1. 3D 그래픽 기초(3D Graphics Fundamentals)

(3) 소프트웨어 렌더러(Software Renderer)의 구현

앞에서 구현한 소프트웨어 렌더러를 DirectXMath 라이브러리를 사용하여 벡터와 행렬을 사용하여 구현하자.

소프트웨어 렌더러 응용프로그램은 CVertex, CPolygon, CMesh, CCubeMesh, CAirplaneMesh, CGameObject, CScene, CPlayer, CAirplanePlayer, CViewport, CCamera, CGameFramework, CGraphicsPipeline 클래스를 통하여 구현될 것이다.

① 새로운 프로젝트 추가하기

앞에서 작성한 "LabProjects" 솔루션에 새로운 프로젝트 "LabProject03"를 추가(응용 프로그램 마법사를 실행)하자. "따라하기(02)"에서와 같이 프로젝트 "LabProject03"를 수정한다.

② "stdafx.h" 파일 수정하기

헤더 파일 포함 부분에 다음을 추가한다.

```
#include <Mmsystem.h>
#pragma comment(lib, "winmm.lib")

#include <DirectXMath.h>
#include <DirectXPackedVector.h>
#include <DirectXColors.h>
#include <DirectXCollision.h>

using namespace DirectX;
using namespace DirectX::PackedVector;

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

namespace Matrix4x4
{
   inline XMFLOAT4X4 Identity()
   {
      XMFLOAT4X4 xmmtx4x4Result;
      XMStoreFloat4x4(&xmmtx4x4Result, XMMatrixIdentity());
      return(xmmtx4x4Result);
   }
```

```
}
```

③ “Mesh.h” 파일을 다음과 같이 수정한다.

❶ “Mesh.h” 파일의 CVertex 클래스를 다음과 같이 수정한다.

```
class CVertex
{
public:
   CVertex() { }
   CVertex(float x, float y, float z) { m_xmf3Position = XMFLOAT3(x, y,
z); }
   virtual ~CVertex() { }

   XMFLOAT3    m_xmf3Position;
};
```

❷ “CAirplaneMesh” 클래스를 다음과 같이 선언한다.

```
class CAirplaneMesh : public CMesh
{
public:
   CAirplaneMesh(float fWidth, float fHeight, float fDepth);
   virtual ~CAirplaneMesh() { }
};
```

④ “Mesh.cpp” 파일을 다음과 같이 수정한다.

❶ “Mesh.cpp” 파일에 다음 헤더 파일 포함을 추가한다.

```
#include "stdafx.h"
#include "Mesh.h"
#include "GraphicsPipeline.h"
```

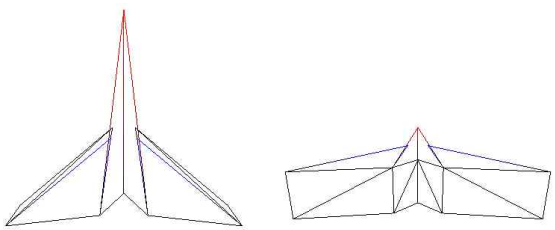
❷ Draw2DLine() 전역 함수를 다음과 같이 수정한다.

```
void Draw2DLine(HDC hDCFrameBuffer, XMFLOAT3& f3PreviousProject,
XMFLOAT3& f3CurrentProject)
{
   XMFLOAT3 f3Previous =
CGraphicsPipeline::ScreenTransform(f3PreviousProject);
   XMFLOAT3 f3Current =
CGraphicsPipeline::ScreenTransform(f3CurrentProject);
   ::MoveToEx(hDCFrameBuffer, (long)f3Previous.x, (long)f3Previous.y,
NULL);
   ::LineTo(hDCFrameBuffer, (long)f3Current.x, (long)f3Current.y);
}
```

❸ “CMesh” 클래스의 Render() 함수를 다음과 같이 수정한다.

```
void CMesh::Render(HDC hDCFrameBuffer)
{
   XMFLOAT3 f3InitialProject, f3PreviousProject;
   bool bPreviousInside = false, bInitialInside = false, bCurrentInside
= false, bIntersectInside = false;

   for (int j = 0; j < m_nPolygons; j++)
   {
      int nVertices = m_ppPolygons[j]->m_nVertices;
      CVertex* pVertices = m_ppPolygons[j]->m_pVertices;

      f3PreviousProject = f3InitialProject =
CGraphicsPipeline::Project(pVertices[0].m_xmf3Position);
      bPreviousInside = bInitialInside = (-1.0f <= f3InitialProject.x)
&& (f3InitialProject.x <= 1.0f) && (-1.0f <= f3InitialProject.y) &&
(f3InitialProject.y <= 1.0f);
      for (int i = 1; i < nVertices; i++)
      {
         XMFLOAT3 f3CurrentProject =
CGraphicsPipeline::Project(pVertices[i].m_xmf3Position);
         bCurrentInside = (-1.0f <= f3CurrentProject.x) &&
(f3CurrentProject.x <= 1.0f) && (-1.0f <= f3CurrentProject.y) &&
(f3CurrentProject.y <= 1.0f);
         if (((0.0f <= f3CurrentProject.z) && (f3CurrentProject.z <=
1.0f)) && ((bCurrentInside || bPreviousInside)))
::Draw2DLine(hDCFrameBuffer, f3PreviousProject, f3CurrentProject);
         f3PreviousProject = f3CurrentProject;
         bPreviousInside = bCurrentInside;
      }
      if (((0.0f <= f3InitialProject.z) && (f3InitialProject.z <= 1.0f))
&& ((bInitialInside || bPreviousInside))) ::Draw2DLine(hDCFrameBuffer,
f3PreviousProject, f3InitialProject);
   }
}
```

❹ “CAirplaneMesh” 클래스의 생성자를 다음과 같이 정의한다.
삼각형 24개로 다음 그림과 같은 비행기를 표현하기 위한 메쉬이다.



비행기 메쉬

```
CAirplaneMesh::CAirplaneMesh(float fWidth, float fHeight, float fDepth)
: CMesh(24)
{
   float fx = fWidth*0.5f, fy = fHeight*0.5f, fz = fDepth*0.5f;
```

```
   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
   CPolygon *pFace = new CPolygon(3);
   pFace->SetVertex(0, CVertex(0.0f, +(fy + y3), -fz));
   pFace->SetVertex(1, CVertex(+x1, -y1, -fz));
   pFace->SetVertex(2, CVertex(0.0f, 0.0f, -fz));
   SetPolygon(i++, pFace);

   pFace = new CPolygon(3);
   pFace->SetVertex(0, CVertex(0.0f, +(fy + y3), -fz));
   pFace->SetVertex(1, CVertex(0.0f, 0.0f, -fz));
   pFace->SetVertex(2, CVertex(-x1, -y1, -fz));
   SetPolygon(i++, pFace);

   pFace = new CPolygon(3);
   pFace->SetVertex(0, CVertex(+x2, +y2, -fz));
   pFace->SetVertex(1, CVertex(+fx, -y3, -fz));
   pFace->SetVertex(2, CVertex(+x1, -y1, -fz));
   SetPolygon(i++, pFace);

   pFace = new CPolygon(3);
   pFace->SetVertex(0, CVertex(-x2, +y2, -fz));
   pFace->SetVertex(1, CVertex(-x1, -y1, -fz));
   pFace->SetVertex(2, CVertex(-fx, -y3, -fz));
   SetPolygon(i++, pFace);

//비행기 메쉬의 아래쪽 면
   pFace = new CPolygon(3);
   pFace->SetVertex(0, CVertex(0.0f, +(fy + y3), +fz));
   pFace->SetVertex(1, CVertex(0.0f, 0.0f, +fz));
   pFace->SetVertex(2, CVertex(+x1, -y1, +fz));
   SetPolygon(i++, pFace);

   pFace = new CPolygon(3);
   pFace->SetVertex(0, CVertex(0.0f, +(fy + y3), +fz));
   pFace->SetVertex(1, CVertex(-x1, -y1, +fz));
   pFace->SetVertex(2, CVertex(0.0f, 0.0f, +fz));
   SetPolygon(i++, pFace);

   pFace = new CPolygon(3);
   pFace->SetVertex(0, CVertex(+x2, +y2, +fz));
   pFace->SetVertex(1, CVertex(+x1, -y1, +fz));
   pFace->SetVertex(2, CVertex(+fx, -y3, +fz));
   SetPolygon(i++, pFace);

   pFace = new CPolygon(3);
```

```
    pFace->SetVertex(0, CVertex(-x2, +y2, +fz));
    pFace->SetVertex(1, CVertex(-fx, -y3, +fz));
    pFace->SetVertex(2, CVertex(-x1, -y1, +fz));
    SetPolygon(i++, pFace);

  //비행기 메쉬의 오른쪽 면
    pFace = new CPolygon(3);
    pFace->SetVertex(0, CVertex(0.0f, +(fy + y3), -fz));
    pFace->SetVertex(1, CVertex(0.0f, +(fy + y3), +fz));
    pFace->SetVertex(2, CVertex(+x2, +y2, -fz));
    SetPolygon(i++, pFace);

    pFace = new CPolygon(3);
    pFace->SetVertex(0, CVertex(+x2, +y2, -fz));
    pFace->SetVertex(1, CVertex(0.0f, +(fy + y3), +fz));
    pFace->SetVertex(2, CVertex(+x2, +y2, +fz));
    SetPolygon(i++, pFace);

    pFace = new CPolygon(3);
    pFace->SetVertex(0, CVertex(+x2, +y2, -fz));
    pFace->SetVertex(1, CVertex(+x2, +y2, +fz));
    pFace->SetVertex(2, CVertex(+fx, -y3, -fz));
    SetPolygon(i++, pFace);

    pFace = new CPolygon(3);
    pFace->SetVertex(0, CVertex(+fx, -y3, -fz));
    pFace->SetVertex(1, CVertex(+x2, +y2, +fz));
    pFace->SetVertex(2, CVertex(+fx, -y3, +fz));
    SetPolygon(i++, pFace);

  //비행기 메쉬의 뒤쪽/오른쪽 면
    pFace = new CPolygon(3);
    pFace->SetVertex(0, CVertex(+x1, -y1, -fz));
    pFace->SetVertex(1, CVertex(+fx, -y3, -fz));
    pFace->SetVertex(2, CVertex(+fx, -y3, +fz));
    SetPolygon(i++, pFace);

    pFace = new CPolygon(3);
    pFace->SetVertex(0, CVertex(+x1, -y1, -fz));
    pFace->SetVertex(1, CVertex(+fx, -y3, +fz));
    pFace->SetVertex(2, CVertex(+x1, -y1, +fz));
    SetPolygon(i++, pFace);

    pFace = new CPolygon(3);
    pFace->SetVertex(0, CVertex(0.0f, 0.0f, -fz));
    pFace->SetVertex(1, CVertex(+x1, -y1, -fz));
    pFace->SetVertex(2, CVertex(+x1, -y1, +fz));
    SetPolygon(i++, pFace);

    pFace = new CPolygon(3);
```

```
    pFace->SetVertex(0, CVertex(0.0f, 0.0f, -fz));
    pFace->SetVertex(1, CVertex(+x1, -y1, +fz));
    pFace->SetVertex(2, CVertex(0.0f, 0.0f, +fz));
    SetPolygon(i++, pFace);

  //비행기 메쉬의 왼쪽 면
    pFace = new CPolygon(3);
    pFace->SetVertex(0, CVertex(0.0f, +(fy + y3), +fz));
    pFace->SetVertex(1, CVertex(0.0f, +(fy + y3), -fz));
    pFace->SetVertex(2, CVertex(-x2, +y2, -fz));
    SetPolygon(i++, pFace);

    pFace = new CPolygon(3);
    pFace->SetVertex(0, CVertex(0.0f, +(fy + y3), +fz));
    pFace->SetVertex(1, CVertex(-x2, +y2, -fz));
    pFace->SetVertex(2, CVertex(-x2, +y2, +fz));
    SetPolygon(i++, pFace);

    pFace = new CPolygon(3);
    pFace->SetVertex(0, CVertex(-x2, +y2, +fz));
    pFace->SetVertex(1, CVertex(-x2, +y2, -fz));
    pFace->SetVertex(2, CVertex(-fx, -y3, -fz));
    SetPolygon(i++, pFace);

    pFace = new CPolygon(3);
    pFace->SetVertex(0, CVertex(-x2, +y2, +fz));
    pFace->SetVertex(1, CVertex(-fx, -y3, -fz));
    pFace->SetVertex(2, CVertex(-fx, -y3, +fz));
    SetPolygon(i++, pFace);

  //비행기 메쉬의 뒤쪽/왼쪽 면
    pFace = new CPolygon(3);
    pFace->SetVertex(0, CVertex(0.0f, 0.0f, -fz));
    pFace->SetVertex(1, CVertex(0.0f, 0.0f, +fz));
    pFace->SetVertex(2, CVertex(-x1, -y1, +fz));
    SetPolygon(i++, pFace);

    pFace = new CPolygon(3);
    pFace->SetVertex(0, CVertex(0.0f, 0.0f, -fz));
    pFace->SetVertex(1, CVertex(-x1, -y1, +fz));
    pFace->SetVertex(2, CVertex(-x1, -y1, -fz));
    SetPolygon(i++, pFace);

    pFace = new CPolygon(3);
    pFace->SetVertex(0, CVertex(-x1, -y1, -fz));
    pFace->SetVertex(1, CVertex(-x1, -y1, +fz));
    pFace->SetVertex(2, CVertex(-fx, -y3, +fz));
    SetPolygon(i++, pFace);

    pFace = new CPolygon(3);
```

```
   pFace->SetVertex(0, CVertex(-x1, -y1, -fz));
   pFace->SetVertex(1, CVertex(-fx, -y3, +fz));
   pFace->SetVertex(2, CVertex(-fx, -y3, -fz));
   SetPolygon(i++, pFace);
}
```

⑤ “CGameObject” 클래스 선언을 다음과 같이 수정한다.

❶ “GameObject.h” 파일에 다음 헤더 파일 포함을 추가한다.

```
#include "Mesh.h“
#include "Camera.h"
```

❷ “CGameObject” 클래스를 다음과 같이 수정한다.

```
class CGameObject
{
public:
   CGameObject() { }
   ~CGameObject();

public:
   bool        m_bActive = true;

//게임 객체의 모양(메쉬, 모델)이다.
   CMesh*      m_pMesh = NULL;
//게임 객체의 월드 변환 행렬이다.
   XMFLOAT4X4  m_xmf4x4World = Matrix4x4::Identity();

//게임 객체의 색상(선분의 색상)이다.
   DWORD       m_dwColor = RGB(255, 0, 0);

//게임 객체의 이동 방향을 나타내는 벡터이다.
   XMFLOAT3    m_xmf3MovingDirection = XMFLOAT3(0.0f, 0.0f, 1.0f);
   float       m_fMovingSpeed = 0.0f;
   float       m_fMovingRange = 0.0f;

//게임 객체의 회전축을 나타내는 벡터이다.
   XMFLOAT3    m_xmf3RotationAxis = XMFLOAT3(0.0f, 1.0f, 0.0f);
   float       m_fRotationSpeed = 0.0f;

public:
   void SetMesh(CMesh *pMesh) { m_pMesh = pMesh; if (pMesh)
pMesh->AddRef(); }

   void SetActive(bool bActive) { m_bActive = bActive; }
   void SetColor(DWORD dwColor) { m_dwColor = dwColor; }

   void SetPosition(float x, float y, float z);
   void SetPosition(XMFLOAT3& xmf3Position);
```

```
    void SetMovingDirection(XMFLOAT3& xmf3MovingDirection);
    void SetMovingSpeed(float fSpeed) { m_fMovingSpeed = fSpeed; }
    void SetMovingRange(float fRange) { m_fMovingRange = fRange; }

    void SetRotationAxis(XMFLOAT3& xmf3RotationAxis);
    void SetRotationSpeed(float fSpeed) { m_fRotationSpeed = fSpeed; }

    void Move(XMFLOAT3& vDirection, float fSpeed);

    void Rotate(float fPitch, float fYaw, float fRoll);
    void Rotate(XMFLOAT3& xmf3Axis, float fAngle);

    virtual void OnUpdateTransform() { }
    virtual void Animate(float fElapsedTime);
    virtual void Render(HDC hDCFrameBuffer, CCamera* pCamera);
};
```

⑥ “CGameObject” 클래스를 다음과 같이 구현한다.

❶ “GameObject.cpp” 파일에 다음 헤더 파일 포함을 추가한다.

```
#include "stdafx.h"
#include "GameObject.h“
#include "GraphicsPipeline.h"
```

❷ CGameObject 클래스의 소멸자를 다음과 같이 정의한다.

```
CGameObject::~CGameObject()
{
    if (m_pMesh) m_pMesh->Release();
}
```

❸ SetPosition() 함수를 다음과 같이 정의한다.

```
void CGameObject::SetPosition(float x, float y, float z)
{
    m_xmf4x4World._41 = x;
    m_xmf4x4World._42 = y;
    m_xmf4x4World._43 = z;
}

void CGameObject::SetPosition(XMFLOAT3& xmf3Position)
{
    m_xmf4x4World._41 = xmf3Position.x;
    m_xmf4x4World._42 = xmf3Position.y;
    m_xmf4x4World._43 = xmf3Position.z;
}
```

❹ SetMovingDirection() 함수를 다음과 같이 정의한다.

```
void CGameObject::SetMovingDirection(XMFLOAT3& xmf3MovingDirection)
```

```
{
   XMStoreFloat3(&m_xmf3MovingDirection,
XMVector3Normalize(XMLoadFloat3(&xmf3MovingDirection)));
}
```

❺ SetRotationAxis() 함수를 다음과 같이 정의한다.

```
void CGameObject::SetRotationAxis(XMFLOAT3& xmf3RotationAxis)
{
   XMStoreFloat3(&m_xmf3RotationAxis,
XMVector3Normalize(XMLoadFloat3(&xmf3RotationAxis)));
}
```

❻ Rotate() 함수를 다음과 같이 정의한다.

```
//오일러 각도의 회전, 왼손좌표계에서 회전(자전) 행렬은 평행이동 행렬 왼쪽에 곱해야 한다.
void CGameObject::Rotate(float fPitch, float fYaw, float fRoll)
{
   XMMATRIX xmmtxRotate =
XMMatrixRotationRollPitchYaw(XMConvertToRadians(fPitch),
XMConvertToRadians(fYaw), XMConvertToRadians(fRoll));
   XMStoreFloat4x4(&m_xmf4x4World, XMMatrixMultiply(xmmtxRotate,
XMLoadFloat4x4(&m_xmf4x4World)));
}

//회전축을 중심으로 회전, 왼손좌표계에서 회전(자전) 행렬은 평행이동 행렬 왼쪽에 곱해야 한다.
void CGameObject::Rotate(XMFLOAT3& xmf3RotationAxis, float fAngle)
{
   XMMATRIX xmmtxRotate =
XMMatrixRotationAxis(XMLoadFloat3(&xmf3RotationAxis),
XMConvertToRadians(fAngle));
   XMStoreFloat4x4(&m_xmf4x4World, XMMatrixMultiply(xmmtxRotate,
XMLoadFloat4x4(&m_xmf4x4World)));
}
```

❼ Move() 함수를 다음과 같이 정의한다.

```
void CGameObject::Move(XMFLOAT3& vDirection, float fSpeed)
{
   SetPosition(m_xmf4x4World._41 + vDirection.x * fSpeed,
m_xmf4x4World._42 + vDirection.y * fSpeed, m_xmf4x4World._43 +
vDirection.z * fSpeed);
}
```

❽ Animate() 함수를 다음과 같이 정의한다.

```
void CGameObject::Animate(float fElapsedTime)
{
   if (m_fRotationSpeed != 0.0f) Rotate(m_xmf3RotationAxis,
m_fRotationSpeed * fElapsedTime);
   if (m_fMovingSpeed != 0.0f) Move(m_xmf3MovingDirection,
m_fMovingSpeed * fElapsedTime);
}
```

❾ Render() 함수를 다음과 같이 정의한다.

```
void CGameObject::Render(HDC hDCFrameBuffer, CCamera* pCamera)
{
   if (m_pMesh)
   {
      CGraphicsPipeline::SetWorldTransform(&m_xmf4x4World);

      HPEN hPen = ::CreatePen(PS_SOLID, 0, m_dwColor);
      HPEN hOldPen = (HPEN)::SelectObject(hDCFrameBuffer, hPen);
      m_pMesh->Render(hDCFrameBuffer);
      ::SelectObject(hDCFrameBuffer, hOldPen);
      ::DeleteObject(hPen);
   }
}
```

⑦ “Camera.h” 파일을 다음과 같이 수정한다.

❶ “Camera.h” 파일에서 CViewport 클래스를 다음과 같이 선언한다.

```
class CViewport
{
public:
   CViewport() { }
   virtual ~CViewport() { }

   int               m_nLeft;
   int               m_nTop;
   int               m_nWidth;
   int               m_nHeight;

   void SetViewport(int nLeft, int nTop, int nWidth, int nHeight);
};
```

❷ “Camera.h” 파일에서 CCamera 클래스를 다음과 같이 수정한다.
3인칭 카메라를 구현한다.

```
class CPlayer;

class CCamera
{
public:
   CCamera() { }
   virtual ~CCamera() { }

private:
//카메라의 위치(월드좌표계) 벡터이다.
   XMFLOAT3          m_xmf3Position = XMFLOAT3(0.0f, 0.0f, 0.0f);
//카메라의 로컬 x-축(Right), y-축(Up), z-축(Look)을 나타내는 벡터이다.*/
```

```
   XMFLOAT3          m_xmf3Right = XMFLOAT3(1.0f, 0.0f, 0.0f);
   XMFLOAT3          m_xmf3Up = XMFLOAT3(0.0f, 1.0f, 0.0f);
   XMFLOAT3          m_xmf3Look = XMFLOAT3(0.0f, 0.0f, 1.0f);

//카메라의 시야각, 투영 사각형까지의 거리
   float             m_fFOVAngle = 90.0f;
   float             m_fProjectRectDistance = 1.0f;

//뷰포트의 가로 길이와 세로 길이의 비율(종횡비: Aspect ratio)
   float             m_fAspectRatio = float(FRAMEBUFFER_WIDTH) /
float(FRAMEBUFFER_HEIGHT);

public:
//카메라 변환 행렬
   XMFLOAT4X4        m_xmf4x4View = Matrix4x4::Identity();
//원근 투영 변환 행렬
   XMFLOAT4X4        m_xmf4x4Project = Matrix4x4::Identity();
//카메라 변환 행렬 * 원근 투영 변환 행렬
   XMFLOAT4X4        m_xmf4x4ViewProject = Matrix4x4::Identity();
//뷰포트
   CViewport         m_Viewport;

public:
   void SetFOVAngle(float fFOVAngle);

//카메라 변환 행렬을 생성한다.
   void GenerateViewMatrix();
//투영 변환 행렬을 생성한다.
   void GeneratePerspectiveProjectionMatrix(float fNearPlaneDistance,
float fFarPlaneDistance, float fFOVAngle);

   void SetViewport(int nLeft, int nTop, int nWidth, int nHeight);

//3인칭 카메라에서 카메라가 바라보는 지점을 설정한다. 일반적으로 플레이어를 바라보도록 설정한다.
   void SetLookAt(XMFLOAT3& xmf3LookAt, XMFLOAT3& xmf3Up);
   void SetLookAt(XMFLOAT3& xmf3Position, XMFLOAT3& xmf3LookAt,
XMFLOAT3& xmf3Up);

//카메라를 xmf3Shift 만큼 이동한다.
   void Move(XMFLOAT3& xmf3Shift);
   void Move(float x, float y, float z);
//카메라를 x-축, y-축, z-축으로 회전한다.
   void Rotate(float fPitch = 0.0f, float fYaw = 0.0f, float fRoll =
0.0f);
//카메라의 이동, 회전에 따라 카메라의 정보를 갱신하는 가상함수이다.
   void Update(CPlayer* pPlayer, XMFLOAT3& xmf3LookAt, float
fTimeElapsed = 0.016f);
};
```

⑧ “CCamera” 클래스를 다음과 같이 구현한다.

❶ “Camera.cpp” 파일에 다음 헤더 파일 포함을 추가한다.

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

❷ “CViewport” 클래스의 SetViewport() 함수를 다음과 같이 정의한다.

```
void CViewport::SetViewport(int nLeft, int nTop, int nWidth, int
nHeight)
{
   m_nLeft = nLeft;
   m_nTop = nTop;
   m_nWidth = nWidth;
   m_nHeight = nHeight;
}
```

❸ “CCamera” 클래스의 GenerateViewMatrix() 함수를 다음과 같이 정의한다.

```
/*카메라가 여러 번 회전을 하게 되면 누적된 실수 표현의 문제 때문에 카메라의 로컬 x-축(Right), y-
축(Up), z-축(Look)이 서로 직교하지 않을 수 있다. 카메라의 로컬 x-축(Right), y-축(Up), z-축(Look)
이 서로 직교하도록 만들어준다.*/
void CCamera::GenerateViewMatrix()
{
//카메라의 z-축을 기준으로 카메라의 좌표축들이 직교하도록 만든다.
   XMVECTOR xmvLook = XMVector3Normalize(XMLoadFloat3(&m_xmf3Look));
   XMVECTOR xmvUp = XMVector3Normalize(XMLoadFloat3(&m_xmf3Up));
   XMVECTOR xmvRight = XMVector3Normalize(XMVector3Cross(xmvUp,
xmvLook));
   xmvUp = XMVector3Normalize(XMVector3Cross(xmvLook, xmvRight));

   XMStoreFloat3(&m_xmf3Look, xmvLook);
   XMStoreFloat3(&m_xmf3Right, xmvRight);
   XMStoreFloat3(&m_xmf3Up, xmvUp);

//카메라 변환 행렬은 카메라 월드변환 행렬의 역행렬(전치행렬)이다.
   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;

   XMVECTOR xmvPosition = XMLoadFloat3(&m_xmf3Position);
   m_xmf4x4View._41 = -XMVectorGetX(XMVector3Dot(xmvPosition,
xmvRight));
   m_xmf4x4View._42 = -XMVectorGetX(XMVector3Dot(xmvPosition, xmvUp));
   m_xmf4x4View._43 = -XMVectorGetX(XMVector3Dot(xmvPosition,
xmvLook));
```

```
   XMStoreFloat4x4(&m_xmf4x4ViewProject,
XMMatrixMultiply(XMLoadFloat4x4(&m_xmf4x4View),
XMLoadFloat4x4(&m_xmf4x4Project)));
}
```

❹ “CCamera” 클래스의 SetLookAt() 함수를 다음과 같이 정의한다.

```
//카메라가 xmf3LookAt을 바라보도록 카메라 변환 행렬을 갱신한다.
void CCamera::SetLookAt(XMFLOAT3& xmf3Position, XMFLOAT3& xmf3LookAt,
XMFLOAT3& xmf3Up)
{
   m_xmf3Position = xmf3Position;
   XMStoreFloat4x4(&m_xmf4x4View,
XMMatrixLookAtLH(XMLoadFloat3(&m_xmf3Position),
XMLoadFloat3(&xmf3LookAt), XMLoadFloat3(&xmf3Up)));

//카메라 변환 행렬에서 카메라의 x-축, y-축, z-축을 구한다.
   XMVECTORF32 xmf32vRight = { m_xmf4x4View._11, m_xmf4x4View._21,
m_xmf4x4View._31, 0.0f };
   XMVECTORF32 xmf32vUp = { m_xmf4x4View._12, m_xmf4x4View._22,
m_xmf4x4View._32, 0.0f };
   XMVECTORF32 xmf32vLook = { m_xmf4x4View._13, m_xmf4x4View._23,
m_xmf4x4View._33, 0.0f };

   XMStoreFloat3(&m_xmf3Right, XMVector3Normalize(xmf32vRight));
   XMStoreFloat3(&m_xmf3Up, XMVector3Normalize(xmf32vUp));
   XMStoreFloat3(&m_xmf3Look, XMVector3Normalize(xmf32vLook));
}

void CCamera::SetLookAt(XMFLOAT3& xmf3LookAt, XMFLOAT3& xmf3Up)
{
//현재 카메라의 위치에서 플레이어를 바라보기 위한 카메라 변환 행렬을 생성한다.
   SetLookAt(m_xmf3Position, xmf3LookAt, xmf3Up);
}
```

❺ “CCamera” 클래스의 SetViewport()와 SetFOVAngle() 함수를 다음과 같이 정의한다.

```
void CCamera::SetViewport(int nLeft, int nTop, int nWidth, int nHeight)
{
   m_Viewport.SetViewport(nLeft, nTop, nWidth, nHeight);
   m_fAspectRatio = float(m_Viewport.m_nWidth) /
float(m_Viewport.m_nHeight);
}

void CCamera::SetFOVAngle(float fFOVAngle)
{
   m_fFOVAngle = fFOVAngle;
   m_fProjectRectDistance = float(1.0f /
tan(XMConvertToRadians(fFOVAngle * 0.5f)));
}
```

❻ “CCamera” 클래스의 GeneratePerspectiveProjectionMatrix() 함수를 다음과 같이 정의한다.

```
//원근 투영 변환 행렬을 생성한다.
void CCamera::GeneratePerspectiveProjectionMatrix(float
fNearPlaneDistance, float fFarPlaneDistance, float fFOVAngle)
{
   float fAspectRatio = (float(m_Viewport.m_nWidth) /
float(m_Viewport.m_nHeight));
   XMStoreFloat4x4(&m_xmf4x4Project,
XMMatrixPerspectiveFovLH(XMConvertToRadians(fFOVAngle), fAspectRatio,
fNearPlaneDistance, fFarPlaneDistance));
}
```

❼ “CCamera” 클래스의 Move() 함수를 다음과 같이 정의한다.

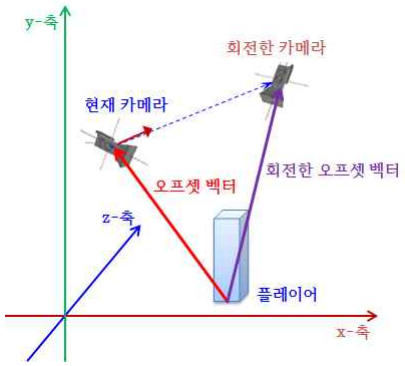
```
//카메라를 이동한다.
void CCamera::Move(XMFLOAT3& xmf3Shift)
{
   XMStoreFloat3(&m_xmf3Position,
XMVectorAdd(XMLoadFloat3(&m_xmf3Position), XMLoadFloat3(&xmf3Shift)));
}

void CCamera::Move(float x, float y, float z)
{
   Move(XMFLOAT3(x, y, z));
}
```

❽ “CCamera” 클래스의 Rotate() 함수를 다음과 같이 정의한다.

```
//카메라의 로컬 x-축(Right), y-축(Up), z-축(Look)을 기준으로 회전하는 함수이다.
void CCamera::Rotate(float fPitch, float fYaw, float fRoll)
{
   if (fPitch != 0.0f)
   {
      XMMATRIX xmmtxRotate =
XMMatrixRotationAxis(XMLoadFloat3(&m_xmf3Right),
XMConvertToRadians(fPitch));
//카메라의 로컬 x-축(Right)을 중심으로 회전하는 행렬로 y-축(Up), z-축(Look)을 회전한다.
      XMStoreFloat3(&m_xmf3Look,
XMVector3TransformNormal(XMLoadFloat3(&m_xmf3Look), xmmtxRotate));
      XMStoreFloat3(&m_xmf3Up,
XMVector3TransformNormal(XMLoadFloat3(&m_xmf3Up), xmmtxRotate));
   }
   if (fYaw != 0.0f)
   {
      XMMATRIX xmmtxRotate =
XMMatrixRotationAxis(XMLoadFloat3(&m_xmf3Up),
XMConvertToRadians(fYaw));
      XMStoreFloat3(&m_xmf3Look,
XMVector3TransformNormal(XMLoadFloat3(&m_xmf3Look), xmmtxRotate));
```

```
      XMStoreFloat3(&m_xmf3Right,
XMVector3TransformNormal(XMLoadFloat3(&m_xmf3Right), xmmtxRotate));
   }
   if (fRoll != 0.0f)
   {
      XMMATRIX xmmtxRotate =
XMMatrixRotationAxis(XMLoadFloat3(&m_xmf3Look),
XMConvertToRadians(fRoll));
      XMStoreFloat3(&m_xmf3Up,
XMVector3TransformNormal(XMLoadFloat3(&m_xmf3Up), xmmtxRotate));
      XMStoreFloat3(&m_xmf3Right,
XMVector3TransformNormal(XMLoadFloat3(&m_xmf3Right), xmmtxRotate));
   }
}
```

❾ “CCamera” 클래스의 Update() 함수를 다음과 같이 정의한다.
플레이어가 회전할 때 카메라가 천천히 플레이어의 회전을 따라 하게 한다. 3인칭 카메라에서 플레이어가 회전하면 카메라는 플레이어를 중심으로 공전을 하게 된다.

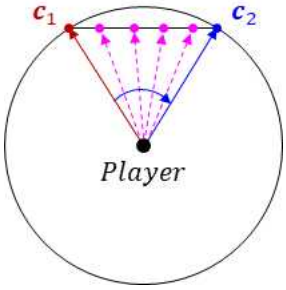


3인칭 카메라의 지연 회전

```
void CCamera::Update(CPlayer* pPlayer, XMFLOAT3& xmf3LookAt, float
fTimeElapsed)
{
//플레이어의 로컬 x-축, y-축, z-축 벡터로부터 회전 행렬(플레이어와 같은 방향을 나타내는 행렬)을
생성한다.
   XMMATRIX xmmtx4Rotate;
   xmmtx4Rotate.r[0] = XMVectorSet(pPlayer->m_xmf3Right.x,
pPlayer->m_xmf3Right.y, pPlayer->m_xmf3Right.z, 0.0f);
   xmmtx4Rotate.r[1] = XMVectorSet(pPlayer->m_xmf3Up.x,
pPlayer->m_xmf3Up.y, pPlayer->m_xmf3Up.z, 0.0f);
   xmmtx4Rotate.r[2] = XMVectorSet(pPlayer->m_xmf3Look.x,
pPlayer->m_xmf3Look.y, pPlayer->m_xmf3Look.z, 0.0f);
   xmmtx4Rotate.r[3] = XMVectorSet(0.0f, 0.0f, 0.0f, 1.0f);

   XMVECTOR xmvPosition = XMLoadFloat3(&m_xmf3Position);
//카메라 오프셋 벡터를 회전 행렬로 변환(회전)한다.
   XMVECTOR xmvOffset =
XMVector3TransformCoord(XMLoadFloat3(&pPlayer->m_xmf3CameraOffset),
xmmtx4Rotate);
```

```
//회전한 카메라의 위치는 플레이어의 위치에 회전한 카메라 오프셋 벡터를 더한 것이다.
   XMVECTOR xmvNewPosition =
XMVectorAdd(XMLoadFloat3(&pPlayer->m_xmf3Position), xmvOffset);
//현재의 카메라의 위치에서 회전한 카메라의 위치까지의 방향과 거리를 나타내는 벡터이다.
   XMVECTOR xmvDirection = XMVectorSubtract(xmvNewPosition,
xmvPosition);
   float fLength = XMVectorGetX(XMVector3Length(xmvDirection));
   xmvDirection = XMVector3Normalize(xmvDirection);
```

/*카메라의 래그(Lag)는 플레이어가 회전하더라도 카메라가 동시에 따라서 회전하지 않고 약간의 시차를 두고 회전하는 효과를 구현하기 위한 것이다. 다음 그림에서 플레이어가 회전할 때 카메라는 c1에서 c2로 시차를 두고 천천히(나누어서) 이동하면서 플레이어를 바라보도록 한다. 4.0을 40.0으로 바꾸어 회전을 해보면 차이를 느낄 수 있을 것이다.*/




```
   float fTimeLagScale = fTimeElapsed * 4.0f;
   float fDistance = fLength * fTimeLagScale;
   if (fDistance > fLength) fDistance = fLength;
   if (fLength < 0.01f) fDistance = fLength;
   if (fDistance > 0)
   {
//카메라를 공전하지 않고 이동을 한다(회전의 각도가 작은 경우 회전 이동은 선형 이동과 거의 같다).
      XMStoreFloat3(&m_xmf3Position, XMVectorAdd(xmvPosition,
XMVectorScale(xmvDirection, fDistance)));
//카메라가 플레이어를 바라보도록 한다.
      SetLookAt(pPlayer->m_xmf3Position, pPlayer->m_xmf3Up);
   }
}
```

⑨ “CPlayer” 클래스를 다음과 같이 수정한다.

❶ “Player.h” 파일에 다음 헤더 파일 포함을 추가한다.

```
#include "GameObject.h"
#include "Camera.h"
```

❷ “Player.h” 파일에서 “CPlayer” 클래스를 다음과 같이 수정한다.

```
class CPlayer : public CGameObject
{
public:
   CPlayer() { }
   virtual ~CPlayer() { if (m_pCamera) delete m_pCamera; }
```

```
public:
//플레이어의 위치 벡터, x-축(Right), y-축(Up), z-축(Look) 벡터이다.
   XMFLOAT3    m_xmf3Position = XMFLOAT3(0.0f, 0.0f, 0.0f);
   XMFLOAT3    m_xmf3Right = XMFLOAT3(1.0f, 0.0f, 0.0f);
   XMFLOAT3    m_xmf3Up = XMFLOAT3(0.0f, 1.0f, 0.0f);
   XMFLOAT3    m_xmf3Look = XMFLOAT3(0.0f, 0.0f, 1.0f);

//3인칭 카메라의 오프셋(Offset) 벡터이다.
   XMFLOAT3    m_xmf3CameraOffset = XMFLOAT3(0.0f, 0.0f, 0.0f);
//플레이어의 이동 속도를 나타내는 벡터이다.
   XMFLOAT3    m_xmf3Velocity = XMFLOAT3(0.0f, 0.0f, 0.0f);

//플레이어에 작용하는 마찰력을 나타낸다.
   float       m_fFriction = 125.0f;

//플레이어가 로컬 x-축(Right), y-축(Up), z-축(Look)으로 얼마만큼 회전했는가를 나타낸다.
   float       m_fPitch = 0.0f;
   float       m_fYaw = 0.0f;
   float       m_fRoll = 0.0f;

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

public:
   void SetPosition(float x, float y, float z);
   void SetRotation(float x, float y, float z);

//플레이어가 어떤 지점을 향하도록 하는 함수이다.
   void LookAt(XMFLOAT3& xmf3LookAt, XMFLOAT3& xmf3Up);

//플레이어를 이동하는 함수이다.
   void Move(DWORD dwDirection, float fDistance);
   void Move(XMFLOAT3& xmf3Shift, bool bUpdateVelocity);
   void Move(float x, float y, float z);

//플레이어를 회전하는 함수이다.
   void Rotate(float fPitch, float fYaw, float fRoll);

   void SetCameraOffset(XMFLOAT3& xmf3CameraOffset);

//플레이어의 위치와 회전 정보를 경과 시간에 따라 갱신하는 함수이다.
   void Update(float fTimeElapsed = 0.016f);

//플레이어의 위치와 회전축으로부터 월드 변환 행렬을 생성하는 함수이다.
   virtual void OnUpdateTransform();
   virtual void Animate(float fElapsedTime);

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

```
   CCamera* GetCamera() { return(m_pCamera); }
};
```

❸ “Player.h” 파일에서 “CAirplanePlayer” 클래스를 다음과 같이 정의한다.

```
class CAirplanePlayer : public CPlayer
{
public:
   CAirplanePlayer();
   virtual ~CAirplanePlayer();

   virtual void OnUpdateTransform();
};
```

⑩ “CPlayer” 클래스를 다음과 같이 구현한다.

❶ “Player.cpp” 파일에 다음 헤더 파일 포함을 추가한다.

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

❷ “CPlayer” 클래스의 SetPosition()과 SetCameraOffset() 함수를 다음과 같이 정의한다.

```
void CPlayer::SetPosition(float x, float y, float z)
{
   m_xmf3Position = XMFLOAT3(x, y, z);
   CGameObject::SetPosition(x, y, z);
}

void CPlayer::SetCameraOffset(XMFLOAT3& xmf3CameraOffset)
{
   m_xmf3CameraOffset = xmf3CameraOffset;
   XMFLOAT3 xmf3CameraPosition;
   XMStoreFloat3(&xmf3CameraPosition,
XMVectorAdd(XMLoadFloat3(&m_xmf3Position),
XMLoadFloat3(&m_xmf3CameraOffset)));
   m_pCamera->SetLookAt(xmf3CameraPosition, m_xmf3Position, m_xmf3Up);

   m_pCamera->GenerateViewMatrix();
}
```

❸ “CPlayer” 클래스의 Move()와 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)
{
   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) XMStoreFloat3(&xmf3Shift,
XMVectorAdd(XMLoadFloat3(&xmf3Shift),
XMVectorScale(XMLoadFloat3(&m_xmf3Look), fDistance)));
      if (dwDirection & DIR_BACKWARD) XMStoreFloat3(&xmf3Shift,
XMVectorAdd(XMLoadFloat3(&xmf3Shift),
XMVectorScale(XMLoadFloat3(&m_xmf3Look), -fDistance)));
//화살표 키 '→'를 누르면 로컬 x-축 방향으로 이동한다. '←'를 누르면 반대 방향으로 이동한다.
      if (dwDirection & DIR_RIGHT) XMStoreFloat3(&xmf3Shift,
XMVectorAdd(XMLoadFloat3(&xmf3Shift),
XMVectorScale(XMLoadFloat3(&m_xmf3Right), fDistance)));
      if (dwDirection & DIR_LEFT) XMStoreFloat3(&xmf3Shift,
XMVectorAdd(XMLoadFloat3(&xmf3Shift),
XMVectorScale(XMLoadFloat3(&m_xmf3Right), -fDistance)));
//'Page Up'을 누르면 로컬 y-축 방향으로 이동한다. 'Page Down'을 누르면 반대 방향으로 이동한다.
      if (dwDirection & DIR_UP) XMStoreFloat3(&xmf3Shift,
XMVectorAdd(XMLoadFloat3(&xmf3Shift),
XMVectorScale(XMLoadFloat3(&m_xmf3Up), fDistance)));
      if (dwDirection & DIR_DOWN) XMStoreFloat3(&xmf3Shift,
XMVectorAdd(XMLoadFloat3(&xmf3Shift),
XMVectorScale(XMLoadFloat3(&m_xmf3Up), -fDistance)));

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

void CPlayer::Move(XMFLOAT3& xmf3Shift, bool bUpdateVelocity)
{
//bUpdateVelocity가 참이면 플레이어를 이동하지 않고 속도 벡터를 변경한다.
   if (bUpdateVelocity)
   {
//플레이어의 속도 벡터를 xmf3Shift 벡터만큼 변경한다.
      XMStoreFloat3(&m_xmf3Velocity,
XMVectorAdd(XMLoadFloat3(&m_xmf3Velocity), XMLoadFloat3(&xmf3Shift)));
   }
   else
   {
//플레이어를 현재 위치 벡터에서 xmf3Shift 벡터만큼 이동한다.
      XMStoreFloat3(&m_xmf3Position,
XMVectorAdd(XMLoadFloat3(&m_xmf3Position), XMLoadFloat3(&xmf3Shift)));
//플레이어의 위치가 변경되었으므로 카메라의 위치도 xmf3Shift 벡터만큼 이동한다.
      if (m_pCamera) m_pCamera->Move(xmf3Shift);
   }
}

void CPlayer::Move(float x, float y, float z)
{
   Move(XMFLOAT3(x, y, z), false);
```

```
}
```

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

```
//플레이어를 로컬 x-축, y-축, z-축을 중심으로 회전한다.
void CPlayer::Rotate(float fPitch, float fYaw, float fRoll)
{
//카메라를 x, y, z 만큼 회전한다. 플레이어를 회전하면 카메라가 회전하게 된다.
   m_pCamera->Rotate(fPitch, fYaw, fRoll);

   if (fPitch != 0.0f)
   {
      XMMATRIX xmmtxRotate =
XMMatrixRotationAxis(XMLoadFloat3(&m_xmf3Right),
XMConvertToRadians(fPitch));
      XMStoreFloat3(&m_xmf3Look,
XMVector3TransformNormal(XMLoadFloat3(&m_xmf3Look), xmmtxRotate));
      XMStoreFloat3(&m_xmf3Up,
XMVector3TransformNormal(XMLoadFloat3(&m_xmf3Up), xmmtxRotate));
   }
   if (fYaw != 0.0f)
   {
      XMMATRIX xmmtxRotate =
XMMatrixRotationAxis(XMLoadFloat3(&m_xmf3Up),
XMConvertToRadians(fYaw));
      XMStoreFloat3(&m_xmf3Look,
XMVector3TransformNormal(XMLoadFloat3(&m_xmf3Look), xmmtxRotate));
      XMStoreFloat3(&m_xmf3Right,
XMVector3TransformNormal(XMLoadFloat3(&m_xmf3Right), xmmtxRotate));
   }
   if (fRoll != 0.0f)
   {
      XMMATRIX xmmtxRotate =
XMMatrixRotationAxis(XMLoadFloat3(&m_xmf3Look),
XMConvertToRadians(fRoll));
      XMStoreFloat3(&m_xmf3Up,
XMVector3TransformNormal(XMLoadFloat3(&m_xmf3Up), xmmtxRotate));
      XMStoreFloat3(&m_xmf3Right,
XMVector3TransformNormal(XMLoadFloat3(&m_xmf3Right), xmmtxRotate));
   }

/*회전으로 인해 플레이어의 로컬 x-축, y-축, z-축이 서로 직교하지 않을 수 있으므로 z-축(Look 벡
터)을 기준으로 하여 서로 직교하고 단위벡터가 되도록 한다.*/
   XMVECTOR xmvLook = XMVector3Normalize(XMLoadFloat3(&m_xmf3Look));
   XMVECTOR xmvUp = XMVector3Normalize(XMLoadFloat3(&m_xmf3Up));
   XMVECTOR xmvRight = XMVector3Normalize(XMVector3Cross(xmvUp,
xmvLook));
   xmvUp = XMVector3Normalize(XMVector3Cross(xmvLook, xmvRight));

   XMStoreFloat3(&m_xmf3Right, xmvRight);
   XMStoreFloat3(&m_xmf3Up, xmvUp);
```

```
   XMStoreFloat3(&m_xmf3Look, xmvLook);
}
```

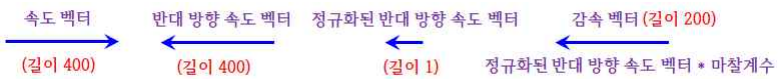
❺ “CPlayer” 클래스의 LookAt() 함수를 다음과 같이 정의한다.

```
void CPlayer::LookAt(XMFLOAT3& xmf3LookAt, XMFLOAT3& xmf3Up)
{
   XMFLOAT4X4 xmf4x4View;
   XMStoreFloat4x4(&xmf4x4View,
XMMatrixLookAtLH(XMLoadFloat3(&m_xmf3Position),
XMLoadFloat3(&xmf3LookAt), XMLoadFloat3(&xmf3Up)));

   XMVECTORF32 xmf32vRight = { xmf4x4View._11, xmf4x4View._21,
xmf4x4View._31, 0.0f };
   XMVECTORF32 xmf32vUp = { xmf4x4View._12, xmf4x4View._22,
xmf4x4View._32, 0.0f };
   XMVECTORF32 xmf32vLook = { xmf4x4View._13, xmf4x4View._23,
xmf4x4View._33, 0.0f };

   XMStoreFloat3(&m_xmf3Right, XMVector3Normalize(xmf32vRight));
   XMStoreFloat3(&m_xmf3Up, XMVector3Normalize(xmf32vUp));
   XMStoreFloat3(&m_xmf3Look, XMVector3Normalize(xmf32vLook));
}
```

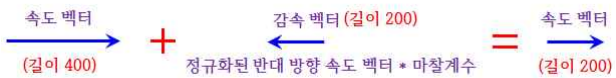
❻ “CPlayer” 클래스의 Update() 함수를 다음과 같이 정의한다.

```
//이 함수는 매 프레임마다 프레임워크에서 호출된다. 플레이어의 속도 벡터에 중력과 마찰력 등을 적용하여 플레이어를 이동한다.
void CPlayer::Update(float fTimeElapsed)
{
   Move(m_xmf3Velocity, false);

   m_pCamera->Update(this, m_xmf3Position, fTimeElapsed);
   m_pCamera->GenerateViewMatrix();
```

/*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

```
   XMVECTOR xmvVelocity = XMLoadFloat3(&m_xmf3Velocity);
   XMVECTOR xmvDeceleration =
XMVector3Normalize(XMVectorScale(xmvVelocity, -1.0f));
```

```
   float fLength = XMVectorGetX(XMVector3Length(xmvVelocity));
   float fDeceleration = m_fFriction * fTimeElapsed;
   if (fDeceleration > fLength) fDeceleration = fLength;
   XMStoreFloat3(&m_xmf3Velocity, XMVectorAdd(xmvVelocity,
XMVectorScale(xmvDeceleration, fDeceleration)));
}
```

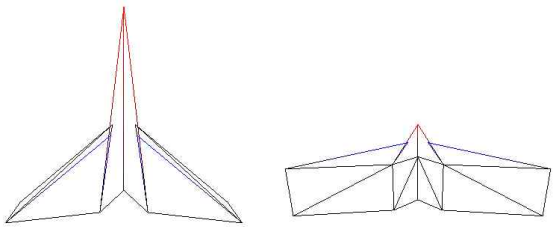
❼ "CPlayer" 클래스의 Animate() 함수를 다음과 같이 정의한다.

```
void CPlayer::Animate(float fElapsedTime)
{
//플레이어의 위치과 방향 벡터로 부터 따라 월드 변환 행렬을 구한다.
   OnUpdateTransform();

   CGameObject::Animate(fElapsedTime);
}
```

❽ "CPlayer" 클래스의 OnUpdateTransform() 함수를 다음과 같이 정의한다.

```
/*플레이어의 위치와 회전축으로부터 월드 변환 행렬을 생성하는 함수이다. 플레이어의 Right 벡터가 월
드 변환 행렬의 첫 번째 행 벡터, Up 벡터가 두 번째 행 벡터, Look 벡터가 세 번째 행 벡터, 플레이어
의 위치 벡터가 네 번째 행 벡터가 된다.*/
void CPlayer::OnUpdateTransform()
{
   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;
}
```

❾ "CAirplanePlayer" 클래스의 OnUpdateTransform() 함수를 다음과 같이 정의한다.
메쉬가 왼쪽 그림과 같이 앞 방향이 모델좌표계의 y-축 방향이므로 오른쪽 그림과 같이 렌더링하기 위하여 x-축을 중심으로 90도 회전(자전)한다.



비행기 메쉬

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

   XMStoreFloat4x4(&m_xmf4x4World,
XMMatrixMultiply(XMMatrixRotationRollPitchYaw(XMConvertToRadians(90.0f
```

```
), 0.0f, 0.0f), XMLoadFloat4x4(&m_xmf4x4World)));
}
```

⑪ “CGraphicsPipeline” 클래스를 다음과 같이 수정한다.

❶ “GraphicsPipeline.h” 파일에 다음 헤더 파일 포함을 추가한다.

```
#include "GameObject.h"
#include "Camera.h"
```

❷ “CGraphicsPipeline” 클래스를 다음과 같이 수정한다.

```
class CGraphicsPipeline
{
private:
//현재 렌더링할 게임 객체의 월드 변환 행렬이다.
   static XMFLOAT4X4*       m_pxmf4x4World;
//현재 카메라 객체의 (카메라 변환 행렬 * 원근 투영 변환 행렬)이다.
   static XMFLOAT4X4*       m_pxmf4x4ViewProject;
   static CViewport*        m_pViewport;

public:
   static void SetWorldTransform(XMFLOAT4X4* pxmf4x4World) {
m_pxmf4x4World = pxmf4x4World; }
   static void SetViewProjectTransform(XMFLOAT4X4* pxmf4x4ViewProject)
{ m_pxmf4x4ViewProject = pxmf4x4ViewProject; }

   static void SetViewport(CViewport* pViewport) { m_pViewport =
pViewport; }

   static XMFLOAT3 ScreenTransform(XMFLOAT3& xmf3Project);
   static XMFLOAT3 Project(XMFLOAT3& xmf3Model);
};
```

⑫ “CGraphicsPipeline” 클래스를 다음과 같이 구현한다.

❶ “GraphicsPipeline.cpp” 파일에 다음 헤더 파일 포함을 추가한다.

```
#include "stdafx.h"
#include "GraphicsPipeline.h"
```

❷ “CGraphicsPipeline” 클래스의 정적 멤버를 다음과 같이 정의한다.

```
XMFLOAT4X4* CGraphicsPipeline::m_pxmf4x4World = NULL;
XMFLOAT4X4* CGraphicsPipeline::m_pxmf4x4ViewProject = NULL;
CViewport* CGraphicsPipeline::m_pViewport = NULL;
```

❸ “CGraphicsPipeline” 클래스의 Project() 함수를 다음과 같이 정의한다.

```
XMFLOAT3 CGraphicsPipeline::Project(XMFLOAT3& xmf3Model)
```

```
{
   XMMATRIX xmmtxModelToProject =
XMMatrixMultiply(XMLoadFloat4x4(m_pxmf4x4World),
XMLoadFloat4x4(m_pxmf4x4ViewProject));
   XMFLOAT3 xmf3Project;
   XMStoreFloat3(&xmf3Project,
XMVector3TransformCoord(XMLoadFloat3(&xmf3Model),
xmmtxModelToProject));

   return(xmf3Project);
}
```

❹ “CGraphicsPipeline” 클래스의 ScreenTransform() 함수를 다음과 같이 정의한다.

```
XMFLOAT3 CGraphicsPipeline::ScreenTransform(XMFLOAT3& xmf3Project)
{
   XMFLOAT3 f3Screen = xmf3Project;

   float fHalfWidth = m_pViewport->m_nWidth * 0.5f;
   float fHalfHeight = m_pViewport->m_nHeight * 0.5f;
   f3Screen.x = m_pViewport->m_nLeft + (xmf3Project.x * fHalfWidth) +
fHalfWidth;
   f3Screen.y = m_pViewport->m_nTop + (-xmf3Project.y * fHalfHeight) +
fHalfHeight;

   return(f3Screen);
}
```

⑬ “CScene” 클래스 선언을 다음과 같이 수정한다.

❶ “Scene.h” 파일에 다음 헤더 파일 포함을 추가한다.

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

❷ “CScene” 클래스를 다음과 같이 수정한다.

```
class CScene
{
public:
   CScene(CPlayer *pPlayer) { m_pPlayer = pPlayer; }
   virtual ~CScene() { }

private:
   int                m_nObjects = 0;
   CGameObject**      m_ppObjects = NULL;

   CPlayer*           m_pPlayer = NULL;

public:
```

```
   virtual void BuildObjects();
   virtual void ReleaseObjects();

   virtual void Animate(float fElapsedTime);
   virtual void Render(HDC hDCFrameBuffer, CCamera* pCamera);

//윈도우 메시지(키보드, 마우스)를 처리한다.
   virtual void OnProcessingMouseMessage(HWND hWnd, UINT nMessageID,
WPARAM wParam, LPARAM lParam) { }
   virtual void OnProcessingKeyboardMessage(HWND hWnd, UINT nMessageID,
WPARAM wParam, LPARAM lParam) { }
};
```

⑭ “CScene” 클래스를 다음과 같이 구현한다.

❶ “Scene.cpp” 파일에 다음 헤더 파일 포함을 추가한다.

```
#include "stdafx.h"
#include "Scene.h"
#include "GraphicsPipeline.h"
```

❷ “CScene” 클래스의 BuildObjects()와 ReleaseObjects() 함수를 다음과 같이 정의한다.

```
void CScene::BuildObjects()
{
   CCubeMesh* pCubeMesh = new CCubeMesh(4.0f, 4.0f, 4.0f);

   m_nObjects = 5;
   m_ppObjects = new CGameObject * [m_nObjects];

   m_ppObjects[0] = new CGameObject();
   m_ppObjects[0]->SetMesh(pCubeMesh);
   m_ppObjects[0]->SetColor(RGB(255, 0, 0));
   m_ppObjects[0]->SetPosition(-13.5f, 0.0f, +14.0f);
   m_ppObjects[0]->SetRotationAxis(XMFLOAT3(1.0f, 1.0f, 0.0f));
   m_ppObjects[0]->SetRotationSpeed(90.0f);
   m_ppObjects[0]->SetMovingDirection(XMFLOAT3(1.0f, 0.0f, 0.0f));
   m_ppObjects[0]->SetMovingSpeed(0.5f);

   m_ppObjects[1] = new CGameObject();
   m_ppObjects[1]->SetMesh(pCubeMesh);
   m_ppObjects[1]->SetColor(RGB(0, 0, 255));
   m_ppObjects[1]->SetPosition(+13.5f, 0.0f, +14.0f);
   m_ppObjects[1]->SetRotationAxis(XMFLOAT3(0.0f, 1.0f, 1.0f));
   m_ppObjects[1]->SetRotationSpeed(180.0f);
   m_ppObjects[1]->SetMovingDirection(XMFLOAT3(-1.0f, 0.0f, 0.0f));
   m_ppObjects[1]->SetMovingSpeed(1.5f);

   m_ppObjects[2] = new CGameObject();
   m_ppObjects[2]->SetMesh(pCubeMesh);
```

```
   m_ppObjects[2]->SetColor(RGB(0, 255, 0));
   m_ppObjects[2]->SetPosition(0.0f, +5.0f, 20.0f);
   m_ppObjects[2]->SetRotationAxis(XMFLOAT3(1.0f, 0.0f, 1.0f));
   m_ppObjects[2]->SetRotationSpeed(30.15f);
   m_ppObjects[2]->SetMovingDirection(XMFLOAT3(1.0f, -1.0f, 0.0f));
   m_ppObjects[2]->SetMovingSpeed(0.0f);

   m_ppObjects[3] = new CGameObject();
   m_ppObjects[3]->SetMesh(pCubeMesh);
   m_ppObjects[3]->SetColor(RGB(0, 255, 255));
   m_ppObjects[3]->SetPosition(0.0f, 0.0f, 40.0f);
   m_ppObjects[3]->SetRotationAxis(XMFLOAT3(0.0f, 0.0f, 1.0f));
   m_ppObjects[3]->SetRotationSpeed(40.6f);
   m_ppObjects[3]->SetMovingDirection(XMFLOAT3(0.0f, 0.0f, 1.0f));
   m_ppObjects[3]->SetMovingSpeed(0.0f);

   m_ppObjects[4] = new CGameObject();
   m_ppObjects[4]->SetMesh(pCubeMesh);
   m_ppObjects[4]->SetColor(RGB(128, 0, 255));
   m_ppObjects[4]->SetPosition(10.0f, 10.0f, 50.0f);
   m_ppObjects[4]->SetRotationAxis(XMFLOAT3(0.0f, 1.0f, 1.0f));
   m_ppObjects[4]->SetRotationSpeed(50.06f);
   m_ppObjects[4]->SetMovingDirection(XMFLOAT3(0.0f, 1.0f, 1.0f));
   m_ppObjects[4]->SetMovingSpeed(0.0f);
}

void CScene::ReleaseObjects()
{
   for (int i = 0; i < m_nObjects; i++) if (m_ppObjects[i]) delete
m_ppObjects[i];

   if (m_ppObjects) delete[] m_ppObjects;
}
```

❸ “CScene” 클래스의 Animate() 함수를 다음과 같이 정의한다.

```
void CScene::Animate(float fElapsedTime)
{
   for (int i = 0; i < m_nObjects; i++)
m_ppObjects[i]->Animate(fElapsedTime);
}
```

❹ “CScene” 클래스의 Render() 함수를 다음과 같이 정의한다.

```
void CScene::Render(HDC hDCFrameBuffer, CCamera* pCamera)
{
   CGraphicsPipeline::SetViewport(&pCamera->m_Viewport);

CGraphicsPipeline::SetViewProjectTransform(&pCamera->m_xmf4x4ViewProje
ct);

   for (int i = 0; i < m_nObjects; i++)
```

```
m_ppObjects[i]->Render(hDCFrameBuffer, pCamera);
}
```

⑮ “CGameFramework” 클래스 선언을 다음과 같이 수정한다.

❶ “GameFramework.h” 파일에 다음 헤더 파일 포함을 추가한다.

```
#include "Player.h"
#include "Scene.h"
#include "Timer.h"
```

❷ “CGameFramework” 클래스를 다음과 같이 수정한다.

```
class CGameFramework
{
public:
   CGameFramework() { }
   ~CGameFramework() { }

private:
//윈도우 응용프로그램의 인스턴스 핸들과 주 윈도우 핸들이다.
   HINSTANCE   m_hInstance = NULL;
   HWND        m_hWnd = NULL;

//주 윈도우의 클라이언트 사각형 영역이다.
   RECT        m_rcClient;

//렌더링의 대상이 되는 화면에 해당하는 비트맵과 비트맵 디바이스 컨텍스트(Device Context)이다.
   HDC         m_hDCFrameBuffer = NULL;
   HBITMAP     m_hBitmapFrameBuffer = NULL;
   HBITMAP     m_hBitmapSelect = NULL;

//플레이어 객체이다.
   CPlayer*    m_pPlayer = NULL;
//게임 객체들을 포함하는 씬(게임 세계) 클래스이다.
   CScene*     m_pScene = NULL;

//프레임 레이트를 관리하기 위한 객체이다.
   CGameTimer  m_GameTimer;

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

//프레임 레이트를 출력하기 위한 문자열이다.
   _TCHAR      m_pszFrameRate[50];

public:
//프레임워크를 생성하는 함수이다(주 윈도우가 생성되면 호출된다).
   void OnCreate(HINSTANCE hInstance, HWND hMainWnd);
```

```
//프레임워크를 소멸하는 함수이다(응용프로그램이 종료되면 호출된다).
   void OnDestroy();

//게임 세계를 렌더링할 비트맵 표면을 생성하고, 지우고, 클라이언트 영역으로 복사한다.
   void BuildFrameBuffer();
   void ClearFrameBuffer(DWORD dwColor);
   void PresentFrameBuffer();

//렌더링할 메쉬와 게임 객체를 생성하고 소멸하는 함수이다.
   void BuildObjects();
   void ReleaseObjects();

//프레임워크의 핵심(사용자 입력, 애니메이션, 렌더링)을 구성하는 함수이다.
   void ProcessInput();
   void AnimateObjects();
   void FrameAdvance();

//윈도우 메시지(키보드, 마우스)를 처리한다.
   void OnProcessingMouseMessage(HWND hWnd, UINT nMessageID, WPARAM
wParam, LPARAM lParam);
   void OnProcessingKeyboardMessage(HWND hWnd, UINT nMessageID, WPARAM
wParam, LPARAM lParam);
   LRESULT CALLBACK OnProcessingWindowMessage(HWND hWnd, UINT
nMessageID, WPARAM wParam, LPARAM lParam);
};
```

⑯ “CGameFramework” 클래스를 다음과 같이 구현한다.

❶ “GameFramework.cpp” 파일에 다음 헤더 파일 포함을 추가한다.

```
#include "stdafx.h“
#include "GameFramework.h"
```

❷ “CGameFramework” 클래스의 OnCreate()와 OnDestroy() 함수를 다음과 같이 정의한다. OnCreate() 함수는 주 윈도우가 생성되면 호출되고, OnDestroy() 함수는 응용프로그램이 종료될 때 호출된다.

```
void CGameFramework::OnCreate(HINSTANCE hInstance, HWND hMainWnd)
{
   ::srand(timeGetTime());

   m_hInstance = hInstance;
   m_hWnd = hMainWnd;

   BuildFrameBuffer();

   BuildObjects();

   _tcscpy_s(m_pszFrameRate, _T("LabProject ("));
```

```
}

void CGameFramework::OnDestroy()
{
   ReleaseObjects();

   if (m_hBitmapFrameBuffer) ::DeleteObject(m_hBitmapFrameBuffer);
   if (m_hDCFrameBuffer) ::DeleteDC(m_hDCFrameBuffer);
}
```

❸ "CGameFramework" 클래스의 BuildFrameBuffer(), ClearFrameBuffer(), 그리고 PresentFrameBuffer() 함수를 다음과 같이 정의한다(이전과 같음).

```
void CGameFramework::BuildFrameBuffer()
{
   ...
}

void CGameFramework::ClearFrameBuffer(DWORD dwColor)
{
   ...
}

void CGameFramework::PresentFrameBuffer()
{
   ...
}
```

❹ "CGameFramework" 클래스의 BuildObjects()와 ReleaseObjects() 함수를 다음과 같이 정의한다. BuildObjects() 함수는 카메라를 생성하여 설정하고, 플레이어 객체를 생성하고, 그리고 씬(게임 세계)을 생성한다. ReleaseObjects() 함수는 BuildObjects() 함수에서 생성한 객체들을 소멸한다.

```
void CGameFramework::BuildObjects()
{
   CCamera* pCamera = new CCamera();
   pCamera->SetViewport(0, 0, FRAMEBUFFER_WIDTH, FRAMEBUFFER_HEIGHT);
   pCamera->GeneratePerspectiveProjectionMatrix(1.01f, 500.0f, 60.0f);
   pCamera->SetFOVAngle(60.0f);

//비행기 메쉬를 생성하고 플레이어 객체에 연결한다.
   CAirplaneMesh* pAirplaneMesh = new CAirplaneMesh(6.0f, 6.0f, 1.0f);

   m_pPlayer = new CAirplanePlayer();
   m_pPlayer->SetPosition(0.0f, 0.0f, 0.0f);
   m_pPlayer->SetMesh(pAirplaneMesh);
   m_pPlayer->SetColor(RGB(0, 0, 255));
   m_pPlayer->SetCamera(pCamera);
//카메라는 플레이어 객체 뒤쪽 위에서 플레이어를 바라본다.
   m_pPlayer->SetCameraOffset(XMFLOAT3(0.0f, 5.0f, -15.0f));
```

```
   m_pScene = new CScene(m_pPlayer);
   m_pScene->BuildObjects();
}

void CGameFramework::ReleaseObjects()
{
//씬 객체의 게임 객체들을 소멸하고, 씬 객체와 플레이어 객체를 소멸한다.
   if (m_pScene) m_pScene->ReleaseObjects();
   if (m_pScene) delete m_pScene;
   if (m_pPlayer) delete m_pPlayer;
}
```

❺ "CGameFramework" 클래스의 OnProcessingMouseMessage() 함수를 다음과 같이 정의한다.

```
void CGameFramework::OnProcessingMouseMessage(HWND hWnd, UINT
nMessageID, WPARAM wParam, LPARAM lParam)
{
   switch (nMessageID)
   {
//마우스 캡쳐를 하고 현재 마우스 위치를 가져온다.
   case WM_RBUTTONDOWN:
   case WM_LBUTTONDOWN:
      ::SetCapture(hWnd);
      ::GetCursorPos(&m_ptOldCursorPos);
      break;
//마우스 캡쳐를 해제한다.
   case WM_LBUTTONUP:
   case WM_RBUTTONUP:
      ::ReleaseCapture();
      break;
   case WM_MOUSEMOVE:
      break;
   default:
      if (m_pScene) m_pScene->OnProcessingMouseMessage(hWnd,
nMessageID, wParam, lParam);
      break;
   }
}
```

❻ "CGameFramework" 클래스의 OnProcessingKeyboardMessage() 함수를 다음과 같이 정의한다.

```
void CGameFramework::OnProcessingKeyboardMessage(HWND hWnd, UINT
nMessageID, WPARAM wParam, LPARAM lParam)
{
   switch (nMessageID)
   {
   case WM_KEYDOWN:
      switch (wParam)
      {
```

```
      case VK_ESCAPE:
         ::PostQuitMessage(0);
         break;
      case VK_RETURN:
         break;
      case VK_CONTROL:
         break;
      default:
         if (m_pScene) m_pScene->OnProcessingKeyboardMessage(hWnd,
nMessageID, wParam, lParam);
         break;
      }
      break;
   default:
      break;
   }
}
```

❼ “CGameFramework” 클래스의 OnProcessingWindowMessage() 함수를 다음과 같이 정의한다.

```
LRESULT CALLBACK CGameFramework::OnProcessingWindowMessage(HWND hWnd,
UINT nMessageID, WPARAM wParam, LPARAM lParam)
{
   switch (nMessageID)
   {
   case WM_LBUTTONDOWN:
   case WM_RBUTTONDOWN:
   case WM_LBUTTONUP:
   case WM_RBUTTONUP:
   case WM_MOUSEMOVE:
      OnProcessingMouseMessage(hWnd, nMessageID, wParam, lParam);
      break;
   case WM_KEYDOWN:
   case WM_KEYUP:
      OnProcessingKeyboardMessage(hWnd, nMessageID, wParam, lParam);
      break;
   }
   return(0);
}
```

❽ “CGameFramework” 클래스의 ProcessInput() 함수를 다음과 같이 정의한다.

```
void CGameFramework::ProcessInput()
{
   static UCHAR pKeyBuffer[256];
/*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))
   {
      DWORD dwDirection = 0;
```

```
        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;
//키 입력이 있으면 플레이어를 dwDirection 방향으로 이동한다(실제로는 속도 벡터를 변경한다).
        if (dwDirection) m_pPlayer->Move(dwDirection, 0.15f);
    }

    if (::GetCapture() == m_hWnd)
    {
/*마우스를 캡쳐했으면(마우스 클릭이 일어났으면) 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 윈도우가 현재 윈도우이면 마우스 버튼이 현재 윈도우의 클라이언트 영역에서 눌려진 상태를 의미한다. 마우스 버튼이 눌려진 상태에서 마우스를 좌우 또는 상하로 움직이면 플레이어를 x-축 또는 y-축으로 회전한다.*/
//마우스 커서를 화면에서 없앤다(보이지 않게 한다).
        ::SetCursor(NULL);
        POINT ptCursorPos;
//현재 마우스 커서의 위치를 가져온다.
        ::GetCursorPos(&ptCursorPos);
//마우스 버튼이 눌린 상태에서 마우스가 움직인 양을 구한다.
        float cxMouseDelta = (float)(ptCursorPos.x - m_ptOldCursorPos.x) /
3.0f;
        float cyMouseDelta = (float)(ptCursorPos.y - m_ptOldCursorPos.y) /
3.0f;
//마우스 커서의 위치를 마우스가 눌려졌던 위치로 설정한다.
        ::SetCursorPos(m_ptOldCursorPos.x, m_ptOldCursorPos.y);
        if (cxMouseDelta || cyMouseDelta)
        {
//마우스 이동이 있으면 플레이어를 회전한다.
/*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(cyMouseDelta, 0.0f, -cxMouseDelta);
            else
                m_pPlayer->Rotate(cyMouseDelta, cxMouseDelta, 0.0f);
        }
    }

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

❾ “CGameFramework” 클래스의 AnimateObjects() 함수를 다음과 같이 정의한다.

```
void CGameFramework::AnimateObjects()
{
    float fTimeElapsed = m_GameTimer.GetTimeElapsed();
```

```
   if (m_pPlayer) m_pPlayer->Animate(fTimeElapsed);
   if (m_pScene) m_pScene->Animate(fTimeElapsed);
}
```

⑩ “CGameFramework” 클래스의 FrameAdvance() 함수를 다음과 같이 정의한다.

```
void CGameFramework::FrameAdvance()
{
//타이머의 시간이 갱신되도록 하고 프레임 레이트를 계산한다. Tick(0.0f)과 비교해보라.
   m_GameTimer.Tick(60.0f);

   ProcessInput();
   AnimateObjects();

   ClearFrameBuffer(RGB(75, 45, 105));

   CCamera* pCamera = m_pPlayer->GetCamera();
   if (m_pScene) m_pScene->Render(m_hDCFrameBuffer, pCamera);
//플레이어(비행기)를 렌더링한다.
   if (m_pPlayer) m_pPlayer->Render(m_hDCFrameBuffer, pCamera);

   PresentFrameBuffer();

//현재 프레임 레이트를 윈도우 캡션(타이틀 바)에 출력한다.
   m_GameTimer.GetFrameRate(m_pszFrameRate + 12, 37);
   ::SetWindowText(m_hWnd, m_pszFrameRate);
}
```

⑰ “CGameTimer” 클래스를 생성하고(파일 이름은 “Timer.h”와 “Timer.cpp”) 다음과 같이 수정한다.

❶ “Timer.h”를 다음과 같이 수정한다.

```
// 50회의 프레임 처리시간을 누적하여 평균한다.
const ULONG MAX_SAMPLE_COUNT = 50;

class CGameTimer
{
public:
   CGameTimer();
   virtual ~CGameTimer();

//타이머의 시간을 갱신한다.
   void Tick(float fLockFPS = 0.0f);

//프레임 레이트를 문자열로 반환한다.
   unsigned long GetFrameRate(LPTSTR lpszString = NULL, int
nCharacters=0);
//현재 프레임의 경과 시간을 반환한다.
```

```
    float GetTimeElapsed();

private:
    double            m_fTimeScale;
    float             m_fTimeElapsed;

    __int64           m_nBasePerformanceCounter;
    __int64           m_nPausedPerformanceCounter;
    __int64           m_nStopPerformanceCounter;
    __int64           m_nCurrentPerformanceCounter;
    __int64           m_nLastPerformanceCounter;

    __int64           m_PerformanceFrequencyPerSec;

    float             m_fFrameTime[MAX_SAMPLE_COUNT];
    ULONG             m_nSampleCount;

    unsigned long     m_nCurrentFrameRate;
    unsigned long     m_FramePerSecond;
    float             m_fFPSTimeElapsed;
};
```

⑱ “Timer.cpp” 파일을 다음과 같이 수정한다.

❶ 헤더 파일 부분을 다음과 같이 수정한다.

```
#include "stdafx.h"
#include "Timer.h"

/*컴퓨터가 성능 카운터(Performance Counter) 하드웨어를 가지고 있으면 성능 카운터와 성능 주파수(Performance Frequency)를 사용하여 시간 단위를 설정한다. 성능 주파수를 사용할 수 없으면 멀티미디어 타이머를 사용한다. 이 경우 시간 단위는 0.001초(Millisecond)이다. 성능 카운터에 대한 자세한 설명은 윈도우 API 도움말을 참조하라.*/
CGameTimer::CGameTimer()
{
    ::QueryPerformanceFrequency((LARGE_INTEGER
*)&m_PerformanceFrequencyPerSec);
    ::QueryPerformanceCounter((LARGE_INTEGER
*)&m_nLastPerformanceCounter);
    m_fTimeScale = 1.0 / (double)m_PerformanceFrequencyPerSec;

    m_nBasePerformanceCounter = m_nLastPerformanceCounter;
    m_nPausedPerformanceCounter = 0;
    m_nStopPerformanceCounter = 0;

    m_nSampleCount = 0;
    m_nCurrentFrameRate = 0;
    m_FramePerSecond = 0;
    m_fFPSTimeElapsed = 0.0f;
}
```

```
CGameTimer::~CGameTimer()
{
}
```

❷ “CGameTimer” 클래스의 Tick() 함수를 다음과 같이 정의한다.

```
void CGameTimer::Tick(float fLockFPS)
{
   float fTimeElapsed;

   ::QueryPerformanceCounter((LARGE_INTEGER
*)&m_nCurrentPerformanceCounter);
//이 함수를 마지막으로 호출한 이후 경과한 시간을 계산한다.
   fTimeElapsed = float((m_nCurrentPerformanceCounter -
m_nLastPerformanceCounter) * m_fTimeScale);

//이 함수의 파라메터(fLockFPS)가 0보다 크면 이 시간만큼 호출한 함수를 기다리게 한다.
   if (fLockFPS > 0.0f)
   {
      while (fTimeElapsed < (1.0f / fLockFPS))
      {
         ::QueryPerformanceCounter((LARGE_INTEGER
*)&m_nCurrentPerformanceCounter);
//이 함수를 마지막으로 호출한 이후 경과한 시간을 계산한다.
         fTimeElapsed = float((m_nCurrentPerformanceCounter -
m_nLastPerformanceCounter) * m_fTimeScale);
      }
   }

//현재 시간을 m_nLastTime에 저장한다.
   m_nLastPerformanceCounter = m_nCurrentPerformanceCounter;

/* 마지막 프레임 처리 시간과 현재 프레임 처리 시간의 차이가 1초보다 작으면 현재 프레임 처리 시간
을 m_fFrameTime[0]에 저장한다. */
   if (fabsf(fTimeElapsed - m_fTimeElapsed) < 1.0f)
   {
      ::memmove(&m_fFrameTime[1], m_fFrameTime, (MAX_SAMPLE_COUNT - 1) *
sizeof(float));
      m_fFrameTime[0] = fTimeElapsed;
      if (m_nSampleCount < MAX_SAMPLE_COUNT) m_nSampleCount++;
   }

//초당 프레임 수를 1 증가시키고 현재 프레임 처리 시간을 누적하여 저장한다.
   m_FramePerSecond++;
   m_fFPSTimeElapsed += fTimeElapsed;
   if (m_fFPSTimeElapsed > 1.0f)
   {
      m_nCurrentFrameRate   = m_FramePerSecond;
      m_FramePerSecond = 0;
```

```
      m_fFPSTimeElapsed = 0.0f;
   }

//누적된 프레임 처리 시간의 평균을 구하여 프레임 처리 시간을 구한다.
   m_fTimeElapsed = 0.0f;
   for (ULONG i = 0; i < m_nSampleCount; i++) m_fTimeElapsed +=
m_fFrameTime[i];
   if (m_nSampleCount > 0) m_fTimeElapsed /= m_nSampleCount;
}
```

❸ “CGameTimer” 클래스의 GetFrameRate() 멤버 함수를 다음과 같이 정의한다.

```
//마지막 프레임 레이트를 문자열과 정수형으로 반환한다.
unsigned long CGameTimer::GetFrameRate(LPTSTR lpszString, int
nCharacters)
{
   if (lpszString)
   {
//현재 프레임 레이트를 문자열로 변환하여 lpszString 버퍼에 쓰고 “ FPS”와 결합한다.
      ::_itow_s(m_nCurrentFrameRate, lpszString, nCharacters, 10);
      ::wcscat_s(lpszString, nCharacters, _T(" FPS)"));
   }

   return(m_nCurrentFrameRate);
}
```

❹ “CGameTimer” 클래스의 GetTimeElapsed() 멤버 함수를 다음과 같이 정의한다.

```
//현재 프레임 레이트를 반환한다.
float CGameTimer::GetTimeElapsed()
{
   return(m_fTimeElapsed);
}
```

⑲ “LabProject03.cpp” 파일을 다음과 같이 수정한다.

❶ WndProc() 함수를 다음과 같이 수정한다.

```
LRESULT CALLBACK WndProc(HWND hWnd, UINT message, WPARAM wParam, LPARAM
lParam)
{
   switch (message)
   {
      case WM_LBUTTONDOWN:
      case WM_LBUTTONUP:
      case WM_RBUTTONDOWN:
      case WM_RBUTTONUP:
      case WM_MOUSEMOVE:
      case WM_KEYDOWN:
      case WM_KEYUP:
         gGameFramework.OnProcessingWindowMessage(hWnd, message, wParam,
```

```
lParam);
        break;
     case WM_DESTROY:
        ::PostQuitMessage(0);
        break;
     default:
        return(DefWindowProc(hWnd, message, wParam, lParam));
  }
  return(0);
}
```

⑳ 프로그램을 빌드하고 실행하면 다음과 같이 회전하며 천천히 이동하는 정육면체들을 볼 수 있다. 화살표 키(←: -x 축, →: +x 축, ↑: +z 축, ↓: : -z 축)를 누르거나 “PageUp” 키(+y 축)와 “PageDown” 키(-y 축)를 누르면 플레이어(카메라)를 이동할 수 있다. 마우스 오른쪽 또는 왼쪽 버튼을 누른 상태에서 이동하면 플레이어(카메라)가 회전한다.