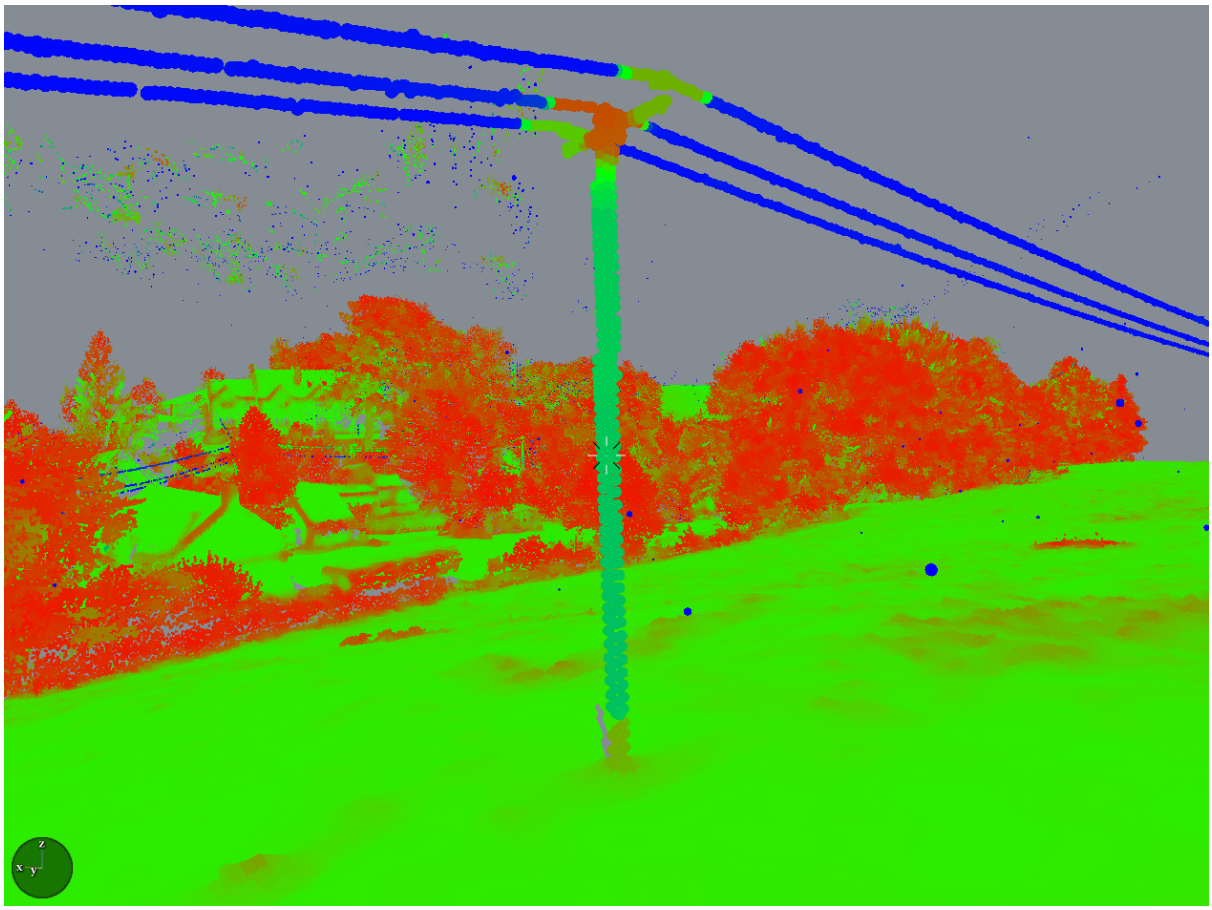
Signals for Point Clouds



Contents

1 Signals

1.1 Making signals

Points in a point cloud possess basic attributes. These may include, and may not be limited to:

- A position (x, y, z) in 3D space. (Three 32-bit integers when unscaled, or three 64-bit floats when scaled.)
- An intensity. (An unsigned 16-bit integer.)
- Return number and number of returns (An unsigned 8-bit integer)
- Classification. (Unsigned 8-bit integers.)
- Time. (A 64-bit float.)

A **signal** is an attribute of the points in the point cloud which may be derived from the basic attributes. These attributes will be positive real numbers, and we will plot them by storing their values as the intensity of points in a new LAS file. If a point cannot have such an attribute assigned to it for any particular reason, we could exclude it from the signal. Often our signals will be given by geometric statistics, but they may also be given by the selection of a family of points in the point cloud which have a special property, i.e. they may be boolean signals. If our signals are boolean, then we will plot the signals as raw classification rather than as intensity. In other words, we use intensity for anything which may be considered "continuous", and classification for those signals which may be considered "discrete".

Remark 1.1.1. Some of our signals are given by regression coefficients. These statistics only attain values between 0 and 1, so it's important to remain aware of the caveat that if we store these raw numbers as intensities (i.e. 16-bit integers), they will be set to either 0 or 1. Therefore, we will multiply these signals by 1000 before plotting them. When we colour our point clouds in Displaz, we will divide by 1000 again to get a number between 0 and 1, which will allow us to represent a colour, in order to make the value of the signal visibly high or low.

Suppose we have a collection of 3-vectors

$$\{\mathbf{r}_1 = (x_1, y_1, z_1), \mathbf{r}_2 = (x_2, y_2, z_2), \ldots, \mathbf{r}_n = (x_n, y_n, z_n)\}$$

which constists of positions in the point cloud.

Signal. *We can compute the* ***covariance*** *of the vector* $(x_1, x_2, \ldots, x_n)$ *and* $(y_1, y_2, \ldots, y_n)$*, given by*

$$c_{\text{xy}} = \frac{1}{n}\sum_i (x_i - \overline{x})(y_i - \overline{y}),$$

where $\overline{x}$ *and* $\overline{y}$ *are the* ***means****, given by*

$$\overline{x} = \frac{1}{n}\sum_i x_i$$

$$\overline{y} = \frac{1}{n}\sum_i y_i.$$

We could also take yz-covariance and zx-covariance. So, we get the covariances

$$c_{\mathrm{xy}},\, c_{\mathrm{yz}},\, c_{\mathrm{zy}}$$

and standard deviations,

$$\sigma_{\mathrm{x}} = c_{\mathrm{xx}}^{1/2},\, \sigma_{\mathrm{y}} = c_{\mathrm{yy}}^{1/2},\, \sigma_{\mathrm{z}} = c_{\mathrm{zz}}^{1/2}.$$

Many of the signals herein are built out of statistics such as these. But we can't rightly call a single statistic a "signal". We'll instead take statistics at a local scale. Many of our signals will have parameters:

- a **radius** $\epsilon > 0$,
- a **k-number** $k > 0$, $k \in \mathbb{Z}$.

There are more parameters to follow, but their purpose takes more explaining.

Given any statistic which makes sense for a family of 3-vectors, we can build a function f by using the following recipe to evaluate $f(\mathbf{r})$ at any point $\mathbf{r} = (x, y, z)$:

- Find the k points in the point cloud that are nearest to the position $\mathbf{r}$, with positions $\mathbf{r}_0, \mathbf{r}_1, \ldots, \mathbf{r}_{k-1}$. (Including $\mathbf{r}_0 = \mathbf{r}$ itself, if a point has position $\mathbf{r}$.)
- Discard the points $\mathbf{r}_i$ such that $\|\mathbf{r} - \mathbf{r}_i\| \geqslant \epsilon$.
- Take the statistic for the remaining points, and let this be the value $f(\mathbf{r})$.

This function becomes a signal, because we can take it as a new attribute for any point in our point cloud with at least m neighbours within a sphere of radius ϵ. If a point has position $\mathbf{p}$ and has enough near neighbours, it now obtains the new attribute $f(\mathbf{p})$.

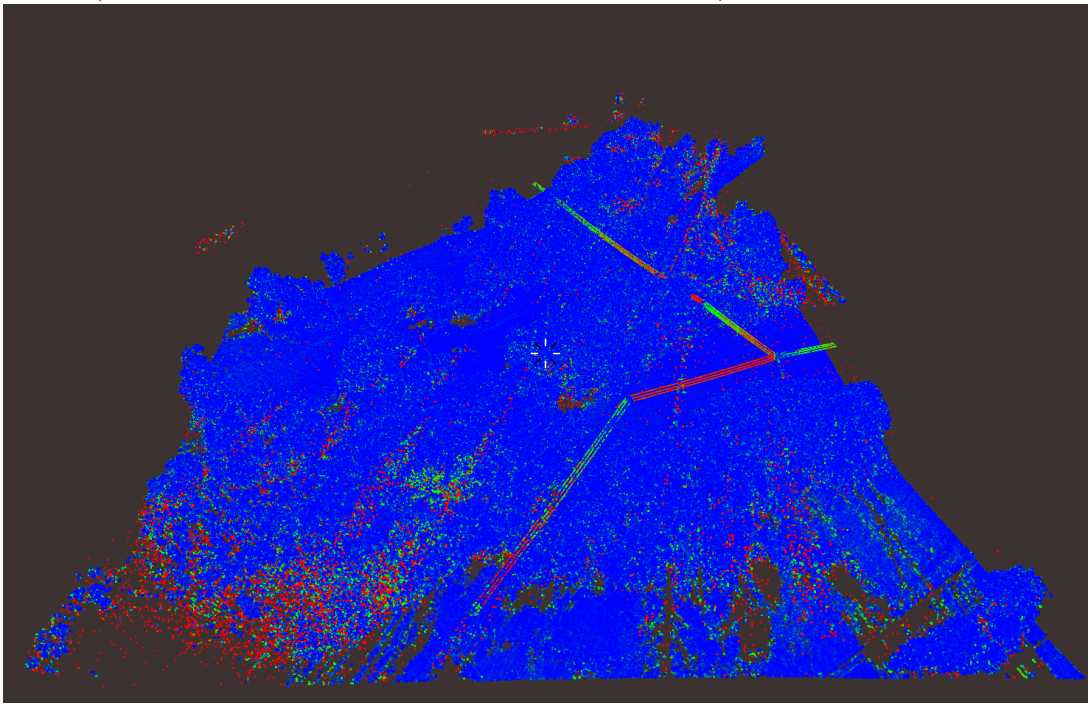
Therefore, for any point cloud, we obtain covariance attributes $c_{\mathrm{xy}}(\mathbf{p})$, $c_{\mathrm{yz}}(\mathbf{p})$, $c_{\mathrm{zy}}(\mathbf{p})$ and standard deviation functions $\sigma_{\mathrm{x}}(\mathbf{p})$, $\sigma_{\mathrm{y}}(\mathbf{p})$, and $\sigma_{\mathrm{z}}(\mathbf{p})$. Consequently, we obtain the corresponding signals.

Remark 1.1.2. The reasons we insist on these three parameters, ϵ, k, are the following.

- Keeping our points within a sphere of radius ϵ stops our k nearest neighbours from being so far apart that we cannot rely on the result. If two object types are sat close together in the landscape, e.g. two cars, then the nearest neighbours of points on one object may be actually from the other object. We want to aviod the mixing of objects.
- Increasing k allows us to increase the quality of our signals which measure the relationship between near neighbours. We may also want to increase k to combat local oversampling. Putting an upper bound on the number of neighbours keeps calculations down.

Addendum 1.1.3. One of the problems in lidar data is oversampling caused by the aircraft slowing down or rotating, thus causing some points to have far more neighbours than other points. This is indicated by the space-time diagram (with elevation set to time) in Figures 3, 4. We'll fix this problem later, but it will require us to introduce a new parameter to some signals.

Figure 1: (Just an example of what our signals will look like) Linear regression, $k = 50$, $\epsilon = 0.75$



Points which form near-straight lines have higher values

Figure 2: (Just an example of what our signals will look like) Linear regression, $k = 50$, $\epsilon = 0.75$

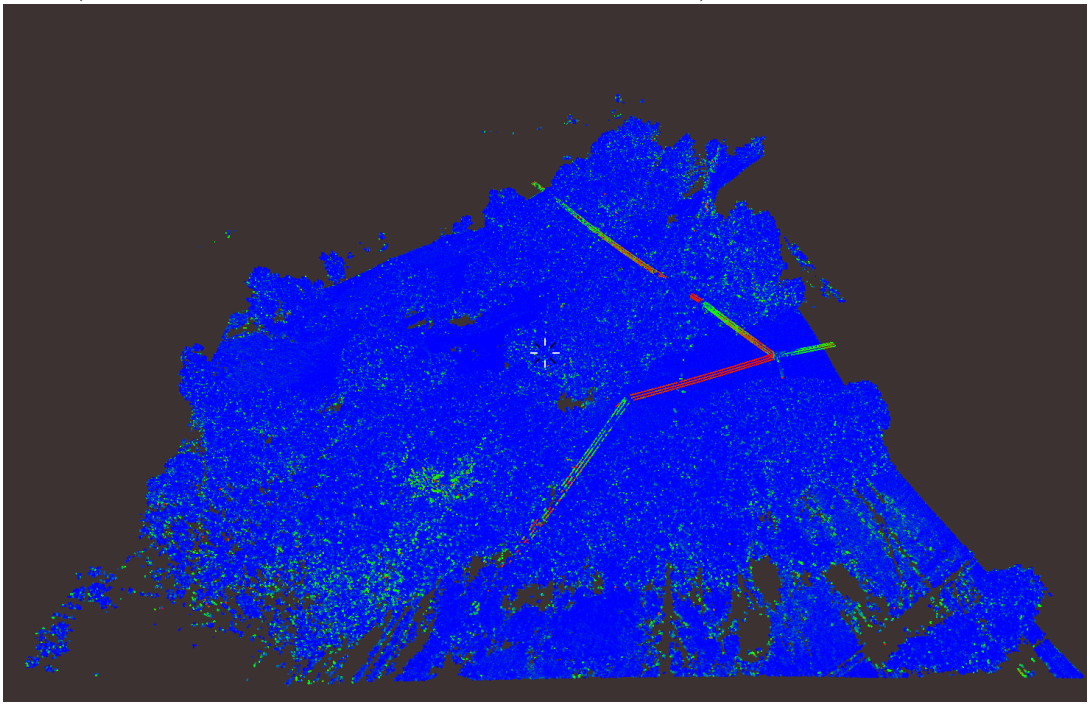
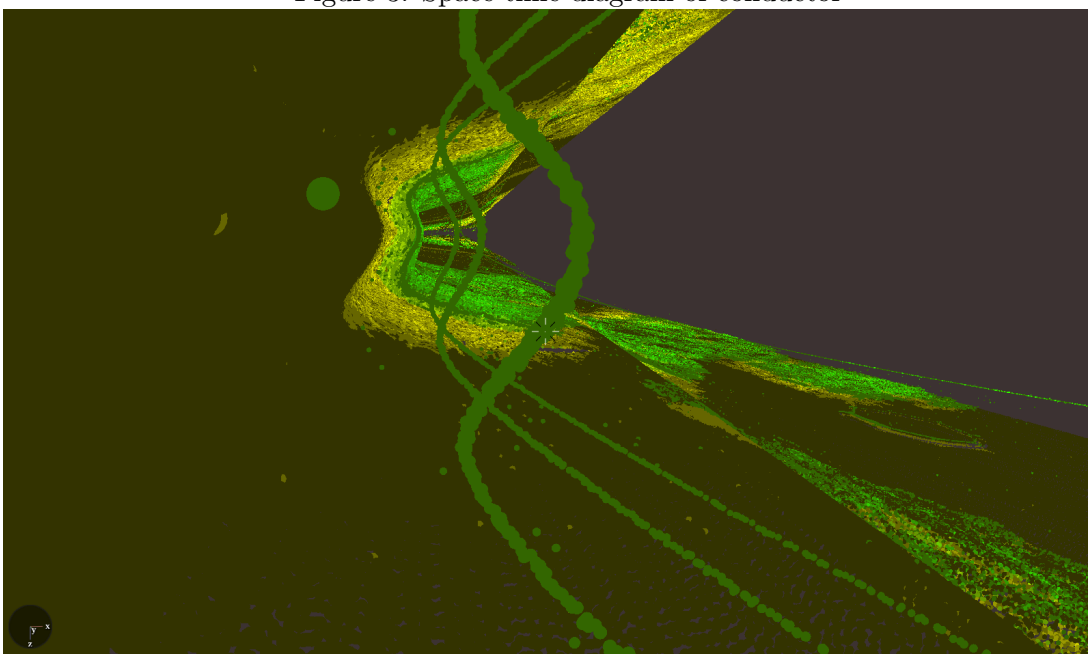
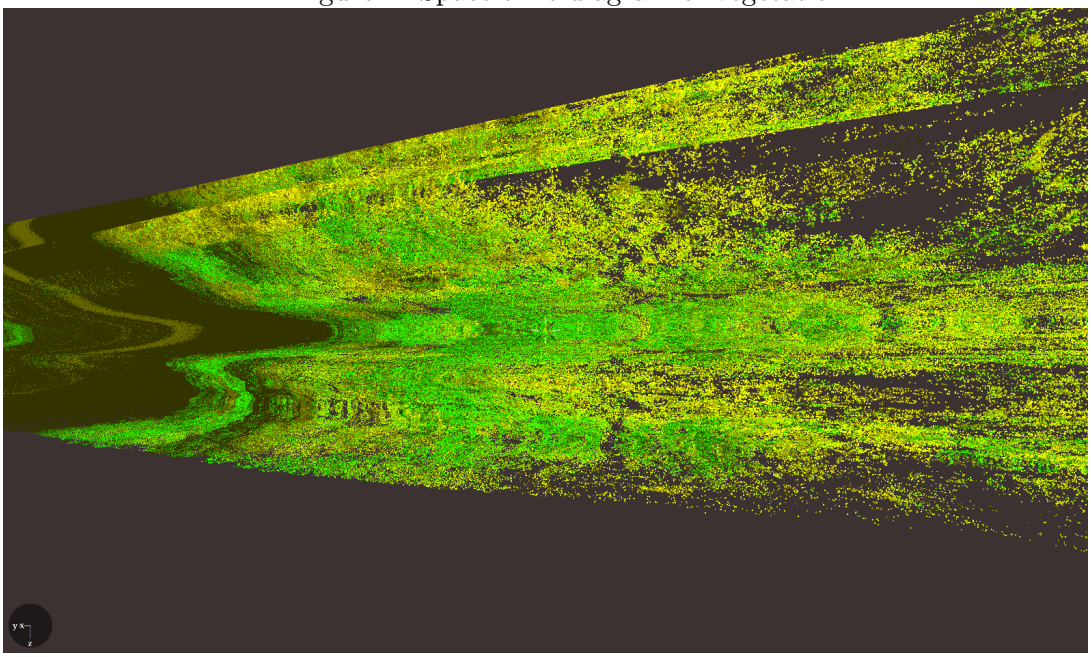


Figure 3: Space-time diagram of conductor



Space-time bending back on itself corresponds to turning point in attitude of aircraft

Figure 4: Space-time diagram of vegetation



Oversampling is caused at the “folds”

1.2 Covariance signals

Suppose we have a family of points $\{(x_i, y_i, z_i) : i = 0, 1, , 2 \ldots, n-1\}$.

Signal. *The* ***covariance matrix*** *is given by*

$$C = \begin{pmatrix} \sigma_{\mathrm{x}}^2 & c_{\mathrm{xy}} & c_{\mathrm{xz}} \\ c_{\mathrm{yx}} & \sigma_{\mathrm{y}}^2 & c_{\mathrm{yz}} \\ c_{\mathrm{zx}} & c_{\mathrm{zy}} & \sigma_{\mathrm{z}}^2 \end{pmatrix}.$$

Application domain. *The covariance matrix is an operator which encodes the shape of a family of points. We will see how the nature of the shape can be deduced.*

Since the covariance matrix is symmetric, it is orthogonally diagonalisable – so says the Spectral Theorem [1, Theorem 5.20].

For any vector

$$\mathbf{v} = \begin{pmatrix} a \\ b \\ c \end{pmatrix}$$

we have

$$\mathbf{v}^T C \mathbf{v} = \overline{w}$$

where $\overline{w} = \frac{1}{n} \sum_{i=1}^{n} w_i$ is the mean of the derived attribute

$$w_i = (a(x_i - \overline{x}) + b(y_i - \overline{y}) + c(z_i - \overline{z}))^2.$$

of our points. Now, the points (x_i, y_i, z_i) are coplanar if and only if $w_1 = w_2 = \cdots = w_n = 0$, or equivalently $\overline{w} = 0$, for some vector $\mathbf{v} \neq \mathbf{0}$.

Since C is orthogonally diagonialisable, there is an orthonormal basis $\{\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2\}$ of $\mathbb{R}^3$ which consists of eigenvectors of C. Let $\{\lambda_0, \lambda_1, \lambda_2\}$ be the correpsonding eigenvalues. If $\mathbf{v} = a_0\mathbf{v}_0 + a_1\mathbf{v}_1 + a_2\mathbf{v}_2$ then the corresponding attribute w is such that

$$\overline{w} = \mathbf{v}^T C \mathbf{v} = a_0^2\lambda_0 + a_1^2\lambda_1 + a_2^2\lambda_2.$$

Since C generates a non-negative bilinear form, $\lambda_0, \lambda_1, \lambda_2 \geqslant 0$. The points (x_i, y_i, z_i) are coplanar if and only if the matrix C is *not* positive definite, i.e. if $\det C = \lambda_0\lambda_1\lambda_2 = 0$.

This matrix is the parent to many signals which we can plot:

- Planar regression.
- XY-linear regression.
- 3D linear regression.
- Ruggedness.
- Eigenvalues and rank.

The eigenvalues determine the behaviour of the points according to the following table.

Eigenvalues close to 0	Behaviour
None	Points spread out
λ_0, not λ_1	Coplanar, not colinear
λ_0, λ_1, not λ_2	Colinear, not concentrated around a point
λ_0, λ_1, λ_2	Concentrated around a point

We will prove that this table is right in Section 1.2.5.

1.2.1 Planar regression

We would like a measure for the coplanarity of our points. We will derive this measure from the coplanarity matrix C which we have already met.

Now, since C is a non-negative matrix,

$$\det C = (\sigma_{\mathrm{x}}^2\sigma_{\mathrm{y}}^2\sigma_{\mathrm{z}}^2 + 2c_{\mathrm{xy}}c_{\mathrm{yz}}c_{\mathrm{zx}}) - (\sigma_{\mathrm{x}}^2c_{\mathrm{yz}}^2 + \sigma_{\mathrm{y}}^2c_{\mathrm{xz}}^2 + \sigma_{\mathrm{z}}^2c_{\mathrm{xy}}^2) \geqslant 0$$

so

$$\frac{\sigma_{\mathrm{x}}^2c_{\mathrm{yz}}^2 + \sigma_{\mathrm{y}}^2c_{\mathrm{xz}}^2 + \sigma_{\mathrm{z}}^2c_{\mathrm{xy}}^2}{\sigma_{\mathrm{x}}^2\sigma_{\mathrm{y}}^2\sigma_{\mathrm{z}}^2 + 2c_{\mathrm{xy}}c_{\mathrm{yz}}c_{\mathrm{zx}}} \leqslant 1$$

with equality if and only if the points (x_i, y_i, z_i) are coplanar. Note that if the denominator is zero, so is the numerator, and this ensures that the points are coplanar anyway.

Signal. *We define the statistic*

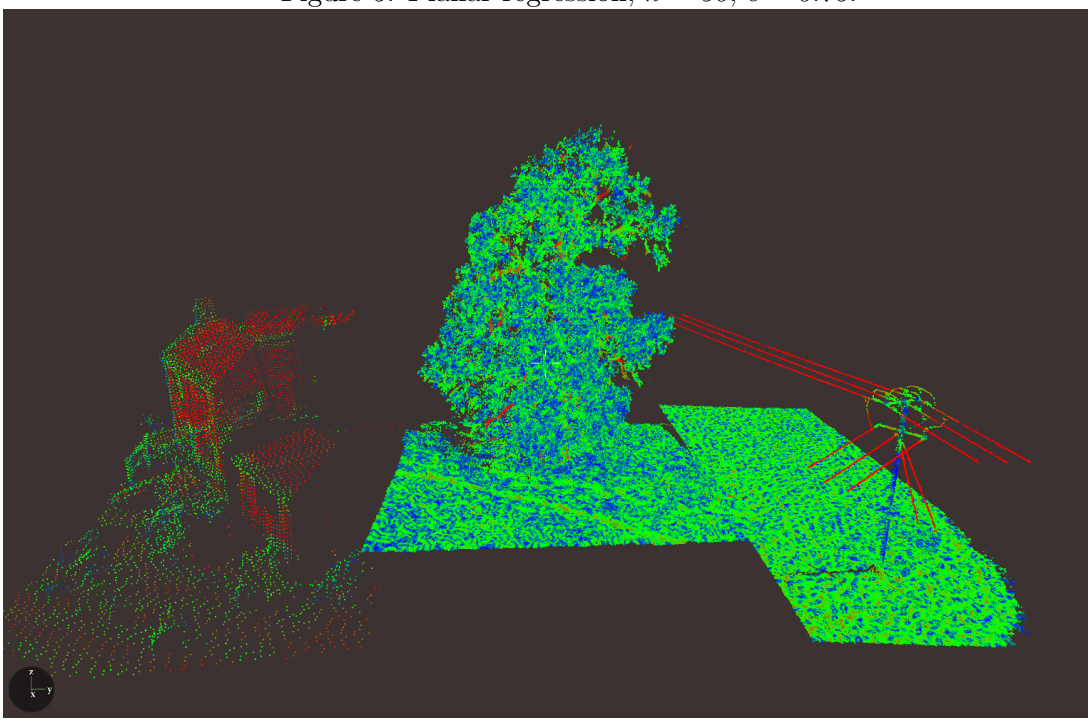
$$p_{\mathrm{xyz}} = \begin{cases} 1 & \sigma_{\mathrm{x}}^2\sigma_{\mathrm{y}}^2\sigma_{\mathrm{z}}^2 + 2c_{\mathrm{xy}}c_{\mathrm{yz}}c_{\mathrm{zx}} = 0 \\ \dfrac{\sigma_{\mathrm{x}}^2c_{\mathrm{yz}}^2 + \sigma_{\mathrm{y}}^2c_{\mathrm{xz}}^2 + \sigma_{\mathrm{z}}^2c_{\mathrm{xy}}^2}{\sigma_{\mathrm{x}}^2\sigma_{\mathrm{y}}^2\sigma_{\mathrm{z}}^2 + 2c_{\mathrm{xy}}c_{\mathrm{yz}}c_{\mathrm{zx}}} & \textit{otherwise} \end{cases}$$

which we call ***planar regression****.*

A plot of planar regression on different object types is given on a test area in Figure 5. As we expect, the building, being made of flat surfaces, has high values. The conductor is locally a straight line (and hence coplanar) so also has high values. In Figure 6, the local planar regression of a building is taken, and most of the points are red, which represents values closer to 1, because the house is made of flat planar surfaces. We can detect the corner of the house, because it shows up as having very small planar regression – observe the blue band down the center of Figure 6. Some high values exist within the tree in Figure 5, and these are where the trunk is locally flat or where thin branches are locally straight.

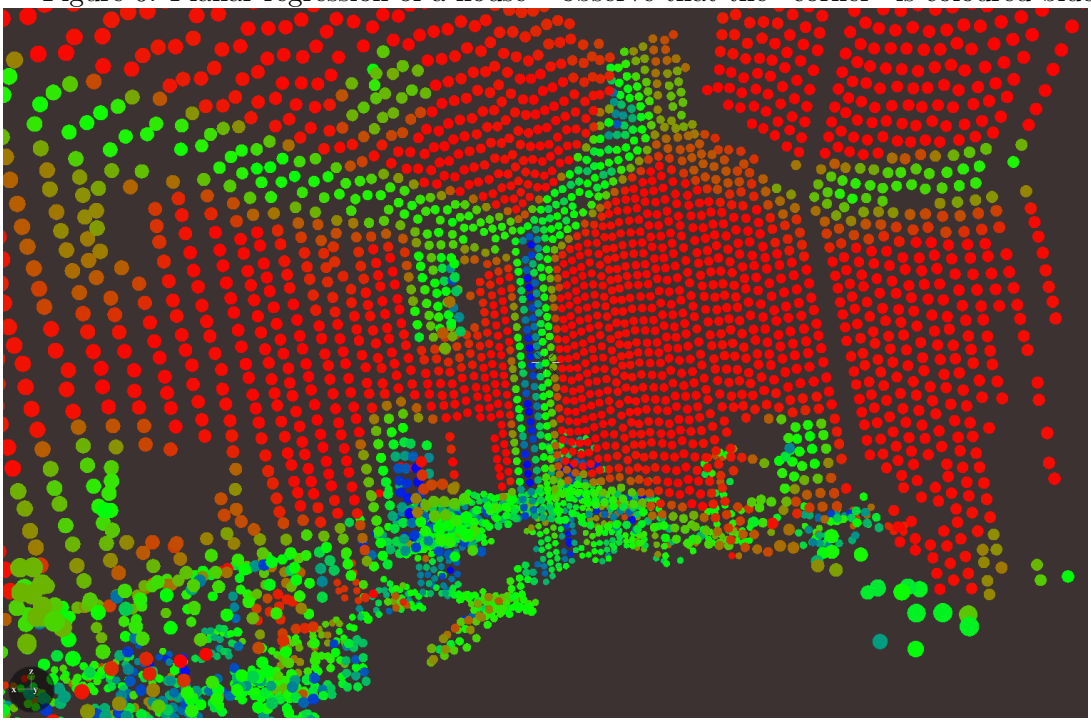
Application domain. *Detecting flat, typlically man-made surfaces such as roofs, walls, patio, etc. Because colinear objects are also coplanar, it detects 1D objects such as conductors and small branches.*

Figure 5: Planar regression, $k = 50$, $\epsilon = 0.75$.



Flat planar surfaces or straight lines have high values

Figure 6: Planar regression of a house – observe that the "corner" is coloured blue



Values drop off at corners of buildings but are large on flat surfaces

1.2.2 XY-linear regression

XY-linear regression or **vertically planar regression** is a signal whose values lie between 0 and 1. If a point has XY-linear regression close to 1, it is regarded as being part of a vertical plane (at the local scale which is dictated by the radius ϵ). Note that any two points automatically lie in a straight line, so some noise will enjoy undeservedly high values of XY-linear regression, and similar comments apply for other types of regression (e.g. any three points are coplanar).

Signal. *At points for which $\sigma_x \sigma_y > 0$, the value of this signal is*

$$r_{xy} = \frac{|c_{xy}|}{\sigma_x \sigma_y}.$$

This expression is derived even more simply than the expression for planar regression, but instead of a the XYZ-covariance matrix, we just start from the XY-covariance matrix

$$\begin{pmatrix} \sigma_x^2 & c_{xy} \\ c_{yx} & \sigma_y^2 \end{pmatrix}$$

At points where $\sigma_x \sigma_y = 0$ we can set $r_{xy} = 1$. Similarly, we could have defined YZ-linear and ZX-linear regression.

Figure 7 shows a plot of XY-linear regression. The conductors give very high values, as we would expect. The walls of the house also give high values, as we would expect, because they lie in planes which are orthogonal to the XY-plane.

Figures 14, 15 and 16 show the result of taking various clips on XY-linear regression.

Application domain. *Detecting cables and vertical walls of buildings, and wall fencing. Very low values on ground, roofs and in some parts of vegetation.*

1.2.3 3D linear regression

Linear regression is a signal whose values lie between 0 and 1. If a point has 3D linear regression close to 1, it is regarded as being part of a straight line (at the local scale which is dictated by the radius ϵ). We should set $m > 2$ for these signals, since any two points automatically lie in a straight line.

Signal. *The **3D linear regression** of a set of points is given by*

$$r_{xy} r_{yz} r_{zx}$$

where r_{xy} is XY-linear regression.

Where this signal is close to 1, we can say that we can draw a straight line through the neighbourhood of our point with most of the points in the neighbourhood lying quite close to the line.

Figure 8 shows a plot of 3D linear regression. Values closer to one (correlated) are coloured with more red, and values closer to zero (uncorrelated) are coloured with more blue. As expected, the conductors are the reddest areas.

Figures 9 and 10 demonstrate the fact that vegetation has some areas of very high local correlation, both linear and planar.

Application domain. *Can detect perfectly straight lines quite well – not good enough for conductors in lidar data.*

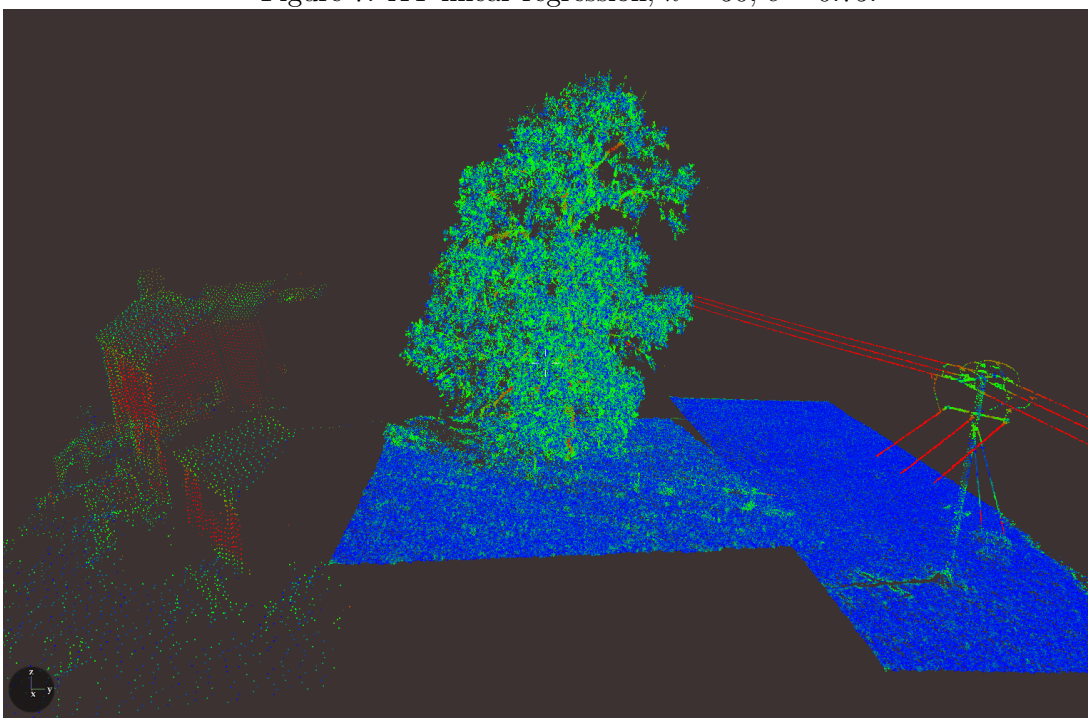
1.2.4 Ruggedness

Signal. ***Ruggedness*** *is just a fancy term for* σ_z*, the standard deviation of elevation.*

It has high values in areas which are vertically spread out. If we take too few nearest neighbours (k), then points which are part of a vertically spread area which suffer from being a little or over sampled, will give disappointingly small values. Figure 11 gives the general idea: Objects with some height give higher values. Compare Figures 12 and 13 – in Figure 12 the pylon was very thin, and in 13 it was very thick.

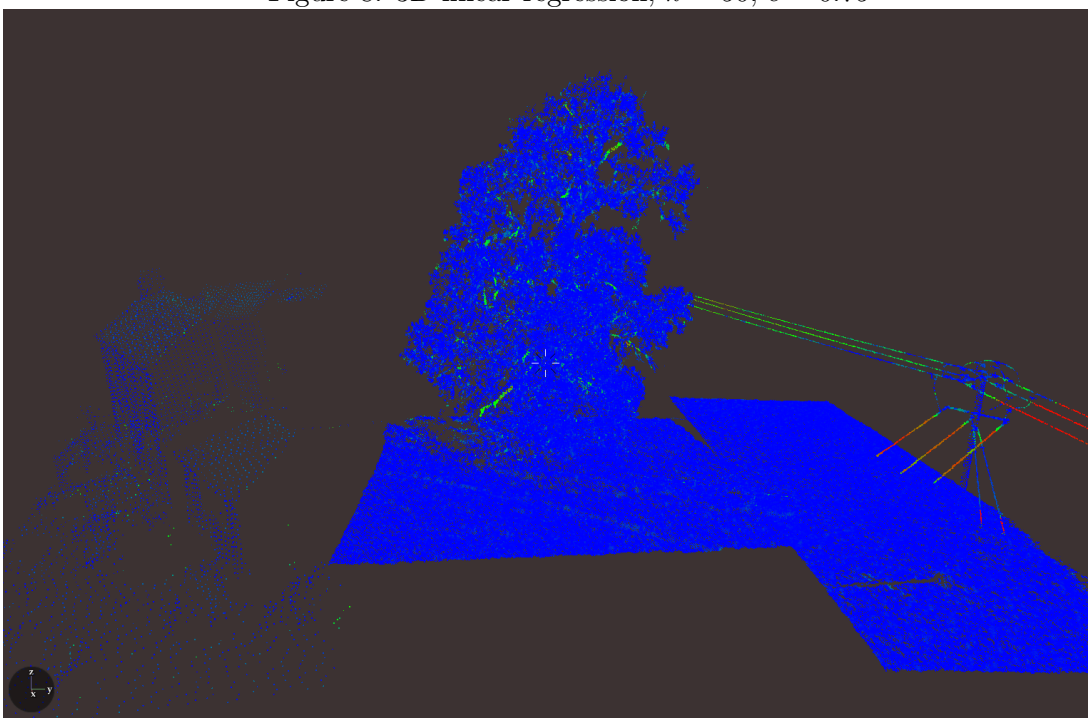
Application domain. *Seperate walls from ground.*

Figure 7: XY-linear regression, $k = 50$, $\epsilon = 0.75$.



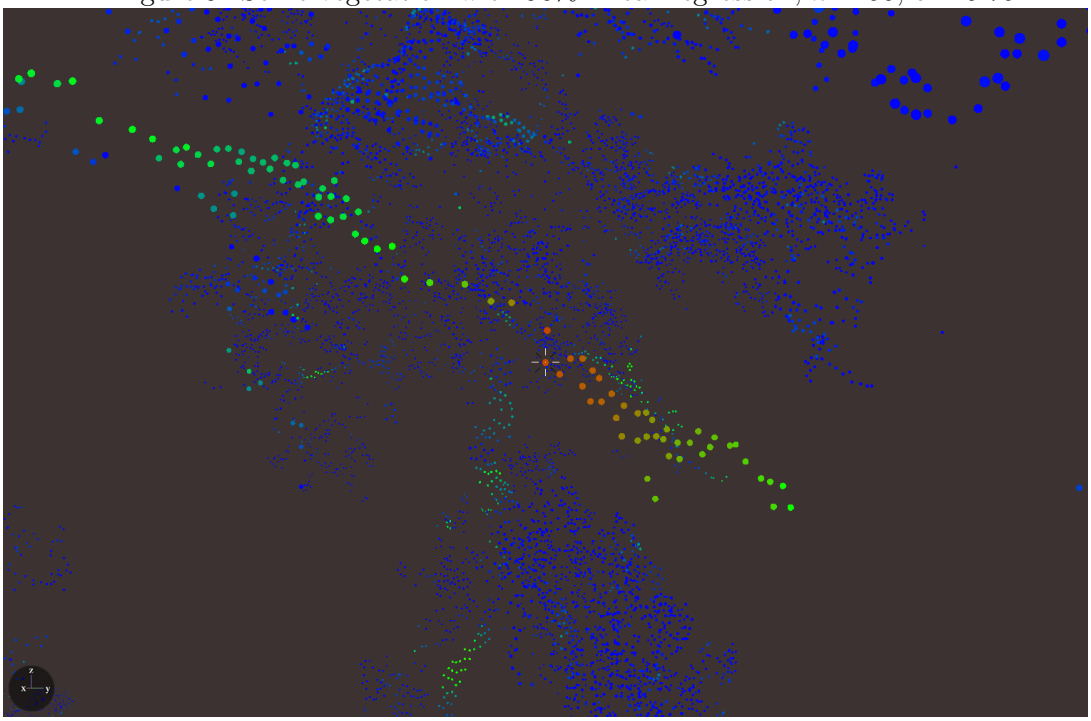
Anything which looks like a straight line from above – building sides and conductors, have high values

Figure 8: 3D linear regression, $k = 50$, $\epsilon = 0.75$



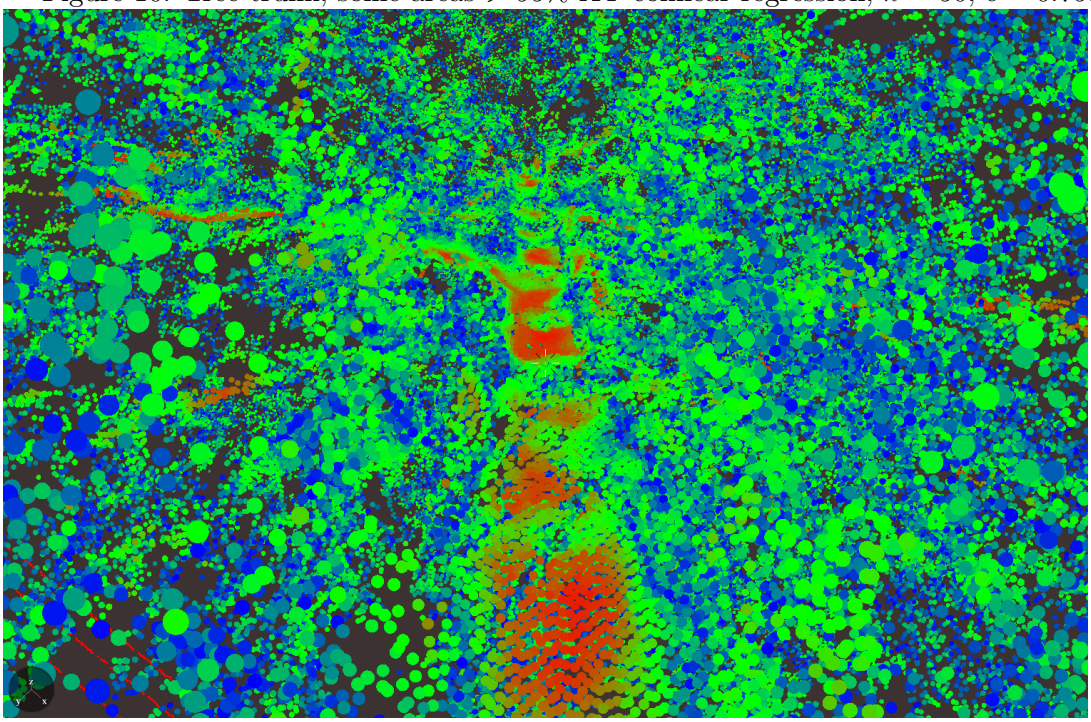
Some high values on conductors but this signal is too sensitive

Figure 9: Some vegetation with 90% linear regression, $k = 50$, $\epsilon = 0.75$.



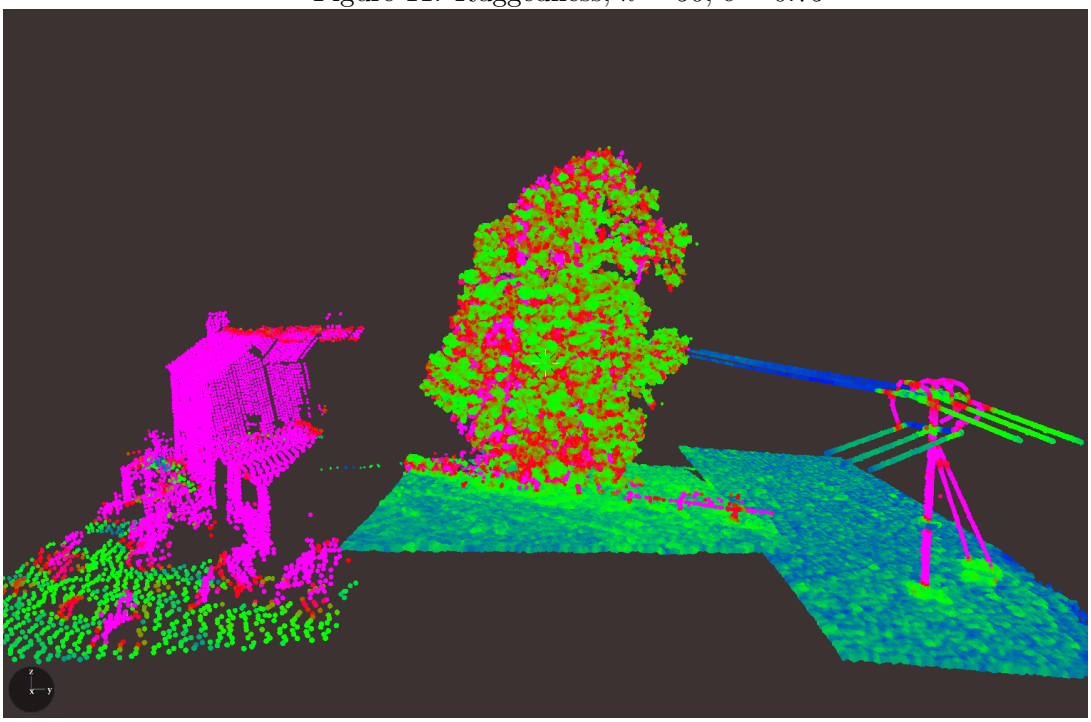
Trees have many attributes; They are "bushy" but also contain some straight lines

Figure 10: Tree trunk, some areas $> 95\%$ XY-colinear regression, $k = 50$, $\epsilon = 0.75$.



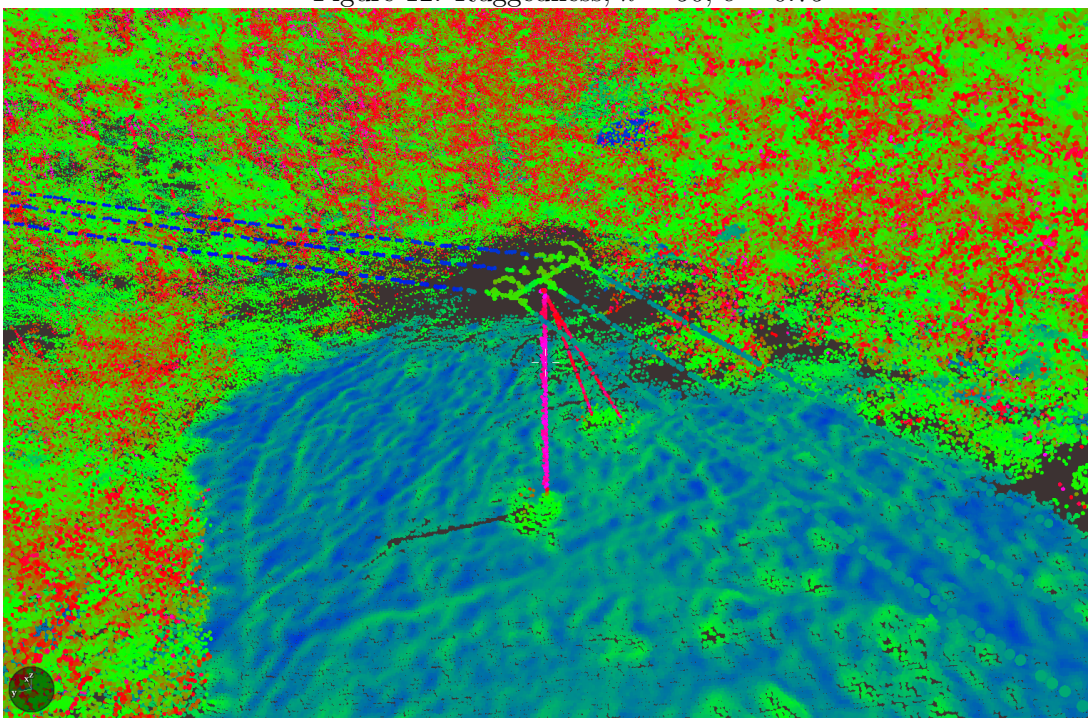
The trunk stands out from the rest of the tree as a locally flat surface

Figure 11: Ruggedness, $k = 50$, $\epsilon = 0.75$



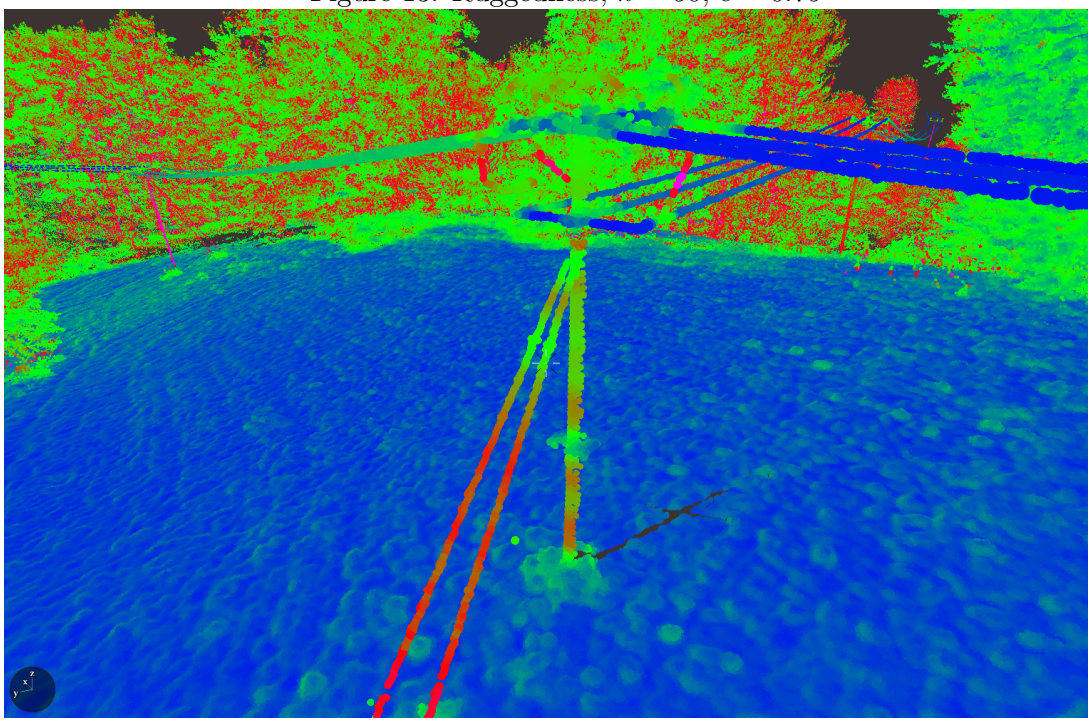
Areas with a vertical spread of points have high values

Figure 12: Ruggedness, $k = 50$, $\epsilon = 0.75$



Houses are distinguished from ground and some pylons from conductors

Figure 13: Ruggedness, $k = 50$, $\epsilon = 0.75$



Thicker or oversampled pylons don't give as good a result

Figure 14: Test area

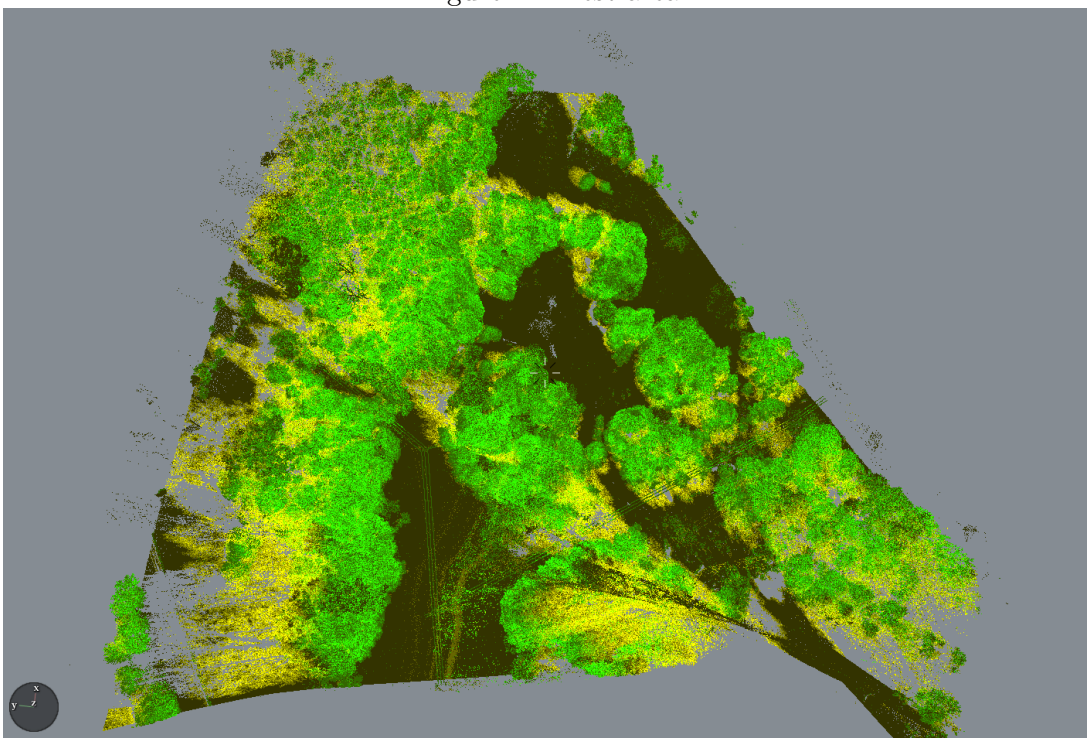
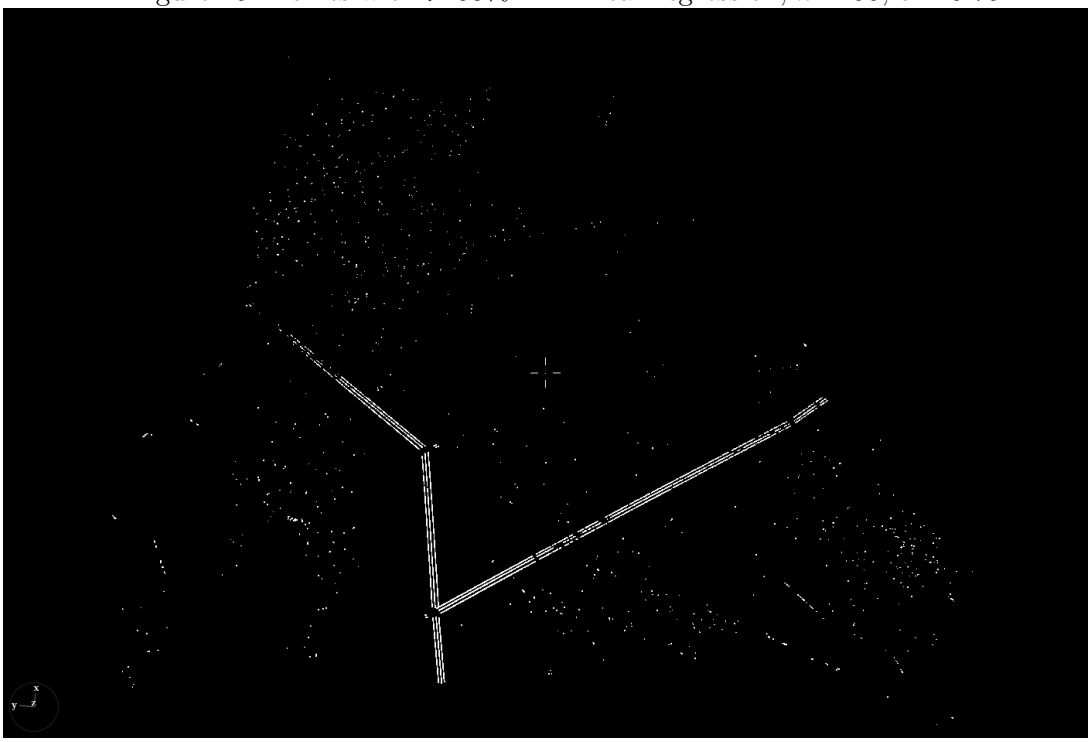
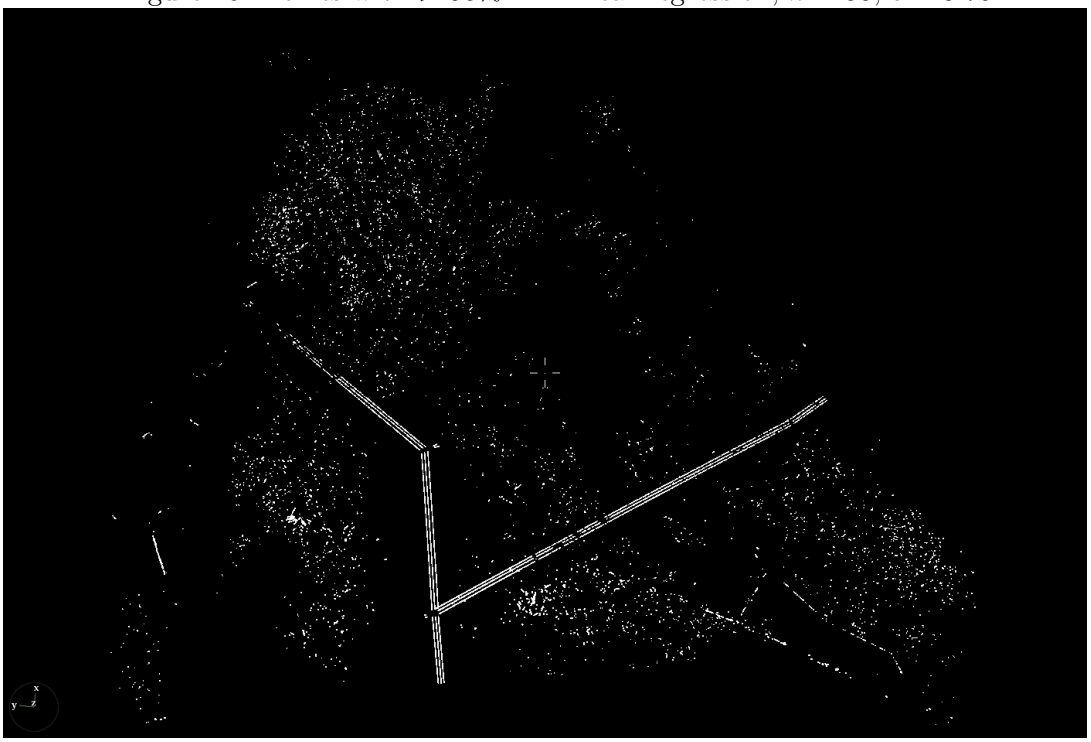


Figure 15: Points with $> 99\%$ XY-linear regression, $k = 50$, $\epsilon = 0.75$.



The conductor is extracted along with a few tree branches, and fences

Figure 16: Points with $> 95\%$ XY-linear regression, $k = 50$, $\epsilon = 0.75$.



N

1.2.5 Eigenvalues and rank

The covariance matrix

$$C = \begin{pmatrix} \sigma_{\mathrm{x}}^2 & c_{\mathrm{xy}} & c_{\mathrm{xz}} \\ c_{\mathrm{yx}} & \sigma_{\mathrm{y}}^2 & c_{\mathrm{yz}} \\ c_{\mathrm{zx}} & c_{\mathrm{zy}} & \sigma_{\mathrm{z}}^2 \end{pmatrix}.$$

This matrix has three eigenvalues

$$0 \leqslant \lambda_0 \leqslant \lambda_1 \leqslant \lambda_2.$$

Since these depend on the points chosen, given any point cloud, and any chosen radius $\epsilon > 0$, we have three functions

$$0 \leqslant \lambda_0(\mathbf{r}) \leqslant \lambda_1(\mathbf{r}) \leqslant \lambda_2(\mathbf{r})$$

which can be evaluated at any point $\mathbf{r} = (x, y, z)$.

Signal. *We have three new signals, given by the eigenvalues λ_0, λ_1 and λ_2 of the covariance matrix*

Application domain. *As we shall see, due to singular value decomposition, the eigenvalues contain enough information to describe the shape of a family of points. Some important signals will be derived from the comparison of the three eigenvalues.*

Remark 1.2.1. Note that, because the matrix C is real and symmetric, it is also self-adjoint, and therefore we can guarantee that its eigenvalues are real. They are non-negative because, as remarked before, $\mathbf{v}^T C \mathbf{v} \geqslant 0$ for any 3-vector $\mathbf{v}$.

Let $\{\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2\}$ be an orthonormal basis of $\mathbb{R}^3$ such that $C\mathbf{v}_i = \lambda_i \mathbf{v}_i$ for $i = 0, 1, 2$. Recall that for any arbitrary vector $\mathbf{v} = a_0\mathbf{v}_0 + a_1\mathbf{v}_1 + a_2\mathbf{v}_2$,

$$\mathbf{v}^T C \mathbf{v} = a_0^2\lambda_0 + a_1^2\lambda_1 + a_2^2\lambda_2,$$

and if this is zero for some $\mathbf{v} \neq \mathbf{0}$, then they points $\mathbf{r}_i$ are coplanar with $\mathbf{v}$ normal to the plane.

Since C is symmetric, it is orthogonally diagonalisable, i.e. the eigenvectors $\mathbf{v}_i$ are mutually orthogonal and span $\mathbb{R}^3$ – this is the spectral theorem. There are several possibilities:

- $0 \neq \lambda_0$ and the points are not coplanar.
- $0 = \lambda_0 \neq \lambda_1$. The points are coplanar but not colinear, and the plane is given by

  $$(\overline{x}, \overline{y}, \overline{z}) + \mathbb{R}\mathbf{v}_1 + \mathbb{R}\mathbf{v}_2.$$

  The eigenvector $\mathbf{v}_0$ is a normal to this plane.
- $0 = \lambda_0 = \lambda_1 \neq \lambda_2$. The points are colinear and the line is given by

  $$(\overline{x}, \overline{y}, \overline{x}) + \mathbb{R}\mathbf{v}_2.$$

  The line has direction $\mathbf{v}_2$.
- $0 = \lambda_0 = \lambda_1 = \lambda_2$. The points are all in the same position $(\overline{x}, \overline{y}, \overline{z})$.

The value of these eigenvalues comes in comparing them according to the table given at the beginning of this section (the argument above proves the table). In Figures 17, 18, 19 and 20, the eigenvalues and determinant of the covariance matrix are plotted at each point in our test tile. In Figure 18, the conductor shows low values but shows high values in 19. This agrees with the theoretical considerations above.

Figure 20 shows a plot the determinant. This is a signal which takes high values only when the point is part of a voluminuous object. Indeed, it is equal to $\lambda_0\lambda_1\lambda_2$. We only find values of any significance inside vegetation.

Comparison of Figures 21 and 22 demonstrates that, as expected, on the flat surfaces of a building, eigenvalue 0 is small but eigenvalue 1 is large. The same is true of the ground, but eigenvalue 1 is not as large for the ground. Figure 23 shows a conductor with high values for eigenvalue 2 and Figure 22 shows it with low values for eigenvalue 1.

To state these methods more rigorously, note that the matrix

$$A = \frac{1}{\sqrt{n}} \begin{pmatrix} x_0 - \overline{x} & x_1 - \overline{x} & x_2 - \overline{x} \dots \\ y_0 - \overline{y} & y_1 - \overline{y} & y_2 - \overline{y} \dots \\ z_0 - \overline{z} & z_1 - \overline{z} & z_2 - \overline{z} \dots \end{pmatrix}$$

where (x_i, y_i, z_i) are positions of points in the points cloud, has singular value matrix

$$\Sigma = \begin{pmatrix} 0 & \dots & 0 & \lambda_2^{1/2} & 0 & 0 \\ 0 & \dots & 0 & 0 & \lambda_1^{1/2} & 0 \\ 0 & \dots & 0 & 0 & 0 & \lambda_0^{1/2} \end{pmatrix}$$

and hence the rank of A, the dimension of the region of space spanned by our points, is indeed the number of non-zero eigenvalues. However, because it will be rare for a bunch of points to be *precisely* coplanar or colinear, it would be far too harsh to compute this rank precisely. Therefore, to compute it, we should decide on some threshold θ and count how many eigenvalues are $\leqslant \theta$.

Signal. *The ranks of the matrices A give us a new signal which depends on the threshold parameter θ. We call this signal* ***rank****.*

Figures 25, 26, 27 and 28 give plots of rank at various thresholds. The key to the colour map is given below.

Colour	Rank
Red	3
Blue	2
Green	1
Black	0

As is evident from the plots, too large a threshold θ means that many of our points are given a rank which is too small. A threshold $\theta = 0.001$ seems quite good.

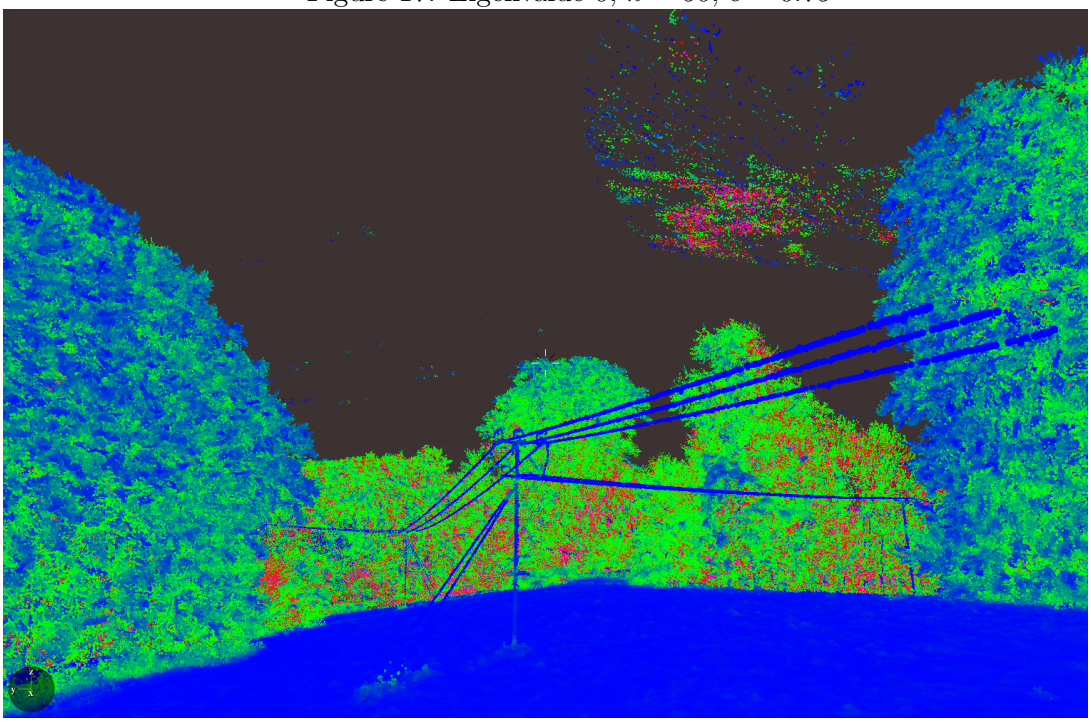
There are some problems which can't be overcome without new tools. Figures 29 and 30 show plots in which points have lower rank than they should (especially points with rank 0), despite us plotting with $\theta = 0.001$. The reason for this is that the acquisition of data is not made at a constant rate across space – and we can prove this with some interesting plots. Figures

31 and 32 show space-time diagrams[1] which correspond to Figures 29 and 30 respectively. These diagrams demonstrate that the objects with mistakenly low rank occur precisely in the regions where time passes but positions remain unchanged – this may be where the aircraft has rotated (which is the case for the points here) or where the aircraft has slowed down to change direction.

Application domain. *Rank can be used to labels all points with a "dimension". Conductors should be rank 1, buildings should be rank 2 or 3, and vegetation varies between rank 1, 2 and 3. Exterior vegetatation is typically of rank 3, and the ground is typically of rank 2. However, this is a discrete signal so it can be too harsh: Noise and varying point density can change the nature of this signal.*

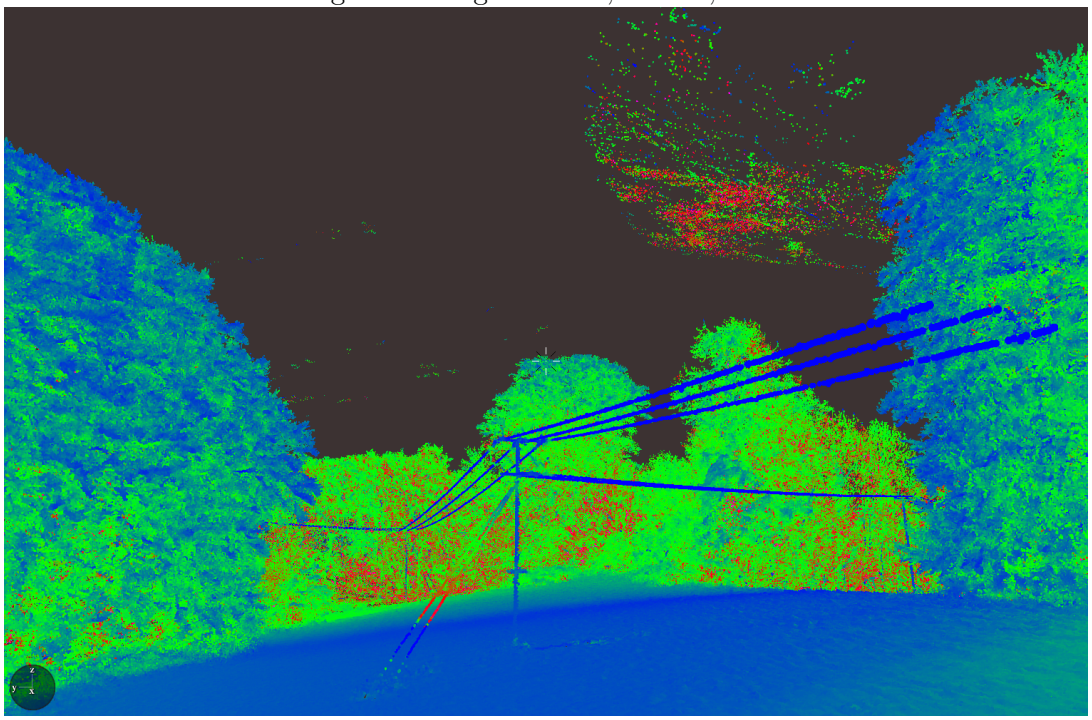
[1] In a space-time diagram, x and y are left unchanged, but z is replaced by time.

Figure 17: Eigenvalue 0, $k = 50$, $\epsilon = 0.75$



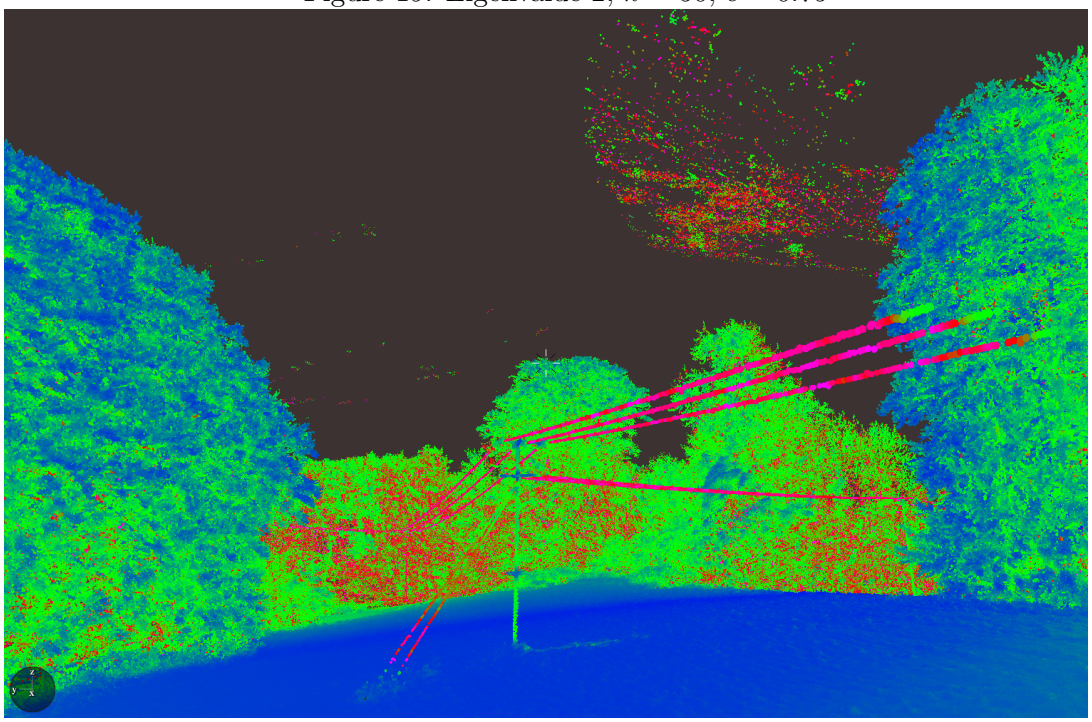
Has smaller values in non-voluminous regions (ground and conductors)

Figure 18: Eigenvalue 1, $k = 50$, $\epsilon = 0.75$



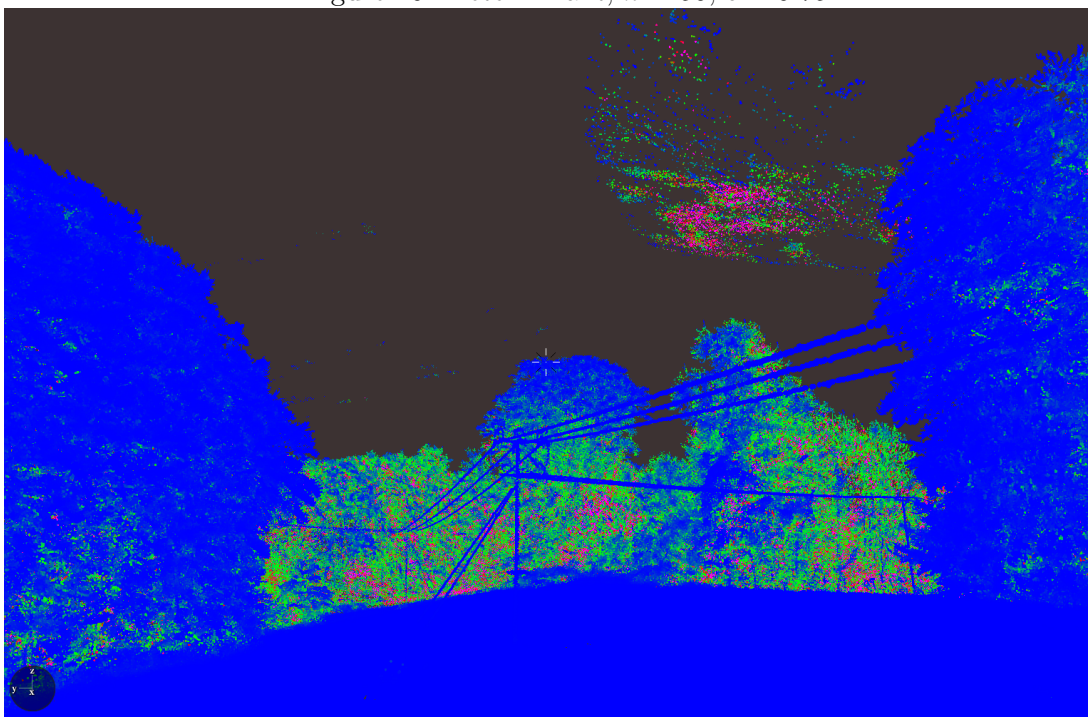
Has smaller values in 1D objects (conductors)

Figure 19: Eigenvalue 2, $k = 50$, $\epsilon = 0.75$



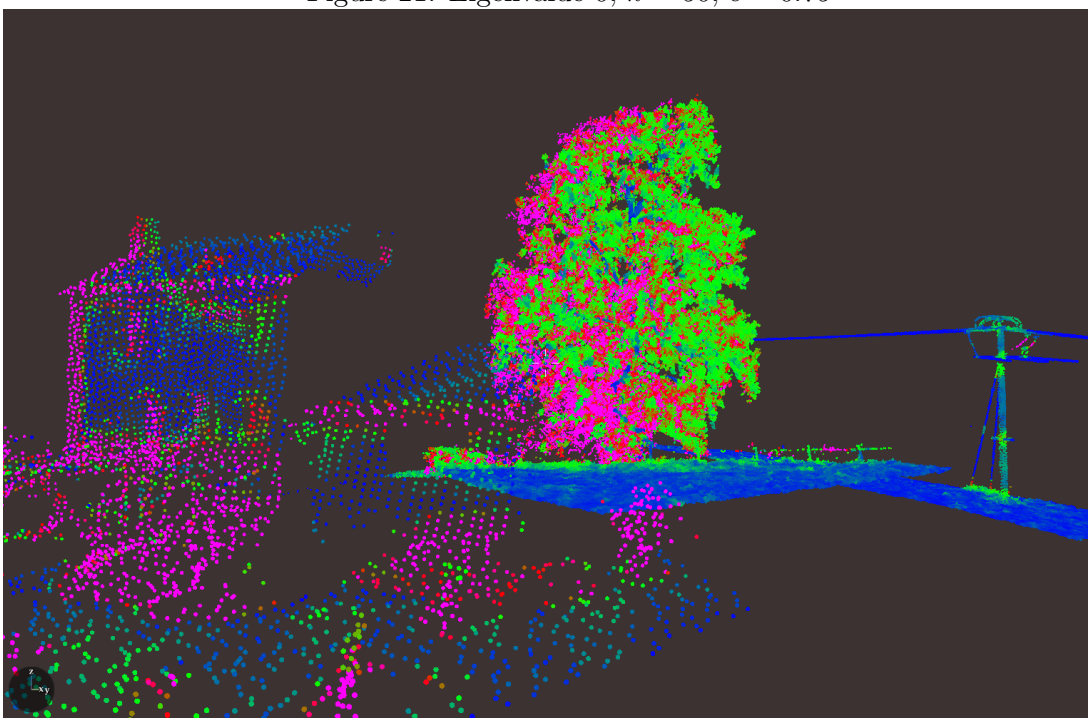
Has small values in heavily sampled regions where points are close together. Very high values on 1D objects (conductors, branches).

Figure 20: Determinant, $k = 50$, $\epsilon = 0.75$



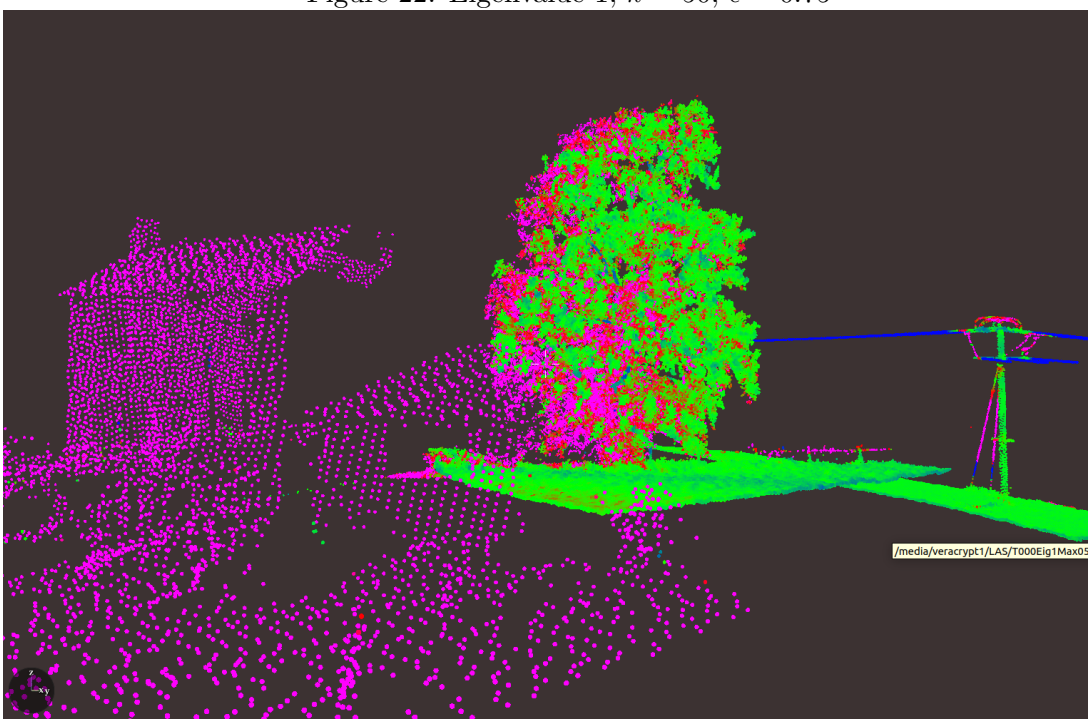
Low values in the most oversampled (closest to flight line) regions and in areas which are not 3D (away from vegetation)

Figure 21: Eigenvalue 0, $k = 50$, $\epsilon = 0.75$



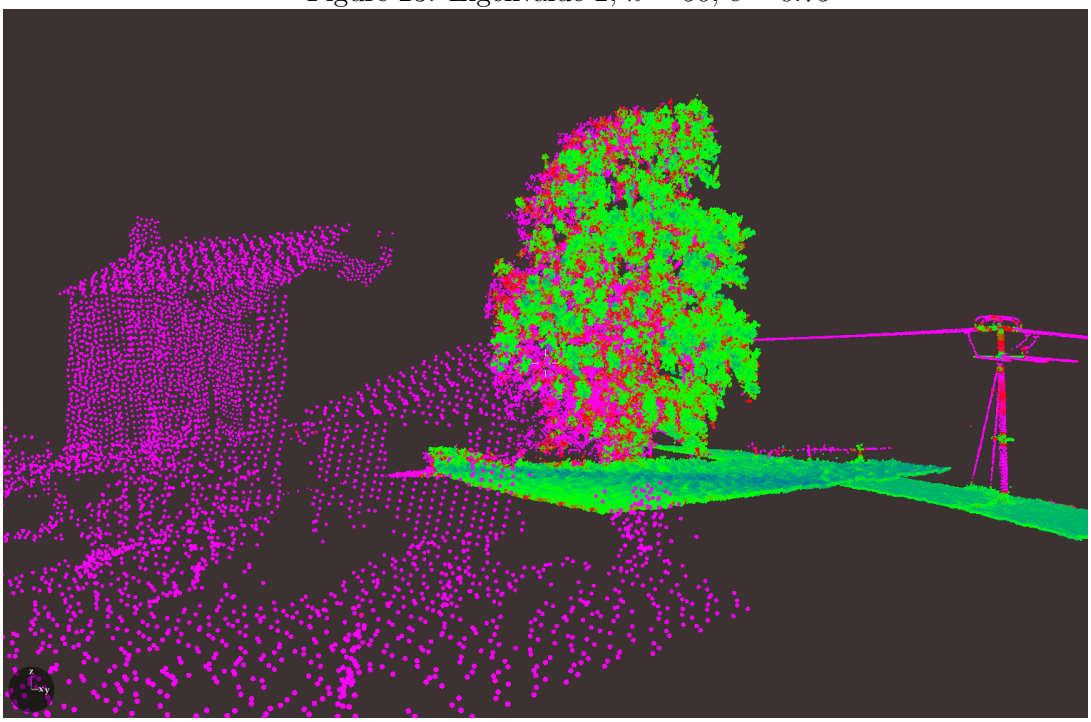
Tree and building corners (voluminous regions) have higher values. 2D and 1D objects have lower values.

Figure 22: Eigenvalue 1, $k = 50$, $\epsilon = 0.75$



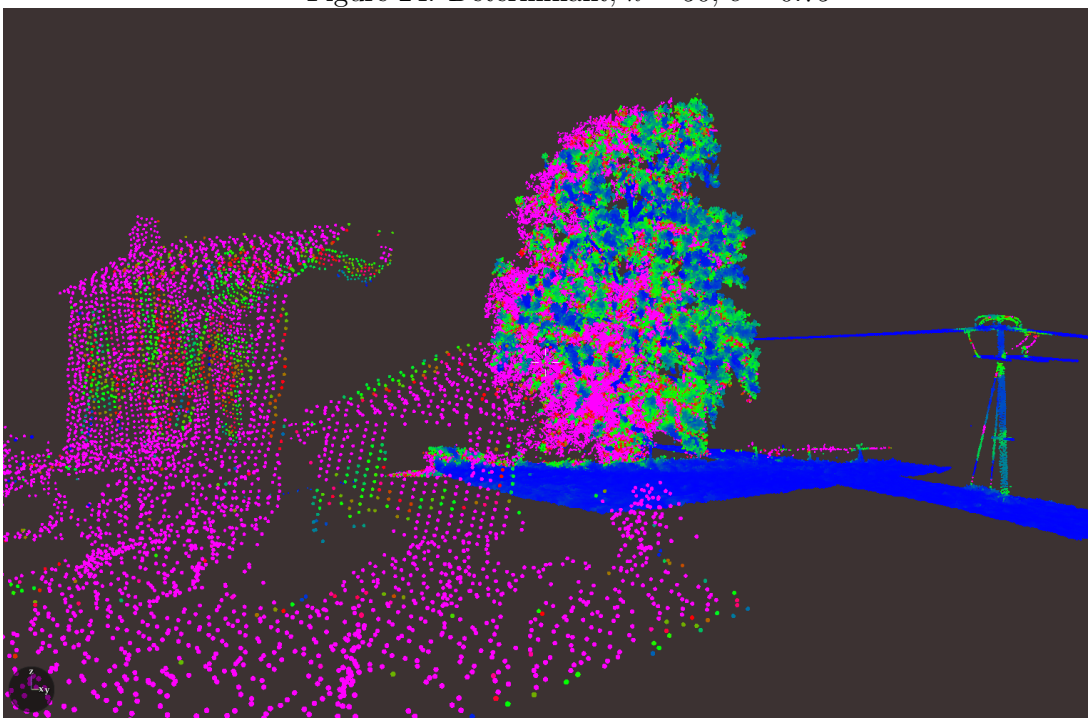
We observe an increase from λ_0 in the 2D objects.

Figure 23: Eigenvalue 2, $k = 50$, $\epsilon = 0.75$



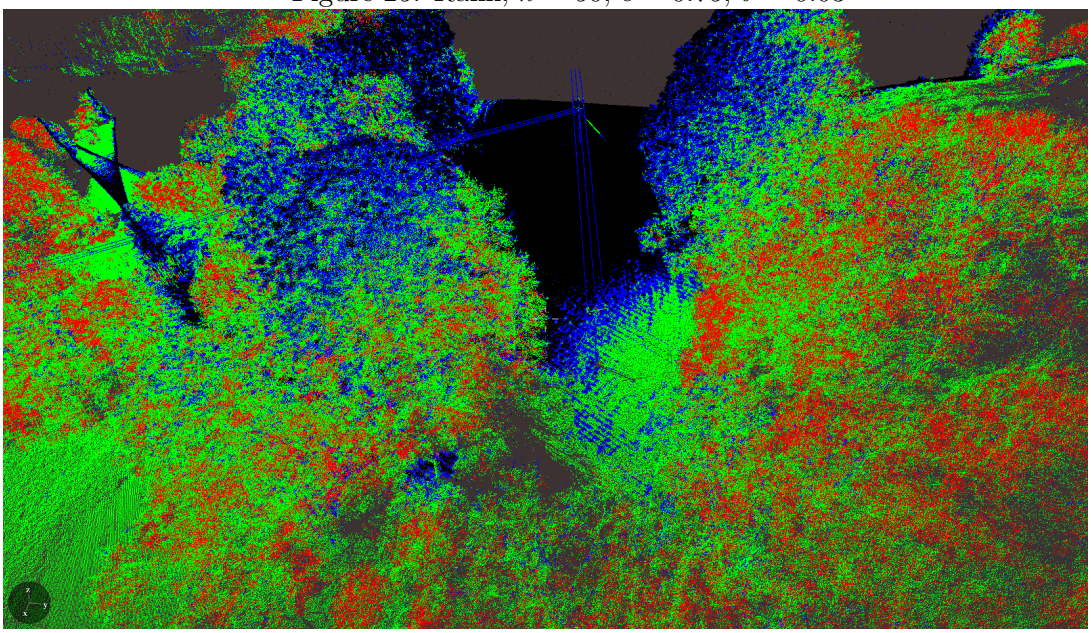
Most objects have quite high values except the vegetation which has a high point density.

Figure 24: Determinant, $k = 50$, $\epsilon = 0.75$



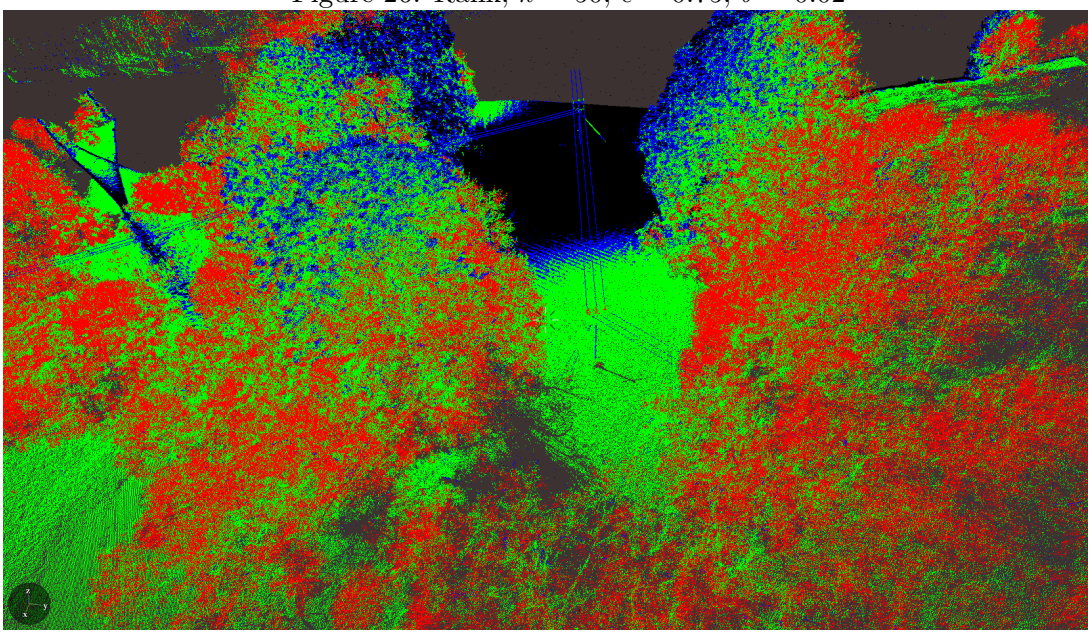
Least degenerate objects (e.g. vegetation and edges) have highest values.

Figure 25: Rank, $k = 50$, $\epsilon = 0.75$, $\theta = 0.03$



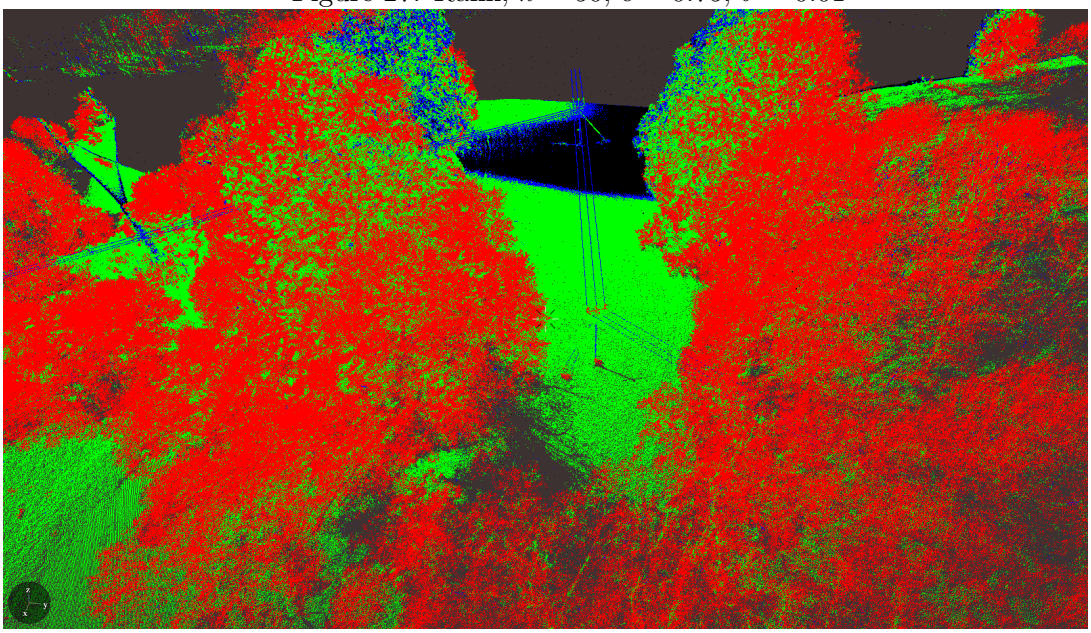
Rank should be a representation of dimension – but we've picked a threshold too large!

Figure 26: Rank, $k = 50$, $\epsilon = 0.75$, $\theta = 0.02$



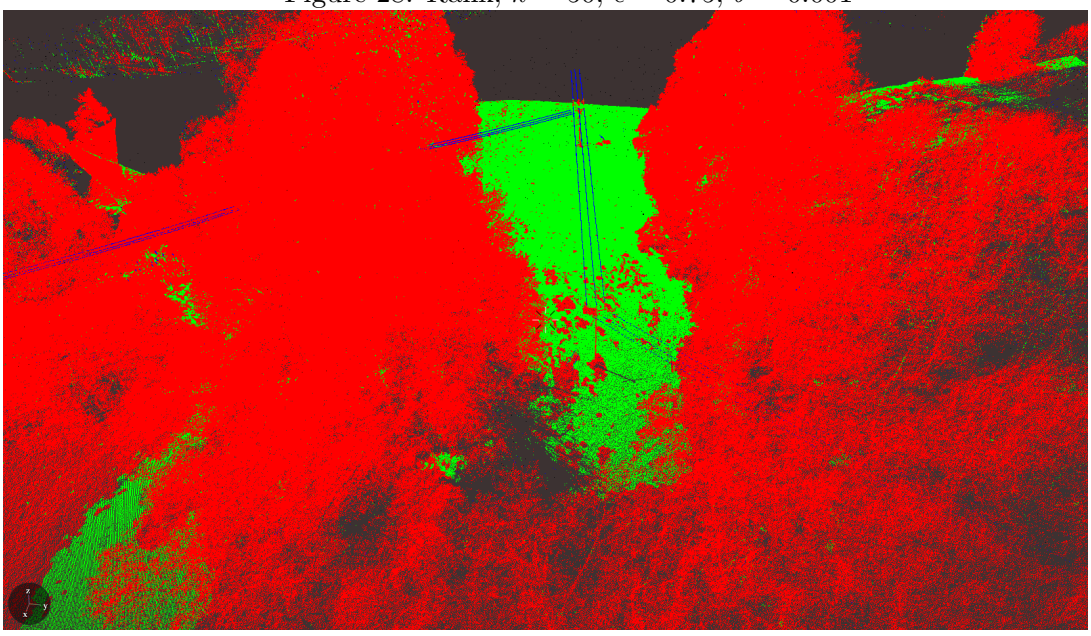
Reducing threshold...

Figure 27: Rank, $k = 50$, $\epsilon = 0.75$, $\theta = 0.01$



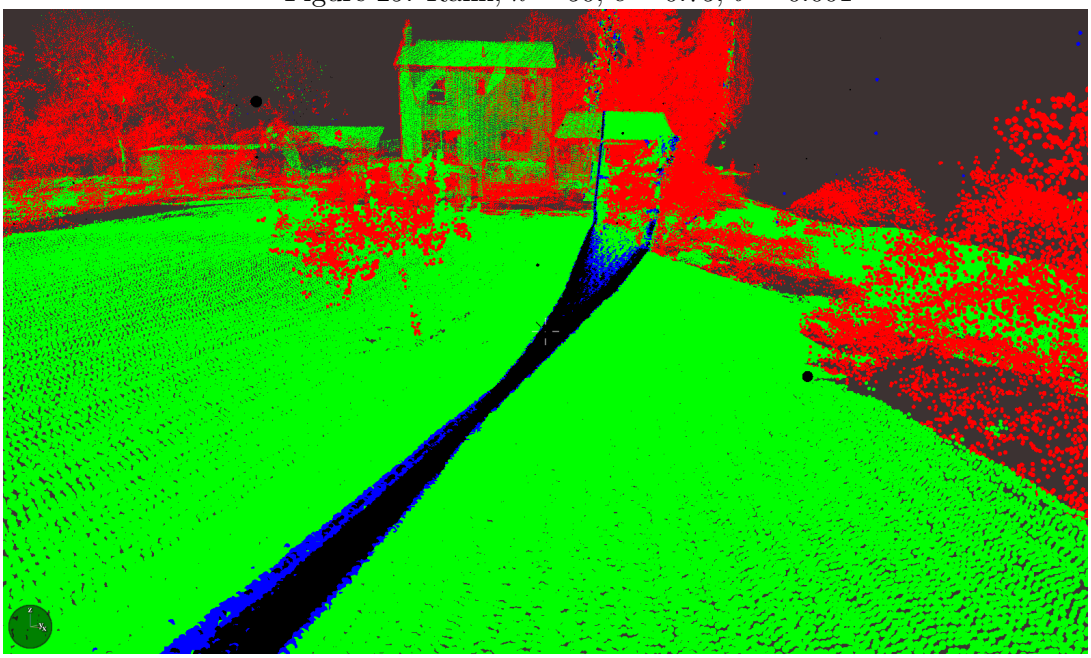
Reducing threshold...

Figure 28: Rank, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$



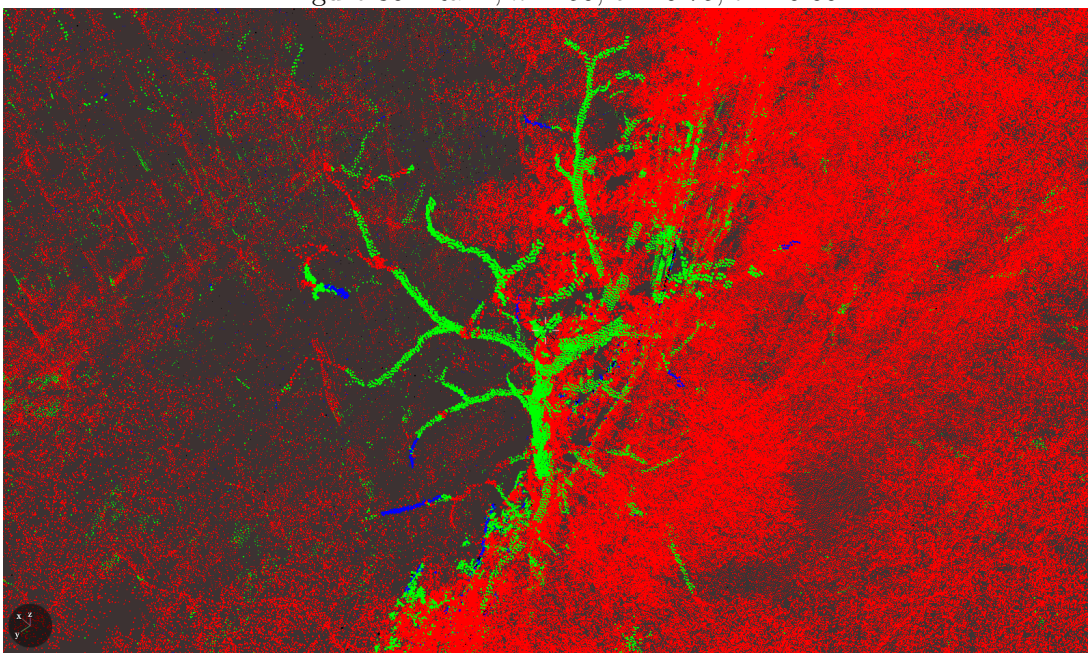
Much better – a good representation of dimension. Conductors are 1D (blue), ground 2D (green), threes 3D (red)

Figure 29: Rank, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$



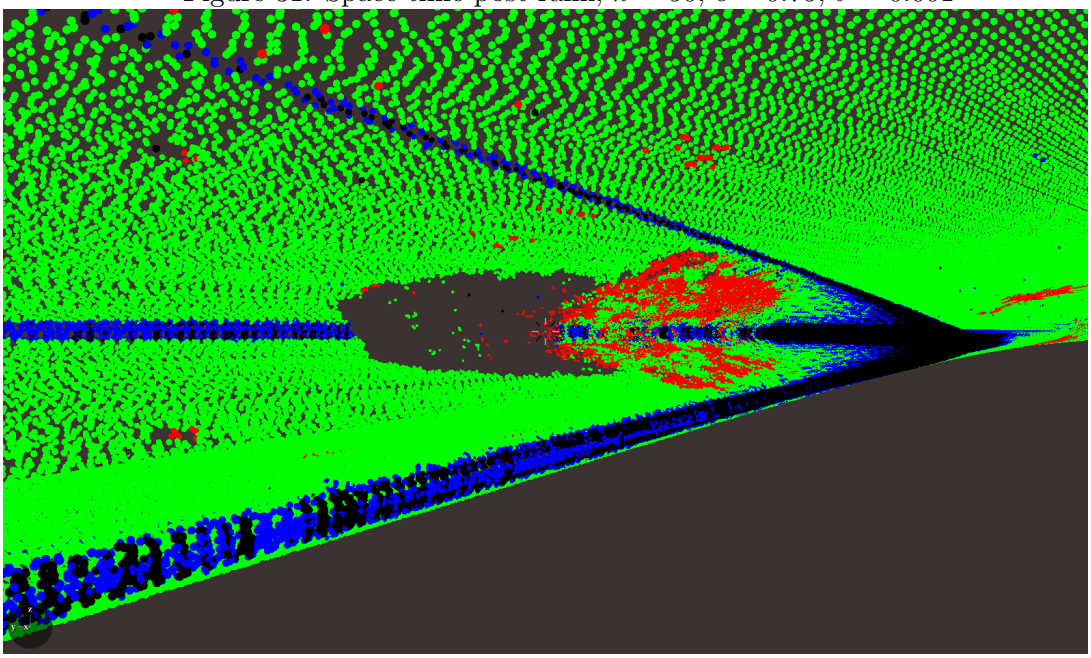
A nasty stain on our rank plot from oversampling

Figure 30: Rank, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$



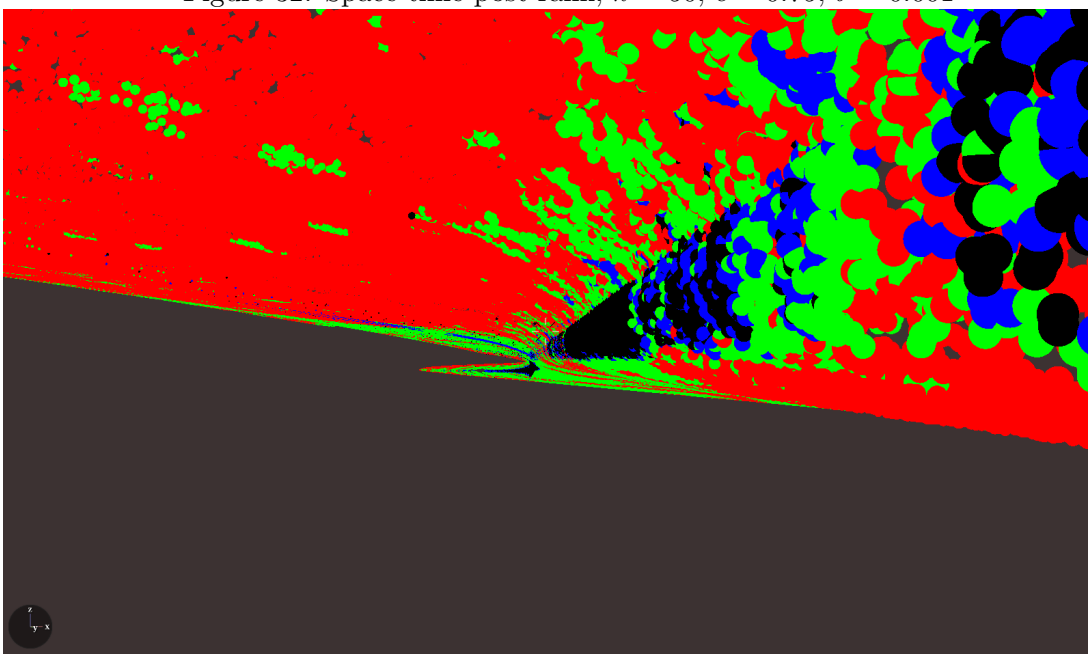
Trees with no leaves on them are mostly 2D objects

Figure 31: Space-time post rank, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$



Evidence that turning points of aircraft cause low ranks – see lower ranks in the folds, which should be 2D (green) ground objects

Figure 32: Space-time post rank, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$



Evidence that turning points of aircraft cause low ranks – see lower ranks in the folds, which should be 3D (red) vegetation objects

1.2.6 Eigenvalue ratios

Signal. *The ratios of successive eigenvalues* λ_2/λ_1 *and* λ_1/λ_0.

NEED TO MAKE REMARK ABOUT DIVISION BY 25 and 6.

SVD rank is a rather harsh measurement, because it imposes a *global* threshold on raw eigenvalues in order to compute a rank. It could be more forgiving to instead look at the successive ratios λ_2/λ_1 and λ_1/λ_0 of eigenvalues.

Figures 33 and 34 show plots of λ_2/λ_1. The objects with heighest value are the 1D regions of the point cloud, and this aligns with the idea that such regions occur when λ_2 is large but λ_1 is small. Note also the *edge detecting properties* of this signal: The edges of a flat plane experience an uplift in value of this signal in comparison to the plane's interior.

Figures 35 and 36 show plots of λ_1/λ_0. The areas with highest value seem to the 2-dimensional regions of the point cloud. The lowest values are given to the 3D and 1D objects. Note that, as shown in Figure 35, the space-time problem which we have discussed in previous sections shows up again.

Application domain. λ_2/λ_1 *can be used for 2D edge detection. Eigenvalue ratios be used as an alternative to rank for determining the dimensions of various objects.*

1.2.7 Remark on means – should we use them?

When we produce statistical signals from a family of n points, we start with:

- Vectors $x = (x_1, x_2, x_3, \ldots, x_n)$, $y = (y_1, y_2, y_3, \ldots, y_n)$, and $z = (z_1, z_2, z_3, \ldots, z_n)$, where (x_i, y_i, z_i) is the position of the i-th point.
- Matrices X, Y and Z of shape $n \times k$ where X_{ij} is the x-component of the nearest neighbour of point, and similarly for Y and Z, where k is the number of neighbours we have chosen to use.
- We then produce vectors $\overline{x}$, $\overline{y}$ and $\overline{z}$ of length n, where $\overline{x}_i = \frac{1}{k}\sum_j X_{ij}$, the average x-component of the k neighbours of point i, and similarly for $\overline{y}$ and $\overline{z}$.
- Produce the matrices

$$A_i = \frac{1}{\sqrt{k}} \begin{pmatrix} X_{i0} - \overline{x}_i & X_{i1} - \overline{x}_i & X_{i2} - \overline{x}_i \ldots X_{ik} - \overline{x}_i \\ Y_{i0} - \overline{y}_i & Y_{i1} - \overline{y}_i & Y_{i2} - \overline{y}_i \ldots Y_{ik} - \overline{y}_i \\ Z_{i1} - \overline{z}_i & Z_{i1} - \overline{z}_i & Z_{i2} - \overline{z} \ldots Z_{ik} - \overline{z}_i \end{pmatrix}.$$

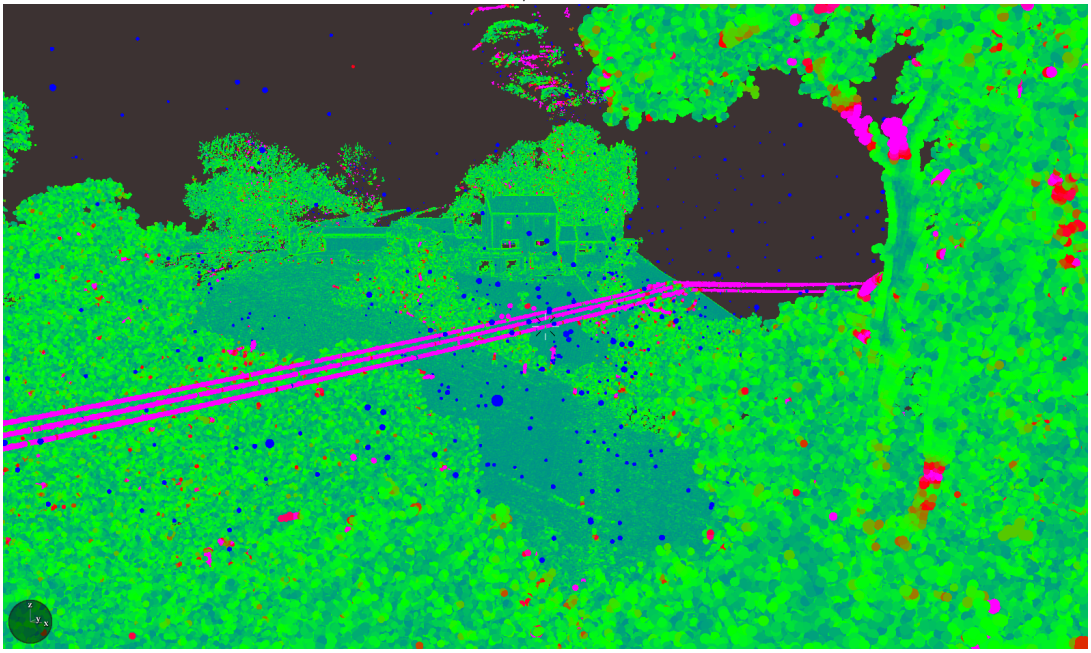
- Obtain the covariance matrices $C_i = A_i A_i^T$, and proceed to find eigenvalues, etc.

There's a minor point here which could be adjusted. The idea is that the rank of A_i shows us the dimension of the object that point i is living in. However, it does this in a curious way: By looking at that objection "from the perspective of $\overline{x}_i$". And, although it's traditional to take the arithmetic mean of the points in the neighbourhood of point i, we actually really needn't

do that. We could instead take $\overline{x} = x$ and $\overline{y} = y$ and $\overline{z} = z$ – that is, we could choose to not take means at all! And this makes complete sense, because it means that we no longer look at the object from the point of view of the average point, but we look at it from the point of view of our target point when taking signals. As far as I can tell, there's not much wrong with doing this, and it saves us from doing some calculations. It will mean, however, that we get different values for some signals, but in principal it should be recognised that we can alter the meaning of $\overline{x}$, $\overline{y}$ and $\overline{z}$.

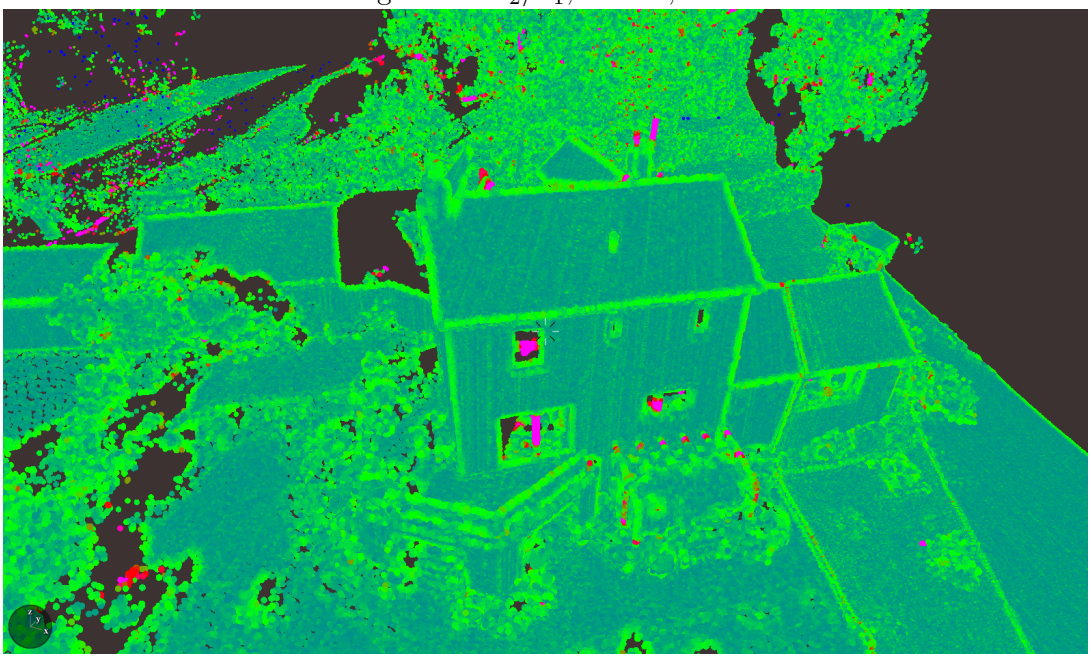
Our decision on this issue doesn't make a lot of difference, so we will actually plot with $\overline{x} = x$, $\overline{y} = y$ and $\overline{z} = z$. This saves some computational effort.

Figure 33: λ_2/λ_1, $k = 50$, $\epsilon = 0.75$



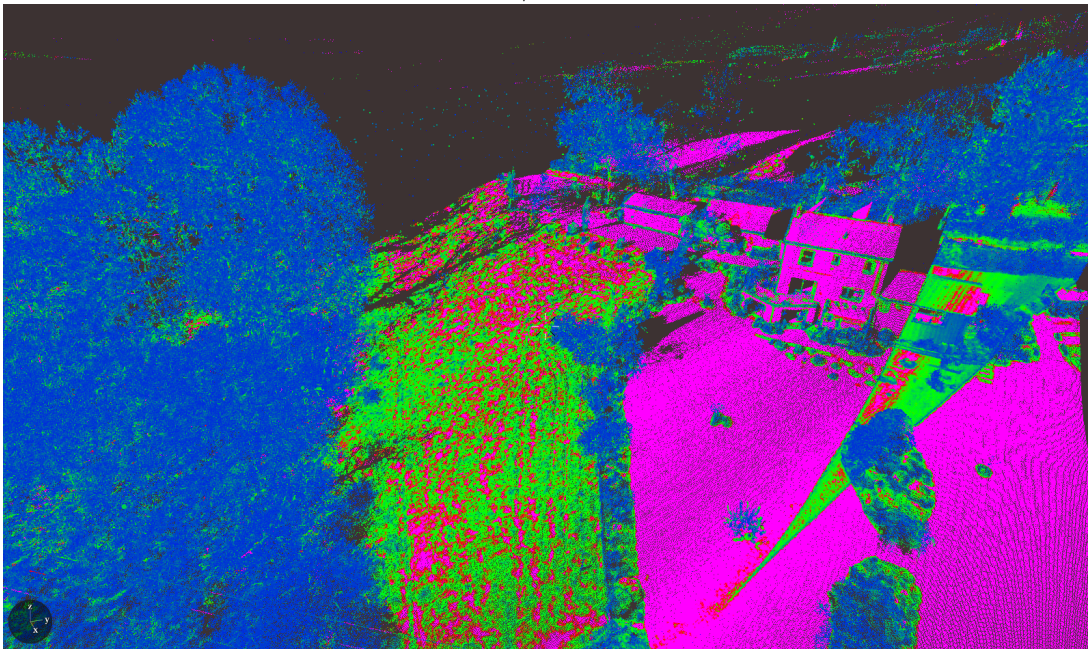
1D objects have very high (pink) values. Isolated noise has low values

Figure 34: λ_2/λ_1, $k = 50$, $\epsilon = 0.75$



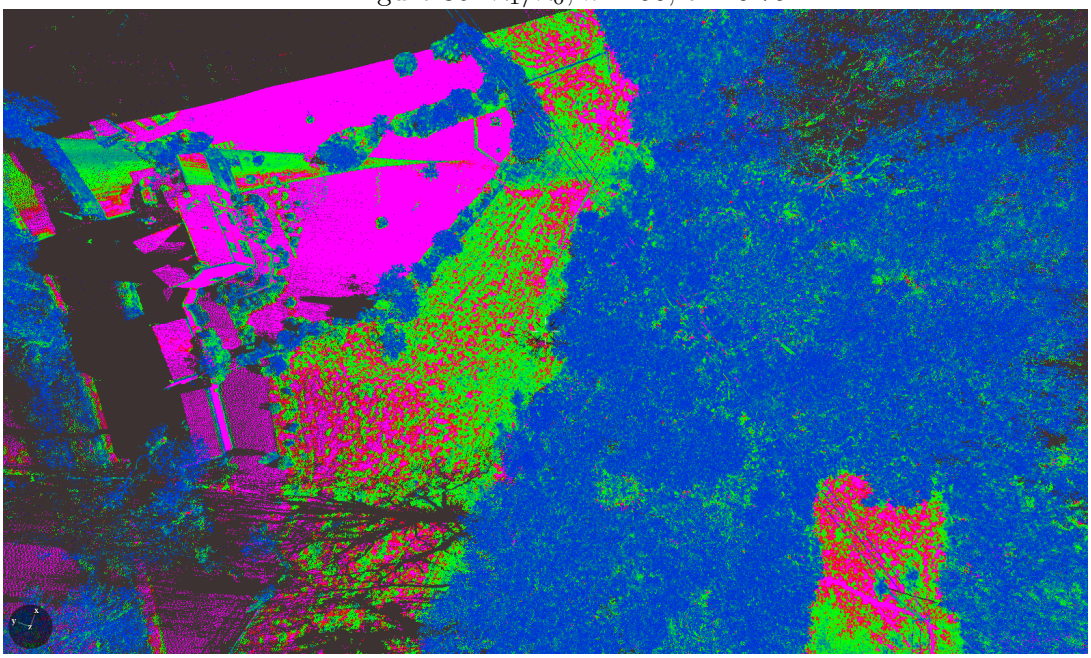
Observe edge-detecting properties

Figure 35: λ_1/λ_0, $k = 50$, $\epsilon = 0.75$



(Ignore the obvious flaw - we will show how to clean later.) 2D objects (ground, buildings) have very high values.

Figure 36: λ_1/λ_0, $k = 50$, $\epsilon = 0.75$



(Ignore the obvious flaw - we will show how to clean later.) 2D objects (ground, buildings) have very high values. 1D and 3D objects have lower values.

1.2.8 Curvature, isotropy, and angles

Suppose the points (x_i, y_i, z_i), $i = 1, 2, \ldots, k$, which we are considering are concurrent on a surface, e.g. a tree trunk, side of a building, or the ground. It can be shown (using pseudoinverses of matrices – see [1, Theorem 7.8]) that the equation for a point $\mathbf{p}$ on the plane of best fit through $\overline{\mathbf{r}}$ is

$$(\mathbf{p} - \overline{\mathbf{r}}) \cdot \mathbf{v}_0 = 0.$$

where $\overline{\mathbf{r}} = (\overline{x}, \overline{y}, \overline{z})$ and $\mathbf{v}_0$ is the eigenvector for λ_0. Recall that the eigenvectors of the covariance matrix C form a basis $\mathcal{B} = \{\mathbf{v}_0, \mathbf{v}_1, \mathbf{v}_2\}$ of $\mathbb{R}^3$. Since $C\mathbf{v}_i = \lambda_i \mathbf{v}_i$, $\lambda_i^{1/2}$ is the "standard deviation along the direction $\mathbf{v}_i$".

Signal. *Since the total standard deviation from the center $\overline{\boldsymbol{r}}$ is $\lambda_0 + \lambda_1 + \lambda_2$, the ammount by which $\boldsymbol{v}_0$ is to blame for the deviation is*

$$\frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}.$$

*We call this the **curvature** or **normal curvature** of the surface given by these points.*

Application domain. *Some objects have a very specific set of values for normal curvature. 1D lines and curves like conductors should have a very low normal curvature because they tend to fall within a plane. Obviously, flat planes also have a low curvature.*

The maximum value which this ratio can have is $1/3$, in which case the points can be said to be isotropic.

To measure isotropy (i.e. how evenly spread our points are), we need to measure how close the vector $\mathbf{L} = (\lambda_0, \lambda_1, \lambda_2)$ is to being equal to $(\lambda_2, \lambda_2, \lambda_2)$.

Signal. *Therefore, we will measure it by the cosine of the acute angle between $\boldsymbol{L}$ and $\boldsymbol{d} = (1, 1, 1)$. That is, we define the new quantity **isotropy***

$$\frac{|\boldsymbol{L} \cdot \boldsymbol{d}|}{\|\boldsymbol{L}\| \cdot \|\boldsymbol{d}\|} = \frac{\lambda_0 + \lambda_1 + \lambda_2}{\sqrt{3(\lambda_0^2 + \lambda_1^2 + \lambda_2^2)}}.$$

Application domain. *Isotropy is another signal which can be used in the selection of objects. For example, a perfect straight line would have $\lambda_0 = \lambda_1 = 0$ and therefore would have isotropy of $1/\sqrt{3} \approx 0.5773$, so we would expect 1D objects to have roughly that value of isotropy.*

Isotropy is similar to regression – it is a number between 0 and 1. When it is zero, then there is no spread or "directionality" whatsoever in the set of points (note that when $\lambda_2 = 0$,

we set isotropy to 0). When isotropy is close to 1, then there is a spread of points along three orthogonal directions in space.

Figures 37 and 38 show plots of curvature. We can see in those plots that mown lawn and flat, man-made surfaces, have low curvature. Also, in Figure 38, we can see the curved, flat surfaces of a tree trunk which have high values, whereas flatter areas have low values.

Figures 39 and 40 show plots of isotropy. Isotropy has high values at areas which are spread in all directions, medium values in areas with 2D, and lower values in areas with 1D spread. In particular, high values are given to foliage, distinguishing it from other parts of vegetation, and the conductor has only a very particular set of values: About 57-61% isotropy. One can see the difference between a tree which has no foliage and a tree which has some in Figure 39.

In Figures 41 - 43, it is demonstrated that the conductor exists in a narrow band of isotropies as mentioned above. Some vegetation also exists in this range, but buildings are almost completely unseen in that same range.

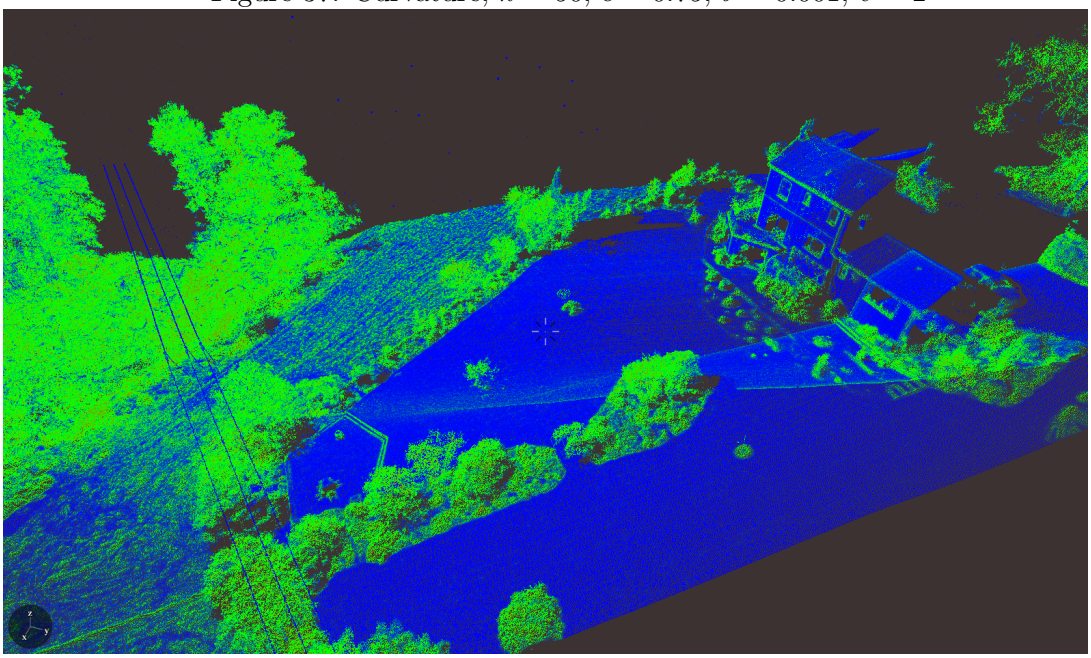
Because we know that eigenvector zero, $\mathbf{v}_0$, is the normal to the plane of best fit, we can calculate the angle between $\mathbf{v}_0$ and the vertical direction in order to give a measurement for the gradient of a surface. See Figures 45 and 46 for plots. (Note that this signal is *damaged* by doing calculations in space-time in the sense of Section 1.3.4.)

Similarly, eigenvector two, $\mathbf{v}_2$, is the direction of the line of best fit, so we can calculate the angle between $\mathbf{v}_2$ and the vertical direction in order to give a measurement for the gradient of a line. See Figures 47 and 48 for plots.

Signal. *From the eigenvector $\boldsymbol{v}_0$ we can deduce the signal for **planar angle**, the angle between the vertical direction and plane of best fit. From $\boldsymbol{v}_2$ we can deduce **linear angle**, the angle between the vertical direction and the line of best fit.*

Application domain. *Certain objects only exist within a certain family of linear and planar angles. For example, conductors only "dip" so much, so we can use linear angles to refine a search for conductors, and pylons have a near-vertical planar angle.*

Figure 37: Curvature, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 2$



Outer vegatation and edges of buildings have highest curvature

Figure 38: We can see the most violent "turning poings" in the trunk were curvature rises.

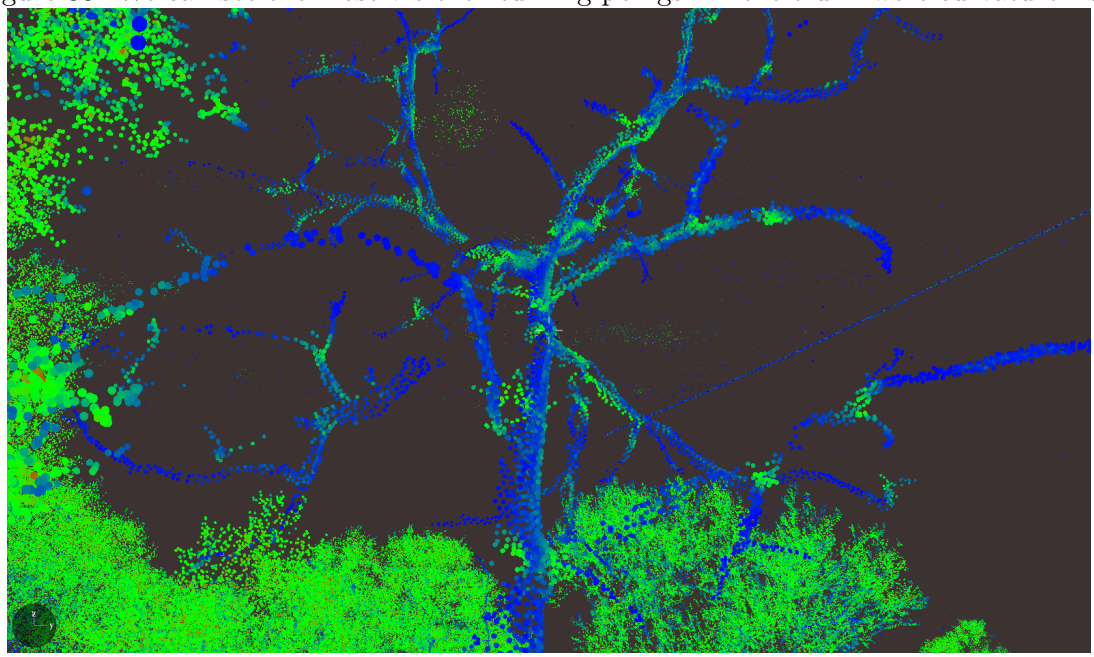
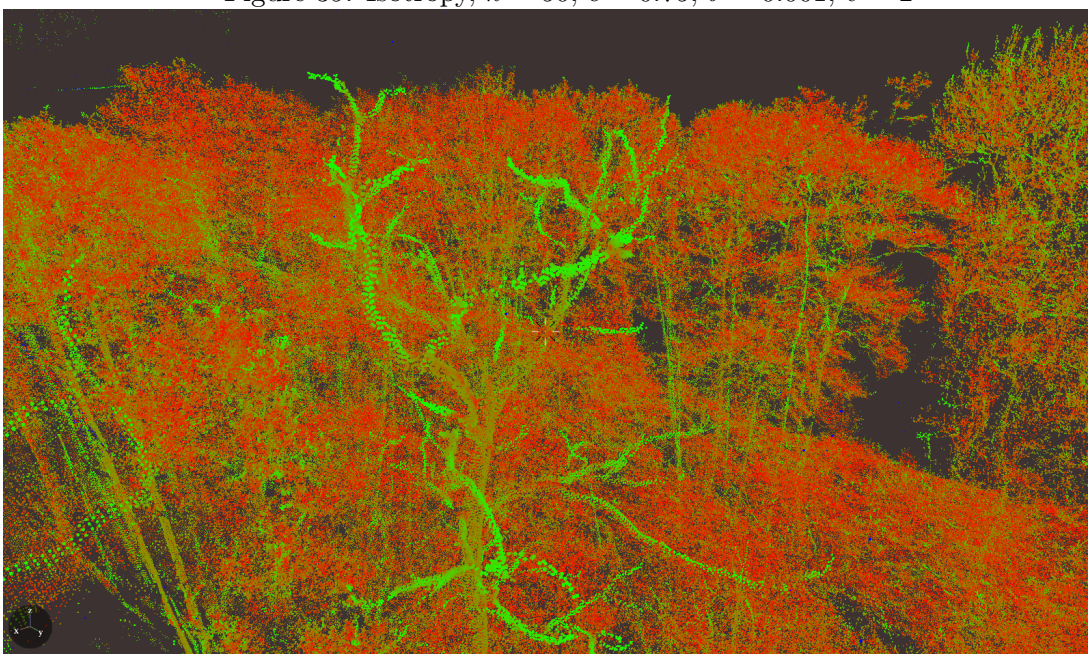
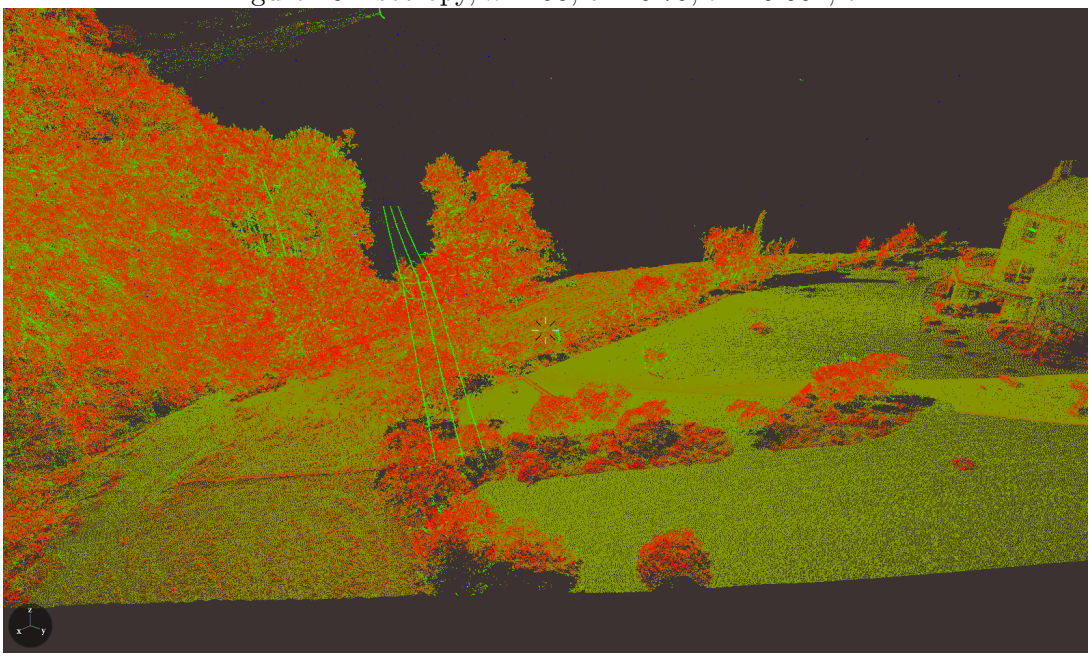


Figure 39: Isotropy, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 2$



Outer vegetation spreads out in all directions, giving highest possible values.

Figure 40: Isotropy, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 2$



The conductor (and window cross-bars) exists in a very narrow band of isotropies.

Figure 41: Isotropy, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 2$

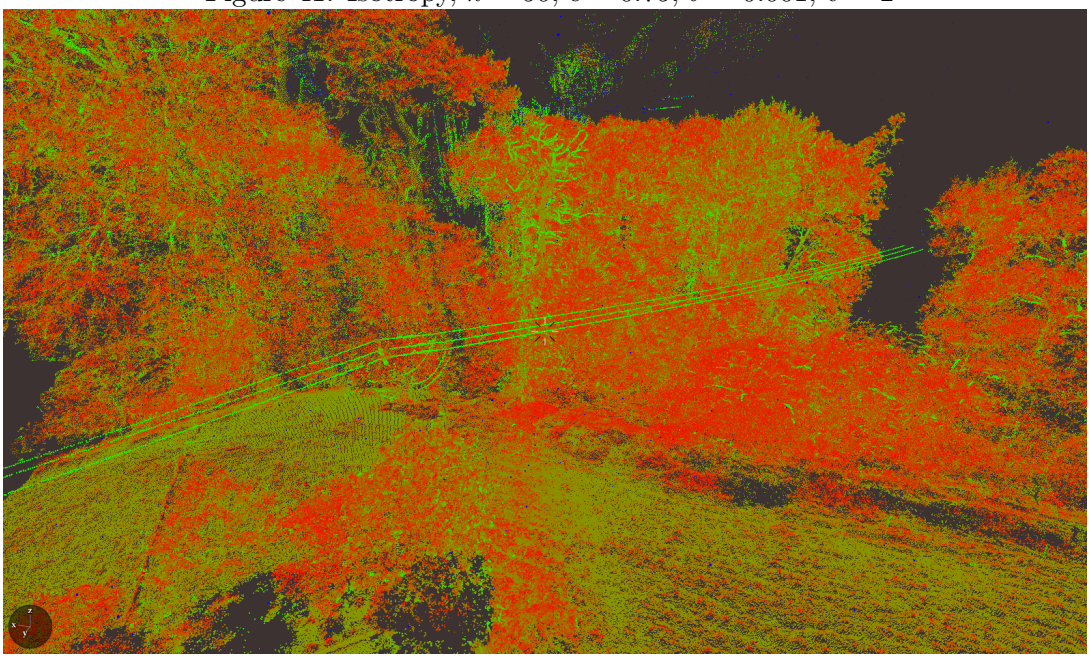
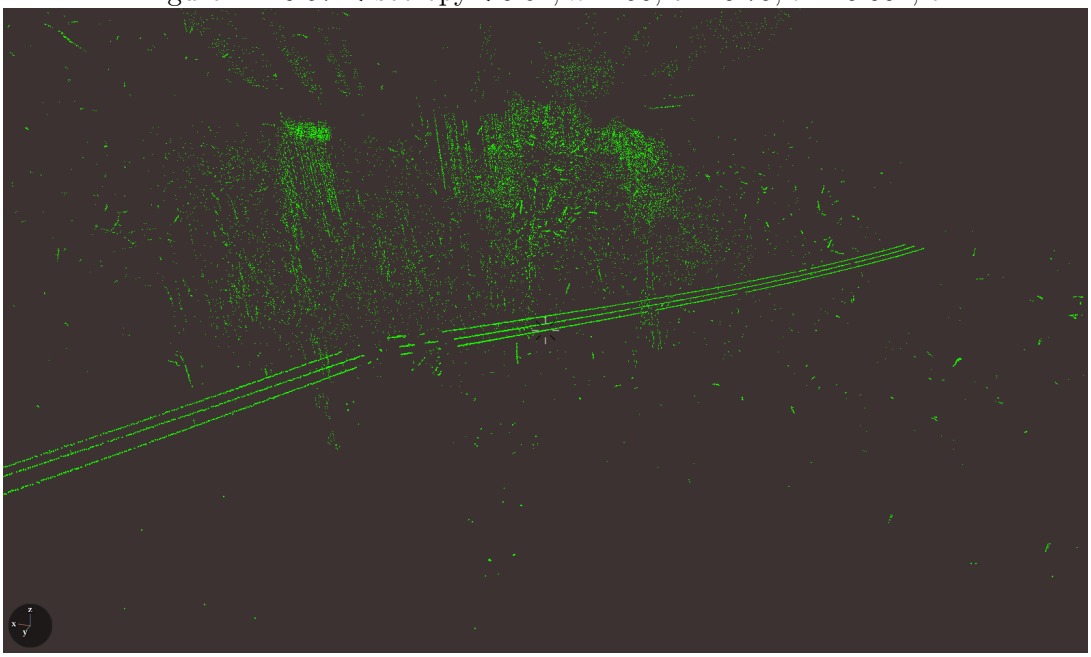
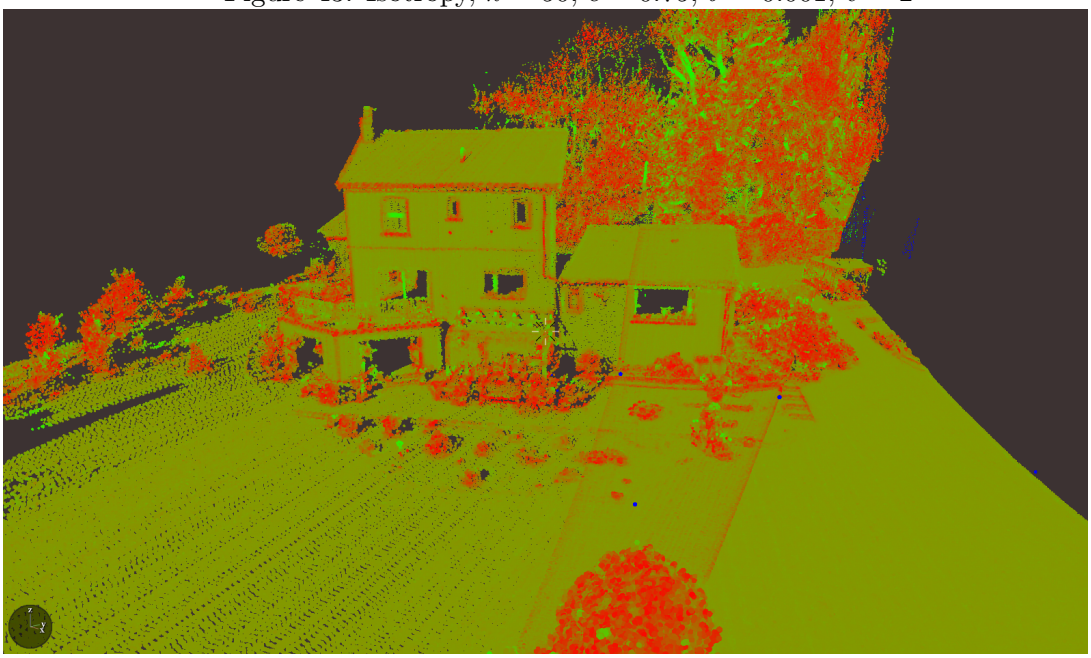


Figure 42: 0.57 <Isotropy< 0.61, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 2$



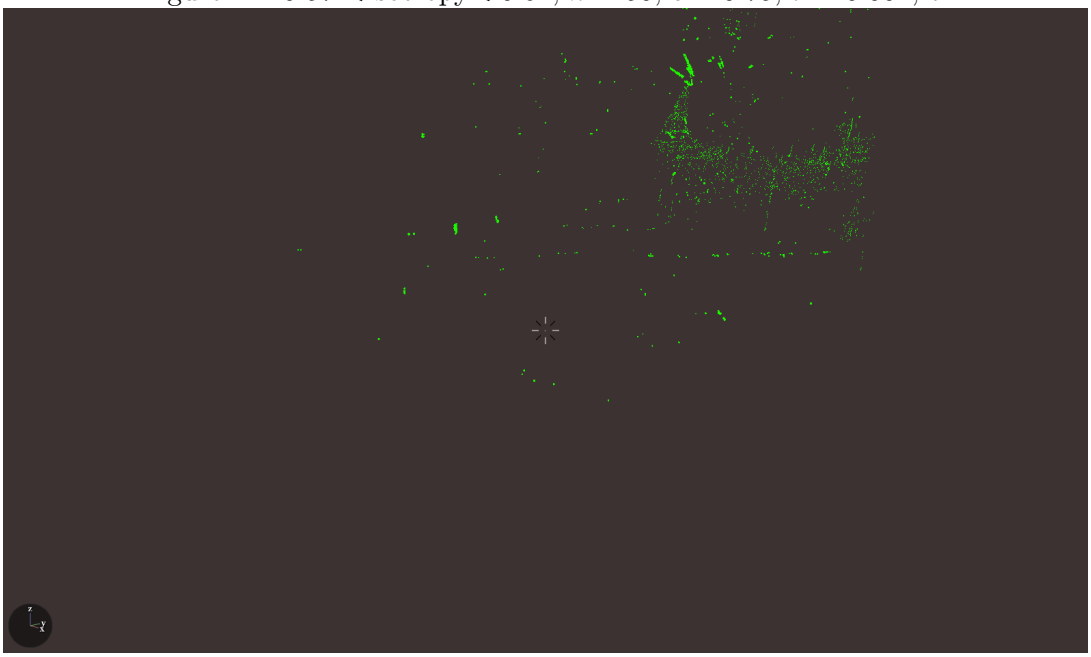
Demonstrating that we can extract the conductor plus "noise" in the form of other small 1D objects by using isotropy

Figure 43: Isotropy, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 2$



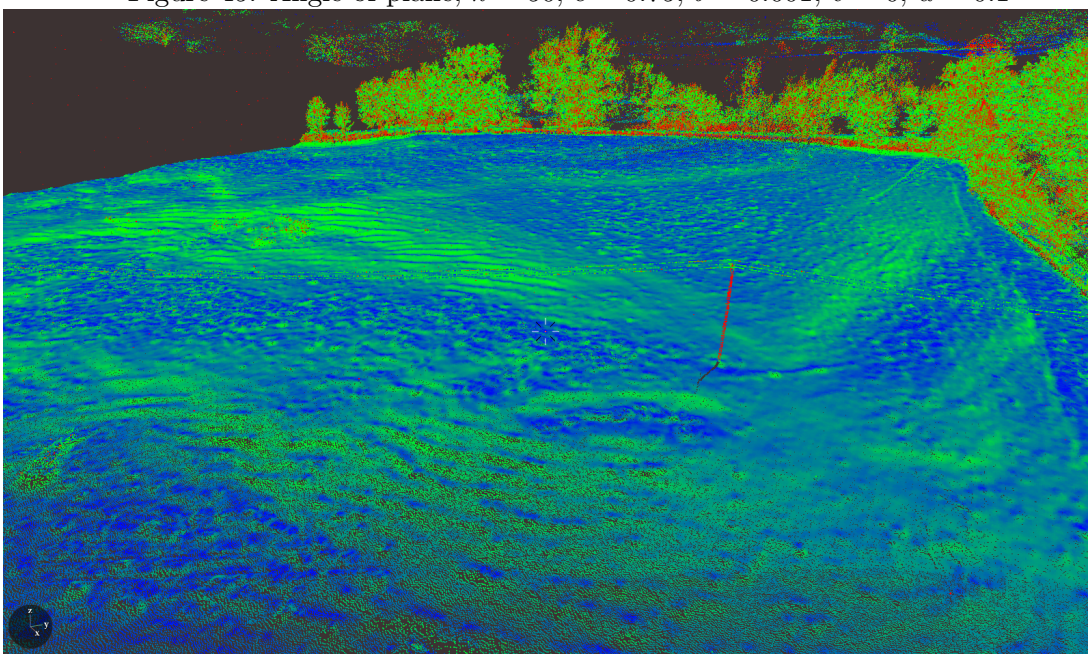
2D objects spread out in only two dimensions, so have a middle value of isotropy

Figure 44: 0.57 <Isotropy< 0.61, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 2$



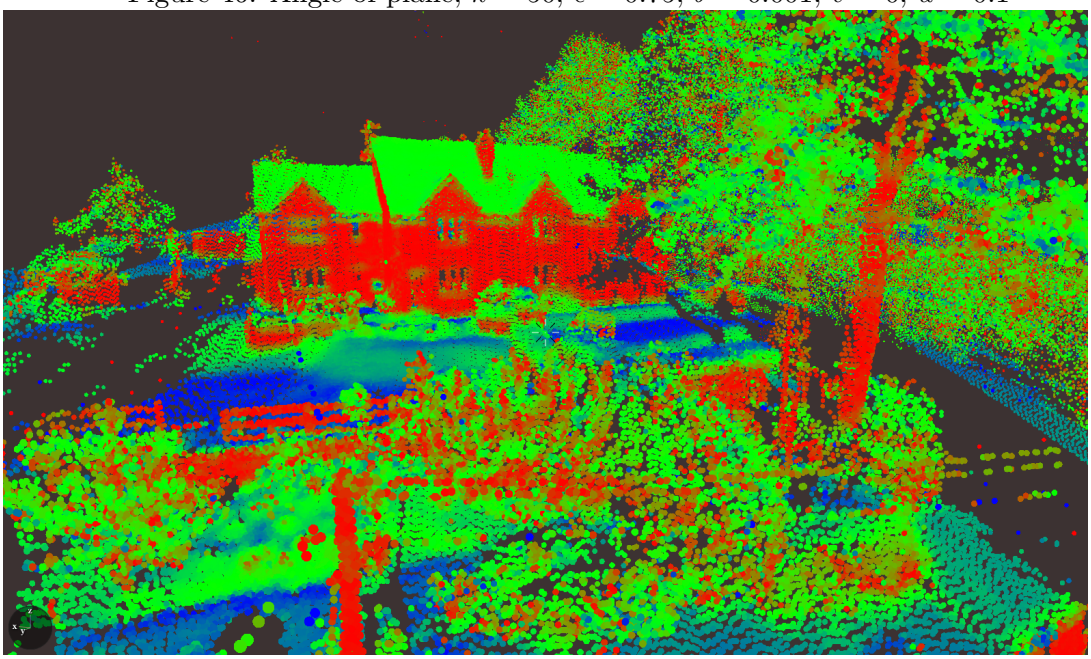
Only small features are left behind, from non-conductors, after applying this isotropy clip

Figure 45: Angle of plane, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 0$, $u = 0.1$



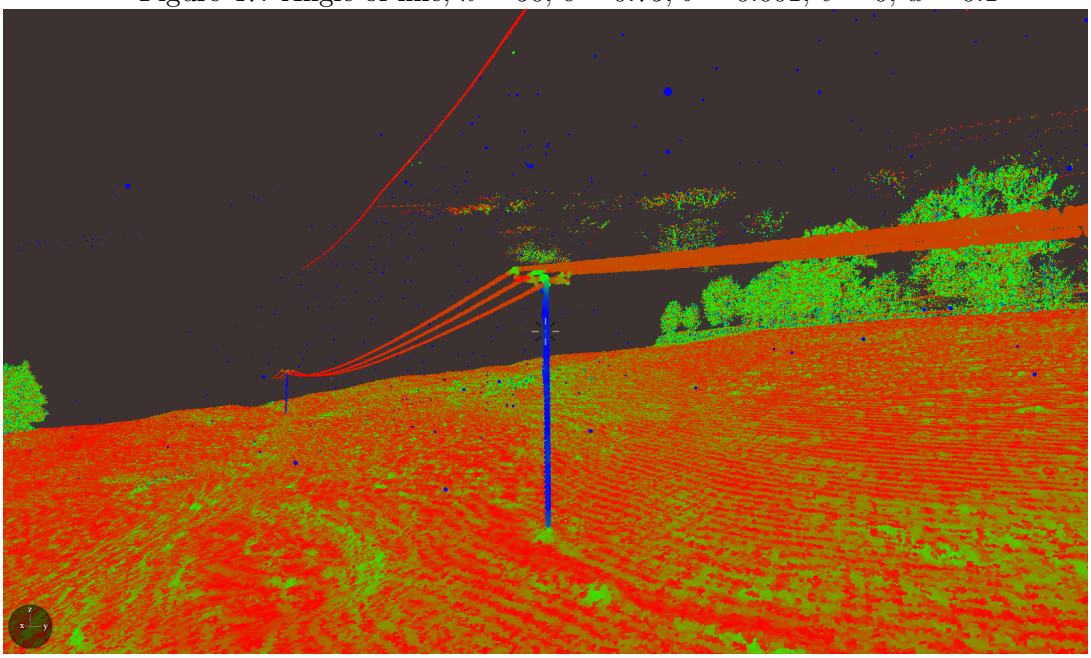
Undulating ground gives higher gradient. Pylones and other vertical objects have high values.

Figure 46: Angle of plane, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 0$, $u = 0.1$



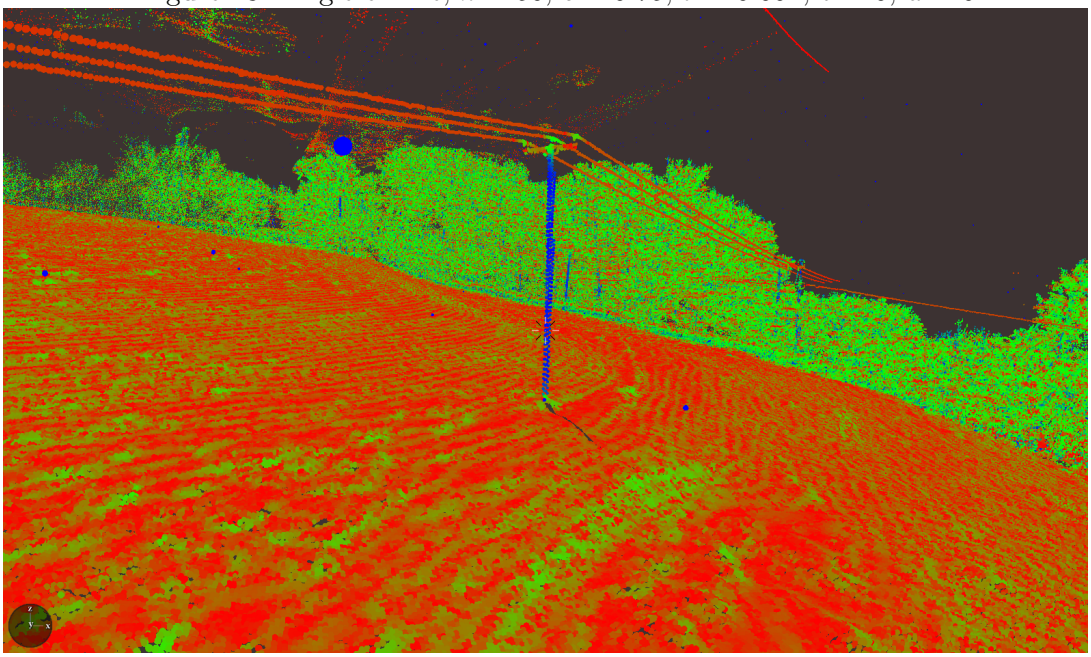
Detects sides of houses, fences, and distinguishes them from flat ground. Distinguishes major tree trunk from outer vegetation.

Figure 47: Angle of line, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 0$, $u = 0.1$



Conductors exist at a high and very restricted set of values.

Figure 48: Angle of line, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 0$, $u = 0.1$



Distinguishes conductor from pylon.

1.3 Data quality signals

Our method of plotting – using k nearest neighbours, before imposing a maximum distance ϵ – has some advantages. Binning points into spheres is computationally cumbersome, but taking nearest neighbours isn't so bad because there are some reasonably fast algorithms to do so. However, this method of plotting does give rise to some nasty phenomena. In this section we'll take some plots which illustrate and explain these issues, and discuss some methods of solving those problems.

1.3.1 Point density

When making our plots our plots, we first find the k nearest neighbours of our point and then find the corresponding k distances:

$$0 = d_0 \leqslant d_1 \leqslant d_2 \leqslant \cdots \leqslant d_{k-1}.$$

Therefore, there are k points within a sphere of radius d_{k-1}. This is efficient to compute: It is an improvement on more obvious methods to plot point density, which involve binning points into spheres.

Signal. *The **point density** is given by*

$$\frac{3k}{4\pi d_{k-1}^3}.$$

A plot of this signal is shown in Figure 49. While this signal is quite simple, it's an important one to keep in mind, because it shows us how well-sampled an area is. Indeed, the density of points in the trees gives credence to our complaint at the end of Section 1.2.5 that the rank was badly affected by over-sampling (which we will soon see is caused by the rotation of the aircraft).

Application domain. *Getting rid of some "noise" or undersampled data.*

1.3.2 Nearest neighbour distance

Signal. *We can find the nearest neighbour for each point in the point cloud. We can make arbitrary choices to break ties, but the **distance** to nearest neighbours is unambiguous, and gives us a new signal.*

In this section we will plot the distance of a point to its nearest neighbour.

Figure 52 shows a histogram of nearest neigbour distance. Compare this to the plot in Figure 51 which shows nearest neighbour distance as intensity. It's quite clear that the points which have the lowest values are densely packed near the flight line. In fact, as the histogram

illustrates, around 90% of the points exist within 5 centimeters of their nearest neighbour, and these points are typically very close to the flightline.

Application domain. *Identifying low quality data or irregularly sampled data.*

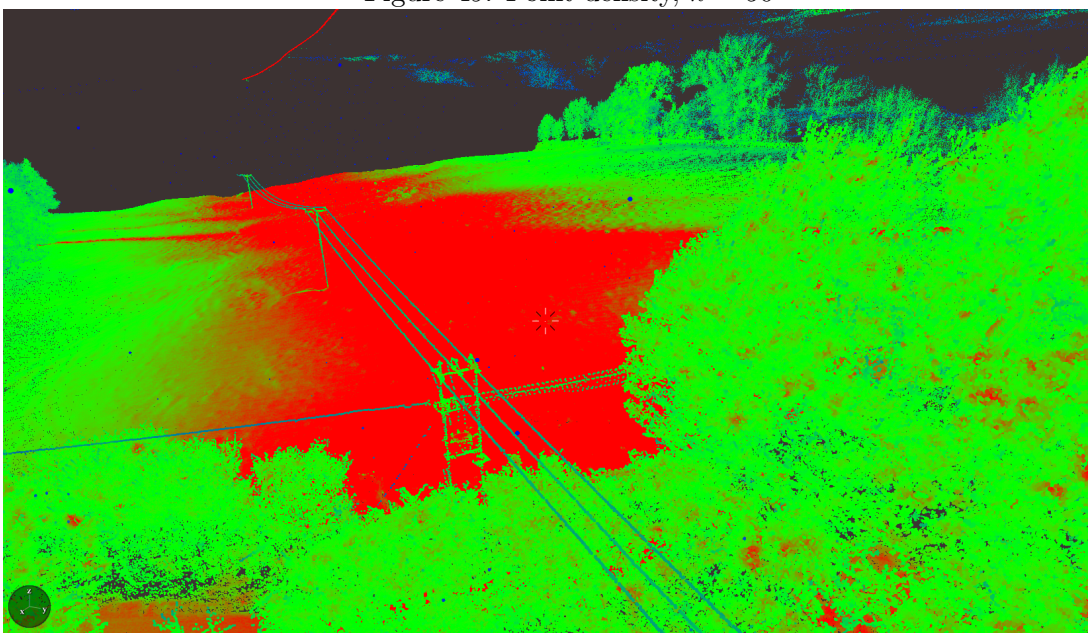
1.3.3 Time as intensity

Signal. *When working with LAS files, we can manipulate or even swap attributes. This is very simple: Here we replace intensity by GPS time in order to make a point.*

At the beginning of the last section, Addendum 1.1.3 described an issue in which points may be given more neighbours than others when the aircraft acquiring the data slows down or rotates. This was also discussed in Section 1.2.5 because of its impact on rank. We used space-time diagrams to illustrate the problem in that section, but we can also replace the intensity of points by time. Figure 53 shows a plot of time as intensity, and Figure 54 shows a view of the same plot in which we've zoomed in on the regions which caused the over-sampling problems in Section 1.2.5. The mixture of colours (=intensities) demonstrates the rotation of the aircraft.

Application domain. *Identify regions where sampling problems might be caused by changes in the attitude of the aircraft.*

Figure 49: Point density, $k = 50$

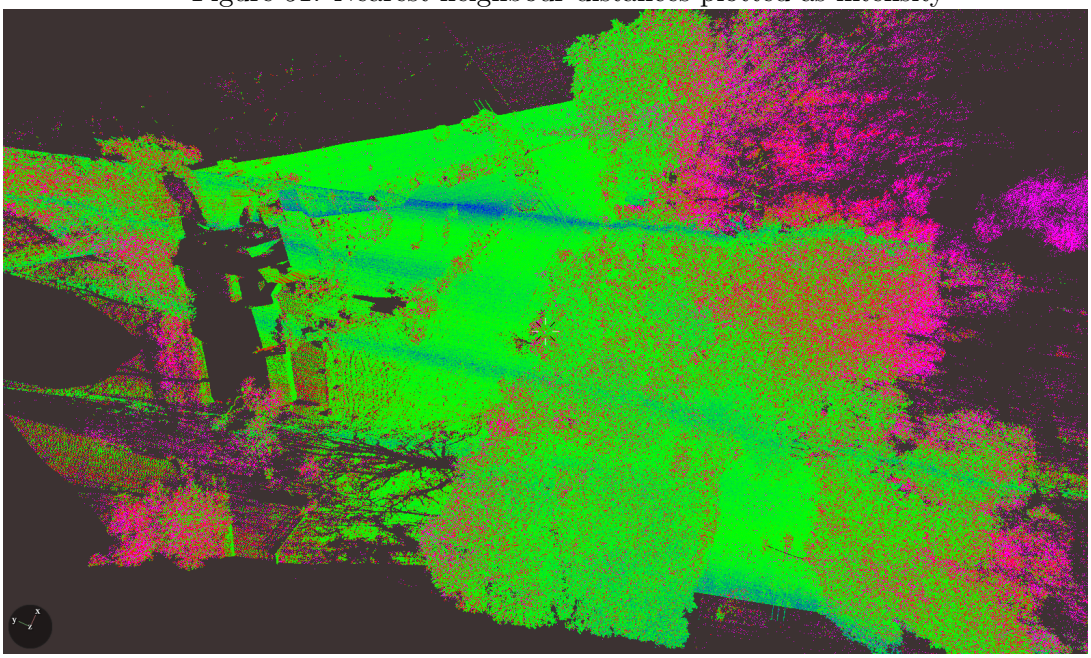


Most of the points are collected close to the flight line.

Figure 50: Point density, $k = 50$

Demonstrates the irregular sampling mentioned in Section 1.2.5.

Figure 51: Nearest neighbour distances plotted as intensity



Points are not nicely distributed around the flight line. As mentioned earlier, this is due to the trajectory of the aircraft. As expected points get further apart away from the flightline.

Figure 52: Histogram of nearest neighbour distances

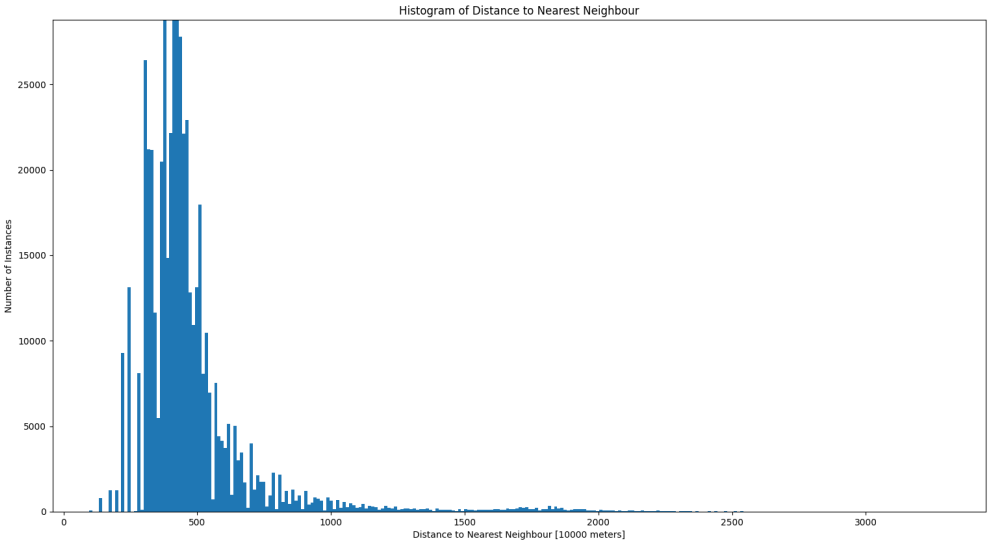
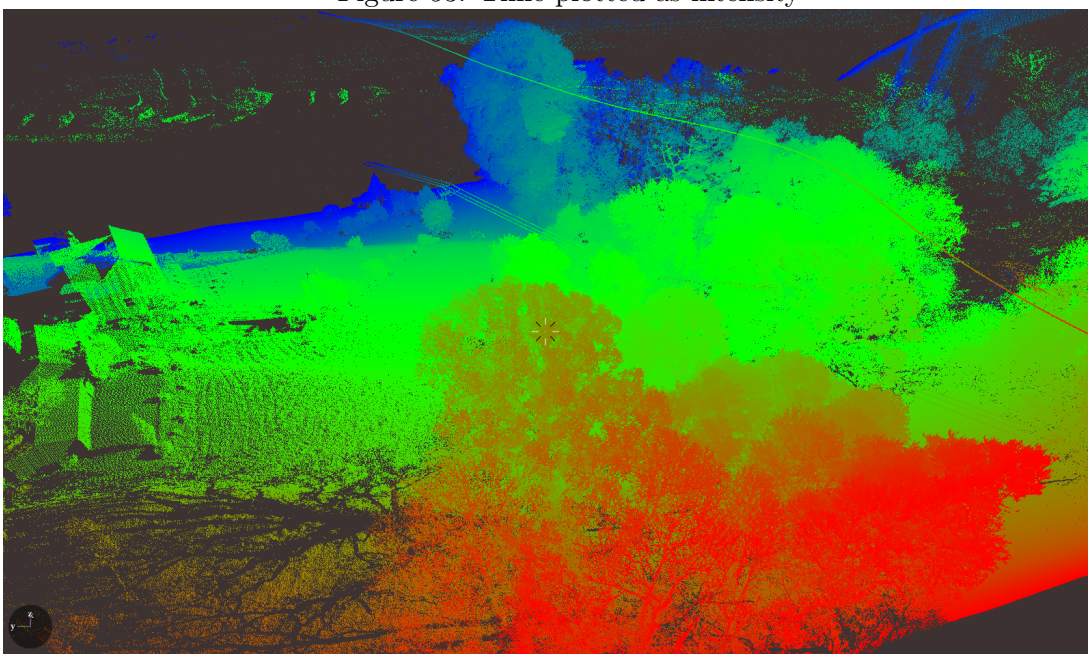
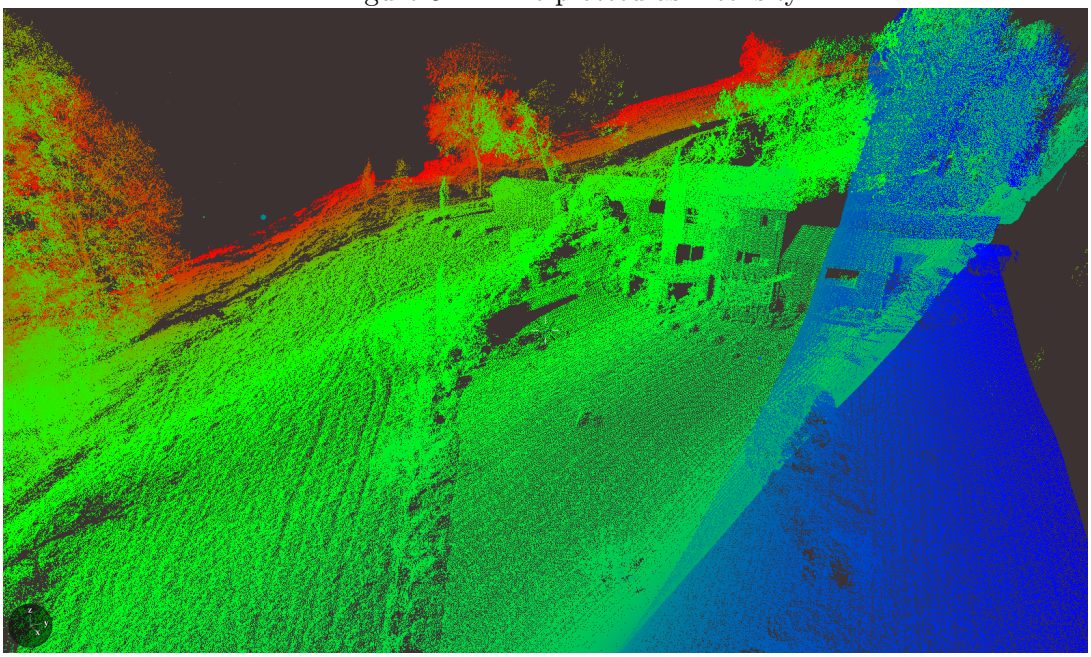


Figure 53: Time plotted as intensity



Blue-red shows progression of time.

Figure 54: Time plotted as intensity



The mixing of colours demonstrates our space-time problem caused by the aircraft's trajectory.

1.3.4 Computing in space-time

We'll now address the problems which we've been discussing in which the aircraft decides to rotate or turn around, causing oversampling at turning points, and in turn messing up our plots with small values of rank where we are not expecting them.

We will mitigate these issues by viewing our points in our point cloud as points (x, y, z, vt) rather than points (x, y, z), where v is a parameter (i.e. a constant of our choice) which we will call **virtual speed**, and t obviously stands for time. This comes at a computational cost: Storing a whole dimension's worth of information (and, worse, doing hefty calculations with it) doesn't come for free – but that's a practical detail which is easy to overcome. The effect of this transformation is two-fold: It pushes points apart which are close together in space but far apart in time, and it "unfolds" the point cloud so that dimensions of objects seem more appropriate.

This method may have an adverse affect, e.g. suppose an isolated point is sampled a few times. Because of varying time, the point would be separated out into a 1D straight line. However, remember that we use a radius ϵ to only capture points which are close enough to the target point. This radius is now a radius in space-time, so hopefully this would exclude the detrimental phenomenon of inadvertently increasing dimensions of objects too much.

Figures 55 - 60 show plots of rank at various virtual speeds. We can see that as the virtual speed increases, the suspiciously low ranks begin to increase to the value which we expect them to have.

Now let's consider eigenvalues.

Signal. *Note that, when plotting in space-time, there are actually* four *eigenvalues*

$$\mu_0 \leqslant \mu_1 \leqslant \mu_2 \leqslant \mu_3,$$

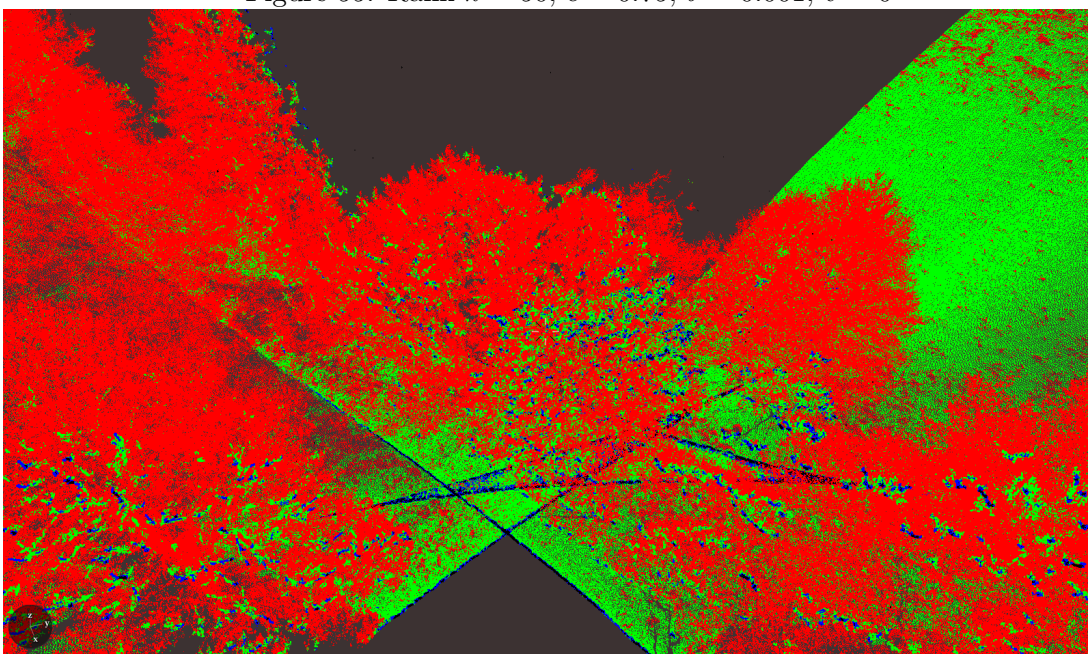
since there are four dimensions now.

Roughly speaking, we expect $\mu_0 \approx 0$ (we don't expect any 4-dimensional events, and indeed $\mu_0 = 0$ when $v = 0$) and μ_{i+1} corresponds to λ_i (the original eigenvalues in normal space) for $i = 0, 1, 2$. In fact, we now must *define* $\lambda_i = \mu_{i+1}$. Figures 61 - 70 show the plots of ratios λ_2/λ_1 and λ_1/λ_0 as we vary the virtual speed v. Note that λ_2/λ_1 still has the edge detection properties which we mentioned in Section 1.2.6.

While λ_2/λ_1 seems to be "fixed" by increasing v, λ_1/λ_0 seems to stabilise on a a less satisfactory plot as we increase v.

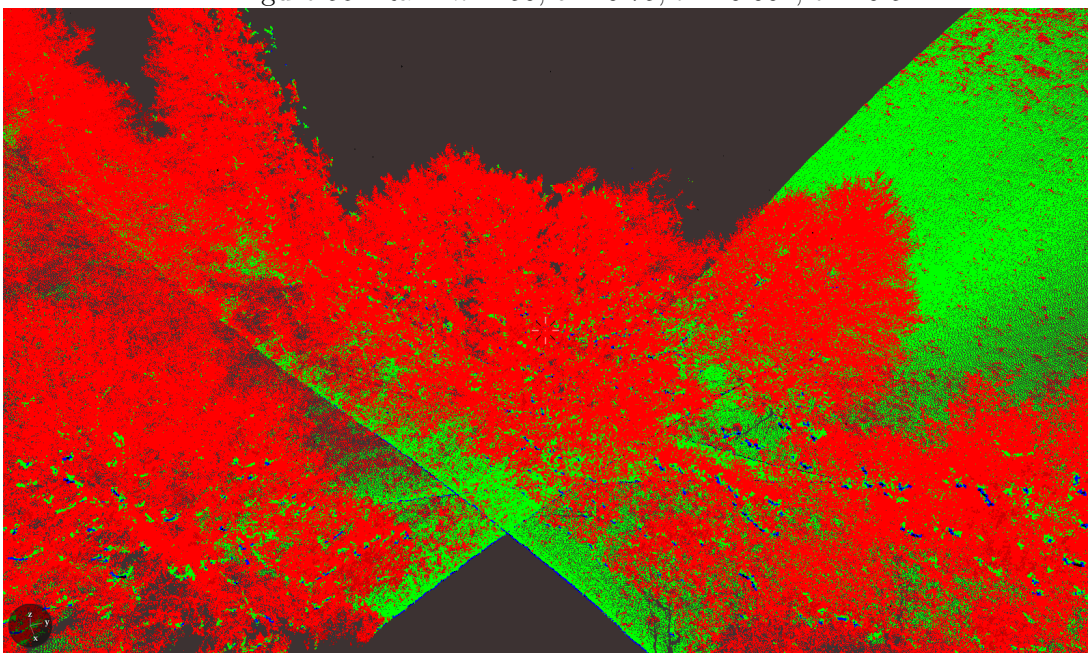
Application domain. *Mitigate anomolies in data caused by turning points in acquisition equipment. In lidar, we have anomolies caused by aircraft changing its attitude.*

Figure 55: Rank $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 0$



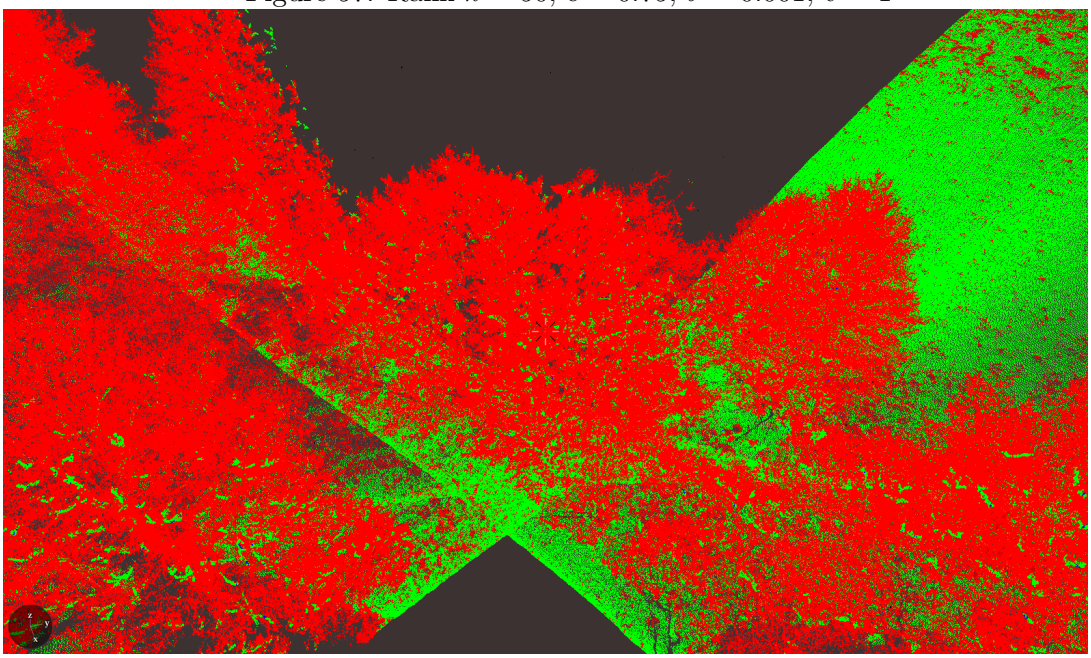
Anomolies in rank caused by violent rotations in aircraft.

Figure 56: Rank $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 0.5$



Increasing the virtual speed mitigates the anomolies.

Figure 57: Rank $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 1$



Eventually anomolies are almost completely gone, with a high enough virtual speed.

Figure 58: Rank $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 1.5$

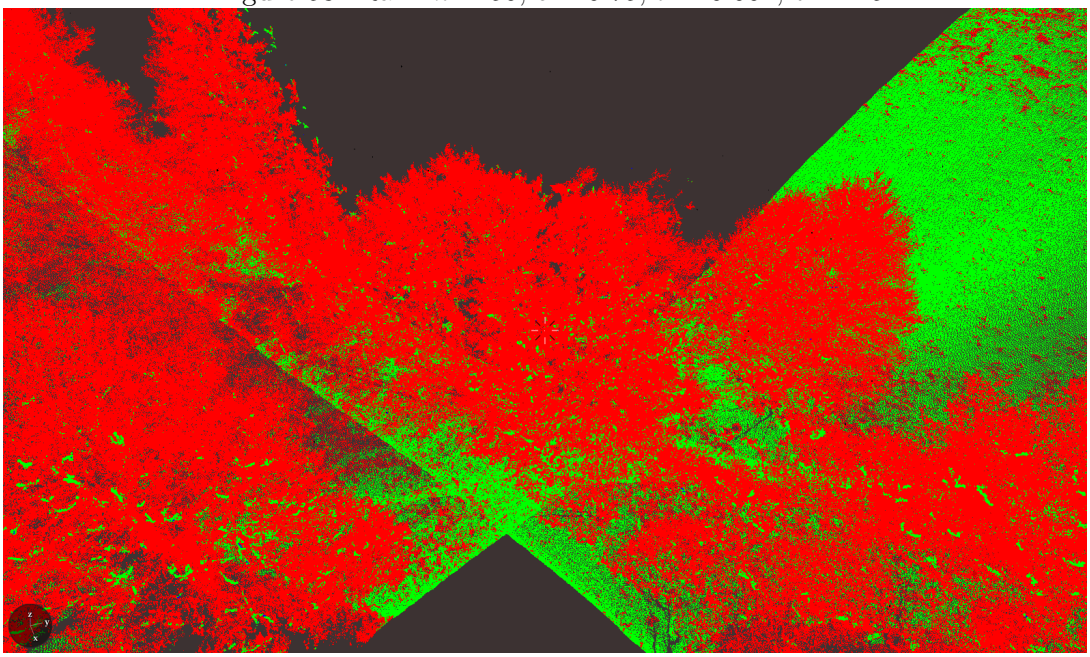
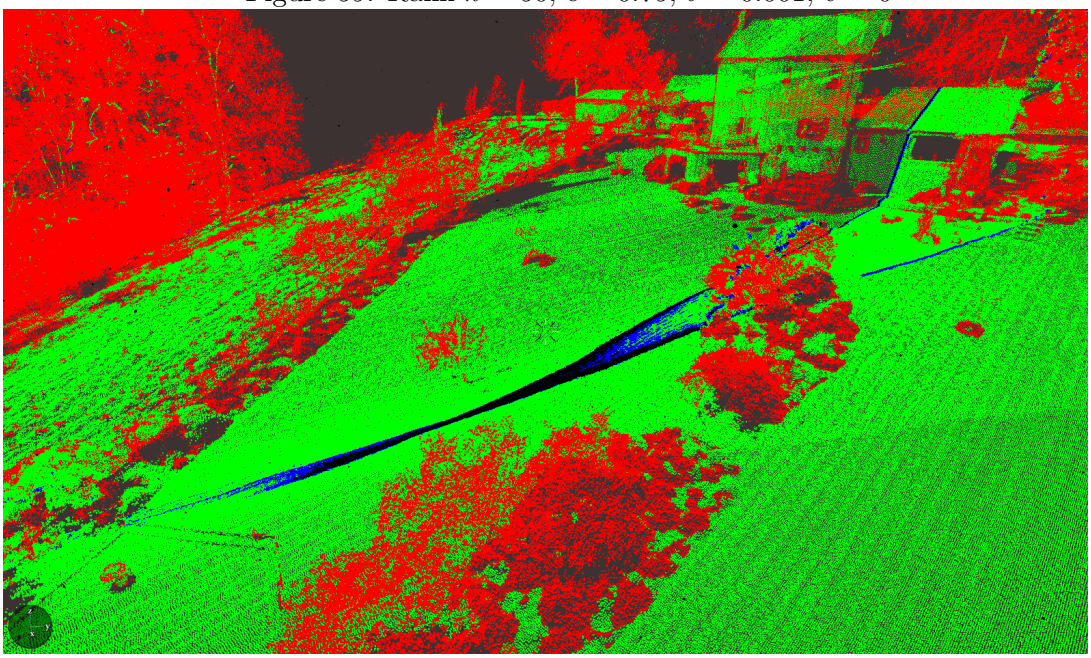
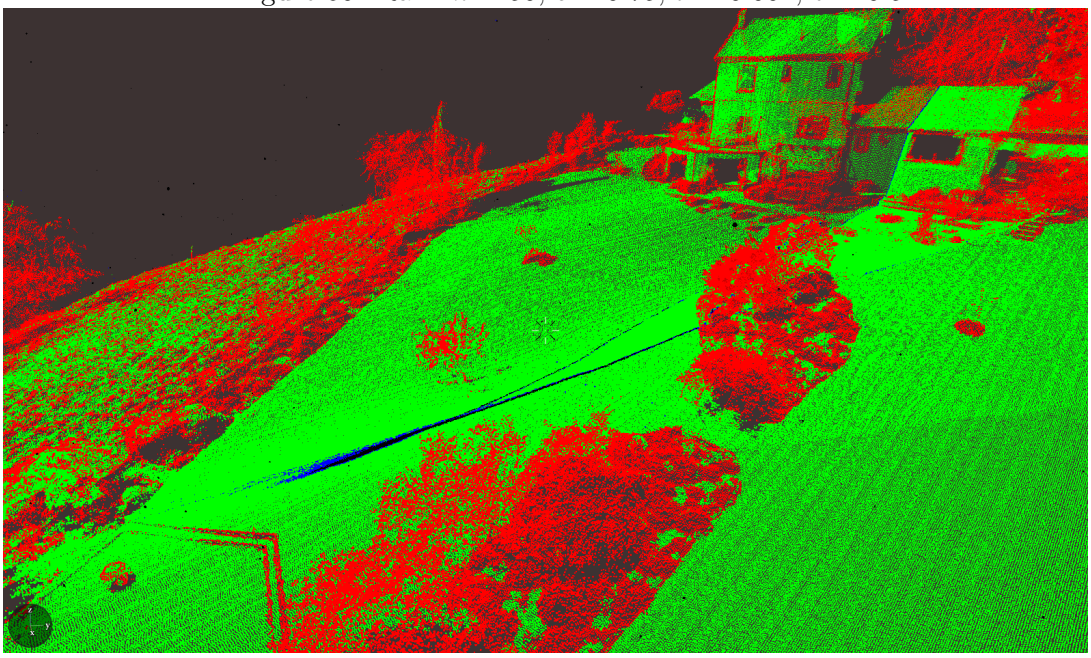


Figure 59: Rank $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 0$



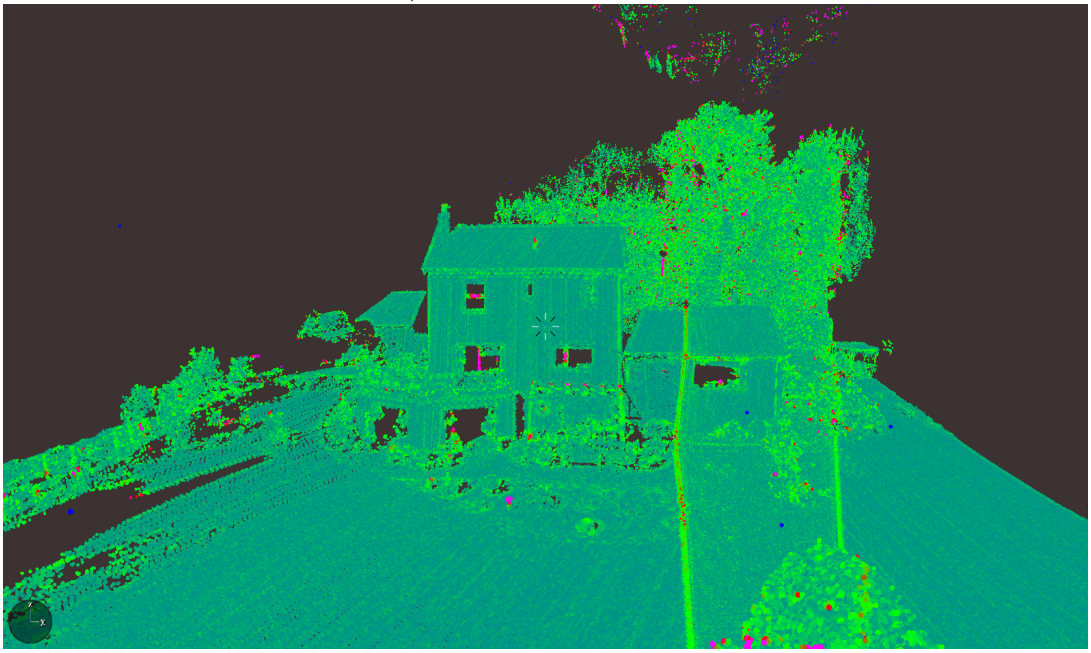
Anomolies in rank caused by quite subtle rotations in aircraft.

Figure 60: Rank $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 0.5$



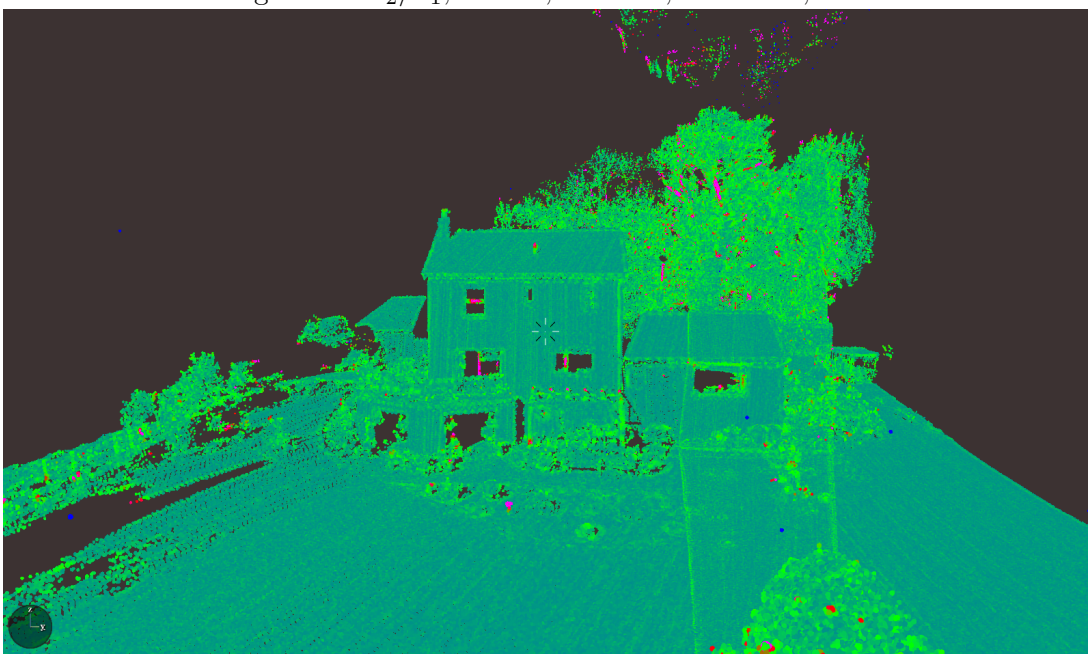
Increasing virtual speed mitigates anomalies.

Figure 61: λ_2/λ_1, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 1$



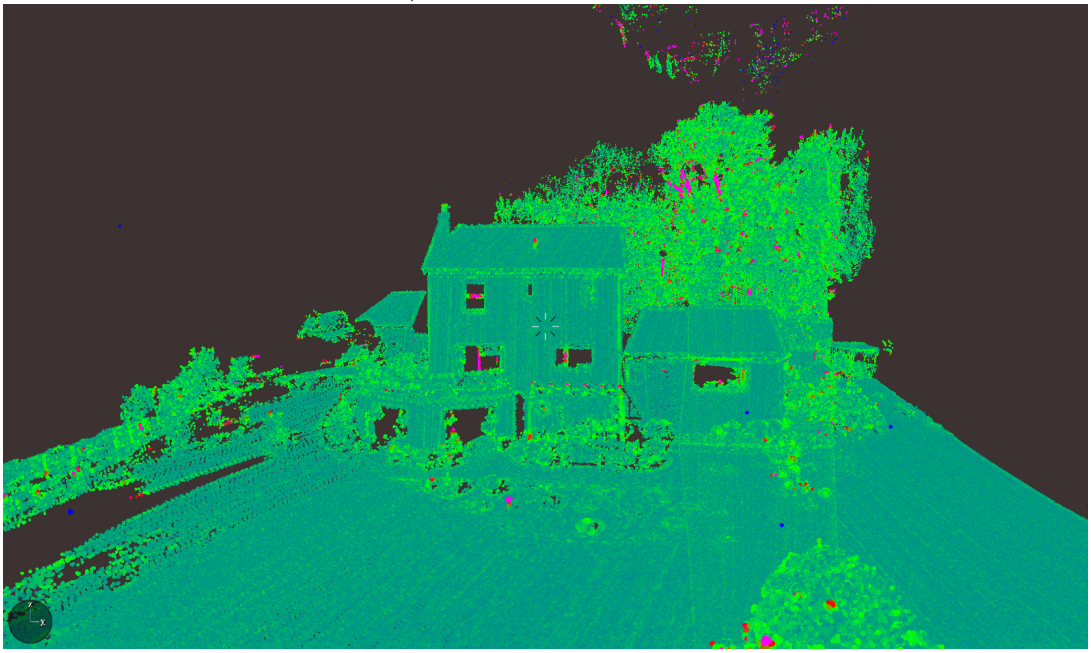
Anomolies in λ_2/λ_1 caused by quite subtle rotations in aircraft.

Figure 62: λ_2/λ_1, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 1.5$



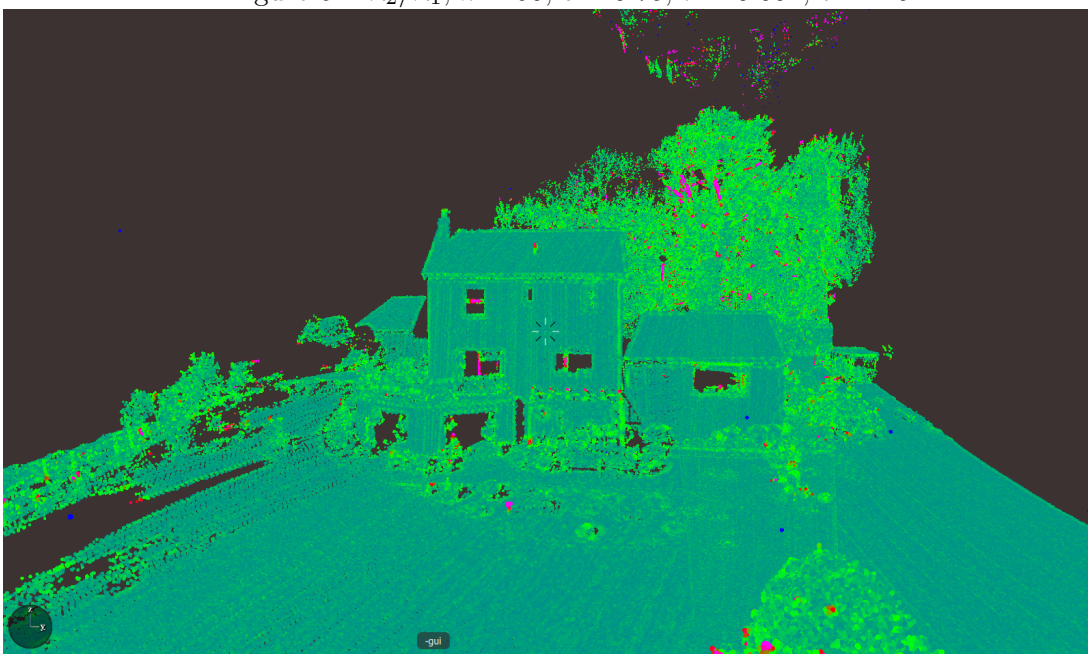
Anomolies in λ_2/λ_1 caused by quite subtle rotations in aircraft.

Figure 63: λ_2/λ_1, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 1$



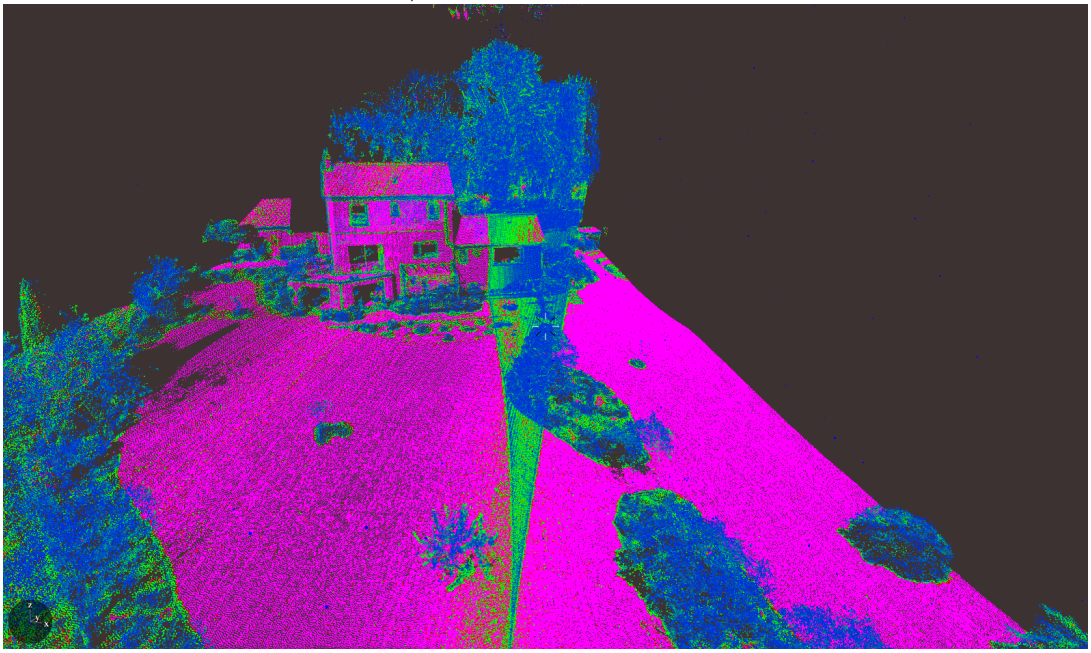
Eventually anomolies are almost completely gone, with a high enough virtual speed.

Figure 64: λ_2/λ_1, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 1.5$



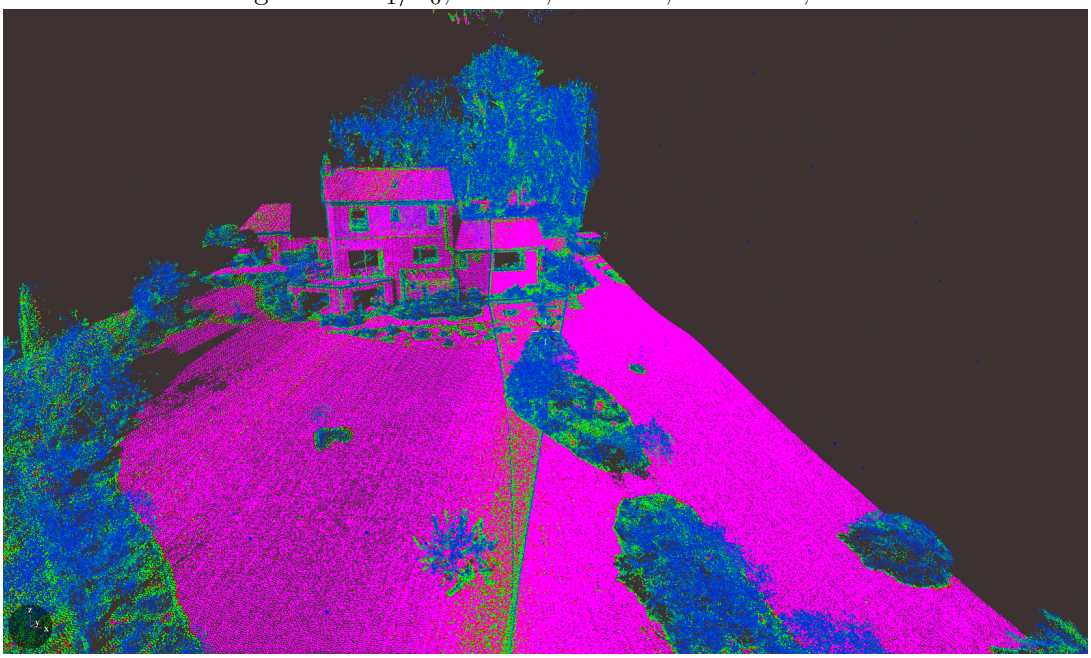
Anomolies essentially cured.

Figure 65: λ_1/λ_0, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 0$



λ_1/λ_0 very drastically impacted by space-time problems.

Figure 66: λ_1/λ_0, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 0.5$



Increasing virtual speed helps a bit.

Figure 67: λ_1/λ_0, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 1$

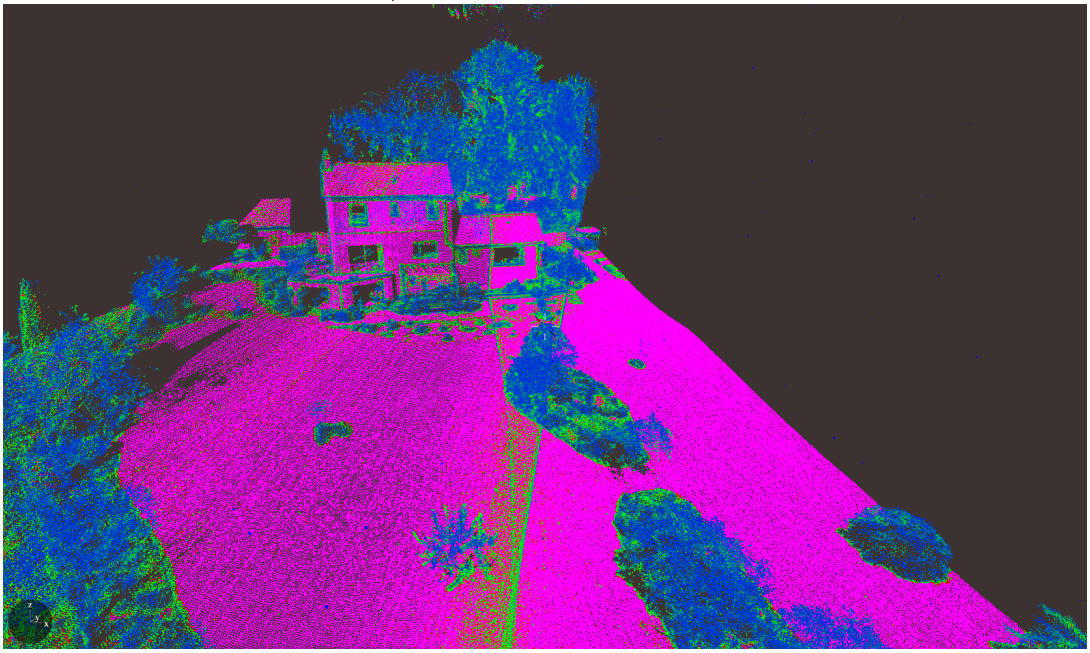
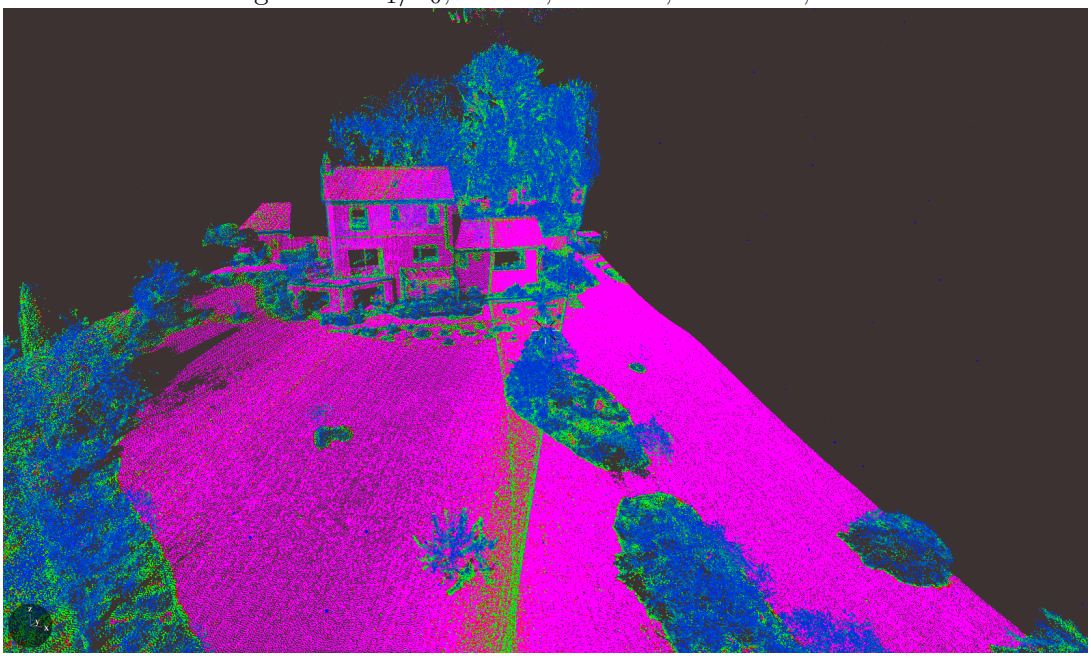


Figure 68: λ_1/λ_0, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 1.5$



Increasing virtual speed only helps so much – let's try increasing more before we give up!

Figure 69: λ_1/λ_0, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 2$

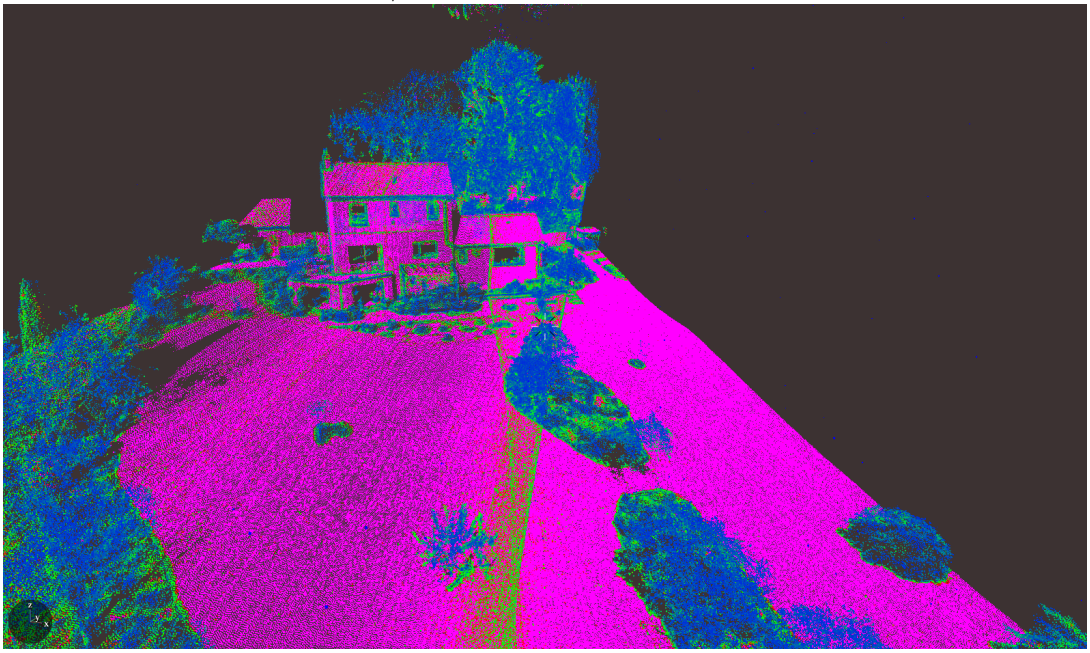
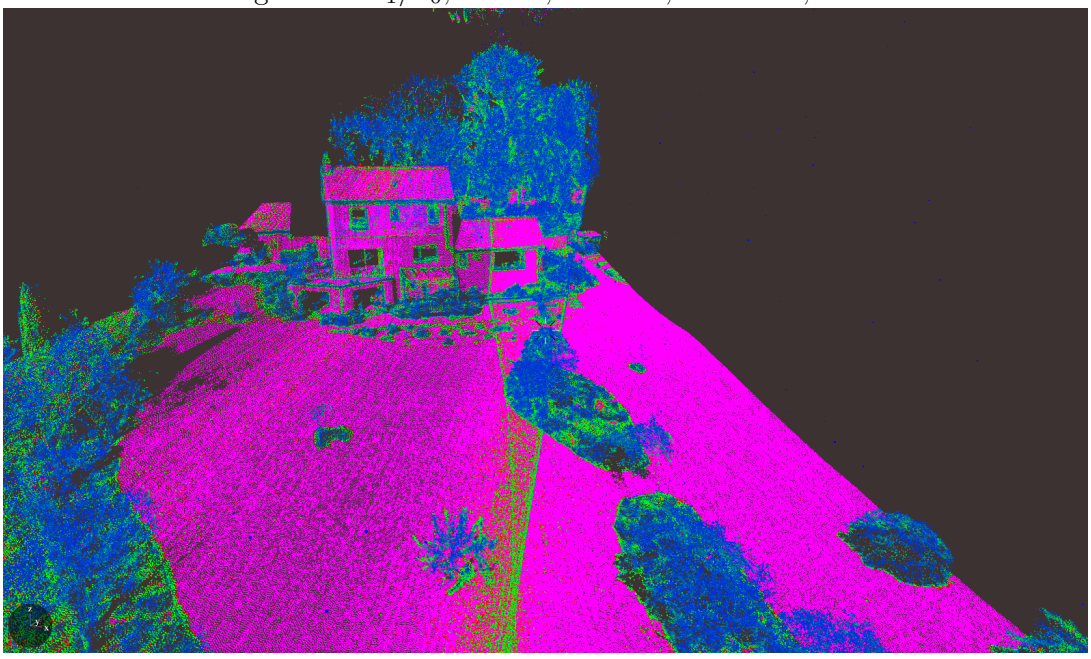


Figure 70: λ_1/λ_0, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 2.5$



Increasing virtual speed only helps so much – we'll see what else we can do in the next section.

1.3.5 Uniform decimation

We can forget enough points in order to ensure that, on average, there is only one point per u^3 cubic meters, where u is a positive constant of our choice. To do this, if x, y and z are the coordinates in our entire point cloud, we set

$$D = \text{int}\left(\frac{1}{u}\begin{pmatrix} x_0 & x_1 & x_2 & \dots & x_{n-1} \\ y_0 & x_1 & y_2 & \dots & y_{n-1} \\ z_0 & z_1 & z_2 & \dots & z_{n-1} \end{pmatrix}\right)$$

and finding the array of indices $i = (i_0, i_1, \dots, i_{m-1})$ $(m \leqslant n)$ such that the restricted matrix $D[:, 0:3]$ has unique columns. Then, we delete all points other than the points with indices in i. Obviously this method depends on the order in which points in our point cloud are generated.

If we take the resulting decimated set of points $\mathcal{D}$, and compute a signal on those points, we can then map a signal back onto the original points by taking, for each point, its image inside $\mathcal{D}$ and giving it the same value.

Signal. *We can plot the* ***impact of the decimation*** *by plotting, as a signal, the number of points which fell into each u^3 cubic meter. We can think of this signal as an improved version of the point density which we considered in Section 1.3.1.*

This is a function of points in the decimated sample, but of course we can map this back onto the original points by the process we just mentioned. A plot for the tile in Figure 71 is given in Figure 72. In that plot, over 200 points had to be removed from some of the $10 \times 10 \times 10\text{cm}^3$ regions in order to achieve the decimation. However, the mean was 1.74 and the standard deviation was 1.86. Therefore we chose to clip the plot at 10, and plot the count as intensity. Therefore, points which appear bright red in 72 are in the most densely populated regions. We note that those regions are precisely the ones we have seen in previous plots which are affected by the changing attitude of the aircraft.

The next few plots show the impact that decimation can have on some signals. Both rank and isotropy become more uniform – compare Figures 73 and 74 and Figures 75 and 76.

Recall that in Section 1.3.4 we were attempting to plot λ_1/λ_0 and found that there was a limit to how much computing our signal in space-time could help us. We find that decimating before computing in space-time is quite beneficial. To see this, compare Figures 77 and 78, which show λ_1/λ_0 computed in space time with $v = 2$, both before and after decimation.

Application domain. *Improve quality of signals by reducing over-sampling. Also because computations are done on a smaller set of points, decimation causes us to use less memory when plotting signals.*

Figure 71: A test tile

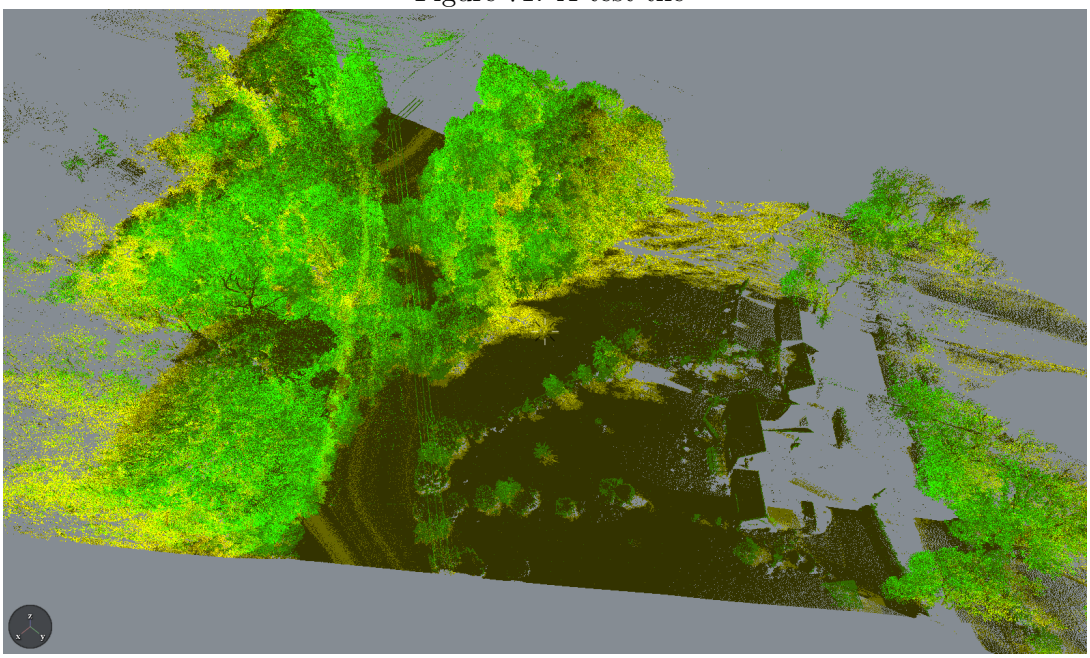
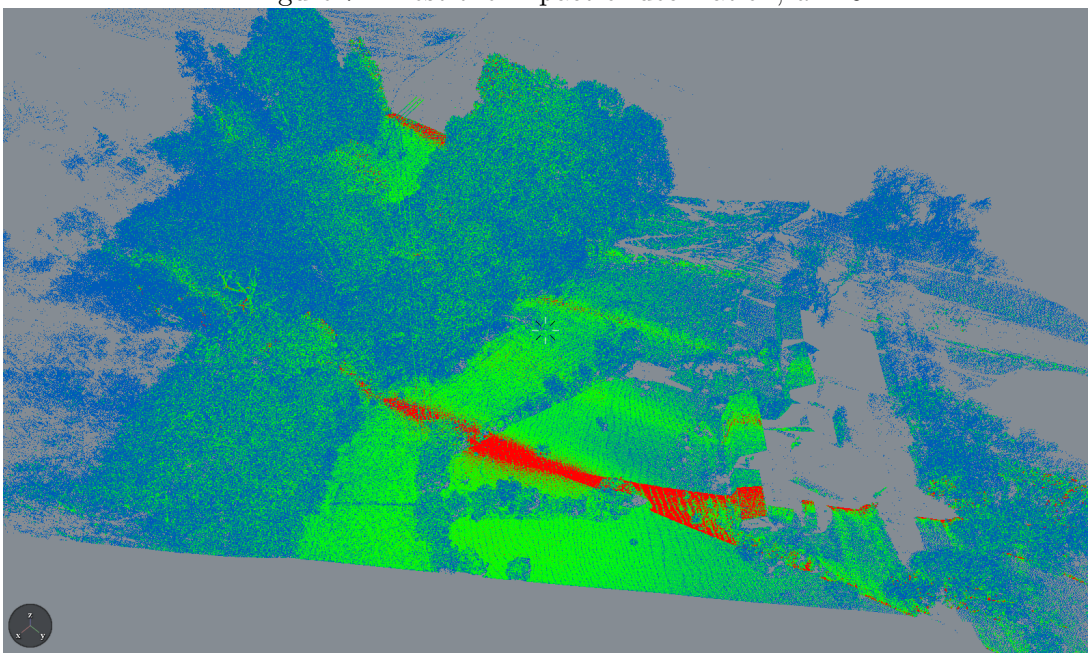
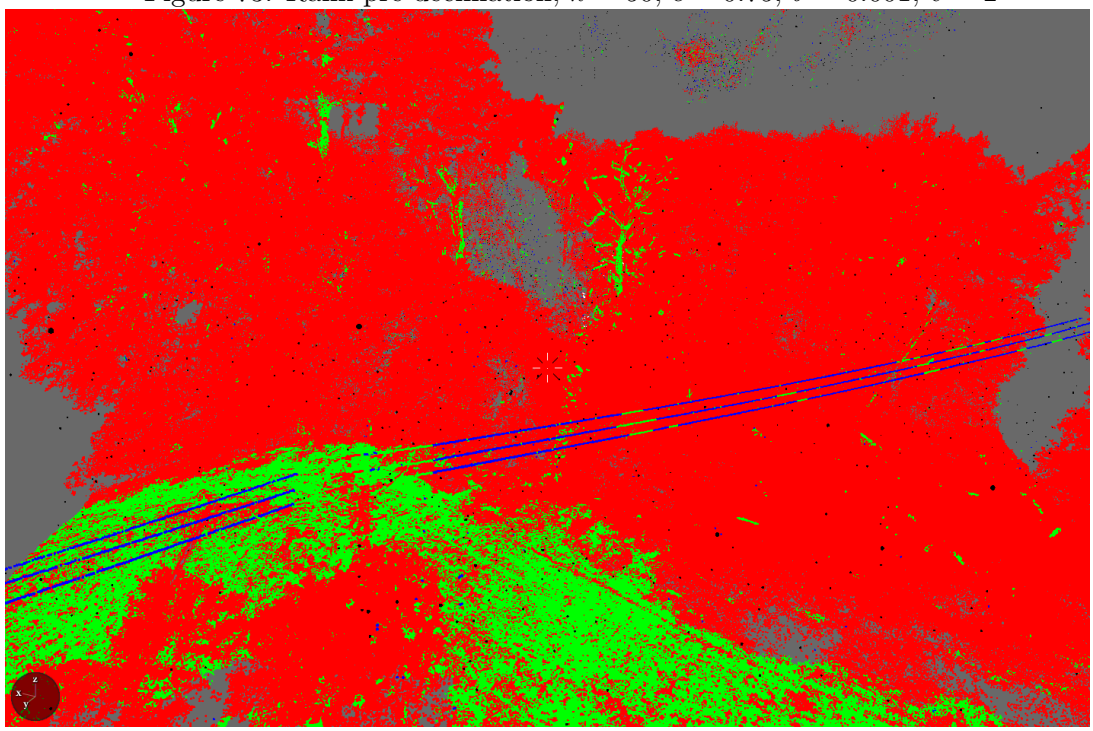


Figure 72: Test tile impact of decimation, $u = 0.1$



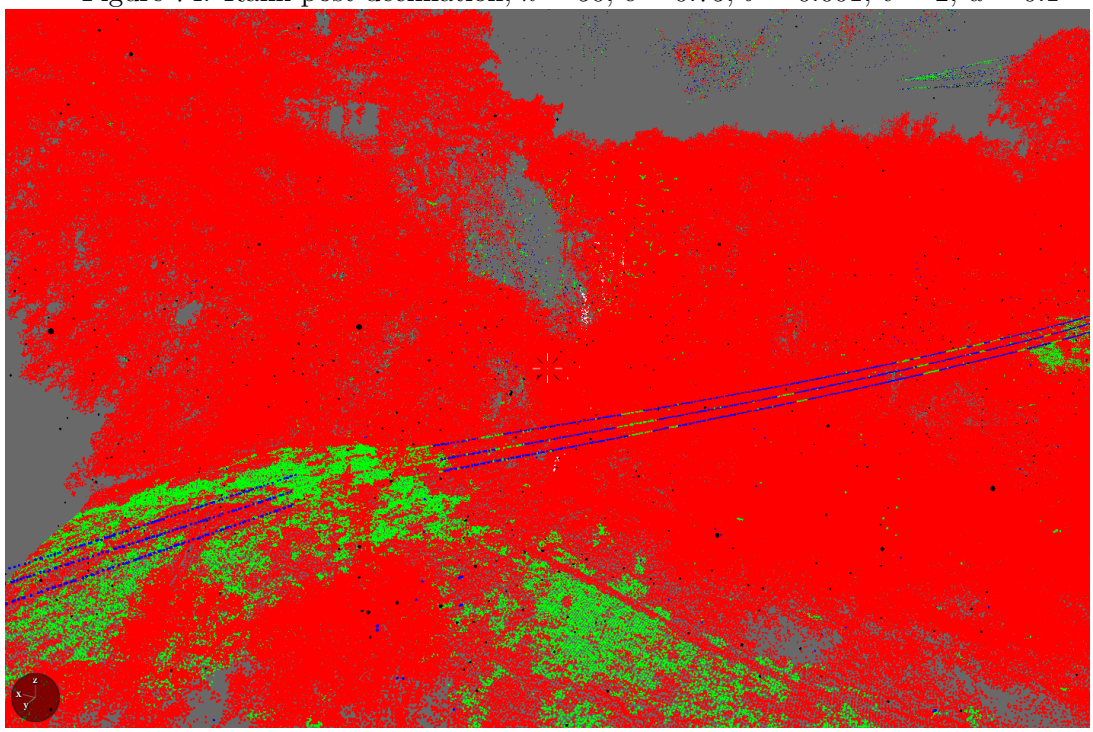
The red region is precisely the one which has caused us problems in previous plots.

Figure 73: Rank pre-decimation, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 2$



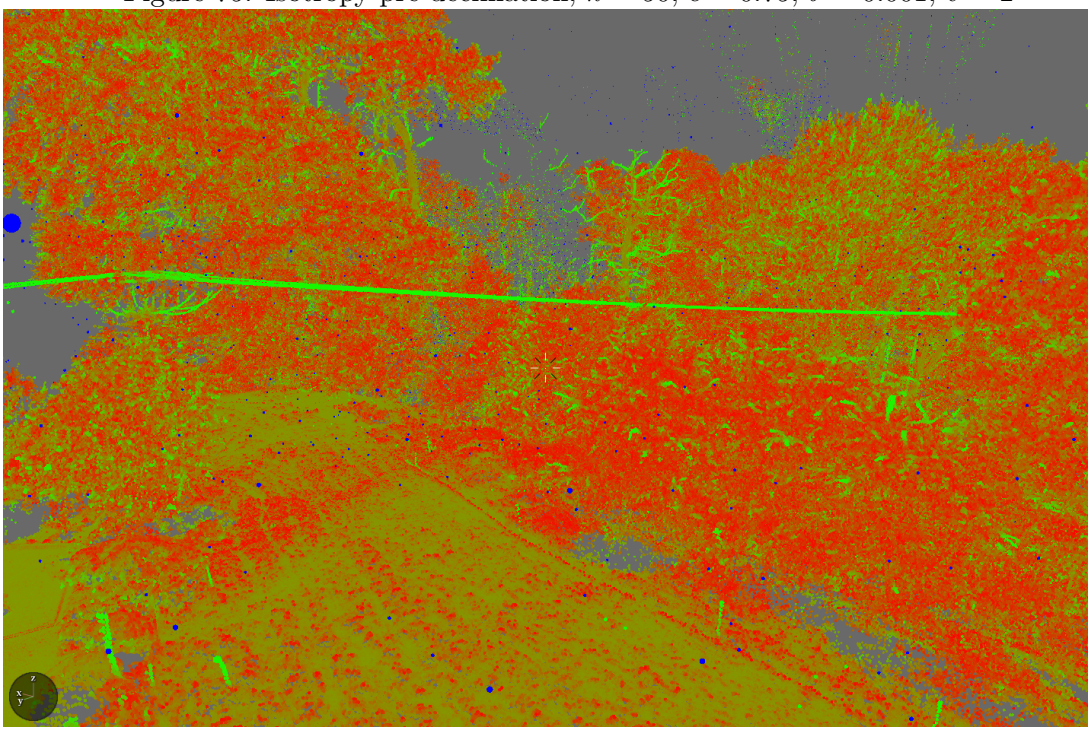
Recall that rank should represent dimension. 3D = red, 2D = green, 1D = blue, 0D = black.

Figure 74: Rank post-decimation, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 2$, $u = 0.1$



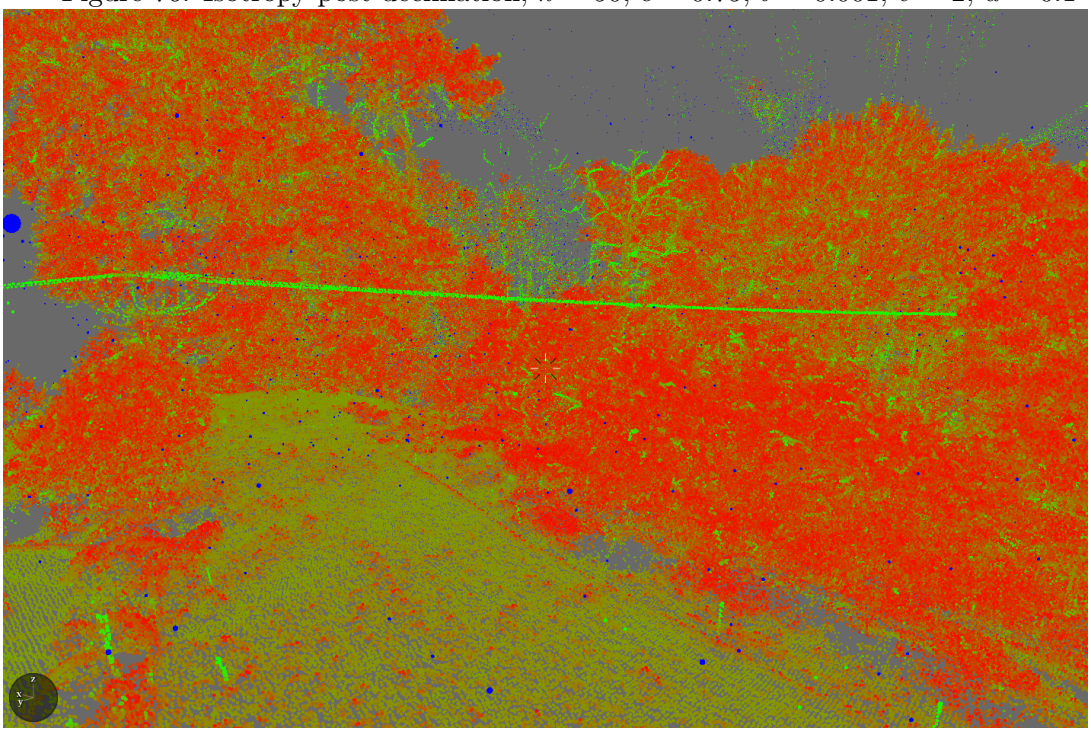
Decimation improves consistency.

Figure 75: Isotropy pre-decimation, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 2$



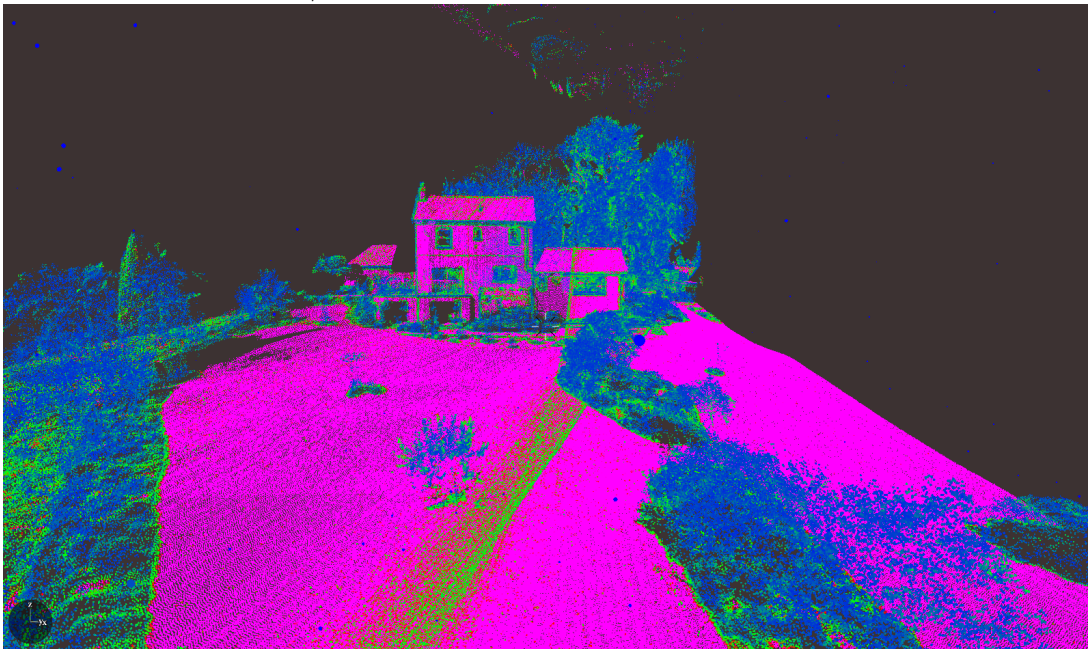
Isotropy shows how spread out an object is in all directions.

Figure 76: Isotropy post-decimation, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 2$, $u = 0.1$



Decimation makes signal more vivid.

Figure 77: λ_1/λ_0 pre-decimation, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 2$



Recall that we couldn't fix this signal with virtual speed.

Figure 78: λ_1/λ_0 post-decimation, $k = 50$, $\epsilon = 0.75$, $\theta = 0.001$, $v = 2$, $u = 0.1$

Decimation helps a lot.

1.3.6 Clustering

1.4 Shannon entropy

Whenever we record the results of an experiment, we gain some information about the state of the system being measured. The problem which entropy solves is the following:

When recording the result of an experiment, what is the average amount of information gained?

To get some intuition, suppose that we have a d-sided dice, for $d \geqslant k$, and suppose that we colour each face with k colours $C_0, C_1, \ldots, C_{k-1}$. The probability $p_i = \Pr(C_i)$ of landing on a face with colour C_i when we roll the dice is given by $p_i = m_i/d$ where m_i is the number of faces with colour C_i.

How can we record the result of rolling this dice? We could, of course, simply record the face $i \in \{0, 1, 2, \ldots, k-1\}$ which the die lands on, which would also allow us to determine the colour C_i which it lands on. This would take up $\log_2(d)$ bits of memory. If we only record the colour C_i, and we do not record the face number, then we written down $\log_2(m_i)$ bits of information, since there are m_i faces with the colour C_i. Since we need to write down $\log_2(d)$ bits of information to fully understand the state of the system, and we will have written down $\log_2(m_i)$ by observing the colour C_i, the information gained in doing so is

$$\log_2(d) - \log_2(m_i) = -\log_2(p_i).$$

Therefore, the average information gained from an experiment is

$$E(p_0, p_1, \ldots, p_{k-1}) = -\sum_{i=0}^{k-1} p_i \log_2(p_i).$$

This quantity is known as **shannon entropy**. Maximum entropy occurs when we have "equilibrium" in the sense that $p_0 = p_1 \cdots = p_{k-1} = 1/k$. If any of the probabilities p_i are zero then they do not contribute to the sum (this is justified since $\lim_{p \to 0} p \log_2(p) = 0$). Indeed, with k states, we could use $\log_k$ instead of $\log_2$, and therefore the maximum entropy would be $E = 1$, which would be shared when each of the k states have an equal probability. The only way to obtain zero entropy would be when $p_i = 1$ for some i and all the other probabilities are zero. For this reason, entropy is often viewed as a measure of **disorder** – i.e. how spread out a system is between different states. We'll consider some entropies which can be associated with point clouds.

1.4.1 Eigenentropy

We will now define three signals which give the local probability of a point being isolated, part of a line, part of a plane, or part of a voluminous scattering of points. Suppose we have our eigenvalues $\lambda_2 \geqslant \lambda_1 \geqslant \lambda_0$.

- The **linearity**
$$p_2 = \frac{\lambda_2 - \lambda_1}{\lambda_2}.$$
- The **planarity**
$$p_1 = \frac{\lambda_1 - \lambda_0}{\lambda_2}.$$

- The **scattering**

$$p_0 = \frac{\lambda_0}{\lambda_2}.$$

In the event that $\lambda_2 = 0$, these probabilities are meaningless and we can disregard these probabilities and set the entropy to be $E = 0$.

Signal. *The entropy*

$$-p_0 \log_3(p_0) - p_1 \log_3(p_1) - p_2 \log_3(p_2)$$

is called ***eigenentropy****.*

Since we are measuring the entropy contributed by three states (except when $\lambda_2 = 0$) we instead take $\log_3$ instead of $\log_2$, because then the maximum value of entropy becomes $3 \cdot -\frac{1}{3}\log_3(\frac{1}{3}) = 1$.

Figures 79 and 80 show a plot of eigenentropy. Objects which look most like one type of dimension – point-like, linear, planar, voluminous – should have low entropy. But, of course, an object cannot be linear without being planar, and an object cannot be planar without being slightly voluminous, so we should expect to see entropy rise similarly to rank. That is, voluminous objects should have higher entropy than planar objects, and similarly planar objects should have higher entropy than linear objects, and linear objects should have higher entropy than point-like objects. Entropy, therefore, should be seen as a continuous form of rank.

1.4.2 Curvature entropy

The probabilities used in eigenentropy are a parameter in themselves: We can actually make up new probabilities and compute the entropy based on those. We'll look at a similar set of probabilities which give a slightly different entropy. In this case, the probabilities tell us to which degree a point is spread in the directions of its free eigenvectors. These probabilities can be considered to be measurements of curvature (in a particular direction) – hence the name.

Signal. *The probabilities we will use are given by*

$$p_0 = \frac{\lambda_0}{\lambda_0 + \lambda_1 + \lambda_2}$$

$$p_1 = \frac{\lambda_1}{\lambda_0 + \lambda_1 + \lambda_2}$$

$$p_2 = \frac{\lambda_2}{\lambda_0 + \lambda_1 + \lambda_2}$$

We therefore have ***curvature entropy****,*

$$E = -p_0 \log_3(p_0) - p_1 \log_3(p_1) - p_2 \log_3(p_2).$$

Curvature entropy seems similar to eigenentropy – see Figures 81 and 82. However, zooming in, we see that curvature entropy is much more stable – compare Figures 83 and 84.

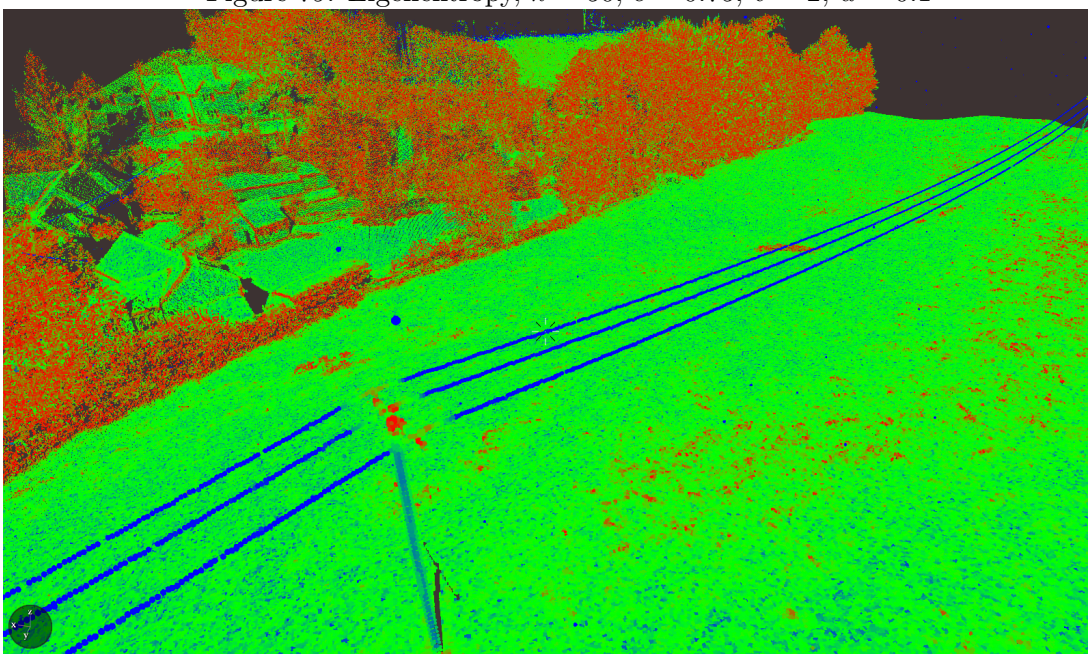
If we assume that $p_0 = 0$, then $p_1 + p_2 = 1$ and the entropy $E(p_0, p_1, p_2)$ becomes $E(p_1, p_2)$, and it's new maximum value is $\log_3(2) \approx 61.3\%$. But note if $p_0 = 0$ and the curvature entropy is at this maximum value, i.e. $\lambda_1 = \lambda_2$ then in this situation, eigenentropy is quite different: The planarity statistic is

$$\frac{\lambda_1 - \lambda_0}{\lambda_2} = 1$$

and therefore eigenentropy is $E = 0$! So, in fact, despite looking globally quite similar when "zoomed out", eigenentropy and curvature entropy can behave in drastically different ways. This is why we see the differences between Figures 84 and 83.

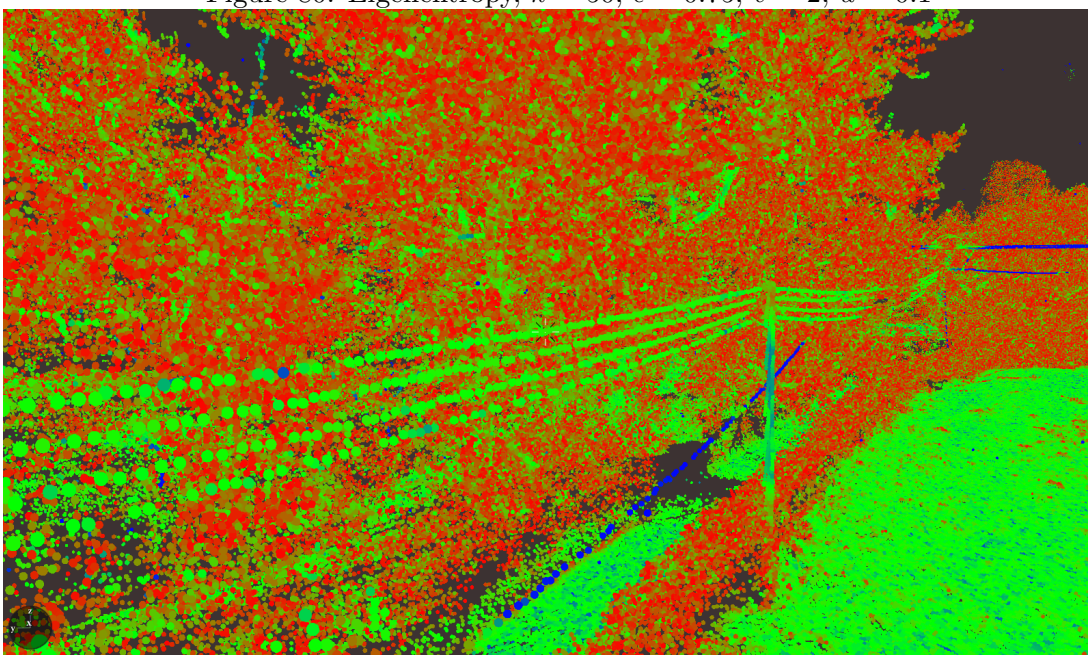
Application domain. *Entropy gives a continuous (and therefore more forgiving) idea of the "dimension" of an object. Certain object types inhabit a narrow range of entropies, so it can be used to enhance a search for those objects. For example, an evenly-sampled planar surface would have entropies in the region of 0.613, as mentioned above.*

Figure 79: Eigenentropy, $k = 50$, $\epsilon = 0.75$, $v = 2$, $u = 0.1$



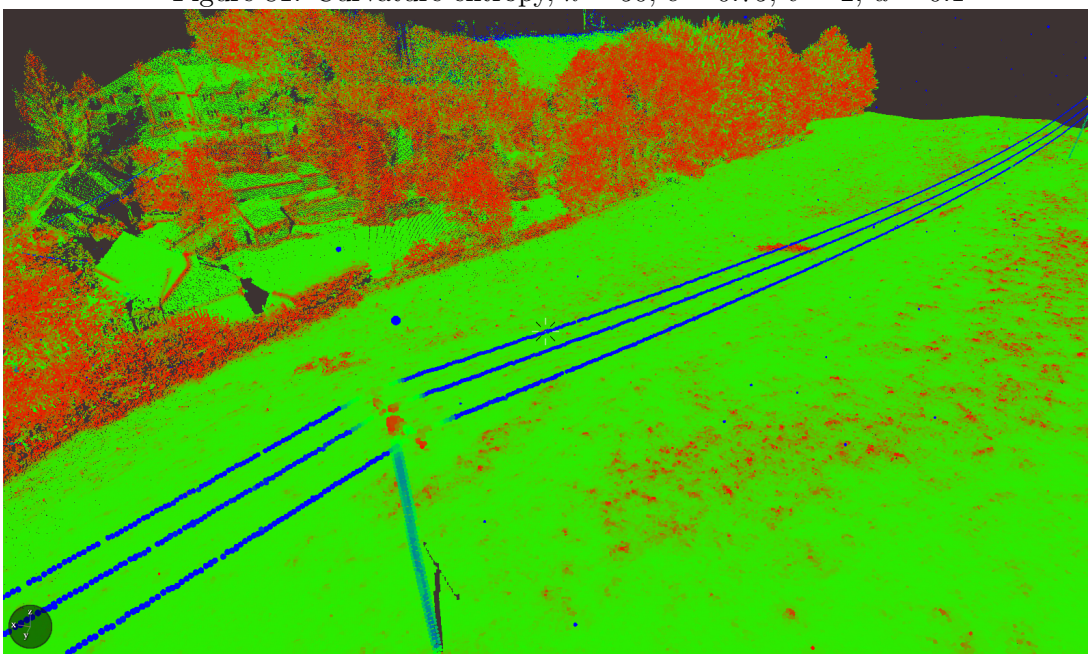
Entropy offers a continous version of dimension. 0D and 1D objects (e.g. noise and conductors) have lowest values, rising up to 3D values with highest values (e.g. trees and corners of buidlings)

Figure 80: Eigenentropy, $k = 50$, $\epsilon = 0.75$, $v = 2$, $u = 0.1$



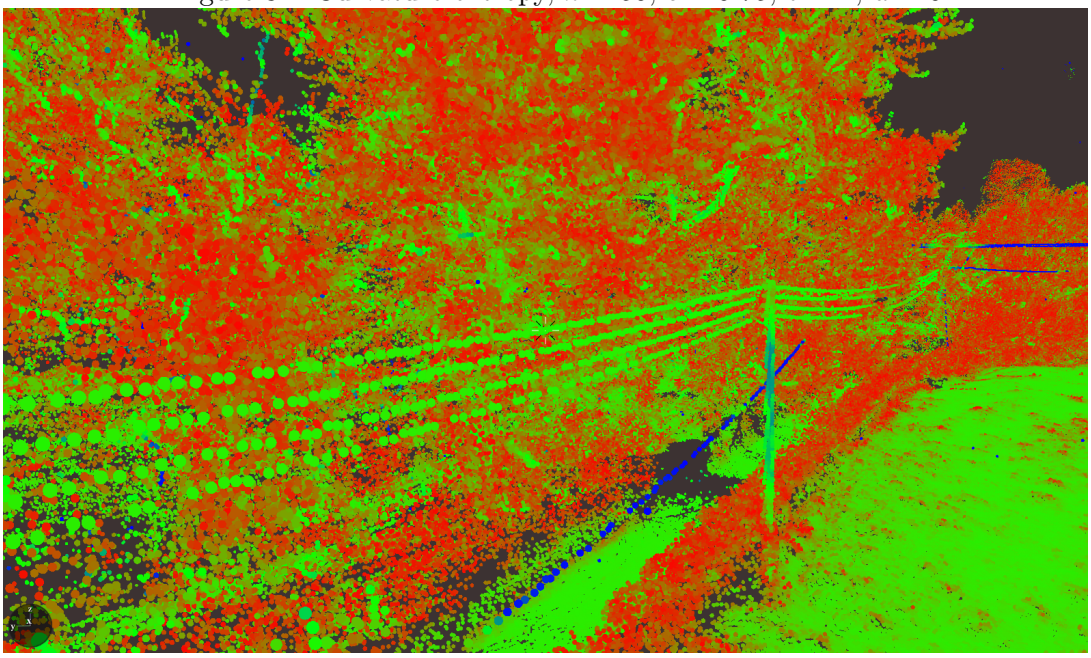
This conductor is of a different type and appears more 2D – as shown by its higher entropy.

Figure 81: Curvature entropy, $k = 50$, $\epsilon = 0.75$, $v = 2$, $u = 0.1$



Our curvature entropy is much more consistent It plays a similar role to eigenentropy but doesn't have the problem in which it takes lower values the flatter an object becomes...

Figure 82: Curvature entropy, $k = 50$, $\epsilon = 0.75$, $v = 2$, $u = 0.1$



Note the lack of blue dots on this conductor – a 2D object. Eigenentropy gave some lower values.

Figure 83: Eigenentropy, $k = 50$, $\epsilon = 0.75$, $v = 2$, $u = 0.1$

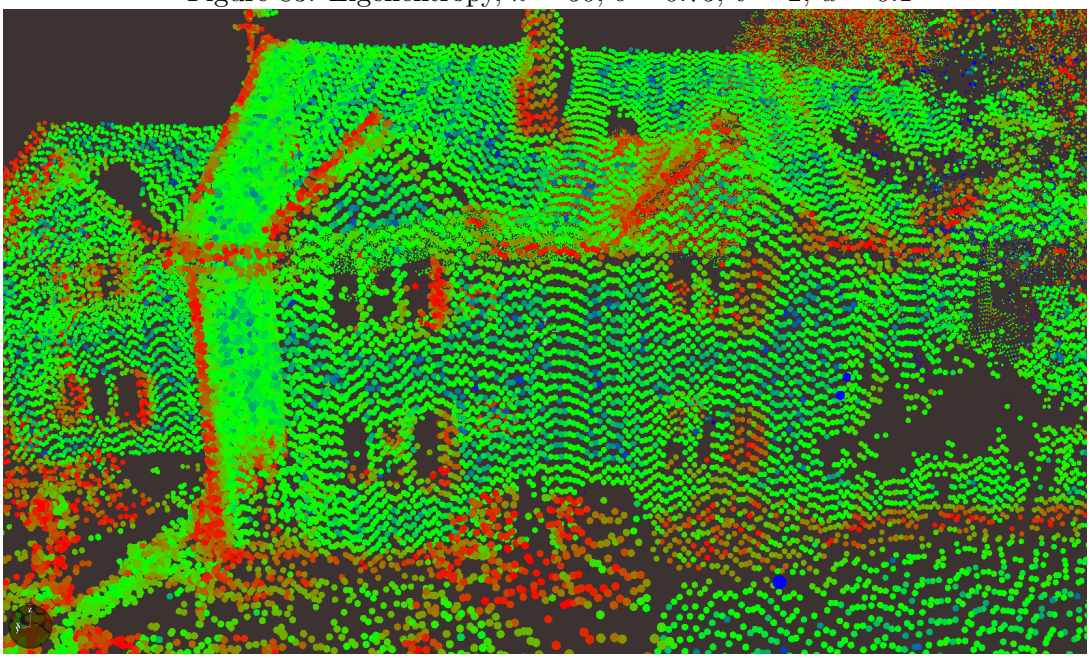
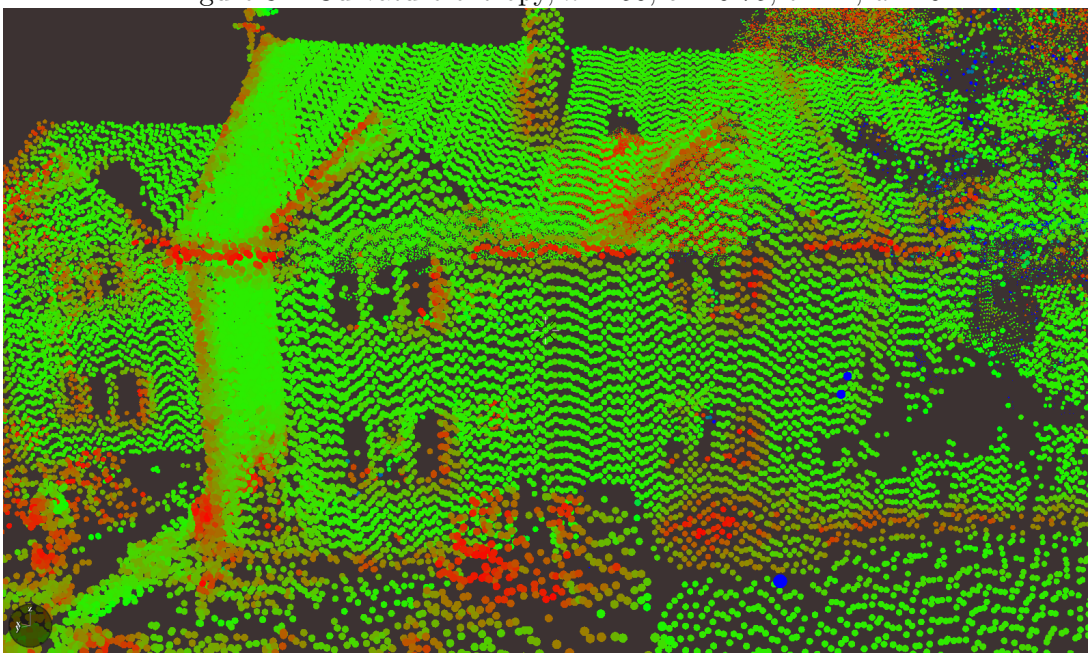


Figure 84: Curvature entropy, $k = 50$, $\epsilon = 0.75$, $v = 2$, $u = 0.1$



Direct comparison of eigenentropy and curvature entropy – note that there are some areas where the value of eigenentropy sinks and our curvature entropy remains consistent.

1.5 Hough space

In this section we'll give an explanation of the concept of Hough spaces. Most commonly Hough spaces are used for extracting straight lines, but they may Hough space is $H = [0, \pi) \times \mathbb{R}$. The set of all lines in $\mathbb{R}^2$ is in bijection with H, with any point (r, θ) corresponding to the line which is defined by $x \cos\theta + y \cos\theta = r$, whose normal is $(\cos\theta, \sin\theta)$ and whose distance to the origin is $|r|$.

Any point $\mathbf{p} = (a, b)$ in $\mathbb{R}^2$ is gives a curve $C(\mathbf{p}) \subseteq H$ which is given by

$$a \cos\theta + b \sin\theta = r.$$

It is easy to see that the points $\mathbf{p}_1, \mathbf{p}_2, \ldots, \mathbf{p}_n$ are colinear if and only if $\bigcap_{i=1}^{n} C(\mathbf{p}_i) \neq \varnothing$,

Figure 85: The point $\mathbf{p} = (1, 1)$ is represented in Hough space by the curve $r = \sqrt{2}\sin(\theta + \pi/4)$

i.e. if the curves $C(\mathbf{p}_i)$ intersect. Thus the problem of determining colinearity is equivalent to determining the concurrence of this kind of curve in H. That is, points lying on the same curve in H corresponds to lines going through the same point.

Consider the points $(1, 0)$, $(2, 1)$, $(3, 2)$ and $(4, 3)$. We can see that they lie on the straight line $y = x - 1$. The intersection of the curves in Figure 86 illustrates this. They intersect at $(3\pi/4, -1/\sqrt{2})$, which is the point corresponding to the line

$$x \cos\frac{3\pi}{4} + y \sin\frac{3\pi}{4} = -\frac{1}{\sqrt{2}},$$

i.e. $-x + y = -1$, as expected.

Suppose we were to change the point $(1, 0)$ to $(1, 1)$, so that instead we look at the points $(1, 1)$, $(2, 1)$, $(3, 2)$ and $(4, 3)$ these are not colinear but three of them are. This would cause one of the curves in Figure 86 to change so that it no longer meets the other three simultaneously. See Figure 87 for the graph.

This is the standard hough transformation: Lines are sent to points and points are sent to curves. However, this is not the end of the story. One can concievably carry out this process for any parametrised family of curves, and this process works esspecially well if we are working with curves whose equations exhibit some symmetry between the parameters and the variables. For example, consider the equation

$$(x - a)^2 + (y - b)^2 = 1$$

for a circle with radius 1 and center (a, b). If we were to exchange the roles of (x, y) and (a, b), the equation would be unchanged. This is the symmetry to which we refer, but it is not essential – it just makes the Hough transform prettier. Consider $P = \mathbb{R}^2$, which we will call the **picture space** ($\mathbb{R}^2$ was the picture space earlier). The **circle Hough space** is also $H = \mathbb{R}^2$. A circle in the picture space corresponds to its center in hough space, i.e. circles correspond to points

in Hough space. However, a point in the picture space is also a circle in circle Hough space: $(x, y) \in P$ corresponds to the circle

$$C(x, y) = \{(a, b) : (x - a)^2 + (y - b)^2 = 1\}$$

in Hough space.

Let points $(1, 0)$, $(1/\sqrt{2}, 1/\sqrt{2})$ and $(\sqrt{3}/2, -1/2)$ be given. The circles which correspond to these points are pictured in Figure 88. The intersection of these circles at the origin represents the fact that these points all lie on the unit circle centered at the origin.

We could also have represented circles as points in the Hough space $H = \mathbb{R}^2 \times (0, \infty)$, where a point (a, b, R) in this picture space would represent a cirlce with center (a, b) and radius $R > 0$. We could then send a point $(x, y) \in \mathbb{R}^2$ to the cones

$$\{(a, b, R) : (x - a)^2 + (y - b)^2 = R^2, R \geqslant 0\}.$$

Given two points (x_1, y_1) and (x_2, y_2), with corresponding cone S_1 and S_2, the intersection $S_1 \cap S_2$ represents the infinite family of circles which contain both of those points. However, given any three points (x_1, y_1), (x_2, y_2) and (x_3, y_3), with corresponding surfaces S_1, S_2, S_3, a unique circle contains those points, and so $S_1 \cap S_2 \cap S_3$ contains just one point. In Figure 89 the points $(2, 0)$, $(\sqrt{2}, \sqrt{2})$ and $(\sqrt{3}, -1)$ are represented in this picture space as cones. The fact that these cones intersect at a unique point, $(0, 0, 2)$, corresponds to the fact our points lie on the circle with center $(0, 0)$ and radius 2.

For a final example, we will verify the fact that any three points non-colinear points lie in a unique parabola. A parabola of the form

$$y = ax^2 + bx + c$$

can clearly be represented as the point $(a, b, c) \in \mathbb{R}^3$. A (x, y) gives rise to a plane

$$\{(a, b, c) : ax^3 + bx + c = y\}$$

in the Hough space $\mathbb{R}^3$. Planes of this form are called **Vandemonte planes**, and it is a fact from linear algebra that any three non-parallel Vandemonte planes coincide at a single point. In Figure 90, the points $(1, 5)$, $(2, 10)$, and $(3, 17)$ are represented as planes in this Hough space, and the fact that these planes coincide in a single point $(0, 2, 2)$ correpsonds to the fact that these three points lie on the unique parabola $y = x^2 + 2x + 2$.

It is clear that any reasonably parametrised family of curves can be detected by intersections of other objects in a Hough space. The more concurrence between objects in Hough space, the more likely it is that the correpsonding points lie on a common object of interest. For this reason, the Hough transform is often thought of as a geometrical "voting system" – the more some points agree on a particular shape, the more closely they will vote for it. The table below summarises these examples Hough spaces.

Objects of interest	Picture space P	Hough space H	Pts in P correspond to...
Straight lines	$\mathbb{R}^2$	$[0, \pi) \times \mathbb{R}$	Sinusoidal curve
Circles, radius R	$\mathbb{R}^2$	$\mathbb{R}^2$	Circles, radius R
Circles	$\mathbb{R}^2$	$\mathbb{R}^2 \times (0, \infty)$	Cones
Vertical parabolas	$\mathbb{R}^2$	$\mathbb{R}^3$	Vandemonte planes

Figure 86: The points $(1,0)$, $(2,1)$, $(3,2)$, and $(4,3)$ represented in Hough space

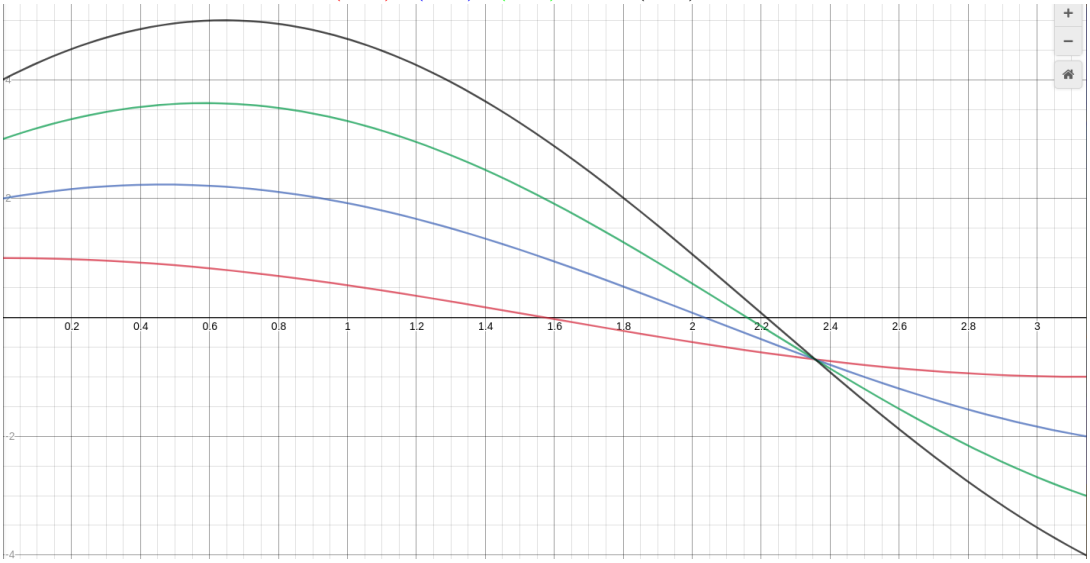


Figure 87: The points $(1,1)$, $(2,1)$, $(3,2)$, and $(4,3)$ represented in Hough space

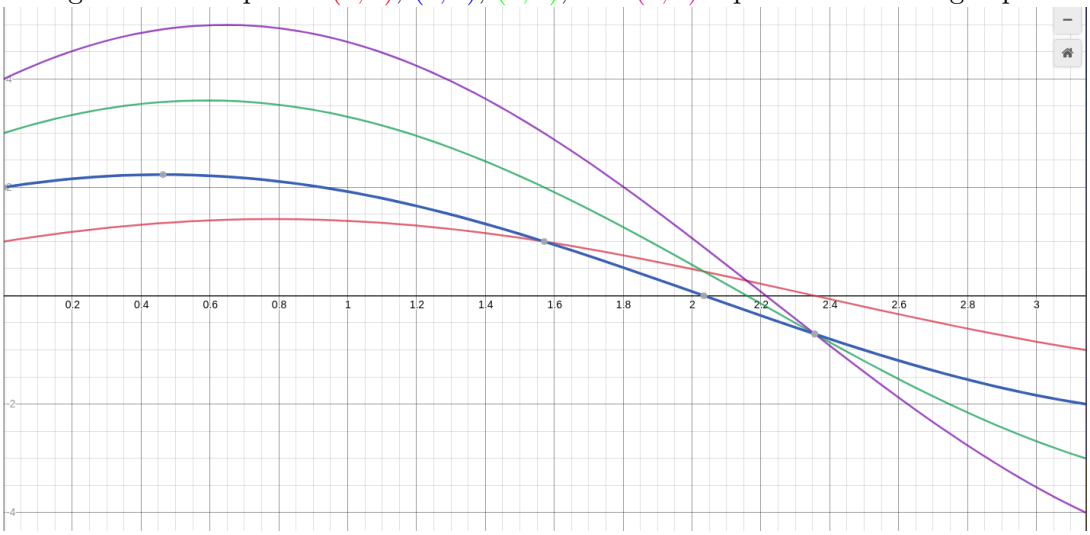


Figure 88: The points $(1, 0)$, $(1/\sqrt{2}, 1/\sqrt{2})$ and $(\sqrt{3}/2, -1/2)$ represented in circle Hough space

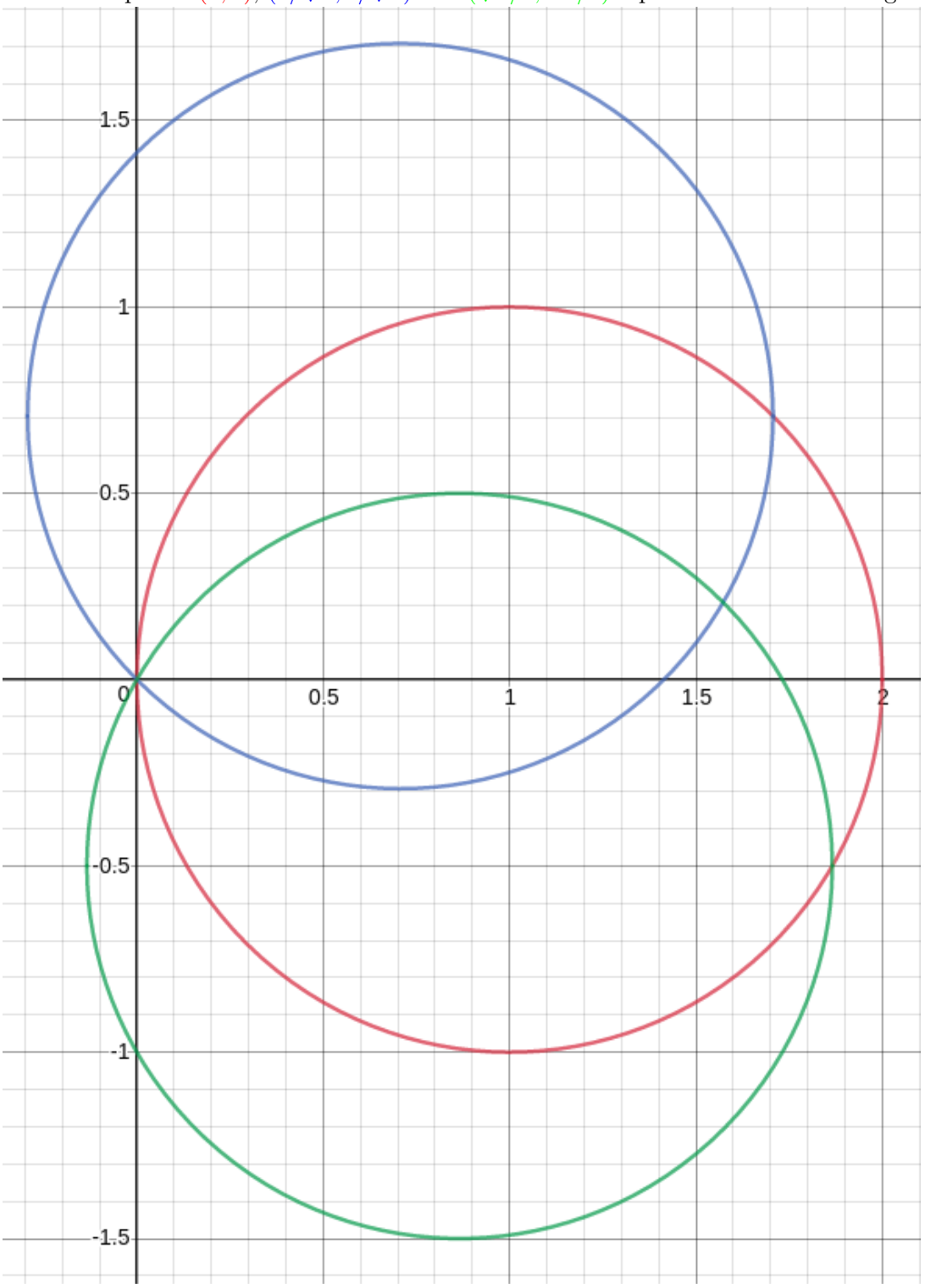


Figure 89: The points $(2, 0)$, $(\sqrt{2}, \sqrt{2})$, $(\sqrt{3}, 1)$ represented in a Hough space

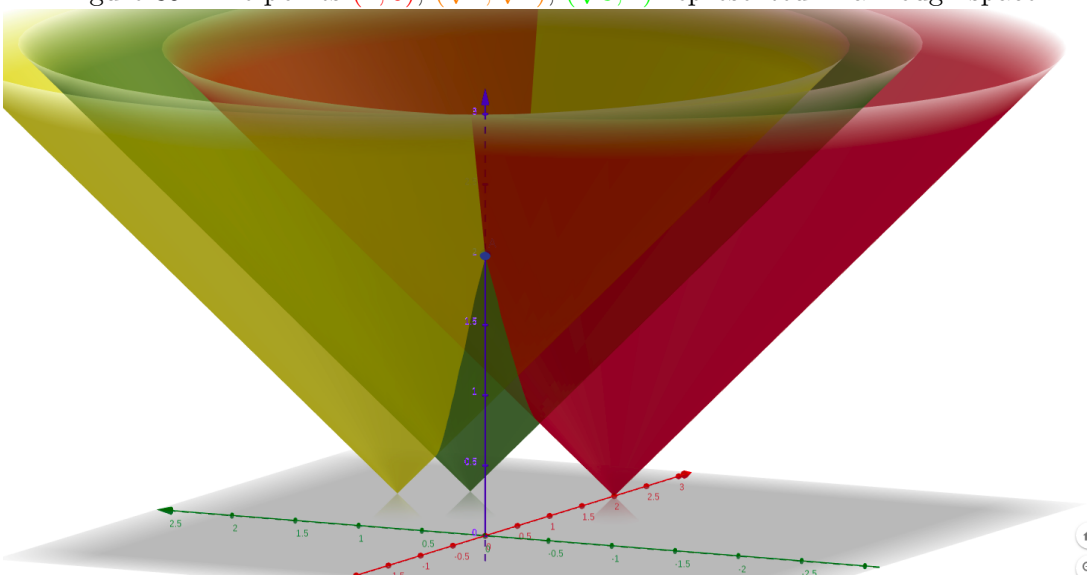
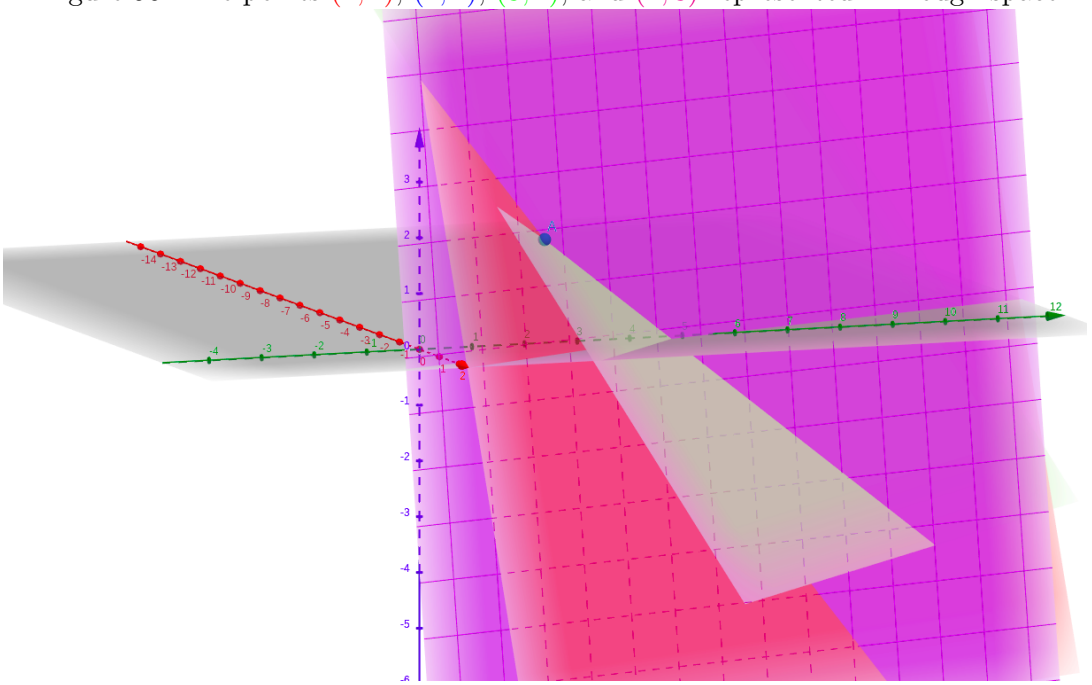


Figure 90: The points $(1, 1)$, $(2, 1)$, $(3, 2)$, and $(4, 3)$ represented in Hough space



1.6 SMRF, HAG and related signals

In this section, we discuss the mathematics which backs up the SMRF – the Simple Morphological Filter. This is essentially PDAL's "ground routine". This of course gives rise to HAG – Height Above Ground. The modularisation of these signals into their compositional components is naturally benefitial.

1.6.1 The dual rank operator

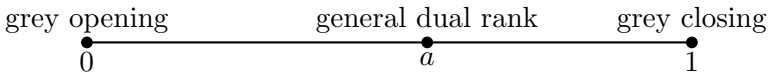
When we are looking at a signal with erratically changing values, it can be useless for some purposes to "smooth" its values. The various methods for smoothing give rise to a parametrised family of signals. The first type of such signal is the **grey opening**: A maximum filter followed by a minimum filter. Historically, this has been applied with square and circular masks, but, for convenience, we'll plot it using k-nearest neighbours. That is, the grey opening is applied in the following fashion:

- For a particular point, find its k-nearest neighbours, with intensities
$$i_0, i_1, \ldots, i_{k-1}.$$
- For that point, find its **minimum intensity**
$$j = \max(i_0, i_1, \ldots, i_{k-1}).$$
- Now find, for it's k-nearest neighbours, the corresponding minimum intensities
$$j_0, j_1, \ldots, j_{k-1}.$$
- The grey opening of the point is then
$$g = \min(j_0, j_1, \ldots, j_{k-1}).$$

Signal. *A minimum filter of a signal, followed by a maximum signal, is called a* ***grey opening*** *of that signal. Dually, a maximum filter followed by a minimum filter is called a* ***grey closing****.*

On a noisey signal, the grey opening is far too harsh: Maxima and minima can be astronomically large outliers. (PROVE THIS WITH EIGENVALUES.) To solve this issue, Eckstein and Munkelt proposed the dual rank filter in [2]. Instead of being a maximum filter followed by a minimum filter, it is the quantile[2] a filter followed by a quantile $1-a$ filter, for some $a \in [0,1]$.

Signal. *For $a \in [0,1]$, a quantile a filter followed by a quantile $1-a$ filter is known as a a* ***dual rank operator*** *with parameter a.*



[2]These aren't the precise terms in which Eckstein and Munkelt gave this filter, but it conveniently captures the essence of it. It should be noted that we ought to depart from the original a little bit, mostly because Eckstein and Munkelt were analysing orthophotographs, not 3D point clouds.

1.6.2 Ground extraction

The dual rank operator or grey opening can also be used to produce a DEM (Digital Elevation Map). This is achieved by taking the z values and applying a dual rank operator to them within an (x, y)-window. The idea is that this causes the features of the landscape to be pushed into the ground. The ground gets smoothed – perhaps too much if this process is carried out too violently – and the other objects "melt".

Figure 91: Road, raw intensity

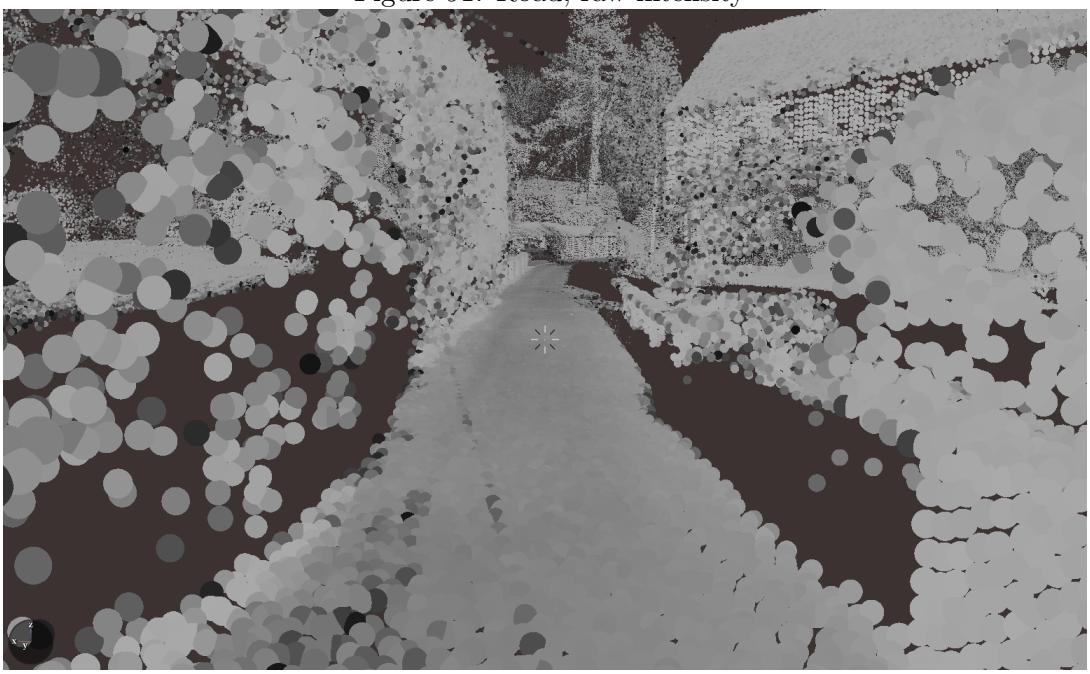


Figure 92: Road, grey opening intensity

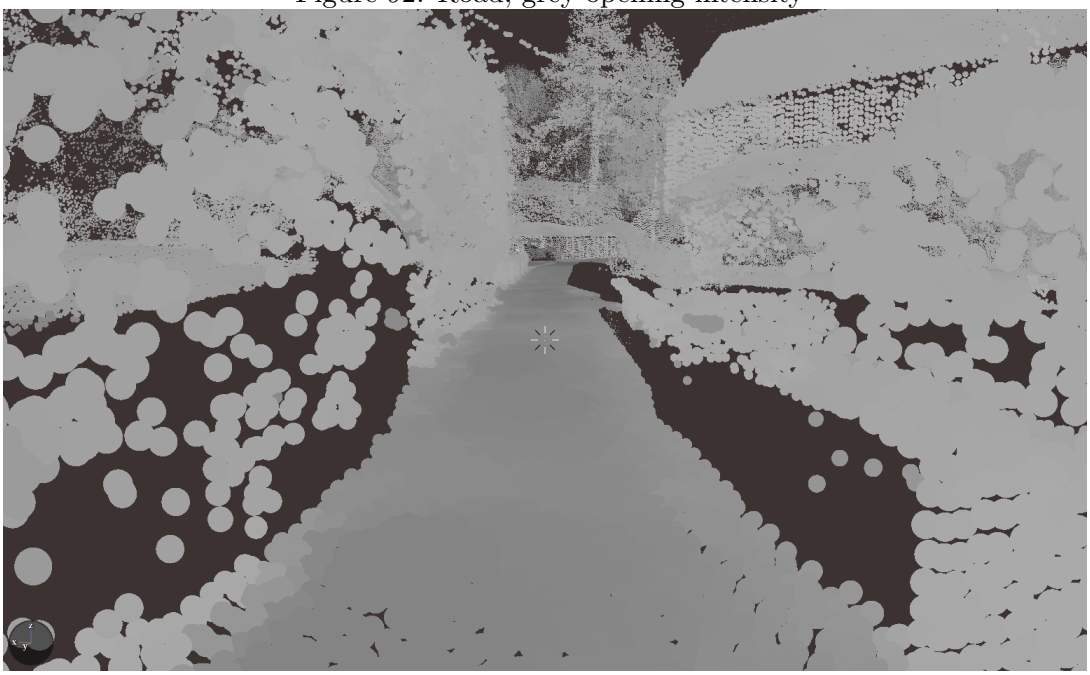


Figure 93: Curvature entropy, raw, $k = 50$, $\epsilon = 0.75$, $v = 2$, $u = 0.1$

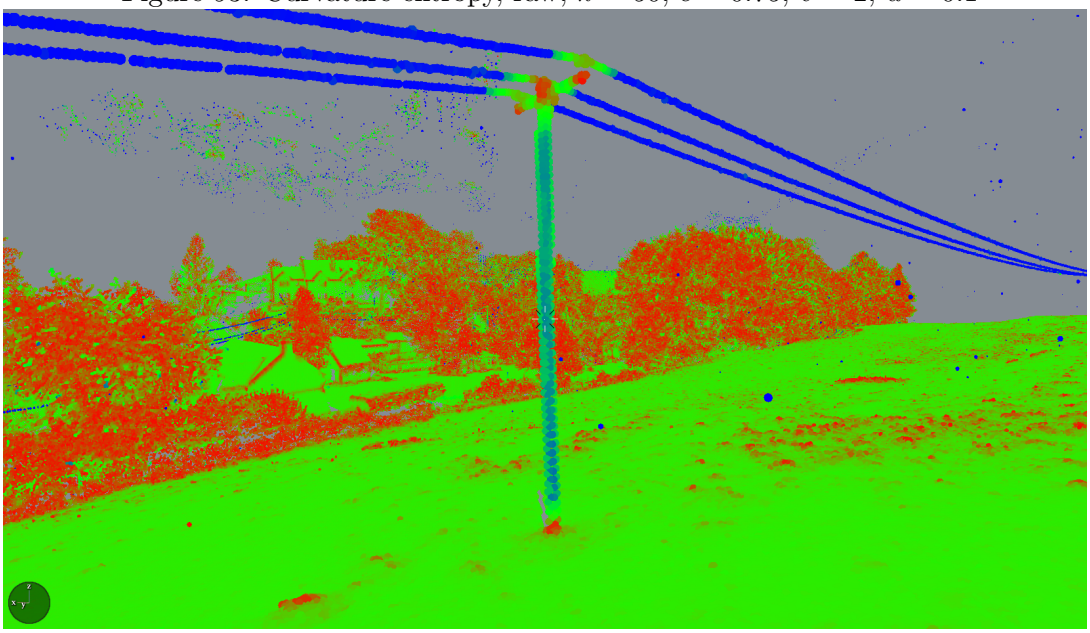


Figure 94: Curvature entropy, dual operator, $k = 50$, $\epsilon = 0.75$, $v = 2$, $u = 0.1$, $a = 0.25$

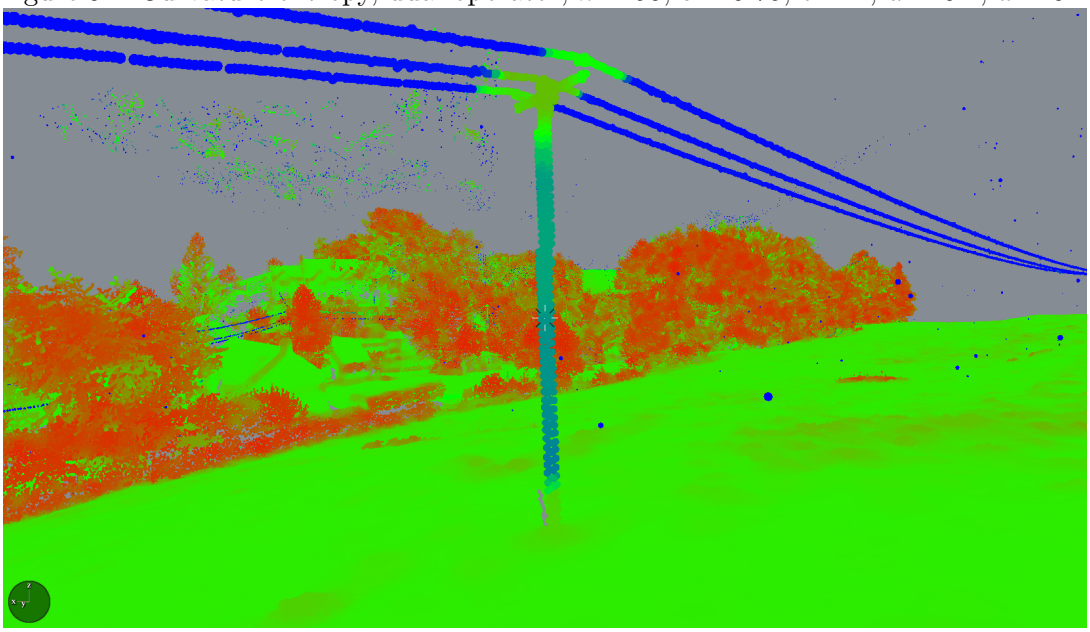
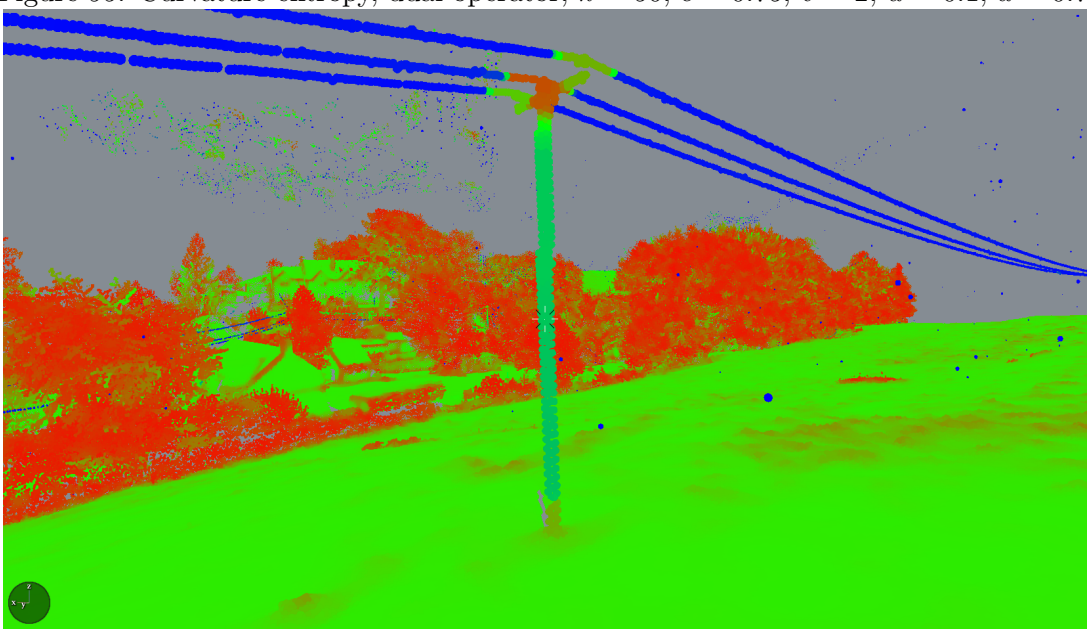


Figure 95: Curvature entropy, dual operator, $k = 50$, $\epsilon = 0.75$, $v = 2$, $u = 0.1$, $a = 0.75$



2 General approaches: Supervised learning and waveforms

We now have a library of signals. The idea is that the number of signals we have should be sufficient to select features from whichever landscape we're studying.

We should begin by updating our language slightly. What we've considered as signals so far are simply new attributes for the points in a point cloud. We'll call these **DAs (derived attributes)**.

Thus we have a family of DAs, which inhabit a family of data types $S_0, S_1, S_2, \ldots$ and we should be able to write down definitions of the different features which we wish to extract, using only these signals (though, we may have to iterate this process). The elementary view of point clouds is of course to think of them as subsets of $\mathbb{R}^3$ (or, technically, D^3 where D is the appropriate data type) maybe with other attributes thrown in to accomodate return numbers. However, during processing we think of them as subset of the **DA space**

$$S_0 \times S_1 \times S_2 \times \ldots.$$

When we filter points through a range of values for these DAs, we are selecting points which inhabit a hypercube inside DA space. Sometimes we need to iterate the process, producing new signals for the selected points and applying another selection to the new points of signal space.

We can view this subset of signal space as *the signal* for the point cloud. If we have a sufficient family of elementary signals, we should be able to use this signal to map a classification onto our points which fits our purpose. However, we make two general points:

- The relations within DA space which define various objects (vegetation, buildings, etc.) may be too complicated for us to spot. They might not be given by filters (=hypercubes).
- We need a method for "zooming out". All of our methods thus far have been to use local geometric properties. This is clearly insufficient: As humans, we realise what objects are by comparing them to other objects. We can tell the difference between a thin tree branch and a conductor (in a point cloud) because the thin tree branch is surrounded by other things, and the conductor clears some of its path.

We will begin to discuss a few methods which go tackling these problems. Broadly speaking, we will look at distributions as a method for "zooming out". Machine learning is a possible route to finding the more complex relations in DA space.

2.1 Distributions

We will take point clouds which have the DAs already attached to them – correlation coefficients, eigenvalues, curvature, entropy, etc. The DAs tell us the "local geometric properties" of points which we just mentioned. For example, a straight line or small cluster of points has a low entropy (a perfectly straight line or point has entropy 0), an evenly spread out plane has entropy $\log_3(2) \approx 0.63$, and an evenly spread out volume has entropy 1 (here we are discussing curvature entropy). Imperfect objects have values which are distributed about these three numbers – 0, 0.63 and 1. If we begin to look at a window in the point cloud (rather than a single point and its nearest neighbours), we can take the distribution of entropies inside that window (normalised so that we get a probability density function), and we should expect to see a distribution with three modes, if it contains all three types of object. Figure 97 shows a 3 meter window selected from a the base of the building in Figure 96 – it appears to show a greenhouse, the ground, and a wall. Figures 98 and 99 show two attribute distributions for Window 1 in Figure 97. As predicted, all three object types are present – with low entropy points (straight lines and

point-likes) only making a small appearence. The overall shape of the entropy plot is trimodal but there are some spikes which we didn't necessarily expect – perhaps these represent specific objects which we don't immediately notice in the picture. The plot of XY-linear regression in Figure 99 is also broadly what we expected because the picture does contain a lot of non-linear objects but also a small family of straight lines (the windows of the greenhouse) which presumably correspond to the spike at 1.

These plots are produced by takign the vector corresponding to a *numerical* attribute of points, and producing its histogram. Then the histogram is scaled down so that it becomes a probability density function.

Let's generate some more examples. We will look at a conductor on it's own to demonstate the nature of a fairly pure object rather than a mixed one. Figure 101 is a 1 meter window containing just conductor from Figure 100. As we can see from Figures 102 and 103, the entropies all occur near low values and the XY-linear regressions all occur near high values.

Figure 105 shows a 1 meter window selected from the top of a pylon in Figure 104. This is a more interesting object because it is a confusing mixture of states: It involves two crossed, thick cylindrical objects and some thin wires. We would expect certain parts of it to look locally like a straight line and some parts to look like a plane (the surface of the cylinders) and some parts to look 3D (corners) but not many perfectly voluminous areas. Indeed, the entropy plot in Figure 106 reflects this: There are some but not many low values of Entropy, quite a lot of modes in the immediate values and a sharp drop before we reach the very highest values. The XY-linear regression plot in 109 is similarly chaotic - with a small but not insignificant number of points giving the largest values.

Figure 96: “Greenhouse”

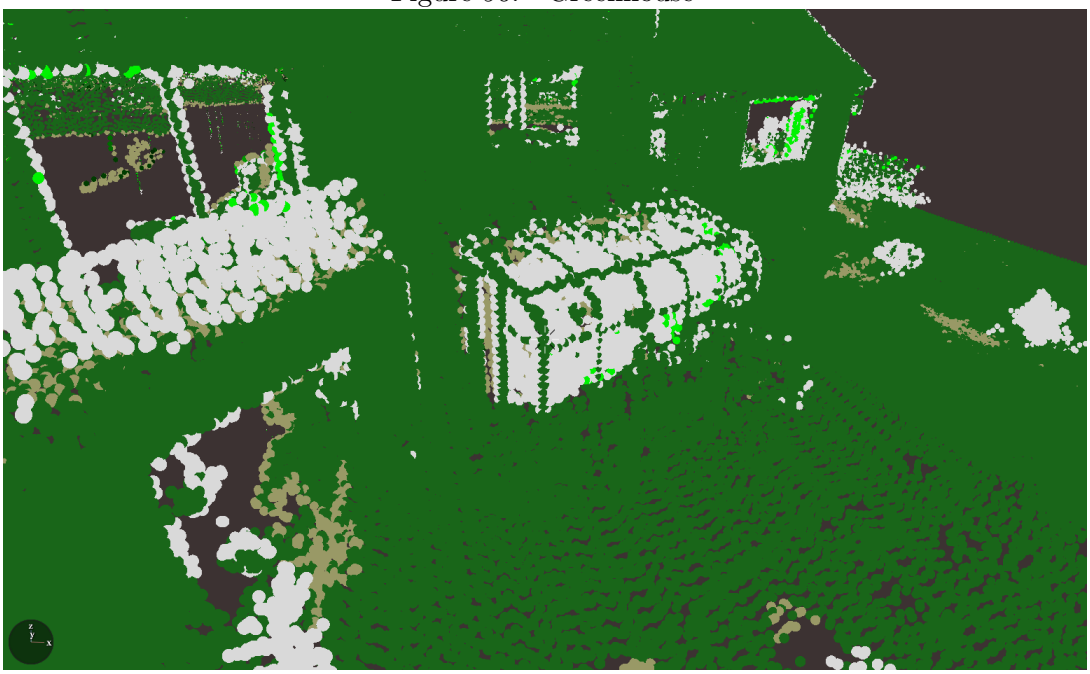


Figure 97: Window 1

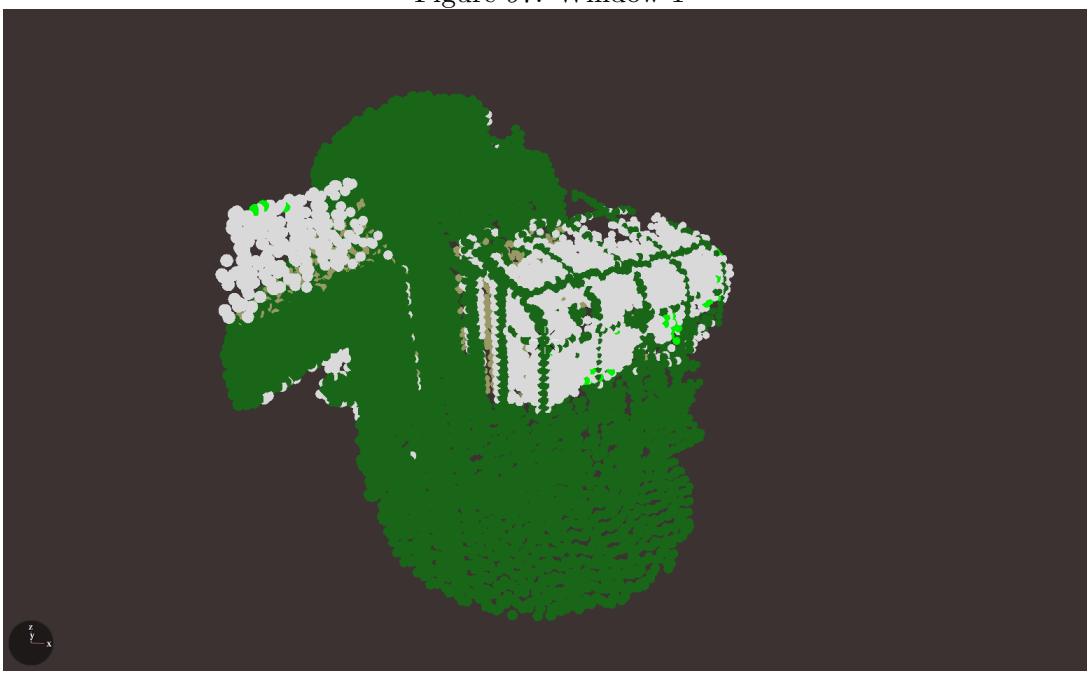


Figure 98:

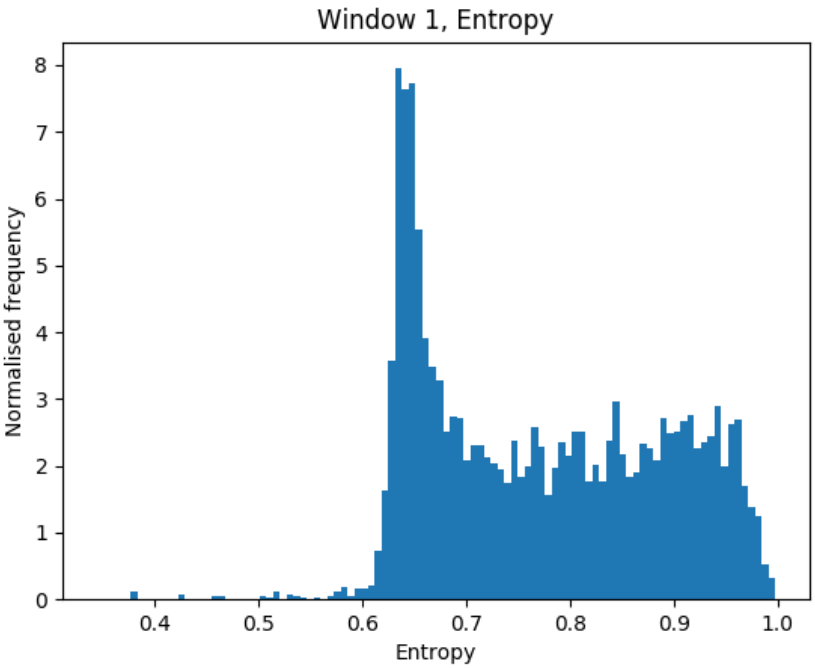


Figure 99:

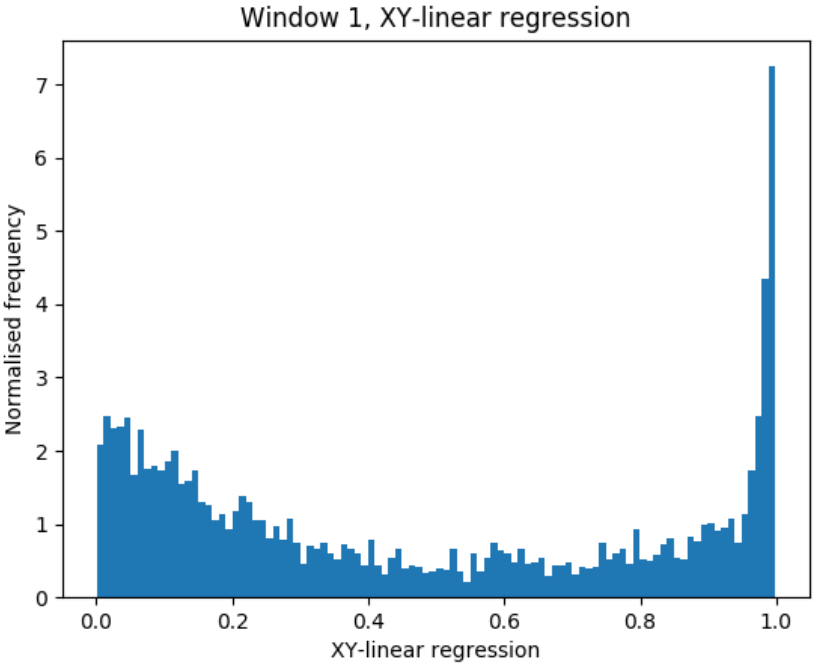


Figure 100:

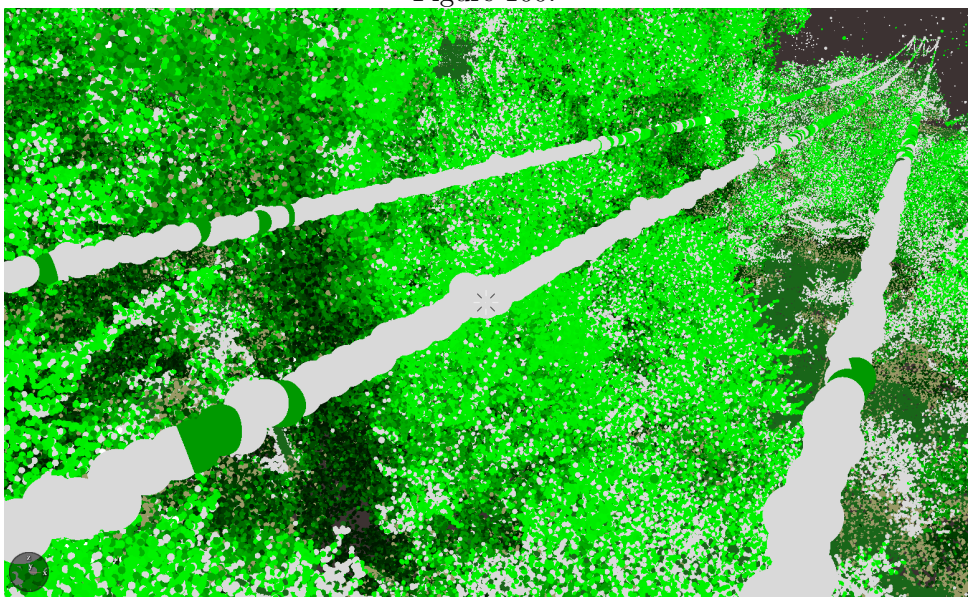


Figure 101: Window 2

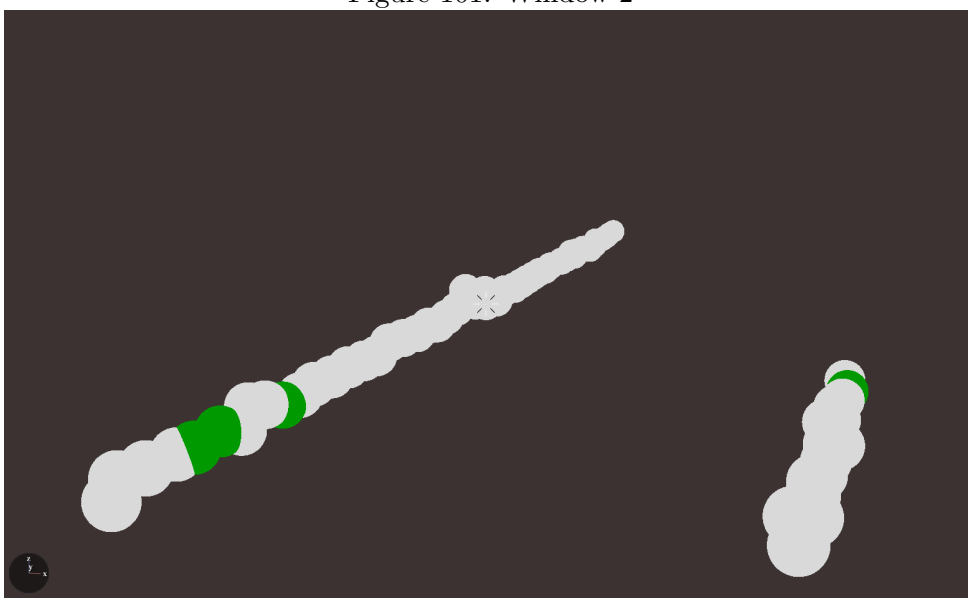


Figure 102:

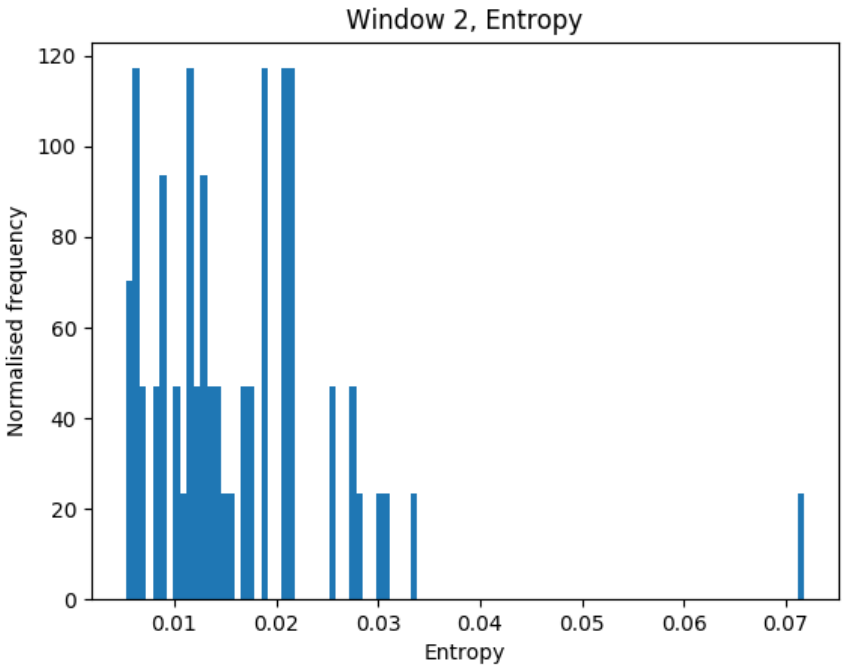


Figure 103:

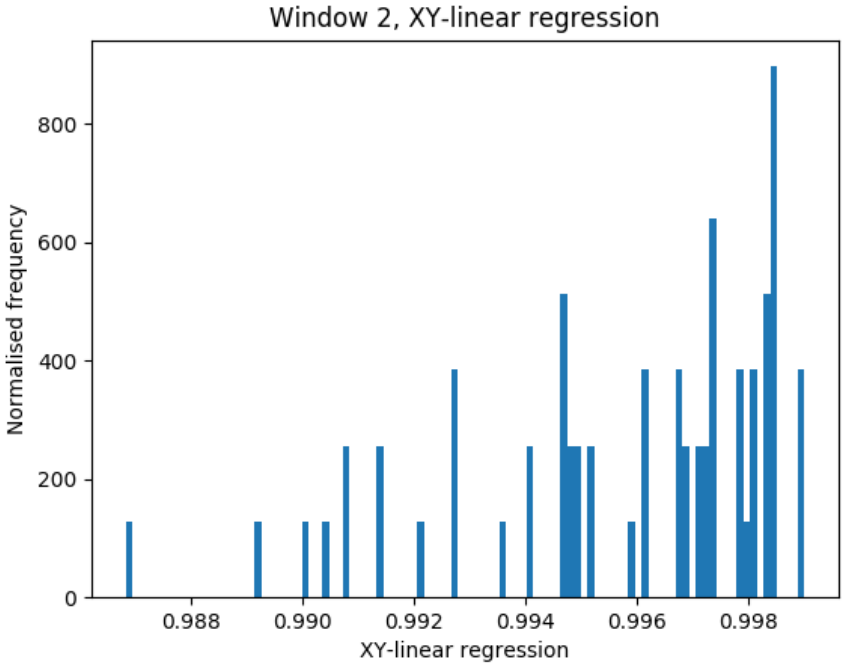


Figure 104:

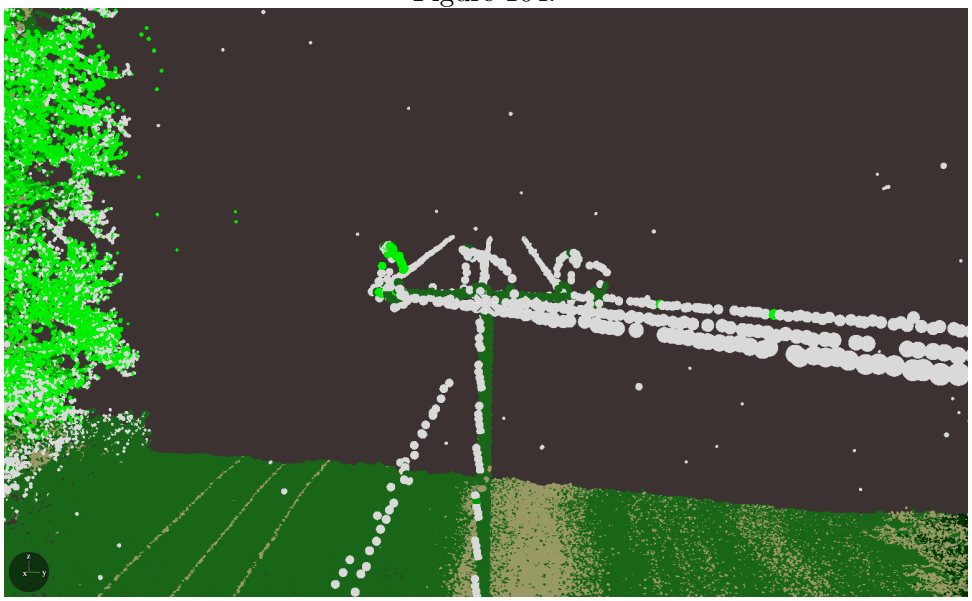


Figure 105: Window 3

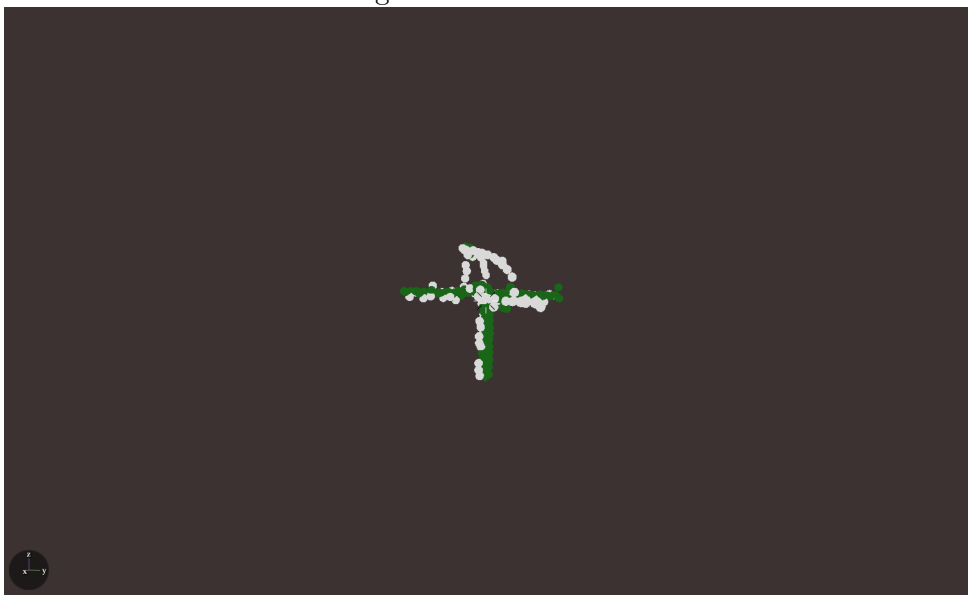


Figure 106:

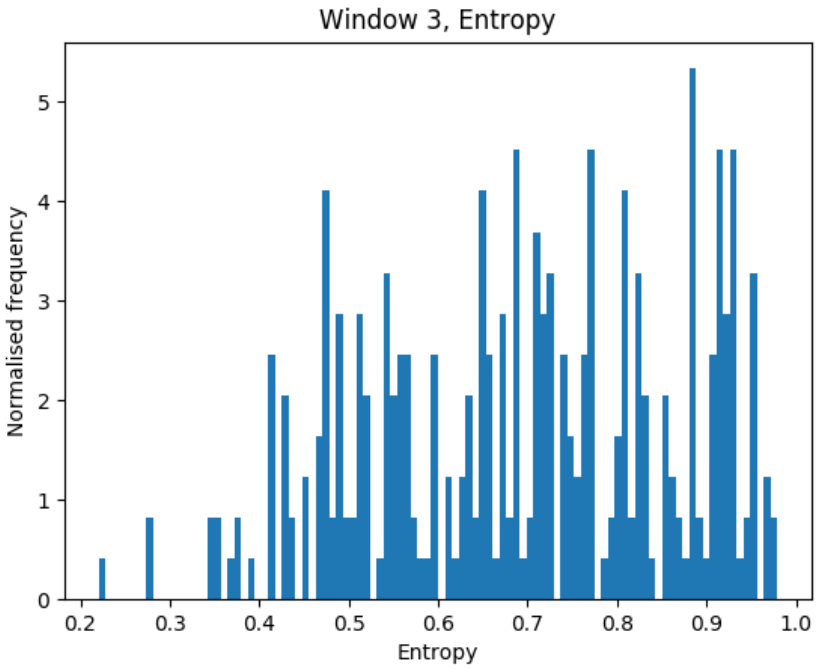
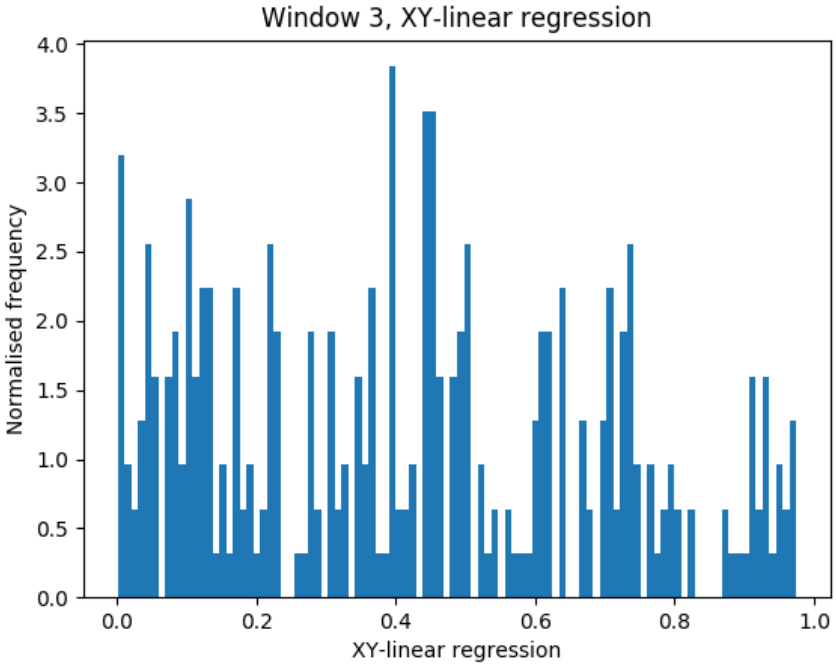


Figure 107:



2.2 The Jensen-Shannon Divergence

A **probability n-vector** is an array

$$p = \begin{pmatrix} p_0 & p_1 & p_2 & \dots & p_{n-1} \end{pmatrix} \in [0,1]^n$$

such that $\sum_i p_i = 1$. The distributions we have looked at so far are probability vectors: To produce them, we choose a number n of bins which we want to partition our attributes into, count the number of points in each bin, and then calculate the proportion p_i of points occuring in the i-th bin.

We would like a method of comparing distributions. Of course, a probability n-vector is simply an element of the $(n-1)$-simplex

$$\Delta^{n-1} = \left\{ \begin{pmatrix} p_0 & p_1 & p_2 & \dots & p_{n-1} \end{pmatrix} \in \mathbb{R}^n : p_i \geqslant 0, \sum_i p_i = 1 \right\} \subseteq \mathbb{R}^n,$$

it is tempting to use any of the metrics on $\mathbb{R}^n$ in order to decided what it means for two probability vectors to be "close". However, these are typically not sufficiently sensitive for the purpose of measuring distributions. For these purposes, we need a new metric. Consider, for example, the vectors

$$\begin{aligned} p &= \begin{pmatrix} 0.8 & 0.2 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\ q &= \begin{pmatrix} 0.9 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \\ r &= \begin{pmatrix} 0.8 & 0.1 & 0 & 0 & 0 & 0 & 0 & 0.1 & 0 \end{pmatrix}. \end{aligned}$$

In the metric d defined by $d(x,y) = \max_i |x_i - y_i|$, $d(p,q) = 0.2 = d(p,r)$. However, in the metric j defined by

$$j(x,y) = H\left(\frac{1}{2}(x+y)\right) - \frac{1}{2}\left(H(x) + H(y)\right),$$

where H denotes shannon entropy of a probability vector, $j(p,q) \approx 0.020$ and $j(p,r) \approx 0.005$, a completely different answer. We call j the **Jensen-Shannon divergence**. The spheres in the Jensen-Shannon divergence are of a different shape to those induced by the metrics on $\mathbb{R}^n$, as we see in Figure 108. We will use the Jensen-Shannon entropy j instead of any of the metrics inhereted from Euclidean space because the shape of its spheres are much more appropriate for comparing probability distributions.

Figure 108: Spheres in the Jensen-Shannon metric on Δ^2

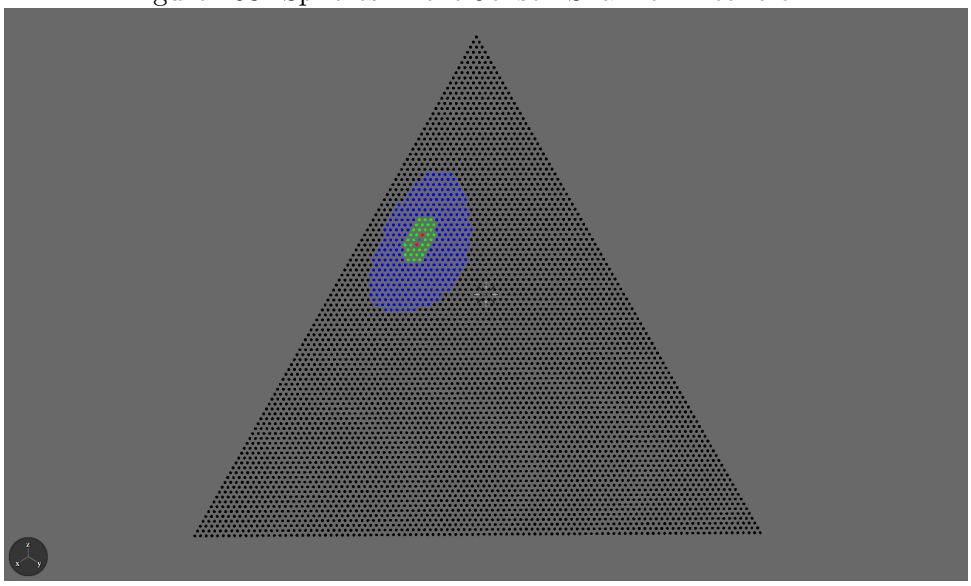
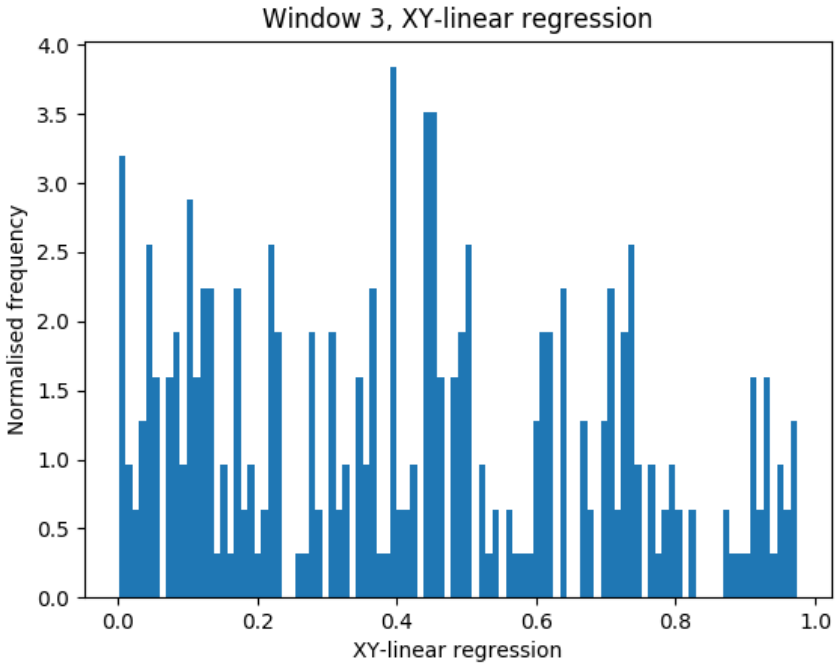


Figure 109:



2.3 Signal: An array of histograms

Suppose we have a point cloud and we split it into a lattice of bins which are made of cubes of the same size. This will give us a family $C_0, C_1, \ldots, C_{m-1}$ of cubes, each of which contains a family of points. Those points will have attributes which are derived in previous steps, e.g. entropy, isotropy, linear angle.

Now let's suppose we choose a collection of signals. Perhaps we choose entropy and planar angle. We can choose a number of bins n into which we wish to split the values of entropy and planar angle, and hence choosing n^2 bins for the combination. Therefore, in each cube C_i we have a probability vector p of length n^2 which is actually the histogram for the (entropy, planar angle)-pairs.

2.4 Supervised machine learning: training the signal space

Suppse we wish to extract a certain feature, e.g. conductors. So far the strategy has been to compute a signal for the point cloud, potentially with some chosen parameters, and find a way of using that signal to protect the desired feature while eliminating others. For example, most conductor points have isometry between 0.57 and 0.65, but many other features do not, so the isometry signal drastically reduces the search for conductors. However, such choosing of parameters is a time-consuming, subjective task, and can be hard to generalise and automate. So, we could skip it, by training the machine to recognise features in signal space. The training process is as follows:

- Take a point cloud which is representative of the data we will use.
- Classify it manually.
- Compute the signals for the classified point cloud.
- Transform each point in the point cloud into a point $(s_0, s_1, s_2, \ldots)$ in signal space. We will call these points **example points**.

Then, in order to classify a new point cloud, we would:

- Compute the signals for the new point cloud.
- Transform each point in the point cloud into a point in signal space. We will call these points **test points**.
- For each test point, find its nearest example point.
- Label each test point with the same classification as its nearest example point, where by "nearest" we mean according to some metric (possibly Euclidean) in signal space.
- Map the labels of the test points onto the classifications in the point cloud, thus outputting a classified point cloud.

An interesting side-effect is that there may be relations between different objects of which we were not aware, and in fact it may teach us whether or not our library of signals is sufficient.

3 Signals in production

Signals are highlighted in this colour. The greatest risks to genericity are highlighted in this colour.

3.0.1 Production pipeline 22

We begin with a large LAS file.

JSON000

Extract a tile from the large file using the chosen time interval.

JSON001

Read in the tile from JSON000.

Find the points which have classification other than 10.

JSON002

Read in the tile from JSON000.

Find the points which have classification equal to 10.

JSON003 *Aim: Denoise the flightline.*

Read in the file produced in JSON002 (i.e. points on the flightline).

Copy intensity into intensitySnapshot.

Identify points with intensity in the range $[630, 730]$. (Relies on nature of equipment.)[3]

Using a python script which implements the KD-three algorithm with $k = 40$, for the points with intensity in the range $[630, 730]$, replace intensity of each point with the number of neighbours it has within a 1 meter radius, with a maximum of 40 neighbours being recorded. This is, of course, a variant of point density.

Identify points which now have the (replaced) intensity $\geqslant 40$ (i.e. $= 40$), within the points which originally had intentsity within $[630, 730]$.

Within these points, repeat the process above with $k = 80$, and radius 0.25.

Identify points which now have the (replaced) intensity $\geqslant 80$ (i.e. $= 80$).

Classify the points from the output of JSON041 which do not have an intensity in $[630, 730]$ as classification 7.

Classify the points which were originally in $[630, 730]$ which now have intensity < 80 as classification 7.

Copy intensitySnapshot into intensity.

Write the result as a new file.

JSON004

Merge the results of JSON001 and JSON003.

JSON010 *Aim – Label silly points as noise*

[3] This range was chosen from noticing that the points in the central flightline fall within it.

This step classifies points which have logically inconsistent or very unusual attributes as noise, i.e. classification 7 (e.g. below sea-level, negative intensity, high return numbers).

JSON020 *Aim – Use SMRF to detect ground*

Use SMRF (Simple Morphological Filter) to identify points which are above ground, with the following parameters:

```
"slope": 0.1,
"cut": 0.0,
"window": 18,
"cell": 1.0,
"scalar": 0.5,
"threshold": 0.5,
```

JSON050 *Aim – Give some basic classifications to above ground points before further processing (in which those classifications might be changed)*

Classify the above ground points which are not on the flight line (i.e. not userData 10) as classification 4.

Classify the above ground points which are on the flight line (userData 10) to classification 10.

Classify the above ground points which are on the flight line and have userData 7 to classification 7.

Write reclassified data to a new file.

JSON060 *Aim – Set points too far from the flightline to "noise". Same goal in 061.*

Select the non-noise points which are not above ground (classification $\neq 4$).

Within those points, select the points which are 100 meters from the flightline.

Set the points outside of that 100 meter distance to classification 7.

Deselect classification 7 points.

JSON061

Select the points which are not classification 2 (i.e. non-ground).

Within those points, select the points which are 100 meters from the flightline.

Set the points outside of that 100 meter distance to classification 7.

Select only classification 4 points from those.

JSON071 *Aim of next few steps – Denoise based on relation to nearest neighbours*

Among first returns, find the k-distance with $k = 1$, and set the points with k-distance $\leqslant 1$ as classification 7, i.e. noise.

Among non-first returns, do the same.

Merge the results.

JSON081

Same as JSON081 and JSON082 in previous production.

JSON091

Same as JSON090 in previous production.

JSON100 *Aim – Use signals to begin extraction of conductors*

Set Z equal to HAG. (This step to make sure we can use linear angle to extract conductor.)

Compute the following signals and set them to corresponding attributes: XY-linear regression, linear regression, planar regression, eigenvalues 0, 1, 2, point density, rank, curvature, isotropy, entropy, planar angle, linear angle.

Select points with the following

$$\begin{aligned}
\text{xy linear regression} &\in [0.8, 1] \\
\text{linear regression} &\in [0, 1] \\
\text{planar regression} &\in [0, 1] \\
\text{eigenvalue 0} &\in [0, 0.001] \\
\text{eigenvalue 1} &\in [0, 10] \\
\text{eigenvalue 2} &\in [0.003, \infty) \\
\text{rank} &\in \{0, 1, 2\} \\
\text{curvature} &\in [0, 0.0033] \\
\text{isometry} &\in [0.57, 0.65] \\
\text{entropy} &\in [0, 0.5] \\
\text{planar angle} &\in [0, 1] \\
\text{linear angle} &\in [0.5, 1]
\end{aligned}$$

JSON 110-01,02,03,04,05,06 *Aim – A second pass to clean up the selection, eliminating as much vegatation as possible before hough lines*

To speed up the exposition, I will use a notation to describe some of the filtering. For a k-distance filter, we have two parameters: A number k and a distance d. Then we filter

points whose k-th nearest is more than d meters away. We apply the following (k, d)-pairs:

$$
\begin{aligned}
(1, 1) \\
(2, 2) \\
(2, 2) \\
(1, 1) \\
(3, 2.5) \\
(3, 2.5) \\
(2, 2) \\
(1, 1) \\
(4, 2.75) \\
(3, 2.5) \\
(2, 2) \\
(1, 1) \\
(5, 3) \\
(5, 3) \\
(4, 2.75) \\
(3, 2.5) \\
(2, 2) \\
(1, 1)
\end{aligned}
$$

Among those points, cluster (using PDAL) with a tolerance of 2 meters and a minimum number of points of 1.

Inside each cluster, take the average return index $(\overline{a}, \overline{b})$ (i.e. return $\overline{a}/\overline{b}$).

Select points with $\overline{a} = 1$ and $1 \leqslant \overline{b} \leqslant 3$, *and* which are non-last returns.

Among the resulting points, cluster using a tolerance of 1 meter and minimum number of points 1.

Select points which have at least 3 points in their cluster.

With a tolerance of 2 meters, cluster the resulting points.

Select points which have at least 6 points in their cluster.

With a tolerance of 4 meters, cluster the resulting points.

Select points which have at least 10 points in their cluster.

Select points with

$$
\begin{aligned}
\text{eigenvalue 2} &\in [0.002, \infty) \\
\text{rank} &\in \{1, 2\} \\
\text{isometry} &\in [0.001, \infty) \\
\text{entropy} &\in [0.001, \infty) \\
\text{linear angle} &\in [0.7, 1]
\end{aligned}
$$

More more k-distance filters – *identical to above.*

More clustering methods with tolerance-minimum pairs given by $(2, 10)$, $(3, 20)$, $(4, 40)$, $(4, 80)$.

JSON111

Copy Z into UserZ and set $Z = \text{HAG}$.

JSON120,125,130

These have not changed from production 21. The filtered points are given classification 14.

JSON140

Using the hough lines, find a "corridor" around the conductors.

Select the non-conductor points (classification $\neq 14$).

JSON141 *Aim – Generate signals to select pylons plus some intersecting vegetation*

Compute the signals with the following parameters:

```
"{\"k\":\"50\",\"radius\":\"0.50\",\"thresh\":\"0.001\",
\"spacetime\":\"True\",\"vspeed\":\"2\",\"decimate\":\"False\",\"u\"
:\"0.1\",\"N\":\"6\"}"
```

In order to begin picking pylons, select points with

$$\begin{aligned} \text{planar regression} &\in [0, 0.8] \\ \text{rank} &= 3 \\ \text{eigenvalue 1} &\neq 0 \\ \text{curvature} &\in [0, 0.2] \\ \text{isometry} &\in [0.6, 1] \\ \text{planar angle} &\in [0.5, 1] \\ \text{linear angle} &\in [1, 0.5] \end{aligned}$$

JSON142

Hough lines to pick pylons.

JSON150

Set pylons to classification 13.

3.0.2 Production pipeline 21

- **JSON000**
  - Extract a tile from the large file using the chosen time interval.
- **JSON005**

- Read in tile, adding HAG (= height above ground) as an extra dimension.
- Cut up file into equal time intervals.
- Write the output.

- **JSON010**
  - Read in the file.
  - Copy (or, in PDAL language, "ferry") classification into userData. We do this because we will want to refer to the original classifications later (1 is non-flightline, 10 is flight line), but we want to start changing classifications now.
  - Set points with intensity < 0 to classification 7 and userData 7.
  - Set points with $Z < 0$ (below sea level) to classification 7 and userData 7.

- **JSON020**
  - Use SMRF (Simple Morphological Filter) to identify points which are above ground, with the following parameters:

```
"slope": 0.1,
"cut": 0.0,
"window": 18,
"cell": 1.0,
"scalar": 0.5,
"threshold": 0.5,
```

  - Note that AC thinks the ground routine is unreliable in different contexts.

- **JSON030**
  - Classify the above ground points which are not on the flight line (i.e. not userData 10) as classification 4.
  - Classify the above ground points which are on the flight line (userData 10) to classification 10.
  - Classify the above ground points which are on the flight line and have userData 7 to classification 7.
  - Write reclassified data to a new file.

- **JSON040**
  - Read in the file produced in JSON030.
  - Write a new file containing just classification $\neq 10$ points from that file (i.e. flightline points).

- **JSON041**
  - Read in the file produced in JSON030.
  - Write a new file containing just classification $= 10$ points from that file (i.e. non-flightline points).

- **JSON042**
  - Read in the file produced in JSON041 (i.e. points on the flightline).

- Copy intensity into intensitySnapshot.
- Identify points with intensity in the range $[630, 730]$. (Relies on nature of equipment.)[4]
- Using a python script which implements the KD-three algorithm with $k = 40$, for the points with intensity in the range $[630, 730]$, replace intensity of each point with the number of neighbours it has within a 1 meter radius, with a maximum of 40 neighbours being recorded. This is, of course, a variant of point density.
- Identify points which now have the (replaced) intensity $\geqslant 40$ (i.e. $= 40$), within the points which originally had intentsity within $[630, 730]$.
- Within these points, repeat the process above with $k = 80$, and radius 0.25.
- Identify points which now have the (replaced) intensity $\geqslant 80$ (i.e. $= 80$).
- Classify the points from the output of JSON041 which do not have an intensity in $[630, 730]$ as classification 7.
- Classify the points which were originally in $[630, 730]$ which now have intensity < 80 as classification 7.
- Copy intensitySnapshot into intensity.
- Write the result as a new file.

- **JSON043**
  - Merge the output from steps 40 and 42.
- **JSON050**
  - Populate the HAG attribute using the appropriate filter.
  - Classify points with HAG < -0.25 as classification 7 (below ground noise).
  - Write the result as a new file.
- **JSON071**
  - Copy X into KDistance (presumably just to get a new attribute of appropriate length and data type).
  - Set KDistance $= 0$.
- **JSON072**
  - Read file with HAG and KDistance.
  - Identify points with classification $\neq 7$. (That is, deselect noise.)

    Note that PRD020 says return number $\neq 7$. This is a mistake.
  - Find the distance to k-th nearest neighbour among those pionts using PDAL's kdistance filter, $k = 1$. That is, find nearest neighbour distance.
  - Note that, in the PDAL documentation we have the following comment:

    ```
    The K-distance filter is deprecated and has been replaced by
    ‘filters.nndistance‘_.
    ```
  - Merge the classification $\neq 7$ and classification 7 points back together.

[4]This range was chosen from noticing that the points in the central flightline fall within it.

- Write k-distance into KDistance.

- **JSON073**
  - Identify points with Classification $\neq 7$, KDistance > 1 and return number 1. Give those points classification 7. (Not huge risk here unless coordinates are measured in different units.)
  - Merge those points back in and set KDistance $= 0$ for all points.

    Note that the same return number/classification mix up happens here, too.

- **JSON081**
  - Read file with HAG and KDistance.
  - Identify the points with Classification $\neq 7$, and perform the kdistance filter, $k = 3$.
  - Merge with rest of points, writing k-distance into KDistance.

- **JSON082**
  - Identify points with Classification $\neq 7$, KDistance > 1.2 and return number 1. Give those points classification 7.
  - Merge those points back in and set KDistance $= 0$ for all points.

    Note that the same return number/classification mix up happens here, too.

- **JSON090**
  - Identify the returns 1/1 with classification 4 (above ground) and intensity < 600. Set those points to classification 7.
  - Merge those points back in.

- **JSON100**
  - Identify points with classification 4 (above ground).
  - For those points, copy classification into USERReturnNumber (just to get an attribute of correct type and length.
  - Set USERReturnNumber $= 10a + b$ where a/b is the return index.
  - Identify points with USERReturnNumber $= 11, 22, 33, 44, 55$. (i.e. last returns, assuming numberOfReturns < 6).

    According to AC it's better not to do this.

    What follows involves in this JSON heavy choice of parameters. This is risky.
  - Find normal curvature with $k = 8$. Copy this curvature into lovelyCurves008. (After multiplying by 100 with a python filter.)
  - Find normal curvature with $k = 101$. Copy this curvature into lovelyCurves101. (After multiplying by 100 with a python filter.)
  - Copy curvature into lovelyCurves and set lovelyCurves $= 0$.
  - Take lovelyCurves $= \min($lovelyCurves008, lovelyCurves101$)$. (Using a python filter.)

– Identify points with lovelyCurves $\leqslant 1$. For these points compute the same lovelyCurvature as above, but relative only to other points with lovelyCurves $\leqslant 1$. Repeat this process. And repeat again.

I think the curvature on line 173 should have been lovely curvature, to fit this pattern

– Identify points with lovelyCurves $\leqslant 1$. Identify the points among those with eigenvalue 0 $\leqslant 0.001$, and repeat this step for a further 7 passes.

– Write out the result along with HAG.

- **JSON110**
  - WRITE UP THE CLUSTERING STEPS.
- **JSON111**
  - Copy Z into userZ.
  - Copy HAG into Z.
  - Write out the result.
- **JSON120**
  - Perform hough3D python filter. The hough lines are found by the following steps.
    * Clip $4 \leqslant$ HAG $\leqslant 12$.
    * Select class 2 (non-ground, non-noise).
    * On 6.4m $\times$ 6.4m subtiles, compute 3d hough lines (using another python script).
    * Reject hough lines which are more than 82 degrees from the vertical.
    * After this we put, around each hough line, two polygons $A \subseteq B$ and reject hough lines which have points in $B - A$. (According to AC there is a threshold on $|B - A|$.)
    * All selected lines are then written into a file.
- **JSON125**
  - Read file with HAG and userZ.
  - Perform hough3D2 python filter which is a second pass of hough3D to find some points which were missed. (We seem to be lacking clear knowledge of what this step is doing, precisely.)
  - According to AC there is a bug in which hough3D2 overwrites hough3D – presumably the result is equivalent to just doing JSON120 and not doing JSON 125.
- **JSON130**
  - Set points on hough lines to classification 14 by using applyHough.
- **JSON135**
  - Read in file with HAG.
  - Perform the "virus" filter with

```
"pdalargs":"{\"itter\":\"1\",\"class\":\"14\",\"clip\":\"0.25\"}"
```

The virus tool finds all points within a clip (0.25 in this instance) of a certain class of points (classification 14 here), and sets them to the same classification, and repeats a certain number of times (this time we don't repeat). This causes the chosen classification to leak out a little.

- **JSON140**
  - Apply the pylon extracting python filter to non-classification 14 points (to preserve the conductor we've already extracted). This tool works similarly to the extraction of conductors: Instead of finding horizontal 3D lines, we find vertical ones by selecting angles < 5 degrees from the vertical. This is similar to using the angles between $\mathbf{v}_0$ and the vertical.
- **JSON150**
  - Set points on conductors to classification 13 using applyHough.
- **JSON160**
  - Find all points within a 3m $\times$ 3m cube of conductor (classification 14) which is not classification 14, 13, 10, 7, or 2.
  - Set those points to classification 15.
- **JSON170**
  - **01** Find all points with classification 13, 14 or 15 (putatively, dangerous vegetation, conductor, and pylons, respectively).
  - **02** Find all points with classification 15 (putative pylon).
  - **03** Find all points with classification 14 (putative conductor).
  - **04** Find all points with classification 13 (putative dangerous vegetation).
  - **12**
    - Take the output of 02. Calculate curvature $\times 100$ with $k = 101$.
    - By taking at most 251 neighbours of any point which fall within a 1 meter radiues, calculate the local median of each point and set to the new curvature. Then do that again, twice. This is an instance of the dual rank operator.
    - Find the points with curvature > 2, to be thought of as vegetation.
    - Find the points with curvature $\leqslant 2$, to be thought of as buildings.
    - Reclassify the points with curvature $\leqslant 2$ as classification 6.
    - Merge curvature > 2 and curvature $\leqslant 2$ back together.
    - Use virus to spread classification 6 by 1 meter. Repeat this with clips of 0.5, 0.25, and again by 0.25.
  - **13**
    - Find classification 6 points and cluster them with a tolerance of 1 meter and at least 1 point in each cluster, using PDAL's clustering filter.
    - Copy the clusterIDs into clusterIDCountMax and set clusterIDCountMax $= 0$.
    - Set clussterIDCountMax at any point to the number of points which share the same cluster ID as that point. Cap that count at 10000.

    * Find points with clusterIDCountMax < 1000, and reclassify them as classification 4.
    * Merge the classification 15, 6 and 4 back together.)
  - **14** Merge the result of 13 and 4: The new classification 15, 6, 4 and 13, back together.
  - **15** Take points with classification 6 (building) and 13 (vegetation), and use the virus tool to with

    ```
    {\"itter\":\"1\",\"class\":\"6\",\"clip\":\"2.00\"}"
    ```

    to spread classification 6 points over a 2 meter clip. Repeat this step three times.
  - Merge classification 4, 6, 13 and 15.
  - **16** Merge the result of **03** and **15**.
  - **17**Among the points with classification 6 or 14, use the virus tool with

    ```
    "{\"itter\":\"1\",\"class\":\"6\",\"clip\":\"0.5\"}"
    ```

    Repeat this process.
  - **18** Merge the results of **16** and **01**.

- **JSON175**
  - Load points with HAG.
  - Sort points according to GPS time, with ascending order.
  - Take classification 15 points and cluster them with a tolerance of 1 meter, using PDAL's cluster filter.
  - Copy cluserID into clusterIDCountMax.
  - Set clusterIDCountMax to the number of points which share the same cluster ID as that point. Cap that count at 10000.
  - Set points with clusterIDCountMax < 500 as classification 4.
  - Merge these points back in with other points.

- **JSON200** Using a python function, select points within a shape file.

References

[1] David Poole, *Linear Algebra: A Modern Introduction.* Thomson Brooks/Cole, 2006.

[2] W. Eckstein and O. Munkelt, *Extracting objects from digital terrain models.* Proceedings of the International Society for Optical Engineering: Remote Sensing and Reconstruction for Three-Dimensional Objects and Scenes, 1995.