NVIDIA® GVDB VOXELS 1.1

SAMPLE DESCRIPTIONS

NVIDIA(R) GVDB Voxels was developed to provide a GPU-based framework for sparse volumetric data storage, processing and rendering in the fields of motion pictures, 3D printing, and scientific visualization. GVDB is designed as a basic primitive for high quality, large scale, efficient data processing for research and industry. The **NVIDIA(R) GVDB SDK library** is released as an open source library with multiple samples to demonstrate a range of capabilities.

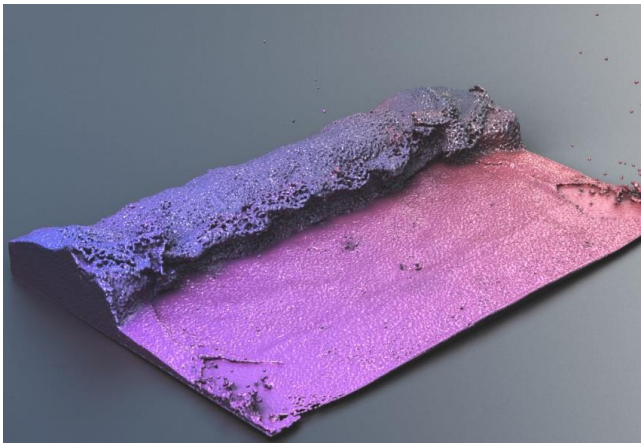
1. 3DPRINT

A useful application for volumes is 3D Printing. 3D printed parts have a well defined surface and interior making them ideally suited for sparse storage. While parts are often defined by the user as polygons, voxels enable a broader range of operations. A common task in 3D Printing is to extract the cross-sections of a part. This sample uses GVDB's polygon-to-voxel conversion, a fast hierarchical algorithm, to convert a polygonal model to sparse voxels, and then generates a sequence of cross-sectional slices which are saved as png files. The sample also renders a picture of the part as a trilinear surface for reference.

2. DEPTHMAP

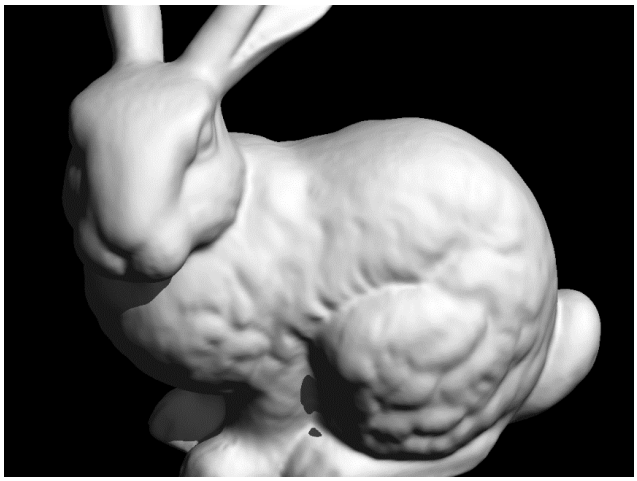
This sample demonstrates depth merging of an OpenGL rendered scene with a GVDB volume. OpenGL is rendered with FBOs into a depth buffer which is used as input to indicate when rays should terminate. The resulting GVDB buffer is then alpha-blended over the OpenGL scene to create a depth merged rendering. This example may be used as the basis for integrated rendering of sparse volumes with arbitrary OpenGL scenes.

3. FluidSurface



This in situ point-to-voxels surfacing example uses the open source Fluids v.3 SPH simulator (Rama Hoetzlein, 2011) to model a dynamic fluid using millions of particles. The moving particles are used to dynamically reconstruct a GVDB sparse volume on every frame, convert from points-to-voxels, and then directly raytrace the resulting voxel surface without polygons. The sample demonstrates core features of GVDB such as dynamic topology construction on the GPU and integrated OptiX raytracing. It shows how to clear and recreate a new topology without having to re-allocate the GVDB memory pools.

4. ImportVDB



GVDB 1.1 enables support for loading of sparse volume data from OpenVDB 4.0.0. In this example, the bunny model data set is imported and rendering as a level set using GVDB Voxels. To enable OpenVDB support, download the OpenVDB library build for windows and set the USE_OPENVDB flag during cmake build of the GVDB library, then build and run this sample.

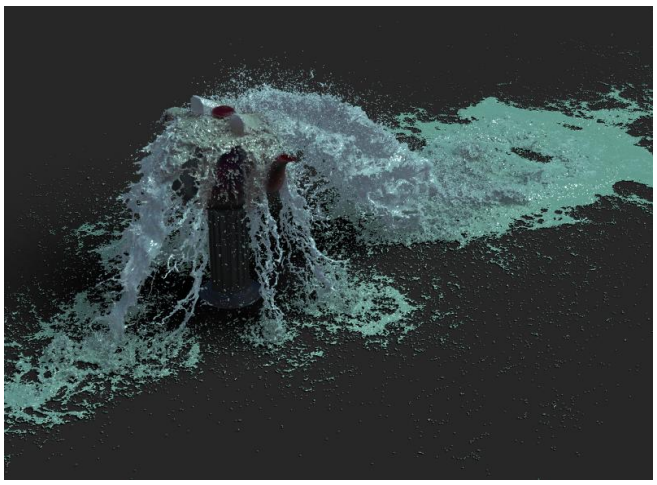
5. InteractiveGL

This sample demonstrates how to use GVDB to do volume rendering interactively in OpenGL. Windowing and OpenGL contexts are first created by sample utility helpers. A full screen texture is allocated in OpenGL. During the interactive display loop, GVDB renders the volume to a CUDA output buffer. This buffer is then transferred to the OpenGL screen texture using ReadRenderTexGL, which performs a fast gpu-to-gpu transfer of the CUDA buffer to the GL texture. A final call to renderScreenQuadGL renders the output texture to the window. This sample demonstrates real-time interactive rendering of sparse volumes.

6. InteractiveOptix

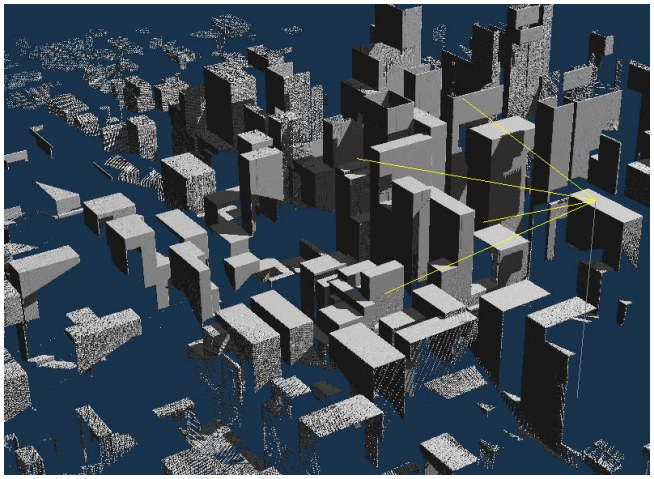
The InteractiveOptix sample shows how to mix GVDB with OptiX to support high quality scattering, soft shadows, and generic raytracing of sparse volumes. This flexible sample abstracts scene creation with an OptixScene class that can add multiple gvdb volumes to an optix graph. The sample illustrates scattering between a sparse volume (cloud) and a polygonal model. New OptiX vol_intersect programs, provided with the sample, support intersections of trilinear isosurfaces, level set surfaces, and deep volume sampling via integration with the GVDB CUDA raycasting functions. These intersection programs enable rays to hit either polygons or volumes, for seamless operation with shading and scattering programs. The sample is interactive, with 3 rays per pixel (primary, shadow, bounce), with convergence to an anti-aliased result over multiple sample-frames.

7. PointCloud



Rendering of point clouds is an important application in many fields. While points can be rendered directly, conversion to voxels enables high quality surfaces, hole filling, and other operations. This highly flexible example loads a scene file which specifies polygonal models and point cloud data to be read from disk in binary format. The points are used to construct a sparse GVDB topology of voxels, then converted from points to a level set volume, and finally raytraced using OptiX for high quality reflections and shading.

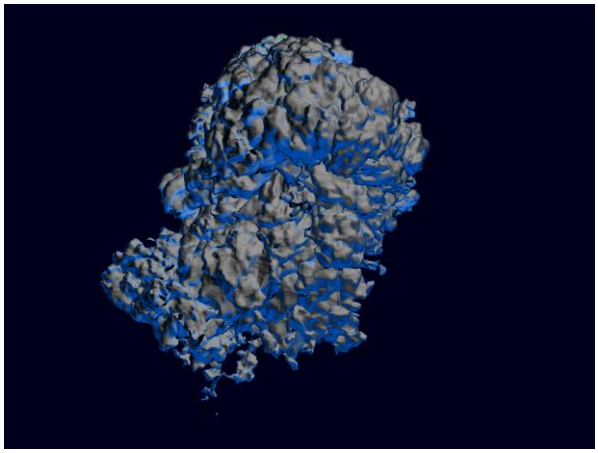
8. PointFusion



Voxels are well suited to certain problems in computer vision and robotics. One useful application area is the modeling of LIDAR and range data, which provide a cloud of points in real-time that represent the immediate surroundings of a vehicle or drone. By fusing the points over time into a volume it is possible to construct a global 3D map which improves as more data is collected. This sample generates a hypothetical city as a procedural model, and then raytraces it on-the-fly to emulate the range data acquired from a drone. It then uses this range data as input to dynamically update a GVDB volume, fuses the new points into the volume, and then displays the resulting 3D map with voxel raytracing. All steps are performed in real-time so that a three dimensional map is constructed interactively. This sample could be adopted to insert real data from a vehicle or robot.

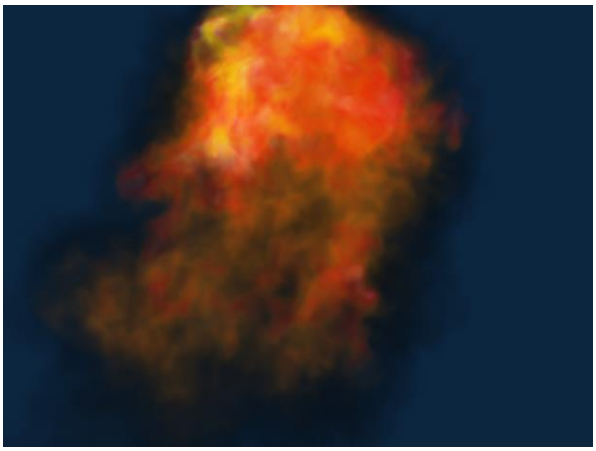
9. RenderKernel

Custom user-defined rendering kernels can be provided to the GVDB library. Here a user-defined CUDA kernel is invoked to render a VBX data volume which is loaded from disk. The custom kernel first performs surface raytracing with GVDB, and then uses the resulting hit and normal information to define a customized rendering appearance. In this example the reflection vector is calculated to simulate a chrome-like surface appearance.

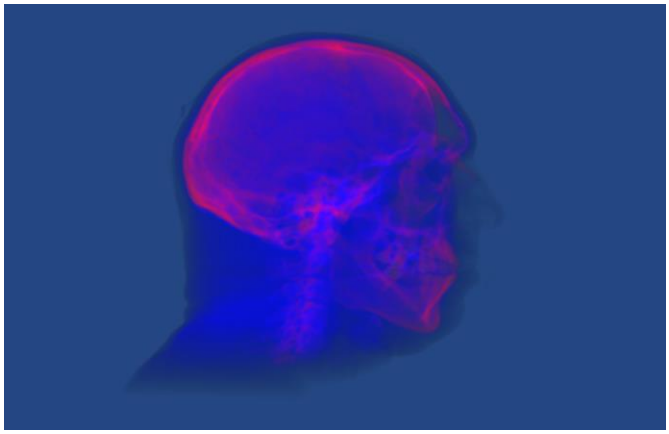


10. RenderToFile

This sample demonstrates how to load a VBX file and produce a raytraced image which is saved to disk as a png image file. Volume rendering is performed in pure CUDA, and the sample runs from the command-line without a graphics context. An arbitrary transfer function is specified with piecewise linear interpolation. This sample also writes out the GVDB data structure information, showing detailed statistics on how the volume is stored in memory.

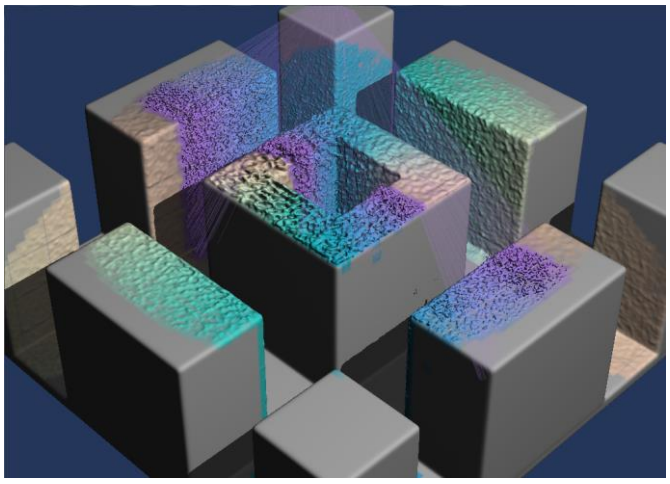


11. RESAMPLE



Medical data produced from MRI or CT scans is often generated as a dense volume or *.raw* file. However the area outside the body is typically empty air, making it suitable for sparse storage with GVDB Voxels. This example shows how a dense volume can be resampled to produce a sparse volume that gives the same result with a lower memory footprint. The data is efficiently retrieved by first down-sampling to determine which GVDB bricks must be sparsely activated in the topology. Then, a second pass loads only those bricks that contain data in the original model. In this way the performance depends on the sparse volume rather than the original dense model. The final result is raytraced in real-time with a custom transfer function to show the skull and surrounding tissue.

12. SPRAYDEPOSIT



Process engineering is the simulation of interactions between materials. This sample demonstrates *spray deposition* of a paint-like material onto a solid part. The particle spray is simulated by a number of rays that originate from a spray wand, which is modeled as a coherent, rotating bundle of rays above the part. The gvdb::Raytrace function provides a way to trace arbitrary rays and return their hit points on a sparse volume. The ray hit positions are then particle splatted into the volume to build up deposited material using the gvdb::SplatParticles function. To emulate the high viscosity nature of paint, a smoothing effect is applied which erodes the deposited material as the simulation proceeds. This sample also draws the rays in OpenGL, and demonstrates drawing of the GVDB tree topology.