

一、討論各個 textrue mode 的差異

1.Observe Linear / Nearest magnification filtering

(1)magnification：textrue image 較小，貼圖區域較大，因此需要放大 textrue。

(2)GL_NEAREST：選擇最臨近的畫素的顏色，magnification（放大）時：由於多個片元會在同一個紋理畫素上面取值，所以最終得到的圖片顆粒度很大，會有鋸齒。

(3)GL_LINEAR：根據鄰近四個的畫素點的颜色值，做線性的插值計算，得到最終的颜色。magnification（放大）時：不會產生鋸齒，會比較平滑。

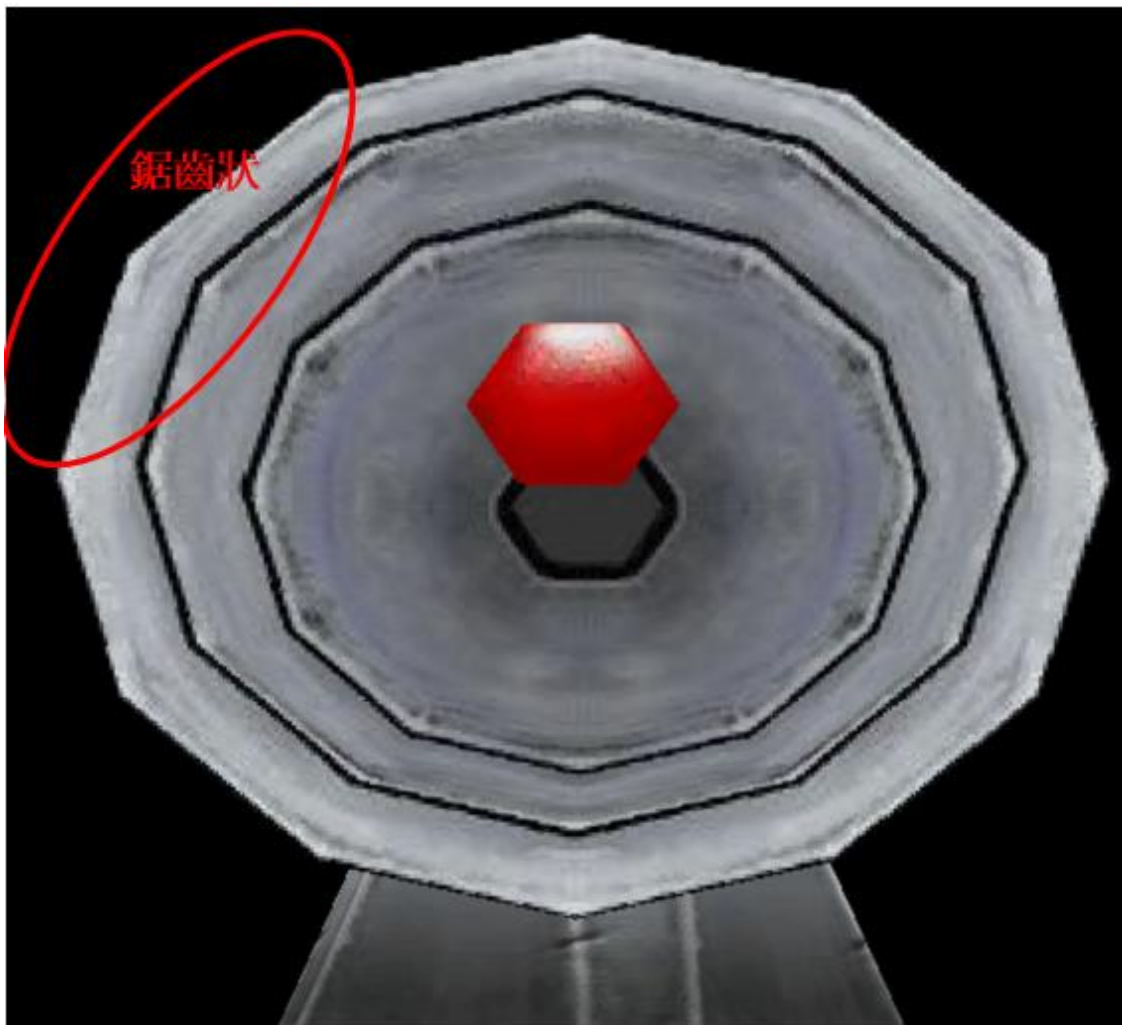
(4)HW3 當中，按 f 可以切換 Linear / Nearest。

(5)結果：

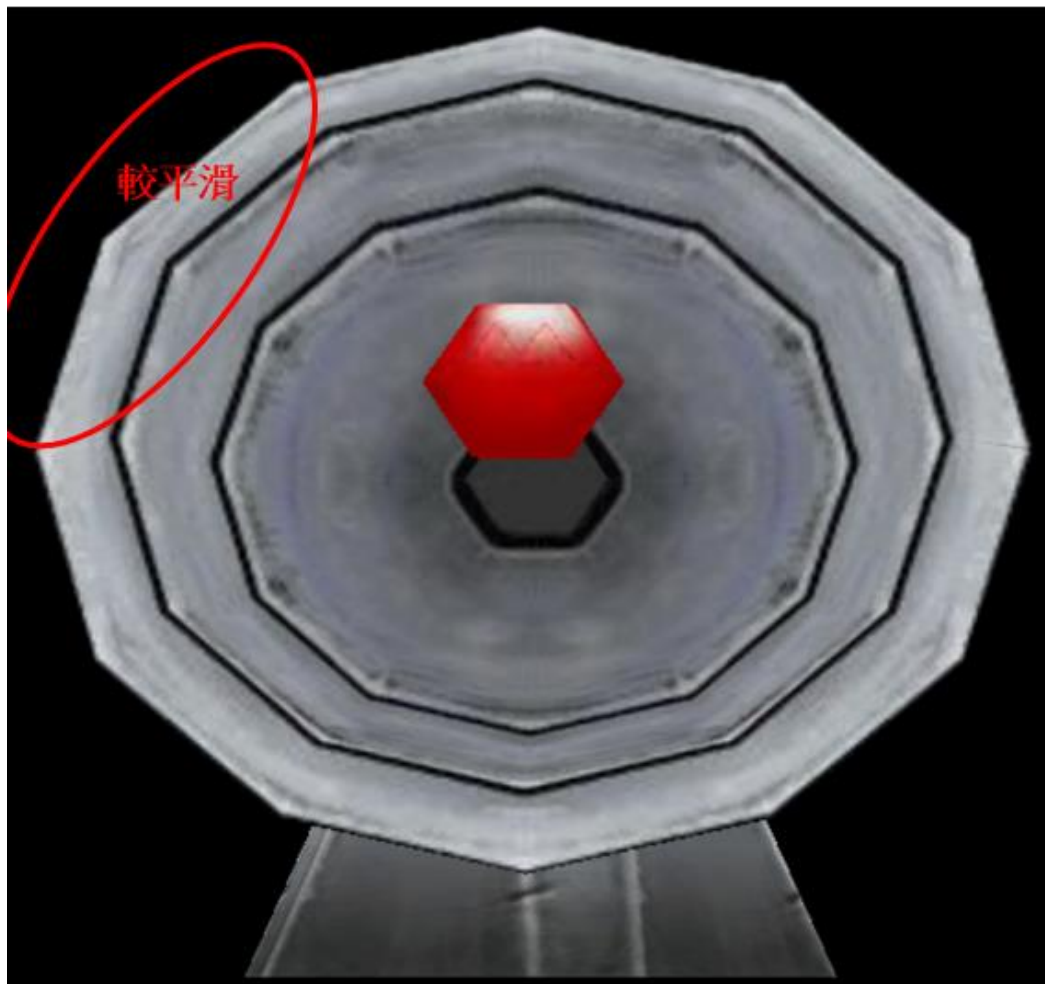
Nearest magnification filtering 在輪廓上可以看出會有鋸齒，以下截圖幾組比較明顯的。

①第一組：

Nearest magnification filtering



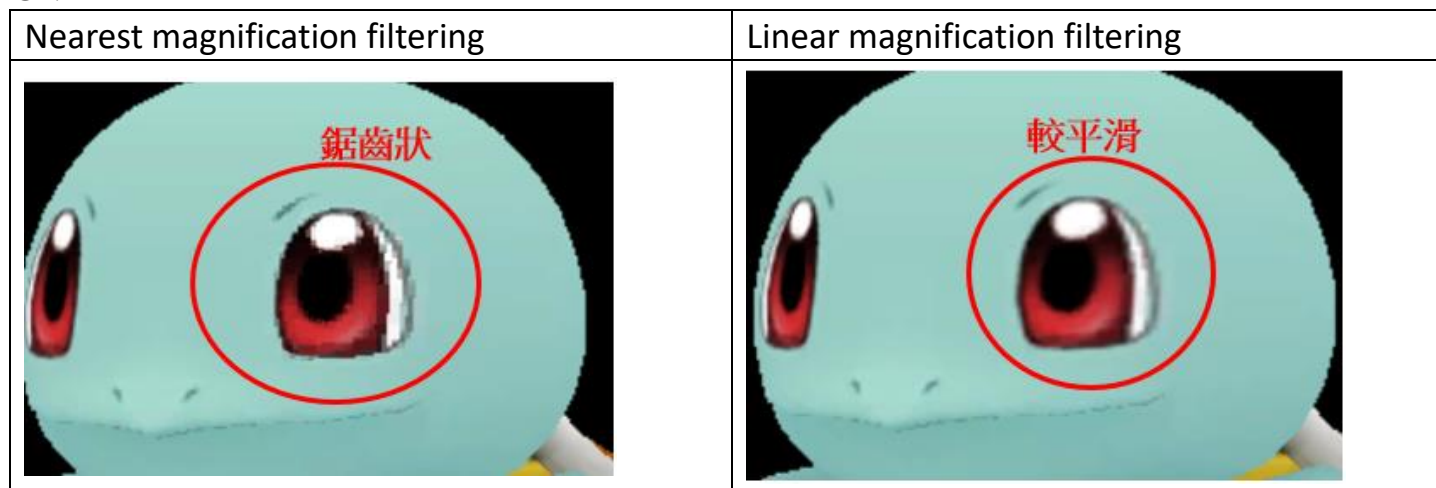
Linear magnification filtering



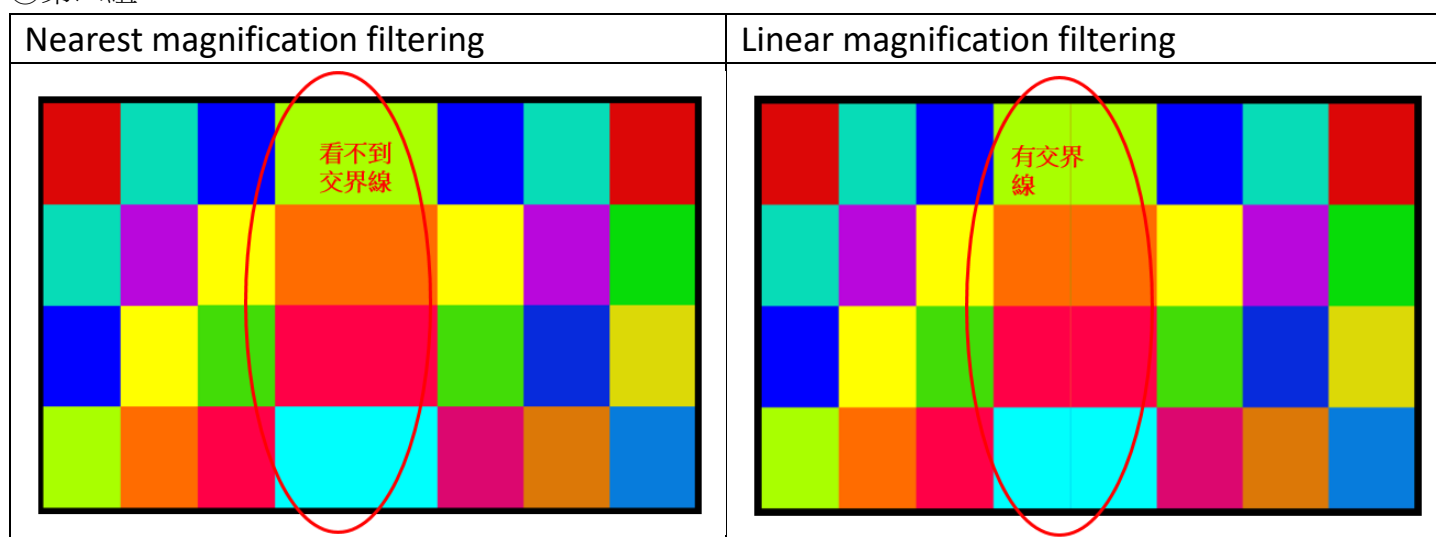
②第二組

Nearest magnification filtering	Linear magnification filtering
<p>鋸齒狀</p>	<p>較平滑</p>
<p>鋸齒狀</p>	<p>較平滑</p>

③第三組



④第四組



2.Observe Linear / Nearest minification filtering

(1) minification : texture image 較大，貼圖區域較小，需要縮小 texture 顯示。

在做放大和縮小的操作的時候的具體的策略如下：

(2)GL_NEAREST：選擇最臨近的畫素的顏色。

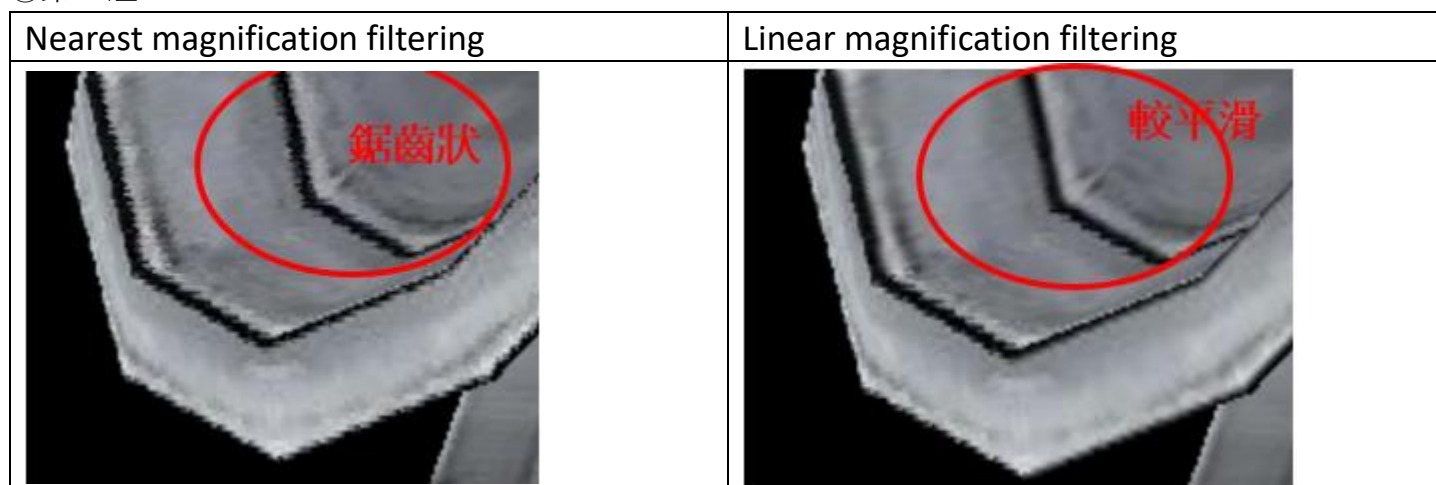
(3)GL_LINEAR：根據鄰近四個的畫素點的颜色值，做線性的插值計算，得到最終的颜色。

(4)HW3 當中，按 g 可以切換 Linear / Nearest。



(5)結果：

Nearest minification filtering 在輪廓上可以看出會有鋸齒，以下截圖幾組比較明顯的。

①第一組



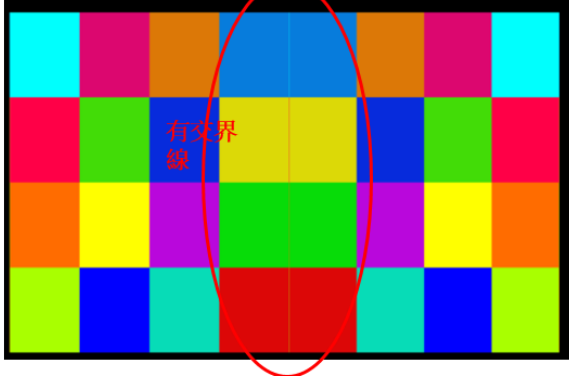
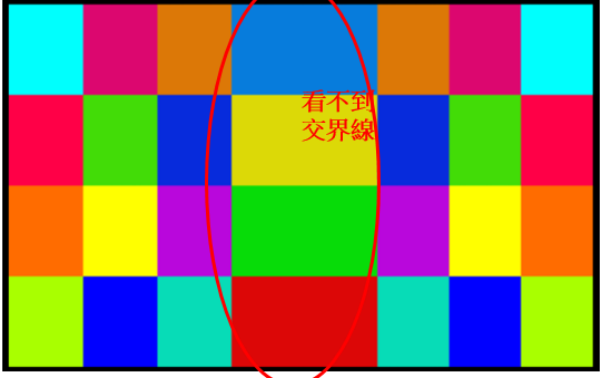
②第二組

Nearest magnification filtering	Linear magnification filtering
 <p>鋸齒狀</p>	 <p>較平滑</p>

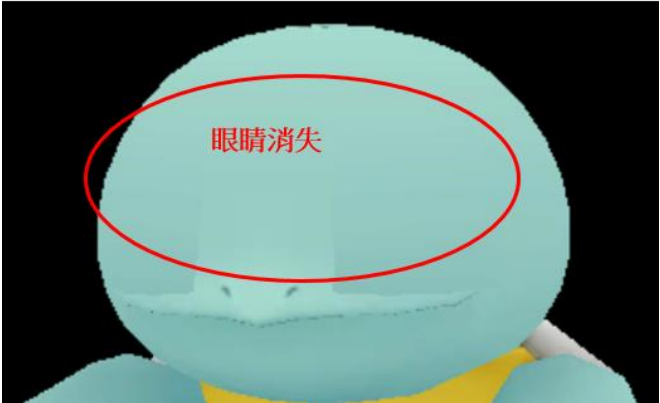

3.Observe REPEAT / CLAMP_TO_EDGE on texture s/t wrap

- (1)REPEAT : texture 重複鋪設。
- (2)CLAMP_TO_EDGE : texture 邊緣進行拉伸。
- (3) HW3 當中，按 h 可以切換 REPEAT / CLAMP_TO_EDGE。
- (4)結果：

①第一組

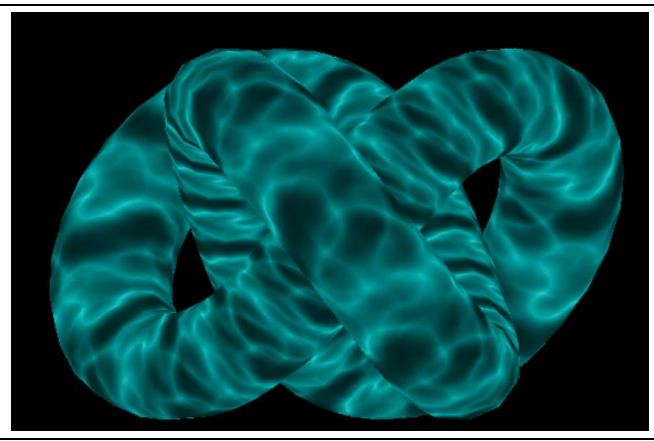
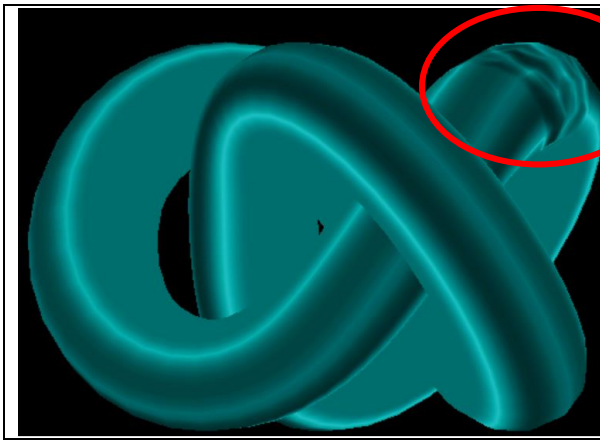
CLAMP_TO_EDGE	REPEAT
 <p>有交界線</p>	 <p>看不到交界線</p>

②第二組

CLAMP_TO_EDGE	REPEAT
 <p>眼睛消失</p>	

③第三組

CLAMP_TO_EDGE	REPEAT
---------------	--------



二、Load texture 的流程

1.如何找到對應的材質檔名

應在 `traverseColorModel` function 裡：

`texName` 即為材質檔名。

```
while (group)
{
    // If there exist material in this group
    if (strlen(m.obj->materials[group->material].textureImageName) != 0)
    {
        string texName = "../../../TextureModels/" + string(m.obj->materials[group->material].textureImageName);

        cout << texName << endl;
    }
}
```

2.如何 load texture 進 memory

<code>glGenTextures(1, &m.group[curGroupIdx].texNum);</code>	產生一個 texture memory 在 <code>m.group[curGroupIdx].texNum</code> 的位置。
<code>glBindTexture(GL_TEXTURE_2D, m.group[curGroupIdx].texNum);</code>	將 <code>m.group[curGroupIdx].texNum</code> 與 <code>GL_TEXTURE_2D</code> bind 在一起，告訴 OpenGL 有個 texture memory 在 <code>m.group[curGroupIdx].texNum</code> 。
<code>glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, m.group[curGroupIdx].infoHeader.Width, m.group[curGroupIdx].infoHeader.Height, 0, GL_BGR, GL_UNSIGNED_BYTE, m.group[curGroupIdx].image);</code>	透過 <code>GL_TEXTURE_2D</code> 將 image 資訊存到 <code>m.group[curGroupIdx].texNum</code> 這個位置。

3.如何把 memory 裡的 texture 搬運到 GPU

<pre>glBindBuffer(GL_ARRAY_BUFFER, positionBufferHandle); glBufferData(GL_ARRAY_BUFFER, sizeof(float)*m.group[i].numTriangles*9, m.group[i].vertices, GL_DYNAMIC_DRAW); glBindBuffer(GL_ARRAY_BUFFER, normalBufferHandle); glBufferData(GL_ARRAY_BUFFER, sizeof(float)*m.group[i].numTriangles * 6, m.group[i].texcoords, GL_DYNAMIC_DRAW);</pre>	
<p><code>drawModel</code> function 裡這幾行:</p> <ol style="list-style-type: none"> 1.先指定綁定在哪種容器上，再將先前產生的 <code>buffer</code> 綁定到適當的容器中。 2.指定容器，將後面參數資料填入指定容器。 	

m.group[i].texcoords 是要傳遞的資料。	
<pre>// Map index 0 to the position buffer glBindBuffer(GL_ARRAY_BUFFER, positionBufferHandle); glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, NULL); // Map index 1 to the color buffer glBindBuffer(GL_ARRAY_BUFFER, normalBufferHandle); glVertexAttribPointer(1, 2, GL_FLOAT, GL_FALSE, 0, NULL);</pre>	<p>將資料填入 memory 等待 Vertex Fetching 到 GPU 中的 layout。</p> <p>第 1 個 parameter：指定要進入 shader 的哪個 location。</p>
<pre>layout (location = 0) in vec3 v_vertex; layout (location = 1) in vec2 v_texcoord; uniform mat4 um4mvp; out vec2 f_texcoord; void main() { f_texcoord = v_texcoord; gl_Position = um4mvp * vec4(v_vertex, 1.0); }</pre>	<p>shader.vert</p> <p>GPU 在開始執行 Vertex Shader。</p> <p>把 f_texcoord 傳給 shader.frag。</p>
<pre>in vec2 f_texcoord; out vec4 fragColor; uniform sampler2D tex; void main() { // sample color from texture with uv coordinates(f_texcoord) fragColor = vec4(texture(tex, f_texcoord).rgb, 1.0); }</pre>	<p>shader.frag</p> <p>處理 fragment 的 texture。</p>