ABC: A Big CAD Model Dataset For Geometric Deep Learning Supplementary Material

Sebastian Koch
TU Berlin
s.koch@tu-berlin.de

Albert Matveev
Skoltech, IITP
albert.matveev@skoltech.ru

Zhongshi Jiang
New York University
jiangzs@nyu.edu

Francis Williams
New York University
francis.williams@nyu.edu

Alexey Artemov
Skoltech
a.artemov@skoltech.ru

Evgeny Burnaev
Skoltech
e.burnaev@skoltech.ru

Marc Alexa
TU Berlin
marc.alexa@tu-berlin.de

Denis Zorin
New York University
dzorin@cs.nyu.edu

Daniele Panozzo
New York University
panozzo@nyu.edu

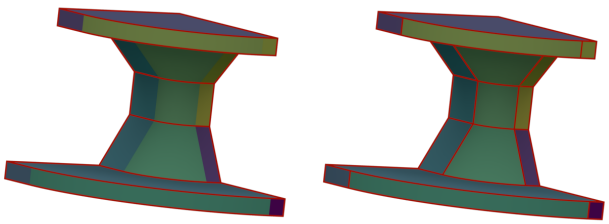

Figure 1: Example model with differently colored patches and highlighted sharp feature curves on the left as well as all feature curves on the right.

1. Model Filtering and Post-Processing

We filter out defective and low quality models in the Onshape collection using a set of automatic filters, which users can modify depending on their application. First, empty CAD models are filtered by file size of the original STEP files. Second, models that only consist of a single primitive or models that require shape healing during the translation and meshing phase are also filtered out. At this stage, we also filter by file size of the resulting meshes to avoid extremely large meshes that occur in some corner cases, where the meshing algorithm overly refines the CAD model. For the benchmark datasets we have two additional post-processing steps. The full models are generated by resampling the triangle meshes to match the defined numbers of vertices [6]. For the patches, we perform breadth-first search starting from a random selection of vertices to generate patches of pre-defined sizes.

2. File Types Description

Every model in the ABC-Dataset is stored in three different representations and multiple filetypes.

2.1. Boundary Representation/CAD

This is the original format acquired from Onshape, which contains an explicit description of the topology and geometry information of the CAD models in STEP and Parasolid format. The STEP files can be read and processed with Open Cascade [1] and Gmsh [7]. The processing allows for example to sample at arbitrary resolution, to generate meshes and to extract differential quantities.

2.2. Discrete Triangle Meshes

The discrete triangle meshes are supplied in two formats. The first is an STL file which is generated from the Parasolid format by Onshape with a high resolution. While these meshes are faithful approximations of the original geometry, the mesh quality is low: the triangles may have bad aspect ratios, and the sampling can be highly non-uniform, which is undesirable for many geometry processing algorithms. We thus also provide a second mesh, in OBJ format, produced by our processing pipeline. Our result is fairly regular, with a uniform vertex distribution and most triangles have angles close to $60°$. In addition to the triangle mesh itself, differential properties are analytically derived from the boundary representation and stored in these OBJ files. The vertices and faces of the OBJ are matched with the curves and patches stored in the YAML representation described in Section 2.3. Note that OBJ uses 1-indexing of the vertices, whereas we use 0-indexing in YAML.

2.3. Curve and Patch Features

The boundary representation of the STEP files defines surfaces and curves of different types. In addition to the geometrical information in the files listed above, we store the defining properties of surfaces and curves with references to the corresponding vertices and faces of the discrete triangle mesh representation. All this information is stored in YAML files [3], which contain a list of patches, and a list of curves, describing the boundary of the patches. Figure 1 shows one example model where different patches are highlighted in different colors and feature curves are drawn as red lines, all loaded from the OBJ and YAML files.

Curves. The curves list contains all curves of the CAD model. For each curve, different information is given depending on its type.

type Line, Circle, Ellipse, BSpline, Other.

sharp True if this curve is a sharp feature curve.

vert_indices List of all mesh vertex indices that are sampled from the curve (0-indexed).

vert_parameters List of the parameters that describe the corresponding mesh vertices.

Line $\mathbf{c}(t) = \mathbf{l} + t \cdot \mathbf{d}$

- location($\mathbf{l}$): The location vector of the line.
- direction($\mathbf{d}$): The direction vector of the line.

Circle $\mathbf{c}(t) = \mathbf{l} + r \cdot \cos(t) \cdot \mathbf{x} + r \cdot \sin(t) \cdot \mathbf{y}$

- location($\mathbf{l}$): The center of the circle.
- z_axis: The normal axis of the plane of the circle.
- radius(r): The radius of the circle.
- x_axis($\mathbf{x}$): The first axis of the local coordinate system.
- y_axis($\mathbf{y}$): The second axis of the local coordinate system.

Ellipse $\mathbf{c}(t) = \mathbf{l} + r_x \cdot \cos(t)\mathbf{x} + r_y \cdot \sin(t) \cdot \mathbf{y}$

- focus1: The first focal point of the ellipse.
- focus2: The second focal point of the ellipse.
- x_axis($\mathbf{x}$): The longer/major axis of the ellipse.
- y_axis($\mathbf{y}$): The shorter/minor axis of the ellipse.
- z_axis: The normal axis of the plane of the ellipse.
- x_radius(r_x): The major radius of the ellipse.
- y_radius(r_y): The minor radius of the ellipse.

BSpline Spline curves defined by control points, knots, and optionally weights

- rational: True if the B-Spline is rational.
- closed: True if the B-Spline describes a closed curve.
- continuity: The order of continuity of the B-Spline functions.
- degree: The degree of the B-Spline polynomial functions.
- poles: The control points of the B-Spline.
- knots: The knot vector with duplicate knots in case of multiplicity greater than 1.
- weights: The weights of the B-Spline curve (only used if it is a rational NURBS curve).

Patches. The patches list contains all patches of the CAD model. For each patch, different information is given depending on its type.

type Plane, Cylinder, Cone, Sphere, Torus, Revolution, Extrusion, BSpline, Other.

vert_indices List of all mesh vertex indices that are part of the patch (0-indexed).

vert_parameters List of the parameters that describe the according mesh vertices.

face_indices List of all face indices that are part of the patch (0-indexed).

Plane $\mathbf{p}(u, v) = \mathbf{l} + u \cdot \mathbf{x} + v \cdot \mathbf{y}$

- location($\mathbf{l}$): The location vector of the plane.
- x_axis($\mathbf{x}$): The first axis of the plane coordinate system.
- y_axis($\mathbf{y}$): The second axis of the plane coordinate system.
- z_axis: The normal axis of the plane.
- coefficients: Coefficients for the cartesian description of the plane: $c[0] \cdot x + c[1] \cdot y + c[2] \cdot z + c[3] = 0.0$.

Cylinder $\mathbf{p}(u, v) = \mathbf{l} + r \cdot cos(u) \cdot \mathbf{x} + r \cdot sin(u) \cdot \mathbf{y} + v \cdot \mathbf{z}$

- location($\mathbf{l}$): The location vector defining the base plane.
- x_axis($\mathbf{x}$): The first axis of the cylinder coordinate system.
- y_axis($\mathbf{y}$): The second axis of the cylinder coordinate system.

- z_axis($\mathbf{z}$): The rotation/center axis of the cylinder.
- coefficients: Coefficients for the cartesian quadric description of the cylinder: $c[0] \cdot x^2 + c[1] \cdot y^2 + c[2] \cdot z^2 + 2 \cdot (c[3] \cdot x \cdot y + c[4] \cdot x \cdot z + c[5] \cdot y \cdot z) + 2 \cdot (c[6] \cdot x + c[7] \cdot y + c[8] \cdot z) + c[9] = 0.0$.

Cone $\mathbf{p}(u, v) = \mathbf{l} + (r + v \cdot \sin(a)) \cdot (\cos(u) \cdot \mathbf{x} + \sin(u) \cdot \mathbf{y}) + v \cdot \cos(a) \cdot \mathbf{z}$

- location($\mathbf{l}$): The location vector defining the base plane.
- x_axis($\mathbf{x}$): The first axis of the cone coordinate system.
- y_axis($\mathbf{y}$): The second axis of the cone coordinate system.
- z_axis(z): The rotation/center axis of the cone.
- coefficients: Coefficients for the Cartesian quadric description of the cone: $c[0] \cdot x^2 + c[1] \cdot y^2 + c[2] \cdot z^2 + 2 \cdot (c[3] \cdot x \cdot y + c[4] \cdot x \cdot z + c[5] \cdot y \cdot z) + 2 \cdot (c[6] \cdot x + c[7] \cdot y + c[8] \cdot z) + c[9] = 0.0$.
- radius(r): The radius of the circle that describes the intersection of the cone and base plane.
- angle(a): The half-angle at the apex of the cone.
- apex: The apex/tip of the cone.

Sphere $\mathbf{p}(u, v) = \mathbf{l} + r \cdot cos(v) \cdot (cos(u) \cdot \mathbf{x} + sin(u) \cdot \mathbf{y}) + r \cdot sin(v) \cdot \mathbf{z}$

- location($\mathbf{l}$): The location vector defining center of the sphere.
- x_axis($\mathbf{x}$): The first axis of the sphere coordinate system.
- y_axis($\mathbf{y}$): The second axis of the sphere coordinate system.
- z_axis($\mathbf{z}$): The third axis of the sphere coordinate system.
- coefficients: Coefficients for the Cartesian quadric description of the sphere: $c[0] \cdot x^2 + c[1] \cdot y^2 + c[2] \cdot z^2 + 2 \cdot (c[3] \cdot x \cdot y + c[4] \cdot x \cdot z + c[5] \cdot y \cdot z) + 2 \cdot (c[6] \cdot x + c[7] \cdot y + c[8] \cdot z) + c[9] = 0.0$.
- radius(r): The radius of the sphere.

Torus $\mathbf{p}(u, v) = \mathbf{l} + (r_{max} + r_{min} \cdot cos(v)) \cdot (cos(u) \cdot \mathbf{x} + \sin(u) \cdot \mathbf{y}) + r \cdot \sin(v) \cdot \mathbf{z}$

- location($\mathbf{l}$): The location defining center of the torus.
- x_axis($\mathbf{x}$): The first axis of the torus coordinate system.
- y_axis($\mathbf{y}$): The second axis of the torus coordinate system.
- z_axis($\mathbf{z}$): The rotation/center axis of the torus.
- max_radius(r_{max}): The major/larger radius of the torus.
- min_radius(r_{min}): The minor/smaller radius of the torus.

Revolution Surface of revolution: a curve is rotated around the rotation axis.

- location: A point on the rotation axis
- z_axis: The rotation axis dirction.
- curve: The rotated curve that can be of any of the curve types.

Extrusion Surface of linear extrusion: a curve is extruded along a direction.

- direction: The linear extrusion direction of the surface (v parameter).
- curve: The extruded curve that can be of any of the curve types (u parameter).

BSpline Spline patch defined by control points, knots, and optionally weights.

- u_rational: True if the B-Spline is rational in u direction.
- v_rational: True if the B-Spline is rational in v direction.
- u_closed: True if the B-Spline describes a closed surface in u direction.
- v_closed: True if the B-Spline describes a closed surface in v direction.
- continuity: The order of continuity of the B-Spline functions.
- u_degree: The degree of the B-Spline polynomial functions in u direction.
- v_degree: The degree of the B-Spline polynomial functions in v direction.
- poles: 2D array of control points. The first dimension corresponds to the u direction, the second dimension to the v direction.
- u_knots: The knot vector for u with duplicate knots in case of multiplicity greater than 1.
- v_knots: The knot vector for v with duplicate knots in case of multiplicity greater than 1.
- weights: 2D array of the weights of the NURBS patch, corresponding to the control points (only used if the patch is rational).

Method	PN++	DGCNN	PwCNN	PCNN	Laplace	PCPN	ExtOp
Parameters	1402180	1724100	10207	11454787	1003395	3469255	8189635

Table 1: Overview of the network capacities for the machine learning approaches.

3. Implementation Details

Parameters. The overall number of parameters used by each method is listed in Table 1, while the running time is listed in Table 3 in this document. In the following we give a brief overview of the modifications and settings we used for each method in our comparison. One common change that we performed on all methods is to switch to the cosine loss described in the main article (Section 5.1), and we provided a maximum allowed time of 3 days.

Point Convolutional Neural Networks by Extension Operators [4]. The architecture is the same as the one proposed for classification, but with the last layer producing 3 values instead of 10. For each of the input sizes, we changed the size of the input layer accordingly. We used a step size of 10^{-3} using stochastic gradient descent. For the datasets with patch sizes of 512 and 1024, we used a minibatch size of 32. For the datasets with a patch size of 2048, we used a minibatch size of 16. We trained each network for up to 250 epochs.

Surface Networks [12]. In some rare cases, there are degenerate triangles in the triangle meshes, which pose challenge for discrete Laplacian operator computation. When the coefficients overflow, we replace these coefficients with 1. For training, we use 300 epochs, with Adam [11] optimizer and learning rate starting from 10^{-3}, and after 100 epochs, halved at every 20 epochs.

PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space [14]. Since no experiments for normal estimation were provided in the paper, we decided that the most natural way to predict normals with PointNet is by modifying the segmentation architecture, predicting three continuous numbers and normalizing them. We trained for 100 epochs with default settings and batch size 16.

Dynamic Graph CNN for Learning on Point Clouds [15]. We adapted the segmentation architecture by changing the last layer of the model to output 3 continuous values. Normalization was applied to ensure the output vector has unit length. We trained for 100 epochs with the default settings used for segmentation in the original implementation, batch size 16.

PCPNet: Learning Local Shape Properties from Raw Point Clouds [8]. We used exactly the same architecture specified in the original paper [8]. We trained with the Adam [11] optimizer using a step size of 1×10^{-3}, $\beta = (0.9, 0.99)$, and $\epsilon = 1 \times 10^{-8}$, and no weight decay. Training was run for up to 2000 epochs.

Pointwise Convolutional Neural Networks [9]. We adapted PwCNN in the same fashion as PointNet++ and DGCNN. We used 100 epochs for training, and took default segmentation settings from the original implementation with batch size 16.

PointCNN [13]. We took the PointCNN segmentation architecture, changed the output dimensionality to 3, and ran the training procedure with 100 epochs and default settings, batch size 8.

Osculating Jets [5]. For the mesh version we use the default parameters of the CGAL implementation. In the point cloud version, we use the 10 nearest neighbours for the jet fitting.

Robust Statistical Estimation on Discrete Surfaces [10]. For the mesh version of RoSt we use the default parameters set by the authors. The point cloud version is supplied with estimated normals from locally fitting a plane to the 10 nearest neighbours of each point. The method then refines these normals.

4. Running Times

We report the training time for data-driven methods in Table 2, and the running times for analytic methods in Table 3. We capped the *training* time to three days, although the instances are trained on different machines. The colors in the table denote the GPU model that has been used for the training. For the analytic methods, the times are measured for running the normal estimation on one CPU core (Intel Core i7).

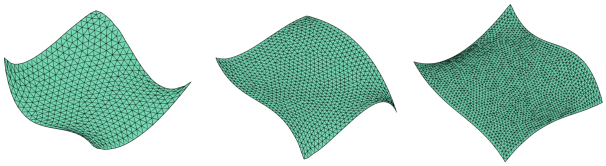

Figure 2: Example NURBS patches of the three different sampling densities with 512, 1024, and 2048 vertices (from left to right).

Method	#V	Full Models 10k	Full Models 50k	Full Models 100k	Patches 10k	Patches 50k	Patches 100k
PN++	512	173	857	1319	173	871	1241
	1024	184	919	1615	183	920	1613
	2048	226	1196	2124	227	1139	2328
DGCNN	512	155	734	819	155	784	814
	1024	299	1158	1257	222	1163	1268
	2048	490	2462	2467	487	2401	2498
PwCNN	512	258	1173	971	244	1256	891
	1024	515	2626	2027	471	3413	1773
	2048	1293	TO	TO	1319	TO	4179
PCNN	512	614	3037	3375	616	3071	3483
	1024	675	3370	3599	659	3368	3727
	2048	862	4248	4298	848	4110	TO
Laplace	512	492	1438	2793	285	1402	2747
	1024	1001	*3134**	TO	983	2845	*TO**
	2048	1939	*TO**	*TO**	1251	2807	TO

Table 2: Training time (in minutes) for all evaluated methods for the full model and patch benchmarks. Coloring indicates different NVidia GPUs, with default GTX 1080Ti, Tesla V100, Tesla P100, Titan V, Titan X, Titan Xp. TO: training process time-out of 3-day (4320 min) limit. This occurred for all trainings of PCPNet and ExtOp. *: Using more than two GPU (serially) during single training instance, typically with GTX 1080Ti and Titan Xp.

Method	#V	Full Models 10k	Full Models 50k	Full Models 100k	Patches 10k	Patches 50k	Patches 100k
RoSt PC	512	9.6	48.7	–	9.2	46.9	–
	1024	17.4	87.1	–	16.7	81.8	–
	2048	32.6	161.6	–	30.4	148.1	–
Jets PC	512	6.5	32.3	–	5.9	29.7	–
	1024	12.3	61.6	–	11.3	57.63	–
	2048	23.6	117.8	–	22.1	114.9	–
Uniform	512	0.5	2.8	5.7	0.4	2.7	6.6
	1024	0.8	3.9	7.9	0.6	2.9	8.8
	2048	1.1	6.0	12.0	1.1	4.6	12.1

Table 3: Running time (in minutes) for traditional methods for the full model and patch benchmarks.

5. NURBS Patches Experiment

We perform an additional experiment on a synthetic dataset, where normals are supposedly easier to infer as the space of possible shapes is artificially restricted, giving an advantage to data-driven methods over traditional ones. We generated two datasets of 10k and 50k random bi-variate B-Spline surface patches of the same three different sampling densities: 512, 1024 and 2048 (see Figure 2). Each patch was generated by randomly picking between $0.1 \times$ samples and $2.0 \times$ samples points, choosing random values with uniform probability between 0 and 1, and computing an interpolating bicubic spline surface using the function `scipy.interpolate.SmoothBivariateSpline` [2]. By construction, the surface is smooth and the normals are thus easier to predict than on real-world geometric models. The relative performance of data-driven and traditional methods is very similar to the one reported in the paper (Section 5), confirming the trend observed on real-world models.

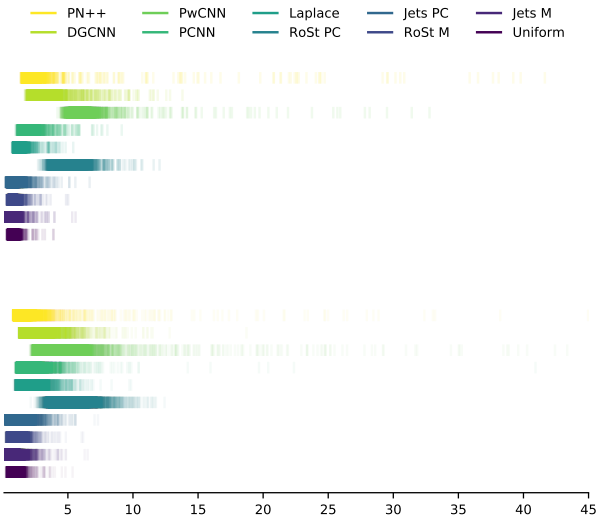

Figure 3: Plot of angle deviation error for the lower resolution NURBS (512 points) benchmark, using different sample size (top to bottom: 10k and 50k).

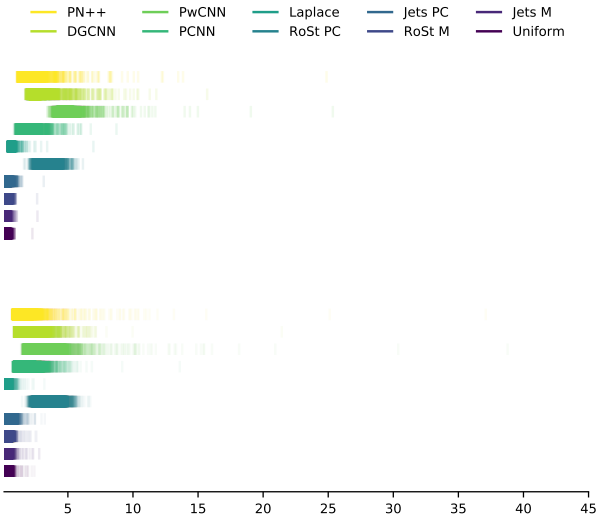

Figure 4: Plot of angle deviation error for the medium resolution NURBS (1024 points) benchmark, using different sample size (top to bottom: 10k and 50k).

References

[1] Open CASCADE Technology OCCT. https://www.opencascade.com/. Accessed: 2018-11-11.

[2] SciPy Interpolate Package. https://docs.scipy.org/doc/scipy/reference/generated/scipy.interpolate.SmoothBivariateSpline.html. Accessed: 2018-11-20.

[3] Yaml. http://yaml.org/. Accessed: 2018-11-14.

[4] M. Atzmon, H. Maron, and Y. Lipman. Point convolutional neural networks by extension operators. *ACM Trans. Graph.*, 37(4):71:1–71:12, July 2018.

[5] F. Cazals and M. Pouget. Estimating differential quantities using polynomial fitting of osculating jets. *Computer Aided Geometric Design*, 22(2):121 – 146, 2005.

[6] M. Garland and P. S. Heckbert. Surface simplification using quadric error metrics. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '97, pages 209–216, New York, NY, USA, 1997. ACM Press/Addison-Wesley Publishing Co.

[7] C. Geuzaine and J. F. Remacle. Gmsh: a three-dimensional finite element mesh generator with built-in pre- and post-processing facilities. *International Journal for Numerical Methods in Engineering*, 2009.

[8] P. Guerrero, Y. Kleiman, M. Ovsjanikov, and N. J. Mitra. PCPNet: Learning local shape properties from raw point clouds. *Computer Graphics Forum*, 37(2):75–85, 2018.

[9] B.-S. Hua, M.-K. Tran, and S.-K. Yeung. Pointwise convolutional neural networks, 2017. cite arxiv:1712.05245Comment: 10 pages, 6 figures, 10 tables. Paper accepted to CVPR 2018.

[10] E. Kalogerakis, P. Simari, D. Nowrouzezahrai, and K. Singh. Robust statistical estimation of curvature on discretized surfaces. In *Proceedings of the Fifth Eurographics Symposium on Geometry Processing*, SGP '07, pages 13–22, Aire-la-Ville, Switzerland, Switzerland, 2007. Eurographics Association.

[11] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[12] I. Kostrikov, Z. Jiang, D. Panozzo, D. Zorin, and B. Joan. Surface networks. In *IEEE Conference on Computer Vision and Pattern Recognition, CVPR*, 2018.

[13] Y. Li, R. Bu, M. Sun, and B. Chen. Pointcnn. *CoRR*, abs/1801.07791, 2018.

[14] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. Pointnet++: Deep hierarchical feature learning on point sets in a metric space. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 5099–5108. Curran Associates, Inc., 2017.

[15] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. Dynamic graph cnn for learning on point clouds. *CoRR*, abs/1801.07829, 2018.