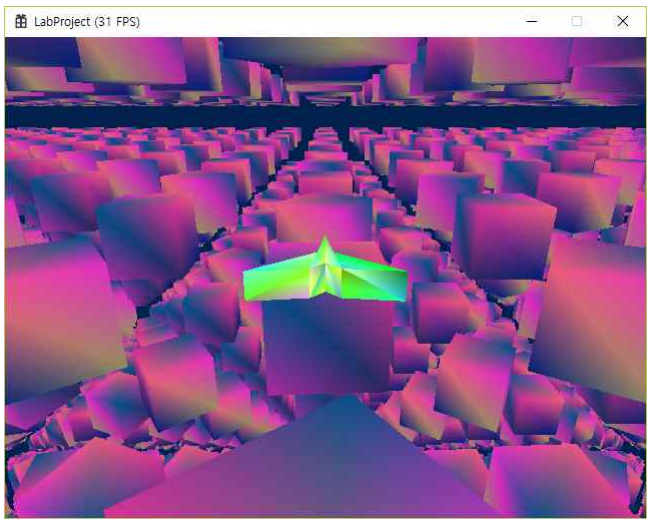
□ 예제 프로그램 12: LabProject11(카메라와 플레이어 객체)

예제 프로그램 LabProject10을 기반으로 플레이어 객체를 추가한다. 플레이어 객체가 카메라(1인칭 카메라, 3인칭 카메라, 스페이스-쉽 카메라)를 가질 수 있도록 구현한다. 3가지 카메라 중 하나를 선택하여 전환하는 기능을 추가한다. 키보드(화살표 키)를 사용하여 플레이어 객체를 이동하고 마우스를 사용하여 플레이어 객체를 회전할 수 있도록 하자.



① 새로운 프로젝트의 생성

먼저 새로운 프로젝트 LabProject11을 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 “LabProject11”을 입력하고 『확인』을 선택한다.

❶ 파일 탐색기에서 프로젝트 “LabProject10” 폴더의 다음 파일을 선택하여 프로젝트 “LabProject11” 폴더에 복사한다.

- GameFramework.h
- GameFramework.cpp
- Mesh.h
- Mesh.cpp
- Camera.h
- Camera.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 “LabProject11”에 추가한다. 오른쪽 마우스 버튼으로 『LabProject11』을 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 “LabProject11”에 추가된다.

② LabProject11.cpp 파일 수정하기

이제 “LabProject11.cpp" 파일의 내용을 “LabProject10.cpp" 파일의 내용으로 바꾸도록 하자. “LabProject10.cpp" 파일의 내용 전체를 “LabProject11.cpp" 파일로 복사한다. 이제 “LabProject11.cpp" 파일에서 “LabProject10”을 “LabProject11”로 모두 바꾼다.
그리고 “LABPROJECT10”을 “LABPROJECT11”로 모두 바꾼다.

③ “Mesh.h” 파일 수정하기

“Mesh.h” 파일을 다음과 같이 수정한다.

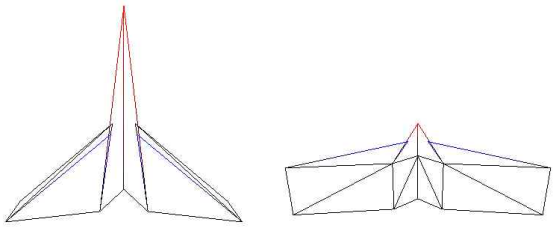
❶ “CAirplaneMeshDiffused” 클래스를 다음과 같이 선언한다.

```
class CAirplaneMeshDiffused : public CMesh
{
public:
   CAirplaneMeshDiffused(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList, float fWidth = 20.0f, float fHeight = 20.0f, float fDepth = 4.0f,
XMFLOAT4 xmf4Color = XMFLOAT4(1.0f, 1.0f, 0.0f, 0.0f));
   virtual ~CAirplaneMeshDiffused();
};
```

④ “Mesh.cpp” 파일 수정하기

“Mesh.cpp” 파일을 다음과 같이 수정한다.

❶ “CAirplaneMeshDiffused” 클래스의 생성자와 소멸자를 다음과 같이 정의한다.



비행기 메쉬

```
CAirplaneMeshDiffused::CAirplaneMeshDiffused(ID3D12Device *pd3dDevice,
ID3D12GraphicsCommandList *pd3dCommandList, float fWidth, float fHeight, float fDepth,
XMFLOAT4 xmf4Color) : CMesh(pd3dDevice, pd3dCommandList)
{
   m_nVertices = 24 * 3;
   m_nStride = sizeof(CDiffusedVertex);
   m_nOffset = 0;
   m_nSlot = 0;
   m_d3dPrimitiveTopology = D3D_PRIMITIVE_TOPOLOGY_TRIANGLELIST;

   float fx = fWidth*0.5f, fy = fHeight*0.5f, fz = fDepth*0.5f;

//위의 그림과 같은 비행기 메쉬를 표현하기 위한 정점 데이터이다.
   CDiffusedVertex pVertices[24 * 3];

   float x1 = fx * 0.2f, y1 = fy * 0.2f, x2 = fx * 0.1f, y3 = fy * 0.3f, y2 = ((y1 - (fy -
y3)) / x1) * x2 + (fy - y3);
   int i = 0;

//비행기 메쉬의 위쪽 면
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, +(fy + y3), -fz),
Vector4::Add(xmf4Color, RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x1, -y1, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, 0.0f, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, +(fy + y3), -fz),
Vector4::Add(xmf4Color, RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, 0.0f, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-x1, -y1, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x2, +y2, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+fx, -y3, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x1, -y1, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-x2, +y2, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-x1, -y1, -fz), Vector4::Add(xmf4Color,
```

```
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-fx, -y3, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

//비행기 메쉬의 아래쪽 면
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, +(fy + y3), +fz),
Vector4::Add(xmf4Color, RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, 0.0f, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x1, -y1, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, +(fy + y3), +fz),
Vector4::Add(xmf4Color, RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-x1, -y1, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, 0.0f, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x2, +y2, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x1, -y1, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+fx, -y3, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-x2, +y2, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-fx, -y3, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-x1, -y1, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

//비행기 메쉬의 오른쪽 면
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, +(fy + y3), -fz),
Vector4::Add(xmf4Color, RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, +(fy + y3), +fz),
Vector4::Add(xmf4Color, RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x2, +y2, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x2, +y2, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, +(fy + y3), +fz),
Vector4::Add(xmf4Color, RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x2, +y2, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x2, +y2, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x2, +y2, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+fx, -y3, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
```

```
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+fx, -y3, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x2, +y2, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+fx, -y3, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

//비행기 메쉬의 뒤쪽/오른쪽 면
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x1, -y1, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+fx, -y3, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+fx, -y3, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x1, -y1, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+fx, -y3, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x1, -y1, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, 0.0f, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x1, -y1, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x1, -y1, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, 0.0f, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(+x1, -y1, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, 0.0f, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

//비행기 메쉬의 왼쪽 면
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, +(fy + y3), +fz),
Vector4::Add(xmf4Color, RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, +(fy + y3), -fz),
Vector4::Add(xmf4Color, RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-x2, +y2, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, +(fy + y3), +fz),
Vector4::Add(xmf4Color, RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-x2, +y2, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-x2, +y2, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-x2, +y2, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-x2, +y2, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
```

```
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-fx, -y3, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-x2, +y2, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-fx, -y3, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-fx, -y3, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

//비행기 메쉬의 뒤쪽/왼쪽 면
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, 0.0f, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, 0.0f, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-x1, -y1, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

   pVertices[i++] = CDiffusedVertex(XMFLOAT3(0.0f, 0.0f, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-x1, -y1, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-x1, -y1, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-x1, -y1, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-x1, -y1, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-fx, -y3, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-x1, -y1, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-fx, -y3, +fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));
   pVertices[i++] = CDiffusedVertex(XMFLOAT3(-fx, -y3, -fz), Vector4::Add(xmf4Color,
RANDOM_COLOR));

   m_pd3dVertexBuffer = ::CreateBufferResource(pd3dDevice, pd3dCommandList, 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;
}

CAirplaneMeshDiffused::~CAirplaneMeshDiffused()
{
}
```

⑤ “Camera.h” 파일 수정하기

“Camera.h” 파일을 다음과 같이 수정한다.

❶ “Camera.h” 파일의 앞부분에 다음을 추가한다.

```
//카메라의 종류(모드: Mode)를 나타내는 상수를 다음과 같이 선언한다.
#define FIRST_PERSON_CAMERA          0x01
#define SPACESHIP_CAMERA             0x02
#define THIRD_PERSON_CAMERA          0x03

//프레임 버퍼의 크기와 종횡비(Aspect Ratio)를 나타내는 상수를 다음과 같이 선언한다.
#define ASPECT_RATIO   (float(FRAME_BUFFER_WIDTH) / float(FRAME_BUFFER_HEIGHT))

class CPlayer;
```

❷ “CCamera” 클래스를 다음과 같이 수정한다.

```
class CCamera
{
protected:
//카메라의 위치(월드좌표계) 벡터이다.
   XMFLOAT3                          m_xmf3Position;
//카메라의 로컬 x-축(Right), y-축(Up), z-축(Look)을 나타내는 벡터이다.*/
   XMFLOAT3                          m_xmf3Right;
   XMFLOAT3                          m_xmf3Up;
   XMFLOAT3                          m_xmf3Look;

//카메라가 x-축, z-축, y-축으로 얼마만큼 회전했는 가를 나타내는 각도이다.
   float                             m_fPitch;
   float                             m_fRoll;
   float                             m_fYaw;

//카메라의 종류(1인칭 카메라, 스페이스-쉽 카메라, 3인칭 카메라)를 나타낸다.
   DWORD                             m_nMode;

//플레이어가 바라볼 위치 벡터이다. 주로 3인칭 카메라에서 사용된다.
   XMFLOAT3                          m_xmf3LookAtWorld;
//플레이어와 카메라의 오프셋을 나타내는 벡터이다. 주로 3인칭 카메라에서 사용된다.
   XMFLOAT3                          m_xmf3Offset;
//플레이어가 회전할 때 얼마만큼의 시간을 지연시킨 후 카메라를 회전시킬 것인가를 나타낸다.
   float                             m_fTimeLag;

   XMFLOAT4X4                        m_xmf4x4View;
   XMFLOAT4X4                        m_xmf4x4Projection;

   D3D12_VIEWPORT                    m_d3dViewport;
   D3D12_RECT                        m_d3dScissorRect;

//카메라를 가지고 있는 플레이어에 대한 포인터이다.
   CPlayer                           *m_pPlayer = NULL;

public:
```

```
    CCamera();
    CCamera(CCamera *pCamera);
    virtual ~CCamera();

//카메라의 정보를 셰이더 프로그램에게 전달하기 위한 상수 버퍼를 생성하고 갱신한다.
    virtual void CreateShaderVariables(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList);
    virtual void ReleaseShaderVariables();
    virtual void UpdateShaderVariables(ID3D12GraphicsCommandList *pd3dCommandList);

//카메라 변환 행렬을 생성한다.
    void GenerateViewMatrix();
    void GenerateViewMatrix(XMFLOAT3 xmf3Position, XMFLOAT3 xmf3LookAt, XMFLOAT3 xmf3Up);
/*카메라가 여러번 회전을 하게 되면 누적된 실수 연산의 부정확성 때문에 카메라의 로컬 x-축(Right), y-축(Up), z-
축(LookAt)이 서로 직교하지 않을 수 있다. 카메라의 로컬 x-축(Right), y-축(Up), z-축(LookAt)이 서로 직교하도록
만들어준다.*/
    void RegenerateViewMatrix();

//투영 변환 행렬을 생성한다.
    void GenerateProjectionMatrix(float fNearPlaneDistance, float fFarPlaneDistance, float
fAspectRatio, float fFOVAngle);

    void SetViewport(int xTopLeft, int yTopLeft, int nWidth, int nHeight, float fMinZ =
0.0f, float fMaxZ = 1.0f);
    void SetScissorRect(LONG xLeft, LONG yTop, LONG xRight, LONG yBottom);

    virtual void SetViewportsAndScissorRects(ID3D12GraphicsCommandList *pd3dCommandList);

    void SetPlayer(CPlayer *pPlayer) { m_pPlayer = pPlayer; }
    CPlayer *GetPlayer() { return(m_pPlayer); }

    void SetMode(DWORD nMode) { m_nMode = nMode; }
    DWORD GetMode() { return(m_nMode); }

    void SetPosition(XMFLOAT3 xmf3Position) { m_xmf3Position = xmf3Position; }
    XMFLOAT3& GetPosition() { return(m_xmf3Position); }

    void SetLookAtPosition(XMFLOAT3 xmf3LookAtWorld) { m_xmf3LookAtWorld =
xmf3LookAtWorld; }
    XMFLOAT3& GetLookAtPosition() { return(m_xmf3LookAtWorld); }

    XMFLOAT3& GetRightVector() { return(m_xmf3Right); }
    XMFLOAT3& GetUpVector() { return(m_xmf3Up); }
    XMFLOAT3& GetLookVector() { return(m_xmf3Look); }

    float& GetPitch() { return(m_fPitch); }
    float& GetRoll() { return(m_fRoll); }
    float& GetYaw() { return(m_fYaw); }

    void SetOffset(XMFLOAT3 xmf3Offset) { m_xmf3Offset = xmf3Offset; }
    XMFLOAT3& GetOffset() { return(m_xmf3Offset); }

    void SetTimeLag(float fTimeLag) { m_fTimeLag = fTimeLag; }
    float GetTimeLag() { return(m_fTimeLag); }
```

```
    XMFLOAT4X4 GetViewMatrix() { return(m_xmf4x4View); }
    XMFLOAT4X4 GetProjectionMatrix() { return(m_xmf4x4Projection); }
    D3D12_VIEWPORT GetViewport() { return(m_d3dViewport); }
    D3D12_RECT GetScissorRect() { return(m_d3dScissorRect); }

//카메라를 xmf3Shift 만큼 이동한다.
    virtual void Move(XMFLOAT3& xmf3Shift) { m_xmf3Position.x += xmf3Shift.x;
m_xmf3Position.y += xmf3Shift.y; m_xmf3Position.z += xmf3Shift.z; }
//카메라를 x-축, y-축, z-축으로 회전하는 가상함수이다.
    virtual void Rotate(float fPitch = 0.0f, float fYaw = 0.0f, float fRoll = 0.0f) { }
//카메라의 이동, 회전에 따라 카메라의 정보를 갱신하는 가상함수이다.
    virtual void Update(XMFLOAT3& xmf3LookAt, float fTimeElapsed) { }
//3인칭 카메라에서 카메라가 바라보는 지점을 설정한다. 일반적으로 플레이어를 바라보도록 설정한다.
    virtual void SetLookAt(XMFLOAT3& xmf3LookAt) { }
};
```

❸ “CCamera” 클래스에서 파생된 클래스 “CSpaceShipCamera”를 다음과 같이 선언한다.

```
class CSpaceShipCamera : public CCamera
{
public:
    CSpaceShipCamera(CCamera *pCamera);
    virtual ~CSpaceShipCamera() { }

    virtual void Rotate(float fPitch = 0.0f, float fYaw = 0.0f, float fRoll = 0.0f);
};
```

❹ “CCamera” 클래스에서 파생된 클래스 “CFirstPersonCamera”를 다음과 같이 선언한다.

```
class CFirstPersonCamera : public CCamera
{
public:
    CFirstPersonCamera(CCamera *pCamera);
    virtual ~CFirstPersonCamera() { }

    virtual void Rotate(float fPitch = 0.0f, float fYaw = 0.0f, float fRoll = 0.0f);
};
```

❺ “CCamera” 클래스에서 파생 클래스 “CThirdPersonCamera”를 다음과 같이 선언한다.

```
class CThirdPersonCamera : public CCamera
{
public:
    CThirdPersonCamera(CCamera *pCamera);
    virtual ~CThirdPersonCamera() { }

    virtual void Update(XMFLOAT3& xmf3LookAt, float fTimeElapsed);
    virtual void SetLookAt(XMFLOAT3& vLookAt);
};
```

⑥ ”Camera.cpp“ 파일의 내용을 다음과 같이 수정한다.

❶ “Camera.cpp” 파일의 앞쪽에 다음과 같이 헤더 파일을 포함시키도록 변경한다.

```
#include "stdafx.h"
#include "Player.h"
#include "Camera.h“
```

❷ CCamera 클래스의 생성자를 다음과 같이 변경한다.

```
CCamera::CCamera()
{
   m_xmf4x4View = Matrix4x4::Identity();
   m_xmf4x4Projection = Matrix4x4::Identity();
   m_d3dViewport = { 0, 0, FRAME_BUFFER_WIDTH , FRAME_BUFFER_HEIGHT, 0.0f, 1.0f };
   m_d3dScissorRect = { 0, 0, FRAME_BUFFER_WIDTH , FRAME_BUFFER_HEIGHT };
   m_xmf3Position = XMFLOAT3(0.0f, 0.0f, 0.0f);
   m_xmf3Right = XMFLOAT3(1.0f, 0.0f, 0.0f);
   m_xmf3Look = XMFLOAT3(0.0f, 0.0f, 1.0f);
   m_xmf3Up = XMFLOAT3(0.0f, 1.0f, 0.0f);
   m_fPitch = 0.0f;
   m_fRoll = 0.0f;
   m_fYaw = 0.0f;
   m_xmf3Offset = XMFLOAT3(0.0f, 0.0f, 0.0f);
   m_fTimeLag = 0.0f;
   m_xmf3LookAtWorld = XMFLOAT3(0.0f, 0.0f, 0.0f);
   m_nMode = 0x00;
   m_pPlayer = NULL;
}

CCamera::CCamera(CCamera *pCamera)
{
   if (pCamera)
   {
//카메라가 이미 있으면 기존 카메라의 정보를 새로운 카메라에 복사한다.
      *this = *pCamera;
   }
   else
   {
//카메라가 없으면 기본 정보를 설정한다.
      m_xmf4x4View = Matrix4x4::Identity();
      m_xmf4x4Projection = Matrix4x4::Identity();
      m_d3dViewport = { 0, 0, FRAME_BUFFER_WIDTH , FRAME_BUFFER_HEIGHT, 0.0f, 1.0f };
      m_d3dScissorRect = { 0, 0, FRAME_BUFFER_WIDTH , FRAME_BUFFER_HEIGHT };
      m_xmf3Position = XMFLOAT3(0.0f, 0.0f, 0.0f);
      m_xmf3Right = XMFLOAT3(1.0f, 0.0f, 0.0f);
      m_xmf3Look = XMFLOAT3(0.0f, 0.0f, 1.0f);
      m_xmf3Up = XMFLOAT3(0.0f, 1.0f, 0.0f);
      m_fPitch = 0.0f;
      m_fRoll = 0.0f;
      m_fYaw = 0.0f;
      m_xmf3Offset = XMFLOAT3(0.0f, 0.0f, 0.0f);
      m_fTimeLag = 0.0f;
      m_xmf3LookAtWorld = XMFLOAT3(0.0f, 0.0f, 0.0f);
```

```
        m_nMode = 0x00;
        m_pPlayer = NULL;
    }
}
```

❸ CCamera 클래스의 GenerateViewMatrix() 함수와 RegenerateViewMatrix() 함수를 다음과 같이 변경한다.

```
void CCamera::GenerateViewMatrix(XMFLOAT3 xmf3Position, XMFLOAT3 xmf3LookAt, XMFLOAT3
xmf3Up)
{
    m_xmf3Position = xmf3Position;
    m_xmf3LookAtWorld = xmf3LookAt;
    m_xmf3Up = xmf3Up;

    GenerateViewMatrix();
}

/*카메라 변환 행렬을 생성한다. 카메라의 위치 벡터, 카메라가 바라보는 지점, 카메라의 Up 벡터(로컬 y-축 벡터)를
파라메터로 사용하는 XMMatrixLookAtLH() 함수를 사용한다.*/
void CCamera::GenerateViewMatrix()
{
    m_xmf4x4View = Matrix4x4::LookAtLH(m_xmf3Position, m_xmf3LookAtWorld, m_xmf3Up);
}

void CCamera::RegenerateViewMatrix()
{
//카메라의 z-축을 기준으로 카메라의 좌표축들이 직교하도록 카메라 변환 행렬을 갱신한다.
//카메라의 z-축 벡터를 정규화한다.
    m_xmf3Look = Vector3::Normalize(m_xmf3Look);
//카메라의 z-축과 y-축에 수직인 벡터를 x-축으로 설정한다.
    m_xmf3Right = Vector3::CrossProduct(m_xmf3Up, m_xmf3Look, true);
//카메라의 z-축과 x-축에 수직인 벡터를 y-축으로 설정한다.
    m_xmf3Up = Vector3::CrossProduct(m_xmf3Look, m_xmf3Right, true);

    m_xmf4x4View._11 = m_xmf3Right.x; m_xmf4x4View._12 = m_xmf3Up.x; m_xmf4x4View._13 =
m_xmf3Look.x;
    m_xmf4x4View._21 = m_xmf3Right.y; m_xmf4x4View._22 = m_xmf3Up.y; m_xmf4x4View._23 =
m_xmf3Look.y;
    m_xmf4x4View._31 = m_xmf3Right.z; m_xmf4x4View._32 = m_xmf3Up.z; m_xmf4x4View._33 =
m_xmf3Look.z;
    m_xmf4x4View._41 = -Vector3::DotProduct(m_xmf3Position, m_xmf3Right);
    m_xmf4x4View._42 = -Vector3::DotProduct(m_xmf3Position, m_xmf3Up);
    m_xmf4x4View._43 = -Vector3::DotProduct(m_xmf3Position, m_xmf3Look);
}
```

m_xmf3Right.x	m_xmf3Right.y	m_xmf3Right.z	0
m_xmf3Up.x	m_xmf3Up.y	m_xmf3Up.z	0
m_xmf3Look.x	m_xmf3Look.y	m_xmf3Look.z	0
m_xmf3Position.x	m_xmf3Position.y	m_xmf3Position.z	1

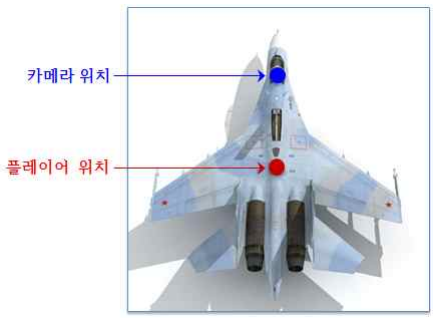
월드좌표계에서 카메라의 위치와 방향을 나타내는 행렬

m_xmf3Right.x	m_xmf3Up.x	m_xmf3Look.x	0
m_xmf3Right.y	m_xmf3Up.y	m_xmf3Look.y	0
m_xmf3Right.z	m_xmf3Up.z	m_xmf3Look.z	0
-(m_xmf3Position • m_xmf3Right)	-(m_xmf3Position • m_xmf3Up)	-(m_xmf3Position • m_xmf3Look)	1

카메라 변환 행렬

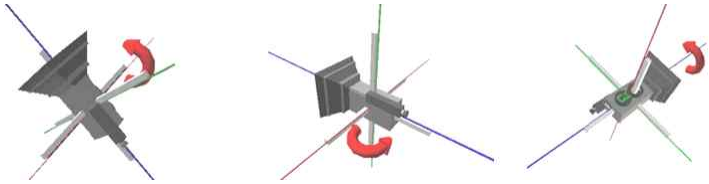
❹ 스페이스-쉽 카메라를 표현하기 위한 "CSpaceShipCamera" 클래스의 생성자를 다음과 같이 정의한다. 일반적으로 스페이스-쉽 카메라는 다음 그림과 같이 플레이어 객체에 물리적으로 고정된다. 그러므로 카메라의 이동과 회전은 플레이어 객체의 이동과 회전과 일치하게 된다.

```
CSpaceShipCamera::CSpaceShipCamera(CCamera *pCamera) : CCamera(pCamera)
{
   m_nMode = SPACESHIP_CAMERA;
}
```



플레이어와 스페이스-쉽 카메라

❺ "CSpaceShipCamera" 클래스의 Rotate() 함수를 다음과 같이 정의한다.
일반적인 스페이스-쉽 카메라의 회전은 플레이어를 회전하여 이루어진다. 스페이스-쉽 카메라를 따로 회전하는 것을 허용하지 않는다. 플레이어의 로컬 x-축을 기준으로 플레이어를 회전하면 카메라가 같은 회전을 하게 되므로 카메라의 로컬 x-축, y-축, z-축이 바뀐다. 플레이어의 중심 위치와 카메라의 중심 위치가 다르면 플레이어의 회전(자전)에 의해 카메라는 공전을 하게 된다.

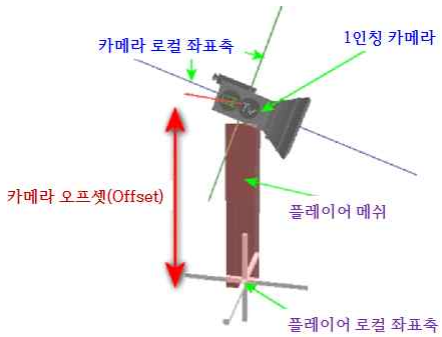


카메라의 회전(Pitch, Yaw, Roll)

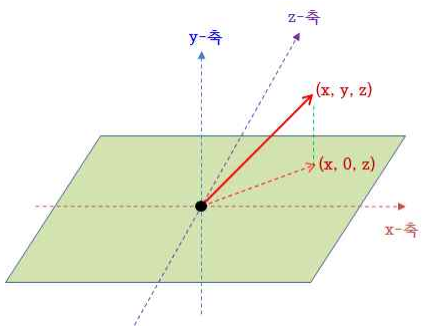
```
//스페이스-쉽 카메라를 플레이어의 로컬 x-축(Right), y-축(Up), z-축(Look)을 기준으로 회전하는 함수이다.
void CSpaceShipCamera::Rotate(float x, float y, float z)
{
   if (m_pPlayer && (x != 0.0f))
   {
//플레이어의 로컬 x-축에 대한 x 각도의 회전 행렬을 계산한다.
      XMFLOAT3 xmf3Right = m_pPlayer->GetRightVector();
      XMMATRIX xmmtxRotate = XMMatrixRotationAxis(XMLoadFloat3(&xmf3Right),
XMConvertToRadians(x));
//카메라의 로컬 x-축, y-축, z-축을 회전한다.
      m_xmf3Right = Vector3::TransformNormal(m_xmf3Right, xmmtxRotate);
      m_xmf3Up = Vector3::TransformNormal(m_xmf3Up, xmmtxRotate);
      m_xmf3Look = Vector3::TransformNormal(m_xmf3Look, xmmtxRotate);
/*카메라의 위치 벡터에서 플레이어의 위치 벡터를 뺀다. 결과는 플레이어 위치를 기준(원점)으로 한 카메라의 위치
벡터이다.*/
      m_xmf3Position = Vector3::Subtract(m_xmf3Position, m_pPlayer->GetPosition());
//플레이어의 위치를 중심으로 카메라의 위치 벡터(플레이어를 기준으로 한)를 회전한다.
      m_xmf3Position = Vector3::TransformCoord(m_xmf3Position, xmmtxRotate);
//회전시킨 카메라의 위치 벡터에 플레이어의 위치를 더하여 카메라의 위치 벡터를 구한다.
      m_xmf3Position = Vector3::Add(m_xmf3Position, m_pPlayer->GetPosition());
   }
   if (m_pPlayer && (y != 0.0f))
   {
      XMFLOAT3 xmf3Up = m_pPlayer->GetUpVector();
      XMMATRIX xmmtxRotate = XMMatrixRotationAxis(XMLoadFloat3(&xmf3Up),
XMConvertToRadians(y));
      m_xmf3Right = Vector3::TransformNormal(m_xmf3Right, xmmtxRotate);
      m_xmf3Up = Vector3::TransformNormal(m_xmf3Up, xmmtxRotate);
      m_xmf3Look = Vector3::TransformNormal(m_xmf3Look, xmmtxRotate);
      m_xmf3Position = Vector3::Subtract(m_xmf3Position, m_pPlayer->GetPosition());
      m_xmf3Position = Vector3::TransformCoord(m_xmf3Position, xmmtxRotate);
      m_xmf3Position = Vector3::Add(m_xmf3Position, m_pPlayer->GetPosition());
   }
   if (m_pPlayer && (z != 0.0f))
   {
      XMFLOAT3 xmf3Look = m_pPlayer->GetLookVector();
      XMMATRIX xmmtxRotate = XMMatrixRotationAxis(XMLoadFloat3(&xmf3Look),
XMConvertToRadians(z));
      m_xmf3Right = Vector3::TransformNormal(m_xmf3Right, xmmtxRotate);
      m_xmf3Up = Vector3::TransformNormal(m_xmf3Up, xmmtxRotate);
      m_xmf3Look = Vector3::TransformNormal(m_xmf3Look, xmmtxRotate);
      m_xmf3Position = Vector3::Subtract(m_xmf3Position, m_pPlayer->GetPosition());
      m_xmf3Position = Vector3::TransformCoord(m_xmf3Position, xmmtxRotate);
      m_xmf3Position = Vector3::Add(m_xmf3Position, m_pPlayer->GetPosition());
   }
}
```

❻ 1인칭 카메라를 표현하기 위한 “CFirstPersonCamera” 클래스의 생성자를 다음과 같이 정의한다. 일반적으로 1인칭 카메라는 사람의 눈을 의미한다. 1인칭 카메라의 y-축과 z-축의 회전은 플레이어를 기준으로 회전을 하지만 x-축으로는 카메라를 기준으로 회전(사람은 목을 회전하는 것과 유사)을 할 수 있다.

```
CFirstPersonCamera::CFirstPersonCamera(CCamera *pCamera) : CCamera(pCamera)
{
   m_nMode = FIRST_PERSON_CAMERA;
   if (pCamera)
   {
/*1인칭 카메라로 변경하기 이전의 카메라가 스페이스-쉽 카메라이면 카메라의 Up 벡터를 월드좌표의 y-축이 되도록 한다. 이것은 스페이스-쉽 카메라의 로컬 y-축 벡터가 어떤 방향이든지 1인칭 카메라(대부분 사람인 경우)의 로컬 y-축 벡터가 월드좌표의 y-축이 되도록 즉, 똑바로 서있는 형태로 설정한다는 의미이다. 그리고 로컬 x-축 벡터와 로컬 z-축 벡터의 y-좌표가 0.0f가 되도록 한다. 이것은 다음 그림과 같이 로컬 x-축 벡터와 로컬 z-축 벡터를 xz-평면(지면)으로 투영하는 것을 의미한다. 즉, 1인칭 카메라의 로컬 x-축 벡터와 로컬 z-축 벡터는 xz-평면에 평행하다.*/
      if (pCamera->GetMode() == SPACESHIP_CAMERA)
      {
         m_xmf3Up = XMFLOAT3(0.0f, 1.0f, 0.0f);
         m_xmf3Right.y = 0.0f;
         m_xmf3Look.y = 0.0f;
         m_xmf3Right = Vector3::Normalize(m_xmf3Right);
         m_xmf3Look = Vector3::Normalize(m_xmf3Look);
      }
   }
}
```



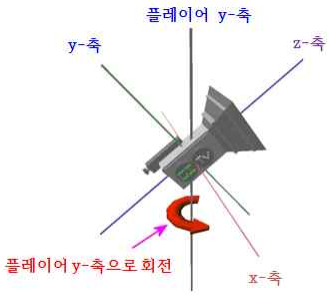
플레이어와 1인칭 카메라



xz-평면으로의 투영

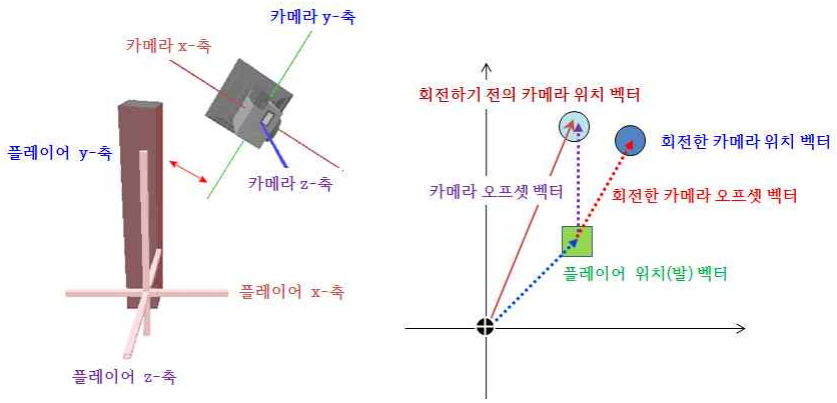
❼ “CFirstPersonCamera” 클래스의 Rotate() 함수를 다음과 같이 정의한다.

1인칭 카메라의 x-축 회전은 카메라의 로컬 x-축(Right 벡터)으로 회전하는 것이다. 1인칭 카메라의 y-축 회전은 플레이어의 로컬 y-축(Up 벡터)으로 회전하는 것이다. 1인칭 카메라의 z-축 회전은 플레이어의 로컬 z-축(Look 벡터)을 기준으로 회전시킨다. 1인칭 카메라의 x-축 회전은 고개를 위/아래 로 움직이는 것에 해당한다(플레이어의 몸 전체를 x-축으로 회전하는 것이 아니라 고개를 들고 숙이는 것이다). 1인칭 카메라의 y-축 회전은 다음 그림과 같이 플레이어 좌표계의 y-축으로 회전하는 것이다(월드 좌표계의 y-축을 사용할 수도 있다).



1인칭 카메라의 y-축 회전

1인칭 카메라의 z-축 회전은 다음 그림과 같이 플레이어의 로컬 z-축을 기준으로 회전하는 것이다(FPS 게임에서 이러한 회전을 특별히 린(Lean)이라고 한다. 플레이어의 몸통은 벽에 기대어 감추고 고개를 살짝 내밀어 적을 살피는 동작에 해당한다. 실제 몸통이나 고개의 회전은 일어나지 않는다).



1인칭 카메라의 z-축 회전

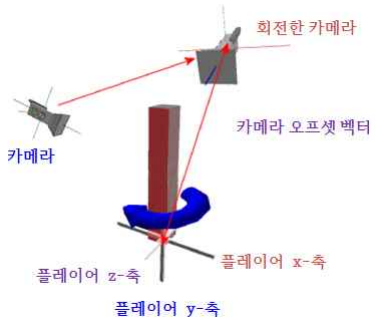
```
void CFirstPersonCamera::Rotate(float x, float y, float z)
{
   if (x != 0.0f)
   {
//카메라의 로컬 x-축을 기준으로 회전하는 행렬을 생성한다. 사람의 경우 고개를 끄떡이는 동작이다.
      XMMATRIX xmmtxRotate = XMMatrixRotationAxis(XMLoadFloat3(&m_xmf3Right),
```

```
XMConvertToRadians(x));
//카메라의 로컬 x-축, y-축, z-축을 회전 행렬을 사용하여 회전한다.
      m_xmf3Look = Vector3::TransformNormal(m_xmf3Look, xmmtxRotate);
      m_xmf3Up = Vector3::TransformNormal(m_xmf3Up, xmmtxRotate);
      m_xmf3Right = Vector3::TransformNormal(m_xmf3Right, xmmtxRotate);
   }
   if (m_pPlayer && (y != 0.0f))
   {
//플레이어의 로컬 y-축을 기준으로 회전하는 행렬을 생성한다.
      XMFLOAT3 xmf3Up = m_pPlayer->GetUpVector();
      XMMATRIX xmmtxRotate = XMMatrixRotationAxis(XMLoadFloat3(&xmf3Up),
XMConvertToRadians(y));
//카메라의 로컬 x-축, y-축, z-축을 회전 행렬을 사용하여 회전한다.
      m_xmf3Look = Vector3::TransformNormal(m_xmf3Look, xmmtxRotate);
      m_xmf3Up = Vector3::TransformNormal(m_xmf3Up, xmmtxRotate);
      m_xmf3Right = Vector3::TransformNormal(m_xmf3Right, xmmtxRotate);
   }
   if (m_pPlayer && (z != 0.0f))
   {
//플레이어의 로컬 z-축을 기준으로 회전하는 행렬을 생성한다.
      XMFLOAT3 xmf3Look = m_pPlayer->GetLookVector();
      XMMATRIX xmmtxRotate = XMMatrixRotationAxis(XMLoadFloat3(&xmf3Look),
XMConvertToRadians(z));
 //카메라의 위치 벡터를 플레이어 좌표계로 표현한다(오프셋 벡터).
      m_xmf3Position = Vector3::Subtract(m_xmf3Position, m_pPlayer->GetPosition());
 //오프셋 벡터 벡터를 회전한다.
      m_xmf3Position = Vector3::TransformCoord(m_xmf3Position, xmmtxRotate);
 //회전한 카메라의 위치를 월드 좌표계로 표현한다.
      m_xmf3Position = Vector3::Add(m_xmf3Position, m_pPlayer->GetPosition());
//카메라의 로컬 x-축, y-축, z-축을 회전한다.
      m_xmf3Look = Vector3::TransformNormal(m_xmf3Look, xmmtxRotate);
      m_xmf3Up = Vector3::TransformNormal(m_xmf3Up, xmmtxRotate);
      m_xmf3Right = Vector3::TransformNormal(m_xmf3Right, xmmtxRotate);
   }
}
```

❽ 3인칭 카메라를 표현하기 위한 “CThirdPersonCamera” 클래스의 생성자를 다음과 같이 정의한다. 일반적으로 3인칭 카메라는 제 3자의 눈으로 플레이어를 바라보는 것을 의미한다. 보통 플레이어에서 일정한 거리(카메라 오프셋)를 두고 플레이어의 일정한 지점(LookAt 벡터)을 바라보도록 한다. 3인칭 카메라의 움직임은 플레이어와 카메라를 막대기로 연결하여 고정한 것으로 상상하면 이해하기 쉽다. 플레이어를 움직이면 카메라는 자동적으로(막대로 고정되어 있으므로) 따라서 움직이게 된다. 카메라만 따로 움직일 수는 없다.

```
CThirdPersonCamera::CThirdPersonCamera(CCamera *pCamera) : CCamera(pCamera)
{
   m_nMode = THIRD_PERSON_CAMERA;
   if (pCamera)
   {
/*3인칭 카메라로 변경하기 이전의 카메라가 스페이스-쉽 카메라이면 카메라의 Up 벡터를 월드좌표의 y-축이 되도록 한다. 이것은 스페이스-쉽 카메라의 로컬 y-축 벡터가 어떤 방향이든지 3인칭 카메라(대부분 사람인 경우)의 로컬 y-
```

```
축 벡터가 월드좌표의 y-축이 되도록 즉, 똑바로 서있는 형태로 설정한다는 의미이다. 그리고 로컬 x-축 벡터와 로컬 z-축 벡터의 y-좌표가 0.0f가 되도록 한다. 이것은 로컬 x-축 벡터와 로컬 z-축 벡터를 xz-평면(지면)으로 투영하는 것을 의미한다. 즉, 3인칭 카메라의 로컬 x-축 벡터와 로컬 z-축 벡터는 xz-평면에 평행하다.*/
        if (pCamera->GetMode() == SPACESHIP_CAMERA)
        {
            m_xmf3Up = XMFLOAT3(0.0f, 1.0f, 0.0f);
            m_xmf3Right.y = 0.0f;
            m_xmf3Look.y = 0.0f;
            m_xmf3Right = Vector3::Normalize(m_xmf3Right);
            m_xmf3Look = Vector3::Normalize(m_xmf3Look);
        }
    }
}
```



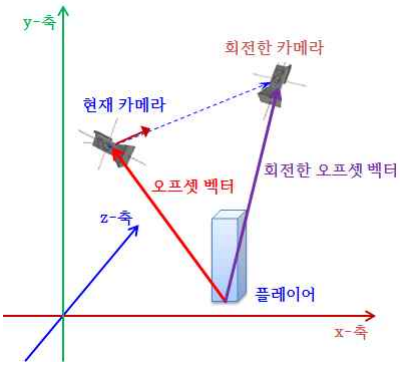
3인칭 카메라의 회전

❾ “CThirdPersonCamera” 클래스의 Update() 함수를 다음과 같이 정의한다.

3인칭 카메라의 회전은 플레이어를 회전하는 것이다. 플레이어의 회전은 앞의 그림과 같이 플레이어의 로컬 y-축을 중심으로 하는 회전만 가능하다. 플레이어가 회전하면 카메라는 플레이어의 y-축을 중심으로 회전(공전)을 한 다음 플레이어를 바라보아야 한다.

```
void CThirdPersonCamera::Update(XMFLOAT3& xmf3LookAt, float fTimeElapsed)
{
/*플레이어가 있으면 플레이어의 회전에 따라 3인칭 카메라도 회전해야 한다.
    if (m_pPlayer)
    {
        XMFLOAT4X4 xmf4x4Rotate = Matrix4x4::Identity();
        XMFLOAT3 xmf3Right = m_pPlayer->GetRightVector();
        XMFLOAT3 xmf3Up = m_pPlayer->GetUpVector();
        XMFLOAT3 xmf3Look = m_pPlayer->GetLookVector();
//플레이어의 로컬 x-축, y-축, z-축 벡터로부터 회전 행렬(플레이어와 같은 방향을 나타내는 행렬)을 생성한다.
        xmf4x4Rotate._11 = xmf3Right.x; xmf4x4Rotate._21 = xmf3Up.x; xmf4x4Rotate._31 =
xmf3Look.x;
        xmf4x4Rotate._12 = xmf3Right.y; xmf4x4Rotate._22 = xmf3Up.y; xmf4x4Rotate._32 =
xmf3Look.y;
        xmf4x4Rotate._13 = xmf3Right.z; xmf4x4Rotate._23 = xmf3Up.z; xmf4x4Rotate._33 =
xmf3Look.z;

//카메라 오프셋 벡터를 회전 행렬로 변환(회전)한다.
        XMFLOAT3 xmf3Offset = Vector3::TransformCoord(m_xmf3Offset, xmf4x4Rotate);
//회전한 카메라의 위치는 플레이어의 위치에 회전한 카메라 오프셋 벡터를 더한 것이다.
```

```
		XMFLOAT3 xmf3Position = Vector3::Add(m_pPlayer->GetPosition(), xmf3Offset);
//현재의 카메라의 위치에서 회전한 카메라의 위치까지의 방향과 거리를 나타내는 벡터이다.
		XMFLOAT3 xmf3Direction = Vector3::Subtract(xmf3Position, m_xmf3Position);
		float fLength = Vector3::Length(xmf3Direction);
		xmf3Direction = Vector3::Normalize(xmf3Direction);
/*3인칭 카메라의 래그(Lag)는 플레이어가 회전하더라도 카메라가 동시에 따라서 회전하지 않고 약간의 시차를 두고 회전하는 효과를 구현하기 위한 것이다. m_fTimeLag가 1보다 크면 fTimeLagScale이 작아지고 실제 회전(이동)이 적게 일어날 것이다. m_fTimeLag가 0이 아닌 경우 fTimeElapsed를 곱하고 있으므로 3인칭 카메라는 1초의 시간동안 (1.0f / m_fTimeLag)의 비율만큼 플레이어의 회전을 따라가게 될 것이다.*/
		float fTimeLagScale = (m_fTimeLag) ? fTimeElapsed * (1.0f / m_fTimeLag) : 1.0f;
		float fDistance = fLength * fTimeLagScale;
		if (fDistance > fLength) fDistance = fLength;
		if (fLength < 0.01f) fDistance = fLength;
		if (fDistance > 0)
		{
//실제로 카메라를 회전하지 않고 이동을 한다(회전의 각도가 작은 경우 회전 이동은 선형 이동과 거의 같다).
			m_xmf3Position = Vector3::Add(m_xmf3Position, xmf3Direction, fDistance);
//카메라가 플레이어를 바라보도록 한다.
			SetLookAt(xmf3LookAt);
		}
	}
}
```



3인칭 카메라의 지연 회전

❿ “CThirdPersonCamera” 클래스의 SetLookAt() 함수를 다음과 같이 정의한다.

3인칭 카메라를 회전시키면 카메라의 위치와 방향이 바뀌게 된다. 그리고 카메라가 플레이어를 바라보도록 해야 하므로 새로운 카메라의 위치 벡터를 사용하여 카메라의 방향 벡터(Right, Up, Look 벡터)를 다시 지정한다. 이를 위해 카메라 변환 행렬을 사용한다. 카메라 변환 행렬의 첫 번째 열 벡터가 Right 벡터, 두 번째 열 벡터가 Up 벡터, 세 번째 열 벡터가 Look 벡터가 된다.

```
void CThirdPersonCamera::SetLookAt(XMFLOAT3& xmf3LookAt)
{
//현재 카메라의 위치에서 플레이어를 바라보기 위한 카메라 변환 행렬을 생성한다.
	XMFLOAT4X4 mtxLookAt = Matrix4x4::LookAtLH(m_xmf3Position, xmf3LookAt,
m_pPlayer->GetUpVector());
//카메라 변환 행렬에서 카메라의 x-축, y-축, z-축을 구한다.
	m_xmf3Right = XMFLOAT3(mtxLookAt._11, mtxLookAt._21, mtxLookAt._31);
	m_xmf3Up = XMFLOAT3(mtxLookAt._12, mtxLookAt._22, mtxLookAt._32);
```

```
    m_xmf3Look = XMFLOAT3(mtxLookAt._13, mtxLookAt._23, mtxLookAt._33);
}
```

⑦ ”CPlayer” 클래스 생성하기

게임 프로그램의 기본적 요소를 추가하기 위해 ”CPlayer” 클래스를 생성한다. 솔루션 탐색기에서 오른쪽 마우스 버튼으로 ”LabProject10“을 선택하고 『추가』, 『클래스』를 차례로 선택한다. 클래스 추가 대화상자가 나타나면 ”설치된 템플릿“에서 『C++』을 선택하고 『C++ 클래스』를 선택한 후 『추가』 버튼을 누른다. ”일반 C++ 클래스 마법사“가 나타나면 『클래스 이름』에 ”CPlayer”를 입력한다. 그러면 “.h 파일”의 이름이 “Player.h” 그리고 “.cpp 파일”의 이름이 “Player.cpp”가 된다. 『마침』을 선택하면 클래스 마법사가 두 개의 파일을 프로젝트에 추가하여 준다.

⑧ “Player.h” 파일 수정하기

❶ “Player.h” 파일의 앞부분에 다음을 추가한다.

```
#define DIR_FORWARD                 0x01
#define DIR_BACKWARD                0x02
#define DIR_LEFT                    0x04
#define DIR_RIGHT                   0x08
#define DIR_UP                      0x10
#define DIR_DOWN                    0x20

#include "Object.h"
#include "Camera.h"
```

❷ “CPlayer” 클래스를 다음과 같이 수정한다.

```
class CPlayer : public CGameObject
{
protected:
//플레이어의 위치 벡터, x-축(Right), y-축(Up), z-축(Look) 벡터이다.
   XMFLOAT3                     m_xmf3Position;
   XMFLOAT3                     m_xmf3Right;
   XMFLOAT3                     m_xmf3Up;
   XMFLOAT3                     m_xmf3Look;

//플레이어가 로컬 x-축(Right), y-축(Up), z-축(Look)으로 얼마만큼 회전했는가를 나타낸다.
   float                        m_fPitch;
   float                        m_fYaw;
   float                        m_fRoll;

//플레이어의 이동 속도를 나타내는 벡터이다.
   XMFLOAT3                     m_xmf3Velocity;
//플레이어에 작용하는 중력을 나타내는 벡터이다.
   XMFLOAT3                     m_xmf3Gravity;
//xz-평면에서 (한 프레임 동안) 플레이어의 이동 속력의 최대값을 나타낸다.
   float                        m_fMaxVelocityXZ;
```

```
//y-축 방향으로 (한 프레임 동안) 플레이어의 이동 속력의 최대값을 나타낸다.
   float                        m_fMaxVelocityY;
//플레이어에 작용하는 마찰력을 나타낸다.
   float                        m_fFriction;

//플레이어의 위치가 바뀔 때마다 호출되는 OnPlayerUpdateCallback() 함수에서 사용하는 데이터이다.
   LPVOID                       m_pPlayerUpdatedContext;
//카메라의 위치가 바뀔 때마다 호출되는 OnCameraUpdateCallback() 함수에서 사용하는 데이터이다.
   LPVOID                       m_pCameraUpdatedContext;

//플레이어에 현재 설정된 카메라이다.
   CCamera                      *m_pCamera = NULL;

public:
   CPlayer();
   virtual ~CPlayer();

   XMFLOAT3 GetPosition() { return(m_xmf3Position); }
   XMFLOAT3 GetLookVector() { return(m_xmf3Look); }
   XMFLOAT3 GetUpVector() { return(m_xmf3Up); }
   XMFLOAT3 GetRightVector() { return(m_xmf3Right); }

   void SetFriction(float fFriction) { m_fFriction = fFriction; }
   void SetGravity(XMFLOAT3& xmf3Gravity) { m_xmf3Gravity = xmf3Gravity; }
   void SetMaxVelocityXZ(float fMaxVelocity) { m_fMaxVelocityXZ = fMaxVelocity; }
   void SetMaxVelocityY(float fMaxVelocity) { m_fMaxVelocityY = fMaxVelocity; }
   void SetVelocity(XMFLOAT3& xmf3Velocity) { m_xmf3Velocity = xmf3Velocity; }

/*플레이어의 위치를 xmf3Position 위치로 설정한다. xmf3Position 벡터에서 현재 플레이어의 위치 벡터를 빼면 현재 플레이어의 위치에서 xmf3Position 방향으로의 벡터가 된다. 현재 플레이어의 위치에서 이 벡터 만큼 이동한다.*/
   void SetPosition(XMFLOAT3& xmf3Position) { Move(XMFLOAT3(xmf3Position.x - m_xmf3Position.x, xmf3Position.y - m_xmf3Position.y, xmf3Position.z - m_xmf3Position.z), false); }

   XMFLOAT3& GetVelocity() { return(m_xmf3Velocity); }
   float GetYaw() { return(m_fYaw); }
   float GetPitch() { return(m_fPitch); }
   float GetRoll() { return(m_fRoll); }

   CCamera *GetCamera() { return(m_pCamera); }
   void SetCamera(CCamera *pCamera) { m_pCamera = pCamera; }

//플레이어를 이동하는 함수이다.
   void Move(ULONG nDirection, float fDistance, bool bVelocity = false);
   void Move(const XMFLOAT3& xmf3Shift, bool bVelocity = false);
   void Move(float fxOffset = 0.0f, float fyOffset = 0.0f, float fzOffset = 0.0f);
//플레이어를 회전하는 함수이다.
   void Rotate(float x, float y, float z);

//플레이어의 위치와 회전 정보를 경과 시간에 따라 갱신하는 함수이다.
   void Update(float fTimeElapsed);

//플레이어의 위치가 바뀔 때마다 호출되는 함수와 그 함수에서 사용하는 정보를 설정하는 함수이다.
```

```
   virtual void OnPlayerUpdateCallback(float fTimeElapsed) { }
   void SetPlayerUpdatedContext(LPVOID pContext) { m_pPlayerUpdatedContext = pContext; }
//카메라의 위치가 바뀔 때마다 호출되는 함수와 그 함수에서 사용하는 정보를 설정하는 함수이다.
   virtual void OnCameraUpdateCallback(float fTimeElapsed) { }
   void SetCameraUpdatedContext(LPVOID pContext) { m_pCameraUpdatedContext = pContext; }

   virtual void CreateShaderVariables(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList
*pd3dCommandList);
   virtual void ReleaseShaderVariables();
   virtual void UpdateShaderVariables(ID3D12GraphicsCommandList *pd3dCommandList);

//카메라를 변경하기 위하여 호출하는 함수이다.
   CCamera *OnChangeCamera(DWORD nNewCameraMode, DWORD nCurrentCameraMode);
   virtual CCamera *ChangeCamera(DWORD nNewCameraMode, float fTimeElapsed) {
return(NULL); }

//플레이어의 위치와 회전축으로부터 월드 변환 행렬을 생성하는 함수이다.
   virtual void OnPrepareRender();
//플레이어의 카메라가 3인칭 카메라일 때 플레이어(메쉬)를 렌더링한다.
   virtual void Render(ID3D12GraphicsCommandList *pd3dCommandList, CCamera *pCamera =
NULL);
};
```

❸ “CPlayer” 클래스에서 파생된 클래스 “CAirplanePlayer”를 다음과 같이 선언한다.

```
class CAirplanePlayer : public CPlayer
{
public:
   CAirplanePlayer(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList *pd3dCommandList,
ID3D12RootSignature *pd3dGraphicsRootSignature);
   virtual ~CAirplanePlayer();

   virtual CCamera *ChangeCamera(DWORD nNewCameraMode, float fTimeElapsed);
   virtual void OnPrepareRender();
};
```

⑨ ”Player.cpp” 파일 수정하기

❶ ”Player.cpp” 파일의 앞부분에 다음을 추가한다.

```
#include "Shader.h“
```

❷ “CPlayer” 클래스의 생성자와 소멸자를 다음과 같이 변경한다.

```
CPlayer::CPlayer()
{
   m_pCamera = NULL;

   m_xmf3Position = XMFLOAT3(0.0f, 0.0f, 0.0f);
   m_xmf3Right = XMFLOAT3(1.0f, 0.0f, 0.0f);
```

```
    m_xmf3Up = XMFLOAT3(0.0f, 1.0f, 0.0f);
    m_xmf3Look = XMFLOAT3(0.0f, 0.0f, 1.0f);

    m_xmf3Velocity = XMFLOAT3(0.0f, 0.0f, 0.0f);
    m_xmf3Gravity = XMFLOAT3(0.0f, 0.0f, 0.0f);
    m_fMaxVelocityXZ = 0.0f;
    m_fMaxVelocityY = 0.0f;
    m_fFriction = 0.0f;

    m_fPitch = 0.0f;
    m_fRoll = 0.0f;
    m_fYaw = 0.0f;

    m_pPlayerUpdatedContext = NULL;
    m_pCameraUpdatedContext = NULL;
}

CPlayer::~CPlayer()
{
    ReleaseShaderVariables();

    if (m_pCamera) delete m_pCamera;
}
```

❸ “CPlayer” 클래스의 CreateShaderVariables() 함수와 ReleaseShaderVariables() 함수 그리고 UpdateShaderVariables() 함수를 다음과 같이 정의한다.

```
void CPlayer::CreateShaderVariables(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList *pd3dCommandList)
{
    CGameObject::CreateShaderVariables(pd3dDevice, pd3dCommandList);

    if (m_pCamera) m_pCamera->CreateShaderVariables(pd3dDevice, pd3dCommandList);
}

void CPlayer::ReleaseShaderVariables()
{
    CGameObject::ReleaseShaderVariables();

    if (m_pCamera) m_pCamera->ReleaseShaderVariables();
}

void CPlayer::UpdateShaderVariables(ID3D12GraphicsCommandList *pd3dCommandList)
{
    CGameObject::UpdateShaderVariables(pd3dCommandList);
}
```

❹ “CPlayer” 클래스의 Move() 함수와 Rotate() 함수를 다음과 같이 정의한다.

```
/*플레이어의 위치를 변경하는 함수이다. 플레이어의 위치는 기본적으로 사용자가 플레이어를 이동하기 위한 키보드를 누를 때 변경된다. 플레이어의 이동 방향(dwDirection)에 따라 플레이어를 fDistance 만큼 이동한다.*/
void CPlayer::Move(DWORD dwDirection, float fDistance, bool bUpdateVelocity)
{
    if (dwDirection)
```

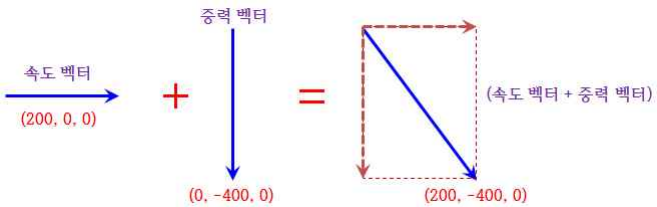
```
   {
      XMFLOAT3 xmf3Shift = XMFLOAT3(0, 0, 0);
//화살표 키 '↑'를 누르면 로컬 z-축 방향으로 이동(전진)한다. '↓'를 누르면 반대 방향으로 이동한다.
      if (dwDirection & DIR_FORWARD) xmf3Shift = Vector3::Add(xmf3Shift, m_xmf3Look,
fDistance);
      if (dwDirection & DIR_BACKWARD) xmf3Shift = Vector3::Add(xmf3Shift, m_xmf3Look,
-fDistance);
//화살표 키 '→'를 누르면 로컬 x-축 방향으로 이동한다. '←'를 누르면 반대 방향으로 이동한다.
      if (dwDirection & DIR_RIGHT) xmf3Shift = Vector3::Add(xmf3Shift, m_xmf3Right,
fDistance);
      if (dwDirection & DIR_LEFT) xmf3Shift = Vector3::Add(xmf3Shift, m_xmf3Right,
-fDistance);
//'Page Up'을 누르면 로컬 y-축 방향으로 이동한다. 'Page Down'을 누르면 반대 방향으로 이동한다.
      if (dwDirection & DIR_UP) xmf3Shift = Vector3::Add(xmf3Shift, m_xmf3Up, fDistance);
      if (dwDirection & DIR_DOWN) xmf3Shift = Vector3::Add(xmf3Shift, m_xmf3Up,
-fDistance);

//플레이어를 현재 위치 벡터에서 xmf3Shift 벡터만큼 이동한다.
      Move(xmf3Shift, bUpdateVelocity);
   }
}

void CPlayer::Move(const XMFLOAT3& xmf3Shift, bool bUpdateVelocity)
{
//bUpdateVelocity가 참이면 플레이어를 이동하지 않고 속도 벡터를 변경한다.
   if (bUpdateVelocity)
   {
//플레이어의 속도 벡터를 xmf3Shift 벡터만큼 변경한다.
      m_xmf3Velocity = Vector3::Add(m_xmf3Velocity, xmf3Shift);
   }
   else
   {
//플레이어를 현재 위치 벡터에서 xmf3Shift 벡터만큼 이동한다.
      m_xmf3Position = Vector3::Add(m_xmf3Position, xmf3Shift);
//플레이어의 위치가 변경되었으므로 카메라의 위치도 xmf3Shift 벡터만큼 이동한다.
      m_pCamera->Move(xmf3Shift);
   }
}

//플레이어를 로컬 x-축, y-축, z-축을 중심으로 회전한다.
void CPlayer::Rotate(float x, float y, float z)
{
   DWORD nCameraMode = m_pCamera->GetMode();
//1인칭 카메라 또는 3인칭 카메라의 경우 플레이어의 회전은 약간의 제약이 따른다.
   if ((nCameraMode == FIRST_PERSON_CAMERA) || (nCameraMode == THIRD_PERSON_CAMERA))
   {
/*로컬 x-축을 중심으로 회전하는 것은 고개를 앞뒤로 숙이는 동작에 해당한다. 그러므로 x-축을 중심으로 회전하는
각도는 -89.0~+89.0도 사이로 제한한다. x는 현재의 m_fPitch에서 실제 회전하는 각도이므로 x만큼 회전한 다음
Pitch가 +89도 보다 크거나 -89도 보다 작으면 m_fPitch가 +89도 또는 -89도가 되도록 회전각도(x)를 수정한다.*/
      if (x != 0.0f)
      {
         m_fPitch += x;
         if (m_fPitch > +89.0f) { x -= (m_fPitch - 89.0f); m_fPitch = +89.0f; }
```

```
            if (m_fPitch < -89.0f) { x -= (m_fPitch + 89.0f); m_fPitch = -89.0f; }
        }
        if (y != 0.0f)
        {
//로컬 y-축을 중심으로 회전하는 것은 몸통을 돌리는 것이므로 회전 각도의 제한이 없다.
            m_fYaw += y;
            if (m_fYaw > 360.0f) m_fYaw -= 360.0f;
            if (m_fYaw < 0.0f) m_fYaw += 360.0f;
        }
        if (z != 0.0f)
        {
/*로컬 z-축을 중심으로 회전하는 것은 몸통을 좌우로 기울이는 것이므로 회전 각도는 -20.0~+20.0도 사이로 제한된다. z는 현재의 m_fRoll에서 실제 회전하는 각도이므로 z만큼 회전한 다음 m_fRoll이 +20도 보다 크거나 -20도보다 작으면 m_fRoll이 +20도 또는 -20도가 되도록 회전각도(z)를 수정한다.*/
            m_fRoll += z;
            if (m_fRoll > +20.0f) { z -= (m_fRoll - 20.0f); m_fRoll = +20.0f; }
            if (m_fRoll < -20.0f) { z -= (m_fRoll + 20.0f); m_fRoll = -20.0f; }
        }
//카메라를 x, y, z 만큼 회전한다. 플레이어를 회전하면 카메라가 회전하게 된다.
        m_pCamera->Rotate(x, y, z);

/*플레이어를 회전한다. 1인칭 카메라 또는 3인칭 카메라에서 플레이어의 회전은 로컬 y-축에서만 일어난다. 플레이어의 로컬 y-축(Up 벡터)을 기준으로 로컬 z-축(Look 벡터)와 로컬 x-축(Right 벡터)을 회전시킨다. 기본적으로 Up 벡터를 기준으로 회전하는 것은 플레이어가 똑바로 서있는 것을 가정한다는 의미이다.*/
        if (y != 0.0f)
        {
            XMMATRIX xmmtxRotate = XMMatrixRotationAxis(XMLoadFloat3(&m_xmf3Up),
XMConvertToRadians(y));
            m_xmf3Look = Vector3::TransformNormal(m_xmf3Look, xmmtxRotate);
            m_xmf3Right = Vector3::TransformNormal(m_xmf3Right, xmmtxRotate);
        }
    }
    else if (nCameraMode == SPACESHIP_CAMERA)
    {
/*스페이스-쉽 카메라에서 플레이어의 회전은 회전 각도의 제한이 없다. 그리고 모든 축을 중심으로 회전을 할 수 있다.*/
        m_pCamera->Rotate(x, y, z);
        if (x != 0.0f)
        {
            XMMATRIX xmmtxRotate = XMMatrixRotationAxis(XMLoadFloat3(&m_xmf3Right),
XMConvertToRadians(x));
            m_xmf3Look = Vector3::TransformNormal(m_xmf3Look, xmmtxRotate);
            m_xmf3Up = Vector3::TransformNormal(m_xmf3Up, xmmtxRotate);
        }
        if (y != 0.0f)
        {
            XMMATRIX xmmtxRotate = XMMatrixRotationAxis(XMLoadFloat3(&m_xmf3Up),
XMConvertToRadians(y));
            m_xmf3Look = Vector3::TransformNormal(m_xmf3Look, xmmtxRotate);
            m_xmf3Right = Vector3::TransformNormal(m_xmf3Right, xmmtxRotate);
        }
        if (z != 0.0f)
        {
            XMMATRIX xmmtxRotate = XMMatrixRotationAxis(XMLoadFloat3(&m_xmf3Look),
XMConvertToRadians(z));
```

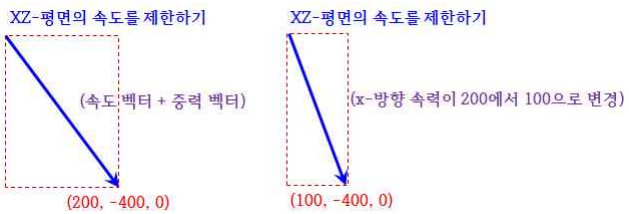
```
            m_xmf3Up = Vector3::TransformNormal(m_xmf3Up, xmmtxRotate);
            m_xmf3Right = Vector3::TransformNormal(m_xmf3Right, xmmtxRotate);
        }
    }

/*회전으로 인해 플레이어의 로컬 x-축, y-축, z-축이 서로 직교하지 않을 수 있으므로 z-축(LookAt 벡터)을 기준으로 하여 서로 직교하고 단위벡터가 되도록 한다.*/
    m_xmf3Look = Vector3::Normalize(m_xmf3Look);
    m_xmf3Right = Vector3::CrossProduct(m_xmf3Up, m_xmf3Look, true);
    m_xmf3Up = Vector3::CrossProduct(m_xmf3Look, m_xmf3Right, true);
}
```

❺ "CPlayer" 클래스의 Update() 함수를 다음과 같이 변경한다.

```
//이 함수는 매 프레임마다 호출된다. 플레이어의 속도 벡터에 중력과 마찰력 등을 적용한다.
void CPlayer::Update(float fTimeElapsed)
{
/*플레이어의 속도 벡터를 중력 벡터와 더한다. 중력 벡터에 fTimeElapsed를 곱하는 것은 중력을 시간에 비례하도록 적용한다는 의미이다.*/
    m_xmf3Velocity = Vector3::Add(m_xmf3Velocity, Vector3::ScalarProduct(m_xmf3Gravity, fTimeElapsed, false));

/*플레이어의 속도 벡터의 XZ-성분의 크기를 구한다. 이것이 XZ-평면의 최대 속력보다 크면 속도 벡터의 x와 z-방향 성분을 조정한다.*/
```



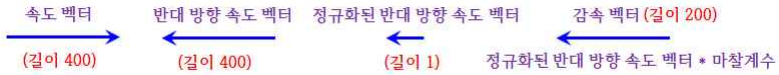
플레이어에 중력 적용하기



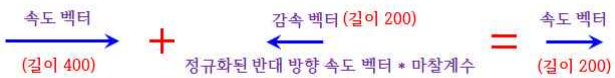
XZ-평면의 속도 제한하기

```
    float fLength = sqrtf(m_xmf3Velocity.x * m_xmf3Velocity.x + m_xmf3Velocity.z * m_xmf3Velocity.z);
    float fMaxVelocityXZ = m_fMaxVelocityXZ * fTimeElapsed;
    if (fLength > m_fMaxVelocityXZ)
    {
        m_xmf3Velocity.x *= (fMaxVelocityXZ / fLength);
        m_xmf3Velocity.z *= (fMaxVelocityXZ / fLength);
    }
/*플레이어의 속도 벡터의 y-성분의 크기를 구한다. 이것이 y-축 방향의 최대 속력보다 크면 속도 벡터의 y-방향 성
```

```
분을 조정한다.*/
   float fMaxVelocityY = m_fMaxVelocityY * fTimeElapsed;
   fLength = sqrtf(m_xmf3Velocity.y * m_xmf3Velocity.y);
   if (fLength > m_fMaxVelocityY) m_xmf3Velocity.y *= (fMaxVelocityY / fLength);

//플레이어를 속도 벡터 만큼 실제로 이동한다(카메라도 이동될 것이다).
   Move(m_xmf3Velocity, false);

/*플레이어의 위치가 변경될 때 추가로 수행할 작업을 수행한다. 플레이어의 새로운 위치가 유효한 위치가 아닐 수도 있고 또는 플레이어의 충돌 검사 등을 수행할 필요가 있다. 이러한 상황에서 플레이어의 위치를 유효한 위치로 다시 변경할 수 있다.*/
   if (m_pPlayerUpdatedContext) OnPlayerUpdateCallback(fTimeElapsed);

   DWORD nCameraMode = m_pCamera->GetMode();
//플레이어의 위치가 변경되었으므로 3인칭 카메라를 갱신한다.
   if (nCameraMode == THIRD_PERSON_CAMERA) m_pCamera->Update(m_xmf3Position,
fTimeElapsed);
//카메라의 위치가 변경될 때 추가로 수행할 작업을 수행한다.
   if (m_pCameraUpdatedContext) OnCameraUpdateCallback(fTimeElapsed);
//카메라가 3인칭 카메라이면 카메라가 변경된 플레이어 위치를 바라보도록 한다.
   if (nCameraMode == THIRD_PERSON_CAMERA) m_pCamera->SetLookAt(m_xmf3Position);
//카메라의 카메라 변환 행렬을 다시 생성한다.
   m_pCamera->RegenerateViewMatrix();

/*플레이어의 속도 벡터가 마찰력 때문에 감속이 되어야 한다면 감속 벡터를 생성한다. 속도 벡터의 반대 방향 벡터를 구하고 단위 벡터로 만든다. 마찰 계수를 시간에 비례하도록 하여 마찰력을 구한다. 단위 벡터에 마찰력을 곱하여 감속 벡터를 구한다. 속도 벡터에 감속 벡터를 더하여 속도 벡터를 줄인다. 마찰력이 속력보다 크면 속력은 0이 될 것이다.*/
```



감속 벡터 생성



속도 벡터에 감속 벡터 적용

```
   fLength = Vector3::Length(m_xmf3Velocity);
   float fDeceleration = (m_fFriction * fTimeElapsed);
   if (fDeceleration > fLength) fDeceleration = fLength;
   m_xmf3Velocity = Vector3::Add(m_xmf3Velocity, Vector3::ScalarProduct(m_xmf3Velocity,
-fDeceleration, true));
}
```

❻ “CPlayer” 클래스의 OnChangeCamera() 함수를 다음과 같이 정의한다.

```
/*카메라를 변경할 때 ChangeCamera() 함수에서 호출되는 함수이다. nCurrentCameraMode는 현재 카메라의 모드이고 nNewCameraMode는 새로 설정할 카메라 모드이다.*/
CCamera *CPlayer::OnChangeCamera(DWORD nNewCameraMode, DWORD nCurrentCameraMode)
{
```

```
//새로운 카메라의 모드에 따라 카메라를 새로 생성한다.
   CCamera *pNewCamera = NULL;
   switch (nNewCameraMode)
   {
      case FIRST_PERSON_CAMERA:
         pNewCamera = new CFirstPersonCamera(m_pCamera);
         break;
      case THIRD_PERSON_CAMERA:
         pNewCamera = new CThirdPersonCamera(m_pCamera);
         break;
      case SPACESHIP_CAMERA:
         pNewCamera = new CSpaceShipCamera(m_pCamera);
         break;
   }
/*현재 카메라의 모드가 스페이스-쉽 모드의 카메라이고 새로운 카메라가 1인칭 또는 3인칭 카메라이면 플레이어의 Up 벡터를 월드좌표계의 y-축 방향 벡터(0, 1, 0)이 되도록 한다. 즉, 똑바로 서도록 한다. 그리고 스페이스-쉽 카메라의 경우 플레이어의 이동에는 제약이 없다. 특히, y-축 방향의 움직임이 자유롭다. 그러므로 플레이어의 위치는 공중(위치 벡터의 y-좌표가 0보다 크다)이 될 수 있다. 이때 새로운 카메라가 1인칭 또는 3인칭 카메라이면 플레이어의 위치는 지면이 되어야 한다. 그러므로 플레이어의 Right 벡터와 Look 벡터의 y 값을 0으로 만든다. 이제 플레이어의 Right 벡터와 Look 벡터는 단위벡터가 아니므로 정규화한다.*/
   if (nCurrentCameraMode == SPACESHIP_CAMERA)
   {
      m_xmf3Right = Vector3::Normalize(XMFLOAT3(m_xmf3Right.x, 0.0f, m_xmf3Right.z));
      m_xmf3Up = Vector3::Normalize(XMFLOAT3(0.0f, 1.0f, 0.0f));
      m_xmf3Look = Vector3::Normalize(XMFLOAT3(m_xmf3Look.x, 0.0f, m_xmf3Look.z));

      m_fPitch = 0.0f;
      m_fRoll = 0.0f;
/*Look 벡터와 월드좌표계의 z-축(0, 0, 1)이 이루는 각도(내적=cos)를 계산하여 플레이어의 y-축의 회전 각도 m_fYaw로 설정한다.*/
      m_fYaw = Vector3::Angle(XMFLOAT3(0.0f, 0.0f, 1.0f), m_xmf3Look);
      if (m_xmf3Look.x < 0.0f) m_fYaw = -m_fYaw;
   }
   else if ((nNewCameraMode == SPACESHIP_CAMERA) && m_pCamera)
   {
/*새로운 카메라의 모드가 스페이스-쉽 모드의 카메라이고 현재 카메라 모드가 1인칭 또는 3인칭 카메라이면 플레이어의 로컬 축을 현재 카메라의 로컬 축과 같게 만든다.*/
      m_xmf3Right = m_pCamera->GetRightVector();
      m_xmf3Up = m_pCamera->GetUpVector();
      m_xmf3Look = m_pCamera->GetLookVector();
   }

   if (pNewCamera)
   {
      pNewCamera->SetMode(nNewCameraMode);
//현재 카메라를 사용하는 플레이어 객체를 설정한다.
      pNewCamera->SetPlayer(this);
   }

   if (m_pCamera) delete m_pCamera;

   return(pNewCamera);
}
```

❼ “CPlayer” 클래스의 OnPrepareRender() 함수와 Render() 함수를 다음과 같이 정의한다.

```
/*플레이어의 위치와 회전축으로부터 월드 변환 행렬을 생성하는 함수이다. 플레이어의 Right 벡터가 월드 변환 행렬의 첫 번째 행 벡터, Up 벡터가 두 번째 행 벡터, Look 벡터가 세 번째 행 벡터, 플레이어의 위치 벡터가 네 번째 행 벡터가 된다.*/
void CPlayer::OnPrepareRender()
{
   m_xmf4x4World._11 = m_xmf3Right.x;
   m_xmf4x4World._12 = m_xmf3Right.y;
   m_xmf4x4World._13 = m_xmf3Right.z;
   m_xmf4x4World._21 = m_xmf3Up.x;
   m_xmf4x4World._22 = m_xmf3Up.y;
   m_xmf4x4World._23 = m_xmf3Up.z;
   m_xmf4x4World._31 = m_xmf3Look.x;
   m_xmf4x4World._32 = m_xmf3Look.y;
   m_xmf4x4World._33 = m_xmf3Look.z;
   m_xmf4x4World._41 = m_xmf3Position.x;
   m_xmf4x4World._42 = m_xmf3Position.y;
   m_xmf4x4World._43 = m_xmf3Position.z;
}

void CPlayer::Render(ID3D12GraphicsCommandList *pd3dCommandList, CCamera *pCamera)
{
   DWORD nCameraMode = (pCamera) ? pCamera->GetMode() : 0x00;
//카메라 모드가 3인칭이면 플레이어 객체를 렌더링한다.
   if (nCameraMode == THIRD_PERSON_CAMERA) CGameObject::Render(pd3dCommandList, pCamera);
}
```

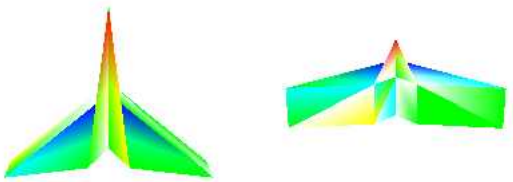
❽ “CAirplanePlayer” 클래스의 생성자와 소멸자를 다음과 같이 정의한다.

```
CAirplanePlayer::CAirplanePlayer(ID3D12Device *pd3dDevice, ID3D12GraphicsCommandList *pd3dCommandList, ID3D12RootSignature *pd3dGraphicsRootSignature)
{
//비행기 메쉬를 생성한다.
   CMesh *pAirplaneMesh = new CAirplaneMeshDiffused(pd3dDevice, pd3dCommandList, 20.0f, 20.0f, 4.0f, XMFLOAT4(0.0f, 0.5f, 0.0f, 0.0f));

   SetMesh(pAirplaneMesh);
//플레이어의 카메라를 스페이스-쉽 카메라로 변경(생성)한다.
   m_pCamera = ChangeCamera(SPACESHIP_CAMERA, 0.0f);

//플레이어를 위한 셰이더 변수를 생성한다.
   CreateShaderVariables(pd3dDevice, pd3dCommandList);

//플레이어의 위치를 설정한다.
   SetPosition(XMFLOAT3(0.0f, 0.0f, -50.0f));

//플레이어(비행기) 메쉬를 렌더링할 때 사용할 셰이더를 생성한다.
   CPlayerShader *pShader = new CPlayerShader();
   pShader->CreateShader(pd3dDevice, pd3dGraphicsRootSignature);
   SetShader(pShader);
}
```

```
CAirplanePlayer::~CAirplanePlayer()
{
}
```

❾ “CPlayer” 클래스의 OnPrepareRender() 함수를 다음과 같이 정의한다.

```
void CAirplanePlayer::OnPrepareRender()
{
   CPlayer::OnPrepareRender();

//비행기 모델을 그리기 전에 x-축으로 90도 회전한다.
   XMMATRIX mtxRotate = XMMatrixRotationRollPitchYaw(XMConvertToRadians(90.0f), 0.0f,
0.0f);
   m_xmf4x4World = Matrix4x4::Multiply(mtxRotate, m_xmf4x4World);
}
/*3인칭 카메라일 때 플레이어 메쉬를 로컬 x-축을 중심으로 +90도 회전하고 렌더링한다. 왜냐하면 비행기 모델 메쉬
는 다음 그림과 같이 y-축 방향이 비행기의 앞쪽이 되도록 모델링이 되었기 때문이다. 그리고 이 메쉬를 카메라의 z-
축 방향으로 향하도록 그릴 것이기 때문이다.*/
```



비행기 메쉬의 x-축 90도 회전

❿ “CPlayer” 클래스의 ChangeCamera() 함수를 다음과 같이 정의한다.

```
//카메라를 변경할 때 호출되는 함수이다. nNewCameraMode는 새로 설정할 카메라 모드이다.
CCamera *CAirplanePlayer::ChangeCamera(DWORD nNewCameraMode, float fTimeElapsed)
{
   DWORD nCurrentCameraMode = (m_pCamera) ? m_pCamera->GetMode() : 0x00;
   if (nCurrentCameraMode == nNewCameraMode) return(m_pCamera);
   switch (nNewCameraMode)
   {
      case FIRST_PERSON_CAMERA:
//플레이어의 특성을 1인칭 카메라 모드에 맞게 변경한다. 중력은 적용하지 않는다.
         SetFriction(200.0f);
         SetGravity(XMFLOAT3(0.0f, 0.0f, 0.0f));
         SetMaxVelocityXZ(125.0f);
         SetMaxVelocityY(400.0f);
         m_pCamera = OnChangeCamera(FIRST_PERSON_CAMERA, nCurrentCameraMode);
         m_pCamera->SetTimeLag(0.0f);
         m_pCamera->SetOffset(XMFLOAT3(0.0f, 20.0f, 0.0f));
         m_pCamera->GenerateProjectionMatrix(1.01f, 5000.0f, ASPECT_RATIO, 60.0f);
         m_pCamera->SetViewport(0, 0, FRAME_BUFFER_WIDTH, FRAME_BUFFER_HEIGHT, 0.0f,
1.0f);
         m_pCamera->SetScissorRect(0, 0, FRAME_BUFFER_WIDTH, FRAME_BUFFER_HEIGHT);
         break;
      case SPACESHIP_CAMERA:
```

```
//플레이어의 특성을 스페이스-쉽 카메라 모드에 맞게 변경한다. 중력은 적용하지 않는다.
         SetFriction(125.0f);
         SetGravity(XMFLOAT3(0.0f, 0.0f, 0.0f));
         SetMaxVelocityXZ(400.0f);
         SetMaxVelocityY(400.0f);
         m_pCamera = OnChangeCamera(SPACESHIP_CAMERA, nCurrentCameraMode);
         m_pCamera->SetTimeLag(0.0f);
         m_pCamera->SetOffset(XMFLOAT3(0.0f, 0.0f, 0.0f));
         m_pCamera->GenerateProjectionMatrix(1.01f, 5000.0f, ASPECT_RATIO, 60.0f);
         m_pCamera->SetViewport(0, 0, FRAME_BUFFER_WIDTH, FRAME_BUFFER_HEIGHT, 0.0f,
1.0f);
         m_pCamera->SetScissorRect(0, 0, FRAME_BUFFER_WIDTH, FRAME_BUFFER_HEIGHT);
         break;
      case THIRD_PERSON_CAMERA:
//플레이어의 특성을 3인칭 카메라 모드에 맞게 변경한다. 지연 효과와 카메라 오프셋을 설정한다.
         SetFriction(250.0f);
         SetGravity(XMFLOAT3(0.0f, 0.0f, 0.0f));
         SetMaxVelocityXZ(125.0f);
         SetMaxVelocityY(400.0f);
         m_pCamera = OnChangeCamera(THIRD_PERSON_CAMERA, nCurrentCameraMode);
//3인칭 카메라의 지연 효과를 설정한다. 값을 0.25f 대신에 0.0f와 1.0f로 설정한 결과를 비교하기 바란다.
         m_pCamera->SetTimeLag(0.25f);
         m_pCamera->SetOffset(XMFLOAT3(0.0f, 20.0f, -50.0f));
         m_pCamera->GenerateProjectionMatrix(1.01f, 5000.0f, ASPECT_RATIO, 60.0f);
         m_pCamera->SetViewport(0, 0, FRAME_BUFFER_WIDTH, FRAME_BUFFER_HEIGHT, 0.0f,
1.0f);
         m_pCamera->SetScissorRect(0, 0, FRAME_BUFFER_WIDTH, FRAME_BUFFER_HEIGHT);
         break;
      default:
         break;
   }
//플레이어를 시간의 경과에 따라 갱신(위치와 방향을 변경: 속도, 마찰력, 중력 등을 처리)한다.
   Update(fTimeElapsed);

   return(m_pCamera);
}
```

⑩ “GameFramework.h” 파일 변경하기

❶ “CGameFramework” 클래스에 다음을 추가한다.

```
#include "Player.h“
```

❷ “CGameFramework” 클래스에 다음 멤버 변수를 추가한다.

```
public:
//플레이어 객체에 대한 포인터이다.
   CPlayer *m_pPlayer = NULL;

//마지막으로 마우스 버튼을 클릭할 때의 마우스 커서의 위치이다.
   POINT m_ptOldCursorPos;
```

⑪ “GameFramework.cpp” 파일 변경하기

❶ “CGameFramework” 클래스의 OnProcessingMouseMessage() 멤버 함수를 다음과 같이 변경한다.

```
switch (nMessageID)
{
   case WM_LBUTTONDOWN:
   case WM_RBUTTONDOWN:
//마우스 캡쳐를 하고 현재 마우스 위치를 가져온다.
      ::SetCapture(hWnd);
      ::GetCursorPos(&m_ptOldCursorPos);
      break;
   case WM_LBUTTONUP:
   case WM_RBUTTONUP:
//마우스 캡쳐를 해제한다.
      ::ReleaseCapture();
      break;
```

❷ “CGameFramework” 클래스의 OnProcessingKeyboardMessage() 멤버 함수에서 WM_KEYUP 메시지 처리 부분에 다음을 추가한다.

```
    case WM_KEYUP:
      switch (wParam)
      {
/*'F1' 키를 누르면 1인칭 카메라, 'F2' 키를 누르면 스페이스-쉽 카메라로 변경한다, 'F3' 키를 누르면 3인칭 카메라로 변경한다.*/
         case VK_F1:
         case VK_F2:
         case VK_F3:
            if (m_pPlayer) m_pPlayer->ChangeCamera(m_pd3dDevice, (wParam-VK_F1+1), m_GameTimer.GetTimeElapsed());
            break;
```

❸ “CGameFramework” 클래스의 BuildObjects() 멤버 함수를 다음과 같이 변경한다.

```
void CGameFramework::BuildObjects()
{
   m_pd3dCommandList->Reset(m_pd3dCommandAllocator, NULL);

   m_pScene = new CScene();
   if (m_pScene) m_pScene->BuildObjects(m_pd3dDevice, m_pd3dCommandList);

   CAirplanePlayer *pAirplanePlayer = new CAirplanePlayer(m_pd3dDevice, m_pd3dCommandList, m_pScene->GetGraphicsRootSignature());
   m_pPlayer = pAirplanePlayer;
   m_pCamera = m_pPlayer->GetCamera();

   m_pd3dCommandList->Close();
   ID3D12CommandList *ppd3dCommandLists[] = { m_pd3dCommandList };
   m_pd3dCommandQueue->ExecuteCommandLists(1, ppd3dCommandLists);
```

```
   WaitForGpuComplete();
   if (m_pScene) m_pScene->ReleaseUploadBuffers();

   m_GameTimer.Reset();
}
```

❹ “CGameFramework” 클래스의 ProcessInput() 멤버 함수를 다음과 같이 변경한다.

```
void CGameFramework::ProcessInput()
{
   static UCHAR pKeyBuffer[256];
   DWORD dwDirection = 0;
/*키보드의 상태 정보를 반환한다. 화살표 키(‘→’, ‘←’, ‘↑’, ‘↓’)를 누르면 플레이어를 오른쪽/왼쪽(로컬 x-축), 앞/뒤(로컬 z-축)로 이동한다. ‘Page Up’과 ‘Page Down’ 키를 누르면 플레이어를 위/아래(로컬 y-축)로 이동한다.*/
   if (::GetKeyboardState(pKeyBuffer))
   {
      if (pKeyBuffer[VK_UP] & 0xF0) dwDirection |= DIR_FORWARD;
      if (pKeyBuffer[VK_DOWN] & 0xF0) dwDirection |= DIR_BACKWARD;
      if (pKeyBuffer[VK_LEFT] & 0xF0) dwDirection |= DIR_LEFT;
      if (pKeyBuffer[VK_RIGHT] & 0xF0) dwDirection |= DIR_RIGHT;
      if (pKeyBuffer[VK_PRIOR] & 0xF0) dwDirection |= DIR_UP;
      if (pKeyBuffer[VK_NEXT] & 0xF0) dwDirection |= DIR_DOWN;
   }
   float cxDelta = 0.0f, cyDelta = 0.0f;
   POINT ptCursorPos;
/*마우스를 캡쳐했으면 마우스가 얼마만큼 이동하였는 가를 계산한다. 마우스 왼쪽 또는 오른쪽 버튼이 눌러질 때의 메시지(WM_LBUTTONDOWN, WM_RBUTTONDOWN)를 처리할 때 마우스를 캡쳐하였다. 그러므로 마우스가 캡쳐된 것은 마우스 버튼이 눌려진 상태를 의미한다. 마우스 버튼이 눌려진 상태에서 마우스를 좌우 또는 상하로 움직이면 플레이어를 x-축 또는 y-축으로 회전한다.*/
   if (::GetCapture() == m_hWnd)
   {
//마우스 커서를 화면에서 없앤다(보이지 않게 한다).
      ::SetCursor(NULL);
//현재 마우스 커서의 위치를 가져온다.
      ::GetCursorPos(&ptCursorPos);
//마우스 버튼이 눌린 상태에서 마우스가 움직인 양을 구한다.
      cxDelta = (float)(ptCursorPos.x - m_ptOldCursorPos.x) / 3.0f;
      cyDelta = (float)(ptCursorPos.y - m_ptOldCursorPos.y) / 3.0f;
//마우스 커서의 위치를 마우스가 눌려졌던 위치로 설정한다.
      ::SetCursorPos(m_ptOldCursorPos.x, m_ptOldCursorPos.y);
   }
//마우스 또는 키 입력이 있으면 플레이어를 이동하거나(dwDirection) 회전한다(cxDelta 또는 cyDelta).
   if ((dwDirection != 0) || (cxDelta != 0.0f) || (cyDelta != 0.0f))
   {
      if (cxDelta || cyDelta)
      {
/*cxDelta는 y-축의 회전을 나타내고 cyDelta는 x-축의 회전을 나타낸다. 오른쪽 마우스 버튼이 눌려진 경우 cxDelta는 z-축의 회전을 나타낸다.*/
         if (pKeyBuffer[VK_RBUTTON] & 0xF0)
            m_pPlayer->Rotate(cyDelta, 0.0f, -cxDelta);
         else
            m_pPlayer->Rotate(cyDelta, cxDelta, 0.0f);
```

```
        }
/*플레이어를 dwDirection 방향으로 이동한다(실제로는 속도 벡터를 변경한다). 이동 거리는 시간에 비례하도록 한다. 플레이어의 이동 속력은 (50/초)로 가정한다.*/
        if (dwDirection) m_pPlayer->Move(dwDirection, 50.0f * m_GameTimer.GetTimeElapsed(), true);
    }

//플레이어를 실제로 이동하고 카메라를 갱신한다. 중력과 마찰력의 영향을 속도 벡터에 적용한다.
    m_pPlayer->Update(m_GameTimer.GetTimeElapsed());
}
```

❺ "CGameFramework" 클래스의 FrameAdvance() 멤버 함수를 다음과 같이 변경한다.

```
//#define _WITH_PLAYER_TOP

void CGameFramework::FrameAdvance()
{
    m_GameTimer.Tick(0.0f);

    ProcessInput();

    AnimateObjects();

    HRESULT hResult = m_pd3dCommandAllocator->Reset();
    hResult = m_pd3dCommandList->Reset(m_pd3dCommandAllocator, NULL);

    D3D12_RESOURCE_BARRIER d3dResourceBarrier;
    ::ZeroMemory(&d3dResourceBarrier, sizeof(D3D12_RESOURCE_BARRIER));
    d3dResourceBarrier.Type = D3D12_RESOURCE_BARRIER_TYPE_TRANSITION;
    d3dResourceBarrier.Flags = D3D12_RESOURCE_BARRIER_FLAG_NONE;
    d3dResourceBarrier.Transition.pResource = m_ppd3dRenderTargetBuffers[m_nSwapChainBufferIndex];
    d3dResourceBarrier.Transition.StateBefore = D3D12_RESOURCE_STATE_PRESENT;
    d3dResourceBarrier.Transition.StateAfter = D3D12_RESOURCE_STATE_RENDER_TARGET;
    d3dResourceBarrier.Transition.Subresource = D3D12_RESOURCE_BARRIER_ALL_SUBRESOURCES;
    m_pd3dCommandList->ResourceBarrier(1, &d3dResourceBarrier);

    D3D12_CPU_DESCRIPTOR_HANDLE d3dRtvCPUDescriptorHandle = m_pd3dRtvDescriptorHeap->GetCPUDescriptorHandleForHeapStart();
    d3dRtvCPUDescriptorHandle.ptr += (m_nSwapChainBufferIndex * m_nRtvDescriptorIncrementSize);

    float pfClearColor[4] = { 0.0f, 0.125f, 0.3f, 1.0f };
    m_pd3dCommandList->ClearRenderTargetView(d3dRtvCPUDescriptorHandle, pfClearColor/*Colors::Azure*/, 0, NULL);

    D3D12_CPU_DESCRIPTOR_HANDLE d3dDsvCPUDescriptorHandle = m_pd3dDsvDescriptorHeap->GetCPUDescriptorHandleForHeapStart();
    m_pd3dCommandList->ClearDepthStencilView(d3dDsvCPUDescriptorHandle, D3D12_CLEAR_FLAG_DEPTH | D3D12_CLEAR_FLAG_STENCIL, 1.0f, 0, 0, NULL);

    m_pd3dCommandList->OMSetRenderTargets(1, &d3dRtvCPUDescriptorHandle, TRUE, &d3dDsvCPUDescriptorHandle);
```

```
    if (m_pScene) m_pScene->Render(m_pd3dCommandList, m_pCamera);

//3인칭 카메라일 때 플레이어가 항상 보이도록 렌더링한다.
#ifdef _WITH_PLAYER_TOP
//렌더 타겟은 그대로 두고 깊이 버퍼를 1.0으로 지우고 플레이어를 렌더링하면 플레이어는 무조건 그려질 것이다.
    m_pd3dCommandList->ClearDepthStencilView(d3dDsvCPUDescriptorHandle,
D3D12_CLEAR_FLAG_DEPTH | D3D12_CLEAR_FLAG_STENCIL, 1.0f, 0, 0, NULL);
#endif
//3인칭 카메라일 때 플레이어를 렌더링한다.
    if (m_pPlayer) m_pPlayer->Render(m_pd3dCommandList, m_pCamera);

    d3dResourceBarrier.Transition.StateBefore = D3D12_RESOURCE_STATE_RENDER_TARGET;
    d3dResourceBarrier.Transition.StateAfter = D3D12_RESOURCE_STATE_PRESENT;
    d3dResourceBarrier.Transition.Subresource = D3D12_RESOURCE_BARRIER_ALL_SUBRESOURCES;
    m_pd3dCommandList->ResourceBarrier(1, &d3dResourceBarrier);

    hResult = m_pd3dCommandList->Close();

    ID3D12CommandList *ppd3dCommandLists[] = { m_pd3dCommandList };
    m_pd3dCommandQueue->ExecuteCommandLists(1, ppd3dCommandLists);

    WaitForGpuComplete();

    m_pdxgiSwapChain->Present(0, 0);

    MoveToNextFrame();

    m_GameTimer.GetFrameRate(m_pszFrameRate + 12, 37);
    ::SetWindowText(m_hWnd, m_pszFrameRate);
}
```

⑫ 프로젝트 빌드하여 실행하기

프로젝트를 실행하면 1인칭 카메라 모드로 실행될 것이다. ‘F2’ 키를 누르면 스페이스-쉽 카메라 모드로 변경되고 ‘F3’ 키를 누르면 3인칭 카메라 모드로 변경되는 것을 확인할 수 있다. 3인칭 카메라 모드에서는 플레이어가 비행기 메쉬로 그려진다. ‘F1’ 키를 누르면 다시 1인칭 카메라로 변경된다. 화살표 키를 누르면 그 방향으로 플레이어(카메라)가 이동하고 마우스 버튼을 클릭한 상태로 마우스를 움직이면 플레이어(카메라)가 회전한다.

“Camera.cpp" 파일의 CAirplanePlayer::ChangeCamera() 함수에서 3인칭 카메라의 카메라 지연 효과의 값을 설정하는 부분을 다음과 같이 변경하여 빌드하고 실행한다.

```
    m_pCamera->SetTimeLag(0.0f);
```

3인칭 카메라로 전환할 때와 3인칭 카메라를 회전할 때 카메라의 동작이 지연 효과의 값이 0.25f일 때와 차이가 있는 것을 발견할 수 있을 것이다.

3인칭 카메라 모드에서 플레이어 객체를 움직일 때 플레이어 객체와 카메라 사이에 직육면체가 있으면 다음 그림과 같이 플레이어 객체를 직육면체가 가리게 될 수 있다. 만약 3인칭 카메라 모드일 때 플레이어가 언제나 항상 보이도록 그려지기를 원한다면 플레이어 객체를 마지막으로 그리도록 하고 그리기 전에

깊이-스텐실 버퍼를 깊이값 1.0으로 지우면 된다.