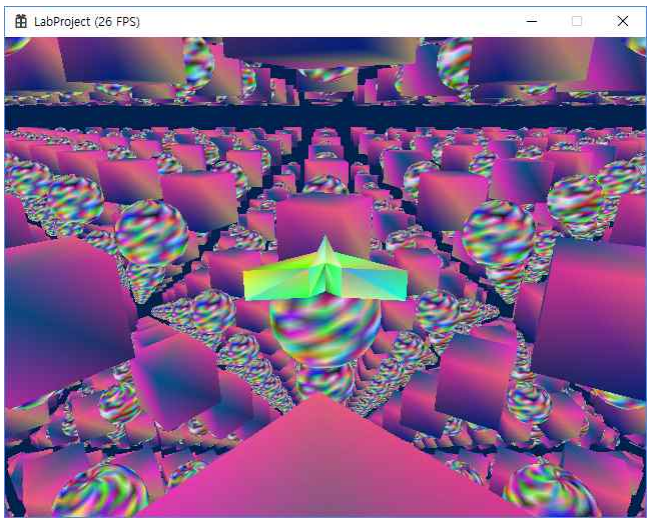
□ 예제 프로그램 17: LabProject16(픽킹: Mouse Picking)

예제 프로그램 LabProject15를 기반으로 여러 개의 직육면체들과 구들을 렌더링하고 마우스로 게임 객체를 선택(픽킹: Picking)할 수 있도록 구현한다.

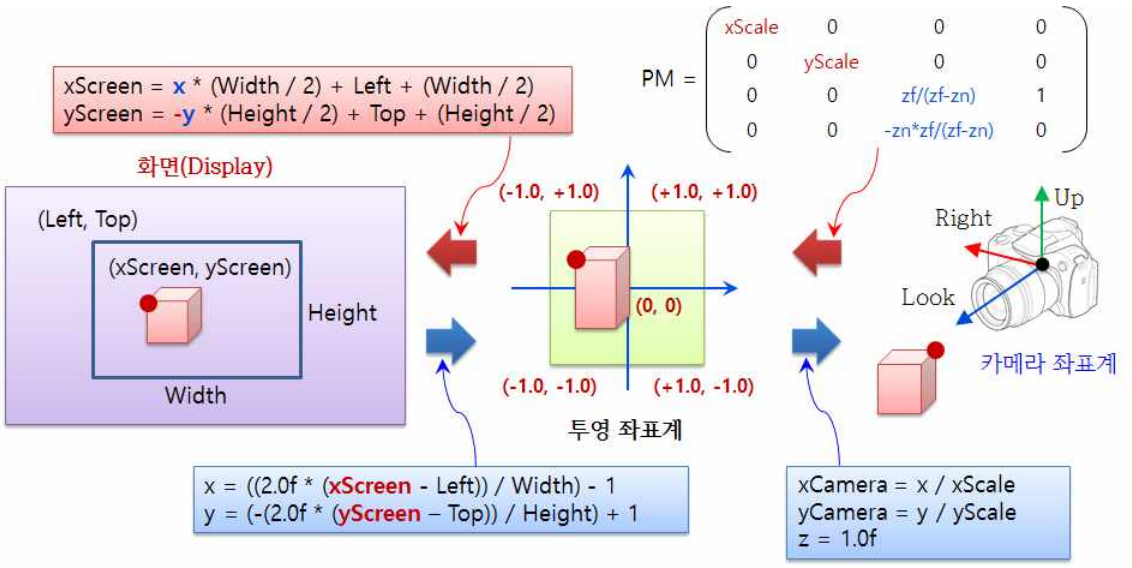


픽킹은 2D 화면 좌표계의 한 점(마우스 클릭 위치)이 주어질 때 이 점이 3D 게임 세상의 어떤 점(게임 객체)에 대응되는 가를 결정(계산)하는 것을 말한다. 렌더링 과정에서 모델 좌표계의 한 점은 다음 그림과 같이 월드 좌표 변환, 카메라 좌표 변환, 투영 변환(원근 투영 나누기), 그리고 화면 좌표 변환을 거치면 화면 좌표계의 픽셀(2D)로 대응된다.

변환 파이프라인과 역변환

화면 좌표계의 한 점(픽셀 좌표)이 주어질 때 이 점에 대응되는 3D 공간(카메라 좌표계, 월드 좌표계, 모델 좌표계)의 한 점을 얻기 위해서는 기본적으로 화면 좌표계의 한 점을 변환 파이프라인의 역변환을 사용하여 변환하면 될 것이다. 다음 그림은 화면 좌표계의 한 점 (xScreen, yScreen)을 화면 좌표 변환의 역변환을 하여 투영 좌표계의 점 (x, y)를 얻고 이 점을 투영 변환의 역변환을 하여 카메라 좌표계의 한 점 (xCamera, yCamera, 1.0)를 얻는 과정을 보여준다.

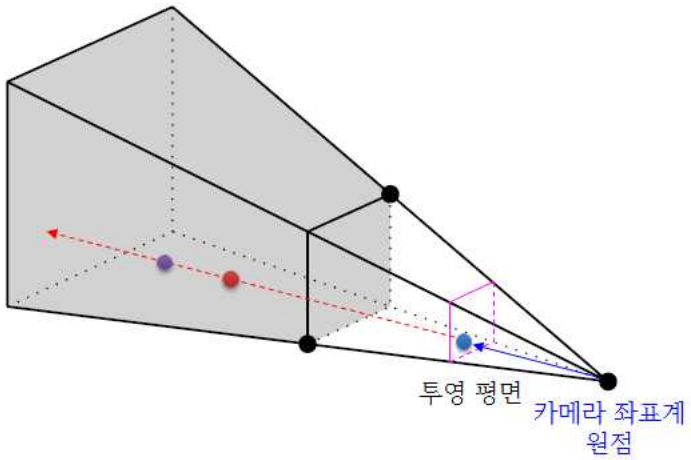
3D 카메라 좌표계의 한 점을 원근 투영 변환(원근 투영 나누기)을 하면 2D 좌표계의 점이 얻어지는데 이 때 3D 좌표계의 z-좌표 정보를 잃게 된다. 원근 투영 변환의 역변환을 하면서 잃어버린 z-좌표를 다시 얻을 수는 없다. 마우스로 선택한 화면 좌표계의 한 점(픽셀 좌표)은 3D 좌표계의 수 많은 점들이 투영된 결과이다. 하지만 사용자는 화면을 보면서 카메라에 가장 가까운 객체를 선택한 것이다. 그러므로 마우스 위치(픽셀 좌표)로 투영되는 많은 3D 점들 중에 카메라에 가장 가까운 점을 찾으면 될 것이다. 90도 FOV에서 투영 평면은 카메라에서 거리가 1만큼 떨어져 있으므로 원근 투영 변환의 역변환으로 구한 카메라 좌표계의 2D 점 (xCamera, yCamera)을 3D 점 (xCamera, yCamera, 1.0)로 설정한다.



픽킹(Picking)의 과정

이제 다음 그림과 같이 카메라 좌표계의 원점 (0, 0, 0)에서 (xCamera, yCamera, 1.0)로 가는 벡터(픽킹 광선: Ray)와 교차하는 카메라 좌표계의 모든 점들은 원근 투영을 하면 (xCamera, yCamera, 1.0)로 투영이 될 것이고 이 점을 2D로 변환하면 투영 좌표계의 점 (x, y)가 될 것이다.

투영 좌표계의 점 (x, y)로 투영이 되는 월드 좌표계의 점은 무수히 많다. 카메라 좌표계에서 픽킹 광선위의 모든 점은 점 (x, y)로 투영이 될 것이다. 카메라 좌표계의 픽킹 광선을 월드 좌표계로 역변환을 하면 월드 좌표계의 픽킹 광선을 구할 수 있다. 즉, 카메라 좌표계의 원점 (0, 0, 0)과 (xCamera, yCamera, 1.0)을 월드 좌표계로 역변환한다. 이제 월드 좌표계에서 픽킹 광선과 교차하는 객체들을 찾고 카메라에 가장 가까운 객체를 선택하면 화면 좌표계의 한 점에 대응되는 객체를 찾은 것이다.



픽킹 광선

① 새로운 프로젝트의 생성

먼저 새로운 프로젝트 LabProject16을 생성한다. “LabProjects" 솔루션을 열고 솔루션 탐색기에서 마우스 오른쪽 버튼으로 『솔루션 LabProjects』를 선택하고 메뉴에서 『추가』, 『새 프로젝트』를 차례로 선택한다. 그러면 『새 프로젝트 대화상자』가 나타난다. 그러면 프로젝트 이름 “LabProject16”을 입력하고 『확인』을 선택한다.

❶ 파일 탐색기에서 프로젝트 “LabProject15” 폴더의 다음 파일을 선택하여 프로젝트 “LabProject16” 폴더에 복사한다.

- GameFramework.h
- GameFramework.cpp
- Mesh.h
- Mesh.cpp
- Camera.h
- Camera.cpp
- Player.h
- Player.cpp
- Object.h
- Object.cpp
- Scene.h
- Scene.cpp
- Shader.h
- Shader.cpp
- stdafx.h
- Timer.h
- Timer.cpp
- Shaders.hlsl

❷ 위에서 복사한 파일을 Visual Studio 솔루션 탐색기에서 프로젝트 “LabProject16”에 추가한다. 오른쪽 마우스 버튼으로 『LabProject16』를 선택하고 『추가』, 『기존 항목』를 차례로 선택한다. 그러면 “기존 항목 추가” 대화 상자가 나타난다. 위의 파일들을 마우스로 선택(Ctrl+선택)하여 『추가』를 누르면 선택된 파일들이 프로젝트 “LabProject16”에 추가된다.

② LabProject16.cpp 파일 수정하기

이제 “LabProject16.cpp" 파일의 내용을 “LabProject15.cpp" 파일의 내용으로 바꾸도록 하자. “LabProject15.cpp" 파일의 내용 전체를 “LabProject16.cpp" 파일로 복사한다. 이제 “LabProject15.cpp" 파일에서 “LabProject15”를 “LabProject16”으로 모두 바꾼다.

그리고 “LABPROJECT15”를 “LABPROJECT16”으로 모두 바꾼다.

③ “Mesh.h” 파일 수정하기

“Mesh.h” 파일을 다음과 같이 수정한다.

❶ “CMesh” 클래스에 다음 멤버 변수를 선언한다.

```
public:
//광선과 메쉬의 교차를 검사하고 교차하는 횟수와 거리를 반환하는 함수이다.
   int CheckRayIntersection(XMFLOAT3& xmRayPosition, XMFLOAT3& xmRayDirection, float
*pfNearHitDistance);
```

❷ “CMesh” 클래스에 다음 멤버 변수를 선언한다.

```
protected:
//정점을 픽킹을 위하여 저장한다(정점 버퍼를 Map()하여 읽지 않아도 되도록).
   CDiffusedVertex     *m_pVertices = NULL;
메쉬의 인덱스를 저장한다(인덱스 버퍼를 Map()하여 읽지 않아도 되도록).
   UINT                *m_pnIndices = NULL;
```

❸ “CMesh” 클래스에 “CSphereMeshDiffused” 클래스를 다음과 같이 선언한다.

```
class CSphereMeshDiffused : public CMesh
{
public:
   CSphereMeshDiffused(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList, float fRadius=2.0f, int nSlices=20, int nStacks=20);
   virtual ~CSphereMeshDiffused();
};
```

④ “Mesh.cpp” 파일 수정하기

“Mesh.cpp” 파일을 다음과 같이 수정한다.

❶ “CMesh” 클래스의 소멸자에 다음을 추가한다.

```
   if (m_pVertices) delete[] m_pVertices;
   if (m_pnIndices) delete[] m_pnIndices;
```

❷ “CCubeMeshDiffused” 클래스의 생성자에서 “pVertices”를 “m_pVertices”로 모두 변경하고 “pnIndices”를 “m_pnIndices”로 모두 변경한다. 그리고 “pVertices”와 “pnIndices” 변수 선언 부분을 다음과 같이 수정한다.

```
   CDiffusedVertex pVertices[8];
```

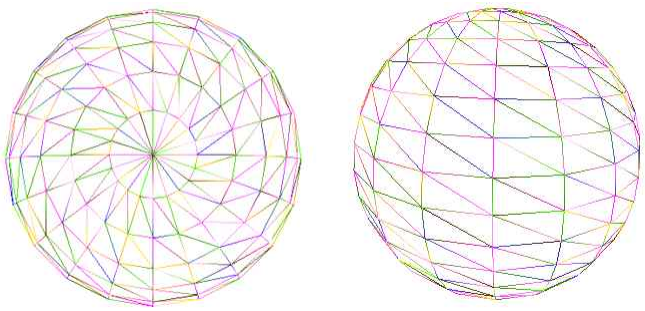
```
m_pVertices = new CDiffusedVertex[m_nVertices];

UINT pnIndices[36];
m_pnIndices = new UINT[m_nIndices];
```

❸ "CAirplaneMeshDiffused" 클래스의 생성자에서 "pVertices"를 "m_pVertices"로 모두 변경한다. 그리고 "pVertices" 변수 선언 부분을 다음과 같이 수정한다.

```
CDiffusedVertex pVertices[24 * 3];
m_pVertices = new CDiffusedVertex[m_nVertices];
```

❹ "CSphereMeshDiffused" 클래스의 생성자와 소멸자를 다음과 같이 정의한다.
구를 메쉬로 표현하기 위해 구 표면을 정점으로 나타내기 위한 좌표계가 필요하다. 구의 반지름을 r이라고 하자. 구를 xz-평면에 평행하게 nStacks번 균일하게 자르면 원기둥이 nStacks개 생긴다. 그리고 각각의 원기둥을 원기둥의 중심을 지나도록 균일하게 잘라서 nSlices개의 조각(파이 또는 피자 자른 모양)이 생기도록 한다고 가정하자. 그러면 다음 그림과 같이 구 표면은 (nStacks × nSlices)개의 사각형으로 표현된다.

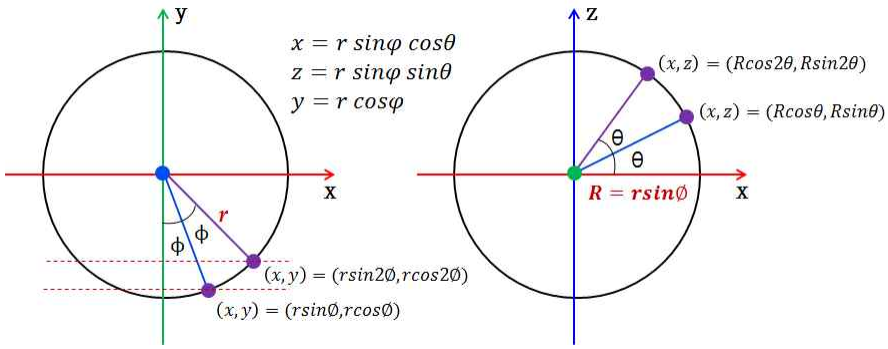


구(Sphere)

이러한 구 표면의 사각형을 표현하기 위한 좌표 (x, y, z)는 다음 그림과 같이 표현할 수 있다.

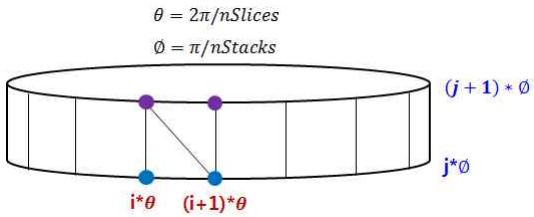
$$x = r\ sin\varphi\ cos\theta$$
$$y = r\ cos\varphi$$
$$z = r\ sin\varphi\ sin\theta$$

구를 xy-평면으로 자른 단면을 다음 그림의 왼쪽 그림이라고 하자. 구를 xz-평면에 평행하게 nStacks 번 자를 때 φ = (180도/nStacks) 만큼씩 x-축에서 각도를 증가시키면서 xz-평면에 평행하게 자르면 다음 그림의 왼쪽 그림의 점선과 같이 자르게 된다. 그러면 첫 번째로 자른 단면의 y 좌표는 (r * $cos\varphi$)가 된다. 그리고 자른 단면의 반지름은 (r * $sin\varphi$)가 된다. 이렇게 자른 단면을 다음 그림의 오른쪽 그림이라고 하자. 이 원의 반지름은 (r * $sin\varphi$)이다. 이 원의 x-축에서 반시계방향으로 θ = (360도/nSlices) 만큼씩 각도를 증가시키면서 원주 위의 점에 대한 좌표를 계산하면 $x = r\ sin\varphi\ cos\theta$, $z = r\ sin\varphi\ sin\theta$ 이다.



구 좌표계(Sphere Coordinates)

구를 xz-평면에 평행하게 자르면 다음 그림과 같은 원기둥 모양이 된다. 이 원기둥 표면의 사각형들의 정점 좌표는 다음 그림과 같이 표현할 수 있다.



구 표면의 정점 좌표

```
CSphereMeshDiffused::CSphereMeshDiffused(ID3D12Device *pd3dDevice,
ID3D12GraphicsCommandList *pd3dCommandList, float fRadius, int nSlices, int nStacks) :
CMesh(pd3dDevice, pd3dCommandList)
{
/*nSlices는 구를 xz-평면에 평행하게 몇 등분할 것인 가를 나타낸다. nStacks은 원기둥을 몇 조각으로 자를 것인가를 나타낸다.*/
   m_nStride = sizeof(CDiffusedVertex);
   m_d3dPrimitiveTopology = D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST;

/*원기둥의 표면에 있는 사각형(줄)의 개수는 {nSlices * (nStacks-2)}이다. 사각형들이 원기둥의 표면을 따라 연속되고 처음과 마지막 사각형이 연결되어 있으므로 첫 번째 원기둥을 제외하고 사각형 하나를 표현하기 위하여 하나의 정점이 필요하다. 첫 번째 원기둥은 위아래로 두 개의 정점이 필요하므로 원기둥의 표면의 사각형들을 표현하기 위하여 필요한 정점의 개수는 {(nSlices * (nStacks-1))}이다. 그런데 구의 위와 아래(구가 지구라고 가정할 때 남극과 북극)를 자르면 원기둥이 아니라 원뿔이 되므로 이 원뿔을 표현하기 위하여 2개의 정점이 더 필요하다. 그러므로 정점의 전체 개수는 {(nSlices * (nStacks-1)) + 2}이다.*/
   m_nVertices = 2 + (nSlices * (nStacks - 1));
   m_pVertices = new CDiffusedVertex[m_nVertices];

//180도를 nStacks 만큼 분할한다.
   float fDeltaPhi = float(XM_PI / nStacks);
//360도를 nSlices 만큼 분할한다.
   float fDeltaTheta = float((2.0f * XM_PI) / nSlices);

   int k = 0;
//구의 위(북극)를 나타내는 정점이다.
   m_pVertices[k++] = CDiffusedVertex(0.0f, +fRadius, 0.0f, RANDOM_COLOR);
```

```
   float theta_i, phi_j;
//원기둥 표면의 정점이다.
   for (int j = 1; j < nStacks; j++)
   {
      phi_j = fDeltaPhi * j;
      for (int i = 0; i < nSlices; i++)
      {
         theta_i = fDeltaTheta * i;
         m_pVertices[k++] = CDiffusedVertex(fRadius*sinf(phi_j)*cosf(theta_i),
fRadius*cosf(phi_j), fRadius*sinf(phi_j)*sinf(theta_i), RANDOM_COLOR);
      }
   }
//구의 아래(남극)를 나타내는 정점이다.
   m_pVertices[k] = CDiffusedVertex(0.0f, -fRadius, 0.0f, RANDOM_COLOR);

   m_pd3dVertexBuffer = ::CreateBufferResource(pd3dDevice, pd3dCommandList, m_pVertices,
m_nStride * m_nVertices, D3D12_HEAP_TYPE_DEFAULT,
D3D12_RESOURCE_STATE_VERTEX_AND_CONSTANT_BUFFER, &m_pd3dVertexUploadBuffer);

   m_d3dVertexBufferView.BufferLocation = m_pd3dVertexBuffer->GetGPUVirtualAddress();
   m_d3dVertexBufferView.StrideInBytes = m_nStride;
   m_d3dVertexBufferView.SizeInBytes = m_nStride * m_nVertices;

/*원기둥의 표면에 존재하는 사각형의 개수는 {nSlices * (nStacks-2)}이고 사각형은 2개의 삼각형으로 구성되므로 삼각형 리스트일 때 필요한 인덱스의 개수는 {nSlices * (nStacks-2) * 2 * 3}이다. 그리고 구의 위아래 원뿔의 표면에 존재하는 삼각형의 개수는 nSlices개이므로 구의 위아래 원뿔을 표현하기 위한 인덱스의 개수는 {(nSlices * 3) * 2}이다. 그러므로 구의 표면을 삼각형 리스트로 표현하기 위하여 필요한 인덱스의 개수는 {(nSlices * 3) * 2 + (nSlices * (nStacks - 2) * 3 * 2)}이다*/
   m_nIndices = (nSlices * 3) * 2 + (nSlices * (nStacks - 2) * 3 * 2);
   m_pnIndices = new UINT[m_nIndices];

   k = 0;
//구의 위쪽 원뿔의 표면을 표현하는 삼각형들의 인덱스이다.
   for (int i = 0; i < nSlices; i++)
   {
      m_pnIndices[k++] = 0;
      m_pnIndices[k++] = 1 + ((i + 1) % nSlices);
      m_pnIndices[k++] = 1 + i;
   }
//구의 원기둥의 표면을 표현하는 삼각형들의 인덱스이다.
   for (int j = 0; j < nStacks-2; j++)
   {
      for (int i = 0; i < nSlices; i++)
      {
//사각형의 첫 번째 삼각형의 인덱스이다.
         m_pnIndices[k++] = 1 + (i + (j * nSlices));
         m_pnIndices[k++] = 1 + (((i + 1) % nSlices) + (j * nSlices));
         m_pnIndices[k++] = 1 + (i + ((j + 1) * nSlices));
//사각형의 두 번째 삼각형의 인덱스이다.
         m_pnIndices[k++] = 1 + (i + ((j + 1) * nSlices));
         m_pnIndices[k++] = 1 + (((i + 1) % nSlices) + (j * nSlices));
         m_pnIndices[k++] = 1 + (((i + 1) % nSlices) + ((j + 1) * nSlices));
      }
   }
```

```
//구의 아래쪽 원뿔의 표면을 표현하는 삼각형들의 인덱스이다.
   for (int i = 0; i < nSlices; i++)
   {
      m_pnIndices[k++] = (m_nVertices - 1);
      m_pnIndices[k++] = ((m_nVertices - 1) - nSlices) + i;
      m_pnIndices[k++] = ((m_nVertices - 1) - nSlices) + ((i + 1) % nSlices);
   }

   m_pd3dIndexBuffer = ::CreateBufferResource(pd3dDevice, pd3dCommandList, m_pnIndices,
sizeof(UINT) * m_nIndices, D3D12_HEAP_TYPE_DEFAULT, D3D12_RESOURCE_STATE_INDEX_BUFFER,
&m_pd3dIndexUploadBuffer);

   m_d3dIndexBufferView.BufferLocation = m_pd3dIndexBuffer->GetGPUVirtualAddress();
   m_d3dIndexBufferView.Format = DXGI_FORMAT_R32_UINT;
   m_d3dIndexBufferView.SizeInBytes = sizeof(UINT) * m_nIndices;

   m_xmBoundingBox = BoundingOrientedBox(XMFLOAT3(0.0f, 0.0f, 0.0f), XMFLOAT3(fRadius,
fRadius, fRadius), XMFLOAT4(0.0f, 0.0f, 0.0f, 1.0f));
}

CSphereMeshDiffused::~CSphereMeshDiffused()
{
}
```

❺ “CMesh” 클래스의 CheckRayIntersection() 함수를 다음과 같이 정의한다.

```
int CMesh::CheckRayIntersection(XMFLOAT3& xmf3RayOrigin, XMFLOAT3& xmf3RayDirection,
float *pfNearHitDistance)
{
//하나의 메쉬에서 광선은 여러 개의 삼각형과 교차할 수 있다. 교차하는 삼각형들 중 가장 가까운 삼각형을 찾는다.
   int nIntersections = 0;
   BYTE *pbPositions = (BYTE *)m_pVertices;

   int nOffset = (m_d3dPrimitiveTopology == D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST) ? 3 : 1;
/*메쉬의 프리미티브(삼각형)들의 개수이다. 삼각형 리스트인 경우 (정점의 개수 / 3) 또는 (인덱스의 개수 / 3), 삼각
형 스트립의 경우 (정점의 개수 - 2) 또는 (인덱스의 개수 - 2)이다.*/
   int nPrimitives = (m_d3dPrimitiveTopology == D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST) ?
(m_nVertices / 3) : (m_nVertices - 2);
   if (m_nIndices > 0) nPrimitives = (m_d3dPrimitiveTopology ==
D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST) ? (m_nIndices / 3) : (m_nIndices - 2);

//광선은 모델 좌표계로 표현된다.
   XMVECTOR xmRayOrigin = XMLoadFloat3(&xmf3RayOrigin);
   XMVECTOR xmRayDirection = XMLoadFloat3(&xmf3RayDirection);
//모델 좌표계의 광선과 메쉬의 바운딩 박스(모델 좌표계)와의 교차를 검사한다.
   bool bIntersected = m_xmBoundingBox.Intersects(xmRayOrigin, xmRayDirection,
*pfNearHitDistance);
//모델 좌표계의 광선이 메쉬의 바운딩 박스와 교차하면 메쉬와의 교차를 검사한다.
   if (bIntersected)
   {
      float fNearHitDistance = FLT_MAX;
/*메쉬의 모든 프리미티브(삼각형)들에 대하여 픽킹 광선과의 충돌을 검사한다. 충돌하는 모든 삼각형을 찾아 광선의
시작점(실제로는 카메라 좌표계의 원점)에 가장 가까운 삼각형을 찾는다.*/
```

```
      for (int i = 0; i < nPrimitives; i++)
      {
         XMVECTOR v0 = XMLoadFloat3((XMFLOAT3 *)(pbPositions + ((m_pnIndices) ?
(m_pnIndices[(i*nOffset)+0]) : ((i*nOffset)+0)) * m_nStride));
         XMVECTOR v1 = XMLoadFloat3((XMFLOAT3 *)(pbPositions + ((m_pnIndices) ?
(m_pnIndices[(i*nOffset)+1]) : ((i*nOffset)+1)) * m_nStride));
         XMVECTOR v2 = XMLoadFloat3((XMFLOAT3 *)(pbPositions + ((m_pnIndices) ?
(m_pnIndices[(i*nOffset)+2]) : ((i*nOffset)+2)) * m_nStride));
         float fHitDistance;
         BOOL bIntersected = TriangleTests::Intersects(xmRayOrigin, xmRayDirection, v0,
v1, v2, fHitDistance);
         if (bIntersected)
         {
            if (fHitDistance < fNearHitDistance)
            {
               *pfNearHitDistance = fNearHitDistance = fHitDistance;
            }
            nIntersections++;
         }
      }
   }
   return(nIntersections);
}
```

⑤ “GameObject.h” 파일 수정하기

“GameObject.h” 파일을 다음과 같이 수정한다.

❶ “CGameObject” 클래스에 다음 멤버 함수를 선언한다.

```
public:
//모델 좌표계의 픽킹 광선을 생성한다.
   void GenerateRayForPicking(XMFLOAT3& xmf3PickPosition, XMFLOAT4X4& xmf4x4View,
XMFLOAT3 *pxmf3PickRayOrigin, XMFLOAT3 *pxmf3PickRayDirection);
//카메라 좌표계의 한 점에 대한 모델 좌표계의 픽킹 광선을 생성하고 객체와의 교차를 검사한다.
   int PickObjectByRayIntersection(XMFLOAT3& xmf3PickPosition, XMFLOAT4X4& xmf4x4View,
float *pfHitDistance);
```

⑥ “GameObject.cpp” 파일 수정하기

“GameObject.cpp” 파일을 다음과 같이 수정한다.

❶ “CGameObject” 클래스의 GenerateRayForPicking() 함수를 다음과 같이 정의한다.

```
void CGameObject::GenerateRayForPicking(XMFLOAT3& xmf3PickPosition, XMFLOAT4X4&
xmf4x4View, XMFLOAT3 *pxmf3PickRayOrigin, XMFLOAT3 *pxmf3PickRayDirection)
{
   XMFLOAT4X4 xmf4x4WorldView = Matrix4x4::Multiply(m_xmf4x4World, xmf4x4View);
   XMFLOAT4X4 xmf4x4Inverse = Matrix4x4::Inverse(xmf4x4WorldView);
```

```
   XMFLOAT3 xmf3CameraOrigin(0.0f, 0.0f, 0.0f);
//카메라 좌표계의 원점을 모델 좌표계로 변환한다.
   *pxmf3PickRayOrigin = Vector3::TransformCoord(xmf3CameraOrigin, xmf4x4Inverse);
//카메라 좌표계의 점(마우스 좌표를 역변환하여 구한 점)을 모델 좌표계로 변환한다.
   *pxmf3PickRayDirection= Vector3::TransformCoord(xmf3PickPosition, xmf4x4Inverse);
//광선의 방향 벡터를 구한다.
   *pxmf3PickRayDirection = Vector3::Normalize(Vector3::Subtract(*pxmf3PickRayDirection,
*pxmf3PickRayOrigin));
}
```

❷ “CGameObject” 클래스의 PickObjectByRayIntersection() 함수를 다음과 같이 정의한다.

```
int CGameObject::PickObjectByRayIntersection(XMFLOAT3& xmf3PickPosition, XMFLOAT4X4&
xmf4x4View, float *pfHitDistance)
{
   int nIntersected = 0;
   if (m_pMesh)
   {
      XMFLOAT3 xmf3PickRayOrigin, xmf3PickRayDirection;
//모델 좌표계의 광선을 생성한다.
      GenerateRayForPicking(xmf3PickPosition, xmf4x4View, &xmf3PickRayOrigin,
&xmf3PickRayDirection);
//모델 좌표계의 광선과 메쉬의 교차를 검사한다.
      nIntersected = m_pMesh->CheckRayIntersection(xmf3PickRayOrigin,
xmf3PickRayDirection, pfHitDistance);
   }
   return(nIntersected);
}
```

⑦ “Scene.h” 파일 변경하기

❶ “CScene” 클래스에 다음 멤버 함수를 선언한다.

```
public:
//씬의 모든 게임 객체들에 대한 마우스 픽킹을 수행한다.
   CGameObject *PickObjectPointedByCursor(int xClient, int yClient, CCamera *pCamera);
```

⑧ “Scene.cpp” 파일 변경하기

❶ “CScene” 클래스의 PickObjectPointedByCursor() 함수를 다음과 같이 정의한다.

```
CGameObject *CScene::PickObjectPointedByCursor(int xClient, int yClient, CCamera
*pCamera)
{
   if (!pCamera) return(NULL);

   XMFLOAT4X4 xmf4x4View = pCamera->GetViewMatrix();
   XMFLOAT4X4 xmf4x4Projection = pCamera->GetProjectionMatrix();
   D3D12_VIEWPORT d3dViewport = pCamera->GetViewport();
```

```
   XMFLOAT3 xmf3PickPosition;
/*화면 좌표계의 점 (xClient, yClient)를 화면 좌표 변환의 역변환과 투영 변환의 역변환을 한다. 그 결과는 카메라 좌표계의 점이다. 투영 평면이 카메라에서 z-축으로 거리가 1이므로 z-좌표는 1로 설정한다.*/
   xmf3PickPosition.x = (((2.0f * xClient) / d3dViewport.Width) - 1) /
xmf4x4Projection._11;
   xmf3PickPosition.y = -(((2.0f * yClient) / d3dViewport.Height) - 1) /
xmf4x4Projection._22;
   xmf3PickPosition.z = 1.0f;

   int nIntersected = 0;
   float fHitDistance = FLT_MAX, fNearestHitDistance = FLT_MAX;
   CGameObject *pIntersectedObject = NULL, *pNearestObject = NULL;
//셰이더의 모든 게임 객체들에 대한 마우스 픽킹을 수행하여 카메라와 가장 가까운 게임 객체를 구한다.
   for (int i = 0; i < m_nShaders; i++)
   {
      pIntersectedObject = m_pShaders[i].PickObjectByRayIntersection(xmf3PickPosition,
xmf4x4View, &fHitDistance);
      if (pIntersectedObject && (fHitDistance < fNearestHitDistance))
      {
         fNearestHitDistance = fHitDistance;
         pNearestObject = pIntersectedObject;
      }
   }
   return(pNearestObject);
}
```

⑨ “Shader.h” 파일 변경하기

❶ “CObjectsShader” 클래스에 다음 멤버 함수를 선언한다.

```
public:
//셰이더에 포함되어 있는 모든 게임 객체들에 대한 마우스 픽킹을 수행한다.
   virtual CGameObject *PickObjectByRayIntersection(XMFLOAT3& xmf3PickPosition,
XMFLOAT4X4& xmf4x4View, float *pfNearHitDistance);
```

⑩ “Shader.cpp” 파일 변경하기

❶ “CObjectsShader” 클래스의 BuildObjects() 함수를 다음과 같이 수정한다.

```
void CObjectsShader::BuildObjects(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList)
{
   CCubeMeshDiffused *pCubeMesh = new CCubeMeshDiffused(pd3dDevice, pd3dCommandList,
12.0f, 12.0f, 12.0f);
//구 메쉬를 생성한다.
   CSphereMeshDiffused *pSphereMesh = new CSphereMeshDiffused(pd3dDevice,
pd3dCommandList, 6.0f, 20, 20);

   int xObjects = 10, yObjects = 10, zObjects = 10, i = 0;
```

```
   m_nObjects = (xObjects * 2 + 1) * (yObjects * 2 + 1) * (zObjects * 2 + 1);

   m_ppObjects = new CGameObject*[m_nObjects];

   float fxPitch = 12.0f * 2.5f;
   float fyPitch = 12.0f * 2.5f;
   float fzPitch = 12.0f * 2.5f;

   CRotatingObject *pRotatingObject = NULL;
   for (int x = -xObjects; x <= xObjects; x++)
   {
      for (int y = -yObjects; y <= yObjects; y++)
      {
         for (int z = -zObjects; z <= zObjects; z++)
         {
            pRotatingObject = new CRotatingObject();
//직육면체와 구 메쉬를 교대로 배치한다.
            pRotatingObject->SetMesh((i % 2) ? (CMesh *)pCubeMesh : (CMesh *)pSphereMesh);
            pRotatingObject->SetPosition(fxPitch*x, fyPitch*y, fzPitch*z);
            pRotatingObject->SetRotationAxis(XMFLOAT3(0.0f, 1.0f, 0.0f));
            pRotatingObject->SetRotationSpeed(10.0f*(i % 10));
            m_ppObjects[i++] = pRotatingObject;
         }
      }
   }

   CreateShaderVariables(pd3dDevice, pd3dCommandList);
}
```

❷ “CObjectsShader” 클래스의 PickObjectByRayIntersection() 함수를 다음과 같이 정의한다.

```
CGameObject *CObjectsShader::PickObjectByRayIntersection(XMFLOAT3& xmf3PickPosition,
XMFLOAT4X4& xmf4x4View, float *pfNearHitDistance)
{
   int nIntersected = 0;
   *pfNearHitDistance = FLT_MAX;
   float fHitDistance = FLT_MAX;
   CGameObject *pSelectedObject = NULL;
   for (int j = 0; j < m_nObjects; j++)
   {
      nIntersected = m_ppObjects[j]->PickObjectByRayIntersection(xmf3PickPosition,
xmf4x4View, &fHitDistance);
      if ((nIntersected > 0) && (fHitDistance < *pfNearHitDistance))
      {
         *pfNearHitDistance = fHitDistance;
         pSelectedObject = m_ppObjects[j];
      }
   }
   return(pSelectedObject);
}
```

⓫ “GameFramework.h” 파일 변경하기

❶ “CGameFramework” 클래스에 다음 멤버 함수를 선언한다.

```
public:
   void ProcessSelectedObject(DWORD dwDirection, float cxDelta, float cyDelta);
```

❷ “CGameFramework” 클래스에 다음 멤버 변수를 선언한다.

```
   CGameObject         *m_pSelectedObject = NULL;
```

⑫ “GameFramework.cpp” 파일 변경하기

❶ “CGameFramework” 클래스의 ProcessSelectedObject() 함수를 다음과 같이 정의한다.

```
void CGameFramework::ProcessSelectedObject(DWORD dwDirection, float cxDelta, float
cyDelta)
{
//픽킹으로 선택한 게임 객체가 있으면 키보드를 누르거나 마우스를 움직이면 게임 개체를 이동 또는 회전한다.
   if (dwDirection != 0)
   {
      if (dwDirection & DIR_FORWARD) m_pSelectedObject->MoveForward(+1.0f);
      if (dwDirection & DIR_BACKWARD) m_pSelectedObject->MoveForward(-1.0f);
      if (dwDirection & DIR_LEFT) m_pSelectedObject->MoveStrafe(+1.0f);
      if (dwDirection & DIR_RIGHT) m_pSelectedObject->MoveStrafe(-1.0f);
      if (dwDirection & DIR_UP) m_pSelectedObject->MoveUp(+1.0f);
      if (dwDirection & DIR_DOWN) m_pSelectedObject->MoveUp(-1.0f);
   }
   else if ((cxDelta != 0.0f) || (cyDelta != 0.0f))
   {
      m_pSelectedObject->Rotate(cyDelta, cxDelta, 0.0f);
   }
}
```

❷ “CGameFramework” 클래스의 OnProcessingMouseMessage() 함수를 다음과 같이 수정한다.

```
void CGameFramework::OnProcessingMouseMessage(HWND hWnd, UINT nMessageID, WPARAM wParam,
LPARAM lParam)
{
   switch (nMessageID)
   {
      case WM_LBUTTONDOWN:
      case WM_RBUTTONDOWN:
//마우스가 눌려지면 마우스 픽킹을 하여 선택한 게임 객체를 찾는다.
         m_pSelectedObject = m_pScene->PickObjectPointedByCursor(LOWORD(lParam),
HIWORD(lParam), m_pCamera);
         ::SetCapture(hWnd);
         ::GetCursorPos(&m_ptOldCursorPos);
         break;
         ...
}
```

❸ “CGameFramework” 클래스의 ProcessInput() 함수를 다음과 같이 수정한다.

```
void CGameFramework::ProcessInput()
{
   ...
   if ((dwDirection != 0) || (cxDelta != 0.0f) || (cyDelta != 0.0f))
   {
//픽킹으로 선택한 게임 객체가 있으면 키보드를 누르거나 마우스를 움직이면 게임 개체를 이동 또는 회전한다.
      if (m_pSelectedObject)
      {
         ProcessSelectedObject(dwDirection, cxDelta, cyDelta);
      }
      else
      {
         if (cxDelta || cyDelta)
         {
            if (pKeysBuffer[VK_RBUTTON] & 0xF0)
               m_pPlayer->Rotate(cyDelta, 0.0f, -cxDelta);
            else
               m_pPlayer->Rotate(cyDelta, cxDelta, 0.0f);
         }
         if (dwDirection) m_pPlayer->Move(dwDirection, 50.0f *
m_GameTimer.GetTimeElapsed(), true);
      }
   }

   m_pPlayer->Update(m_GameTimer.GetTimeElapsed());
}
```