Vrije Universiteit Amsterdam

Centrum Wiskunde & Informatica Amsterdam

Master Thesis

Temporal Interpolation of Dynamic Point Clouds using Convolutional Neural Networks

Author: Jelmer Nicolaas Mulder[1,2] (2526631)

1st supervisor:	Prof.dr. D.C.A. Bulterman[1,2]
daily supervisor:	Dr. P.S. César Garcia[2,3]
	Dr. F. De Simone[2,4]
2nd reader:	Drs. A.J. Jansen[2]

[1]Vrije Universiteit, Amsterdam, The Netherlands
[2]Centrum Wiskunde & Informatica, Amsterdam, The Netherlands
[3]Delft University of Technology, Delft, The Netherlands
[4]École Polytechnique Fédérale de Lausanne, Lausanne, Switzerland

A thesis submitted in fulfillment of the requirements for
Master of Science degree in Parallel and Distributed Computer Systems

August 19, 2019

Abstract

In recent years there has been an increased interest in Cross Reality (XR). One representation that sees frequent use in XR is that of a point cloud, a data structure that models volumetric visual data as a set of individual points in space. Points clouds are voluminous in size, and thus require high bandwidth to transmit. In practise this means that concessions have to be made either in spatial- or temporal resolution. In this thesis we propose a temporal interpolation architecture capable of increasing the temporal resolution of dynamic point clouds. With this technique, dynamic point clouds can be transmitted in a lower temporal resolution, after which a higher temporal resolution can be obtained by performing the interpolation on the receiving side.

Our interpolation architecture works by first downsampling the point clouds to a lower spatial resolution, then estimating scene flow, and finally upsampling the result back to the original spatial resolution. To improve the smoothness of the interpolation result, we additionally apply a novel technique called *neighbour snapping*. In order to estimate scene flow, we use a newly designed neural network architecture. To be able to train and evaluate this network, we have created a synthetic point cloud data set of animated human bodies.

We evaluate our architecture through a small-scale user study and with objective quality metrics. Existing objective quality metrics for point clouds are known to have poor correlation with user perception. Our findings confirm this, as the metrics we report correlate poorly with themselves and with the results from the user study. The user study shows that on average, participants prefer the temporally interpolated sequences generated by our architecture over current state-of-the-art or sequences that have not been interpolated.

We have shown that our approach is capable of performing temporal interpolation on synthetic dynamic point clouds. Further interesting research directions include making the architecture run in real-time, improving performance when the motion between consecutive frames is large, and training and evaluating the system using a real-world data set.

Contents

List of Figures

List of Tables

1

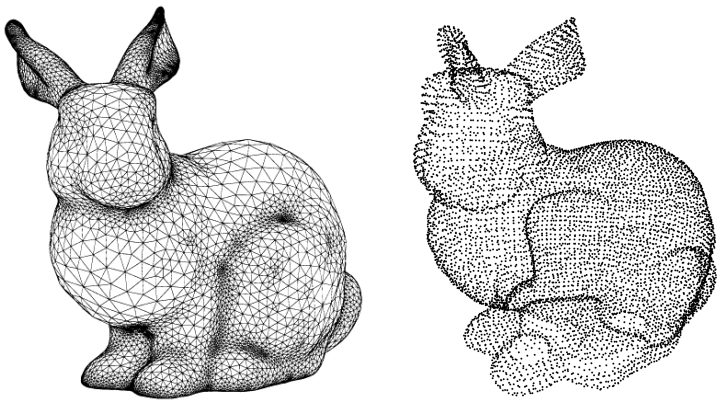
Introduction

In recent years there has been an increased interest in Cross Reality (XR). One representation that sees frequent use in XR technologies is that of a point cloud (PC), a data structure that models volumetric visual data as a set of individual points in space. Point clouds are voluminous in size, and thus require high bandwidth to transmit. In practise this means that concessions have to be made either in spatial resolution or in temporal resolution. In this thesis we propose a temporal interpolation architecture capable of increasing the temporal resolution of a point cloud sequence. With this technique, point cloud sequences can be transmitted in a lower temporal resolution, after which a higher temporal resolution can be obtained by performing the interpolation on the receiving side.

In Section 1.1 we first provide some background about points clouds. We discuss what point clouds are, why they are useful, how they are captured, what challenges come with their use, and what benefits can be had from applying temporal interpolation on them. In Section 1.2 we more formally identify the goals of this research, by detailing our Research Objectives. In Section 1.3 we discuss the methodology that will be used to address these Research Objectives. In Section 1.4 we then provide an overview of the contributions of this work. Lastly, in Section 1.5, we explain the structure of the rest of this thesis.

1.1 Background & context

Cross Reality (XR) is a collective term for a range of cross reality techniques, such as Augmented Reality (AR), Virtual Reality (VR), and Mixed Reality (MR). These immersive techniques create new forms of reality by bringing digital objects into the physical world, or bringing physical objects into the digital world. Of particular interest is creating 3D digital representations of the physical world, allowing end-users to freely navigate this



(a) Polygon mesh **(b)** Point cloud

Figure 1.1: Comparison of polygon mesh and point cloud. Both renderings represent the Stanford bunny (2).

representation with Six Degrees of Freedom (6DoF). Cross reality techniques have seen an increased popularity in recent years, which has created a demand for efficient representation of dynamic volumetric visual data. Currently, the two most frequently used representations are polygonal meshes and point clouds (1). Figure 1.1 shows the Stanford bunny (2) represented both as mesh and point cloud.

In a polygon mesh, objects are modelled as a collection of faces. These faces are commonly triangles, but can also be more-sided polygons. While polygon meshes can be used as compact representations of dense surfaces, they have problems representing noisy geometry. Additionally, reconstructing meshes is a computationally intensive process, which limits the use of meshes in real-time applications. A point cloud models objects as a collection of points, where each point has a spatial coordinate, and optionally additional information like color, reflectivity, or surface normals. Points clouds do not require time-consuming reconstruction, which makes them more suitable for real-time systems. On the other hand, point clouds typically cannot model surfaces as compactly as meshes, requiring more space. Both meshes and point clouds can be used to model dynamic scenes, simply by creating a sequence out of the still captures. In the case of meshes this would be called a *time variant mesh* (3), in the case of point clouds this would be called a *dynamic point cloud* (4). In both cases we will refer to an individual still capture as a *frame* of that sequence. The *frame rate* is then defined as how many frames the sequence contains per second. In this thesis we will be working with dynamic point clouds.

Point clouds are not just used in XR, but in various fields. For example, they see use in gaming (5), free viewpoint videos (6), cultural heritage (7), automated driving (8), real-

(a) LiDAR scanner (12) **(b)** Microsoft Kinect v2 (13) **(c)** Intel RealSense D435 (14)

Figure 1.2: Examples of point cloud capture devices

time immersive telepresence (9, 10), and more (11). The two main acquisition techniques for point clouds are the use of LiDAR scanners and depth cameras, each being more suited for different use cases.

LiDAR is a 3D scanning technique that measures the distance to an object by emitting laser pulses and measuring the reflected pulses. Distance to the object can then be estimated based on differences in laser return times and wave lengths. LiDAR is flexible to lighting conditions, and little computation is required to process the measurements. On the other hand, LiDAR systems only output shape information, and thus do not capture any color information. Additionally, LiDAR systems are at this moment expensive, and the achievable resolution and frame rate are typically relatively low. LiDAR is commonly used in automatic driving solutions and for scanning of geographical regions. An example of a LiDAR scanner is shown in Figure 1.2a.

An alternative capture method is to use depth cameras. Some of the advantages of these cameras over LiDAR are that they are cheap, they provide color information, and they often support a higher resolution and frame rate. Contrarily, measurements from depth cameras usually require a higher degree of computation, and output is highly dependent on the lighting situations. In recent years an increasing number of affordable depth camera solutions have been introduced to the market, making them more accessible to consumers, and positioning them as interesting enablers of XR applications. Examples of such affordable depths cameras are the Microsoft Kinect (5) (Figure 1.2b) and Intel RealSense (15) (Figure 1.2c). Figure 1.3 shows an example of a consumer-grade capture setup made out of depth cameras.

The dynamic point cloud sequences generated by these capture setups can grow large in size. Let us look for example at the sequence named `Loot` from the 8iVFB data set (16)

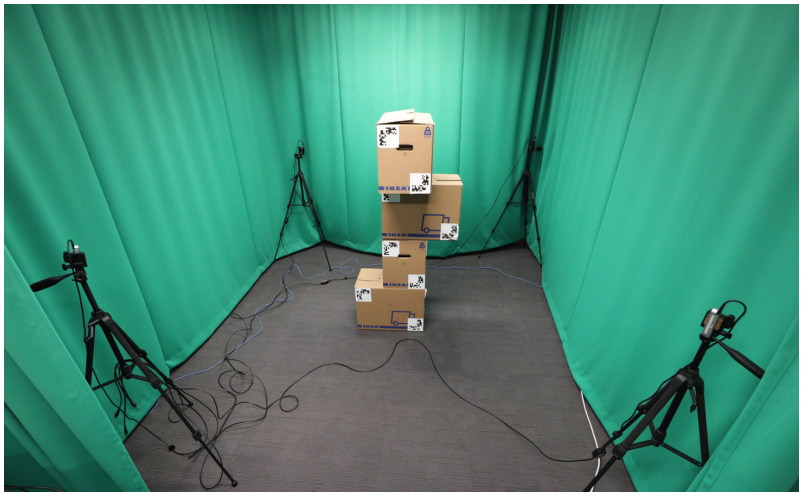


Figure 1.3: Consumer-grade capture setup This setup consists of four Intel RealSense D415 depth cameras mounted on tripods. The room is fitted with a green screen, to allow the background to be easily removed. In this picture, the cameras a capturing a stack of marked boxes, to facilitate the alignment process.

(more information in Section 2.8). Figure 1.4 shows a rendering of one frame of this sequence. In total, the sequence is 10 seconds long, contains 30 frames per second with approximately 793 000 points per frame, and takes 5.1 GB of storage space for the entire sequence, or 522 MBps to stream it in real-time.

For certain XR applications, for example real-time immersive telepresence, we might be interested in streaming dynamic point clouds in real-time from one machine to another over the internet. Due to the large size dynamic point clouds can reach, bandwidth can become a bottleneck when using consumer hardware. Let us consider for example the consumer-grade capture setup shown in Figure 1.3, consisting of four Intel RealSense D415 depth cameras. Even in this modest setup, we run into bottlenecks both in the interface between camera and host, and in the network between the two hosts.

First, there is the channel between the capture device and the host. This might be USB 3.0 for instance, as is the case with the Intel RealSense cameras. USB 3.0 theoretically supports 640 MBps throughput, but in practise not more than 410 MBps, or even not more than 154 MBps if multiple devices are connected simultaneously (17). On an Intel RealSense D415 camera using the USB 3.0 interface, this imposes a maximum achievable resolution of 1280x720 at 30 frames per second for single camera setups, or 840x480 at 30 frames per second per camera for our four camera setup (17).

The internet could also become a bottleneck when attempting to stream a dynamic point cloud sequence from one host to a remote host, especially when real-time stream-

Figure 1.4: Example of high resolution point cloud. This Figure shows a rendering of a frame of the `Loot` sequence from the 8iVFB data set. It consists of approximately 793 000 points, and is 17 MB large.

ing is required. Internet transfer speeds vary largely by geographic region, so general conclusions cannot be easily drawn. However, in 2018 the average broadband download speed was 59.45 MBps (18), far less than would be required to stream captures from the aforementioned four camera capture setup or a sequence such as `Loot`.

Due to this limited bandwidth, trade-offs have to be made between spatial resolution (the amount of points in each frame), and temporal resolution (the frame rate). Low frame rate sequences are generally considered to be less pleasant to look at than higher frame rate sequences (19), and jerkiness is often perceived in them. Thus, simply reducing the temporal resolution is likely to have a strong negative effect on the Quality of Experience (QoE) (19).

To palliate having to make such a trade-off, we can leverage temporal interpolation. Temporal interpolation is a technique to increase the frame rate of a temporal sequence, which is done by constructing frames in between existing frames. It then becomes possible to transmit or capture a point cloud sequence at a relatively low frame rate, to then perform the temporal interpolation on the other end of the channel in order to reconstruct the desirable frame rate. Such reconstruction can drastically reduce the bandwidth

requirements of streaming and capturing dynamic point cloud sequences, allowing this bandwidth to be used for transmitting higher spatial resolution.

Temporal interpolation can also be applied to 2D video, in which case it is sometimes also referred to as *motion interpolation* or *video frame interpolation* (20). Similarly to temporal interpolation on point clouds, the goal of video frame interpolation is to generate new frames in between existing frames of a sequence, only for 2D videos instead of dynamic point clouds. Typically this is done by iterating over all pairs of successive frames, for each pair creating one or more frames that will fit in between (21). This way, the frame rate of the source video can be increased by a factor 2 or more, depending on the amount of frames interpolated between each pair of frames (see to Section 2.3 for more details on video frame interpolation). Because of the fundamental differences between 2D videos and dynamic point clouds, existing interpolation techniques cannot easily be extended to work on dynamic point clouds. In this thesis we will study new techniques, in order to be able to reap the benefits of temporal interpolation also for dynamic point clouds.

1.2 Problem statement

The goal of the research presented in this thesis is to perform temporal interpolation of dynamic point clouds. The objective is to generate new frames in between existing frames in order to increase the frame rate. More formally we can define this problem as follows: given a pair of frames (f_i, f_{i+2}) in a dynamic point cloud, we want to generate the frame f'_{i+1}, keeping motion consistency. If a ground-truth is available, a triplet of frames (f_i, f_{i+1}, f_{i+2}), the algorithm should minimize the *distance* between the reference frame f_{i+1} and the reconstructed frame f'_{i+1}. The problem is thus

$$\arg\min_{f'_{i+1}} D(f_{i+1}, f'_{i+1}) \tag{1.1}$$

where $D(p_i, p_j)$ is a distance metric that calculates some notion of distance between point clouds p_i and p_j. We will discuss various distance metrics that can be used in Section 2.7, however the intuition is that frames that appear highly similar to a human observer should be assigned a low distance score, while frames that appear different should be assigned a high distance score.

Existing 2D interpolation techniques generally rely on the grid structure exhibited by the video frames, and thus cannot be applied to point clouds, since the points are inherently unstructured (22). On the other hand, when interpolating point clouds we are free to place the points at any position (the point do not have to fall on a grid), thus it allows for more

flexibility. In any case, traditional 2D interpolation techniques cannot straightforwardly be applied to point clouds, and as such new methods have to be researched.

In recent years neural networks have brought about a revolution in the area of computer vision, and they have now become the dominant approach for almost all recognition and detection tasks (23). Neural networks have recently also been used for video frame interpolation, which is the task of performing temporal interpolation on 2D videos (21, 24, 25, 26). In this task they make up also the state-of-the-art. Moreover, neural networks have also been used with great success in various recognition tasks on point clouds (27, 28, 29, 30, 31). Given all these successes, we aim to utilize neural networks in order to perform temporal interpolation of dynamic point clouds. This leads us to our first main Research Objective.

> **Research Objective 1** *Design an architecture capable of performing temporal interpolation on dynamic point clouds.*

Such an architecture would take as input a dynamic point cloud sequence, and output a sequence with a higher frame rate. It would be desirable for this architecture to be able to process dynamic point clouds of any spatial resolution, that is, with any number of points. This is challenging for neural networks for two reasons. First, most neural network architectures used for learning on point clouds scale poorly with number of points, due to reliance on techniques such as k-nearest neighbour search (29) or farthest-point sampling (27, 28). Second, certain neural network components might require a static number of points to be used, and require retraining if the number of points changes.

> **Research Objective 1.1** *Design an architecture that can manage the large and variable spatial resolution of dynamic point clouds.*

Most research efforts of applying deep learning to point clouds so far have been directed at tasks such as classification (given a point cloud, assign it one out of a set of labels) and segmentation (assign to each point in a point cloud one out of a set of labels). In such tasks the input consists of one point cloud. Typically features are learned, which are used to assign the label(s). Contrarily, in our task the input consists of two point clouds. At some point in the architecture, information from these two point clouds will have to be mixed, so that the interpolation result can be generated. We will need to design a neural network component capable of this.

> **Research Objective 1.2** *Design a neural network component that can combine information from two temporally sequential point clouds.*

With such a component at hand, the last remaining design challenge is the design of the neural network architecture itself. Here we need to make decisions such as what components do we use, how are they connected, how many filters to use, what kernel size, what activation function, when to apply batch-normalization, and so on.

Research Objective 1.3 *Design a* neural network *architecture capable of performing temporal interpolation of dynamic point clouds.*

Designing the architecture is only half of the challenge, we also need to train it. This brings us to our second main Research Objective.

Research Objective 2 *Train our neural network.*

To train our neural network, we will use a supervised training approach. In order to be able to apply supervised learning, we require two things. First, we need to have a data set of inputs, labeled with the desired ground truth output. Second, we need to define a loss function, which describes for every output generated by the network how close it is to the ground truth output. A low loss function means that the output closely resembles the ground truth, while a high loss function means that they are vastly different. We then apply a variant of the gradient descent algorithm, which will attempt to minimize the loss function by adjusting the weights inside the neural network. The choice of loss function is critical to the performance of the neural network, because it is this loss function that decides what we optimize towards, and thus what the network learns.

Research Objective 2.1 *Develop a suitable loss function to train a neural network to perform temporal interpolation of dynamic point clouds.*

The stochastic gradient descent (SGD) algorithm which we use to train our neural network is an iterative algorithm. Repeatedly, we will input data into the network, evaluate the loss function, and the stochastic gradient descent algorithm will adjust the weights accordingly. In order for this training process to be successful, it is essential to have a large training data set. If the training data set is not sufficiently large, it is likely that the network will *overfit*, meaning that it will perform good only with the training data set, but not with other inputs. Convolutional neural networks (which we will be using) are known to require an especially large amount of training data. The amount of publicly available dynamic point cloud sequences is currently limited (see Section 2.8), thus becoming an extra challenge for training our network.

Research Objective 2.2 *Train a convolutional neural network despite the limited availability of dynamic point cloud data sets.*

1.3 Methodology

We design a neural network architecture that is capable of performing temporal interpolation on dynamic point cloud sequences. In order to train this network we require a large amount of training data. As real-world dynamic point cloud data sets are limited in availability, we create two synthetic data sets of our own in order to train and evaluate the network. The first data set is created by applying a series of rigid transformations to models from the Modelnet data set (32). The second data set is created by applying animations to models of human bodies. Both these data sets are described in more detail in Section 4.1. We evaluate our architecture using the data set of human bodies, as this is the more realistic data set. We split the synthetic data set into a training set, a validation set, and a test set. We then train the network using the training set and the validation set, and evaluate its performance on the test set. We compare our results against the state-of-the-art in temporal interpolation of point clouds. We also compare our results against the non-interpolated input sequences, in order to analyze the benefit of temporal interpolation. Lastly, we also conduct a small-scale user study, for a subjective evaluation of our results.

1.4 Contributions

In order to accomplish the Research Objectives outlined in Section 1.2, this work makes a number of contributions.

> **Research Objective 1.1** *Design an architecture that can manage the large and variable spatial resolution of dynamic point clouds.*

To be able to perform temporal interpolation of point clouds with a high spatial resolution, we have devised a downsampling/upsampling scheme. We downsample the input point clouds to a lower spatial resolution, allowing them to be fed through our interpolation network. We then upsample the interpolation results back to the original spatial resolution. This way, we can bypass the poor scaling properties of our interpolation network, and interpolate even high resolution point clouds.

> **Research Objective 1.2** *Design a neural network component that can combine information from two temporally sequential point clouds.*

We have designed a novel neural network component, called the point matching module. This module can be used to merge information from two separate point clouds, and can this way learn correspondence between points across the two point clouds. From this

correspondence we can make an initial interpolation prediction, which is then refined in later stages of the neural network.

> **Research Objective 1.3** *Design a neural network architecture capable of performing temporal interpolation of dynamic point clouds.*

In addition to our point matching module, we also design a complete neural network architecture, that proves to be capable of performing temporal interpolation of dynamic point clouds.

> **Research Objective 2.1** *Develop a suitable loss function to train a neural network to perform temporal interpolation of dynamic point clouds.*

In this work we examine a number of loss functions to be used to train our neural network. In particular, we share our experiences with the use of point cloud distortion metrics as loss functions, and discuss our limited success with them. Additionally, we show that scene flow L1 loss can be successfully used to train a neural network to perform temporal interpolation of dynamic point clouds.

> **Research Objective 2.2** *Train a convolutional neural network, despite the limited availability of dynamic point cloud data sets.*

In order to train our neural network, despite the limited availability of public dynamic point cloud data sets, we have created a novel synthetic data set. This data set has been created by applying realistic animations to human models. We use this data set to train and evaluate our architecture.

All in all, we have designed and implemented an architecture that can increase the frame rate of existing dynamic point cloud sequences by temporally interpolating them. We have performed a small-scale user study and find that participants on average prefer the dynamic point cloud sequences interpolated using our architecture over sequences interpolated by the current state-of-the-art or sequences that have not been interpolated.

1.5 Structure

The remainder of this thesis is structured as follows. In Chapter 2 we discuss the related work. We first introduce some fundamental concepts and notations related to point clouds and deep learning. We then proceed to discuss the state-of-the-art solutions for video frame interpolation, deep learning on point clouds, upsampling point clouds, and scene flow estimation. We also look into point cloud distance metrics and data sets of dynamic point clouds that are publicly available. Then, in Chapter 3, we detail the architecture

of our interpolation system, and the components used in it. In Chapter 4 we describe the synthetic data sets that we have created, as well as the setup of our experimental evaluation and its results. In Chapter 5 we share some insights that we have obtained throughout this project, we discuss the limitations of our work, and we propose a number of directions in which this work could be extended in the future. Finally, we summarize the work carried out in this thesis, and conclude.

2

Related Work

In this Chapter we provide an overview of existing research on various subjects closely related to the problem of temporal interpolation of dynamic point clouds. We first introduce the basic concepts of static- and dynamic point clouds and their notation in Section 2.1. In Section 2.2 we provide a brief introduction to deep learning, and we discuss all the components required in order to design a neural network architecture. After introducing these basic concepts, we review recent state-of-the-art on various related topics. In Section 2.3 we discuss a number of solutions to video frame interpolation, the 2D image equivalent of our temporal interpolation problem. By looking at video frame interpolation techniques, we can learn the general concepts of temporal interpolation, and gain inspiration for the design of an architecture that can perform temporal interpolation on dynamic point clouds. Then, in Section 2.4, we review the state-of-the-art of applying machine learning to point clouds, in order to learn in what ways machine learning algorithms can be applied to point clouds. In Section 2.5 we discuss a number of works that upsample point cloud spatially. In these works we can learn about techniques that output point clouds directly. In Section 2.6 we then look into scene flow, which is the 3D equivalent of optical flow. Scene flow estimation can be used to perform temporal interpolation, and is the most closely related state-of-the-art to our work. In Section 2.7 we explore point cloud distance metrics, which potentially have uses both in training our neural network, as well as in the evaluation of our results. Lastly, in Section 2.8 we discuss available dynamic point cloud data sets, which we will need to train and evaluate our architecture.

2.1 Point clouds

Point cloud A point cloud is a set of 3D points, with no ordering relations or spatial connections among the individual points. To describe point clouds, we adopt a notation similar to (29). A D-dimensional point cloud with N points can be denoted by $\mathcal{P} = \{p_1, ..., p_N\} \subseteq \mathbb{R}^{N \times D}$. The most basic point clouds have $D = 3$, where each point is associated with its 3D coordinates (spatial attribute), i.e. $p_i = \{x_i, y_i, z_i\}$ with $i = [1, ..., N]$. Each point, in addition to having a spatial attribute, may also be associated with a number of other attributes such as color, reflectance or surface normal. Throughout this thesis, unless specified otherwise, we consider 6-dimensional ($D = 6$) point clouds that have spatial and color attributes, i.e. $p_i = \{x_i, y_i, z_i, r_i, g_i, b_i\}$ with $i = [1, ..., N]$, where r_i, g_i and b_i represent the coordinates in the RGB color space.

Dynamic point cloud A dynamic point cloud is a temporal sequence of point clouds, used to represent time-changing 3D scenes or objects. We denote a D-dimensional dynamic point cloud with duration L as $\mathcal{DP} = \{\mathcal{P}^1, ..., \mathcal{P}^L\}$, where $\mathcal{P}^j = \{p_1^j, ..., p_{N_j}^j\} \subseteq \mathbb{R}^{N_j \times D}$ is a point cloud at instant j, with N_j points, $j = [1, ..., L]$. We will sometimes refer to such a point cloud $\mathcal{P}^j$ part of a dynamic point cloud as a *frame*. Dynamic point clouds are captured at a certain *capture frame rate*, describing how many frames are captured per second. For simplicity, we will usually assume this capture frame rate to be constant throughout the sequence. The *playback frame rate* describes at how many frames per second the dynamic point cloud is being rendered during playback.

Scene flow Scene flow is the 3D equivalent of optical flow. It describes for each point the 3D motion of that point between two subsequent frames (33). We denote the scene flow between two dynamic point cloud frames $\mathcal{P}^i$ and $\mathcal{P}^{i+1}$ as $\mathcal{F}^{i \to i+1} = \{f_1^{i \to i+1}, ..., f_{N_{\mathcal{P}^i}}^{i \to i+1}\}$. We will say that a scene flow estimation is *accurate* if $p_j^i + f_j^{i \to i+1} = p_j^{i+1}$ for all pairs $i = [1, ..., L-1]$, $j = [1, ...N_{\mathcal{P}^i}]$, that is, if the motion is accurately predicted for all points.

We will say that a scene flow estimation is *forward-connecting* if $\exists k \in \{1, ..., N_{i+1}\} \mid p_j^i + f_j^{i \to i+1} = p_k^{i+1}$ for all pairs $i = [1, ..., L-1]$, $j = [1, ..., N_{\mathcal{P}^i}]$, that is, if each point from $\mathcal{P}^i$ is mapped to some point in $\mathcal{P}^{i+1}$. Similarly, we say that a scene flow estimation is *backward-connecting* if $\exists j \in \{1, ..., N_i\} \mid p_j^i + f_j^{i \to i+1} = p_k^{i+1}$ for all pairs $i = [1, ..., L-1]$, $k = [1, ..., N_{\mathcal{P}^{i+1}}]$, that is, if each point from $\mathcal{P}^{i+1}$ has a point from $\mathcal{P}^i$ mapping to it. Lastly, we will say a scene flow estimation is *fully-connecting* if it is both forward-connecting and

backward-connecting. Note that if a scene flow estimation is accurate, it is necessarily also fully-connecting.

Surface normals Surface normals, or simply *normals*, are vectors that are perpendicular to the modeled object. They earn their name from the fact that their length is normalized, meaning that only the direction of the vector is relevant, not their magnitude. Normals are required for a variety of algorithms and distance metrics. Some forms of point cloud acquisition might output surface normals directly, but in most cases the normals will have to be estimated using a normal estimation algorithm (34). Figure 2.1 shows an example point cloud with surface normals rendered.

Voxel grid A voxel grid is a regular three-dimensional grid, where each element is called a *voxel*. In a voxel grid, each voxel typically does not explicitly carry its spatial location, which is instead inferred from the position in the data structure in which it is stored. A key property of a voxel grid is its *resolution*, which describes how many voxels it contains in each dimension. For example, a voxel grid with a resolution of $16 \times 16 \times 16$ would contain 16 voxels in each dimension, for a total of $16 * 16 * 16 = 4096$ voxels. Voxel grids allow for fast look-up of elements based on spatial coordinates. They are commonly implemented through *octrees*.

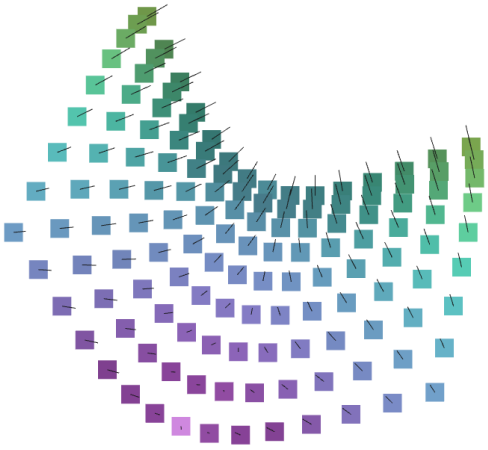
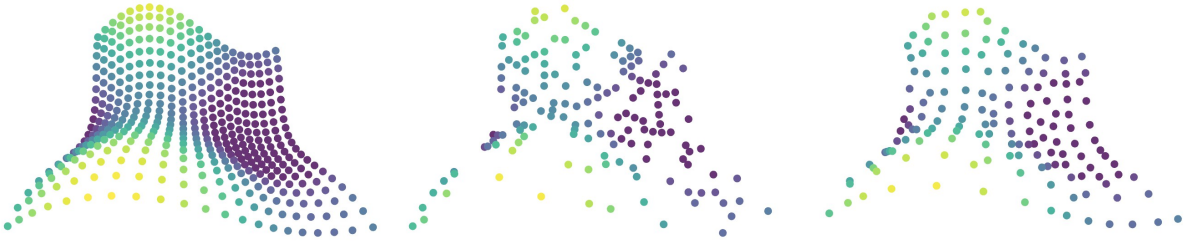


Figure 2.1: Example point cloud with surface normals. Here the black lines represent the surface normals. It can be seen that these normals are always perpendicular to the surface of the object.



(a) No downsampling **(b)** Uniform downsampling **(c)** Voxel downsampling

Figure 2.2: Comparison of downsampling techniques (a) shows the original point cloud, without any downsampling, (b) shows the result of uniform downsampling, and (c) shows the result of voxel downsampling. It can be seen that the voxel downsampling has more informative points.

Octree An octree is a tree where each internal node has precisely 8 children. In each node, the space is further split up into two equals parts along each dimension, resulting in 8 equal sized *octants*. A node at depth k thus covers 2^{-k} times the total space in each dimension. Note that the tree does not have to be balanced, and as such it is permissible for one node at a certain depth to have children, while another node at the same depth might not. This way, the octree does not waste storage space for areas which are unoccupied.

Voxelization Voxelization is the process of converting a point cloud into a voxel grid. First, a specific voxel grid resolution has to be chosen. Next, all points that fall within one voxel are aggregated into one point. Common aggregation operators include simply taking the mean value of all points, or picking one point at random.

Uniform downsampling Uniform downsampling is a technique to reduce the number of points (downsample) in a point cloud. This is done simply by randomly selecting the desired number of points. Figure 2.2b shown an example of uniform downsampling.

Voxel downsampling Voxel downsampling is another downsampling technique, that is similar to the process of voxelization. In the case of voxel downsampling, points are aggregated just as in voxelization, but are output as a point cloud instead of as a voxel grid. Compared to uniform downsampling this technique tends to result in more informative points (points that are not close to other points, and thus present new information). Figure 2.2c shows an example of voxel downsampling.

kd-tree A kd-tree is a tree-based data structure similar to an octree, which can be used to store points. In a kd-tree, each internal node has precisely two children. At each node, the space is split along one dimension, in such a way that both child-nodes cover approximately the same number of points. kd-trees allow for efficient look up of higher-dimensional data, and are especially useful for efficient nearest-neighbourhood search.

2.2 Deep learning

Deep learning is a class of machine learning techniques that aims to model complex relationships between data by learning on multiple levels of representation (23). It creates a hierarchy of features, where higher-level features can be learned from lower-level features. It earns its name from creating a *deep* hierarchy of features, in the form of a neural network with many layers.

Deep learning has been applied in a wide range of applications, and has drastically improved the state-of-the-art in many of these. Some applications include image classification (35, 36, 37, 38), optical character recognition (39, 40, 41), speech recognition (42, 43, 44, 45), and game agents (46, 47). For a more thorough overview of deep learning applications, see (48).

A neural network is a network composed of simple elements called neurons. Each neuron takes a number of input values, and generates an output value based on these input values,

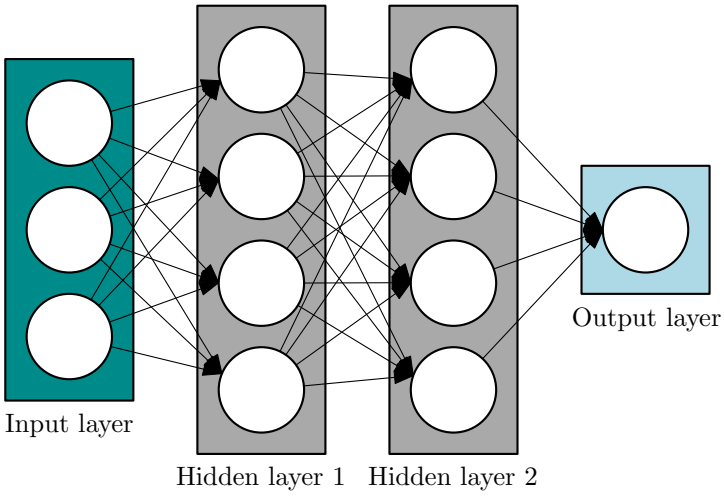


Figure 2.3: Example neural network architecture. This Figure shows an example of a simple neural network architecture. Here each circle represents an individual neuron. This network takes three input values, and produces one output.

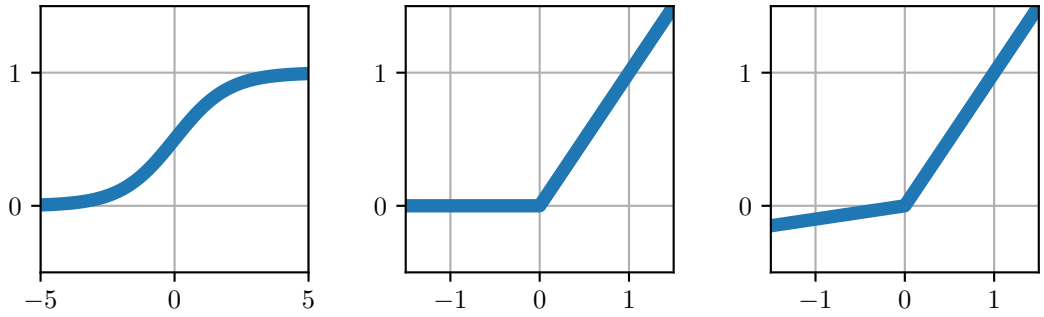
some learnable weights, and a learnable bias. A typical neuron might generate its output value based on the following formula:

$$g(x) = f(W^T x + b) \tag{2.1}$$

Here $g(x)$ is the output of the neuron given an input vector x, W is a vector of weights, and W^T is its transpose vector, b is a bias vector, and f is an activation function. Multiple such neurons can be organized into a layer, and multiple layers in turn form the network. Figure 2.3 shows an example of a simple network architecture. In practice, neural networks consist of vastly more neurons and layers.

In the remainder of this section we discuss a number of components commonly used in designing network architectures and training them. We discuss convolutional neural networks in more detail at the end of this Section.

Activation function An activation function is a function that takes a single number and performs some mathematical operation on it to produce an output. They are applied to the intermediary output of neurons, as is shown in Equation 2.1. The use of an activation function has two benefits. First, they help towards bounding the output values to a finite domain, preventing numbers from becoming increasingly larger as the network grows deeper. Second, it introduces non-linearity. Intuitively, this allows a neuron to *fire* or *not-fire*, which can help the network learn more complicated features. Figure 2.4 shows a number of activation functions, ReLU being among the most popular of all. For a deeper understanding of the benefits of activation functions and the strengths and weaknesses of particular functions, see (49).



(a) Sigmoid **(b)** ReLU **(c)** Leaky ReLU

Figure 2.4: Activation functions

Loss function A loss function is a function that assigns a score to the output generated by the network, describing how far this output is from the desired output. In other words, it captures the error. If the generated output is identical to the desired output, the loss function should be 0. The worse the generated output is, the higher the loss function should be. This loss function is then used by the training algorithm to determine how well the network is performing, and in which direction the weights in the network should be adjusted. The loss function is problem dependent, and as such a new loss function has to be defined for each problem.

Optimizer An optimizer is responsible for adjusting the trainable weights in a network in such a way that the network achieves high performance in whatever its task is. It does this by attempting to minimize the loss function. Most modern optimizers are variations of the stochastic gradient descent (SGD) algorithm (49). Pseudocode for a simple implementation of SGD is shown in Algorithm 1.

Algorithm 1 Stochastic Gradient Descent

```
procedure SGD(data, learning_rate, loss_fn)
    weights ← initialize_weights()
    while not converged AND not maximum iterations reached do
        batch ← next_batch(data)
        result ← evaluate(batch, weights)
        loss ← loss_fn(result)
        gradients ← calculate_gradients(loss, weights)
        weights ← weights - gradients × learning_rate
    end while
    return weights
end procedure
```

Simply put, some data is fed through the network, the output is evaluated by the loss function, gradients are then calculated for each weight in each neuron, and lastly these weights are adjusted based on their gradient and the learning rate. The learning rate thus determines the magnitude of the adjustments to the weights. Selecting a proper learning rate is important. Namely, selecting a too low learning rate might mean that training will take a long time to converge, or that it will become stuck in a local minima, while picking a too high learning rate might mean that you constantly overshoot the optimal solution. To this end, more advanced optimizers have been introduced that dynamically

adjust the learning rate, as well as to address other shortcomings of SGD. Example of such optimizers are SGD+momentum (50), AdaGrad (51), RMSProp (52), or Adam (53).

Dropout Dropout is the act of randomly disabling a portion of the neurons during training. During each forward- or backward pass all neurons are randomly enabled or disabled. This technique has been shown to reduce overfitting, as the network is forced not to rely too much on the output of individual neurons. It also reduces the number of computations required per pass, though this is offset by the fact that generally more iterations are required to converge. During inference, all neurons are always used.

Convolutional neural networks

A special variant of neural networks are convolutional neural networks (also known as CNN or ConvNet). Like any neural network, a CNN consists of an input layer, an output layer, and a number of hidden layers. The hidden layers of a CNN consist of a series of convolutional layers, which are typically interspersed with pooling- and normalization layers. These convolutional layers are then often followed by some fully connected layers (54).

The parameters of the convolutional layer consist of a set of learnable filters. Each filter is relatively small in width and height, but covers the full depth of the previous layer. These filters are moved over the input, and for each position the dot product of that input

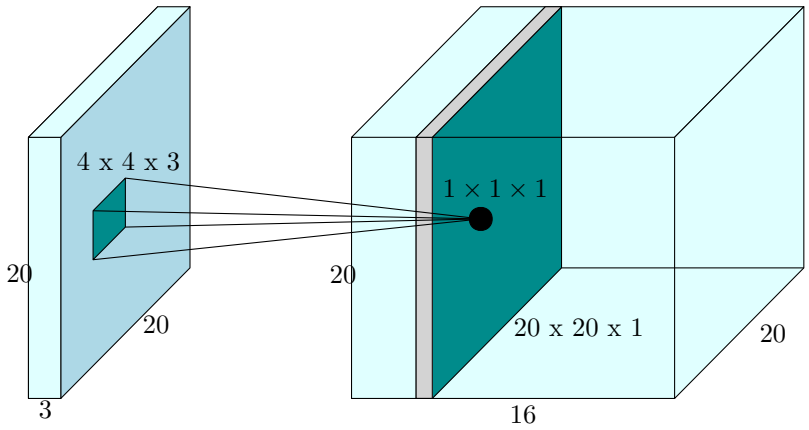


Figure 2.5: 2D convolutional layer example. In this example, the input has the dimensions $20 \times 20 \times 3$. Each filter is of the dimensions $4 \times 4 \times 3$. This filter is moved over each position of the input, and for each position an output value is calculated and stored in the feature map. One such feature map is highlighted, it has dimensions $20 \times 20 \times 1$. Notice that the width and height are identical to these of the input. In this example the output has dimensions $20 \times 20 \times 16$, which means that 16 separate filters are used.

patch and the weights of the filter is calculated. The result of this dot product becomes a value in the output of the layer, called a feature map. The distance the filter moves is determined by the stride hyper-parameter. Note that if the stride is larger than 1, the output will be of lower resolution than the input. The input is padded to ensure that the filter can be centered around each individual input value. For each filter, we will obtain one feature map, so we can configure the depth of the output by using that many filters. Figure 2.5 shows an example of a convolutional layer.

After each convolutional layer, the feature maps are typically ran through an activation layer, for example ReLU, which improves the ability of the network to learn (55). After one or a number of such convolutional layers, a so-called pooling layer is typically applied. A pooling layer *pools* together multiple values into a single value, in order to reduce the resolution. An example of a commonly used pooling layer is the max-pooling layer, where the input is divided into fixed-size regions, and the output consists of the maximum value of each region. The intuition here is that each filter in a convolutional layer will be trained to detect a certain pattern. If such a pattern is present at any location in a region, the feature map will contain a high output value for that region, otherwise it will contain a low output value. Thereby it reduces the resolution of the feature map, and creates an invariance to small shifts and distortions (23).

2.3 Video frame interpolation

Video frame interpolation is one of the basic video processing techniques, where the goal is to generate a new frame in between each pair of frames in the input video. This is most often done by predicting the optical flow, and generating the interpolation results based on that (56). Estimating the optical flow is a challenging problem in itself, which in particular has trouble dealing with occlusion, large motion, lack of texture, and blur (57). As a result, the estimated optical flow is often noisy. The success of deep neural networks in many computer vision tasks has inspired researchers to explore their use in the task of frame interpolation. Neural network based approaches now represent that state-of-the-art in video frame interpolation (56). This success of neural networks in video frame interpolation has inspired us to use neural networks as part of our architecture for temporal interpolation of dynamic point clouds. In this section we will discuss some of these state-of-the-art solutions in video frame interpolation.

Adaptive Separable Convolution (25) This work builds on the prior work on adaptive convolution (AdaConv) (26), which we will dicsus first. AdaConv is a convolutional

neural network based approach to perform video frame interpolation. Output pixels are synthesized simply by performing local convolution over the two input frames. This approach yields good results when the motion between frames is small, but falls behind when the motion is larger than the size of the convolution kernels. In the follow-up work, Adaptive Separable Convolution (SepConv) (25) 1D kernels are used instead of 2D kernels. This vastly reduces the number of parameters in the network, while also allowing the network to handle larger motion, and achieve more visually pleasing frames. The network is able to deal with occlusion reasonably well, but large motion is still an issue.

Context-aware Synthesis (24) Context-aware Synthesis (CtxSyn) combines context information, flow estimation, and a synthesis network to perform video frame interpolation. To extract per-pixel context information, the pre-trained image classification network ResNet (58) is used. Next, PWC-Net (59) is used to estimate the bidirectional optical flow. A synthesis network consisting of various upsampling-, downsampling-, and convolution layers then generates the interpolation result based on the context information and optical flow. This approach is able to make context-aware decisions, and managed to achieve state-of-the-art performance.

Super SloMo (21) The interpolation techniques we have discussed so far all focus on generating one interpolated frame per pair of input frames. Super SloMo takes a different approach, and focuses on optimizing the interpolation result of videos as a whole. First, visibility maps are generated, describing for each pixel in what moment in time they become occluded (if at all). For each pair of input frames the bidirectional optical flow is then estimated through a flow computation CNN. The bidirectional optical flow needs to be merged into a single optical flow, which is done by linearly fusing them. The influence of each optical flow is dependent on the visibility maps (such that occluded pixels have no influence), as well as the desired moment in time of the interpolated frame. The obtained initial optical flow estimation is then further refined through a flow interpolation network. This approach makes the assumption that movement between consecutive frames is linear, meaning that while performance on already high frame rate input is phenomenal, but on lower frame rate input performance will degrade.

2.4 Learning on point clouds

Historically, neural networks have most commonly been used to process Euclidean data (data that follows a grid structure), for example images, text, and video. As a result, most research efforts have been directed at developing learning techniques that operate

on such Euclidean data. On non-Euclidean data, traditional neural network approaches, like for example convolution, are not always well-defined. Point clouds do not exhibit a Euclidean structure, and thus are non-Euclidean. There are two main strategies that can be employed when attempting to learn on point clouds, 1) convert the point cloud to some intermediate Euclidean structure, and perform learning on that, or 2) develop learning operators to learn on non-Euclidean data directly. In this section we will explore a variety of approaches, that cover both strategies.

It is worth mentioning that most learning efforts on point clouds so far have been aimed towards classification (given a point cloud, predict to which of a set of classes it belongs), or segmentation (given a point cloud, segment the points into various categories). While neither of these resembles our application, many of the applied techniques and architectures transfer well to different applications, and are thus of interest to us.

2.4.1 2D view-based methods

View-based methods work by generating a set of 2D views of the input point cloud, and by then applying existing 2D learning techniques to these 2D views. The seminal work in view-based methods is Multi-View CNN (MVCNN) (30). In MVCNN, 20 views from different viewpoints are rendered from the input point cloud. For each view, four rotations are taken (0°, 90°, 180°, and 270°), and for each of these rotations a shape descriptor is generated. A special CNN is then used to merge these 80 separate shape descriptors into one global shape descriptor, from which the final classification prediction is then made.

An advantage of view-based methods is that they can leverage the large body of existing research in 2D learning techniques to process the rendered 2D views. A challenge is to decide how many views to generate, and from which viewpoints. If too many views are generated, this adds overhead and complexity to the model. On the other hand, if an insufficient number of views is used, or if the wrong viewpoints are used, view-based methods might have difficulties dealing with occlusion.

In the context of interpolation of dynamic point clouds specifically there is an additional challenge, namely that we want the output of the interpolation process to be new point cloud frames. This means that any architecture using a view-based approach, would at some stage need to synthesize new point clouds based on these 2D views or their derived features. To avoid the complexity added by this synthesis process, we decide against using view-based methods in this thesis.

2.4.2 Volumetric methods

Similar to view-based methods, volumetric methods can learn on point cloud data by first converting the point clouds to an intermediate format. In their case, this intermediate format is a volumetric representation, for example an occupancy grid, KD-tree or other structure. Also similar to view-based methods, volumetric methods have the issue that we would have to convert from the intermediate representation back to point cloud. Additionally, volumetric methods tend to scale poorly with resolution, which means a relatively low resolution would have to be used. This can be troublesome for the task of temporal interpolation, where fine-grained motion plays a crucial role. As a consequence, we decide against using volumetric-based methods in this thesis. For completeness, we still provide a brief overview of the state-of-the-art in volumetric methods.

VoxelNet (31) VoxNet was the first architecture to employ a volumetric representation for point cloud learning. Their approach is simple: the point cloud is converted to an occupancy grid. An occupancy grid is a 3D grid structure where each point, called a voxel, gets a value based on all the points that are in it spatially. On this occupancy grid more conventional neural network approaches can be applied, such as 3D convolution-, pooling-, and fully connected layers. VoxNet uses stochastic search over hundreds of potential architectures to find the best performing one. The main benefit of their approach is that it is simple and allows the use of traditional neural network techniques. A downside is that the memory requirement of occupancy grids is $\mathcal{O}(n^3)$ with voxel size, which demands a relatively low resolution to be used. VoxNet commonly uses a $32 \times 32 \times 32$ resolution, for example, which depending on the use case may- or may not be sufficient for learning.

PointGrid (60) Another successful volumetric learning method is PointGrid. Similarly to VoxelNet, PointGrid works with occupancy grids. With PointGrid, however, each cell in the grid is allowed to have a constant number of points in it, allowing the network to learn higher order local approximation functions. By allowing multiple points per grid cell, they find that a grid sizes as small as 16×16 are sufficient to obtain good performance. Due to this smaller grid size, PointGrid achieves significantly smaller memory footprint than other volumetric methods, while still achieving comparable- or better performance.

Deep kd-networks (61) Kd-networks use kd-trees instead of occupancy grids to more efficiently represent the points. A kd-tree is a tree where in each node, the data in one dimension is split into roughly equal-sized parts. When the number of dimensions is limited, such a tree can be efficiently generated and queried. Unlike occupancy grids, kd-trees only have to store points that are present in the input, and thus no space is

wasted storing unoccupied regions. Kd-networks achieve better scaling than occupancy grid-based architectures, but fundamentally still operate by bounding points into groups, and thus do not use the true neighbourhood of each point.

2.4.3 Geometric deep learning & PointNets

The view-based- and volumetric methods that we have discussed so far learn on point cloud data by first converting it to a Euclidean representation. Recently there has been a growing interest in applying learning with neural networks directly on non-Euclidean data, for example on social networks, sensor networks, and genetics (62). *Geometric deep learning* is a class of emerging techniques that apply learning directly on non-Euclidean data. A subclass of geometric deep learning are PointNets, which focus specifically on learning on point clouds. In this Section we provide a concise overview of PointNets. For a more in-depth introduction to geometric deep learning as a whole, we refer the reader to (62).

Point clouds in particular pose a difficult challenge for neural networks. Namely, the points in a point cloud are not in a specific order, which means the network needs to be invariant to any permutation of point order. At the same time, the geometry of the points and the relation between a point and its neighbours is meaningful, and needs to be taken into consideration by the network. PointNets are an emerging class of network architectures designed specifically with these constraints in mind. As we will discuss in more detail in Chapter 3, we will use a PointNet-style approach in our architecture. We now first look at the state-of-the-art in PointNets.

PointNet (27) is the pioneer of PointNets, and offers architectures for both classification and segmentation. PointNet works by first feeding all the input points through a shared Multilayer Perceptron (MLP), creating a local feature vector for each point. Next, a symmetric function (for example max pooling) is applied along the first axis, resulting in a global feature vector that is invariant to the order of the input. In the case of the classification network this global feature vector is fed through another MLP which then produces the output scores. In the case of the segmentation network, the global feature vector is concatenated to each of the points, which are then fed through more MLPs to generate the output scores. This ensures that the segmentation scores take into consideration both local features and global features.

PointNet++ (28) is the successor of PointNet. A shortcoming of the original PointNet architecture is that it does not capture local neighbourhood structure, preventing it from recognizing fine-grained patterns and generalizing to complex scenes. PointNet++ first

partitions the input sets into overlapping local regions, after which the original PointNet architecture is used as a building block to extract features from these local regions. This process is repeated hierarchically to generate increasingly high-level features. While PointNet++ achieved state-of-the-art results at the time, it still uses PointNet, which means points in local regions are still processed independently, and that it does thus not consider relationships between these points.

Dynamic Graph CNN (DGCNN) (29) is an approach inspired by PointNet and convolutional neural networks. It aims to improve performance by taking into consideration information of the local neighbourhood. It achieves this by first calculating the k-nearest neighbour graph. That is, for each input point p an edge is drawn to the k nearest points $n_1, \ldots n_k$. For each input point p, k feature vectors are generated based on both p and $n_i - p$, which allows these feature vectors to capture both global shape information (from p), and local neighbourhood information (from $n_i - p$). Convolution is then applied on these feature vectors. To make an analogy to traditional 2D convolution: the input point p corresponds to the central pixel in 2D convolution, and the neighbours correspond to the patch of pixels around this central pixel. Similarly to PointNet, DGCNN proceeds to apply a symmetric function to ensure invariance to the order of the input.

PointCNN (63) is another method that directly applies convolutional operators to point in a point cloud. They perform k-nearest neighbour search, and use Multi-Layer Perceptrons on the resulting neighbourhood features to learn a transformation χ. This transformation χ has two functions: it weighs the input features, and it permutes the input points into a latent and potentially canonical order. After the input points have been transformed using the learned χ-transformation, convolution can be applied.

2.5 Upscaling point clouds

Little work has been done on upscaling point clouds, and to the best of our knowledge there is no existing work attempting to increase the frame rate of dynamic point clouds. In this section we explore a number of works that perform other forms of upscaling, for example spatial upscaling.

PU-net (64) is a neural network architecture designed to upsample point clouds. That is, to increase the number of points in the point cloud. The main success criteria for added points is that 1) they fall on the underlying geometry of the point cloud, and 2) they are informative, and thus do not cluster around existing points. The PU-net architecture consists of four phases: 1) splitting the input into patches, 2) learning deep features on

each of these patches, 3) expanding the learned features, and 4) reconstruct points from these expanded features. PU-net is capable of vastly increasing the density of sparse point clouds. It is, however, not designed for completion, which means it is not capable of filling in large gaps or missing parts.

Spatio-temporal Upsampling (65). Low-budget point cloud capture setups often produce frames that are temporally inconsistent, there might for example be gaps, missing parts, or other forms of noise. This work aims to alleviate these inconsistencies. To this end, the authors propose an approach to upsample point cloud sequences in a spatio-temporally consistent manner. First, Edge Aware Upsampling (66) is applied to a point cloud at frame j to increase its number of points. At the same time the optical flow is calculated for frame j based on the 2D input frames. With this optical flow, the upsampled point cloud from frame j is then projected in time to frame $j + 1$. Finally, this projected point cloud is then merged with the actual frame $j + 1$ in order to produce the output frame.

2.6 Scene flow

Scene flow is the 3D equivalent of optical flow (33). In the context of point clouds, it describes for each point the 3D motion of that point throughout a sequence of frames (33). Scene flow can be used for various applications, for example providing motion cues for 3D segmentation, action recognition, or camera pose estimation (22). If the motion between two consecutive frames can be considered to be linear (a reasonable assumption if the frame rate is sufficiently high), scene flow can also be used directly to obtain good frame interpolation results.

Until recently, most scene flow estimations were generated by combining the optical flow estimations from multiple 2D views. Such methods can only be applied when multiple RGB-D source images are available, which might not always be the case. Additionally, the optical flow acquisition methods are not optimized for scene flow construction, which might result in poor scene flow estimations. New research is being carried out to estimate scene flow directly from point clouds.

Rigid Scene Flow (67) This work focuses on estimating scene flow on point clouds obtained from 3D LiDAR scans. This is is done by viewing the problem as an energy minimization problem, which is then solved using the Levenberg-Marquardt algorithm (68). The assumption is made that all transformations are rigid, and thus that there is no local deformation. This assumption holds up, as the work is focused on LiDAR scans, where

the focus might typically be on moving objects such as cars. While the algorithm is also tested on non-rigid data, there is no convincing comparison to other work.

FlowNet3D (22) FlowNet3D applies a deep learning approach to estimate scene flow directly on point cloud data. The neural network consists of three important components: 1) a convolution layer adopted from PointNet++ (28) in order to learn point features, 2) a novel flow embedding layer to merge features from two separate input frames, and 3) an up-convolution layer to refine the scene flow. The network is trained with synthetic scene flow ground-truth data, which proves to transfer well to the real-world data from the KITTI scene flow data set. The model is only evaluated on data featuring rigid motions, and thus no local deformation.

2.7 Point cloud distance metric

A point cloud distance metric describes how closely two given point clouds resemble each other, assigning a low score for point clouds that match each other closely, while assigning a high score for point clouds that differ greatly. In our work such distance metrics play a crucial role in two ways. Firstly, they can be used as *loss function* for training our neural network. Loss functions are used by the training algorithm to determine which solutions are desirable, and to stir the training process in the right direction. We discuss loss functions in more detail in Section 3.1.5. Secondly, such distance metrics can be used to evaluate the performance of our architecture, by comparing interpolation results to the available ground truths. In this Section we discuss a number of distance metrics and their merits.

Projection distortion (69) One distance metric to be considered is the projection distortion. This metric is based on generating a number of 2D rendering of the 3D point clouds, and comparing these using traditional 2D image distortion metrics, for example the mean squared error. The difficulty here is in deciding how many views to render, from what angles, and what rendering technique to use. If the views are selected poorly, this method is prone to occlusion, and certain points might not be weighed into the score. The projected distortion has as benefits that it is a relatively simple metrics, and it can leverage existing research in 2D image distortion metrics. *Visual Information Fidelity* (VIF) (70) is a 2D distortion metric that has been shown to have a relatively strong correlation with perceived visual similarity (71), and is thus a good candidate to use for projection distortion.

Point-to-point (matching distortion) (69) The matching distortion, or point-to-point distortion, is a metric based on *matching* points from one point cloud to points in the other point cloud. A typical matching strategy might be to for each point $p \in P$ select a point $\hat{p} \in Q$ which minimizes the Euclidean distance between p and $\hat{p}$. The one-way mean squared matching distortion can be calculated by taking the mean squared error between the matched pairs p and $\hat{p}$. The symmetric mean squared matching distortion is then the maximum of the one-way matching distortion from P to Q and the one-way matching distortion from Q to P. A similar metric is the Hausdorff matching distortion, which can be found by instead of taking the averages of the squared error as above, taking their maximum. For a more rigorous definition of the mean matching distortion and Hausdorff matching distortion, please see (69). An arguable shortcoming of the matching distortion metric is that it focuses only on closeness of matched points, instead of whether the two points belong to the same surface. As a result, the matching distortion metric tends to correlate less with perceived similarity than certain other metrics (71).

Point-to-plane (72) The point-to-plane distortion is a metric similar to the matching distortion metric. It also matches each point $p \in P$ to a point $\hat{p} \in Q$ by minimizing the Euclidean distance between p and $\hat{p}$. Instead of simply taking the mean squared error then, however, the point error is projected along the normal of p, meaning that we only consider the error in the direction of the normal. As a result, an error across the surface is not punished, and more focus is put on whether the two point clouds describe the same surface or not. The point-to-plane metric appears to correlate better to perceived similarity on human subjects than the matching distortion metric (71). One additional requirement this metric has, however, is that normals must be known for at least one of the point clouds. These normals can of course be estimated using one of various normal estimation techniques (34), should they not be available directly.

Plane-to-plane (73) Another class of distance metrics is that of plane-to-plane. These metrics are based on angular similarity of the modeled surface. Once again, given two point clouds P and Q, we find for each point $p \in P$ a point $\hat{p} \in Q$ that minimizes that distance between p and $\hat{p}$. Then an angular similarity between the normals of p and $\hat{p}$ is calculated, which is a measure that describes how much the normals deviate from each other. These angular similarities are then aggregated across all points, for example by taking their average. This metric outperforms the point-to-point and point-to-plane metrics in correlation with perceived similarity on human bodies (71). In order to use the plane-to-plane metric, normals must be known for both point clouds that are to be compared.

2.8 Data sets

In order to train our neural network and to perform our evaluation, we require a data set of dynamic point cloud sequences. Such data sets are currently scarce. The two main data sets that we have been able to find are hosted by JPEG PLENO. This is an initiative by the JPEG standardization committee, which aims to provide a standard framework for capture, representation, and exchange of, among others, point clouds (74). One of the services they provide is a unrestricted database of point cloud data sets. Of these, two are dynamic point cloud data sets.

- **8i Voxelized Full Bodies (16)** This data set, also known as *8iVFB v2*, contains four dynamic point cloud sequence. Each sequence captures a different human subject, at 30 FPS over a 10 second period. The point clouds are captured from 42 RGB cameras configured in 14 clusters, capturing the subject from all directions. Due to post-processing, this data set is very clean; there is no visible noise or artifacts. See Table 2.1 for more details about the individual sequences, see Figure 2.6 for previews.

(a) Long Dress **(b)** Loot **(c)** Red and Black **(d)** Soldier

Figure 2.6: 8i Voxelized Full Bodies data set

Name	Frames	Mean points	File size
Long Dress	300	834 315	5.8 GB
Loot	300	793 821	5.1 GB
Red and Black	300	727 070	4.8 GB
Soldier	300	1 075 299	7.3 GB

Table 2.1: 8i Voxelized Full Bodies data set. Each sequence lasts 10 seconds at 30 FPS, for a total of 300 frames. The point clouds are voxelized at a voxel depth of 10, resulting in a voxel size of approximately 1.75mm per voxel.

- **Microsoft Voxelized Upper Bodies (75)** This data set contains five dynamic point cloud sequences of the upper bodies of human subjects. The sequences are captured using four frontal RGBD cameras, and as such only the front of each subject is visible. The sequences are captured at 30 FPS, at are between 7- and 10 seconds each. The data set is offered both at voxel depth 9 and voxel depth 10, corresponding to a voxel size of 1.5mm and 0.75mm, respectively. There is a reasonable amount of noise in this data set, especially around the edges of the subjects. See Table 2.2 for more details about the individual sequences, see Figure 2.7 for previews.

(a) Andrew9 **(b)** Sarah9 **(c)** Phil9

(d) David9 **(e)** Ricardo9

Figure 2.7: Microsoft Voxelized Upper Bodies

Name	Frames	Mean points	File size
Andrew9	318	283 363	1.9 GB
David9	216	349 173	1.5 GB
Phil9	245	332 252	1.7 GB
Ricardo9	216	226 225	1.0 GB
Sarah9	207	259 689	1.1 GB

Table 2.2: Microsoft Voxelized Upper Bodies data set. The point clouds are voxelized at a voxel depth of 9, resulting in a voxel size of approximately 0.75mm per voxel.

3

Architecture

In this Chapter, we discuss the architecture that we have developed in order to perform temporal interpolation of dynamic point cloud sequences. In Section 3.1 we discuss a number of architectural decisions that have been made, and we elaborate our approach. In Section 3.2 we present a high-level overview of our architecture. In the remaining Sections of this Chapter we discuss various components of our architecture in more detail.

3.1 Architectural decisions

When designing any architecture, a number of architectural decisions have to be made. In the problem of temporal interpolation of point clouds, these are decisions such as: what interpolation approach to use, what format the input and output should be in, or what type of loss function is most suitable. In this Section we discuss the architectural decisions we have made for this project, and the alternatives that we have considered.

3.1.1 Interpolation approach

We identify two approaches that could be taken to perform the temporal interpolation:

- **Frame-by-frame** Performing the interpolation frame-by-frame is arguably the simplest method of interpolation. Here frames are interpolated based on information only from the two surrounding frames. This approach is illustrated in Figure 3.1.
- **Sequence as a whole** The alternative to frame-by-frame interpolation is to try to optimize the interpolation of an entire dynamic point cloud sequence as a whole. This way information not just from the two surrounding frames can be leveraged, but also information from other frames.

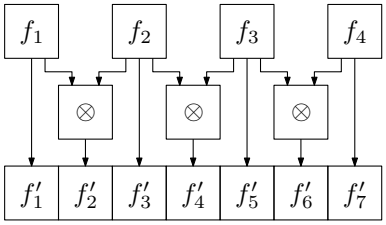


Figure 3.1: Frame-by-frame interpolation approach. Here the input sequence consists of frames f_1, f_2, f_3, and f_4. The operator $\otimes$ represents temporal interpolation. Pairs of consecutive input frames are interpolated, and the resulting frames are then interwoven with the input frames, in this case doubling the frame rate of the sequence. Note that it would also be possible to interpolate more than one frame between each pair of input frames.

One major advantage from the frame-by-frame approach is that it keeps things simple, which is a welcome benefit to an already challenging problem. With the frame-by-frame approach, it is likely that a simpler model will be sufficient to properly make use of all available information. Additionally, we suspect that the training will be easier and require less training data, as each training pass would only require two or three point clouds as opposed to an entire dynamic point cloud sequence. An advantage of interpolating the sequence as a whole is that more information is available, and thus theoretically a better interpolation could be made, if the model is capable of handling this information. Ultimately we have decided to employ the frame-by-frame approach, as we believe the benefits outweigh this theoretical disadvantage.

3.1.2 Input representation

As discussed in Section 2.4, there are various representations that can be used when applying machine learning on point clouds. Three representations that could be used are the following:

- **View-based** One or a number of 2D views are rendered from the point cloud, on which learning is then performed.
- **Volumetric** The point cloud is converted to a volumetric representation, such as a voxel grid, on which learning is then performed.
- **Point cloud** The point cloud is directly used for learning, without being converted to an intermediate format.

View-based- and volumetric representations have the advantage that they can leverage existing deep learning algorithms, as these typically work on data Euclidean data. On

the other hand, they each have their own limitations. For example, they might have difficulties dealing with occlusion, or their computational requirements might scale rapidly when high spatial resolutions are used. For the task of temporal interpolation, these representations add the additional challenge that the output has to be converted back to point clouds, which can be challenging to do in a stable manner. Other characteristics of these representations are discussed in more detail in Section 2.4.1 and Section 2.4.2.

We opt to create an architecture that can consume point clouds directly. By avoiding conversion to a different representation, we do not have to worry about concerns such as converting the output back to point clouds, or the loss of information during conversion. Additionally, this representation has seen much success in other tasks, such as classification and segmentation.

A challenge of learning on point clouds directly, is that point clouds do not exhibit a grid-like structure, their points are in fact completely order independent. This means that many traditional deep learning techniques cannot be applied, and new techniques have to be developed.

3.1.3 Input features

There is also the matter of what input features we permit ourselves to use in order to perform the interpolation. At a minimum, we will be working with 6-dimensional point clouds, where each point has a 3D spatial location, and RGB color values. Other features, such as surface normals or especially scene flow are likely not to be available, and thus should ideally not be required for inference.

For training, the features we can use are dictated by whatever features are available in the training data set. As we will discuss in Section 4.1, our training data set is extracted from meshes, meaning that we can generate ground truth surface normal- and scene flow data. We can thus permit ourselves to use surface normals and scene flow data during the training process.

3.1.4 Output

Another decision to be made is what data should be output by the architecture, and in what format. We consider two options:

- **Direct point output** The architecture directly outputs the interpolated frame(s).

- **Scene flow** The output is the scene flow for each point. Using linear interpolation, an arbitrary number of interpolated frames can trivially be generated during the rendering process.

The scene flow approach makes the assumption that the motion between points across the two frames is approximately linear. Whether this is a fair assumption depends on the motion of the dynamic point cloud sequence in question. We find that in the sequences of human bodies that we have experimented with, this assumption of approximate linearity holds up. However, the direct point output approach does not make this assumption, and as such could theoretically yield better results when motion between points is not approximately linear. A downside of the direct point output approach is that it can only interpolate a predetermined number of frames between each pair of input frames, and changing this number of interpolated frames will require architectural changes and retraining of the network. With the scene flow approach, an arbitrary number of frames can be interpolated trivially. Additionally, the use of scene flow as output allows us to use a simpler loss function, as we will discuss in Section 3.1.5. We favor the flexibility of the scene flow output approach, and have thus designed our architecture to output scene flow estimations.

3.1.5 Loss function

For the loss function, we are again presented with two options:

- **Point cloud distance metric** The first option is to use point cloud distance metrics. Here training would proceed with triplets of consecutive frames (f_1, f_2, f_3), where the frame $f_2' = \texttt{interpolate}(f_1, f_3)$ would then be compared against the ground truth frame f_2 using a point cloud distance metric.

- **Scene flow error** If scene flow is chosen as output format (Section 3.1.4), and if scene flow ground truth data is available, then scene flow error can be used as loss function. The estimated scene flow would simply be compared against the ground truth scene flow using existing error measures, such as the L1 norm or L2 norm.

There is a variety of point cloud distance metrics that could be used as loss function (see Section 2.7). Unfortunately, these metrics have a number of shortcomings when used as loss function, such as sub-optimal correlation with perceived similarity and lack of smooth gradient during training. We discuss our experiences with the usage of point cloud distance

metrics as loss functions in more detail in Section 5.1.6. From early experiments we learned that scene flow error tends to work better as a loss function, so we use scene flow error with L2 norm in our final architecture.

3.2 High-level architecture

We outline our architecture, based on the architectural decisions described above. See Figure 3.2 for a high-level overview of the architecture. As input we take two 6-dimensional point clouds, pc_1 and pc_2, each having spatial position- and color information. As the interpolation network uses nearest-neighbour search, and thus scales $\mathcal{O}(n^2)$ with n points in computation time, we can only feed it point clouds of limited spatial resolution. For this reason, we first downsample pc_1 and pc_2 to 2048 points each (Section 3.3). The interpolation network then takes these two downsampled point clouds, and estimates the corresponding scene flow (Section 3.4). Next, we upsample this scene flow estimation back up to the original resolution using 3d interpolation (Section 3.5). Lastly, we apply our neighbour snapping algorithm to increase the smoothness of the scene flow estimation (Section 3.6).

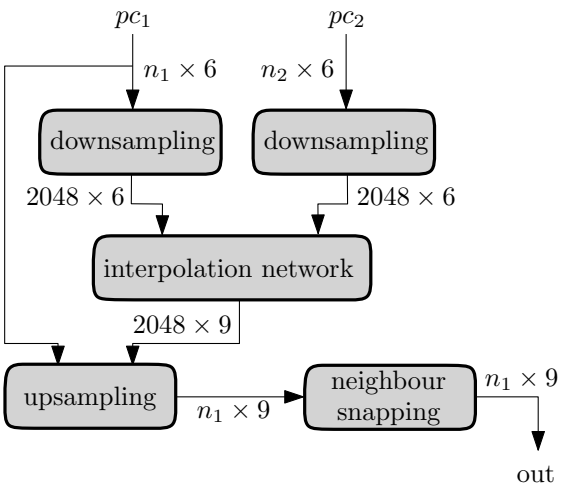


Figure 3.2: High-level architecture

3.3 Downsampling

We will first downsample the input point clouds to a lower spatial resolution. This is necessary because 1) the computational time and memory requirement of our interpolation

network scales $\mathcal{O}(n^2)$ with n input points, and as such cannot process large point clouds in reasonable time, and because 2) the interpolation network requires that all inputs have the same spatial resolution. Based on empirical testing, we have decided to downsample all inputs to a spatial resolution of 2048 points. We find that this resolution is small enough that computation is fast, yet large enough to describe the geometry with sufficient detail.

In Section 2.1 we already discussed the uniform- and voxel downsampling techniques. While voxel downsampling might result in more informative point clouds, it has the problem that the number of output points is not easily configurable, as this is dependent on the voxel size used. In our experience, a spatial resolution of 2048 points is large enough that uniform downsampling provides reasonably informative point clouds. For these reasons, we will apply uniform downsampling.

3.4 Interpolation network

The interpolation network is the central part of our architecture. It is the component that takes as input two point clouds, and estimates their scene flow. As shown in Figure 3.3, our interpolation network consists of two modules. The first module is the point matching module (Section 3.4.1). Given two point clouds pc_1 and pc_2, this module will learn a soft-mapping from the points in pc_1 to points in pc_2. The second module is the flow refinement module (Section 3.4.2). This module will further refine the output of the point matching module.

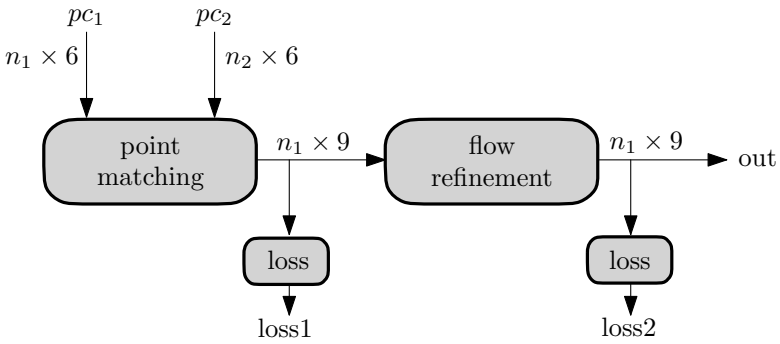


Figure 3.3: Neural network architecture

From early experimentation we have learned that end-to-end training does not achieve good results on this architecture. We speculate that this was because the flow refinement layer is encouraged to be highly conservative in its scene flow estimation, that is, to always

predict no- or low motion, and that this kills the gradient, preventing the point matching layer from learning properly.

To remedy this issue, we employ a two-phase training approach. First, we train only the weights of the point matching layer, based only on the output from the point matching layer (`loss1` in Figure 3.3). Once the network has sufficiently learned how to perform the point matching, we move on to the second phase. In this second phase, we freeze the weights in the point matching layer, preventing its weights from being changed during the remainder of the training process. We then train the flow refinement layer to convergence (based on `loss2` in Figure 3.3).

3.4.1 Point matching

Our point matching module takes as input two point clouds, $\mathcal{P} = \{p_1, ..., p_{N_p}\}$ and $\mathcal{Q} = \{q_1, ..., q_{N_q}\}$, and learns a soft mapping between points from $\mathcal{P}$ to points from $\mathcal{Q}$. That is, it learns a weight $w_{i,j}$ for all pairs $i \in \{1, ..., N_p\}$ and $j \in \{1, ..., N_q\}$, where weight $w_{i,j}$ describes how much point p_i is matched to point q_j. We impose that $\forall i \in \{1, ..., N_p\} \mid \sum_{j=0}^{N_q} w_{i,j} = 1$. From these weights, we calculate our first scene flow estimation $\mathcal{S}$ according to Equation 3.1. In other words, each weight $w_{i,j}$ describes how much point $p_i \in \mathcal{P}$ should be moved in the direction of point $q_j \in \mathcal{Q}$.

$$\mathcal{S} \leftarrow \left\{ \sum_{j=1}^{N_q} w_{i,j} * (q_j - p_i) \right\}_{i=1}^{N_p} \tag{3.1}$$

To learn these weights, we employ the architecture shown in Figure 3.4. First, we find for each point p in pc_1 the k-nearest neighbours, where k is a hyper-parameter. Higher values of k mean more neighbours have to be considered, and thus require more computation. Lower values of k mean smaller neighbourhoods, making it less likely that the true semantically corresponding point is in this neighbourhood, in which case the network can likely not make a good scene flow estimation. After empirical testing we use the value $k = 50$. We discuss the implications of this parameter in more detail in Section 5.1.5. For each neighbour p_n we now derive the features $\{p, p_n, p_n - p\}$. The features p and p_n provide information about the absolute position of the points. The feature $p_n - p$ (referred to as *neighbour flow* in Figure 3.4) provides information about the relative position of the neighbouring points. These features are fed through multiple Multilayer Perceptrons (MLP), which is where the actual learning takes place. In the last MLP layer, we use only a single filter, in order to obtain an output of dimensions $n_1 \times k \times 1$,

or simply $n_1 \times k$. We normalize the sum of outputs to 1 for each base point, and then use these values as weight w to derive the estimated scene flow as described in Equation 3.1.

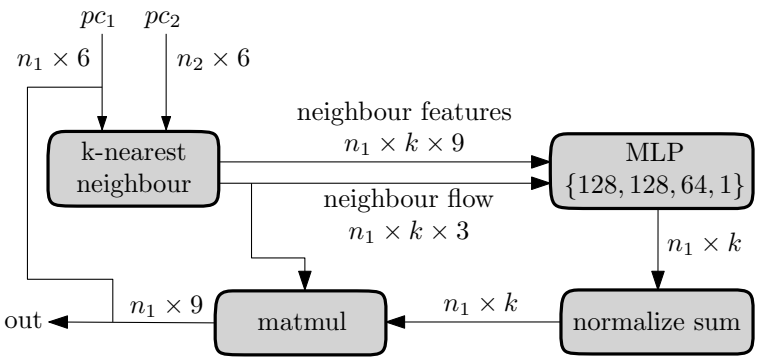


Figure 3.4: Point matching module architecture

3.4.2 Flow refinement

Our flow refinement layer takes as input one point cloud $\mathcal{P} = \{p_1, ..., p_{N_p}\}$ with scene flow estimation $\mathcal{S} = \{s_1, ..., s_{N_p}\}$, and will attempt to refine this scene flow estimation. Similarly to our point matching layer, it does this by learning a set of weights, where weight $w_{i,j}$ describes how much the refined scene flow estimation of point p_i will depend on the original scene flow estimation s_j. The refined scene flow estimation $\mathcal{S}'$ is then calculated according to Equation 3.2.

$$\mathcal{S}' \leftarrow \left\{ \sum_{j=1}^{N_p} w_{i,j} * s_j \right\}_{i=1}^{N_p} \tag{3.2}$$

The architecture of the flow refinement module is shown in Figure 3.5. First, we apply EdgeConv (29) in order to learn high-level features for each scene flow estimation. As explained briefly in Section 2.4.3, EdgeConv works by applying convolution to features from a local neighbourhood of points. The intention is that this will allow the module to learn how to detect- and correct outliers. After the EdgeConv layer, we apply another couple MLP. Similarly to the point matching module, we then normalize the sum of outputs to 1 for each base point and use these values as weight w, and calculate the refined scene flow as described in Equation 3.2.

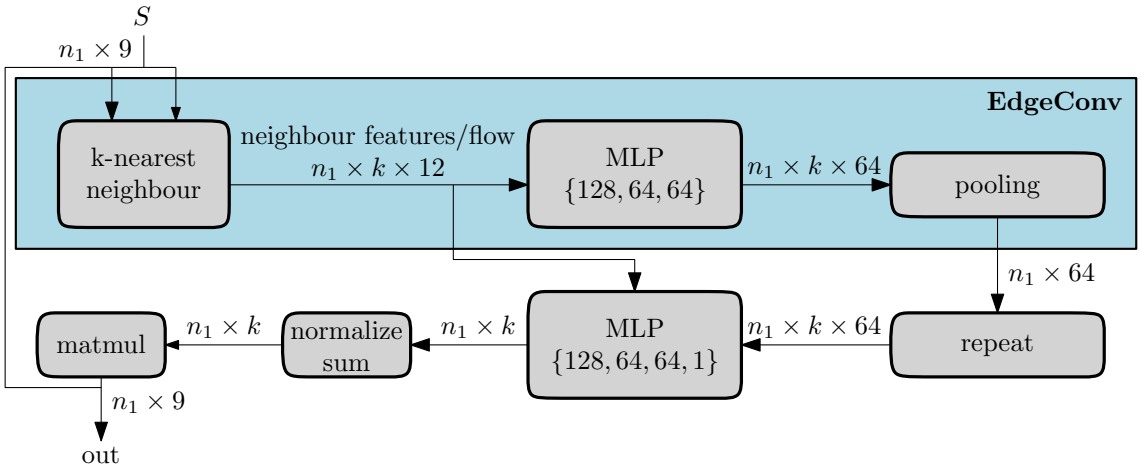


Figure 3.5: Flow refinement module architecture The blue-shaded area represents the EdgeConv module from the Dynamic Graph CNN architecture (29), which is used to learn features.

3.5 Upsampling

Our architecture so far is capable of estimating scene flow, but due to $\mathcal{O}(n^2)$ scaling can only do so at relatively low spatial resolution. As discussed in Section 3.3, we therefor downsample the input point clouds to a spatial resolution of 2048 points. In order to obtain a high resolution scene flow estimation, we will thus need to upsample back our low resolution estimation. We do this through 3D interpolation (28). Given a high resolution point cloud $\mathcal{P} = \{p_1, ..., p_{N_p}\}$, the downsampled low resolution point cloud $\mathcal{P}' = \{p'_1, ..., p'_{N_{p'}}\}$, and the low resolution scene flow estimation $\mathcal{S}' = \{s'_1, ..., s'_{N_{p'}}\}$, we calculate the upsampled scene flow estimation $\mathcal{S}$ as described in Equation 3.3. Here w is a normalized inverse-distance weight function, assigning higher weights to points that are closer.

$$\mathcal{S} \leftarrow \left\{ \sum_{p'_j \in knn(p_i)} w(p_i, p'_j) * s'_j \right\}_{i=1}^{N_p} \tag{3.3}$$

In other words, to obtain the scene flow estimation for some point $p_i \in \mathcal{P}$, we take a weighted average of the scene flow estimations of all points from $\mathcal{P}'$ that are in the k-nearest neighbourhood of p_i, assigning higher weights to points that are closer to p_i. Figure 3.6 shows an example of this upsampling.

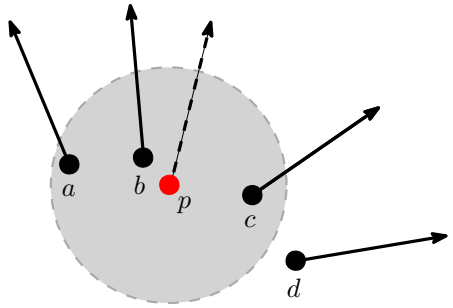


Figure 3.6: Example of flow upsampling In this Figure we illustrate how the upsampling of the point p might occur when using $k = 3$. The solid arrows represent that low resolution scene flow estimations, the dashed arrow the scene flow estimation resulting from the 3D interpolation. Here points a, b, and c fall in the 3-nearest neighbourhood of p, and will thus be considered for the interpolation, while point d falls outside of the neighbourhood, and will not be considered. As point b is closest to p, it will have the largest influence on the final estimated scene flow for p. Point a and c are both approximately as far away from p, and will thus have approximately equal influence, but less than point b.

3.6 Neighbour snapping

Our architecture so far already outputs an estimated scene flow for the input points clouds, which can be used to perform the temporal interpolation. One shortcoming of this estimated scene flow is that in almost all cases it is not fully-connecting (for definition, see Section 2.1). An example of such a non-connecting scene flow estimation can be seen in Figure 3.8a.

When used for temporal interpolation, this non-connectingness of the scene flow estimation can have negative effects on the quality of the produced output sequence. The problem occurs between a last interpolated frame and the subsequent source frame. If the

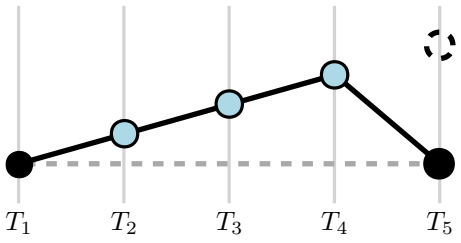


Figure 3.7: Necessity of neighbour snapping This Figure shows the position of a point (vertical axis) over time (horizontal axis). T_1 and T_5 are source frames, while T_2, T_3, and T_4 are interpolated frames. The dashed black point shows the prediction based on the estimated scene flow at T_5. The dashed line shows the ground truth scene flow. The motion between T_4 and T_5 is drastically different from the motion between prior frames.

scene flow is not fully-connecting, the motion between these two frames might significantly deviate from the motion over the last few frames, which in our experience is noticeable during viewing, and might be perceived as choppy and not smooth. We illustrate an example of this behavior in Figure 3.7.

To lessen this issue, we propose neighbour snapping, a technique that makes an existing scene flow estimation fully-connecting, in order to provide more smoothness in the temporal interpolation. The pseudo-code of the algorithm is shown in Algorithm 2.

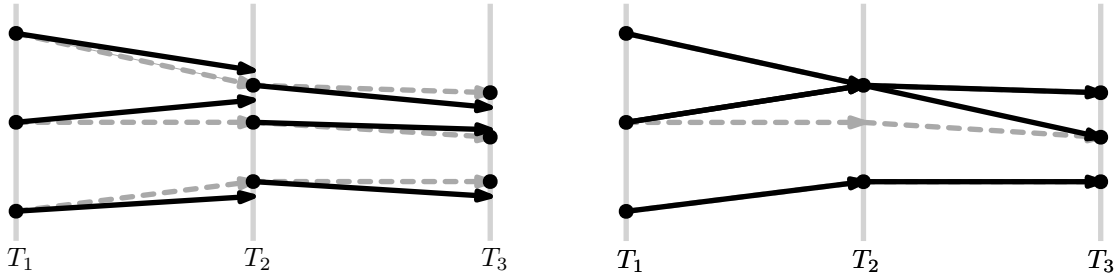
Neighbour snapping works as follows. Assume two point clouds $\mathcal{P}^i$ and $\mathcal{P}^{i+1}$ and their estimated scene flow $\mathcal{F}^{i \to i+1}$. To achieve forward-connectingness, we find for each point in $\mathcal{P}^i$ the point in $\mathcal{P}^{i+1}$ closest to its scene flow target location, and let that point be the new scene flow target location. This happens on Line 2-6 in Algorithm 2. In order to ensure backward-connectingness, we find all points in $\mathcal{P}^{i+1}$ that are not backward-connecting yet, and we change their spatial location to that of the nearest point in $\mathcal{P}^i$. The scene flow for these points is adjusted accordingly so that the target location remains the same. These adjustments happen on Line 7-11 in Algorithm 2.

Figure 3.8 shows an example of how neighbour snapping can make an existing scene flow estimation fully-connecting. In 3.8a none of the points are forward- or backward connecting. By applying Algorithm 2, all points are now fully-connecting. Notice that some points might be mapped inconsistently in regard to their ground truth, as happens to the middle point in Figure 3.8b at T_2. This inconsistency is in our experience not noticeable, and will be quickly corrected in future frames. On the other hand, the increased

Algorithm 2 Neighbour snapping

```
1:  procedure SNAP(pc1, pc2)
2:      for point p1 in pc1 do
3:          target ← p1.xyz + p1.sceneflow
4:          target_nn ← nearest_neighbour(pc2, target)
5:          p1.sceneflow ← target_nn.xyz − p1.xyz
6:      end for
7:      for point p2 in pc2 do
8:          nn ← nearest_backward_connecting_neighbour(pc2, p2)
9:          p2.sceneflow ← p2.xyz + p2.sceneflow − nn.xyz
10:         p2.xyz ← nn.xyz
11:     end for
12: end procedure
```

smoothness is in our experience highly noticeable, as points no longer appear to be jumping all over the place.



(a) Without neighbour snapping

(b) With neighbour snapping

Figure 3.8: Neighbour snapping These Figures show the position of a number of points (vertical axis) over time (horizontal axis). Specifically, (a) shows a scene flow estimation that is not fully-connecting, while (b) shows the same scene flow estimation after neighbour snapping has been applied.

4

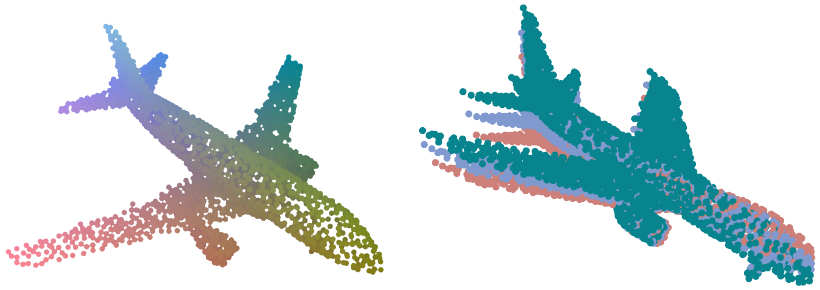
Results

In this Chapter we evaluate the performance of our interpolation architecture. To this end, we first introduce in Section 4.1 a data set we have created for training and evaluation of our architecture. In Section 4.2 we present our results, in the form of visual results, objective metrics, and a small-scale user study. In Section 4.3 we then proceed to further analyze these results.

4.1 Data sets

In order to train- and evaluate our network, we require a sufficiently large data set. It is well known that convolutional neural networks in particular demand a vast amount of training data during the training process. In Section 2.8 we have taken a brief look at available dynamic point cloud data sets. Unfortunately, these publicly available data sets are small in size, and they would not be sufficient to train our network to a point where it would achieve satisfactory results on unseen data. For this reason, we have created two synthetic data sets. The first data set consists of a series of rigid transformations on models from the Modelnet40 (32) data set, the second data set consists of a series of animated human bodies, generated from meshes obtained from Adobe Mixamo (76). Apart from the size, another great benefit of these synthetic data sets is that we can extract ground truth surface normals and scene flow, which will be used extensively both for training our network and for evaluating our results.

- **Synthetic - Rigid** This data set is based on the Modelnet40 (32) data set. Modelnet40 does not contain any color data, so we first add color values to each point based on its spatial coordinates. Next, we apply a series of rigid transformations to



(a) Single frame **(b)** Triple

Figure 4.1: Synthetic - Rigid data set Figure (a) shows one frame of an airplane. (b) shows a triple generated for training. In this figure each frame has been colored uniformly for illustrative purposes, in the data it has similar colors to the model shown in (a).

each object, resulting in an animation. Each transformation has a number of parameters controlling the magnitude and speed of the transformation. For each such parameter we specify a minimum- and a maximum value. We then randomly select a model from the Modelnet40 data set, randomly sample parameters within the specified ranges, and then use these to create a triplet of three consecutive dynamic point cloud frames. In Table 4.1 we provide an overview of the used transformations and their parameters. These used ranges have been empirically selected. Note that since we train using scene flow error as loss function, we only need to use two frames of the triplet. When using point cloud distortion metrics as loss function, all three frames are required.

- **Synthetic - Human** This data set consists of animated human bodies. For the creation of this data set we use the online service Mixamo (76). In Mixamo you can select one of a number default character models (or upload your own), and apply one of the many available animations. Next we use Blender (77) to render the animations, providing us with one mesh per frame. Finally, we convert these meshes to point clouds by randomly sampling points from the faces of the mesh. For instructions on how to use our data set, refer to our Github repository. For more details on how the data set was created, see Appendix A. Figure 4.2 shows a number of example frames from this data set. The training data set consists of 5 models, each having 12 animations applied them, for a total of 60 sequences. The test data set contains an additional 8 sequences. Each sequence ranges in length from 30 to 250 frames. In total the data set consists of 10 590 frames.

Parameter	Description	Min	Max
CIRCLE_RADIUS	radius of the translation spiral.	0.7	2.0
CIRCLE_HEIGHT	height gained when one translation spiral is finished.	2.0	5.0
FRAMES_PER_CIRCLE	frames it takes to complete one translation spiral.	50	80
FRAMES_PER_ROTATION_X	frames per full rotation on X-axis.	50	80
FRAMES_PER_ROTATION_Y	frames per full rotation on Y-axis.	50	80
FRAMES_PER_ROTATION_Z	frames per full rotation on Z-axis.	50	80
FRAMES_PER_SHRINK	frames to perform one shrink cycle.	50	80
SHRINK_FACTOR	maximum shrink factor during a shrink cycle.	1.3	2.2
FRAME_INDEX	independently sampled for each transformation.	0	360

Table 4.1: Synthetic - Rigid parameters This table shows the parameters that are used to generate our Synthetic - Rigid data set. For each parameter a value is randomly sampled between the Min. and Max. value. In the case of FRAME_INDEX, a new value is sampled for each transformation.

(a) Remy **(b)** Regina **(c)** Stefani

Figure 4.2: Synthetic - Human data set examples.

Data augmentation

In order to further increase the size and diversity of training set, we apply data augmentation. For the augmentation operations Flip, Rotation, and Scale we randomly sample

one set of parameters per frame pair, and then apply these operations with the same parameter to each of the two frames in the triple. The `Shuffle` and `Subsample` operations are independently applied to each individual frame.

- `Flip` With a probability 0.5 we flip the point clouds horizontally. With an independent probability 0.5 we flip the point clouds vertically.
- `Rotation` For all three axes we uniformly sample a degree between 0° and 360°, and rotate the point clouds with these degrees.
- `Scale` We randomly sample a scale factor between 0.9 and 1.1, and scale the point clouds with this factor.
- `Shuffle` We shuffle the order of the points in the point cloud randomly.
- `Subsample` We randomly sample down each point cloud to the desired number of points.

4.2 Evaluation

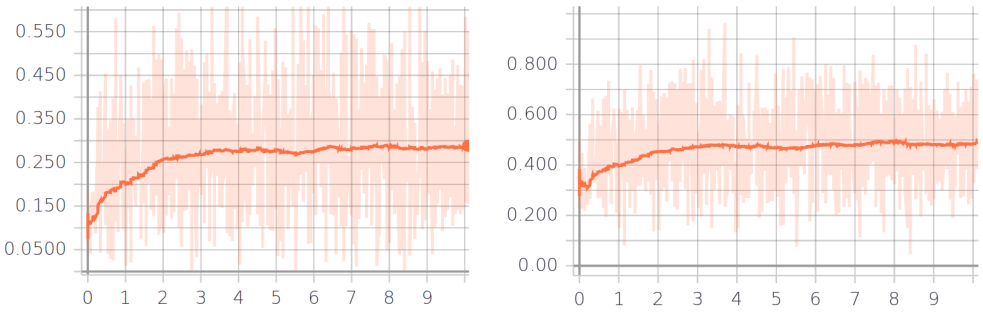
In this Section we present an evaluation of our architecture. In Section 4.2.1 we discuss our software implementation and the hardware that we used. In Section 4.2.2 we proceed to detail our training process and the training metrics that we monitored during training. In Section 4.2.3 we show the runtime performance of the system. We then evaluate the qualitative performance of our architecture in a number of ways. In Section 4.2.4 we give a visual demonstration of the output of our architecture. In Section 4.2.5 we evaluate our architecture using a number of objective metrics. Lastly, in Section 4.2.6 we discuss the results of a small-scale user study that we have carried out.

4.2.1 Implementation

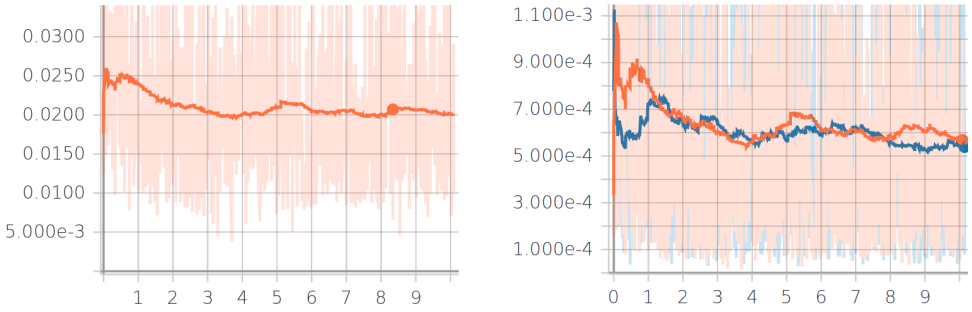
We have implemented our architecture in Python using the Tensorflow (78) framework. Our implementation is publicly available on Github at `https://github.com/jelmr/pc_temporal_interpolation`. It contains modules for training, evaluation and inference. For more information on how to run the architecture, please see the README file in this repository. Additionally, instructions are included for how to recreate our data set. All experiments in this Chapter have been carried out on machine equipped with an Intel i7-7800X CPU running at 3.50 GHz, 16GB RAM, and an NVIDIA GeForce RTX 2080 Ti.

4.2.2 Training

As detailed in Section 3.4, we train our network in two phases. First we only train our point matching layer. In the second phase we freeze the weights from the point matching layer, and train the flow refinement module. Figure 4.3 shows the training metrics for the first phase of the training process, based on the output of the point matching layer. Figure 4.4 shows these metrics for the second phase, based on the output of the flow refinement layer. The model is trained on the Synthetic Human data set (see Section 4.1). This data set consists of 10522 pairs of frames. We split the data set in training, validation and test sets. The training set (8152 of pairs) is used to fit the model. The validation set (500 pairs) is used to monitor the performance of network during training. Lastly, the test set (1870 pairs) is used to generate the results we present in this Section. For the test set we only use models and animations that are not present in the training- or validation set.

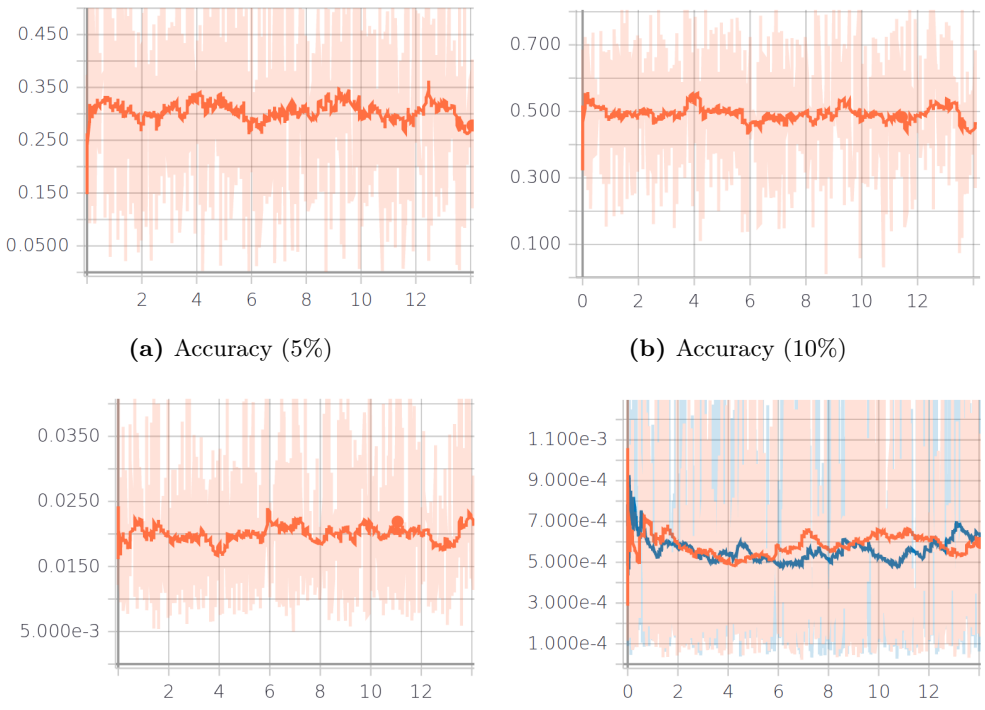


(a) Accuracy (5%) **(b)** Accuracy (10%)



(c) End point error **(d)** Loss

Figure 4.3: Training metrics for phase 1 This Figure shows objective metrics during the first phase of training (training the point matching module). All metrics are calculated over the validation set, except for the blue line in (d), which is calculated over the training set itself. End point error is the L_2 distance between the ground truth- and estimated scene flow. The accuracy is the portion of points for which the end point error is below a certain percentage of the ground truth scene flow.



(a) Accuracy (5%)

(b) Accuracy (10%)

(c) End point error

(d) Loss

Figure 4.4: Training metrics for phase 2 This Figure shows objective during the second phase of training (training the flow refinement module). See Figure 4.3 for an explanation of the metrics.

4.2.3 Runtime performance

Table 4.2 shows the runtime performance of various stages of our architecture.

Stage	Run time	Frame rate
Downsample	1.92×10^{-4}s	5.21×10^{3}
Interpolation network	6.55×10^{-2}s	15.3
Upsample	0.520s	1.92
Neighbour snapping	0.266s	3.76

Table 4.2: Runtime performance This table shows the runtime for various stages of the architecture. For stages that process data in batches, this batch processing time is amortized over the individual frames. The frame rate considers how many frames could be processed per second if the entire machine was dedicated to that stage only.

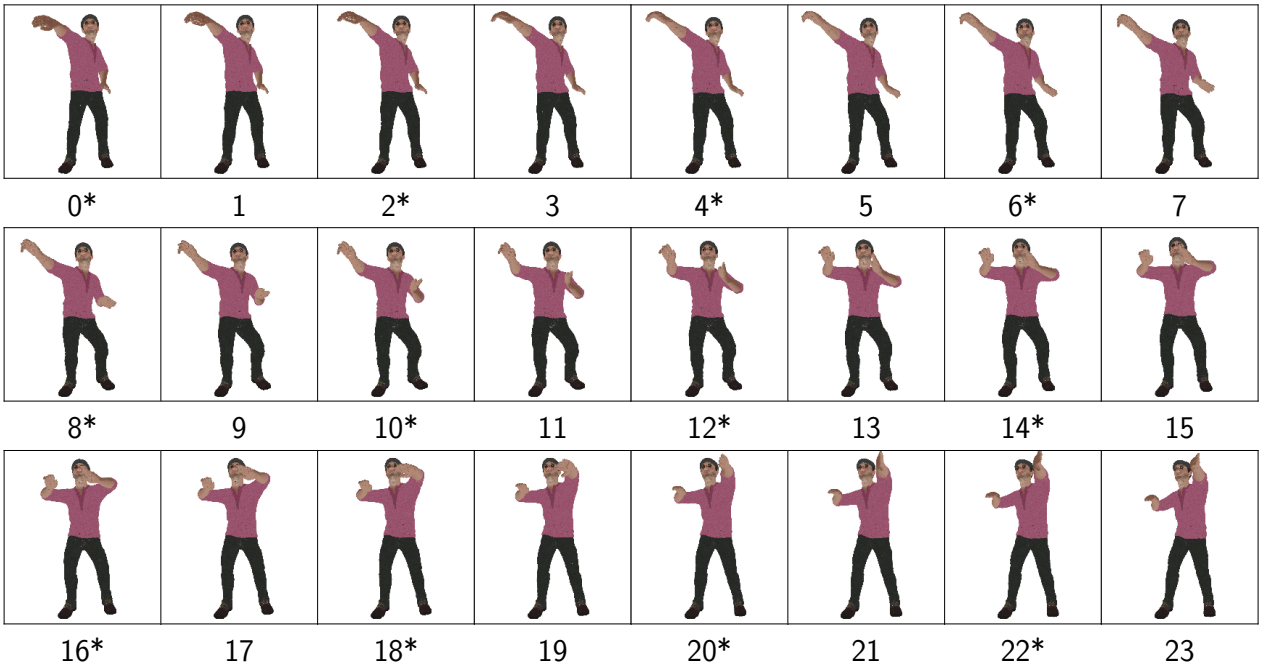
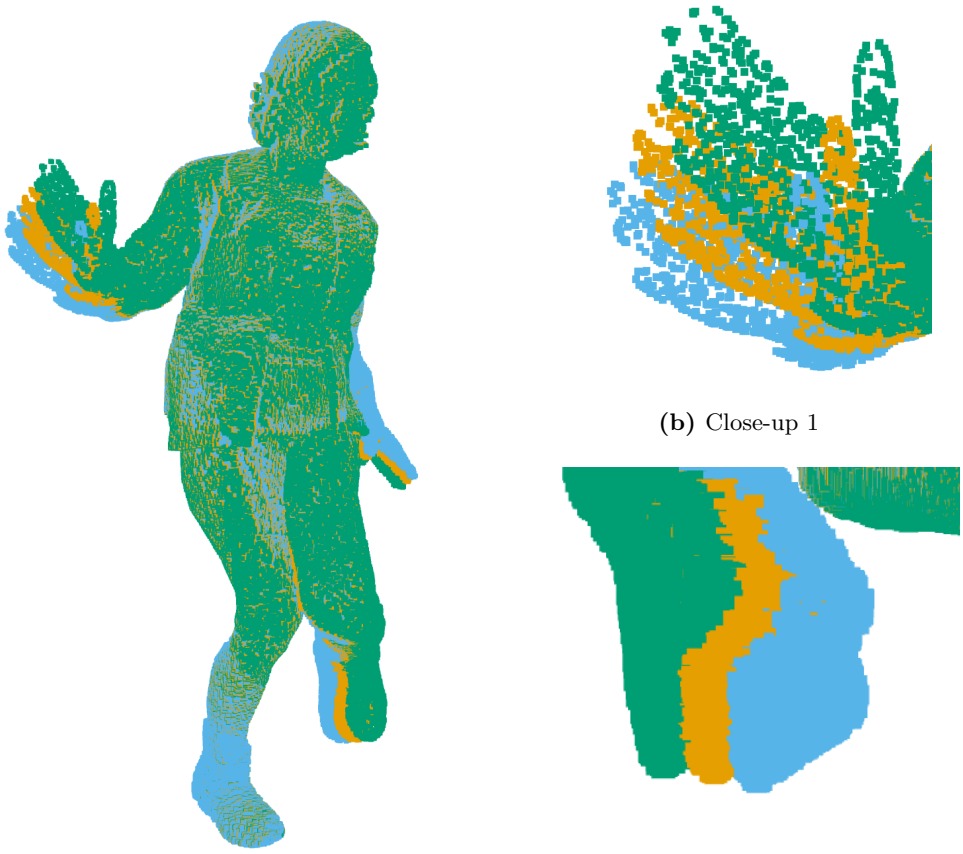


Figure 4.5: Example of interpolated sequence. This Figure shows an example of a dynamic point cloud that has been interpolated using our architecture. Frames marked with an asterisk are the original input frames, unmarked frames have been interpolated by our architecture.

4.2.4 Visual results

In this Section we give a visual demonstration of the interpolation results of our architecture. Video demonstrations can be found on our Github repository. Figure 4.5 shows a short sequence from our evaluation set that has been interpolated using our architecture. Because it can be hard to spot small motions in a sequence like this, we have only interpolated one frame in between each pair of frames. Our Github page contains a video of the same sequence, where in between each pair of frames, 5 new frames have been generated.

It can be hard to compare the motion across the separate frames in Figure 4.5. To make comparison easier, we have in Figure 4.6 overlapped two source frames together with the resulting interpolated frame. It can be seen that in these frames, the interpolated frame falls nicely in between the two surrounding frames, which is the desired result.



(a) Full

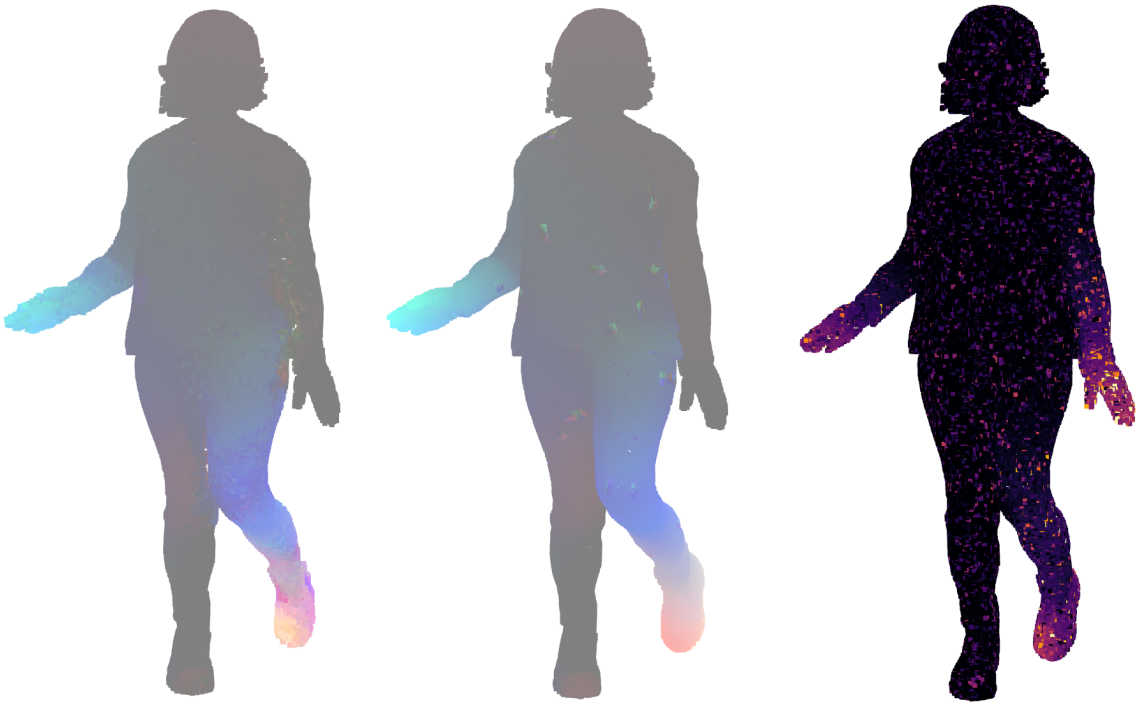
(b) Close-up 1

(c) Close-up 2

Figure 4.6: Interpolated frame comparison (a) shows an example of an interpolated frame compared to the two source frames. Each frame has been colored uniformly, with source frame 1 being blue, source frame 2 being green, and the interpolated frame being orange. (b) and (c) shows close-ups of high motion areas of the same frames (at different angles).

In Figure 4.7 we show an alternative visualization of the scene flow estimation generated by our architecture. In Figure 4.7a and Figure 4.7b we use a color mapping to visualize the scene flow both for our interpolation result and for the ground truth. In Figure 4.7c we show the error between the interpolated result and the ground truth using a heatmap. In this Figure, only one frame is shown. Figure 4.8 shows the scene flow error for a short sequence of frames. Lastly, a video showing this visualization for an entire sequence can be found on our Github page.

As can be seen in these Figures, the largest scene flow estimation errors seem to occur mostly in the hands and feet. These regions are also the regions where the largest motion occurs in the sequences we use. This suggests there might be a correlation between motion distance and scene flow error.



(a) Interpolation **(b)** Ground truth **(c)** Error

Figure 4.7: Scene flow visualization (a) and (b) show scene flow of the interpolation generated by our architecture and the ground truth, respectively. In these Figures, the direction of the scene flow is shown through a color coding. A scene flow indicating no motion is represented using a gray color. Motion in x-, y-, and z-axis are then mapped to variation in red-, green- and blue color spectrum, respectively. Thus, if two points have a similar color, that means they are moving in a similar direction, allowing for easy visual comparison of the scene flow estimation. Larger motions correspond to brighter- or darker colors. (c) shows the error between the interpolation result and ground truth. Here error ranges from black (no error), through purple (small error), through bright yellow (large error). 'Large error' is here defined as being equal or larger to 50% of the largest ground truth motion throughout the sequence.

4.2.5 Objective metrics

For a more objective evaluation, we also assess our architecture using a number of objective metrics. We use the following metrics:

- `EPE:` The end point error (EPE) is the average L_2 distance between the estimated- and ground truth scene flow in centimeters. Lower is better.
- `Accuracy` The accuracy is the portion of points for which the end point error is less than a certain threshold. We report accuracy for 0.5, 1.0, and 2.0 cm. Accuracy ranges from 0.0 to 1.0, with a higher score being better.

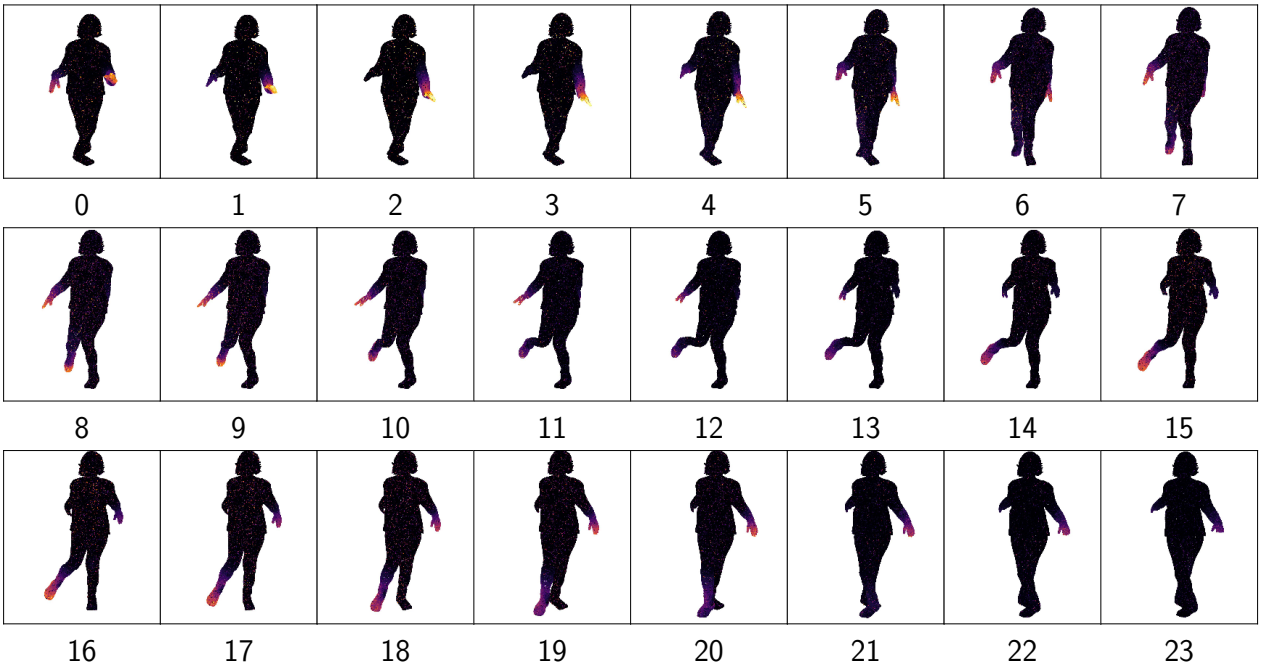


Figure 4.8: Error of interpolated sequence. Each frames shows the scene flow error between our interpolation result and ground truth. Error is indicated through a heatmap, ranging from black no error), through purple (small error), through bright yellow(large error). To improve visibility, in this Figure 'large error' is defined as being equal or larger to 25% of the largest ground truth motion throughout the sequence.

- `po2point_xyz_PSNR:` The peak signal-to-noise ratio (PSNR) of the spatial component (xyz) of the point-to-point distortion metric (see Section 2.7). A higher score means the estimation is closer to the ground truth, and is thus better.

- `po2point_rgb_PSNR:` Same as `po2point_xyz_PSNR`, but this is the color component of the point-to-point distortion metric.

- `po2plane_PSNR:` The peak signal-to-noise ratio (PSNR) of the point-to-plane distortion metric (see Section 2.7). Note that this metric only considers spatial similarity, and does not factor in color in any way. A higher score means the estimation is closer to the ground truth, and is thus better.

- `VIF (projection):` This is the projected distortion calculated using the VIF distortion metric (see Section 2.7). It ranges from 0.0 to 1.0. Again, a higher score is better.

In all cases, the metrics are calculated by comparison against ground truth. We show results for the following algorithms:

- **Ours (snap):** The output of our full architecture, including neighbour snapping.
- **Ours (no snap):** The output of our architecture but without applying neighbour snapping.
- **Flownet3D (snap):** The Flownet3D (22) architecture. The network has been retrained using our synthetic data set (the same data set that is used to train our own architecture). For the high resolution evaluation data set, we find that Flownet3D cannot process the large point clouds in reasonable time, and thus take the following approach: we use our full architecture, but replace our own interpolation network with Flownet3D. This means Flownet3D takes as input point clouds downsampled to 2048 points, and the generated output is then upsampled, and neighbour snapping is applied.
- **No interpolation:** No interpolation is performed. For the metrics based on scene flow, the scene flow estimation is set to 0. For the other metrics, instead of generating an interpolated frame, the last input frame is simply used.

As evaluation set, we use 8 sequences from our synthetic data set. These sequences consist of 8 unique animation applied to 2 different models. None of the models or animations in the evaluation set are used in the training- or test set during the training process. We show the results both for the high resolution version of the data set (100k points, Table 4.3), and the low resolution downsampled version (2048 points, Table 4.4). For each pair of input frames we interpolate 4 frames. For each frame we then calculate the metric values, and average these out over all frames to get to the values reported in Tables 4.3 and 4.4. For the low resolution version, down- and upsampling is not required, so `Ours` is simply our interpolation network, and `Flownet3D` is simply the original Flownet3D architecture, without any upsampling or neighbour snapping. We do not report the VIF projection metric for the low resolution version, due to the difficulty of making informative projections of point clouds that are of this low resolution.

Metric	Ours (no snap)	Ours (snap)	Flownet3D (snap)	No interpolation
EPE	1.533	1.607	2.353	**1.294**
Accuracy (0.5 cm)	0.323	0.287	0.084	**0.351**
Accuracy (1.0 cm)	0.522	0.493	0.205	**0.546**
Accuracy (2.0 cm)	0.753	0.738	0.500	**0.806**
po2point_xyz_PSNR	58.35	**58.56**	56.01	56.18
po2point_rgb_PSNR	73.75	73.72	71.57	**74.93**
po2plane_PSNR	104.75	106.7	**136.5**	94.19
VIF (projection)	0.809	**0.838**	0.745	0.710

Table 4.3: Objective metrics (100k points)

Metric	Ours	Flownet3D	No interpolation
EPE	2.584	2.986	**2.579**
Accuracy (0.5 cm)	0.202	0.145	**0.243**
Accuracy (1.0 cm)	0.336	0.290	**0.348**
Accuracy (2.0 cm)	**0.566**	0.505	0.548
po2point_xyz_PSNR	**53.03**	47.64	48.07
po2point_rgb_PSNR	**81.81**	76.25	76.42
po2plane_PSNR	**90.73**	80.84	80.32

Table 4.4: Objective metrics (2048 points)

4.2.6 User study

In order to get a subjective evaluation of our results, we conduct a small-scale user study ($n = 8$). In the study we asked participants to rate the quality of the motion of a number of pre-rendered dynamic point cloud sequences. All videos we used can be found on our Github page. In this Section we first discuss the setup of this user study, then we present the results.

User study setup For this study we prepared 36 pre-rendered dynamic point cloud sequences. The sequences are divided equally into four categories:

- **Ground truth:** The original sequence at high frame rate (48 FPS).
- **Low FPS:** The original sequence at a low frame rate (8 FPS).
- **Ours (interpolated):** The result of our architecture after interpolating the low frame rate sequence (48 FPS).
- **Flownet3D (interpolated):** The result of the Flownet3D (22) architecture, with our upsampling and neighbour snapping applied in order to get back high resolution point clouds (48 FPS).

Participants are handed three forms. The first form is an Informed Consent Form (Appendix B.1). The sheet explains briefly the goal of the experiment and the procedure, and asks participants to give their consent. The second form is the Participant Information Form (Appendix B.2). In this form we ask the participant for their age, and whether they have any prior experience in visual quality evaluations. The last form is the Video Rating Form (Appendix B.3), on which the participants can fill in their rating for each sequence.

Participants are first asked to fill in and sign the Informed Consent- and Participant Information Forms. Participants are then put in front of a laptop, where we start the demonstration session. Using a simple interface (Appendix B.4, Figure B.1a) the users are shown four short video clips, one of each category. The goal of this demonstration is to make the user familiar with the viewing interface, and to give them an idea what the range of quality is between different sequences. After the demonstration we move on to the video rating session, where each participant rates the remaining 32 sequences. Participants are asked to rate the quality of the motion on the Video Rating Form between 1 (terrible) and 7 (excellent). We instruct the participants to try and not factor in the quality of the model and textures itself, but only the quality of the motion. Using a similar interface (Appendix B.4, Figure B.1b), the users are shown the remaining 32 sequences in a randomized order. Whenever one sequence finishes playing, the next sequence is loaded, but paused. There is thus no opportunity to watch a sequence more than once (we are interested in their first impression). Once the participant has rated a sequence, they can press the play button to start the next sequence. Once all 32 sequences have been rated this way, the experiment is concluded.

User study results We now present the results of the user study that we conducted. In Figure 4.9 and Figure 4.10 we show the distribution of ratings given by participants.

In Figure 4.11 and Figure 4.12 we compare the ratings given for the four variants of each sequence. We further analyze and interpret these results in Section 4.3.

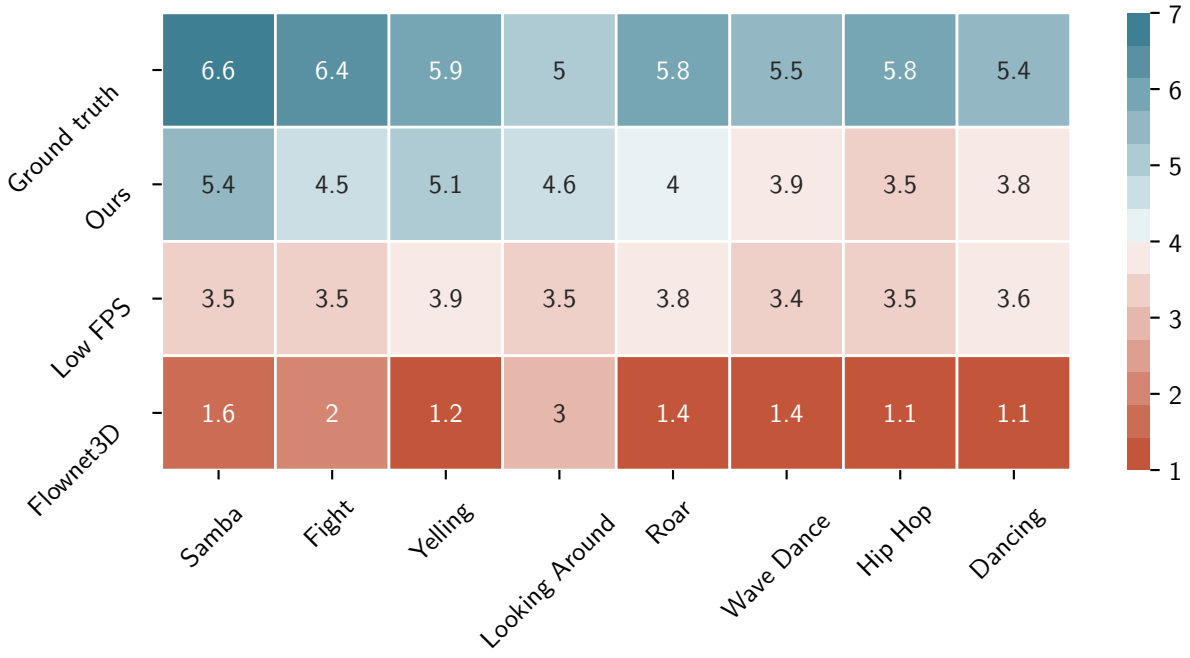


Figure 4.9: Mean rating per sequence This Figure shows for each variant of each of the eight sequences the mean rating assigned to it by the participants in our user study.

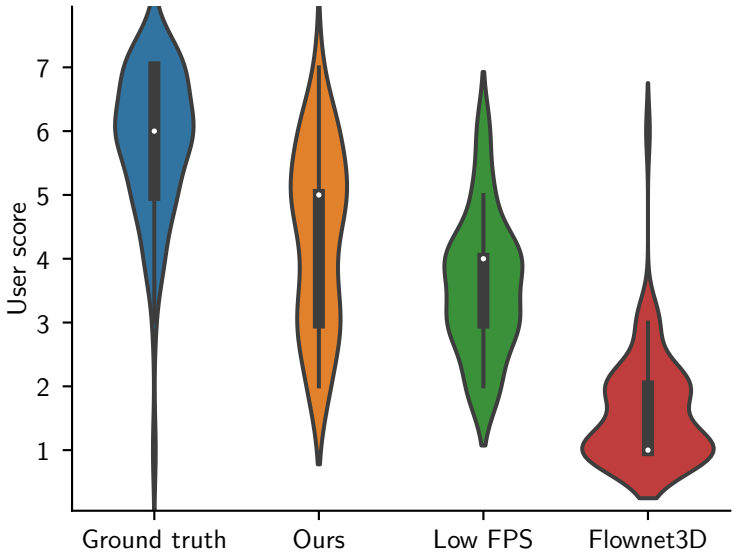


Figure 4.10: Rating distribution per variant In this Figure the distribution of ratings is shown, aggregated by variant.

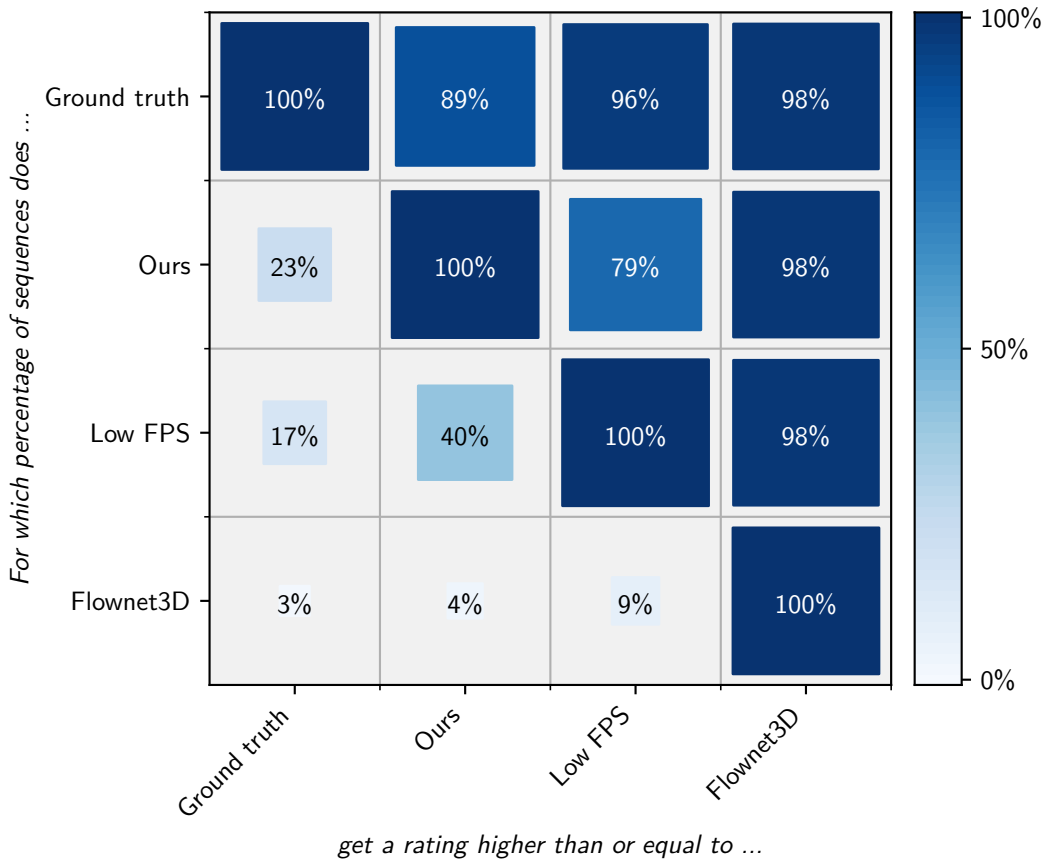
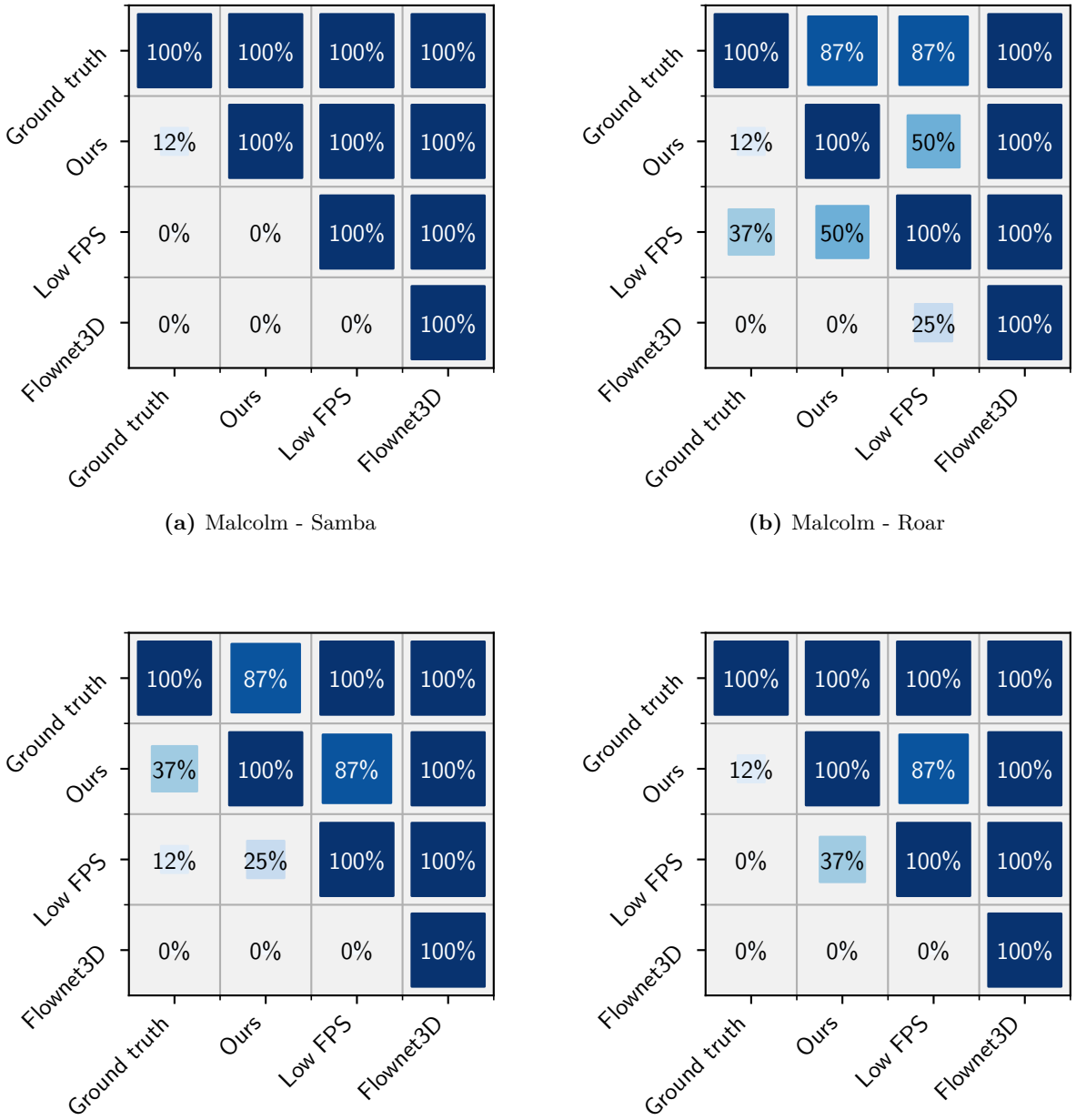


Figure 4.11: User study variant matrix (overall) This Figure shows for each variant (*Ground truth*, *Ours*, *Flownet3D*, *Low FPS*) for what percentage of the sequences it is rated higher than or equal to other variants (this calculation is done per sequence per participant, we do not average it first). This percentage is additionally encoded in the size- and color of the squares. For example, we can read that *Ground truth* is rated at least as high as the *Low FPS* variant for 96% of the sequences, or that our architecture gets a score at least as high as the *Low FPS* variant for 79% of the sequences.



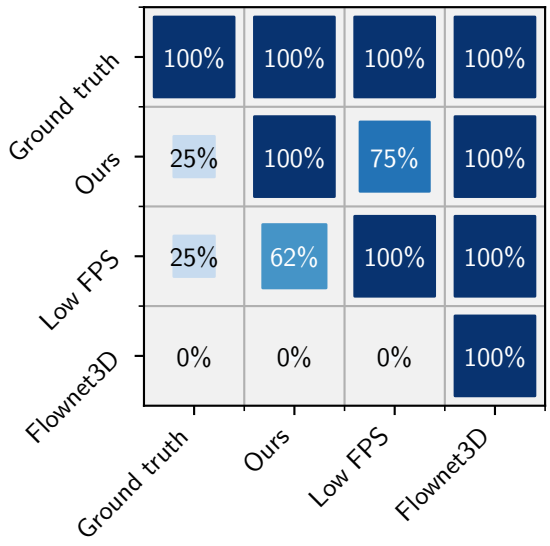
(a) Malcolm - Samba

(b) Malcolm - Roar

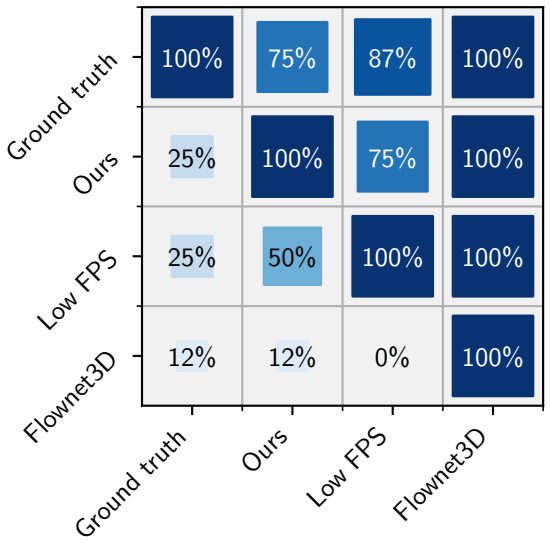
(c) Malcolm - Yelling

(d) Malcolm - Fight

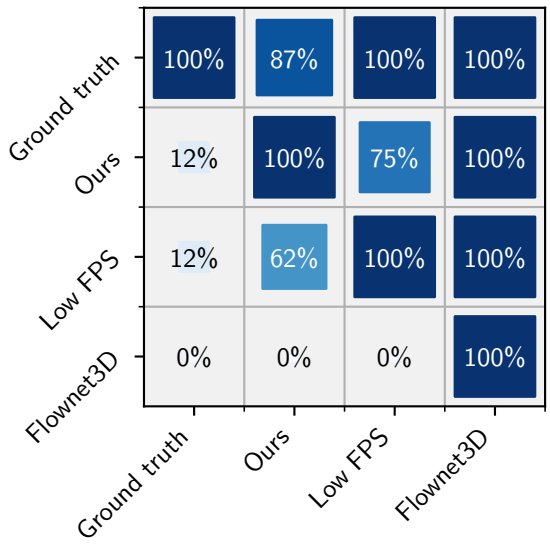
Figure 4.12: User study variant matrix (per sequence This Figure shows a variant matrix similar to Figure 4.11, see that Figure for a description on how to interpret this Figure. Here we show *per sequence* the percentage of participants that rated one variant at least as high as another variant.



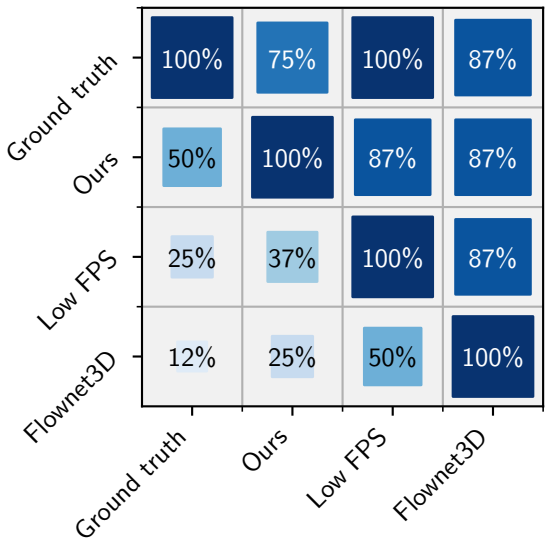
(e) Shae - Dancing



(f) Shae - Wave Dance



(g) Shae - Hip Hop Dancing



(h) Shae - Looking Around

4.3 Analysis

In this Section we provide an analysis of the results presented earlier in this Chapter. We first comment briefly on the runtime performance of the architecture. Next we discuss the visual results, and finally we look into the correlation between the objective metric results and the outcome of our user study.

Runtime performance Table 4.2 shows the runtime performance for various stages of the architecture. Downsampling is very fast, and does not take significant time compared to the other stages. The interpolation network itself is reasonably fast, and could be considered being real-time. It is able to process 15 frame pairs per second, which should be enough for normal use cases. The upsampling and neighbour snapping stages are currently the bottlenecks, and can only run at a little under 2 to 4 FPS. To make the architecture as a whole capable of performing real-time interpolation, significant improvements will have to be made in these two stages. We discuss the topic of real-time interpolation in more detail in Section 5.1.4.

Visual results In Figure 4.7, we compare the scene flow estimated by our network with the ground truth scene flow. Overall, our network seems to estimate similar scene flow to the ground truth, as can be seen by the similarity between Figures 4.7a and 4.7b. In Figure 4.7c we can more easily see the error between the two. When looking at the sequence displayed in Figure 4.8 or the corresponding video on our Github page, it becomes evident that larger errors tend to occur in regions where the motion per frame is large, for example in the hands and feet. When the motion is smaller, the architecture estimates the scene flow with little error. Another interesting observation is that the ground truth scene flow data has some small artifacts as well. This can be seen in Figure 4.7b in the form of small deviating triangles, especially present in the chest area. Our interpolation network learns to ignore these artifacts, and does not reproduce them.

Objective metrics & User study The objective metrics and the results from the user study tell a very different story, even though the metrics have been calculated over the same dynamic point cloud sequences that were used during the user study. For the metrics based on scene flow error (`EPE` and the various versions of `Accuracy`) the clear winner is `No interpolation`, with the results from `Ours` a little bit behind. The values for the point-to-point metrics are very close, with `Ours` being slightly ahead on the spatial component, and `No interpolation` again being ahead on the color component. For the

point-to-plane metric `Flownet3D` wins by a large margin. `Ours` comes ahead by a decent margin in the VIF projection metric, which has been reported by Torlig et al. (71) to have the strongest correlation with subjective user ratings on point clouds of human bodies.

It might be considered surprising that `No interpolation` (which simply repeats the last frame) would achieve the highest score for so many metrics. A possible explanation for this is the following. All aforementioned metrics (except the VIF projection metric) calculate the average of some notion of error over all points. The majority of the points move very little (as can be seen in Figure 4.7b). By always predicting a scene flow of 0, like `No interpolation` does, a low error will be achieved for these points. For the smaller number of points that *do* move far, a higher error will be assigned, but the mean will be dominated by the low error scores from the many low-motion points.

For the user study the results are more coherent. From Figures 4.9 and 4.10 it is evident that `Ground truth` receives on average the highest ratings, followed by `Ours`, then `Low FPS`, and then finally `Flownet3D`. This is also the conclusion we arrive at when analyzing Figure 4.11. `Ground truth` is clearly the highest rated, and is only beaten in a small number of sequences (11% for `Ours`, 4% for `Low FPS`, 2% for `Flownet3D`). Next is `Ours`, which is generally rated a bit below `Ground truth`, but rated equal or preferable to `Low FPS` for 79% of the sequences. `Flownet3D` is clearly the lowest rated in almost all sequences.

When we look at Figure 4.12, we can see that the results vary a bit per sequence. For example, the `Malcolm - Samba` sequence (Figure 4.12a) exhibits an almost perfect hierarchy of `Ground truth` over `Ours` over `Low FPS` over `Flownet3D`, the only exception being that `Ours` is rated equal to `Ground truth` in 12% of the sequences. In the sequence `Malcolm - Roar` (Figure 4.12b), the interpolation created by our architecture has quite a lot of noticeable artifacts (which we suspect is due to the sequence having relatively large motion). As a result, `Low FPS` pulls ahead here. The final sequence we discuss is `Shae - Looking Around` (Figure 4.12h). This is a slow sequence with not a lot of motion. Supposedly due to this low amount of motion, both `Ours` and `Flownet3D` do a relatively good job interpolating the sequence without many noticeable artifacts. On the other hand, because the sequence is so slow, the lower frame rate from `Low FPS` becomes less noticeable. As a result, the ratings for this sequence are closer together, and less clear-cut.

From manually comparing the objective metrics with the user study results, it can already be seen that the results are not in agreement with each other. In Figure 4.12 we show the correlation matrix between the user scores and objective metrics using the Pearson Correlation Coefficient. We see that none of metrics have a particularly strong correlation with user scores. This is in accordance with findings of other research, such as

Torlig et al. (71). We see in particular that the various accuracy measures and end-point-error correlate poorly with the user score. The point-to-plane, point-to-point (spatial), and VIF projection metrics perform a bit better, but still leave much to be desired.

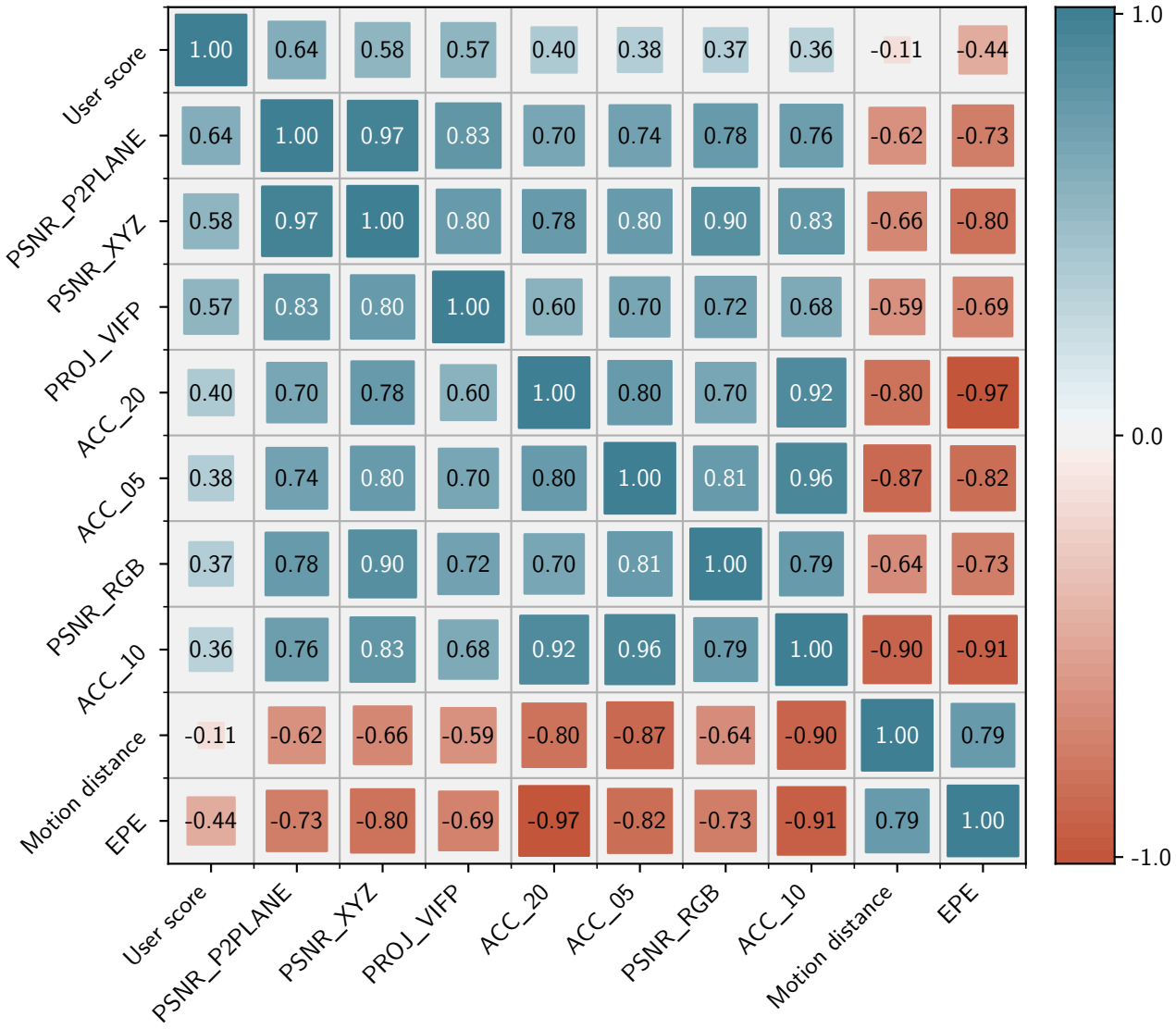


Figure 4.12: User study and objective metric correlation matrix This Figure shows the Pearson Correlation Coefficient for the objective metrics and user score. Ground truth sequences were excluded, because they by definition have a PSNR of ∞, and it is unclear what would be a suitable PSNR value to use instead for the correlation calculation. *Motion distance* is the average motion (ground truth scene flow) over the top 10% highest motion points per frame.

5

Conclusion

In this thesis we aimed to design and train an architecture capable of performing temporal interpolation of dynamic point clouds. We identified a number of Research Objectives that needed to be resolved in order to achieve this goal. We have then created- and implemented such an architecture, and have shown through a diverse evaluation that it is indeed capable of performing this interpolation. In this Chapter we will now first provide a discussion on the difficulties we encountered and the lessons we learned along the way, and remaining challenges (Section 5.1). Next, we make suggestions for how this work could be extended in future research (Section 5.2). Finally, we briefly revisit our Research Objectives and conclude (Section 5.3).

5.1 Discussion

Throughout this project we have had to overcome a number of difficulties and challenges to get to our end goal of performing temporal interpolation of dynamic point clouds. In this Section we share experiences and some of the insights that we have gained during this process. We also reflect on the limitations of our current approach, as well as make suggestions for future improvements.

In Section 5.1.1 we discuss the limited availability of dynamic point cloud sequences, and how this affected our research. In Section 5.1.2 we discuss our limited success with reusing existing neural network architectures, and the need to design new architectures for the problem of temporal interpolation of dynamic point clouds. In Section 5.1.3 we comment on the scaling properties of our architecture with regards to spatial resolution. Next, in Section 5.1.4, we discuss the runtime performance of our architecture, and the changes required to make the system capable of real-time interpolation. In Section 5.1.5 we look

into the performance of the network as a function of the magnitude of the motion in the input sequence. Then, in Section 5.1.6 we discuss the viability of point cloud distortion metrics as loss function. Lastly, in Section 5.1.7, we discuss the use of objective metrics for quality evaluation, and how these objective metrics correlate with user perception.

5.1.1 Limited availability of data sets

One of the first issues we encountered during this project is the limited availability of public dynamic point cloud data sets. We have only been able to find two high quality data sets, which in total span less than 90 seconds of footage. In order to train a convolutional neural network, such an amount of data is simply insufficient, which is why we have created our own synthetic data set. This synthetic data set has allowed us to train and evaluate our architecture, but it is not without downsides. The point clouds in our synthetic data set inevitably have different properties than real-world captured point clouds may have. One example is that in our data set, each point in a frame has a unique one-to-one mapping to another point in other frames (that is, each point has exactly one semantically corresponding point in other frames). In real-world data sets, this will generally not be the case. Another example is that in our data set, points will never change color, they will only be translated spatially, whereas in real-world data sets colors are likely to change, for example due to variable lighting conditions.

Neural networks will exploit these kind of properties during the training process, which is likely the reason that we see poor results when interpolating real-world data sets using our architecture. We identify two possibilities:

- Because the network is trained on synthetic data with different properties, it has not learned how to interpolate real-world point clouds. If the network were to be trained with a real-world data set, it *would* learn properly how interpolate real-world data.
- The network architecture itself is not suited to perform temporal interpolation on real-world data, and different architectures need to be explored.

Without access to sufficiently large real-world data sets, we cannot conclude which of these two possibilities holds true. In future work, we would like to further investigate and improve our architecture specifically with real-world data sets. To this end, we hope that larger public data sets will become available in the future. An additional challenge here is that for real-world data sets, ground truth scene flow data is likely not as easily available as it is for our synthetic data sets.

5.1.2 Reusability of network architecture

Transfer learning is a machine learning technique that reuses a model developed for one task as a starting point for completion of another task (79). Many tasks lend themselves well to transfer learning. In the context of point clouds, for instance, many classification- and segmentation solutions share the same core architecture, and vary only in the last couple layers (necessary because the outputs have different dimensions). Examples of such solutions are PointNet (27) and DGCNN (29).

Early on in this project, we investigated reusing such architectures to perform temporal interpolation of dynamic point clouds, but we had little success. We speculate this lack of success could be attributed to the following. Classification and segmentation are tasks that demand high level inference: the network will need to aggregate the local information of individual points into higher level features. After a number of aggregation steps, these higher level features can be used to classify- or segment the point cloud. In contrast, we suspect that in the problem of temporal interpolation it are mostly the local features that are of importance. The crux of the interpolation problem is point matching, which matches points from the one point cloud to semantically matching points in the other point cloud. To perform this matching, the network needs to find corresponding surfaces, which is for the most part a local problem. There can of course be a benefit to having high level features too, because certain types of surfaces might move and behave in different ways, but we suspect this is secondary to local features. It is then logical that architectures that focus on inference of high level features do not perform as good on the problem of temporal interpolation.

Even the Flownet3D (22) architecture, which has been designed for scene flow estimation, does not perform that well on our data set of human bodies. We suspect this is because Flownet3D has been designed and evaluated around the KITTI (80) and FlyingThings3D (81) data sets. The first is a data set of outdoor scenes for autonomous vehicles, the second is a data set consisting of many objects rigidly moving through a 3D space. Both are 2D data sets that have been converted to 3D point clouds. To perform well on these data sets, it is important to segment the different objects from each other. Once the objects have been segmented, the motion for each object is relatively rigid, and most points in an object will travel in the same direction. For these data sets, Flownet3D will thus learn to segment the objects and predict in which direction each object as a whole is moving. When we apply Flownet3D to our data set, we see this same behavior,

where it will attempt to move the entire human body in one direction, which leads to poor performance.

5.1.3 Scaling

The neural network modules used in this project and in related work rely on some form of nearest neighbour search. Such nearest neighbour search is $\mathcal{O}(n^2)$ with n input points, and thus scales poorly with the spatial resolution of input point clouds. High resolution dynamic point clouds can consist of hundreds of thousands of points per frame, which makes it computationally expensive to process them in the network directly, especially when taking into consideration the many training iterations required to fully train the network. In this project, we have circumvented this problem by downsampling the input point clouds to 2048 points, performing the scene flow estimation on these downsampled point clouds, and then upsampling them back to the original spatial resolution. The upsampling technique that we use still requires nearest neighbour search, but it only has to be performed once, and it is not part of the neural network training loop, making it quite manageable.

It could be advantageous to be able to perform the scene flow estimation on point clouds of higher resolution than 2048 points, as this would allow the network to make a more fine-grained estimation. It is likely such improvements would be especially noticeable in the facial region, which is currently represented by a relatively small number of points, while also exhibiting very complex motions. We see two approaches which could enable the scene flow estimation of higher resolution point clouds:

- Develop neural network modules for point cloud learning that do not rely on nearest neighbour search. Such modules would need to work fundamentally different than the modules presented in this thesis, and it is unclear how such modules would work.

- More efficient nearest neighbour search algorithms could be investigated. Our current implementation of k-nearest neighbour is a naive brute-force approach. Due to the algorithm being embarrassingly parallel, we achieve reasonable performance on a GPU. Still, we conjecture that it is possible to develop a more efficient implementation for our use case. One avenue to explore, for example, is to utilize techniques similar to cell linked lists (82). By setting a maximum neighbourhood distance, cell linked lists allow for more efficient nearest neighbour search.

5.1.4 Real-time interpolation

As we show in Table 4.2, our architecture is currently not capable of performing temporal interpolation in real-time. We now discuss the performance of the individual parts, and what kind of improvements need to happen to allow for real-time interpolation.

- **Uniform downsampling** This part of the architecture is fast, and does not pose an issue for real-timeness.
- **Interpolation network** The interpolation network runs at approximately 15 frames per second, which could be considered real-time. Do note that this frame rate is determined by amortizing the run time over the batch size, so to reach this frame rate during real-time streaming, frames will need to be buffered to fill the batches, adding some latency to the process. The performance of the interpolation network could be further improved with a more efficient nearest neighbour implementation (see Section 5.1.3), or by attempting to reduce the number of weights in the network.
- **Upsampling, neighbour snapping** The upsampling and neighbour snapping modules are the slowest parts of the architecture, by a significant margin. The techniques used for upsampling and neighbour snapping are very similar, and depend heavily on nearest neighbour search. In these modules, however, we have to deal with the high resolution point clouds (100k points) instead of the low resolution point clouds (2048 points), which makes in infeasible to apply the naive brute-force nearest neighbour search that we use in our interpolation network. Our current implementation builds a KD-tree of the low resolution point clouds, and iteratively queries this KD-tree for each point of the high resolution point cloud. This is a rather inefficient implementation, which additionally runs on only one thread on the CPU. We expect that by employing a more efficient algorithm (potentially on the GPU, or at least multi-threaded) we could achieve large speedups, potentially bringing the performance to real-time level.

Ultimately we believe that the system is not far away from being real-time. We expect huge speedups in the upsampling- and neighbour snapping modules with a more efficient nearest neighbour algorithm, which we consider to be mostly an engineering problem, rather than a research one. We expect that the performance of the interpolation network can be improved with a better nearest neighbour algorithm or by optimizing the number of weights in the network, but that the gains will be smaller than in the upsampling- and neighbour snapping modules.

5.1.5 Motion distance

Fundamentally, our interpolation network works by analyzing the k-nearest neighbourhood of each point, and learning a soft mapping between the base point and the neighbouring points. The idea is that the network will learn to select the true semantically corresponding point, and base its scene flow estimation on that. For this to work, the true semantically corresponding point should ideally fall in the k-nearest neighbourhood, for if it does not, the network might have a difficult time learning a proper mapping. This means that the maximum motion between two frames that our network is able to capture, is parameterized by k (the parameter from k-nearest neighbour).

With more time, we would have liked to perform a more in-depth investigation of the impact of various values of k on the performance of the network when varying the maximum motion between two frames. We speculate that increasing k would allow the network to obtain better performance on sequences with larger motion between frames. Note also that frames that are temporally further apart, generally exhibit a larger motion between them. This means that if we can make the network successfully interpolate larger motions, we could drop more frames, resulting in larger bandwidth savings.

5.1.6 Distortion metrics as loss function

We started out this project with the intention to use point cloud distortion metrics as loss function for our neural network. We considered the distortion metrics of projection distortion, point-to-point, point-to-plane, and plane-to-plane (all discussed in Section 2.7 in more detail). With each of these metrics we encountered difficulties when attempting to use them as loss function for our neural network:

- **Projection distortion** To calculate the projection distortion, a number of 2D renderings of the point clouds have to be created first, which proved to be challenging. This is because the point clouds used in our neural network are downsampled to 2048 points, which makes it hard to make informative renderings (meaning the renderings convey the essential properties of the point cloud). If a low resolution is chosen for the projection, little detail will be available, and the metric would only account for large deviations between our interpolation and the ground truth. If a high resolution is chosen, the projection will contain many holes, which can lead to two point clouds being labeled as very different, even if the underlying geometry they describe is very similar. Figure 5.1 shows examples of both a low- and high resolution projection of a point cloud.

(a) Low resolution projection **(b)** High resolution projection

Figure 5.1: Example of inadequate projections Figure (a) shows a low resolution projection of a point cloud (19×30 pixels). While this projection does not have any holes, it is pixelated, and contains very little information. (b) shows a higher resolution projection of the same input point cloud (126×203 pixels). Because of the low resolution of the input point cloud, this projection is full of holes, making comparison using 2D distortion metrics difficult.

- **Plane-to-plane** Using plane-to-plane distortion requires surface normal information for both the point clouds that are being compared. Since our neural network does not estimate these surface normals, plane-to-plane metrics cannot be directly used. One solution would be to use established algorithms to estimate the surface normals, but doing so makes output of the loss function dependent on the quality of the chosen normal estimation algorithm. Additionally, performing the surface normal estimation would add significant computational overhead to each training pass, which adds up rapidly due to the large number of training passes required to train a neural network to convergence.

- **Point-to-point, point-to-plane** Unlike the projection distortion and plane-to-plane distortion, these two metrics did not have fundamental issues preventing us from using them as loss functions. When using them as loss function, however, we never managed to obtain good results. Gaining insight in the training process of a neural network is notoriously difficult, and as such we will now speculate why these metrics did not perform well as loss function. Both metrics work by finding for

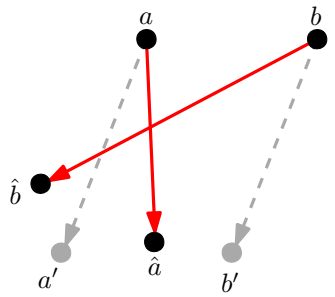


Figure 5.2: Example of failure of point-to-point and point-to-plane metrics as loss function. Consider the scene flow estimation between frame $t = \{a, b\}$ and frame $t' = \{a', b'\}$. The ground truth scene flow is shown using the grey dashed arrows. Since the weights in the network are initialized randomly, the initial scene flow prediction could be anywhere. Assume that at some point the network predicts scene flow $\hat{a} - a$ for point a, and $\hat{b} - b$ for point b (the red arrows). Now when calculating the point-to-point or point-to-plane error, b' will be matched to $\hat{a}$, and a' will be matched to $\hat{b}$, and the error will be calculated using these matchings. As a result, the network will attempt to minimize the distance between b' and $\hat{a}$, and between a' and $\hat{b}$, which is an erroneous adjustment with regards to the ground truth.

each point in the one point cloud the nearest point in the other point cloud, and then calculating some measure of error between these two matched points. Note however, that the matched point is not guaranteed (or even likely) to be the true semantically corresponding point. The metrics only consider closeness to *any* point, not closeness to the correct point. We illustrate an example of this behavior in Figure 5.2. Additionally, which points get matched together is determined by the scene flow estimation made by the network, meaning that as scene flow estimations change throughout the training process, so will the point matchings. This can cause jumps in the error measure throughout training iterations, which results in a non-smooth gradient. We suspect that the combination of these two factors makes it difficult for the network to learn to interpolate the point clouds properly.

Altogether, our experiences lead us to question whether traditional point cloud distortion metrics can at all be useful as loss functions for temporal interpolation of dynamic point clouds or other tasks that output point clouds. In order to use point cloud distortion metrics as loss function, further research into new distortion metrics is needed.

5.1.7 Objective metrics for evaluation

The objective metrics we report in Section 4.2.5 paint a different picture than the results from our user study in Section 4.2.6. As can be seen in Figure 4.12, the metrics have limited

correlation with the user scores form our user study. All this matches findings from other research, such as Torlig et al. (71). This is problematic, as our work is highly dependent on these metrics, both for their use as loss function to train our neural network, and for evaluation of our architecture. For the purpose of evaluation, this means that objective metrics alone can not serve as a replacement for user studies. Objective metrics can still be useful to quickly get an approximate idea of the performance of the architecture, but for accurate evaluation, a user study, albeit costly and time-consuming, is preferred. We hope that in the future new objective quality metrics for point clouds will be researched, that can correlate better with human perception of visual quality.

5.2 Future work

We now identify a number areas where improvements could be made, and which could be explored in future research. In the context of our architecture specifically we focus on two issues. First, it would be useful to further study the effect of the neighbourhood radius on the performance of the network when the motion between frames is large. Second, more efficient nearest neighbour implementations could speed up the runtime performance of the architecture significantly, and could help a lot towards making the system capable of real-time interpolation.

More general to the problem of temporal interpolation of dynamic point clouds, or even to the general application of machine learning to dynamic point clouds, we identify another set of problems that need to be addressed. First, the number of publicly available dynamic point cloud data sets is currently limited, which makes it difficult to train neural networks with them. It would be highly useful if more public dynamic point cloud data sets were made available. Next, there is the issue of scaling with spatial resolution. Many architectures that apply deep learning on point clouds rely on techniques like furthest point sampling or nearest neighbour search, which do not scale well computationally with the number of input points. By researching either alternatives or improvements to these techniques, it might become possible to process point clouds with more points. Lastly, we believe new research into objective quality metrics is needed. Existing quality metrics only exhibit a weak correlation with user perception, which makes it hard to accurately evaluate the quality of any interpolation architecture without performing costly user studies. Having access to better objective quality metrics could thus speed up the development process of such architectures.

5.3 Conclusion

In this thesis we set out to design an architecture capable of performing temporal interpolation on dynamic point clouds. By transmitting point clouds in a lower frame rate and successively upsampling their frame rate using such an architecture on the receiving side, the bandwidth requirements of streaming dynamic point clouds can be reduced, without making concessions on the Quality of Experience of the viewer. We identified a number of Research Objectives that needed to be resolved in order to achieve this goal. In this Section we now briefly revisit our primary Research Objectives, and discuss how we have addressed them. Finally, we conclude.

> **Research Objective 1** *Design an architecture capable of performing temporal interpolation on dynamic point clouds.*

We have designed, implemented and evaluated a new architecture capable of performing temporal interpolation of dynamic point clouds. We first downsample the input point clouds to a more manageable size using uniform downsampling. Next, our interpolation network estimates the scene flow between these downsampled point clouds. This interpolation network first estimates a correspondence between points using our novel point matching module, which is later refined using our flow refinement module. Afterwards, we upsample this scene flow estimation using 3D interpolation. The upsampled scene flow estimation then serves as the basis of the interpolation. Finally, we apply the novel neighbour snapping technique to improve the smoothness of the interpolation.

> **Research Objective 2** *Train our neural network.*

In order to be able to train our neural network, we have created a novel synthetic data set consisting of animated human bodies, complete with ground truth scene flow and surface normals. This data set has allowed us to train our neural network despite the limited availability of real-world dynamic point cloud data sets. Additionally, we have experimented with a number of objective metrics as loss functions, and have discussed their limitations in this capacity. Remaining research challenges include making the architecture capable of real-time interpolation, and exploring its application on real-world data.

We have evaluated our work extensively using a variety of objective metrics, as well as by conducting a user study. Our results reaffirm the findings from related research that existing objective quality metrics have a poor correlation with subjective user perception. Additionally, our user study shows that participants generally prefer sequences interpolated using our architecture over these interpolated by current state-of-the-art or sequences that have not been interpolated.

Appendix A

Data set creation

In this Appendix we discuss in more the detail the creation of our synthetic data set. We start by downloading a number of animated sequences from Adobe Mixamo (76) in the Filmbox (.FBX) format. In this Appendix we explain how we first convert these to meshes in Wavefront OBJ format (83) (.OBJ and .MTL), and how we later convert these meshes to point clouds in .PLY format. Lastly, we discuss how we calculate surface normal- and scene flow data.

Filmbox to Wavefront OBJ We use a custom Blender (77) script to extract from the Filmbox file the textures and to generate per frame in the animation a mesh in the Wavefront OBJ format (83) (.OBJ and .MTL). This is done with the `fbx_to_obj.py` script, which can be found on our Github page, along instructions on how to run it. All the heavy lifting here is done by Blender, so we do not go into much detail about this step.

Wavefront OBJ to point cloud Next we wish to convert these meshes to point cloud format (.PLY), for this we use the `obj_to_ply.py` script, also found on our Github 4.2.1 page. This conversion works as follows. The meshes in OBJ format consist of a list of m triangles. Each triangle is described by three corner points. Each corner point is described by a spatial coordinate (x, y, z), and a texture coordinate, which determines which part of the texture will be mapped to that point during rendering. We want to randomly sample n points, and we wish to sample from each triangle proportional to its surface area. To do this, we need to calculate for each triangle its surface area. Let the spatial coordinates for the three corner points be denoted as $P_1 =< a_1, a_2, a_3 >$, $P_2 =< b_1, b_2, b_3 >$ and $P_3 =< c_1, c_2, c_3 >$. We first create two vectors describing the triangle:

$$\vec{P_1P_2} =< b_1 - a_1, b_2 - a_2, b_3 - a_3 >=< x_1, y_1, z_1 > \tag{A.1}$$

$$\vec{P_1P_3} =< c_1 - a_1, c_2 - c_2, b_3 - a_3 >=< x_2, y_2, z_2 > \tag{A.2}$$

We can then calculate their orthogonal vector by taking their cross product:

$$\vec{u} = \vec{P_1P_2} \times \vec{P_1P_3} =< x_3, y_3, z_3 > \tag{A.3}$$

We can then calculate the magnitude of this orthogonal vector. This gives us the area of the parallelogram described by the two orthogonal vectors:

$$|\vec{u}| = \sqrt{(x_3)^2 + (y_3)^2 + (z_3)^2} \tag{A.4}$$

The area of the triangle is then simply half the area of this parallelogram:

$$area = \frac{|\vec{u}|}{2} \tag{A.5}$$

We calculate for each triangle its surface area according to the Equations above, and also calculate the sum of the surface area of each triangle. We then randomly sample n triangle indices between 0 and m proportional to their surface area. For each sampled index we will randomly sample one point in that triangle. It is possible (and likely) that certain triangles will be sampled multiple times, or that some triangles might not be sampled at all. In order to randomly select a point on a triangle, we can use the barycentric coordinate system (84). We randomly sample barycentric coordinates u and v between 0 and 1. In the event that $u + v > 1$ the point would fall outside the triangle, so if that is the case we adjust them to $u = 1 - u$ and $v = 1 - v$. The third barycentric coordinate now becomes:

$$w = 1 - (u + v) \tag{A.6}$$

To obtain the spatial coordinate of the sampled point, we can now calculate the Cartesian coordinates as follows:

$$P_{sampled} = u * P_1 + v * P_2 + w * P_3 \tag{A.7}$$

We can sample the 2D texture coordinates similarly. Given the three texture coordinates $T_1 =< a_1, a_2 >$, $T_2 < b_1, b_2 >$, and $T3 =< c_1, c_2 >$, we can sample the random texture point $T_{sampled} =< d_1, d_2 >$ as follows:

$$d_1 = u * a_1 + u * b_1 + u * c_1 \tag{A.8}$$

$$d_2 = v * a_2 + v * b_2 + v * c_2 \tag{A.9}$$

To obtain the color of the sampled point, we simply look up the pixel at coordinate $T_{sampled} =< d_1, d_2 >$ in the texture.

Surface normals We also wish to generate surface normals, for use in certain distortion metrics. To calculate these surface normals, we can reuse the orthogonal vectors $\vec{P_1P_2} = < x_1, y_1, z_1 >$ and $\vec{P_1P_3} =< x_2, y_2, z_2 >$ that we calculated earlier. The normal vector $\hat{P} =< \hat{x}, \hat{y}, \hat{z} >$ can be calculated as follows:

$$\hat{x} = y_1 * z_2 - z_1 * y_2 \tag{A.10}$$

$$\hat{y} = z_1 * x_2 - x_1 * z_2 \tag{A.11}$$

$$\hat{z} = x_1 * y_2 - y_1 * x_2 \tag{A.12}$$

Note that all points that are sampled from a given triangle will share the same surface normal. In our data set the meshes contain a relatively high number of triangles, so this is not a problem. If the triangles are larger, normal interpolation techniques like Phong shading (85, 86) could be applied to create a smoother surface approximation.

Scene flow Additionally, we also want to generate scene flow ground truth data, for use in training our neural network. In the meshes in Wavefront OBJ format, a triangle at index i in a frame at time t will semantically correspond to the triangle at index i in any frame at time t'. By reusing the triangle indices and barycentric coordinates that we randomly sampled we can thus sample points that semantically correspond to each other. For any two semantically corresponding points we can then trivially calculate their scene flow by subtracting their spatial coordinates.

Appendix B

User study material

In this Appendix we include the material used during the user study we conducted. Section B.1 show the the informed consent form, which explains the goal and procedure of the study. Section B.2 is the Participant Information form, used to ask the participant for some additional information. Section B.3 Video Rating Form, used by participants to rate the point cloud sequences. In Section B.4 we show the video viewing interface used during the study.

B.1 Informed consent form

Informed Consent

Temporal interpolation of dynamic point clouds evaluation

Research Team

Jelmer Mulder (VU / CWI), j.mulder@vu.nl

Goal

The goal of this evaluation is to assess user perception of dynamic point cloud sequences of various frame rates, and to study the impact of temporal interpolation. You will be asked to watch a number of videos and rate them.

Participation

- The expected duration of this evaluation session is 25 minutes, including an introduction of the project (5 minutes), and a video rating session (20 minutes).
- Participation is voluntary. You are free to abandon the experiment at any time, for any reason.
- The researcher will answer all further questions regarding the experiment: now, during the experiment, or afterwards.

Privacy

- All data will be analyzed anonymously. The results of this research might be scientifically published. Data will always be presented anonymously.
- Personal data will not be shared with third parties. All personal data will be removed 6 months after the end of the research.

Procedure

- You will be shown 32 short animated video sequences, spanning 10 to 15 seconds each. You are only allowed to view each sequence once, as we are interested in your first impression.
- Please rate on the *Video Rating Form* the quality of the motion on a scale from ‘1’ to ‘7’, with ‘1’ meaning the motion is of terrible quality, and ‘7’ meaning the motion is of excellent quality.
- You should only consider the quality of the motion. The quality of the model should not be factored into your rating.

Please sign if you understand this information, and agree to participate in this experiment:

Name: _____ **Date:**_____ **Signature:**_____

B.2 Participant information form

Participant Information Form

Temporal interpolation of dynamic point clouds evaluation

Name: _____

Age: _____

Do you have any prior experience in visual quality evaluation? Yes / No

Date:_____ **Signature:**_____

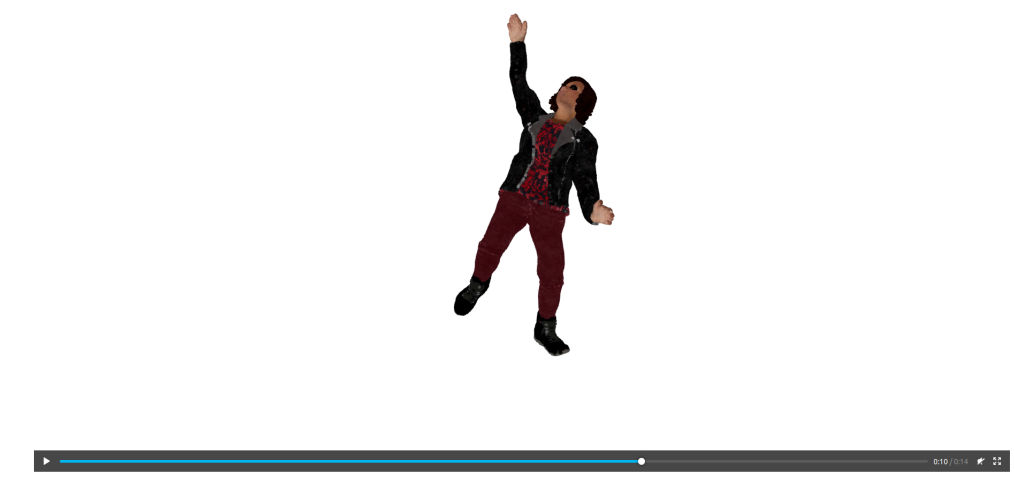
B.3 Video rating form

Video rating form

	(Terrible) 1	2	3	4	5	6	(Excellent) 7
Sequence 01	○	○	○	○	○	○	○
Sequence 02	○	○	○	○	○	○	○
Sequence 03	○	○	○	○	○	○	○
Sequence 04	○	○	○	○	○	○	○
Sequence 05	○	○	○	○	○	○	○
Sequence 06	○	○	○	○	○	○	○
Sequence 07	○	○	○	○	○	○	○
Sequence 08	○	○	○	○	○	○	○
Sequence 09	○	○	○	○	○	○	○
Sequence 10	○	○	○	○	○	○	○
Sequence 11	○	○	○	○	○	○	○
Sequence 12	○	○	○	○	○	○	○
Sequence 13	○	○	○	○	○	○	○
Sequence 14	○	○	○	○	○	○	○
Sequence 15	○	○	○	○	○	○	○
Sequence 16	○	○	○	○	○	○	○
Sequence 17	○	○	○	○	○	○	○
Sequence 18	○	○	○	○	○	○	○
Sequence 19	○	○	○	○	○	○	○
Sequence 20	○	○	○	○	○	○	○
Sequence 21	○	○	○	○	○	○	○
Sequence 22	○	○	○	○	○	○	○
Sequence 23	○	○	○	○	○	○	○
Sequence 24	○	○	○	○	○	○	○
Sequence 25	○	○	○	○	○	○	○
Sequence 26	○	○	○	○	○	○	○
Sequence 27	○	○	○	○	○	○	○
Sequence 28	○	○	○	○	○	○	○
Sequence 29	○	○	○	○	○	○	○
Sequence 30	○	○	○	○	○	○	○
Sequence 31	○	○	○	○	○	○	○
Sequence 32	○	○	○	○	○	○	○

B.4 Video viewing interface

(a) During demonstration



(b) During rating

Figure B.1: User study interface Figure (a) shows the interface used during the demonstration, it is used to show the participant four sequences in order to get an idea of the range in quality of the sequences, and to become familiar with the interface. (b) shows the interface used during the video rating session itself. Participants are shown a number of 10s-15s clips. When the clip ends, the next clip is loaded (but paused), so each clip can only be viewed once. The participants can then start the next clip at their own volition.

References

[1] Aljoscha Smolic. **3D video and free viewpoint videoFrom capture to display**. *Pattern recognition*, **44**(9):1958–1968, 2011. 2

[2] Greg Turk and Marc Levoy. **The stanford bunny**, 2005. 2

[3] Kyriaki Christaki, Konstantinos C Apostolakis, Alexandros Doumanoglou, Nikolaos Zioulis, Dimitrios Zarpalas, and Petros Daras. **Space Wars: An AugmentedVR Game**. In *International Conference on Multimedia Modeling*, pages 566–570. Springer, 2019. 2

[4] Sebastian Schwarz, Marius Preda, Vittorio Baroncini, Madhukar Budagavi, Pablo Cesar, Philip A Chou, Robert A Cohen, Maja Krivokuća, Sébastien Lasserre, Zhu Li, et al. **Emerging MPEG standards for point cloud compression**. *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, **9**(1):133–148, 2018. 2

[5] Zhengyou Zhang. **Microsoft kinect sensor and its effect**. *IEEE multimedia*, **19**(2):4–10, 2012. 2, 3

[6] Yebin Liu, Qionghai Dai, and Wenli Xu. **A point-cloud-based multiview stereo algorithm for free-viewpoint video**. *IEEE transactions on visualization and computer graphics*, **16**(3):407–418, 2010. 2

[7] J Bedford. **Photogrammetric Applications for Cultural Heritage**. *Historic: Swindon, UK*, page 128, 2017. 2

[8] Charles R Qi, Wei Liu, Chenxia Wu, Hao Su, and Leonidas J Guibas. **Frustum pointnets for 3d object detection from rgb-d data**. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 918–927, 2018. 2

[9] **Making virtual Reality Human, Create Volumetric video of real people**. http://8i.com. Online, accessed April 2019. 3

[10] Sergio Orts-Escolano, Christoph Rhemann, Sean Fanello, Wayne Chang, Adarsh Kowdle, Yury Degtyarev, David Kim, Philip L Davidson, Sameh Khamis, Mingsong Dou, et al. **Holoportation: Virtual 3d teleportation in real-time**. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, pages 741–754. ACM, 2016. 3

[11] Kazuo Sugimoto, Robert A Cohen, Dong Tian, and Anthony Vetro. **Trends in efficient representation of 3D point clouds**. In *2017 Asia-Pacific Signal and Information Processing Association Annual Summit and Conference (APSIPA ASC)*, pages 364–369. IEEE, 2017. 3

[12] **Velodyne Lidar HDL-64e**. https://velodynelidar.com/hdl-64e.html. Online, accessed April 2019. 3

[13] **Microsoft Kinect v2**. https://www.microsoft.com/en-us/p/k/91hq5578vksc. Online, accessed April 2019. 3

[14] **Intel RealSense D435**. https://newsroom.intel.com/news/new-intel-realsense-d435i-stereo-depth-camera-adds-6-degrees-freedom-tracking/. Online, accessed April 2019. 3

[15] Leonid Keselman, John Iselin Woodfill, Anders Grunnet-Jepsen, and Achintya Bhowmik. **Intel realsense stereoscopic depth cameras**. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 1–10, 2017. 3

[16] Taos Myers Eugene dEon, Bob Harrison and Philip A. Chou. **8i Voxelized Full Bodies, version 2 - A Voxelized Point Cloud Dataset**, 2017. ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document m40059/M74006. 3, 30

[17] **Multi-Camera Configuration for Intel RealSense D400 Series Depth Sensors**. https://www.intel.com/content/www/us/en/support/articles/000028140/emerging-technologies/intel-realsense-technology.html?wapkw=multi-camera+configuration+for+intel. Online, accessed May 2019. 4

[18] **Speedtest Global Index**. https://www.speedtest.net/global-index. Online, accessed May 2019. 5

[19] Amin Banitalebi-Dehkordi, Mahsa T Pourazad, and Panos Nasiopoulos. **The effect of frame rate on 3D video quality and bitrate**. *3D Research*, **6**(1):1, 2015. 5

[20] Yao Wang, Jörn Ostermann, and Ya-Qin Zhang. *Video processing and communications*. Signal Processing Series. Prentice Hall, 2002. 6

[21] Huaizu Jiang, Deqing Sun, Varun Jampani, Ming-Hsuan Yang, Erik Learned-Miller, and Jan Kautz. **Super slomo: High quality estimation of multiple intermediate frames for video interpolation**. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 9000–9008, 2018. 6, 7, 22

[22] Xingyu Liu, Charles R Qi, and Leonidas J Guibas. **FlowNet3D: Learning Scene Flow in 3D Point Clouds**. *arXiv preprint arXiv:1806.01411*, 2018. 6, 27, 28, 55, 57, 67

[23] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. **Deep learning**. *nature*, **521**(7553):436, 2015. 7, 17, 21

[24] Simon Niklaus and Feng Liu. **Context-aware synthesis for video frame interpolation**. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1710, 2018. 7, 22

[25] Simon Niklaus, Long Mai, and Feng Liu. **Video frame interpolation via adaptive separable convolution**. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 261–270, 2017. 7, 21, 22

[26] Simon Niklaus, Long Mai, and Feng Liu. **Video frame interpolation via adaptive convolution**. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 670–679, 2017. 7, 21

[27] Charles R Qi, Hao Su, Kaichun Mo, and Leonidas J Guibas. **Pointnet: Deep learning on point sets for 3d classification and segmentation**. *Proc. Computer Vision and Pattern Recognition (CVPR), IEEE*, **1**(2):4, 2017. 7, 25, 67

[28] Charles Ruizhongtai Qi, Li Yi, Hao Su, and Leonidas J Guibas. **Pointnet++: Deep hierarchical feature learning on point sets in a metric space**. In *Advances in Neural Information Processing Systems*, pages 5099–5108, 2017. 7, 25, 28, 41

[29] Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. **Dynamic graph CNN for learning on point clouds**. *arXiv preprint arXiv:1801.07829*, 2018. 7, 14, 26, 40, 41, 67

[30] Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. **Multi-view convolutional neural networks for 3d shape recognition**. In *Proceedings of the IEEE international conference on computer vision*, pages 945–953, 2015. 7, 23

[31] Daniel Maturana and Sebastian Scherer. **Voxnet: A 3d convolutional neural network for real-time object recognition**. In *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 922–928. IEEE, 2015. 7, 24

[32] Zhirong Wu, Shuran Song, Aditya Khosla, Fisher Yu, Linguang Zhang, Xiaoou Tang, and Jianxiong Xiao. **3d shapenets: A deep representation for volumetric shapes**. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1912–1920, 2015. 9, 45

[33] Sundar Vedula, Simon Baker, Peter Rander, Robert Collins, and Takeo Kanade. **Three-dimensional scene flow**. In *Proceedings of the Seventh IEEE International Conference on Computer Vision*, **2**, pages 722–729. IEEE, 1999. 14, 27

[34] Klaas Klasing, Daniel Althoff, Dirk Wollherr, and Martin Buss. **Comparison of surface normal estimation methods for range sensing applications**. In *2009 IEEE International Conference on Robotics and Automation*, pages 3206–3211. IEEE, 2009. 15, 29

[35] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. **Imagenet classification with deep convolutional neural networks**. In *Advances in neural information processing systems*, pages 1097–1105, 2012. 17

[36] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew

Rabinovich. **Going deeper with convolutions**. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015. 17

[37] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. **Inception-v4, inception-resnet and the impact of residual connections on learning.** In *AAAI*, **4**. 17

[38] Karen Simonyan and Andrew Zisserman. **Very deep convolutional networks for large-scale image recognition**. *arXiv preprint arXiv:1409.1556*, 2014. 17

[39] Sara Sabour, Nicholas Frosst, and Geoffrey E Hinton. **Dynamic routing between capsules**. In *Advances in neural information processing systems*, pages 3856–3866, 2017. 17

[40] Li Wan, Matthew Zeiler, Sixin Zhang, Yann Le Cun, and Rob Fergus. **Regularization of neural networks using dropconnect**. In *International conference on machine learning*, pages 1058–1066, 2013. 17

[41] Dan Cireşan, Ueli Meier, and Jürgen Schmidhuber. **Multi-column deep neural networks for image classification**. *arXiv preprint arXiv:1202.2745*, 2012. 17

[42] Geoffrey Hinton, Li Deng, Dong Yu, George Dahl, Abdel-rahman Mohamed, Navdeep Jaitly, Andrew Senior, Vincent Vanhoucke, Patrick Nguyen, Brian Kingsbury, et al. **Deep neural networks for acoustic modeling in speech recognition**. *IEEE Signal processing magazine*, **29**, 2012. 17

[43] George E Dahl, Dong Yu, Li Deng, and Alex Acero. **Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition**. *IEEE Transactions on audio, speech, and language processing*, **20**(1):30–42, 2012. 17

[44] Jeffrey Dean, Greg Corrado, Rajat Monga, Kai Chen, Matthieu Devin, Mark Mao, Andrew Senior, Paul Tucker, Ke Yang, Quoc V Le, et al. **Large scale distributed deep networks**. In *Advances in neural information processing systems*, pages 1223–1231, 2012. 17

[45] Li Deng, Geoffrey Hinton, and Brian Kingsbury. **New types of deep neural network learning for speech recognition and related applications:**

An overview. In *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, pages 8599–8603. IEEE, 2013. 17

[46] Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. **Playing atari with deep reinforcement learning**. *arXiv preprint arXiv:1312.5602*, 2013. 17

[47] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. **Mastering the game of Go with deep neural networks and tree search**. *nature*, **529**(7587):484, 2016. 17

[48] Li Deng, Dong Yu, et al. **Deep learning: methods and applications**. *Foundations and Trends® in Signal Processing*, **7**(3–4):197–387, 2014. 17

[49] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep learning*. MIT press, 2016. 18, 19

[50] Boris T Polyak. **Some methods of speeding up the convergence of iteration methods**. *USSR Computational Mathematics and Mathematical Physics*, **4**(5):1–17, 1964. 20

[51] John Duchi, Elad Hazan, and Yoram Singer. **Adaptive subgradient methods for online learning and stochastic optimization**. *Journal of Machine Learning Research*, **12**(Jul):2121–2159, 2011. 20

[52] Tijmen Tieleman and Geoffrey Hinton. **Lecture 6.5-rmsprop, coursera: Neural networks for machine learning**. *University of Toronto, Technical Report*, 2012. 20

[53] Diederik P Kingma and Jimmy Ba. **Adam: A method for stochastic optimization**. *arXiv preprint arXiv:1412.6980*, 2014. 20

[54] Waseem Rawat and Zenghui Wang. **Deep convolutional neural networks for image classification: A comprehensive review**. *Neural computation*, **29**(9):2352–2449, 2017. 20

[55] Xavier Glorot, Antoine Bordes, and Yoshua Bengio. **Deep sparse rectifier neural networks**. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011. 21

[56] Hui Men, Hanhe Lin, Vlad Hosu, Daniel Maurer, Andrés Bruhn, and Dietmar Saupe. **Technical Report on Visual Quality Assessment for Frame Interpolation**. *arXiv preprint arXiv:1901.05362*, 2019. 21

[57] Daniel J Butler, Jonas Wulff, Garrett B Stanley, and Michael J Black. **A naturalistic open source movie for optical flow evaluation**. In *European Conference on Computer Vision*, pages 611–625. Springer, 2012. 21

[58] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. **Deep residual learning for image recognition**. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016. 22

[59] Deqing Sun, Xiaodong Yang, Ming-Yu Liu, and Jan Kautz. **Pwc-net: Cnns for optical flow using pyramid, warping, and cost volume**. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 8934–8943, 2018. 22

[60] Truc Le and Ye Duan. **Pointgrid: A deep network for 3d shape understanding**. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 9204–9214, 2018. 24

[61] Roman Klokov and Victor Lempitsky. **Escape from cells: Deep kd-networks for the recognition of 3d point cloud models**. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 863–872, 2017. 24

[62] Michael M Bronstein, Joan Bruna, Yann LeCun, Arthur Szlam, and Pierre Vandergheynst. **Geometric deep learning: going beyond euclidean data**. *IEEE Signal Processing Magazine*, **34**(4):18–42, 2017. 25

[63] Yangyan Li, Rui Bu, Mingchao Sun, Wei Wu, Xinhan Di, and Baoquan Chen. **PointCNN: Convolution On X-Transformed Points**. In *Advances in Neural Information Processing Systems*, pages 828–838, 2018. 26

[64] Lequan Yu, Xianzhi Li, Chi-Wing Fu, Daniel Cohen-Or, and Pheng-Ann Heng. **Pu-net: Point cloud upsampling network**. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2790–2799, 2018. 26

[65] Matthew J Moynihan, Rafael Pagés, and Aljoa Smolić. **Spatio-temporal Upsampling for Free Viewpoint Video Point Clouds**. In *VISIGRAPP*, 2019. 27

[66] Hui Huang, Shihao Wu, Minglun Gong, Daniel Cohen-Or, Uri Ascher, and Hao Richard Zhang. **Edge-aware point set resampling**. *ACM transactions on graphics (TOG)*, **32**(1):9, 2013. 27

[67] Ayush Dewan, Tim Caselitz, Gian Diego Tipaldi, and Wolfram Burgard. **Rigid scene flow for 3d lidar scans**. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1765–1770. IEEE, 2016. 27

[68] Jorge J Moré. **The Levenberg-Marquardt algorithm: implementation and theory**. In *Numerical analysis*, pages 105–116. Springer, 1978. 27

[69] Philip A Chou, Eduardo Pavez, Ricardo L de Queiroz, and Antonio Ortega. **Dynamic polygon clouds: Representation and compression for vr/ar**. *arXiv preprint arXiv:1610.00402*, 2016. 28, 29

[70] Hamid R Sheikh and Alan C Bovik. **Image information and visual quality**. In *2004 IEEE International Conference on Acoustics, Speech, and Signal Processing*, **3**, pages iii–709. IEEE, 2004. 28

[71] Eric M Torlig, Evangelos Alexiou, Tiago A Fonseca, Ricardo L de Queiroz, and Touradj Ebrahimi. **A novel methodology for quality assessment of voxelized point clouds**. In *Applications of Digital Image Processing XLI*, **10752**, page 107520I. International Society for Optics and Photonics, 2018. 28, 29, 63, 64, 73

[72] Dong Tian, Hideaki Ochimizu, Chen Feng, Robert Cohen, and Anthony Vetro. **Geometric distortion metrics for point cloud compression**. In *2017 IEEE International Conference on Image Processing (ICIP)*, pages 3460–3464. IEEE, 2017. 29

[73] Evangelos Alexiou and Touradj Ebrahimi. **Point cloud quality assessment metric based on angular similarity**. In *2018 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1–6. IEEE, 2018. 29

[74] Touradj Ebrahimi, Siegfried Foessel, Fernando Pereira, and Peter Schelkens. **Jpeg pleno: Toward an efficient representation of visual reality**. *Ieee Multimedia*, **23**(4):14–20, 2016. 30

[75] Sergio Orts Escolano Charles Loop, Qin Cai and Philip A. Chou. **Microsoft Voxelized Upper Bodies - A Voxelized Point Cloud Dataset**, 2016. ISO/IEC JTC1/SC29 Joint WG11/WG1 (MPEG/JPEG) input document m38673/M72012. 31

[76] **Adobe Mixamo**. https://www.mixamo.com, 2019. Online, accessed March 2019. 45, 46, 75

[77] Blender Online Community. *Blender - a 3D modelling and rendering package*. Blender Foundation, Blender Institute, Amsterdam, 2019. 46, 75

[78] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**, 2015. Software available from tensorflow.org. 48

[79] Sinno Jialin Pan and Qiang Yang. **A survey on transfer learning**. *IEEE Transactions on knowledge and data engineering*, **22**. 67

[80] Moritz Menze and Andreas Geiger. **Object scene flow for autonomous vehicles**. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3061–3070, 2015. 67

[81] Nikolaus Mayer, Eddy Ilg, Philip Hausser, Philipp Fischer, Daniel Cremers, Alexey Dosovitskiy, and Thomas Brox. **A large dataset to train convolutional networks for disparity, optical flow, and scene flow estimation**. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4040–4048, 2016. 67

[82] William Mattson and Betsy M Rice. **Near-neighbor calculations using a modified cell-linked list method**. *Computer Physics Communications*, **119**(2-3):135–148, 1999. 68

[83] **Wavefront OBJ format**. http://paulbourke.net/dataformats/obj, 2019. Online, accessed March 2019. 75

[84] August Ferdinand Möbius. *Der barycentrische Calcul, ein Hülfsmittel zur analytischen Behandlung der Geometrie (etc.)*. Barth, 1827. 76

[85] Bui Tuong-Phong. **Illumination for computer-generated images**. *Technical Report 129, UTEC-CSC-73, Computer Science*, 1973. 77

[86] Alphonsus Aloisius Maria Kuijk and Edwin H Blake. **Faster phong shading via angular interpolation**. In *Computer Graphics Forum*, **8**, pages 315–324. Wiley Online Library, 1989. 77