PGRe
DirectX

Jozef Kobrtek

Brno University of Technology, Faculty of Information Technology
Božetěchova 1/2, 612 66 Brno - Královo Pole
ikobrtek@fit.vutbr.cz

Today

- history of DirectX/3D APIs
- DirectX components
- Direct3D
- DirectX x OpenGL

Microsoft DirectX

Set of libraries:

- 2D/3D graphics
- sound
- input devices
- video
- networking

- 1994 – Windows 95 almost ready
- game devs not satisfied with Win95
- no direct access to hardware (graphic card, input devices, sound card)
- DOS still better platform for games

Solution: Windows Games SDK

- developed within months
- released September 1995
- later renamed to **DirectX 1.0**
- no games released using Direct3D

DirectX 2.0

- June 1996
- developers very cautious (possible future obsolescence of DX)

-> DirectX 2.0 now part of Win95 OSR2 and NT 4.0

Road Rash, DirectX 2.0

- September 1996
- MMX-optimized
- new components: Direct3D and DirectPlay
- Microsoft acquired Rendermorphics, their Reality Lab API is basis for Direct3D

MechWarrior2, DirectX 3.0 (supported DirectPlay), 1996

Descent 2, DirectX 3.0, 1996

- DX4 – skipped (planned for December 1996)
- DX5 - released with Windows 98 (June 1998)
- multiple monitors
- force-feedback controllers

Redline Racer, DirectX 5.0 (1998)

The others…

SW:

- OpenGL
- 3dfx – Glide (Windows, Linux)
- Rendition - Speedy3D(DOS)/Rredline(Windows)
- Apple – QuickDraw 3D
- Microsoft – DirectX

HW:

- many vendors (S3, SiS, 3dfx, nVidia, Ati, Matrox, Rendition, Trident, Oak, 3Dlabs, Chromatic,…)

3dfx Glide

- fastest API of late 1990s
- subset of OpenGL
- simple to code in
- high-quality output
- only 3dfx Voodoo cards
- very popular

Microsoft in trouble

DirectX used, but:

- not the fastest
- not elegant to work in
- new versions going out very often - chaos
- only partial support in HW/drivers

-> closer cooperation with HW vendors

- August 1998, Windows 98
- port for Windows CE
- SSE and 3DNow! support
- multitexturing (nVidia)
- stencil buffer
- DXTC texture compression (S3)
- bump mapping (Matrox)

- 1999
- close cooperation with nVidia (release of GeForce 256)
- Transformation, clipping and lightning (**T&L**)
- specular lightning, reflections
- environmental cube maps
- DDS – DirectDraw Surface
- DX surpasses OpenGL – vertex buffers on graphic card
- last fixed-function pipeline DX

	1996	1997	1998	1999
Application tasks (move objects according to application, move/aim camera)	CPU	CPU	CPU	CPU
Scene level calculations (object level culling, select detail level, create object mesh)	CPU	CPU	CPU	CPU
 Transform	CPU	CPU	CPU	**GPU**
 Lighting	CPU	CPU	CPU	**GPU**
 Triangle Setup and Clipping	CPU	Graphics Processor	Graphics Processor	**GPU**
 Rendering	Graphics Processor	Graphics Processor	Graphics Processor	**GPU**

3D Application and API

3D Graphics Pipeline

- T&L was implemented in 3DMark 2000 as well
- because many vendors had only SW support of T&L (= transformations still on CPU), their cards scored less – which reflected in reviews and sales
- lot of vendors left market/went bankrupt

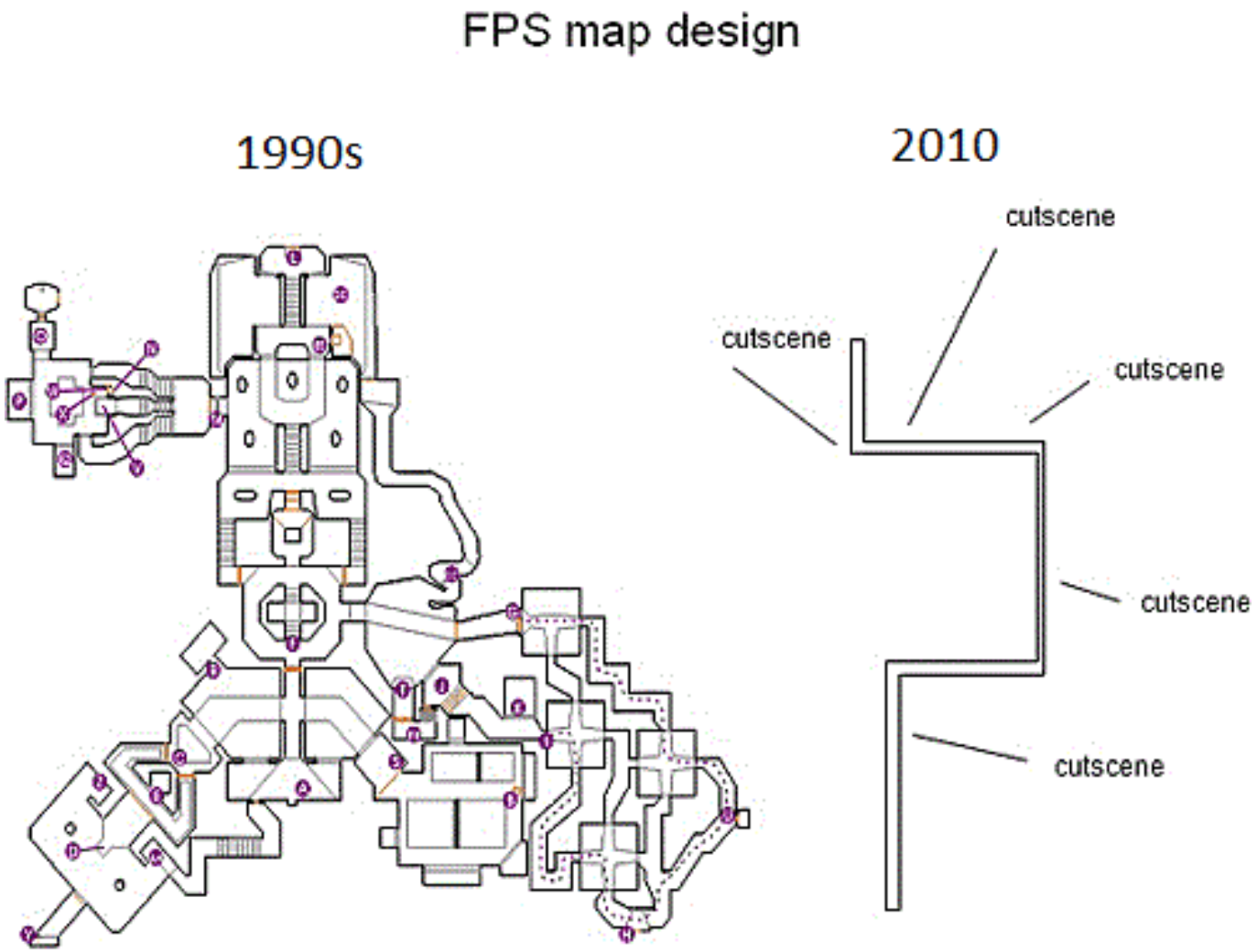
- 3DMark 2000 video: LINK

End of fixed-function pipeline

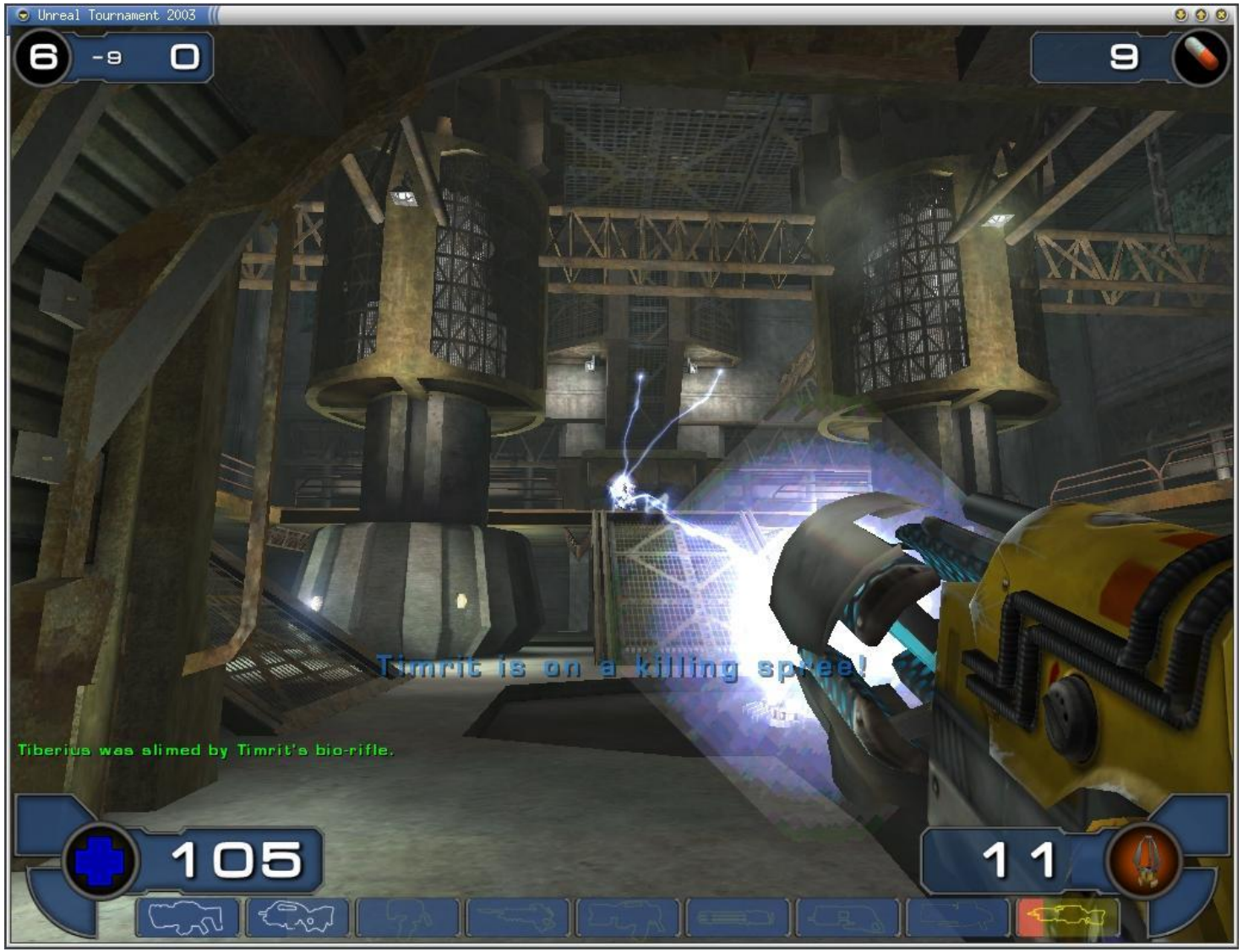
- ✓ easy
- ✓ more focus on level-design in games
- ✓ used even after newer DX versions (2010!)

- only basic effects (= features supported by HW)
- low flexibility
- low programmability

Level design…



- November 2000
- programmable vertex and pixel shader (Shader Model 1.0 – 1.4)
- shader programs written in assembly
- limited shader code length (up to 128 VS instructions, only 12-22 PS instructions)
- last DX for Windows 95



Unreal Tournament 2003, DirectX 8, 2002

Max Payne 2, DirectX 8.1, 2003

- 2002 – 2006
- Shader Model 2.0, 3.0
- 256 VS instructions (512 in SM 3.0)
- 96 PS instructions (512 in SM 3.0)
- HLSL (High-Level Shading Language)
- multiple render targets
- long list of new features

Crysis, DirectX 9 Shader Model 2.0, 2007

- 2006 (Windows Vista+ only)
- Shader Model 4.0, unified shader architecture
- very long shaders (65K instructions), many constants (16*4K)
- new pipeline stage - Geometry Shader
- new driver model in Vista
- no more extensions, strict API
- some extra features in 10.1 (cubemap arrays, more shader inputs)

Assassin's Creed, DirectX 10, 2007

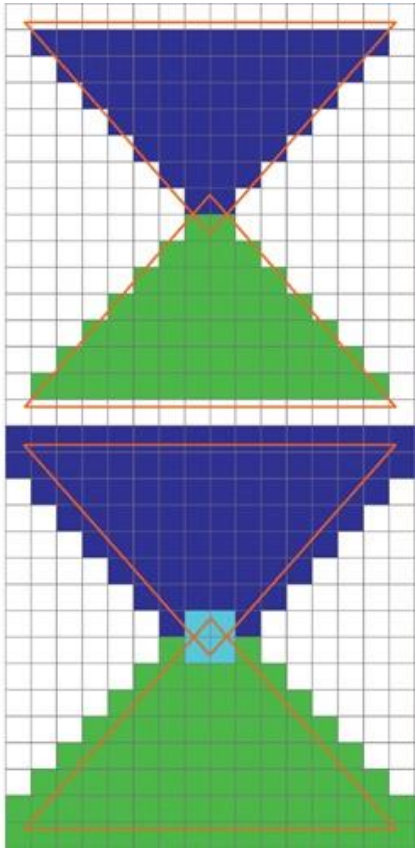
- 2009, Windows 7
- Shader Model 5.0
- Tessellation
- multi-threaded rendering (limited…)
- Compute Shaders (3 years before OpenGL)

Dirt 3, DirectX 11, 2011

- late 2015 (PC, Xbox One)
- another clone of AMD Mantle
- focused on reducing CPU overhead
- multi-core, closer to HW
- RayTacing (DXR)

New features:

- **conservative rasterization**
- bindless resources
- JPEG texture compression
- new blend modes



Think of DX12 as "DirectX without training wheels, without brakes, and at the moment maybe without tires“

- lighter driver -> more work for developer
  - must manage physical memory, all synchronization (almost nothing is implicitly synchronized), command queues, etc
  - **No validation during runtime**
  - Debug layer -> validation can be turned on during development
- Support for multi-GPU rendering in one context
- definitely NOT for 3D graphics beginners…

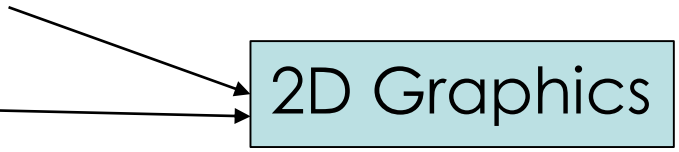
DIRECTX COMPONENTS

DX 10 and newer - Components

- DirectX Graphics
  - DirectDraw (obsolete)
  - Direct2D
  - Direct3D
  - DXGI
  - DirectWrite
- DirectCompute
- Games for Windows Live
- XInput
- XAudio
- DirectX Media Objects
- DirectSetup

DX 10 and newer - Components

- DirectX Graphics
  - DirectDraw (obsolete)
  - Direct2D
  - Direct3D
  - DXGI
  - DirectWrite
- DirectCompute
- Games for Windows Live
- XInput
- XAudio
- DirectX Media Objects
- DirectSetup

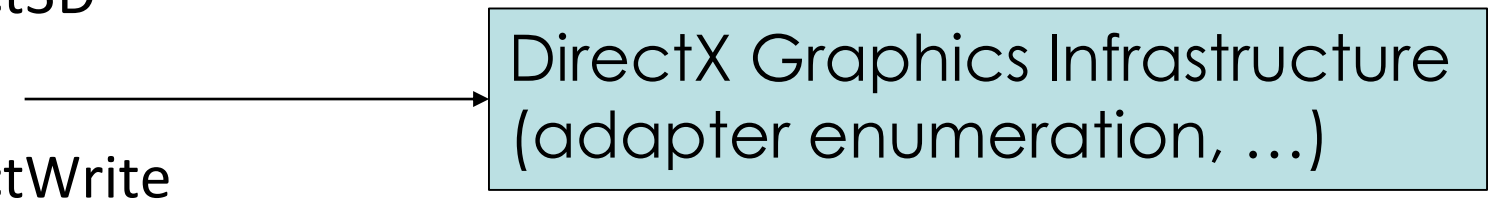


DX 10 and newer - Components

- DirectX Graphics
  - DirectDraw (obsolete)
  - Direct2D
  - Direct3D → 3D Graphics
  - DXGI
  - DirectWrite
- DirectCompute
- Games for Windows Live
- XInput
- XAudio
- DirectX Media Objects
- DirectSetup

DX 10 and newer - Components

- DirectX Graphics
  - DirectDraw (obsolete)
  - Direct2D
  - Direct3D
  - DXGI
  - DirectWrite
- DirectCompute
- Games for Windows Live
- XInput
- XAudio
- DirectX Media Objects
- DirectSetup

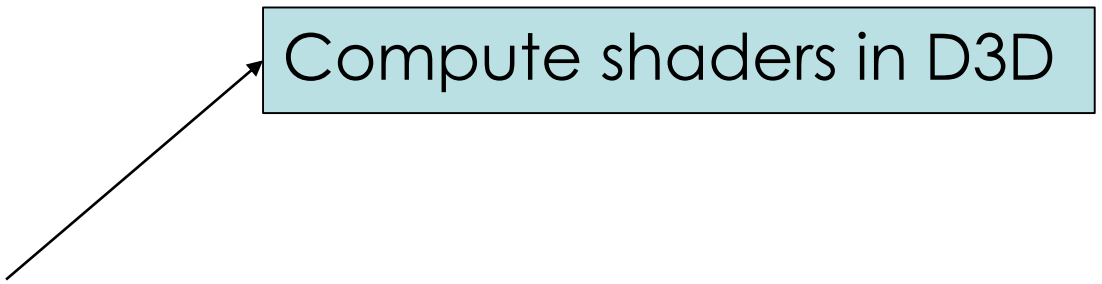


DX 10 and newer - Components

- DirectX Graphics
  - DirectDraw (obsolete)
  - Direct2D
  - Direct3D
  - DXGI
  - DirectWrite → Font rendering
- DirectCompute
- Games for Windows Live
- XInput
- XAudio
- DirectX Media Objects
- DirectSetup

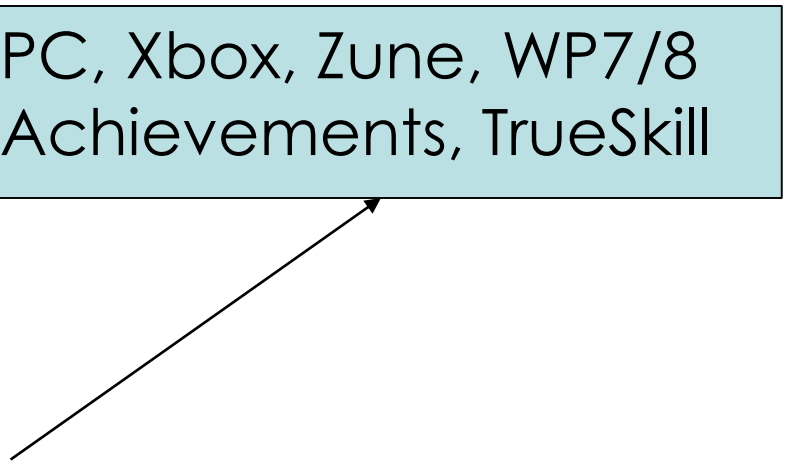
DX 10 and newer - Components

- DirectX Graphics
  - DirectDraw (obsolete)
  - Direct2D
  - Direct3D
  - DXGI
  - DirectWrite
- DirectCompute
- Games for Windows Live
- XInput
- XAudio
- DirectX Media Objects
- DirectSetup



DX 10 and newer - Components

- DirectX Graphics
  - DirectDraw (obsolete)
  - Direct2D
  - Direct3D
  - DXGI
  - DirectWrite
- DirectCompute
- Games for Windows Live
- XInput
- XAudio
- DirectX Media Objects
- DirectSetup

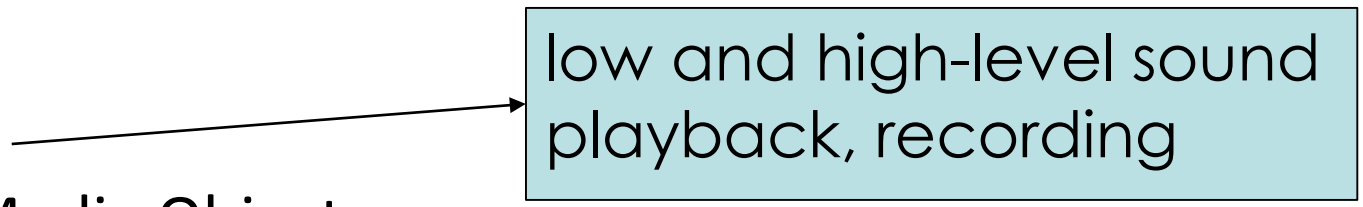


DX 10 and newer - Components

- DirectX Graphics
  - DirectDraw (obsolete)
  - Direct2D
  - Direct3D
  - DXGI
  - DirectWrite
- DirectCompute
- Games for Windows Live
- XInput → Input from next-gen controllers
- XAudio
- DirectX Media Objects
- DirectSetup

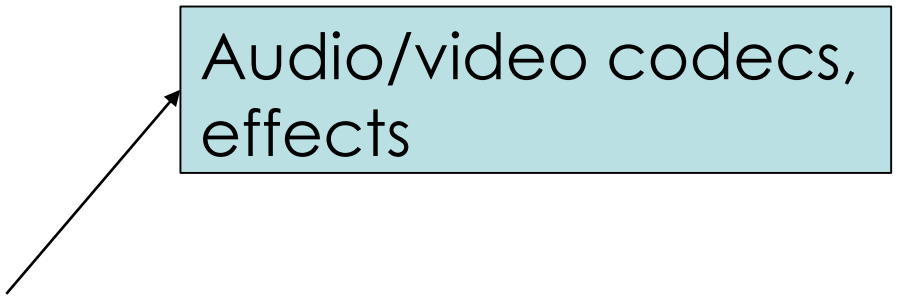
DX 10 and newer - Components

- DirectX Graphics
  - DirectDraw (obsolete)
  - Direct2D
  - Direct3D
  - DXGI
  - DirectWrite
- DirectCompute
- Games for Windows Live
- XInput
- XAudio
- DirectX Media Objects
- DirectSetup



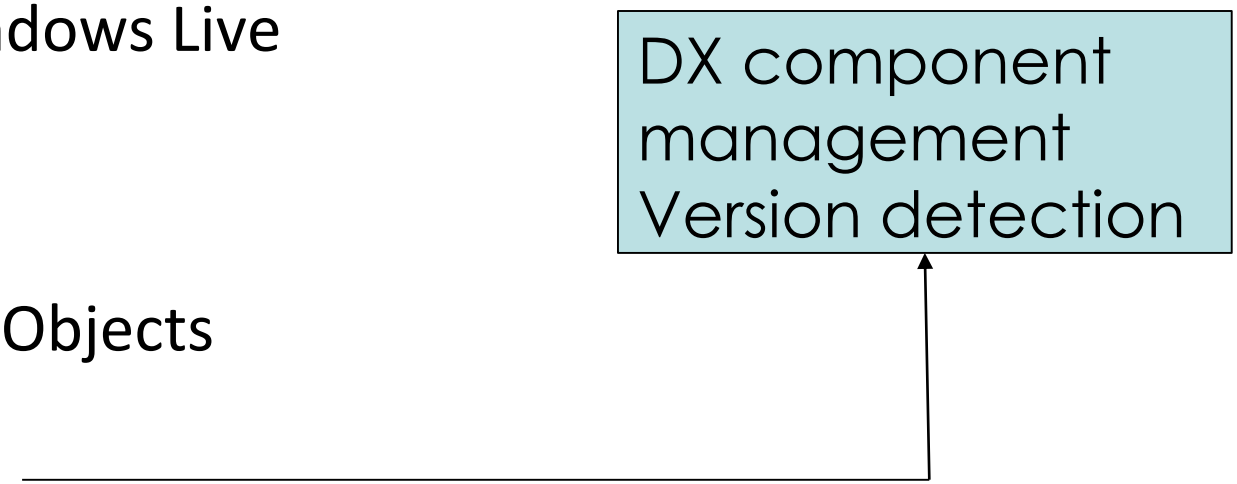
DX 10 and newer - Components

- DirectX Graphics
  - DirectDraw (obsolete)
  - Direct2D
  - Direct3D
  - DXGI
  - DirectWrite
- DirectCompute
- Games for Windows Live
- XInput
- XAudio
- DirectX Media Objects
- DirectSetup

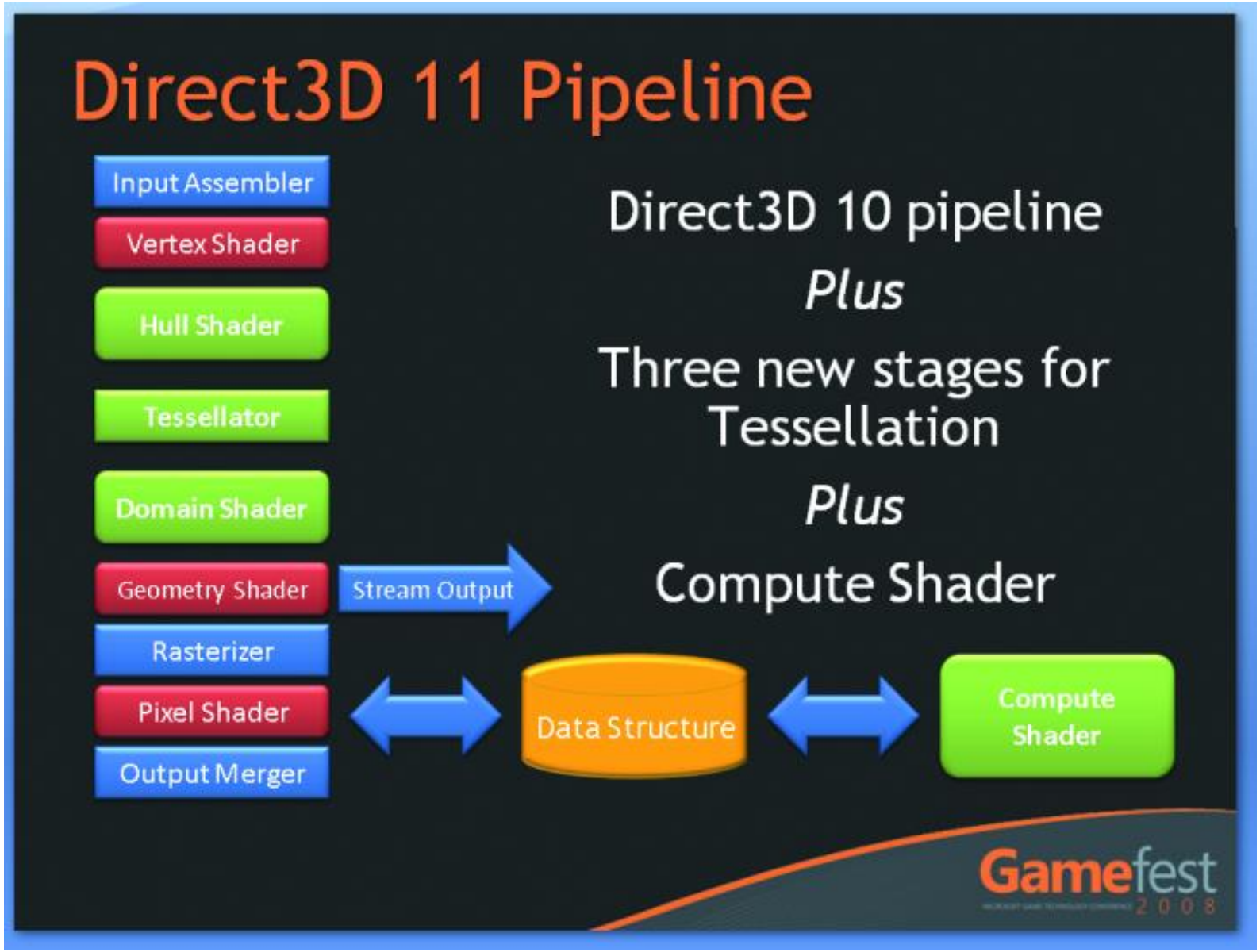


DX 10 and newer - Components

- DirectX Graphics
  - DirectDraw (obsolete)
  - Direct2D
  - Direct3D
  - DXGI
  - DirectWrite
- DirectCompute
- Games for Windows Live
- XInput
- XAudio
- DirectX Media Objects
- DirectSetup



DIRECT3D 11


Direct3D 11 Pipeline
Input Assembler
Vertex Shader
Hull Shader
Tessellator
Domain Shader
Geometry Shader
Stream Output
Rasterizer
Pixel Shader
Output Merger
Direct3D 10 pipeline
Plus
Three new stages for Tessellation
Plus
Compute Shader
Data Structure
Compute Shader
Gamefest

- 2 important handles
  - device (ID3D11Device)
  - device context (ID3D11DeviceContext)
- objects created on device, used on context
  - fill in a structure
  - register on device
  - use on context
- context functions begin with pipeline stage prefix (f.e. PSSetShader – sets pixel shader program) – IA, VS, RS, PS, OM,…

Context creation:

- specify device type
  - D3D_DRIVER_TYPE_HARDWARE
  - D3D_DRIVER_TYPE_WARP
  - D3D_DRIVER_TYPE_REFERENCE
  - D3D_DRIVER_TYPE_SOFTWARE

- specify feature level
  - D3D_FEATURE_LEVEL_11_0
  - D3D_FEATURE_LEVEL_10_1
  - D3D_FEATURE_LEVEL_10_0

Direct3D 11 – creating 2D texture

```
//Create tex2D
D3D11_TEXTURE2D_DESC tex_desc;
ID3D11Texture2D* texture;
ZeroMemory(&tex_desc, sizeof(tex_desc)); //built-in macro for memory zeroing
tex_desc.Width = width;
tex_desc.Height = height;
tex_desc.ArraySize = 1;
tex_desc.SampleDesc.Count = 1;
tex_desc.SampleDesc.Quality = 0;
tex_desc.Format = DXGI_FORMAT_R8G8B8A8_UNORM;
tex_desc.Usage = D3D11_USAGE_DEFAULT;
tex_desc.BindFlags = D3D11_BIND_SHADER_RESOURCE; //can be also bound f.e as render target, or both
tex_desc.CPUAccessFlags = 0;
tex_desc.MiscFlags = 0;
tex_desc.MipLevels = 1; //no mipmaps
//Texture data descriptor – data loaded from a file
D3D11_SUBRESOURCE_DATA tData;
tData.pSysMem = textureImageData;
tData.SysMemPitch = 4*tex_desc.Width;
tData.SysMemSlicePitch = 0;
//Create texture and load it's data
result = device->CreateTexture2D( &tex_desc, &tData, &texture);
if( FAILED( result ) )
{
    //some error
}
```

- Texture is created, but not bindable to pipeline!
- We need additional object – textures are shader resources in general, so we need a **Shader Resource View**
- can be also bound f.e. with Render Target (if we want to render to texture)

```
//Create SRV from texture
//We will use texture we created earlier
D3D11_SHADER_RESOURCE_VIEW_DESC srvDesc;
ID3D11ShaderResourceView* srv;

ZeroMemory(&srvDesc, sizeof(srvDesc)); //Macro for memory zeroing

srvd.Format = tex_desc.Format; //Use same format as texture
srvd.ViewDimension = D3D11_SRV_DIMENSION_TEXTURE2D; //It's a 2D texture
srvd.Texture2D.MostDetailedMip = 0;
srvd.Texture2D.MipLevels = tex_desc.MipLevels;

result = dev->CreateShaderResourceView(texture,&srvDesc, srv);
if( FAILED( result ) )
{
    //some error
}
```

Now texture (as SRV) can be bound to pipeline

OpenGL - create 2D texture

```
glBindTexture(GL_TEXTURE_2D, texid);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER,
GL_NEAREST);
glTexParameterf(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER,
GL_NEAREST);
//Supply data
glTexImage2D(GL_TEXTURE_2D, 0, GL_RGBA16F, w, h, 0,
GL_RGBA, GL_FLOAT, data);
glBindTexture(GL_TEXTURE_2D, 0);
```

But everything else is set to default

- shaders are **NOT** linked to a program
  - VS input must match input layout
  - VS output must match PS input
- can be stored in one file
- have more flexible syntax (no “main” function)

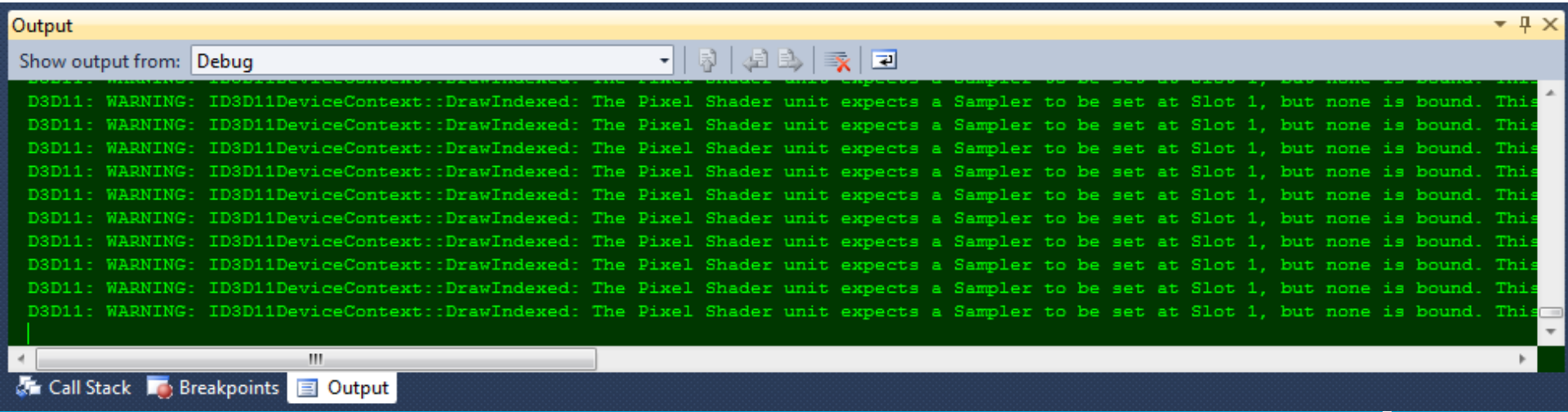
```
cbuffer MatrixBuffer     : register(b0)  //register numbers for binding buffers from CPU
{
        matrix mWorld;
        matrix mView;
        matrix mProj;
};
//input descriptors
struct VS_INPUT
{
        float3 pos              : POSITION;
};

struct PS_INPUT
{
        float4 pos              : SV_POSITION; //SV means System Value, SV_POSITION is transformed position. Defines, how
pipeline treats the value
};
//Vertex shader
PS_INPUT vShader(VS_INPUT input)
{
        PS_INPUT ret;
        ret.pos = mul(float4(input.pos, 1.0f), mWorld);
        ret.pos = mul(ret.pos, mView);
        ret.pos = mul(ret.pos, mProj);
        return ret;
}
//Pixel shader
float4 pShader(PS_INPUT input) : SV_Target
{
        return float4(1.0f, 0.0f, 0.0f, 1.0f);
}
```

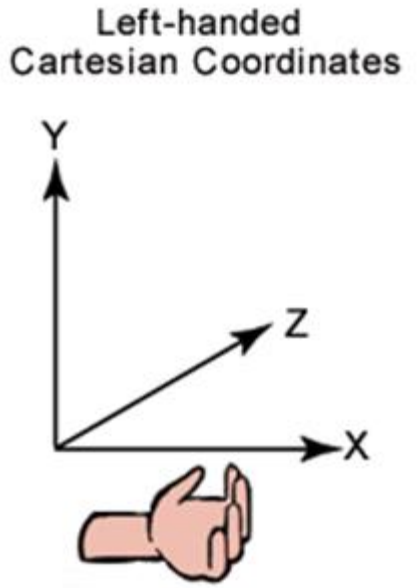
Debugging is **awesome**!

DirectX tells you, what's wrong

- When debugging, specify debug flag to device creation
- In DX control panel, add your app to show debug output

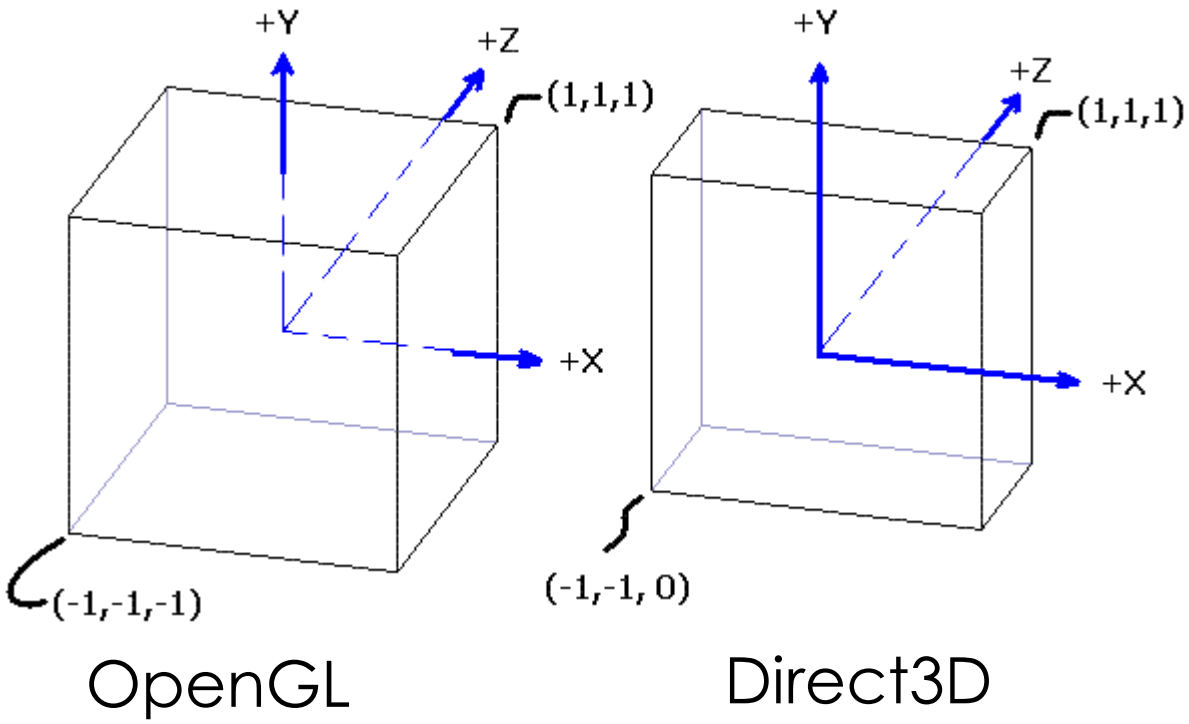


- DirectX is **left-handed**
  - worlds space, view space, clip space, NDC, screen
- Uses **row-major** notation
- transformations are applied from the **right**



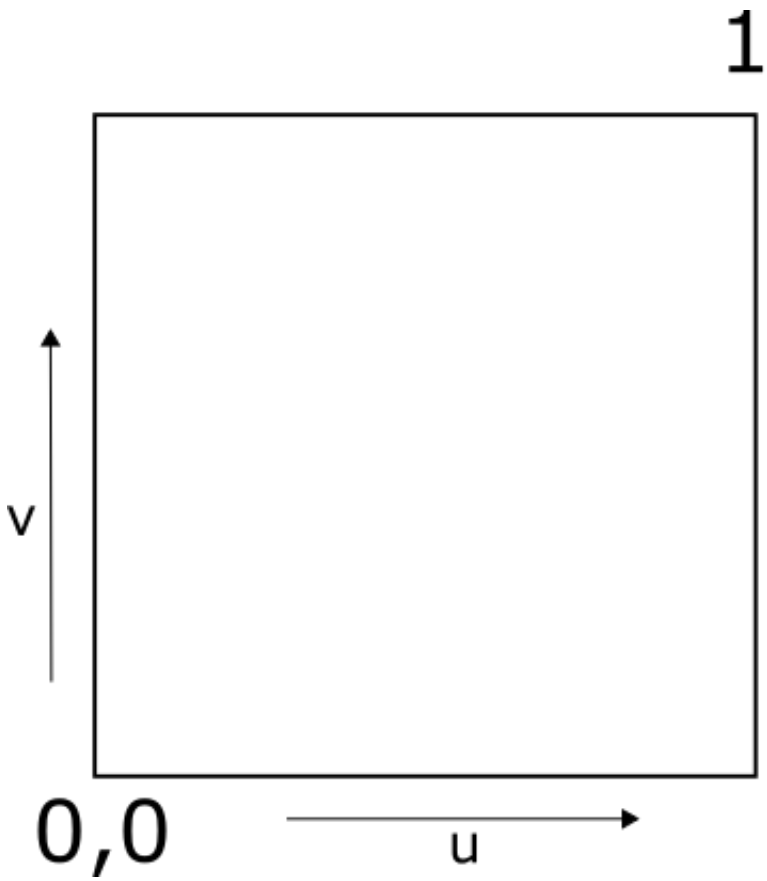
NDC is not a cube!

- x, y in <-1, 1>
- **z in <0, 1>**

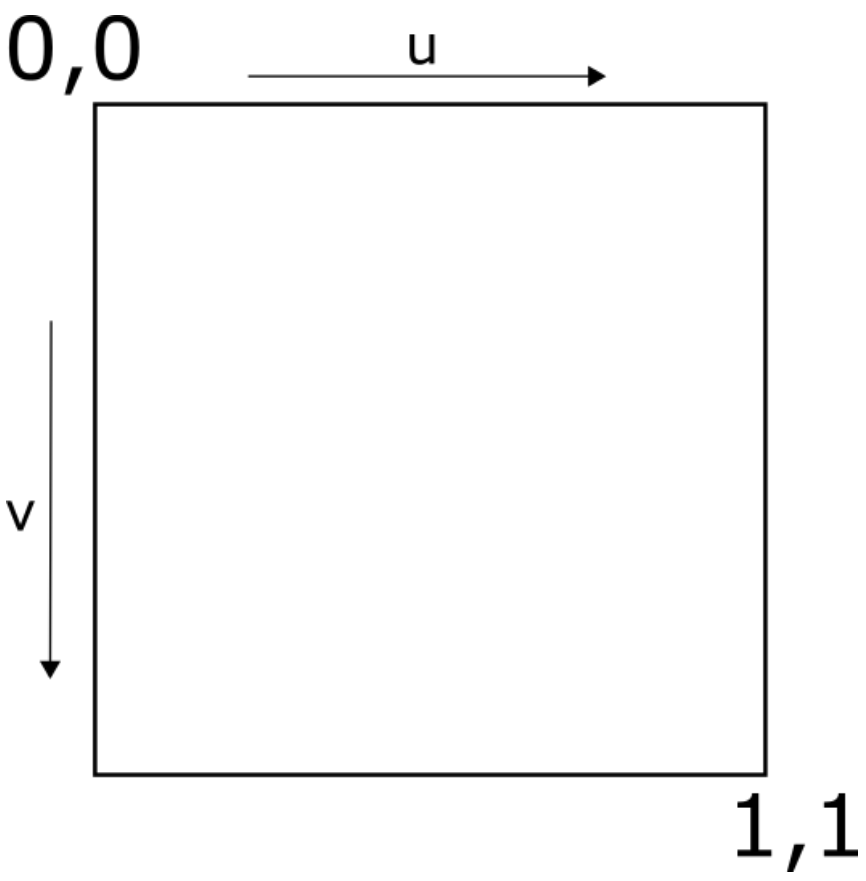


OpenGL Direct3D

- Texture space is different

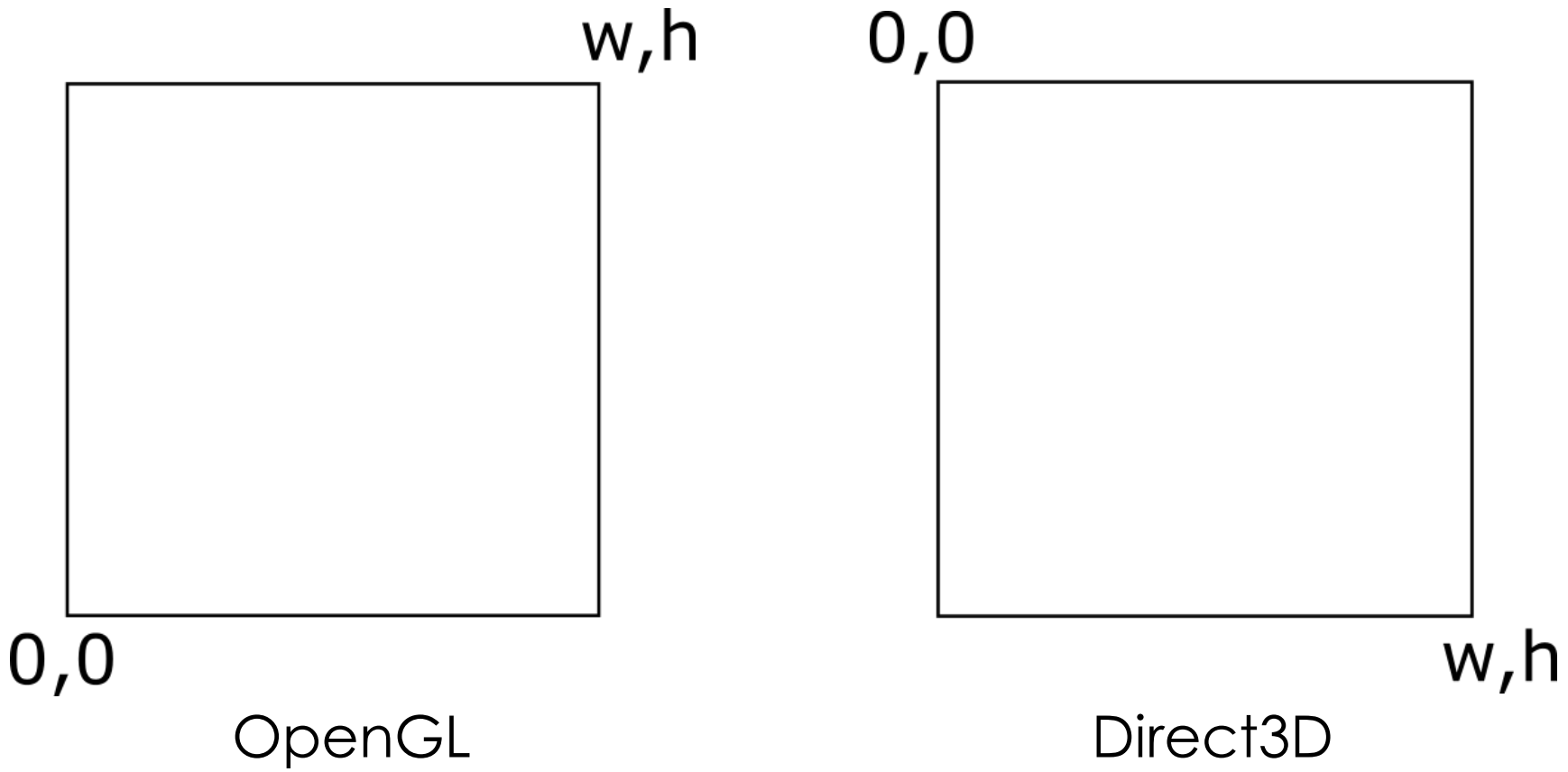


OpenGL



Direct3D

- …and so is screen space



Thank You For Your Attention!