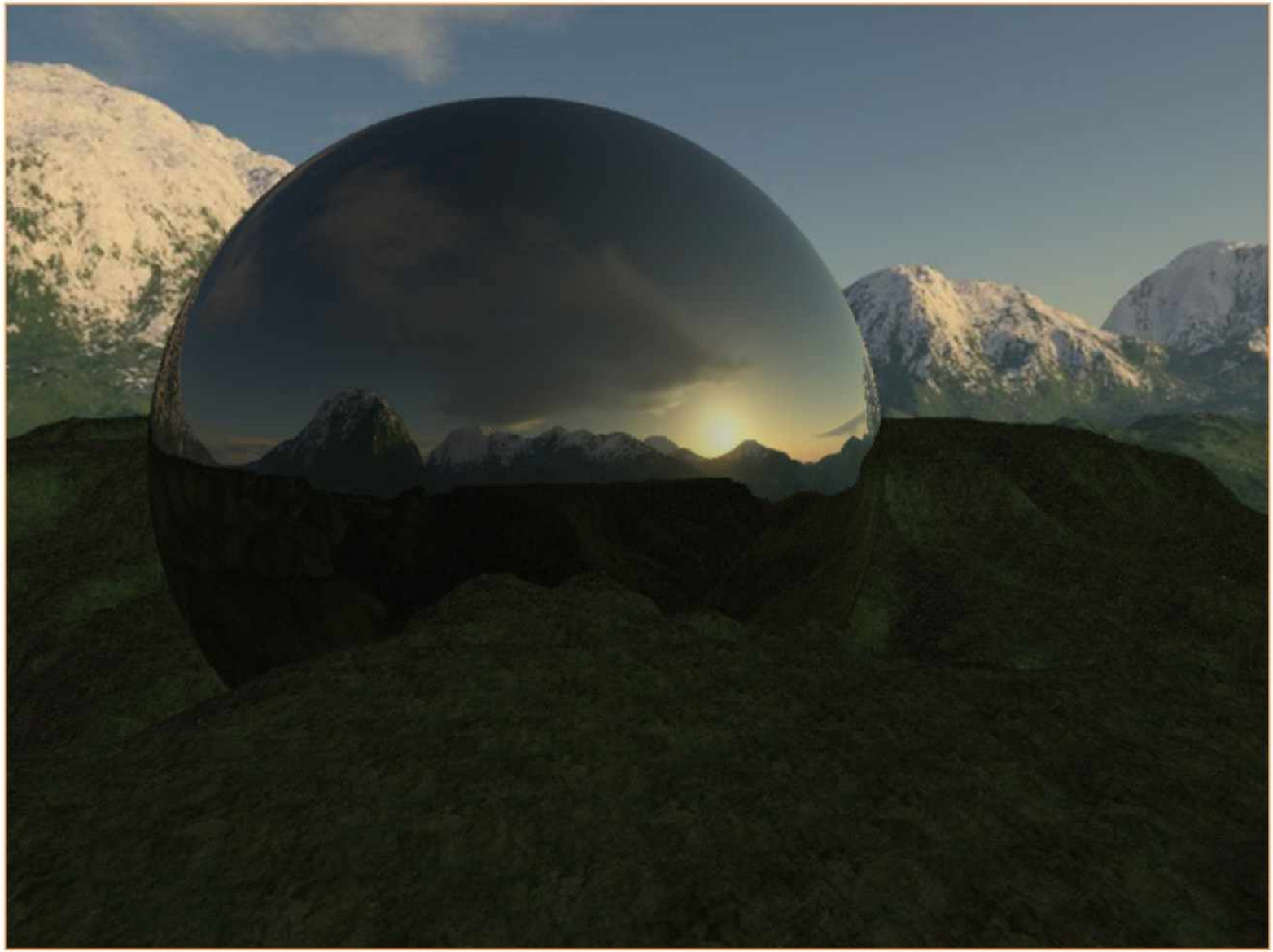
Game Programming with DirectX

Direct3D Graphics Pipeline (Shader Samples)

Cube Mapping

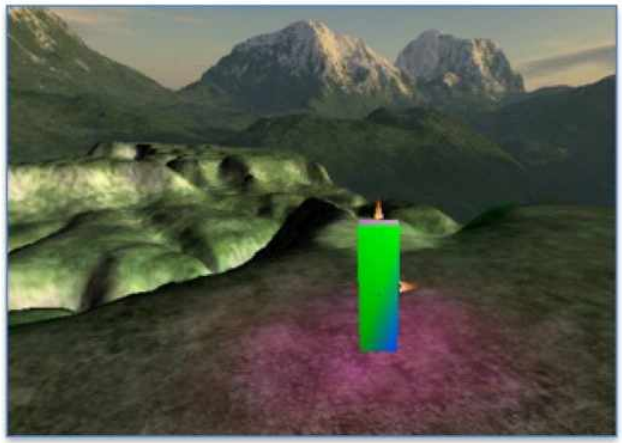
큐브 매핑(Cube Mapping)

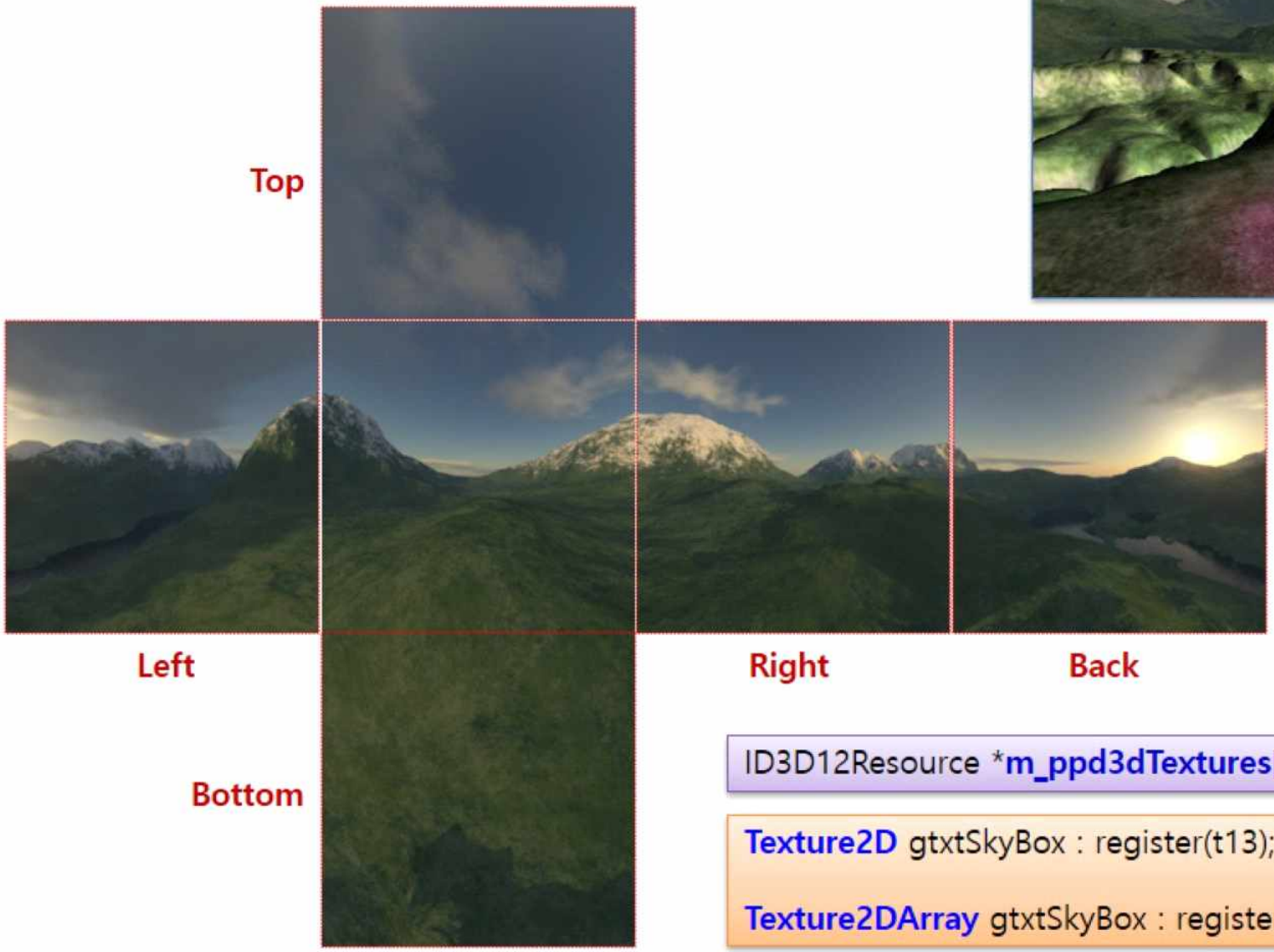
- 큐브 매핑(Cube Mapping)



큐브 매핑(Cube Mapping)

- 스카이 박스(Sky Box) – 텍스쳐 배열





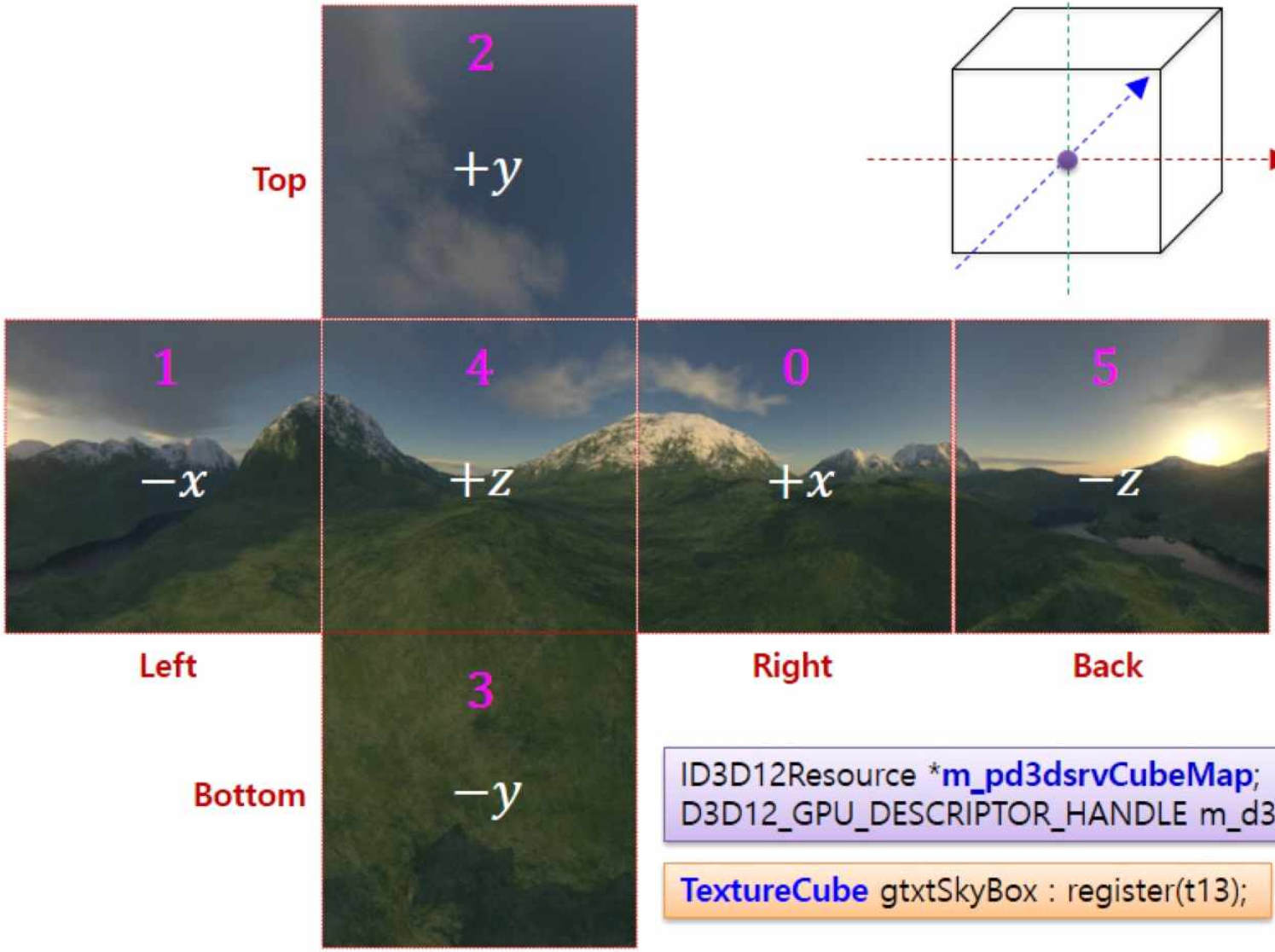

```
ID3D12Resource *m_ppd3dTextures[6];
```

```
Texture2D gtxtSkyBox : register(t13);

Texture2DArray gtxtSkyBox : register(t13);
```

큐브 매핑(Cube Mapping)

- 스카이 박스(Sky Box) – 큐브 맵(Cube Map)




```
ID3D12Resource *m_pd3dsrvCubeMap;
D3D12_GPU_DESCRIPTOR_HANDLE m_d3dSrvGpuDescriptor;
```

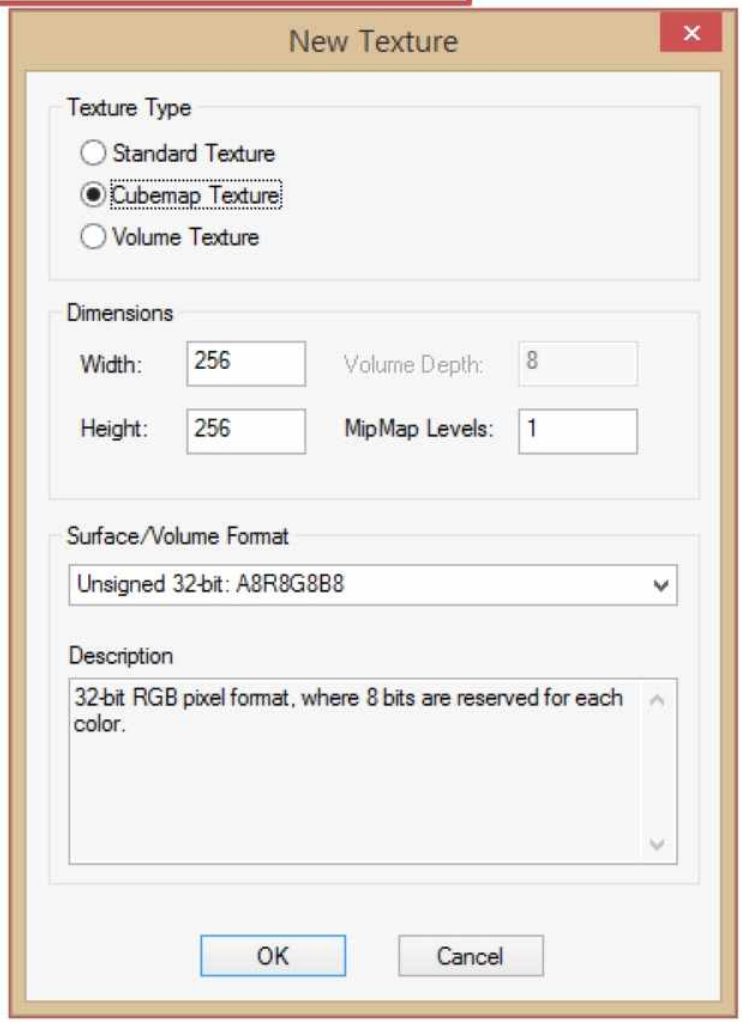
```
TextureCube gtxtSkyBox : register(t13);
```

큐브 매핑(Cube Mapping)

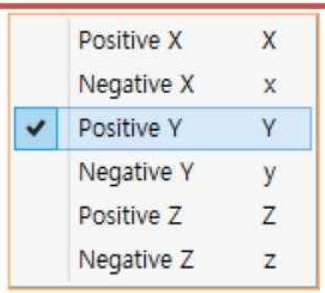
- **큐브 맵(Cube Map) 생성**

C:₩Program Files (x86)₩Microsoft DirectX SDK (June 2010)₩Utilities₩bin₩x86₩**DxTex.exe**

① File → New Texture...



② View → Cube Map Face →



③ File → Open Onto Cubemap Face... (텍스쳐 불러오기)

(1) Positive X
(2) Negative X
(3) Positive Y
(4) Negative Y
(5) Positive Z
(6) Negative Z

DDS

④ File → Save As... (큐브 맵 저장하기)

"SkyBox.dds"

https://developer.nvidia.com/legacy-texture-tools
http://planetside.co.uk/products/terragen3

큐브 매핑(Cube Mapping)

- **큐브 맵(Cube Map) 생성**
  - https://github.com/Microsoft/DirectXTex

DirectXTex_Desktop_2017.sln

DirectXTex-master\Texassemble\Bin\Desktop_2017\Win32\Debug\texassemble.exe

texassemble <**command**> <options> <**files**>

texassemble cube –o skybox.dds -y skybox01.bmp ... skybox06.bmp

- cube
- volume
- array
- cubearray

입력 파일의 크기와 형식이 같아야 함
입력 파일은 밉맵 레벨이 1이어야 함
출력 파일은 항상 dds 형식임

texassemble <**command**> <options> <**files**>

-r	wildcard filename search is recursive
-w <n>	width
-h <n>	height
-f <**format**>	format
-if <**filter**>	image filtering
-srgb{i|o}	sRGB {input, output}
-o <filename>	output filename
-y	overwrite existing output file (if any)
-sepalpha	resize alpha channel separately from color channels
-wrap, -mirror	texture addressing mode (wrap, mirror, or clamp)
-alpha	convert premultiplied alpha to straight alpha
-dx10	Force use of 'DX10' extended header
-nologo	suppress copyright message
-tonemap	apply a tonemap operator based on maximum luminance

- POINT
- LINEAR
- CUBIC
- FANT
- BOX
- TRIANGLE
- POINT_DITHER
- LINEAR_DITHER
- CUBIC_DITHER
- FANT_DITHER
- BOX_DITHER
- TRIANGLE_DITHER
- ...

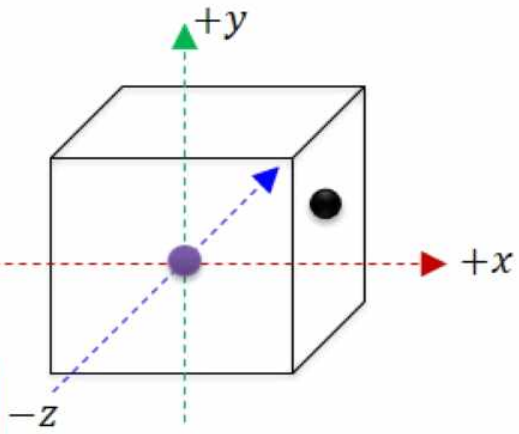
- R32G32B32A32_FLOAT
- R32G32B32A32_UINT
- R32G32B32A32_SINT
- R32G32B32_FLOAT
- R32G32B32_UINT
- R32G32B32_SINT
- R16G16B16A16_FLOAT
- R8G8B8A8_UNORM
- R8G8B8A8_UNORM_SRGB
- R8G8B8A8_UINT
- R8G8B8A8_SNORM
- R8G8B8A8_SINT
- R32_FLOAT
- R32_UINT
- R32_SINT
- ...

큐브 매핑(Cube Mapping)

- **큐브 맵(Cube Map) 텍스쳐 좌표**

점(픽셀) 위치 벡터: (x, y, z), 큐브의 중심에서 점까지의 벡터: (x, y, z)

위치 벡터가 주어지면 큐브의 면과 텍스쳐 좌표를 계산할 수 있음
① 위치 벡터의 각 성분의 절대값의 최대값(α)과 부호에 해당하는 **면**을 선택
2개 이상의 성분이 절대값이 같으면 z, y, x의 순서로 선택
② 나머지 두 성분을 α로 나눔(-1.0 ~ +1.0)
③ 두 성분에 1.0을 더하고 2로 나누면 텍스쳐 좌표(0 ~ 1.0)를 얻음

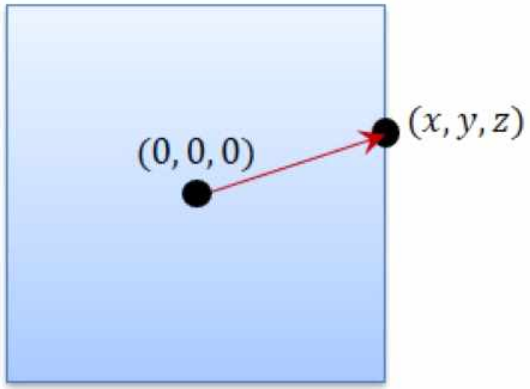


방향 벡터만을 사용하여 큐브 맵 샘플링을 수행

$$(-3.2, 5.1, -8.4) \rightarrow -z$$

$$(-3.2, 5.1, -8.4) \rightarrow \left(\frac{-3.2}{8.4}, \frac{5.1}{8.4}\right)$$

$$\left(\left(\frac{-3.2}{8.4}+1\right)*0.5, \left(\frac{5.1}{8.4}+1\right)*0.5\right) \rightarrow (0.31, 0.80)$$

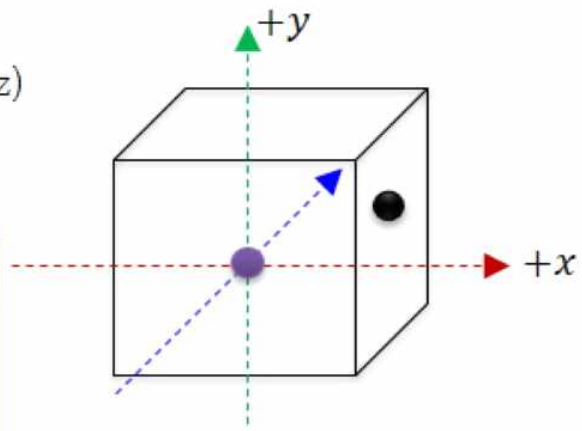


그래픽 하드웨어(Direct3D 샘플러)는 위치 벡터로부터 큐브의 면과 텍스쳐 좌표를 계산하여 샘플링을 수행

큐브 매핑(Cube Mapping)

- **큐브 맵(Cube Map) 텍스쳐 좌표**

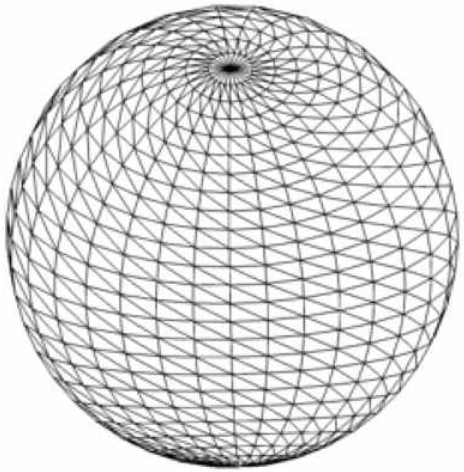
  점(픽셀) 위치 벡터: (x, y, z), 큐브의 중심에서 점까지의 벡터: (x, y, z)

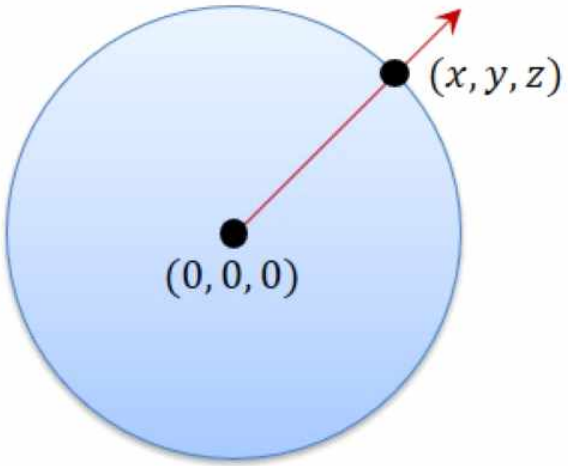


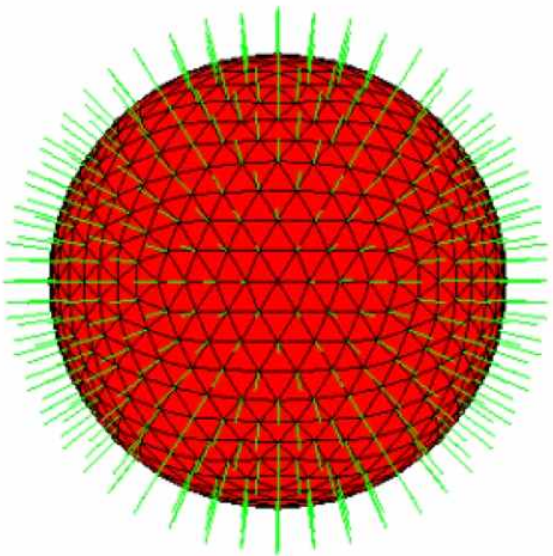
면(Face)	u	v	행렬 $M_{face \to camera}$
$+x$	$\frac{1}{2}\left(1-\frac{z}{x}\right)$	$\frac{1}{2}\left(1-\frac{y}{x}\right)$	$\begin{bmatrix} 0 & 0 & -1 \\ 0 & -1 & 0 \\ +1 & 0 & 0 \end{bmatrix}$
$-x$	$\frac{1}{2}\left(1-\frac{z}{x}\right)$	$\frac{1}{2}\left(1+\frac{y}{x}\right)$	$\begin{bmatrix} 0 & 0 & +1 \\ 0 & -1 & 0 \\ -1 & 0 & 0 \end{bmatrix}$
$+y$	$\frac{1}{2}\left(1+\frac{x}{y}\right)$	$\frac{1}{2}\left(1+\frac{z}{y}\right)$	$\begin{bmatrix} +1 & 0 & 0 \\ 0 & 0 & +1 \\ 0 & +1 & 0 \end{bmatrix}$
$-y$	$\frac{1}{2}\left(1-\frac{x}{y}\right)$	$\frac{1}{2}\left(1+\frac{z}{y}\right)$	$\begin{bmatrix} +1 & 0 & 0 \\ 0 & 0 & -1 \\ 0 & -1 & 0 \end{bmatrix}$
$+z$	$\frac{1}{2}\left(1+\frac{x}{z}\right)$	$\frac{1}{2}\left(1-\frac{y}{z}\right)$	$\begin{bmatrix} +1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & +1 \end{bmatrix}$
$-z$	$\frac{1}{2}\left(1+\frac{x}{z}\right)$	$\frac{1}{2}\left(1+\frac{y}{z}\right)$	$\begin{bmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \end{bmatrix}$

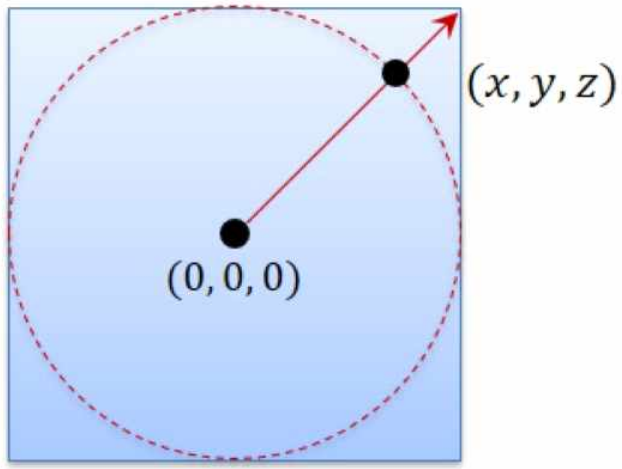
큐브 매핑(Cube Mapping)

- **큐브 매핑(Cube Mapping)**
  - 구(Sphere)









큐브 매핑(Cube Mapping)

- **큐브 맵핑(Cube Mapping)**
  - 스카이 박스(Sky Box)

```
CTexture *pSkyBoxTexture = new CTexture(1, RESOURCE_TEXTURE_CUBE, 0);
pSkyBoxTexture->LoadTextureFromFile(pd3dDevice, pd3dCommandList, L"SkyBox.dds", 0);
```

```
pd3dInputElementDescs[0] ={ "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0, ..., 0 };
```

스카이 박스 메쉬(육면체)의 모든 면들은 안쪽을 향해야 함
큐브 맵 텍스쳐를 리소스로 사용
  텍스쳐 좌표는 샘플링을 할 때 자동적으로 계산됨

```
struct VS_SKYBOX_INPUT
{
   float3 position : POSITION;
};
```

```
struct VS_SKYBOX_OUTPUT
{
   float3 position : POSITION;
   float4 positionH: SV_POSITION;
};
```

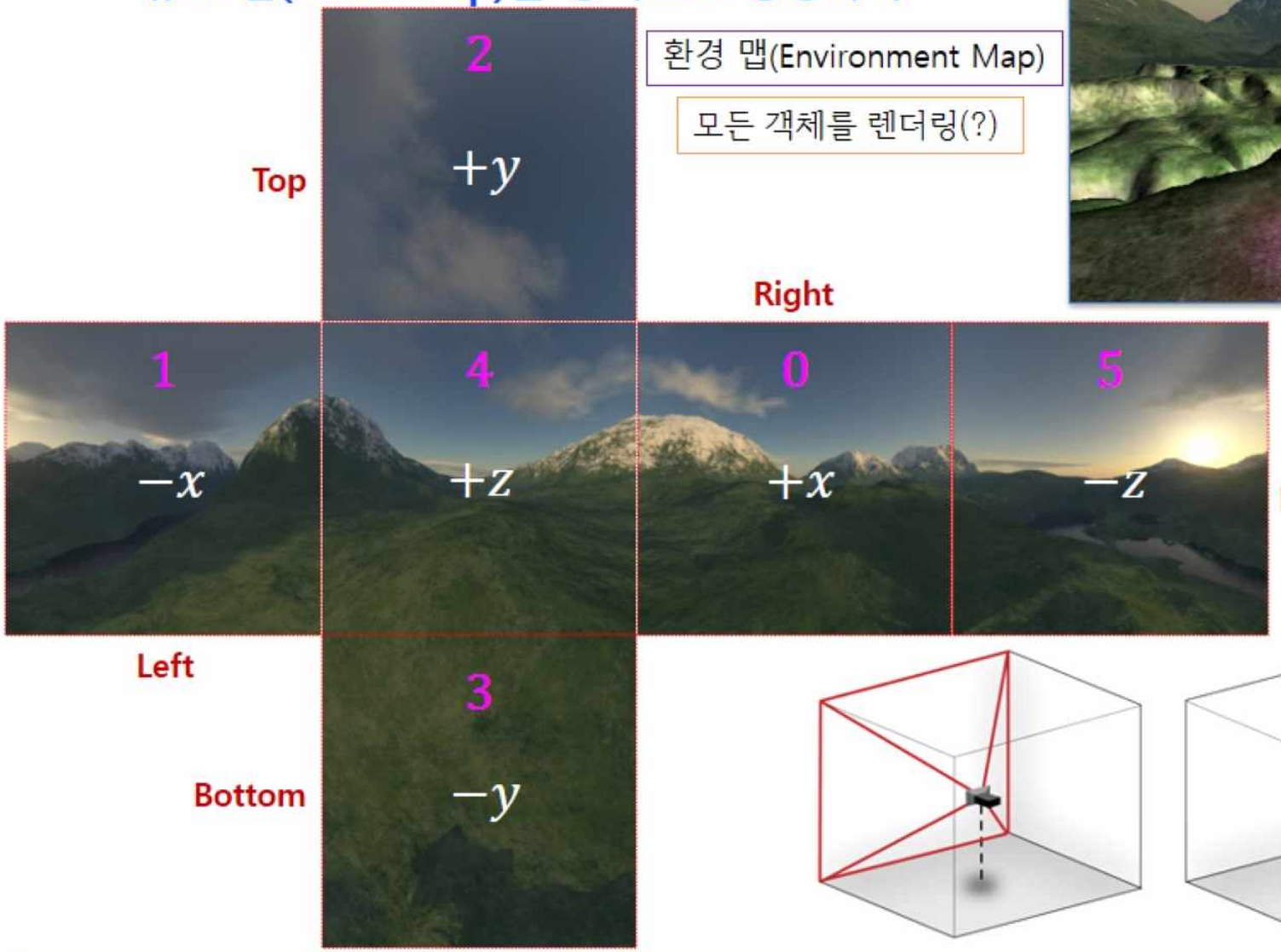
```
VS_SKYBOX_OUTPUT VSSkyBox(VS_SKYBOX_INPUT input)
{
   VS_SKYBOX_OUTPUT output = (VS_SKYBOX_OUTPUT)0;
   output.positionH = mul(mul(mul(float4(input.position, 1.0f), gmtxWorld), gmtxView), gmtxProjection);
   output.position = input.position;
   return(output);
}
```

```
TextureCube gtxtCubeMapSkyBox : register(t1);
SamplerState gssSkyBox : register(s1);
```

```
float4 PSSkyBox(VS_SKYBOX_OUTPUT input) : SV_Target
{
   float4 cColor = gtxtCubeMapSkyBox.Sample(gssSkyBox, input.position);
   return(cColor);
}
```

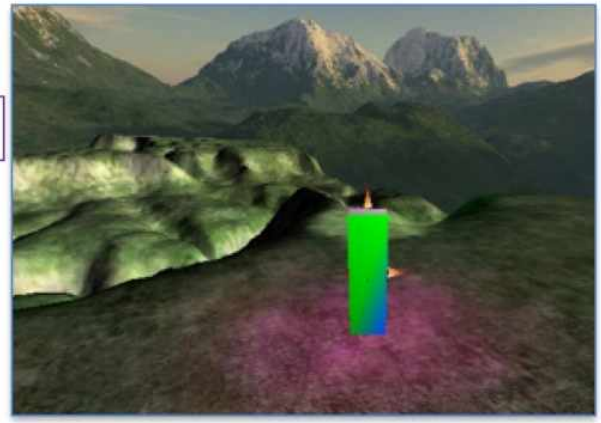
큐브 매핑(Cube Mapping)

- 큐브 맵(Cube Map)을 동적으로 생성하기



환경 맵(Environment Map)

모든 객체를 렌더링(?)

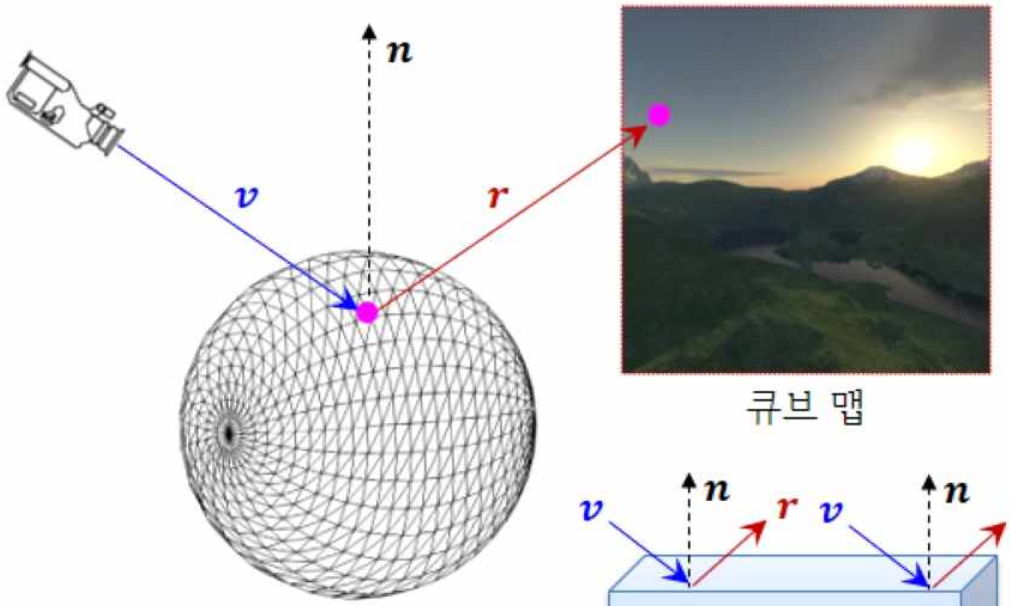


카메라 FOV: 90도

큐브 매핑을 할 객체의 위치에서 카메라가 90도씩 회전하면서 렌더링(사진을 찍음)하여 6개의 텍스쳐를 구하면 됨

큐브 매핑(Cube Mapping)

- 큐브 매핑(Cube Mapping)

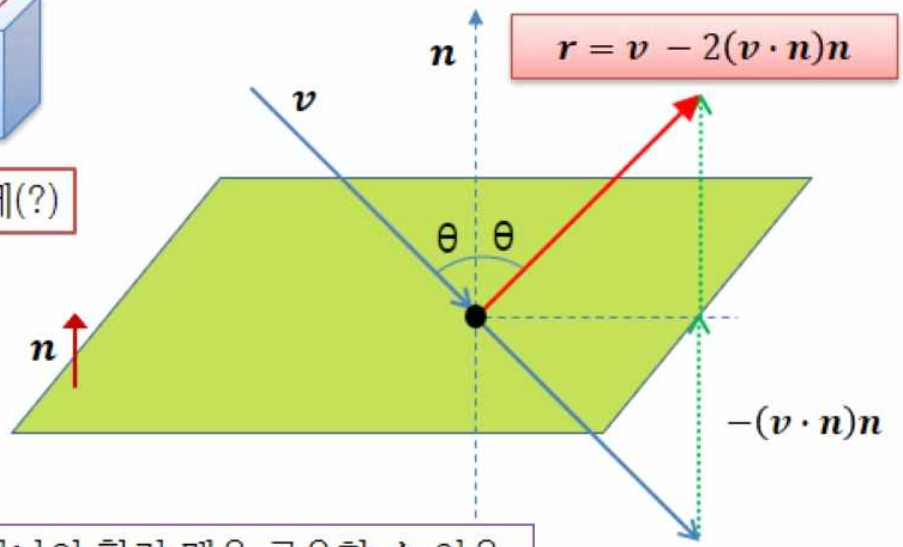


큐브 맵

방향 벡터만을 사용하여 큐브 맵 샘플링을 수행: 어떤 문제(?)

v = 점 위치 벡터 – 카메라 위치 벡터
벡터 v의 반사 벡터 r을 구함
큐브 맵에서 반사 벡터 r을 사용하여 샘플링

$r = (x, y, z)$의 가장 큰 요소가 면을 결정
나머지 요소는 텍스쳐 좌표



여러 객체가 하나의 환경 맵을 공유할 수 있음

큐브 매핑(Cube Mapping)

- **큐브 매핑(Cube Mapping)** – 환경 매핑(Environmental Mapping)

```
void CCubeMappingShader::CreateShader(ID3D12Device *pd3dDevice) {
   D3D12_INPUT_ELEMENT_DESC d3dInputLayout[] = {
      { "POSITION", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 0, D3D12_INPUT_PER_VERTEX_DATA, 0 },
      { "NORMAL", 0, DXGI_FORMAT_R32G32B32_FLOAT, 0, 12, D3D12_INPUT_PER_VERTEX_DATA, 0 }
   };
   UINT nElements = ARRAYSIZE(d3dInputLayout);
   CreateVertexShaderFromFile(pd3dDevice, L"Effect.fx", "VSLightingColor", ..., &m_pd3dVertexLayout);
   CreatePixelShaderFromFile(pd3dDevice, L"Effect.fx", "PSCubeMapping", "ps_4_0", &m_pd3dPixelShader);
}
```

```
VS_LIGHTING_OUTPUT VSLightingColor(VS_LIGHTING_INPUT input) {
   VS_LIGHTING_OUTPUT output = (VS_LIGHTING_OUTPUT)0;
   output.normalW = mul(input.normal, (float3x3)gmtxWorld);
   output.positionW = mul(input.position, (float3x3)gmtxWorld);
   output.position = mul(float4(input.position, 1.0f), mtxWorldViewProjection);
   return(output);
}
```

```
struct VS_LIGHTING_INPUT {
   float3 position : POSITION;
   float3 normal : NORMAL;
};
```

```
struct VS_LIGHTING_OUTPUT {
   float4 position : SV_POSITION;
   float3 positionW : POSITION;
   float3 normalW : NORMAL;
};
```

```
float4 PSCubeMapping(VS_LIGHTING_OUTPUT input) : SV_Target {
   float4 cIllumination = Lighting(input.positionW, input.normalW);
   float3 vFromCamera = normalize(input.positionW - gvCameraPosition.xyz);
   float3 vReflected = normalize(reflect(vFromCamera, input.normalW)); //정점 쉐이더에서 계산하면?
   float4 cCubeTextureColor = gtxtCubeMapped.Sample(gssCubeMapped, vReflected);
   return(cIllumination * cCubeTextureColor);
}
```

```
TextureCube gtxtCubeMap : register(t6);
SamplerState gssCubeMap : register(s5);
```

큐브 매핑(Cube Mapping)

- **큐브 매핑(Cube Mapping)** – 환경 매핑(Environmental Mapping)

```
VS_SKYBOX_OUTPUT VSToCubeMap(VS_SKYBOX_INPUT input) {
   VS_SKYBOX_OUTPUT output = (VS_SKYBOX_OUTPUT)0;
   output.position = mul(float4(input.position, 1.0f), gmtxWorld);
   output.positionL = input.position;
   return(output);
}
```

```
struct VS_SKYBOX_INPUT {
   float3 position : POSITION;
};
```

```
struct VS_SKYBOX_OUTPUT {
   float3 positionL : POSITION;
   float4 position : SV_POSITION;
};
```

```
[maxvertexcount(18)]
void GSSkyBox(triangle VS_SKYBOX_OUTPUT input[3], inout TriangleStream<GS_SKYBOX_OUTPUT> s) {
   for (int i = 0; i < 6; i++) {
      GS_SKYBOX_OUTPUT output;
      output.renderTarget = i;
      matrix mtxViewProjection = mul(gmtxCubeMappingViews[i], gmtxCubeMappingProjection);
      for (int j = 0; j < 3; j++) {
         output.position = mul(input[j].position, mtxViewProjection);
         output.positionL = input[j].positionL;
         s.Append(output);
      }
      s.RestartStrip();
}
```

```
struct GS_SKYBOX_OUTPUT {
   float3 positionL : POSITION;
   float4 position: SV_POSITION;
   uint renderTarget : SV_RenderTargetArrayIndex;
};
```

```
float4 PSToCubeMap(GS_SKYBOX_OUTPUT input) : SV_Target {
   float4 cColor = gtxtSkyBox.Sample(gssSkyBox, input.positionL);
   return(cColor);
}
```

```
TextureCube gtxtSkyBox : register(t13);
SamplerState gssSkyBox : register(s4);
```

```
cbuffer cbCubeMappingRender : register(b0) {
   matrix gmtxCubeMappingViews[6];
   matrix gmtxCubeMappingProjection;
};
```

큐브 매핑(Cube Mapping)

- **큐브 매핑(Cube Mapping)** – 환경 매핑(Environmental Mapping)

```
VS_SKYBOX_OUTPUT VSToCubeMap(VS_SKYBOX_INPUT input) {
   VS_SKYBOX_OUTPUT output = (VS_SKYBOX_OUTPUT)0;
   output.position = mul(float4(input.position, 1.0f), gmtxWorld);
   output.positionL = input.position;
   return(output);
}
```

```
struct VS_SKYBOX_INPUT {
   float3 position : POSITION;
};
```

```
struct VS_SKYBOX_OUTPUT {
   float3 positionL : POSITION;
   float4 position : SV_POSITION;
};
```

```
[maxvertexcount(3)]
[Instance(6)]
void GSSkyBox(triangle VS_SKYBOX_OUTPUT input[3], uint nInstance: SV_GSInstanceID, inout
TriangleStream<GS_SKYBOX_OUTPUT> stream) {
   GS_SKYBOX_OUTPUT output;
   output.renderTarget = nInstance; //+x, -x, +y, -y, +z, -z
   matrix mtxViewProjection = mul(gmtxCubeMappingViews[nInstance], gmtxCubeMappingProjection);
   for (int j = 0; j < 3; j++) {
      output.position = mul(input[j].position, mtxViewProjection);
      output.positionL = input[j].positionL;
      stream.Append(output);
   }
   stream.RestartStrip();
}
```

```
struct GS_SKYBOX_OUTPUT {
   float3 positionL : POSITION;
   float4 position: SV_POSITION;
   uint renderTarget : SV_RenderTargetArrayIndex;
};
```

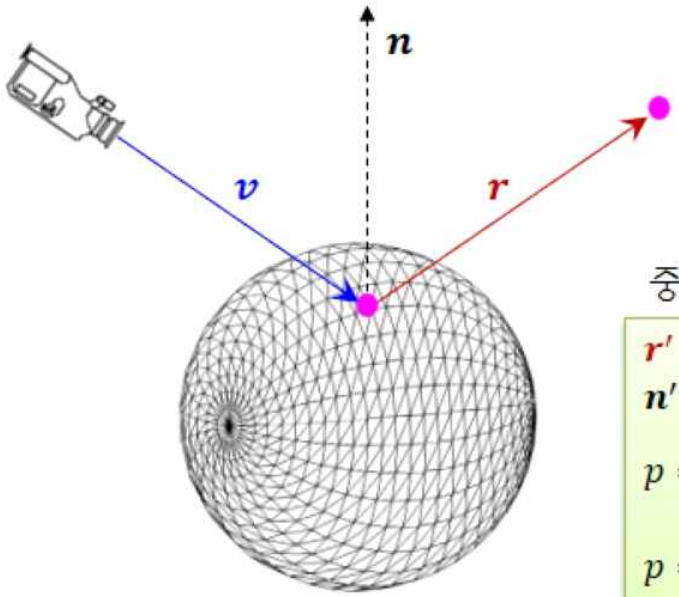
```
float4 PSToCubeMap(GS_SKYBOX_OUTPUT input) : SV_Target {
   float4 cColor = gtxtSkyBox.Sample(gssSkyBox, input.positionL);
   return(cColor);
}
```

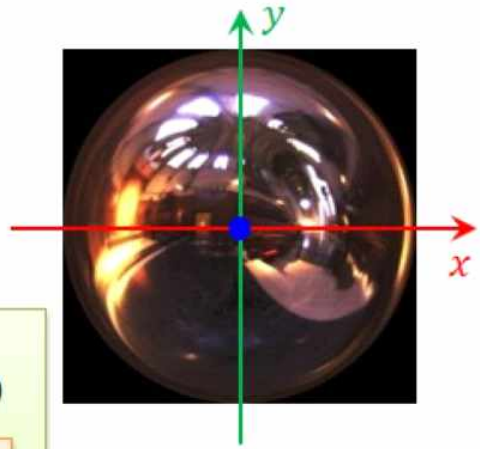
```
TextureCube gtxtSkyBox : register(t13);
SamplerState gssSkyBox : register(s4);
```

```
cbuffer cbCubeMappingRender : register(b0) {
   matrix gmtxCubeMappingViews[6];
   matrix gmtxCubeMappingProjection;
};
```

구 매핑(Sphere Mapping)

- 구 환경 매핑(Spherical Environment Mapping)
  - 구 맵(Sphere Map)
    주변의 환경을 반사하는 구를 사진을 찍어서 생성(반구만 보임)

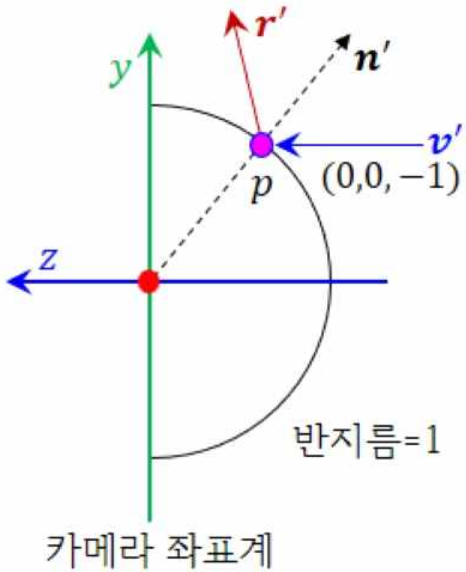




중심이 원점, 반지름이 1인 구를 가정

$$\boldsymbol{r}' = (r_x, r_y, r_z)$$
$$\boldsymbol{n}' = (r_x, r_y, r_z) - (0,0,-1) = (r_x, r_y, r_z + 1)$$
$$p = (p_x, p_y, p_z) = \frac{\boldsymbol{n}'}{|\boldsymbol{n}'|}$$
$$p = \left(\frac{r_x}{|\boldsymbol{n}'|}, \frac{r_y}{|\boldsymbol{n}'|}, \frac{r_z + 1}{|\boldsymbol{n}'|}\right)$$
$$(u, v) = \left(\frac{r_x}{2|\boldsymbol{n}'|} + 0.5, \frac{r_y}{2|\boldsymbol{n}'|} + 0.5\right)$$

$-1 \le p_x \le 1$
$-1 \le p_y \le 1$



$$B = \begin{bmatrix} r_x & r_y & r_z & 0 \\ u_x & u_y & u_z & 0 \\ l_x & l_y & l_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

텍스쳐를 생성한 카메라의 행렬

$$\boldsymbol{v}' = \boldsymbol{v}\boldsymbol{B}^{-1}$$
$$\boldsymbol{n}' = \boldsymbol{n}\boldsymbol{B}^{-1}$$
$$\boldsymbol{r}' = \boldsymbol{v}' - 2(\boldsymbol{v}' \cdot \boldsymbol{n}')\boldsymbol{n}'$$

카메라 좌표계

$$u = asin(\boldsymbol{n}_x)/\pi + 0.5$$
$$v = asin(\boldsymbol{n}_y)/\pi + 0.5$$

$$u = 0.5\boldsymbol{n}_x + 0.5$$
$$v = 0.5\boldsymbol{n}_y + 0.5$$