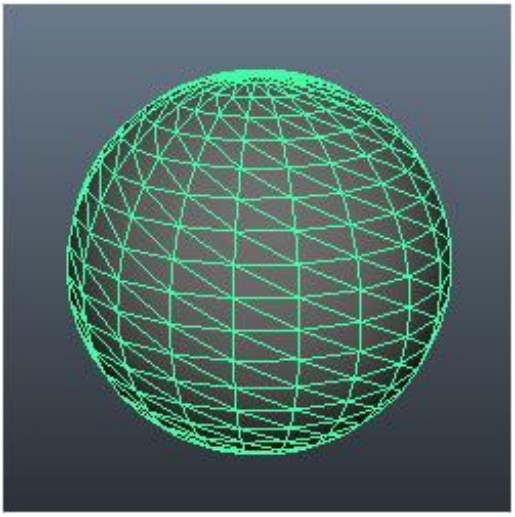
Buffers, BufferPools and Vertices…

rev1.2 Oct 24, 2013

Vertex Buffers (Courtesy of RasterTek:- http://www.rastertek.com/dx10tut04.html)

The first concept to understand is vertex buffers. To illustrate this concept let us take the example of a 3D model of a sphere:

The 3D sphere model is actually composed of hundreds of triangles:



Each of the triangles in the sphere model has three points to it, we call each point a vertex. So for us to render the sphere model we need to put all the vertices that form the sphere into a special data array that we call a vertex buffer. Once all the points of the sphere model are in the vertex buffer we can then send the vertex buffer to the GPU so that it can render the model.

Index Buffers

Index buffers are related to vertex buffers. Their purpose is to record the location of each vertex that is in the vertex buffer. The GPU then uses the index buffer to quickly find specific vertices in the vertex buffer. The concept of an index buffer is similar to the concept using an index in a book, it helps find the topic you are looking for at a much higher speed. The DirectX SDK documentation says that using index buffers can also increase the possibility of caching the vertex data in faster locations in video memory, which will increase overall performance..

Vertex Shaders

Vertex shaders are small programs that are written mainly for transforming the vertices from the vertex buffer into 3D space. There are other calculations that can be done such as calculating normals for each vertex. The vertex shader program will be called by the GPU for each vertex it needs to process. For example a 5,000 polygon model will run your vertex shader program 15,000 times each frame just to draw that single model. So if you lock your graphics program to 60 fps it will call your vertex shader 900,000 times a second to draw just 5,000 triangles. As you can tell writing efficient vertex shaders is important.

Pixel Shaders

Pixel shaders are small programs that are written for doing the coloring of the polygons that we draw. They are run by the GPU for every visible pixel that will be drawn to the screen. Coloring, texturing, lighting, and most other effects you plan to do to your polygon faces are handled by the pixel shader program. Pixel shaders must be efficiently written due to the number of times they will be called by the GPU.

HLSL

HLSL is the language we use in DirectX 10 to code these small vertex and pixel shader programs. The syntax is pretty much identical to the C language with some pre-defined types. HLSL program files are composed of global variables, type defines, vertex shaders, pixel shaders, and geometry shaders.

BufferPools in FSX

…….. Paul J

With major guidance from Steve Parson's WordPress Blog site

Below are some examples of buffer definitions, and how buffers are used in other applications in the IT world:

IBM: https://publib.bould...oc/tuning39.htm

Informix: http://www.informix-...ng-io-with.html

GPU: 2004 - PD: http://www.cse.ohio-.../2-Hardware.pdf

Buffering: general links: http://social.techne...inement=41&ac=2

Wiki: http://en.wikipedia....iki/Framebuffer

DirectX v9.x Buffer Management:

A buffer "pool" in the *DX9* FSX world, is a large number of memory buffers, created as a major component of a rendering *pipeline* system which will allow graphics processing to be shared between the CPU and the GPU.

They contain the raw geometric data in a one-dimensional array of Vertex data, which is used to build up a scene. In DX9, the buffer pool improves system performance by allowing the CPU to fill the buffers with vertex data, while at the same time allowing the GPU to process earlier batches of vertex data. This is akin to parallel processing, and was necessary with the early versions of FSX, because graphics cards had but a smidgeon of speed and memory that gpu cards have nowadays, and much graphics processing was done by the main cpu, with the buffer parameters pretty much set by Microsoft, and not touched or tweaked by the user.

DirectX v10.x Buffer Management:

In DX9.x, the data in the rendering pipeline was shared between the CPU and the GPU: in DX10, this is not the case - all graphics processing is undertaken by the GPU, thus freeing up system memory, roughly a size equivalent to what is currently in-use by the GPU. This is the single, most significant change, this alone, providing a boost in smoothness and performance for the system side.

Buffer Size:

Because most data manipulation takes place in buffer pools, right-sizing the buffer(s) is important. Too small and there may well be a crash from a 'buffer overflow': too large, and memory space is wasted.

(quote from 'Raf' - ACES Team, in a 2007 post "In RTM, the default setting was 1MB (1000000). The lower this number, the more pools the allocator will have to rummage through to find space for buffers and the more stutters you may have. In Sp1, we raised the default to 4MB (4000000) and optimized the underlying algorithm for finding free buffers. So be careful here, making this smaller can hurt you, since searching for space takes time and can cause stutters, and making the number too large can waste space. 4-10m is probably the range to be thinking about using unless you have a very high memory graphics card (>512) (/quote)

With the release of SP2, the buffer size was moved to 8MB (8388608).

For the purposes of the examples below I will suggest a "given" GPU having 2 GB of video memory. In default FSX - graphics data is sent from the CPU to the GPU in a "pipeline". When each data object gets to the GPU - it is assigned to a buffer - an address space in video ram. The GPU creates a pool of such buffers, with FSX controlling the size for each buffer - normally defaulting to the FSX SP2 default of 8MB (8388608 bytes). There is no "overt" user input in the default system, however - since the Jesus "Bojote" Altuve revelations 2009 - 2010 - the FSX user can also have control - hence this document.

Setting **UsePools=1** in the fsx.cfg file will tell FSX to control the creation of shared vertex buffers - dynamic buffers initially, at the default size of 8MB, and then to place all of the data objects arriving into those buffers until they're full, the old data being replaced by fresh data as it's processed out, with more buffers being created as necessary, and discarded (questionable) when not needed, and so on, as the datastream data quantity increases and decreases. At some point it will level off, if there's no *more* pressure from the CPU. It is not necessary to enter this setting on its own into the fsx.cfg unless one wished to change the buffer size with PoolSize=xxxxxxx

Setting **UsePools=0** in the fsx.cfg file will cause the *GPU* (in DX10) to create as large a number of small "dedicated" static buffers as the pipeline data supply needs, and will put every data object coming from the CPU into its own dedicated buffer. Static buffers will generally show a performance gain, and nice if

your GPU is fast with lots of memory! This setting allows the GPU to control the numbers of, and size of each buffer.

At this point in the dialogue then we have two methods of buffer control, if you like, one being UsePools=0, where all buffers are static and dedicated, with the GPU in control, and the other method being UsePools =1, with all buffers being dynamic and shared, with the size controlled by FSX or the user.

Having the FSX user control the buffering process with UsePools=1 introduces some issues, such as - "When is a buffer too large?" or "How many is too many?" We don't want to waste memory space: buffers too small may well result in more buffers than the GPU has room for, or, if an object is too large for a buffer, a stall and a spike may - or will result.
When using UsePools=0, it is easy - the GPU handles the creation decisions.

What is needed here, then - is some method of optimizing the performance of the buffering system, when UsePools=1, so that it matches the throughput of the pipeline. This can be achieved with the addition of a RejectThreshold number, in bytes. A RejectThreshold will allow the FSX user to set an object-size limit to the shared buffer, so that when used, the GPU will create a dedicated buffer for any incoming object which is larger than the RT number. RejectThreshold=xxxxxx therefore provides a method whereby the balance - the ratio - between shared and dedicated buffers can be altered - the lower the RT number the more objects get their own buffer, and theoretically, if low enough, it would end up having the same effect as reducing the threshold to zero, with every object getting its own dedicated, static buffer. The result would then be the same as UsePools=0.

The general rules are:-

1). If the GPU is very fast, and with a good-sized memory, say 2 - 4 gig - it can *eat* all of the incoming data and wait for more, so we allow the GPU to give every object its own buffer by UsePools=0.
To control any spiking, lowering of the IQ sliders in FSX is the best course of action. If spiking or artifacting occurs, then switch to UsePools=1, and experiment with RT sizes.

2). If the GPU is kinda "medium-range" - a 660 or 670 - fed by a 4.6 - 4.7-gig CPU - then we should use UsePools=1, making all buffers shared. In this case - as with most systems - this will be, or what should be used, and by adding a RejectThreshold - a small number - say 262144 (256kB) under the UsePools=1 line. It will look like this:- RejectThreshold=262144

The RejectThreshold will act as a throttle, by allowing the creation of dedicated buffers - which is then of course, a little bit of UsePools=0 mixing in - it could mean that 20% of the buffers are now dedicated, while the other 80% are shared. If we make that number smaller, then *more* dedicated buffers will be created, with the resulting improved performance. In this example (above) this number - 262144 - might be perfect: if not, then larger - or smaller - number will show the way. This needs the user's testing.
Spiking will require lowering of the IQ sliders in FSX *or* increasing the size of the RT, *or* raising the size of each buffer.

3). If, however - the CPU is stronger than the GPU, such as a 4.5-gig CPU feeding a 560TI with only a gig of video ram - we will definitely need UsePools=1 for sure, but with a big-ish RT, say 1MB (10485760). Other larger or smaller numbers and testing will show the way. Some benefit may be gained by experimenting with the buffer size, too, as shown below:-

[BufferPools]

UsePools =1
Poolsize=10475760
RejectThreshold=524288

Other RT numbers may be tried, (e.g. 65536, 131072,262144 ,524288, 786432, 10475760) .

The final numbers will depend upon a number of other factors, most which are set in the fsx,cfg, along with the entries in RadeonPro or Nvidia Inspector. These settings will also be dependent upon monitor size and the overall "pressure" that are the *user's expectations*!

I hope this helps!

pj, Oct 23rd, 2013 revised Mar 21, 2014