Computer Graphics

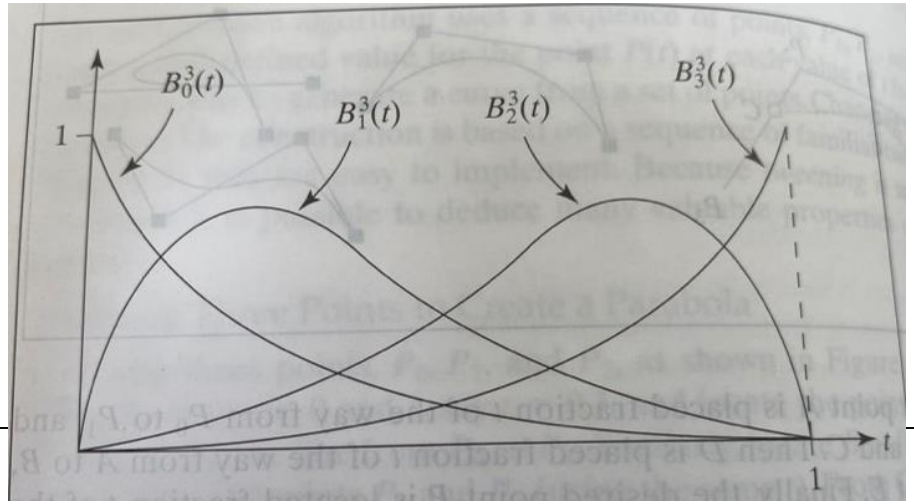
Prof. Jibum Kim

Department of Computer Science & Engineering

Incheon National University

- **Blending function and Piecewise polynomial**

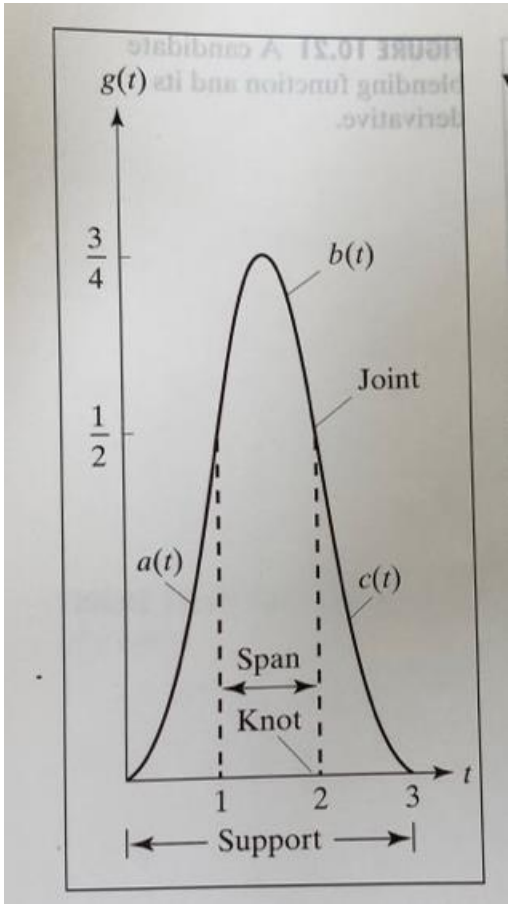
- Bernstein polynomial is “active” (0이 아님) over the entire interval [0, 1]. The interval over which a function is nonzero is often called its support
- 아래 예: Bernstein polynomial, n=3
- $B_0^3(t) = (1-t)^3, B_1^3(t) = 3(1-t)^2 t, B_2^3(t) = 3(1-t)^1 t^2, B_3^3(t) = t^3$
- B*ézier*Curve는 이 Bernstein polynomial이 조합되어 구성된다
- 아래 그림과 같이 t값이 변함에 따라서 모든 Bernstein polynomial에 영향을 주므로 결과적으로 any control point affects the shape of the curve everywhere with no local control



- 앞에서 본 Bernstein polynomial과 같이 여러 polynomial 함수가 조합되어 curve를 구성하는 함수를 blending function (혹은 basis function)이라 부른다
- Blending function의 조건
- 1. be easy to compute and numerically stable (?)
- 2. sum to one at every t in [a, b]
- 3. have small support to offer local control
- 4. be smooth enough (smooth 명확한 정의는 뒤에)
- 5. interpolate certain control points, chosen by the designer

- **Piecewise polynomial (조각 다항)**
- **앞에서 본 Bernstein polynomial의 문제점들을 인지하고 이를 해결하고자 한다. 어떤 문제점들?**
- **To attain more flexibility, we try piecing together several low-degree polynomials**
- **Such curves are defined by different polynomials in different t-intervals and are called piecewise polynomials**

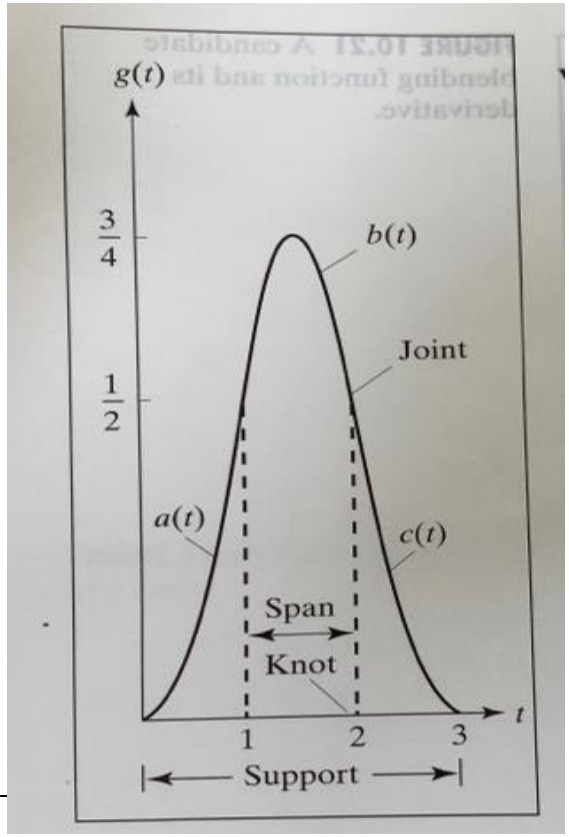
- Piecewise polynomial의 예
- $g(t)$ 는 3개의 polynomial segment로 구성됨
- $g(t)$ 의 support는 [0, 3], 각 구간을 span 이라 부름
- Segment가 만나는 점을 joints
- Knots are values of t where segments meet
- 각 polynomial segment는 저차 다항식임
- $a(t) = \frac{1}{2}t^2, t \in [0, 1]$
- $b(t) = \frac{3}{4} - \left(t - \frac{3}{2}\right)^2, t \in [1, 2]$
- $c(t) = \frac{1}{2}(3 - t)^2, t \in [2, 3]$



- **Spline function**

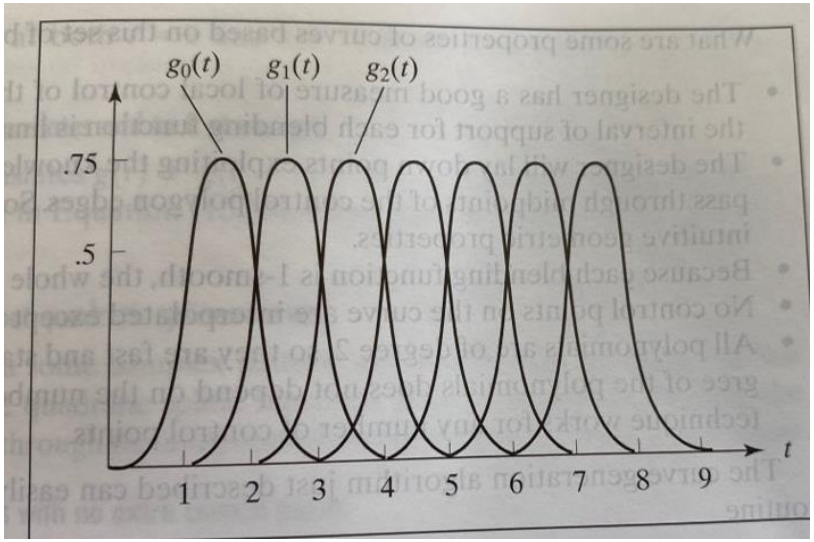
- Curve의 smoothness 정의
- A curve is 0-smooth in an interval if it is continuous (연속)
- A curve is 1-smooth in an interval if its first derivative (일차 도함수) exists and is continuous throughout the given interval
- A curve is 2-smooth if its first and second derivatives exist and are continuous throughout the given interval

- 정의: An **Mth-degree spline function** is a piecewise polynomial of degree M that is **(M-1)-smooth at each knot**
- 앞의 piecewise polynomial, $g(t)$,은 1-smooth at each knot?
- 확인해보자
- 이 $g(t)$ 는 quadratic (2차)-spline 함수이다
- $a(t) = \frac{1}{2}t^2, , a'(t) = t, \quad t \in [0, 1]$
- $b(t) = \frac{3}{4} - \left(t - \frac{3}{2}\right)^2, b'(t) = -2t + 3, t \in [1, 2]$
- $c(t) = \frac{1}{2}(3 - t)^2, \mathbf{c'(t)} = \mathbf{t - 3}, t \in [2, 3]$



- 그렇다면 원래 문제로 돌아와서 이러한 $g(t)$ 와 같은 spline 함수를 사용하여 어떻게 blending 함수를 만들 수 있을까?
- 가장 단순하면서 흔히 사용되는 방법
- Use translated version of $g(t)$
- 즉, $g_k(t) = g(t-k)$, for k=0, 1, …
- It is crucial that we translate each function by an integer, to make the shapes line up properly so they sum to 1

- $g_k(t) = g(t-k)$, for k=0, 1, …6

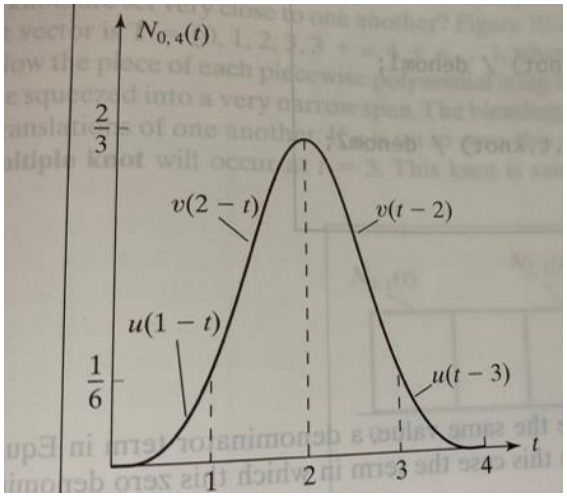


- Blending function: $\sum_{k=0}^{6} g(t-k) = 1$ for t in [2, 7]
- 1. only values of t between 2 and 7 can be used
- 2. 각 구간 (t가 2에서 7사이)에서 3개의 함수만 active
- 3. t=2, 3, ..7에서 only two of the functions are active and they both have value of 0.5

- 이러한 spline함수를 사용했을 때의 장점
- 1. Local control 가능: the interval of support for each blending function is limited to length 3 (앞의 그림 확인)
- 2. Each blending function is 1-smooth, the whole curve is 1-smooth
- 3. All polynomials are of degree 2 (low-degree)

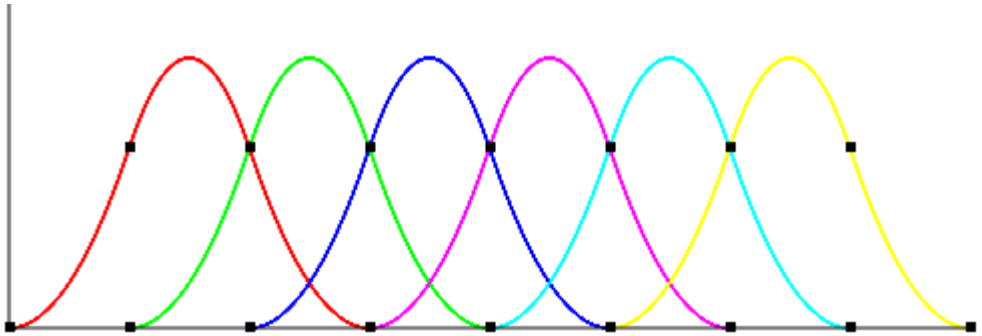
- 앞에서 본 spline 함수 ($g(t)$)는 B-spline 함수의 한 예이다. B (basis), 이를 quadratic (2차) B-spline 함수라 한다
- B-spline 함수는 blending function이 smallest support를 갖으면서 greatest local control을 가능하게 한다
- B-spline 함수 중에 가장 널리 쓰이는 형태는 cubic B-spline 함수이다

- **Cubic B-spline 함수의 예**



- $g(t) = \begin{cases} u(1-t), 0 \le t \le 1 \\ v(2-t), 1 \le t \le 2 \\ v(t-2), 2 \le t \le 3 \\ u(t-3), 3 \le t \le 4 \\ 0, otherwise \end{cases}$
- 여기서 $u(t) = \frac{1}{6}(1-t)^3, v(t) = \frac{1}{6}(3t^3 - 6t^2 + 4)$
- t=2를 기준으로 대칭함수

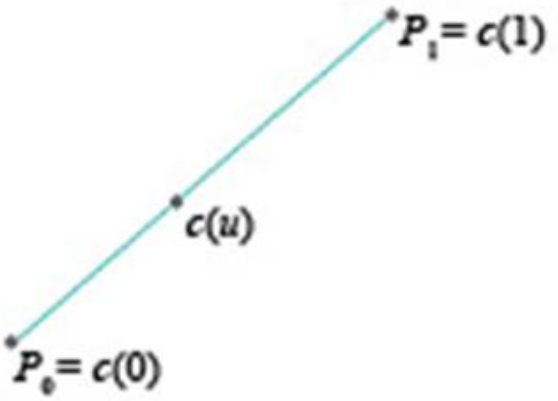
- 이를 앞에서와 같이 t 축에서 정수만큼 translate 시켜서 사용함



- **Bézier** Surface
- Bilinear **Bézier** Patch

- 지금까지 **Bézier Curve를 만드는 방법을 간단하게 배웠다**
- 이를 확장하여 곡선이 아닌 직선 기반의 표면, Surface, 를 만드는 방법인 Bilinear (양방향) Bezier Patch 에 대해서 알아보자
- **Bézier Curve 는** 파라미터 u하나로 곡선을 만들었지만 평면 (혹은 곡면)을 만들기 위해서는 파라미터 v를 하나 더 사용한다

- Linear $\mathbf{B\acute{e}zier}$ curve
- 두 개의 control point, P0, P1만 있다고 하자
- P0와 P1을 연결하는 $\mathbf{B\acute{e}zier}$ curve (선분)는 다음과 같이 표현 가능
- $c(u) = (1-u)P_0 + uP_1$, 단, $0 \le u \le 1$



- **Bilinear Bézier Patch**
- **Linear Bézier curve 를 확장 한 것**
- **4개의 control points (a, b, c, d)가 주어져 있고 두 개의 파라미터 u, v 사용**

1. 선분 a 와 b 사이의 한 점: $e(u) = (1-u)a + ub,\ 0 \le u \le 1$

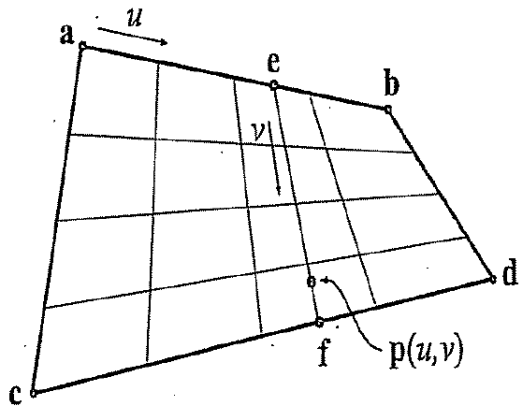
2. 선분 c 와 d 사이의 한 점: $f(u) = (1-u)c + ud,\ 0 \le u \le 1$

3. 선분 e 와 f 사이의 한 점

$p(u,v) = (1-v)e + vf, 0 \le v \le 1$

$p(u,v) = (1-u)(1-v)a + u(1-v)b + v(1-u)c + uvd$

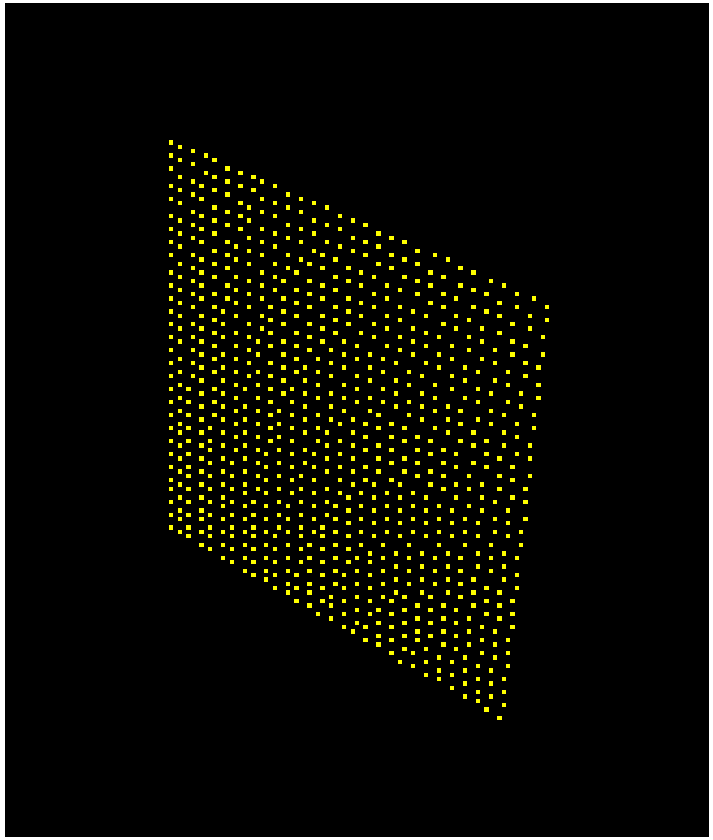
$,\ 0 \le u \le 1, 0 \le v \le 1$



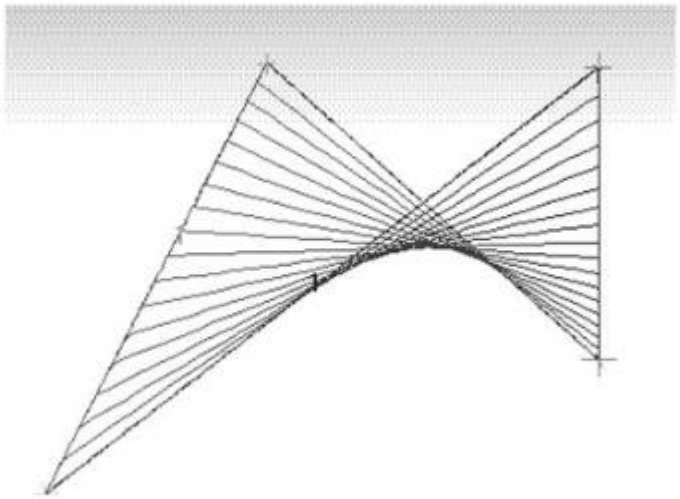
- Bilinear *Bézier* Patch 를 이용하면 surface를 만들 수 있다
- a=[0, 1, 0], b=[1, 1, 0], c=[0, 0, 0], d=[1, 0, 0]
- 식을 구해 보자

- Bilinear *Bézier* Patch 를 이용하면 surface를 만들 수 있다
- a=[0, 1, 0], b=[1, 1, 0], c=[0, 0, 0], d=[1, 0, 0]
- 식을 구해 보자

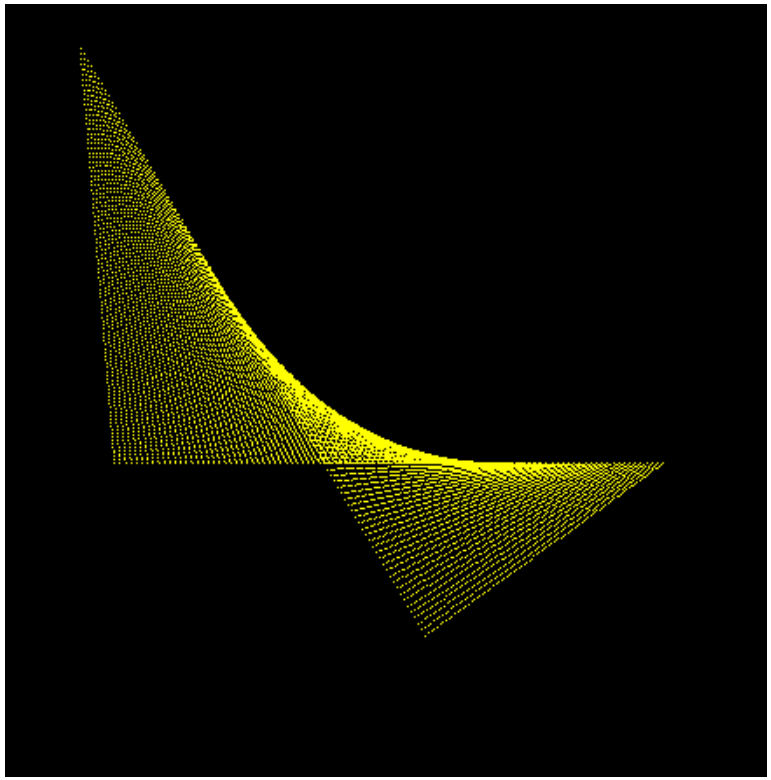
```
#include <GL/glut.h>          //  we will use GLUT (GL UTILITY TOOLKIT)
void Display(){
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 1.0, 0.0);
  glLoadIdentity();
gluLookAt (3.0,3.0, 3.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

 int i;
 int j;
 int NUM=30;
 double u;
 double v;
glBegin(GL_POINTS);
 for (i=0; i< NUM; i++)
 {

                    for (j=0; j< NUM; j++)
                    {
                    u=double(i)/(NUM*1.0);
   v=double(j)/(NUM*1.0);
                    glVertex3f( u, 1-v, 0);
                    }
 }
  glEnd();

  glFlush();
 }

void resize(int w, int h)
{
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(60.0, (float)w/(float)h, 1.0, 20.0);
  glMatrixMode(GL_MODELVIEW);
}


int main(){
  glutInitWindowSize(600,600);
  glutInitWindowPosition(300,300);
  glutCreateWindow("OpenGL Hello World!");
  glutDisplayFunc(Display);
  glClearColor(0.0, 0.0, 0.0, 0.0);
   glutReshapeFunc(resize);
  glutMainLoop();
  return 0;
}
```



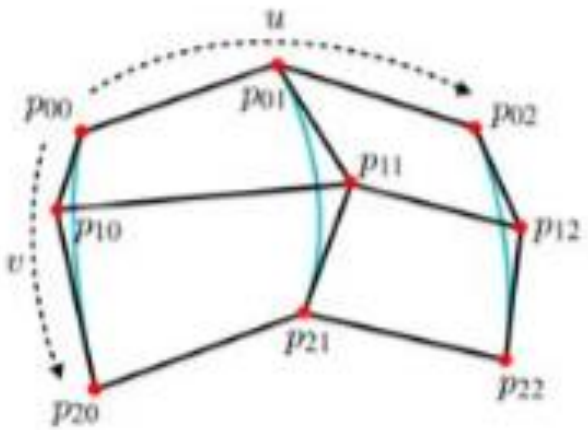
- Control points 위치에 따라서 다양한 surface를 만들 수도 있다
- 4개의 control point를 이용하여 Bilinear Bézier Patch 만듬
- a=[-1, 0, 0,], b=[1, 0, -2], c=[-1, 2, 0], d=[1, 0, 0]



```
#include <GL/glut.h>          //  we will use GLUT (GL UTILITY TOOLKIT)
void Display(){
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0, 1.0, 0.0);
  glLoadIdentity();
gluLookAt (3.0,3.0, 3.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0);

 int i;
 int j;
 int NUM=100;
 double u;
 double v;
glBegin(GL_POINTS);
 for (i=0; i< NUM; i++)
 {

                    for (j=0; j< NUM; j++)
                    {
                    u=double(i)/(NUM*1.0);
   v=double(j)/(NUM*1.0);
                    glVertex3f( (1-u)*(1-v)*-1 + u*(1-v)*1 + v*(1-u)*-1 + u*v*1,  (1-u)*(1-v)*0 + u*(1-v)*0 + v*(1-u)*2 + u*v*0, -2*u*(1-v));
                    }
 }
  glEnd();

  glFlush();
 }

void resize(int w, int h)
{
  glViewport(0, 0, w, h);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluPerspective(60.0, (float)w/(float)h, 1.0, 20.0);
  glMatrixMode(GL_MODELVIEW);
}


int main(){
  glutInitWindowSize(400,400);
  glutInitWindowPosition(300,300);
  glutCreateWindow("OpenGL Hello World!");
  glutDisplayFunc(Display);
  glClearColor(0.0, 0.0, 0.0, 0.0);
   glutReshapeFunc(resize);
  glutMainLoop();
  return 0;
}
```

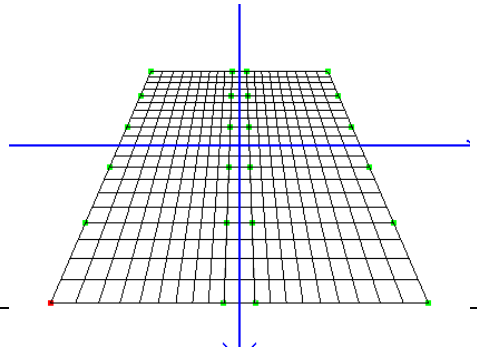


- **Biquadratic *Bézier* Patch (surface)**
- **비슷한 방식으로 아래 그림과 같이 두 개의 파라미터 u, v를 사용하여 2차 *Bézier* Patch 를 만들 수 있다**
- **직접 유도해 보자**
- **비슷한 방식으로 Bicubic *Bézier* Patch 도 확장 가능하다**



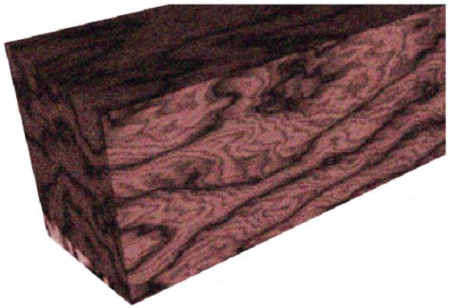
$$p(u,v) = (1-u)^2(1-v)^2 p_{00} + 2u(1-u)(1-v)^2 p_{01} + u^2(1-v)^2 p_{02} + 2(1-u)^2 v(1-v) p_{10} + 4uv(1-u)(1-v) p_{11} + 2u^2 v(1-v) p_{12} + (1-u)^2 v^2 p_{20} + 2u(1-u)v^2 p_{21} + u^2 v^2 p_{22}$$

- **Bézier surface OpenGL 코드 예**
- **https://www.dropbox.com/s/4e4cz5xfx3zzu36/bezier_surface.txt?dl=0**
- **space 키와 tab키로 control point 선택**
- **오른쪽/왼쪽 키로 control point x축에서 위아래 이동**
- **위/아래 키로 control point y축에서 위아래 이동**
- **page up/down 키로 control point z축에서 위아래 이동**
- **x,y,z키로 회전**

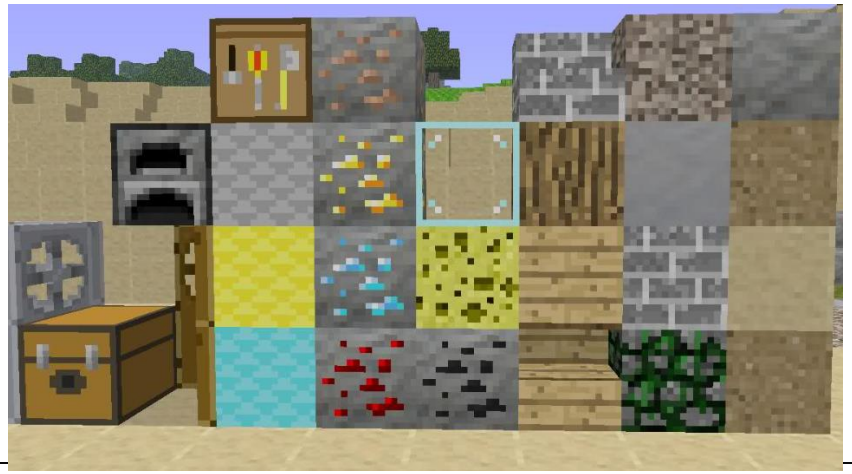


- **Texture (Texture mapping)**

- 아래 그림과 같 목재 표면을 모델링하는 데에는 무수히 많은 다각형이 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 사용**하지만 1, 2, 3차원 영상 모두를 texture로 사용할 수 있다



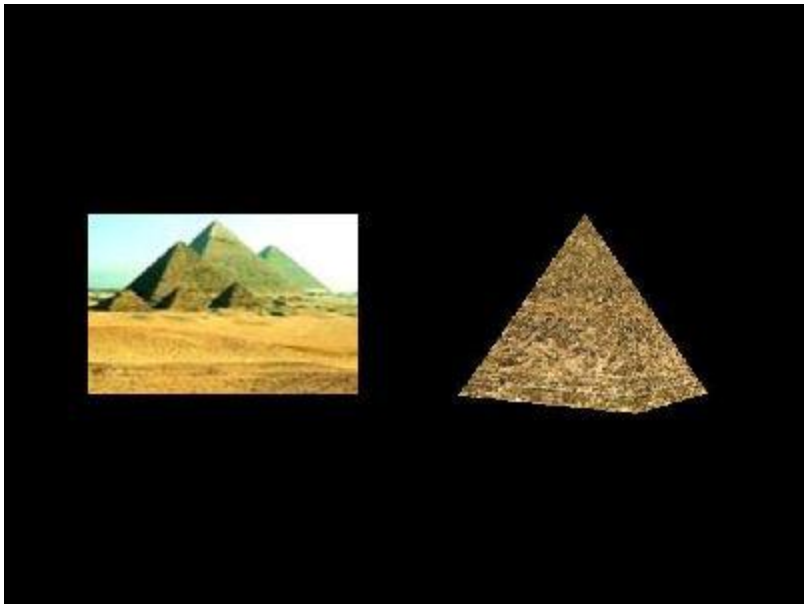
- Texture (texture mapping)는 간단히 말하면 앞에서 설명한 것과 같이 polygon의 표면에 image를 입히는 기법
- 주로, 2D bitmap image와 같은 그림 파일 이미지를 polygon에 적용한다
- OpenGL에서 rasterization 단계에서 적용



- 예)

2D images (textures)

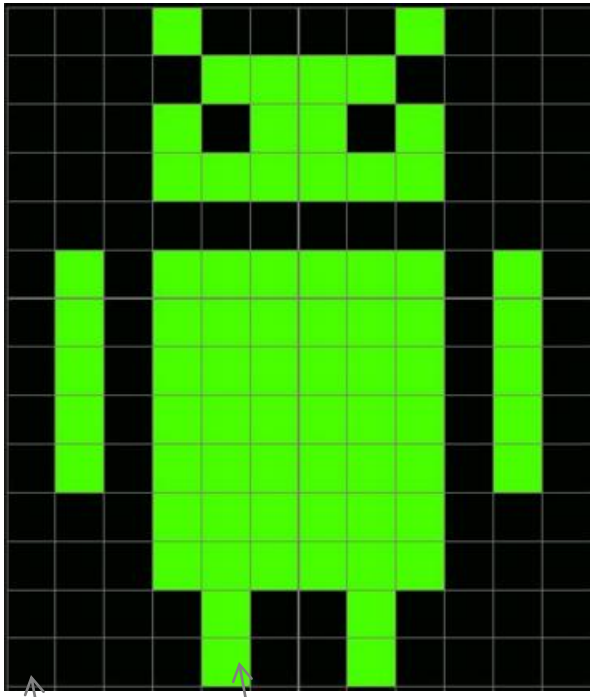
Example of texture mapping



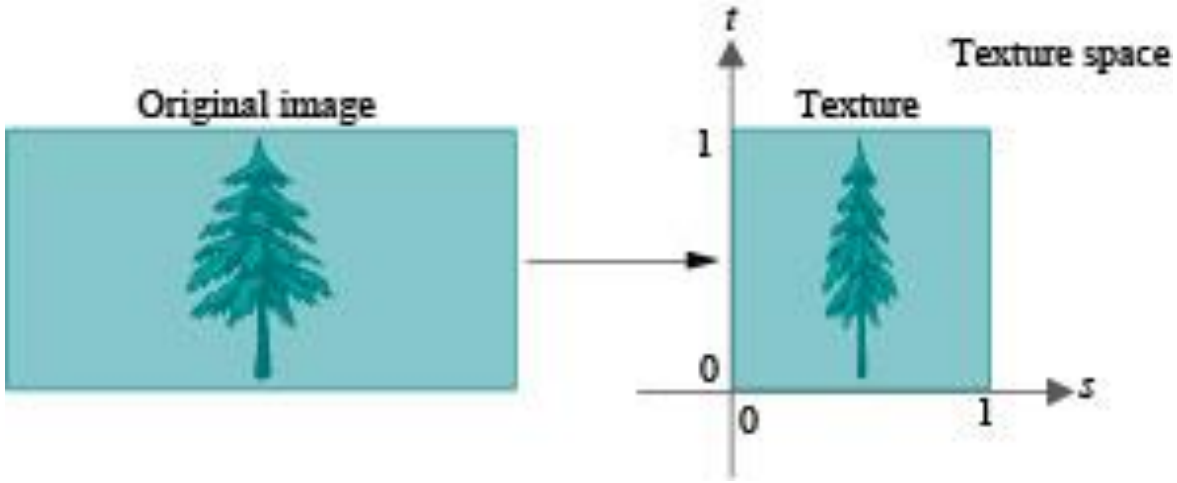
polygons

- **Texture basics**

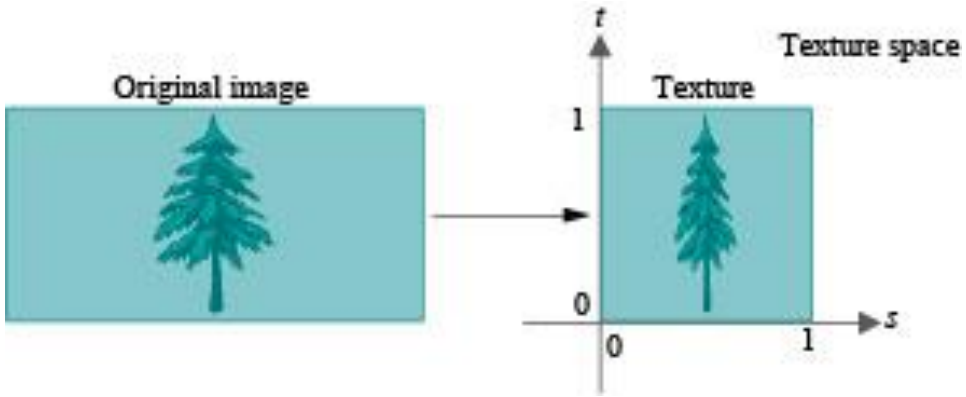
- 수업시간에 다루는 Texture는
- 주로 bitmap image 사용
- Bitmap image 파일 (.bmp, .jpg, .png..)
- 열어 보면 2D array 로 구성됨
- 기본 단위 (사각형 하나) : Pixel
- E.g., 12x14 size의 2D array
- Total 168개의 Pixel 있음
- 각 Pixel은 color값 저장 (0~255까지)
- (R, G, B), or (R, G, B, A)



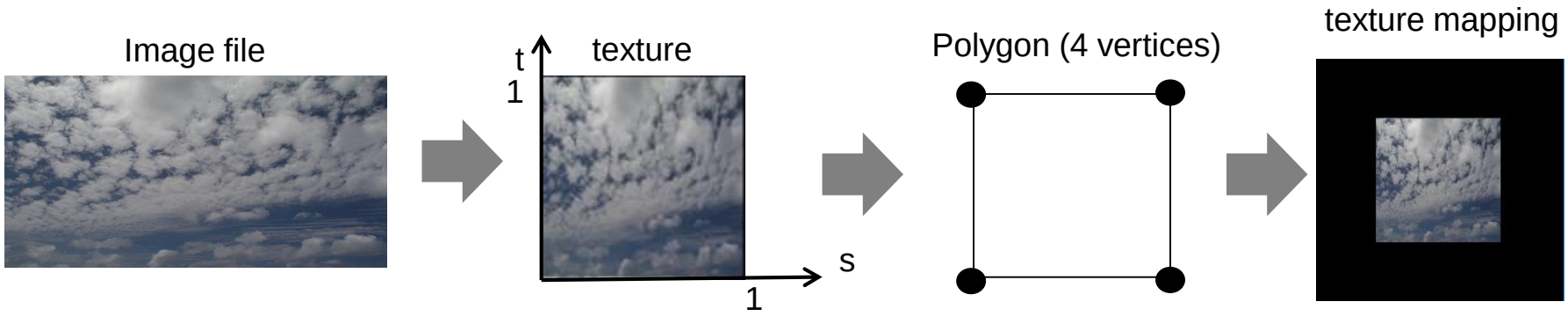
- Texture란? bitmap image file과 같은 2D array를 가상의 texture space로 옮긴 것을 의미 한다
- 이 가상의 texture space란 아래 그림과 같이 s축 t축으로 이루어짐
- 기본적으로 어떤 bitmap image file을 texture로 만들면 아래 그림과 같이 s ~ [0, 1] , t ~ [0, 1] 의 공간 (정사각형 공간)을 차지 한다



- 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
- Texture 좌표
- 좌하단: (0,0), 우하단: (1,0), 좌상단: (0,1), 우상단: (1,1)



- **Texture mapping: (s, t)로 표현된 texture를 polygon으로 mapping (사상) 시키는 것**
- **아래 예: 2D texture mapping**



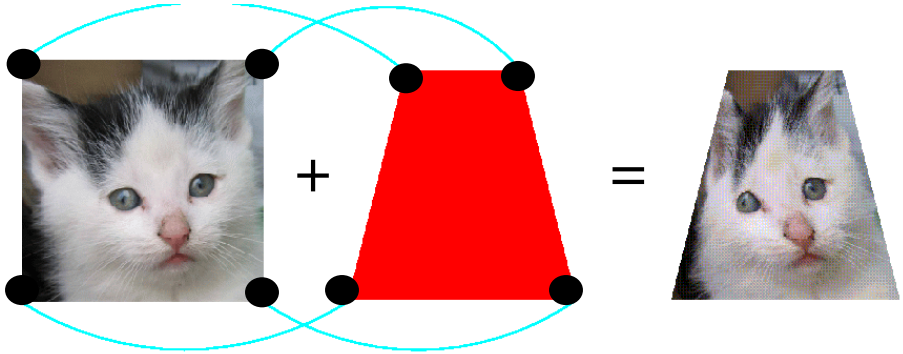
- Texture mapping은 자동과 수동 모두 가능하다
- 수동 texture mapping
- Texture의 각 vertex와 polygon의 각 vertex를 대응 (mapping) 시킨다

Texture: (s, t)=(0.0, 0.0) => Polygon의 왼쪽 아래

(s, t)=(0.0, 1.0) => Polygon의 왼쪽 위

(s, t)=(1.0, 1.0) => Polygon의 오른쪽 위

(s, t)=(1.0, 0.0) => Polygon의 오른쪽 아래



Texture Object Mapped Texture

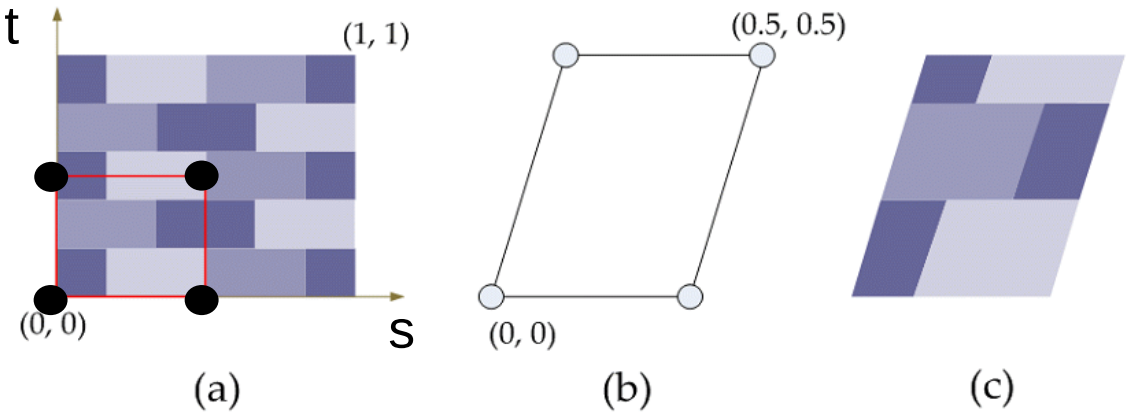
- Texture 전부가 아닌 Texture의 일부만 Polygon에 mapping 시킬 수도 있다
- (a) Texture (b) Polygon (c) Result of texture mapping

Texture: (s, t)=(0.0, 0.0) => Polygon의 왼쪽 아래

(s, t)=(0.0, 0.5) => Polygon의 왼쪽 위

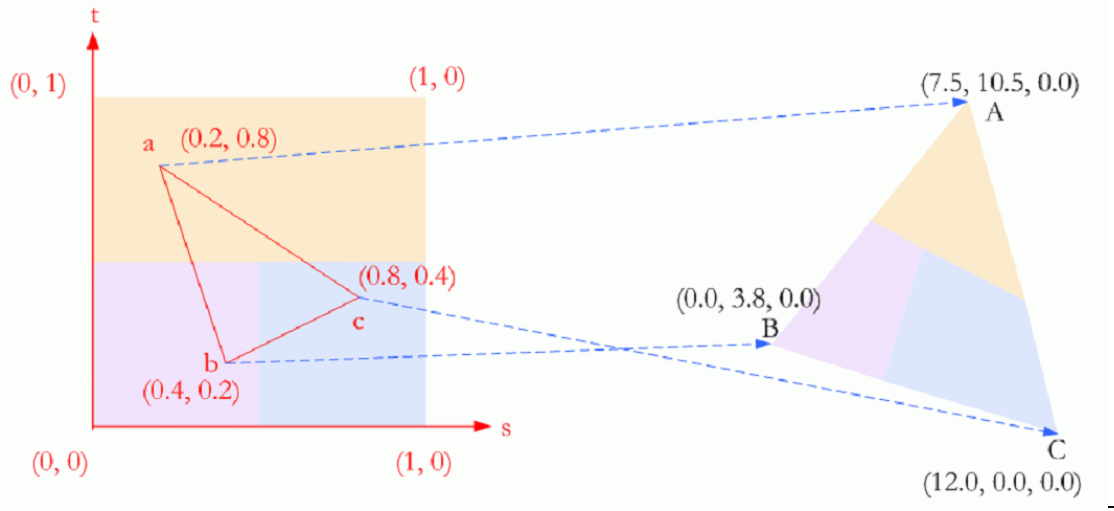
(s, t)=(0.5, 0.5) => Polygon의 오른쪽 위

(s, t)=(0.5, 0.0) => Polygon 의 오른쪽 아래



- **OpenGL에서의 수동 texture mapping 예**

```
glBegin(GL_POLYGON);
    glTexCoord2f(0.2, 0.8);          텍스쳐 정점 a를
    glVertex3f(7.5, 10.5, 0.0);      물체 정점 A에 할당
    glTexCoord2f(0.4, 0.2);          텍스쳐 정점 b를
    glVertex3f(0.0, 3.8, 0.0);       물체 정점 B에 할당
    glTexCoord2f(0.8, 0.4);          텍스쳐 정점 c를
    glVertex3f(12.0, 0.0, 0.0);      물체 정점 C에 할당
glEnd( );
```



- **OpenGL에서의 Texture 사용**

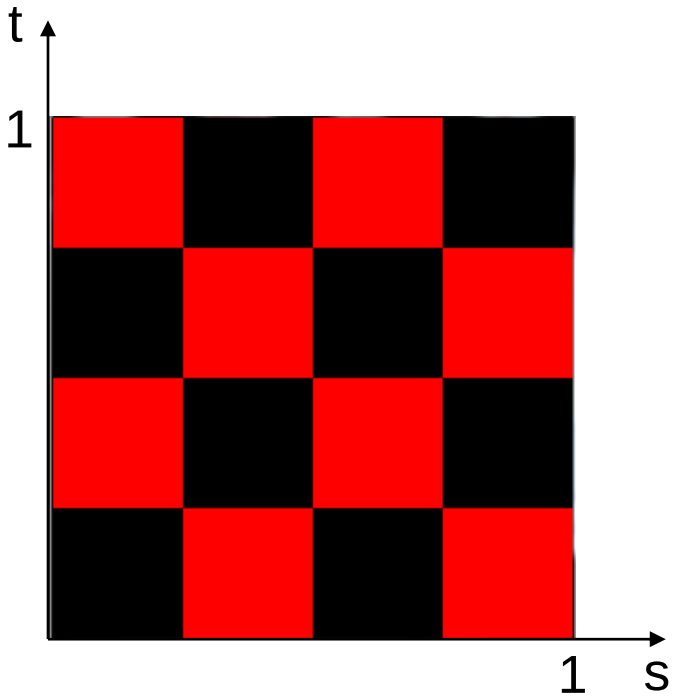
- **Enable Texturing (Texture 기능 활성화, 비활성화)**

```
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);**

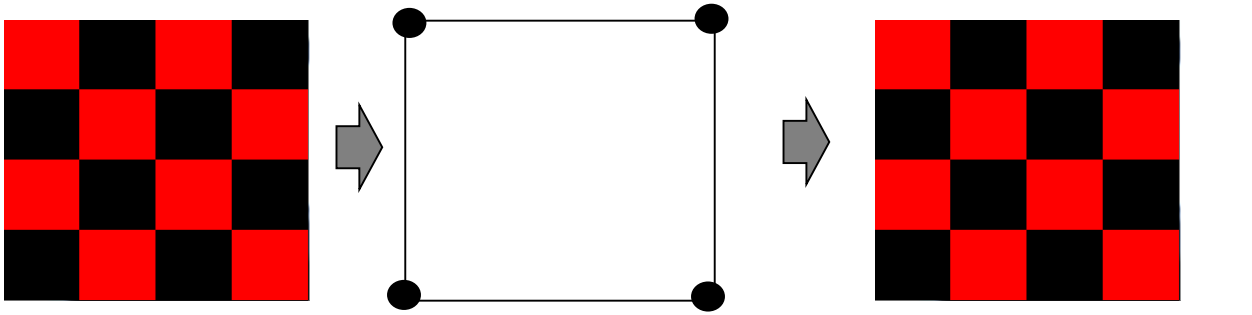
- Texture는 2D array로 직접 만들 수도 있다
- Texture의 기본 단위는 Texel 이라 한다. Texel의 R, G, B,는 0-255 사이
- 0: darkest, 255:brightest
- checkboard texture, WIDTH=4, HEIGHT=4
- Total 16 texels

```
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();
```



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

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

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

// WIDTH: texture의 width, HEIGHT: texture의 height

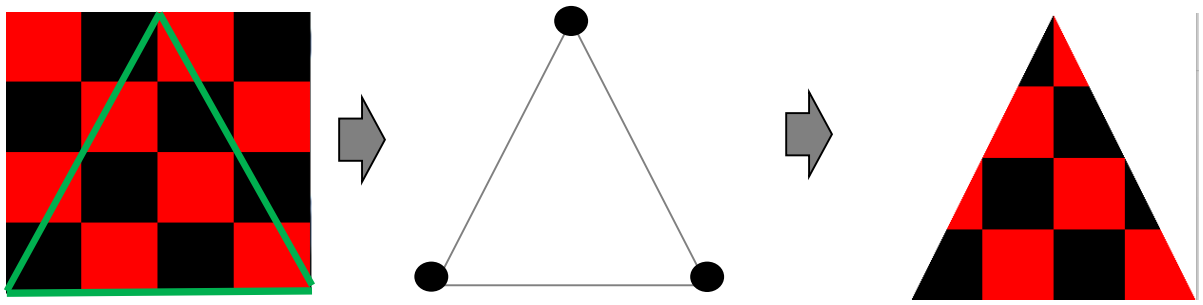
// 마지막 인자: 실제 texture image가 저장된 배열명

// 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)

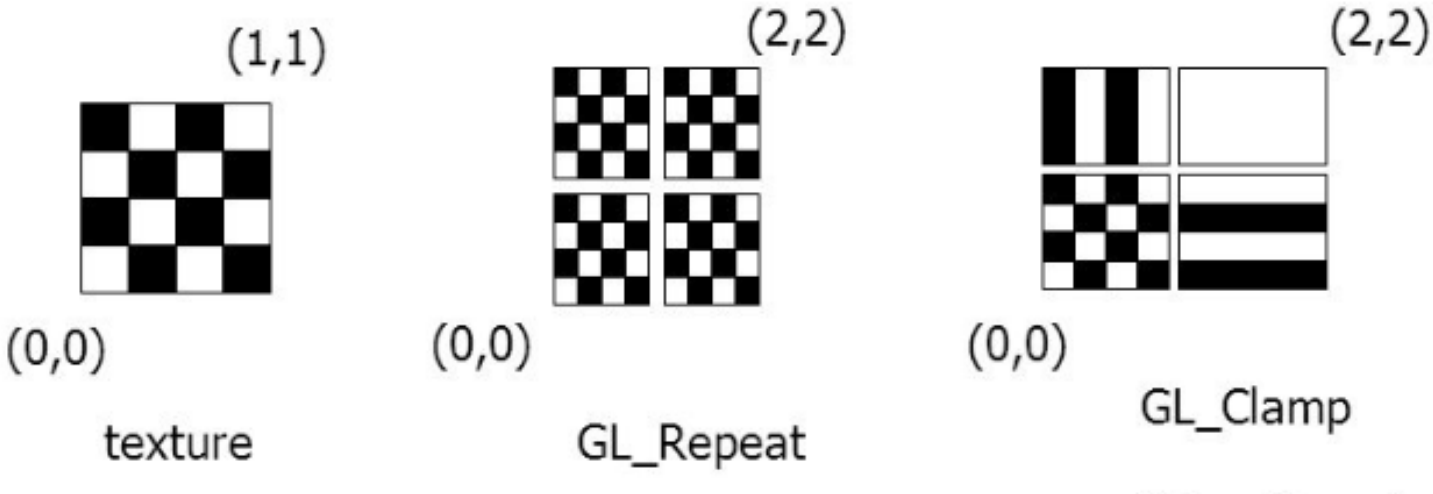
```
glBegin(GL_QUADS);
?
glEnd();
```



Texture Polygon Texture mapping

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

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

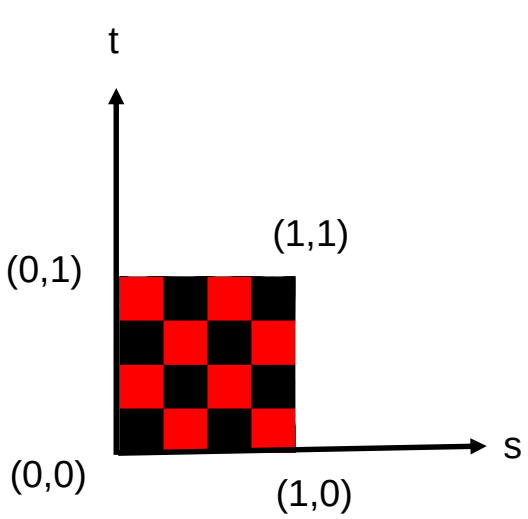


- 지금까지는 texture coordinates을 사용할 때에 s와 t 축 모두에서 [0, 1]사이의 값만을 사용하였다
- 만일 그 범위를 넘어서는 값을 주면 어떻게 될까?

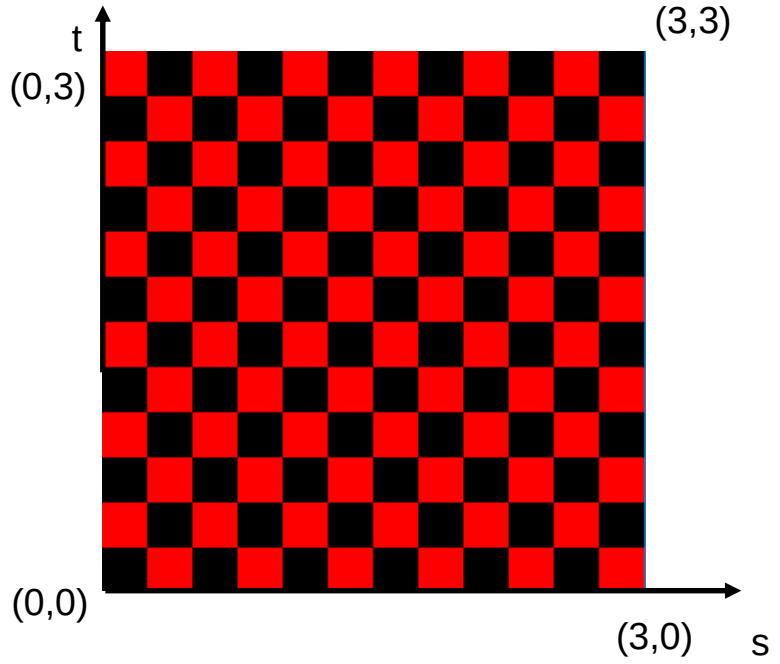
```
glBegin(GL_QUADS);
        glTexCoord2f(0.0, 0.0); glVertex3f(-1.0, -1.0, 0.0);
        glTexCoord2f(0.0, 3.0); glVertex3f(-1.0, 1.0, 0.0);
        glTexCoord2f(3.0, 3.0); glVertex3f(1.0, 1.0, 0.0);
        glTexCoord2f(3.0, 0.0); glVertex3f(1.0, -1.0, 0.0);
    glEnd();
glTexParameterf(GL_TEXTURE_2D,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

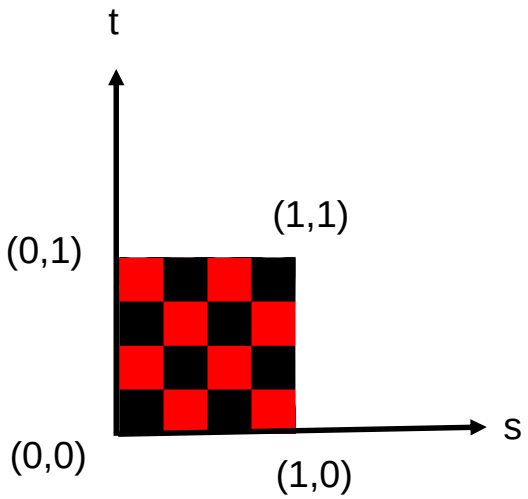
- **GL_REPERAT: 원래 S와 T축에서 [0, 1], 사이에 정의된 것을 S와 T축에서 반복하여 확장**



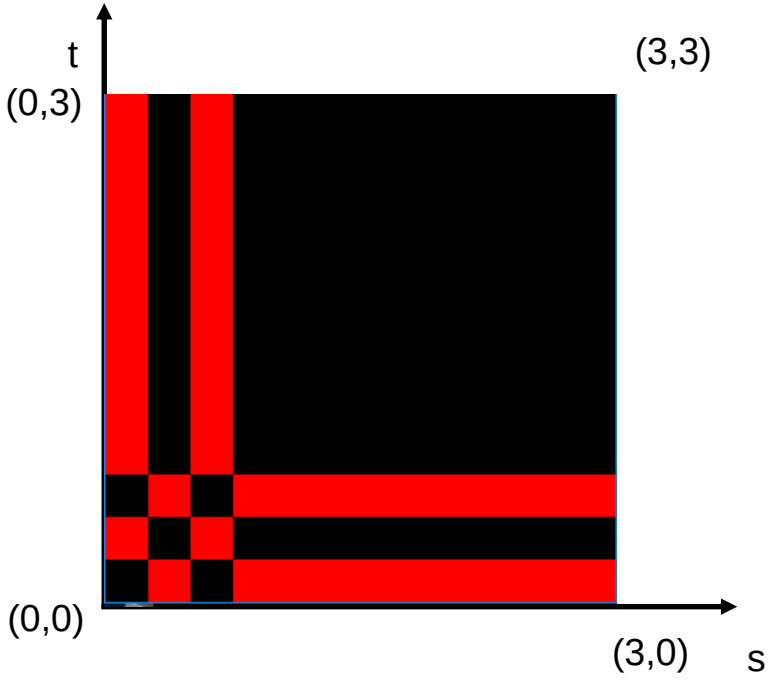
원래 Texture



- **GL_REPERAT: 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);
```