

# Texture Mapping

---

514780  
2019년 가을학기  
11/18/2019  
단국대학교 박경신

## OpenGL Texturing

---

- OpenGL에서 텍스처 맵핑(texture mapping)을 위한 3 단계
  - 텍스처 활성화
    - `glEnable(GL_TEXTURE_2D)`
  - 텍스처 맵핑방법 (랩핑, 필터 등) 정의
    - `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_LINEAR)`
    - `glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR)`
    - `glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, imageData)`
  - 텍스처 좌표 (texture coordinates) 지정
    - `// 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));`

## OpenGL Texturing

---

- Modern OpenGL 텍스처 맵핑(texture mapping)은 fragment shading에서 sampler로 구현됨
  - Sampler는 텍스처 색을 반환

```
//TextureFragmentShader.fs
in vec2 texCoordPass; // texture coordinate from rasterizer
out vec4 Color; // output color
uniform sampler2D gTextureSampler; //texture object from application
void main() {
    Color = texture2D( gTextureSampler, texCoordPass );
}
//TransformVertexShader.vs
in vec3 vPosition;// vertex position (in model space)
in vec2 vTexCoord;// texture coordinate (in model space)
out vec2 texCoordPass; // Output data will be interpolated for each fragment.
uniform mat4 gMVP; // Values that stay constant for the whole mesh.
void main() {
    gl_Position = gMVP * vec4(vPosition,1); // Output position of the vertex
    texCoordPass = vTexCoord;
}
```

## OpenGL Texture Names

---

- 텍스처의 이름(name) 지정하기 (textureID)
  - `GLuint textureID;`
  - `glGenTextures(1, &textureID);`
  - `glBindTexture(GL_TEXTURE_2D, textureID);`
  - `glTexImage2D(...);`
  - `glTexParameterf(.....);`
  - `glBindTexture(GL_TEXTURE_2D, 0);`
  - ...
- 텍스처 이름으로 텍스처 불러오기
  - `glActiveTexture(GL_TEXTURE0);`
  - `glBindTexture(GL_TEXTURE_2D, textureID);`

## glTexImage2D

---

- `glTexImage2D(GLenum target, GLint level, GLint internalFormat, GLsizei width, GLsizei height, GLint border, GLenum format, GLenum type, const GLvoid *pixels);`
  - `target`: `GL_TEXTURE_2D`
  - `level`: 텍스처 맵의 다양한 해상도를 지원하기 위해 설정.
    - 1개의 해상도를 지정하려면 1로 설정.
    - mip맵(mipmapping)에 사용
    - 각 텍스처를 위한 다수의 크기를 가지고 있는 이미지를 사용하지 않는다면 0으로 지정
  - `internalFormat`: 일반적으로 `format`과 같음. RGB라면 3, RGBA라면 4로 설정
  - `width, height`: 텍스처 이미지의 너비와 높이는 2의 자승으로 되어야 함 (즉, 2, 4, 8, 16, 32, 64, 128, 256, 512, etc)
  - `border`: 텍스처의 경계선 너비를 지정. 보통 0이고 이미지 데이터가 `border`를 가지고 있으면 1로 지정.
  - `type`: 텍스처 이미지 데이터의 형식을 설정.

## glTexImage1D

---

- `glTexImage1D(GLenum target, GLint level, GLint internalFormat, GLsizei width, GLint border, GLenum format, GLenum type, const GLvoid *pixels);`
- `glTexImage2D()` 함수는 2차원 텍스처 이미지를 정의하고 `glTexImage1D()` 함수는 1차원 텍스처 이미지를 정의한다. `glTexImage1D()` 함수와 `glTexImage2D()` 함수의 사용 방법과 인자의 의미는 거의 동일하며 `glTexImage2D()` 함수에만 이미지 텍스처의 `height`(높이) 인자가 추가된다.

## glTexSubImage2D

---

- 텍스처 크기가 2의 승수(e.g., 64x64, 128x256, ..)가 아닌 경우 텍스처 이미지의 일부분만 읽어 들일 때 사용함
- `glTexSubImage2D(GLenum target, GLint level, GLint xoffset, GLint yoffset, GLsizei width, GLsizei height, GLenum format, GLenum type, const GLvoid *pixels);`

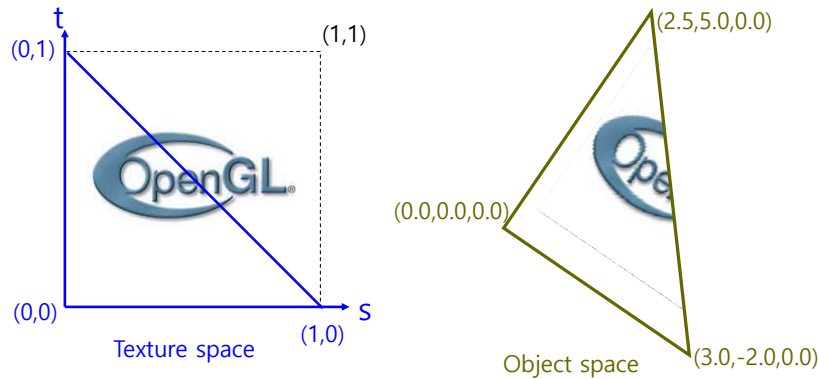
## OpenGL Texture Coordinates

---

- 텍스처 맵핑이 사용되려면 객체에 텍스처 좌표를 정의해야 한다.
- GLUT teapot은 텍스처 좌표를 포함하고 있다.
- GLU quadrics도 텍스처 좌표를 옵션으로 정의할 수 있다.
  - `gluQuadricTexture(quadric, GL_TRUE)` 를 사용하여 텍스처 맵핑 활성화

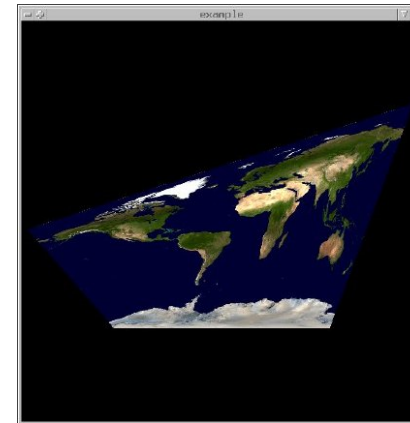
## OpenGL Texture Coordinates

- OpenGL에서 텍스처 좌표는 텍스처 이미지의 각 방향 (S, T)의 0부터 1의 영역으로 형성됨.
- OpenGL에서 각 정점별로 텍스처 좌표를 지정해야 함.
- 정점의 텍스처 좌표가 객체의 표면에 보간되어 나타남.



## OpenGL Texture Coordinates

- 텍스처는 다각형에 고르게 입힐 필요는 없음.
- 기하학적 모델이나 텍스처 좌표 사용에 따라서, 이미지가 왜곡되게 나타날 수도 있음.



Texture Coordinates

(0,0)  
(1,0)  
(1,1)  
(0,1)

Vertex Positions

(-1.0, -1.0, 0.0)  
(1.0, -1.0, 0.0)  
(rMousePosX, rMousePosY, 0.0)  
(lMousePosX, lMousePosY, 0.0)

## OpenGL Texture Filtering

- OpenGL에서 텍스처 좌표가 [0, 1]영역을 벗어날 경우에 texture wrapping 방법으로 정의함:
  - GL\_CLAMP, GL\_CLAMP\_TO\_EDGE, GL\_REPEAT, GL\_MIRRORED\_REPEAT
- 사각형에 4개의 텍스처좌표를 (0,0), (0,3.5), (3.5,0), (3.5,3.5) 로 정의한 예



Repeat

Clamp

Repeat & Clamp

## OpenGL Texture Filtering

- Mipmap은 이전 mipmap 너비와 폭의 절반 크기임.
  - 텍스처가 작아질 수록, 보다 많은 텍셀이 한 픽셀에 적용되어야하므로 GL\_NEAREST나 GL\_LINEAR 필터가 정확한 계산결과를 만들지 않을 수 있음. 따라서 객체가 움직일 때 텍스처가 flickering하게 나타날 수 있음.
  - 미맵은 이런 flickering문제를 줄여줄 수 있음. 그러나 일반적으로 희미하게 보임.



GL\_LINEAR



GL\_LINEAR\_MIPMAP\_LINEAR

## OpenGL Texture Filtering

- 텍스처 맵핑을 위한 필터링 방법
  - 최근점 필터 (nearest neighbor filter)
    - GL\_NEAREST
  - 이선형 필터 (bilinear interpolation filter)
    - GL\_LINEAR
  - 삼선형 필터 (trilinear interpolation filter) – mipmap filter
    - GL\_LINEAR\_MIPMAP\_LINEAR
  - 혼합 필터 (hybrid filter)
    - GL\_NEAREST\_MIPMAP\_LINEAR
    - GL\_LINEAR\_MIPMAP\_NEAREST
    - GL\_NEAREST\_MIPMAP\_NEAREST

## OpenGL Texture Filtering

- `glTexParameter{if}v(GLenum target, GLenum pname, TYPE *param);`
  - GL\_TEXTURE\_WRAP\_S, GL\_TEXTURE\_WRAP\_T
    - GL\_CLAMP, GL\_CLAMP\_TO\_EDGE, GL\_REPEAT, GL\_MIRRORED\_REPEAT
  - GL\_TEXTURE\_MAG\_FILTER
    - GL\_NEAREST, GL\_LINEAR
  - GL\_TEXTURE\_MIN\_FILTER
    - GL\_NEAREST, GL\_LINEAR (Mipmap을 사용하지 않는 경우)
    - GL\_NEAREST\_MIPMAP\_NEAREST, GL\_NEAREST\_MIPMAP\_LINEAR, GL\_LINEAR\_MIPMAP\_NEAREST, GL\_NEAREST\_MIPMAP\_NEAREST
  - GL\_TEXTURE\_BORDER\_COLOR
    - [0.0, 1.0]영역의 값
  - GL\_TEXTURE\_PRIORITY
    - 0 또는 1

## Texture Environment Parameters

- `glTexEnv{f|i}[v](.)`를 사용하여 텍스처의 색 ( $C_t, A_t$ ) 과 현재 처리하는 프레임버퍼의 픽셀색 ( $C_f, A_f$ )을 어떻게 혼합할 지 설정함
  - GL\_TEXTURE\_ENV\_MODE의 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\_BLEND의 합성색은 GL\_TEXTURE\_ENV\_COLOR으로 지정함
 

```
GLfloat blendcolor[] = {0.0, 1.0, 0.0, 0.5};
glTexEnvfv(GL_TEXTURE_ENV, GL_TEXTURE_ENV_COLOR, blendcolor);
```

## Texture Environment Parameters

int. format	GL_REPLACE	GL_MODULATE
GL_ALPHA	$C = C_f, A = A_t$	$C = C_f, A = A_f A_t$
GL_LUMINANCE	$C = I_t, A = A_f$	$C = C_f I_t, A = A_f$
GL_LUMINANCE_ALPHA	$C = I_t, A = A_t$	$C = C_f I_t, A = A_f A_t$
GL_INTENSITY	$C = I_t, A = I_t$	$C = C_f I_t, A = A_f I_t$
GL_RGB	$C = C_t, A = A_f$	$C = C_f C_t, A = A_f$
GL_RGBA	$C = C_t, A = A_t$	$C = C_f C_t, A = A_f A_t$
int. format	GL_DECAL	GL_BLEND
GL_ALPHA	undefined	$C = C_f, A = A_f A_t$
GL_LUMINANCE	undefined	$C = C_f(1 - I_t) + C_c I_t, A = A_f$
GL_LUMINANCE_ALPHA	undefined	$C = C_f(1 - I_t) + C_c I_t, A = A_f A_t$
GL_INTENSITY	undefined	$C = C_f(1 - I_t) + C_c I_t, A = A_f(1 - I_t) + A_c I_t$
GL_RGB	$C = C_t, A = A_f$	$C = C_f(1 - C_t) + C_c C_t, A = A_f$
GL_RGBA	$C = C_f(1 - A_t) + C_t A_t, A = A_f$	$C = C_f(1 - C_t) + C_c C_t, A = A_f A_t$

## OpenGL Texture Movies

- 텍스처 이미지 sequence를 이용하여 flipbook 애니메이션 제작
  - initTexture()함수에서 전체 텍스처 이미지를 읽어 들임
  - idle() 함수에서 currentTextureID를 update함
  - drawTexture()함수에서는 동일한 정점좌표와 텍스처 좌표에 glBindTexture(GL\_TEXTURE\_2D, currentTextureID);를 사용하여 프레임당 한 텍스처를 binding함 - 애니메이션 효과를 줌



## OpenGL Texture Coordinate Generation

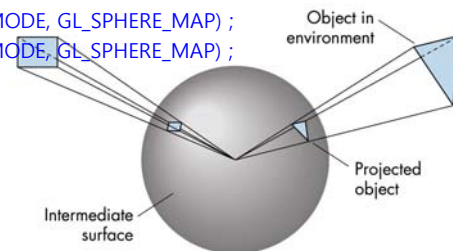
- OpenGL에서는 텍스처 좌표를 자동적으로 생성할 수 있음
  - S, T 방향으로 텍스처 좌표 자동생성을 활성화해야 함
    - glEnable(GL\_TEXTURE\_GEN\_S), glEnable(GL\_TEXTURE\_GEN\_T)
  - GL\_TEXTURE\_GEN\_MODE 모드:
    - GL\_OBJECT\_LINEAR, GL\_EYE\_LINEAR, GL\_SPHERE\_MAP
  - 평면 (plane)을 지정해야 함 - 평면으로부터의 거리에 바탕을 둔 텍스처 좌표를 생성
    - glTexGenfv(GL\_S, GL\_OBJECT\_PLANE, planeCoefficients)

```
planeCoefficients = { 1, 0, 0, 0 };
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_OBJECT_LINEAR);
glTexGenfv(GL_S, GL_OBJECT_PLANE, planeCoefficients);
glEnable(GL_TEXTURE_GEN_S);
glBegin(GL_QUADS);
glVertex3f(-3.25, -1, 0); glVertex3f(-1.25, -1, 0);
glVertex3f(-1.25, 1, 0); glVertex3f(-3.25, 1, 0);
glEnd();
```

## OpenGL Sphere Mapping

- OpenGL에서는 구형 맵핑 (sphere mapping) 지원
  - 구형 맵핑 텍스처 좌표는 view 벡터가 구 표면의 법선 벡터에 반사된 reflection 벡터로 계산됨.
  - 반사벡터를 2차원 텍스처 좌표로 맵핑하는 것이 간단하고 하드웨어, 소프트웨어로도 구현이 가능.
  - 그러나 원형 이미지를 구하는 것이 어려움 (360도의 주변환경을 담은 이미지여야 함). 아주 넓은 광학 렌즈에 의한 원근 투영을 구하거나, 입방체 투영을 이용하여 근사한 값을 얻음.

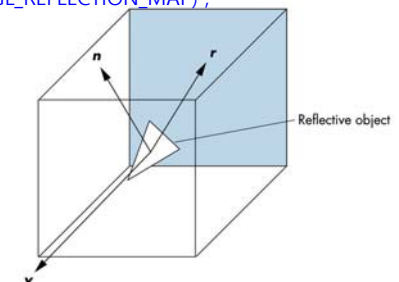
```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_SPHERE_MAP);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
```



## OpenGL Box Mapping

- OpenGL에서는 입방체 맵핑 (box mapping) 지원
  - 입방체맵은 반사 맵핑(reflection mapping)의 하나임
  - 그러나 입방체맵은 3차원 텍스처 좌표를 사용해야 함
  - 반사 텍스처는 환경을 둘러 싸고 있는 입방체의 6면 2차원 텍스처

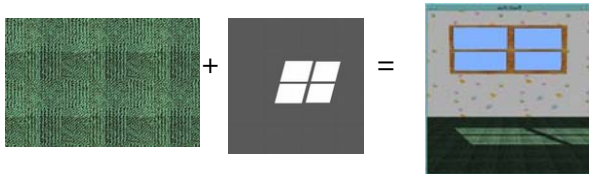
```
glTexGeni(GL_S, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
glTexGeni(GL_T, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
glTexGeni(GL_R, GL_TEXTURE_GEN_MODE, GL_REFLECTION_MAP);
glEnable(GL_TEXTURE_GEN_S);
glEnable(GL_TEXTURE_GEN_T);
glEnable(GL_TEXTURE_GEN_R);
glEnable(GL_TEXTURE_CUBE_MAP);
```



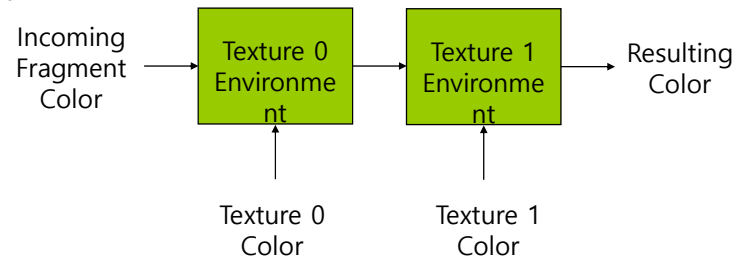
## Multitexturing

### 멀티 텍스처 (Multitexturing)

- 하나 이상의 텍스처를 객체에 적용해서 렌더링 효과를 높이는 경우

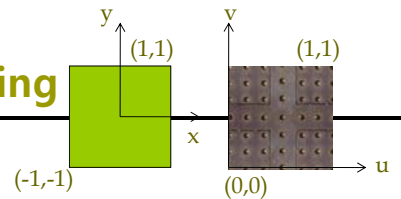


### OpenGL 1.2.1 revision (**ARB\_multitexture** extension)



## Single-Pass Multitexturing

```
void SetMultitexturSquareData() { // 중간생략..
glGenBuffers(4, &vbo[0]);
glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
glBufferData(GL_ARRAY_BUFFER, 4*sizeof(glm::vec3), &quadVertices[0], GL_STATIC_DRAW);
glVertexAttribPointer(0, 3, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(0);
glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
glBufferData(GL_ARRAY_BUFFER, 4*sizeof(glm::vec3), &quadNormals[0], GL_STATIC_DRAW);
glVertexAttribPointer(1, 3, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(1);
glBindBuffer(GL_ARRAY_BUFFER, vbo[2]);
glBufferData(GL_ARRAY_BUFFER, 4*sizeof(glm::vec2), &quadTextureCoords[0], GL_STATIC_DRAW);
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(2);
glBindBuffer(GL_ARRAY_BUFFER, vbo[3]);
glBufferData(GL_ARRAY_BUFFER, 4*sizeof(glm::vec2), &quadTextureCoords[0], GL_STATIC_DRAW);
glVertexAttribPointer(3, 2, GL_FLOAT, GL_FALSE, 0, 0);
glEnableVertexAttribArray(3);
}
```



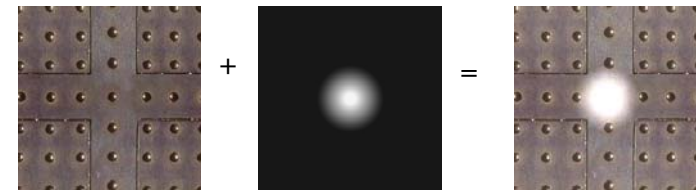
## Multitexturing

### Single-Pass vs. Multi-Pass Multitexturing

- 싱글 패스 다중 텍스처링은 하나의 렌더링 패스 안에서 텍스처를 여러 개 입히는 것
- 다중 패스 다중 텍스처링은 블렌딩으로 장면이나 다각형 자체를 여러 번 렌더링하는 것

### 라이트 매핑 (Light Mapping)

- 물체면의 밝기를 계산하는 대신 텍스처와 조명 결과를 혼합하여 결과적으로 영상을 직접 물체 면에 입힘 (e.g. Quake 등 게임)



## Single-Pass Multitexturing

### Bind and enable two 2D multitextures to draw a quad

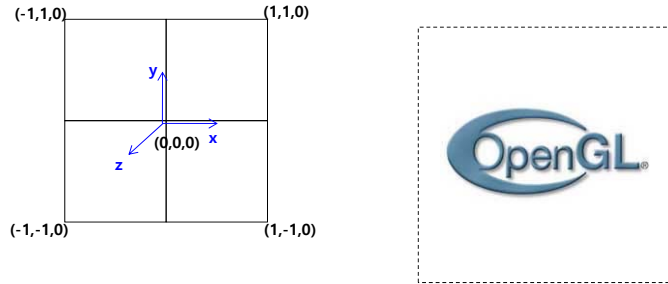
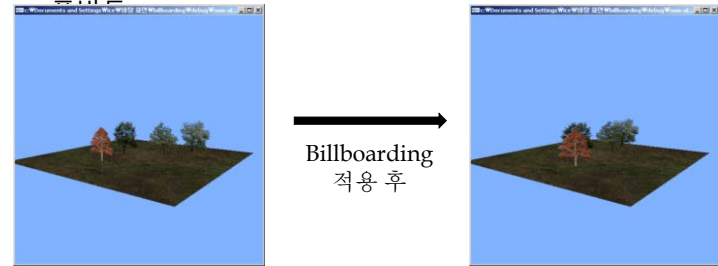
```
// stage 0 activate
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture0);
// stage 1 activate
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, texture1);

drawSquare();
// texture disabled
glBindTexture(GL_TEXTURE_2D, 0);
```

## Billboarding

### ▣ 빌보드 기법

- 사각형의 정면이 항상 카메라의 정면을 향하여 바라보도록 만드는 것으로, 결과적으로 카메라가 어느 방향에서 바라보아도 사각형은 항상 같은 면을 보여주게 됨
- 사용 예로써, 넓은 지면 위에 나무를 나타내고자 할 때 나무를 메쉬 모델링으로 표현하지 않고 나무 이미지를 빌보드로 구현함
- 알파 텍스처 기법과 애니메이션이 결합된 빌보드 기법은 고체 표면을 갖지 않는 여러 현상을 표현하는데 사용 : 연기, 불, 안개,



## Billboarding

### ▣ 빌보드 원리

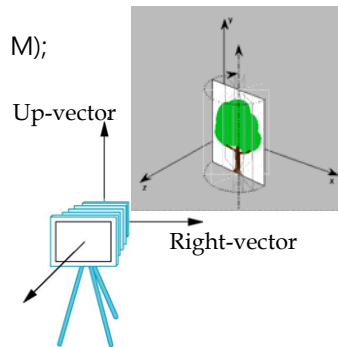
- 구현의 핵심은, 항상 시점을 바라보도록 Modelview 행렬을 이용하여 빌보드 사각형을 이루는 vertex를 조정해야 함
- Modelview 행렬에는 관찰자의 시점에 대한 수직벡터(up vector)와 우측벡터(right vector) 정보가 있음

GLfloat M[16];

glGetFloatv(GL\_MODELVIEW\_MATRIX, M);

Right-vector Up-vector Look-vector

$m_0$	$m_4$	$m_8$	$m_{12}$
$m_1$	$m_5$	$m_9$	$m_{13}$
$m_2$	$m_6$	$m_{10}$	$m_{14}$
$m_3$	$m_7$	$m_{11}$	$m_{15}$



## Billboarding

### ▣ Axial Symmetry

- 빌보드 사각형이 vertical axis를 중심으로 회전 (rotate) 해야 함
- Modelview matrix M에서부터 카메라의 -yaw angle을 계산함

$\theta = \text{atan2f}(M[8], M[10]);$   
 Look.x    Look.z

- 빌보드 사각형의 Rotation matrix R은 arbitrary axis (일반적으로 up vector=(0, 1, 0))과 angle (카메라 yaw angle의 반대)로 계산함

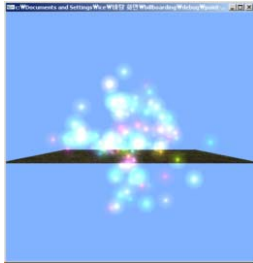
$R = I \cos \theta + \text{Symmetric} (1 - \cos \theta) + \text{Skew} \sin \theta$

$$= \begin{bmatrix} a_x^2 + \cos \theta (1 - a_x^2) & a_x a_y (1 - \cos \theta) - a_z \sin \theta & a_x a_z (1 - \cos \theta) + a_y \sin \theta \\ a_x a_y (1 - \cos \theta) + a_z \sin \theta & a_y^2 + \cos \theta (1 - a_y^2) & a_y a_z (1 - \cos \theta) - a_x \sin \theta \\ a_x a_z (1 - \cos \theta) - a_y \sin \theta & a_y a_z (1 - \cos \theta) + a_x \sin \theta & a_z^2 + \cos \theta (1 - a_z^2) \end{bmatrix}$$

## Point Sprites

---

- Point Sprites이란 하드웨어 기반의 빌보드 기법으로, 사용 예로써 파티클 시스템 (Particle System)에서 다수의 입자 (Particle)을 텍스처 사각형 메쉬 (Quad) 대신 점 (Point)를 사용하여 지정함
- OpenGL extension (**GL\_ARB\_point\_parameters** & **GL\_ARB\_point\_sprite**)



## Reference

---

- OpenGL Billboarding Tutorial  
<http://www.lighthouse3d.com/opengl/billboarding>
- SIGGRAPH'97 Advanced OpenGL Programs  
<http://www.opengl.org/resources/code/samples/advanced/advanced97/programs/programs.html>
- OpenGL Point Sprites  
<http://www.informit.com/articles/article.aspx?p=770639&seqNum=7>
- Particle Systems <http://www.gamedev.net/>