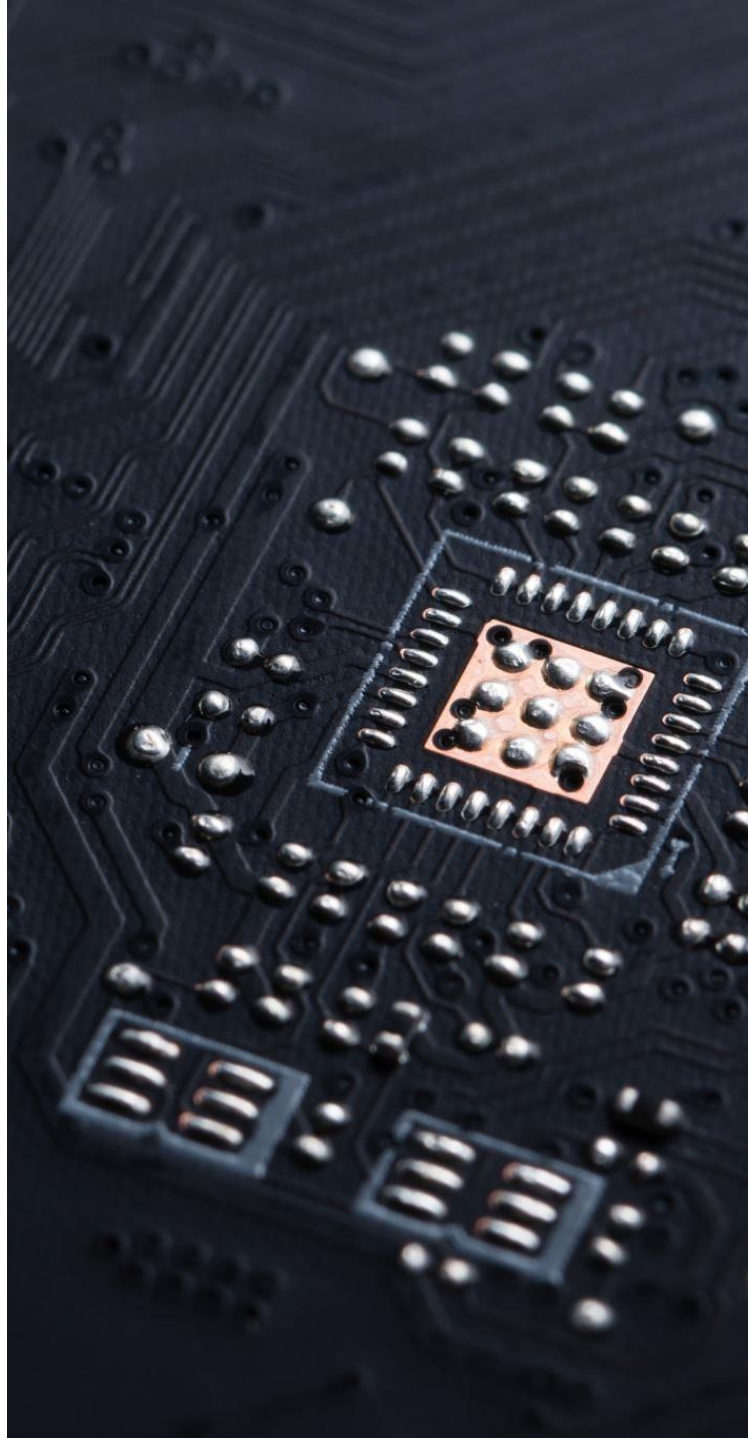
EVALUATION OF DIRECT LEVEL-SET RAY TRACING PERFORMANCE ON GPUS FOR PROCESS SIMULATION APPLICATIONS

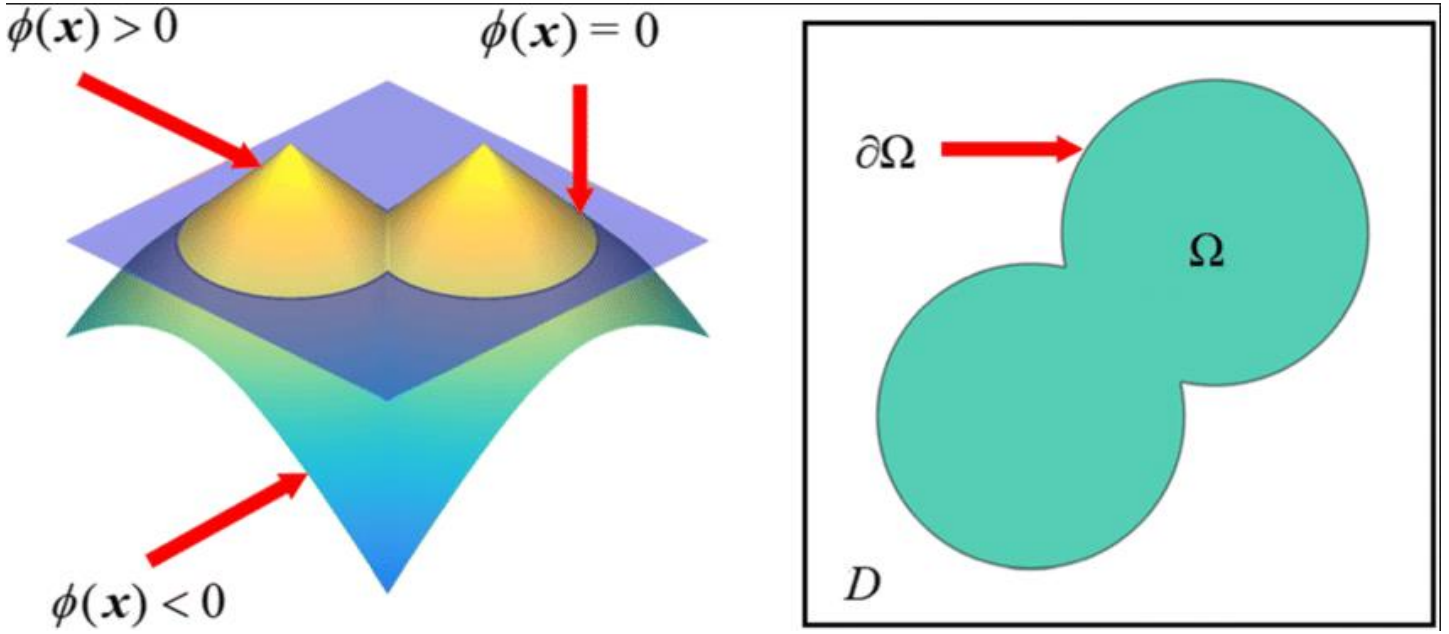
360.245 SELECTED TOPICS - COMPUTATIONAL ELECTRONICS

MARIO HITI



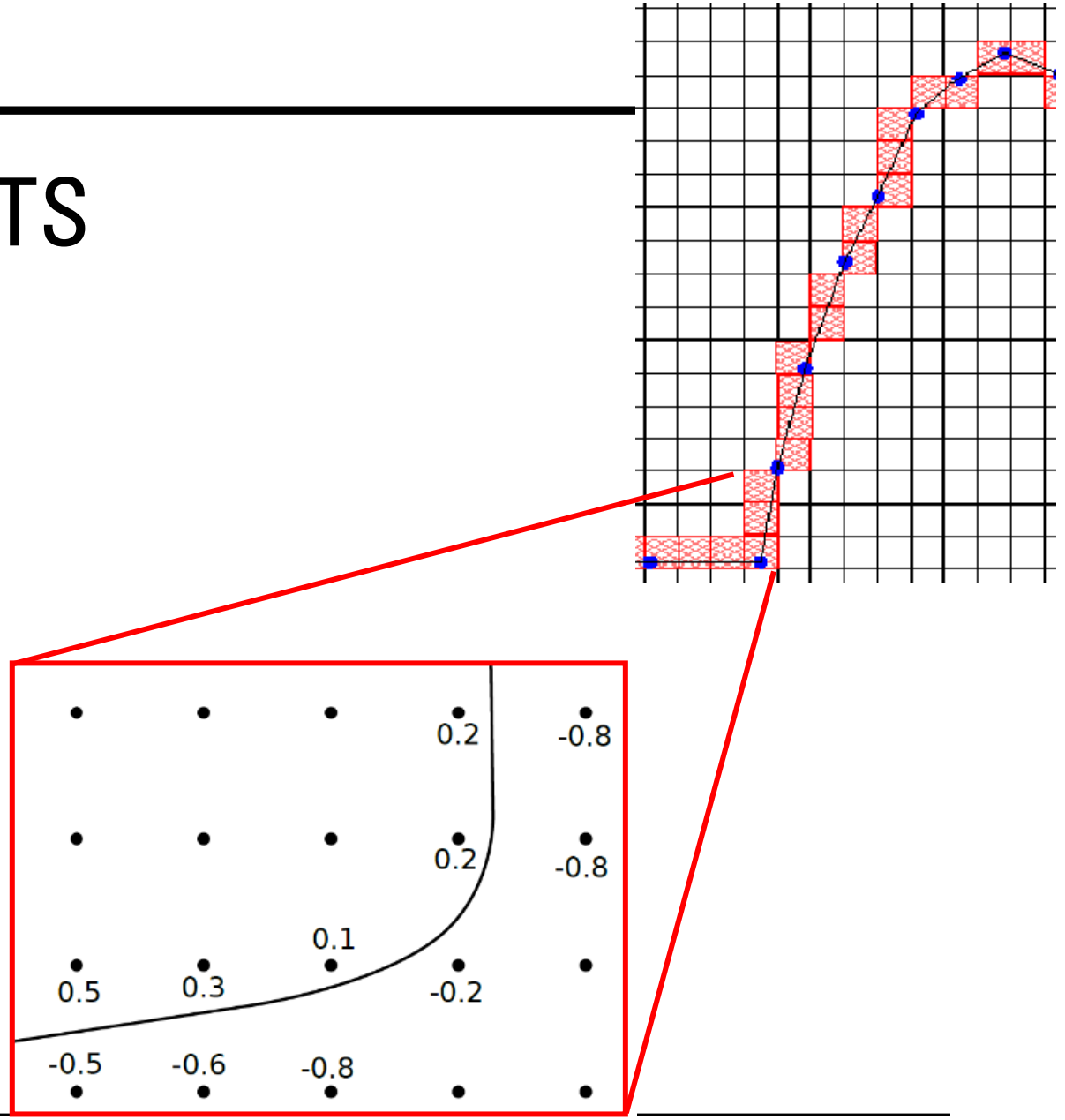
SIGNED DISTANCE FUNCTION Φ

- *Φ(x)* determines the shortest distance between x and the surface *∂Ω*
- Sign determines if point is on the inside or outside
- Very easy to simulate surface advection

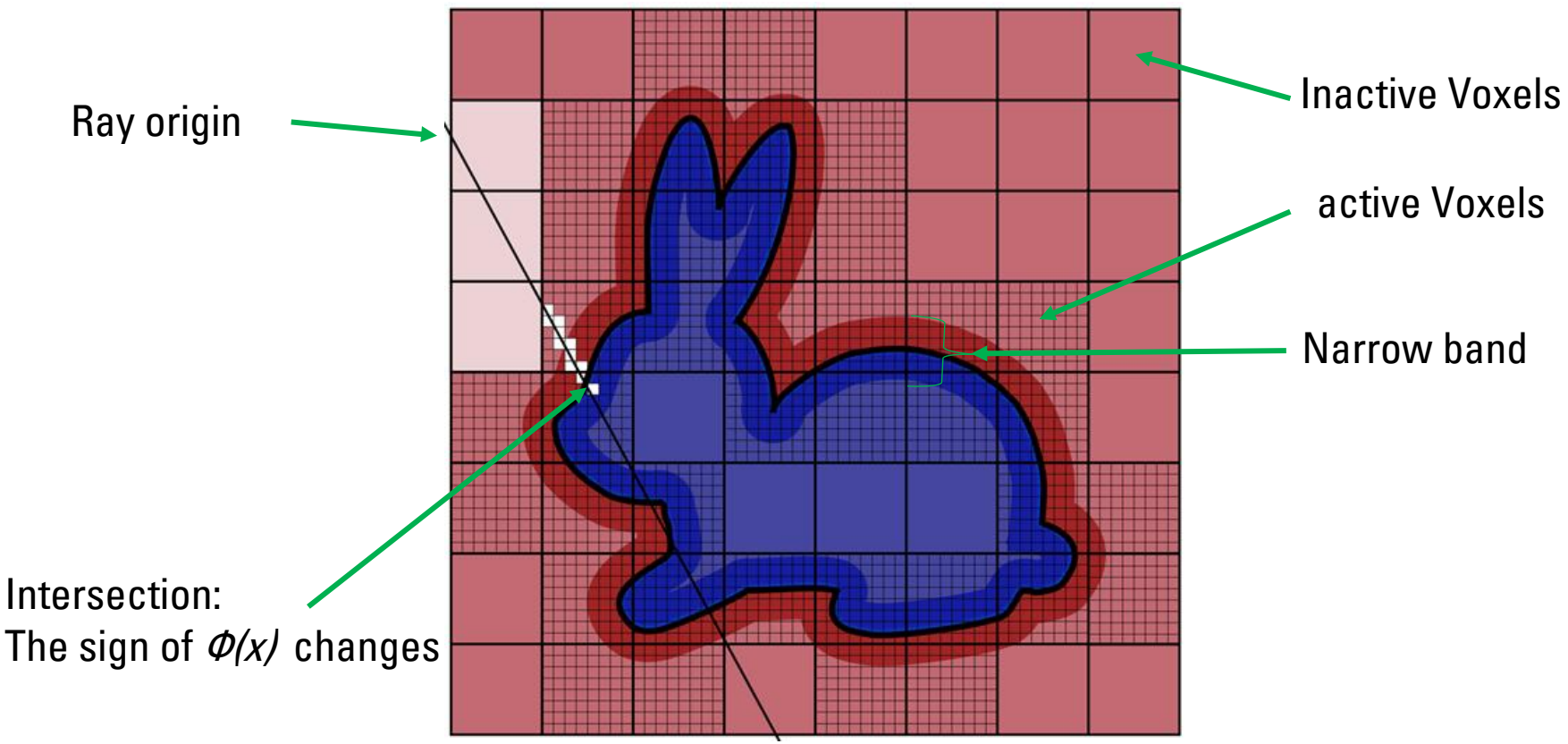


NARROW BAND LEVEL SETS

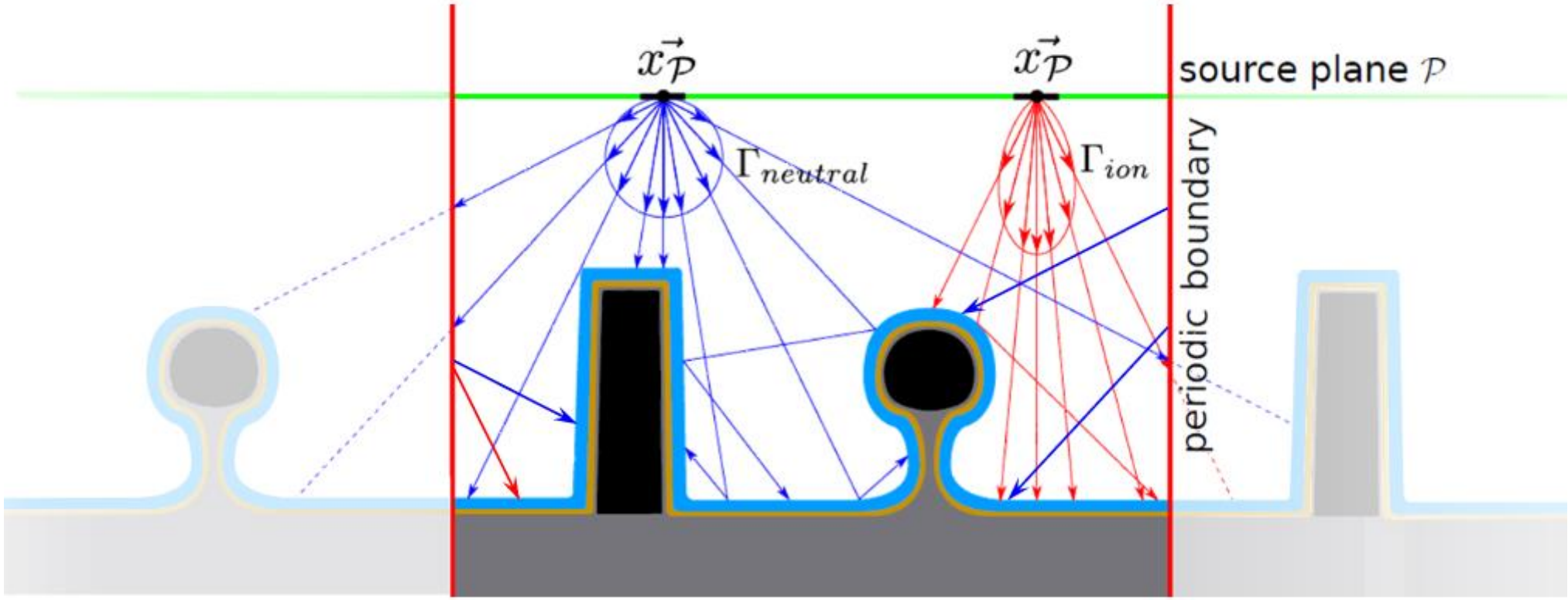
- Store values of $\Phi(x)$ in a discrete grid
- $\Phi(|x| >> 0)$ irrelevant
  - only store values near the surface boundary
  - All other voxels are inactive
- Very well suited for raytracing



RAYTRACING


Ray origin
Inactive Voxels
active Voxels
Narrow band
Intersection:
The sign of $\Phi(x)$ changes

APPLICATIONS



FRAMEWORKS

OpenVDB

- Released in 2012
- Mature and rich in features
- CPU only
- Allows changes to the narrow band
- Complex build process
- Several dependencies

NanoVDB

- Released in 2021
- Many features missing or incomplete
- **CPU and/or GPU (CUDA)**
- Recompute everything when narrow band changes
- Header only
- No dependencies

GOAL

- NVIDIA released a benchmark promising a 60x increase in performance

	OpenVDB (TBB)	NanoVDB (TBB)	NanoVDB (CUDA)
LevelSet	148.182	11.554	2.427
FogVolume	243.35	223.195	7.71
Collision	–	120.324	10.131

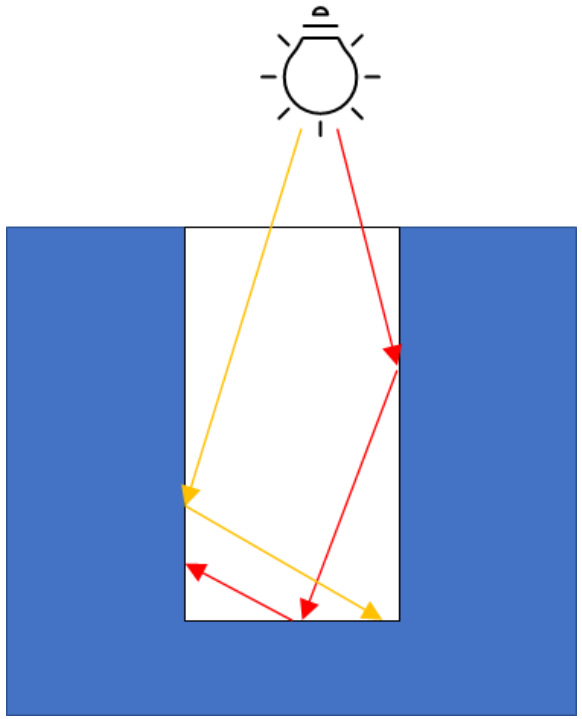
60x faster!

- **Can we expect the same performance for our simulations?**

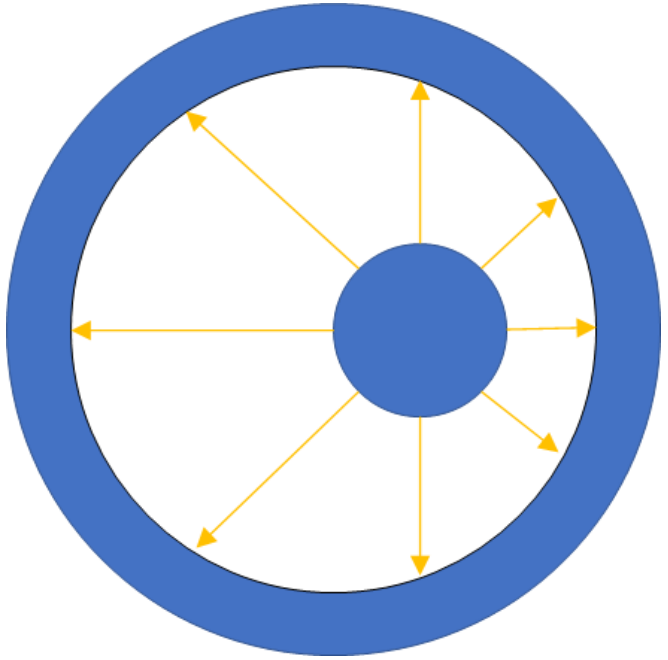
https://developer.nvidia.com/blog/accelerating-openvdb-on-gpus-with-nanovdb

THE WORST-CASE SCENARIO

Typical Application



Modified Setup



Workflow

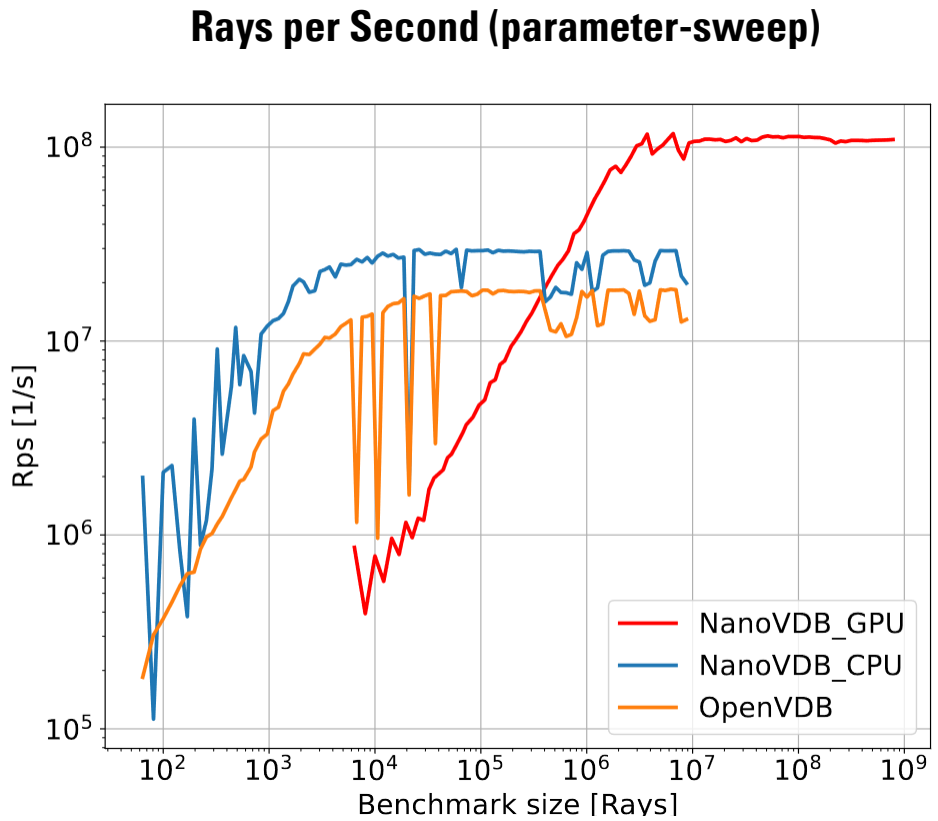
1. Pre-comupte n rays
2. **Calculate intersections**
3. Verify results
4. Increase n and goto 1.

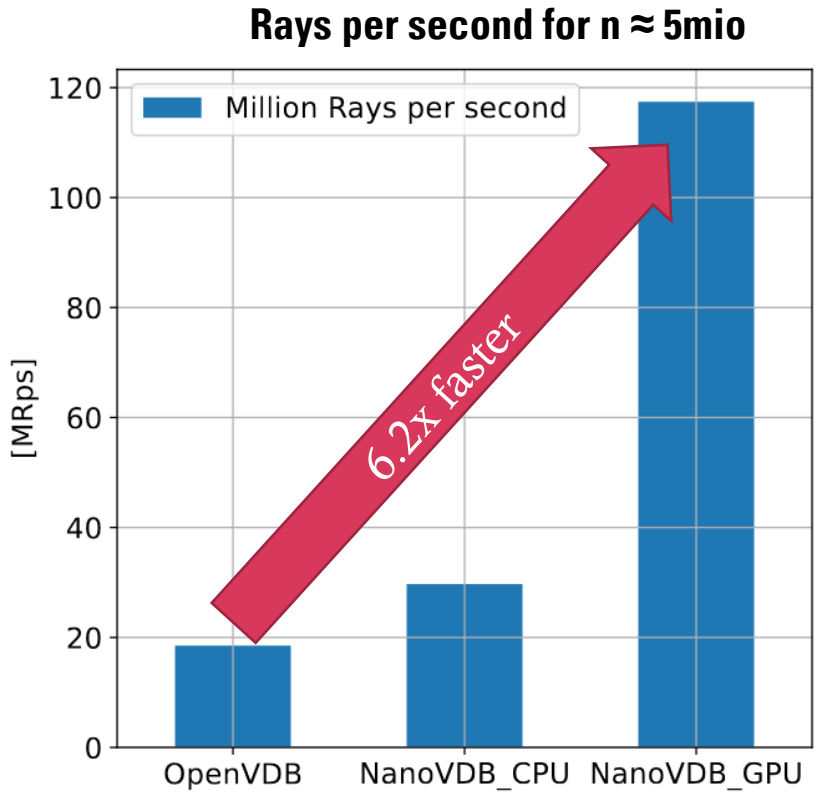
HARDWARE

- Benchmark performed on a TCAD Cluster provided by IuE

Hardware	Price	Power Consumption	Cores
Intel Xeon 6248	3.300 €	105W	20 Cores, 40 Threads
NVIDIA Tesla T4	3.000 €	70W	2.560 CUDA-Cores

BENCHMARK RESULTS





NANOVDB: THE GOOD

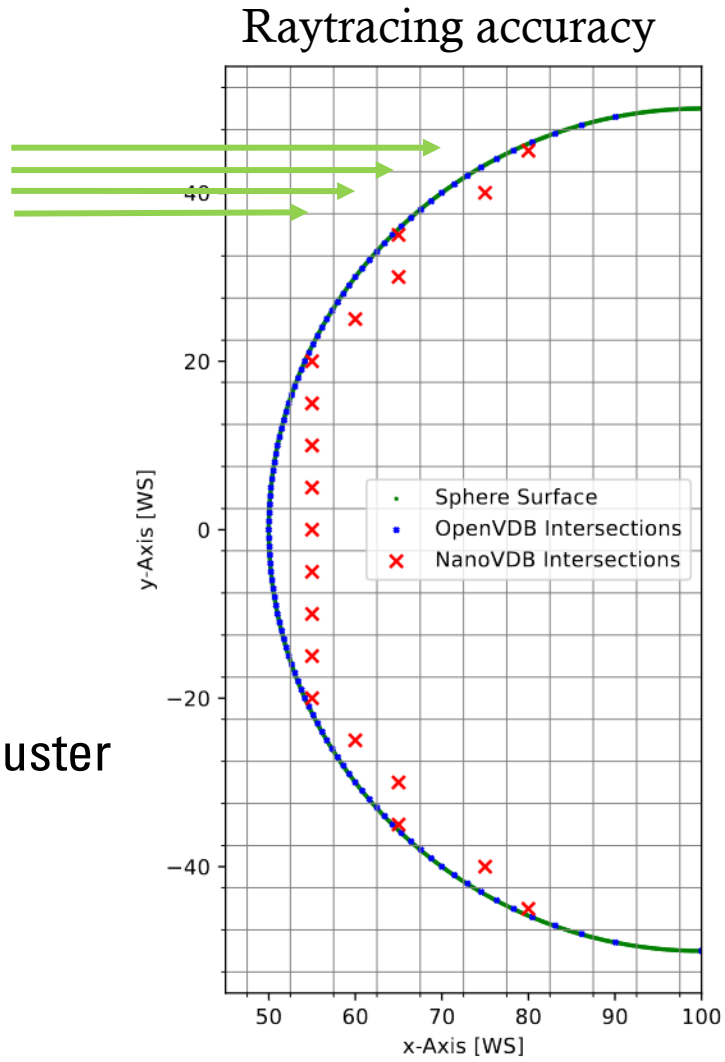
- Performance is great
- Very good code quality, no bugs encountered
- Target platform is determined at compile time
  - Same code for CPU and GPU
  - Develop on PC (no GPU necessary), deploy on cluster
  - Calculation can run on CPU and GPU in parallel → add performance of both

NANOVDB: THE BAD

- No boundary conditions (periodic or reflective)
  - ✓ Can be implemented

- Raytracing is inaccurate (see image)
  - ✓ Can be improved

- Realistically OpenVDB is still needed → difficult to compile on cluster
  - ✓ Can be automated



NANOVDB: THE UGLY

- Difficult to learn
- poor documentation (especially for NanoVDB)
- Limited compatibility between OpenVDB and NanoVDB
- Lots of template meta-programming and boiler-plate code

```
openvdb::tools::LevelSetRayIntersector<OpenGridT, openvdb::tools::LinearSearchImpl<OpenGridT, 0, float>,
                                       OpenGridT::TreeType::RootNodeType::ChildNodeType::LEVEL, OpenRayT>
    open_intersector(open_grid);
```

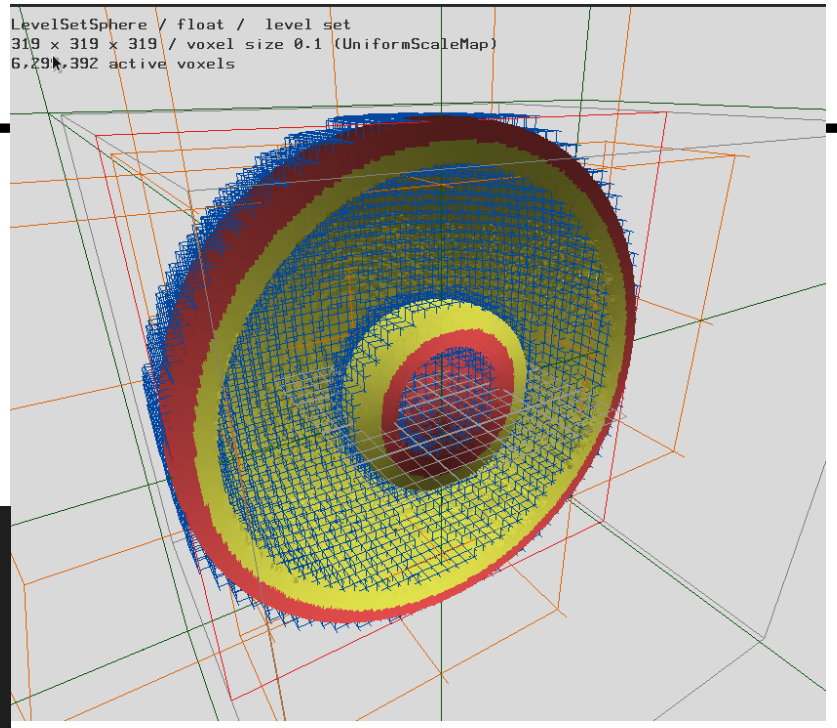
OUTLOOK AND FURTHER TOPICS

- Expand NanoVDB for process simulations
  - Implement reflections
  - Implement boundary conditions
  - Increase accuracy of raytracing
  - Moving ray sources
  - …
- Further improve performance
  - branchless programming
  - Use more GPU features such as CUDA streams
  - Multiple GPUs
  - Combine CPU + (multiple) GPU
  - RTX?
  - …

```
inline __hostdev__ bool ZeroCrossing(RayT& ray, AccT& acc, Coord& ijk,
{
    if (!ray.clip(acc.root().bbox()) || ray.t1() > 1e20)
        return false; // clip ray to bbox
    static const float Delta = 1.0001f;
    ijk = RoundDown<Coord>(ray.start()); // first hit of bbox
    HDDA<RayT, Coord> hdda(ray, acc.getDim(ijk, ray));
    const auto        v0 = acc.getValue(ijk);
    while (hdda.step()) {
        ijk = RoundDown<Coord>(ray(hdda.time() + Delta));
        hdda.update(ray, acc.getDim(ijk, ray));
        if (hdda.dim() > 1 || !acc.isActive(ijk))
            continue; // either a tile value or an inactive voxel
        while (hdda.step() && acc.isActive(hdda.voxel())) { // in the
            v = acc.getValue(hdda.voxel());
            if (v * v0 < 0) { // zero crossing
                ijk = hdda.voxel();
                t = hdda.time();
                return true;
            }
        }
    }
    return false;
}
```

QUESTIONS

```
cmake \
    -D OPENVDB_BUILD_CORE=ON \
    -D OPENVDB_BUILD_BINARIES=ON \
    -D OPENVDB_BUILD_VDB_PRINT=OFF \
    -D OPENVDB_BUILD_VDB_LOD=OFF \
    -D OPENVDB_BUILD_VDB_RENDER=OFF \
    -D OPENVDB_BUILD_VDB_VIEW=OFF \
    -D OPENVDB_BUILD_UNITTESTS=OFF \
    -D OPENVDB_BUILD_VDB_RENDER=OFF \
    -D OPENVDB_BUILD_NANOVDB=ON \
    -D OPENVDB_INSTALL_CMAKE_MODULES=ON \
    -D OPENVDB_USE_DEPRECATED_ABI=ON \
    -D OPENVDB_FUTURE_DEPRECATION=OFF \
    -D TBB_INCLUDEDIR=$TBB_DIR/include \
    -D TBB_LIBRARYDIR=$TBB_DIR/lib/intel64/gcc4.8 \
    -D BLOSC_INCLUDEDIR=$BLOSC_BUILD_DIR/include \
    -D BLOSC_LIBRARYDIR=$BLOSC_BUILD_DIR/lib \
    -D CMAKE_INSTALL_PREFIX=$OPENVDB_INSTALL_DIR \
    -D CMAKE_INSTALL_LIBDIR=lib \
    -B $OPENVDB_BUILD_DIR \
    -S $OPENVDB_DIR
make -C $OPENVDB_BUILD_DIR -j$NJOBS
make -C $OPENVDB_BUILD_DIR install
```



minal Help NanoVDB.h - SelectedTopicsCompElectron

hmarker.hpp benchmarker.cpp nanoVDB_GPU.cu

usr > local > include > nanovdb > C NanoVDB.h > {} nanovdb >

```
1419        return Vec3T(fma(static_cast<double>(xyz
1420                         fma(static_cast<double>(xyz
1421                         fma(static_cast<double>(xyz
1422    }
1423
1424    template<typename Vec3T>
1425    __hostdev__ inline Vec3T matMult(const float
1426    {
1427        return Vec3T(fmaf(xyz[0], mat[0], fmaf(x
```

Exception has occurred. ×

Segmentation fault

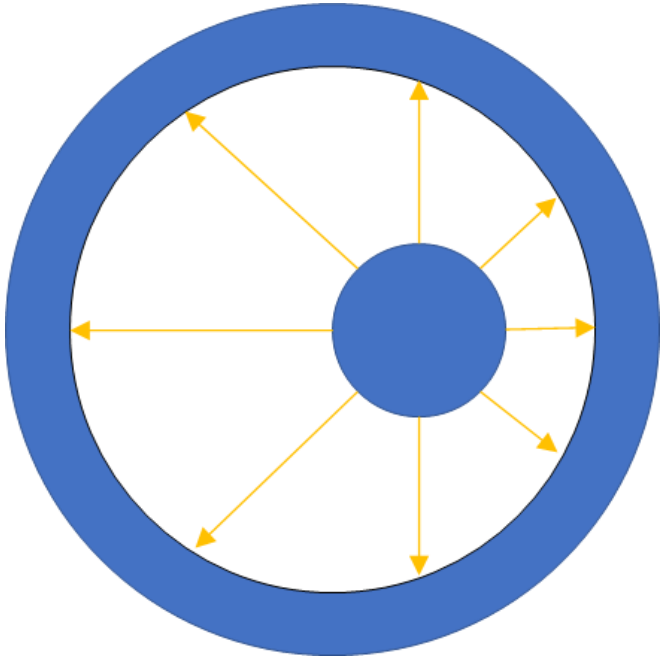
```
1428                         fmaf(xyz[0], mat[3], fmaf(x
1429                         fmaf(xyz[0], mat[6], fmaf(x
1430    }
```

```
Benchmarker::run_openVDB(size_t)':
/home/hiti/Workspace/SelectedTopicsCompElectronics/src/benchmarker/benchmarker.cpp:69:56: error: no matching
function for call to 'openvdb::v9_0::tools::LevelSetRayIntersector<openvdb::v9_0::Grid<openvdb::v9_0::tree::T
ree<openvdb::v9_0::tree::RootNode<openvdb::v9_0::tree::InternalNode<openvdb::v9_0::tree::InternalNode<openvdb
::v9_0::tree::LeafNode<float, 3>, 4>, 5> > > > >::intersectsWS(__gnu_cxx::__alloc_traits<std::allocator<openv
db::v9_0::math::Ray<float> >, openvdb::v9_0::math::Ray<float> >::value_type&, __gnu_cxx::__alloc_traits<std::
allocator<openvdb::v9_0::math::Vec3<float> >, openvdb::v9_0::math::Vec3<float> >::value_type&)'
   69 |     ray_intersector.intersectsWS(rays[i], calculated[i]);
      |                                  ^

In file included from /home/hiti/Workspace/SelectedTopicsCompElectronics/src/benchmarker/benchmarker.hpp:5,
                 from /home/hiti/Workspace/SelectedTopicsCompElectronics/src/benchmarker/benchmarker.cpp:1:
/usr/local/include/openvdb/tools/RayIntersector.h:164:10: note: candidate: 'bool openvdb::v9_0::tools::LevelS
etRayIntersector<GridT, SearchImplT, NodeLevel, RayT>::intersectsWS(const RayType&) const [with GridT = openv
db::v9_0::Grid<openvdb::v9_0::tree::Tree<openvdb::v9_0::tree::RootNode<openvdb::v9_0::tree::InternalNode<open
vdb::v9_0::tree::InternalNode<openvdb::v9_0::tree::LeafNode<float, 3>, 4>, 5> > > >; SearchImplT = openvdb::v
9_0::tools::LinearSearchImpl<openvdb::v9_0::Grid<openvdb::v9_0::tree::Tree<openvdb::v9_0::tree::RootNode<open
vdb::v9_0::tree::InternalNode<openvdb::v9_0::tree::InternalNode<openvdb::v9_0::tree::LeafNode<float, 3>, 4>,
5> > > >, 0, double>; int NodeLevel = 2; RayT = openvdb::v9_0::math::Ray<double>; openvdb::v9_0::tools::LevelS
etRayIntersector<GridT, SearchImplT, NodeLevel, RayT>::RayType = openvdb::v9_0::math::Ray<double>]'
  164 |     bool intersectsWS(const RayType& wRay) const
      |          ^~~~~~~~~~~~
/usr/local/include/openvdb/tools/RayIntersector.h:164:10: note:   candidate expects 1 argument, 2 provided
/usr/local/include/openvdb/tools/RayIntersector.h:174:10: note: candidate: 'bool openvdb::v9_0::tools::LevelS
etRayIntersector<GridT, SearchImplT, NodeLevel, RayT>::intersectsWS(const RayType&, openvdb::v9_0::tools::Lev
elSetRayIntersector<GridT, SearchImplT, NodeLevel, RayT>::RealType&) const [with GridT = openvdb::v9_0::Grid<
openvdb::v9_0::tree::Tree<openvdb::v9_0::tree::RootNode<openvdb::v9_0::tree::InternalNode<openvdb::v9_0::tree
::InternalNode<openvdb::v9_0::tree::LeafNode<float, 3>, 4>, 5> > > >; SearchImplT = openvdb::v9_0::tools::Lin
earSearchImpl<openvdb::v9_0::Grid<openvdb::v9_0::tree::Tree<openvdb::v9_0::tree::RootNode<openvdb::v9_0::tree
::InternalNode<openvdb::v9_0::tree::InternalNode<openvdb::v9_0::tree::LeafNode<float, 3>, 4>, 5> > > >, 0, do
uble>; int NodeLevel = 2; RayT = openvdb::v9_0::math::Ray<double>; openvdb::v9_0::tools::LevelSetRayIntersect
or<GridT, SearchImplT, NodeLevel, RayT>::RayType = openvdb::v9_0::math::Ray<double>; openvdb::v9_0::tools::Le
velSetRayIntersector<GridT, SearchImplT, NodeLevel, RayT>::RealType = double]'
  174 |     bool intersectsWS(const RayType& wRay, RealType &wTime) const
      |          ^~~~~~~~~~~~
/usr/local/include/openvdb/tools/RayIntersector.h:174:38: note:   no known conversion for argument 1 from '__
gnu_cxx::__alloc_traits<std::allocator<openvdb::v9_0::math::Ray<float> >, openvdb::v9_0::math::Ray<float> >::
value_type' {aka 'openvdb::v9_0::math::Ray<float>'} to 'const RayType&' {aka 'const openvdb::v9_0::math::Ray<
double>&'}
  174 |     bool intersectsWS(const RayType& wRay, RealType &wTime) const
      |                       ~~~~~~~~~~~~~~~^~~~
/usr/local/include/openvdb/tools/RayIntersector.h:185:10: note: candidate: 'bool openvdb::v9_0::tools::LevelS
etRayIntersector<GridT, SearchImplT, NodeLevel, RayT>::intersectsWS(const RayType&, openvdb::v9_0::tools::Lev
elSetRayIntersector<GridT, SearchImplT, NodeLevel, RayT>::Vec3Type&) const [with GridT = openvdb::v9_0::Grid<
openvdb::v9_0::tree::Tree<openvdb::v9_0::tree::RootNode<openvdb::v9_0::tree::InternalNode<openvdb::v9_0::tree
::InternalNode<openvdb::v9_0::tree::LeafNode<float, 3>, 4>, 5> > > >; SearchImplT = openvdb::v9_0::tools::Lin
earSearchImpl<openvdb::v9_0::Grid<openvdb::v9_0::tree::Tree<openvdb::v9_0::tree::RootNode<openvdb::v9_0::tree
::InternalNode<openvdb::v9_0::tree::InternalNode<openvdb::v9_0::tree::LeafNode<float, 3>, 4>, 5> > > >, 0, do
uble>; int NodeLevel = 2; RayT = openvdb::v9_0::math::Ray<double>; openvdb::v9_0::tools::LevelSetRayIntersect
or<GridT, SearchImplT, NodeLevel, RayT>::RayType = openvdb::v9_0::math::Ray<double>; openvdb::v9_0::tools::Le
velSetRayIntersector<GridT, SearchImplT, NodeLevel, RayT>::Vec3Type = openvdb::v9_0::math::Vec3<double>]'
  185 |     bool intersectsWS(const RayType& wRay, Vec3Type& world) const
      |          ^~~~~~~~~~~~
/usr/local/include/openvdb/tools/RayIntersector.h:185:38: note:   no known conversion for argument 1 from '__
gnu_cxx::__alloc_traits<std::allocator<openvdb::v9_0::math::Ray<float> >, openvdb::v9_0::math::Ray<float> >::
value_type' {aka 'openvdb::v9_0::math::Ray<float>'} to 'const RayType&' {aka 'const openvdb::v9_0::math::Ray<
double>&'}
  185 |     bool intersectsWS(const RayType& wRay, Vec3Type& world) const
      |                       ~~~~~~~~~~~~~~~^~~~
/usr/local/include/openvdb/tools/RayIntersector.h:199:10: note: candidate: 'bool openvdb::v9_0::tools::LevelS
etRayIntersector<GridT, SearchImplT, NodeLevel, RayT>::intersectsWS(const RayType&, openvdb::v9_0::tools::Lev
elSetRayIntersector<GridT, SearchImplT, NodeLevel, RayT>::Vec3Type&, openvdb::v9_0::tools::LevelSetRayInterse
ctor<GridT, SearchImplT, NodeLevel, RayT>::RealType&) const [with GridT = openvdb::v9_0::Grid<openvdb::v9_0::
tree::Tree<openvdb::v9_0::tree::RootNode<openvdb::v9_0::tree::InternalNode<openvdb::v9_0::tree::InternalNode<
openvdb::v9_0::tree::LeafNode<float, 3>, 4>, 5> > > >; SearchImplT = openvdb::v9_0::tools::LinearSearchImpl<o
penvdb::v9_0::Grid<openvdb::v9_0::tree::Tree<openvdb::v9_0::tree::RootNode<openvdb::v9_0::tree::InternalNode<
openvdb::v9_0::tree::InternalNode<openvdb::v9_0::tree::LeafNode<float, 3>, 4>, 5> > > >, 0, double>; int Node
Level = 2; RayT = openvdb::v9_0::math::Ray<double>; openvdb::v9_0::tools::LevelSetRayIntersector<GridT, Searc
hImplT, NodeLevel, RayT>::RayType = openvdb::v9_0::math::Ray<double>; openvdb::v9_0::tools::LevelSetRayInters
ector<GridT, SearchImplT, NodeLevel, RayT>::Vec3Type = openvdb::v9_0::math::Vec3<double>; openvdb::v9_0::tool
s::LevelSetRayIntersector<GridT, SearchImplT, NodeLevel, RayT>::RealType = double]'
  199 |     bool intersectsWS(const RayType& wRay, Vec3Type& world, RealType &wTime) const
      |          ^~~~~~~~~~~~
/usr/local/include/openvdb/tools/RayIntersector.h:199:10: note:   candidate expects 3 arguments, 2 provided
/usr/local/include/openvdb/tools/RayIntersector.h:214:10: note: candidate: 'bool openvdb::v9_0::tools::LevelS
etRayIntersector<GridT, SearchImplT, NodeLevel, RayT>::intersectsWS(const RayType&, openvdb::v9_0::tools::Lev
elSetRayIntersector<GridT, SearchImplT, NodeLevel, RayT>::Vec3Type&, openvdb::v9_0::tools::LevelSetRayInterse
ctor<GridT, SearchImplT, NodeLevel, RayT>::Vec3Type&) const [with GridT = openvdb::v9_0::Grid<openvdb::v9_0::
tree::Tree<openvdb::v9_0::tree::RootNode<openvdb::v9_0::tree::InternalNode<openvdb::v9_0::tree::InternalNode<
openvdb::v9_0::tree::LeafNode<float, 3>, 4>, 5> > > >; SearchImplT = openvdb::v9_0::tools::LinearSearchImpl<o
penvdb::v9_0::Grid<openvdb::v9_0::tree::Tree<openvdb::v9_0::tree::RootNode<openvdb::v9_0::tree::InternalNode<
openvdb::v9_0::tree::InternalNode<openvdb::v9_0::tree::LeafNode<float, 3>, 4>, 5> > > >, 0, double>; int Node
Level = 2; RayT = openvdb::v9_0::math::Ray<double>; openvdb::v9_0::tools::LevelSetRayIntersector<GridT, Searc
hImplT, NodeLevel, RayT>::RayType = openvdb::v9_0::math::Ray<double>; openvdb::v9_0::tools::LevelSetRayInters
ector<GridT, SearchImplT, NodeLevel, RayT>::Vec3Type = openvdb::v9_0::math::Vec3<double>]'
  214 |     bool intersectsWS(const RayType& wRay, Vec3Type& world, Vec3Type& normal) const
      |          ^~~~~~~~~~~~
/usr/local/include/openvdb/tools/RayIntersector.h:214:10: note:   candidate expects 3 arguments, 2 provided
/usr/local/include/openvdb/tools/RayIntersector.h:230:10: note: candidate: 'bool openvdb::v9_0::tools::LevelS
etRayIntersector<GridT, SearchImplT, NodeLevel, RayT>::intersectsWS(const RayType&, openvdb::v9_0::tools::Lev
elSetRayIntersector<GridT, SearchImplT, NodeLevel, RayT>::Vec3Type&, openvdb::v9_0::tools::LevelSetRayInterse
ctor<GridT, SearchImplT, NodeLevel, RayT>::Vec3Type&, openvdb::v9_0::tools::LevelSetRayIntersector<GridT, Sea
rchImplT, NodeLevel, RayT>::RealType&) const [with GridT = openvdb::v9_0::Grid<openvdb::v9_0::tree::Tree<open
vdb::v9_0::tree::RootNode<openvdb::v9_0::tree::InternalNode<openvdb::v9_0::tree::InternalNode<openvdb::v9_0::
tree::LeafNode<float, 3>, 4>, 5> > > >; SearchImplT = openvdb::v9_0::tools::LinearSearchImpl<openvdb::v9_0::G
rid<openvdb::v9_0::tree::Tree<openvdb::v9_0::tree::RootNode<openvdb::v9_0::tree::InternalNode<openvdb::v9_0::
tree::InternalNode<openvdb::v9_0::tree::LeafNode<float, 3>, 4>, 5> > > >, 0, double>; int NodeLevel = 2; RayT
 = openvdb::v9_0::math::Ray<double>; openvdb::v9_0::tools::LevelSetRayIntersector<GridT, SearchImplT, NodeLev
el, RayT>::RayType = openvdb::v9_0::math::Ray<double>; openvdb::v9_0::tools::LevelSetRayIntersector<GridT, Se
archImplT, NodeLevel, RayT>::Vec3Type = openvdb::v9_0::math::Vec3<double>; openvdb::v9_0::tools::LevelSetRayI
ntersector<GridT, SearchImplT, NodeLevel, RayT>::RealType = double]'
  230 |     bool intersectsWS(const RayType& wRay, Vec3Type& world, Vec3Type& normal, RealType &wTime) const
```

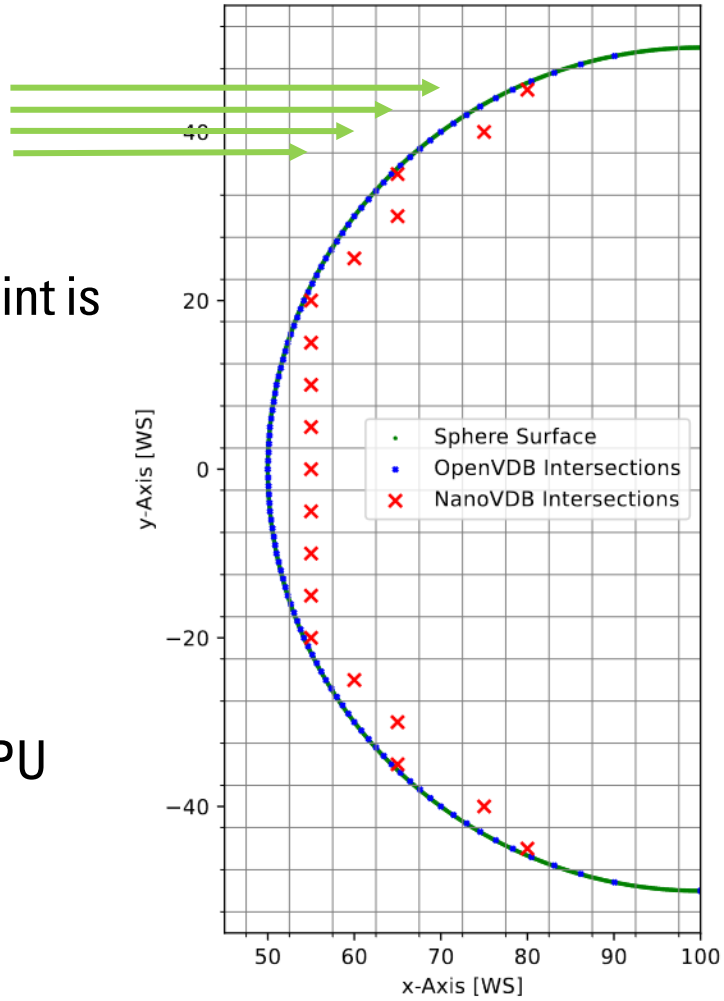
APPENDIX: SIMULATION SETUP

- 3D spheres
- Rays are shuffled in memory
- No reflections
- Volumetric ray source
- Offset sphere for distribution of ray lengths
- No ray leaves bounding box
- All rays leave narrow band, traverse inactive cells and reenter narrow band
- Time for ray generation not included
- Rays are reused if (V)RAM is full
- All calculations are verified
- No repetitions
- Only single precision



APPENDIX: ACCURACY

- NanoVDB stops when a voxel containing the intersection point is found
- Realistically only useful to determine IF the ray intersects
- OpenVDB uses neighboring voxels to approximate real intersection point
- Most likely (part of) the reason why NanoVDB is faster on CPU



APPENDIX: BRANCHLESS PROGRAMMING

- GPUs use SIMT architecture (single instruction multiple thread)
- Threads are grouped in “warps” (usually 32 threads = 1 warp)
- Problem: branches
- Threads execute both branches sequentially and a mask determines which branch has an effect
- Branchless programming:

```
x = (a > b) ? C : D; // branches
x = (a > b) * C + (a <= b) * D; // no branches
```

```
inline __hostdev__ bool ZeroCrossing(RayT& ray, AccT& acc, Coord& ijk,
{
    if (!ray.clip(acc.root().bbox()) || ray.t1() > 1e20)
        return false; // clip ray to bbox
    static const float Delta = 1.0001f;
    ijk = RoundDown<Coord>(ray.start()); // first hit of bbox
    HDDA<RayT, Coord> hdda(ray, acc.getDim(ijk, ray));
    const auto        v0 = acc.getValue(ijk);
    while (hdda.step()) {
        ijk = RoundDown<Coord>(ray(hdda.time() + Delta));
        hdda.update(ray, acc.getDim(ijk, ray));
        if (hdda.dim() > 1 || !acc.isActive(ijk))
            continue; // either a tile value or an inactive voxel
        while (hdda.step() && acc.isActive(hdda.voxel())) { // in the
            v = acc.getValue(hdda.voxel());
            if (v * v0 < 0) { // zero crossing
                ijk = hdda.voxel();
                t = hdda.time();
                return true;
            }
        }
    }
    return false;
}
```