KAPLAN: A 3D Point Descriptor for Shape Completion

Audrey Richard¹, Ian Cherabier¹, Martin R. Oswald¹, Marc Pollefeys¹,² and Konrad Schindler¹
¹ETH Zürich, ²Microsoft Mixed Reality & AI Zürich Lab

Abstract

We present a novel 3D shape completion method that operates directly on unstructured point clouds, thus avoiding resource-intensive data structures like voxel grids. To this end, we introduce KAPLAN, a 3D point descriptor that aggregates local shape information via a series of 2D convolutions. The key idea is to project the points in a local neighborhood onto multiple planes with different orientations. In each of those planes, point properties like normals or point-to-plane distances are aggregated into a 2D grid and abstracted into a feature representation with an efficient 2D convolutional encoder. Since all planes are encoded jointly, the resulting representation nevertheless can capture their correlations and retains knowledge about the underlying 3D shape, without expensive 3D convolutions. Experiments on public datasets show that KAPLAN achieves state-of-the-art performance for 3D shape completion.

1. Introduction

Common 3D sensing technologies like range cameras, laser scanners or multi-view stereo systems deliver 3D point clouds. These point clouds then constitute the input for downstream computer vision and graphics tasks across a wide range of applications. As such, irregular collections (“clouds”) of 3D points have become a fundamental 3D data representation. Due to deficiencies of the acquisition process (occlusions, specularities, low albedo, matching ambiguities, etc.), real point clouds are typically incomplete: they have “holes” – local regions where no point samples of the object surface are available.

Shape completion is the task to fill such holes, so as to obtain a complete representation of an object’s shape. The aim of our work is to perform shape completion directly on the point cloud, without having to transform it into a memory-demanding volumetric shape representation (e.g., a global voxel grid or signed distance function). In other words, we must learn the shape statistics of local surface patches, so that we can then sample points on the expected surface and fill the holes. With the natural decision to center the patches on existing 3D points, this is equivalent to predicting 3D point descriptors from incomplete data.

The state-of-the-art for local descriptors are learned, convolutional encodings. These can be easily implemented on voxel grids using 3D convolutions, e.g. [11, 12], but are memory-hungry. Another way to deal with unordered points is the approach of PointNet [35, 36], to extract per-point features and aggregate them with permutation-invariant operations. With that representation it is, however, not straight-forward to predict new points for shape completion. It has also been proposed to deal with the orderless structure of point clouds by locally learning 2D descriptors [45]. This is achieved by finding the tangent plane of the object surface, projecting the object points into it, and performing convolutional encoding in the 2D tangent plane. Unfortunately, this only works well if the tangent plane approximation is valid and unique, i.e., when there is a single, smooth surface; but fails to capture high-curvature structures like edges and fine surface details. Moreover, the tangent approximation only makes sense over small receptive fields, which contradicts a main strength of convolutional networks, to learn long-range, object-level context.
To efficiently capture 3D structure without the limitation to a single (tangent) plane, we bring back an old idea in computer vision and computational geometry: project the 3D geometry along multiple directions to obtain a set of 2D “images”, which together form the descriptor (e.g., [9]). The core contribution of our work is a new shape descriptor called KAPLAN (see Fig. 1), which combines that classical idea with modern convolutional feature encoding. Instead of a single 2D projection or expensive 3D feature aggregation, we use $K$ projections with different orientations and concatenate them, such that the feature extractor can use standard 2D operations. Importantly, all $K$ projections are encoded jointly, such that the descriptor is still optimised for 3D “awareness”, not independently per projection. Our contributions can be summarised as follows:

- We present a novel shape completion approach that fills in 3D points without costly convolutions on volumetric 3D grids. The approach operates both locally and globally via a multi-scale pyramid.

- To that end, we design KAPLAN, a 3D descriptor that combines projections from 3D space to multiple 2D planes with a modern convolutional feature extractor to obtain an efficient, scalable 3D representation.

- The combination of KAPLAN and our multi-scale pyramid approach automatically detects the missing region, and allows to perform shape completion without needing to regenerate the whole object.

While our target application is shape completion, KAPLAN can potentially be used for a range of other point cloud analysis tasks. It is simple, computationally efficient, and easy to implement; but still leverages the power of deep convolutional learning to obtain expressive visual representations.

2. Related Work

Traditional shape completion approaches. Early works fill holes either directly on a triangle mesh [8], or they use a volumetric grid and intrinsically minimize the surface area [13], curvature [23, 24] or exploit object symmetries [47, 42]. Pauly et al. [33] perform example-based completion using a shape database. These works capture local surface properties and mostly perform interpolation.


Learned shape representations. Several works proposed to learn implicit shape representations that can be leveraged to complete shapes. Ladicky et al. [26] estimate an iso-surface from an point cloud with a random forest. Similarly, deep neural networks have been used to predict occupancies [30, 6] or signed distance functions [32]. The aforementioned methods rely on global feature extraction, often at the cost of local structures. DISN [53] addresses this by adding a local feature extractor module. While these works need explicit supervision, Sitzman et al. [40] proposed an unsupervised approach using images with known poses.

Point cloud-based learning and descriptors. A series of works studied how neural networks can be applied to unstructured data like point clouds. We only highlight some relevant works and refer to [16, 2] for more comprehensive surveys. As one of the first methods, PointNet [35] uses per-point multi-layer perceptrons and max-pooling to compute features on point clouds. PointNet++ [36] extends that idea with the introduction of a hierarchical structure that gradually aggregates features. Both networks aggregate information via spatial max-pooling which negatively affects the localisation of features. The same idea has also been shown to work well for point cloud denoising [38]. Our approach follows a similar idea, but generalizes the concept to a larger number of arbitrarily aligned patches. SPLATNet [44] projects features of the point cloud onto a high-dimensional lattice where convolutions are performed. The network jointly aggregates 2D and 3D features and uses sparse representation for efficiency. PointCNN [27] follows a hierarchical network design and lifts the points to a higher-dimensional space before computing convolutions on local point neighborhoods. SparseConvNet [14] computes 3D convolutions efficiently by means of a sparse implementation of the voxel grid, binning the input points.
into a hash-table. While sufficiently rich for tasks like segmentation, these features seem less suitable for shape completion, since they deliberately avoid filling empty voxels to preserve sparsity. PointConv [52] introduces a learned, weighted convolutional kernel which is used to define learning operations on a 3D point cloud. KPConv [48] defines the concept of kernel point convolutions for direct feature computation on point clouds. FPConv [28] reduces the feature computation on point clouds to 2D convolutions, via a learned flattening operation. Similar to our work, Point-PlaneNet [34] considers local and global feature aggregation using planes, however, they learn plane orientation and their feature aggregation is inspired by PointNet [35]. In [54] an encoder-decoder architecture is used to extract global features from the incomplete data and describe them into a complete object that does not retain the input points. Similarly, in [1] an autoencoder is designed which, if trained appropriately, will synthesise a complete model when fed an incomplete one. The related [39] additionally uses reinforcement learning to better control the (adversarial) loss function. In [5], separate encoders are trained for complete and incomplete scans, which are then used to train a GAN that complete point clouds. Encoders are not a necessity, as has been shown in [46] where the network consists of a single decoder. Strategies were proposed that focus on the holes, without altering the observed parts of the point cloud [49, 22]. Unlike our approach via a new 3D point descriptor, they directly feed 3D points into custom network architectures trained in an adversarial manner. In [29], a coarse completion is first extracted, and then fused with the input cloud via sampling. A Skip Attention Network was proposed in [51] to also address this issue.

3. Method

Our method starts from an (incomplete) set of 3D points. We assume that for each point a normal vector is either available or can be estimated, noting that the method is generic and can be readily adapted to use other point attributes such as colour or laser intensity. The core of our shape completion is KAPLAN, an efficient learnable descriptor to represent 3D shapes based on irregularly sampled points. Since the descriptor is learnable, it can be trained to always output an encoding of the complete local object geometry, even if fed with an incomplete point cloud. Thus, the output descriptor represents the expected geometry, rather than the actually observed one. This autoencoder-like behaviour makes it possible to use KAPLAN for shape completion, by inverting the descriptor back to a point cloud. The shape completion pipeline is embedded in an explicit multi-resolution framework. In this way, the method starts by filling large holes using coarse, global context and gradually refines and densifies the point cloud with more fine-grained, local context. In the following, we detail the descriptor and associated network architecture. The overall process is depicted in Fig. 3.

3.1. From 3D points to 2D images

This part corresponds to the “Point Cloud Processing” section in Fig. 3. Our aim is an efficiently computable descriptor, hence we aim to avoid volumetric computations that involve convolutions on 4D tensors. A simple, but effective trick is to project the 3D geometry onto a 2D plane along some “canonical” direction. One implementation of that principle are Tangent Convolutions [45] which use the local surface normal as the canonical direction. Note that the projection does not greatly increase computational cost, as the normals and the associated projections are fixed and can be precomputed. A disadvantage of a single projection from 3D space onto the tangent plane is the implicit assumption that the object geometry can locally be parametrised as a single function over a 2D domain. When this condition is not fulfilled, information will be lost. A classical trick to mitigate the loss of information is to project onto multiple different planes, thus creating a “multi-view” representation of the 3D geometry. This strategy used to be popular in the early days of 3D object representation and retrieval, e.g., [9]. More recently, it has been used in the context of deep learning, for semantic segmentation of 3D point clouds, e.g., [3].

KAPLAN combines the two ideas: the geometry is projected along \( K \) directions to minimize the loss of 3D information, the set of projections then serves as input for a convolutional architecture that encodes it into a feature representation. We point out a further, more subtle advantage of using multiple projections: contrary to [45], it is no longer necessary to align the projection plane with the local surface, the descriptor therefore is less affected by inaccurate point-wise normals. Normals can still be used as input, but are no longer crucial to the method. This is a considerable benefit in our application, as normal estimation is unstable on the border of data gaps, where there are no supporting 3D points. Also note that \( K \geq 3 \) projections are enough to avoid problems at sharp crease edges and 90° corners, which are frequent on man-made objects. In our implementation, we align the KAPLAN coordinate system with the gravity direction, which is known for most 3D point clouds.

Fig. 2 illustrates the accumulation of 3D point information into 2D projection planes. Empirically, already the minimal configuration with \( K = 3 \) planes in canonical orientation provides a good representation. If necessary one can increase \( K \), either systematically (e.g., rotating planes around each coordinate axis in fixed steps) or by randomly drawing plane normals. For each plane, nearby points are selected according to a box constraint to preserve locality (red box in Fig. 2b) and projected orthogonally onto the plane to form a depth image and a normal image. The 2D
projection plane is discretised into a grid of $R \times R$ cells ("pixels"), if multiple points are projected into the same cell their normals and depth values are averaged. Additionally, we record a valid flag image that indicates whether any points have been projected to a given cell (value 1) or whether the cell is empty (value 0). The $c = 5$ channels in each plane (depth $\times 1$, valid flag $\times 1$, normal $\times 3$) are stacked into a tensor $K^0$ of dimension $(R \times R \times Kc)$. Note that, while KAPLAN can in principle be constructed at every 3D point, in practice we only sample descriptors at $Q$ query points, since nearby descriptors are highly redundant.

### 3.2. Descriptor Network

To map the raw point properties in $K^0$ into a shape representation that preserves the geometric layout, we pass it through a convolutional autoencoder, as shown in the "K-Planes Processing" section in Fig. 3. In our implementation, we chose a variant of U-net with two rounds of pooling in the encoder and the corresponding two rounds of unpooling in the decoder. We use MISH activations [31] after every convolution layer instead of ReLUs for better accuracy. The decoder has three separate heads (with identical layer structure) for the three modalities depth, normal and valid flag, whose outputs are stacked back together into the final descriptor $K$ as illustrated in Fig. 2e.

Since we apply KAPLAN for shape completion, the main task of the network is to fill empty descriptor cells whereas the other cells remain unchanged. We therefore add a skip connection directly from the input $K^0$ to the output $K$, such that the network can focus on learning to fill in the empty cells. Note that the predictions for the valid flag are continuous values in $[0, 1]$, for further processing we threshold them at 0.5. If a KAPLAN cell was empty in the input, but is predicted as non-empty, we reconstruct the corresponding 3D point from the predicted depth value.

### 3.3. Coarse-to-fine Approach

KAPLAN encodes local shape information. In order to employ long-range shape context and fill large holes, we embed the shape completion in a coarse-to-fine scheme.

We start at an initial, coarse level $\ell^0$, where the size of the KAPLAN neighbourhood (defined by the box constraint) is the bounding box size of the full object. At this level, the descriptors capture global context. As at most one point per descriptor cell is reconstructed, the first level will result in a point cloud that still has low density in the newly completed regions, but that is complete, in the sense that a minimum point density is ensured everywhere on the surface. That initial reconstruction can now guide the next hierarchy level: the newly generated points serve as query points to compute KAPLAN descriptors at the next finer scale and further densify the point cloud. The refinement can be iterated until the desired point density has been reached. In our implementation, we use three levels (see Fig. 3). In practice, instantiating a KAPLAN at every newly created point from the previous level would be highly redundant, so we subsample the query points with a spatial filtering block (see Sec. 4.3 and supplementary material).

We train the coarse-to-fine scheme sequentially, starting at the coarsest level. Back-propagation across hierarchy levels is not possible, because the instantiation of new points from the KAPLAN projections is not differentiable.

### 3.4. Loss Function

The learnable parameters of KAPLAN at each hierarchy level are the autoencoder weights. To train them we minimize the following loss function:

$$L = \lambda_v L_v(V, \hat{V}) + \lambda_d L_d(D, \hat{D}) + \lambda_n L_n(N, \hat{N})$$  \hspace{1cm} (1)

The loss is composed of three terms, one for each modality. For the valid flag, we use a standard $L^2$ regression loss between the predictions $\hat{V}$ and the ground truth $V$, computed over all pixels of the projection plane $i \in \Omega \subset \mathbb{R}^2$:

$$L_v(V, \hat{V}) = \frac{1}{K} \cdot \frac{1}{|\Omega|} \sum_{k \in [1,K]} \sum_{i \in \Omega} \|V_i - \hat{V}_i\|_2$$  \hspace{1cm} (2)

For the depth, we compute a masked $L^2$ loss between the predictions $\hat{D}$ and the ground truth $D$, only at non-empty cells $j \in \Omega_m \subseteq \Omega$ according to the ground truth valid flag:

$$L_d(D, \hat{D}) = \frac{1}{K} \cdot \frac{1}{|\Omega_m|} \sum_{k \in [1,K]} \sum_{j \in \Omega_m} \|D_j - \hat{D}_j\|_2$$  \hspace{1cm} (3)
Figure 3: **Overview of our multi-scale pipeline.** It exhibits the three levels: coarse level $\ell^0$, medium level $\ell^1$ and fine level $\ell^2$. Each level consists of the same autoencoder: a single shared encoder and three separate decoders for each modality.

For the normals, we penalize the angular deviation between the predictions $\hat{N}$ and the ground truth $N$, following [37]. As for the depth, the loss is computed only over non-empty cells using the ground truth valid flag:

$$L_n(N, \hat{N}) = \frac{1}{K} \frac{1}{|\Omega_m|} \sum_{k \in [1, K]} \sum_{j \in \Omega_m} \left( 1 - \frac{\langle N_j, \hat{N}_j \rangle}{\|N_j\| \|\hat{N}_j\|} \right)$$

(4)

In practice, we use $\lambda_v = 0.75$, $\lambda_d = 1$ and $\lambda_n = 0.01$.

4. Implementation Details

This section elaborates on implementation details of KAPLAN (see also the supplementary material).

4.1. KAPLAN Precomputation (3D to 2D)

During KAPLAN generation all points within the bounding box are projected into cells on the plane (c.f. Sec. 3.1) and attributes like depth and normals are averaged per cell. However, even with the box constraint it can still happen that projected points originate from different surfaces. In that case naive averaging would entail information loss, e.g., the depths of two distinct surfaces cannot be recovered from their average. To prevent that we apply a simple heuristic that bins the points into distinct surfaces and projects only the points on the surface with the lowest depth.

4.2. Point Prediction (2D to 3D)

The network predicts a complete KAPLAN, which is a set of 2D image channels. To convert the descriptor back to 3D points, one simply lifts the center point of each cell to the appropriate depth and transforms the resulting 3D point back to the global coordinate system. Our goal is to complete missing regions, whereas we aim to avoid oversampling regions already covered by existing 3D points. To decide where to introduce new points, the simplest criterion is to look for KAPLAN cells where the valid flag was 0 in the input and got switched to 1. In practice this criterion is too strict, as even cells in regions of missing geometry will sometimes not be completely empty. For instance, if another surface is in the range of the descriptor, its points will project into the 2D image of the hole, which then would not be recognised. Thus, we additionally instantiate new 3D points for cells that were marked valid in both the input and the output, but where the predicted depth significantly differs from the input depth.

4.3. Prediction Filtering

As described in Sec. 3.3, we filter output points of a coarser hierarchy level before passing them to the next-finer level, to avoid redundant points and to remove outliers. To integrate information across multiple KAPLAN descriptors and ensure consistency, we use a volumetric representation only **locally** around the predicted hole regions. Note that the “voxels” serve only to collect points in a small 3D region – the filtering looks at them one-by-one, so one need not store an explicit voxel space.

**Inter-KAPLAN consistency.** In order to remove outliers, we keep track which query point (and corresponding KAPLAN) spawned each predicted point. Points not supported
by a second prediction from a different query point (falling into the same voxel) are discarded.

**Representative points.** To ensure an even density of generated points, we only take one point per voxel. We compute a weighted average of all predictions falling into a voxel with Gaussian weights inversely proportional to a point’s KAPLAN depth (giving predictions with lower depth higher confidence). We retain the prediction closest to the average and discard all other points in the voxel.

## 5. Experiments

We implemented KAPLAN in Tensorflow and run it on a GTX1080Ti GPU (12GB RAM). We use the Adam optimiser [25] with learning rate $10^{-6}$ and batch size 128. By default, we use the KAPLAN configuration ($K=3, R=35$) found via an ablation study (see supplementary material).

**Data generation.** We use ShapeNet [4] to create our dataset of input-output pairs, i.e., incomplete ($\mathcal{P}_i$) and complete ($\mathcal{P}_c$) versions of the same point clouds. We use the pre-processed data of [30]. Each object comes in the form of a gravity aligned point cloud consisting of 100k points with normals, sampled on the surface, and normalized to a bounding box with largest dimension 1.

These constitute our ground truth at the finest level, $\mathcal{P}^2_c$. We then generate incomplete point clouds with different sizes of holes, by removing $\{2\%, 5\%, 10\%, 20\%, 30\%\}$ points. A point is picked randomly as the “center” of the hole, then the appropriate number of surrounding points $\mathcal{P}^2_m$ is found with a kd-tree search and cut out to obtain $\mathcal{P}^2_i$, such that $\mathcal{P}^2_i = \mathcal{P}^2_m \cup \mathcal{P}^2_c$. In order to create data for the coarser level $\mathcal{P}^1$, it is not advisable to downsample $\mathcal{P}^2_c$ and repeat the same procedure, as this may lead to inconsistencies between the sampling of the fine and coarser levels. Instead, we individually downsample $\mathcal{P}^2_i$ and $\mathcal{P}^2_m$ to obtain $\mathcal{P}^1_i$ and $\mathcal{P}^1_m$. Their union yields the downsampled version of the complete point cloud $\mathcal{P}^1_c = \mathcal{P}^1_i \cup \mathcal{P}^1_m$. The procedure is repeated once more to obtain the coarsest level $\mathcal{P}^0$ (see Fig. 3).

We select five object categories plane, chair, lamp, sofa and table, and split them into training and test sets following [7]. Since our goal is to learn expressive shape priors, not only local surface interpolation, we train a dedicated network for each category, as in [32].

**Baselines.** We compare our method against five baselines:

1. Poisson Surface Reconstruction (PSR) [23] reconstructs a mesh from a point cloud. It can interpolate smooth surfaces with small holes, but not larger missing areas.
2. Point Completion Network (PCN) [54] is a learning-based method which also performs shape completion directly on the incomplete point cloud.
4. Occupancy Networks [30] estimate the occupancy at sampled points in the 3D bounding volume (similar to a binary classifier) to reconstruct an implicit surface.
5. DeepSDF [32] learns to encode a signed distance function of one or multiple shapes in a latent space.

The baselines follow two different principles for shape completion: the first three methods directly reconstruct the missing surface points, without explicitly extracting a shape representation. The last two methods focus on learning a (complete) shape representation, then extract the missing points by feeding the incomplete input into the representation and decoding it into a complete model.

For all baselines, we used implementation and trained models provided by the authors, except for DeepSDF, which we retrained following the authors’ instructions. The code available to us did not include the version used for shape completion, i.e. we did not have access to the exact values of the SDF sampling distance as well as the weight of the free space loss. This explains why our results differ from those presented in their paper.

## 5.1. Results

We conduct different experiments to evaluate the shape completion performance of our KAPLAN. First, we compare it against several baselines, for different levels of incompleteness. We then perform ablation and parameter studies at coarse level $\mathcal{P}^0$ to analyse different KAPLAN configurations. For all quantitative experiments, we used two error metrics: the Chamfer distance (CD) between prediction and ground truth, and the F1-score between predicted and true points. Details about both metrics can be found in the supplementary material.

**Missing region detection.** contrary to some other works, e.g., [17], we do not rely on a separate detection step to find points on the boundary of a hole. Instead, our approach detects the missing data regions during descriptor computation, via the prediction of the valid flags (at the coarsest level). In Fig. 4 we show planes from different KAPLAN descriptors of an airplane, extracted at the coarsest level $\mathcal{P}^0$. Note how the descriptor captures the global shape, for instance in the example on the left, where the projection plane is aligned with the airplane’s wings. The network uses the pixel-wise valid flags to determine where points need to be added, as can be seen from the predicted flags, where the hole has been filled correctly. The predicted normals and depth at those pixels are then used to instantiate new points.

**Quantitative evaluation.** Fig. 5 shows the average F1 and CD values for all object categories, values for individual object instances are given in Fig. 6. For our method, we provide the values separately for each hierarchy level. We recall that for the F1-score, larger values (corresponding to higher precision and recall) are better, whereas for CD,
Our results are shown with a gradation of red, to further distinguish the different coarse-to-fine levels. On the contrary, PSR and ours only fill in new input points and replacing them by necessarily imperfect predictions. On the other hand, KAPLAN appears to strike a good compromise between global object shape and local surface properties, for instance it is able to recover thin structures, such as the legs of a foldable chair or the pole of the lamp, which are not captured correctly by the baselines.

**Ablation and parameter study.** We conducted studies for the airplane category, at level $\ell^0$ with 10 query points.

**Ablation – normals.** To assess here the contribution of the normals in our method, we simply remove them from the input, as well as from the decoder. With normals, we obtain $F_1 = 94.29$, respectively $10^3 \cdot CD = 0.31$. Without normals, these values drop to $F_1 = 93.72$, respectively $10^3 \cdot CD = 0.35$. We conclude that KAPLAN still performs very well for unoriented point clouds (with no normal information). Still, although normal estimation could in principle be learned implicitly, feeding in explicit normals does help the network and leads to more accurate predictions (as seen from the significant decrease of the CD). A possible interpretation is that depth and normals are complementary when it comes to discontinuities and smooth surfaces.

**KAPLAN parameters.** Two experiments are carried out to study the impact of the number of planes $K$ and the resolution $R$. In each experiment, we fix one parameter and vary the other. For $K$, we also vary the plane orientation. We start with the simplest setup, a single tangential plane aligned with the local surface normal. We then test ran-
domly oriented planes, constrained to pairwise angles of at least 30° to avoid degenerate configurations with multiple very similar planes. Lastly, we test “canonical” planes with uniformly distributed orientations.

It turns out that good resolution within a plane is beneficial: from Tab. 1, we observe a significant gain by increasing $R$ from 15 to 35. On the other hand, at most a small improvement is possible when increasing the number of planes beyond 3. Interestingly, randomly oriented planes perform a lot worse than canonical ones, as the network has difficulties to learn the prediction of the valid flag. This could be due to the fact that objects in ShapeNet have main directions aligned to the canonical coordinates. Please see the supplementary material for further analysis.

Figure 6: Qualitative and quantitative comparison to state-of-the-art shape completion methods. The values below the pictures reproduce results from Fig. 5 shown as $10^3 \cdot CD | F_1$. Best values are bold, 2nd best ones are blue. As explained, the DeepSDF results differ from the original paper, since we did not have access to the original code for shape completion.

6. Conclusion

We presented a new approach to shape completion for 3D point clouds. It is based on KAPLAN, a novel 3D point descriptor that locally aggregates features in multiple 2D projections, making the 3D data amenable to standard 2D convolutional encoding and decoding. Importantly, the encoding includes a valid flag to mark unobserved object regions, thus making it possible to fill in new data only where needed, while keeping original samples if possible. We also embed KAPLAN in a coarse-to-fine scheme to reconcile the use of global context with the need for local geometric detail. Empirically, the proposed approach is able to complete missing areas effectively, reaching high accuracy in terms of the predicted point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respectively surface) locations, while keeping existing point (respe


