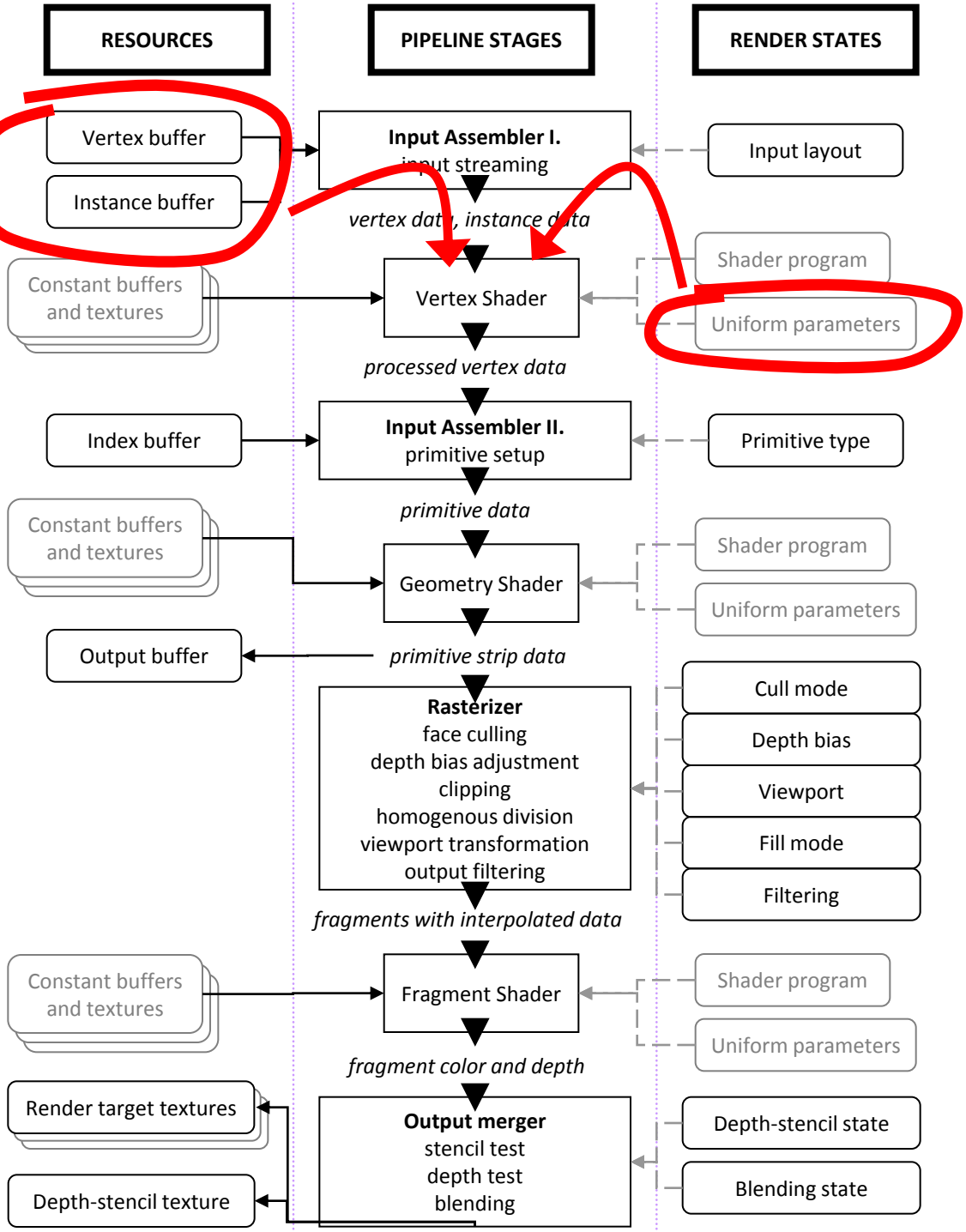
HLSL programozás

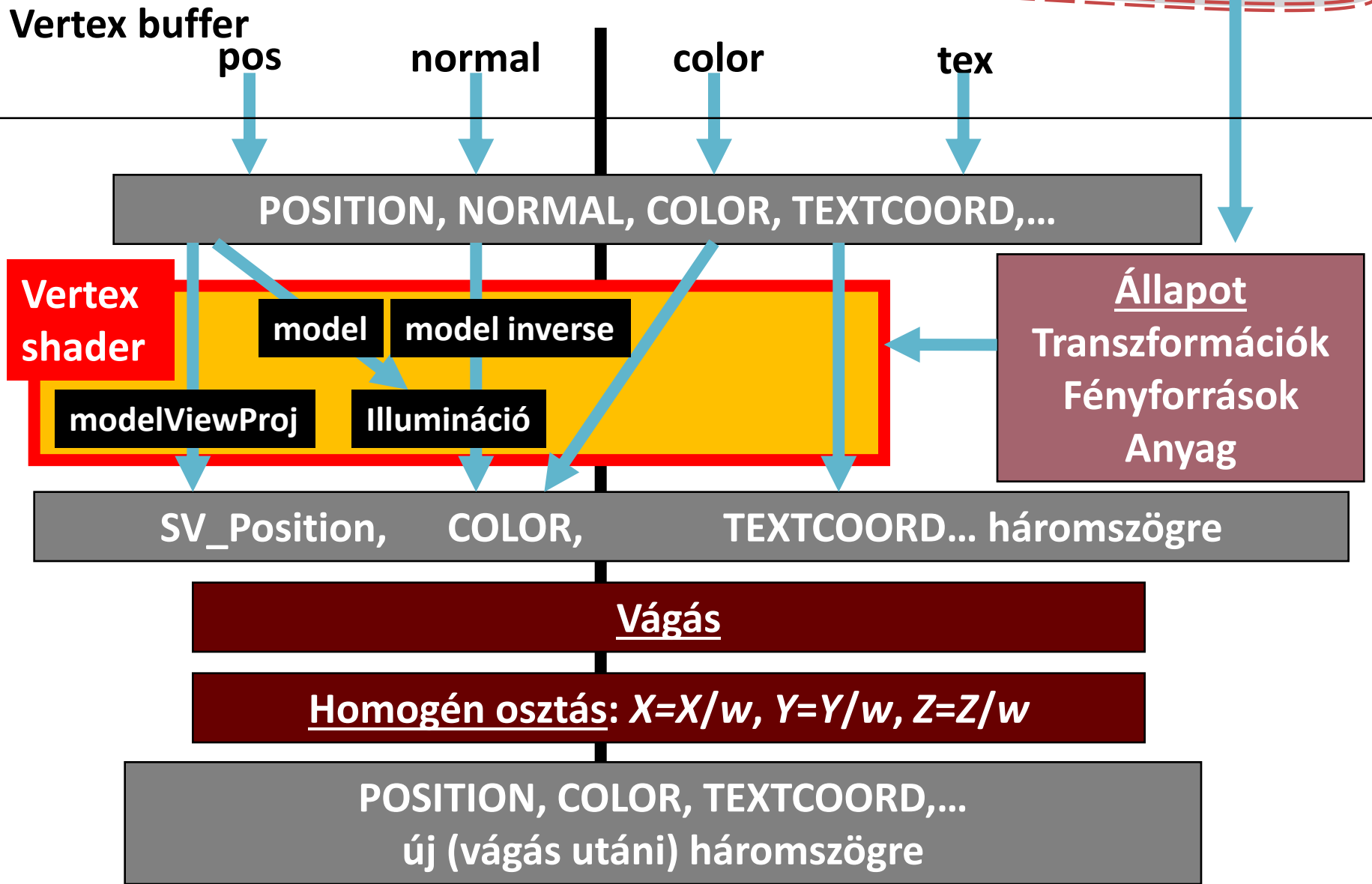
Grafikus játékok fejlesztése

Szécsi László

2013.02.16. t06-hlsl


RESOURCES
PIPELINE STAGES
RENDER STATES
Vertex buffer
Instance buffer
Input Assembler I.
input streaming
Input layout
vertex data, instance data
Constant buffers and textures
Vertex Shader
Shader program
Uniform parameters
processed vertex data
Index buffer
Input Assembler II.
primitive setup
Primitive type
primitive data
Constant buffers and textures
Geometry Shader
Shader program
Uniform parameters
Output buffer
primitive strip data
Rasterizer
face culling
depth bias adjustment
clipping
homogenous division
viewport transformation
output filtering
Cull mode
Depth bias
Viewport
Fill mode
Filtering
fragments with interpolated data
Constant buffers and textures
Fragment Shader
Shader program
Uniform parameters
fragment color and depth
Render target textures
Depth-stencil texture
Output merger
stencil test
depth test
blending
Depth-stencil state
Blending state

Vertex shader és környezete



Alap vertex shader

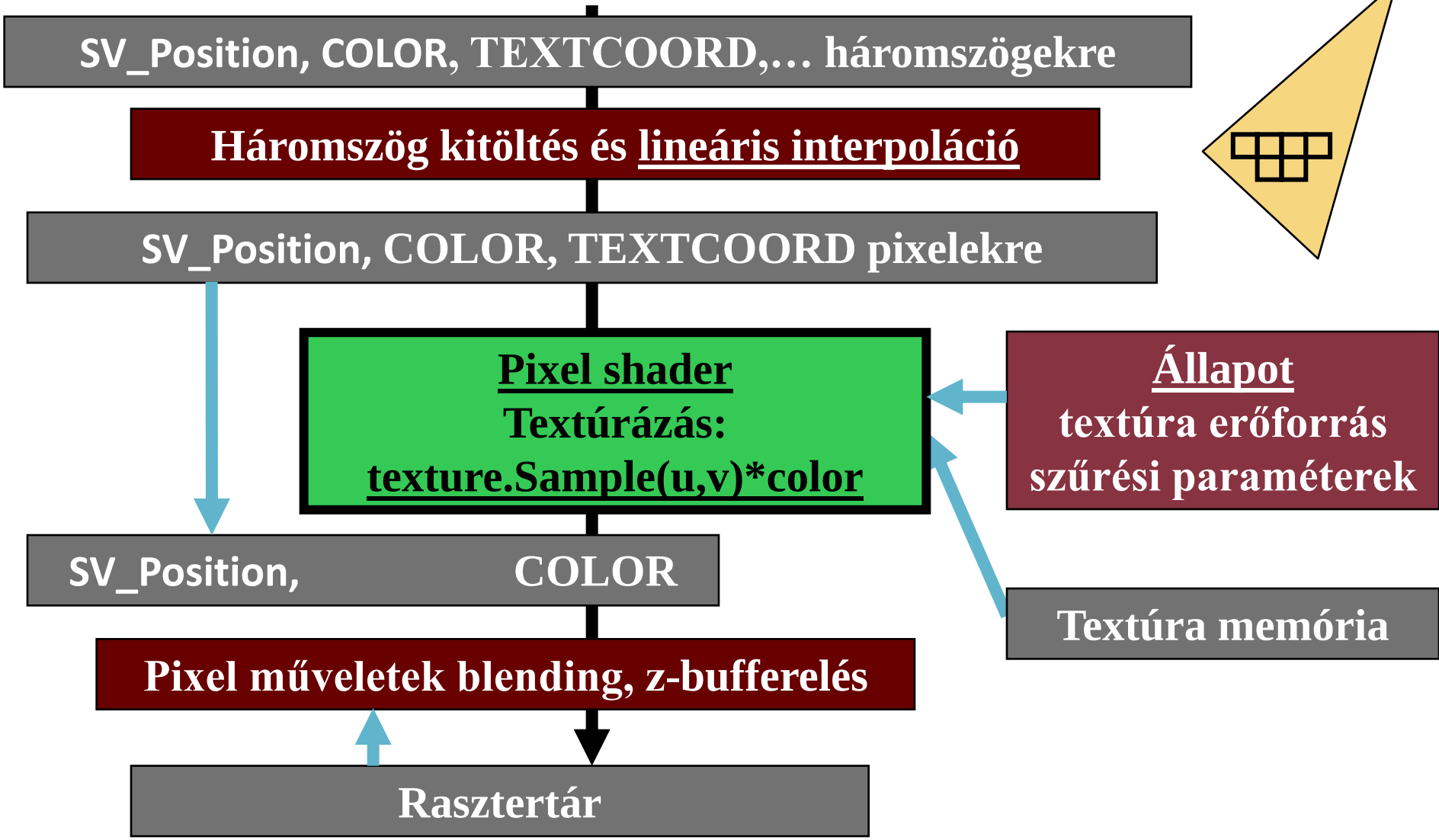
```
float4x4 modelViewProjMatrix;

struct IaosSimple {
    float4 pos        : POSITION;
    float4 color      : COLOR;
    float2 tex        : TEXCOORD;
};

struct VsosSimple {
    float4 hpos       : SV_Position;
    float3 color      : COLOR;
    float2 tex        : TEXCOORD;
};

VsosSimple vsSimple(IaosSimple input) {
    VsosSimple output = (VsosSimple)0;
    output.hpos = mul(input.pos, modelViewProjMatrix);
    output.tex = input.tex;
    output.color = input.color;
    return output;
}
```

Pixel shader és környezete



Alap pixel shader

```
SamplerState linearSampler {
    Filter = MIN_MAG_MIP_LINEAR;
    AddressU = Wrap;
    AddressV = Wrap;
};
Texture2D colorTexture;

float4 psSimple(VsosSimple input) : SV_Target {
   float4 color =
      colorTexture.sample(linearSampler, input.tex)
       * input.color;
   return color;
}
```

textúraszűrési beállítások

uniform textúra paraméter

formailag ugyanaz, mint ami a vertex shader outputja volt, de 3 vertex között interpolált értékek vannak benne

Effect file szerkesztés

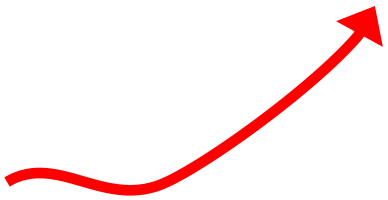
- VS 2008/2010
  - NShader extension: syntax highlighting
- FX composer
  - syntax highlighting
  - azonnali fordítás és hibajelzés
  - eredmény látható
  - uniform paraméterek kézzel állíthatók
- fxc.exe
  - megnézni a gépi kódot

Effect filebe mit írunk

- globális változók, textúra samplerek
- HLSL függvények
- technique definíciók
  - pass definíciók
    - render state
    - vertex shader
    - pixel shader

Shader fordítás effect fileból

```
RasterizerState wireframeRasterizer {
  CullMode = none;
  FillMode = wireFrame;
};
technique11 simple {
 pass simple {
  VertexShader = compile vs_5_0 vsSimple();
  SetRasterizerState( wireframeRasterizer );
  PixelShader  = compile ps_5_0 psSimple();
  }
}
```



Profil

- milyen gépi kódot ért a GPU?
  - szerencsére nem minden kártyán más
  - szabványosítva van
  - ezek a Shader Model verziók
  - illetve azokon belül a különböző shaderekhez tartozó profilok
    - mert más utasítások lehetnek pl. egy vertex és egy pixel shaderben

Profilok

- Shader Model 1: vs_1_0, ps_1_0 … ps_1_4
  - pár utasítás, regisztertologatás, ...
- Shader Model 2: vs_2_0, ps_2_0
  - 256 utasítás, swizzle, 16 textúra
- Shader Model 3 : vs_3_0, ps_3_0
  - vertex texture, dynamic flow, 512 op, dep. read
- Shader Model 4: vs_4_0, gs_4_0, ps_4_0
  - végtelen minden, stream, texture object...
- Shader Model 5: HW tesszelláció, dyn. bind…

Shader Model 5

- utasításból nem fogunk kifogyni
- textúraolvasás bármely shaderből
  - de a mipmap szint csak pixel shaderből automatikus
- geometry shader, tesszellátor
- textúratömbök
- random access írható/olvasható textúrák
- atomi műveletek
- int aritmetika, bitműveletek

Vektorműveletek

- beépített vektortípusok
  - float, float2, float3, float4 (bool, int, uint ugyanígy)
- változó definíció, mint C++-ban

```
float4 color;
```

- inicializálás értékadás numerikus konstruktorral

```
float4 color = float4(1, 1, 0.3, 1);
color = float4(0.1, 0, 0.3, 1);
```

Swizzle

```
float4 color = float4(1, 1, 0.3, 1);
color.x = 0.5;
float2 twochannels = color.rg;

float4 pos;
pos.xyz = float3(1, 0, 0);
pos.w = 1;
```

Műveletek az összes elemre vonatkoznak

```
float3 v1 = float3(1, 5, 0.3);
float3 v2 = float3(0, 0, -0.3);
bool3 b1 = v1 < v2;
if( all(b1) )
  v1 = v2;
v1 *= 2; //összes elem * 2
v1 *= v2; //elemenként

float scalarProd = dot(v1, v2);
float3 crossProd = cross(v1, v2);
crossProd = normalize(crossProd);
```