NVIDIA® GVDB Sparse Volumes SDK Samples [BETA]

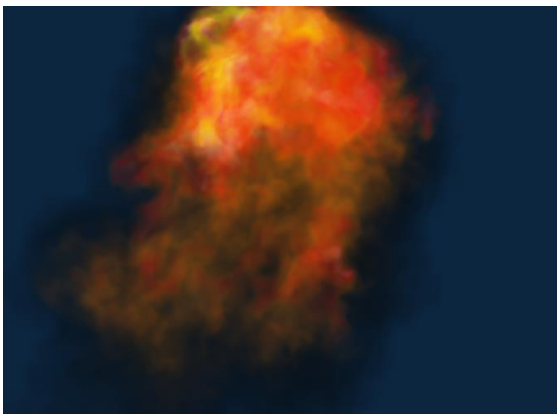
NVIDIA(R) GVDB was developed to provide a GPU-based framework for sparse volumetric data storage, processing and rendering in the fields of motion pictures, 3D printing, and scientific visualization. GVDB is designed as a basic primitive for high quality, large scale, efficient data processing for research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

This Beta release represents a subset of samples that will be included in the final public release of GVDB SDK 1.0. Note that we may make changes to the API and samples between Beta and Final releases.

Requirements

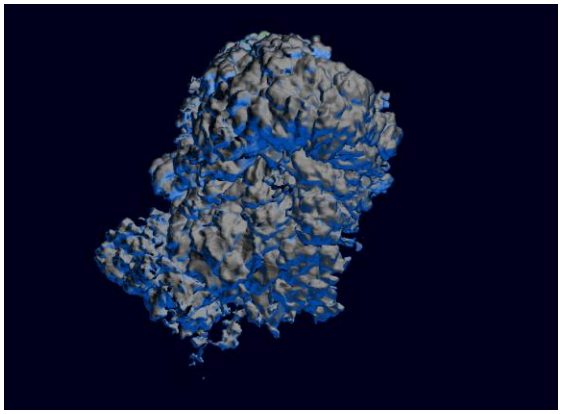
- NVIDIA Kepler generation or later GPU
- CUDA Toolkit 7.5 (Windows package). Please obtain this separately under https://developer.nvidia.com/cuda-75-downloads-archive prior to building and running GVDB and samples
- Windows 7, 8, 10 64-bit
- Microsoft Visual Studio 2010 or 2012
- CMake-GUI 2.7 or later

RenderToFile



This sample demonstrates how to load a VBX file and produce a ray-traced image which is saved to disk as a png image file. Volume rendering is performed in 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.

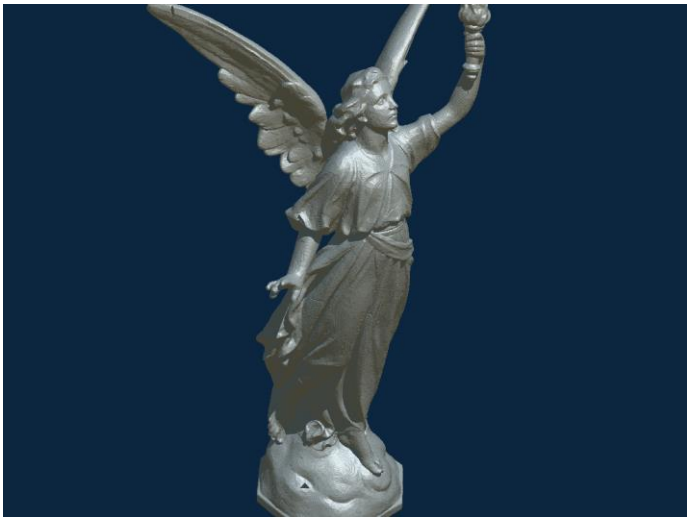
2. RenderKernel



Custom user-defined rendering kernels can be provided to the GVDB library. This sample shows how a user-defined CUDA kernel is invoked to render a VBX data volume. 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 appearance.

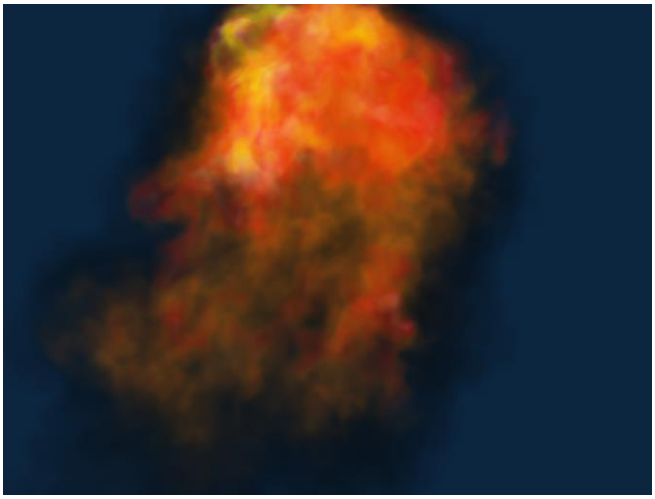
3. 3DPrint

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.



The sample generates a part model that is 100mm high, with a voxel size of 50 microns, giving an effective resolution of 2000 voxels high.

4. 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 spare volumes.

5. InteractiveOptix

OptiX 3.9.0 required (Included in GVDB SDK package)

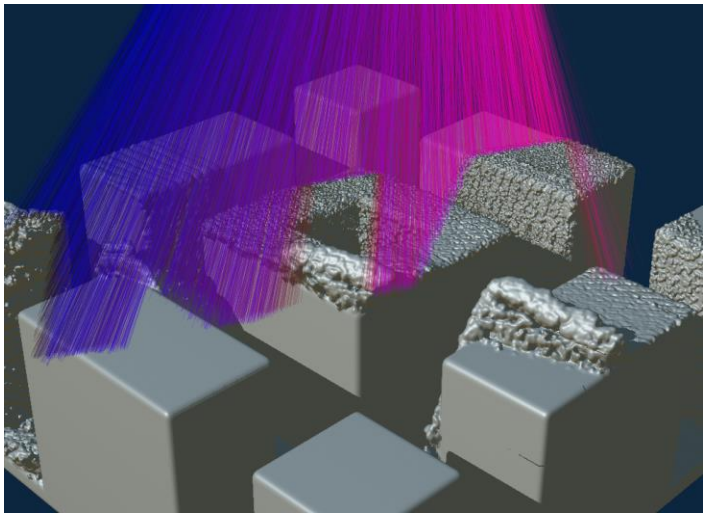
The InteractiveOptix sample shows how to mix GVDB with OptiX to support high quality scattering, soft shadows, and generic raytracing of sparse volumes. This flexible sample abstracts scene creation with an OptixScene class that can add multiple gvdb volumes to an optix graph. The sample illustrates 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.

6. IMPORTVDB

OpenVBD for Windows required (Please obtain separately under https://github.com/rchoetzlein/win_openvdb)

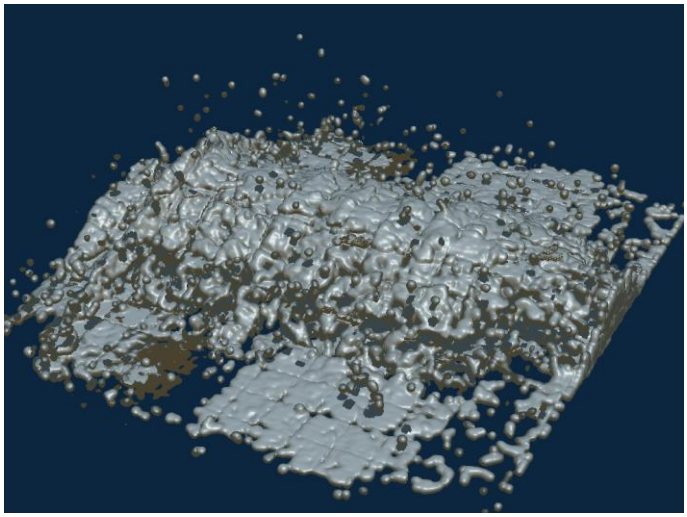
This import sample demonstrates compatibility with OpenVDB by loading .vdb files into the GVDB library. With only two lines of code, LoadVDB and SaveVBX, it is possible to load an OpenVDB formatted file and save to a VBX file for use with GVDB. Later, the data can be read again with LoadVBX. A quick reference for the open VBX file format specification is provided in the GVDB_FILESPEC.txt, and the GVDB Programming Guide gives a detailed list of benefits of the VBX format for GPU sparse volumes. Since the source data is a level set surface, the remainder of the sample performs a level set rendering, and saves the output image to a png file on disk.

7. SPRAYDEPOSIT



A useful application of GVDB Sparse Volumes is simulated modeling and design. This simulation sample demonstrates *spray deposition* of paint or other materials on 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.

8. FLUIDSIM



This in-situ simulation sample uses the open source Fluids v.3 SPH simulator (Rama Hoetzlein, 2011) to model a dynamic fluid using millions of particles. The particles are then used to dynamically construct a GVDB sparse volume on every frame, splat the particles into the volume, and then raytracing them for display. The sample demonstrates core features of GVDB such as topology construction using the gvdb::ActivateSpace and gvdb::FinishTopology functions. The sample shows how to use these functions to clear and recreate a new topology without having to re-allocate the GVDB memory pools.