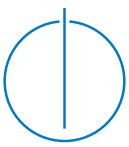
DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Free-Form Surface Texture Inpainting Using Graph Neural Networks

John Flynn



DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Free-Form Surface Texture Inpainting Using Graph Neural Networks

Freiform-Vervollständigung von Oberflächentexturen mittels graph-basierten neuronalen Netzwerken

Author:	John Flynn
Supervisor:	Prof. Dr. Matthias Nießner
Advisor:	Prof. Dr. Matthias Nießner
Submission Date:	17.01.2022

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 17.01.2022 John Flynn

Acknowledgements

This thesis is the capstone of a long and challenging academic journey made possible by a tremendous amount of help and support along the way.

First I am grateful to my supervisor Prof. Dr. Matthias Nießner for guiding me through my first steps as a young researcher over the past 1.5 years. His time and insights have helped me discover a research field I feel passionate about.

I would also like to thank Anton Troynikov for helping me get settled in Munich, walking me through the initial steps of my degree and regularly checking in on my progress. Anton's mentorship was invaluable and set the course of my graduate studies.

Most importantly, I am deeply grateful for all the support from my family that made my studies possible; especially my grandparents, whose influence started my academic journey and my Dad, who made my happiness and prosperity his principal concern.

Further Mentions: Yawar Siddiqui (technical discussions on thesis topic), Guy Gafni (informal advice about navigating the Visual Computing Group), Jihan Kahssay (help moving to Germany), Filippo Martinoni and Tülin Kol (guidance studying in Germany), Angadh Nanjangud (academia mentor), Steven Flynn (math discussions), Lancelot Kao (support system at CCSF), Stefan Su (printing this thesis), Danger crew (eating my homemade chimichangas).

Abstract

In this master's thesis in Informatics, we present the Surface Texture Inpainting Network (STINet), a graph neural network-based model that generates complete surface texture for partially textured 3D meshes. In contrast to 2D image inpainting which focuses on predicting missing pixel values on a fixed regular grid, STINet aims to inpaint color information on mesh surfaces of varying geometry and topology. STINet learns from spatial information such as vertex positions and normals as well as graph connectivity to effectively predict vertex color. Through experiments on 2D textures and real-world 3D reconstructions, we demonstrate the potential of STINet to inpaint two-dimensional manifolds with results that are perceptually comparable to classic 2D convolutions. To our knowledge, this is the first work to utilize graph neural networks for surface texture completion.

Contents

1 Introduction

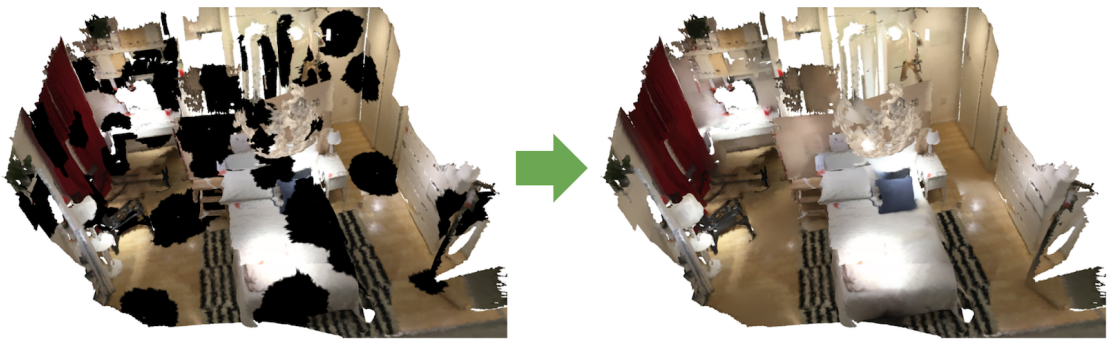


Figure 1.1: **Surface Texture Inpainting Example**. 3D mesh surface texture inpainting results (missing color left, inpainting right) using a graph-based neural network.

Image inpainting is the task of synthesizing alternative content in missing regions of an image such that the new content is perceptually convincing to humans. It allows distracting objects such as text, people, occlusions or personally identifiable information to be removed from images without affecting their perceptual quality. In the context of 3D models, inpainting could help complete partially textured surfaces such as in a 3D reconstruction of an object where texture is only available from a limited number of viewpoints. Similarly, if the texture of a sofa is re-targeted to a different object such as a bed, inpainting could help estimate a texture for the empty surface regions.

3D inpainting presents two significant challenges. First, it is inherently an ill-posed problem since for any missing texture on the surface of a 3D model there is no single right answer to how it should look. Take for instance the image inpainting example in figure 2.1 where a dog is cut from a photo and grass is inpainted into the background. Does this mean that filling the hole with grass is incorrect? If so, then how would anyone without the original picture be expected to know a dog was there (apart from the silhouette of course)?

Second, inpainting the surface texture of a 3D model necessitates inferring color in 3D space. Since the majority of modern inpainting techniques utilize generative neural networks, one might be tempted to generalize 2D convolutions into 3D. However their resolution is typically far too low (2cm typical) to discriminate fine-scale semantic patterns in surface texture. Instead of attempting to represent color as a point value in 3D space, a promising approach is to evaluate 2D convolutions directly on the mesh surface. Many practical works successfully apply surface convolutions to semantic segmentation tasks but these approaches require special care to map a 2D convolution window to a surface in 3D with minimal distortion.

To address these limitations, instead of taking the approach of quantizing a model in a voxel grid or stretching its surface into a tangent plane that is agreeable with 2D convolutions, we change the paradigm and use graph convolutions that are defined on the surface of a 3D model. In recent years, graph neural networks have been enormously successful at drawing inferences on data with irregular structure such as the structure of molecules, human joints or the relationship between objects in a scene. Our insight is that 3D models can also benefit from this representation.

One of the most significant drawbacks with generalizing image inpainting to surfaces is that there are virtually no metrics available to measure perceptual quality. We therefore first evaluate our approach on images by representing them as graphs. In summary, we make the following contributions:

- We develop a graph neural network capable of inpainting surface texture and demonstrate its efficacy on 3D scene reconstructions.
- We make a direct comparison between graph and 2D convolutions for the image inpainting task and show that in some instances they achieve almost identical performance.

The remainder of this thesis proceeds as follows: Chapter 2 gives some background on surface and graph convolutions and discusses related work in these fields. Chapter 3 presents our graph neural network model for image inpainting and details each design decision we made. Chapter 4 covers implementation challenges related to training graph neural networks and details the hyperparameters chosen for our network. Chapter 5 presents the results of our network on 2D image inpainting and 3D surface inpainting. Chapter 6 provides several paths for future research. Chapter 7 concludes our work and Chapter 8 contains the appendix.

2 Background & Related Work

In this chapter we review existing literature in image inpainting, surface convolutions and graph neural networks. Since 3D inpainting is a relatively new field we introduce the fundamentals of these fields and discuss their connection to surface texture inpainting.

2.1 Image Inpainting

Figure 2.1: **Image inpainting**. (left) Outline of an aesthetically pleasing dog. (center) Region containing the dog is removed from the image. (right) Empty region is inpainted with perceptually convincing texture.

Image inpainting is the process of estimating the most perceptually convincing value of missing pixels in an image. The majority of modern algorithms take a deep learning approach whereby missing pixel values are generated using a feed forward neural network in an encoder-decoder architecture [44, 20, 9, 45, 19].

Most approaches typically differ in their choice of loss function [13] or convolution parameters. Patch-Based Image Inpainting [9] utilizes dilated convolutions to increase their network's receptive field. In a similar fashion we formulate dilated convolutions on graphs representing 3D data. Pix2pix [20] shows that Resnet blocks perform better

than UNet [30] on many image-to-image translation tasks. We leverage this insight by defining residual blocks over graph convolutions. Many state of the art inpainting techniques also adapt partial convolutions [22]. Although they would remove the bias induced by default color values at missing vertices, partial convolutions would also mask spatial inputs which makes them inappropriate for surface inpainting.

2.2 Convolutions on Meshes

Many graphics applications require convolutional operators that can process 3D models. While 2D convolutions easily generalize to the 3D domain their resolution is typically too low to be useful. Sparse methods [14] were developed to increase the resolution of voxel grids and have had success in fine-detail geometry processing [7] but is still far below the necessary resolution for color.

An alternative approach is to define convolutions directly on the mesh surface. In the following section we explore this concept in detail.

2.2.1 Surface Convolutions

To tractably convolve high-resolution features on a mesh, many recent works attempt to define convolution operators directly on the mesh surface. The main idea is to have a convolution window slide over a mesh surface the same way it slides across an image except in 3D. However defining a convolution on a surface introduces two main problems.

The first is that we need to be able to transport a convolution window smoothly along all points on the surface of a mesh so that adjacent convolution windows always align. That way adjacent convolutions don't have large differences in their orientation and consequently extract significantly different features. However at any given point on the surface of a mesh, orientation is not immediately clear. Although there are many techniques for determining a surface orientation such as computing principal curvature or a direction field [39, 18], topological surfaces unfortunately lack shift invariance in general. Without it there will inevitably be a step change in the orientation of two adjacent conv operations at certain positions on the surface. In Figure 2.2 for example, two convolution windows undergo parallel transport along the equator of a sphere. They start with the same position and orientation but when they meet at the singularity at the top they have different orientations. It is therefore possible for a convolution window to have one of many orientations at a given point on a surface depending on the path it followed to get there. This is known as the frame inconsistency problem and

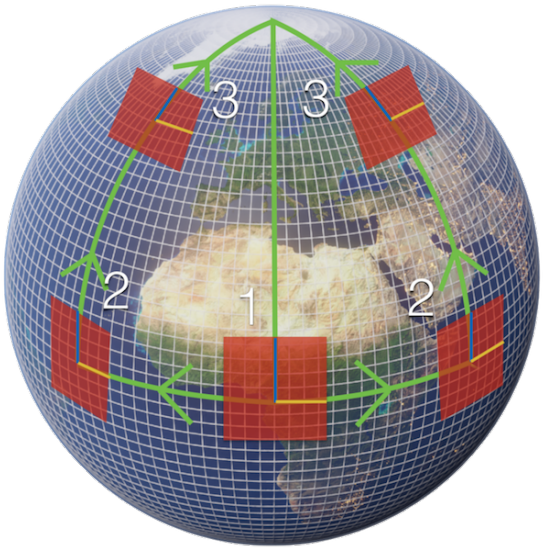


Figure 2.2: **Parallel transport of convolution windows**. (1) Window with a local axis starts at equator. (2) Two copies transport east and west and continue to agree on which directions are forward and right. (3) However as they approach the singularity at the north pole they fall out of agreement on direction.

addressing it is critical when performing convolutions on a surface in 3D.

The second is that in order to apply a 2D convolution to a mesh surface we first need to map (parameterize) the local surface relevant for the convolution to its tangent plane. Many mapping functions exist such as orthographic projection or unfolding of the geodesic neighborhood. However mappings often are not isometry-invariant (surface distance preserving) and most inevitably introduce distortion to the parameterized surface.

An early work called Tangent Convolutions [35] applies 2D convolutions to a mesh surface by aligning the convolution window with principle curvature and orthographically projecting the local surface to the tangent plane. While this often aligns well with shape features the principal curvature is not well-defined on planes and noisy surfaces. Both TextureNet [17] and PFCNN [26] significantly improve on this by imposing a rectangular direction field on the surface of the mesh that provides canonical coordinates to each point on the mesh surface. However the weights in TextureNet are symmetric up to 90 degrees which limits their expressiveness and the direction fields

in both works inevitably contain singularities. In contrast, our method proposed in chapter 3 works on any surface geometry without symmetry constraints.

Instead of applying 2D convolutions to the surface, the recent work Geodesic convolutional neural networks on Riemannian manifolds [25] define radial convolutions that more naturally capture anisotropic features (features that exist in a certain direction) and are isometry-invariant. Learning shape correspondence with anisotropic convolutional neural networks [2] take this a step further by introducing anisotropic heat kernels derived from principal curvatures. Radial convolutions however still suffer from large distortions when folding the local surface into the tangent plane which our approach overcomes by processing features directly on the vertices and edges that define the surface.

Finally, parametric models [1] exhibit a limited form of surface texture completion. Real-Time Expression Transfer for Facial Reenactment[38, 37] uses parametric face models to estimate full face albedo from a partial face scan. While these models are well suited to estimate missing texture they are domain specific. In contrast our method is not restricted to any model domain.

2.2.2 Graph Neural Networks

While spatial convolutions effectively capture hidden patterns of Euclidean data, in many domains data is best represented as graphs [43, 3]. Polygon meshes fit this representation almost perfectly with vertices as graph nodes and face edges as undirected graph edges. Graphs however are irregular with variable size of unordered nodes, each node having different numbers of neighbors, making some important operations like convolutions challenging to define.

Convolutions actually have a solid mathematical foundation in spectral graph theory. However each spectral graph convolution operation requires an eigendecomposition of the graph Laplacian which has $\mathcal{O}(n^3)$ complexity and learned features are transductive (they do not generalize to graphs of different structures). Many different flavors exist that utilize efficient and inductive approximations of spectral graph convolutions including Graph Convolutional Networks [8, 21], SAGEConv [15] and Graph Attention Networks[41].

If a graph represents a geometric structure then a popular formulation is to define spatial convolutions over the surface in the form of Message Passing Neural Networks

(MPNN). The generic convolution operator in all MPNNs can be described as

$$\boldsymbol{x}_i^{(k)} = \gamma^{(k)} \left(\boldsymbol{x}_i^{(k-1)}, \square_{j \in \mathcal{N}(i)} \phi^{(k)} \left(\boldsymbol{x}_i^{(k-1)}, \boldsymbol{x}_j^{(k-1)}, \boldsymbol{e}_{j,i} \right) \right) \tag{2.1}$$

where $\boldsymbol{x}_i^{(k-1)} \in \mathbb{R}_F$ detonates node features of node i in layer $(k-1)$, $\boldsymbol{e}_{j,i} \in \mathbb{R}_D$ detonates edge features from node j to node i, $\square$ detonates a differentiable, permutation invariant function like *sum*, *mean* or *max*, and γ and ϕ detonate differentiable functions such as Multi Layer Perceptrons.

In our work we adopt EdgeConv [42], a message passing convolution operator that showed competitive results in graph-based 3D semantic segmentation [31]. We chose this operator because it is translation invariant, inductive and naturally models local graph features. Our reasoning is that a generic operator with few restrictions is ideal to serve as a baseline for many future works in the young field of surface texture inpainting.

3 The Surface Texture Inpainting Network

In this chapter we introduce our algorithm for inpainting surface texture on a mesh. We first propose a neural network architecture to learn the inpainting task. Then we propose a definition of free-form masks on graphs. Finally, we discuss a loss function suitable for training the model on 3D scenes.

3.1 Overview

In this section we develop Surface Texture Inpainting Network (STINet), a network capable of inpainting surface texture on 3D meshes. STINet utilizes both vertex color and surface geometry information to optimally generate texture. To this end we view surface texture inpainting as a generalization of image inpainting to two-dimensional manifolds.

To formally introduce the problem, consider a polygon mesh represented by a directed graph of vertices and edges $\boldsymbol{G} = (\boldsymbol{V}, \boldsymbol{E})$ where each face edge in the polygon mesh is mapped to two opposite-facing directed edges in $\boldsymbol{E}$. Further, vertices $\boldsymbol{V}$ in the graph $\boldsymbol{G}$ are composed of colors, positions and normals $\boldsymbol{V} = (\boldsymbol{C}, \boldsymbol{P}, \boldsymbol{N})$ from the polygon mesh vertices (see section 6.1.3 for additional ways to map a polygon mesh to a graph). Finally, we have vertex mask $\boldsymbol{M} \in \mathbb{N}^{|\boldsymbol{V}|}$ where each mask entry $m_i \in \boldsymbol{M}$ is 1 if the associated vertex color value $c_i \in \boldsymbol{C}$ is valid (not masked), otherwise 0 (masked).

Given a set of directed graphs $\mathcal{G} = \{\boldsymbol{G}_1, \boldsymbol{G}_2, ..., \boldsymbol{G}_N\}$ and vertex masks $\mathcal{M} = \{\boldsymbol{M}_1, \boldsymbol{M}_2, ..., \boldsymbol{M}_N\}$, our goal is to optimize a neural network model $\mathcal{S}^*$ with a suitable loss $\mathcal{L}$ to estimate perceptually convincing vertex color values for all masked vertices $\mathcal{M}$ in $\mathcal{G}$.

$$\mathcal{S}^* = \arg\min_{\mathcal{S}} \sum_{i=1}^{N} \mathcal{L}(\boldsymbol{M}_i \odot \boldsymbol{G}_i, \boldsymbol{M}_i, \boldsymbol{G}_i | \mathcal{S}) \tag{3.1}$$

Note the generality of this formulation. We do not restrict the meshes to trimeshes, the graphs in $\mathcal{G}$ to manifolds or the masks to follow any particular pattern.

3.2 Architecture

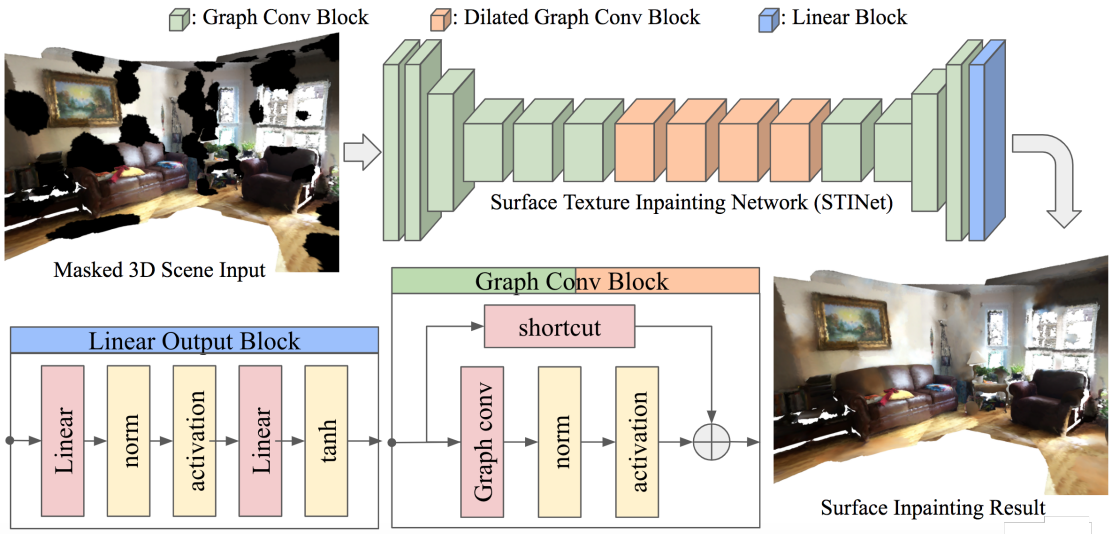


Figure 3.1: **Overview of STINet**, illustrating encoder-decoder architecture and contents of each processing block. STINet inputs a 3D model with masked vertices (black) and outputs the same 3D model with inpainted vertex color.

The Surface Texture Inpainting Network (STINet) shown in figure 3.1 is a feedforward nerual network that generalizes the image-to-image translation task to 3D surfaces. Inspired by recent worked in 2D inpainting it uses an encoder-decoder architecture that computes per-vertex features and a graph convolution operator called EdgeConv that defines a notion of convolution directly on the mesh surface. At each step in the encoder and decoder STINet pools and unpools features using simplified versions of the input mesh. To aid in increasing the receptive field of the bottleneck, STINet utilizes dilated graph convolutions in direct analogy with 2D dilated convolutions.

In this section we discuss each of these components in detail.

3.2.1 Residual Encoder-Decoder

The encoder downsamples feature maps to extract the most meaningful features and reduce both the number of learnable parameters and the amount of computation to perform in the network. Inpainting primarily occurs in the bottleneck where the lower resolution mesh and repeated blocks help maximize the spread of encoded features

along the surface. Finally the decoder upsamples the result back to the full resolution mesh.

STINet is divided into a series of three different types of blocks, each of which contains a number of convolutional or linear layers and their respective normalization and activation functions. *Graph convolution blocks* compute standard graph convolutions over the surface while *dilated graph convolution blocks* help increase the receptive field [23] (the maximum distance along the surface at which two surface points can still share information about their texture) by operating on a larger and sparser section of the surface (see section 3.2.4). The *linear output block* connects along the feature dimension of the final graph convolution output to help resolve its features back to vertex color. The linear layer therefore resembles a 1x1 convolution and can operate on a graph of any size.

Of note is the residual connection [16] in the graph convolution blocks. Instead of directly learning a filter $y = B(x)$ the network learns a residual $y = R(x) + x$ where the trainable component $R(x)$ is added to input x. This turns out to be an easier optimization problem since inpainting can be thought of as adding or subtracting color from a given image. Further, residual connections mitigate the vanishing gradient problem in neural networks by forwarding the unmodified input x to the output y.

3.2.2 Graph Convolution Operation: EdgeConv

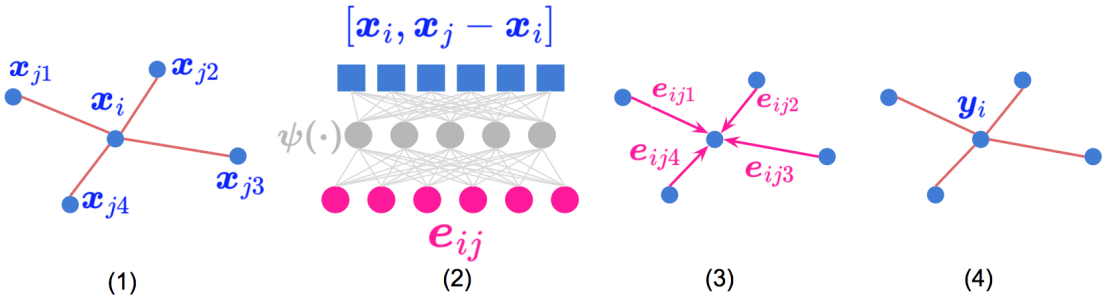


Figure 3.2: **The steps of EdgeConv**. (1) Initial edge features $\boldsymbol{x}_i$ and $\boldsymbol{x}_j \in \mathcal{N}(\boldsymbol{x}_i)$. (2) Concatenate and embed per-edge features with $\psi(\cdot)$. (3) Aggregate embedded edge features $\boldsymbol{e}_{ij}$ on center vertex i. (4) Updated vertex i with feature $\boldsymbol{y}_i$.

Learning on color and spatial data motivates the selection of a graph convolution operator that naturally models both global and local mesh features. To this end we

utilize EdgeConv [42] (as seen in figure 3.2); a simple yet powerful geodesic convolution operator that computes relative features across edges and aggregates them with absolute feature values at each vertex.

For every edge $\boldsymbol{e}_{ji} \in \boldsymbol{E}$ in the one-hop neighborhood $\mathcal{N}_i$ of $\boldsymbol{v}_i$, EdgeConv computes the difference between the source and destination vertices $\boldsymbol{x}_j - \boldsymbol{x}_i$ and passes it concatenated with destination vertex feature $\boldsymbol{x}_i$ to a Multi Layer Perceptron ψ. Then the mean aggregation of each result is computed at all vertices. Specifically, the output feature $\boldsymbol{y}_i \in \mathbb{R}^{F_{out}}$ of vertex $\boldsymbol{v}_i$ with input feature $\boldsymbol{x}_i \in \mathbb{R}^{F_{in}}$ is defined by

$$\boldsymbol{y}_i = \frac{1}{|\mathcal{N}_i|} \sum_{j \in \mathcal{N}_i} \psi([\boldsymbol{x}_i, \boldsymbol{x}_j - \boldsymbol{x}_i]; \Theta) \tag{3.2}$$

This is an especially useful representation for geometric data since it encodes relative features by construction. From the definition of Message Passing Neural Networks in section <>, message function $\phi(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{e}_{ji}) = \psi([\boldsymbol{x}_i, \boldsymbol{x}_j - \boldsymbol{x}_i]; \Theta)$ parameterized by Θ, $\square := mean(\cdot)$ and $\gamma(\boldsymbol{x}) = \boldsymbol{x}$. $[\cdot, \cdot]$ is a concatenation operator along the feature dimension.

For the message function we use a two-layer MLP given by

$$\psi(\boldsymbol{x}) = \boldsymbol{W}_2 \zeta(\boldsymbol{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2 \tag{3.3}$$

where $\boldsymbol{W}_1 \in \mathbb{R}^{F_{in} \times 2F_{out}}, \boldsymbol{b}_1 \in \mathbb{R}^{2F_{out}}, \boldsymbol{W}_2 \in \mathbb{R}^{2F_{out} \times F_{out}}, \boldsymbol{b}_2 \in \mathbb{R}^{F_{out}}$ and $\zeta(\cdot)$ is an activation function. F_{in} and F_{out} are analogous to input and output channels of a 2D convolution. We limit the depth to two layers since deeper fully connected networks quickly become difficult to train and instead increased the hidden layer to the largest size that safely fit in the memory of a single GPU.

Since EdgeConv embeds both absolute and relative features, care must be taken to preserve translation invariance for spatial inputs. We modify the very first EdgeConv operation in the network to only use differences as input.

$$\phi_{firstconv}(\boldsymbol{x}_i, \boldsymbol{x}_j, \boldsymbol{e}_{ji}) = \psi([\boldsymbol{x}_j - \boldsymbol{x}_i]; \Theta) \tag{3.4}$$

While this formulation also makes color a relative quantity, in our experiments it did not have a negative impact on model performance so long as all subsequent convolutions are still capable of embedding absolute features.

So far the definition of EdgeConv does not remove global orientation since there is no explicit local axis to which neighboring vertices can orient. Although local position

is trivial to compute, as mentioned in chapter 2 it is not possible to compute smooth direction fields on every mesh surface without the existence singularities (locations where step changes in orientation will occur). Instead we push the problem of modeling orientation to the network itself by randomly rotating each mesh about the z-axis every epoch.

$$P' = \boldsymbol{R}_z(\theta)P, \quad \theta \leftarrow RandomSample(0, 2\pi) \tag{3.5}$$

This has the effect of making the network *approach* orientation invariance in the xy plane. Orientation about the x or y axis is not necessary in this specific work as mesh reconstructions of 3D scenes are not expected to be upside down.

3.2.3 Pooling and Unpooling

Pooling provides an approach to downsample feature maps (for example, an image or surface) by summarizing features within a local region. While the regular structure of images make pooling operations such as MaxPool and AvgPool trivial to implement, pooling vertices on a mesh surface is not as straightforward.

We build a level hierarchy of increasingly simplified meshes $(\boldsymbol{G}^0, ..., \boldsymbol{G}^l, ..., \boldsymbol{G}^L)$ by collapsing edges and keeping track of fused vertices between them. We then define pooling operations using maps $(\boldsymbol{T}^{(0,1)}, ..., \boldsymbol{T}^{(L-1,L)})$ between original vertices and fused vertices in the next level.

Mesh Simplification

Inpainting on the mesh surface motivates a mesh simplification algorithm that minimally alters mesh geometry. We use Quadric Error Metric [12], a surface simplification algorithm that iteratively collapses vertex pairs $(\boldsymbol{v}_1, \boldsymbol{v}_2)$ as shown in figure 3.3 to a new representation $\bar{\boldsymbol{v}}$ according to an approximate error in the geometric distortion that this contraction induces.

Each vertex is associated with a set of planes defined by the normals of the faces that meet the vertex, and the error induced by each vertex is defined by the sum of squared distances to the set of its planes.

$$\Delta(\boldsymbol{v}) = \sum_{\boldsymbol{p} \in planes(\boldsymbol{v})} (\boldsymbol{p}^T \boldsymbol{v})^2 = \boldsymbol{v}^T \left(\sum_{\boldsymbol{p} \in planes(\boldsymbol{v})} \boldsymbol{p}\boldsymbol{p}^T \right) \boldsymbol{v} = \boldsymbol{v}^T \boldsymbol{Q} \boldsymbol{v} \tag{3.6}$$

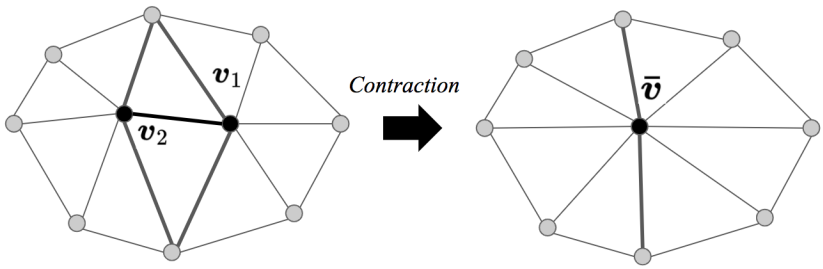


Figure 3.3: **Edge Contraction**. Vertices v_1 and v_2 are contracted into a single vertex $\bar{v}$. Highlighted edges of adjacent vertices are joined.

Computing the error of each vertex contraction $(v_1, v_2) \rightarrow \bar{v}$ is then as simple as summing the squared distances between $\bar{v}$ and all the planes associated with v_1 and v_2. As shown in equation 3.6 this is equivalent to summing each vertex's quadric $\boldsymbol{Q}$.

$$e_{\bar{v}} = \bar{v}^T(\boldsymbol{Q}_{v_1} + \boldsymbol{Q}_{v_2})\bar{v} \tag{3.7}$$

The Quadric Error Metrics algorithm computes the error $e_{\bar{v}}$ for all candidate contractions $(v_1, v_2) \rightarrow \bar{v}$ in $\boldsymbol{G}^l$ and iteratively applies each contraction to the mesh starting from the smallest error until the desired number of contractions is reached, creating $\boldsymbol{G}^{l+1}$.

At each mesh simplification step we enforce the mesh to contain 30% of the vertices as the previous hierarchical level (see figure 3.4). This resembles a pixel reduction of 25% induced by standard pooling operations on images while ensuring low-resolution geometry in our ScanNet dataset (see section 5.2.1) doesn't disappear entirely.

Vertex Mapping

Every pooling operation needs to know how vertices in $\boldsymbol{G}^l$ are connected to vertices in the simplified mesh $\boldsymbol{G}^{l+1}$. After an application of the Quadric Error Metric algorithm, multiple vertices $\{v_i^l\} \subset \boldsymbol{V}^l$ are collapsed into a single representative vertex $v^{l+1} \in \boldsymbol{V}^{l+1}$. These connections are recorded in the non-injective surjective vertex map $\boldsymbol{T}^{(l,l+1)} \in \mathbb{N}^{|V^l|}$ where each element $i \in \boldsymbol{T}^{(l,l+1)}$ is the index of a vertex $v_i^{l+1} \in \boldsymbol{V}^{l+1}$.

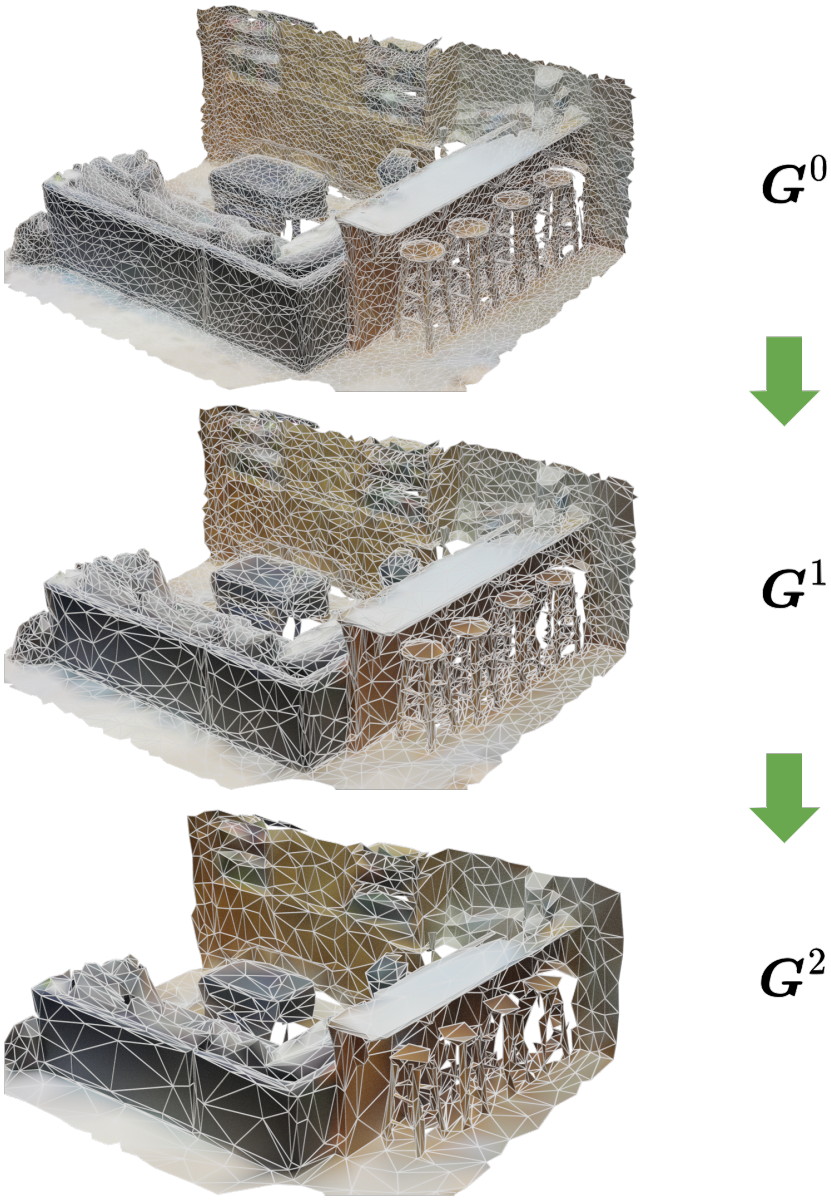


Figure 3.4: **Mesh Simplification Hierarchy**. A mesh of an indoor scene repeatedly decimated using Quadric Error Metrics. Each level contains 30% as many vertices as the previous level. Trimesh edges are superimposed in white.

Pooling & Unpooling Operators

Vertex Maps enable various pooling operations that send multiple vertex features in a higher resolution mesh to a single vertex in a lower resolution mesh where they are aggregated. STINet implements pooling using *max aggregation*.

Similarly, unpooling operators can reverse-lookup vertices in Vertex Maps. STINet implements unpooling using *nearest neighbor interpolation*.

3.2.4 Dilation on Graphs

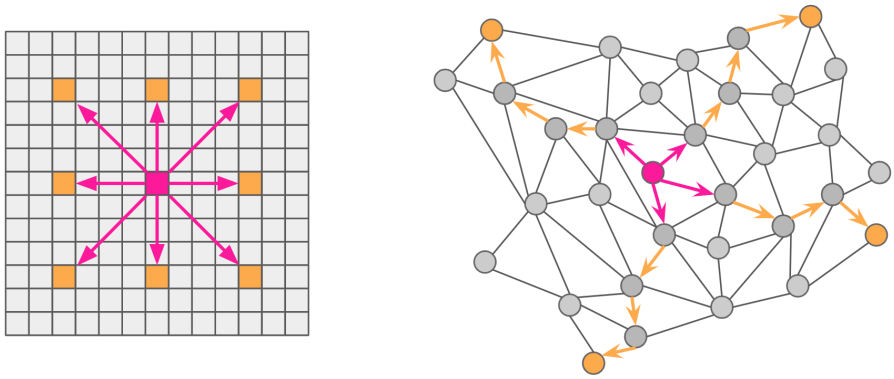


Figure 3.5: **Image Dilation vs Graph Dilation**. Two analogous examples of dilation with distance 4. In an image (left) the dilated pixel positions follow straight lines. On a graph (right) the dilated vertex neighborhood is found by following surface geodesics.

A common strategy to increase the receptive field of a network is to dilate convolutions. Several successful 2D inpainting networks [44] utilize consecutive dilated convolutions in their bottleneck layers to spread out distant features in an image. We extend this concept to meshes by selecting a more distant neighborhood for each vertex, connected by directed graph edges.

We define dilation as a traversal along a geodesic on a two-dimensional manifold (a straight line on a curved surface) starting from the center vertex. Although we are technically traversing vertices in a graph, we think of graph traversal as an approximation of the straight line path we would follow on a continuous representation of the mesh surface.

The main idea is that the direction vectors between a vertex $\boldsymbol{v}$ and each of its one-hop neighbors $\boldsymbol{v}_i^1 \in \mathcal{N}_1(\boldsymbol{v})$ define the straight line dilation directions $\boldsymbol{d}_i = \boldsymbol{v}_i^1 - \boldsymbol{v}$. Starting with each one-hop neighbor $\boldsymbol{v}_i^1$ we iteratively project the geodesic path's direction vector $\boldsymbol{d}_i$ into the plane defined by that vertex's normal $\hat{\boldsymbol{n}}_i^1$ using $\boldsymbol{d}_i \leftarrow \boldsymbol{d}_i - \frac{\boldsymbol{d}_i \cdot \hat{\boldsymbol{n}}_i^1}{\|\boldsymbol{d}_i\|} \hat{\boldsymbol{n}}_i^1$ which updates the path's direction on the curved surface. Then select one of $\boldsymbol{v}_i^1$'s unvisited neighbors $\boldsymbol{v}_i^2$ that is closest to the path drawn by $\boldsymbol{d}_i$ and repeat the process until vertex $\boldsymbol{v}_i^d$ where the desired dilation d is reached. The final vertex $\boldsymbol{v}_i^d$ defines a directed edge to the center vertex $(\boldsymbol{v}_i^d, \boldsymbol{v})$. The full algorithm is given in algorithm 1.

Algorithm 1 Computes a set of directed edges on graph G for vertex *center*.

$\mathcal{N}_G(v) :=$ one-hop neighborhood of vertex v
$\text{PROJ}_n(a) :=$ projection of a onto the *plane* defined by normal n
function ComputeDilations($center, G = (E, V)$, pos, norms, dilations)
  for $onehop \in \mathcal{N}_G(center)$ **do**
    $last \leftarrow center$
    $current \leftarrow onehop$
    $dir_{current} \leftarrow \text{pos}[current] - \text{pos}[last]$
    for $dilation = 2 \rightarrow \text{MAX}(\text{dilations})$ **do**
      $next \leftarrow None$
      $maxsim \leftarrow 0$ ▷ 0 enforces max angle $\leq 90°$
      $dir_{current} \leftarrow \text{PROJ}_{\text{norm}[current]}(dir_{current})$
      for $neighbor \in \mathcal{N}_G(current) \setminus (\mathcal{N}_G(center) \cup \{last\})$ **do**
        $dir_{neighbor} \leftarrow \text{PROJ}_{\text{norm}[current]}(\text{pos}[neighbor] - \text{pos}[current])$
        $sim \leftarrow \text{CosineSimilarity}(dir_{current}, dir_{neighbor})$
        if $sim \geq maxsim$ **then**
          $next \leftarrow neighbor$
          $maxsim \leftarrow sim$
      if $next$ **then**
        dilated_edges[dilation].ADD($(next, center)$)
        $last \leftarrow current$
        $current \leftarrow next$
      else
        break
  return dilated_edges

3.3 Free-From Mask Generation

A free-form mask is a mask that may take any shape. In line with keeping the inpainting problem formulation general we do not wish to place any restrictions on the mask. Free-form in this case implies that the mask M may cover any subset of the vertices in V. We model free-form masks on surface texture as randomly placed geodesic circles of radius $r = 16$ vertex hops that cover a threshold $\tau \geq 25\%$ of the mesh vertices.

Since vertex density varies on both the surface of a mesh and between meshes, it is difficult to know how many geodesic circles are required to cover the desired number of vertices. The mask generation algorithm therefore takes an iterative approach: First it randomly selects 10 center vertices, generates geodesic circles around each using a breadth first search and then labels all neighboring vertices belonging to each geodesic circle as masked. If the fraction of masked vertices is still less than the threshold τ then additional center vertices are sampled in proportion to the deficit fraction of masked vertices and the process is repeated.

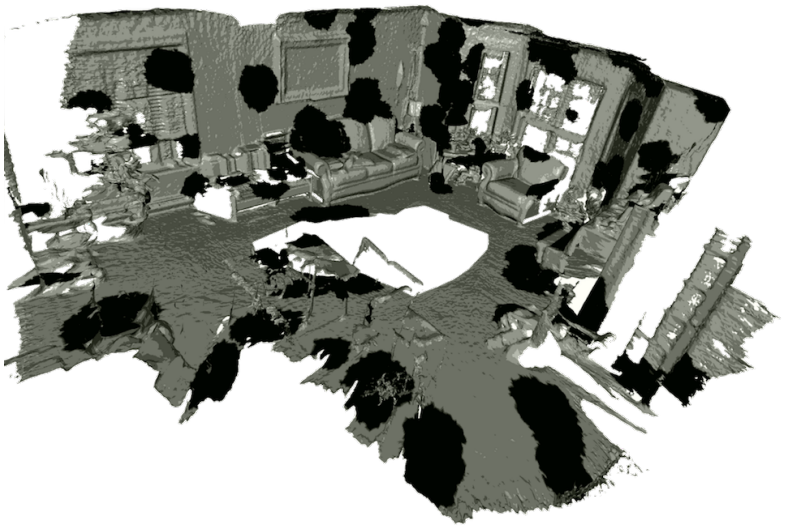


Figure 3.6: **Scene Masks**. Circular geodesic masks superimposed on mesh geometry. Each circle is set at a random location. Enough circles are placed to cover a minimum fraction of vertices determined by threshold τ.

3.4 Masked Spatially Discounted L1 Loss

We formulate a vertex-wise masked and weighted L_1 loss over the ground truth and inpainted color values $\boldsymbol{C}_i$ and $\mathcal{S}(\cdot)$ respectively for all N scenes. L_1 is chosen as it is less sensitive to outliers and therefore promotes sharper edges in image generation tasks than L_2 loss. The network $\mathcal{S}$ takes masked color $\boldsymbol{M}_i \odot \boldsymbol{C}_i$, position $\boldsymbol{P}_i$, normals $\boldsymbol{N}_i$ and vertex mask $\boldsymbol{M}_i$ as a concatenated input tensor in addition to graph edges $\boldsymbol{E}_i$. All L_1 losses over valid vertices are ignored by element-wise multiplication with $(1 - \boldsymbol{M}_i)$.

Intuitively, masked vertices near the mask boundary have much less ambiguity than vertices closer to the mask center. It is therefore preferable to encourage the network to optimize over these vertices when there is an otherwise equal prediction error on the center vertices. To account for this we use a spatially discounted weighting of the loss at each vertex. For a graph i the loss of each masked vertex v_k^i is weighted by the geodesic distance $d_k^i \in \boldsymbol{D}_i \in \mathbb{N}^{|\boldsymbol{V}_i|}$ (in number of node hops) to the nearest unmasked vertex. The weighting function is given by $\gamma^{\boldsymbol{D}_i}$ for which we use $\gamma = 0.99$.

Putting this all together, the total loss is given by,

$$L = \sum_{i=1}^{N} \gamma^{\boldsymbol{D}_i} (1 - \boldsymbol{M}_i) \odot |\mathcal{S}([\boldsymbol{M}_i \odot \boldsymbol{C}_i, \boldsymbol{P}_i, \boldsymbol{N}_i, \boldsymbol{M}_i], \boldsymbol{E}_i) - \boldsymbol{C}_i| \tag{3.8}$$

Note that in this formulation $\boldsymbol{M}_i = (\boldsymbol{D}_i == 0)$.

4 Implementation Details

In this chapter we discuss the techniques required to run STINet on a physical machine. First we discuss how each scene is divided into chunks that are small enough to fit on a single GPU. Then we review the data augmentation techniques and model hyperparameters to train STINet.

4.1 Scene Cropping

Typically in the 2D domain all images are the same size and all pixels have the same positional relationship with their neighbors. However the size and connectivity of any two graphs can differ which means that the graphs of all samples in a batch must each be loaded into memory. This creates two problems. 1) Samples take up much more memory for a given batch size and 2) The memory requirements of every batch is unknown before runtime. It is therefore much more difficult to fit data into memory and to prevent out-of-memory errors.

To ensure a set of 3D meshes will fit into memory during runtime, each mesh is cropped into 3x3 meter chunks with a 1.5 meter stride along the xy-axis. (the axis that run along the floors). A half stride with a sufficiently large crop ensures that all semantically meaningful regions of a mesh are entirely captured in at least one chunk. Even at this size one chunk requires most of our GPU memory so we do not utilize batching.

A benefit of cropping is that we can ignore processing chunks that don't contain enough information to be useful for training. If a chunk contains less than 50 vertices or if less than 2% of its vertices are masked then the chunk is excluded from training.

4.2 Data Augmentation

We utilize several transformations on the spatial features of the data. In addition to random rotation used for orientation invariance in EdgeConv, each vertex's normal and position are randomly flipped about the z axis. Vertex positions also undergo

small random perturbations in the entire linear map space. This not only augments the mesh's position and rotation but also applies scaling and horizontal shearing.

The masks themselves also undergo augmentation. We generate 16 different masks for each scene and randomly choose one every epoch.

4.3 Model Parameters and Training

The encoder and decoder of our Surface Texture Inpainting Network uses 2 downsample and 2 upsampling layers respectively to balance the network bottleneck's representability with training feasibility. Each downsampling layer uses max pooling while each upsampling layer uses nearest neighbor interpolation which we found performs better than mean or learned strided pooling. The bottleneck layer uses 9 ResNet blocks. Following convention in recent image inpainting papers, we use instance normalization, exponential linear units [5] $ELU()$ as activation functions and $Tanh()$ on the output layer. Filters on each layer use twice as many filters as the previous layer starting with 64 filters on the first layer to promote learning fewer but more complex channel features. We train our model with the popular Adam optimizer and chose a learning rate of $7e{-}5$ using random hyperparameter search. In total SCINet contains 4.2 million trainable parameters.

5 Evaluation

In this chapter we evaluate STINet on real-world data. We start by comparing STINet to 2D convolution-based networks in the image domain. We then use these results to motivate design choices in STINet and evaluate it on 3D reconstructions of real-world scenes.

5.1 2D Image Inpainting

The greatest challenge in evaluating the performance of 3D inpainting is finding suitable benchmarks and metrics. There is very little prior research to help motivate design decisions and to the best of our knowledge no perceptual metrics exist that evaluate 3D data directly. In contrast, 2D Image inpainting is a well-developed field with many different existing models to use as benchmarks as well as a number of 2D perceptual metrics that can evaluate the photo-realistic quality of a generated image.

We leverage the insight that an image can be interpreted as a graph with a regular grid structure to evaluate graph convolutions on images and compare them directly with 2D convolutions. We then argue that if our network performs well on 2D image inpainting its performance will also generalize to 3D data.

5.1.1 Graph Convolution on an Image

A 2D convolution window typically passes over adjacent vertices in a rectangular region of an image to utilize the principle of locality; that is, neighboring pixels are typically correlated. This relationship can similarly be modeled with a graph where pixels are vertices and neighboring pixels are connected with edges. Figure 5.1 shows three of such possible relationships. In the case of a 3x3 2D convolutional kernel, star connectivity directly represents the equivalent pixel neighborhood. However the drawbacks of star connectivity are that it isn't manifold (edges cross) and the number of edges is directly proportional to computation time. Tri connectivity explicitly models the image as a trimesh but it is also asymmetric.

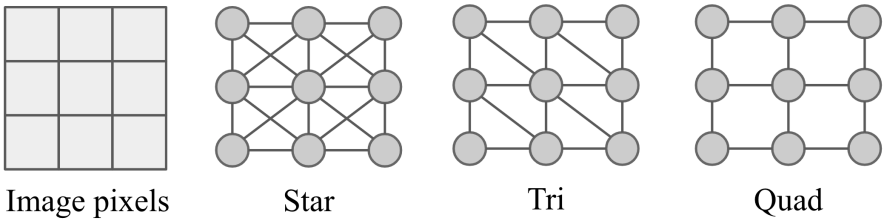


Figure 5.1: **Image as a Graph**. Three different edge connectivity kernels representing a center pixel's neighborhood as a graph. Star connects each pixel with its eight neighbors. Tri connects all but right diagonal neighbors so that the image can be interpreted as a trimesh. Quad connects horizontal and vertical neighbors so that the image can be interpreted as a quadmesh.

In our experiments we use quad connectivity since it strikes a balance between symmetry, neighborhood representation and computation time. We also compare the performance of quad connectivity with that of star connectivity.

5.1.2 Texture Dataset

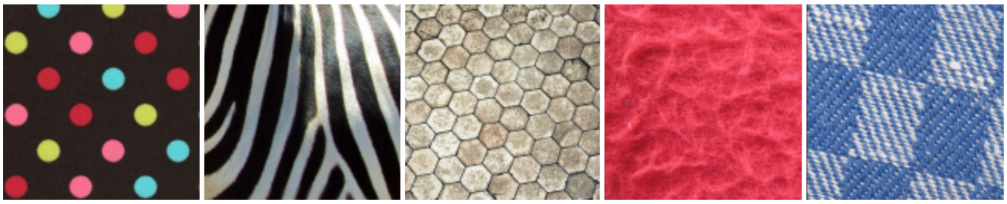


Figure 5.2: **Texture Dataset**. Collection of textures with various surface properties.

There is no shortage of online image datasets containing a wide variety of photographs in specific domains such as faces, anime characters, facades, cats or even celebrities eating. However our domain is narrower and focuses primarily on the surface texture of 3D models constructed from real objects and scenes such as a chair or bathoom. We therefore need an image dataset that is comprised primarily of *textures* and not photographs.

In our experiments we use an aggregate of several smaller texture datasets (compiled by Yawar Siddiqui) that includes images from Describable Textures [40], Salzburg Texture Image Database [29], Pixar 128 [33], KTH-TIPS [24] and Colored Brodatz Texture [10]. In total the dataset contains 4000 images with a variety of patterns such as

checkered, fibrous, striped, zigzagged and wrinkled. We divide this dataset into 3300 training and 700 validation images for our experiments.

5.1.3 Evaluation Metrics

Quantitative evaluation of inpainting results is notoriously difficult because few metrics exist that correlate well with human perceived quality. While it is straightforward to measure the differences in color values between the pixels of ground truth and generated images, there are many possible alterations that a human would unlikely notice but that would nonetheless induce a large mathematical error. For example, if an image is shifted a few pixels in a certain direction or has a slight change in contrast, a human would not perceive any degradation in quality while the change in every pixel value would accumulate to a massive error. To complicate matters further, humans are subjective in their own judgment of quality.

A glimmer of hope is that when viewed together, a collection of different metrics can be very informative. We therefore use the following five metrics to evaluate the quality of our generated images:

- **L1** *Least Absolute Deviation* measures the absolute difference between each pixel value for all pixels. It is perhaps the most simple of all metrics.
- **PSNR** *Peak Signal to Noise Ratio* is commonly used to measure reconstruction quality in images and video subject to lossy compression. It measures the strength of the mean squared error with respect to the range of possible pixel values.
- **FID** *Fréchet inception distance* [28] is a perceptual metric that measures the similarity between two datasets of ground truth and generated images. It computes the distance between two distributions fitted to feature representations of each dataset generated by the Inception network [34]. It was shown to correlate well with human judgment of visual quality.
- **LPIPS** *Learned Perceptual Image Path Similarity* [46] is also a perceptual metric that aims to characterize the abstract notion of perceptual similarity. It measures the difference in activations of two images passed through the middle layers of the VGG network [32].
- **Human Eye** There is of course (unfortunately) no true substitute for human perceptual judgment. It is not practical (and not without significant bias) to individually judge every generated image in every iteration of development. However as always, we keep our eyes peeled for patterns and do the best that we can.

5.1.4 Analogous 2D Conv-based Model

We compare STINet with a directly analogous Residual Encoder-Decoder called STINet2D that replaces graph convolutions with 3x3 2D convolutions. All pooling, unpooling, normalization and activation layers are identical to STINet. STINet2D only inputs color since normals and displacements are identical for all pixels. It has 6.1M trainable parameters in total.

In the same style as for surfaces we randomly place circular masks on each image such that no less than a threshold fraction of pixels $\tau = 0.25$ are removed.

5.1.5 Image Inpainting Results

In figure 5.3 we evaluate STINet on images using three different graph convolutions as well as a 2D convolution. Each ground truth image is masked by four circles and the rightmost two columns contain color predictions for those masked regions by EdgeConv and a 2D convolution with a 3x3 kernel. Surprisingly, their results are nearly perceptually identical except for EdgeConv's weaker ability to generate anisotropic features (completion pattern in EdgeConv's 4th row is not straight). Quantitative results in table 5.1 are similar: EdgeConv nearly matches Conv2D's perceptual metrics.

Since we use EdgeConv with a quad kernel it is important to evaluate against alternative choices. Table 5.1 additionally gives the metrics for EdgeConv with a star kernel. We conclude that while a star kernel more closely matches the pattern of a 2D convolutional kernel it must learn a multi-layer perceptron over a larger number of neighbors. This likely doesn't help because the additional nodes in a star kernel do not contain much additional information. We also include SAGEConv [15]: another graph convolution that is similar to EdgeConv except that it samples the local neighborhood instead of aggregating all features and it does not compute features across edges. While SAGEConv is effective at learning with various vertex degrees, in surface and image

Category	Method	ℓ_1 loss ↓	PSNR ↑	FID ↓	LPIPS ↓
Graph	SAGEConv	7.93%	18.19	125.9	.274
	EdgeConv Star	7.35%	18.74	120.3	.257
	EdgeConv Quad (ours)	**7.10%**	**19.04**	**105.1**	**.241**
2D	Conv2D 3x3	7.02%	19.30	100.7	.235

Table 5.1: Quantitative comparison of graph-based image inpainting results with 2D benchmark.

inpainting most vertices have roughly the same number of neighbors. This could explain why SAGEConv is suboptimal for the inpainting task.

Our 2D inpainting results are very informative about the performance of graph convolutions on 3D inpainting. As shown in table 5.1 EdgeConv performs nearly as well at inpainting images as 2D convolutions which provides a promising upper bound on EdgeConv's capacity to learn. Figure 5.3 illustrates that STINet with EdgeConv has a large enough receptive field to cover our proposed masks. These results are easy for a human to evaluate since they are given on images. Finally, the image domain allows us to evaluate EdgeConv with perceptual metrics which corroborates estimates of quality by the L_1 metric.

5.2 3D Scene Inpainting

5.2.1 ScanNet Dataset

To evaluate the performance of STINet on surface texture inpainting we use the ScanNet [6] dataset. ScanNet contains a wide variety of indoor scene reconstructions such as bedrooms, kitchens and offices. Each scene contains a 3D mesh with per-vertex colors that is reconstructed using footage from an RGB-D camera. Further, ScanNet provides each mesh in full-resolution and low resolution. We use low resolution meshes in our experiments since the average full-resolution ScanNet scene contains several million vertices—far too many to fit into a single GPU during a forward pass through STINet.

We utilize the public train, validation and test split of 1201, 312 and 100 scans, respectively. Preprocessing of ScanNet including mesh simplification levels, graph dilation, mask generation and scene cropping took 2.5 days on 12 CPU cores. Training STINet on ScanNet took 3 days on an NVIDIA GeForce GTX 1080 Ti.

5.2.2 Evaluation Metrics

Much like the problem of defining metrics to evaluate perceptual quality of images, surface texture has virtually no metrics for quantitative evaluation. In fact, there are currently no existing perceptual metrics that evaluate surface texture directly on 3D data. We therefore rely on the assertion that per-pixel metrics ℓ_1 and PSNR correlate reasonably well with perceptual quality, which the image inpainting results in section 5.1.5 corroborate. ℓ_1 and PSNR extend trivially to graphs (replace pixels with vertices) so these are our primary metrics to measure the quality of our network.

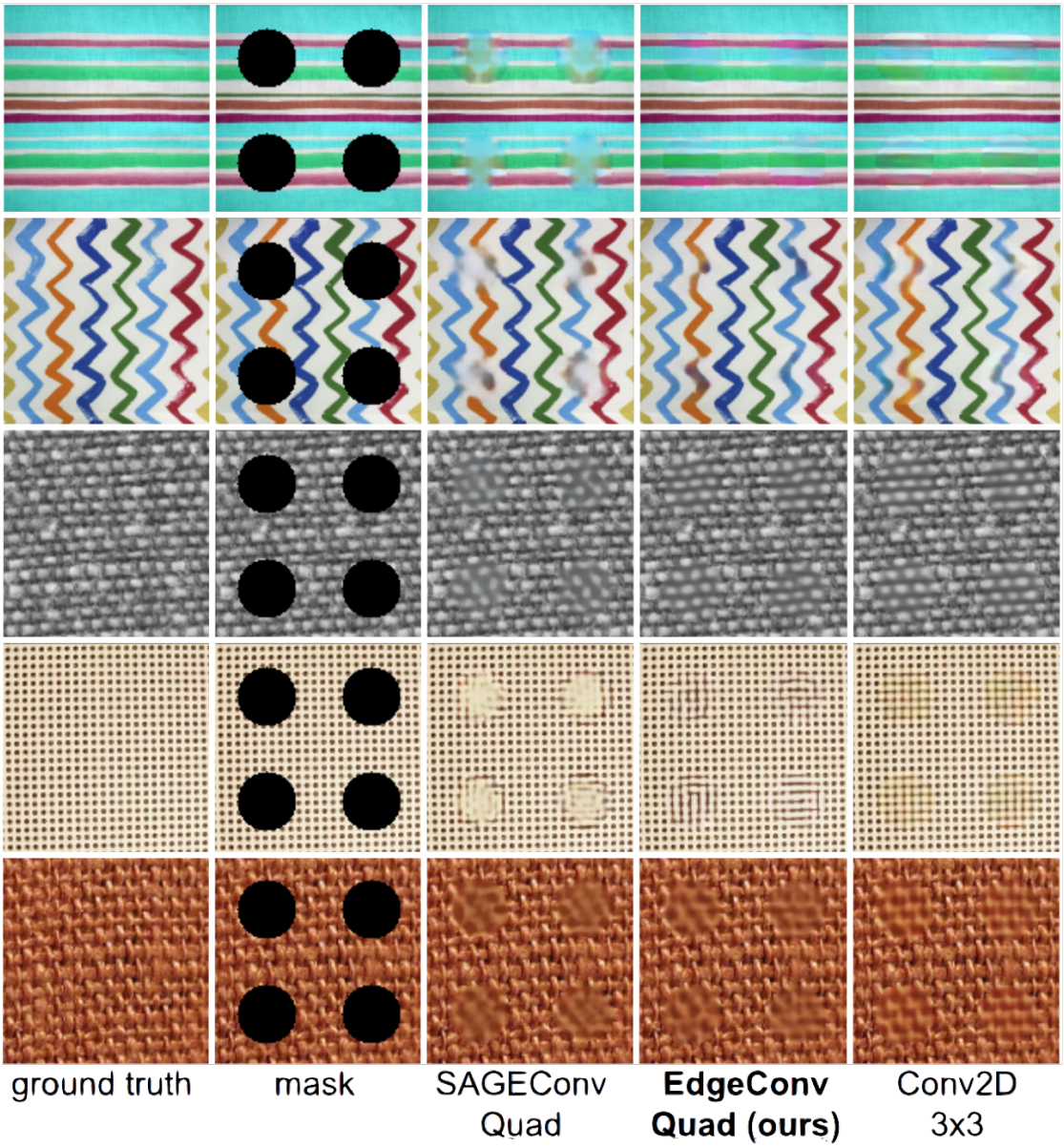


Figure 5.3: **Image Inpainting with Graph Convs**. Results of SAGEConv and EdgeConv at the image inpainting task by representing pixels as vertices and neighborhoods as quad edges. Conv2D results given as pixel-based comparison.

In a final attempt to quantitatively evaluate surface inpainting results, we leverage the observation that low resolution meshes provided by ScanNet tend to be blurry. Although there isn't any one objective measure of blurriness in an image, they typically have less prominent edges. In our experiments we quantify this by computing the variance in the graph Laplacian $Var(\nabla \boldsymbol{G})$ (called *LapVar*) for each mesh. That is, for each mesh we compute the Laplacian over vertex color values to characterize edges on the surface. Then we compute the variance in these edges with the expectation that stronger edges will have a larger variance.

5.2.3 Surface Inpainting Results

We evaluate STINet on two ScanNet scenes. Figures 5.4 and 5.5 show a couch and an office with missing surface texture and their inpainted results. In figure 5.4 all masked regions are inpainted with matching color palettes that blend with their neighborhood and colors do not blur between surfaces. While the inpainted region on the couch does not match perfectly to the existing surface texture, it does resemble vague stripes and it follows the contour of the couch interior.

Figure 5.4 demonstrates the ability of STINet to utilize surface geometry for color prediction. The predicted texture for the masks that are superimposed on each door contains sharp boundaries where the yellow wall meets the brown doorframe. At this point the surface geometry curves into the door which is SCINet recognizes and establishes as a boundary.

Some surfaces such as the carpet in figure 5.4 remain a challenge to inpaint since the geodesic neighborhood containing the rest of the carpet is occluded by the chair. Since STINet is capable of inpainting high quality texture on flat surfaces such as images, we believe that lower quality performance at surface texture inpainting is due primarily to the complexity of 3D data and not the representative ability of graph convolutions.

We provide additional 3D Scene Inpainting results in Appendix 8.1.

Figure 5.4: **Surface Inpainting with Graph Convs**. From top to bottom: Ground truth, mask and prediction. The expanded windows highlight texture inpainting results.

Figure 5.5: **Surface Inpainting with Graph Convs**. From top to bottom: Ground truth, mask and prediction. The expanded windows highlight STINet's ability to follow geometric contours when inpainting.

Figure 5.6: **Effect of Spatial Inputs**. An empty region (left) is inpainted by two different networks. The model that only trains on color (middle) does not easily change color at spatial boundaries. In contrast, the model that additionally trains on normals and position (right) creates sharper color at the transition from table to wall.

Method	ℓ_1 loss ↓	PSNR ↑	LapVar ↑
EdgeConv no pos/norm	5.84%	20.04	.2370
EdgeConv no pos	5.62%	20.44	.2376
EdgeConv no norm	5.61%	20.45	.2371
EdgeConv (ours)	**5.57%**	**20.51**	**.2376**

Table 5.2: Results of STINet with EdgeConv for every combination of spatial inputs.

5.2.4 Ablation: Effect of Spatial Inputs

Since intuition tells us that spatial information helps STINet inpaint surface texture, we attempt to quantify how much it contributes to inpainting quality. Table 5.2 shows the results of four different versions of STINet, each trained on a combination of the two spatial inputs *positions* and *normals*. As expected STINet with all inputs achieves the top score in all metrics while STINet without positions or normals scores the lowest. When two different objects meet there is often a sudden change in the surface curvature such as the table meeting a wall in figure 5.6. Without any information about the position and orientation of the surface, STINet cannot confidently stop one color and start another at any point. The result is that the colors of both objects are blurred together at the boundary.

5.2.5 Ablation: Alternative Graph Convolutions

Finally, we evaluate our network using alternative graph convolution operations in table 5.3 to motivate our design choices for EdgeConv. Without translation invariance our network is forced to learn from absolute positional features that encode a graph's global position and orientation. At the input of STINet we permit translation invariance by omitting the absolute term in EdgeConv. Similarly, while SAGEConv is resilient to varying numbers of neighbors, it does not model the local neighborhood with respect to spatial features. Recall that EdgeConv evaluates the difference across edges of a graph which naturally models a local Euclidean neighborhood.

Method	ℓ_1 loss ↓	PSNR ↑	LapVar ↑
SAGEConv	6.03%	20.05	.2363
EdgeConv no translation inv	5.68%	20.40	.2370
EdgeConv (ours)	**5.57%**	**20.51**	**.2376**

Table 5.3: Evaluation of graph convolution operators in STINet

6 Limitations and Future Work

6.1 Improvements

Although STINet produces promising texturization results on two-dimensional manifolds, there are several challenging problems induced by the 3D domain that STINet does not consider. In this section we discuss some of these limitations and suggest approaches to solve them.

6.1.1 Addition of Euclidean Space Convolutions

Throughout this work we define graph convolutions over the geodesic neighborhood of a mesh surface. This formulation does not utilize the semantic meaning of two points on a surface that have a large geodesic distance but are close together in 3D Euclidean space.

For example, although two points on each handle of an armchair are distant in geodesic space (shortest path routes through the chair's backrest) the texture of one arm is extremely informative about the texture of the other. Additional graph edges (or a similar construction) could be added to represent the Euclidean relationship between surface parts.

6.1.2 Contextual Attention

Unlike images it is often extremely difficult to have a receptive field cover the entire surface of a medium-sized 3D mesh like that of a ScanNet scene. Contextually relevant surface information is often too distant in geodesic space to be useful. For example, STINet is unable to leverage wallpaper to inpaint an identical wall on the opposite side of a room.

Recent work in image inpainting [45] addresses this issue with a contextual attention layer that computes similarities between distant image patches and inpaints missing regions with patches thought to be the most similar. Extending a contextual attention

layer to graph-based inpainting could potentially help expand the very limited receptive field on mesh surfaces.

6.1.3 Remapping Surface to a Regular Grid

While it may seem natural to use the edge-vertex graph structure of a mesh to represent surface texture, mesh vertices and edges actually encode surface topology and not the connectivity of texels that make up its texture. This causes STINet to lose vital surface geometry information during mesh simplification.

Surface texture is represented more naturally as texels in an approximately-regular grid structure (which is also a graph) that is imposed on the mesh surface. This way texture is decoupled from mesh topology and more specialized convolution and pooling operations that work directly on edge-connected texels are possible.

6.2 Novel Research Directions

STINet is an early work in the young research field of surface textue inpainting. We recommend a number of viable extensions that future researchers may pursue.

6.2.1 Conditional Surface Texture Synthesis using Semantic Labels

Similar to the task of inpainting is image synthesis from semantic labels [27]. Given an image with rough hand-drawn regions of semantic labels such as sky, mountains and grass, conditional image synthesis aims to generate realistic photos using the labeled image as a prior. Applications naturally extend to mesh surfaces such as in 3D modeling software that could texturize a mesh given sketched labels from a user. Further, many 3D datasets with semantic labels or classes already exist such as ScanNet [6], 3D-Front [11] and ShapeNet [4].

6.2.2 Inpainting Other Surface Properties

A curious insight is that inpainting is not limited to color. Mesh surfaces have a host of possible properties that one might desire to fill such as per-pixel geometry, illumination or even position and normals. A particularly interesting example is to inpaint Neural Textures [36] on a mesh surface; themselves being learned features that are used to render surfaces in higher quality. This could potentially help extend photo-realistic novel view synthesis to extreme angles.

7 Conclusion

In this thesis we developed the Surface Texture Inpainting Network (STINet) to generate texture in the missing regions of partially textured 3D meshes. STINet uses graph convolutions in a residual encoder-decoder network and pools features directly on the surface of simplified versions of the mesh. Through experimentation on 2D images we showed that STINet achieves inpainting results that are comparable to classic 2D convolutions. Then we translated the domain to 3D where STINet successfully utilized surface geometry to improve its inpainting results. We believe that STINet shows promise in the new domain of surface texture inpainting and that it serves as a good benchmark for future work.

8 Appendix

8.1 Additional 3D Scene Inpainting Results

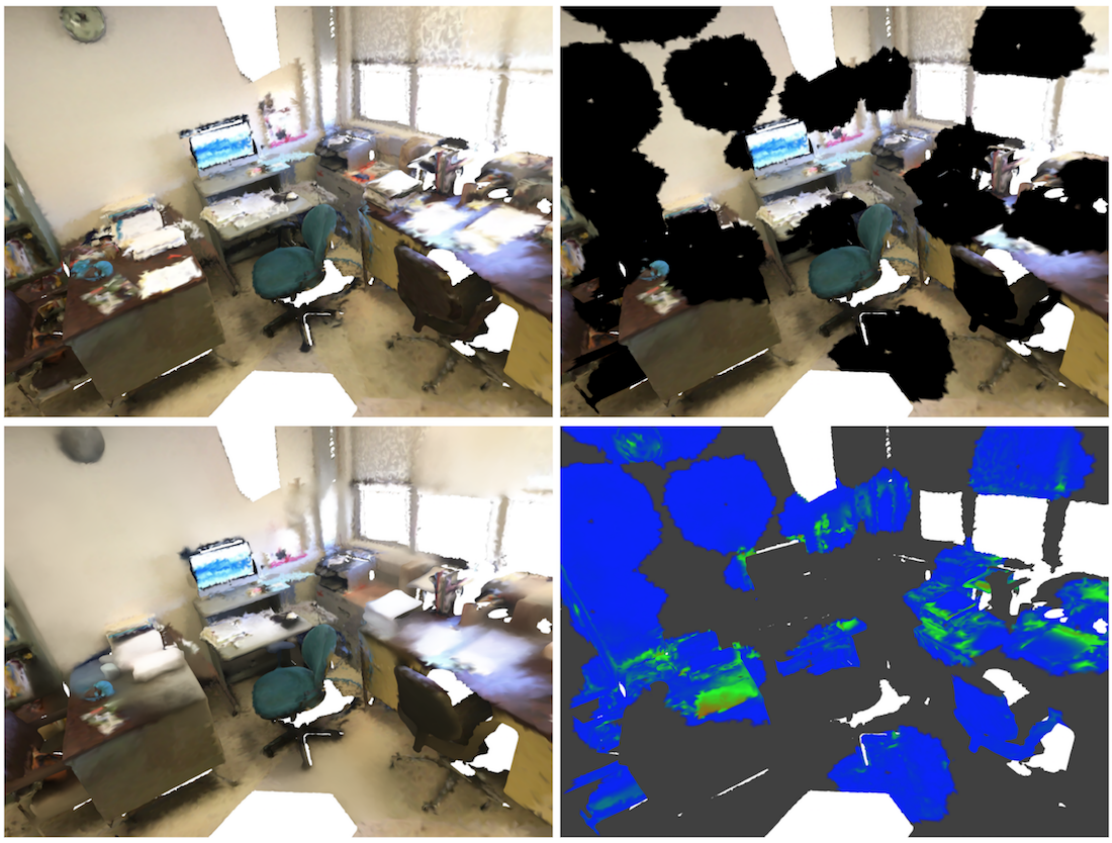


Figure 8.1: **Error Heatmap**. Starting from the top left and going left to right: (1) Original surface texture of a 3D office reconstruction. (2) Mask superimposed on the scene. (3) Predicted surface texture in masked regions. (4) ℓ_1 error in prediction at each vertex. Blue is 0% error, green is 50% and red is 100%.

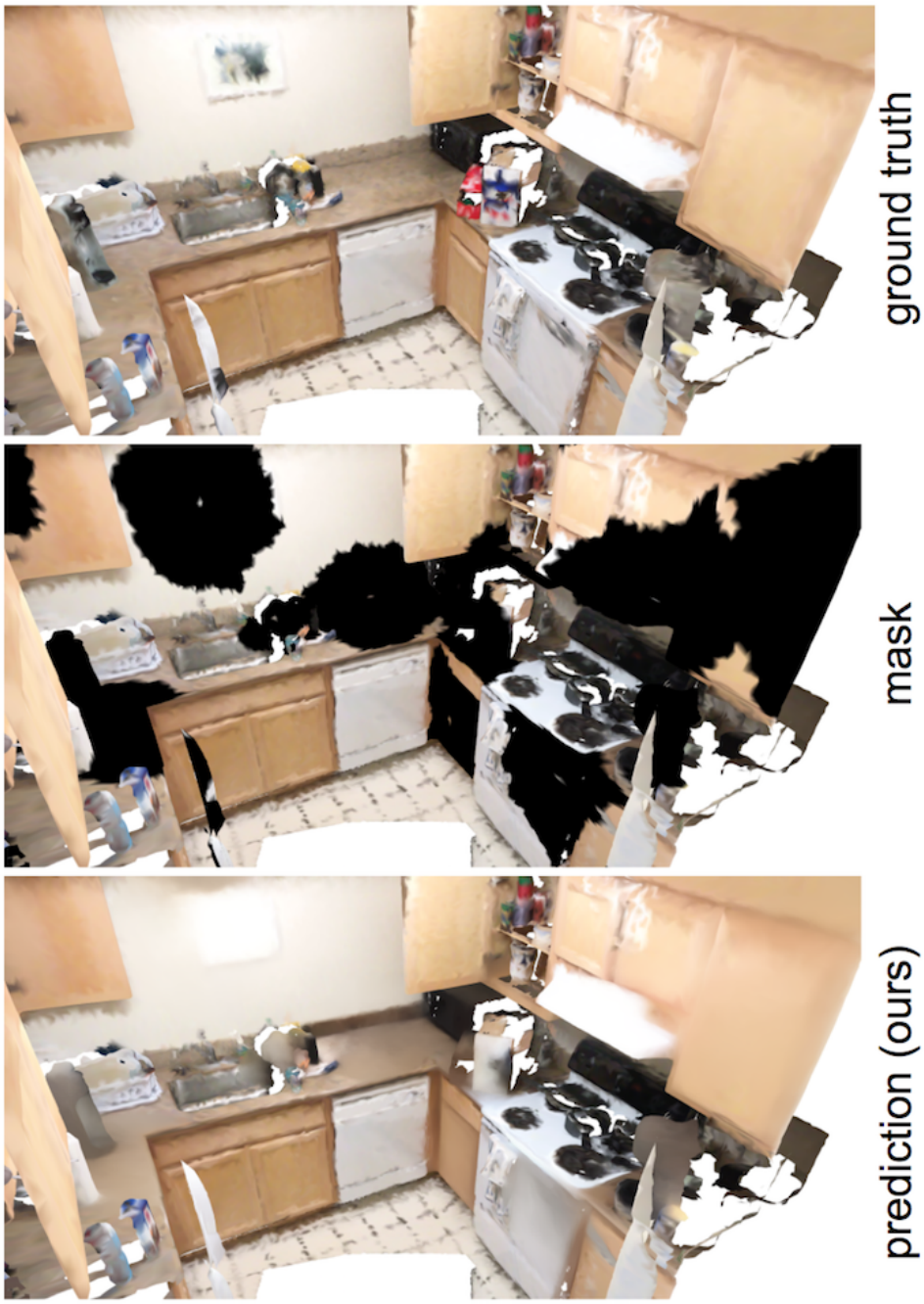


Figure 8.2: **Bathroom Mesh**. Ground truth, mask and predicted surface texture for a 3D reconstruction of a bathroom.

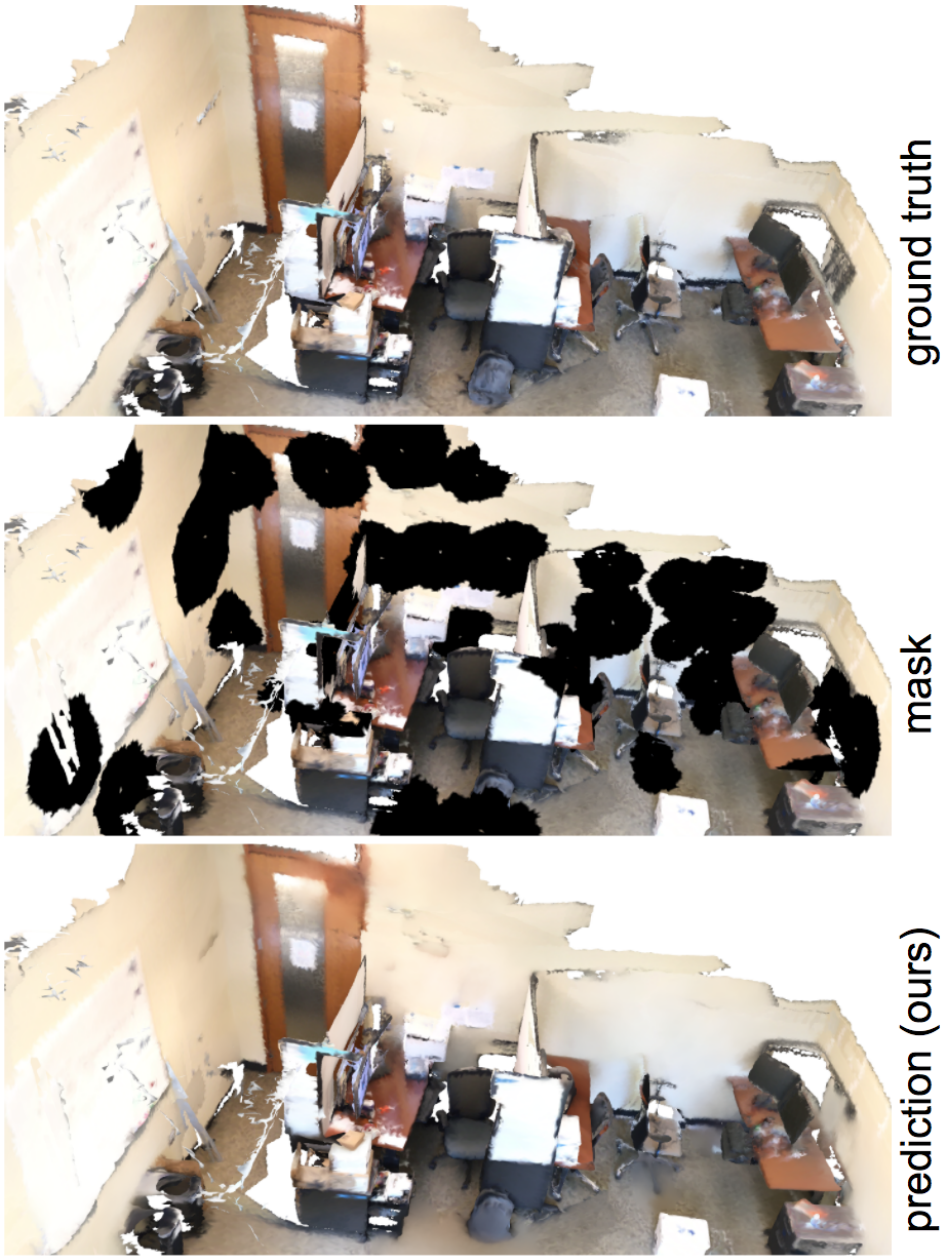


Figure 8.3: **Office Space Mesh**. Ground truth, mask and predicted surface texture for a 3D reconstruction of a big office.

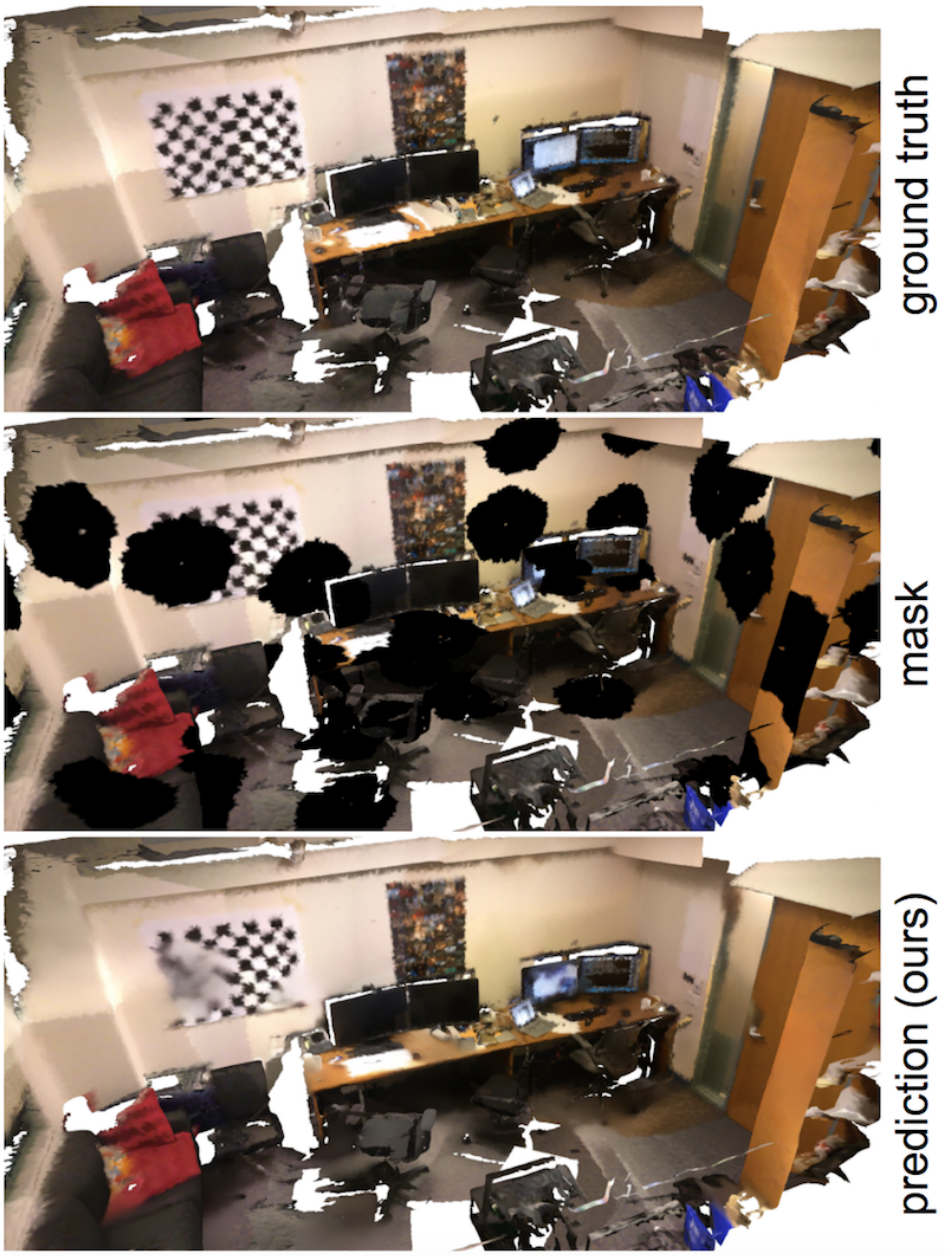


Figure 8.4: **Computer Room Mesh**. Ground truth, mask and predicted surface texture for a 3D reconstruction of a computer room.

List of Figures

List of Tables

Bibliography

[1] E. P. Alla Sheffer and K. Rose. "Mesh Parameterization Methods and Their Applications." In: *Foundations and Trends® in Computer Graphics and Vision: Vol. 2: No. 2, pp 105-171* (2007).

[2] D. Boscaini, J. Masci, E. Rodolà, and M. M. Bronstein. *Learning shape correspondence with anisotropic convolutional neural networks*. 2016. arXiv: 1605.06437 [cs.CV].

[3] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst. "Geometric Deep Learning: Going beyond Euclidean data." In: *IEEE Signal Processing Magazine* 34.4 (July 2017), pp. 18–42. ISSN: 1558-0792. DOI: 10.1109/msp.2017.2693418.

[4] 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*. 2015. arXiv: 1512.03012 [cs.GR].

[5] D.-A. Clevert, T. Unterthiner, and S. Hochreiter. *Fast and Accurate Deep Network Learning by Exponential Linear Units (ELUs)*. 2016. arXiv: 1511.07289 [cs.LG].

[6] A. Dai, A. X. Chang, M. Savva, M. Halber, T. A. Funkhouser, and M. Nießner. "ScanNet: Richly-annotated 3D Reconstructions of Indoor Scenes." In: *CoRR* abs/1702.04405 (2017). arXiv: 1702.04405.

[7] A. Dai, C. Diller, and M. Nießner. "SG-NN: Sparse Generative Neural Networks for Self-Supervised Scene Completion of RGB-D Scans." In: *CoRR* abs/1912.00036 (2019). arXiv: 1912.00036.

[8] M. Defferrard, X. Bresson, and P. Vandergheynst. *Convolutional Neural Networks on Graphs with Fast Localized Spectral Filtering*. 2017. arXiv: 1606.09375 [cs.LG].

[9] U. Demir and G. B. Ünal. "Patch-Based Image Inpainting with Generative Adversarial Networks." In: *CoRR* abs/1803.07422 (2018). arXiv: 1803.07422.

[10] A. S. Dong-Chen. "Colored Brodatz Texture." In:

[11] H. Fu, B. Cai, L. Gao, L.-X. Zhang, J. Wang, C. Li, Q. Zeng, C. Sun, R. Jia, B. Zhao, et al. "3d-front: 3d furnished rooms with layouts and semantics." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2021, pp. 10933–10942.

[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*. SIGGRAPH '97. USA: ACM Press/Addison-Wesley Publishing Co., 1997, pp. 209–216. ISBN: 0897918967. DOI: 10.1145/258734.258849.

[13] I. J. Goodfellow, J. Pouget-Abadie, M. Mirza, B. Xu, D. Warde-Farley, S. Ozair, A. Courville, and Y. Bengio. *Generative Adversarial Networks*. 2014. arXiv: 1406.2661 [stat.ML].

[14] B. Graham and L. van der Maaten. *Submanifold Sparse Convolutional Networks*. 2017. arXiv: 1706.01307 [cs.NE].

[15] W. L. Hamilton, R. Ying, and J. Leskovec. "Inductive Representation Learning on Large Graphs." In: *CoRR* abs/1706.02216 (2017). arXiv: 1706.02216.

[16] K. He, X. Zhang, S. Ren, and J. Sun. "Deep Residual Learning for Image Recognition." In: *CoRR* abs/1512.03385 (2015). arXiv: 1512.03385.

[17] J. Huang, H. Zhang, L. Yi, T. A. Funkhouser, M. Nießner, and L. J. Guibas. "TextureNet: Consistent Local Parametrizations for Learning from High-Resolution Signals on Meshes." In: *CoRR* abs/1812.00020 (2018). arXiv: 1812.00020.

[18] J. Huang, Y. Zhou, M. Niessner, J. R. Shewchuk, and L. J. Guibas. "QuadriFlow: A Scalable and Robust Method for Quadrangulation." In: *Computer Graphics Forum* 37.5 (2018), pp. 147–160. DOI: https://doi.org/10.1111/cgf.13498. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1111/cgf.13498.

[19] S. Iizuka, E. Simo-Serra, and H. Ishikawa. "Globally and locally consistent image completion." In: *ACM Transactions on Graphics (TOG)* 36 (2017), pp. 1–14.

[20] P. Isola, J. Zhu, T. Zhou, and A. A. Efros. "Image-to-Image Translation with Conditional Adversarial Networks." In: *CoRR* abs/1611.07004 (2016). arXiv: 1611.07004.

[21] T. N. Kipf and M. Welling. "Semi-Supervised Classification with Graph Convolutional Networks." In: *CoRR* abs/1609.02907 (2016). arXiv: 1609.02907.

[22] G. Liu, F. A. Reda, K. J. Shih, T. Wang, A. Tao, and B. Catanzaro. "Image Inpainting for Irregular Holes Using Partial Convolutions." In: *CoRR* abs/1804.07723 (2018). arXiv: 1804.07723.

[23] W. Luo, Y. Li, R. Urtasun, and R. S. Zemel. "Understanding the Effective Receptive Field in Deep Convolutional Neural Networks." In: *CoRR* abs/1701.04128 (2017). arXiv: 1701.04128.

[24] B. C. Mario Fritz Eric Hayman. "KTH-TIPS: Textures under varying Illumination, Pose and Scale." In: 2006.

[25] J. Masci, D. Boscaini, M. M. Bronstein, and P. Vandergheynst. "ShapeNet: Convolutional Neural Networks on Non-Euclidean Manifolds." In: *CoRR* abs/1501.06297 (2015). arXiv: 1501.06297.

[26] H. Pan, S. Liu, Y. Liu, and X. Tong. "Convolutional Neural Networks on 3D Surfaces Using Parallel Frames." In: *CoRR* abs/1808.04952 (2018). arXiv: 1808.04952.

[27] T. Park, M.-Y. Liu, T.-C. Wang, and J.-Y. Zhu. *Semantic Image Synthesis with Spatially-Adaptive Normalization*. 2019. arXiv: 1903.07291 [cs.CV].

[28] G. Parmar, R. Zhang, and J. Zhu. "On Buggy Resizing Libraries and Surprising Subtleties in FID Calculation." In: *CoRR* abs/2104.11222 (2021). arXiv: 2104.11222.

[29] P. M. Roland Kwitt. "Salzburg Texture Image Database." In:

[30] O. Ronneberger, P. Fischer, and T. Brox. *U-Net: Convolutional Networks for Biomedical Image Segmentation*. 2015. arXiv: 1505.04597 [cs.CV].

[31] J. Schult, F. Engelmann, T. Kontogianni, and B. Leibe. "DualConvMesh-Net: Joint Geodesic and Euclidean Convolutions on 3D Meshes." In: *CoRR* abs/2004.01002 (2020). arXiv: 2004.01002.

[32] K. Simonyan and A. Zisserman. *Very Deep Convolutional Networks for Large-Scale Image Recognition*. 2015. arXiv: 1409.1556 [cs.CV].

[33] D. Sisson. "Pixar One Twenty Eight." In: 2018.

[34] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. *Going Deeper with Convolutions*. 2014. arXiv: 1409.4842 [cs.CV].

[35] M. Tatarchenko, J. Park, V. Koltun, and Q.-Y. Zhou. "Tangent Convolutions for Dense Prediction in 3D." In: *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2018, pp. 3887–3896. DOI: 10.1109/CVPR.2018.00409.

[36] J. Thies, M. Zollhöfer, and M. Nießner. *Deferred Neural Rendering: Image Synthesis using Neural Textures*. 2019. arXiv: 1904.12356 [cs.CV].

[37] J. Thies, M. Zollhöfer, M. Nießner, L. Valgaerts, M. Stamminger, and C. Theobalt. "Real-Time Expression Transfer for Facial Reenactment." In: *ACM Trans. Graph.* 34.6 (Oct. 2015). ISSN: 0730-0301. DOI: 10.1145/2816795.2818056.

[38] J. Thies, M. Zollhöfer, M. Stamminger, C. Theobalt, and M. Nießner. *Face2Face: Real-time Face Capture and Reenactment of RGB Videos*. 2020. arXiv: 2007.14808 [cs.CV].

[39] A. Vaxman, M. Campen, O. Diamanti, D. Bommes, K. Hildebrandt, M. B.-C. Technion, and D. Panozzo. "Directional Field Synthesis, Design, and Processing." In: *ACM SIGGRAPH 2017 Courses*. SIGGRAPH '17. Los Angeles, California: Association for Computing Machinery, 2017. ISBN: 9781450350143. DOI: 10.1145/3084873.3084921.

[40] Vedaldi. "Describing Textures in the Wild." In: 2014.

[41] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio. *Graph Attention Networks*. 2018. arXiv: 1710.10903 [stat.ML].

[42] Y. Wang, Y. Sun, Z. Liu, S. E. Sarma, M. M. Bronstein, and J. M. Solomon. "Dynamic Graph CNN for Learning on Point Clouds." In: *CoRR* abs/1801.07829 (2018). arXiv: 1801.07829.

[43] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu. "A Comprehensive Survey on Graph Neural Networks." In: *CoRR* abs/1901.00596 (2019). arXiv: 1901.00596.

[44] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. Huang. *Free-Form Image Inpainting with Gated Convolution*. 2019. arXiv: 1806.03589 [cs.CV].

[45] J. Yu, Z. Lin, J. Yang, X. Shen, X. Lu, and T. S. Huang. "Generative Image Inpainting with Contextual Attention." In: *CoRR* abs/1801.07892 (2018). arXiv: 1801.07892.

[46] R. Zhang, P. Isola, A. A. Efros, E. Shechtman, and O. Wang. "The Unreasonable Effectiveness of Deep Features as a Perceptual Metric." In: *CoRR* abs/1801.03924 (2018). arXiv: 1801.03924.