A Generalized Differentiable Evaluation Plug-in for Loop Subdivision in Surface Reconstruction Pipelines

Tianhao Xie
Brian A. Barsky
Sudhir Mudur
Tiberiu Popa

Electrical Engineering and Computer Sciences
University of California, Berkeley

Technical Report No. UCB/EECS-2024-180
http://www2.eecs.berkeley.edu/Pubs/TechRpts/2024/EECS-2024-180.html

August 15, 2024

A Generalized Differentiable Evaluation Plug-in for Loop Subdivision in Surface Reconstruction Pipelines

Tianhao Xie*
Concordia University

Brian Barsky†
University of California, Berkeley

Sudhir Mudur‡
Concordia University

Tiberiu Popa§
Concordia University

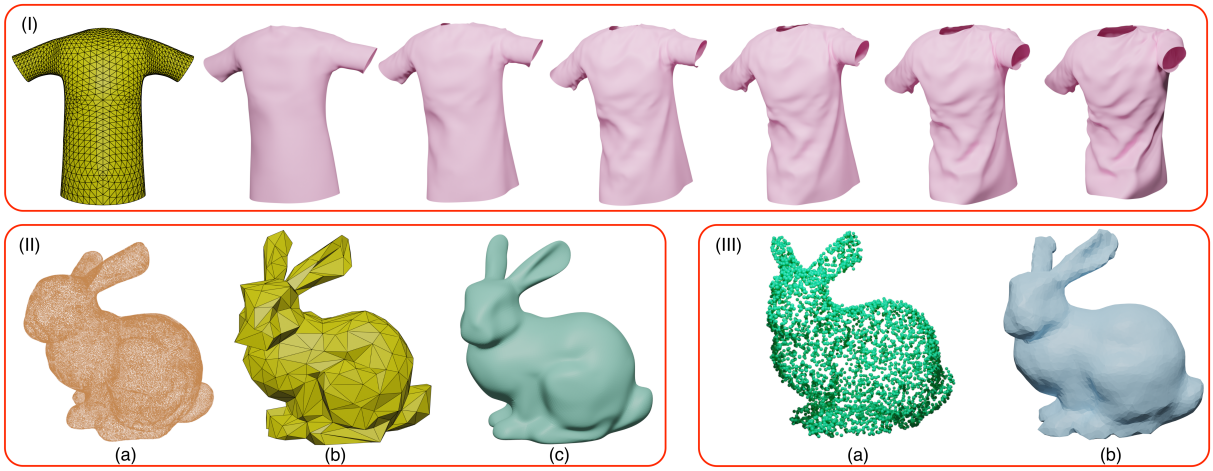

Figure 1: Loop subdivision evaluation plug-in in different applications. (I) Fitting subdivision surface to spatial-temporal sequences. (II) Fitting subdivision surface to the static point cloud. (a) Point cloud. (b) Fitted control mesh. (c) Subdivision surface of (b). (III) Integrating the plug-in into Deep Marching Tetrahedra [46]. (a) Point cloud. (b) Fitted subdivision surface.

Abstract

Continuity in surface representations is extremely important for many design, analysis, simulation and visualization tasks in aero, hydro, automotive and graphics industries. While piecewise continuous NURBS have been ubiquitous, handling topologically complex surfaces can be cumbersome. Hence, linear piecewise polygonal/triangular meshes have been increasingly dominant. Today, these are reconstructed by suitably trained deep learning networks usually from multi-view images or point clouds, however, these meshes are not smooth along the edges and at vertices. In this paper, we present a powerful differentiable surface fitting method which can be integrated into surface reconstruction pipelines. We use the Loop subdivision surface, which in the limit yields the smooth surface underlying the point cloud, and can also handle complex surface topology. The principal idea is to stage the Loop subdivision scheme such that it enables generalized analytical evaluation on any triangulation, i.e., without any constraints on vertex valences. Importantly, this in turn enables us to formulate the subdivision process as an unconstrained minimization problem of a differentiable function which can be solved with standard numerical solvers. The other contribution is the use of the Implicit Moving Least Squares (IMLS) surface fitting as an energy loss to add shape-awareness to the commonly used Chamfer loss for improving output quality. We demonstrate our plug-in in multiple contexts such as smooth surface reconstruction from the point cloud using classical Poisson reconstruction, or in an end-to-end deep neural network pipeline such as deep marching tetrahedra. We further apply our method to spatial-temporal reconstruction through a differentiable renderer. We have both a CPU as well as a fast GPU implementation of our technique that can be easily plugged into any deep-learning pipeline for point clouds or meshes. The code will be made publicly available.

*e-mail: tianhao.xie@mail.concordia.ca
†e-mail: barsky@berkeley.edu
‡e-mail: mudur@cse.concordia.ca
§e-mail: tiberiu.popa@concordia.ca

1 Introduction

Surface reconstruction is a fundamental problem in 3D digital geometric processing and as such has received a lot of attention over the years. A large subset of such methods focus on mesh reconstruction from point clouds or, more recently, from images using recent advances in differential rendering [43]. While polygonal meshes are the most popular surface representations and are used in many contexts, they are piece-wise linear representations which are not smooth along the edges and at vertices. Therefore, for many problems such as fluid flow [50] cloth simulation [38] or coupled shape optimization [60], polygonal meshes lead to discontinuous derivatives that pose major challenges to the underlying optimization problem.

A popular solution is to replace triangular meshes with either parametric analytical surfaces such as NURBS [2, 41] etc. or C^1 subdivision schemes [9] such as Loop [31] or Catmul-Clark [5]. Fitting a complex of NURBS patches with C^1 continuity everywhere to topologically complex shapes remains a challenging task. Subdivision surfaces are a popular alternative. Subdivision surfaces are particularly appealing to many high-level applications such as surface optimization and analysis, simulation, modeling, and animation [9]: not only they are very compact, they do not require explicit sub patch decomposition and alignment as NURBS do [45]. Thus subdivision surfaces are ideal to use for fitting more complex surface topology. A subdivision surface is represented by a compact

polygonal mesh which gets subdivided by introducing new vertices, using, for example, Loop subdivision formulation [31]; in the limit, this subdivision process leads to a smooth shape. An additional advantage is also the compact representation of the surface needing only a relatively small number of control variables compared to a triangular mesh - this lower dimensionality coupled with the inherent smoothness both facilitates the convergence and acts as an implicit regularizer in shape optimization problems [37].

Fitting subdivision surfaces to point clouds has many challenges. Existing fitting methods [7, 11, 33] rely on an optimization function that uses iterative point-to-point correspondences. This optimization function is non-differentiable, is not robust to noise and outliers, and also tends to fail if the initial guess of the control mesh is too far from the solution. The differentiability of our subdivision fitting formulation is a key feature that allows its integration in end-to-end deep learning pipelines such as Deep Marching Tetrahedra [46], and also its integration with differentiable renderers.

Among existing subdivision schemes, Loop subdivision [31] checks nearly all boxes: it is a triangular scheme that can approximate any shape independent of topology, it is C^1 everywhere, and under some constraints on vertex valances, it has a differentiable analytical formulation for its limit surface [49]. More specifically, the constraint is that no two adjacent vertices are irregular (i.e. degree other than 6). This condition makes it difficult to obtain an appropriate control mesh. A user might be required to do some local connectivity editing to achieve that. But, in the context of an end-to-end network based pipeline such as DMT [46], the connectivity of the mesh outputted by the DMT pipeline that becomes an input to our Loop evaluation is constantly changing in an unpredictable way with every iteration of the optimization, therefore the Loop evaluation must be robust enough to work on any and all possible input meshes.

Our main objective in this work is to overcome this limitation and define a more versatile Loop evaluation scheme that is fully differentiable and can be easily integrated into existing 3D mesh reconstruction pipelines providing greater access to subdivision models for CAD, simulation, and optimization simulation applications. To that end, we make the following contributions:

1. We stage the Loop subdivision evaluation in a way that removes the valence constraints, enabling analytic evaluation of the limit surface for any control mesh, and resulting in a versatile minimization formulation of an unconstrained and differentiable function on any general control mesh.

2. We showcase the virtues of an unconstrained differentiable optimization formulation on static point clouds as well as reconstructing compatible spatial-temporal sequences (i.e. same triangulation from frame to frame) using a differentiable renderer.

3. We integrate the Loop formulation with the deep marching tetrahedra pipeline, a state-of-the-art end-to-end deep learning pipeline.

4. Importantly, we integrate the shape-aware IMLS loss [38] to avoid artifacts due to the shape-agnostic nature of the Chamfer loss.

2 Related Work

The subdivision process defines a smooth curve or surface as the limit of a sequence of mesh refinement steps starting from a control mesh. This makes the final surface controlled by a small number of control vertices in the starting mesh, thus resulting in a very compact surface representation. Several subdivision schemes have been developed over the years and are widely used in different applications [5, 9, 10, 30, 31]. In particular, Loop subdivision is a subdivision scheme based on quartic box spline on triangular meshes [31]. It is guaranteed that, in the limit, the subdivision surface has C^2 continuity in regular vertices (degree 6) and C^1 continuity in irregular vertices. In 1998, Jos Stam developed an analytical evaluation method of Loop subdivision [49], which was based on a conversion from Box splines to B-Nets [24]. This analytical and differentiable evaluation makes this scheme ideal for differentiable shape optimization and we will use it in our novel subdivision fitting pipeline.

2.1 Fitting subdivision surface to target shape

It is a common task to fit a smooth surface representation to a target shape in computer graphics. One typical solution for this task is to fit a piecewise smooth surface to the target, such as a B-spline surface or a subdivision surface. Considerable work has been done on fitting B-splines to point clouds by squared distance minimization [58, 61]. Since our focus in this work is on fitting subdivision surfaces, we will limit our discussion of related work primarily to subdivision surface fitting.

Hoppe *et al.* [15] and Lavoue *et al.* [26] fit subdivision surfaces to CAD models by minimizing the squared distance energy. Litke *et al.* [28] used quasi-interpolation to fit the Catmull-Clark subdivision surface to a given shape within a prescribed tolerance. Ma *et al.* [32] described a method to fit a Loop subdivision surface to a dense triangular mesh by linear least square fitting.

The geometric data captured in the wild is almost always in the form of an unstructured point cloud, with noise, outliers, and missing geometry. A large body of work has focused on fitting subdivision surfaces to point cloud data [7, 11, 33, 34]. Cheng *et al.* [7] fit the subdivision surface by iteratively minimizing a quadratic approximant of the squared distance function of a target shape. Their approach first samples points on the Loop subdivision surface based on a method by Stam [49]. Then, they solve a linear system of the control mesh variables to minimize the squared distance between the sample points and the target shape. Marinov *et al.* [33] introduced an algorithm based on exact closest point search on Loop surfaces which combines Newton iteration and non-linear minimization. In more recent research, Esteller *et al.* [11] used second-order approximation of the squared distance function and the tangent space alignment to achieve robust fitting of the subdivision surface for shape analysis. Similar to methods in [7] and [33], Esteller *et al.* also sampled the points on the subdivision surface to establish the error function – error between the subdivision surface and the target shape. These methods need to solve a sequence of constrained least-squares problems to minimize the error function. The method in [16] could be optimized by the gradient-descent method. However, instead of fitting the limit surface, they could only fit a specific level of subdivision surface to the target shape. In contrast to many of these methods, our proposed solution frames the fitting problem as an optimization of a completely differentiable function that can be solved using standard differentiable optimization methods.

Some learning-based methods to fit a surface to a target shape have also been previously proposed. Most of these approaches fit parametric polynomial surfaces of some form to point clouds. Yumer and Kara used a neural network to generate NURBS from input point sets [59]. DeepFit incorporated a neural network to learn point-wise weights for weighted least squares polynomial surface fitting [3]. Sharma *et al.* described a method using neural networks to fit B-spline patches to input point cloud data [45]. Our fitting method is not deep learning based; however, being differentiable, it can be used to bridge the gap between deep learning-based methods in a 3D domain and traditional subdivision surface techniques.

2.2 Spatio-temporal surface reconstruction

Reconstructing representations for time-varying 3D data is a common problem in graphics animation and simulation. A common

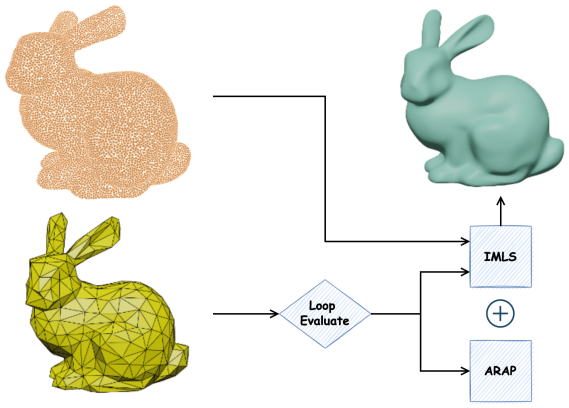

Figure 2: Overview of our method for fitting a subdivision surface to a static point cloud.

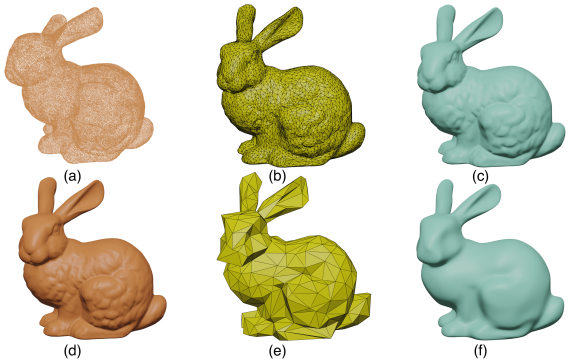

Figure 3: Stanford Bunny [51]:(a) Point cloud with 72,027 vertices. (b) Optimized control mesh with 4667 vertices. (c) Subdivision surface of (b). (d) Screened Poisson reconstructed mesh with 155,008 vertices. (e) Optimized control mesh with 314 vertices. (f) Subdivision surface of (e).

approach is to fit a template mesh to the consecutive time-series point cloud or mesh. This is used to reconstruct coherent dynamic geometry from time-varying point clouds captured by real-time 3D scanning techniques. One widely used method is to reconstruct meshes for all frames first and then to fit a template mesh to all reconstructed meshes [1, 18, 52, 53]. These methods always need additional markers or landmarks which must be specified by the users. Another method is to generate a template from the first frame and then fit the template directly to the remaining frames [47, 54]. In [54], Sussmuth *et al.* followed the Multi-level Partition of Unity (MPU) Implicits approach to reconstruct the implicit function that approximates the time-varying surface defined by the time-varying point cloud and used the As-Rigid-As-Possible constraint to the moving of the points. When compared to our method, 1) their method does not fit a subdivision surface to the 4D data and thus the final resulting surface is not smooth; 2) while we use distance field energy, they used an implicit function to represent the point cloud surface, which must be optimized by solving a sequence of least squares problems.

A few other methods perform template-free reconstruction [36, 42, 44]. In [36], Mitra *et al.* directly computes the motion of the scanned object in all frames and estimates the time-deforming object by kinematic properties. In [44], Sharf *et al.* used a space-time solid incompressible flow prior to the reconstruction of moving and deforming objects from point data. In [42] a template is constructed gradually by mapping consecutive frames in a pyramidal fashion. In [57], Wand *et al.* they reconstructed 3D scanner data by pairwise scanning alignment. Tevs *et al.* [55] introduced Animation Cartography, an intrinsic reconstruction of shape and motion, based on robust estimation of dense correspondences under topological noise and landmark tracking in temporally coherent and incoherent data. In addition, there are also some real-time reconstruction methods for general objects [27, 39, 62].

2.3 Neural subdivision

In [29], a learning-based framework for coarse-to-fine geometry modeling, called Neural Subdivision was introduced. This method shows better ability to preserve geometry features compared to the traditional subdivision method. The method presented in [46] used Deep subdivision to refine the final mesh, but this data-driven subdivision scheme does not guarantee smoothness and has no closed-form solution for the limit surface.

3 Method Overview

Subdivision surfaces are constructed via an iterative process from an initial control mesh $M^0(V^0,F^0)$. In every iteration a new mesh is created by moving the existing vertices using a weighted combination of the neighbors using a pre-defined weight mask and by adding new vertices typically along edges also positioned using a weighted combination of the nearby vertices. In the end we obtain a sequence of meshes $M^0(V^0,F^0), M^1(V^1,F^1), M^2(V^2,F^2)$ whose positions converge to a limit surface. While such a construction is simple to implement and intuitive to use for a modeling application, which was one of the design goals of subdivision surfaces, it does not have in general an analytical expression for the limit surface, which is required for optimization problems and exact reconstruction. However, for the Loop subdivision scheme, Stam [49] derived an analytical formula for the limit surface. His method has one caveat: it can be applied for meshes that do not have adjacent irregular vertices (i.e. vertices that have a degree other than 6). One observation is that by applying one iteration of subdivision to the Loop scheme, $M^1(V^1,F^1)$ it is easy to see that for closed meshes no adjacent vertices are irregular as new regular vertices are created in the middle of all edges. However, this new control mesh will have 4 times as many faces thus increasing significantly the size of the mesh. A second option is to extend Stam's derivation to include all possible combinations, but this will result in a combinatorially impractical large number of cases to handle, making it error-prone, and very difficult to implement efficiently, especially on a GPU in a way that it can take advantage of the hardware parallelism.

We address this challenge by staging the Loop evaluation into two stages and propose a practical and elegant divide-and-conquer approach to address this challenge. We model the evaluation of the limit surface as a function of the control mesh as a composition of two simpler differentiable functions. The first one takes as input the control mesh and outputs another mesh corresponding to one level of subdivision yielding a mesh satisfying the required vertex valence constraints for analytic evaluation in the limit, and the second one applies Stam's solution to compute the limit surface. We can now obtain the evaluation of an arbitrary point on any general control mesh by evaluating these two functions, and we can extract the derivatives needed for solvers using the chain rule. This method retains the low dimensionality of the starting control mesh making it practical for use in optimization frameworks and end-to-end neural networks pipelines but removes entirely any limitation related to vertex valences in the control mesh. More specifically, given a point $\hat{Q}$ on the control mesh M^0, in order to compute its position on the final smooth 3D surface we first compute its position $\tilde{Q}$ on M^1 by

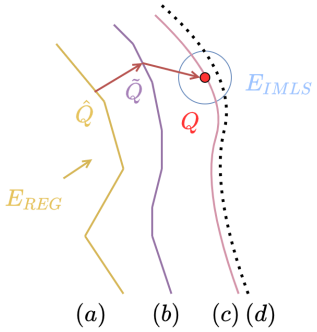

Figure 4: A schematic view of our optimization process. (a) Control mesh (M^0). (b) Control mesh after one level of subdivision (M^1). The vertices of this mesh are the Loop subdivision control points. (c) Loop subdivision surface. (d) Target point cloud. We optimize for M^0 by using an IMLS fitted to the point cloud and an ARAP regularizer on the control mesh M^0.

using the Loop subdivision mask [31]. Then after adjusting the triangle index and getting new barycentric coordinates we compute the position on the limit surface as per Stam [49]. This operator $L(\cdot)$ that maps the point $\hat{Q}$ on the control mesh to the point Q onto the final subdivision surface is both analytical and differentiable and only depends on the original vertices of M^0. We provide both a CPU and a GPU implementation of this general evaluation scheme and we integrate it in multiple surface reconstruction contexts: (i) directly fitting a subdivision surface to a point cloud, (ii) fitting a subdivision surface to point cloud integrating it in the end-to-end Deep Marching Tetrahedra network, and (iii) fitting a spatial-temporal sequence of subdivision surfaces using a differentiable renderer.

4 Subdivision Surface Fitting

4.1 Fitting a static point cloud

Given a point cloud $P = \{P_i\}$ with associated normals $N = \{N_i\}$, our goal is to compute the control mesh $M^0(V^0, F^0)$ of a Loop subdivision surface that best fits the point cloud. The first step in our process is to create the control mesh for the Loop subdivision surface $M^0(V^0, F^0)$. Although the position of the control mesh vertices will be determined by our optimization, the number of vertices as well as the topology of this mesh must be determined a priori. For this, we compute an initial triangular mesh that fits the point cloud using existing meshing methods; we use Screened Poisson [22] method in MeshLab [8]. We then simplify this triangulation using quadratic edge collapse [12] until we obtain the desired number of vertices requested by the user.

Next, we fit our template control mesh $M^0(V^0, F^0)$ using the following optimization:

$$\min_{V^0} E_{dist}(L(M^0, \hat{Q})) + \alpha \cdot E_{reg}(M^0, \bar{M}^0) \tag{1}$$

where $E_{dist}(\cdot)$ is the IMLS fit energy [40] (eq. 4), $L(\cdot)$ is the 3D position on the subdivision surface of a set of points $\hat{Q}$ sampled from the control mesh, $\bar{M}_0$ is the undeformed control mesh, $E_{Reg}(\cdot)$ is the ARAP regularizer [48] (eq. 5), and α is the weight of the regularizer term. An overview of the fitting model is shown in Figure 2.

IMLS fit energy: Oztireli *et al.* introduced an Implicit Moving Least Squares(IMLS) surface in [40], which gave us a definition for the point cloud surface as:

$$f(x) = \frac{\sum n_i^T (x - x_i)\phi_i(x)}{\sum \phi_i(x)} \tag{2}$$

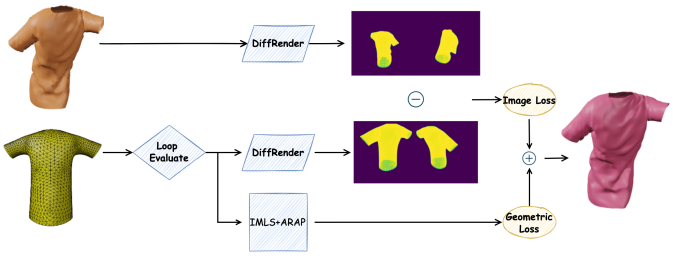

Figure 5: Overview of fitting subdivision surfaces to a spatial-temporal sequence by combining Implicit Moving Least Squares (IMLS) with differential rendering (DR) optimization

where ϕ is a locally supported kernel function that vanishes beyond the cut-off distance h. h is the radius we search for neighbor points and needs to be manually selected.

$$\phi(r) = (1 - \frac{r^2}{h^2})^4, \tag{3}$$

We can use the implicit surface definition in equation 2 to derive a fit energy [38]

$$E_{dist} = \sum_i (\frac{\sum_k N_k^T (Q_i - P_k)\phi(\|Q_i - P_k\|)}{\sum_k \phi(\|Q_i - P_k\|)})^2, \tag{4}$$

where P_k and N_k are the 3D positions and normals of points in the input point cloud (Figure 4d) and Q_i are points on the subdivision surface sampled from the control mesh (Figure 4a-c). For simplicity, in all our examples we only use the vertices of the control mesh, but we analyze the pros and cons of using more sampled points in the following sections and illustrate this in Figure 10.

Regularizer: We experimented with several reguralizers and the As-Rigid-As-Possible (ARAP) regularizer [48] yields the best results. The ARAP regularizer does not penalize any isometric deformations allowing local rotations, but it penalizes local stretches. More specifically:

$$E_{reg} = \sum_i \sum_{j \in N(i)} w_{ij} \|(V_i^0 - V_j^0) - R_i(\bar{V}_i^0 - \bar{V}_j^0)\|^2, \tag{5}$$

where $N(i)$ is the set of vertices adjacent to V_i^0, $\bar{V}_i^0$ is the initial vertex position and R_i is the local estimation rotation matrix for the one ring of vertices around vertex i. w_{ij} is the standard cotangent Laplacian weight [35]. At every iteration, R_i can be computed analytically using SVD decomposition on the local co-variance matrix [56].

Optimization: Unlike previous methods, ours is formulated as an unconstrained optimization problem of a differentiable analytical function that can be solved efficiently using standard off-the-shelf numerical methods.

4.2 Fitting a spatial-temporal model

For the spatial-temporal case, we fit the subdivision surface defined by the control mesh iteratively to the temporally changing sequence of shapes, using the solution from one frame as an initial guess for the subsequent frame. Although this approach is popular and widely used [34], it often fails due to accumulated drift arising from the inherently local nature of the geometric distance. Consequently, additional information is used to correct it, usually either in the form of boundary constraints [11] or other visual queues such as optical flow [4, 42]. Recently, with the development of differentiable

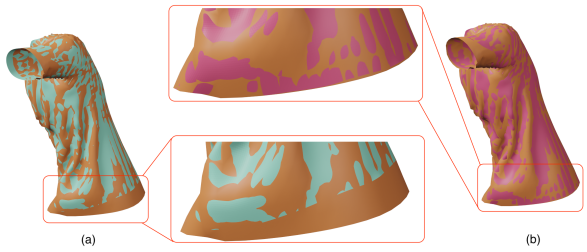

Figure 6: Comparison between the result of T-shirt data fitting. The brown color is the target. Green (a) using the geometric IMLS fit. Red (b) combines the geometric IMLS fit together with the image loss from the differential renderer. Note the drift in (a) at the bottom of the T-Shirt that is resolved in (b)

renderers, rendered image difference metrics can be used to optimize shape [19]. Adding the image difference loss from the differential renderer complements our pipeline, adding a global structure to our local geometric fit thus eliminating the drift and yielding a more accurate fit.

The input to our pipeline is a temporal sequence of target shapes S^i in the form of a set of triangular meshes. These meshes can be computed independently from point clouds as they do not require the triangular meshes to have the same connectivity. For the spatial-temporal case, we employ meshes as target shapes instead of directly using point clouds only because there are currently no available reliable differentiable renderers for point clouds and we want to take advantage of differentiable rendering since we want to combine it with our differentiable fitting to reconstruct spatial-temporal surfaces. But we would like to emphasize here that our method poses no conceptual limitations for using point clouds even for the spatial-temporal case. The output of our method is a subdivision surface defined by a control mesh M^0. For the spatial-temporal fitting, the vertices of M^0 will have different 3D positions in each frame.

Differentiable rendering: The emergence of differentiable rendering (DR) [25, 43] paved the way for a new set of tools in 2D to 3D surface reconstruction. It allows 3D shape optimization and modeling from rendered 2D images [19–21, 43]. In image space, DR-based optimization can give us global loss energy when fitting to a mesh, which is complementary to our local geometric IMLS loss. Inspired by this, we introduce a new pipeline for fitting subdivision surfaces to spatial-temporal (4D) mesh data by combining our method with DR.

Optimization: Similar to the static case, given a control mesh M^0 and a sequence of spatial-temporal target mesh S^i, we are sequentially fitting the control mesh to each target mesh, using the solution of the current frame as an initial guess for the next one. Optimizing using only the geometric energy functionals described above leads to temporal drift as can be seen in Figure 6 (a). To overcome this we add an image loss term that provides a global stabilization of the optimization, eliminating the drift as can be seen in Figure 6 (b).

In every iteration's forward pass, we use the DR to render the target mesh in different camera positions k which gives us target images I^k_{TARGET} At the same time, we use the same DR to render the limit surface of the template mesh which gives us predicted images I^k_{PRED} in the same camera positions as used for rendering the target images. Suppose the number of pixels for a rendered image is N, we compute image loss l_{image} by

$$l_{image} = \sum_k \frac{(I^k_{PRED} - I^k_{TARGET})^2}{N}. \quad (6)$$

As for the geometric loss, we compute it by the same method for computing energy provided in Section 3. Thus, the total loss l_{total} is

$$l_{total} = E_{dist}(L(M^0), \hat{Q}) + \alpha \cdot E_{reg}(M^0, \bar{M^0}) + \beta \cdot l_{image}. \quad (7)$$

In our implementation, we use the DR available in PyTorch3D [43] and for the backward pass, we use the gradient descent method, Adam optimizer [23], to optimize the control mesh.

4.3 Integrating Loop subdivision in end-to-end surface reconstruction networks

More recently, end-to-end reconstruction networks have been proposed for triangular mesh reconstruction from point clouds or voxels such as Deep Marching Tetrahedra (DMT) [46]. In [46], a multilayer perceptron was used to predict the signed distance function (SDF) and deformation of every tetrahedral grid vertex. Then, by using differentiable marching tetrahedra, a triangular mesh can be reconstructed based on the predicted SDF and deformed tetrahedra grid. In [46], after the reconstruction of the mesh, a graph convolutional network and deep subdivision were used to refine the reconstructed mesh. Since the code for the full pipeline in [46] was only partially released, we use only the parts shown in Figure 7 from (a) to (e).

Our Loop subdivision plug-in fits naturally into the DMT pipeline as shown in Figure 7 and the network can be trained end to end, inclusive of our Loop plug-in module. We experimented with both Chamfer Loss and IMLS loss. Chamfer loss is very robust to the initial location of the reconstructed mesh. It also does not require normals to be computed on the point which allows for more sparsity in the input point cloud, but it does not encode any shape information about the surface. The IMLS loss, on the other hand, takes into account the shape but it requires a fairly close initial guess and works better if the point cloud is dense. We reconcile these two competing losses by employing a strategy where we first train 5000 iterations using Chamfer loss only and then train for extra 1000 iterations on the entire pipeline using both Chamfer and IMLS loss. For the last 1000 iterations we train on the entire pipeline Figure 7 (a)-(g)). For the first 5000 iterations we experimented both by training on the entire pipeline and by training only on DMT (Figure 7 (a)-(e)). The difference in performance is not significant. Since the control mesh can change at every iteration, in the initial stages of the training it is possible to have occasional occurrence of large degree vertices. In the GPU implementation we have to set a static value for the highest valence that can occur, and the processing of high valence vertices is slower as explained in section 5.4. Due to the nature of the GPU parallelism, all data follows the same control path resulting in a significant overall performance penalty even when very few vertices have large valence.

5 Results and Discussion

5.1 Static surface fitting

We used our method to fit a number of synthetic models (the Stanford Bunny and Lucy), see Figures 3 and 8 as well as a point cloud of our own acquired using the Microsoft Azure Kinect device (a Koala toy) Figure 9. The starting searching radius h_0 and weight of ARAP regularizer α were selected manually. We scaled the point clouds to a unit box before fitting to increase the numerical stability of the optimization. For the Stanford Bunny and Lucy, we used $h_0 = 0.0005$. For the toy Koala, we used $h_0 = 0.05$. As for the α, it depends on the noise level of the point cloud. When the point cloud is noisy, you need a bigger weight, such as 0.1. When the point cloud is very clean, α should be set to very small, such as 0.01. For the Stanford Bunny and Lucy, we set $\alpha = 0.01$. For the toy Koala, we set $\alpha = 0.1$.

For the bunny(Figure 3) the original point cloud has 72,027 vertices and the reconstructed mesh using Screened Poisson [22] has 155,008 vertices. We demonstrate two reconstructions. The first

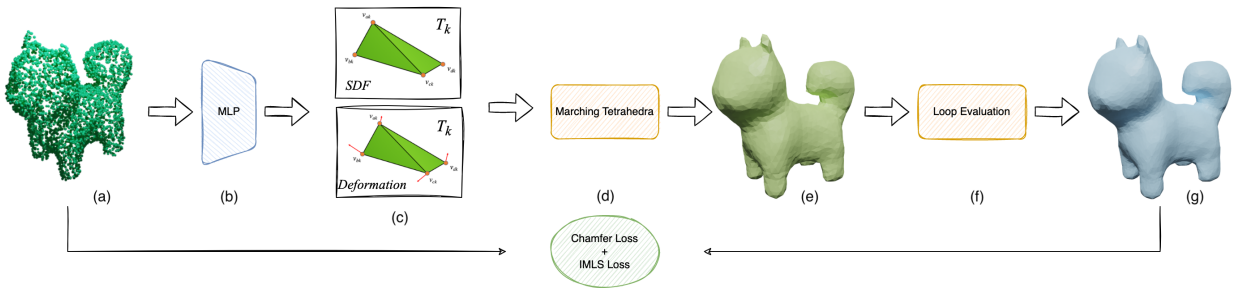

Figure 7: Integrating Loop subdivision evaluation plug-in into DMT. (a) Input point cloud. (b) Multi-layer perceptron. (c) Predicted SDF and deformations for every tetrahedra grid vertex by MLP. (d) Differentiable marching tetrahedra. (e) Reconstructed mesh by DMT. (f) Loop subdivision evaluation plug-in. (g) Fitted Loop subdivision surface.

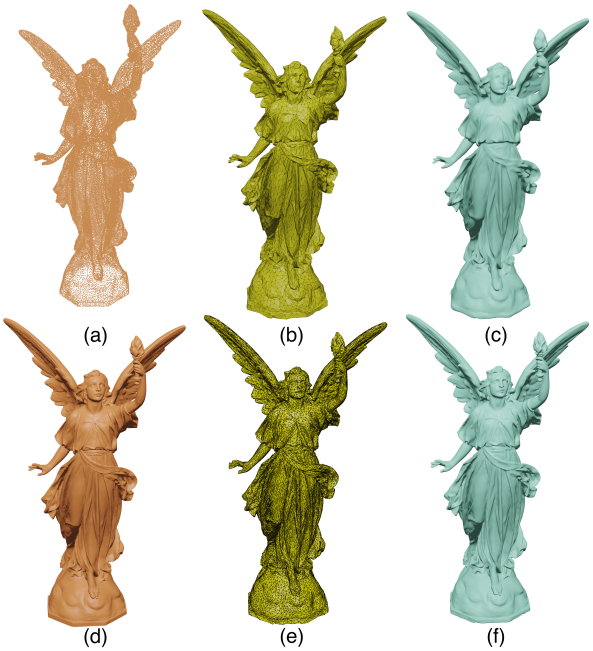

Figure 8: Stanford Lucy [51]:(a) Point cloud with 49,987 vertices. (b) Optimized control mesh with 8002 vertices. (c) Subdivision surface of (b). (d) Screened Poisson reconstructed mesh with 262,909 vertices. (e) Optimized control mesh with 20,002 vertices. (f) Subdivision surface of (e).

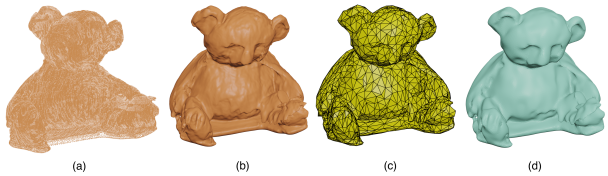

Figure 9: Kinect scanned koala:(a) Point cloud with 1,018,126 vertices. (b) Screened Poisson reconstructed mesh with 276,529 vertices. (c) Optimized control mesh with 2,465 vertices. (d) Subdivision surface of (c).

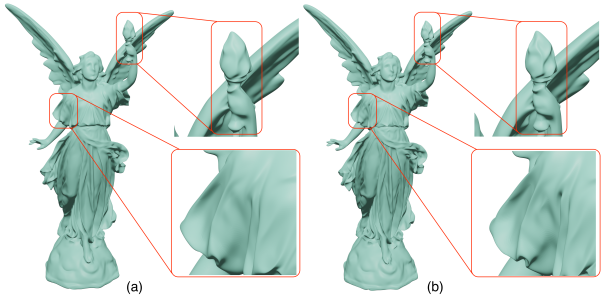

Figure 10: Comparison between (a) using only the control mesh vertices to compute the IMLS fit, and (b) using the vertices after one level of subdivision.

one with a template mesh of 4667 vertices (Figure 3 (c)) that shows no visual difference to the original, but uses only around 3% of the Screened Poisson reconstruction. The second one uses only 314 vertices, or only 0.2% of the Screened Poisson reconstruction (Figure 3 (f)). While a number of details are absent due to very high mesh compression, the main shape is still reconstructed fairly well.

For the more detailed and complicated Lucy model (Figure 8), with only 3% of the Screened Poisson reconstruction vertices, we could retain most of the intricate objects and folds.

In Figure 9 we show the reconstruction of a koala toy. The physical scanned model is furry so while the original reconstruction is very detailed it also contained a lot of noise. With only 0.2% of the original number of vertices and 0.8% of Screened Poisson reconstruction vertices, we provide a reconstruction that retains the shape and many of the important details.

The performance of the IMLS distance depends on the number of sampled points on the subdivision surface that we use in the computation. By default, in all our examples we only use the points in the control mesh. However, it is possible to select more samples. Figure 10 shows this trade-off. Figure 10 (a) is the reconstruction of the Lucy model using only the vertices in the original control mesh. Figure 10 (b) is the reconstruction using the vertices obtained after one level of subdivision (i.e. four times more). The result is slightly improved, some areas contain more detail, however optimization takes about three times as long.

5.2 Spatial-temporal fitting

We tested our spatial-temporal method on two sequences: a synthetic sequence generated using a cloth simulation of a T-Shirt in Blender [14], and a spatial-temporal capture of a cow toy using a multi-view stereo setup. Both sequences have 30 frames and in both cases we made a template from the first frame. For the cloth sequence, we used for simulation a mesh of 2000 vertices that we randomly re-sampled in every frame to simulate a real capture to $100,000$ vertices (or 2% of the total vertices). The template mesh has 2046 vertices, For the

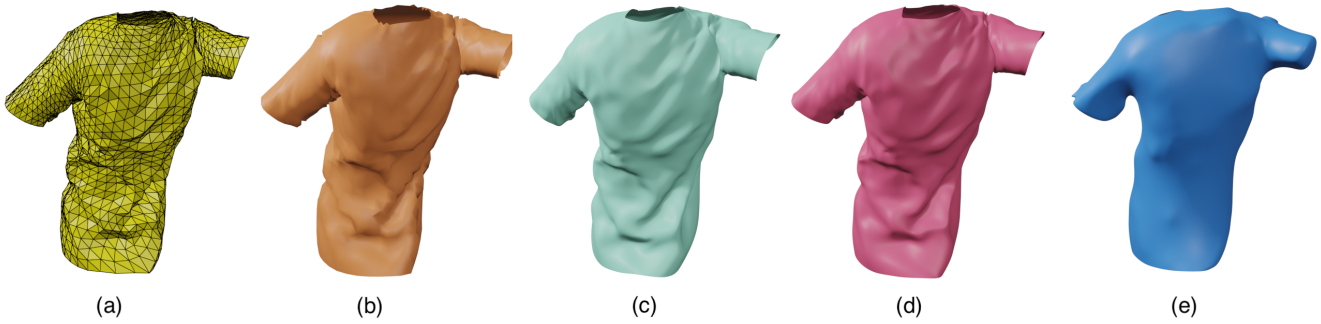

Figure 11: Fitting result for t-shirt simulation: (a) Optimized control mesh of using both IMLS and DR. (b) Simulation result from Blender [14]. (c) Fitting result by only IMLS energy(section 3). (d) Fitting result by combining IMLS and DR(section 4.2). (e) Fitting result by only DR.

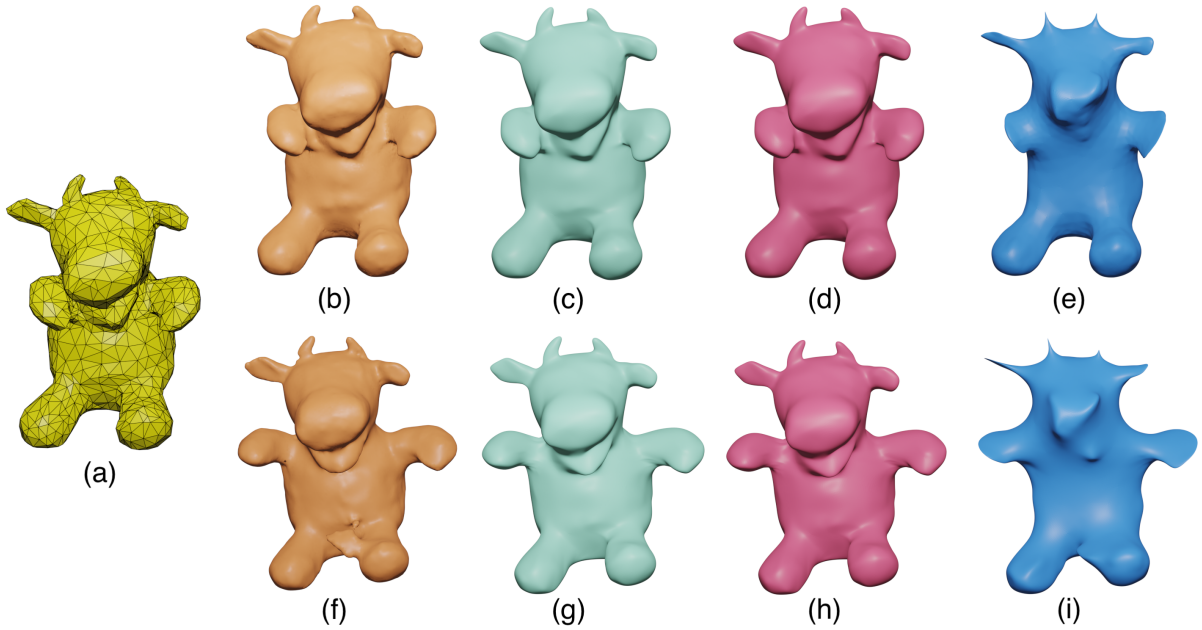

Figure 12: Fitting result for a real scanned puppet. (a) Template control mesh with 1,252 vertices. (b), (f) Reconstructed per frame meshes using [22] (c), (g) Fitted subdivision surface using IMLS energy (section 3) (d), (h) Fitted subdivision surface using a combination of IMLS energy and DR(section 4.2) (e), (i) The fitting result only using DR. The shapes (b)-(e) correspond to the initial frame. The shapes (f)-(i) correspond to the last frame in the sequence.

puppet sequence, the target mesh has around $123,000$ vertices and the template mesh of 1252 vertices (1% of the total vertices).

The settings for the DR are adapted from the PyTorch3D [43] tutorial. We used Soft Silhouette shader whose image size is 256×256, blur radius is $log(1/(1e^{-4}-1)*1e^{-4})$ and faces per pixel is 100. When rendering the target shape, we had 20 different camera views in total. However, in every iteration, we only randomly select 2 views to render the images of the template to reduce unnecessary rendering time. The hyper-parameters α and β were set to 0.1 and 1. The T-Shirt sequence has a lot of geometric details that are well preserved in the reconstruction. In contrast, the puppet sequence has less detail and in some cases, some reconstruction artifacts (see Figure 12 (f)) stay fixed in the reconstruction due to the continuity properties of the subdivision surfaces. In Figures 11 and 12 we compare the IMLS fitting scheme with the DR fitting scheme. Using the DR fitting scheme by itself results in loss of a lot of details: Figures 11 (e), 12 (e), (i) This is not unexpected as we only use silhouette loss. However, the geometric detail between IMLS and IMLS+DR is very similar (Figures 11 (c), (d), Figures 12 (c), (d), Figures 12 (g), (h)). The main gain from adding the DR term is the reduced drift (Figure 6). We also perform a quantitative evaluation using the Hausdorff distance between the target mesh and the subdivision surface. For the subdivision surface, we computed the Hausdorff distance using 3 iterations of subdivision. Results are presented in Figure 15. The combination of IMLS + DR largely outperforms either of them used separately.

5.3 Integrating Loop subdivision evaluation plug-in into Deep Marching Tetrahedra

In evaluating our method we use a methodology similar to [46]. We selected 12 meshes from TurboSquid website[1], ShapeNet [6], and Stanford scanning repository [51] and sampled 5000 points on every mesh with added Gaussian noise $\mu = 0$, $\sigma^2 = 0.005$, as shown in the top of figure 13. The reconstructed meshes by algorithm 7 are shown at the bottom of figure 13.

[1] https://www.turbosquid.com/

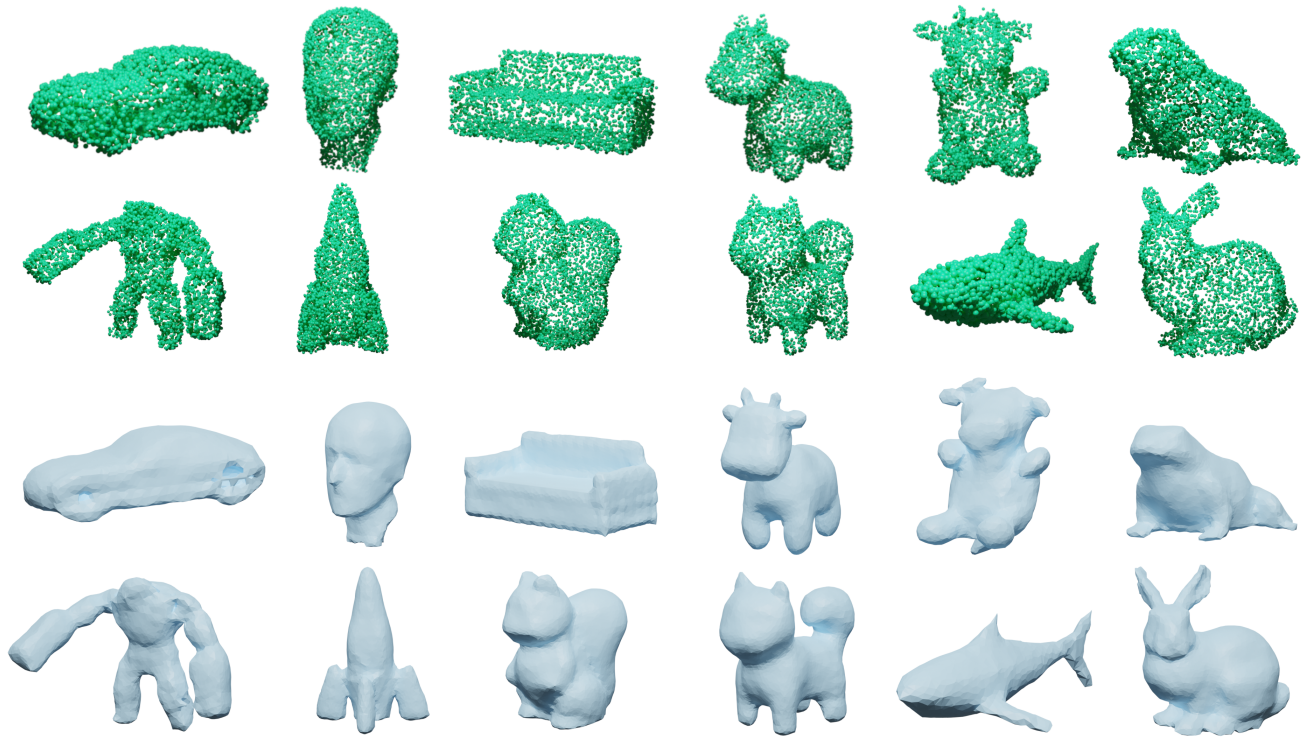

Figure 13: Top: Sampled point clouds. Bottom: Reconstructed meshes using DMT and our Loop subdivision evaluation.

Chamfer distance, Hausdorff distance, and IMLS functions are usually computed between point clouds. Since our output is a subdivision surface, we evaluate these functions by using the limit position of the control mesh vertices of the subdivision surface. We report the Hausdorff distance and Chamfer distance for the result of extra 1000 iterations training by (i) Chamfer loss only (Figure 16 DMT), (ii) Loop evaluation adding Chamfer loss (Figure 16 DMT+Loop) and (iii) using loop evaluation, Chamfer loss, and IMLS loss (Figure 16 DMT+Loop+IMLS). Our subdivision surface has similar Chamfer and Hausdorff distances compared to DMT but our model has the extra benefit of being C^1 continuous everywhere. In addition to the closeness of the fit, we also evaluate the mesh quality of the subdivision mesh, an important feature in some applications. We measure the per-face triangular quality using the function in MeshLab [8] as follows: (1) $\frac{area\,of\,the\,face}{max\,side\,of\,the\,triangle}$ (2) $\frac{radii\,of\,incircle}{circumcircle}$ (3) $\frac{area}{a*a+b*b+c*c}$. Figure 17 shows that the Loop limit surface has better-shaped triangles than the mesh from DMT for similarly sized meshes. Figure 18 illustrates this visually with an example.

As per the quantitative results obtained, fitting the subdivision surface by Chamfer loss has the smallest distance to the point cloud. However, because Chamfer distance measures the distance using a point-to-point function, it can sometimes over-fit to noisy input; as can be seen in Figure 14 (a), there is a sharp artifact between the arm and the cheek of the puppet. Since the IMLS loss is computed based on local implicit surfaces of the point cloud, it can avoid or mitigate this kind of over-fitting as can be seen in figure 14 (b). What's more, since the IMLS energy is more geometry-aware, the result produced by IMLS has better visual smoothness as shown in Figure 14 (e) and (f). Combining the loop evaluation, IMLS loss, and Chamfer loss yields the best visual result among all.

5.4 Implementation

We implemented our Loop subdivision evaluation plug-in as a PyTorch extension so that it can be easily used in any PyTorch-based deep-learning pipeline. We implemented both the C++ extension and the CUDA extension to enable our plug-in to work on both CPU and GPU. The C++ extension is based on Eigen [13] and Libigl [17].

To implement the CUDA extension, there are two main problems that needed to be solved: how to handle the list structure properly to store the adjacency list of a triangular mesh which is required by the evaluation, and how to parallelize the computations so that the kernel function can be executed efficiently on the GPU. Since the adjacency list is not usable in CUDA, we replace it with a padded matrix that has the number of its rows as the number of vertices and the number of columns as the maximum number of valences, which we have set to 12 in our implementation.

Since most of the Deep-learning pipelines are executed on the GPU, the CUDA extension also saves time for transferring data between the CPU and GPU. We report our execution times in Figure 19. The CUDA implementation is up to 6.6x faster on the Loop subdivision evaluation and up to 9.4x faster on the Loop subdivision evaluation using IMLS energy.

6 Conclusion, Limitations and Future work

In this work, we have presented a generalized differentiable evaluation plug-in for Loop subdivision. Differentiability is achieved by suitably staging the loop subdivision process to enable defining the derivative as a composite. It works with any triangular control mesh and hence can be easily integrated into deep learning-based pipelines for different applications. We demonstrate our method on subdivision surface fitting for static point clouds, subdivision surface fitting for spatial-temporal shape sequences using a differentiable renderer, and integration of the plug-in into the Deep Marching Tetrahedra pipeline. The results show that our plug-in is versatile

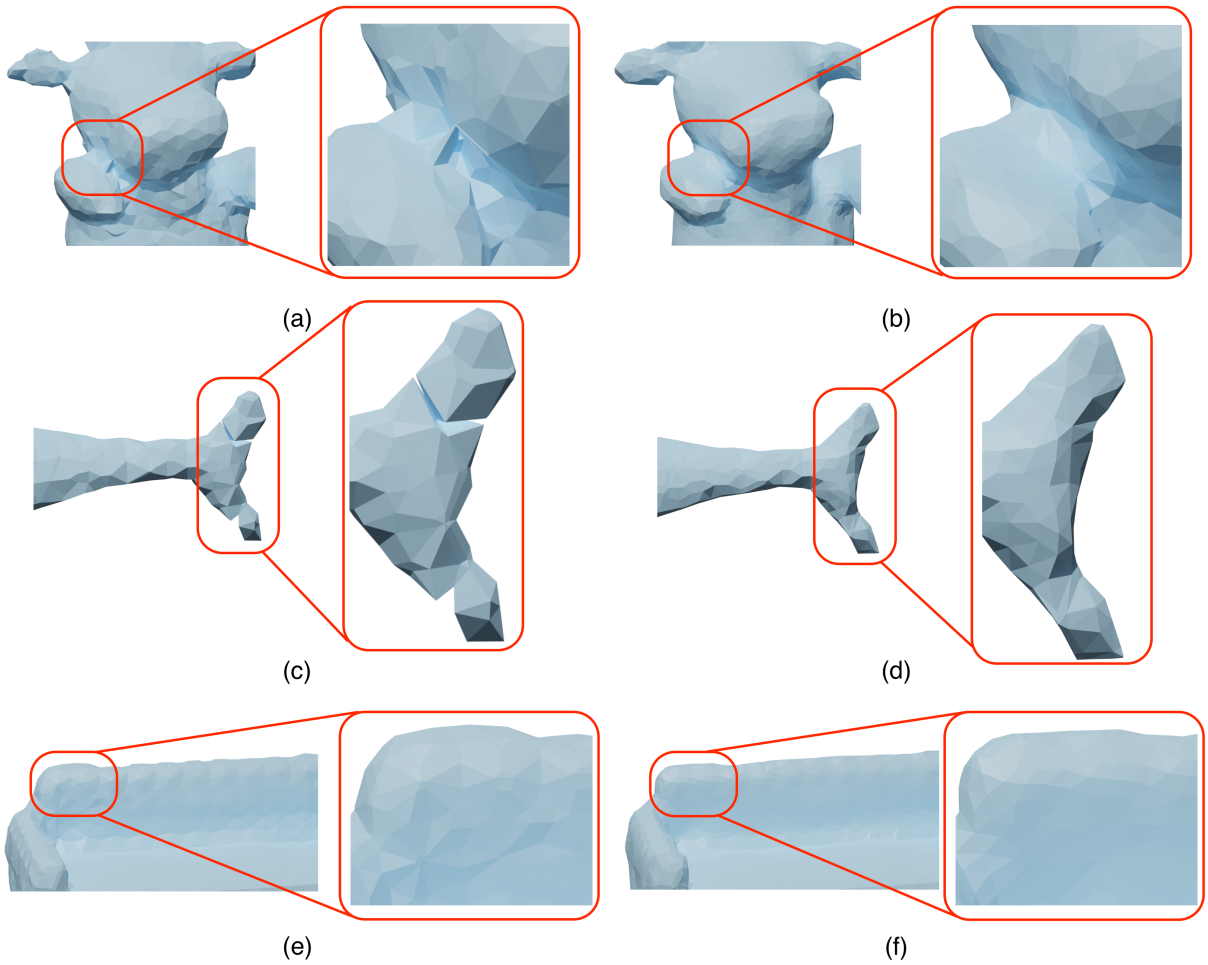

Figure 14: Qualitative comparison between with and without Loop evaluation and IMLS loss. Left column: reconstructed mesh with DMT only. Right column: Reconstructed mesh with DMT and subdivision surface fitting using the combined Chamfer and IMLS losses.

	T-shirt		Puppet	
	Mean	RMS	Mean	RMS
IMLS	0.003673	0.007482	**0.000839**	0.002866
IMLS+DR	**0.003045**	**0.005374**	0.000868	**0.002818**
DR	0.005955	0.008770	0.013661	0.017071

Figure 15: Hausdorff distance between fitting result and target shape(w.r.t bounding box diagonal)

	DMT	DMT+Loop	DMT+Loop+IMLS
Hausdorff (e-3)	2.8416	**2.8048**	2.9070
Chamfer (e-4)	1.8476	**1.8248**	1.8549

Figure 16: Quantitative results of different methods.

	1	2	3
DMT	0.4835	0.7185	0.7285
DMT+Loop	**0.5986**	0.8653	0.8750
DMT+Loop+IMLS	0.5984	**0.8654**	**0.8751**

Figure 17: Per-face quality metrics. A larger number is indicative of better triangulation quality

to work with different applications and yields improved results. Beyond that, the GPU implementation improves the plug-in's usability in Deep-learning based pipelines.

Our method has some limitations. The spatial-temporal reconstruction relies on a differential renderer and those that are currently available only support surface meshes. Therefore, for the spatial-temporal examples, it was necessary to reconstruct a triangular mesh from each static point cloud. Since the IMLS energy is based on the nearest neighbor search, the optimization can fail when the distance between the template control mesh and the target shape is too large. Thus, in the case of spatial-temporal examples, the frame-to-frame motion of the data needs to be relatively small. For the same reason, when integrating our method into Deep Marching Tetrahedra, the IMLS loss cannot be used in the early epochs of the training.

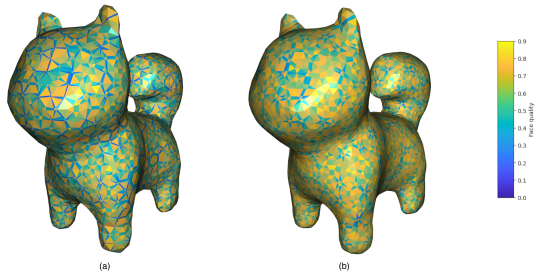

Figure 18: Per face quality $\frac{area\ of\ the\ face}{max\ side\ of\ the\ triangle}$ (a) Without Loop evaluation, (b) Adding Loop evaluation.A larger number is indicative of better triangulation quality.

	# of Vertices	Loop evaluation	Loop evaluation +IMLS
CUDA	1600	12.81ms	17.08ms
	5000	33.53ms	38.16ms
	9000	61.02ms	66.49ms
	14000	104.52ms	114.45ms
C++	1600	70.66ms	115.88ms
	5000	219.55ms	323.75ms
	9000	403.92ms	629.62ms
	14000	669.9ms	996.81ms

Figure 19: Execution time of PyTorch C++ extension and CUDA extension. All tests were executed on Nvidia RTX 3090 and Intel i9-12900k.

As mentioned in Section 5.4, in the CUDA implementation, we replace the list with a padded matrix, which has the limitation that the maximum valence of the input mesh should not exceed 12 in our implementation. Although this number can be set higher, that will lead to a performance and memory penalty. Note that valence of 12 is very unusual and may happen only in degenerate cases in the early epochs of training.

Since our focus is on geometric fitting, for fitting spatial-temporal sequence, we selected an image loss based on silhouette only. It would also be of interest to explore other image losses and point-based differential renderers.

In the future, we could also improve our method by using more accurate implicit surface reconstruction techniques from point clouds such as the one proposed by Liu et al. [30].

REFERENCES

[1] B. Allen, B. Curless, and Z. Popović. Articulated body deformation from range scan data. *ACM Transactions on Graphics (TOG)*, 21(3):612–619, 2002.

[2] R. H. Bartels, J. C. Beatty, and B. A. Barsky. *An Introduction to Splines for Use in Computer Graphics and Geometric Modeling*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1987.

[3] Y. Ben-Shabat and S. Gould. Deepfit: 3d surface fitting via neural network weighted least squares. In *European Conference on Computer Vision*, pp. 20–34. Springer, 2020.

[4] A. Bozic, P. Palafox, M. Zollhöfer, A. Dai, J. Thies, and M. Nießner. Neural non-rigid tracking. *Advances in Neural Information Processing Systems*, 33:18727–18737, 2020.

[5] E. Catmull and J. Clark. Recursively generated b-spline surfaces on arbitrary topological meshes. *Computer-aided design*, 10(6):350–355, 1978.

[6] A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. ShapeNet: An Information-Rich 3D Model Repository. Technical Report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago, 2015.

[7] K.-S. Cheng, W. Wang, H. Qin, K.-Y. Wong, H. Yang, and Y. Liu. Fitting subdivision surfaces to unorganized point data using sdm. In *12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings.*, pp. 16–24. IEEE, 2004.

[8] P. Cignoni, M. Callieri, M. Corsini, M. Dellepiane, F. Ganovelli, and G. Ranzuglia. MeshLab: an Open-Source Mesh Processing Tool. In V. Scarano, R. D. Chiara, and U. Erra, eds., *Eurographics Italian Chapter Conference*. The Eurographics Association, 2008. doi: 10.2312/LocalChapterEvents/ItalChap/ItalianChapConf2008/129-136

[9] T. DeRose, M. Kass, and T. Truong. Subdivision surfaces in character animation. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pp. 85–94, 1998.

[10] D. Doo and M. Sabin. Behaviour of recursive division surfaces near extraordinary points. *Computer-Aided Design*, 10(6):356–360, 1978.

[11] V. Estellers, F. Schmidt, and D. Cremers. Robust fitting of subdivision surfaces for smooth shape analysis. In *2018 International Conference on 3D Vision (3DV)*, pp. 277–285. IEEE, 2018.

[12] 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*, pp. 209–216, 1997.

[13] G. Guennebaud, B. Jacob, et al. Eigen v3. http://eigen.tuxfamily.org, 2010.

[14] R. Hess. *Blender Foundations: The Essential Guide to Learning Blender 2.6*. Focal Press, 2010.

[15] H. Hoppe, T. DeRose, T. Duchamp, M. Halstead, H. Jin, J. McDonald, J. Schweitzer, and W. Stuetzle. Piecewise smooth surface reconstruction. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pp. 295–302, 1994.

[16] S. Ilic. Using subdivision surfaces for 3-d reconstruction from noisy data. In *Workshop on Image Registration in Deformable Environments (DEFORM)*, pp. 1–10. Citeseer, 2006.

[17] A. Jacobson, D. Panozzo, et al. libigl: A simple C++ geometry processing library, 2018. https://libigl.github.io/.

[18] K. Kähler, J. Haber, H. Yamauchi, and H.-P. Seidel. Head shop: Generating animated head models with anatomical structure. In *Proceedings of the 2002 ACM SIGGRAPH/Eurographics symposium on Computer animation*, pp. 55–63, 2002.

[19] A. Kanazawa, S. Tulsiani, A. A. Efros, and J. Malik. Learning category-specific mesh reconstruction from image collections. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pp. 371–386, 2018.

[20] H. Kato, D. Beker, M. Morariu, T. Ando, T. Matsuoka, W. Kehl, and A. Gaidon. Differentiable rendering: A survey. *arXiv preprint arXiv:2006.12057*, 2020.

[21] H. Kato and T. Harada. Learning view priors for single-view 3d reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 9778–9787, 2019.

[22] M. Kazhdan and H. Hoppe. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG)*, 32(3):1–13, 2013.

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

[24] M.-J. Lai. Fortran subroutines for b-nets of box splines on three-and four-directional meshes. *Numerical Algorithms*, 2(1):33–38, 1992.

[25] S. Laine, J. Hellsten, T. Karras, Y. Seol, J. Lehtinen, and T. Aila. Modular primitives for high-performance differentiable rendering. *ACM Transactions on Graphics (TOG)*, 39(6):1–14, 2020.

[26] G. Lavoué, F. Dupont, and A. Baskurt. Subdivision surface fitting for efficient compression and coding of 3d models. In *Visual Communications and Image Processing 2005*, vol. 5960, pp. 1159–1170. SPIE, 2005.

[27] H. Li, B. Adams, L. J. Guibas, and M. Pauly. Robust single-view geometry and motion reconstruction. *ACM Transactions on Graphics (ToG)*, 28(5):1–10, 2009.

[28] N. Litke, A. Levin, and P. Schroder. Fitting subdivision surfaces. In *Proceedings Visualization, 2001. VIS'01.*, pp. 319–568. IEEE, 2001.

[29] H.-T. D. Liu, V. G. Kim, S. Chaudhuri, N. Aigerman, and A. Jacobson. Neural subdivision. *arXiv preprint arXiv:2005.01819*, 2020.

[30] S.-L. Liu, H.-X. Guo, H. Pan, P.-S. Wang, X. Tong, and Y. Liu. Deep implicit moving least-squares functions for 3d reconstruction. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 1788–1797, 2021.

[31] C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, University of Utah, USA, 1987.

[32] W. Ma, X. Ma, S.-K. Tso, and Z. Pan. Subdivision surface fitting from a dense triangle mesh. In *Geometric Modeling and Processing. Theory and Applications. GMP 2002. Proceedings*, pp. 94–103, 2002. doi: 10.1109/GMAP.2002.1027500

[33] M. Marinov and L. Kobbelt. Optimization methods for scattered data approximation with subdivision surfaces. *Graphical Models*, 67(5):452–473, 2005.

[34] K. Mendhurwar, G. Handa, L. Zhu, S. Mudur, E. Beauchesne, M. LeVangie, A. Hallihan, A. Javadtalab, and T. Popa. A system for acquisition and modelling of ice-hockey stick shape deformation from player shot videos. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops*, pp. 890–891, 2020.

[35] M. Meyer, M. Desbrun, P. Schröder, and A. H. Barr. Discrete differential-geometry operators for triangulated 2-manifolds. In *Visualization and mathematics III*, pp. 35–57. Springer, 2003.

[36] N. J. Mitra, S. Flory, M. Ovsjanikov, N. Gelfand, L. Guibas, and H. Pottmann. Dynamic geometry registration. In *Symposium on Geometry Processing*, pp. 173–182, 2007.

[37] N. Mohammad Khalid, T. Xie, E. Belilovsky, and T. Popa. Clip-mesh: Generating textured meshes from text using pretrained image-text models. In *SIGGRAPH Asia 2022 Conference Papers*, pp. 1–8, 2022.

[38] J. Montes, B. Thomaszewski, S. Mudur, and T. Popa. Computational design of skintight clothing. *ACM Transactions on Graphics (TOG)*, 39(4):105–1, 2020.

[39] R. A. Newcombe, D. Fox, and S. M. Seitz. Dynamicfusion: Reconstruction and tracking of non-rigid scenes in real-time. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 343–352, 2015.

[40] A. C. Öztireli, G. Guennebaud, and M. Gross. Feature preserving point set surfaces based on non-linear kernel regression. In *Computer graphics forum*, vol. 28, pp. 493–501. Wiley Online Library, 2009.

[41] L. Piegl and W. Tiller. *The NURBS Book*. Springer-Verlag, New York, NY, USA, second ed., 1996.

[42] T. Popa, I. South-Dickinson, D. Bradley, A. Sheffer, and W. Heidrich. Globally consistent space-time reconstruction. In *Computer Graphics Forum*, vol. 29, pp. 1633–1642. Wiley Online Library, 2010.

[43] N. Ravi, J. Reizenstein, D. Novotny, T. Gordon, W.-Y. Lo, J. Johnson, and G. Gkioxari. Accelerating 3d deep learning with pytorch3d. *arXiv preprint arXiv:2007.08501*, 2020.

[44] A. Sharf, D. A. Alcantara, T. Lewiner, C. Greif, A. Sheffer, N. Amenta, and D. Cohen-Or. Space-time surface reconstruction using incompressible flow. *ACM Transactions on Graphics (TOG)*, 27(5):1–10, 2008.

[45] G. Sharma, D. Liu, S. Maji, E. Kalogerakis, S. Chaudhuri, and R. Měch. Parsenet: A parametric surface fitting network for 3d point clouds. In *European Conference on Computer Vision*, pp. 261–276. Springer, 2020.

[46] T. Shen, J. Gao, K. Yin, M.-Y. Liu, and S. Fidler. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis. In *Advances in Neural Information Processing Systems (NeurIPS)*, 2021.

[47] M. Shinya. Unifying measured point sequences of deforming objects. In *Proceedings. 2nd International Symposium on 3D Data Processing, Visualization and Transmission, 2004. 3DPVT 2004.*, pp. 904–911. IEEE, 2004.

[48] O. Sorkine and M. Alexa. As-rigid-as-possible surface modeling. In *Symposium on Geometry processing*, vol. 4, pp. 109–116, 2007.

[49] J. Stam. Evaluation of loop subdivision surfaces. In *SIGGRAPH'98 CDROM Proceedings*. Citeseer, 1998.

[50] J. Stam. Flows on surfaces of arbitrary topology. *ACM Transactions On Graphics (TOG)*, 22(3):724–731, 2003.

[51] Stanford. The stanford 3d scanning repository. `http://graphics.stanford.edu/data/3Dscanrep/`.

[52] C. Stoll, Z. Karni, C. Rössl, H. Yamauchi, and H.-P. Seidel. Template deformation for point cloud fitting. In *PBG@ SIGGRAPH*, pp. 27–35, 2006.

[53] R. W. Sumner and J. Popović. Deformation transfer for triangle meshes. *ACM Transactions on graphics (TOG)*, 23(3):399–405, 2004.

[54] J. Süßmuth, M. Winter, and G. Greiner. Reconstructing animated meshes from time-varying point clouds. In *Proceedings of the Symposium on Geometry Processing*, pp. 1469–1476, 2008.

[55] A. Tevs, A. Berner, M. Wand, I. Ihrke, M. Bokeloh, J. Kerber, and H.-P. Seidel. Animation cartography—intrinsic reconstruction of shape and motion. *ACM Transactions on Graphics (TOG)*, 31(2):1–15, 2012.

[56] S. Umeyama. Least-squares estimation of transformation parameters between two point patterns. *IEEE Transactions on Pattern Analysis & Machine Intelligence*, 13(04):376–380, 1991.

[57] M. Wand, B. Adams, M. Ovsjanikov, A. Berner, M. Bokeloh, P. Jenke, L. Guibas, H.-P. Seidel, and A. Schilling. Efficient reconstruction of nonrigid shape and motion from real-time 3d scanner data. *ACM Transactions on Graphics (TOG)*, 28(2):1–15, 2009.

[58] W. Wang, H. Pottmann, and Y. Liu. Fitting b-spline curves to point clouds by curvature-based squared distance minimization. *ACM Transactions on Graphics (ToG)*, 25(2):214–238, 2006.

[59] M. E. Yumer and L. B. Kara. Surface creation on unstructured point sets using neural networks. *Computer-Aided Design*, 44(7):644–656, 2012.

[60] J. Zehnder, S. Coros, and B. Thomaszewski. Designing structurally-sound ornamental curve networks. *ACM Transactions on Graphics (TOG)*, 35(4):1–10, 2016.

[61] W. Zheng, P. Bo, Y. Liu, and W. Wang. Fast b-spline curve fitting by l-bfgs. *Computer Aided Geometric Design*, 29(7):448–462, 2012.

[62] M. Zollhöfer, M. Nießner, S. Izadi, C. Rehmann, C. Zach, M. Fisher, C. Wu, A. Fitzgibbon, C. Loop, C. Theobalt, et al. Real-time non-rigid reconstruction using an rgb-d camera. *ACM Transactions on Graphics (ToG)*, 33(4):1–12, 2014.