Direct3D interface

Grafikus játékok fejlesztése

Szécsi László

2011.02.23. t04-pipecontrol

A pipeline vezérlése

- erőforrások allokálása
  - vertex buffer, index buffer, textúrák
- rajzolási állapot beállítása
  - ..., culling, blending, ...
- shader programok, uniform változók
  - HLSL forrás alapján
- működés indítása
  - draw call

Direct3D API - device

- eszköz [device]
  - a grafikus kártya memóriájának absztrakciója
  - erőforrások (és más előregyártott objektumok, állapotstruktúrák, shader programok) kezelésére szolgáló felület
  - ID3D11Device interface

```
ID3D11Device* device;
```

Direct3D API - context

- kontextus [context]
  - a pipelineelemek állapotának absztrakciója
  - rajzolási állapot beállítására, rajzolásra szolgáló felület
  - alapvetően egy van belőle egy devicehoz
    - immediate context
    - deferred context többszálú működéshez jó
  - ID3D11DeviceContext interface

ID3D11DeviceContext* context;

DXUT

- DirectX utility toolkit
  - olyan mint az OpenGLhez a GLUT
  - oprendszer funkciók elrejtése (ablaknyitás pl.)
  - sokkal többet tud
  - ezért bonyolultabb
- de a magja ugyanaz
  - általunk írt callback függvények regisztrálása
  - vezérlés átadása
  - eseményekkor meghívja a mi callback függvényeinket

Eszköz-események

- létrejött [CreateDevice]
  - program indulásakor
- új ablak [ResizedSwapChain]
  - induláskor, új oprendszer-összerendeléskor (pl. átméretezés után)
- régi ablak eltűnik [ReleasingSwapChain]
  - hiba esetén, az oprendszer-felület változásakor (pl. átméretezéskor)
- megszűnt [DestroyDevice]

További fontos események

- OnFrameMove
  - paraméterként kapja: a dt időlépcsőt
  - virtuális világ frissítése
- OnFrameRender
  - paraméterként kapja: a device contextet
  - frame bufferbe rajzolás
- MsgProc
  - bármilyen windows üzenet
  - billentyű le/fel, egérklikk, egérmozgatás, …

OO eseménykezelés laboron

- Egg::App osztály metódusait hívják a callbackek
  - CreateDevice → new, createResources
  - DestroyDevice → releaseResources, delete
  - ResizedSwapChain → setSwapChain, createSwapChainResources
  - ReleasingSwapChain → releaseSwapChainResources
  - FrameMove → animate
  - FrameRender → render
  - MsgProc → processMessage

Erőforrástípusok

- ID3D11Buffer
  - vertex, index, instance, constant, shader resource
- ID3D11Texture1D, ID3D11Texture2D, ID3D11Texture3D
  - valahány elemű tömbje
  - a valahány mipmap-szinttel rendelkező
  - valamilyen formátumú pixelek tömbjeinek
- hatelemű textúratömb = cube map

Erőforrás-kezelési módok (usage)

- immutable
  - létrehozáskor inicializálható, utána csak olvasható
- dynamic
  - rátölthetünk új adatot akár minden frameben
- default
  - nem tölthetünk rá, de szerepelhet a pipelineban mint kimenet (pl. render-to-texture)
- staging
  - CPU memóriában van, átmásolható bele egy default erőforrás tartalma ha olvasni akarjuk

Melyik usage mikor kell?

- immutable
  - betöltött, fix dolgok, modellek, textúrák
- dynamic
  - mozgó dolgok, részecskerendszerek
- default
  - GPU outputként is szereplő elemek
- staging
  - ha valami GPU-n kiszámoltat vissza kell olvasni a rendszermemóriába (ne kelljen…)

Erőforrások kötési módjai (bind)

- vertex buffer
- index buffer
- shader resource
- render target
- depth-stencil
- stream out
- constant buffer

Vertex buffer létrehozása

```
D3D11_BUFFER_DESC desc;
desc.Usage = D3D11_USAGE_IMMUTABLE;
desc.BindFlags = D3D11_BIND_VERTEX_BUFFER;
desc.ByteWidth = sizeof(D3DXVECTOR3) * 3;
desc.StructureByteStride = sizeof(D3DXVECTOR3);

D3DXVECTOR3 vertexPositionArray[3] = {
    D3DXVECTOR3(0, 0, 0.5),
    D3DXVECTOR3(0, 1, 0.5),
    D3DXVECTOR3(1, 0, 0.5) };
D3D11_SUBRESOURCE_DATA initData;
initData.pSysMem = vertexPositionArray;

pd3dDevice->CreateBuffer(
                &desc, &initData, &vertexBuffer);
```

Input layout létrehozása

```
D3D11_INPUT_ELEMENT_DESC positionElement;
positionElement.AlignedByteOffset = 0;
positionElement.Format =
  DXGI_FORMAT_R32G32B32_FLOAT;
positionElement.InputSlot = 0;
positionElement.InputSlotClass =
  D3D11_INPUT_PER_VERTEX_DATA;
positionElement.SemanticName = "POSITION";
positionElement.SemanticIndex = 0;
pd3dDevice->CreateInputLayout(
      &positionElement, 1,
      vertexShaderByteCode->GetBufferPointer(),
      vertexShaderByteCode->GetBufferSize(),
      &inputLayout);
```

Erőforrások felszabadítása

- COM objektumok
- referenciaszámlált
- createValami növeli a referenciszámlálót
- ha végeztünk vele
  - valami->Release();
  - csökkenti a referenciaszámlálót
  - ha máshol sem kell már fel lesz szabadítva

Erőforrásnézetek

- vannak esetek amikor az erőforrás simán beköthető, nem kell extra info
  - pl. vertex buffer a pipeline inputra
- de van amikor kell
  - pl. textúra olvasásra – melyik szeletek, melyik mipmap szintek
  - textúra render targetnek – melyik szelet melyik mipmap szintjére renderelünk
- ilyenkor létre kell hozni egy erőforrásnézetet, és azt lehet bekötni

Textúra és nézet

```
D3D11_TEXTURE2D_DESC opaqueTextureDesc; //...
device->CreateTexture2D( &opaqueTextureDesc, NULL,
  &opaqueTexture );

D3D11_SHADER_RESOURCE_VIEW_DESC opaqueSrvDesc;
opaqueSrvDesc.Format = opaqueTextureDesc.Format;
opaqueSrvDesc.ViewDimension =
  D3D11_SRV_DIMENSION_TEXTURE2D;
opaqueSrvDesc.Texture2D.MostDetailedMip = 0;
opaqueSrvDesc.Texture2D.MipLevels = 1;

device->CreateShaderResourceView( opaqueTexture,
  &opaqueSrvDesc, &opaqueSrv );
```

Egy lépésben fileból betöltve

- a Texture2D objektum nem érdekes, mindig csak a nézetet fogjuk használni

```
D3DX11CreateShaderResourceViewFromFile(
    device,
    L"media/giraffe.jpg",
    NULL, NULL, &kdSrv, NULL);
```

Erőforrásnézetek fajtái

- árnyaló-erőforrásnézet [shader resource view]
  - shader input
- rajzolásicél-nézet [render target view]
  - output merger input/output
- mélység–stencil-nézet [depth stencil view]
  - output merger input/output
- szabad hozzáférésű nézet [unordered access view]
  - pixel/compute shader input/output

Állapot-objektumok

- ezeket is a device metódusaival hozzuk létre és a context metódusaival kötjük be
- input layout ID3D11InputLayout
  - VB elemek és VS inputok összrendelése
- vertex shader ID3D11VertexShader
  - program
- pixel shader ID3D11PixelShader
  - program
- RasterizerState, BlendState, DepthStencilState

Rajzolás

- OnFrameRender esemény
- context metódusainak hívásával
- be kell állítani
  - render target
  - render state
  - shaderek, uniform paraméterek
  - textúrák
  - vertex, index buffer
- draw call

Rajzolás

ekkorát kell lépni a következő vertexhez

```
unsigned int stride = sizeof(D3DXVECTOR3);
unsigned int offset = 0;
context->IASetVertexBuffers(0, 1,
     &vertexBuffer, &stride, &offset);
context->IASetPrimitiveTopology(
     D3D11_PRIMITIVE_TOPOLOGY_TRIANGLELIST);
context->IASetInputLayout(inputLayout);
context->PSSetShaderResources(0, 1, &kdSrv);

context->VSSetShader(vertexShader, NULL, 0);
context->PSSetShader(pixelShader,  NULL, 0);

context->Draw(3, 0);
```

innen kezdve

1 db VB

elemek értelmezése

textúranézet

ennyi vertexet

innen kezdve

Effect framework

- egyszerű módszer
  - HLSL shader kódok betöltésére
  - render state beállításra
  - uniform paraméterek beállítására
    - név alapján, nem kell nekünk kitalálni, melyik regiszterben legyen
  - textúrák hozzárendelésére
    - név alapján, nem nekünk kell valamelyik textúrázó egységhez rendelni

HLSL shaderek

- effect fileban egy függvény
  - szemantika
    - input-output változók megcímkézhetők
    - az input layouttal együtt megadják, melyik paraméter értéket honnan vesszük a vertex (ill. instance) bufferből
- pass
  - render state, shader program beállító script
- technique11
  - passok gyűjteménye

HLSL shader függvény

```
struct IaosTrafo // Input Assembler Output Struct
{
    float4 pos    : POSITION;
};

struct VsosTrafo // Vertex Shader Output Struct
{
    float4 pos      : SV_POSITION;
};

VsosTrafo vsTrafo(IaosTrafo input)
{
    VsosTrafo output = (VsosTrafo)0;
    output.pos = mul(input.pos, modelViewProjMatrix);
    return output;
}
```

Technique, pass definíció

```
technique11 basic
{
  pass basic
  {
    SetVertexShader (
        CompileShader( vs_5_0, vsTrafo() ) );
    SetPixelShader(
        CompileShader( ps_5_0, psBasic() ) );
  }
}
```

Effect11

- forráskódban kiadott C++ osztálykönyvtár
  - <dxsdk dir>/Samples/C++/Effects11
- le kell fordítani magunknak
- be kell linkelni
- de bele lehet nyúlni

Effect interface

- ID3DX11Effect interface
- a device context feletti magasabb absztrakciós szint
- ugyanazt lehet vele megcsinálni, mint a device context metódusaival

Effect nélkül

- a shader összes paraméterét fel kell sorolni és megszámozni a HLSL kódban
- ID3D11DeviceContext::PSSetShaderResources
  - paraméter: erőforrásnézetek tömbje
  - a shader által várt sorrendben

Effecttel

- globális változok az effect fileban
- bármely shader használhatja őket
- beállítás név szerint
  - effect->GetVariableByName("kdTexture")->AsShaderResource()->SetResource(kdSrv);
- nem kell foglalkozni vele melyik shader melyiket használja

Több effect file használata

- sok shaderünk van, ne egy kupacon legyen
- de csak egy .fx-filet tölthetünk be
- megoldás: lesz egy main.fx, amibe beincludoljuk azokat, amik kellenek
- laboron:
  - EffectInclude osztály: saját include kereső
  - ID3DInclude interfaceből származtatható
  - effect betöltésekor megadható
  - "" a project folderben, <> a közös folderben keres

Effect és médiafileok elérési útja

- honnan tudjuk futásidőben, hogy mi a projektünk elérési útja?
  - kapja meg a programunk parancssori paraméterként
  - Project Properties/Configuration Properties/Debugging/Command Arguments

```
--solutionPath:"$(SolutionDir)"
--projectPath:"$(ProjectDir)"
```

- parancssori paraméterek kezelésére Egg::SystemEnvironment osztály

UTF8 ↔ UTF16

- elvileg mindenhol Unicode stringeket használunk
- wchar_t*, LPWSTR, std::wstring
- const wchar_t*, LPCWSTR, const std::wstring
- ha mégis UTF8 (char*, std::string) kell, akkor konverzió Egg::UftConverter segítségével