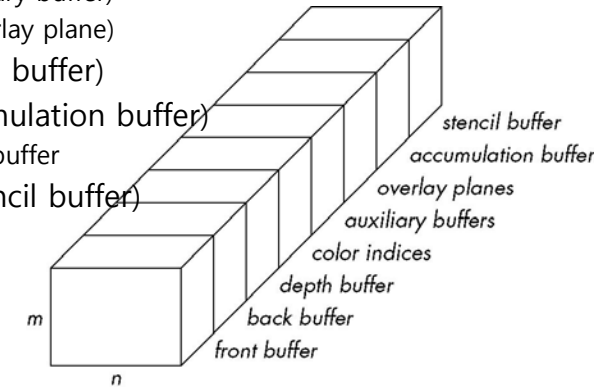


# Buffer, Image, and Texture Mapping

514780  
2018년 가을학기  
11/15/2018  
단국대학교 박경신

## OpenGL Frame Buffer

- 색 버퍼 (Color buffers)
  - 전면 버퍼 (Front buffer)
  - 후면 버퍼 (Back buffer)
  - 보조 버퍼 (Auxiliary buffer)
  - 오버레이면 (Overlay plane)
- 깊이 버퍼 (Depth buffer)
  - High resolution buffer
- 누적 버퍼 (Accumulation buffer)
  - Holds masks
- 스텐실 버퍼 (Stencil buffer)
  - Holds masks



## Buffer Selection

- OpenGL은 front, back, depth buffer 등 모든 버퍼로부터 데이터를 읽는 것이 가능
- 기본적 프레임버퍼로 back buffer가 선택되어 있으며, **glReadBuffer**를 이용하여 다른 버퍼로 바꿀 수 있음
- Note: 프레임 버퍼에 픽셀 포맷은 프로세서 메모리에 포맷과는 다름. 또한 이 메모리는 각각 다른 곳에 저장됨.
  - 따라서, 데이터 packing과 unpacking이 필요함
  - Reading은 다소 느릴 수 있음

## OpenGL Buffer Management Functions

- 버퍼를 clear
  - `glClear(GLbitfield mask);` // 특정 버퍼를 clear함
  - `GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_ACCUM_BUFFER_BIT | GL_STENCIL_BUFFER_BIT`
  - `glClearBuffer();`
- 버퍼의 clear value set
  - `glClearColor(); glClearDepth(); glClearDepthf(); glClearStencil();`
- 버퍼를 mask (i.e., enabled or disabled)
  - `glColorMask[i](GLboolean red, GLboolean green, GLboolean blue, GLboolean alpha);` // r,g,b,a 색을 프레임 버퍼에 쓸지 아닌지 지정
  - `glDepthMask(GLboolean flag);` // depth 버퍼에 쓸지 아닌지 지정
  - `glStencilMask(GLuint mask);` // stencil 버퍼에 bit mask를 쓸지 아닌지 지정

## Images

- 이미지 데이터는 파일에서부터 읽어 들이거나 직접 생성
- 일반적인 이미지 파일 포맷:
  - JPEG, TIFF, PNG, GIF, RGB, EPS, BMP, etc
- 이미지 포맷:
  - Color channel: greyscale, RGB, RGBA
  - Bit resolution
  - Compression: 손실 부호화 (lossy coding), 무손실 부호화 (lossless coding)

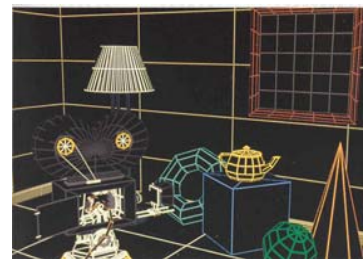
## COIN3D simage

- <http://www.coin3d.org/lib/simage>
- COIN3D simage library가 제공하는 이미지 파일 포맷
  - JPEG, TIFF, PNG, PIC, TGA, EPS, GIF, RGB, etc
- COIN3D의 simage library를 사용하려면, 프로젝트에 additional library and include directory를 추가해야 한다.
  - Project -> Properties(ALT+F7) -> Configuration Properties -> C/C++ -> General에 Additional Include Directories에 **.Wininclude**를 넣는다.
  - Project -> Properties(ALT+F7) -> Configuration Properties -> C/C++ -> Preprocessor에 Preprocessor Definitions에 **;SIMAGE\_DLL**를 넣는다.
  - Project -> Properties(ALT+F7) -> Configuration Properties -> Linker -> General에 Additional Library Directories에 **.Wlibwdebug**를 넣는다.
  - Project -> Properties(ALT+F7) -> Configuration Properties -> Linker -> Input에 Additional Dependencies에 **simage1.lib**를 넣는다.

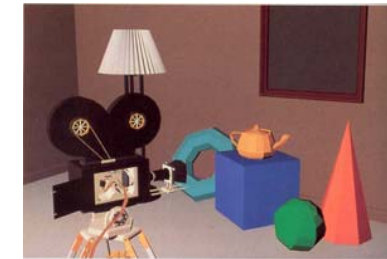
## COIN3D simage Example

```
unsigned char *imgPtr;
unsigned char *imageData;
unsigned char *rescaledImageData;
int imageWidth = 0, imageHeight = 0, numComponents = 0;
imageData = simage_read_image(filename, &imageWidth,
                             &imageHeight, &numComponents); // read
GLsizei xdim2, ydim2;      // 만약 이미지 크기가 2의 승수가 아니면, resize
GLenum type;
xdim2 = 1;
while (xdim2 <= imageWidth)
    xdim2 *= 2;
xdim2 /= 2;
ydim2 = 1;
while (ydim2 <= imageHeight)
    ydim2 *= 2;
ydim2 /= 2;
if ((imageWidth != xdim2) || (imageHeight != ydim2)) {
    rescaledImageData = simage_resize(imageData, imageWidth, imageHeight,
                                     numComponents, xdim2, ydim2);
    imgPtr = rescaledImageData;
} else
    imgPtr = imageData;
```

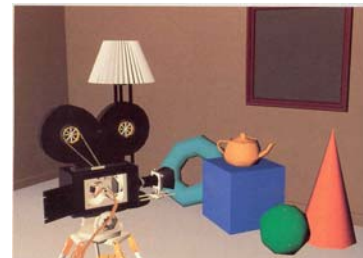
## Texture Mapping



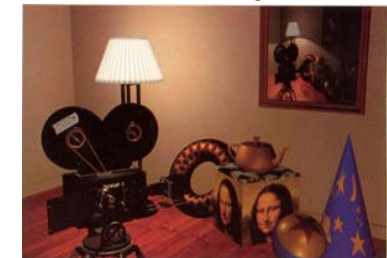
Wireframe



Flat shading



Smooth shading



Texture mapping

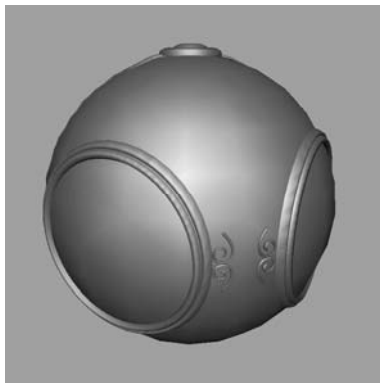
## The Limits of Geometric Modeling

- 기하학적 모델링
  - 고성능 그래픽카드가 초당 1000만개의 폴리곤을 렌더링할 수 있다 하더라도 구름, 초원, 지형, 스킨 (skin) 등과 같은 객체/현상을 기하학적 텍스처들로 모델링하여 처리하기엔 무리가 있음.
- 텍스처 맵핑
  - 2차원 영상을 직접 평면 표면에 입힘.
  - 제한된 수의 다각형을 사용해야 하는 실시간 렌더링에 있어서 비교적 적은 추가 비용으로 이미지의 사실성을 상당히 높일 수 있는 기법임.

## Three Types of Mapping

- 텍스처 맵핑 (Texture Mapping)
  - 이미지를 사용하여 다각형 (polygon)의 내부에 그려짐.
- 환경 맵핑 (Environment/Reflection mapping)
  - 환경 이미지가 렌더링될 표면 위에 그려짐.
  - 반사맵 (reflection map) 또는 환경맵 (environmental map)은 반사광을 추적하지 않고도 광선 추적법에 의한 매우 반짝이는 표면 (highly specular surface)을 만들 수 있음.
- 범프 맵핑 (Bump mapping)
  - 렌더링 음영 작업시 법선벡터 (normals)를 왜곡시켜 실제 오렌지 표면의 융기와 같은 모양을 생성.

## Texture Mapping



geometric model

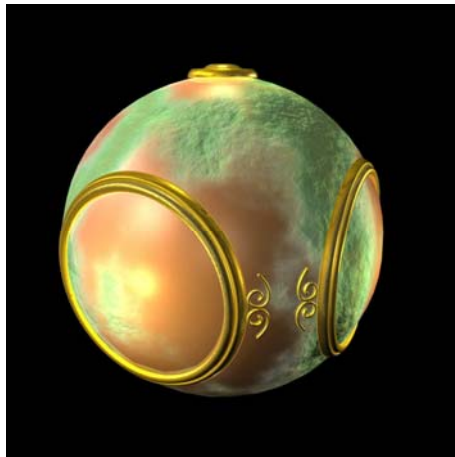


texture mapped

## Environment Mapping

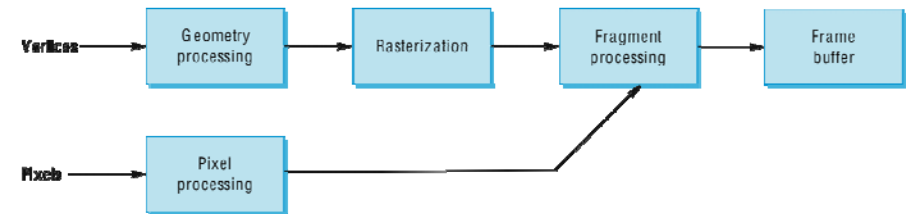


## Bump Mapping



## Where does texture mapping take place?

- 텍스처 맵핑은 렌더링 파이프라인의 마지막 단계에서 실행

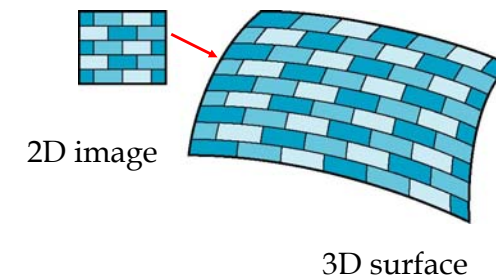


## Texture Mapping

- 개념적인 2차원 텍스처 맵핑 과정
  - 표면 매개화 (surface parameterization) 과정
    - 텍스처 이미지를 물체에 어떻게 입힐 것인가?
    - 물체의 각 점에 맵핑되는 텍스처 이미지의 좌표가 필요함
    - $(x_0, y_0, z_0) \Rightarrow (x_t, y_t)$
  - 기하 변환 과정
    - 기하 변환은 물체의 각 점과 투영 화면에서의 위치간의 맵핑 관계를 결정.
    - $(x_0, y_0, z_0) \Rightarrow (x_s, y_s)$
  - 래스터 과정
    - 각 기하물체가 투영되는 화소들을 찾아주는 과정
  - 텍스처 색 계산 과정
    - 각 화소에 적절히 텍스처 색으로 칠해주는 과정
    - 각 화소를 통하여 보이는 텍스처 이미지의 색을 어떻게 계산할 것인가?
    - 계산된 텍스처 색을 물체의 원래의 색깔과 어떻게 혼합할 것인가?

## 2D Texture Mapping

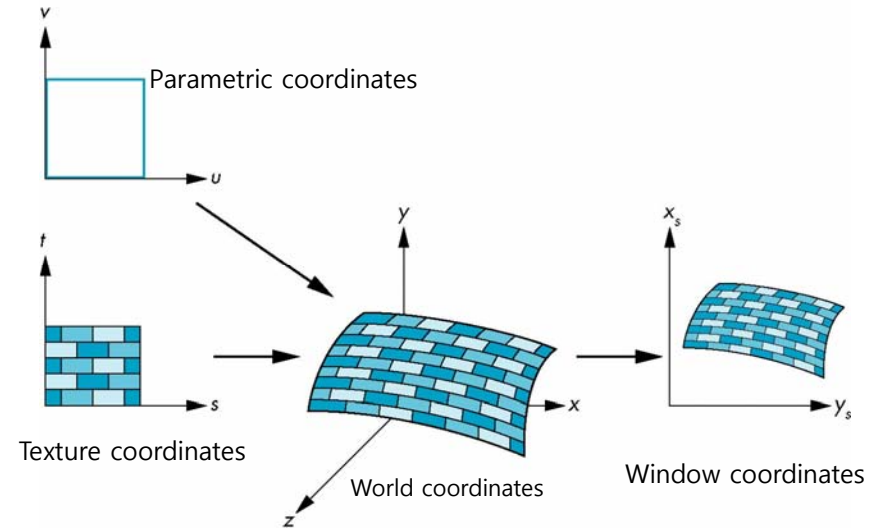
- 텍스처 맵핑은 2차원의 이미지를 3차원의 표면에 맵핑
  - 모두 3~4 개의 좌표계간 맵핑 처리를 포함한 일련의 과정이 필요함



## Coordinate Systems

- Parametric coordinates
  - 곡선과 곡면 (curves & surfaces) 모델링에 사용하는 매개변수형 좌표계 (u, v)
- Texture coordinates
  - 텍스처 내의 위치지정을 위해 사용하는 텍스처 좌표계 (s, t)
- Object or World Coordinates
  - 텍스처가 입혀질 물체를 기술하는 객체 좌표계 또는 세상 좌표계 (x, y, z)
- Window Coordinates
  - 최종적인 이미지가 형성되는 윈도우 좌표계 (x<sub>w</sub>, y<sub>w</sub>)

## Texture Mapping



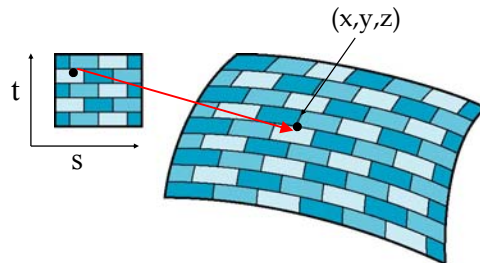
## Mapping Functions

- 텍스처는 생성하여 프로세서 메모리에 배열로 저장됨
  - 텍스처는 응용 프로그램에 의해 생성되거나 사진으로부터 스캔하여 생성할 수 있음
  - 이 배열의 요소들을 텍셀 (texture element = texel)이라 부름
- 텍스처 좌표에서 표면의 한 점으로 맵핑하는 함수
  - 텍스처 좌표:  $T(s, t)$
  - 객체 좌표:  $P(x, y, z)$

$$x = x(s, t)$$

$$y = y(s, t)$$

$$z = z(s, t)$$

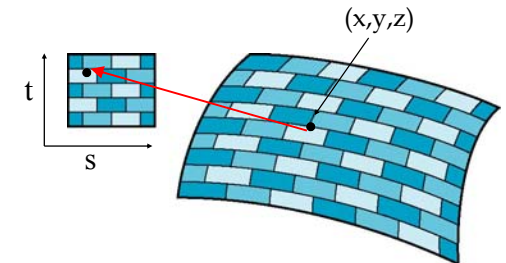


## Backward Mapping

- 물체 위의 점 (x, y, z)이 주어졌을 때, 대응하는 텍스처 좌표를 어떻게 찾을 것인가?
  - 한 픽셀의 음영을 결정할 때 텍스처 이미지의 어떤 점을 사용할지를 결정해야 할 경우, 화면좌표로부터 텍스처 좌표로의 맵핑을 필요로 함
- 텍셀  $T(s, t)$ 를 찾는 역함수

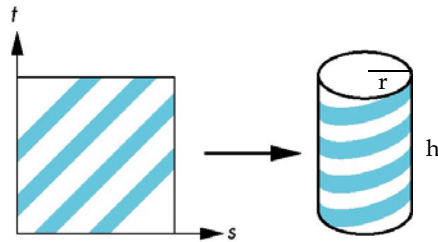
$$s = s(x, y, z)$$

$$t = t(x, y, z)$$



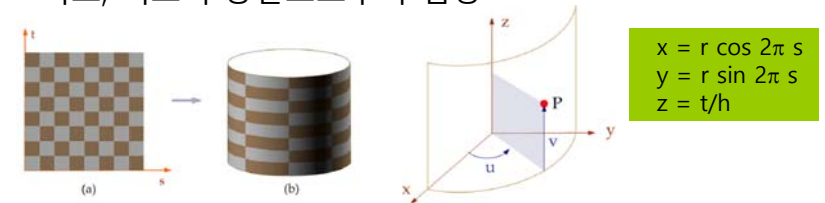
## Two-part mapping

- 맵핑 문제의 해법: Two-part mapping
  - 텍스처를 먼저 구(sphere), 원기둥(cylinder), 입방체와 같은 간단한 3차원 매개변수형 표면에 텍스처를 맵핑
- 예제: 원기둥 (cylinder)으로 맵핑



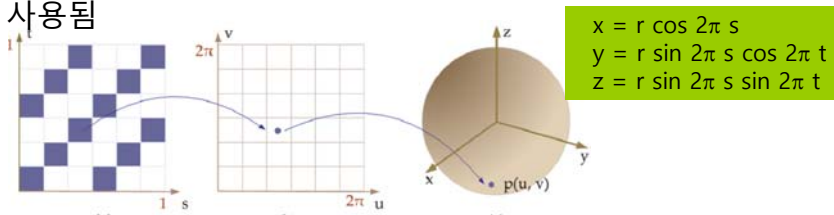
## Cylindrical Mapping

- 매개변수형 원기둥 함수 (parametric cylinder)
  - $x = r \cos 2\pi u$      $u: (0,1)$
  - $y = r \sin 2\pi u$      $v: (0,1)$
  - $z = v/h$
- $u, v$  공간의 사각형을 세상 좌표계의 반지름  $r$ 과 높이  $h$ 를 가진 원기둥에 맵핑
  - $s = u$
  - $t = v$
- 그리고, 텍스처 공간으로부터 맵핑



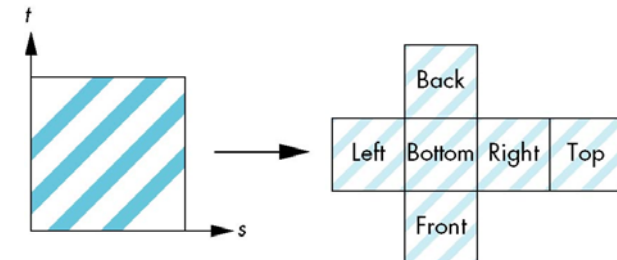
## Spherical Map

- 매개변수형 구 함수 (parametric sphere)
  - $x = r \cos 2\pi u$
  - $y = r \sin 2\pi u \cos 2\pi v$
  - $z = r \sin 2\pi u \sin 2\pi v$
- 원기둥 함수와 비슷함. 그러나 구와 같이 닫힌 객체에 맵핑할 경우 형상의 일그러짐이 필요함.
  - 메르카토르(Mercator) 투영의 경우, 양 극(pole)에 가장 큰 일그러짐을 만듦
- Spherical mapping은 환경맵(environmental maps)에서 사용됨



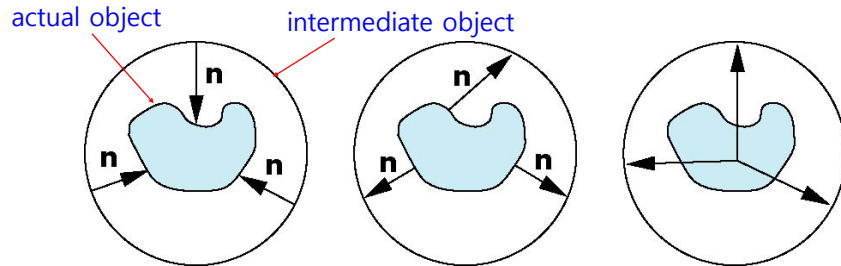
## Box Mapping

- 간단한 직교 투영 (simple orthographic projection)과 사용하기에 편리함
- Box mapping은 환경맵(environmental maps)에서 사용됨



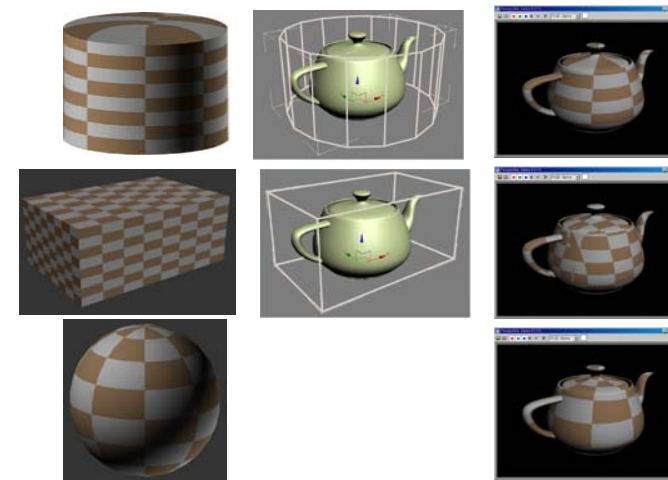
## Second Mapping

- 두번째 맵핑 (second mapping): 중간 객체 (intermediate object) 상의 텍스처 값을 원하는 표면 (actual object)에 맵핑
  - 중간 매개 표면으로부터의 법선 사용
  - 객체 표면으로부터의 법선 사용
  - 객체 중심으로부터의 벡터 사용



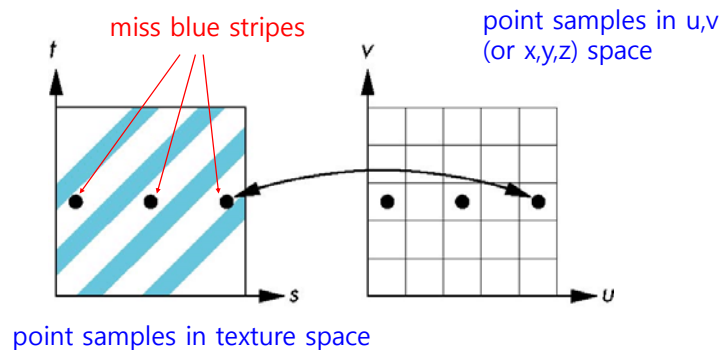
## Second Mapping

- 물체를 중개면 내부에 넣고 물체 면에 텍스처를 입힘.



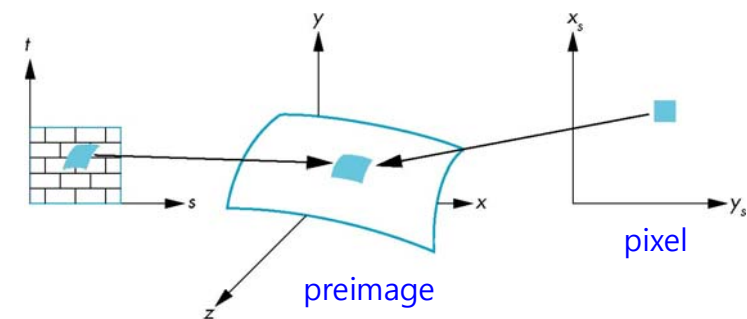
## Aliasing

- 텍스처의 포인트 샘플링 (point sampling) 방법은 앨리어싱 문제(aliasing error)를 만들 수 있음
  - 포인트 샘플링 - 점에서 점으로의 맵핑



## Area Averaging

- 보다 효과적이거나 다소 느린 영역 평균법 (area averaging) 방법을 사용 가능함
  - 영역 평균법 - 영역에서 영역으로의 맵핑



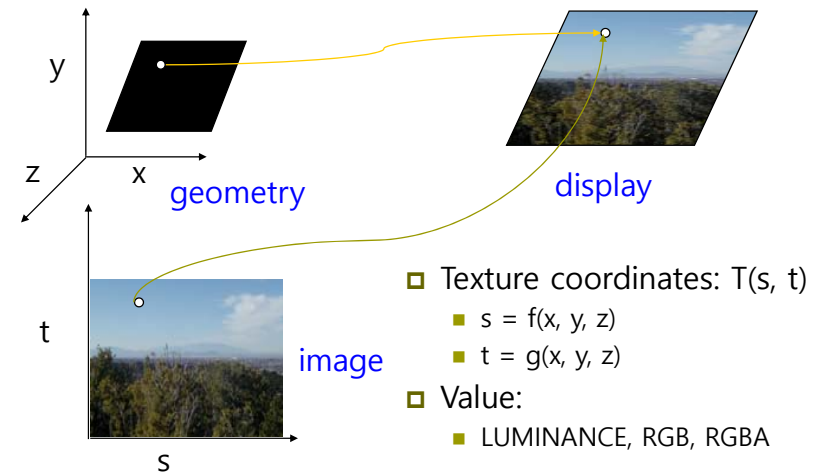
- Note: 화소의 전이미지 (preimage)는 곡면임

## Basic Strategy

### □ 텍스처를 입히기 위한 3가지 단계

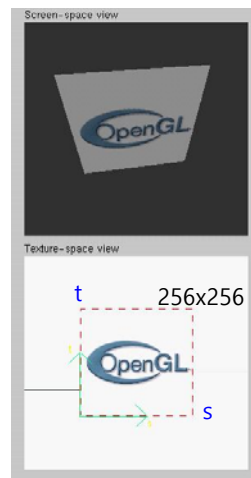
1. 텍스처를 지정
  - 이미지를 파일에서 읽거나 직접 생성
  - 텍스처에 할당
  - 텍스처링 활성화
2. 각 정점에 텍스처 좌표를 할당
  - 프로그래머가 직접 적절한 맵핑 함수 작성
3. 텍스처 맵핑 방법을 지정
  - 랩핑 (wrapping)
  - 필터링 (filtering)

## Texture Mapping



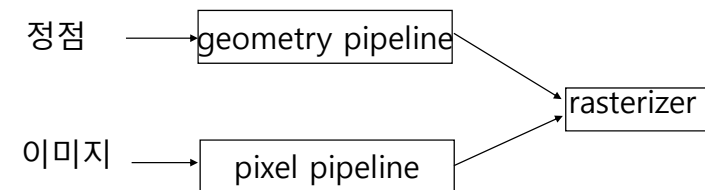
## OpenGL Texture Example

- 다음 예제는 256 x 256 이미지를 사각형 (rectangular polygon)에 텍스처 맵핑하여 원근 투영한 모습을 보여주고 있음.



## Texture Mapping in the OpenGL Pipeline

- 이미지와 기하학적 객체는 각기 다른 파이프라인에서 연산된 후 래스터 과정에서 합성됨
- 즉, 복잡한 텍스처가 있다 하더라도 기하학적 객체의 복잡성 (complexity)에 전혀 영향을 주지 않음





## Specifying a Texture Image in OpenGL

- 512x512크기의 이미지 imageData 텍스처 이미지를 프로그램에서 생성하거나 파일로부터 읽어서 CPU 메모리에 배열로 정의함
  - GLubyte imageData[512][512];
- 텍스처 맵핑을 활성화
  - glEnable(GL\_TEXTURE\_2D)
  - OpenGL은 1,2,3,4-차원 텍스처 맵을 지원함

## Define Image as a Texture

- glTexImage2D(target, level, components, width, height, border, format, type, texels );
  - target: 텍스처 타입, e.g., GL\_TEXTURE\_2D
  - level: mipmapping에서 사용
  - components: texel별 요소, e.g., RGB
  - width, height: texel의 너비와 높이 (in pixels)
  - border: smoothing에 사용
  - format, type: texel을 정의하는 포맷과 타입
  - texels: texel 배열에 대한 포인터

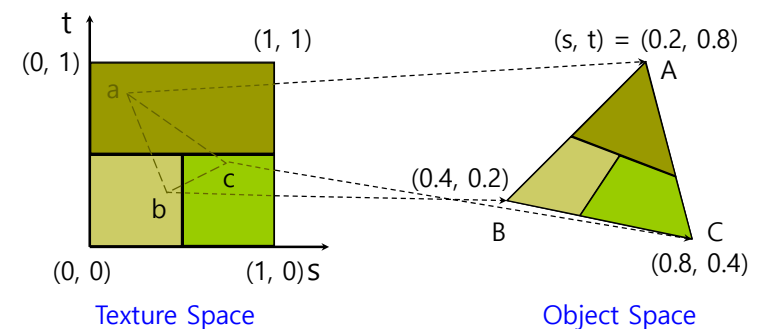
```
glTexImage2D(GL_TEXTURE_2D, 0, RGB, imageWidth, imageHeight,  
0, GL_RGB, GL_UNSIGNED_BYTE, imageData);
```

## Converting A Texture Image

- OpenGL은 텍스처 이미지 크기가 2의 승수 (power of 2)로 지정되어야 함
  - 64x64, 64x128, 512x512, ...
- 만약 이미지의 크기가 2의 승수가 아니면, 프로그램상에서 gluScaleImage를 사용하여 크기조절을 해야 함
  - gluScaleImage( format, w\_in, h\_in, type\_in, \*data\_in, w\_out, h\_out, type\_out, \*data\_out );
    - data\_in - 원본 이미지
    - data\_out - 이미지 크기가 조절된 최종 이미지
- 크기조절 시 이미지는 자동적으로 보간과 필터링됨

## Mapping a Texture

- 매개변수형 텍스처 좌표에 바탕을 두고 있음
- 각 정점별로 텍스처 좌표를 지정해야 함



## Mapping a Texture

- 각 정점별로 텍스처 좌표 지정

```
// Square Vertices Positions
```

```
squareVertices.push_back(glm::vec3(-0.75f, -0.75f, 0.0f));
squareVertices.push_back(glm::vec3(0.75f, -0.75f, 0.0f));
squareVertices.push_back(glm::vec3(0.75f, 0.75f, 0.0f));
squareVertices.push_back(glm::vec3(-0.75f, 0.75f, 0.0f));
```

```
// Square Vertices Texture Coordinates
```

```
squareTextureCoords.push_back(glm::vec2(0.0f, 0.0f));
squareTextureCoords.push_back(glm::vec2(1.0f, 0.0f));
squareTextureCoords.push_back(glm::vec2(1.0f, 1.0f));
squareTextureCoords.push_back(glm::vec2(0.0f, 1.0f));
```

- Note: 코드의 효율을 향상시키기 위하여 정점 배열을 사용

## Interpolation

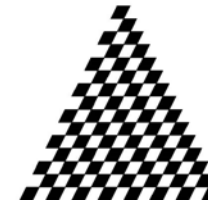
- OpenGL은 다각형 내부점에 대한 텍스처 좌표를 계산하기 위해 보간(interpolation)을 이용함

good selection  
of tex coordinates

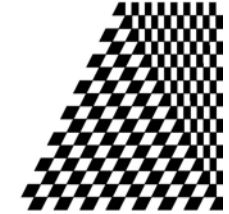


Checkerboard 텍스처를 삼각형에 맵핑하기

poor selection  
of tex coordinates



texture stretched  
over trapezoid  
showing effects of  
bilinear interpolation



Checkerboard 텍스처를 사다리꼴에 맵핑하기

## Texture Parameters

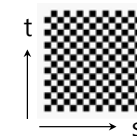
- OpenGL에서 텍스처가 어떻게 적용될지 결정해주는 다양한 인자(parameters)를 제공하고 있음
  - 랩핑 (Wrapping) - s, t가 (0, 1)의 범위를 넘었을 때 어떻게 s, t 값을 어떻게 정할 것인지 결정해주는 인자. CLAMP, REPEAT
  - 필터 (Filter modes) - 포인트 샘플링 대신 영역 평균법을 사용하도록 함.
  - mip맵 (Mipmapping) - 텍스처를 여러 개의 해상도를 사용할 수 있도록 함.
  - 환경인자 (Environment parameters) - 텍스처 맵핑이 음영 (shading)과 어떻게 상호작용할 지 결정해줌

## Wrapping Mode

- Clamp: (0, 1) 범위 내로 값을 강제 조정
  - 만약 s, t가 1보다 크면, 1을 사용
  - 만약 s, t가 0보다 작으면, 0을 사용
- Repeat: (0, 1) 범위를 넘어선 값에 대해 텍스처 반복
  - s, t % 1을 사용

```
glTexParameteri(GL_TEXTURE_2D,  
GL_TEXTURE_WRAP_S, GL_CLAMP)
```

```
glTexParameteri(GL_TEXTURE_2D,  
GL_TEXTURE_WRAP_T, GL_REPEAT)
```



texture



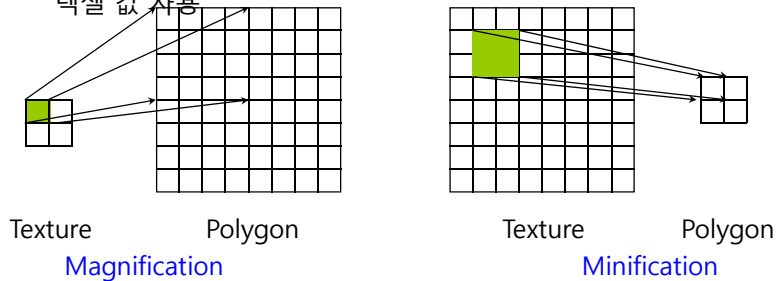
GL\_REPEAT  
wrapping



GL\_CLAMP  
wrapping

## Magnification and Minification

- 확대 (Magnification) – texel이 한 pixel보다 클 때
- 축소 (Minification) – texel이 한 pixel보다 작을 때
- 포인트 샘플링 (Point sampling)
  - 선형 필터링 (linear filtering) – 포인트 샘플링에 의해 결정된 텍셀의 이웃을 포함한 텍셀 그룹의 가중평균을 사용
  - 최근점 (nearest) – 선형보간에 의해 계산된 값에 가장 가까운 텍셀 값 사용



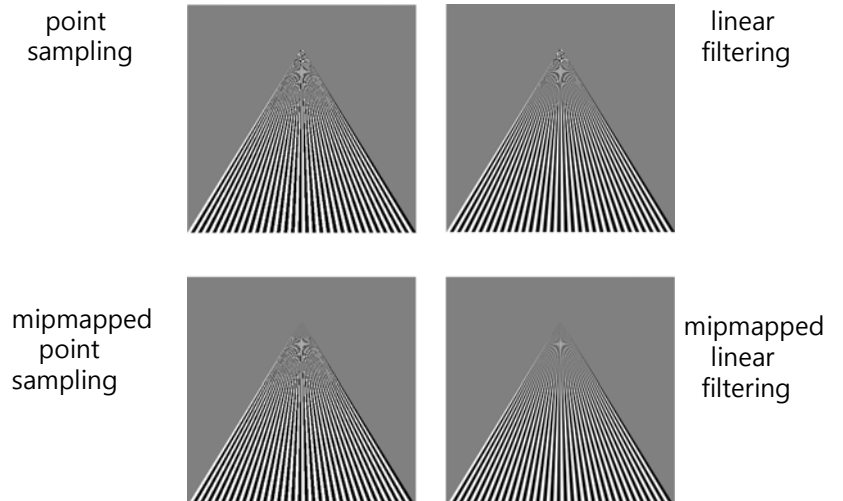
## Filter Modes

- 텍스처 필터 지정
  - `glTexParameteri( target, type, mode )`
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);`
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);`

## Mipmapped Textures

- mipmapping (Mipmapping)은 일련의 축소된 크기의 텍스처 배열 (prefiltered texture map)을 생성하는 것을 허용
- mipmapping은 작은 객체에 텍스처 맵핑을 할 시 보간문제를 줄여줌
- 텍스처 정의할 시 mipmapping 레벨을 지정할 수 있음
  - `glTexImage2D( GL_TEXTURE_*D, level, ... )`
- mipmapping을 통해서 자동으로 축소된 크기의 텍스처 생성
  - `glGenerateMipmap(GL_TEXTURE_2D)`
- OpenGL에서 최적의 mipmapping과 포인트 샘플링을 이용하기 위해 다음 옵션 사용
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST_MIPMAP_NEAREST)`
  - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR)`

## Aliasing Example



## Texture Environment

- `glTexEnv(fi){v}()`를 사용하여 텍스처와 음영간의 상호작용을 지정함
  - `glTexEnvf(GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, mode);`
    - `GL_TEXTURE_ENV_MODE`의 모드:
      - `GL_MODULATE`: 텍스처의 색 성분과 음영에서 주어지는 색성분을 곱함으로써 텍스처 맵핑없이 할당될 음영을 변조가능
      - `GL_DECAL`: 텍스처의 색이 객체의 색을 완전히 결정
      - `GL_BLEND`: 환경색과 합성함
      - `GL_REPLACE`: 텍스처 색만 사용함
    - `GL_TEXTURE_ENV, GL_TEXTURE_ENV_MODE, GL_MODULATE`
- 합성할 색은 `GL_TEXTURE_ENV_COLOR`으로 지정함

## Generating Texture Coordinates

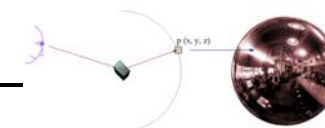
- `glTexGen{ifd}{v}()`를 사용하여 텍스처 좌표를 자동적으로 생성할 수 있음
    - 평면 (plane)을 지정해야 함 - 평면으로부터의 거리에 바탕을 둔 텍스처 좌표를 생성
    - 모드:
      - `GL_OBJECT_LINEAR`: 탁자 상단에 나무결을 입힐 때
      - `GL_EYE_LINEAR`: 동적으로 움직이는 객체의 외형을 표현할 때
      - `GL_SPHERE_MAP`: 환경맵에서 사용함
- ```

GLfloat planes[] = {0.5, 0.0, 0.0, 0.5} // s=x/2 + 1/2
GLfloat planet[] = {0.0, 0.5, 0.0, 0.5} // t=y/2 + 1/2
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGenfv(GL_S, GL_OBJECT_LINEAR, planes);
glTexGenfv(GL_T, GL_OBJECT_LINEAR, planet);
  
```

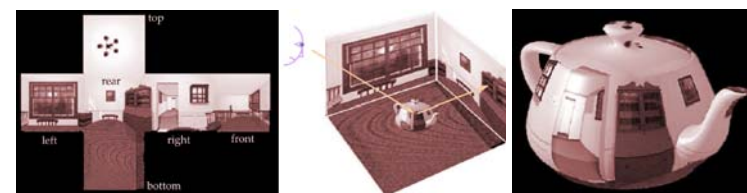
## Texture Objects

- 텍스처는 OpenGL state임
  - 다른 객체마다 다른 텍스처를 가진다면, OpenGL은 CPU 메모리에서 텍스처 메모리로 아주 큰 데이터를 움직여야 함.
- 최신 OpenGL은 텍스처 객체 (texture objects)를 가짐
  - 텍스처 객체당 1개 이미지
  - 텍스처 메모리 (Texture memory)는 여러 개 텍스처 객체를 가질 수 있음

## Environment Mapping



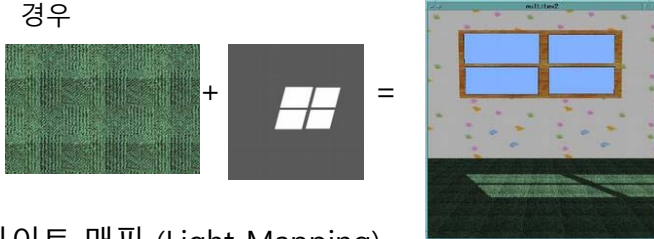
- 환경맵 (Environment Maps)
  - 물체를 둘러싸고 있는 주위 환경이 이 물체에 반사되는 결과를 원할 경우
  - 3차원 공간이 매우 큰 가상적 구(sphere) 또는 입방체(cube)에 의해 둘러싸였다고 가정하고 Second mapping에서 시점 반사벡터를 사용 - 시점 위치에 따라 서로 다른 모습을 보임.
  - Spherical environment mapping - 180도 와이드 앵글 렌즈로 찍은 환경 이미지로부터 spherical map 생성 후 자동 텍스처 좌표 생성을 사용



## Multitexturing

### □ 멀티 텍스처링 (Multitexturing)

- 하나 이상의 텍스처를 객체에 적용해서 렌더링 효과를 높이는 경우



### □ 라이트 매핑 (Light Mapping)

- 물체면의 밝기를 계산하는 대신 텍스처와 조명 결과를 혼합하여 결과적으로 영상을 직접 물체면에 입힘

