

Visual Studio's Graphics Analyzer

AKA The Graphics Debugger

VS Graphics Analyzer

- ▶ Allows the capture and analysis (debugging) of individual frames of a DirectX application
- ▶ Including actual shader debugging!
 - Full step-over/into/out
 - Locals window with variables' contents
- ▶ Extremely useful for debugging DX drawing
 - Especially if objects aren't drawing at all

Graphics Analyzer overview

Report20181101-0747.vsglog - Visual Studio Graphics Analyzer

File Edit View Debug Window Help

Event List View: GPU Work Search

- 201: obj:3->UpdateSubresource(obj:10)
- 202: obj:3->UpdateSubresource(obj:12)
- 209: obj:3->DrawIndexed(4560,0,0)
- 216: obj:3->UpdateSubresource(obj:10)
- 217: obj:3->UpdateSubresource(obj:12)
- 224: obj:3->DrawIndexed(4560,0,0)
- 231: obj:3->UpdateSubresource(obj:10)
- 232: obj:3->UpdateSubresource(obj:12)
- 239: obj:3->DrawIndexed(4560,0,0)
- 246: obj:3->UpdateSubresource(obj:10)
- 247: obj:3->UpdateSubresource(obj:12)
- 254: obj:3->DrawIndexed(4560,0,0)
- 284: obj:3->DrawIndexed(1818,7272,0)
- 288: obj:1->Present(0,0)=S_OK

Event Call Stack

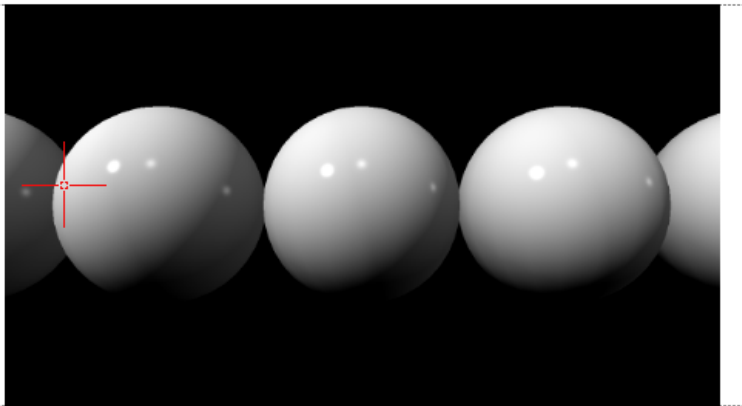
254: obj:3->DrawIndexed(4560,0,0)

Name

- DX11Starter.exe!Game::Draw
- DX11Starter.exe!DXCore::Run
- DX11Starter.exe!WinMain
- DX11Starter.exe!invoke_main
- DX11Starter.exe!_srt_common_main_seh
- DX11Starter.exe!_srt_common_main
- DX11Starter.exe!WinMainCRTStartup
- KERNEL32.DLL!13034
- ntdll.dll!71551

Render Target Frame Analysis

[RTVs: obj:5] [Mode: Normal] 254: obj:3->DrawIndexed(4560,0,0)



Frame List (1/1) Playback Machine: MediaCenter

Zoom % 36 Selected Pixel 107,323 R:0.855 G:0.855 B:0.855 A:1

State

Name Value

- * D3D11DeviceContext
- = Input Assembler
- = Vertex Shader
- Hull Shader
- Domain Shader

Pixel History

Final Pixel Color

R: 0.854901969 G: 0.854901969 B: 0.854901969 A: 1

Frame: 28 Pixel: 10

Initial

- 147: obj:3->ClearRenderTargetView
- 179: obj:3->DrawIndexed(4560,0,0)
- 194: obj:3->DrawIndexed(4560,0,0)
- 284: obj:3->DrawIndexed(1818,7272,0)
- Final

State Pinned State Object Table Pipeline Stages

Ready

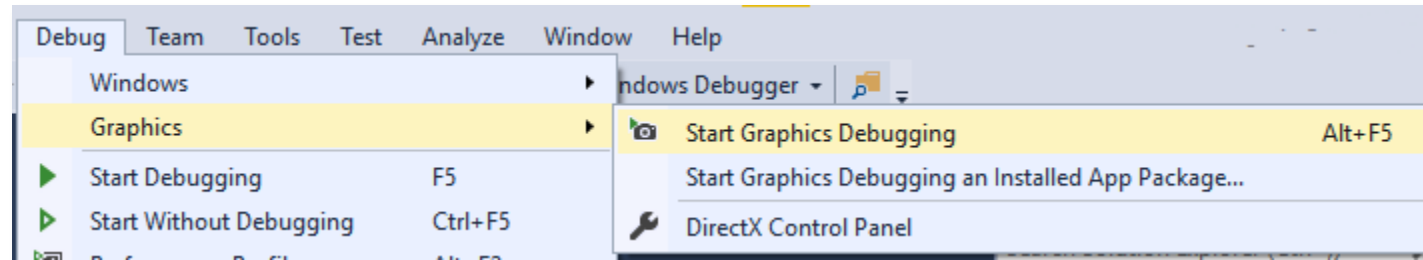
Graphics Analyzer features

- ▶ **Event List** – lists all DirectX API calls made
- ▶ **Event Call Stack** – shows *where* DX calls made
- ▶ **State** – Lists entire DX state for current event
- ▶ **Object Table** – Lists every live DX resource
- ▶ **Render Target** – Shows current output
- ▶ **Pixel History** – Lists changes to selected pixel
- ▶ **Pipeline Stages** – Visualizes each pipeline step

- ▶ **Shader debugging**
 - Can select individual verts/pixels to “debug”

Step 1: Start Graphics Debugging

- ▶ Shortcut is Alt+F5

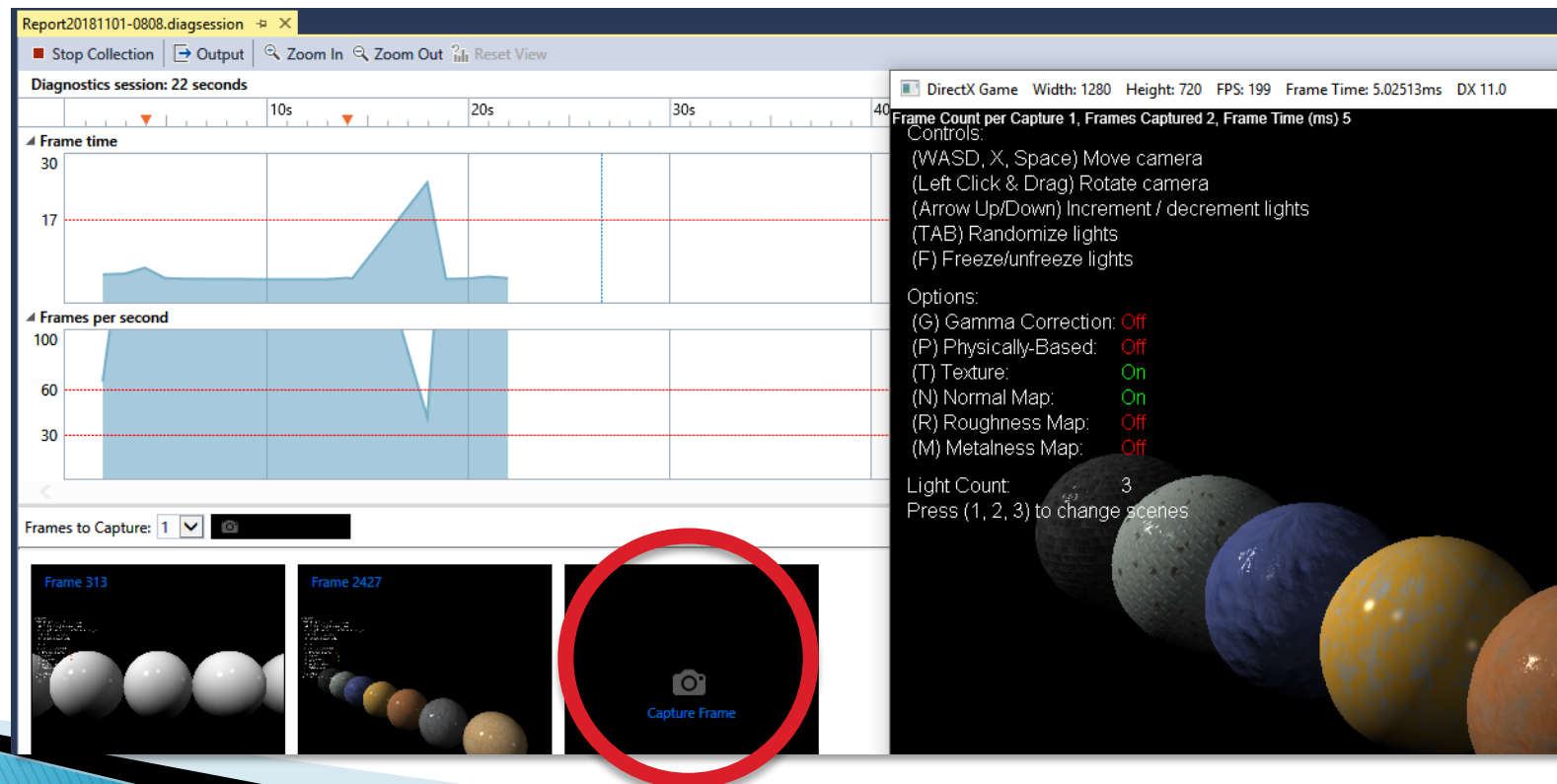


- ▶ Allow VS to capture data (every time!)



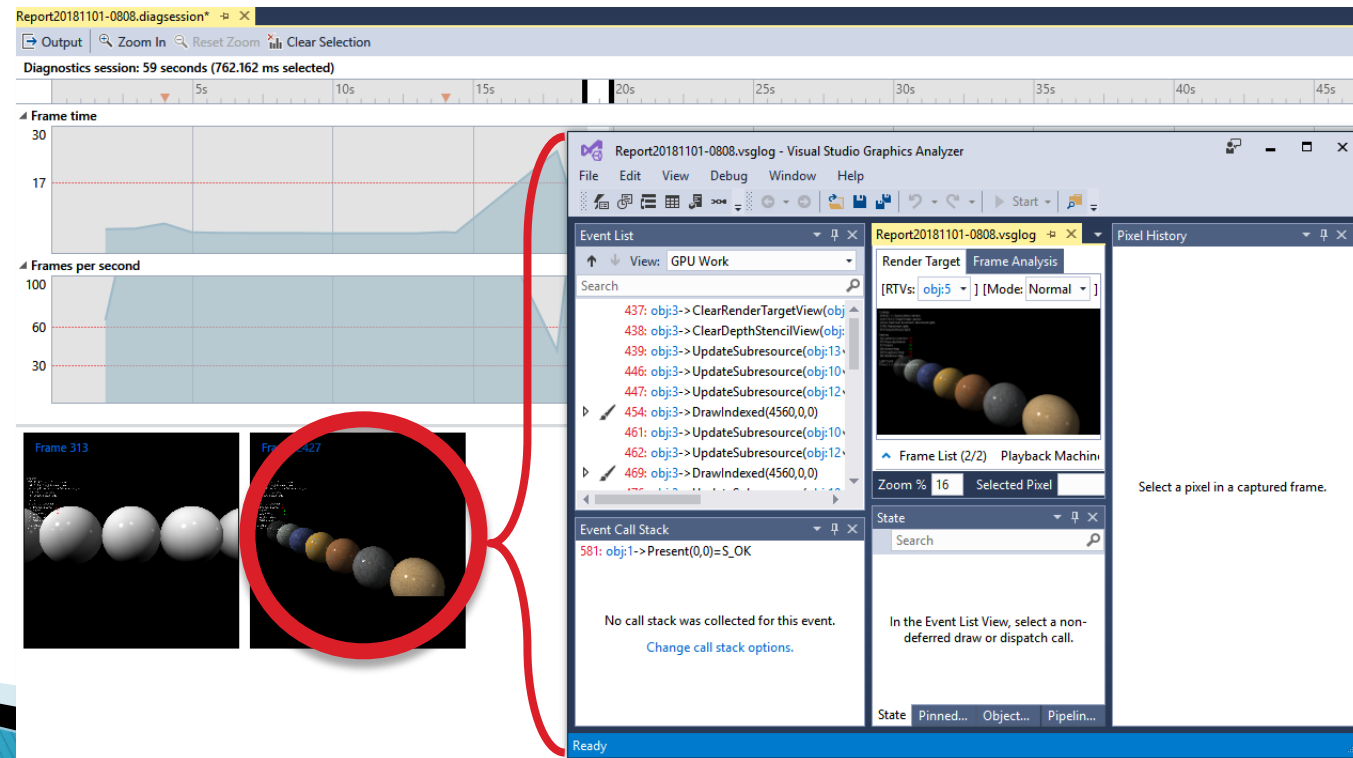
Step 2: Capture frame(s)

- ▶ Press “Print Screen” key
 - Or “Capture Frame” button in Visual Studio



Step 3: Pick a frame to analyze

- ▶ 3a: Close your program
- ▶ 3b: Double click on a captured frame
 - This opens the Graphics Analyzer window



Step 4: Dig through your frame

The screenshot displays the Visual Studio Graphics Analyzer interface, titled "Report20181112-2046.vsglog - Visual Studio Graphics Analyzer". The interface is divided into several panels:

- Event List:** A list of GPU events. The event "549: obj:3->DrawIndexed(4560,0,0)" is selected and highlighted in blue.
- Render Target / Frame Analysis:** Shows the current render target and frame analysis. It includes a 3D visualization of a scene with a yellow and orange curved surface. Below the visualization, it displays "Frame List (2/2)" and "Playback Machine: DESKTOP-V768QQV".
- Pixel History:** A panel showing the "Final Pixel Color" and a list of events. The "Final Pixel Color" is displayed as a black square with the following values: R: 0.0196078438, G: 0.0156862754, B: 0.0117647061, A: 1. The "Pixel History" list shows a sequence of events, with "549: obj:3->DrawIndexed(4560,0,0)" selected.
- Event Call Stack:** A panel showing the call stack for the selected event. It displays "549: obj:3->DrawIndexed(4560,0,0)" and a message: "No call stack was collected for this event. Change call stack options."
- State:** A panel showing the state of the GPU. It includes a search bar and a table with columns "Name" and "Value". The table lists various GPU state elements, including "D3D11DeviceContext", "Input Assembler", "Vertex Shader", "Hull Shader", "Domain Shader", "Geometry Shader", "Rasterizer", "Compute Shader", "Pixel Shader", and "Output Merger".

The bottom status bar indicates "Ready".

Event List

Report20181112-2045.vsglog - Visual Studio Graphics Analyzer

File Edit View Debug Window Help

Event List View: GPU Work Search

- 493: obj:3->UpdateSubresource(obj:12)
- 501: obj:3->DrawIndexed(4560,0,0)
- 508: obj:3->UpdateSubresource(obj:10)
- 509: obj:3->UpdateSubresource(obj:12)
- 517: obj:3->DrawIndexed(4560,0,0)
- 524: obj:3->UpdateSubresource(obj:10)
- 525: obj:3->UpdateSubresource(obj:12)
- 533: obj:3->DrawIndexed(4560,0,0)
- 540: obj:3->UpdateSubresource(obj:10)
- 541: obj:3->UpdateSubresource(obj:12)
- 549: obj:3->DrawIndexed(4560,0,0)
- 556: obj:3->UpdateSubresource(obj:10)
- 557: obj:3->UpdateSubresource(obj:12)
- 565: obj:3->DrawIndexed(4560,0,0)
- 572: obj:3->UpdateSubresource(obj:10)
- 573: obj:3->UpdateSubresource(obj:12)
- 581: obj:3->DrawIndexed(4560,0,0)
- 589: obj:3->UpdateSubresource(obj:10)

Event Call Stack

548: obj:3->DrawIndexed(4560,0,0)

No call stack was collected for this event.

Change call stack options

State Pinned State Object Table Pipeline Stages

Ready

A timeline of DirectX API calls.

Allows you to “back up” and see a frame being built, one call at a time. Just click the desired event.

Two event list views:

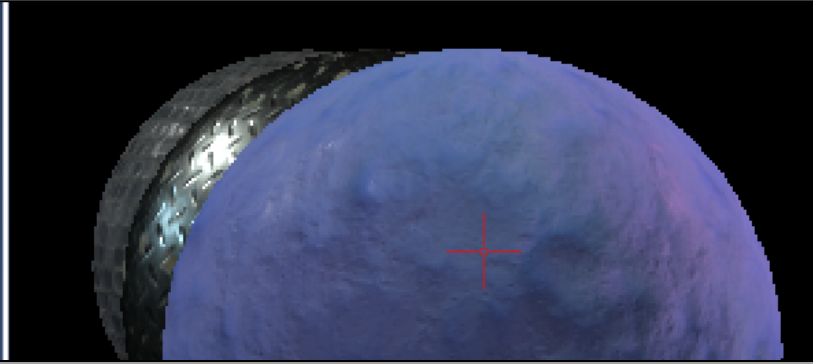
- GPU Work
 - Data updates
 - Draw calls
- Timeline
 - All DX calls

Event List – Timeline traversal

Event
517



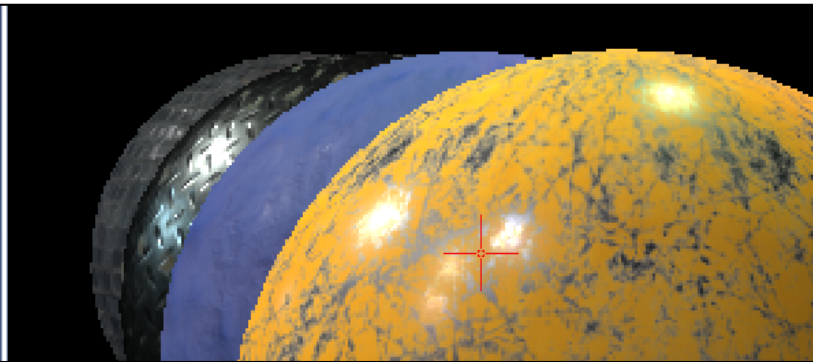
```
507: obj:3->DrawIndexed(4560,0,0)
508: obj:3->UpdateSubresource(obj:10
509: obj:3->UpdateSubresource(obj:12
517: obj:3->DrawIndexed(4560,0,0)
524: obj:3->UpdateSubresource(obj:10
525: obj:3->UpdateSubresource(obj:12
533: obj:3->DrawIndexed(4560,0,0)
540: obj:3->UpdateSubresource(obj:10
541: obj:3->UpdateSubresource(obj:12
549: obj:3->DrawIndexed(4560,0,0)
556: obj:3->UpdateSubresource(obj:10
557: obj:3->UpdateSubresource(obj:12
565: obj:3->DrawIndexed(4560,0,0)
572: obj:3->UpdateSubresource(obj:10
```



Event
533



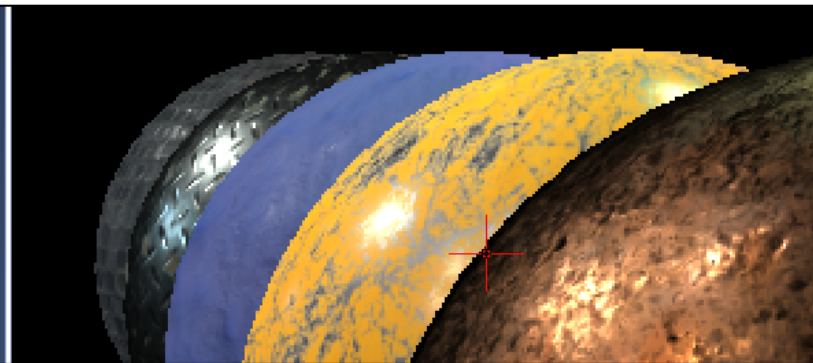
```
507: obj:3->DrawIndexed(4560,0,0)
508: obj:3->UpdateSubresource(obj:10
509: obj:3->UpdateSubresource(obj:12
517: obj:3->DrawIndexed(4560,0,0)
524: obj:3->UpdateSubresource(obj:10
525: obj:3->UpdateSubresource(obj:12
533: obj:3->DrawIndexed(4560,0,0)
540: obj:3->UpdateSubresource(obj:10
541: obj:3->UpdateSubresource(obj:12
549: obj:3->DrawIndexed(4560,0,0)
556: obj:3->UpdateSubresource(obj:10
557: obj:3->UpdateSubresource(obj:12
565: obj:3->DrawIndexed(4560,0,0)
572: obj:3->UpdateSubresource(obj:10
```



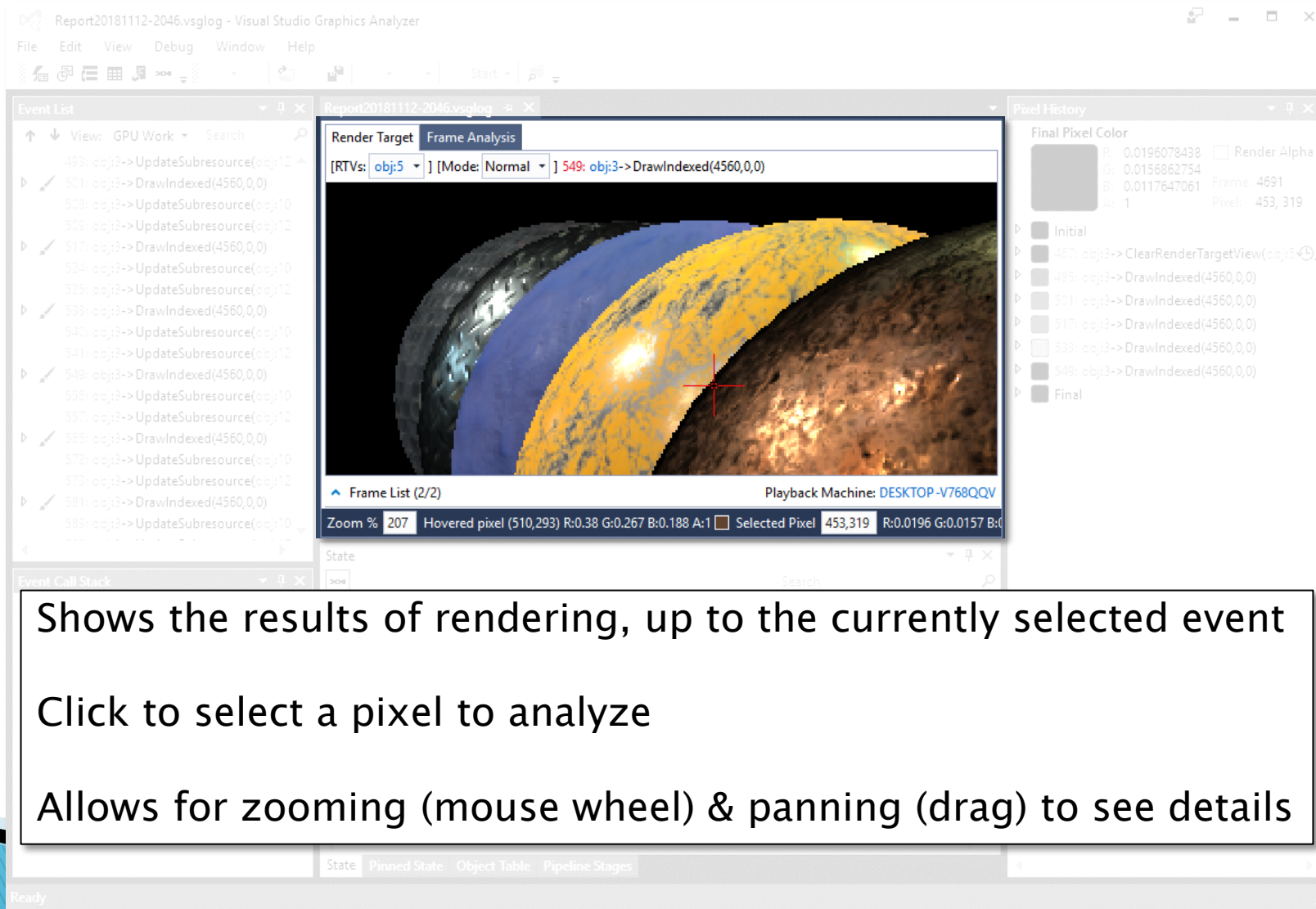
Event
549



```
507: obj:3->DrawIndexed(4560,0,0)
508: obj:3->UpdateSubresource(obj:10
509: obj:3->UpdateSubresource(obj:12
517: obj:3->DrawIndexed(4560,0,0)
524: obj:3->UpdateSubresource(obj:10
525: obj:3->UpdateSubresource(obj:12
533: obj:3->DrawIndexed(4560,0,0)
540: obj:3->UpdateSubresource(obj:10
541: obj:3->UpdateSubresource(obj:12
549: obj:3->DrawIndexed(4560,0,0)
556: obj:3->UpdateSubresource(obj:10
557: obj:3->UpdateSubresource(obj:12
565: obj:3->DrawIndexed(4560,0,0)
572: obj:3->UpdateSubresource(obj:10
```



Render Target



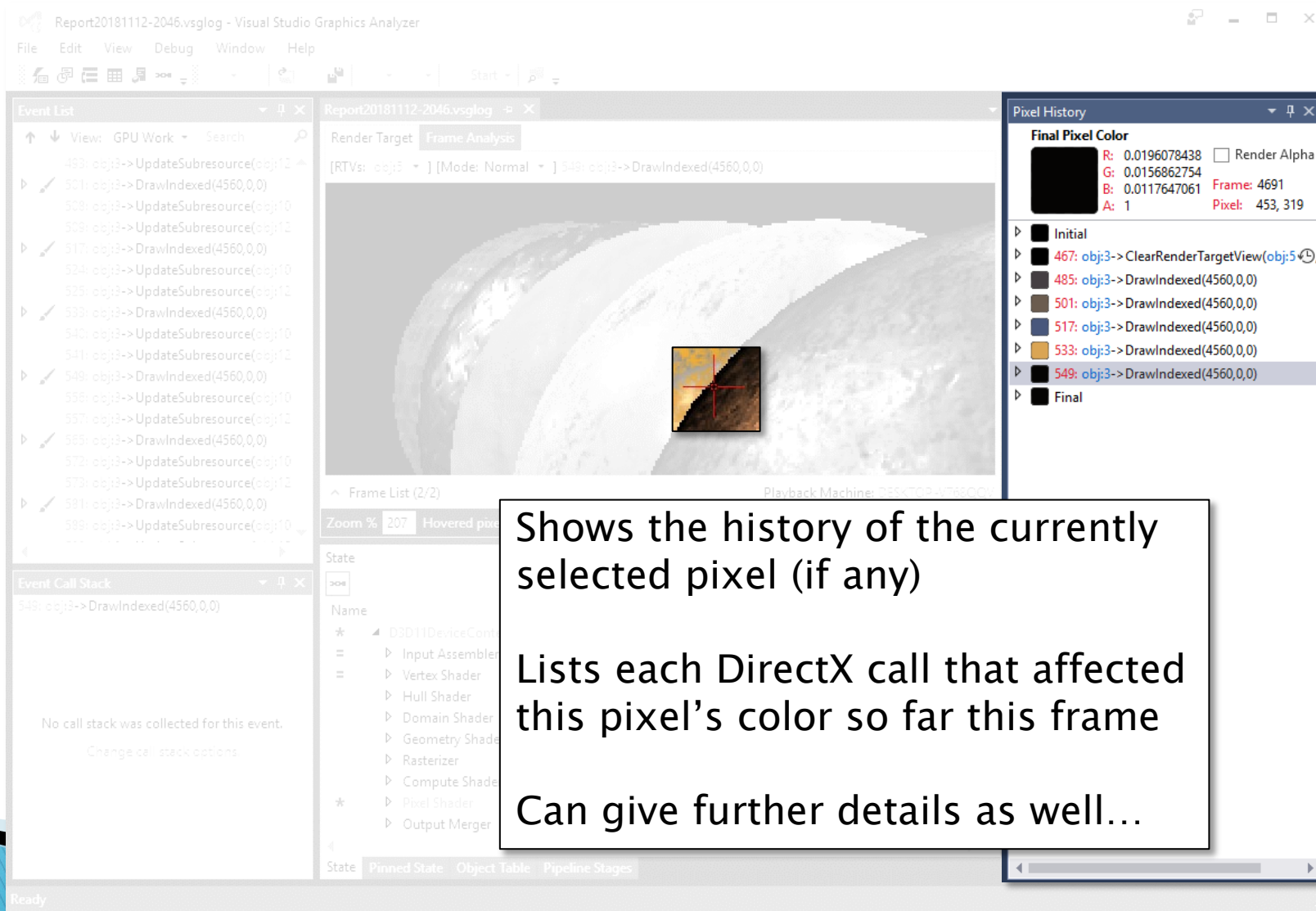
The screenshot displays the Visual Studio Graphics Analyzer interface. The central 'Render Target' window shows a 3D scene with a yellow and orange curved surface. A red crosshair is positioned over a pixel. The 'Event List' on the left shows a sequence of GPU events, with 'obj:3->DrawIndexed(4560,0,0)' selected. The 'Pixel History' on the right shows the 'Final Pixel Color' as R: 0.0196078438, G: 0.0156862754, B: 0.0117647061, A: 1. The 'Frame List' at the bottom indicates the current frame is 2/2. The 'Zoom %' is set to 207, and the 'Hovered pixel' coordinates are (510,293) with RGB values R:0.38, G:0.267, B:0.188, A:1. The 'Selected Pixel' coordinates are (453,319) with RGB values R:0.0196, G:0.0157, B:0.0118, A:1.

Shows the results of rendering, up to the currently selected event

Click to select a pixel to analyze

Allows for zooming (mouse wheel) & panning (drag) to see details

Pixel History



The screenshot displays the Visual Studio Graphics Analyzer interface. The main window shows a 3D scene with a red crosshair highlighting a pixel. The **Pixel History** panel on the right lists the sequence of DirectX calls that affected the selected pixel. The **Event List** panel on the left shows a log of all events, including `obj3->DrawIndexed(4560,0,0)`. The **Event Call Stack** panel at the bottom left shows the call stack for the selected event.

Pixel History

Final Pixel Color

R: 0.0196078438 G: 0.0156862754 B: 0.0117647061 A: 1

Frame: 4691 Pixel: 453, 319

- Initial
- 467: obj3->ClearRenderTargetView(obj5)
- 485: obj3->DrawIndexed(4560,0,0)
- 501: obj3->DrawIndexed(4560,0,0)
- 517: obj3->DrawIndexed(4560,0,0)
- 533: obj3->DrawIndexed(4560,0,0)
- 549: obj3->DrawIndexed(4560,0,0)
- Final

Event List

483: obj3->UpdateSubresource(obj12)
521: obj3->DrawIndexed(4560,0,0)
528: obj3->UpdateSubresource(obj12)
529: obj3->UpdateSubresource(obj12)
517: obj3->DrawIndexed(4560,0,0)
524: obj3->UpdateSubresource(obj12)
525: obj3->UpdateSubresource(obj12)
533: obj3->DrawIndexed(4560,0,0)
540: obj3->UpdateSubresource(obj12)
541: obj3->UpdateSubresource(obj12)
548: obj3->DrawIndexed(4560,0,0)
555: obj3->UpdateSubresource(obj12)
557: obj3->UpdateSubresource(obj12)
565: obj3->DrawIndexed(4560,0,0)
572: obj3->UpdateSubresource(obj12)
573: obj3->UpdateSubresource(obj12)
581: obj3->DrawIndexed(4560,0,0)
588: obj3->UpdateSubresource(obj12)

Event Call Stack

548: obj3->DrawIndexed(4560,0,0)

No call stack was collected for this event.
Change call stack options.

State

State Pinned State Object Table Pipeline Stages

Ready

Shows the history of the currently selected pixel (if any)

Lists each DirectX call that affected this pixel's color so far this frame

Can give further details as well...

Pixel History – Details

- ▶ Expand a call for details
- ▶ Triangles rasterized here
 - Output of pixel shader
 - Result of O.M. stage
- ▶ Triangles that were culled
 - Not drawn for some reason

The screenshot shows the 'Pixel History' window with a yellow title bar. It displays a list of rendering calls, each with a color swatch and a label. The calls are: Initial (black), 467: obj:3->ClearRenderTargetView(obj:5, {0,0,0,0}) (black), 485: obj:3->DrawIndexed(4560,0,0) (dark gray), 501: obj:3->DrawIndexed(4560,0,0) (brown), 517: obj:3->DrawIndexed(4560,0,0) (dark blue), 533: obj:3->DrawIndexed(4560,0,0) (orange, selected), and 549: obj:3->DrawIndexed(4560,0,0) (black). The selected call (533) is expanded, showing 'Vertices: 4560, Primitives: 1520'. Below this, a tree view shows 'Triangle[513]' expanded, containing 'Vertex Shader obj:8' and 'Pixel Shader obj:11'. The 'Output Merger' section shows the 'Pixel Shader Output (SRC)' with values R: 0.854901969, G: 0.647058845, B: 0.329411775, A: 1, and the 'Render Target (DEST)' with values R: 0.282352954, G: 0.34117648, B: 0.486274511, A: 1. The 'Result' is shown as a color swatch with values R: 0.854901969, G: 0.647058845, B: 0.329411775, A: 1. The window also shows 'Triangle[1273]' and 'Final' at the bottom.

Pixel History – More details

- ▶ Expand “Vertex Shader” for individual vertices
 - And their data
- ▶ Click green “play button” to **debug a shader**
 - Line by line debugging
 - For a particular vertex
 - Or this very pixel

The screenshot displays a graphics debugger interface with the following structure:

- 533: obj:3->DrawIndexed(4560,0,0)**
Vertices: 4560, Primitives: 1520
 - Triangle[513]**
 - Vertex Shader obj:8**
 - Vertex 1539**
 - POSITION x=0.237764, y=0.293893, z=-0.327254
 - TEXCOORD x=0.700032, y=0.3
 - NORMAL x=0.478652, y=0.580401, z=-0.658808
 - TANGENT x=0.7743912, y=0.07454433, z=0.6283005
 - Vertex 1540**
 - POSITION x=0.327254, y=0.293893, z=-0.237764
 - TEXCOORD x=0.800032, y=0.3
 - NORMAL x=0.658808, y=0.580401, z=-0.478652
 - TANGENT x=0.6283005, y=-0.07454433, z=0.7743912
 - Vertex 1541**
 - POSITION x=0.36042, y=0.226995, z=-0.26186
 - TEXCOORD x=0.800032, y=0.35
 - NORMAL x=0.723291, y=0.447994, z=-0.525502
 - TANGENT x=0.6119635, y=-0.06327742, z=0.7883506
 - Pixel Shader obj:11**
 - Output Merger**
 - Pixel Shader Output (SRC)**
 - R: 0.854901969
 - G: 0.647058845
 - B: 0.329411775
 - A: 1
 - Render Target (DEST)**
 - R: 0.282352954
 - G: 0.34117648
 - B: 0.486274511
 - A: 1
 - Result**
 - R: 0.854901969
 - G: 0.647058845
 - B: 0.329411775
 - A: 1

At the bottom, a play button icon is shown next to **Triangle[1273]**.

State

The screenshot displays the Visual Studio Graphics Analyzer interface. A central white box with a black border contains the text "Shows current DirectX state info, including:" followed by a bulleted list:

- Render state options
 - Primitive topology
 - Cull mode
 - Etc.
- Currently bound shaders & resources

The background shows the Graphics Analyzer's Event List, Frame List, and Event Call Stack. The State window is open, showing a tree view of the current DirectX state, including D3D11DeviceContext, Input Assembler, Vertex Shader, Hull Shader, Domain Shader, Geometry Shader, Rasterizer, Compute Shader, Pixel Shader, and Output Merger.

Object Table

Lists every active DirectX resource, including:

- Textures
- Shader Resource Views
- Samplers
- Rasterizer States
- Shaders
- Etc.

Double click on a resource to view it

The screenshot shows the Visual Studio interface with the Object Table window open. The Object Table lists the following resources:

Name	Type	Active	Size	Format
obj:14	D3D11 Pixel Shader	*	12,432	ps_5_0
DirectXTK:CommonS	D3D11 Rasterizer State	*		
obj:5	D3D11 Render Target View	*		R8G8B8A8_UNORM
DirectXTK:CommonS	D3D11 Sampler State	*		
obj:76	D3D11 Sampler State	*		
DirectXTK:SpriteFont	D3D11 Shader Resource View	*		BC2_UNORM
Textures/bronze_albe	D3D11 Shader Resource View	*		B8G8R8A8_UNORM
Textures/bronze_metal	D3D11 Shader Resource View	*		B8G8R8A8_UNORM_SRC
Textures/bronze_norm	D3D11 Shader Resource View	*		B8G8R8A8_UNORM
Textures/bronze_rough	D3D11 Shader Resource View	*		R8_UNORM

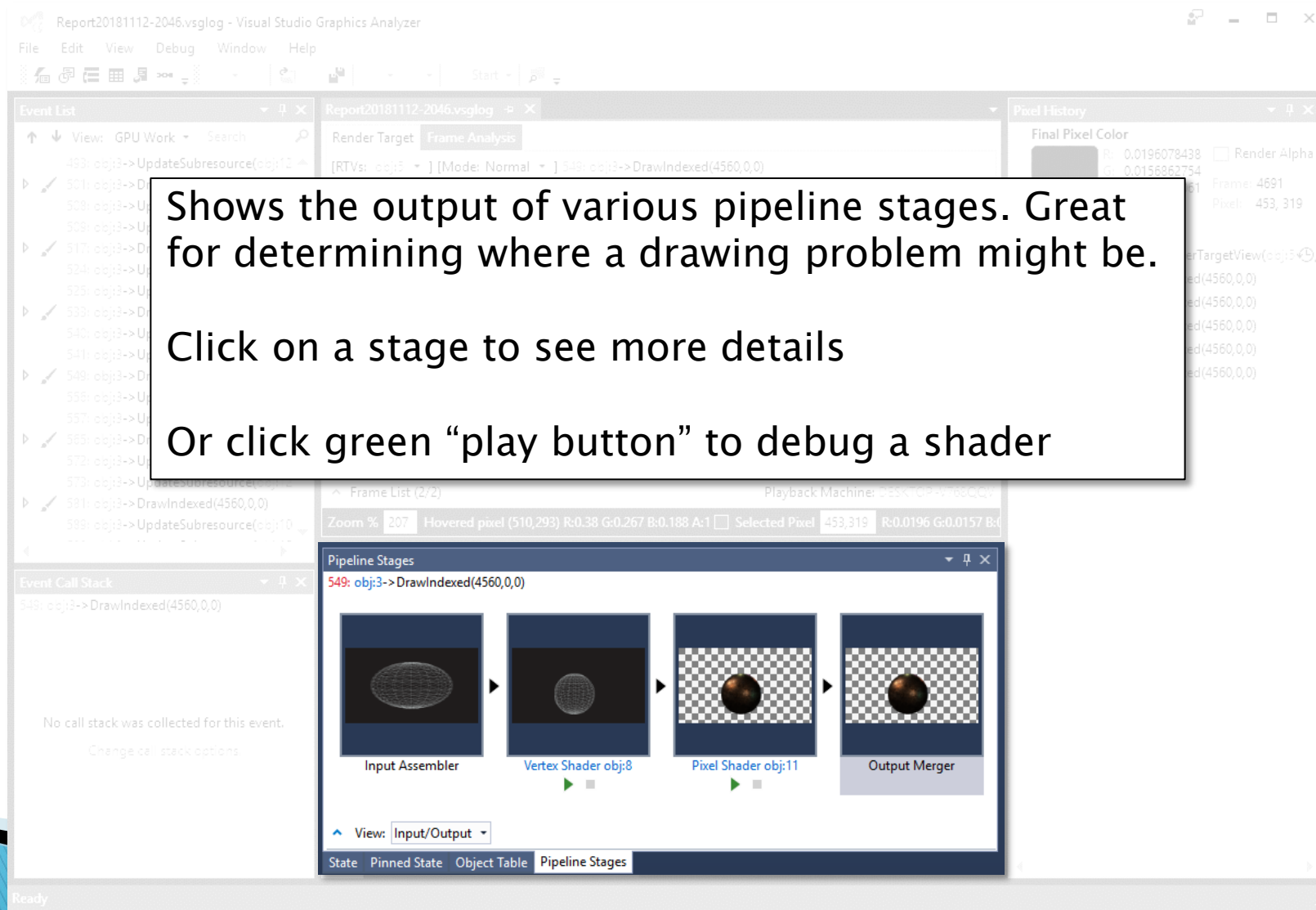
Object Table – Viewing a resource

- ▶ Double click the depth buffer to actually see it
 - Format is D24...
 - D for Depth

The screenshot shows the Visual Studio Graphics tool interface. At the top, the window title is 'Report20181112-2046.vsglog'. The main view displays a red square with a black circle in the center, representing a depth buffer. To the right of this view is a 'Channel Histogram' section with checkboxes for 'D' (checked) and 'S' (checked), and a histogram showing a distribution of values. Below the histogram are 'Texture Properties' for 'DXGI_FORMAT_D24_UNORM_S8_UINT' with a width of 1280 and height of 720. At the bottom is the 'Object Table' with a search bar and a list of resources. A blue arrow points from the 'obj:6' entry in the Object Table to the depth buffer visualization.

Name	Type	Active	Size	Format
Textures/wood_albed	D3D11 Shader Resource	new		B8G8R8A8_UNORM
Textures/wood_metal	D3D11 Shader Resource	new		R8G8B8A8_UNORM_SRC
Textures/wood_norm	D3D11 Shader Resource	new		B8G8R8A8_UNORM
Textures/wood_rough	D3D11 Shader Resource	new		R8_UNORM
DirectXTK:SpriteFont	D3D11 Texture2D		11,264	BC2_UNORM
obj:6	D3D11 Texture2D	*	3,686,400	D24_UNORM_S8_UINT
WICTextureLoader	D3D11 Texture2D		5,592,404	R8G8B8A8_UNORM
WICTextureLoader(1)	D3D11 Texture2D		11,184,808	R16G16B16A16_UNORM
WICTextureLoader(2)	D3D11 Texture2D		1,398,101	R8_UNORM

Pipeline Stages



The screenshot displays the Visual Studio Graphics Analyzer interface. The main window shows a list of GPU events on the left, including 'obj3->UpdateSubresource' and 'obj3->DrawIndexed'. The central pane shows the 'Render Target' and 'Frame Analysis' tabs. The 'Pipeline Stages' pane at the bottom shows a sequence of four stages: 'Input Assembler', 'Vertex Shader obj:8', 'Pixel Shader obj:11', and 'Output Merger'. The 'Pixel Shader obj:11' stage is highlighted with a green play button. A text overlay box is positioned over the center of the image, containing the following text:

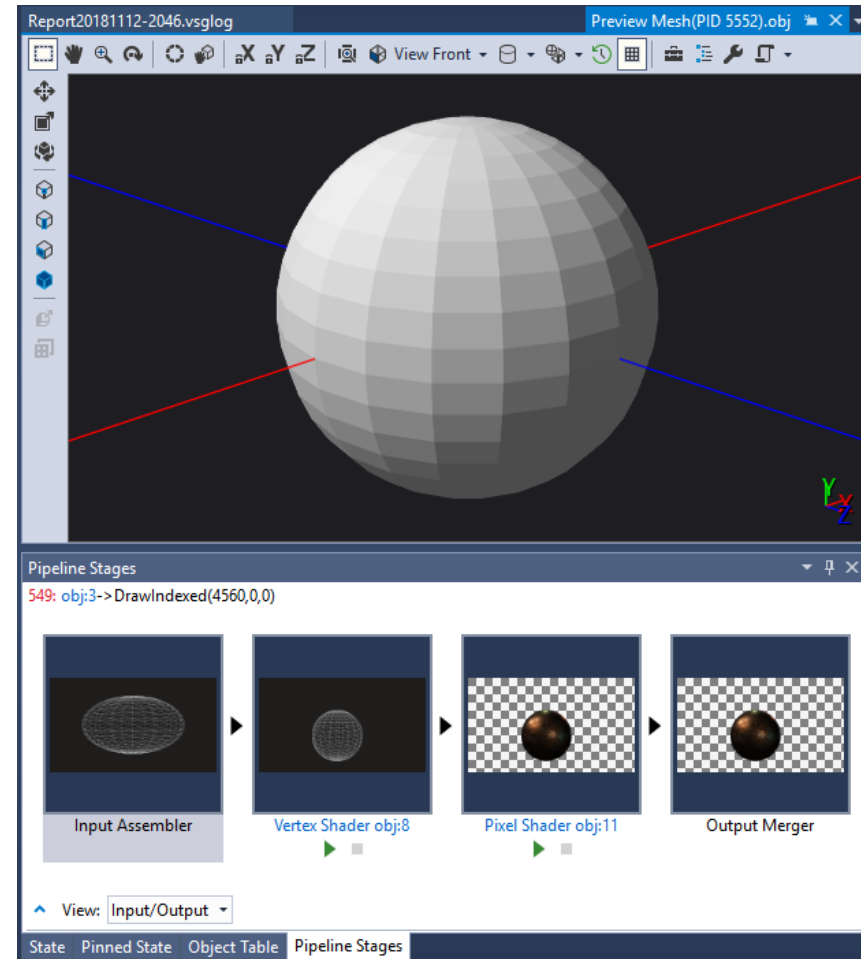
Shows the output of various pipeline stages. Great for determining where a drawing problem might be.

Click on a stage to see more details

Or click green “play button” to debug a shader

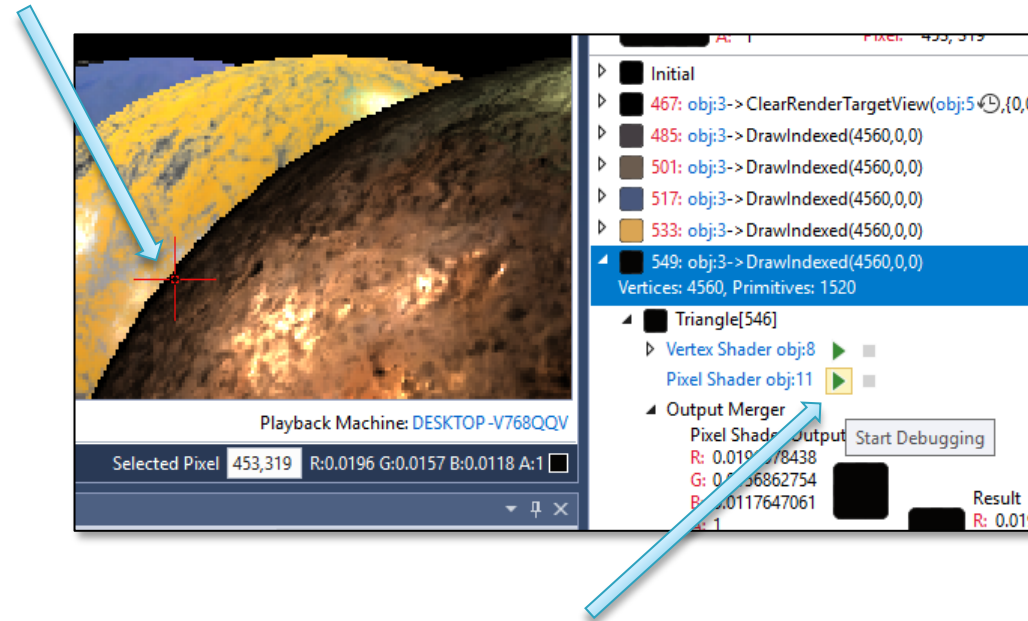
Pipeline Stage example: Input Assembler

- ▶ Click I.A. stage for a mesh preview
 - Right in Visual Studio
- ▶ Helps ensure you're drawing the proper geometry



Shader debugging

- ▶ Need to debug a shader?
 - First choose a pixel
 - And a draw call



- ▶ Then click the green “play button”