

Maximum Loadable Volumes on VP500

This paper will discuss issues related to storing volume data on the VolumePro 500 product. In particular, the largest sized volumes that will fit in VP500's voxel memory at one time will be addressed. The summary will include tables that detail the maximum sized volumes that can be loaded onto a VP500 at one time for different configurations of voxels, slice dimensions, rendering options and voxel memory. Note that even larger sized volumes can be supported by VLI but require swapping out of data to and from Voxel Memory which can severely limit the effective rendering frame rate.

Numerous issues affect how efficiently voxel memory can be utilized. Some of these are the following:

- Amount of Voxel Memory: 128 or 256 MegaBytes
- Bits Per Voxel: 8 or 16 Bits
- Tile Size: 32, 64, 128 or 256 voxels in each dimension
- Voxel Memory Organization: four or eight 32 MegaByte, non-continuous banks
- Padding Volume Extremities - this is necessary by the user in order to view all voxels of a data set.
- Fragmentation of Voxel Memory caused during the loading of subvolumes of a super volume.

The next sections will discuss the above issues and describe the organization of Voxel Memory with regard to loading volume data sets. An example will be given of a typical medical data set.

1 Voxel Memory Organization

VP500's Voxel Memory comes in either 128 MB or 256 MB configurations. These configurations are either 4 or 8 32 MB Banks of memory (comprised of 4 256Mbit SDRAMs each). For the purposes of this discussion, each bank is considered separate, independent and non-continuous with the other banks. The VG500 volume rendering ASIC on the VP500 board can only render a baseplane image from one of these banks at a time. This means that larger volumes (more than 256^3 voxels), called super volumes, must be split-up into multiple subvolumes of size 256^3 voxels or less and stored across all the banks of Voxel Memory.

An additional requirement in breaking up supervolumes into subvolumes for rendering is that the seams between the subvolumes must be "overlapped" by 3 voxels. This is necessary for the VG500 ASIC to compute valid gradients (needed for lighting) for samples along the edges of the subvolumes. This further limits the voxel memory utilization efficiency as some data must be replicated.

Another restriction of volume storage in Voxel Memory is that memory must be allocated in $32 \times 32 \times 32$ chunks of voxels (referred to as **Storage Units** in the rest of this discussion). This restriction has to do with the way the ASIC was optimized for optimal memory bandwidth and processing performance. This means that memory allocation granularity is $32 \times 32 \times 32 @ 2\text{Bytes} = 64\text{KBytes}$ for 16-bit voxels (or 12-bit) and half that, 32KBytes, for 8-bit voxels. This translates into 512 **Storage Units** per 32 MB bank for 16-bit voxels.

At first glance, one might think that a volume of size $512 \times 512 \times 512 @ 16\text{Bits}$ could fit entirely within the 256 MB configuration. However, due to the overlapping required when sub-dividing the supervolume, and the **Storage Unit** granularity of 32^3 , this is not the case (tables later in this paper will identify the exact size limitations).

In order to reduce fragmentation of voxel memory and provide the most efficient use of voxel memory, subvolumes of a supervolume are sorted by Storage Unit sizes and allocated in largest to smallest order. This guarantees the best packing of subvolumes in voxel memory.

1.1 Example $512 \times 512 \times 445 @ 16\text{-Bit}$ Volume

As an example, consider a typical medical data set of size $512 \times 512 \times 445 @ 16\text{ Bits}$ per voxel that is loaded onto a 256 MB VP500. This supervolume must be divided into 18 different subvolumes of varying sizes. The following table lists these subvolumes, their sizes and which of the 8 banks of voxel memory it would be stored in.

Vol#	U_{min}	U_{max}	V_{min}	V_{max}	W_{min}	W_{max}	Units
0	0	255	0	255	0	255	512
1	253	508	0	255	0	255	512
2	506	511	0	255	0	255	64
3	0	255	253	508	0	255	512
4	253	508	253	508	0	255	512
5	506	511	253	508	0	255	64
6	0	255	506	511	0	255	64
7	253	508	506	511	0	255	64
8	506	511	506	511	0	255	8
9	0	255	0	255	253	444	384
10	253	508	0	255	253	444	384
11	506	511	0	255	253	444	48
12	0	255	253	508	253	444	384
13	253	508	253	508	253	444	384
14	506	511	253	508	253	444	48
15	0	255	506	511	253	444	48
16	253	508	506	511	253	444	48
17	506	511	506	511	253	444	6

Table 1-1 SubVolumes of a 512x512x445 @ 16 Bits Volume

These subvolumes are then sorted, according to size (as previously mentioned), and stored in voxel memory. Subvolumes are stored in the lowest number bank that they will fit into. The following table shows which banks of voxel memory the sub-volume is stored in and the number of **Storage Units** per bank after each subvolume is loaded.

Vol#	Units	Bank	0	1	2	3	4	5	6	7
0	512	0	512							
1	512	1		512						
3	512	2			512					
4	512	3				512				
9	384	4					384			
10	384	5						384		
12	384	6							384	
13	384	7								384
2	64	4					448			
5	64	4					512			
6	64	5						448		
7	64	5						512		
11	48	6							432	
14	48	6							480	
15	48	7								432
16	48	7								480
8	8	7							487	
17	6	7							495	
remain			0	0	0	0	0	0	17	32

Table 1-2 Bank Allocation for 512x512x445 @ 16 Bits Volume

Note that all the subvolumes can be loaded at one time into VP500's voxel memory on a 256 MB board. However, the next example shows why even just one more slice of voxel data pushes the required voxel memory over 256 MB and thus not all of this larger volume can fit, at one time, in voxel memory.

1.2 Example 512x512x446 @ 16-Bit Volume

Now lets consider the same volume as above, but with one more slice of data, namely a 512x512x446 @ 16-Bits volume, again loaded on a 256 MB VP500 card. The volume is again divided into 18 different subvolumes, but the W dimension of 446 causes a leap in allocation in the W dimension. 31 slices in W must be wasted due to the **Storage Unit** granularity of 32 voxels in each dimension.

As before, the following table identifies each subvolume and its dimensions and the total number of **Storage Units** required in voxel memory.

Vol#	U_{min}	U_{max}	V_{min}	V_{max}	W_{min}	W_{max}	Units
0	0	255	0	255	0	255	512
1	253	508	0	255	0	255	512
2	506	511	0	255	0	255	64
3	0	255	253	508	0	255	512
4	253	508	253	508	0	255	512
5	506	511	253	508	0	255	64
6	0	255	506	511	0	255	64
7	253	508	506	511	0	255	64
8	506	511	506	511	0	255	8
9	0	255	0	255	253	445	448
10	253	508	0	255	253	445	448
11	506	511	0	255	253	445	56
12	0	255	253	508	253	445	448
13	253	508	253	508	253	445	448
14	506	511	253	508	253	445	56
15	0	255	506	511	253	445	56
16	253	508	506	511	253	445	56
17	506	511	506	511	253	445	7

Table 1-3 SubVolumes of a 512x512x446 @ 16 Bits Volume

Note how subvolumes 9, 10, 12 and 13 have jumped from 384 **Storage Units** to 448! This is the major cause of why this particular supervolume cannot be loaded whereas a volume with only one less slice can be loaded.

The following table shows how the subvolumes are allocated in order. After subvolume #7 is loaded, all voxel memory has been allocated and the remaining 6 subvolumes cannot be loaded. The rendering of this supervolume is possible, however, but only by swapping in the remaining 6 subvolumes which slows down render performance.

Vol#	Units	Bank	0	1	2	3	4	5	6	7
0	512	0	512							
1	512	1		512						
3	512	2			512					
4	512	3				512				
9	448	4					448			
10	448	5						448		
12	448	6							448	
13	448	7								448
2	64	4					512			
5	64	4						512		
6	64	5							512	
7	64	5								512
11	56	X								
14	56	X								
15	56	X								
16	56	X								
8	8	X								
17	7	X								
remain			0	0	0	0	0	0	0	0

Table 1-4 Bank Allocation for 512x512x446 @ 16 Bits Volume

2 Other Issues

This section discusses a few issues that may affect the results listed in this paper.

2.1 Volume Tile Size

The previous examples of dividing a supervolume into multiple subvolumes used a tile size of 256^3 voxels. This is the maximum sized subvolume that can be rendered by the VG500 ASIC at one time. It is possible to support smaller tile sizes that are also both a multiple of 32 (the minimum granularity of voxel storage in each dimension) and a power of 2 (for ease of dividing and certain required computations). The full set of tile sizes are the following:

- 32 voxels
- 64 voxels
- 128 voxels
- 256 voxels

These different tile sizes may be used in future releases of VLI to accomplish one or multiple of the following tasks:

- Allow better rendering performance when a supervolume is cropped or limited (as with active subvolumes) by only rendering the non-culled portions rather than all the subvolumes of the supervolume. Obviously, as tile size goes down, the performance improvements may be gained over smaller changes in areas of interest.
- Allow better load balancing in the system by overlapping rendering and software stitching of subvolume baseplanes more efficiently.

The use of smaller tile sizes, however, may have the following negative effects:

- Reduction in size of the maximum sized volume that can fit in voxel memory at one time. With smaller tile size comes more seams between subvolumes and thus overlap of voxels and redundant data.
- Reduction in performance when rendering all subvolumes of a supervolume. There will be more software intervention required to produce the final image as there are more subvolume baseplane images that must be stitched together in software.

The use of these alternate tile sizes is still "up in the air" as of the writing of this paper.

2.2 Padding Voxel Data

The VG500 ASIC does not produce pixels for all of the voxels of a volume. Some samples around all edges of the volume are considered invalid because they lack sufficient information to compute gradients that are used for lighting. One way of avoiding this is to "pad" the volume on all 6 sides with 2 voxels worth of data. This would allow the proper computation of gradients for all the intended voxels and only the padded voxel samples would be "thrown away".

Currently, VLI does not perform this padding. It is currently planned that the user/application must perform this padding if desired. This will result in a reduced effective size of the maximum sized volume that can be stored in VP500's voxel memory. With typical medical data sets, it may only be desirable to pad in the W dimension (add leading and trailing slices of data). This is because there are typically a number of open air voxels around the edges of the volume in the other 2 dimensions, U and V.

3 Summary

The following table lists the maximum loadable sized volume on VP500 cards at different tile sizes, slice sizes, voxel memory configurations and voxel sizes.

Tile Size	SizeU/V	Volume Memory	Bits/Voxel	Slices W
256	512	128MB	8	445
			16	224
		256 MB	8	887
			16	445
	509	128MB	8	509
			16	256
		256 MB	8	1015
			16	509
	516	128MB	8	445
			16	224
		256 MB	8	887
			16	445
128	512	128MB	8	439
			16	221
		256 MB	8	878
			16	439
64	512	128MB	8	430
			16	857
		256 MB	8	215
			16	430

Table 3-1 Loadable Volume Sizes across Configurations

Tile Size	SizeU/V	Volume Memory	Bits/Voxel	Slices W
32	509, 512 or 516	128MB	8	351
			16	177
		256 MB	8	728
			16	351
256	256	128MB	8	2,027
			16	1,015
		256 MB	8	4,051
			16	2,027

Table 3-1 Loadable Volume Sizes across Configurations

The following table lists the maximum loadable volume sizes on a 256 MB VP500 card for 16-bit voxel datasets at different tile sizes.

SizeU/V	Tile Size	Slices W
512 to 538	256	445
	128	439
	64	430
	32	351
509	256	509
	128	439
	64	430
	32	351
256	256	2,027
	128	1,564
	64	1,528
	32	1,453

Table 3-2 Loadable Volume Sizes @ 16 Bits/Voxel and 256 MB of Voxel Memory

