Cooperative Vectors in Gigi

Cooperative vectors are a preview feature of directX which simplify and accelerate large matrix multiplies and similar operations. Their goal is to aid machine learning workloads, but as machine learning is just a big pile of linear algebra, there are many other uses too, including convolution (image filtering!).

A high level overview of cooperative vectors is at: <https://devblogs.microsoft.com/directx/enabling-neural-rendering-in-directx-cooperative-vector-support-coming-soon/>

As this is a preview feature of DX12, the final feature may have a different interface. That means that Gigi too may have a different interface. Any techniques made with the cooperative vector interface as it is now may not work in the future without modification. Cooperative vectors are a viewer only feature at the moment, and not supported by any code generation targets yet.

Intel has a nice demo using cooperative vectors to do neural texture synthesis.  
<https://github.com/GameTechDev/TextureSetNeuralCompressionSample>

Deeper technical information about cooperative vectors in DX12 is at: <https://devblogs.microsoft.com/directx/cooperative-vector/>

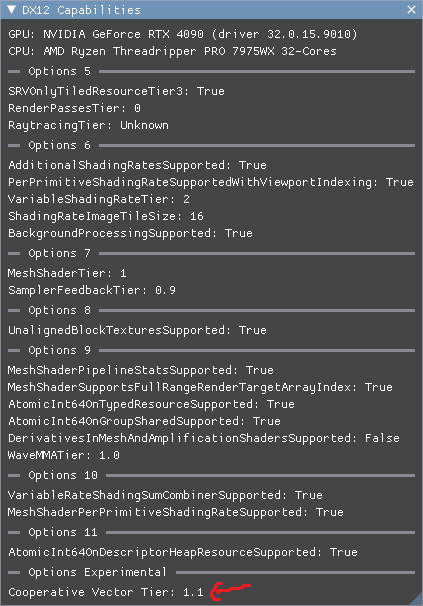
Apparently, the cooperative does (or can?) use tensor cores on the GPU when available.

# Setup

To get cooperative vectors working in Gigi you must…

* Get special drivers that support them. (<https://devblogs.microsoft.com/directx/cooperative-vector/#get-running>)
* Turn on developer mode (<https://learn.microsoft.com/en-us/windows/apps/get-started/enable-your-device-for-development>)
* Run the Gigi Viewer with the command line parameters -AgilitySDKPreview -nopixcapture -norenderdoc.
  + **-AgilitySDKPreview** – This tells the viewer to use the preview version of directx which supports cooperative vectors.
  + **-nopixcapture -norenderdoc** – This turns off pix and renderdoc captures. These interfere with cooperative vector support, likely because they hook the DX12 API and so break some of this future functionality.

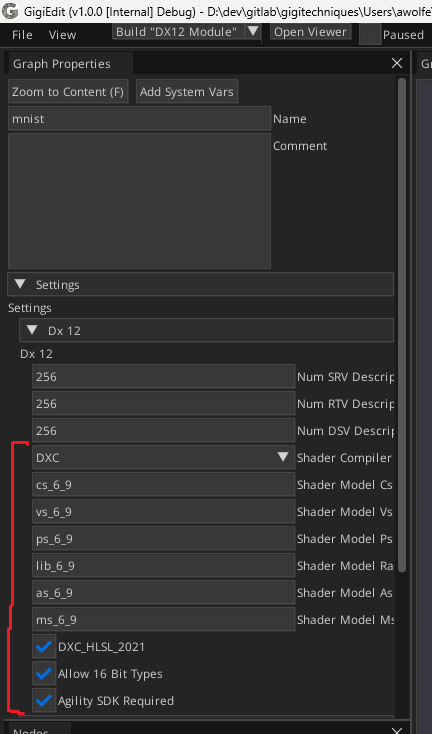
You can see if these steps were successful in the viewer by going to the View menu and selecting DX12 Capabilities. At the bottom of the window you will see the Cooperative Vector Tier. If it says “None” or “Unknown”, you either did the steps wrong, or your computer does not support cooperative vectors.



Cooperative vector tier 1.0 supports MatrixVectorMul and MatrixVectorMulAdd shader intrinsics. Tier 1.1 supports OuterProductAccumulate and VectorAccumulate.

# Setting Up A Gigi Project

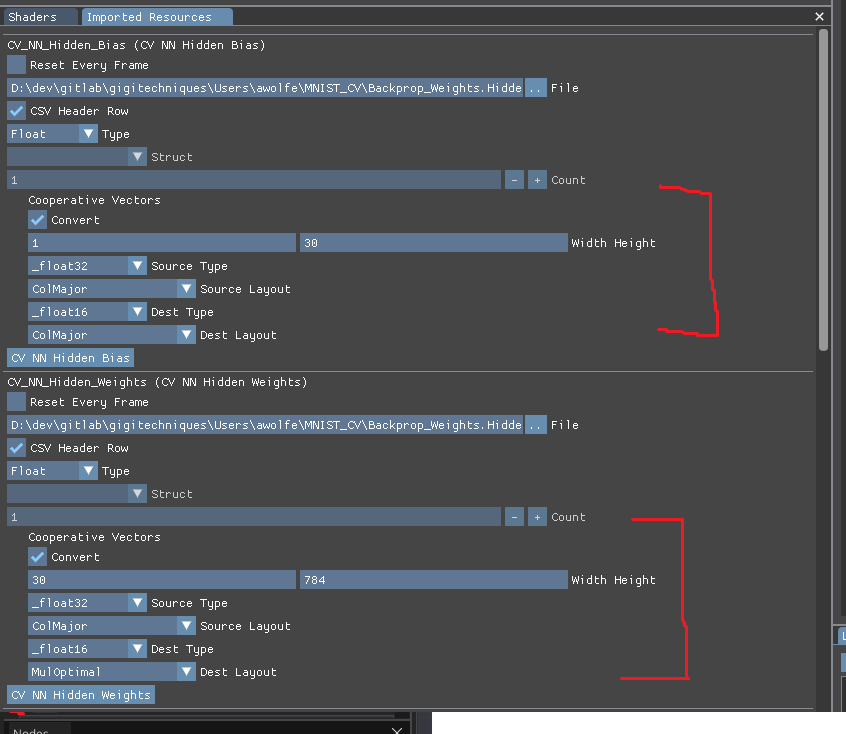
Cooperative vectors require using the DXC compiler, using shader model 6.9, the Agility SDK, and you are probably going to want 16 bit types, as well as HLSL 2021. You can set these things up in the “Graph Properties” window under **Settings->DX12**.



# Setting Up Data Buffers

Cooperative vectors support a variety of data formats including 16 bit floats (halfs), uint8s, and some exotic 8 bit floating point formats. When doing operations with this data, you also have to specify what layout the data is in: row major, column major, optimized for multiplication or optimized for outer products.

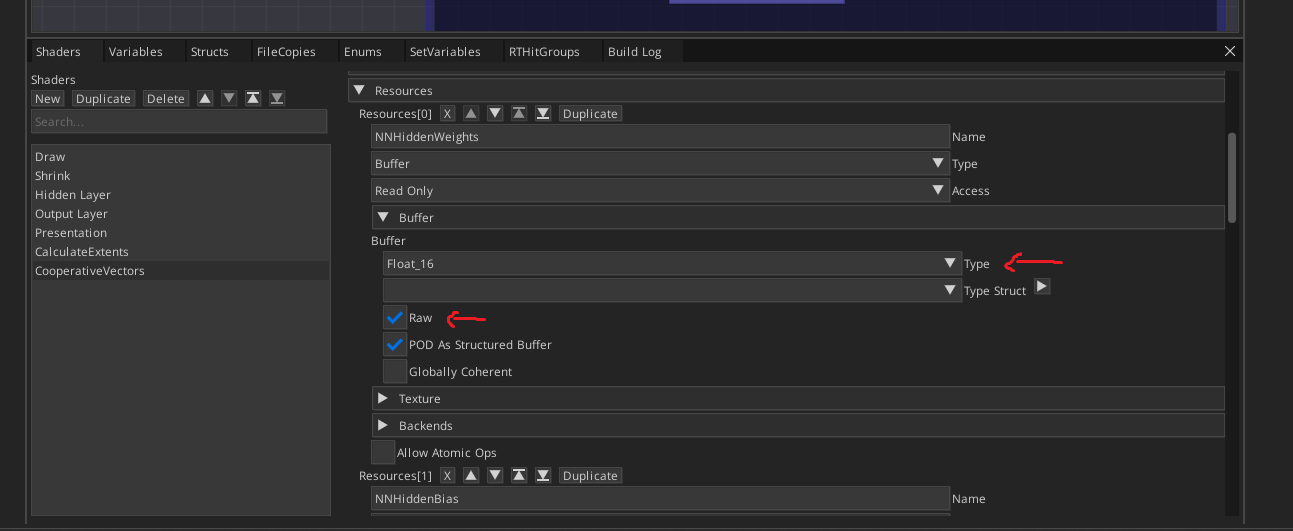
To help you get the data into these formats and layouts, a DX12 API is exposed, which Gigi exposes to you in the viewer for imported resources.



I hit some crashes when trying different layouts and formats, but following the path that the intel demo followed (16 bit floats, and “MulOptimal” layout for matrices) that things were a lot more stable. One crash I hit was that the 32 bit float format isn’t supported. Your mileage may vary with different driver vendors or future driver versions.

# Writing A Shader

Using cooperative vectors in a shader is pretty easy luckily. The buffer resources involved need to be passed as **Raw** (ByteAddressBuffer or RWByteAddressBuffer), and you should still specify a type in the **Type** drop down.



In the shader itself you need to include a shader header “linalg.h”. Gigi handles putting that file into your shader folder. I prefer a “using namespace” line to make less typing.

A black background with white text

AI-generated content may be incorrect.

The cooperative vector shader API gives 4 functions

* Mul(MatrixRef M, InterpretedVector V) – Result = M\*V
  + Just a matrix / vector multiply.
* MulAdd(MatrixRef M, InterpretedVector V1, VectorRef V2) – Result = M\*V1+V2
  + This is useful for running a neural network layer: Weights\*Input+Bias.
* OuterProductAccumulate(vector V1, vector V2, RWMatrixRef M)
  + M += V1 ⊗ V2
* VectorAccumulate(vector V, RWByteAddressBuffer Buffer, uint Offset)
  + Vector(Buffer[Offset]) += V

For the data types it exposes, MatrixRef and VectorRef (and RW variation which allows writing) don’t actually load the data into memory but are just wrappers to let the functions load the data as needed. The InterpretedVector is a thin wrapper to a vector (called “long vectors” in DX12 terminology) and vectors are fully loaded into memory.

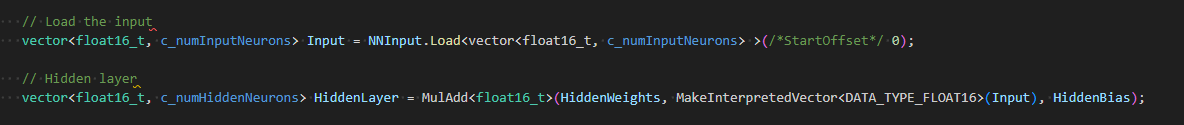
To get a MatrixRef, you tell it the data type, the rows and columns, and the layout as template parameter. You also have to give it a raw buffer (ByteAddressBuffer), the location in the buffer where the matrix begins, and a stride.



To get a VectorRef, you only need to give a data type as the template parameter, and you give a raw Buffer and the location in the buffer where the vector begins.



To get an InterpretedVector, you have to load data into a vector and then call MakeInterpretedVector(). In this snippet, NNInput is a raw buffer too.



The functions Mul and MulAdd return a vector, as shown in the snippet above. If you want to write these out, you can do so with 1 store statement to a raw buffer accessed as read/write (UAV access to a RWByteAddressBuffer). Doing activation functions to vectors is also super simple and written like regular shader code, which is very nice.

A black background with colorful lines

AI-generated content may be incorrect.

# Working Demo in Gigi Browser – Cooperative Vector MNIST

In the Gigi Browser is a demo named “Cooperative Vector MNIST” that does real time inference on a numeric digit that you draw in a box. It has code paths for both cooperative vectors inference, as well as regular compute shaders.

This is a demo of functionality, but not a good demo of performance and actually slows down slightly when you use cooperative vectors instead of regular compute shaders. The reason for this is that the demo runs a neural network once per frame, while cooperative vectors are more designed to run once per pixel or similar. On my machine, the frame time is 0.140ms with compute shaders, and 0.155ms with cooperative vectors.

Assuming you did the setup work described earlier, after downloading the technique in the Gigi browser, you need to run the viewer with the command line parameters **-AgilitySDKPreview -nopixcapture -norenderdoc**. Then you can load the technique in the viewer.

Demo:A screenshot of a computer program

AI-generated content may be incorrect.

Render Graph:

A screenshot of a computer screen

AI-generated content may be incorrect.

Gigi Shader:

A screenshot of a computer program

AI-generated content may be incorrect.

Processed Shader:

A screenshot of a computer program

AI-generated content may be incorrect.