Institut für Kartographie und Geoinformatik | Leibniz Universität Hannover

Classification II
Random Forests,
Features for LiDAR data

Claus.Brenner@ikg.uni-hannover.de

Leibniz Universität Hannover

Random Decision Forests

Random Decision Forests

- Random Forests, Decision Forests
- Observation:
  - Often, a combination of several “weak” classifiers is very successful
  - C.f. learning meta-algorithm “AdaBoost” (adaptive boosting)
- In our case: instead of one tree, we want to use a lot of trees = a forest
  - → **Forest**, Decision **Forest**
  - Also called: ensemble (general term – not only for forests)
- If we train several trees using the previous algorithm, the results will be identical
  - Since the algorithm is deterministic
  - Therefore: training needs to have a random component
  - → **Random** Decision Forest

Randomization by training data selection

- Instead of using the entire training dataset, we generate new datasets using sampling with replacement
- This is called: „bagging“ (bootstrap aggregating)
- The new training datasets may have less elements, but also may have the same number of elements
- Since sampling with replacement is used, they will, in general, have duplicate elements

$$M = \left\{\mathbf{x}^{(1)}, \mathbf{x}^{(2)}, \mathbf{x}^{(3)}, \mathbf{x}^{(4)}, \mathbf{x}^{(5)}, \mathbf{x}^{(6)}, \mathbf{x}^{(7)}, \mathbf{x}^{(8)}\right\}$$

$$M_1 = \left\{\mathbf{x}^{(7)}, \mathbf{x}^{(1)}, \mathbf{x}^{(7)}, \mathbf{x}^{(4)}\right\}$$

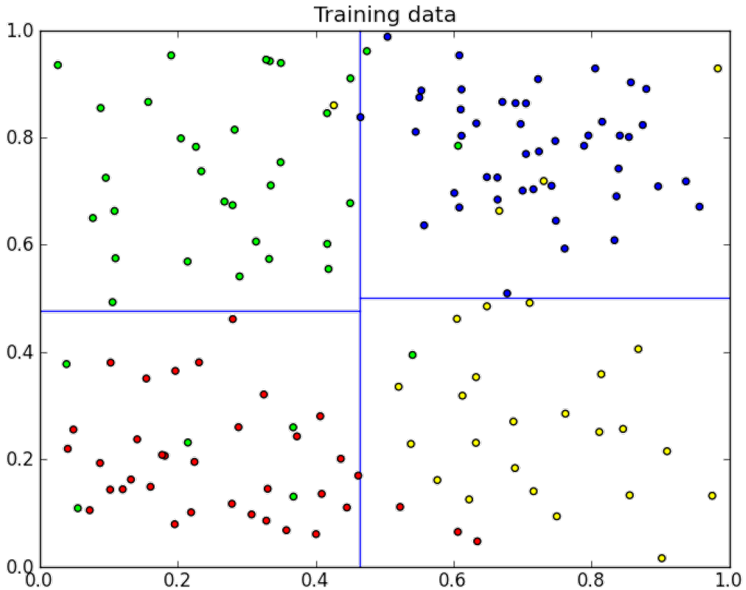
$$M_2 = \left\{\mathbf{x}^{(6)}, \mathbf{x}^{(2)}, \mathbf{x}^{(2)}, \mathbf{x}^{(6)}\right\}$$

$$M_3 = \left\{\mathbf{x}^{(1)}, \mathbf{x}^{(3)}, \mathbf{x}^{(4)}, \mathbf{x}^{(5)}\right\}$$

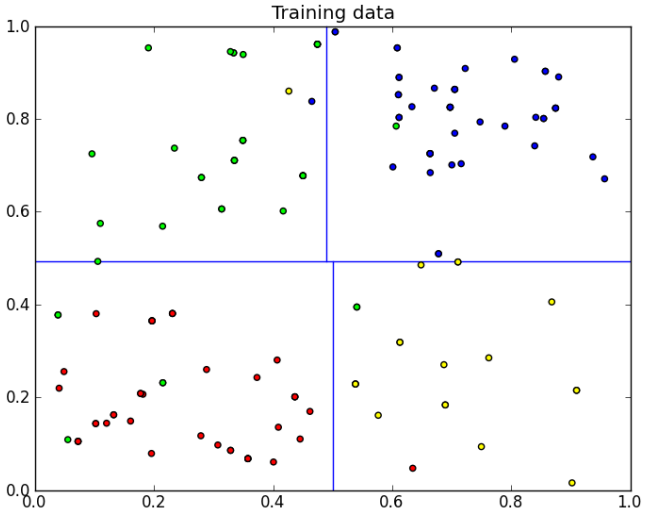
...

Randomization by training data selection

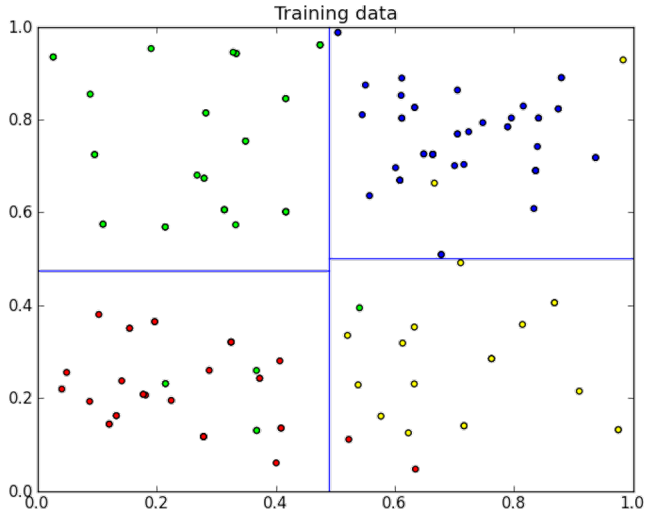
- Each set is used to train another tree
- Therefore, although each tree is obtained deterministically, the trees are different



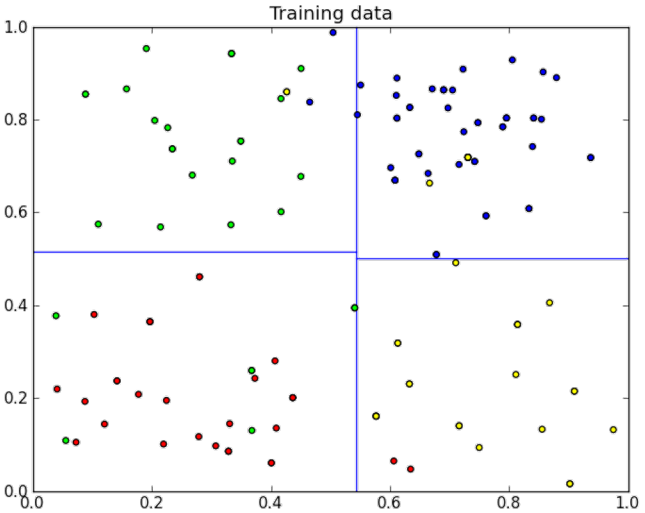
M



M_1



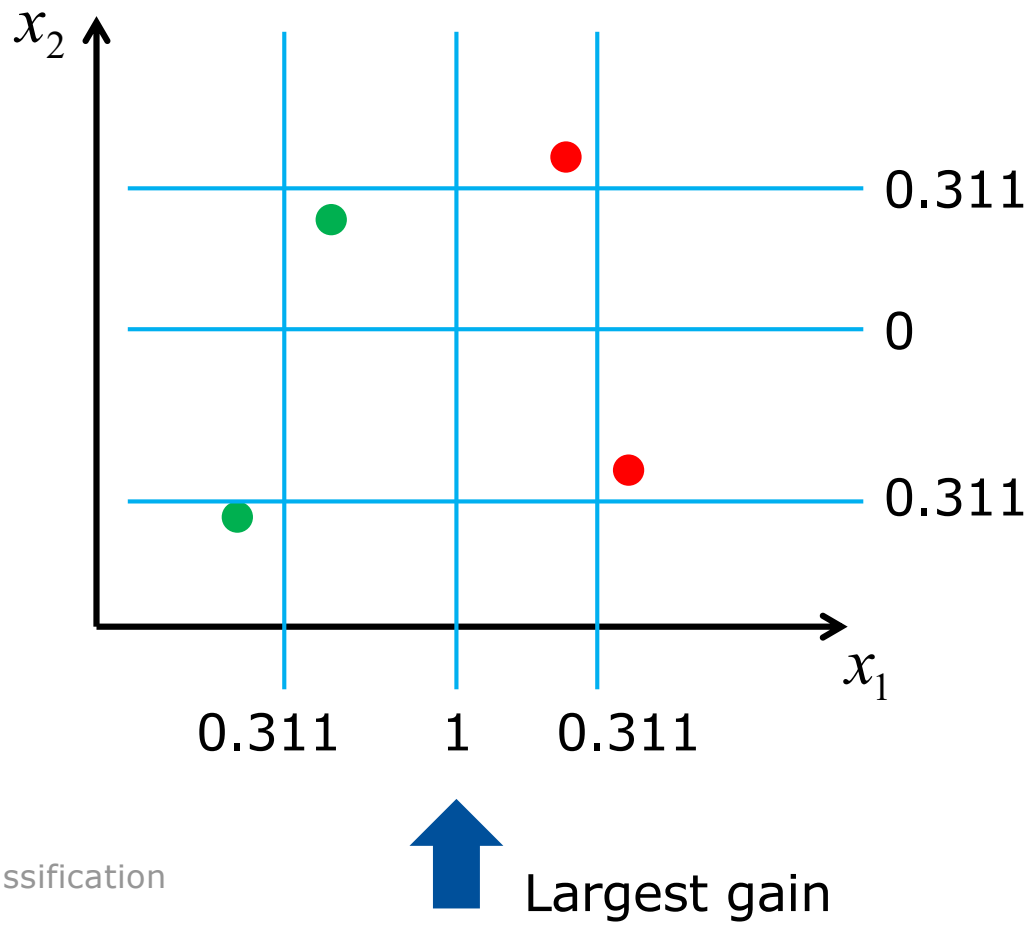
M_2



M_3

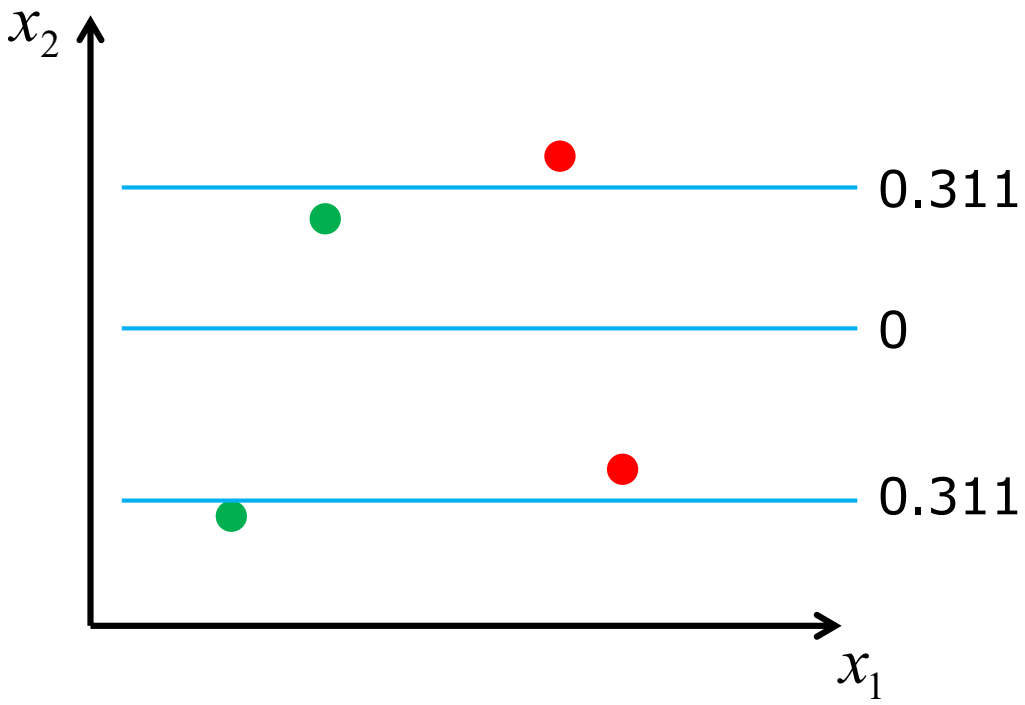
Randomized node optimization

- So far: deterministic optimization
  - Search over "all possible subdivisions"



Randomized node optimization

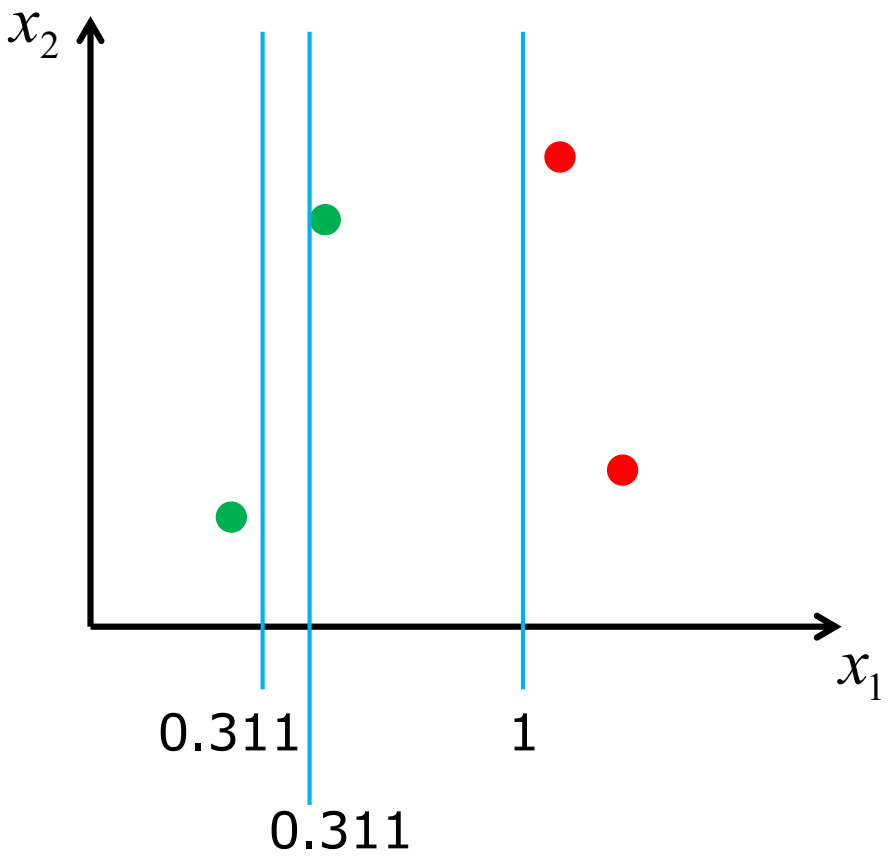
- Now: only optimize over a subset of all possibilities
  - E.g.: randomly pick one dimension. Only optimize along this dimension.



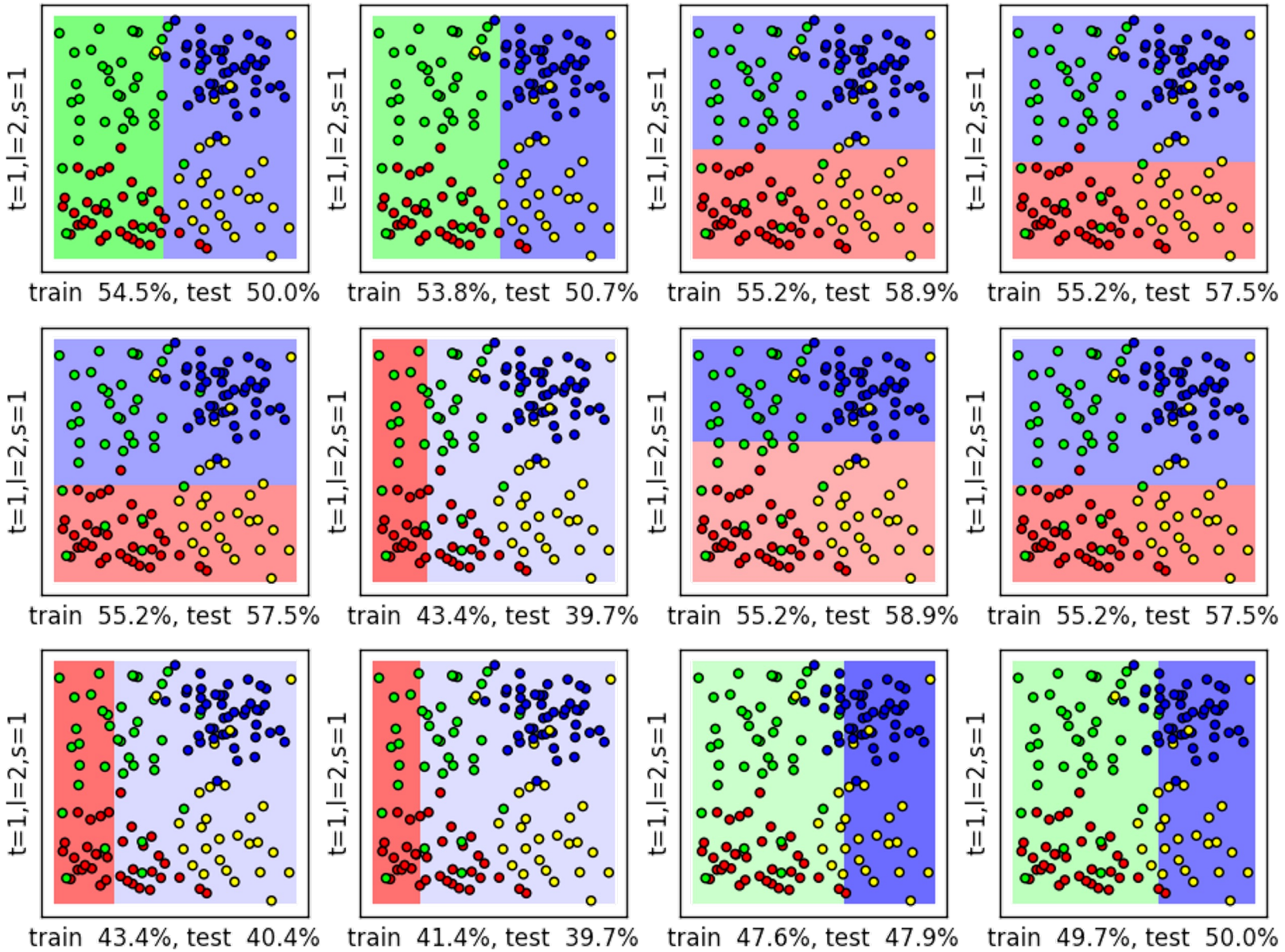
e.g. pick 2nd dimension. only optimize gain for all choices of x_2

Randomized node optimization

- Now: only optimize over a subset of all possibilities
  - Or: do not test all possible subdivision values, but test a number of random choices, then select the optimum among these

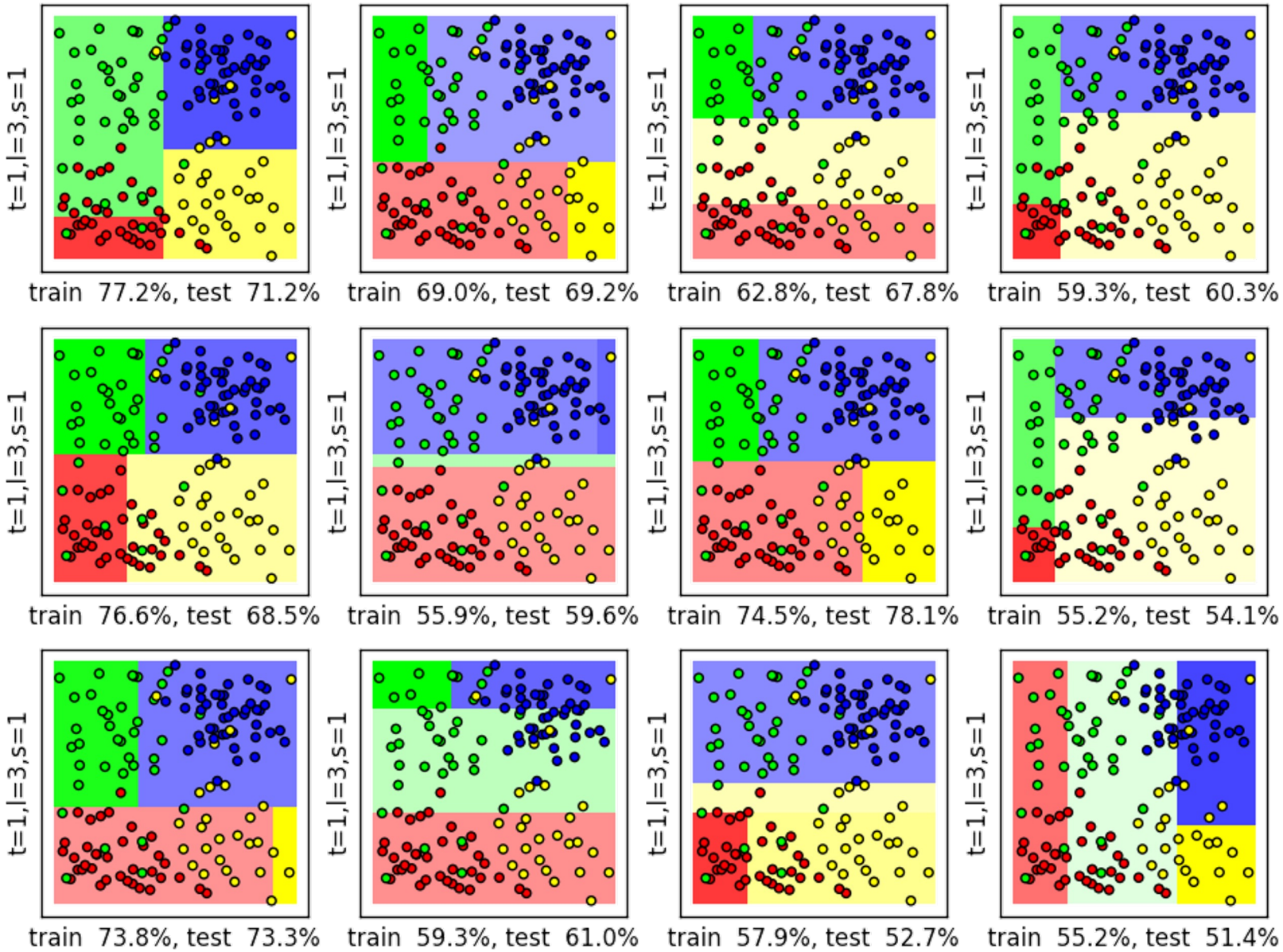


E.g. “test one (random) subdivision per dimension”



(tree depth 2)

E.g. “test one (random) subdivision per dimension”



(tree depth 3)

Randomization

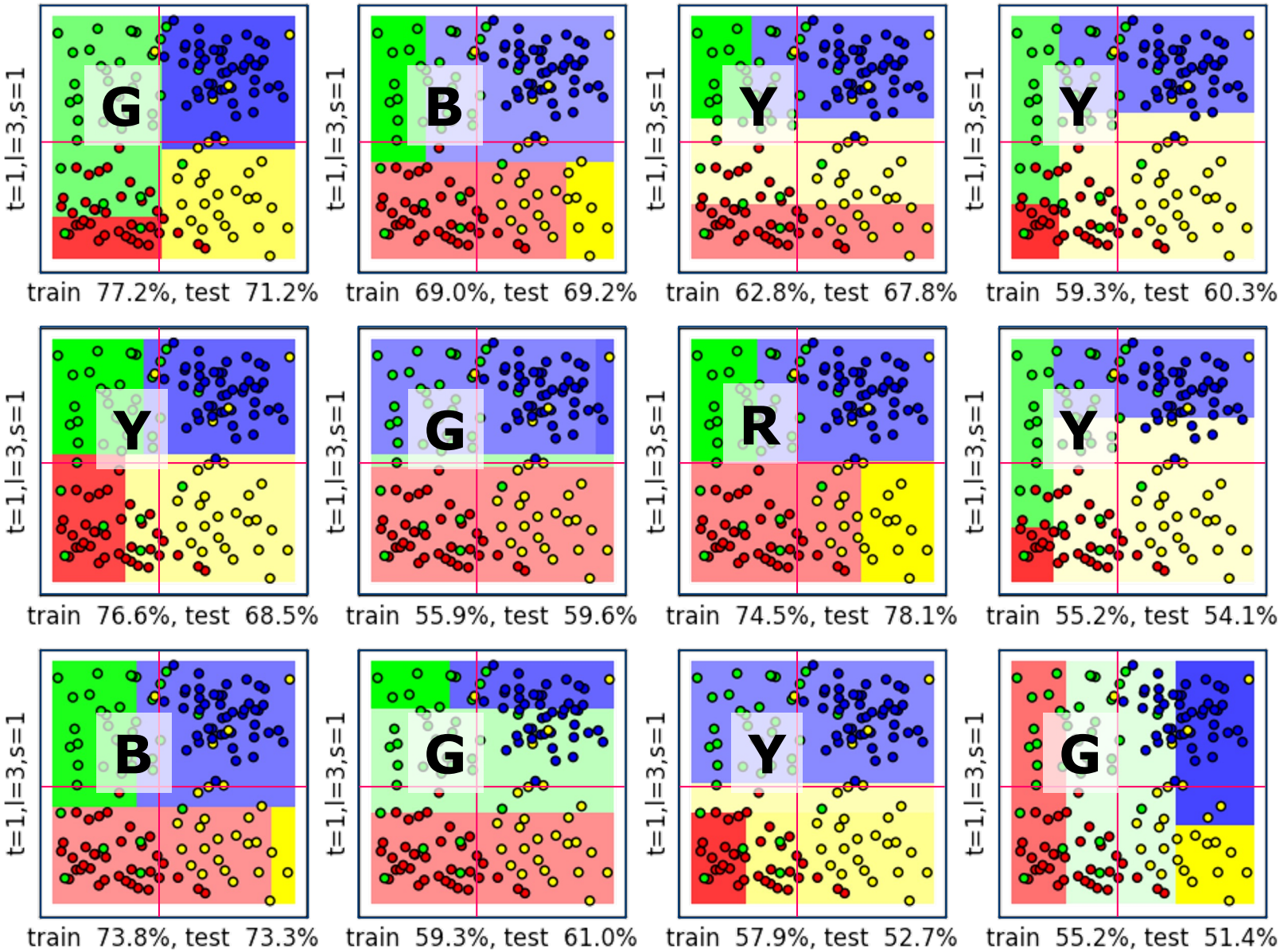
- Randomization of the trees can be obtained using
  - Random selection of training data (bagging) or
  - Randomized optimization of the nodes
  - or both!

- Variant: „extremely randomized trees“:
  - Select a subset K of dimensions
    - E.g.: $\mathbf{x} = (x_1, x_2, x_3, x_4, x_5, x_6, x_7, x_8)$
    - Select subset, e.g. K={2,3,6,7}
  - For each selected dimension, pick one (randomly selected) subdivision point: $\{\overline{x}_2, \overline{x}_3, \overline{x}_6, \overline{x}_7\}$
  - Among all those candidates, find the one with the largest information gain

The ensemble model

- How do we obtain the classification result for a (newly) given instance?
- All we have so far are T (independent) trees
- The trees will usually arrive at different results
- How can we compute a single, global result from all individual results?
  - Majority (most frequent classification result)?

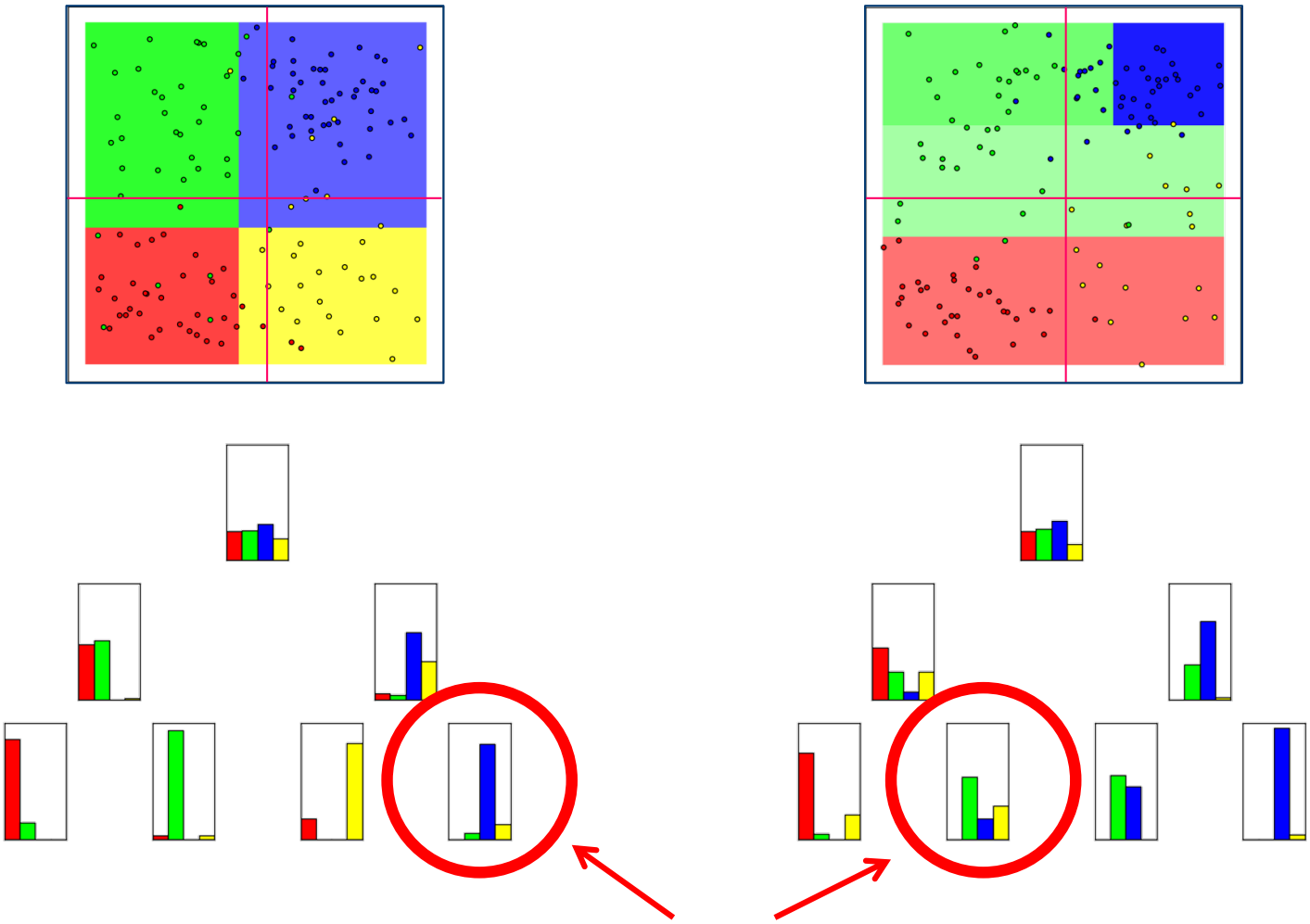
Example: new instance, most frequent result wins



R=1, G=4, B=2, Y=5 → Yellow

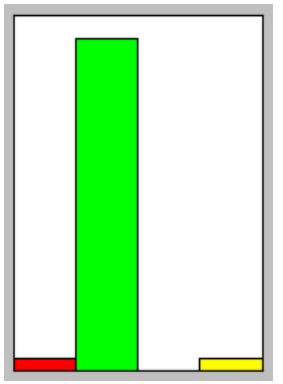
The ensemble model

- Usually, the class is found by averaging the “distributions” (the histograms) in the leaf nodes



The ensemble model

- Formally: Classification method (for new instances)
- Let $\mathbf{x}$ be the given (test) instance
- For each tree t, t=1,…,T:
  - Perform all tests along the interior nodes until the leaf is reached
  - Retrieve the histogram of the leaf (This represents the training data!)



$p(\text{red} \mid \mathbf{x})$
$p(\text{green} \mid \mathbf{x})$
$p(\text{blue} \mid \mathbf{x})$
$p(\text{yellow} \mid \mathbf{x})$

- Average all histograms to obtain a single overall histogram

$$p(c \mid \mathbf{x}) = \frac{1}{T} \sum_{t=1}^{T} p_t(c \mid \mathbf{x})$$

- The result (class) is obtained by computing the maximum of this averaged histogram:

$$c^* = \arg\max_c \, p(c \mid \mathbf{x})$$

The ensemble model - remarks

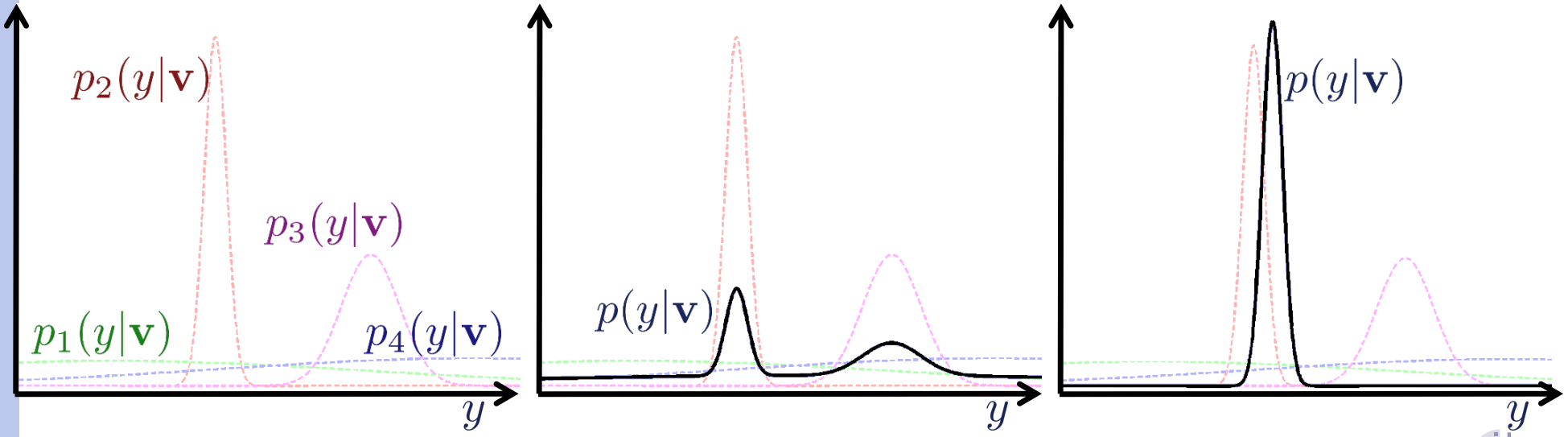
- Alternatively, one could use the product

$$p(c \mid \mathbf{x}) = \frac{1}{Z} \prod_{t=1}^{T} p_t(c \mid \mathbf{x})$$

(Z=partitioning function, for normalization)

- The product is more susceptible to noise
- Example (for continuous distributions):

$$p(y|\mathbf{v}) = \frac{1}{T} \sum_{t}^{T} p_t(y|\mathbf{v}) \qquad p(y|\mathbf{v}) = \frac{1}{Z} \prod_{t}^{T} p_t(y|\mathbf{v})$$



[Criminisi et al. 2011]

The ensemble model - remarks

- Note: Randomization takes place only during training, not when new instances (to be classified) are tested:
  - Training is randomized
  - Test is completely deterministic
- During testing (application), each tree is traversed independently according to the attributes of $\mathbf{x}$
  - Easy to parallelize on contemporary CPUs or GPUs.

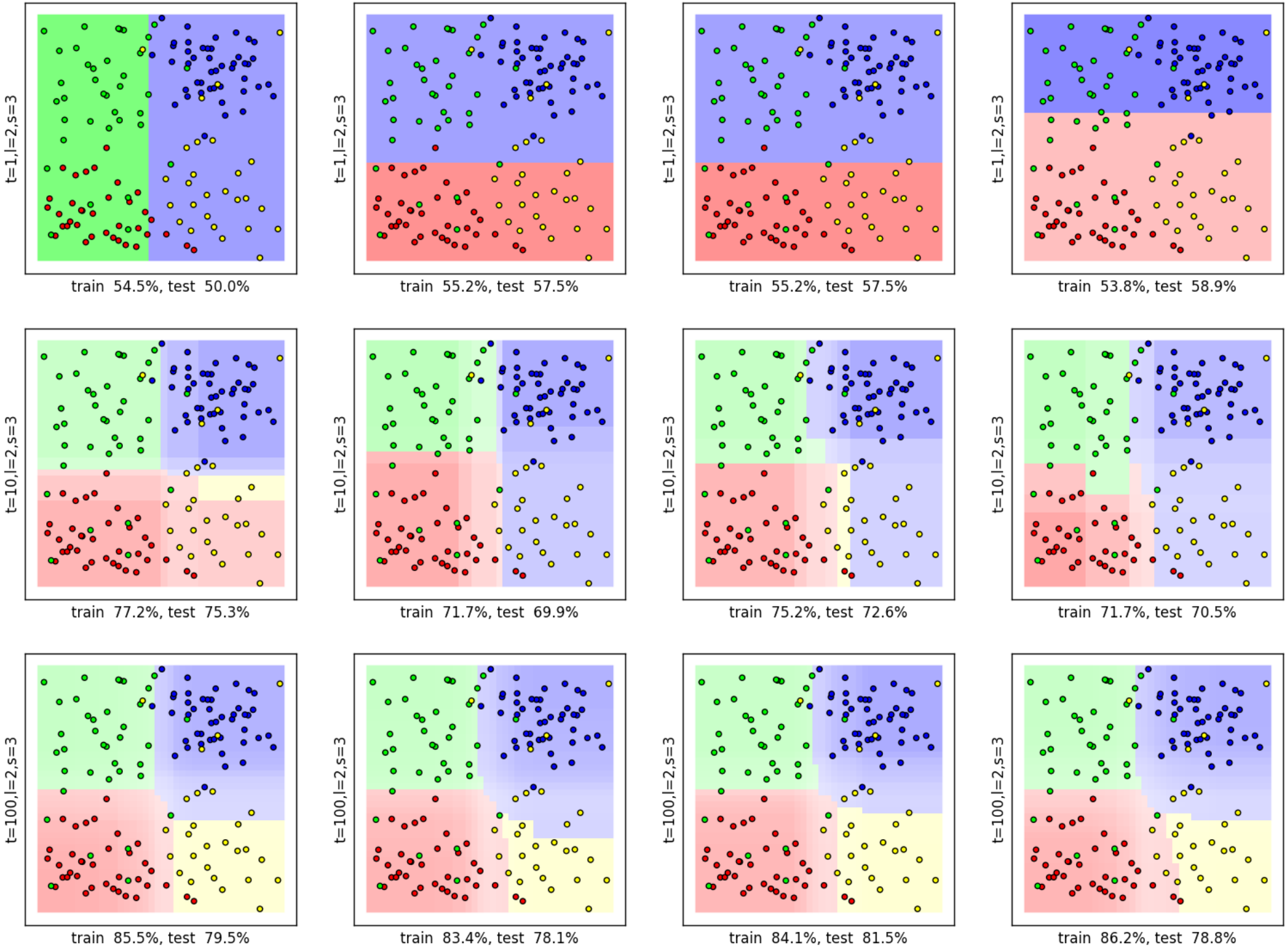
The ensemble model - remarks

- By computing the entropy of the averaged histogram, one obtains a “confidence” value for the classification result
  - This is in contrast to other classification methods, e.g. SVM, which do not (directly) deliver a confidence value
  - Reasoning: if many trees agree on one class, then the averaged histogram will have a clear peak, in which case the entropy will be small.

Example

- 1, 10, 100 trees
- All trees have maximum depth 2 (including root)
- Randomization: 3 random picks along each axis
- Observations (following slides):
  - Remember the trees have maximum depth 2
  - Thus, they can distinguish **two classes**, at most
  - This can be seen in the first row of the plots (one tree)
  - When using 10 or 100 trees, the forest is nevertheless able to distinguish **four classes**!
  - Boundaries between regions appear, which are not axis parallel (even though all single trees generate only axis parallel boundaries)!
  - Event though tree depth is only 2, approximately 80% of correctly classified instances are reported during test.

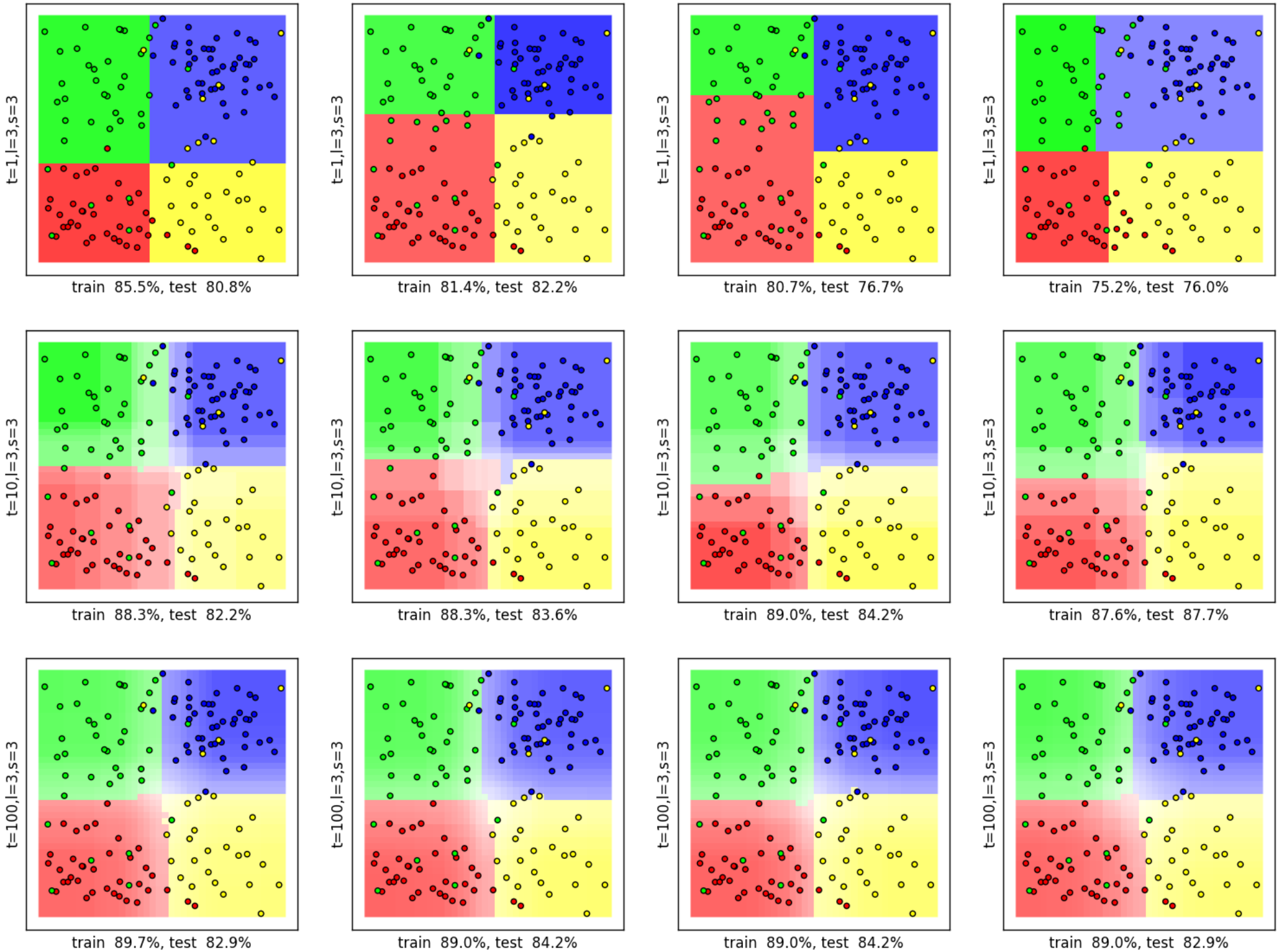
(Note the colors: saturated colors are used for small entropies.)


t=1,|=2,s=3
train 54.5%, test 50.0%
t=1,|=2,s=3
train 55.2%, test 57.5%
t=1,|=2,s=3
train 55.2%, test 57.5%
t=1,|=2,s=3
train 53.8%, test 58.9%
t=10,|=2,s=3
train 77.2%, test 75.3%
t=10,|=2,s=3
train 71.7%, test 69.9%
t=10,|=2,s=3
train 75.2%, test 72.6%
t=10,|=2,s=3
train 71.7%, test 70.5%
t=100,|=2,s=3
train 85.5%, test 79.5%
t=100,|=2,s=3
train 83.4%, test 78.1%
t=100,|=2,s=3
train 84.1%, test 81.5%
t=100,|=2,s=3
train 86.2%, test 78.8%

Example

- Same as previous example, but now using tree depth 3

- Observations:
  - Now, every single tree is able to distinguish 4 classes
  - This yields a smaller entropy (saturated colors in the plot)
  - The test results are (when using more than one tree) *all above* 80%


t=1,l=3,s=3
train 85.5%, test 80.8%
t=1,l=3,s=3
train 81.4%, test 82.2%
t=1,l=3,s=3
train 80.7%, test 76.7%
t=1,l=3,s=3
train 75.2%, test 76.0%
t=10,l=3,s=3
train 88.3%, test 82.2%
t=10,l=3,s=3
train 88.3%, test 83.6%
t=10,l=3,s=3
train 89.0%, test 84.2%
t=10,l=3,s=3
train 87.6%, test 87.7%
t=100,l=3,s=3
train 89.7%, test 82.9%
t=100,l=3,s=3
train 89.0%, test 84.2%
t=100,l=3,s=3
train 89.0%, test 84.2%
t=100,l=3,s=3
train 89.0%, test 82.9%

Parameters

- The most important parameters of random forests:
  - Number of trees, *T*
  - Maximum tree depth
  - Randomization during training
    - Bagging and/or node optimization
  - Optimization function during training
    - We used: “information gain”
  - Selection of the single (“weak”) learning function
    - We used: “axis parallel subdivisions”
  - Selection of the features
    - How the feature vector is computed from the original data
    - (We assumed this has been done already).

Alternatives for weak classifiers

- So far: **axis parallel** lines:

$$x_i < \text{const}$$

- **Arbitrarily oriented lines** (or, hyperplanes):

$$ax_1 + bx_2 + c < 0 \qquad \langle \mathbf{n}, \mathbf{x} \rangle + c < 0$$

- **Quadratic functions** (conic sections):

$$\begin{bmatrix} x_1, x_2, 1 \end{bmatrix} \begin{bmatrix} a & b/2 & d/2 \\ b/2 & c & e/2 \\ d/2 & e/2 & f \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ 1 \end{bmatrix} < 0$$

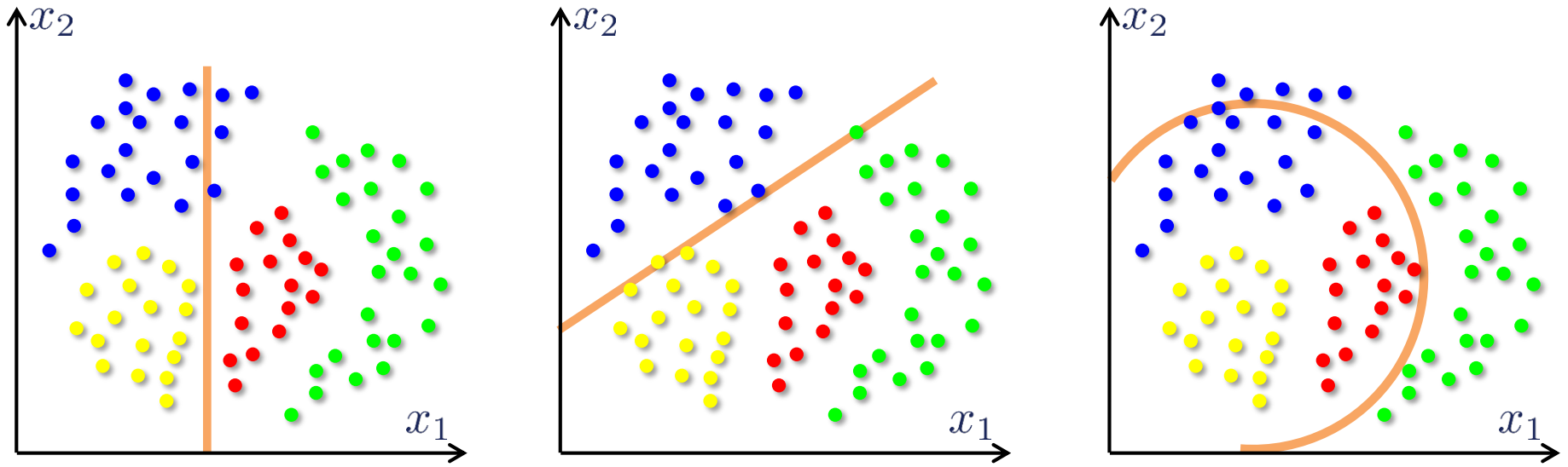
$$ax_1^2 + bx_1x_2 + cx_2^2 + dx_1 + ex_2 + f < 0$$

In general:

$$\mathbf{x}^T \mathbf{M} \mathbf{x} < 0$$

Alternatives for weak classifiers

- Axis parallel, arbitrarily oriented, conic section



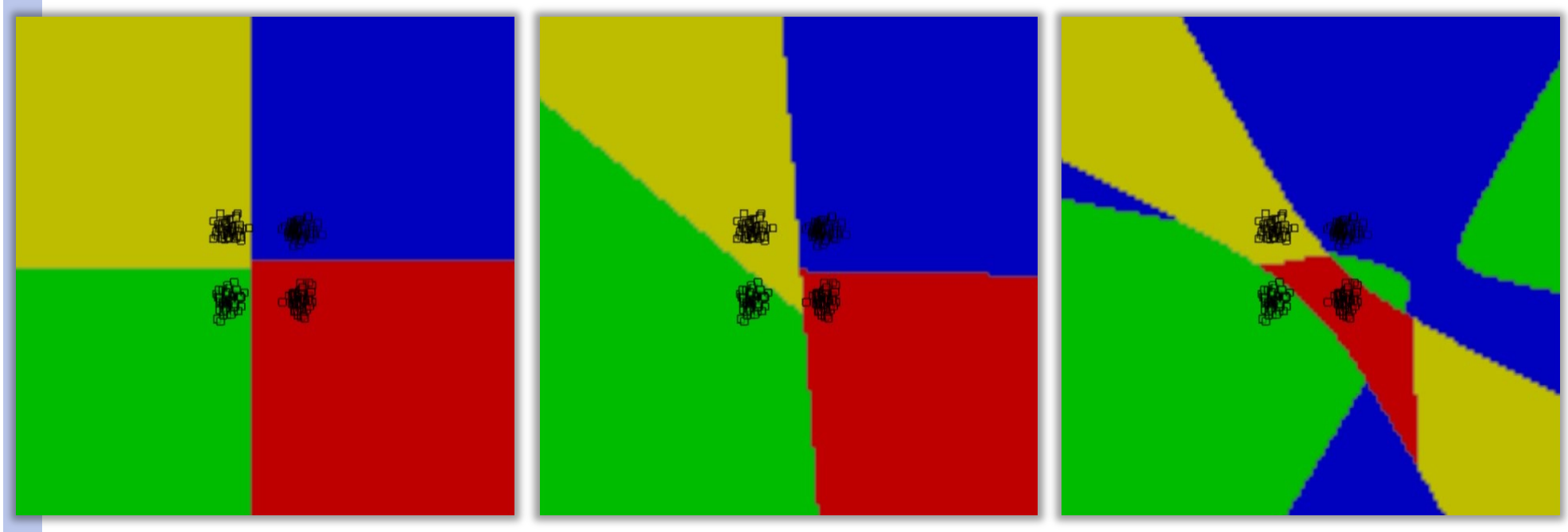
[Criminisi et al. 2011]

Alternatives for weak classifiers

- Note:
  - The node optimization usually applies the weak classifiers only on low dimensional subspaces
  - E.g.:
    - Feature vector has dimension 200
    - In one node of the tree, three dimensions are selected
    - The selection of the best hyperplane takes place in this three-dimensional space (rather than in 200 dimensional space)

Example: 4 classes

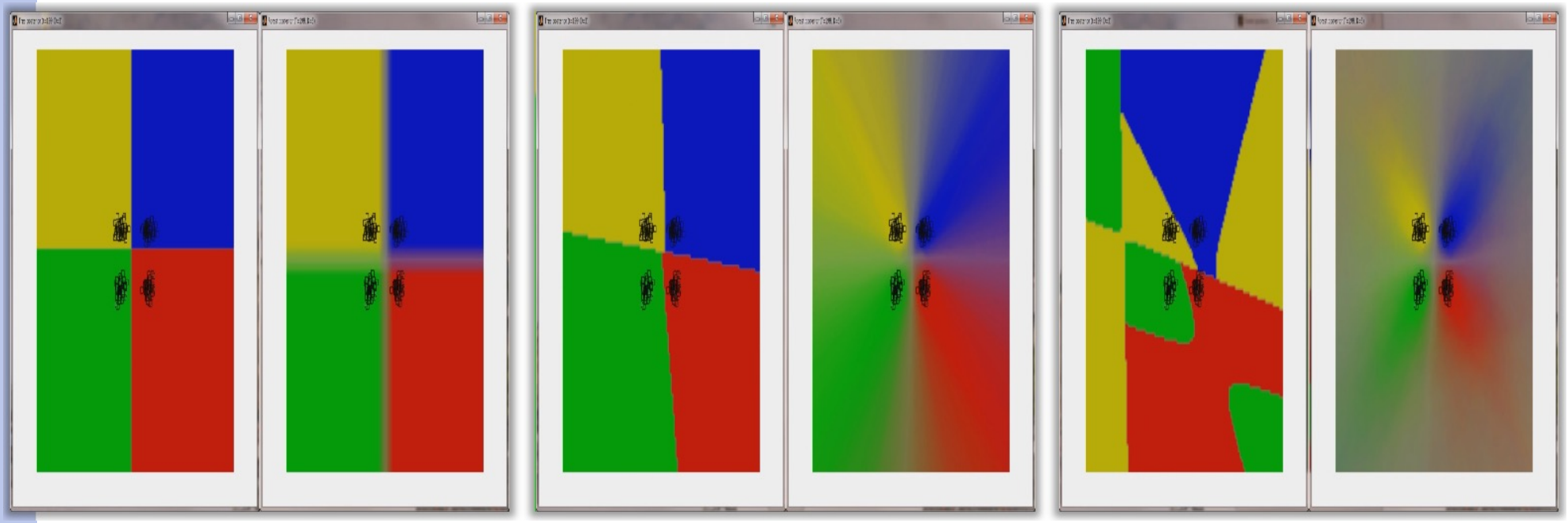
- Axis parallel, arbitrarily oriented, conic section
- Images show one tree
- Animation shows evolution of the forest



[Criminisi et al. 2011]

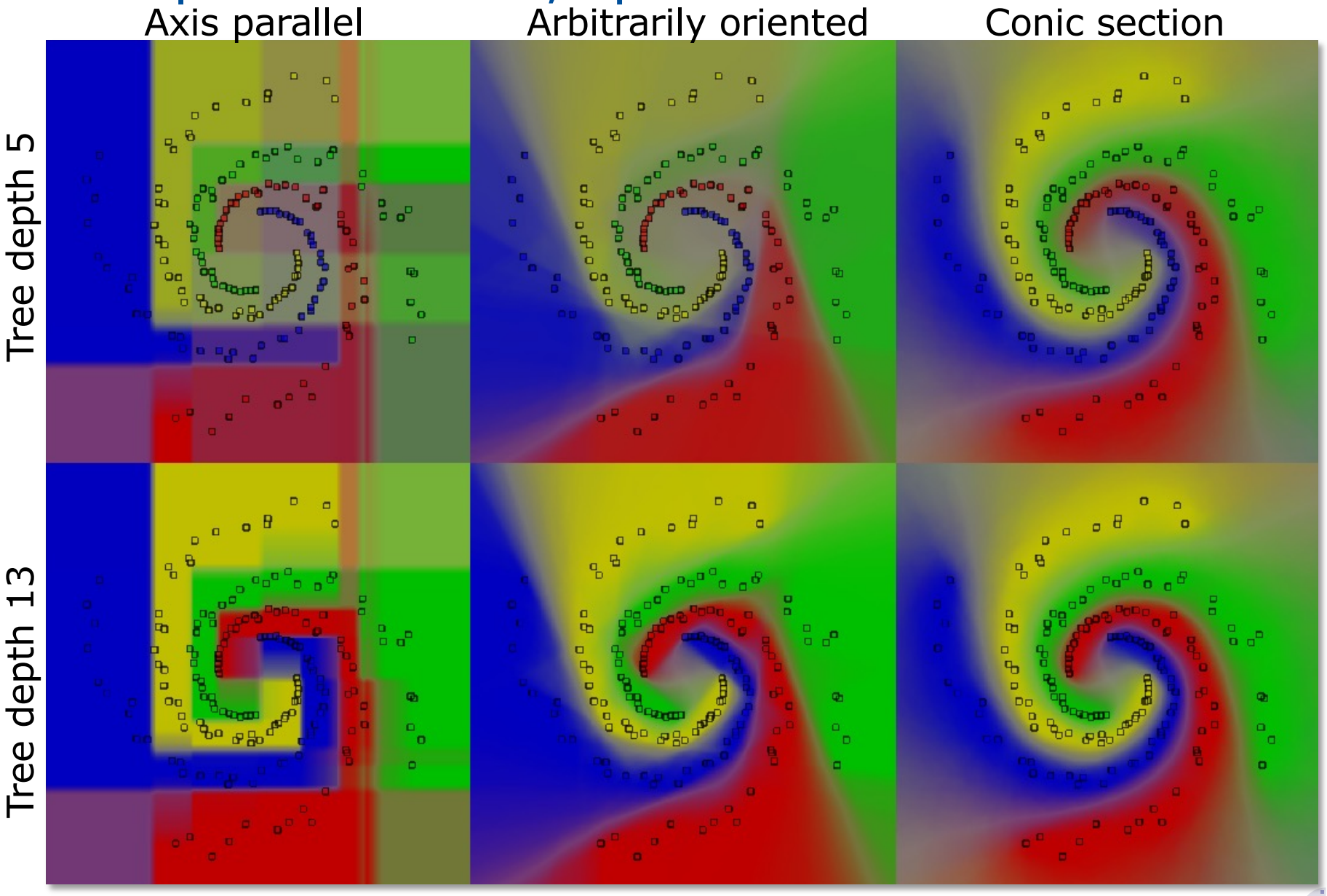
Example: 4 classes

- Axis parallel, arbitrarily oriented, conic section
- Images show one tree
- Animation shows evolution of the forest



[Criminisi et al. 2011]

Example: 4 classes, spiral

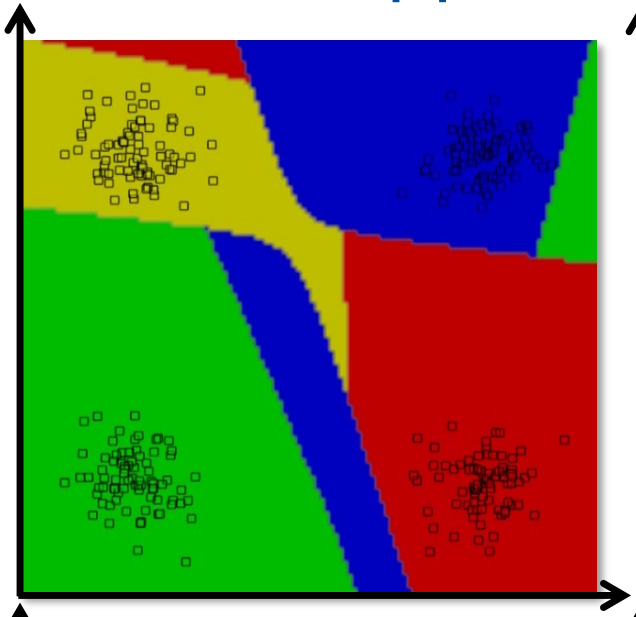


[Criminisi et al. 2011]

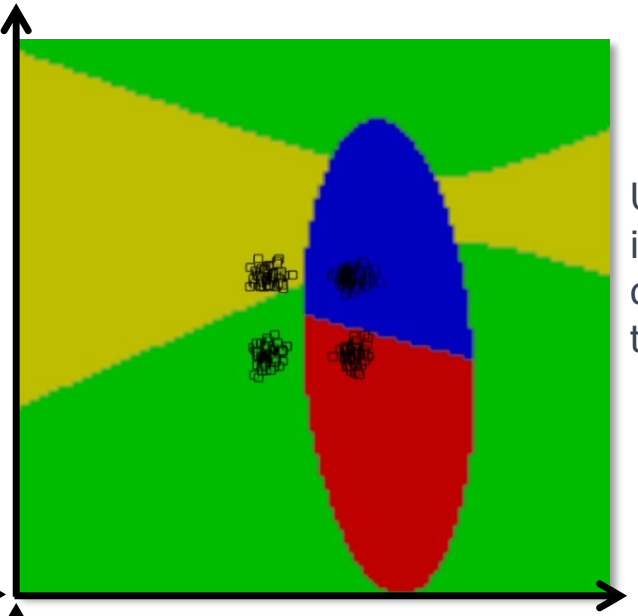
(400 trees)

Comparison to Support Vector Machines (SVM)

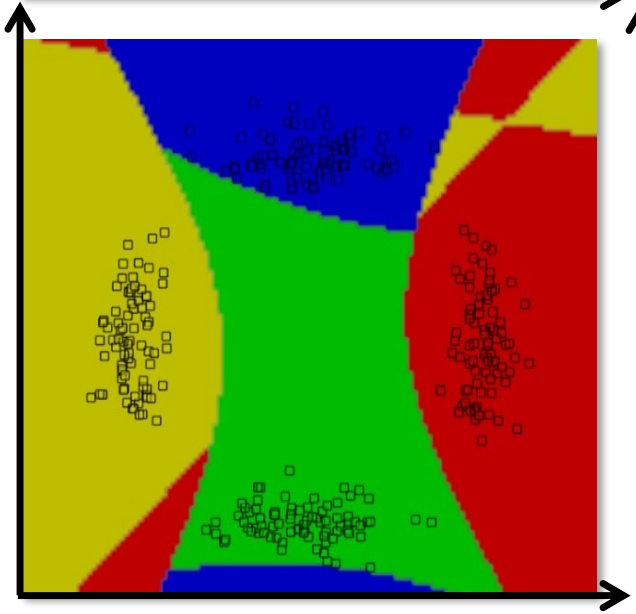
Finds maximum margin – see SVM.



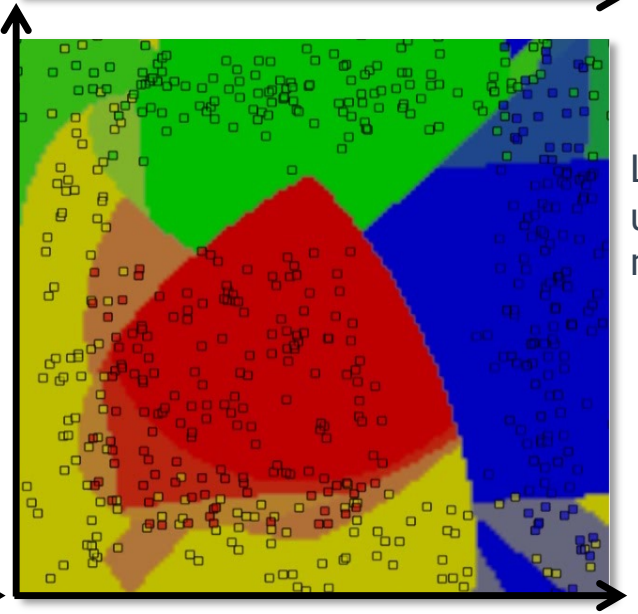
Uncertainty increases with distance from the training data



Finds maximum margin – see SVM.



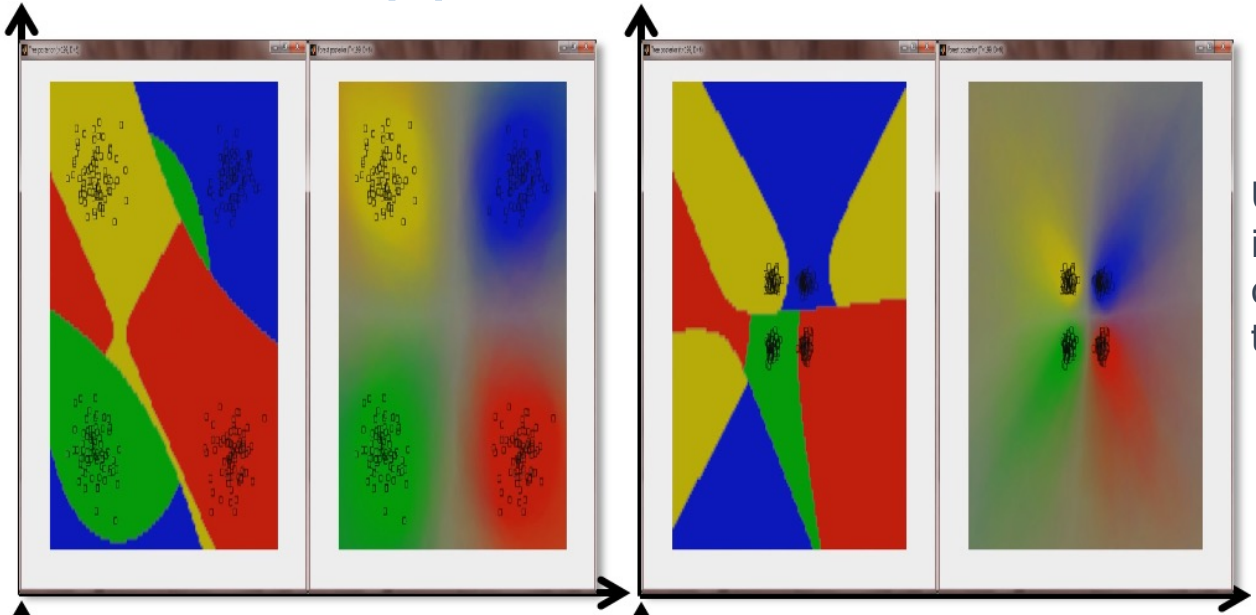
Larger uncertainty in mixed regions



[Criminisi et al. 2011]

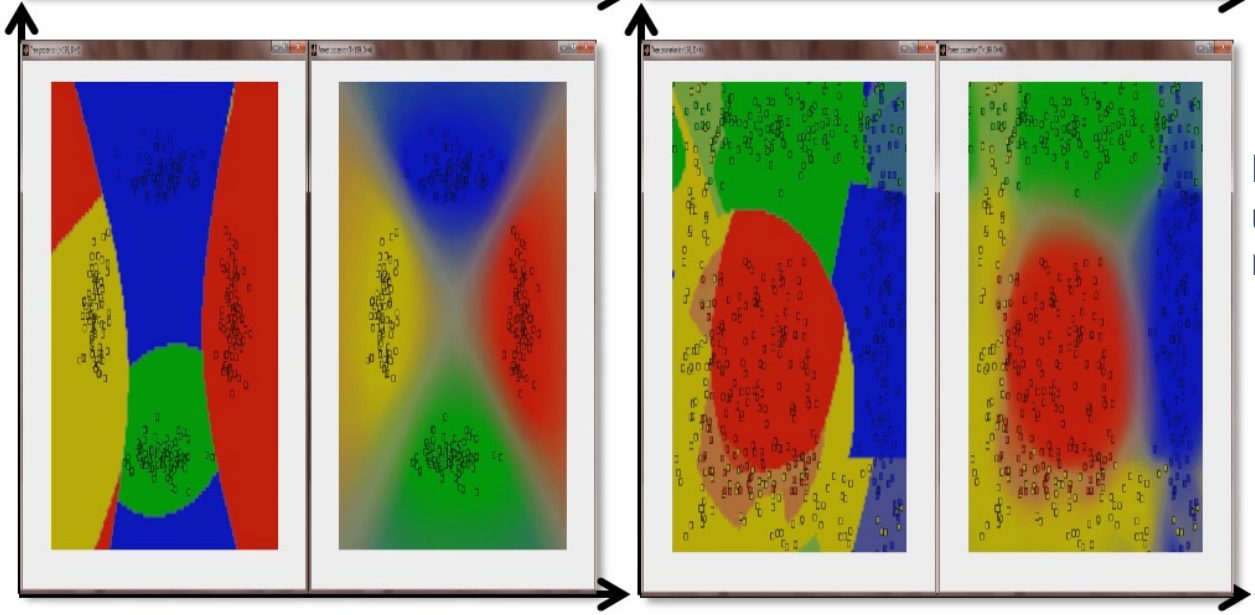
Comparison to Support Vector Machines (SVM)

Finds maximum margin – see SVM.



Uncertainty increases with distance from the training data

Finds maximum margin – see SVM.

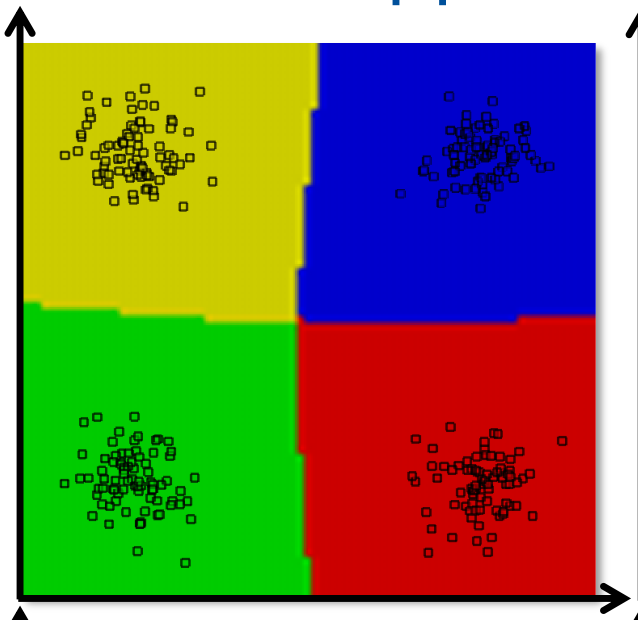


Larger uncertainty in mixed regions

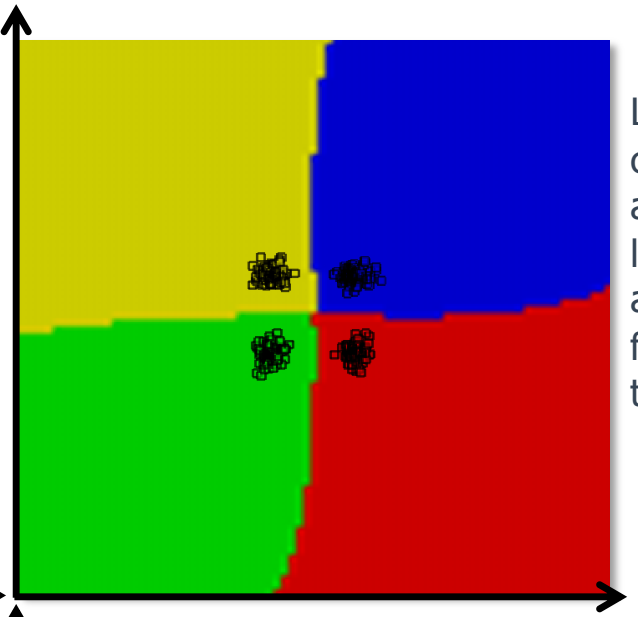
[Criminisi et al. 2011]

Comparison to Support Vector Machines (SVM)

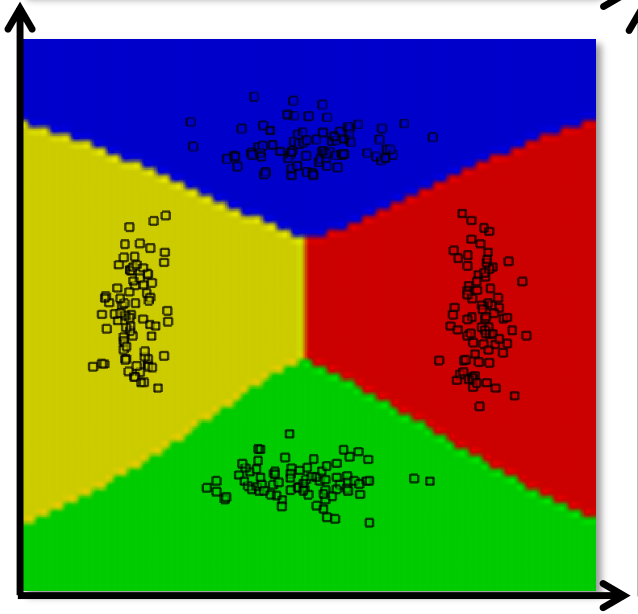
SVM does not compute confidence value



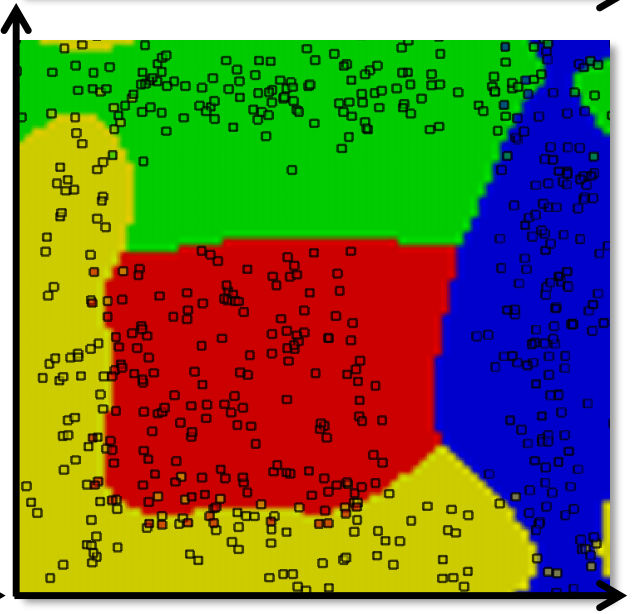
Large confidence, also at locations which are far away from the training data



No symmetry (follows from one-against-all approach)



Overfitting and high confidence

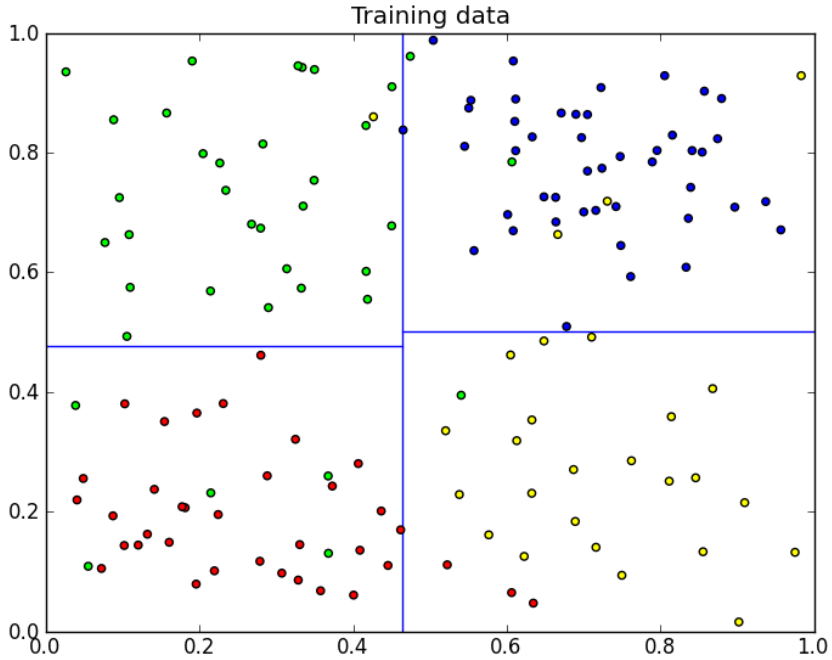


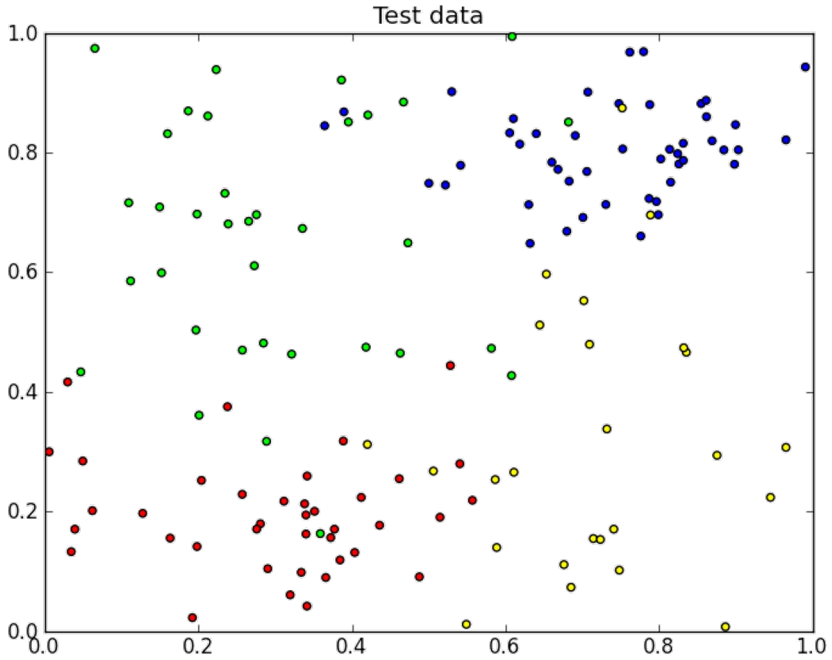
[Criminisi et al. 2011]

Evaluation of classification results

When the distribution is known

- In the previous lecture:
  - Draw points from known distributions → Training
  - Draw again points from known distributions → Test





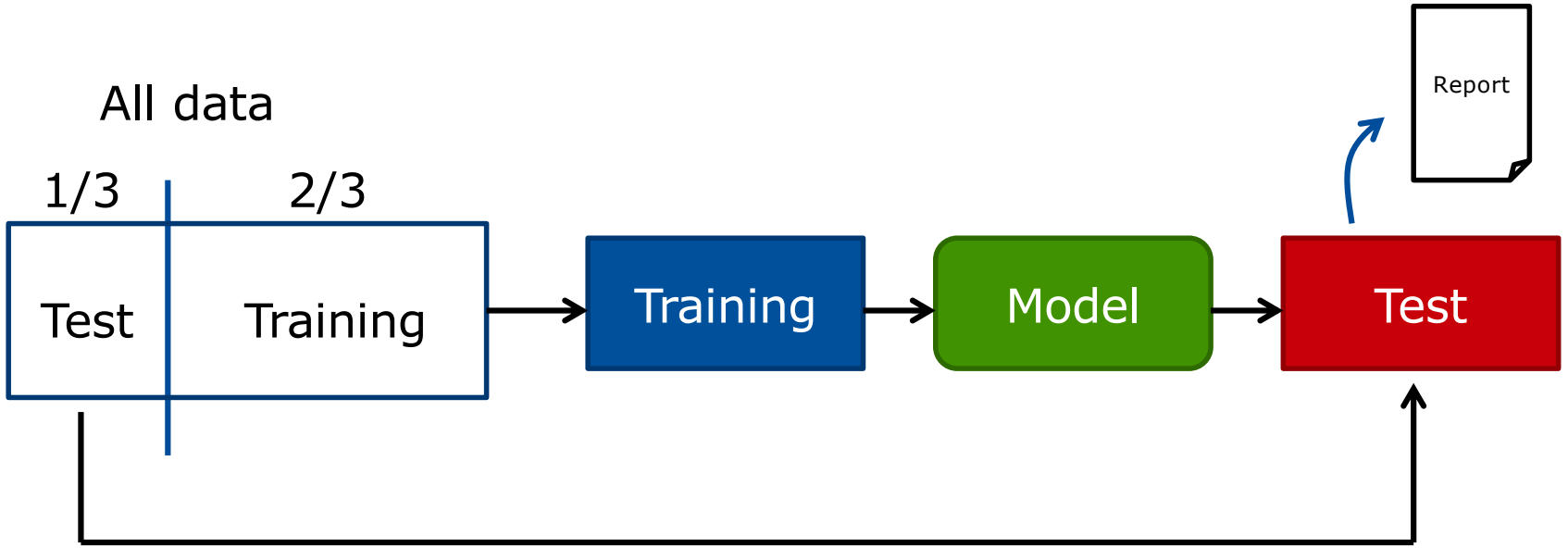
When the distribution is unkown

- Normally, the distribution of the classes is unknown and has to be “discovered” by the classifier

- In a supervised classification, the training data is given
  - E.g. by manually assigning correct classes (“ground truth”)
- Manually classified data is precious!

Method 1: hold back

- Method 1: hold back part of the data for testing
  - E.g. 2/3 training, 1/3 test

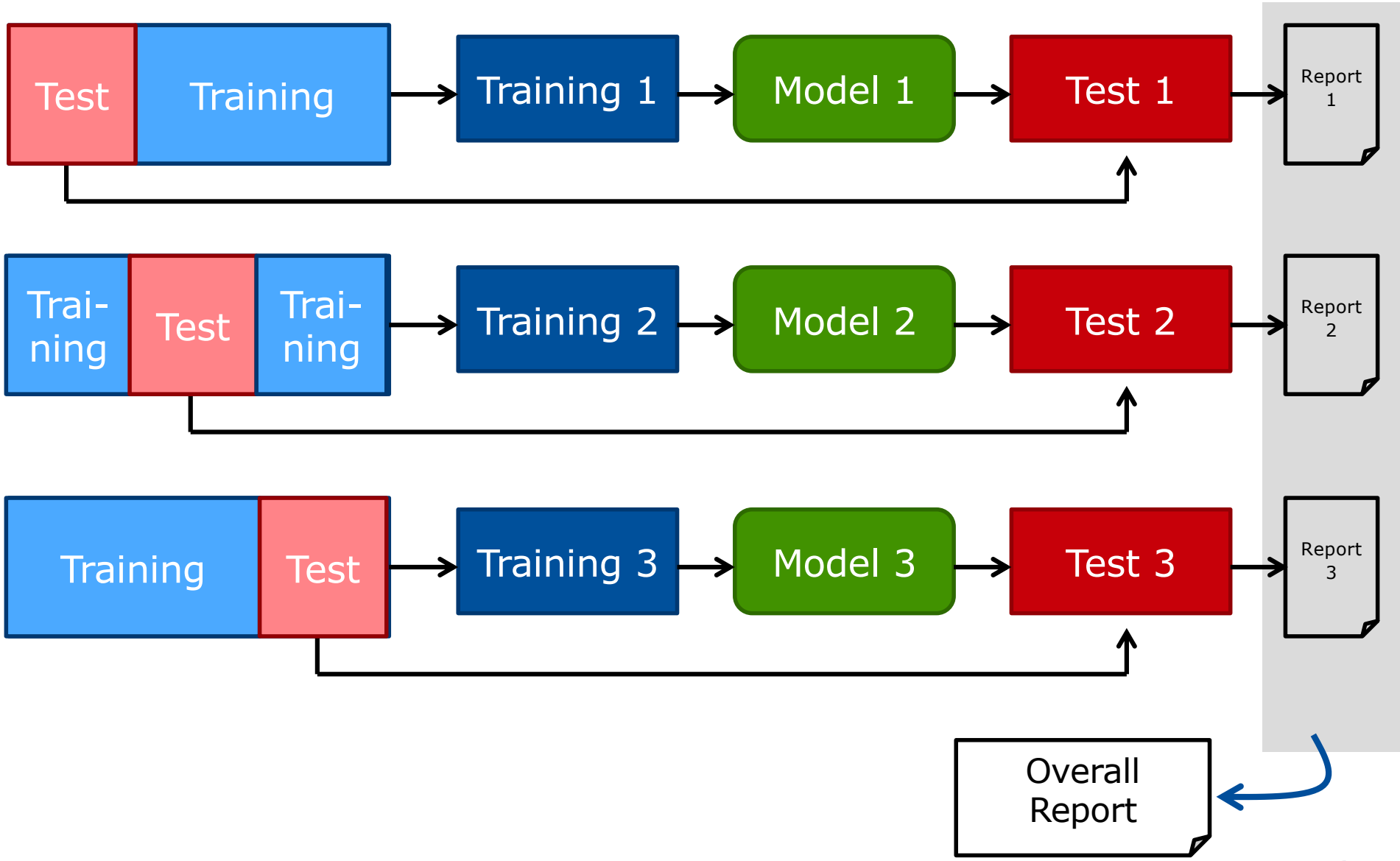


- Disadvantage:
  - Training data precious → use as little test data as possible
  - Selection of test data/ training data must be representative
    - (e.g. test data should not all be from the "blue" class)

Method 2: cross validation

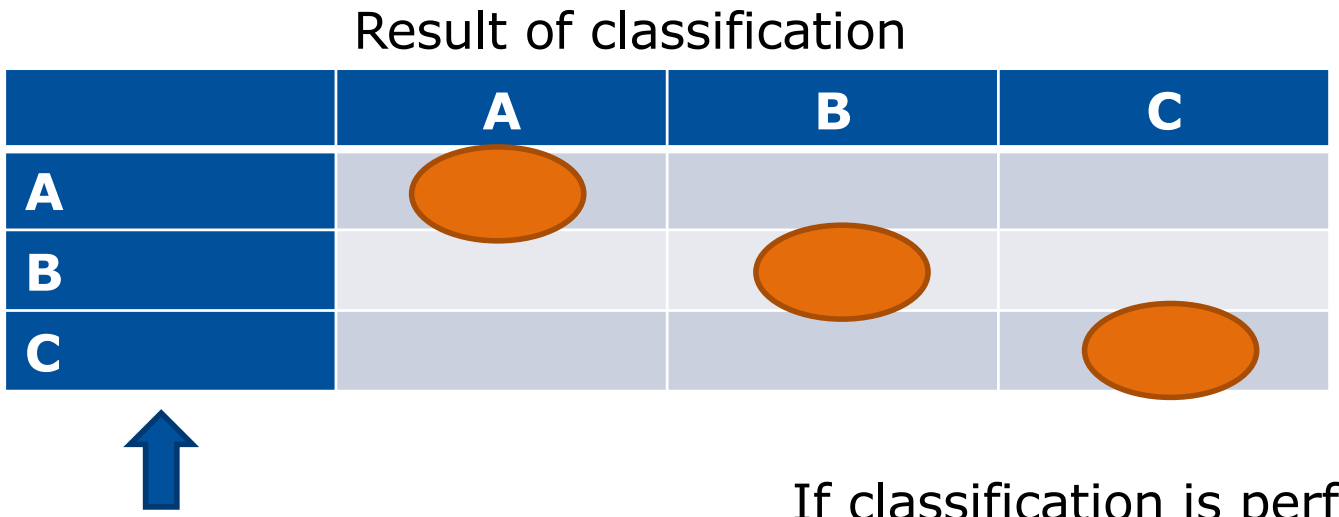
- Repeat procedure 3x with different choice of the test data set
- Average test results
- This is known as “three-fold cross validation”
- Often used: 10-fold cross validation.

Method 2: cross validation



Evaluation of the performance

- Count / percentage of correctly classified instances
- Confusion matrix



If classification is perfect, nonzero values will be only along the main diagonal

Example (Weka):

```
=== Confusion Matrix ===

     a     b     c     d   <-- classified as
     0     0     0     0 |     a = ?
     0  3346     0     0 |     b = G
     0     0 15929     0 |     c = B
     0     0     0  9166 |     d = C
```

True positive, false positive, …

- Based on the confusion matrix, performance indices can be defined

Result of classification

Truth	A	not A
A	TP	FN
not A	FP	TN

TP – True positive (correctly classified as A)
FP – False positive (classified as A, but is another class)
FN – False negative (not classified as A, although from A)
TN – True negative (correctly classified as being not from A)

Precision and recall

Result of classification

Truth	A	not A
A	TP	FN
not A	FP	TN

$$\text{Precision} = \frac{\text{TP}}{\text{TP} + \text{FP}}$$

- "How many of the instances classified as A are correct?"
- "How many of the A that were found are correct"
- In the ideal case: precision = 1.

Precision and recall

		Result of classification: A	Result of classification: not A
Truth	A	TP	FN
	not A	FP	TN

$$\text{Recall} = \frac{\text{TP}}{\text{TP} + \text{FN}}$$

- "How many of the A instances were correctly classified?"
- "How many of the A elements in the dataset were found"
- In the ideal case: recall = 1.

F-measure

- Also F-score, F_1 score
- Harmonic mean of precision and recall:

$$F_1 = \left(\frac{\text{Precision}^{-1} + \text{Recall}^{-1}}{2} \right)^{-1}$$

- Value between 0 and 1 (as Precision and Recall)
- Usually, there is a trade-off between Precision and Recall
- F-measure includes both in a symmetric manner.

Example of WEKA output

```
=== Evaluation result ===

Scheme: RandomForest
Options: -I 10 -K 0 -S 1 -depth 3
Relation: laser_features-weka.filters.unsupervised.attribute.Remove-R1-5


Correctly Classified Instances      137291               98.1814 %
Incorrectly Classified Instances      2543                1.8186 %
Kappa statistic                          0.9696
Mean absolute error                      0.0341
Root mean squared error                  0.0991
Relative absolute error                 11.3798 %
Root relative squared error             25.6183 %
Total Number of Instances           139834

=== Detailed Accuracy By Class ===

               TP Rate   FP Rate   Precision   Recall  F-Measure   ROC Area  Class
                 0         0          0         0         0          ?        ?
                 0.935     0.011      0.925     0.935     0.93       0.997    G
                 1         0          1         1         1          1        B
                 0.975     0.014      0.978     0.975     0.976      0.999    C
Weighted Avg.    0.982     0.007      0.982     0.982     0.982      0.999

=== Confusion Matrix ===

     a     b     c     d   <-- classified as
     0     0     0     0 |     a = ?
     0 16799     2  1172 |     b = G
     0     1 68001     0 |     c = B
     0  1368     0 52491 |     d = C
```

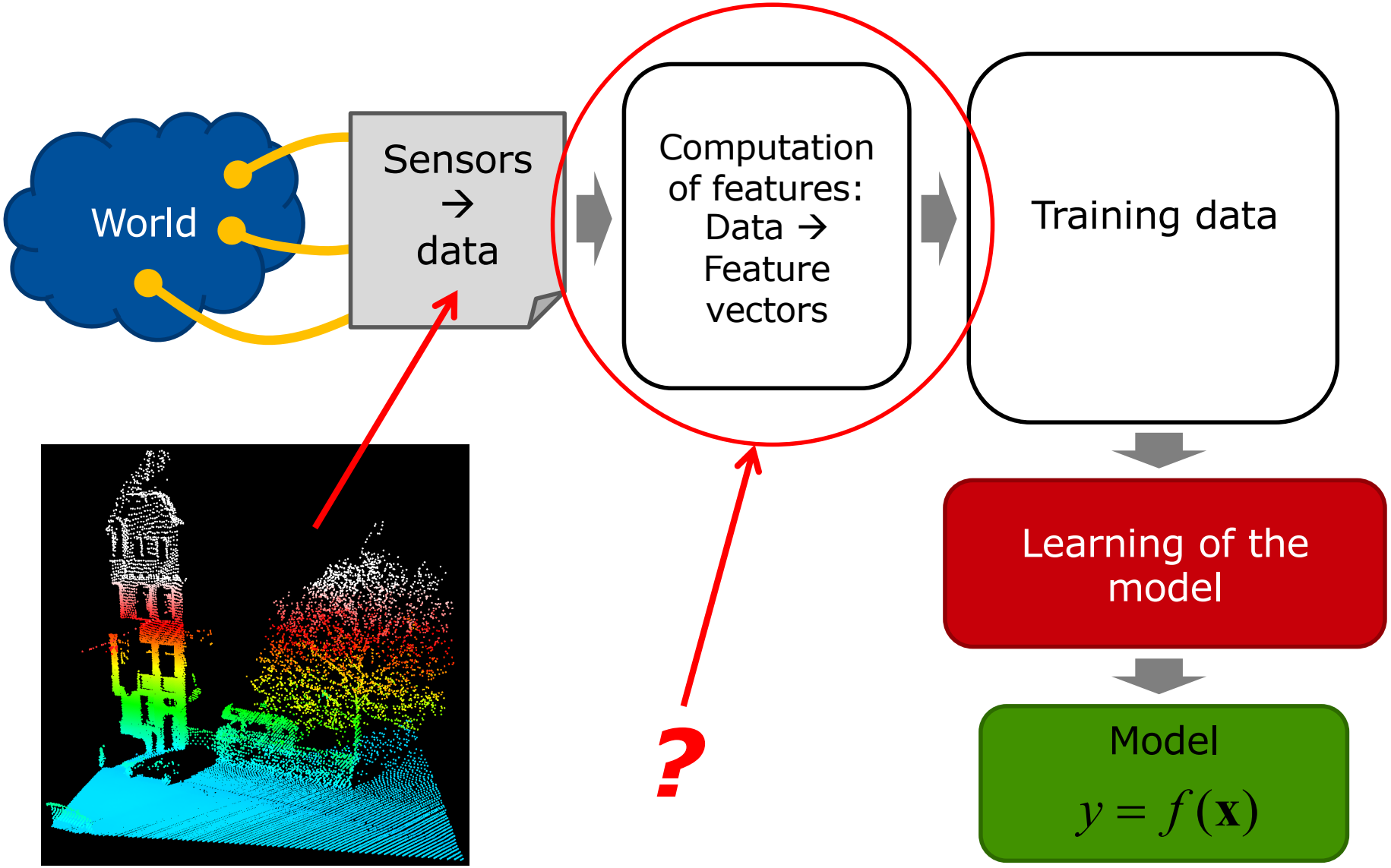
Correctly and incorrectly classified instances

TP, FP, Precision, Recall, F-measure

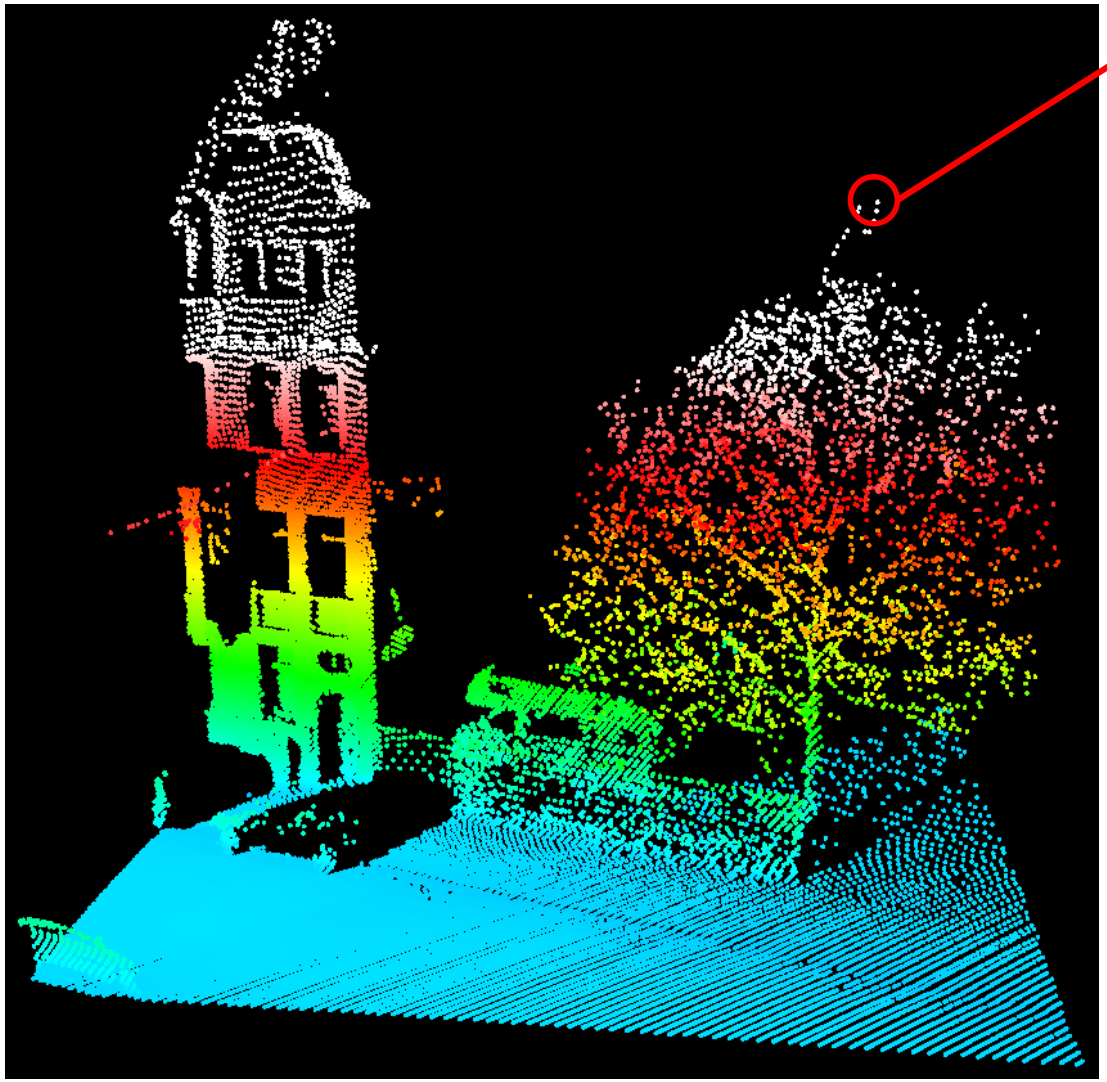
Confusion matrix

Computation of features from LiDAR data

Revisited: Computation of feature vectors



Attributes present in the original data

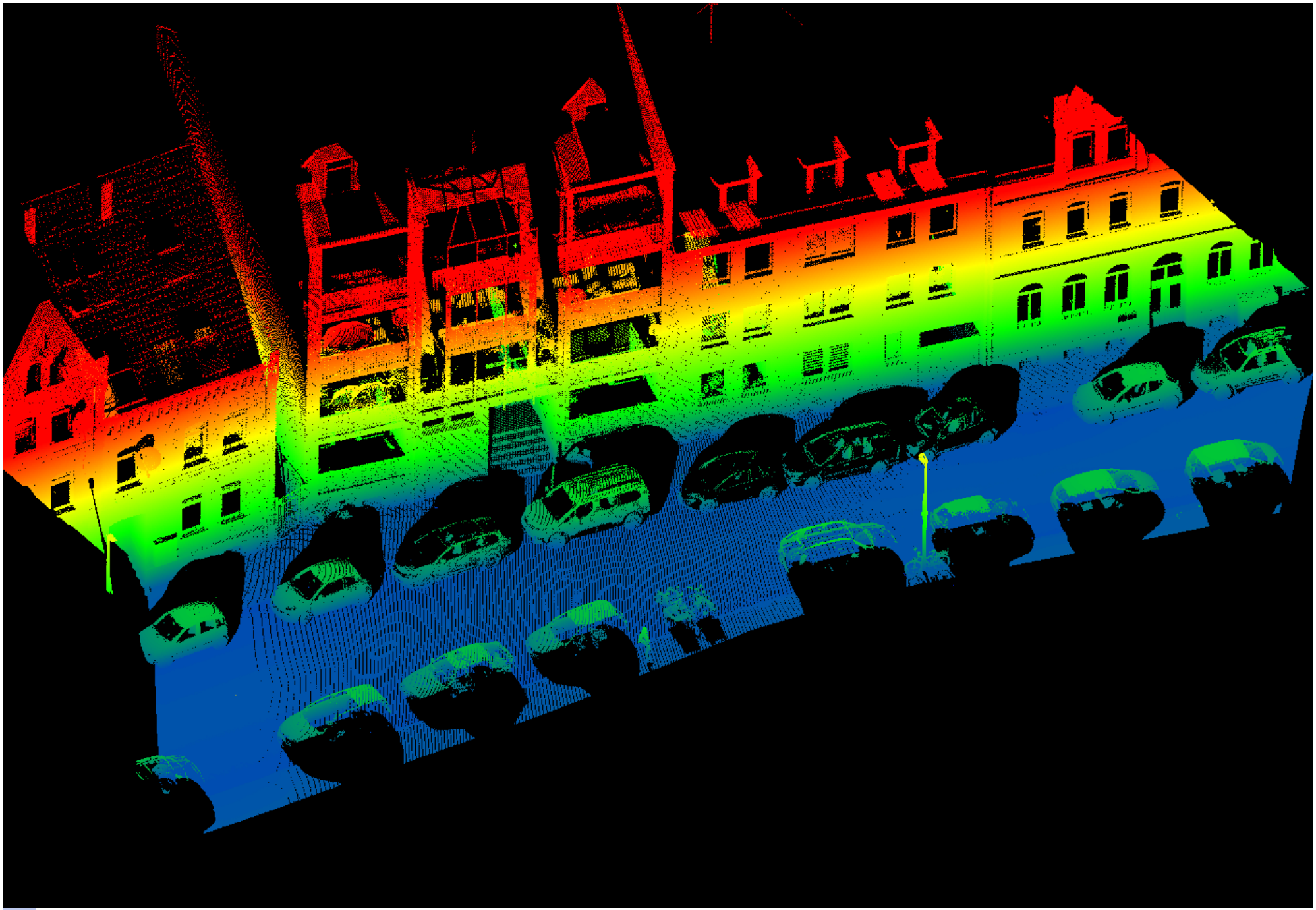


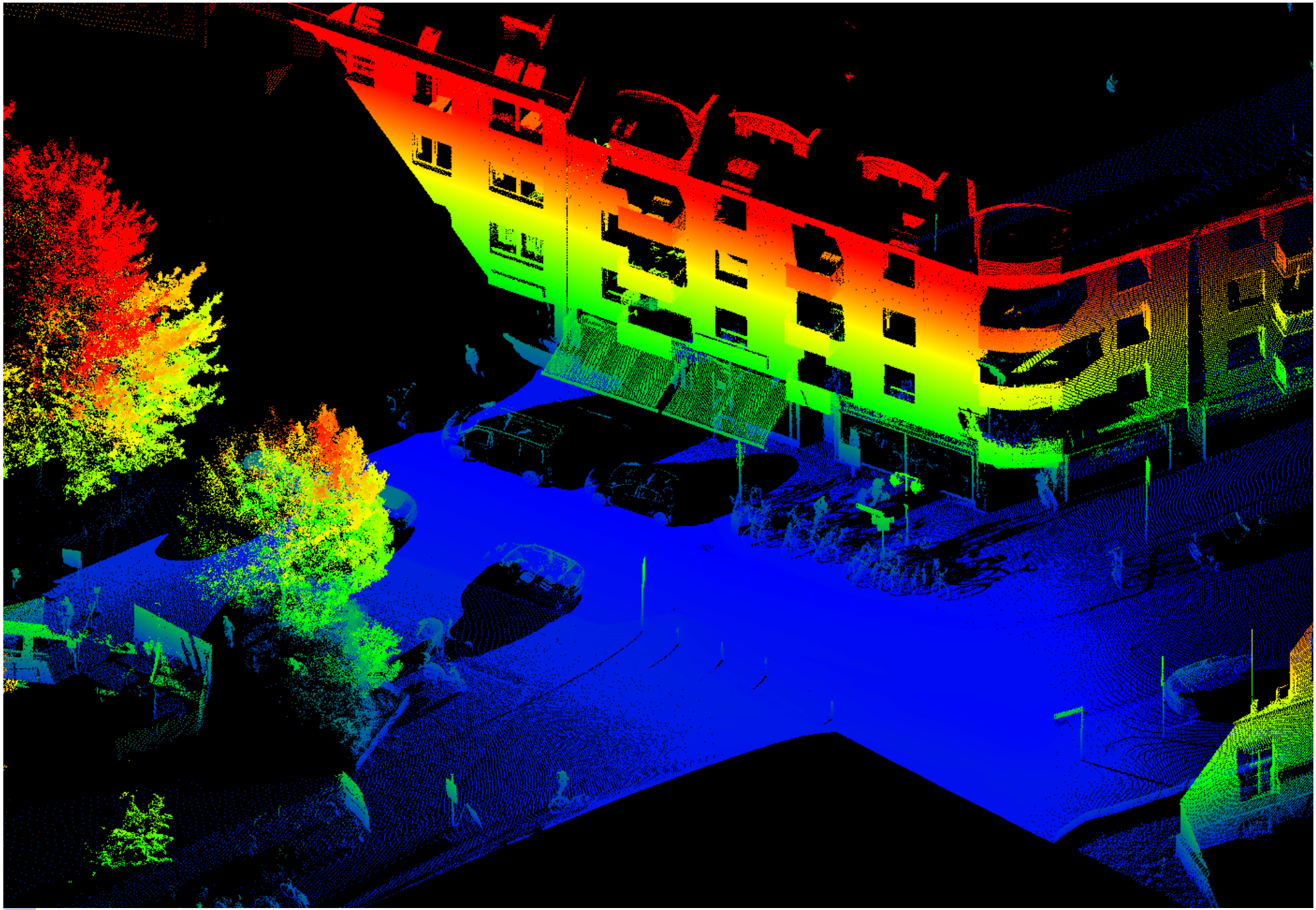
- Typically available:
  For each point
  - x, y, z (Cartesian)
  - Amplitude/
    reflectance
  - Number of echoes
  - Width of echo

Attributes: x, y, z

- x, y (position (in the plane))
- z (height)
- Direct usage of these attributes does not make sense
  - A classifier would learn the location of objects by heart
  - E.g. the position of trees

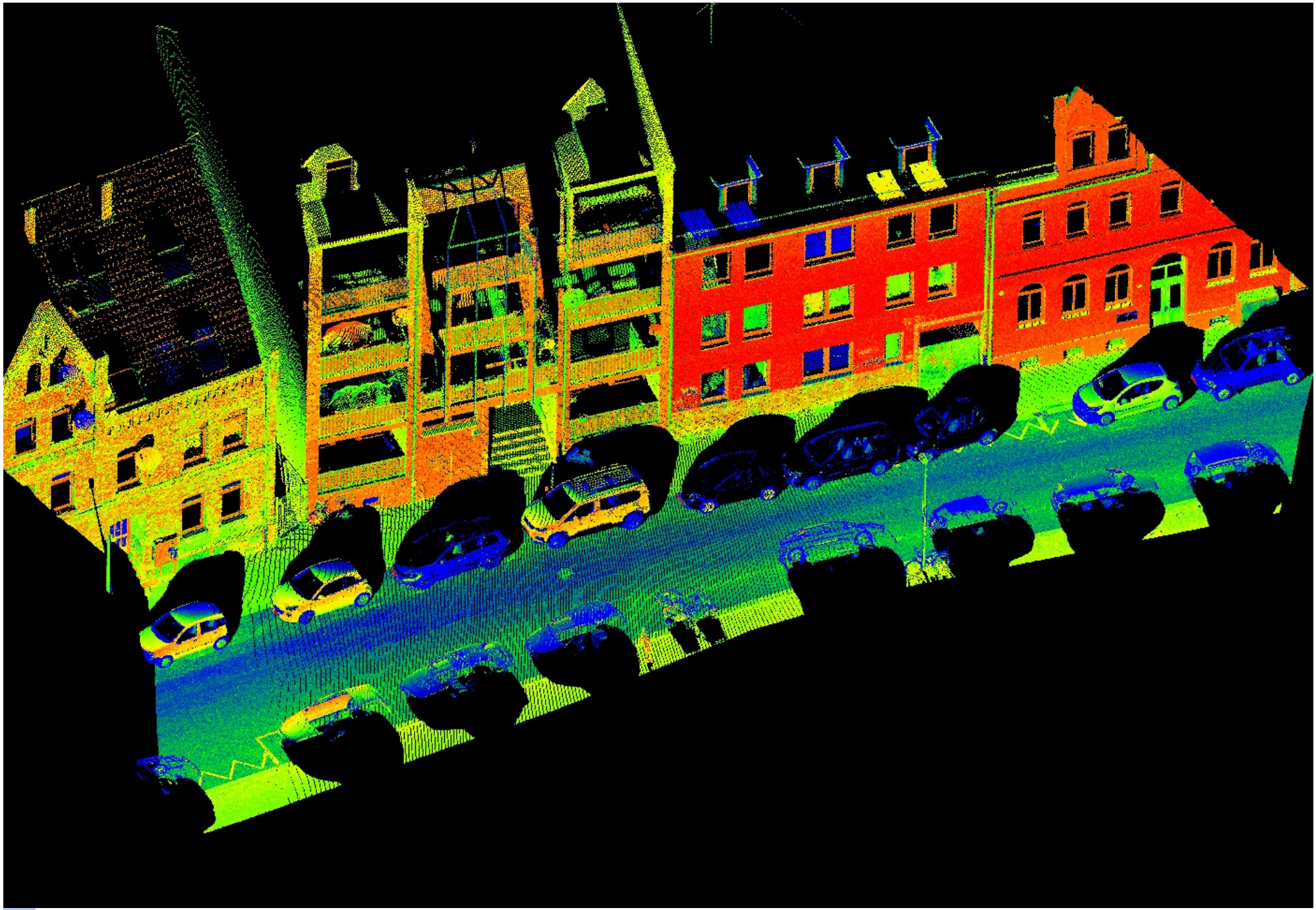
- But: the "**height above ground**" can make sense
  - E.g. tree crowns are in a certain height above ground
  - Definition of the local "ground" can be hard (ambiguous)
  - For mobile mapping, the "height of the road centerline" may be used.

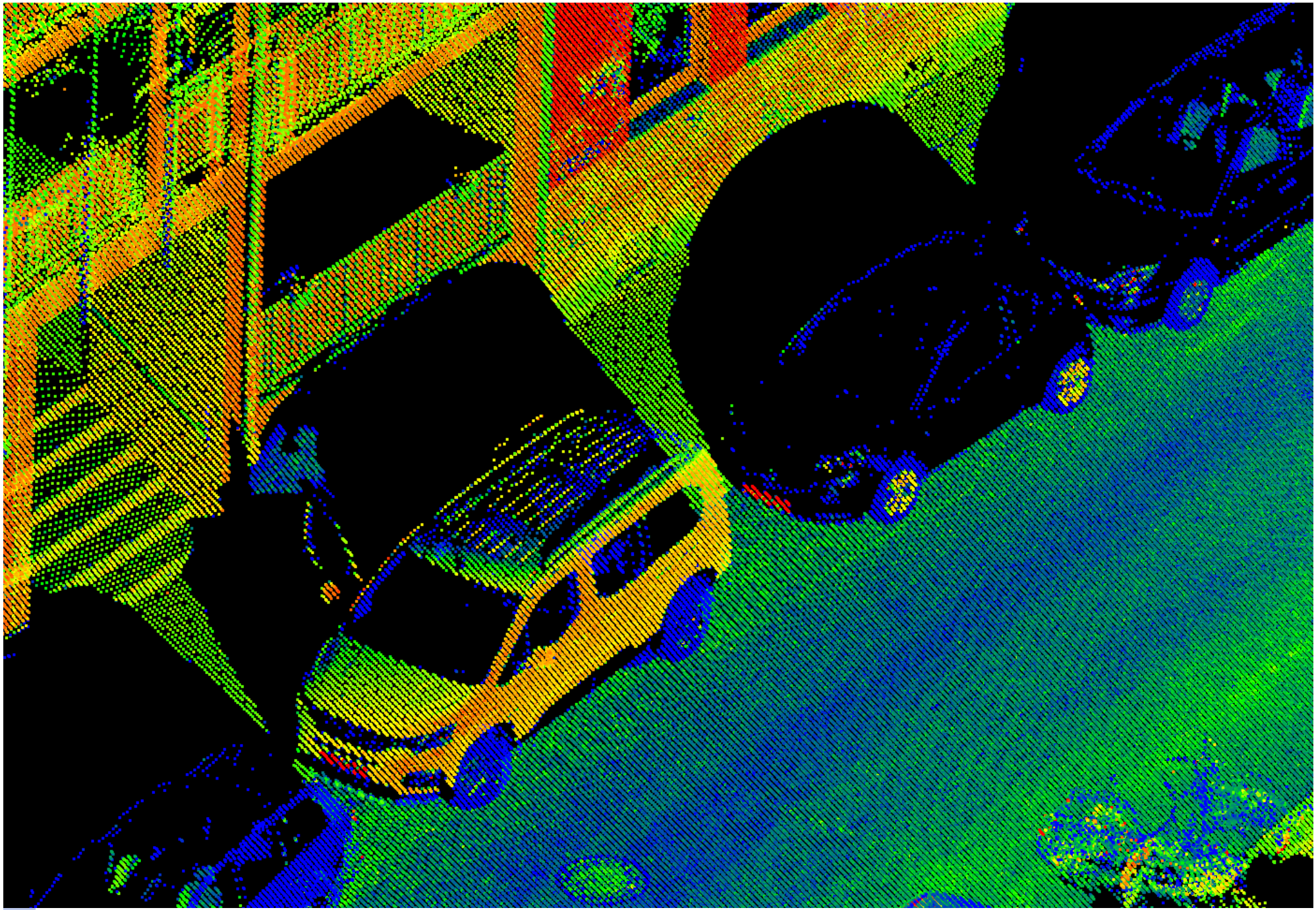


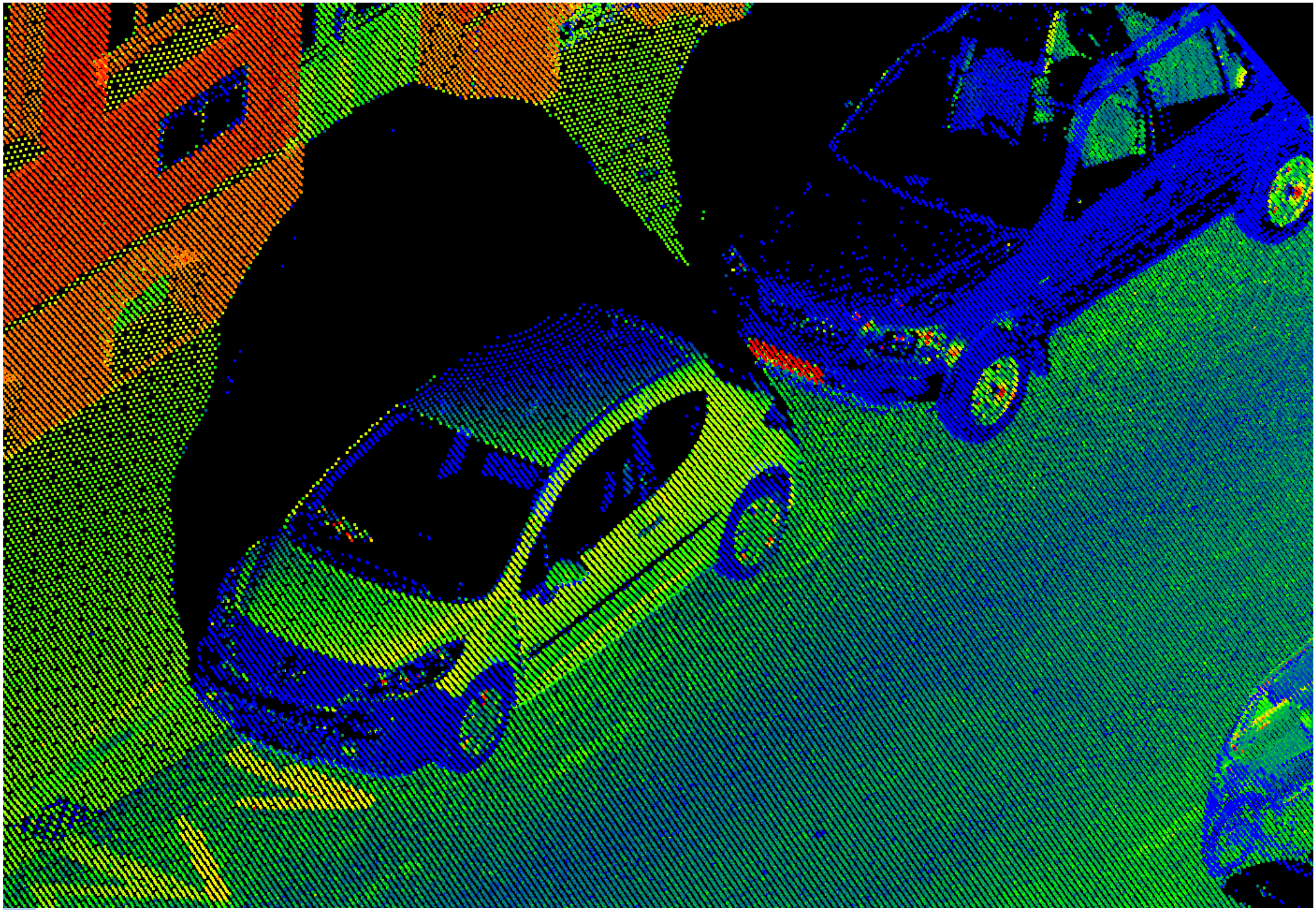


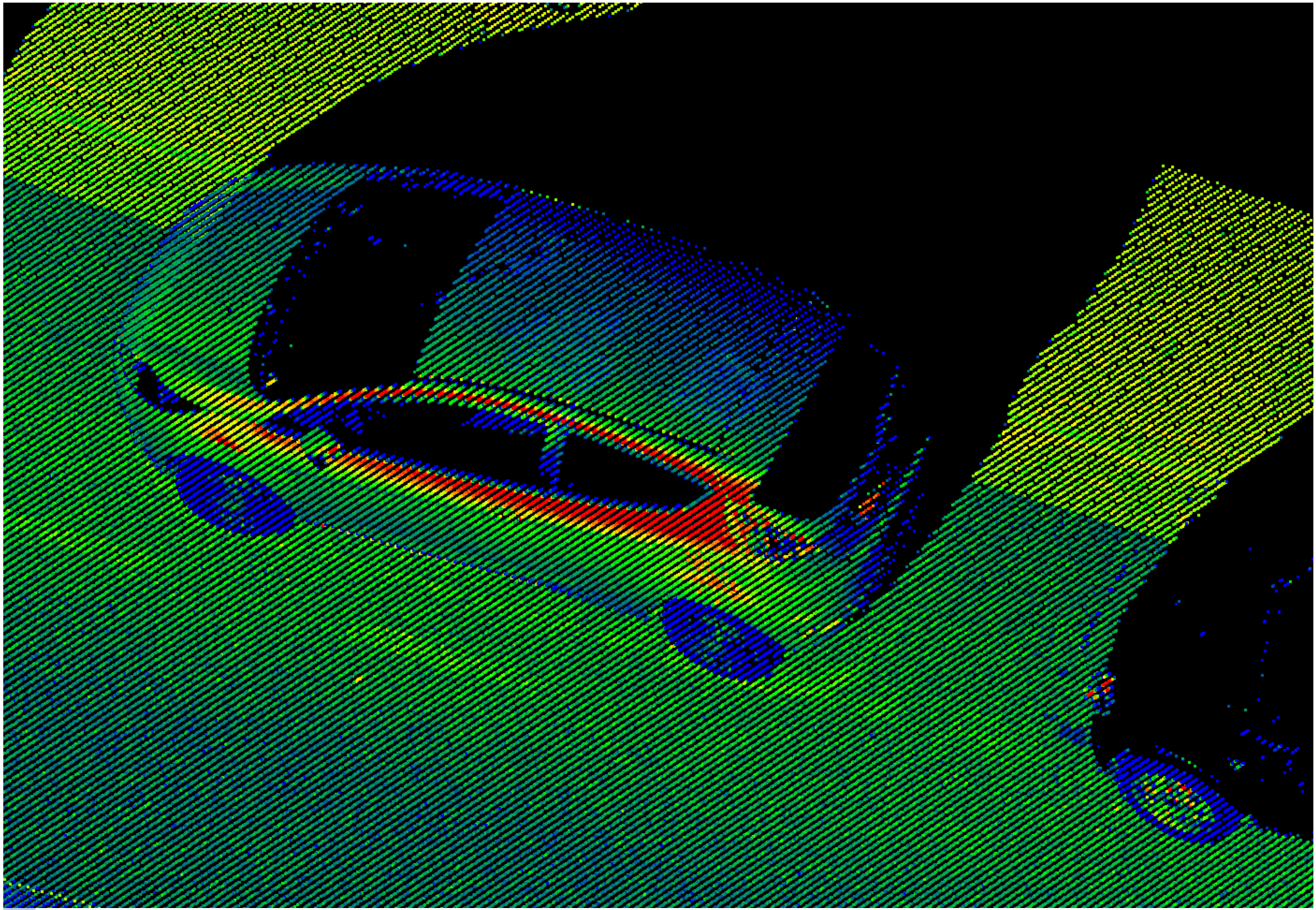
Attribute: “reflectance”

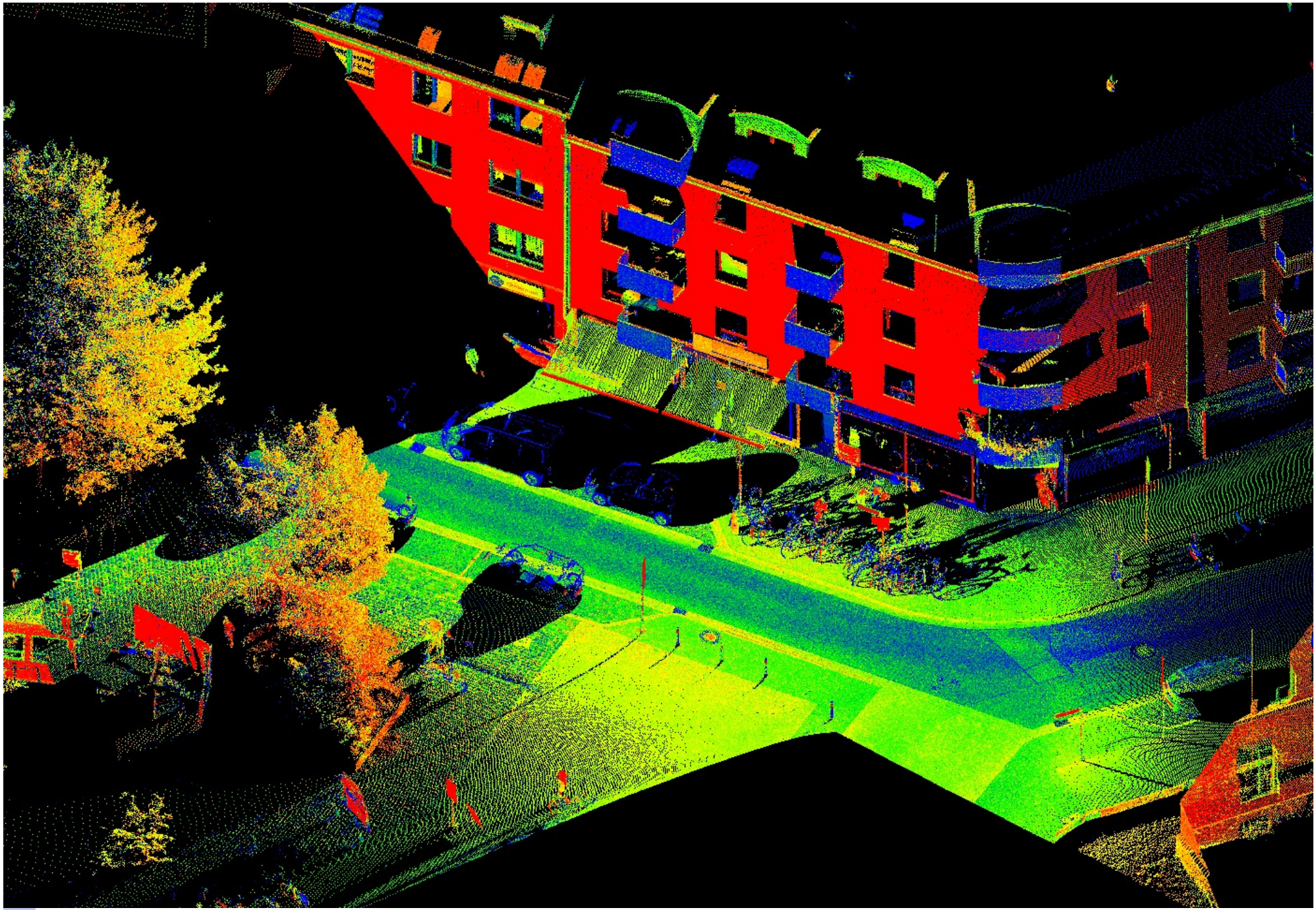
- Amplitude of the reflected laser pulse, corrected by distance
- Depends on the reflectivity of the object surface
- Road markings and street signs are easily visible
- Retro-reflective material (cat’s eye reflectors, reflective (plastic) sheets, reflective color)
- Façade structures can be discerned
  - But note: the intensity is different from a panchromatic image
- Cars may lead to largely varying results (depending on varnish)
  - Sometimes, points are only sparse
  - (Note that moving cars, as any moving objects, are geometrically distorted.)





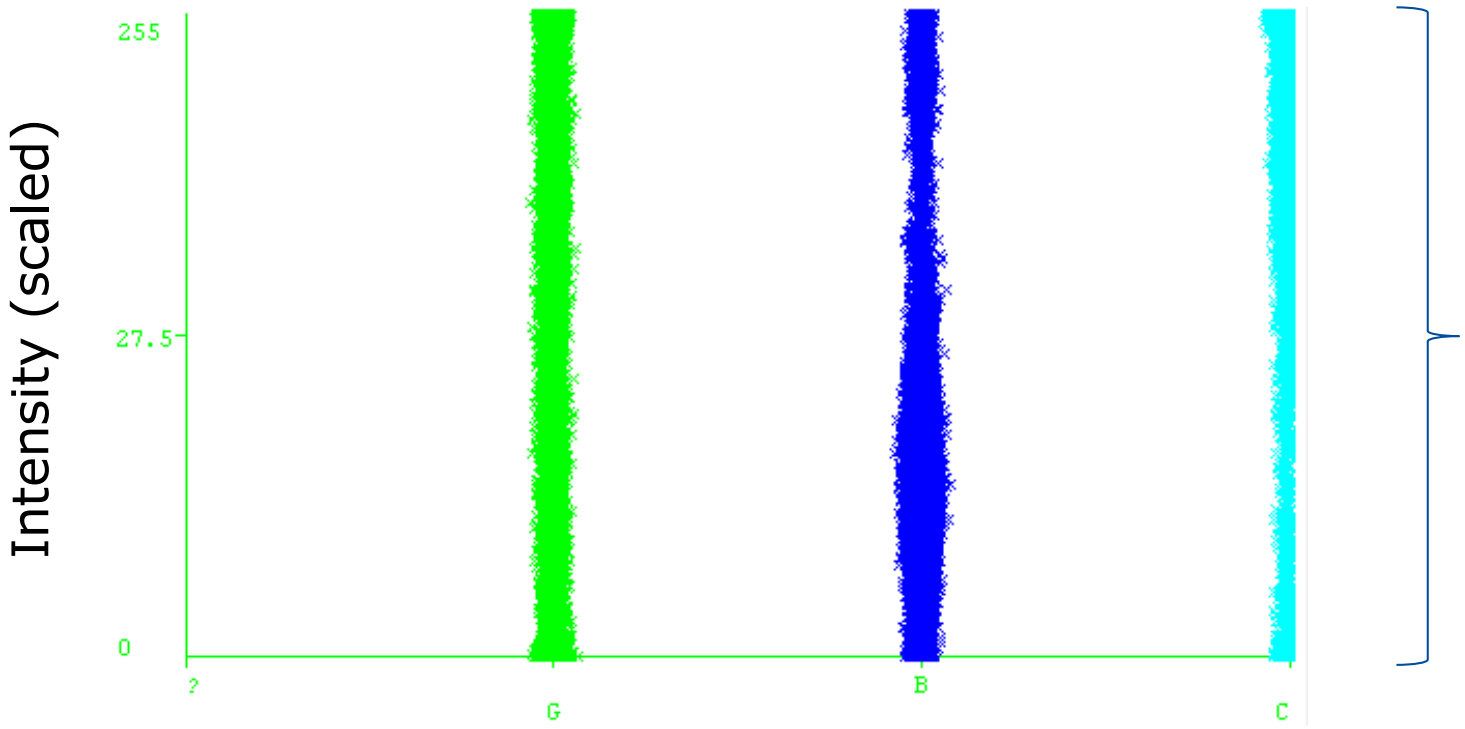






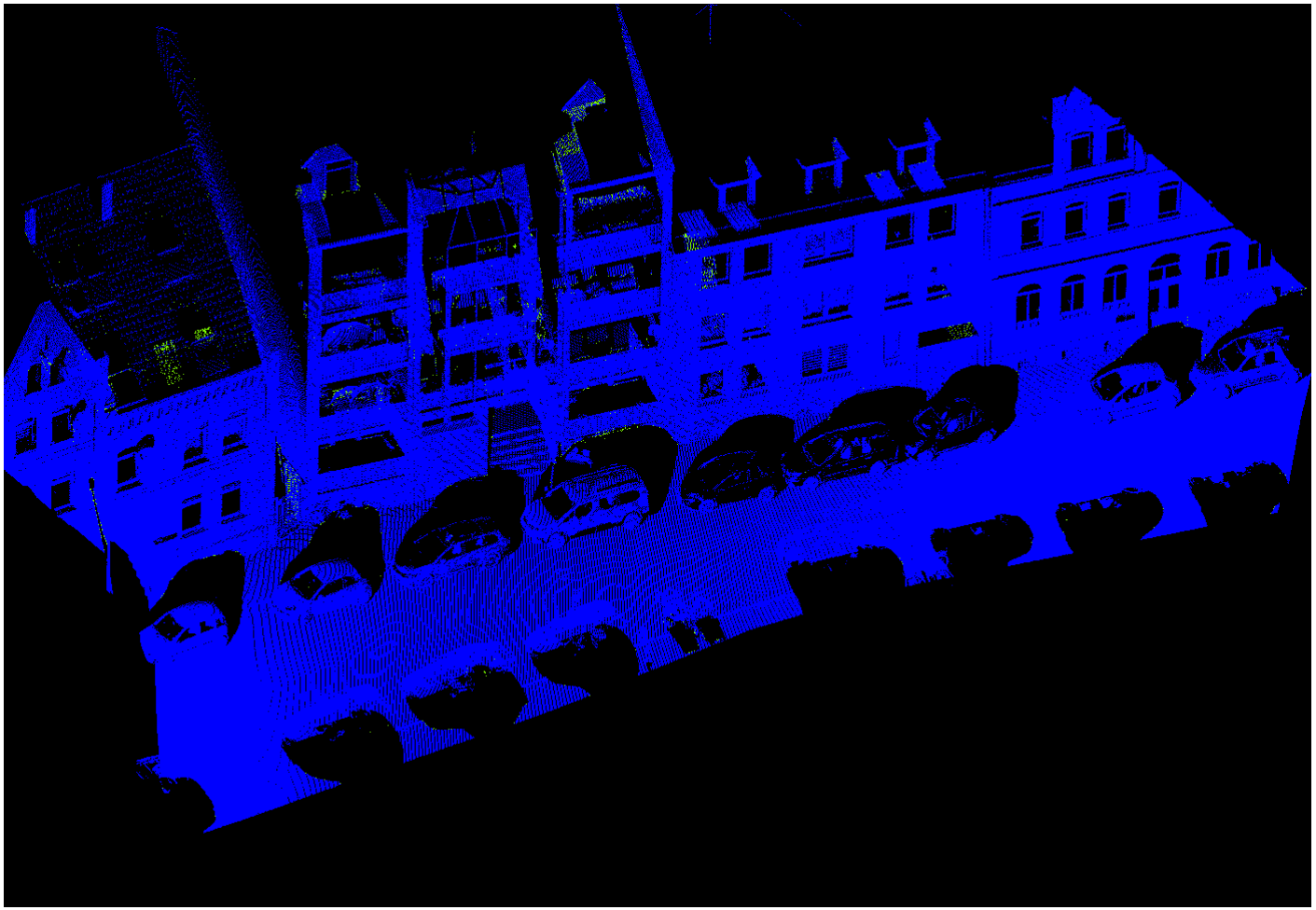
Attribute: “intensity” (amplitude)

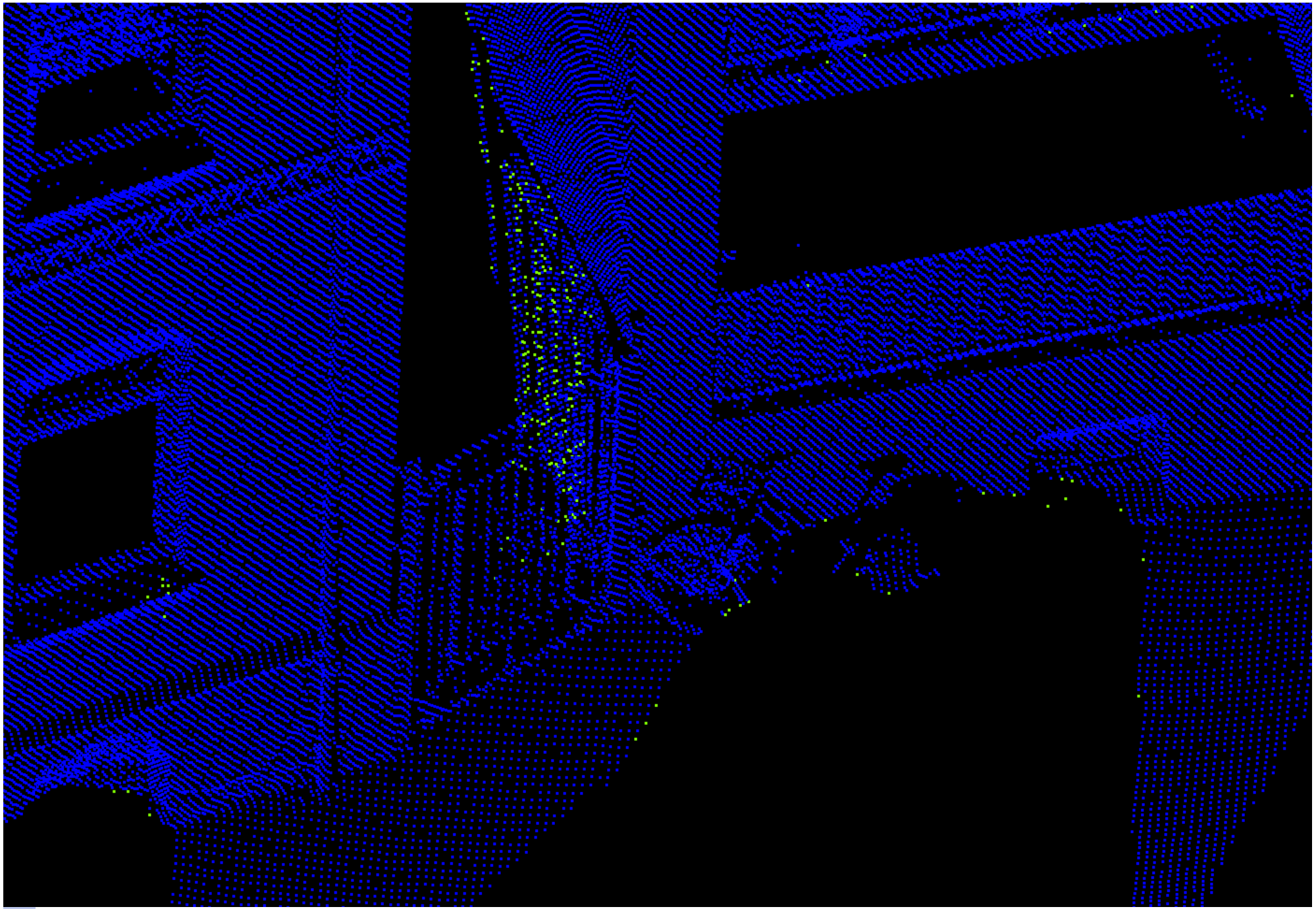
- Intensity is not a meaningful attribute

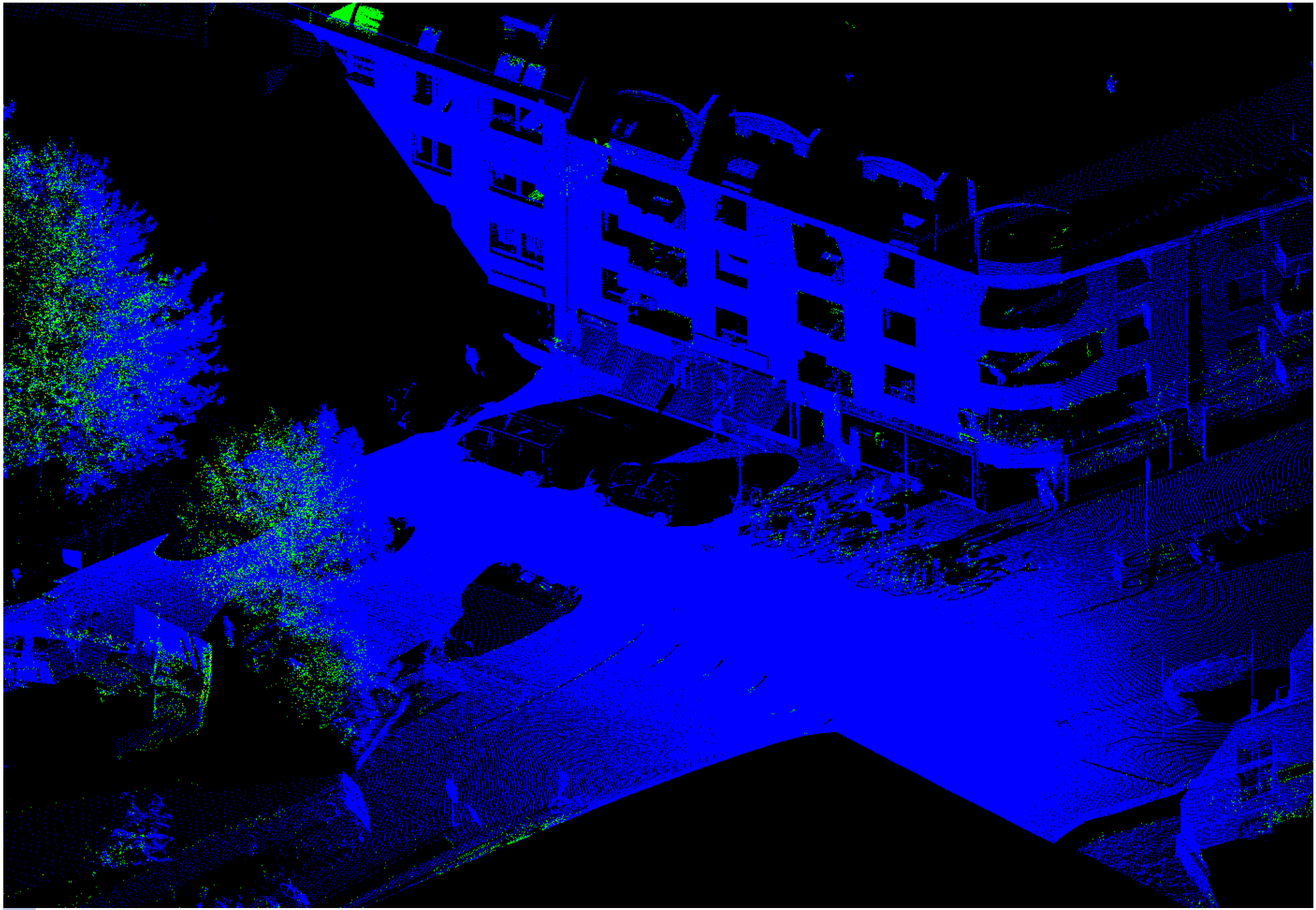


Attribute: “number of echoes”

- Laser scanner can record multiple echoes per emitted pulse
- (Blue: 1 echo)

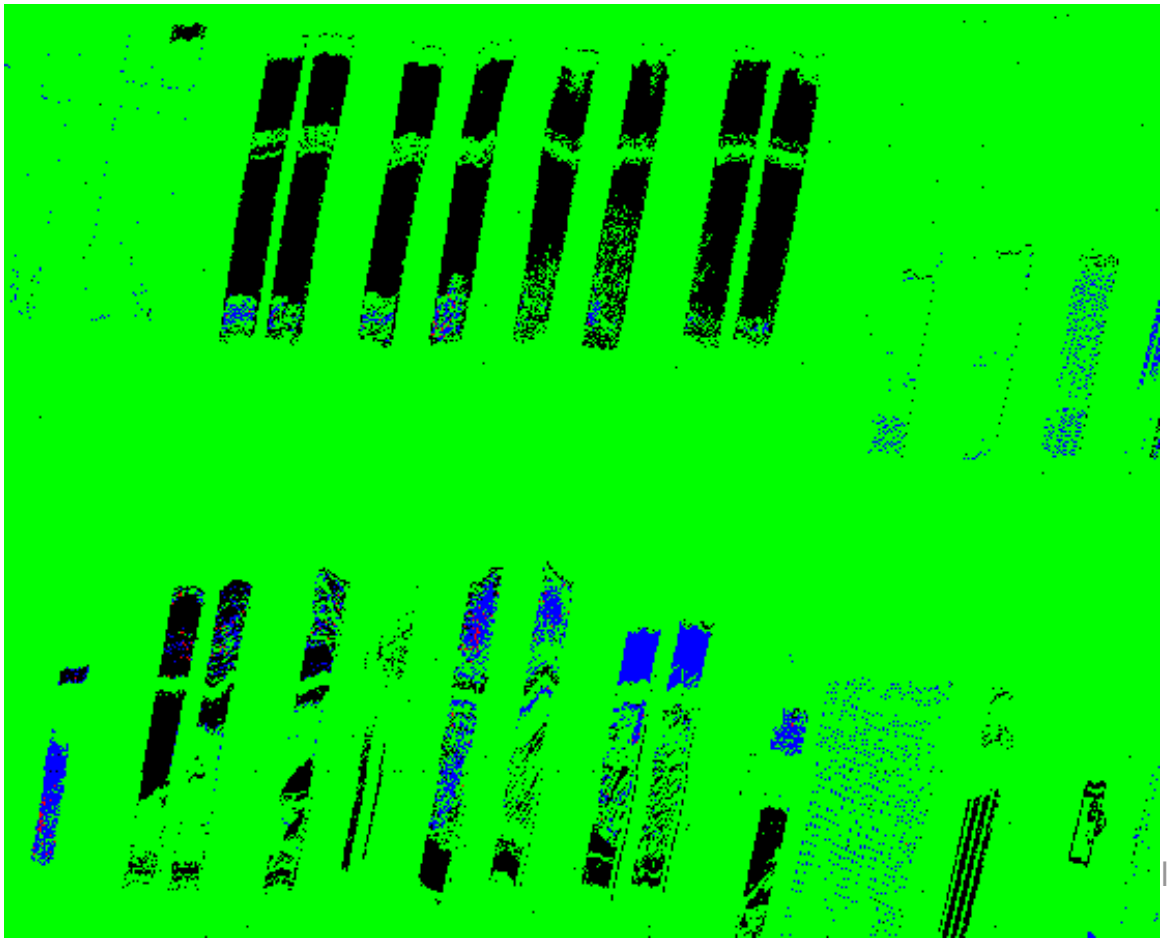






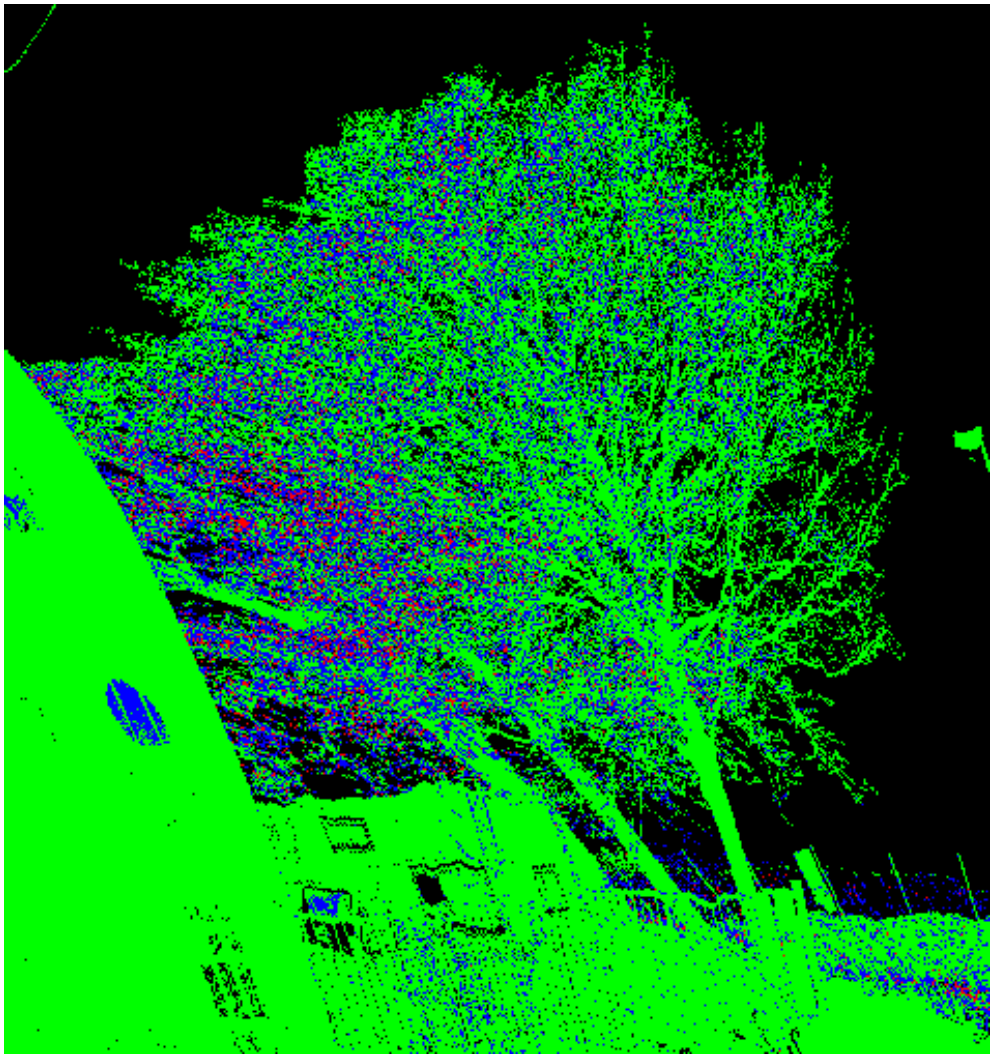
Attribute: “number of echoes”

- Usually, only one echo on the ground
- Facades also yield only one echo, windows may lead to multiple echoes



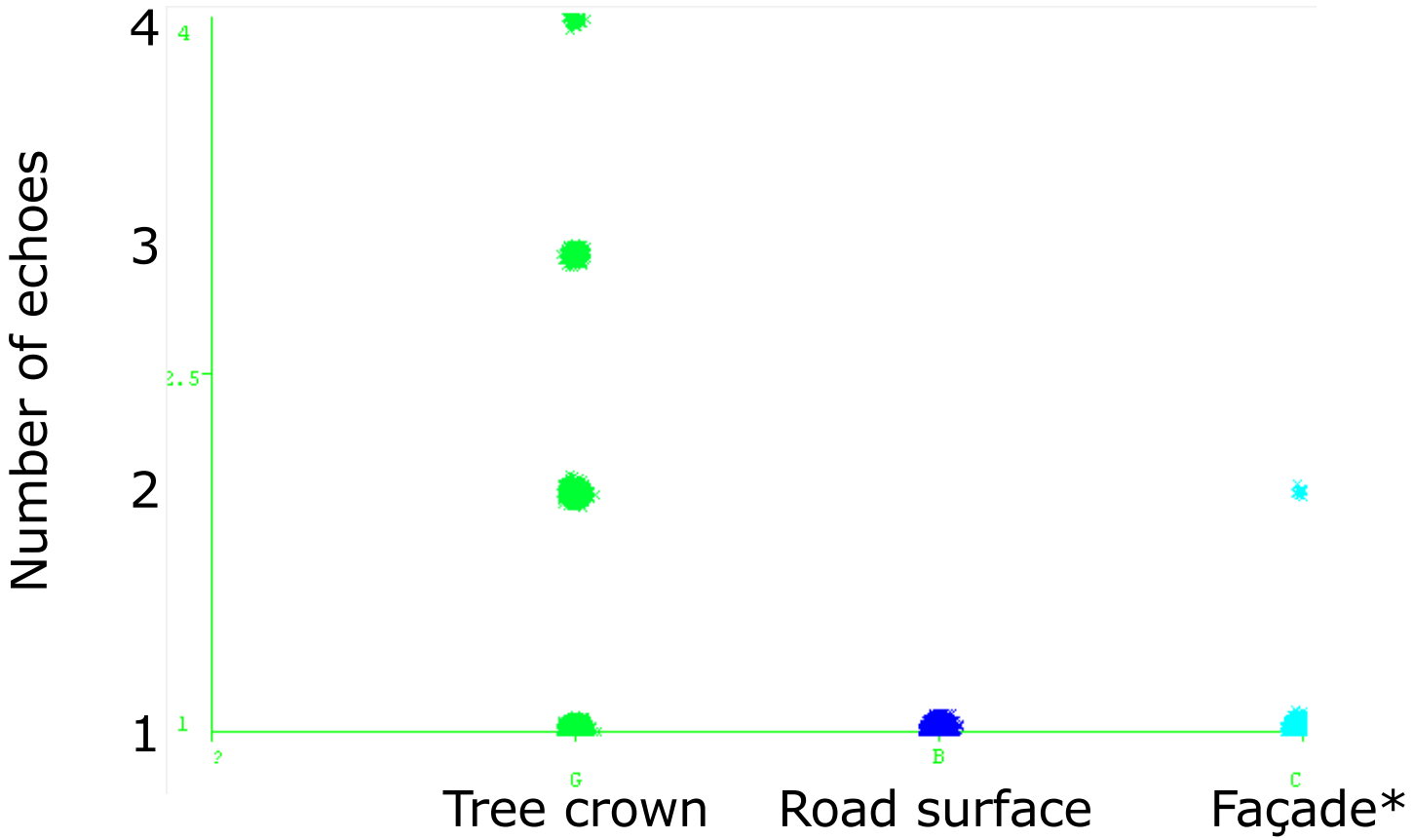
Attribute: “number of echoes”

- Trees may have 2-4 echoes in the tree crown
  - Normally, only one echo on the tree trunk



Attribute: “number of echoes”

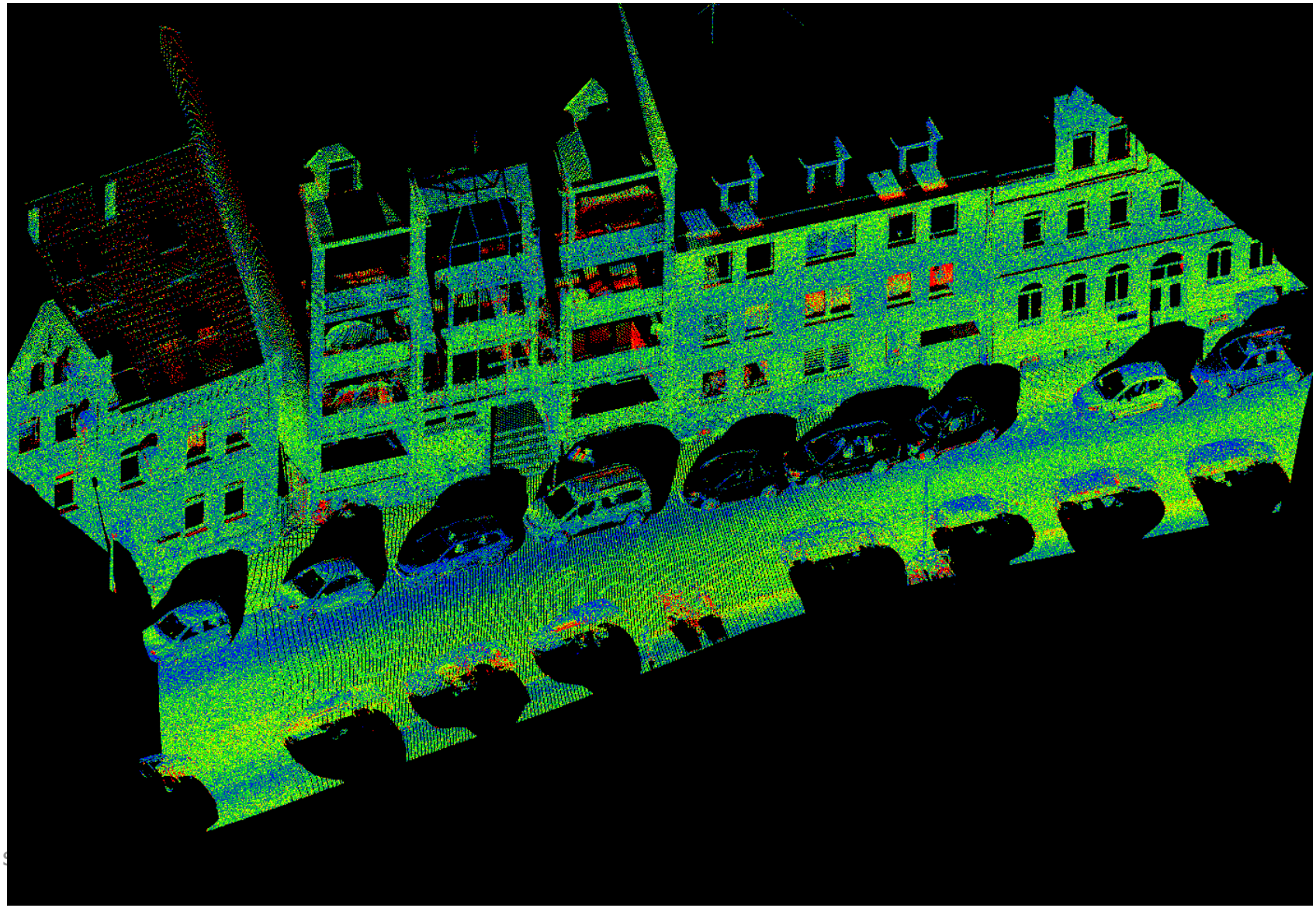
- Tree crowns have multiple echoes

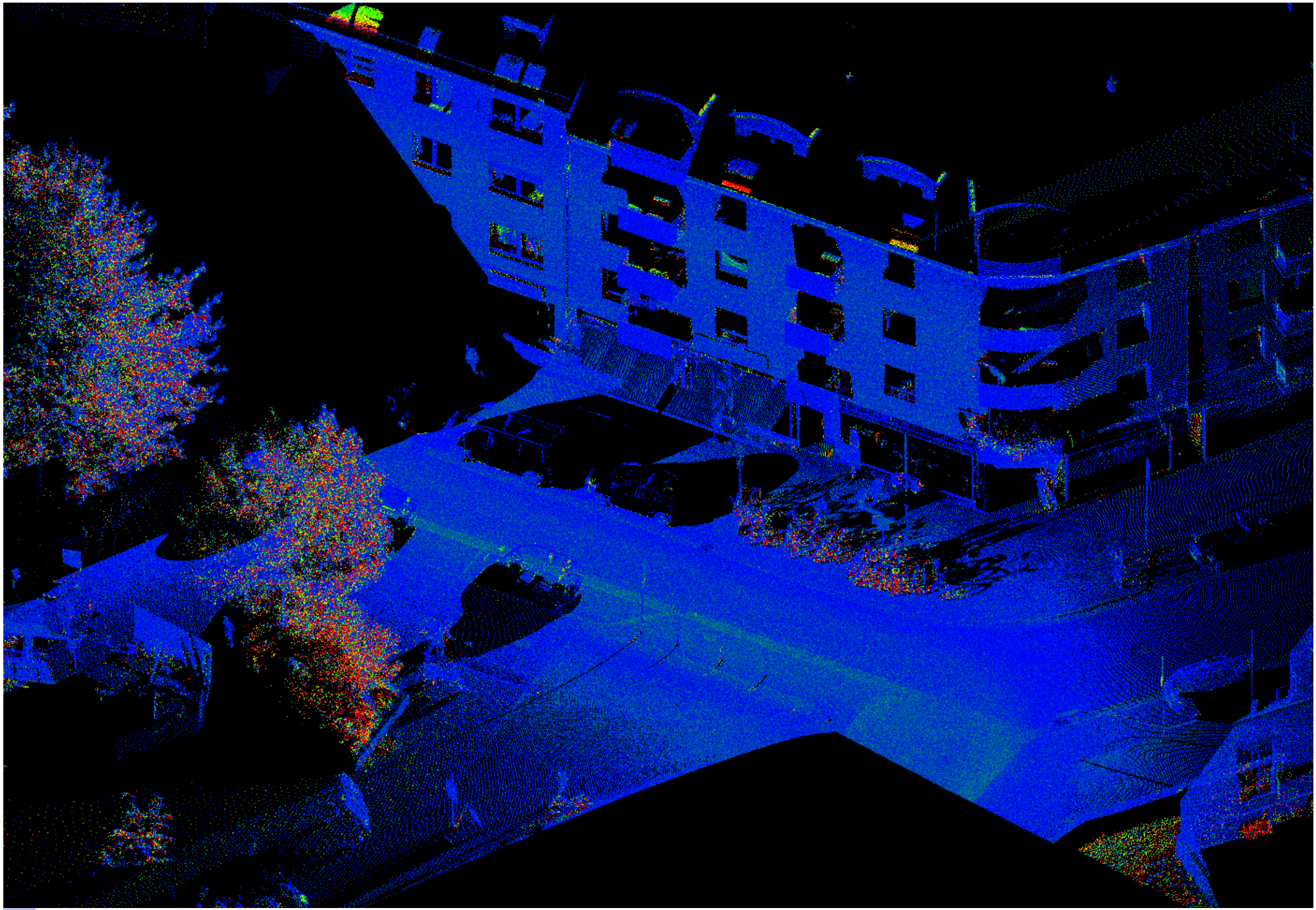


*No windows were marked.

Attribute: “width”

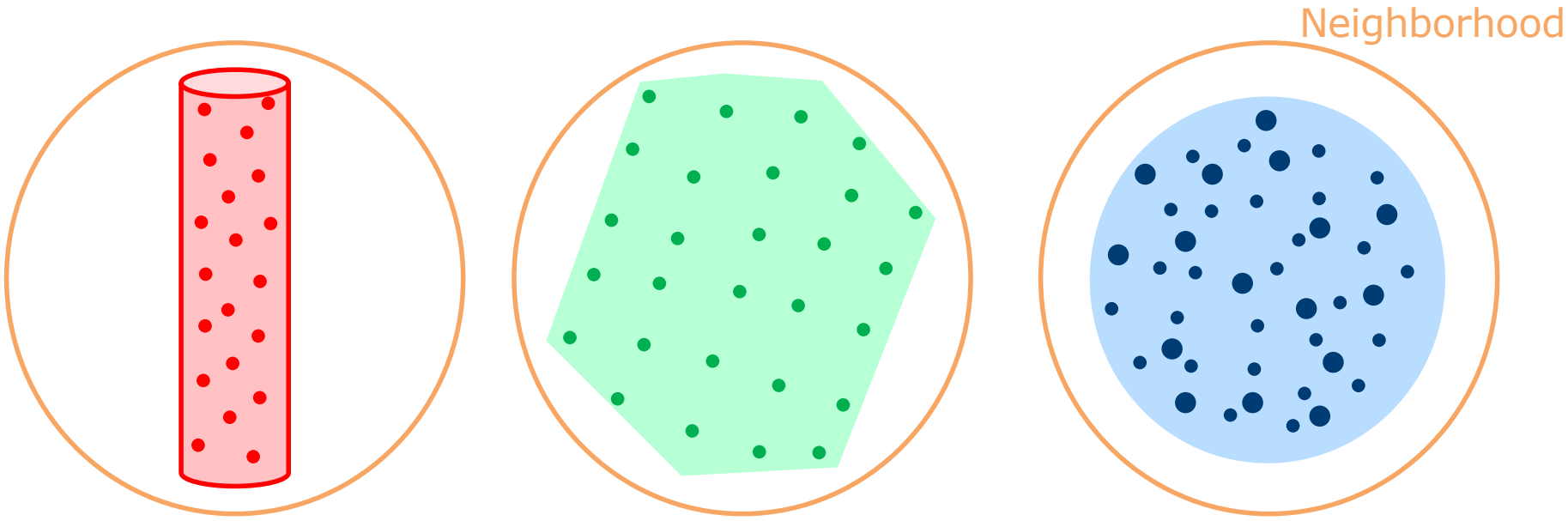
- Some scanners record the width of the returned pulse





Computed attributes: linear-planar-scatter

- Similar to the case in image processing (pixels), a single 3D point is not very informative
- To compute more meaningful features, the local neighborhood of the point has to be considered
- The following cases can be distinguished:



Computed attributes: linear-planar-scatter

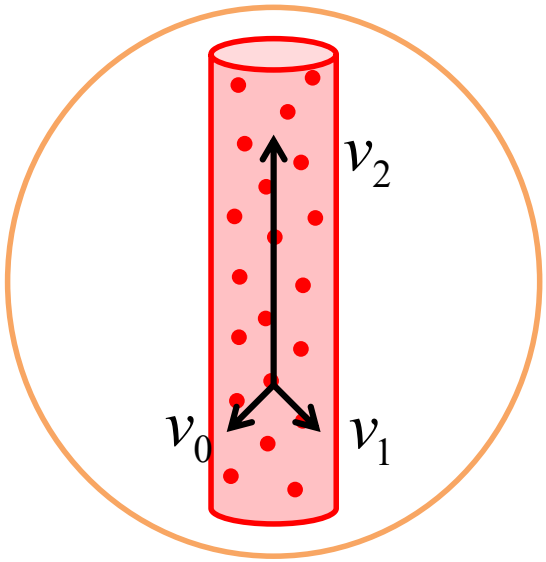
- How can these cases be distinguished?
- Reminder: principal component analysis (used previously in our plane estimation algorithm)
- Compute eigenvalues and eigenvectors of the following matrix (computed from the points, reduced by the center of mass):

$$\mathbf{A}^{\mathrm{T}}\mathbf{A} = \begin{bmatrix} \sum x_i'^2 & \sum x_i' y_i' & \sum x_i' z_i' \\ \cdot & \sum y_i'^2 & \sum y_i' z_i' \\ \cdot & \cdot & \sum z_i'^2 \end{bmatrix}$$

Computed attributes: linear-planar-scatter

- This yields:

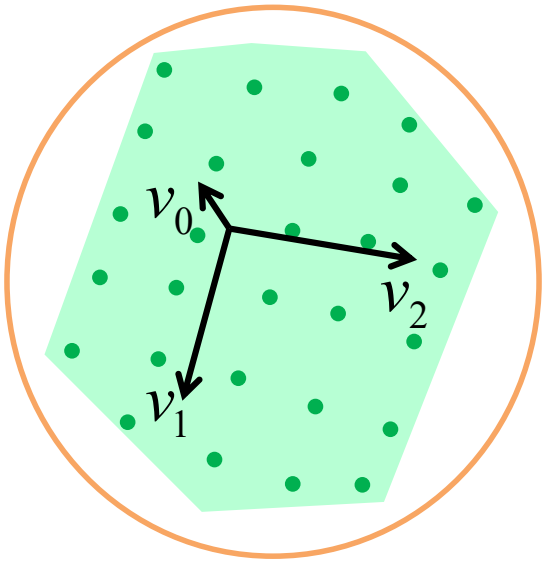
Linear (1D)



$\lambda_0 \approx \lambda_1 << \lambda_2$

"small-small-large"

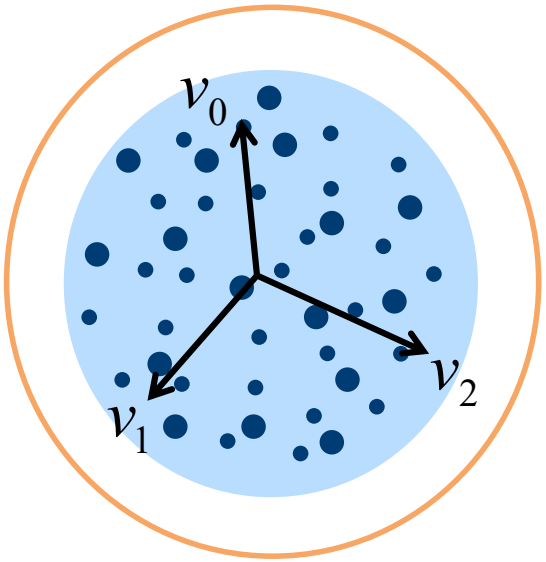
Planar (2D)



$\lambda_0 << \lambda_1 \approx \lambda_2$

"small-large-large"

Scatter (3D)



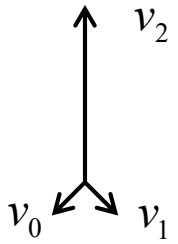
$\lambda_0 \approx \lambda_1 \approx \lambda_2$

"large-large-large"

Computed attributes: linear-planar-scatter

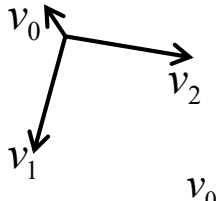
- From this, the following features can be defined:
  - Linear:

$$l = \frac{\lambda_2 - \lambda_1}{\lambda_2}$$



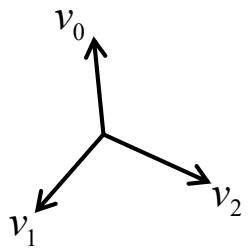
  - Planar:

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



  - Scatter:

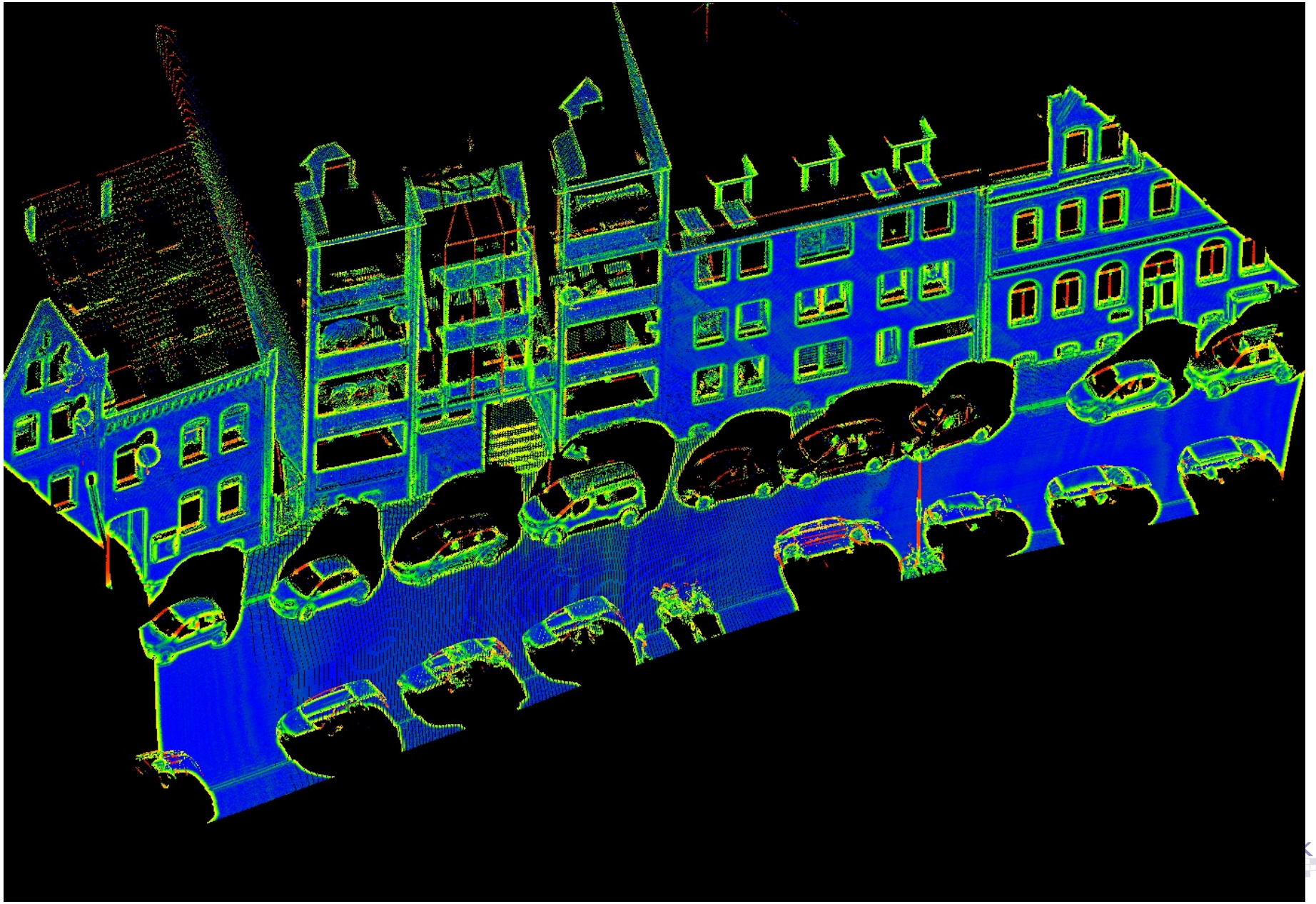
$$v = \frac{\lambda_0}{\lambda_2}$$



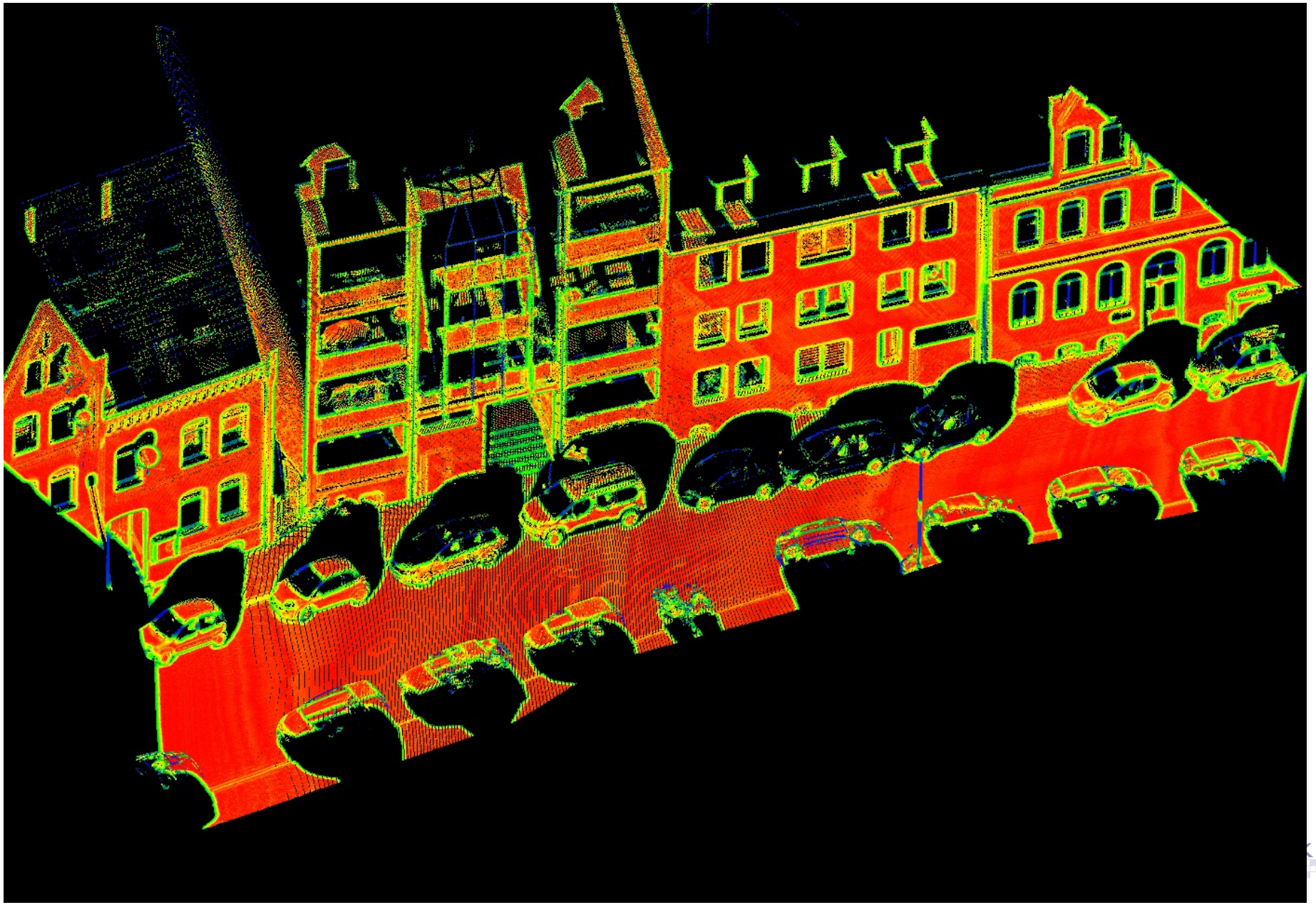
- Note:

$$l + p + v = 1$$

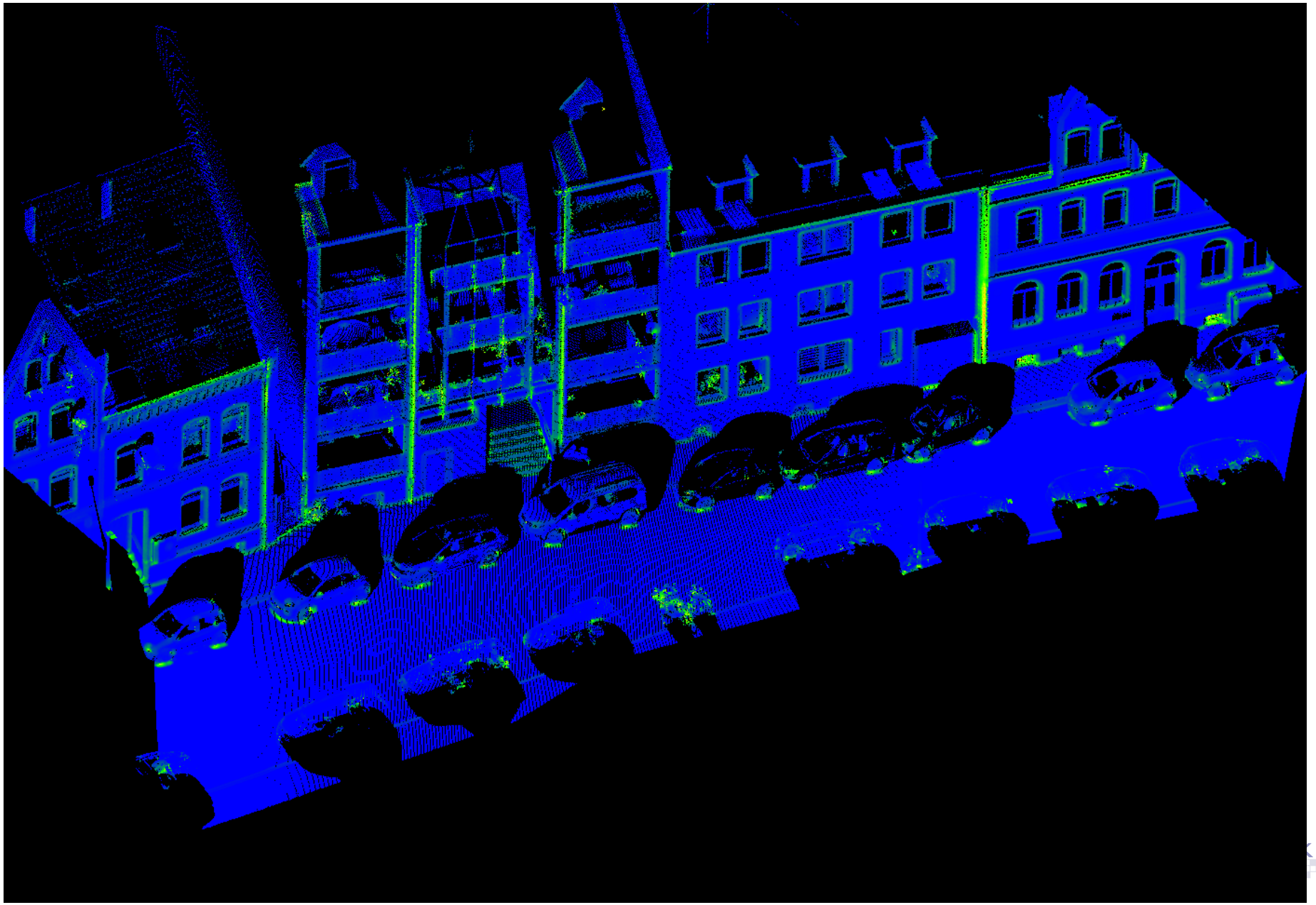
Linear – planar – scatter with r = 20 cm: Linear



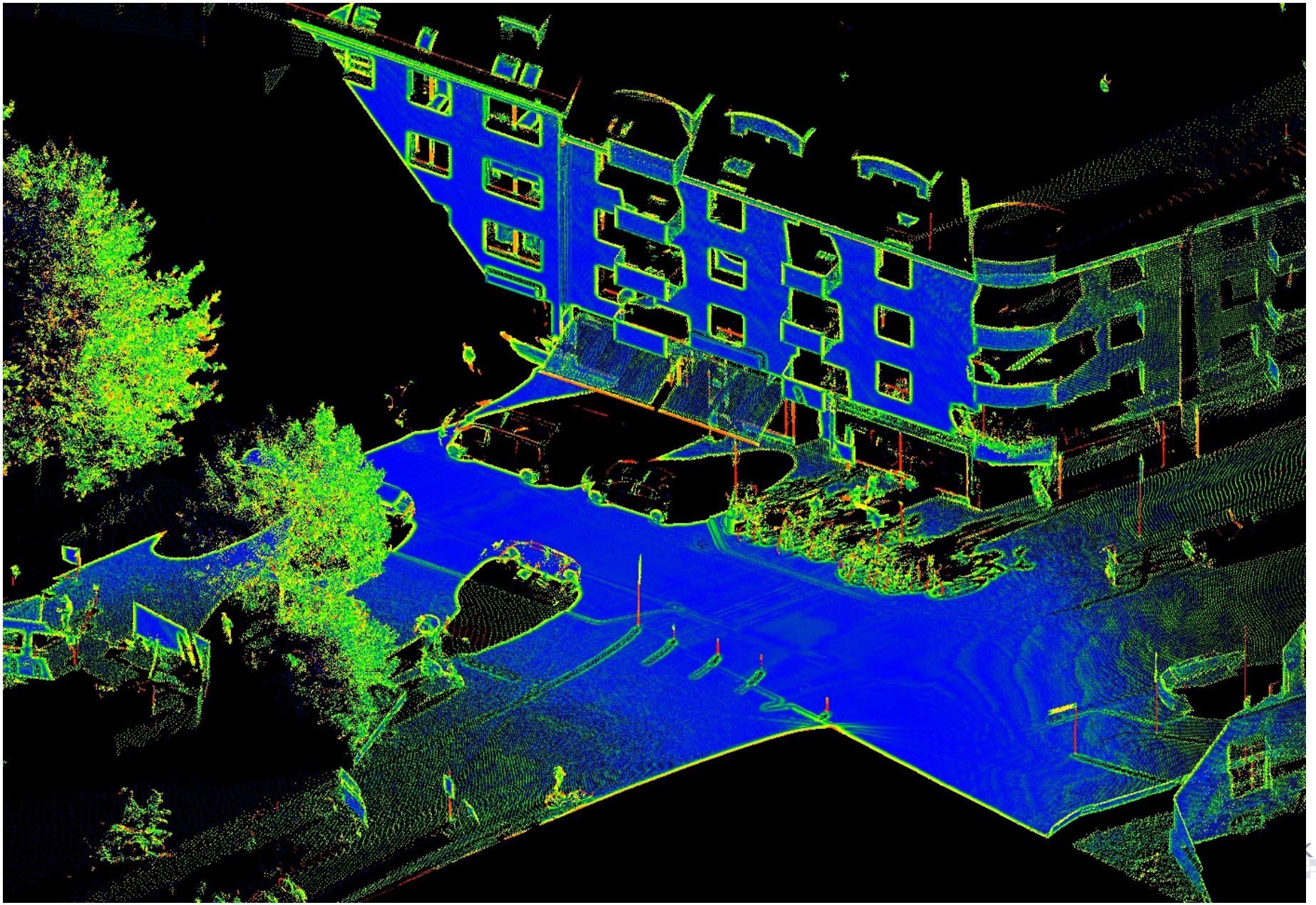
Linear – planar – scatter with r = 20 cm: Planar



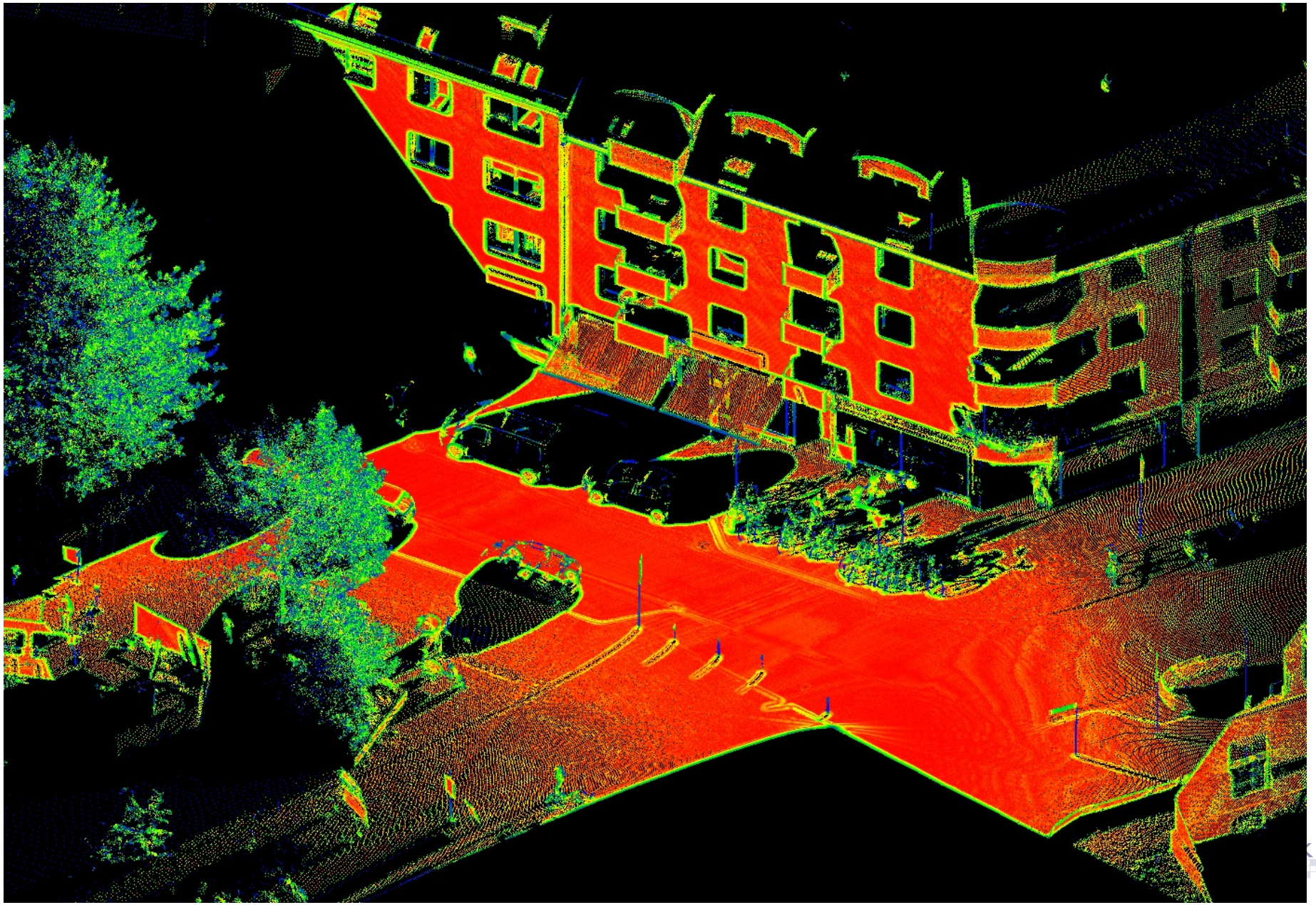
Linear – planar – scatter with r = 20 cm: Scatter



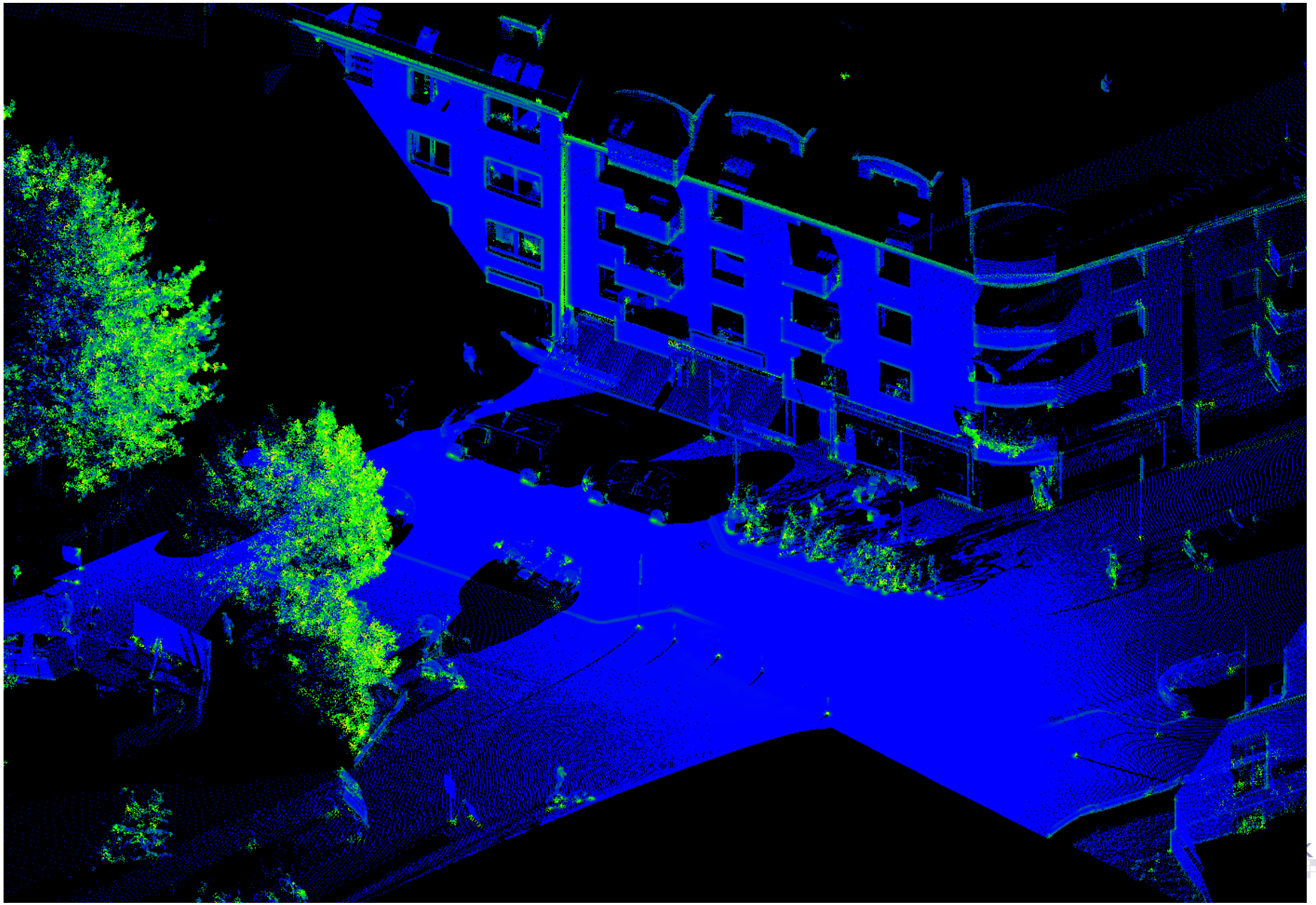
Linear – planar – scatter with r = 20 cm: Linear



Linear – planar – scatter with r = 20 cm: Planar



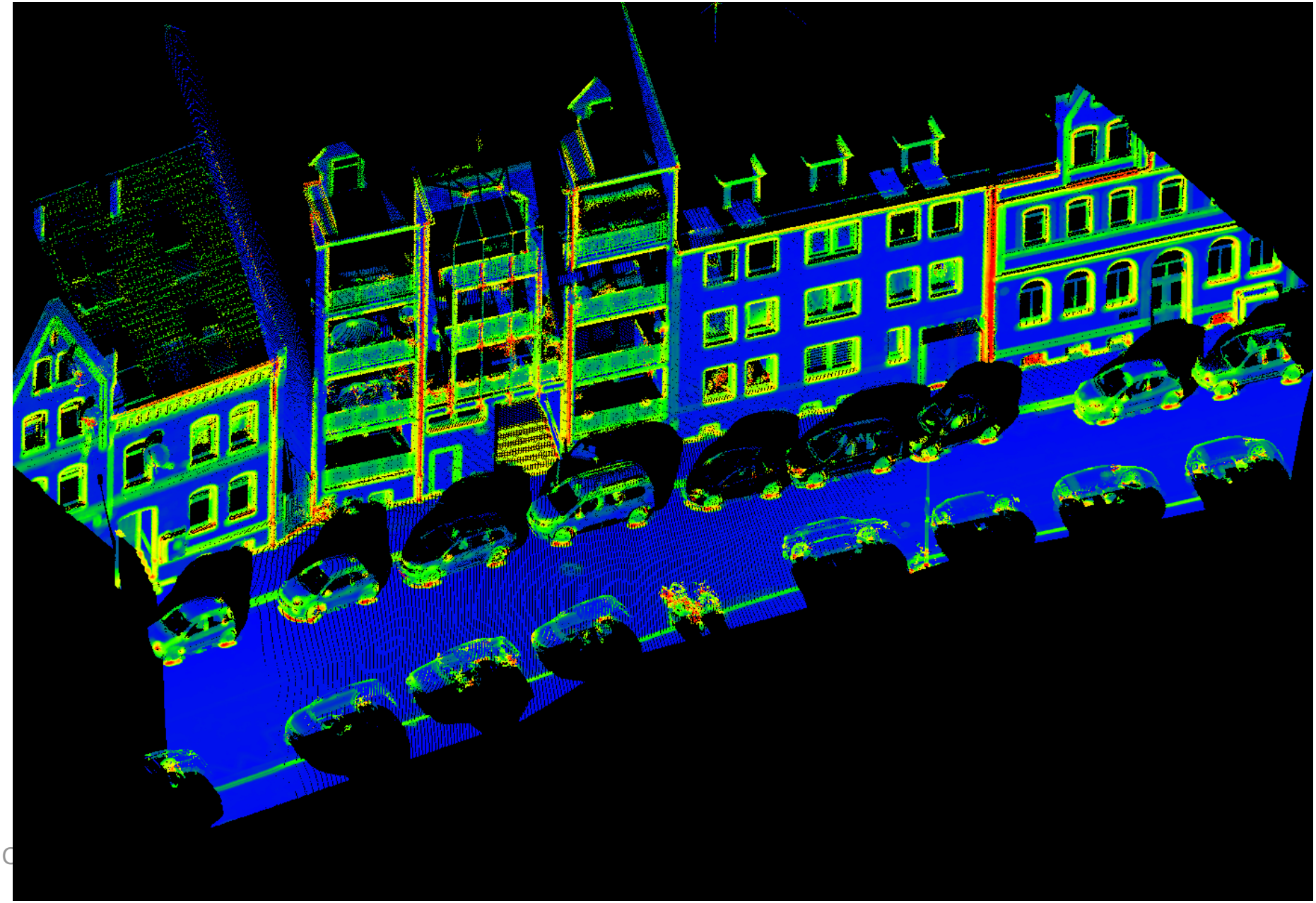
Linear – planar – scatter with r = 20 cm: Scatter

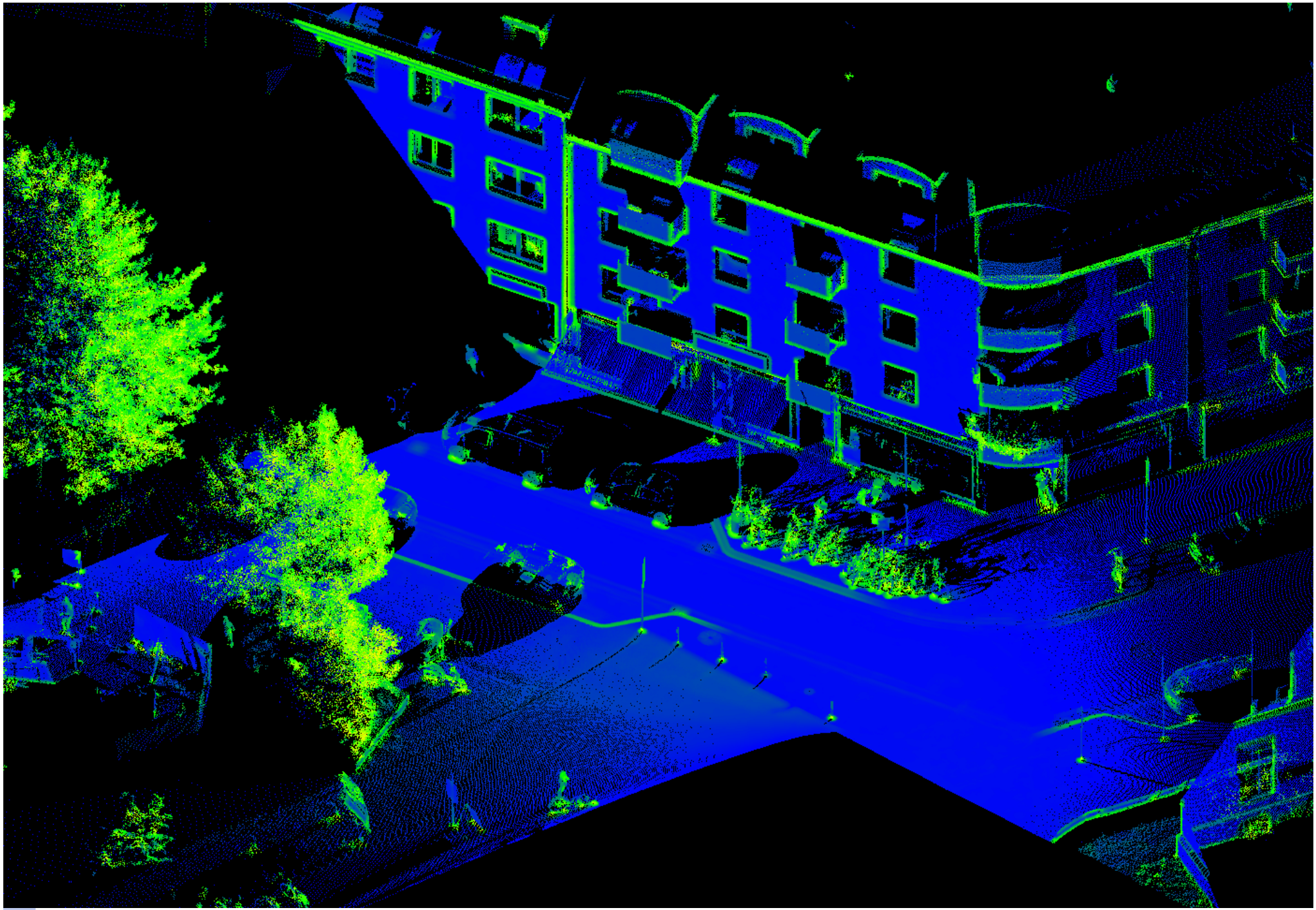


Observations: linear-planar-scatter

- **Poles** are identified as “linear”
- **Tree crowns** are identified as “scatter” depending on radius
- To get a more reliable “scatter” classification, one would have to enlarge the radius
  - However, this would blur all results → tradeoff
- **Corners of windows** are identified as “linear”
  - Just because of the shape of the intersection with the PCA ball

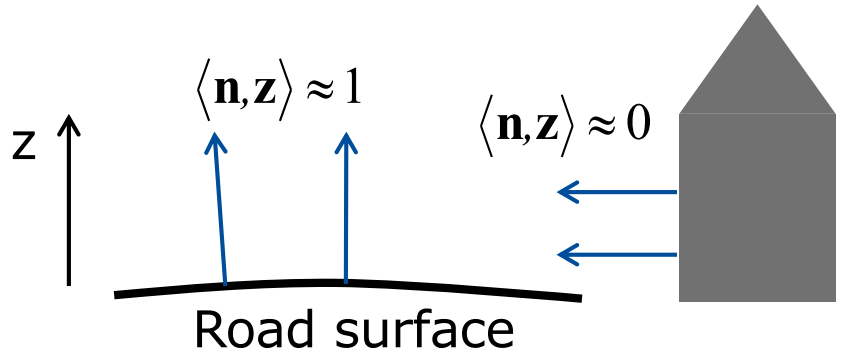
Computed attribute: square root of the smallest eigenvalue (= standard deviation of est. plane)



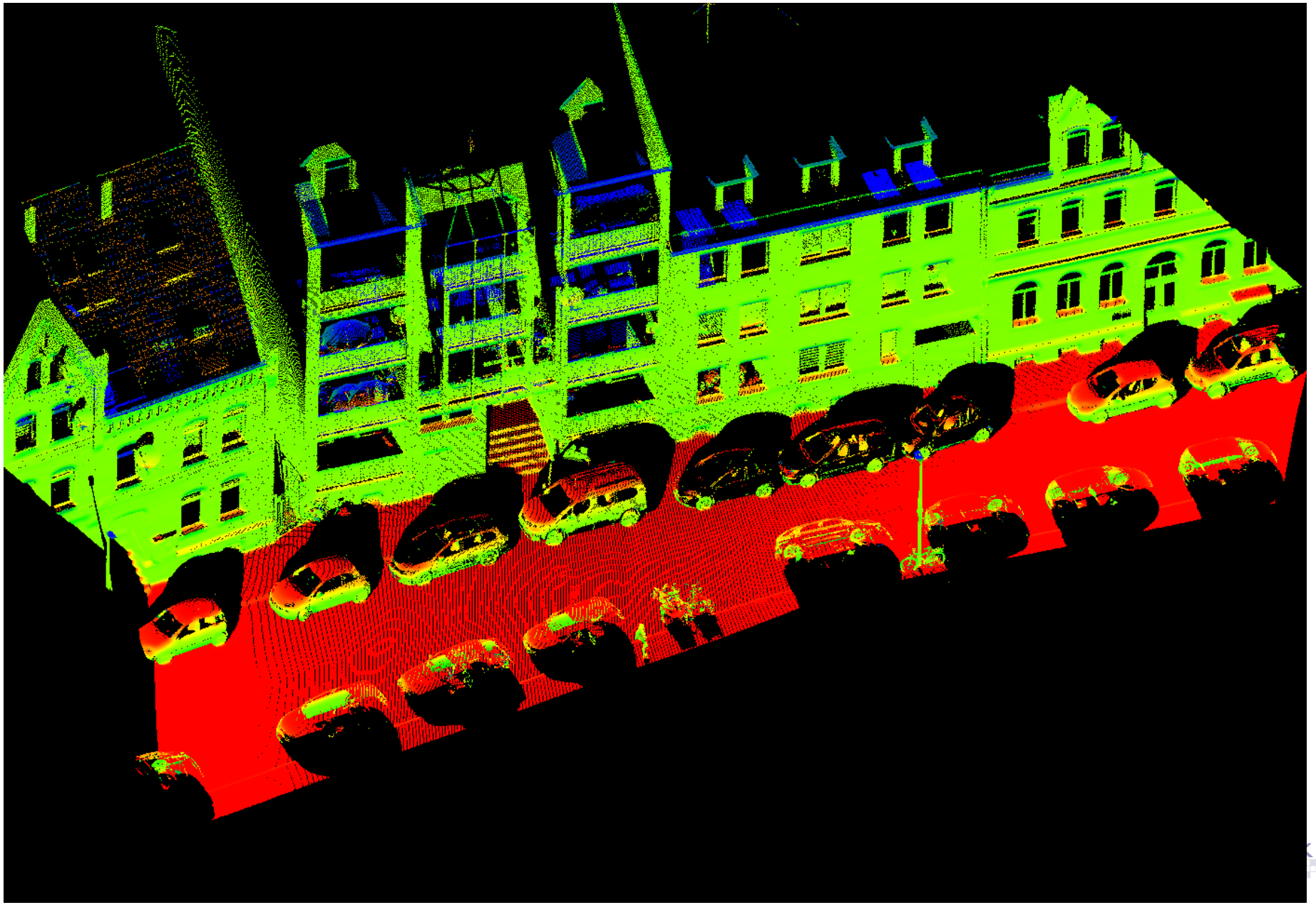


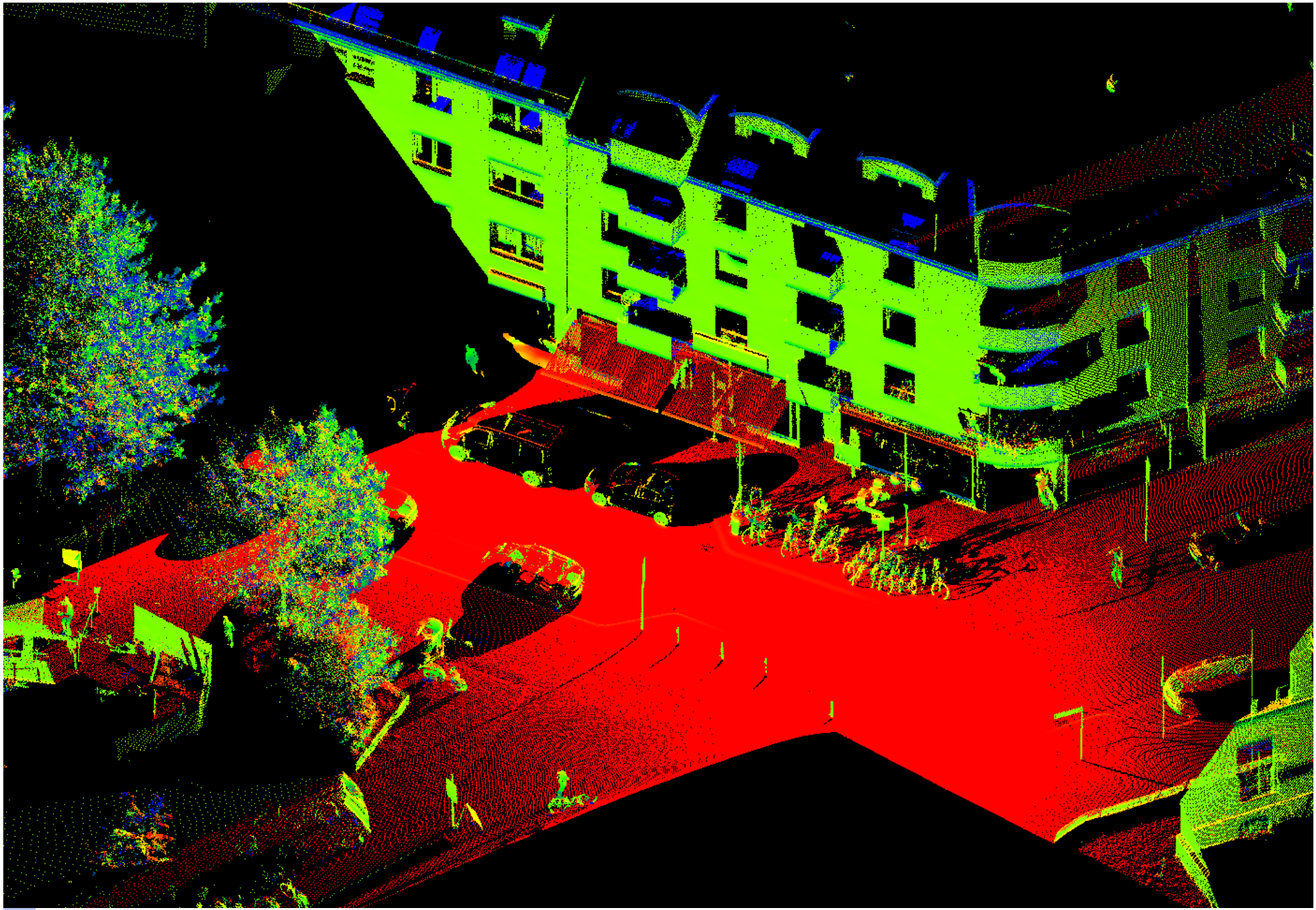
Computed attribute: Scalar product EV0, z-axis

- When performing the "linear-planar-scatter" analysis, a plane estimation is done (anyways)
- The eigenvector (EV0) belonging to the **smallest** eigenvalue is the normal vector of the plane
- Compute the scalar product (which is the cosine of the angle) with the Z-axis
- Reasoning:
  - For the road surface, this is largest (1.0)
  - For facades, it should be mostly zero
  - For overhanging balconies, it is -1.0



Computed attribute: Scalar product EV0, z-axis

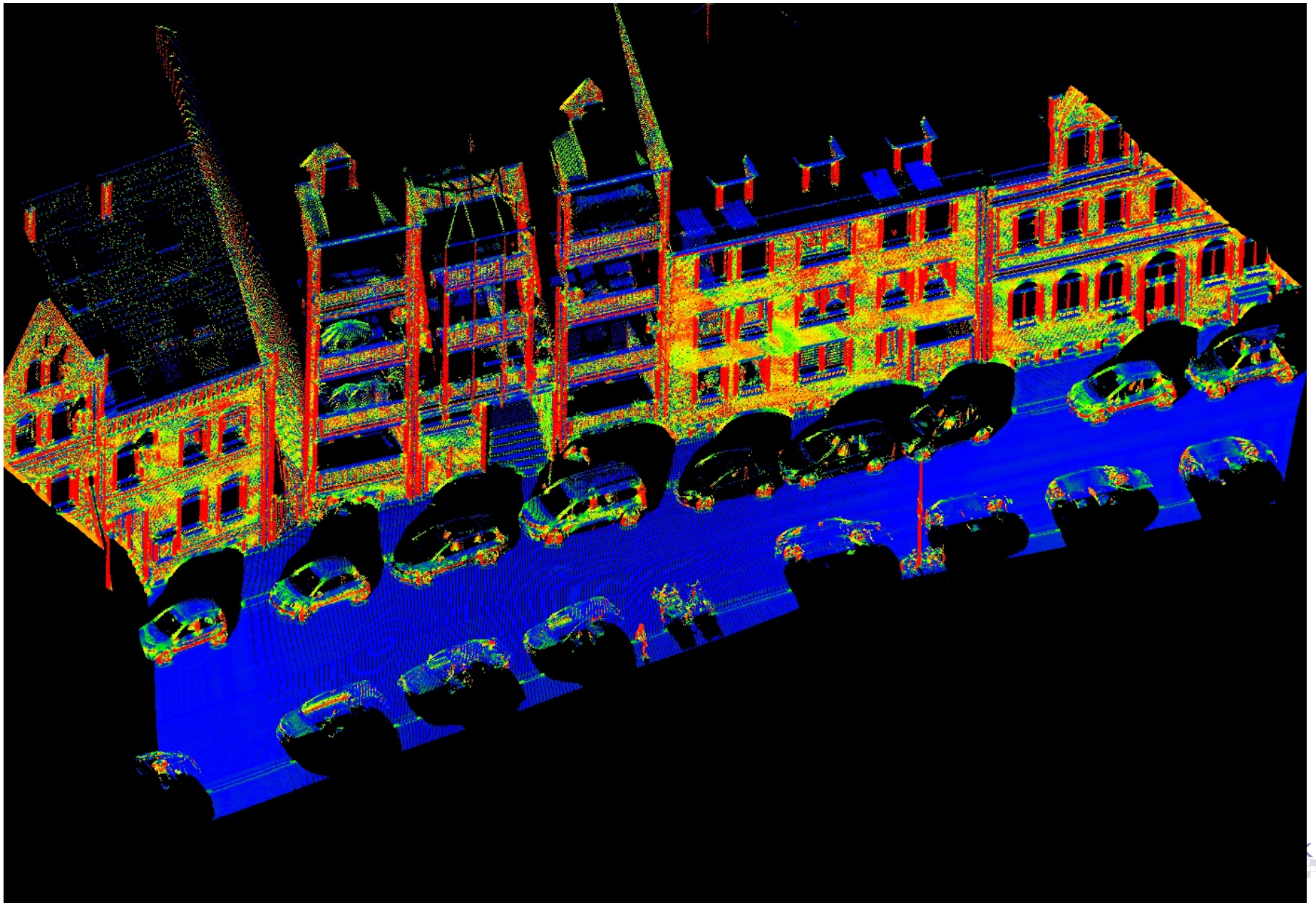


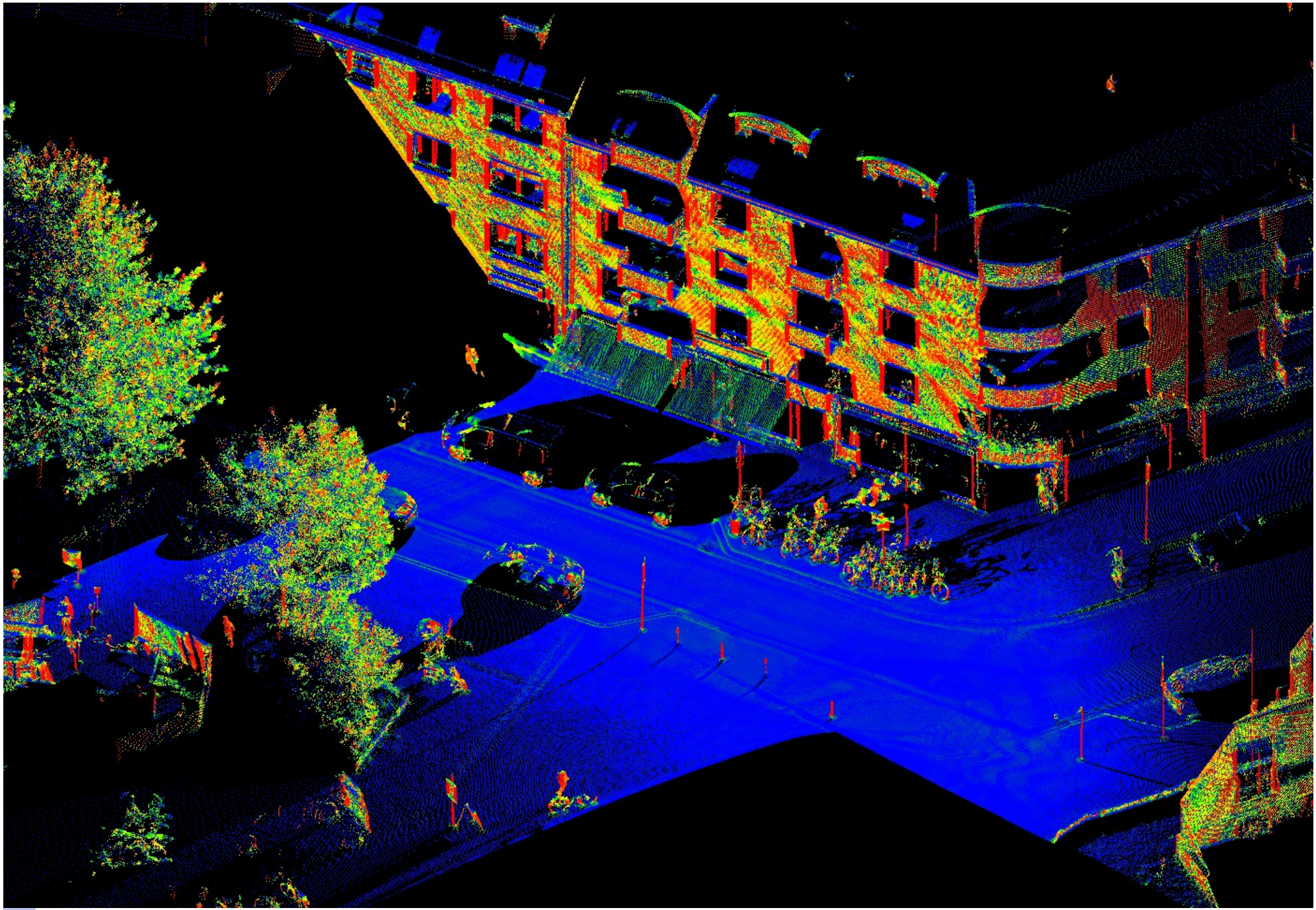


Computed attribute: Scalar product EV2, z-axis

- Similarly to EV0, the eigenvector belonging to the largest eigenvalue (EV2) yields the direction vector of linear structures
- Again, compute scalar product with z-axis
- This allows to distinguish upright linear structures (poles, bollards, tree trunks) from horizontal ones (overhead cables, cross bars)
- Note for planes the EV2 vector may be an arbitrary vector in the plane
  - Therefore, results on planes are arbitrary
  - On the ground, the scalar product is close to zero.

Computed attribute:
Scalar product EV2, z-axis





Summary of all attributes

Attributes (computed for each point)

- Reflectance [reflectance]
- Number of echoes [num_targets]
- Linear, planar, scatter [dist_linear, dist_planar, dist_scatter]
- Square root of smallest eigenvalue [sqrt_ev0]
- Scalar product of EV0 with z-axis [eve0_z]
- Scalar product of EV2 with z-axis [eve2_z]
- Height above ground [rel_height]

- Total: 9 features, i.e., 9 dimensional feature vector.

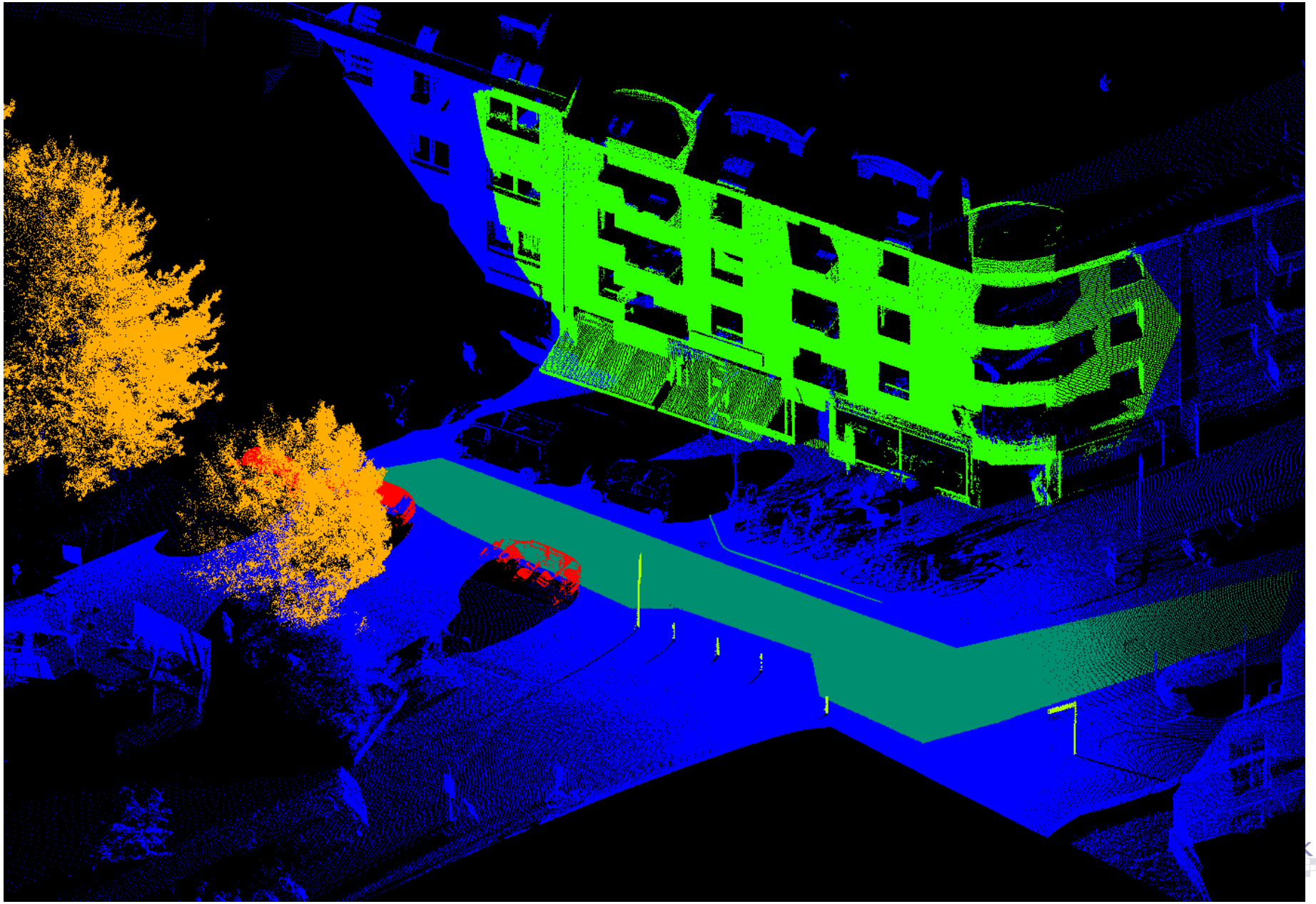
Classification examples

Using a Random Forest

- …from scikit-learn implementation
- 32 trees
- Balanced weights
- Entropy criterion
- No maximum tree depth (splits when >= 2 samples)
- Standard scaler applied before training/ test
- Training : test = 70 : 30

First attempt: Sloppy training data

Marked: ground, curb, building, pole-like, tree, closed-vehicle



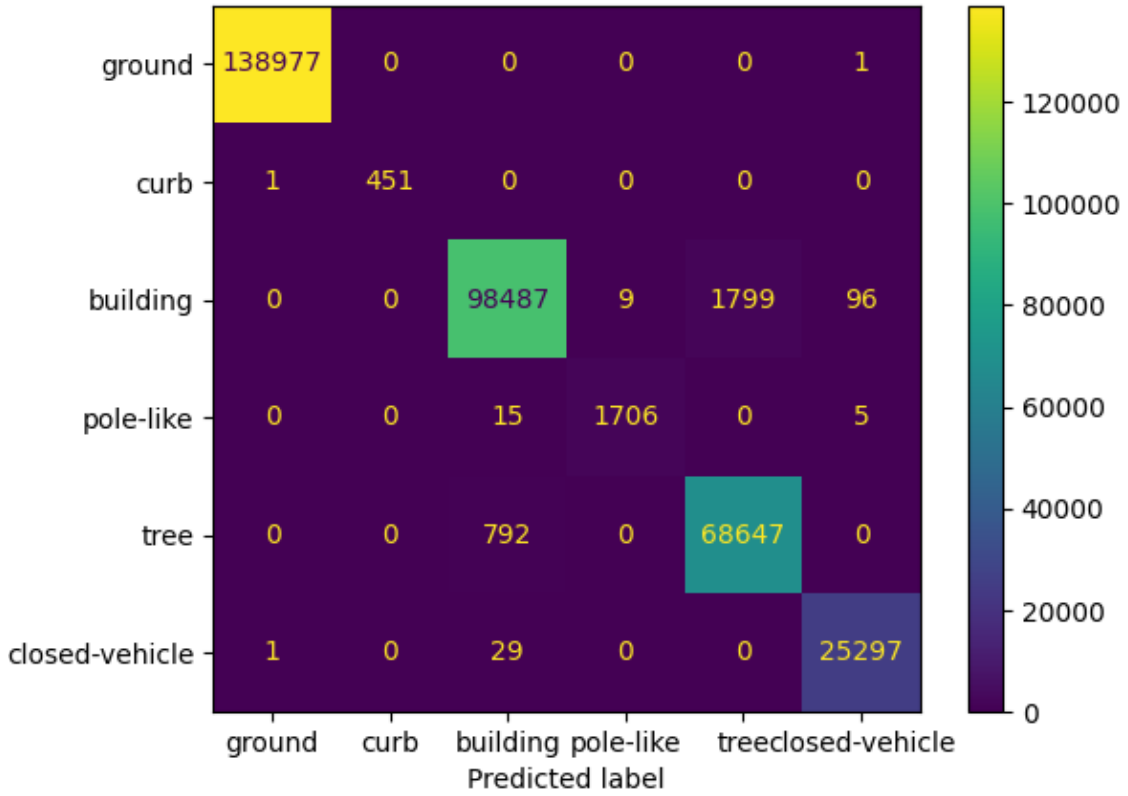
Marked: ground, curb, building, pole-like, tree, closed-vehicle

- Training : Test = 70 : 30

```
Training: reading 210903_123059_Scanner_1_simple
  Labels: 1121043
Total labels: 1121043
Class statistics
          Class   Count
         ground  462568
           curb    1432
       building  335355
      pole-like    5719
           tree  231595
 closed-vehicle   84374
```

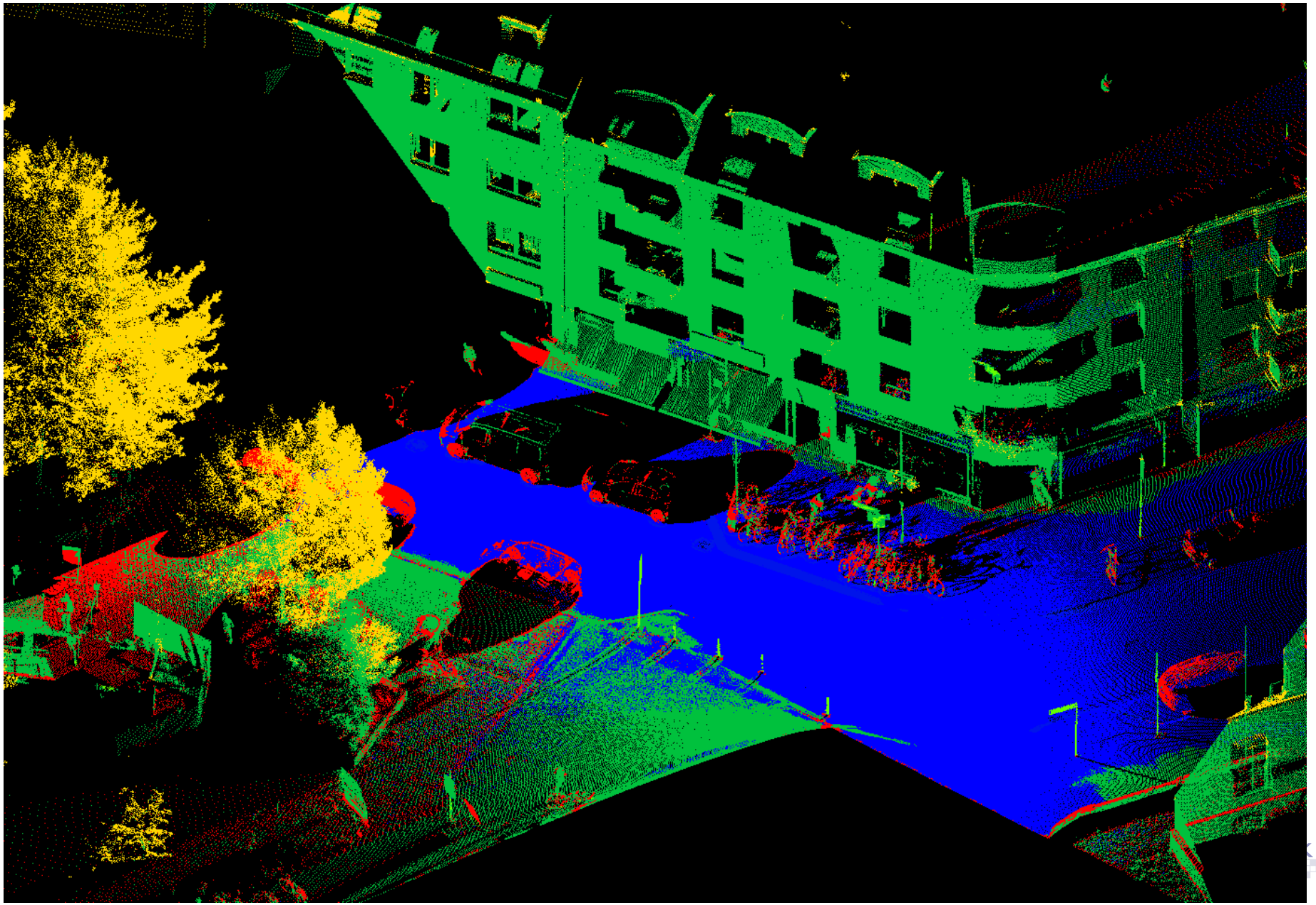
Result on test set

- Looks „perfect“

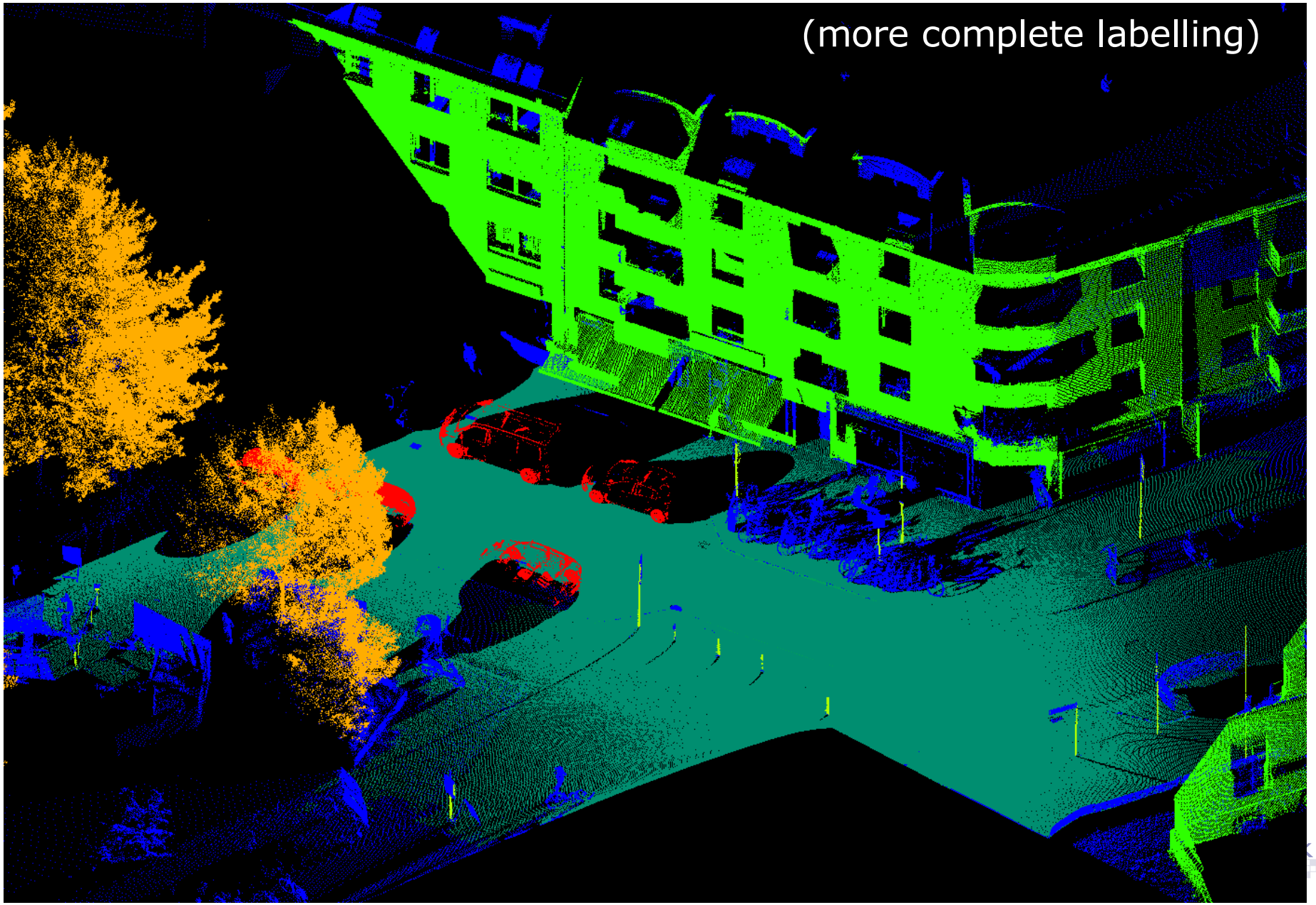


Class	Precision	Recall	F1 score
ground	1.000	1.000	1.000
curb	1.000	0.998	0.999
building	0.992	0.981	0.986
pole-like	0.995	0.988	0.992
tree	0.974	0.989	0.981
closed-vehicle	0.996	0.999	0.997

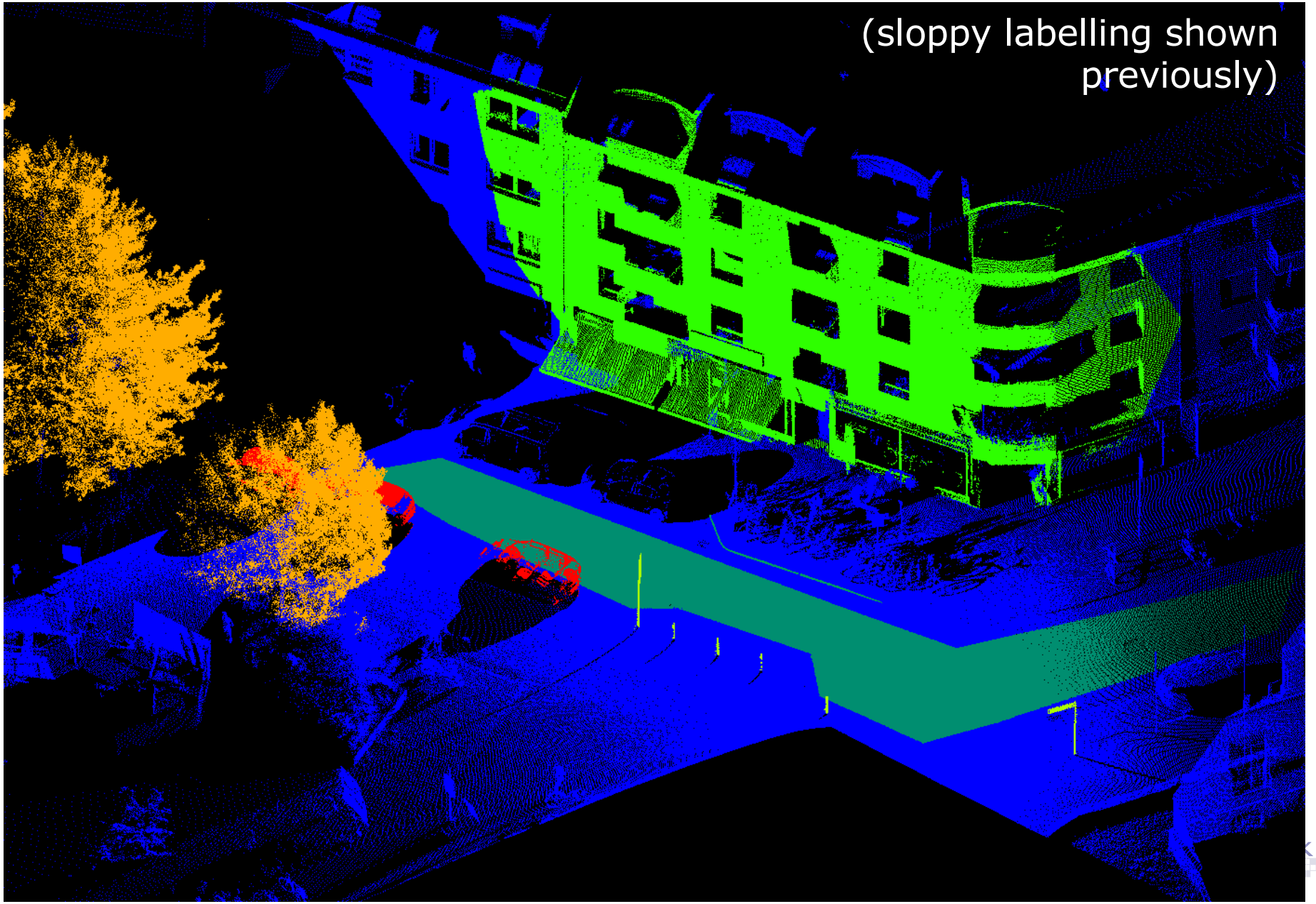
Visual result on full scan strip



Compare classification with more complete manual labelling (of the same scene)



C.f. sloppy labelling



Test results on more complete labelling

- Note this is actually too optimistic because the “sloppy labelling” is a subset of the “more complete labelling”
- Much worse results!

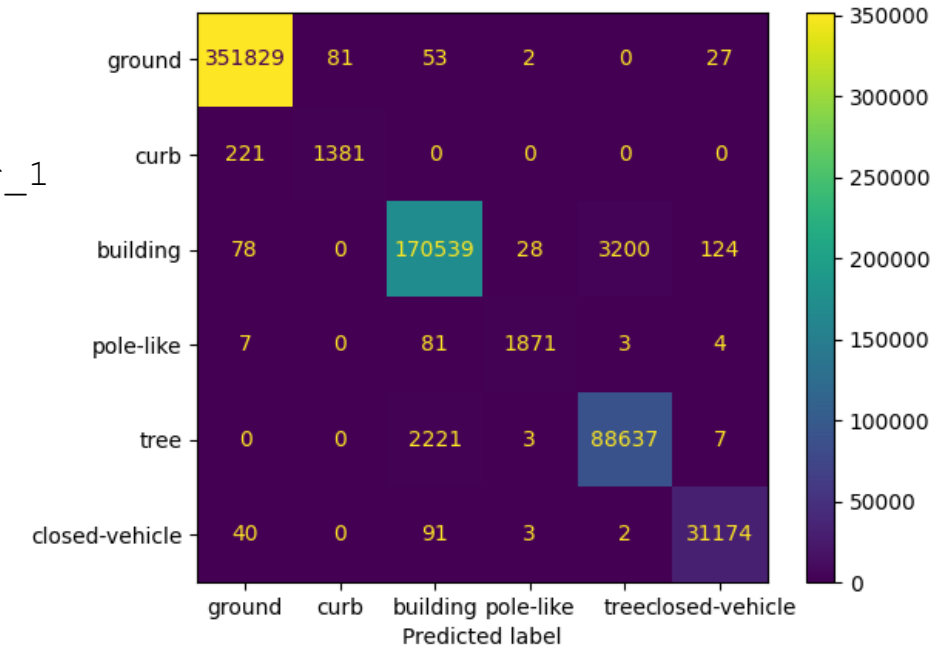
Class	Precision	Recall	F1 score
ground	0.998	0.814	0.897
curb	0.073	0.225	0.111
building	0.759	0.961	0.848
pole-like	0.939	0.586	0.722
tree	0.939	0.943	0.941
closed-vehicle	0.659	0.982	0.789

Second attempt: Use the more complete labelling for training

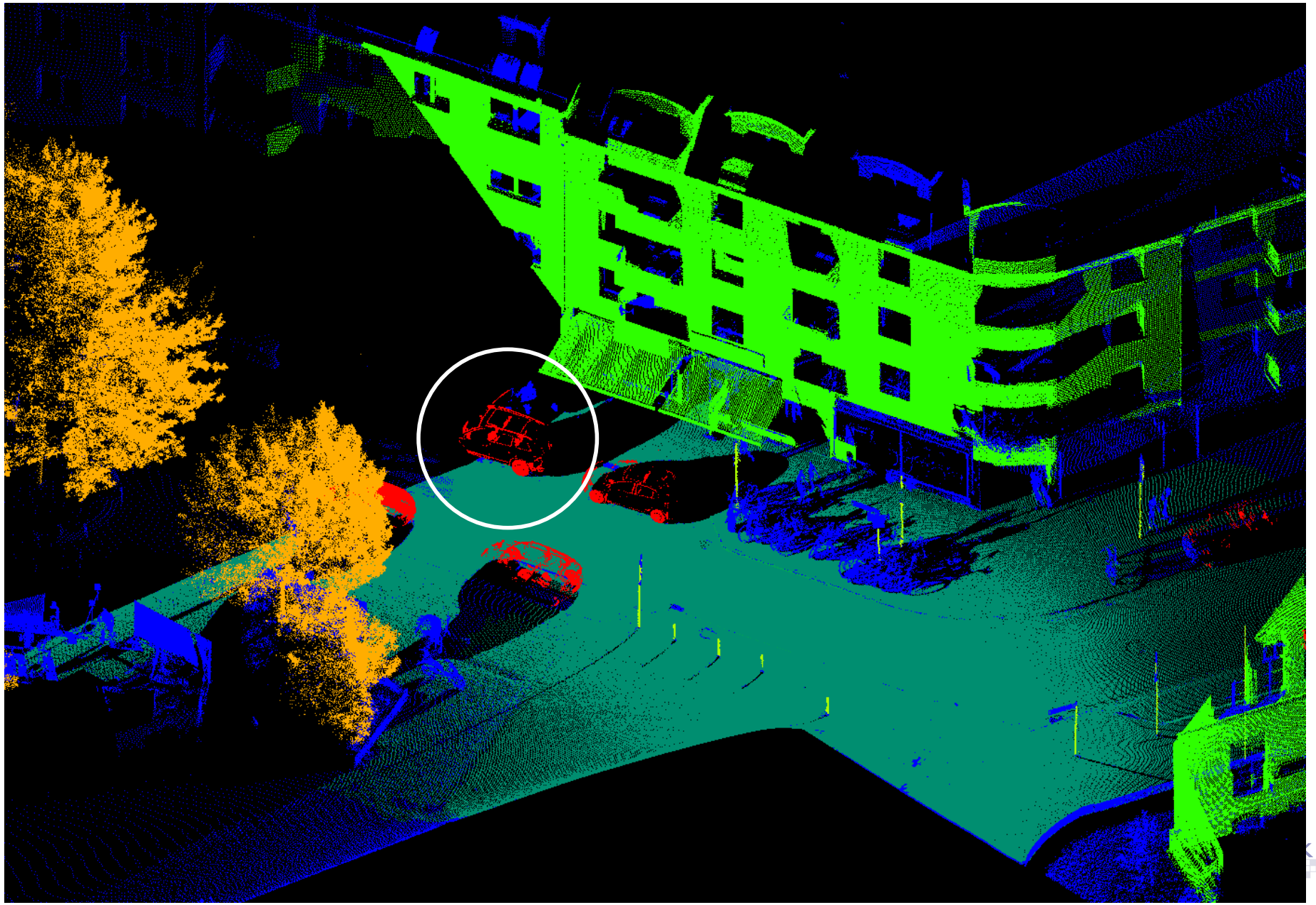
Test result with more complete labelling

- Training with more complete labelling, training : test = 70 : 30
- Results are again very good

```
Training: reading 210903_123059_Scanner_1
  Labels: 2172356
Total labels: 2172356
Class statistics
          Class   Count
         ground 1171855
           curb    5377
       building  581290
      pole-like    6341
           tree  302763
 closed-vehicle  104730

Test performance
          Class Precision    Recall  F1 score
         ground     0.999     1.000     0.999
           curb     0.945     0.862     0.901
       building     0.986     0.980     0.983
      pole-like     0.981     0.952     0.966
           tree     0.965     0.975     0.970
 closed-vehicle     0.995     0.996     0.995
```



Test with another scan strip of the same area: Same scene, scan taken 16 minutes later

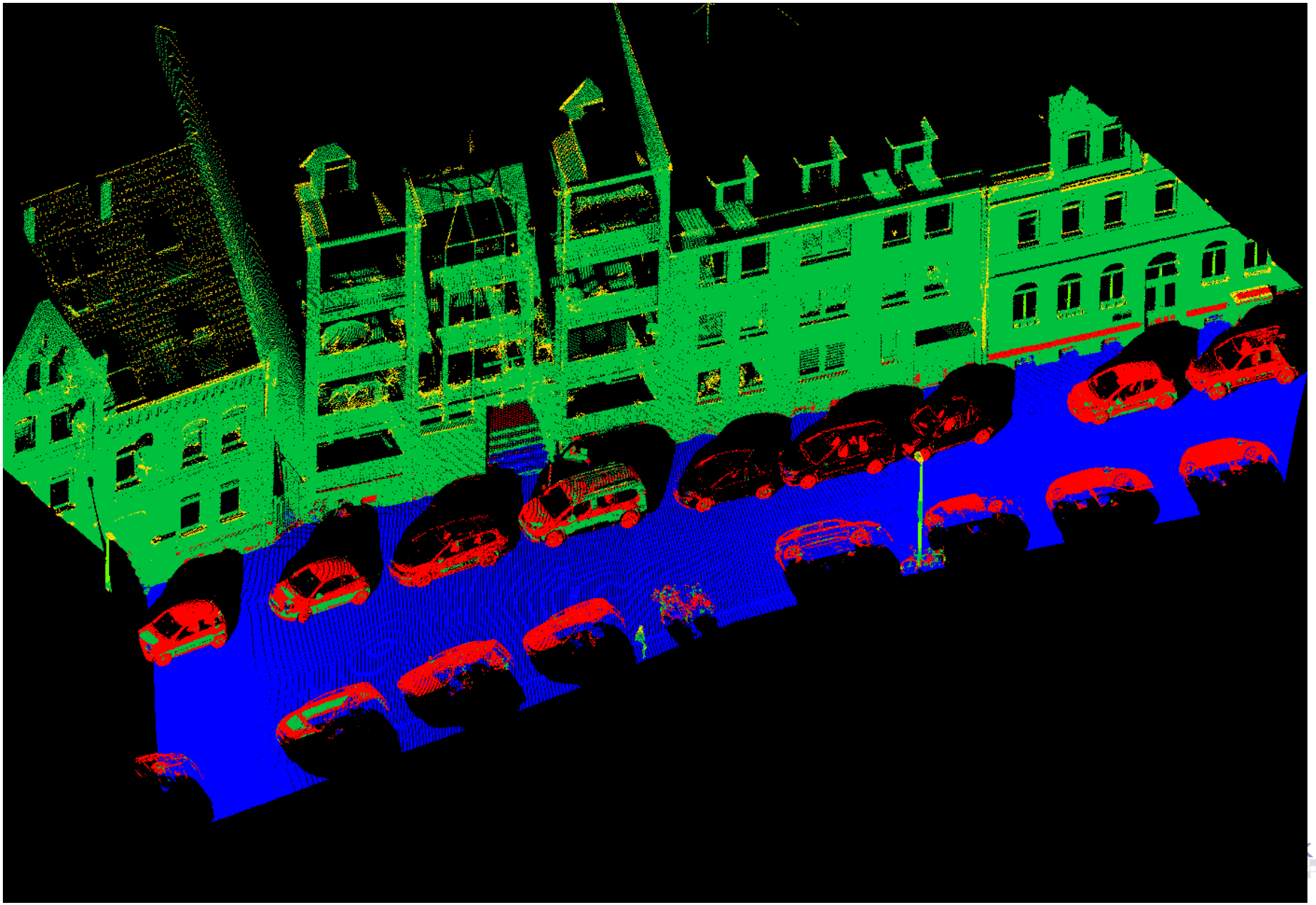


Numerical results

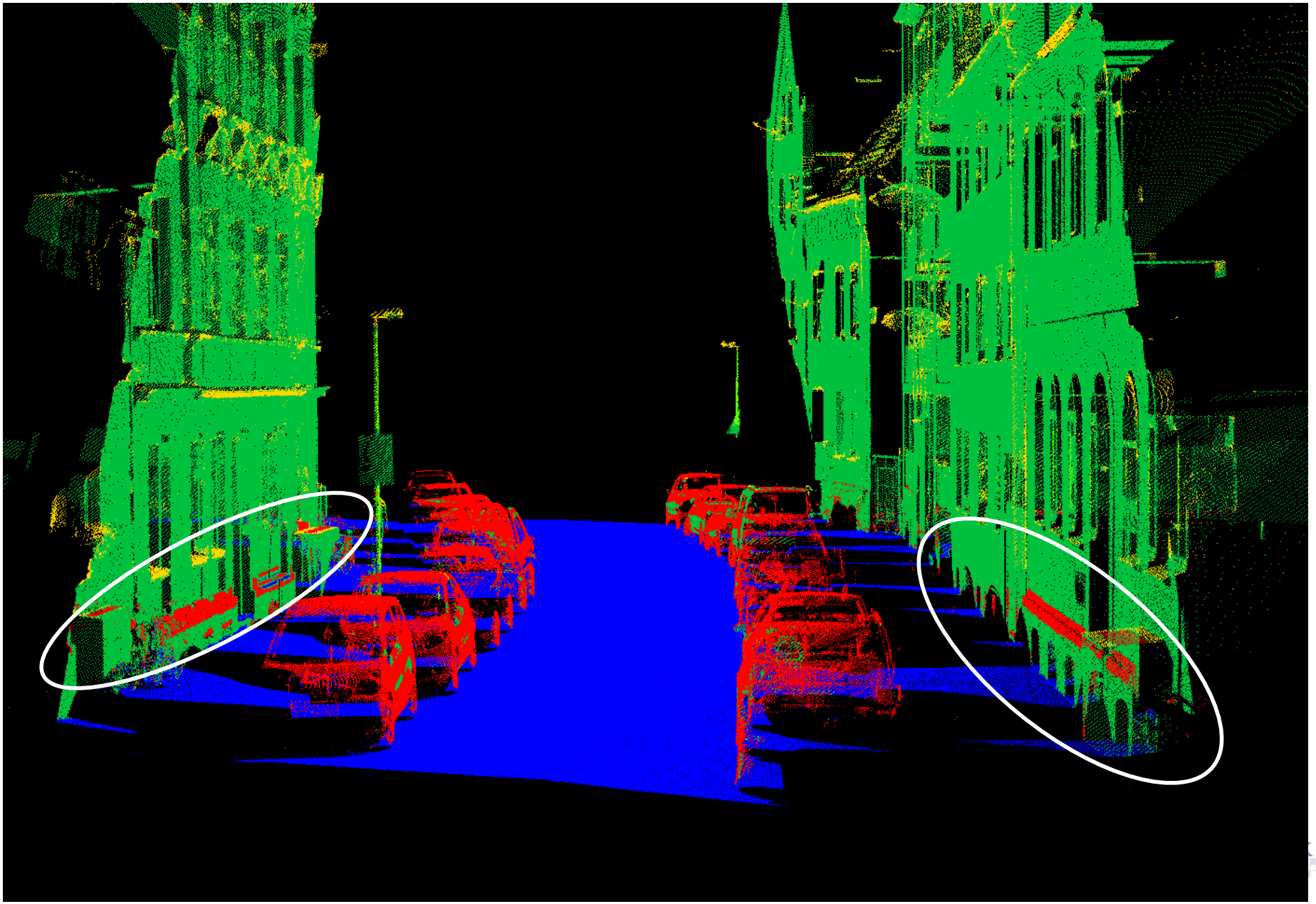
- Test results look OK
- Does this mean our classifier “generalizes well”?

```
File: 210903_124650_Scanner_1
          Class Precision    Recall  F1 score
         ground     0.998     1.000     0.999
           curb     0.927     0.709     0.803
       building     0.979     0.975     0.977
      pole-like     0.982     0.924     0.952
           tree     0.958     0.962     0.960
 closed-vehicle     0.980     0.987     0.983
```

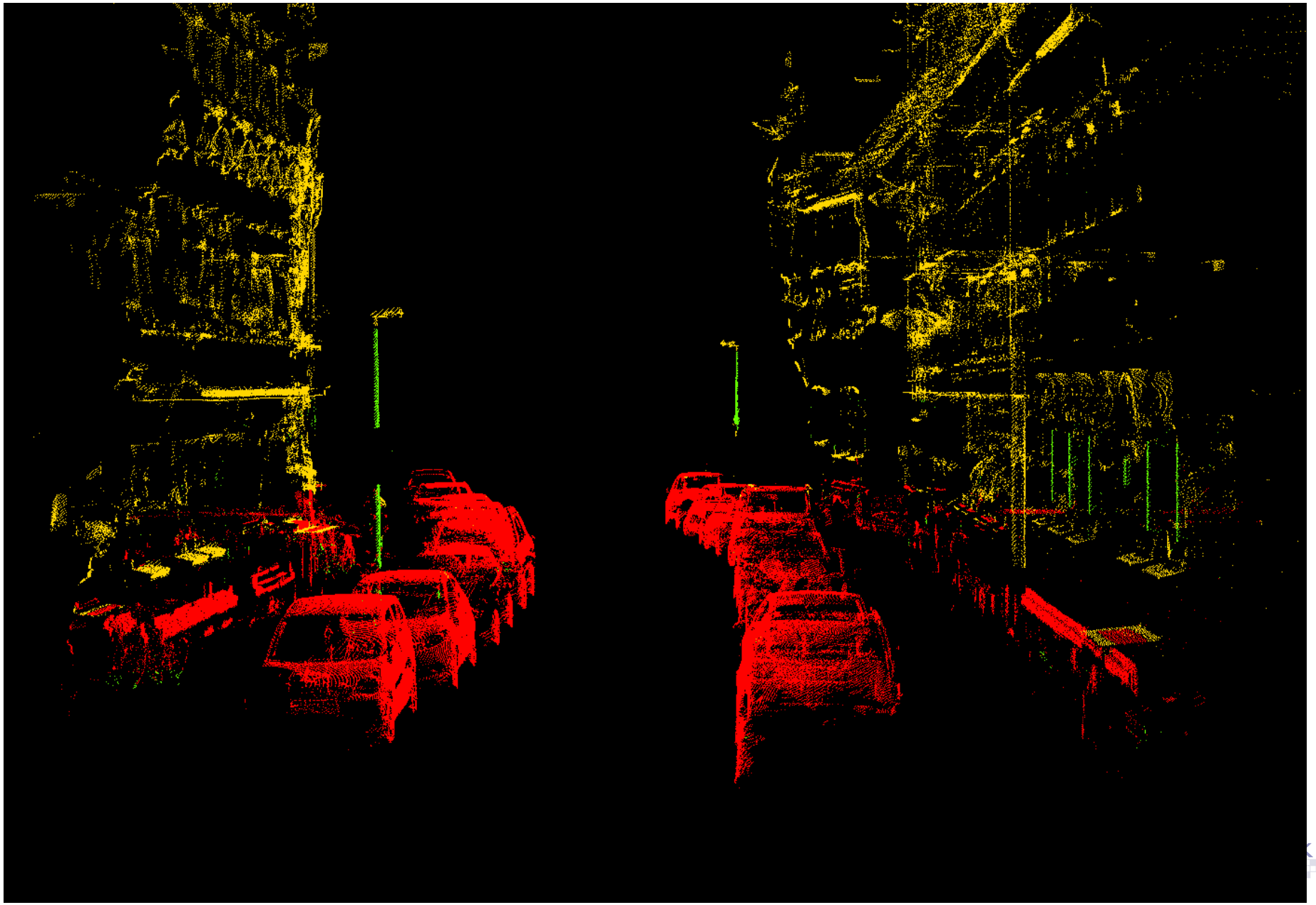
Test with another scan strip of a different area



Test with another scan strip of a different area



Test with another scan strip of a different area



Test result for this strip

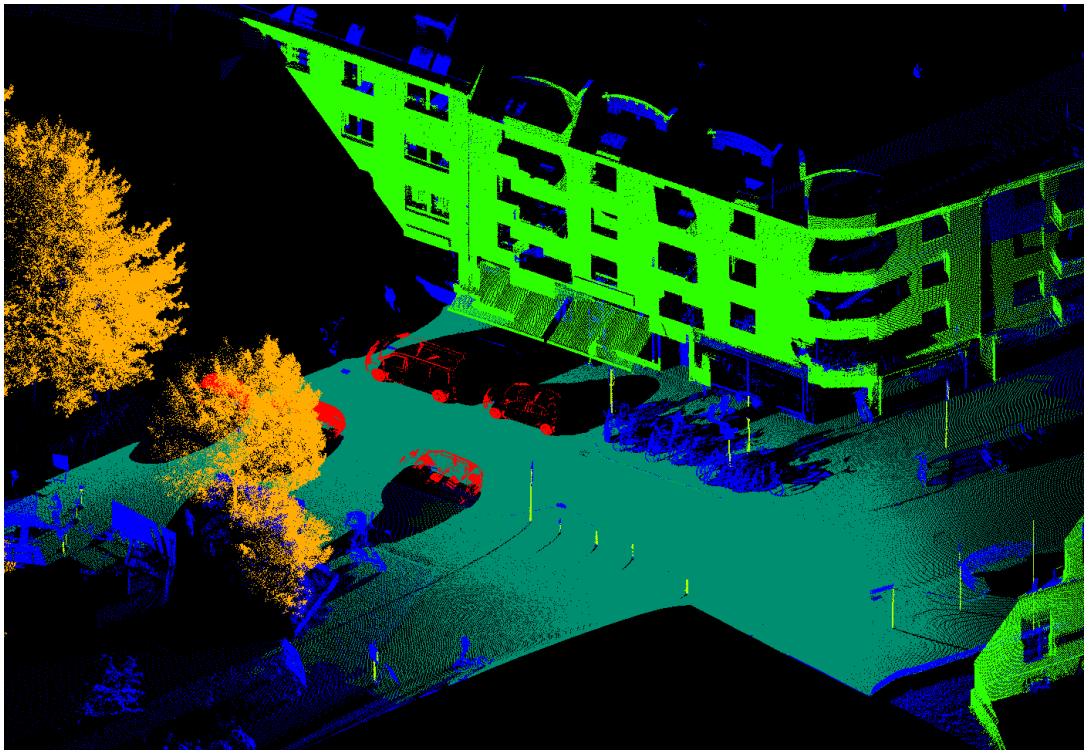
- For the different location, results are again much worse
- It seems that “relative height” has a strong influence on the prediction of “closed-vehicle” (see previous slide)
- So even though it generalizes OK on one location, it does not generalize well to other places

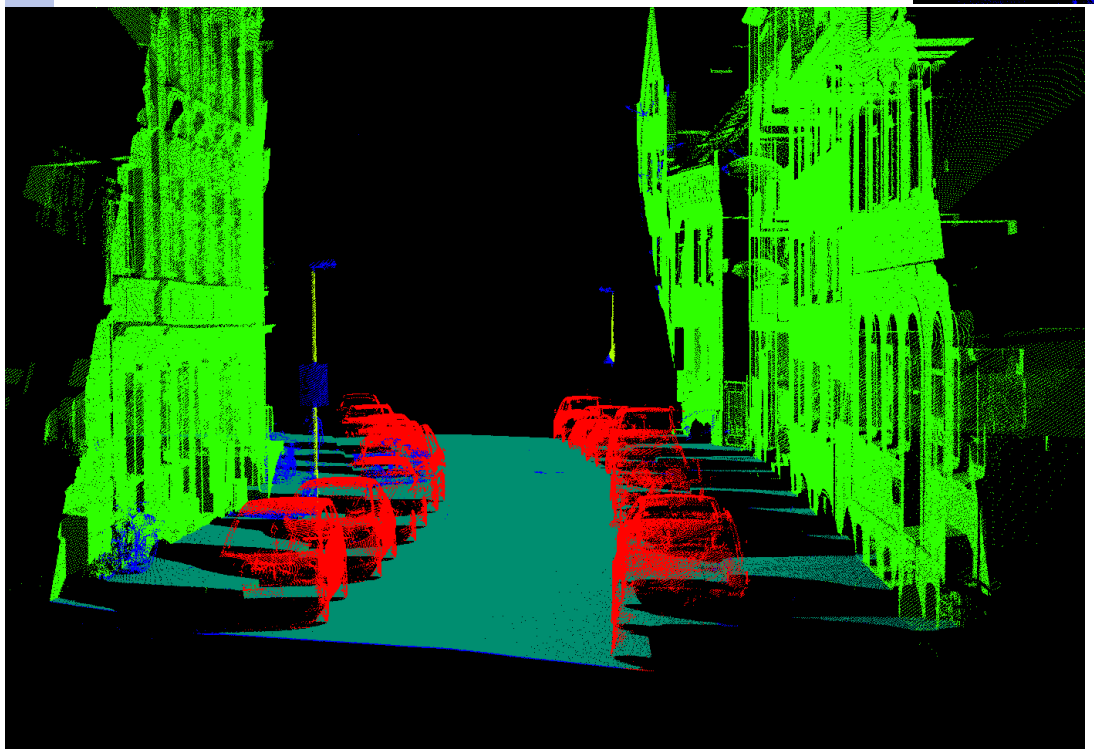
File: 210903_130259_Scanner_1

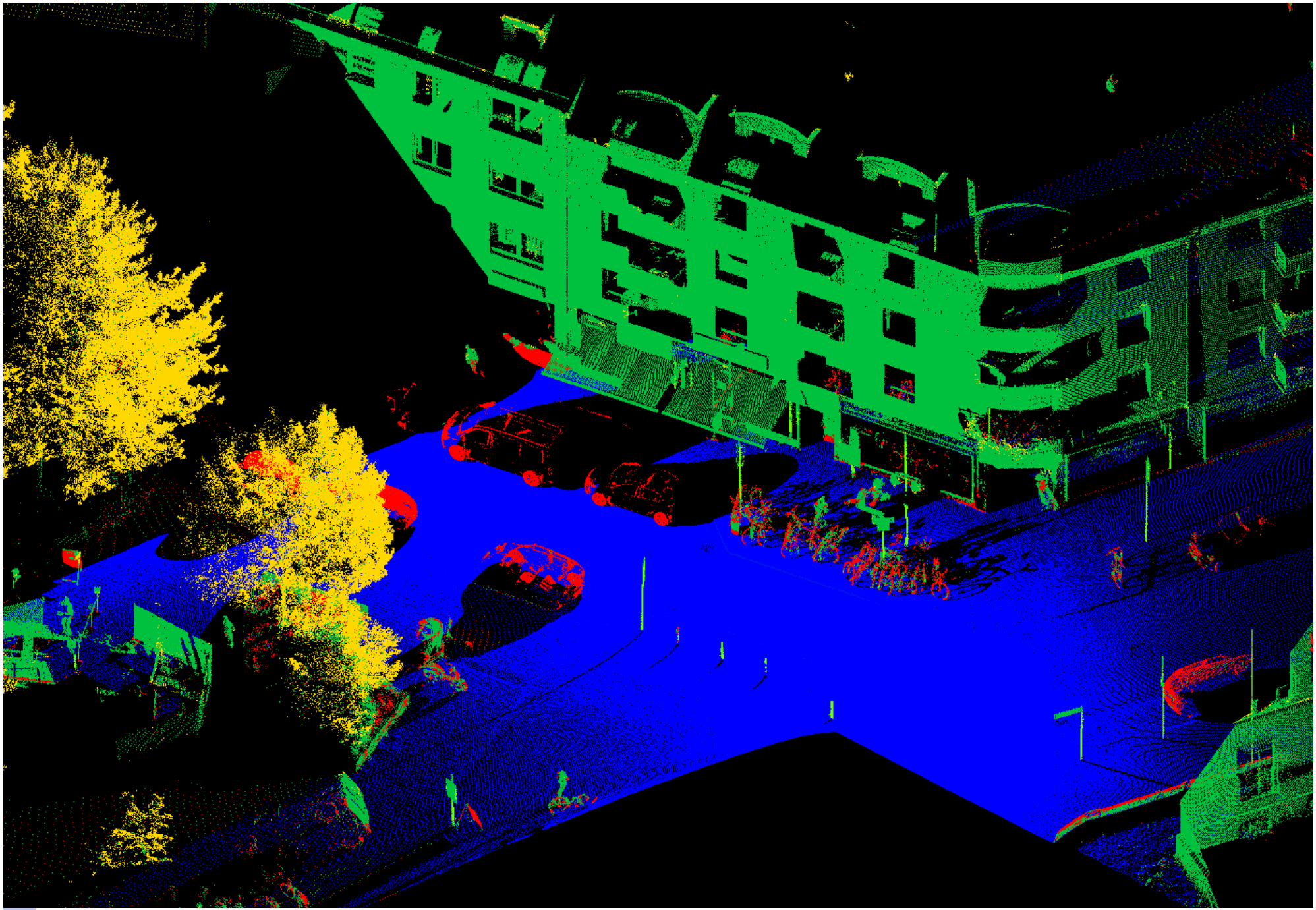
Class	Precision	Recall	F1 score	
ground	0.977	1.000	0.988	
curb	0.189	0.010	0.018	
building	0.981	0.924	0.951	
pole-like	0.604	0.803	0.690	
tree	0.000	0.000	0.000	(no trees)
closed-vehicle	0.889	0.867	0.878	

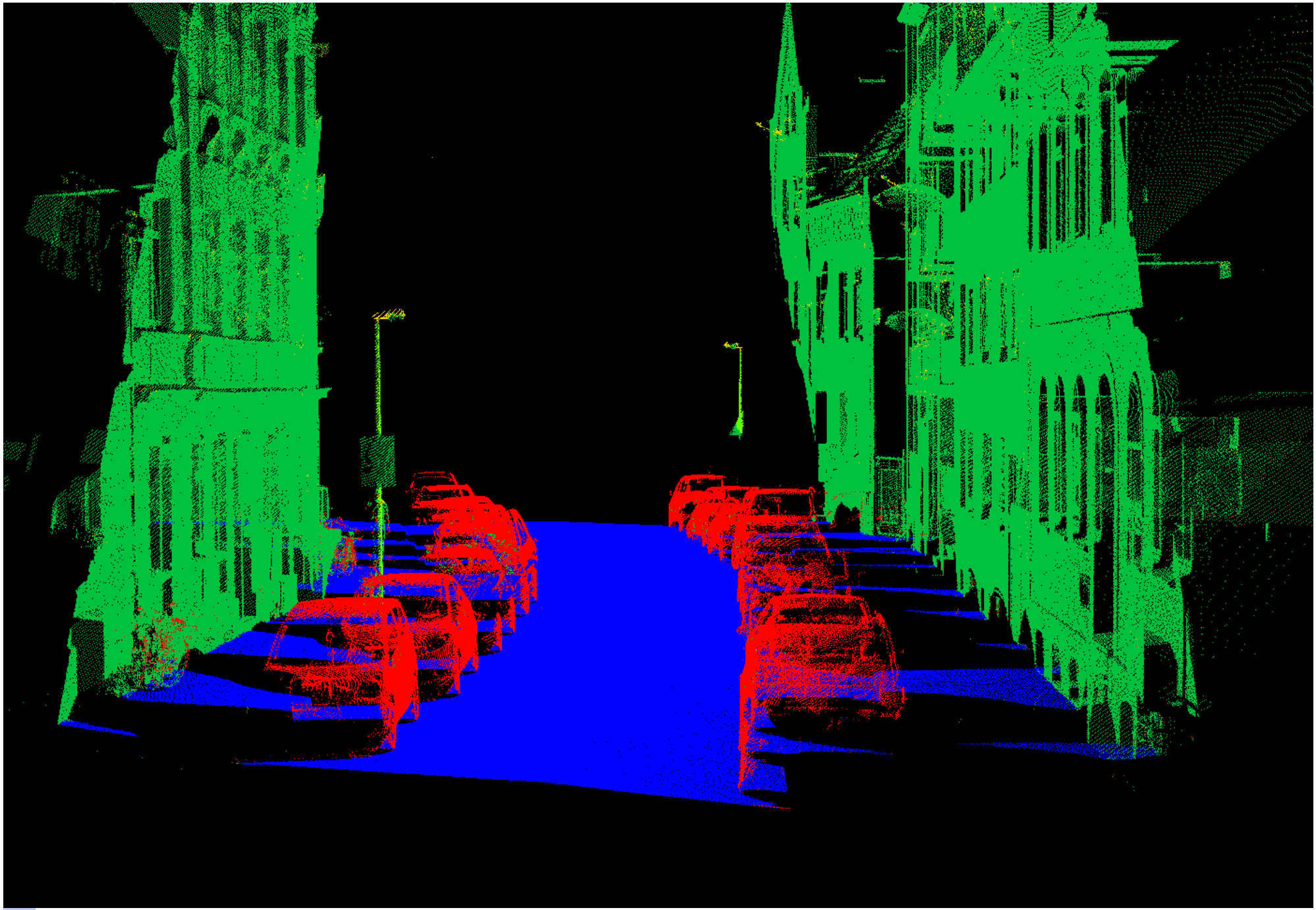
Third experiment:
Train using several scan strips

Train using two scenes







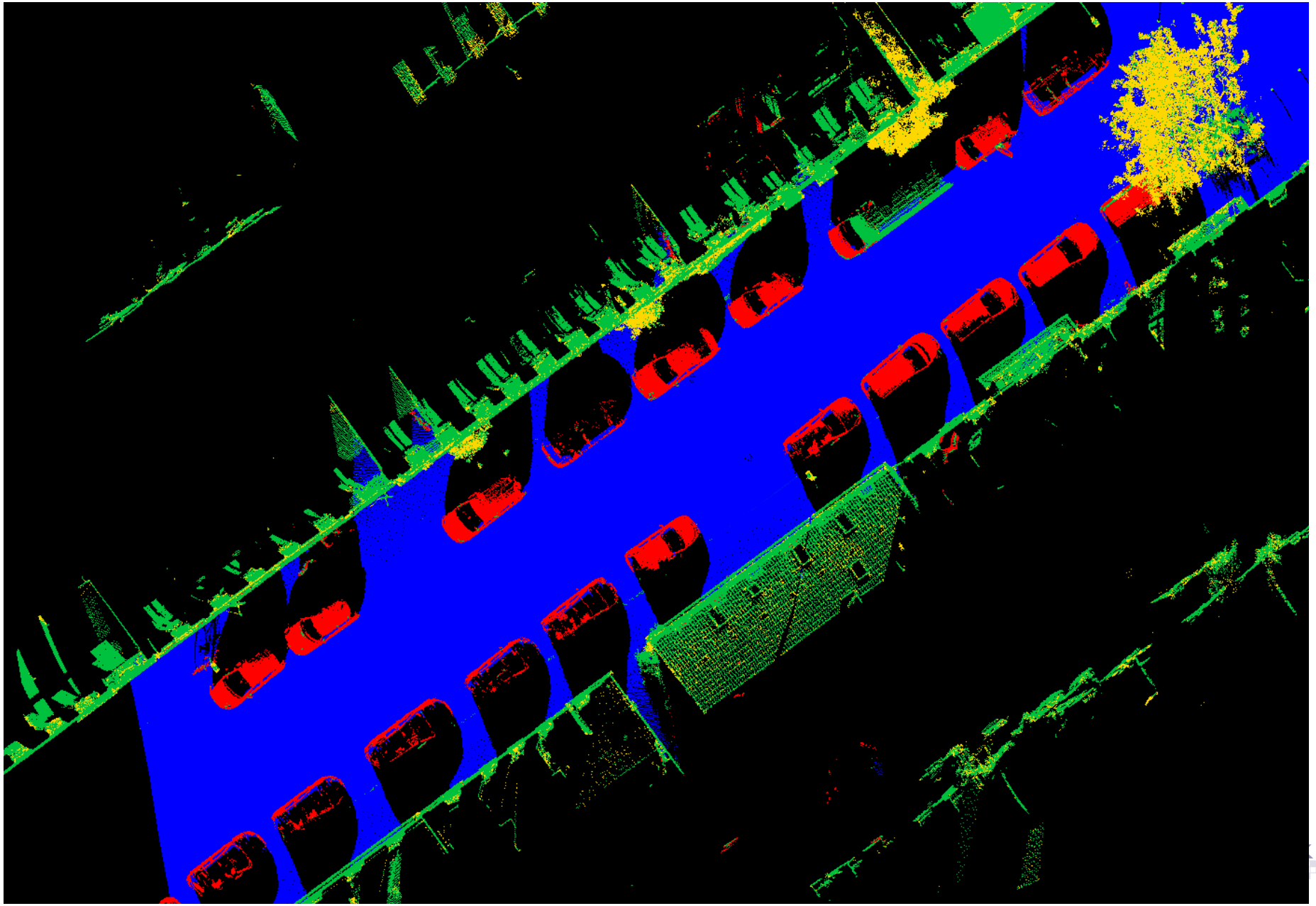


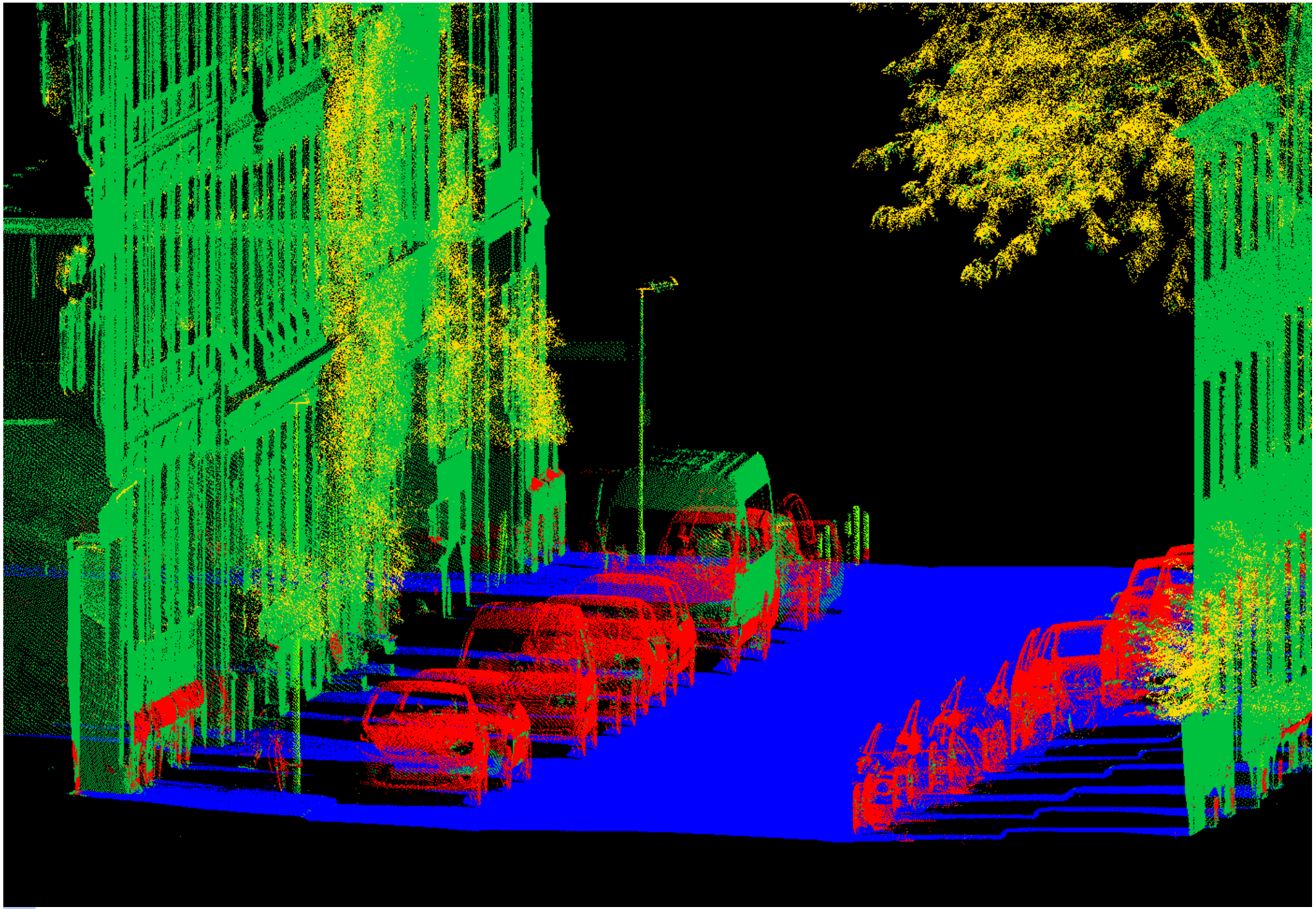
Numerical results

- (Note: too optimistic because this tests on training data.)

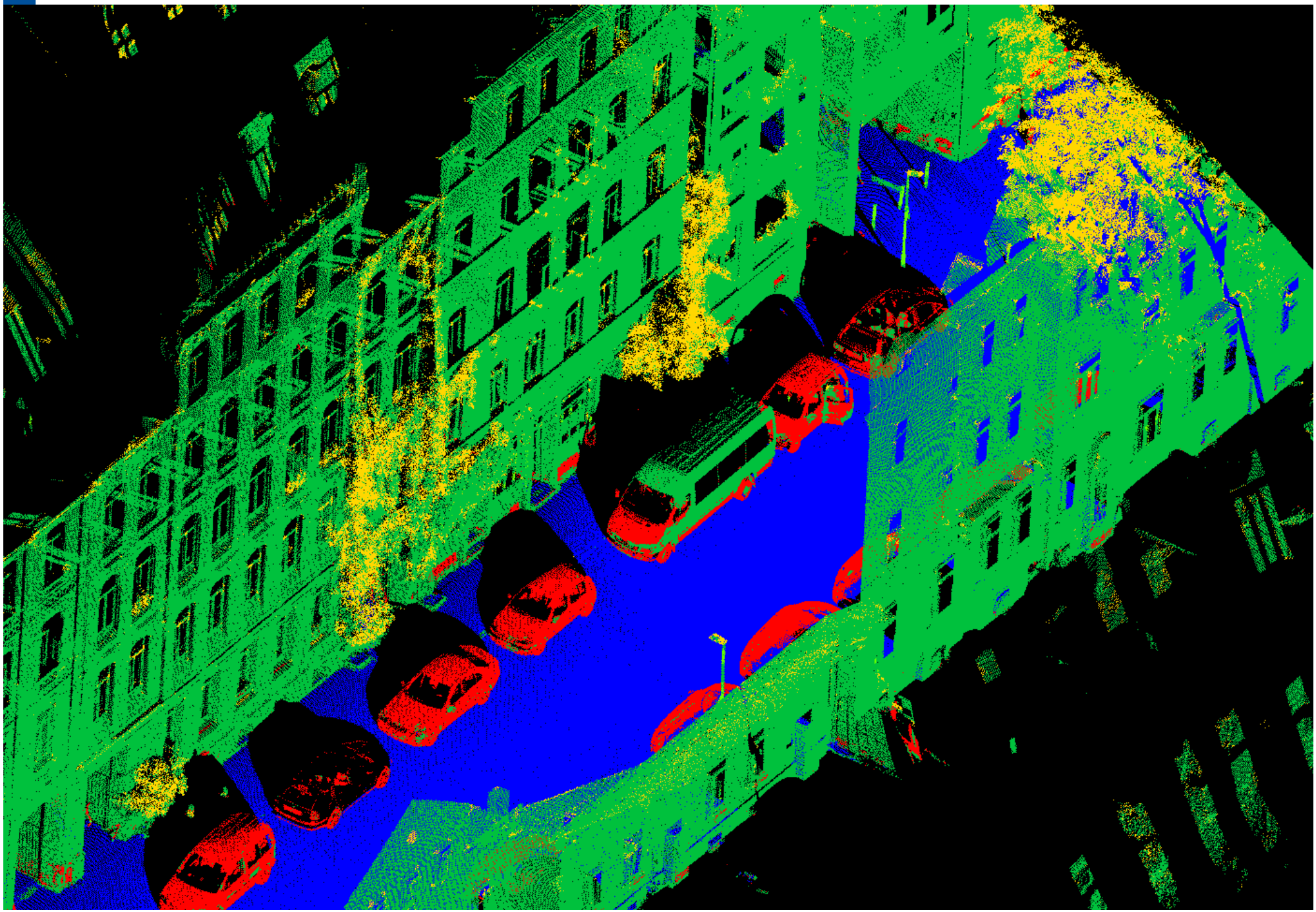
```
File: 210903_123059_Scanner_1
Class Precision    Recall  F1 score
         ground     1.000     1.000     1.000
           curb     0.981     0.956     0.968
       building     0.988     0.996     0.992
      pole-like     0.997     0.978     0.987
           tree     0.993     0.978     0.985
 closed-vehicle     0.996     0.998     0.997

File: 210903_130259_Scanner_1
Class Precision    Recall  F1 score
         ground     0.999     1.000     0.999
           curb     0.974     0.973     0.974
       building     0.999     0.996     0.997
      pole-like     0.990     0.993     0.991
           tree     0.000     0.000     0.000   (no trees)
 closed-vehicle     0.996     0.993     0.994
```

…and a last test: visual result on another scan strip

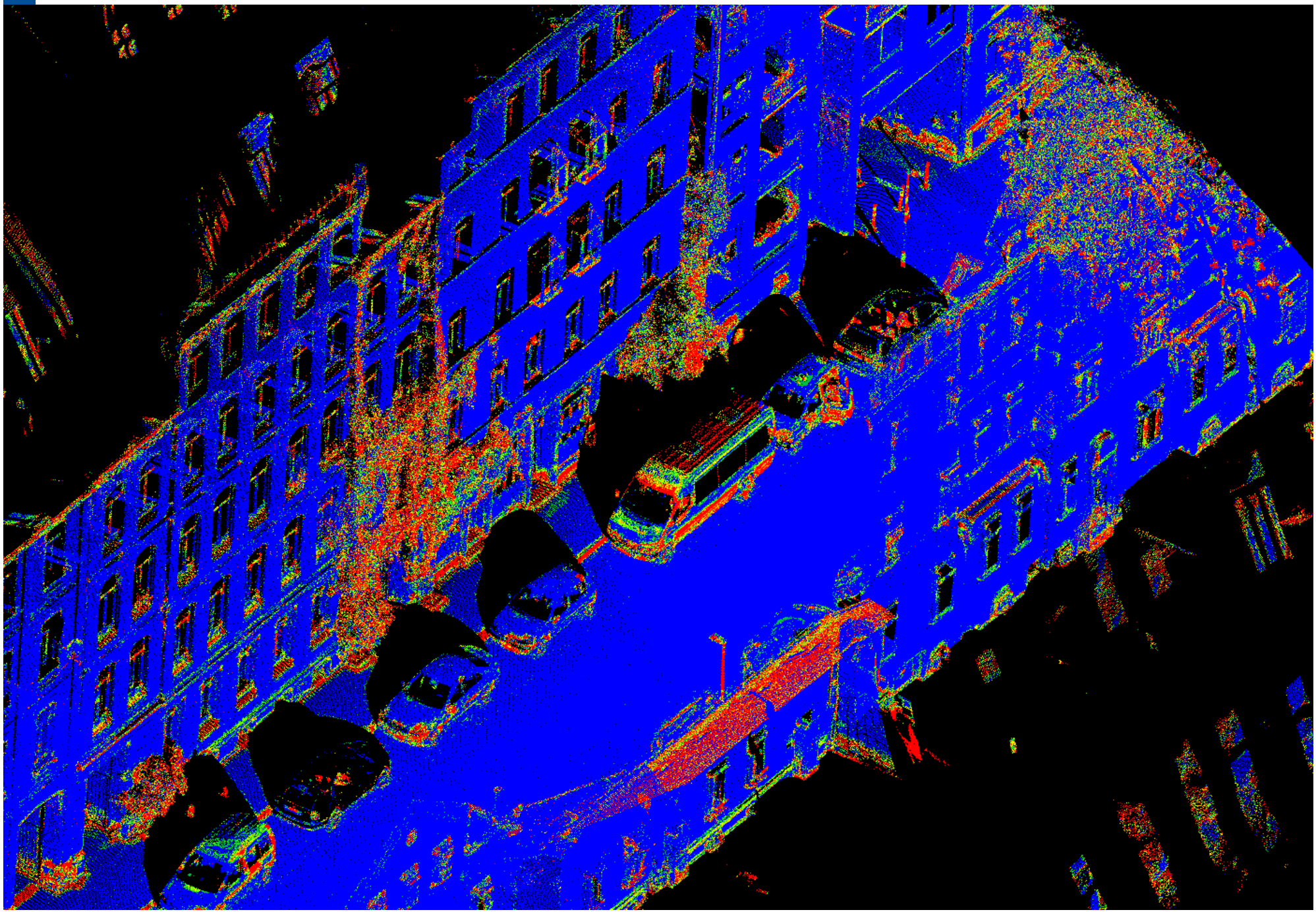




Random forest entropy as a “quality indicator”



Random forest entropy as a “quality indicator”



Concluding remarks

Concluding remarks

- WEKA allows to test many different classifiers in an interactive environment, using a GUI
- Cons:
  - Need to write data in WEKA format
  - Slow, needs lots of memory
  - Loads all data into memory (bad for huge datasets)
- Currently quite popular: scikit-learn
  - Uses/ integrates with popular "data science" languages and libraries (Python, Numpy, Matplotlib, Pandas,…)
  - Efficient (e.g., random forest uses multi-threading).

References

- Criminisi, A., Shotton, J., Konukoglu, E. (2011): Decision Forests for Classification, Regression, Density Estimation, Manifold Learning and Semi-Supervised Learning. Microsoft Research technical report TR-2011-114
- Witten, Ian H., Frank, E., Hall, Mark A. (2016): Data Mining: practical machine learning tools and techniques, 4th ed., Morgan Kaufmann.
- https://scikit-learn.org/