Symmetry-Based 3D Object Completion using Local Geometry Features

Daria Omelkina[1], Pavlo Hilei[1], Oles Dobosevych[1], Rostyslav Hryniv[1], Taras Rumezhak[1,2], Vladyslav Selotkin[2] and Volodymyr Karpiv[2]

[1]Ukrainian Catholic University ML Lab, Ukrainian Catholic University, Lviv, Ukraine
[2]SoftServe, Ukraine

Abstract

3D scanning of real-world objects has become an integral step of many manufacturing processes and digital image processing applications. Despite the impressive progress in modern 3D scanning technology, the produced 3D models often suffer from defects due to occlusions, poor light conditions, special surface reflection properties, scanner displacements, imperfect algorithms, etc. Point cloud completion is a standard post-processing step aiming at removing or smoothing such irregularities.

In this article, we propose a novel method for damage completion of 3D objects possessing reflection symmetry. Such symmetries are identified by matching local shape features such as curvatures and edges in mesh and point cloud representations of the objects. We describe the pipeline of our method, justify its steps, and evaluate its performance on the ModelNet40, a Princeton 3D object dataset. The results demonstrate comparable improvement in the damaged 3D object completion to popular neural network-based and symmetry-based approaches.

Keywords

completion, point cloud, mesh, curvatures, reflection symmetry

1. Introduction

Advances in modern 3D scanning technology have led to a significant boost in computer vision post-processing tools for 3D models. One of the standard tasks of 3D scanning is completing the obtained 3D models that suffer from poor light conditions, occlusions, surface reflection, etc. This completion task has been approached in many different ways, in particular, using neural networks pre-trained on certain object classes. Although such methods show good results for new objects from these classes, their performance on previously unseen object types drops significantly. Therefore, alternative algorithms capable of completing previously unseen objects are of vital importance.

In this paper, we suggest a novel approach to 3D model completion based on the geometric features of mirror-symmetric objects. Since this type of symmetry is typical in man-made environments, such an assumption is not very restrictive. Finding plane symmetry in approximately symmetric 3D models is the most important step; we then use it to complete any damages and holes which could occur during scanning or were already present in the real-life scanned object (e.g., in archaeological findings that often suffer from significant time damage).

The main idea is that in mirror-symmetric objects, all local shape features are also symmetrically distributed. For instance, the principal curvatures of symmetric points are the same, and the principal directions enjoy the same symmetry; likewise, all linear edges in symmetric objects can be split into symmetric pairs. The search for global mirror symmetries can be based on pairs of points with similar curvatures and/or edges. Each such pair creates a potential symmetry plane, and the majority vote determines the best mirror symmetry. The latter is then used to compare and combine the object and its mirror image and thus complete the missing or damaged parts.

That approach demonstrated competitive results in comparison to a known traditional symmetry-based method of point cloud completion and three deep learning method. More details on the methods for comparison are given in the following sections.

2. Related work

As mentioned above, damages in 3D objects can be repaired by using missing information from their symmetrically transformed images. There are two main directions for symmetry plane detection in 2D/3D data: classical and deep learning approaches. In this paper, we aim to extend the classical approach as more robust for previously unseen objects and thus review the corresponding algorithms first.

One of the approaches for symmetry-based point cloud

26th Computer Vision Winter Workshop, Robert Sablatnig and Florian Kleber (eds.), Krems, Lower Austria, Austria, Feb. 15-17, 2023
omelkina@ucu.edu.ua (D. Omelkina); p.hilei@ucu.edu.ua (P. Hilei); dobosevych@ucu.edu.ua (O. Dobosevych); rhryniv@ucu.edu.ua (R. Hryniv); rumezhak@ucu.edu.ua (T. Rumezhak); vselo@softserveinc.com (V. Selotkin); vkarpi@softserveinc.com (V. Karpiv)

completion is discussed in [1], where the authors propose a three-step algorithm consisting of symmetry plane estimation via Principal Component Analysis (PCA), iterative improvement of the symmetry plane via Iterative Closest Point (ICP), and, finally, hole detection and filling. The paper [1] has served as motivation for the current work, in which we tried to make symmetry plane search more geometry-aware and preserve the robustness and light weight of the algorithm (cf. Section 4.2 for performance comparison).

The principal idea of our geometry-aware symmetry detection is based on the paper [2]. In this paper, Mitra et al. used the so-called signatures of the geometric shapes represented by principal curvatures, principal directions, and normal vectors to detect local symmetries in the objects. We borrow the idea of matching local geometries from [2] but also add edge-based symmetry detection to the pipeline to include man-made objects of predominantly planar shape. The edge detection method in point clouds suggested by Ahmed et al. in [3] uses very simple and clear geometric arguments, and we have implemented it in our algorithm.

The paper [4] by Mitra et al. is a detailed report on different symmetry detection and application methods for 3D objects. They include symmetry detection methods for meshes and/or point clouds: graph-based approaches, RANSAC-based verification (e.g., sliding dockers approach in [5]), multidimensional scaling (e.g., intrinsic structure detection in [6]), voting for a symmetry transform (e.g., partial intrinsic reflection symmetries extraction in [7]), etc.

Recently, deep learning algorithms have become state-of-the-art solutions in different areas, especially for computer vision tasks. Various neural networks were proposed for symmetry detection, data completion, segmentation, classification, and other tasks for processing 3D data. Examples of successful solutions include but are not limited to [8], [9], [10], [11] etc.

The authors of MSN (Morphing and Sampling Network) thoroughly describe their model in [12]. Their approach to point cloud completion consists of two stages. Firstly, they use an auto-encoder to predict the completed object by morphing the unit squares into a collection of surface elements. Secondly, they merge that output with the original damaged object and feed a subset point cloud to a residual network. After that second stage, they obtain the final completed point cloud. The approach in [13] is also based on residual networks. They use 3D grids as an intermediate representation of point clouds and introduce two novel gridding layers. This approach allows using 3D convolutions on the unordered and irregular point cloud data. The authors of [14] introduce PoinTr – a transformer encoder-decoder model. They represented point clouds as unordered groups of points with position embeddings and added a geometry-aware block for modelling local geometries to the transformer.

We compare the performance of our method to the performance of one traditional [1] and three deep learning [12], [13], [14] methods.

3. The completion algorithm

The pipeline of the proposed method consists of the following steps (discussed below in more detail):

1. 3D model representation: triangle mesh, point cloud and principal curvatures.
2. Finding symmetry planes using the curvature-based algorithm.
3. Finding symmetry planes using the edge-based algorithm.
4. Mean Shift Clustering and determining the best symmetry plane.
5. Using the best symmetry plane for 3D model completion.

As a pre-processing step, we scale the meshes of 3D objects to boost the parameter tuning and to make the final results statistically interpretable.

3.1. Principal curvatures and principal directions of the 3D model

The input of our method is a (scaled) triangular 3D mesh of an object, which is used to calculate the principal curvatures and directions. **Principal curvatures** at a surface point characterize locally extremal surface bending in the respective **principal directions** in the tangent plane. Principal curvatures and directions describe well the approximate local surface shape. For example, principal curvatures of a small absolute value represent a relatively flat surface, while greater curvatures correspond to saddle-like or ellipsoid-like behavior. Abrupt changes of principal curvatures indicate a possible edge.

Under mirror symmetry, the pairs of symmetric points share the same curvatures, while their principal directions and normals are mirror symmetric. As suggested in [2], this can be used to detect the mirror symmetry: we cluster the points with equal (up to a threshold) principal curvatures, single out those pairs of points that have compatible principal directions, and then calculate potential plane symmetry for each compatible pair. If this plane symmetry passes the patch symmetry check (i.e., if the plane mirrors these points into one another along with some of their neighborhoods), we approve it to participate in the further majority voting for the most optimal plane.

We use the `principal_curvature` function from the **libigl** library [15] to calculate the principal curvatures

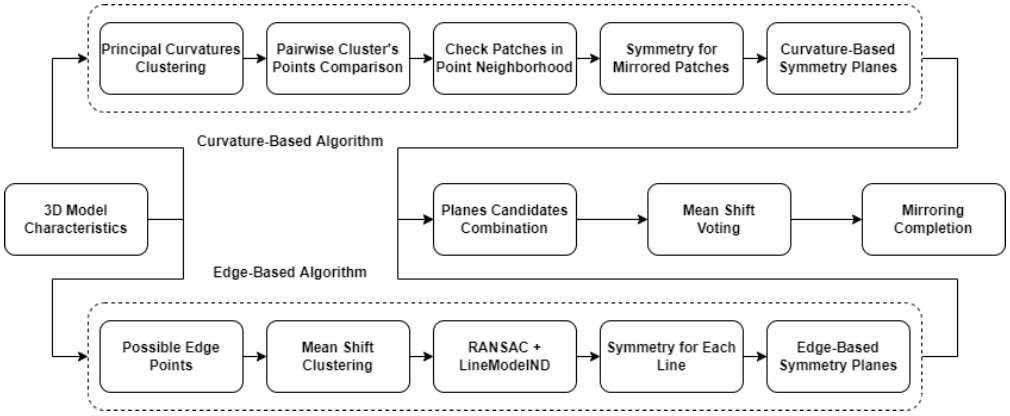

Figure 1: Overview of the main pipeline steps.

of the mesh. As suggested in [16], this function first fits locally a quadratic polynomial

$$f(x, y) = ax^2 + 2bxy + cy^2 + dx + ey \qquad (1)$$

to the surface and then computes the principal curvatures and directions as eigenvalues k_1, k_2 and eigenvectors $\mathbf{d}_1, \mathbf{d}_2$ of the corresponding *shape operator* (a 2×2 matrix) explicitly given in terms of the parameters a to e. Observe that signs of the principal curvatures depend on the surface orientation: if $\mathbf{d}_1$ and $\mathbf{d}_2$ are interchanged, then the curvatures k_1, k_2 become $-k_2, -k_1$. We, therefore, order the principal curvatures so that $k_1 \geq |k_2|$.

We would like to mention that the `principal_curvature` function does not allow small triangle clusters (i.e., sets of triangles connected in a mesh). Therefore, before computing curvatures, we extract information about triangle clusters in the mesh and delete the clusters of size less than 10.

3.2. Curvature-based algorithm

Curvature-based algorithm uses principal curvature information to match potentially symmetric points on the ***point cloud*** representing the 3D object. It proceeds in the following steps:

1. Data preparation
2. Curvature-based point clustering
3. Within-cluster symmetry plane detection and validation

The result of these steps is an array-like structure of approved planes that take part in the final best symmetry plane voting described in Section 3.4.

3.2.1. Data preparation

Given a 3D mesh with vertices $\mathbf{x}_i$, $i = 1, \ldots, n$, we form the point cloud and then create a k-d tree for fast neighborhood search in later stages. Also, for each vertex $\mathbf{x}_i$, we calculate the corresponding principal curvatures $k_1^{(i)}, k_2^{(i)}$ and save the results in a separate file as a dictionary with keys $\mathbf{x}_i$ and principal curvatures as values.

3.2.2. Curvature-based point clustering

Candidates for symmetry planes are collected by detecting symmetric patches in the point cloud. We start with two compatible points, determine their symmetry plane, and then verify if it matches some neighborhoods of these points. Since symmetric patches have symmetric local shapes, we run through point pairs with close principal curvatures. We cluster the vertices of the point cloud with similar curvatures using the Mean Shift algorithm. To speed up bandwidth estimation for the Mean Shift algorithm, we fix the quantile at 0.005 and use only point samples of size 500; we also enable the *bin_seeding* option for the Mean Shift algorithm to decrease its running time.

3.2.3. Within-cluster symmetry plane detection and validation

For each cluster, we iterate through all point pairs A and B and first check that their principal curvatures

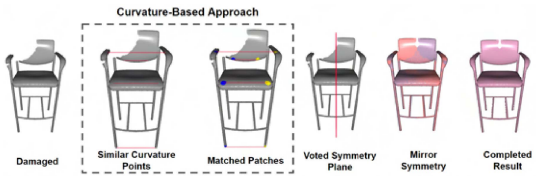

Figure 2: Curvature-based algorithm diagram. For demonstration purposes, only a couple of point pairs are shown

$(k_{1,A}, k_{2,A})$ and $(k_{1,B}, k_{2,B})$ are approximately equal, in the sense that

$$\left|\frac{k_{1,A}}{k_{1,B}} - 1\right| \le \varepsilon_1, \qquad \left|\frac{k_{2,A}}{k_{2,B}} - 1\right| \le \varepsilon_1, \tag{2}$$

with some predefined *curvature closeness threshold* ε_1; if $|k_{1,B}| \le \varepsilon_1$ or $|k_{2,B}| \le \varepsilon_1$, then we require instead that $|k_{1,A}| \le \varepsilon_1$ or $|k_{2,A}| \le \varepsilon_1$, respectively.

For each point pair A, B that has passed the above test, we draw the median perpendicular plane π and represent it via its Hough coordinates—the unit normal vector $\mathbf{n}$ and the (signed) distance d to the origin. The mirror reflection S_π is then given by

$$S_\pi(\mathbf{x}) = (I_3 - 2\mathbf{n}\mathbf{n}^\top)\mathbf{x} + 2d\mathbf{n}, \tag{3}$$

where I_3 is the 3×3 identity matrix. We fix the direction of $\mathbf{n}$ so that its first non-zero entry is positive; the signed distance d is then the scalar product of $\mathbf{d}$ and the vector $\overrightarrow{OC}$ from the origin O to the midpoint C of A and B.

In the next step, we perform the **patch symmetry validation** of S_π. We form patches P_A and P_B of the points A and B consisting of their closest $m = 200$ points of the point cloud. Then we mirror reflect P_A to get $P'_A = S_\pi(P_A)$ and check if adding it to P_B does not significantly change the geometry of the latter. We transform $P'_A \cup P_B$ into a mesh, recalculate the curvatures $(k'_1(\mathbf{x}), k'_2(\mathbf{x}))$ of each point $\mathbf{x} \in P_B$ in this larger mesh, and compare them with the initial curvatures $(k_1(\mathbf{x}), k_2(\mathbf{x}))$. If the mean squared error (MSE)

$$\mathrm{mse}(\pi) = \frac{1}{m} \sum_{\mathbf{x} \in P_B} \left(|k'_1(\mathbf{x}) - k_1(\mathbf{x})|^2 + |k'_2(\mathbf{x}) - k_2(\mathbf{x})|^2\right) \tag{4}$$

does not exceed some threshold K, then the potential symmetry plane π is validated, gets the *weight* $w(\pi) := 1/\mathrm{mse}(\pi)$, and is passed to the final majority vote.

To speed up the algorithm, we keep an array of planes that have already been rejected, and if a new plane π is very close to one in the array, we skip the patch symmetry step and update the array with π. To transform point clouds into meshes, we used the Ball Pivoting algorithm from the Open3D library [17].

3.3. Edge-based algorithm

Many man-made objects have edges or planar parts, for which curvatures are not well-defined or are uninformative. However, the edge lines in mirror-symmetric objects are also symmetric and can be used for symmetry detection. The **edge-based** algorithm exploits this and proceeds as follows:

1. Data preparation and parameters tuning
2. Edge point detection and clustering
3. Line fitting and expansion
4. Finding symmetry planes

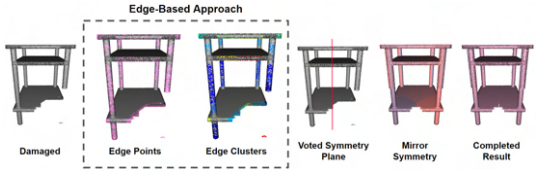

Figure 3: Edge-Based algorithm diagram.

3.3.1. Data preparation and parameter tuning

Here we work directly with the point cloud representation of the 3D object. The point cloud is obtained from the mesh using the Poisson Disc Sampling algorithm implemented in the Open3D library [17].

The edge-based algorithm requires several parameters; most remain default (see Section 4.3). However, the following need to be tuned for each model:

- *quantile* in the edge point clustering affects the number of clusters and thus the number of fitted lines;
- *line closeness threshold* ε_2 sets tolerance within which two lines are considered parallel or intersecting and thus influences mirror symmetry precision;
- *line expansion* option leads to improvement in some models;
- *edge detection threshold parameter* λ can be left at the default value for most 3D objects, but, in some cases, a smaller value (allowing more edge candidates) leads to better performance.

3.3.2. Edge point detection and clustering

In the first step, we iterate over points in the point cloud and apply the detection algorithm of [3] to find potential edge points. The algorithm is based on the observation that if the surface is smooth around its point $\mathbf{p}$, i.e., can be represented in local coordinates via (1), then the difference between $\mathbf{p}$ and the centroid of k nearest neighbors (kNN) of $\mathbf{p}$ is of the second order of the xy span of these neighbors.

Following this observation, we fix $k = 200$, denote by $\mathcal{N}(\mathbf{p})$ the kNN of $\mathbf{p}$, by $\mathbf{c}(\mathbf{p})$ the centroid of $\mathcal{N}(\mathbf{p})$, introduce the spacing parameter

$$r := \min_{\mathbf{q} \in \mathcal{N}(\mathbf{p})} \|\mathbf{p} - \mathbf{q}\|, \tag{5}$$

and declare $\mathbf{p}$ a potential edge point if

$$\|\mathbf{p} - \mathbf{c}(\mathbf{p})\| > \lambda r \tag{6}$$

with *edge detection threshold parameter* λ. In that case, $\mathbf{p}$ is added to the list of potential edge points.

In the next step, we apply the Mean Shift clustering algorithm to detect clusters in the set of potential edge points, which will be fitted to lines. We enable *bin_seeding* for speed-up and use a predefined set of the *quantile* parameters for bandwidth estimation.

3.3.3. Line fitting and expansion

For each cluster of potential edge points of size at least 3, we run RANSAC with the LineModelND least square estimation [18] to fit a line. The algorithm returns a point on the fitted line and its unit direction vector. We save the number of inliers for each fitted line to be used as *weights* during the final symmetry plane voting. To ensure reproducibility, we set 100 as a predefined random state in RANSAC.

If the *line expansion* option is on, for each fitted line ℓ, we calculate the largest distance δ from ℓ to its inliers from the respective potential edge points cluster, then find all the points of the point cloud that are within δ-neighbourhood of ℓ, join them to the inlier set, and refit the line ℓ to this enlarged inlier set using LineModelND. Line expansion often helps in cases where edge detection is not accurate and some of the points on edges are missing.

3.3.4. Finding symmetry planes

At this point, we have a set of lines fitted to the edges of the 3D object, each parametrized by one of its points and unit direction vector $\mathbf{d}$. Now we iterate through all line pairs, for those that are parallel or intersect, construct the corresponding symmetry planes, and skip pairs of skew lines.

Assume that given are the lines $\ell_1(O_1, \mathbf{d}_1)$ and $\ell_2(O_2, \mathbf{d}_2)$; we set $\mathbf{q} = \overrightarrow{O_1O_2}$ and perform the following steps.

1. *Coplanarity check:* The lines ℓ_1 and ℓ_2 are considered coplanar if the mixed product of the vectors $\mathbf{d}_1$, $\mathbf{d}_1$, and $\mathbf{q}$ is sufficiently small, in the sense that
$$|(\mathbf{d}_1 \times \mathbf{d}_2) \cdot \mathbf{q}| < \varepsilon_2 \|\mathbf{q}\| \tag{7}$$
with *line closeness threshold* ε_2. Otherwise, the lines are considered skew and thus are skipped.
2. *Parallel lines:* The lines ℓ_1 and ℓ_2 are parallel if their direction vectors $\mathbf{d}_1$ and $\mathbf{d}_2$ are collinear up to some tolerance, e.g., if
$$|\mathbf{d}_1 \cdot \mathbf{d}_2| > 1 - \varepsilon_2; \tag{8}$$
to make sure the lines do not coincide, we require that also $\|\mathbf{q}\| > \varepsilon_2$ and
$$|\mathbf{d}_1 \cdot \mathbf{q}| < (1 - \varepsilon_2)\|\mathbf{q}\|. \tag{9}$$
If the lines ℓ_1 and ℓ_2 have been declared parallel, then their symmetry plane π is determined by its point C that is the midpoint of O_1 and O_2 and the unit normal vector $\mathbf{n}$ that is collinear to $\mathbf{q} - (\mathbf{q} \cdot \mathbf{d}_1)\mathbf{d}_1$; the distance d from the plane π to the origin in the Hough parametrization is then $d = (\overrightarrow{OO_1} + \overrightarrow{OO_2}) \cdot \mathbf{n}/2$.
3. *Intersecting lines:* Once the lines ℓ_1 and ℓ_2 have been found coplanar, not coincident, and not parallel, they are considered intersecting. Then there are two symmetry planes through the intersection point D and with the unit normal vectors $\mathbf{n}$ collinear to $\mathbf{d}_1 + \mathbf{d}_2$ or $\mathbf{d}_1 - \mathbf{d}_2$, respectively. We identify D as the midpoint of the interval D_1D_2 of the shortest length when $D_1 \in \ell_1$ and $D_2 \in \ell_2$. We find $D_1 = O_1 + s_1\mathbf{d}_1$ and $D_2 = O_2 + s_2\mathbf{d}_2$ by noting that the vector $\overrightarrow{D_1D_2}$ must be orthogonal to $\mathbf{d}_1$ and $\mathbf{d}_2$, so that
$$(\mathbf{q} + s_2\mathbf{d}_2 - s_1\mathbf{d}_1) \cdot \mathbf{d}_1 = 0, \tag{10}$$
$$(\mathbf{q} + s_2\mathbf{d}_2 - s_1\mathbf{d}_1) \cdot \mathbf{d}_2 = 0. \tag{11}$$
Solving this linear system for s_1 and s_2, we find D_1, D_2, their midpoint D, and thus the two symmetry planes.

Symmetry planes π constructed this way get their *weights* $w(\pi)$ equal to the smaller inlier numbers for the lines ℓ_1 and ℓ_2. After all symmetry planes have been found, we start the final symmetry plane voting.

3.4. Voting for the best symmetry plane

In the previous two steps, we constructed lists of potential symmetry planes for the 3D object using curvature-based and edge-based approaches. The best symmetry plane can now be chosen in three different ways: among the curvature-based planes only, among the edge-based planes only, or the best in the combined list. In the latter case, some preprocessing is needed to eliminate the effect of different scales of plane weights generated by the two methods.

The *first option* is to take n best symmetry planes from each list, where n is the smaller of two sizes, and the planes are ordered by their weights, from the largest to the smallest one. The *second option* is to rescale the weights for the curvature-based planes so that the total weights in both sets become equal. Experiments show that for some 3D objects, the first option is better, while for others—the second one, so we try both during the grid search of parameters.

Given the list of potential symmetry planes, we apply the Mean Shift clustering algorithm to determine the largest cluster. As a pre-processing step, we rescale the Hough coordinate d for the planes to
$$\tilde{d} := d/(\max|d|) \tag{12}$$

so that it should lie within the interval $[-1, 1]$. The reason is that the normal vectors of the planes are of the unit norm while their distances d to the origin are typically on the scale from -10^4 to 10^4 and thus would dominate during the voting. As before, we use the Mean Shift clustering with the enabled *bin_seeding* option and estimate the bandwidth using the predefined quantile parameter set.

Now we choose the biggest cluster $\mathcal{L}$ and take its weighted average as the best symmetry plane:

$$\mathbf{n}^* = c\frac{\sum_{\pi\in\mathcal{L}} w(\pi)\mathbf{n}(\pi)}{\sum_{\pi\in\mathcal{L}} w(\pi)}, \qquad \tilde{d}^* = \frac{\sum_{\pi\in\mathcal{L}} w(\pi)d(\pi)}{\sum_{\pi\in\mathcal{L}} w(\pi)}; \tag{13}$$

we then rescale $\tilde{d}^*$ to its actual value $d^* = \tilde{d}^* \cdot (\max|d|)$, cf. (12).

3.5. Model completion

After the best hypothetical mirror symmetry S_π for the point cloud $\mathcal{C}$ has been chosen, we use it to fill in the missing or damaged parts of $\mathcal{C}$ following the steps described in [1]. We form the mirrored object $\mathcal{C}' := S_\pi(\mathcal{C})$, and for each point $\mathbf{p}$ in $\mathcal{C}'$ decide if it should be added to $\mathcal{C}$. We do that if and only if there are no points of $\mathcal{C}$ among the $k = 6$ nearest neighbors of $\mathbf{p}$ in the combined point cloud $\mathcal{C}\cup\mathcal{C}'$. To speed up the kNN search, a k-d tree from this combined point cloud is constructed beforehand.

If the ground truth model $\mathcal{C}_0$ is available, we can decide on the quality of completion using the Chamfer distance between $\mathcal{C}_0$ and the completed $\mathcal{C}$; otherwise, some other measures are used (see Section 4.2).

4. Algorithm evaluation results

4.1. Dataset

The suggested completion algorithm described in Section 3 uses both mesh representation of objects as well as point clouds created from these meshes. We chose the Princeton ModelNet40 3D object dataset [19] for algorithm evaluation. It offers objects in 40 different categories. We selected 5 objects from each category that visually seemed symmetric. In this **200-object** dataset, we transformed every mesh into a point cloud (of 50,000 points each) so that each original undamaged object is represented both by a mesh and a point cloud.

To make 3D objects suitable for the completion task, we inflicted damage to each of the meshes. From each object, a random number (between **1 and 10**) of randomly placed regions of random size totaling approximately **15%** of the original size was removed; in addition, for the curvature computing algorithm to work, we removed all small triangle clusters. Before damaging, each mesh was scaled to have approximately a unit surface area; this makes the resulting metrics better comparable. In some cases, we performed subdivision of triangles of original meshes to create more realistic damages.

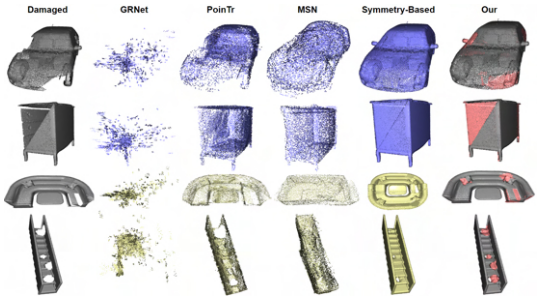

Figure 4: Comparing different approaches to object completion: [13], [14], [12], [1], ours.

4.2. Results and analysis

We evaluated the developed algorithm on the generated dataset. For each object, we used a grid search with a time limit to find optimal parameters for solo and combined versions of the algorithm (see Section 4.3). With those optimal parameters, we ran the algorithm to produce the symmetry planes and the corresponding completion metrics and saved the best results for solo and combined algorithms. We used following ***metrics*** for result evaluation:

1. The Chamfer distance between the original undamaged object and the completed object. This metric tells us how well the damaged object was completed by our algorithm.
2. Improvement rate metric shows how much better is completion $\mathcal{C}^*$ relative to leaving the object $\mathcal{C}$ uncompleted. When we get 0 it means that there was no completion, positive values indicate successful completion, and negative shows that during completion, points were added in places where they originally did not belong, and the object was deformed to some extent. With CD standing for the Chamfer distance and $\mathcal{C}_0$ the original (undamaged) object, this value is

$$100 * (1 - CD(\mathcal{C}, \mathcal{C}_0)/CD(\mathcal{C}^*, \mathcal{C}_0)). \tag{14}$$

Statistics, such as mean, median, minimum, and maximum values of the improvement rate, are presented in Table 1. It should be noted that for testing, we used models pretrained on ShapeNet, which were available to us; thus, it can be expected that methods that require training, showed smaller improvement metrics than traditional methods, meant for unseen data. In general, our algorithm performed similarly to the traditional method from [1].

Completion Method		Statistics of Improvement rate (in %) (without skipping)				(skipping results $\leq 0\%$)			
		Mean	Median	Min	Max	Mean	Median	Min	Max
MSN [12]		-224.2	-187.5	-1031.8	22.9	6.5	3.8	1.6	13.7
PoinTr [14]		**-9.2**	-6.4	**-103.7**	36.9	10.8	8.6	0.1	36.9
GRNet [13]		-10498	-4690	-118744	-8.1	-	-	-	-
Symmetry-Based [1]		-2751.4	0.5	-132980	85.5	**27.2**	23.8	0.1	85.5
Our	Edge-based	-202.1	-29.4	-1389	71.8	22.5	20.6	0.1	71.8
	Curvature-based	-386.8	**5.3**	-24147	**85.6**	27.1	24.8	**0.2**	**85.6**
	Combined	-177.0	-9.4	-13893	82.1	26.4	**25.3**	0.1	82.1

Table 1
Comparison of methods: statistical results with the improvement rate metric.

4.3. Parameter tuning

There are several parameters that control different aspects of the method. Some can only slightly alter precision, while others significantly affect the performance of the algorithm. To fine-tune the most important parameters for each object, we performed a grid search over a set of predefined choices established in numerous experiments on different meshes from several datasets; here is the resulting list:

- *Curvature closeness threshold* ε_1 controls the strictness of the comparison of curvatures of two points. The smaller it is (e.g., 0.001), the fewer points will be used to create symmetry planes, but the accuracy can be better. Usually, the real-world data contain noise and damages, so it is better to use a less strict threshold (e.g., 0.1) and mostly rely on a symmetry plane check on a neighboring patch of the points. Predefined range: $10^{-8}, 10^{-3}, 10^{-3}, 0.01, 0.05, 0.07, 0.1, 0.8, 1$.
- *Surface patch closeness threshold* defines how strict the patch symmetry verification is. We mirror reflect a neighbourhood (the kNN with $k = 200$) of the point A, superimpose it with a neighbourhood of its symmetric candidate B, and detect a change in the geometric shape. The threshold bounds from above the MSE of curvatures in the neighbourhood of B and thus controls the number of planes that pass. Predefined range: 0.001, 0.01, 0.05, 0.08, 0.1, 0.5, 0.8, 1, 1.2, 1.5, 1.8, 2, 2.5, 3.
- *Number of neighbours* in edge detection (200 by default) gives the size of a neighbourhood of a point $\mathbf{p}$ to determine if it is an edge point or not.
- *Edge detection threshold parameter* λ (default 3 or 0.1) determines which points are regarded as edge points. Smaller λ softens the condition and increases the number of edge points.
- *Edge points clustering quantile* for bandwidth estimation is used during Mean Shift clustering for edge detection. Small quantiles create more edge clusters but slower computation; too big quantiles lead to fewer clusters and make the algorithm less accurate. Predefined range: 0.001, 0.00525, 0.007, 0.01, 0.0125, 0.025, 0.03, 0.035, 0.0375, 0.05, 0.07, 0.075, 0.1, 0.125, 0.15, 0.2.
- *Line closeness threshold* ε_2 decides when points are considered coplanar and direction vectors collinear (absolute tolerance). Predefined range: 0.0001, 0.001, 0.003, 0.007, 0.01, 0.025, 0.0375, 0.05, 0.075, 0.1, 0.525, 0.7, 1, 1.5, 2.
- In RANSAC and linear model for line fitting, we used a default value 10 for *error* and limited *maximum number of iterations* by 1000.
- The boolean *Enabling/disabling line expansion* determines if the algorithm will perform an additional refitting step for each line or not.
- *Symmetry plane clustering quantile* for bandwidth estimation, used during Mean Shift clustering for best symmetry plane detection; should not be too big as otherwise plane averaging over the largest cluster will suffer from noise. Predefined range: 0.0005, 0.001, 0.005, 0.008, 0.01, 0.015, 0.02, 0.05, 0.08, 0.1, 0.15, 0.2, 0.25, 0.3, 0.5, 0.7, 0.8, 1, 1.1, 1.5, 2, 2.5.

Predefined sets of parameters, which we used for the grid search, were established through numerous experiments, including manual fine-tuning, modified bisection search, incrementing, etc. The grid search was then performed on each object from the dataset.

5. Conclusions

In this paper, we proposed, described, and evaluated a four-stage pipeline (*curvature matching*, *edge matching*, *symmetry plane voting*, *object completion*) of solving the completion task for previously unseen 3D objects possessing global mirror symmetry. This method uses the

local geometries of an object represented by principal curvatures and/or edges. We presented the results by solo algorithms (based on curvatures matching or on edges matching) and a combined algorithm (that performs the voting on a combined set of weighted planes) on the part of the ModelNet40 dataset with random occlusions. The approach demonstrates competitive results in comparison with the best deep learning and a symmetry-based method.

Limitations of the proposed method are that

(a) the object must possess a mirror symmetry;
(b) if missing parts are mirror symmetric or have a large intersection with their mirrored image, the object cannot be reconstructed by the algorithm;
(c) when the symmetry plane is found rather poorly (e.g., due to incomplete parameter tuning), then the completion is performed poorly as well;
(d) there is a significant number of parameters, some of which must be fine-tuned for each separate object, making our method less robust.

References

[1] T. Rumezhak, O. Dobosevych, R. Hryniv, V. Selotkin, V. Karpiv, M. Maksymenko, Towards realistic symmetry-based completion of previously unseen point clouds, in: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops, 2021.

[2] N. J. Mitra, L. J. Guibas, M. Pauly, Partial and approximate symmetry detection for 3d geometry, in: ACM SIGGRAPH 2006 Papers, 2006.

[3] S. M. Ahmed, Y. Z. Tan, C. M. Chew, A. A. Mamun, F. S. Wong, Edge and corner detection for unorganized 3d point clouds with application to robotic welding, in: 2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2018.

[4] N. J. Mitra, M. Pauly, M. Wand, D. Ceylan, Symmetry in 3d geometry: Extraction and applications, Comput. Graph. Forum (2013).

[5] M. Bokeloh, M. Wand, V. Koltun, H.-P. Seidel, Pattern-aware shape deformation using sliding dockers, ACM Trans. Graph. (2011).

[6] N. J. Mitra, A. Bronstein, M. Bronstein, Intrinsic regularity detection in 3d geometry, in: K. Daniilidis, P. Maragos, N. Paragios (Eds.), Computer Vision – ECCV 2010, 2010.

[7] K. Xu, H. Zhang, A. Tagliasacchi, L. Liu, G. Li, M. Meng, Y. Xiong, Partial intrinsic reflectional symmetry of 3d shapes, ACM Trans. Graph. (2009).

[8] P. Ji, X. Liu, A fast and efficient 3d reflection symmetry detector based on neural networks, Multimedia Tools and Applications (2019).

[9] H. Xie, H. Yao, S. Zhou, J. Mao, S. Zhang, W. Sun, Grnet: Gridding residual network for dense point cloud completion, 2020.

[10] Y. Zhou, S. Liu, Y. Ma, Nerd: Neural 3d reflection symmetry detector, 2021.

[11] W. Yuan, T. Khot, D. Held, C. Mertz, M. Hebert, Pcn: Point completion network, 2018.

[12] M. Liu, L. Sheng, S. Yang, J. Shao, S.-M. Hu, Morphing and sampling network for dense point cloud completion, Proceedings of the AAAI Conference on Artificial Intelligence (2020).

[13] H. Xie, H. Yao, S. Zhou, J. Mao, S. Zhang, W. Sun, Grnet: Gridding residual network for dense point cloud completion, 2020.

[14] X. Yu, Y. Rao, Z. Wang, Z. Liu, J. Lu, J. Zhou, Pointr: Diverse point cloud completion with geometry-aware transformers, 2021.

[15] A. Jacobson, D. Panozzo, et al., libigl: A simple C++ geometry processing library, 2018.

[16] D. Panozzo, E. Puppo, L. Rocca, Efficient multi-scale curvature and crease estimation, in: 2nd International Workshop on Computer Graphics, Computer Vision and Mathematics, GraVisMa 2010 - Workshop Proceedings, 2010.

[17] Q.-Y. Zhou, J. Park, V. Koltun, Open3D: A modern library for 3D data processing, arXiv:1801.09847 (2018).

[18] S. Van der Walt, J. L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J. D. Warner, N. Yager, E. Gouillart, T. Yu, scikit-image: image processing in Python, PeerJ (2014).

[19] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, J. Xiao, 3d shapenets: A deep representation for volumetric shapes, in: Proceedings of the IEEE conference on computer vision and pattern recognition, 2015.