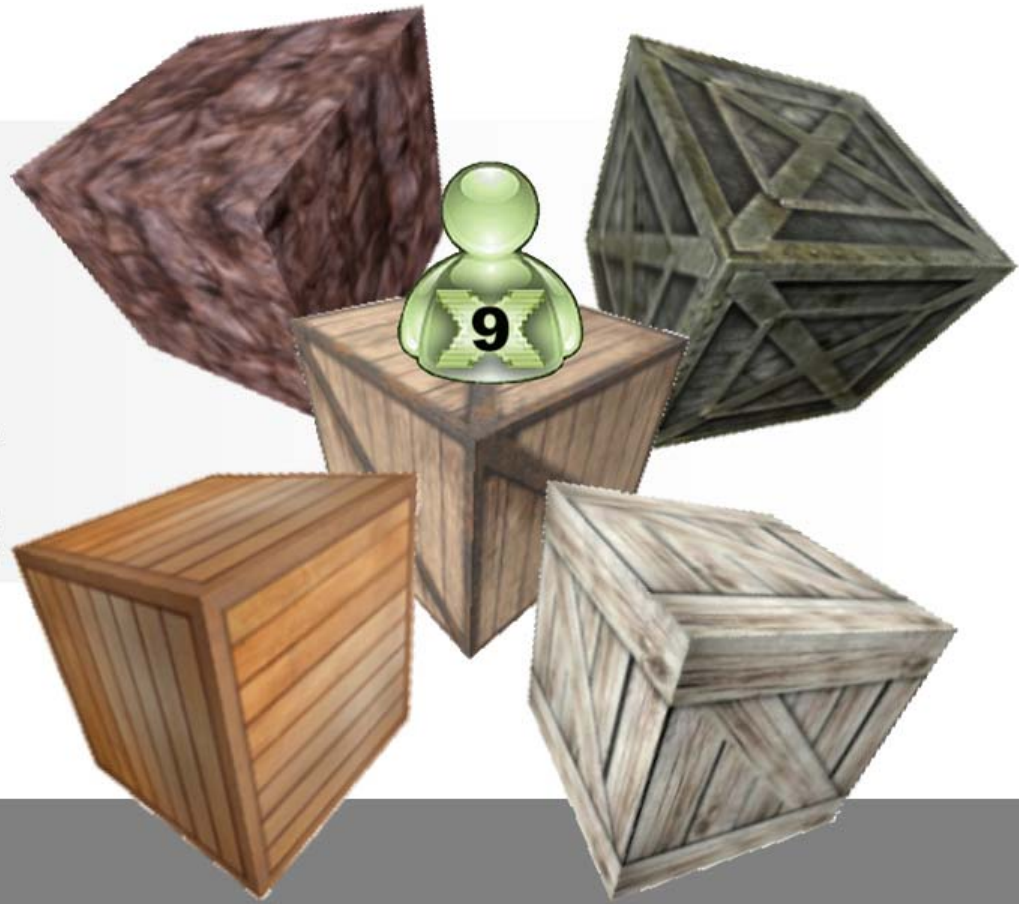
Farkas Gyula Szakkollégium

Bit- és számtologatók

DirectX9

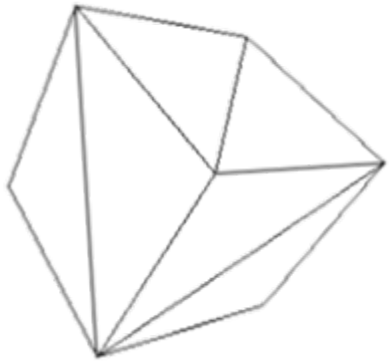
1. **Szín, fény, textúra**
2. **Stencil buffer használata (tükörkép, hamis árnyék)**



2006. május 10., 23.

Róth Ágoston

Vertex vs ColorVertex



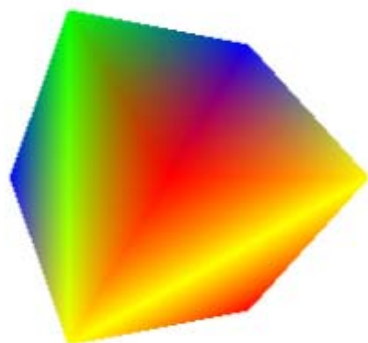
exe

Eddig:

```
struct Vertex
{
  Vertex(){}
  Vertex(float x, float y, float z)
  {
    _x = x;  _y = y;  _z = z;
  }
  float _x, _y, _z;
  static const DWORD FVF;
};

const DWORD Vertex::FVF = D3DFVF_XYZ;
```

Most:

```
struct ColorVertex
{
  ColorVertex(){}
  ColorVertex(float x, float y, float z, D3DCOLOR color)
  {
    _x=x,_y=y,_z=z,_color=color;
  }
  float _x,_y,_z;
  D3DCOLOR _color;
  static const DWORD FVF;
};

const DWORD ColorVertex::FVF=D3DFVF_XYZ | D3DFVF_DIFFUSE;
```



exe

d3dUtility.h – bővítések (1)

```
namespace d3d
{
  ...
  // színek
  const D3DXCOLOR   WHITE( D3DCOLOR_XRGB(255,255,255) );
  const D3DXCOLOR   BLACK( D3DCOLOR_XRGB(  0,  0,  0) );
  const D3DXCOLOR     RED( D3DCOLOR_XRGB(255,  0,  0) );
  const D3DXCOLOR   GREEN( D3DCOLOR_XRGB(  0,255,  0) );
  const D3DXCOLOR    BLUE( D3DCOLOR_XRGB(  0,  0,255) );
  const D3DXCOLOR  YELLOW( D3DCOLOR_XRGB(255,255,  0) );
  const D3DXCOLOR    CYAN( D3DCOLOR_XRGB(  0,255,255) );
  const D3DXCOLOR MAGENTA( D3DCOLOR_XRGB(255,  0,255) );
  ...
}
```

cube.cpp – módosítások a keretrendszer függvényeiben (1)

```
IDirect3DDevice9* Device = 0;
IDirect3DVertexBuffer9* VB = 0;
IDirect3DIndexBuffer9*  IB = 0;

bool Setup()
{
  // vertex buffer létrehozása
  Device->CreateVertexBuffer(
  8 * sizeof(Vertex),
  D3DUSAGE_WRITEONLY,
  Vertex::FVF,
  D3DPOOL_MANAGED,
  &VB,
  0);

  // index buffer létrehozása
  ...

  // a kocka 8 csúcsa
  Vertex* vertices;
  VB->Lock(0, 0, (void**)&vertices, 0);

  vertices[0]=ColorVertex(-1.0f, -1.0f, -1.0f);
  vertices[1]=ColorVertex(-1.0f,  1.0f, -1.0f);
  vertices[2]=ColorVertex( 1.0f,  1.0f, -1.0f);
  vertices[3]=ColorVertex( 1.0f, -1.0f, -1.0f);

  vertices[4]=ColorVertex(-1.0f, -1.0f,  1.0f);
  vertices[5]=ColorVertex(-1.0f,  1.0f,  1.0f);
  vertices[6]=ColorVertex( 1.0f,  1.0f,  1.0f);
  vertices[7]=ColorVertex( 1.0f, -1.0f,  1.0f);

  VB->Unlock();

  // a kocka oldallapjait alkotó
  // háromszögek meghatározása
  ...
}
```

```
IDirect3DDevice9* Device = 0;
IDirect3DVertexBuffer9* VB = 0;
IDirect3DIndexBuffer9*  IB = 0;

bool Setup()
{
  // vertex buffer létrehozása
  Device->CreateVertexBuffer(
  8 * sizeof(ColorVertex),
  D3DUSAGE_WRITEONLY,
  ColorVertex::FVF,
  D3DPOOL_MANAGED,
  &VB,
  0);

  // index buffer létrehozása (ugyanaz)
  ...

  // a kocka 8 csúcsa
  ColorVertex* vertices;
  VB->Lock(0, 0, (void**)&vertices, 0);

  vertices[0]=ColorVertex(-1.0f, -1.0f, -1.0f, d3d::RED);
  vertices[1]=ColorVertex(-1.0f,  1.0f, -1.0f, d3d::GREEN);
  vertices[2]=ColorVertex( 1.0f,  1.0f, -1.0f, d3d::BLUE);
  vertices[3]=ColorVertex( 1.0f, -1.0f, -1.0f, d3d::YELLOW);

  vertices[4]=ColorVertex(-1.0f, -1.0f,  1.0f, d3d::YELLOW);
  vertices[5]=ColorVertex(-1.0f,  1.0f,  1.0f, d3d::BLUE);
  vertices[6]=ColorVertex( 1.0f,  1.0f,  1.0f, d3d::GREEN);
  vertices[7]=ColorVertex( 1.0f, -1.0f,  1.0f, d3d::RED);

  VB->Unlock();

  // a kocka oldallapjait alkotó
  // háromszögek meghatározása (ugyanaz)
  ...
}
```

cube.cpp – módosítások a keretrendszer függvényeiben (2)

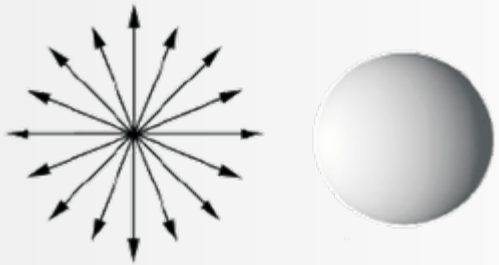
```
void Cleanup()
{
  d3d::Release<IDirect3DVertexBuffer9*>(VB);
  d3d::Release<IDirect3DIndexBuffer9*> (IB);
}
```

```
void Cleanup() //változatlan
{
  d3d::Release<IDirect3DVertexBuffer9*>(VB);
  d3d::Release<IDirect3DIndexBuffer9*> (IB);
}
```

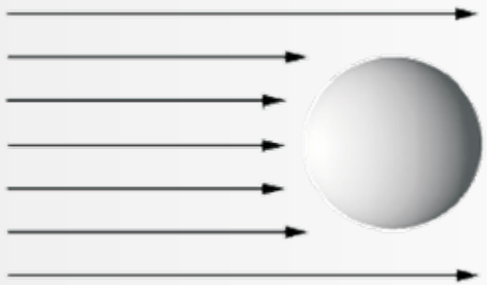
```
bool Display(float timeDelta)
{
  if( Device )
  {
    // transzformációs mátrixok
    ...

    // a színtér
    Device->Clear(
            0, 0,
            D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER,
            0xffffffff, 1.0f, 0);
    Device->BeginScene();
      Device->SetStreamSource(
              0, VB,
              0, sizeof(Vertex));

      Device->SetIndices(IB);
      Device->SetFVF(Vertex::FVF);

      // kocka kirajzolása
      Device->DrawIndexedPrimitive(
              D3DPT_TRIANGLELIST,
              0,
              0, 8,
              0, 12);
    Device->EndScene();
    Device->Present(0, 0, 0, 0);
  }
  return true;
}
```

```
bool Display(float timeDelta)
{
  if( Device )
  {
    // transzformációs mátrixok (ugyanaz)
    ...

    // a színtér
    Device->Clear(
            0, 0,
            D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER,
            0xffffffff, 1.0f, 0);
    Device->BeginScene();
      Device->SetStreamSource(
              0, VB,
              0, sizeof(ColorVertex));

      Device->SetIndices(IB);
      Device->SetFVF(Vertex::FVF);

      // kocka kirajzolása
      Device->DrawIndexedPrimitive(
              D3DPT_TRIANGLELIST,
              0,
              0, 8,
              0, 12);
    Device->EndScene();
    Device->Present(0, 0, 0, 0);
  }
  return true;
}
```

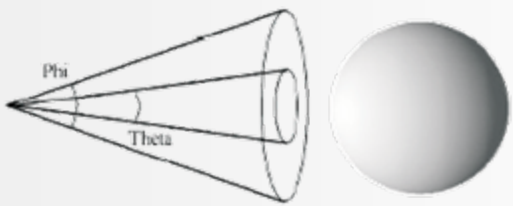
Fényforrások



Pontszerű



Irányított



Reflektor

Fényforrás reprezentálása

```
typedef struct _D3DLIGHT9 {
  D3DLIGHTTYPE Type;        // fényforrás típusa

  D3DCOLORVALUE Diffuse;    // a fényforrás által kibocsátott szórt (diffúz) komponensének a színe
  D3DCOLORVALUE Specular;   // a fényforrás által kibocsátott tükröző (spekuláris) komponensének a színe
  D3DCOLORVALUE Ambient;    // a fényforrás által kibocsátott környezeti (ambiens) komponensének a színe

  D3DVECTOR Position;       // a fényforrás helyzetvektora

  D3DVECTOR Direction;      // a fényforrás irányítása (feltéve, ha irányított típusú)

  float Range;              // az a maximális távolság melyet a fény bejárhat, mielőtt elhalna (<=sqrt(FLT_MAX))

  float Falloff;            // csak reflektorszerű fényforrás esetén használatos
                            // azt mutatja meg, hogyan csökken a fény intenzitása (erőssége)
                            // a belső kúptól a külső kúpig (értéke általában: 1.0)

  float Attenuation0;       // a következő három paraméter azt határozza meg, hogyan csökken a fény erőssége
  float Attenuation1;       // a távolsággal; csak pont- és reflektorszerű fényforrások esetén használatosak
  float Attenuation2;

  float Theta;              // reflektorszerű fényforrás esetén a belső kúp nyilásszögét szabja meg (rad)
  float Phi;                // reflektorszerű fényforrás esetén a külső kúp nyilásszögét határozza meg (rad)

} D3DLIGHT9;
```

d3dUtility.h/d3dUtility.cpp – bővítések (2)

d3dUtility.h

```
namespace d3d
{
...

  // fényforrások

  // irányított
  D3DLIGHT9 InitDirectionalLight(D3DXVECTOR3* direction, D3DXCOLOR* color);

  // pontszerű
  D3DLIGHT9 InitPointLight(D3DXVECTOR3* position, D3DXCOLOR*  color);

  // reflektorszerű
  D3DLIGHT9 InitSpotLight(D3DXVECTOR3* position, D3DXVECTOR3* direction, D3DXCOLOR* color);
  ...
}
```

d3dUtility.cpp

```
D3DLIGHT9 d3d::InitDirectionalLight(D3DXVECTOR3* direction, D3DXCOLOR*  color)
{
  D3DLIGHT9 result;
  ::ZeroMemory(&result,sizeof(D3DLIGHT9));

  result.Type=D3DLIGHT_DIRECTIONAL;
  result.Ambient=*color*0.4f;
  result.Diffuse=*color;
  result.Specular=*color*0.6f;
  result.Direction=*direction;

  return result;
}
```

d3dUtility.h/d3dUtility.cpp – bővítések (3)

d3dUtility.cpp

```
D3DLIGHT9 d3d::InitPointLight(D3DXVECTOR3* position, D3DXCOLOR*  color)
{
  D3DLIGHT9 result;
  ::ZeroMemory(&result,sizeof(D3DLIGHT9));

  result.Type           =D3DLIGHT_POINT;
  result.Ambient        =*color * 0.6f;
  result.Diffuse        =*color;
  result.Specular       =*color * 0.6f;
  result.Position       =*position;
  result.Range          =1000.0f;
  result.Falloff        =1.0f;
  result.Attenuation0   =1.0f;
  result.Attenuation1   =0.0f;
  result.Attenuation2   =0.0f;
  return result;
}

D3DLIGHT9 d3d::InitSpotLight(D3DXVECTOR3* position, D3DXVECTOR3* direction, D3DXCOLOR* color)
{
  D3DLIGHT9 result;
  ::ZeroMemory(&result,sizeof(D3DLIGHT9));

  result.Type           = D3DLIGHT_SPOT;
  result.Ambient        = *color * 0.0f;
  result.Diffuse        = *color;
  result.Specular       = *color * 0.6f;
  result.Position       = *position;
  result.Direction      = *direction;
  result.Range          = 1000.0f;
  result.Falloff        = 1.0f;
  result.Attenuation0   = 1.0f;
  result.Attenuation1   = 0.0f;
  result.Attenuation2   = 0.0f;
  result.Theta          = 0.4f;
  result.Phi            = 0.9f;
  return result;
}
```

Felületek anyagi jellemzőinek reprezentálása (material)

```
typedef struct _D3DMATERIAL9 {
  D3DCOLORVALUE Diffuse,  // a felület által visszatükrözött diffúz fény mennyisége
                Ambient,  // a felület által visszatükrözött ambiens fény mennyisége
                Specular, // a felület által visszatükrözött spekuláris fény mennyisége
                Emissive; // egy olyan komponens, mely az előbbi háromhoz adódik hozzá,
                          // a felület világosabb lesz általa, mintha saját maga bocsátaná ki a saját fényét

  float Power;            // a spekuláris fénysávok élességét határozza meg (minél nagyobb, annál élesebb)
} D3DMATERIAL9;
```

```
// példa: piros felületű tárgy
// csak a piros fényt tükrözi vissza, minden más színűt elnyel

D3DMATERIAL9 red;
::ZeroMemory(&red, sizeof(red));
red.Diffuse  =          D3DXCOLOR(1.0f, 0.0f, 0.0f, 1.0f); // piros szórt fényt tükröz vissza
red.Ambient  =          D3DXCOLOR(1.0f, 0.0f, 0.0f, 1.0f); // piros ambiens fényt tükröz vissza
red.Specular =          D3DXCOLOR(1.0f, 0.0f, 0.0f, 1.0f); // piros spekuláris fényt tükröz vissza
red.Emissive =          D3DXCOLOR(0.0f, 0.0f, 0.0f, 1.0f); // nem bocsát ki fényt
red.Power = 5.0f;

...

Device->SetMaterial(&red); // material beállítása
```

d3dUtility.h/d3dUtility.cpp – bővítések (4)

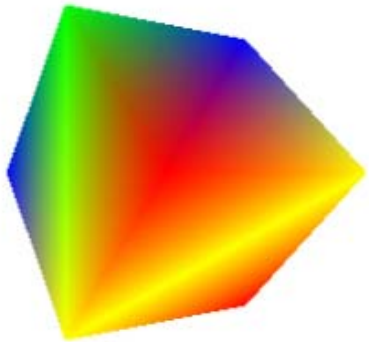
d3dUtility.h

```
namespace d3d
{
  ...
  D3DMATERIAL9 InitMaterial(
               D3DXCOLOR ambient,
               D3DXCOLOR diffuse,
               D3DXCOLOR specular,
               D3DXCOLOR emissive,
               float power);

  const D3DMATERIAL9 WhiteMaterial   = InitMaterial(WHITE,WHITE,WHITE,BLACK,8.0f);
  const D3DMATERIAL9 RedMaterial     = InitMaterial(RED,RED,RED,BLACK,8.0f);
  const D3DMATERIAL9 GreenMaterial   = InitMaterial(GREEN,GREEN,GREEN,BLACK,8.0f);
  const D3DMATERIAL9 BlueMaterial    = InitMaterial(BLUE,BLUE,BLUE,BLACK,8.0f);
  const D3DMATERIAL9 YellowMaterial  = InitMaterial(YELLOW,YELLOW,YELLOW,BLACK,8.0f);
  ...
}
```

d3dUtility.cpp

```
D3DMATERIAL9 d3d::InitMaterial(
                  D3DXCOLOR ambient,
                  D3DXCOLOR diffuse,
                  D3DXCOLOR specular,
                  D3DXCOLOR emissive,
                  float power)
{
            D3DMATERIAL9 result;

            result.Ambient=ambient;
            result.Diffuse=diffuse;
            result.Specular=specular;
            result.Emissive=emissive;
            result.Power=power;

            return result;
}
```

ColorVertex vs (NormalVertex & Material)



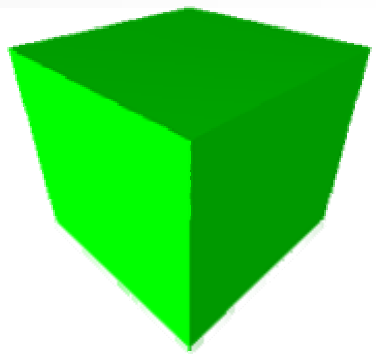
exe

Eddig:

```
struct ColorVertex
{
  ColorVertex(){}
  ColorVertex(
    float x, float y, float z,
    D3DCOLOR color)
  {
    _x=x,_y=y,_z=z,_color=color;
  }
  float _x,_y,_z;
  D3DCOLOR _color;
  static const DWORD FVF;
};


const DWORD
ColorVertex::FVF=D3DFVF_XYZ | D3DFVF_DIFFUSE;
```

Most:

```
struct NormalVertex
{
  NormalVertex(){}
  NormalVertex(
    float x,  float y,  float z,
    float nx, float ny, float nz)
  {
    _x  = x;  _y  = y;  _z  = z;
   _nx = nx; _ny = ny; _nz = nz;
  }
  float  _x,  _y,  _z;
  float _nx, _ny, _nz;
  static const DWORD FVF;
};

const DWORD
NormalVertex::FVF = D3DFVF_XYZ | D3DFVF_NORMAL;
```



exe

(NormalVertex & Material) vs (NormalTexVertex & Material)

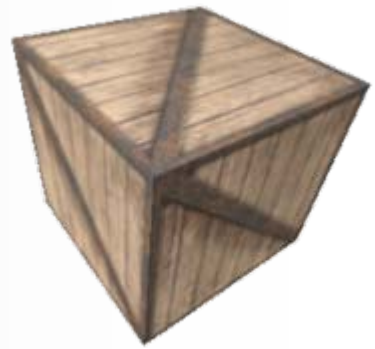
exe

Eddig:

```
struct NormalVertex
{
  NormalVertex(){}
  NormalVertex(
    float x,  float y,  float z,
    float nx, float ny, float nz)
  {
    _x  = x;  _y  = y;  _z  = z;
   _nx = nx; _ny = ny; _nz = nz;
  }
  float  _x,  _y,  _z;
  float _nx, _ny, _nz;
  static const DWORD FVF;
};

const DWORD
NormalVertex::FVF = D3DFVF_XYZ | D3DFVF_NORMAL;
```

Most:

```
struct NormalTexVertex
{
  NormalTexVertex(){}
  NormalTexVertex(
    float x,  float y,   float z,
    float nx, float ny,  float nz,
    float u, float v)
  {
    _x=x,_y=y,_z=z;
    _nx=nx,_ny=ny,_nz=nz;
    _u=u,_v=v;
  }
  float _x, _y, _z;
  float _nx, _ny, _nz;
  float _u, _v;
  static const DWORD FVF;
};

const DWORD
NormalTexVertex=D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_TEX1;
```



exe

Új: vertex.h

vertex.h

```
#ifndef __vertexH__
#define __vertexH__

#include <d3dx9.h>

struct NormalTexVertex
{
  NormalTexVertex(){}
  NormalTexVertex(
    float x,  float y,  float z,
    float nx, float ny, float nz,
    float u,  float v)
  {
    _x=x, _y=y, _z=z;
    _nx=nx, _ny=ny, _nz=nz;
    _u=u, _v=v;
  }
  float _x,  _y,  _z;
  float _nx, _ny, _nz;
  float _u,  _v;
};

#define FVF_VERTEX (D3DFVF_XYZ | D3DFVF_NORMAL | D3DFVF_TEX1)

#endif  __vertexH__
```

Új: cube.h/cube.cpp

cube.h

```
#ifndef __cubeH__
#define __cubeH__

#include <d3dx9.h>

class Cube
{
public:
  Cube(IDirect3DDevice9* device);
  ~Cube();
  bool Draw(D3DXMATRIX* world, D3DMATERIAL9* mtrl, IDirect3DTexture9* tex);

private:
  IDirect3DDevice9*       _device;
  IDirect3DVertexBuffer9* _vb;
  IDirect3DIndexBuffer9*  _ib;
};

#endif  __cubeH__
```

cube.cpp (1)

```
#include "cube.h"
#include "vertex.h"

Cube::Cube(IDirect3DDevice9 *device) //konstruktor
{
  _device=device;
  _device->CreateVertexBuffer(
            24*sizeof(Vertex),
            D3DUSAGE_WRITEONLY,
            FVF_VERTEX,
            D3DPOOL_MANAGED,
            &_vb,
            0);
```

cube.cpp (folytatás)

cube.cpp (2)

```
_device->CreateIndexBuffer(
          36*sizeof(WORD),
          D3DUSAGE_WRITEONLY,
          D3DFMT_INDEX16,
          D3DPOOL_MANAGED,
          &_ib,
          0);

Vertex *v;
_vb->Lock(0,0,(void**)&v,0);

// kocka csúcsai a megfelelő normálisokkal és textúra-koordinátákkal

// elől
v[0] = NormalTexVertex(-1.0f, -1.0f, -1.0f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f);
v[1] = NormalTex Vertex(-1.0f,  1.0f, -1.0f, 0.0f, 0.0f, -1.0f, 0.0f, 1.0f);
v[2] = NormalTex Vertex( 1.0f,  1.0f, -1.0f, 0.0f, 0.0f, -1.0f, 1.0f, 1.0f);
v[3] = NormalTex Vertex( 1.0f, -1.0f, -1.0f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f);

// hátul
v[4] = NormalTex Vertex(-1.0f, -1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f, 0.0f);
v[5] = NormalTex Vertex( 1.0f, -1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f, 1.0f);
v[6] = NormalTex Vertex( 1.0f,  1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f, 1.0f);
v[7] = NormalTex Vertex(-1.0f,  1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f, 0.0f);

// felül
v[8]  = NormalTex Vertex(-1.0f, 1.0f, -1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f);
v[9]  = NormalTex Vertex(-1.0f, 1.0f,  1.0f, 0.0f, 1.0f, 0.0f, 0.0f, 1.0f);
v[10] = NormalTex Vertex( 1.0f, 1.0f,  1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 1.0f);
v[11] = NormalTex Vertex( 1.0f, 1.0f, -1.0f, 0.0f, 1.0f, 0.0f, 1.0f, 0.0f);

// alul
v[12] = NormalTex Vertex(-1.0f, -1.0f, -1.0f, 0.0f, -1.0f, 0.0f, 0.0f, 0.0f);
v[13] = NormalTex Vertex( 1.0f, -1.0f, -1.0f, 0.0f, -1.0f, 0.0f, 0.0f, 1.0f);
v[14] = NormalTex Vertex( 1.0f, -1.0f,  1.0f, 0.0f, -1.0f, 0.0f, 1.0f, 1.0f);
v[15] = NormalTex Vertex(-1.0f, -1.0f,  1.0f, 0.0f, -1.0f, 0.0f, 1.0f, 0.0f);
```

cube.cpp (folytatás)

cube.cpp (3)

```
  // bal
  v[16] = NormalTex Vertex(-1.0f, -1.0f,  1.0f, -1.0f, 0.0f, 0.0f, 0.0f, 0.0f);
  v[17] = NormalTex Vertex(-1.0f,  1.0f,  1.0f, -1.0f, 0.0f, 0.0f, 0.0f, 1.0f);
  v[18] = NormalTex Vertex(-1.0f,  1.0f, -1.0f, -1.0f, 0.0f, 0.0f, 1.0f, 1.0f);
  v[19] = NormalTex Vertex(-1.0f, -1.0f, -1.0f, -1.0f, 0.0f, 0.0f, 1.0f, 0.0f);

  // jobb
  v[20] = NormalTex Vertex( 1.0f, -1.0f, -1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 0.0f);
  v[21] = NormalTex Vertex( 1.0f,  1.0f, -1.0f, 1.0f, 0.0f, 0.0f, 0.0f, 1.0f);
  v[22] = NormalTex Vertex( 1.0f,  1.0f,  1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 1.0f);
  v[23] = NormalTex Vertex( 1.0f, -1.0f,  1.0f, 1.0f, 0.0f, 0.0f, 1.0f, 0.0f);

  _vb->Unlock();

  WORD *i;
  _ib->Lock(0,0,(void**)&i,0);
  // elől
  i[0] = 0; i[1] = 1; i[2] = 2;
  i[3] = 0; i[4] = 2; i[5] = 3;
  // hátul
  i[6] = 4; i[7]  = 5; i[8]  = 6;
  i[9] = 4; i[10] = 6; i[11] = 7;
  // felül
  i[12] = 8; i[13] =  9; i[14] = 10;
  i[15] = 8; i[16] = 10; i[17] = 11;
  // alul
  i[18] = 12; i[19] = 13; i[20] = 14;
  i[21] = 12; i[22] = 14; i[23] = 15;
  // bal
  i[24] = 16; i[25] = 17; i[26] = 18;
  i[27] = 16; i[28] = 18; i[29] = 19;
  // jobb
  i[30] = 20; i[31] = 21; i[32] = 22;
  i[33] = 20; i[34] = 22; i[35] = 23;
  _ib->Unlock();
}
```

cube.cpp (folytatás)

cube.cpp (4)

```
Cube::~Cube() // destruktor
{
  if (_vb) {_vb->Release();_vb=0;}
  if (_ib) {_ib->Release();_ib=0;}
}

bool Cube::Draw(D3DXMATRIX* world, D3DMATERIAL9* mtrl,IDirect3DTexture9 *tex)
{
  if (world)
    _device->SetTransform(D3DTS_WORLD,world);
  if (mtrl)
    _device->SetMaterial(mtrl);
  if (tex)
    _device->SetTexture(0,tex);
  _device->SetStreamSource(0,_vb,0,sizeof(NormalTexVertex));
  _device->SetIndices(_ib);
  _device->SetFVF(FVF_VERTEX);
  _device->DrawIndexedPrimitive(D3DPT_TRIANGLELIST,0,0,24,0,12);
  return true;
}
```

main.cpp

```
Cube *Box;
IDirect3DDevice9*    Device = 0;
IDirect3DTexture9*   Tex = 0;

bool Setup()
{
            ...
            Box = new Cube(Device);

            // irányított fény
            D3DLIGHT9 light;
            ::ZeroMemory(&light, sizeof(light));
            light.Type      = D3DLIGHT_DIRECTIONAL;
            light.Ambient   = D3DXCOLOR(0.8f, 0.8f, 0.8f, 1.0f);
            light.Diffuse   = D3DXCOLOR(1.0f, 1.0f, 1.0f, 1.0f);
            light.Specular  = D3DXCOLOR(0.2f, 0.2f, 0.2f, 1.0f);
            light.Direction = D3DXVECTOR3(1.0f, -1.0f, 0.0f);

            // fény engedélyezése
            Device->SetLight(0, &light);
            Device->LightEnable(0, true);

            Device->SetRenderState(D3DRS_NORMALIZENORMALS, true);
            Device->SetRenderState(D3DRS_SPECULARENABLE, true);

            // textúra beolvasása
            D3DXCreateTextureFromFile(Device, "lada.jpg", &Tex);

            // textúra filter állapotok
            Device->SetSamplerState(0, D3DSAMP_MAGFILTER, D3DTEXF_LINEAR);
            Device->SetSamplerState(0, D3DSAMP_MINFILTER, D3DTEXF_LINEAR);
            Device->SetSamplerState(0, D3DSAMP_MIPFILTER, D3DTEXF_LINEAR);
            ...

}
```

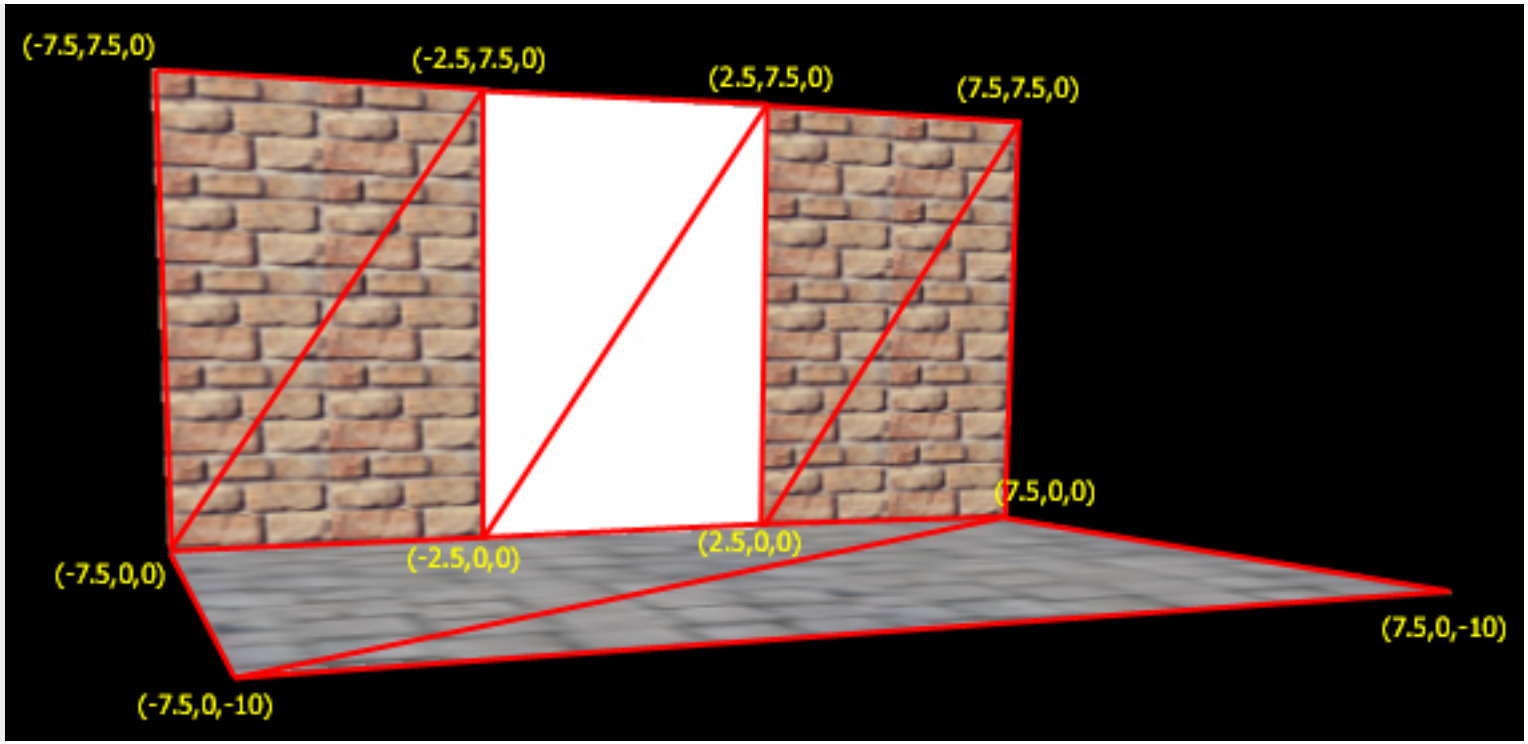
main.cpp (folytatás)

```
void Cleanup()
{
            d3d::Delete<Cube*>(Box);
            d3d::Release<IDirect3DTexture9*>(Tex);
}


bool Display(float timeDelta)
{
            if( Device )
            {
                        ...
                        Device->SetTransform(D3DTS_WORLD,&World);

                        Device->BeginScene();

                                    D3DMATERIAL9 mtrl=d3d::WHITE_MTRL;
                                    Box->Draw(&World, &mtrl, Tex);

                        Device->EndScene();
                        Device->Present(0, 0, 0, 0);
            }
            return true;
}
```

Stencil-bufferek használata – Síktükör

exe

Síktükör – A színtér geometriája



fal.jpg

macskako.jpg

tukor.jpg

lada.jpg

main.cpp – globális változók, segéd eljárások

```
#include "d3dUtility.h"
#include "vertex.h"
#include "cube.h"


// globális változók

IDirect3DDevice9* Device = 0;

IDirect3DVertexBuffer9* VB = 0; // a színtér geometriáját tartalmazó vertex-buffer

IDirect3DTexture9* FloorTex  = 0;
IDirect3DTexture9* BoxTex    = 0;
IDirect3DTexture9* WallTex   = 0;
IDirect3DTexture9* MirrorTex = 0;

D3DMATERIAL9 BoxMtrl    = d3d::WHITE_MTRL;
D3DMATERIAL9 FloorMtrl  = d3d::WHITE_MTRL;
D3DMATERIAL9 WallMtrl   = d3d::WHITE_MTRL;
D3DMATERIAL9 MirrorMtrl = d3d::WHITE_MTRL;

Cube *Box=0;
D3DXVECTOR3 BoxPosition(0.0f, 1.0f, -7.5f);

// segéd eljárások
void RenderScene();  // az egyszerű színteret kirajzoló függvény
void RenderMirror(); // a tükörképet kirajzoló függvény (stencil-buffer)
```

main.cpp – keretrendszer-függvények

```
bool Setup()
{

  // a láda és a falak a spekuláris fény kis százalékát tükrözik vissza
  WallMtrl.Specular = d3d::WHITE * 0.2f;
  BoxMtrl.Specular  = d3d::WHITE * 0.01f;

  // a láda létrehozása
  Box=new Cube(Device);

  Device->CreateVertexBuffer(24 * sizeof(NormalTexVertex),0, FVF_NormalTexVertex, D3DPOOL_MANAGED, &VB, 0);

  NormalTexVertex* v = 0;
  VB->Lock(0, 0, (void**)&v, 0);

  // padló
  v[0] = NormalTexVertex(-7.5f, 0.0f, -10.0f, 0.0f, 1.0f, 0.0f, 0.0f, 2.0f);
  v[1] = NormalTexVertex(-7.5f, 0.0f,   0.0f, 0.0f, 1.0f, 0.0f, 0.0f, 0.0f);
  v[2] = NormalTexVertex( 7.5f, 0.0f,   0.0f, 0.0f, 1.0f, 0.0f, 2.0f, 0.0f);
  v[3] = NormalTexVertex(-7.5f, 0.0f, -10.0f, 0.0f, 1.0f, 0.0f, 0.0f, 2.0f);
  v[4] = NormalTexVertex( 7.5f, 0.0f,   0.0f, 0.0f, 1.0f, 0.0f, 2.0f, 0.0f);
  v[5] = NormalTexVertex( 7.5f, 0.0f, -10.0f, 0.0f, 1.0f, 0.0f, 2.0f, 2.0f);

  // fal
  v[6]  = NormalTexVertex(-7.5f, 0.0f, 0.0f, 0.0f, 0.0f, -1.0f, 0.0f, 5.0f);
  v[7]  = NormalTexVertex(-7.5f, 7.5f, 0.0f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f);
  v[8]  = NormalTexVertex(-2.5f, 7.5f, 0.0f, 0.0f, 0.0f, -1.0f, 2.0f, 0.0f);
  v[9]  = NormalTexVertex(-7.5f, 0.0f, 0.0f, 0.0f, 0.0f, -1.0f, 0.0f, 5.0f);
  v[10] = NormalTexVertex(-2.5f, 7.5f, 0.0f, 0.0f, 0.0f, -1.0f, 2.0f, 0.0f);
  v[11] = NormalTexVertex(-2.5f, 0.0f, 0.0f, 0.0f, 0.0f, -1.0f, 2.0f, 5.0f);
  v[12] = NormalTexVertex(2.5f, 0.0f, 0.0f, 0.0f, 0.0f, -1.0f, 0.0f, 5.0f);
  v[13] = NormalTexVertex(2.5f, 7.5f, 0.0f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f);
  v[14] = NormalTexVertex(7.5f, 7.5f, 0.0f, 0.0f, 0.0f, -1.0f, 2.0f, 0.0f);
  v[15] = NormalTexVertex(2.5f, 0.0f, 0.0f, 0.0f, 0.0f, -1.0f, 0.0f, 5.0f);
  v[16] = NormalTexVertex(7.5f, 7.5f, 0.0f, 0.0f, 0.0f, -1.0f, 2.0f, 0.0f);
  v[17] = NormalTexVertex(7.5f, 0.0f, 0.0f, 0.0f, 0.0f, -1.0f, 2.0f, 5.0f);
```

main.cpp – keretrendszer-függvények

```
  // tükör
  v[18] = NormalTexVertex(-2.5f, 0.0f, 0.0f, 0.0f, 0.0f, -1.0f, 0.0f, 3.0f);
  v[19] = NormalTexVertex(-2.5f, 7.5f, 0.0f, 0.0f, 0.0f, -1.0f, 0.0f, 0.0f);
  v[20] = NormalTexVertex( 2.5f, 7.5f, 0.0f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f);

  v[21] = NormalTexVertex(-2.5f, 0.0f, 0.0f, 0.0f, 0.0f, -1.0f, 0.0f, 3.0f);
  v[22] = NormalTexVertex( 2.5f, 7.5f, 0.0f, 0.0f, 0.0f, -1.0f, 1.0f, 0.0f);
  v[23] = NormalTexVertex( 2.5f, 0.0f, 0.0f, 0.0f, 0.0f, -1.0f, 1.0f, 3.0f);

  VB->Unlock();

  // textúrák beolvasása

  D3DXCreateTextureFromFile(Device, "macskako.jpg", &FloorTex);
  D3DXCreateTextureFromFile(Device, "fal.jpg",      &WallTex);
  D3DXCreateTextureFromFile(Device, "tukor.jpg",    &MirrorTex);
  D3DXCreateTextureFromFile(Device, "lada.jpg",     &BoxTex);

  ... // fény, kamera, vetítési mátrix, stb.

} //Setup


void Cleanup()
{
            d3d::Release<IDirect3DVertexBuffer9*>(VB);
            d3d::Release<IDirect3DTexture9*>(FloorTex);
            d3d::Release<IDirect3DTexture9*>(WallTex);
            d3d::Release<IDirect3DTexture9*>(MirrorTex);
            d3d::Release<IDirect3DTexture9*>(BoxTex);
            delete Box;Box=0;
}
```

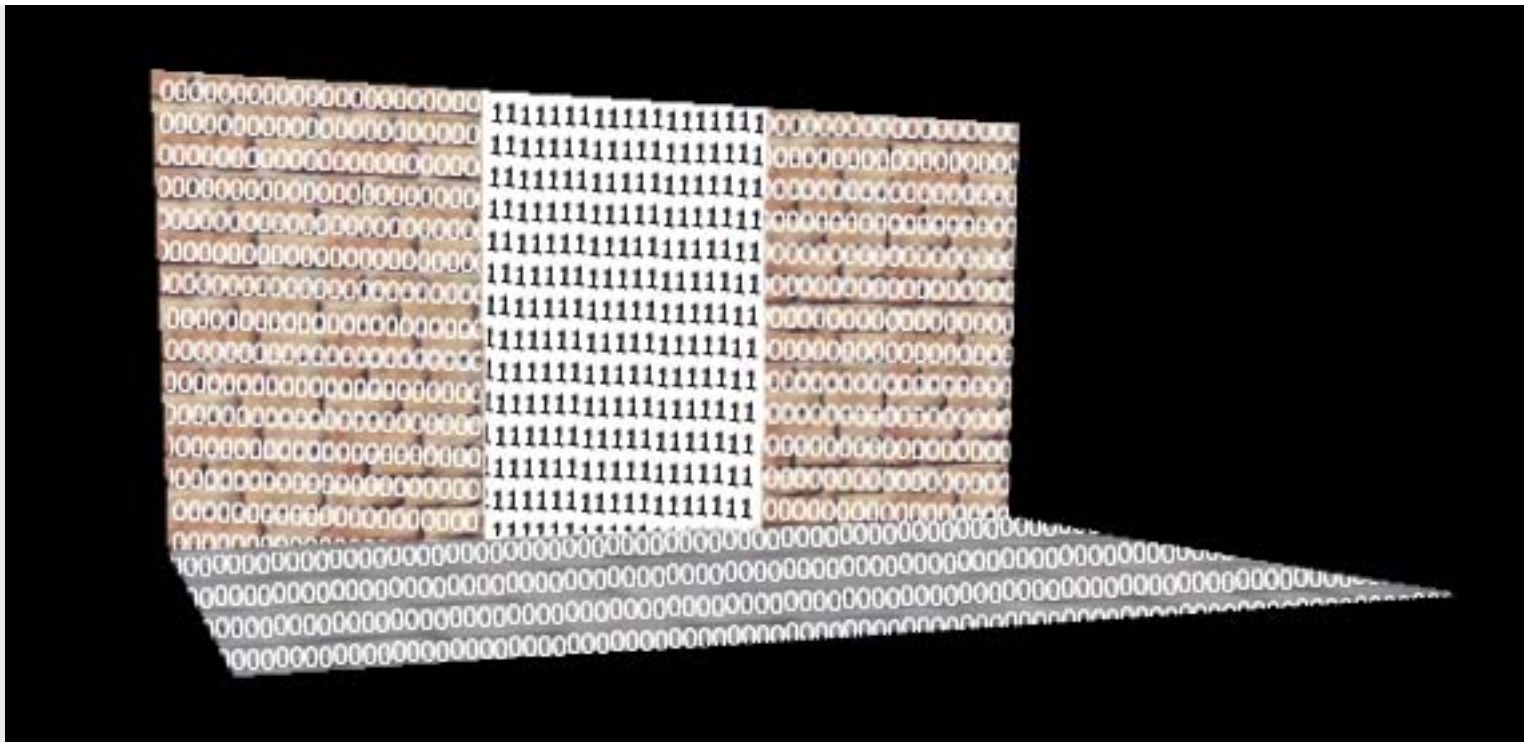
main.cpp – keretrendszer-függvények

```
bool Display(float timeDelta)
{
  if( Device )
  {
    ... // események kezelése

    Device->Clear(0, 0, D3DCLEAR_TARGET | D3DCLEAR_ZBUFFER | D3DCLEAR_STENCIL,0xff000000, 1.0f, 0L);
    Device->BeginScene();
      RenderScene();
      RenderMirror();
    Device->EndScene();
    Device->Present(0, 0, 0, 0);
   }
   return true;
}
```

main.cpp – segéd eljárások: void RenderScene();

```
void RenderScene()
{
  // láda kirajzolása
  D3DXMATRIX W; // a láda világ-mátixa
  D3DXMatrixTranslation(&W,BoxPosition.x, BoxPosition.y, BoxPosition.z);
  Box->Draw(&W,0,BoxTex);

  D3DXMATRIX I;
  D3DXMatrixIdentity(&I); // egységmátrix
  Device->SetTransform(D3DTS_WORLD, &I);

  Device->SetStreamSource(0, VB, 0, sizeof(NormalTexVertex));
  Device->SetFVF(FVF_NormalTexVertex);

  // padló
  Device->SetMaterial(&FloorMtrl);
  Device->SetTexture(0, FloorTex);
  Device->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 2);

  // fal
  Device->SetMaterial(&WallMtrl);
  Device->SetTexture(0, WallTex);
  Device->DrawPrimitive(D3DPT_TRIANGLELIST, 6, 4);

  // tükör
  Device->SetMaterial(&MirrorMtrl);
  Device->SetTexture(0, MirrorTex);
  Device->DrawPrimitive(D3DPT_TRIANGLELIST, 18, 2);
}
```

main.cpp – segéd eljárások: void RenderMirror(); [1]



main.cpp – segéd eljárások: void RenderMirror(); [2]

```
void RenderMirror()
{
  // csak a tükört alkotó háromszögeket rajzoljuk ki a stencil-bufferbe;
  // így a stencil bufferben a tükör pixeleinek megfelelő bitek állítódnak be;
  // később a tükrözött kockának csak azon pixelei rajzolódnak ki,
  // melyekhez tartozó stencil-bitek be voltak állítva

  Device->SetRenderState(D3DRS_STENCILENABLE,    true);
  Device->SetRenderState(D3DRS_STENCILFUNC,      D3DCMP_ALWAYS);
  Device->SetRenderState(D3DRS_STENCILREF,       0x1);
  Device->SetRenderState(D3DRS_STENCILMASK,      0xffffffff);
  Device->SetRenderState(D3DRS_STENCILWRITEMASK, 0xffffffff);
  Device->SetRenderState(D3DRS_STENCILZFAIL,     D3DSTENCILOP_KEEP);
  Device->SetRenderState(D3DRS_STENCILFAIL,      D3DSTENCILOP_KEEP);
  Device->SetRenderState(D3DRS_STENCILPASS,      D3DSTENCILOP_REPLACE);

  // letiltjuk a mélység- és háttér-bufferekbe való írást
  Device->SetRenderState(D3DRS_ZWRITEENABLE, false);
  Device->SetRenderState(D3DRS_ALPHABLENDENABLE, true);
  Device->SetRenderState(D3DRS_SRCBLEND,  D3DBLEND_ZERO);
  Device->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_ONE);

  // csak a tükört alkotó háromszögeket rajzoljuk ki
  Device->SetStreamSource(0, VB, 0, sizeof(NormalTexVertex));
  Device->SetFVF(FVF_NormalTexVertex);
  Device->SetMaterial(&MirrorMtrl);
  Device->SetTexture(0, MirrorTex);
  D3DXMATRIX I;
  D3DXMatrixIdentity(&I);
  Device->SetTransform(D3DTS_WORLD, &I);
  Device->DrawPrimitive(D3DPT_TRIANGLELIST, 18, 2);

  // mélység-buffer használatát újból engedélyezzük
  Device->SetRenderState( D3DRS_ZWRITEENABLE, true );
```

main.cpp – segéd eljárások: void RenderMirror(); [3]

```
  // a tükrözött kockának csak azon pixeleit jelenítjük meg
  // ahová a tükör volt kirajzolva
  Device->SetRenderState(D3DRS_STENCILFUNC,  D3DCMP_EQUAL);
  Device->SetRenderState(D3DRS_STENCILPASS,  D3DSTENCILOP_KEEP);

  D3DXMATRIX W,    // világ-mátrix
             T,    // transzlációs mátrix
             R;    // tükrözési mátrix

  D3DXPLANE plane(0.0f, 0.0f, 1.0f, 0.0f); // xOy síkra merőlegesen tükrözünk
  D3DXMatrixReflect(&R, &plane);
  D3DXMatrixTranslation(&T,BoxPosition.x, BoxPosition.y, BoxPosition.z);
  W = T * R;

  // mélység-buffer törlése és a tükrözött kocka valamint a tükör pixeleinek "összemosása"
  Device->Clear(0, 0, D3DCLEAR_ZBUFFER, 0, 1.0f, 0);
  Device->SetRenderState(D3DRS_SRCBLEND,  D3DBLEND_DESTCOLOR);
  Device->SetRenderState(D3DRS_DESTBLEND, D3DBLEND_ZERO);

  // tükrözés során a látható lapok irányítása megváltozik
  Device->SetRenderState(D3DRS_CULLMODE, D3DCULL_CW);

  // a láda tükörlépe
  Device->SetTransform(D3DTS_WORLD, &W);
  Box->Draw(&W,0,BoxTex);

  // a padló tükörképe
  Device->SetStreamSource(0, VB, 0, sizeof(NormalTexVertex));
  Device->SetFVF(FVF_NormalTexVertex);
  Device->SetTransform(D3DTS_WORLD, &R);
  Device->SetMaterial(&FloorMtrl);
  Device->SetTexture(0, FloorTex);
  Device->DrawPrimitive(D3DPT_TRIANGLELIST, 0, 2);

  // renderelő állapotok visszaállítása
  Device->SetRenderState(D3DRS_ALPHABLENDENABLE, false);
  Device->SetRenderState( D3DRS_STENCILENABLE, false);
  Device->SetRenderState(D3DRS_CULLMODE, D3DCULL_CCW);

} // RenderMirror
```