Proiect la Sisteme Multimedia

Bucur Dan-Alexandru

Semigrupa 243/2

Clasa camera:

```
class CXCamera
{
private:
protected:
D3DXVECTOR3 m_Position;
D3DXVECTOR3 m_LookAt;
D3DXVECTOR3 m_Right;
D3DXVECTOR3 m_Up;

float m_fRotAboutUp;
   float m_fRotAboutRight;
   float m_fRotAboutFacing;

D3DXMATRIX m_ViewTransform;

LPDIRECT3DDEVICE9 m_pDevice;
bool m_UpdateRequired;

HRESULT UpdateCameraMatrices();
public:
void LookAtPos(D3DXVECTOR3 *Position, D3DXVECTOR3 *LookAt, D3DXVECTOR3 *Up);
void SetPosition(FLOAT X, FLOAT Y, FLOAT Z);
D3DXVECTOR3* GetPosition() {return &m_Position;}
D3DXVECTOR3* GetLookAt() {return &m_LookAt;}
D3DXVECTOR3* GetRight() {return &m_Right;}
D3DXVECTOR3* GetUp() {return &m_Up;}
```

```
D3DXMATRIX* GetViewMatrix() {return &m_ViewTransform;}
CXCamera(LPDIRECT3DDEVICE9 pDevice);
HRESULT Update();

void RotateDown(float fAngle);// in radian nu grade
void RotateRight(float fAngle);
void Roll(float fAngle);
void MoveForward(float fDist);
void MoveRight(float fDist);
void MoveUp(float fDist);
void MoveInDirection(float fDist, D3DXVECTOR3* Dir);
};

//-----------------------------------------------------------------------------

CXCamera::CXCamera(LPDIRECT3DDEVICE9 pDevice)
{
m_Position = D3DXVECTOR3(0.0f,0.0f,0.0f);
m_LookAt = D3DXVECTOR3(0.0f,0.0f,1.0f);
m_Right = D3DXVECTOR3(1.0f,0.0f,0.0f);
m_Up = D3DXVECTOR3(0.0f,1.0f,0.0f);
m_UpdateRequired = false;

m_fRotAboutUp = m_fRotAboutRight = m_fRotAboutFacing = 0.0f;
D3DXMatrixIdentity(&m_ViewTransform);

m_pDevice = pDevice;
}

HRESULT CXCamera::Update()
{
```

```
if(m_pDevice)
{
if(m_UpdateRequired)
return UpdateCameraMatrices();

return m_pDevice->SetTransform(D3DTS_VIEW, &m_ViewTransform);
}

return E_FAIL;
}

HRESULT CXCamera::UpdateCameraMatrices()
{
D3DXMATRIX matTotal;
D3DXMATRIX matRotAboutUp, matRotAboutRight, matRotAboutFacing;

D3DXMatrixRotationAxis(&matRotAboutRight,&m_Right,m_fRotAboutRight);
   D3DXMatrixRotationAxis(&matRotAboutUp,&m_Up,m_fRotAboutUp);
   D3DXMatrixRotationAxis(&matRotAboutFacing,&m_LookAt,m_fRotAboutFacing);

D3DXMatrixMultiply(&matTotal,&matRotAboutUp,&matRotAboutRight);
   D3DXMatrixMultiply(&matTotal,&matRotAboutFacing,&matTotal);

D3DXVec3TransformCoord(&m_Right,&m_Right,&matTotal);
   D3DXVec3TransformCoord(&m_Up,&m_Up,&matTotal);
   D3DXVec3Cross(&m_LookAt,&m_Right,&m_Up);

if(fabs(D3DXVec3Dot(&m_Up,&m_Right)) > 0.01)
{
D3DXVec3Cross(&m_Up,&m_LookAt,&m_Right);
   }
```

```
D3DXVec3Normalize(&m_Right,&m_Right);
   D3DXVec3Normalize(&m_Up,&m_Up);
   D3DXVec3Normalize(&m_LookAt,&m_LookAt);

float fView41,fView42,fView43;
   fView41 = -D3DXVec3Dot(&m_Right,&m_Position);
   fView42 = -D3DXVec3Dot(&m_Up,&m_Position);
   fView43 = -D3DXVec3Dot(&m_LookAt,&m_Position);

m_ViewTransform = D3DXMATRIX(m_Right.x, m_Up.x, m_LookAt.x, 0.0f,
 m_Right.y, m_Up.y, m_LookAt.y, 0.0f,
                        m_Right.z, m_Up.z, m_LookAt.z, 0.0f,
                        fView41, fView42,  fView43,  1.0f);

m_fRotAboutUp = m_fRotAboutRight = m_fRotAboutFacing = 0.0f;
m_UpdateRequired = false;

return m_pDevice->SetTransform(D3DTS_VIEW, &m_ViewTransform);
}

void CXCamera::LookAtPos(D3DXVECTOR3 *Position, D3DXVECTOR3 *LookAt, D3DXVECTOR3 *Up)
{
D3DXMatrixLookAtLH(&m_ViewTransform, Position, LookAt, Up);

m_Position = *(Position);

m_Right.x = m_ViewTransform._11;
m_Right.y = m_ViewTransform._21;
m_Right.z = m_ViewTransform._31;
```

```
m_Up.x = m_ViewTransform._12;
m_Up.y = m_ViewTransform._22;
m_Up.z = m_ViewTransform._32;

m_LookAt.x = m_ViewTransform._13;
m_LookAt.y = m_ViewTransform._23;
m_LookAt.z = m_ViewTransform._33;

m_fRotAboutUp = m_fRotAboutRight = m_fRotAboutFacing = 0.0f;
}

void CXCamera::SetPosition(FLOAT X, FLOAT Y, FLOAT Z)
{
m_Position = D3DXVECTOR3(X, Y, Z);
m_UpdateRequired = true;
}

void CXCamera::RotateDown(float fAngle)
{
m_fRotAboutRight += fAngle;
m_UpdateRequired = true;
}

void CXCamera::RotateRight(float fAngle)
{
m_fRotAboutUp += fAngle;
m_UpdateRequired = true;
}

void CXCamera::Roll(float fAngle)
{
```

```
m_fRotAboutFacing += fAngle;
m_UpdateRequired = true;
}

void CXCamera::MoveForward(float fDist)
{
m_Position += fDist * m_LookAt;
m_UpdateRequired = true;
}

void CXCamera::MoveRight(float fDist)
{
m_Position += fDist * m_Right;
m_UpdateRequired = true;
}

void CXCamera::MoveUp(float fDist)
{
m_Position += fDist * m_Up;
m_UpdateRequired = true;
}

void CXCamera::MoveInDirection(float fDist, D3DXVECTOR3* Dir)
{
D3DXVECTOR3 DirToMove(0,0,0);
D3DXVec3Normalize(&DirToMove, Dir);
m_Position += fDist*DirToMove;
m_UpdateRequired = true;
}
```

Main:

```
#include <Windows.h>
#include <mmsystem.h>
#include <d3dx9.h>
#include <d3dx9tex.h>
#include "camera.h"
#include <dinput.h>
#include <iostream>
#include "dshow.h"

#pragma comment (lib, "d3d9.lib")
#pragma comment (lib, "d3dx9.lib")
#pragma comment (lib, "dinput8.lib")
#pragma comment (lib, "dxguid.lib")

#define DIMOUSE_LEFTBUTTON 0
#define WM_GRAPHNOTIFY  WM_APP + 1

LPDIRECT3D9           g_pD3D = NULL;
LPDIRECT3DDEVICE9     g_pd3dDevice = NULL;

CXCamera                                  g_camera = NULL;

LPD3DXMESH            g_pMesh = NULL;
D3DMATERIAL9* g_pMeshMaterials = NULL;
LPDIRECT3DTEXTURE9* g_pMeshTextures = NULL;
```

```
DWORD               g_dwNumMaterials = 0L;
LPDIRECT3DVERTEXBUFFER9 g_pMeshVBuffer = NULL;

IGraphBuilder* graphBuilder = NULL;
IMediaControl* mediaControl = NULL;
IMediaEventEx* mediaEvent = NULL;

LPDIRECTINPUT8                  g_pDin;
LPDIRECTINPUTDEVICE8        g_pDinKeyboard;
LPDIRECTINPUTDEVICE8        g_pDinmouse;
DIMOUSESTATE                    g_pMousestate;
BYTE                            g_Keystate[256];

LPDIRECT3DVERTEXBUFFER9   g_pSkyVertexBuffer = NULL;
LPDIRECT3DTEXTURE9          g_SkyTextures;

double rot_degreeX = 4.8;
double rot_degreeY = 4.8;
double rot_degreeZ = 4.8;
double movmentSpeed = 0.5;
float PozX = 0.0f;
float PozY = -48.0f;
float PozZ = 0.0f;

struct CUSTOMVERTEX
{
   float x, y, z;
   float tu, tv;
};

#define FVF_FLAGS (D3DFVF_XYZ | D3DFVF_TEX1)
```

```
CUSTOMVERTEX g_SkyboxMesh[24] =
{
   {-50.0f, -50.0f,  50.0f,  0.0f, 1.0f },
   {-50.0f,  50.0f,  50.0f,  0.0f, 0.0f },
   { 50.0f, -50.0f,  50.0f,  1.0f, 1.0f },
   { 50.0f,  50.0f,  50.0f,  1.0f, 0.0f },
   { 50.0f, -50.0f, -50.0f,  0.0f, 1.0f },
   { 50.0f,  50.0f, -50.0f,  0.0f, 0.0f },
   {-50.0f, -50.0f, -50.0f,  1.0f, 1.0f },
   {-50.0f,  50.0f, -50.0f,  1.0f, 0.0f },
   {-50.0f, -50.0f, -50.0f,  0.0f, 1.0f },
   {-50.0f,  50.0f, -50.0f,  0.0f, 0.0f },
   {-50.0f, -50.0f,  50.0f,  1.0f, 1.0f },
   {-50.0f,  50.0f,  50.0f,  1.0f, 0.0f },
   { 50.0f, -50.0f,  50.0f,  0.0f, 1.0f },
   { 50.0f,  50.0f,  50.0f,  0.0f, 0.0f },
   { 50.0f, -50.0f, -50.0f,  1.0f, 1.0f },
   { 50.0f,  50.0f, -50.0f,  1.0f, 0.0f },
   {-50.0f,  50.0f,  50.0f,  0.0f, 1.0f },
   {-50.0f,  50.0f, -50.0f,  0.0f, 0.0f },
   { 50.0f,  50.0f,  50.0f,  1.0f, 1.0f },
   { 50.0f,  50.0f, -50.0f,  1.0f, 0.0f },
   {-50.0f, -50.0f, -50.0f,  0.0f, 1.0f },
   {-50.0f, -50.0f,  50.0f,  0.0f, 0.0f },
   { 50.0f, -50.0f, -50.0f,  1.0f, 1.0f },
   { 50.0f, -50.0f,  50.0f,  1.0f, 0.0f }
};

HRESULT BuildSkybox()
{
```

```
   HRESULT hRet;
   hRet = g_pd3dDevice->CreateVertexBuffer(sizeof(CUSTOMVERTEX) * 24, 0, FVF_FLAGS,
D3DPOOL_MANAGED, &g_pSkyVertexBuffer, NULL);
   if (FAILED(hRet))
   {
      MessageBox(NULL, "Failed to create the vertex buffer!", "Error in BuildSkybox()", MB_OK |
MB_ICONSTOP);
      return false;
   }

   void* pVertices = NULL;
   g_pSkyVertexBuffer->Lock(0, sizeof(CUSTOMVERTEX) * 24, (void**)&pVertices, 0);
   memcpy(pVertices, g_SkyboxMesh, sizeof(CUSTOMVERTEX) * 24);
   g_pSkyVertexBuffer->Unlock();

   hRet = D3DXCreateTextureFromFile(g_pd3dDevice, ("sabaton.bmp"), &g_SkyTextures);

   if (FAILED(hRet))
   {
      MessageBox(NULL, "Failed to open 1 or more images files!", "Error Opening Texture Files",
MB_OK | MB_ICONSTOP);
      return E_FAIL;
   }

   return S_OK;
}

VOID RenderSkyBox()
{
   g_pd3dDevice->SetFVF(FVF_FLAGS);
   g_pd3dDevice->SetStreamSource(0, g_pSkyVertexBuffer, 0, sizeof(CUSTOMVERTEX));
```

```
    for (ULONG i = 0; i < 6; ++i)
    {
        g_pd3dDevice->SetTexture(0, g_SkyTextures);
        g_pd3dDevice->DrawPrimitive(D3DPT_TRIANGLESTRIP, i * 4, 2);
    }
}

HRESULT InitD3D(HWND hWnd)
{
    if (NULL == (g_pD3D = Direct3DCreate9(D3D_SDK_VERSION)))
        return E_FAIL;

    D3DPRESENT_PARAMETERS d3dpp;
    ZeroMemory(&d3dpp, sizeof(d3dpp));
    d3dpp.Windowed = TRUE;
    d3dpp.SwapEffect = D3DSWAPEFFECT_DISCARD;
    d3dpp.BackBufferFormat = D3DFMT_UNKNOWN;
    d3dpp.EnableAutoDepthStencil = TRUE;
    d3dpp.AutoDepthStencilFormat = D3DFMT_D16;

    if (FAILED(g_pD3D->CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_HAL, hWnd,
        D3DCREATE_SOFTWARE_VERTEXPROCESSING,
        &d3dpp, &g_pd3dDevice)))
    {
        if (FAILED(g_pD3D->CreateDevice(D3DADAPTER_DEFAULT, D3DDEVTYPE_REF, hWnd,
            D3DCREATE_SOFTWARE_VERTEXPROCESSING,
            &d3dpp, &g_pd3dDevice)))
            return E_FAIL;
    }

    g_pd3dDevice->SetRenderState(D3DRS_ZENABLE, TRUE);
```

```
    g_pd3dDevice->SetRenderState(D3DRS_AMBIENT, 0xffffffff);

    return S_OK;
}

HRESULT InitDShow(HWND hWnd) {

    HRESULT hr = CoCreateInstance(CLSID_FilterGraph,
        NULL,
        CLSCTX_INPROC_SERVER,
        IID_IGraphBuilder,
        (void**)&graphBuilder);

    hr = graphBuilder->QueryInterface(IID_IMediaControl, (void**)&mediaControl);
    hr = graphBuilder->QueryInterface(IID_IMediaEventEx, (void**)&mediaEvent);

    hr = graphBuilder->RenderFile(L"tiger.avi", NULL);
    mediaEvent->SetNotifyWindow((OAHWND)hWnd, WM_GRAPHNOTIFY, 0);
    return S_OK;
}

HRESULT InitDInput(HINSTANCE hInstance, HWND hWnd)
{
    DirectInput8Create(hInstance,
        DIRECTINPUT_VERSION,
        IID_IDirectInput8,
        (void**)&g_pDin,
        NULL);

    g_pDin->CreateDevice(GUID_SysKeyboard,
```

```
        &g_pDinKeyboard,
        NULL);

    g_pDin->CreateDevice(GUID_SysMouse,
        &g_pDinmouse,
        NULL);

    g_pDinKeyboard->SetDataFormat(&c_dfDIKeyboard);
    g_pDinKeyboard->SetCooperativeLevel(hWnd, DISCL_NONEXCLUSIVE | DISCL_FOREGROUND);

    g_pDinmouse->SetDataFormat(&c_dfDIMouse);
    g_pDinmouse->SetCooperativeLevel(hWnd, DISCL_NONEXCLUSIVE | DISCL_FOREGROUND);

    return S_OK;
}

VOID DetectInput()
{
    g_pDinKeyboard->Acquire();
    g_pDinmouse->Acquire();

    g_pDinKeyboard->GetDeviceState(256, (LPVOID)g_Keystate);
    g_pDinmouse->GetDeviceState(sizeof(DIMOUSESTATE), (LPVOID)&g_pMousestate);
}

VOID CleanDInput()
{
    g_pDinKeyboard->Unacquire();
    g_pDinmouse->Unacquire();
    g_pDin->Release();
}
```

```
HRESULT InitCamera()
{
   D3DXVECTOR3 vPosition(0.0f, -46.0f, -23.0f);
   D3DXVECTOR3 vLookatPt(0.0f, -49.0f, 1.0f);
   D3DXVECTOR3 vUpVec(0.0f, 1.0f, 0.0f);

   g_camera = CXCamera(g_pd3dDevice);
   g_camera.SetPosition(vPosition.x, vPosition.y, vPosition.z);
   g_camera.LookAtPos(&vPosition, &vLookatPt, &vUpVec);

   return S_OK;
}

VOID SetupMatrices()
{
   D3DXMATRIXA16 matProj;
   D3DXMatrixPerspectiveFovLH(&matProj, D3DX_PI / 4, 1.0f, 1.0f, 300.0f);
   g_pd3dDevice->SetTransform(D3DTS_PROJECTION, &matProj);
}

VOID ComputeInput(HWND* hWnd)
{
   if (g_pMousestate.rgbButtons[DIMOUSE_LEFTBUTTON] & 0x80)
   {
      if (g_pMousestate.lX < 0)
         g_camera.RotateRight(D3DXToRadian(-1.0f));
      else if (g_pMousestate.lX > 0)
         g_camera.RotateRight(D3DXToRadian(1.0f));

      if (g_pMousestate.lY < 0)
```

```
        g_camera.RotateDown(D3DXToRadian(-1.0f));
    else if (g_pMousestate.lY > 0)
        g_camera.RotateDown(D3DXToRadian(1.0f));
}

if (g_Keystate[DIK_W] & 0x80)
    g_camera.MoveForward(1.0f);

if (g_Keystate[DIK_S] & 0x80)
    g_camera.MoveForward(-1.0f);

if (g_Keystate[DIK_A] & 0x80)
    g_camera.MoveRight(-1.0f);

if (g_Keystate[DIK_D] & 0x80)
    g_camera.MoveRight(1.0f);

if (g_Keystate[DIK_UP] & 0x80)
{
    PozZ += 0.5;
}

if (g_Keystate[DIK_DOWN] & 0x80)
{
    PozZ -= 0.5;
}

if (g_Keystate[DIK_RIGHT] & 0x80)
{
    PozX += 0.5;
}
```

```
if (g_Keystate[DIK_LEFT] & 0x80)
{
   PozX -= 0.5;
}

if (g_Keystate[DIK_Z] & 0x80)
{
   PozY += 0.5f;
}

if (g_Keystate[DIK_X] & 0x80)
{
   PozY -= 0.5f;
}

if (g_Keystate[DIK_U] & 0x80)
{
   rot_degreeX -= D3DXToRadian(1);
}

if (g_Keystate[DIK_J] & 0x80)
{
   rot_degreeX += D3DXToRadian(1);
}

if (g_Keystate[DIK_I] & 0x80)
{
   rot_degreeY -= D3DXToRadian(1);
}
```

```
if (g_Keystate[DIK_K] & 0x80)
{
   rot_degreeY += D3DXToRadian(1);
}

if (g_Keystate[DIK_O] & 0x80)
{
   rot_degreeZ -= D3DXToRadian(1);
}

if (g_Keystate[DIK_L] & 0x80)
{
   rot_degreeZ += D3DXToRadian(1);
}

if (g_Keystate[DIK_R] & 0x80)
{
   InitCamera();
   rot_degreeX = 4.8;
   rot_degreeY = 4.8;
   rot_degreeZ = 4.8;
   PozX = 0.0f;
   PozY = -48.0f;
   PozZ = 0.0f;
}

if (g_Keystate[DIK_P] & 0x80)
{
   mediaControl->Run();
}
```

```
    if (g_Keystate[DIK_ESCAPE] & 0x80)
        PostMessage(*hWnd, WM_DESTROY, 0, 0);
}

HRESULT InitGeometry()
{
    LPD3DXBUFFER pD3DXMtrlBuffer;

    if (FAILED(D3DXLoadMeshFromX("PanzerTiger.x", D3DXMESH_SYSTEMMEM,
        g_pd3dDevice, NULL,
        &pD3DXMtrlBuffer, NULL, &g_dwNumMaterials,
        &g_pMesh)))
    {
        MessageBox(NULL, "Could not find PanzerTiger.x", "Meshes.exe", MB_OK);
        return E_FAIL;
    }

    D3DXMATERIAL* d3dxMaterials = (D3DXMATERIAL*)pD3DXMtrlBuffer->GetBufferPointer();

    g_pMeshMaterials = new D3DMATERIAL9[g_dwNumMaterials];
    g_pMeshTextures = new LPDIRECT3DTEXTURE9[g_dwNumMaterials];

    for (DWORD i = 0; i < g_dwNumMaterials; i++)
    {
        g_pMeshMaterials[i] = d3dxMaterials[i].MatD3D;
        g_pMeshMaterials[i].Ambient = g_pMeshMaterials[i].Diffuse;

        g_pMeshTextures[i] = NULL;
        if (d3dxMaterials[i].pTextureFilename != NULL &&
            lstrlen(d3dxMaterials[i].pTextureFilename) > 0)
        {
```

```
            if (FAILED(D3DXCreateTextureFromFile(g_pd3dDevice,
                d3dxMaterials[i].pTextureFilename,
                &g_pMeshTextures[i])))
            {
                MessageBox(NULL, "Could not find texture map", "Meshes.exe", MB_OK);
            }

        }
    }

    pD3DXMtrlBuffer->Release();

    if (BuildSkybox() == E_FAIL)
        return E_FAIL;

    return S_OK;
}

VOID Cleanup()
{
    if (g_pMeshMaterials != NULL)
        delete[] g_pMeshMaterials;

    if (g_pMeshTextures)
    {
        for (DWORD i = 0; i < g_dwNumMaterials; i++)
        {
            if (g_pMeshTextures[i])
                g_pMeshTextures[i]->Release();
        }
```

```
        delete[] g_pMeshTextures;
    }

    if (g_pMesh != NULL)
        g_pMesh->Release();

    g_SkyTextures->Release();
    g_SkyTextures = NULL;

    if (graphBuilder)
        graphBuilder->Release();

    if (mediaControl)
        mediaControl->Release();

    if (mediaEvent)
        mediaEvent->Release();

    if (g_pd3dDevice != NULL)
        g_pd3dDevice->Release();


    if (g_pD3D != NULL)
        g_pD3D->Release();
}

VOID Render()
{
    g_pd3dDevice->Clear(0, NULL, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER, D3DCOLOR_XRGB(0, 0,
0), 1.0f, 0);
```

```
if (SUCCEEDED(g_pd3dDevice->BeginScene()))
{
    SetupMatrices();

    D3DXMATRIXA16 matWorldOrig;
    g_pd3dDevice->GetTransform(D3DTS_WORLD, &matWorldOrig);

    RenderSkyBox();

    D3DXMATRIXA16 matWorld, leMat;

    g_pd3dDevice->GetTransform(D3DTS_WORLD, &matWorld);
    leMat = matWorld;

    D3DXMatrixRotationX(&matWorld, rot_degreeX);
    leMat *= matWorld;

    D3DXMatrixRotationY(&matWorld, rot_degreeY);
    leMat *= matWorld;

    D3DXMatrixRotationZ(&matWorld, rot_degreeZ);
    leMat *= matWorld;

    D3DXMatrixTranslation(&matWorld, PozX, PozY, PozZ);
    leMat *= matWorld;

    g_pd3dDevice->SetTransform(D3DTS_WORLD, &leMat);

    g_pd3dDevice->SetFVF(g_pMesh->GetFVF());
    for (DWORD i = 0; i < g_dwNumMaterials; i++)
    {
```

```
            g_pd3dDevice->SetMaterial(&g_pMeshMaterials[i]);
            g_pd3dDevice->SetTexture(0, g_pMeshTextures[i]);
            g_pMesh->DrawSubset(i);
        }

        g_camera.Update();

        g_pd3dDevice->SetTransform(D3DTS_WORLD, &matWorldOrig);

        g_pd3dDevice->EndScene();
    }

    g_pd3dDevice->Present(NULL, NULL, NULL, NULL);
}


LRESULT WINAPI MsgProc(HWND hWnd, UINT msg, WPARAM wParam, LPARAM lParam)
{
    switch (msg)
    {
    case WM_DESTROY:
        Cleanup();
        CleanDInput();
        PostQuitMessage(0);
        return 0;
    }

    return DefWindowProc(hWnd, msg, wParam, lParam);
}
```

```
INT WINAPI WinMain(HINSTANCE hInst, HINSTANCE, LPSTR, INT)
{
   WNDCLASSEX wc = { sizeof(WNDCLASSEX), CS_CLASSDC, MsgProc, 0L, 0L,
                     GetModuleHandle(NULL), NULL, NULL, NULL, NULL,
                     "D3D Proiect", NULL };
   RegisterClassEx(&wc);

   HWND hWnd = CreateWindow("D3D Proiect", "Bucur Dan-Alexandru: Proiect la SM",
      WS_OVERLAPPEDWINDOW, 100, 100, 1024, 768,
      GetDesktopWindow(), NULL, wc.hInstance, NULL);

   HRESULT hr = CoInitialize(NULL);

   if (SUCCEEDED(InitD3D(hWnd)))
   {

      if (SUCCEEDED(InitGeometry()))
      {
         InitDInput(hInst, hWnd);
         InitCamera();

         ShowWindow(hWnd, SW_SHOWDEFAULT);
         UpdateWindow(hWnd);
         InitDShow(hWnd);

         MSG msg;
         ZeroMemory(&msg, sizeof(msg));
         while (msg.message != WM_QUIT)
         {
            if (PeekMessage(&msg, NULL, 0U, 0U, PM_REMOVE))
```

```
            {
                TranslateMessage(&msg);
                DispatchMessage(&msg);
            }
            else
            {
                DetectInput();
                ComputeInput(&hWnd);
                Render();
            }
        }
    }
    CoUninitialize();
    UnregisterClass("D3D Project", wc.hInstance);
    return 0;
}
```