

게임엔진

제14강 광원과 조명

한국산업기술대학교 이대현



학습 안내

■ 학습 목표

- 오우거 엔진의 광원을 이용하여 3D 공간에서 광원을 구현해본다.
- 조명 모델을 이해하고, 조명 방정식을 구성해본다.

■ 학습 내용

- 평면 메시의 생성 방법
- 광원의 종류 및 구현 방법
- 조명 모델과 조명 방정식

광원의 종류: 주변광원

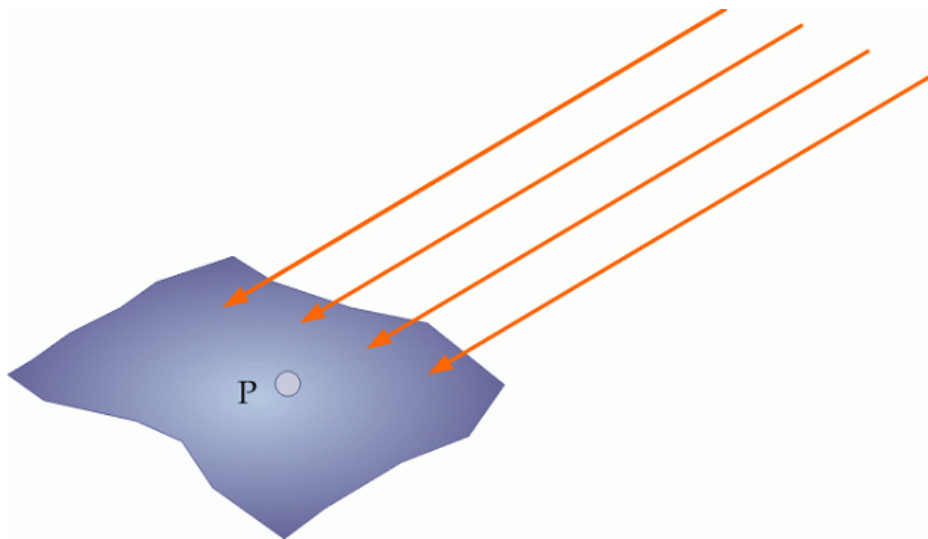
■ 주변 광원(Ambient Light)

- 동일한 밝기의 빛이 장면 안의 모든 물체의 표면에서 일정하게 반사되는 것.
- 공간 안에 존재하는 빛의 평균값
- 이론적인 광원

광원의 종류: 지향 광원

■지향 광원(Directional Light)

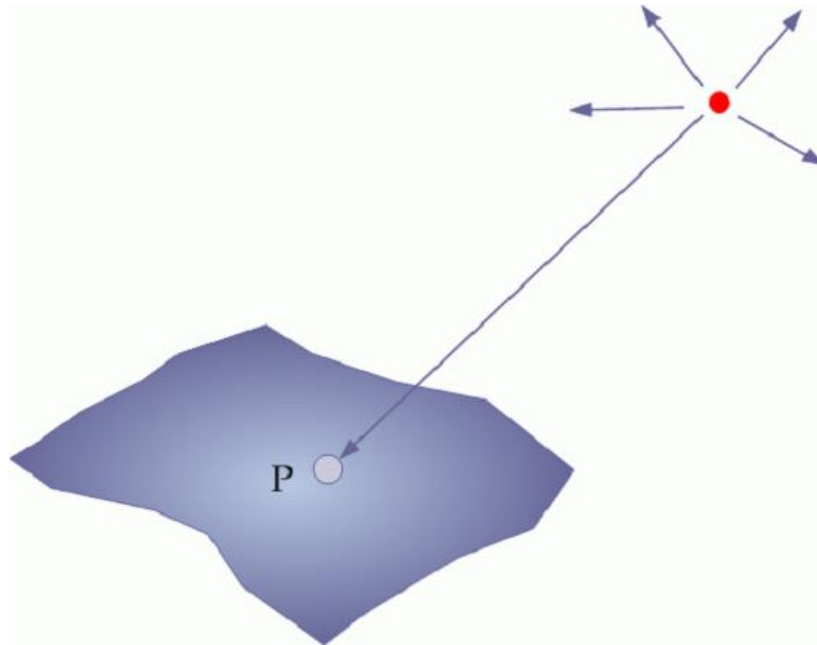
- 한 방향으로 무한히 뻗어나가는 빛.
- 빛이 물체면을 향하여 일정한 방향으로 진행.
- 거리에 상관없이 특정한 한 방향(벡터)에 대해서 빛의 세기가 일정하게 주어진다 → 방향이 중요
- 태양을 흉내낼 때 주로 쓰임.
- OGRE 엔진: LT_DIRECTIONAL



광원의 종류: 점 광원

■점 광원(Point Light)

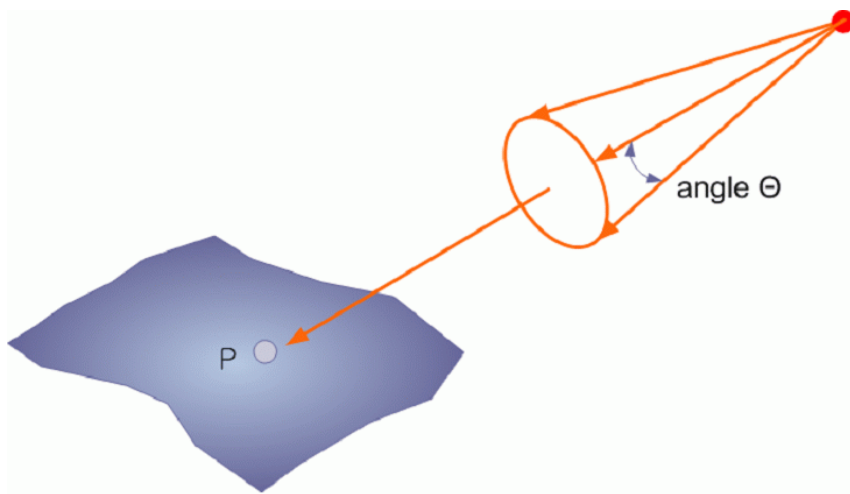
- 공간 안의 한 점에서 모든 방향으로 동일하게 뿔어나가는 빛.
- 백열 전구를 모델링.
- 광원과 물체 표면과의 거리의 제곱에 비례하여 밝기가 약해짐(감쇄: attenuation) → 거리가 중요
- OGRE 엔진: LT_POINT



광원의 종류: 점적 광원

■점적 광원(Spot Light)

- 정해진 위치와 범위만 비추는 광원.
- 일종의 점광원이지만, 모든 방향으로만 퍼지는 것이 아니고, 특정 방향으로 지정된 각도만큼 빛이 퍼져나감.
- 무대 조명을 모델링.
- 거리에 따라서 빛의 세기가 약해짐.
- OGRE 엔진: LT_SPOTLIGHT



실습



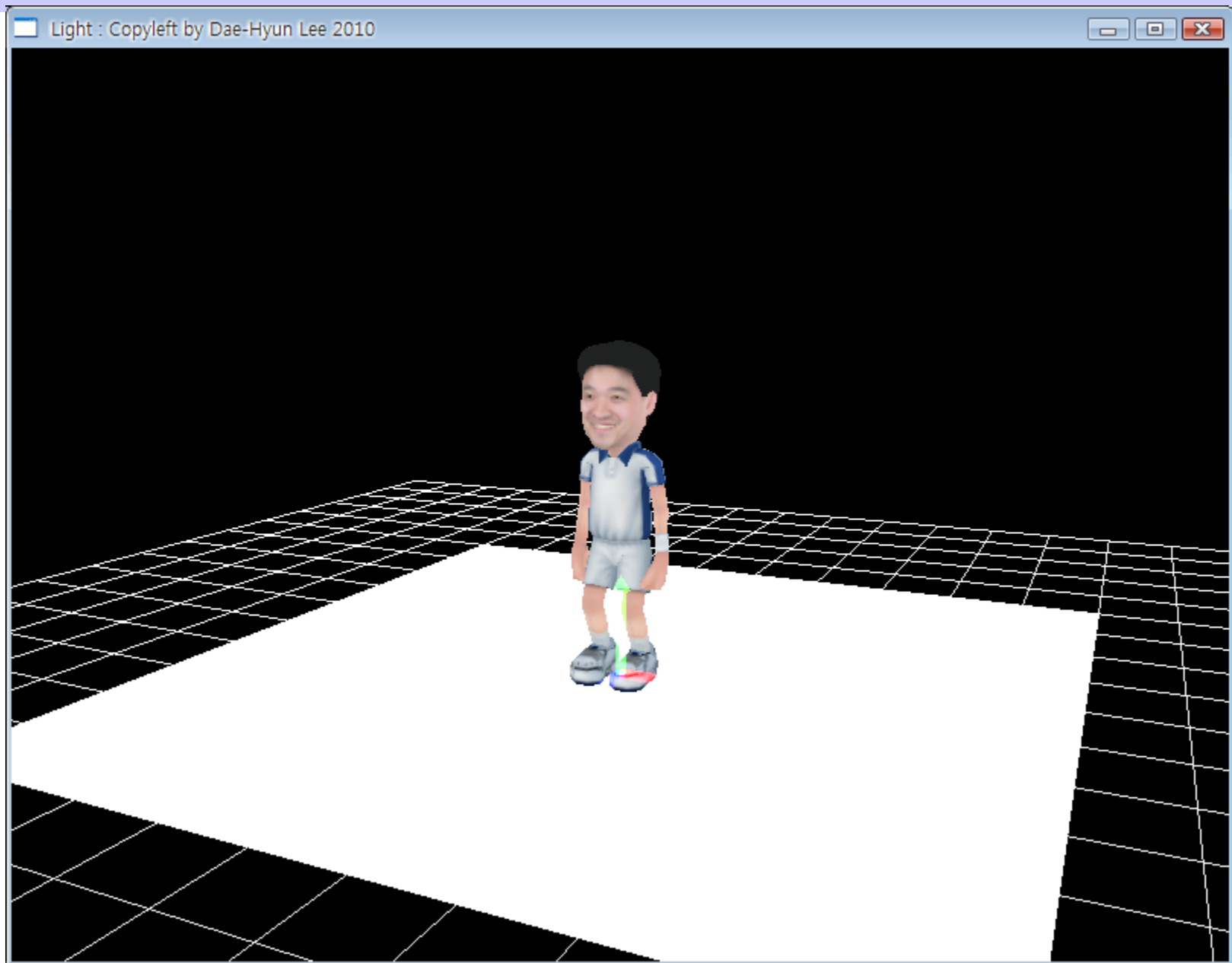
Light
다양한 광원 설정



```
void _drawGroundPlane(void)
{
    Plane plane( Vector3::UNIT_Y, 0 );
    MeshManager::getSingleton().createPlane(
        "Ground",
        ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME,
        plane,
        500,500,
        1,1,
        true,1,2,2,
        Vector3::NEGATIVE_UNIT_Z
    );

    Entity* groundEntity = mSceneMgr->createEntity("GroundPlane", "Ground" );
    mSceneMgr->getRootSceneNode()->createChildSceneNode()->attachObject(groundEntity);
}
```


실행 결과



평면 메쉬의 생성

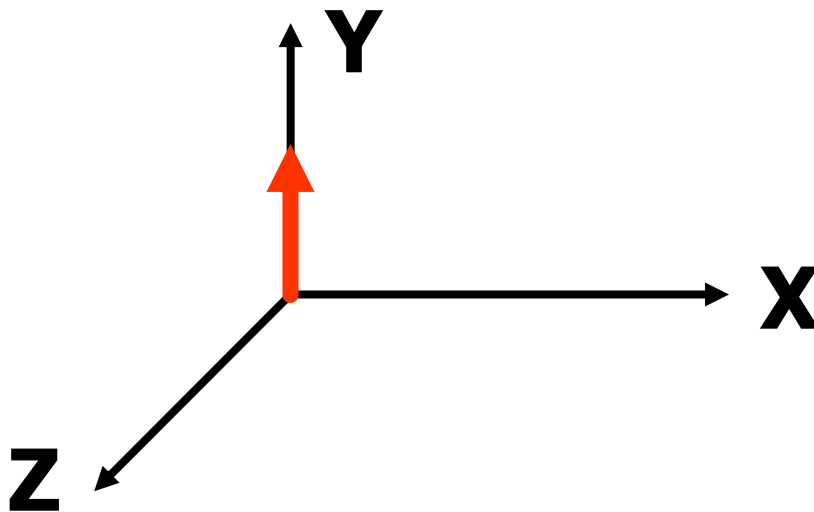
■ 생성 방법

- 먼저 평면을 생성하고(Plane 클래스 사용), 그리고 이것을 메쉬로 변환한다(MeshManager 클래스 사용)

■ 평면의 생성

- 두개의 정보가 필요: 평면의 법선 벡터(normal vector) 및 평면과 원점의 거리

```
Plane plane(Vector3::UNIT_Y, 0);
```



■ 메쉬의 생성

□ MeshManager 클래스

- 프로그램에서 로드된 모든 메쉬들을 관리하는 클래스

• 메쉬 이름.

```
MeshManager::getSingleton().createPlane(  
    "Ground",  
    ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME,  
    plane,  
    500,500,  
    1,1,  
    true,1,2,2,  
    Vector3::NEGATIVE_UNIT_Z  
);
```

• 평면 정보.

• 너비: 500
• 높이: 500

■엔터티와 장면노드의 생성

- 앞에서 만든 메시의 이름.

```
Entity* groundEntity = mSceneMgr->createEntity( "GroundPlane", "Ground" );  
mSceneMgr->getRootSceneNode()->createChildSceneNode()->attachObject(groundEntity);
```

- 장면노드의 생성

- 장면노드에 엔터티 배치.



```
void _drawGroundPlane(void)
{
    Plane plane( Vector3::UNIT_Y, 0 );
    MeshManager::getSingleton().createPlane(
        "Ground",
        ResourceGroupManager::DEFAULT_RESOURCE_GROUP_NAME,
        plane,
        500,500,
        1,1,
        true,1,10,10,
        Vector3::NEGATIVE_UNIT_Z
    );

    Entity* groundEntity = mSceneMgr->createEntity("GroundPlane", "Ground" );
    mSceneMgr->getRootSceneNode()->createChildSceneNode()->attachObject(groundEntity);
    groundEntity->setMaterialName("KPU_LOGO");
}
```

■ 텍스처 입히기

- `setMaterialName()` 함수 사용

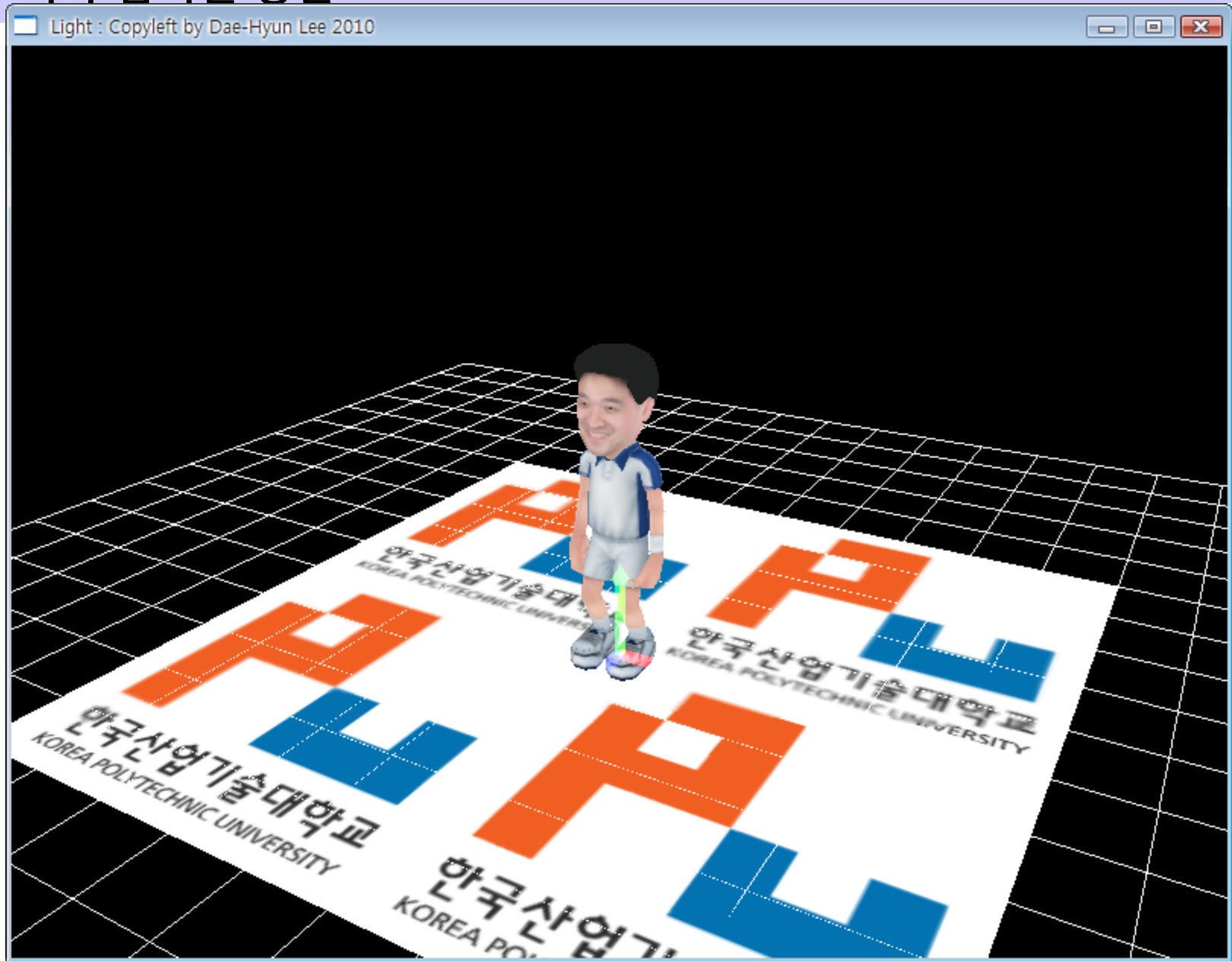
```
Entity* groundEntity = mSceneMgr->createEntity("GroundPlane", "Ground" );  
mSceneMgr->getRootSceneNode()->createChildSceneNode()->attachObject(groundEntity);  
groundEntity->setMaterialName("KPU_LOGO");
```

```
material KPU_LOGO  
{  
    technique  
    {  
        pass  
        {  
            texture_unit  
            {  
                texture KPU_LOGO.gif  
            }  
        }  
    }  
}
```

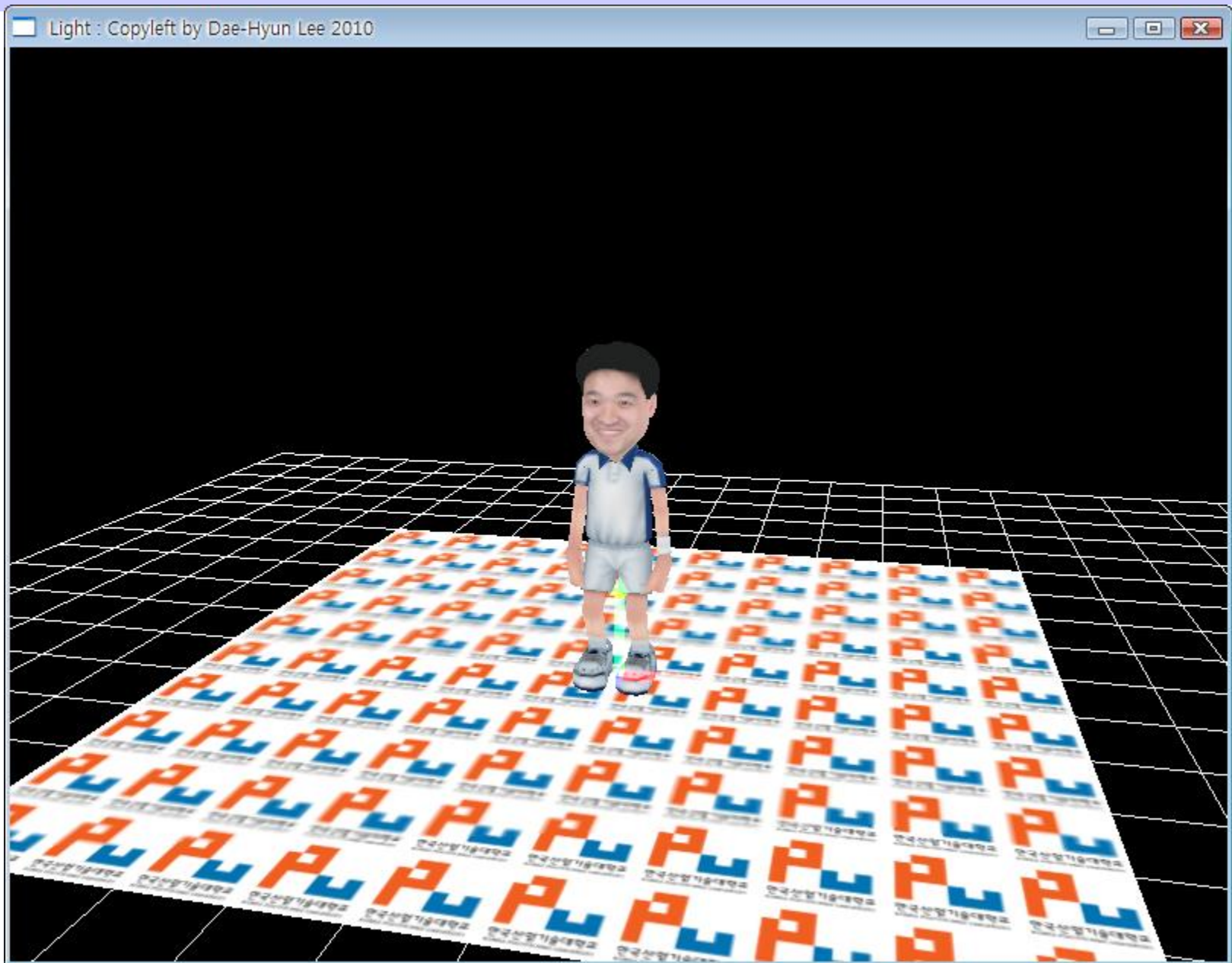
- *resource.zip* 내의 *KPU_LOGO.material* 에 위치.
- *KPU_LOGO.gif* 를 타일 이미지로 사용.



텍스처가 입혀진 평면



타일 갯수의 변화(10x10)





```
void _setLights(void)
{
    mSceneMgr->setAmbientLight(ColourValue(0.0f, 0.0f, 0.0f));

    mLightD = mSceneMgr->createLight("LightD");
    mLightD->setType(Light::LT_DIRECTIONAL);
    mLightD->setDirection( Vector3( 1, -2.0f, -1 ) );
    mLightD->setVisible(false);

    mLightP = mSceneMgr->createLight("LightP");
    mLightP->setType( Light::LT_POINT );
    mLightP->setPosition( Vector3(-250, 50, 250) );
    mLightP->setVisible(false);

    mLightS = mSceneMgr->createLight("LightS");
    mLightS->setType( Light::LT_SPOTLIGHT );
    mLightS->setDirection(Ogre::Vector3::NEGATIVE_UNIT_Y);
    mLightS->setPosition( Vector3( 250, 900, 250) );
    mLightS->setSpotlightRange( Degree(10), Degree(80));
    mLightS->setVisible(false);
}
```



```
bool keyPressed( const OIS::KeyEvent &evt )
{
    switch(evt.key)
    {
        case OIS::KC_A:
        {
            static float a = 0.0f;
            a = (a >= 1.0f) ? 0.0f : a + 0.1f;
            mSceneMgr->setAmbientLight(ColourValue(a, a, a));
        }
        break;
        case OIS::KC_D: mLightD->setVisible(!mLightD->getVisible()); break;
        case OIS::KC_P: mLightP->setVisible(!mLightP->getVisible()); break;
        case OIS::KC_S: mLightS->setVisible(!mLightS->getVisible()); break;
    }

    return true;
}
```

PlayState.cpp - 그림자 추가

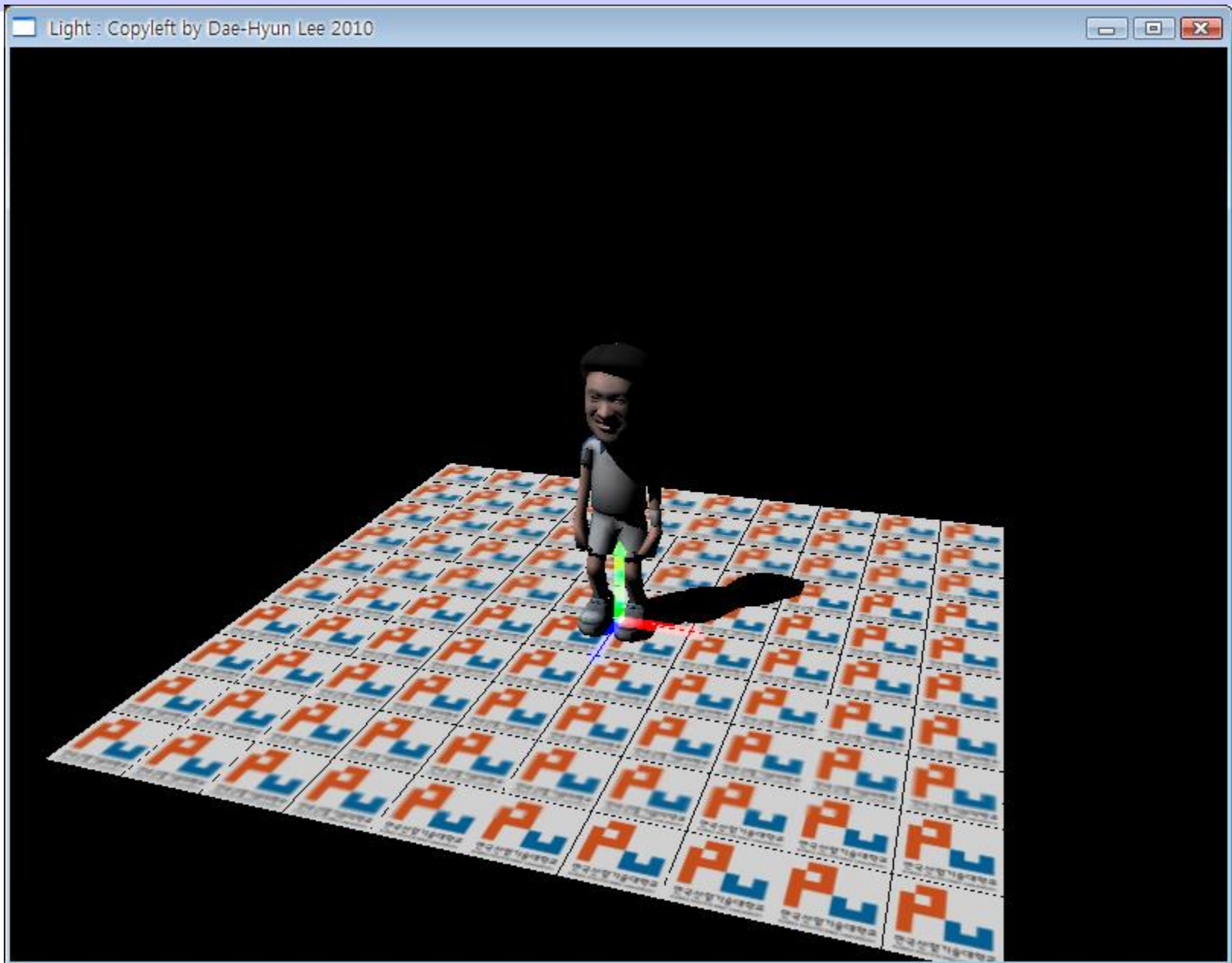


```
void _setLights(void)
{
    mSceneMgr->setAmbientLight(ColourValue(0.0f, 0.0f, 0.0f));
    mSceneMgr->setShadowTechnique(SHADOWTYPE_STENCIL_ADDITIVE);
    ... 중략 ...
}

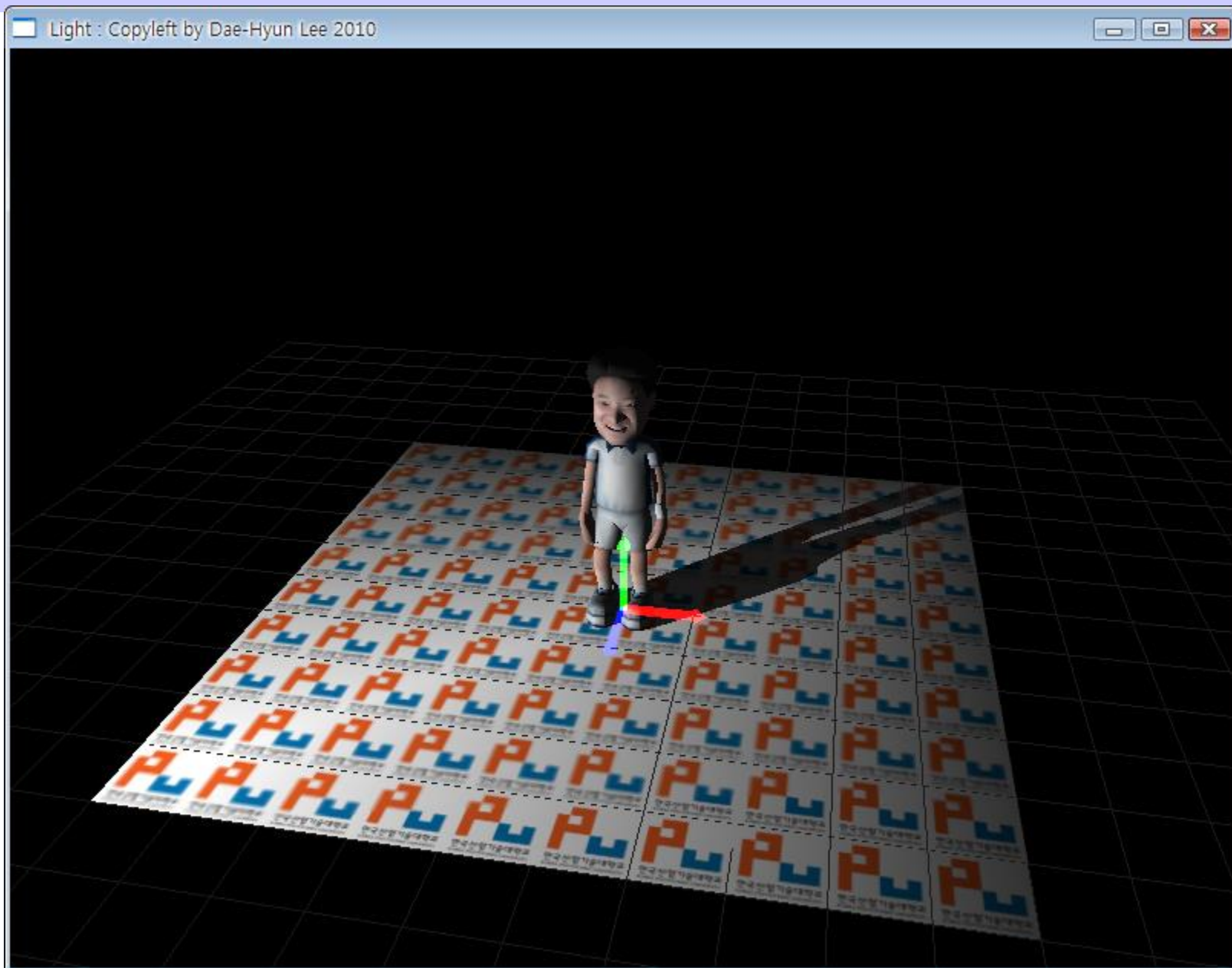
void _drawGroundPlane(void)
{
    ... 중략 ...
    groundEntity->setMaterialName("KPU_LOGO");
    groundEntity->setCastShadows(false);
}

void go(void)
{
    ... 중략 ...
    professorYaw->attachObject(entity);
    entity->setCastShadows(true);
}
```

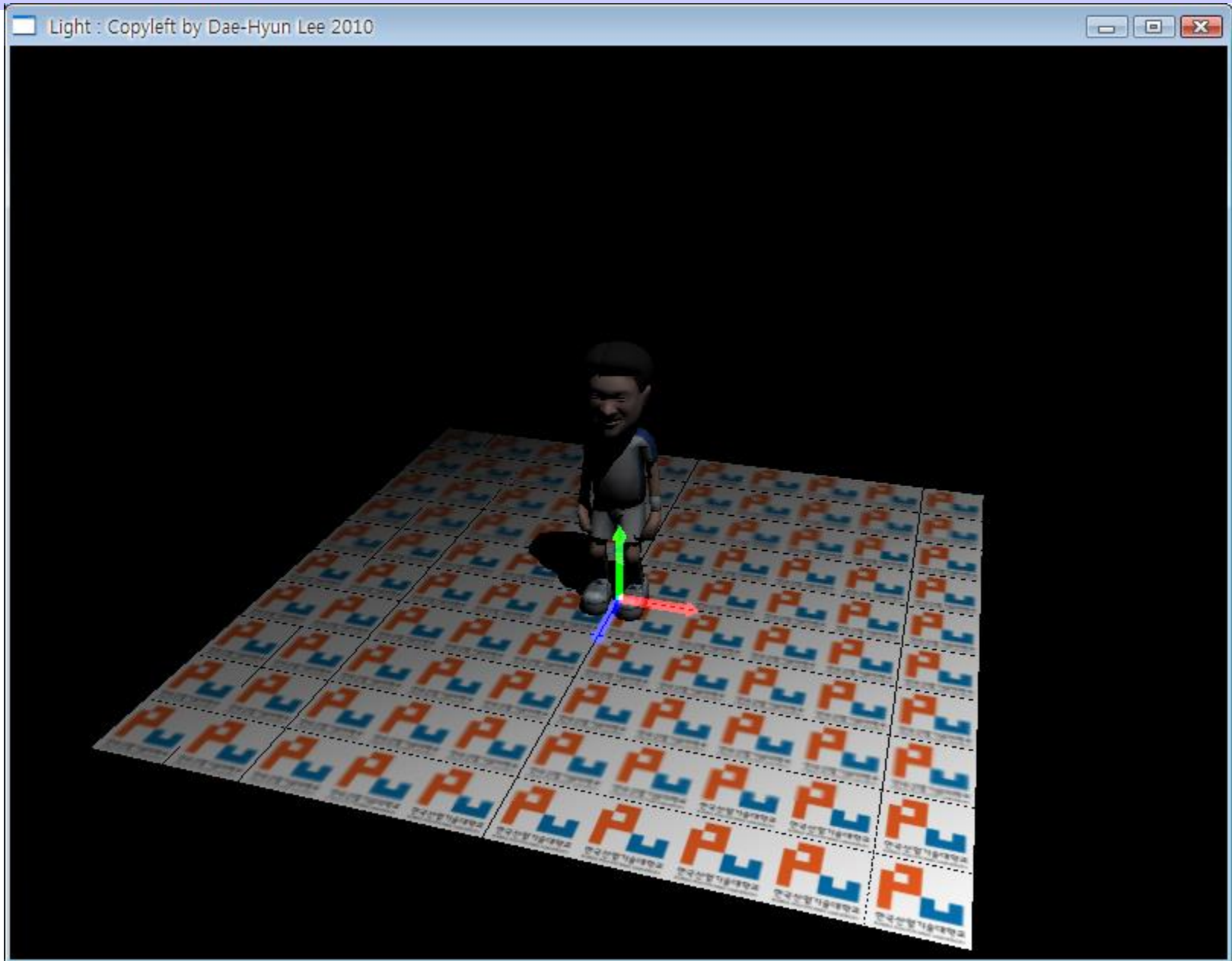
Directional Light



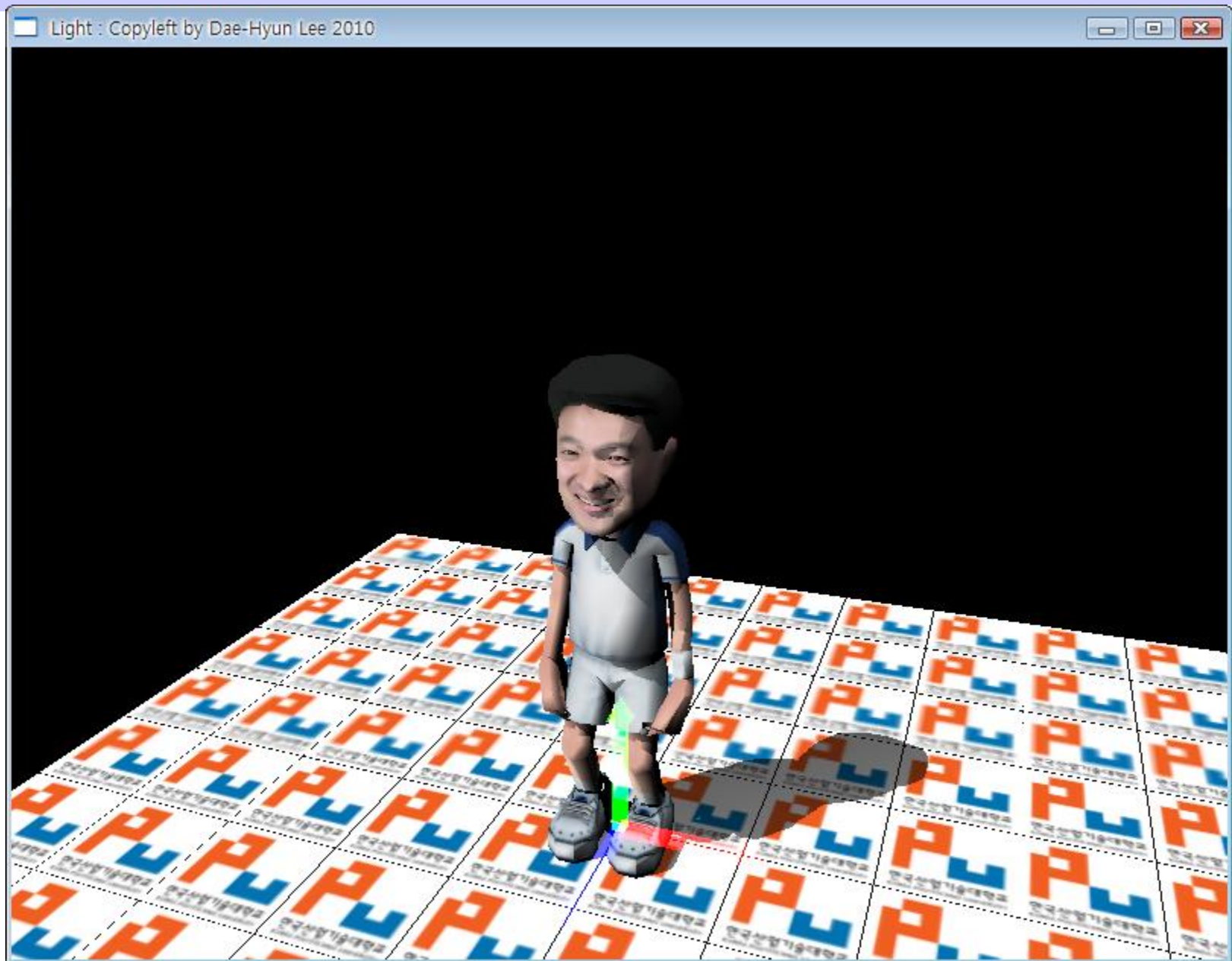
Point Light



Spot Light



모든 광원을 ON



광원의 생성

■ Light 클래스

- SceneManager::createLight(): 광원 생성
- Light::setType(): 광원 종류 설정
- Light::setPosition(): 광원의 위치 설정
- Light::setDirection(): 광원의 방향 설정
- 광원 객체를 장면노드에 소속시키면, 광원을 이동하는 것이 가능함.
 - → 캐릭터를 따라다니는 광원을 구현할 수 있음.

• 광원의 생성

• 광원의 종류를 설정. 지향 광원으로 설정함.

```
mLightD = mSceneMgr->createLight("LightD");  
mLightD->setType(Light::LT_DIRECTIONAL);  
mLightD->setDirection( Vector3( 1, -2.0f, -1 ) );  
mLightD->setVisible(false);
```

• 광원의 방향 설정.

• 광원 ON/OFF

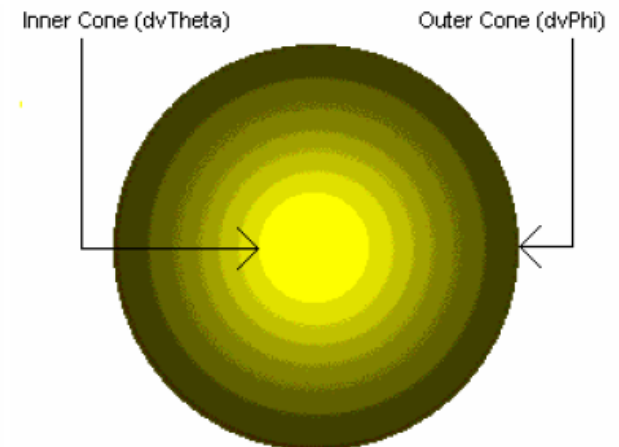
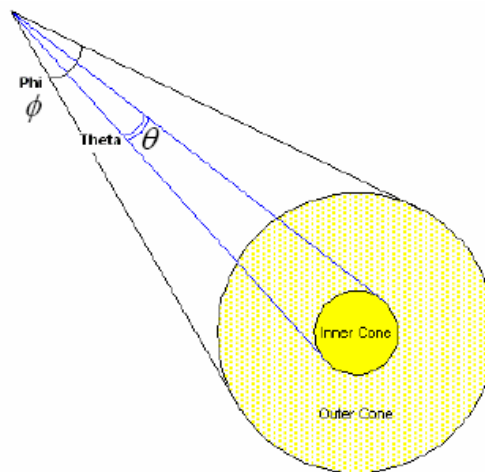
점광원 및 점적광원의 생성 및 설정

```
mLightP = mSceneMgr->createLight("LightP");  
mLightP->setType( Light::LT_POINT );  
mLightP->setPosition( Vector3(-250, 50, 250) );  
mLightP->setVisible(false);
```

• 점광원의 위치 결정.

```
mLightS = mSceneMgr->createLight("LightS");  
mLightS->setType( Light::LT_SPOTLIGHT );  
mLightS->setDirection(Ogre::Vector3::NEGATIVE_UNIT_Y);  
mLightS->setPosition( Vector3( 250, 900, 250) );  
mLightS->setSpotlightRange( Degree(10), Degree(80));  
mLightS->setVisible(false);
```

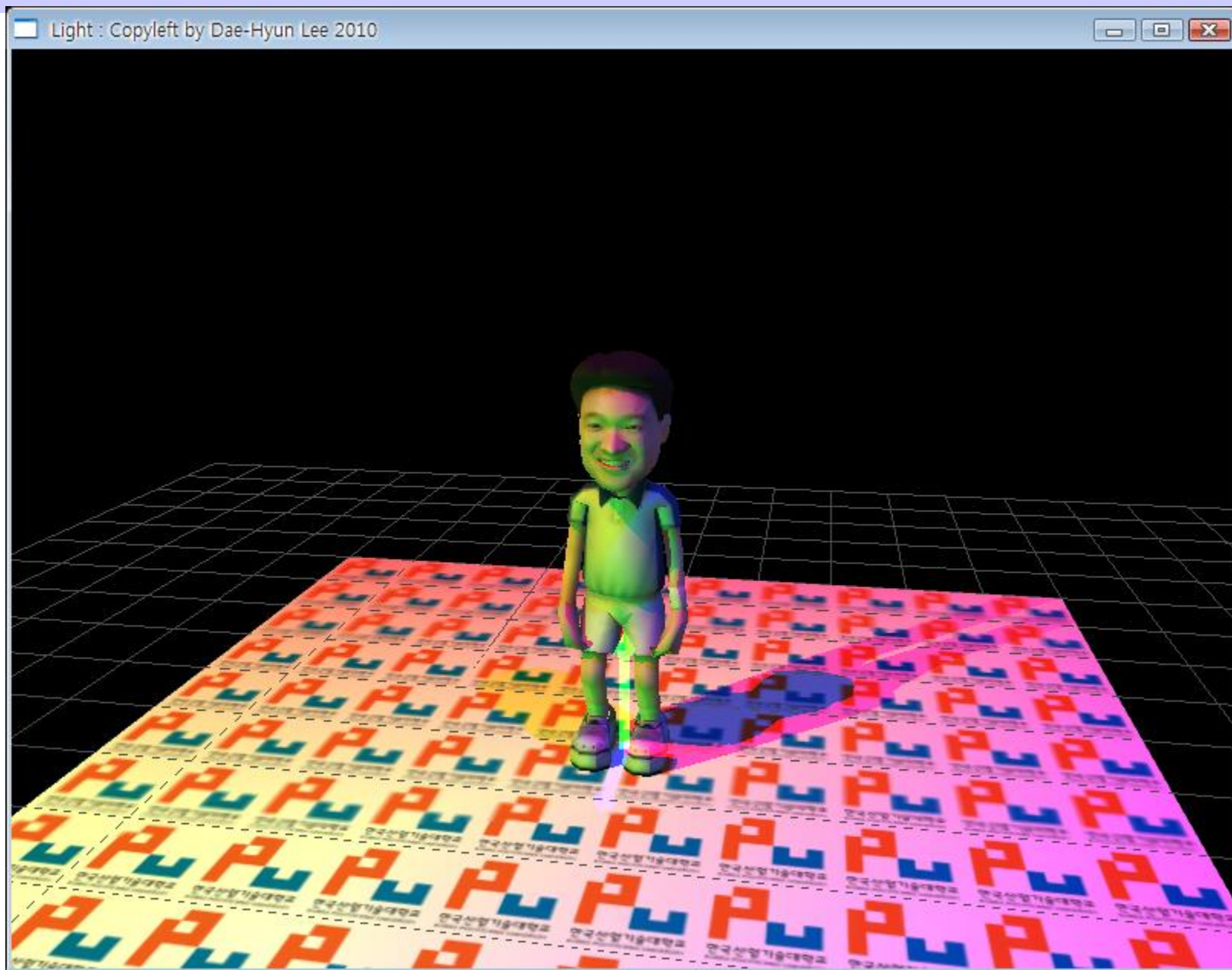
- 점적 광원의 확산 각도(안쪽각도,바깥쪽각도) 설정.
- 안쪽각도는 *Direct3D*에만 적용 가능.
- *OpenGL*에서는 0도로 간주.





```
void _setLights(void)
{
    ... 중략 ...
    mLightD->setDiffuseColour(1.0f, 0.0f, 0.0f);
    mLightP->setDiffuseColour(0.0f, 1.0f, 0.0f);
    mLightS->setDiffuseColour(0.0f, 0.0f, 1.0f);
}
```

광원 색상 혼합



조명(Illumination) 모델

■ 조명 모델

- 광원으로부터 공간상의 점들까지의 조도를 계산하는 방법.

■ 직접조명과 전역 조명

- 직접 조명(direct illumination) 모델

- 물체표면의 점들이 장면 내의 모든 광원들로부터 직접적으로 받는 빛만을 고려.

- 전역조명(global illumination) 모델

- 다른 물체를 거쳐서 받는 빛까지 모두 포함한 모델.
- 계산량이 많이 요구됨.

재질(Material)

■재질이란?

- 물체가 가지고 있는 고유의 색상.
- 우리가 물체를 볼 수 있는 것은 광자가 그 물체의 표면에 부딪혀 튕겨나와 우리 눈에 도달하기 때문인데, 그 물체의 표면에서 튕겨나올 때의 광자의 속성은 물체 표면의 재질에 따라 달라지게 됨.

■재질의 색상의 종류

- 주변재질색상(Ambient Material Color)
 - 주변광원에 대한 재질의 색상
- 분산재질색상(Diffuse Material Color)
 - 물체 표면 자체의 색상
- 전반사재질색상(Specular Material Color)
 - 광원이 반사되어 물체의 가장 밝게 빛나는 부분의 색상.
- 방사재질색상(Emissive Material Color)
 - 발광체인 물체를 표현할 때 사용하는 색상.

■ 재질색상의 해석

□ 재질 표면에 도달한 광자를 재질표면이 반사시키는 비율로 해석할 수 있음.

□ Ex. 적색공이 적색공으로 보이는 이유는?

- 주변광원의 색상 순백색 ($R=1, G=1, B=1$) 임.
- 적색공의 주변재질색상이 적색 ($R=1, G=0, B=0$) 임.
- 적색공은 주변광원으로 받는 광자를 모두 다 반사시킴. 따라서 적색으로 나타남.


□ Ex. 주변광원 색상이 순녹색 ($R=0, G=1, B=0$)으로 바뀌게 되면?

• 자체 발광 성분.

• 주변광원에 따른 성분.

• 빛의 난반사에 의한 성분.

• 빛의 전반사에 의한 성분.

$$I = I_e + I_a + I_d + I_s$$


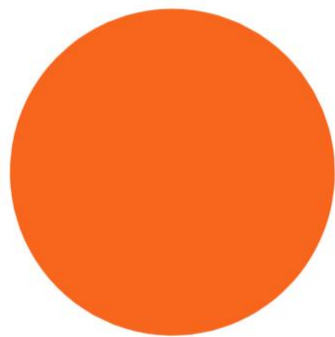
The diagram illustrates the four components of direct lighting: I_e (emission), I_a (ambient), I_d (diffuse reflection), and I_s (specular reflection). Each component is represented by a sphere or circle below the equation, with an orange arrow pointing from the corresponding text box to the term in the equation.

주변 반사(Ambient Reflection) 성분

■ 주변 반사 성분

- 주변광원에 대한 재질의 반응을 표현.
- 직접조명모델에서는 물체가 광원으로부터 직접 받는 빛만을 고려하기 때문에, 다른 물체에 반사된 빛을 처리하지 못함.
- 따라서 다른 물체를 통해서 반사된 빛을 근사적으로 간략하게 처리하기 위해 도입한 성분임.

$$I_a = K_a \otimes L_a$$



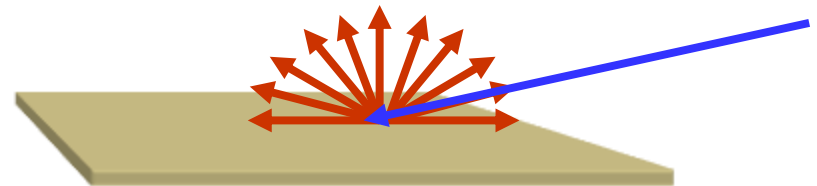
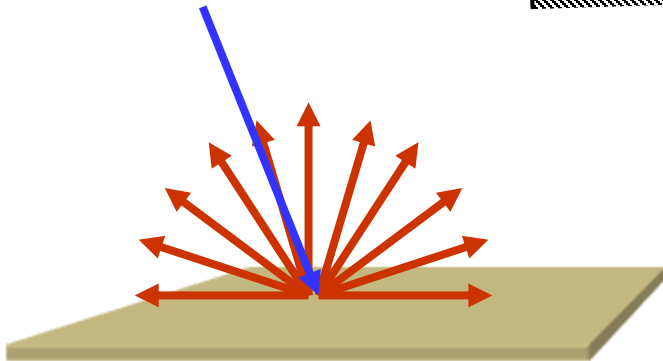
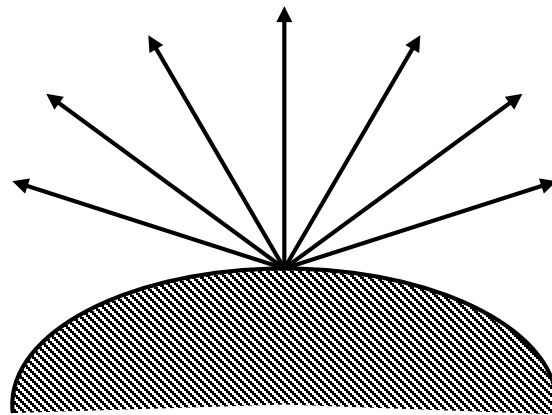
• 주변재질색상

- 광원 색상.
- `setAmbientLight()` 로 설정.

분산(Diffuse) 성분

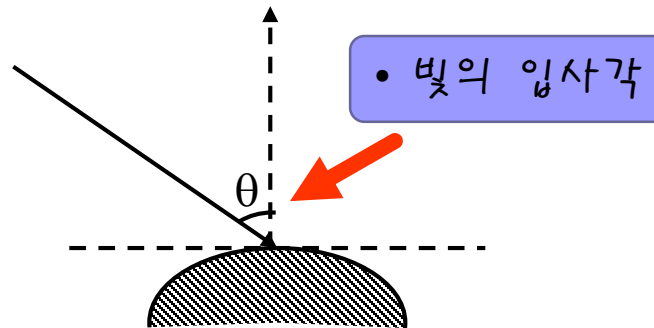
■ 분산 성분

- 광원으로부터 빛이 들어올 때, 모든 방향으로 똑같은 양을 반사시키는 완전한 난반사체를 모델링.
→ 재질의 거친 표면을 모델링.
- 관찰자의 위치에 상관없이 동일한 양의 빛이 반사됨.



■ 분산 성분의 계산

- 반사되는 빛의 세기는 빛의 입사각에 따라 달라짐.



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

• 재질 색상

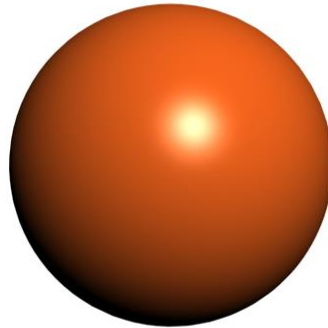
• 광원 색상.
• `setDiffuseColor()` 로 설정.



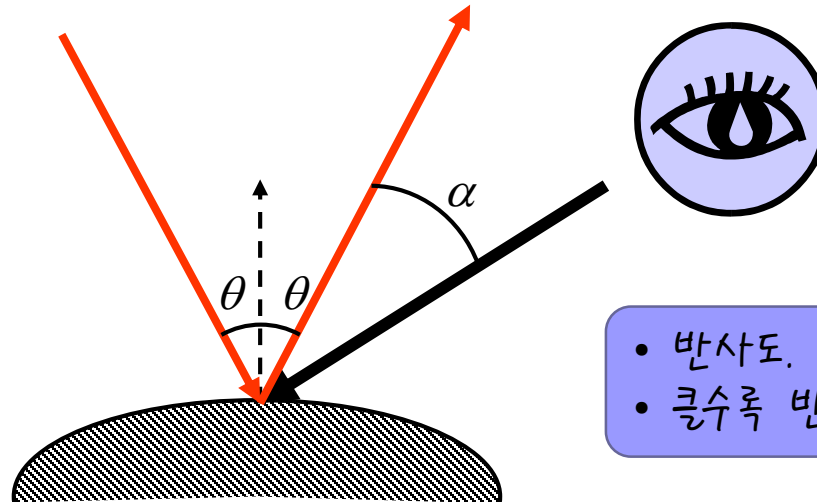
전반사(Specular) 성분

■전반사 성분

- 매끄러운 재질의 표면을 모델링.
- 관찰자의 시점의 따라서 반짝거리는 정도가 달라짐.
- 전반사 재질 색상과 반사도에 따라 영향을 받음.



■ 전반사 성분의 계산



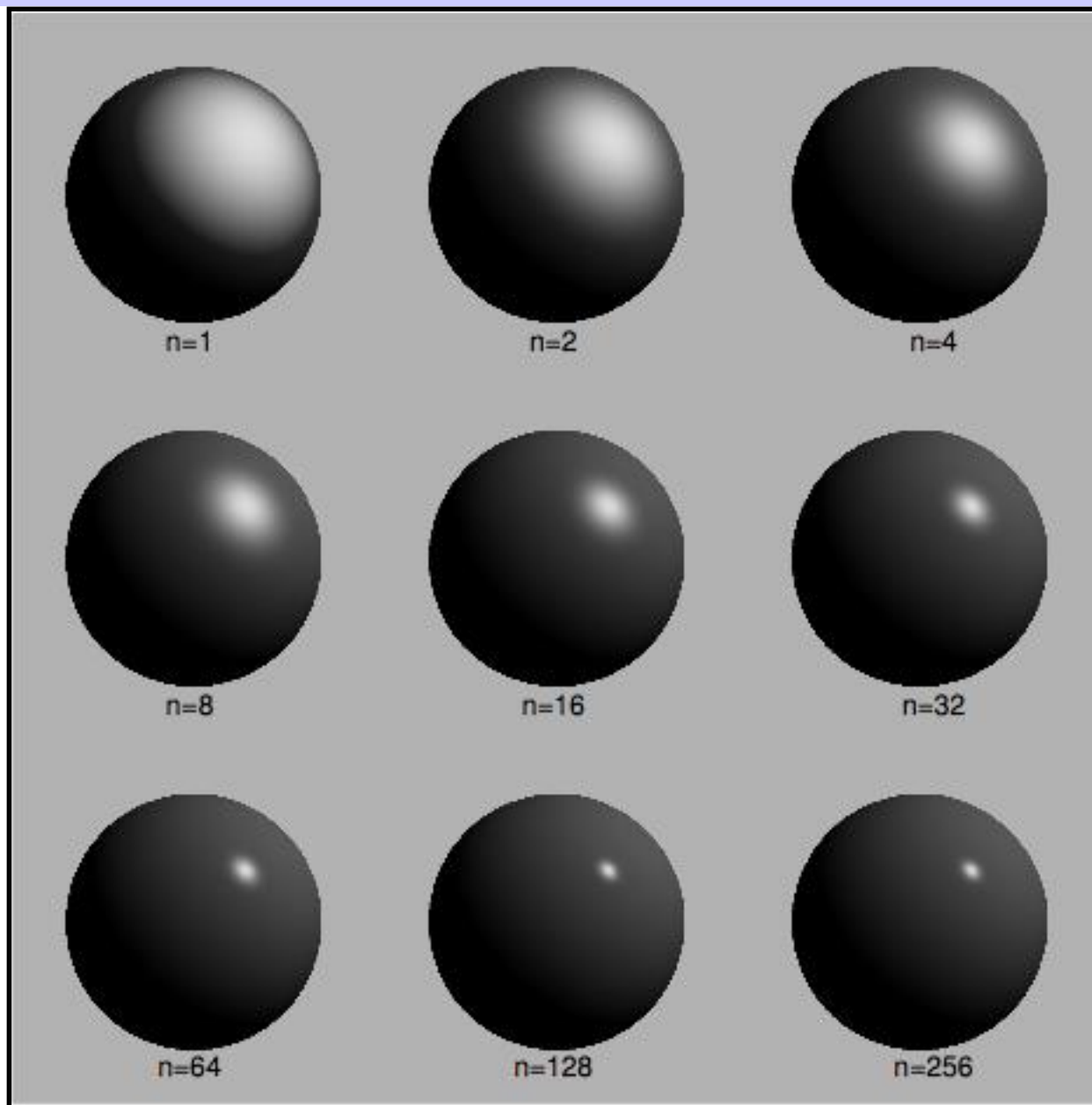
- 반사도.
- 클수록 반사 범위가 좁아짐.

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

- 재질 색상

- 광원 색상.
- `setSpecularColor()` 로 설정.

반사도에 따른 변화



셰이딩(Shading)

■ 직접 조명 모델에서 가장 중요한 요소는?

□ 법선 벡터!

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

입사각 → 법선에 따라 바뀜. 시야각과 반사값의 차이

■ 광원의 종류

- 주변광원 / 지향광원 / 점광원 / 점적광원

■ 광원의 생성 및 설정

- SceneManager::createLight(): 광원 생성
- Light::setType(): 광원 종류 설정
- Light::setPosition(): 광원의 위치 설정
- Light::setDirection(): 광원의 방향 설정

■ 조명 방정식

- 법선 벡터가 중요함.

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