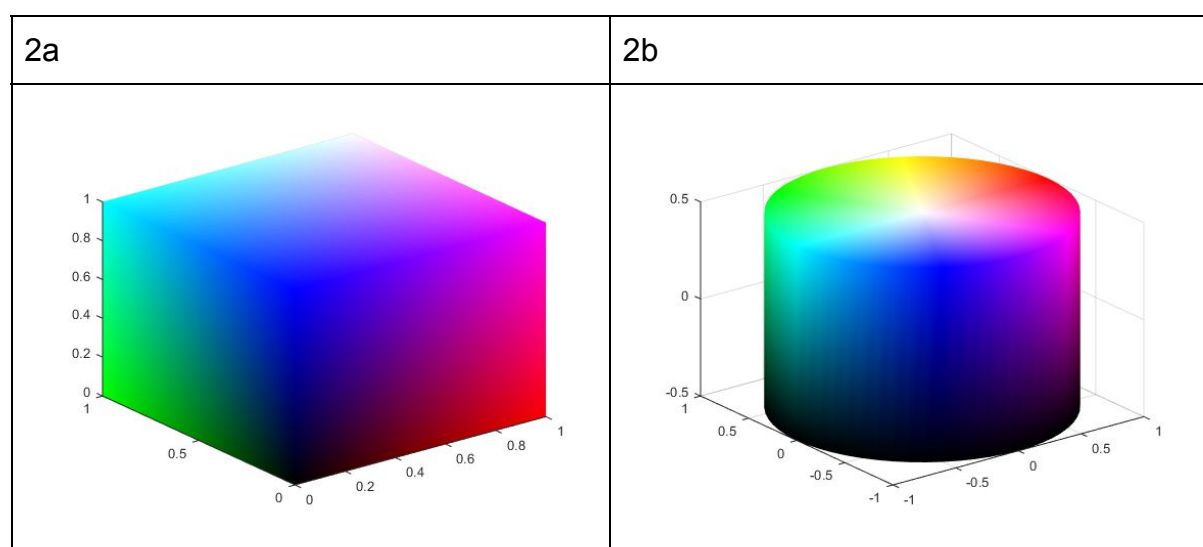
II. Discuss the results between different sampling rates and different levels of detail. sampling rate低的兔子，灰色面積覆蓋於粉紅色面積上明顯形狀有落差而sampling rate高的兔子幾乎看不到粉紅色的面積，表示Bezier curve和原本的兔子外圍較吻合。同sampling rate 不同levels差異不大，可能是因為我取的間距本身就頗小，所以在之間多差幾個點的效果不顯著。

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

bitmap直接放大的曲線會有一塊一塊的感覺而不是平滑曲線，而用放大後用Bezier curve再做一次兔子的曲線會發現平滑很多!!但是執行時間真的很長，一度我還以為有bug所以卡住了!等很久後才發現原來真的有正確執行運算，只是真的很久。相較來說用bitmap直接放大雖然效果沒有Bezier curve那麼好，但是執行時間塊非常多，幾乎是一執行程式就有結果了。

2. 3D Models

I. Show the screenshot of (2a) (2b) and discuss how to build these models.



畫3D object時，先對object標頂點，然後用這些頂點做三角形，每一片三角形即為object的表面。

(2a)

1先定義正方體的8個vertex:

(0,1,0),(0,1,1),(1,1,1),(1,1,0),(0,0,0),(0,0,1),(1,0,1),(1,0,0)

2定義8個vertex的index:

為了方便之後定義face時，可以簡化成index表示vertex，而不用把每個vertex的座標打出

3定義face:

因為object是由三角形組成表面，所以雖然正方體有6個面，但以三角形表示需要 $6 \times 2 = 12$ 片三角形(即一個面由兩片三角形組成、三個頂點組成一個三角形，所以一個face由三個vertex組成)所以要在faces加入12塊三角片。

注意，在定義face時，我們利用填vertex組成face，平面的法向量朝柱體內或朝柱體外結果可能不同，如果發現明明定義過某face而且也給vertex正確的顏色，但是某些三角片仍黑黑的，可以考慮換vertex的組合來改變normal方向。

4定義顏色:

由

```
trisurf(faces,verts(:,1),verts(:,2),verts(:,3),'FaceVertexCData',  
vertColors,'FaceColor','interp','EdgeAlpha',0);  
'FaceVertexCData'
```

可知顏色是跟著vertex定義的，所以我們要對每一個vertex定義顏色，而此正方體剛好是邊長都是單位長，剛好符合rgb的值域，所以頂點座標即為rgb值。

至於顏色漸層的原因是'interp'，此特性會內差三角片頂點間的顏色。

(2a)

1製造圓柱體底部的"圓形":

可先由pizza一片一片的狀態想像，如果把pizza分成角度及小的很多份，每一小份可以看成一片三角形的pizza，同理，如果我們有多個角度極小的三角形，繞著同一個頂點排好，可近似出圓形。

此題我用了60塊三角形建構圓形，故在圓取出60個頂點。

2定義圓柱體頂點座標

由1我們得到一片圓形的頂點，而圓柱體所需要的頂點，就是構成兩片平行的圓形的頂點

3定義vertex的index

為了之後在定義face時，可以較簡化表達 vertex

4定義圓柱face

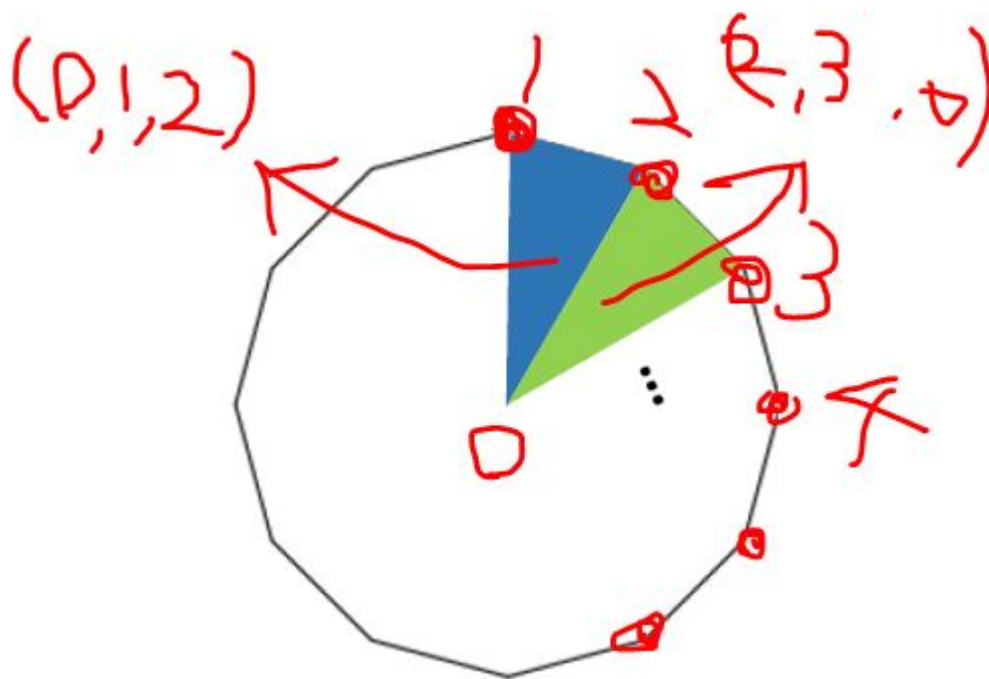
圓柱可分割成三個類別的面，top、bottom的圓形和兩片圓形之間的side face

top、bottom face

0,1,2...表vertex的index

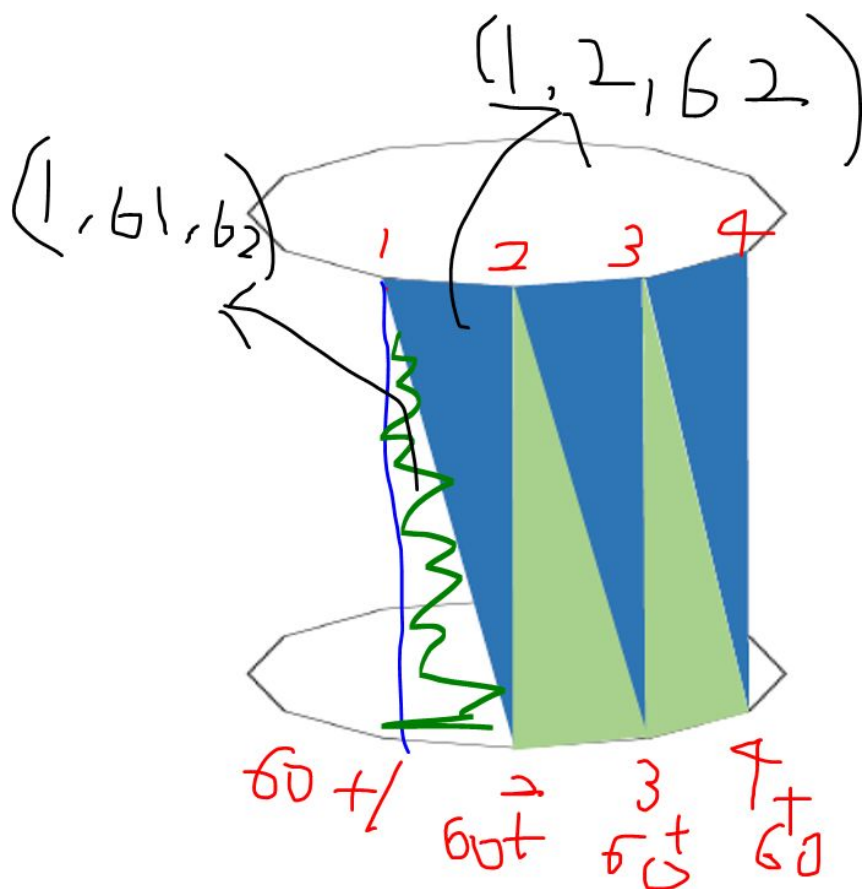
藍色可由(0,1,2)表示成face

綠色可由(2,3,0)表示成face



side face

bottom的點要記得加60是因為本題我定義上下員皆是由60個三角形片(故有60個頂點)構成



把face分成這三大類後，將構成這三類的全部三角形片定義完才算完成5定義顏色

先用hsv定義顏色再轉乘rgb，而在存顏色時也是和定義cube顏色一樣，要對頂點定義顏色(因為`trisurf(faces,verts(:,1),verts(:,2),verts(:,3),'FaceVertexCData', RGB,'FaceColor','interp','EdgeAlpha', 0);`)定義完所有頂點的hsv在轉成rgb才算完成定義顏色。

HSV的意義:

H varies from 0 to 1, the resulting color varies from red through yellow, green, cyan, blue, and magenta, and returns to red.

<-此顏色順序及我們要的顏色變化，故0~1切成60等分assign給top side的vertex

When S is 0, the colors are unsaturated (i.e., shades of gray). When H(:,2) is 1, the colors are fully saturated (i.e., they contain no white component).

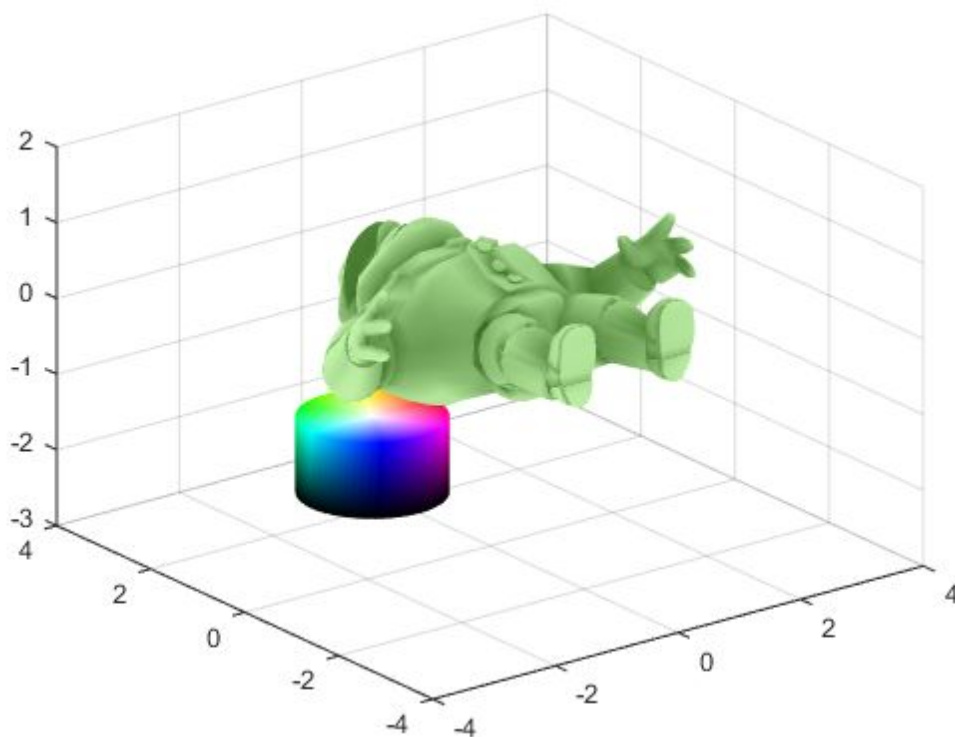
<-bottom side的顏色很黑，所以assign s=0給所有底部圓形的vertex

<top side顏色飽和!所以assign s=1給所有上部圓形的vertex

As V varies from 0 to 1, the brightness increases.

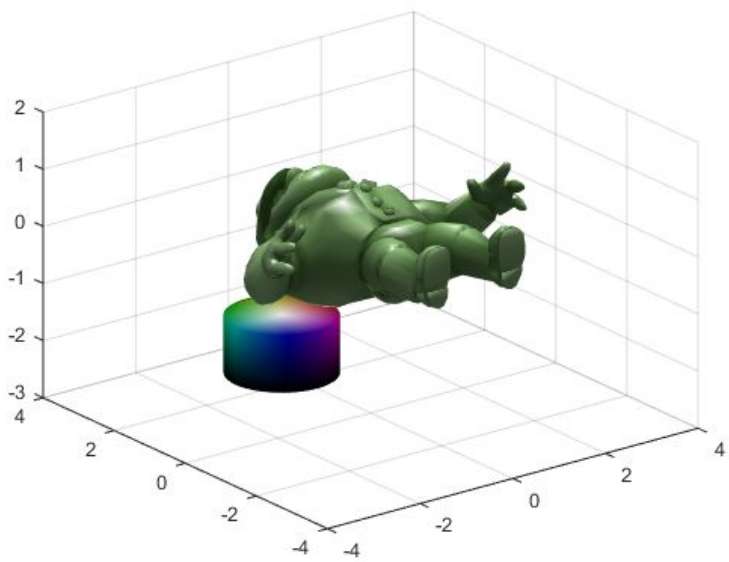
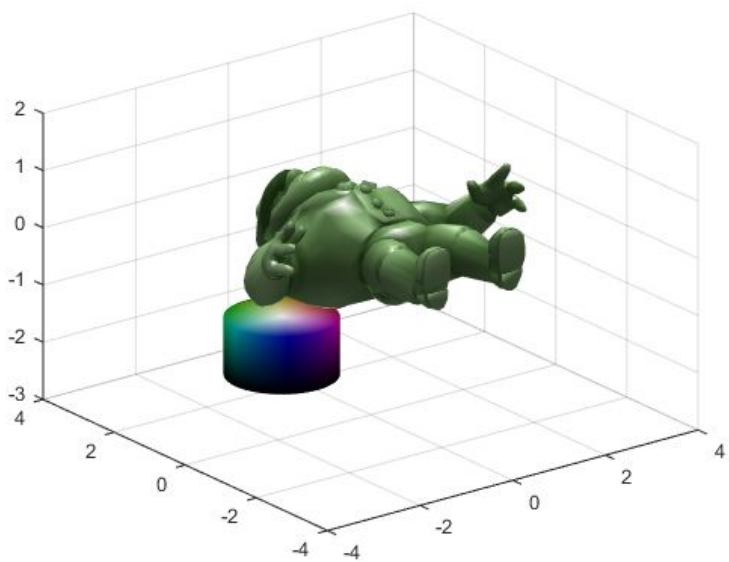
II. Show the screenshot of (2c) and discuss how to implement the transformation.

在2b中已經實作過圓柱的3 D object，只需要把上一題圓柱的vertex、face和本題load進來object的vertex、face、color合併，即可使用做同一個object的方法用`trisurf`印出。不過合併後要注意如果face是用vertex index定義，記得平移! 顏色因為是跟著vertex定義，所以需要注意顏色矩陣合併後的順序要和vertex的順序相同。



III. Show the screenshot of (2d) and discuss the difference of different kind of light.

position light的結果看起來光源比較集中，反光的地方亮的部分比directional light還亮，但是不仔細看我覺得差異沒有很明顯

<p>position light from (0,0,5)</p>	
<p>directional light from (0,0,5)</p>	

IV. Show the screenshot of (2e) and discuss the difference from the results.

以下圖形的光源皆來自(-3, -3, -3)directional light

(ka , kd , ks)分別表ambient/diffuse/specular strength of the objects。

這四張圖形的材質看起來都不一樣

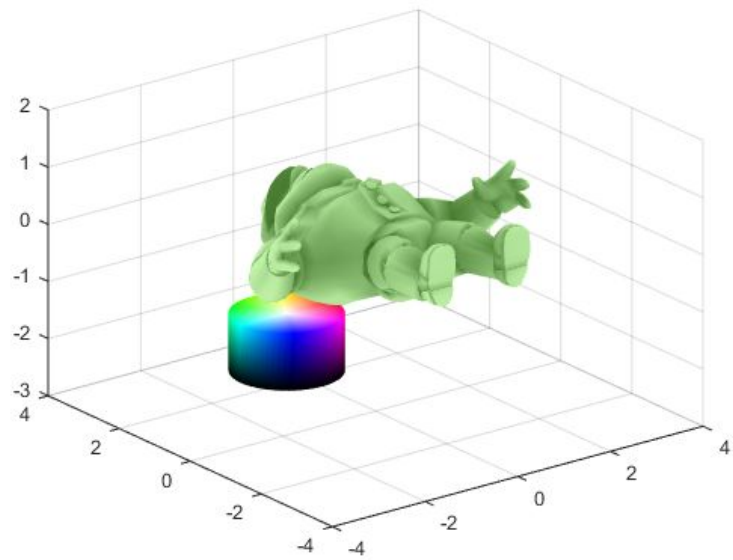
第一張圖看起來跟原始圖形看起來幾乎一樣，感覺沒有改變材質過

第二張到第四張圖的金屬感看起來隨著編號增加有遞增的感覺

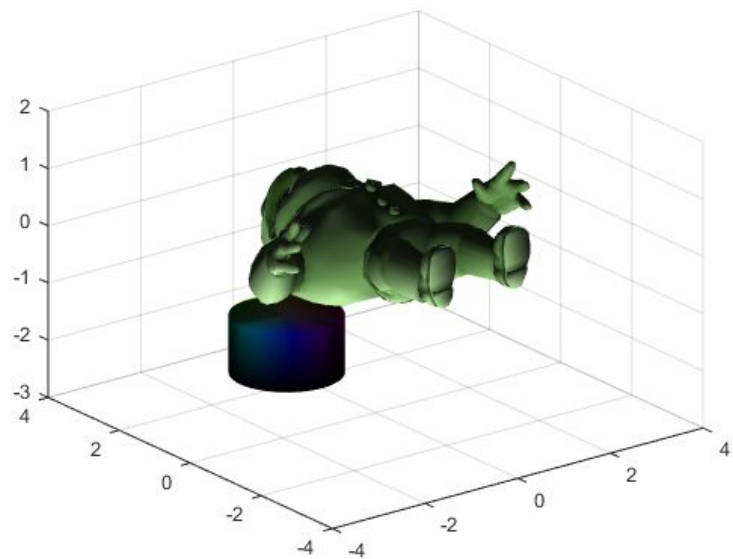
第三章黑黑的看起來像鋼的材質

第四張仍保持原色，但是金屬感、反光程度增加，且一開始定義的face仔細看也看得出一片一片的

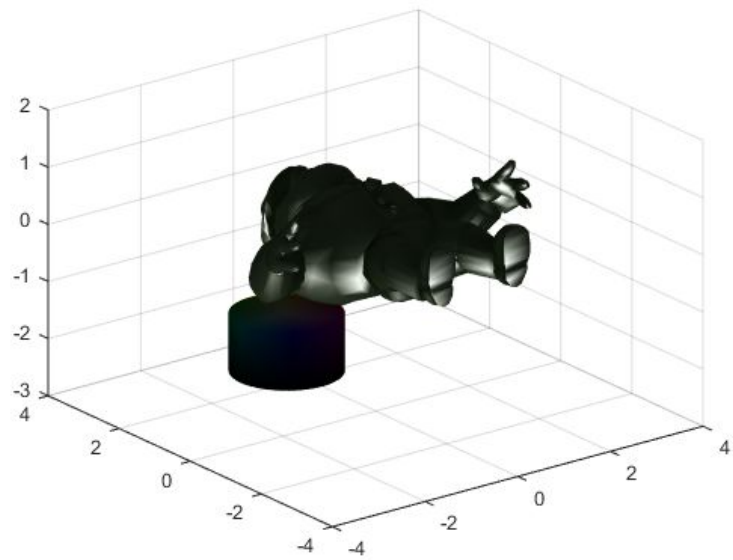
$(k_a, k_d, k_s) =$
 $(1.0, 0.0, 0.0)$



$(k_a, k_d, k_s) =$
 $(0.1, 1.0, 0.0)$



$(k_a, k_d, k_s) =$
 $(0.1, 0.1, 1.0)$



$(k_a, k_d, k_s) =$
 $(0.1, 0.8, 1.0)$

