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

(3) 소프트웨어 렌더러(Software Renderer)의 구현

지금까지 이해한 변환 파이프라인을 사용하여 3D로 표현된 모델을 2D 화면(픽셀) 좌표로 변환하여 렌더링하는 간단한 응용프로그램을 구현해 보자. 이 프로그램은 Direct3D를 사용하지 않고, 렌더링의 전체 과정을 C++ 프로그래밍 언어를 사용하여 소프트웨어적으로 구현한다. 모델을 구성하는 다각형의 변(에지)을 단순하게 선분으로 그리는 와이어프레임(Wireframe) 형태로 그린다. 이러한 선분을 그리는 API로 윈도우 API를 사용할 것이다.

소프트웨어 렌더러 응용프로그램은 CPoint, CVertex, CPolygon, CMesh, CGameObject, CScene, CPlayer, CCamera, CGameFramework, CGraphicsPipeline 클래스를 통하여 구현될 것이다.

구현하려고 하는 프로그램은 여러 C++ 클래스들을 선언과 상속 관계를 사용하며, 앞에서 설명한 3D 그래픽 기초의 내용을 기반으로 하며, 간단하지만 윈도우 API를 사용하여 윈도우 응용프로그램을 구현하기 때문에 구현 과정이 처음에는 다소 어려울 수 있다. 앞으로 Direct3D를 사용한 게임 프로그램을 작성할 때 이 내용들에 대한 이해와 구현 능력은 필수이다. 지금까지 여러 C++ 클래스들을 사용한 프로그래밍에 익숙하지 않은 학생들은 이 과정을 반드시 숙지하기 바란다. 또한, 프로그램 작성 과정에서 발생하는 문제들에 대한 이해와 해결을 하는 능력을 키워야 한다. 그러므로 단순하게 “따라하기”를 하는 것보다 제시하는 과정을 이해하려고 노력하고, 문제가 발생한다면 그 이유를 생각해보는 시간을 먼저 갖기를 바란다.

① 새로운 프로젝트 추가하기

앞에서 작성한 “LabProjects" 솔루션에 새로운 프로젝트 ”LabProject02-1“을 추가(응용프로그램 마법사를 실행)하자. ”따라하기(01)“에서와 같이 프로젝트 ”LabProject02-1“를 수정한다.

② “stdafx.h" 파일 수정하기

“stdafx.h" 파일의 마지막에 다음을 추가한다.

헤더 파일 포함 부분에 다음을 추가한다.

```
#include <stdlib.h>
#include <malloc.h>
#include <memory.h>
#include <tchar.h>
#include <math.h>
```

생성할 윈도우의 클라이언트 영역의 크기를 나타내는 상수를 다음과 같이 정의한다.

```
#define FRAMEBUFFER_WIDTH      640
#define FRAMEBUFFER_HEIGHT     480
```

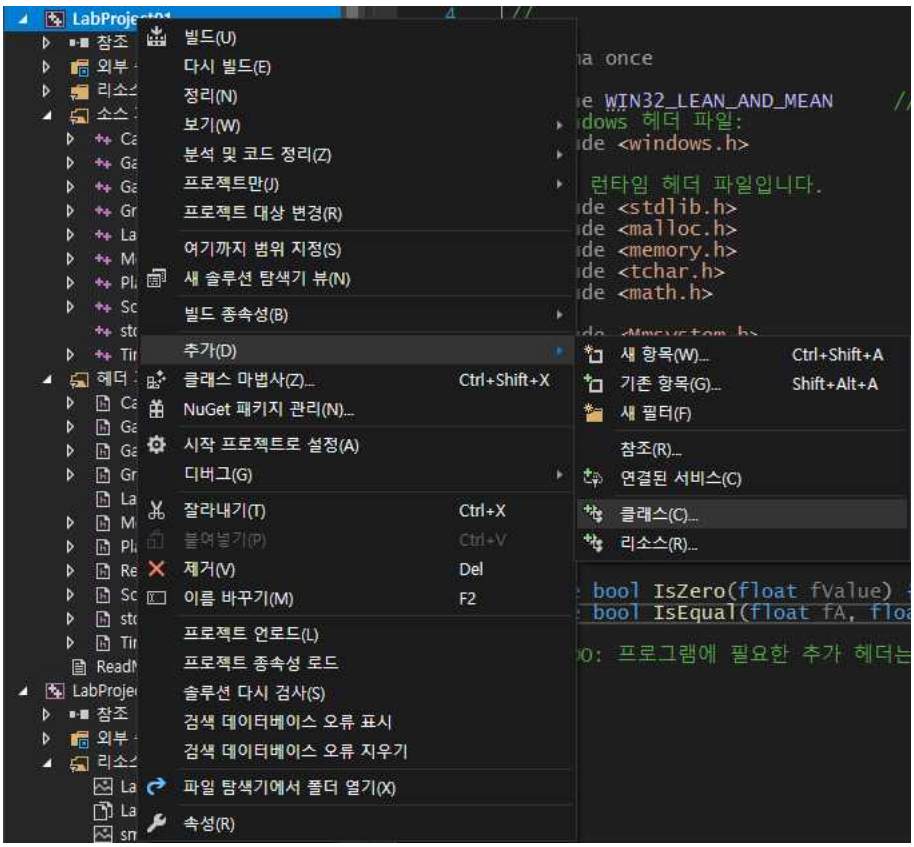
각도(Degree)를 라디언(Radian)으로 변환하는 매크로를 다음과 같이 정의한다.

```
#define DegreeToRadian(x)      float((x)*3.141592654f/180.0f)
```

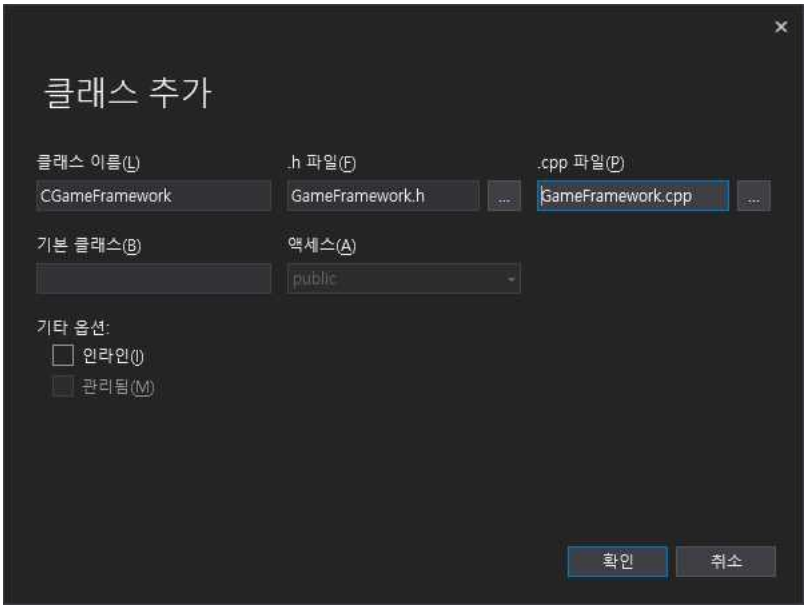
③ “CGameFramework” 클래스 생성하기

프로젝트 LabProject02-1에 게임 프로그램의 기본적 요소를 추가하기 위해 하나의 클래스(Class)를 만들도록 하자. 클래스의 이름은 “CGameFramework”이다. 이 클래스는 게임 프로그램이 가져야 될 기본적인 골격(형태)을 표현한다. 이 ”CGameFramework” 클래스는 게임 프로그램의 뼈대를 나타낸다. 우리는 이 클래스를 게임 프레임워크(Game Framework)이라고 부를 것이다. 이 클래스는 출력 화면을 생성하고 관리하며 화면 출력을 위한 여러 가지 처리(게임 객체의 생성과 관리, 사용자 입력, 애니메이션 등)를 담당한다.

다음 그림과 같이 솔루션 탐색기에서 오른쪽 마우스 버튼으로 ”LabProject02-1“을 선택하고 『추가』, 『클래스』를 차례로 선택한다. 그러면 ”클래스 추가“ 대화상자가 나타난다.



“클래스 추가”하기 메뉴



"클래스 추가"하기 대화상자

"클래스 추가" 대화상자가 나타나면『클래스 이름』에 "CGameFramework"를 입력한다. 그러면 자동적으로 ".h 파일"의 이름이 "CGameFramework.h" 그리고 ".cpp 파일"의 이름이 "CGameFramework.cpp"가 된다. ".h 파일"의 이름을 "GameFramework.h"로 수정하고 ".cpp 파일"의 이름을 "GameFramework.cpp"로 수정한다.『확인』을 선택하여 "클래스 추가" 대화상자가를 닫으면 클래스 마법사가 두 개의 파일을 생성하여 프로젝트에 추가하여 준다. 다음은 프로젝트 LabProject02-1에 추가된 "GameFramework.h" 파일과 "GameFramework.cpp" 파일의 내용이다.

- "GameFramework.h" 파일

```
#pragma once

class CGameFramework
{
};
```

- "GameFramework.cpp" 파일

```
#include "GameFramework.h"
```

앞으로 새로운 클래스를 추가할 때 이 방법을 사용하도록 한다.

④ 나머지 클래스를 생성한다.

위와 같은 방법으로 다음 클래스들을 프로젝트 LabProject02-1에 추가한다.

- CScene 파일: Scene.h, Scene.cpp
- CMesh 파일: Mesh.h, Mesh.cpp
- CGameObject 파일: GameObject.h, GameObject.cpp

- CPlayer 파일: Player.h, Player.cpp
- CCamera 파일: Camera.h, Camera.cpp
- CGraphicsPipeline 파일: GraphicsPipeline.h, GraphicsPipeline.cpp

⑤ "Mesh.h" 파일을 다음과 같이 수정한다.

❶ "Mesh.h" 파일에 CPoint3D, CVertex, CPolygon 클래스를 다음과 같이 선언한다.

```
class CPoint3D
{
public:
    CPoint3D() { }
    CPoint3D(float x, float y, float z) { this->x = x; this->y = y;
this->z = z; }
    virtual ~CPoint3D() { }

    float          x = 0.0f;
    float          y = 0.0f;
    float          z = 0.0f;
};

class CVertex
{
public:
    CVertex() { }
    CVertex(float x, float y, float z) { m_f3Position = CPoint3D(x, y,
z); }
    virtual ~CVertex() { }

    CPoint3D       m_f3Position;
};

class CPolygon
{
public:
    CPolygon() { }
    CPolygon(int nVertices);
    virtual ~CPolygon();

//다각형(면)을 구성하는 정점들의 리스트이다.
    int            m_nVertices = 0;
    CVertex        *m_pVertices = NULL;

    void SetVertex(int nIndex, CVertex vertex);
};
```

❷ "CMesh" 클래스를 다음과 같이 수정한다.

```
class CMesh
{
```

```
public:
    CMesh() { }
    CMesh(int nPolygons);
    virtual ~CMesh();

private:
//인스턴싱(Instancing)을 위하여 메쉬는 게임 객체들에 공유될 수 있다.
//다음 참조값(Reference Count)은 메쉬가 공유되는 게임 객체의 개수를 나타낸다.
    int            m_nReferences = 1;

public:
//메쉬가 게임 객체에 공유될 때마다 참조값을 1씩 증가시킨다.
    void AddRef() { m_nReferences++; }
//메쉬를 공유하는 게임 객체가 소멸될 때마다 참조값을 1씩 감소시킨다.
//참조값이 0이되면 메쉬를 소멸시킨다.
    void Release() { m_nReferences--; if (m_nReferences <= 0) delete
this; }

private:
//메쉬를 구성하는 다각형(면)들의 리스트이다.
    int            m_nPolygons = 0;
    CPolygon       **m_ppPolygons = NULL;

public:
    void SetPolygon(int nIndex, CPolygon *pPolygon);

//메쉬를 렌더링한다.
    virtual void Render(HDC hDCFrameBuffer);
};

//직육면체 클래스를 선언한다.
class CCubeMesh : public CMesh
{
public:
    CCubeMesh(float fWidth = 4.0f, float fHeight = 4.0f, float fDepth
= 4.0f);
    virtual ~CCubeMesh();
};
```

⑥ “Mesh.cpp” 파일을 다음과 같이 수정한다.

먼저, “Mesh.cpp” 파일에 다음 헤더 파일 포함을 추가한다.

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

❶ “CPolygon” 클래스의 생성자, 소멸자, SetVertex() 함수를 다음과 같이 정의한다.

```
CPolygon::CPolygon(int nVertices)
{
```

```
    m_nVertices = nVertices;
    m_pVertices = new CVertex[nVertices];
}

CPolygon::~CPolygon()
{
    if (m_pVertices) delete[] m_pVertices;
}

void CPolygon::SetVertex(int nIndex, CVertex vertex)
{
    if ((0 <= nIndex) && (nIndex < m_nVertices) && m_pVertices)
    {
       m_pVertices[nIndex] = vertex;
    }
}
```

❷ "CMesh" 클래스의 생성자와 소멸자를 다음과 같이 정의한다.

```
CMesh::CMesh(int nPolygons)
{
    m_nPolygons = nPolygons;
    m_ppPolygons = new CPolygon*[nPolygons];
}

CMesh::~CMesh()
{
    if (m_ppPolygons)
    {
       for (int i = 0; i < m_nPolygons; i++) if (m_ppPolygons[i])
delete m_ppPolygons[i];
       delete[] m_ppPolygons;
    }
}
```

❸ "CMesh" 클래스의 SetPolygon() 함수를 다음과 같이 정의한다.

```
void CMesh::SetPolygon(int nIndex, CPolygon *pPolygon)
{
//메쉬의 다각형을 설정한다.
    if ((0 <= nIndex) && (nIndex < m_nPolygons)) m_ppPolygons[nIndex]
= pPolygon;
}
```

❹ Draw2DLine() 전역 함수를 다음과 같이 정의한다.

```
void Draw2DLine(HDC hDCFrameBuffer, CPoint3D& f3PreviousProject,
CPoint3D& f3CurrentProject)
{
//투영 좌표계의 2점을 화면 좌표계로 변환하고 변환된 두 점(픽셀)을 선분으로 그린다.
    CPoint3D f3Previous =
CGraphicsPipeline::ScreenTransform(f3PreviousProject);
```

```
        CPoint3D f3Current =
    CGraphicsPipeline::ScreenTransform(f3CurrentProject);
        ::MoveToEx(hDCFrameBuffer, (long)f3Previous.x,
    (long)f3Previous.y, NULL);
        ::LineTo(hDCFrameBuffer, (long)f3Current.x, (long)f3Current.y);
    }
```

❺ "CMesh" 클래스의 Render() 함수를 다음과 같이 정의한다.

```
    void CMesh::Render(HDC hDCFrameBuffer)
    {
        CPoint3D f3InitialProject, f3PreviousProject, f3Intersect;
        bool bPreviousInside = false, bInitialInside = false,
    bCurrentInside = false, bIntersectInside = false;

//메쉬를 구성하는 모든 다각형들을 렌더링한다.
        for (int j = 0; j < m_nPolygons; j++)
        {
            int nVertices = m_ppPolygons[j]->m_nVertices;
            CVertex* pVertices = m_ppPolygons[j]->m_pVertices;

//다각형의 첫 번째 정점을 원근 투영 변환한다.
            f3PreviousProject = f3InitialProject =
    CGraphicsPipeline::Project(pVertices[0].m_f3Position);
//변환된 점이 투영 사각형에 포함되는 가를 계산한다.
            bPreviousInside = bInitialInside = (-1.0f <=
    f3InitialProject.x) && (f3InitialProject.x <= 1.0f) && (-1.0f <=
    f3InitialProject.y) && (f3InitialProject.y <= 1.0f);

//다각형을 구성하는 모든 정점들을 원근 투영 변환하고 선분으로 렌더링한다.
            for (int i = 1; i < nVertices; i++)
            {
                CPoint3D f3CurrentProject =
    CGraphicsPipeline::Project(pVertices[i].m_f3Position);
//변환된 점이 투영 사각형에 포함되는 가를 계산한다.
                bCurrentInside = (-1.0f <= f3CurrentProject.x) &&
    (f3CurrentProject.x <= 1.0f) && (-1.0f <= f3CurrentProject.y) &&
    (f3CurrentProject.y <= 1.0f);
//변환된 점이 투영 사각형에 포함되면 이전 점과 현재 점을 선분으로 그린다.
                if (((f3PreviousProject.z >= 0.0f) || (f3CurrentProject.z >=
    0.0f)) && ((bCurrentInside || bPreviousInside)))
    ::Draw2DLine(hDCFrameBuffer, f3PreviousProject, f3CurrentProject);
                f3PreviousProject = f3CurrentProject;
                bPreviousInside = bCurrentInside;
            }
//다각형의 마지막 정점과 다각형의 시작점을 선분으로 그린다.
            if (((f3PreviousProject.z >= 0.0f) || (f3InitialProject.z >=
    0.0f)) && ((bInitialInside || bPreviousInside)))
    ::Draw2DLine(hDCFrameBuffer, f3PreviousProject, f3InitialProject);
        }
```

```
}
```

❻ “CCubeMesh” 클래스의 생성자와 소멸자를 다음과 같이 정의한다.

```
CCubeMesh::CCubeMesh(float fWidth, float fHeight, float fDepth) :
CMesh(6)
{
   float fHalfWidth = fWidth * 0.5f;
   float fHalfHeight = fHeight * 0.5f;
   float fHalfDepth = fDepth * 0.5f;

   CPolygon *pFrontFace = new CPolygon(4);
   pFrontFace->SetVertex(0, CVertex(-fHalfWidth, +fHalfHeight,
-fHalfDepth));
   pFrontFace->SetVertex(1, CVertex(+fHalfWidth, +fHalfHeight,
-fHalfDepth));
   pFrontFace->SetVertex(2, CVertex(+fHalfWidth, -fHalfHeight,
-fHalfDepth));
   pFrontFace->SetVertex(3, CVertex(-fHalfWidth, -fHalfHeight,
-fHalfDepth));
   SetPolygon(0, pFrontFace);

   CPolygon *pTopFace = new CPolygon(4);
   pTopFace->SetVertex(0, CVertex(-fHalfWidth, +fHalfHeight,
+fHalfDepth));
   pTopFace->SetVertex(1, CVertex(+fHalfWidth, +fHalfHeight,
+fHalfDepth));
   pTopFace->SetVertex(2, CVertex(+fHalfWidth, +fHalfHeight,
-fHalfDepth));
   pTopFace->SetVertex(3, CVertex(-fHalfWidth, +fHalfHeight,
-fHalfDepth));
   SetPolygon(1, pTopFace);

   CPolygon *pBackFace = new CPolygon(4);
   pBackFace->SetVertex(0, CVertex(-fHalfWidth, -fHalfHeight,
+fHalfDepth));
   pBackFace->SetVertex(1, CVertex(+fHalfWidth, -fHalfHeight,
+fHalfDepth));
   pBackFace->SetVertex(2, CVertex(+fHalfWidth, +fHalfHeight,
+fHalfDepth));
   pBackFace->SetVertex(3, CVertex(-fHalfWidth, +fHalfHeight,
+fHalfDepth));
   SetPolygon(2, pBackFace);

   CPolygon *pBottomFace = new CPolygon(4);
   pBottomFace->SetVertex(0, CVertex(-fHalfWidth, -fHalfHeight,
-fHalfDepth));
   pBottomFace->SetVertex(1, CVertex(+fHalfWidth, -fHalfHeight,
-fHalfDepth));
   pBottomFace->SetVertex(2, CVertex(+fHalfWidth, -fHalfHeight,
+fHalfDepth));
```

```
    pBottomFace->SetVertex(3, CVertex(-fHalfWidth, -fHalfHeight,
+fHalfDepth));
    SetPolygon(3, pBottomFace);

    CPolygon *pLeftFace = new CPolygon(4);
    pLeftFace->SetVertex(0, CVertex(-fHalfWidth, +fHalfHeight,
+fHalfDepth));
    pLeftFace->SetVertex(1, CVertex(-fHalfWidth, +fHalfHeight,
-fHalfDepth));
    pLeftFace->SetVertex(2, CVertex(-fHalfWidth, -fHalfHeight,
-fHalfDepth));
    pLeftFace->SetVertex(3, CVertex(-fHalfWidth, -fHalfHeight,
+fHalfDepth));
    SetPolygon(4, pLeftFace);

    CPolygon *pRightFace = new CPolygon(4);
    pRightFace->SetVertex(0, CVertex(+fHalfWidth, +fHalfHeight,
-fHalfDepth));
    pRightFace->SetVertex(1, CVertex(+fHalfWidth, +fHalfHeight,
+fHalfDepth));
    pRightFace->SetVertex(2, CVertex(+fHalfWidth, -fHalfHeight,
+fHalfDepth));
    pRightFace->SetVertex(3, CVertex(+fHalfWidth, -fHalfHeight,
-fHalfDepth));
    SetPolygon(5, pRightFace);
}

CCubeMesh::~CCubeMesh()
{
}
```

⑦ “CGameObject” 클래스 선언을 다음과 같이 수정한다.

먼저, “GameObject.h” 파일에 다음 헤더 파일 포함을 추가한다.

```
#include "Mesh.h"
```

❶ “CGameObject” 클래스를 다음과 같이 수정한다.

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

private:
//게임 객체의 위치(월드 좌표계)이다.
    float          m_fxPosition = 0.0f;
    float          m_fyPosition = 0.0f;
    float          m_fzPosition = 0.0f;
```

```
//게임 객체의 x-축, y-축, z-축 회전 양(축을 기준으로 반시계 방향)이다.
	float		m_fxRotation = 0.0f;
	float		m_fyRotation = 0.0f;
	float		m_fzRotation = 0.0f;
//게임 객체의 x-축, y-축, z-축 회전 양이다.
	float		m_fxRotationSpeed = 0.0f;
	float		m_fyRotationSpeed = 0.0f;
	float		m_fzRotationSpeed = 0.0f;

//게임 객체의 모양(메쉬, 모델)이다.
	CMesh		*m_pMesh = NULL;
//게임 객체의 색상(선분의 색상)이다.
	DWORD		m_dwColor = RGB(255, 0, 0);

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

	void SetColor(DWORD dwColor) { m_dwColor = dwColor; }

	void SetPosition(float x, float y, float z) { m_fxPosition = x;
m_fyPosition = y; m_fzPosition = z; }
	void SetRotation(float x, float y, float z) { m_fxRotation = x;
m_fyRotation = y; m_fzRotation = z; }
	void SetRotationSpeed(float x, float y, float z) {
m_fxRotationSpeed = x; m_fyRotationSpeed = y; m_fzRotationSpeed = z;
}

//게임 객체를 x-축, y-축, z-축으로 이동한다.
	void Move(float x, float y, float z) { m_fxPosition += x;
m_fyPosition += y; m_fyPosition += z; }
//게임 객체를 x-축, y-축, z-축을 기준으로 회전한다.
	void Rotate(float x, float y, float z) { m_fxRotation += x;
m_fyRotation += y; m_fzRotation += z; }

public:
//메쉬의 정점 하나를 게임 객체의 위치와 방향을 사용하여 월드 좌표 변환을 한다.
	CPoint3D WorldTransform(CPoint3D& f3Model);

//게임 객체를 애니메이션 한다.
	virtual void Animate(float fElapsedTime);
//게임 객체를 렌더링한다.
	virtual void Render(HDC hDCFrameBuffer);
};
```

⑧ “CGameObject” 클래스를 다음과 같이 구현한다.

먼저, “GameObject.cpp” 파일에 다음 헤더 파일 포함을 추가한다.

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

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

❶ CGameObject 클래스의 소멸자를 다음과 같이 정의한다.

```
CGameObject::~CGameObject(void)
{
//이 게임 객체는 더 이상 메쉬를 참조하지 않으므로 메쉬의 참조값을 1 감소한다.
    if (m_pMesh) m_pMesh->Release();
}
```

❷ WorldTransform() 함수를 다음과 같이 정의한다.

```
CPoint3D CGameObject::WorldTransform(CPoint3D& f3Model)
{
    float fPitch = DegreeToRadian(m_fxRotation);
    float fYaw = DegreeToRadian(m_fyRotation);
    float fRoll = DegreeToRadian(m_fzRotation);

    CPoint3D f3World = f3Model;
    CPoint3D f3Rotated = f3Model;
//회전 변환
    if (fPitch != 0.0f)
    {
      f3Rotated.y = float(f3World.y * cos(fPitch) - f3World.z *
sin(fPitch));
      f3Rotated.z = float(f3World.y * sin(fPitch) + f3World.z *
cos(fPitch));
      f3World.y = f3Rotated.y;
      f3World.z = f3Rotated.z;
    }
    if (fYaw != 0.0f)
    {
      f3Rotated.x = float(f3World.x * cos(fYaw) + f3World.z *
sin(fYaw));
      f3Rotated.z = float(-f3World.x * sin(fYaw) + f3World.z *
cos(fYaw));
      f3World.x = f3Rotated.x;
      f3World.z = f3Rotated.z;
    }
    if (fRoll != 0.0f)
    {
      f3Rotated.x = float(f3World.x * cos(fRoll) - f3World.y *
sin(fRoll));
      f3Rotated.y = float(f3World.x * sin(fRoll) + f3World.y *
cos(fRoll));
      f3World.x = f3Rotated.x;
      f3World.y = f3Rotated.y;
    }
//평행 이동 변환
    f3World.x += m_fxPosition;
    f3World.y += m_fyPosition;
```

```
    f3World.z += m_fzPosition;

    return(f3World);
}
```

❸ Animate() 함수를 다음과 같이 정의한다.

```
void CGameObject::Animate(float fElapsedTime)
{
    Rotate(m_fxRotationSpeed * fElapsedTime, m_fyRotationSpeed *
fElapsedTime, m_fzRotationSpeed * fElapsedTime);
}
```

❹ Render() 함수를 다음과 같이 정의한다.

```
void CGameObject::Render(HDC hDCFrameBuffer)
{
    HPEN hPen = ::CreatePen(PS_SOLID, 0, m_dwColor);
    HPEN hOldPen = (HPEN)::SelectObject(hDCFrameBuffer, hPen);

    if (m_pMesh) m_pMesh->Render(hDCFrameBuffer);

    ::SelectObject(hDCFrameBuffer, hOldPen);
    ::DeleteObject(hPen);
}
```

⑨ “Camera.h” 파일을 다음과 같이 수정한다.

먼저, “Camera.h” 파일에 다음 헤더 파일 포함을 추가한다.

```
#include "Mesh.h"
```

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

```
class CViewport
{
public:
    CViewport(int nLeft, int nTop, int nWidth, int nHeight) { m_nLeft
= nLeft; m_nTop = nTop; m_nWidth = nWidth; m_nHeight = nHeight; }
    virtual ~CViewport() { }

    int             m_nLeft;
    int             m_nTop;
    int             m_nWidth;
    int             m_nHeight;
};
```

❷ “Camera.h” 파일에서 CCamera 클래스를 다음과 같이 수정한다.

```
class CCamera
{
public:
    CCamera() { }
    virtual ~CCamera() { if (m_pViewport) delete m_pViewport; }
```

```
    private:
//카메라의 위치(월드 좌표계)
        float          m_fxPosition = 0.0f;
        float          m_fyPosition = 0.0f;
        float          m_fzPosition = 0.0f;
//카메라의 회전(카메라 좌표계)
        float          m_fxRotation = 0.0f;
        float          m_fyRotation = 0.0f;
        float          m_fzRotation = 0.0f;
//카메라의 시야각, 투영 사각형까지의 거리
        float          m_fFOVAngle = 90.0f;
        float          m_fProjectRectDistance = 1.0f;
//뷰포트
        CViewport*     m_pViewport = NULL;
//뷰포트의 가로 길이와 세로 길이의 비율(종횡비: Aspect ratio)
        float          m_fAspectRatio = float(FRAMEBUFFER_WIDTH) /
    float(FRAMEBUFFER_HEIGHT);

    public:
//카메라 변환, 투영 변환, 화면 변환을 수행한다.
        CPoint3D CameraTransform(CPoint3D& f3World);
        CPoint3D ProjectionTransform(CPoint3D& f3Camera);
        CPoint3D ScreenTransform(CPoint3D& f3Projection);

        void SetPosition(float x, float y, float z) { m_fxPosition = x;
    m_fyPosition = y; m_fzPosition = z; }
        void SetRotation(float fPitch, float fYaw, float fRoll) {
    m_fxRotation = fPitch; m_fyRotation = fYaw; m_fzRotation = fRoll; }

//카메라의 뷰포트와 시야각을 설정한다.
        void SetViewport(int xStart, int yStart, int nWidth, int
    nHeight);
        void SetFOVAngle(float fFOVAngle);

//카메라를 이동하고 회전한다.
        void Move(float x, float y, float z);
        void Rotate(float fPitch, float fYaw, float fRoll);
    };
```

⑩ “CCamera” 클래스를 다음과 같이 구현한다.

먼저, “Camera.cpp” 파일에 다음 헤더 파일 포함을 추가한다.

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

❶ “CCamera” 클래스의 SetViewport()와 SetFOVAngle(), 소멸자 함수를 다음과 같이 정의한다.

```
void CCamera::SetViewport(int nLeft, int nTop, int nWidth, int
nHeight)
{
    m_pViewport = new CViewport(nLeft, nTop, nWidth, nHeight);
    m_fAspectRatio = float(m_pViewport->m_nWidth) /
float(m_pViewport->m_nHeight);
}

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

❷ “CCamera” 클래스의 Move()와 Rotate() 함수를 다음과 같이 정의한다.

```
void CCamera::Move(float x, float y, float z)
{
    m_fxPosition += x;
    m_fyPosition += y;
    m_fzPosition += z;
}

void CCamera::Rotate(float fPitch, float fYaw, float fRoll)
{
    m_fxRotation += fPitch;
    m_fyRotation += fYaw;
    m_fzRotation += fRoll;
}
```

❸ “CCamera” 클래스의 CameraTransform() 함수를 다음과 같이 정의한다.
CameraTransform()은 월드 좌표계의 점을 카메라 좌표계로 변환하는 함수이다.

```
CPoint3D CCamera::CameraTransform(CPoint3D& f3World)
{
//카메라를 월드 좌표계의 원점으로 이동한다.
    CPoint3D f3Camera = f3World;
    f3Camera.x -= m_fxPosition;
    f3Camera.y -= m_fyPosition;
    f3Camera.z -= m_fzPosition;

    float fPitch = DegreeToRadian(-m_fxRotation);
    float fYaw = DegreeToRadian(-m_fyRotation);
    float fRoll = DegreeToRadian(-m_fzRotation);

//카메라를 월드 좌표계의 축과 일치하도록 회전한다.
    CPoint3D f3Rotated = f3Camera;
    if (fPitch != 0.0f)
    {
      f3Rotated.y = float(f3Camera.y * cos(fPitch) - f3Camera.z *
```

```
sin(fPitch));
      f3Rotated.z = float(f3Camera.y * sin(fPitch) + f3Camera.z *
cos(fPitch));
      f3Camera.y = f3Rotated.y;
      f3Camera.z = f3Rotated.z;
   }
   if (fYaw != 0.0f)
   {
      f3Rotated.x = float(f3Camera.x * cos(fYaw) + f3Camera.z *
sin(fYaw));
      f3Rotated.z = float(-f3Camera.x * sin(fYaw) + f3Camera.z *
cos(fYaw));
      f3Camera.x = f3Rotated.x;
      f3Camera.z = f3Rotated.z;
   }
   if (fRoll != 0.0f)
   {
      f3Rotated.x = float(f3Camera.x * cos(fRoll) - f3Camera.y *
sin(fRoll));
      f3Rotated.y = float(f3Camera.x * sin(fRoll) + f3Camera.y *
cos(fRoll));
      f3Camera.x = f3Rotated.x;
      f3Camera.y = f3Rotated.y;
   }

   return(f3Camera);
}
```

❹ "CCamera" 클래스의 ProjectionTransform() 함수를 다음과 같이 정의한다.
ProjectionTransform()은 카메라 좌표계의 점을 투영 좌표계로 변환하는 함수이다.

```
CPoint3D CCamera::ProjectionTransform(CPoint3D& f3Camera)
{
   CPoint3D f3Project = f3Camera;
   if (f3Camera.z != 0.0f)
   {
//카메라의 시야각이 90°가 아닌 경우 투영 사각형까지의 거리를 곱한다.
      f3Project.x = float((f3Camera.x * m_fProjectRectDistance) /
(m_fAspectRatio * f3Camera.z));
      f3Project.y = float((f3Camera.y * m_fProjectRectDistance) /
f3Camera.z);
//투영 좌표계는 2차원이므로 z-좌표에 카메라 좌표계 z-좌표를 저장한다.
      f3Project.z = f3Camera.z;
   }

   return(f3Project);
}
```

❺ "CCamera" 클래스의 ScreenTransform() 함수를 다음과 같이 정의한다.
ScreenTransform()은 투영 좌표계의 점을 화면 좌표계로 변환하는 함수이다.

```
CPoint3D CCamera::ScreenTransform(CPoint3D& f3Projection)
{
    CPoint3D f3Screen = f3Projection;

    float fHalfWidth = m_pViewport->m_nWidth * 0.5f;
    float fHalfHeight = m_pViewport->m_nHeight * 0.5f;
    f3Screen.x = (f3Projection.x * fHalfWidth) + m_pViewport->m_nLeft
+ fHalfWidth;
    f3Screen.y = (-f3Projection.y * fHalfHeight) +
m_pViewport->m_nTop + fHalfHeight;

    return(f3Screen);
}
```

⑪ “CPlayer” 클래스를 다음과 같이 수정한다.

먼저, “Player.h” 파일에 다음 헤더 파일 포함을 추가한다.

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

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

```
//플레이어 객체도 게임 객체이므로 CGameObject 클래스에서 상속하여 클래스를 생성한다.
class CPlayer : public CGameObject
{
public:
    CPlayer() { }
    virtual ~CPlayer() { if (m_pCamera) delete m_pCamera; }

private:
//플레이어 객체에 포함된 카메라이다.
    CCamera*      m_pCamera = NULL;

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

    void Move(float x, float y, float z);
    void Rotate(float x, float y, float z);

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

⑫ “CPlayer” 클래스를 다음과 같이 구현한다.

먼저, “Player.cpp” 파일에 다음 헤더 파일 포함을 추가한다.

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

❶ “CPlayer” 클래스의 SetPosition()와 SetRotation() 함수를 다음과 같이 정의한다.

```
void CPlayer::SetPosition(float x, float y, float z)
{
//플레이어 객체의 위치와 카메라의 위치를 설정한다.
	CGameObject::SetPosition(x, y, z);
	if (m_pCamera) m_pCamera->SetPosition(x, y, z);
}

void CPlayer::SetRotation(float x, float y, float z)
{
//플레이어 객체와 카메라의 회전 각도를 설정한다.
	CGameObject::SetRotation(x, y, z);
	if (m_pCamera) m_pCamera->SetRotation(x, y, z);
}
```

❷ “CPlayer” 클래스의 Move()와 Rotate() 함수를 다음과 같이 정의한다.

```
void CPlayer::Move(float x, float y, float z)
{
//플레이어 객체와 카메라를 이동한다.
	if (m_pCamera) m_pCamera->Move(x, y, z);
	CGameObject::Move(x, y, z);
}

void CPlayer::Rotate(float fPitch, float fYaw, float fRoll)
{
//플레이어 객체와 카메라를 회전한다.
	if (m_pCamera) m_pCamera->Rotate(fPitch, fYaw, fRoll);
	CGameObject::Rotate(fPitch, fYaw, fRoll);
}
```

⑬ “CGraphicsPipeline” 클래스를 다음과 같이 수정한다.

먼저, “GraphicsPipeline.h” 파일에 다음 헤더 파일 포함을 추가한다.

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

❶ “CGraphicsPipeline” 클래스를 다음과 같이 수정한다.

“CGraphicsPipeline” 클래스는 모델 좌표계의 점을 월드 변환, 카메라 변환, 원근 투영 변환을 수행하기 위한 클래스이다.

```
class CGraphicsPipeline
{
private:
	static CGameObject*  m_pGameObject;
	static CCamera*      m_pCamera;

public:
	static void SetGameObject(CGameObject* pGameObject) {
```

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

    static CPoint3D ScreenTransform(CPoint3D& f3Projection);
    static CPoint3D Project(CPoint3D& f3Model);
};
```

⑭ “CGraphicsPipeline” 클래스를 다음과 같이 구현한다.

먼저, “GraphicsPipeline.cpp” 파일에 다음 헤더 파일 포함을 추가한다.

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

❶ “CGraphicsPipeline” 클래스의 정적 멤버를 다음과 같이 정의한다.

```
CGameObject* CGraphicsPipeline::m_pGameObject = NULL;
CCamera* CGraphicsPipeline::m_pCamera = NULL;
```

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

Project() 함수는 모델 좌표계의 점을 월드 변환, 카메라 변환, 원근 투영 변환을 순차적으로 수행한다. 월드 변환은 게임 객체의 정보(위치와 방향)가 필요하다. 카메라 변환은 카메라의 정보(위치와 방향)가 필요하다. 원근 투영 변환은 카메라의 정보(시야각과 종횡비)가 필요하다.

```
CPoint3D CGraphicsPipeline::Project(CPoint3D& f3Model)
{
    CPoint3D f3World = m_pGameObject->WorldTransform(f3Model);
    CPoint3D f3Camera = m_pCamera->CameraTransform(f3World);
    CPoint3D f3Projection = m_pCamera->ProjectionTransform(f3Camera);

    return(f3Projection);
}
```

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

ScreenTransform() 함수는 투영 좌표계의 점을 화면 변환을 수행한다. 화면 변환은 카메라의 정보(뷰포트)가 필요하다.

```
CPoint3D CGraphicsPipeline::ScreenTransform(CPoint3D& f3Projection)
{
    CPoint3D f3Screen = m_pCamera->ScreenTransform(f3Projection);

    return(f3Screen);
}
```

⑮ “CScene” 클래스 선언을 다음과 같이 수정한다.

먼저, “Scene.h” 파일에 다음 헤더 파일 포함을 추가한다.

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

❶ “CScene” 클래스를 다음과 같이 수정한다.

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

private:
//게임 객체들의 개수와 게임 객체들의 리스트(배열)이다.
    int            m_nObjects = 0;
    CGameObject    **m_ppObjects = NULL;

public:
//게임 객체들을 생성하고 소멸한다.
    virtual void BuildObjects();
    virtual void ReleaseObjects();

//게임 객체들을 애니메이션한다.
    virtual void Animate(float fElapsedTime);
//게임 객체들을 렌더링한다.
    virtual void Render(HDC hDCFrameBuffer, CCamera* pCamera);
};
```

⑯ “CScene” 클래스를 다음과 같이 구현한다.

먼저 “Scene.cpp” 파일에 다음 헤더 파일 포함을 추가한다.

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

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

```
void CScene::BuildObjects()
{
//직육면체 메쉬를 생성한다.
    CCubeMesh *pCubeMesh = new CCubeMesh(8.0f, 8.0f, 8.0f);

//게임 객체 2 개를 생성한다.
    m_nObjects = 2;
    m_ppObjects = new CGameObject*[m_nObjects];

    m_ppObjects[0] = new CGameObject();
    m_ppObjects[0]->SetMesh(pCubeMesh);
    m_ppObjects[0]->SetPosition(-8.5f, 0.0f, -14.0f);
    m_ppObjects[0]->SetRotation(0.0f, 0.0f, 0.0f);
    m_ppObjects[0]->SetRotationSpeed(5.0f, 30.0f, 9.0f);
    m_ppObjects[0]->SetColor(RGB(255, 0, 0));

    m_ppObjects[1] = new CGameObject();
    m_ppObjects[1]->SetMesh(pCubeMesh);
```

```
    m_ppObjects[1]->SetPosition(+8.5f, 0.0f, -14.0f);
    m_ppObjects[1]->SetRotation(0.0f, 0.0f, 0.0f);
    m_ppObjects[1]->SetRotationSpeed(30.0f, 9.0f, 5.0f);
    m_ppObjects[1]->SetColor(RGB(0, 0, 255));
}

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;
}
```

❷ “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);
}
```

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

```
void CScene::Render(HDC hDCFrameBuffer, CCamera* pCamera)
{
//현재 카메라를 렌더링 파이프라인에 설정한다.
    if (pCamera) CGraphicsPipeline::SetCamera(pCamera);

    for (int i = 0; i < m_nObjects; i++)
    {
//현재 게임 객체를 렌더링 파이프라인에 설정한다.
        CGraphicsPipeline::SetGameObject(m_ppObjects[i]);
//현재 게임 객체를 렌더링한다.
        m_ppObjects[i]->Render(hDCFrameBuffer);
    }
}
```

⑰ “CGameFramework” 클래스 선언을 다음과 같이 수정한다.

먼저, “GameFramework.h” 파일에 다음 헤더 파일 포함을 추가한다.

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

❶ “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;

    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();
    };
```

⑱ “CGameFramework” 클래스를 다음과 같이 구현한다.
먼저 “GameFramework.cpp” 파일에 다음 헤더 파일 포함을 추가한다.

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

❶ “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)
{
    m_hInstance = hInstance;
    m_hWnd = hMainWnd;
//렌더링 화면을 생성한다.
    BuildFrameBuffer();
//플레이어와 게임 세계(씬)을 생성한다.
    BuildObjects();
}

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

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

❷ “CGameFramework” 클래스의 BuildFrameBuffer(), ClearFrameBuffer(), 그리고 PresentFrameBuffer() 함수를 다음과 같이 정의한다. 화면 깜박임을 줄이기 위하여 윈도우의 클라이언트 영역의 디바이스 컨텍스트를 사용하여 직접 클라이언트 표면에 그리지 않는다. BuildFrameBuffer() 함수는 클라이언트 표면을 대신할 비트맵 생성하고 이 비트맵을 메모리 디바이스 컨텍스트로 생성한다. ClearFrameBuffer() 함수는 렌더링을 시작하기 전에 비트맵 표면을 원하는 색상으로 지운다. 씬을 화면(클라이언트 영역)으로 렌더링하기 위하여 먼저 씬의 게임 객체들을 비트맵 표면으로 렌더링한다. PresentFrameBuffer() 함수는 비트맵 표면으로 렌더링된 비트맵을 클라이언트 영역으로 옮긴다.

```
void CGameFramework::BuildFrameBuffer()
{
    ::GetClientRect(m_hWnd, &m_rcClient);

    HDC hDC = ::GetDC(m_hWnd);
    m_hDCFrameBuffer = ::CreateCompatibleDC(hDC);
    m_hBitmapFrameBuffer = ::CreateCompatibleBitmap(hDC,
m_rcClient.right - m_rcClient.left, m_rcClient.bottom -
m_rcClient.top);
    ::SelectObject(m_hDCFrameBuffer, m_hBitmapFrameBuffer);
    ::ReleaseDC(m_hWnd, hDC);
    ::SetBkMode(m_hDCFrameBuffer, TRANSPARENT);
}

void CGameFramework::ClearFrameBuffer(DWORD dwColor)
{
    HPEN hPen = ::CreatePen(PS_SOLID, 0, dwColor);
    HPEN hOldPen = (HPEN)::SelectObject(m_hDCFrameBuffer, hPen);
    HBRUSH hBrush = ::CreateSolidBrush(dwColor);
    HBRUSH hOldBrush = (HBRUSH)::SelectObject(m_hDCFrameBuffer,
hBrush);
```

```
    ::Rectangle(m_hDCFrameBuffer, m_rcClient.left, m_rcClient.top,
m_rcClient.right, m_rcClient.bottom);
    ::SelectObject(m_hDCFrameBuffer, hOldBrush);
    ::SelectObject(m_hDCFrameBuffer, hOldPen);
    ::DeleteObject(hPen);
    ::DeleteObject(hBrush);
}

void CGameFramework::PresentFrameBuffer()
{
    HDC hDC = ::GetDC(m_hWnd);
    ::BitBlt(hDC, m_rcClient.left, m_rcClient.top, m_rcClient.right -
m_rcClient.left, m_rcClient.bottom - m_rcClient.top,
m_hDCFrameBuffer, m_rcClient.left, m_rcClient.top, SRCCOPY);
    ::ReleaseDC(m_hWnd, hDC);
}
```

❸ “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()
{
//카메라를 생성하고 뷰포트와 시야각(FOV)를 설정한다.
    CCamera* pCamera = new CCamera();
    pCamera->SetViewport(0, 0, FRAMEBUFFER_WIDTH,
FRAMEBUFFER_HEIGHT);
    pCamera->SetFOVAngle(60.0f);

//플레이어 게임 객체를 생성하고 카메라와 위치를 설정한다.
    m_pPlayer = new CPlayer();
    m_pPlayer->SetCamera(pCamera);
    m_pPlayer->SetPosition(0.0f, 3.0f, -40.0f);

//씬 객체를 생성하고 게임 객체들을 생성한다.
    m_pScene = new CScene();
    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;
}
```

❹ “CGameFramework” 클래스의 ProcessInput() 함수를 다음과 같이 정의한다. ProcessInput() 함수는 키보드 입력을 처리하여 플레이어 객체를 이동한다.

```
void CGameFramework::ProcessInput()
{
    static UCHAR pKeyBuffer[256];
    if (::GetKeyboardState(pKeyBuffer))
    {
       float cxKeyDelta = 0.0f, cyKeyDelta = 0.0f, czKeyDelta = 0.0f;
       if (pKeyBuffer[VK_UP] & 0xF0) czKeyDelta = +0.125f;
       if (pKeyBuffer[VK_DOWN] & 0xF0) czKeyDelta = -0.125f;
       if (pKeyBuffer[VK_LEFT] & 0xF0) cxKeyDelta = -0.125f;
       if (pKeyBuffer[VK_RIGHT] & 0xF0) cxKeyDelta = +0.125f;
       if (pKeyBuffer[VK_PRIOR] & 0xF0) cyKeyDelta = +0.125f;
       if (pKeyBuffer[VK_NEXT] & 0xF0) cyKeyDelta = -0.125f;

       m_pPlayer->Move(cxKeyDelta, cyKeyDelta, czKeyDelta);
    }
}
```

❺ "CGameFramework" 클래스의 AnimateObjects() 함수를 다음과 같이 정의한다. AnimateObjects() 함수는 씬의 게임 객체들을 애니메이션한다.

```
void CGameFramework::AnimateObjects()
{
    if (m_pScene) m_pScene->Animate(1.0f / 60.0f);
}
```

❻ "CGameFramework" 클래스의 FrameAdvance() 함수를 다음과 같이 정의한다. FrameAdvance() 함수는 윈도우 메시지 루프에서 반복적으로 호출된다. 즉, 응용프로그램이 실행되는 동안 이 함수가 반복적으로 계속 실행된다. 이 함수는 사용자 입력을 받아 플레이어 또는 게임 세계의 게임 객체들을 움직이고 그 결과에 따라 게임 세계를 화면으로 렌더링한다.

```
void CGameFramework::FrameAdvance()
{
//사용자 입력을 처리한다.
    ProcessInput();
//게임 세계를 애니메이션(움직이게)한다.
    AnimateObjects();

//렌더링을 할 대상 화면(비트맵)을 지운다.
    ClearFrameBuffer(RGB(90, 103, 224));

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

//렌더링을 한 화면(비트맵)을 클라이언트 영역으로 복사한다.
    PresentFrameBuffer();
}
```

⑲ “LabProject02-1.cpp” 파일을 다음과 같이 수정한다.

❶ 헤더 파일 포함 부분을 다음과 같이 수정한다.

```
#include "stdafx.h"
#include "LabProject02-1.h"
#include "GameFramework.h"
```

❷ “CGameFramework” 클래스 객체를 전역변수로 선언한다. 이 객체는 게임 프로그램의 골격을 나타내는 객체이다.

```
CGameFramework          gGameFramework;
```

❸ WinMain() 함수의 메시지 루프(Message Loop) 부분을 다음과 같이 변경한다.

```
...
while (1)
{
   if (::PeekMessage(&msg, NULL, 0, 0, PM_REMOVE))
   {
      if (msg.message == WM_QUIT) break;
      if (!::TranslateAccelerator(msg.hwnd, hAccelTable, &msg))
      {
         ::TranslateMessage(&msg);
         ::DispatchMessage(&msg);
      }
   }
   else
   {
      gGameFramework.FrameAdvance();
   }
}
gGameFramework.OnDestroy();
...
```

Visual Studio 응용 프로그램 마법사가 생성한 코드의 메시지 루프는 응용 프로그램이 처리해야 할 윈도우 메시지가 메시지 큐에 있으면 꺼내서 처리를 하고, 처리할 메시지가 없으면(메시지 큐가 비어 있으면) CPU를 운영체제로 반납하도록 되어있다(GetMessage() API 함수). 그러나 게임 프로그램은 프로그램이 처리할 메시지가 없더라도 화면 렌더링, 사용자 입력처리, 길찾기 등의 작업이 계속 진행되어야 한다. 그러므로 만약 처리할 메시지가 없더라도 CPU를 반납하지 않고 게임 프로그램이 계속 실행(CPU를 사용)되도록 메시지 루프를 변경해야 한다. 이를 위해서 윈도우 메시지 루프를 PeekMessage() API 함수를 사용하여 변경한다. PeekMessage() API 함수는 메시지 큐를 살펴보고 메시지가 있으면 메시지를 꺼내고 TRUE를 반환하고, 만약 메시지 큐에 메시지가 없으면 FALSE를 반환한다. 그러므로 PeekMessage() 함수가 TRUE를 반환하는 경우(응용 프로그램이 처리해야 할 윈도우 메시지가 메시지 큐에 있으면) 정상적인 윈도우 메시지 처리 과정을 수행해야 한다. 그러나 메시지 큐가 비어있으면(처리할 윈도우 메시지가 없어서 PeekMessage() 함수가 FALSE를 반환하면) gGameFramework.FrameAdvance() 함수를 호출하여 게임 프로그램

이 CPU를 계속 사용할 수 있도록 해야 메시지 루프를 수정한다.

응용프로그램이 종료되면 PostQuitMessage() 함수가 호출되고 WM_QUIT 메시지가 메시지 큐에 들어간다. 메시지 루프에서 꺼내온 메시지가 WM_QUIT 메시지이면 메시지 루프가 종료된다. 메시지 루프가 종료되면 gGameFramework.OnDestroy() 함수를 호출하여 프레임워크를 종료(OnDestroy()를 호출)하도록 한다.

❹ MyRegisterClass() 함수를 다음과 같이 변경한다.

```
ATOM MyRegisterClass(HINSTANCE hInstance)
{
    WNDCLASSEX wcex;

    wcex.cbSize = sizeof(WNDCLASSEX);
    wcex.style = CS_HREDRAW | CS_VREDRAW;
    wcex.lpfnWndProc = WndProc;
    wcex.cbClsExtra = 0;
    wcex.cbWndExtra = 0;
    wcex.hInstance = hInstance;
    wcex.hIcon = ::LoadIcon(hInstance,
MAKEINTRESOURCE(IDI_LABPROJECT02));
    wcex.hCursor = ::LoadCursor(NULL, IDC_ARROW);
    wcex.hbrBackground   = (HBRUSH)(COLOR_WINDOW+1);
//주 윈도우의 메뉴가 나타나지 않도록 한다.
    wcex.lpszMenuName = NULL;
    wcex.lpszClassName   = szWindowClass;
    wcex.hIconSm = ::LoadIcon(wcex.hInstance,
MAKEINTRESOURCE(IDI_SMALL));

    return ::RegisterClassEx(&wcex);
}
```

❺ InitInstance() 함수를 다음과 같이 변경한다.

```
BOOL InitInstance(HINSTANCE hInstance, int nCmdShow)
{
    hInst = hInstance;
//주 윈도우의 클라이언트 영역의 크기를 원하는 크기로 설정한다.
    RECT rc = { 0, 0, FRAME_BUFFER_WIDTH, FRAME_BUFFER_HEIGHT };
    DWORD dwStyle = WS_OVERLAPPED | WS_CAPTION | WS_MINIMIZEBOX |
WS_SYSMENU | WS_BORDER;
    AdjustWindowRect(&rc, dwStyle, FALSE);
    HWND hMainWnd = CreateWindow(szWindowClass, szTitle, dwStyle,
CW_USEDEFAULT, CW_USEDEFAULT, rc.right - rc.left, rc.bottom -
rc.top, NULL, NULL, hInstance, NULL);
    if (!hMainWnd) return(FALSE);

//프로그램의 주 윈도우가 생성되면 CGameFramework 클래스의 OnCreate() 함수를 호출하여 프레임
워크 객체를 초기화하도록 한다.
    gGameFramework.OnCreate(hInstance, hMainWnd);
```

```
    ::ShowWindow(hMainWnd, nCmdShow);
    ::UpdateWindow(hMainWnd);

    return(TRUE);
}
```

⑳ 프로그램을 빌드하고 실행하면 다음과 같이 회전하는 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.