# Computer Graphics

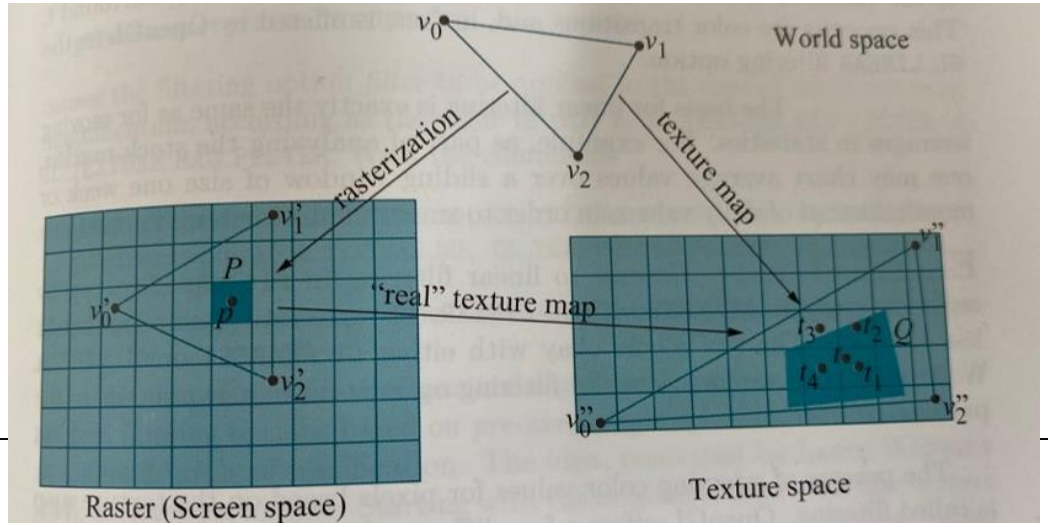
**Prof. Jibum Kim**

**Department of Computer Science & Engineering**

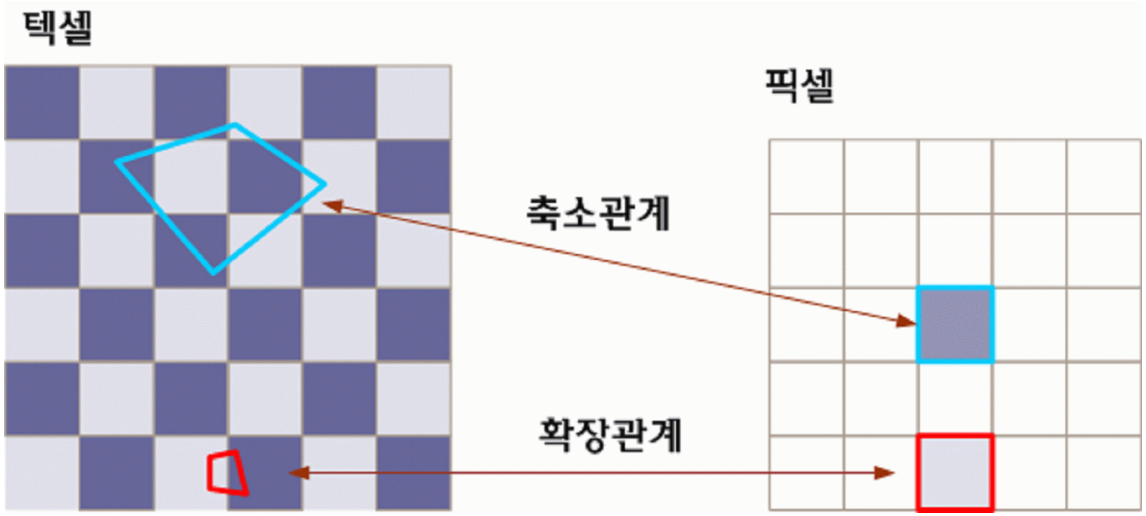
**Incheon National University**

- **Filtering**

- 다음과 같은 예를 살펴보자
- 세계 좌표로 정의된 삼각형 v0v1v2이 rasterization을 통하여 v0'v1'v2'로 mapping 되었다
- 또한 이 삼각형 v0v1v2는 앞에서 배운 것과 같이 texture mapping을 통하여 v0→ v0'', v1 → v1'', v2 → v2'' texture space로 mapping 되었다고 하자
- 여기서 pixel P가 texture space의 Q에 대응한다고 하면 pixel P 하나가 Q의 여러 texel에 대응하므로. P의 색을 Q의 어떤 위치의 texel의 색으로 정해야 하는지에 대한 문제가 생긴다. 이를 Texture에서의 filtering 문제라 한다

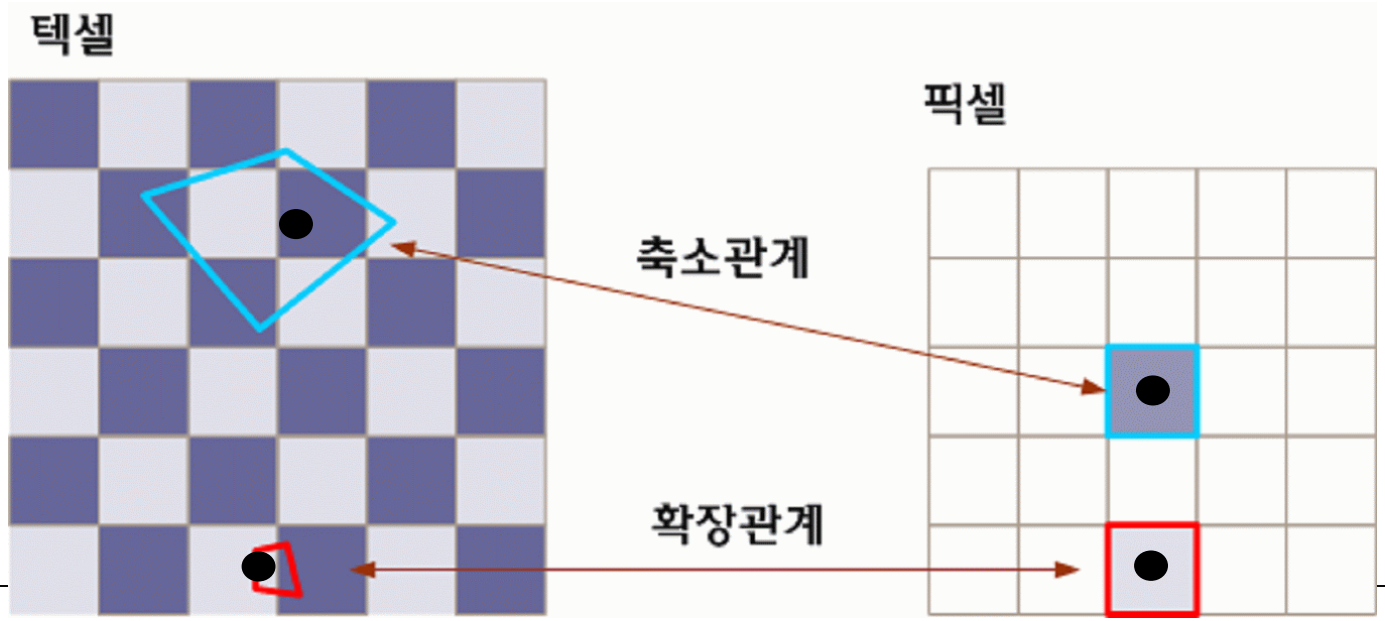


- Texture에서 polygon으로의 texture mapping이 일어날 때에는 확장 관계 (magnification)와 축소 관계 (minification)이 발생할 수 있다
- 확장 관계 (magnification): 텍셀 크기 이하가 한 화소로 mapping
- 축소 관계 (minification): 여러 texel이 한 화소로 mapping
- 두 경우 모두 aliasing 발생 가능

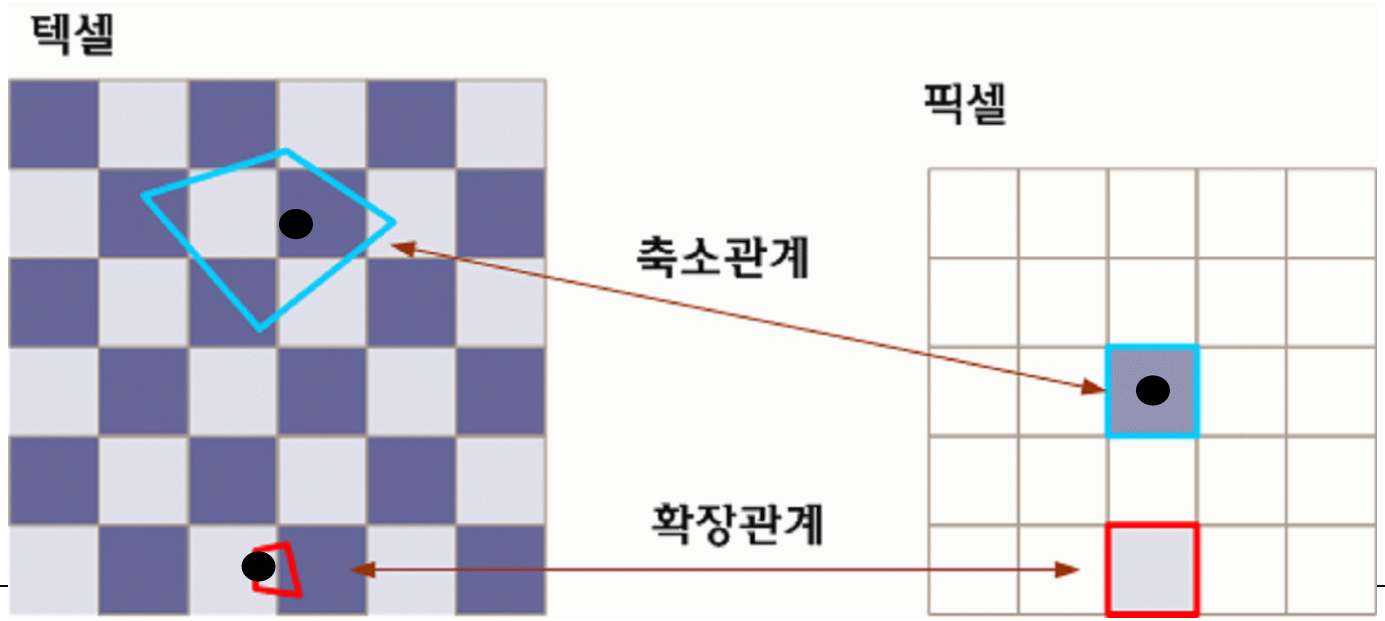


이러한 경우 한가지 방법은 point sampling과 유사하게 픽셀의 중앙점이 픽셀을 대표한다고 가정하고 픽셀의 중심 (P)가 텍셀의 어느 위치에 mapping 되는지 찾고 다음의 방법들이 사용 가능하다

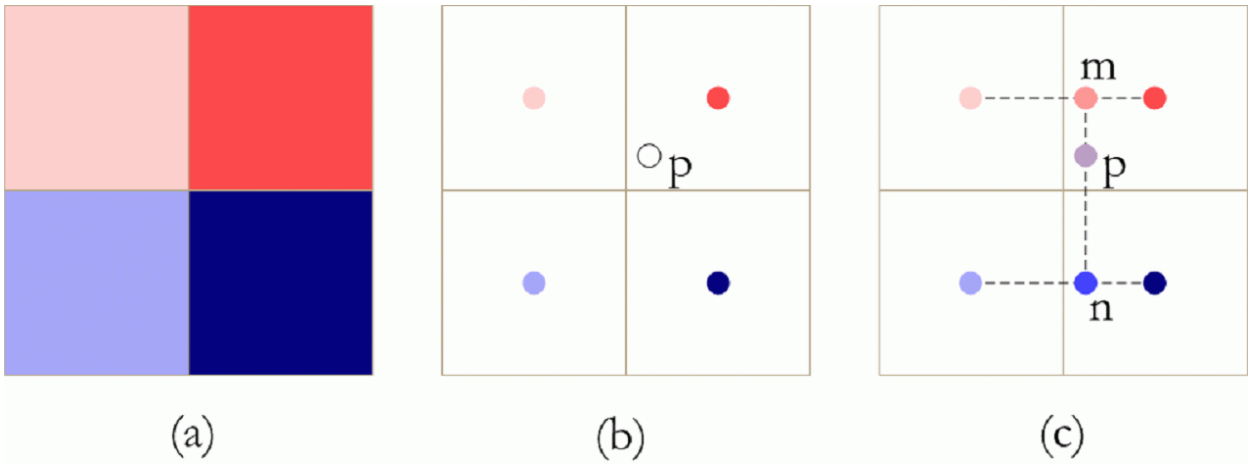
- 1) nearest neighbor filtering : 픽셀의 색을 이 texel의 색으로 결정
- glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
- glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);



- 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);

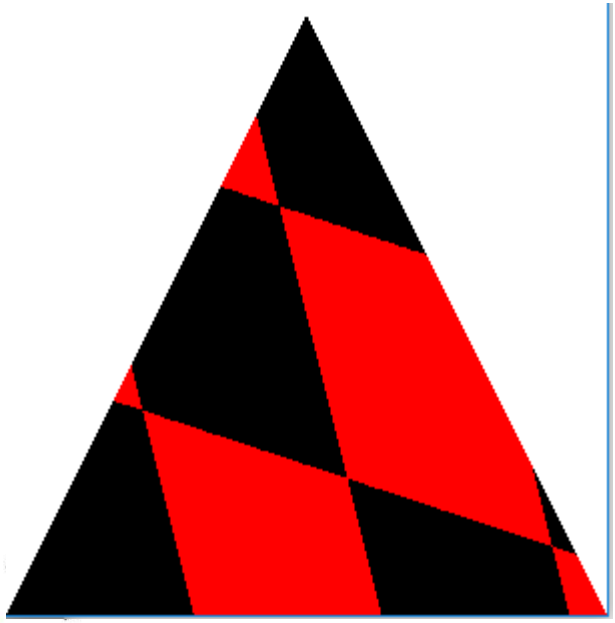


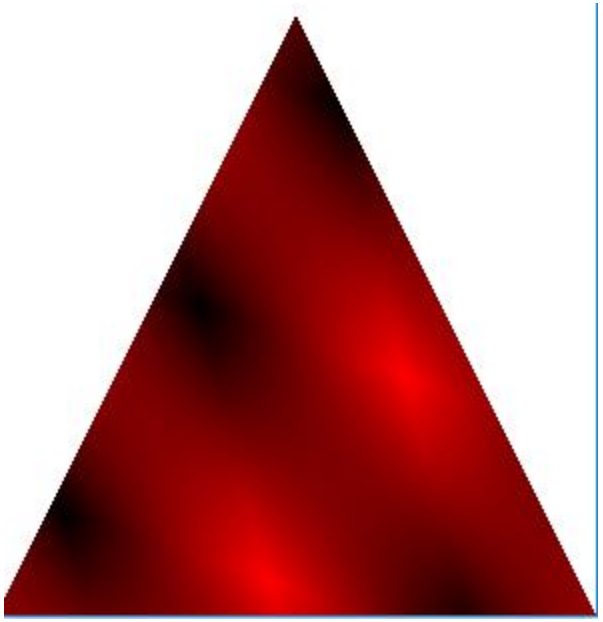
- 예: 만일 pixel의 중앙점이 texel의 점 p로 사상되었을 경우
- 1. Nearest neighbor filtering: 그 픽셀은 적색이 된다. 그림 (b)
- 2. Linear filtering: 점 p에서 가장 가까운 4개의 texel을 선택한 후에 양방향 선형 보간으로 p의 texture 색을 구한다



- 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);

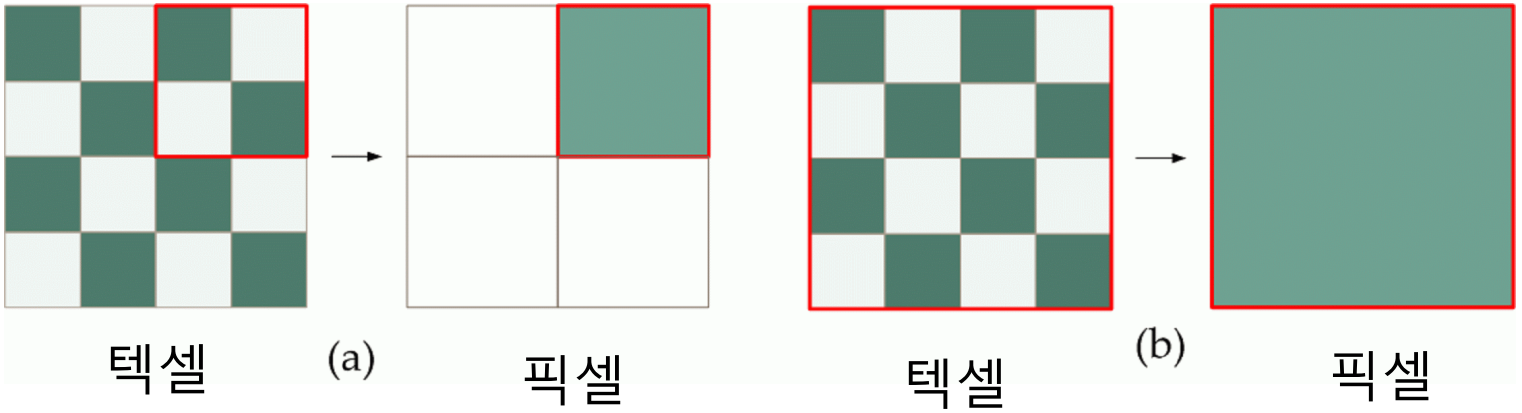




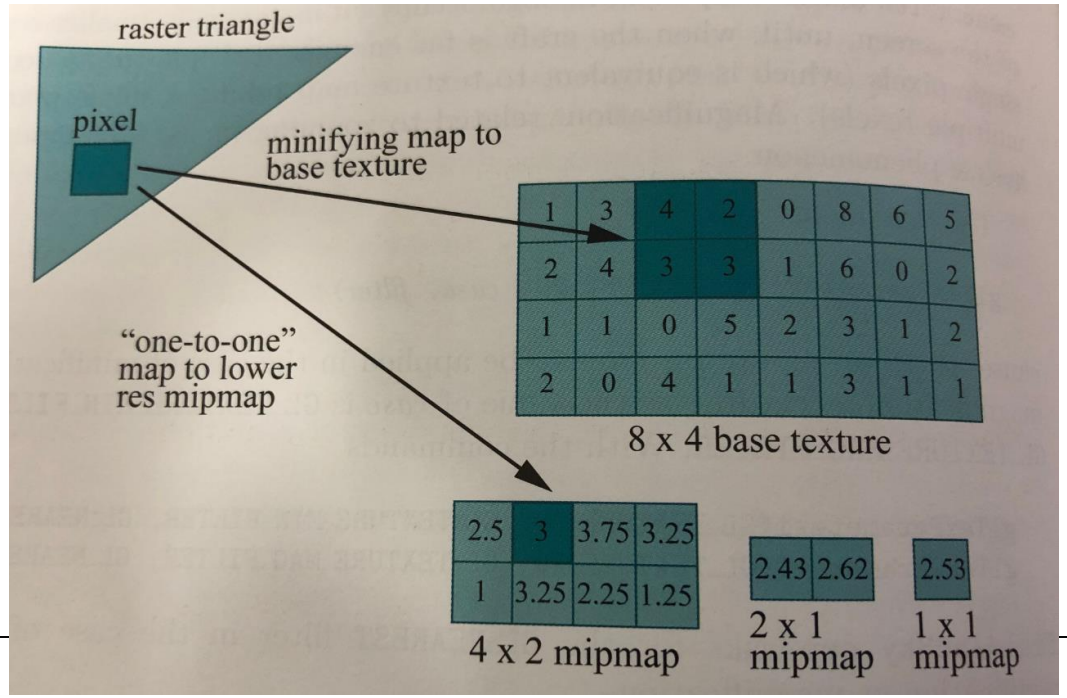
- **https://www.dropbox.com/s/ohkxgg8lygriz8f/texture_3.txt?dl=0**

- **밉 매핑 (MipMapping)**

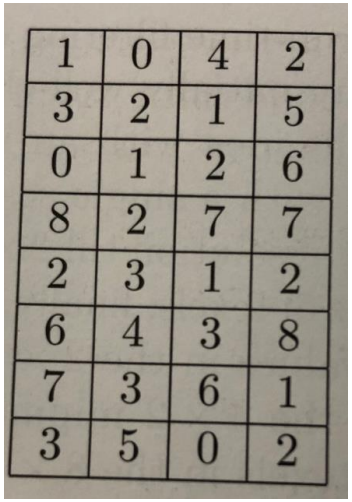
- 아래 그림 (a)를 보면 한 화소가 4개의 텍셀에 걸쳐 있다. (b)에는 한 화소가 16개의 텍셀에 결쳐 있다
- 이러한 **축소 관계**인 경우에는 텍셀의 평균을 구하여 해당 화소를 칠함으로써 anti-aliasing 효과를 줄 수 있다. 이러한 계산을 **미리 수행하여 저장**함으로 써 처리 속도를 높인 것을 **밉 매핑 (MipMapping)**이라 한다



- MipMapping의 예: 다음은 가로 8개 , 세로 4개로 구성된 (8x4, 너비x높이), 총 32개의 텍셀에 대하여 Mipmapping을 수행한 예이다
- 단순화 하기 위하여 각 텍셀에는 R,G,B가 아닌 하나의 color 값을 갖는다고 가정하였다
- 2x2 텍셀 (정사각형 형태의)에 대하여 color 값을 평균내면서 동작한다
- 이 예의 경우 최초 8x4에서 4x2, 2x1까지 줄고 최종적으로 1x1 mipmap까지 계산할 수 있다



- 예: 다음의 4x8 texture는 각 texel에 대한 single color 값들이 저장되어 있다. 모든 mipmap을 찾아보자



| | | | |
|---|---|---|---|
| 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 |

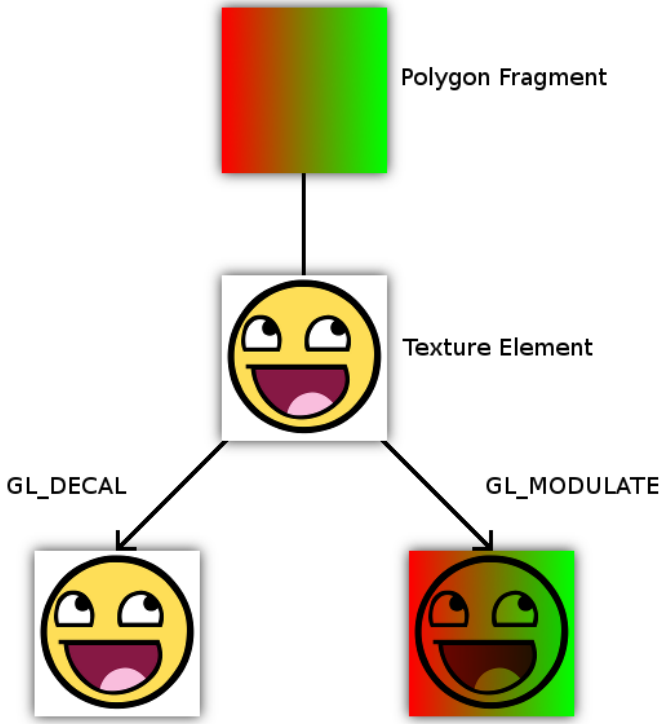
- **OpenGL에서의 texture 환경 명시**

- Texture를 물체면에 그대로 입힐 수도 있지만 물체색과 조합 가능
- 이러한 기능을 수행하는 함수가 glTexEnvf ()이다

```
예) glTexEnvf(GL_TEXTURE_ENV,
GL_TEXTURE_ENV_MODE, GL_DECAL);
```

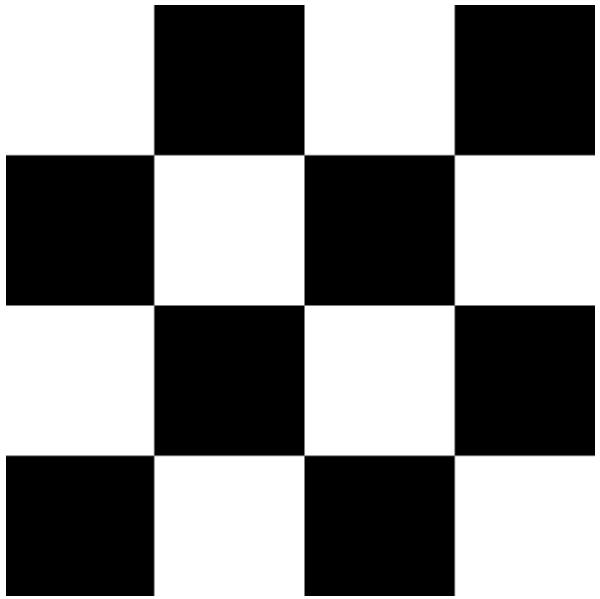
- 첫번째 인자: 반드시 GL_TEXTURE_ENV
- 두번째 인자: GL_TEXTURE_ENV_MODE로 하면 texture의 색과 물체면의 색을 어떻게 조합할지 명시
- 세번째 인자:
- GL_REPLACE: 기존 물체면의 색을 완전히 텍스쳐 색으로 대체
- GL_MODULATE: 기존 물체면의 색과 TEXTURE의 색을 곱함

# GL_MODULATE



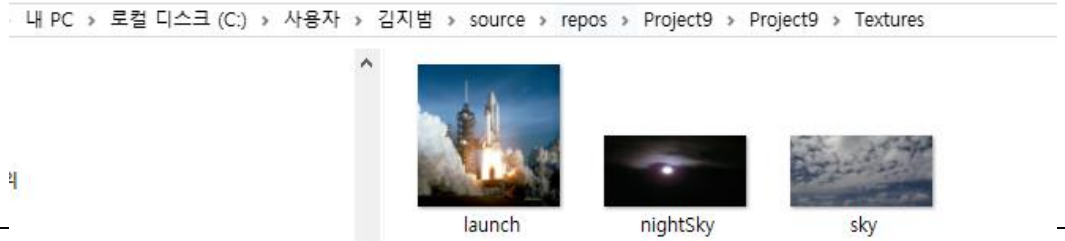
- **https://www.dropbox.com/s/g5o951ed8y2ks92/texture_4.txt?dl=0**

- **glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);**



- 실제 image 파일의 texture mapping

- 이번엔 실제 image 파일을 이용해 texture mapping을 해본다
- 시작하기 전에
- 1. 현재 프로젝트가 있는 폴더에 Textures 폴더를 만들고 다음과 같은 *.bmp 파일들을 옮겨 놓자
- (e-learning에 texture로 쓸 image 파일 세 개 있음)
- 2. 현재 코드는 24-bit BMP 이미지 포맷을 사용하므로 다른 이미지 포맷을 사용하려면 .bmp 포맷으로 변환해야 한다
- 3. 이러한 조건이 맞지 않으면 image들을 resize해야 한다
- ** 이미지의 가로 세로 크기가 $2^n$(예: 512 by 256 )



- **https://www.dropbox.com/s/dwoc78m774ii4ir/texture_5.txt?dl=0**

- // sky.bmp 이미지 파일을 불러서 texture로 만든후 간단한 사각형 QUAD에 texture mapping 시킨다
- // Projection: glOrtho(-2, 2, -2, 2, -1, 1);
- 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();

- 코드 분석
- 1. 먼저 BitMapFile 구조체를 만든후에
- 2. getBMPData 함수로 image 파일 읽는다
- image[0] = getBMPData("Textures/sky.bmp");
- 3. 아래와 같이 이  texture에 대해 glTexImage2D 를 이용해 Texture 를 명시
- glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image[0]->sizeX,
- image[0]->sizeY, 0, GL_RGB, GL_UNSIGNED_BYTE,
- image[0]->data);
- // 4번째 인자: texture image의 width, 5번째 인자: texture image의 height
- // 마지막 인자: 이미지 데이터의 포인터 설정

- **4. Texture mapping**
- **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();**
- **5. Texture parameter 명시**
- **glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT);**
- **glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);**
- **glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,**
  **GL_NEAREST);**
- **glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,**
  **GL_NEAREST);**

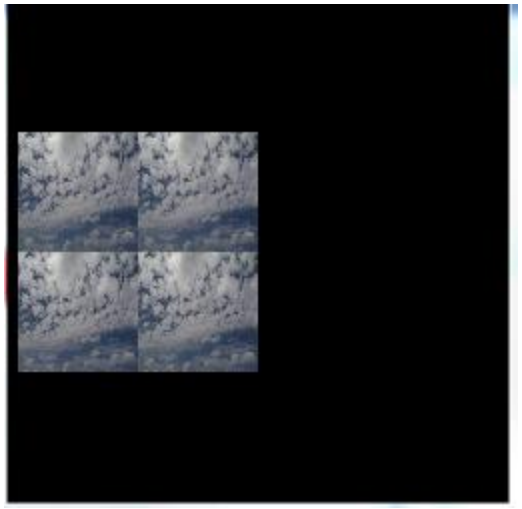
**void glPixelStorei( GLenum *pname*, GLint *param* );**

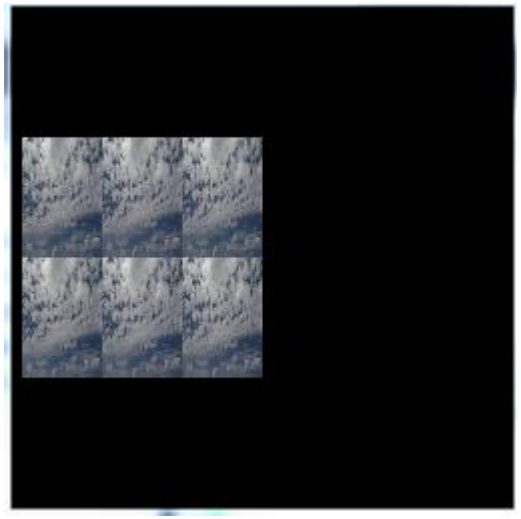
:이미지를 메모리로부터 읽어오거나 저장함.

GL_UNPACK : 이미지를 메모리로부터 읽어옴.

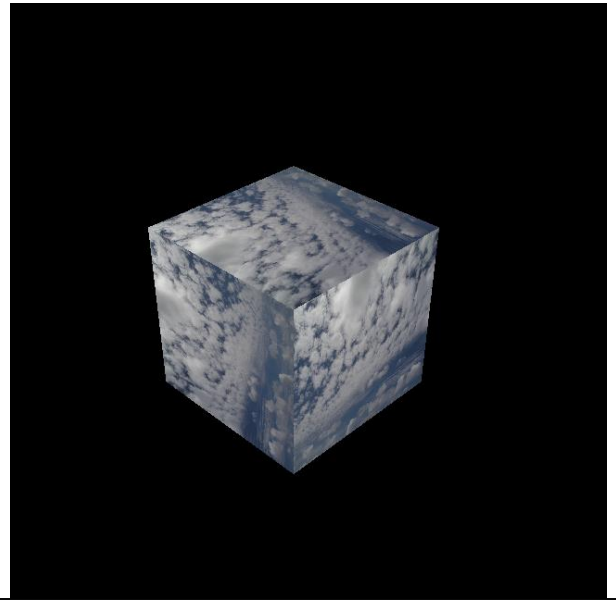
ALIGNMENT ,1: 순차적으로 메모리를 참조함.

- 예: texture 좌표를 적절히 조절하고 GL_REPEAT을 사용하여 다음과 같이 만들어 보자



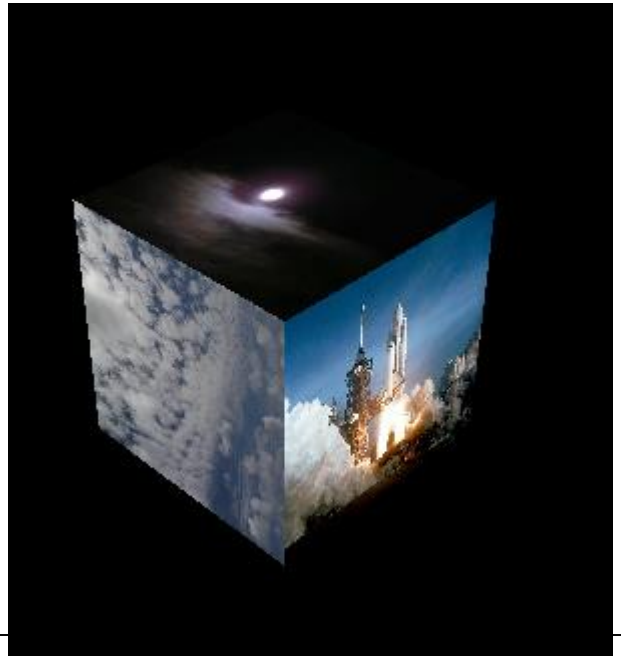


- 예: 앞에서 사용한 3개의 polygon에 texture를 씌워서 다음과 같이 보이게 만들어 보았다
- Projection:  gluPerspective(45.0,  1.0, 1, 50.0);
- viewing transformation:  gluLookAt(5, 5, 5, 0 ,0, 0, 0, 1, 0);



- **https://www.dropbox.com/s/tzs32kzllia3odh/texture_6.txt?dl=0**

- 각 polygon 면에 다른 texture image를 입혀보았다



- **https://www.dropbox.com/s/w0nxlkpy0a5xwdm/texture_7.txt?dl=0**

- **Texture 객체**

- 만일 장면 내에 여러 물체 (혹은 polygon)가 있고, 물체 별로  서로 다른 texture를 적용하려면 그 때마다 CPU 메모리로 부터 새로운 texture를 로드해야 하므로 효율성이 떨어진다
- 이러한 단점을 없애기 위하여 texture object 개념이 도입되었다
- 즉, texture와 해당 texture에 해당하는 파라미터 값을 묶어서 하나의 texture object 를 정의 할 수 있다

- **OpenGL에서의 texture object 사용**

```
void glGenTextures(GLsizei N, GLuint  *TextureNameArray
```

- **N: 할당될 객체의 수, TextureNameArray: Texture 객체명이 저장될 배열명**
- **예: glGenTextures(2, texName);  => texName[0], texName[1] 사용**
- **Texture 객체 생성**

```
void glBindTexture(GLenum  Target, Gluint TextureName);
```

- **Target: Texture 종류, TextureName: Texture 이름**
- **예: glBindTexture(GL_TEXTURE_2D, texName[0]);**

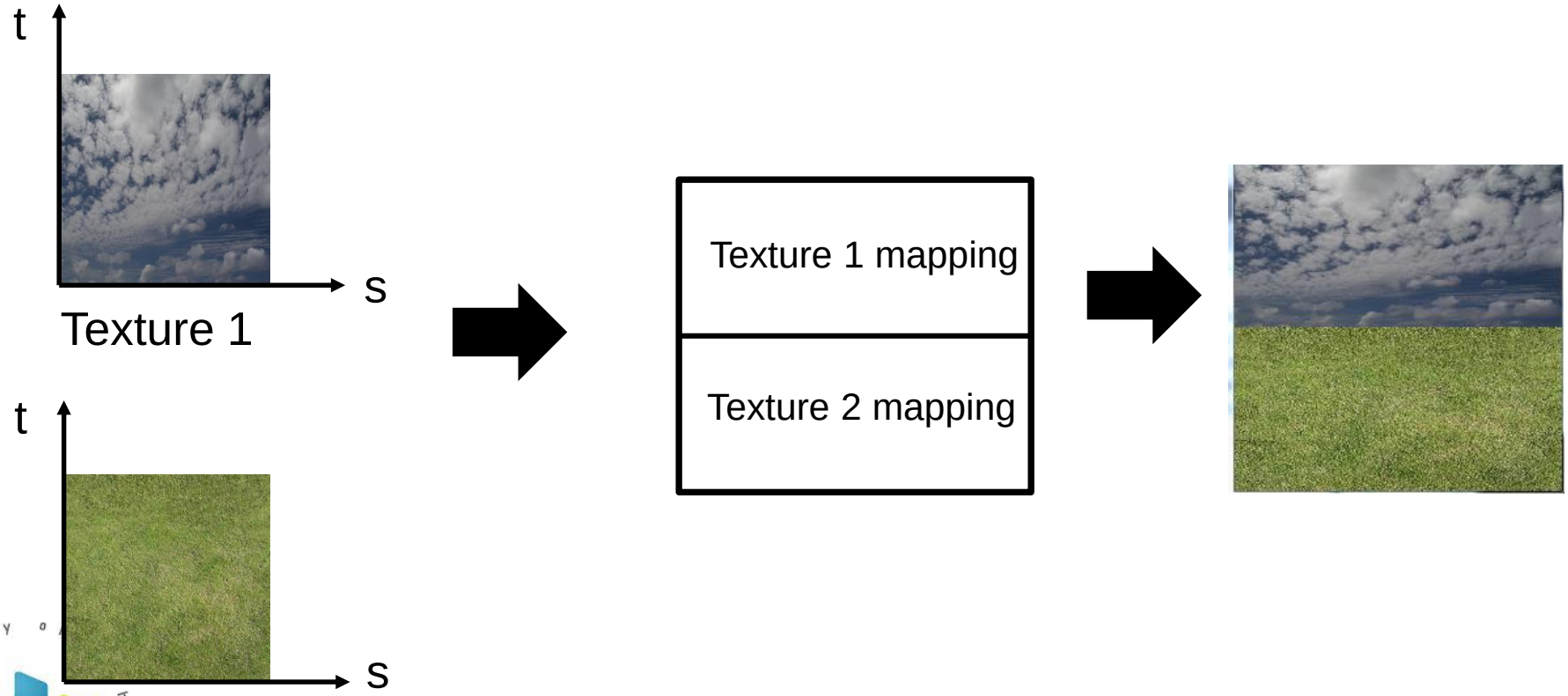
- glBindTexture() 함수는 texture object 를 생성할 때도 호출되지만 생성된 texture object 를 물체에 적용할 때도 호출된다

```
glBindTexture(GL_TEXTURE_2D, texName[0]);

glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_REPEAT)
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, image[0]->sizeX, image[0]->sizeY, 0, GL_RGB, GL_UNSIGNED_BYTE,
        image[0]->data);
```

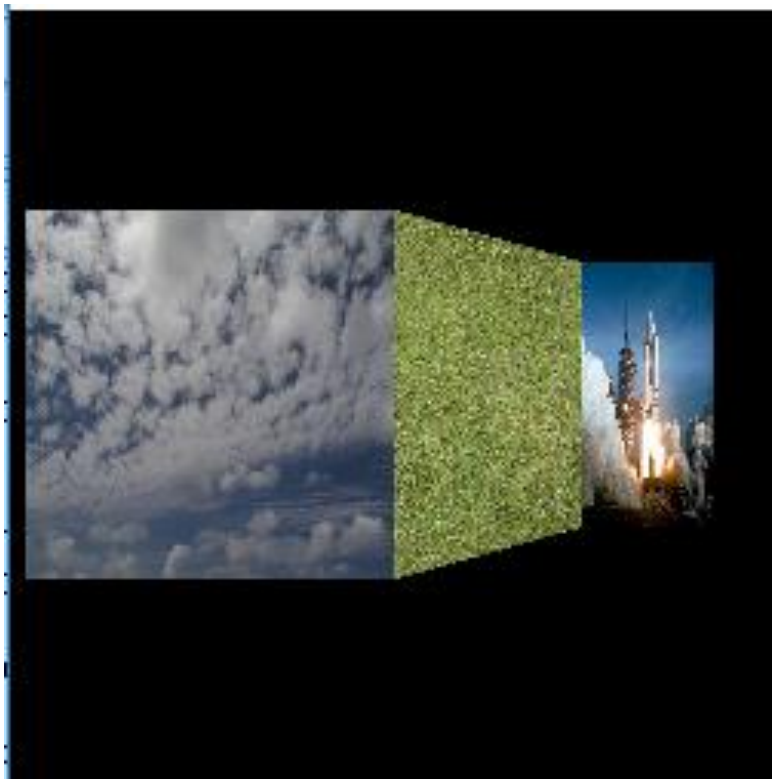
- **Texture object 를 이용한 texture 예제**

- **Textutre 객체를 사용 (glGenTextures(2, texName);) , 2개의 texture 객체 만든후 GL_QUADS를 2번 사용하여 위 polygon에는 texture1 (texName[0]), 아래 polygon에는 texture 2 (texName[1])를 mapping 시켜 보았다**



- **https://www.dropbox.com/s/59buclj2ci2pa80/texture_10.txt?dl=0**

- texture object 를 사용하고 3개의 texture를 생성한 후 다음과 같이 polygon에 texture를 mapping해 보았다



- **https://www.dropbox.com/s/47ieb13bts6fcab/texture_9.txt?dl=0**

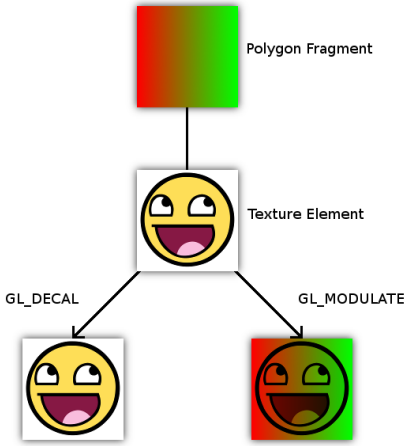
- **Lighting textures**

- Lighting과 texture는 동시에 옵션 조절을 통하여 사용 가능하다

```
glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, parameter)
```

- parameter가 GL_REPLEACE인 경우에는 texture의 color가 현재 pixel의 color (즉, material의 color)를 덮어 씌운다
- 만일 parameter가 GL_MODULATE라면 각 픽셀에서 원래 material의 color에다가 texture mapping의 결과로 나온 색을 곱해서 최종 색으로 사용한다
- Texture에 lighting을 같이 사용하는 경우 이 옵션을 가장 흔하게 사용한다

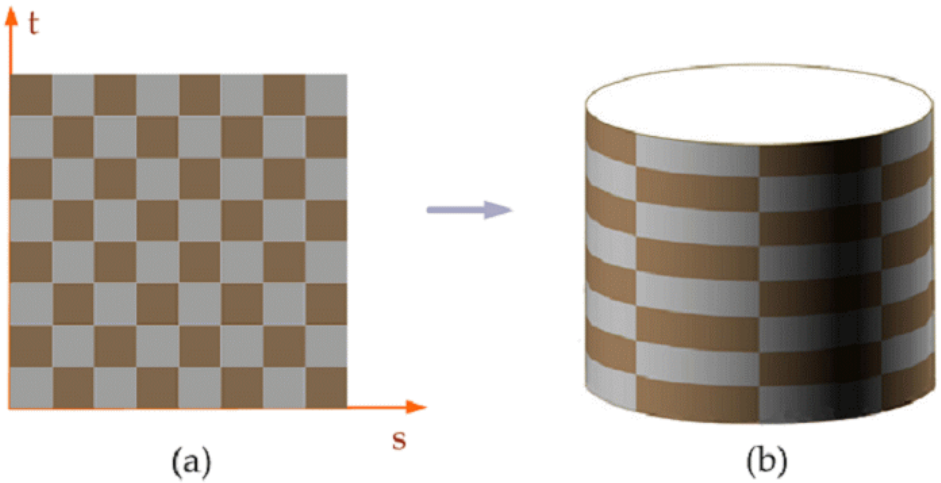
- 다음 코드는 앞에서 사용한 field and sky 예제에다가 lighting 을 적용한 것이다. 최초에는 아래와 같이 'GL_REPLACE'로 되어 있다
- glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_REPLACE);
- 이를 'GL_MODULATE'로 바꾸어 보자




- 코드에서는 방향성 광원을 사용하였고 이를 키보드 콜백함수와 연동시켰다
- float lightPos[] = { cos( (PI/180.0)*theta ), sin( (PI/180.0)*theta ), 0.0, 0.0 };
- 키보드 방향을 통하여 빛의 세기 및 방향을 조절할 수 있다
- https://www.dropbox.com/s/tlco8odc7zywg3q/texture_8.txt?dl=0

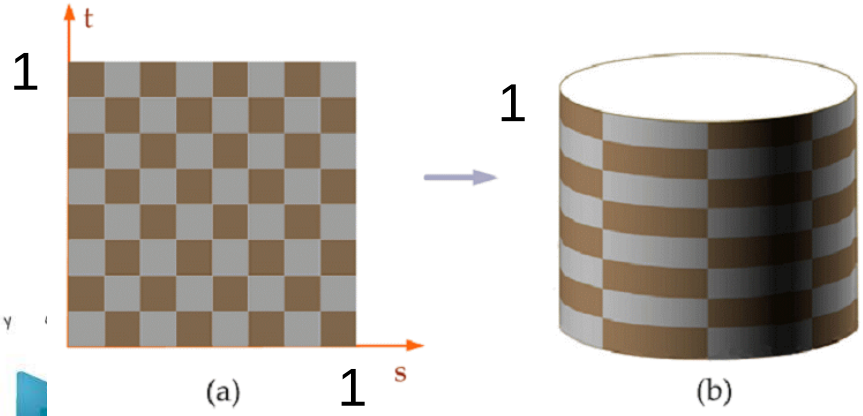
- 곡면의 parametric form을 이용한 texture mapping

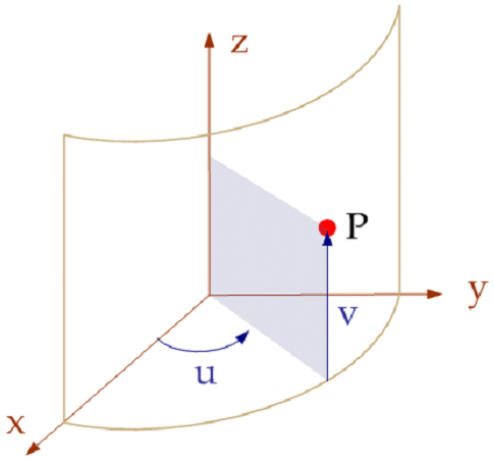
- 지금까지는 평면에 texture mapping을 수행하였다
- 곡면에 texture mapping을 수행하는 일반적인 방법은 곡면의 parametric form을 이용하는 것이다. 아래와 같이 평면에 있는 2차원의 Texture를 매개변수를 이용하여 곡면에 texture mapping을 하는 방법에 대하여 공부해 보자
- 먼저, 곡면을 매개 변수 (u ,v)로  표현할 수 있는 경우에 대해서 살펴보자
- 이를 곡면의 parametric form (매개변수)을 이용한 표현이라고 한다
- 이러한 경우 비교적 손쉽게 texture space와 곡면을 mapping 시킬 수 있다



- 원기둥 표면 상의 점 P (x, y, z)는 parametric form으로 P (u, v) 로 표현 가능
- r : 원기둥의 반지름 (고정 값)
- u: 점 P를 x-y평면에 투영시 양의 x축 방향에서 반 시계 방향으로 측정한 각 u
- v: 점 P의 양의 z축에 대한 높이

$$x = r\cos u,\ y = r\sin u,\ z = v$$

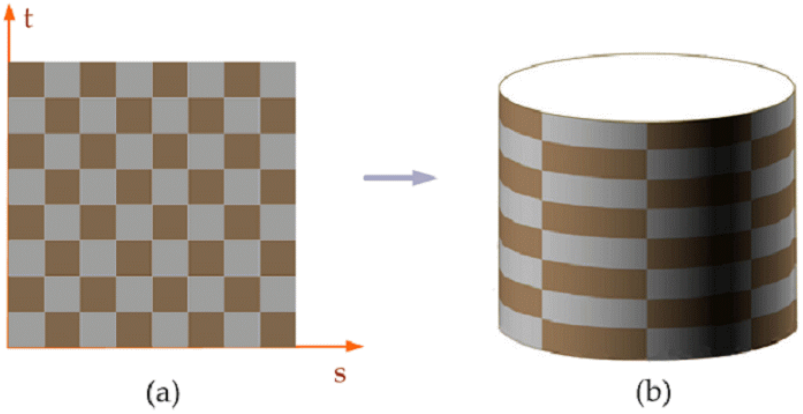


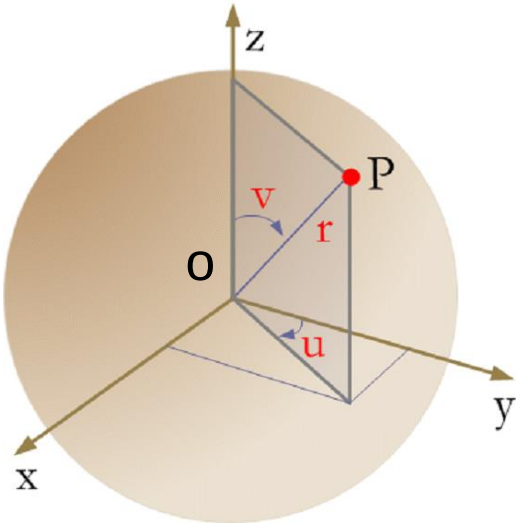
- Texture space에서 s: [0, 1] , t: [0, 1] 에서 정의됨. u: [0, 2π], v: [0, 1] 정의
- 즉, s가 0에서 1로 갈 때, u는 0에서 2π 각도를 커버해야 한다
- t가 0에서 1로 갈 때, v는 0에서 1로 가면 된다
- $u = 2\pi s, v = t$, 앞에서 $x = r\cos u,\ y = r\sin u,\ z = v$

$$x = r\cos 2\pi s,\ y = r\sin 2\pi s,\ z = t$$

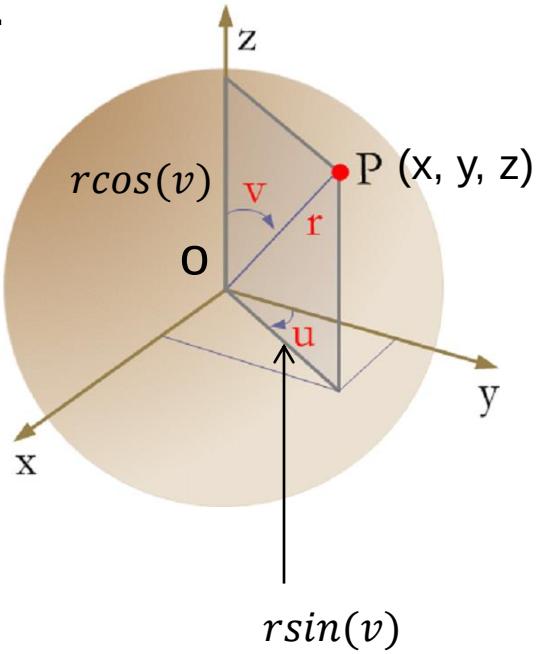
- 즉, Texture space 상의 (s, t)를 원통 표면의 (x, y, z)로 mapping 가능



- 이와 같은 곡면의 parametric form을 이용한 texture mapping은 비슷한 방법으로 구 (sphere)에도 적용 가능하다
- 지구를 구라고 생각하면 지구 표면 위의 어떤 위치를 위도와 경도로 나타내는 것과 유사하다
- 구 표면 상의 점 P (x, y, z)는 P (u, v) 로 표현 가능
- r: 원의 반지름 (고정 값)
- v : 선분 OP와 z축의 양의 방향이 이루는 각
- u: P를 x-y 평면에 투영했을 y축의
- 양의 방향과 이루는 각



- 구 표면 상의 점 P (x, y, z)는 P (u, v) 로 표현 가능
- r: 원의 반지름 (고정 값)
- v : 선분 OP와 z축의 양의 방향이 이루는 각
- u: P를 x-y 평면에 투영했을 y축의
- 양의 방향과 이루는 각



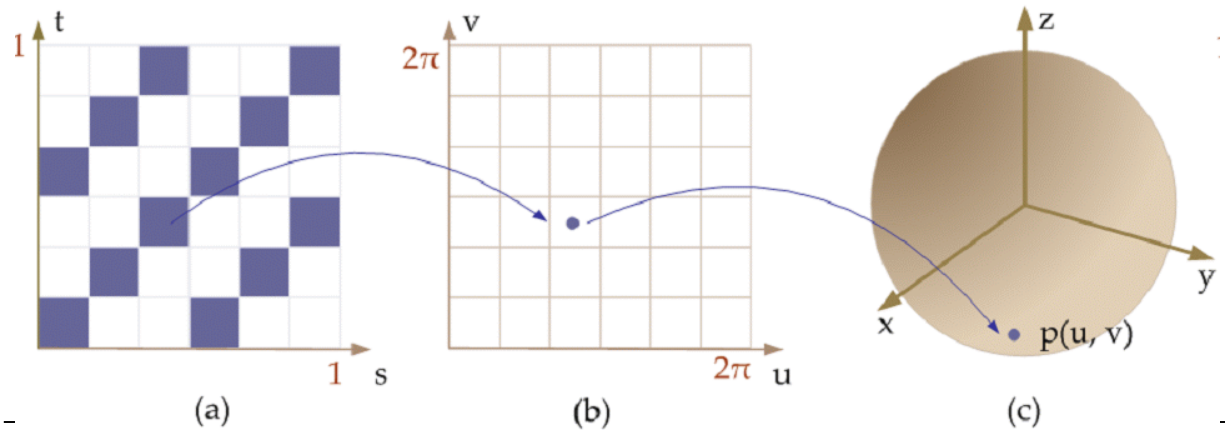

$$z = r\cos v$$

$$y = r\sin v\ \cos u$$

$$x = r\sin v\ \sin u$$

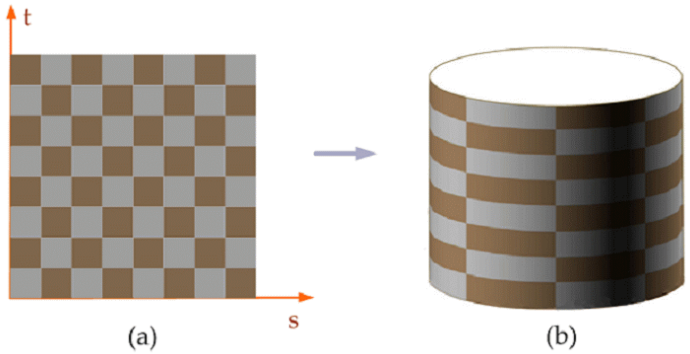
- Texture space에서 s: [0, 1] , t: [0, 1] 에서 정의됨
- u: [0, 2π], v: [0, 2π]에서 정의
- 즉, s가 0에서 1로 갈 때 u는 0에서 2π 각도를 커버
- t가 0에서 1로 갈 때 v는 0에서 2π 각도를 커버
- $u = 2\pi s$ , $v = 2\pi t$, 이 식들을 앞에 식에 대입

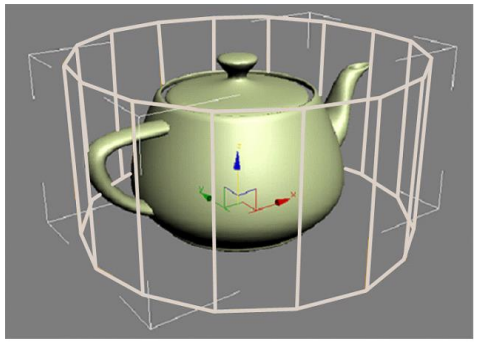
$$
\begin{aligned}
z &= r\cos 2\pi t \\
y &= r\sin 2\pi t \ \cos 2\pi s \\
x &= r\sin 2\pi t \ \sin 2\pi s
\end{aligned}
$$



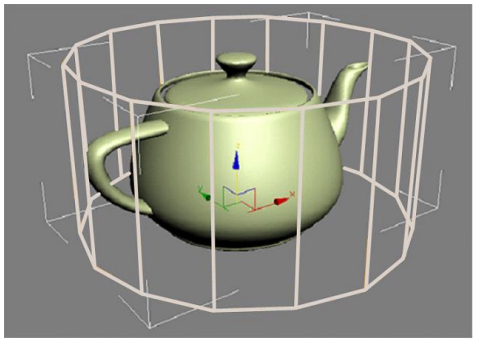
- **2-stage mapping을 이용한 곡면의 texture mapping**

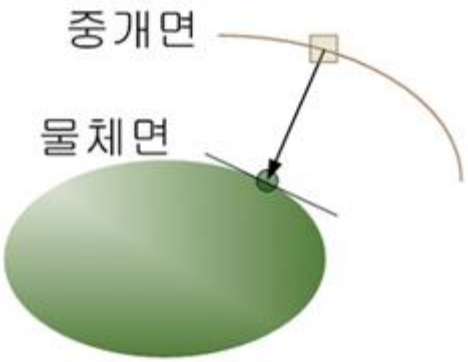
- 하지만, 많은 경우에는 곡면이 원기둥이나, 구가 아니다. 이러한 경우에는 어떻게 texture mapping을 진행해야 하나?
- 이러한 경우에는 2단계 mapping (2-stage mapping)을 이용한다
- 1단계 mapping: S 매핑: Texture를 중간 단계면 (중개면)에 입힌다
- 중개면으로는 원기둥, 구, 육면체 등이 흔히 사용된다



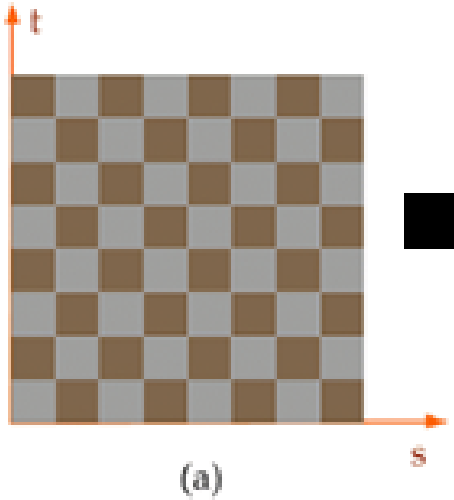



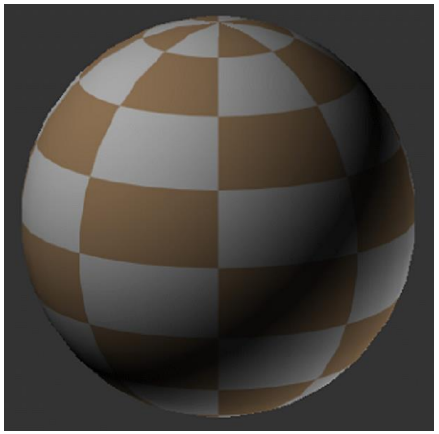
- **2단계 mapping: O 매핑: 중개면 내부에 실제로 TEXTURE를 가할 물체를 넣고 texture를 입히는 작업**
- **물체면의 법선 벡터가 중개면과 만나는 점 (위치)를 구한 뒤, 그 점의 texture값을 해당 물체면의 texture로 사상한다**



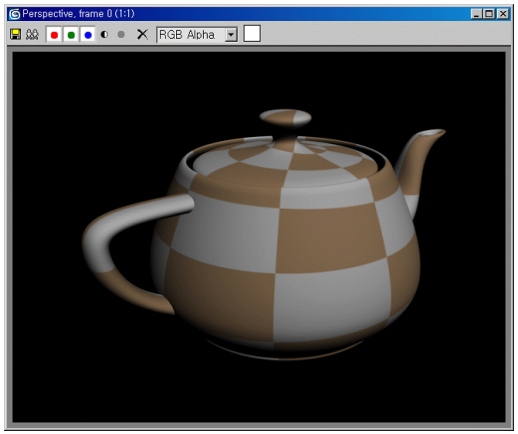


- 주전자 (곡면)에 대한 2단계 mapping (2-stage mapping)의 예



1단계 mapping



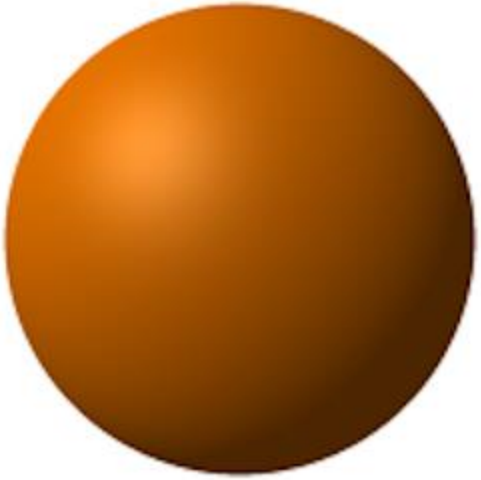

2단계 mapping

- **Parametric form을 이용한 torus로의 texture mapping 예**
- **https://www.dropbox.com/s/r8sos7h02urxjty/texture_2.txt?dl=0**

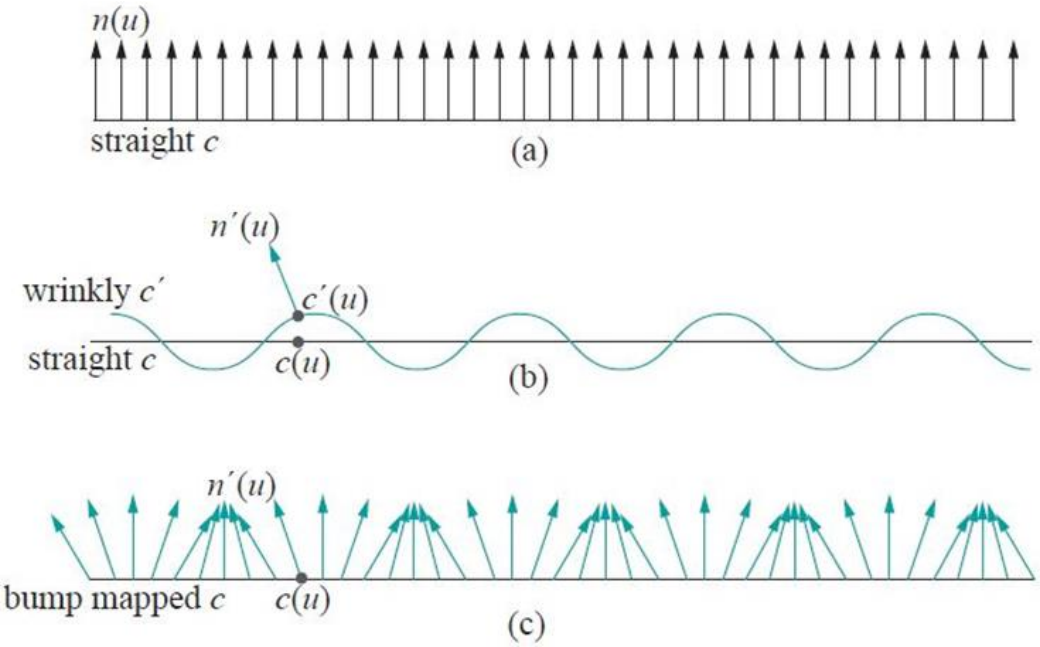
- **Bump mapping**

- Bump 의 의미: small deviations (벗어남) along the normal direction from the surface
- Bump mapping
- Texture mapping과 더불어서 visual적인 realism을 위한 bump mapping 기법이 있다
- 랜더링될 물체를 픽셀마다 표면의 법선 벡터를 perturb (교란, 왜곡)시켜서 높낮이가 있어 보이게 만드는 기술

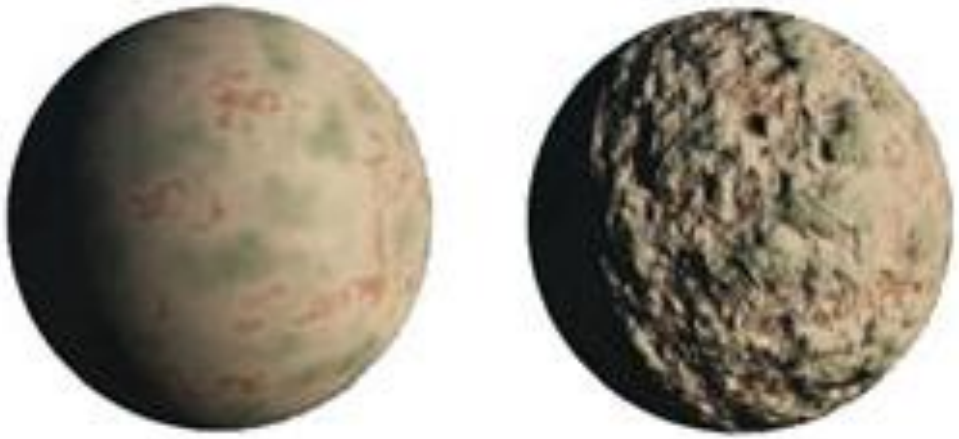
- **Bump mapping**
- (우)의 경우 오렌지와 비슷하게 표면이 올록볼록해 보인다
- (좌) bump mapping 적용전의 구
- (우) bump mapping 적용후의 구



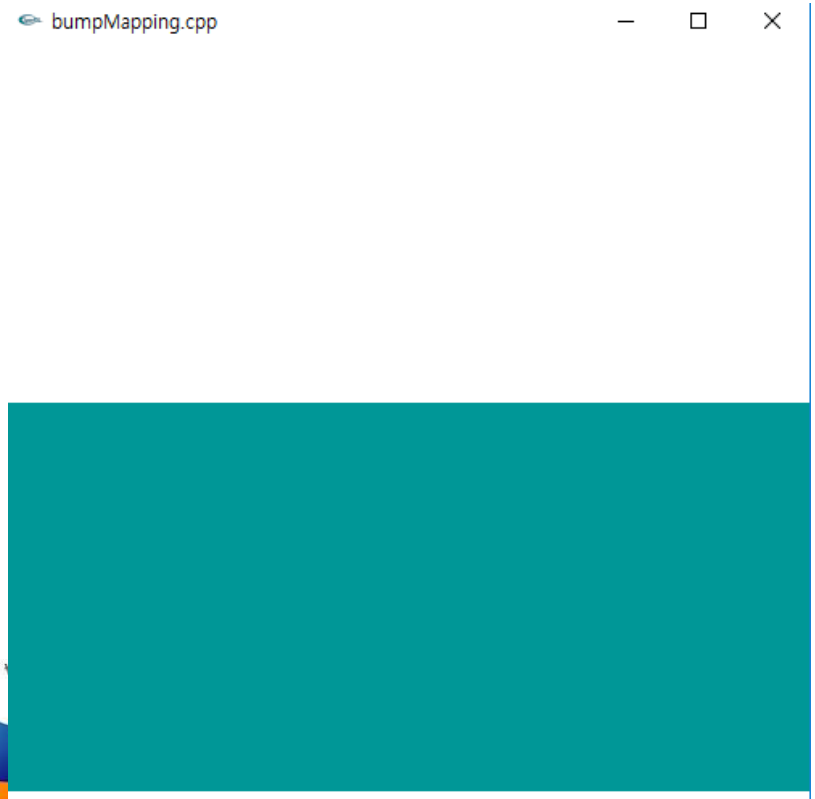
- **Bump mapping의 idea**
- **(a) the original curve c and its true unit normal n(u)**
- **(b) the wrinkled curve c' and its unit normal n'(u) at a single point c'(u)**
- **(c) bump-mapped c with redefined n'(u)**

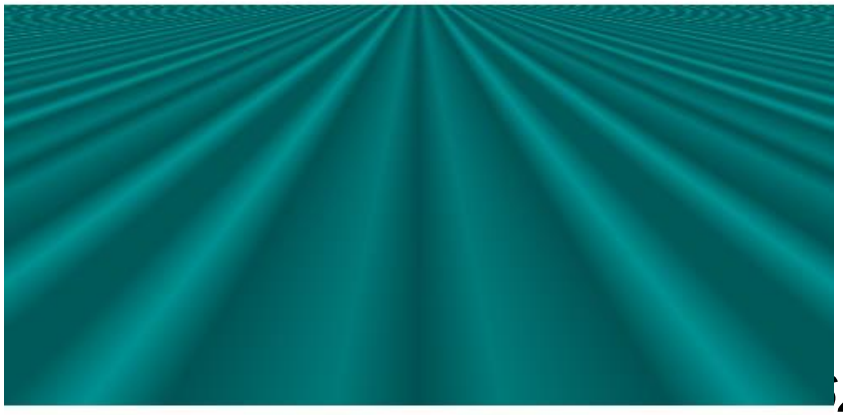


# 구에 texture mapping후 Bump mapping 전, 후의 비교



- 다음은 plane에 bump mapping을 적용 전 후의 비교이다
- ‘space’ 키로 bump mapping 전, 후를 비교해 보자



**https://www.dropbox.com/s/ebpf89zkbjgv0co/bumpmapping.txt?dl=0**