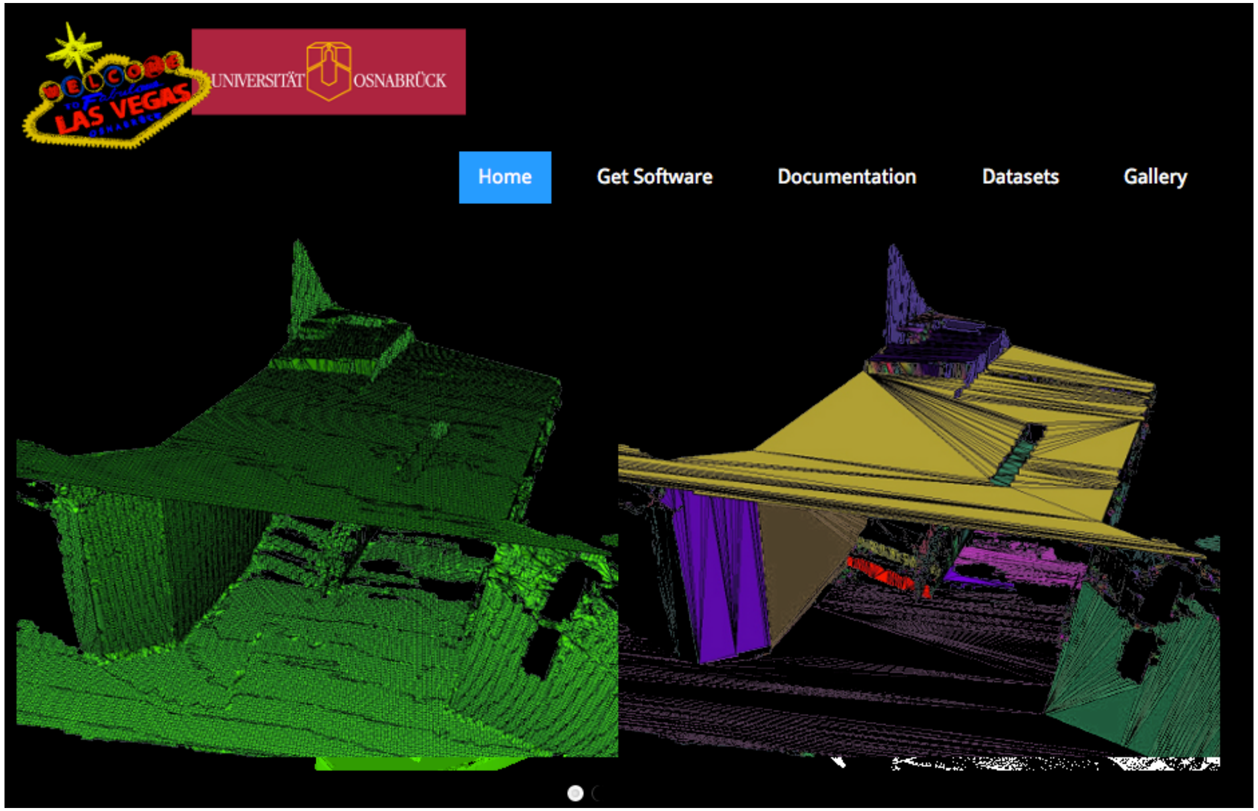
Data Structures for Large Scale Point Cloud Processing

Meshing on Large Point Clouds

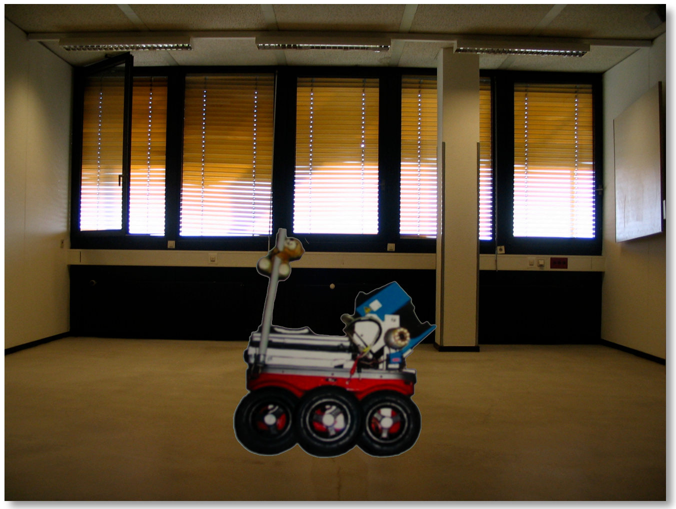
Thomas Wiemann, Andreas Nüchter



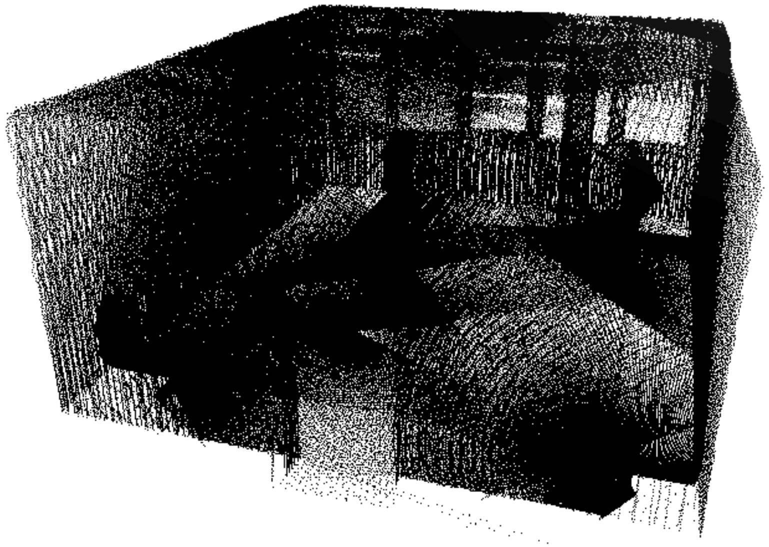
Software: http://www.las-vegas.uni-osnabrueck.de

Introduction

- 3D sensors are commonly used to sample a robot‘s environment



6 DoF SLAM*
with global relaxation

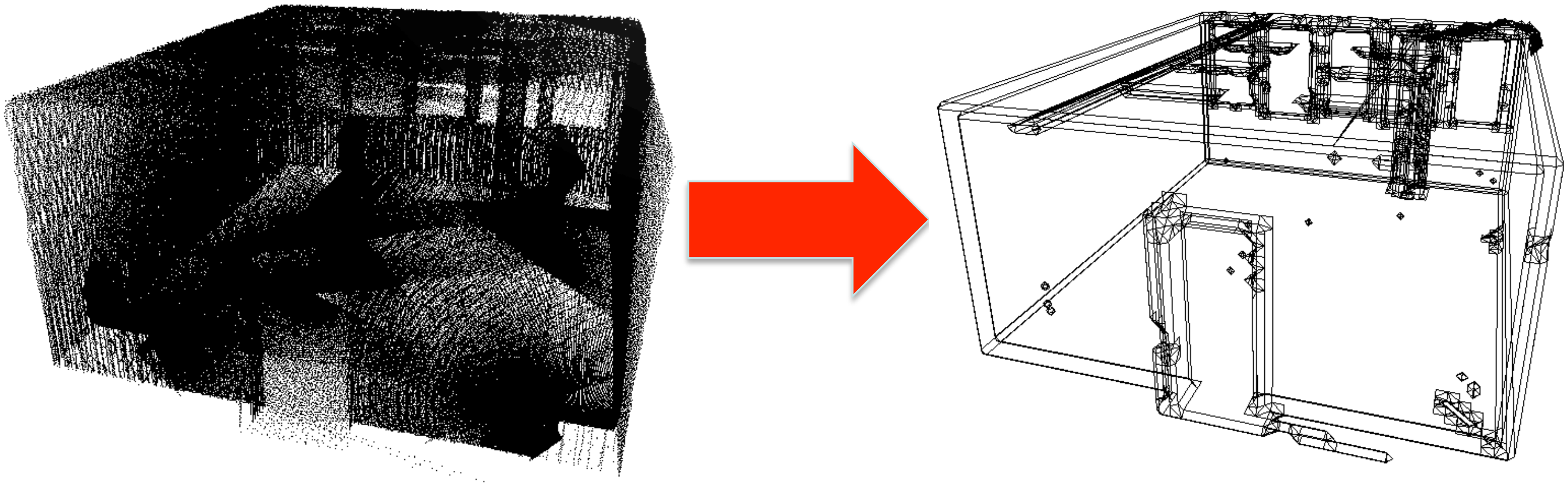


- But we do not get a surface representation, only samples

*Bormann et al. 2008

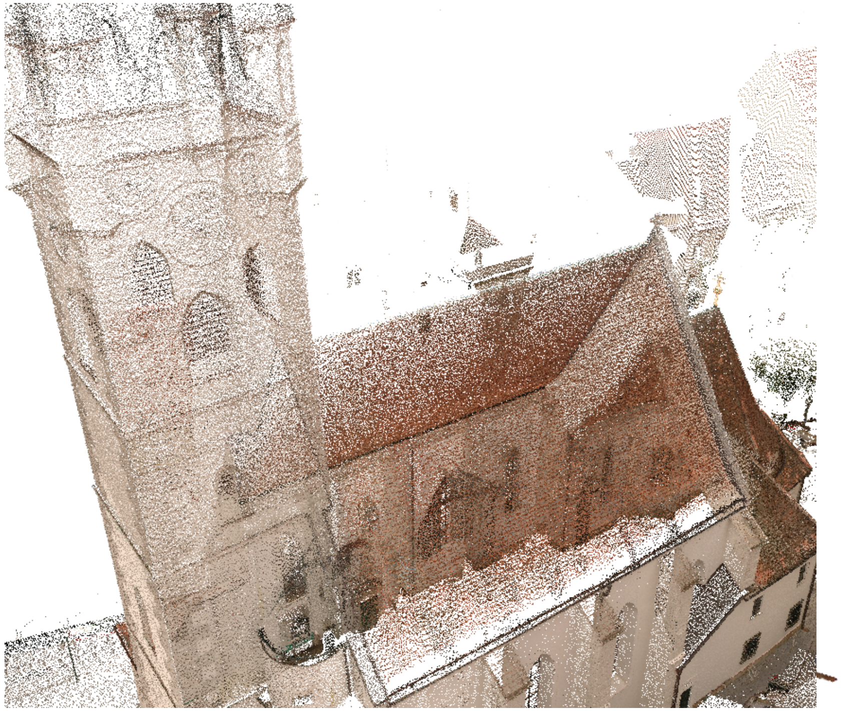
Introduction

- Point Clouds can cantain millions of primitives
- We need a more compact and flexible representation
- Approximate the data with polygons



Approximation Algorithms have been developed in CG

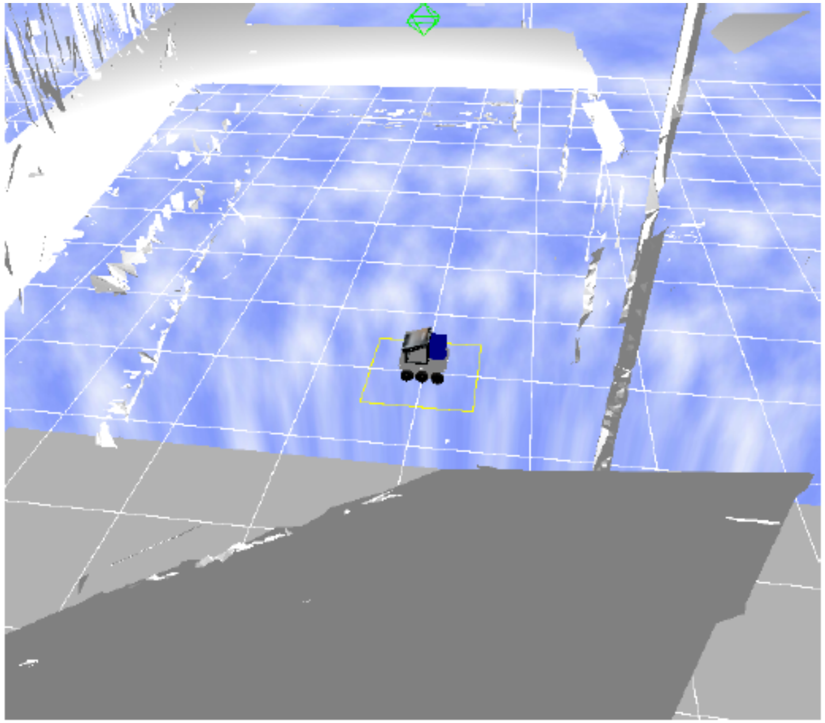
Motivation

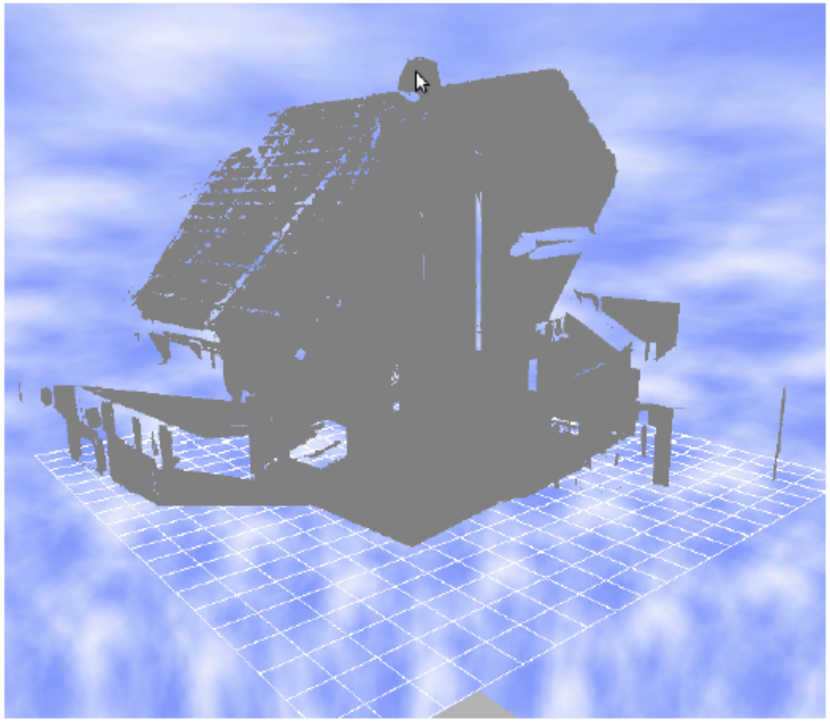


Daten: RIEGL

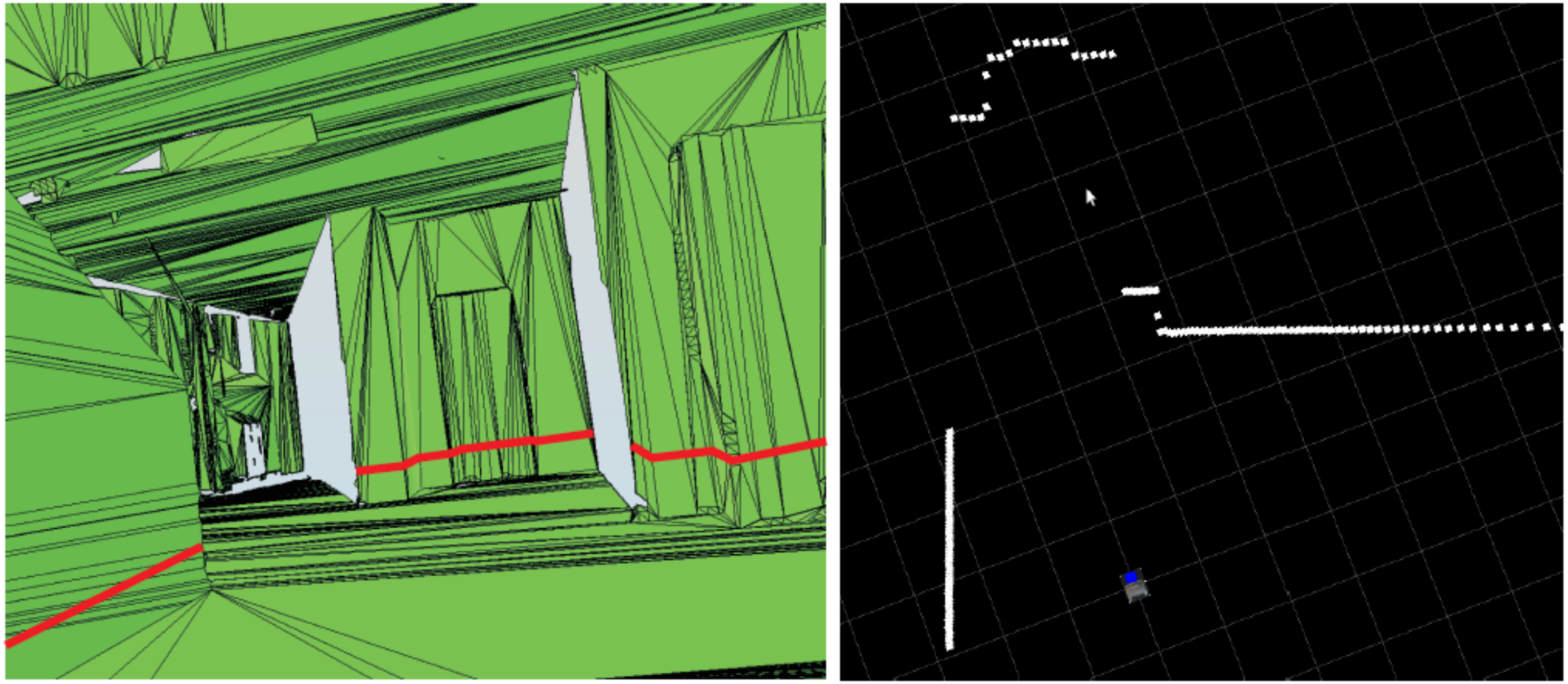
Las Vegas Reconstruction - Example

Application Example

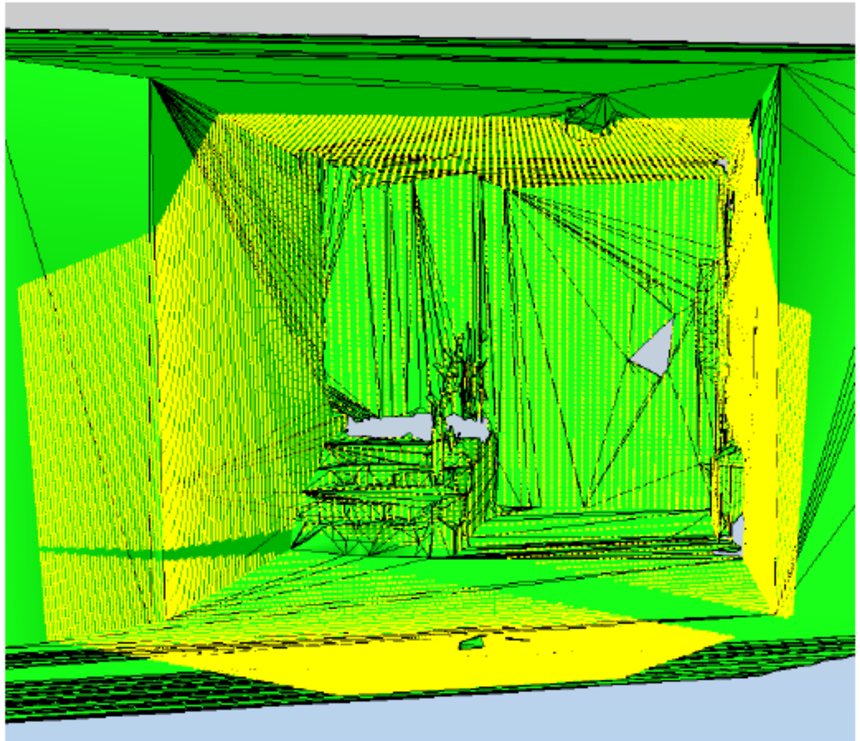


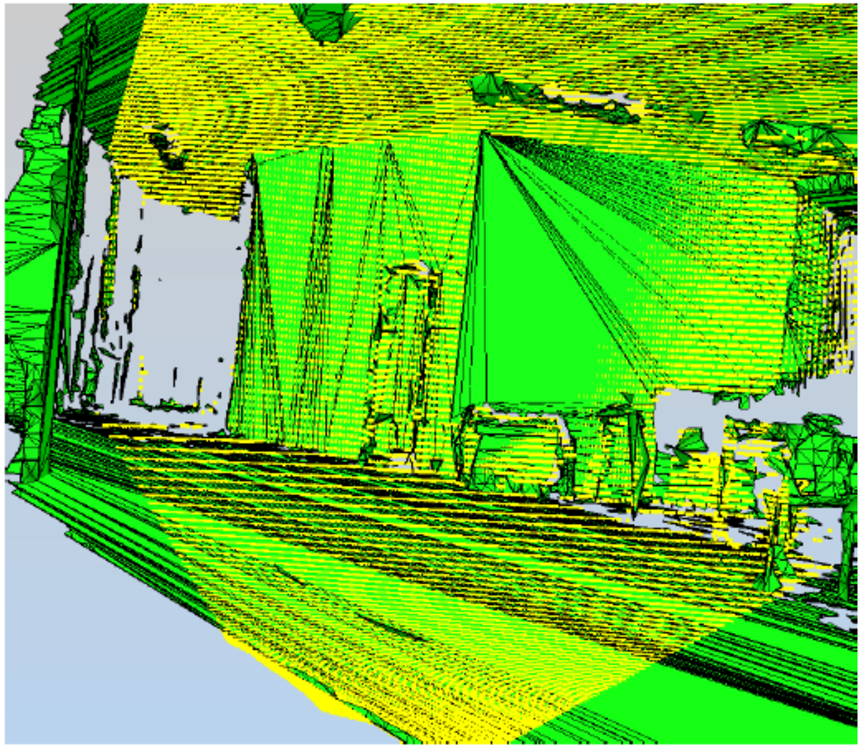


Application Example

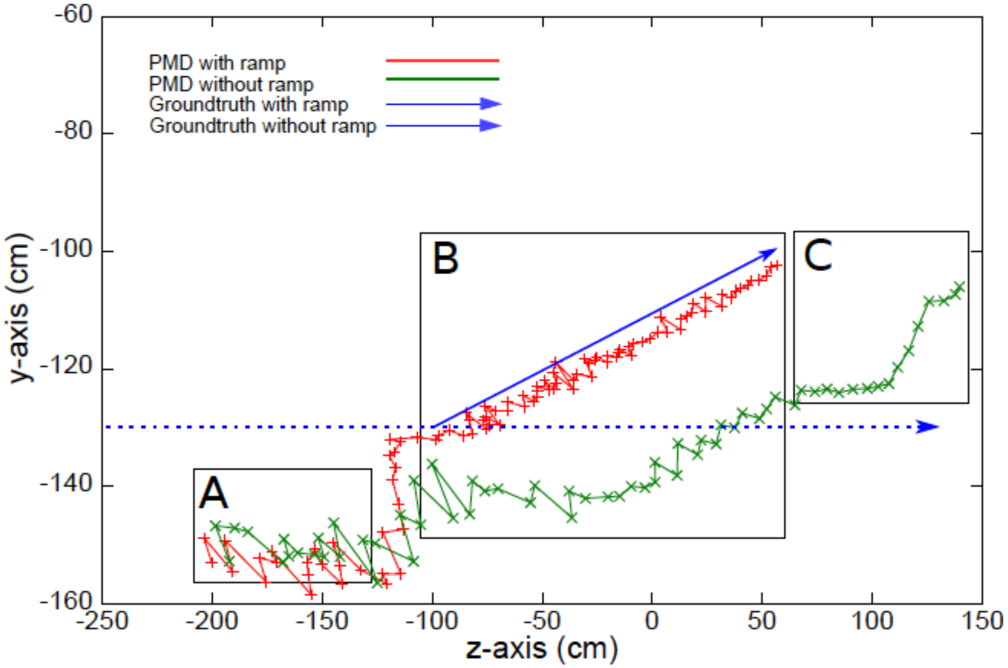


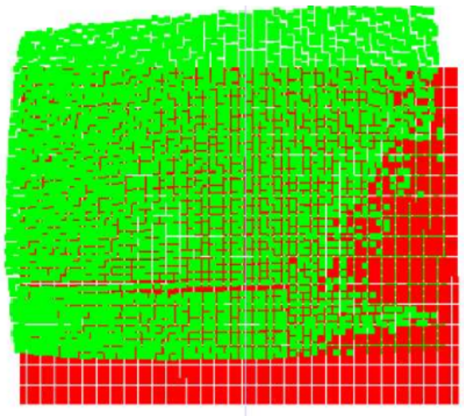
Application Example

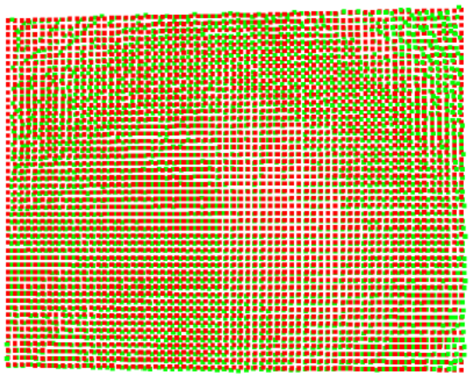


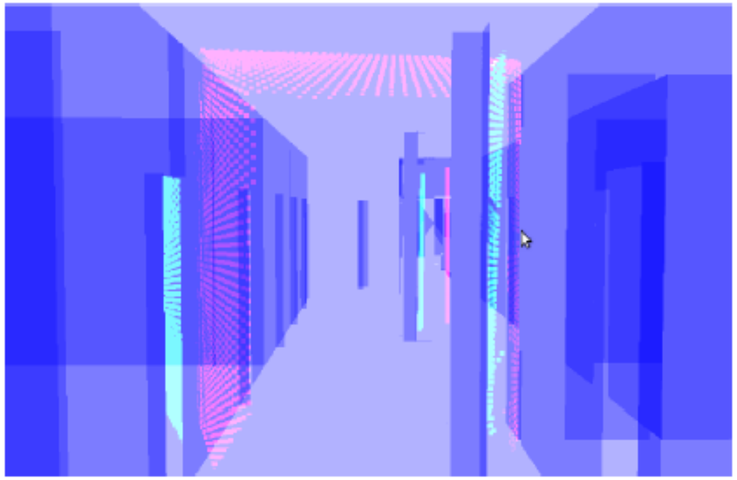


Application Example



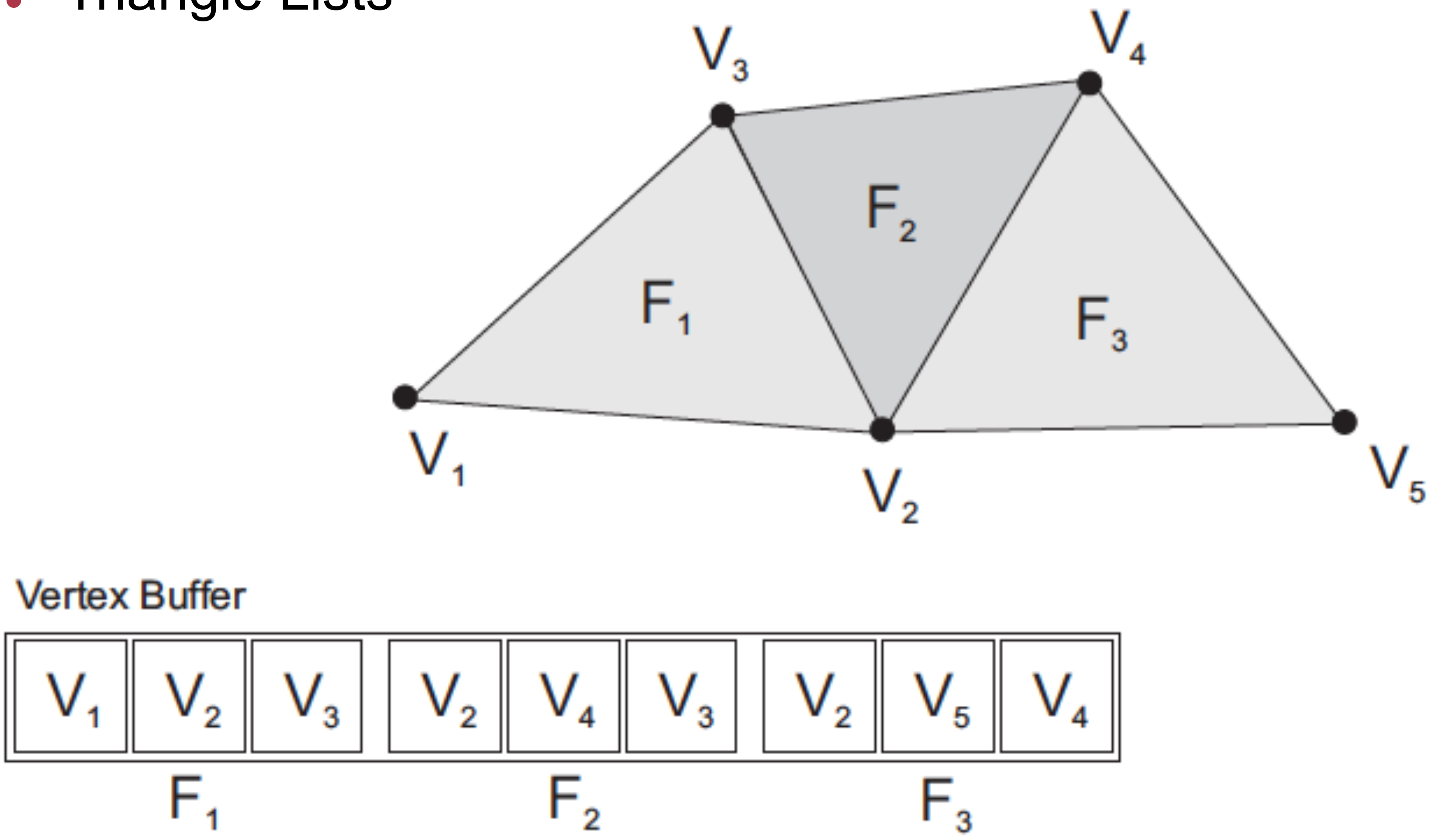






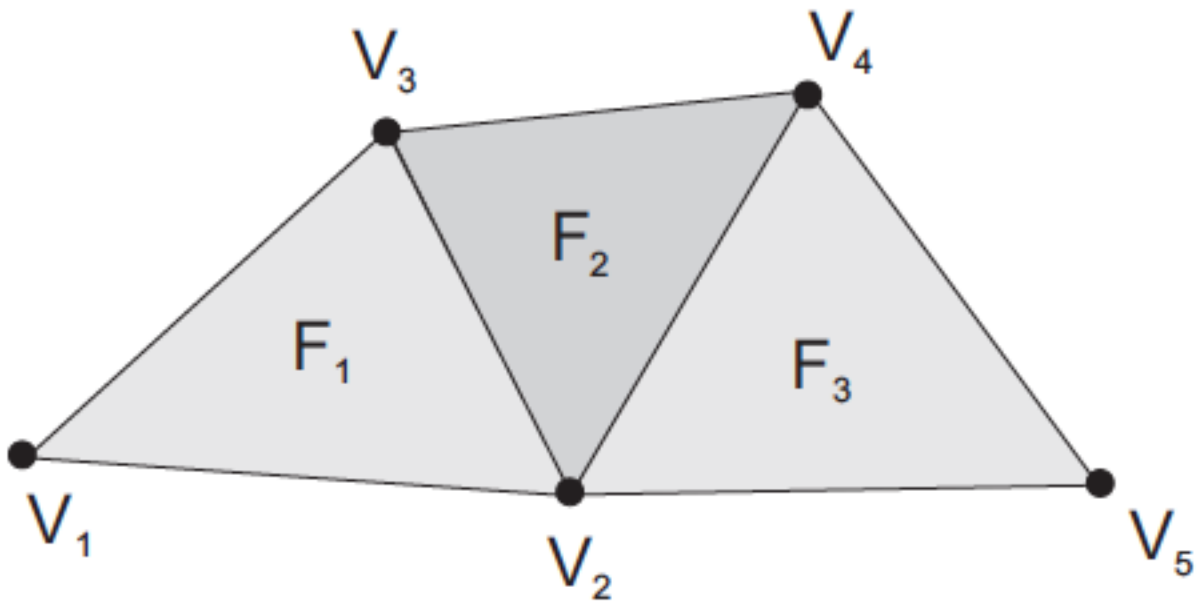
Reconstruction – Mesh Representation

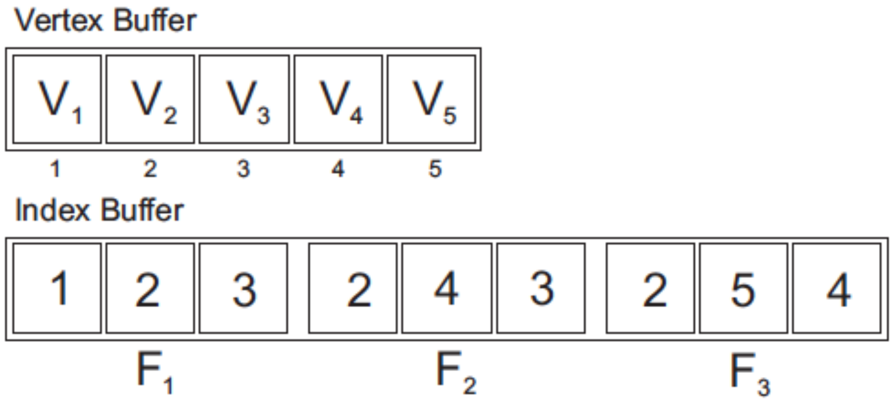
- Triangle Lists



Reconstruction – Mesh Representation

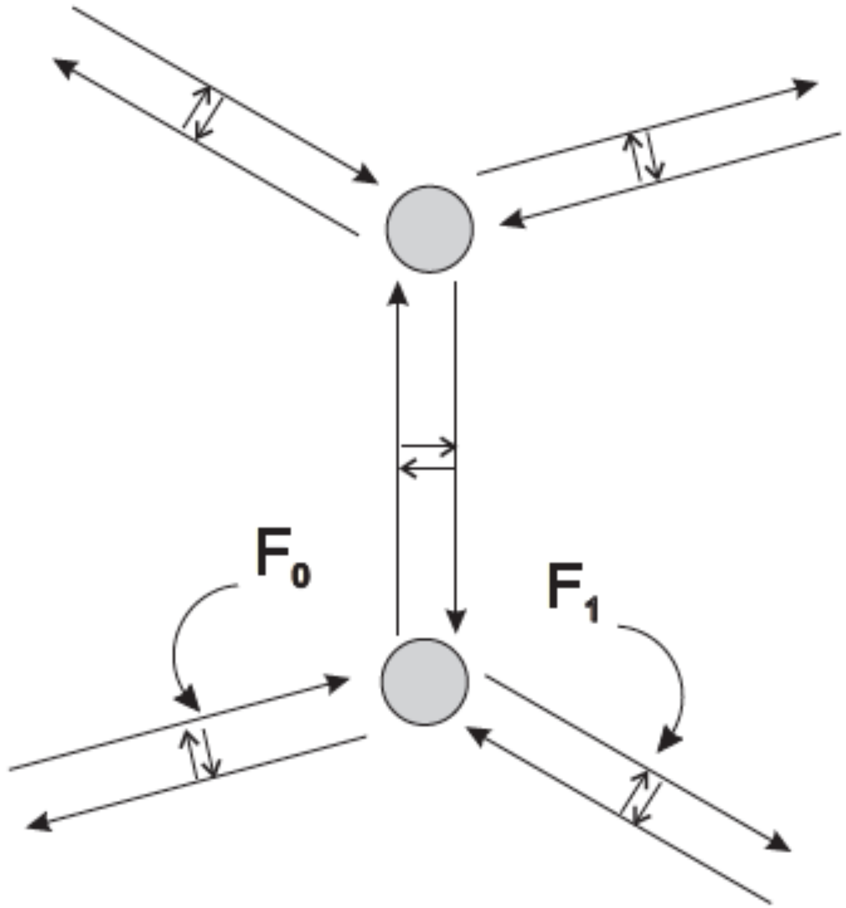
- Indexed buffers avoid redundancies





Reconstruction – Mesh Representation

- Linked data structures allow to find neighboring triangles in constant time
- „Half Edge Mesh“



File Formats

- Stanford .ply:

```
ply
format ascii 1.0
comment this file is a cube
element vertex 8
property float x
property float y
property float z
element face 6
property list uchar int vertex_index
end_header
0 0 0
0 0 1
0 1 1
0 1 0
1 0 0
1 0 1
1 1 1
1 1 0
4 0 1 2 3
4 7 6 5 4
4 0 4 5 1
4 1 5 6 2
4 2 6 7 3
4 3 7 4 0
```

```
ply
format binary_little_endian 1.0
element vertex 3710
property float x
property float y
property float z
property uchar red
property uchar green
property uchar blue
property float nx
property float ny
property float nz
element face 3908
property list uchar int vertex_indices
end_header
<C0>^_9<C2>^H<CD>T<BF><F3>-<C1>C^@<C8>^@<DE>!<B
C^@<C8>^@<DE>!<BA><BD><D7>64<BF>|S4?<C0>^_a<C2>
8o<BF>@<87><AB>>s:I<C2><FF><E5>6@<D3>m<CB>C^@<C
```

File Formats

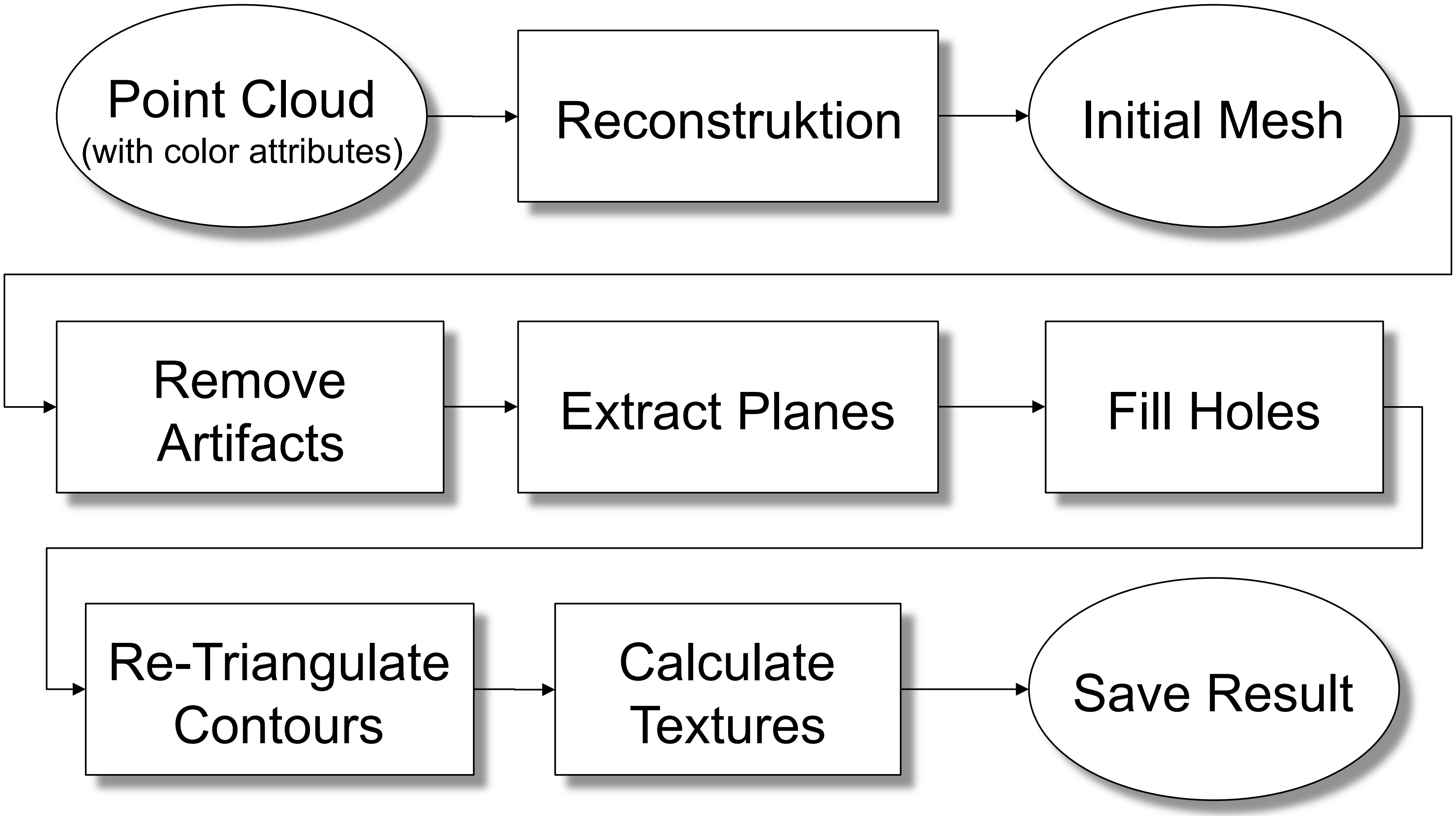
- Wavefront .obj allows to store material and texture information

```
mtllib textures.mtl


## Beginning of vertex definitions.
v 2.21313 27.5737 -3.24302
v 1.93148 27.6632 -3.34341
v 2.20294 27.5767 -3.24272
v 2.21307 27.4955 -3.21163
v 2.13287 27.4975 -3.20067
v 2.05273 27.4996 -3.18979
v 1.97235 27.5011 -3.17861
v 1.89092 27.4997 -3.16612
v 4.91 45.9197 11.3388
v 4.91925 46.0006 11.3388
v 4.8698 46.0366 11.4142
v 4.8478 46.0222 11.414
```

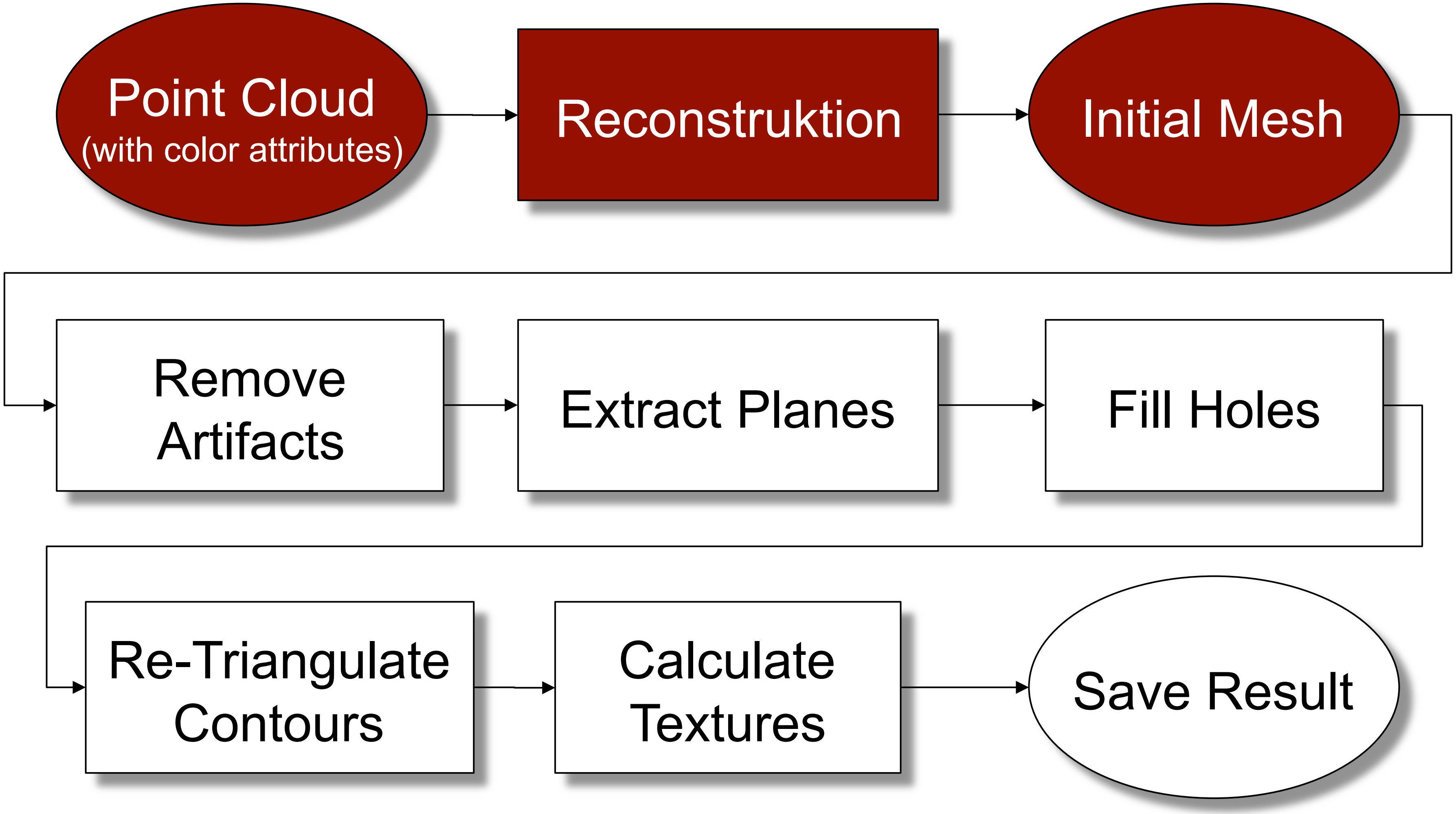
```
newmtl texture_226
Ka 1.000 1.000 1.000
Kd 1.000 1.000 1.000
map_Kd texture_226.ppm

newmtl texture_227
Ka 1.000 1.000 1.000
Kd 1.000 1.000 1.000
map_Kd texture_227.ppm
```

```
f 430141/430141/430141 430120/430120/430120 430118/430118/430118
usemtl texture_227
f 430117/430117/430117 430128/430128/430128 430115/430115/430115
usemtl texture_227
f 430142/430142/430142 430118/430118/430118 430114/430114/430114
usemtl texture_227
f 430143/430143/430143 430127/430127/430127 430125/430125/430125
usemtl texture_227
f 430144/430144/430144 430109/430109/430109 430106/430106/430106
```

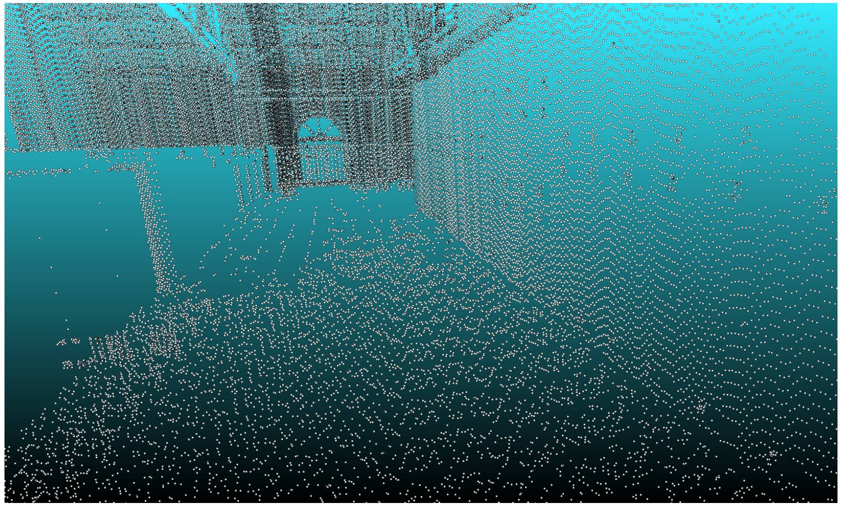
Processing Pipieline

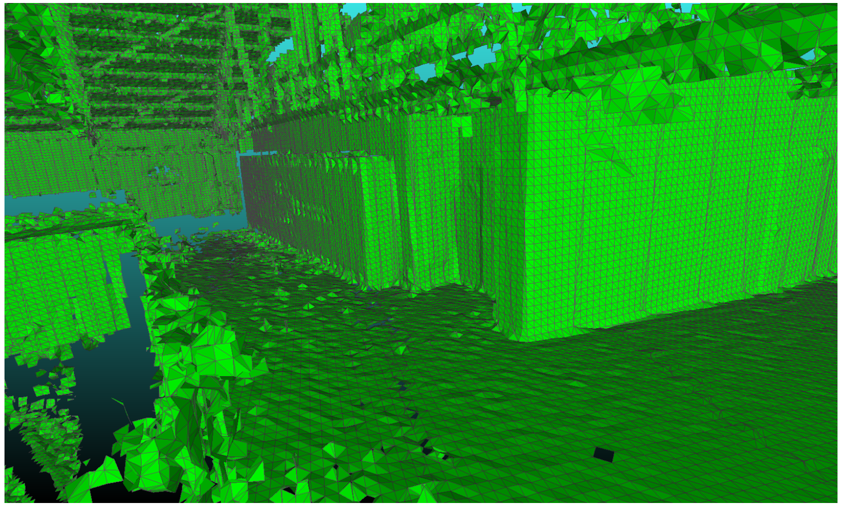


Processing Pipieline



Reconstruction





- Reconstruction using Marching Cubes variants
- Using Hoppe‘s signed distance function
- Different methods for normal estimation
- Store the mesh a Half-Edge-Represention
- Do everything in parallel if possible

Initial Mesh Generation

- Idea: Use a modified Marching Cubes Algorithm*:
  - Divide space into cubic cells of equal size
  - Determine the cell corners, that are outside a given surface
  - Use pre-computed patterns to approximate the surface

- Output: List of triangles that approximate the surface

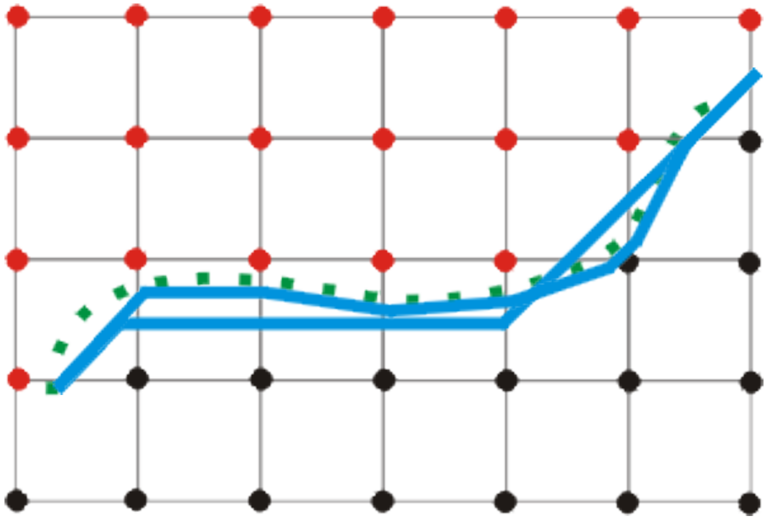
- Enhancements
  - Use hashing and look-up tables to find duplicate vertices
  - Modified octree to generate a grid
  - Integrate the found triangles into a half edge representation
  - Find adjacent faces and surrounding edges in constant time

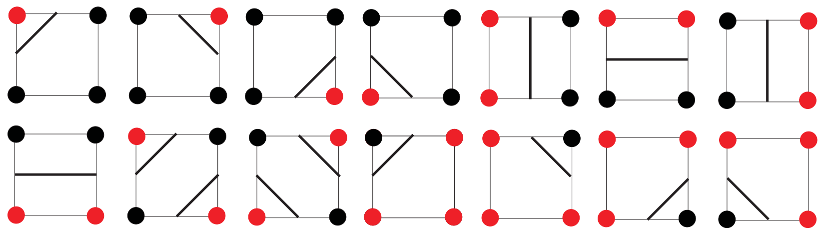
➪ Implementation issues

*Lorensen & Cline 1987

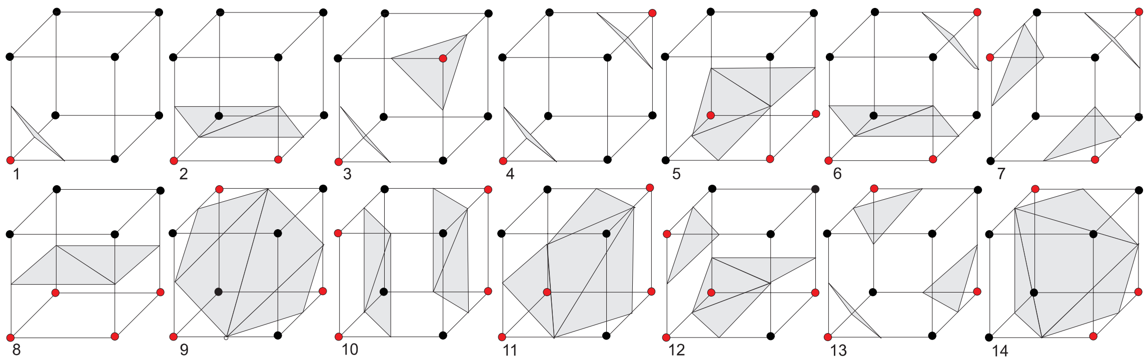
Marching Cubes

2D Example:





In 3D 14 basic patterns are needed:



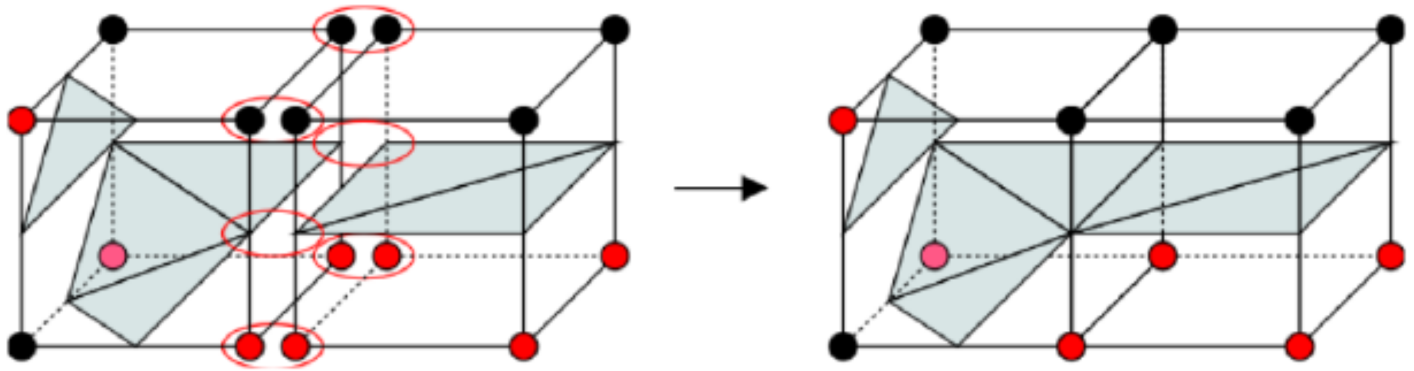
Grid Representation

- LVR uses a hashing based grid for reconstruction

$$i = \lfloor \frac{x - x_{min}}{v} \rfloor, \qquad j = \lfloor \frac{y - y_{min}}{v} \rfloor, \qquad k = \lfloor \frac{z - z_{min}}{v} \rfloor$$

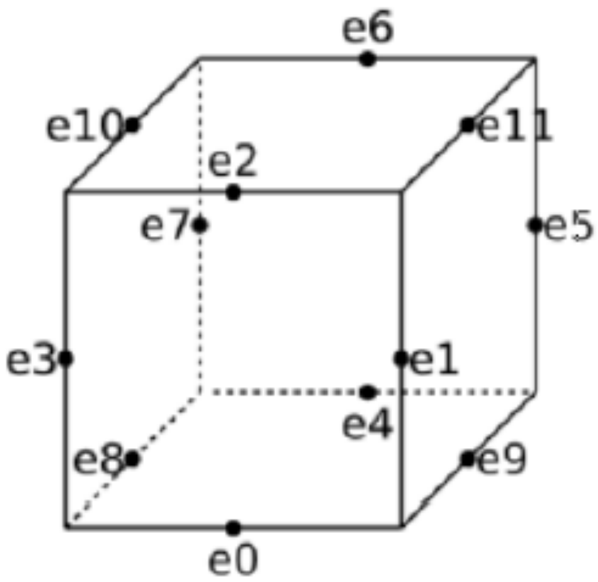
$$\mathcal{H}(i, j, k) = i \cdot dim_x \cdot dim_y + j \cdot dim_y + k$$

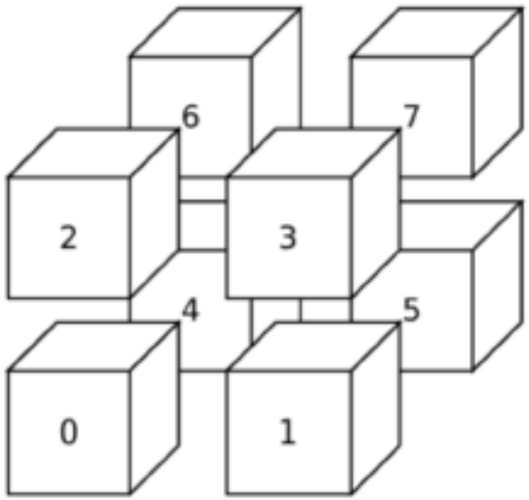
- Easy to search for redundancies



Search for Redundancies

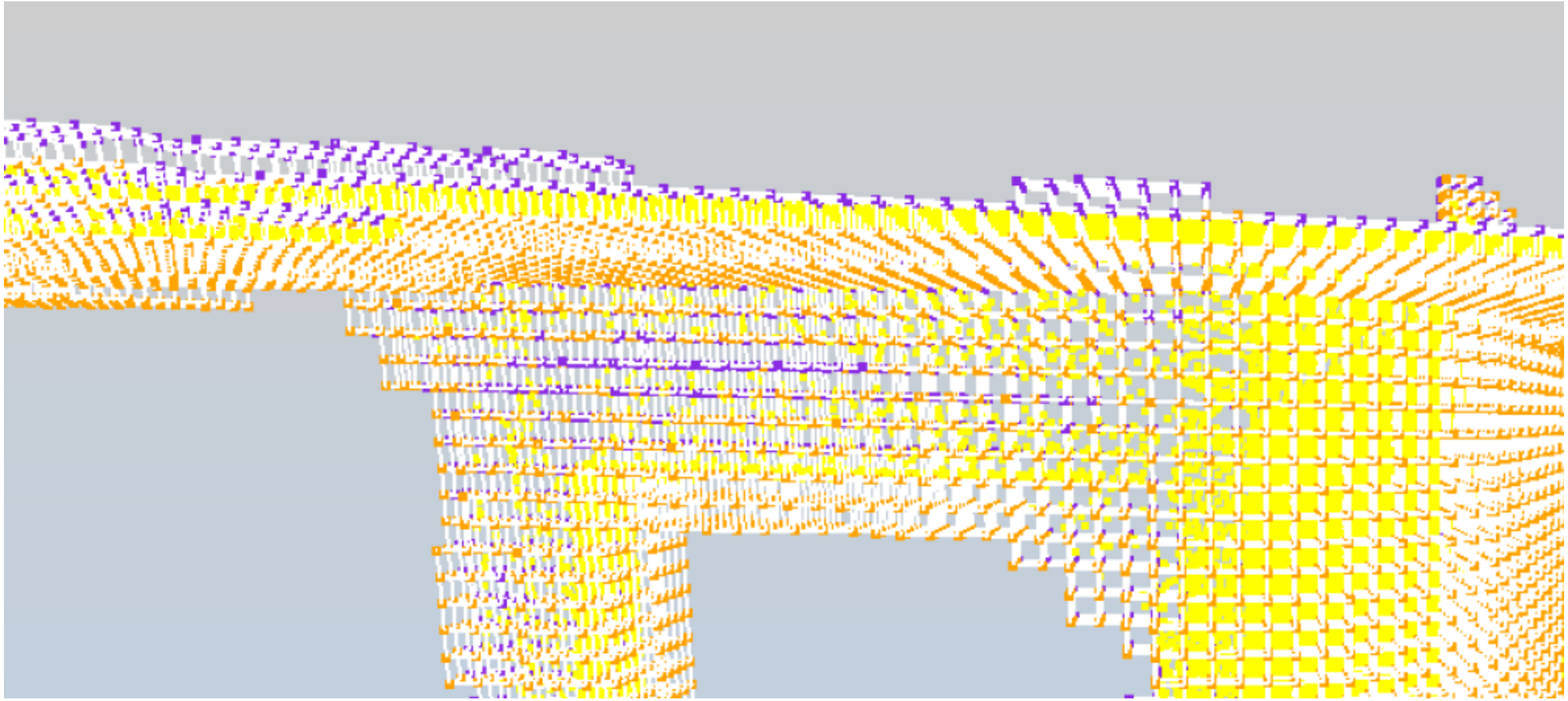
- Efficient search for neighbour voxels



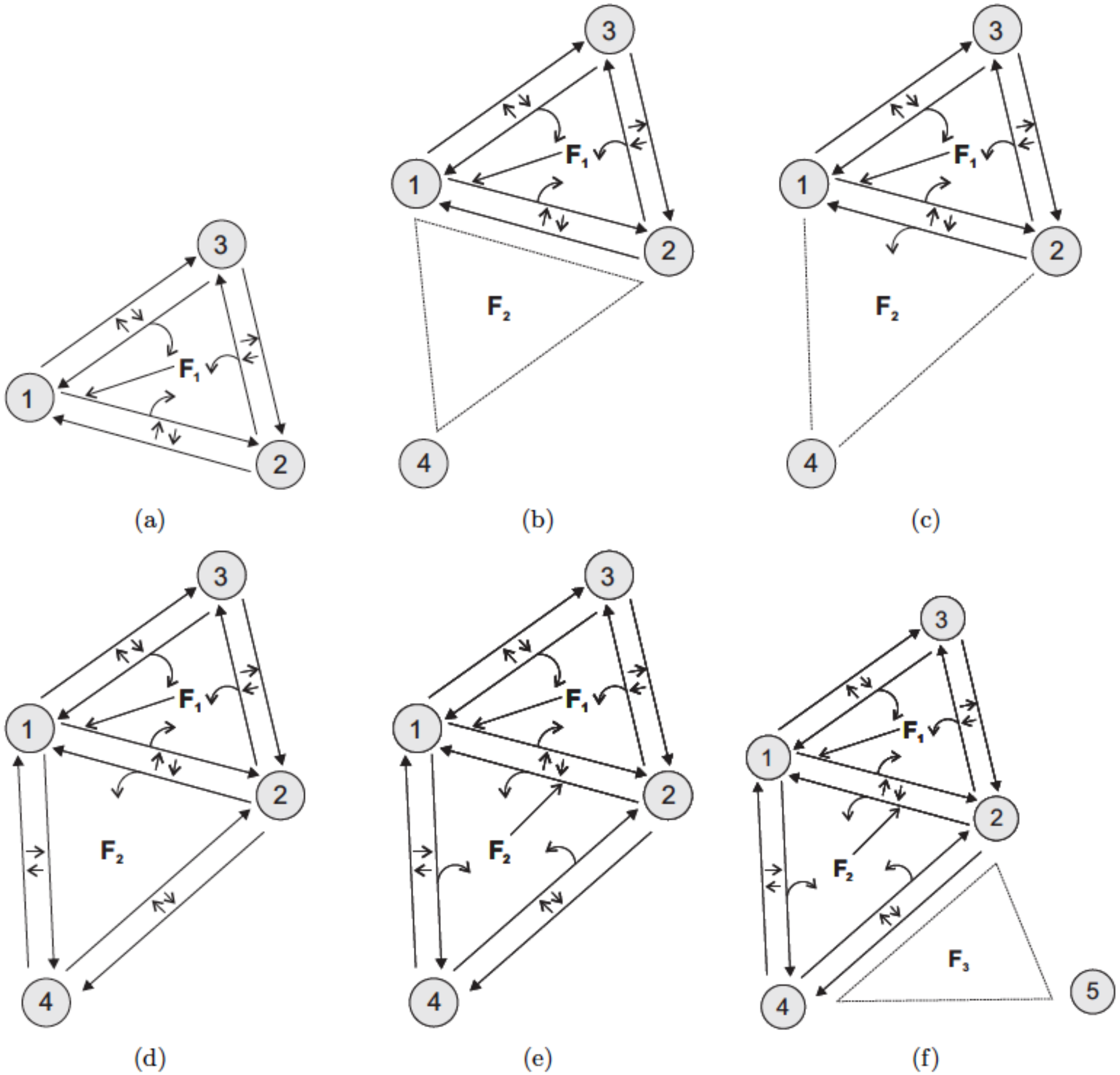


Vertex	Neighbors		
e_0	4	2	6
e_1	3	5	7
e_2	0	6	4
e_3	1	5	7
	...		

Grid Example



Generating a Half Edge Mesh

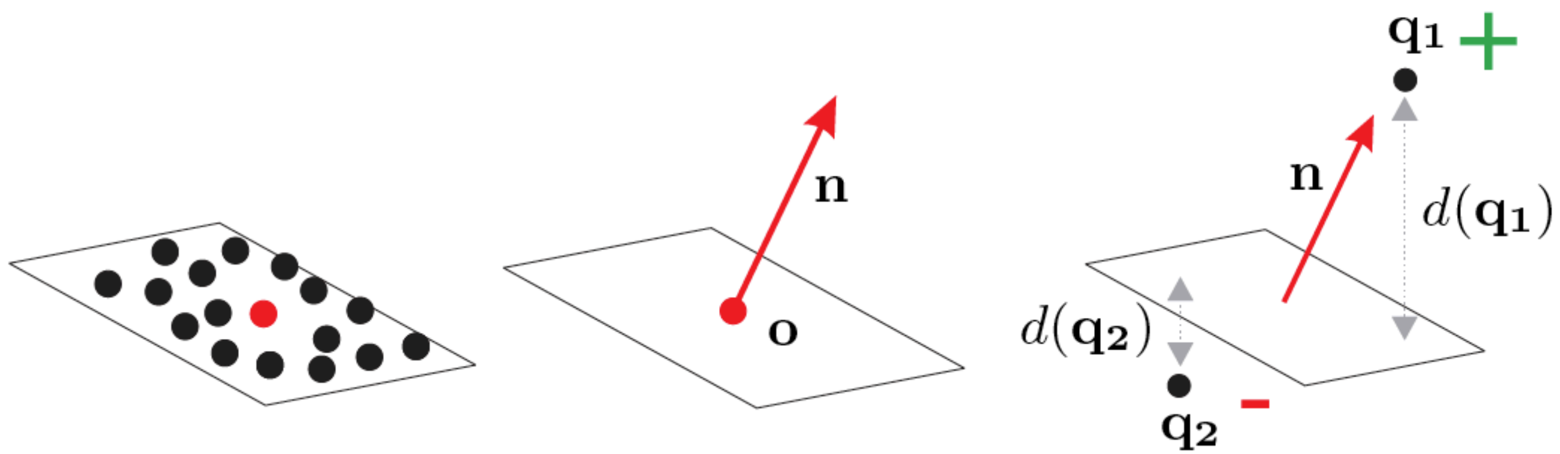


Generating a Half Edge Mesh

```
edges ← empty list of half edges
faces ← empty half edge face
for i = 0 to 3 do
    currentVertex ← vertexIndices[i]
    nextVertex ← vertexIndices[(i + 1) mod 3]
    if edge to current vertex exists then
        edges[i] ← pair-edge of edge to vertex
        edges[i].face ← currentFace
    else
        create new edge with corrensponding pair and link them
        update in and out lists of edge vertices
        edges[i] ← newly created half edge
    end if
end for
for i = 0 to 3 do
    edges[i].next ← edges[(i + 1) mod 3]
end for
face ← edges[0]
add face to mesh
```

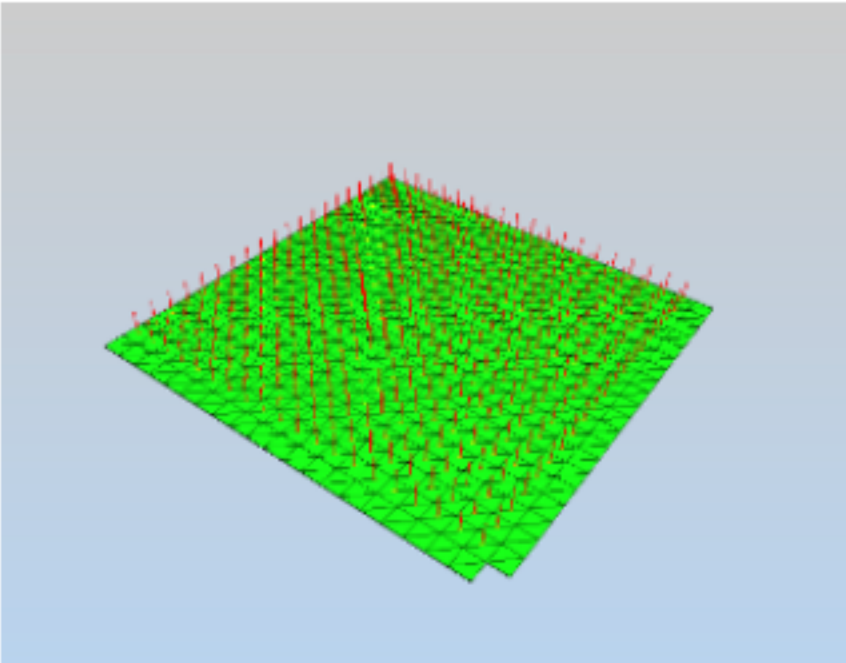
Surface Interpolation

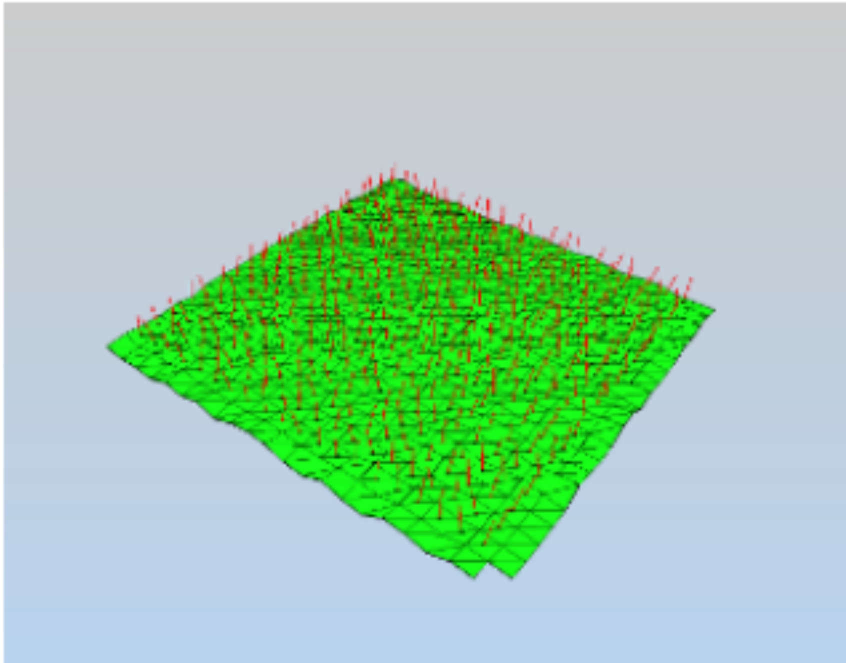
- Hoppe's signed distance function (1992):



- Get consistent normal orientations
- Flip normals towards scanning position
- Interpolate surface intersection using the signed distance on two corners

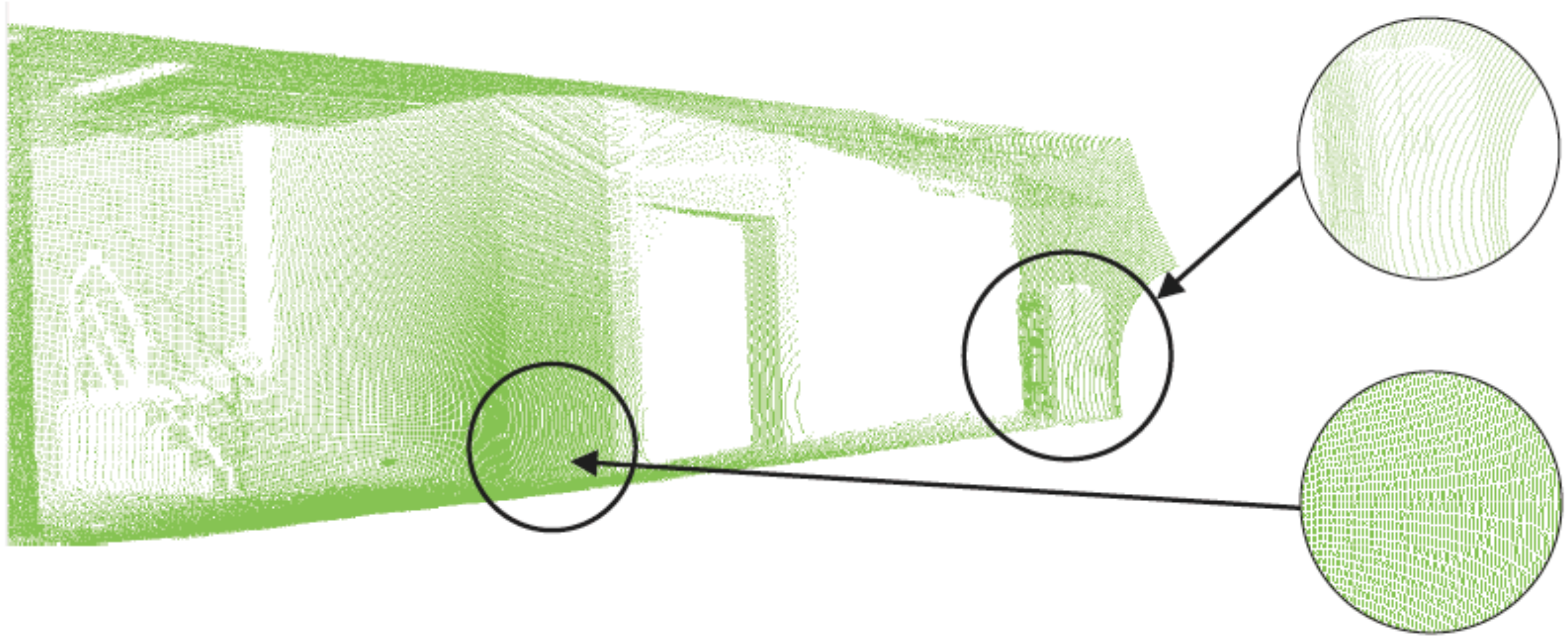
Normal Quality





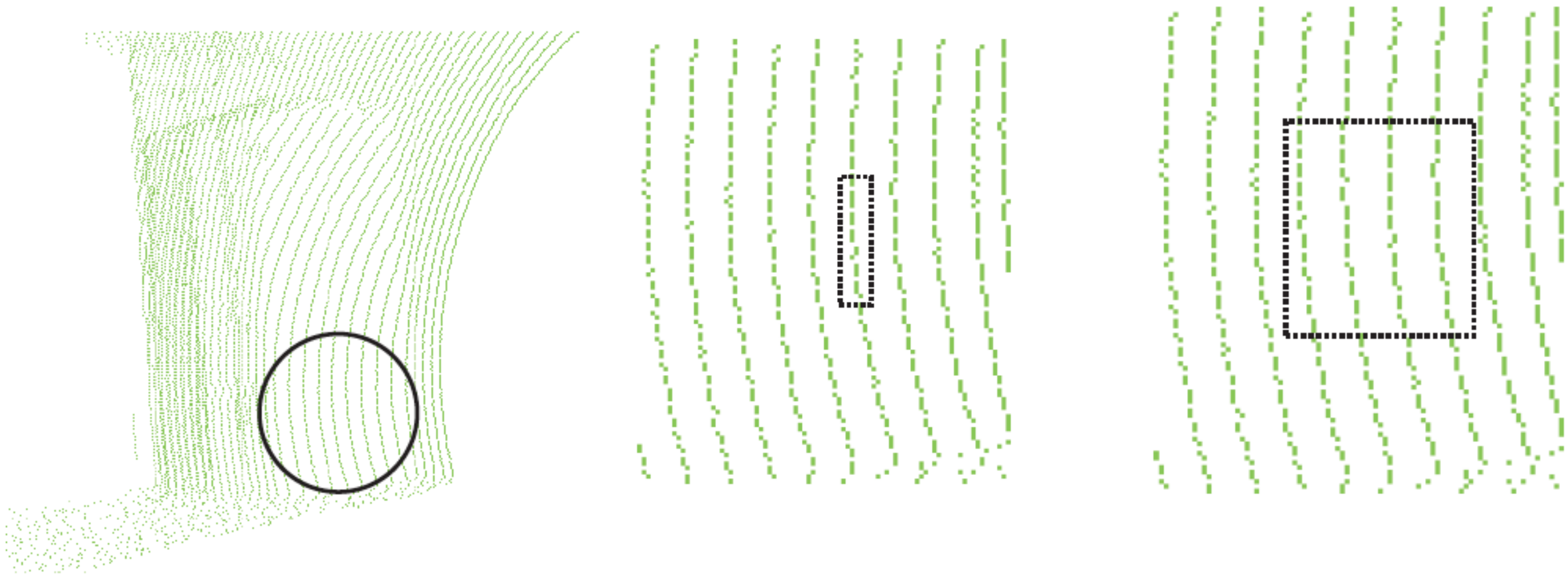
Surface Interpolation

- Hoppe's approach works fine for dense data
- But:



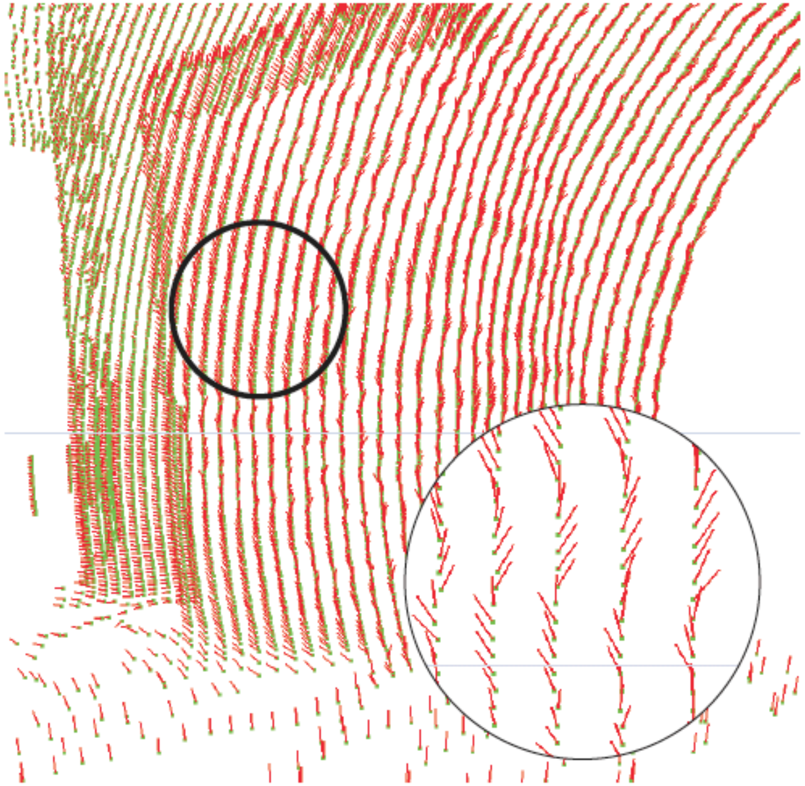
Robust Normal Estimation

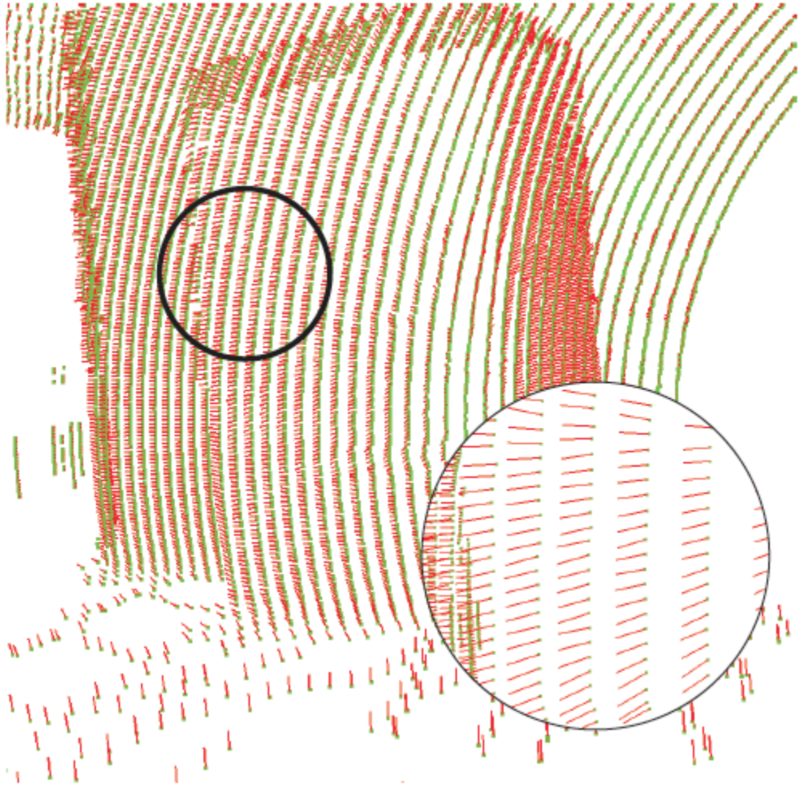
- The number of points needed for a robust normal estimation depends on noise and point density
- Use heuristic to determine the optimal number
- Analyze the bounding box of the *k*-neighborhood



Robust Normal Estimation

- Results:



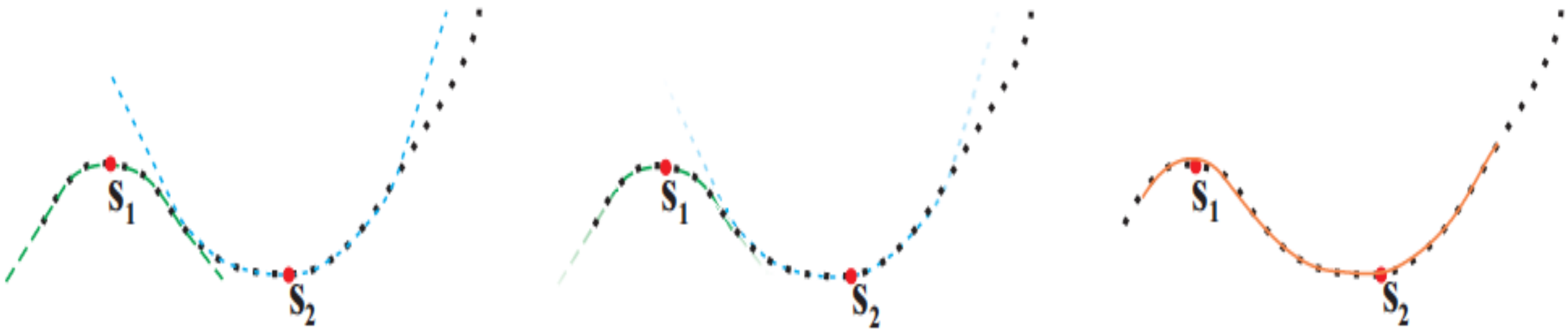


Robust Normal Estimation

- Influence on reconstruction accuracy:

Moving Least Squares

- Higher order approximations
- Moving Least Squares:

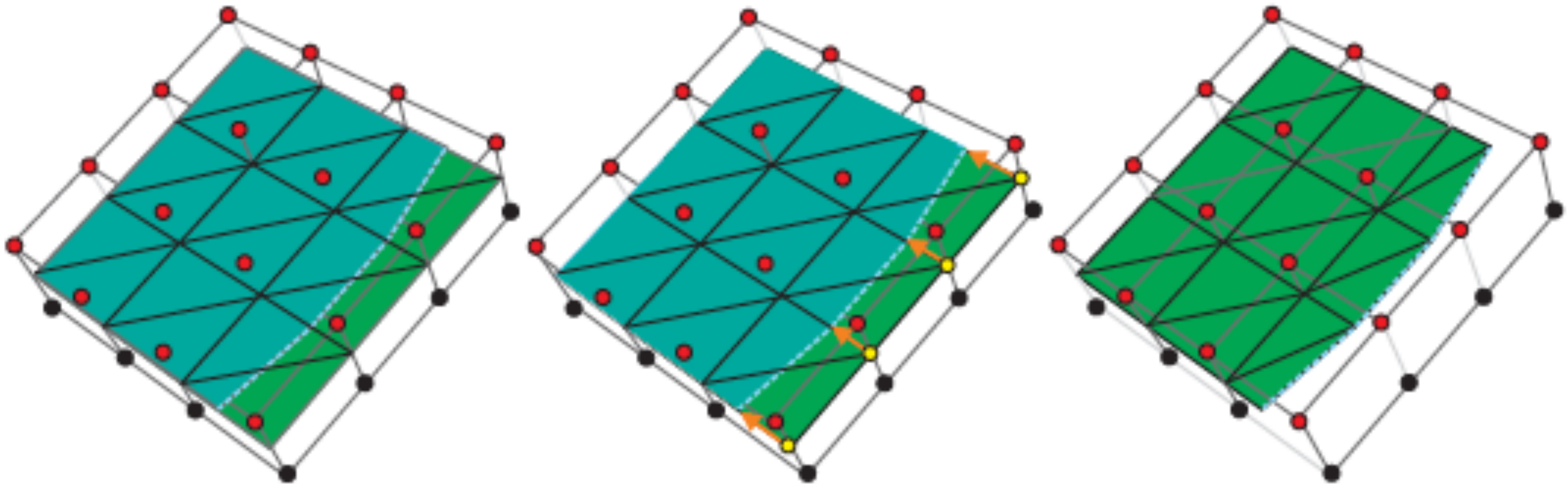


Testing it!

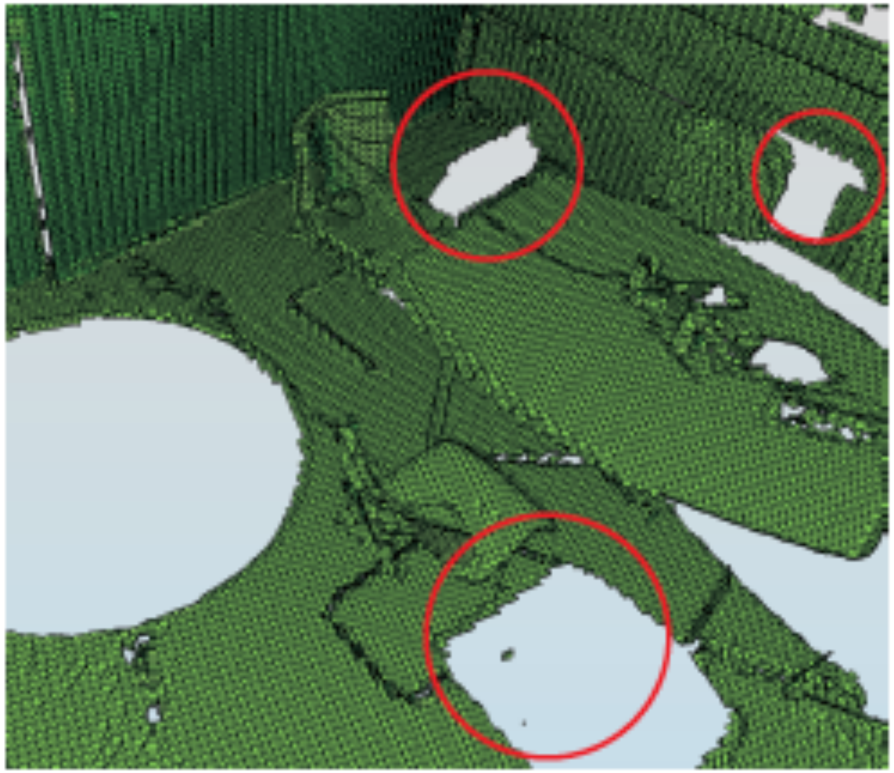
- Call `bin/reconstruct dat/scan.pts`
- Further elevant parameters:
- `-v --kd --kn --ki`
- Voxelsize, NN-Search parameters
- What is the correct voxelsize?
- `-i`
- Try different parameter sets for yourself on the datasets in `dat/scanxxx.3d`
- Hint: use `--e` to export good normals
- Use `bin/qviewer` to display the results

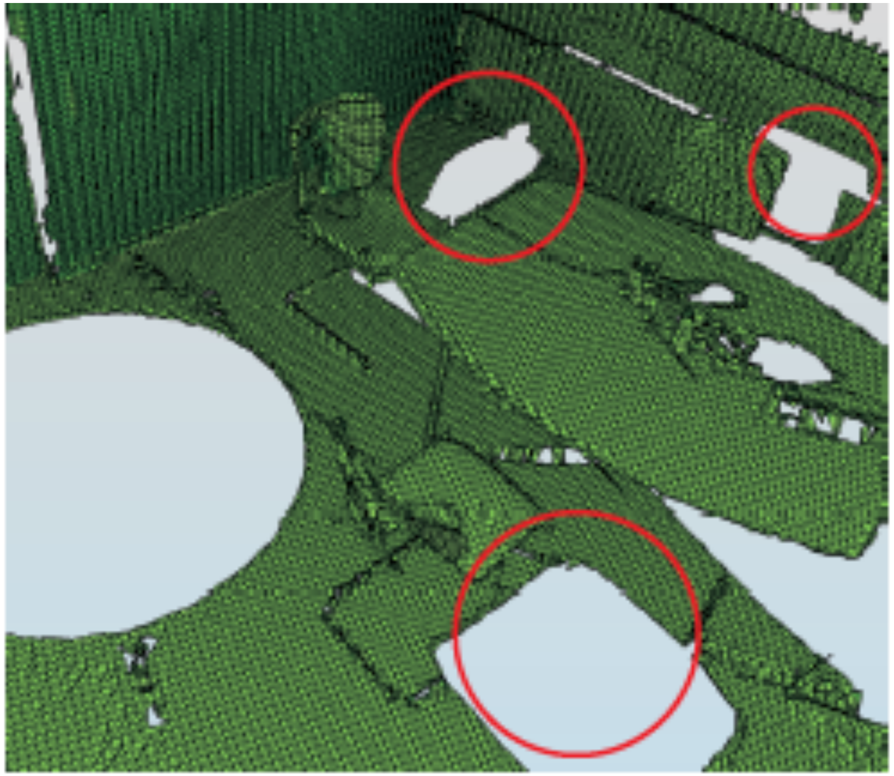
Finding the Edges

- “Planar Marcing Cubes”

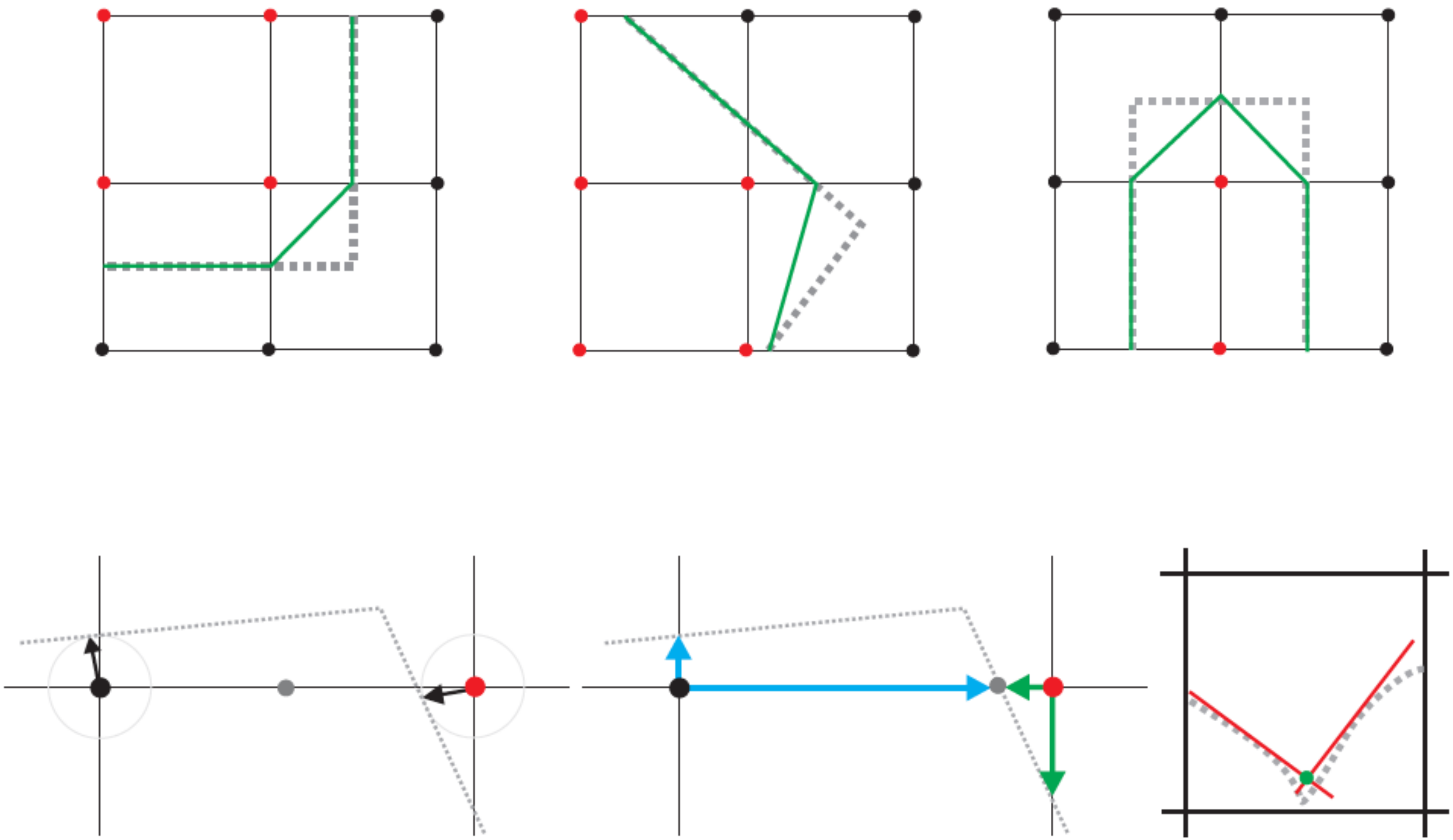


Planar Marching Cubes

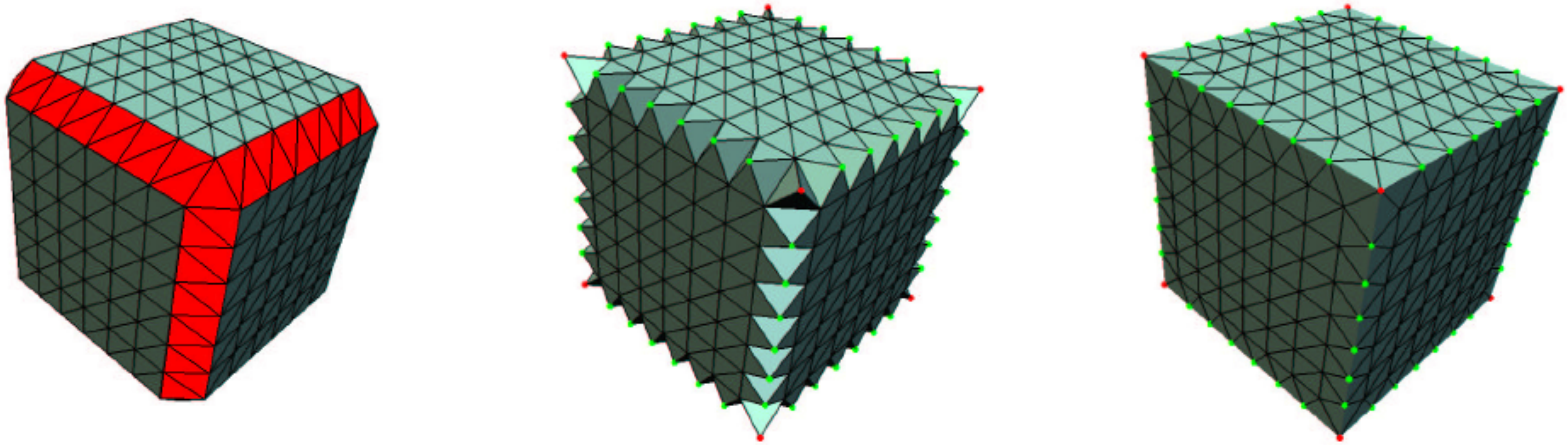


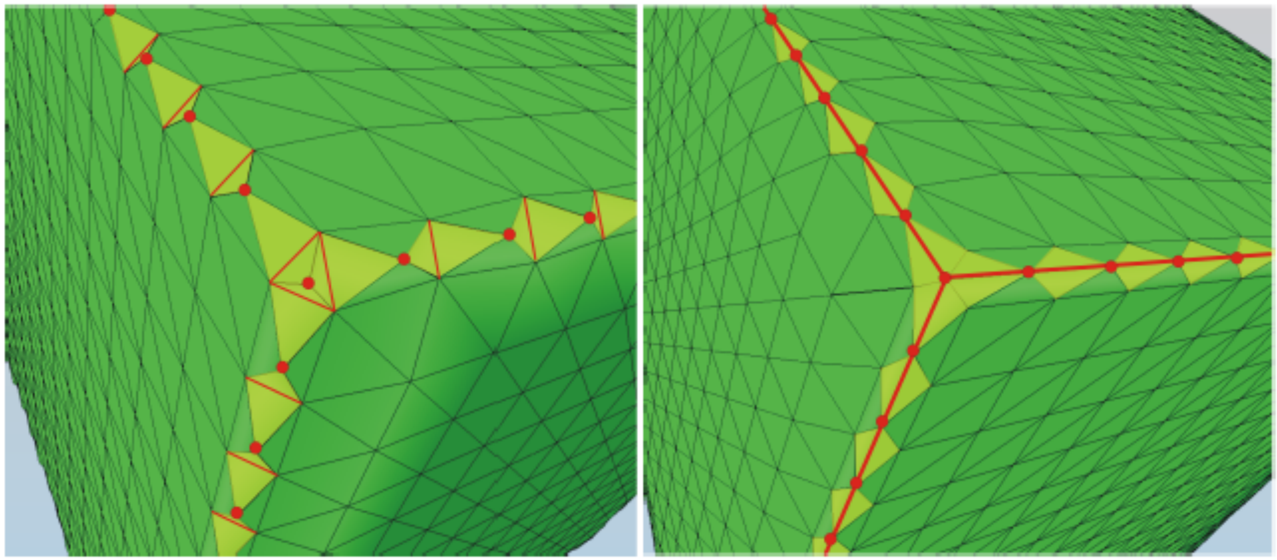


Extended Marching Cubes



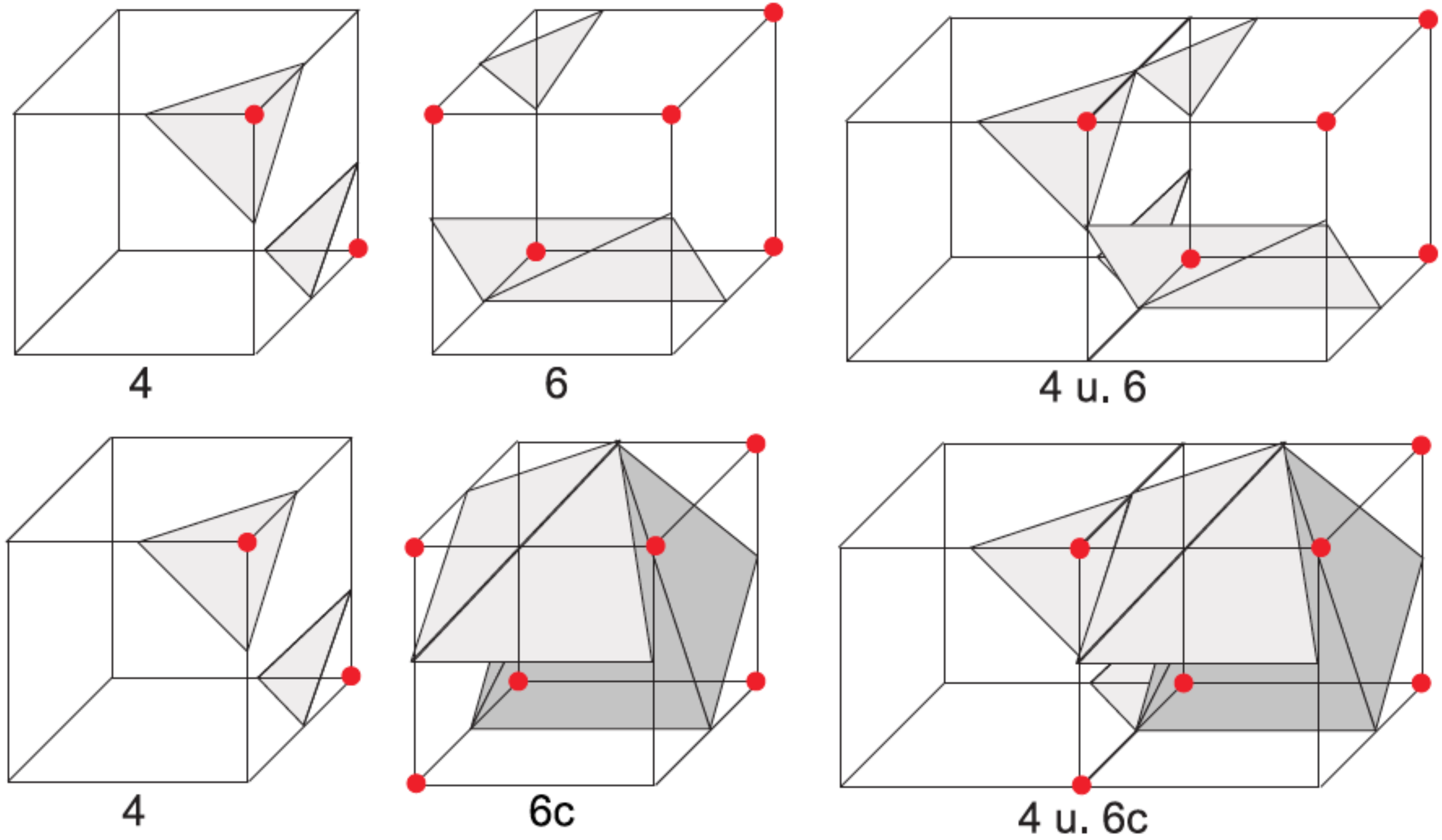
Extended Marching Cubes



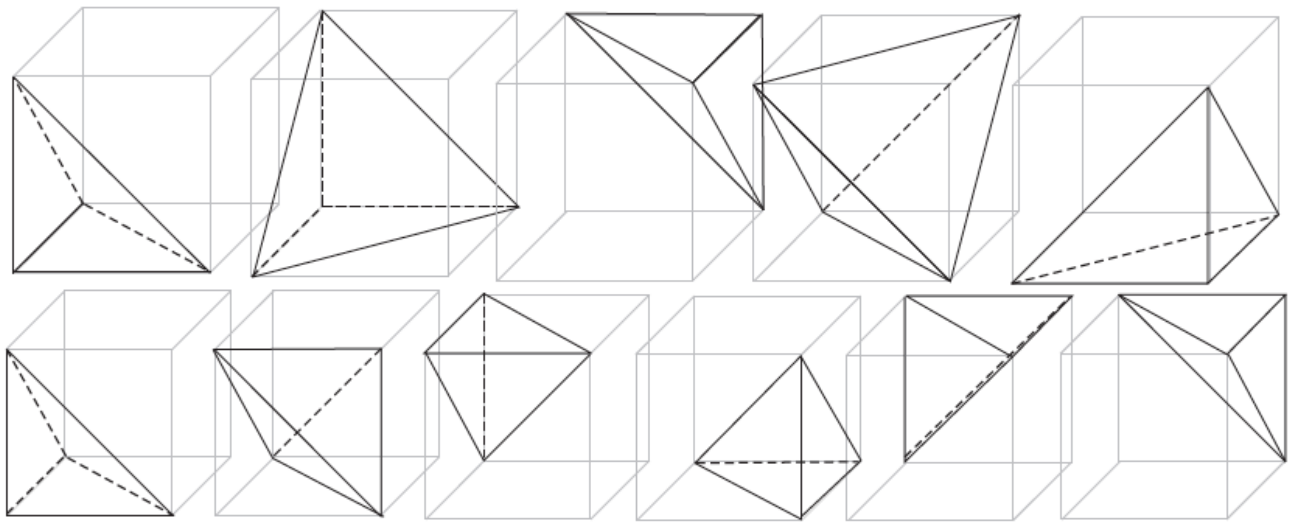


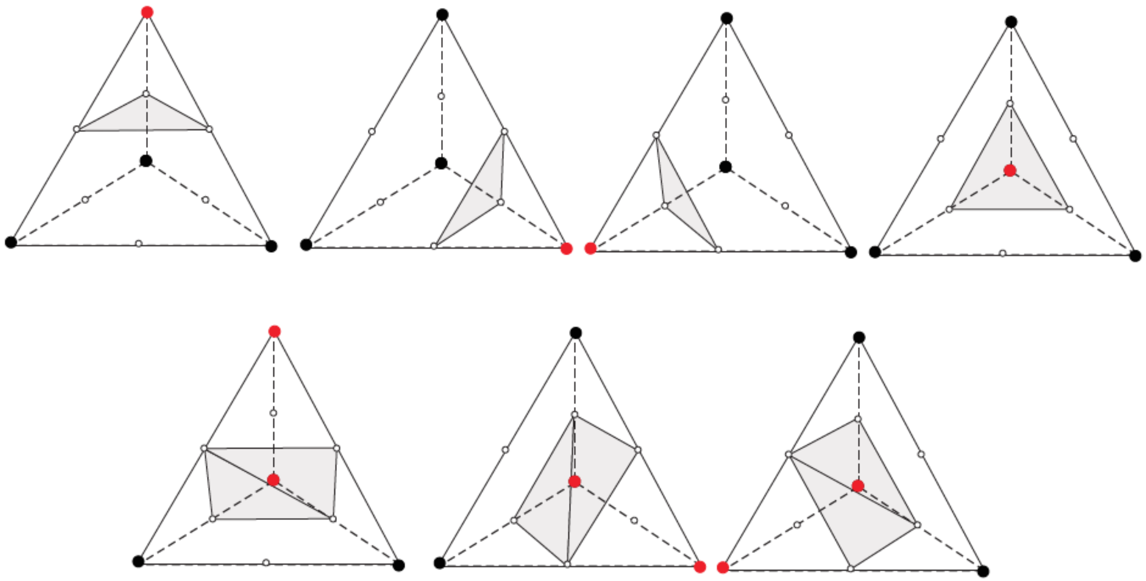
UNIVERSITÄT OSNABRÜCK

Inconsistencies

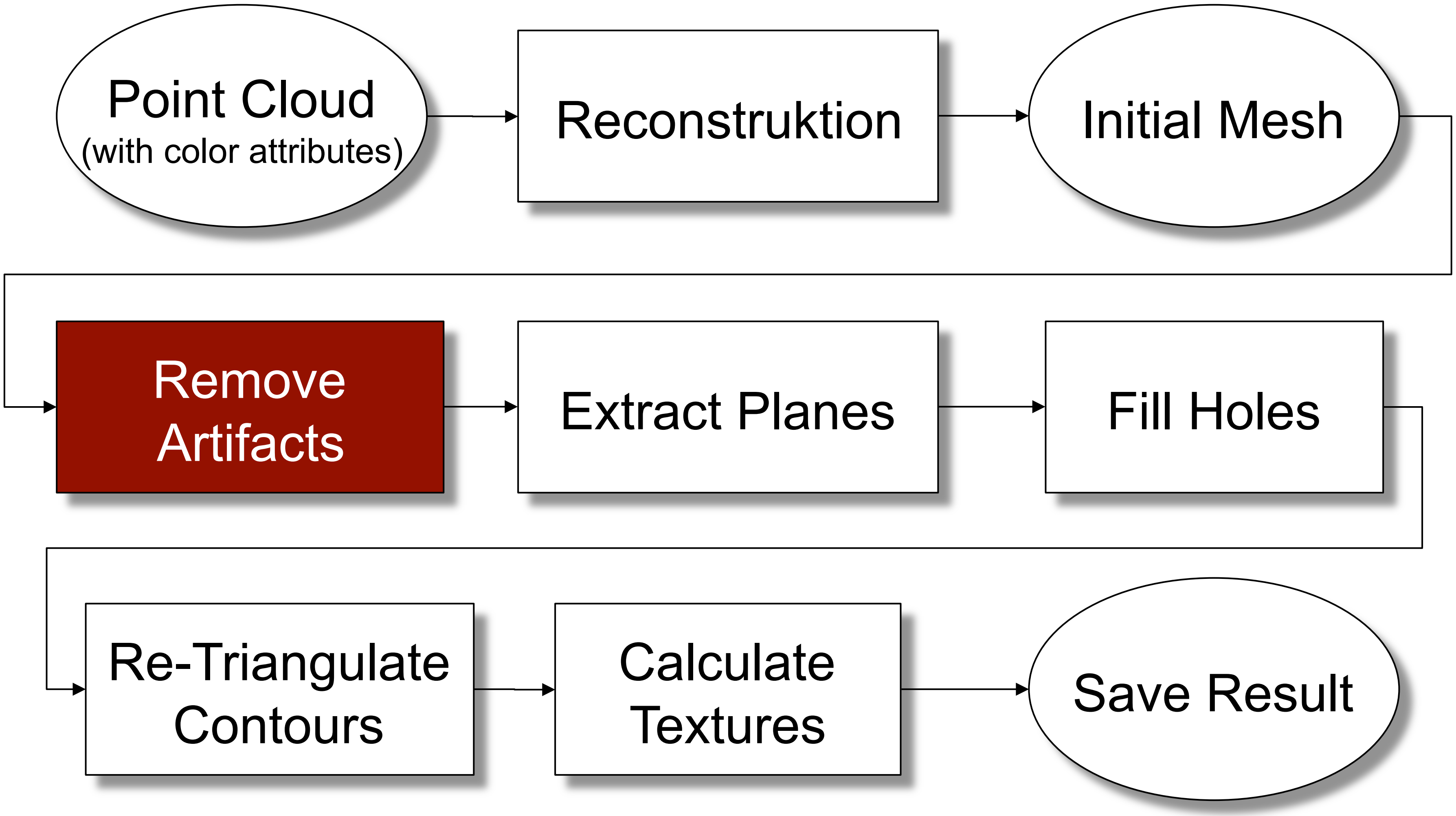


Marching Tetrahedra



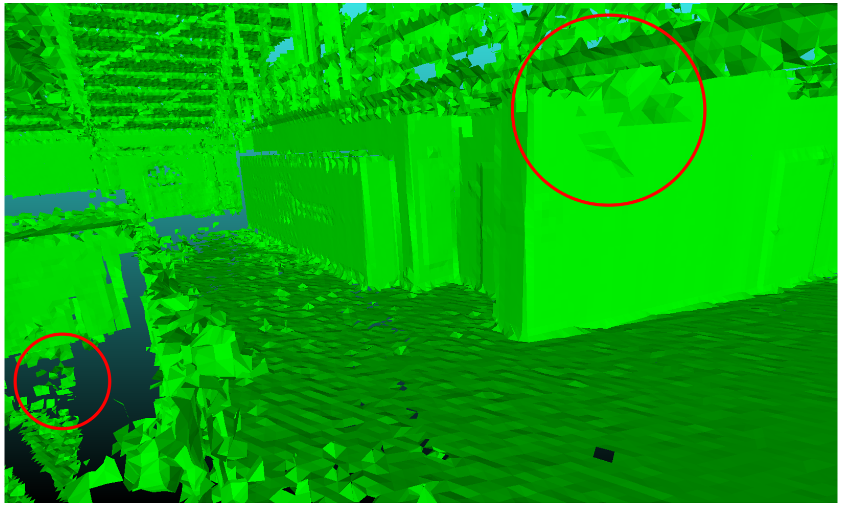


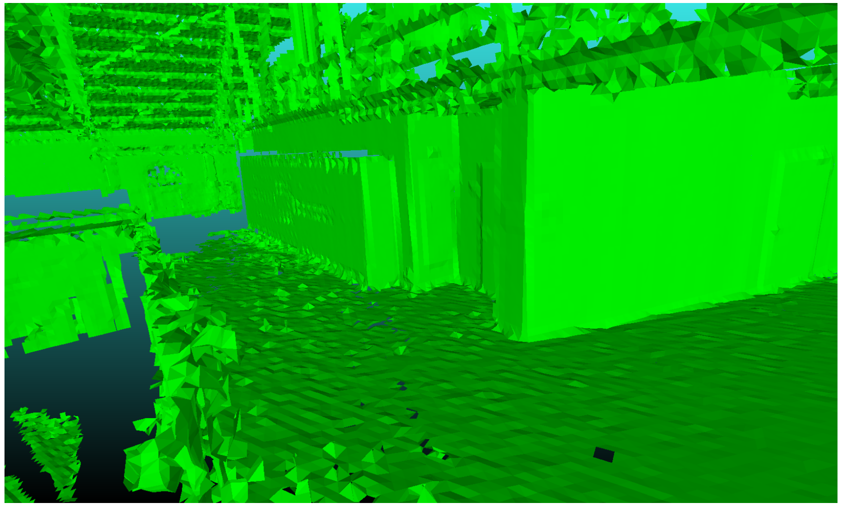
Processing Pipieline



UNIVERSITÄT OSNABRÜCK

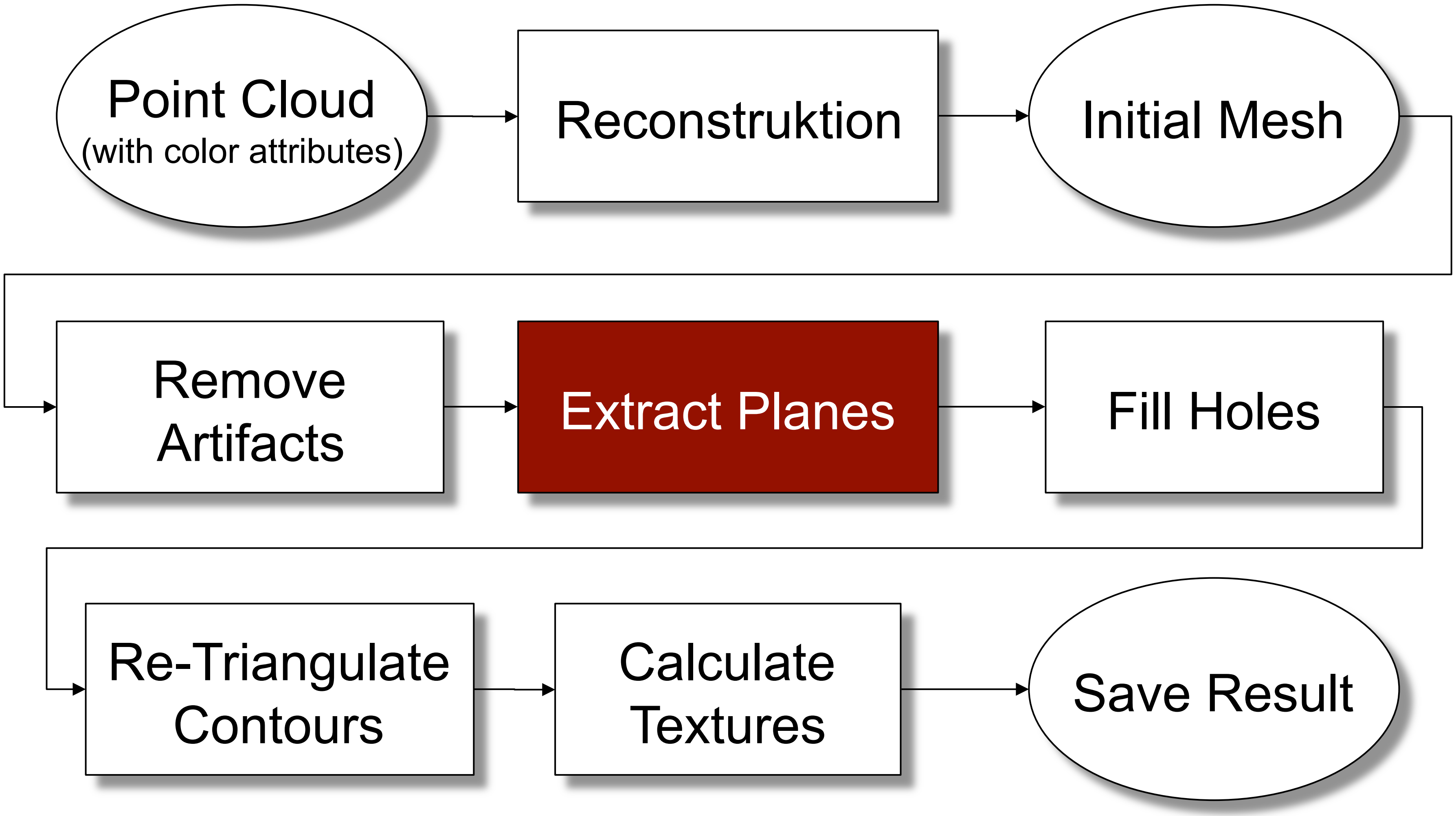
Remove Artifacts



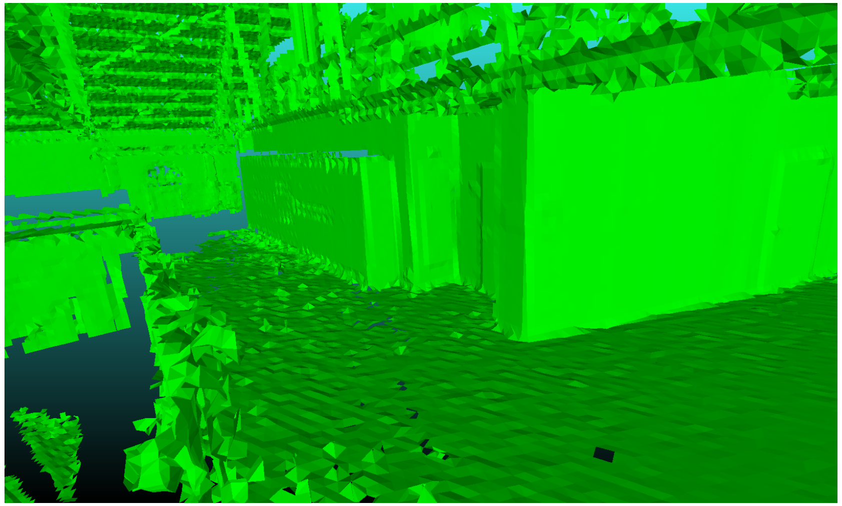


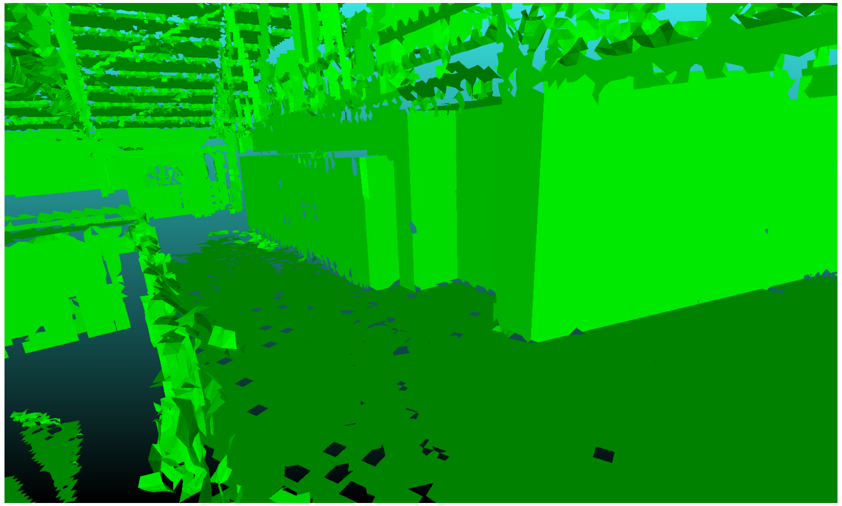
- Remove triangle clusters unconnected to the mesh
- Using recursive region growing
- Heuristic: Number of triangles in cluster
- Done before holes are closed

Processing Pipieline



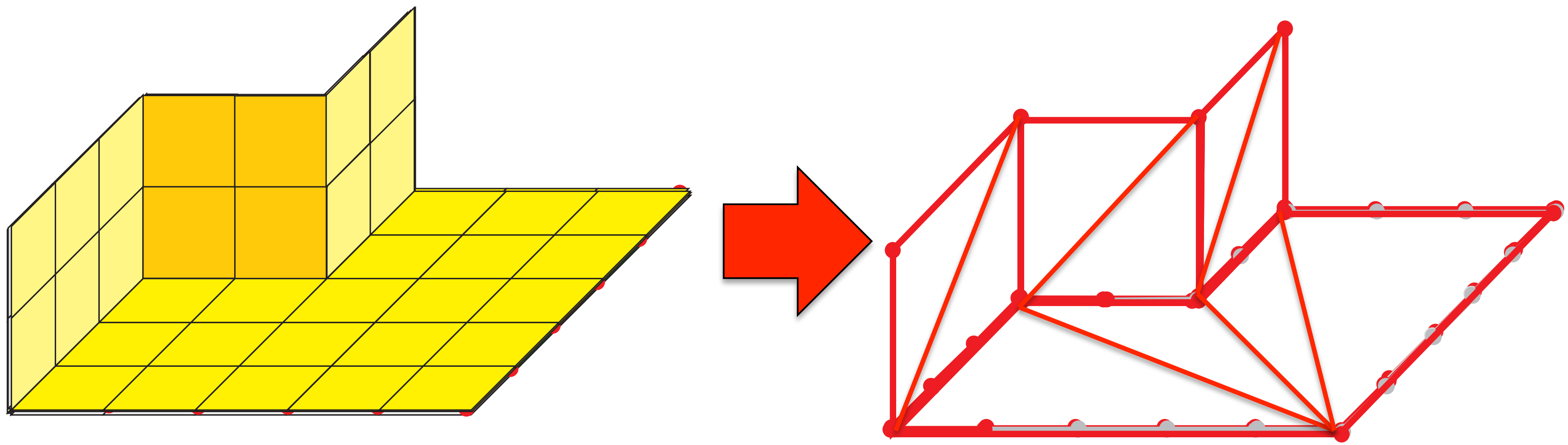
Extract and Optimize Planes





- Detect planes via region growing with normal threshold
- Optimize vertex positions by dragging them into the plane
- Make this iteratively to merge planes that come closer
- Delete small regions that do not belog to planes

Mesh Optimization – Region Growing

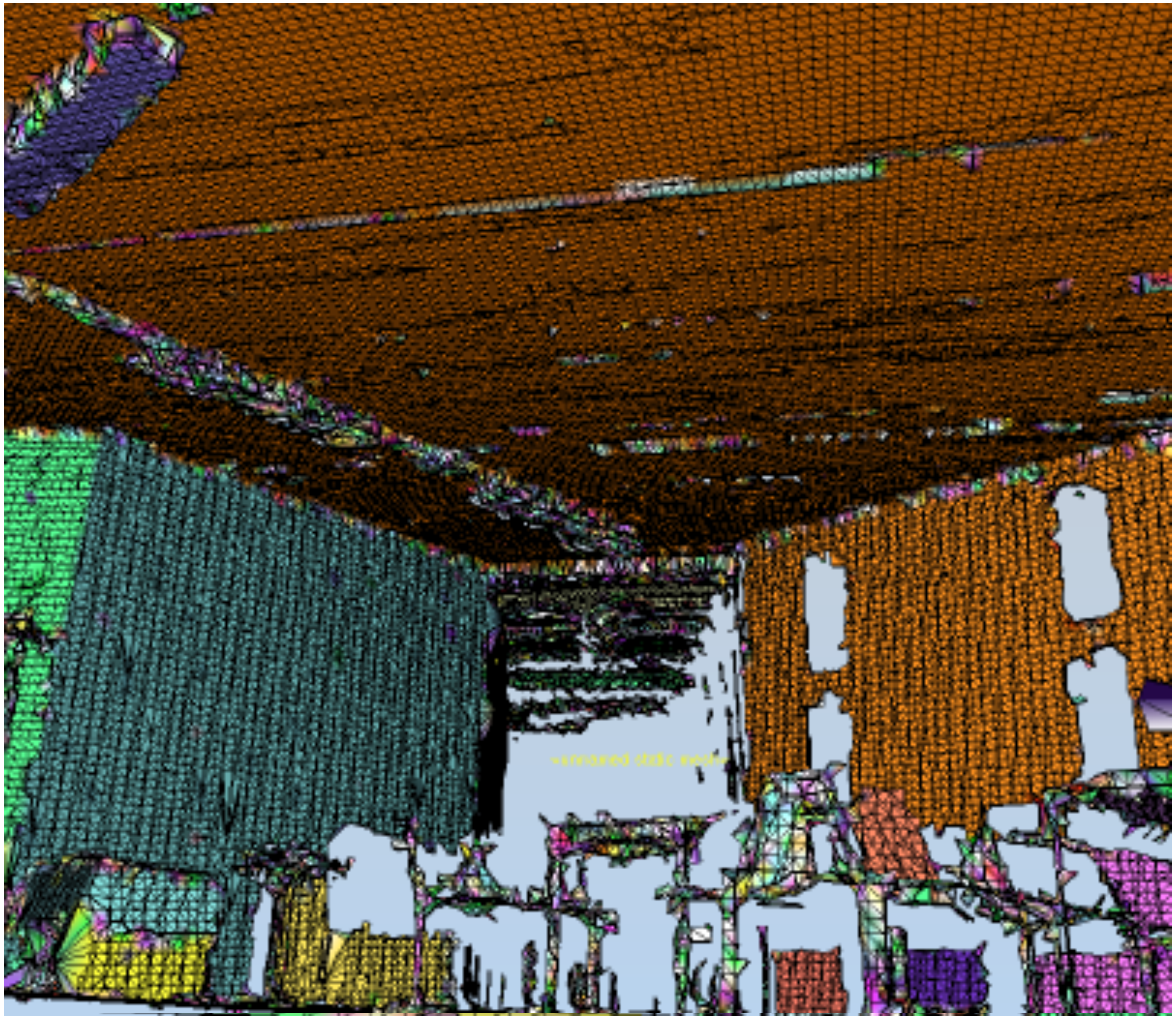


- Every triangle is checked exactly once
- Neighbor edges can be found in *O*(1) time

  ➩ Linear time for polygon extraction

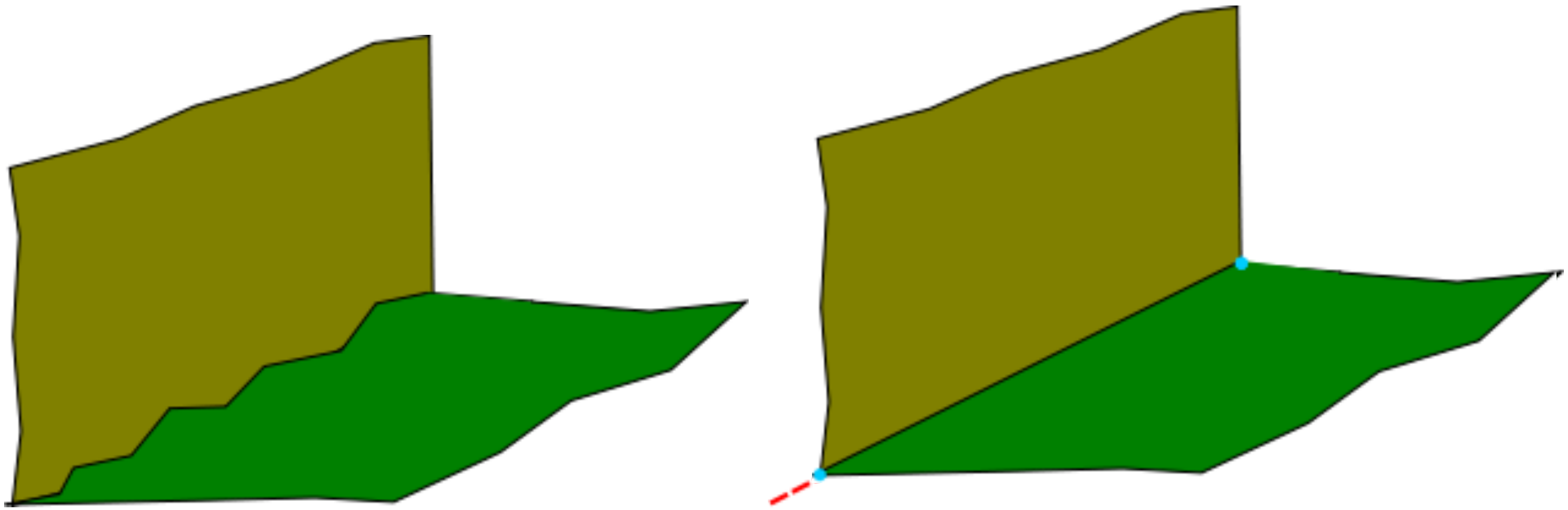
- After all planes have been found: Re-Triangulation

Iterative Plane Optimization



Plane Optimization - Intersections

- Drag all vertices into common plane
- Optimize the intersections of planar regions
  - Calculate the exact intersection line
  - Drag affected vertices into the computed straight line
  - Fuse edges that are on the same line to reduce number of segments

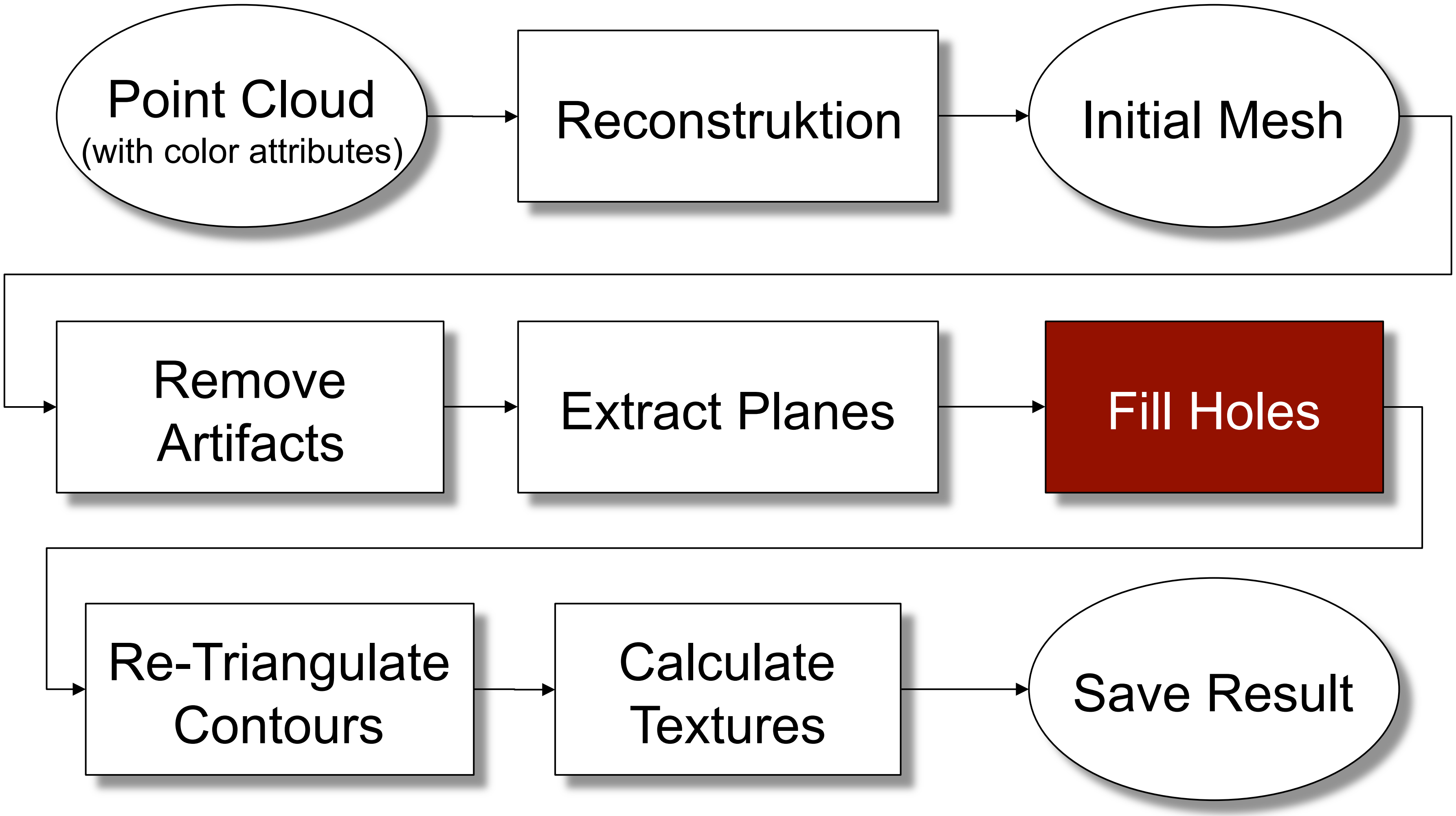


UNIVERSITÄT OSNABRÜCK

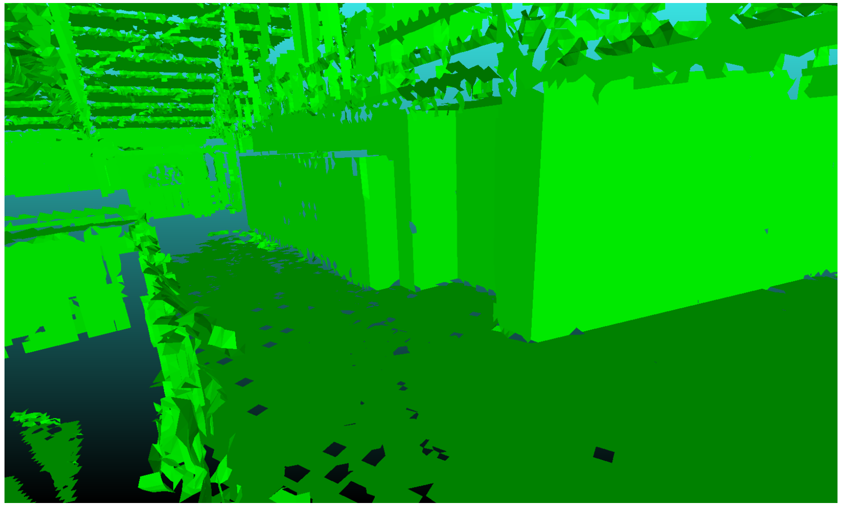
Testing it!

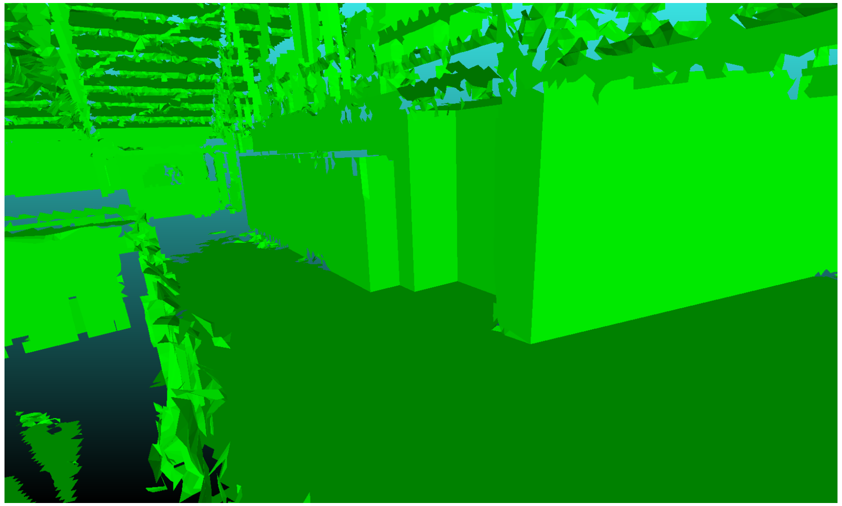
- Relevant parameters:
- `-o --pnt --lft -t`
- “Optimize planes”
- “Plane normal threshold” – Normal criterion
- “Line Fusion Threshold”
- Re-Tesselate
- `--planeIterations`
- Try different parameter sets for yourself on the datasets in `dat/scanxx.3d`

Processing Pipieline



Fill Holes

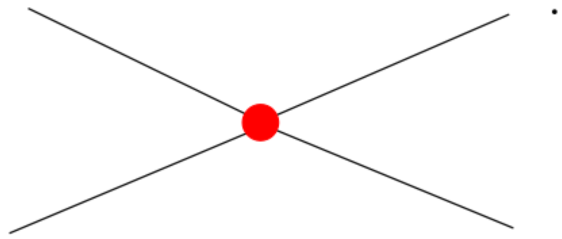




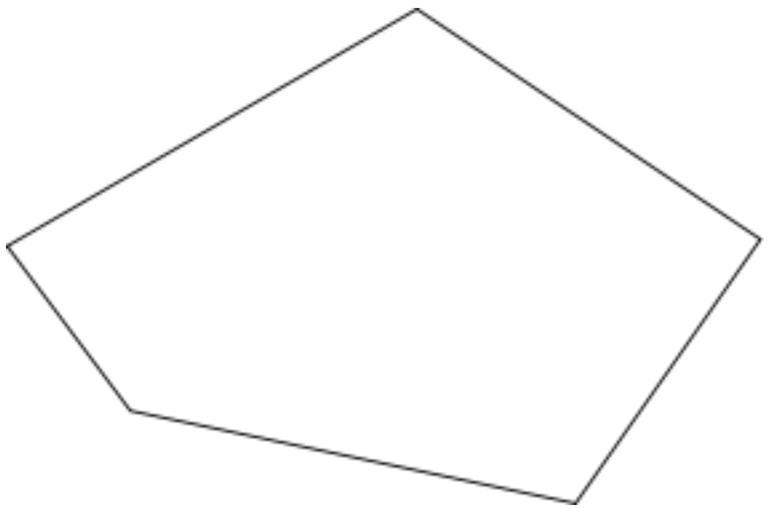
- Trace contours within planes
- Close contours up to a given size
- Number of edges in the hole polygon
- Close by edge collapsing

Hole Filling

- Close holes by edge collapsing

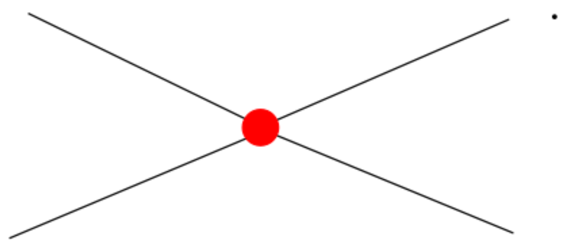


- Example:

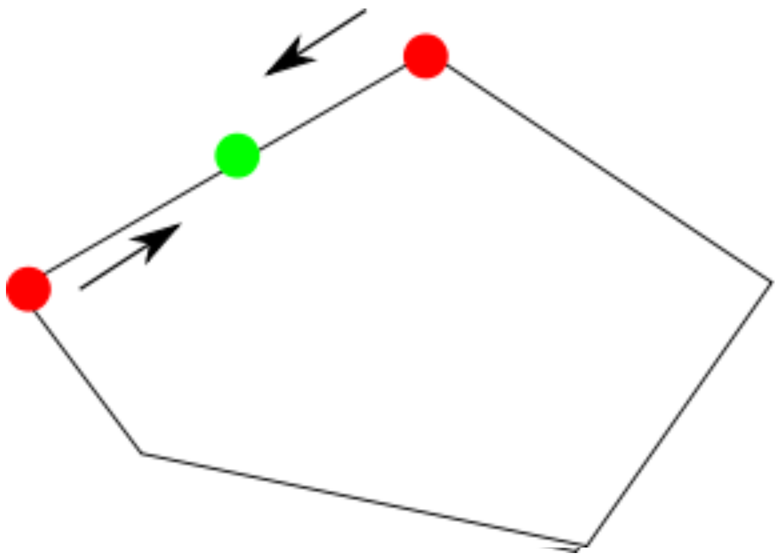


Hole Filling

- Close holes by edge collapsing

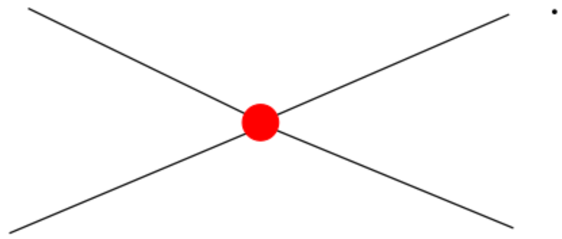


- Example:

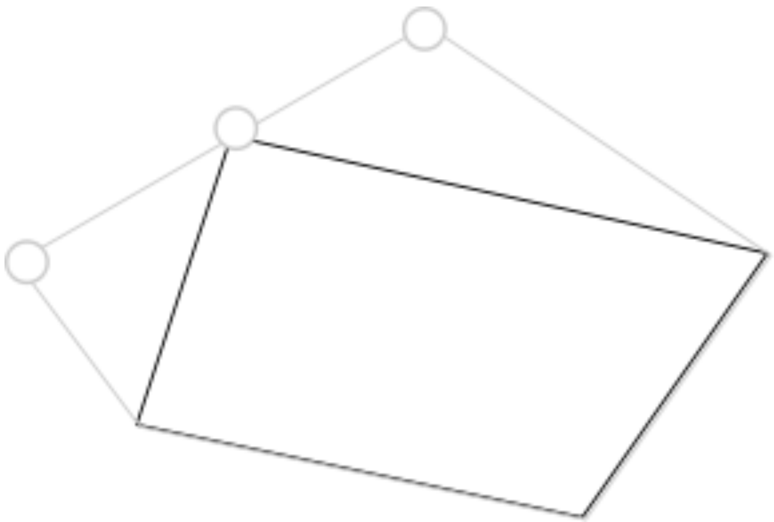


Hole Filling

- Close holes by edge collapsing

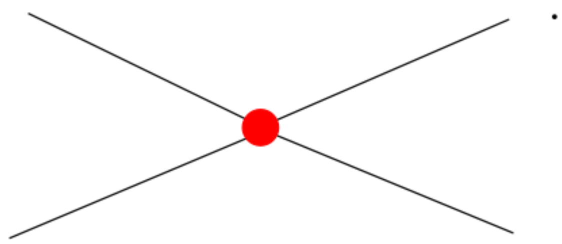


- Example:

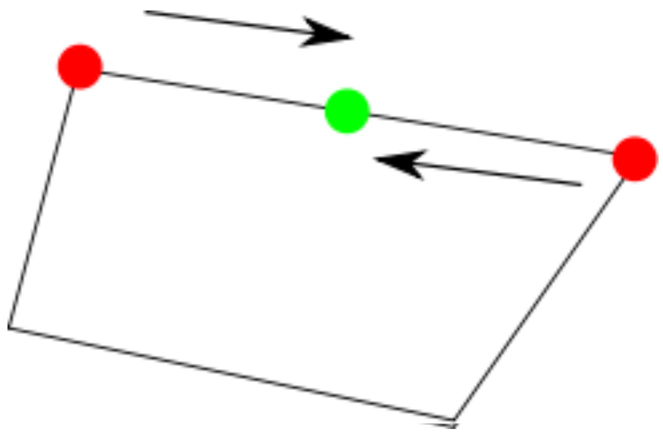


Hole Filling

- Close holes by edge collapsing

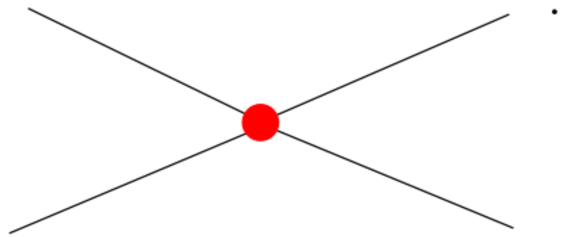


- Example:

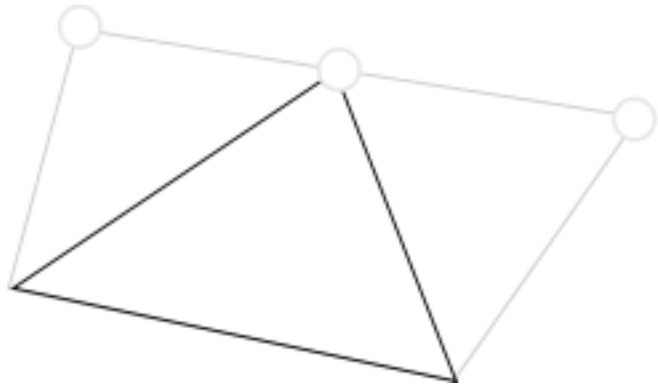


Hole Filling

- Close holes by edge collapsing

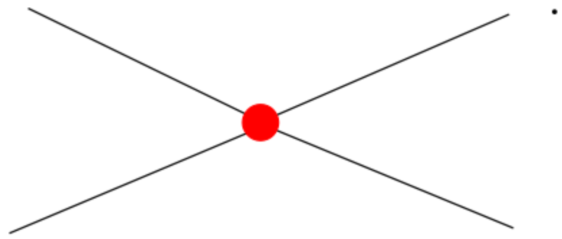


- Example:

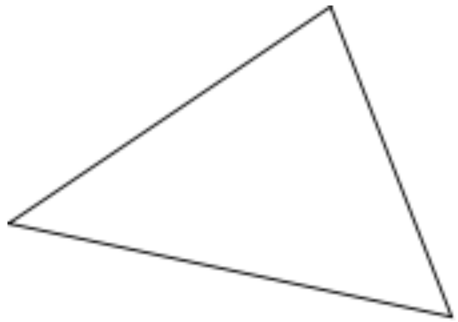


Hole Filling

- Close holes by edge collapsing

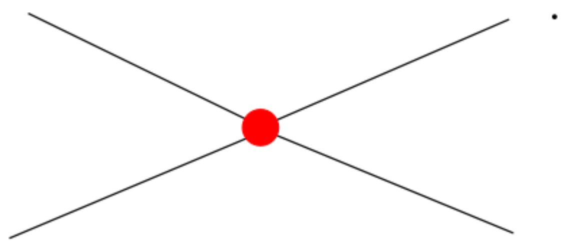


- Example:

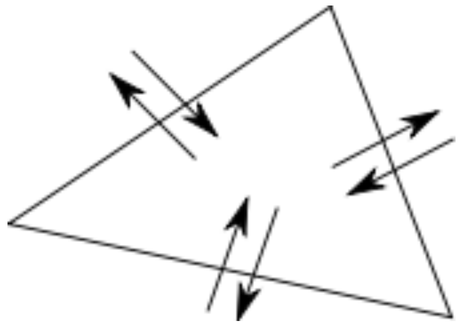


Hole Filling

- Close holes by edge collapsing

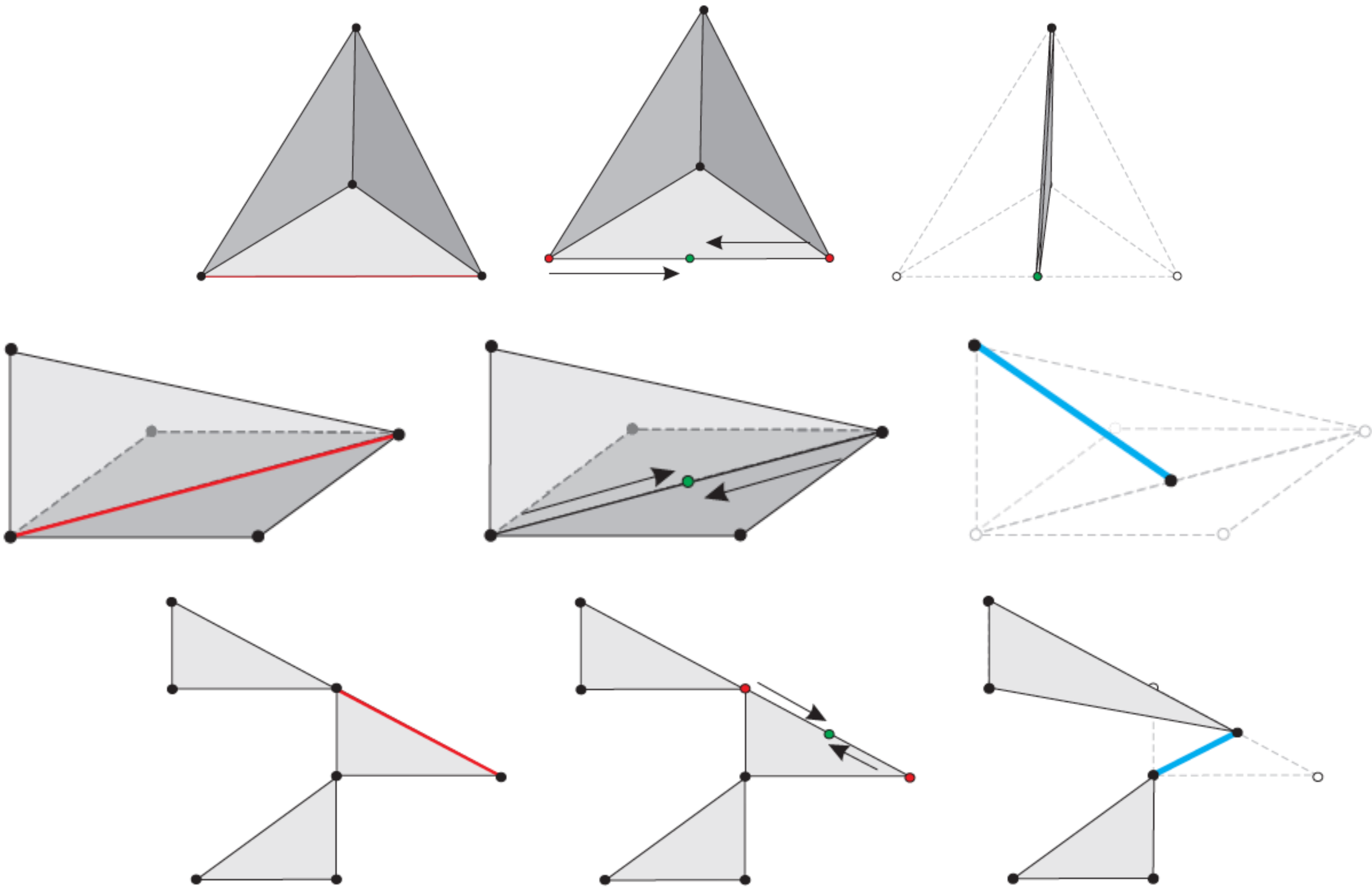


- Example:



Much more complex in the implementation…

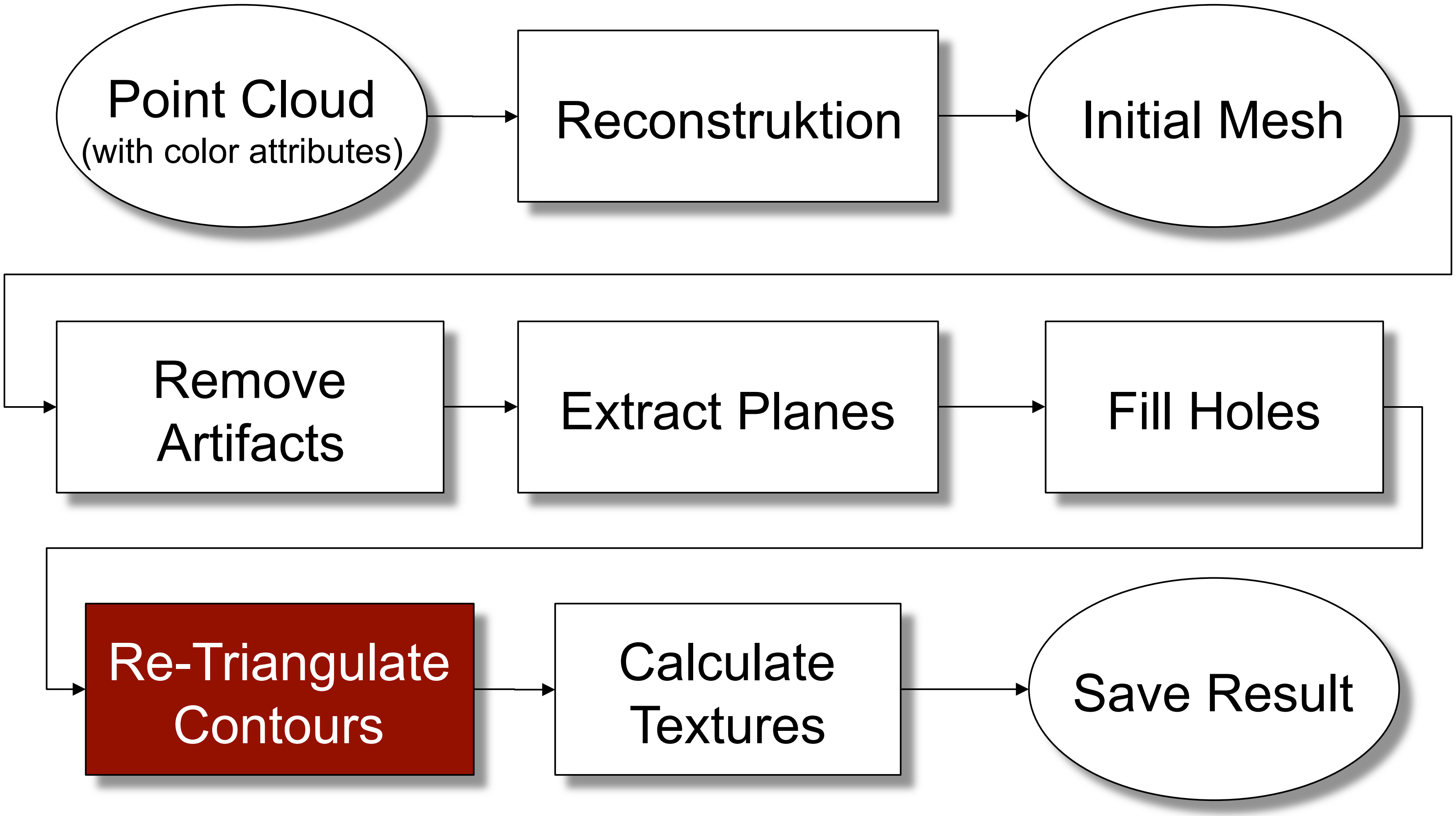
Problematic Topologies



Testing it!

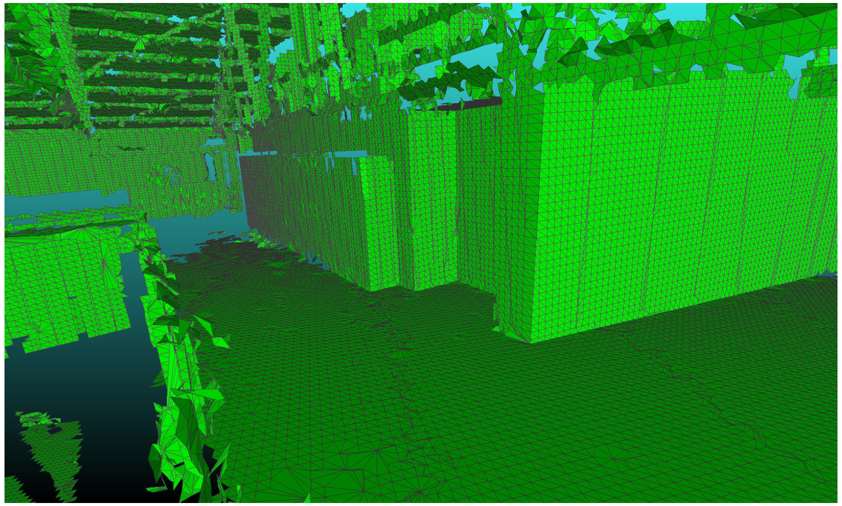
- Relevant parameters:
- `--smallRegionThreshold`
- `--fillHoles`
- Try different parameter sets for yourself on the datasets in `dat/scanxxx.3d`

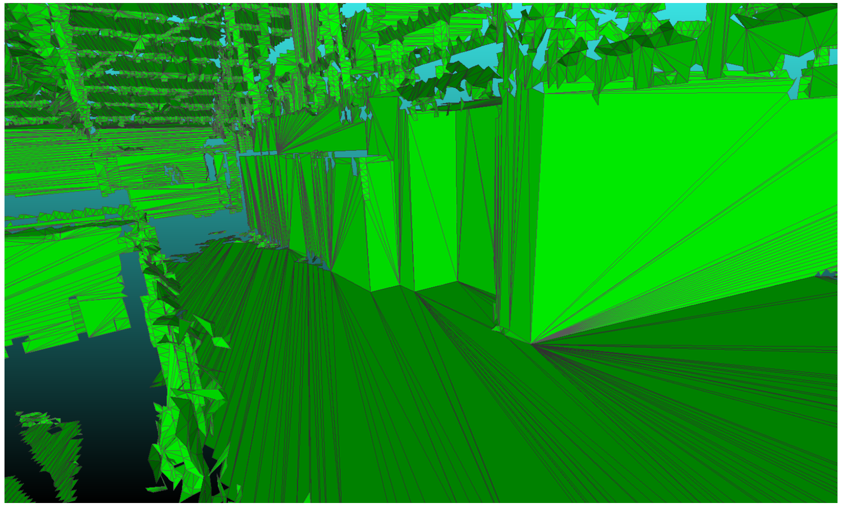
Processing Pipieline



UNIVERSITÄT OSNABRÜCK

Re-Triangulate



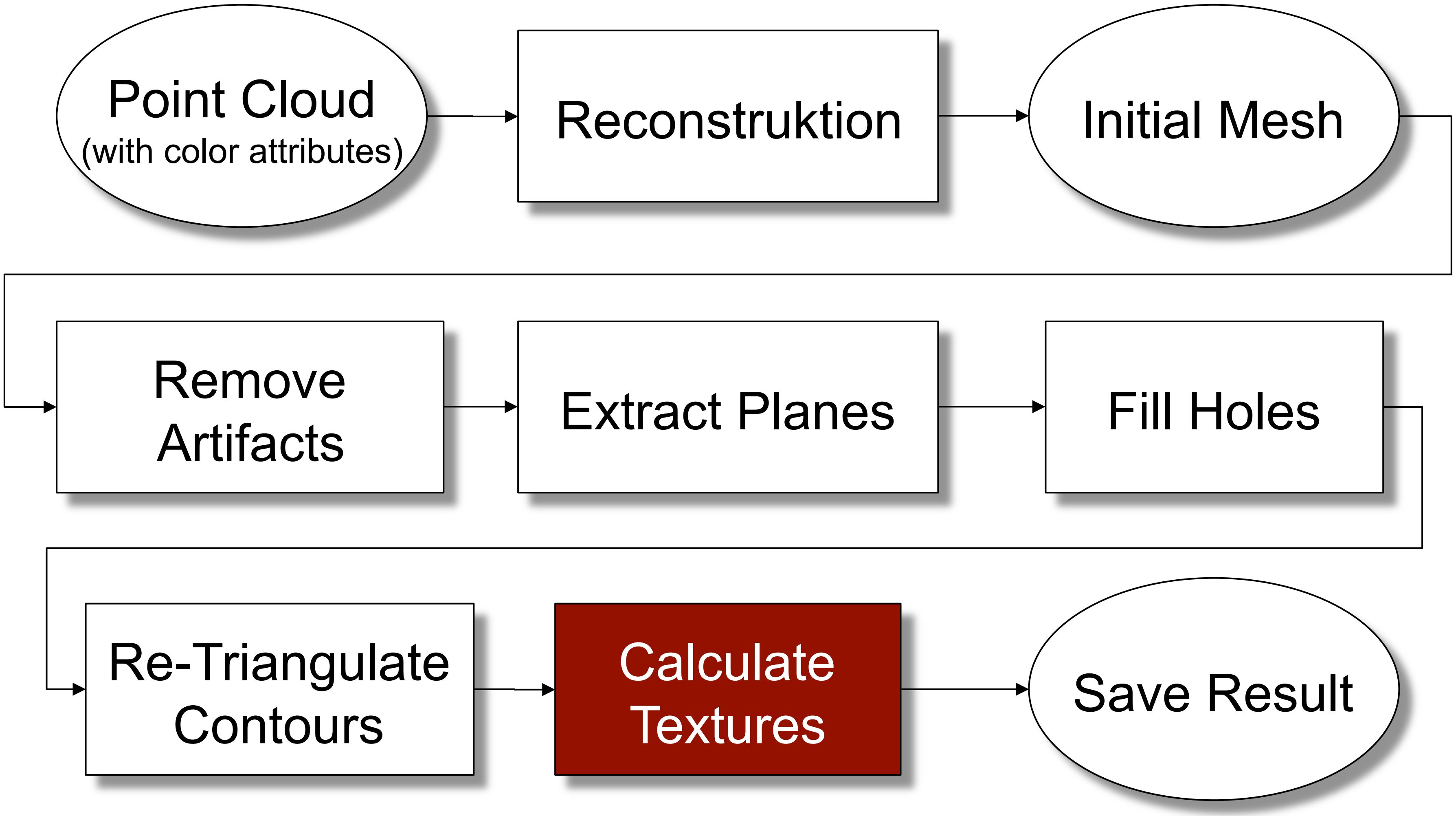


- Generate new triangulation of plane contours
- Use the OpenGL-tesselator
- Usually computed on graphics card
- No change of geometry, but topology

Re-Triangulate

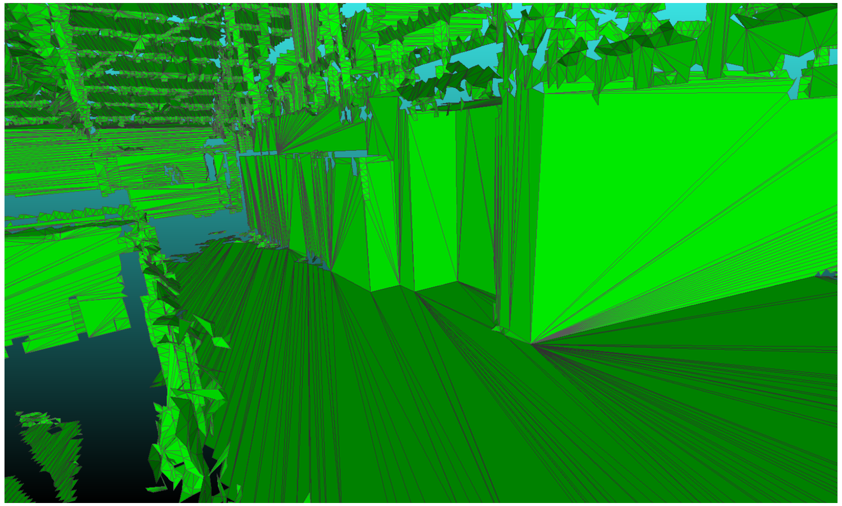
	File Size [MB]	Num Points	Num Faces
Initial Point Cloud	132.5	4,253,689	-
Mesh without Re-Triangulation	10.8	221,443	371,460
Mesh with Re-Triangulation	4.5	119,557	98,648

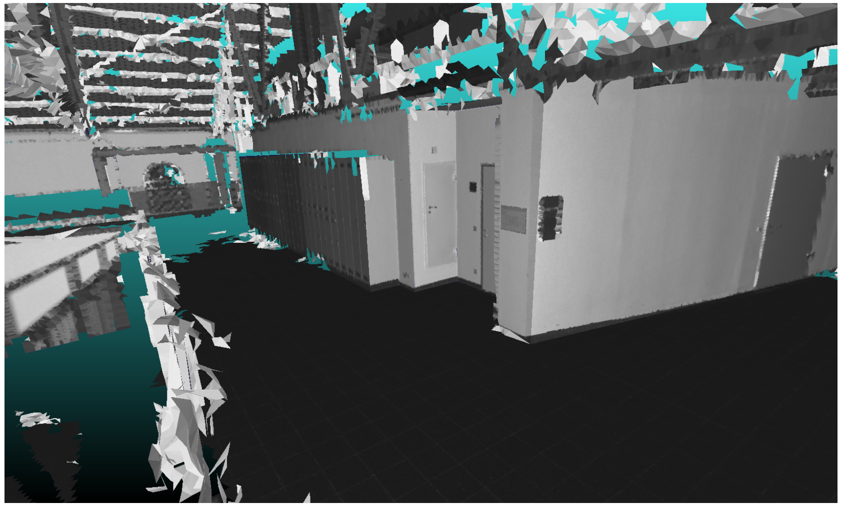
Processing Pipieline



UNIVERSITÄT OSNABRÜCK

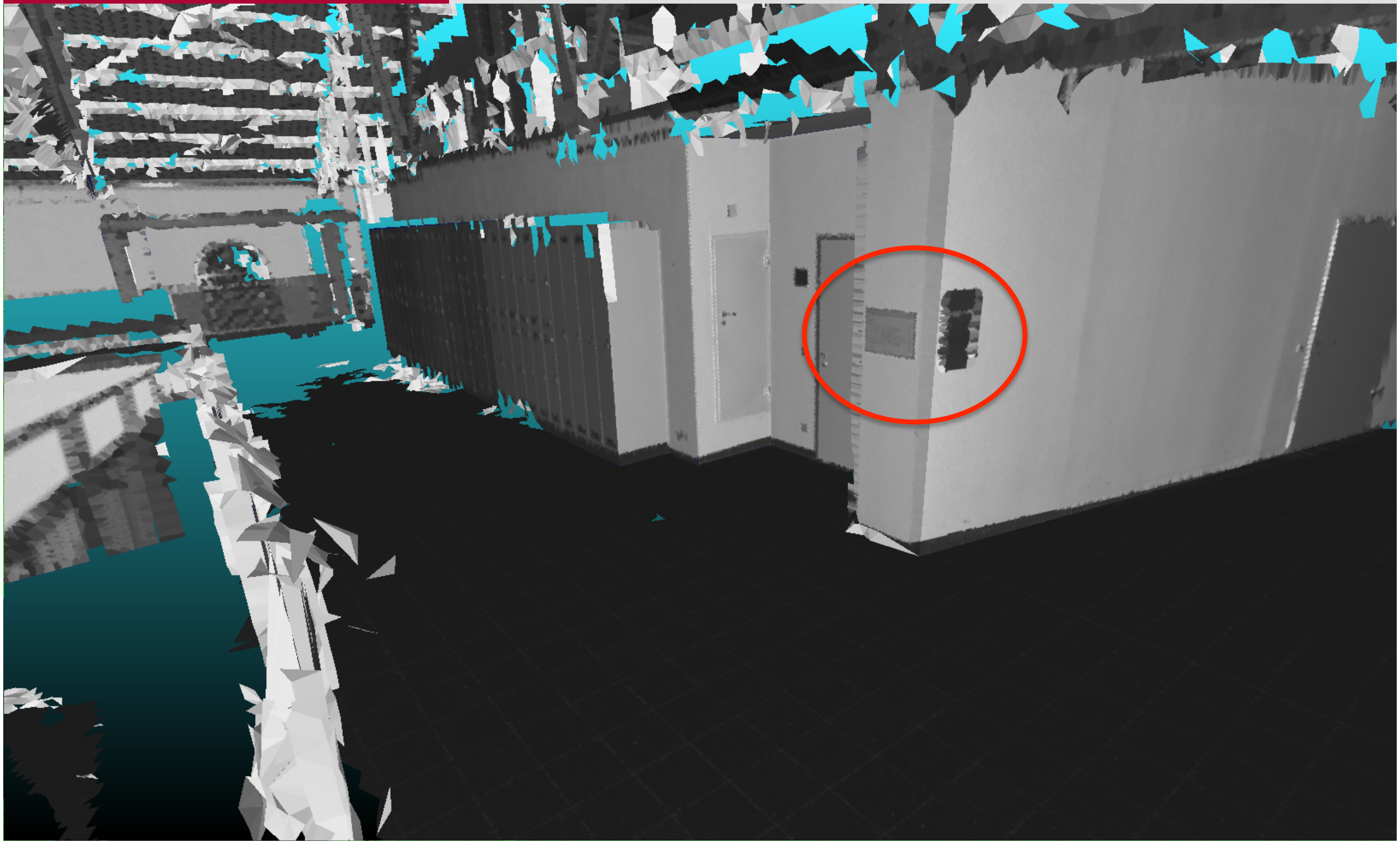
Generate Textures



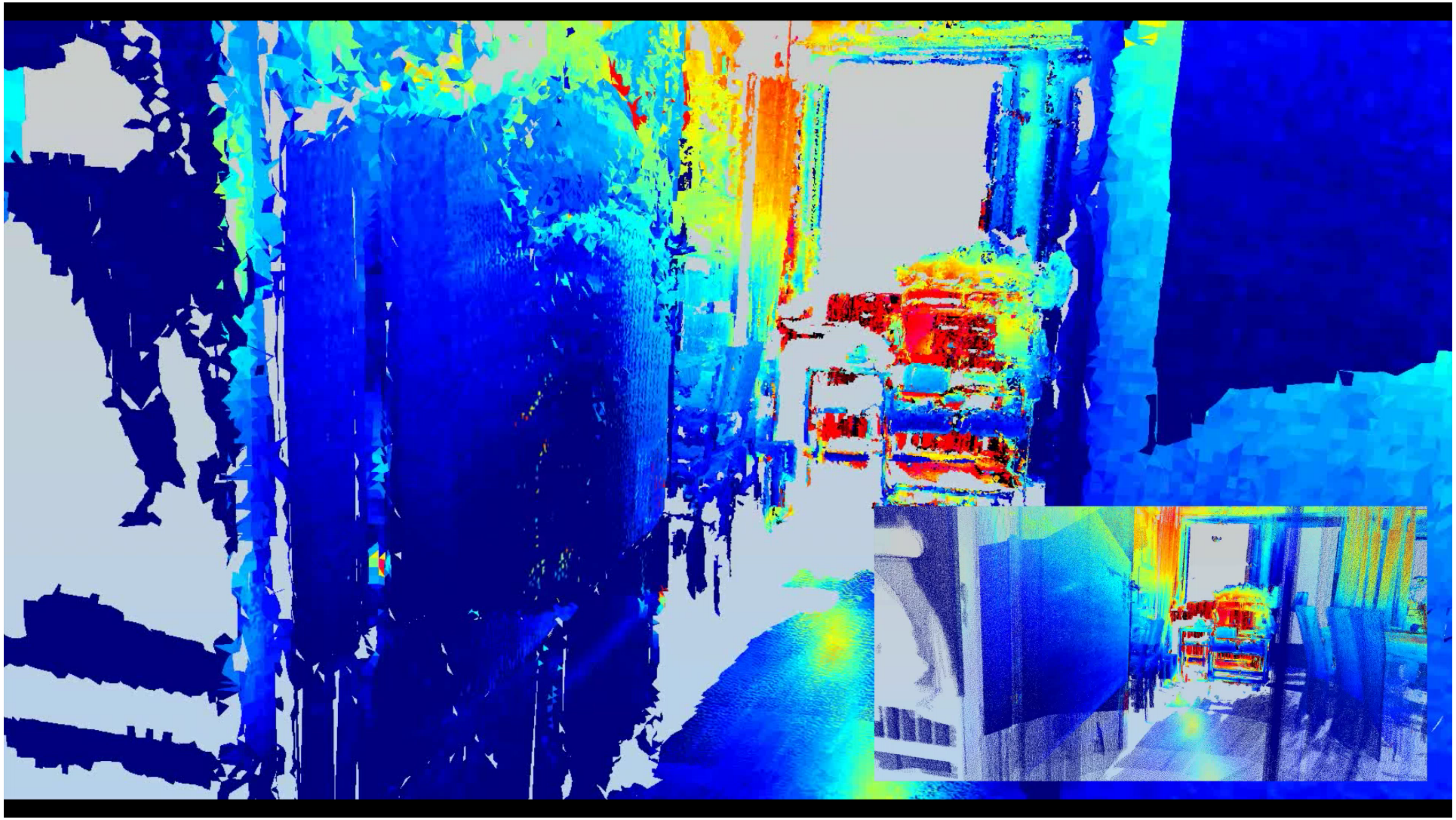


- Every plane can be associated with a bitmap texture
- Small regions are rendered with a suitable color
- Colores are generated from the information in the input clouds
- File format: Wavefront OBJ

Example

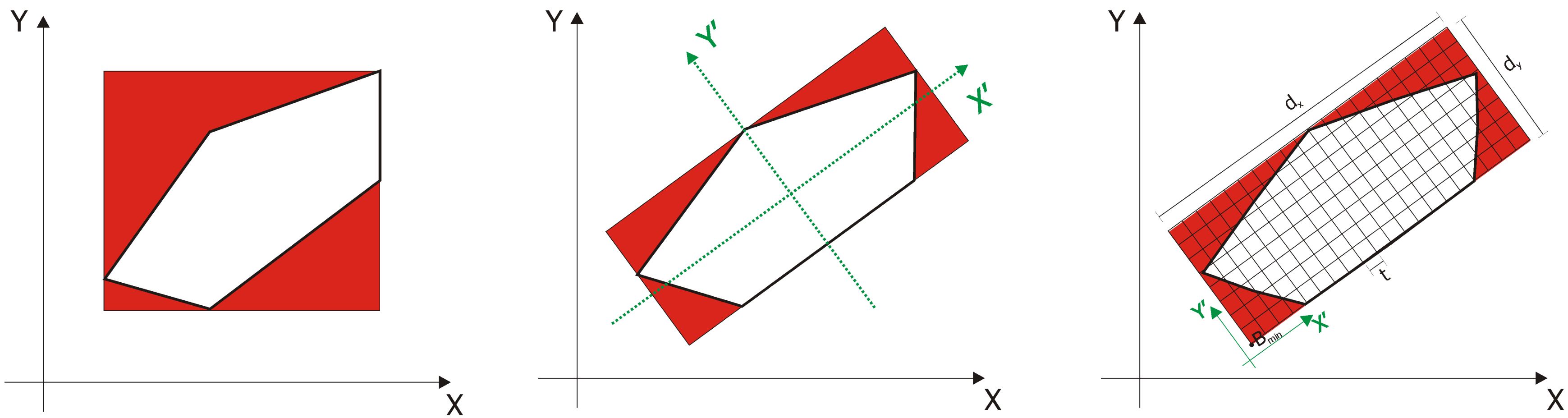


Example: Thermal data



Generate Textures

- "Inverse Texture Mapping":
  - Put a pixelmap map over polygon
  - For each pixel: Search nearest points in data set
  - Color pixel according to input data
  - Color non-plane triangles with single color

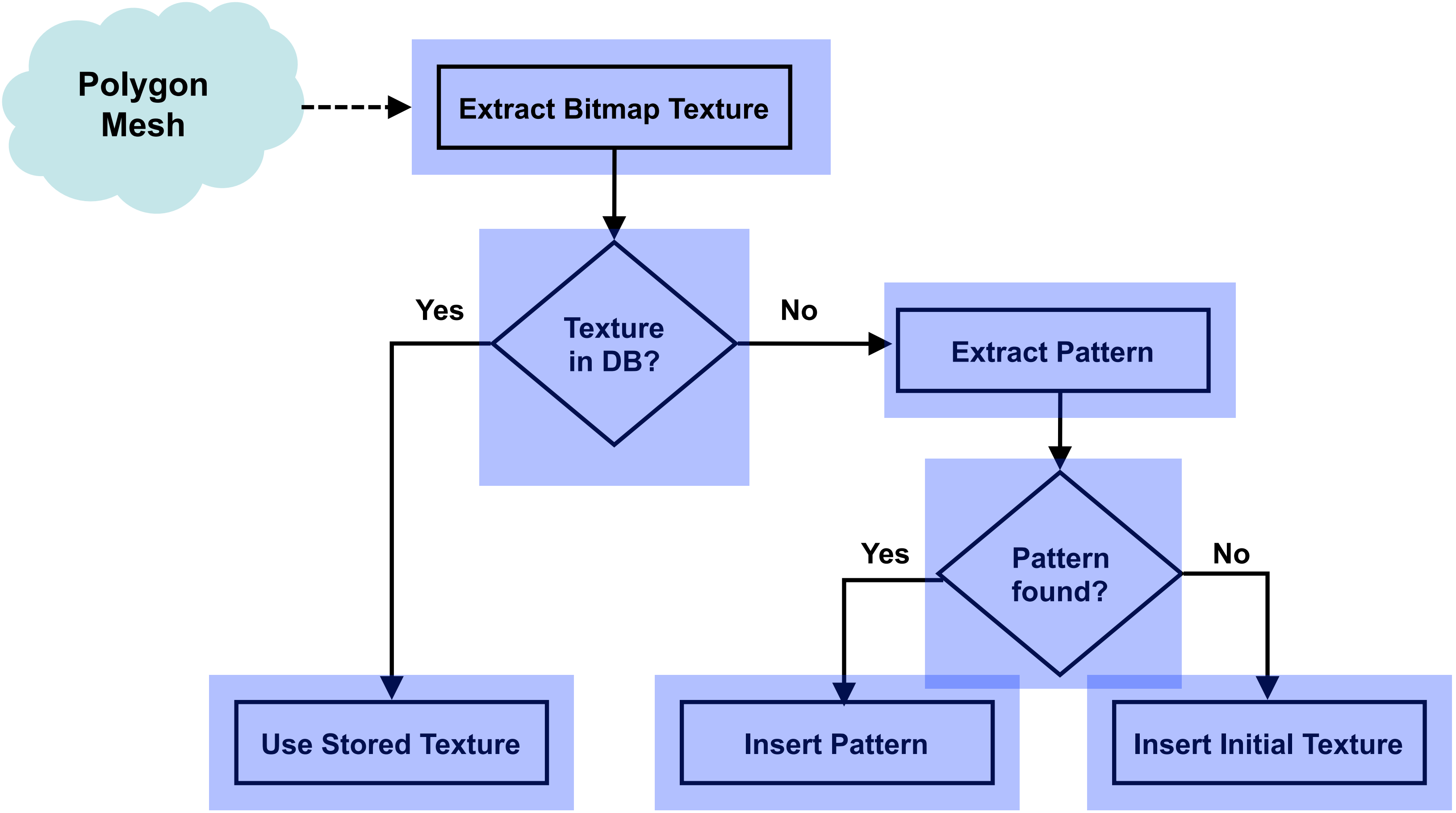


Testing it!

- Relevant parameters:
- `--generateTextures –texelSize`
- Hmm, OK ;-)
- Size of pixels
- Depending on the scale of your input data
- Try different parameter sets on `dat/`
- Start with

```
bin/lvr_reconstruct dat/horncolor.ply –v 20 -o
–t --kd 100 --pnt 0.95 --fillHoles 0 --generateTextures --texelSize 5
```

Workflow



Polygon Mesh
Extract Bitmap Texture
Texture in DB?
Yes
No
Extract Pattern
Pattern found?
Yes
No
Use Stored Texture
Insert Pattern
Insert Initial Texture

Texture Matching

- Searching for textures in the data base:
  - **Color Coherence Matching (CCM)**
    fast, very low rate of false negatives,
    but high rate of false positives
  - **Cross correlation**
    fast in Fourier space,
    generally good results, but sensible to threshold setting
  - **Feature based matching**
    best results, but slow

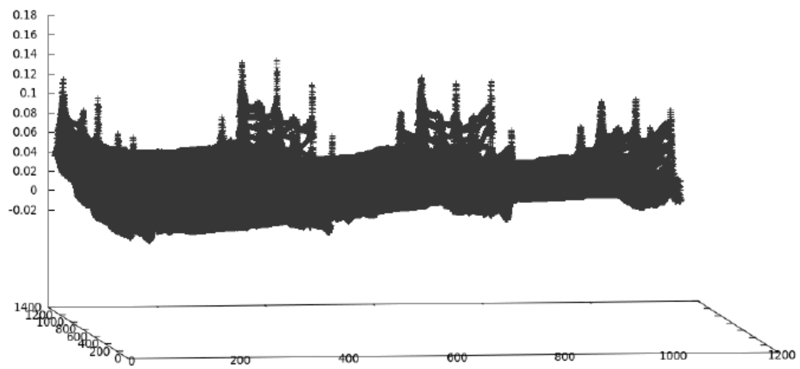
- Approach:
  1. Check with CCM: In case of „no match“, reject.
  2. Otherwise: Combine CC & Features

Pattern Detection

- Check with Cross Correlation if an already detected and archived pattern is present in the current texture bitmap.
- Moving pattern over the current texture:

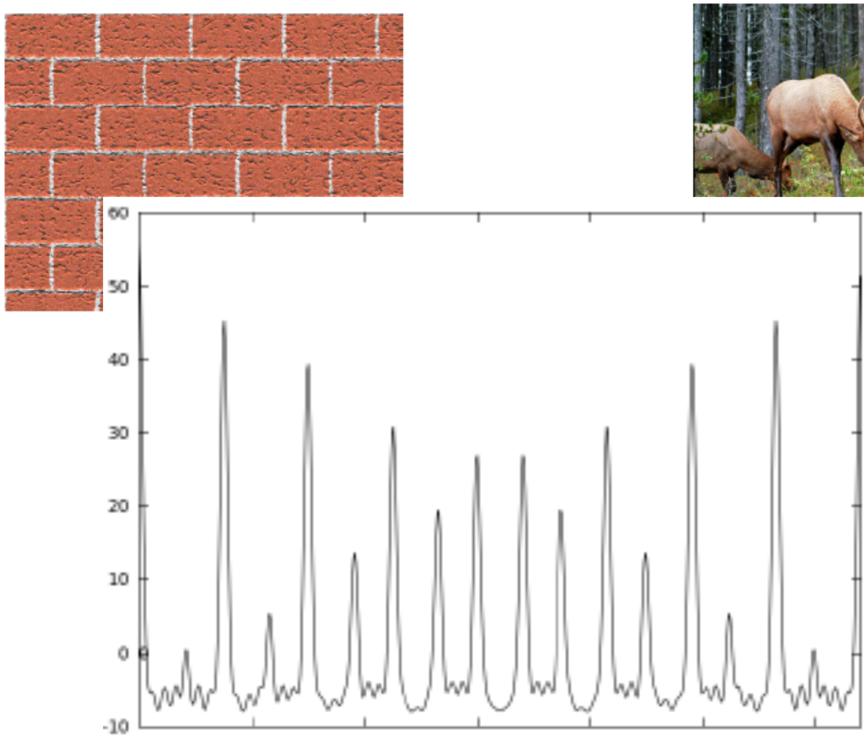
current texture

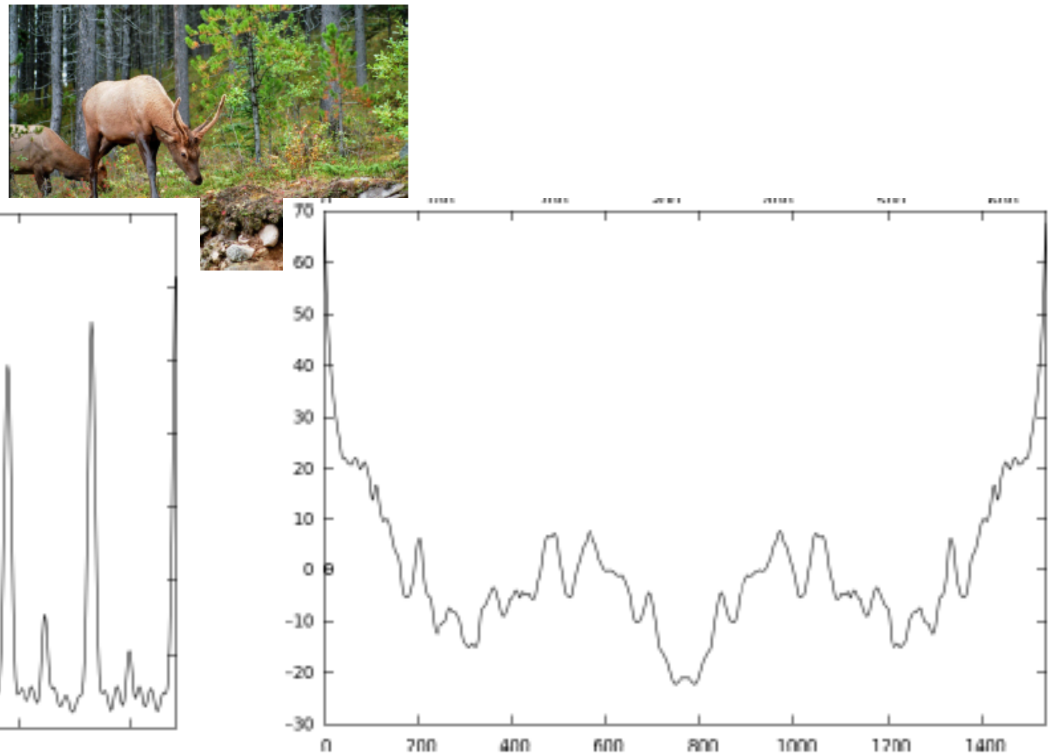
DB



Pattern Extraction

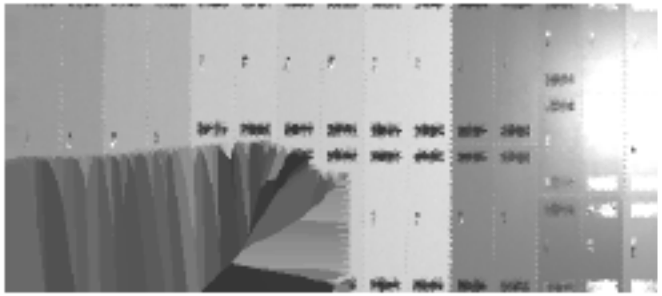
- Does the image contain a pattern?
  If so, where is the optimal cut of that pattern?
- **Pattern check**: auto-correlate the image with itself:

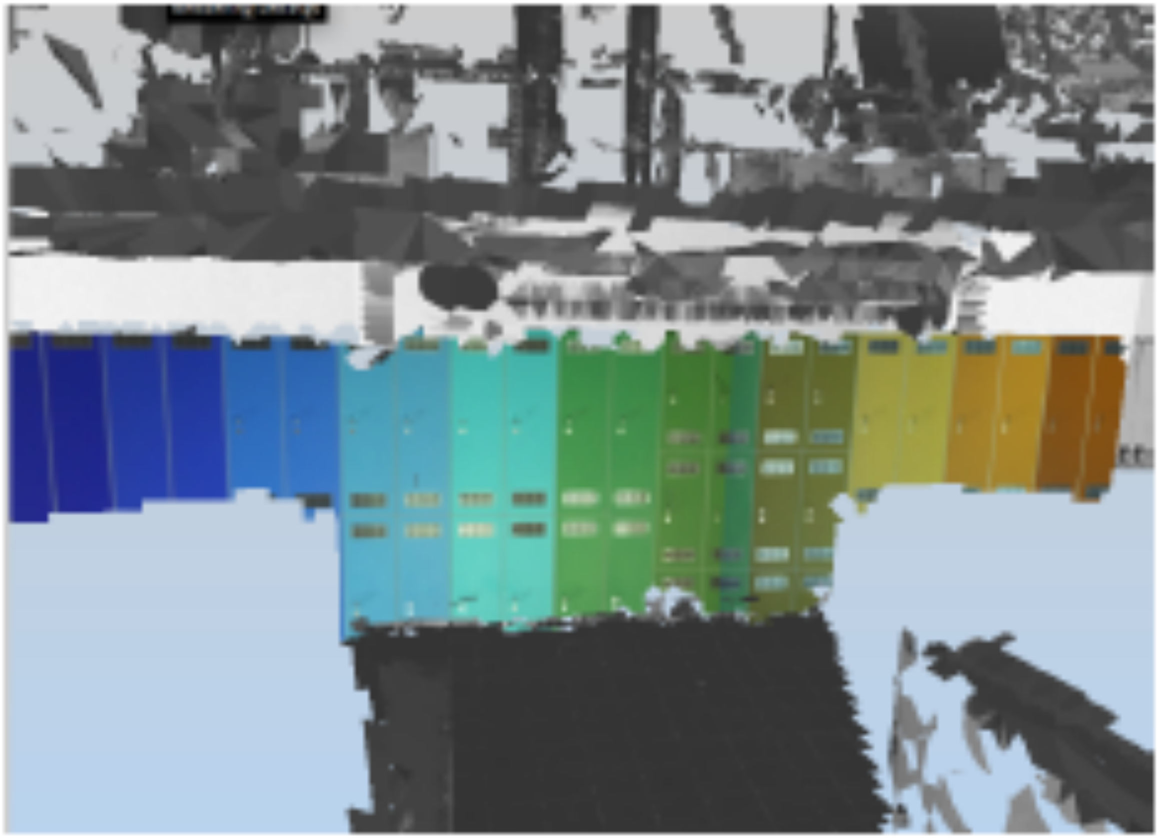




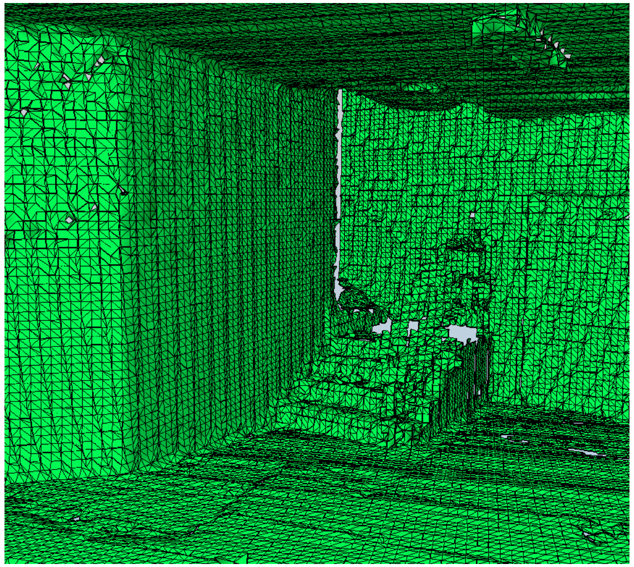
Pattern Extraction

Texture Matching





Other Reconstruction Methods

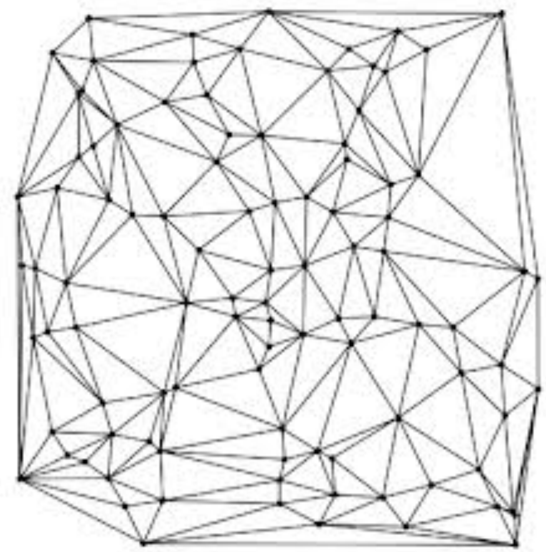


Marching Cubes with Implicit Surface

Signed Distance Function
RIMLS, APSS
Poisson Reconstruction

Marching Tetrahedra, Extended MC,…

Need Point Normal Information



Point Cloud Triangulation

Delaunay Triangulation
Power Crust
Alpha Shapes
Ball Pivoting…

No Normal Information needed!

Poisson Reconstruction

- Reconstruction as Poisson Problem
- Normals are samplings of the derivative of an "Indicator Fuction" χ
- Find global solution to the following Poisson Problem:

$$\Delta\chi \equiv \nabla \cdot \nabla\chi = \nabla \cdot \vec{V}.$$

Available Implementations

	LVR	Meshlab	CGAL	PCL
Marching Cubes	3 Variants*, SDF	RIMLS, APSS	-	Greedy
Alpha Shapes	-	Yes	Yes	-
Poisson Reconstruction	-	Yes	Yes	Yes
Ball Pivoting	-	Yes	-	-
Power Crust*	-	-	-	-

- LVR, CGAL and PCL can be integrated easily into robot control architectures
- Meshlab can be used as server to process files

*) Standard Marching Cubes, Planar Marching Cubes, Extended Marching Cubes, Marching Tetrahedrons

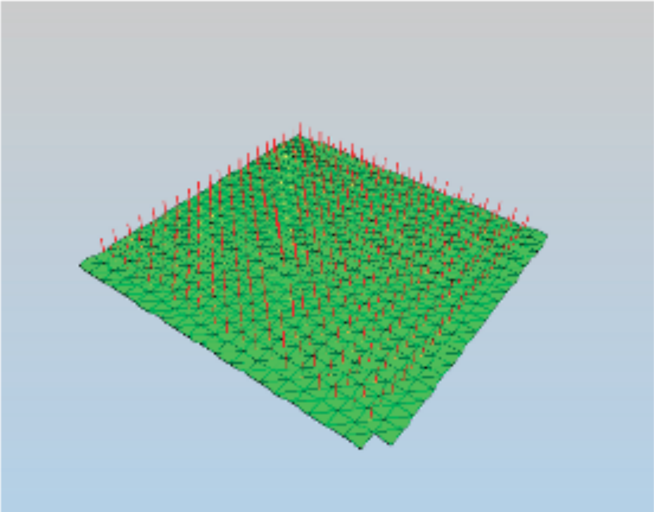
**) Proprietary implementation under GPL

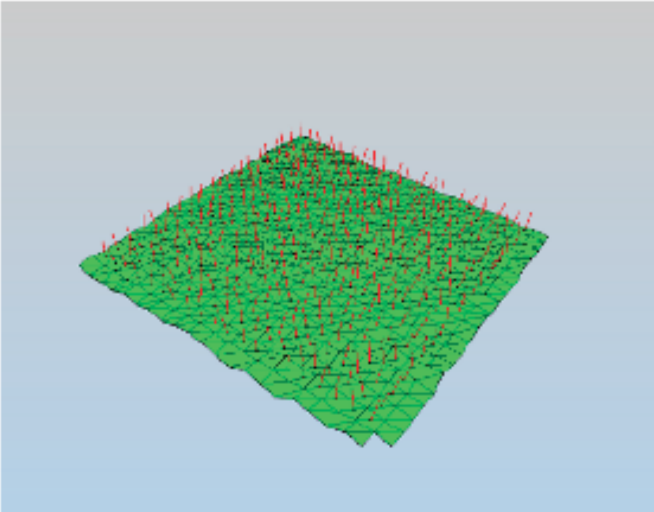
Evaluation Criteria

- Influence of the source geometry / input quality
  - Is the method at hand capable of reconstructing what I want?
  - Closed surfaces vs. open ones?
  - Do I need a closed surface representation
- Runtime
- Geometric Precision
- Topological soundness
  - Degenerated Faces
  - Faulty linkage
  - Isolated vertiex

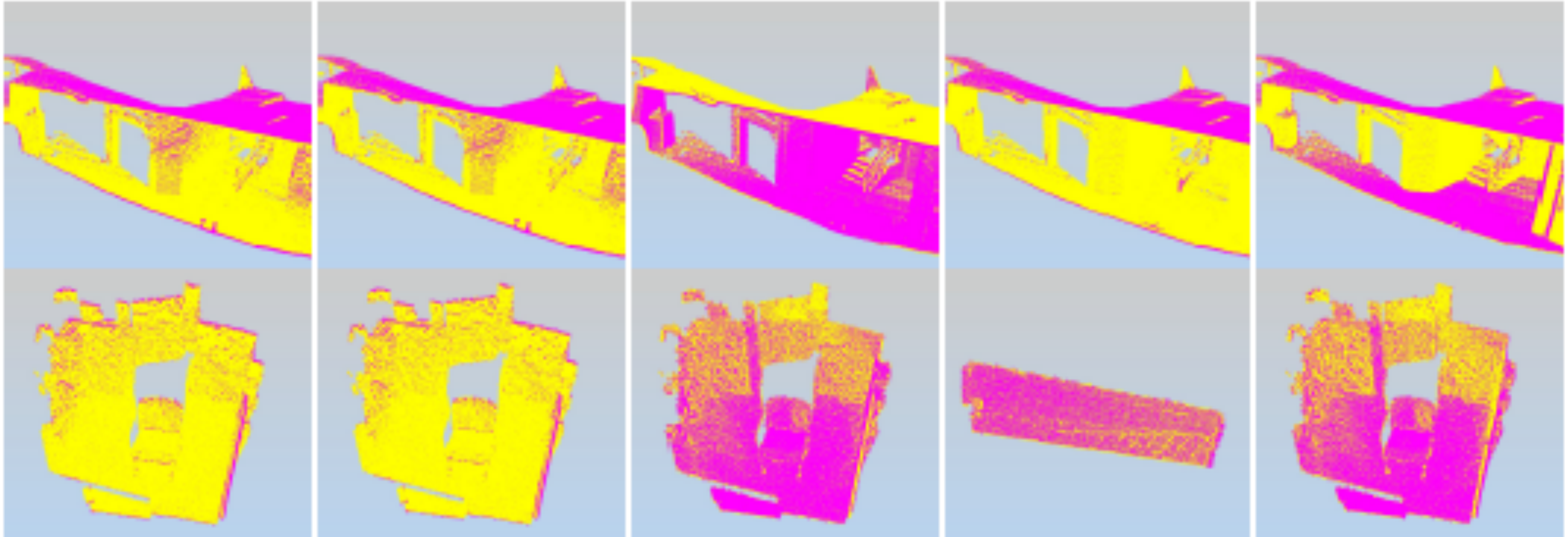
Evaluation

- Evaluate all methods on different data sets that cover a variety geometries
- Test on real data from different sensors
- Use artificially noised data to compare impact of inaccurate measurements to ground truth
- Always try to get the parameterization that generates the best visual results





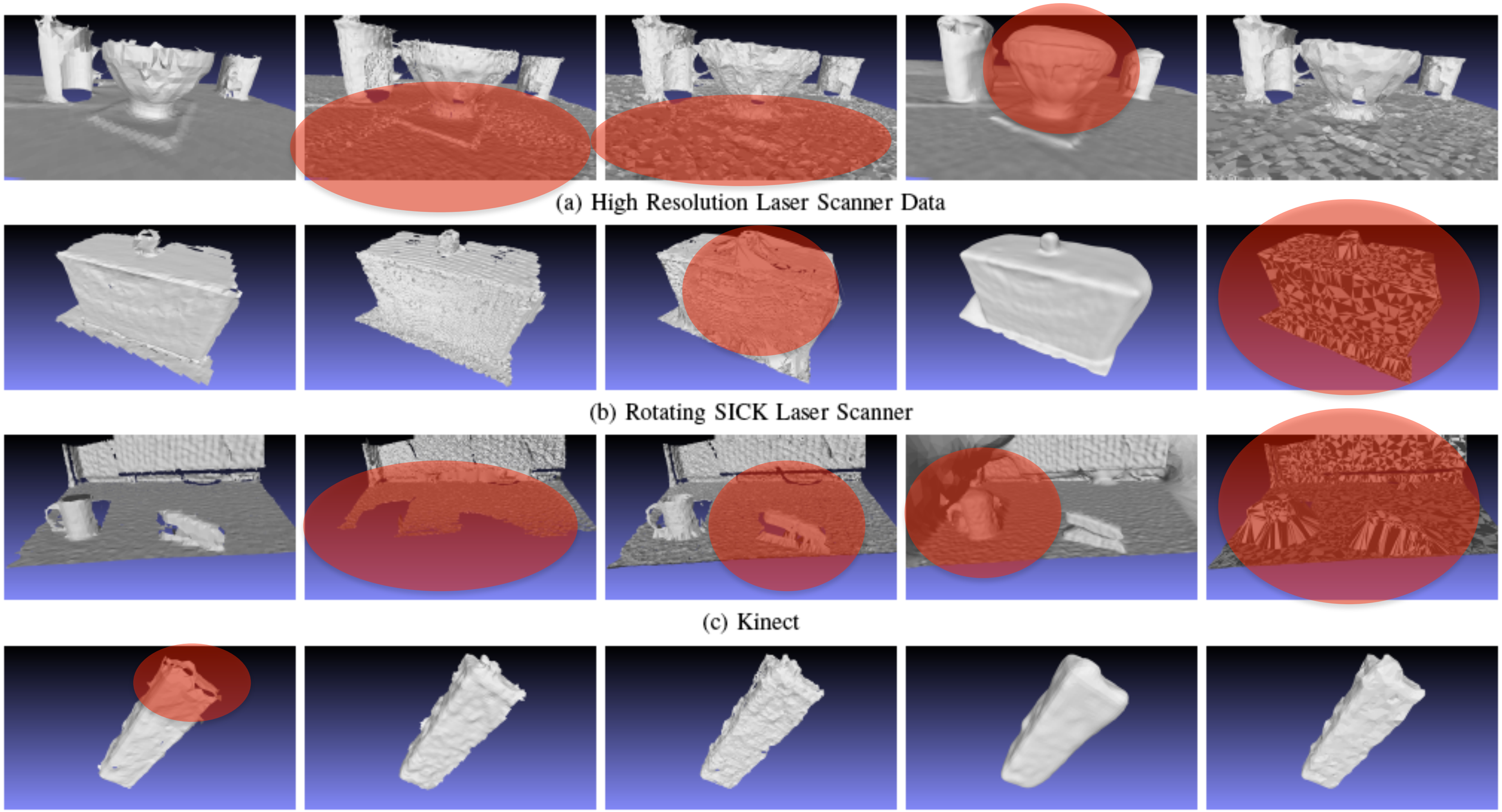
Normal Quality



LVR LVR RANSAC PCL CGAL MESHLAB

Geometry and Data Quality

LVR APSS Ball Pivoting Poisson Alpha Shape



(a) High Resolution Laser Scanner Data

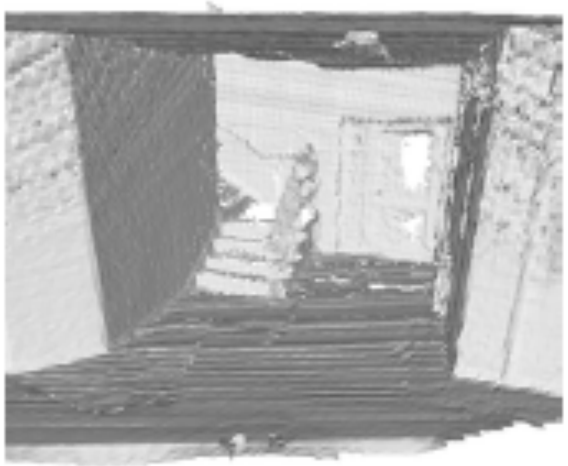
(b) Rotating SICK Laser Scanner

(c) Kinect

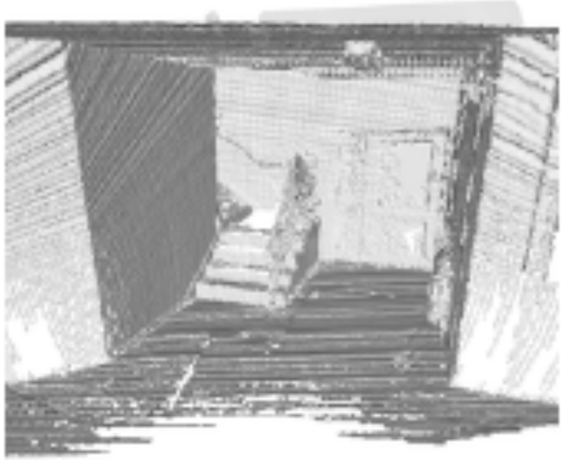
(d) Segmented Object from Kinect Data

UNIVERSITÄT OSNABRÜCK

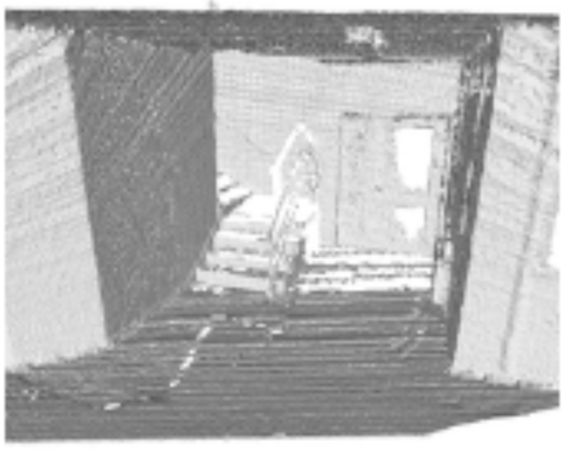
Geometry and Data Quality



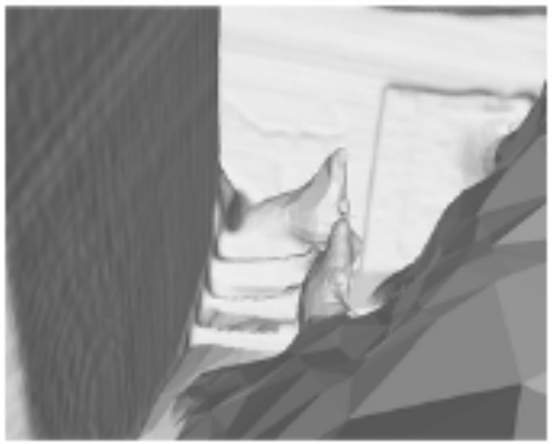
LVR



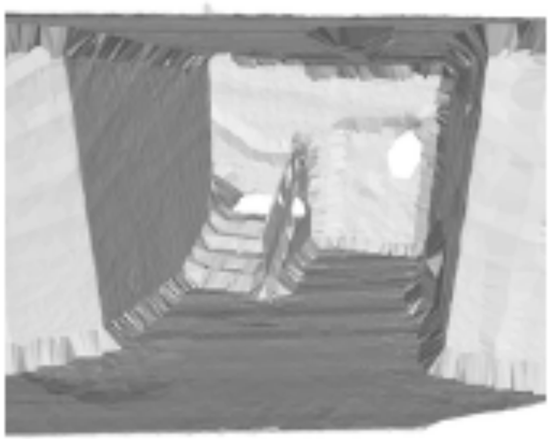
APSS



Ball Pivoting



Poisson



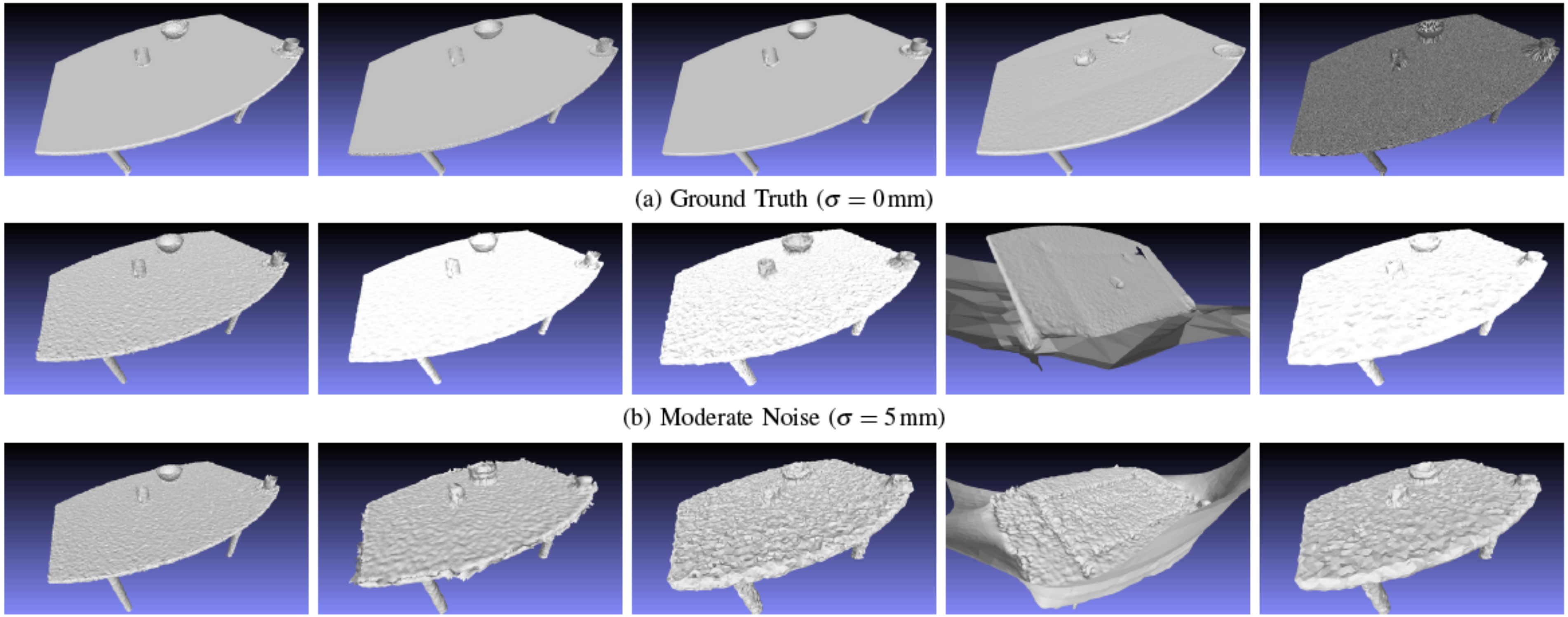
Alpha Shapes

Comparison

Dataset	Method	Time	Dist.	Edges/Vertices	Triangles
SICK	LVR	0.10	0.0257	0 / 6	6 267
	APSS	14.01	0.0307	0 / 8	478 061
	Ball Pivoting	11.90	0.0166	0 / 4 168	36 363
	Poisson	28.07	0.3794	0 / 0	67 094
	Alpha Shape	1.20	0.0024	186 188 / 1	301 091
Leica	LVR	0.30	0.0910	0 / 74	26 615
	APSS	32.20	0.0320	0 / 133	292 101
	Ball Pivoting	830.35	0.0022	0 / 5 595	476 919
	Poisson	59.32	0.1610	0 / 0	613 087
	Alpha Shape	39.53	0.0004	2 578 906 / 0	4 382 256
Kinect	LVR	0.70	0.2039	0 / 48	60 628
	APSS	158.00	0.1995	0 / 1284	595 269
	Ball Pivoting	448.59	0.1297	0 / 4861	252 678
	Poisson	64.01	0.2257	0 / 0	596 046
	Alpha Shape	27.98	0.0339	1 999 144 / 9	3 408 572

Noisy Data

LVR APSS Ball Pivoting Poisson Alpha Shape



Noisy Data

Method	σ = 0.0 mm Time	Mean. Dist.	Max. Dist.	σ = 0.5 mm Time	Mean. Dist.	Max. Dist.	σ = 1.0 mm Time	Mean. Dist.	Max. Dist.
LVR	0.90	0.01	1.26	2.00	0.20	2.52	3.50	0.73	3.00
APSS	936.21	0.01	1.26	105.77	0.25	2.95	221.88	0.82	6.62
Poisson	105.10	0.41	8.83	62.53	24.00	93.20	28.62	26.53	87.21
Alpha Shapes	3.66	0.50	2.52	6.00	0.58	3.87	66.85	0.93	5.26
Ball Pivoting	1138.34	0.03	1.26	1133.32	0.58	3.87	1334.97	1.090	5.26

- LVR delivers best results on noisy data while having the best run time
- Poission has the highest mean and max distance (in this experiment)
- Ball Pivoting has the highest run time

Conclusion

- Direct triangulation methods deliver *globally* good results but are highly sensitive to noise and have long run times
- Poisson reconstructions are topologically correct and can handle occlusions, if objects are segmented and symmetric
- Marching Cubes reconstructions can be used on arbitrary surfaces and can be robust against noise
- LVR is fast, noise resistant and can be used on arbitrary surfaces