BITPLANE
an Oxford Instruments company

Imaris 9 Surfaces Rendering Technology for Large Images

Application Note

New imaging modalities along with tissue clearing and expansion techniques enable the acquisition of extremely large (100s of GBs) and complex images. Until now the automated analysis of these images was difficult or impossible. Imaris 9.0 revolutionizes analysis of these images with a new Surface rendering model as well as improved image handling for faster processing of Surfaces calculations.

Imaris 9.0 Surfaces Technology

Surfaces in Imaris are 3D models computed from 3D images by a sequence of pre-processing, segmentation, and connected component labelling steps. Surfaces are used to identify relevant entities within an image, to visualize them and to get measurements of interesting structures (Area, Volume, Intensity, Position, Ellipticity and many more).

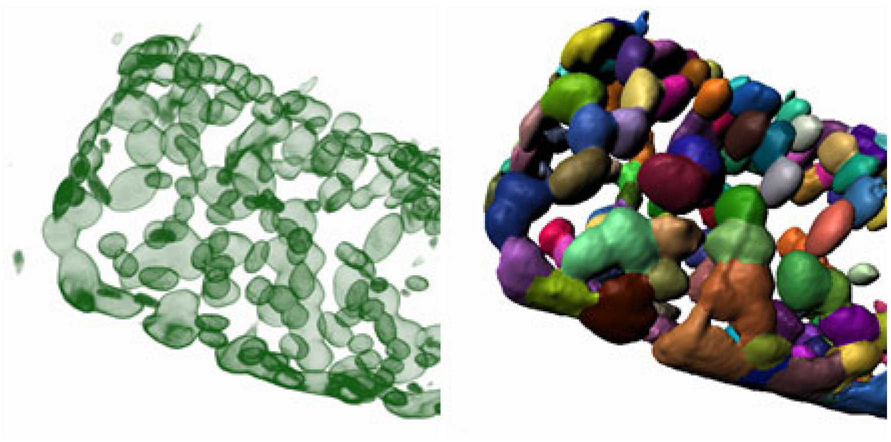


Figure 1 The panel above shows an Imaris example of Volume Rendering, Surfaces Model Rendering and the selection of measurements that user can get after Surfaces are detected.

Measurements from Surface

Ellipsoid Axis Length B	1,51	Intensity Center	80,00
Ellipsoid Axis Length C	4,79	Intensity Max	54,00
Ellipticity (oblate)	0,16	Intensity Max	95,00
Ellipticity (prolate)	0,80	Intensity Mean	19,50
Generation	0,00	Intensity Mean	49,43
Ellipsoid Axis Length A	1,33	Intensity Median	20,01
Intensity Center	22,00	Intensity Median	47,04

With Imaris 9.0 it is now possible to interactively render Surfaces from very large images which was either impossible in previous versions or would have required expensive dedicated hardware. The technology that enables fast rendering of very large surfaces is based on the following features:

- Multi-resolution format in 3D block-wise layout
- Renderer loads optimal surface resolution levels to match screen resolution
- Data Caching in VRAM (GPU) and RAM
- Multi-threading of rendering, decompression and loading

In combination, these features enable fast rendering of Surfaces of 100 GB size or more using commodity hardware. In the following paragraphs the above features will be explained in more details.

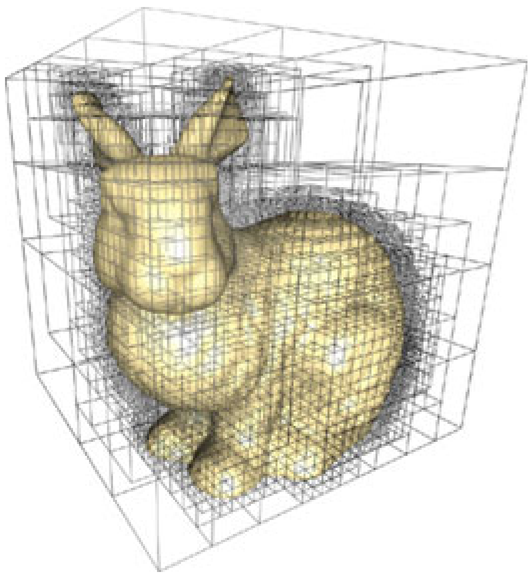


Figure 2 The figure shows how the Surface model is stored as a sparse octree where blocks of voxels are only subdivided if they are not entirely “inside” or entirely “outside”. Image reprinted from GPU Gems 2: Programming Techniques for High-Performance Graphics and General-Purpose Computation

Multi-resolution format in 3D block-wise layout

A Surface is a model that stores for each position in space whether it is “inside” or “outside” of the surface. To achieve this efficiently the model is stored as a **sparse octree 1,2,3,4** where blocks of voxels are only subdivided if they are not entirely “inside” or entirely “outside” as shown in Figure 2.

Each node of the octree receives “inside” or “outside” information for the Surface so that it is possible to retrieve information on any resolution level of the octree as shown in Figure 3.

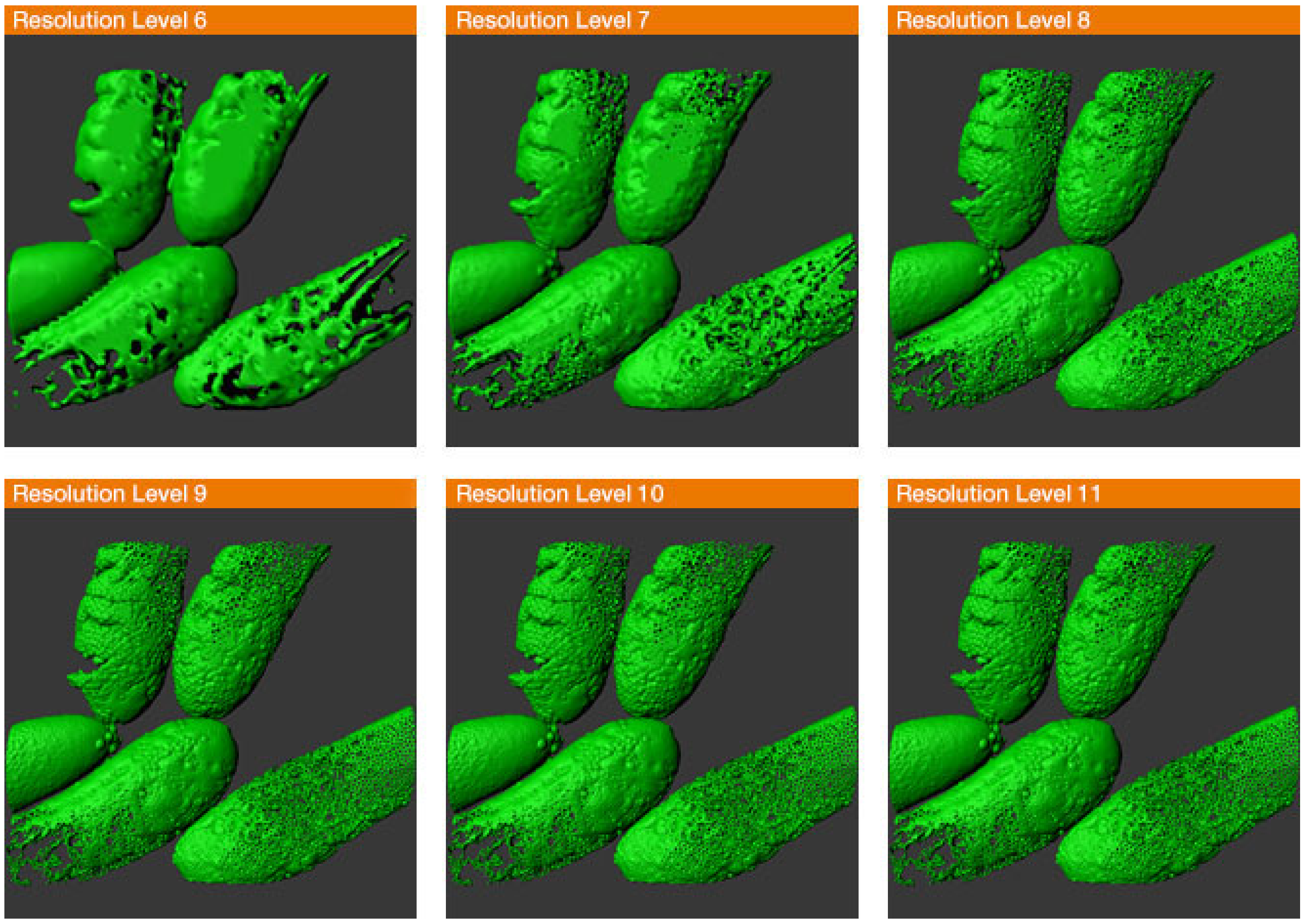


Figure 3 The above figure shows the same Surface model displayed using different resolution levels of an octree (resolution levels from 6 to 11). At

the resolution of this document you are looking at, the quality of the rendering does not improve from level 9 up. The best display for the given screen resolution is already achieved at level 9, while level 8 is close to optimal.

To achieve high quality rendering of the Surface model, the nodes of the sparse octree not only contain information about "inside" or "outside" but also encode information about the local surface geometry.

The above Figure 3 shows that at the resolution of this document the quality of the rendering does not improve from level 9 to the higher levels. Even though this Surface contains 11 levels, the best display is already achieved at level 9 while level 8 is close to optimal. The next section describes that an optimal choice of resolution is not just applied to a Surface as a whole, but it is actually applied block-by-block to every block of the octree.

Renderer loads optimal Surface resolution levels to match screen resolution

Based on the described sparse octree data structure the Imaris 9.0 Surface rendering is view-dependent, which means that for each part of the scene the renderer computes the optimal resolution required to match the projected resolution of each octree block to the resolution of the display. As a result, parts of a Surface that are far away from the camera will be rendered at lower resolution than parts that are very close to the camera as shown in the Figure 4 where each block of the octree is rendered with a different color. The adaptation of the Surface resolution to display resolution takes effect within one scene as shown in Figure 4. It also takes effect when the scene is rotated or zoomed and when the window size or the resolution of the display is changed.

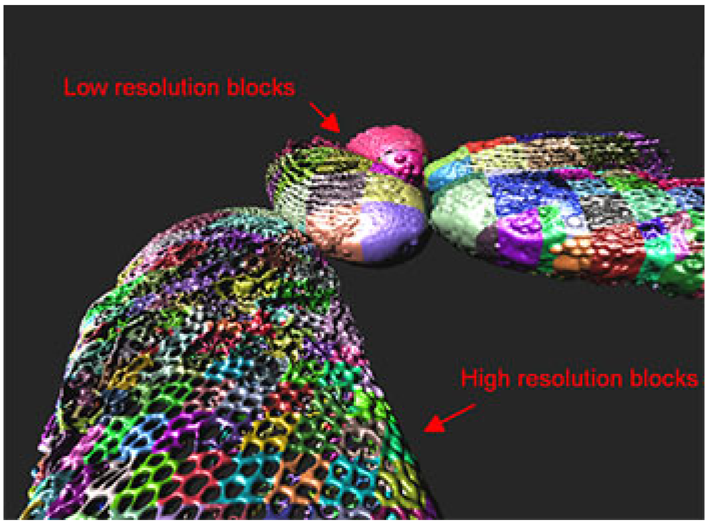


Figure 4. Imaris 9.0 Surface model in which each block of the octree is rendered with a different color. High resolution blocks are close to the camera and low resolution blocks are far away from the camera.

Data Caching in VRAM and RAM

When an IMS file that contains Surfaces is opened in Imaris 9.0 the renderer will demand blocks from the octree to be loaded into VRAM. These blocks will thus first be read from the file to RAM and then they will be copied from RAM to VRAM. Imaris 9.0 caches resolution blocks both in RAM and in VRAM so when the user modifies the view (rotates, pans or zooms), only those blocks that are not on the GPU already will have to be uploaded by the renderer to the GPU. In the same way it will only have to read those blocks that are not already in RAM.

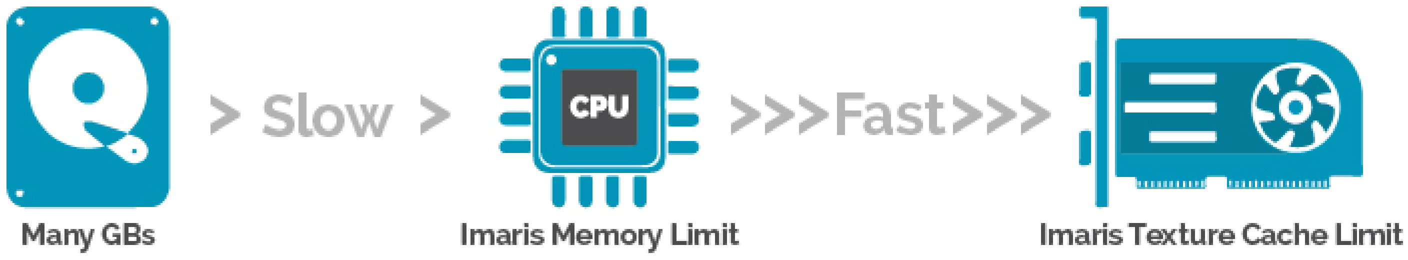
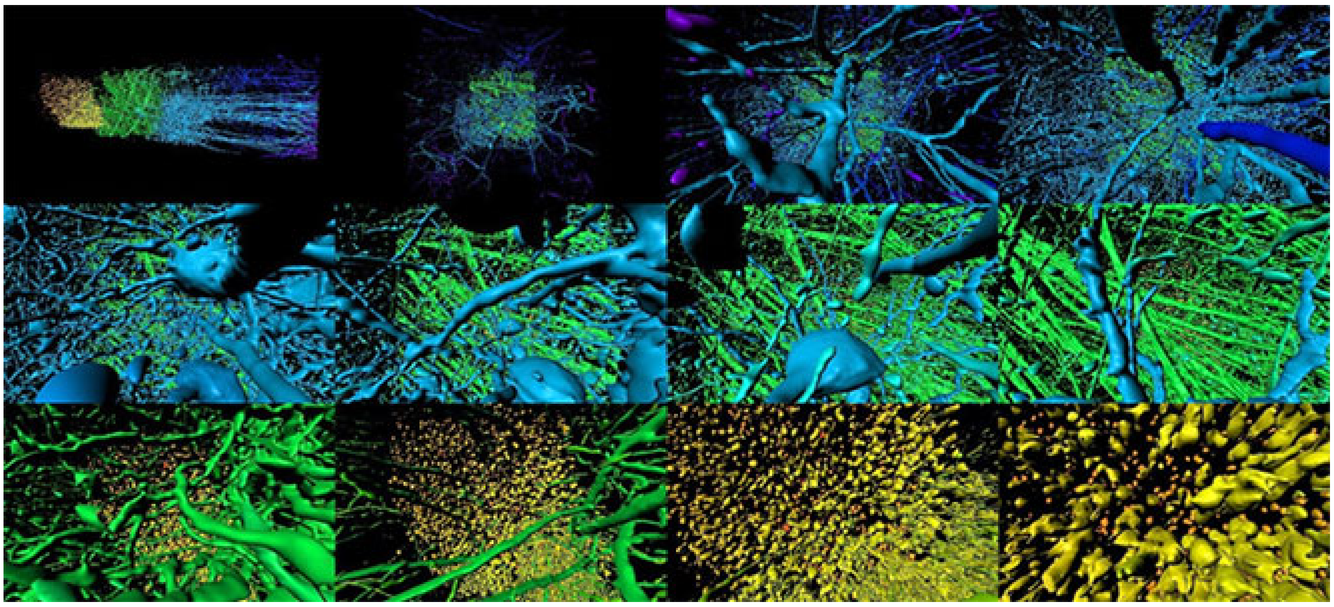


Figure 5. Data transfer from storage drives to RAM to VRAM. Caching minimizes the amount of transfer which is particularly slow from storage drives to RAM, but also time consuming between RAM and VRAM.

Multi-threading of rendering, decompression and loading

To achieve interactive frame rates for Surface rendering it is important that Imaris does not have to wait for data to be loaded from RAM or from the file which could take seconds or even minutes. At any point in time Imaris 9.0 renders the data that is available on the GPU at that moment. In parallel a loading thread uploads to the GPU the data that has been requested by the renderer with the assumption that data requested for the previous frame has a very high likelihood to be necessary for the next frame (an assumption that is often valid during rotation, pan and zoom of a scene). To prepare data as fast as possible the loader itself is again a pipeline consisting of a file reading thread and multiple decompression threads which all work in parallel.

With this technology Imaris 9.0 can render interactively massive Surfaces consisting of hundreds of Gigabytes of data.



* To gain a technical understanding of out of sparse octree voxel rendering we recommend the work of Laine and Karras 2010 , Crassin 2011, Demir and Westermann 2015 and Gibson 1998.

1. CRASSIN C.: GigaVoxels: A Voxel-Based Rendering Pipeline For Efficient Exploration Of Large And Detailed Scenes, PhD Thesis, Grenoble University 2011
2. DEMIR I., WESTERMANN R.: Vector-to-Closest-Point Octree for Surface Ray-Casting. Vision Modelling and Visualization 2015: 65-72
3. GIBSON S. F. F.: Using distance maps for accurate surface representation in sampled volumes. In Proceedings of the 1998 IEEE Symposium on Volume Visualization (1998), VVS ’98, pp. 23–30.
4. LAINE S., KARRAS T.: Efficient sparse voxel octrees. In Proceedings of the 2010 ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games (2010), I3D ’10, pp. 55–63.