Isosurface Mesh Extraction from Signed Distance Fields

Jakub Nawrocki[*]
TU Wien

Philipp Erler[†]
TU Wien

Abstract

The problem of extracting explicit mesh representations from signed distance functions has a long history in Computer Graphics, with the first algorithm, Marching Cubes, being published as early as 1987. It finds application in many areas such as medicine, modelling and physical simulations, where its main application is either aiding visualization of volumetric data defined as implicit functions or converting the implicit representations into explicit formats more suitable for the task. Another prominent application of mesh extraction, is its application in differentiable mesh optimization problems which are commonly used in generative modelling and photogrammetry, where novel techniques are being developed to perform optimizations directly on polygonal meshes.

In this paper we present a variety of isosurface mesh extraction techniques that focus on extracting the explicit mesh from a signed distance field. For the five selected methods: Marching Cubes, Neural Dual Contouring, Deep Marching Tetrahedra, Flexicubes and Reach for The Spheres, we perform in-depth evaluations to determine the quality and performance of the extraction process. To that end, we prepare and describe a custom dataset for evaluating the selected techniques.

Keywords: isosurfacing, isosurface mesh extraction, signed distance field

1 Introduction

The problem of Isosurface Mesh Extraction refers to computing an explicit polygonal mesh from a given Signed Distance Field(SDF) and a chosen distance value(usually zero) which represents an isosurface. SDFs are an important representation of 3D objects and find many applications in areas such as modeling, animation, scientific simulation, and visualization [de Araújo et al. 2015]. They are also suitable for representing volume data such as scans obtained from computer tomography and magnetic resonance imaging [de Araújo et al. 2015]. They are however, hard to visualize, requiring costly algorithms such as ray tracing or particle-based methods [de Araújo et al. 2015]. On the other hard, polygonal meshes are a widely adopted representation in Computer Graphics. They are much cheaper to visualize but can only approximate a continous SDF. This tradeoff between speed and accuracy is often acceptable for many applications and therefore, a large number of methods have been developed to convert SDFs into polygonal meshes, with Marching Cubes [Lorensen and Cline 1987], one of the earliest proposed algorithms, being published in 1987.

With the recent advances in Machine Learning and Neural Networks, SDFs have become an even more important topic, as they lend themselves much better to gradient-based optimization problems than explicit represntations. Some examples of applications include: photogrammetry, generative modelling and physics simulations [Shen et al. 2023]. However, the problem of visualizing SDFs and using them in downstream applications remains and SDFs are often used as an intermediate step, before being converted to polygonal meshes. This has sparked recent developments in Isosurface Mesh Extraction methods which aim to make the extraction process differentiable and thus allowing to optimize downstream tasks directly on the extracted meshes [Shen et al. 2021] [Shen et al. 2023].

In this paper, we only consider isosurface mesh extraction from SDFs, not the general problem of 3D reconstruction which can use different input representations such as points clouds [Kazhdan 2005] [Kazhdan and Hoppe 2013] to produce explicit representations.

We aim to present and evaluate the following Mesh Extraction methods:

- **Marching Cubes**
- **Neural Dual Contouring**
- **Deep Marching Tetrahedra**
- **Flexicubes**
- **Reach for The Spheres**

Marching Cubes is one of the earliest and most popular extraction methods. This makes it a good baseline for comparisons with more advanced techniques. Neural Dual Contouring [Chen et al. 2022] is the most recent technique using the Dual Contouring [Ju et al. 2002] approach. Deep Marching Tetrahedra [Shen et al. 2021] and Flexicubes [Shen et al. 2023] are chosen to demonstrate the latest methods in the Gradient-Based Mesh Optimization problem. Finally, Reach for Three Spheres [Sellán et al. 2023] is a recent method which takes a largely different approach, by iteratively fitting an initial mesh of the same topology, to the target mesh defined by a Signed Distance Field. The choice of methods is limited by source code availability, however we aim to focus on the most recent and novel techniques in their respective categories, with the exception of Marching Cubes.

Throughout this paper, we first present the evaluated methods, as well as their predecessors and other related techniques. Next, we discuss the structure and composition of the dataset used in the benchmarking process. Then, we explain the techniques used to evaluate the extraction process, which focus on three aspects: performance, accuracy of mesh extraction in respect to the ground truth mesh and quality of the extracted mesh, irrespective of the ground truth mesh. Finally, we present and discuss the obtained.

2 Mesh Extraction Methods

This list is not exhaustive, however we include all methods relevant to the benchmarked techniques or otherwise important to the problem of isosurface mesh extraction.

[*]email: e12231223@student.tuwien.ac.at
[†]email: perler@cg.tuwien.ac.at

2.1 Spatial Subdivision Methods

According to the categorization of isosurface extraction methods by [de Araújo et al. 2015], the following algorithms belong to spatial subdivision methods. These algorithms work by subdividing the space over which the SDF is defined and approximating the isosurface locally.

2.1.1 Marching Cubes

The first and most known representative of this class of methods is the original Marching Cubes algorithm [Lorensen and Cline 1987]. It works by subdividing the SDF space into a uniform grid of cubes. Each cube is the processed independently, by first sampling the SDF at the corners of the cube. Only the sign of the SDF at the corners is important, as it determines whether the point is inside or outside of the volume defined by the isosurface. The key idea of the Marching Cubes method is defining a unique triangulation of the approximated surface inside the cube, based on the signs of the SDF at the corners of the cube. Since there are eight corners and their sign can be either positive or negative, there are $2^8 = 256$ different possible combinations which requires 256 unique triangulations of the surface inside the cube. These triangulations are prepared in such a way that the vertices on neighbouring faces of two cubes can be merged to create a manifold surface. Computing the triangulation inside every cube and joining the polygons with each neighbour, produces the final extracted polygonal mesh. Many of the 256 cases are symmetrical and can be therefore reduced, to 15 unique cases, shown in Figure 1. Black dots in the corners represent a negative SDF value (inside the volume) and the lack of a dot represents a positive value (outside of the volume).

For the chosen triangulation, the points not lying in the corners of the cube are positioned along the edge by computing the intersection of the SDF contour with the edge.

The original implementation suffers from ambiguities in the triangulation process which can result in incorrect topology of the extracted mesh. Various attempts have been made to resolve these ambiguities [Nielson and Hamann 1991] [Tcherniaev 1996]. [Thomas Lewiner and Tavares 2003] provides an efficient implementation with guarantees of topological correctness.

An inherent weakness of Marching Cubes, is its inability to represent sharp and thin features, especially if their size is smaller than the cube grid subdivision size. The rigid structure of the cube grid and limited vertex position, reduce the detail of the features that can be expressed.

2.1.2 Marching Tetrahedra

Marching Tetrahedra [Doi and Koide 1991] is an evolution of the original Marching Cubes algorithm, where the space is subdivided into tetrahedra instead of cubes. This is done by subdividing each cube into 5 tetrahedra, as shown in Figure 2.

In the case of tetrahedra the number of possible combinations of signs of SDF values at the corners is $2^4 = 16$. After considering the symmetries of this representation, the number of unique triangulation cases can be reduced to three. This method is aimed at removing the ambiguities present in the Marching Cubes algorith but is not entirely successful at removing all ambiguities [Zhou et al. 1995].

The Marching Tetrahedra produces a slightly better resolution in the extracted mesh compared to Marching Cubes when using the same sampling resolution on the SDF. This is due to the dense packing of the tetrahedra compared to cubes. For each 8 sampled SDF values a cube represents 12 edges which can contain a vertex. Meanwhile the 5 tetrahedra contained in the same space uses the same 8 SDF samples to provide 18 edges. Even with this small improvement, Marching Tetrahedra suffers from the same weakness as Marching Cubes - the inability to capture sharp and thin features.

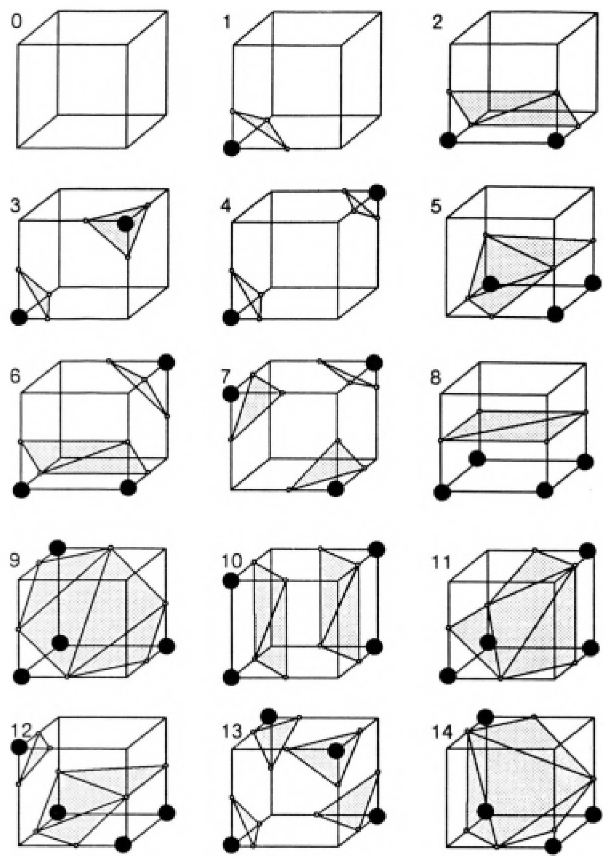

Figure 1: 15 unique triangulations of a cube. Figure 3 taken from [Lorensen and Cline 1987]

2.1.3 Dual Contouring

Dual Contouring [Ju et al. 2002] focuses on improving reconstruction quality of sharp and thin features. Instead of using a fixed number of triangulations for each cube, vertices are allowed to be moved to an arbitrary point in the cube, which best matches the countour of the object. First, in each cube which exhibits a sign change(i.e., contains at least one corner which is different than the others), each edge with different signs at the corners is considered. For these edges the intersection point is found between the SDF contour and the edge, similarly to Marching Cubes. For each intersection point found along the edge, the normal of the countour at that point. In order to find the normals at intersection points, Dual Contouring requires the input SDF to not only hold a scalar value but also the gradient at each point. Finally a vertex is generated inside the cube, which minimizes the following expression:

$$E(x) = \sum_i (n_i \cdot (x - p_i))^2 \tag{1}$$

After all vertices have been generated, for each edge which exhibited a sign change, the generated vertices from the four cubes surrounding that edge are connected to form a quadrilateral. The authors further extend the above formulation to work with octrees by

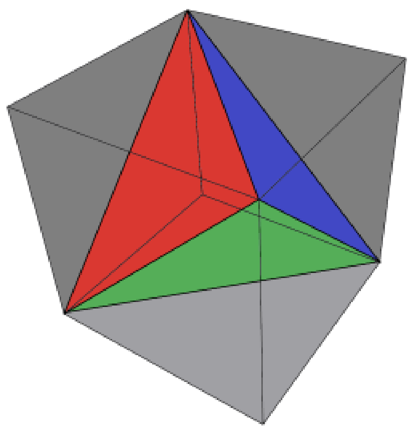

Figure 2: Subdivision of a cube into 5 tetrahedra. Fig. 3 taken from [Bagley et al. 2016]

expanding only the leaf cubes which are not homogenous in sign values at the corners. Since in the average case, most cubes in a fixed grid are homogenous, this saves a lot of computation time and memory, as well as, allows for very detailed feature extraction along the contour.

The name Dual Contouring comes from the fact that the extracted mesh is dual to the original cube grid. Each cube in the original grid corresponds to a vertex in the extracted mesh and each edge corresponds to a quadrilateral face in the extracted mesh.

Figure 3 compares the effect of Marching Cubes and Dual Contouring on a simple 2D example. The sharp feature in the upper right quadrant is correctly extracted by Dual Contouring, while Marching Cubes produces smooth approximation, not representative of the real contour.

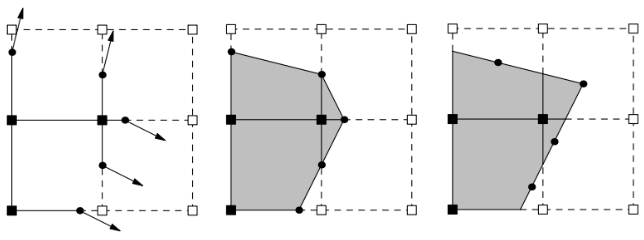

Figure 3: Comparison of extraction from computed intersection points(*left*), using Marching Cubes(*center*) and Dual Contouring(*right*). Figure adapted from Fig. 2 in [Ju et al. 2002]

The ability to reposition verticies inside the cube cells also generates high quality triangles with low aspect ratios. This is thanks to the quadratic optimization function, which tries to minimize the angles in the extracted polygons.

The main drawback of this method is that it requires a lot of extra data in the input SDF, in the form of gradients/surface normals at each sampled point.

2.1.4 Neural Dual Contouring

(find 2023 better recent paper) Neural Dual Contouring [Chen et al. 2022] aims to improve the main drawback of Dual Contouring, which is its dependency on access to the gradient of the SDF. Instead of calculating the positions of vertices in the heterogenous cubes, uses a trained neural network to directly predict them from SDF signs at the corners.

Furthermore, the authors introduce another variant of their network which predicts vertices based on a boolean flag for each edge, which determines if it intersects the SDF contour. Since this model does not rely on the signs of the SDF, it is called Unsigned Neural Dual Contouring.

Additionally, the authors show that both the SDF signs at corners and the boolean edge flags, can be predicted by a separate neural networks. These networks can be trained to use a wide range of data as input, such as: SDFs, Grids of Binary Voxels, Unsigned SDFs and Point clouds [Chen et al. 2022].

Figure 4 shows a simple outline of both regular and Unsigned Neural Dual Contouring.

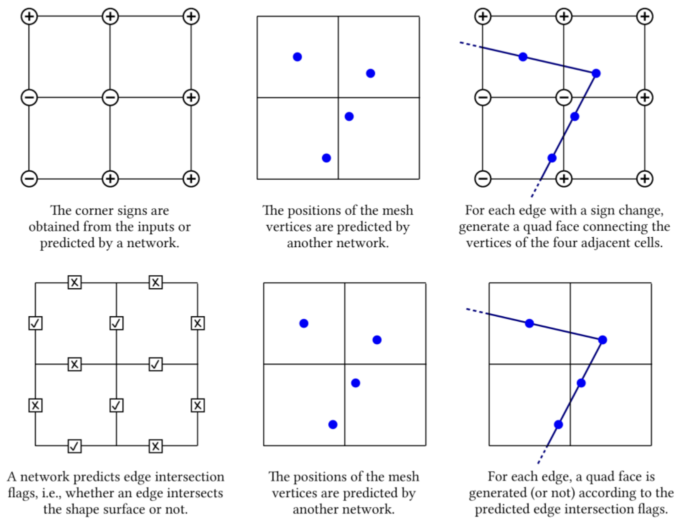
The corner signs are obtained from the inputs or predicted by a network.
The positions of the mesh vertices are predicted by another network.
For each edge with a sign change, generate a quad face connecting the vertices of the four adjacent cells.
A network predicts edge intersection flags, i.e., whether an edge intersects the shape surface or not.
The positions of the mesh vertices are predicted by another network.
For each edge, a quad face is generated (or not) according to the predicted edge intersection flags.

Figure 4: Outline of Neural Dual Contouring(*top row*) and Unsigned Neural Dual Contouring(*bottom row*). Figure adapted from [Chen et al. 2022]

2.1.5 Neural Marching Cubes (Maybe remove?)

2.1.6 Dual Marching Cubes

The Dual Marching Cubes [Schaefer and Warren 2004] algorithm is, to some extent, a combination of Dual Contouring and Marching Cubes. The general goal of this technique is to perform Marching Cubes on a grid that is aligned to the sharp features in the SDF, instead of a uniform grid, which would miss sharp features.

The first step of the algorithm, defines an octree over the whole extraction space and subdivides it adaptively. The subdivision process is controlled by a quadratic error term which judges if a given point lies on a feature of the SDF. By minimizing this error over some amount of SDF samples, the most important feature point in the sample area can be found. Each cell of the quadtree is sampled using this strategy and a minimal error is obtained. If this error is greater than some tolerance ε, the cell is subdivided and the process is repeated recursively on the new leaf cells.

After computing the octree, each cell is once again sampled using the quadratic feature error function to find the vertex in each cell, which is most likely an important feature. These vertices are analogous to the dual vertices obtained when using Dual Contouring. Another posible way of positioning the vertices in each cell is using the centroids of faces in the original grid.

In order to find the connectivity of the new dual grid with the found vertices, a recursive traversal of the octree is performed. Figure 5 illustrates an example original and dual grid overlayed in 2D.

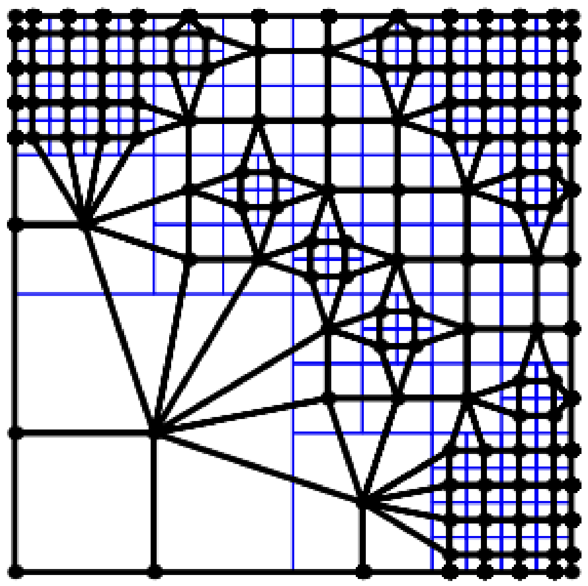

Figure 5: Example original quadtree(*blue*) and its dual grid(*black*) overlayed. Figure taken from [Schaefer and Warren 2004]

In the final step an extended Marching Cubes variant is used to extract the final mesh from the dual grid. The extension allows Marching Cubes to run on cube grids with arbitraty vertex positions. Note however, that the dual grid has the same topology as the original octree grid.

This formulation has many desirable properties. The vertices of the dual grid which are given to Marching Cubes are aligned with the sharp features in the SDF, which enhances their extraction. The adaptive octree, allows for small details to be extracted without large subdivisions of the uniform grid. The extracted meshes are guaranteed to be manifold.

2.2 Gradient-Based Mesh Optimization Methods

This set of methods still belongs to the Spatial Subdivision category. However, these methods have been developed to address the problem of Mesh Optimization. As stated before, SDFs have become a popular representation for many gadient-based techniques, thanks to their constant encoding size and intuitive differentiation. A typical architecture in downstream applications, such as generative modelling, will predict an SDF from vaious types of input data, using a deep neural network. Differentiable Isosurface Mesh Extraction techniques aim to extend the mesh optimization problem beyond the SDF and allow loss to be defined directly on the polygonal meshes.

This category of algorithms is not directly comparable with the non-differentiable ones, as the problem definition of mesh optimization is dfferent to mesh extraction. In fact, both of the techniques presented below use Marching Cubes based algorithms with minor modifications in the extraction step. However, by adjusting the optimization targets, a fair comparison can be made to evaluate the quality of the extraction performed by both differentiable and non-differentiable methods [Shen et al. 2023]. The details of this comparison is discussed in a later section of this paper.

2.2.1 Deep Marching Tetrahedra

Deep Marching Tetrahedra [Shen et al. 2021] is a generative model which predicts explicit surface representations, using point clouds or coarse voxelized shapes. The discrete SDF serves as the intermediate representation in the optimization process and is extracted into a polygonal mesh using a differentiable marching tetrahedra layer in the network.

The architecture is split up into a Generator and Discriminator, who perform their usual functions in a generative adversarial network.

The Generator extracts the final explicit mesh. First the input data is encoded into 3-dimensional feature vectors. Based on these vectors an initial SDF is predicted. The algorithm performs many small optimizations at the SDF stage to increase the quality and performance of the network. One such improvement is using a deformable tetrahedral grid [Gao et al. 2020]. During iteration, before the explicit mesh is extracted, the tetrahedral grid is refined and subdivided along the contour of the surface. This improves the quality of the extraction, similar to the adaptive subdivision found in Dual Marching Cubes. The refining process also shifts the grid vertices and modifies their SDF values. This deformations improve the ability of Marching Tetrahedra to extract sarp features. The resulting SDF is then used to extract the ploygonal mesh. Before evaluating the loss function, additional subdivision and refinement is performed, using Loop Subdivisio and prediciting offsets for the vertices of the extracted mesh. All SDF and vertex offsets and subdivision parameters are learnable parameters of the network.

The Discriminator is a 3-dimensional convolutional neural network which runs on the ”ground truth” SDF computed from the extracted mesh and the input feature vector. It predicts the probability, that the SDF came from a real mesh.

The Loss function contains three factors. Surface Alignment Loss is calculated by sampling a set of point from the predicted and ground truth mesh. The loss is calcualted as the Chamfer Distance and Normal Consistency Loss between the two point sets. Adversarial loss is used to drive the GAN network. Finally a Regularization loss is added to control SDF values which are not part of the tetrahedra on the countour of the extracted object, as the losses for those points do not get propagated backwards from the extracted mesh. The final loss is a weighted sum of all loss components.

2.2.2 Flexicubes

Flexicubes [Shen et al. 2023] is a recent method, which extends Dual Marching Cubes to define a differentiable mesh optimization technique. Just as in Deep Marching Tetrahedra, the SDF is the intermediate representation from which the explicit mesh is extracted.

The Dual Marching Cubes extraction used in Flexicubes paper considers the centroid of original grid faces as a way to position the dual vertices. The authors remark that on its own this technique hinders the ability of the extraction process to capture sharp features, as it limits the available positions for the vertices. In order to improve the the extraction quality, three additional sets of learnable parameters are chosen.

The first set of parameters are Interpolation Weights used to position the dual vertices in the original cell of the octree. Each grid cell is ammended with eight positive α-weights and twelve positive β-weights. The α-weights interpolate the position of the intersection of the mesh isosurface along each of the eight edges of the cube. The β-weights interpolate the placement of the dual vertex in respect to the edges of the octree cell.

The second set of parameters γ is used, per cell, to control how quadrilaterals are split into triangles after extraction. Since Dual Marching Cubes produces quadrilateral faces, which are not necessarily planar, slpitting the quadrilateral along the wrong diagonal can lead to artifacts. In order to train the model to avoid splitting along the wrong diagonal, a midpoint is inserted into each of the extracted quadrilaterals during optimization. The coordinates are interpolated between the four vertices of the resulting quadrilateral, controlled by γ. During the final extraction step, no midpoint is added. Instead the quadrilateral face is split along the diagonal which has a larger product of γ values from their respective cells.

The third set of parameters is similar to the grid offsets introduced in Deep Marching Tetrahedra. Each vertex in the SDF grid is assigned a parameter δ, which describes the offset of the grid vertex in space.

Flexicubes can be optimized using auto-differentiation using the provided parameters. It can be flexibly fittied as a differentiable mesh extraction technique into various mesh optimization problems.

2.3 Reach For The Spheres

Reach for The Spheres [Sellán et al. 2023] is a representative of the shrink-wrapping techniques, categorized by [de Araújo et al. 2015]. This method fits a starting mesh to the SDF by observing that the isosurface can be defined as being tangent to all spheres placed around points in the SDF with radius equal to the size of absolute value of the SDF at that point. The authors define an energy minimization problem using the SDF Energy term:

$$E_{\phi}(\Omega) = \frac{1}{2}\sum_{i=1}^{n}(\phi(p_i,\Omega) - s_i)^2 \tag{2}$$

where Ω is the surface that is being minimized, p_i is a sample point, $\phi(p_i,\Omega)$ is the value of the minimized surface Ω at p_i and s_i is the value of the ground truth isosurface at p_i.

Starting at some chosen surface, it is iteratively fitted towards the discrete, ground truth SDF using the gradient flow of the SDF Energy term.

Since this method does not modify the topology of the initial mesh during the initial optimization process, the genus of the initial surface and the target isosurface must match. One way to circumvent this limitation, suggested by the authors, is to first use a genus invariant algorithm such as Marching Cubes to get a rough mesh of the correct genus and then apply Reach for The Spheres.

According to the authors the iterative optimization process quickly introduces "degeneracies, flipped and thin triangles, and self-intersections"[Sellán et al. 2023]. In order to avoid this behaviour, the optimized mesh is re-meshed after each iteration, with a target edge length.

3 Benchmarking

3.1 Method

The testing of the selected extraction methods is divided into two categories, following the example of [Shen et al. 2023]. The first category includes non-differentiable methods: *Marching Cubes*, *Neural Dual Contouring* and *Reach for The Spheres*. These methods extract the explicit mesh representation from a fixed SDF. For each model in the prepared dataset, we calculate the SDF using pysdf[1]. Then all three methods are run on the resulting SDF to produce the extracted mesh. The original mesh in the dataset is used as the ground truth.

As mentioned before, we use the optimized version of the *Marching Cubes* algorithm [Thomas Lewiner and Tavares 2003] which is available in the scikit-image[2] python package.

The *Dual Neural Contouring* implementation is taken from the authors github repository[3]. We use pre-trained weights provided by the authors of the paper, available in the repository.

The *Reach for The Spheres* implementation is available as part of the gpytoolbox[4] python package.

The second category includes both differentiable methods: *Deep Marching Tetrahedra* and *Flexicubes*. Since the SDF in these techniques is learned by the neural network, rather than fixed, the authors of the Flexicubes paper suggest a different setup to test the extraction process. The loss function is defined directly using the ground truth mesh. Each iteration the depth and silhouette images are rendered using a differentiable renderer from a randomly sampled camera pose for the extracted and ground truth mesh. The loss is calculated from the differences between these images. Additionally, SDF loss is computed to minimize the differences between the predicted and ground truth SDF, by sampling 1000 points on the surface of the extracted mesh and computing their SDF value in the original mesh.

For the *Flexicubes* implementation, we directly use the code provided in the github repository[5] linked in the paper, using the setup described above.

The *Deep Marching Tetrahedra* implementation is also taken from NVIDIAs Kaolin github repository[6]. we replicate the same setup for the loss function based on the ground truth mesh.

3.2 Dataset Preparation

The dataset used to evaluate the extraction methods is prepared manually, using models found in Thingi10K [Zhou and Jacobson 2016], which is a collection of 3D models catalogued by various properties, such as water-tightness, self-intersections, genus and many others. The whole dataset is divided into subsets of models based on their properties, which allows me to test individual features of the Mesh Extraction methods individually. These models serve as the ground truth for the extraction process.

The first part of the dataset is constructed to measure how the techniques behave on ground truth meshes of different resolution. It consists of three sets of 100 models each. First, 100 models are downloaded from Thingy10K, using the querry in Appendix A.1, with the following features:

[1]https://github.com/sxyu/sdf
[2]https://scikit-image.org/
[3]https://github.com/czq142857/NDC
[4]https://gpytoolbox.org/0.2.0/reach_for_the_spheres/
[5]https://github.com/nv-tlabs/FlexiCubes
[6]https://github.com/NVIDIAGameWorks/kaolin/

- Water-tight
- No self-intersections
- number of faces >20.000

Each of these high polygon models is decimated three times to produce three sets of 100 models with a face count of approx. 20.000, 2.000 and 200 faces. Together these three sets contain 3 levels of detail of each of the original meshes. The lower detail geometry is generated using the **meshing_decimation_quadric_edge_collapse** filter from the pymeshlab[7] library.

The second part of the dataset aims to find the effect of self-intersections on the extracted mesh. This part consists of two sets of 100 models, one with and one without self-intersections. We obtain the sets from Thingi10K, using the querry in Appendix A.2, with the following properties:

- Water-tight
- With self-intersections for first set, without for the second set
- number of vertices >1.000

Here the level of detail is not as important, as long as it is high enough to produce interesting features.

The third part of the dataset is used to measure the effect the effect of thin and sharp features on the extraction quality. It consists of two sets of 100 models, one with and one without thin and sharp features. Once again, we obtain the sets from Thingi10K, using the querry in Appendix A.3, with the following properties:

- Water-tight
- With thin and sharp features for first set, without for the second set
- number of vertices >1.000

Each dataset element is additionally normalised before any computation. The meshes are centered at the origin and scaled, so the longest side of the axis-aligned bounding box of each mesh ranges from [-1, 1].

3.3 Evaluation Techniques

3.3.1 Performance

In order to measure the performance of each method, we record the running time of the extraction process.

3.3.2 Accuracy of extraction compared to ground truth mesh

This set of metrics is used to measure how well the extracted mesh matches the ground truth mesh. In order to define the accuracy metrics, we randomly sample 100.000 points on the surface of both meshes to create two point clouds. Using these two point clouds, we calculate the first two metrics:

- **Chamfer Distance**

  The Chamfer Distance is calculated by taking the average of the squared distances from each point in one point cloud to the nearest point in the other point cloud. This is done for both point clouds and the averages are summed. The Chamfer Distance can be written as:

$$\frac{w_1}{|P_1|}\sum_{p_{1i}\in P_1}\min_{p_{2j}\in P_2}(||p_{1i}-p_{2j}||_2^2)+\frac{w_2}{|P_2|}\sum_{p_{2j}\in P_2}\min_{p_{1i}\in P_1}(||p_{2j}-p_{1i}||_2^2) \tag{3}$$

  where P_1 is the extracted mesh point cloud and P_2 is the ground truth point cloud. w_1 and w_2 are used to weight the influence of each point clouds but are set to the default value of 1.

- **F1-Score**

  The F1-score is defined as the harmonic mean of precision and recall. Precision is calculated by considering each point in the extracted mesh point cloud. If the distance to the nearest point in the ground truth point cloud is smaller than a *threshold = 0.01*, it is counted as a true positive. If no ground truth point is found within the threshold distance, it is counted as a false positive. Recall is similarly calculated by considering each point in the ground truth point cloud. If the distance to the nearest point in the extracted mesh point cloud is smaller then the threshold, it is counted as a true positive. If no point is found it is counted as a false negative.

We calculate both *Chamfer Distance* and *F1-Score* using the Kaolin point cloud metrics module[8].

The next pair of metrics requires computing the edge set of both the extracted and grount truth point clouds. The edge set is a subset of the original point clouds which only includes points close to sharp features of the mesh. When sampling the original point clouds, for each point we also store the normal of the face which this point belongs to. Then for each point in one point cloud, we find all points within a *distance threshold = 0.1*. We calcualte the dot product between the point's normal and each of it's neighbour. If the minimum absolute value of the dot product is below the *normal threshold = 0.2* this point is added to the edge set. This operation is performed separately for the ground truth and extracted point cloud. Having both edge set point clouds, we once again calculate:

- **Edge Chamfer Distance**
- **Edge F1-Score**

which are defined in the same way as the regular Chamfer Distance and F1-Score, except on the edge set point clouds.

The final extraction accuracy metric is **Normal Inaccuracy**. It is defined as the *mean squared error* between normals of each point in the extracted point cloud and nearest neighbout in the ground truth point cloud.

3.3.3 Quality of extracted mesh

In order to judge the quality of the extracted mesh, we use the following metrics:

- **Triangle Aspect Ratios**

  Triangle Aspect Ratio is defined as the ratio of the circumradius to twice its inradius. The final formula is defined as follows:

$$\begin{aligned} s &= \frac{a+b+c}{2} \\ AspectRatio &= \frac{abc}{8(s-a)(s-b)(s-c)} \end{aligned} \tag{4}$$

[7]https://github.com/cnr-isti-vclab/PyMeshLab

[8]https://kaolin.readthedocs.io/en/latest/modules/kaolin.metrics.pointcloud.html

Sliver triangles will result in a very high aspect ratio while regular triangles with sides of similar length will produce an aspecet ratio near 1. We report the average aspect ratio across all triangles in the extracted mesh.

- **Percentage of Sliver Triangles**

  Percentage of Sliver Triangles measures the ratio of sliver triangles to the total number of triangles in the extracted mesh. A sliver triangle is defined as having an angle $<10°$.

4 Results

5 Conclusion

References

BAGLEY, B., SASTRY, S. P., AND WHITAKER, R. T. 2016. A marching-tetrahedra algorithm for feature-preserving meshing of piecewise-smooth implicit surfaces. *Procedia engineering 163*, 162–174.

CHEN, Z., TAGLIASACCHI, A., FUNKHOUSER, T., AND ZHANG, H. 2022. Neural dual contouring. *ACM Transactions on Graphics 41*, 4 (July), 1–13.

DE ARAÚJO, B. R., LOPES, D. S., JEPP, P., JORGE, J. A., AND WYVILL, B. 2015. A survey on implicit surface polygonization. *ACM Comput. Surv. 47*, 4 (may).

DOI, A., AND KOIDE, A. 1991. An efficient method of triangulating equi-valued surfaces by using tetrahedral cells. *IEICE TRANSACTIONS on Information and Systems 74*, 1, 214–224.

GAO, J., CHEN, W., XIANG, T., JACOBSON, A., MCGUIRE, M., AND FIDLER, S. 2020. Learning deformable tetrahedral meshes for 3d reconstruction. *Advances In Neural Information Processing Systems 33*, 9936–9947.

JU, T., LOSASSO, F., SCHAEFER, S., AND WARREN, J. 2002. Dual contouring of hermite data. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, 339–346.

KAZHDAN, M., AND HOPPE, H. 2013. Screened poisson surface reconstruction. *ACM Transactions on Graphics (ToG) 32*, 3, 1–13.

KAZHDAN, M. 2005. Reconstruction of Solid Models from Oriented Point Sets. In *Eurographics Symposium on Geometry Processing 2005*, The Eurographics Association, M. Desbrun and H. Pottmann, Eds.

LORENSEN, W. E., AND CLINE, H. E. 1987. Marching cubes: A high resolution 3d surface construction algorithm. *SIGGRAPH Comput. Graph. 21*, 4 (aug), 163–169.

NIELSON, G., AND HAMANN, B. 1991. The asymptotic decider: resolving the ambiguity in marching cubes. In *Proceeding Visualization '91*, 83–91.

SCHAEFER, S., AND WARREN, J. 2004. Dual marching cubes: Primal contouring of dual grids. In *12th Pacific Conference on Computer Graphics and Applications, 2004. PG 2004. Proceedings.*, IEEE, 70–76.

SELLÁN, S., BATTY, C., AND STEIN, O., 2023. Reach for the spheres: Tangency-aware surface reconstruction of sdfs.

SHEN, T., GAO, J., YIN, K., LIU, M.-Y., AND FIDLER, S., 2021. Deep marching tetrahedra: a hybrid representation for high-resolution 3d shape synthesis.

SHEN, T., MUNKBERG, J., HASSELGREN, J., YIN, K., WANG, Z., CHEN, W., GOJCIC, Z., FIDLER, S., SHARP, N., AND GAO, J. 2023. Flexible isosurface extraction for gradient-based mesh optimization. *ACM Transactions on Graphics 42*, 4 (jul), 1–16.

TCHERNIAEV, E. 1996. Marching cubes 33: Construction of topologically correct isosurfaces.

THOMAS LEWINER, HÉLIO LOPES, A. W. V., AND TAVARES, G. 2003. Efficient implementation of marching cubes' cases with topological guarantees. *Journal of Graphics Tools 8*, 2, 1–15.

ZHOU, Q., AND JACOBSON, A. 2016. Thingi10k: A dataset of 10,000 3d-printing models. *arXiv preprint arXiv:1605.04797*.

ZHOU, Y., CHEN, W., AND TANG, Z. 1995. An elaborate ambiguity detection method for constructing isosurfaces within tetrahedral meshes. *Computers & Graphics 19*, 3, 355–364.