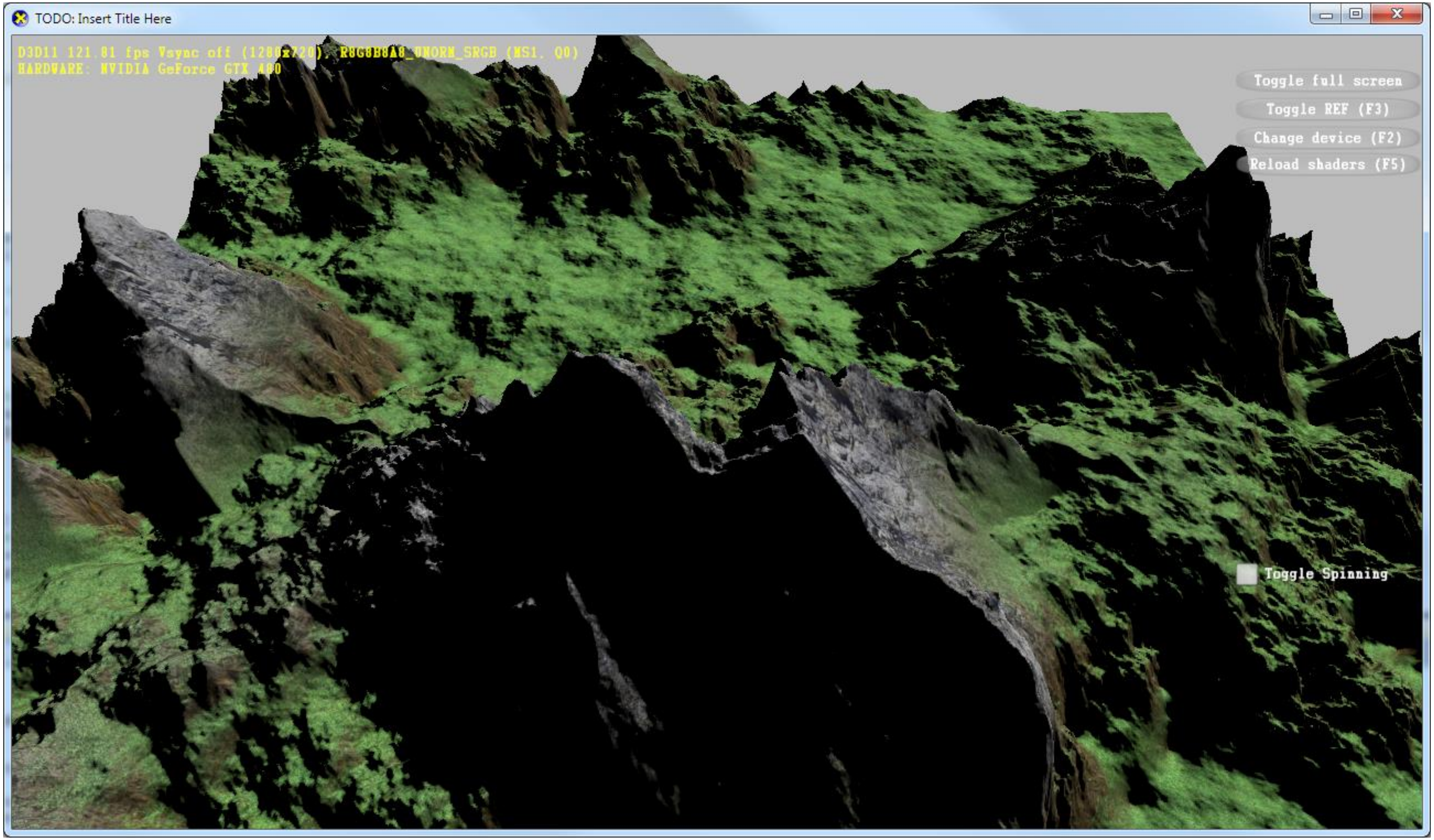
Praktikum: Echtzeit Computergrafik

tum.3D

computer graphics & visualization

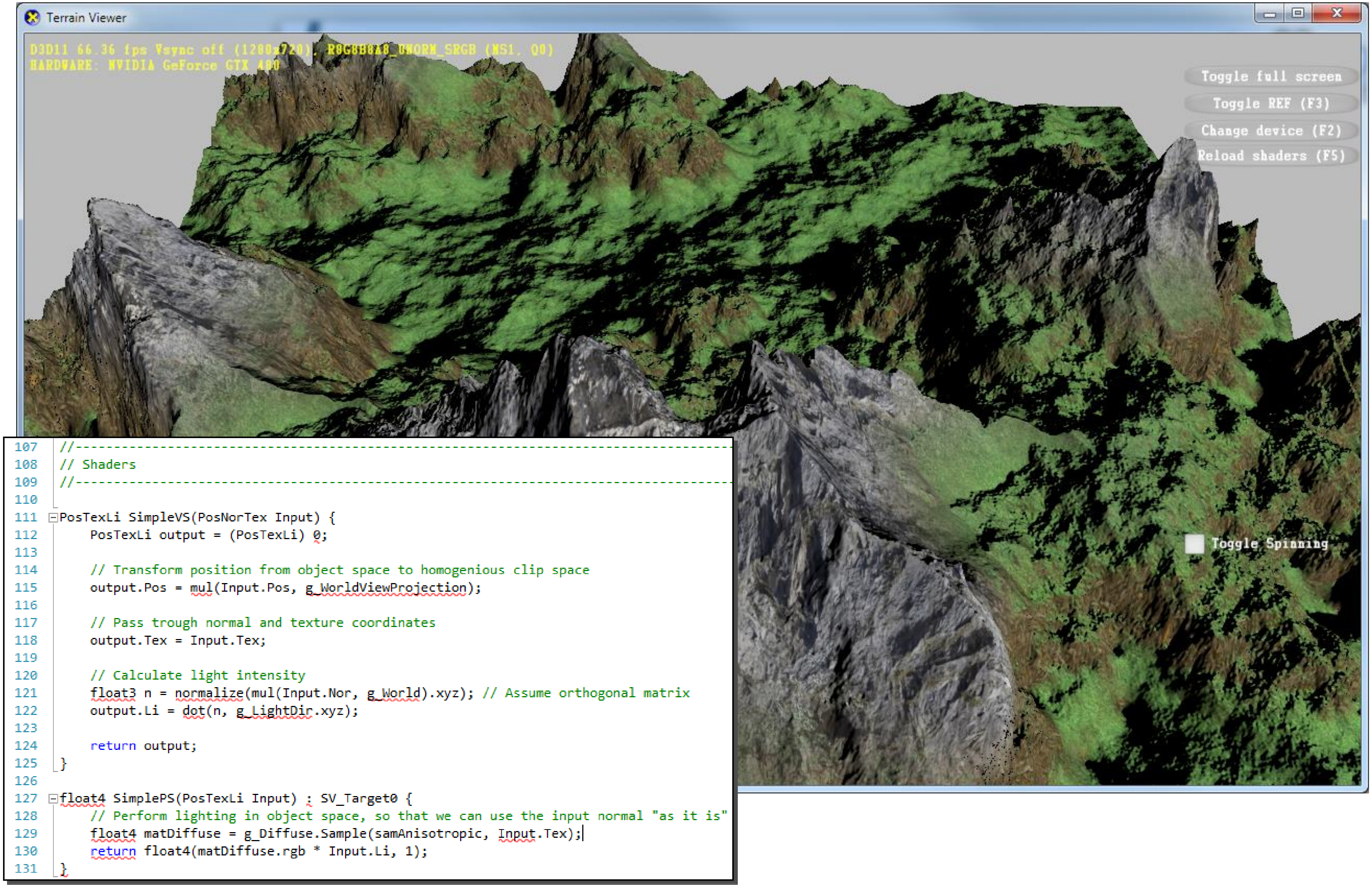
This week: Realtime Rendering…



TODO: Insert Title Here

```
D3D11 121.81 fps Vsync off (1280x720), R8G8B8A8_UNORM_SRGB (MS1, Q0)
HARDWARE: NVIDIA GeForce GTX 480
```

…using shaders



```
D3D11 66.36 fps Vsync off (1280x720), R8G8B8A8_UNORM_SRGB (MS1, Q0)
HARDWARE: NVIDIA GeForce GTX 480
```

```
//--------------------------------------------------------------------------------------
// Shaders
//--------------------------------------------------------------------------------------

PosTexLi SimpleVS(PosNorTex Input) {
    PosTexLi output = (PosTexLi) 0;

    // Transform position from object space to homogenious clip space
    output.Pos = mul(Input.Pos, g_WorldViewProjection);

    // Pass trough normal and texture coordinates
    output.Tex = Input.Tex;

    // Calculate light intensity
    float3 n = normalize(mul(Input.Nor, g_World).xyz); // Assume orthogonal matrix
    output.Li = dot(n, g_LightDir.xyz);

    return output;
}

float4 SimplePS(PosTexLi Input) : SV_Target0 {
    // Perform lighting in object space, so that we can use the input normal "as it is"
    float4 matDiffuse = g_Diffuse.Sample(samAnisotropic, Input.Tex);
    return float4(matDiffuse.rgb * Input.Li, 1);
}
```

Assignment 5

Goal:

- Write the vertex shader and the pixel shader programs, which control how the graphics card renders your terrain

- Vertex Shader:
  - Calculation of the Vertex Positions (instead of vertex buffer)
  - Transformation

- Pixel Shader:
  - (Simple) Texturing
  - (Simple) Normal-Mapping
  - (Simple) Lighting

Terrain Renderer Old vs. New

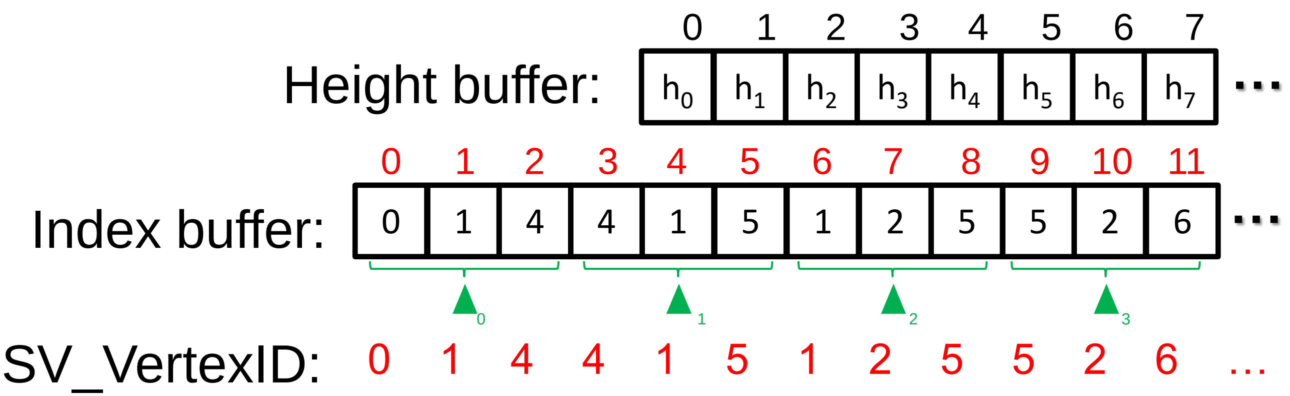
„Fixed Function“

- CPU
  - calculate 3D vertex positions from heightmap
  - calculate 3D vertex normals from heightmap
  - calculate texture coordinates
  - triangulate terrain with index buffer
  - feed geometry as vertex- and index-buffer to the graphics pipeline
- GPU: Vertex Processing
  - transform vertex position and normal
  - calculate per-vertex lighting
- GPU: Fragment Processing
  - sample color texture
  - combine vertex color (lighting) with texture color (material)

Programmable

- CPU
  - triangulate terrain with index buffer, but don‘t calculate any vertex positions
  - bind terrain maps to the graphics pipeline (height, normal, color)
  - tell the graphics pipeline what indices to draw, but don‘t feed any geometry
- GPU: Vertex Shader
  - calculate 3D vertex position from heightmap
  - transform vertex position
  - calculate texture coordinates
- GPU: Pixel Shader
  - sample color and normal texture
  - calculate per-pixel lighting
  - Combine light color with texture color (material)

Vertex Displacement in Shader



Height buffer:

0	1	2	3	4	5	6	7	
h_0	h_1	h_2	h_3	h_4	h_5	h_6	h_7	…

Index buffer:

0	1	2	3	4	5	6	7	8	9	10	11	
0	1	4	4	1	5	1	2	5	5	2	6	…

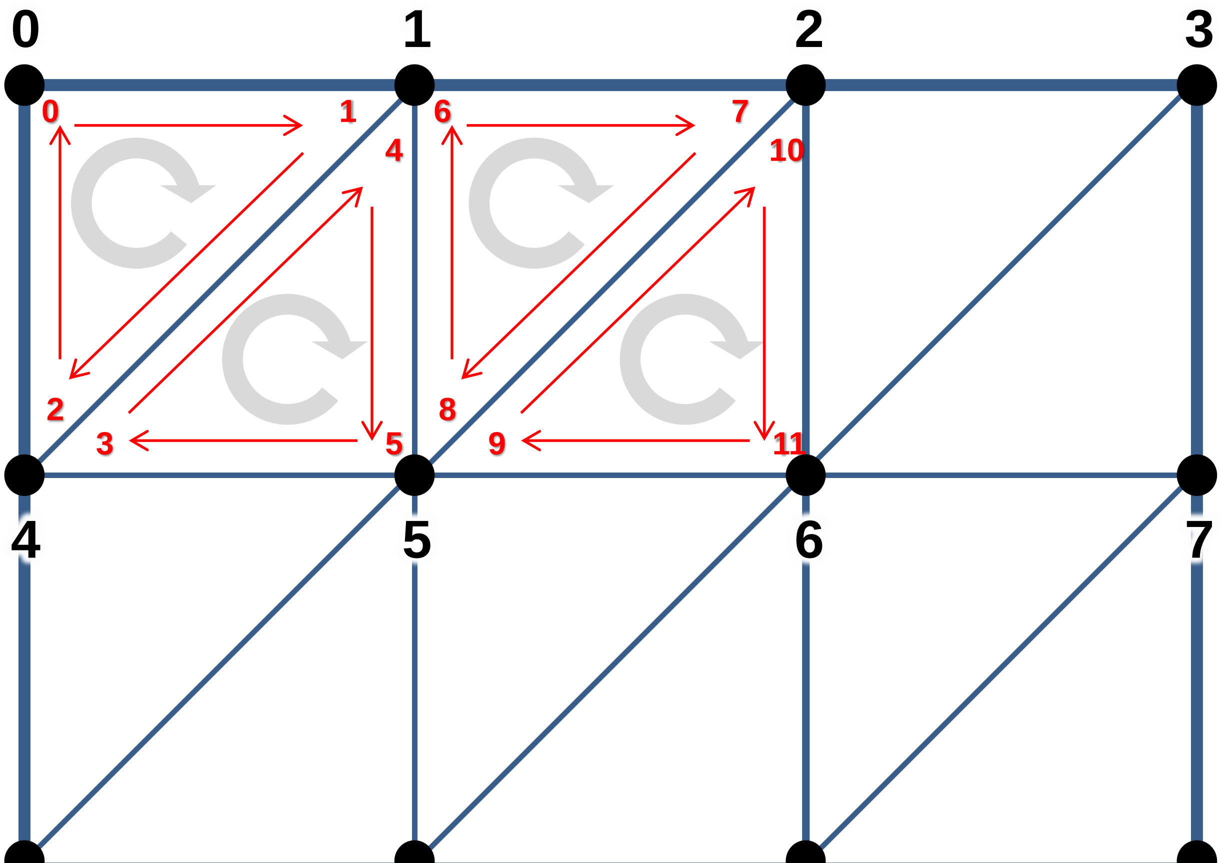
SV_VertexID: 0 1 4 4 1 5 1 2 5 5 2 6 …

Bind **no** vertex buffer, only system-generated values needed

For each Vertex:

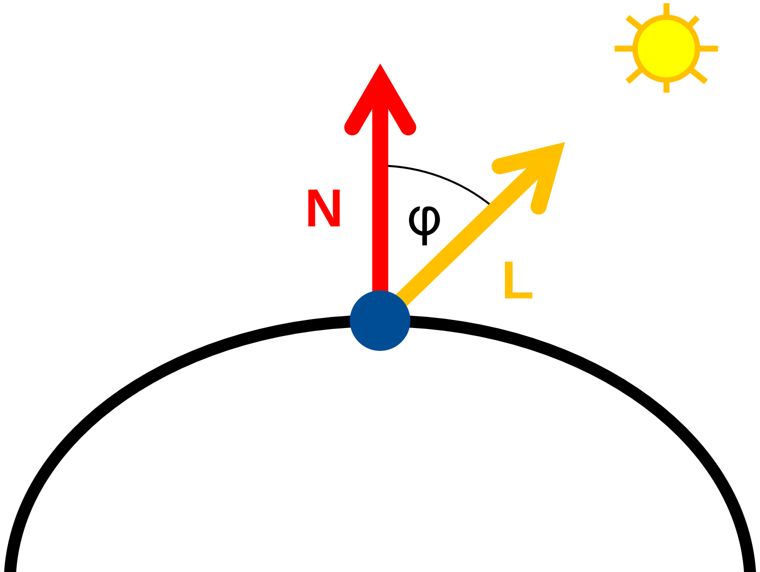
- Calc x / z position
- Look up height value
- Calculate world position
- Transform to clip space

Index buffer stays the same!



$N \cdot L$ Lighting

- Most simple lighting in computer graphics
- More on lighting later this semester
- For now:
  - N is the normal vector of the surface
  - L is the direction vector to the light
  - The scalar product (= dot product) between two normalized vectors yields the cosine of the angle between them



Rendering Effects

- In Direct3D most of the rendering state is specified in a ***Rendering Effect***
- Each rendering effect is stored in a separate effect source code file (.fx)
- The FX compiler ("fxc") converts an effect source code file into a compiled effect file (.fxo)
- The function `D3DX11CreateEffectFromMemory` creates an effect object from a compiled effect in main memory
  - This already happens in the template

Rendering Effects

- A rendering effect can contain multiple ***Effect Techniques*** to achieve the same rendering effect
  - The effect member function `GetTechniqueByName` retrieves a handle to an effect technique
- An effect technique can contain multiple ***Render Passes*** to achieve a certain rendering effect
  - The technique member function `GetPassByName` retrieves a handle to a render pass
- A render pass defines the state of the graphics pipeline during a draw
  - It sets the state of the fixed function stages of the graphics pipeline
  - It controls which ***Shaders*** are used in the programmable stages

Shaders

- Techniques combine multiple rendering passes
  - A rendering pass specifies which shaders to use (vertex, pixel, geometry, …)
  - May also specify pipeline states (Rasterizer, DepthStencil, Blend…)

```
//---------------------------------------------------------------------------------------
// Techniques
//---------------------------------------------------------------------------------------
technique11 Render
{
    pass P0
    {
        SetVertexShader(CompileShader(vs_4_0, TerrainVS()));
        SetGeometryShader(NULL);
        SetPixelShader(CompileShader(ps_4_0, TerrainPS()));

        SetRasterizerState(rsCullNone);
        SetDepthStencilState(EnableDepth, 0);
        SetBlendState(NoBlending, float4(0.0f, 0.0f, 0.0f, 0.0f), 0xFFFFFFFF);
    }
}
```

Shaders

- A shader is a program that is executed on the GPU

- The same shader program is executed for many elements of graphics data in parallel
  - Vertex Shader
  - Pixel Shader
  - etc.
  - SIMD = Single Instruction Multiple Data

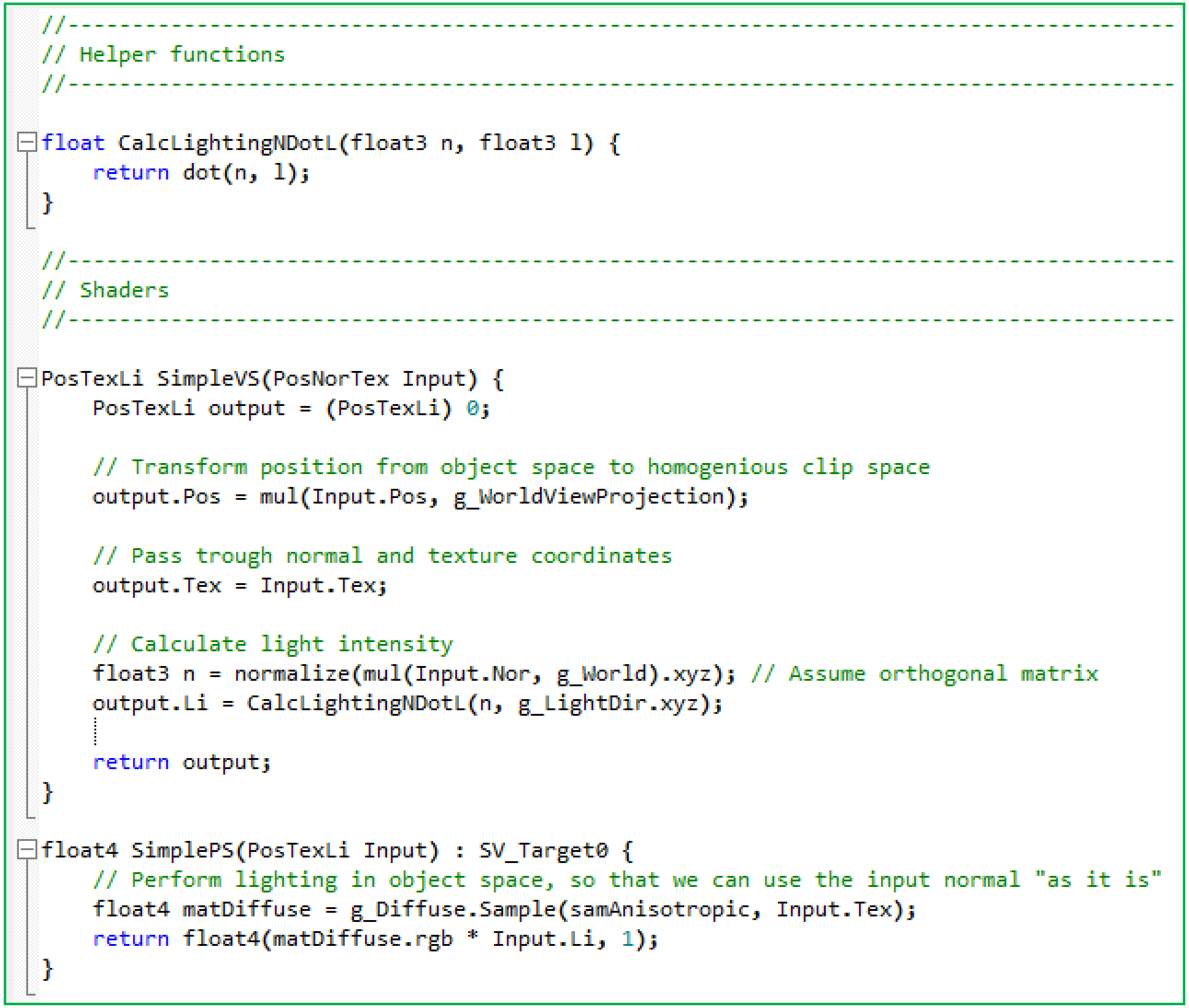
- In Direct3D, shaders are written in HLSL
  (High Level Shading Language)

- Major DirectX versions correspond to major HLSL versions
  (DirectX 10 -> HLSL 4, DirectX 11 -> HLSL 5)

- HLSL Documentation
  - „Programming Guide for HLSL“ and „Reference for HLSL“
    in the „Windows DirectX Graphics Documentation “ help file

Shaders

- In HLSL, shaders are defined very similar to functions



```
//--------------------------------------------------------------------------------------
// Helper functions
//--------------------------------------------------------------------------------------

float CalcLightingNDotL(float3 n, float3 l) {
    return dot(n, l);
}

//--------------------------------------------------------------------------------------
// Shaders
//--------------------------------------------------------------------------------------

PosTexLi SimpleVS(PosNorTex Input) {
    PosTexLi output = (PosTexLi) 0;

    // Transform position from object space to homogenious clip space
    output.Pos = mul(Input.Pos, g_WorldViewProjection);

    // Pass trough normal and texture coordinates
    output.Tex = Input.Tex;

    // Calculate light intensity
    float3 n = normalize(mul(Input.Nor, g_World).xyz); // Assume orthogonal matrix
    output.Li = CalcLightingNDotL(n, g_LightDir.xyz);

    return output;
}

float4 SimplePS(PosTexLi Input) : SV_Target0 {
    // Perform lighting in object space, so that we can use the input normal "as it is"
    float4 matDiffuse = g_Diffuse.Sample(samAnisotropic, Input.Tex);
    return float4(matDiffuse.rgb * Input.Li, 1);
}
```

Shaders

- Syntax similar to C++
- Special vector datatypes: `float2, float4, int2`…
  - Component access `.x, .y, .xyzw`…
  - Swizzling: `.xzy`…
  - Operators +-/*% for vector x vector and scalar x vector
    - int2 v = int2(10, 10) % 5
  - Common vector operations like dot(), cross(), normalize()…
    - Check the documentation before you write your own function!

- Special matrix types: float4x4
  - `mul(float4, float4x4)` to apply matrix transformation to a vector

Shader Semantics

- Data exchange between pipeline stages is controlled by *semantics*

```
float4 MyVertexShader(in float4 pos: POSITION): SV_POSITION
{
	// Just pass the position
	return pos;
}
```

Semantic of the input parameter „pos“

Semantic of the return value

Optional „in“, „out“ or „inout“; „in“ is default

  - A pipeline stage‘s *output* is the next one‘s *input*
  - Semantics are arbitrary, but must match between stages
    - Input assembler must output a float4 with semantic POSITION
    - This is defined in the input layout!
  - Predefined system values SV_*
    - SV_POSITION, SV_TARGET0, SV_VERTEXID…

Shader Semantics

- Alternative 1:

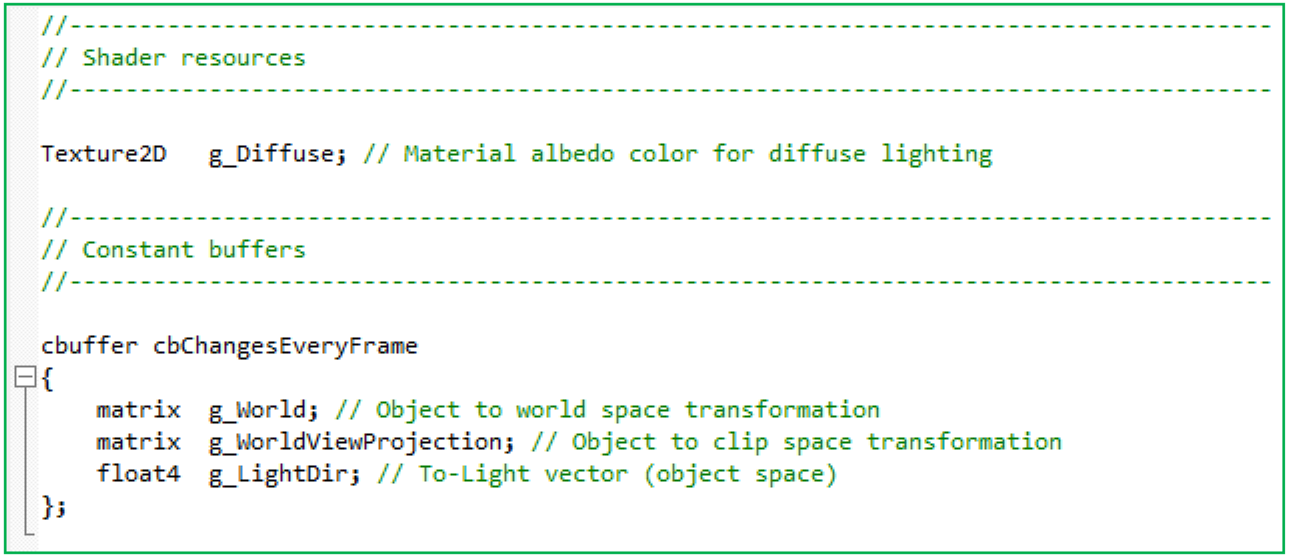
```
void MyVertexShader(in float4 posIn: POSITION, out float4 posOut: SV_POSITION)
{
        // Just pass the position
        posOut = posIn;
}
```

- Alternative 2:

```
struct MyVertex
{
        float4 pos: POSITION;
};

struct MyFragment
{
        float4 pos: SV_POSITION;
};

void MyVertexShader(in MyVertex input, out MyFragment output)
{
        output = (MyFragment)0;

        // Just pass the position
        output.pos = input.pos;
}
```

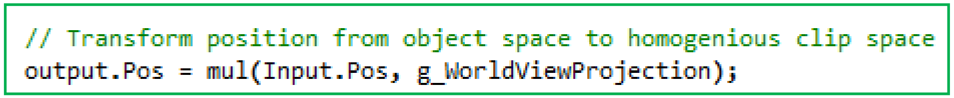
Effect variables

- Effect variables are used to pass information from your C++ CPU code to your HLSL shader code
- In the .fx file on HLSL side, texture and buffer resources are defined as global variables while simpler types are combined to constant buffers



```
//--------------------------------------------------------------------------------------
// Shader resources
//--------------------------------------------------------------------------------------

Texture2D   g_Diffuse; // Material albedo color for diffuse lighting

//--------------------------------------------------------------------------------------
// Constant buffers
//--------------------------------------------------------------------------------------

cbuffer cbChangesEveryFrame
{
    matrix  g_World; // Object to world space transformation
    matrix  g_WorldViewProjection; // Object to clip space transformation
    float4  g_LightDir; // To-Light vector (object space)
};
```

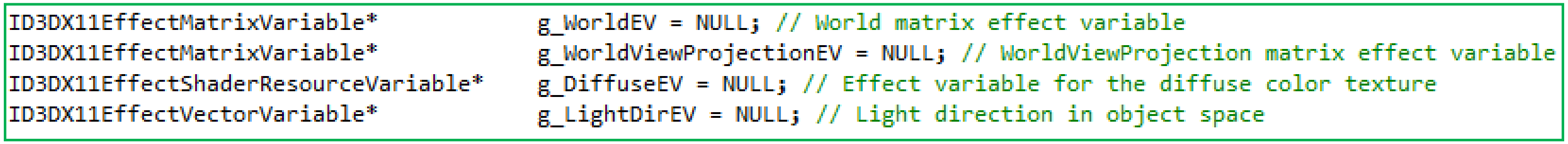
- In the shaders, both types can be accessed like global variables though



```
// Transform position from object space to homogenious clip space
output.Pos = mul(Input.Pos, g_WorldViewProjection);
```

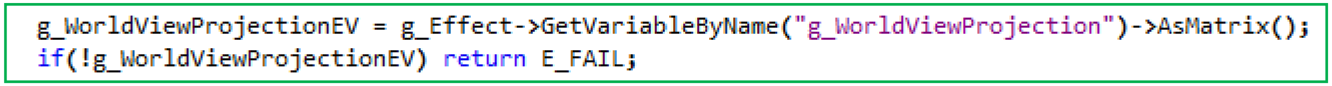
Effect variables

- In the .cpp file on C++ side: **effect variables** act as „pointers“ to the variables in the HLSL shader on the GPU



```
ID3DX11EffectMatrixVariable*           g_WorldEV = NULL; // World matrix effect variable
ID3DX11EffectMatrixVariable*           g_WorldViewProjectionEV = NULL; // WorldViewProjection matrix effect variable
ID3DX11EffectShaderResourceVariable*   g_DiffuseEV = NULL; // Effect variable for the diffuse color texture
ID3DX11EffectVectorVariable*           g_LightDirEV = NULL; // Light direction in object space
```

- The „GetVariableByName“ method of the rendering effect is used to bind the CPU variable to its GPU counterpart

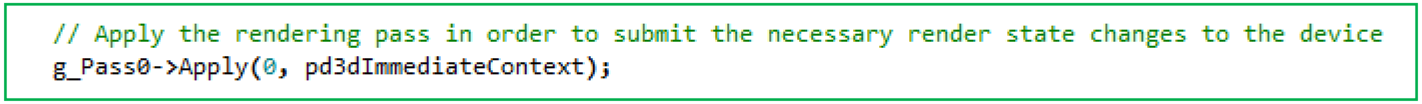


```
g_WorldViewProjectionEV = g_Effect->GetVariableByName("g_WorldViewProjection")->AsMatrix();
if(!g_WorldViewProjectionEV) return E_FAIL;
```

- The „Set*“ method of an effect variable tells the effect framework the updated value for a variable

```
XMFLOAT4X4 w;
XMStoreFloat4x4(&w, worldViewProj);
V(g_gameEffect.worldViewProjectionEV->SetMatrix((float*)&w));
```

- The upload of the updated values to the GPU happens when the rendering pass is applied



```
// Apply the rendering pass in order to submit the necessary render state changes to the device
g_Pass0->Apply(0, pd3dImmediateContext);
```

D3D11_RESOURCE_VIEW_DESC

- Direct3D can create resource views for textures without further information

```
device->CreateShaderResourceView(diffuseTexture, NULL, &diffuseTextureSRV)
```

- This time, we‘ll need to pass a description
  - D3D11_SHADER_RESOURCE_VIEW_DESC
    http://msdn.microsoft.com/en-us/library/windows/desktop/ff476211(v=vs.85).aspx
  - ViewDimension: D3D11_SRV_DIMENSION_BUFFER
  - Format: DXGI_FORMAT_R32_FLOAT
  - Exactly **one** of the remaining structs needs to be filled
    - Buffer in our case
    - FirstElement = 0
    - NumElements = res * res

```
typedef struct D3D11_SHADER_RESOURCE_VIEW_DESC {
  DXGI_FORMAT          Format;
  D3D11_SRV_DIMENSION ViewDimension;
  union {
    D3D11_BUFFER_SRV          Buffer;
    D3D11_TEX1D_SRV           Texture1D;
    D3D11_TEX1D_ARRAY_SRV     Texture1DArray;
    D3D11_TEX2D_SRV           Texture2D;
    D3D11_TEX2D_ARRAY_SRV     Texture2DArray;
    D3D11_TEX2DMS_SRV         Texture2DMS;
    D3D11_TEX2DMS_ARRAY_SRV Texture2DMSArray;
    D3D11_TEX3D_SRV           Texture3D;
    D3D11_TEXCUBE_SRV         TextureCube;
    D3D11_TEXCUBE_ARRAY_SRV TextureCubeArray;
    D3D11_BUFFEREX_SRV        BufferEx;
  };
} D3D11_SHADER_RESOURCE_VIEW_DESC;
```

Shader Editing Tips

- The template project is configured to compile .fx files with FXC when you build the project
  - If your shader doesn‘t compile, the build fails
  - Double-click the error message to jump into the shader file
  - The compiled .fxo effect is loaded when the programm starts
  - You can also compile the .fx file manually (Strg + F7)

- The template project includes a „Reload Shader“ button (F5) which allows shader editing at runtime
  - Loads the compiled .fxo effect at runtime and immediately uses the updated shader for rendering
  - This only works if all effect variables are retrieved in „ReloadShader()“ and updated („Set*()“) after the reload
  - Compiling doesn‘t work during debugging, so start without debugging (Strg+F5) if you work on your shader code

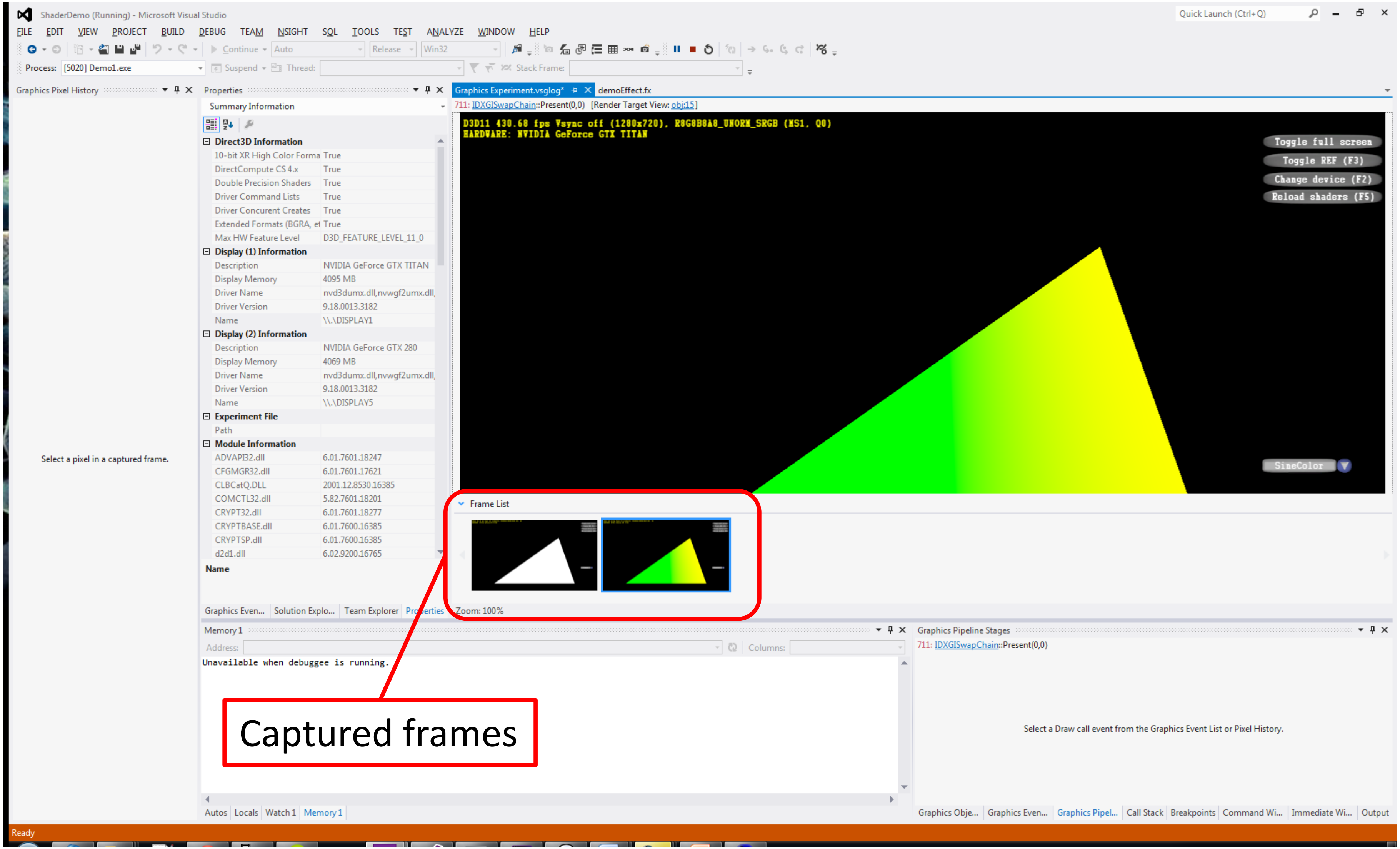
Graphics Debugging

- Visual Studio 2012 contains a Graphics Debugger!

- „Post mortem“ debugging
  - You need to explicitly select a rendered frame for debugging
  - All pipeline states, resources etc. are downloaded to the CPU
  - Each draw call can now be investigated step by step

- Start your program from „Debug -> Graphics -> Start Diagnostics“ (or hit Alt + F5)
  - Press „Print“ to capture the last rendered frame

Graphics Debugging



Process: [5020] Demo1.exe

Select a pixel in a captured frame.

Summary Information

Direct3D Information	
10-bit XR High Color Forma	True
DirectCompute CS 4.x	True
Double Precision Shaders	True
Driver Command Lists	True
Driver Concurent Creates	True
Extended Formats (BGRA, et	True
Max HW Feature Level	D3D_FEATURE_LEVEL_11_0
Display (1) Information	
Description	NVIDIA GeForce GTX TITAN
Display Memory	4095 MB
Driver Name	nvd3dumx.dll,nvwgf2umx.dll,
Driver Version	9.18.0013.3182
Name	\\.\DISPLAY1
Display (2) Information	
Description	NVIDIA GeForce GTX 280
Display Memory	4069 MB
Driver Name	nvd3dumx.dll,nvwgf2umx.dll,
Driver Version	9.18.0013.3182
Name	\\.\DISPLAY5
Experiment File	
Path	
Module Information	
ADVAPI32.dll	6.01.7601.18247
CFGMGR32.dll	6.01.7601.17621
CLBCatQ.DLL	2001.12.8530.16385
COMCTL32.dll	5.82.7601.18201
CRYPT32.dll	6.01.7601.18277
CRYPTBASE.dll	6.01.7600.16385
CRYPTSP.dll	6.01.7600.16385
d2d1.dll	6.02.9200.16765

Name

711: IDXGISwapChain::Present(0,0) [Render Target View: obj:15]

```
D3D11 430.68 fps Vsync off (1280x720), R8G8B8A8_UNORM_SRGB (MS1, Q0)
HARDWARE: NVIDIA GeForce GTX TITAN
```

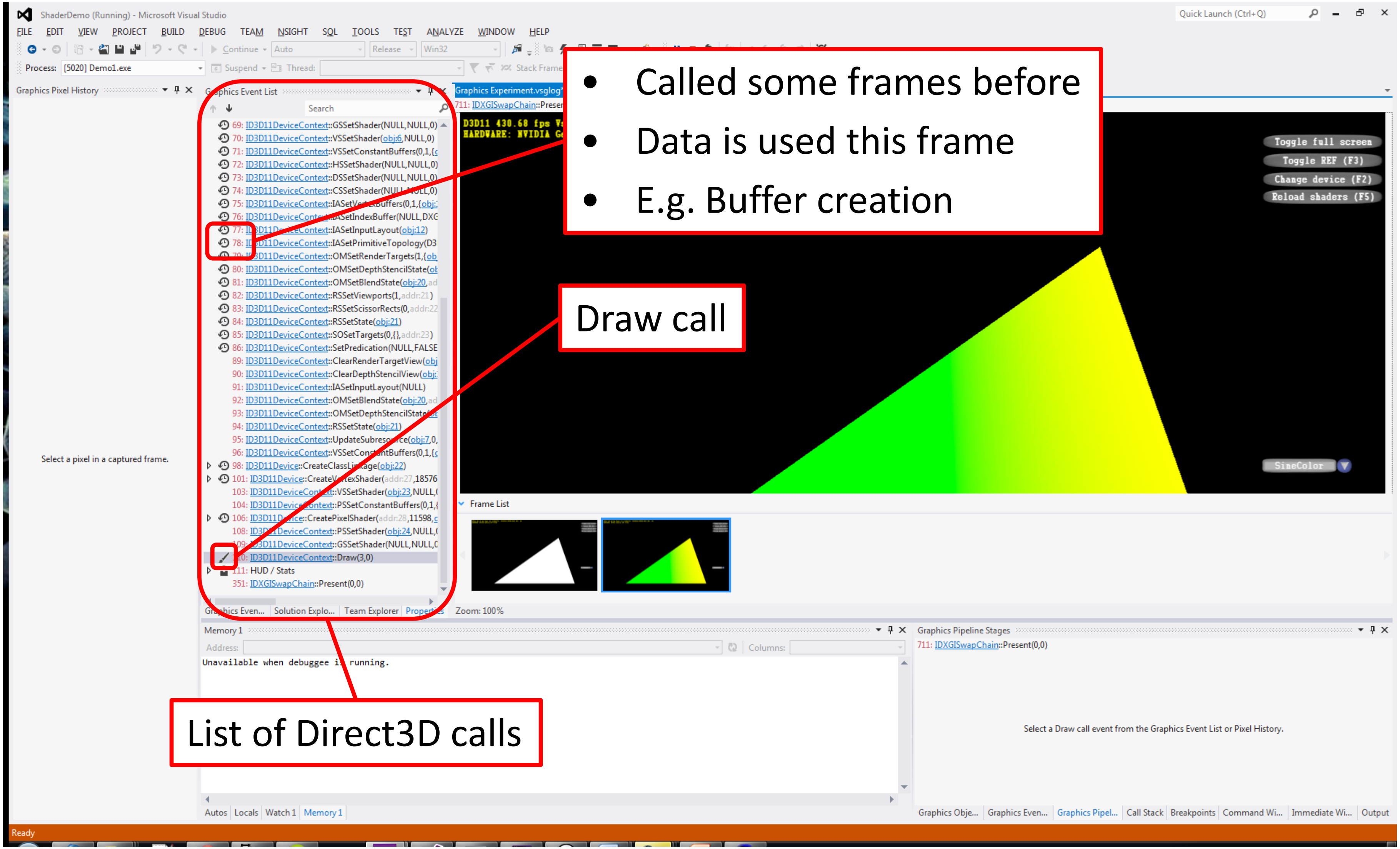
```
Unavailable when debuggee is running.
```

711: IDXGISwapChain::Present(0,0)

Select a Draw call event from the Graphics Event List or Pixel History.

Captured frames

Graphics Debugging



```
69: ID3D11DeviceContext::GSSetShader(NULL,NULL,0)
70: ID3D11DeviceContext::VSSetShader(obj:6,NULL,0)
71: ID3D11DeviceContext::VSSetConstantBuffers(0,1,{
72: ID3D11DeviceContext::HSSetShader(NULL,NULL,0)
73: ID3D11DeviceContext::DSSetShader(NULL,NULL,0)
74: ID3D11DeviceContext::CSSetShader(NULL,NULL,0)
75: ID3D11DeviceContext::IASetVertexBuffers(0,1,{obj:
76: ID3D11DeviceContext::IASetIndexBuffer(NULL,DXG
77: ID3D11DeviceContext::IASetInputLayout(obj:12)
78: ID3D11DeviceContext::IASetPrimitiveTopology(D3
79: ID3D11DeviceContext::OMSetRenderTargets(1,{ob
80: ID3D11DeviceContext::OMSetDepthStencilState(ob
81: ID3D11DeviceContext::OMSetBlendState(obj:20,ad
82: ID3D11DeviceContext::RSSetViewports(1,addr:21)
83: ID3D11DeviceContext::RSSetScissorRects(0,addr:22
84: ID3D11DeviceContext::RSSetState(obj:21)
85: ID3D11DeviceContext::SOSetTargets(0,{},addr:23)
86: ID3D11DeviceContext::SetPredication(NULL,FALSE
89: ID3D11DeviceContext::ClearRenderTargetView(obj
90: ID3D11DeviceContext::ClearDepthStencilView(obj:
91: ID3D11DeviceContext::IASetInputLayout(NULL)
92: ID3D11DeviceContext::OMSetBlendState(obj:20,ad
93: ID3D11DeviceContext::OMSetDepthStencilState(ob
94: ID3D11DeviceContext::RSSetState(obj:21)
95: ID3D11DeviceContext::UpdateSubresource(obj:7,0,
96: ID3D11DeviceContext::VSSetConstantBuffers(0,1,{
98: ID3D11Device::CreateClassLinkage(obj:22)
101: ID3D11Device::CreateVertexShader(addr:27,18576
103: ID3D11DeviceContext::VSSetShader(obj:23,NULL,
104: ID3D11DeviceContext::PSSetConstantBuffers(0,1,
106: ID3D11Device::CreatePixelShader(addr:28,11598,
108: ID3D11DeviceContext::PSSetShader(obj:24,NULL,
109: ID3D11DeviceContext::GSSetShader(NULL,NULL,
110: ID3D11DeviceContext::Draw(3,0)
111: HUD / Stats
351: IDXGISwapChain::Present(0,0)
```

Select a pixel in a captured frame.

```
Unavailable when debuggee is running.
```

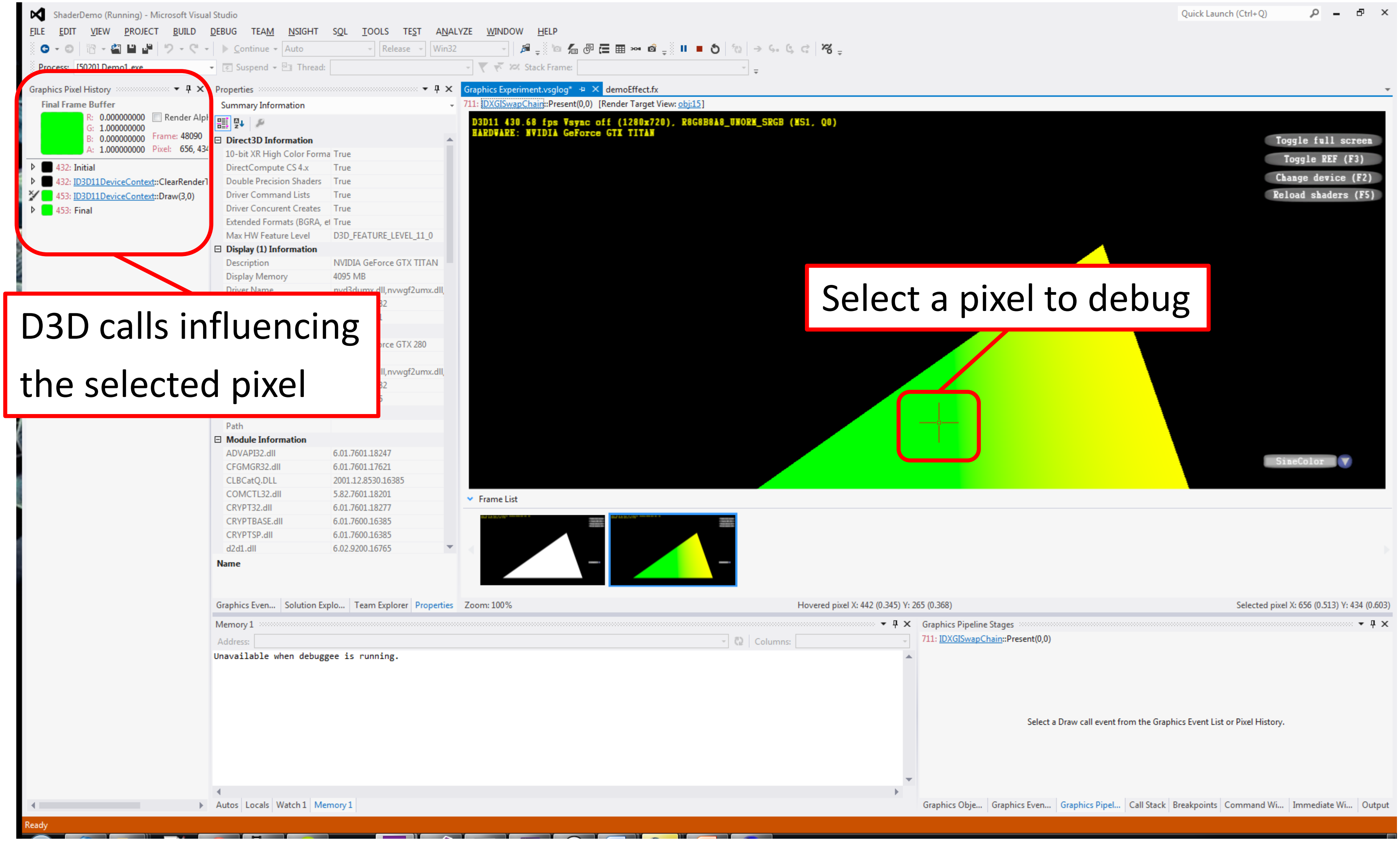
Select a Draw call event from the Graphics Event List or Pixel History.

- Called some frames before
- Data is used this frame
- E.g. Buffer creation

Draw call

List of Direct3D calls

Graphics Debugging



Final Frame Buffer

R: 0.000000000
G: 1.000000000
B: 0.000000000
A: 1.000000000

☐ Render Alph

Frame: 48090
Pixel: 656, 434

- 432: Initial
- 432: ID3D11DeviceContext::ClearRenderT
- 453: ID3D11DeviceContext::Draw(3,0)
- 453: Final

Summary Information

Direct3D Information	
10-bit XR High Color Forma	True
DirectCompute CS 4.x	True
Double Precision Shaders	True
Driver Command Lists	True
Driver Concurrent Creates	True
Extended Formats (BGRA, et	True
Max HW Feature Level	D3D_FEATURE_LEVEL_11_0
Display (1) Information	
Description	NVIDIA GeForce GTX TITAN
Display Memory	4095 MB
Driver Name	nvd3dumx.dll,nvwgf2umx.dll,
Path	
Module Information	
ADVAPI32.dll	6.01.7601.18247
CFGMGR32.dll	6.01.7601.17621
CLBCatQ.DLL	2001.12.8530.16385
COMCTL32.dll	5.82.7601.18201
CRYPT32.dll	6.01.7601.18277
CRYPTBASE.dll	6.01.7600.16385
CRYPTSP.dll	6.01.7600.16385
d2d1.dll	6.02.9200.16765

Name

711: IDXGISwapChain::Present(0,0) [Render Target View: obj:15]

D3D11 430.68 fps Vsync off (1280x720), R8G8B8A8_UNORM_SRGB (MS1, Q0)
HARDWARE: NVIDIA GeForce GTX TITAN

Zoom: 100%

Hovered pixel X: 442 (0.345) Y: 265 (0.368)

Selected pixel X: 656 (0.513) Y: 434 (0.603)

Unavailable when debuggee is running.

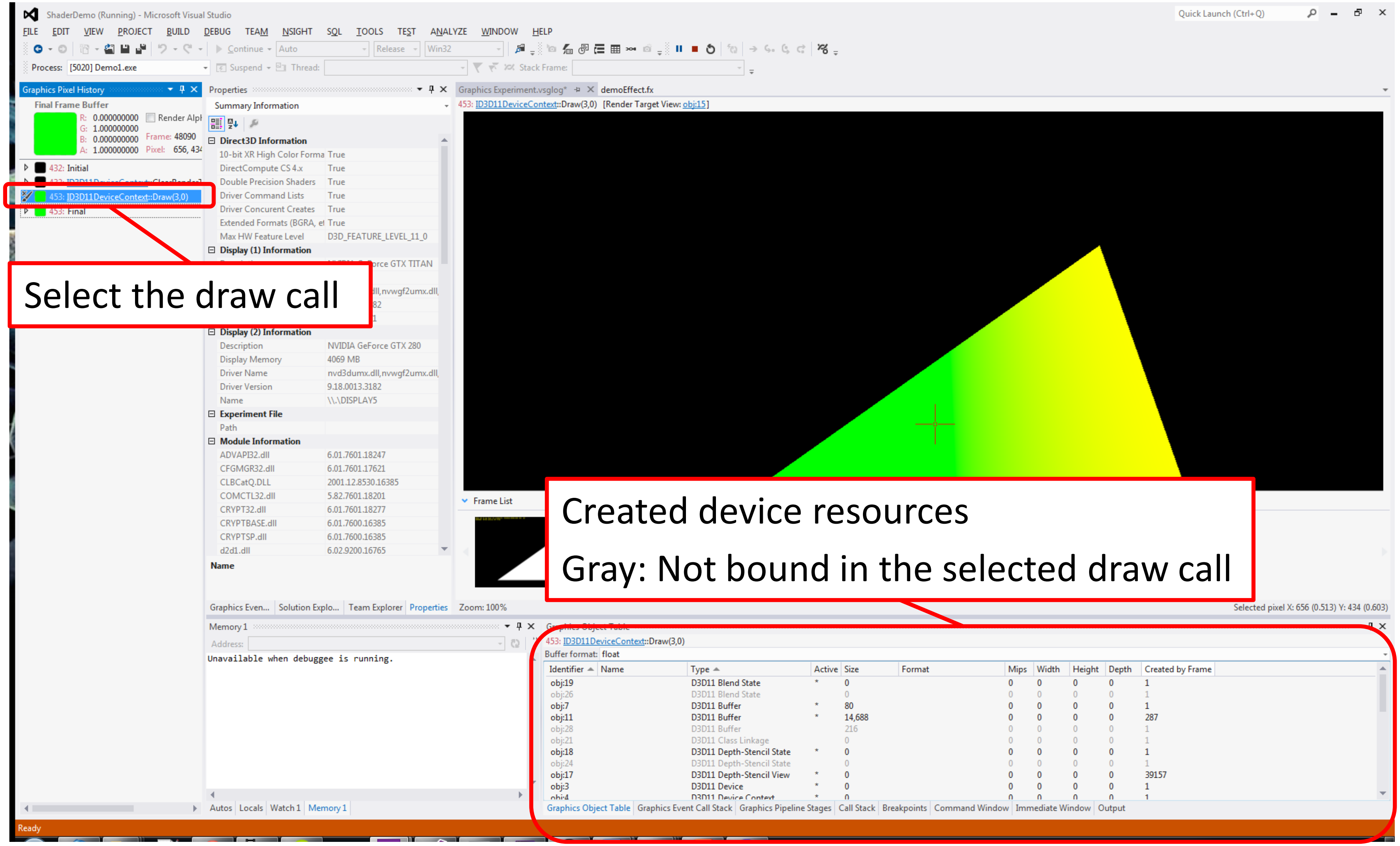
711: IDXGISwapChain::Present(0,0)

Select a Draw call event from the Graphics Event List or Pixel History.

D3D calls influencing the selected pixel

Select a pixel to debug

Graphics Debugging



ShaderDemo (Running) - Microsoft Visual Studio

Graphics Pixel History

Final Frame Buffer

R: 0.000000000
G: 1.000000000
B: 0.000000000
A: 1.000000000

Frame: 48090
Pixel: 656, 434

- 432: Initial
- 453: ID3D11DeviceContext::Draw(3,0)
- 453: Final

Select the draw call

Properties – Summary Information

Direct3D Information

10-bit XR High Color Forma	True
DirectCompute CS 4.x	True
Double Precision Shaders	True
Driver Command Lists	True
Driver Concurent Creates	True
Extended Formats (BGRA, et	True
Max HW Feature Level	D3D_FEATURE_LEVEL_11_0

Display (2) Information

Description	NVIDIA GeForce GTX 280
Display Memory	4069 MB
Driver Name	nvd3dumx.dll,nvwgf2umx.dll,
Driver Version	9.18.0013.3182
Name	\\.\DISPLAY5

Module Information

ADVAPI32.dll	6.01.7601.18247
CFGMGR32.dll	6.01.7601.17621
CLBCatQ.DLL	2001.12.8530.16385
COMCTL32.dll	5.82.7601.18201
CRYPT32.dll	6.01.7601.18277
CRYPTBASE.dll	6.01.7600.16385
CRYPTSP.dll	6.01.7600.16385
d2d1.dll	6.02.9200.16765

453: ID3D11DeviceContext::Draw(3,0) [Render Target View: obj:15]

Created device resources

Gray: Not bound in the selected draw call

Selected pixel X: 656 (0.513) Y: 434 (0.603)

Memory 1: Unavailable when debuggee is running.

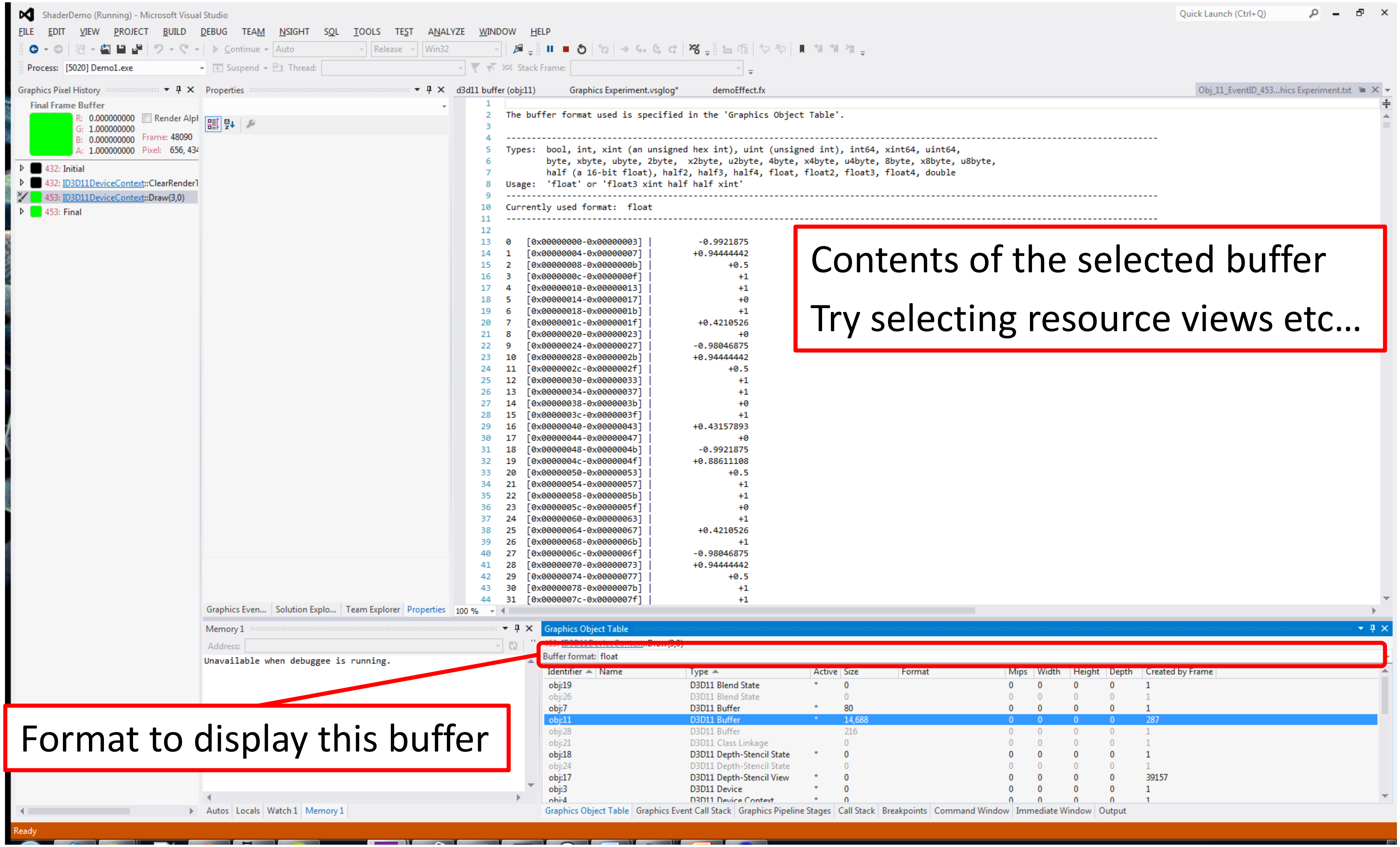
Graphics Object Table

453: ID3D11DeviceContext::Draw(3,0)

Buffer format: float

Identifier	Name	Type	Active	Size	Format	Mips	Width	Height	Depth	Created by Frame
obj:19		D3D11 Blend State	*	0		0	0	0	0	1
obj:26		D3D11 Blend State		0		0	0	0	0	1
obj:7		D3D11 Buffer	*	80		0	0	0	0	1
obj:11		D3D11 Buffer	*	14,688		0	0	0	0	287
obj:28		D3D11 Buffer		216		0	0	0	0	1
obj:21		D3D11 Class Linkage		0		0	0	0	0	1
obj:18		D3D11 Depth-Stencil State	*	0		0	0	0	0	1
obj:24		D3D11 Depth-Stencil State		0		0	0	0	0	1
obj:17		D3D11 Depth-Stencil View	*	0		0	0	0	0	39157
obj:3		D3D11 Device	*	0		0	0	0	0	1
obj:4		D3D11 Device Context	*	0		0	0	0	0	1

Graphics Debugging



Final Frame Buffer

R: 0.000000000
G: 1.000000000
B: 0.000000000
A: 1.000000000

☐ Render Alpha

Frame: 48090
Pixel: 656, 434

Address:

```
Unavailable when debuggee is running.
```

```

The buffer format used is specified in the 'Graphics Object Table'.

----------------------------------------------------------------------------------------------------
Types:  bool, int, xint (an unsigned hex int), uint (unsigned int), int64, xint64, uint64,
        byte, xbyte, ubyte, 2byte,  x2byte, u2byte, 4byte, x4byte, u4byte, 8byte, x8byte, u8byte,
        half (a 16-bit float), half2, half3, half4, float, float2, float3, float4, double
Usage:  'float' or 'float3 xint half half xint'
----------------------------------------------------------------------------------------------------
Currently used format:  float
----------------------------------------------------------------------------------------------------

0   [0x00000000-0x00000003] |        -0.9921875
1   [0x00000004-0x00000007] |       +0.94444442
2   [0x00000008-0x0000000b] |              +0.5
3   [0x0000000c-0x0000000f] |                +1
4   [0x00000010-0x00000013] |                +1
5   [0x00000014-0x00000017] |                +0
6   [0x00000018-0x0000001b] |                +1
7   [0x0000001c-0x0000001f] |        +0.4210526
8   [0x00000020-0x00000023] |                +0
9   [0x00000024-0x00000027] |       -0.98046875
10  [0x00000028-0x0000002b] |       +0.94444442
11  [0x0000002c-0x0000002f] |              +0.5
12  [0x00000030-0x00000033] |                +1
13  [0x00000034-0x00000037] |                +1
14  [0x00000038-0x0000003b] |                +0
15  [0x0000003c-0x0000003f] |                +1
16  [0x00000040-0x00000043] |       +0.43157893
17  [0x00000044-0x00000047] |                +0
18  [0x00000048-0x0000004b] |        -0.9921875
19  [0x0000004c-0x0000004f] |       +0.88611108
20  [0x00000050-0x00000053] |              +0.5
21  [0x00000054-0x00000057] |                +1
22  [0x00000058-0x0000005b] |                +1
23  [0x0000005c-0x0000005f] |                +0
24  [0x00000060-0x00000063] |                +1
25  [0x00000064-0x00000067] |        +0.4210526
26  [0x00000068-0x0000006b] |                +1
27  [0x0000006c-0x0000006f] |       -0.98046875
28  [0x00000070-0x00000073] |       +0.94444442
29  [0x00000074-0x00000077] |              +0.5
30  [0x00000078-0x0000007b] |                +1
31  [0x0000007c-0x0000007f] |                +1
```

Buffer format: float

Identifier	Name	Type	Active	Size	Format	Mips	Width	Height	Depth	Created by Frame
obj:19		D3D11 Blend State	*	0		0	0	0	0	1
obj:26		D3D11 Blend State		0		0	0	0	0	1
obj:7		D3D11 Buffer	*	80		0	0	0	0	1
obj:11		D3D11 Buffer	*	14,688		0	0	0	0	287
obj:28		D3D11 Buffer		216		0	0	0	0	1
obj:21		D3D11 Class Linkage		0		0	0	0	0	1
obj:18		D3D11 Depth-Stencil State	*	0		0	0	0	0	1
obj:24		D3D11 Depth-Stencil State		0		0	0	0	0	1
obj:17		D3D11 Depth-Stencil View	*	0		0	0	0	0	39157
obj:3		D3D11 Device	*	0		0	0	0	0	1
obj:4		D3D11 Device Context	*	0		0	0	0	0	1

Contents of the selected buffer

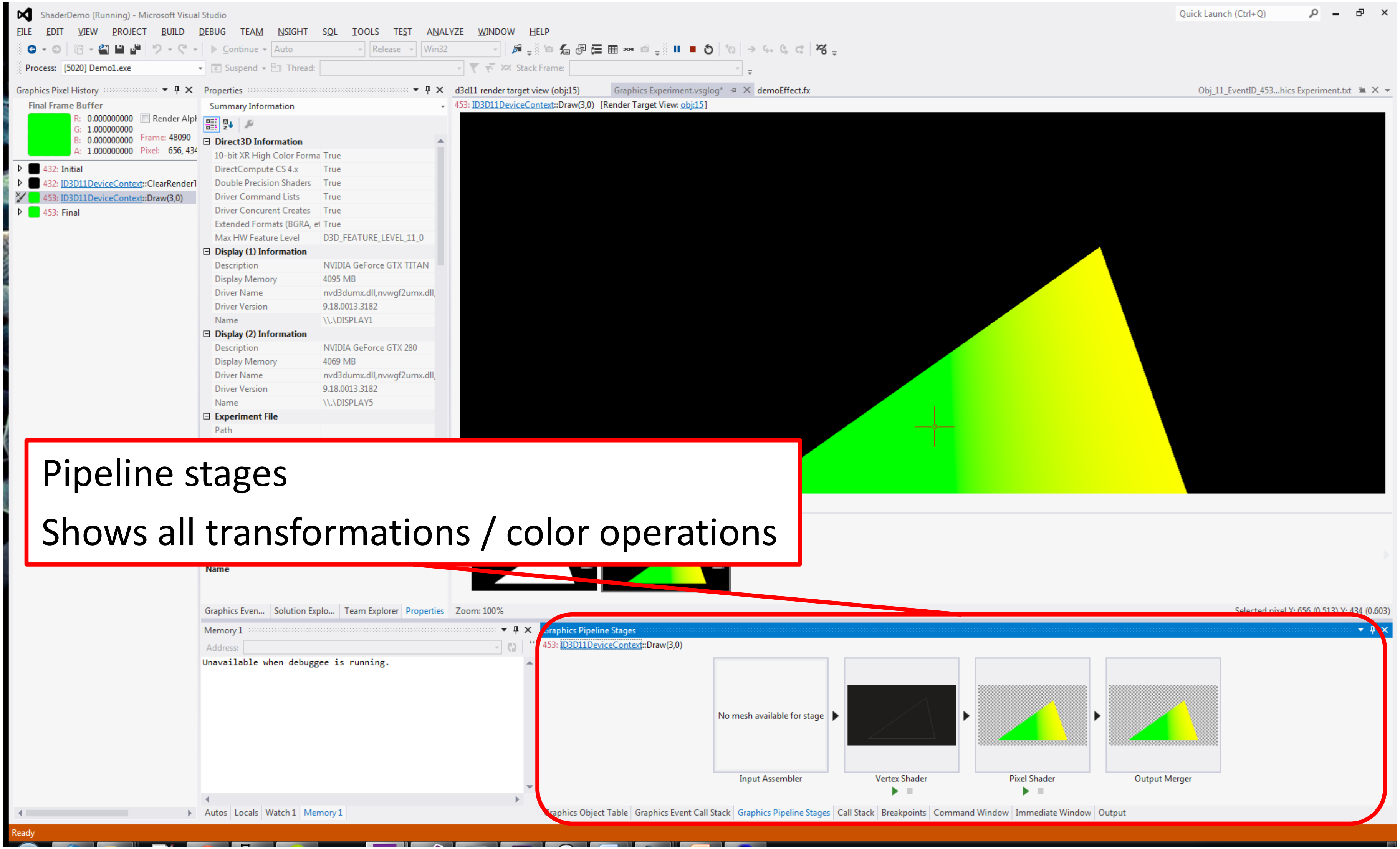
Try selecting resource views etc…

Format to display this buffer

Graphics Debugging

Pipeline stages

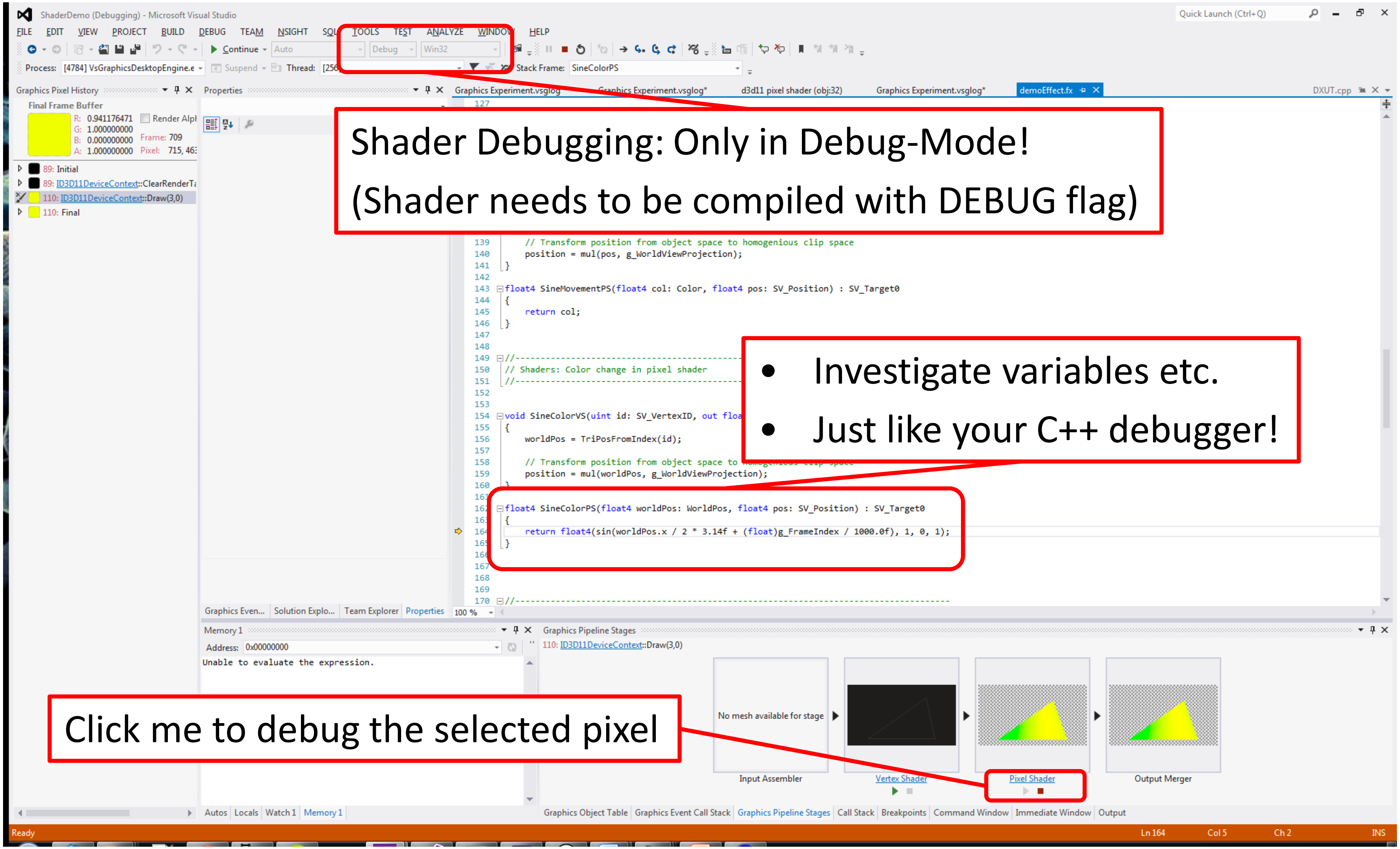
Shows all transformations / color operations



Direct3D Information	
10-bit XR High Color Forma	True
DirectCompute CS 4.x	True
Double Precision Shaders	True
Driver Command Lists	True
Driver Concurent Creates	True
Extended Formats (BGRA, et	True
Max HW Feature Level	D3D_FEATURE_LEVEL_11_0
Display (1) Information	
Description	NVIDIA GeForce GTX TITAN
Display Memory	4095 MB
Driver Name	nvd3dumx.dll,nvwgf2umx.dll,
Driver Version	9.18.0013.3182
Name	\\.\DISPLAY1
Display (2) Information	
Description	NVIDIA GeForce GTX 280
Display Memory	4069 MB
Driver Name	nvd3dumx.dll,nvwgf2umx.dll,
Driver Version	9.18.0013.3182
Name	\\.\DISPLAY5
Experiment File	
Path	

```
Unavailable when debuggee is running.
```

Graphics Debugging



```
    // Transform position from object space to homogenious clip space
    position = mul(pos, g_WorldViewProjection);
}

float4 SineMovementPS(float4 col: Color, float4 pos: SV_Position) : SV_Target0
{
    return col;
}


//--------------------------------------------
// Shaders: Color change in pixel shader
//--------------------------------------------


void SineColorVS(uint id: SV_VertexID, out floa
{
    worldPos = TriPosFromIndex(id);

    // Transform position from object space to
    position = mul(worldPos, g_WorldViewProjection);
}

float4 SineColorPS(float4 worldPos: WorldPos, float4 pos: SV_Position) : SV_Target0
{
    return float4(sin(worldPos.x / 2 * 3.14f + (float)g_FrameIndex / 1000.0f), 1, 0, 1);
}



//--------------------------------------------------------------------------------
```

```
Unable to evaluate the expression.
```

Shader Debugging: Only in Debug-Mode!

(Shader needs to be compiled with DEBUG flag)

- Investigate variables etc.
- Just like your C++ debugger!

Click me to debug the selected pixel

Shader Playground

Git: external/examples/shader_demo

- Simple demo for Vertex / Pixel shaders
- Test the shader debugger
- Try to modify and reload the shaders

Q&A

Questions?