Direction Field Extraction through DiffusionNet for Quad Mesh Retopology

Balice Matteo, Doronzo Antonio Giuseppe

ABSTRACT: Current state-of-the-art quadrangulation methods heavily rely on dense direction fields to align parametrizations for quad mesh extraction. In scenarios where user-provided directions are unavailable, these fields are typically interpolated from principal curvature directions, guided by heuristics aiming to identify significant surface regions. However, existing methods often struggle to capture the intricate structures present in meshes crafted by experienced designers who leverage domain knowledge to optimize meshes for specific applications. To address this limitation, we use an approach using diffusion networks for computing direction fields that emulate the structures found in expert-designed meshes. Our method utilizes a diffusion network to generate a direction field, guiding a subsequent parametrization-based quad meshing technique. Unlike traditional methods relying solely on curvature information, our approach incorporates higher-level domain knowledge encoded in existing meshes, whether created by human experts or other sources. Our results showcase a reasonable alignment with the ground truth structure, highlighting the potential of our proposed approach for advancing quad mesh generation in computer graphics applications.

1. INTRODUCTION

Mesh quadrangulation, the process of converting a given triangle mesh into a quadrilateral mesh, is a fundamental problem in computer graphics and geometry processing with applications in character animation and physics simulation. Unfortunately, manually quadrangulating a triangle mesh is both labor-intensive and cumbersome, as it requires the user to manually place individual quads on the surface of the input mesh. Consequently, several authors have proposed fully automatic quadrangulation techniques.

These fully automatic techniques work well on input shapes for which a meaningful alignment of quads can be computed from local curvature information. However, they encounter problems when faced with shapes that do not offer strong curvature guidance. This issue is compounded by the fact that, depending on the application, a designer might prefer an alignment that is not directly related to principal curvature directions. In character modeling and animation, for instance, artists usually place additional edge loops around surface regions likely to deform, such as the eyes or mouth of a human character. Furthermore, irregular vertices are often placed in approximately planar regions to hide visual artifacts. For numerical simulation on quad meshes, designers align and specify different sizes of quads based on their expert knowledge of how simulation solvers behave. It is unclear how local curvature information relates to the expert knowledge in these various domains. To address this issue, several authors have proposed methods that incorporate user guidance into the remeshing process.

2. Related works

2.1. Quad Meshing Methods

The computer-aided generation of quad meshes has been a well-researched topic in recent years. Methods range from fully automatic pipelines to interactive approaches, where users manually specify most of the quad mesh geometry and connectivity.

Field-guided quad meshing algorithms, such as Huang et al. (2018), Jakob et al. (2015), are of particular interest because they produce high-quality results by dividing the process into two steps. First, a guiding field is generated, specifying the positions and degrees of irregular vertices, as well as the desired orientation and size of the resulting quad elements. Second, a parametrization is computed that aligns its gradients with the specified directions of the guiding field, with integer iso-lines defining the edges of the resulting quad mesh. The quality of the results heavily depends on the guiding field, and many methods have been proposed for its generation. Common among these methods is that the desired alignment is derived from the surface geometry, primarily its curvature. This approach works well for shapes where alignment to principal curvature is sufficient. For cases where different alignment is desired, user input can be incorporated during field generation.

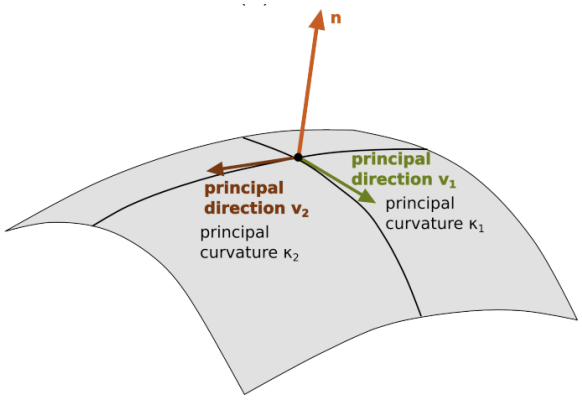

Fig. 1: Principal curvatures and principal directions.

Other interactive quad meshing methods require the user to partition the surface into patches, which are then filled with suitable quad grids. While these methods offer detailed and explicit control over the resulting mesh, they also demand extensive user input, thereby increasing the overall time required to generate quad meshes.

2.2. Data-driven Remeshing

Data-driven remeshing refers to the process of using data from existing mesh structures to inform and guide the remeshing of new shapes. This approach leverages patterns, features, and domain knowledge embedded in previously generated meshes to improve the quality and efficiency of the remeshing process. By analyzing and learning from these existing datasets, data-driven methods aim to produce more accurate and application-specific remeshes, reducing the need for manual intervention and expert knowledge.

A similar work of our project, which has inspired our work, is Dielen et al. (2021): their approach use a neural network to aggregate global and local shape information, computing direction fields that better mimic expert designs. This method produces more accurate and application-specific quad meshes, combining the strengths of data-driven and model-driven techniques.

3. Differential Geometry Background

Consider a smooth oriented surface $\mathbf{S}$ and denote k_{max} and k_{min} its maximal and minimal principal curvatures, $k_{max} \geq k_{min}$. Let t_{max} and t_{min} be the corresponding principal directions. Denote by e_{max} and e_{min} the derivatives of the principal curvatures along their corresponding curvatures directions:

$$e_{max} = \frac{\partial k_{max}}{\partial t_{max}}, e_{min} = \frac{\partial k_{min}}{\partial t_{min}}$$

Given a mesh $\mathbf{M}$ approximating a smooth surface $\mathbf{S}$, in order to achieve a fast and accurate estimation of the principal curvatures and their derivatives a bivariate polynomial is fitted locally to each mesh vertex. In our work, we used the method proposed in Yoshizawa et al. (2005) to derive principal curvatures.

For each mesh vertex $p \in \mathbf{M}$, its one-link neighborhood is considered and a new vertex p' is obtained as the arithmetic mean of the centroids of the mesh triangles adjacent to p. These new vertices p' form a new mesh $\mathbf{M}'$ which is smoother than $\mathbf{M}$. After that, for each vertex $\mathbf{p}' \in \mathbf{M}'$ we calculate its unit normal. Then, for each vertex $\mathbf{p}' \in \mathbf{M}'$, a set of its neighboring vertices is obtained from the k-link neighborhood, paying attention to removing those vertices whose normals make obtuse angles with the normal at $\mathbf{p}'$.

Finally, a cubic polynomial $h(x, y)$ is fitted with least-square approach to $\mathbf{p}'$ and its neighboring vertices.

$$h(x,y) = \frac{1}{2}(b_0x^2 + 2b_1xy + b_2y^2) +$$

$$\frac{1}{6}(c_0x^3 + 3c_1x^2y + 3c_2xy^2 + c_3y^3)$$

That set of neighbors of $\mathbf{p}'$ is obtained from the k-link neighborhood of $\mathbf{p}'$ by removing those vertices whose normals make obtuse angles with the normal at $\mathbf{p}'$. Next the curvature tensor and extremality coefficients e are expressed via derivatives of local cubic polynomial h(x, y). Finally these curvature attributes are assigned to the original vertices $\mathbf{p}$ of mesh $\mathbf{M}$.

4. Dataset

The dataset used in this project is based on the Skinned Multi-Person Linear Model (SMPL). More information about SMPL can be found at `https://smpl.is.tue.mpg.de/index.html`.

The dataset contains three characters animated. We decided to divide it into the training set and test set in this way: the training set consists of two characters for a total of 933 frames. The test set consists of 1 character for 230 frames.

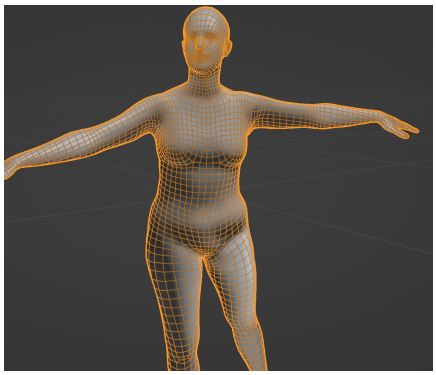

Fig. 2: Sample topology of the dataset.

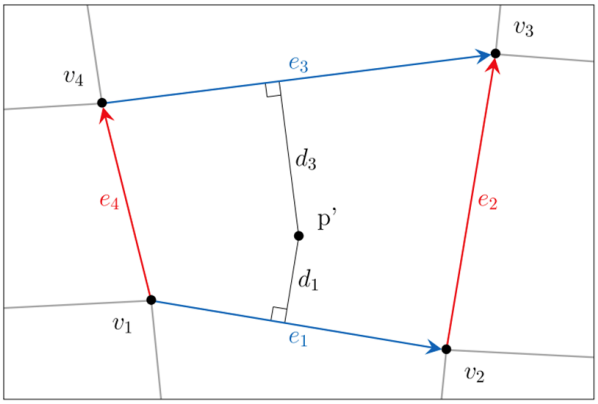

Fig. 3: Shortest distance d_1, d_3

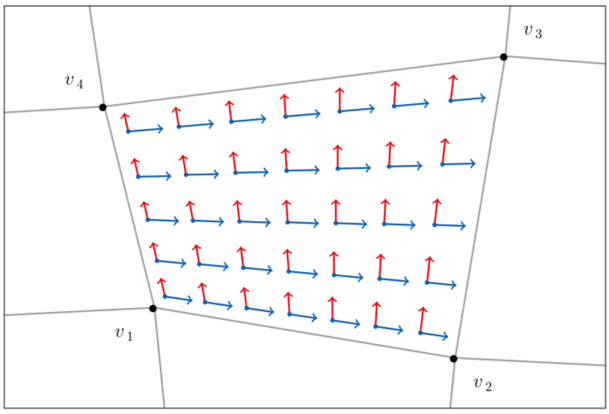

Fig. 4: Ground truth cross field.

4.1. Extracting Ground Truth Cross Fields

Since our dataset contains quad meshes, it's straightforward to extract the ground truth cross fields for each vertex of the triangle mesh.

For each vertex p of the triangle mesh:

1. First, we select the closest face of the quad mesh to p;
2. Second, we project p onto p' (p' belongs to the quad mesh);
3. Then, we compute the shortest distances d_1, d_2, d_3, and d_4 (fig. 3);
4. We compute the two vectors **u** and **v** (cross fields) as follows:

$$\mathbf{u} = \frac{d_3}{d_1 + d_3}\mathbf{e}_1 + \frac{d_1}{d_1 + d_3}\mathbf{e}_3,$$

$$\mathbf{v} = \frac{d_4}{d_2 + d_4}\mathbf{e}_2 + \frac{d_2}{d_2 + d_4}\mathbf{e}_4$$

5. Convert **u** and **v** to spherical coordinates.

The spherical coordinates are calculated as follows:

$$r = 1$$

$$\theta = \arccos\left(\frac{z}{r}\right) = \arccos(z)$$

$$\phi = \arctan 2(y, x)$$

We convert to spherical coordinates because we are interested only in the direction of the vectors, not their specific positions. This allows us to simplify the problem by fixing the radius to a constant value (1). With a fixed radius, we only need to predict two angles (theta and phi) to define the direction, instead of three coordinates (x, y, and z) required in Cartesian coordinates. This reduces the number of outputs we need to predict from six to four (theta and phi for both the vectors).

5. Proposed Method

Given an unstructured triangle mesh T, our goal is to generate a quad mesh Q that not only represents the same shape as T, but also exhibits the structure found in meshes created by domain experts. Directly outputting such a quad mesh is a non-trivial task that would require the network to generate a valid mesh topology, which is difficult due to global consistency requirements. As a consequence, we propose using a neural network to compute a direction field that can be fed to an existing field-guided parametrization-based quadrangulation technique. This approach allows us to control the structure of the resulting mesh Q using a direction field that can be more easily inferred by a network operating on the input mesh T. For the purpose of this project we do only rely on outputting direction fields from the neural network without generating the quad mesh.

5.1. Extracting information from mesh

First, given a triangulated mesh, we use the C++ library provided by Yoshizawa et al. (2005) to extract some information for each vertex such as $k_1, k_2, ks_1, ks_2, t_1, t_2$, and the normal (see Section 3 for an explanation about these values). These pieces of information associated with each vertex are the input of the neural network, along with the vertices, edges, and faces of the mesh.

5.2. Neural Network

The neural network we used is called DiffusionNet Sharp et al. (2020). DiffusionNet introduced relies on three core components: multilayer perceptrons (MLPs) that model scalar functions based on features at each point, a tailored diffusion process for information transmission across the space, and local spatial gradient features that enhance the network's filter space, moving beyond merely radially symmetric filters. A detailed explanation of these numerical

elements underpins the construction of a highly effective architecture.
Since this network learns functions on surfaces, making the spatial diffusion along them the main operation, we can say that considering V vertices and a collection of D scalar features defined at each vertex, the basic building block is a pointwise function $f : \mathbb{R}^D \to \mathbb{R}^D$, applied independently at every vertex to transform the features. These pointwise functions are represented as MLPs.
In order to propagate features on surfaces like meshes or point clouds is used the heat equation $\frac{d}{dt}u_t = \Delta u_t$. The diffusion process, represented by $H_t(u_0) = \exp(t\Delta)u_0$, evolves an initial distribution over time to an average across the surface. The Laplacian Δ is approximated by a matrix L for computational efficiency. Features are diffused at varying times to achieve both local and global distribution. This approach replaces traditional convolutions in neural networks, optimizing diffusion as a network parameter and maintaining the expressive power of convolutional networks.
Local spatial gradient enhances the representation of three-dimensional surfaces by computing gradients at the vertices of a mesh or point cloud, using scalar transformations. These gradients, calculated as $z_u = Gu$, are then used to generate robust new features through inner products, resulting invariant to rotations. For an even richer representation, these gradients are transformed using a learned complex matrix, A. The process involves calculating the scalar product of transformed gradients, followed by the use of the real part of this complex product, which is then modulated by a nonlinear tanh function, as expressed in the formula $g_u = \tanh(\mathrm{Re}(w_0 A w_y))$. This approach not only captures finer details of the surface but also adjusts to the direction of rotation, providing a powerful and orientation-sensitive geometric representation.

5.3. Loss Function

The loss function we initially used was designed to measure the discrepancy between the model's predictions and the target values for the angular variables θ and ϕ.

It consists of two components:

- Loss for θ: This is calculated using the squared sine of the difference between the predicted and target θ values:

$$\mathrm{Loss}_\theta = \sin^2(\theta_{\mathrm{pred}} - \theta_{\mathrm{target}})$$

  We used the sine as loss function to account for the periodic nature of the angle θ, ensuring that the maximum error is when we have a 90° angular difference, because θ is periodic every π.

- Loss for ϕ: Similarly, this is calculated using the squared sine of half the difference between the predicted and target ϕ values:

$$\mathrm{Loss}_\phi = \sin^2\left(\frac{\phi_{\mathrm{pred}} - \phi_{\mathrm{target}}}{2}\right)$$

  Dividing the angular difference by 2 ensures that the maximum error is when we have a 180° angular difference, because ϕ is periodic every 2π.

The total loss is the sum of these two components:

$$\mathrm{Loss} = \mathrm{Loss}_\theta + \mathrm{Loss}_\phi$$

Since the neural network must predict two different vectors (cross fields), it does not know which one to compare with the ground truth labels. For example: the predictions of the neural network are two vectors u_1 and v_1, while the two true vectors are u_2 and v_2. The network does not know if to compare u_1 with u_2 (so as to minimize this error) or with v_2.

For this reason, we combine two different predictions and targets to ensure robustness:

- Prediction of u_1 with u_2 and v_1 with v_2.
- Prediction of u_1 with v_2 and v_1 with v_1.

The minimum sum of these combinations is taken as the final loss. This approach ensures that the model considers the best possible match between predictions and targets, enhancing the model's robustness to angular prediction errors.

However, this loss function, although it allowed the network to learn successfully, did not give total satisfactory results, so it was decided to opt for another one.
This new function is the cosine similarity of the predicted and target vectors.

$$\mathrm{Loss} = 1 - \frac{|u \cdot v|}{||u||||v||}$$

The cosine similarity square was used for training, while the absolute value was used for testing. In fact, the loss must always be positive because we only want the direction of the cross fields.

6. Results

To firstly compare the result of the two different loss function, we plotted an example of the ground truth direction field and the predicted direction field. For the first loss function the result is that it seems there is no correlation between the real and the predicted direction field, as we can notice in Fig. 6.

However, in the plot of the second loss function (Fig. 7), the one we effectively used, the predicted direction field is quite similar to the real one. This shows that the cosine similarity loss function is more

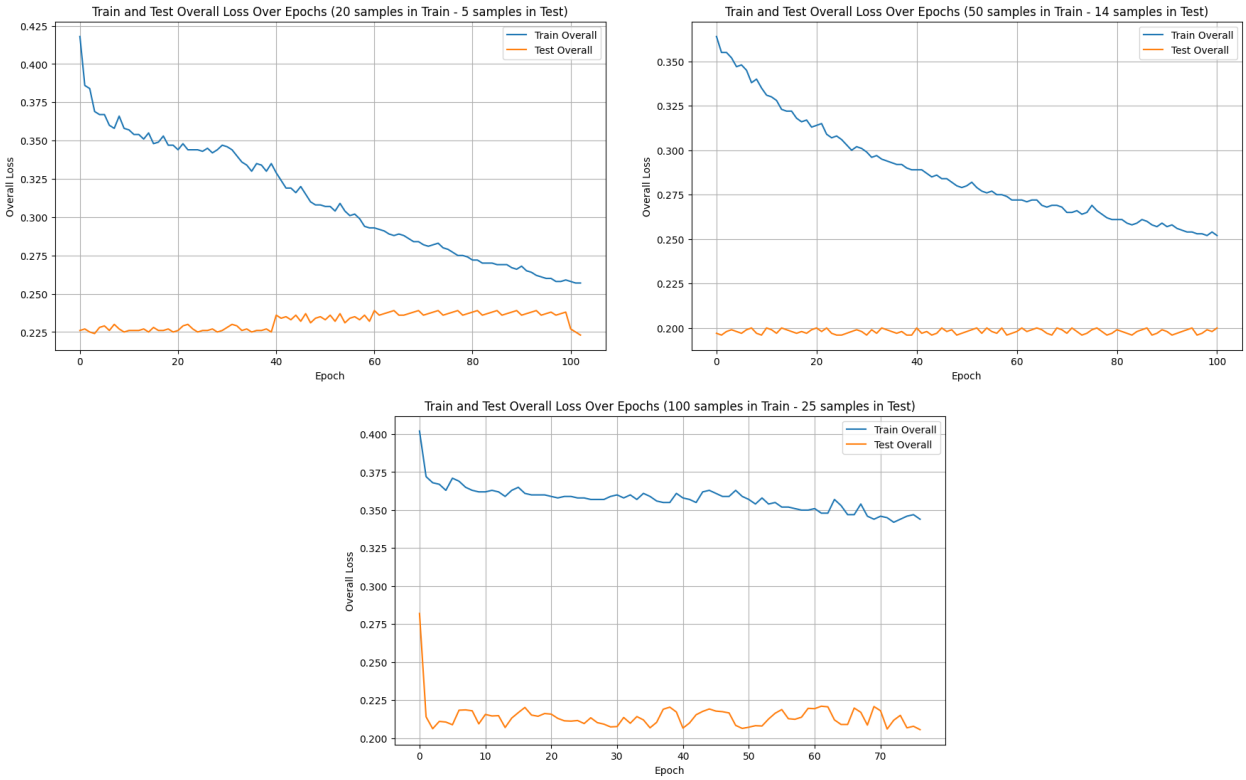

Fig. 5: Losses of second loss function.

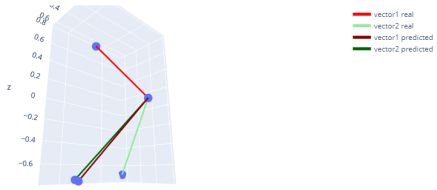

Fig. 6: Direction fields of the first loss function. It seems that the real and predicted vectors are not correlated one to each other.

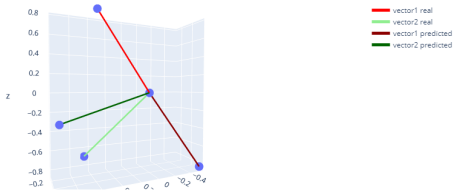

Fig. 7: Direction fields of the second loss function. In this case, we can notice that the predicted direction fields are quite similar to the real ones.

suitable for our task.

The values got after the predictions respectively represent the discrepancy between the model's predictions and the target values for the angular variables θ and ϕ during the training and test phases.
To derive the total error, the calculation is as follows:

$$\text{Error}_{\text{test}} = arcos(1 - Loss_{test})$$

$$\text{Degrees} = \frac{\text{Error}_{\text{test}} \cdot 180}{\pi}$$

In Fig. 5 there are some comparisons with different sizes of the dataset. This shows us that dataset size is very important and using more samples is very important in order to get better results.

All these results are summarised in Table 1, which compares results of the two loss functions and shows that the error got using the most number of samples is the lower.

From the first loss function, we ended up with a total error of 47.76°, which is not entirely bad, but using the second one there was definitely a marked improvement.

While our method currently exhibits a higher mean error in quad remeshing compared to other techniques (table 2), the primary reason we chose diffusionNet is its generative capabilities. Models like

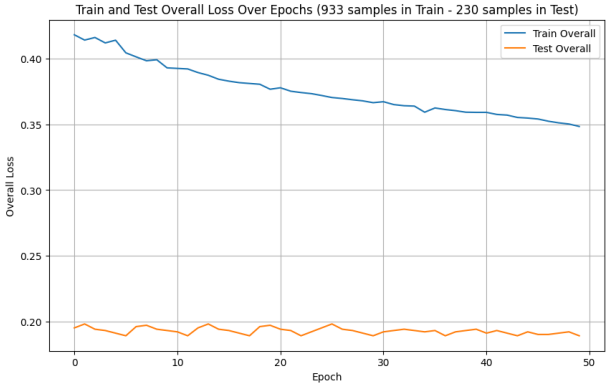

Fig. 8: Loss of the second loss function for the complete dataset.

Samp.	1° Err. (Rel).	2° Err. (Rel.)
20 tr. - 5 ts.	49.31° (0.348)	39.01° (0.223)
50 tr. - 14 ts.	49.06° (0.344)	37.53° (0.207)
100 tr. - 25 ts.	48.96° (0.343)	37.40° (0.206)
933 tr. - 230 ts.	47.76° (0.327)	**35.49° (0.189)**

Table 1: Table representing test results with training with different dataset sizes.

diffusionNet are designed to generate new data points from learned distributions, similar to how DALL-E 2 generates high-quality images from text descriptions (read future works section). We believe that the error can be lowered even more by addressing data scarcity or/and changing some settings in the architecture of diffusionNet.

7. Conclusion

In this project, we investigated the use of diffusion networks to compute direction fields for quad mesh retopology. Our goal was to develop a method that could capture the intricate structures found in meshes crafted by experienced designers. This approach leverages domain knowledge encoded in existing meshes, allowing the network to generate direction fields that better mimic expert designs.

We achieved a test error (angular error) of approximately 35° in predicting the angular variables for the direction field. While this indicates moderate accuracy, there's certainly room for improvement. We are sure that this is not the best result we can achieve because the test loss decreases as we increase the number of examples in the training set, but due to the lack of training examples and also to computational resources we can not fully exploit the potential of our model.

Method	Mean Error
Dielen et al. (2021)	5.727°
Du et al. (2015)	$\approx 11°$
Ours	35.49°

Table 2: Brief comparison with others quad remeshing techniques.

8. Future works

In the realm of quad mesh generation, the desired mesh topology often varies depending on the specific application. While our diffusion network approach effectively captures the intricate structures found in expert-designed meshes, it may not fully address the need for user-controlled topology adjustments (for instance the same mesh could have a different topology if we are in the context of character animation or in physics simulation). To address this limitation, we suggest integrating a transformer-based architecture into our method. Transformers have demonstrated remarkable capabilities in natural language processing tasks, particularly in capturing long-range dependencies within text sequences. This ability aligns well with the task of incorporating user-provided topology information, as it often involves understanding relationships between mesh elements across the entire structure. By incorporating a transformer-based module, we can provide a mechanism for users to specify their desired mesh topology. The user could input a text description or a high-level sketch of the desired topology, and the transformer would then extract and translate this information into a form compatible with our diffusion network.

REFERENCES

Dielen, A., Lim, I., Lyon, M., and Kobbelt, L. 2021, Computer Graphics Forum, 40, 181

Du, P., Zhou, J., and Tu, C. 2015, , 11, 6861

Huang, J., Zhou, Y., Nießner, M., Shewchuk, J., and Guibas, L. 2018, in

Jakob, W., Panozzo, D., and Sorkine-Hornung, O. 2015, ACM Transactions on Graphics, 34, 1

Sharp, N., Attaiki, S., Crane, K., and Ovsjanikov, M. 2020, CoRR, abs/2012.00888

Yoshizawa, S., Belyaev, A., and Seidel, H.-P. 2005, 227–232