A Pipeline to Generate Synthetic Dataset for Ceramic Fragments Recognition and Pairing

Serena Barone[1,2(✉)], Marta Magalini[2], Rodolfo Panerai[2], and Alessandro Bombini[2]

[1] Università di Firenze, Dipartimento di Fisica e Astronomia, Florence, Italy
serena.barone@unifi.it
[2] Istituto Nazionale di Fisica Nucleare, Firenze, Italy

Abstract. In this contribution, we present a pipeline for generating synthetic datasets of ceramic fragments. The pipeline simulates thousands of breakages of 3D ceramic objects, starting from an `.stl` file, with user control over the number of fragments produced. Following the breaking simulation, a custom-designed erosion process is applied to replicate the natural degradation that pottery fragments undergo during burial over thousands of years.

This approach directly tackles the issue of limited and hard-to-annotate datasets in archaeological settings, and with a single pipeline, it enables multiple deep learning tasks, such as fragment classification, orientation regression, and fragment pairing. Each resulting fragment is stored as an `.stl` file with a unique UUID, while adjacency information between fragments is collected and preserved.

The pipeline is implemented in Python using the Blender Python API and is fully containerised with a custom Singularity container. This enables parallelisation and distribution of multiple breaking simulations, facilitating large-scale dataset generation. The code is freely available ([4] https://github.com/CHNet-AI-Barumini/Synthetic-Fragment-Dataset-Generation). The resulting dataset is released under CC-BY-0 license.

Keywords: Machine Learning · Cultural Heritage · Open data · big data · Computer graphics

1 Introduction

The reconstruction of ceramic vessels from fragmented remains is a fundamental, yet arduous, task in archaeology. It provides critical insights into chronology, technology, trade, and cultural practices of past societies [25]. However, the process is often compared to solving a highly complex, three-dimensional puzzle with most of the pieces missing or degraded. Archaeologists routinely contend with excavations yielding thousands of irregular fragments (sherds), where the

© The Author(s), under exclusive license to Springer Nature Switzerland AG 2027
O. Gervasi et al. (Eds.): ICCSA 2026 Workshops, LNCS 16758, pp. 3–15, 2027.
https://doi.org/10.1007/978-3-032-30497-1_1

manual search for joints, a process known as refitting or pairing, is exceptionally time-consuming and requires a high degree of specialised skill and visual acuity [14,16,24]. Previous work has addressed fragment reassembly using geometric matching [14] or 3D scanning of pottery [16], but these approaches typically rely on hand-crafted features and do not scale to large collections. This challenge is particularly acute when fragments have suffered from post-depositional surface erosion, which smooths diagnostic features and fracture edges, obscuring the very evidence needed to determine if two pieces once belonged to the same vessel.

The Nuragic civilisation of Sardinia (1700 B.C.E–1000 B.C.E), Italy, provides a compelling case study of these difficulties. Nuragic sites are rich in ceramic material, but as detailed in [5], the pottery assemblage is far from uniform. It exhibits significant variability in production techniques, surface treatments (e.g., smooth surfaces versus red slip wares), and temper materials (e.g., calcite versus volcanic rock fragments), reflecting technological evolution and cultural contacts over centuries. This inherent variability, combined with the erosive effects of the Sardinian environment, means that Nuragic sherds from different periods and contexts can appear superficially similar, while adjacent fragments from the same vessel may differ significantly due to different preservation conditions. Consequently, the accurate reassembly of Nuragic ceramics remains a bottleneck, heavily reliant on the knowledge of expert archaeologists and restorers, and unsustainable for the vast quantities of material recovered.

In recent years, the field of 3D computer vision has made significant strides in addressing similar geometric reasoning problems through deep learning. The direct analysis of 3D point clouds, a representation well-suited for scanned archaeological fragments with photogrammetry or laser scanners, has been revolutionised by architectures that can handle unstructured data. The pioneering PointNet [22] achieved permutation invariance through point-wise multi-layer perceptrons (MLPs) and a symmetric aggregation function, but its ability to capture local geometric context was limited. This was addressed by PointNet++ [23], which introduces a hierarchical architecture with set abstraction modules to learn local features at multiple scales.

Concurrently, graph-based methods like Dynamic Graph CNN (DGCNN) have demonstrated impressive performance [27]. DGCNN constructs a dynamic k-nearest neighbor graph in the feature space, applying an EdgeConv operation to learn relationships between points and their neighbors, effectively capturing both local and global structures. More recently, Point Transformers [31] have emerged, applying attention mechanisms to model long-range dependencies and contextual relationships within point sets, achieving state-of-the-art results in tasks like completion and classification. These architectures have been successfully applied in fields ranging from autonomous driving and urban scene segmentation [13] to robotics [21]. For archaeological fragment analysis, these methods suggest a clear pathway: classification of fragments by type, regression to estimate their original orientation in 3D space, and, crucially, learning geometric features to predict which fragments are likely to pair.

Despite the promise of these deep learning models, their application to ceramic fragment reassembly is fundamentally hindered by a lack of suitable training data. Existing 3D point cloud datasets, while valuable for benchmarking, are inadequate for this specific archaeological task. Widely used synthetic datasets like **ModelNet40** [29], **ShapeNetPart** [30], and **Breaking Bad** [26] contain clean, complete, and manifold 3D models of chairs, tables, airplanes, and other manufactured objects. They lack the irregular fractures, surface erosion, and partial volumes that are the defining characteristics of real-world archaeological sherds. Real-world scene datasets such as **S3DIS** [3], **ScanNet** [7], and **Semantic3D** [12] focus on large-scale urban or indoor environments, with classes like 'building', 'vegetation', 'chair', and 'table'. They do not contain examples of fragmented objects or the subtle features required for reassembly. Crucially, no publicly available dataset with a permissive license (e.g., CC-BY) provides 3D models of broken and eroded ceramic fragments complete with the ground-truth adjacency information needed to train a supervised model for fragment pairing. Annotating such data manually from real archaeological finds would be prohibitively expensive and labour-intensive, requiring the very expert effort the method aims to supplement.

To overcome this data bottleneck, we propose a synthetic data generation pipeline. By simulating both the fracture and subsequent erosion processes on 3D models of whole ceramic vessels, we can generate an unlimited, varied, and perfectly annotated dataset. This approach directly addresses the limitations of existing datasets, providing the necessary scale and ground-truth labels to train deep learning models for fragment classification, orientation regression, and, most importantly, fragment pairing.

2 Methodology

2.1 Breaking Ceramics with Blender API

To simulate the breaking of a ceramic object, we relied on the Python API of the free and open source (FOSS) program Blender [6], a software for 3D content creation, sculpting and rendering. This allowed us to perform operations on 3D objects programmatically. For the breaking, we used the Blender Add-on `CellFracture`, which utilises a Voronoi tessellation approach: it creates a bounding box around the object, randomly selects a user-tunable number of seeds within the box, and then constructs a 3D Voronoi diagram using the seeds as cell centers. Subsequently, it performs boolean intersection operations between each Voronoi region and the original object, yielding the individual fragments.

The logical steps of the pipeline are as follows:

1. Import the object from an `.stl` file;
2. Fragment the object into a user-tunable number of pieces, obtaining a list of fragments, and assign a universally unique identifier (UUID) to each fragment;
3. Iterate over the fragments; for each fragment:
   (a) Perform re-triangulation and sharp remesh operations, which are necessary to properly sample mesh points on the fracture surfaces;

Input : *do_center*: boolean flag (moves the fragment's centre of mass to the coordinates (0,0, 0)), *min_mesh_pts_pieces*: minimum points per fragment, *max_fractures*: maximum fracture count, *path_to_stl*: input file path, *path_to_store*: output directory path
Output : Set of cleaned and exported fragments with UUIDs

```
// Import STL file
max_size, max_size_input ← LoadSTL(path_to_stl);
// Apply fracture simulation
BreakObject(max_fractures)
// Retrieve fracture collection
fracture_coll ← [obj | obj ∈ GetCollection('Amphora_breaks')];
// Generate UUIDs for all fragments
piece_uuid_map ← CreateUUIDDict(fracture_coll);
foreach obj ∈ fracture_coll do
    // Triangulate and remesh
    TriangulateRemesh(obj);
    // Get fragment size
    max_size_fragment ← GetObjectSize(obj);
    is_piece_ok ←
      CheckPiece(obj, min_mesh_pts_pieces, max_size_fragment);
    // Delete if too small
    if not is_piece_ok then
        Delete(obj);
        continue;
    end
    // Export valid fragment
    ExportObjToMesh(obj, path_to_store, do_center);
end
```

Algorithm 1: Fragment generation from single `.stl` object

(b) Validate the mesh: the fragment is retained only if it satisfies the following criteria: (i) it is a manifold mesh (i.e., each edge is shared by exactly two faces), (ii) it contains no self-intersections, (iii) it has a sufficient number of mesh points exceeding a user-defined threshold `min_mesh_pts_pieces`, and (iv) its bounding box dimensions exceed a minimum size threshold `MIN_MAX_SIZE`. Fragments failing any of these checks are discarded;
(c) Export the validated fragment mesh as an `.stl` file named `UUID.stl`, where `UUID` is the unique identifier assigned in step 2;

4. (Optional) Compute the adjacency matrix of fragments and store it as a JSON file. The JSON contains a dictionary mapping each fragment's UUID to a list of UUIDs of its neighbouring fragments.

The algorithm pseudocode is reported in Algorithm 1.

The pipeline described above generates a single breaking simulation for one object. Scaling this to create a large dataset, comprising hundreds of break simulations per object across dozens of object types is straightforward due to the

embarrassingly parallel[1] nature of the problem. While a single breaking job is relatively fast (on the order of tens of seconds), aggregating thousands of simulations can become time-consuming. However, because each simulation is independent, distributed computing is an ideal strategy. A distributed approach is preferable over simple multi-threading or multi-processing, as Blender's API already leverages multi-threading internally for its own operations.

To facilitate distribution, we containerised the pipeline using Apptainer (formerly known as Singularity) [17,18]. Containerising Blender for headless execution (i.e., without a graphical user interface) presents some challenges. Notably, Blender ships with its own Python distribution, which must be used instead of the system Python to ensure access to Blender's global variables and API. This requires using `/opt/blender/4.5/python/bin/pip` and `/opt/blender/4.5/python/bin/python` for package installation and script execution. Additionally, enabling the required add-ons necessitates running a dedicated Python script via:

```
/opt/blender/blender -background -factory-startup -python
./enable_addons.py
```

The complete Apptainer definition file is provided in Listing 1.1.

```
Bootstrap: docker
From: ubuntu:25.04
stage: build
%files
    # copy add-ons
    ./blender_modules/add-on-cell-fracture-v0.2.1.zip
        ↪ ./add-on-cell-fracture-v0.2.1.zip
    # copy script to enable addons
    ./build_scripts/enable_addons.py ./enable_addons.py
%post
    apt update && apt install -y \
        wget unzip \
        xz-utils \
        libgl1 libx11-6 libxi6 libxxf86vm1 libxcursor1 libxrandr2
            ↪ libxinerama1 libegl1 libdbus-1-3 libxkbcommon0 libsm6
    # Install Blender
    wget \url{https://download.blender.org/release/Blender4.5}
        ↪ /blender-4.5.0-linux-x64.tar.xz
    tar -xf blender-4.5.0-linux-x64.tar.xz
    mv blender-4.5.0-linux-x64 /opt/blender
    # Install pip for Blenders Python
    /opt/blender/4.5/python/bin/python3.11 -m ensurepip
    /opt/blender/4.5/python/bin/python3.11 -m pip install --upgrade
        ↪ pip
    # Install custom modules (example: numpy, tqdm)
    /opt/blender/4.5/python/bin/pip install numpy tqdm
    # install add-on
    # Create Blender add-ons directory
    mkdir -p /opt/blender/4.5/scripts/addons
    # Unzip your add-on into that folder
    unzip ./add-on-cell-fracture-v0.2.1.zip -d
        ↪ /opt/blender/4.5/scripts/addons
    # install
```

[1] An embarrassingly parallel workload is a computing problem that can be trivially divided into many independent sub-tasks requiring no communication or dependency between them.

```
    /opt/blender/blender --background --factory-startup --python
        ↪ ./enable_addons.py
%environment
    export PATH=/opt/blender:$PATH
    export PYTHONPATH=
        ↪ /opt/blender/4.5/python/lib/python3/site-packages
    export BLENDER_ADDON_PATHS=/opt/blender/4.5/scripts/addons
%runscript
    exec /opt/blender/blender "$@"
```

Listing 1.1. Apptainer definition file for the ceramic fragment simulation pipeline.

2.2 Erosion with PyTorch via ODE Resolution

To simulate erosion we use an ordinary differential equation (ODE) approach [9,10], relating mesh points to mesh curvature [8,15], plus an additional random noise to account for conservation quirks such as collisions.

The idea is that the variation in time of the position of mesh points $\Sigma = \{\mathbf{p}_i(t) \in \mathbb{R}^3\}_{i=1,\dots,N}$ is proportional to the mean curvature $\mathbf{H}_i = H_i \hat{\mathbf{n}}_i$ at that point, where $\hat{\mathbf{n}}_i$ is the normal outward vector to the surface Σ: the higher the curvature, the higher the variation, i.e.

$$\partial_t \mathbf{p}_i(t) \propto -H_i \hat{\mathbf{n}}_i \,, \qquad \forall i = 1, \dots, N.$$

The full formula is thus

$$\partial_t \mathbf{p}_i(t) = -\left(\mu H_i(t) \hat{\mathbf{n}}_i(t) + \sigma \xi_i(t)\right), \qquad \forall i = 1, \dots, N, \tag{1}$$

where μ, σ are two parameters (floating point), and $\xi_i \in \mathbb{R}^3$ is a random inward vector redrawn at each t.

To compute the mean curvature, we describe the (discretised) surface as a graph $\mathcal{G} = \{\mathbf{p}_i \in \mathbb{R}^3; e_{ij} \in [\{0\}, \{1\}]\}$, where e_{ij} is the adjacency matrix in sparse formulation. We can then estimate the mean curvature via the Laplace-Beltrami operator.

A standard and geometrically consistent way to discretise the Laplace - Beltrami operator on triangle meshes is the *cotangent formula*, first appearing in [20] and later systematised in discrete differential geometry and computer graphics [19].

Given a triangulated surface with vertices $\{\mathbf{p}_i\}$ and adjacency structure induced by the mesh, the Laplace-Beltrami operator at a vertex i is discretised as a weighted sum over its 1-ring neighbours:

$$\Delta \mathbf{p}_i \;\approx\; \frac{1}{2A_i} \sum_{j \in \mathcal{N}(i)} w_{ij} (\mathbf{p}_j - \mathbf{p}_i), \qquad w_{ij} = \frac{1}{2} \left(\cot \alpha_{ij} + \cot \beta_{ij}\right).$$

Here, α_{ij} and β_{ij} denote the two angles opposite to the edge (i, j), and A_i is the mixed (Voronoi or barycentric) area associated to vertex i [19]. This operator is a discrete counterpart of the smooth identity

$$\Delta_\Sigma \mathbf{p} = 2H\hat{\mathbf{n}},$$

where $H\hat{\mathbf{n}}$ is the mean curvature normal. Therefore, the vector

$$\mathbf{H}_i \approx \frac{1}{2A_i} \sum_j w_{ij}(\mathbf{p}_j - \mathbf{p}_i)$$

provides a consistent approximation of the mean curvature normal at vertex i, and the scalar mean curvature magnitude is obtained as

$$H_i = \|\mathbf{H}_i\|.$$

This discretisation is widely used because it is intrinsic, second-order accurate under mesh refinement, and enjoys strong variational properties [28]. It is also the formulation underlying many mesh processing algorithms such as smoothing, denoising, and curvature-flow evolution.

The PyTorch implementation (Listing 1.2) is a direct translation of this classical cotangent scheme: cotangent weights are assembled per face, aggregated into a sparse Laplacian stencil, and normalised by the dual areas to obtain the mean curvature magnitude.

The integration of the differential equation is performed for an user-defined number of time step of value Δt (another user tunable parameter) using a simple forward euler scheme

$$\mathbf{p}_i(t + \Delta t) = \mathbf{p}_i(t) - \Delta t \left(\mu H_i(t)\hat{\mathbf{n}}_i(t) + \sigma\xi_i(t)\right), \qquad \forall i = 1, \ldots, N. \tag{2}$$

Listing 1.2. Example usage of the ErodeMesh class.

```
from utils.generic_utils import open_stl, export_stl
from utils.custom_erosion import ErodeMesh

# 0. Open stl
piece = open_stl(path_to_file)
# 1. Instantiate erosion class
erode = ErodeMesh(
    steps = integration_steps,
    dt = dt,
    mu = mu, # 0.3-0.5
    sigma = sigma, # 0.05 - 0.2
    clamp_alpha = 0.10,
    seed = 42
)
# 2. Transform data with GPU acceleration
data_transformed = erode(piece.to(device)).cpu()
# 3. Store data
export_stl(data_transformed, f"{path_to_store}/
    custom_eroded_{filename}.stl")
```



Fig. 1. The five selected classes to validate the pipeline: from left to right: (a) Anfora, (b) Boccale, (c) Olla, (d) Tripode, (e) Brocca Askoide.

3 Application: Nuragic Pottery Fragments

We tested the pipeline on 5 different classes of nuragic ceramic objects. We selected them from the Italy "Ministero della Cultura" official website[2], choosing objects of Nuragic origin, with images in CC-BY-4.0 license [4]. We selected

1. Class 1: *Anfora*[3]
2. Class 2: *Boccale*[4]
3. Class 3: *Olla*[5]
4. Class 4: *Tripode*[6]
5. Class 5: *Brocca Askoide*[7]

We thus sculpted the 3D objects on Blender, starting from the 2D images and the geometric information present in the website. A visual representation of the sculpted 3D objects is reported in Fig. 1.

We have then created a dataset of fragments, by breaking each object 60 times, with a random number of breaking, uniformly distributed in the $[6, 16]$ range. This gave us a dataset of 3732 fragments.

For the erosion, we uniformly draw $\mu \in (0.3, 0.5), \sigma \in (0.05, 0.2)$, 3–5 integration steps, and Δt estimated as 0.1 of the mean curvature squared, which produced visually realistic erosion on the fracture surfaces while preserving overall shape.

In Fig. 2, we report the effect of erosion. It is easy to see that the effect of the Mean curvature smoothing effectively reduces the mesh sharpness on the breaking surface. Furthermore, the effect of noise presents the emergence of small

[2] https://catalogo.beniculturali.it/.
[3] https://catalogo.beniculturali.it/detail/ArchaeologicalProperty/2000202578.
[4] https://catalogo.beniculturali.it/detail/SARDEGNA/ArchaeologicalProperty/2000116164.
[5] https://catalogo.beniculturali.it/detail/ArchaeologicalProperty/2000202521.
[6] https://catalogo.beniculturali.it/detail/SARDEGNA/ArchaeologicalProperty/2000116220.
[7] https://catalogo.beniculturali.it/detail/ArchaeologicalProperty/2000202515.

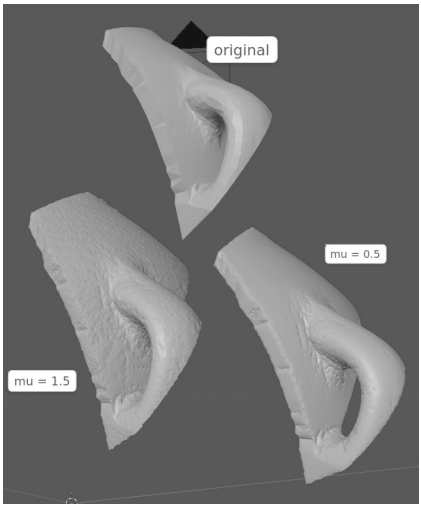

Fig. 2. Example of a fragment obtained from the pipeline: on top, uneroded; bottom left, eroded with $\mu = 1.5$; bottom right, eroded with $\mu = 0.5$.

concave regions. This effectively simulate the erosion, with adequate approximation, especially once a point cloud sampling is performed on the `.stl` (a standard pre-processing step in 3D computer vision).

In Fig. 3, we report the comparison of an original class object, one of its breaking, and that breaking with applied erosion. It is immediately evident that the effect of erosion is more pronounced on the fragment surfaces (as visible by the empty space created between fragments).



Fig. 3. Comparison, from left to right, of original, broken, and broken & eroded Anfora.

4 Conclusions and Outlook

In this work, we addressed the challenge of data scarcity in computational archaeology, particularly for the task of ceramic fragment reassembly. The manual reconstruction of broken ceramics, especially in contexts such as Nuragic archaeology, remains a time-consuming process that relies heavily on expert knowledge. While deep learning approaches have shown promise for 3D geometric reasoning tasks, their application to fragment analysis has been hindered by the lack of suitable training data that captures the irregular fractures, surface erosion, and adjacency relationships characteristic of real archaeological finds.

To overcome this bottleneck, we presented a fully containerised, open-source pipeline for generating synthetic datasets of ceramic fragments. The pipeline can easily simulate thousands of breakages of 3D ceramic objects using Blender python API, applying a custom-designed erosion process to replicate post- depositional surface degradation, and stores each fragment as an `.stl` file with a unique UUID while preserving adjacency information.

The main contributions of this work are threefold:

- A **ready-to-distribute, containerised pipeline** packaged with Apptainer, enabling reproducible and scalable fragment generation across distributed computing environments without complex dependency management;
- A **custom erosion module** implemented in PyTorch, which solves the mean curvature flow ordinary differential equation with cotangent Laplace-Beltrami discretisation, leveraging GPU acceleration for efficient processing of large mesh collections;
- A set of 5 `.stl` files, representing 5 selected classes of nuragic origin, released with CC-BY-0 license.
- A **freely available CC-BY-0 dataset** comprising 3.732 fragments derived from five distinct Nuragic ceramic vessel classes, complete with ground-truth adjacency information and unique identifiers for each fragment; we provide both the *uneroded* and *eroded* version of the dataset. See [4].

This work lays the groundwork for several areas of future research. The dataset and pipeline provide the necessary foundation for training deep learning models on tasks directly relevant to archaeological practice. Specifically, the generated data can be used to: (a) train fragment classification models to recognise the original vessel type from a single sherd; (b) develop orientation regression networks to estimate the original pose of a fragment in 3D space; (c) learn fragment pairing representations to predict which sherds are likely to join, based on geometric features of their fracture surfaces.

Beyond these immediate applications, the modular nature of the pipeline allows for easy extension: additional vessel types can be incorporated, erosion parameters can be tuned to model different depositional environments, and alternative fracture algorithms can be substituted.

Furthermore, the containerised approach facilitates seamless distribution of the data generation pipeline across heterogeneous computing nodes, from local workstations to high-performance computing clusters. This portability reduces

the data generation burden in two ways: it allows parallel execution on multiple nodes, drastically cutting the time required for large-scale dataset creation; and it streamlines usability by eliminating the need for manual environment configuration or dependency adaptation on each target system.

By releasing both the pipeline and the dataset under permissive open-source licenses, we aim to lower the barrier to entry for applying modern 3D computer vision techniques to archaeological problems, ultimately contributing to more efficient and scalable documentation and reconstruction of cultural heritage.

Acknowledgments. To conduct the work presented in this contribution, we resorted to the cloud computing Software-as-a-Service infrastructure made available by AI_INFN initiative [1] (formerly known as ML_INFN initiative, "Machine Learning at INFN" [2]), aim to foster Machine Learning activities at the Italian National Institute for Nuclear Physics (INFN), as well as some servers made available by INFN-CHNet [11], the network devoted to the application of nuclear techniques to Cultural Heritage.

The work of A.B. was funded by Progetto ICSC - Spoke 2 - Codice CN 00000013 - CUP I53C21000340006 - Missione 4 Istruzione e ricerca - Componente 2 Dalla ricerca all'impresa âĂŞ Investimento 1.4.

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Anderlini, L., et al.: Supporting the development of machine learning for fundamental science in a federated cloud with the ai_infn platform. arXiv preprint arXiv:2502.21266 (2025)
2. Anderlini, L., et al.: ML_INFN project: status report and future perspectives. EPJ Web of Conf. **295**, 08013 (2024). https://doi.org/10.1051/epjconf/202429508013
3. Armeni, I., et al.: 3D semantic parsing of large-scale indoor spaces. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1534–1543 (2016). https://doi.org/10.1109/CVPR.2016.170
4. Barone, S., Magalini, M., Bombini, A.: Synthetic dataset of ceramic fragments of nuragic origin, March 2026. https://doi.org/10.15161/oar.it/396dr-9yj86
5. Campus, F., Leonelli, V.: La tipologia della ceramica nuragica. Il materiale edito (2000)
6. Blender Online Community: Blender - a 3D modelling and rendering package. Blender Foundation, Stichting Blender Foundation, Amsterdam (2018). http://www.blender.org
7. Dai, A., Chang, A.X., Savva, M., Halber, M., Funkhouser, T., Nießner, M.: ScanNet: richly-annotated 3D reconstructions of indoor scenes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 5828–5839 (2017). https://doi.org/10.1109/CVPR.2017.261
8. Desbrun, M., Meyer, M., Schröder, P., Barr, A.H.: Implicit fairing of irregular meshes using diffusion and curvature flow. In: Proceedings of the 26th Annual Conference on Computer Graphics and Interactive Techniques, pp. 317–324 (1999)
9. Domokos, G., Gibbons, G.: Geometrical and physical models of abrasion. arXiv preprint arXiv:1307.5633 (2013)

10. Domokos, G., Sipos, A.Á., Várkonyi, P.L.: Countinuous and discrete models for abrasion processes. Periodica Polytechnica Architecture **40**(1), 3–8 (2009)
11. Giuntini, L., et al.: Detectors and cultural heritage: the INFN-CHNet experience. Appl. Sci. **11**(8) (2021). https://doi.org/10.3390/app11083462
12. Hackel, T., Savinov, N., Ladicky, L., Wegner, J.D., Schindler, K., Pollefeys, M.: Semantic3D.net: a new large-scale point cloud classification benchmark. ISPRS Ann. Photogram. Remote Sens. Spat. Inf. Sci. **IV-1/W1**, 91–98 (2017). https://doi.org/10.5194/isprs-annals-IV-1-W1-91-2017
13. Hu, Q., et al.: RandLA-Net: efficient semantic segmentation of large-scale point clouds. In: Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR), pp. 11108–11117 (2020). https://doi.org/10.1109/CVPR42600.2020.01112
14. Huang, Q.X., Flöry, S., Gelfand, N., Hofer, M., Pottmann, H.: Reassembling fractured objects by geometric matching. In: ACM SIGGRAPH 2006 Papers, pp. 569–578 (2006)
15. Huisken, G.: Flow by mean curvature of convex surfaces into spheres. J. Differ. Geom. **20**(1), 237–266 (1984)
16. Kampel, M., Sablatnig, R.: Virtual reconstruction of broken and unbroken pottery. In: Proceedings of the Fourth International Conference on 3-D Digital Imaging and Modeling, 2003, 3DIM 2003, pp. 318–325. IEEE (2003)
17. Kurtzer, G., Muriki, K.: Singularity v1.x, April 2016. https://doi.org/10.5281/zenodo.1310023, https://www.osti.gov/biblio/code-54923
18. Kurtzer, G.M., Sochat, V., Bauer, M.W.: Singularity: scientific containers for mobility of compute. PLoS ONE **12**(5), e0177459 (2017)
19. Meyer, M., Desbrun, M., Schröder, P., Barr, A.H.: Discrete differential-geometry operators for triangulated 2-manifolds. In: Hege, HC., Polthier, K. (eds.) Visualization and Mathematics III, pp. 35–57. Springer, Heidelberg (2003). https://doi.org/10.1007/978-3-662-05105-4_2
20. Pinkall, U., Polthier, K.: Computing discrete minimal surfaces and their conjugates. In: Experimental Mathematics, vol. 2, pp. 15–36. Citeseer (1993)
21. Qi, C.R., Liu, W., Wu, C., Su, H., Guibas, L.J.: Frustum PointNets for 3D object detection from RGB-D data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 918–927 (2018). https://doi.org/10.1109/CVPR.2018.00102
22. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: PointNet: deep learning on point sets for 3D classification and segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 652–660 (2017). https://doi.org/10.1109/CVPR.2017.16
23. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: PointNet++: deep hierarchical feature learning on point sets in a metric space. In: Advances in Neural Information Processing Systems (NeurIPS), vol. 30, pp. 5099–5108 (2017)
24. Rasheed, N.A., Nordin, M.J.: Classification and reconstruction algorithms for the archaeological fragments. J. King Saud Univ. Comput. Inf. Sci. **32**(8), 883–894 (2020)
25. Rice, P.M.: Pottery Analysis: A Sourcebook, 2nd edn. University of Chicago Press (2015)
26. Sellán, S., Chen, Y.C., Wu, Z., Garg, A., Jacobson, A.: Breaking bad: a dataset for geometric fracture and reassembly. Adv. Neural. Inf. Process. Syst. **35**, 38885–38898 (2022)

27. Wang, Y., Sun, Y., Liu, Z., Sarma, S.E., Bronstein, M.M., Solomon, J.M.: Dynamic graph CNN for learning on point clouds. ACM Trans. Graph. (TOG) **38**, 146:1–146:12 (2019). https://doi.org/10.1145/3326362
28. Wardetzky, M.: Convergence of the cotangent formula: an overview. In: Bobenko, A.I., Sullivan, J.M., Schröder, P., Ziegler, G.M. (eds.) Discrete Differential Geometry. Oberwolfach Seminars, pp. 275–286 (2008). https://doi.org/10.1007/978-3-7643-8621-4_15
29. Wu, Z., et al.: 3D ShapeNets: a deep representation for volumetric shapes. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1912–1920 (2015). https://doi.org/10.1109/CVPR.2015.7298801
30. Yi, L., et al.: A scalable active framework for region annotation in 3D shape collections. ACM Trans. Graph. (TOG) **35**(6), 210:1–210:12 (2016). https://doi.org/10.1145/2980179.2980238
31. Zhao, H., Jiang, L., Jia, J., Torr, P.H., Koltun, V.: Point transformer. In: Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV), pp. 16259–16268 (2021). https://doi.org/10.1109/ICCV48922.2021.01595