Advanced Lighting (1)

GPU Programming

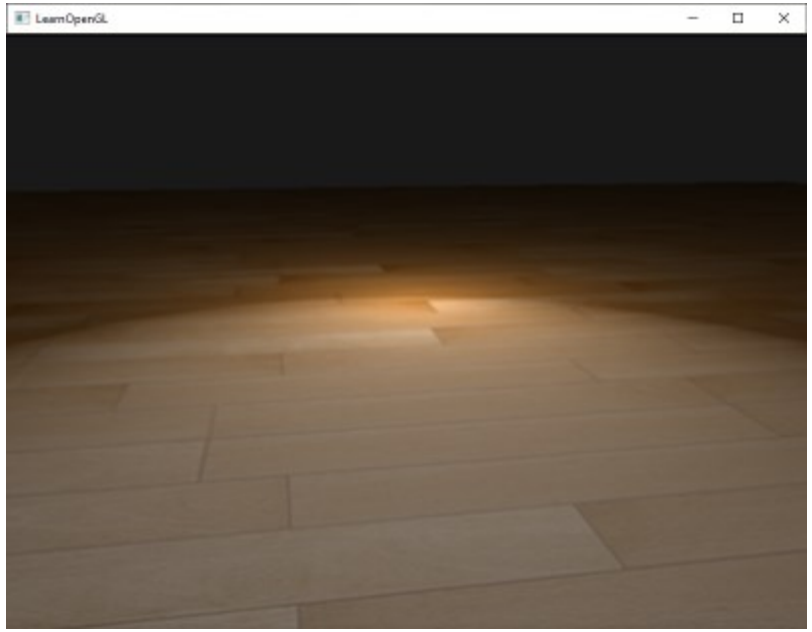
나재호

ⓒ nvidia

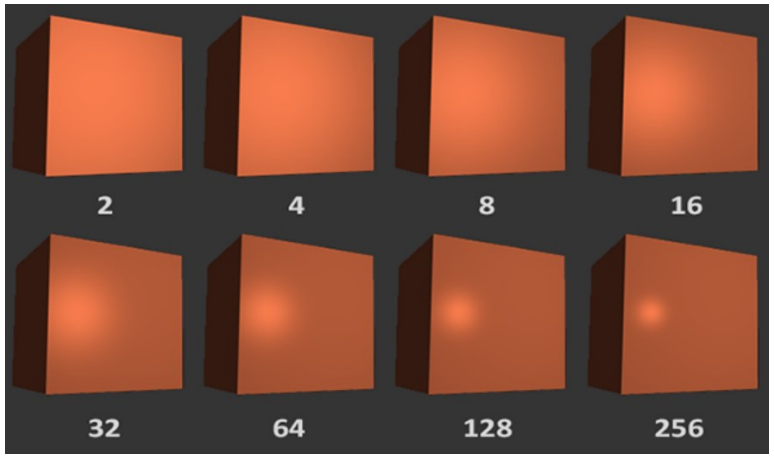
Blinn-Phong

Blinn-Phong

- Phong lighting의 단점
  - 정반사(specular reflections)가 shininess가 낮은 특정 조건에서 제대로 표현되지 않는 경우가 있음
  - 오른쪽과 같이 평평한 텍스처된 면에서,
    1.0의 shininess 지수 사용시 (아주 rough한 material)
    가장자리에서 반사 영역이 끊어짐
  - View 벡터와 반사 벡터 사이의 각도가 90도 이상일 경우,
    내적의 결과가 음수가 되기 때문에 이를 0으로 보정하기 때문

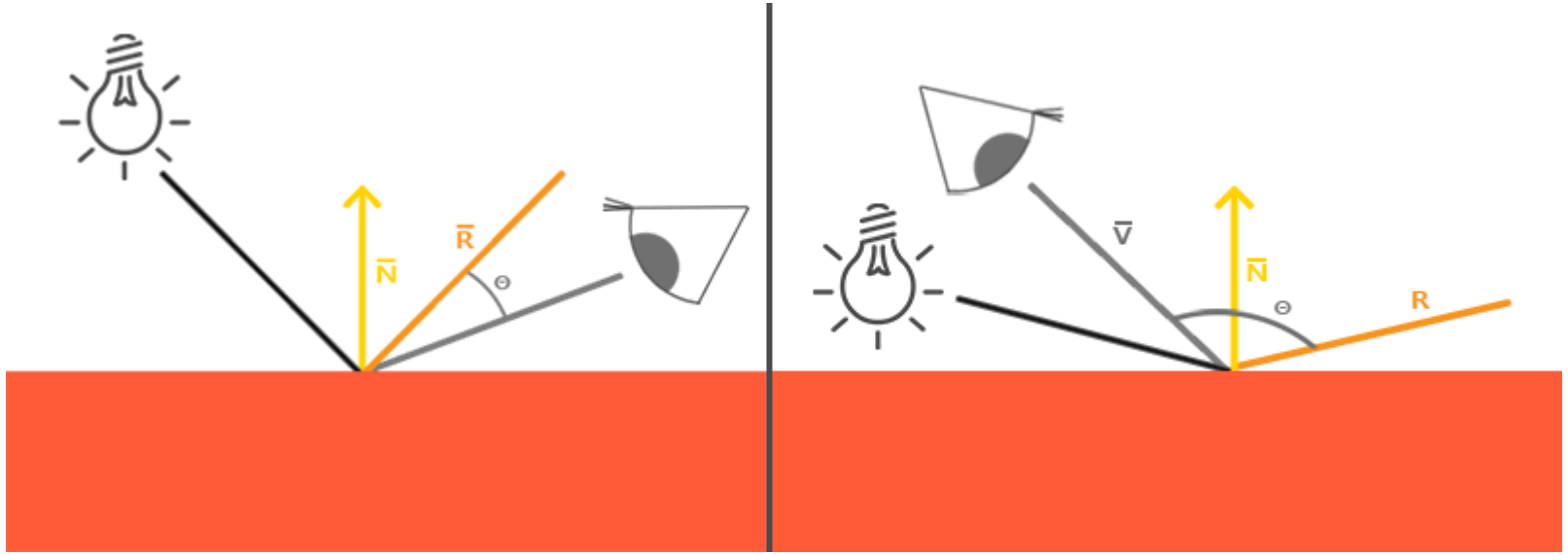



```
float spec = pow(max(dot(viewDir, reflectDir), 0.0), 32);
vec3 specular = specularStrength * spec * lightColor;
```



Blinn-Phong

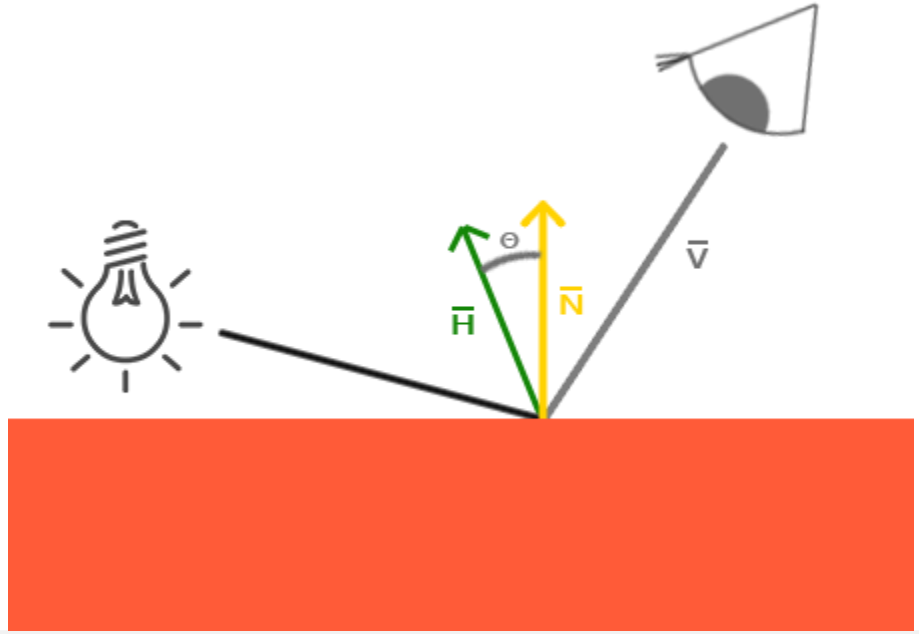
- Phong lighting의 단점 (cont.)
  - 하지만 벡터 V와 벡터 R 사이의 각(θ) 가 90도를 넘어가는 경우에도 반사 기여도는 반영되어야 함



Blinn-Phong

- Blinn-Phong shading model
  - 1977년 James F. Blinn이 Phong shading을 확장하여 제안한 방법
  - Specular lighting 계산시, view의 방향과 light의 방향 중간에 있는 단위 벡터인 halfway 벡터 (H) 사용
  - 이 벡터 H가 노멀 벡터 N과 일치할수록 specular 기여도가 높아지게 됨
  - Phong shading과 마찬가지로,
    view 방향이 기존 반사 벡터 R과 일치하면 specular 기여도가 최고가 됨
  - 벡터 H와 벡터 N간의 각도는 절대 90도를 넘지 않음

$$\bar{H} = \frac{\bar{L} + \bar{V}}{||\bar{L} + \bar{V}||}$$



Blinn-Phong

- GLSL 구현

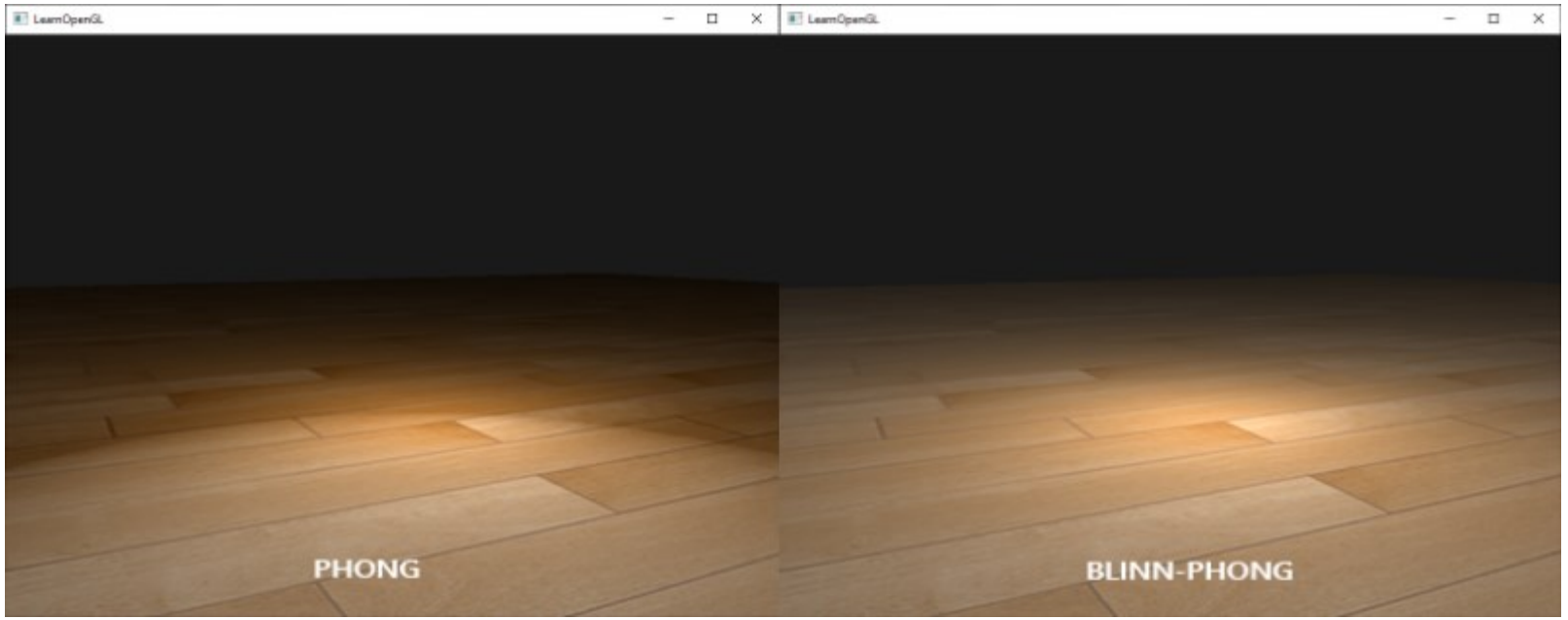
```
vec3 lightDir   = normalize(lightPos - FragPos);
vec3 viewDir    = normalize(viewPos - FragPos);
vec3 halfwayDir = normalize(lightDir + viewDir);
```

FS

```
float spec = pow(max(dot(normal, halfwayDir), 0.0), shininess);
vec3 specular = lightColor * spec;
```

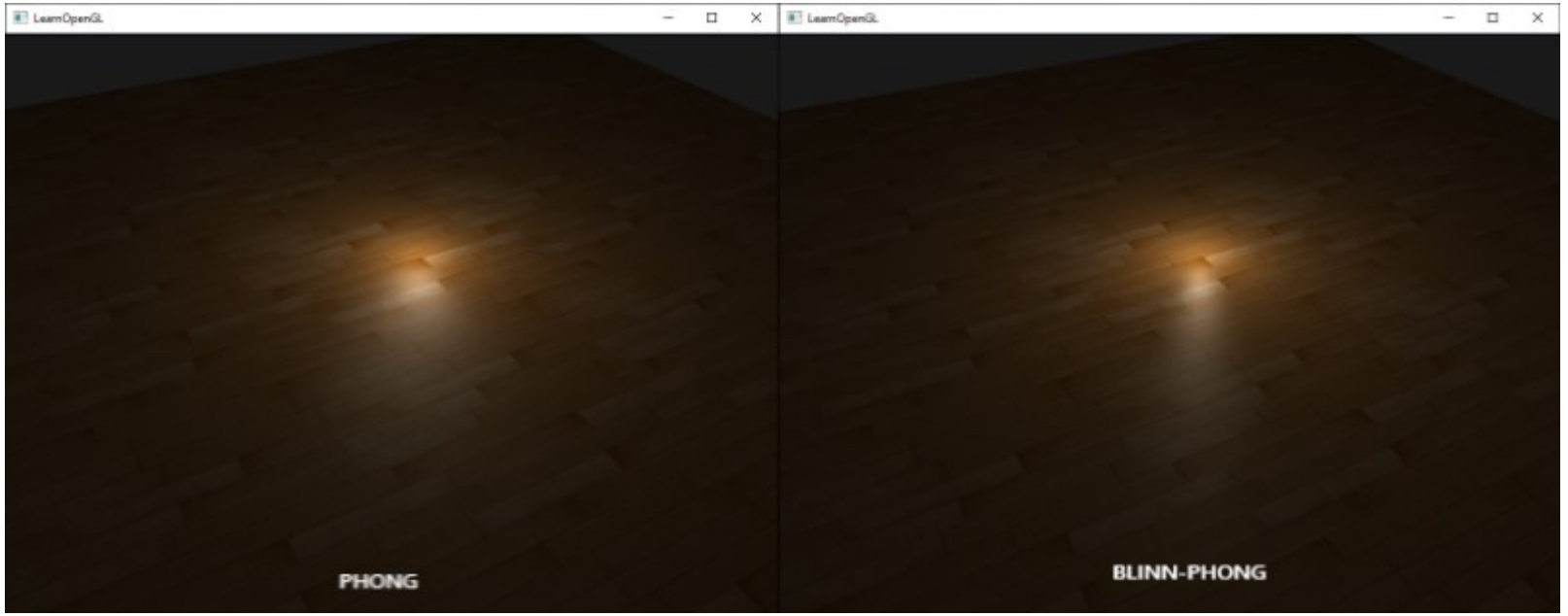
$$\bar{H} = \frac{\bar{L} + \bar{V}}{||\bar{L} + \bar{V}||}$$

- 결과 화면 비교
  - Specular exponent는 0.5



Blinn-Phong

- Blinn-Phong의 H와 N 벡터 사이의 각도는 Phong의 V와 R 벡터 사이 각도보다 짧음
  - 일반적으로 specular exponent값을 Phong의 2~4배로 설정하면 이러한 차이를 보정 가능
  - Phong과 Blinn-Phong의 exponent값을 각각 8, 32으로 설정한 예



Blinn-Phong

- Blinn-Phong과 Phong 전환 데모를 위한 shader 코드

FS

```
void main()
{
    [...]
    float spec = 0.0;
    if(blinn)
    {
        vec3 halfwayDir = normalize(lightDir + viewDir);
        spec = pow(max(dot(normal, halfwayDir), 0.0), 16.0);
    }
    else
    {
        vec3 reflectDir = reflect(-lightDir, normal);
        spec = pow(max(dot(viewDir, reflectDir), 0.0), 8.0);
    }
```

ⓒ nvidia

Gamma Correction

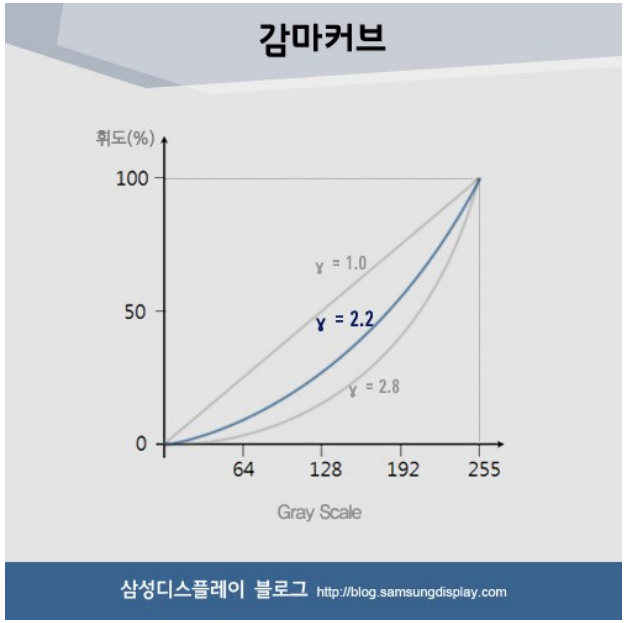
Gamma Correction

- 예전 CRT 모니터에서는 입력 전압의 세기와 휘도가 비례하지 않았음
  - 자승(gamma값)이 2.2인 것과 유사한 지수 관계를 가짐
- 현재의 디스플레이는 종류별로 각기 다른 감마 특성을 가지나, 대략적으로 2.2 내외를 가짐 (sRGB 색상 공간과 유사)
  - 중·저계조 영역에서 출력되는 휘도의 변화가 작음
  - 고계조로 갈수록 Gray Level에 따른 휘도의 변화가 큼
- 사람 눈도 마찬가지로 이러한 비선형성을 띔

Perceived (linear) brightness =	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0
Physical (linear) brightness =	0.0	0.1	0.2	0.3	0.4	0.5	0.6	0.7	0.8	0.9	1.0

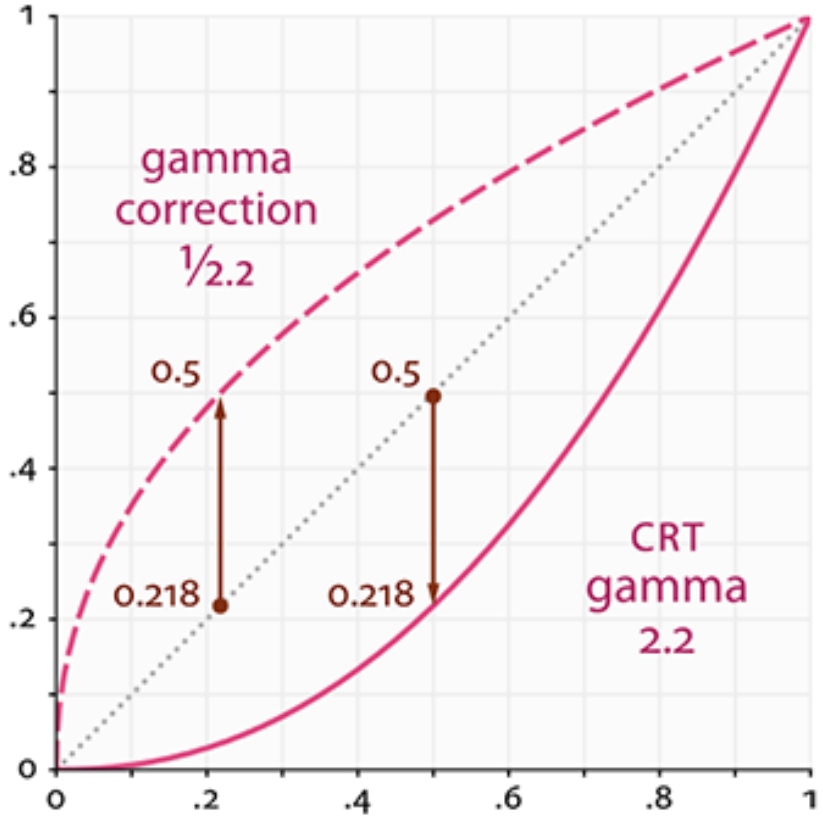
- 이러한 비선형적인 특성을 선형으로 바꾸기 위해 감마 보정을 사용

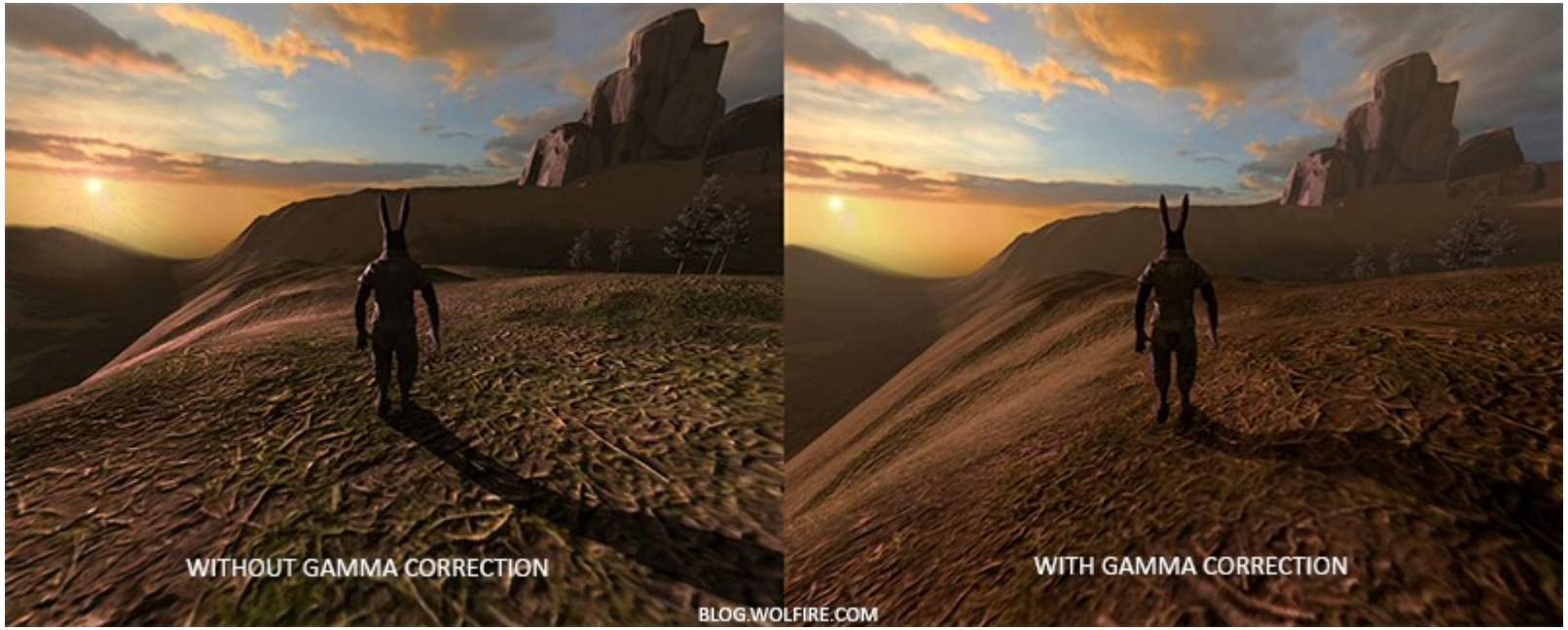
[디스플레이 상식] 감마에 대해 알아봅시다! (samsungdisplay.com)



Gamma Correction

- 감마 보정 방법
  - 모니터 gamma값의 역을 최종 출력 색상에 적용한 후 이를 모니터에 출력
  - 이러한 과정을 통해, lighting 계산을 선형 공간에서 수행하여 사실적이고 보기 좋은 결과를 만들 수 있음





Gamma Correction

- OpenGL에 내장된 sRGB 프레임버퍼를 사용하여 감마 보정을 사용하는 방법
  - 최종적으로 디스플레이에 출력하기 전에 하드웨어에서 감마 보정 수행

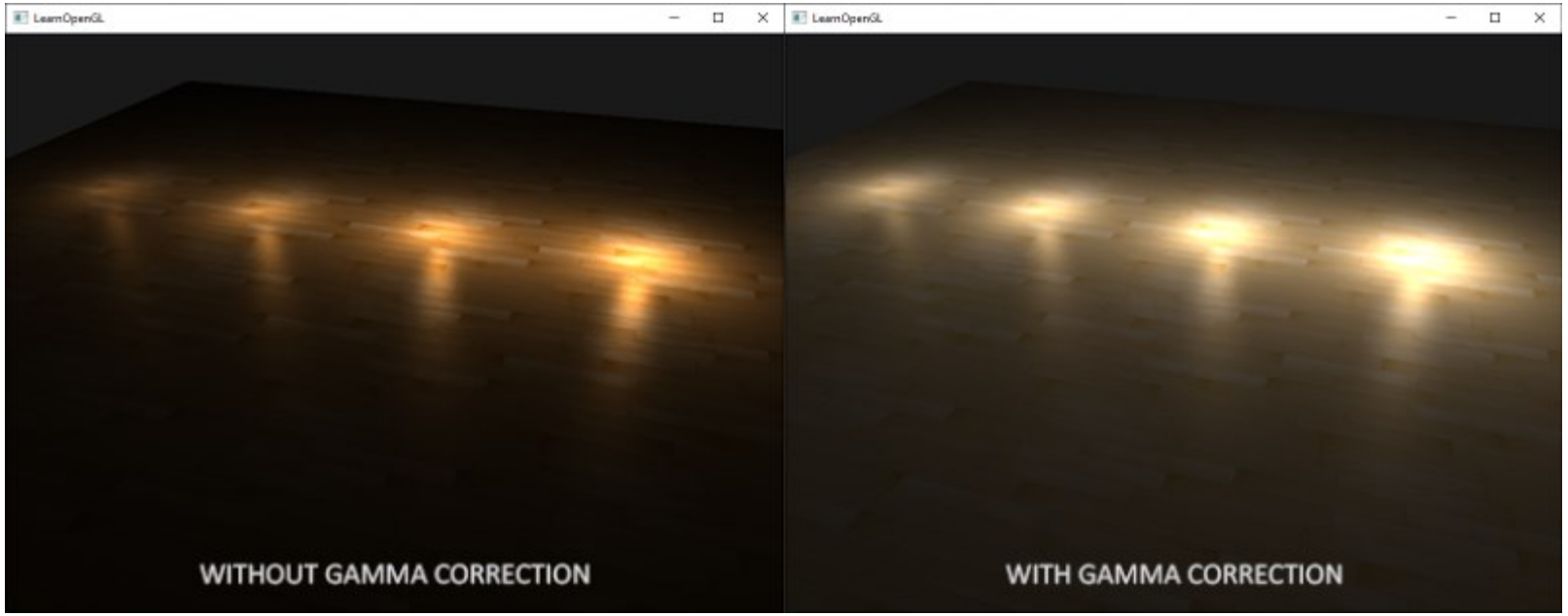
```
glEnable(GL_FRAMEBUFFER_SRGB);
```
CPU

- Fragment shader에서 직접 감마 보정 수행
  - 감마값을 자유롭게 바꿀 수 있음
  - 각 FS별로 감마 보정 코드를 아래와 같이 추가하거나, 별도 post-processing 단계를 통해 감마 보정 가능

FS
```
void main()
{
    // do super fancy lighting in linear space
    [...]
    // apply gamma correction
    float gamma = 2.2;
    FragColor.rgb = pow(fragColor.rgb, vec3(1.0/gamma));
}
```

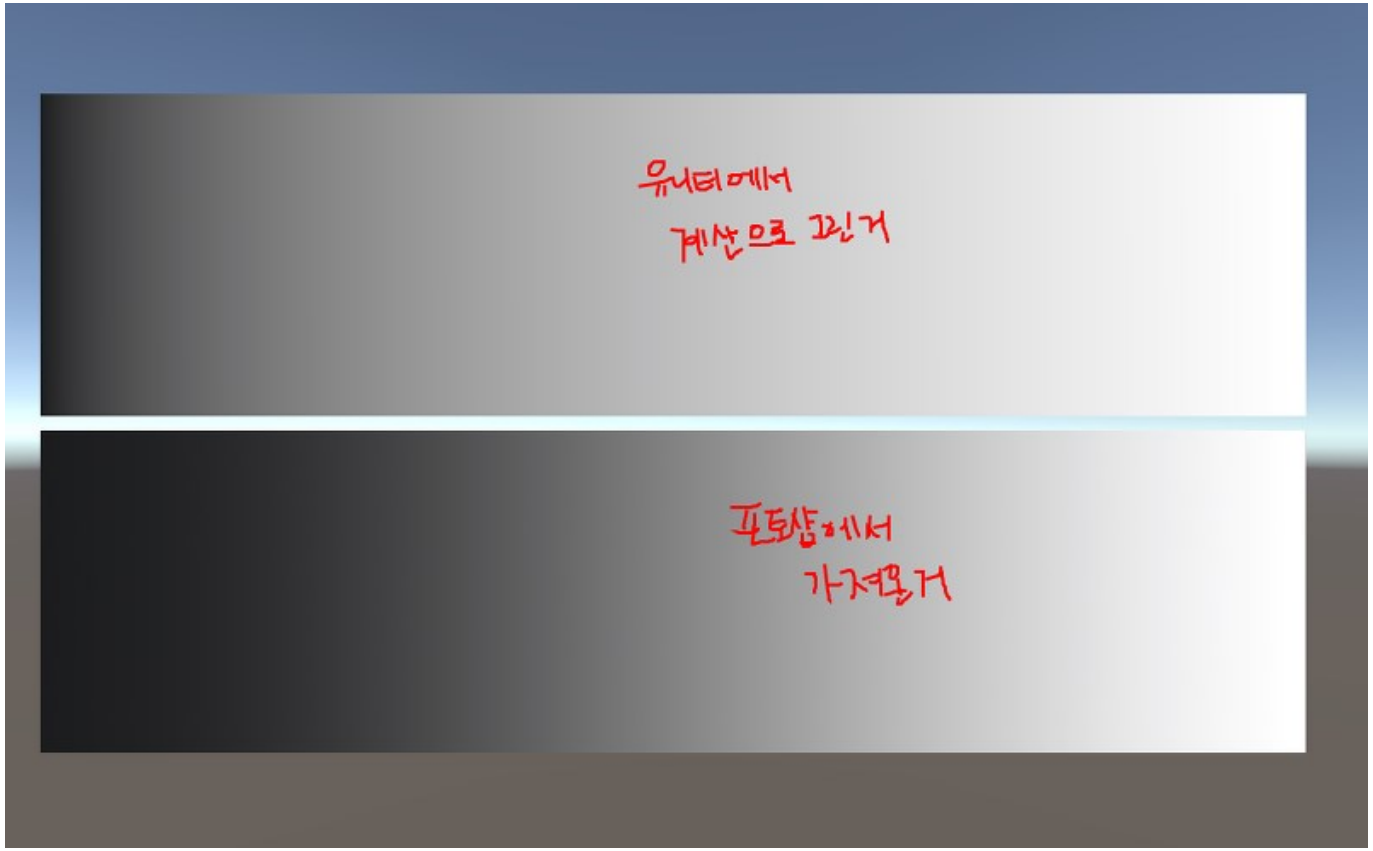
sRGB Texture

- 컴퓨터에서 그림을 그리거나 편집할 때, 모니터에는 감마가 적용된 색이 표시
  - 이 때 고르는 색상이 선형 공간이 아니라 sRGB 공간에 있음을 의미
- 텍스처 아티스트가 똑같은 sRGB 공간에서 디자인을 하면,
  같은 sRGB 색상 공간 상에서 텍스처 매핑이 수행되므로 문제 없음
- 그러나 이 텍스처를 감마 보정한 후 선형 공간에 표시하면 아래와 같이 전체적으로 밝아짐
  (모니터에서 감마 보정된 이미지로 작업을 한 후, renderer에서 감마 보정을 또 수행하기 때문)



sRGB Texture

- 앞선 문제를 방지하는 첫번째 방법: 아티스트가 선형 공간에서 텍스처 디자인을 수행
  - 직관적이지 않으므로 선호되는 방식은 아님



감마 코렉션 : 감마 보정 Gamma Correction 에 대한 텍스쳐 옵션 정리 (tistory.com)

sRGB Texture

- 앞선 문제를 방지하는 두번째 방법: 셰이더에서 sRGB 텍스처를 선형 공간으로 다시 수정

```
float gamma = 2.2;
vec3 diffuseColor = pow(texture(diffuse, texCoords).rgb, vec3(gamma));
```
FS

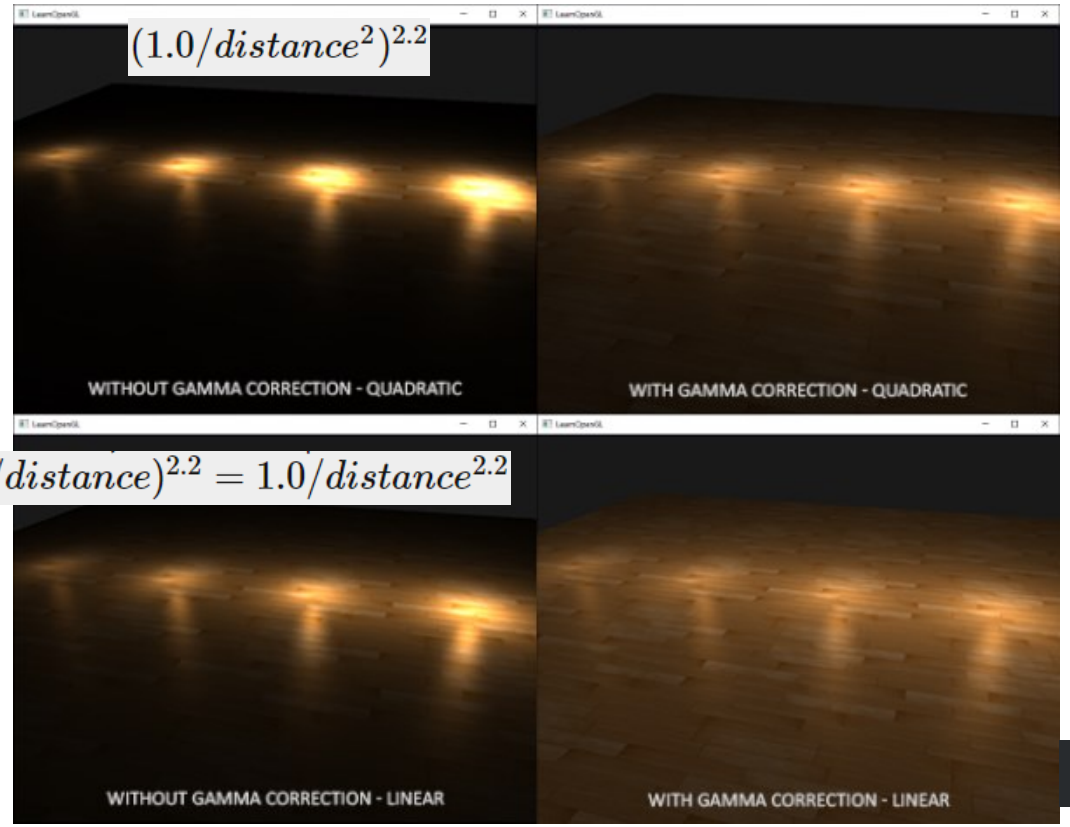
- 앞선 문제를 방지하는 세번째 방법: GL_SRGB 또는 GL_SRGB_ALPHA를 텍스처의 내부 포맷으로 지정

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

- 텍스처의 종류에 따라 sRGB 또는 선형 공간을 사용할 수 있음에 유의
  - 물체에 색을 입히는 diffuse texture는 sRGB 공간에,
    Lighting을 위해 쓰이는 specular 및 normal maps는 선형 공간에 있는 경우가 대부분

Attenuation

- Gamma correction은 attenuation(감쇠)에도 영향을 끼침
  - 감마 보정 미적용시, attenuation에서 밝기를 변경한 후 이를 선형 공간에서 scene을 시각화하지 않음
    → 물리적으로 정확하지 않은 linear attenuation이 더 그럴듯하게 나옴
  - 감마 보정 적용시에는 물리적으로 보다 정확한 quadratic attenuation이 더 나은 결과를 냄




```
float attenuation = 1.0 / (distance * distance);
```

```
float attenuation = 1.0 / distance;
```

Gamma Correction

- 감마 보정의 필요성
  - 물리적 세계에서 의미가 있는 선형 공간에서 모든 셰이더/조명 계산을 수행 가능
  - Lighting에 사용되는 대부분의 물리 방정식이 좀 더 실제로 좋은 결과(예: 빛 감쇠 등)를 제공
  - 감마 보정 후 lighting 파라미터를 조정하는 것을 권장

ⓒ nvidia

마무리

마무리

- Advanced lighting의 첫번째 시간으로, 아래와 같은 내용을 살펴보았습니다.
  - Blinn-Phong
  - Gamma correction

- 다음 실습 시간에는 간단하게 예제 코드를 설명하고 실행해 볼 예정입니다.
  - LearnOpenGL 5.2