Applied Data Structures
Disclaimer

This lecture is ...

... not a mathematically rigorous treatise of data structures

but

... concerned with **combining** data structures to solve complex problems

... focused on practical aspects of **end-to-end** implementations

... a winding journey of trade-offs :-(
The Audience ... 

... is expected to have first-hand experience with (or at least some general understanding)

- standard computer graphics data structures
- algodat
- geometry
- linear algebra

also useful

- general understanding of computer architecture (OS, Memory, CPU, I/O, ...)

---
Terminology

- **Vectors**
  - V[234][fdil]
  - e.g. V3d is a vector of 3 double values

- **Matrices**
  - M[234][234][fdil]
  - e.g. M44f is a 4x4 matrix of float values

- **Axis-aligned Bounding Boxes**
  - Box[23][fdil]
  - e.g. Box3d is a 3-dim double-precision AABB ranging from .Min to .Max

https://github.com/aardvark-platform/aardvark.docs/wiki/Vectors-and-Matrices
Overview

● starting with a non-trivial problem

● we will identify big and small challenges

● solve them step-by-step

● by choosing and combining various data structures and algorithms

● designing a full end-to-end system
But First:

Some Thoughts on Data Representation
Representing Data (Iteration 1)

class PointCloud
{
    V3d[] Positions;
}

Trivial problem statement:
Storing a set of points ...
What about Multiple Attributes?
Multiple Attributes (Iteration 2)

class Point
{
    V3d Position;
    C4b Color;
}

class PointCloud
{
    Point[] Points;
}

interface IPoint { ... }
interface IPointWithColor { ... }
class Point : IPoint
{
    V3d Position;
}

class ColoredPoint
    : Point, IPointWithColor
{
    C4b Color;
}
...

class PointCloud<T> where T > IPoint
{
    T[] Points;
}
class Point
{
  V3d Position;
  C4b Color;
}

class ColoredPoint : Point
{
  C4b Color;
}

class PointCloud<T>
{
  T[] Points;
}

Multiple Attributes (Iteration 2)

Why?

- positions and colors interleaved in memory
- bad cache coherence
- different attributes usually processed and managed independently (queries, GPU, …)
  -> data separation? -> excessive copying
- bad extensibility (backwards compatibility!)
class PointCloud
{
    V3d[] Positions;
    C4b[] Colors;
}

Multiple Attributes (Iteration 3)

- transpose your data *(store columns, not rows)*
- data stored as **dense** as possible => optimal cache coherence
- **optimize** your algorithms for **dense arrays**
- (de)serialization is basically **memcpy**, no parsing overhead
Extensibility
Representing Data (Iteration 4)

class PointCloud
{
    V3d[] Positions;
    C4b[] Colors;
}

- existing data is not affected

- if there are **many optional** attributes, or **custom** attributes, we can even go further -> next slide
Representing Data (Iteration 5)

class PointCloud
{
    Dictionary<string, Array> data;

    // standard attributes ...
    V3d[] Positions => (V3d[])data["Positions"];
    C4b[] Colors => (C4b[])data["Colors"];

    // custom attributes ...
    T[] Get<T>(string key) => (T[])data[key];
}

● switch to **key/value representation** (hash table) - named attributes
● with strongly typed accessors for standard attributes
A More Complex Example
class Mesh
{
    V3d[] Positions;
    int[] FirstIndexArray;
    int[] VertexIndexArray;
}

Positions

| v0 | v1 | v2 | v3 | v4 | v5 |

FirstIndexArray

| 0  | 4  | 8  | 11 | 15 | 18 |

VertexIndexArray

| 0  | 1  | 2  | 3  | 4  | 5  | 6  | 7  | 8  | 9  | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 |
|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|----|
| 0  | 1  | 4  | 3  | 1  | 2  | 5  | 4  | 3  | 4  | 5  | 2  | 0  | 3  | 5  | 0  | 2  | 1  |
class Mesh
{
    V3d[] Positions;
    int[] FirstIndexArray;
    int[] VertexIndexArray;
}

class Face
{
    Mesh m;
    int fi; // face index

    V3d[] Points =>
    {
        var start = FirstIndexArray[fi];
        var count = FirstIndexArray[fi + 1] - start;
        return m.VertexIndexArray
            .Map(start, count, i => m.Positions[i])
            ;
    }
}
Mesh Topology (out-of-scope)

- store topology as additional index-arrays
  - face-edge and edge-face refs (via indices)
- provide facades for faces, edges, vertices
- traversal via facades for higher-level algorithms
  - face -> vertices
  - face -> edges
  - edge -> start- and end-vertex
  - edge -> adjacent faces
  - vertex -> adjacent edges
  - vertex -> adjacent faces

OO modeling of **Face, Edge, Vertex** with pointers or references to adjacent objects is extremely inefficient for large meshes:

- expensive memory management
- memory fragmentation
- no cache coherence
- large memory overhead (unfavorable pointer to payload ratio)
- serialization nightmare
A Non-Trivial Problem
Wishlist

Create an engine for **laser scan data**!

- rendering
- querying
- editing
  - advanced (geometry fitting, segmentation, labeling, …)

**No limit** on data size.

**Full fidelity** (preserve original data).

**Real-time.**

An initial non-real-time pre-processing pass (import) over the raw laser scan data file(s) is allowed.
Wishlist

Create an engine for **laser scan data**!

- rendering
- querying
- editing
  - advanced (geometry fitting, segmentation, labeling, …)

**No limit** on data size.

**Full fidelity** (preserve original data).

**Real-time.**

An initial non-real-time pre-processing pass (import) over the raw laser scan data file(s) is allowed.

<table>
<thead>
<tr>
<th>Questions</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>what is laser scan data</td>
<td>?</td>
</tr>
<tr>
<td>data format(s)</td>
<td>?</td>
</tr>
<tr>
<td>memory (RAM)</td>
<td>?</td>
</tr>
<tr>
<td>storage (disk, cloud)</td>
<td>?</td>
</tr>
<tr>
<td>rendering full data in real-time</td>
<td>?</td>
</tr>
<tr>
<td>editing full data in real-time</td>
<td>?</td>
</tr>
<tr>
<td>queries</td>
<td>?</td>
</tr>
</tbody>
</table>
Example (I) - Terrestrial Scans

Image from Elseberg's paper [4].
### Non-Uniform Point Density in Terrestrial Scans

<table>
<thead>
<tr>
<th>minDist</th>
<th>pointCount</th>
</tr>
</thead>
<tbody>
<tr>
<td>0,000</td>
<td>503 077 362</td>
</tr>
<tr>
<td>0,001</td>
<td>412 719 452</td>
</tr>
<tr>
<td>0,002</td>
<td>295 266 794</td>
</tr>
<tr>
<td>0,003</td>
<td>218 579 612</td>
</tr>
<tr>
<td>0,004</td>
<td>170 176 333</td>
</tr>
<tr>
<td>0,005</td>
<td>134 221 818</td>
</tr>
<tr>
<td>0,006</td>
<td>108 402 370</td>
</tr>
<tr>
<td>0,007</td>
<td>88 099 143</td>
</tr>
<tr>
<td>0,008</td>
<td>73 300 114</td>
</tr>
<tr>
<td>0,009</td>
<td>51 733 965</td>
</tr>
<tr>
<td>0,010</td>
<td>53 235 078</td>
</tr>
<tr>
<td>0,011</td>
<td>46 581 533</td>
</tr>
<tr>
<td>0,012</td>
<td>40 895 708</td>
</tr>
<tr>
<td>0,013</td>
<td>36 254 266</td>
</tr>
<tr>
<td>0,014</td>
<td>32 309 962</td>
</tr>
<tr>
<td>0,015</td>
<td>28 857 681</td>
</tr>
<tr>
<td>0,016</td>
<td>25 497 297</td>
</tr>
<tr>
<td>0,017</td>
<td>22 825 726</td>
</tr>
<tr>
<td>0,018</td>
<td>20 683 994</td>
</tr>
<tr>
<td>0,019</td>
<td>18 835 069</td>
</tr>
<tr>
<td>0,020</td>
<td>17 268 076</td>
</tr>
</tbody>
</table>
Example (II) - Airborne Scans

- mostly uniform point density
- scanned from constant height
Example (III) - Photogrammetric Point Clouds
Test Data

- try to get
  - as much test data as possible
  - as diverse as possible
  - as soon as possible

- currently, data sets with
  - $10^5$ to $10^8$ points are quite easy to find on the internet
  - $10^8$ up to $10^9$s exist and are available in research and/or commercial settings
  - larger data sets are mostly aggregated from multiple scans, e.g.
    - airborne laser scans of whole regions or countries
    - terrestrial scans of cities
    - ...
Data Example (I)

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>(intensity)</th>
<th>R</th>
<th>G</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>-7166.761089</td>
<td>256966.445991</td>
<td>708.332506</td>
<td>-1980 17 10 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7166.577801</td>
<td>256966.331306</td>
<td>708.548418</td>
<td>-1906 6 2 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7166.604809</td>
<td>256966.280403</td>
<td>708.439592</td>
<td>-1938 9 1 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7166.432018</td>
<td>256966.152442</td>
<td>708.516283</td>
<td>-1916 6 2 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7166.269207</td>
<td>256968.251136</td>
<td>708.547441</td>
<td>-1967 10 6 5</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7165.113873</td>
<td>256971.195075</td>
<td>708.262438</td>
<td>-1861 10 7 2</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7165.124554</td>
<td>256971.175849</td>
<td>708.065416</td>
<td>-1776 9 6 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7165.123882</td>
<td>256971.177070</td>
<td>707.967394</td>
<td>-1740 9 5 6</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7166.600414</td>
<td>256969.778907</td>
<td>705.126268</td>
<td>-1939 3 3 3</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7166.595013</td>
<td>256969.788398</td>
<td>705.023577</td>
<td>-1987 1 1 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7165.346966</td>
<td>256971.978981</td>
<td>705.400560</td>
<td>-1942 4 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7164.675823</td>
<td>256971.175758</td>
<td>708.665148</td>
<td>-1977 9 6 1</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7165.126080</td>
<td>256971.173072</td>
<td>707.869249</td>
<td>-1912 6 2 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7165.143841</td>
<td>256971.141090</td>
<td>707.672411</td>
<td>-1857 4 0 0</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7164.823223</td>
<td>256971.321083</td>
<td>708.713060</td>
<td>-1990 10 7 2</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

A common open file format for laser scan data is LAS

http://www.asprs.org/a/society/committees/standards/LAS_1_4_r13.pdf

or LASzip, the compressed variant

These are binary formats, so we use .pts (left) for demonstration purposes.

Example .pts file (ASCII)
### Data Example (II)

<table>
<thead>
<tr>
<th>X</th>
<th>Y</th>
<th>Z</th>
<th>I</th>
<th>R</th>
<th>G</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>-7166.761089</td>
<td>256966.445991</td>
<td>708.332506</td>
<td>-1980.17</td>
<td>10</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>-7166.577801</td>
<td>256966.331306</td>
<td>708.548418</td>
<td>-1906.62</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7166.604809</td>
<td>256966.280403</td>
<td>708.439592</td>
<td>-1938.91</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7166.432018</td>
<td>256966.152442</td>
<td>708.516283</td>
<td>-1916.62</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7166.269207</td>
<td>256968.251136</td>
<td>708.547441</td>
<td>-1967.10</td>
<td>6</td>
<td>5</td>
<td></td>
</tr>
<tr>
<td>-7165.113873</td>
<td>256971.195075</td>
<td>708.262438</td>
<td>-1861.10</td>
<td>7</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>-7165.124554</td>
<td>256971.177676</td>
<td>708.96</td>
<td>-1976</td>
<td>9</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>-7165.123882</td>
<td>256971.177676</td>
<td>708.96</td>
<td>-1976</td>
<td>9</td>
<td>6</td>
<td>1</td>
</tr>
<tr>
<td>-7166.600414</td>
<td>256969.778907</td>
<td>705.126268</td>
<td>-1939.33</td>
<td>3</td>
<td>3</td>
<td></td>
</tr>
<tr>
<td>-7165.395013</td>
<td>256969.788398</td>
<td>705.023577</td>
<td>-1987.11</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>-7165.346966</td>
<td>256971.978981</td>
<td>705.400560</td>
<td>-1942.40</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td>-7164.675823</td>
<td>256971.175758</td>
<td>708.665148</td>
<td>-1977.96</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7165.126080</td>
<td>256971.173072</td>
<td>707.869249</td>
<td>-1912.62</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7165.143841</td>
<td>256971.141090</td>
<td>707.672411</td>
<td>-1857.40</td>
<td>0</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-7164.823223</td>
<td>256971.321083</td>
<td>708.713060</td>
<td>-1990.10</td>
<td>7</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

**Questions**

<p>| | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>what is laser scan data</td>
<td>OK</td>
</tr>
<tr>
<td>data format(s)</td>
<td>OK</td>
</tr>
<tr>
<td>memory (RAM)</td>
<td>?</td>
</tr>
<tr>
<td>storage (disk, cloud)</td>
<td>?</td>
</tr>
<tr>
<td>rendering full data in real-time</td>
<td>?</td>
</tr>
<tr>
<td>editing full data in real-time</td>
<td>?</td>
</tr>
<tr>
<td>queries</td>
<td>?</td>
</tr>
<tr>
<td>coordinate systems</td>
<td>?</td>
</tr>
<tr>
<td>precision</td>
<td>?</td>
</tr>
</tbody>
</table>

**Interesting!**

12 significant digits
Coordinate Systems (Out-of-Scope)

EPSG Registry (currently 6000+ coordinate reference systems)
http://www.epsg.org/

<table>
<thead>
<tr>
<th>Code</th>
<th>CRS Name</th>
<th>Area code</th>
<th>CRS Type</th>
<th>Coord Sys code</th>
<th>Datum code</th>
<th>Base CRS Code</th>
<th>Proj Conv Code</th>
<th>Compd</th>
<th>Compd Vert</th>
<th>CRS Scope</th>
</tr>
</thead>
<tbody>
<tr>
<td>2307</td>
<td>Qornooq 1277 / Greenland zone 8 east</td>
<td>3846</td>
<td>projected</td>
<td>1031</td>
<td>4104</td>
<td>18428</td>
<td>Topographic mapping.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2308</td>
<td>Batala / TM 109 SE</td>
<td>2577</td>
<td>projected</td>
<td>4400</td>
<td>4211</td>
<td>16709</td>
<td>Used by Arc and BP for ONWJ.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2309</td>
<td>WGS 84 / TM 110 SE</td>
<td>2578</td>
<td>projected</td>
<td>4400</td>
<td>4326</td>
<td>16716</td>
<td>Used by BP for Terang-Sirasun.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2310</td>
<td>WGS 84 / TM 132 SE</td>
<td>2589</td>
<td>projected</td>
<td>4400</td>
<td>4326</td>
<td>10732</td>
<td>Used for hydrocarbons exploration and production.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2311</td>
<td>WGS 84 / TM 0 NE</td>
<td>2591</td>
<td>projected</td>
<td>4400</td>
<td>4326</td>
<td>10406</td>
<td>Oil industry exploration and production.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2312</td>
<td>Garoua / UTM zone 33N</td>
<td>2590</td>
<td>projected</td>
<td>4400</td>
<td>4197</td>
<td>10033</td>
<td>Oil industry exploration and production.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2313</td>
<td>Kousoin / UTM zone 33N</td>
<td>2591</td>
<td>projected</td>
<td>4400</td>
<td>4198</td>
<td>10033</td>
<td>Oil industry exploration and production.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2314</td>
<td>Trinidad 1903 / Trinidad Grid (TCA)</td>
<td>1339</td>
<td>projected</td>
<td>4400</td>
<td>4302</td>
<td>19975</td>
<td>Large and medium scale topographic mapping.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2315</td>
<td>Campo Inchauspe / UTM zone 196</td>
<td>2586</td>
<td>projected</td>
<td>4400</td>
<td>4221</td>
<td>16119</td>
<td>Oil industry exploration.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2316</td>
<td>Campo Inchauspe / UTM zone 209</td>
<td>2589</td>
<td>projected</td>
<td>4400</td>
<td>4221</td>
<td>16120</td>
<td>Oil industry exploration.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2317</td>
<td>PAGS56 / ICDN Regional</td>
<td>3327</td>
<td>projected</td>
<td>4409</td>
<td>4248</td>
<td>19976</td>
<td>Oil industry exploration and production.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2318</td>
<td>Ain el Abd / Arousco Lambert</td>
<td>3303</td>
<td>projected</td>
<td>4404</td>
<td>4304</td>
<td>19977</td>
<td>Oil industry exploration and production.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2319</td>
<td>ED60 / TM27</td>
<td>1524</td>
<td>projected</td>
<td>4530</td>
<td>4230</td>
<td>16305</td>
<td>Large scale topographic mapping, cad.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2320</td>
<td>ED60 / TM30</td>
<td>1525</td>
<td>projected</td>
<td>4530</td>
<td>4230</td>
<td>16370</td>
<td>Large scale topographic mapping, cad.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2321</td>
<td>ED60 / TM33</td>
<td>1526</td>
<td>projected</td>
<td>4530</td>
<td>4230</td>
<td>16370</td>
<td>Large scale topographic mapping, cad.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2322</td>
<td>ED60 / TM35</td>
<td>1527</td>
<td>projected</td>
<td>4530</td>
<td>4230</td>
<td>16372</td>
<td>Large scale topographic mapping, cad.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2323</td>
<td>ED50 / TM30</td>
<td>1528</td>
<td>projected</td>
<td>4530</td>
<td>4230</td>
<td>16307</td>
<td>Large scale topographic mapping, cad.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2324</td>
<td>ED50 / TM42</td>
<td>1529</td>
<td>projected</td>
<td>4530</td>
<td>4230</td>
<td>16374</td>
<td>Large scale topographic mapping, cad.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2325</td>
<td>ED50 / TM45</td>
<td>1530</td>
<td>projected</td>
<td>4530</td>
<td>4230</td>
<td>16308</td>
<td>Large scale topographic mapping, cad.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2326</td>
<td>Hong Kong 1080 Grid System</td>
<td>1118</td>
<td>projected</td>
<td>4500</td>
<td>4011</td>
<td>19976</td>
<td>Large scale topographic mapping, cad.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2327</td>
<td>Xian 1980 / Gauss-Krüger zone 13</td>
<td>1587</td>
<td>projected</td>
<td>4530</td>
<td>4010</td>
<td>10213</td>
<td>Medium scale topographic mapping.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2328</td>
<td>Xian 1980 / Gauss-Krüger zone 14</td>
<td>1588</td>
<td>projected</td>
<td>4530</td>
<td>4010</td>
<td>10214</td>
<td>Medium scale topographic mapping.</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2329</td>
<td>Xian 1980 / Gauss-Krüger zone 15</td>
<td>1590</td>
<td>projected</td>
<td>4530</td>
<td>4010</td>
<td>10216</td>
<td>Medium scale topographic mapping.</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Coordinate Systems (Out-of-Scope)

Data can be

- single scan with scanner as origin (local coordinate system)
- multiple (overlapping) scans registered to some local origin
- scan(s) registered to some terrestrial coordinate system

Additional Questions (out-of-scope)

- conversion
- keeping track of metadata (e.g. coordinate systems)
- other data sources, e.g. photogrammetric points clouds
- registration (of different point clouds, coordinate systems)
Precision

32-bit floating point

<table>
<thead>
<tr>
<th>distance</th>
<th>resolution</th>
</tr>
</thead>
<tbody>
<tr>
<td>1 m</td>
<td>~1 μm</td>
</tr>
<tr>
<td>1 km</td>
<td>~1 mm</td>
</tr>
<tr>
<td>10 km</td>
<td>~1 cm</td>
</tr>
<tr>
<td>6.371 km</td>
<td>~1 m</td>
</tr>
</tbody>
</table>

64-bit floating point

| 6371 km   | ~1 nm      |
| 384.402 km| ~1 μm      |
| 150.000.000 km | ~1 mm   |
Precision, Again!

- original data is double precision (float64)
- double precision necessary to precisely represent objects in large spaces
- GPU usually works with float32
- float64 effectively doubles memory consumption
- **solution**: use float64 local origin, and float32 offsets per point
Representing Data (Iteration 6)

class PointCloud
{
    V3d Offset;
    V3f[] Positions;
    C4b[] Colors;
}

Questions

- what is laser scan data: OK
- data format(s): OK
- memory (RAM): ?
- storage (disk, cloud): ?
- rendering full data in real-time: ?
- editing full data in real-time: ?
- queries: ?
- coordinate systems: OK
- precision: OK
- precision (GPU): ?
How to Set Up Correct Transforms for GPU (float32)

O. real-world space

P. point cloud space

\[
\text{offset : } \mathbb{V}^3_d \\
\text{p : } \mathbb{V}^3_f
\]
How to set up correct float32 transforms for GPU

O. real-world space

P. point cloud space
  offset : V3d
  p     : V3f

W. world space (rendering)
  w     : V3d
How to set up correct float32 transforms for GPU

O. real-world space

P. point cloud space
   
   offset : V3d
   p : V3f

W. world space (rendering)
   
   w : V3d

C. camera space
   
   c : M44d
How to set up correct float32 transforms for GPU

O. real-world space

P. point cloud space
  \( \text{offset} : \mathbb{V}^3_d \)
  \( p : \mathbb{V}^3_f \)

W. world space (rendering)
  \( w : \mathbb{V}^3_d \)

C. camera space
  \( c : \mathbb{M}^{44}_d \)

view transform \( v = c - w \)

model transform \( m = \text{offset} - w \)
How to set up correct float32 transforms for GPU

O. real-world space

P. point cloud space

\[ \text{offset} : \mathbf{V3d} = (251987, 0, 0) \]
\[ \mathbf{p} : \mathbf{V3f} = (1, 2, 0) \]

W. world space (rendering)

\[ \mathbf{w} : \mathbf{V3d} = (251000, 0, 0) \]

C. camera space

\[ \mathbf{c} : \mathbf{M44d} = (251980, 0, 0) \]

view transform \[ \mathbf{v} = \mathbf{c} - \mathbf{w} \]

model transform \[ \mathbf{m} = \text{offset} - \mathbf{w} \]

GPU-ready can be cast to float32 without loss of precision
Spatial Data Structures
Basic Idea

tree-like structures

recursively divide space containing primitives

to quickly traverse to region(s) of interest

instead of testing all $n$ primitives, e.g. for queries, we get away with approximately $\log(n)$ tests

$\sim O(\log n)$

instead of

$O(n)$
Queries

- n-closest points
  - point
  - ray
  - plane
- points contained in X
  - aabb
  - sphere
  - frustum (special case of convex hull)
  - convex hull
- resampling
  - e.g. density (similar to n-closest points)

Questions

<table>
<thead>
<tr>
<th>Question</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>what is laser scan data</td>
<td>OK</td>
</tr>
<tr>
<td>data format(s)</td>
<td>OK</td>
</tr>
<tr>
<td>memory (RAM)</td>
<td>?</td>
</tr>
<tr>
<td>storage (disk, cloud)</td>
<td>?</td>
</tr>
<tr>
<td>rendering full data in real-time</td>
<td>?</td>
</tr>
<tr>
<td>editing full data in real-time</td>
<td>?</td>
</tr>
<tr>
<td>queries</td>
<td>?</td>
</tr>
<tr>
<td>coordinate systems</td>
<td>OK</td>
</tr>
<tr>
<td>precision</td>
<td>OK</td>
</tr>
<tr>
<td>precision (GPU)</td>
<td>OK</td>
</tr>
</tbody>
</table>
Choosing Appropriate Data Structures

Octree

- regular space subdivision, blocks
- easy processing, storage
- LoDs, subsets, contained-in

kd-Tree

- n-closest points queries, picking

Editing

- e.g. point cloud cleansing -> removing/deleting points or regions
- Which data structures? We’ll see later ...

Questions

<table>
<thead>
<tr>
<th></th>
<th>OK</th>
</tr>
</thead>
<tbody>
<tr>
<td>what is laser scan data</td>
<td></td>
</tr>
<tr>
<td>data format(s)</td>
<td></td>
</tr>
<tr>
<td>memory (RAM)</td>
<td>?</td>
</tr>
<tr>
<td>storage (disk, cloud)</td>
<td>?</td>
</tr>
<tr>
<td>rendering full data in real-time</td>
<td>?</td>
</tr>
<tr>
<td>editing full data in real-time</td>
<td>?</td>
</tr>
<tr>
<td>queries</td>
<td>OK</td>
</tr>
<tr>
<td>coordinate systems</td>
<td>OK</td>
</tr>
<tr>
<td>precision</td>
<td>OK</td>
</tr>
<tr>
<td>precision (GPU)</td>
<td>OK</td>
</tr>
</tbody>
</table>
Octree
Octree

- recursively subdivide box into its 8 octants
- we use it to subdivide space containing points
- is also frequently used to represent volume data, where the octree itself is the object of interest
class OctreeNode
{
    Box3d BBox;
    V3d Offset => BBox.Center;
    V3f[] Positions;
    OctreeNode[] Subnodes;
}

// Conventions
// -----------
// BBox ................... new Box3d(Positions)
// Subnodes == null ....... leaf node
// Subnodes.Length == 8 ... inner node
// Subnodes[i] == null .... i-th subnode contains no points, no subtree
Octree - Indexing

```c
int GetSubnodeIndex(V3d p) {
    var i = 0;
    if (p.X >= c.X) i = 0b0001;  // 1
    if (p.Y >= c.Y) i |= 0b0010; // 2
    if (p.Z >= c.Z) i |= 0b0100; // 4
    return i;
}
```

// where c = BBox.Center

Scheme is easily extended to any dimensions.
Octree - Construction

OctreeNode BuildOctree(Box3d bounds, V3d[] positions, int splitLimit)
{
    if (positions == null || positions.Length == 0) // empty
        return null;

    if (positions.Length <= splitLimit) // leaf node
        return new OctreeNode(positions);

    var c = bounds.Center; // inner node (split)
    var buckets = InitArray(8, _ => new List<V3d>());
    foreach (var p in positions) buckets[GetSubnodeIndex(p, c)].Add(p);
    var subnodes = buckets.Map(
        (ps, i) => BuildOctree(bounds.GetOctant(i), ps, splitLimit)
    );
    return new OctreeNode(bounds, subnodes);
}
Octree - Discussion

**pros:**
- fast
- simple implementation
- current implementation is immutable -> easy to reason about, easy to share

**cons:** memory-bound

**missing:** out-of-core, levels of details, queries, editing
Octree - Out-of-Core (I)

class OctreeNode
{
    Box3d BBox;
    V3d Offset => BBox.Center;
    V3f[] Positions;
    OctreeNode[] Subnodes;
}

PROBLEM:
In-memory references.

SOLUTION:
Break up strong linking of in-memory nodes.

HOW?
Octree - Out-of-Core (II)

if we have nodes, which

● each have a **unique identifier** (ideally globally unique)
● are **persistent** (outlive the process in which they were created)

it is possible to

● replace in-memory refs with references to persistent nodes
● load persistent nodes on-demand

**HOW?**
Universally Unique Identifiers

“A universally unique identifier (UUID) is a 128-bit number used to identify information in computer systems. The term globally unique identifier (GUID) is also used.” [https://en.wikipedia.org/wiki/universally_unique_identifier]

- standardized
- readily available for all programming languages and environments
- probability of collisions is so small it can be ignored in practice
  (e.g. when generating $10^9$ UUIDs / second for 85 years -> 50% chance to create a collision)

Example:

123e4567-e89b-12d3-a456-426655440000
class OctreeNode
{
    string Id; // could also be Guid, or byte[16]
    Box3d BBox;
    V3d Offset => BBox.Center;
    V3f[] Positions;
    OctreeNode[] Subnodes;
}
Persistent Storage

We can use a key/value store

- **key**: unique id
- **value**: node data

**toy API** for key/value store

- void **Add**(string key, byte[] blob);
- byte[] **Get**(string key);
Persistent Storage - Key/Value Stores

- **File Systems**
  not optimized for large numbers of (relatively small) files

- **NoSQL Databases**
  not optimized for large numbers of (relatively large) blobs

- **memory-mapped file**
  PRO: highest performance for local storage
  CON: complicated, equivalent to implement a memory manager or file system (efficient layout, avoid fragmentation)

- **cloud-based blob storage**
  available with key/value APIs
  useful when going fully distributed (data exceeds local storage, or serving customers)
Memory-Mapped File (General Idea)

Index:

<table>
<thead>
<tr>
<th>key offset</th>
<th>c650e37f...</th>
<th>c1a66bce...</th>
<th>1273f4bd...</th>
<th>d4ed1e11...</th>
<th>647f4c3c...</th>
<th>7e8737c8...</th>
<th>d1af05dd...</th>
<th>867a4706...</th>
<th>bf912ec1...</th>
<th>abb1eae0...</th>
<th>5125f428...</th>
</tr>
</thead>
</table>

Data

blobs
class PersistentRef<T>
{
    public string Key { get; } // only thing that needs to be persisted,
    // everything else is runtime stuff

    public T GetValue() // lazy (on demand) ... 
    {
        if (cache != null && cache.TryGetValue(out T result)) return result;
        result = load(Key);
        cache = new WeakReference<T>(result);
        return result;
    }

    public bool TryGetValue(out T value) { /* ... */ }

    private Func<string, T> load; // function to retrieve T (init at creation)
    private WeakReference<T> cache;
}
Laziness - Breaking the In-Memory Link

class OctreeNode
{
    string Id; // e.g. random UUID
    Box3d BBox;
    V3d Offset => BBox.Center;
    V3f[] Positions;
    PersistentRef<OctreeNode>[] Subnodes;
}
class OctreeNode
{
    string Id; // e.g. random UUID
    Box3d BBox;
    V3d Offset => BBox.Center;
    PersistentRef<V3f[]> Positions;
    PersistentRef<OctreeNode>[] Subnodes;
}

now we can load only structural octree information without pulling in huge amounts of data into memory due to caching in persistent references, the garbage collector (GC) will automatically keep a nice least-recently-used working set in memory
Laziness + Out-of-Core => Levels-of-Detail

now that we can pull in only portions of the octree into memory
we need simplified representations of the point cloud higher up the tree
in order to have something to render/query/edit ...

Therefore, a lazy out-of-core data structure implies levels-of-detail.
Levels-of-Detail - Toy Implementation
(Out-of-Scope)

class OctreeNode
{
    string Id;
    Box3d BBox;
    V3d Offset => BBox.Center;
    PersistentRef<V3f[]> Positions;
    PersistentRef<OctreeNode>[] Subnodes;
}

up to now, original points are stored in leaf-nodes

=> we can use Positions in inner nodes to store a simplified representation of the subtree

e.g. inner nodes could store 1/8th of the points of their subnodes
Levels-of-Detail - Toy Impl. (II)

out-of-scope:

- LoD selection metric
- LoD rendering
- ...

Questions

<table>
<thead>
<tr>
<th>Question</th>
<th>Status</th>
</tr>
</thead>
<tbody>
<tr>
<td>what is laser scan data</td>
<td>OK</td>
</tr>
<tr>
<td>data format(s)</td>
<td>OK</td>
</tr>
<tr>
<td>memory (RAM)</td>
<td>OK</td>
</tr>
<tr>
<td>storage (disk, cloud)</td>
<td>OK</td>
</tr>
<tr>
<td>rendering full data in real-time</td>
<td>OK</td>
</tr>
<tr>
<td>editing full data in real-time</td>
<td>?</td>
</tr>
<tr>
<td>queries</td>
<td>OK</td>
</tr>
<tr>
<td>coordinate systems</td>
<td>OK</td>
</tr>
<tr>
<td>precision</td>
<td>OK</td>
</tr>
<tr>
<td>precision (GPU)</td>
<td>OK</td>
</tr>
</tbody>
</table>
Building Out-of-Core Octrees
Know your baseline: Theoretical Limits

Let’s assume \( n = 1 \) Billion points.

Memory:

**Case 1:** \( n \times (3 \times \text{sizeof}(\text{V3d}) + \text{sizeof}(\text{C4b})) \Rightarrow \sim 28 \text{ GB} \\
**Case 2:** \( n \times (3 \times \text{sizeof}(\text{V3f}) + \text{sizeof}(\text{C4b})) \Rightarrow \sim 16 \text{ GB} \\
**Case 3:** \( n \times (3 \times \text{sizeof}(\text{V3f}) + \text{sizeof}(\text{C3b})) \Rightarrow \sim 15 \text{ GB} \) (alignment problems)

I/O, Bandwidth:

- SSD \( \sim 500 \text{ MB/s} \) \( 16\text{GB} / 500\text{MB/s} \) \( \Rightarrow \sim 32\text{s} \)
- HDD, Gigabit Ethernet \( \sim 100 \text{ MB/s} \) \( 16\text{GB} / 100\text{MB/s} \) \( \Rightarrow \sim 160\text{s} \)
- 100 MBit Ethernet \( \sim 10 \text{ MB/s} \) \( 16\text{GB} / 10\text{MB/s} \) \( \Rightarrow \sim 1600\text{s} \)
Building the out-of-core octree

directly build out-of-core octree from stream of raw points?
  => IO-bound, would take a very long time

**better idea:** read data in large chunks, that fit into memory

foreach chunk:
  parse data
  build octree
  store out-of-core

Now that we have created many small octrees without exceeding memory limits. What next?
Map-Reduce to the Rescue

like divide-and-conquer, but **bottom-up** instead of top-down

repeatedly **merge two octrees into one** (out-of-core)

until we end up with a single gigantic octree

This is “easy” if we keep our octrees **immutable**!

What?
[Q] Merging Immutable Octrees?

- in immutable data structures each operation changing the tree returns a copy (conceptually) of the tree which includes the change
- you can simultaneously have a reference to both the new and the old tree
- in practice, both trees share as many underlying nodes as possible
Simple Example: Immutable Binary Tree Insert

```plaintext
a
  /\   \
 5 3 7
 /     /
2  4  

b
  /
? 
```

\( a.\text{Insert}(9) \rightarrow b \)
Simple Example: Immutable Binary Tree Insert

structural overhead is appr. $O(\log n)$

almost vanishes for non-trivial nodes (e.g. our OctreeNode)
Simple Example: Immutable Binary Tree Insert

by the way, duplicated nodes 5 and 7 can still share the same data with nodes 5 and 7 via PersistentRef<T> fields
Map-Reduce (cont.)

with immutable data structures

- we can do processing in parallel
- locally on multiple cores
- distributed on a cluster of machines

in the context of octrees

- **map**: parsing chunks, building octree from chunk
- **reduce**: merging octrees

Without having to deal with shared state and synchronization!
Actual Merge - How?

Problem:

- octrees with non-aligned cells
- merge: impossible to reuse cells
- wasting storage (disk), I/O (bandwidth) and processing (CPU)
- => slow

Solution:

- make cell structure independent of actual data
- such that octrees that are created independently “accidently” end up with perfectly aligned cell bounds
Separating Spatial Layout from Data

power-of-two cell scheme

- cells of size $2^n$
- aligned at $2^n$ boundaries

struct Cell
{
    long X; long Y; long Z; int Exponent;
    Box3d ComputeBounds()
    {
        var d = Math.Pow(2.0, Exponent);
        var min = new V3d(X * d, Y * d, Z * d);
        var max = min + d; // element-wise addition
        return new Box3d(min, max);
    }
}


Question:
What happens at the origin?
**Power-of-two Cell Scheme**

- globally unique cells
- \((index_x, index_y, index_z, exponent)\)
- allows for consistent hashing schemes

<table>
<thead>
<tr>
<th>(index_x, index_y, index_z, exponent)</th>
</tr>
</thead>
<tbody>
<tr>
<td>(-2, 1, 0, 0)</td>
</tr>
<tr>
<td>(-1, 1, 0, 0)</td>
</tr>
<tr>
<td>(0, 1, 0, 0)</td>
</tr>
<tr>
<td>(1, 1, 0, 0)</td>
</tr>
<tr>
<td>(-2, 0, 0, 0)</td>
</tr>
<tr>
<td>(-1, 0, 0, 0)</td>
</tr>
<tr>
<td>(0, 0, 0, 0)</td>
</tr>
<tr>
<td>(1, 0, 0, 0)</td>
</tr>
<tr>
<td>(-2, -1, 0, 0)</td>
</tr>
<tr>
<td>(-1, -1, 0, 0)</td>
</tr>
<tr>
<td>(0, -1, 0, 0)</td>
</tr>
<tr>
<td>(1, -1, 0, 0)</td>
</tr>
<tr>
<td>(-2, -2, 0, 0)</td>
</tr>
<tr>
<td>(-1, -2, 0, 0)</td>
</tr>
<tr>
<td>(0, -2, 0, 0)</td>
</tr>
<tr>
<td>(1, -2, 0, 0)</td>
</tr>
</tbody>
</table>

Diagram showing the power-of-two cell scheme with indices and exponents.
Immutable Merge
Merging Out-of-Core Octrees (Pseudocode)

OctreeNode $\text{Merge}(\text{OctreeNode } a, \text{OctreeNode } b)$
{
(1) if $a$ is empty return $b$
    if $b$ is empty return $a$

(2) if $a$ and $b$ do not intersect then
    find smallest cell $c$ containing both $a$ and $b$
    $a' = \text{extend } a \text{ upwards until reaching } c$
    $b' = \text{extend } b \text{ upwards until reaching } c$
    return $c$

(3) if $a$ and $b$ have identical root cells then
    return $\text{MergeIdent}(a, b)$

(4) now $b$ must be rooted in exactly 1 of $a$'s subcells
    $b' = \text{extend } b \text{ upwards until reaching } a$'s root
    return $\text{MergeIdent}(a, b')$
}
Merging Out-of-Core Octrees (Pseudocode)

OctreeNode MergeIdent(OctreeNode a, OctreeNode b)
{
    precondition: a and b have same root cell (same bounds)

    (1) if a is leaf and b is leaf
        if a is empty return b; if b is empty return a
        return BuildOctree(a.Points ++ b.Points)

    (2) if a is tree and b is leaf
        return AddPointsImmutable(a, b.Points)

    (3) if a is leaf and b is tree
        return AddPointsImmutable(b, a.Points)

    (4) if a is tree and b is tree
        return new OctreeNode([0..7].Map(i => MergeIdent(a.Subnodes[i], b.Subnodes[i])))
}
Merging Out-of-Core Octrees (Pseudocode)

OctreeNode AddPointsImmutable(OctreeNode a, V3d[] ps)
{
    (1) if a is leaf
        return new OctreeNode(a.Points ++ ps)

    (2) return new OctreeNode(
            foreach subnode of a
                AddPointsImmutable(subnode, ps inside subnode)
            )
}
kd-Tree
kd-tree

Recursively split along a single dimension.
+ very **efficient n-closest** point queries (in dense data sets)
+ can be represented as **single index array** into positions!

**But we already have an octree?**
- building a separate kd-tree would double storage!
- also extremely slow for so many points!
- how to play together with octree’s LoDs?
  e.g. picking only visible points?

**Solution**: hybrid octree/kd-tree structure
* build a small kd-tree for each octree-cell (its points)
* coarse traversal using octree, then continue with kd-tree in cells that can not be excluded via octree

Hybrid Octree / kd-Tree

class OctreeNode
{
    string Id;
    Box3d BBox;
    V3d Offset => BBox.Center;
    PersistentRef<V3f[]> Positions;
    PersistentRef<KdTree> KdTree;
    PersistentRef<OctreeNode>[] Subnodes;
}

- kd-Tree can be stored as dense int-array
- indexing into Positions
- can be loaded on-demand
Vantage Point Tree (vp-tree)

instead of splitting along coordinate values,’ at each level a vantage point + radius is selected => partitions points into near and far set

+ works on all dimensions at once
+ better for low-dim structures embedded in higher-dim space (sparse data sets)


a kd-tree, but
for each split point we compute and store
the radius of the sphere containing all points
in its left and right subtrees

+ mostly retains or exceeds the kd-tree
  in dense data sets
+ mostly retains or exceeds the vp-tree
  in sparse data sets


n-Closest Point Queries in Large Point Sets. In Proceedings of
Computer Graphics International (CGI’11), Ottawa, Ontario, Canada,
Editing
Editing (General Idea)

Selection Operations (with lasso):
- select
- union
- subtract
- xor
- invert
- delete
Editing - Remove Surroundings via Invert
Selection is defined by

- world-to-screen transform
- 2-dim selection polygon (screenspace)

Inside/Outside tests

- recursively compute octree-cell contours (polygons in screenspace)
- fast 2d-polygon intersection tests
- in leafs: test points against lasso
Cell/Lasso Inside-Outside Tests

Selection is defined by
- world-to-screen transform
- 2-dim selection polygon (screenspace)

Inside/Outside tests
- recursively compute octree-cell contours (polygons in screenspace)
- fast 2d-polygon intersection tests
- in leafs: test points against lasso
Editing - Immediate Visual Feedback

- supply shader with selection definition
- perform tests on per-fragment basis
- color “selected” fragments
- discard “deleted” fragments
Immutable Delete

OctreeNode DeleteSelection(OctreeNode n, Selection s) {
  (1) if n.Bounds outside s then return n
  (2) if n.Bounds inside s then return null
  (3) if n is leaf then
    return new OctreeNode(
      n.Points.Filter(p => p outside s)
    )
  (4) return new OctreeNode(  
    [0..7].Map(i => DeleteSelection(n.Subnodes[i], s))  
  )
}

Questions

what is laser scan data | OK
data format(s) | OK
memory (RAM) | OK
storage (disk, cloud) | OK
rendering full data in real-time | OK
editing full data in real-time | OK
queries | OK
coordinate systems | OK
precision | OK
precision (GPU) | OK
Undo/Redo

- already solved!
- each editing step creates new immutable octree
- immutable data structure is natural solution
- you would probably invent an equivalent implementation
Out-of-Core$^2$ => Distributed Data Structures
Distributed Data Structures

General Idea:

- distribute your data structure over multiple computers
  => potentially unlimited size

- once we have an out-of-core implementation
  => this is only a small step

- simplest solution:
  use existing distributed key/value stores
  e.g. blob storage in the cloud
Data Locality

General Idea:

- move computation to where the data is
- don’t move large amounts of data around for computation
- example: use spatial index (cells) to store spatially close data on same machine
Consistent Hashing

General Idea:

- a distributed system (e.g. map/reduce cluster) can be seen as a hash table with each machine/node being a slot

- consistent hashing:
  only ~\( \frac{K}{n} \) keys need to be remapped when resizing hash table, i.e. if a machine is added or removed
  \((K \ldots \text{number of keys}, n \ldots \text{number of slots/machines})\)
Cryptographic Hash Functions

General Idea:

- deterministic key creation
  - no coordination required
  - independent actors can come up with identical keys for same data/results

- use cryptographic hash function over data
  - e.g. MD5 over raw point data

- lookup if data has already been created/processed,
  - i.e. (distributed) key/value store as cache
Hash-Trees / Merkle-Trees

General Idea:

- leaf nodes labelled with data or hash of data
- inner nodes are labelled with hash of labels of child nodes
- such a tree might represent series of computations on data (e.g. out-of-core octree generation via map/reduce)

\[ f(\text{inputdata}) \rightarrow \text{outputdata} \]
\[ \text{hash}(\text{key}_{\text{inputdata}}, \text{key}_f) \rightarrow \text{key}_{\text{outputdata}} \]
Summary

- Transposed Data, Facades
- Mesh, Topology
- Geodetic Coordinate Systems
- Precision (32 vs 64 bits)
- Octrees
- Lazy References, UUIDs
- Levels-of-Detail
- Power-of-two cell scheme
- Persistent Data Structures
- Immutable Octree Merge
- Map/Reduce

- Consistent Hashing
- Cryptographic Hash Functions
- Merkle-Trees, Hash-Trees
- kd-tree, vp-tree, rkd-tree
- Hybrid Octree/kd-tree
- Set Operations on Polygons
- Inside/Outside Tests
- Lazy Operations
Octree Publications

[1] Wimmer, Scheiblauer
Instant Points: Fast Rendering of Unprocessed Point Clouds

[2] Scheiblauer, Wimmer
Analysis of Interactive Editing Operations for Out-of-Core Point-Cloud Hierarchies

An Out-of-core Octree for Massive Point Cloud Processing
http://rs.tudelft.nl/~rlindenbergh/workshop/WenzelIQmulus.pdf

[4] Elseberg, Borrmann, Nüchter
One billion points in the cloud – an octree for efficient processing of 3D laser scans