게임엔진

제16강 OGRE 셰이더

학습 안내

■학습목표

- □ 직접 조명 모델에 기반한 CG 셰이더셰이더를 구현해본다.

■학습 내용

- □ 평면 셰이딩 / 고로 셰이딩 / 퐁 셰이딩
- □ CG 셰이더를 이용한 셰이딩 구현 실습

셰이딩(Shading)

■직접 조명 모델에서 가장 중요한 요소는?

□법선 벡터!

$$I = K_a \otimes L_a + (K_d \otimes L_d)\cos\theta + (K_s \otimes L_s)(\cos\alpha)^n$$

■기본 셰이딩의 종류 - 법선의 계산 방식이 각각 다름

□평면 셰이딩(Flat shading)

□고로 셰이딩(Gouraud shading)

□퐁 셰이딩(Phong shading)

평면 셰이딩

■계산법

- □ 평면의 법선 벡터를 이용한 조명 계산 결과값으로 평면 내부를 모두 채우는 방법
- □ 폴리곤 면 전체가 동일한 색상으로 칠해짐.

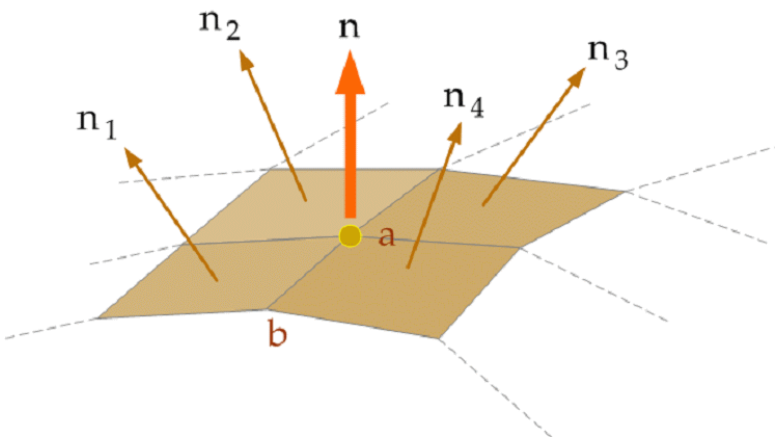
■특징

- □ 빠른 실행 속도.
- □ 렌더링 품질이 떨어짐.

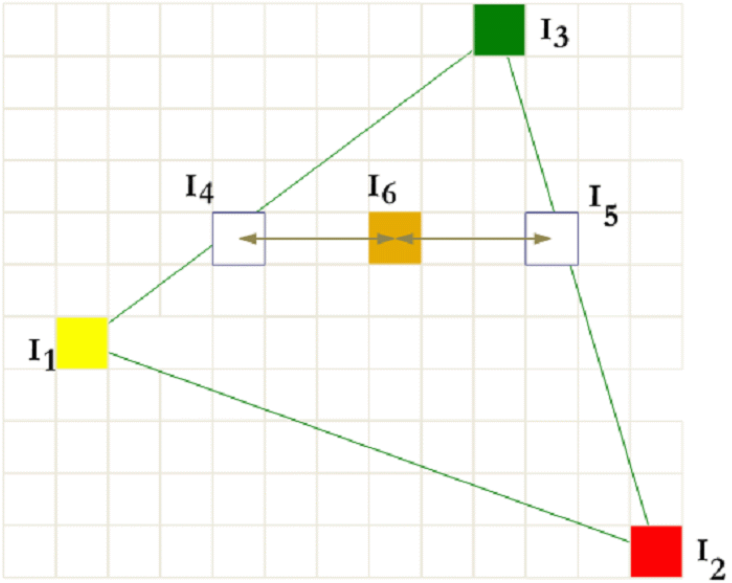
고로 셰이딩(Gouraud Shading)

■계산법

- □ 정점의 색상을 계산.
  - ■ 정점의 법선벡터는 인접면의 법선벡터를 평균하여 구함
- □ 정점의 색으로부터 내부면의 점들의 색을 선형보간.



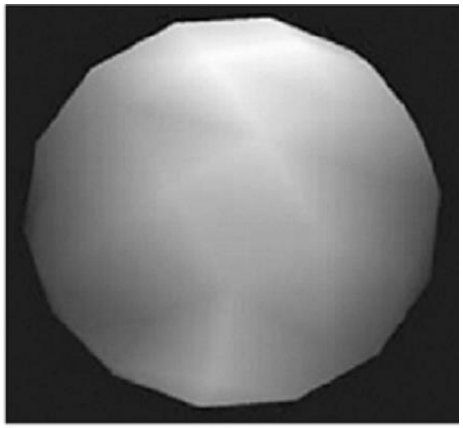
$$n = \frac{n_1 + n_2 + n_3 + n_4}{|n_1 + n_2 + n_3 + n_4|}$$



■특징

- □ 플랫 셰이딩보다는 부드러움
- □ 마하 밴드 효과는 그대로 남아있음.
- □ 전반사를 제대로 표현할 수 없음.

(a)

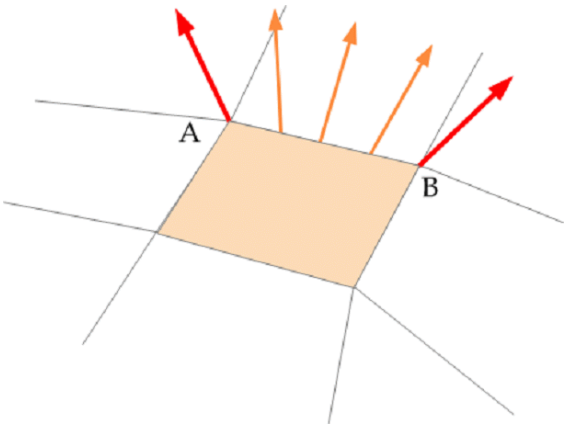


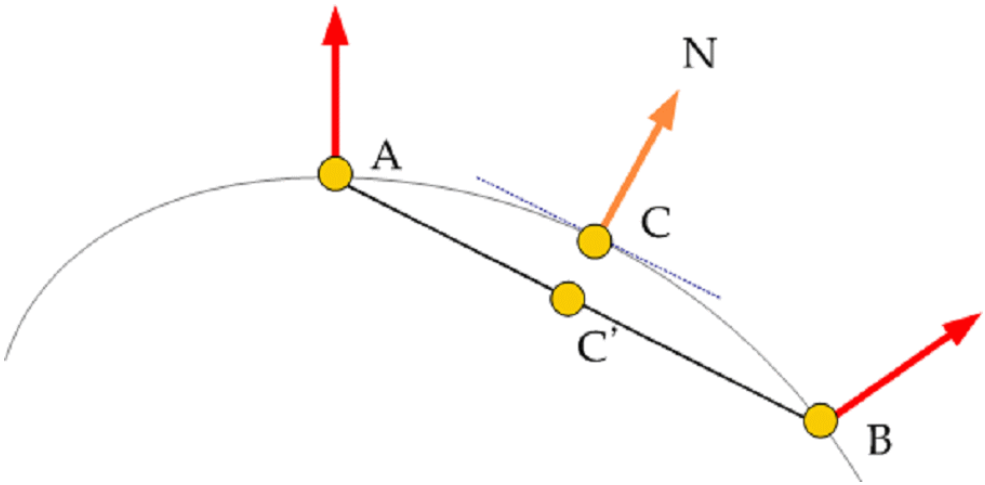
(b)

퐁 셰이딩(Phong Shading)

■계산법

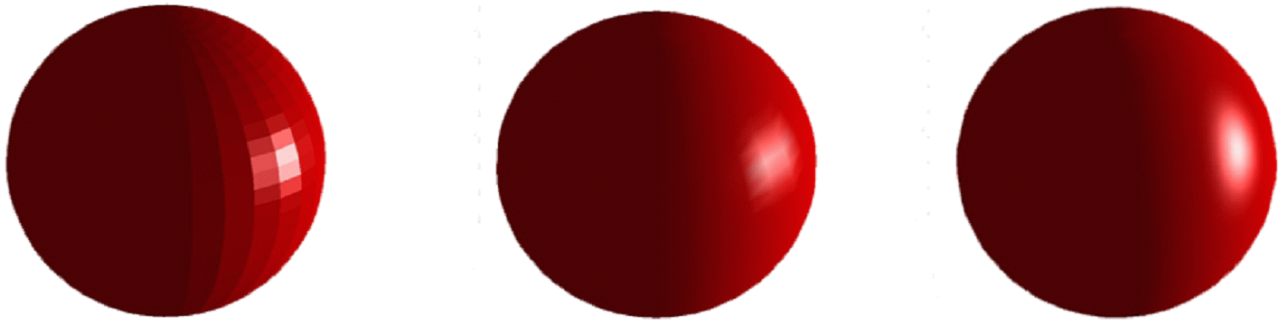
- □ 정점의 법선 벡터를 계산.
- □ 평면 상의 점들의 법선 벡터을 일일이 보간.
- □ 점들의 법선 벡터를 이용하여, 최종 색상을 계산함.





■특징

- □ 매우 사실적인 표현이 가능 - 전반사 성분도 훌륭하게 표현.
- □ 상당한 계산량이 필요하므로, 하드웨어적인 지원을 받지 못하면 실시간 구현이 매우 어려움.



각종 재질의 파라미터

Material	GL_AMBIENT	GL_DIFFUSE	GL_SPECULAR	GL_SHININESS
Brass	0.329412 0.223529 0.027451 1.0	0.780392 0.568627 0.113725 1.0	0.992157 0.941176 0.807843 1.0	27.8974
Bronze	0.2125 0.1275 0.054 1.0	0.714 0.4284 0.18144 1.0	0.393548 0.271906 0.166721 1.0	25.6
Polished Bronze	0.25 0.148 0.06475 1.0	0.4 0.2368 0.1036 1.0	0.774597 0.458561 0.200621 1.0	76.8
Chrome	0.25 0.25 0.25 1.0	0.4 0.4 0.4 1.0	0.774597 0.774597 0.774597 1.0	76.8
Copper	0.19125 0.0735 0.0225 1.0	0.7038 0.27048 0.0828 1.0	0.256777 0.137622 0.086014 1.0	12.8
Polished Copper	0.2295 0.08825 0.0275 1.0	0.5508 0.2118 0.066 1.0	0.580594 0.223257 0.0695701 1.0	51.2
Gold	0.24725 0.1995 0.0745 1.0	0.75164 0.60648 0.22648 1.0	0.628281 0.555802 0.366065 1.0	51.2
Polished Gold	0.24725 0.2245 0.0645 1.0	0.34615 0.3143 0.0903 1.0	0.797357 0.723991 0.208006 1.0	83.2
Pewter	0.105882 0.058824 0.113725 1.0	0.427451 0.470588 0.541176 1.0	0.333333 0.333333 0.521569 1.0	9.84615

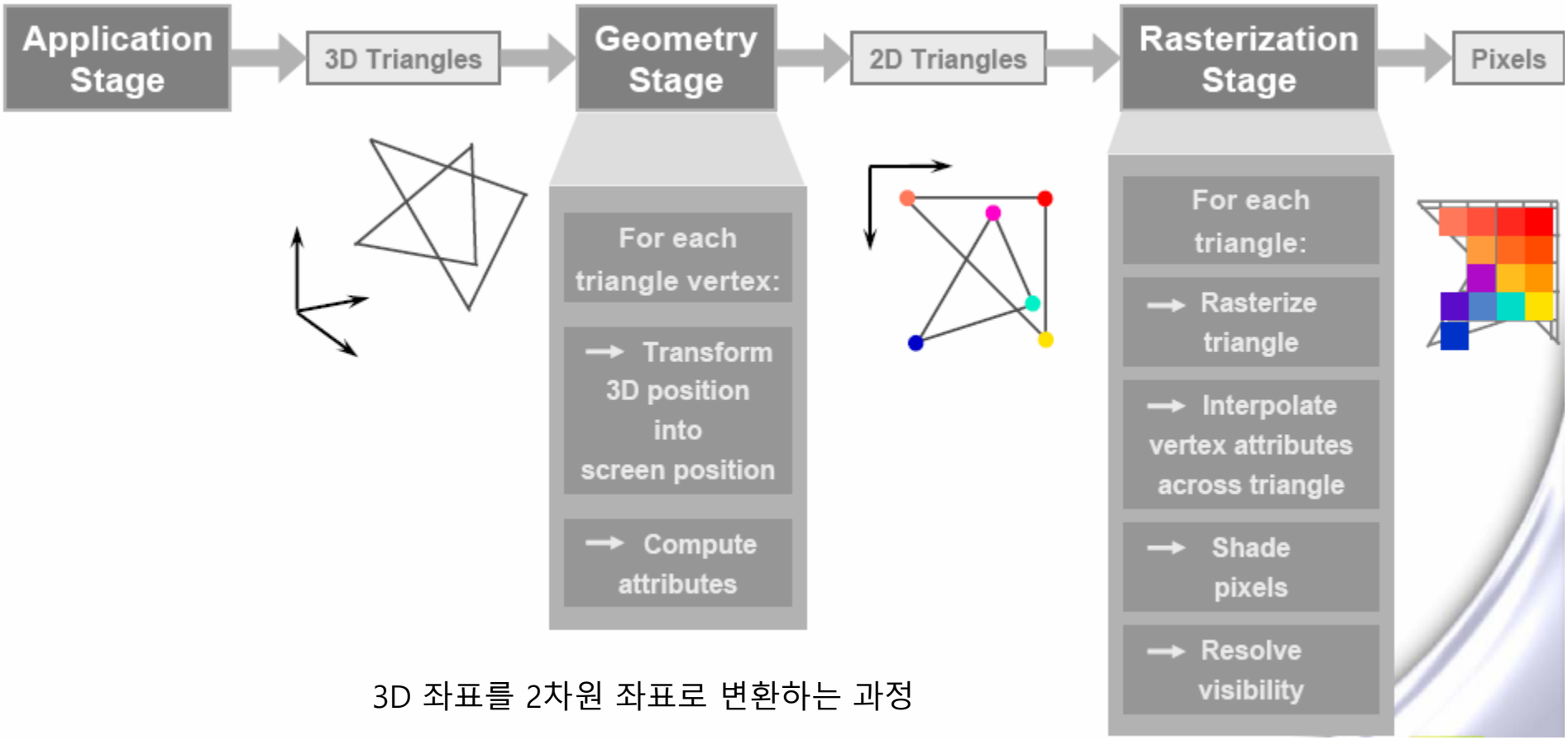
Technique & Pass

■**Technique:**
- □ 어떤 물체를 렌더링하는 하나의 "방법"
- □ 1~N개의 Pass로 구성됨

■**Pass:**
- □ 실제로 렌더링하는 작업에 대해 정의
- □ 1~N개 사용

■**Multi-pass의 예**
- □ 1) 기본적인 쉐이딩
  - ▪ 조명 계산, 텍스처 매핑 등
- □ 2) Glow 효과 연출

3D 그래픽 렌더링 파이프라인



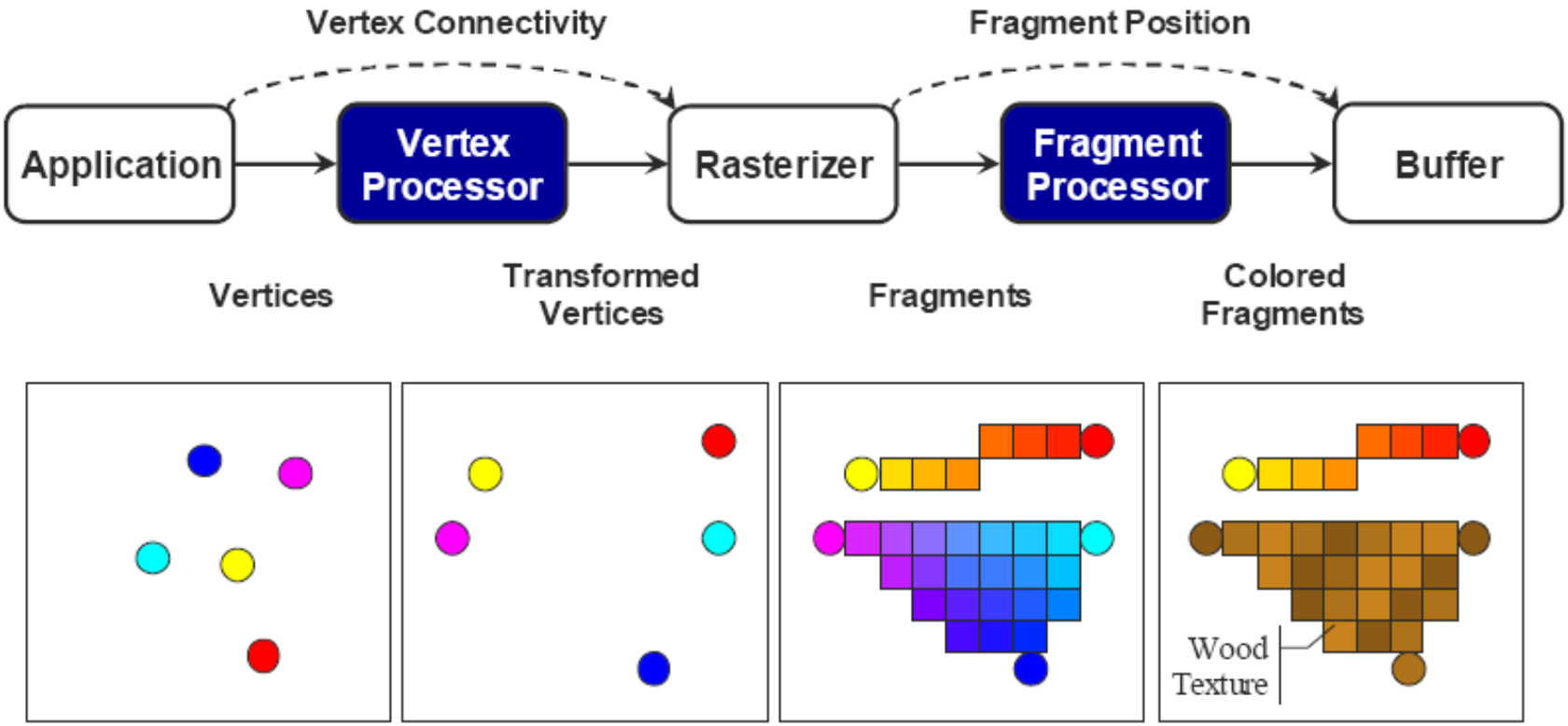
3D 좌표를 2차원 좌표로 변환하는 과정

점, 선분, 다각형등꼭지점으로표현된기하프리미티브들을이미지를구성하는픽셀단위로재구성하는과정

셰이더란?

- 정점(vertex)와 화소(pixel)의 색상을 결정하는 프로그램
  - 정점 셰이더는 색상 뿐만 아니라, 2D 공간에서의 위치도 결정.

- GPU 의 vertex processor와 fragment processor에서 실행되는 프로그램



셰이더의 입출력 및 기능

- Vertex shader
  - 입력: 정점 속성
  - 기능
    - 정점 위치 변환
    - 정점당 조명 계산
    - 정규화 변환
    - 텍스쳐 좌표 변환 및 생성
    - 정점 색상 계산
    - 하드웨어 스키닝
  - 출력: 정점 위치(클리핑 공간) 및 색상

- Fragment shader
  - 입력: 정점 셰이더에서 출력된 정점으로 보간된 각 화소 속성
  - 기능
    - 텍스쳐링
    - 화소당 조명 계산
    - 화소 색상 계산
  - 출력: 화소 색상

Cg란?

- C for graphics

- GPU 프로그래밍 언어
  - an open-source high-level shading language to make graphics programming faster and easier
- 하드웨어 셰이딩 언어
  - High-level, looks like C
  - API-independent
    - GLSL tied to OpenGL; HLSL tied to Direct3D
  - Platform-independent
    - PlayStation3, ATI, NVIDIA, Linux, MacOS X, Windows, etc.
- Cg was developed in close collaboration with Microsoft and is syntactically equivalent to HLSL, the shading language in DirectX 9

왜 Cg 인가?

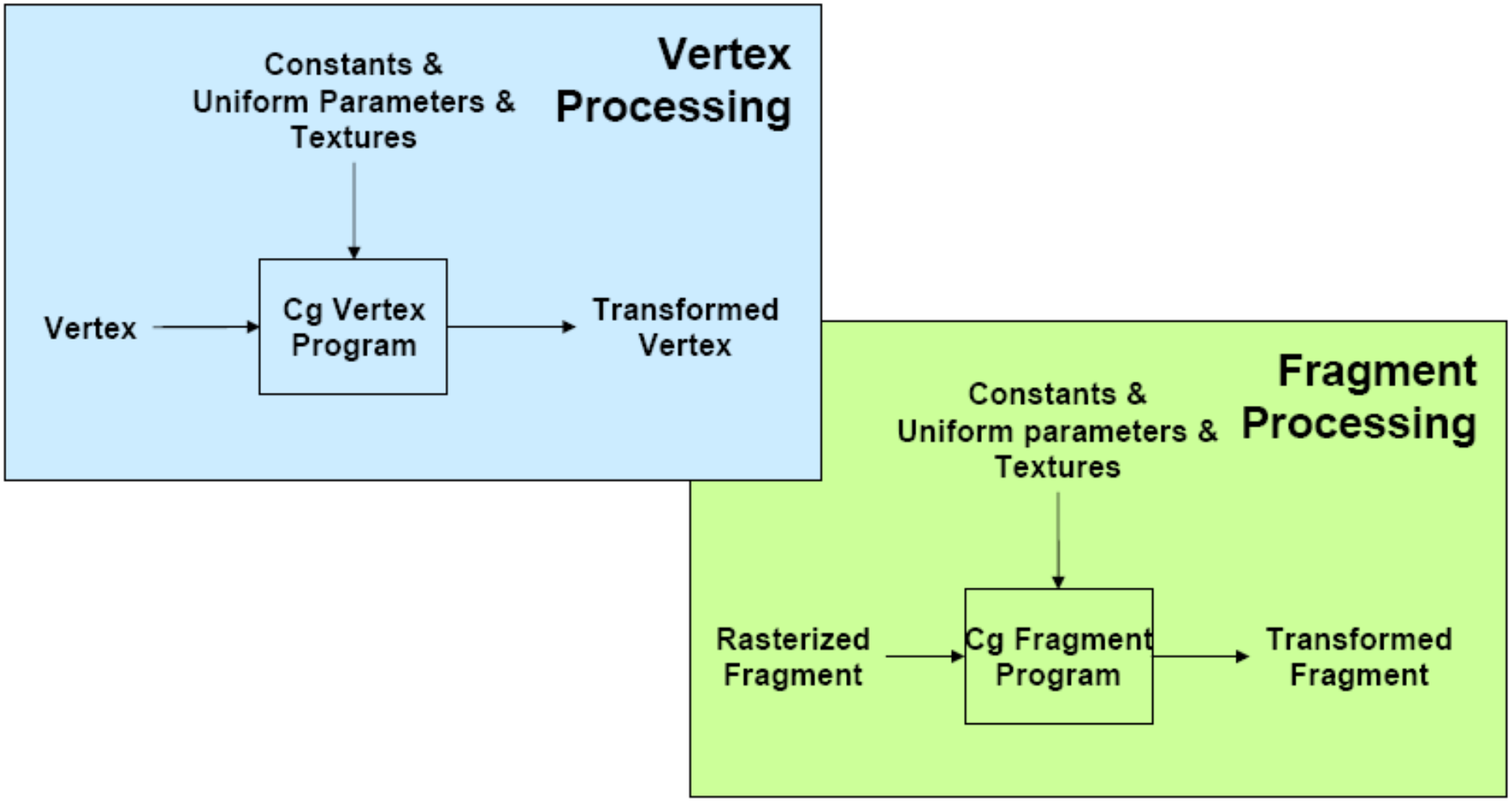
- Graphics hardware has become increasingly more powerful
- Programming powerful hardware with assembly code is hard
- GeForce FX supports programs more than 1,000 assembly instructions long
- Programmers need the benefits of a high-level language:
  - Easier programming
  - Easier code reuse
  - Easier debugging

Assembly

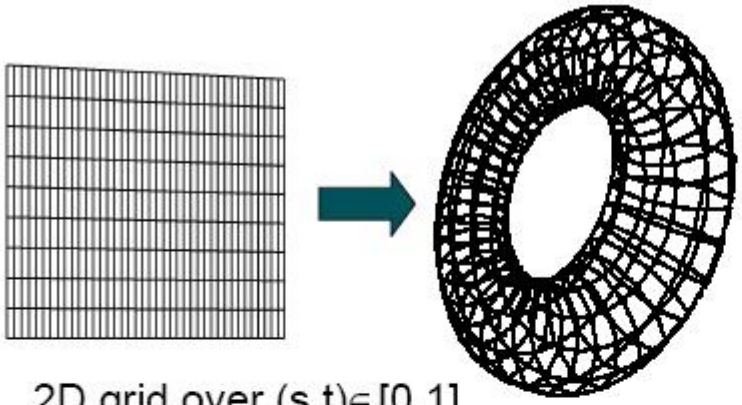
```
...
DP3 R0, c[11].xyzx, c[11].xyzx;
RSQ R0, R0.x;
MUL R0, R0.x, c[11].xyzx;
MOV R1, c[3];
MUL R1, R1.x, c[0].xyzx;
DP3 R2, R1.xyzx, R1.xyzx;
RSQ R2, R2.x;
MUL R1, R2.x, R1.xyzx;
ADD R2, R0.xyzx, R1.xyzx;
DP3 R3, R2.xyzx, R2.xyzx;
RSQ R3, R3.x;
MUL R2, R3.x, R2.xyzx;
DP3 R2, R1.xyzx, R2.xyzx;
MAX R2, c[3].z, R2.x;
MOV R2.z, c[3].y;
MOV R2.w, c[3].y;
LIT R2, R2;
...
```

```
float3 cSpecular = pow(max(0, dot(Nf, H)),
                       phongExp).xxx;
float3 cPlastic = Cd * (cAmbient + cDiffuse) +
                  Cs * cSpecular;
```

Cg 셰이더의 역할



정점 셰이더와 프래그먼트 셰이더의 역할 예시

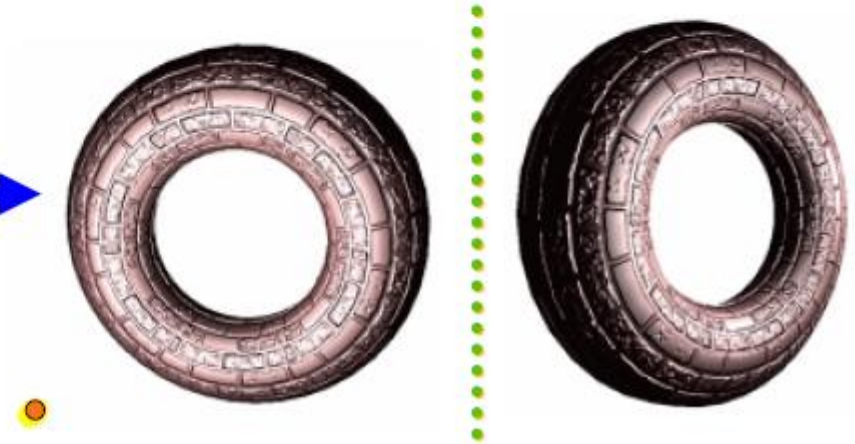


2D grid over (s,t)∈[0,1]

Tessellated torus

← Vertex program

Fragment program →



실습

Simple Shader
고로, 퐁, 및 데칼 텍스쳐 셰이딩

class InputController


```
bool keyPressed( const OIS::KeyEvent &evt )
{
  switch(evt.key)
  {
  case OIS::KC_O: mSceneMgr->getEntity("Ninja")->setMaterialName("Examples/Ninja"); break;
  case OIS::KC_P: mSceneMgr->getEntity("Ninja")->setMaterialName("PhongShading"); break;
  case OIS::KC_T: mSceneMgr->getEntity("Ninja")->setMaterialName("PhongShadingDecal"); break;
  case OIS::KC_G: mSceneMgr->getEntity("Ninja")->setMaterialName("GouraudShading"); break;

  }

  return true;

}
```

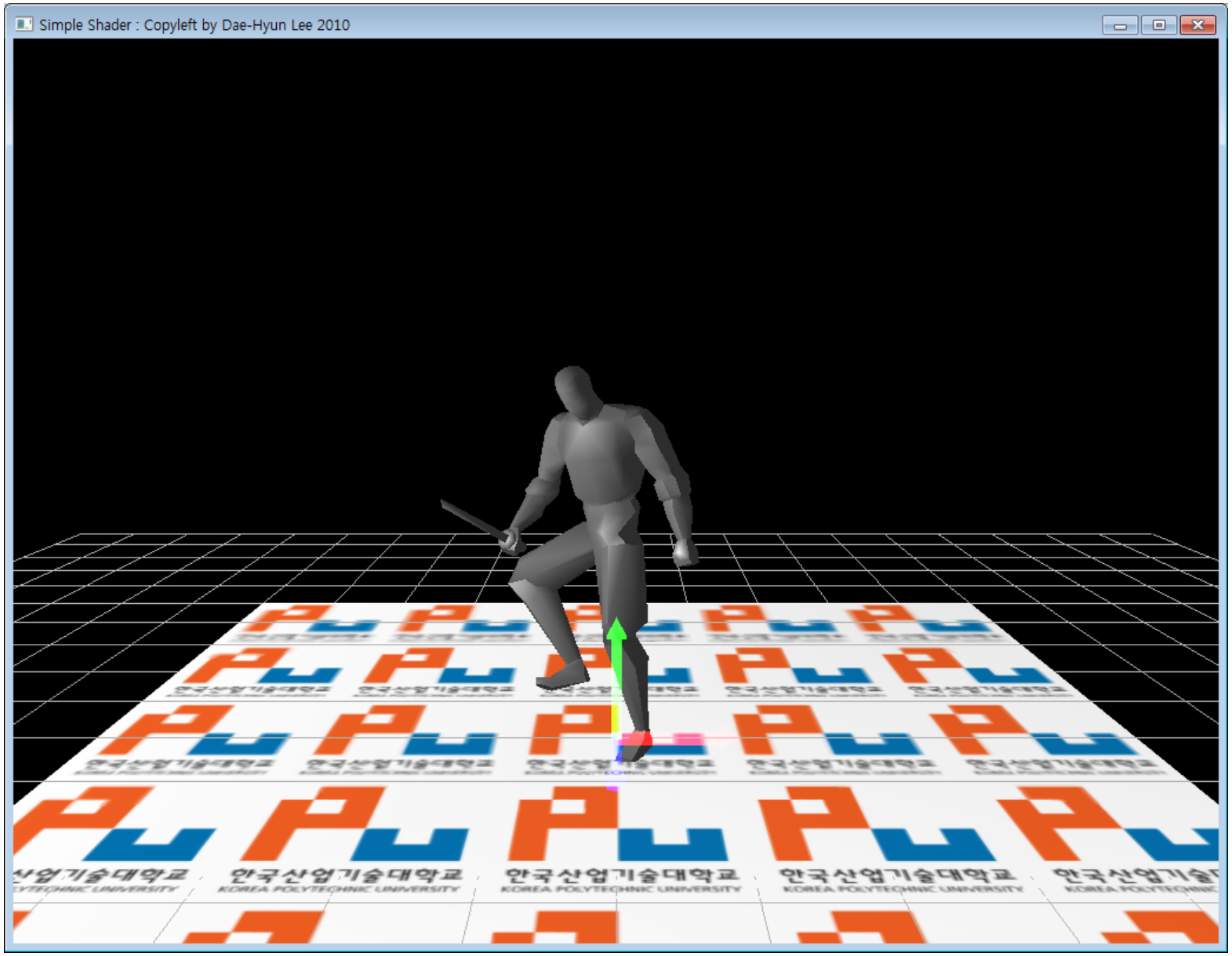
GouraudShading.cg (1)


```
void main_vp(   float4 position  : POSITION,
                float3 normal    : NORMAL,

                out float4 oPosition : POSITION,
                out float4 color     : COLOR,

                uniform float4x4 worldViewProj,
                uniform float4 globalAmbient,
                uniform float4 lightDiffuseColor,
                uniform float4 lightSpecularColor,
                uniform float3 lightPosition,
                uniform float3 eyePosition,
                uniform float4 Ke,
                uniform float4 Ka,
                uniform float4 Kd,
                uniform float4 Ks,
                uniform float  shininess)
{
    oPosition = mul(worldViewProj, position);

    float3 P = position.xyz;
    float3 N = normal;

    float4 emissive = Ke;
    float4 ambient = Ka * globalAmbient;
```

GouraudShading.cg(2)


```
    float3 L = normalize(lightPosition - P);
    float diffuseLight = max(dot(N, L), 0);
    float4 diffuse = Kd * lightDiffuseColor * diffuseLight;

    float3 V = normalize(eyePosition - P);
    float3 H = normalize(L + V);
    float specularLight = pow(max(dot(N, H), 0), shininess);
    if (diffuseLight <= 0) specularLight = 0;
    float4 specular = Ks * lightSpecularColor * specularLight;

    color = emissive + ambient + diffuse + specular;

}

void main_fp(float4 color : COLOR,
                out float4 oColor : COLOR)
{
    oColor = color;
}
```

실행 결과 - 고로 셰이딩



조명 모델 파라미터 전달

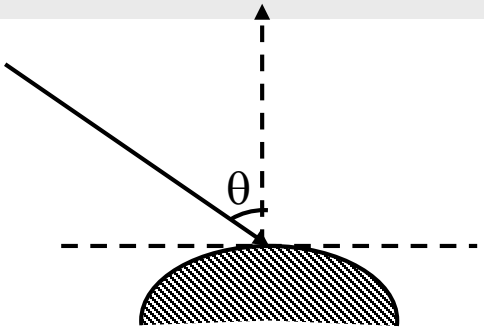
```
default_params
{
    param_named_auto worldViewProj worldviewproj_matrix
    param_named_auto globalAmbient ambient_light_colour
    param_named_auto lightDiffuseColor light_diffuse_colour 0
    param_named_auto lightSpecularColor light_specular_colour 0
    param_named_auto lightPosition light_position_object_space 0
    param_named_auto eyePosition camera_position_object_space

    param_named Ke float4 0.0 0.0 0.0 0.0
    param_named Ka float4 0.1745 0.01175 0.01175 0.55
    param_named Kd float4 0.61424 0.04136 0.04136 0.55
    param_named Ks float4 0.727811 0.626959 0.626959 0.55
    param_named shininess float 76.8
}
```

정점 셰이더

```
oPosition = mul(worldViewProj, position);

float3 P = position.xyz;
float3 N = normal;

float4 emissive = Ke;
float4 ambient = Ka * globalAmbient;

float3 L = normalize(lightPosition - P);
float diffuseLight = max(dot(N, L), 0);
float4 diffuse = Kd * lightDiffuseColor * diffuseLight;
```

$$I_a = K_a \otimes L_a$$

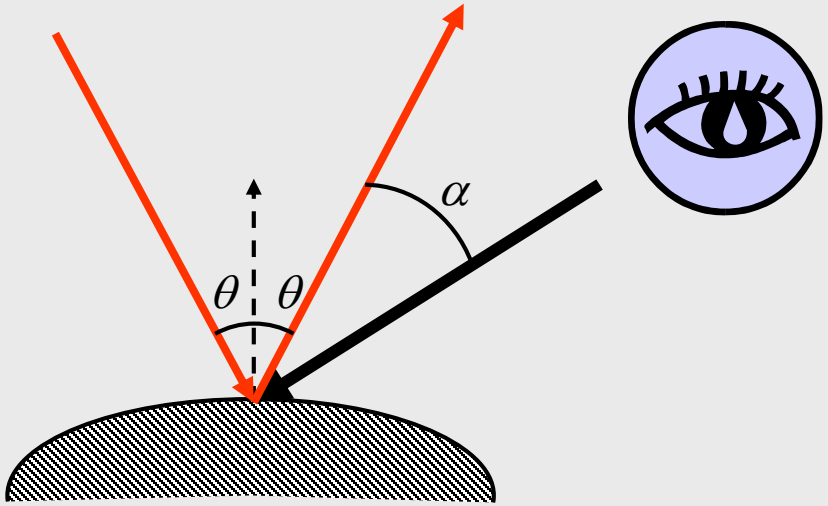
$$I_d = (K_d \otimes L_d)\cos\theta$$



정점 셰이더

```
float3 V = normalize(eyePosition - P);
float3 H = normalize(L + V);
float specularLight = pow(max(dot(N, H), 0), shininess);
if (diffuseLight <= 0) specularLight = 0;
float4 specular = Ks * lightSpecularColor * specularLight;
```

$$I_s = (K_s \otimes L_s)(\cos\alpha)^n$$




```
color = emissive + ambient + diffuse + specular;

oTexCoord = texCoord;
```

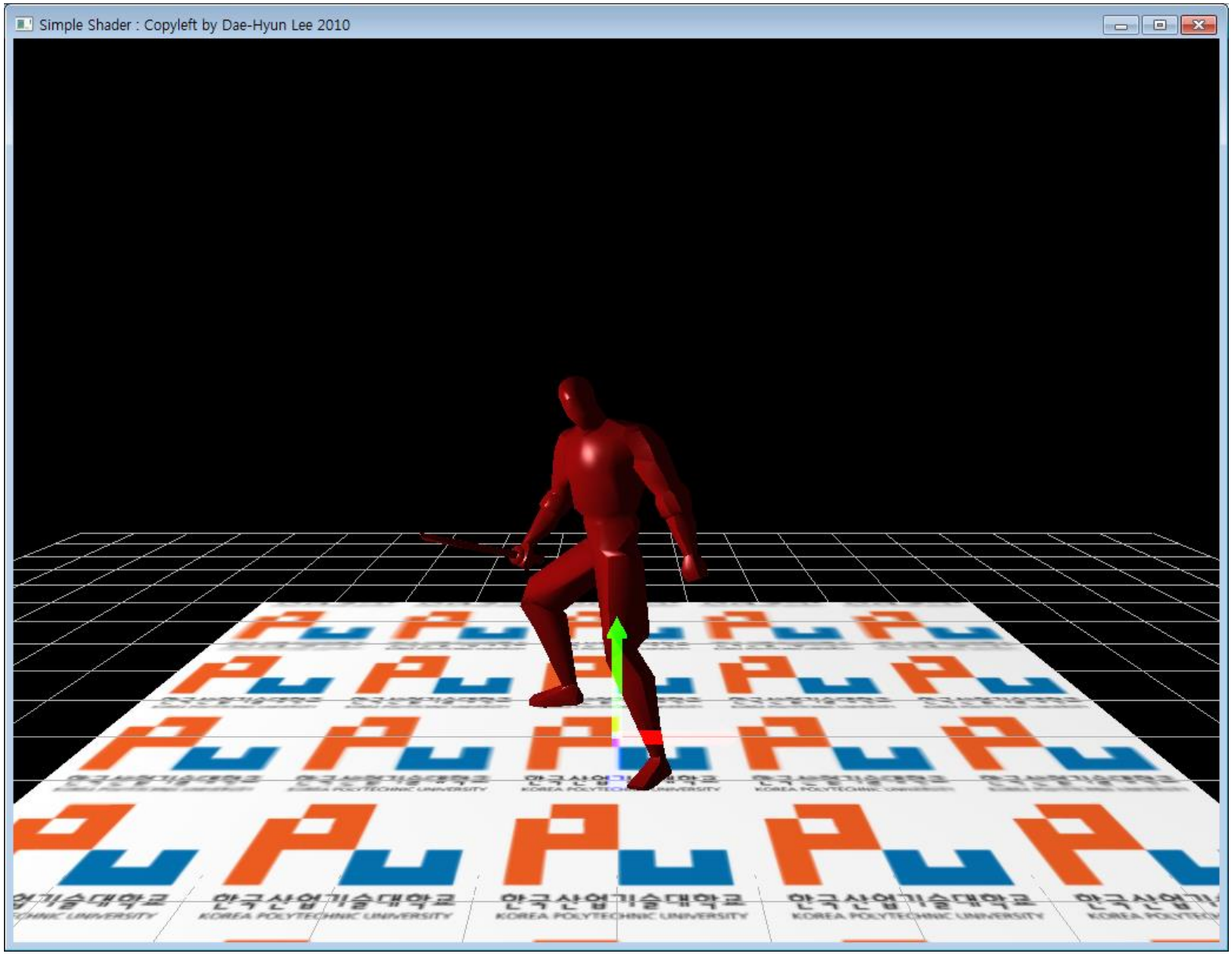
PhongShading.cg (1)


```
void PhongShadingMainVP(   float4 position  : POSITION,
                float3 normal    : NORMAL,

                out float4 oPosition : POSITION,
                out float3 oObjectPos : TEXCOORD0,
                out float3 oNormal : TEXCOORD1,

                uniform float4x4 worldViewProj)
{
   oPosition = mul(worldViewProj, position);
   oObjectPos = position.xyz;
   oNormal = normal;
}

void PhongShadingMainFP (    float3 position  : TEXCOORD0,
                float3 normal    : TEXCOORD1,
                out float4 color     : COLOR,
                uniform float4 globalAmbient,
                uniform float4 lightDiffuseColor,
                uniform float4 lightSpecularColor,
                uniform float3 lightPosition,
                uniform float3 eyePosition,
                uniform float4 Ke,
                uniform float4 Ka,
                uniform float4 Kd,
                uniform float4 Ks,
                uniform float  shininess)
```

PhongShading.cg (2)


```
{
    float3 P = position;
    float3 N = normalize(normal);

    float4 emissive = Ke;

    float4 ambient = Ka * globalAmbient;


    float3 L = normalize(lightPosition - P);
    float diffuseLight = max(dot(N, L), 0);
    float4 diffuse = Kd * lightDiffuseColor * diffuseLight;


    float3 V = normalize(eyePosition - P);
    float3 H = normalize(L + V);
    float specularLight = pow(max(dot(N, H), 0), shininess);
    if (diffuseLight <= 0) specularLight = 0;
    float4 specular = Ks * lightSpecularColor * specularLight;

    color = (emissive + ambient + diffuse + specular);
}
```

실행 결과 - 퐁 셰이딩 효과



정점 및 노말 벡터의 전달

```
void PhongShadingMainVP (   float4 position  : POSITION,
                float3 normal    : NORMAL,

                out float4 oPosition : POSITION,
                out float3 oObjectPos : TEXCOORD0,
                out float3 oNormal : TEXCOORD1,

                uniform float4x4 worldViewProj)
{
   oPosition = mul(worldViewProj, position);
   oObjectPos = position.xyz;
   oNormal = normal;
}

void PhongShadingMainFP (    float3 position  : TEXCOORD0,
                float3 normal    : TEXCOORD1,
                out float4 color     : COLOR, … 생략…)
{
   float3 P = position;
   float3 N = normalize(normal);
```

PlayState.cpp

```
bool PlayState::keyPressed(GameManager* game, const FrameEvent& evt)
{
  … 중략 …

 case KC_O: mNinjaEntity->setMaterialName("Examples/Ninja"); break;
 case KC_P: mNinjaEntity->setMaterialName("PhongShading"); break;
 case KC_T: mNinjaEntity->setMaterialName("PhongShadingDecal"); break;

… 후략 …
}
```

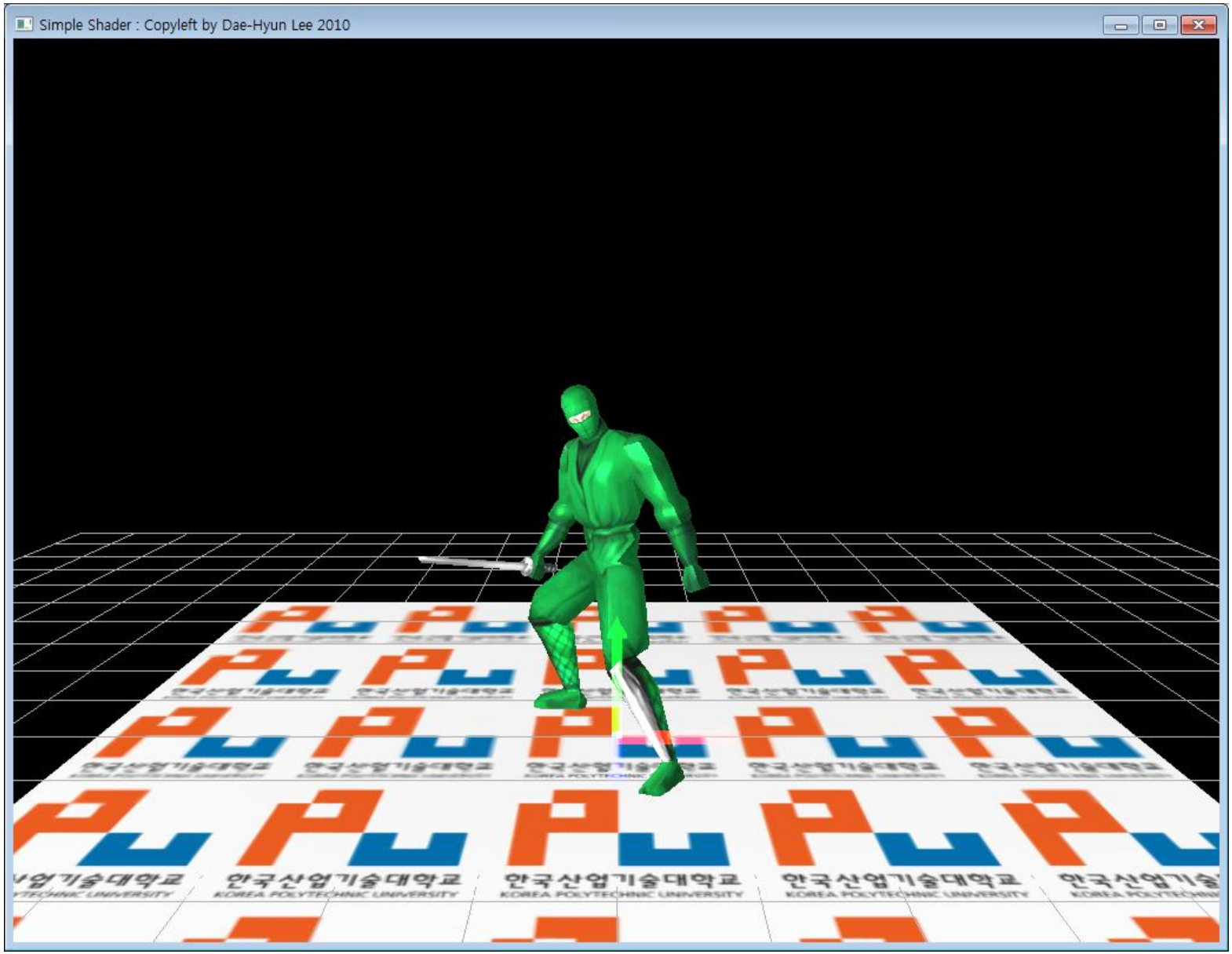
데칼 텍스쳐 표현 - PhongShading.cg (1)


```
void PhongShadingDecalMainVP (   float4 position  : POSITION,
                  float3 normal    : NORMAL,
                  float2 texCoord : TEXCOORD0,

                  out float4 oPosition : POSITION,
                  out float3 oObjectPos : TEXCOORD0,
                  out float3 oNormal : TEXCOORD1,
                      out float2 oTexCoord : TEXCOORD2,

                  uniform float4x4 worldViewProj)
{
   oPosition = mul(worldViewProj, position);
   oObjectPos = position.xyz;
   oNormal = normal;
    oTexCoord = texCoord;
}
```

데칼 텍스쳐 표현 - PhongShading.cg (2)


```
void PhongShadingDecalMainFP (    float3 position  : TEXCOORD0,
                  float3 normal    : TEXCOORD1,
                      float2 texCoord : TEXCOORD2,
                  out float4 color     : COLOR,
                  uniform float4 globalAmbient,
                  uniform float4 lightDiffuseColor,
                  uniform float4 lightSpecularColor,
                  uniform float3 lightPosition,
                  uniform float3 eyePosition,
                  uniform float  shininess,
                      uniform sampler2D ninjaTexture)
{
   float3 P = position;  float3 N = normalize(normal);
   float4 ambient = globalAmbient;
   float3 L = normalize(lightPosition - P);
   float diffuseLight = max(dot(N, L), 0);
   float4 diffuse = lightDiffuseColor * diffuseLight;

   float3 V = normalize(eyePosition - P);
   float3 H = normalize(L + V);
   float specularLight = pow(max(dot(N, H), 0), shininess);
   if (diffuseLight <= 0) specularLight = 0;
   float4 specular = lightSpecularColor * specularLight;

      color = (ambient + diffuse + specular) * tex2D(ninjaTexture, texCoord);
}
```

실행 결과


Simple Shader : Copyleft by Dae-Hyun Lee 2010
한국산업기술대학교
KOREA POLYTECHNIC UNIVERSITY

텍스쳐 좌표의 전달

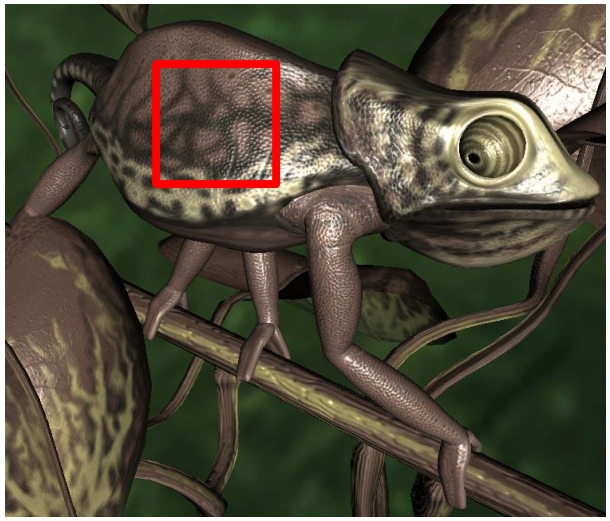
```
void PhongShadingDecalMainVP (   float4 position  : POSITION,
                  float3 normal    : NORMAL,
                  float2 texCoord : TEXCOORD0,

                  out float4 oPosition : POSITION,
                  out float3 oObjectPos : TEXCOORD0,
                  out float3 oNormal : TEXCOORD1,
                  out float2 oTexCoord : TEXCOORD2,

                  uniform float4x4 worldViewProj)
{
   oPosition = mul(worldViewProj, position);
   oObjectPos = position.xyz;
   oNormal = normal;
   oTexCoord = texCoord;
}
```

데칼 텍스쳐 결합

```
void PhongShadingDecalMainFP (... 중략 ...
                  float2 texCoord : TEXCOORD2,
                  ... 중략 ...
                  uniform sampler2D ninjaTexture)
{
   float3 P = position;
   float3 N = normalize(normal);

   float4 ambient = globalAmbient;
   float3 L = normalize(lightPosition - P);
   float diffuseLight = max(dot(N, L), 0);
   float4 diffuse = lightDiffuseColor * diffuseLight;

   float3 V = normalize(eyePosition - P);
   float3 H = normalize(L + V);
   float specularLight = pow(max(dot(N, H), 0), shininess);
   if (diffuseLight <= 0) specularLight = 0;
   float4 specular = lightSpecularColor * specularLight;

   color = (ambient + diffuse) * tex2D(ninjaTexture, texCoord) + specular;
}
```

이런 멋진 모델을 렌더링하려면?

범프란?



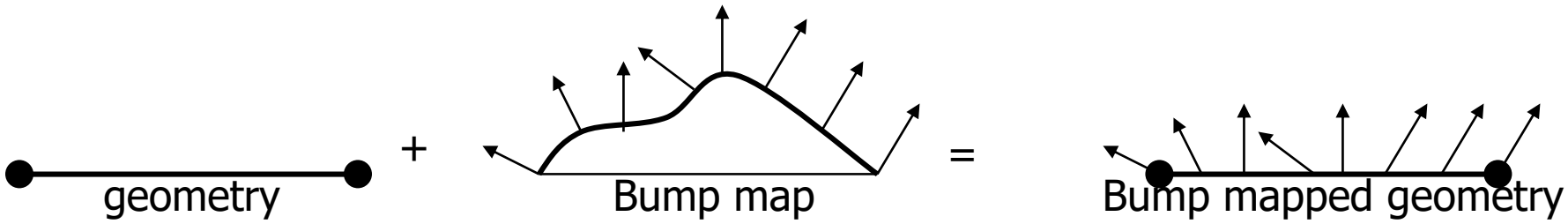
bump [bʌmp] PLAY 단어장에 추가

—verb (bumped, bumping)
1. *tr & intr* (*especially* bump into or against something or someone) to knock or hit them or it, especially heavily or with a jolt.
2. to hurt or damage (eg one's head) by hitting or knocking it.
3. (*usually* bump together) *intrans* said of two moving objects: to collide.

범프를 3D 메쉬로 일일이 모델링? → 엄청난 GPU 부하가 걸림.

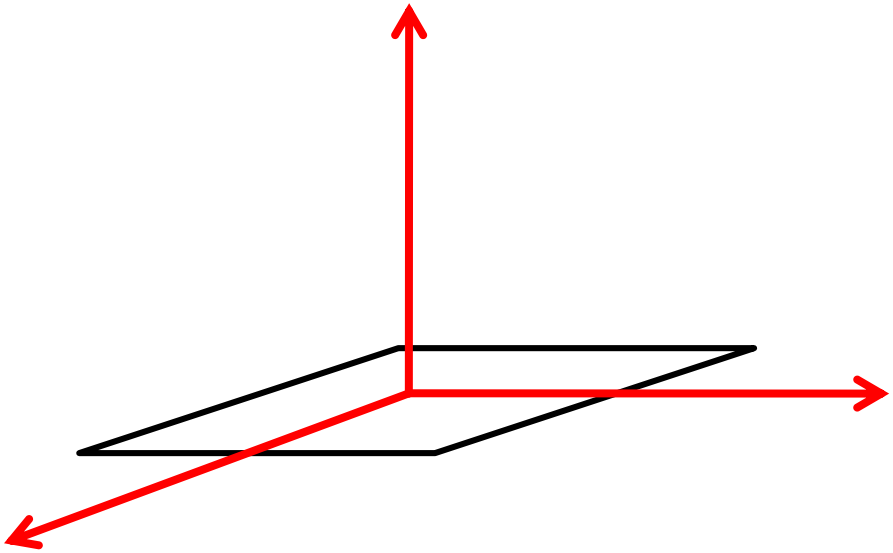
범프 매핑

- 1978년 블린이 고안.
- 표면의 주름이라든가 범프 등을 렌더링하는 효율적인 방법.
- 텍스쳐를 이용하여, 표면의 법선 정보를 저장하고, 이를 이용하여 색상을 계산.



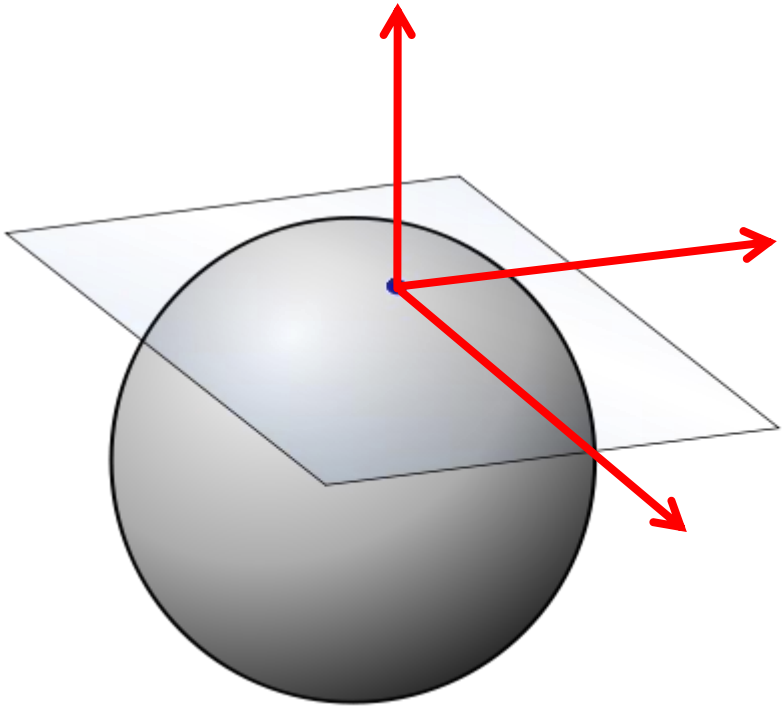
범프 맵(Bump Map)

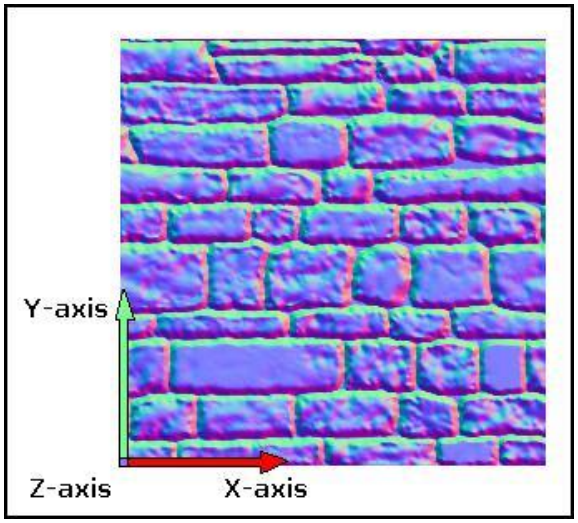
- **표면의 기하학적인 정보(그 중 가장 중요한 법선 벡터)를 텍스쳐 맵으로 표현한 것.**
  - 법선 벡터 (x,y,z) → 색상 (R,G,B)
  - x,y,z 값의 범위 [-1, 1] → 색상값의 범위 [0, 1]
  - 기준 좌표계: 객체공간(Object Space) 또는 접선공간(Tangent Space)



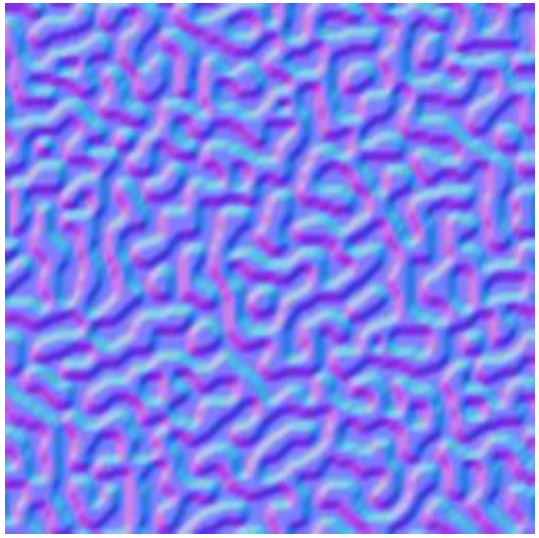
접선 공간(Tangent Space)

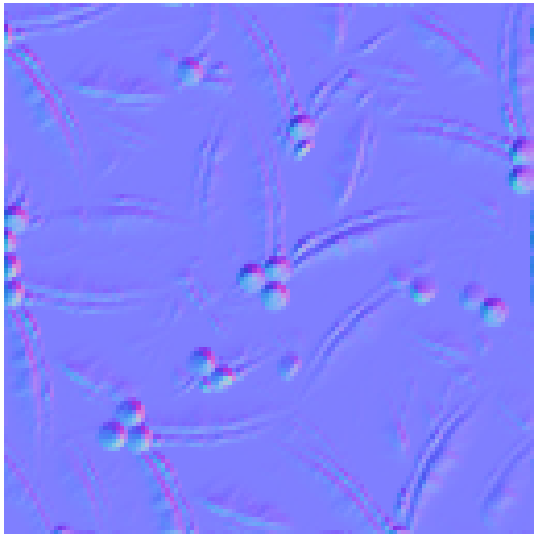
- 범프맵에 저장된 normal을 사용하여 조명 계산하기 위한 공간
- Vertex의 normal을 Z 축으로 하는 공간
- 정점 공간(Vertex Space)라고도 함.
- 일반적으로 tangent (X), binormal (Y), normal (Z)을 미리 계산하여 정점 정보로 사용.

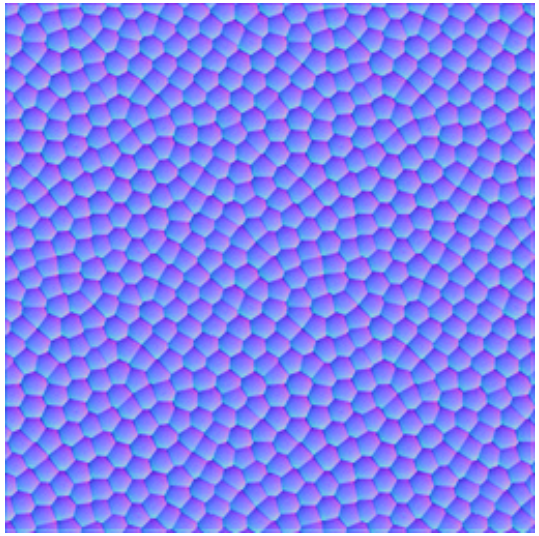


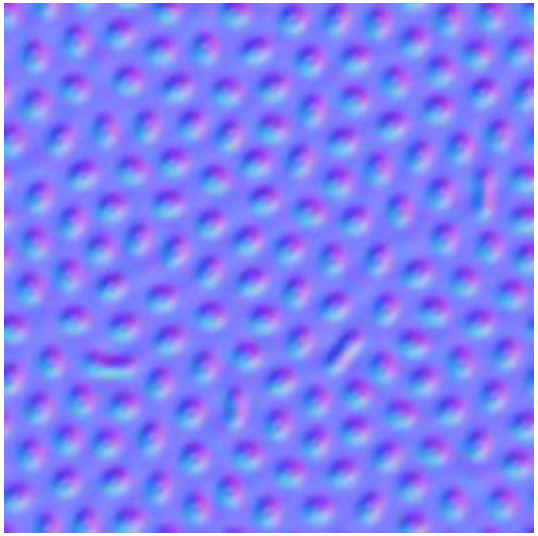


실제 범프 맵들



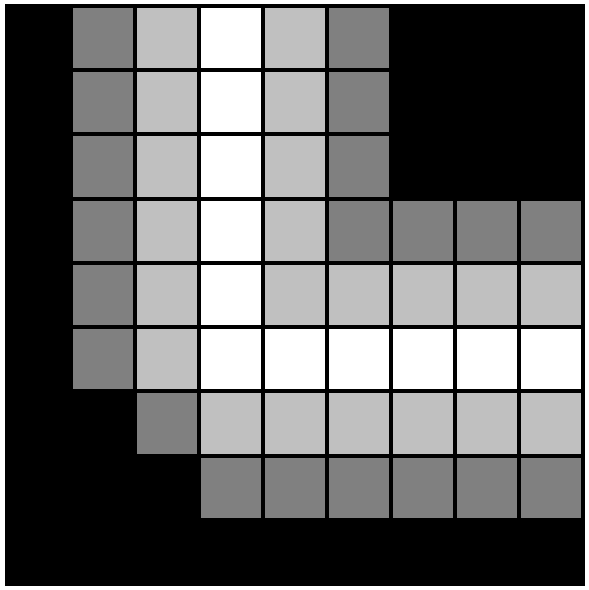


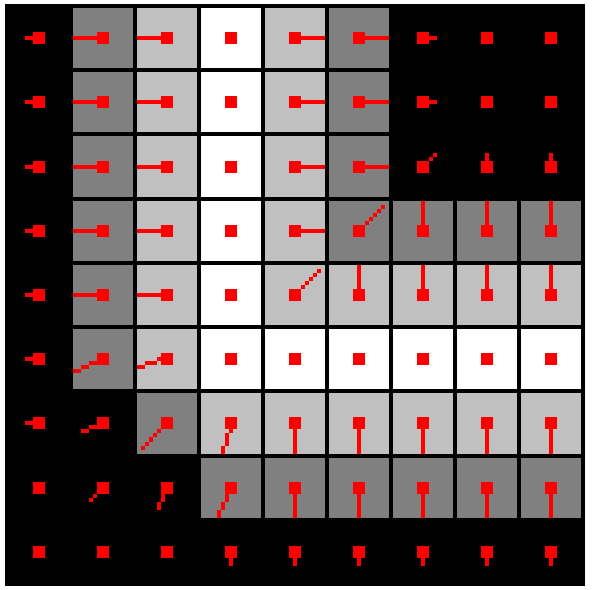




범프 맵의 생성

■높이맵으로부터 범프 맵을 생성할 수 있음.

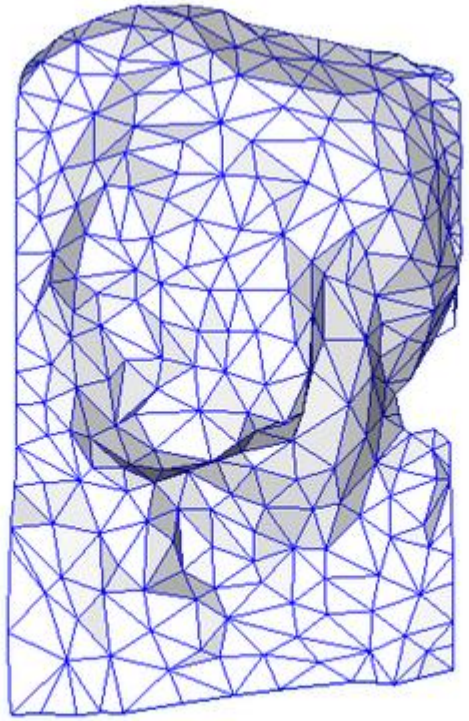




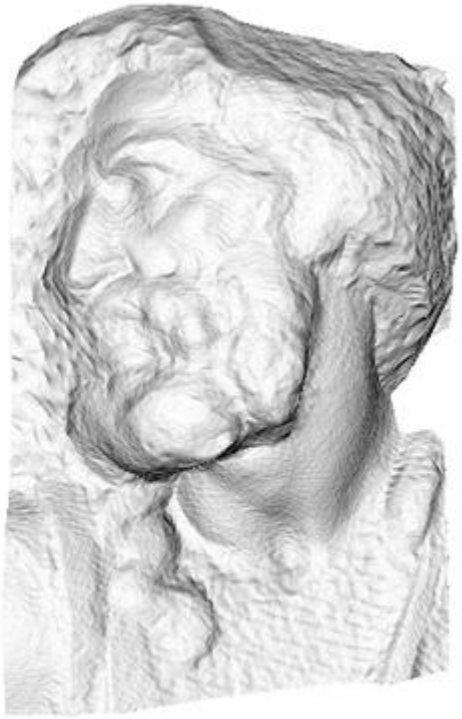
노말 매핑(Normal Mapping)

- 범프매핑의 효율성을 이용하여, 아예 기하모델링을 노말 맵으로 처리.
- 복잡하고 정교한 3D 모델의 폴리곤 개수를 대폭 줄일 수 있음.

original mesh
4M triangles



simplified mesh
500 triangles



simplified mesh
and normal mapping
500 triangles