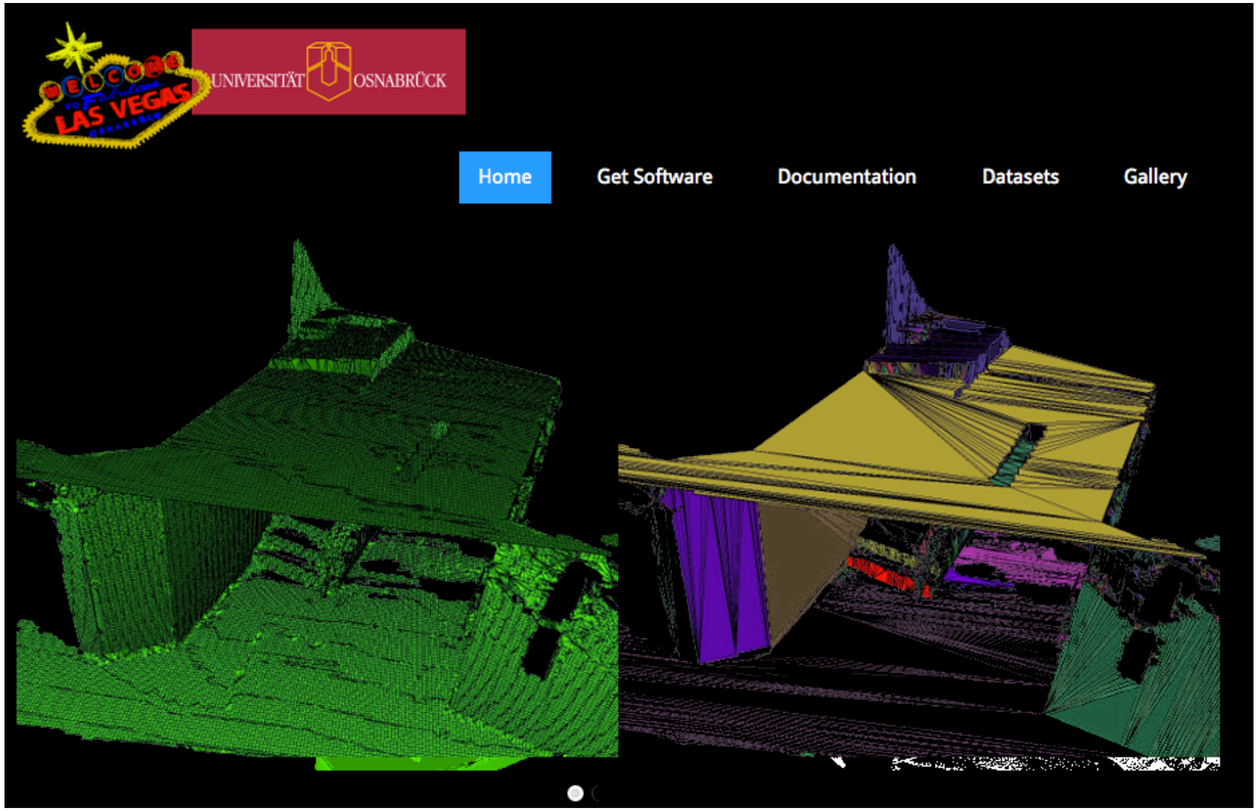
Large Scale Point Cloud Processing Tutorial

Meshing on Large Point Clouds

Thomas Wiemann, Andreas Nüchter

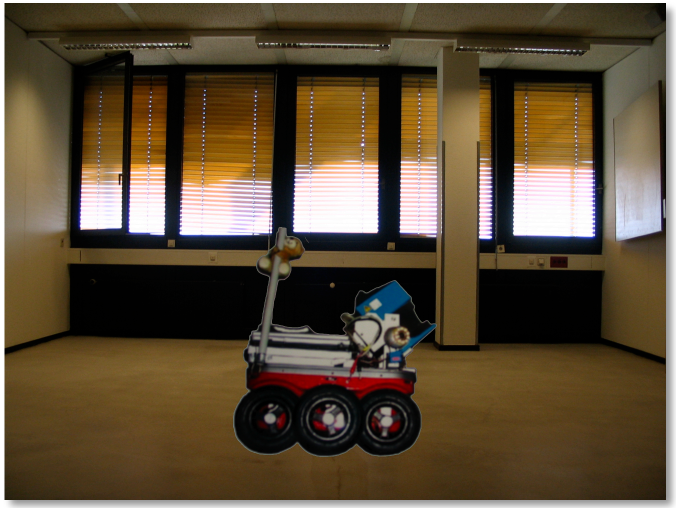


Software: http://www.las-vegas.uni-osnabrueck.de

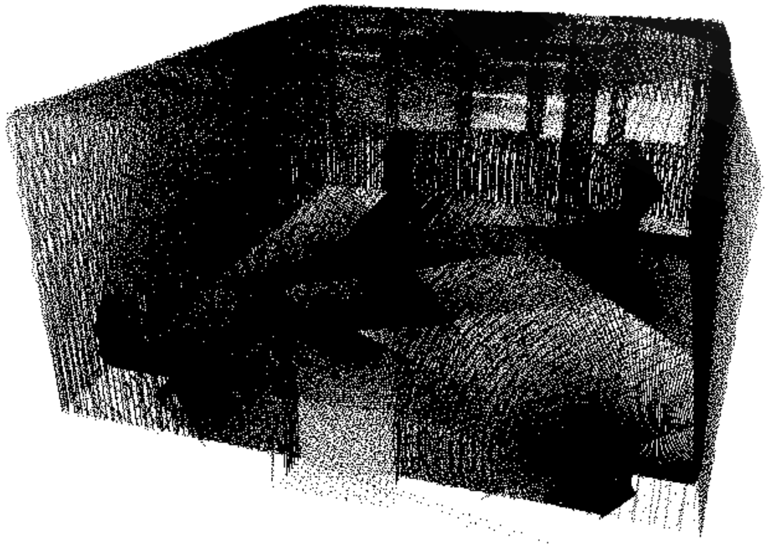
UNIVERSITÄT OSNABRÜCK

Introduction

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



6 DoF SLAM*
with global relaxation

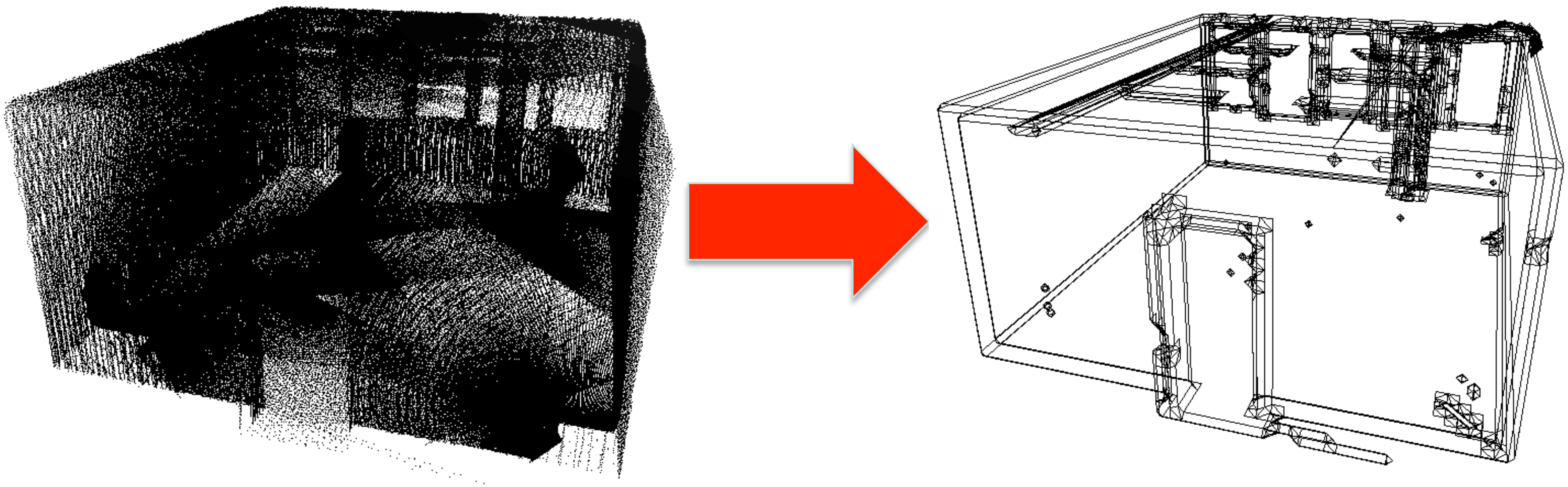


- But we do not get a surface representation, only samples

*Bormann et al. 2008

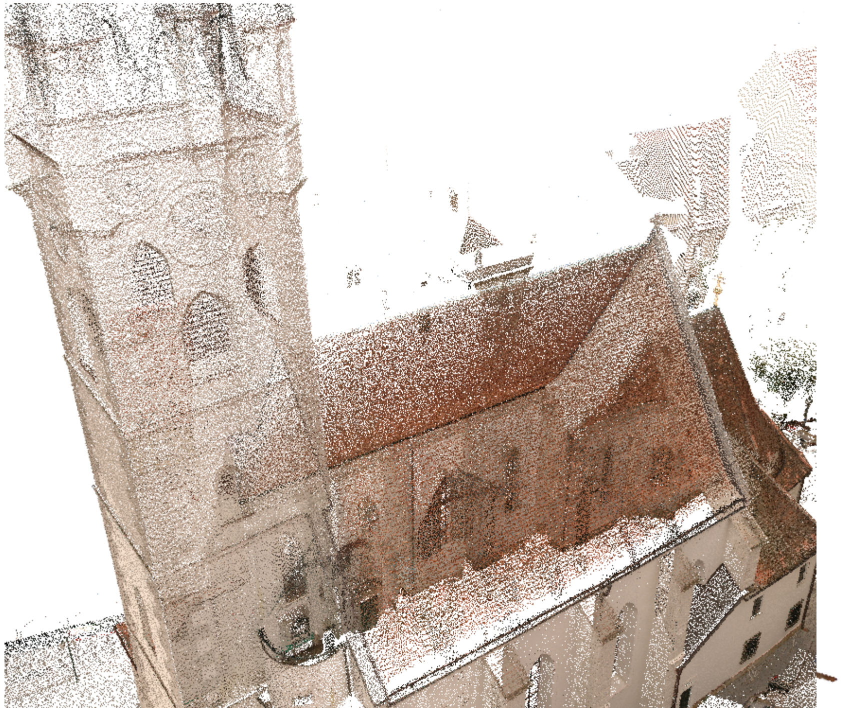
Introduction

- 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

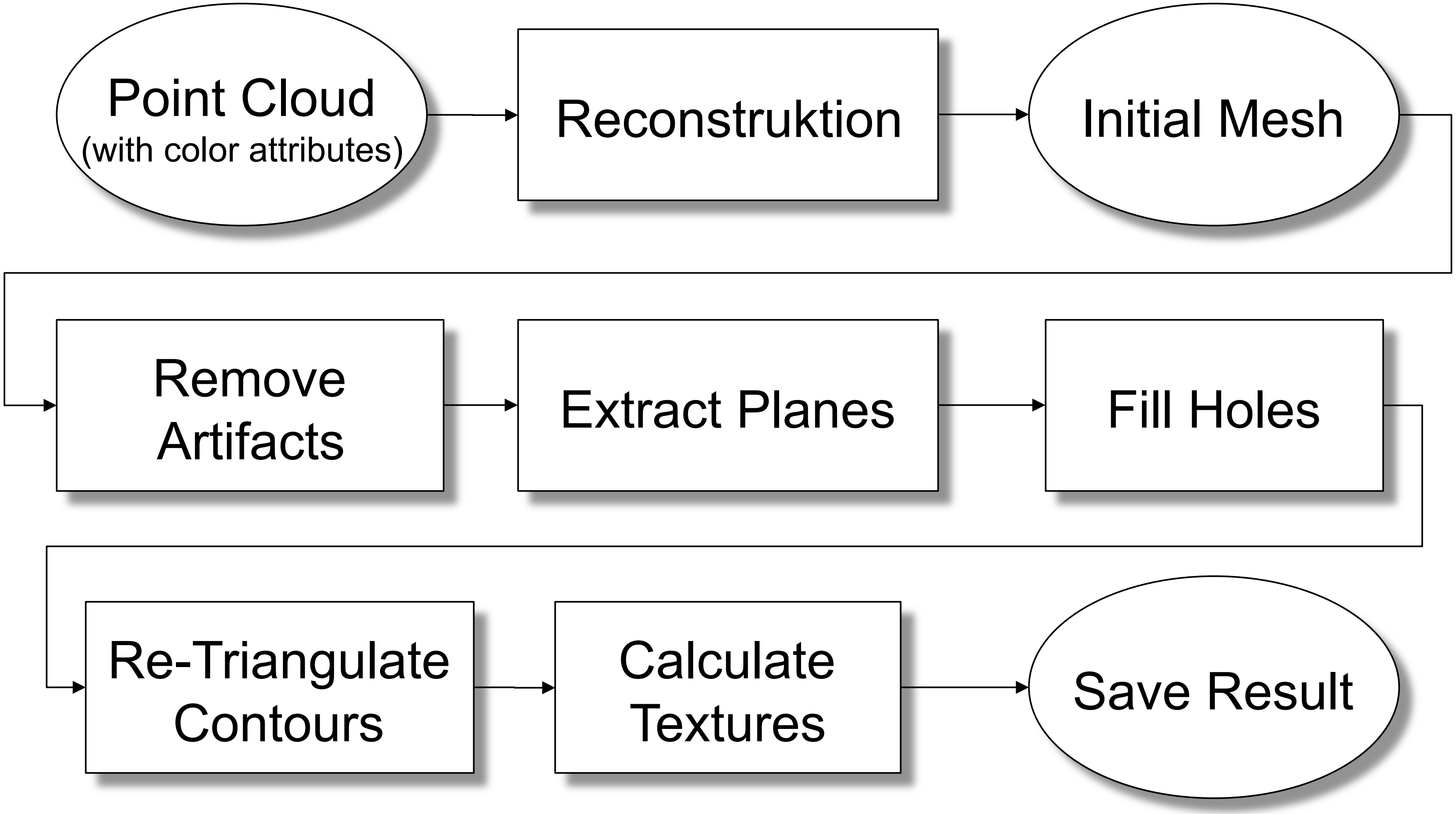
Motivation



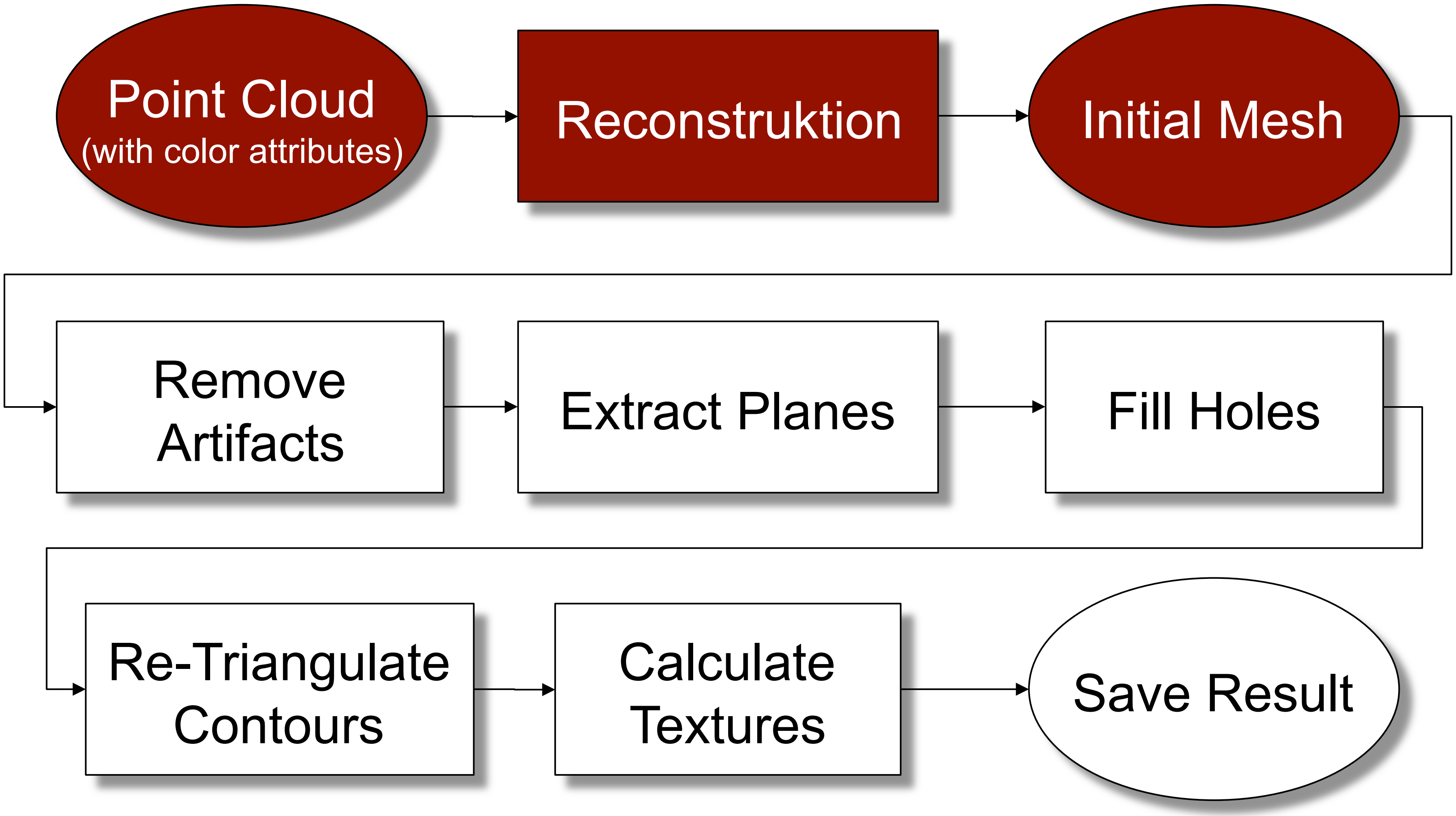
Daten: RIEGL

Las Vegas Reconstruction - Example

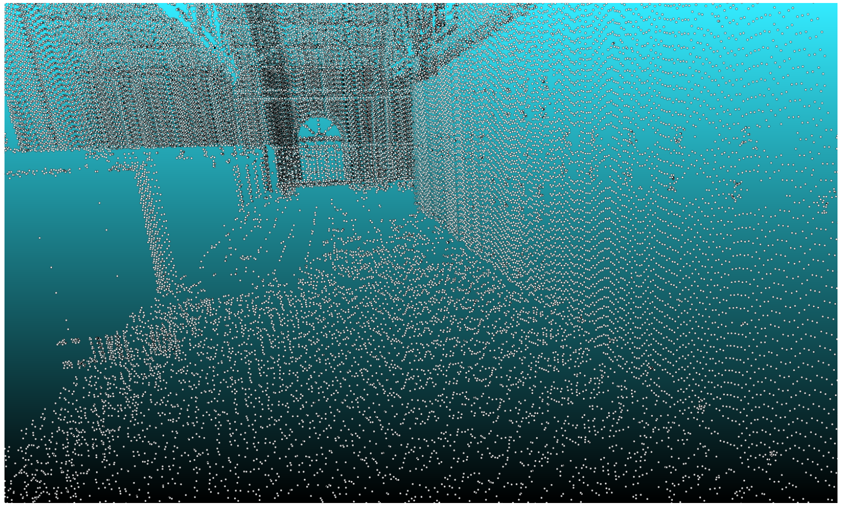
Processing Pipieline

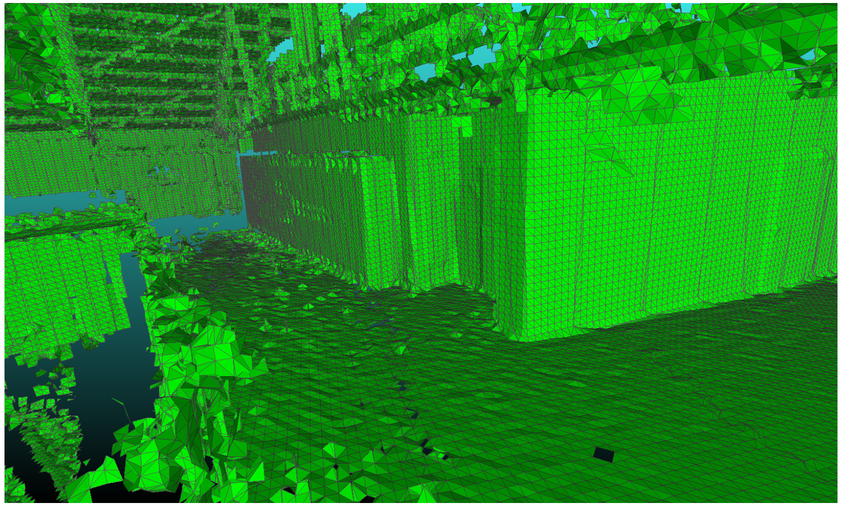


Processing Pipieline



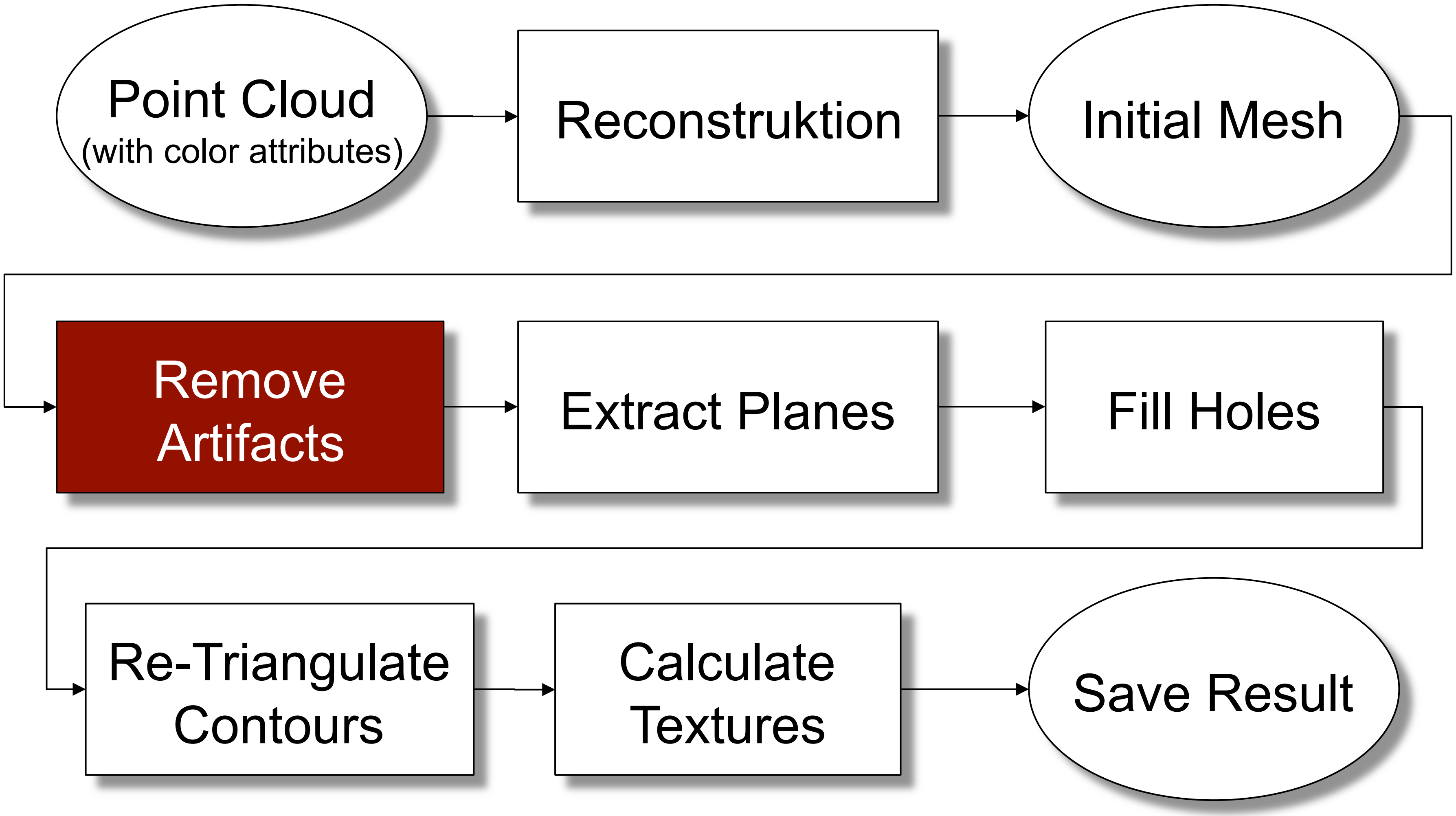
Reconstruktion



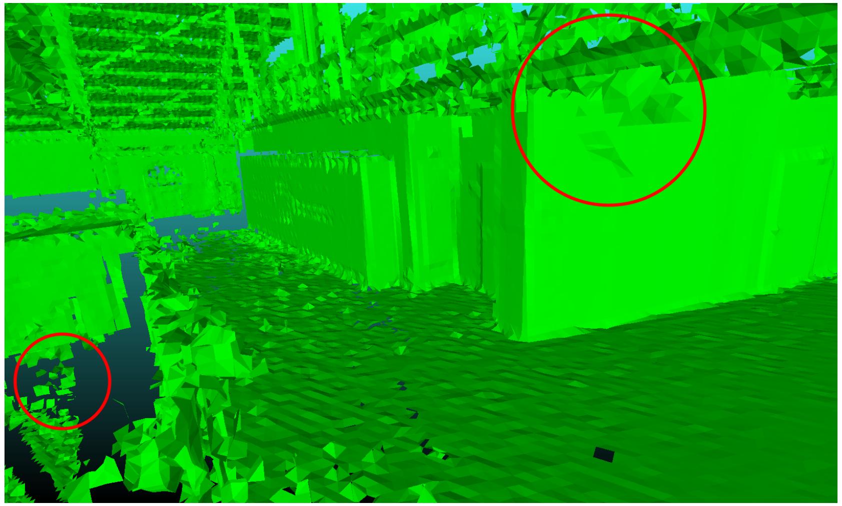


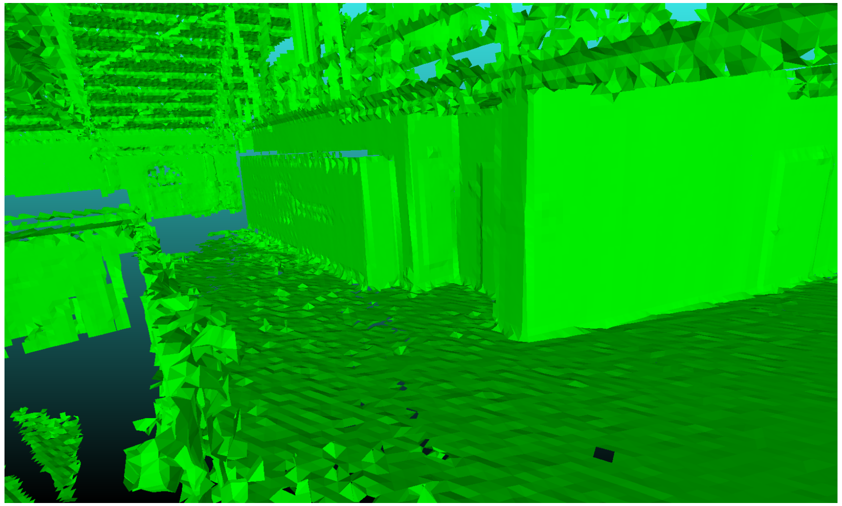
- Reconstruction using Marching Cubes variants
- Using Hoppe‘s signed distance function
- Different methods for normal estimation
- Store the mesh a Half-Edge-Represention
- Do everything in parallel if possible

Processing Pipieline



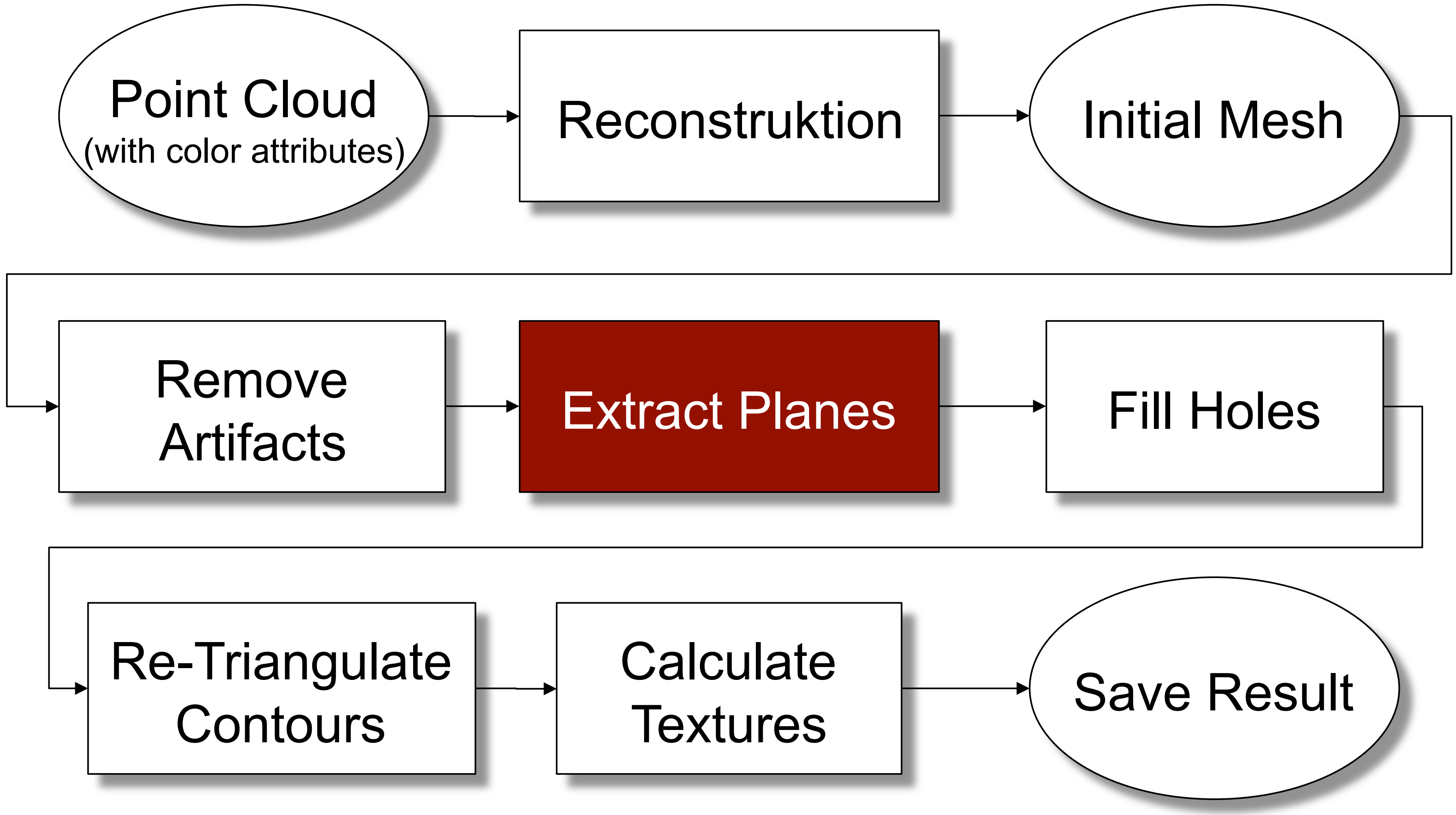
Remove Artifacts



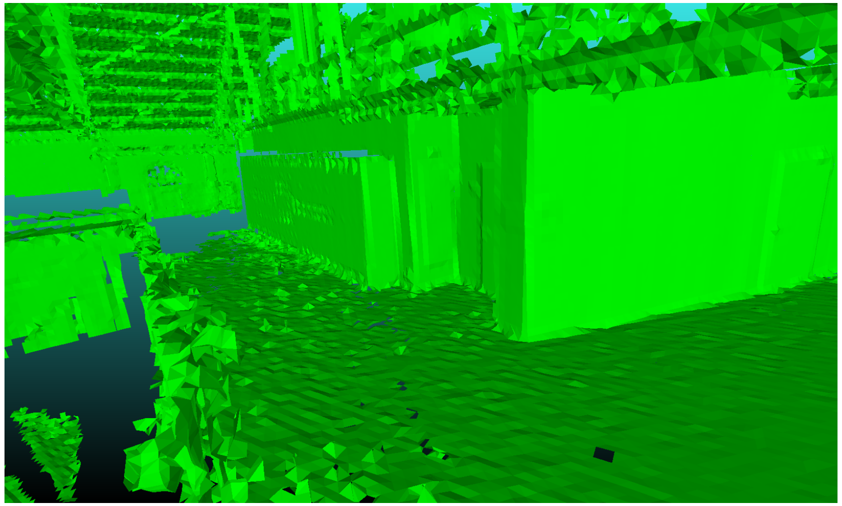


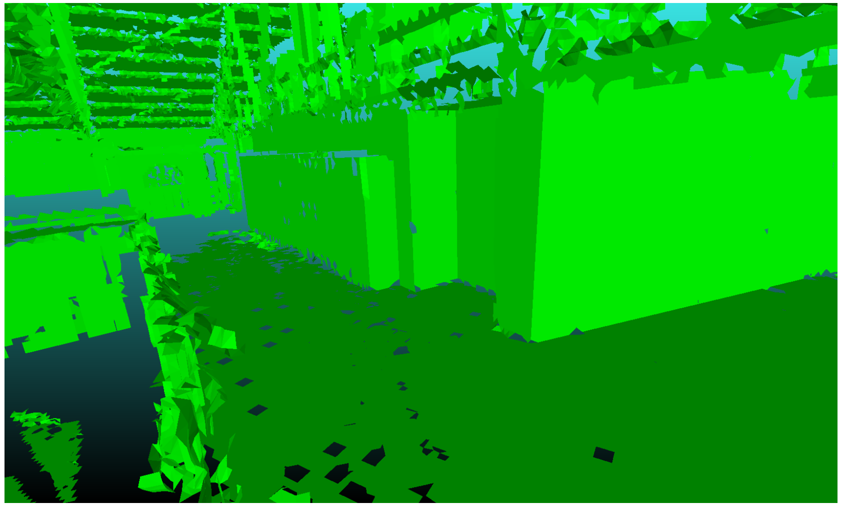
- Remove triangle clusters unconnected to the mesh
- Using recursive region growing
- Heuristic: Number of triangles in cluster
- Done before holes are closed

Processing Pipieline



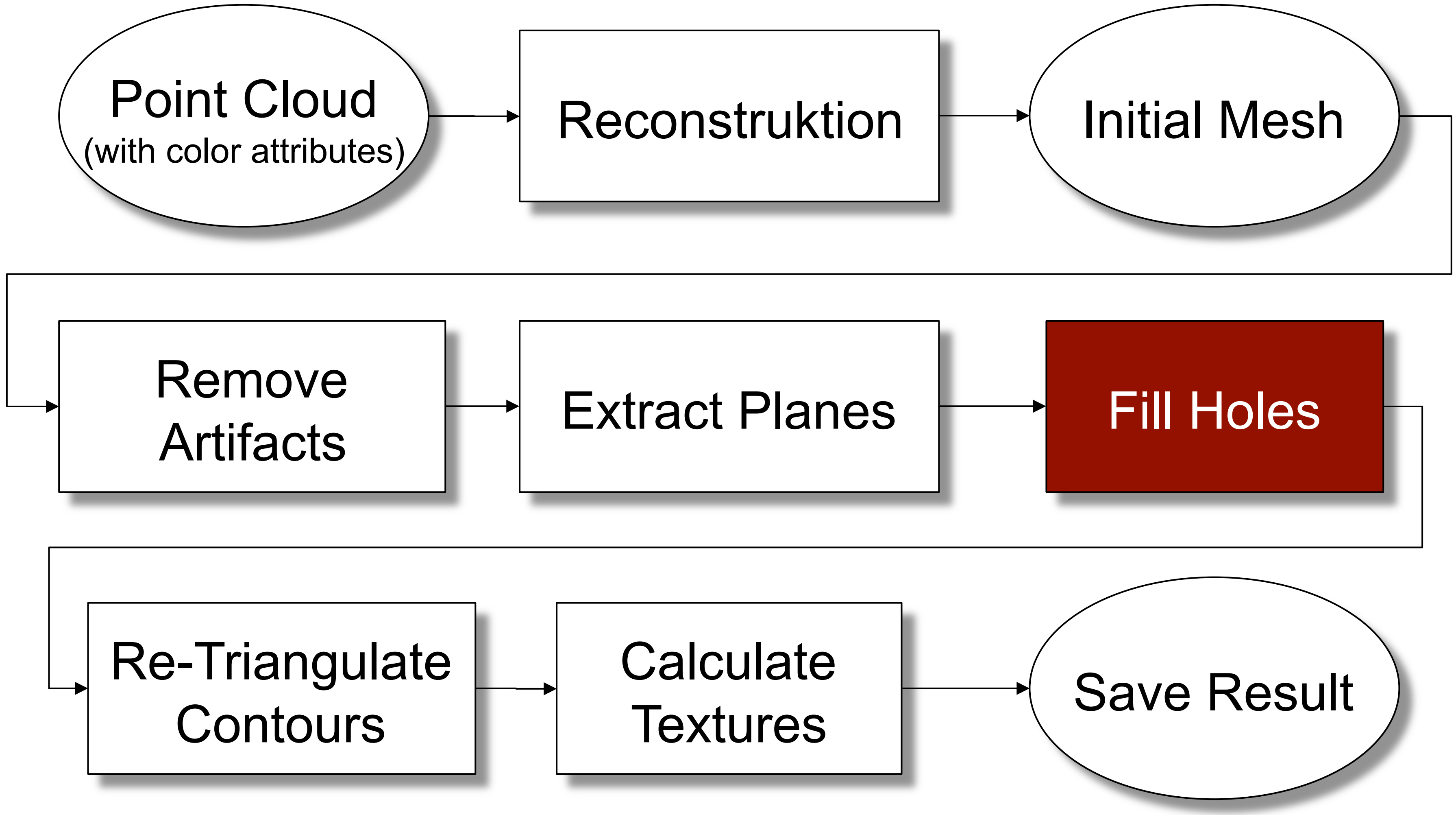
Extract and Optimize Planes





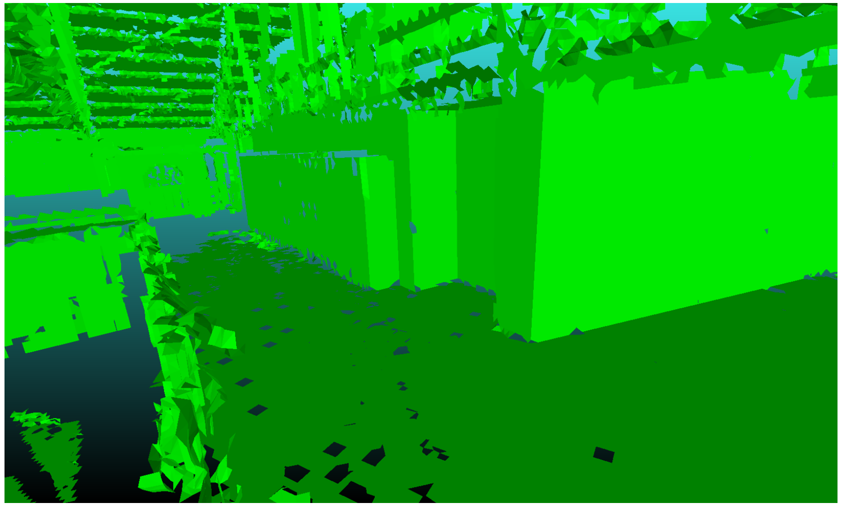
- Detect planes via region growing with normal threshold
- Optimize vertex positions by dragging them into the plane
- Make this iteratively to merge planes that come closer
- Delete small regions that do not belog to planes

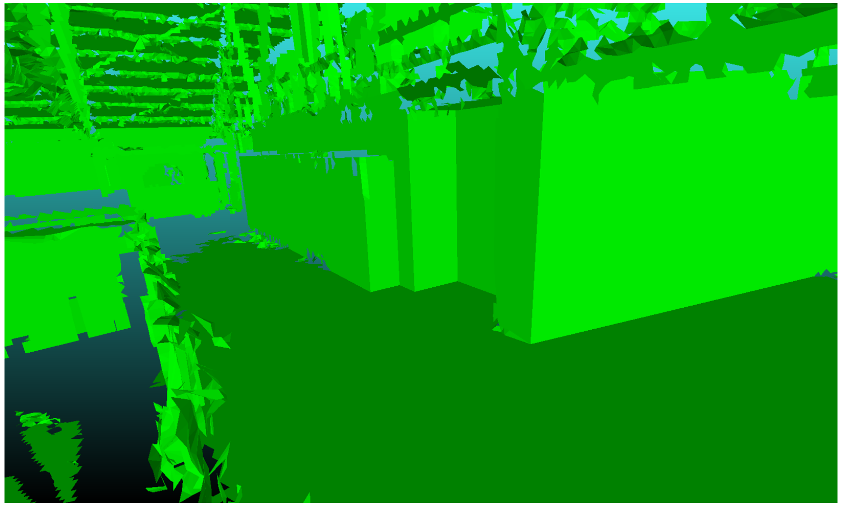
Processing Pipieline



UNIVERSITÄT OSNABRÜCK

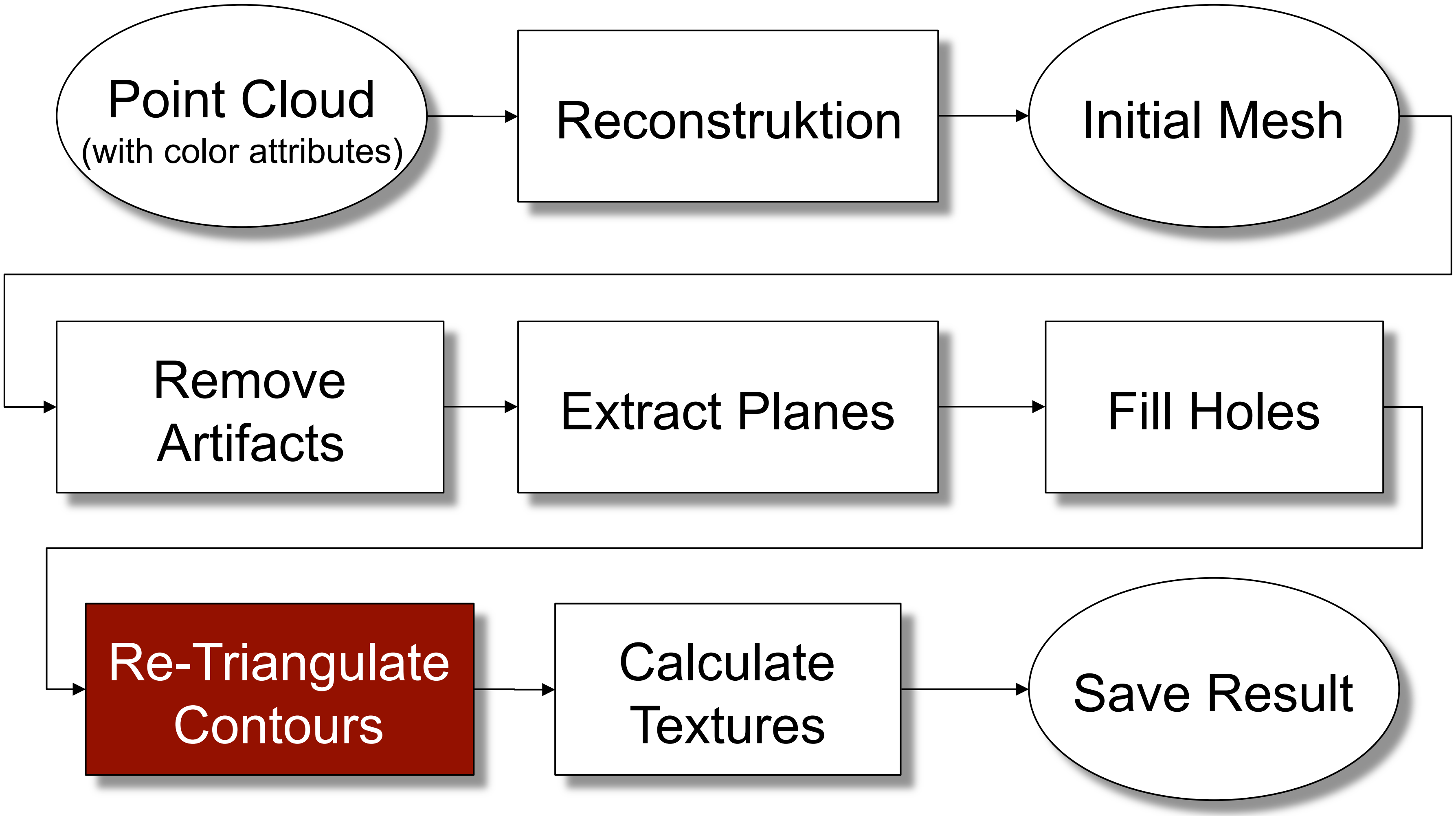
Fill Holes



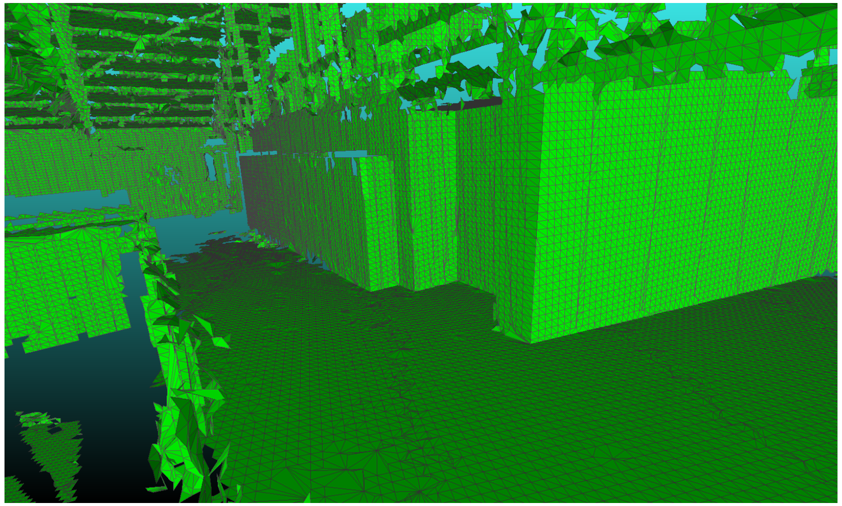


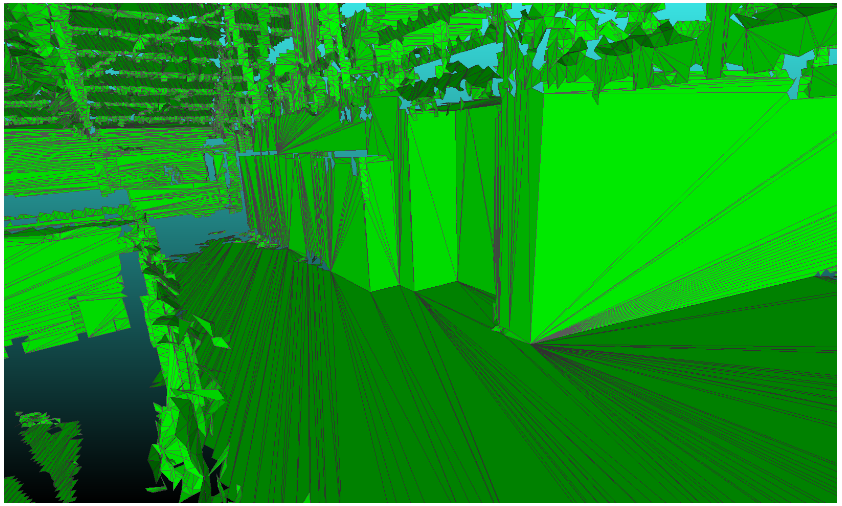
- Trace contours within planes
- Close contours up to a given size
- Number of edges in the hole polygon
- Close by edge collapsing

Processing Pipieline



Re-Triangulate



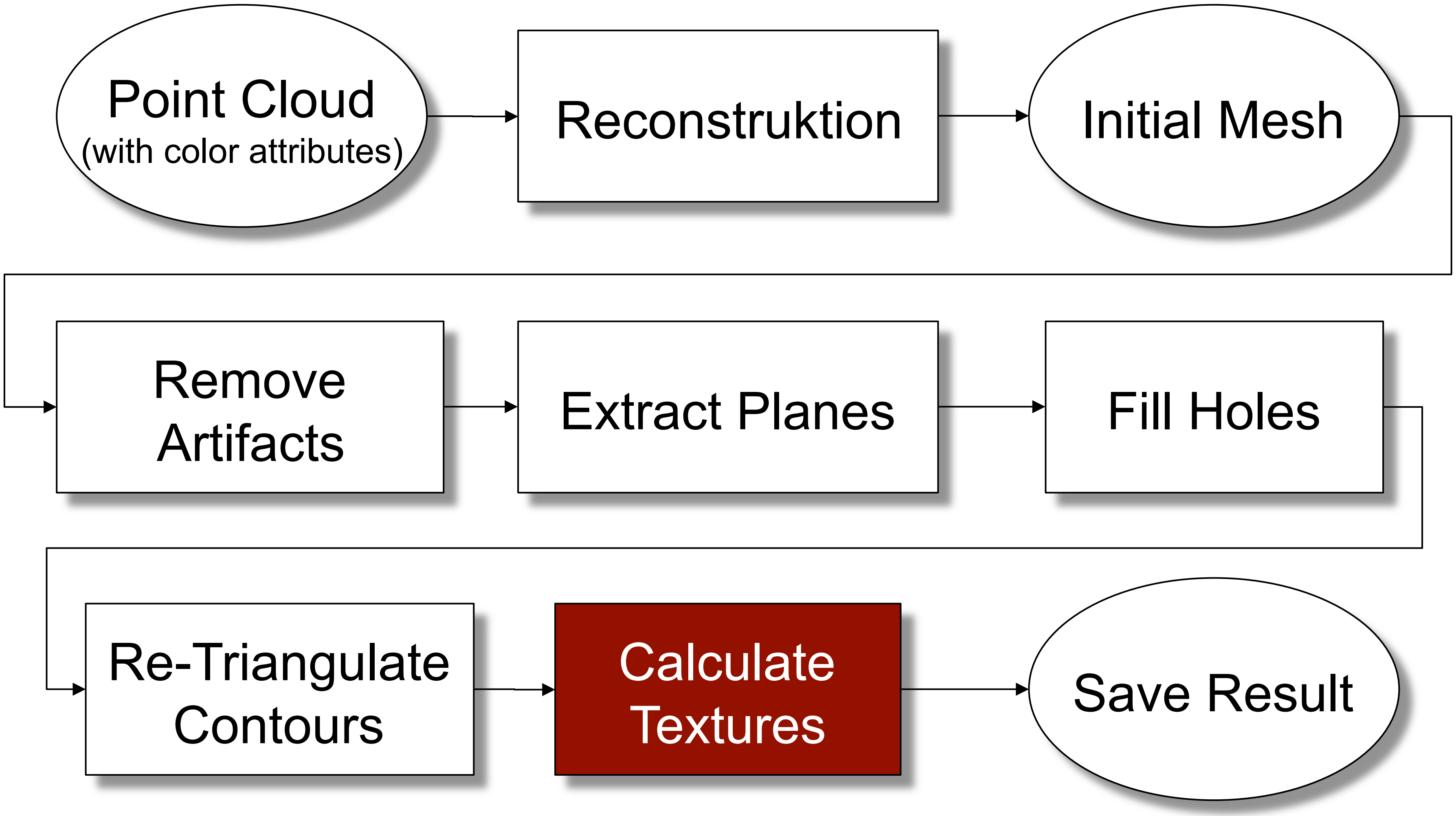


- Generate new triangulation of plane contours
- Use the OpenGL-tesselator
- Usually computed on graphics card
- No change of geometry, but topology

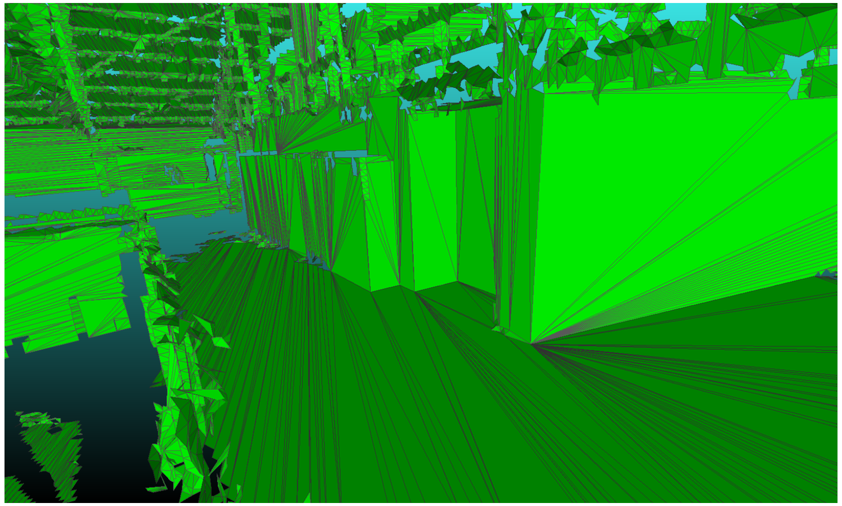
Re-Triangulate

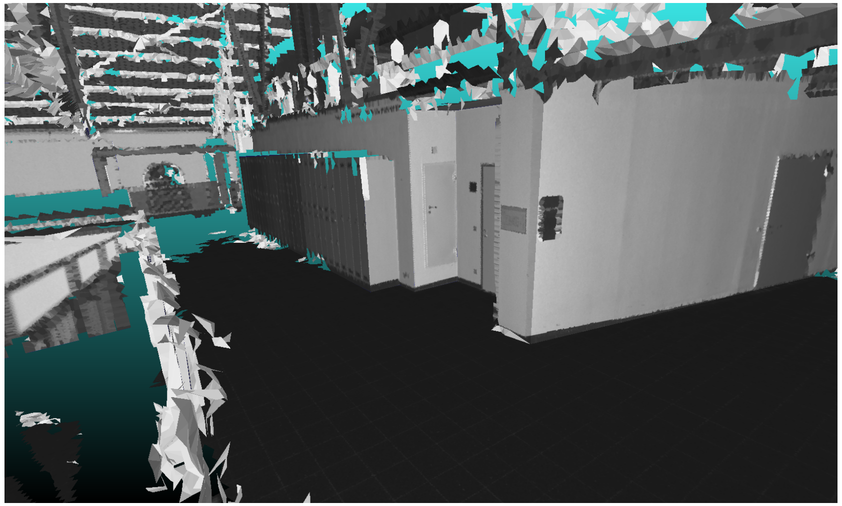
	File Size [MB]	Num Points	Num Faces
Initial Point Cloud	132.5	4,253,689	-
Mesh without Re-Triangulation	10.8	221,443	371,460
Mesh with Re-Triangulation	4.5	119,557	98,648

Processing Pipieline



Calculate Textures

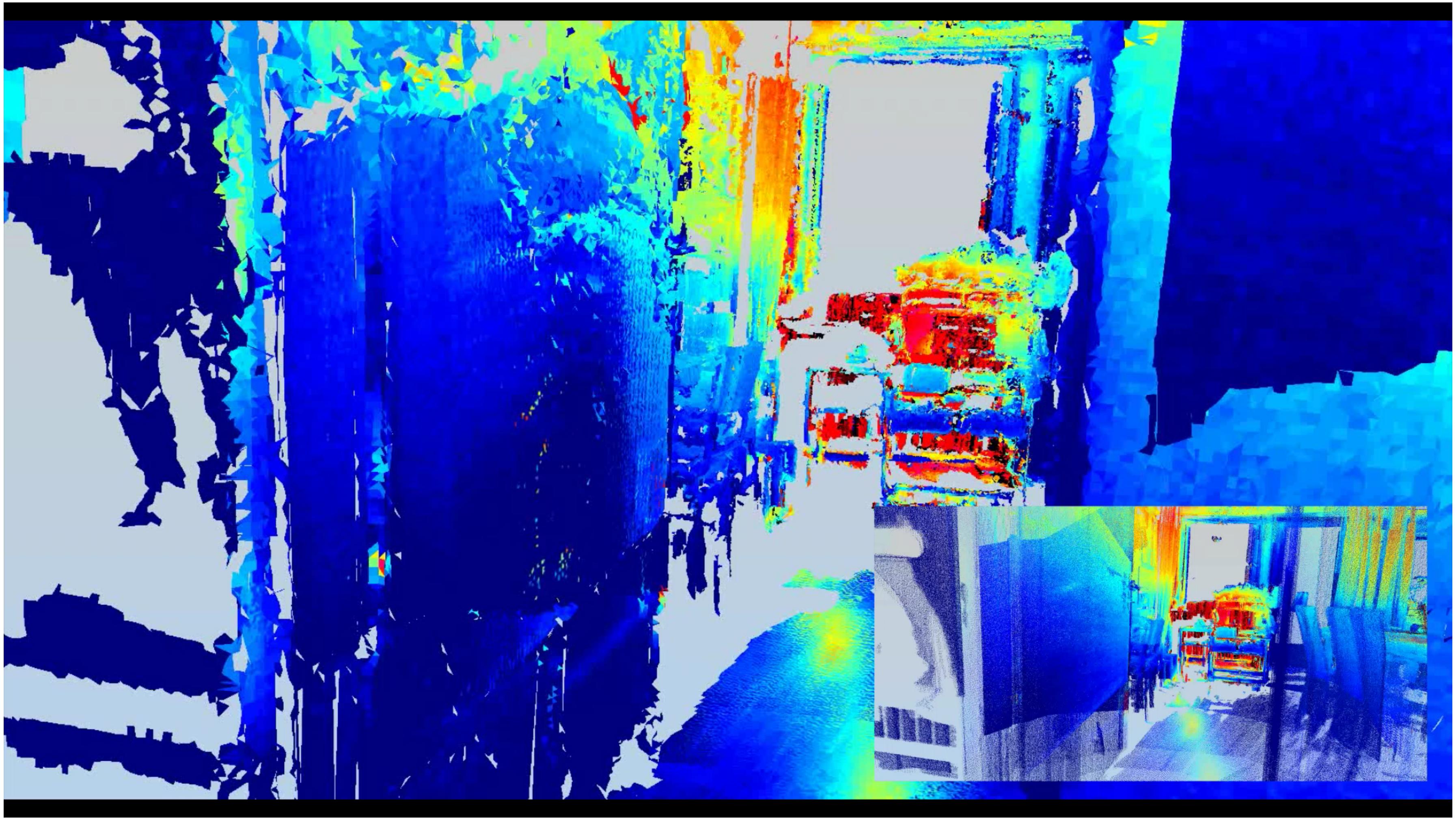




- Every plane can be associated with a bitmap texture
- Small regions are rendered with a suitable color
- Colores are generated from the information in the input clouds
- File format: Wavefront OBJ

Example

Example: Thermal data



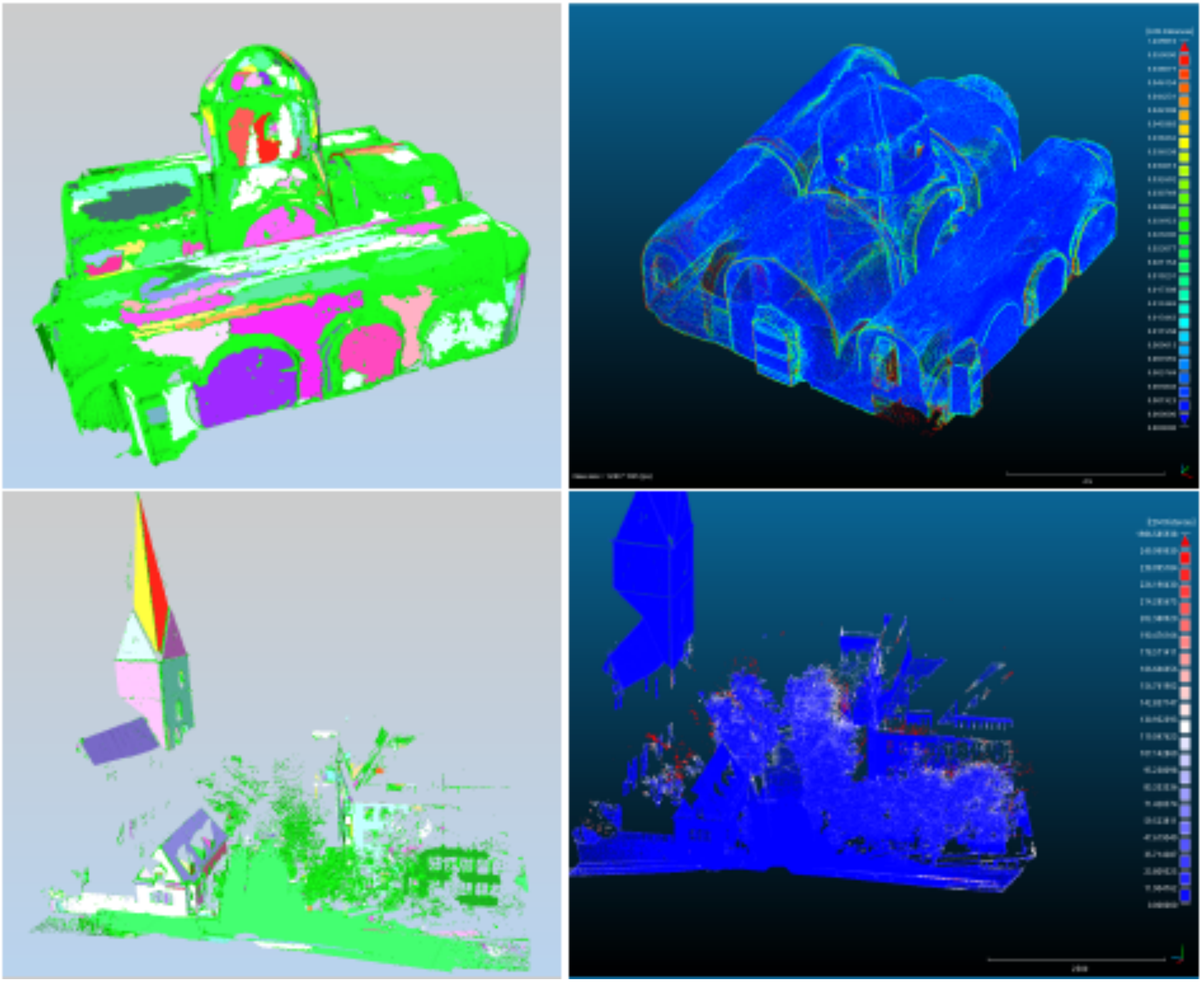
Accuracy

- Manual comparism with ground truth

	Ceiling	Width	Depth	Door Width
Original Geometry	2.99 m	5.89 m	7.09 m	0.96 m
Map	2.96 m	5.85 m	7.06 m	0.94 m

Accuracy

- Approximation quality with respect to the input data



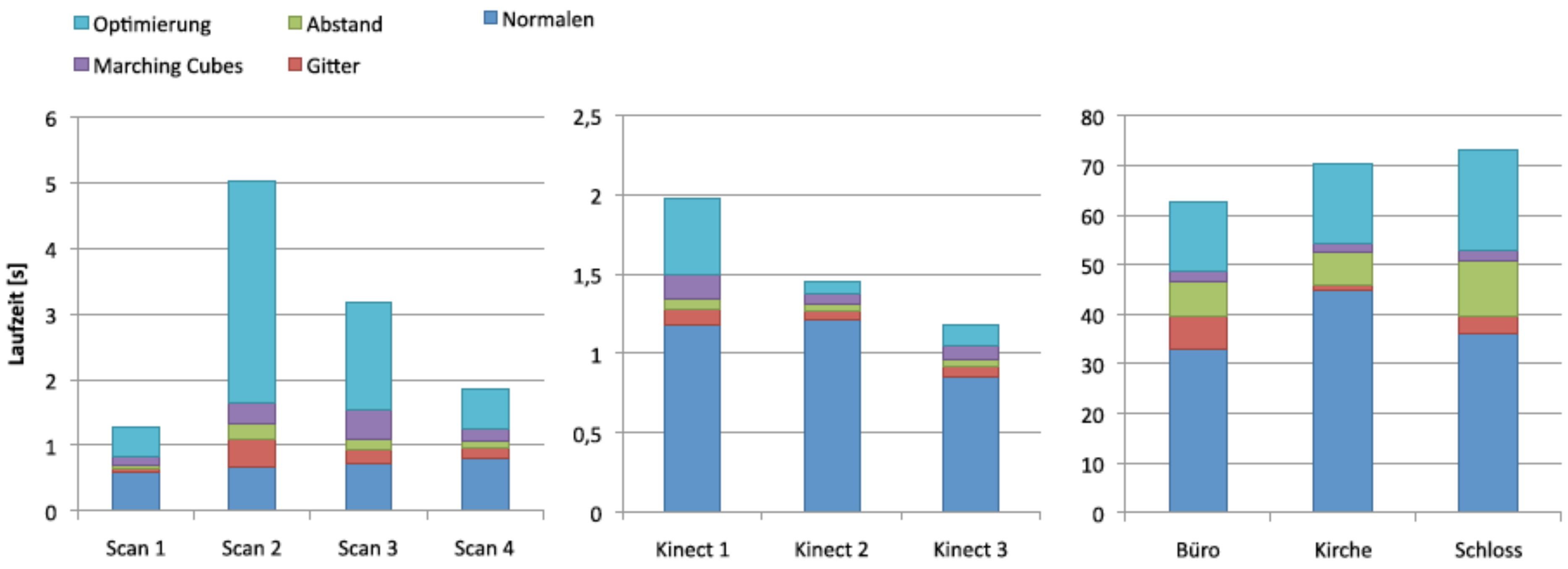
Accuracy

- Impact of mesh optimization pipeline:

Data Set	**Dev. Rec.[mm]**	**Dev. Opt [mm]**	**Compression**
Kinect	2,52	11,99	73%
Laserscan	10,31	7,03	66%
Office	1,48	0,58	75%
Church	1,40	4,22	48%
Street	2,15	12,39	56%

- Good preservation / improvement in planar environments
- Errors within the range of the sensor noise
- Optimization reduces significantly

Runtime



Optimierung
Abstand
Normalen
Marching Cubes
Gitter
Laufzeit [s]
0
1
2
3
4
5
6
Scan 1
Scan 2
Scan 3
Scan 4
0
0,5
1
1,5
2
2,5
Kinect 1
Kinect 2
Kinect 3
0
10
20
30
40
50
60
70
80
Büro
Kirche
Schloss

Step 1: Initial Mesh Generation

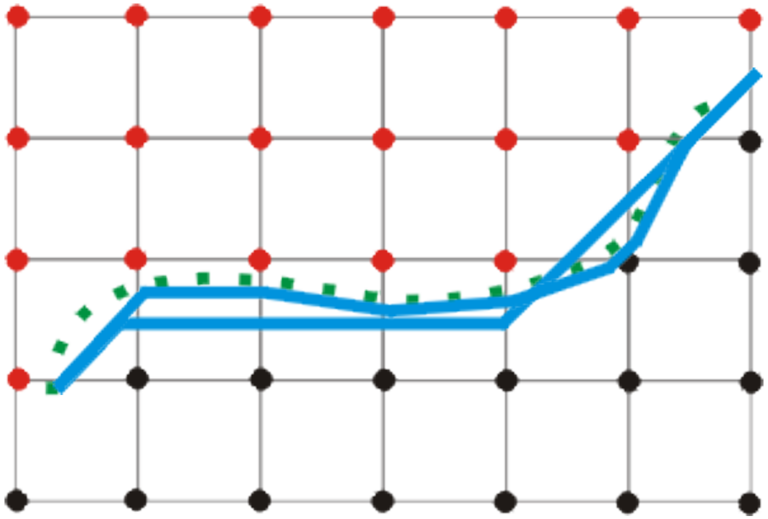
- Idea: Use a modified Marching Cubes Algorithm*:
  - Divide space into cubic cells of equal size
  - Determine the cell corners, that are outside a given surface
  - Use pre-computed patterns to approximate the surface
- Output: List of triangles that approximate the surface
- Enhancements
  - Use hashing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

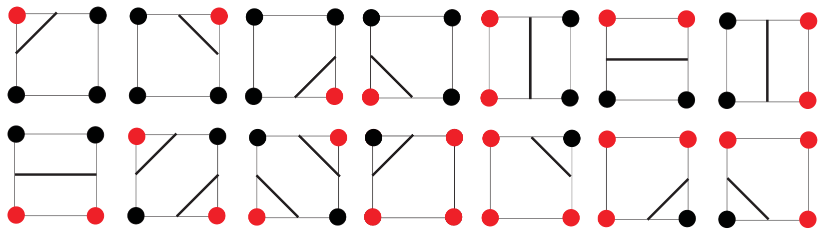
➪ Implementation issues

*Lorensen & Cline 1987

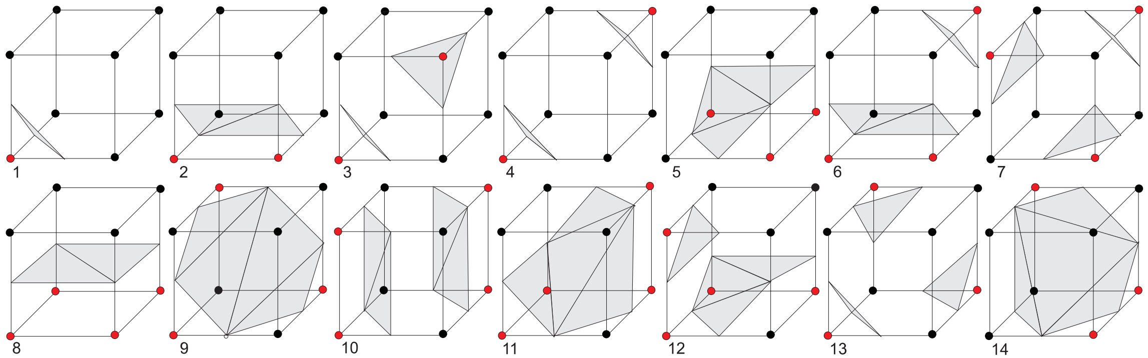
Step 1: Marching Cubes

2D Example:



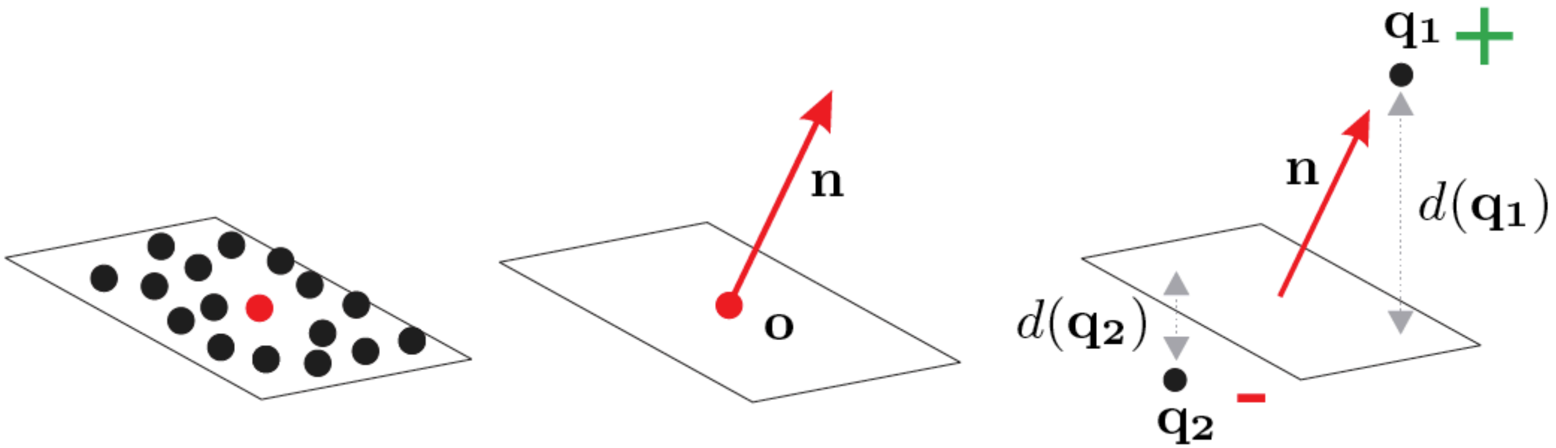


In 3D 14 basic patterns are needed:



Surface Interpolation

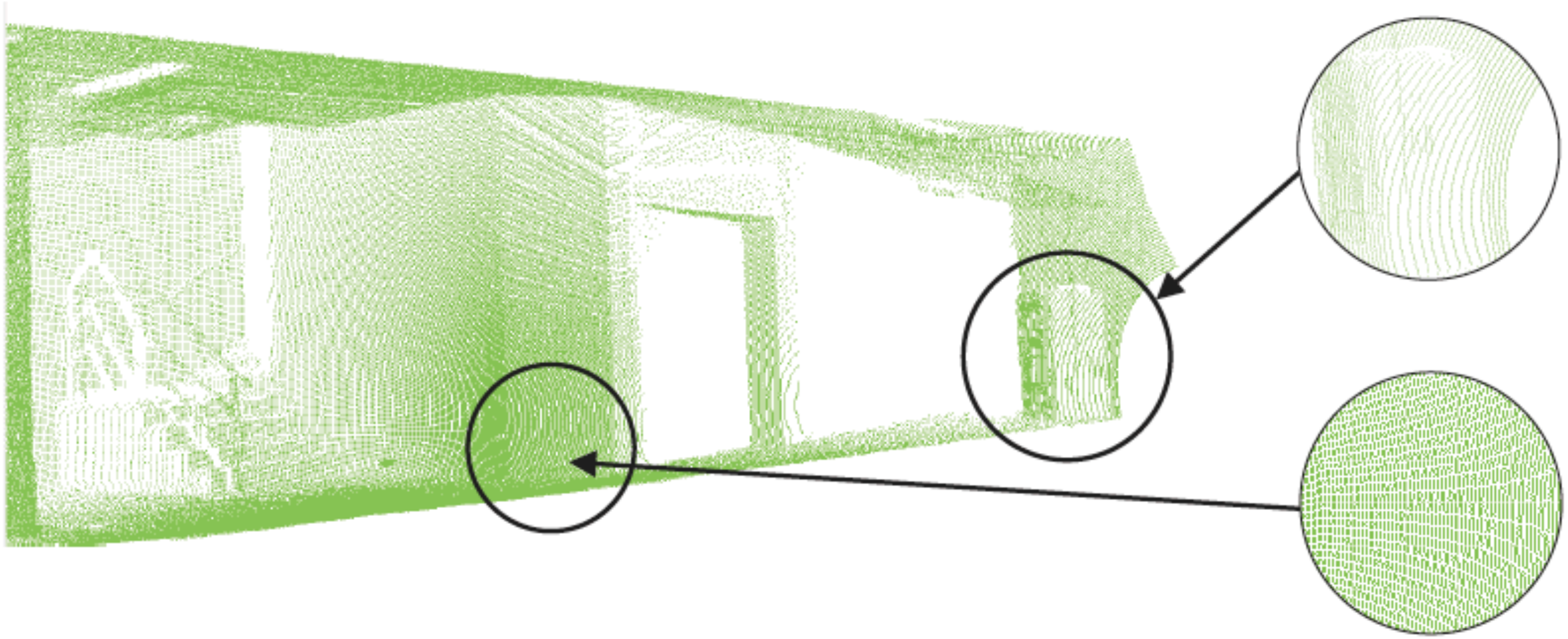
- Hoppe’s signed distance function (1992):



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

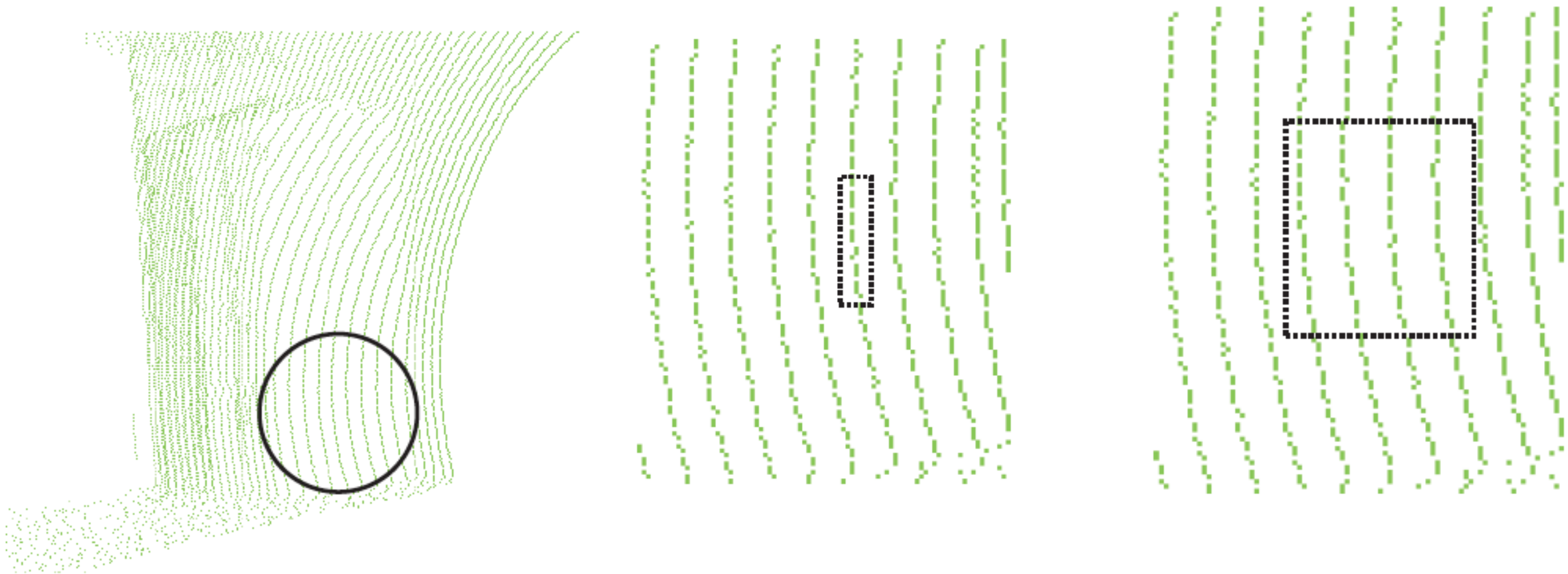
Surface Interpolation

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



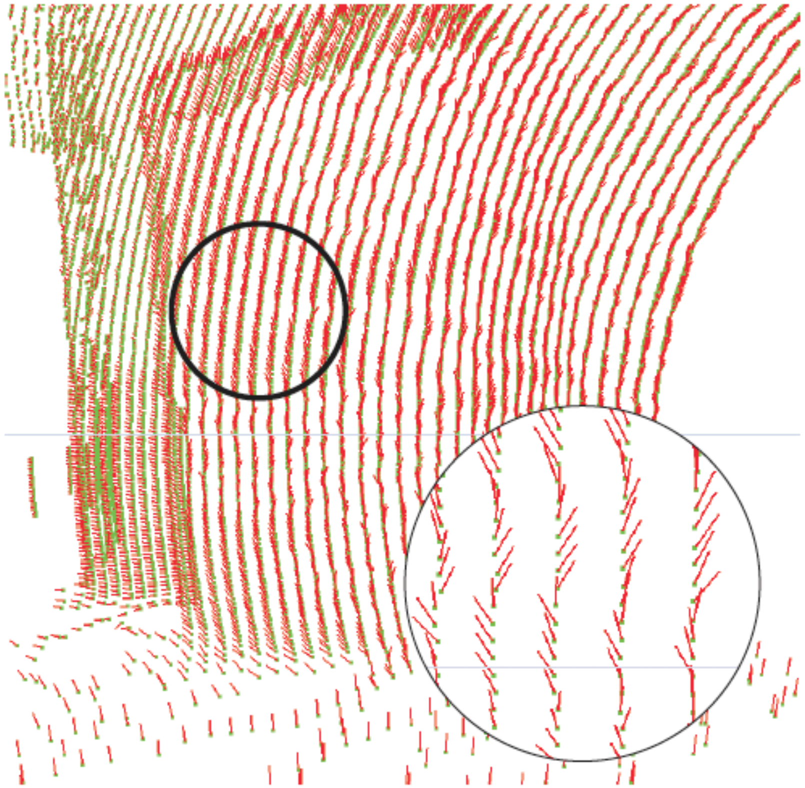
Robust Normal Estimation

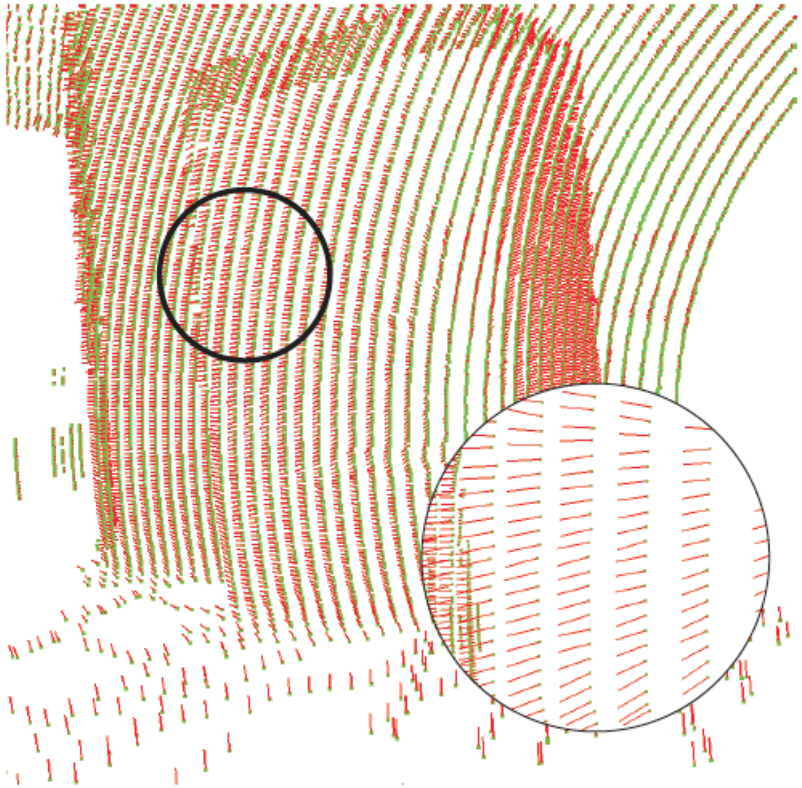
- The number of points needed for a robust normal estimation depends on noise and point density
- Use heuristic to determine the optimal number
- Analyze the bounding box of the *k*-neighborhood



Robust Normal Estimation

- Results:





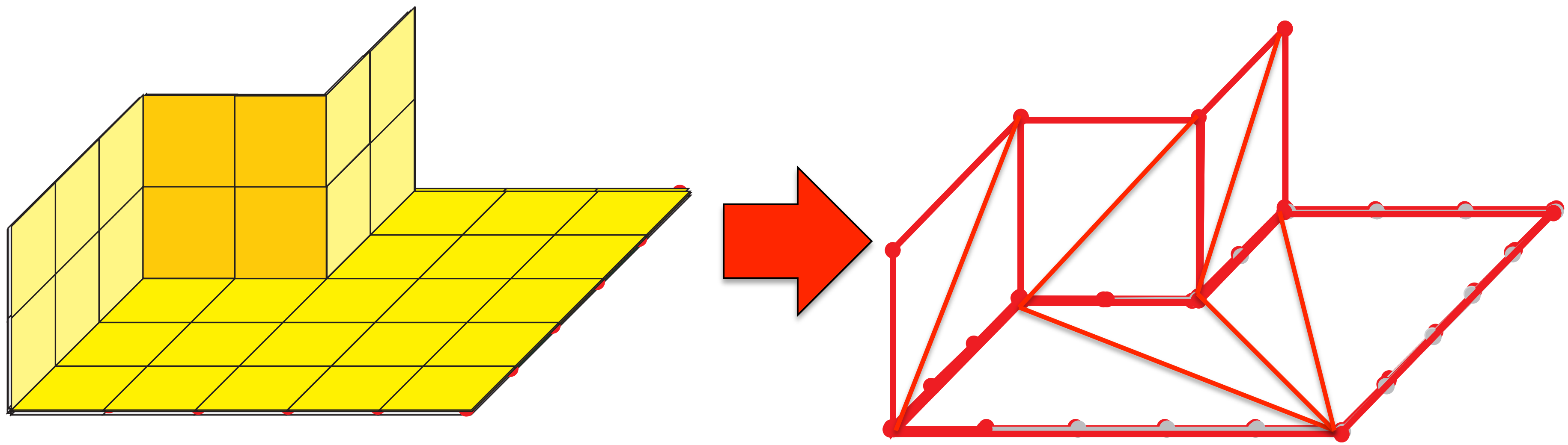
Robust Normal Estimation

- Influence on reconstruction accuracy:

Testing it!

- Call `bin/reconstruct -v5 ~/dat/sick_scans`
- 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/sick_scans`
- Hint: use `--e` to export good normals
- Use `bin/qviewer` to display the results

Mesh Optimization – Region Growing

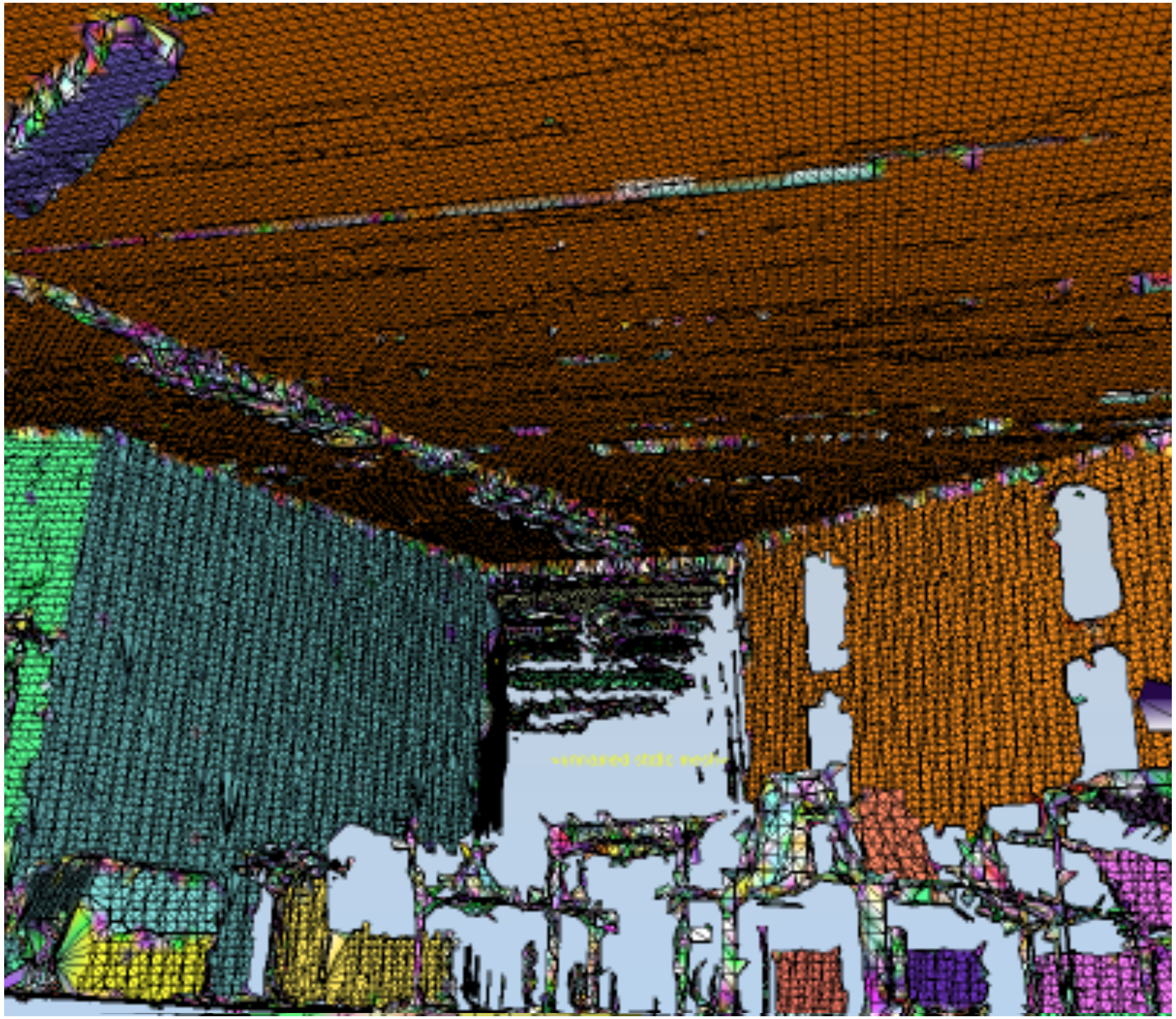


- Every triangle is checked exactly once
- Neighbor edges can be found in *O*(1) time

  ⇨ Linear time for polygon extraction

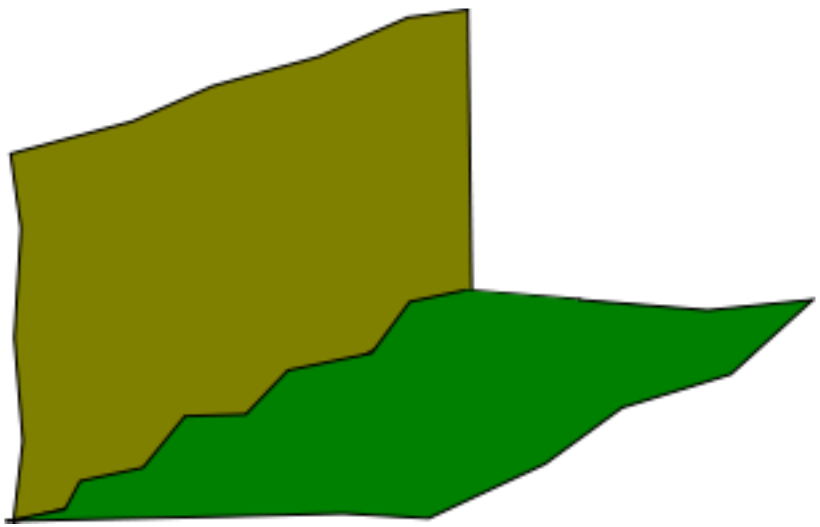
- After all planes have been found: Re-Triangulation

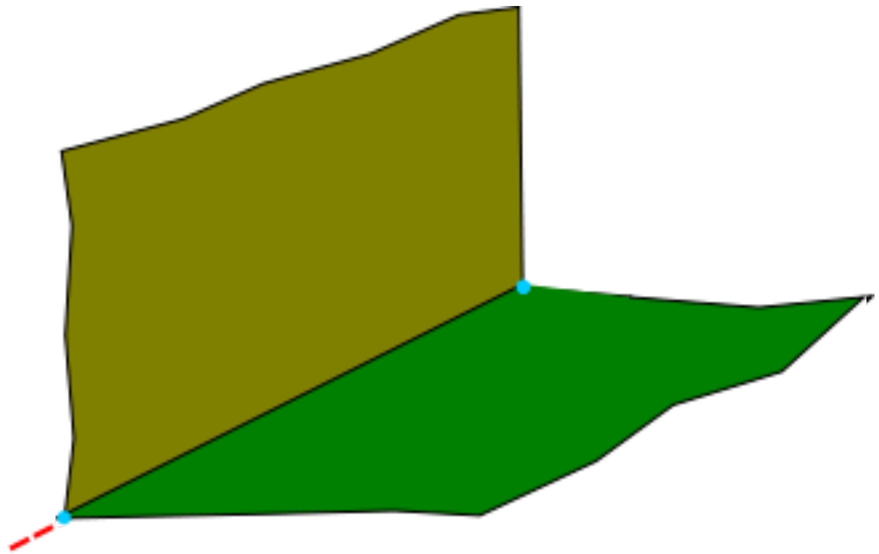
Why doing it iteratively?



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



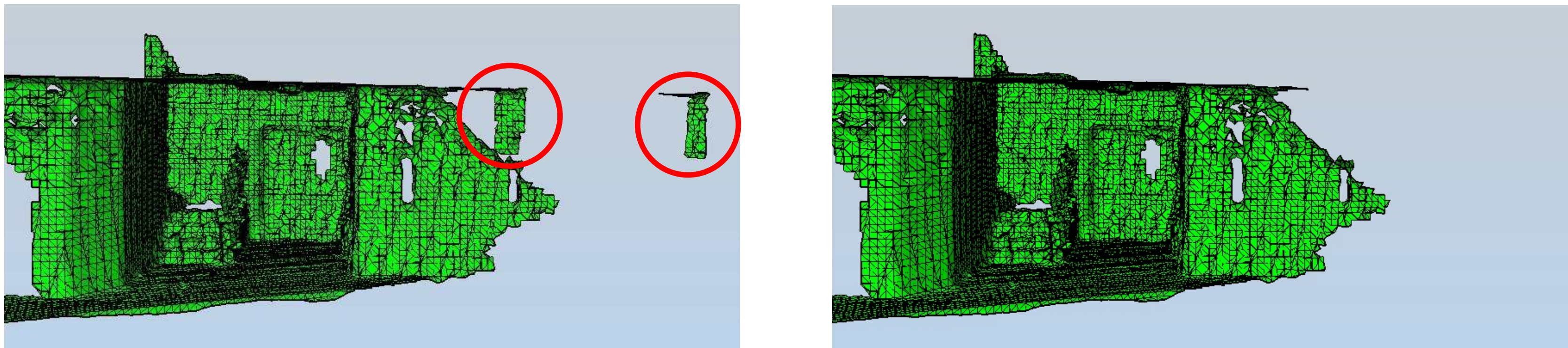


Testing it!

- 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/sick_scans`

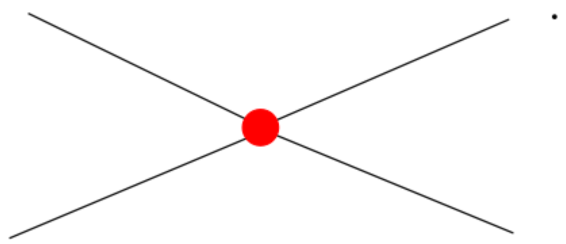
Other Optimization Steps

- "RDA" – Remove Dangling Artifacts
  - Remove unconnected clusters up to a certain size
  - Fill holes in the mesh
  - Delete small regions within the reconstruction
  - … just to use hole filling to kill the newly created holes

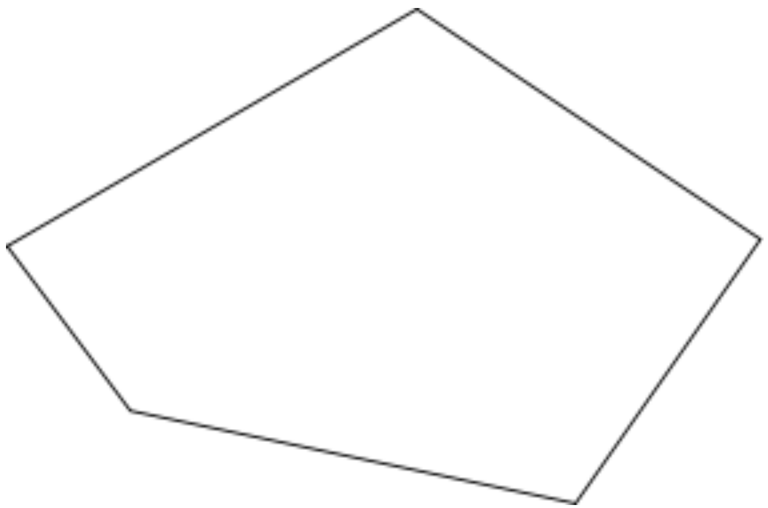


Hole Filling

- Close holes by edge collapsing

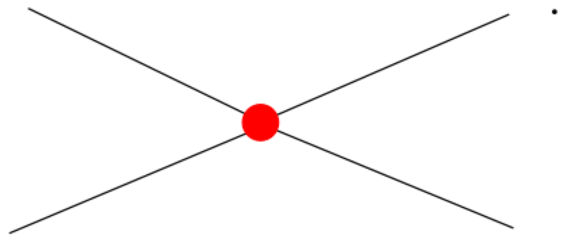


- Example:

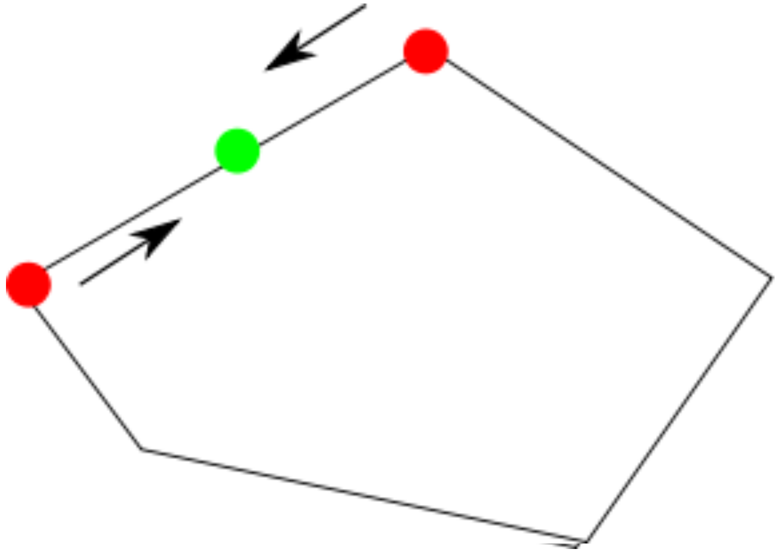


Hole Filling

- Close holes by edge collapsing



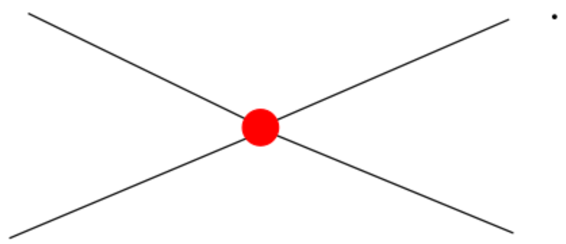
- Example:



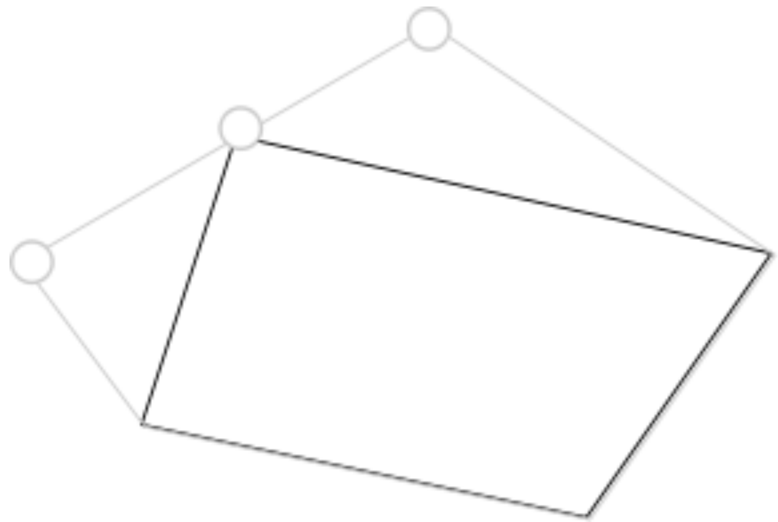
UNIVERSITÄT OSNABRÜCK

Hole Filling

- Close holes by edge collapsing

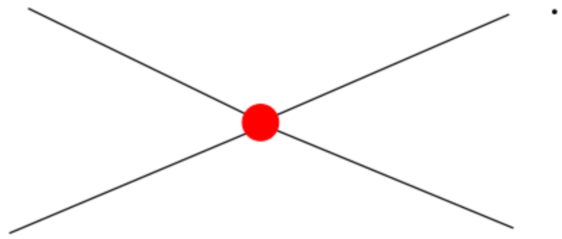


- Example:

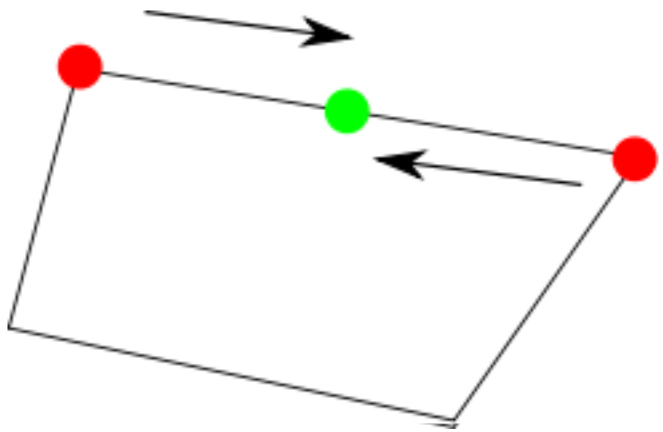


Hole Filling

- Close holes by edge collapsing

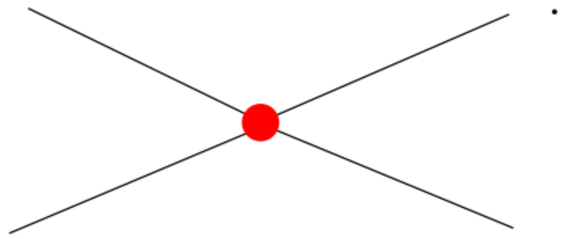


- Example:

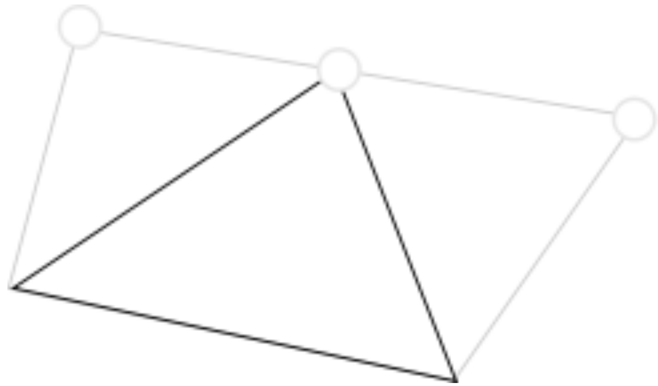


Hole Filling

- Close holes by edge collapsing

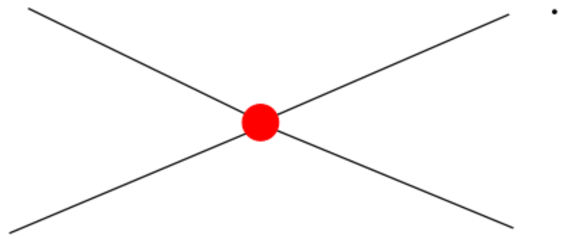


- Example:

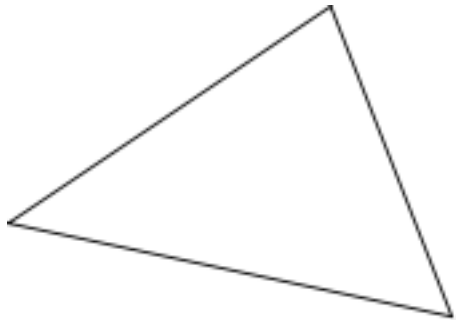


Hole Filling

- Close holes by edge collapsing

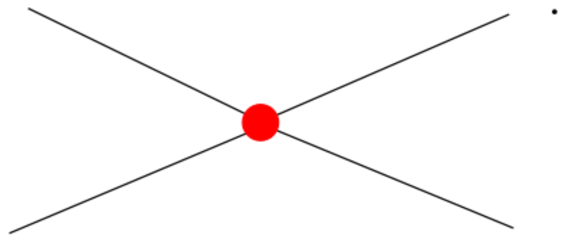


- Example:

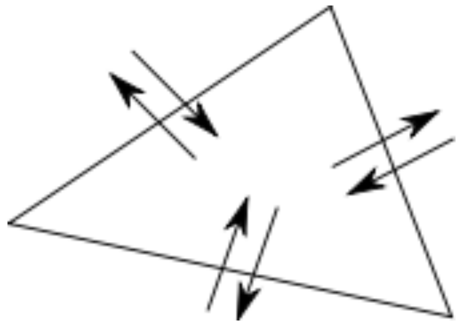


Hole Filling

- Close holes by edge collapsing



- Example:

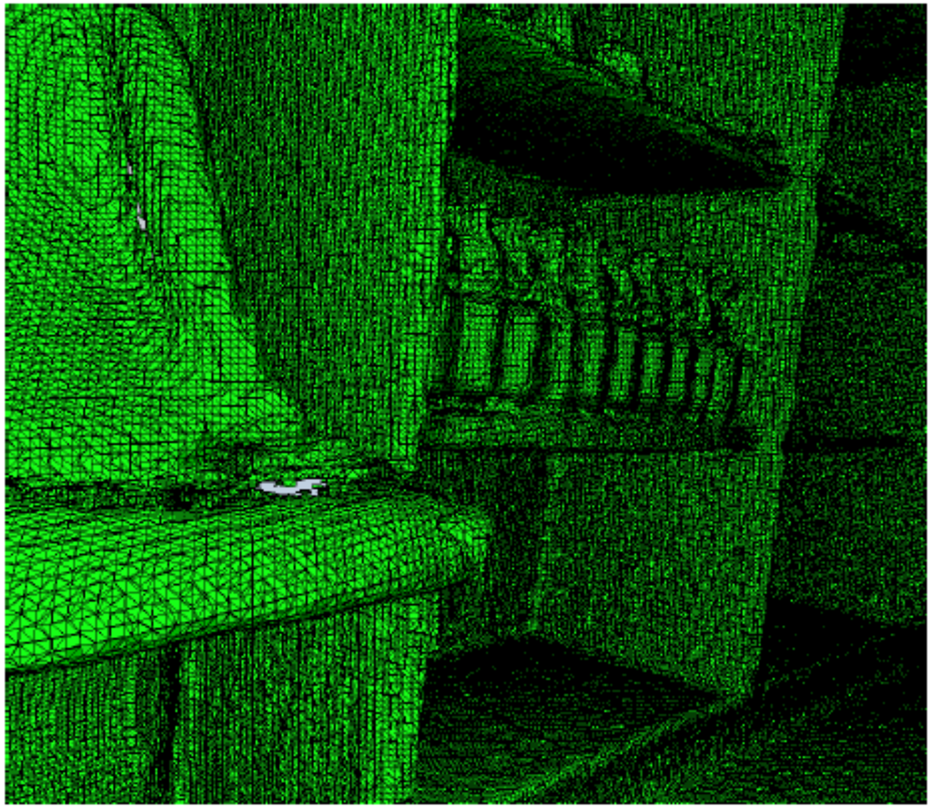


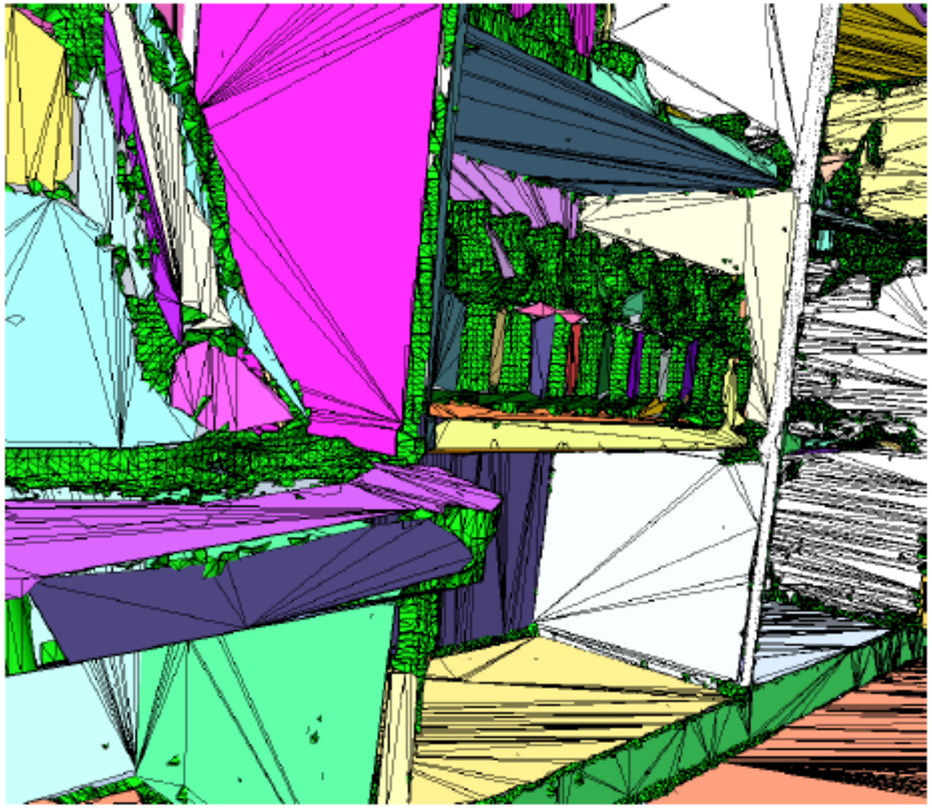
Much more complex in the implementation…

Testing it!

- Relevant parameters:
- `--smallRegionThreshold`
- `--fillHoles`
- Try different parameter sets for yourself on the datasets in `dat/sick_scans`

LVR and Kinect Fusion



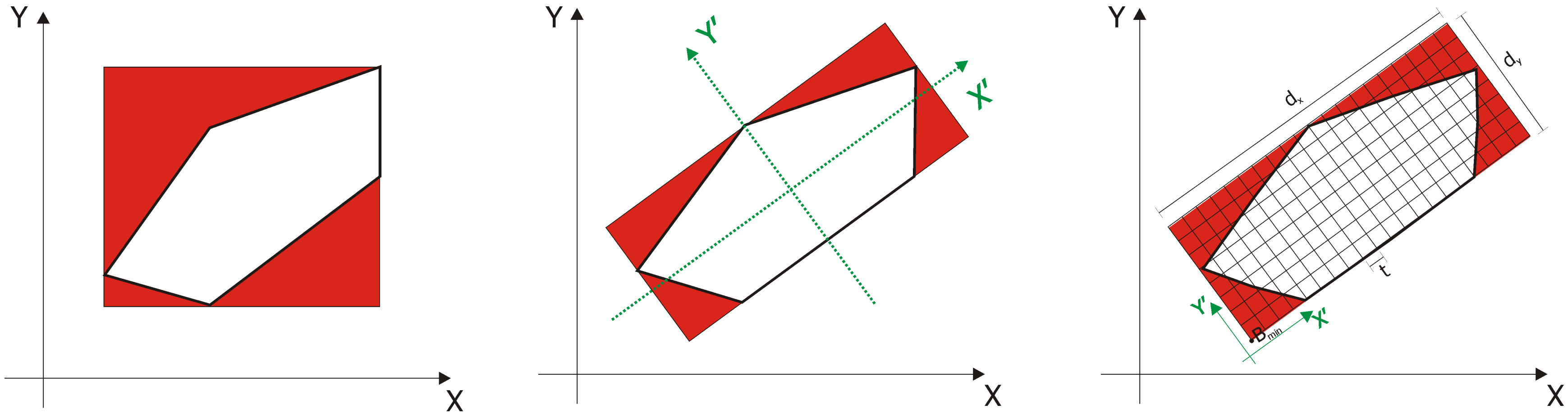


You have to

- … fix the Topologie of the exported meshes
- … then you can use `bin/meshopt` to optimize the meshes
- Fix topology using Meshlab:
  - File / Import / dat/kinfu/mesh01.ply
  - Filters / Cleaning and Repairing
    - Remove Duplicate Vertex
    - Remove Duplicate Face
  - Export the file to .ply
- Notice the numbers
- Test `bin/meshopt` with known parameters on the

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

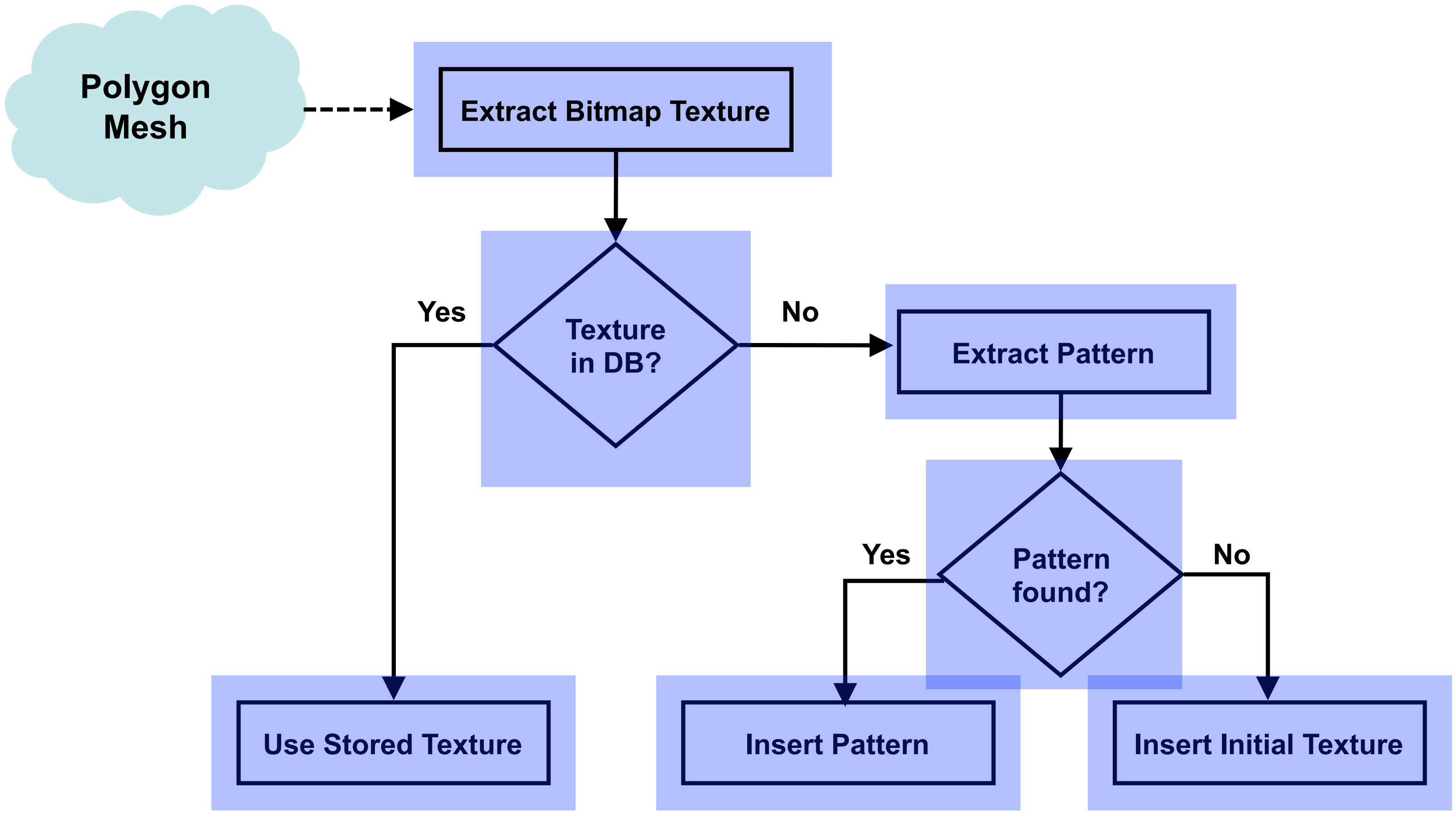


Testing it!

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

```
bin/reconstruct ../dat/texture_generation/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

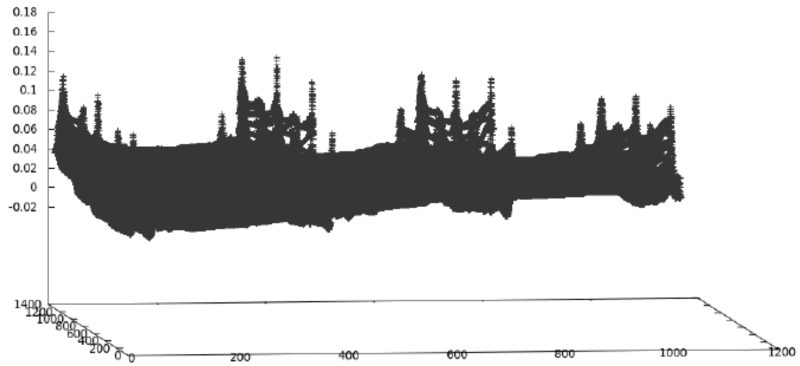
- Approach:
  1. Check with CCM: In case of „no match“, reject.
  2. Otherwise: Combine CC & Features

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:

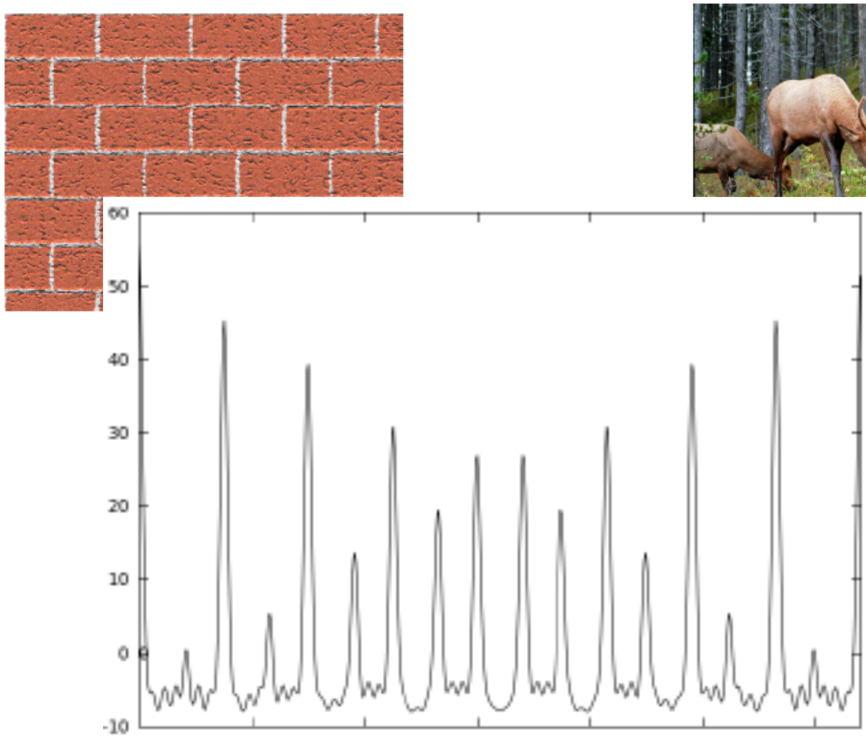
current texture

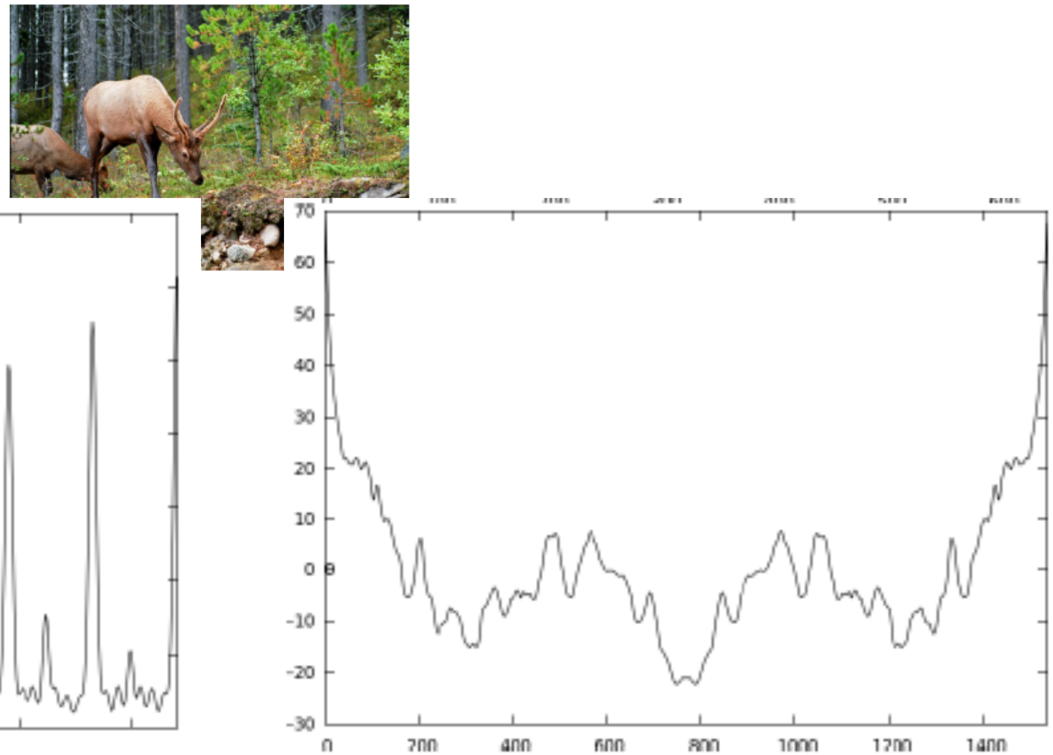
DB



Pattern Extraction

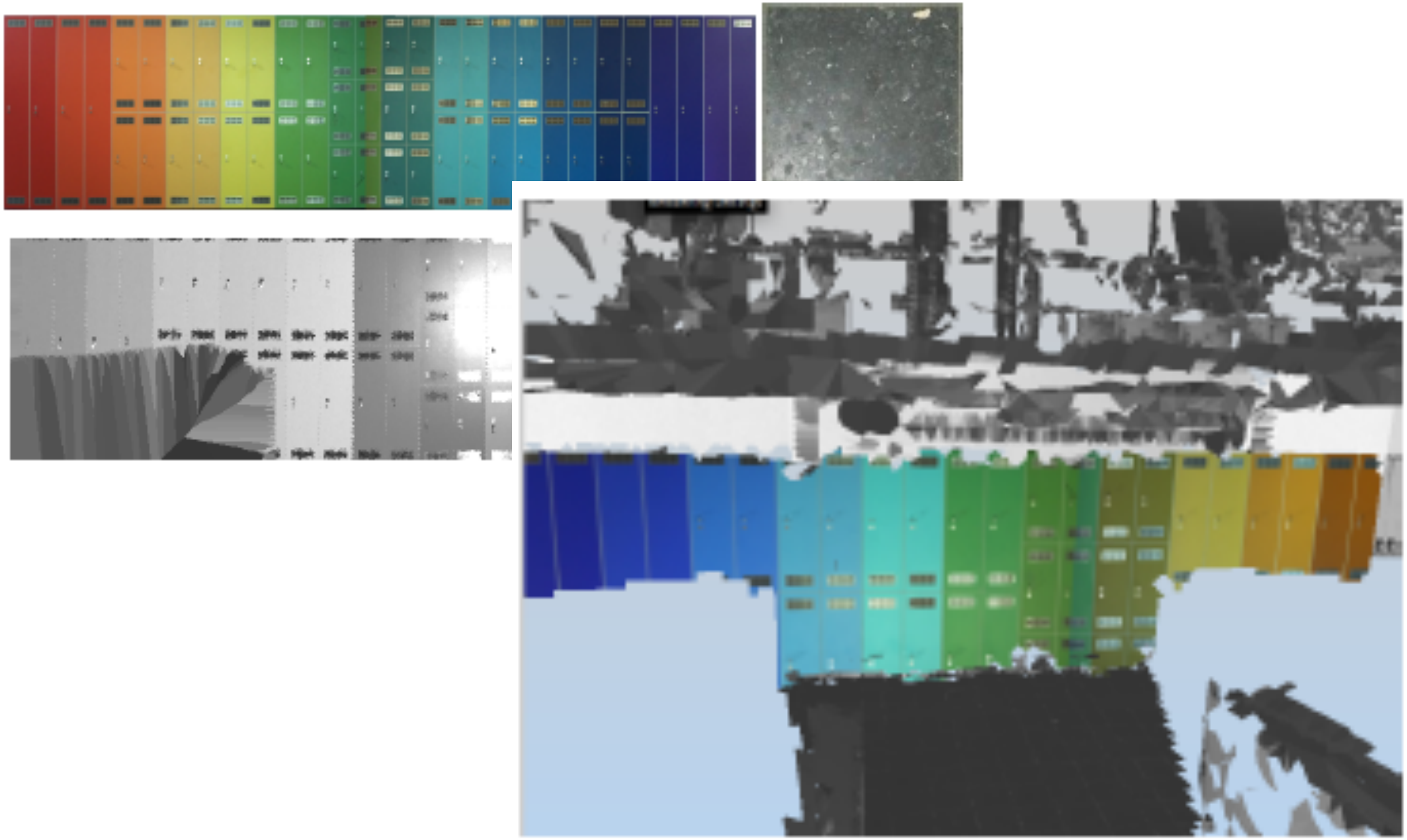
- Does the image contain a pattern?
  If so, where is the optimal cut of that pattern?
- **Pattern check**: auto-correlate the image with itself:



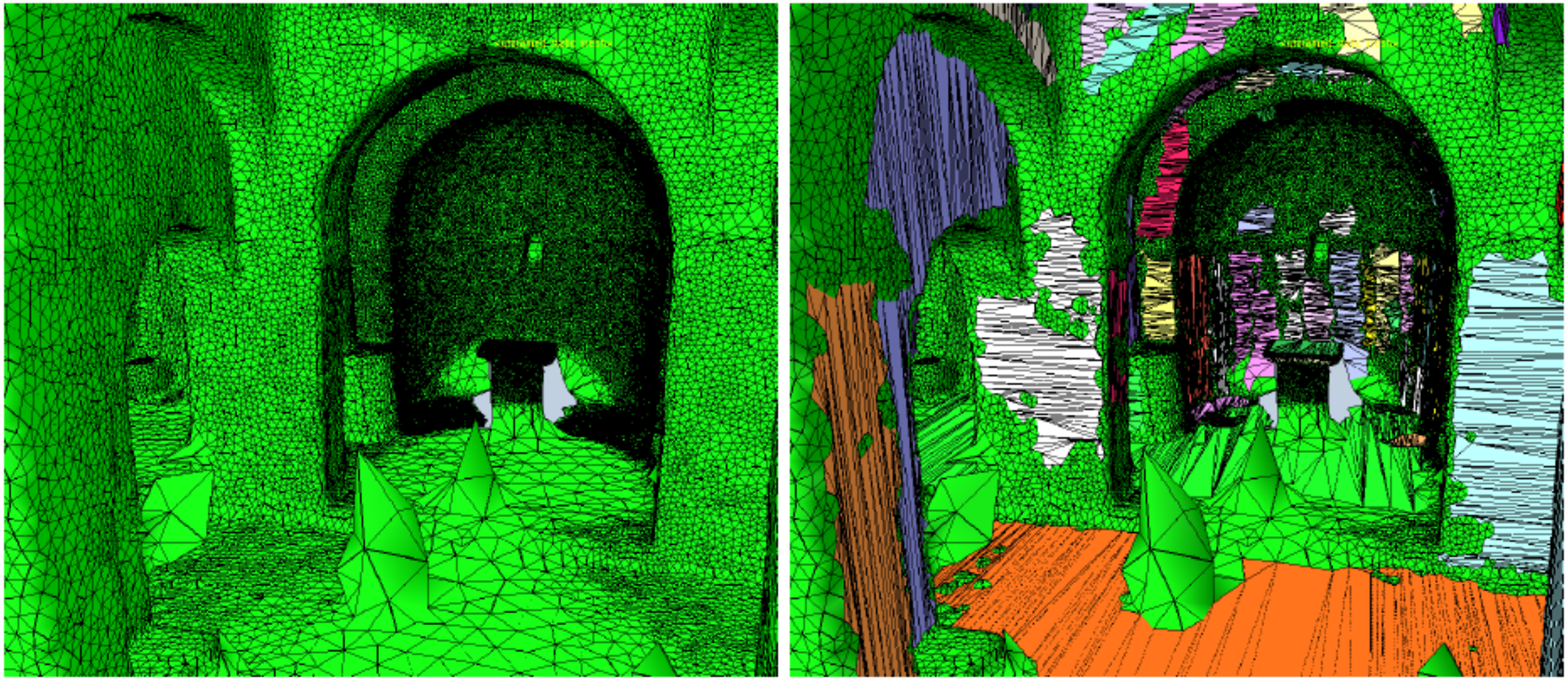


Pattern Extraction

Texture Matching



Comparism with GCS



Comparism with Poisson