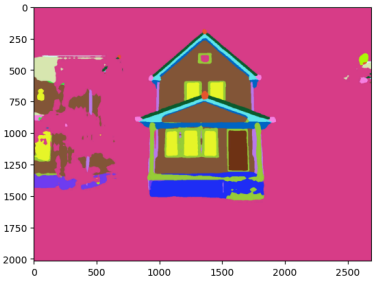
23.0.24

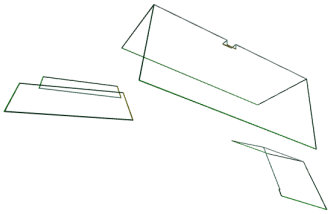
The task seems daunting. For now I will explore the data and get acquainted with it. There are lot of parameters, no documentation, and the parameters cannot be pprinted.
I wonder why authors ignore ade20k segmentation. This could be used to clean up the image. Why didn't they put colour mappings in a file? Instead, they have it in the table on the page. At least it is almost plain HTML. I could write code that scrapes the tables into csv files.
Ok, the segmentation consists of too many classes, many of which aren't related to the competition's goal.
But i guess, i could use some background classes as a blacklist mask to clean up the gestalt segmentation. For example, if it is sky in ade20k, the same pixel in gestalt becomes black.

I implemented this.
It is very slow, despite use of sets for lookup. 10 s for one image. And the output - just a slightly cropped version of gestalt, where important edges are removed and everything that is undefined is black.

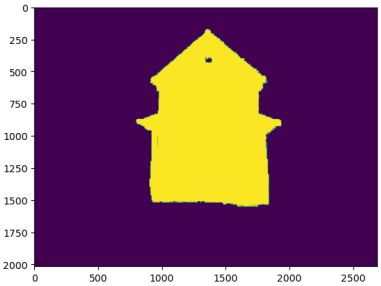
But this failure gives me another idea. Why don't I set everything that is undefined as black?

The result is much more faster and almost the same (except important edges and vertices are preserved)

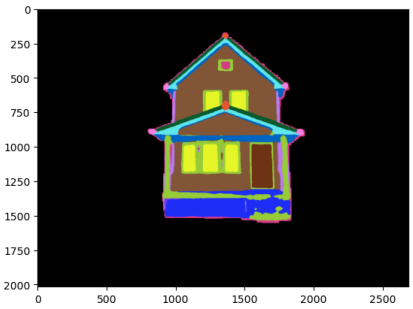




Only one building is in the result. I need to remove the other buildings somehow
Okay, now a function from stack overflow does it for me.



But to remove any holes in the desired object i will MORPH_CLOSE the mask for 11 iterations



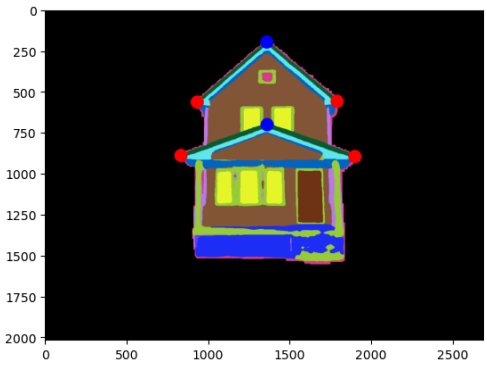
Nice

In order to detect the vertices I need to find the color for the apex and end points.
With this task cv2.inRange(...) is for help
Hm… Despite setting color range nothing gets highlighted.
I will try with a bigger interval - maybe this works.
It works.
As far as I remember we should not use cv2.inRange with RGB. It is for HSV.
Maybe later I will fix it, but now I am satisfied with what I have.

To make the points more round and normal, I will use a series of dilations, followed by a smaller number of erosions.

And finally - connected components to find the vertex centroids

As a cherry on top, here is the image with highlighted vertices



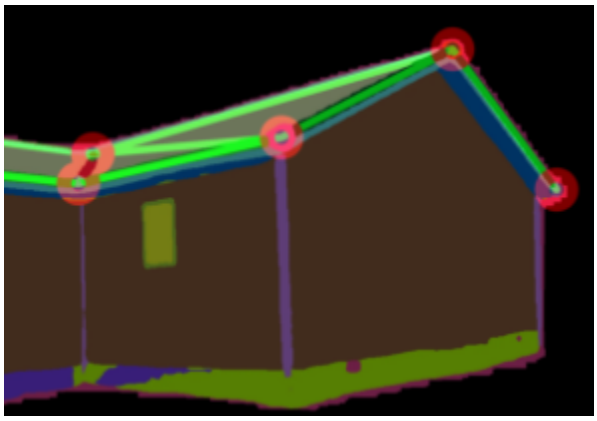
10 submissions a day? Seems perfect! At least for now i won't have to write evaluation code.
I will focus on the proposed solution.

24.05.24

It appears that the color mappings were present in competition's library

25.05.24

I have no idea where are the phantom connections from
E.g.



Maybe there is a flaw in line detection
Its funny to see code suspiciously close to stackoverflow solution
I substituted the line detection algo for hough lines
But “timber framing” problem persists
Although now it looks like it is in the same plane as the roof
And it occurs even when there are no such lines in the mask

It must be the result of distance calculation

I will use squared euclidean since it is faster (but some parameters must be scaled as well)

```
begin_distances = cdist(apex_pts, edges[:, :2], metric="sqeuclidean")
end_distances = cdist(apex_pts, edges[:, 2:], metric="sqeuclidean")
pts_to_edges_dist = np.minimum(begin_distances, end_distances)
```

Why it is the minimum of both??? It vanishes half of the points… that might be the reason for diagonal connections and absent lines

Oh it connects all points in range

It is wrong
One edge end point should connect at most one vertex

Would be nice if circle size depended on the distance

26.05.24

Timber framing still bothers me. I noticed that it occurs when fitting line ends to the corresponding vertex and the algorithm selects single closest point, which doesn't make sense when discussing the image in the context of 3d space. It should select all points that the line end is in range of. And then only select such points that the fitted edge deviates from by some threshold degree
The drawback of hough lines - there are to many invisible lines and sometimes longer lines may cover up shorter ones. Thus there are many lines that are essentially the same. Although the solution with canny edge detection on top is much cleaner, the problem arises when choosing the right line (with this modification algorithm detects edges of the edges, but not the a actual middle line)
Reducing the number of the same lines would greatly improve the performance of the solution. Given that the line ends of the detected lines are close, i could use some sort of clustering algorithm to merge the lines. K means would be perfect, but we don't know the number of clusters apriori.

Also the algorithm that detects the vertices should oslo try and guess other vertices, based on the edges

The point matching algorithm must iterate over every pair of vertices that are in range
What if multiple lines are matched? The one with smallest direction deviation should be selected

27.05.24

I should make the point radius proportional to the blob size.

Grid search did not yield desired results, in the sense that it did not improve the metric much than handpicked parameters. At least now I know which parameters don't work.

From this experiment i shall make the conclusion that the bottleneck is not in the line search part or that the algorithm needs improvements, which exclude parameter picking

28.05.24

After some errors, independent of me , I could run the solution on the test set. A delightful observation – the dynamics between test and validation set are the same, e.g. what performs well on val set does the same on test set.

30.05.24

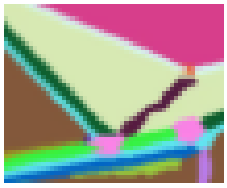
After some pause I can finally continue the work. Now to the missing parts of the segmentation - undetected edges and vertices must be dealt with

Helpful insight would be to see the images where reconstruction loss is more than 3

Weird lime blobs (those are bricks)

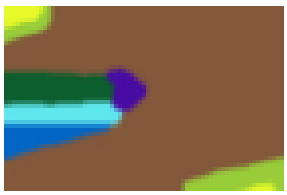
No vertex

A vertex in the middle of the roof



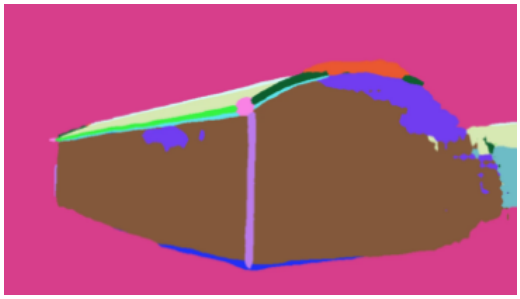
What is the label of the cherry line? Valley? (it was the valley)

Vertex in the middle of the edge



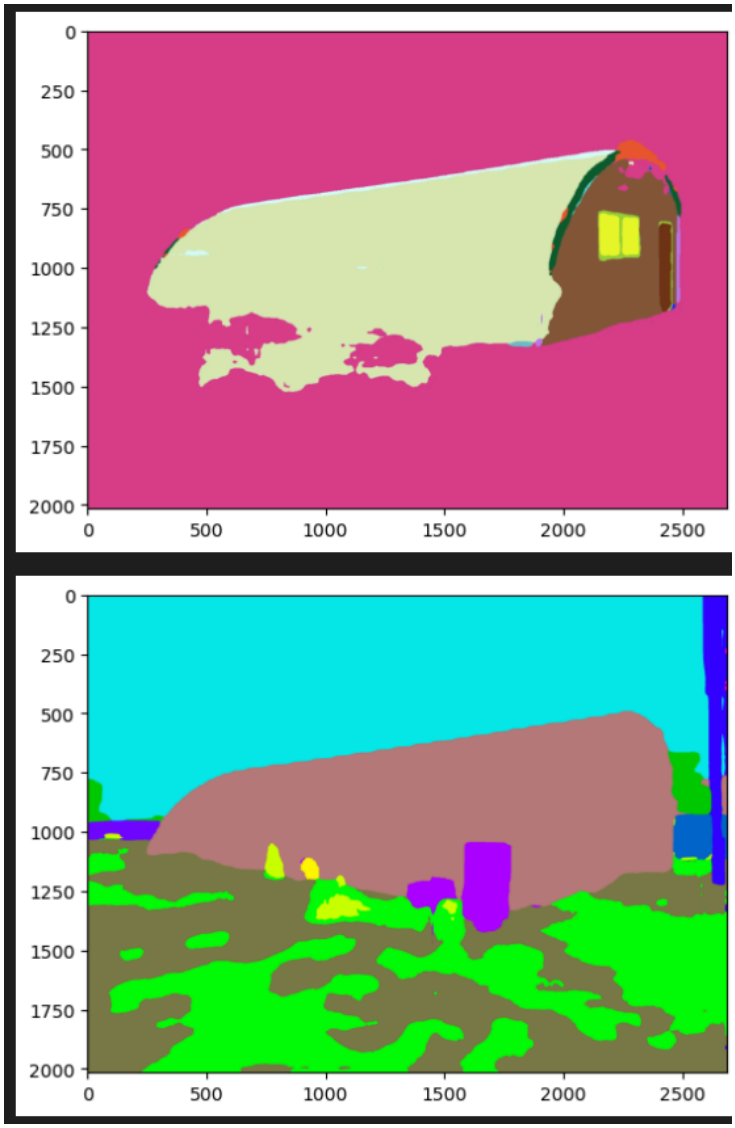
Violet point? (flashing_end_point)

White vs turquoise



Ew

Sometimes buildings are not simple linear wireframes. Curves should also be accounted for. What ade20k says about this building?



Perhaps, occlusion is the reason for such artefacts.

As far as I can see, the missing vertices are between dark green and white edges.

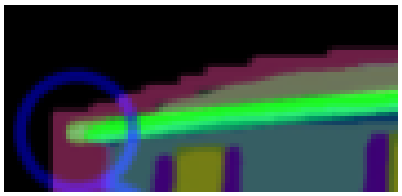
Perhaps, I don't need a complex line intersection algorithm. I could find a way to detect the point where the two colours match and just fill it in with colour for apex

I could do it with template matching.
Or I could separate the image in masks for green and white, dilate the masks, and perform "and" operation between the masks
The second option is more appealing, because there are really a lot of templates for possible line intersection. And template matching overall is more expensive than mask overlapping.And the masks must be found anyway

Hips don't lie or why it appears that they should be included as well

An unwanted intersection point

Also the point must be accepted only if there are no other nearby points. Otherwise the point centroid would be shifted in such blob
Or if I don't want any thresholds I could check if the apex and inferred points mask intersect and reject those that do. Ok, I believe that it is just another threshold , just based on the kernel size and number of dilations
I could combat unwanted intersections with the requirement that the unclassified(pink) area must be over 180 degrees, but now it will require true line detection



Some labels are too thin. There should also be some unclassified edge detection, which later is coloured based on the sparse detected colours.

It appears that KDtrees could be used instead of cdist to get a set of points in range.
From what i can gather cdist is O(n x m x d) where n - number in first set, m – in the second, and d – the number of dimensions.
With kd tree it could be O(log n x m x d)

2.36 is currently the best validation score
Mean running time 01:30

After the gridsearch on merge threshold, I can assure that the value does not matter.

Maybe, i could train a model, that fills in the depth artefacts

There is no improvement. The currently implemented vertex finder does not yield better results. In contrary it is worse. Probably because of the unwanted intersections

And, anyway, the current edge detection only improves the score by 0.01 per day. This is bad

Wow, increasing scale_estimation_coefficient to 3 changed the loss to 2.24
With 4 it is 2.14

(2.39 on the test set)
36-14=22
58-39=19
The change is approx the same

To prove that the scale matters I need to compare how the loss changes with scale for every instance of the val set. If the loss change is monotonic for each image (or has the same shape (up to scale)), then I can just pick a good scale coefficient. In the other case, I will have to make the coefficient more dynamic and figure out which factors would influence it.

In order to afford the luxury of grid search, I need to improve the speed of the algorithm

The requirement of 180 degrees is pretty dumb. A better restriction would be that the both lines end at that particular point
I guess I'll have to add line detection

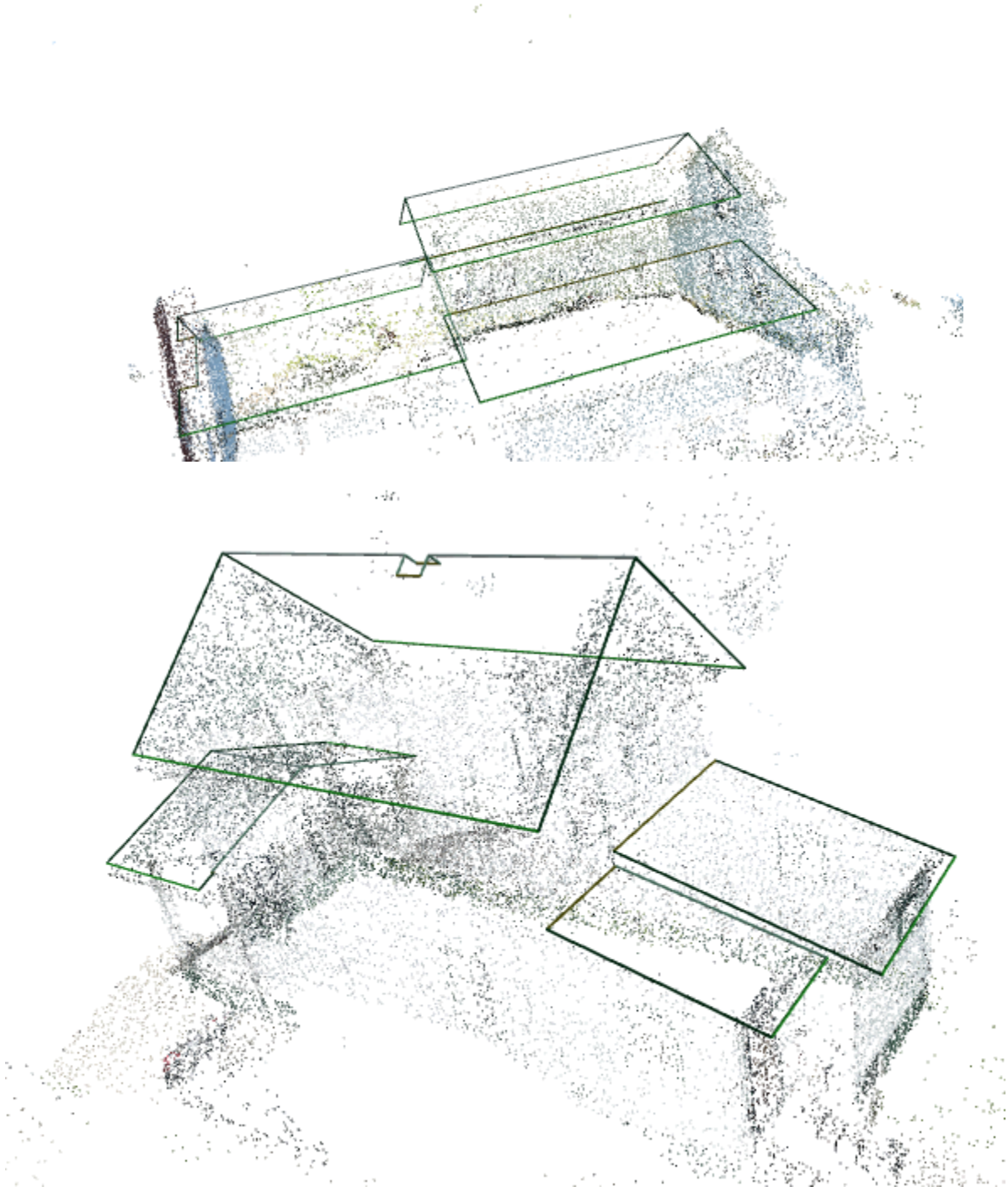
WED_mu score is suspiciously close to the validation score

Without real performance benchmarks I won't optimise much. So far in the segmentation algorithm i managed to shave off 7 seconds

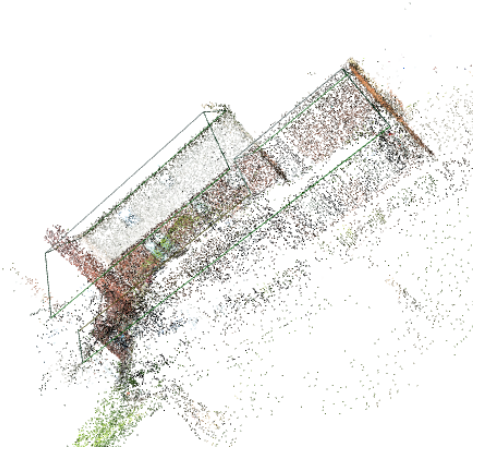
31.05.24

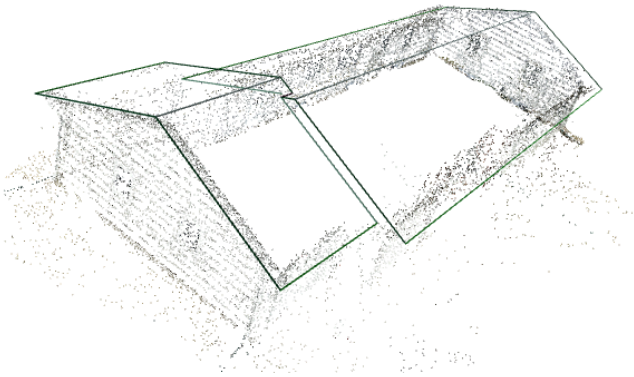
Why does there need to be a coefficient for a depth map? Isn't it already centimetres?
The handcrafted solution does not use point clouds???
I could, at least, use it to correct the depth image.
How do I figure out if the coordinate system is the same
Segmentation camera parameters looked similar to camera parameters, when I examined some examples from the train set. I wonder if it holds true for the whole set. If that is the case, there is a whole new window of possibility.
I could validate the edginess of the edges, or validation of depth maps, or go straight to wireframe estimation. I really need to read that paper on building of wireframes
And what is the scale of the pointcloud? To my knowledge it could be any.
But i also can check

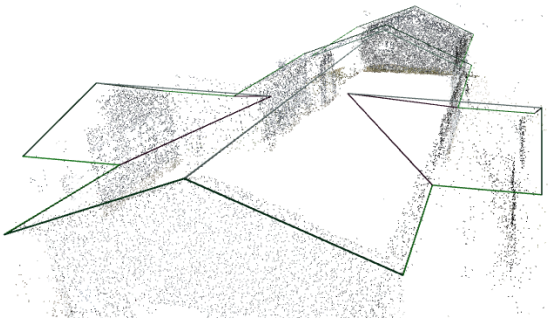
01.06.24



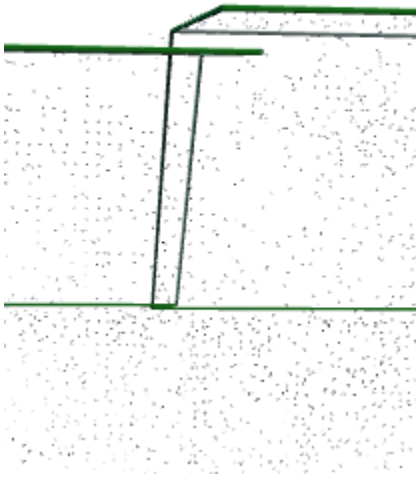
How am I supposed to detect that hole?



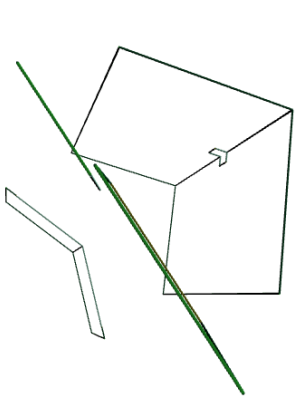




So far the scale of gt wireframe seems to be the same as the scale of the pointcloud.
There is a distinct lack of points on the roofs.
And depth map denoising is only required for the roof parts



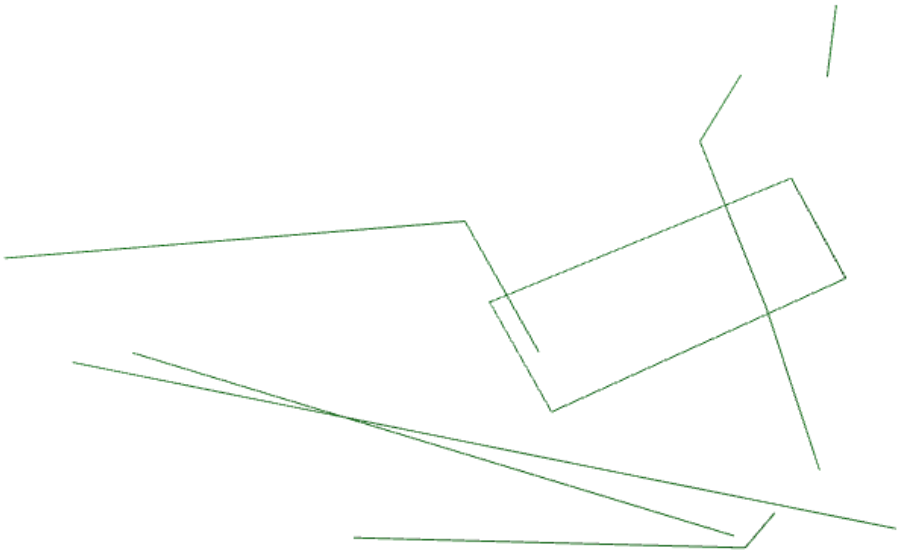
What is that overhang??



Many disjoint parts. I wonder if “prune_not_connected” messes with it - i need to see the prediction

That is such a mess, jeez

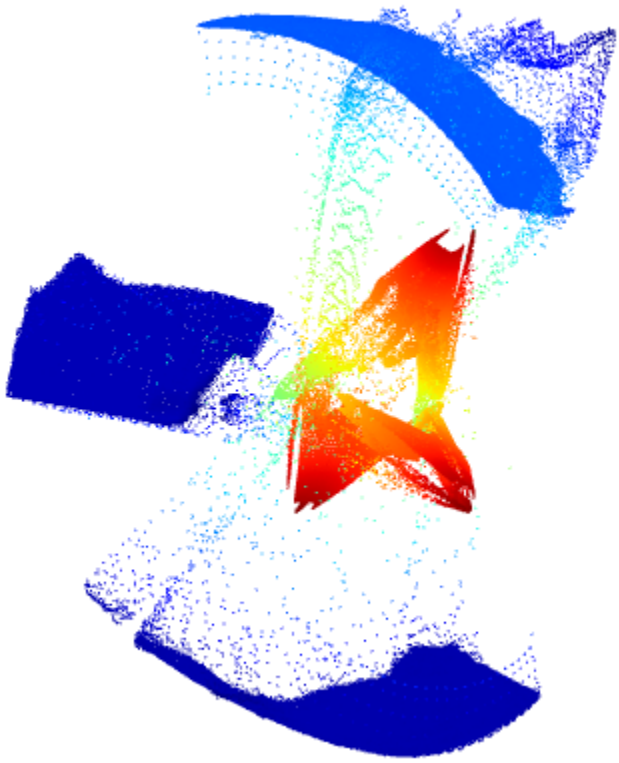
What is this



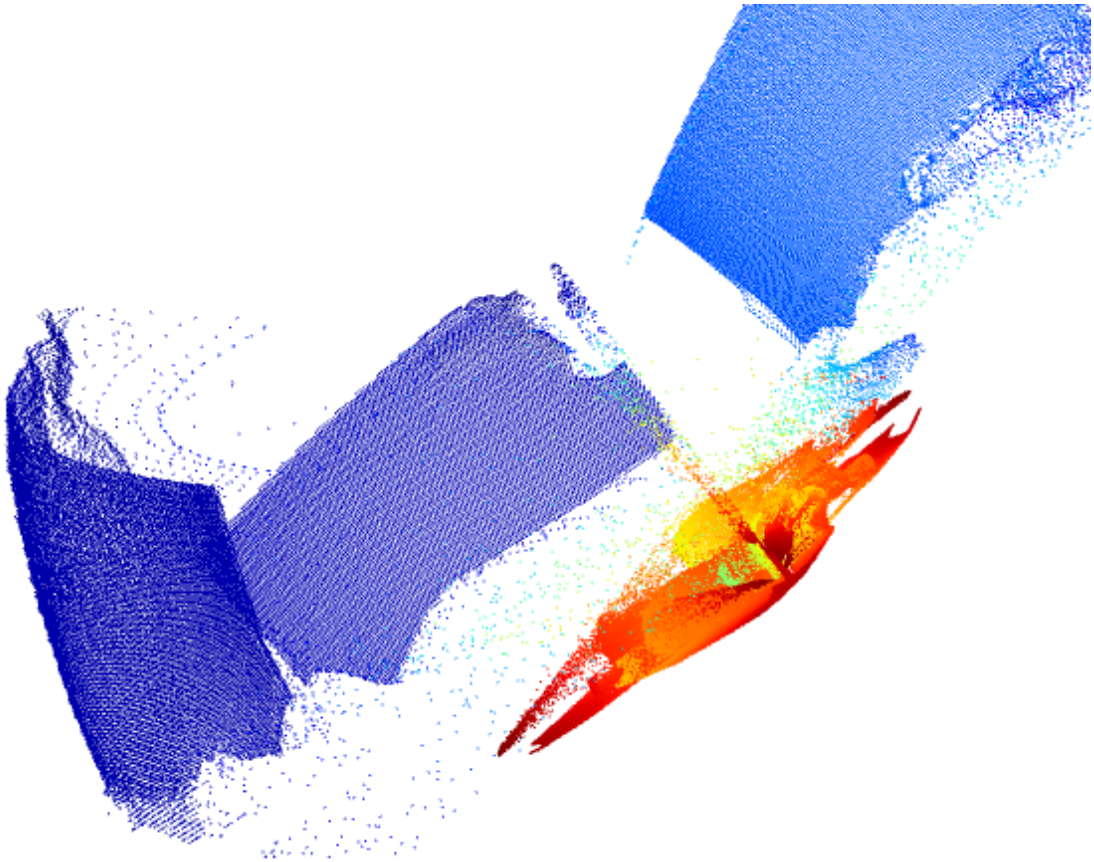
I would be so angry if someone had me work with this

This is what it looks like at best
The depthcm scale coefficient looks suspiciously close to conversion rate between inches and centimeters
To be fair, I just visualized with the wrong parameters

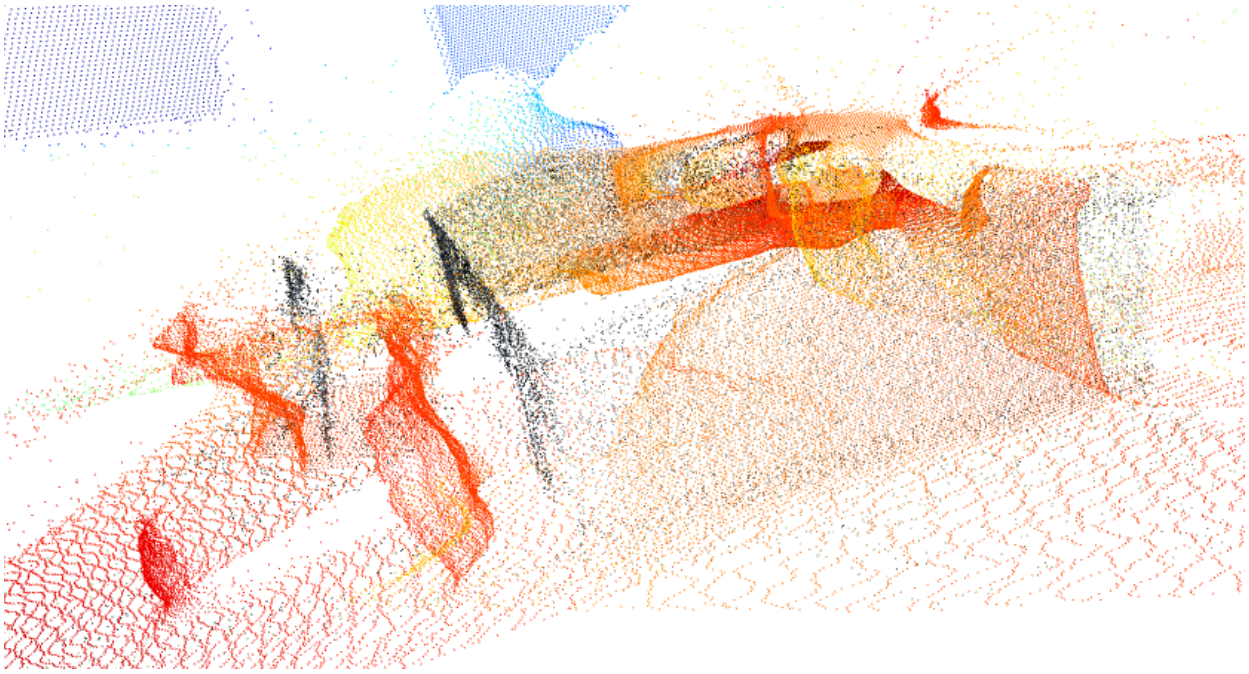
It didn’t change much
Although it made a stickman dunking a basketball
What if I remove all pruning and merging?
Nothing changed
“Images” and “cameras” are missing in the reconstruction



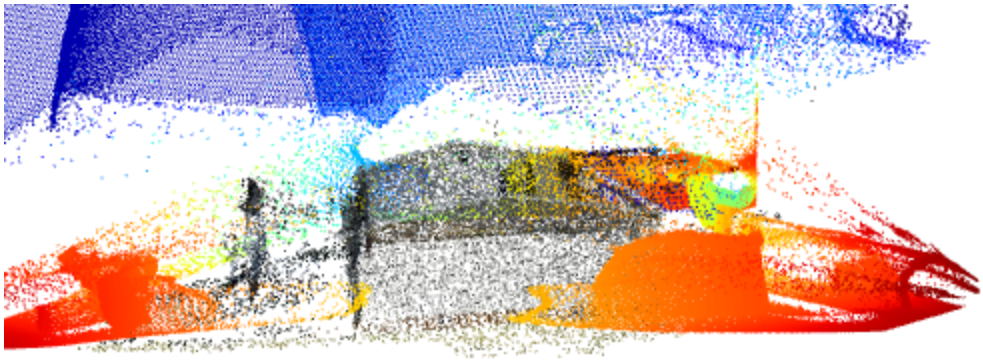
No, it is not a biblically accurate depiction of an angel,this is how the depth maps are projected onto 3d space with the handcrafted algorithm



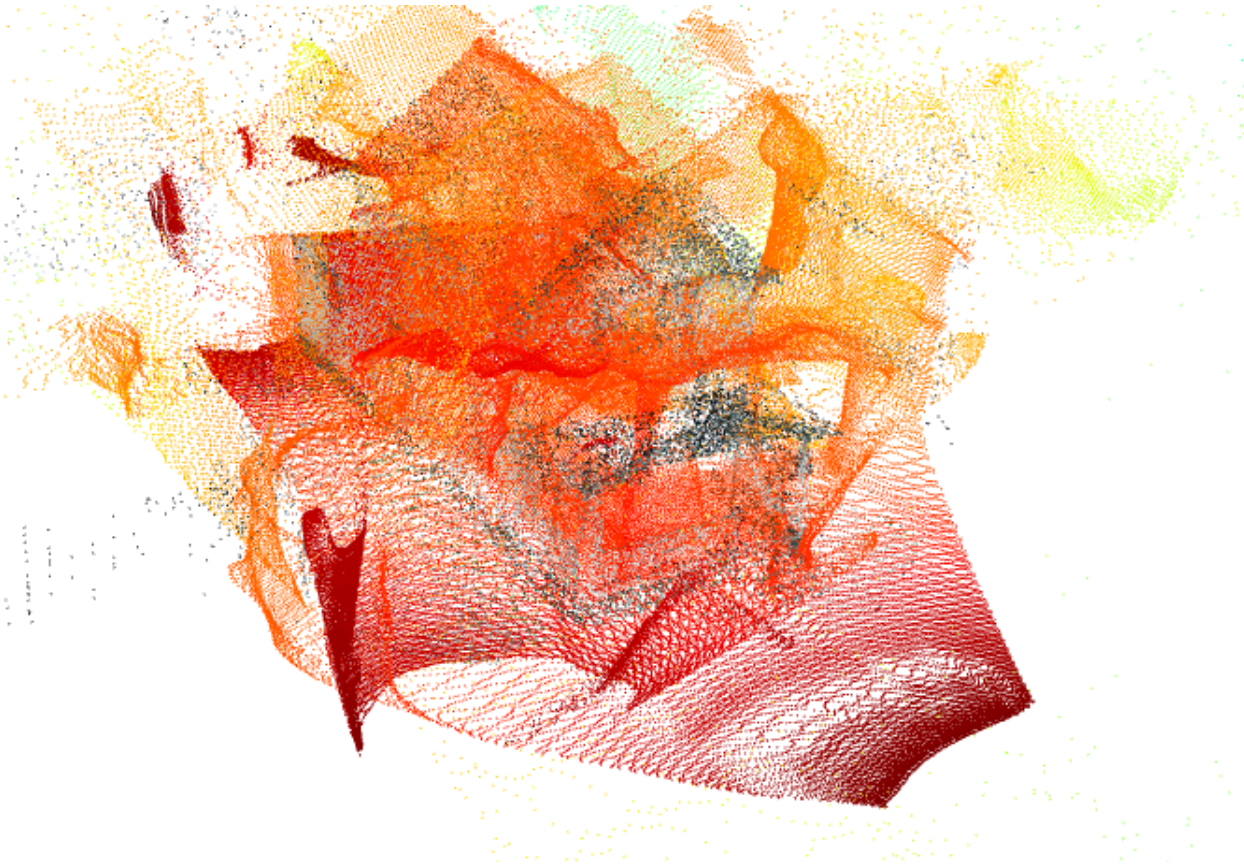
It was me who messed up the indices



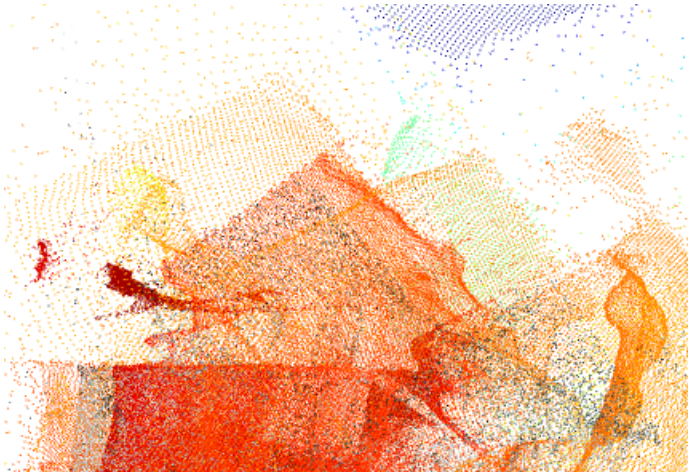
Depthmap projection compared to pointclound, scale factor 2.54

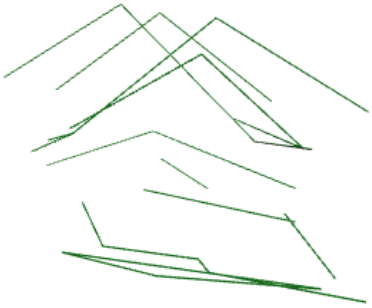


Factor 4.5
And this preforms the best, despite the obvious deviation from the truth

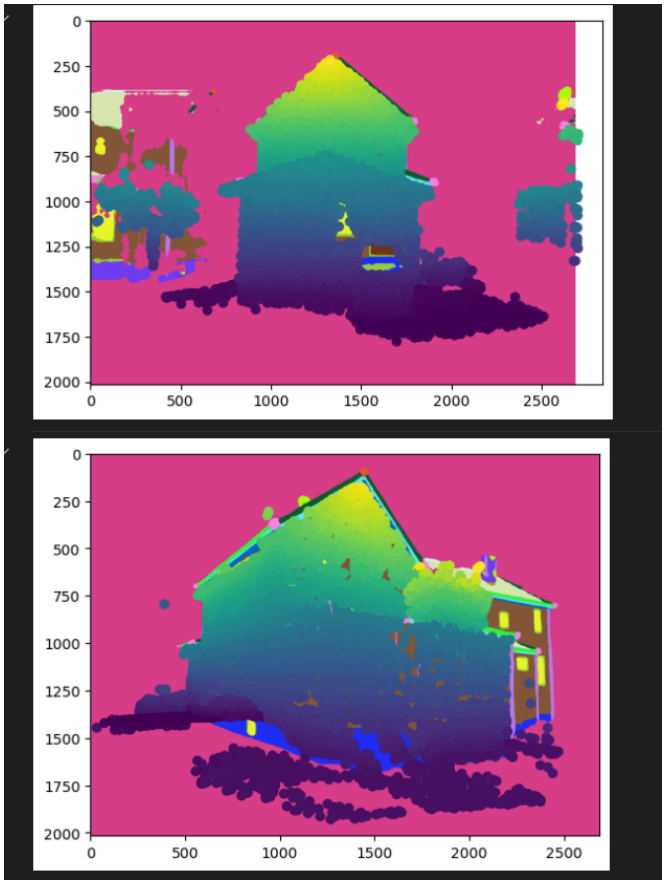


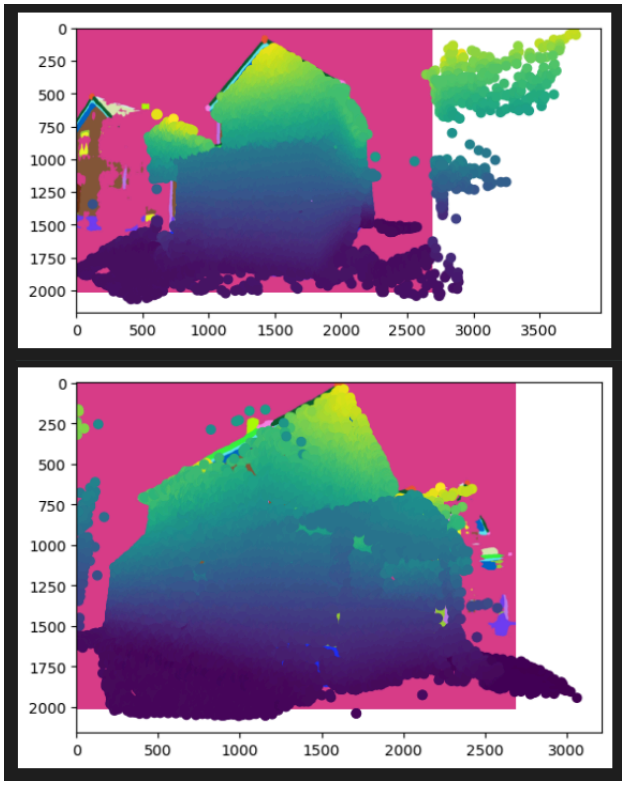
Depthmaps are extremely bad
The only valid reason to use them is that they are defined in almost any point and they show a roof





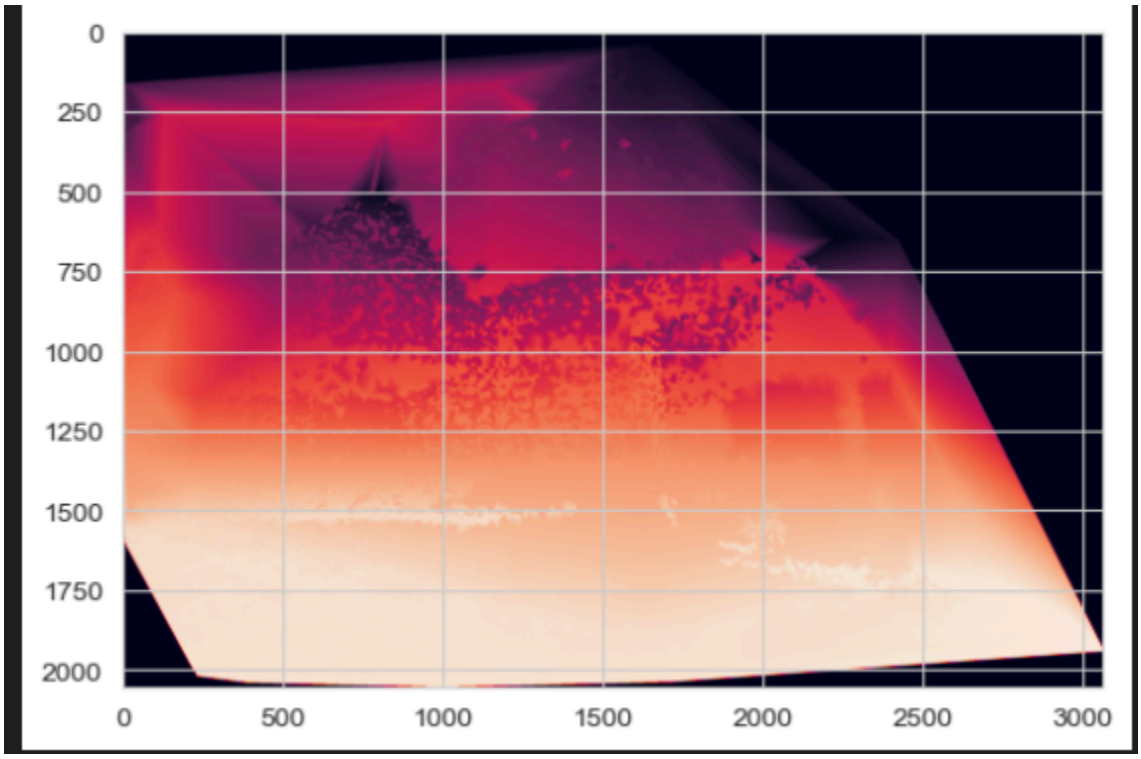
I can actually see where these lines form
As they say, garbage in, garbage out
Given that other contestant’s scores are not much further than mine, I believe that they face the same problem. It is unknown whether anyone discovered this particular issue



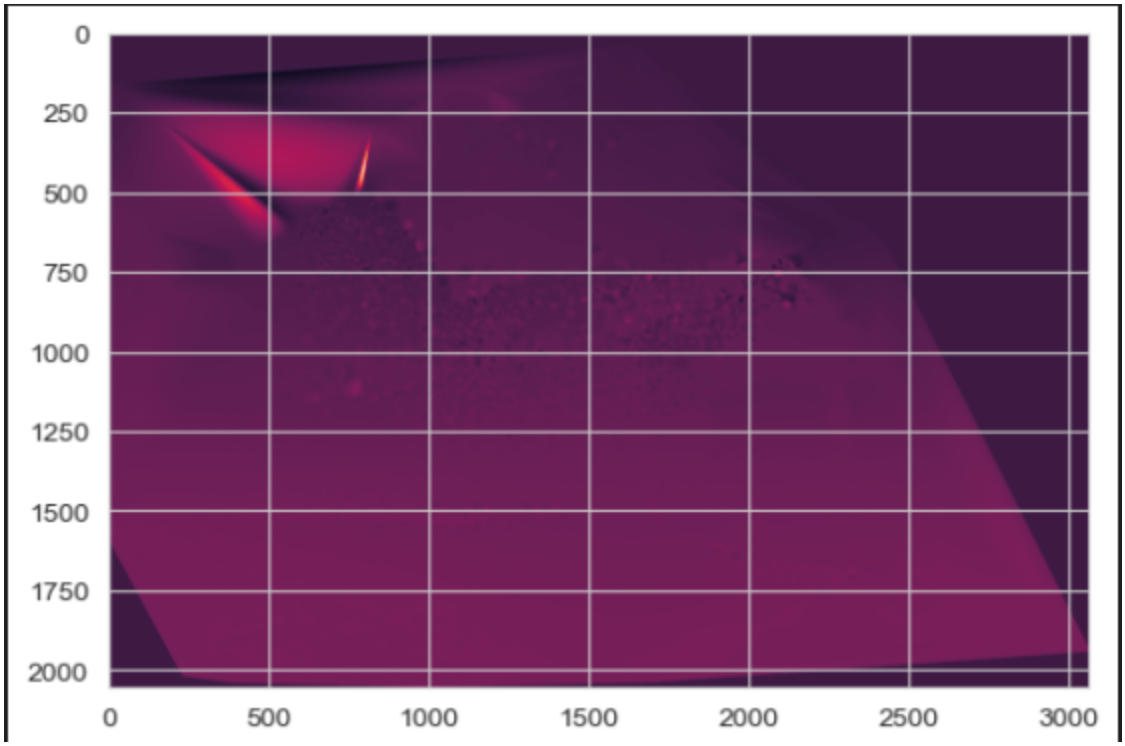


So far so good. And these are real points.

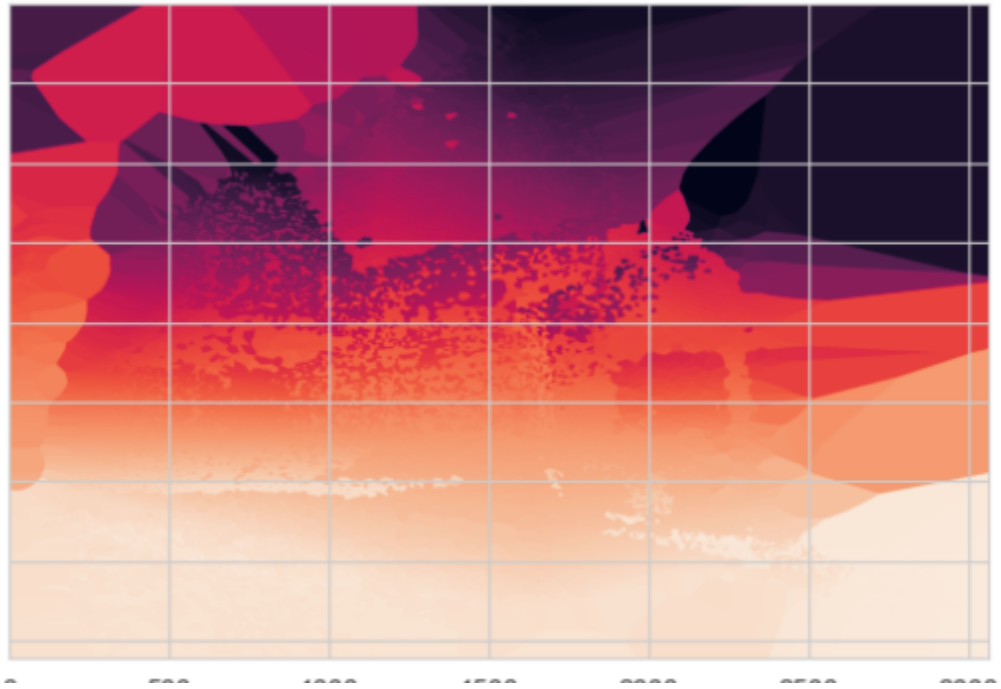
I could remove points from the pointcloud that are not connected to anything
But before that I need to check how this updated depth map works.
Also I could keep the distances as z coordinates of the pointcloud, since it is all the depthmaps does.
Now the following questions arise: what do I do with the missing roofs and how do I render the depthmap? Squares or circles that fill in a certain value? Do I interpolate? Perhaps, some linear interpolation would work best since buildings are fairly linear



Linear interpolation



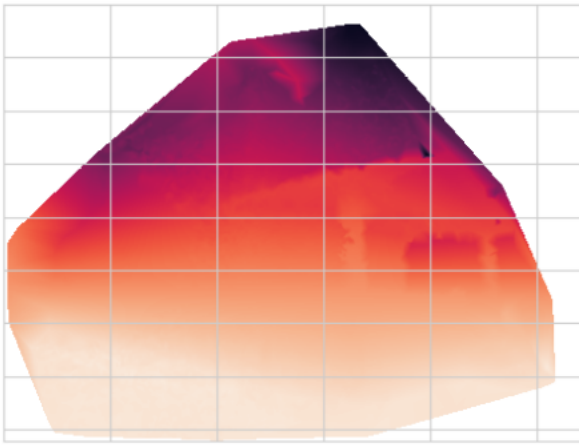
Clough-tocher



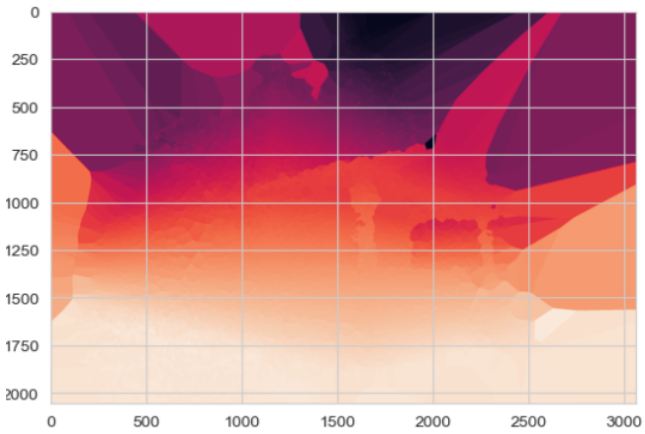
Nearest neighbour interpolation

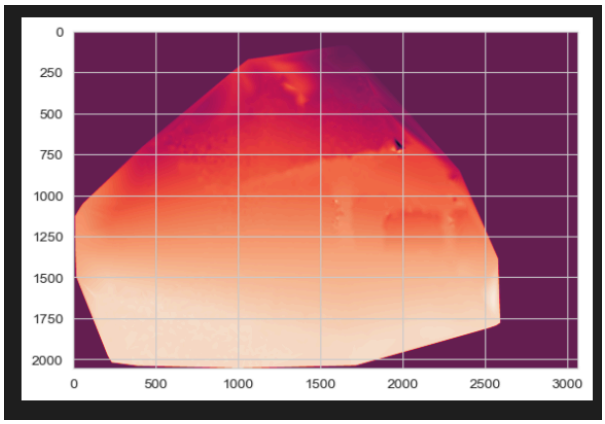
I do not like how spongy the results look
It is as if the missing space is not filled enough
The scatterplot of points is almost what i want - but it isn't measurable in any way and some edges are empty

Also, for optimisation, I could interpolate only near the vertices



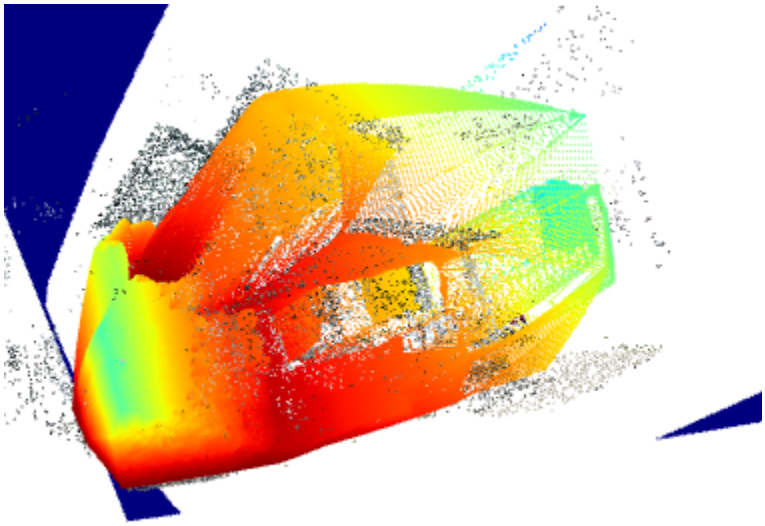
Now we are talking. I had a bug that essentially interpolated on the whole pointcloud, instead of the projected points



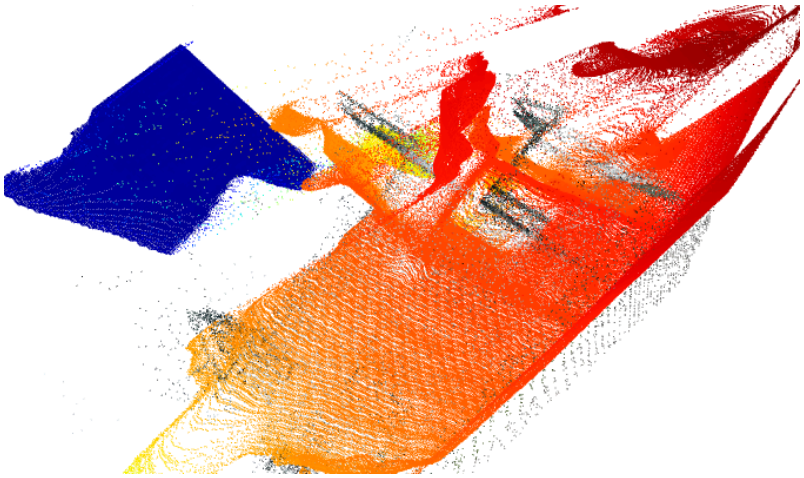


I will use nearest neighbours since it is fast

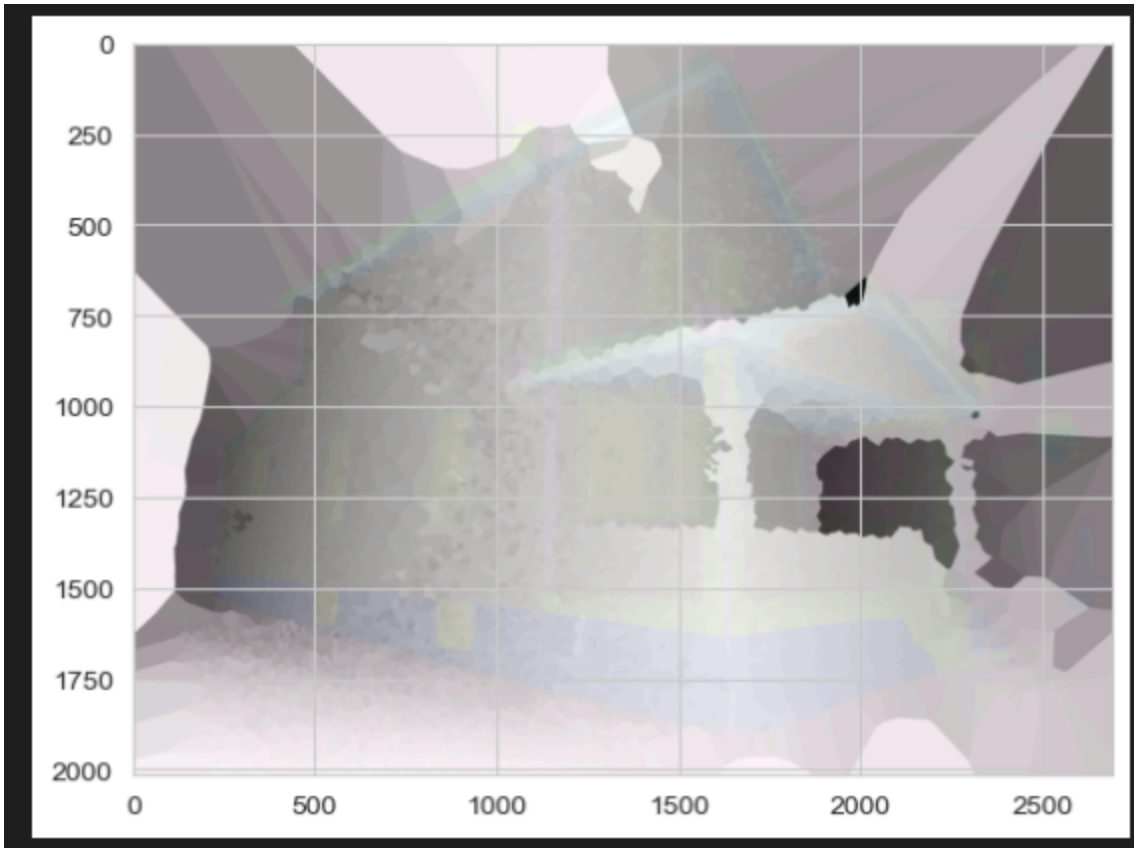
02.06.24



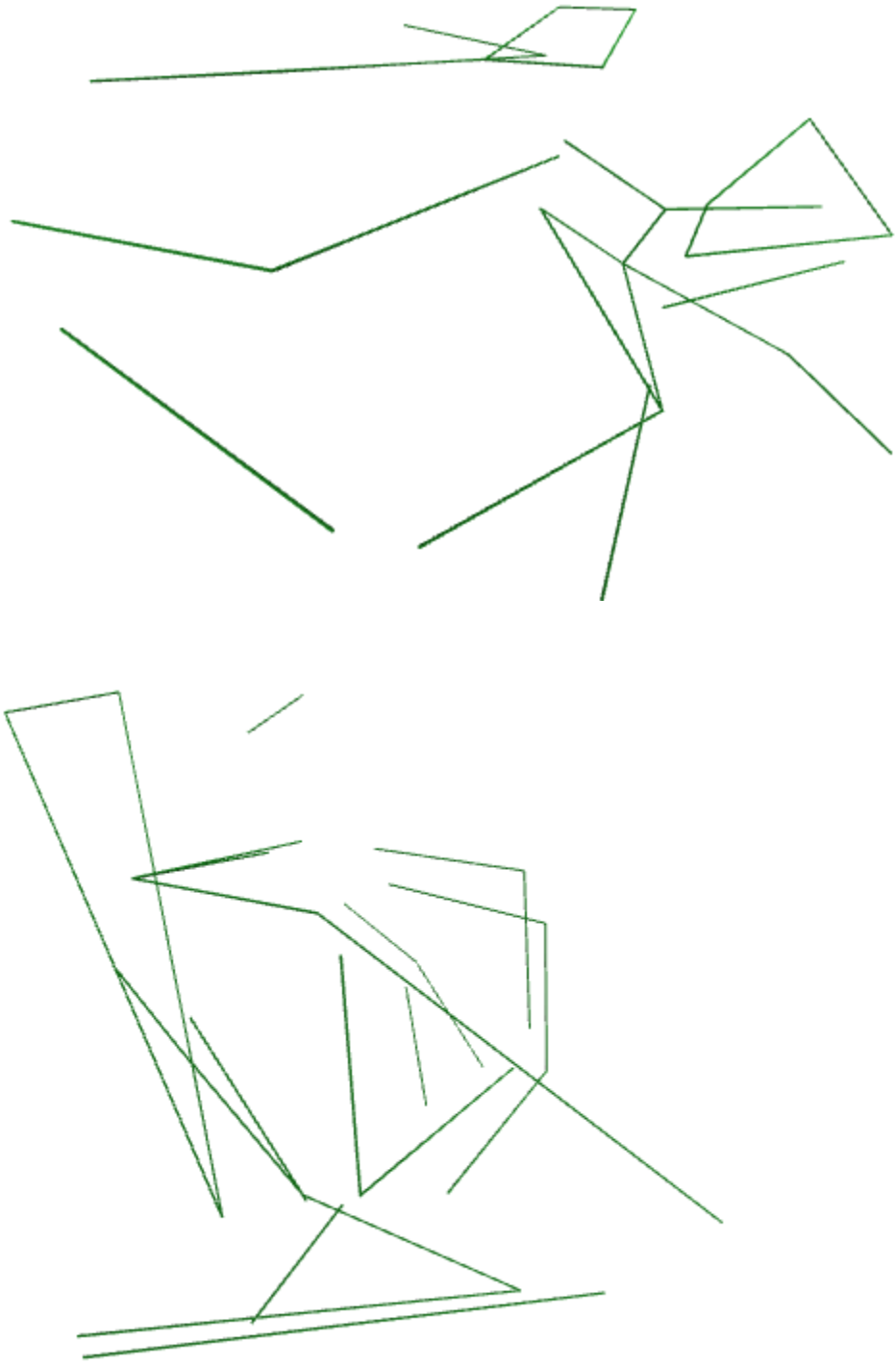
This is a lot better than a depth map



(the depth map)



I need to se if homogeneous coordinates would yield better results
I quickly and dirtly transferred the code from the notebook
Old best 2.123433669870156
New best 2.080892197643399



Still a mess but is a lot closer to the truth
I believe that it is because not every image has the corresponding colmap image - so I need to decide which points belong to the image without leaking any through

The merging threshold had no effect because it was to little

9- {"WED2": 2.2365818170232257, "WED_mu": 2.022978236529062, "WED_p5": 1.3706636650342114, "WED_p25": 1.6818073005037268, "WED_p50": 1.9635509039881809, "WED_p75": 2.3662294499896226, "WED_p95": 2.723216310552435}

I- {"WED2": 2.3758007736394062, "WED_mu": 2.142185903729573, "WED_p5": 1.5188658485477358, "WED_p25": 1.8632011869613698, "WED_p50": 2.1514899711483464, "WED_p75": 2.390973780053338, "WED_p95": 2.833726446040298}

- {"WED2": 2.361242166750851, "WED_mu": 2.131100231249114, "WED_p5": 1.513840126403259, "WED_p25": 1.844103202803491, "WED_p50": 2.141487646549891, "WED_p75": 2.3866591905873755, "WED_p95": 2.811138622453549}

What is the difference between wed2 and wed mu?
Ok, I am blind. I thought that 2.236 is 2.36.

rank	id	WED2	WED_mu	WED_p5	WED_p25	WED_p50	WED_p75	WED_p95
1	Ana-Geneva	2.2317	2.0611	1.6082	1.8417	2.0254	2.2011	2.6087
2	Siromanec	2.2366	2.023	1.3707	1.6818	1.9636	2.3662	2.7232
3	rozumden	2.3195	2.135	1.5368	1.8946	2.1056	2.3642	2.7178
4	kcml	2.4531	2.2423	1.58	1.987	2.2484	2.4979	2.8687
5	Yurii	2.6283	2.3893	1.6307	2.0866	2.4184	2.6716	3.0773

I finally moved up
I have the smallest WED_mu

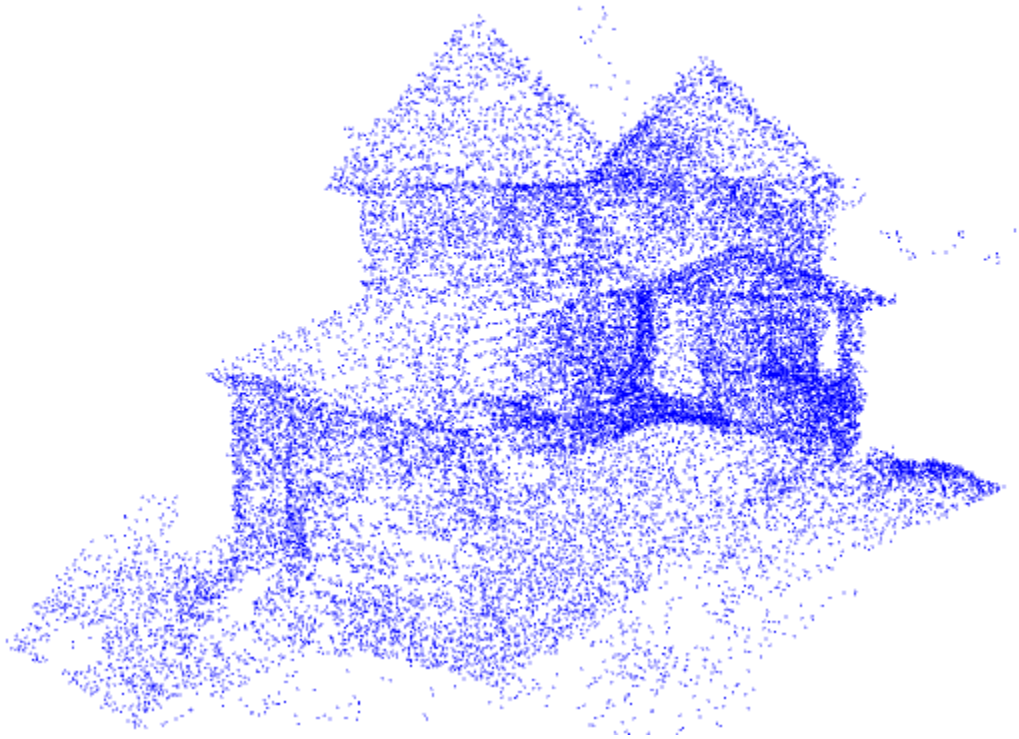
Also the distribution of WEDs is more spread out. Which may mean that there are more edgecases for my algorithm. The problem is to identify such cases and handle them accordingly
Maybe I need to rerun the grid search.

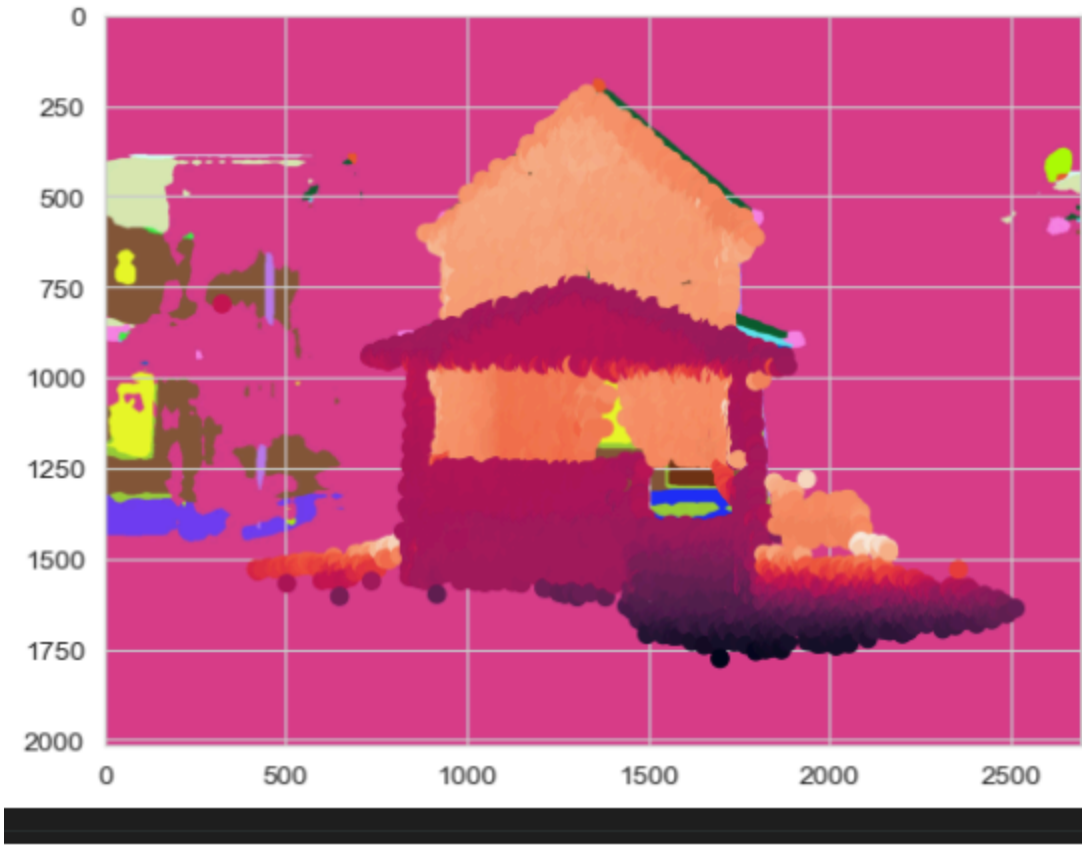
But before that I will do clustering of the pointcloud in order to remove unwanted objects.

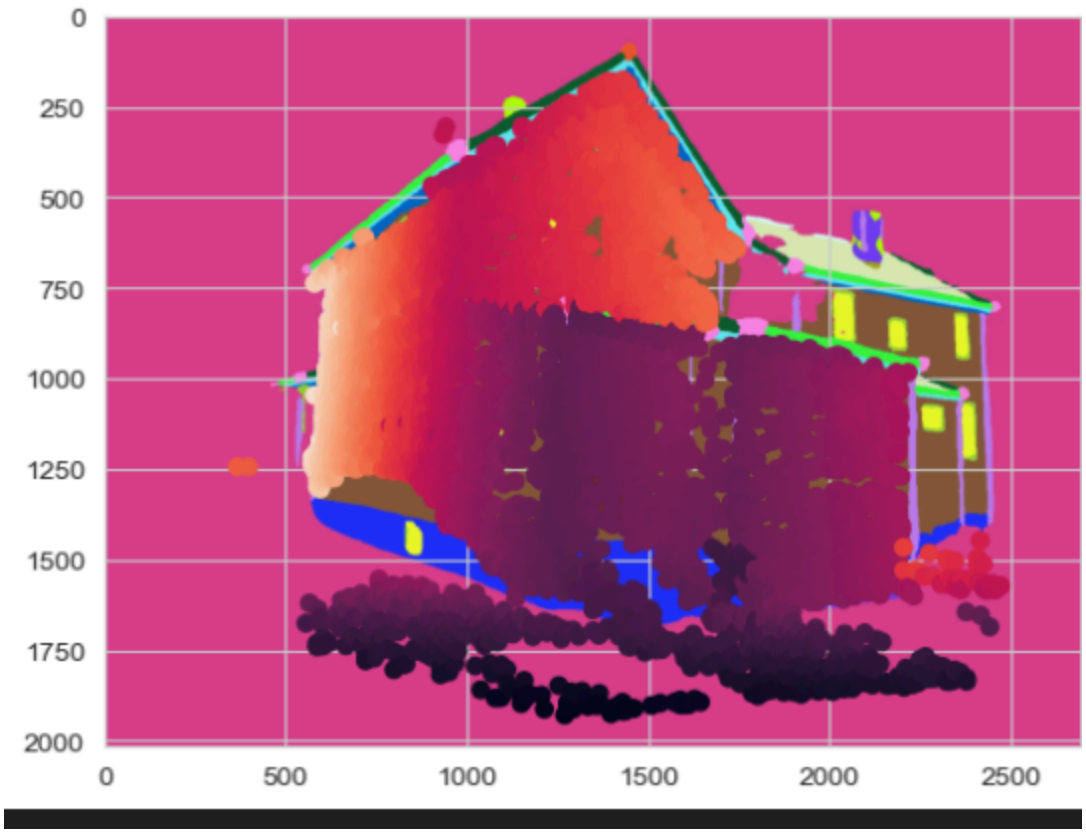
	Public Score	Submission Comment
c-4	{"WED2": 2.2610446233021495, "WED_mu": 2.0573966695586883, "WED_p5": 1.4974253279118122, "WED_p25": 1.7890735079844347, "WED_p50": 2.0580992380753864, "WED_p75": 2.3047831100496423, "WED_p95": 2.6676379622723743}	merge_th=3
09-1	{"WED2": 2.2365818170232257, "WED_mu": 2.022978236529062, "WED_p5": 1.3706636650342114, "WED_p25": 1.6818073005037268, "WED_p50": 1.9635509039881809, "WED_p75": 2.3662294499896226, "WED_p95": 2.723216310552435}	some depth maps are now retrieved from the pointcloud

Merging has some impact. Maybe there is a modification to the original algorithm
May sound outlandish but what if i add chimney as another vertex class

Added pointcloud clustering and now it is a lot cleaner




0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000
2500


0
250
500
750
1000
1250
1500
1750
2000
0
500
1000
1500
2000
2500

Perhaps i have to avoid clustering since it has some insane memory requirements (n^2 for a point cloud is alot)

I made a compromise. Point Clouds beyond a certain size are unclusterable.
New best 2.0244502584047663

rank	id	WED2	WED_mu	WED_p5	WED_p25	WED_p50	WED_p75	WED_p95	submission_datetime
1	Siromanec	2.199	1.9788	1.3114	1.6497	1.9427	2.2892	2.6754	2024-06-02 15:43:45
2	Ana-Geneva	2.2317	2.0611	1.6082	1.8417	2.0254	2.2011	2.6087	2024-05-31 20:52:44
3	rozumden	2.3195	2.135	1.5368	1.8946	2.1056	2.3642	2.7178	2024-05-29 09:34:11
4	kcml	2.4531	2.2423	1.58	1.987	2.2484	2.4979	2.8687	2024-06-01 06:48:03

Looks like meat's back on the menu, boys!

I should try burnout clustering

A point is set on fire. It spreads if another point is in range. Once the flame stops, burned points are stored as a cluster and a random point is ignited to generate a new cluster.
Since I only need the largest cluster, I could prune the algorithm, when the size of the cluster is greater than the number of points that are left.
With KDtrees it is a breeze.

Changed the clustering algorithm to DBSCAN, since it only requires O(n) memory
New best score 2.001624486134143
I need to experiment with epsilon parameter (distance)

03.06.24

Now the question is how do i get sklearn onto evaluation server

clustering_eps=150
2.0004826993042033
evaluation is already running for an hour and I don’t fancy it (it usually takes under 30 minutes).
On validation test it only decreased loss value by 0.0008
If I could I would stop the evaluation
clustering_eps=80
1.996261918701589
Faster,better, stronger

The problem with all point clouds - the roofs are missing. Thus it is impossible to correctly predict where a ridge is
You know where the roofs are slightly less missing? In the depth maps.

I need an algorithm that would fit the roof from the depthmap onto the pointcloud like a hat
What I have:

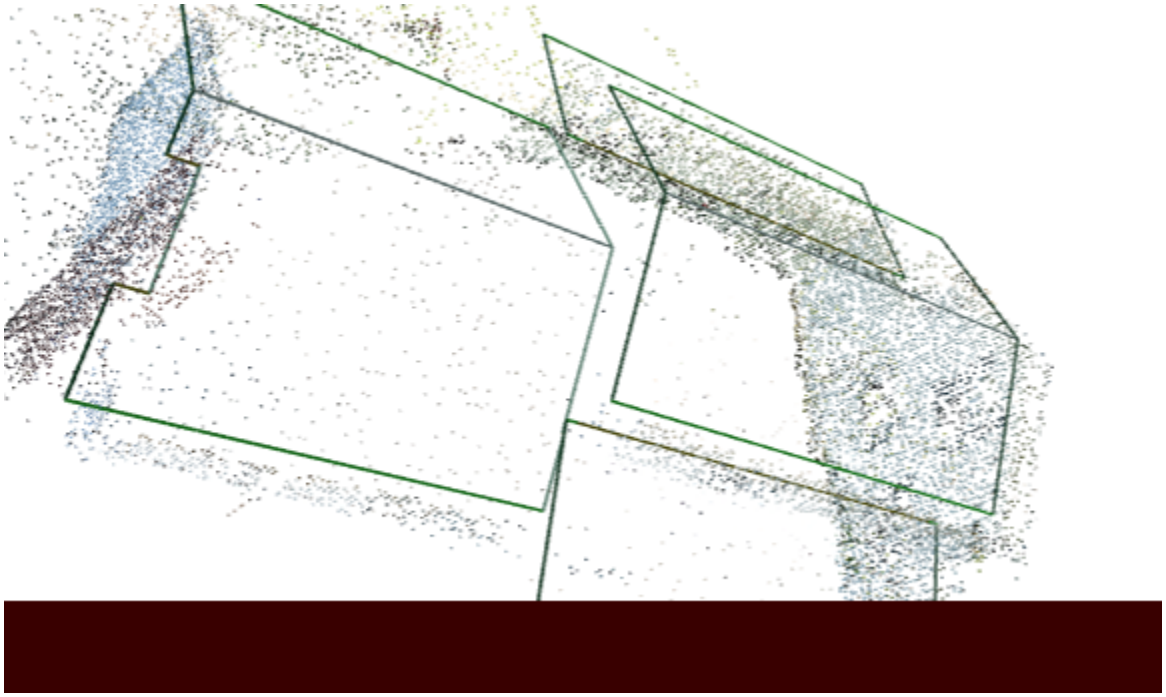
- depth map of multiple views
- Roof segmentation
- Pointclouds without a roof

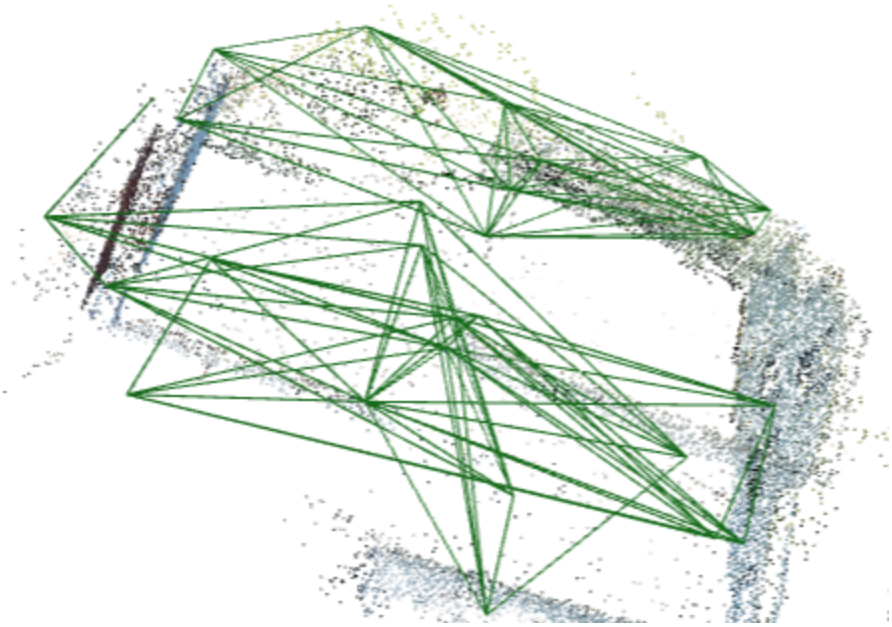
The problem: I don't want to do it myself

I also don't like that the samples are loaded all at once

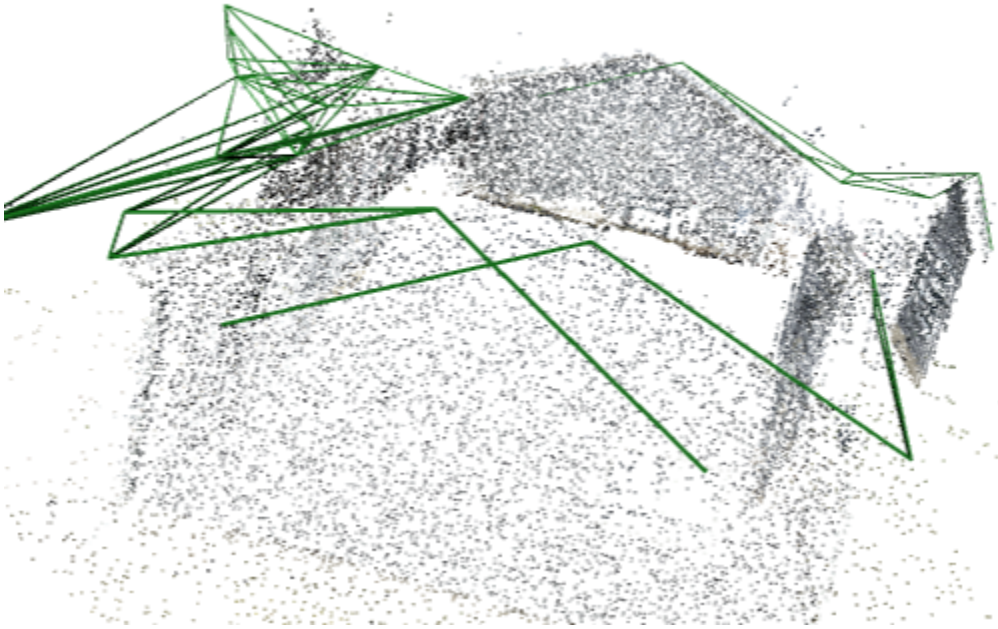
04.06.24

Out of curiosity I want to try to fill the depth map with zeros (as it is still used, where there is no exact match for the pointcloud image)
Loss increase of 0.005





(I relaxed the conditions for vertex connections,)
Either the camera parameters are bad or the distance calculation
I can see some parts of the roof but they are very disjoint

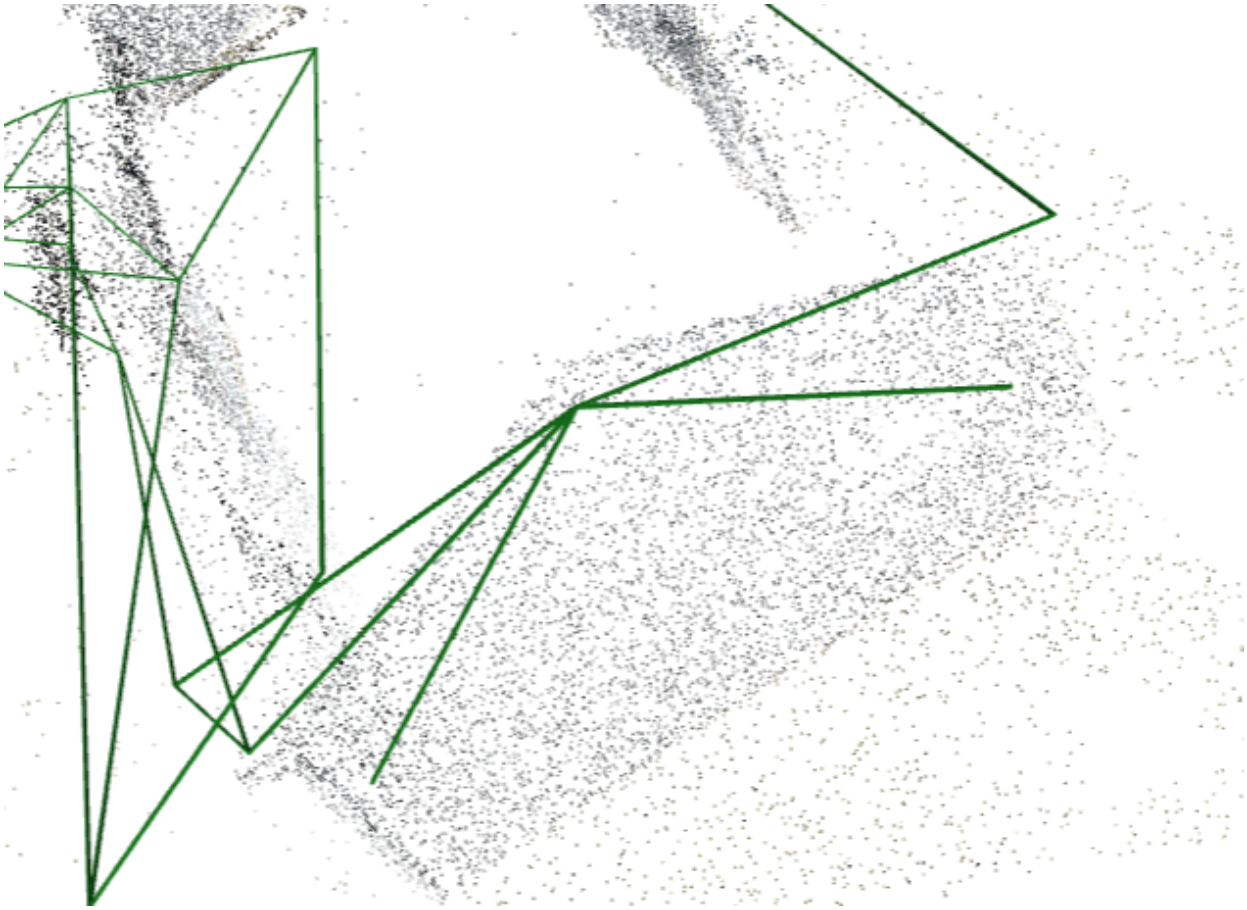


Another roof
Why are the two apexes disjoint?
And the lines seem to far away from the building.
What if I multiply all distances by some coefficients

New best score 1.8942137992717705’
Decrease of 0.02 which is something, I guess I can add this coefficient as a parameter

05.06.24

Still, it didn’t fix misaligned roofs
Well it did in some cases



But it is not exactly on the roof
Also I wonder how adding a ridge would influence the score
What if I add a rule to alway connect two highest points
Before

```
minmax=(0.7715468088942786, 2.91199278094749), mean=1.8942137992717705,
```

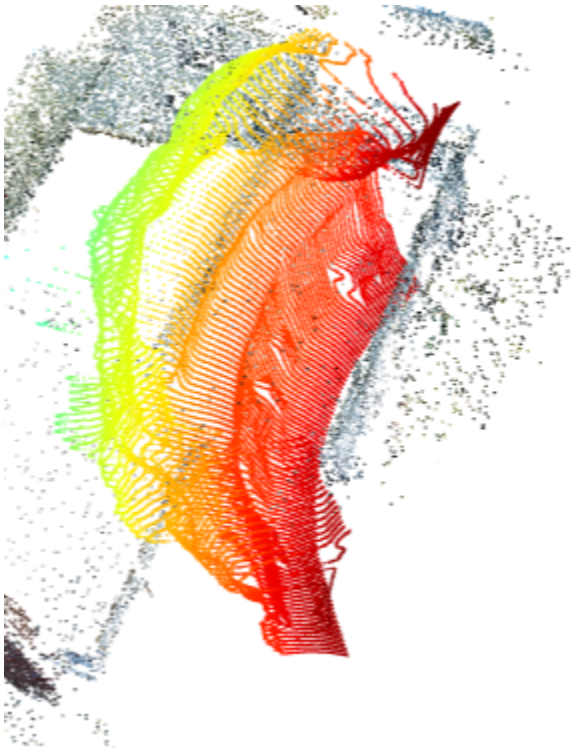
after

```
minmax=(0.7715468088942786, 3.0407162477839016), mean=1.900345392532315, variance=0.17705950227922487,
```

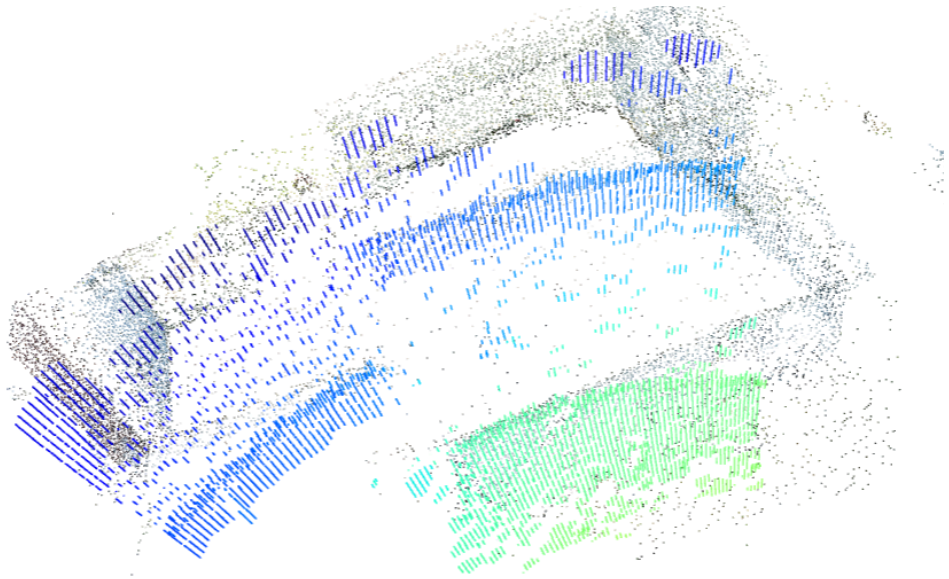
Or I should make a rule that only apexes can be connected that way
What if I try to set vertex points to the closest ones in the pointclouds

06.06.24

What if I train a model that would transform the constructed wireframe into the correct one?
After fitting the points exactly onto the pointcloud the minimal loss decreased by 0.2
But maximum loss increased by the same amount
Lets define a problem. Fit depth map onto projection depthmap. Projection depthmap is sort of a ground truth with some missing data. The catch: the only allowed operations are scaling and addition. And the coefficients are the same for all values. I need to minimise the distance between the two sets of points.
What approaches could I use?



I dont like how distorted it is. Is there a parameter that I an use to straighten it out in the minimization procedure
Maybe i should scale only segmented parts - roofs in particular



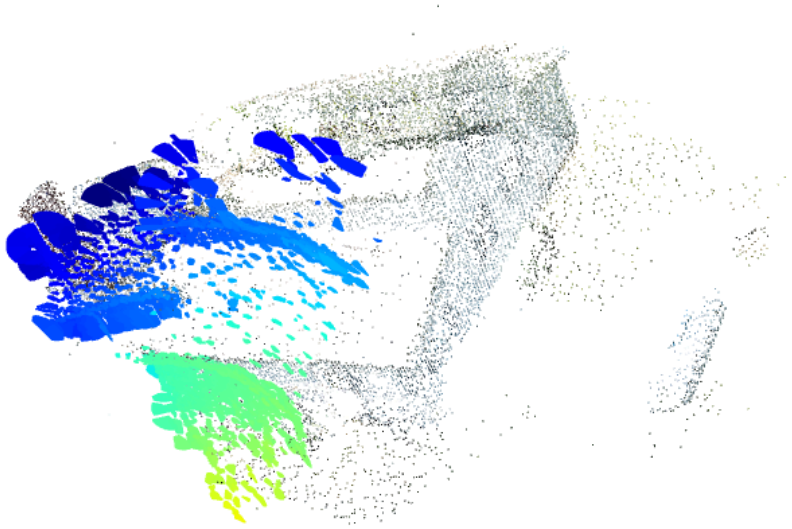
The curvature is present even in interpolation

Why the hell the images with distorted camera parameters are not in fact distorted?
Are only z coordinates distorted

07.06.24

Why the hell the xy_local was normalized??? It gives a sphere by definition

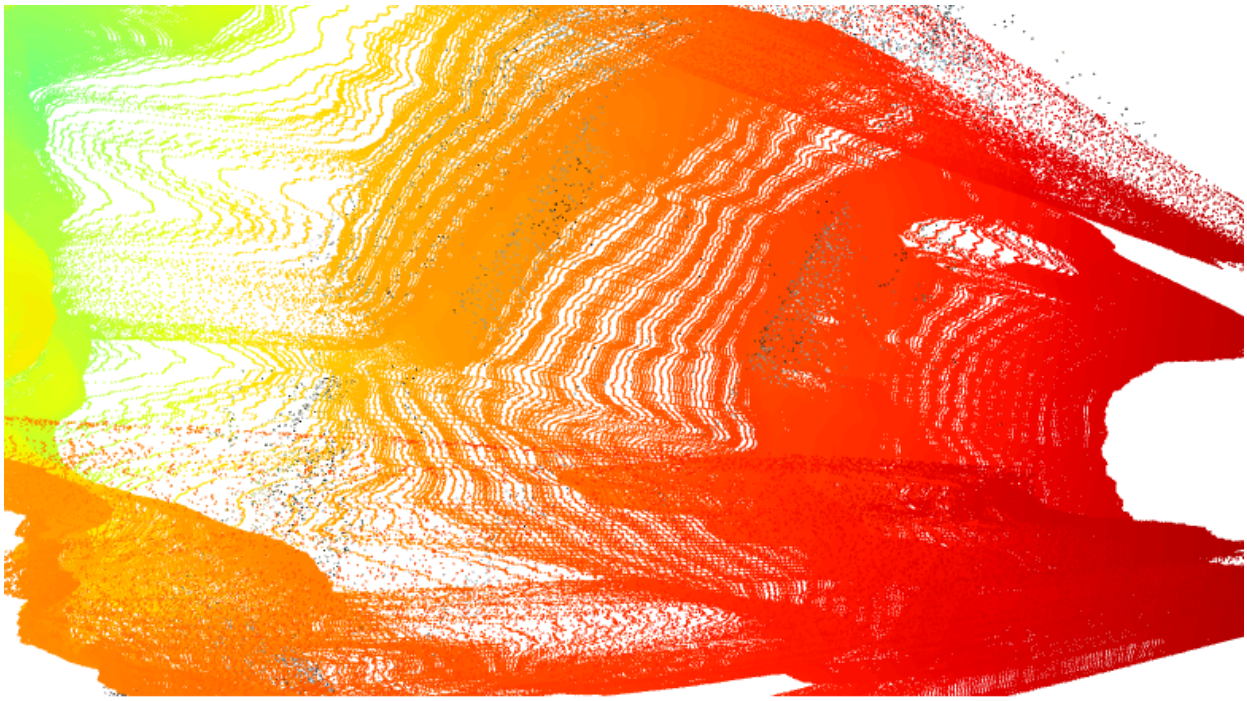
With normalization:



With scaling by the smallest norm

[no image, but it got fixed]

When i divided by the norm of the longest norm suddenly the depthma does not fit as bad

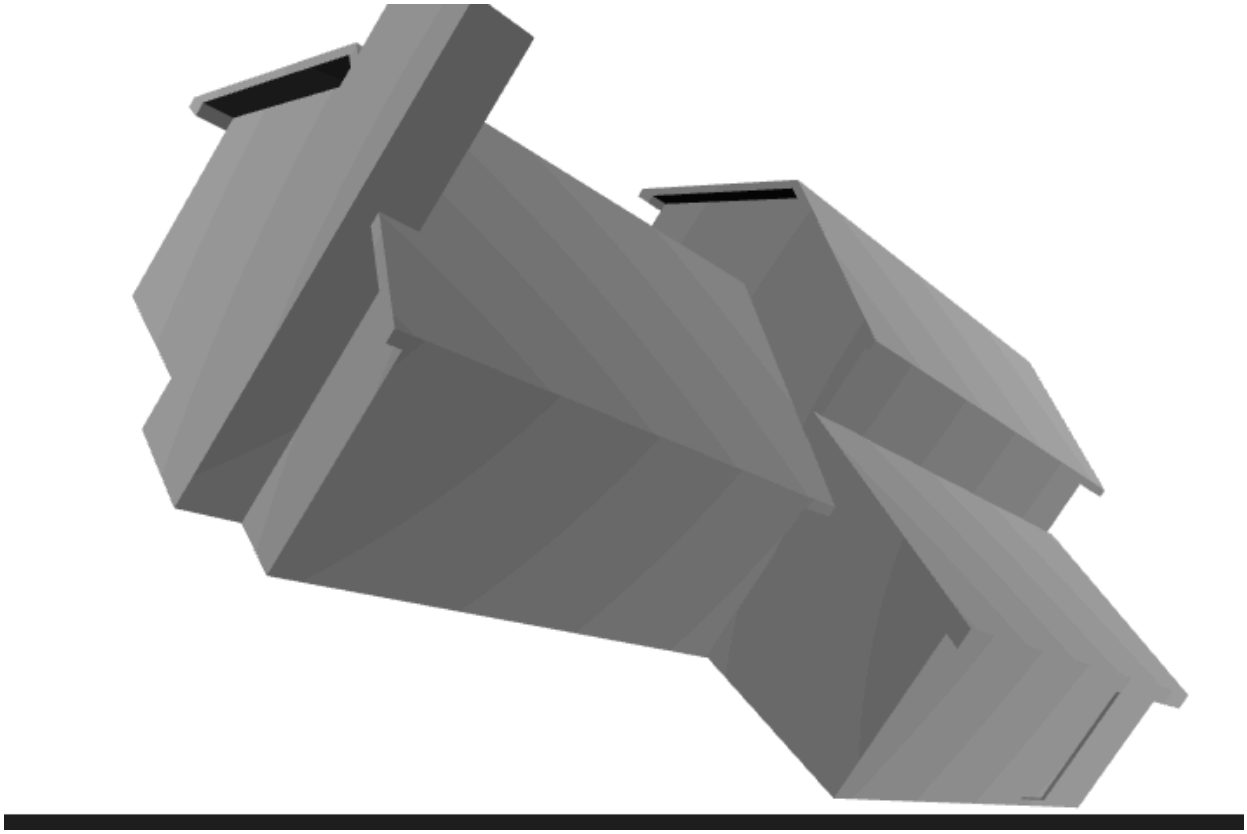


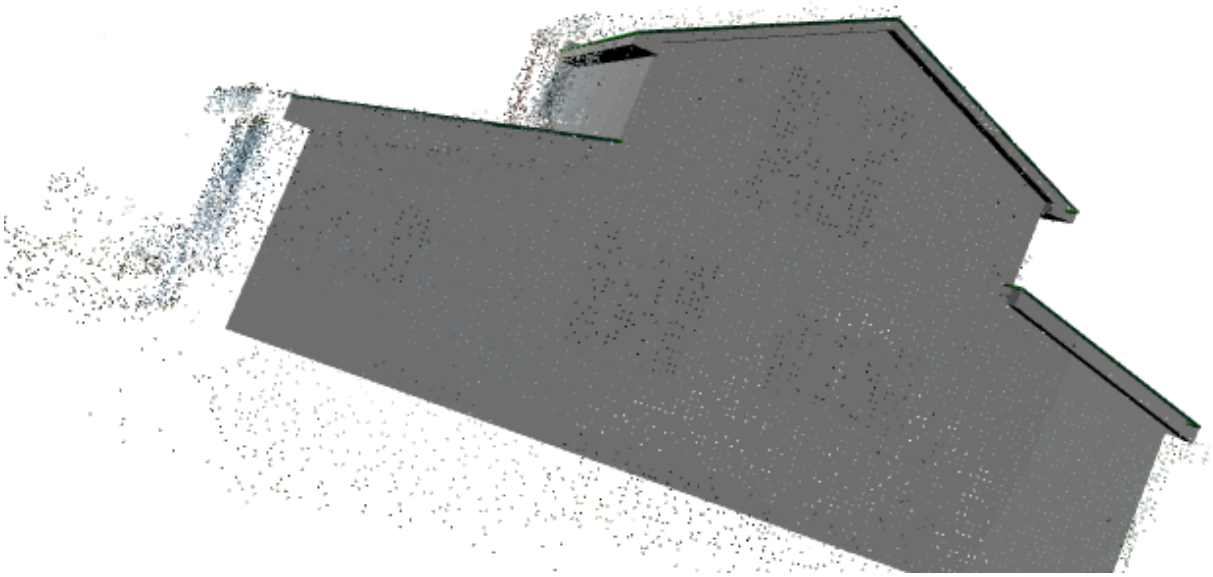
What if i just add a rule that is like this: If there are no points in certain radius to the queried point, the data is taken from the depthmap

I could reuse the interpolator for querying

I am quite frankly surprised that I did not think of it earlier
No improvement after adding it
Although the variance decreased

Wtf is that mesh





Even point cloud is not ground truth

09.06.24

The day of fine tuning it is
Also I will try to add FCNN to try to compute the depth better.
I hope the competition ends tonight

10.06.24

It appears that the competition did not end… It’s so joever… and multiple people have passed me.