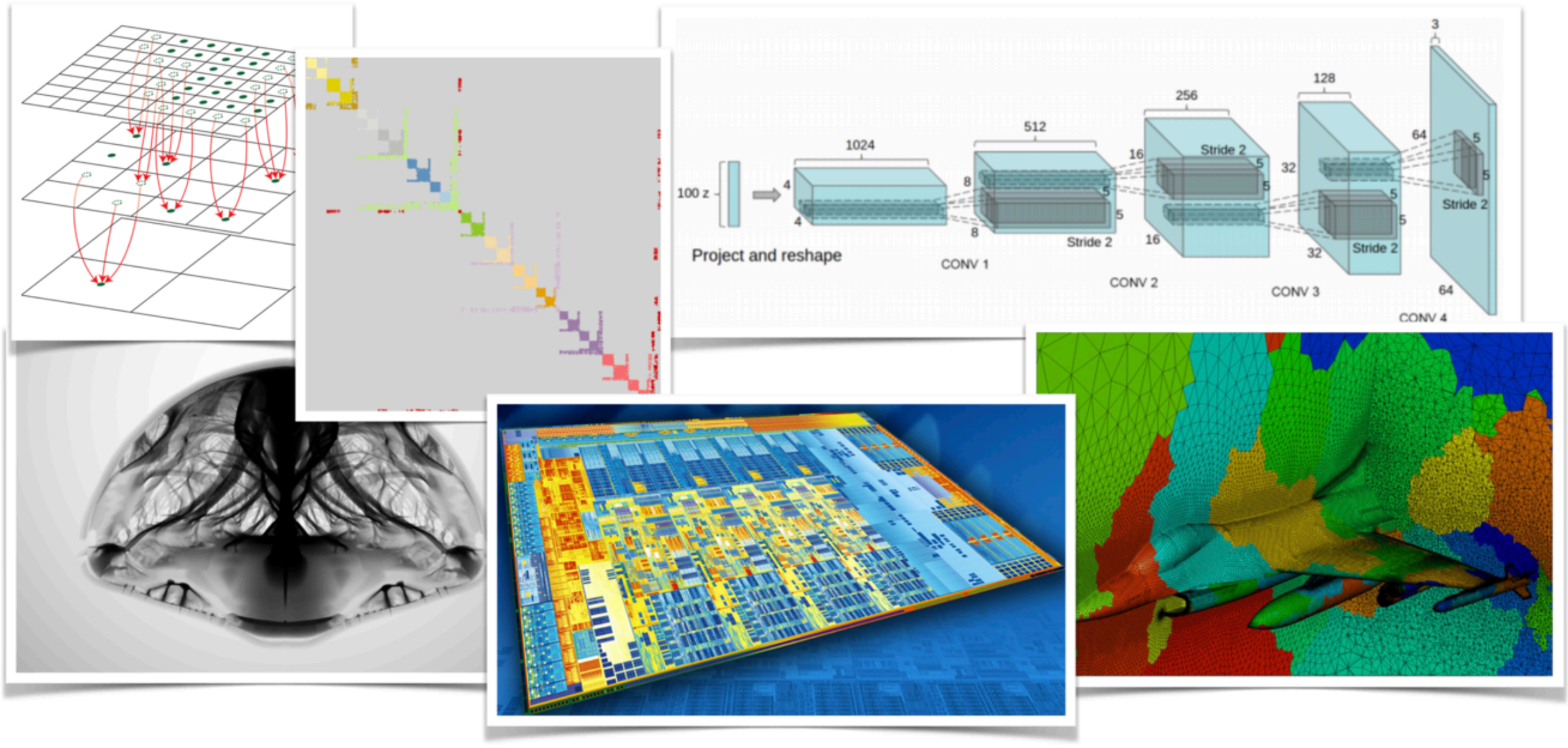


Lecture 24: Laplacian Stencil in NanoVDB/IndexGrid (code review and high-level walkthrough)

Thursday April 20th 2023

Logistics

- Programming Assignment #4 due next Monday
- Midterm grades are posted. HW1 and HW2 should be posted soon!
- Guest lectures by Prof. Matt Sinclair on Apr 25, 27 and May 2nd

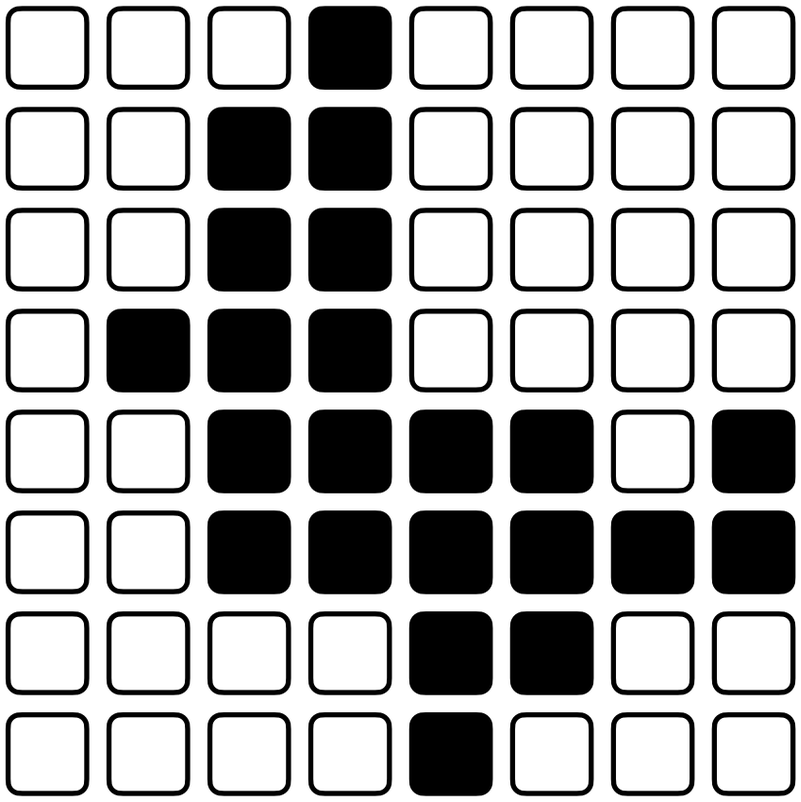
Today's lecture

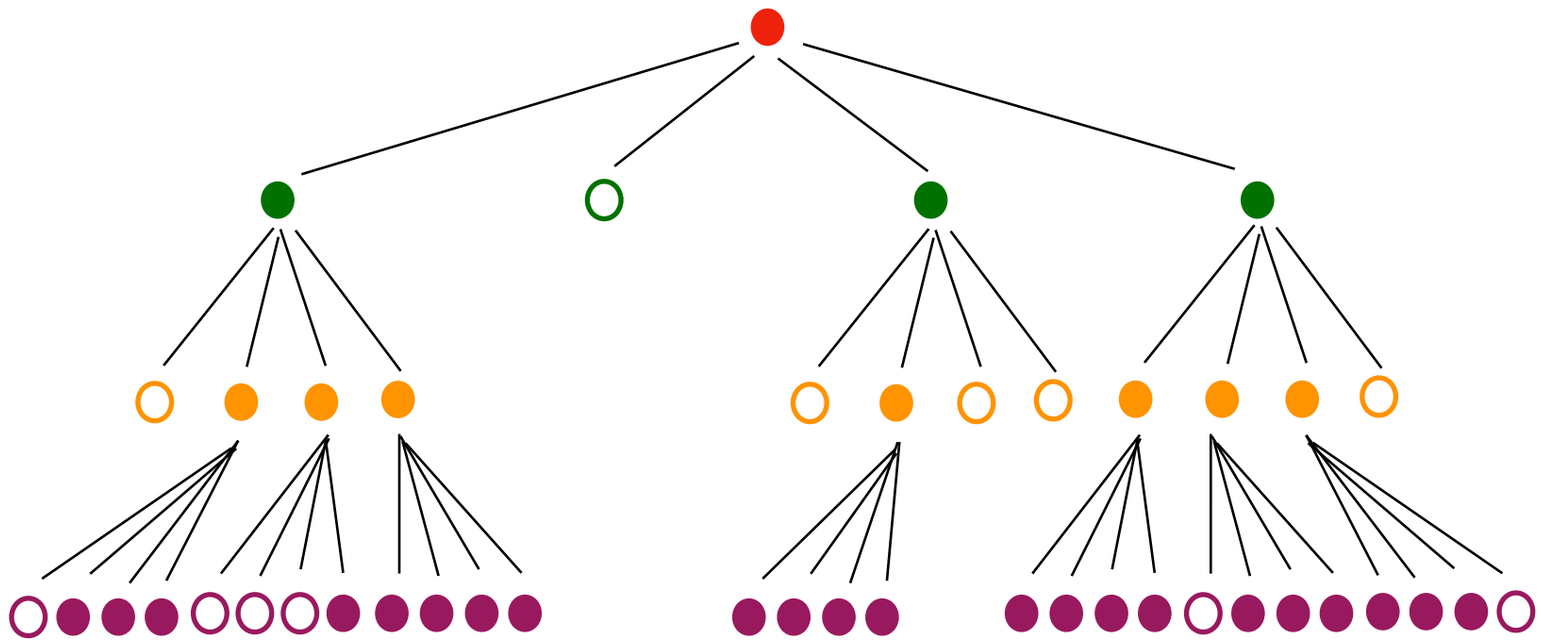
- An example of the Laplacian Stencil benchmark, implemented using NanoVDB/IndexGrid (https://www.openvdb.org/)

- Will *not* be included in your exams (material very fresh this year, very much an experiment in what's the right level of exposure to subject you to)

- Primarily a code walk-through. Please refer to OpenVDB/NanoVDB documentation for details on class structure and API specifics. Today is mostly explanation by example.

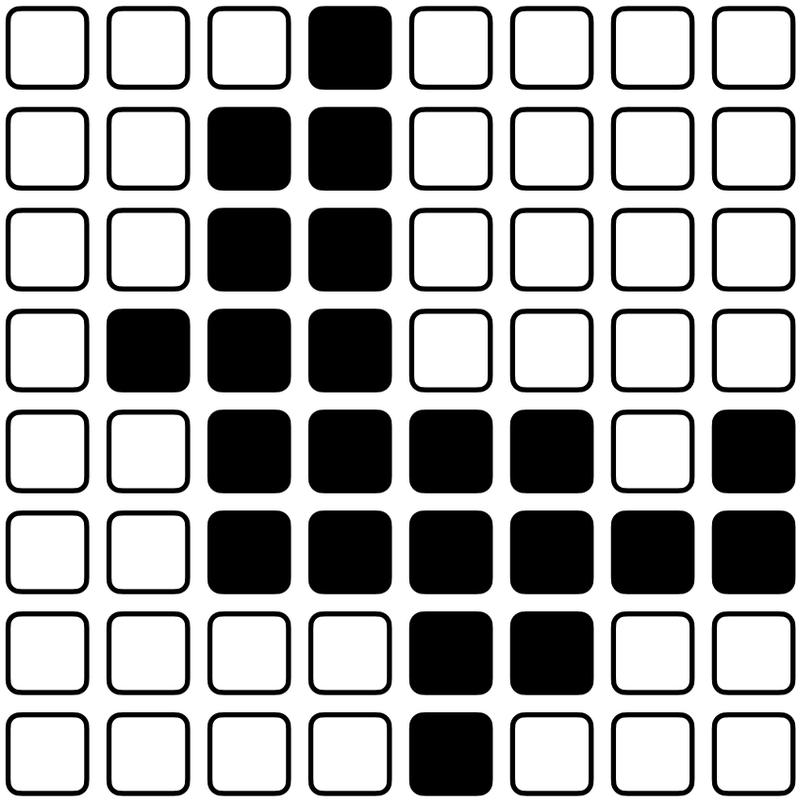
Sparse grids as quadtrees - a specific example

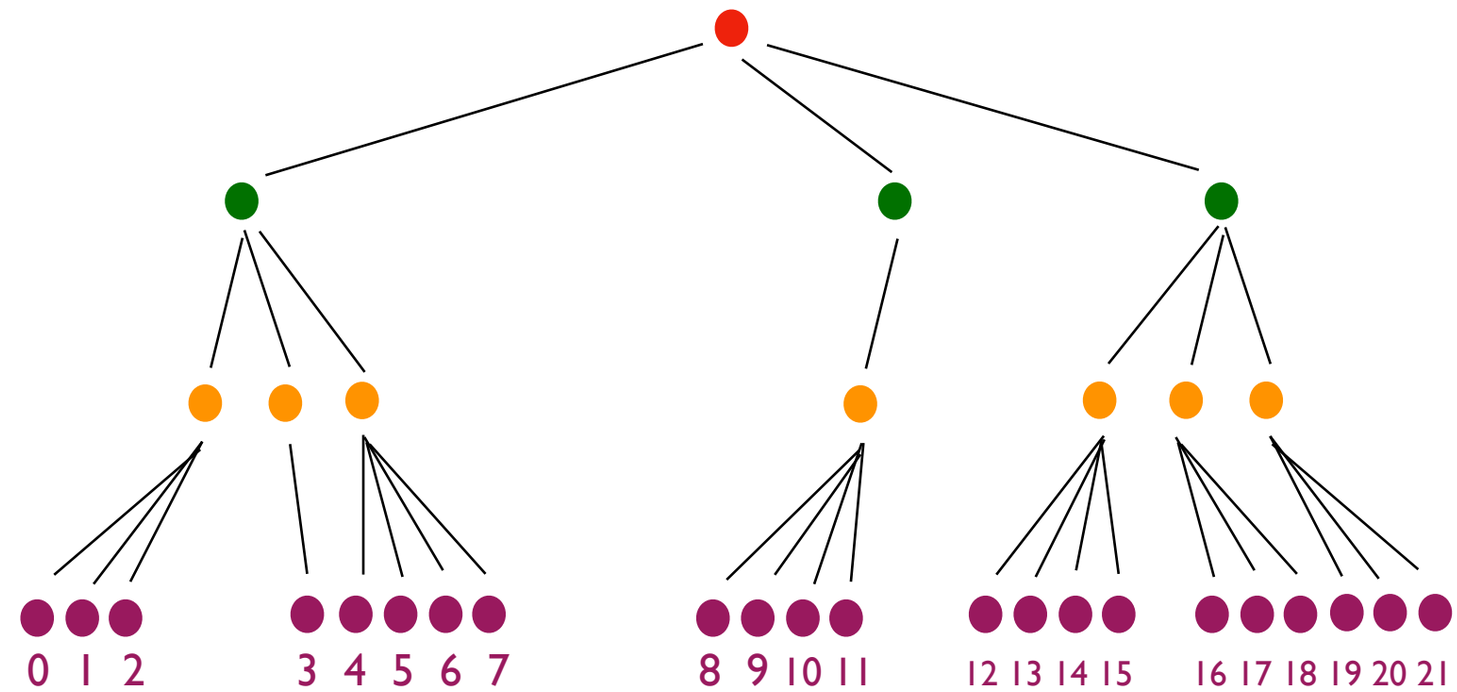


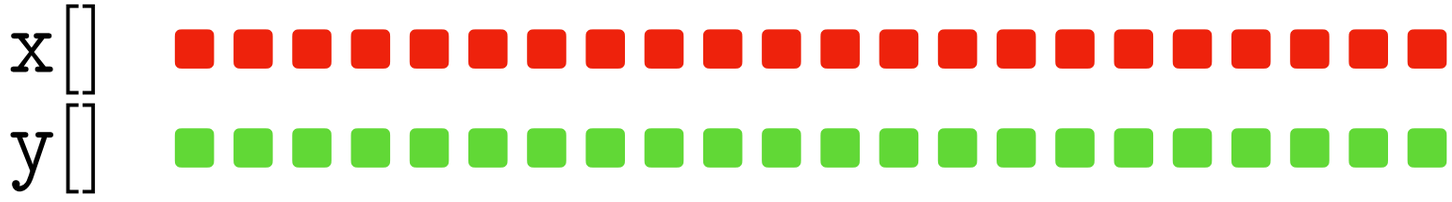


The “standard” quad tree representation (with data stored at leaves)

Sparse grids as quadtrees - a specific example

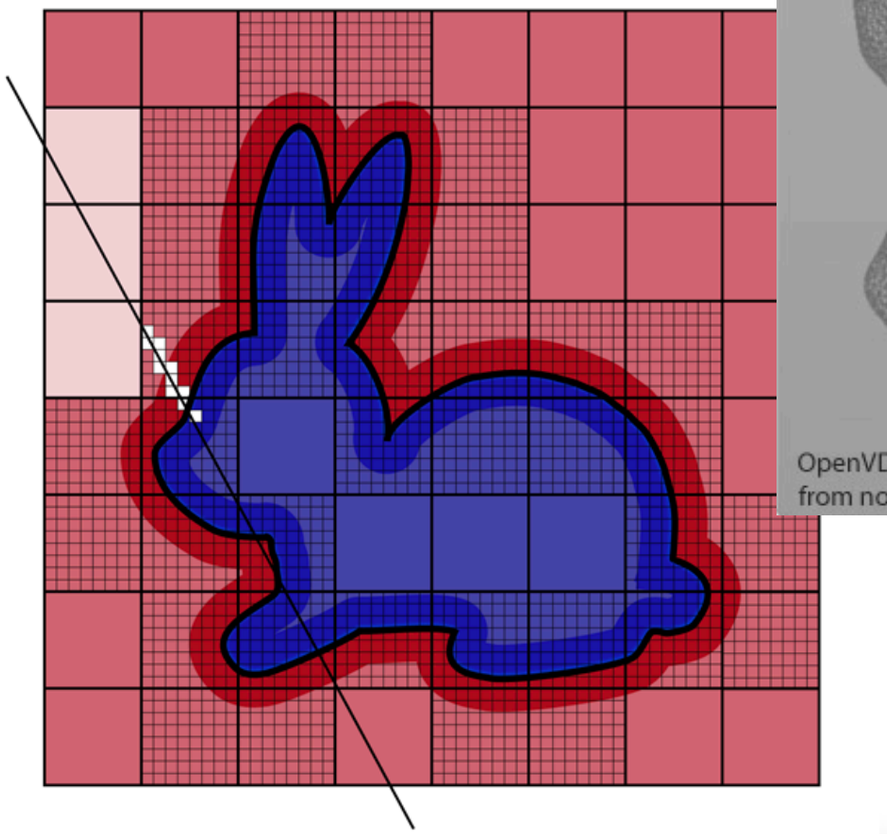


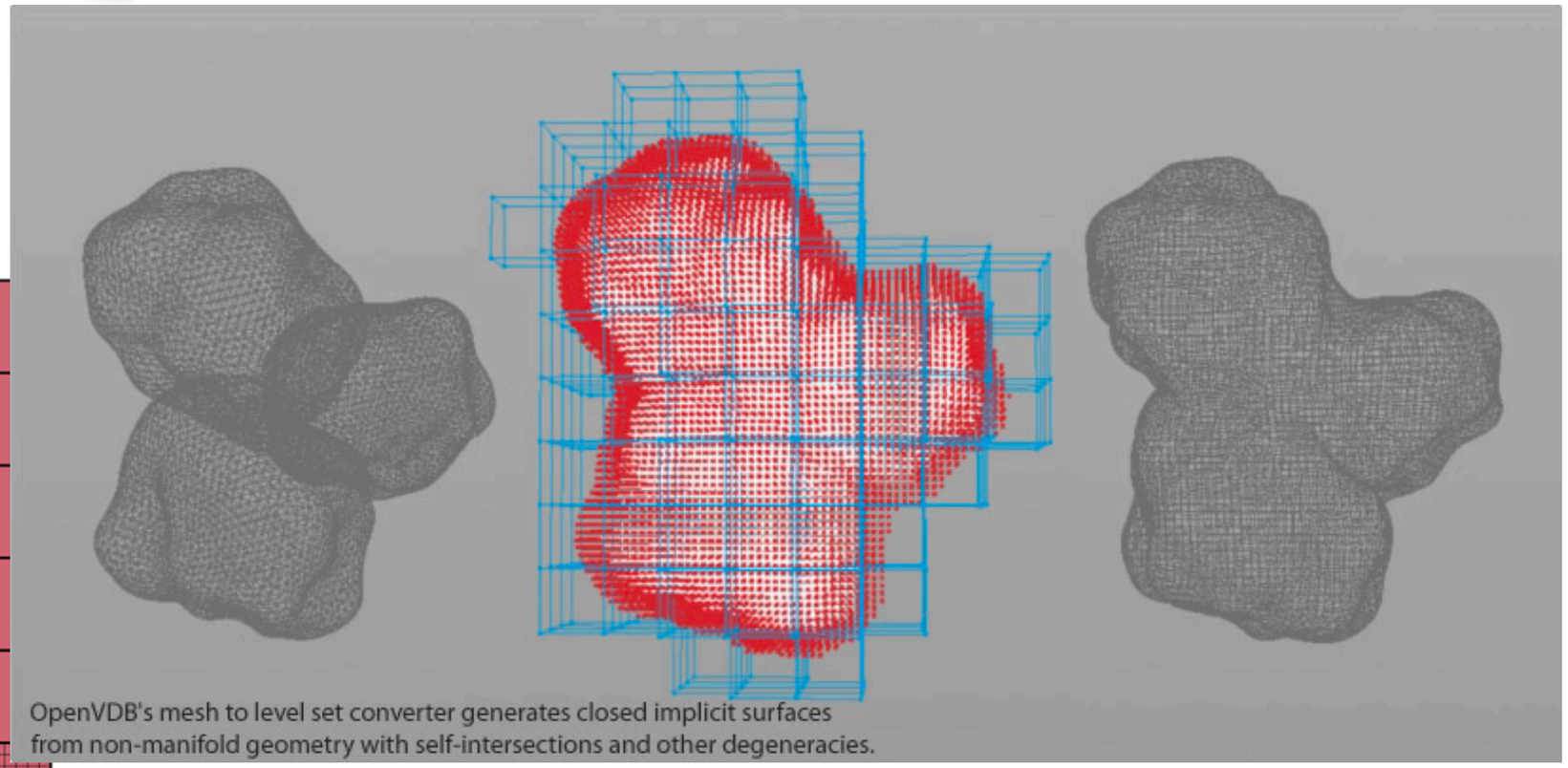




Another interpretation: Instead of the tree holding the actual data, it holds "Indices" into a separate linearized array storage. This concept (which we will call an Index-Grid) will be used a lot in our specific approach!

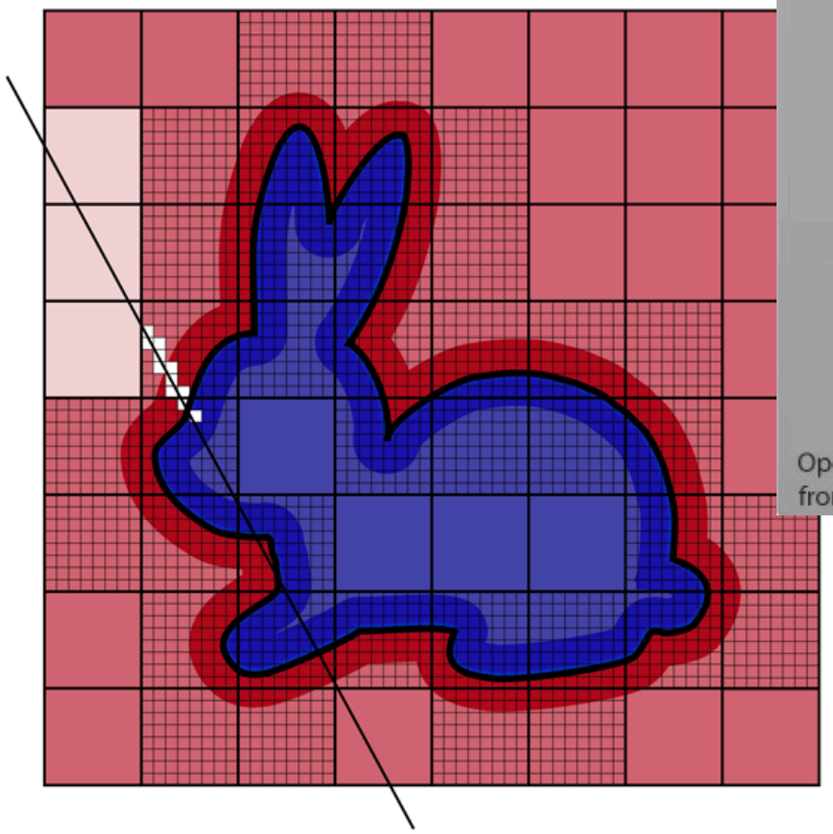
OpenVDB/NanoVDB

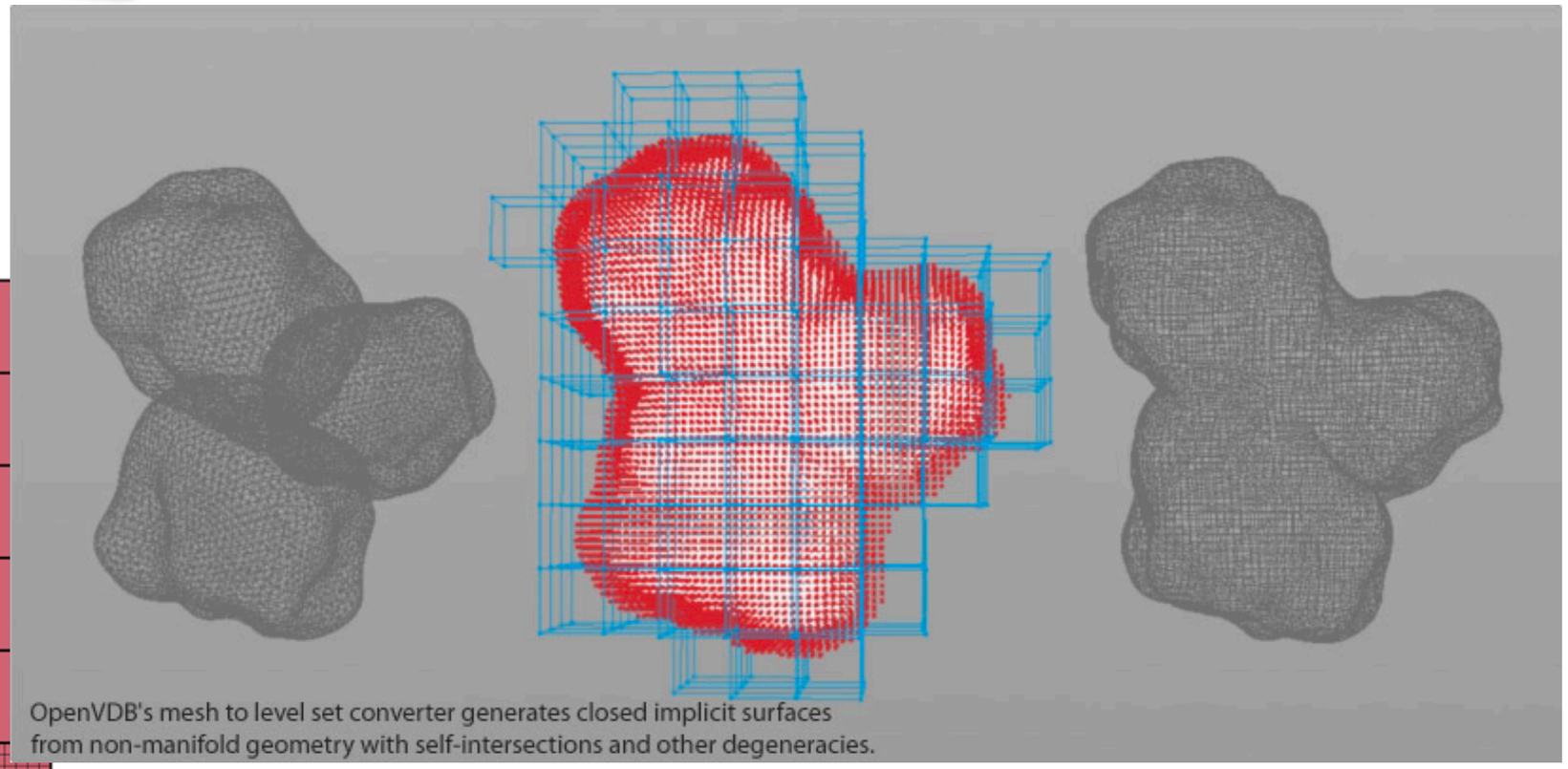




The data structure we will use to address some of these issues is OpenVDB (or its variant NanoVDB). The key idea is that it's a hierarchical (tree-like) sparse storage structure, but instead of each cube being split into 2x2x2 smaller "child" cubes, it's being split into 8x8 (in 2D) or 8x8x8 (in 3D) children at every level of the tree! I.e. each tree node has 512 children!

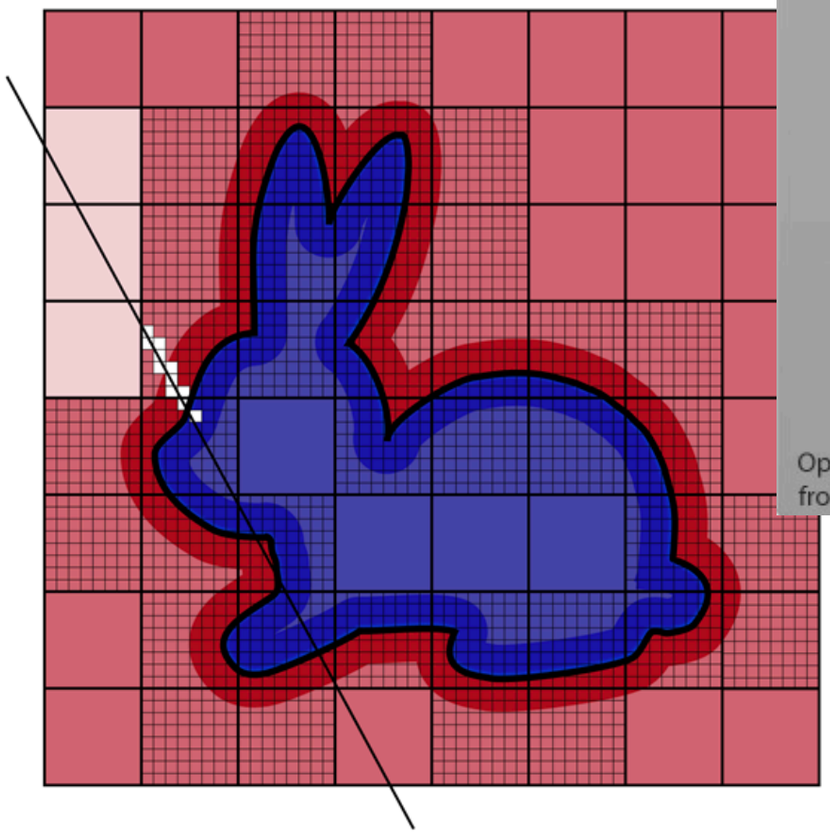
OpenVDB/NanoVDB

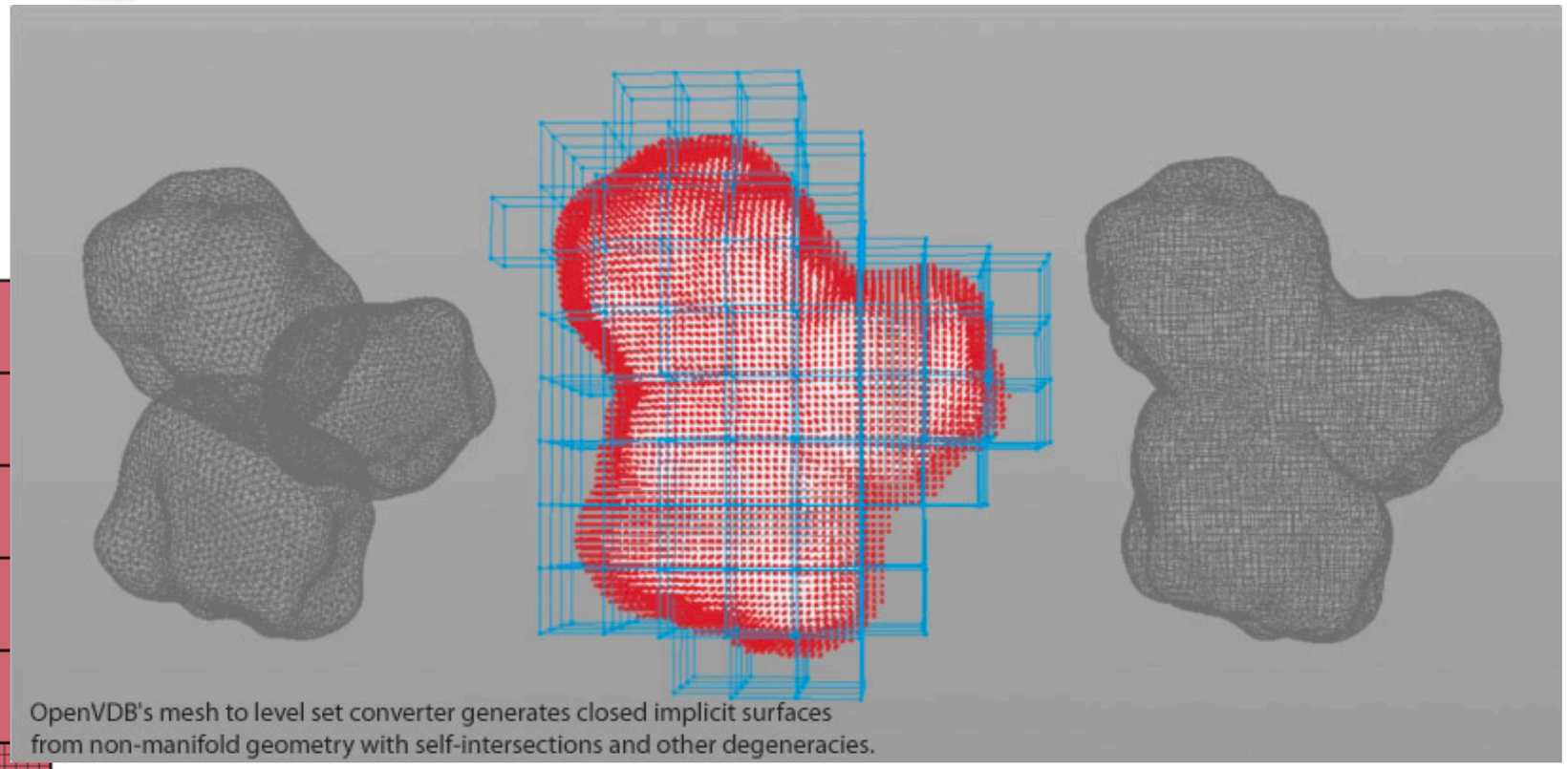




Benefit: At each leaf node of the VDB tree, we don't just have a single pixel (or 4/8 children), but a reasonably-sized 8x8x8 subgrid! This gives us plenty of opportunity to do most stencil operations within that grid!

OpenVDB/NanoVDB





OpenVDB: CPU-only, rich ecosystem, for dynamic sparsity, zoom into certain regions

nanoVDB: CPU/GPU Single header file, static sparsity

Optimization #2: VDB has caching structures that allow quick (or rather quick) access to nearby neighbors of a leaf-level grid.

main.cpp

LaplacianNanoVDB/LaplacianNanoVDB_0_0

Make sure to compile with options:
*g++ -I. -O3 -fopenmp *.cpp*

```
#include "Timer.h"
#include "Laplacian.h"

#include <iomanip>

int main(int argc, char *argv[])
{
    using array_t = float (&) [XDIM][YDIM][ZDIM];

    float *uRaw = new float [XDIM*YDIM*ZDIM];
    float *LuRaw = new float [XDIM*YDIM*ZDIM];
    float *uVDB = new float [XDIM*YDIM*ZDIM];
    float *LuVDB = new float [XDIM*YDIM*ZDIM];
    uint32_t *flagsVDB = new uint32_t [XDIM*YDIM*ZDIM];
    array_t u = reinterpret_cast<array_t>(*uRaw);
    array_t Lu = reinterpret_cast<array_t>(*LuRaw);

    Timer timer;

    timer.Start();
    auto handle = initializeIndexGrid();
    timer.Stop("Initializing indexGrid - Elapsed time :");
    auto *indexGridPtr = handle.grid<nanovdb::ValueIndex>();

    timer.Start();
    initializeData(u, Lu, indexGridPtr, uVDB, LuVDB, flagsVDB);
    timer.Stop("Initializing data - Elapsed time :");

    std::cout << "Discrepancy between dense and VDB (u) = " << compareData(u, indexGridPtr, uVDB)
```

main.cpp

LaplacianNanoVDB/LaplacianNanoVDB_0_0

New arrays for storing (linearized) sparse data

```
#include "Timer.h"
#include "Laplacian.h"

#include <iomanip>

int main(int argc, char *argv[])
{
    using array_t = float (&) [XDIM][YDIM][ZDIM];

    float *uRaw = new float [XDIM*YDIM*ZDIM];
    float *LuRaw = new float [XDIM*YDIM*ZDIM];
    float *uVDB = new float [XDIM*YDIM*ZDIM];
    float *LuVDB = new float [XDIM*YDIM*ZDIM];
    uint32_t *flagsVDB = new uint32_t [XDIM*YDIM*ZDIM];
    array_t u = reinterpret_cast<array_t>(*uRaw);
    array_t Lu = reinterpret_cast<array_t>(*LuRaw);

    Timer timer;

    timer.Start();
    auto handle = initializeIndexGrid();
    timer.Stop("Initializing indexGrid - Elapsed time :");
    auto *indexGridPtr = handle.grid<nanovdb::ValueIndex>();

    timer.Start();
    initializeData(u, Lu, indexGridPtr, uVDB, LuVDB, flagsVDB);
    timer.Stop("Initializing data - Elapsed time :");

    std::cout << "Discrepancy between dense and VDB (u) = " << compareData(u, indexGridPtr, uVDB)
```

main.cpp

LaplacianNanoVDB/LaplacianNanoVDB_0_0

Initialization code for the appropriate NanoVDB (indexGrids) we will use

```
#include "Timer.h"
#include "Laplacian.h"

#include <iomanip>

int main(int argc, char *argv[])
{
    using array_t = float (&) [XDIM][YDIM][ZDIM];

    float *uRaw = new float [XDIM*YDIM*ZDIM];
    float *LuRaw = new float [XDIM*YDIM*ZDIM];
    float *uVDB = new float [XDIM*YDIM*ZDIM];
    float *LuVDB = new float [XDIM*YDIM*ZDIM];
    uint32_t *flagsVDB = new uint32_t [XDIM*YDIM*ZDIM];
    array_t u = reinterpret_cast<array_t>(*uRaw);
    array_t Lu = reinterpret_cast<array_t>(*LuRaw);

    Timer timer;

    timer.Start();
    auto handle = initializeIndexGrid();
    timer.Stop("Initializing indexGrid - Elapsed time :");
    auto *indexGridPtr = handle.grid<nanovdb::ValueIndex>();

    timer.Start();
    initializeData(u, Lu, indexGridPtr, uVDB, LuVDB, flagsVDB);
    timer.Stop("Initializing data - Elapsed time :");

    std::cout << "Discrepancy between dense and VDB (u) = " << compareData(u, indexGridPtr, uVDB)
```

Laplacian.h

LaplacianNanoVDB/LaplacianNanoVDB_0_0

New additions in red; based on 3D dense Laplacian Stencil example (we'll go over them one by one)

```
#pragma once

#include <nanovdb/NanoVDB.h>
#include <nanovdb/util/GridHandle.h>

#define XDIM 512
#define YDIM 512
#define ZDIM 512
#define INDEX_ACTIVE_FLAG 0xffffffff

void ComputeLaplacian(const float (&u)[XDIM][YDIM][ZDIM], float (&Lu)[XDIM][YDIM][ZDIM]);

nanovdb::GridHandle<nanovdb::HostBuffer> initializeIndexGrid();

void initializeData(float (&u)[XDIM][YDIM][ZDIM], float (&Lu)[XDIM][YDIM][ZDIM],
    nanovdb::NanoGrid<nanovdb::ValueIndex>* indexGridPtr,
    float *uBuffer, float *LuBuffer, uint32_t* flagsBuffer);

float compareData(const float (&data)[XDIM][YDIM][ZDIM],
    nanovdb::NanoGrid<nanovdb::ValueIndex>* indexGridPtr,
    const float *dataVDBBuffer);

void computeLaplacianVDB(nanovdb::NanoGrid<nanovdb::ValueIndex>* indexGridPtr,
    float *uBuffer, float *LuBuffer, uint32_t* flagsBuffer);
```

main.cpp

LaplacianNanoVDB/LaplacianNanoVDB_0_0

Initialization code for the appropriate NanoVDB (indexGrids) we will use

```
#include "Timer.h"
#include "Laplacian.h"

#include <iomanip>

int main(int argc, char *argv[])
{
    using array_t = float (&) [XDIM][YDIM][ZDIM];

    float *uRaw = new float [XDIM*YDIM*ZDIM];
    float *LuRaw = new float [XDIM*YDIM*ZDIM];
    float *uVDB = new float [XDIM*YDIM*ZDIM];
    float *LuVDB = new float [XDIM*YDIM*ZDIM];
    uint32_t *flagsVDB = new uint32_t [XDIM*YDIM*ZDIM];
    array_t u = reinterpret_cast<array_t>(*uRaw);
    array_t Lu = reinterpret_cast<array_t>(*LuRaw);

    Timer timer;

    timer.Start();
    auto handle = initializeIndexGrid();
    timer.Stop("Initializing indexGrid - Elapsed time :");
    auto *indexGridPtr = handle.grid<nanovdb::ValueIndex>();

    timer.Start();
    initializeData(u, Lu, indexGridPtr, uVDB, LuVDB, flagsVDB);
    timer.Stop("Initializing data - Elapsed time :");

    std::cout << "Discrepancy between dense and VDB (u) = " << compareData(u, indexGridPtr, uVDB)
```

Laplacian.cpp

LaplacianNanoVDB/LaplacianNanoVDB_0_0

Initialize the sparsity pattern of array, store as IndexGrid (actually "dense" in this example)

```
#include "Laplacian.h"

#include <nanovdb/util/GridBuilder.h>
#include <nanovdb/util/IndexGridBuilder.h>

[...]

nanovdb::GridHandle<nanovdb::HostBuffer>
initializeIndexGrid()
{
    nanovdb::GridBuilder<nanovdb::ValueMask> builder(true);
    auto acc = builder.getAccessor();
    for (int i = 0; i < XDIM; i++)
    for (int j = 0; j < YDIM; j++)
    for (int k = 0; k < ZDIM; k++) {
        nanovdb::Coord xyz(i,j,k);
        acc.setValue(xyz, true);
    }
    auto handle = builder.getHandle();
    auto *dstGrid = handle.grid<nanovdb::ValueMask>();
    nanovdb::IndexGridBuilder<nanovdb::ValueMask> indexBuilder(*dstGrid, false, false);
    return indexBuilder.getHandle();
}

void initializeData(float (&u)[XDIM][YDIM][ZDIM], float (&Lu)[XDIM][YDIM][ZDIM],
    nanovdb::NanoGrid<nanovdb::ValueIndex>* indexGridPtr,
    float *uBuffer, float *LuBuffer, uint32_t* flagsBuffer)
{
```

main.cpp

LaplacianNanoVDB/LaplacianNanoVDB_0_0

Put some initial values in u (dense and VDB) and compare

```
#include "Timer.h"
#include "Laplacian.h"

#include <iomanip>

int main(int argc, char *argv[])
{
    [...]

    timer.Start();
    initializeData(u, Lu, indexGridPtr, uVDB, LuVDB, flagsVDB);
    timer.Stop("Initializing data - Elapsed time :");

    std::cout << "Discrepancy between dense and VDB (u) = “
        << compareData(u, indexGridPtr, uVDB) << std::endl;

    for(int test = 1; test <= 10; test++)
    {
        std::cout << "Running test iteration (dense) " << std::setw(2) << test << " ";
        timer.Start();
        ComputeLaplacian(u, Lu);
        timer.Stop("Elapsed time : ");
    }

    for(int test = 1; test <= 10; test++)
    {
        std::cout << "Running test iteration (VDB) " << std::setw(2) << test << " ";
        timer.Start();
        computeLaplacianVDB(indexGridPtr, uVDB, LuVDB, flagsVDB);
```

Laplacian.cpp

LaplacianNanoVDB/LaplacianNanoVDB_0_0

```
#include "Laplacian.h"

[...]

void initializeData(float (&u)[XDIM][YDIM][ZDIM], float (&Lu)[XDIM][YDIM][ZDIM],
    nanovdb::NanoGrid<nanovdb::ValueIndex>* indexGridPtr,
    float *uBuffer, float *LuBuffer, uint32_t* flagsBuffer)
{
    float *uPtr = &u[0][0][0], *LuPtr = &Lu[0][0][0];

    // Zero out buffers by iterating linearly over them
    for (int n = 0; n < XDIM*YDIM*ZDIM; n++) {
        uPtr[n] = LuPtr[n] = uBuffer[n] = LuBuffer[n] = 0.0;
        flagsBuffer[n] = 0;
    }

    auto acc = indexGridPtr->getAccessor();
    for (int i = 1; i < XDIM-1; i++)
    for (int j = 1; j < YDIM-1; j++)
    for (int k = 1; k < ZDIM-1; k++) {
        nanovdb::Coord xyz(i,j,k);
        auto index = acc.getValue(xyz);
        u[i][j][k] = (float) ((i+j+k)%256-128);
        uBuffer[index] = u[i][j][k];
        flagsBuffer[index] = INDEX_ACTIVE_FLAG;
    }
}

float compareData(const float (&data)[XDIM][YDIM][ZDIM],
```

Populating with initial data

Laplacian.cpp

LaplacianNanoVDB/LaplacianNanoVDB_0_0

```
#include "Laplacian.h"

[...]


float compareData(const float (&data)[XDIM][YDIM][ZDIM],
    nanovdb::NanoGrid<nanovdb::ValueIndex>* indexGridPtr,
    const float *dataVDBBuffer)
{
    float result = 0.;
    auto acc = indexGridPtr->getAccessor();
    for (int i = 0; i < XDIM; i++)
    for (int j = 0; j < YDIM; j++)
    for (int k = 0; k < ZDIM; k++) {
        nanovdb::Coord xyz(i,j,k);
        auto index = acc.getValue(xyz);
        result = std::max( result, std::abs(data[i][j][k]-dataVDBBuffer[index]) );
    }

    return result;
}

void computeLaplacianVDB(nanovdb::NanoGrid<nanovdb::ValueIndex>* indexGridPtr,
    float *uBuffer, float *LuBuffer, uint32_t* flagsBuffer)
{
    auto mgrHandle = createNodeManager(*indexGridPtr);
    auto *mgr = mgrHandle.template mgr<nanovdb::ValueIndex>();
    auto acc = indexGridPtr->getAccessor();
```

Comparing for correctness

main.cpp

LaplacianNanoVDB/LaplacianNanoVDB_0_0

Run and benchmark both versions,
Check that the result is correct

```
#include "Timer.h"
#include "Laplacian.h"

#include <iomanip>

int main(int argc, char *argv[])
{
    [...]

   for(int test = 1; test <= 10; test++)
    {
    std::cout << "Running test iteration (dense) " << std::setw(2) << test << " ";
    timer.Start();
    ComputeLaplacian(u, Lu);
    timer.Stop("Elapsed time : ");
}

for(int test = 1; test <= 10; test++)
{
    std::cout << "Running test iteration (VDB) " << std::setw(2) << test << " ";
    timer.Start();
    computeLaplacianVDB(indexGridPtr, uVDB, LuVDB, flagsVDB);
    timer.Stop("Elapsed time : ");
}

std::cout << "Discrepancy between dense and VDB (Lu) = "
     << compareData(Lu, indexGridPtr, LuVDB) << std::endl;
```

Laplacian.cpp

LaplacianNanoVDB/LaplacianNanoVDB_0_0

```
[...]
void computeLaplacianVDB(nanovdb::NanoGrid<nanovdb::ValueIndex>* indexGridPtr,
    float *uBuffer, float *LuBuffer, uint32_t* flagsBuffer)
{
    auto mgrHandle = createNodeManager(*indexGridPtr);
    auto *mgr = mgrHandle.template mgr<nanovdb::ValueIndex>();
    auto acc = indexGridPtr->getAccessor();

#pragma omp parallel for firstprivate(acc)
    for ( size_t l = 0; l < mgr->nodeCount(0); ++l ) // l enumerates "leaves"
        for( auto iter = mgr->leaf(l).beginValue(); iter; ++iter ){
            auto coord = iter.getCoord(); // this is the coordinate within the leaf
            auto indexCtr = *iter; // this is the "center" index of the stencil;
            if (flagsBuffer[indexCtr] == INDEX_ACTIVE_FLAG) {
                auto indexPlusX  = acc.getValue(nanovdb::Coord(coord.x()+1, coord.y()  , coord.z()  ));
                auto indexMinusX = acc.getValue(nanovdb::Coord(coord.x()-1, coord.y()  , coord.z()  ));
                auto indexPlusY  = acc.getValue(nanovdb::Coord(coord.x()  , coord.y()+1, coord.z()  ));
                auto indexMinusY = acc.getValue(nanovdb::Coord(coord.x()  , coord.y()-1, coord.z()  ));
                auto indexPlusZ  = acc.getValue(nanovdb::Coord(coord.x()  , coord.y()  , coord.z()+1));
                auto indexMinusZ = acc.getValue(nanovdb::Coord(coord.x()  , coord.y()  , coord.z()-1));
                LuBuffer[indexCtr] =
                    -6 * uBuffer[indexCtr]
                    + uBuffer[indexPlusX]
                    + uBuffer[indexMinusX]
                    + uBuffer[indexPlusY]
                    + uBuffer[indexMinusY]
                    + uBuffer[indexPlusZ]
                    + uBuffer[indexMinusZ];
            }
        }
}
```

VDB version of Laplacian Stencil!

Laplacian.cpp

```
[...]
void computeLaplacianVDB(nan
    float *uBuffer, float *L
{
    auto mgrHandle = createN
    auto *mgr = mgrHandle.te
    auto acc = indexGridPtr-

#pragma omp parallel for fir
    for ( size_t l = 0; l <
        for( auto iter = mgr
            auto coord = ite
        auto indexCtr =
        if (flagsBuffer[
            auto indexPl
            auto indexMi
            auto indexPl
            auto indexMi
            auto indexPl
            auto indexMi
            LuBuffer[ind
                -6 * uBu
                + uBuffe
                + uBuffe
                + uBuffe
                + uBuffe
                + uBuffe
                + uBuffe
        }
    }
}
```

LaplacianNanoVDB/LaplacianNanoVDB_0_0

```
[Initializing indexGrid - Elapsed time :244.07ms]
[Initializing data - Elapsed time :775.138ms]
Discrepancy between dense and VDB (u) = 0
Running test iteration (dense)  1 [Elapsed time : 67.4361ms]
Running test iteration (dense)  2 [Elapsed time : 67.0358ms]
Running test iteration (dense)  3 [Elapsed time : 67.3724ms]
Running test iteration (dense)  4 [Elapsed time : 67.0853ms]
Running test iteration (dense)  5 [Elapsed time : 68.2836ms]
Running test iteration (dense)  6 [Elapsed time : 67.192ms]
Running test iteration (dense)  7 [Elapsed time : 67.2768ms]
Running test iteration (dense)  8 [Elapsed time : 67.7054ms]
Running test iteration (dense)  9 [Elapsed time : 67.327ms]
Running test iteration (dense) 10 [Elapsed time : 67.0988ms]
Running test iteration (VDB)  1 [Elapsed time : 326.101ms]
Running test iteration (VDB)  2 [Elapsed time : 330.922ms]
Running test iteration (VDB)  3 [Elapsed time : 328.062ms]
Running test iteration (VDB)  4 [Elapsed time : 324.41ms]
Running test iteration (VDB)  5 [Elapsed time : 328.728ms]
Running test iteration (VDB)  6 [Elapsed time : 324.967ms]
Running test iteration (VDB)  7 [Elapsed time : 326.969ms]
Running test iteration (VDB)  8 [Elapsed time : 325.041ms]
Running test iteration (VDB)  9 [Elapsed time : 328.691ms]
Running test iteration (VDB) 10 [Elapsed time : 330.403ms]
Discrepancy between dense and VDB (Lu) = 0
```

Optimality?

- Parallelization/multithreading of accessors?
- A different paradigm for operating on leaf nodes? (create local copy?)
- Ultimately: could get to 80-90% of dense performance