GigiViewerDX12 Documentation

This documents command line options, as well as the python interface.

The viewer can be run directly by running GigiViewerDx12.exe, or it can be opened from within GigiEdit.exe. When running it from the editor, some functionality is removed or changed, as it is a child process to the editor, instead of being a standalone application.

Command Line Options

GigiViewerDX12.exe can be ran with the following command lines:

- **-editor <IP> <Port>** : The editor launches the viewer with these command line parameters, telling the ip and port to connect to via TCP/IP to receive control messages from the editor.
- **-paused** : run with this command line parameter to start the viewer in a paused state. Useful when you are trying to debug the first frame of execution.
- **-nodebuglayer** : by default, the dx12 debug layer is enabled, but is not shown in the logs unless the "Settings->DX2 Debug Layer" option is turned on. This option makes the dx12 debug layer not be turned on, which can give more realistic performance numbers.
- **-gpuvalidation** : if the debug layer is on (default), this command will also turn on DX12 gpu validation. This can significantly affect perf, but can be useful for providing more information in the Log window.
- **-load <ggfilename>**: Load a .gg file after launching the viewer.
- **-run <pyfilename> <arguments>**: Runs a python script after launching the viewer. All arguments after this run command are passed to the python script in the sys.argv array, as per usual.
- **-norenderdoc:** disaled renderdoc captures
- **-nopixcapture:** disables pix captures. Pix captures being enabled currently interfere with AMD GPU Reshape (v0.9 BETA2).
- **-logdebuglayer:** Turns on the "Log DX12 Debug Layer" option under the settings menu, to log the validation layer to the log window.
- **-compileshadersfordebug:** turns on the "Compile Shaders For Debug" option under the settings menu.
- **-warpadapter**: uses the warp adapter. Useful for determinism across different hardware and other use cases. https://learn.microsoft.com/en-us/windows/win32/direct3darticles/directx-warp.
- **-AgilitySDKPreview**: Uses the preview version of the agility SDK, instead of the retail one which is used by default. The agility SDK version is shown in the log window on startup. These two versions of the AgilitySDK live in external/AgilitySDK.

Imported Resources

Imported resources can be set in the viewer and are loaded by the viewer into the specified resource. Buffers may be loaded as binary files off disk, csv files, or model formats including obj, fbx and ply.

Images formats that can be loaded include png, hdr and exr. Images and buffers can also be saved to disk from within the viewer.

PLY files load fields in order into the buffer they are loaded to. If a ply file has 2 floats and 1 double per vertex, it could be loaded into a structured buffer that had a float2 and a float in it, or it could be loaded into a typed buffer that was typed as a float3. Data conversion will happen on load, but the number of fields loaded is the minimum between how many fields are in the buffer, and how many fields are in the ply file. The buffer zero initialize fields not loaded from the ply file.

When loading a 2darray, 3d, or cube map texture, your filename must contain a %i where an index will go. It will load all matching files found from 0 to N.

When loading a cube map, the indices are in the standard directx order of +x,-x,+y,-y,+z,-z. You may alternately specify a %s in the file name, which will be replaced by the words “Right”, “Left”, “Up”, “Down”, “Front”, “Back”.

Textures used only for read access may also be loaded directly from shaders! See the shader documentation for more information.

The viewer puts a file watch on all shaders and loaded resources, allowing it to reload on changes for rapid iteration.

Save As BVH

The viewer can save out vertex data as bvh files. It uses tinybvh to generate the bvh data (https://github.com/jbikker/tinybvh) and generates them in the “BVH_GPU” format as described in the 2009 Aila & Laine paper (https://research.nvidia.com/sites/default/files/pubs/2009-08_Understanding-the-Efficiency/aila2009hpg_paper.pdf).

```
struct BVHNode
{
    // Alternative 64-byte BVH node layout, which specifies the bounds of
    // the children rather than the node itself. This layout is used by
    // Aila and Laine in their seminal GPU ray tracing paper.
    bvhvec3 lmin; uint32_t left;
    bvhvec3 lmax; uint32_t right;
    bvhvec3 rmin; uint32_t triCount;
    bvhvec3 rmax; uint32_t firstTri; // total: 64 bytes
    bool isLeaf() const { return triCount > 0; }
};
```

Four files are emitted with the following extensions:

- .nodes.bvh – contains the BVHNodes in the format shown above.
- .triindices.bvh – The lists of triangles for each BVH node (firstTri and triCount). Uint32.
- .vertices.bvh – The list of triangle vertex positions. Float4. Vertices for a triangle are found by multiplying the triangle index by 3 to get the first vertex, and the next two indices are the other two vertices.

- .combined.bvh – This contains all other data in a single file. This file begins with two uint32s which specify the size in bytes of the vert data and node data respectively. After that is the vertex data, the node data, and lastly is the triangle indices data.

An example of using these files can be found in the "tinybvh" demo in the Gigi browser.

Pix Captures

To take a pix capture, click the "Pix Capture" button. Next to this button is a text box which lets you specify how many frames to capture.

If you want debug symbols in your shaders for the pix capture, make sure "Compile Shaders For Debug" is turned on, under the Settings menu.

When the profiler window is shown, it will put GPU timing queries into the command list to get the GPU profiling numbers. These timing queries will then also show up in your pix captures as "EndQuery" calls. If you want to remove these, close the profiler window before taking a capture.

If you notice a lot of PCIE traffic in your pix capture, it may be that other applications are fighting with Gigi for vram. Closing things like browsers, photoshop, or similar applications which require a lot of GPU memory can help clear that up to give you more accurate timings.

Python Interface Overview

GigiViewerDX12.exe is able to run python files, and uses python 3.10.

You can load a .gg file and then run a python file, or you can run a python file without loading a .gg file. A python file may also load a .gg file, or it may load multiple, or it may not load any at all, instead choosing to work with whatever .gg file has been previously loaded.

Python scripts are ran in a blocking fashion, where user input and such is blocked until execution is completed. If desired, we could also add the ability to have a "Per Frame" python function call, if letting python run during user interaction was desired.

Python scripts may optionally close the viewer, and specify the application return code.

The python interface allows python to be used for a variety of purposes such as drive automated testing, automated data gathering, and initialization of technique parameters before giving control to the user. Python can also read and write GPU resources, allowing for things like machine learning to run in a tight loop of python and full speed GPU rendering and computation, without needing to use C++ or know any modern graphics APIs.

The Techniques/UnitTests/ folder has a number of python files to test the viewer functionality and the python interface. These can also serve as good examples for seeing how to do ray tracing, rasterization, read and write GPU resources, and more.

Python runs in an isolated mode, using the python within Gigi, to not interfere with whatever python you may (or may not) have installed. If you want to use a package that isn't part of this isolated python, you can use pip to install it!

To install a package, go to GigiViewerDX12\python\Python310 and run that python.exe like the below, which uses matplotlib as an example:

./python.exe -m pip install matplotlib

If you are installing a package you think other people will benefit from having access to, feel free to submit a merge request to get it into the repo.

Video Recording From Python With FFmpeg

Video recording from python involves taking screenshots and combining them with FFmpeg.

The python packages included in gigi do include ffmpeg-python but requires ffmpeg to be installed and on the path to function correctly.

FFMPEG binaries can be downloaded from:
https://ffmpeg.org/download.html#build-windows

I downloaded the latest ffmpeg-master-latest-win64-gpl.zip from this repo specifically:
https://github.com/BtbN/FFmpeg-Builds/releases

I extracted that zip onto my drive and added the bin folder to my path environment variable.

Python Interface Functionality

The python interface has two modules that you need to import to get the full functionality: Host and GigiArray.

The GigiArray module is how the Host module returns large amounts of read only data to python. It uses the buffer protocol and is easily turned into a numpy array with the numpy.array() function.

The Host module is where everything else is exposed, as listed below.

- **Host.LoadGG(fileName)** – loads a .gg file, returning a bool to indicate whether or not it was successful. If a relative path is given, it first checks relative to the python script location. If not found, it then checks relative to the GigiViewerDX12.exe location.
- **Host.Exit(exitCode=0)** – this causes the viewer to exit, with the given exit code. This is useful for situations like automated testing where you can return error codes to a calling application.
- **Host.GetScriptPath()** – Returns the path of the python script file ran, without a file name.
- **Host.GetScriptFileName()** - Returns the file name of the python script file ran, without the path.
- **Host.SetHideUI(set)** – this can hide or unhide the Gigi UI. Hiding the UI shows only one resource being viewed and imgui interface for the public variables. Hiding the UI is useful for doing demos, or for sharing a “quasi standalone application” with users. control+U will unhide the UI.
- **Host.SetVSync(set) –** VSync causes an application to be limited to a maximum of 60 or 120 FPS (depending on your display). Turning off VSync allows more accurate profiling numbers which do not include the artificial slowing down of the application from VSync.
- **Host.SetSyncInterval(syncInterval)** – IDXGISwapChain::Present() parameter: Synchronize presentation after the nth vertical blank.

- **Host.SetStablePowerState(set)** – when turned on, the GPU has more stable performance but is also slower. See: https://learn.microsoft.com/en-us/windows/win32/api/d3d12/nf-d3d12-id3d12device-setstablepowerstate
- **Host.SetProfilingMode(set)** – The viewer does extra work on the GPU to support viewer functionality. Set profiling mode to turn that extra work off, allowing more accurate profiling values. Setting profiling mode also causes vsync to be off.
- **Host.SetVariable(name, value)** – Sets the value of a Gigi variable. The value should be a string, which the Gigi viewer will parse into the appropriate data type. When setting an integer enum variable, you can either give it an integer value, or the enum label (with or without a EnumName:: prefix). In all cases, it should be passed as a string though.
- **Host.GetVariable(name)** – This will return the value of the specified Gigi variable, as a string.
- **Host. DisableGGUserSave(set=true)** – If true, .gguser files will not be saved. This is useful in automated testing or similar, where things will be changed, but it's not desired that the .gguser file is updated. .gguser files store all information set up in the viewer, such as variable values, system variable connections, and the details of imported resources.
- **Host.SetWantReadback(name, set=true)** – This lets the viewer know that you want to readback this resource. The name is the name shown at the top of the resource viewer, such as "Node_1.output: Output (UAV - After)". While those names are a mouth full, they uniquely identify the resource to write, as well as what part of the rendering timeline to write the value. There is a "copy to clipboard" button next to the name for your convenience.
- **Host.Readback(name, arrayIndex=0, mipIndex=0)** – This reads back a resource, returning a GigiArray, as well as a success Boolean. SetWantReadback() must be called, and the technique must be executed, before you can read back the data. ArrayIndex is only used by texture 2d arrays. It is ignored for all other types.
- **Host.SaveAsPNG(filename, resourceName, arrayindex=-1, mipIndex=-1)** – This saves a texture resource to a file. If arrayIndex is -1, then each slice and mip level will be written out. Else you are expected to provide a specific arrayIndex and mipIndex. SetWantReadback() must be called, and the technique must be executed, before you can use this function.
- **Host.SaveAsDDS_BC4(filename, resourceName, isSigned=False, arrayindex=-1, mipIndex=-1)** – This saves a texture resource to a file. If arrayIndex is -1, then each slice and mip level will be written out. Else you are expected to provide a specific arrayIndex and mipIndex. SetWantReadback() must be called, and the technique must be executed, before you can use this function.
- **Host.SaveAsDDS_BC5(filename, resourceName, isSigned=False, arrayindex=-1, mipIndex=-1)** – This saves a texture resource to a file. If arrayIndex is -1, then each slice and mip level will be written out. Else you are expected to provide a specific arrayIndex and mipIndex. SetWantReadback() must be called, and the technique must be executed, before you can use this function.
- **Host.SaveAsDDS_BC6(filename, resourceName, isSigned=False, arrayindex=-1, mipIndex=-1)** – This saves a texture resource to a file. If arrayIndex is -1, then each slice and mip level will be written out. Else you are expected to provide a specific arrayIndex and mipIndex. SetWantReadback() must be called, and the technique must be executed, before you can use this function.

- **Host.SaveAsDDS_BC7(filename, resourceName, isSRGB=True, arrayindex=-1, mipIndex=-1)** – This saves a texture resource to a file. If arrayIndex is -1, then each slice and mip level will be written out. Else you are expected to provide a specific arrayIndex and mipIndex. SetWantReadback() must be called, and the technique must be executed, before you can use this function.
- **Host.SaveAsEXR(filename, resourceName, arrayindex=-1, mipIndex=-1)** – This saves a texture resource to a file. If arrayIndex is -1, then each slice and mip level will be written out. Else you are expected to provide a specific arrayIndex and mipIndex. SetWantReadback() must be called, and the technique must be executed, before you can use this function.
- **Host.SaveAsHDR(filename, resourceName, arrayindex=-1, mipIndex=-1)** – This saves a texture resource to a file. If arrayIndex is -1, then each slice and mip level will be written out. Else you are expected to provide a specific arrayIndex and mipIndex. SetWantReadback() must be called, and the technique must be executed, before you can use this function.
- **Host.SaveAsCSV(filename, resourceName, arrayindex=-1, mipIndex=-1)** – This saves a texture resource to a file. If arrayIndex is -1, then each slice and mip level will be written out. Else you are expected to provide a specific arrayIndex and mipIndex. SetWantReadback() must be called, and the technique must be executed, before you can use this function.
- **Host.SaveAsBinary(filename, resourceName, arrayindex=-1, mipIndex=-1)** – This saves a texture resource to a file. If arrayIndex is -1, then each slice and mip level will be written out. Else you are expected to provide a specific arrayIndex and mipIndex. SetWantReadback() must be called, and the technique must be executed, before you can use this function.
- **Host.RunTechnique(count=1)** – This runs the technique <count> times.
- **Host.Log(Level, message)** – This logs a message to the viewer window. Level can be “Info”, “Warn”, or “Error”. Message needs to be a string.
- **Host.Print(message)** – The same as Host.Log(“Info”, message).
- **Host.Warn(message)** – The same as Host.Log(“Warn”, message).
- **Host.Error(message)** – The same as Host.Log(“Error”, message).
- **Host.SetFrameIndex(index)** – Sets the system variable frame index to a specific value. Useful when the frame index is used as a random number seed or is used in other ways, to be able to set a specific point in time, or just reset back to 0 to restart.
- **Host.WaitOnGPU()** – Waits for all pending GPU work to be finished. It’s best practice to do this before a readback, to make sure you are getting the data you think you are.
- **Host.Pause(pause)** – Pause or unpause execution of the technique in the viewer. Does not effect Host.RunTechnique.
- **Host.PixCaptureNextFrames(filename, frames = 1)** – Causes a pix capture to be taken. If it wants N frames to be captured, you must run the technique that many times, or return control to the user to allow that many frames to be rendered.
- **Host.SetImportedBufferFile(importedBufferName, fileName)** – Sets the filename of an imported buffer.
- **Host.SetImportedBufferStruct(importedBufferName, structName)** – Sets the buffer’s type to be a specific struct type.
- **Host.SetImportedBufferCount(importedBufferName, type)** – Sets the buffer’s type to be a POD (non struct) type. Uses Host.DataFieldType enum. See PythonTypes.txt for more info.

- **Host.SetImportedBufferCSVHeaderRow (importedBufferName, CSVHeaderRow)** – Sets whether or not the buffer's csv file has a header row.
- **Host.SetImportedBufferCount(importedBufferName, count)** – Sets how many items are in the buffer.
- **Host.SetImportedTextureFile(importedTextureName, fileName)** – Sets the filename of an imported texture.
- **Host.SetImportedTextureSourceIsSRGB(importedTextureName, sourceisSRGB)** – Sets whether or not the source texture is SRGB.
- **Host.SetImportedTextureMakeMips(importedTextureName, makeMips)** – Sets whether or not to make mips for the imported texture.
- **Host.SetImportedTextureFormat(importedTextureName, format)** – Sets the format of an imported texture. The importedTextureName is the name of the texture in the "Imported Resources" window. The format is specified using the enum Host.TextureFormat. Append the format you see in the drop down to Host.TextureFormat, such as Host.TextureFormat_RGBA32_Float.
- **Host.SetImportedTextureColor(importedTextureName, R, G, B, A)** – Sets the color of an imported texture.
- **Host.SetImportedTextureSize(importedTextureName, x, y, z)** – Set the size of an imported texture. The importedTextureName is the name of the texture in the "Imported Resources" window. x,y,z should be integers. For a 2D texture, set z to 1, not 0.
- **Host.SetImportedTextureBinarySize(importedTextureName, x, y, z)** – Set the size of an imported texture if it's a binary file. The importedTextureName is the name of the texture in the "Imported Resources" window. x,y,z should be integers. For a 2D texture, set z to 1, not 0.
- **Host.SetImportedTextureBinaryFormat(importedTextureName, format)** – Sets the format of the binary file being loaded for the imported texture.
- **Host.SetFrameDeltaTime(deltaTime)** – in seconds. This sets the system variable for the frame delta time to a fixed value. Useful if wanting to record a video. Set to <= 0 to clear the forced delta time.
- **Host.SetCameraPos(x,y,z)** – set the position of the camera.
- **Host.SetCameraFOV(fov)** – sets the FOV of the camera.
- **Host.SetCameraAltitudeAzimuth(altitude, azimuth)** – in radians. We should add a "look at" function and similar to make working with the camera easier.
- **Host.SetCameraNearFarZ(nearZ, farZ)** – sets the near and far plane of the camera
- **Host.SetCameraFlySpeed(speed)** – sets the fly speed of the camera
- **Host.GetCameraPos()** – returns the x,y,z position of the camera
- **Host.GetCameraAltitudeAzimuth()** – returns the altitude and azimuth of the camera. In radians.
- **Host.WriteGPUResource(name, value, subresourceIndex = 0)** – This causes a gpu resource to be written the next time the technique is executed. The name is the name shown at the top of the resource viewer, such as "Node_1.output: Output (UAV - After)". While those names are a mouth full, they uniquely identify the resource to write, as well as what part of the rendering timeline to write the value. There is a "copy to clipboard" button next to the name for your convenience. The value is expected to be a bytes object that is the same size as the destination resource. The subresourceIndex is only used by texture2darrays, where it is the array index.

- **Host.ForceEnableProfiling(set=True)** – Let the viewer know about your intent to read profiling data. must be called before calling Host.GetProfilingData(). Profiling data is not available until the technique is executed.
- **Host.GetProfilingData()** – Returns a dictionary mapping render graph node names to an array which contains the cpu and gpu timing in milliseconds, respectively. Also contains a "total".
- **Host.IsResourceCreated(nodeName)** – Returns whether the resource node specified has its resource created or not. Takes a node name, not a resource name.
- **Host.SetViewedResource(resourceName)** – Makes the viewer view the specified resource name. This is the same resource name you see in the viewer, such as "Input – Initial State", not the node name.
- **Host.GGEnumValue(enumName, enumLabel)** – This returns the integer value of the enum label within the enum name given.
- **Host.GGEnumLabel(enumName, value)** – This returns the string label of the integer value in the enum.
- **Host.GGEnumCount(enumName)** – Returns an integer count of how many items there are in the enum.
- **Host.GetGPUString()** – Returns a string with the GPU name and driver version.
- **Host.SetShaderAssertsLogging(value)** - Toggles auto error logging of the collected shader asserts after a technique execution. Value is Boolean.
- **Host.GetCollectedShaderAssertsCount()** - Returns the number of collected shader asserts. Assert getters works with this collection.
- **Host.GetShaderAssertFormatStrId(index)** - Returns the ID of format string of the specified shader assert.
- **Host.GetShaderAssertFormatString(index)** - Returns the format string of the specified shader assert.
- **Host.GetShaderAssertDisplayName(index)** - Returns the display name of the specified shader assert.
- **Host.GetShaderAssertMsg(index)** - Returns the message of the specified assert.

Python Interface Gigi Schema Enums

Several enums are defined by Gigi's internal schema reflection system and automatically reflected into python functions and constant values.

These are different than enums defined in the .gg files because they are present even when there is no .gg file loaded.

Every enum has the following things exposed. Replace Host.TextureFormat with the name of the enum. See PythonTypes.txt in this folder for more information.

- **Host.TextureFormatToString(index)** – gives the string name of an integer index, or "Invalid" if it is out of range.
- **Host.TextureFormatFromString(name)** – gives the integer index of a name, or -1 if it doesn't exist.
- **Host.TextureFormat_FIRST** – currently, always 0.
- **Host.TextureFormat_LAST** – the last valid enum value.

- **Host.TextureFormat_COUNT** – currently LAST + 1
- Also, all the values are exposed as integer constants, such as **Host.TextureFormat_RGBA8_Unorm_sRGB**.