PICKNIK

BONXAI

Fast, hierarchical, sparse Voxel Grid

Davide Faconti, ROSCon 2023

About me

- Davide Faconti, nice to meet you!
- Robotic Architect, working at PickNik Robotics
- 20 years, doing robots of all kinds.
- Author of PlotJuggler and BehaviorTree.CPP

What is Bonxai?

A header-only, single file, C++ library
to store volumetric data
in discretized cells,
i.e. a Voxel Grid.

Created primarily to manage
3D maps and occupancy grids.

About Bonxai

Bonxai data structure is:

- Unbounded: can represent an infinite space
- Sparse: only allocates the used cells
- Fast: 10x faster than Octomap
- Hierarchical: more details in the next slides...
- Typical time complexity to access a voxel is O(1)

About Bonxai

Bonxai data structure is:

- Unbou… …ite space
- Sparse… …s
- Fast: 1…
- Hierar… …ext slides
- Typical time complexity to access a voxel is O(1)

This slide contains a few lies.

Pay attention to the presentation, to learn more

Summary of the presentation, in one slide

PICKNIK

Literature

PICKNIK

"VDB: High-Resolution Sparse Volumes with Dynamic Topology", presented at SIGGRAPH 2013.

"Octomap: An Efficient Probabilistic 3D Mapping Framework Based on Octrees", presented at ROSCon 2013

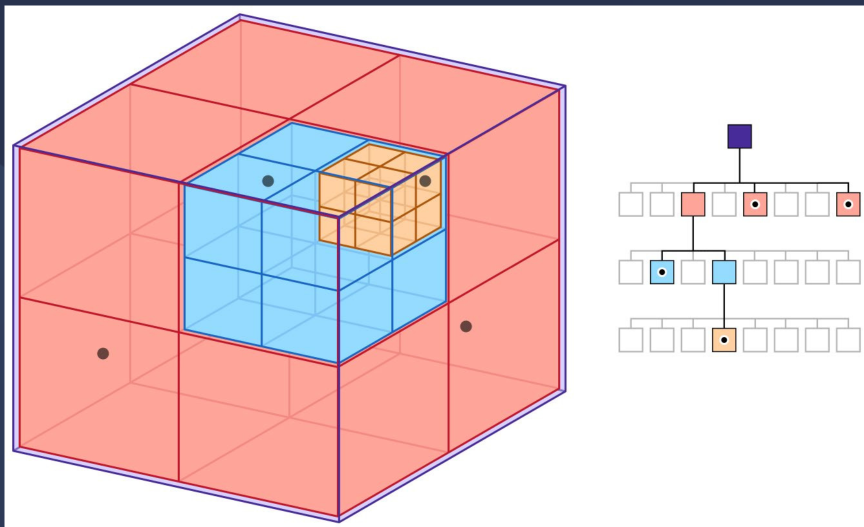
"A Sparse-Dense Approach for Efficient Grid Mapping", ICARSC 2018

How to use it

```cpp
// Each cell will contail a `float` and it will have size 0.05
double voxel_resolution = 0.05;
Bonxai::VoxelGrid<float> grid( voxel_resolution );

// Create this accessor once, and reuse it as much as possible.
auto accessor = grid.createAccessor();

// Create cells with value 42.0 in a 1x1x1 cube.
// Given voxel_resolution = 0.05, this will be equivalent
// to 20x20x20 cells in the grid.

for( double x = 0; x < 1.0; x += voxel_resolution ) {
  for( double y = 0; y < 1.0; y += voxel_resolution ) {
    for( double z = 0; z < 1.0; z += voxel_resolution ) {
      // discretize the position {x,y,z}
      Bonxai::CoordT coord = grid.posToCoord(x, y, z);
      accessor.setValue( coord, 42.0 );
    }
  }
}

// You can read (or update) the value of a cell as shown below.
// If the cell doesn't exist, `value_ptr` will be nullptr,
Bonxai::CoordT coord = grid.posToCoord(x, y, z);
float* value_ptr = accessor.value( coord );
```

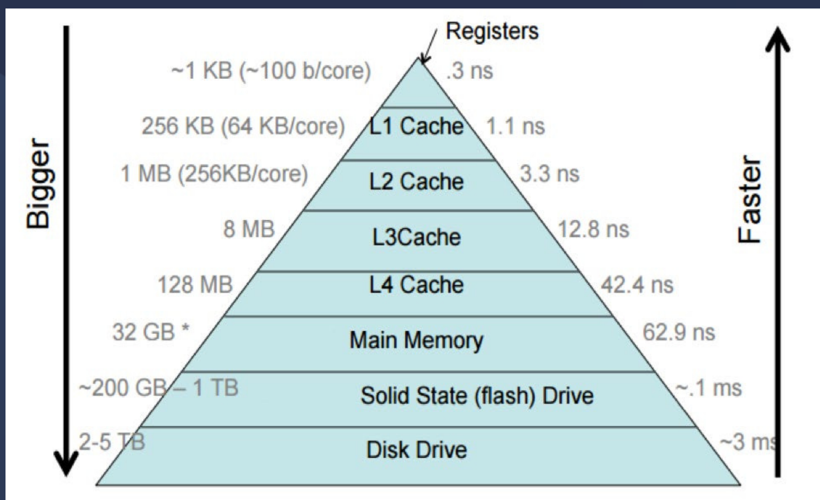
Octrees refresher

PICKNIK



- Subdivision of the space, where each cube (Node) is split in 8 cubes recursively.
- The voxel size will tell us when to stop the recursion.
- Sparse: only defined Nodes are allocated.
- Time complexity of search and update: O(logN), where N is the number of Nodes

Potential problems with Octrees



Many heap allocations, when building the tree.

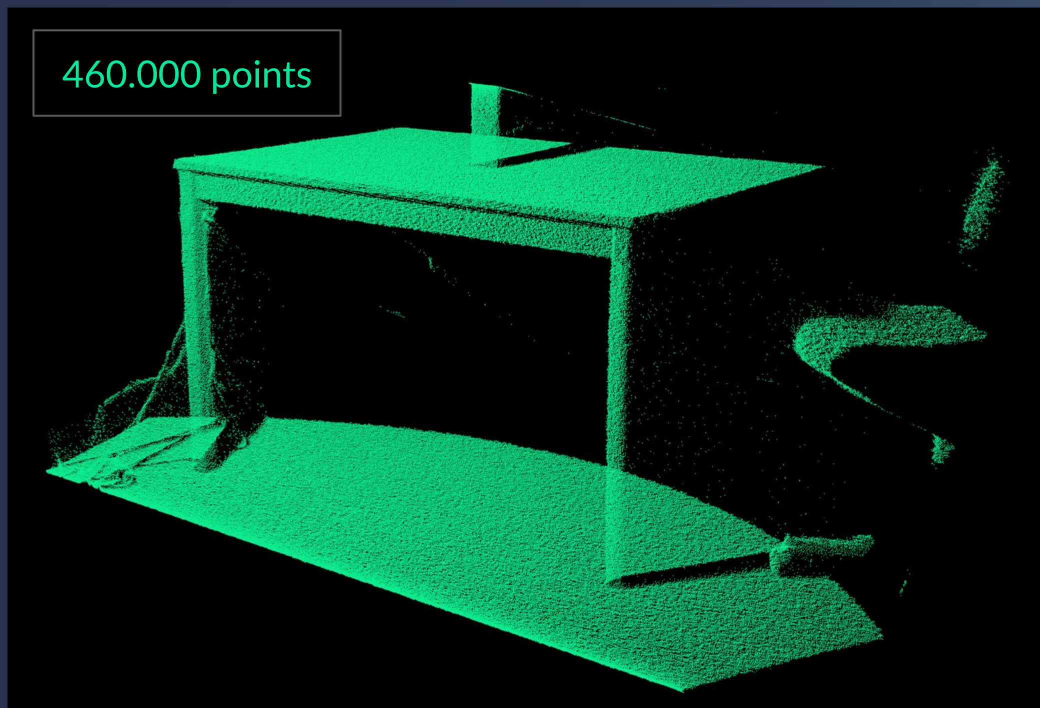
Not cache friendly. Memory is fragmented.

This affects negatively insertion, searching and iteration times.

Bonxai VS Octree

Example, using 2 cm voxels

- Create the grid: 10x faster
- Update/Read existing cells: 6x faster
- Iterate through all cells: 10x faster



Warning! Performance will vary a lot, based on the density of the data, but Bonxai always wins

PICKNIK

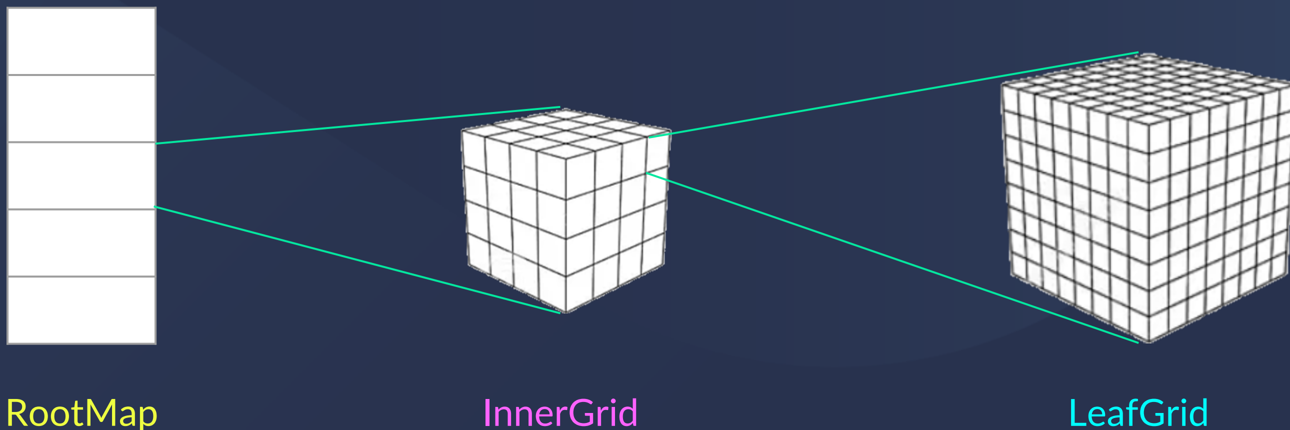
How Bonxai works

under the hood

Bonxai/VDB data structure

Bonxai data structure, implemented by VoxelGrid<DataType>:

- RootMap: a hashmap, where each element contains an instance of InnerGrid
- InnerGrid: a 3D grid *(NxNxN)*, each element contains a pointer to a LeafGrid.
- LeafGrid: a 3D grid *(MxMxM)*, each element contains a DataType
- Grids have a **bitfield mask**, to check if a cell is active or not.



PICKNIK

```cpp
using LeafGrid  = Grid<DataT>;
using InnerGrid = Grid<std::shared_ptr<LeafGrid>>;
using RootMap   = std::unordered_map<CoordT, InnerGrid>;
```

Voxel Coordinates (int32):

X	0	1	2	3	4	5	6	7	…	29	30	31
Y	0	1	2	3	4	5	6	7	…	29	30	31
Z	0	1	2	3	4	5	6	7	…	29	30	31

Given a set of coordinates, we want to find a unique instance of DataT

PICKNIK

```
using LeafGrid  = Grid<DataT>;
using InnerGrid = Grid<std::shared_ptr<LeafGrid>>;
using RootMap   = std::unordered_map<CoordT, InnerGrid>;
```

Voxel Coordinates (int32):

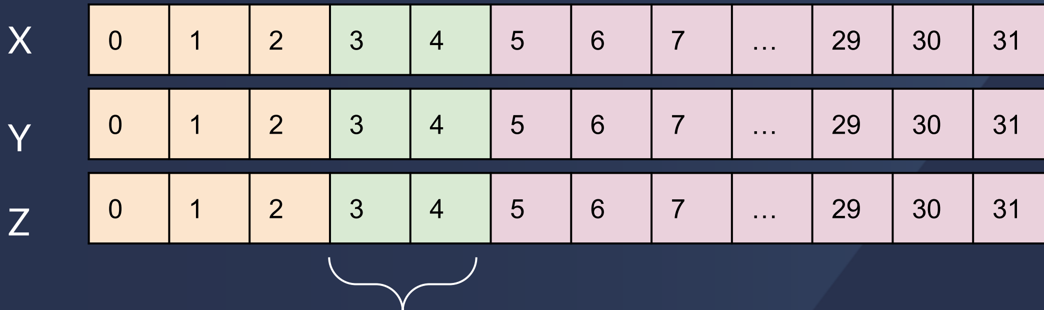
X	0	1	2	3	4	5	6	7	…	29	30	31
Y	0	1	2	3	4	5	6	7	…	29	30	31
Z	0	1	2	3	4	5	6	7	…	29	30	31

```
mask = ~(0b11111);
hash = ((1 << 20) - 1) &
  ((x & mask) * 73856093 ^
   (y & mask) * 19349663 ^
   (z & mask) * 83492791)
```

These bits are used to find a value in the hashmap, i.e. RootMap.

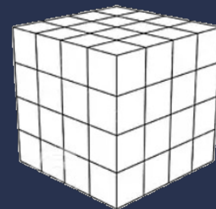
```
using LeafGrid  = Grid<DataT>;
using InnerGrid = Grid<std::shared_ptr<LeafGrid>>;
using RootMap   = std::unordered_map<CoordT, InnerGrid>;
```

Voxel Coordinates (int32):




```
mask_2_bits = 0b11;
x = (x >> 3) & mask_2_bits;
y = (y >> 3) & mask_2_bits;
z = (z >> 3) & mask_2_bits;

index = (x) | (y << 2) | (z << 4);
```

These bits are used to find the index into the 4x4x4 InnerGrid.



PICKNIK

PICKNIK

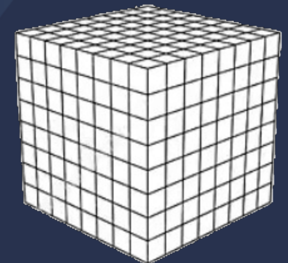
```
using LeafGrid  = Grid<DataT>;
using InnerGrid = Grid<std::shared_ptr<LeafGrid>>;
using RootMap   = std::unordered_map<CoordT, InnerGrid>;
```

Voxel Coordinates (int32):

X	0	1	2	3	4	5	6	7	…	29	30	31
Y	0	1	2	3	4	5	6	7	…	29	30	31
Z	0	1	2	3	4	5	6	7	…	29	30	31

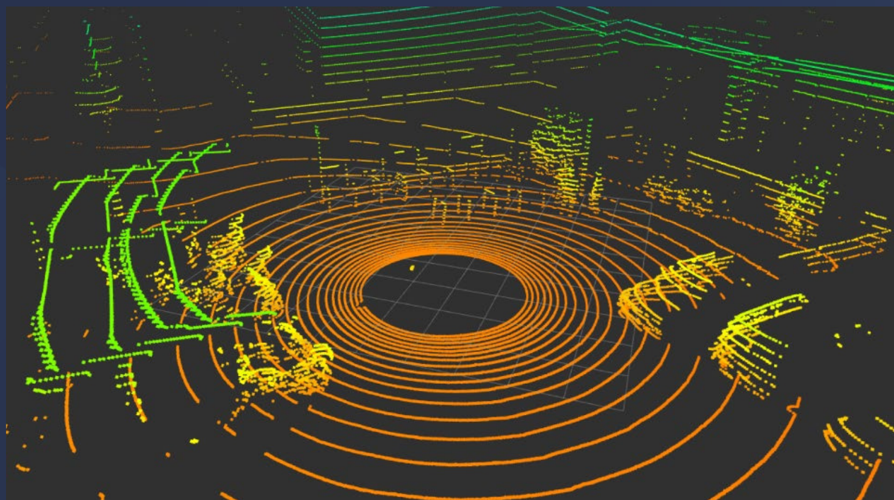
```
mask_3_bits = 0b111;
x = x & mask_3_bits;
y = y & mask_3_bits;
z = z & mask_3_bits;

index = (x) | (y << 3) | (z << 6);
```

These bits are used to find the index into the 8x8x8 LeafGrid.

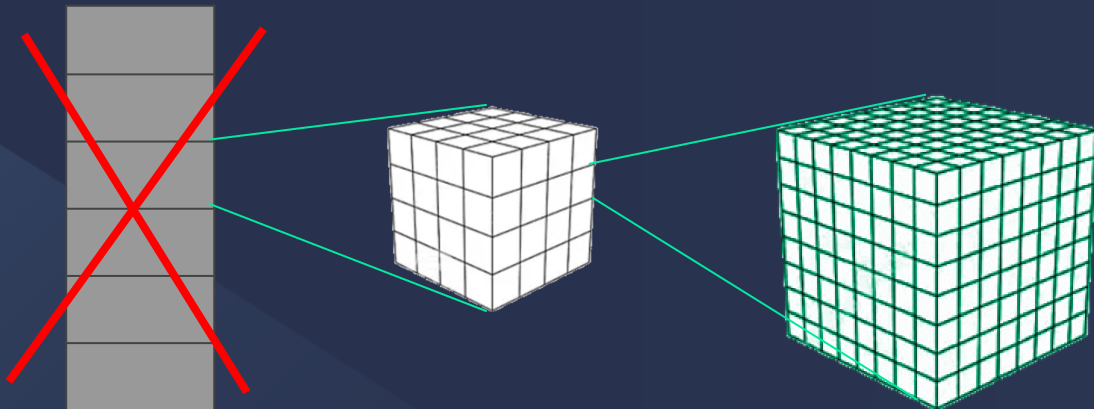


Spatial coherency in LIDAR and RGB-D pointclouds

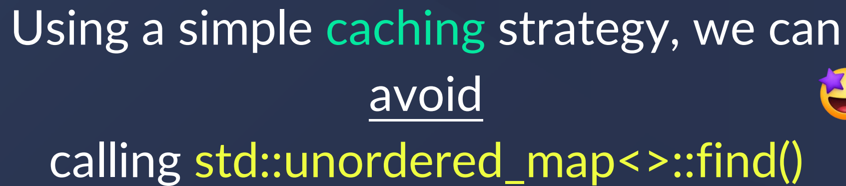
Distribution of points in PointClouds is not completely “random”.
When data is generated by a RGBD camera or a LIDAR,
a point has a high probability to be close of the previous one



Spatial coherency and caching



High probability that the LeafGrid is the same, for two consecutive points

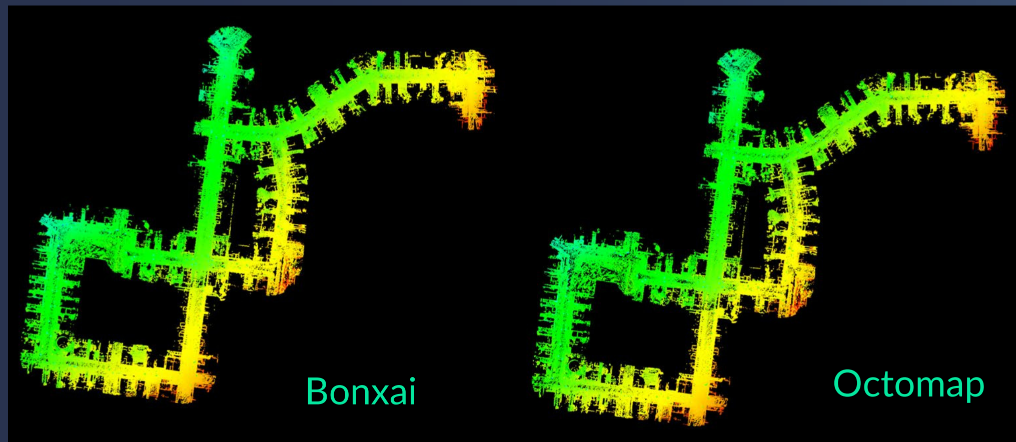


Using a simple caching strategy, we can avoid calling std::unordered_map<>::find()

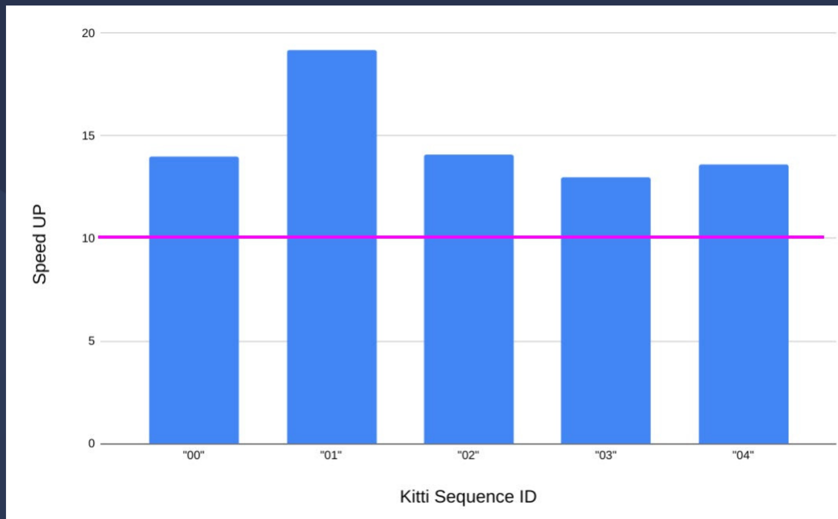
2X faster cell access, using real-world data!!!

Bonxai-based probabilistic mapping

Using Bonxai + Eigen,
I reimplemented the Octomap probabilistic map algorithm, including raycasting,
in 250 lines of code



Lie: “Bonxai is 10x faster than Octomap”

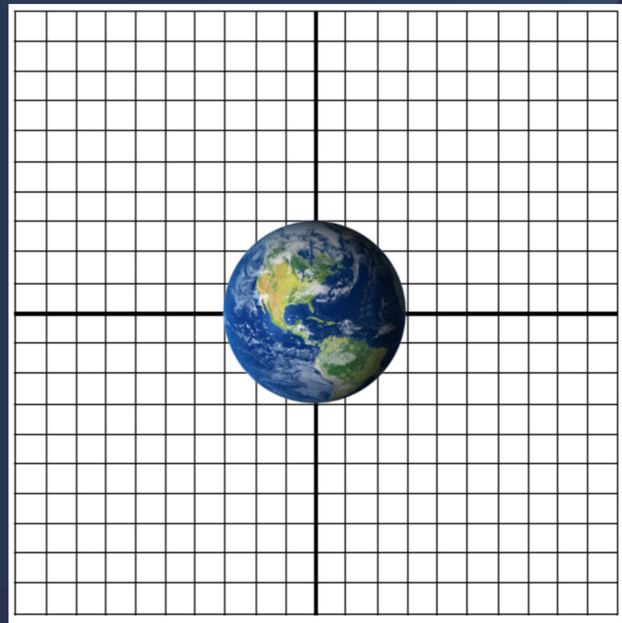


Actually, 13-19x faster on the Kitti LIDAR odometry datasets

Lie: “Bonxai is Unbounded”

Since we use 32 bits indexes to represent the voxel coordinates, the maximum representable space is actually limited.

Given a voxel size equal to 1 cm, the maximum size of the grid is about 40,000 Km.



Lie: “Bonxai is Sparse”

Technically, it is a Sparse-Dense structure, since the grid at the bottom layers is dense.

In practice, memory overhead is not a problem.

Lie: “this presentation is about 3D mapping”

No, this talk is about a data structure
that you can easily add to your projects

Just copy “bonxai/bonxai.hpp” into your “3rdparty” folder and enjoy your life!

PICKNIK

Thank you for your attention :)

https://github.com/facontidavide/Bonxai