Computer Graphics

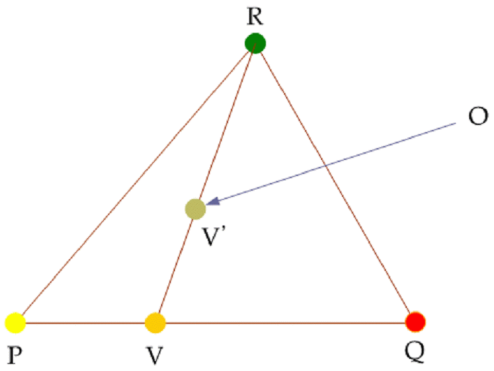
Prof. Jibum Kim

Department of Computer Science & Engineering

Incheon National University

2D Barycentric coordinates and color interpolation

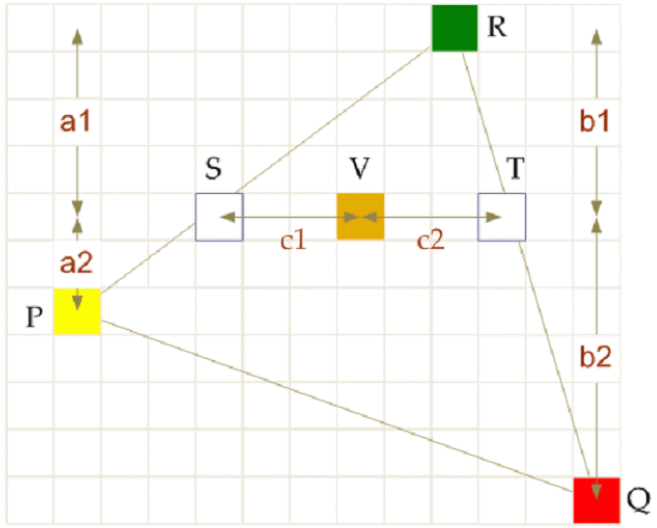
- 선분에서 사용한 무게 중심 좌표의 개념을 2D polygon인 triangle에도 적용 가능하다
- Triangle 내부의 모든 점 V'은 다음과 같이 표현 가능



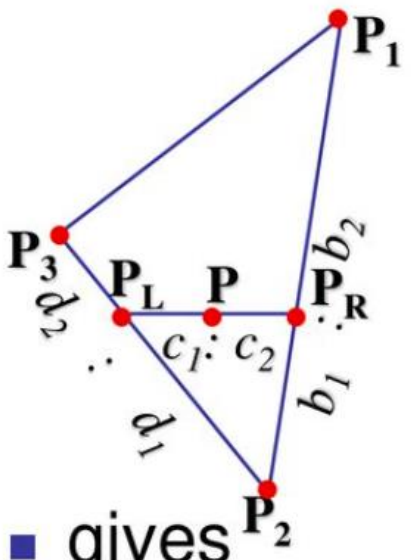
- V'= αP+ βQ+γR, $\alpha+\beta+\gamma=1$,
- 단, $0 \leq \alpha, \beta, \gamma \leq 1$

- 여기서 (α, β, γ)을 삼각형의 무게 중심 좌표 (barycentric coordinates)라 한다

- $1. S = \frac{2}{3}P + \frac{1}{3}R$
- $2. T = \frac{4}{10}Q + \frac{6}{10}R$
- $3. V = \frac{1}{2}S + \frac{1}{2}T$
- 최종: $V = \frac{1}{3}P + \frac{1}{5}Q + \frac{14}{30}R$
- (α, β, γ)=(1/3, 1/5, 14/30)



- combining



$$P = \frac{c_2}{c_1 + c_2} \cdot P_L + \frac{c_1}{c_1 + c_2} \cdot P_R$$

$$P_L = \frac{d_2}{d_1 + d_2} P_2 + \frac{d_1}{d_1 + d_2} P_3$$

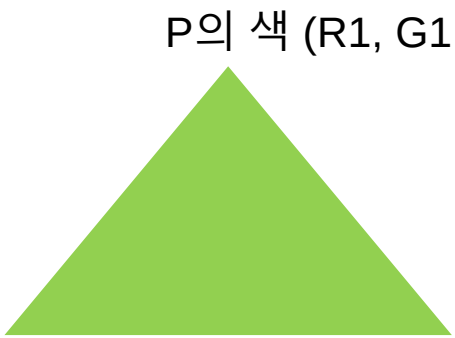
$$P_R = \frac{b_2}{b_1 + b_2} P_2 + \frac{b_1}{b_1 + b_2} P_1$$

- gives

$$P = \frac{c_2}{c_1 + c_2}\left(\frac{d_2}{d_1 + d_2} P_2 + \frac{d_1}{d_1 + d_2} P_3\right) + \frac{c_1}{c_1 + c_2}\left(\frac{b_2}{b_1 + b_2} P_2 + \frac{b_1}{b_1 + b_2} P_1\right)$$

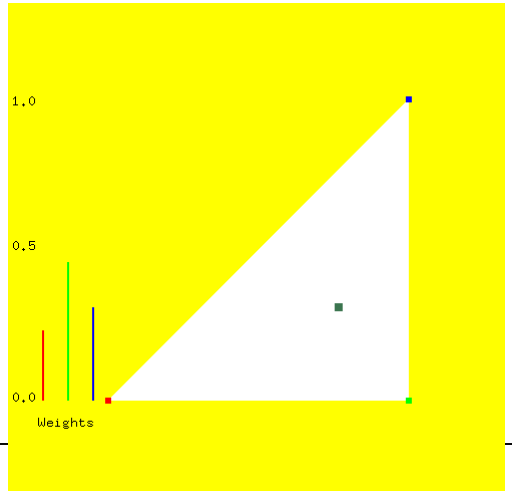
- 예: If P=(0, 0, 0), Q=(20, 0, 0), and R=(20, 30, 0), does the point V=(10, 20, 0) lie on the triangle PQR?
- V= αP+ βQ+γR, α+ β + γ =1,
- α =1/2, β =-1/6, γ =2/3
- As the β does not lie between 0 and 1, we conclude that V does not lie on the triangle PQR

- 삼각형 내부의 점에서의 색 보간
- 앞에서와 같이 삼각형 내부의 점 V'는 다음과 같이 표현 가능
- V'= αP+ βQ+γR, $\alpha+\beta+\gamma=1$,
- 단, $0 \leq \alpha, \beta, \gamma \leq 1$



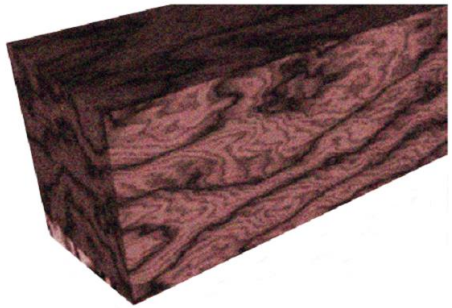
- V'에서의 색 보간 =
- α*(R1, G1, B1) + β(R2, G2, B2) + γ (R3, G3, B3)

- 삼각형의 색 보간을 무게 중심 좌표로 짠 OpenGL 코드 예 키보드로 삼각형 내부의 점 이동 가능
- keyboard callback 함수 이용
- 삼각형 변위에 있으면 barycentric coordinates의 weight (왼쪽 막대그래프)가 어떻게 되는지 확인해 보자
- https://www.dropbox.com/s/9axjetrn6f537ov/color_interpolation.txt?dl=0



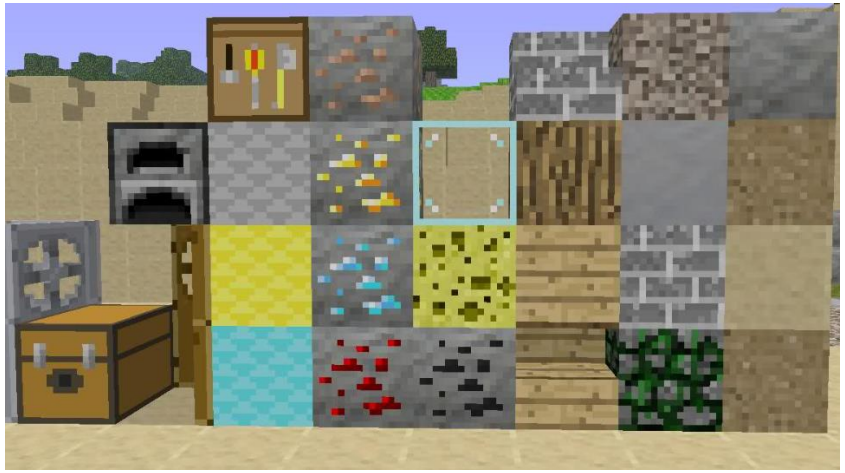
- **Texture (Texture mapping)**

- 아래 그림과 같 목재 표면을 모델링하는 데에는 무수히 많은 polygon이 필요하다. 아주 작은 부분별로 다른 색을 지녔기 때문이다
- 이와 같은 3차원 굴곡을 모델링하는 대신에 목재 표면에 대한 영상 (image)를 polygon에 입혀버리는 것을 어떨까?
- 이렇게 되면 복잡한 기하학적 모델링 대신에 빠른 시간에 표면에 굴곡을 나타낼 수 있다
- 이때 사용된 영상을 Texture (텍스처)라고 한다
- 일반적으로 2차원 영상을 사용하지만 1, 2, 3차원 영상 모두를 texture로 사용할 수 있다

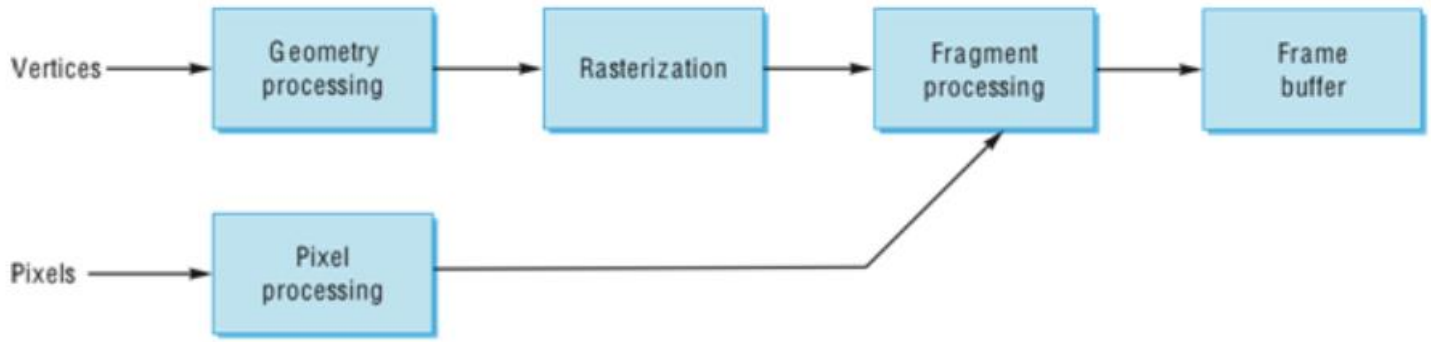


- **Texture mapping**
- **Take a picture of a real orange, scan it, and “paste” onto simple geometric model**
- **This process is known as texture mapping**
- **Use images to fill inside of polygons**

- **Texture mapping의 예**



- **Where does mapping take place?**
- **Mapping techniques are implemented at the end of the rendering pipeline**



- **Texture basics**

- Typically, a texture is an image, which is applied to a polygon (or mesh)
- The pixels in the texture are called texels, each texel storing color values
- The texture itself can be an external image which is imported into an OpenGL program or one generated internally by the program itself
- The former os called an external texture while the latter is called a procedural (synthetic) texture
- Once loaded though there is no difference between the two

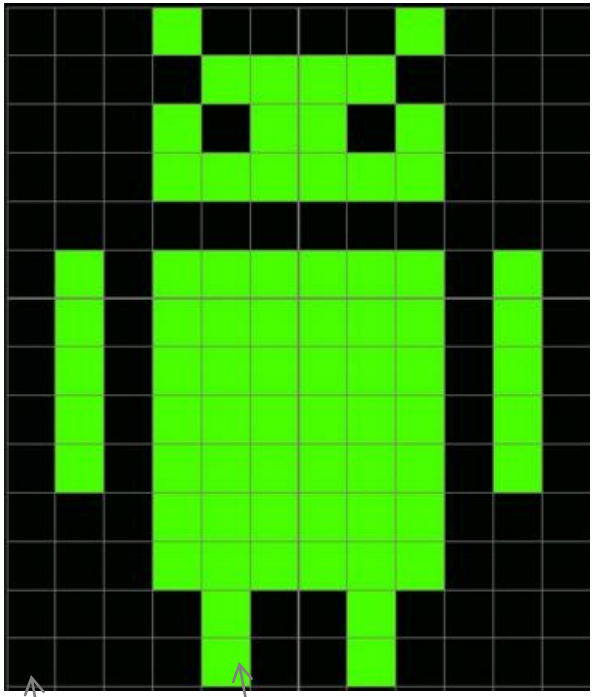
- Chapter12/TexturedSquare/
- External image인 textur와 procedural (synthetic) texture
- Texture images는
- ExperimentSource/Textures 에 있음

(a)

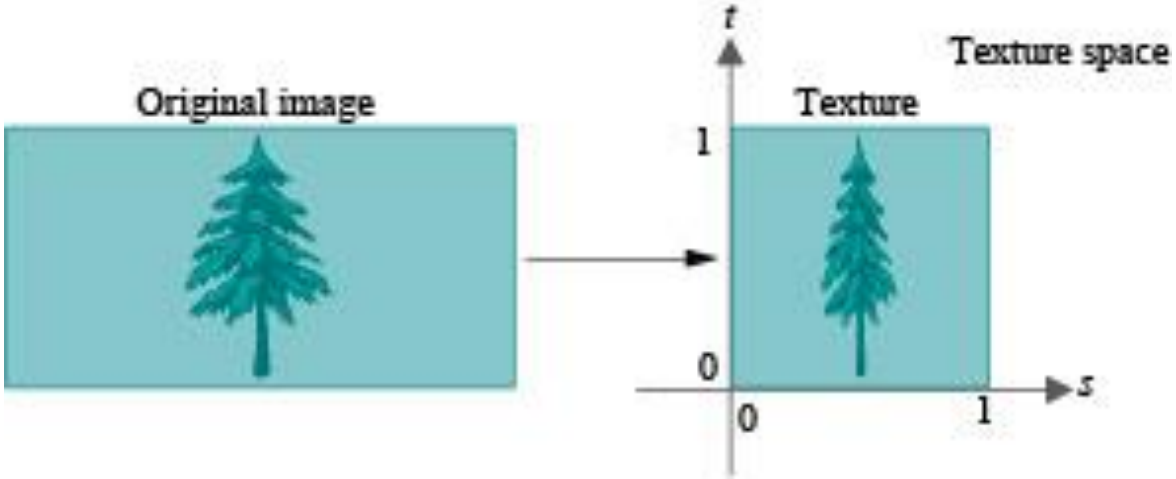
(b)

Figure 12.5: The two textures of `texturedSquare.cpp`: shuttle launch (external, from NASA) and chessboard (procedural).

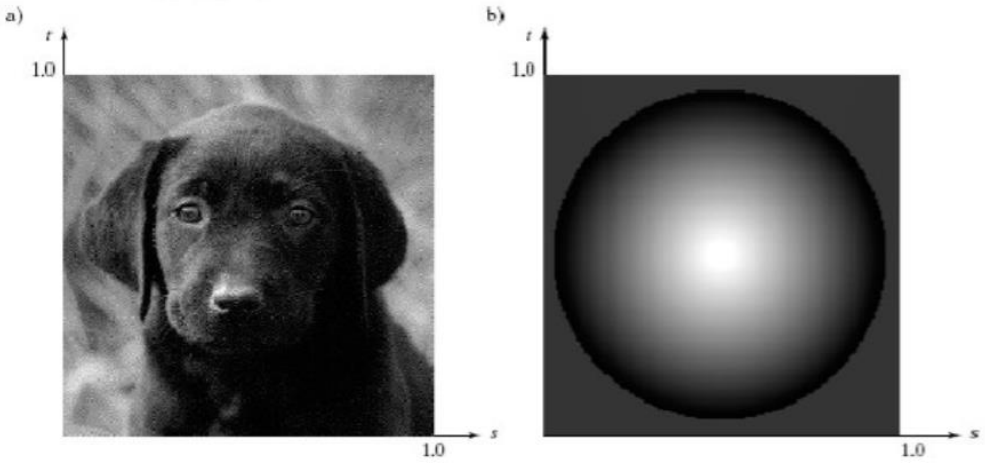
- 수업시간에 다루는 Texture는
- 주로 bitmap image 사용
- Bitmap image 파일 (.bmp, .jpg, .png..)
- 열어 보면 2D array 로 구성됨
- 기본 단위 (사각형 하나) : Pixel
- E.g., 12x14 size의 2D array
- Total 168개의 Pixel 있음
- 각 Pixel은 color값 저장 (0~255까지)
- (R, G, B), or (R, G, B, A)



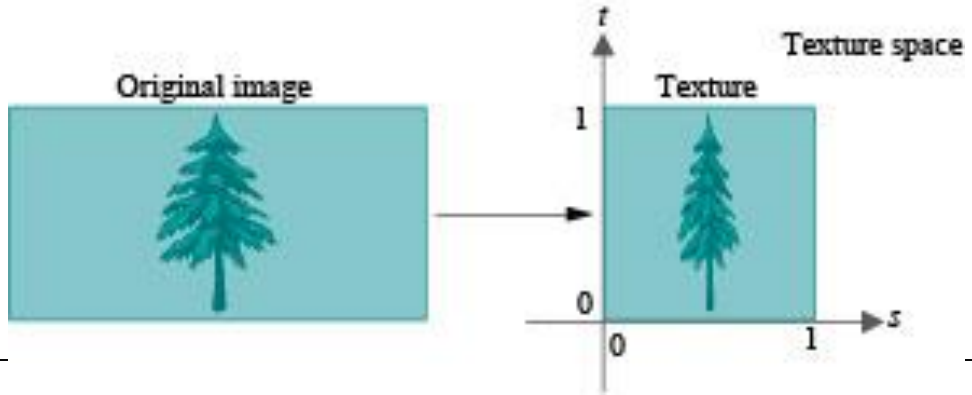
- A texture, once loaded, occupies the unit square with cornders at (0, 0), (1, 0), (1, 1), (0, 1) of an virtual plane called texture space
- This is regardless of whether the original texture image is equal-sided or not
- If it is not, then it is scaled to fit thre square as shown below
- The axes of texture space are denoted s and t



- The texture is a function texture (s, t) that produces a color or intensity value for each value of s and t between 0 and 1
- The most common sources of textures are bitmaps (a matrix like array of pixels). 왼쪽 아래 (external texture)
- 오른쪽 아래 (Artificial image formed according to some calculated function, synthetic texture)



- Texture도 영상의 일종이므로 배열을 사용하여 저장한다
- 화면의 기본요소를 픽셀(pixel, picture element)라 부르듯이 texture를 구성하는 배열 요소 각각을 텍셀 (texel, texture element)라 부른다
- The pixels in a texture are called texels, each texel storing color values
- Texture 좌표
- 좌하단: (0,0), 우하단: (1,0), 좌상단: (0,1), 우상단: (1,1)



- The first statement maps the vertex at (-10.0, -10.0, 0.0) of world space to the point (0.0, 0.0) of texture space
- The coordinate s of the mapped point in texture space are called the texture cooridnates of the vertex
- The mapping of the polygon vertices to texture space is interpolated throughout the polygon to obtain the texture map which is a map from a part of world space (polygon) to texture space

```
// Map the texture onto a square polygon.
glBegin(GL_POLYGON);
glTexCoord2f(0.0, 0.0); glVertex3f(-10.0, -10.0, 0.0);
glTexCoord2f(1.0, 0.0); glVertex3f(10.0, -10.0, 0.0);
glTexCoord2f(1.0, 1.0); glVertex3f(10.0, 10.0, 0.0);
glTexCoord2f(0.0, 1.0); glVertex3f(-10.0, 10.0, 0.0);
glEnd();
```

- **(a) replace every 1.0 in each glTexCoor2f() commands of texturedSqure.cpp with 0.5**

```
glBegin(GL_POLYGON);
glTexCoord2f(0.0, 0.0); glVertex3f(-10.0, -10.0, 0.0);
glTexCoord2f(0.5, 0.0); glVertex3f(10.0, -10.0, 0.0);
glTexCoord2f(0.5, 0.5); glVertex3f(10.0, 10.0, 0.0);
glTexCoord2f(0.0, 0.5); glVertex3f(-10.0, 10.0, 0.0);
glEnd();
```

Figure 12.9: Screenshot of Experiment 12.2.

- **(b) restore the original texturedsquare.cpp and delete the last vertex from the polygon so that the specification is that of a triangle**

```
glBegin(GL_POLYGON);
glTexCoord2f(0.0, 0.0); glVertex3f(-10.0, -10.0, 0.0);
glTexCoord2f(1.0, 0.0); glVertex3f(10.0, -10.0, 0.0);
glTexCoord2f(1.0, 1.0); glVertex3f(10.0, 10.0, 0.0);
glEnd();
```

Figure 12.10: Screenshot of Experiment 12.3: lower-right triangular half of texture not skewed.

- **(c) change the coordinates of the last vertex of the world-space triangle to (0.0, 10.0, 0.0)**
- **Interpolation is clearly evident now**
- **Parts of both launch and chessboard are skewed by texturing, as the triangle specified by texture coordinates is not similar to its world-space counterpart**

Figure 12.11: Screenshot of Experiment 12.3: lower-right triangular half of texture skewed.

```
glBegin(GL_POLYGON);
glTexCoord2f(0.0, 0.0); glVertex3f(-10.0, -10.0, 0.0);
glTexCoord2f(1.0, 0.0); glVertex3f(10.0, -10.0, 0.0);
glTexCoord2f(1.0, 1.0); glVertex3f(0.0, 10.0, 0.0);
glEnd();
```

- **(d) change the texture coordinates of the last vertex of the triangle to (0.5, 1.0)**
- **The texture are no longer skewed as the triangle in texture space is similar to the one being textured**

Figure 12.12: Screenshot of Experiment 12.3: middle triangle of texture not skewed.

- **Texture maps**
- **(a) page 15 (b) page 16**
- **(c) page 17 (d) page 18**

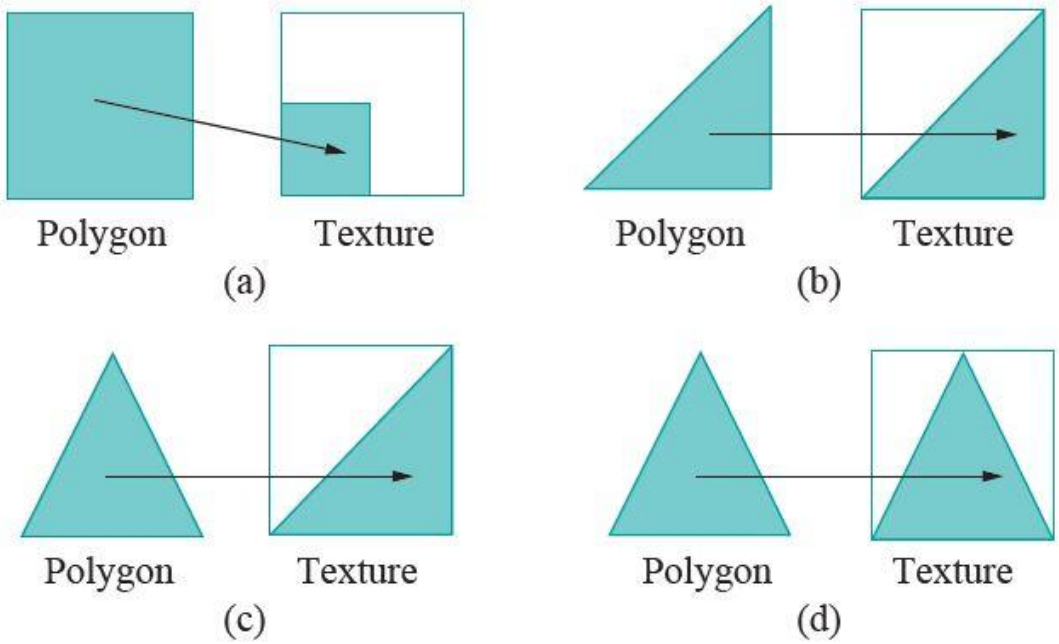


Figure 12.13: Texture maps.

- **Texture in OpenGL**

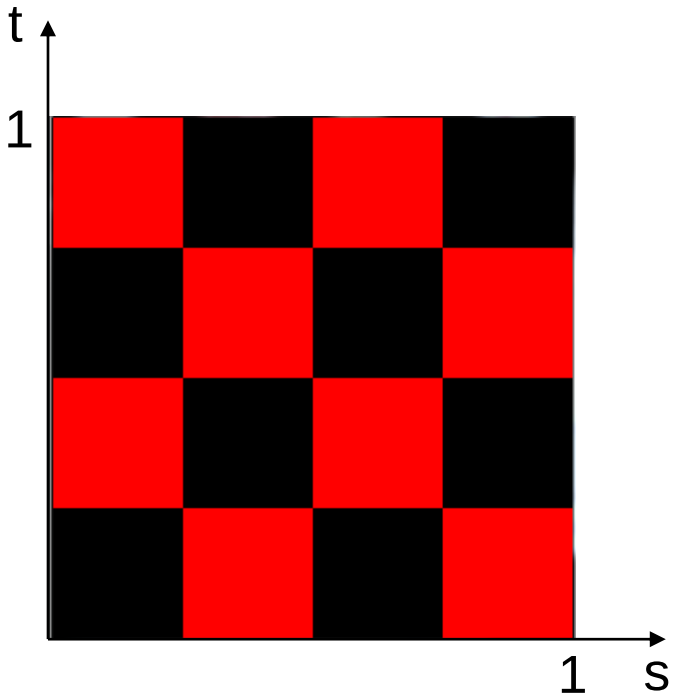
- **Enable Texturing (Texture 기능 활성화, 비활성화)**

```
void glEnable(GLenum mode);
void glDisable(GLenum mode);
```

- **Mode에 가능한 것들: GL_TEXTURE_1D, GL_TEXTURE_2D, GL_TEXTURE_3D**
- **예) glEnable(GL_TEXTURE_2D);**

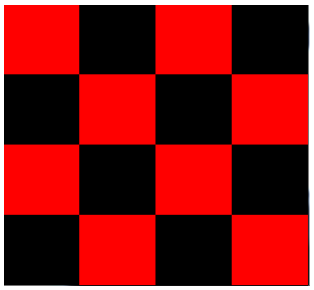
- Texture는 2D array로 직접 만들 수도 있다
- Texture의 기본 단위는 Texel 이라 한다. Texel의 R, G, B,는 0-255 사이
- 0: darkest, 255:brightest
- checkboard texture, WIDTH=4, HEIGHT=4
- Total 16 texels

```
for(s = 0; s < WIDTH; s++) {
      for(t = 0; t < HEIGHT; t++) {
         GLubyte Intensity = ((s + t) % 2) * 255;
         MyTexture[s][t][0] = Intensity;   //Red
        MyTexture[s][t][1] = 0;             //Green
        MyTexture[s][t][2] = 0;             //Blue
      }
   }
```

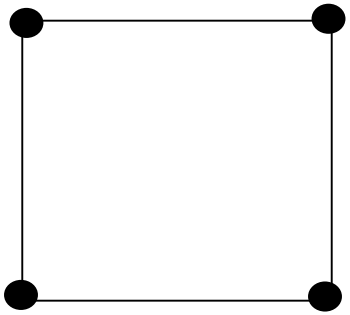


- **Checkboard texture example**
- **Width=4, Height=4, 4x4의 2D array인 아래와 같은 texture생성 후 사각형 polygon에 mapping시킴**

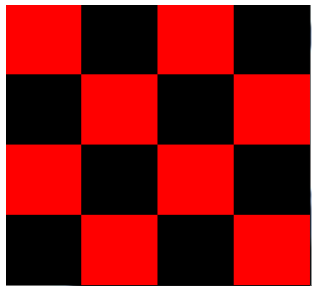
```
glBegin(GL_QUADS);
    glTexCoord2f(0.0, 0.0); glVertex3f(-1.0, -1.0, 0.0);
    glTexCoord2f(0.0, 1.0); glVertex3f(-1.0, 1.0, 0.0);
    glTexCoord2f(1.0, 1.0); glVertex3f(1.0, 1.0, 0.0);
    glTexCoord2f(1.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
  glEnd();
```



Texture



Polygon



Texture mapping

- **https://www.dropbox.com/s/7rpqh15heuqejgl/texture_0.txt?dl=0**

- glTexImage2D: Array로 표현된 image를 Texture image로 사용하기 위해서 필요한 함수

- 예) glTexImage2D(GL_TEXTURE_2D, 0, 3, WIDTH, HEIGHT, 0, GL_RGB,GL_UNSIGNED_BYTE, &MyTexture[0][0][0]);

// WIDTH: texture의 width, HEIGHT: texture의 height

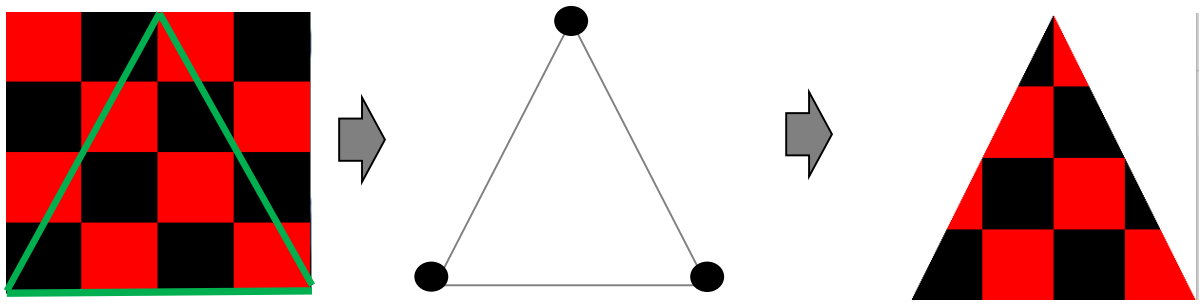
// 마지막 인자: 실제 texture image가 저장된 배열명

// 어떤 array 를 texture로 사용하는지 명시

https://www.opengl.org/sdk/docs/man/html/glTexImage2D.xhtml

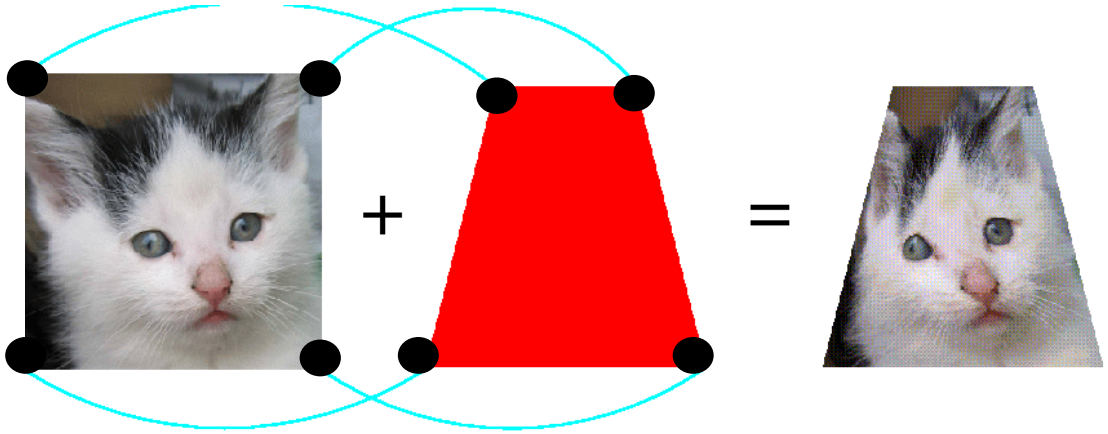
- 예: 원래 Texture에서 아래 초록색 부분과 같이 texture의 일부분 (삼각형 모양)만 선택한 후 삼각형 모양의 polygon에 texture mapping을 수행하여 보자 (가시 공간을 고려하여 좌표 설정)

```
glBegin(GL_POLYGON);
?
glEnd();
```



Texture Polygon Texture mapping

- we associate a point in texture space $P_i = (s_i, t_i)$ with each vertex V_i of the face using the function glTexCoord2f();
- Non-affine transformation: distortion may occur



- **Wrapping (Repeating and clamping textures)**

- **Restore the original texturedSquare.cpp and change the texture coordinates of the polygon as follows:**

```
glBegin(GL_POLYGON);
glTexCoord2f(-1.0, 0.0); glVertex3f(-10.0, -10.0, 0.0);
glTexCoord2f(2.0, 0.0); glVertex3f(10.0, -10.0, 0.0);
glTexCoord2f(2.0, 2.0); glVertex3f(10.0, 10.0, 0.0);
glTexCoord2f(-1.0, 2.0); glVertex3f(-10.0, 10.0, 0.0);
glEnd();
```

- **It seems that the texture space is tiled with the texture**

- **The texture seems repeated in every unit square of texture space with integer vertex coordinates**
- **As the world-space polygon is mapped to a 3x2 rectangle in texture space, it is painted with six copies of the texture**

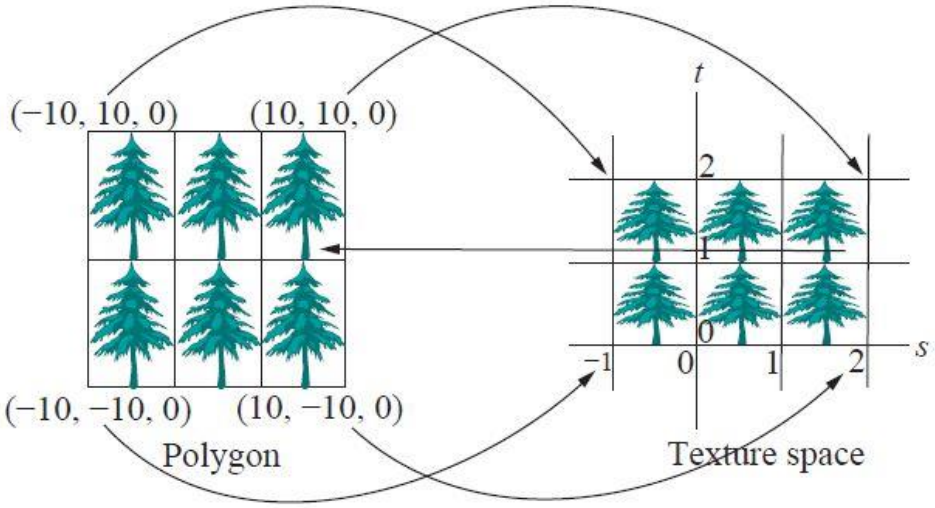
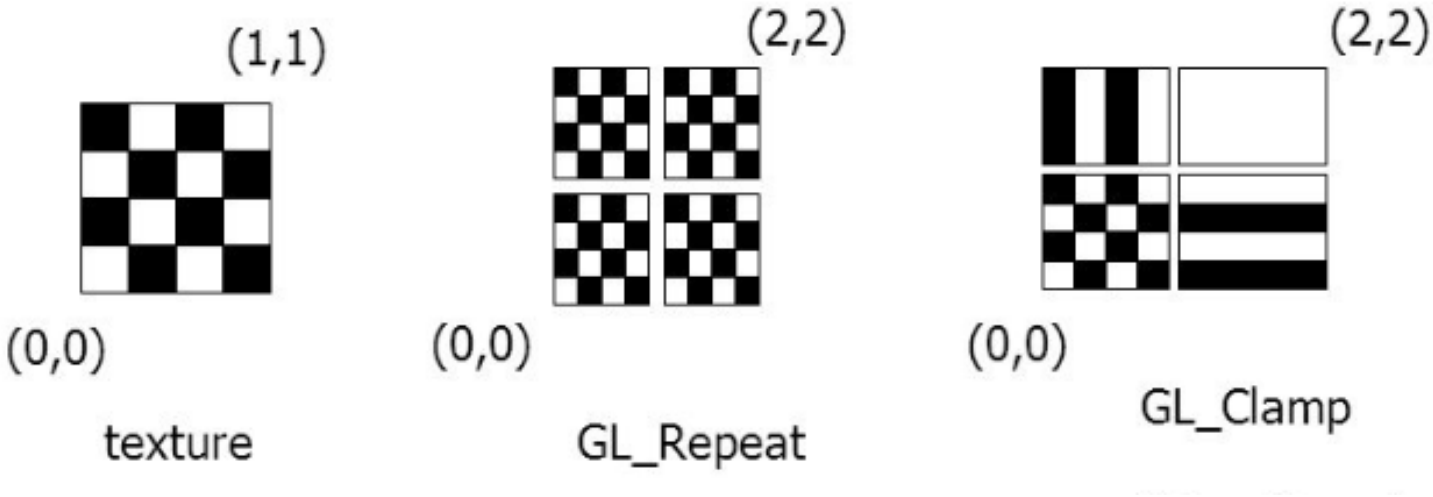


Figure 12.19: Tiling of texture space. The curved arrows indicate the texture map. The straight arrow indicates the painting of one tile onto a sub-rectangle of the polygon; other tiles similarly paint corresponding sub-rectangles.

- 원래 Texture의 범위, [s, t]모두 [0,1], 를 넘어서게 texture coordinates을 준다면 뒤의 옵션에 따라서 결정된다

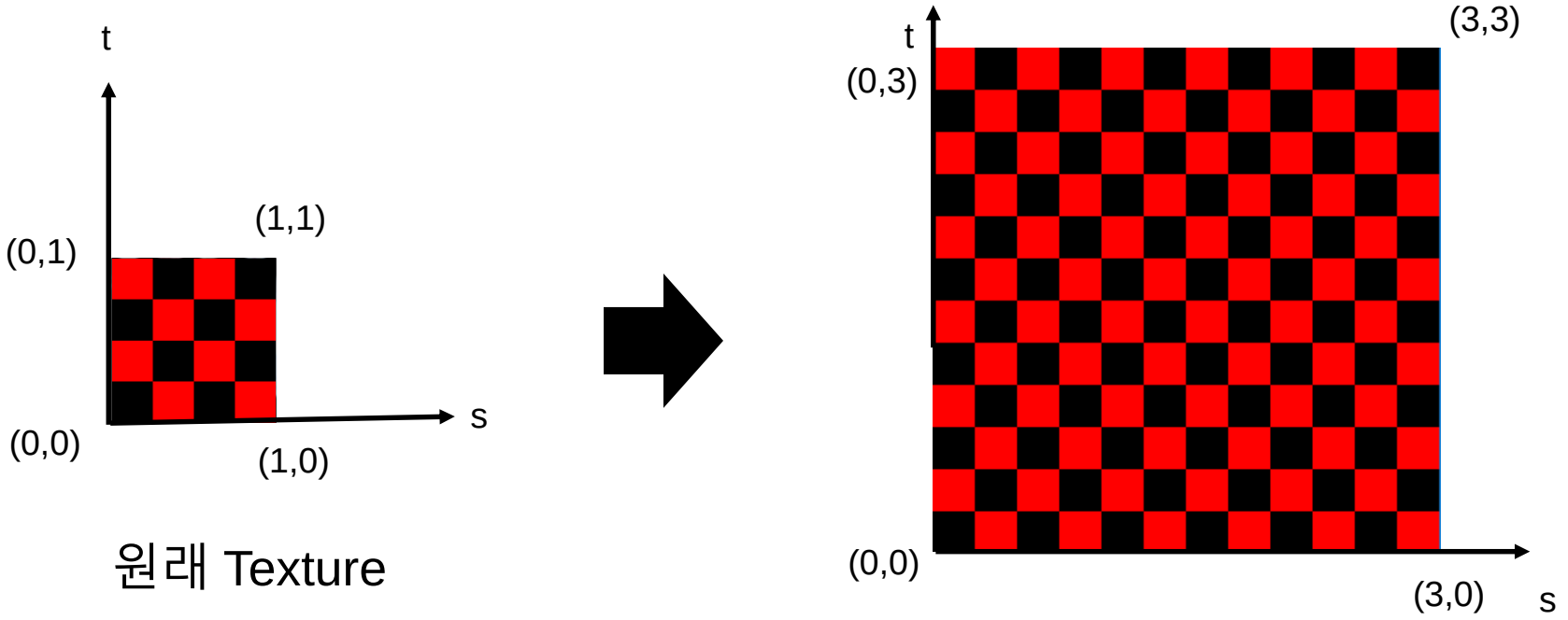


- 지금까지는 texture coordinates을 사용할 때에 s와 t 축 모두에서 [0, 1]사이의 값만을 사용하였다
- 만일 그 범위를 넘어서는 값을 주면 어떻게 될까?

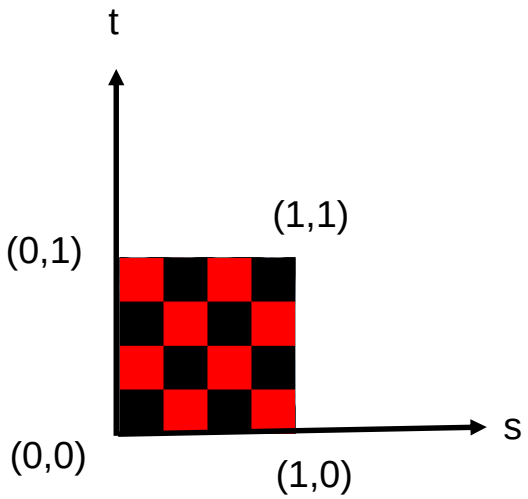
```
glBegin(GL_QUADS);
        glTexCoord2f(0.0, 0.0); glVertex3f(-1.0, -1.0, 0.0);
        glTexCoord2f(0.0, 3.0); glVertex3f(-1.0, 1.0, 0.0);
        glTexCoord2f(3.0, 3.0); glVertex3f(1.0, 1.0, 0.0);
        glTexCoord2f(3.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
    glEnd();
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
```

- **https://www.dropbox.com/s/84nopu5p4b4o40p/texture_1.txt?dl=0**

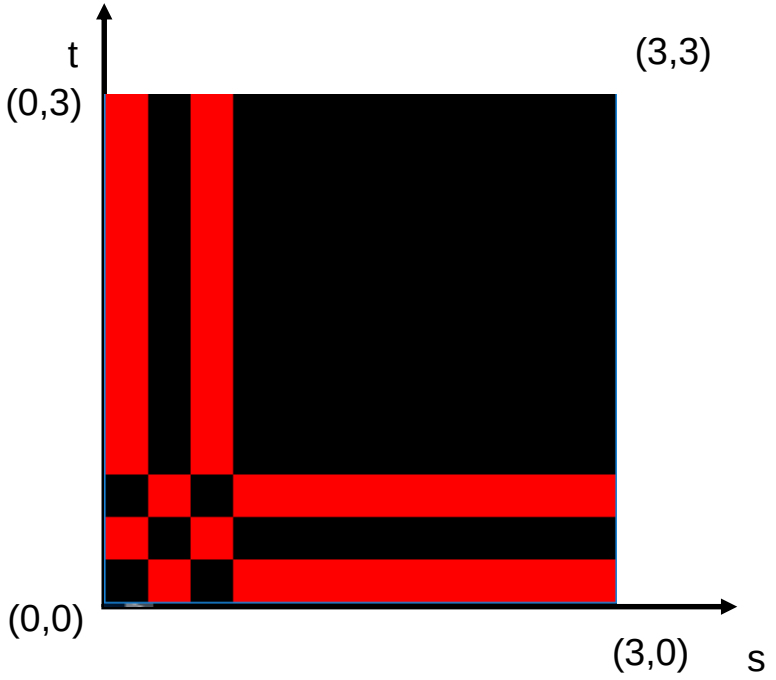
- **GL_REPERAT: 원래 S와 T축에서 [0, 1], 사이에 정의된 것을 S와 T축에서 반복하여 확장**



- **GL_CLAMP: 원래 S와 T축에서 [0, 1], 확장 영역의 색이 경계선의 색으로 고정 (clamping)된다**



원래 Texture



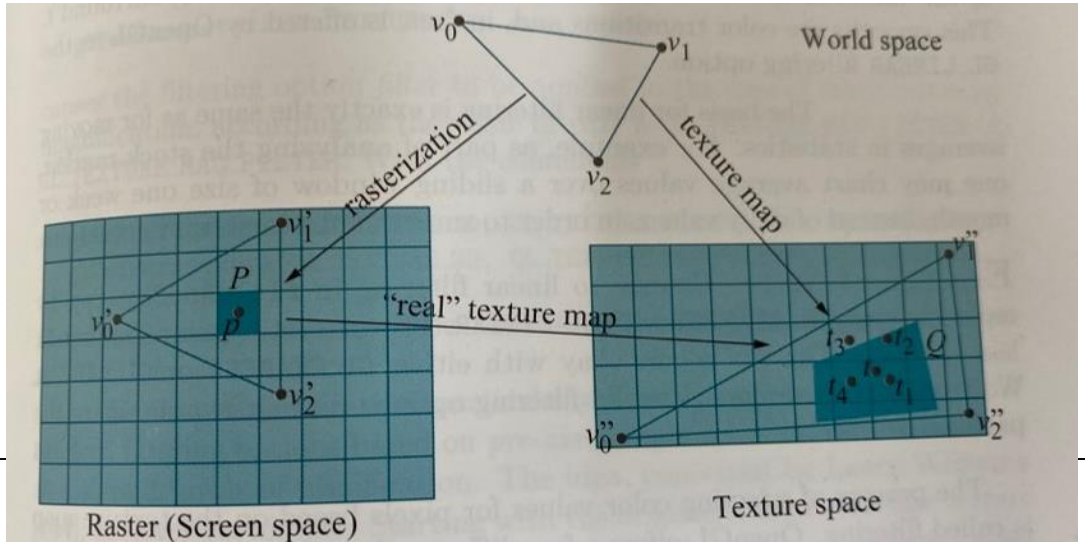
```
glTexParameterf(GL_TEXTURE_2D,
GL_TEXTURE_WRAP_S, GL_CLAMP);

glTexParameterf(GL_TEXTURE_2D,
GL_TEXTURE_WRAP_T, GL_CLAMP);
```

- **Aliasing and filtering in texture mapping**

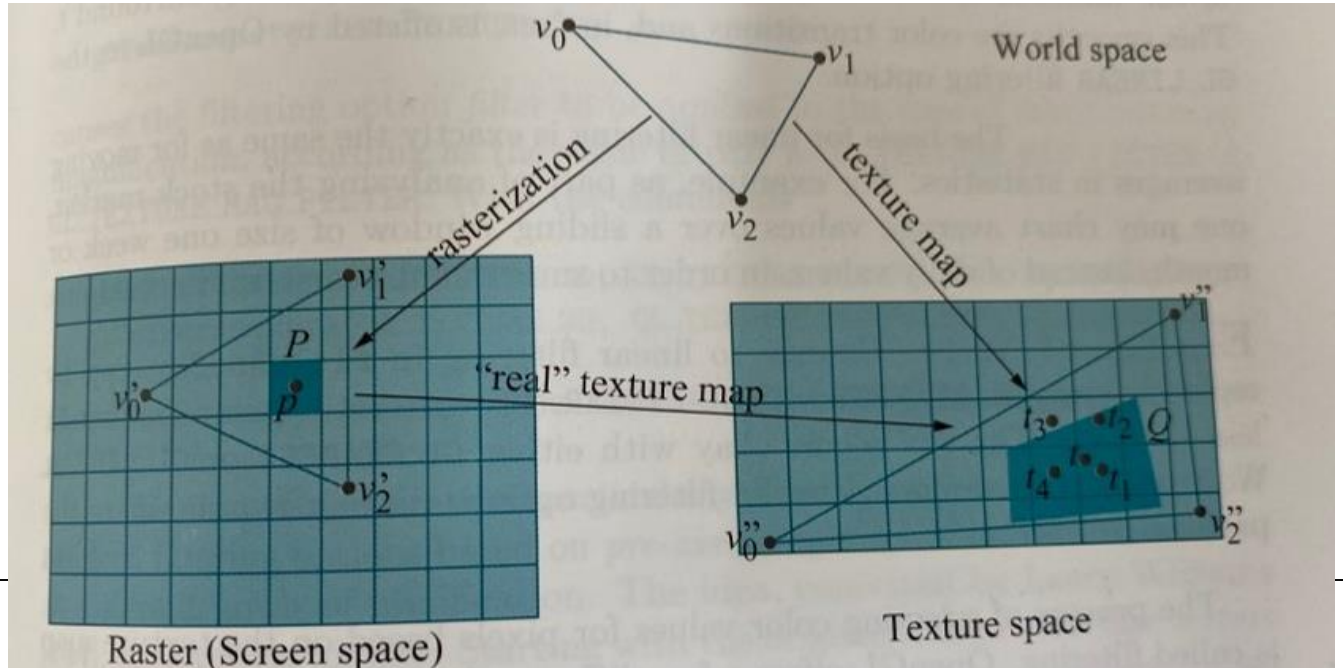
Aliasing problem in texture mapping

Once the polygon has been rasterized – i.e., its set of corresponding pixels determined – the texture map is unlikely to map pixels to texels in a one-to-one manner

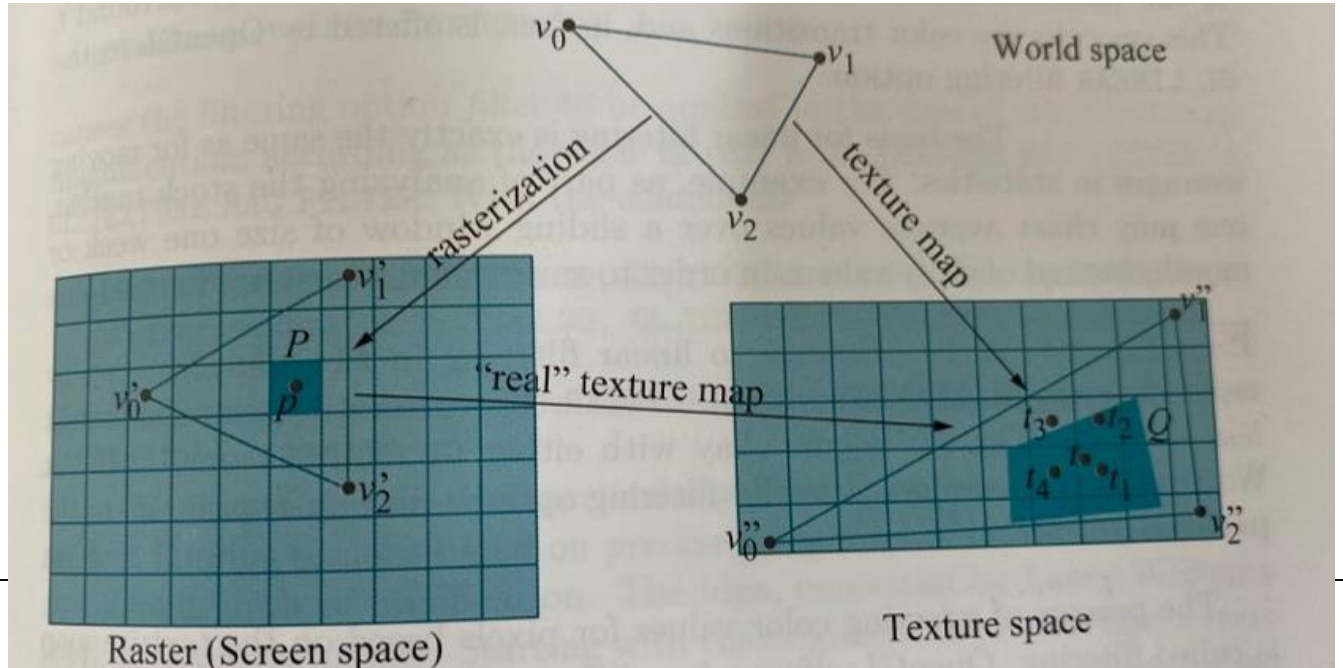
- 예: The triangle $v_0v_1v_2$ in world space is mapped to the raster triangle $v_0'v_1'v_2'$. It is also mapped to the texture space triangle $v_0''v_1''v_2''$ via the texture map
- These two maps induce the “real” texture map from raster to texture space, which takes pixels to texels
- The bold pixel P in the raster map to the bold quad Q in texture space. As Q intersects multiple texles, how should OpenGL choose color values for P? which texel should OpenGL pick to apply its particular color values to P?



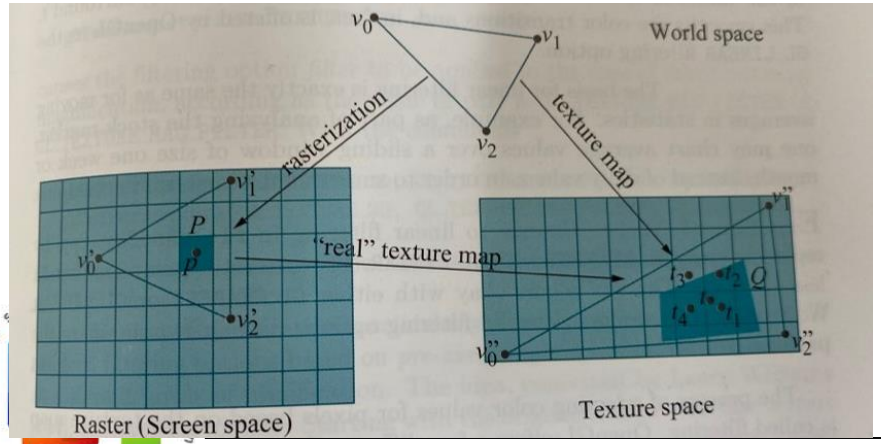
- **Solution 1: if the texture map takes the center p of P to the point t in texture space, then choose the texel whose center is nearest to t**
- **This is called filtering option specified by GL_NEAREST**

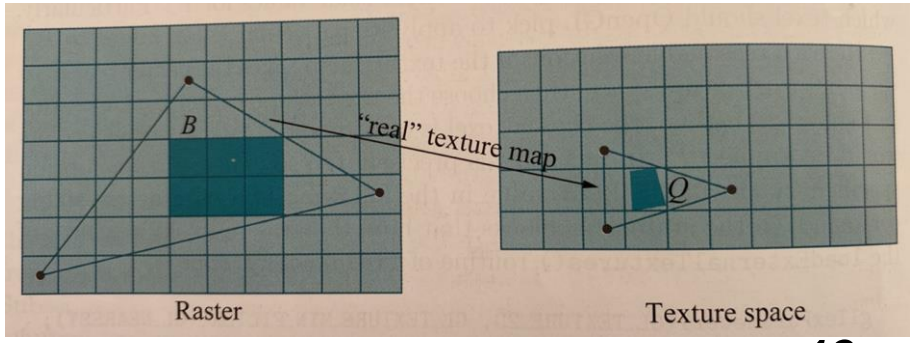


- **Solution 2: Instead of obtaining color values from just the one texel centered at t_1, take an average of the values at the four texles (t_1, t_2, t_3, and t_4) whose centers surround t**
- **This is called filtering option specified by GL_LINEAR**

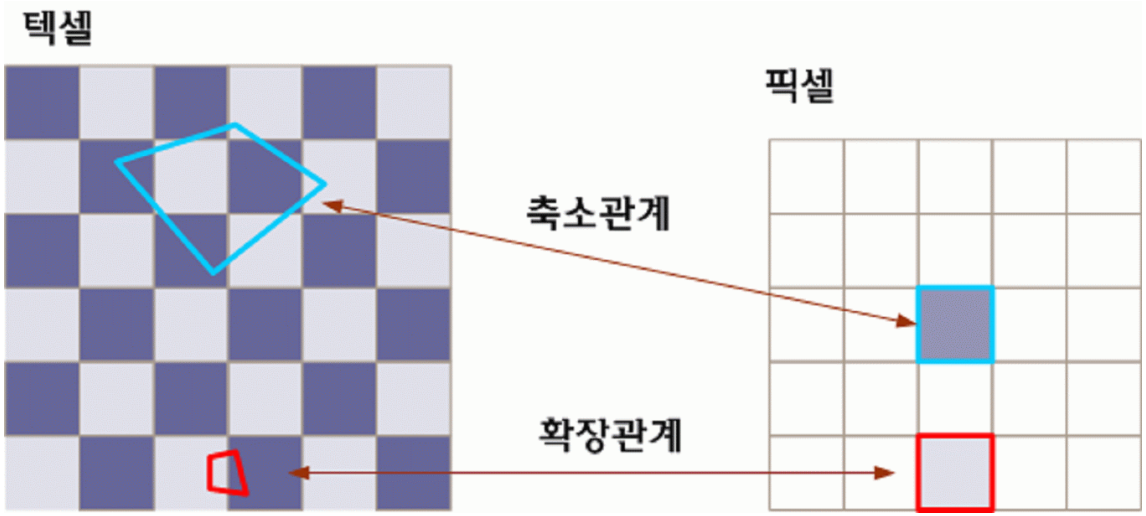


- The process of selecting color values for pixels based on the texture map is called **filtering**
- OpenGL offers a few different filtering option. It allow the user to trade between speed and output quality
- **Minification** (아래 왼쪽) occurs when a pixel is mapped onto multiple texel, while **magnification** (아래 오른쪽) is when many pixels map onto a single tex as shown below



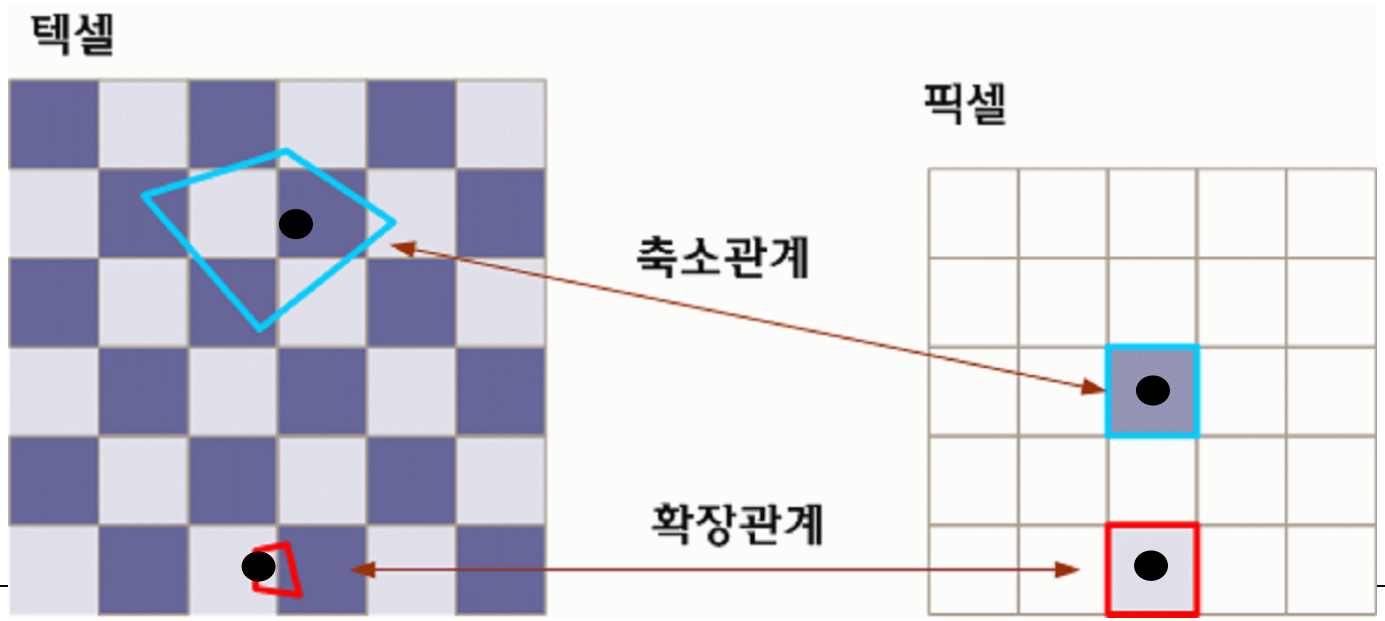


- Texture에서 polygon으로의 texture mapping이 일어날 때에는 확장 관계 (magnification)와 축소 관계 (minification)이 발생할 수 있다
- 확장 관계 (magnification): 텍셀 크기 이하가 한 화소로 mapping
- 축소 관계 (minification): 여러 texel이 한 화소로 mapping
- 두 경우 모두 aliasing 발생 가능

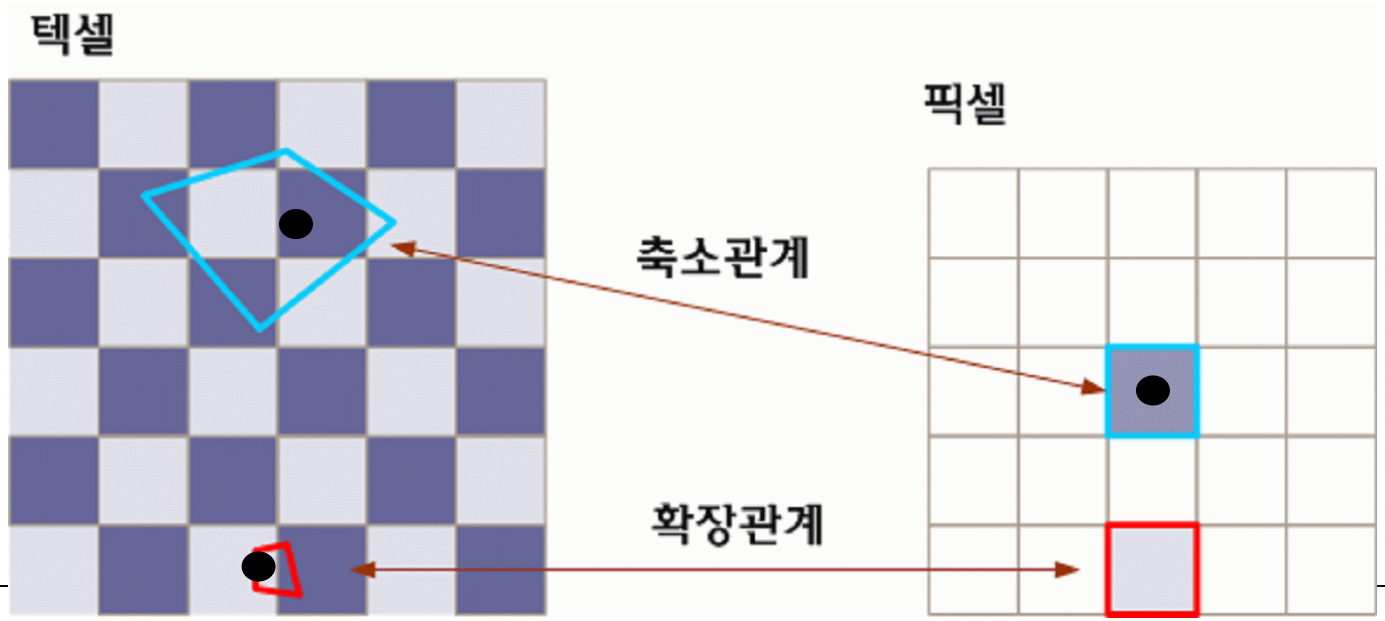


이러한 경우 한가지 방법은 point sampling과 유사하게 픽셀의 중앙점이 픽셀을 대표한다고 가정하고 픽셀의 중심 (P)가 텍셀의 어느 위치에 mapping 되는지 찾고 다음의 방법들이 사용 가능하다

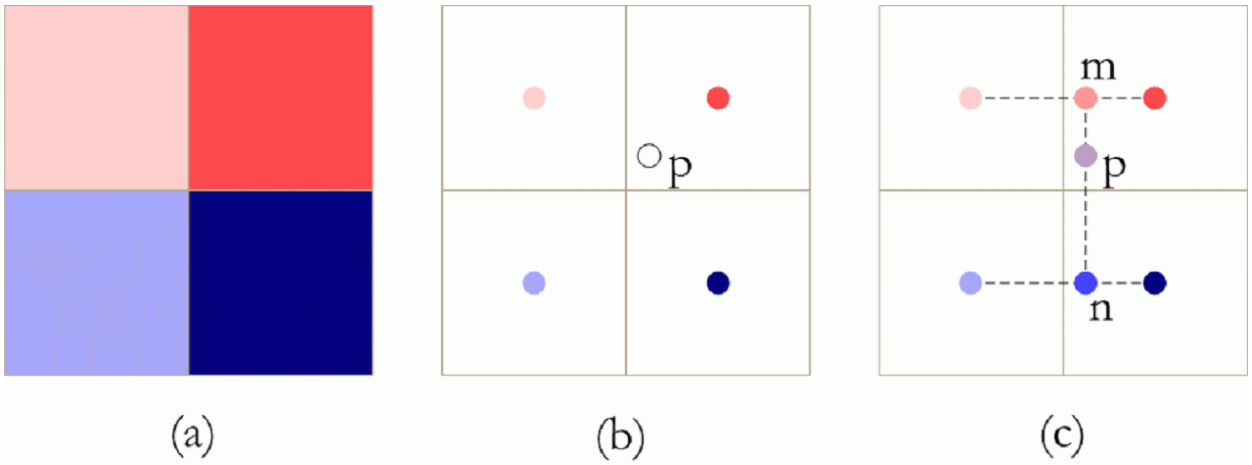
- 1) nearest neighbor filtering : 픽셀의 색을 이 texel의 색으로 결정
- glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
- glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);



- 픽셀의 중심 (P)가 텍셀의 어느 위치에 mapping 되는지 찾고
- 2) linear filtering : 이 위치에서 가장 가까운 텍셀 4곳 (텍셀 중심 기준)을 찾은 후 양방향 선형 보간
- glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
- glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);

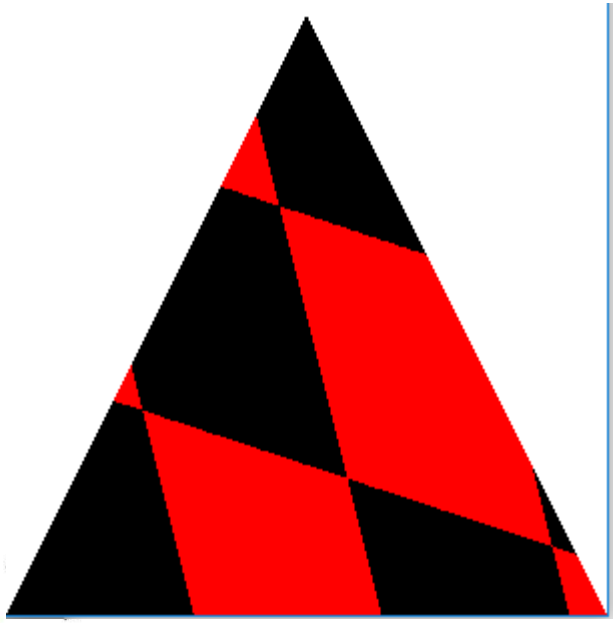


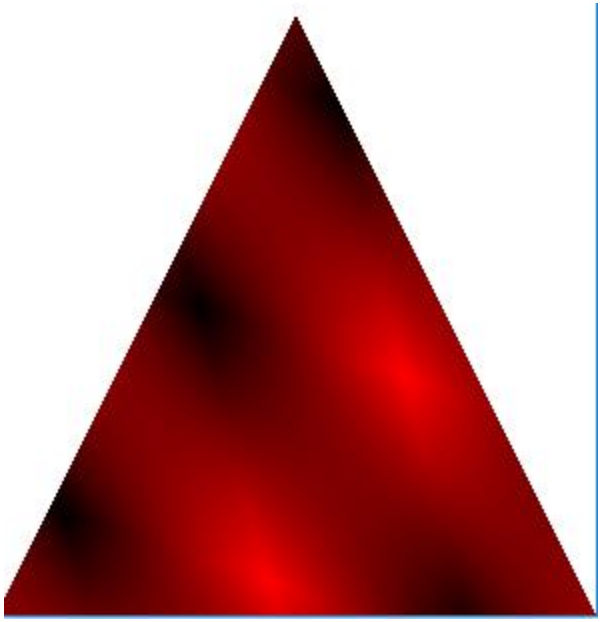
- 예: 만일 pixel의 중앙점이 texel의 점 p로 사상되었을 경우
- 1. Nearest neighbor filtering: 그 픽셀은 적색이 된다. 그림 (b)
- 2. Linear filtering: 점 p에서 가장 가까운 4개의 texel을 선택한 후에 양방향 선형 보간으로 p의 texture 색을 구한다



- 대부분의 경우 양방향 선형 보간을 사용하는 linear filtering 이 주위 texel들의 색을 사용하므로 nearest neighbor filtering 보다 aliasing 면에서 더 성능이 좋다
- 대부분의 그래픽 카드에서는 양방향 선형 보간을 사용하는 linear filtering을 표준으로 사용한다

- 확장 필터: GL_TEXTURE_MAG_FILTER
- 축소 필터: GL_TEXTURE_MIN_FILTER
- MAG_FILTER와 MIN_FILTER를 GL_LINEAR로 바꾸어 보았다
- glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
- glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);



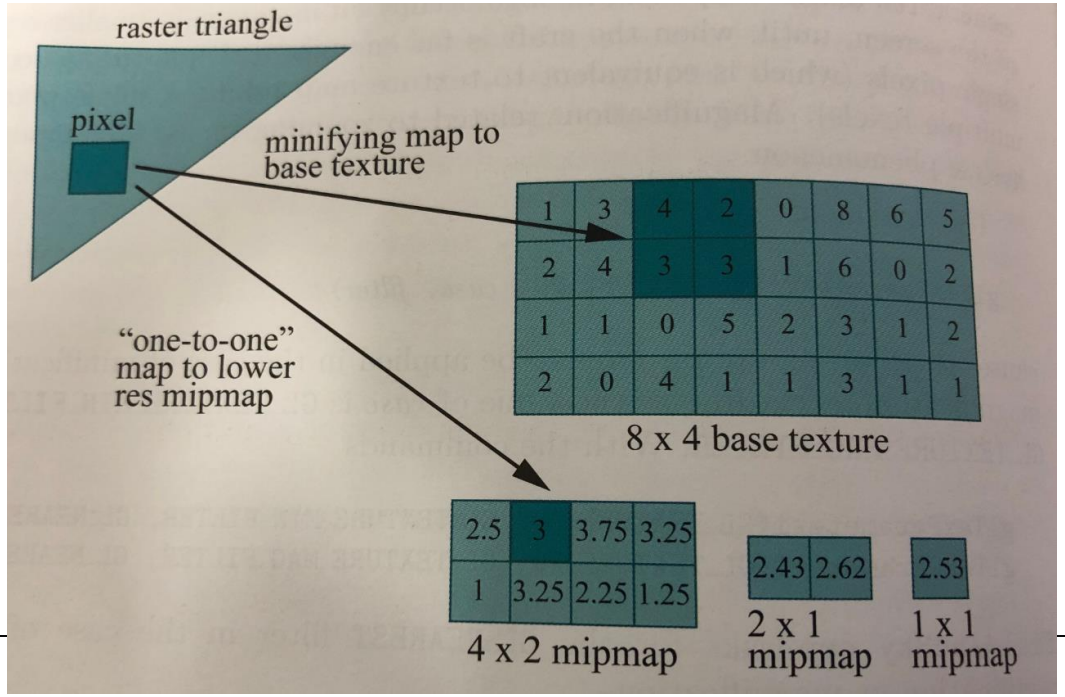


- **https://www.dropbox.com/s/ohkxgg8lygriz8f/texture_3.txt?dl=0**

- 밉 매핑 (MipMapping)
- Pre-filtered textures

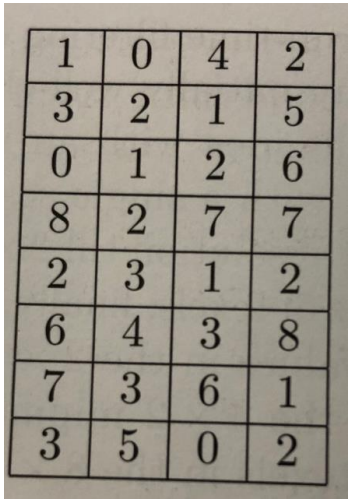
- In the case of minification, OpenGL offers an assortment of efficient filtering options based on pre-assigning a set of textures to be used at different levels of minification
- Starting with the original texture, the base texture, a set of textures of progressively lower resolution, called mipmaps, is prepared

- **The base (original) texture is of resolution 8 x 4 with a single scalar color value at each texel.**
- **Mipmaps of successively lower resolution till 2x1 are computed by averaging the color values in 2x2 squares of texles; finally 1x1 mipmap is computed by average the two color values in the 2x1 mipmap**



- If a base texture of resolution $2^m \times 2^n$ is to be minpmapped, then OpenGL requires mipmaps of resolution $2^{m-1} \times 2^{n-1}, 2^{m-2} \times 2^{n-2}, \ldots,$ obtained by halving both width and height, until one of the dimensions becomes 1
- If the other dimension is still greater than 1, then it must be repeatedly halved and mipmaps provided for each resolution down to 1×1

- **예: if the base texture is 4x8 with color values at the texels as follows. Then find all the mipmaps down to the one of lowest resoution**



1	0	4	2
3	2	1	5
0	1	2	6
8	2	7	7
2	3	1	2
6	4	3	8
7	3	6	1
3	5	0	2

1.5	3.0
2.75	5.5
3.75	3.5
4.5	2.25

3.1875
3.5

3.3437

- **Level-of-detail (LOD)**
- **Mipmapping is one of a class of LOD (level-of-detail) methods, which are important in graphics from the point of view of run-time efficiency**
- **Representing objects by polygonal meshes of varying levels of refinement is another practically important LOD application**

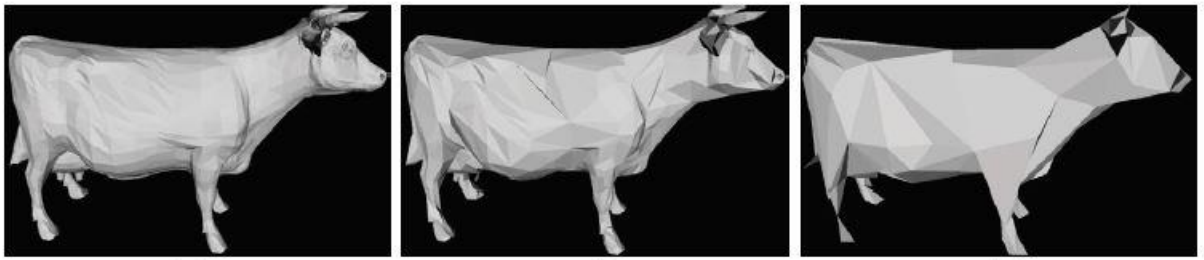


Figure 12.31: Cow at 3 different resolutions: (a) 5804 (b) 1772 (c) 328 triangles.