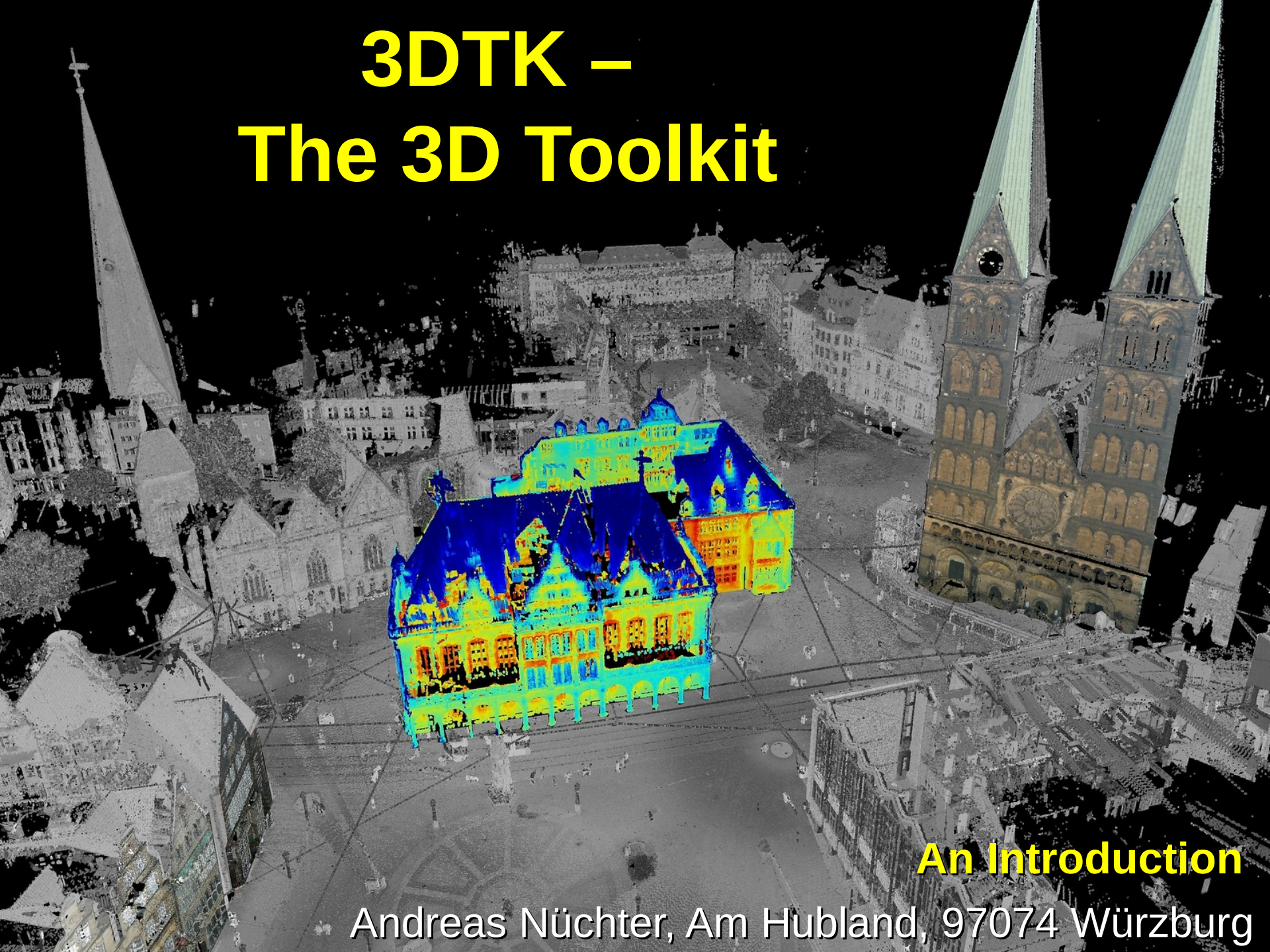
# 3DTK –
# The 3D Toolkit



**An Introduction**

Andreas Nüchter, Am Hubland, 97074 Würzburg

# 3DTK – The 3DToolkit

- Toolkit means

  ...it is no library!

  ...set of programs that perform a certain job

- GNU GPLv3 license with a few additional restrictions
- Same code may be licensed in some other ways as we are the sole authors and copyright holders of the work.

# History

2000-2001

Fraunhofer Institut Autonome Intelligente Systeme

2002-2004

UNIVERSITÄT OSNABRÜCK

2004-2009

Automation JACOBS UNIVERSITY

2009-2013

since 2013

Julius-Maximilians-UNIVERSITÄT WÜRZBURG

# 3DTK @ Sourceforge

3DTK - The 3D Toolkit / Code / [r1286] /trunk - Mozilla Firefox

File Edit View History Bookmarks Tools Help

3DTK - The 3D Toolkit / C...

https://sourceforge.net/p/slam6d/code/HEAD/tree/trunk/

Search

sourceforge | Search | Browse | Enterprise | Blog | Deals | Help | Create | Me ▾ | 7

SOLUTION CENTERS | Go Parallel | Resources | Newsletters | Cloud Storage Providers | Business VoIP Providers | Internet Speed Test | Call Center Providers

Home / Browse / 3DTK - The 3D Toolkit / Code

## 3DTK - The 3D Toolkit

Brought to you by: dborrman, josch1337, klingemann, nuechter

Summary | Files | Reviews | Support | Wiki | News | Discussion | Code | Admin | Tickets | Add New...

▸ Admin - Code

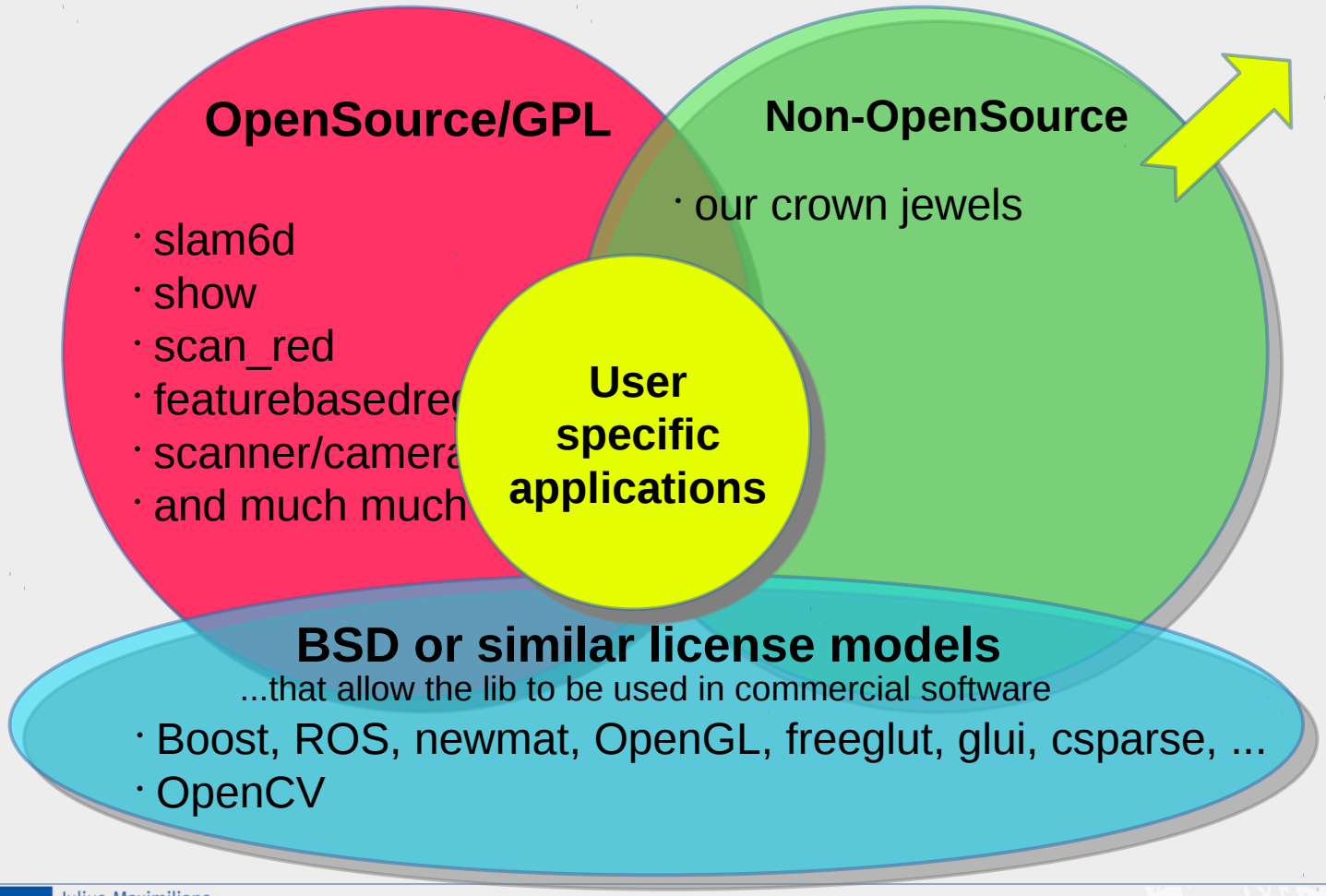
Browse Commits

Tree [r1286] / trunk / | Download Snapshot | History

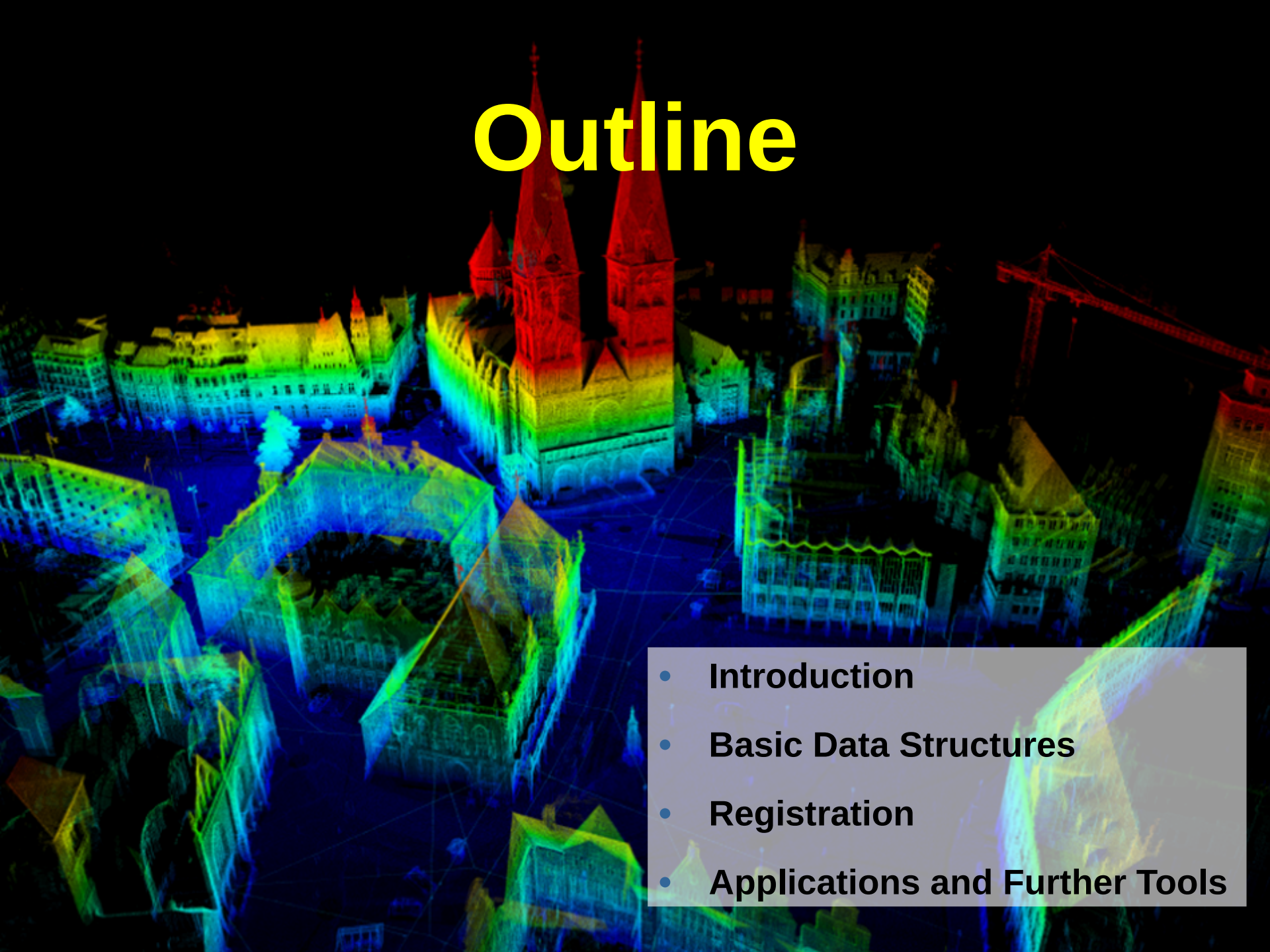
SSH | HTTPS | RO | HTTPS access | svn checkout --username=nuechter https://svn.code.sf.net/p/slam6d/co

| File | Date | Author | Commit |
|---|---|---|---|
| 3rdparty | 2016-10-15 | bleier | [r1221] apriltag: Updated to current version. |
| bin | 2013-10-08 | nuechter | [r855] reactivated xyz |
| bindings | 2017-01-04 | josch1337 | [r1274] bindings/python/py3dtk.cc: KIT scanio was removed |
| dat | 2009-09-30 | klingemann | [r67] Deleted the frame files from the dat directory.... |

# 3DTK – The 3DToolkit

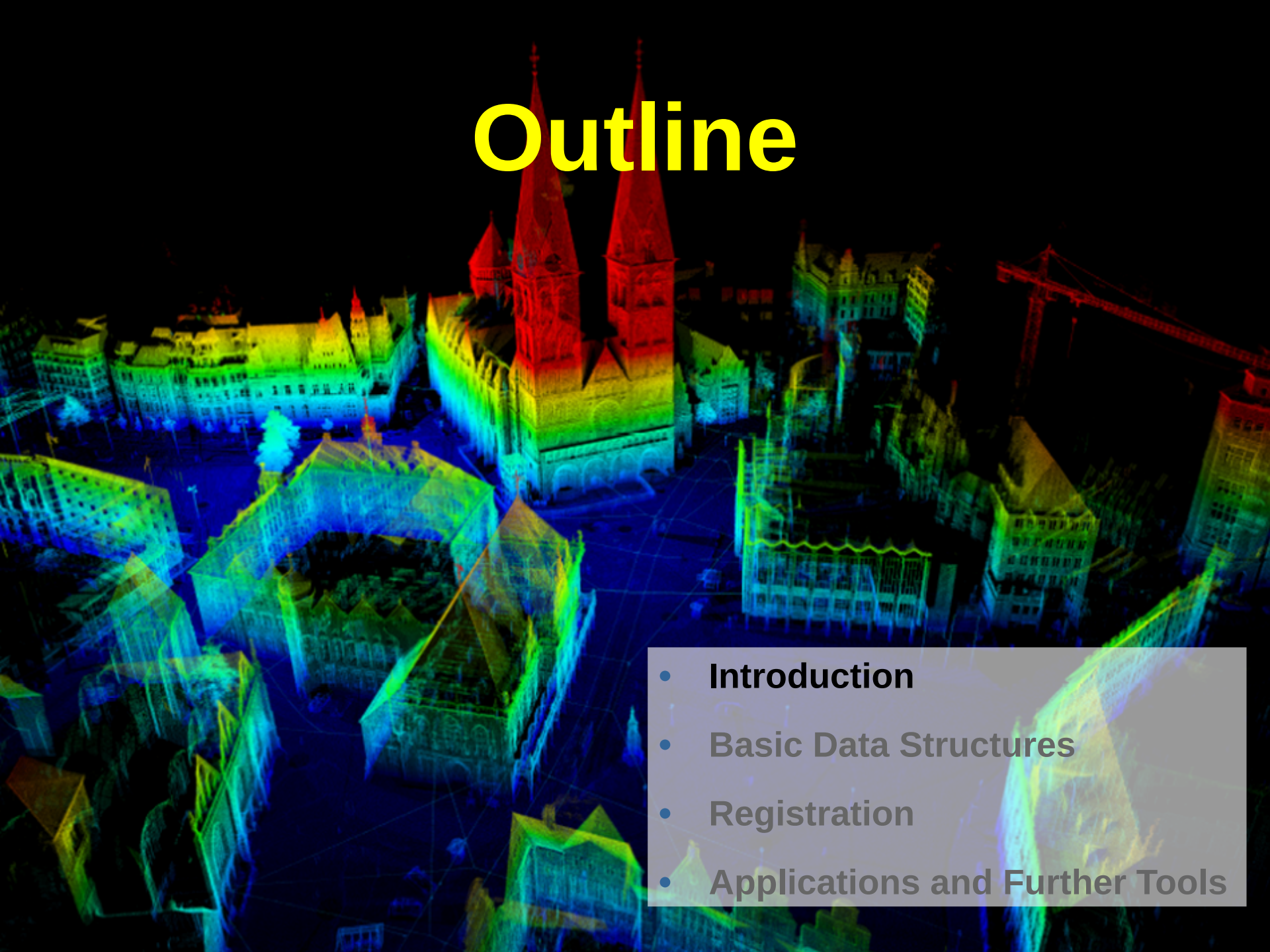


# Outline



- Introduction
- Basic Data Structures
- Registration
- Applications and Further Tools

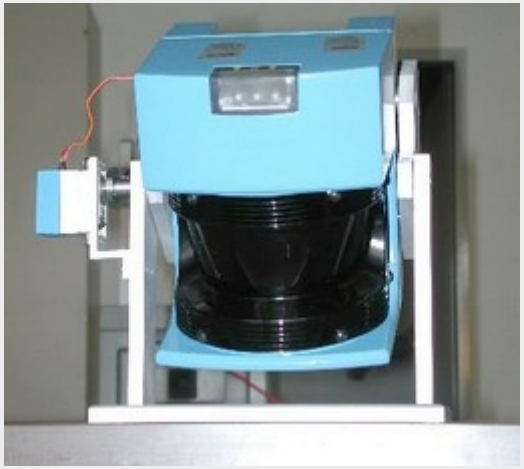
# Outline



- **Introduction**
- Basic Data Structures
- Registration
- Applications and Further Tools

# A Custom Made 3D Laser Scanner

- 3D laser scanner for mobile robots based on SICK LMS



- Based on a regular (e.g., SICK LMS-200) laser scanner
- Relatively cheap sensor
- Controlled pitch motion (120° v)
- Various resolutions and modi, e.g., reflectance measurement {181, 361, 721} [h] x {128, …, 500} [v] points
- Fast measurement, e.g., 3.4 sec (181x256 points)

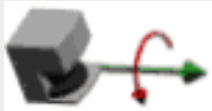
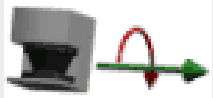
Mounted on mobile robots for 3D collision avoidance and building 3D maps.

(Video Crash)

(Video NoCrash)


RoboCup
HOSTED BY
RoboCup
Lenar
EVG
Sankt Augustin
Henrik
EVG
Sankt Augustin
Morik
EVG
Sankt Augustin

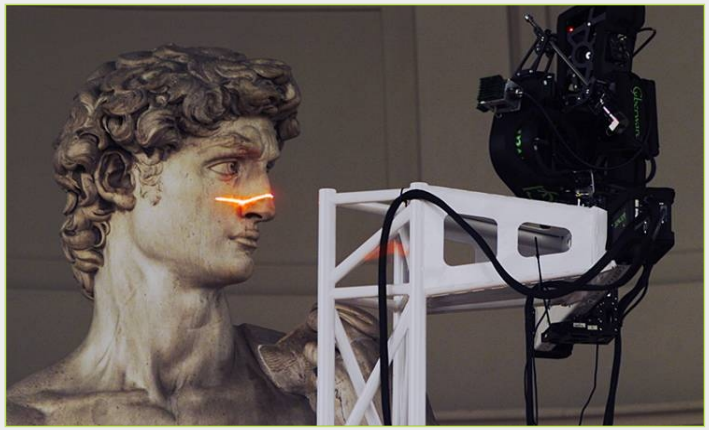
# 3D Scanning Principles

| Mode | Symbol | Cont. rotating | pivoting | Advantages |
|---|---|---|---|---|
| Yaw |  |  |  | + Complete 360° scans<br>+ Good point arrangements<br>- High point density at top |
| Yaw-Top |  |  |  | + Fast scanning (half rot.)<br>- High point density at top<br>- Ground not measured |
| Roll |  |  |  | + Fast scanning (half rot.)<br>+ High point density in front<br>- Unusual point arrangement |
| Pitch |  | |  | - High point density at the sides<br>- Small field of view angle<br>+ Good point arrangements<br>+ Easy to build |

http://www.rts.uni-hannover.de/index.php/%C3%9Cbersicht_der_m%C3%B6glichen_Scannerkonfigurationen

# Professional 3D Scanning

- Professional 3D scanners
  - Structured light (close range)



  - pulsed laser vs. time-of-flight (mid and long range)



Laserscanner
LEICA HDS3000

# 3DTK – Demonstration

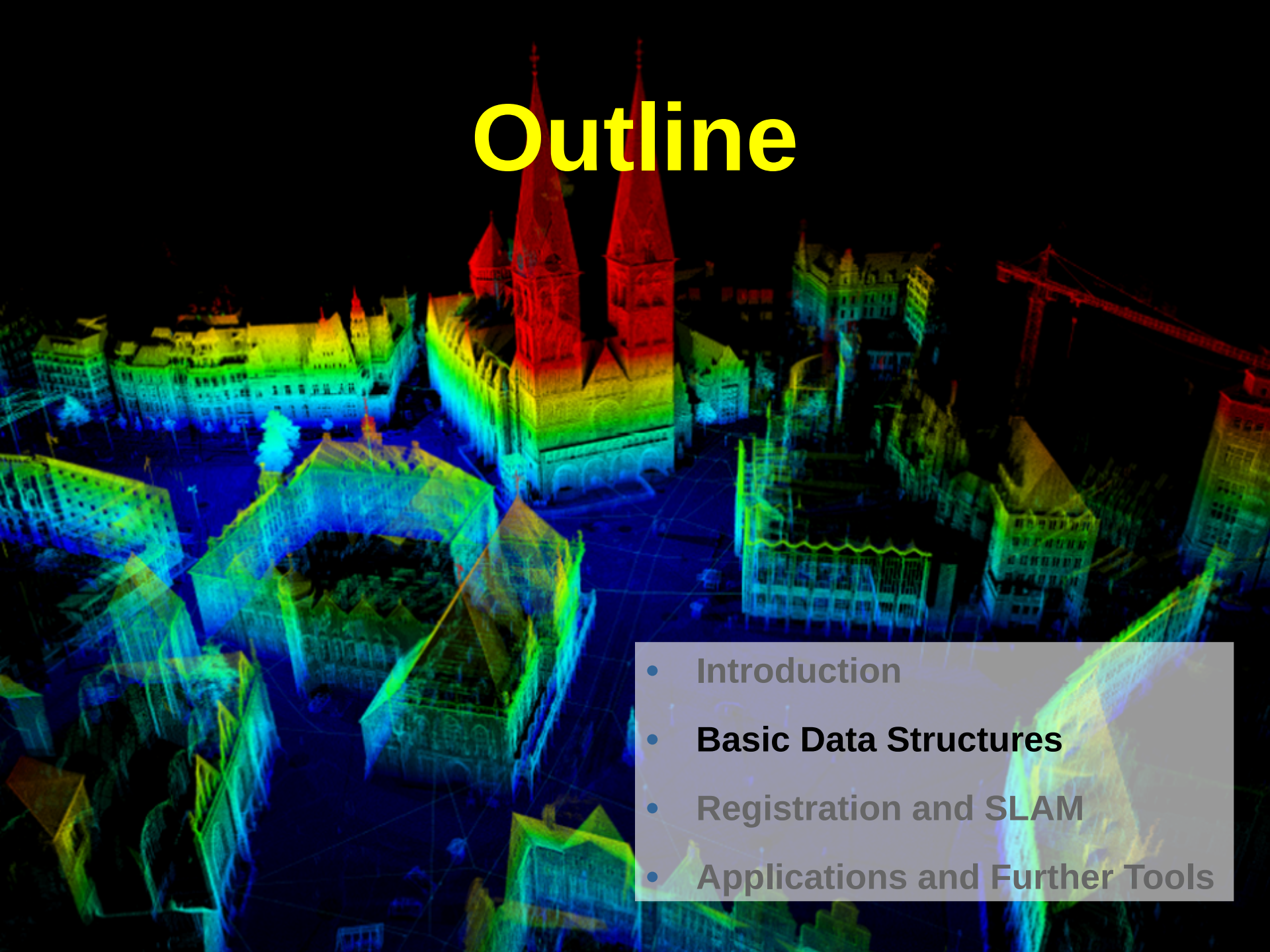
- Things to try
  - Viewing a single small 3D scan acquired in Schloß Dagstuhl (this data set comes with the svn checkout)

```
bin/show -s 0 -e 0 dat
```

  - Viewing a high resolution outdoor 3D scan

```
bin/show -s 0 -e 0 -f rxp --reflectance ~/data/wue_city/
```

# Outline



- Introduction
- **Basic Data Structures**
- Registration and SLAM
- Applications and Further Tools

# 3D Point Cloud as ...

… vector of *(x,y,z)*-values

- In 3DTK we have …
  - While reading a 3D Point Cloud

```
virtual void readScan(const char* dir_path,
                      const char* identifier,
                      PointFilter& filter,
                      std::vector<double>* xyz,
                      std::vector<unsigned char>* rgb,
                      std::vector<float>* reflectance,
                      std::vector<float>* amplitude,
                      std::vector<int>* type,
                      std::vector<float>* deviation);
```

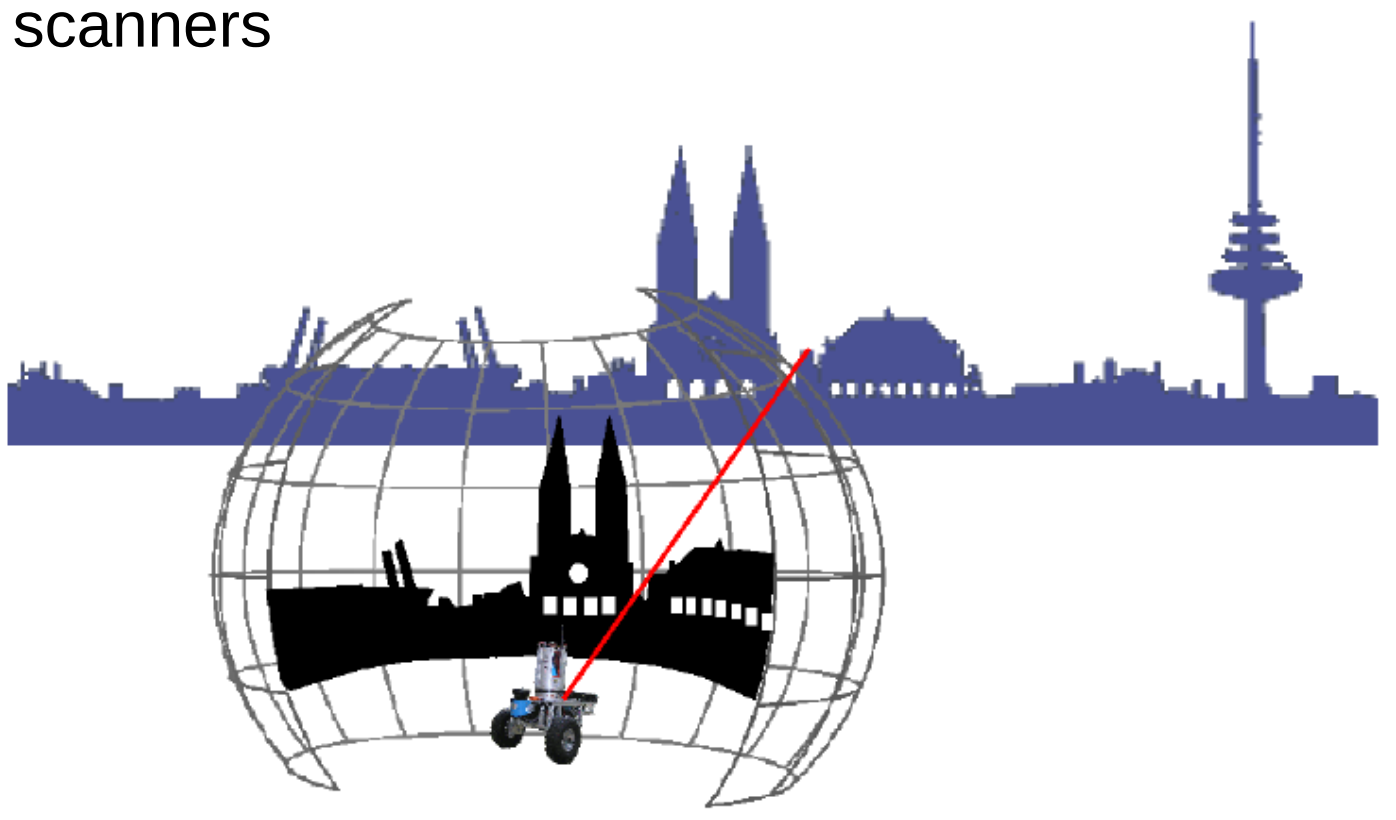
  - Called e.g., in the function `BasicScan::get()`
  - Finally the data ist stored in a STL-map

```
std::map<std::string, std::pair<unsigned char*, unsigned int>>
                      m_data;
```

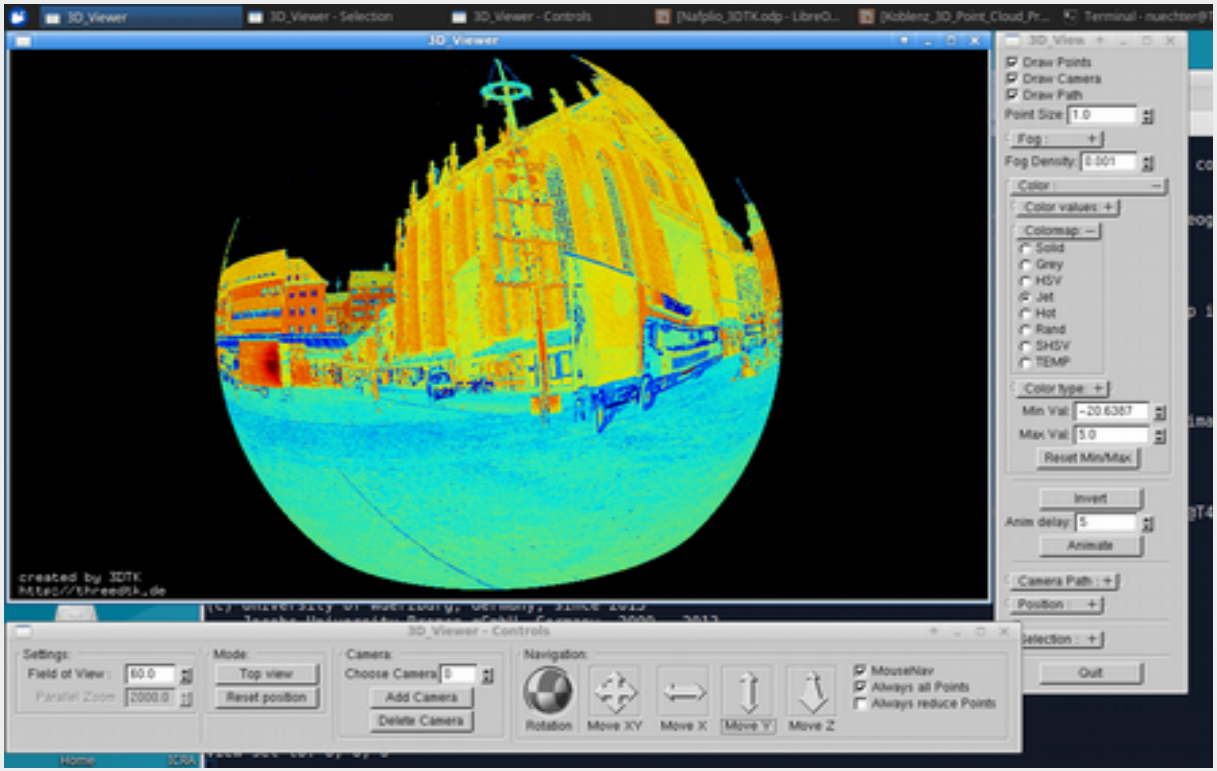
# 3D Point Cloud as ...

… as range / intensity image

- 2D array for kinect-like sensors
- Laser scanners



# 3DTK – Demonstration



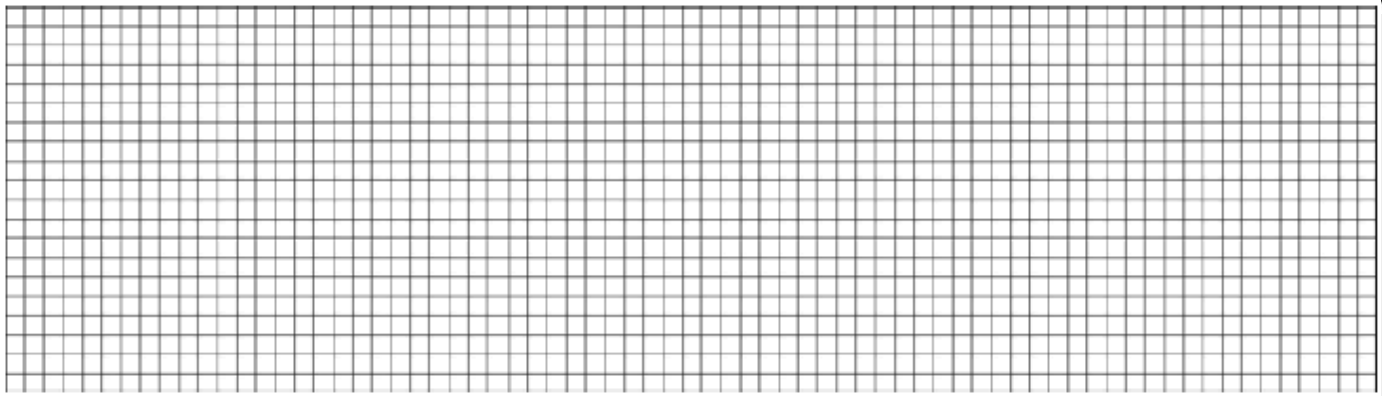

- Things to try
  - Viewing a high resolution outdoor 3D scan as sphere

```
bin/show -s 0 -e 0 -f rxp --reflectance
~/data/wue_city/ --sphere=100
```

# 3D Point Clouds as 2D arrays (1)

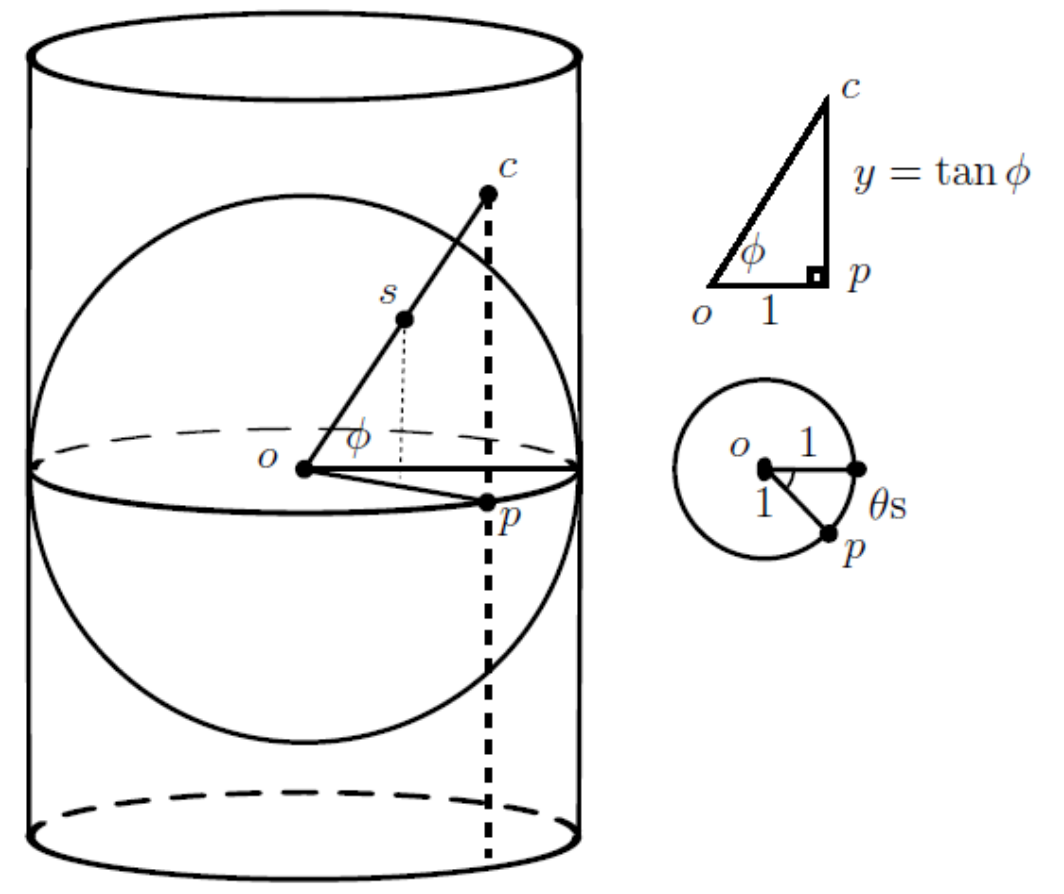
- Laser scanners
  - Equirectangular projection

$$x = \theta$$
$$y = \varphi$$

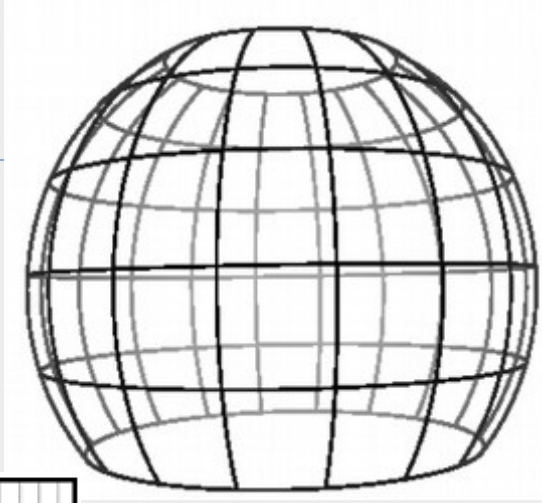


# 3D Point Clouds as 2D arrays (2)

- Laser scanners
  - Cylindrical projection

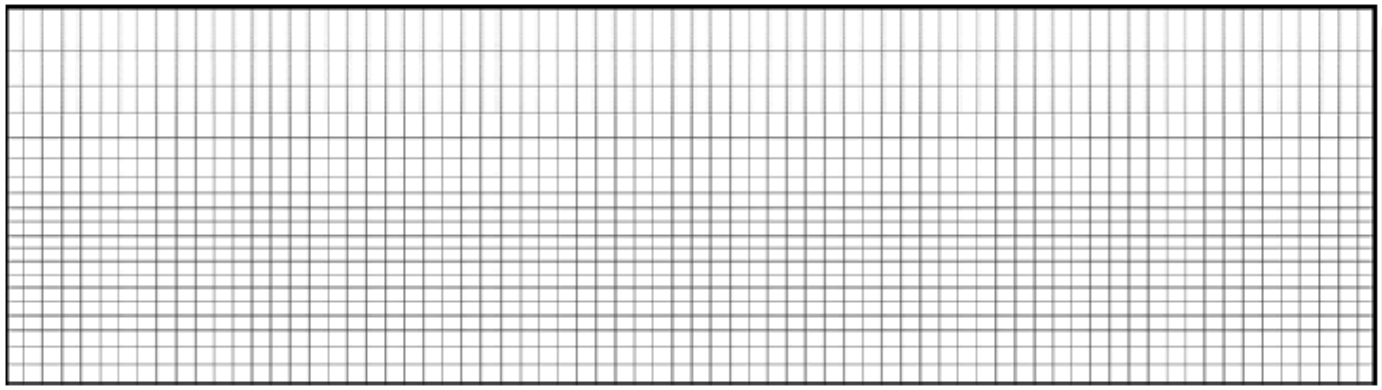


# 3D Point Clouds as 2D arrays (3)



- Laser scanners
  - Cylindrical projection

$$x = \theta$$
$$y = \tan \varphi$$

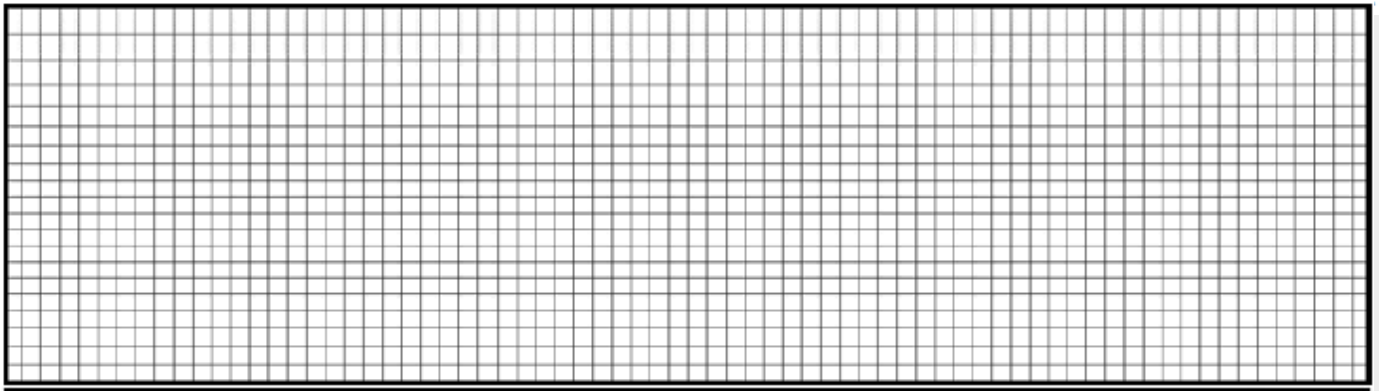


# 3D Point Clouds as 2D arrays (4)

- Laser scanners
  - Mercator projection
    - Cannot be “constructed”, only computational principle
    - The Mercator projection is an isogonic projection, i.e., angles are preserved

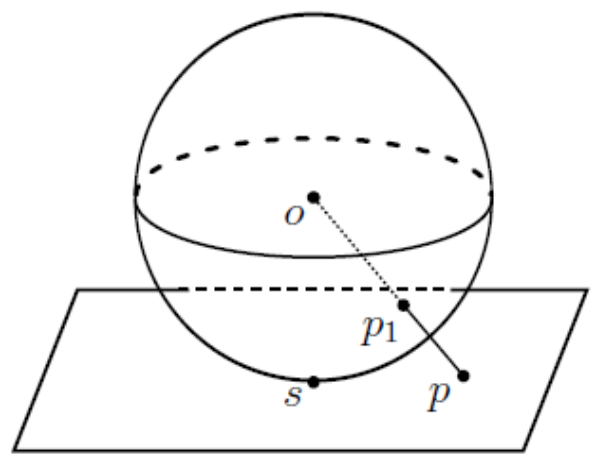
$$x = \theta$$

$$y = \ln\left(\tan\varphi + \frac{1}{\cos\varphi}\right)$$



# 3D Point Clouds as 2D arrays (5)

- Laser scanners
  - Rectilinear
  - also “gnomonic" or “tangentplane" projection.



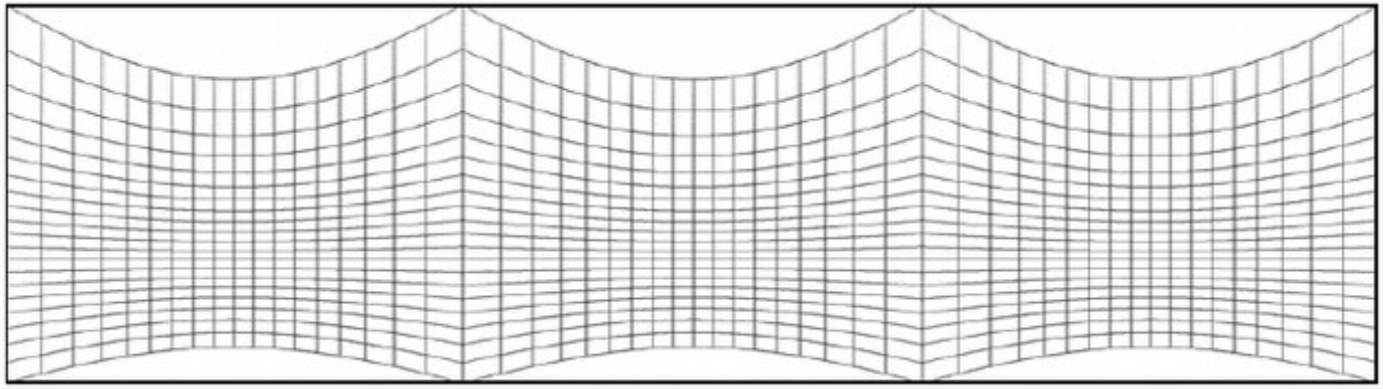

The primary advantage of the rectilinear projection is that it maps straight lines in 3D space to straight lines in the 2D image.

$$x = \frac{\cos\varphi \sin(\theta - \theta_0)}{\sin\varphi_1 \sin\varphi + \cos\varphi_1 \cos\varphi \cos(\theta - \theta_0)}$$

$$y = \frac{\cos\varphi_1 \sin\varphi - \sin\varphi_1 \cos\varphi \cos(\theta - \theta_0)}{\sin\varphi_1 \sin\varphi + \cos\varphi_1 \cos\varphi \cos(\theta - \theta_0)}.$$

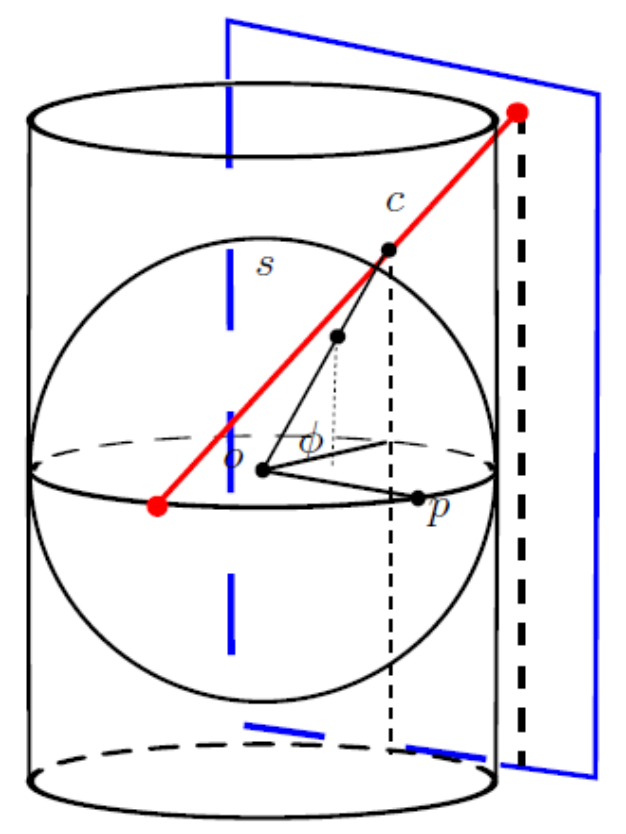
# 3D Point Clouds as 2D arrays (6)

- Laser scanners
  - Rectilinear
  - also “gnomonic" or “tangentplane" projection.



# 3D Point Clouds as 2D arrays (7)

- Laser scanners
  - Pannini, also called Panini or "Recti-Perspective" or "Vedutismo"
    - This projection can be imagined as the rectilin projection of a 3D cylindrical image.



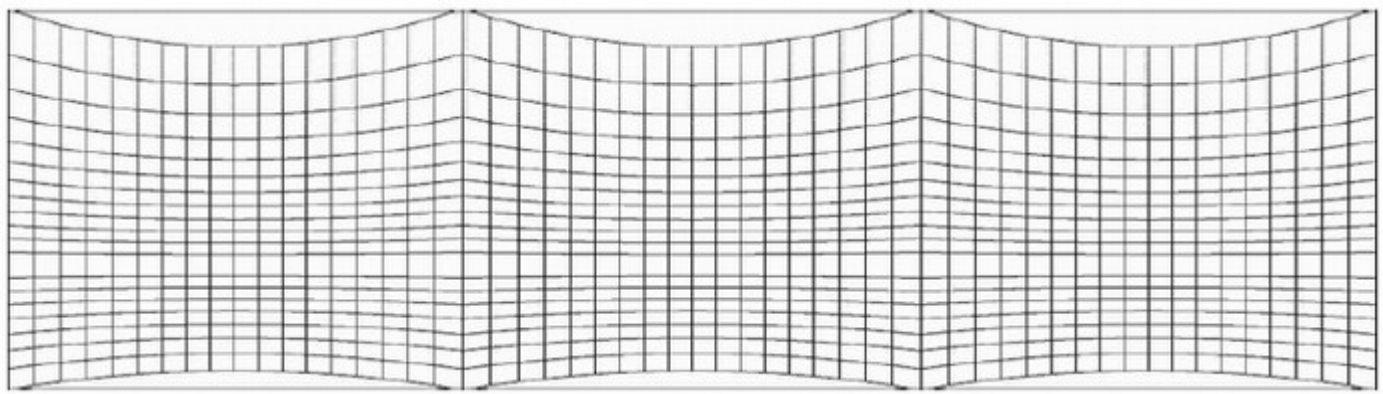

- This image is itself a projection of the sphere onto a tangent cylinder.
- The center of the rectilinear projection can be different and is on the view axis at a distance of d from the cylinder axis
- The recommended field of view for the Pannini projection is less than 150° in both vertical and horizontal directions.

# 3D Point Clouds as 2D arrays (8)

- Laser scanners
  - Pannini projection

$$x = \frac{(d+1)\sin(\theta-\theta_0)}{d+\sin\varphi_1\tan\varphi+\cos\varphi_1\cos(\theta-\theta_0)}$$

$$y = \frac{(d+1)\tan\varphi\left(\cos\varphi_1-\sin\varphi_1\left(\frac{1}{\tan\varphi}\right)\cos(\theta-\theta_0)\right)}{d+\sin\varphi_1\tan\varphi+\cos\varphi_1\cos(\theta-\theta_0)}$$



# 3D Point Clouds as 2D arrays (9)

- Laser scanners
  - Stereographic projection
    - It can be imagined by placing a paper tangent to a sphere and by illuminating it from the opposite pole.

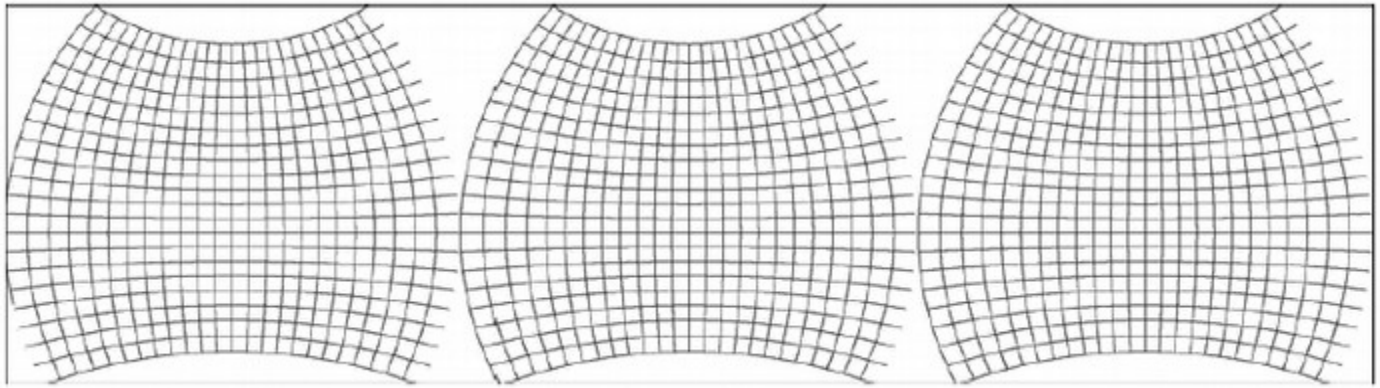
$$x = \frac{2R\cos\varphi\sin(\theta-\theta_0)}{1+\sin\varphi_1\sin\varphi+\cos\varphi_1\cos\varphi\cos(\theta-\theta_0)}$$

$$y = \frac{2R(\cos\varphi_1\sin\varphi-\sin\varphi_1\cos\varphi\cos(\theta-\theta_0))}{1+\sin\varphi_1\sin\varphi+\cos\varphi_1\cos\varphi\cos(\theta-\theta_0)}$$

    - R = 1 generates exactly the same equations as the Pannini projection and high values for R introduce more distortion.

# 3D Point Clouds as 2D arrays (10)

- Laser scanners
  - Stereographic projection

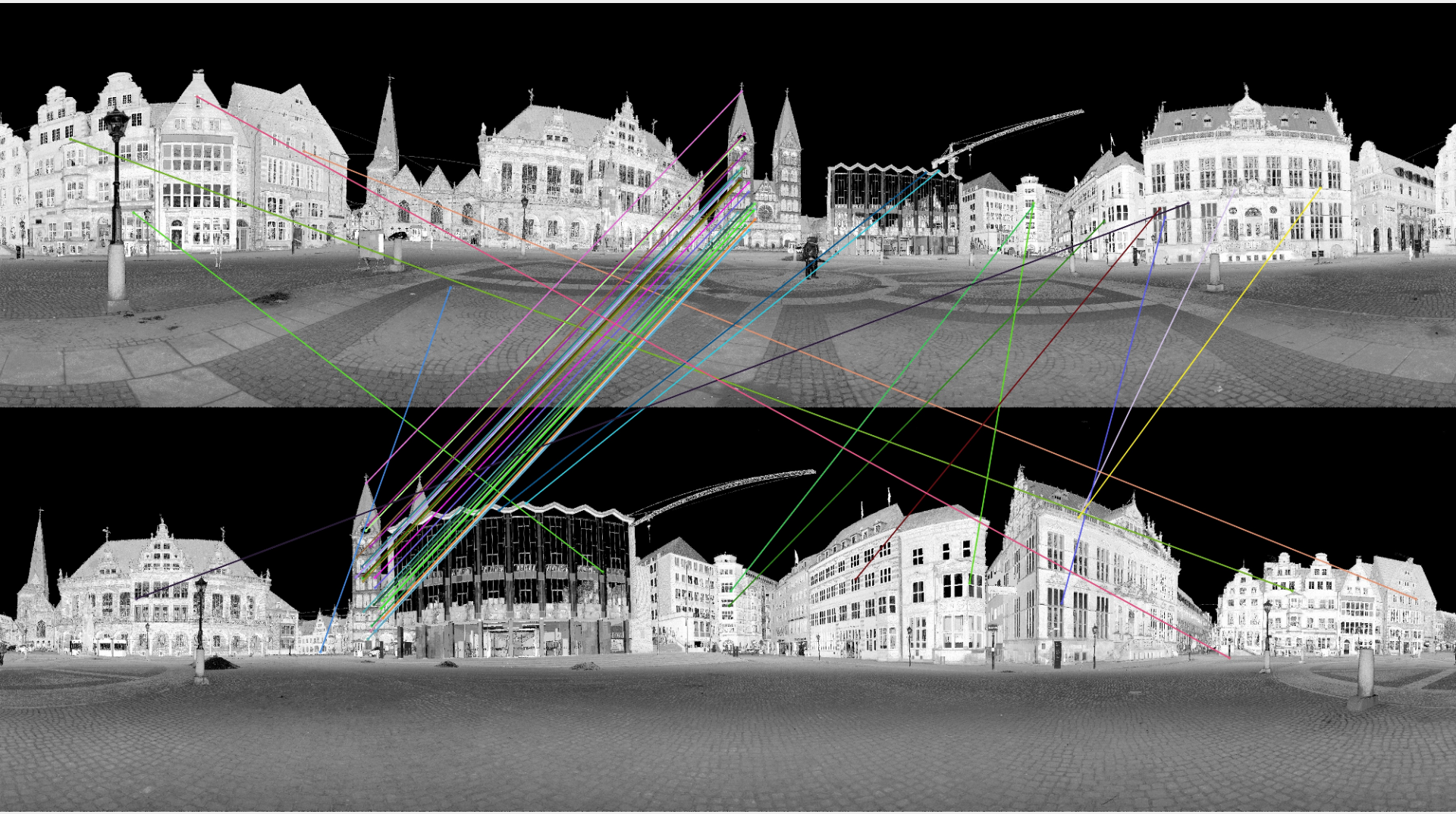


# 3DTK – Demonstration

- Things to try
  - Viewing a high resolution outdoor 3D scan as sphere

```
bin/scan_to_panorama -s 0 -e 0 -f rxp -t RIEGL -R -F PNG
-p RECTILINEAR -N 3 ~/data/wue_city/
```

# Application: Feature-Based Registration



# Oc-trees (1)

- Every node has 8 children

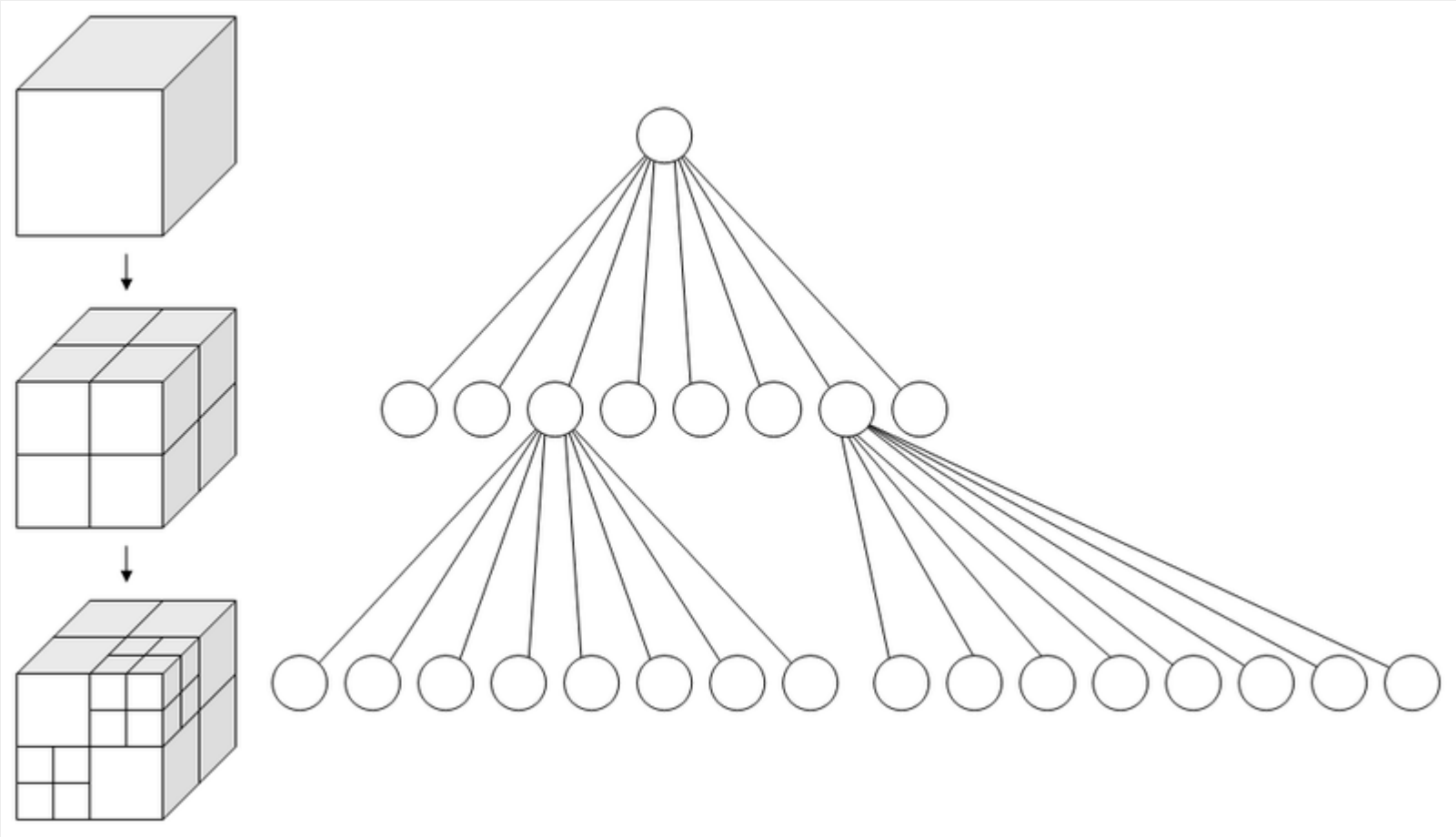


Image: Wikipedia

# Oc-trees (2)

- Empty nodes / voxels are pruned
- Every node has 8 children

```
struct OcTree {
  float center[3];
  float size[3];
  OcTree *child[8];
  int nr_points;
  float **points;
};
```

- Definition of an oc-tree with redundant information and eight pointers to child nodes. The size of this node is 100 Bytes.

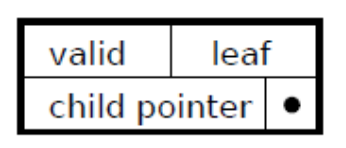
# Oc-trees (3)

- Statistics of the Bremen City data set

| | 100 byte | |
|---|---|---|
| Leaf size (cm) | Mem. Size | Constr. time (ms) |
| 8560 | 1.9 kB | 557.1 |
| 4280 | 4.8 kB | 694.1 |
| 2140 | 13.9 kB | 939.2 |
| 1070 | 45.3 kB | 1165.7 |
| 535 | 130.1 kB | 1279.3 |
| 267 | 405.2 kB | 1529.4 |
| 133 | 1.25 MB | 1656.5 |
| 66.8 | 3.85 MB | 1895.5 |
| 33.4 | 11.91 MB | 2002.3 |
| 16.7 | 36.65 MB | 2146.5 |
| 8.35 | 109.53 MB | 2290.3 |
| 4.17 | 301.28 MB | 2471.9 |
| 2.08 | 742.98 MB | 2576.0 |
| 1.04 | 1.568 GB | 2899.6 |
| 0.52 | 2.643 GB | 3199.9 |

- Exponential growth

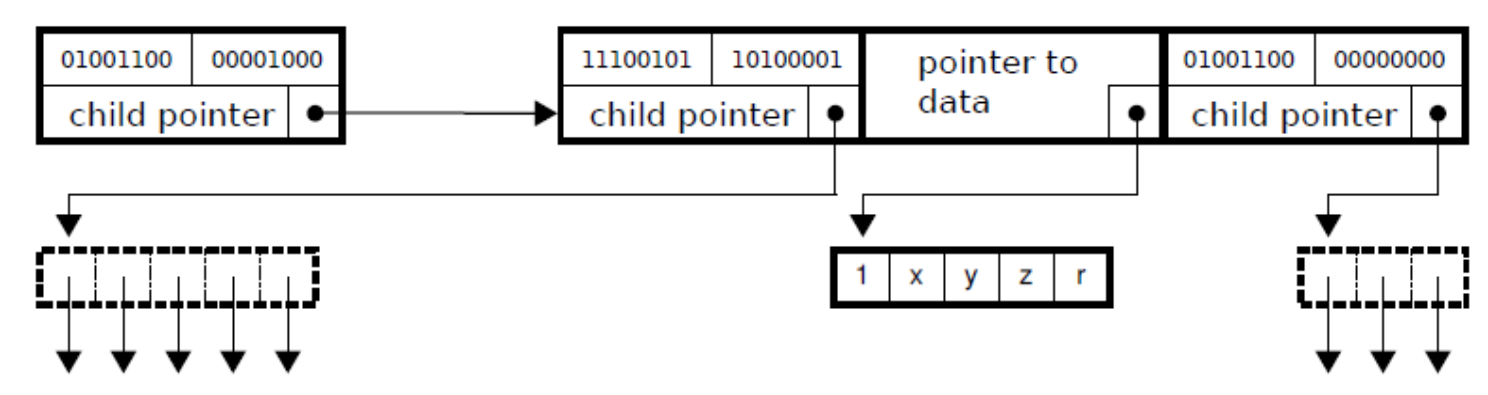
# Efficient Oc-Trees (1)




- 3DTK proposes encodings of an octree node optimized for memory efficiency.
- The child pointer as the relative pointer is the largest part of an octree node, but varies in size to accommodate different systems. In our implementation for 64 bit systems, it is 48 bit. valid and leaf are 8 bit large.

- The proposed encoding with separate bit fields for valid and leaf. An entire node is thus contained in only 8 bytes of memory.

# Efficient Oc-Trees (2)




- An example of a simple oc-tree as it is stored in 3DTK.

- The node in the upper left has three valid children, one of which is a leaf. Therefore, the child pointer only points to 3 nodes stored consecutively in memory. The leaf node in this example is a simple pointer to an array which stores both the number of points and the points with all their attributes.

# Efficient Oc-Trees (3)

| | 8 byte | | | | 100 byte | |
|---|---|---|---|---|---|---|
| Leaf size (cm) | # Nodes | # Leaves | Mem. Size | Constr. time (ms) | Mem. Size | Constr. time (ms) |
| 8560 | 6 | 12 | 192 B | 1019.2 | 1.9 kB | 557.1 |
| 4280 | 18 | 28 | 480 B | 1321.2 | 4.8 kB | 694.1 |
| 2140 | 46 | 86 | 1.4 kB | 1535.7 | 13.9 kB | 939.2 |
| 1070 | 129 | 296 | 4.5 kB | 1706.1 | 45.3 kB | 1165.7 |
| 535 | 408 | 800 | 12.8 kB | 1909.7 | 130.1 kB | 1279.3 |
| 267 | 1166 | 2595 | 40.4 kB | 2081.8 | 405.2 kB | 1529.4 |
| 133 | 3616 | 7993 | 124.8 kB | 2299.7 | 1.25 MB | 1656.5 |
| 66.8 | 11130 | 24587 | 384.1 kB | 2473.3 | 3.85 MB | 1895.5 |
| 33.4 | 33965 | 75999 | 1.18 MB | 2687.5 | 11.91 MB | 2002.3 |
| 16.7 | 102728 | 233413 | 3.62 MB | 2873.3 | 36.65 MB | 2146.5 |
| 8.35 | 302573 | 687529 | 10.67 MB | 3134.9 | 109.53 MB | 2290.3 |
| 4.17 | 814040 | 1808993 | 28.22 MB | 3432.0 | 301.28 MB | 2471.9 |
| 2.08 | 1927234 | 4166979 | 65.42 MB | 3721.0 | 742.98 MB | 2576.0 |
| 1.04 | 4031140 | 7783889 | 125.65 MB | 3901.8 | 1.568 GB | 2899.6 |
| 0.52 | 5592151 | 10142923 | 166.45 MB | 4077.7 | 2.643 GB | 3199.9 |

# Efficient Oc-Trees (4)

- Comparison with other oc-trees

| Library | Node size | Remarks |
|---|---|---|
| xgrt | $144 + x$ | |
| octomap | $72 + x$ | |
| PCL | $64 + x$ | |
| PCL low memory base | $25 + x$ | (since ver 1.1.1. Sept. 2011) |
| CloudCompare | 16 | size of leaf node |
| 3DTK | 8 | |

Note: Serialized pointer-free encodings, employing the Morton order to store only the leaf level. The Morton order or Z-order is an ordering of –in this case– 3D data. Traversing the Octree is not possible in the classical sense. Instead binary search algorithms have to be applied when looking for a certain voxel.

# Using an Oc-tree for 3D Point Cloud Reduction

- Generate an oc-tree until you reached the desired voxel size
- Select the center point of each voxel for the reduced point cloud.

  Or

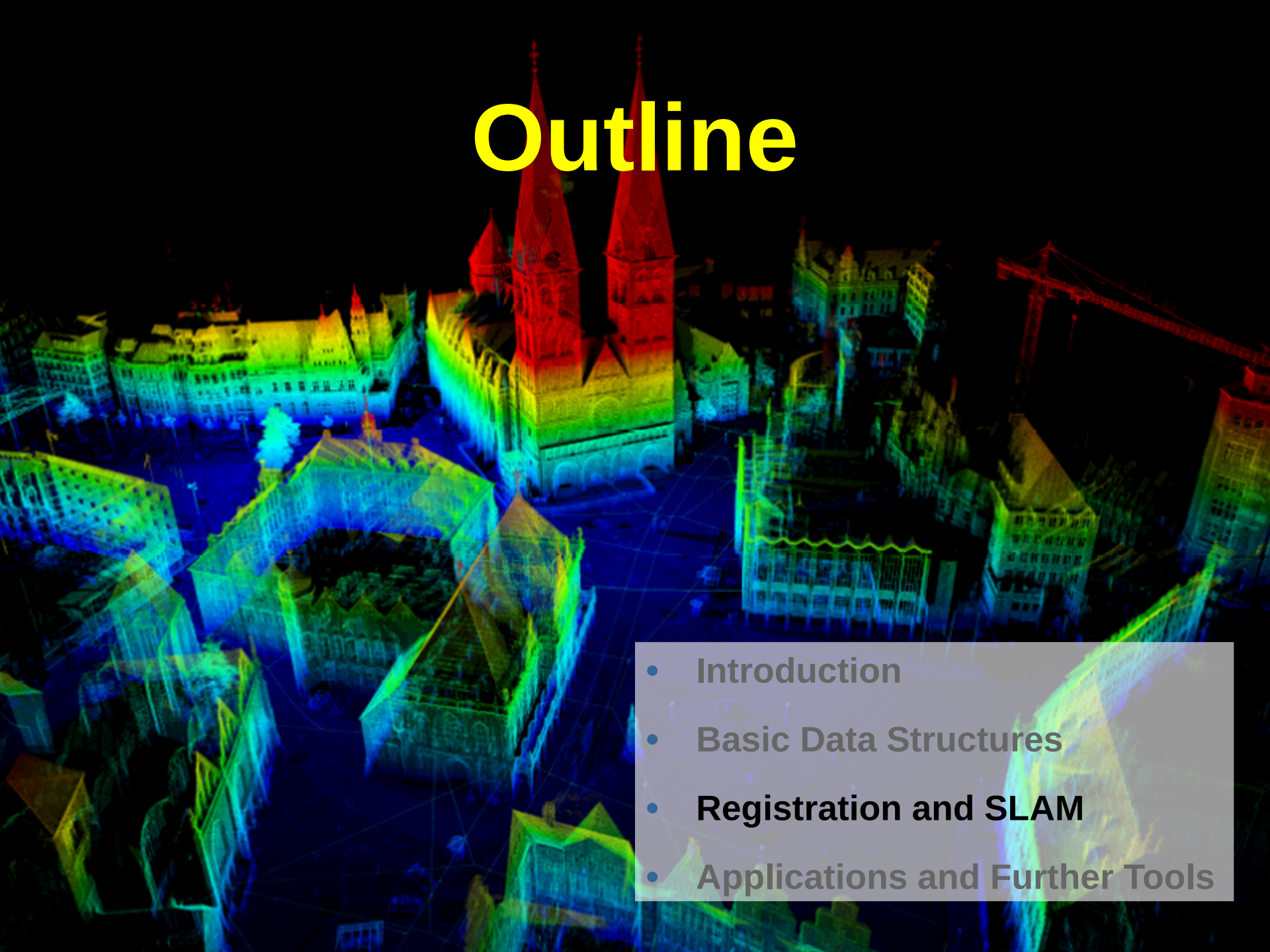
- Select n point randomly from each voxel.

- This is implemented in 3DTK (program `scan_red`)

```
bin/scan_red -s 0 -e 0 -f rxp --reduction OCTREE --voxel 10
--octree 1 --reflectance ~/data/wue_city/

bin/show -s 0 -e 0 -f uosr ~/data/wue_city/reduced
```

# Outline

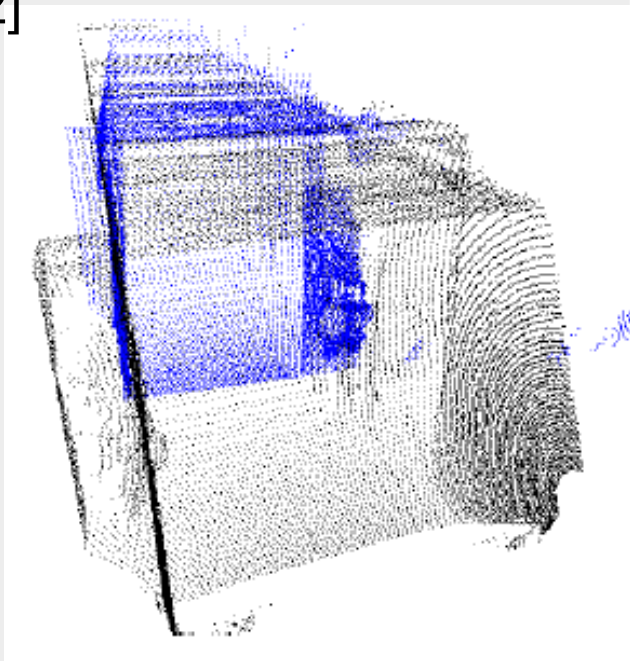


- Introduction
- Basic Data Structures
- **Registration and SLAM**
- Applications and Further Tools

# The ICP Algorithm (1)

**Scan registration** Put two independent scans into one frame of reference

**Iterative Closest Point** algorithm [Besl/McKay 1992]



For prior point set $M$ ("model set") and data set $D$

1. Select point correspondences
2. Minimize for rotation $\mathbf{R}$, translation $\mathbf{t}$

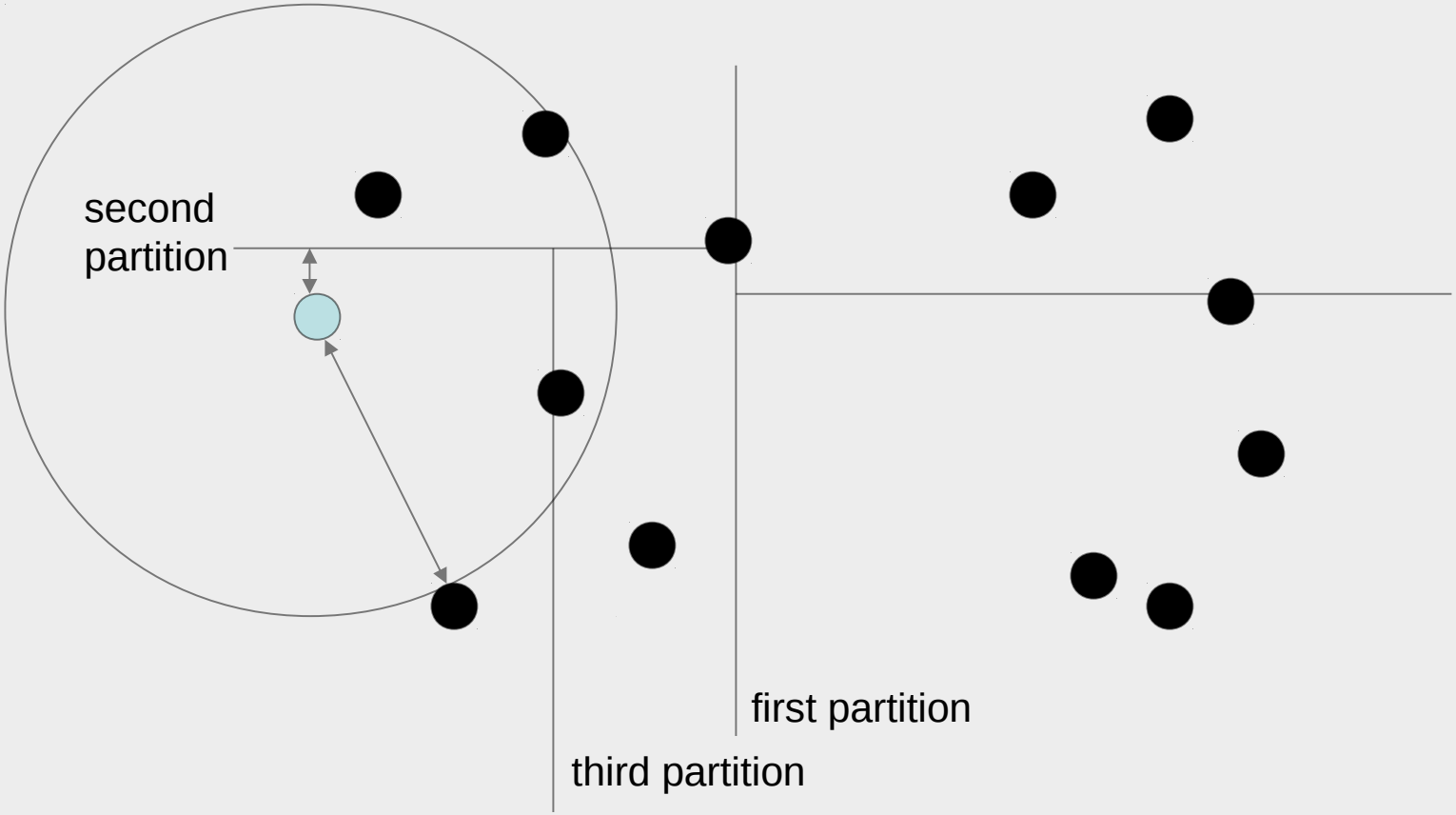
$$E(\mathbf{R}, \mathbf{t}) = \frac{1}{N} \sum_{i=1}^{N} ||\mathbf{m}_i - (\mathbf{R}\mathbf{d}_i + \mathbf{t})||^2$$

3. Iterate **1.** and **2.**

SVD-based calculation of rotation

- works in 3 translation plus 3 rotation dimensions
  $\Rightarrow$ 6D SLAM with closed loop detection and global relaxation.

# K-d Tree based NNS (1)

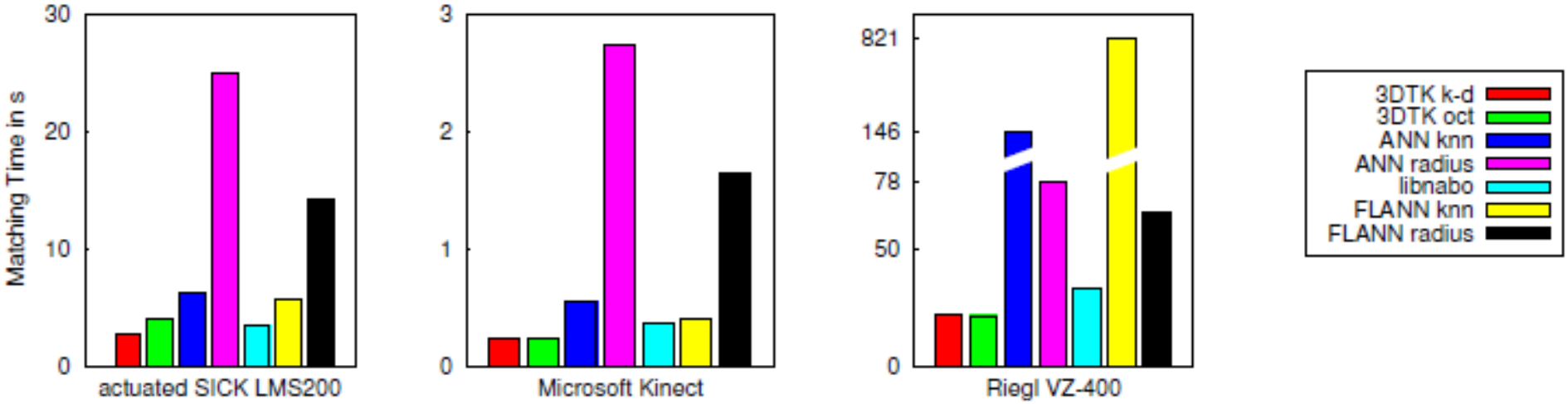



- One has to search all buckets according to the ball-within-bounds-test. ⇨ Backtracking

# NNS Search – the Critical Issue

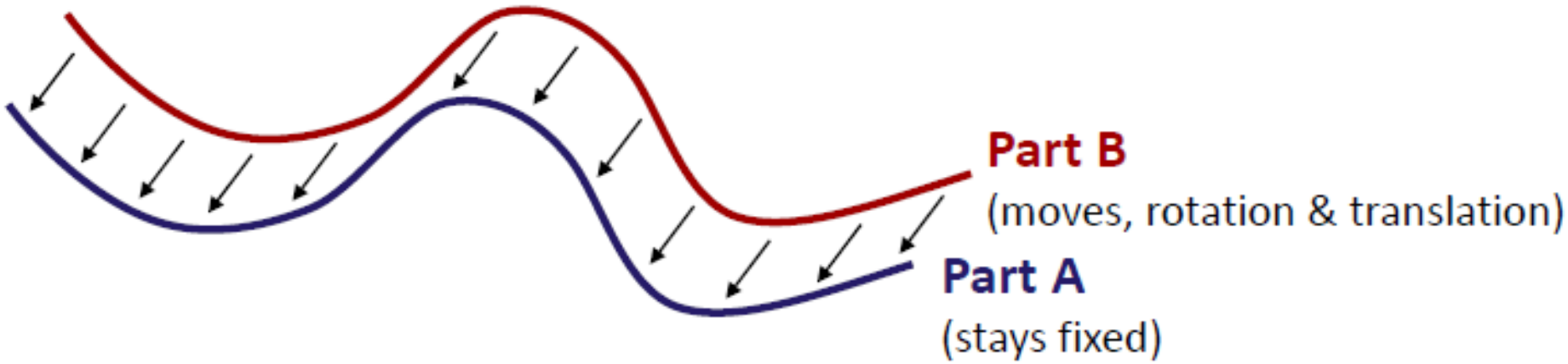
Properties for all tested NNS libraries.

| Library | revision | Data structure | $k$-NN search | fixed radius | ranged search | optimized for |
|---|---|---|---|---|---|---|
| 3DTK [2] | rev. 470 | k-d tree | × | × | ✓ | shape registration |
| 3DTK | rev. 470 | octree | × | × | ✓ | shape registration & efficient storage |
| ANN [3] | Ver. 1.1.1 | k-d tree | ✓ | ✓ | × | |
| CGAL [4] | Ver. 3.5.1-1 | k-d tree | × | ✓ | × | |
| FLANN [5] | bcf3a56e5fed2d4dc3a340725fa341fa36ef79a4 | k-d tree | ✓ | ✓ | × | high dimensions |
| libnabo [6] | Ver. 1.0.0 | k-d tree | ✓ | × | ✓ | |
| SpatialIndex [7] | Ver. 1.4.0-1.1 | R-tree | ✓ | × | × | |
| STANN [8] | Ver. 0.71 beta | SFC | ✓ | × | × | multithreading |



# Registering Surfaces (1)

- Given

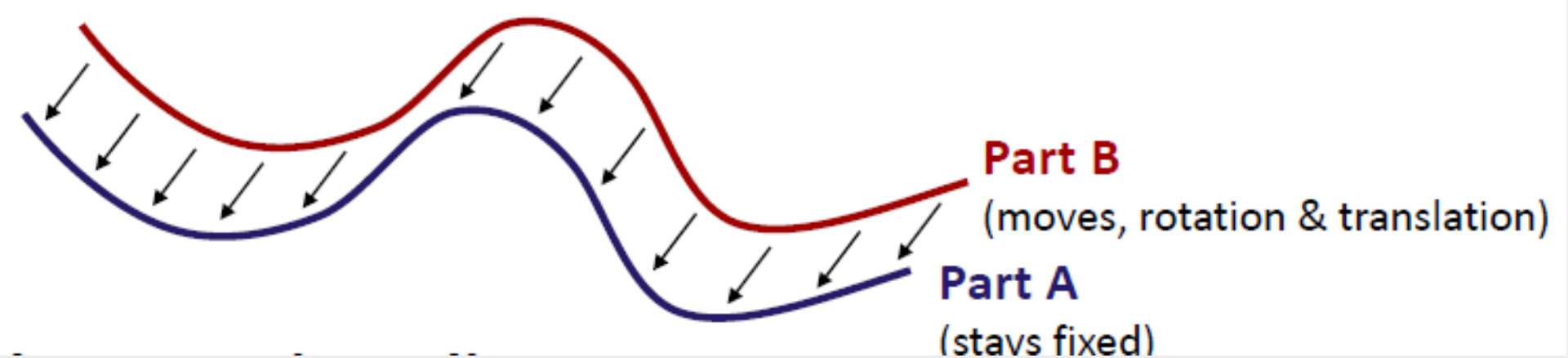



The main idea:

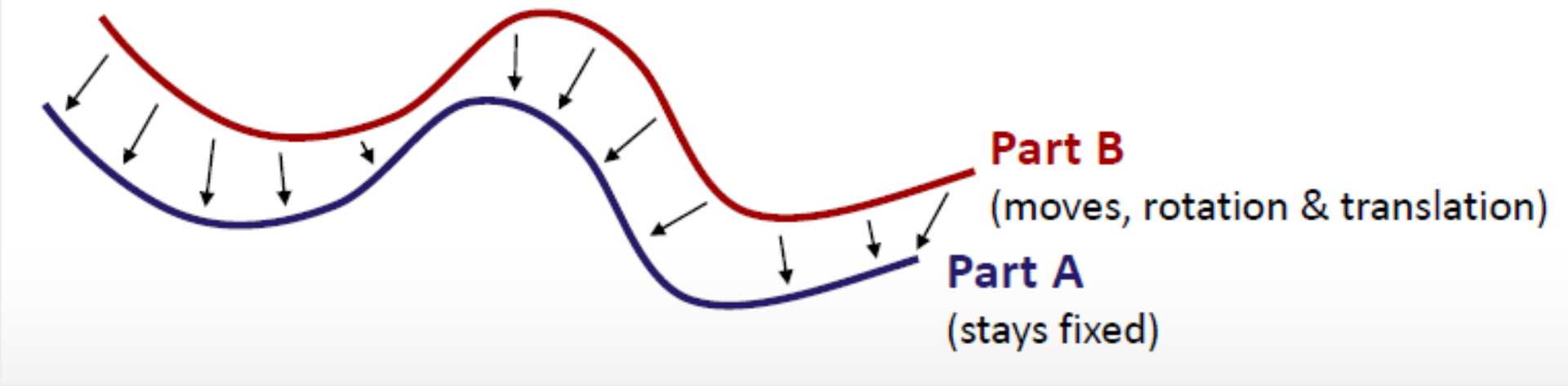
- Pairwise matching technique
- We want to minimize the distance between the two parts
- We set up a variational problem
- Minimize distance “energy” by rigid motion of one part

# Registering Surfaces (3)

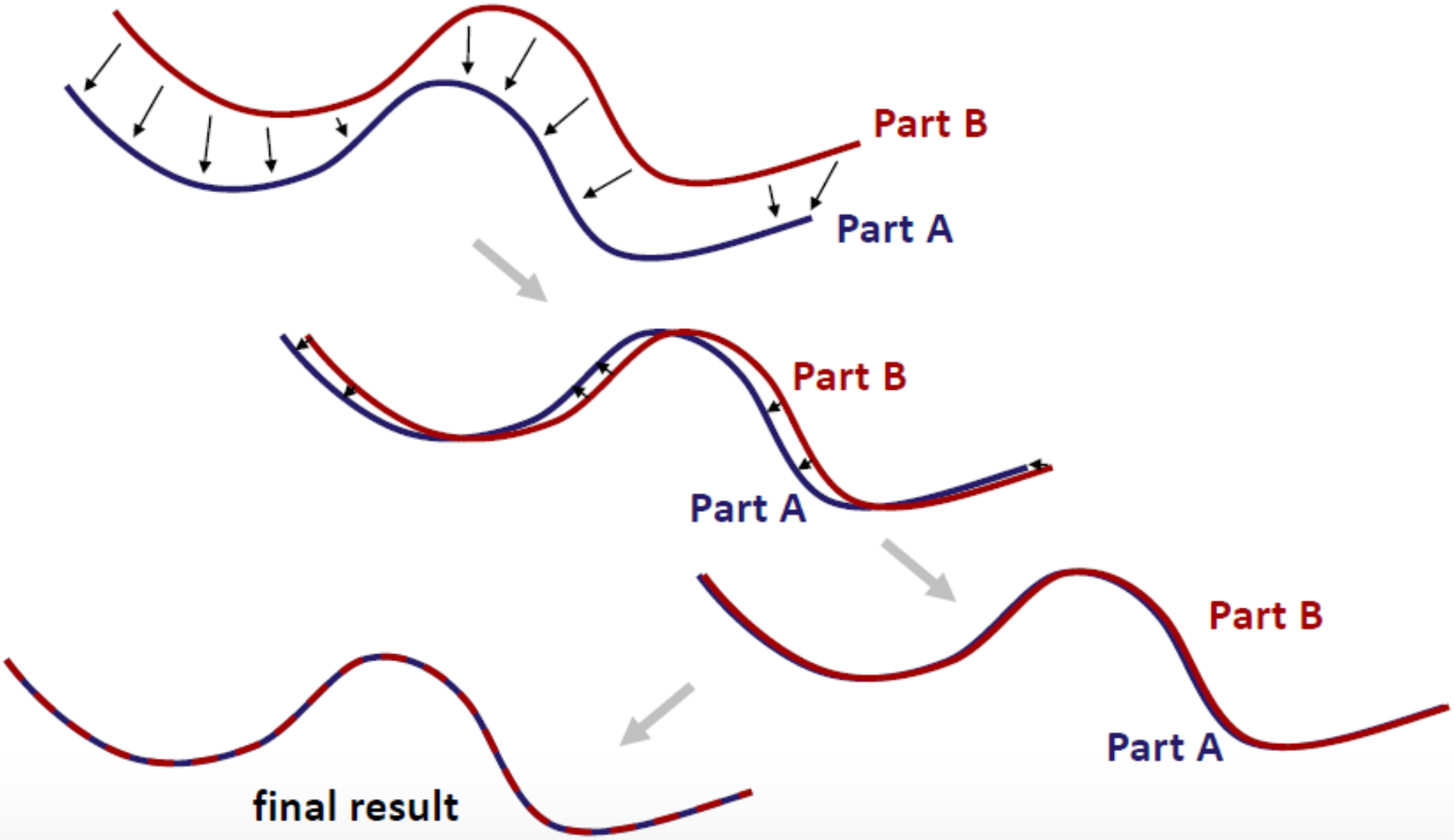
- Distances




- Closest Point Distances



# Registering Surfaces (4) – ICP iterations



# Processing Large Data Sets (1)

```
bin/slam6D -s 0 -e 1 -f rxp ~/data/wue_city/ -r 25
-i 2500 --anim=50 -d 75

bin/show -s 0 -e 1 -f rxp --reflectance
~/data/wue_city/
```

- But large data sets with many 3D scans???
- We see: small matching errors accumulate

```
bin/slam6D -s 1 -e 65 -r 10 -i 100 -d 75
--epsICP=0.00001 ~/dat/hannover1/
```

# 6D SLAM – Global Relaxation (1)

- In SLAM loop closing is the key to build consistent maps
- Notice: Consistent vs. correct or accurate

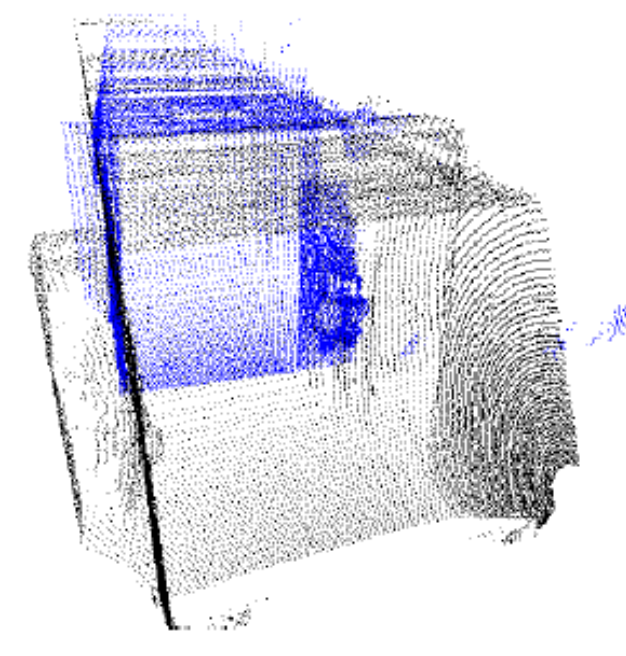
- GraphSLAM
  - Graph Estimation
  - Graph Optimization

- Graph Estimation
  - Simple strategy: Connect poses with graph edges that are close enough
  - Simple strategy: Connect poses, they have enough point pairs (closest points)

# The ICP Algorithm

**Scan registration** Put two independent scans into one frame of reference

**Iterative Closest Point** algorithm [Besl/McKay 1992]



For prior point set *M* (“model set”) and data set *D*

1. Select point correspondences *wi,j* in {0,1}
2. Minimize for rotation **R**, translation **t**

$$E(\mathbf{R},\mathbf{t}) = \frac{1}{N}\sum_{i=1}^{N} ||\mathbf{m}_i - (\mathbf{R}\mathbf{d}_i + \mathbf{t})||^2$$

3. Iterate **1.** and **2.**

Four closed form solution for the minimization

For globally consistent scan matching use the following error function

**Global ICP Algorithm**

$$E = \sum_{j\to k}\sum_{i} |\mathbf{R}_j\mathbf{m}_i + \mathbf{t}_j - (\mathbf{R}_k\mathbf{d}_i + \mathbf{t}_k)|^2$$

Minimize for all rotations $\mathbf{R}$ and translations $\mathbf{t}$ at the same time

# Comparisons of the Parametrizations

**Global ICP**

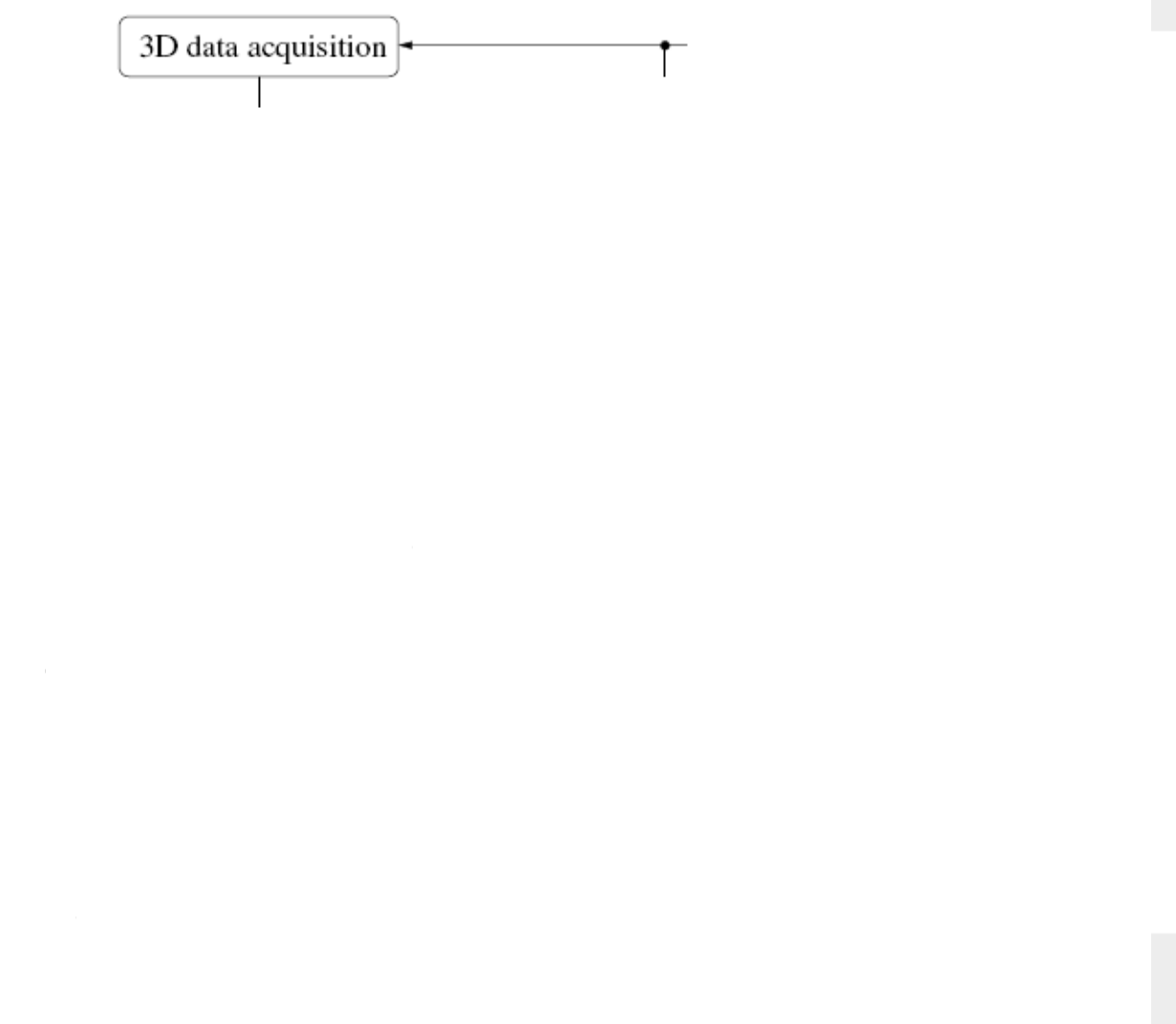
- Gaussian noise in the „3D Point Cloud“ space
- Locally optimal
- ICP-like iterations using new point correspondences

- Riegl Laser Measurement GmbH

(video)

(video) (video) (video)

# Closed Loop Detection and Global Relaxation



# Processing Large Data Sets (2)

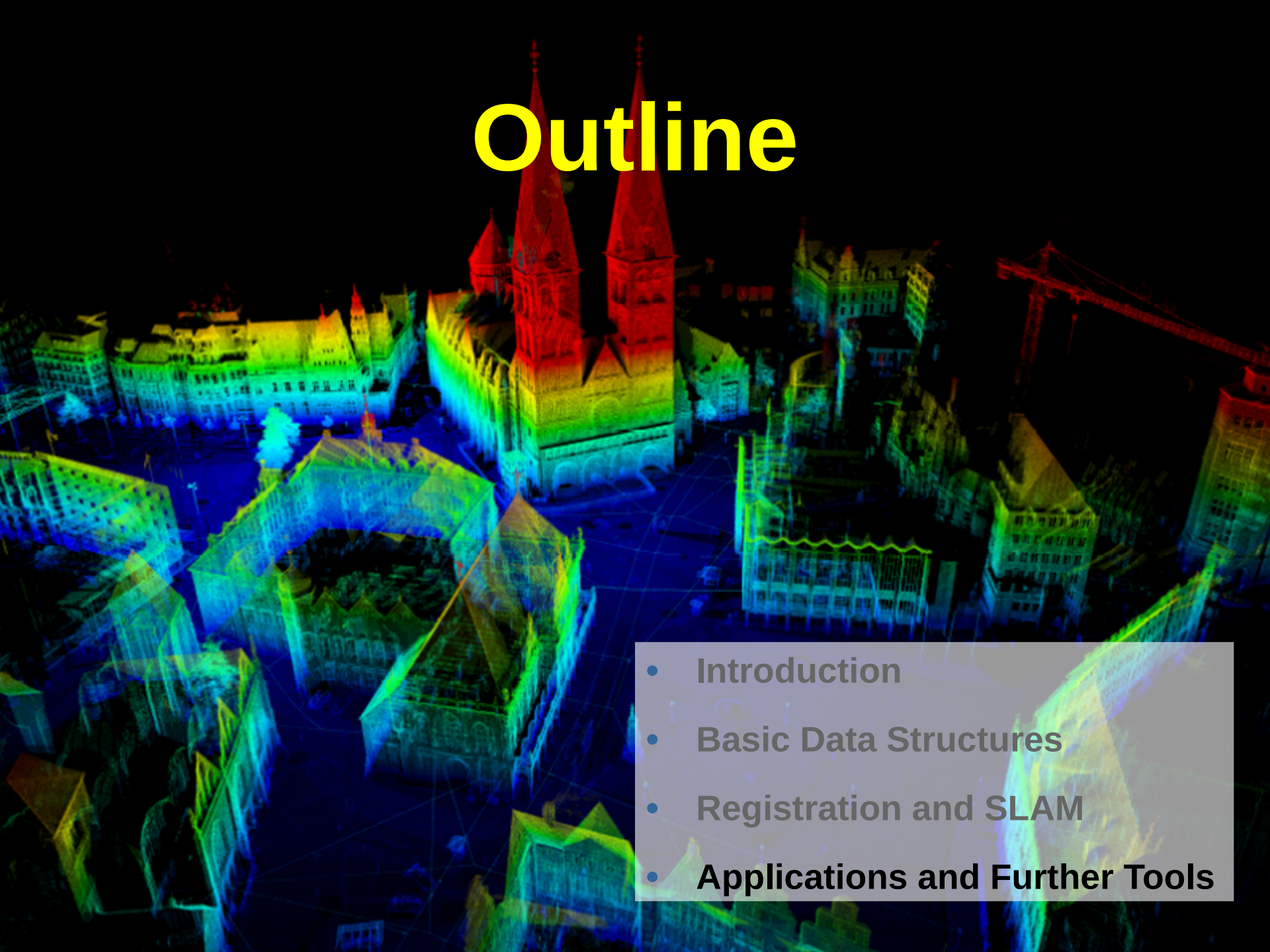
```
bin/slam6D -s 1 -e 65 -r 10 -i 100 -d 75
--epsICP=0.00001 ~/dat/hannover1/
```

We see: small matching errors accumulate

```
bin/slam6D -s 1 -e 65 -r 10 -i 100 -d 75
--epsICP=0.00001 -D 250 -I 50 --cldist=750
-L 0 -G 1 ~/dat/hannover1

bin/show -s 1 -e 65 ~/dat/hannover1
```

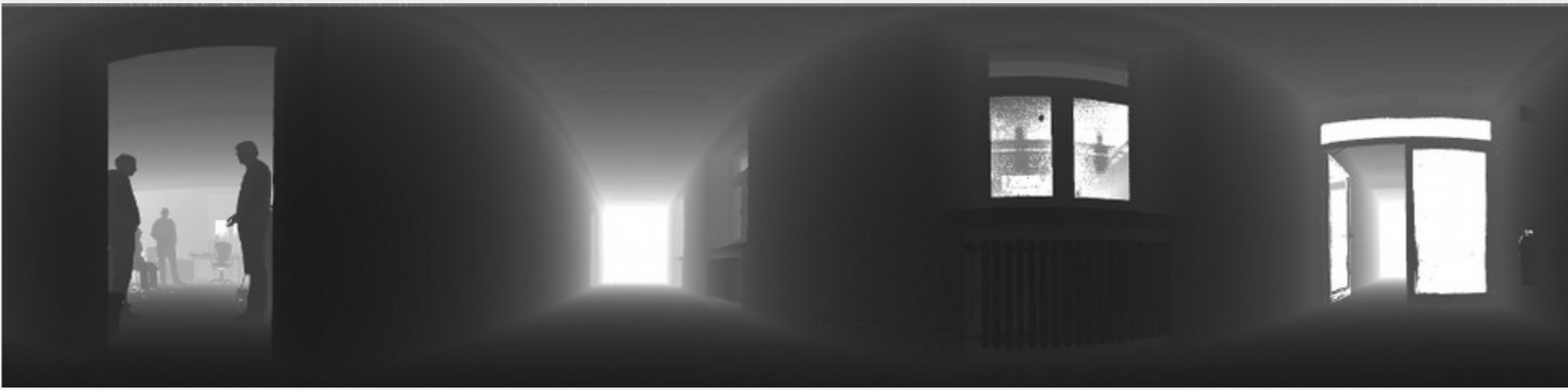
# Outline

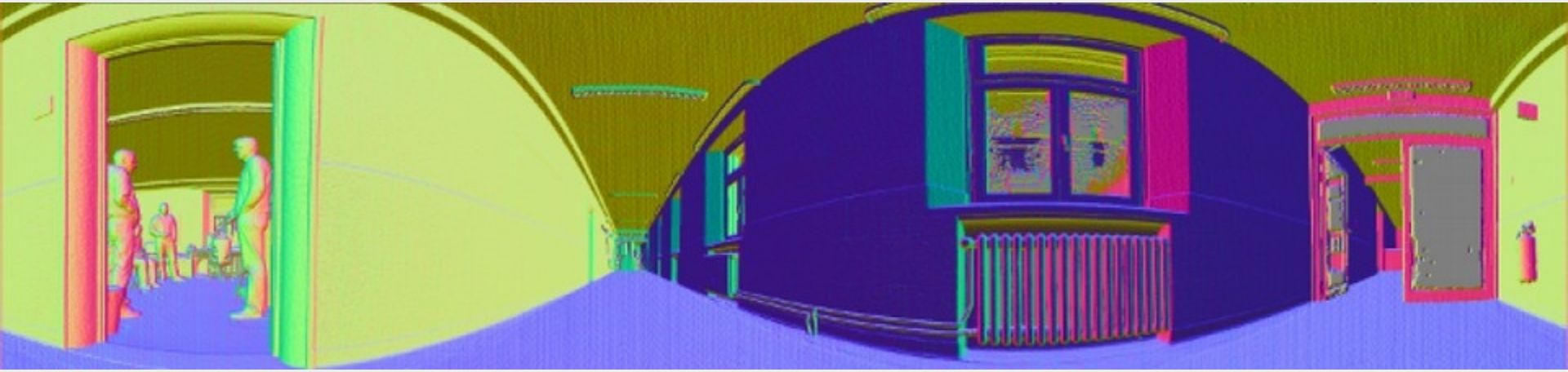


- Introduction
- Basic Data Structures
- Registration and SLAM
- **Applications and Further Tools**

# Calculating Normals

- Using k-NN, adaptive k-NN, radius search, panorama images, ...



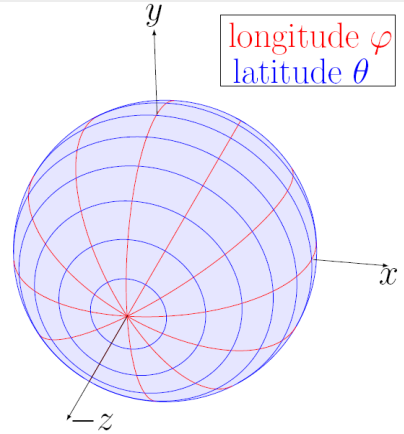


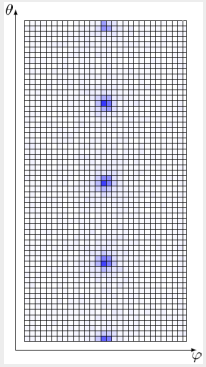


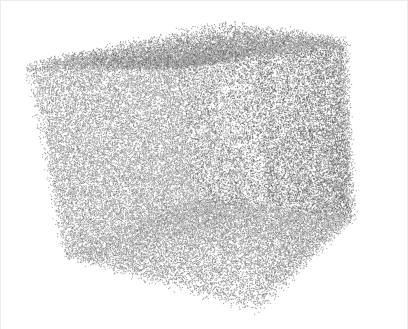
Fast and Accurate Computation of Surface Normals from Range Images, by H. Badino, D. Huber, Y. Park and T. Kanade, ICRA 2011, Shanghai

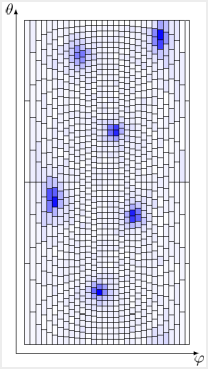
# Plane Estimation

- Hough transform (with novel accumulator array)

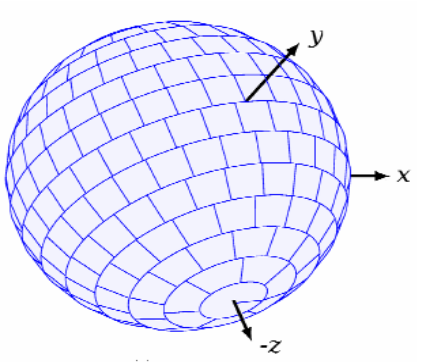


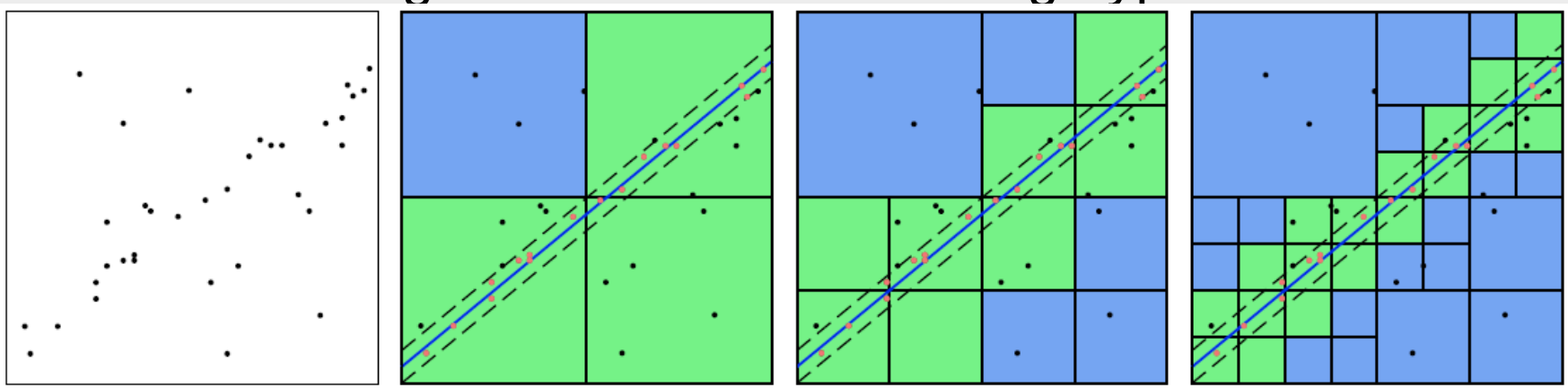




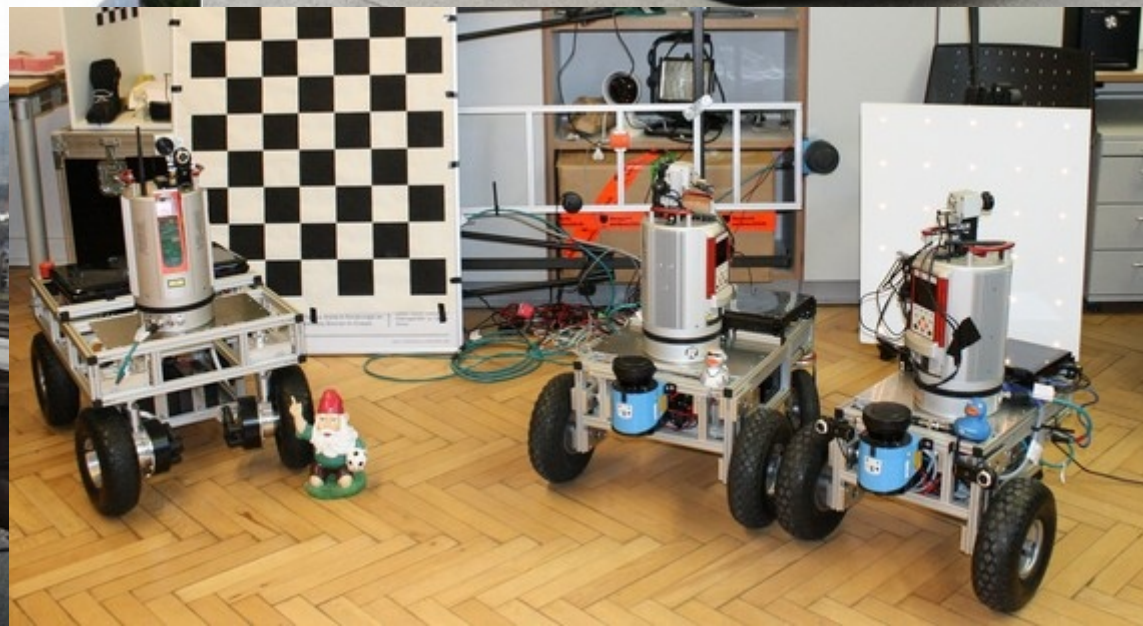
acuumulator array vs. accumulator ball




- RANSAC using the Octree for validating hypotheses

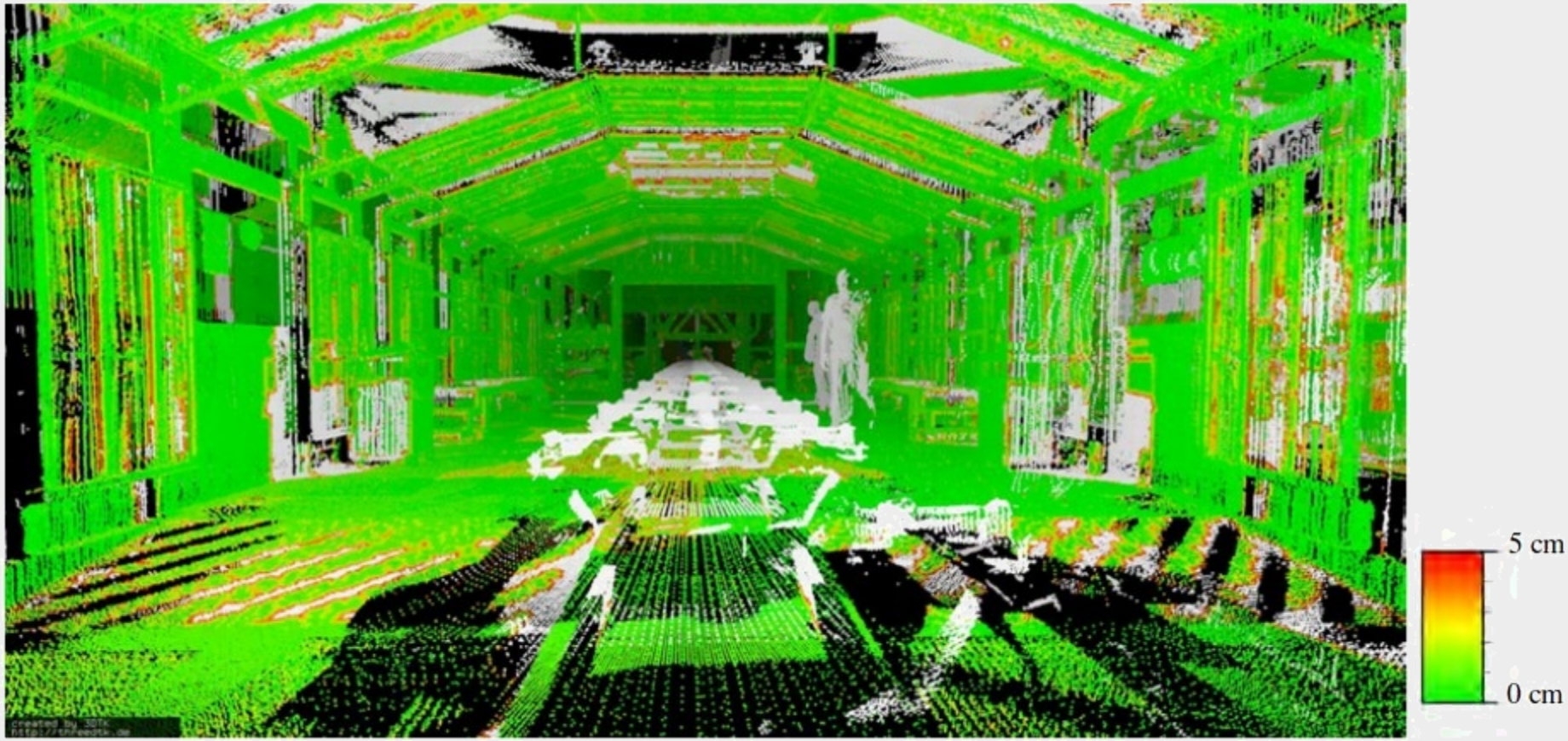


# Applications: Airborne and Mobile Mapping

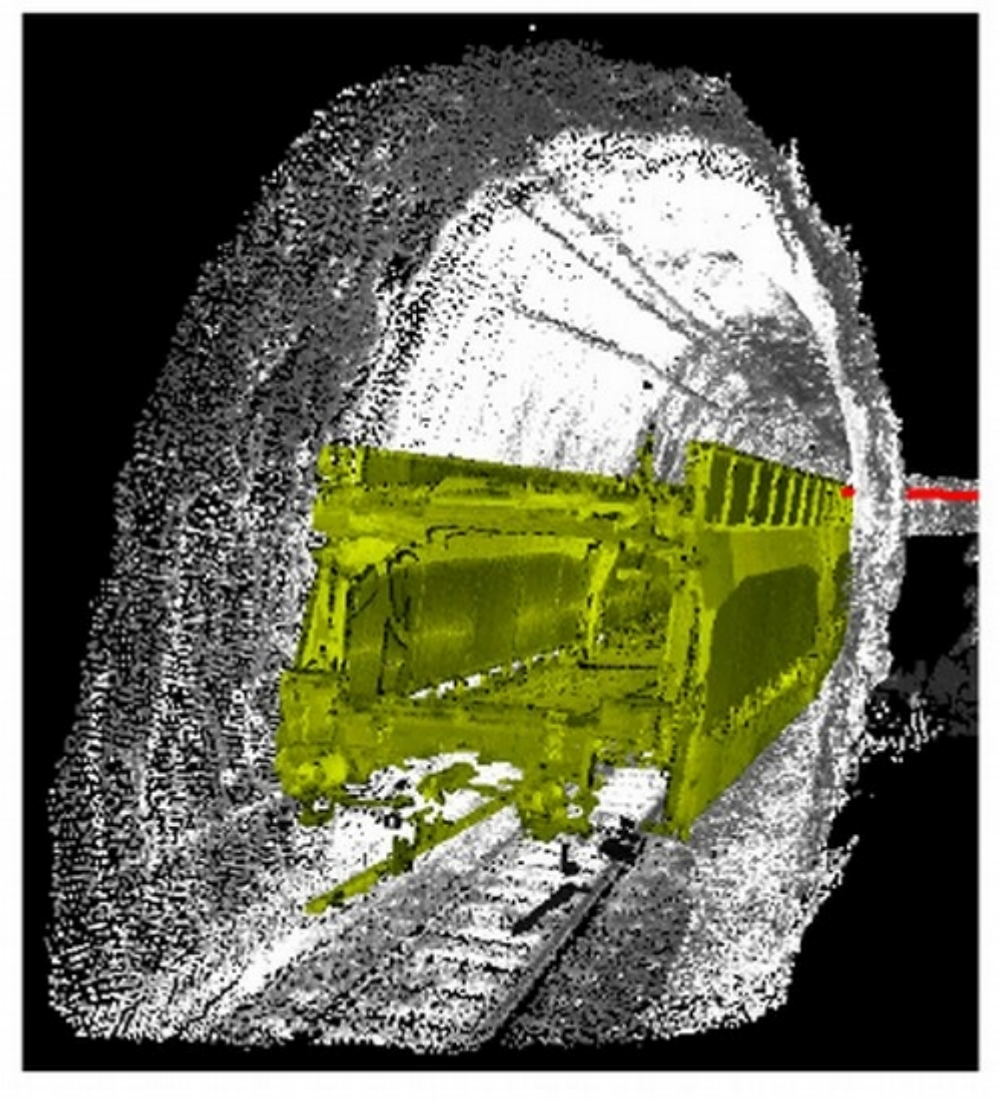
# scan_diff

- Compute the diff between two 3D scans

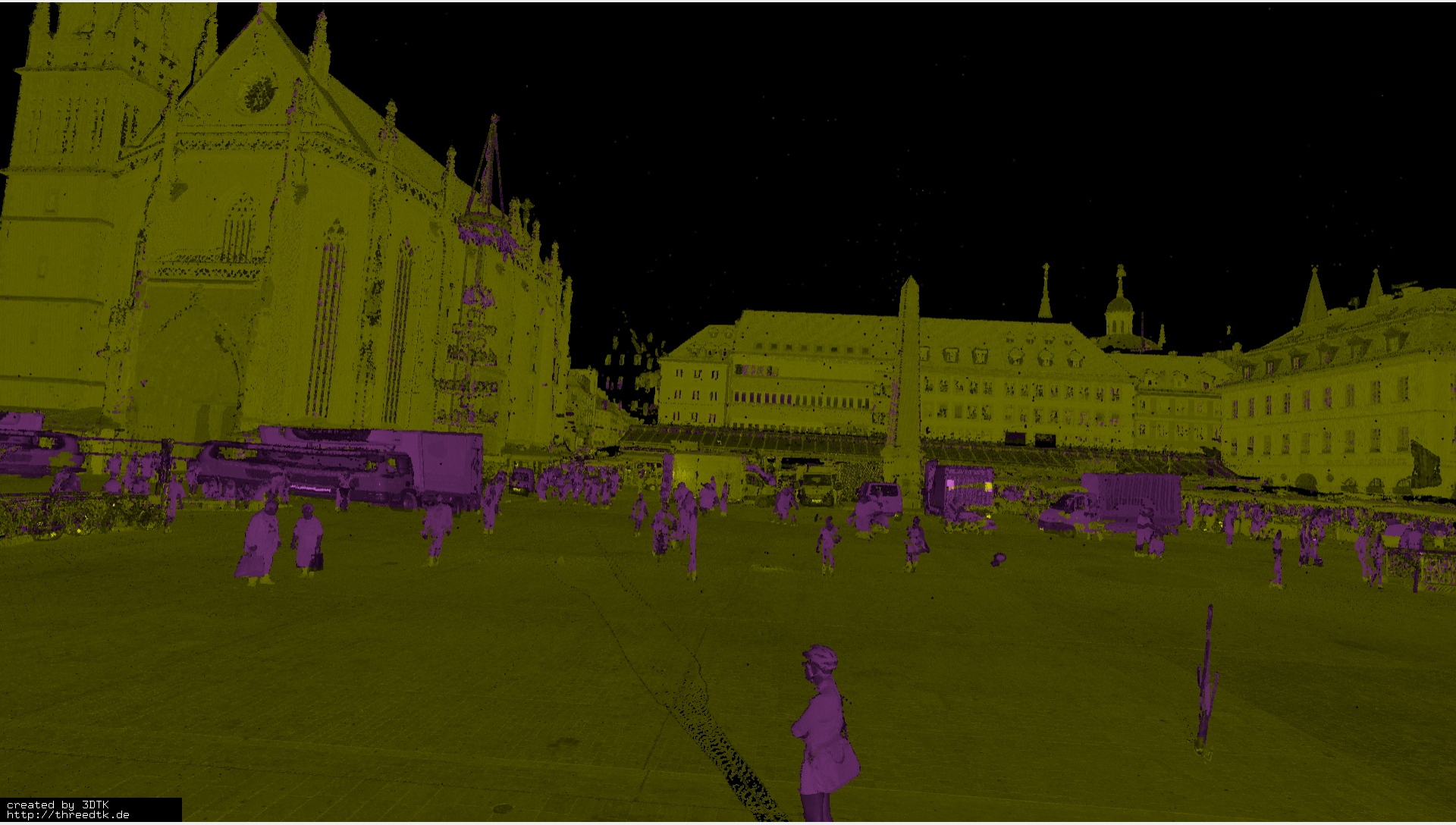


# Collision Detection

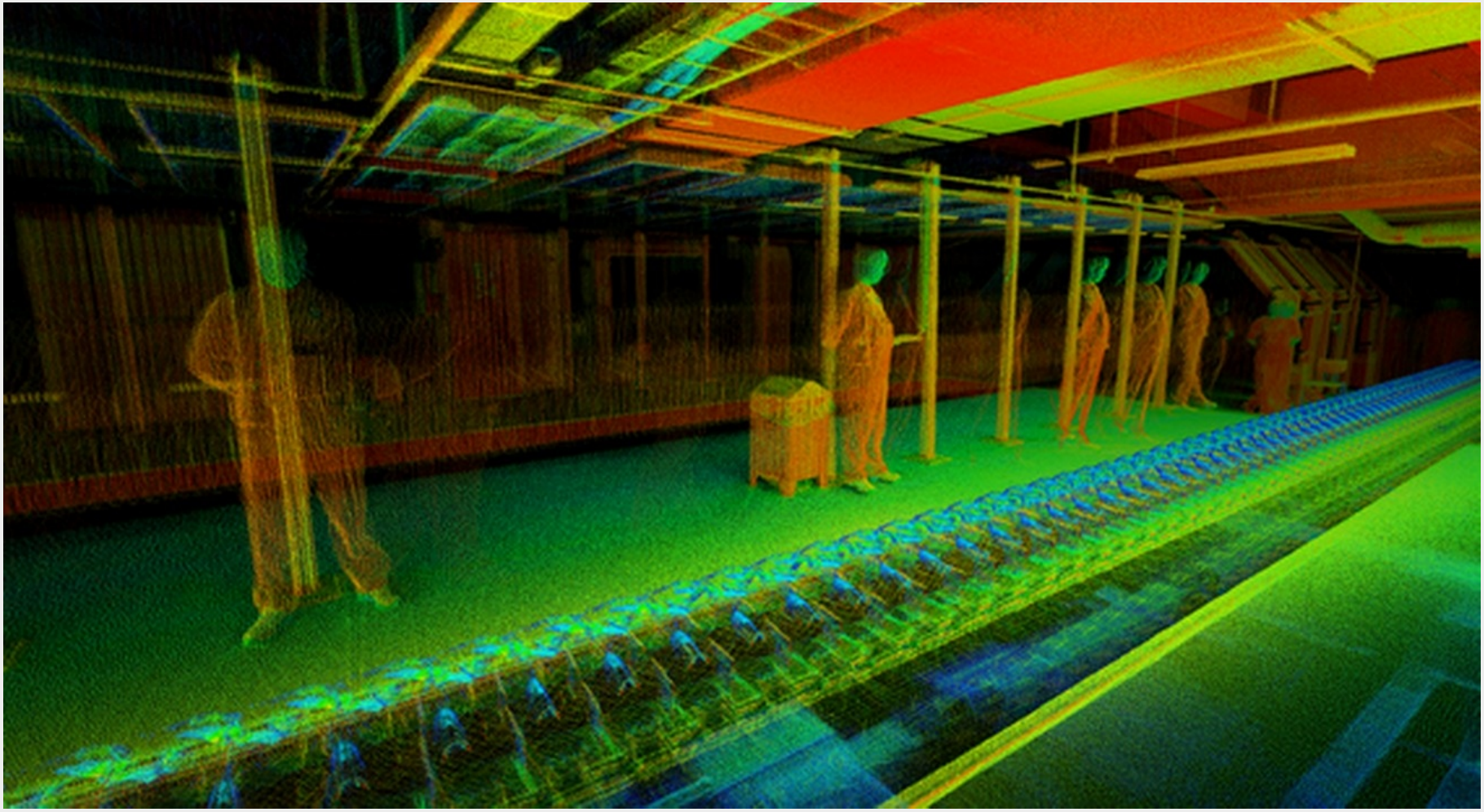
- Between two 3D point clouds, where one is moving along a trajectory



# People Remover - TLS



# People Remover - MLS



Punktwolken am laufenden Band

# Summary and Conclusions

- Efficient algorithms and data structures for processing 3D point clouds
- Global consistent scan matching
  - ⇒ Bundle Adjustment for 3D Point Clouds
  - ⇒ “Dense“ method
- Many tools for efficiently processing large 3D point clouds
- Integrates well with ROS
- Software mostly open-source. Check out:

  3DTK – The 3D Tookit
  **http://threedtk.de**