3D Feature Point Learning for Fractured Object Reassembly

Chaoyu Du (chaoyu.du@arch.ethz.ch), Florian Hürlimann (fhuerliman@student.ethz.ch),
Eshaan Mudgal (emudgal@student.ethz.ch), Ulrich Steger (ulsteger@student.ethz.ch)

ETH Zürich, Rämistrasse 101, 8092 Zürich, Schweiz

Abstract

Reassembly of fractured objects is an important task in archaeology as well as well as medicine. A critical task during reassembly is the identification of suitable features (keypoints) of the fragment point clouds for registration.
In this paper, a novel approach for both keypoint learning and keypoint description based on neural networks is presented. The pipeline is demonstrated for both artificial objects with synthetically generated fractures ('cube_6') as well as input data originating from 3D scans of real fractured objects ('brick'). However, descriptiveness of the resulting features was not sufficient to allow for successful object reassembly. Future work could therefore focus on joint learning of feature detection and description or the use of alternative feature descriptors.

1. Introduction

In this project, we investigate the problem of 3-D fragment reassembly - given a set of point clouds, we want to assemble these fragments to reconstruct the original object. Although there are a few works on fragment reassembly, a majority of them use hand-crafted approaches to accomplish the task. Furthermore, there are a few works on point cloud registration that use learned features, but this task is complementary to the goal of fragment reassembly. Consequently, there is not a lot of data available for fragment reassembly, using learned features and interest points. We propose a pipeline to achieve the above-mentioned task. In section 2, we discuss the related works in this domain. In section 3 we discuss the pipeline consisting: data-set generation using Voronoi diagrams and fractals, key-point detection using an augmented version of the unsupervised stable interest point detection (USIP), keypoint description using features extracted from PointNet++, and augmented using a siamese network, and finally, the 3D-reassembly of fragments using pairwise and triple-wise matching. In section 4, we discuss the experiments conducted to improve the quality of the assembly obtained. Lastly, in section 5 we discuss the conclusions obtained from each section of the pipeline.

2. Related work

Point cloud registration: Point cloud registration is a related task to fragment reassembly, since the underlying goal of aligning 3D point clouds is the same. Although fracture reassembly has some special characteristics, many methods from point cloud registration can, with some adaptions, be also applied to fragment reassembly. While there are methods using hand-crafted descriptors [23], most research in this field has switched to machine learning techniques. [10] and [3] extract interest points and descriptors with a neural network, which are used to align the fragments in a subsequent step. The recent work by Bai *et al.* [3] proposes a method to learn interest points and descriptors jointly to improve descriptiveness. Another approach is to calculate the rotation and translation directly, skipping the need for explicit descriptors. [5] and [24] propose neural networks that directly put out the desired rotation and translation for the alignment. [1] and [19] use PointNet [16] in their pipeline and also calculate rotation and translation directly.

2D image reassembly: [18] uses geometric as well as colour information to find corresponding pairs. [26] additionally employ a graph-based method to improve global alignment after pairwise matching. Also in image reassembly, more recent works focus mainly on using neural networks to solve the reassembly task. Both [9] and [15] use a convolutional neural network (CNN) to extract information from the images. They also use a global optimization step for refinement. The paper [14] uses a siamese neural network to assemble originally 3 dimensional objects by using their 2D RGB images.

3D fracture reassembly: Most of the current work, that tries to solve the problem of reassembling fractured objects use hand-crafted features ([27], [11], [28], [25], [8], [7]). Only [12] uses a convolutional neural network (CNN) in their approach. However the CNN is only used to segment the fracture surfaces, which are aligned using traditional methods. Most methods rely on a segmentation in fractured and intact surfaces. [27], [11] and [12] use fracture-region matching, applying the fracture surfaces itself to match the fragments. In contrast to that, [28] and [25] use only the intact surfaces to find correspondences. [27] additionally employs a template of the whole object if available to improve reassembly. The methods described in [11], [28], [25] and [7] do only a pairwise matching. [8] fuses already matched pairs together, to speed up matching for further fragments. [27] and [12] further refine the matching by optimizing over more than two parts after pairwise matching , which improve the reassembly effectiveness in small fragmented pieces.

3. Pipeline

3.1. Data-set generation

The goal is to create a quantitative data-set for training, which contains fractured pieces with appropriate dimensions and inner fracture regions. We use three-dimensional Voronoi diagram and fractals.

The Voronoi diagram divides the 3D scattered points into convex polyhedrons such that each polyhedron contains exactly one particle (the generating 3D point), and every point in a given Voronoi cell is much closer to the generating point than to any other generating points. The diagram results in a reliable segmentation of the object, which is insensitive to the scale of the object. Furthermore, the object can be fractured parametrically by adjusting the amount and location of the generating points, the same object can be fractured in a variety ways.

Fractals is primarily used in science to describe the great variety of natural structures that are irregular, rough, having irregularities of various sizes that bear a special 'scaling' relationship to one another [13]. We use fractals surface to form the rough and irregular fracture surface.

The Voronoi diagram and fractals are implemented in Blender. To operate the algorithms robustly, here are a base set of rules for the input mesh: the mesh is defined by polygon without duplicated vertices; each vertex in the polygon is stored counter-clockwise; the polygon is convex; approximately equally sized polygon segments; the mesh is manifold and water-tight. After the mesh is pre-processed, it can be fractured and output in .obj format. It's worth noting that the mesh created by one Voronoi cell can contain more than one loose part if the generating point is located outside the input mesh. In this condition, the loose parts need to be exported separately. After, the vertex coordinates and normals can be extracted. Technical explanation of the process above can be found in the attached files.

In the data-set, we use 2 kinds of input mesh. The first kind are meshes which are geometrically complex - containing curvatures changes, sharp crease features (Fig. 1). The second kind are primitives from Blender that contains a smooth intact surface, e.g. cube, cylinder (Fig. 2).

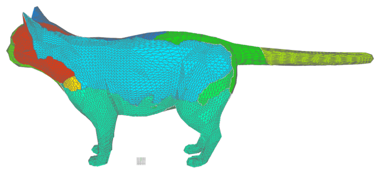
Figure 1. Fracture pieces of *cat*

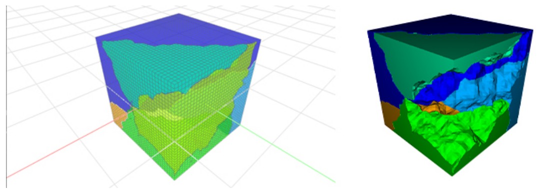
Figure 2. Intact and fracture surface *cube 6*

3.2. Keypoint detection

The keypoint detector is based on the unsupervised stable interest point detection (USIP) proposed by Li and Lee [10]. USIP takes a point cloud with surface normals $\mathbf{X} \in \mathbb{R}^{6\times N}$ as input, where N is the number of points. The outputs are the locations $\mathbf{Q} \in \mathbb{R}^{3\times M}$ and a positive saliency score $\sigma \in \mathbb{R}^{M+}$ for M keypoints. In the following we will briefly review the working principle of USIP.

To train the network, two point clouds $\mathbf{X}$ and $\tilde{\mathbf{X}}$ are fed into a feature Proposal Network (FPN), where $\tilde{\mathbf{X}}$ is a randomly rotated and translated version of $\mathbf{X}$. The FPN gives the location of the keypoints ($\mathbf{Q}$ and $\tilde{\mathbf{Q}}$) as well as the saliency scores (σ and $\tilde{\sigma}$). Now the random rotation and translation on $\tilde{\mathbf{Q}}$ is undone, which results in $\mathbf{Q}'$. It is assumed, that this operation has no influence on the saliency, hence $\sigma' = \tilde{\sigma}$. The goal of the FPN is to produce highly reproducible keypoints (ideally $\mathbf{Q} = \mathbf{Q}'$). This means, that the location should be indifferent to rotations and translations. Additionally the keypoints should be near the point cloud (but not necessarily corresponding to any point of the point cloud). Li and Lee achieve this by proposing a loss function consisting of two parts. The first part is the probabilistic chamfer loss $\mathcal{L}_c$. It ensures repeatability of the keypoints:

$$\mathcal{L}_c = \sum_{i=1}^{M} \left(\ln \sigma_{ij} + \frac{d_{ij}}{\sigma_{ij}} \right) + \sum_{j=1}^{M} \left(\ln \sigma_{ji} + \frac{d_{ji}}{\sigma_{ji}} \right) \tag{1}$$

with

$$\sigma_{ij} = \frac{\sigma_i + \sigma'_j}{2} > 0 \tag{2}$$

and

$$d_{ij} = \min_{Q'_j = \mathbf{Q}'} \|Q_i - Q'_j\|_2 \geq 0 \tag{3}$$

Note that $d_{ij} \neq d_{ji}$ and $\sigma_{ij} \neq \sigma_{ji}$, since the set of nearest neighbours is different. The second part is the point-to-point loss $\mathcal{L}_{point}$. It keeps the keypoints close to the point cloud:

$$\mathcal{L}_{point} = \sum_{i=1}^{M} \min_{X_j=\mathbf{X}} \|Q_i - X_j\|_2^2 + \sum_{i=1}^{M} \min_{\tilde{X}_j=\tilde{\mathbf{X}}} \|\tilde{Q}_i - \tilde{X}_j\|_2^2 \tag{4}$$

Finally, the loss function is defined as:

$$\mathcal{L} = \mathcal{L}_c + \lambda \mathcal{L}_{point} \tag{5}$$

Where λ is a tuning parameter to tune the weighting of the two losses. For further details, the interested reader is referred to the original paper [10].

Since getting repeatable keypoints on opposing sides of a fracture is a fundamentally different task than getting them for a single shape (a convex shape has tho match to a concave shape and vice versa), some changes had to be made on USIP. To this end we use the pretrained version of USIP, that was trained on modelnet [22] and retrain it with our generated data. Instead of feeding the same point cloud twice to the FPN, we use two different fragments, that share a fracture surface. Three approaches where tested to generate these training pairs: (1) All possible combinations of two different fragments sharing a surface (1v1). (2) One single fragment as first part. The second part is a point-cloud consisting of all the parts, sharing a surface to part 1 (1vN). (3) One single fragment as first part. The same fragment as second part, but with “flipped” surface normals (by changing the sign), which leads to a ”negative” with the same shape as part 1 (FN). Two fragments where considered neighbours, if they shared at least 100 points (distance below a threshold of 10^{-3}). The assembled parts were normalized roughly to [-1, 1]. Additionally random Gaussian noise was added to the position of the points (mean: 0, standard deviation: 10^{-3}).

For the 1vN approach the probabilistic chamfer loss (1) was modified as follows:

$$\mathcal{L}_c = 2 \sum_{i=1}^{M} \left(\ln \sigma_{ij} + \frac{d_{ij}}{\sigma_{ij}} \right) \tag{6}$$

This means that only the nearest neighbours to the points in part 1 are considered in the loss. Through this, ”outliers” in part 2 (which consists of several fragments) don’t influence the training. This is beneficial, because these ”outliers” may fit to other fragments than the one in part 1 and should therefore not be penalized. Furthermore the weighting parameter λ from equation (5) was set to 5 for all approaches. The other settings were kept the same as in the original code, except for some minor adjustments. For further information, see the attached code, which is commented at every modification.

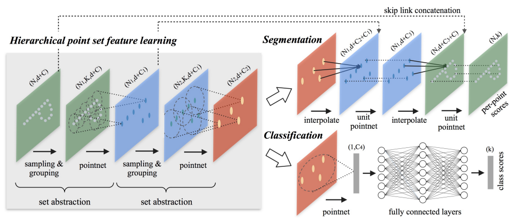

Figure 3. PointNet++ Architecture

3.3. Keypoint description

As for the part of KeyPoint description we use the network architecture of PointNet++ [17] to extract the featured representations of the pointcloud. We use the semantic segmentation implementation of the PointNet++ architecture, and extract the features before they are fed into the final layer of PointNet++. PointNet++ takes a point cloud with surface normals $\mathbf{X} \in \mathbb{R}^{6\times N}$ as input, where N is the number of points. Each point cloud is propagated through layers of Point Net Set Abstraction modules and Point Net Feature Propagation modules, as show in Fig 3.3. Once the features reach the penultimate layer we save these features. Each point cloud leads to features of size $\mathbf{P} \in \mathbb{R}^{128\times N}$

Technically, we do so by implementing a forward hook in PyTorch, which yields us the features extracted from the model of PointNet++ which had been pretrained on Stanford S3DIS dataset [2]. One of the challenges we face is that the key points extracted from the previous step do not correspond to the coordinate locations of points from a the point cloud. So to compute the feature descriptor for a key point we approximate the feature descriptor for a particular key point by taking an inverse distance weighted average of the points in the neighbourhood of that key point. We consider the 20 nearest neighbors (points in the point cloud) of the key point and compute an average of the features obtained for each of these neighbors. The average is a weighted average of the features of the neighbors, and the weights being inversely proportional to the distance of the neighbor from the keypoint. So since we obtain 128 keypoints from the USIP for each pointcloud the output dimension of the extracted features is $\mathbf{F} \in \mathbb{R}^{128\times 128}$.

The averaging is done as

$$f^{(j)}(x) = \frac{\sum_{i=1}^{k} w_i(x) f^{(j)}(i)}{\sum_{i=1}^{k} w_i(x)} \tag{7}$$

where

$$w_i(x) = \frac{1}{d(x, x_i)^p} \tag{8}$$

$j = 1, 2, ..., 128$ for each dimension of the feature and k is the number of neighbors considered for averaging, and $f^{(j)}(i)$ is the j^{th} feature of the i^{th} neighbor. The dis-

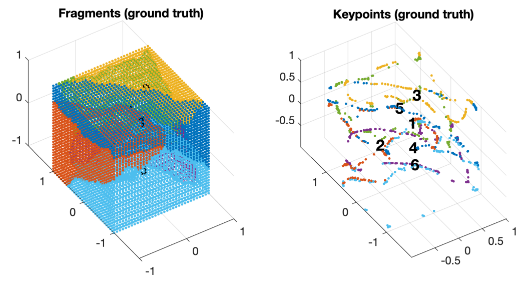

Figure 4. Point cloud and keypoints of data set *cube 6*

tance between the key point and the neighbor is denoted as $d(x, x_i)^p$ and the distance is calculated with $p = 2$.

The descriptors obtained by this method for the key points are substantially descriptive as they are extracted from PointNet++, but the descriptors have not been fine-tuned for our task. To accomplish this we further train a Siamese network using the descriptors obtained from the previous step, and optimise the network using a triplet loss. So to train this network we first need to generate triplet pairs. This is done by using the distance in the spatial domain between each key point. We discuss more on the different methods used to define the triplet pairs in Experiments. We train the siamese network composed of 3 fully connected layers to yield an embedding that minimizes the distance between the anchor and the positive pair, and maximize the distance between the anchor and the negative pair. The network is trained by minimizing the triplet margin loss.

$$L(a, p, n) = max\left\{d(a_i, p_i) - d(a_i, n_i) + margin, 0\right\} \tag{9}$$

where

$$d(x_i, y_i) = \|x_i - y_i\|_p$$

a, p, n are the anchor, positive and negative samples respectively. We use a margin of 0.5, and $p = 2$ when computing the distance.

This embedding should help the 3D assembly operation, since it finds a more descriptive representation in the feature space. And, the positive pairs of key-point correspondences are brought together in the embedding while the negative pairs are separated, thus facilitating the reassembly.

3.4. 3D reassembly optimization

Based on input of the random pose fragment point clouds, keypoints and feature descriptors, the reassembly optimization determines the optimal transformations for each fragment for the reassembly of the object.

Our approach builds upon the work by Cui who implemented a reassembly optimization for 2D photo puzzles [4]. The following steps are performed sequentially.

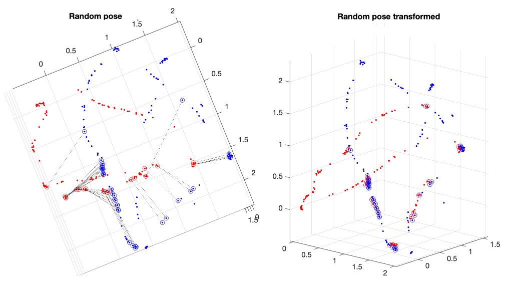

Figure 5. Pairwise matching of two fragments. Corresponding keypoint pairs connected by dashed line. Left: random pose, right: after matching and transformation

3.4.1 Pairwise matching

After importing the input data (Fig. 4), the next step is to find optimal rigid transformations (R, T) for all pairwise combinations of fragments A and B such that the corresponding keypoints of A and B are optimally aligned (minimization of least square of distance) (Fig. 5). The corresponding keypoint pairs are identified with nearest neighbour search in the 32 dimensional feature space, limited to a maximum number of pairs (default 20) and maximum features space euclidean distance (default 0.8). However, at the time of writing, this did not result in usable keypoint pairs as there was insufficient correlation between euclidean distance in feature space (32D) and ground truth xyz (3D) space. Therefore, for demonstration purposes of the reassembly optimization, the corresponding keypoint pairs are identified based on nearest neighbour search in the ground truth space.

The estimation of optimal transformations between two fragments is initially performed by Consensus Maximization with Linear Matrix Inequality Constraints as described by Speciale et al. [21]. Based on the reported inlier set of keypoints, an optimal rigid body transformation (R, T) is estimated [20].

3.4.2 Triple-wise matching

In order to discard false matches resulting from pairwise matching, triple-wise matching is performed for all triple combinations of fragments. A positive triple-wise match occurs for fragments A, B and C if the following conditions for the transformations (R, T) are met:

$$R_{A\to B} + R_{B\to C} + R_{A\to C} \leq \alpha_{max} \tag{10}$$

$$T_{A\to B} + T_{B\to C} + T_{A\to C} \leq d_{max} \tag{11}$$

3.4.3 Graph based reassembly

Based on the result of the triple-wise matching, a graph is established with the nodes denoting the fragments and

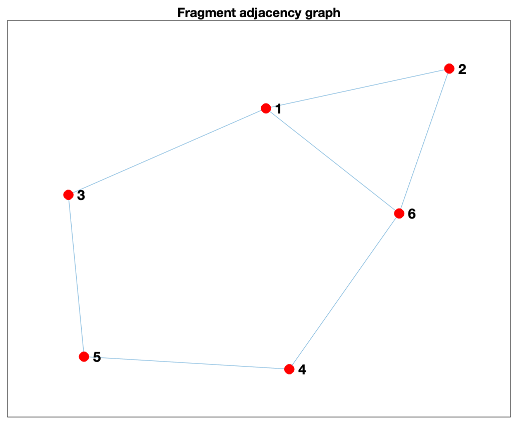

Figure 6. Graph of data set *cube 6*. Note: Missing triple-wise match for fragments 3, 4 and 5

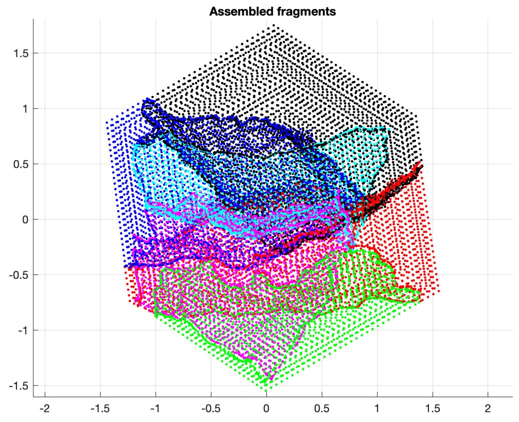

Figure 7. Assembled fragments of data set *cube 6*

the edges the adjacency between fragments (Fig. 6). In case not all fragments are covered by triple-matching, the graph can be augmented by adding edges based on information from the pairwise matching. However, this should be avoided as pairwise matching likely results in both true and false matches. Starting from a manually designated fragment, the graph is traversed, transformations summarized and all fragments reassembled by applying the transformations (Fig. 7).

4. Experiments

4.1. Keypoint detector

To assess the performance of the feature detector, the relative repeatability was calculated. This is the percentage of keypoints on a fragment, that have at least one keypoint on another fragment in a ball environment with radius r when the fragments are aligned. We compare the three training methods introduced in 3.2 to the unchanged USIP trained only on modelnet. Table 1 shows a comparison on our generated datasets cube_6 and cube_20_seed_1. Both were not used for training. All three methods outperform the unchanged USIP, whereas the model trained on the 1vN data shows the best performance. Table 1 also shows a comparison on real 3D scan data (brick and head from [6], courtesy by Vienna University of Technology). Since the brick is bigger than the other datasets, it was downscaled by a factor of 10 during the keypoint detection, to normalize it to roughly [-1,1]. It can be seen, that the method generalizes well to real world data, especially the brick. The head performed worse. This is probably due to the fact, that it is hollow and therefore has rather small fracture surfaces. Since the training set was rather small (468 trainingpairs) and only consisted of two different primitives (cube and cylinder), this values could certainly be improved by using more training data.

dataset (# fragments)		MN	1v1	1vN	FN
cube (6)	min	0.0%	10.9%	33.6%	21.9%
	max	0.8%	32.8%	39.8%	36.0%
	median	0.0%	18.0%	**37.1%**	31.3%
cube (23)	min	0.0%	14.8%	28.9%	23.4%
	max	17.3%	57.8%	98.3%	85.1%
	median	1.9%	41.4%	**51.7%**	47.1%
brick (6)	min	11.8%	21.1%	20.3%	28.0%
	max	53.3%	60.2%	65.6%	62.0%
	median	28.1%	46.9%	45.7%	**51.7%**
head (12)	min	0.0%	0.8%	3.1%	0.8%
	max	36.0%	12.5%	32.8%	42.2%
	median	0.0%	4.3%	**16.0%**	13.7%

Table 1. relative repeatability of some sample parts. For the cube and head dataset the radius is r=0.05. For the brick it is r=0.3, since the brick is roughly 6 times bigger. The number in brackets denotes the number of fragments

4.2. Keypoint Descriptor

The experiments we conduct for this part are pertinent to defining of positive and negative correspondences and training the neural network on the triplet pairs computed using these different definitions.

Firstly, we need to compute the distance, in spatial domain, between keypoints of fragment i and the key-points of all the other fragments. This gives a distance matrix of $\mathbf{D} \in \mathbb{R}^{128\times128*(nfrag-1)}$, where $nfrag$ is the number of fragments for the object.

- Positive pairs for Fragment i: The positive pair for a keypoint j in fragment i is the keypoint which is closest to it out of the other $128 \times nfrag$ keypoints.

- The negative pair for a keypoint j in fragment i is defined as the key point that lies is closest to keypoint

j but outside a ball of radius (positive pair distance + margin)

The rationale behind this approach being that it reduces distance (in embedding space) between keypoints of fragments that lie closest to each other, and increases the distance (in embedding space) between points that lie in the neighborhood of the closest match, which can otherwise make the job of the reassembly algorithm much tougher. The assumption here is the features obtained by PointNet++ already provide enough separation in the feature space for points that lie far apart in the spatial domain, and we use the siamese network to increase the distance in the embedding space for points that lie in the neighborhood of the closest positive match.

We alternatively also try different definitions for finding negative pairs

- The negative pair for a keypoint j in fragment i is defined as the key point that lies is closest to keypoint j but outside a ball of radius (margin)

In our testing of reassembly we find the first approach to work much better than the alternative and we proceed with that for further testing

The negative pairs for a keypoint j in fragment i is defined as the key point all the key-points that lie outside a ball of radius (margin)

4.3. 3D assembly optimization

The functionality of the 3D assembly optimization was demonstrated with the datasets 'cube_6' and 'brick'. During the application, two weaknesses of our approach were identified:

Firstly, triple-wise matching not always results in matches even if three fragments are adjacent. The main reason for that is bad conditioning of the pairwise matching optimization problem. Either because keypoint pair are few (e.g. 2 or 3) or aligned along one axis which results in too many degrees of freedom. Fig. 8 shows and example of two fragments that share a fracture surface but have only 2 keypoint pairs that are aligned on one edge of the fracture surface. This leads to insufficient pairwise matches and subsequently to insufficient triple-wise matches.

Secondly, the identification of keypoint pairs by nearest neighbour search in the n-dimensional feature space did at the time of writing not result in meaningful correspondences. Fig. 9 shows the correlation of distances between one pair of keypoints in feature space and xyz space (ground truth). A significant correlation is not evident. Therefore, for the purpose of demonstrating the 3D assembly, keypoint correspondences were determined based on nearest neighbour search in ground truth xyz space.

As an example, the reassembled fragments of the *brick* 3D scanned data set are shown in Fig. 10.

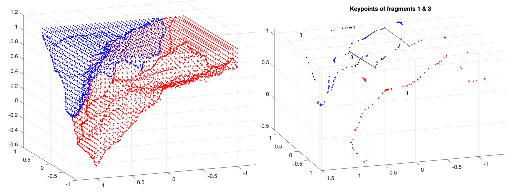

Figure 8. Pairwise matching resulting in insufficient keypoint pairs

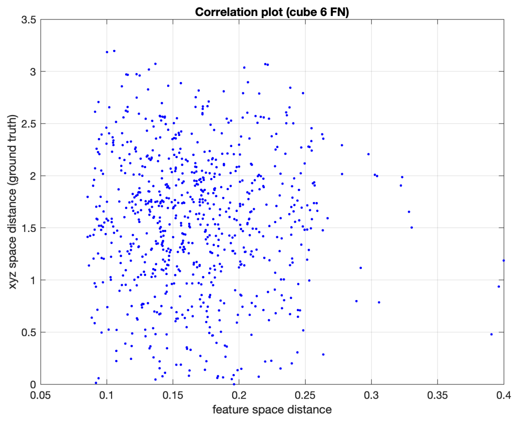

Figure 9. Distance correlation between keypoint pairs in feature space and ground truth xyz space

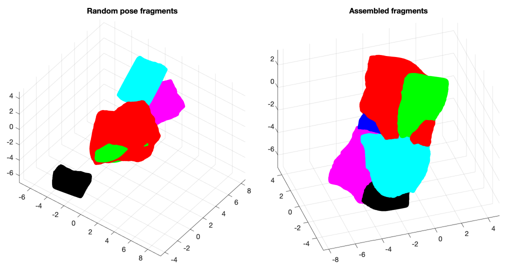

Figure 10. Assembled fragments of data set *brick*

5. Conclusions

The goal of this project was to demonstrate the feasibility of 3D feature point learning for fractured object reassembly. The conclusions are as follows:

1. Data-sets of fractured objects were successfully generated in Blender with use of Voronoi diagrams and fractals and postprocessing in Python. The method produced realistically looking fracture surfaces and proved invariant to the complexity of the input geometry.

2. Keypoint detection with a retrained USIP [10] network was demonstrated successfully. Three novel and dedicated training strategies were implemented resulting in different keypoint locations. However, features are

predominantly located on or near the fragment edges which leads to the following issue: Two adjacent fragments always share one or more contact surface. However, they may not share the same edge along the fracture surface. Therefore having features located also in the fracture surface would be beneficial for the pairwise matching during the 3D reassembly optimization.

3. For keypoint description a method based on PointNet++ [17] was implemented. The original implementation of the PointNet++ architecture was successfully augmented with a Siamese network using a triplet margin loss function to generate an embedding for the keypoints. However, the distances in the embedding space obtained from the siamese network don't yield enough correspondences in for the reasssembly. We speculate the reason for this being the way the negative pairs are selected. Since we only generate 1 negative pair per keypoint, we only train the network to optimise the distances between this negative pair and not the other possible negative pairs for the same keypoint. We could generate the negative pairs for a keypoint j in fragment i by taking all the key-points that lie outside a ball of radius (margin). This will also help overcome the previous assumption that the features obtained by PointNet++ already provide enough separation in the feature space for points that lie far apart in the spatial domain. But this would come at the cost of exponential blowup in the size of the dataset generation.

4. 3D reassembly optimization based on pairwise matching and subsequent triple-wise matching was implemented and demonstrated successfully with the 'cube_6_FN' and 'brick_FN' data-sets. However, limitations apply as the the pairwise matching is only successful with keypoint correspondences retrieved from xyz space (ground truth). Using the feature descriptors for retrieval of keypoint correspondences did not result in meaningful keypoint correspondences.

5. Further improvements could include the use of joint feature detection and description as described in [3] as well as the evaluation of alternative feature description networks.

6. Work split

Individual contributions of each team member:

- Chaoyu Du: Data-set generation and processing
- Ulrich Steger: Implementation of the keypoint detector based on USIP [10]
- Eshaan Mudgal: Implementation of keypoint description usign PointNet++ and siamese network trained with triplet margin loss.
- Florian Hürlimann: Implementation of 3D reassembly optimization in MATLAB based on existing code for 2D reassembly optimization [4].

7. Acknowledgements

The team would like to thank the supervisors Danda Paudel and Martin Oswald for their support. Further thanks go to Pablo Speciale and Luxiao Cui for providing the source code for the 2D reassembly optimization which was of great help for this project.

References

[1] Y. Aoki, H. Goforth, R. A. Srivatsan, and S. Lucey. Pointnetlk: Robust & efficient point cloud registration using pointnet. volume 2019-June, pages 7156–7165. IEEE Computer Society, 6 2019.

[2] I. Armeni, O. Sener, A. R. Zamir, H. Jiang, I. Brilakis, M. Fischer, and S. Savarese. 3d semantic parsing of large-scale indoor spaces. In *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*, 2016.

[3] X. Bai, Z. Luo, L. Zhou, H. Fu, L. Quan, and C.-L. Tai. D3feat: Joint learning of dense detection and description of 3d local features. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 6358–6366, 3 2020.

[4] L. Cui. 2d fragment reassembly. Master's thesis, ETH Zürich, 2019.

[5] Z. Gojcic, C. Zhou, J. D. Wegner, L. J. Guibas, and T. Birdal. Learning multiview 3d point cloud registration. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1756–1766, 1 2020.

[6] Q. X. Huang, S. Flöry, N. Gelfand, M. Hofer, and H. Pottmann. Reassembling fractured objects by geometric matching. volume 25, pages 569–578. ACM PUB27 New York, NY, USA, 7 2006.

[7] S. hui Liao, C. Xiong, S. Liu, Y. qi Zhang, and C. lin Peng. 3d object reassembly using region-pair-relation and balanced cluster tree. *Computer Methods and Programs in Biomedicine*, 197:105756, 12 2020.

[8] C. Jia, L. He, X. Yang, X. Han, B. Chang, and X. Han. Developing a reassembling algorithm for broken objects. *IEEE Access*, 8:220320–220334, 2020.

[9] C. Le and X. Li. Jigsawnet: Shredded image reassembly using convolutional neural network and loop-based composition. *IEEE Transactions on Image Processing*, 28:4000–4015, 8 2019.

[10] J. Li and G. H. Lee. Usip: Unsupervised stable interest point detection from 3d point clouds. *Proceedings of the IEEE International Conference on Computer Vision*, 2019-October:361–370, 3 2019.

[11] Q. Li, G. Geng, and M. Zhou. Pairwise matching for 3d fragment reassembly based on boundary curves and concave-convex patches. *IEEE Access*, 8:6153–6161, 2020.

[12] B. Liu, M. Wang, X. Niu, S. Wang, S. Zhang, and J. Zhang. A fragment fracture surface segmentation method based on learning of local geometric features on margins used for automatic utensil reassembly. *CAD Computer Aided Design*, 132, 3 2021.

[13] B. Mandelbrot, D. Passoja, and A. Paullay. Fractal character of fracture surfaces of metal. *Nature*, 308:721–722, 04 1984.

[14] C. Ostertag and M. Beurton-Aimar. Matching ostraca fragments using a siamese neural network. *Pattern Recognition Letters*, 131:336–340, 3 2020.

[15] M. M. Paumard, D. Picard, and H. Tabia. Image reassembly combining deep learning and shortest path problem. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 11210 LNCS:155–169, 9 2018.

[16] C. R. Qi, H. Su, K. Mo, and L. J. Guibas. Pointnet: Deep learning on point sets for 3d classification and segmentation. volume 2017-January, pages 77–85. Institute of Electrical and Electronics Engineers Inc., 11 2017.

[17] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. *CoRR*, abs/1706.02413, 2017.

[18] R. Saharan and C. Singh. Reassembly of 2d fragments in image reconstruction. *International Journal of Computer Applications*, 19:41–45, 30 2011.

[19] V. Sarode, X. Li, H. Goforth, Y. Aoki, R. A. Srivatsan, S. Lucey, and H. Choset. Pcrnet: Point cloud registration network using pointnet encoding. *Computer Vision and Pattern Recognition*, 8 2019.

[20] U. Shinji. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, Vol. 13, No. 4, 4 1991.

[21] P. Speciale, D. P. Paudel, M. R. Oswald, T. Kroeger, L. V. Gool, and M. Pollefeys. Consensus maximization with linear matrix inequality constraints. volume 2017-January, pages 5048–5056. Institute of Electrical and Electronics Engineers Inc., 11 2017.

[22] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3d shapenets: A deep representation for volumetric shapes. volume 07-12-June-2015, pages 1912–1920. IEEE Computer Society, 10 2015.

[23] J. Yang, Z. Cao, and Q. Zhang. A fast and robust local descriptor for 3d point cloud registration. *Information Sciences*, 346-347:163–179, 6 2016.

[24] Z. J. Yew and G. H. Lee. Rpm-net: Robust point matching using learned features. pages 11821–11830. IEEE Computer Society, 2020.

[25] C. Yin, M. Zhou, Y. Fan, and W. Shui. Template-guided 3d fragment reassembly using gds. volume 875, pages 432–441. Springer Verlag, 4 2018.

[26] K. Zhang and X. Li. A graph-based optimization algorithm for fragmented image reassembly. *Graphical Models*, 76:484–495, 9 2014.

[27] K. Zhang, W. Yu, M. Manhein, W. Waggenspack, and X. Li. 3d fragment reassembly using integrated template guidance and fracture-region matching. IEEE, 12 2015.

[28] Y. Zhang, K. Li, X. Chen, S. Zhang, and G. Geng. A multi feature fusion method for reassembly of 3d cultural heritage artifacts. *Journal of Cultural Heritage*, 33:191–200, 9 2018.