Supplemental Material - DeltaConv: Anisotropic Operators for Geometric Deep Learning on Point Clouds

RUBEN WIERSMA, Delft University of Technology, The Netherlands
AHMAD NASIKUN, Delft University of Technology, The Netherlands and Universitas Gadjah Mada, Indonesia
ELMAR EISEMANN, Delft University of Technology, The Netherlands
KLAUS HILDEBRANDT, Delft University of Technology, The Netherlands

1 DISCRETIZED OPERATORS

Gradient. We construct a discrete gradient using the moving least-squares approach from [Liang and Zhao 2013]. We go through each step to show how to derive the gradient starting with the general formula from Riemannian geometry and simplify terms whenever the setting allows us to do so.

We locally fit a surface patch to estimate the metric at each point p using moving least-squares [Nealen 2004]. The surface patch $\mathbf{\Gamma} : \Omega \subset \mathbb{R}^2 \to \mathbb{R}^3$, often called a Monge patch, describes the surface as a quadratic polynomial $h(u,v)$ over the tangent plane at p and is given by

$$\mathbf{\Gamma}(u,v) = [u, v, h(u,v)]^\top, \tag{1}$$

where u, v denote local coordinates in the tangent plane. Since the surface patch should interpolate the point p and the surface normal of the patch at p should agree with the normal of the tangent plane at p, the constant and linear terms of $h(u,v)$ vanish

$$h(u,v) = \alpha_1 u^2 + \alpha_2 uv + \alpha_3 v^2, \tag{2}$$

$$h_u = 2\alpha_1 u + \alpha_2 v, \tag{3}$$

$$h_v = \alpha_2 u + 2\alpha_3 v. \tag{4}$$

The metric is given as

$$g = \begin{bmatrix} 1 + h_u^2 & h_u h_v \\ h_u h_v & 1 + h_v^2 \end{bmatrix}. \tag{5}$$

And its determinant as

$$|g| = (1 + h_u^2)(1 + h_v^2) - (h_u h_v)^2 \tag{6}$$

$$= 1 + h_u^2 + h_v^2 + h_u^2 h_v^2 - h_u^2 h_v^2 \tag{7}$$

$$= 1 + h_u^2 + h_v^2. \tag{8}$$

Finally, the inverse of g can be computed as

$$g^{-1} = \frac{1}{|g|} \begin{bmatrix} 1 + h_v^2 & -h_u h_v \\ -h_u h_v & 1 + h_u^2 \end{bmatrix}. \tag{9}$$

Conveniently, at the center point $h_u(0,0) = h_v(0,0) = 0$ and thus

$$g_{0,0} = g_{0,0}^{-1} = \mathbf{I},\ |g_{0,0}| = 1. \tag{10}$$

To obtain nodes for the fitting of the quadratic polynomial, we project the points from a local neighborhood of p onto the tangent plane. The gradient of a function X on the surface is given as

$$\operatorname{grad} X = \begin{bmatrix} \partial_u \mathbf{\Gamma} & \partial_v \mathbf{\Gamma} \end{bmatrix} g^{-1} \begin{bmatrix} \partial_u X \\ \partial_v X \end{bmatrix}, \tag{11}$$

where $\partial_u = \partial / \partial u$ is a shorthand for partial derivatives. Plugging Equation 10 into Equation 11, we get

$$\operatorname{grad} X = \partial_u X \partial_u \mathbf{\Gamma} + \partial_v X \partial_v \mathbf{\Gamma}. \tag{12}$$

$\partial_u \mathbf{\Gamma}$ and $\partial_v \mathbf{\Gamma}$ are exactly the basis vectors at point p. Thus, the coefficients of the resulting vectors are given by $\partial_u X$ and $\partial_v X$. The function X is given by function values at the points. To estimate the partial derivatives of X at a point p, we locally fit a quadratic polynomial using the same approach as for fitting a quadratic polynomial to the surface and compute its partial derivatives. As for the fitting of the surface patch, we project the points in a local neighborhood to the tangent plane and use the function values as nodes for fitting the quadratic polynomial [Nealen 2004].

Discrete Divergence. The divergence, including the metric components [O'Neill 1983], on the surface patch $\mathbf{\Gamma}$ is

$$\operatorname{div} V = \partial_u V_u + \partial_v V_v + V_u \partial_u \log \sqrt{|g|} + V_v \partial_v \log \sqrt{|g|}, \tag{13}$$

where $|g|$ denotes the determinant of the metric. At the origin, the metric of our surface patch is the identity and the derivatives of the metric at this point vanish. Hence, divergence is given by

$$\operatorname{div} V = \partial_u V_u + \partial_v V_v. \tag{14}$$

To compute the partial derivatives $\partial_u V_u, \partial_v V_v$ at p_i, we require the coefficients of the vector field at neighboring points $\{p_j \mid j \in \mathcal{N}_i\}$. However, different basis vectors are used at different points. Therefore, we need to map from the basis vectors at p_j to those of p_i. While doing so, we account for metric distortion by $\mathbf{\Gamma}$. The following equation requires a bit more notation to distinguish between vectors at different points. We denote the coordinates of p_j in the tangent space of p_i as (u_j, v_j), the metric of $\mathbf{\Gamma}$ at p_j as g_{u_j,v_j}, the coefficients of a tangent vector at p_j as (α_j^u, α_j^v), and the basis vectors at p_j as $\mathbf{e}_j^u, \mathbf{e}_j^v$. The coefficients of a vector at p_j in p_i's parameter domain are

$$g_{u_j,v_j}^{-1} \begin{bmatrix} \partial_u \mathbf{\Gamma}(u_j,v_j) \cdot \mathbf{e}_j^u & \partial_u \mathbf{\Gamma}(u_j,v_j) \cdot \mathbf{e}_j^v \\ \partial_v \mathbf{\Gamma}(u_j,v_j) \cdot \mathbf{e}_j^u & \partial_v \mathbf{\Gamma}(u_j,v_j) \cdot \mathbf{e}_j^v \end{bmatrix} \begin{bmatrix} \alpha_j^u \\ \alpha_j^v \end{bmatrix}. \tag{15}$$

Equation 14 and Equation 15 are combined to form a sparse matrix $\mathbf{D} \in \mathbb{R}^{N \times 2N}$ representing divergence.

Authors' addresses: Ruben Wiersma, r.t.wiersma@tudelft.nl, Delft University of Technology, The Netherlands; Ahmad Nasikun, a.nasikun@tudelft.nl, Delft University of Technology, The Netherlands, and Universitas Gadjah Mada, Indonesia; Elmar Eisemann, e.eisemann@tudelft.nl, Delft University of Technology, The Netherlands; Klaus Hildebrandt, k.a.hildebrandt@tudelft.nl, Delft University of Technology, The Netherlands.

0730-0301/2022/7-ART105
https://doi.org/10.1145/3528223.3530166

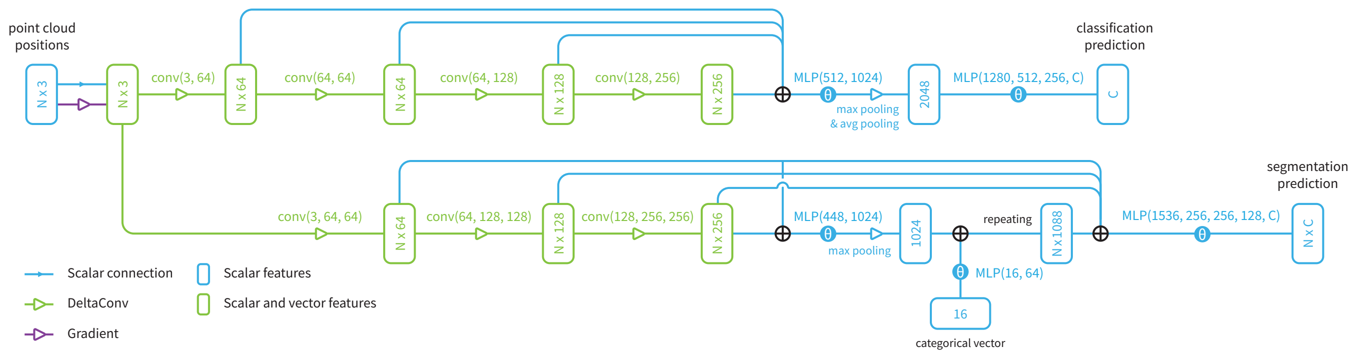

Fig. 1. The two architectures used for classification and segmentation, based on [Wang et al. 2019]. Please refer to Equation 6 and Figure 4 in the main text for the formulation of each convolution and how the streams are combined.

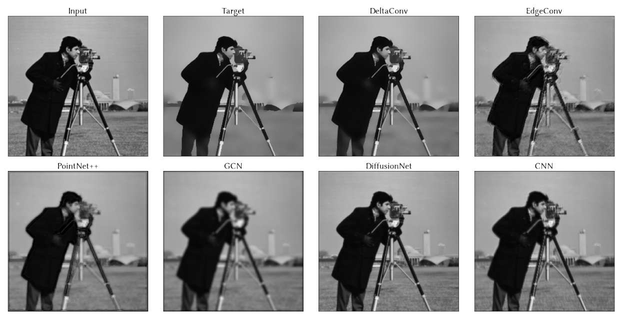

Fig. 2. Comparison of different convolutions optimized to match the result of twenty anisotropic diffusion steps on sample image 'camera'.

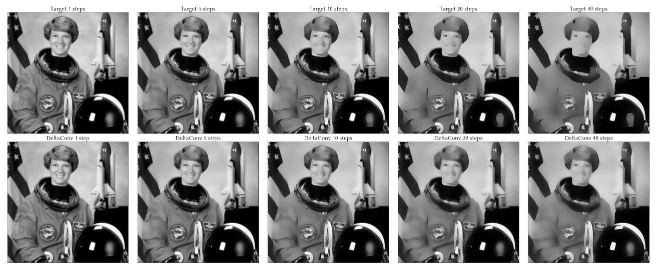

Fig. 3. Comparison of a ResNet with DeltaConv optimized to match the result of varying anisotropic diffusion steps.

2 ARCHITECTURES

We based our architectures on the designs proposed in DGCNN [Wang et al. 2019]. A schematic overview is presented in Figure 1 and more details are provided in the following paragraphs.

Convolutions. Each convolution, denoted as $\textsc{Conv}(C_0, \dots, C_L)$, learns the function h_Θ with an MLP that has L layers. Each layer in the MLP consists of a linear layer with C_i input- and C_{i+1} output channels, batch normalization [Ioffe and Szegedy 2015], and a non-linearity. For scalar features, the non-linearity is a leaky ReLU with slope 0.2 and for vector features a ReLU. We denote MLPs applied per point as $\text{MLP}(C_0, \dots, C_L)$.

Classification network. The classification network has four convolution blocks: $\textsc{Conv}(3, 64)$, $\textsc{Conv}(64, 64)$, $\textsc{Conv}(64, 128)$, $\textsc{Conv}(128, 256)$. Each scalar convolution is interspersed with connections to- and from the vector stream, which mirrors the number of parameters in its vector convolutions. The output of each scalar convolution is concatenated into a feature vector of 512 features and transformed to 1024 features using an MLP. We return a global embedding by taking both the maximum and mean of the features over all points. These are concatenated and fed to a task-specific head: MLP(2048, 512, 256, C), where C is the number of classes in the dataset. This final MLP has dropout [Srivastava et al. 2014] set to 0.5 in between the layers. During training, we optimize a smoothed cross-entropy loss.

Segmentation network. The segmentation network uses three convolutions: $\textsc{Conv}(C_{in}, 64, 64)$, $\textsc{Conv}(64, 128, 128)$, $\textsc{Conv}(128, 256, 256)$. Again, the scalar convolutions are interspersed with connections to- and from the vector stream. The output of each convolution is concatenated into a vector of 448 features per point and transformed to 1024 features with a global MLP. These features are pooled with maximum pooling. This embedding and an embedding of a one-hot encoding of the shape category is concatenated to the output of the convolutions at each point and fed to the task-specific head for segmentation: MLP(1536, 256, 256, 128, C). During training, we optimize a cross-entropy loss.

Table 1. Results on human part segmentation [Maron et al. 2017].

Method	Accuracy
PointNet++ [Qi et al. 2017]	90.8
MDGCNN [Poulenard and Ovsjanikov 2018]	88.6
DGCNN [Wang et al. 2019]	89.7
SNGC [Haim et al. 2019]	91.0
HSN [Wiersma et al. 2020]	91.1
MeshWalker [Lahav and Tal 2020]	**92.7**
CGConv [Yang et al. 2021]	89.9
FC [Mitchel et al. 2021]	92.5
DiffusionNet - xyz [Sharp et al. 2021]	90.6
DiffusionNet - hks [Sharp et al. 2021]	91.7
DeltaNet - xyz	92.2

Table 2. Per-category breakdown of part segmentation results on ShapeNet part dataset. Metric is mIoU(%) on points.

	Mean inst. mIoU	aero	bag	cap	car	chair	ear phone	guitar	knife	lamp	laptop	motor	mug	pistol	rocket	skate board	table
# shapes		2690	76	55	898	3758	69	787	392	1547	451	202	184	283	66	152	5271
PointNet++	85.1	82.4	79.0	87.7	77.3	90.8	71.8	91.0	85.9	83.7	95.3	71.6	94.1	81.3	58.7	76.4	82.6
PointCNN	86.1	84.1	86.5	86.0	80.8	90.6	79.7	92.3	88.4	85.3	96.1	77.2	95.3	84.2	64.2	80.0	83.0
DGCNN	85.2	84.0	83.4	86.7	77.8	90.6	74.7	91.2	87.5	82.8	95.7	66.3	94.9	81.1	63.5	74.5	82.6
KPConv deform	86.4	84.6	86.3	87.2	81.1	91.1	77.8	**92.6**	88.4	82.7	96.2	**78.1**	95.8	85.4	**69.0**	**82.0**	83.6
KPConv rigid	86.2	83.8	86.1	88.2	**81.6**	91.0	80.1	92.1	87.8	82.2	96.2	77.9	95.7	**86.8**	65.3	81.7	83.6
GDANet	86.5	84.2	88.0	**90.6**	80.2	90.7	**82.0**	91.9	88.5	82.7	96.1	75.8	95.7	83.9	62.9	83.1	**84.4**
PointTransformer	86.6	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
PointVoxelTransformer	86.5	85.1	82.8	88.3	81.5	**92.2**	72.5	91.0	88.9	85.6	95.4	76.2	94.7	84.2	65.0	75.3	81.7
CurveNet	86.8	85.1	84.1	89.4	80.8	91.9	75.2	91.8	88.7	**86.3**	96.3	72.8	95.4	82.7	59.8	78.5	84.1
DeltaNet (ours)	86.6	84.9	82.8	89.1	81.3	91.9	79.7	92.2	88.6	85.5	**96.7**	77.2	**95.8**	83.0	61.1	77.5	83.1
Delta-U-ResNet (ours)	**86.9**	**85.3**	**88.1**	88.6	81.4	91.8	78.4	92.0	**89.3**	85.6	96.1	76.4	**95.9**	82.7	65.0	76.6	84.1

U-ResNet architecture The U-ResNet architecture follows the design proposed in KPFCNN [Thomas et al. 2019] (Figure 9 of the supplementary material in [Thomas et al. 2019]). This network consists of an encoder that operates on four scales and a decoder that progressively upsamples the features to the original resolution. In each scale of the encoder, there are two ResNet blocks with a bottleneck. In KPFCNN, the first ResNet block uses strided convolutions, which we replace with pooling followed by a regular ResNet block. Each scale, we subsample to 1/4 points and increase the number of features by two. In the first layer, we use 64 features. We add two additional ResNet blocks with 128 output features after the decoder, as this was shown to be beneficial in CurveNet [Xiang et al. 2021]. We do not use the other changes introduced by CurveNet, such as skip attention in the decoder or squeeze-excitation in the task-specific head. Each convolution block is replaced by a DeltaConv block, which maintains a vector stream in the first three scales and in the final two ResNet blocks. During pooling, scalar features are max-pooled and vector features are averaged with parallel transport to the coordinate system of the sampled point [Wiersma et al. 2020].

3 ADDITIONAL RESULTS AND VISUALIZATIONS

3.1 Visualizations

The anisotropic diffusion experiment was repeated for another input image with 20 anisotropic diffusion steps (Figure 2) and with varying anisotropic diffusion steps (Figure 3), showing that a DeltaConv network can approximate anisotropic diffusion for varying diffusion times.

3.2 ShapeNet

The per-category breakdown of results for ShapeNet are listed in Table 2.

3.3 Human Shape Segmentation

We trained a variant of the simple single-scale DeltaNet (eight layers with each 128 channels) to predict part annotations on the human body dataset proposed by Maron et al. [2017]. This training set is composed of meshes from FAUST (100 shapes) [Bogo et al. 2014], SCAPE (71 shapes) [Anguelov et al. 2005], Adobe Mixamo (41 shapes) [Adobe 2016], and MIT (169 shapes) [Vlasic et al. 2008]. SHREC07 (18 shapes) is used for testing. Each dataset contains human bodies in different styles and poses, e.g., realistic, cartoony, dynamic. We convert the dataset into a point cloud dataset by uniformly sampling $8N$ points from the faces and downsampling these to N points with FPS. We set $N = 1024$, similar to the experiments in Wiersma et al. [2020], $k = 20$ and $\lambda = 0.001$, similar to the other experiments. We normalize the area of the shape before sampling points and augment the input to the network with random rotations around the up-direction, a random scale between 0.8 and 1.25, and a random translation of 0.1 points. The network is optimized with Adam [Kingma and Ba 2015] for 50 epochs with an initial learning rate of 0.01. The results are listed in Table 1. This experiment shows DeltaConv's effectiveness on a deformable shape class and allows us to compare the results to those of other intrinsic (mesh) convolutions. This comparison has its limits, as most of the listed methods are trained on meshes instead of point clouds. Nonetheless, we find that DeltaConv is in line with state-of-the-art approaches, with only raw xyz coordinates as input.

REFERENCES

Adobe. 2016. Adobe Mixamo 3D characters. www.mixamo.com.

Dragomir Anguelov, Praveen Srinivasan, Daphne Koller, Sebastian Thrun, Jim Rodgers, and James Davis. 2005. SCAPE: Shape Completion and Animation of People. *ACM Trans. Graph.* 24, 3 (2005), 408–416.

Federica Bogo, Javier Romero, Matthew Loper, and Michael J. Black. 2014. FAUST: Dataset and evaluation for 3D mesh registration. *CVPR* (2014).

Niv Haim, Nimrod Segol, Heli Ben-Hamu, Haggai Maron, and Yaron Lipman. 2019. Surface Networks via General Covers. *ICCV* (2019).

Sergey Ioffe and Christian Szegedy. 2015. Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift. *ICML* (2015).

Diederik P. Kingma and Jimmy Ba. 2015. Adam: A Method for Stochastic Optimization. In *ICLR*.

Alon Lahav and A. Tal. 2020. MeshWalker: deep mesh understanding by random walks. *ACM Trans. Graph.* 39, 6 (2020), 263:1–263:13.

Jian Liang and Hongkai Zhao. 2013. Solving Partial Differential Equations on Point Clouds. *SIAM J. Sci. Comput.* 35 (2013), A1461–A1486.

Haggai Maron, Meirav Galun, Noam Aigerman, Miri Trope, Nadav Dym, Ersin Yumer, Vladimir G Kim, and Yaron Lipman. 2017. Convolutional neural networks on surfaces via seamless toric covers. *ACM Trans. Graph* 36, 4 (2017), 71:1–71:10.

Thomas W. Mitchel, Vladimir G. Kim, and Michael Kazhdan. 2021. Field Convolutions for Surface CNNs. *ICCV* (2021).

Andrew Nealen. 2004. An As-Short-as-possible introduction to the least squares, weighted least squares and moving least squares methods for scattered data approximation and interpolation. *URL: http://www.nealen.com/projects/mls/asapmls.pdf* 1 (2004).

B. O'Neill. 1983. *Semi-Riemannian Geometry With Applications to Relativity.* Elsevier Science.

Adrien Poulenard and Maks Ovsjanikov. 2018. Multi-directional geodesic neural networks via equivariant convolution. *ACM Trans. Graph.* 37, 6 (2018), 236:1–236:14.

Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. 2017. PointNet++: Deep hierarchical feature learning on point sets in a metric space. *NeurIPS* (2017).

Nicholas Sharp, Souhaib Attaiki, Keenan Crane, and Maks Ovsjanikov. 2021. DiffusionNet: Discretization Agnostic Learning on Surfaces. *ACM Trans. Graph.* 41, 3 (2021), 27:1–27:16.

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *J. Mach. Learn. Res.* 15, 1 (2014), 1929–1958.

Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. 2019. KPConv: Flexible and Deformable Convolution for Point Clouds. *ICCV* (2019).

Daniel Vlasic, Ilya Baran, Wojciech Matusik, and Jovan Popovic. 2008. Articulated Mesh Animation from Multi-View Silhouettes. *ACM Trans. Graph.* 27, 3 (2008), 97.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E. Sarma, Michael M. Bronstein, and Justin M. Solomon. 2019. Dynamic graph CNN for learning on point clouds. *ACM Trans. Graph.* 38, 5 (2019), 146:1–146:12.

Ruben Wiersma, Elmar Eisemann, and Klaus Hildebrandt. 2020. CNNs on surfaces using rotation-equivariant features. *ACM Trans. Graph.* 4 (2020), 92:1–92:12.

Tiange Xiang, Chaoyi Zhang, Yang Song, Jianhui Yu, and Weidong Cai. 2021. Walk in the Cloud: Learning Curves for Point Clouds Shape Analysis. *ICCV* (2021).

Zhangsihao Yang, Or Litany, Tolga Birdal, Srinath Sridhar, and Leonidas Guibas. 2021. Continuous geodesic convolutions for learning on 3d shapes. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision.* 134–144.