Getting Started (3)

GPU Programming

2022학년도
2학기

ⓒ nvidia

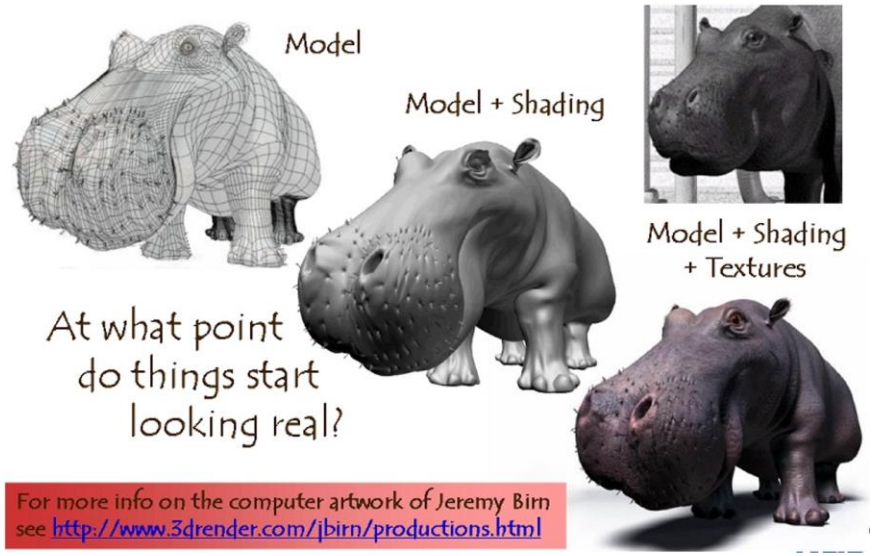
Textures

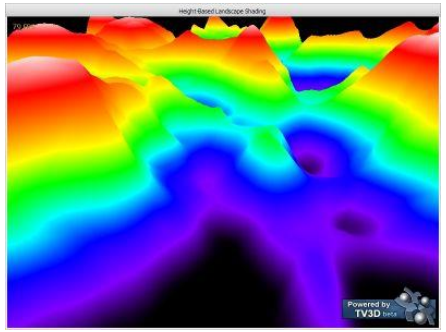
Textures

- Vertex color 사용은 높은 오버헤드 발생 가능
  - Vertex별로 color attribute를 할당하여 질감을 표현해야 하기 때문

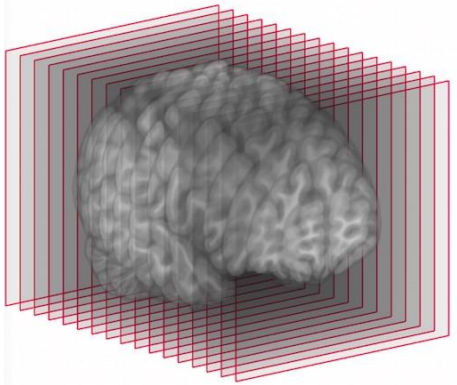
- 대안: 텍스처(texture)의 사용
  - 객체에 입히는 벽지 또는 포장지 같은 개념
  - 적은 vertex로도 디테일한 질감을 표현 가능

- 형식
  - 일반적으로 2D를 주로 사용
  - 3D 텍스처도 볼륨 렌더링 등에 사용 (구름, 불 등)
  - 1D 텍스처도 등고선 등에 간혹 사용
  - 텍스처는 보통 압축된 형태로 사용됨
    (이 내용은 나중에 다시 다룰 예정)





Landscape Height-Based Coloring – The Instruction Limit



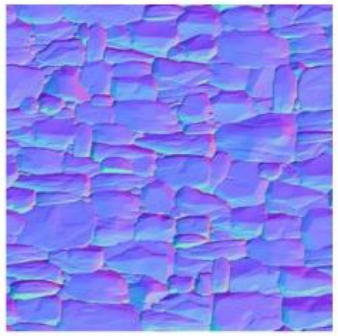
neurolabusc/vx: Simple Volume Renderer (github.com)

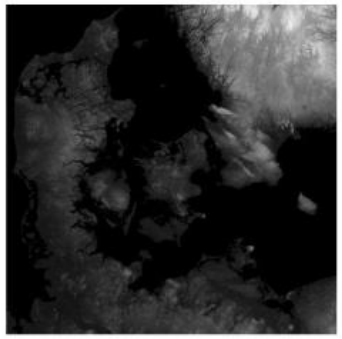
Textures

- 텍스처는 다양한 목적을 위해 여러가지 정보를 담을 수 있음
  - 미리 만들어진 텍스처를 입히기도 하지만, 경우에 따라 GPU에서 매 프레임 텍스처가 만들어지기도 함
  - 노멀 매핑, 환경 매핑, 섀도우 매핑 등의 기법은 추후 다룹니다.



일반적인
질감 표현을 위한
텍스처들


노멀 맵


높이 맵

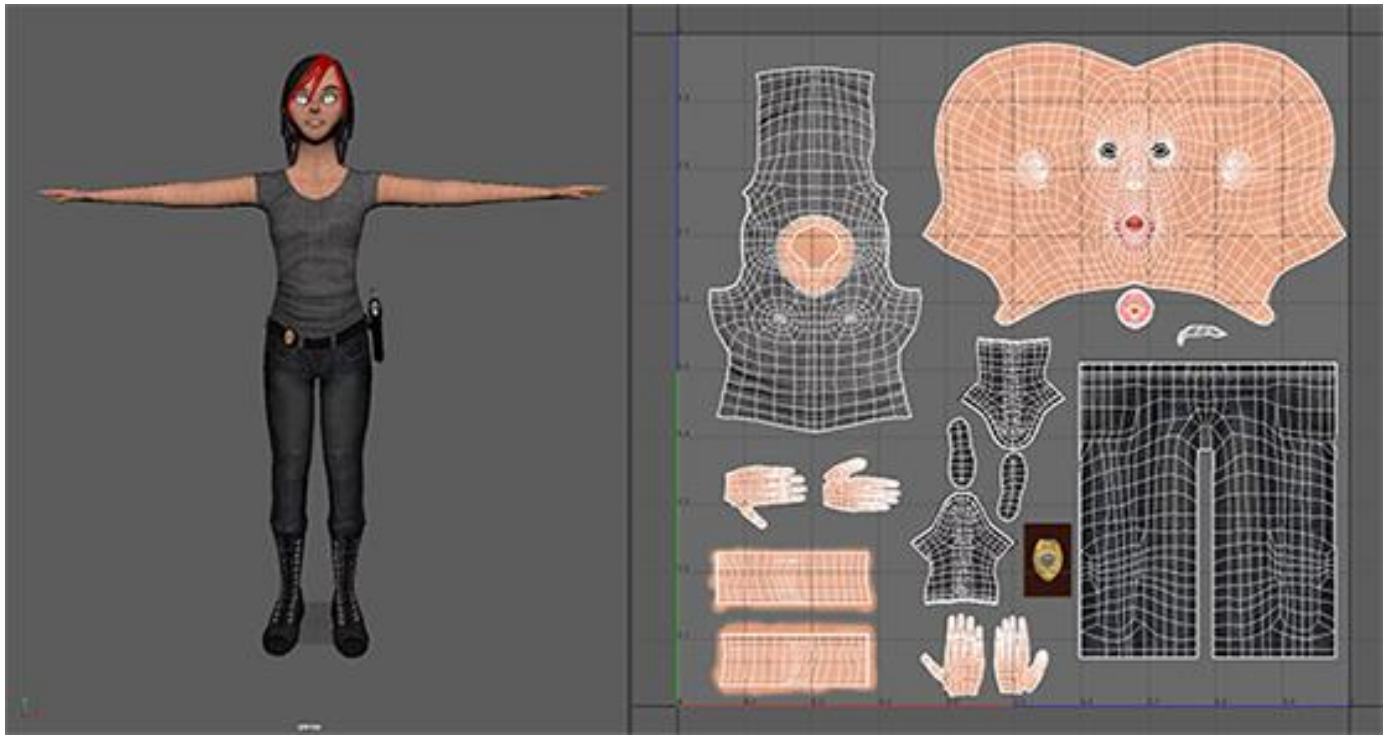

환경 맵


그림자 맵

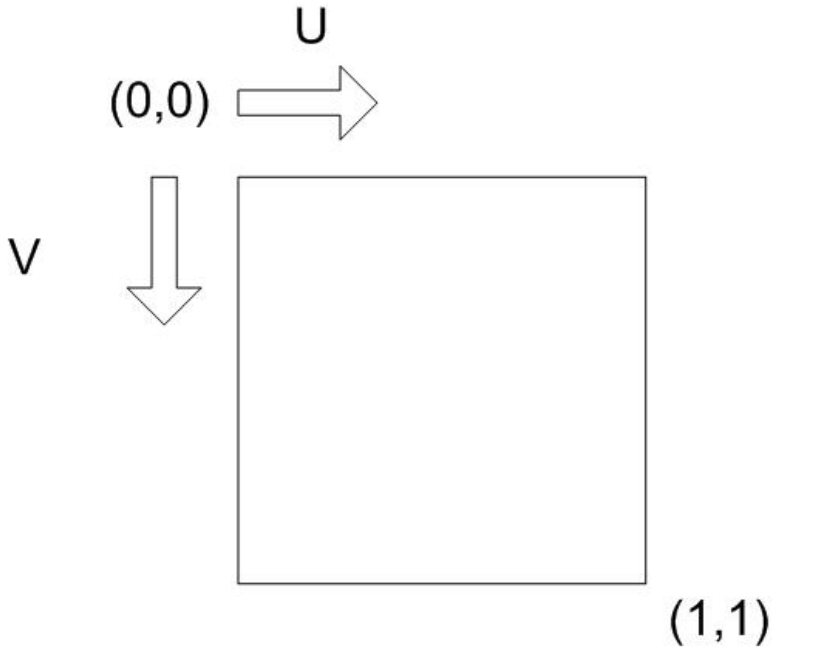

라이트 맵

Texture Mapping

- 텍스처 매핑
  - 텍스처의 좌표값에 따라 텍셀(texel, 텍스처의 픽셀)을 모델의 픽셀에 매핑하는 작업
  - 가로, 세로 축을 보통 u,v라고 보통 칭하나, OpenGL에서는 s,t를 사용



Maya Help | UVs | Autodesk



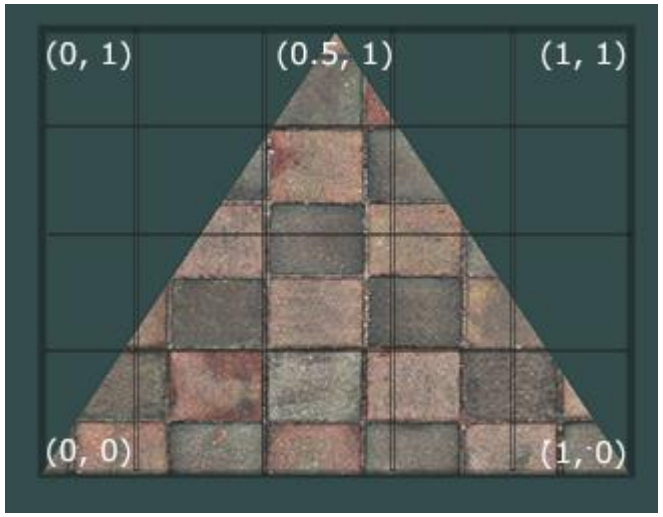
질감 좌표(Direct3D 9) - Win32 apps | Microsoft Docs

Texture Mapping

- Vertex 별로 텍스처 좌표(texture coordinate)를 가짐
  - 샘플링(sampling)할 영역, 즉 컬러를 가져올 텍스처의 영역을 지정
  - 삼각형 내부 fragment는 보간된 텍스처 좌표에 따라 매핑됨

- 오른쪽 삼각형의 벽돌 텍스처 좌표 지정 예

```
float texCoords[] = {
    0.0f, 0.0f,  // lower-left corner
    1.0f, 0.0f,  // lower-right corner
    0.5f, 1.0f   // top-center corner
};
```



Texture Wrapping

- 텍스처 좌표의 범위는 일반적으로 (0,0) ~ (1,1)임
- 좌표 범위 밖의 값을 지정할 경우
  - GL_REPEAT (반복)
  - GL_MIRRORED_REPEAT (뒤집어 반복)
  - GL_CLAMP_TO_EDGE
    (EDGE의 값으로 고정)
  - GL_CLAMP_TO_BORDER
    (미리 지정한 테두리 색상으로 칠함)

- 축 별로 다른 설정 가능

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_MIRRORED_REPEAT);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_MIRRORED_REPEAT);
```

```
float borderColor[] = { 1.0f, 1.0f, 0.0f, 1.0f };
glTexParameterfv(GL_TEXTURE_2D, GL_TEXTURE_BORDER_COLOR, borderColor);
```

Texture Filtering

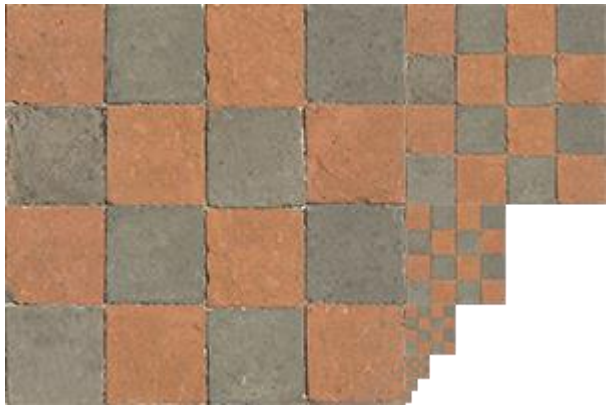
- 텍셀을 텍스처 좌표에 어떻게 매핑할 것인가?
- Nearest neighbor or point filtering (GL_NEAREST)

- (Bi)linear filtering (GL_LINEAR)
  - 텍스처 좌표와 이웃한 |텍셀 4개의 값을 보간 (interpolation)

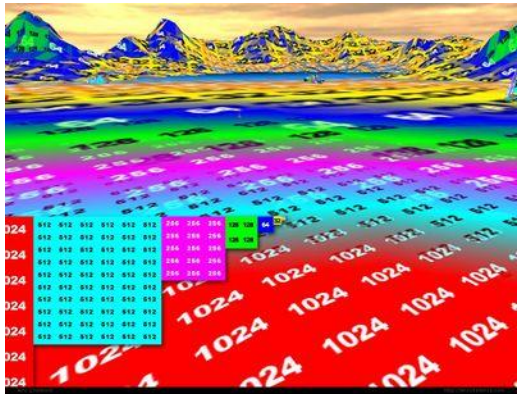
- 텍스처를 확대(magnify)하여 매핑시
  - 텍셀의 크기 > 픽셀의 크기
  - GL_NEAREST의 경우 계단 현상이, GL_LINEAR의 경우 흐려짐 (blurring) 발생

Mipmaps

- 작은 물체에 고해상도의 텍스처를 축소(minify)하여 매핑해야 한다면? (텍셀의 크기 < 픽셀의 크기)
  - 픽셀 하나에 여러 개의 텍셀 중 일부가 선택
  - 메모리 액세스 부하가 늘어남
  - 시점이 바뀔 때마다 다른 텍셀이 선택될 수 있음 → 자글자글함 (aliasing)



- 해결책 – Mipmap
  - 텍스처를 크기별로 이미지 피라미드로 구성 (가로 및 세로 길이가 ½, ¼, 1/8, …)
  - 가능한 한 텍셀의 크기와 픽셀의 크기를 유사하게 매칭
  - 시점에 따라 적합한 밉맵 레벨 선택
    가장 가까우면 0 레벨, 멀수록 높은 레벨 (=작은 크기)
  - glGenerateMipmaps() 함수로 밉맵을 생성하거나,
    각 밉맵 레벨을 미리 만들어 container format(.dds, .ktx)에 저장할 수도 있음



Mip Mapping - polycount

Mipmaps

- 밉맵 레벨 필터링 방법
  - 가까운 레벨 선택하거나(GL_NEAREST), 두 레벨을 보간 (GL_LINEAR)
  - 축소 필터에 대해, 텍스처 샘플링 및 밉맵 레벨 각각에 대해 필터링 옵션 지정 가능 → 총 4가지

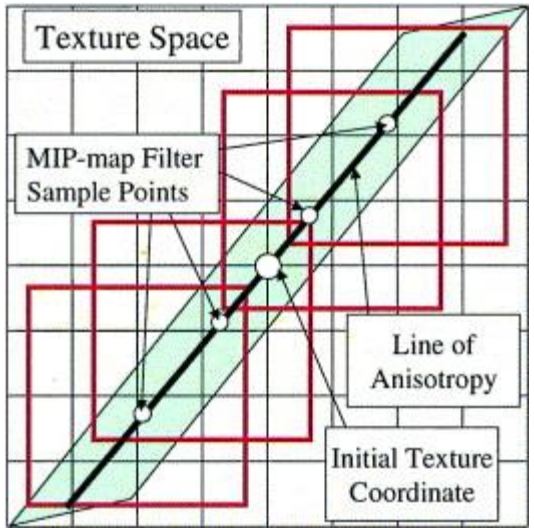
GL_NEAREST_MIPMAP_NEAREST	GL_LINEAR_MIPMAP_NEAREST
GL_NEAREST_MIPMAP_LINEAR	GL_LINEAR_MIPMAP_LINEAR

```
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR_MIPMAP_LINEAR);
glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
```

  - GL_LINEAR_MIP_LINEAR을 trilinear filtering이라고 하기도 함

Anisotropic Filtering

- 지금까지 설명한 필터링은 기본적으로 등방성 (isotropic) 필터링임
  - 가로, 세로 양방향으로 똑같이 필터링.
    즉, 텍셀이 화면에 정사각형으로 보인다고 가정
  - 비스듬한 시점에서는 화질 저하 발생 가능



MIP-map level selection for texture mapping | IEEE Journals & Magazine | IEEE Xplore

- 비등방성 (anisotropic, 이방성) 필터링
  - 텍스처 공간에 시점에 따라 비스듬한 선(line of anisotropy)을 긋고,
    이 선에 따라 여러 개의 밉맵 필터 샘플(2, 4, 8, 16 등)을 취함
  - OpenGL 4.6부터 정식 지원,
    그 이전 버전이나 OpenGL ES에서는
    익스텐션으로 사용 가능 GL_EXT_texture_filter_anisotropic

- 각 필터링 설정에 따른 화질 비교
  - Mipmaps and Anisotropic Filtering - YouTube
  - What is Mipmapping in San Andreas? – YouTube
  - How to enable Anisotropic Filtering for any game on AMD & Nvidia GPU's - YouTube

Loading and creating textures

- 이미지 로드 라이브러리 stb_image.h를 사용하여 container.jpg 읽기

```
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"
```

```
int width, height, nrChannels;
unsigned char *data = stbi_load("container.jpg", &width, &height, &nrChannels, 0);
```

- 텍스처 객체 생성 후 바인딩. 필요시 텍스처 wrapping/filtering 옵션 설정

```
unsigned int texture;
glGenTextures(1, &texture);
```

```
glBindTexture(GL_TEXTURE_2D, texture);
```

- 읽어들인 이미지로 2D 텍스처 생성 및 밉맵 생성

```
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGB, GL_UNSIGNED_BYTE, data);
glGenerateMipmap(GL_TEXTURE_2D);
```

  - glTexImage2D() 함수의 파라미터: target, level, internal format (OpenGL에 저장되는 포맷), width, height, border (항상 0), format (원본 이미지의 포맷), type, data

- 이미지 메모리 반환

```
stbi_image_free(data);
```

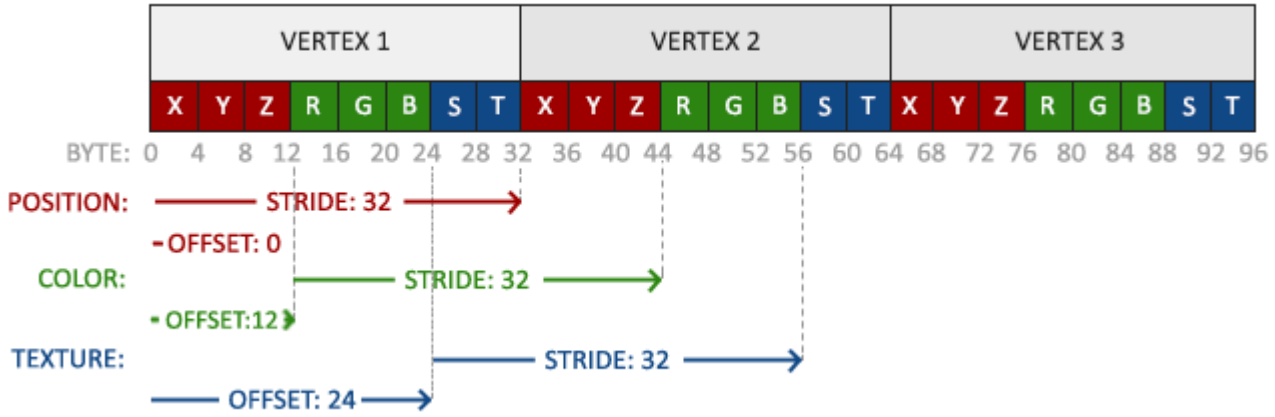
Applying textures

- 사각형의 vertex에 텍스처 좌표 설정

```
float vertices[] = {
    // positions          // colors           // texture coords
     0.5f,  0.5f, 0.0f,   1.0f, 0.0f, 0.0f,   1.0f, 1.0f,   // top right
     0.5f, -0.5f, 0.0f,   0.0f, 1.0f, 0.0f,   1.0f, 0.0f,   // bottom right
    -0.5f, -0.5f, 0.0f,   0.0f, 0.0f, 1.0f,   0.0f, 0.0f,   // bottom left
    -0.5f,  0.5f, 0.0f,   1.0f, 1.0f, 0.0f,   0.0f, 1.0f    // top left
};
```

- OpenGL에게 텍스처 좌표와 관련된 새로운 vertex format을 알려 줌

```
glVertexAttribPointer(2, 2, GL_FLOAT, GL_FALSE, 8 * sizeof(float), (void*)(6 * sizeof(float)));
glEnableVertexAttribArray(2);
```



Applying textures

- VS/FS 수정

```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec3 aColor;
layout (location = 2) in vec2 aTexCoord;

out vec3 ourColor;
out vec2 TexCoord;

void main()
{
    gl_Position = vec4(aPos, 1.0);
    ourColor = aColor;
    TexCoord = aTexCoord;
}
```

```
#version 330 core
out vec4 FragColor;

in vec3 ourColor;
in vec2 TexCoord;

uniform sampler2D ourTexture;

void main()
{
    FragColor = texture(ourTexture, TexCoord);
}
```

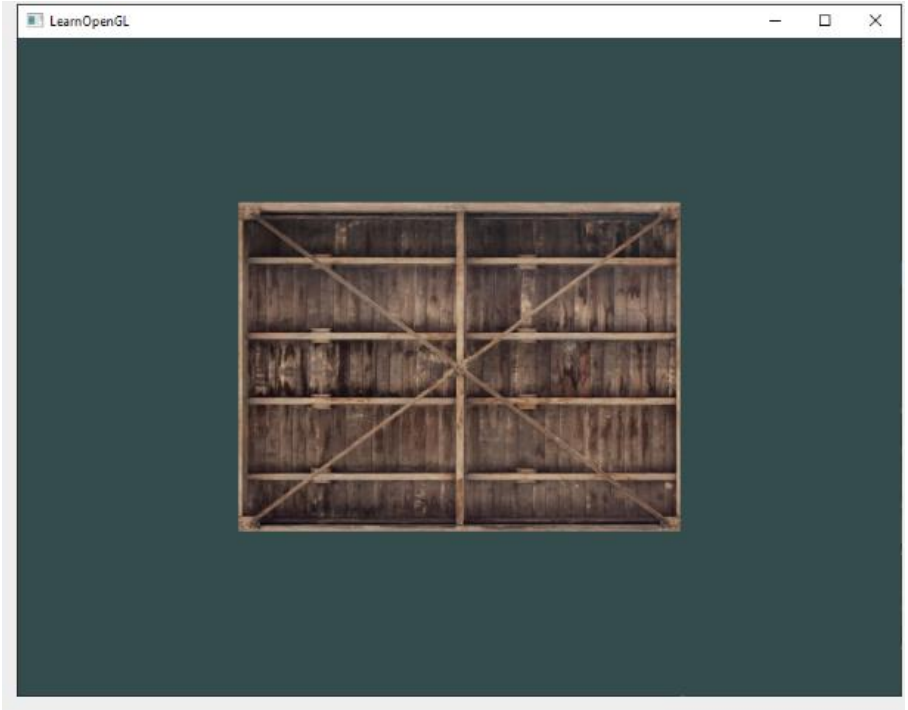
  - uniform Sampler2D형으로 샘플링할 2D 텍스처(sampler)를 읽음
  - GLSL의 texture(sampler, texture coordinate) 함수는 sampler로부터 텍스처 좌표에 해당하는 텍셀값을 읽음

- glDrawElements() 함수 호출 전 텍스처를 바인딩하면, 이제 해당 텍스처가 사각형에 입혀짐

```
glBindTexture(GL_TEXTURE_2D, texture);
glBindVertexArray(VAO);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
```

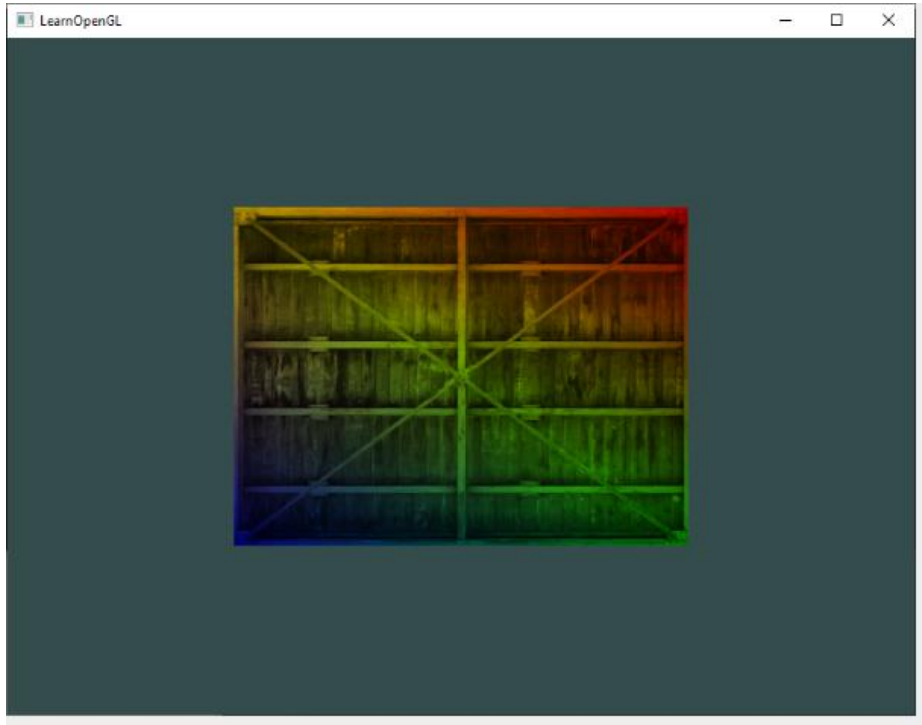
Applying textures

- 결과
  - 잘못 코딩시 희거나 검게 나올 수 있음



- FS에서 vertex color에 texture color를 곱하면?

```
FragColor = texture(ourTexture, TexCoord) * vec4(ourColor, 1.0);
```



Texture Units

- 텍스처 유닛을 사용하여, Shader에서 둘 이상의 텍스처도 사용 가능
  - OpenGL은 최소 16개의 텍스처 유닛을 가짐 (=16개 텍스처를 하나의 모델에 동시에 사용 가능)
  - 각 텍스처 유닛의 번호는 GL_TEXTURE0 ~ GL_TEXTURE15
- glActiveTexture()로 해당 유닛을 활성화 후, 이 유닛에 텍스처를 바인딩

```
glActiveTexture(GL_TEXTURE0); // activate the texture unit first before binding texture
glBindTexture(GL_TEXTURE_2D, texture);
```

- FS에서 두 개의 sampler를 가져와 8:2 비율로 혼합하는 예제

```
#version 330 core
...

uniform sampler2D texture1;
uniform sampler2D texture2;

void main()
{
    FragColor = mix(texture(texture1, TexCoord), texture(texture2, TexCoord), 0.2);
}
```

Texture Units

- 두 번째 텍스처 로드 및 생성 (알파값 존재)

```
unsigned char *data = stbi_load("awesomeface.png", &width, &height, &nrChannels, 0);
if (data)
{
    glTexImage2D(GL_TEXTURE_2D, 0, GL_RGB, width, height, 0, GL_RGBA, GL_UNSIGNED_BYTE, data);
    glGenerateMipmap(GL_TEXTURE_2D);
}
```

- glUniform1i 함수를 사용하여 FS의 sampler 설정

```
ourShader.use(); // don't forget to activate the shader before setting uniforms!
glUniform1i(glGetUniformLocation(ourShader.ID, "texture1"), 0); // set it manually
ourShader.setInt("texture2", 1); // or with shader class

while(...)
{
    [...]
}
```

- 두 텍스처를 두 개의 텍스처 유닛에 각각 바인딩한 후 사각형을 그림

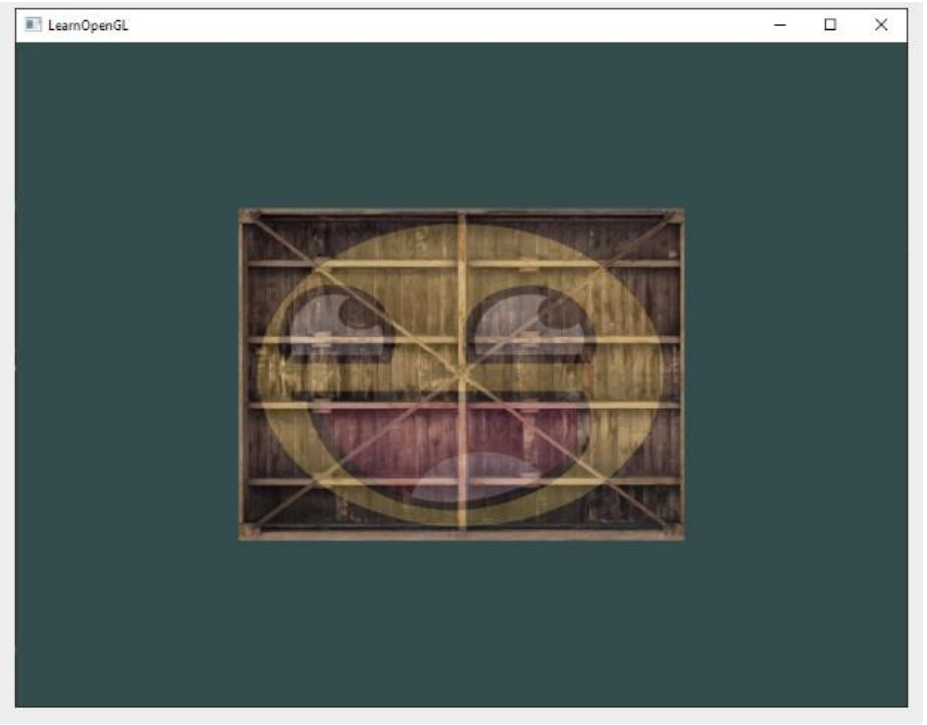
```
glActiveTexture(GL_TEXTURE0);
glBindTexture(GL_TEXTURE_2D, texture1);
glActiveTexture(GL_TEXTURE1);
glBindTexture(GL_TEXTURE_2D, texture2);

glBindVertexArray(VAO);
glDrawElements(GL_TRIANGLES, 6, GL_UNSIGNED_INT, 0);
```

Texture Units

- 결과

- 좌표계 보정 후 결과

```
stbi_set_flip_vertically_on_load(true);
```



ⓒ nvidia

Transformations

Transformations

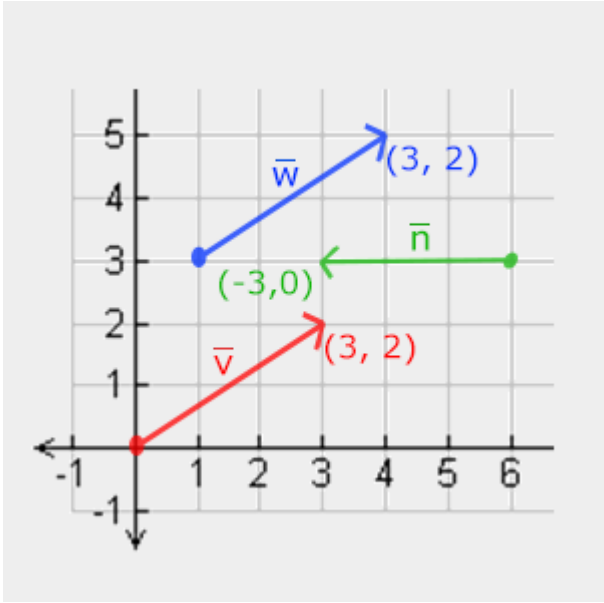
- 물체를 변환(transform), 즉 이동시키거나 변형하는 직관적인 방법
  - 매 프레임마다 vertex buffer 내의 vertex별로 하나하나 값을 변경하여 버퍼에 다시 넣으면 됨
  - 번거롭고 높은 비용을 요구

- 더 널리 쓰이는 방법 – 행렬(matrix)의 사용
  - 컴퓨터수학, 또는 선형대수학 과목에서 배운 간단한 수학적 개념 필요

- 가볍게 행렬과 벡터의 개념에 대해 review하고 넘어갑니다
  - 더 상세하게 알고 싶다면, 아래 온라인 코스 참조
    Vectors | Algebra (all content) | Math | Khan Academy
    Matrices | Algebra (all content) | Math | Khan Academy

Vectors

- 벡터
  - 기본적인 정의 : 방향 (direction)과 크기(magnitude)를 가진 물리량
  - 2~3차원 공간에서 방향을 표현 가능



- 벡터의 표현 방법
  - 기본적으로 열벡터

$$\bar{v} = \begin{pmatrix} x \\ y \\ z \end{pmatrix}$$

- 위치 벡터 (position vector)
  - 3차원 공간에서 원점을 (0, 0, 0)으로 놓으면 생성 가능
  - 한 점의 위치를 표현 가능

Vector Operations

- Scalar vector operations
  - 벡터 내 모든 element에 대해 scalar 값을 덧셈/뺄셈/곱셈/나눗셈

$$\begin{pmatrix}1\\2\\3\end{pmatrix} + x \rightarrow \begin{pmatrix}1\\2\\3\end{pmatrix} + \begin{pmatrix}x\\x\\x\end{pmatrix} = \begin{pmatrix}1+x\\2+x\\3+x\end{pmatrix}$$

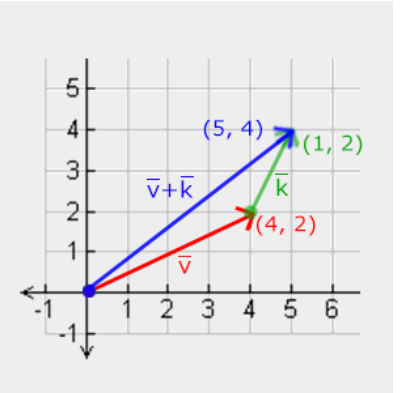
- Vector negation
  - 반대 방향의 벡터를 만듦 (scalar 값 -1을 곱한 것과 동일)

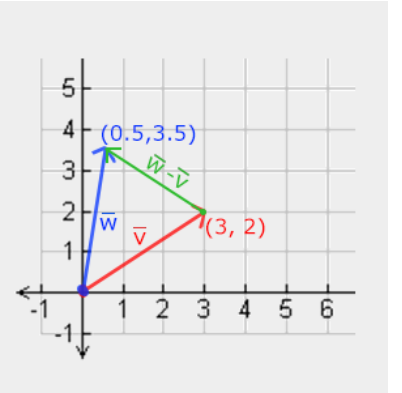
$$-\bar{v} = -\begin{pmatrix}v_x\\v_y\\v_z\end{pmatrix} = \begin{pmatrix}-v_x\\-v_y\\-v_z\end{pmatrix}$$

- Addition and subtraction
  - Component-wise, 즉 벡터의 각 component끼리 연산

$$\bar{v} = \begin{pmatrix}1\\2\\3\end{pmatrix}, \bar{k} = \begin{pmatrix}4\\5\\6\end{pmatrix} \rightarrow \bar{v} + \bar{k} = \begin{pmatrix}1+4\\2+5\\3+6\end{pmatrix} = \begin{pmatrix}5\\7\\9\end{pmatrix}$$

$$\bar{v} = \begin{pmatrix}1\\2\\3\end{pmatrix}, \bar{k} = \begin{pmatrix}4\\5\\6\end{pmatrix} \rightarrow \bar{v} + -\bar{k} = \begin{pmatrix}1+(-4)\\2+(-5)\\3+(-6)\end{pmatrix} = \begin{pmatrix}-3\\-3\\-3\end{pmatrix}$$

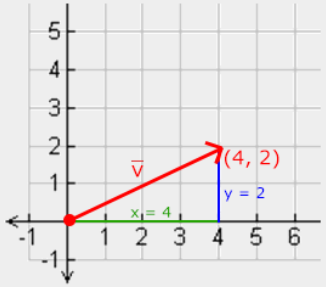




Vector Operations

- Length (크기 또는 길이)
  - 피타고라스 정리 이용
  - L_2 norm이라고 함

$$||\bar{v}|| = \sqrt{x^2 + y^2}$$



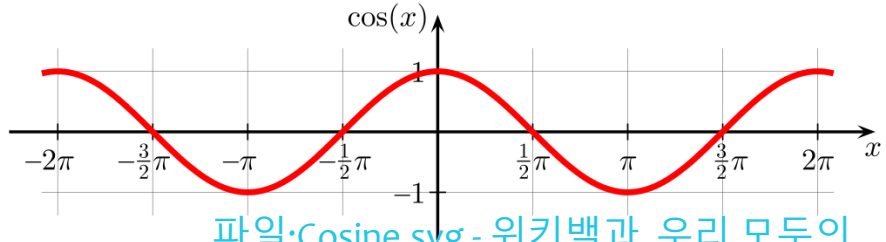
- Normalizing a vector (벡터의 정규화)
  - 어떠한 벡터를 길이가 1인 unit vector로 만드는 과정
  - 방향만 고려하는 벡터를 표현하고 싶을 때 사용

$$\hat{n} = \frac{\bar{v}}{||\bar{v}||}$$

$$||\bar{v}|| = \sqrt{4^2 + 2^2} = \sqrt{16 + 4} = \sqrt{20} = 4.47$$

Vector Operations

- Vector-vector multiplication
  - 내적(inner product)와 외적(크로스곱, cross product) 존재



파일:Cosine.svg - 위키백과, 우리 모두의 백과사전 (wikipedia.org)

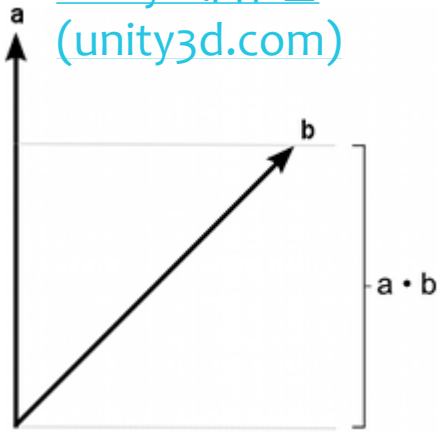
- Dot product
  - 각 벡터의 길이의 스칼라 곱에, 두 벡터 사이 각의 코사인 값을 곱함
    cosθ가 0이면 두 벡터가 직교(orthogonal), 1이면 평행(parallel)
  - 벡터의 컴포넌트끼리 값을 곱한 후, 이를 다 더해 줌

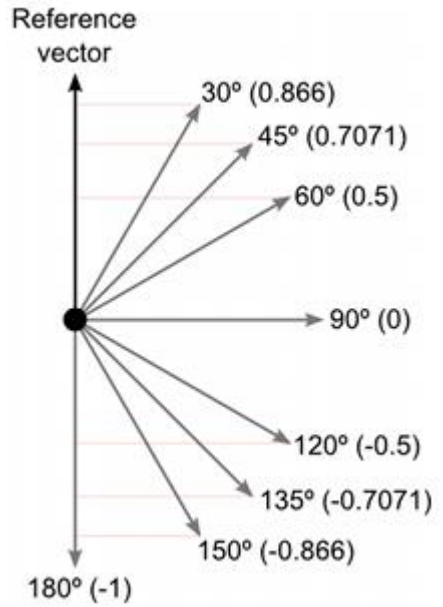
$$\bar{v} \cdot \bar{k} = ||\bar{v}|| \cdot ||\bar{k}|| \cdot \cos\theta$$

$$\begin{pmatrix} 0.6 \\ -0.8 \\ 0 \end{pmatrix} \cdot \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix} = (0.6 * 0) + (-0.8 * 1) + (0 * 0) = -0.8$$

  - 결과값은 스칼라
  - 두 벡터 모두 단위벡터일 경우,
    cosθ는 첫 번째 벡터가 두 번째 벡터의 방향으로 얼마나 기울었는지 표현
  - Lighting 연산 (반사 각도 계산) 등에 많이 사용

벡터 연산 이해 - Unity 매뉴얼 (unity3d.com)

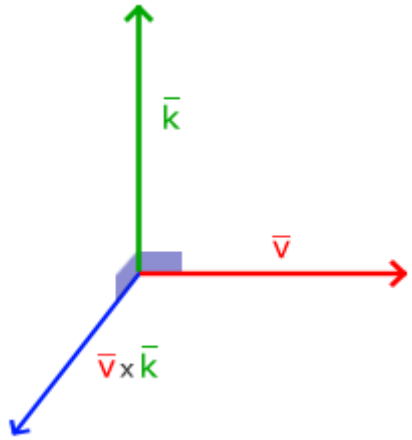




Vector Operations

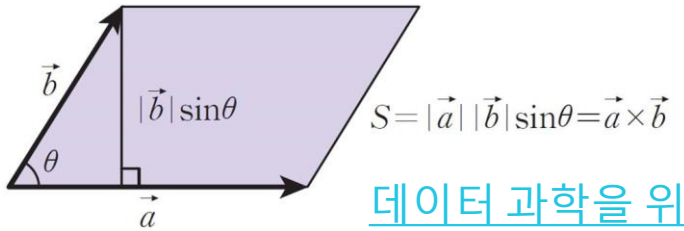
- Cross product
  - 3D 공간에서 평행하지 않는 두 개의 벡터와 직교하는 새로운 벡터 생성 (오른손 법칙에 따라 방향 설정)

$$\begin{pmatrix} A_x \\ A_y \\ A_z \end{pmatrix} \times \begin{pmatrix} B_x \\ B_y \\ B_z \end{pmatrix} = \begin{pmatrix} A_y \cdot B_z - A_z \cdot B_y \\ A_z \cdot B_x - A_x \cdot B_z \\ A_x \cdot B_y - A_y \cdot B_x \end{pmatrix}$$



  - Cross product의 크기 : 두 벡터로 이루어진 평행사변형의 넓이

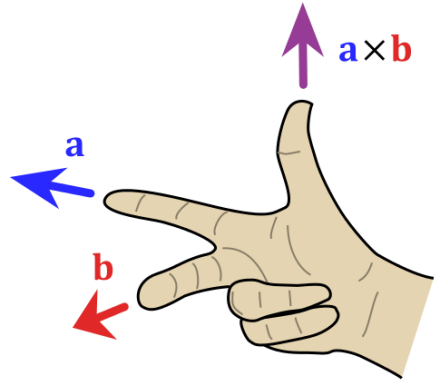
$$|\vec{a} \times \vec{b}| = |\vec{a}|\,|\vec{b}| \sin\theta$$



[그림 6-13] 평행사변형의 넓이와 벡터의 외적

데이터 과학을 위한 기초수학 with 파이썬 (hanbit.co.kr)

  - 평면이 향하는 방향, 즉 노멀 벡터(normal vector)를 구하는 데 사용



오른손법칙(right-hand rule) : 기준(基準, standard) : 네이버 블로그 (naver.com)

Matrices

- 행렬: 숫자, 기호, 수식들의 사각 배열(array)
  - 각각의 성분은 행렬의 요소(element)라고 함
  - M개 행과 N개 열로 이루어진 행렬의 차원(dimension)은 MxN
  - i번째 행, j번째 열의 성분은 (i, j)와 같이 인덱싱
  - 벡터 여러 개를 쌓으면 행렬로 표현 가능

$$\begin{bmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \end{bmatrix}$$

- Addition and subtraction
  - 같은 차원의 행렬끼리만 가능
  - 각 요소별로 덧셈/뺄셈 계산

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} + \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1+5 & 2+6 \\ 3+7 & 4+8 \end{bmatrix} = \begin{bmatrix} 6 & 8 \\ 10 & 12 \end{bmatrix}$$

$$\begin{bmatrix} 4 & 2 \\ 1 & 6 \end{bmatrix} - \begin{bmatrix} 2 & 4 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 4-2 & 2-4 \\ 1-0 & 6-1 \end{bmatrix} = \begin{bmatrix} 2 & -2 \\ 1 & 5 \end{bmatrix}$$

- Matrix-scalar products

$$2 \cdot \begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} = \begin{bmatrix} 2 \cdot 1 & 2 \cdot 2 \\ 2 \cdot 3 & 2 \cdot 4 \end{bmatrix} = \begin{bmatrix} 2 & 4 \\ 6 & 8 \end{bmatrix}$$

Matrix-matrix multiplication

- MxN 행렬과 NxP 행렬간에만 곱셈 가능
  - 결과는 MxP 행렬
  - 교환법칙 불가 A·B≠B·A.

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \cdot \begin{bmatrix} 5 & 6 \\ 7 & 8 \end{bmatrix} = \begin{bmatrix} 1\cdot 5+2\cdot 7 & 1\cdot 6+2\cdot 8 \\ 3\cdot 5+4\cdot 7 & 3\cdot 6+4\cdot 8 \end{bmatrix} = \begin{bmatrix} 19 & 22 \\ 43 & 50 \end{bmatrix}$$

- 왼쪽 행렬의 행 성분과 오른쪽 행렬의 열 성분을 곱해 더하면 됨

$$\begin{bmatrix} 4 & 2 & 0 \\ 0 & 8 & 1 \\ 0 & 1 & 0 \end{bmatrix} \cdot \begin{bmatrix} 4 & 2 & 1 \\ 2 & 0 & 4 \\ 9 & 4 & 2 \end{bmatrix} = \begin{bmatrix} 4\cdot 4+2\cdot 2+0\cdot 9 & 4\cdot 2+2\cdot 0+0\cdot 4 & 4\cdot 1+2\cdot 4+0\cdot 2 \\ 0\cdot 4+8\cdot 2+1\cdot 9 & 0\cdot 2+8\cdot 0+1\cdot 4 & 0\cdot 1+8\cdot 4+1\cdot 2 \\ 0\cdot 4+1\cdot 2+0\cdot 9 & 0\cdot 2+1\cdot 0+0\cdot 4 & 0\cdot 1+1\cdot 4+0\cdot 2 \end{bmatrix} = \begin{bmatrix} 20 & 8 & 12 \\ 25 & 4 & 34 \\ 2 & 0 & 4 \end{bmatrix}$$

Matrix-vector multiplication

- 벡터(=Nx1 차원의 행렬)에 위치, 컬러, 텍스처 좌표를 저장
- NxN 행렬에 벡터(=Nx1 행렬)을 곱하면, 변환(transform)된 벡터(=Nx1 행렬) 산출 가능

- Identity matrix (항등 또는 단위 행렬, unit matrix)
  - 대각 성분에 1을, 나머지 성분에 0을 가지는 NxN의 대각 행렬 (diagonal matrix)
  - 단위 행렬에 벡터를 곱해도 벡터는 그대로임
  - 초기 행렬 설정으로 많이 사용

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix} = \begin{bmatrix} 1 \cdot 1 \\ 1 \cdot 2 \\ 1 \cdot 3 \\ 1 \cdot 4 \end{bmatrix} = \begin{bmatrix} 1 \\ 2 \\ 3 \\ 4 \end{bmatrix}$$

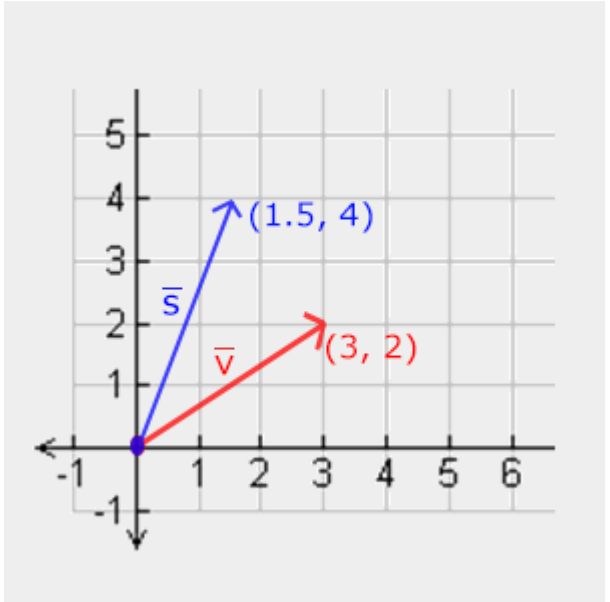
- 4x4 행렬을 사용하는 이유
  - 보통 벡터의 크기가 4이기 때문 (xyzw)

- Homogeneous coordinates (동차 좌표) – 벡터의 w요소
  - Perspective division: x, y, z 좌표를 w 좌표로 나눌 때 사용 (원근감 부여 가능 – 추후 다시 설명)
  - 동차 좌표의 값이 1이면 3D 벡터를 이동 가능, 0이면 방향 벡터이므로 이동 불가 (다음 슬라이드)

Scaling (확대/축소)

- x, y, z축 방향의 scaling factor를 S1, S2, S3에 대입
  - Uniform scale: S1=S2=S3
  - Non-uniform scale: S1, S2, S3 중 하나라도 값이 다름

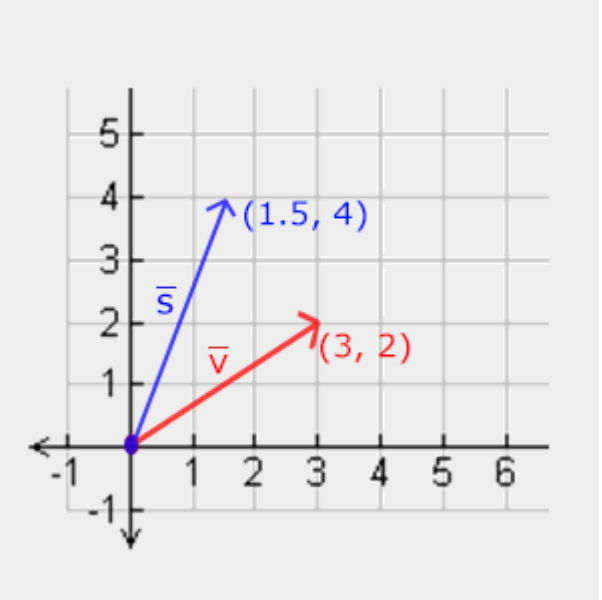
$$\begin{bmatrix} S_1 & 0 & 0 & 0 \\ 0 & S_2 & 0 & 0 \\ 0 & 0 & S_3 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} S_1 \cdot x \\ S_2 \cdot y \\ S_3 \cdot z \\ 1 \end{pmatrix}$$



Translation (이동)

- 이동 벡터 (T_x,T_y,T_z)를 왼쪽 4x4 행렬의 4번째 열의 1~3번째 행 성분으로 넣음
  - 이 값들이 오른쪽 곱하는 벡터의 좌표값에 더해지게 됨

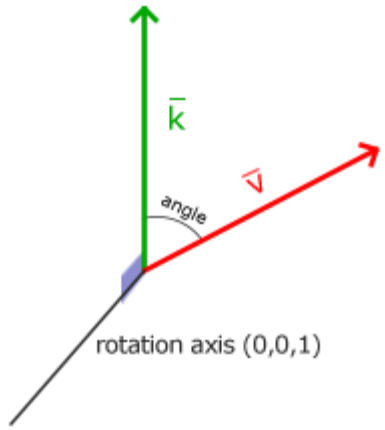
$$\begin{bmatrix} 1 & 0 & 0 & T_x \\ 0 & 1 & 0 & T_y \\ 0 & 0 & 1 & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x + T_x \\ y + T_y \\ z + T_z \\ 1 \end{pmatrix}$$



- 오른쪽 곱하는 벡터의 동차 좌표가 0이면 이동이 실행되지 않음
  - 즉, 정점이면 w에 1을, 방향 벡터이면 w에 0을 넣으면 됨

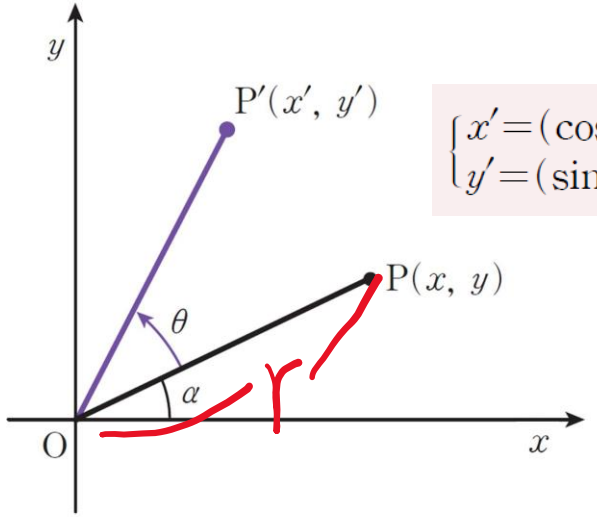
Rotation (회전)

- 2D/3D에서의 회전 정도는 각(angle)으로 나타냄
  - 각도(degree) 또는 라디안(radian)으로 표현 가능 (360도 = 2PI, PI≈3.14159265359)
  - 각도에 의한 각 = 라디안 * (180.0f / PI)
  - 라디안에 의한 각 = 각도 * (PI / 180.0f)



- 2차원 회전변환 공식
  - 점 $P(x,y)$를 원점 O를 중심으로 각 θ만큼 회전시키는 변환

$x' = r\cos(\theta+\alpha)$
$= r\cos\theta\cos\alpha - r\sin\theta\sin\alpha$
$y' = r\sin(\theta+\alpha)$
$= r\sin\theta\cos\alpha + r\cos\theta\sin\alpha$
$x = r\cos\alpha,\ y = r\sin\alpha$ 이므로,
$x' = \cos\theta x - \sin\theta y$
$y' = \sin\theta x + \cos\theta y$



$$\begin{cases} x' = (\cos\theta)x - (\sin\theta)y \\ y' = (\sin\theta)x + (\cos\theta)y \end{cases} \Leftrightarrow \begin{bmatrix} x' \\ y' \end{bmatrix} = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix}$$

데이터 과학을 위한 기초수학 with 파이썬 (hanbit.co.kr)

Rotation (회전)

- 3D 공간에서 x축을 기준으로 회전시 (아래 식에서 $\beta = \theta$)

$$x' = x$$
$$y' = y\cos\beta - z\sin\beta$$
$$z' = y\sin\beta + z\cos\beta$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta & 0 \\ 0 & \sin\theta & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} x \\ \cos\theta \cdot y - \sin\theta \cdot z \\ \sin\theta \cdot y + \cos\theta \cdot z \\ 1 \end{pmatrix}$$

- y축을 기준으로 회전시

$$x' = z\sin\beta + x\cos\beta$$
$$y' = y$$
$$z' = z\cos\beta - x\sin\beta$$

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin\theta & 0 & \cos\theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta \cdot x + \sin\theta \cdot z \\ y \\ -\sin\theta \cdot x + \cos\theta \cdot z \\ 1 \end{pmatrix}$$

- z축을 기준으로 회전시

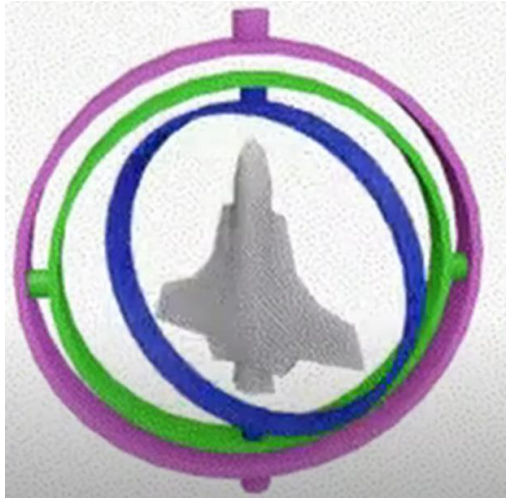
$$x' = x\cos\beta - y\sin\beta$$
$$y' = x\sin\beta + y\cos\beta$$
$$z' = z$$

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 & 0 \\ \sin\theta & \cos\theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} = \begin{pmatrix} \cos\theta \cdot x - \sin\theta \cdot y \\ \sin\theta \cdot x + \cos\theta \cdot y \\ z \\ 1 \end{pmatrix}$$

변환 (Transforms) (5) – 3차원 변환 : 네이버 블로그 (naver.com)

Rotation (회전)

- 각 축에 대한 행렬을 조합하여 사용 가능
  - 짐벌락 (Gimbal lock) 현상 발생 가능
    즉, 두 개 혹은 세 개의 축이 같은 방향으로 겹쳐서 회전축이 소멸하는 상황
  - Gimbal Lock & Quaternions | Breakthrough Junior Challenge 2021 - YouTube



- 복소수를 확장한 사원수 (쿼터니언, quaternion) 개념을 사용하면 짐벌락 예방 가능
  - 유니티와 같은 게임 엔진에서는 오일러각을 쿼터니언으로 변환 후 회전을 수행하는 함수 제공
  - 6. 회전의 수학 II : 사원수 - YouTube

Combining matrices

- 여러 변환을 하나의 행렬에 조합 가능!
  - 행렬 곱은 교환법칙이 성립하지 않으므로, 순서가 중요
  - 스케일, 회전, 이동 순으로 벡터에 적용 권장 (즉, 행렬의 순서는 이동, 회전, 스케일 행렬 순)

$$Trans.Scale = \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 2 \\ 0 & 0 & 1 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 2 & 0 & 0 & 0 \\ 0 & 2 & 0 & 0 \\ 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

- 벡터를 2배만큼 확대하고 x, y, z축으로 1,2,3만큼 이동시키는 행렬

$$\begin{bmatrix} 2 & 0 & 0 & 1 \\ 0 & 2 & 0 & 2 \\ 0 & 0 & 2 & 3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} 2x+1 \\ 2y+2 \\ 2z+3 \\ 1 \end{bmatrix}$$

In practice (GLM)

- GLM (OpenGL Mathematics)
  - 헤더 파일들만 포함하는 수학 라이브러리
  - [glm/glm at master · g-truc/glm (github.com)]

- 대부분의 행렬/벡터 연산 기능은 GLM의 파일들 중 아래 세 개만 include하는 것으로 충분

```
#include <glm/glm.hpp>
#include <glm/gtc/matrix_transform.hpp>
#include <glm/gtc/type_ptr.hpp>
```

- Translation 예시 (210 출력)
  - (1+1, 0+1, 0+0) = (2, 1, 0)

```
glm::vec4 vec(1.0f, 0.0f, 0.0f, 1.0f);
glm::mat4 trans = glm::mat4(1.0f);
trans = glm::translate(trans, glm::vec3(1.0f, 1.0f, 0.0f));
vec = trans * vec;
std::cout << vec.x << vec.y << vec.z << std::endl;
```

- TRS 조합 예시
  - 0.5배로 축소한 후, Z축을 중심으로 90도 회전

```
glm::mat4 trans = glm::mat4(1.0f);
trans = glm::rotate(trans, glm::radians(90.0f), glm::vec3(0.0, 0.0, 1.0));
trans = glm::scale(trans, glm::vec3(0.5, 0.5, 0.5));
```

In practice (Shader)

- glsl 파일 수정

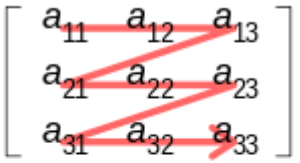
```
#version 330 core
layout (location = 0) in vec3 aPos;
layout (location = 1) in vec2 aTexCoord;

out vec2 TexCoord;

uniform mat4 transform;

void main()
{
    gl_Position = transform * vec4(aPos, 1.0f);
    TexCoord = vec2(aTexCoord.x, aTexCoord.y);
}
```

- c/cpp 파일 수정

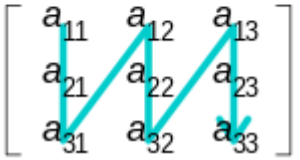
```
unsigned int transformLoc = glGetUniformLocation(ourShader.ID, "transform");
glUniformMatrix4fv(transformLoc, 1, GL_FALSE, glm::value_ptr(trans));
```

  - glUniformMatrix4v()는 4x4 행렬 데이터를 shader로 보내기 위한 함수
  - 각 파라미터: uniform의 location, 행렬 개수, 전치(transpose) 여부, 데이터의 포인터
  - GLM와 OpenGL은 모두 내부적으로 열 우선 순서(column-major ordering)를 따르므로 전치가 따로 필요 없음
  - 렌더 루프 안에 위 코드를 넣어야 매 프레임 업데이트된 행렬이 전달됨

Row-major order



Column-major order



Row- and column-major order - Wikipedia

In practice (Result)

- 결과물
  - Vertex를 업데이트하지 않고도 uniform 행렬 변수만으로 물체의 변환 완료

ⓒ nvidia

마무리

마무리

- 이번 시간에는 아래와 같은 내용을 살펴보았습니다.
  - 텍스처 매핑의 개념과 적용 방법
  - 선형변환을 이용한 스케일링, 이동, 회전 방법

- 실습 문제
  - 지난 시간에 그린 사각형에 구글에서 적당한 텍스처를 다운받아 입혀 보세요.
  - 벽지 문양처럼 반복되도록 해 보세요.
  - 또 다른 텍스처를 다운받아 두 텍스처가 함께 섞여서 입혀지도록 해 보세요.
    (흡사 morphing 처럼 키 입력으로 mix 정도를 조절)
  - 텍스처가 입혀진 사각형이 이동하도록 바꿔 보세요.

- 다음 주차에는 Getting started의 마지막 시간으로써 아래 내용을 다룹니다.
  - Coordinate Systems
  - Camera