Efficient Processing and Learning on Unstructured Data

Dissertation

der Mathematisch-Naturwissenschaftlichen Fakultät

der Eberhard Karls Universität Tübingen

zur Erlangung des Grades eines

Doktors der Naturwissenschaften

(Dr. rer. nat.)

vorgelegt von

Fabian Groh

Heilbronn

Tübingen

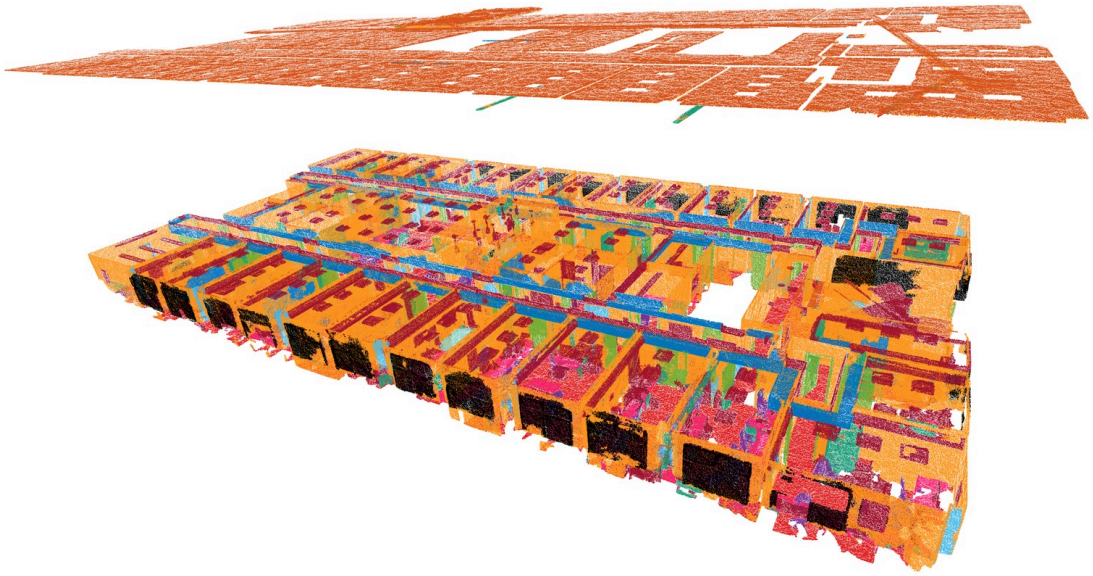
2024

Gedruckt mit Genehmigung der Mathematisch-Naturwissenschaftlichen Fakultät der Eberhard Karls Universität Tübingen.

Tag der mündlichen Qualifikation:	07.02.2025
Dekan:	Prof. Dr. Thilo Stehle
1. Berichterstatter/-in:	Prof. Dr.-Ing. Hendrik P.A. Lensch
2. Berichterstatter/-in:	Prof. Dr. rer. nat. Andreas Schilling

EFFICIENT PROCESSING AND LEARNING ON UNSTRUCTURED DATA

FABIAN GROH



Advanced Techniques for Reconstruction, Deep-Learning, and Searching on Large-Scale, Real-World Point Clouds and Feature Spaces

2024

Efficient Processing and Learning on Unstructured Data

SUPERVISORS:
Prof. Dr.-Ing. Hendrik P.A. Lensch
Prof. Dr. rer. nat. Andreas Schilling

ABSTRACT

Inferring knowledge by visually perceiving the world is the fundamental goal of computer vision. The primary input types are images and videos, particularly in the era of deep learning. Such structured data representations are defined on a fixed grid, implicitly assigning positional information to each data point. Hence, finding neighbors is a simple lookup; data must be defined everywhere by the same resolution.

On the other hand, unstructured data representations like point clouds, feature spaces, or graphs do not have such an underlying structure. However, they can efficiently embed information from multiple sources into a common representation, leveraging knowledge and resolutions otherwise impossible to achieve from a single source. In particular, higher dimensional data is infeasible to represent in a densely structured way. Unfortunately, losing the implicit grid structure renders many algorithms fundamental to computer vision unusable. In particular, the advancements in vision-based deep learning heavily depend on the highly efficient frameworks providing implementations and routines only defined on structured data.

This thesis highlights the importance of unstructured data representations and presents efficient processing and learning techniques in this context. Four projects are forming the core of this work, demonstrating different aspects:

1. A high-precision 3D reconstruction scheme with active illumination [1].

2. A framework to enable fully-convolutional deep learning on point cloud data [2].

3. An end-to-end Fisher Vector embedding for object recognition [3].

4. A superfast proximity graph construction and query for nearest neighbor search on GPUs [4].

It is shown that *Efficient Processing and Learning on Unstructured Data* is possible when algorithms are designed from the ground up in a massively parallelizable fashion and domain knowledge is incorporated. Generally, high efficiency is achieved by scaling up the simple but robust and effective methods through GPUs. The presented projects are evaluated against large-scale, real-world data to showcase their robustness, performance, and application capabilities.

ZUSAMMENFASSUNG

Das grundlegende Ziel von *Computer Vision* ist es aus visueller Wahrnehmung Erkenntnisse zu erlangen. Die wichtigsten Eingabedaten dafür sind Bilder und Videos, insbesondere beim *Deep Learning*. Solche strukturierten Daten sind auf einem fixen Raster definiert die jedem Datenpunkt implizit eine lokale Position zuweisen. Dadurch ist das Suchen von Nachbarpunkten ein einfaches Nachschauen. Aber die Daten müssen überall mit der gleichen Auflösung definiert sein.

Auf der anderen Seite haben unstrukturierte Daten wie Punktwolken, Feature Räume oder Graphen keine derartige Struktur. Diese können jedoch Informationen von mehreren Eingabedaten effizient in eine einzelne gemeinsame Repräsentation einbetten, um sich so Einsichten und Auflösungen zunutze zu machen, die aus einem einzelnen Bild nicht möglich gewesen wären. Insbesondere ist es nicht realisierbar hoch-dimensionale Daten in einer dichten strukturierten Repräsentation darzustellen.

Als Konsequenz geht mit dem Verlust des impliziten Rasters von strukturierten Daten auch der Verlust vieler Algorithmen, die fundamental in der *Computer Vision* sind, einher. Vor allem die Fortschritte von *Deep Learning* im *Vision* Bereich sind überaus abhängig von hocheffizienten Frameworks, welche Implementierungen und Routinen anbieten, die nur auf strukturierten Daten definiert sind.

Diese Dissertation hebt die Bedeutung von unstrukturierten Datenrepräsentationen hervor und präsentiert effiziente Methoden um diese zu verarbeiten und darauf zu lernen. Vier Arbeiten, die verschiedene Aspekte beleuchten, bilden den Kern der Dissertation:

1. Ein Verfahren für hoch-präzise 3D Rekonstruktionen mit aktiver Beleuchtung [1].

2. Eine CNN basierte *Deep Learning*-Architektur für Punktwolken [2].

3. Eine Methode zur Objekterkennung mit Ende-zu-Ende gelernten *Fisher-Vektoren* [3].

4. Ein Schema zum schnellen Suchen und Konstruieren von Nachbarschaftsgraphen auf GPUs [4].

Es wird gezeigt, dass das *effiziente Prozessieren und Lernen auf unstrukturierten Daten* möglich ist, wenn die Algorithmen vom Grund auf in einem massiven parallelen Stiel designt werden und Wissen über die zu grundlegende Domäne integriert wird. Im Allgemeinen wird die hohe Effizienz erreicht, indem die simplen aber robusten und effektiven Methoden auf GPUs hochskaliert werden. Die vorgestellten

Projekte werden anhand von umfangreichen realen Daten evaluiert um ihre Robustheit, Performanz und Anwendungsmöglichkeiten aufzuzeigen.

PUBLICATIONS

Publications and pre-prints that are featured in this thesis.

** shared first authorship.*

[1] Fabian Groh, Benjamin Resch, and Hendrik P.A. Lensch. "Multi-view continuous structured light scanning." In: *German Conference on Pattern Recognition (GCPR)*. Springer. 2017, pp. 377–388.

[2] Fabian Groh, Patrick Wieschollek, and Hendrik P.A. Lensch. "Flex-Convolution: Million-scale point-cloud learning beyond grid-worlds." In: *Asian Conference on Computer Vision (ACCV)*. Springer. 2018, pp. 105–122.

[3] Patrick Wieschollek*, Fabian Groh*, and Hendrik P.A. Lensch. "Backpropagation training for fisher vectors within neural networks." In: *arXiv preprint arXiv:1702.02549* (2017).

[4] Fabian Groh, Lukas Ruppert, Patrick Wieschollek, and Hendrik P.A. Lensch. "GGNN: Graph-based GPU nearest neighbor search." In: *IEEE Transactions on Big Data* 9.1 (2022), pp. 267–279.

Work which is not used in this thesis:

[5] Mark Boss, Fabian Groh, Sebastian Herholz, and Hendrik PA Lensch. "Deep Dual Loss BRDF Parameter Estimation." In: *MAM@ EGSR*. 2018, pp. 41–44.

[6] Dennis R Bukenberger, Katharina Schwarz, Fabian Groh, and Hendrik PA Lensch. "Rotoscoping on Stereoscopic Images and Videos." In: *VMV*. 2015, pp. 111–118.

[7] Fabian Groh. "Gamification: State of the art definition and utilization." In: *Institute of Media Informatics Ulm University* 39 (2012), p. 31.

[8] Sven Reichel, Timo Muller, Oliver Stamm, Fabian Groh, Bjorn Wiedersheim, and Michael Weber. "Mampf: An intelligent cooking agent for zoneless stoves." In: *2011 Seventh International Conference on Intelligent Environments*. IEEE. 2011, pp. 171–178.

Simplicity is the ultimate sophistication.

— Leonardo da Vinci

ACKNOWLEDGMENTS

I would like to extend my sincere thanks to everyone who played a part in making this dissertation a reality. First and foremost, I must thank my wife Natalie for her continuous support and the gentle nudges that motivated me to push this work over the finish line. My parents also deserve a hearty thank you. They have always been supportive and have created a warm, comfortable environment in which I could grow into the person I am today. I am equally grateful to my Professor, not only for the scientific insights but also for the broader wisdom that emerged during our memorable Friday afternoon chats. A special shout-out goes to Patrick for his help throughout my journey and for those unforgettable insights, such as "As long as the shape fits."

CONTENTS

Part I

PERCEIVING AN UNSTRUCTURED WORLD

1 INTRODUCTION

Perception provides us – as humans – with an understanding and representation of our environment. The literature defines perception as organizing, identifying, and interpreting obtained sensory information [9, 10].

There are centuries-old philosophical debates about whether human perception should be considered *direct* or *indirect* conscious experiences of the physical world [11, 12]. Most modern scientists lean towards the *indirect* concept, *i.e.*, an internal representation of the external world is recreated by the brain based on the signal from sensory organs, plus any other available information it has about the world; in contrast, the *direct* idea considers the representation as an instantiation almost solely based on the perceived signal itself [9, 11].

Nevertheless, the perceptual system is undoubtedly beyond simple signal processing and requires some intelligence to transform sensory input into higher-level interpretations [13, 14]. Human perception often goes unnoticed, undervaluing the task's meaning and complexity. For example, when reading these lines, merely looking at black strokes on a white background.

MACHINE PERCEPTION Perceiving the outside physical world meaningfully is the key feature that allows interaction with our surroundings. Be it navigating safely on a sidewalk, planning a path through the living room, preparing a cup of coffee, or communicating with others. Hence, perception is essential for humans and any (intelligent) agent that wants to obtain and interact with the environment, including humans or even other machine agents [15].

An agent can be defined as anything that perceives its environment through sensors and processes the data to act through knowledge-based conditions [15]. In the soring era of autonomous vehicles, complex self-driving cars are coming to mind; however, software programs acting on user inputs are also classifiable as agents by this definition.

Sensing and interpreting by any computer system can be denoted as machine perception [16, 17]. This term often describes the robotics field of mimicking humans' exteroceptive perceptual senses like vision or hearing [18].

In recent years, many groundbreaking achievements have been made in natural language understanding, achieving a high degree of machine hearing, like Amazon's Alexa, Apple's Siri, or Google Assistant.

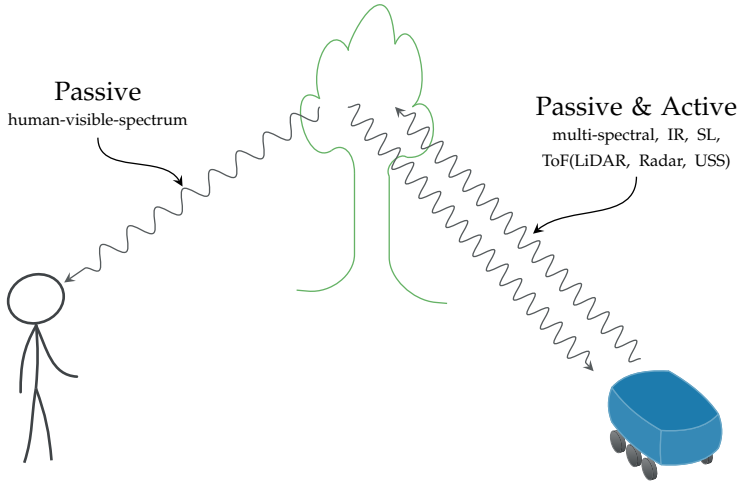


Figure 1.1: **Human and Machine Visual Perception Sensors:** Eyes passively acquire photons that are reflected or emitted from the scene. While cameras are similar, they are not restricted to the visual spectrum of humans, *e.g.*, IR cameras. Further, for 3D sensing, an active signal can be emitted to the scene, like in LiDAR, Radar, or Structured Light.

VISUAL PERCEPTION According to research, vision is the primary sense for humans [19]. Medicine-related work estimates that the human cortex uses up to 50 percent only for visual processing [20] supported by the eye sending data at about 10 million bits per second to the brain [21]. Like human vision, machine vision is likely to carry the most information regarding intelligent agents interacting with the external world, especially in areas of applications like autonomous vehicles, life assistant systems, or medical imaging.

The field of the most common representative, *computer vision (CV)*, established itself parallel to the philosophical and psychological efforts about perception [22]. The overarching goal of *computer vision*, like perception, is to generate a high-level and meaningful scene understanding from **digital images** and **videos**.

Further, *computational photography and imaging* utilize cameras and other sensors in a more general setting to obtain information that would not be achievable with traditional photometric cameras [23, 24]. Therefore, these methods incorporate knowledge and techniques from *computer graphics* and *optics*. In this thesis, *computer vision* will serve as an umbrella term for all these techniques combining *visual perception*. In particular, this thesis does not limit *computer vision* to images and videos but includes any form of sensor.

KNOWLEDGE OF PHYSICS In contrast to the philosophical discussions, computer vision is a considerably more physics-based field. The

broader field covers everything from acquiring methods to processing techniques and analyzing models [22, 25, 26].

Many techniques and hardware sensors provide capabilities way beyond the sensory organs of humans. See Figure 1.1. Visual machine perception can obtain signals in spectra way outside of the human-perceivable range, like *infrared (IR)* radiation for thermal imaging. However, a principal difference is using active sensing, where a signal is actively emitted to the scene to provoke a response. From actively projecting *Structured Light (SL)* on the surface for added information; through to *Time-of-Flight (ToF)* techniques that measure distances either via electromagnetic radiation like radio waves (*Radar*) and lasers (*LiDAR*), or with acoustics sound waves like ultrasonic (*USS*). Further, medical imaging utilizes X-rays (*CT*) and magnetic fields (*MRT*) to see through the skin. Besides that, computer systems can tap into a wealth of exterior knowledge. For example, when aggregating information from images gathered via the internet to create a 3D reconstruction of a building from another part of the world [27]. In general, it is possible to explicitly incorporate additional knowledge like physical processes, sensor characteristics, or the current state of the world.

To fully embrace the capabilities of computer vision, visual perception systems have to go beyond passively acquired images in the human visible spectrum.

1.1 MACHINE LEARNING TAKES OVER

In the last decade, computer vision tasks have improved tremendously with the rise of *machine learning*, particularly *deep learning* [28]. While the dream of a human-comparable general visual scene understanding is still not within short reach, recent developments are achieving even superhuman results in all kinds of years-long computer vision challenges, mostly for domain-specific tasks [22, 29].

Typical computer vision tasks are image classification, object detection, semantic segmentation, depth estimation, 3D reconstruction, motion estimation, action recognition, deblurring, super-resolution, and plausible image generation[1].

The development in this field is not stopping. The number of submissions to *CVPR*, one of the biggest conferences for computer vision, tripled in the last half-decade only, very much an exponential growth [30].

The roaring of start-ups and big tech companies working on next-generation ideas involving visual perception in different domains is not to be overheard. Most prominently and ambitious are autonomous vehicles, in particular, self-driving cars, with big players like Tesla, Waymo, Zoox, Cruise, Rivian, Lyft, etc. Also, other types of autonomous vehicles are getting a good amount of attention, like

1 https://paperswithcode.com/area/computer-vision

cargo drones (Prime Air, Elroy Air), delivery robots (Scout[2], Starship), aerial taxis (SkyDrive, Lilim, Volocopter), agricultural machinery (Polybot, PATS, Trabotyx, Seasony), or naval robotics (Sea Machine). Some entrepreneurs already imagine these vehicles operating together in human-unfriendly conditions like on Mars[3].

An upcoming big field for computer vision and deep learning applications is assisting and helping individuals. Either with robotics interacting in the real world like healthcare assistance, smart homes, or via intelligent agents that, for example, help analyze medical images.

Technical vision-based advancements to society will likely support or replace low-level repetitive and tedious tasks. Examples are sorting recyclable garbage, surveillance, or advanced driver assistant systems (*ADAS*). The ubiquity of computer vision - incorporated in our surroundings - might be one of humankind's next bigger leaps. Arguably, it is already in full swing.

In history, machine learning and computer vision techniques have been highly interwoven [22]. Computer vision as a separate discipline evolved from the interdisciplinary research on artificial intelligence, image processing, and pattern recognition (which later became machine learning) [22].

These data-driven methods like *nearest neighbor (NN)* search, *logistic regression*, or *support vector machines (SVM)* played an indispensable role in computer vision over at least the last 50 years and still do. In classical pipelines, those techniques are often used on top of hand-crafted features (*e.g.*, *SIFT* [31]) [32]. However, with the change to deep learning, classical hand-crafted feature engineering is challenged even for lower-level sub-tasks [33].

Practically, *Deep Neural Networks (DNNs)* only require enough data, a powerful enough architecture (model), and excessive computational power [34, 35]. Significant parts of all modern achievements in computer vision were made on the back of deep *Convolution Neural Networks (CNN)* that uses learnable convolution layers in a stacked and hierarchical fashion. Most notably, it started with *AlexNet* (2012) [36] and was later improved by *VGG*(2014) [37], *GoogLeNet* (2014) [38], *ResNet* (2015) [39], *EfficientNet* [40], and many, many more.

Recently, Transformers, a very successful architecture in the field of *Natural Language Processing (NLP)*, was adopted to vision tasks as well [41]. As Transformers are designed to process sequential input, the main idea is tokenizing an image by splitting it into patches with positional embeddings and treating them similarly to a sentence. In theory, the self-attention mechanism should overcome the inductive bias of convolutions (translation and receptive field) that might be a limiting factor of CNNs. Academia and industry quickly developed various *Vision Transformer (ViT)*-based models with convincing results [42, 43].

2 R.I.P.

3 https://www.spacex.com/human-spaceflight/mars/

However, current work from Liu et al., which adjusts the training and architecture of CNNs similarly to ViTs, gets comparable or even better results while still using convolutions [44]. Similar observations are obtained by the *InternImage* project [45]. Hence, the comparison is still inconclusive and might show that the general architecture is only a single brick next to data augmentation, wideness, deepness layer wiring, *etc*. Further, there are many practical differences regarding training procedure, performance, cost, and inference speed. Many applied deep learning applications still use some form of convolution. Examples are real-time object detection with *YOLO* [46] or semantic segmentation on edge devices with *MobileNet* [47]. Transformers and convolutions are also combined as a pretrained backbone [48] or even as hybrids [49].

In order to achieve a higher level of scene understanding, the future of perceptions will involve machine learning methods, especially deep learning techniques.

1.2 EFFICIENCY IS KEY

The qualitative results of deep neural networks are undeniable. However, a significant downside is their enormous computational cost. In particular, if it is compared with traditional computer vision techniques or even classical machine learning approaches, like *Support Vector Machines (SVM)*, or *k-Nearest Neighbors (KNN)* algorithms [50].

Floating point operations (FLOPs) is a standard measure to indicate the computational power required for deep neural network models. Current approaches are stated in the billions of *FLOPs* to perform *inference*, *i.e.*, predicting one output based on input [49].

The flipside to *inference* is *training*, which describes updating the tuneable parameters of a model based on training samples.

Deep learning relies heavily on annotated big data for an appropriate generalized training procedure. For example, *ImageNet*[4], a standard dataset for evaluating image classification in academia, is grown to over 14 million images [51]. To prevent *overfitting*, *i.e.*, the model memorizes the training samples instead of the general concept; thus, the data is further altered by data augmentation techniques like cropping, rotation, or noise injection [52]. This altering increases the amount of data for deep learning networks during training even further.

Moreover, a single training step *(backward)* is much more costly than the inference *(forward)* in the same network. Hence, training is usually performed on compute clusters with thousands of *deep learning accelerators (DLA)* with distributed learning frameworks. While there are efforts to train basic networks in minutes on these clusters [53], the time required for competitive state-of-the-art models is often measured

4 https://image-net.org/

in days [43]. When projected to a single core, it can be even in the thousands of days [41].

In almost any deep neural network, the backpropagation algorithm is the foundation to update the network parameters. In short, for each training sample, gradients are computed from the prediction back to every single trainable parameter.

FRAMEWORKS Deep neural networks have millions of trainable parameters in hundreds of layers [44]. Fortunately, most of the computation — at least per layer — is massively parallelizable. Popular deep learning frameworks like *Caffe*[5], *Tensorflow*[6], or *PyTorch*[7] provide easy access to highly efficient implementations for layer primitives and standard routines. For example, forward and backward passes for convolutions, poolings, normalizations, activations, and fully connected matrix-matrix multiplications.

These specially tailored hardware and software stacks, like *NVIDIA's cuDNN*, are the engines that enable the computational power that is ultimately needed for deep neural networks to shine. With these frameworks, researchers, engineers, and other early adopters can produce deep learning applications that effortlessly train and infer image-related tasks even on a consumer-graded GPU [40, 44]. Besides GPUs, there are also more specific *Deep Learning Processors (DLPs)*, like *Tensor Processing Units (TPUs)*, or *Neural Processing Units (NPUs)*.

Highly efficient algorithms designed explicitly for massively parallelizable hardware are critical for deep learning-based perception algorithms.

1.3 EXISTENCE ON AND OFF THE GRID

The framework provided layer primitives are heavily optimized and tailored toward their expected input data: **digital images** or other **tensor-like** representations. For example, convolution and pooling layers are usually defined and parametrized by their spatial neighborhood of pixels. Hence, these implementations can only work on data where the underlying representation follows a regular grid structure.

Depending on the subject and topic, the definition of the terms *structured* and *unstructured* can have different meanings [54–56]. This work will solely focus on the data layout and representation in the context of computer vision, *i.e.*, primarily sensor data and feature embeddings.

STRUCTURED DATA Typical examples for *structure data*, besides images, are videos as a sequence of images or voxel grids; see Figure 1.2.

5 caffe.berkeleyvision.org
6 tensorflow.org
7 pytorch.org

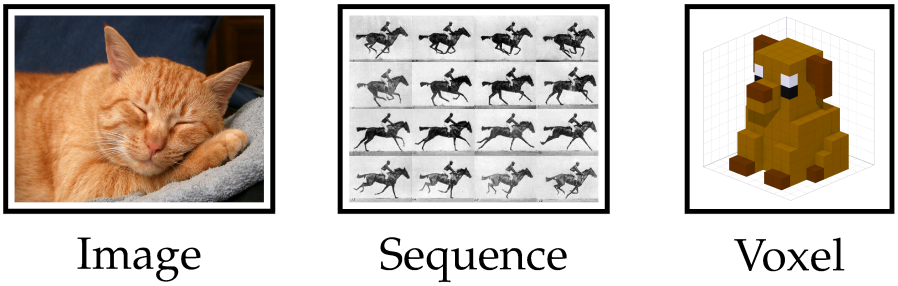


Figure 1.2: **Structured data** are tensor-like representations that are defined by a fixed grid with an implicit neighborhood. Typical examples are images, a sequence of images (video), or a voxel representation. *Cat image [57], Horse sequence [58].*

Their data layout consists of cells on a fixed grid with a predefined resolution. Each location is given implicitly by its position, and finding neighbors is a simple lookup.

In most cases, these cells represent not just single measurements but an aggregation area. Depending on the granularity, some information from multiple samples will be merged, *e.g.*, rasterization, or pooling, while other areas are empty or underrepresented.

Images are still the dominant input source in computer vision, especially in deep learning-based approaches. Nonetheless, computer vision is proficient with many sensory devices and techniques that produce representations beyond the well-defined grid world.

UNSTRUCTURED DATA *Unstructured* numerical data is no longer implicitly nor densely defined by an underlying construct like cells but can exist anywhere in a continuous space. With this, every point consists of coordinates defining the position within the space and potentially some additional information. Whereas often, the pure existence of a point already carries information.

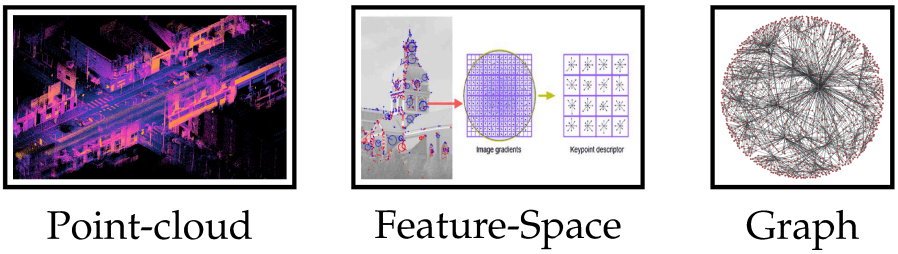


Figure 1.3: **Unstructured data** representations are explicitly determined by the data itself; coordinates define the exact location of the points. Common examples include point clouds, feature spaces, and graph structures. *Point cloud [59], Feature image [60]*

Generally, each point represents a single data measurement, *i.e.*, not aggregated in the coordinates' domain, and there is no uniform distribution; instead, data only is defined wherever it is located in the

space. Well-known examples from the family of unstructured data representations are point clouds, image features, or graph structures, see Figure 1.3.

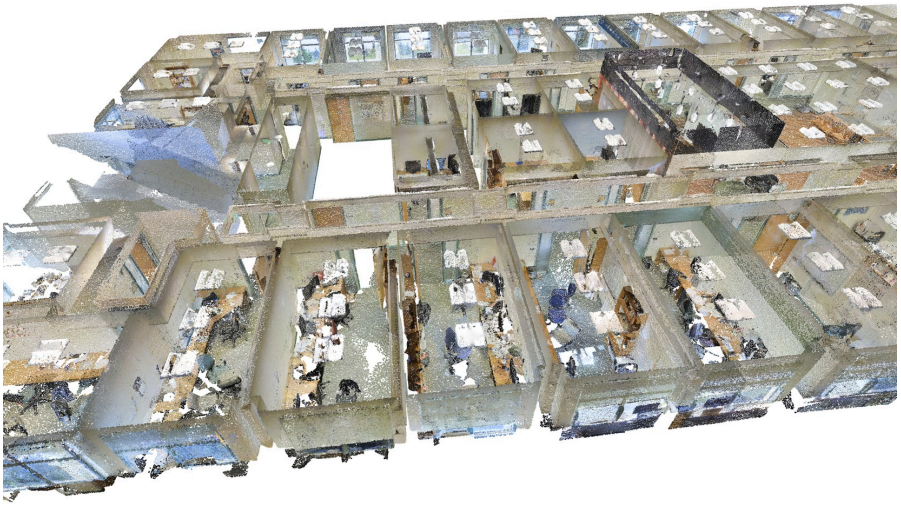


Figure 1.4: Example of a three-dimensional **point cloud** from an indoor scene captured by a Matterport device. Each point is represented by its coordinates in 3D and shaded with the additional stored color information. The visualized data is from *S3DIS* dataset [61].

POINT CLOUDS An established group of unstructured data representations in computer vision are three-dimensional point clouds. They are a natural choice to store estimated surface points from 3D reconstruction techniques of real-world objects and scenes. Figure 1.4 shows a three-dimensional point cloud from a reconstructed real-world indoor scene. The information obtained from multiple captures is aggregated and fused into a single common space. Each estimated surface instance is a single point in this shared space. In this example, the points also carry additional color information.

A major aspect of computer vision is estimating the geometrical relation of a scene with its capturing devices, describing how points of a scene are connected with the imaging sensors [22, 62]. The general concept is to approximate internal (intrinsics) and external (extrinsics) camera parameters. Intrinsics describe a camera's internal transformation of incoming photons, while extrinsics characterize a camera's pose concerning a world coordinate system, see Figure 1.5.

For estimating the internal parameters, usually, cameras pass a separate calibration process that fits captures of specific calibration targets against a pre-defined camera model. A high-quality calibration process can be time-consuming but is primarily stable per device. On the other hand, external poses, *i.e.*, position, and orientation, change per capture (view) within a scene. It is worth noting that most camera

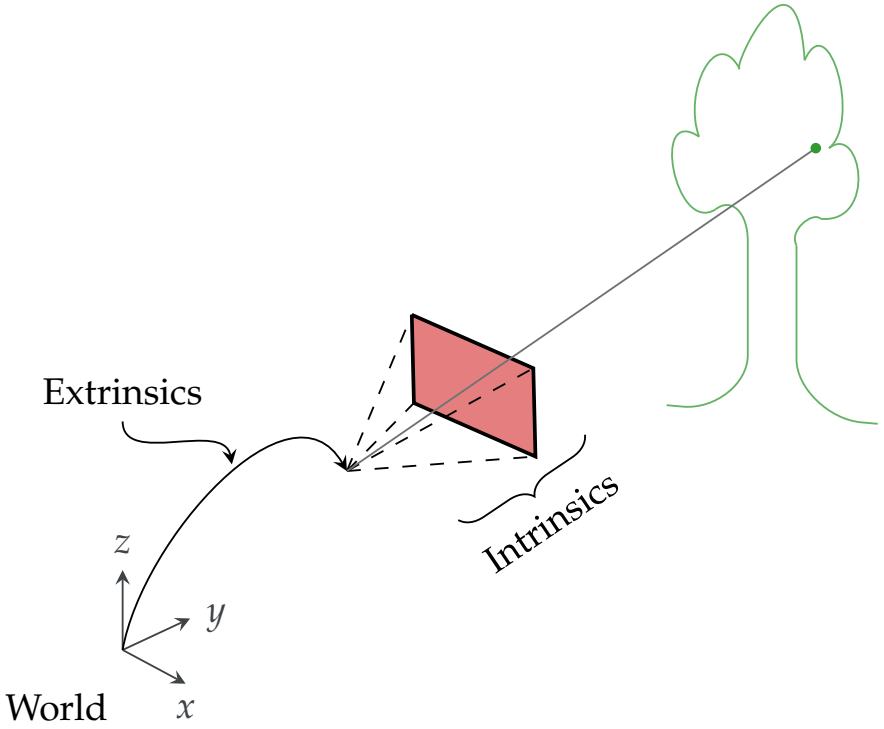


Figure 1.5: A **Scene** characterizes the outside world that sensors capture. While intrinsics describe the internal transformation from a scene point to the device, extrinsics locate a sensor in the world. This information is required to relate scene points from different views and devices.

models neglect noise characteristics of the image generation process because they play only a minor role in the geometry. However, they define working boundaries in practice and can be important when relating different sensors.

By relating corresponding points from multiple views of a scene to each other through projection and triangulation, it is possible to estimate the view's extrinsics. This is the basic principle of *Structure-from-Motion (SfM)* [63, 64] and *Simultaneous Localization and Mapping (SLAM)* [65, 66]. Multi-view geometry uses knowledge of the sensors and the physical world to map measurements from different viewpoints into a cohesive space, reassembling the external world.

Multiple 3D reconstruction and range scanning techniques are available to obtain a surface densely: *Structured Light (SL)* scanning, *Shape from X (SfX)* estimations, *Stereo Depth camera (Stereo)*, *Time-of-flight cameras (ToF)*, *Light Detection and Ranging (LiDAR)*, or *Radio Detection and Ranging (Radar)*. While some methods (*SL* and *Stereo*) rely on traditional cameras that estimate the shading of the scene, time-of-flight-based devices(*e.g.*, *LiDAR* and *Radar*) estimating distance directly

by measuring the time a signal took from the sensor to a surface and back. Although the latter produces depth info instead of capturing illumination, both types are describable as imaging camera systems.

In combination, these techniques, in one way or another, decompose the physical signal-capturing process to transform and aggregate sensory results into a common representation by relating multiple measurements. For example, a full-fledged application for *SfM* and *Multi-View Stereo* is COLMAP [64, 67].

Figure 1.6: **3D Reconstruction Devices:** A selection of commercially available devices that fuse camera information with dense range scanning. The Pro2 uses *SL*, while the BLK2FLY and the Hovermap utilize *LiDAR*.

For a long time, multi-view geometry was a driving force in CV research. Accurate reconstructions of large-scale real-world scenes have involved considerable effort and domain knowledge. Nowadays, solutions and products from this field are available, making three-dimensional capturing simple and hassle-free. For example, companies like Matterport[8], Leica Geosystems[9], or Emesent[10] are offering devices that effortlessly capture the 3D surfaces of large scenes, by simply moving or even flying them around. See Figure 1.6. These devices combine pose estimations (*SfM* and *SLAM*) with range scanning techniques (*SL* and *LiDAR*) and photometric cameras to create dense real-world point clouds with color information, as shown in Figure 1.4.

Autonomous vehicles typically have a whole sensor suite with various devices to cover each other's limitations. Embedding all sensors in the same space is beneficial for joint reasoning.

There is an increasing need for perception tasks to work on data produced by these techniques and sensors. In particular, for tasks that have improved drastically through deep learning methods, like semantic segmentation and object detection.

A significant advantage of deep networks is that they can be robust against many changes in specific domains directly through their latent space (weights), *e.g.*, different illuminations or translations of the same

8 https://matterport.com/
9 https://leica-geosystems.com/
10 https://emesent.com/

object in images. In a typical deep learning fashion, an approach could be to learn everything end-to-end by inputting the sensor information as images and image-like tensors. However, relating measurements from varying viewpoints is a non-trivial task to model in an end-to-end network [68]. There are dedicated networks, but feature-based methods like COLMAP are still superior and often regarded as ground truth [68].

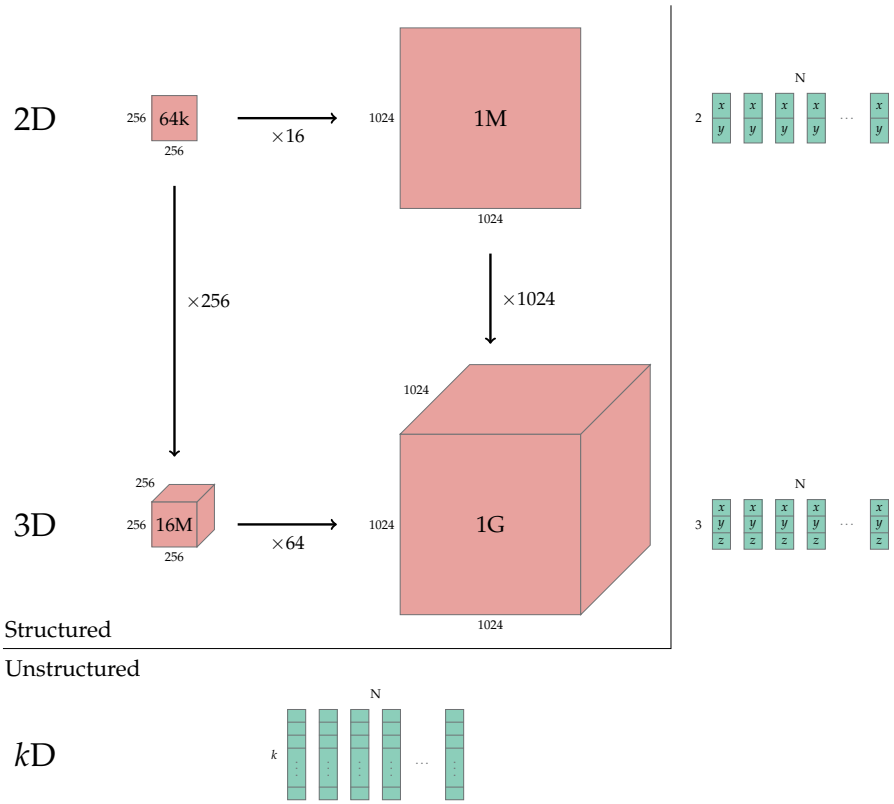


Figure 1.7: **Curse of dimensionality:** Structured data, with its fixed grid, suffers greatly from the curse of dimensionality. While increasing a length by a factor of 4 in 2D is 16 ($= 4^2$) times the memory requirement, in 3D it is already a factor of 64 ($= 4^3$). On the other hand, unstructured data only increases linearly with the number of stored coordinates.

CURSE OF DIMENSIONALITY Rasterizing the unstructured data in a common and fixed grid seems very tempting to enable the highly optimized frameworks and implementations again. In 3d, where the information is discretized into voxels, this is used successfully in multiple projects [69, 70]. Voxel-based approaches work well for tasks where the global shape is most important, *e.g.*, object classification on single objects. However, it has disadvantages in challenges where the resolution can not pick up the necessary details, larger receptive fields are essential for the proper context, or varying densities are needed at

different locations. In particular, large-scale real-world point clouds are challenging to process with lower-resolution grids.

Unfortunately, just increasing the resolution comes at a high cost. For example, in a typical voxel grid, even for a base resolution of 256 bins per axis, 16 million cells already need to be processed and stored. Increasing the grid structure to a cubic resolution of 1024 elevates the number of cells to over 1 billion. Compare Figure 1.7. Even in the three-dimensional case, the curse of dimensionality already strikes quite fiercely. In high-dimensional scenarios, this gets much worse.

In most scenes, the vast majority of space is empty. There are better methods to compress the used space, *e.g.*, octrees, as used by *OctNet* citeriegler2017octnet. However, they again leave the well-defined grid world; thus, they are no longer in the domain of highly efficient implementations for learning, either.

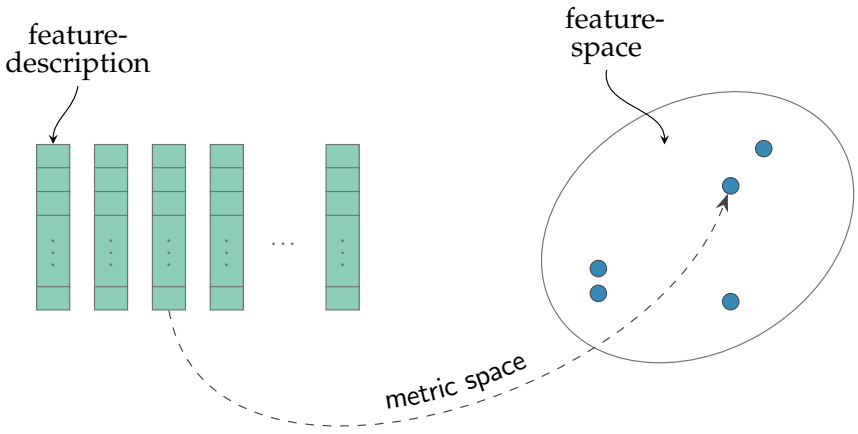


Figure 1.8: A **feature space** embeds knowledge about entities in a common space as points. A metric function measures the similarity between points. A feature space is usually used to re-identify or find similar entities.

FEATURE SPACE Another interesting group of unstructured data representations is a feature space. In practice, a feature can be any specific entity. For example, in computer vision, a feature can be a point in an image (local) or the whole image (global). There are two primary challenges for which features are used. First, re-identifying the same entity across multiple occasions, and second, searching for similar instances in an extensive database. Therefore, information about the entity is embedded in feature vectors, and a metric estimates the similarity. For example, local image features are used to find the same point in two different views of a scene (correspondence), while global image features are applied to recognize similar objects.

Usually, these feature descriptions are high-dimensional, and each one is a single instance in a shared feature space. Hence, the feature vectors (descriptors) are points (position) in a high-dimensional space; see Figure 1.8.

While, traditionally, feature descriptors are handcrafted (*e.g.*, *SIFT*), recently, more and more deep learning-inspired features are emerging (*e.g.*, *neural codes* [71]). Furthermore, with the current trend in *foundation models*, the amount of data embedded in feature vectors is increasing even further. This progress is unlikely to stop, which makes learning and searching in feature spaces a pivotal field.

1.4 THE PILLARS

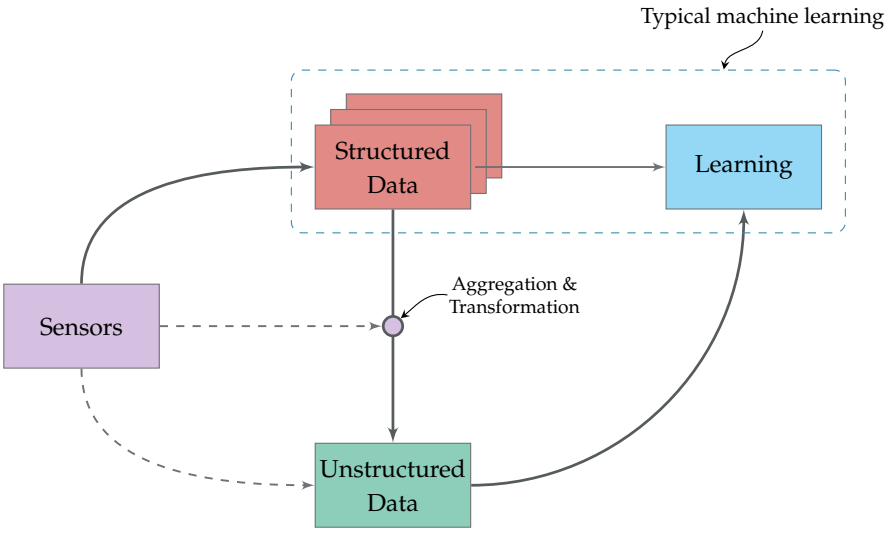


Figure 1.9: Computer vision-based machine learning, especially deep learning, is typically performed on images (*i.e.*, structured data) as input. **Learning on unstructured data** aggregates and transforms the structured data with additional knowledge into a common space and uses it as input.

Machine perception goes beyond the sensory capabilities of humans. Unstructured data representations are the natural choice to embed knowledge like geometric structure or feature properties from different captures in one common domain. They provide inherent access to the rich information that is difficult to retrieve from images alone. Further, they scale by the number of data points, not by a grid's resolution. Unstructured data approaches can seamlessly work on spatially varying, high-dimensional data with little to no extra memory overhead.

The driver of computer vision algorithms is undoubtedly machine learning, particularly deep learning. However, getting off the grid loses access to leverage most of the highly efficient framework routines that are only defined on the implicit layout of structured data. Hence,

highly efficient implementations are crucial to enable learning on unstructured data.

Every method presented in this thesis is designed explicitly from the ground up as a massively parallelizable algorithm to fully leverage the power of GPUs. Implementations are carried out in NVIDIA's *CUDA* programming language, while host code is mainly written in C++ with some Python for interfaces and deep learning model descriptions.

All of this can be condensed into three pillars describing the cornerstones for the featured solutions.

THE PILLARS

1. **Unstructured data**: Algorithms are designed particularly for unstructured data representations.
2. **Efficiency**: Methods are massively parallelizable on GPUs by their *simple* and *effective* nature.
3. **Robust**: Approaches aim to be *robust* against real-world phenomena like noises and different kinds of illumination effects.

FEATURED SOLUTIONS The foundations for this thesis are four projects showcasing different techniques for processing and learning on large-scale, real-world, unstructured data. Two focus on point clouds, and the others on feature spaces, respectively graphs. See Figure 1.10 for an overview sketch. The published papers and the pre-print are included in the Appendix (Part ii). The next chapter establishes the base for unstructured data processing and interconnects the four solutions.

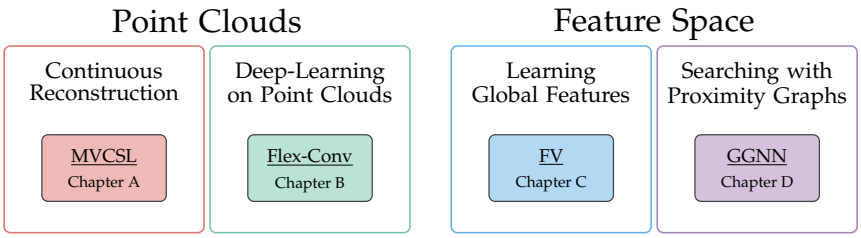


Figure 1.10: **Featured Papers**: Classification of the four projects that form the core of this thesis.

MVCSL [1]: "Multi-view continuous structured light scanning." Fabian Groh, Benjamin Resch, and Hendrik P.A. Lensch *German Conference on Pattern Recognition (GCPR)* (2017)
This approach uses active sinusoidal wave patterns from multiple projectors to form a spatially continuous signal space on the scene.

It improves sparse bundle adjustment from multiple views and generates highly accurate and precise point clouds even for challenging real-world objects with occlusions by finding optima in the signal space.

Flex-Conv [2]: "Flex-Convolution: Million-scale point-cloud learning beyond grid-worlds." Fabian Groh, Patrick Wieschollek, and Hendrik P.A. Lensch *Asian Conference on Computer Vision (ACCV)* (2018)
This work enables fully convolutional neural network architectures on point cloud input data by developing specific generalized layers that perform directly on the unstructured data representation. The performance is demonstrated on a per-point semantic segmentation task that shows the benefit of an increased receptive field when processing multiple millions of points in one forward step.

FV [3]: "Backpropagation training for fisher vectors within neural networks." Patrick Wieschollek*, Fabian Groh*, and Hendrik P.A. Lensch *arXiv preprint arXiv:1702.02549* (2017)
This object detection framework uses well-established dense *SIFT* features as higher-order inputs for end-to-end learned Fisher-Vector-based global image feature embeddings with an SVM classifier.

GGNN [4]: "GGNN: Graph-based GPU nearest neighbor search." Fabian Groh, Lukas Ruppert, Patrick Wieschollek, and Hendrik P.A. Lensch *IEEE Transactions on Big Data* 9.1 (2022)
This solution demonstrates a proximity graph-based approximate k-nearest neighbor search and construction schema on GPUs for high-dimensional billion-scale data. It significantly outperforms previous state-of-the-art methods regarding accuracy, search speed, and graph construction time.

SUMMARY This thesis proposes various efficient, effective, and robust solutions for large-scale real-world unstructured data representations, which allow incorporating knowledge that is otherwise difficult to represent, like geometric relation between views, 3D surface information, or image feature embeddings.

2

FROM STRUCTURED TO UNSTRUCTURED DATA

This chapter defines and describes the relationship between structured and unstructured numerical data representations in the context of computer vision. Generally, the structured information gets aggregated into a common unstructured representation. See Figure 2.1. Such transformations carry additional knowledge that would not be available from a single image.

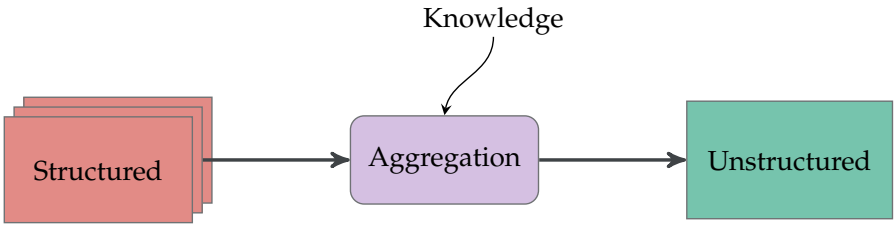


Figure 2.1: From **Structured** to **Unstructured** data representations.

The following sections will establish the foundation and interconnection for the proposed solutions. Different forms of unstructured data representations are discussed, namely *point clouds*, *feature spaces*, and *proximity graphs*. At the end of every section, a specific "*Efficient Processing and Learning on ...*" paragraph outlines and categorizes the presented projects, which are attached in the Appendix (Chapters A - B). These appended papers are self-contained articles that can also be read independently. Nonetheless, this chapter will benefit the reader greatly by providing the underlying story of *Efficient Processing and Learning on Unstructured Data*.

2.1 UNSTRUCTURE THE IMAGE DATA

Computer vision's primary input is digital images. Cameras replicate the function of human eyes to perceive the world visually.

If not otherwise noted, the term *image* will be a synonym for a digital image throughout this thesis. Likewise, a digital camera will be referred to simply as a camera.

An image I consists of *pixels (picture elements)* defined on a 2D regular grid. Each pixel corresponds to a finite location $(x, y) \in \mathbb{Z}^2$ on a structured rectangle defined by width w and height h:

$$I = \{(x, y) \in [0, w] \times [0, h]\} \subseteq \mathbb{Z}^2 \,. \tag{2.1}$$

A pixel (x, y) stores intensity values $v(x, y) \in \mathbb{R}$ as a numerical representation. In grayscale images from monochromatic cameras, v is a

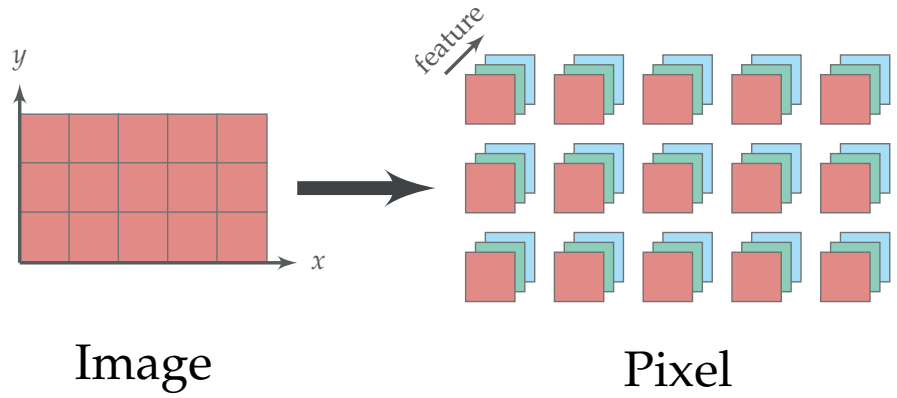


Figure 2.2: An **Image** is a structured representation with a regular grid. The elements are called pixels and typically carry intensity values as a feature vector.

single number describing the shades of gray, also known as the gray level. In *RGB images*, v is a vector corresponding to red, green, and blue intensity values. Typically, these values are quantized into 8-bit integer numbers $v_i(x, y) \in [0, 255]$. In the more general terminology of data representations, such a vector of information per data point is also called a *feature vector* or just *feature*. At the same time, a specific layer in the vector is often referred to as *channel*. Figure 2.2 illustrates the structure of an image.

FEATURE TERMINOLOGY Throughout this thesis, the word *feature* is used in different occasions. Generally, the term *feature* in computer vision and machine learning is very broad and usually tied to the specific task for which it is used [72]. Two meanings are distinguishable but still related:

- The *location* of a point that satisfies specific cues.
- The *content* that is attached to a point and stores associated information.

In this context, *feature detectors* are used to *locate* points of interest, while *feature descriptors* are used to describe the *content*. This thesis focuses on the data-representative aspects of the latter. Classic examples are hand-crafted local image features like *SIFT* [31], *SURF* [73], or *ORB* [74] representing local illumination distributions as a high-dimensional feature vector for correspondence matching [75]. Local image features are discussed in more detail in Section 2.3.1.

Deep neural networks, particularly *Convolutional Neural Networks (CNN)*, encode their latent information in the feature vectors of image pyramids with multiple layers [37]. Recently, neural network-embedded image features that use this fact are also emerging, like *SuperPoint* [76].

CONTINUOUS IMAGE PLANE When visualizing an image, pixels are usually represented by square patches uniformly shaded with their respective intensity values. Adjacent cells can differ in value, forming a dense and connected visual image effect.

In photometric camera systems (*e.g.*, *CMOS*), an image sensor captures the incident light, or electromagnetic radiation in general, by converting photons into current. Consequently, more photons will increase the signal, *i.e.*, higher intensity values. In simplified terms, the image sensor has an individual and spatially separated photodetector for every pixel on its rectangular plane. Thus, each pixel represents its own measurement and can be depicted as a single point on the structured grid. A typical convention is to use the pixel center.

With this in mind, an image is a rectangle with continuous coordinates $(x, y) \in [0, w] \times [0, h] \subseteq \mathbb{R}^2$, and the values $v(x, y) \in \mathbb{R}$ are interpolated from the known measurements. Although an *RGB image* has three channels, only a single intensity value is typically captured per pixel on the sensor itself. The image sensor has an attached *color filter array (CFA)*, allowing only specific bandwidths (colors) to pass on every pixel. A traditional *CFA* is the Bayes filter, which distributes spatially red, green, and blue filters. The demosaicing algorithm then estimates a color value for each color channel. The result is the typical three-channel RGB image.

CAPTURING CONDITIONS While each pixel of an image can be treated as an individual measurement, they also have some properties in common. All are captured with the same camera, with the same view, in the same scene, and in the same time interval. More generally, sensors capture the world under specific conditions. Some properties might be known; some might even be controllable; others are hidden or irrelevant. For example, when taking a picture, the time might be known, but in many cases, it is unimportant.

Environmental lighting conditions have a major impact on images. It is not seldom a primary reason for a shot in aesthetic landscape imagery. While images are taken due to specific lighting effects, those conditions are beyond our reach to control. Exceptions are photometric laboratories that utilize specific devices like a *light-stage* [77], where an object is illuminated by a predefined lighting setting that includes the direction and wavelength. The setup in Chapter A is purposely built to operate inside a *light-stage* to produce geometric alongside photometric surface information. In most cases, however, environmental lighting is not entirely controllable.

On the other hand, the position and orientation of the capturing device (*extrinsics*) and its optical settings, such as field-of-view, aperture, and exposure (*intrinsics*), are controllable or can be estimated.

All conditional parameters influencing a sensor's measurement are *independent variables* θ, whether controllable or not. The outcome f of

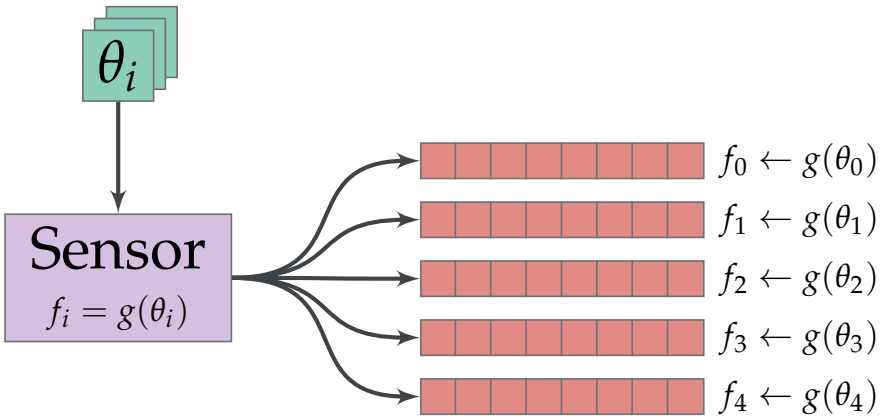


Figure 2.3: **Sensors** obtain the world under specific conditions θ. Each measurement f depends on the sensor's conditioned sensing process $g(\theta)$.

the sensing process $f \leftarrow g(\theta)$ are *dependent variables*. Compare Figure 2.3, for a specific sensor, any measurement f_i directly correlates to the parameters θ. Due to the time component, it is theoretically impossible to have twice the same output; however, *e.g.*, for a static scene, the difference might be negligible. In the case of photometric cameras, f are intensity values. For time-of-flight cameras, the measurements are the round-trip times of an active signal from a transmitter to a receiver, which are recomputed as depth values from the sensor to a surface.

The controllable subset of θ is mostly minimal, while the unknown, irrelevant, or uncontrollable part $\hat{\theta}$ is large. For example, if there is just a single *RGB image* with no further context. Then, it is unknown how the image was obtained, where it was captured, or what was expected in the image. In that case, the controllable part of the independent variables shrinks to the image plane coordinates, and the dependent variables are intensity values r, g, b of the specific color spectra:

$$[r, g, b] \leftarrow g((x, y), \hat{\theta}) \,. \tag{2.2}$$

2.2 THE SCENE

A major aspect of computer vision is estimating the geometrical relation of an imaging sensor to a scene. The following terminology and categorization will give an overview of the main components involved as they are used in this thesis. Refer to Figure 2.4 for further visualization.

Scene: The three-dimensional physical world captured by *sensors*. It is often assumed to be static and limited in location for simplicity. In general, it is not. Every aggregation or combination of data from different *views* has a common *scene*. Therefore, a *scene* origin is defined

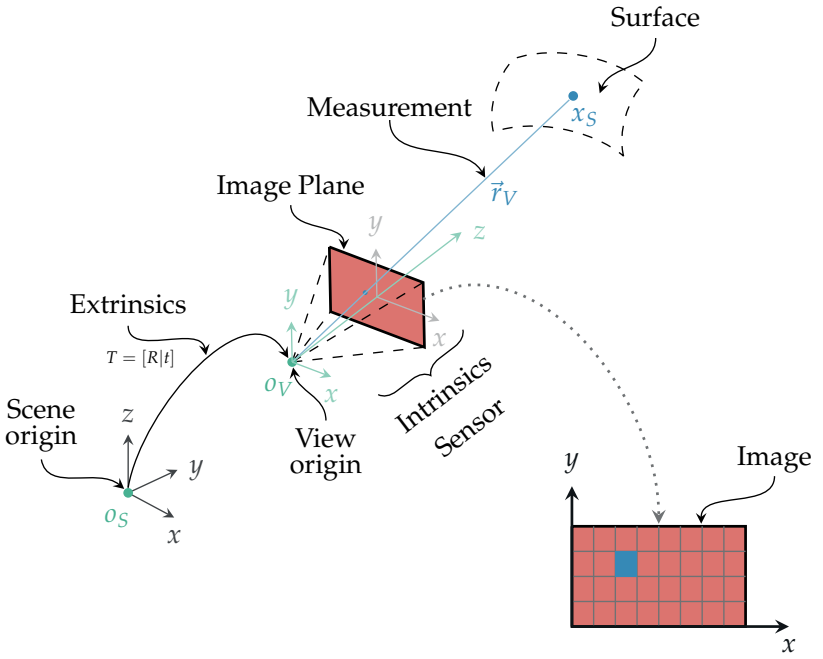


Figure 2.4: The **scene's components** describe the coherences required to map scene points x_S with observations from images. *Extrinsics* relate a sensor's *view origin* o_V with the *scene origin* o_S as a 6DoF pose transformation T. The *intrinsics* describe the sensing process for a *measurement* from a surface point to a sensor's *image plane*, respectively, the *image* itself.

that anchors every *view*, respectively *measurement*. A *scene* is also often referred to as *world*, with its origin as *world* origin.

View: Position of a *sensor* within a *scene*. One *sensor* can have multiple *views*. Every *measurement* is associated with a *view* at a specific time. *Extrinsics* describe the relation of a *view* with its *scene*.

Sensor: Any perceptional device that produces *measurements* from the *scene*. Typical vision examples are photometric or time-of-flight cameras. But there are also imaginary devices based on other techniques, *e.g.*, ultrasonic, computed tomography, or magnetic resonance imaging. *Intrinsics* describe the internal measuring process between the *sensor* and the *scene*. Self-aware *sensors* like gyroscopes, magnetometers, or barometers can support multi-sensor agents in localization within dynamic *scenes*.

Measurements: The captured signal of a *sensor* at a specific time. Usually, the signal is aggregated over time and quantized in intervals. An example is a photometric image with intensity values per pixel corresponding to irradiance, *i.e.*, the number of photons hitting the pixel's image sensor surface during the exposure time. As described

before, a single pixel can already be considered a measurement.

Intrinsics: Description of the *sensor's* signal processing from a *scene* to a *measurement*. Typically for cameras, lens distortions, exposure time, focal length, principal point, resolution, field-of-view, aperture, shutter, and gain. Everything performed on the sensor, like white balancing or quantization levels, can also be included. Even noise characteristics from fixed-pattern to dark current can be considered part of intrinsics. Most of the *intrinsics* of a *sensor* are treated as static during a *scene*, even over multiple *views* of the same *sensor*. Commonly, a calibration procedure attempts to estimate these values in advance. And they can be further optimized if the data is post-processed, like in bundle-adjustment approaches. For cameras, *intrinsics* reduce the complex signal processing procedure to a simple camera model like pinhole, *i.e.*, casting rays from a sensor origin to the scene or vice versa. The specifications of *intrinsics* differ heavily from sensor to sensor.

Extrinsics: Transformation between a *view's* coordination system and the origin of the *scene*. In contrast to *intrinsics*, the structure of *extrinsics* is the same for all *sensors*. In Euclidean space, it has 6 degrees of freedom split in rotation and translation. The translation consists of x-, y-, and z-coordinates, while the rotation has yaw, pitch, and roll angles. *Extrinsics* are also called camera poses, and estimating them is an essential part of multi-view approaches like Structure from Motion.

IMAGE FORMATION The image formation process [78, 79] is well-studied. For the sake of simplicity, in the remainder of this chapter, any vision sensor is assumed to be reduced to the simple pinhole camera model by its known intrinsics. This assumption includes that every point of interest is in focus. Figure 2.4 illustrates this model as it relates to the components of a scene.

If the geometric transformation is given, the relation between a scene's surface point x_S and a point on the image plane x_P is a ray cast within the camera's coordinate system. This simple model uses a forward-facing (virtual) image plane for more straightforward visualization and computations. The camera's origin is at the center of projection, *i.e.*, the focus point of all incident light rays, and the image plane P is at the distance of the focal length f orthogonal to the viewing direction along the z-axis. Figure 2.5 illustrates such a pinhole camera model in 2D.

Let's assume a known distance d for a point x_I in an image I with width w, height h, and field-of-view fov. Then, the relation between

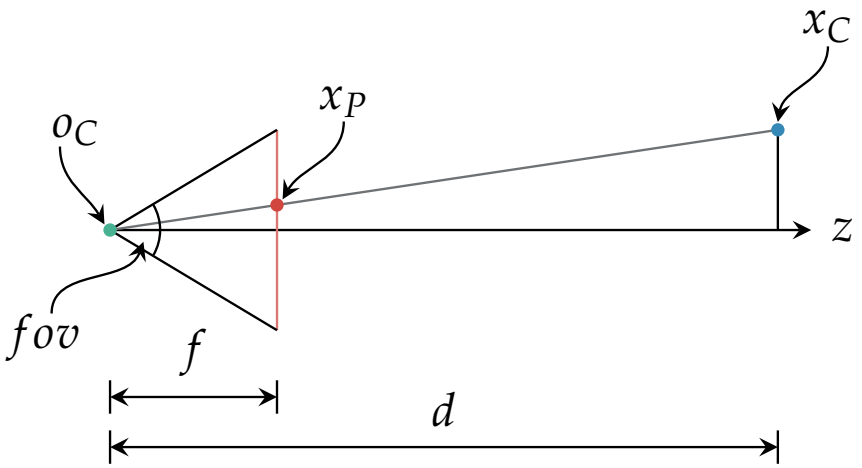


Figure 2.5: In the simple 2D **pinhole model**, points on the image plane x_P with their corresponding points in the camera coordinate system x_C are connected by a line starting at the camera origin o_C. All image plane points are at the distance of the focal length f along the z-axis, while points in the camera coordinate system have an individual depth d.

pixel coordinates in the image space and the camera space can be described by the scale value a:

$$a = \begin{pmatrix} \frac{2f \tan\left(\frac{fov_x}{2}\right)}{w} \\ \frac{2f \tan\left(\frac{fov_y}{2}\right)}{h} \end{pmatrix} \tag{2.3}$$

Consequently, getting from the image coordinates $x_I \in \mathbb{R}^2$ to the image plane $x_P \in \mathbb{R}^2$ is given by:

$$x_P = \begin{pmatrix} (x_I - \frac{w}{2})\, a_x \\ (y_I - \frac{h}{2})\, a_y \end{pmatrix} \tag{2.4}$$

With the intercept theorem, a 3D scene point $x_C \in \mathbb{R}^3$ in the camera coordinate system can be determined by its depth $d \in \mathbb{R}$:

$$x_C = \begin{pmatrix} \frac{x_P}{f} d \\ \frac{y_P}{f} d \\ d \end{pmatrix} \tag{2.5}$$

The reverse is also often required when a scene point is known, but the corresponding image point is needed. Projecting a point back into the image space is known as *back-projection* or *reprojection*.

RAYS Another way to look at the camera model shown in Figure 2.5 is as a linear model or ray casting problem:

$$x_C = o_C + u\,\vec{r}_C \tag{2.6}$$

where $\vec{r}_C$ is the ray connecting the camera origin o_C with the scene point x_C through the image plane P at x_P in camera coordinates, and $u \in \mathbb{R}$ is the depth along the ray. The camera origin o_C is usually set to zero in a single view. Hence, the equation reduces to:

$$x_C = u\,\vec{r}_C \tag{2.7}$$

With $\vec{r}$ being a unit vector, the connection between the orthogonal depth d along the z axis and the depth along the ray u is given as a scaled dot product:

$$d = \langle u\,\vec{r}_C, \begin{pmatrix} 0 \\ 0 \\ 1 \end{pmatrix} \rangle\,, \quad \text{respectively:} \qquad u = \frac{d}{\vec{r}_{z_C}}\,. \tag{2.8}$$

DEPTH INFORMATION The intensity values from a photometric image might be used to determine where a reflective surface is and where it is not. However, they will not indicate the depth of a point. While humans can estimate depth even from a single mono-camera photo by combining semantic knowledge and experience, this is not easily possible for a computer system without such cues. Thus, the actual distance u to the surface of a pixel is unknown and can be anywhere along the ray $\vec{r}$.

Some vision sensors can estimate depth directly. Examples are time-of-flight, structured-light, or stereo cameras. An image with a depth value per pixel instead of intensities is called *depth image* or *depth map*. Very recently, more and more deep learning approaches have succeeded in estimating depth from monocular cameras [80, 81].

2.2.1 *Point Clouds*

With the help of Equation 2.7, every pixel with a valid depth value can be transformed from the 2D plane into the 3D space. The result is a *point cloud* X_C resembling the surface from a single view:

$$X_C = \bigcup_{i \in I} \{\, u_i\,\vec{r}_{C,i} \,\} \tag{2.9}$$

By its nature, a single-view point cloud is inherently missing surface areas due to occlusions and limited coverage. A point cloud generated

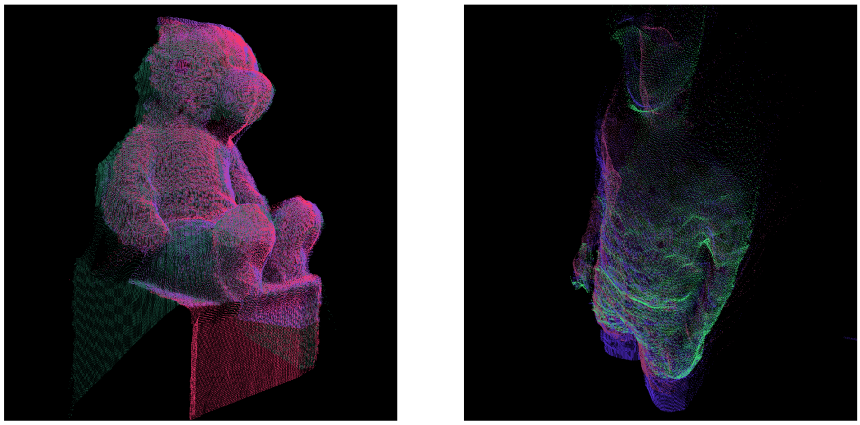


Figure 2.6: The coverage of a **Single-View** is limited. In these two examples, each color corresponds to a single view. *Images from [82].*

from a single view is also referred to as 2.5D to indicate this difference. The two examples in Figure 2.6 show this effect color-coded, where each color represents a single view. By themselves, they would miss essential details.

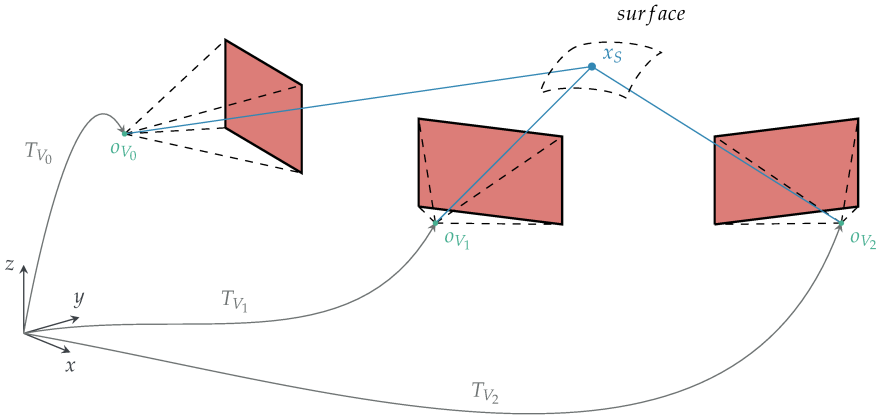


Figure 2.7: In a **Multi-View** setup, multiple views observe the same scene. Each view has its own extrinsics and potential intrinsics. Hence, the information from the local camera coordinate systems per view can be transformed and aggregated into a common representation that overcomes the limitations of a single view.

MULTI-VIEW POINT CLOUDS Combining multiple views into a joint point cloud circumvents these limitations. Additionally, they potentially obtain the same surface from multiple views. Figure 2.7 sketches a multi-view setting. This requires knowing the extrinsics in relation to a common scene origin.

Extrinsics are typically expressed as a rigid-body transformation matrix T with 6 *degrees of freedom (DOF)*, separated in rotation $R \in SO(3)$ and translation $t \in \mathbb{R}^3$. In *homogeneous coordinates*, this is expressible as a linear transformation matrix:

$$T = \left(\begin{array}{c|c} R & t \\ \hline 0 & 1 \end{array}\right)$$

The mapping from a point cloud in camera space X_C to the pose in the joint representation X_V can be written as a matrix-matrix multiplication. Let the points in X_C be represented as a $4xn$ matrix of homogeneous coordinates, with the point entries in the columns. Then, the points are transformed by the corresponding extrinsics T_V:

$$X_V = T_V \cdot X_C \tag{2.10}$$

RAYS, RAYS, RAYS An alternative way is to approach the problem again from a geometrical aspect similar to Equation 2.6. In a linear system, the translation t is the view's origin o_V, and the viewing direction determines the rotation R. Thus, a ray $\vec{r}_C$ has to be rotated accordingly to match the direction of the pose:

$$\vec{r}_V = R_V \cdot \vec{r}_C \tag{2.11}$$

Every mapped point of a view x_V is still determinable by its depth u along a ray r_V:

$$x_V = o_V + u\,\vec{r}_V \tag{2.12}$$

Fusing the data across all views creates a 3D point cloud X_S of the scene that has all the data merged:

$$X_S = \bigcup_{i \in S} X_{V_i} \qquad X_{V_i} = \bigcup_{j \in V_i} \{o_V + u_j\,\vec{r}_j\} \tag{2.13}$$

MULTI-VIEW CONSISTENCY When multiple views observe the same surface point, then each view has its individual depth and ray trajectory to represent this point. Generally, any point x_s in 3D can be projected to every image plane P_{V_i}, see Equation 2.5. Of course, this will often be outside the boundaries of the respective image I_{V_i}. In ideal circumstances, if a point x_s is obtained from two views V_0 and V_1 with know poses, then there is only a single solution with (u_{V_0}, u_{V_1}), see Figure 2.8.

Let's take a list of points $[x_{P_0}, x_{P_1}, \cdots]$ from different image planes that all correspond to the same point x_s on the scene. For the correct camera poses, reprojecting this point to every image plane should

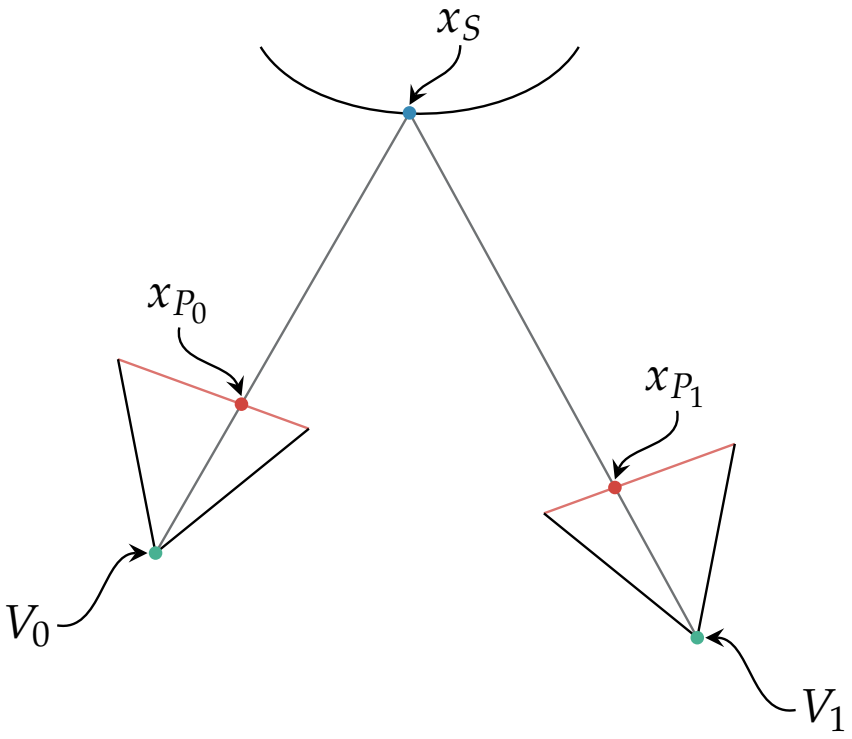


Figure 2.8: **Multi-View consistency**: Corresponding image plane points x_P need to align with their respective scene point x_S. These constraints over multiple correspondences and views allow for estimating the underlying poses.

result in the corresponding image point. The residual r_{P_i} is the distance between a reprojected point $\hat{x}_{P_i}$ and the original point x_{P_i} on the image plane:

$$r_{P_i} = \|x_{P_{V_i}} - \hat{x}_{P_{V_i}}\|_2 \tag{2.14}$$

Having multiple correspondence points well distributed over all views and geometry constrains the problem further. A least-square error for all sets of correspondences $\dot{X}_S$ is then given by:

$$\bar{e}_{X_S} = \sum_{x_k \in \dot{X}_S} \sum_{V_i \in V(x_k)} (x_{P_{V_i},k} - \hat{x}_{P_{V_i},k})^2 . \tag{2.15}$$

Theoretically, the error should be zero in a perfect world where everything is known. In practice, however, this is far from the truth.

Nonetheless, given enough correspondence in a scene, it is possible to reconstruct the camera poses by optimizing the error without knowing the depths u of the points, *i.e.*, only from photometric images. Despite the individual projections being linear, the problem is typically solved in a non-linear least-squares manner for practical reasons [64, 83]. Such solvers, which use a technique like the *Levenberg-Marquardt (LM)* algorithm, are able to deliver reasonable estimates even when

initialized suboptimally [62, 83, 84]. But, there is no guarantee of convergence to global minima. Optimizing camera pose estimations via global bundle adjustment is the backbone of *SfM* algorithms [64, 85]. The task boils down to finding a set of high-quality correspondences $\dot{X}_S$. Section 2.3 will show how to use high-dimensional feature descriptors for finding similar points over multiple views in passively acquired images.

However, since only correspondence points are known until now, the point cloud is very sparse. *Multi-View Stereo (MVS)* methods can estimate a dense representation from poses by applying photometric multi-view consistency across all image planes [22, 86]. In principle, obtaining a depth estimate for any image point is possible if supported by other views.

2.2.2 *Featureful point clouds*

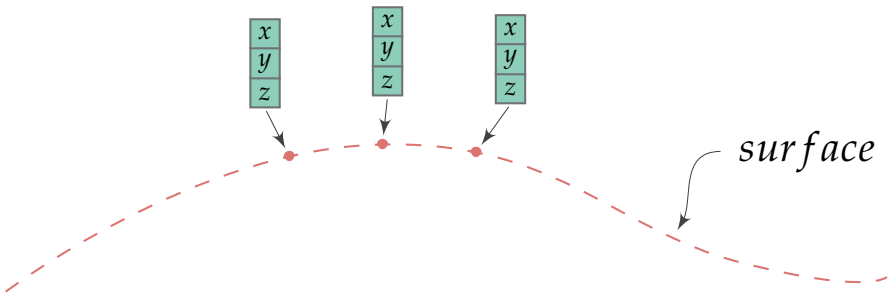


Figure 2.9: **Coordinates** define the location of each point in a point cloud. In tasks like 3D reconstruction, this information of existence is sufficient.

Until now, only positional information (coordinates) per point with $x_s \in \mathbb{R}^3$ has been considered. See Figure 2.9. However, every point can carry arbitrary data as its feature information. The most obvious is to use the intensity values from the underlying images. But, many more dependent data can be stored in the feature vector of a point. For example, any sensor or view information, time of the capture, or known environmental conditions.

CONTINUOUS POINT CLOUDS Given every point's view and depth information, one problem with the point cloud definition of X_S in Equation 2.13 is that it projects every measurement of every view separately. If this is done per pixel of the involved images, the result is a discrete point cloud with points from different viewpoints next to each other. Compare Figure 2.10. Every point x_s only has information from its source, and no data is shared.

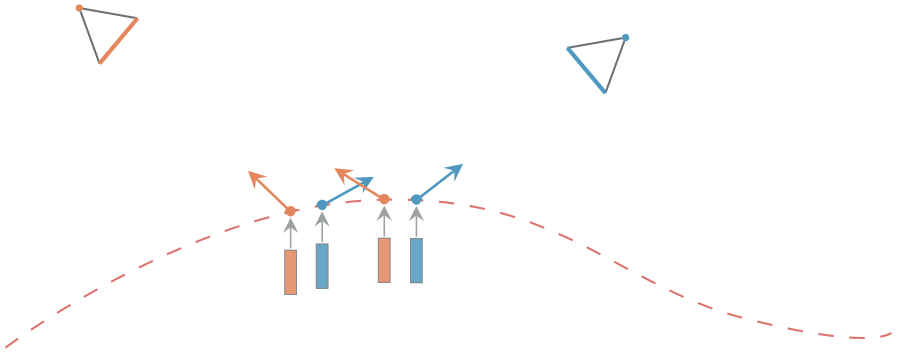


Figure 2.10: In **discrete point clouds**, every point represents a single measurement. Hence, points only have the information assignable from the respective view.

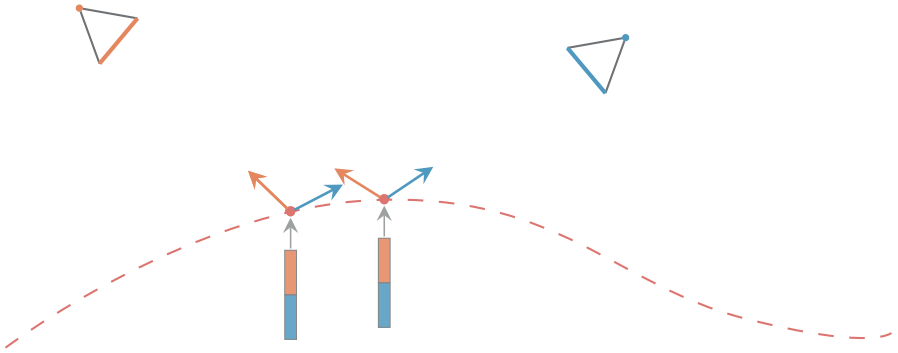


Figure 2.11: In **continuous point clouds**, points are no longer dedicated to measurements but can host information from all contributing views. Any observed surface point can be reprojected to the image planes.

Instead, if points directly on the estimated surface are used, the back-projection gathers information from all contributing views. Such a continuous point cloud is sketched in Figure 2.11.

2.2.3 *Efficient Processing and Learning on Point Clouds*

The geometrical knowledge about the scene, the sensors, their poses, and the involved physics enable the aggregation of the measurements from multiple images into a single representation, a point cloud. This unstructured data representation provides insights far beyond the possibilities of the images by themselves. Multi-view systems can mitigate real-world issues like occlusions, specular reflections, insufficient resolution, or inadequate exposures.

Multi-view continuous structured light scanning
Fabian Groh, Benjamin Resch, and Hendrik P.A. Lensch
German Conference on Pattern Recognition (GCPR), 2017 [1]
(Chapter A)

The work demonstrates a robust and high-precision 3D shape reconstruction scheme in a multi-view setup. Several projectors actively illuminate the scene with spatially continuous phase-shifting patterns, while the signal is obtained from multiple camera views in HDR. Figure 2.12 shows three spatially shifted sinusoidal wave patterns and the resulting computed phase signal. The result is repetitive but continuous.

Figure 2.12: High-frequency **continuous phase signals** (ϕ) reconstructed from projecting shifting sinusoidal wave patterns (θ_i) on the scene. The color gradient represents phase values from $-\pi$ to π and repeats over the scene. *Images from [1] (Chapter A). Reproduced with permission from Springer Nature.*

For pose estimation, the projectors are treated similarly to cameras. However, they are emitting light instead of receiving it. Incorporating the continuous phase-shifting signal into an active sparse bundle adjustment achieves highly accurate scene reconstruction with a mean reprojection error way below the camera's resolution. The combination of the sparse features with the dense phase signal is illustrated in Figure 2.13.

A multi-view consistency validation rejects unmatching areas or corrupted patterns, which allows omitting the phase unwrapping entirely and instead solely relying on high-frequency patterns. This drastically reduces the capturing time. The final depth estimation is formulated as an objective function optimizing across all reliable signals simultaneously. Figure 2.14 shows a reconstruction where the color corresponds to the number of used respectively rejected views.

The depth estimation is entirely performed on the GPU, which is particularly efficient because each point can be optimized individually. Therefore, a complete reconstruction takes only a few seconds, roughly

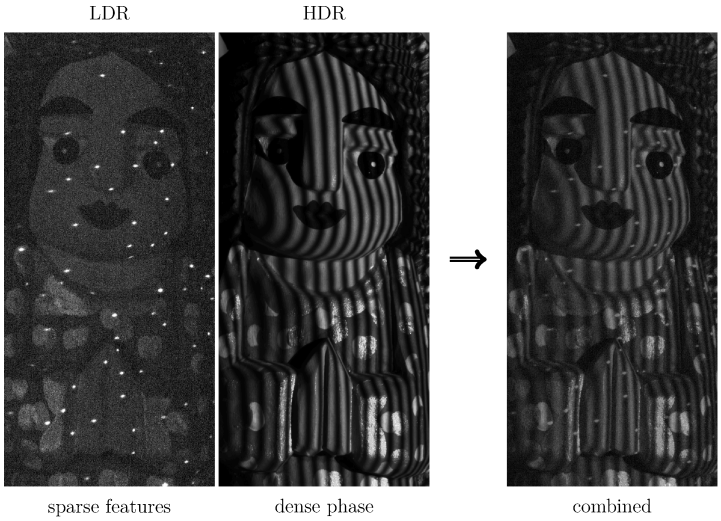


Figure 2.13: **Combined SBA:** Improving the location estimate of the sparse features from the bundle adjustment with a dense phase signal leads to much improved multi-view pose reconstruction. *Images from [1] (Chapter A). Reproduced with permission from Springer Nature.*

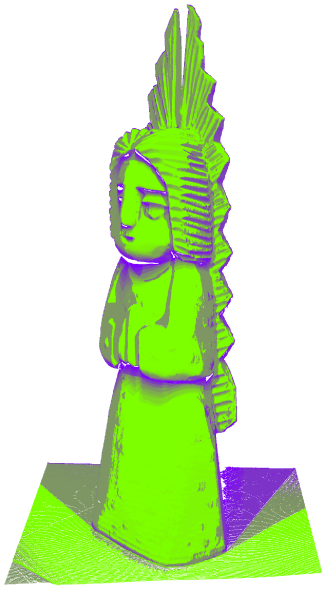


Figure 2.14: **Multi-View Consistency:** This reconstruction is colored depending on the number of reliable views, from green (all) to purple (few). *Images from [1] (Chapter A). Reproduced with permission from Springer Nature.*

2-3 seconds on modern GPUs. The capturing time highly depends on the scene.

In the experimental hardware setup, the projector's spatial resolution on the scene was roughly 500 µm, and 80 µm for the camera. For

the depth estimation, a *Root Mean Squared Error (RMSE)* of 13.9 µm was measured on an evaluation target when using a confidence score. Besides the high-frequency fringe patterns, proper photometric calibration of the sensors and optimal weighted HDR imagining are cornerstones for achieving such high precision and accuracy.

The achievable point cloud density is arbitrarily choosable since the optimization is only based on individual rays through the signal space in 3D. However, the effective resolution is still bounded by the physical properties, calibration, and reconstruction quality.

While the high accuracy and precision achieved are desired for the intended purpose of light-transport estimations, using point clouds with millions of points directly as the basis for learning-based methods was out of reach at this time. In particular, deep learning methods on 3D data, in general, were just emerging [87].

Flex-Convolution: Million-scale point-cloud learning beyond grid-worlds

Fabian Groh, Patrick Wieschollek, and Hendrik P.A. Lensch
Asian Conference on Computer Vision (ACCV), 2018 [2]
(Chapter B)

This framework proposes an extension to image-based CNN architectures to work directly on featureful *RGB* point cloud data. There are three major parts to this approach:

1. *Flex-convolution*: A generalization of the convolution layer for point clouds.
2. *Flex-max-pooling*: A max-pooling layer to construct a hierarchy of point clouds.
3. *IDISS*: An *inverse density importance sub-sampling* scheme to create fast and efficient point cloud hierarchies.

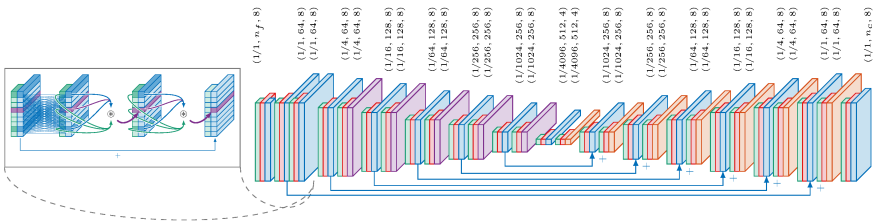


Figure 2.15: *SegNet* inspired **flex-convolution network** for semantic segmentation with skip connections. *Image from [2] (Chapter B). Reproduced with permission from Springer Nature.*

Instead of trainable weights for each cell in an image convolution, *flex-convolution* proposes to learn the parameters of a continuously

defined function that uses the relative position of neighboring points as independent variables to generate the weight for the convolution. The simplicity of the approach is shown by choosing the plain biased dot product as the weight function.

With *flex-convolution*, the well-tested tools and tricks available for CNNs, which made them very successful on structured data, are now also available for point clouds. For example, this is demonstrated by using *SegNet* [88] as the base architecture for the semantic segmentation on *RGB* point clouds. Hence, an excellent-performing network can be simply adapted from the image domain to point cloud data. Figure 2.15 shows the used network architecture for semantic segmentation in a U-net fashion with residual connections.

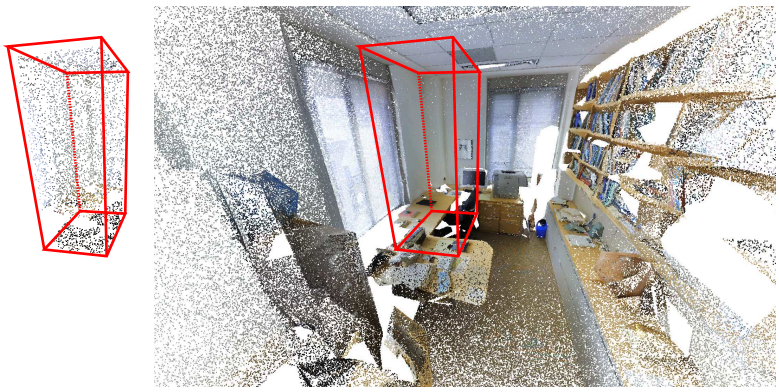


Figure 2.16: **Perceptive-field:** Previous methods (left) are downsampling a predefined area to 4k points. At the same time, the *flex-convolution* network (right) can handle entire rooms with up to 18 million points in a single inference step on a V100 GPU. Potentially having a much higher perceptive field. *Images from [2] (Chapter B). Reproduced with permission from Springer Nature.*

In particular, a fully convolutional architecture significantly benefits real-world challenges. Instead of working on predefining sliding windows with fewer points, the presented method can directly use the full-resolution point clouds, which increases the perceptive field tremendously. Figure 2.16 compares the input of previous methods for semantic segmentation on point clouds to what is used by the proposed *flex-convolution* network.

Everything is implemented as *Tensorflow* layers in *CUDA*, providing forward and backward paths. With this runtime and memory efficiency, the proposed approach outshines other specific point cloud networks not only in terms of quality but also in terms of speed. The graph in Figure 2.17 showcases the correlation between point cloud size and inference speed. While previous methods can barely handle one million points, *flex-convolution* network excels on large-scale point clouds.

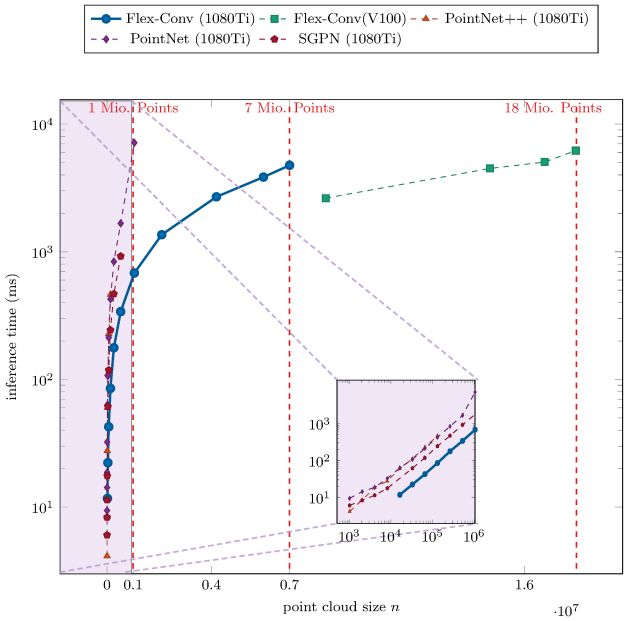


Figure 2.17: **Inference:** Comparison of *flex-convolution* with other point cloud deep learning methods for an inference step. Note that *flex-convolution* can handle up to 7 million points on the same hardware, whereas the others struggle with a million. *Image from [2] (Chapter B). Reproduced with permission from Springer Nature.*

The quality is evaluated on a large-scale, real-world 3D reconstructed office space [89]. Figure 2.18 shows some of the dataset's qualitative semantic segmentation results.

This first-of-its-kind fully convolutional network on million-scale point cloud data exhibited state-of-the-art performance. However, while being very effective, the linear kernel used in the convolution lacks some expressiveness. *KPConv* [90] addresses this by proposing kernel point convolutions, which lead to a more extensive function space for the learnable weights. All these convolution-based techniques rely to some degree on the surface being sampled in relation to the underlying area or pre-processed accordingly. Overly skewed distributions in a local neighborhood would contradict the convolution theory. Similar to the *IDISS* subsampling from Chapter B, the inverse density information could be used as an additional weight to mitigate the problem without neglecting any input point.

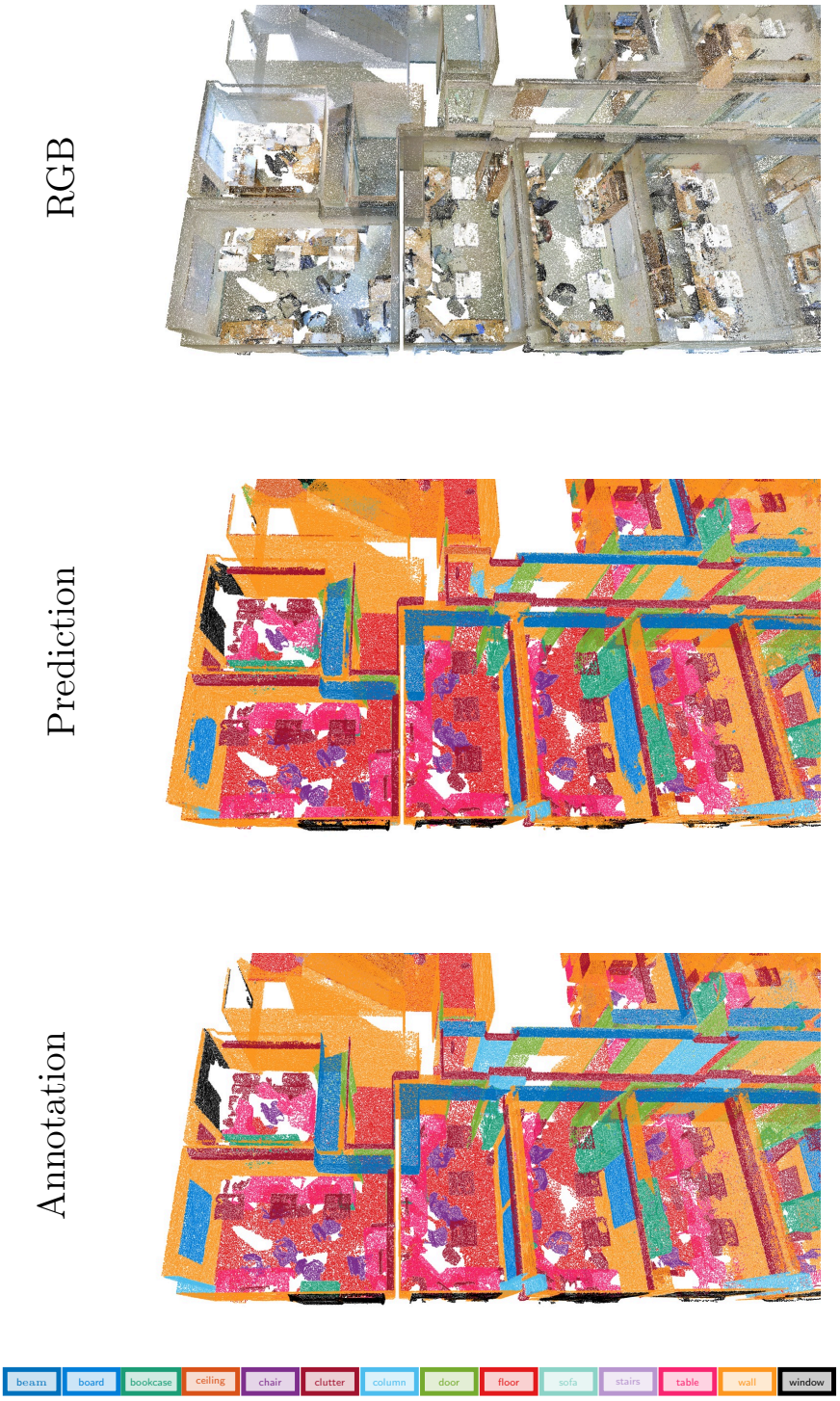


Figure 2.18: **Semantic segmentation** results of a large 3D scanned office space. The *flex-convolution*-based network directly gets the *RGB* point cloud (*top*) as input. It predicts a label per point (*middle*) without seeing the ground-truth annotations (*bottom*) in its training set. More results and quantitative evaluation can be found in Chapter B.

2.3 FEATURES SPACES

As discussed earlier, features can encode information about image points, including their spatial neighborhood. Examples are intensity values, hand-crafted feature descriptors like *SIFT*, or the output of neural network layers. However, they can also represent more global information from images like objects or locations.

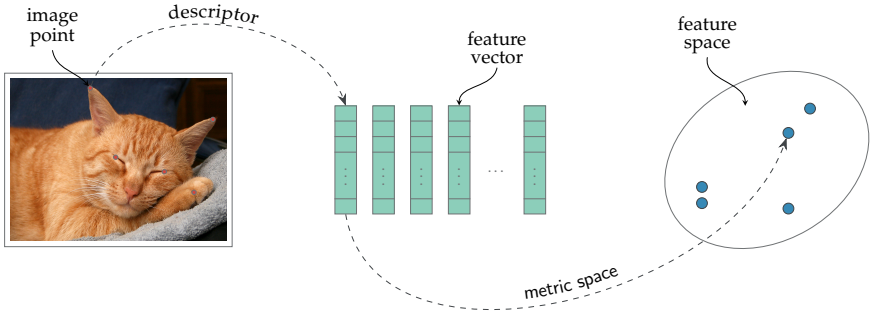


Figure 2.19: **Feature Space**: Descriptors transform information of measured entities, like the local property of image points, into feature vectors. Mapping the features into a metric space makes them comparable. *Cat image [57]*

Descriptors, in one way or another, derive features from measurements and transform the input into high-dimensional vectors. Combined with a distance metric, a feature space emerges that expresses similarities for the desired properties. Figure 2.19 illustrates this relationship. A feature space is an unstructured representation aggregating data from different inputs, usually from structured images.

2.3.1 *Local Image Features*

As mentioned in Section 2.2.1 about camera pose estimation, determining high-quality correspondences between images is essential in computer vision. It is also referred to as the *correspondence problem* [91, 92].

IMPORTANCE OF POINT CORRESPONDENCES Three-dimensional reconstruction techniques like *SfM* are considered offline tasks. Correspondences from different views and sensors are computed, compared, and matched with a global context as a post-processing step, *i.e.*, everything is accessible. At the other end of the spectrum are online approaches. Typically, their task is to estimate motion from consecutive frames on the fly. In robotics, assessing the egomotion of a robot is a significant effort. *Visual Odometry (VO)* and *SLAM* are helping robots to self-localize themselves in the world. Naturally, only views from

the past can be used in an online setting. Compared to the general case of unknown viewpoints, many assumptions help to predict the pose from one frame to the next, *e.g.*, very small baselines or current motion parameters like velocity and acceleration. Next to visual sensors, inertial sensors are used widely to constrain the problem even further. However, the main difference in online approaches compared to offline methods is limiting the time horizon for matching features, *i.e.*, only a few frames from the past are considered.

Nevertheless, all these techniques significantly depend on point correspondences that translate one view into another. Online approaches can allow reliance on intensity values to track points in consecutive frames at a high cadence. For example, the *Visual-Inertial Odometry (VIO)* methods *VINS-MONO* [93] and *BASALT* [94] are using sparse optical flow based on *KLT* [95]. However, even there, it can get tricky to track points on surfaces without enough local structure due to the aperture problem, *i.e.*, ambiguities because of multiple plausible solutions caused by a limited spatial receptive field.

Furthermore, the data association is very short, usually only a few frames. This often leads to estimation drifts. To mitigate this issue, the *SLAM* method *ORB-SLAM3* [66] relies on feature descriptors for mid- and long-term matching. This allows for robust and accurate localization and mapping.

REIDENTIFICATION In the general *SfM* (offline) case, images can be captured under vastly different lighting conditions, rotations, projective deformations, distances, and from various cameras. Hence, it is required to have recognizable and recoverable points from different views.

The major goal of local image feature descriptors is to encode the local structure of an image point to make it identifiable under various conditions. Therefore, the neighborhood around a pixel is analyzed. This area is called a patch and defines the receptive field of a descriptor. Achieving comprehensive expressiveness requires a high-dimensional feature vector to embed all the necessary information. A well-known representative is the *Scale-Invariant Feature Transform (SIFT)* descriptor, which uses 128-dimensional vectors.

The problem to be solved is twofold. Ideally, the same points should have the same values, but different points should have different values. Thus, a second essential characteristic is that a distance metric is defined on the vectors, which approximates similarity. Feature descriptors assemble a high-dimensional vector space. Similar to creating point clouds from multiple views (see Section 2.2.1), local structured input aggregates information into a common unstructured domain. In this case, the underlying information is not a three-dimensional surface but the similarities of descriptors. Here, it is the appearance of image points within their local neighborhood. In the high-dimensional

space, similar points will form clusters that should be distinguishable from points with a different appearance.

MATCHING In order to find a point correspondence, the task is to search for a matching vector in a different view. This is known as the *feature matching* stage. Unfortunately, even when using good feature descriptors, it is unlikely that the same point will have exactly the same values in different views. The assumption, however, is that they are still very similar in the feature domain. Hence, they should have a low value on the feature distance metric δ. Let f_{x_i} be a feature vector of a point x_i from the image I, then finding the correspondence point c_{I',x_i} in the image I' for the point x_i can be expressed as:

$$c_{I',x_i} = \underset{x_j \in I'}{\arg\min}\ \delta(f_{x_i}, f_{x_j})\,. \tag{2.16}$$

Computing the full set of correspondences $C_{I \to I'}$ for all points x_i from image I_i in the target image I_j is defined by:

$$C_{I \to I'} = \bigcup_{x_i \in I} \{c_{I',x_i}\}\,. \tag{2.17}$$

An obvious problem is the immense computational power necessary to compute this in a multi-view setting with potentially hundreds of images. For a single pair of 1080p images, this would already result in over 220 million vector distance evaluations in the naive approach, which is computationally not feasible.

SALIENCY On top of that, when creating a feature descriptor for every point, many are not expressive enough. Numerous places in an image either do not have sufficient local structure or are very common, like homogeneous areas. If the receptive field is too small to catch meaningful and distinguishable structures, features can not be exactly relocated.

In point-correspondence tasks, the general interest is in fewer but higher-quality features. Lesser adequate correspondences are more harmful than beneficial. Hence, a keypoint selection is inevitable, relying only on points with a well-matchable structure and saliency in the feature space. Finding these points respectively areas of interest is the task of *feature detectors*.

The usual order for finding good correspondence point proposals between views is as follows:

1. **Detection**: Select points of interest.
2. **Description**: Encode the local structure of the selected point in a high-dimensional vector.
3. **Matching**: Search for similar points from other views in the feature vector space.

With the detection step in mind, the computation and matching of descriptors are only performed on the selected keypoints; effectively downsizing the problem from Equation 2.17. Further, some feature detectors have sub-pixel accuracy. Hence, finding the reduced set of correspondences $\hat{C}$ for the selected keypoints $\hat{P}$ changes to the image plane domain:

$$\hat{C}_{P \to P'} = \bigcup_{x_i \in \hat{P}} \{\hat{c}_{P',x_i}\} \qquad \hat{c}_{P',x_i} = \underset{x_j \in \hat{P}'}{\arg\min}\ \delta(f_{x_i}, f_{x_j})\,. \tag{2.18}$$

FEATURE DESCRIPTOR FRAMEWORKS Multiple local image feature *detector-descriptor-matcher* ensembles are available since finding correspondences is a cornerstone for various computer vision algorithms, like *SIFT*, *SURF*, *ORB*, *AKAZE* [75, 96]. They differ mainly in their intended purpose. In some scenarios, invariance on scale and rotation is required, while others desire real-time capabilities over the highest quality.

More specific details about local image features and comparisons of the various techniques are provided by the extensive works of Tareen and Saleem [75] and Ma et al. [96].

Next to the hand-crafted and traditional machine-learning techniques, also deep learning descriptor methods are emerging, *e.g.*, SuperPoints [76], *LF-Net* [97], and *RF-Net* [98]. They attempt more domain-specific and data-driven solutions. Very recently, the deep learning framework *LightGlue* also made substantial progress on matching local image features [33].

2.3.2 *Global Features*

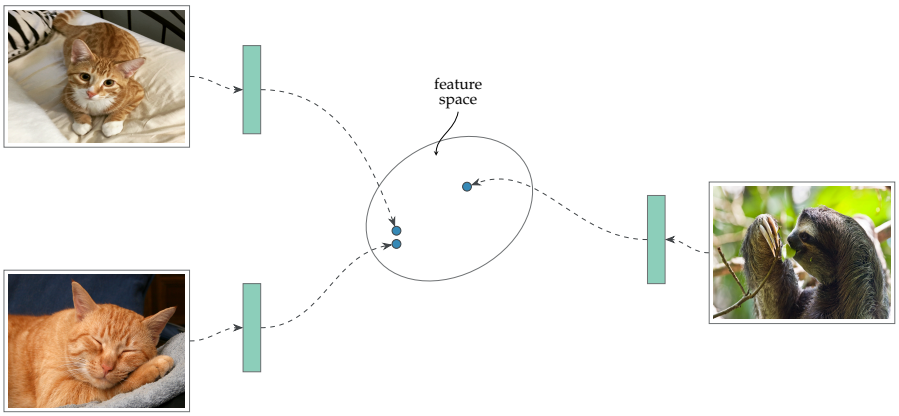


Figure 2.20: **Global feature spaces** involve the vector embeddings of whole entities. Typical examples are using global image features for object detection or place recognition tasks. *Cat image (top) [99], cat image (bottom) [57], and sloth image [100].*

Unlike local image features, which analyze a small receptive field around a point, global image features aim to embed the context of an

entire image into a vector. See Figure 2.20. Local feature descriptors typically have a fixed dimension of neighboring pixels to consider, whereas global features are intended to work for a wide variety of input shapes. Although global descriptors are usually higher dimensional than local descriptors, in relation to the covered receptive field, local feature vectors are more spacious and theoretically can pack more detailed information. For example, *SIFT* uses 128-dimensional vectors for a 16x16 image patch, while *GIST* typically varies between 512 and 960 dimensions for a whole image. The *neural codes* used in Chapter D even go down to 96 to represent an image. As usual, for feature descriptors, the task determines the manifestation of a feature. Finding similar content may require less power than identifying objects or places.

SCENE COLLECTIONS The previous sections always assumed that the views of a particular scene are known. This might be true for consecutive frames from a single camera or a multi-sensor suite that captures data simultaneously, like in autonomous driving or robotics.

Nonetheless, it is also possible to connect different views that are not recorded jointly but show the same object and location. An example is the reconstruction of Rome just from publicly available photos on the Internet [27]. Even when these images are labeled as Rome, millions of images are still left when filtered. The goal is to find images with overlapping views of the same objects at identical locations. Thus, images with similar content are of interest.

SIMILARITIES Global features embed complete instances, *e.g.*, images for tasks like object or place recognition. Therefore, their visual context must be larger relative to what is needed for a local point. These features potentially capture multiple different objects or other visual cues appearing in a scene.

Further, the goal is often much coarser than finding the exact same instance. Some challenges even strive for more diversity in their answers, *e.g.*, in image classification, where the type of an object is more important than the color, or when the interest is in the linguistic or semantic similarity of a word [101]. The descriptors and the selected matching distance metric can incorporate such desired conditions in the underlying vector space.

GLOBAL FEATURES VIA STATISTICS A hand-crafted traditional descriptor for scene recognition is *GIST* [102]. Its global image features embed spectral information averaged in locally divided patches.

Another common approach is using statistics of an image's local feature descriptors. The *Bag of Visual Words (BoVW)* approach builds a frequency histogram from the vectors [103, 104]. For *Fisher Vector (FV)* encodings, a *Gaussian Mixture Model (GMM)* is estimated from

the vectors [105–107]. *Vector of Linearly Aggregated Descriptors (VLAD)* typically uses *KMeans* to encode the feature distribution without the higher-order calculations used in *FV* [108, 109]. Fitting data to statistical models and using their parameters makes them less dependent on the input resolution.

DEEP LEARNING IS TAKING OVER More deep learning techniques have focused on global image features in recent years, working directly on the images without hand-crafted descriptors in the loop.

While image classification and retrieval are separate tasks, they are connected through image similarity. This claim is supported by the work of Krizhevsky, Sutskever, and Hinton [110]. Even though their network is trained for image classification on the *ImageNet* dataset, comparing the feature vectors from the layers before the classification stage demonstrates a good use as image retrieval descriptors by finding the same or very similar objects. The conjecture is that the network is embedding the image's higher-order information in these layers, and it can serve as a global image descriptor for image similarity.

Babenko et al. enhance this work further by retraining on the specific task of image retrieval [71]. They introduced the term *neural codes* for those kinds of deep descriptors. In their validation, the second to last layer's output performed best as a feature descriptor, and it was possible to reduce the dimensionality drastically, with only a slight drop in performance. *Neural codes* enhance this work by using the pretrained model but retraining it on the specific task of image retrieval [71]. The *Deep1B* dataset featured in Chapter D represents 1 billion *neural codes* from a *GoogLeNet* network [38].

2.3.3 *Efficient Processing and Learning on Feature Spaces*

This thesis focuses on the aspects of unstructured high-dimensional feature spaces and the matching of descriptors.

Backpropagation training for fisher vectors within neural networks

Patrick Wieschollek*, Fabian Groh*, and Hendrik P.A. Lensch
arXiv preprint arXiv:1702.02549, 2017 [3]
(Chapter C)

* *shared first authorship.*

This project presents an end-to-end machine-learning technique that uses dense local image feature vectors (*SIFT*) as inputs to create a global image embedding based on *Fisher Vectors (FV)* with the intention of image classification.

The basic concept of *Fisher Vectors* is to encode data by its position and distribution, *i.e.*, mean and covariances from a fitted *Gaussian-*

Mixture-Model (GMM). The concept is illustrated in Figure 2.21. Rather than using the *SIFT* features directly, the input is first transformed through a fully connected *neural network (NN)*.

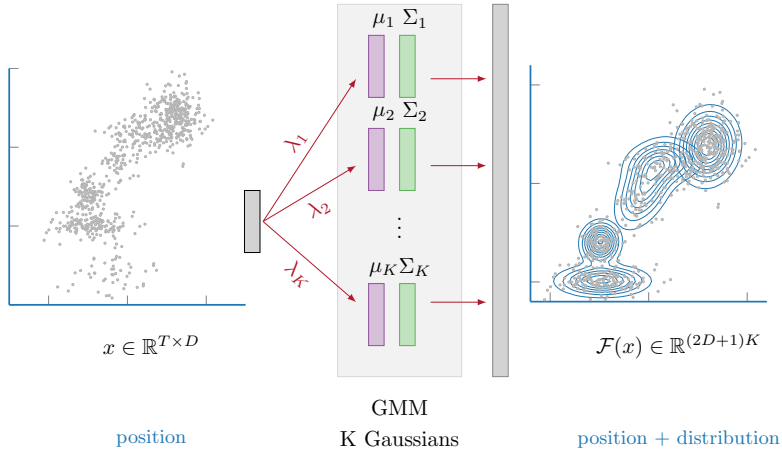


Figure 2.21: The **Fisher Vector (FV)** encodes an image's dense local features distribution via a *Gaussian-Mixture-Model (GMM)* to represent a global image feature. *Image from [3] (Chapter C).*

Traditionally, deep *neural networks* and *Fisher-Vectors* are combined as separate steps, where the feature learning stage is independently conducted before the encoding. Instead, this approach proposes to learn all network layers end-to-end. Hence, the information from the loss is shared through the encoding to the fully connected layers at the beginning. Figure 2.22 sketches the two approaches side-by-side. All components must be derivable for such an end-to-end approach

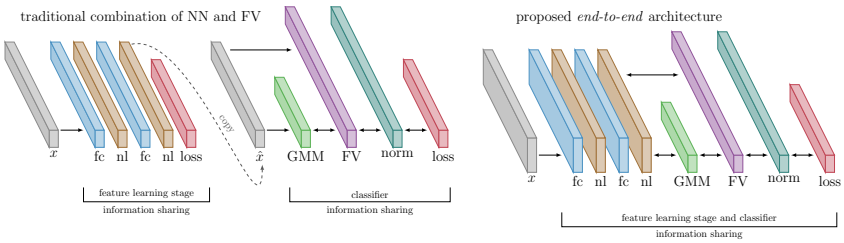


Figure 2.22: **End-to-end:** Instead of handling the *neural network (NN)* and the *Fisher Vector (FV)* encoding separately, the proposed method uses an end-to-end architecture for complete backpropagation. *Image from [3] (Chapter C).*

to allow full backpropagation. Thus, the work provides all necessary derivatives for the *GMM* and *FV* parts.

Further, it is shown that a specifically tailored GPU implementation is necessary to perform backpropagation training with *FVs* in a feasible amount of time. Compared to an already vectorized CPU/Matlab implementation, the GPU version had a speed-up factor of over five thousand.

In the *PASCAL VOC 2007* classification challenge, the approach outperforms the previous methods without the *FV* encoding and the one with the traditionally separated *FV* method. It increases the *average precision (AP)* from 52.1 respectively 52.8 to 54.7 without decreasing the performance on a single class.

Around the same time as this project, *NetVLAD* [111] was presented, which had a similar approach. They use a *CNN* as the transforming neural network, but instead of *Fisher Vectors*, their work utilizes *Vector of Linearly Aggregated Descriptors (VLAD)* for embedding the distribution.

MATCHING Section 2.4.1 will describe a GPU-based fast, large-scale feature matching framework for multiple points in parallel (Chapter D). It effectively matches high-dimensional local and global features up to 960 dimensions at million- and billion-scale datasets.

2.4 APPROXIMATE NEAREST NEIGHBORS

The previous Section 2.3 has focused on building feature spaces by designing descriptors with a distance function to ensemble a desired property, like point or image similarity. A primary purpose is to search for suitable candidates, like in Equation 2.18 for correspondence matching. For the sake of simplicity, let's assume the feature space is the d-dimensional Euclidean space $(\mathbb{R}^d, \delta)$ and the dataset $\mathcal{X} \subset \mathbb{R}^d$ represents all the possible candidates $\mathcal{X} = \{x_0, \dots, x_n\}$. Given a query point $q \in \mathbb{R}^d$, finding the point $p \in \mathcal{X}$ with the smallest distance to q is called *nearest neighbor (NN)* search:

$$p = \arg\min_{x \in \mathcal{X}} \delta(q, x)\,. \tag{2.19}$$

Even the best descriptors are not perfectly accurate. The correct correspondence to a query point is often not the nearest neighbor but only in the proximity. This is especially true when there are many similar occurrences to the one it is searched for. Hence, it is often required to return a list of the k best nearest neighbors (*kNN*), effectively performing a ranged proximity search. Typically, the k candidates are then further validated, *e.g.*, geometrically or photometrically.

RELAXING COMPLEXITY A problem already revealed by Equation 2.17 is the complexity of searching for every point in every other view. The same applies to image similarity searches with millions or billions of entries. In the naive approach, every query would go through every descriptor in the dataset to find the best match. This quadratic complexity is infeasible for large datasets.

There are ways of further restricting the scope if applicable, *e.g.*, sliding windows in video sequences, odometry information in robotics, meta description tags, or *global navigation satellite system (GNSS)* signals.

In the general case of no additional external knowledge, a common approach is relaxing the problem to *approximate nearest neighbor (ANN)* search respectively, *i.e.*, no longer guaranteeing the exact nearest neighbors. This concept uses acceleration structures that accept some loss in retrieval accuracy. Many *ANN* methods still achieve high precision values but significantly speed up the search speed.

ANN search is specifically advantageous for problems with multiple reasonable solutions or when only a subset of queries requires an exact solution. For example, the former is the case in image similarity search, where the interest in finding images with similar objects or locations outweighs the need to retrieve the most interchangeable image. The latter is the subject in estimating camera poses from point correspondences. While the points must match accurately, only a few of the potentially hundreds of valid keypoints are sufficient to estimate a 3D rigid body transformation with six degrees of freedom. On top of that, as already mentioned, the descriptor space is not exact anyway.

The basic techniques of *ANN* search evolve about analyzing the points $\mathcal{X}$ as a pre-processing step. A usual suspect is dimensionality reduction via *principle component analysis (PCA)* or *singular value decomposition (SVD)* [112]. Removing insignificant dimensions decreases the time needed for calculating the distance functions but doesn't lower the number of points to search through.

Another group of methods is derived from the idea of mapping and grouping the points in $\mathcal{X}$ to *buckets* (*Locality-sensitive hashing (LHS)* [113]), *vectors* (*Vector-quantization (VQ) [114]*), or *cells* (*Product-quantization (PQ)* [115, 116]). The retrieval idea is to map the query similarly and compare it with the associated points from the respective entry.

PROXIMITY GRAPHS In the last few years, proximity neighborhood graphs have gained momentum, outperforming other methods [117, 118]. The basic idea is to construct a nearest neighbor graph from the points $\mathcal{X}$ with edges from every point x_i to its local neighborhood $\mathcal{N}_{x_i} \subset \mathcal{X}$ and use this structure to guide a query q. The local neighborhood around the point x_i with its k nearest neighbors is defined by:

$$\mathcal{N}_{x_i}^{k} = \bigcup_{i=1}^{k} \left\{ \underset{x_j \in \mathcal{X} \setminus \left\{ \mathcal{N}_{x_i}^{i-1} \cup \{x_i\} \right\}}{\arg\min} \delta(x_i, x_j) \right\} . \tag{2.20}$$

A naive approach for searching with proximity graphs is a greedy NN-descent traversal [118]. Starting from a point x_t, the algorithm evaluates its neighborhood $\mathcal{N}_{x_t}$ for a closer point to the query q:

$$x_{t+1} = \underset{x_i \in N_{x_t}}{\arg\min}\ \delta(x_i, q) . \tag{2.21}$$

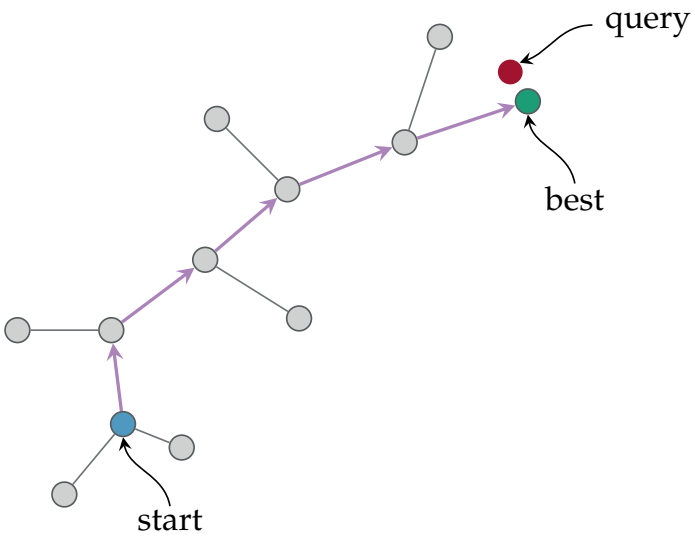


Figure 2.23: **Greedy NN-descent traversal**: The current point validates its neighbors for a shorter distance to the query. If a short distance is found, the procedure will be repeated from the closer point. This might lead to the correct solution in a dense and well-distributed environment with a large enough k.

If the new point x_{t+1} is not closer to the target as the previous point, the greedy method stops. Otherwise, it hops to the closer point and repeats the evaluation. Figure 2.23 illustrates such a traversal.

The idea is to have k small enough because k distances have to be computed on any hop from node to node. The number of hops depends on the starting point and k itself. Since this algorithm is expected only to traverse a tiny part of the space, it should need much fewer distance computations to reach the goal [118, 119]. In degenerated scenarios, *e.g.*, all points in a line, this assumption might not hold.

Of course, such a greedy approach only works in well-structured settings. In every neighborhood, the closest point has to be on a path to the query. Even in the 2D case, it's easy to have constructs that are no longer valid in this regard. In a high-dimensional case, this is much worse. On top of that, cluster islands and outliers are ultimately missing bridging connections.

The nearest neighbor search with proximity graphs is improved from two sides. The first and more straightforward part is to make the search algorithm smarter by using ideas from local pathfinder methods, typically adding backtracking. The second side is to change the construction of the graph to allow better search performance, which is the main focus of the literature in this area [118, 120, 121].

While query speed and performance are essential, the construction speed of proximity graphs is also of interest. Some methods measure their construction time in hours or days [122, 123]. These timespans are

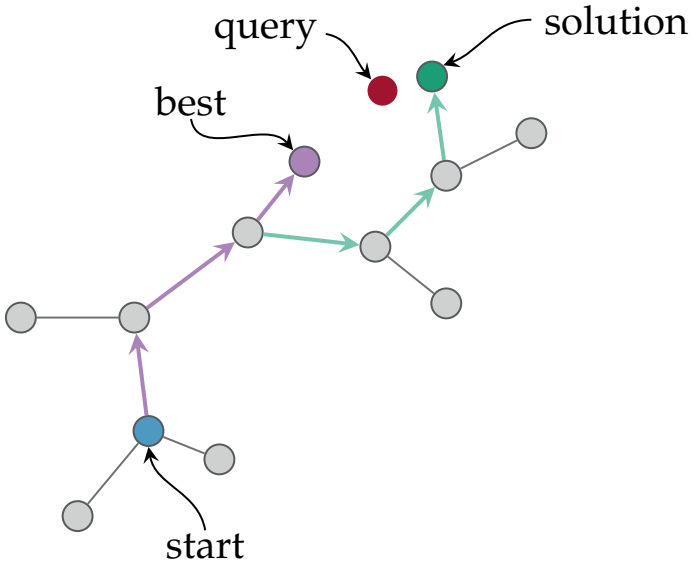


Figure 2.24: **Problems of greedy NN-descent**: In many cases, the greedy algorithm will get stuck in local minima. In higher dimensions, the problems are much more severe. Backtracking can help to find the correct solution when there is a path in the graph.

not bearable when the goal is to examine freshly generated data, *e.g.*, *neural codes* from an updated network or captured data from robotics.

2.4.1 *Efficient Processing and Learning on Proximity Graphs*

This thesis is dedicated to high-performance search in and constructions of proximity graphs for high-dimensional large-scale feature spaces on GPUs.

GGNN: Graph-based GPU nearest neighbor search
Fabian Groh, Lukas Ruppert, Patrick Wieschollek, and Hendrik P.A. Lensch
IEEE Transactions on Big Data, 2022 [4]
(Chapter D)

This paper describes a superfast and accurate batch-wise query and construction scheme for proximity graphs on GPUs. Its effectiveness is shown on million- and billion-scale high-dimensional local- and global feature sets, like on *SIFT1B*, *GIST*, and *Deep1B*.

The framework's core is the proposed massively parallel approximate k-nearest neighbor search algorithm for high-dimensional vectors on GPUs. For example, to retrieve the best neighbors in a dataset of 1 million *SIFT* features, on average, the search takes only about 1.5 μs per query to reach 99% accuracy on an NVIDIA V100 GPU.

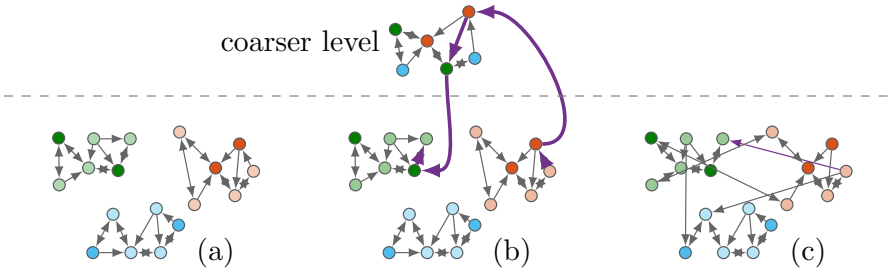


Figure 2.25: **Merging sub-graphs**: For each sub-graph (a), a few points are selected to form a coarser graph above. Each point in the sub-graphs queries through the higher layer to find good neighbors in the other ones (b). Hence, the merged sub-graphs build a new proximity graph (c).

Besides the search, the query algorithm is also the foundation for a parallel bottom-up construction scheme to create proximity graphs from unseen data in seconds for high-dimensional million-scale datasets.

The primary idea is to merge multiple sub-graphs by searching for neighbors in the other sub-graphs. Therefore, a few points per sub-graph are selected to form a higher layer that bridges the bottom layers through a hierarchy. This process is showcased in Figure 2.25. Recursively merging the sub-graphs builds a full proximity graph.

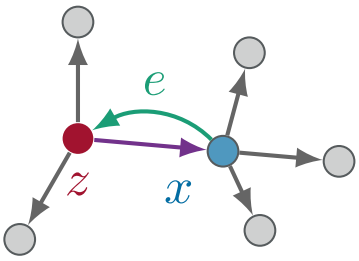


Figure 2.26: A **symmetric-link** e is added if there is no path back from x to z, even if x is a close neighbor of z. This approximates an undirected graph. *Figure from [4] (Chapter D)*. ©2022 IEEE.

The graph is further diversified by adding symmetric links for better search results. These links aim to approximate an undirected graph structure. A reverse query over all directed edges finds the required symmetric links to be added. See Figure 2.26 for an illustration.

A comparison with other proximity graph-based methods for searching in the 1 million *SIFT* vector database *SIFT1M* is shown in Figure 2.27. *GGNN* is outperforming the other approaches in terms of speed while still reaching very high recall rates. For more comparison of different datasets and more details on the evaluation, please refer to Chapter D.

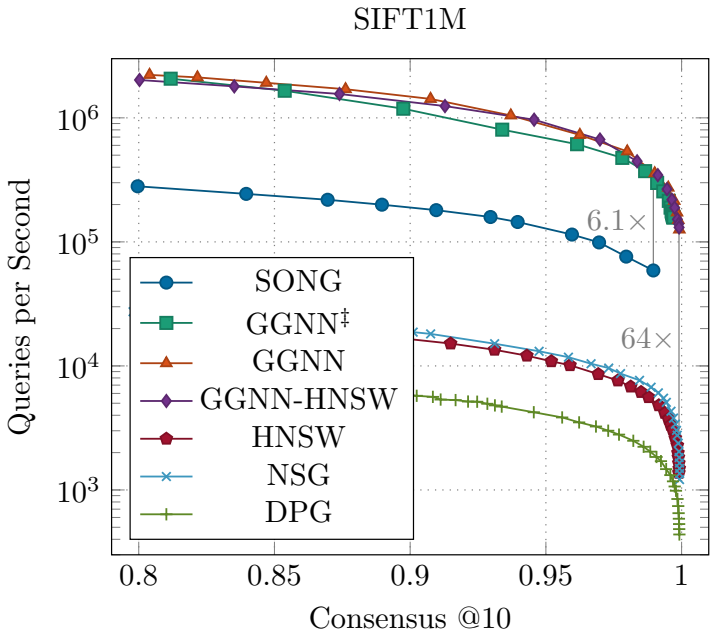


Figure 2.27: A **proximity graph performance comparison** between the proposed method *GGNN* and other methods on the *SIFT1M* dataset, shows that *GGNN* is significantly faster even for very high *Consensus @10* rates. Chapter D contains more details and many more evaluations. *Figure from [4] (Chapter D).* ©2022 IEEE.

Retrieving good neighbors is also essential and critical to many fields outside computer vision [124] Information embedding systems try to impound larger and more extensive knowledge.

Hence, large-scale proximity graphs have a great perspective as they have been shown to retrieve excellent neighbors reliably and effectively. They are adjustable to different needs, trading performance for quality. However, the reliance on several tunable parameters is also one of their weaknesses, as changes to the graph structure are non-linear concerning the retrieval quality for different datasets.

The other downside is the vast memory consumption these systems have. While the proximity graph structure itself is comparably small, performing all distance measurements on all the full vectors requires the whole dataset to be present. This results in out-of-memory strategies to cope with memory limitations in typical systems. Chapter D shows a sharding technique where the data is split to form multiple proximity graphs that can be queried separately and in parallel on multiple GPUs.

Follow-up work has taken on this task with more hardware-centric approaches for graph construction [125] and with a distributed FPGA architecture [126].

3

FUTURE WORK

Efficient processing and learning on unstructured data representation is an enormous field. While the presented projects cover various areas, many more insights can be discovered. This Chapter offers recommendations for future research by discussing high-level findings and limitations of the work in this thesis.

MULTI-MODALITY POINT CLOUDS All projects primarily focus on single data modalities, *e.g.* aggregated information inferred from images. In many real-world tasks, a suite of different sensors is present. Hence, a multi-modality approach is required to fuse the sensors into a common unstructured data representation. However, the contribution from different sensors is probably not equal. Therefore, information about the sensor and its characteristics, including sampling patterns and density, should be baked into the feature vector per point. Learning approaches on unstructured data, like in Chapter B, can utilize these cues to infer predictions and combine them meaningfully.

HIGHER-DIMENSIONAL POINT CLOUDS Typically, point clouds are assumed to be three-dimensional. Nonetheless, there are also occurrences with more dimensions. For example, the experimental hardware in Chapter A was designed to provide point clouds combining geometry and *BRDF* observation. Every captured point on the surface also carries a sample of the light transport from an incident to an outgoing angle with a strong correlation regarding the underlying material. Learning such a relationship would allow photorealistic relightings of real-world objects with a vastly reduced number of captures.

LARGE-SCALE 3D GENERATIVE NETWORKS Recently, large-scale generative networks that produce plausible images from text inputs have been a very active field of research [127]. A natural extension is heading to the third dimension. The first attempts to predict single objects in 3D still use the 2D image path [128, 129]. Generating plausible large-scale environments, like landscapes or cities, directly by describing a scene would significantly impact the movie or games sector. Structured representations with a regular grid are probably too limiting in this regard. Predicting point clouds or meshes directly would be very practical to use.

LEARNING OPTIMAL SEARCH GRAPH DIVERSIFICATION AND DESCRIPTORS The proximity graph structure, with edges between close points, leads to outstanding results, as shown in Chapter D. Quality is primarily controlled via the number of edges between the points. Increasing the number also increases the cost of visiting a point since a hop requires computing all the distances to the neighbors. Graph diversification tries to find a good set of edges for a given graph, but it is a complex and time-consuming problem. A learned approach that generates a search graph directly from a given set of points by optimizing recall quality might be a valuable field for further research. Further, such a system can also optimize for the best possible input by learning a feature descriptor end-to-end with the retrieval quality, similar to Chapter C.

4

CONCLUSION

Unstructured data representations like point clouds or feature spaces embed knowledge that is not easily accessible from single images. However, getting off the grid comes with a price; losing the implicit representation complicates many routines and renders algorithms that are core to computer vision unusable. In particular, machine and deep learning rely heavily on structured input for efficient computations, ultimately enabling large and deep models.

This thesis discusses the importance of unstructured data representations in computer vision and shows the connection to structured data. Further, it demonstrates various methods and strategies to effectively leverage the rich information embedded in unstructured representation for diverse fundamental computer vision problems. From high-precision 3D scanning over deep learning on point clouds and feature spaces to nearest neighbor search with proximity graphs. The presented projects have shown that efficient processing and learning on unstructured data is possible when designing algorithms in a massively parallelizable fashion from the ground up. High efficiency, scalability, and robustness are not achieved in a post-processing step; They are all essential in inventing a solution. Thus, the proposed approaches follow a schema of simplicity scaled up through parallelization on GPUs. Every project has shown its effectiveness on large-scale real-world data to underline these characteristics.

With more and more data being generated and produced, *e.g.*, on edge devices, the need for efficient and fast algorithms is ever-increasing. Simply adding more hardware is not a proper solution in resource-restricted environments. In robotics, particularly autonomous vehicles, where the environment is typically mapped by continuous range scanning and 3D reconstruction techniques, decisions must be made on the spot with very low latency. Efficiently segmenting the external world semantically in the combined space of geometry and color supports fast and sensible reasoning of the surroundings.

High-dimensional nearest neighbors search is a crucial technology essential to many applications, even outside computer vision, *e.g.* recommender systems or DNA sequencing. In the future, it can be expected that even more inferred knowledge will be stored in databases. Hence, a cost-effective and performant search algorithm allows access to such large-scale and high-dimensional datasets. Further, it enables online proximity search on newly obtained data or in learning frameworks.

Generally, unstructured representations require more effort than their structured counterparts for processing and learning. However, they are essential when dealing with high-dimensional or spatially varying data by combining multiple measurements into a common representation. Finding the right balance between unstructured and structured approaches is essential to achieve the best efficiency for a specific task. Using structured data will be bound to a predefined resolution, while unstructured representations will consistently adapt to the data.

LIST OF FIGURES

BIBLIOGRAPHY

[1] Fabian Groh, Benjamin Resch, and Hendrik P.A. Lensch. "Multi-view continuous structured light scanning." In: *German Conference on Pattern Recognition (GCPR)*. Springer. 2017, pp. 377–388.

[2] Fabian Groh, Patrick Wieschollek, and Hendrik P.A. Lensch. "Flex-Convolution: Million-scale point-cloud learning beyond grid-worlds." In: *Asian Conference on Computer Vision (ACCV)*. Springer. 2018, pp. 105–122.

[3] Patrick Wieschollek*, Fabian Groh*, and Hendrik P.A. Lensch. "Backpropagation training for fisher vectors within neural networks." In: *arXiv preprint arXiv:1702.02549* (2017).

[4] Fabian Groh, Lukas Ruppert, Patrick Wieschollek, and Hendrik P.A. Lensch. "GGNN: Graph-based GPU nearest neighbor search." In: *IEEE Transactions on Big Data* 9.1 (2022), pp. 267–279.

[9] Daniel L Schacter, Daniel T Gilbert, and Daniel M Wegner. *Psychology*. Macmillan, 2009.

[10] Stephen P Robbins, Tim Judge, et al. *Essentials of organizational behavior*. Prentice Hall Upper Saddle River, NJ, 2012.

[11] Steven Lehar. "The function of conscious experience: an analogical paradigm of perception and behavior." In: *Consciousness and Cognition* 9.2 (2000).

[12] Steven Lehar. *The World in Your Head: A Gestalt View of the Mechanism of Conscious Experience*. Lawrence Erlbaum, 2003.

[13] Richard L Gregory. "Knowledge in perception and illusion." In: *Philosophical Transactions of the Royal Society of London. Series B: Biological Sciences* 352.1358 (1997), pp. 1121–1127.

[14] D. Bernstein. *Essentials of Psychology*. Cengage Learning, 2018.

[15] Stuart J Russell. *Artificial intelligence a modern approach*. Pearson Education, Inc., 2010.

[16] Taro Makino, Stanisław Jastrzębski, Witold Oleszkiewicz, Celin Chacko, Robin Ehrenpreis, Naziya Samreen, Chloe Chhor, Eric Kim, Jiyon Lee, Kristine Pysarenko, et al. "Differences between human and machine perception in medical diagnosis." In: *Scientific reports* 12.1 (2022), pp. 1–13.

[17] Klaus Dietmayer. "Predicting of Machine Perception for Automated Driving." In: *Autonomous Driving: Technical, Legal and Social Aspects*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2016, pp. 407–424.

[18] Malcolm Tatum. *What Is Machine Perception?* 2023. URL: https://www.easytechjunkie.com/what-is-machine-perception.htm (visited on 01/02/2023).

[19] Michael I Posner, Mary J Nissen, and Raymond M Klein. "Visual dominance: an information-processing account of its origins and significance." In: *Psychological review* 83.2 (1976), p. 157.

[20] Stephen E Palmer. *Vision science: Photons to phenomenology*. MIT press, 1999.

[21] Kristin Koch, Judith McLean, Ronen Segev, Michael A. Freed, Michael J. Berry, Vijay Balasubramanian, and Peter Sterling. "How Much the Eye Tells the Brain." In: *Current Biology* 16.14 (2006), pp. 1428–1434.

[22] Richard Szeliski. *Computer vision: algorithms and applications*. Springer Nature, 2022.

[23] Hendrik PA Lensch, Jan Kautz, Michael Goesele, Wolfgang Heidrich, and Hans-Peter Seidel. "Image-based reconstruction of spatial appearance and geometric detail." In: *ACM Transactions on Graphics (TOG)* 22.2 (2003), pp. 234–257.

[24] Pradeep Sen, Billy Chen, Gaurav Garg, Stephen R Marschner, Mark Horowitz, Marc Levoy, and Hendrik PA Lensch. "Dual photography." In: *ACM SIGGRAPH 2005 Papers*. 2005, pp. 745–755.

[25] Reinhard Klette. *Concise computer vision*. Vol. 233. Springer, 2014.

[26] Linda G Shapiro, George C Stockman, et al. *Computer vision*. Vol. 3. Prentice Hall New Jersey, 2001.

[27] Sameer Agarwal, Yasutaka Furukawa, Noah Snavely, Ian Simon, Brian Curless, Steven M Seitz, and Richard Szeliski. "Building rome in a day." In: *Communications of the ACM* 54.10 (2011), pp. 105–112.

[28] Norman Makoto Su and David J Crandall. "The affective growth of computer vision." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2021, pp. 9291–9300.

[29] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, et al. "Imagenet large scale visual recognition challenge." In: *International journal of computer vision* 115 (2015), pp. 211–252.

[30] IEEE Computer Society. *Welcome to CVPR 2022*. URL: https://cvpr2022.thecvf.com/sites/default/files/CVPR2022-ChairsOpeningMessages.pdf (visited on 06/19/2022).

[31] David G Lowe. "Object recognition from local scale-invariant features." In: *Proceedings of the seventh IEEE international conference on computer vision*. Vol. 2. Ieee. 1999, pp. 1150–1157.

[32] Nasser H Dardas and Nicolas D Georganas. "Real-time hand gesture detection and recognition using bag-of-features and support vector machine techniques." In: *IEEE Transactions on Instrumentation and measurement* 60.11 (2011), pp. 3592–3607.

[33] Philipp Lindenberger, Paul-Edouard Sarlin, and Marc Pollefeys. "LightGlue: Local Feature Matching at Light Speed." In: *arXiv preprint arXiv:2306.13643* (2023).

[34] Daniel A Roberts, Sho Yaida, and Boris Hanin. *The principles of deep learning theory*. Cambridge University Press Cambridge, MA, USA, 2022.

[35] Neil C Thompson, Kristjan Greenewald, Keeheon Lee, and Gabriel F Manso. "The computational limits of deep learning." In: *arXiv preprint arXiv:2007.05558* (2020).

[36] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks." In: *Advances in neural information processing systems* 25 (2012).

[37] Karen Simonyan and Andrew Zisserman. "Very deep convolutional networks for large-scale image recognition." In: *arXiv preprint arXiv:1409.1556* (2014).

[38] 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*. 2015, pp. 1–9.

[39] 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*. 2016, pp. 770–778.

[40] Mingxing Tan and Quoc Le. "Efficientnetv2: Smaller models and faster training." In: *International conference on machine learning*. PMLR. 2021, pp. 10096–10106.

[41] Alexey Dosovitskiy et al. "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale." In: *International Conference on Learning Representations*. 2021.

[42] Ze Liu, Yutong Lin, Yue Cao, Han Hu, Yixuan Wei, Zheng Zhang, Stephen Lin, and Baining Guo. "Swin transformer: Hierarchical vision transformer using shifted windows." In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2021, pp. 10012–10022.

[43] Jiahui Yu, Zirui Wang, Vijay Vasudevan, Legg Yeung, Mojtaba Seyedhosseini, and Yonghui Wu. "Coca: Contrastive captioners are image-text foundation models." In: *arXiv preprint arXiv:2205.01917* (2022).

[44] Zhuang Liu, Hanzi Mao, Chao-Yuan Wu, Christoph Feichtenhofer, Trevor Darrell, and Saining Xie. "A convnet for the 2020s." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 11976–11986.

[45] Wenhai Wang, Jifeng Dai, Zhe Chen, Zhenhang Huang, Zhiqi Li, Xizhou Zhu, Xiaowei Hu, Tong Lu, Lewei Lu, Hongsheng Li, et al. "Internimage: Exploring large-scale vision foundation models with deformable convolutions." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023, pp. 14408–14419.

[46] Chuyi Li, Lulu Li, Hongliang Jiang, Kaiheng Weng, Yifei Geng, Liang Li, Zaidan Ke, Qingyuan Li, Meng Cheng, Weiqiang Nie, et al. "YOLOv6: A single-stage object detection framework for industrial applications." In: *arXiv preprint arXiv:2209.02976* (2022).

[47] Andrew Howard, Mark Sandler, Grace Chu, Liang-Chieh Chen, Bo Chen, Mingxing Tan, Weijun Wang, Yukun Zhu, Ruoming Pang, Vijay Vasudevan, et al. "Searching for mobilenetv3." In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 1314–1324.

[48] Zhigang Dai, Bolun Cai, Yugeng Lin, and Junying Chen. "Up-detr: Unsupervised pre-training for object detection with transformers." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2021, pp. 1601–1610.

[49] Jianyuan Guo, Kai Han, Han Wu, Yehui Tang, Xinghao Chen, Yunhe Wang, and Chang Xu. "Cmt: Convolutional neural networks meet vision transformers." In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2022, pp. 12175–12185.

[50] Niall O'Mahony, Sean Campbell, Anderson Carvalho, Suman Harapanahalli, Gustavo Velasco Hernandez, Lenka Krpalkova, Daniel Riordan, and Joseph Walsh. "Deep learning vs. traditional computer vision." In: *Advances in Computer Vision: Proceedings of the 2019 Computer Vision Conference (CVC), Volume 1 1*. Springer. 2020, pp. 128–144.

[51] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. "Imagenet: A large-scale hierarchical image database." In: *IEEE conference on computer vision and pattern recognition*. Ieee. 2009, pp. 248–255.

[52] Connor Shorten and Taghi M Khoshgoftaar. "A survey on image data augmentation for deep learning." In: *Journal of big data* 6.1 (2019), pp. 1–48.

[53] Masafumi Yamazaki, Akihiko Kasagi, Akihiro Tabuchi, Takumi Honda, Masahiro Miwa, Naoto Fukumoto, Tsuguchika Tabaru, Atsushi Ike, and Kohta Nakashima. "Yet another accelerated sgd: Resnet-50 training on imagenet in 74.7 seconds." In: *arXiv preprint arXiv:1903.12650* (2019).

[54] Sebastian Nowozin, Christoph H Lampert, et al. "Structured learning and prediction in computer vision." In: *Foundations and Trends® in Computer Graphics and Vision* 6.3–4 (2011), pp. 185–365.

[55] Roger Sapsford and Victor Jupp. *Data collection and analysis*. Sage, 1996.

[56] Robert Blumberg and Shaku Atre. "The problem with unstructured data." In: *Dm Review* 13.42-49 (2003), p. 62.

[60] R Bandara. *Bag-of-Features Descriptor on SIFT Features with OpenCV (BoF-SIFT)*. 2017. URL: https://www.codeproject.com/Articles/619039/Bag-of-Features-Descriptor-on-SIFT-Features-with-O (visited on 02/16/2024).

[61] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. "3D Semantic Parsing of Large-Scale Indoor Spaces." In: *Proceedings of the IEEE International Conference on Computer Vision and Pattern Recognition*. 2016.

[62] Richard Hartley and Andrew Zisserman. *Multiple view geometry in computer vision*. Cambridge university press, 2003.

[63] Shimon Ullman. "The interpretation of structure from motion." In: *Proceedings of the Royal Society of London. Series B. Biological Sciences* 203.1153 (1979), pp. 405–426.

[64] Johannes Lutz Schönberger and Jan-Michael Frahm. "Structure-from-Motion Revisited." In: *Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[65] Randall Smith, Matthew Self, and Peter Cheeseman. "Estimating uncertain spatial relationships in robotics." In: *Autonomous robot vehicles*. Springer, 1990, pp. 167–193.

[66] Carlos Campos, Richard Elvira, Juan J Gómez Rodríguez, José MM Montiel, and Juan D Tardós. "Orb-slam3: An accurate open-source library for visual, visual–inertial, and multimap slam." In: *IEEE Transactions on Robotics* 37.6 (2021), pp. 1874–1890.

[67] Johannes Lutz Schönberger, Enliang Zheng, Marc Pollefeys, and Jan-Michael Frahm. "Pixelwise View Selection for Unstructured Multi-View Stereo." In: *European Conference on Computer Vision (ECCV)*. 2016.

[68] Yoli Shavit and Ron Ferens. "Introduction to camera pose estimation with deep learning." In: *arXiv preprint arXiv:1907.05272* (2019).

[69] 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 (CVPR)*. 2015.

[70] David Stutz and Andreas Geiger. "Learning 3d shape completion from laser scan data with weak supervision." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2018, pp. 1955–1964.

[71] Artem Babenko, Anton Slesarev, Alexandr Chigorin, and Victor Lempitsky. "Neural codes for image retrieval." In: *Computer Vision–ECCV 2014: 13th European Conference, Zurich, Switzerland, September 6-12, 2014, Proceedings, Part I 13*. Springer. 2014, pp. 584–599.

[72] Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*. Vol. 4. 4. Springer, 2006.

[73] Herbert Bay, Andreas Ess, Tinne Tuytelaars, and Luc Van Gool. "Speeded-up robust features (SURF)." In: *Computer vision and image understanding* 110.3 (2008), pp. 346–359.

[74] Ethan Rublee, Vincent Rabaud, Kurt Konolige, and Gary Bradski. "ORB: An efficient alternative to SIFT or SURF." In: *International conference on computer vision*. IEEE. 2011, pp. 2564–2571.

[75] Shaharyar Ahmed Khan Tareen and Zahra Saleem. "A comparative analysis of sift, surf, kaze, akaze, orb, and brisk." In: *2018 International conference on computing, mathematics and engineering technologies (iCoMET)*. IEEE. 2018, pp. 1–10.

[76] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. "Superpoint: Self-supervised interest point detection and description." In: *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*. 2018, pp. 224–236.

[77] Boris Ajdin, Manuel Finckh, Christian Fuchs, Johannes Hanika, and Hendrik Lensch. *Compressive higher-order sparse and low-rank acquisition with a hyperspectral light stage*. Citeseer, 2012.

[78] Michael Bass, Eric W Van Stryland, David R Williams, and William L Wolfe. *Handbook of optics*. Vol. 2. McGraw-Hill New York, 1995.

[79] Ajay Kumar Boyat and Brijendra Kumar Joshi. "A review paper: noise models in digital image processing." In: *arXiv preprint arXiv:1505.03489* (2015).

[80] Chaoqiang Zhao, Youmin Zhang, Matteo Poggi, Fabio Tosi, Xianda Guo, Zheng Zhu, Guan Huang, Yang Tang, and Stefano Mattoccia. "Monovit: Self-supervised monocular depth estimation with a vision transformer." In: *2022 International Conference on 3D Vision (3DV)*. IEEE. 2022, pp. 668–678.

[81] Wenliang Zhao, Yongming Rao, Zuyan Liu, Benlin Liu, Jie Zhou, and Jiwen Lu. "Unleashing text-to-image diffusion models for visual perception." In: *arXiv preprint arXiv:2303.02153* (2023).

[83] Roger Mohr, Long Quan, and Françoise Veillon. "Relative 3D reconstruction using multiple uncalibrated images." In: *The International Journal of Robotics Research* 14.6 (1995), pp. 619–632.

[84] Henri P Gavin. "The Levenberg-Marquardt algorithm for nonlinear least squares curve-fitting problems." In: *Department of civil and environmental engineering, Duke University* 19 (2019).

[85] Bill Triggs, Philip F McLauchlan, Richard I Hartley, and Andrew W Fitzgibbon. "Bundle adjustment—a modern synthesis." In: *Vision Algorithms: Theory and Practice: International Workshop on Vision Algorithms Corfu, Greece, September 21–22, 1999 Proceedings*. Springer. 2000, pp. 298–372.

[86] Yasutaka Furukawa, Carlos Hernández, et al. "Multi-view stereo: A tutorial." In: *Foundations and Trends® in Computer Graphics and Vision* 9.1-2 (2015), pp. 1–148.

[87] Anastasia Ioannidou, Elisavet Chatzilari, Spiros Nikolopoulos, and Ioannis Kompatsiaris. "Deep learning advances in computer vision with 3d data: A survey." In: *ACM computing surveys (CSUR)* 50.2 (2017), pp. 1–38.

[88] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. "Segnet: A deep convolutional encoder-decoder architecture for image segmentation." In: *IEEE transactions on pattern analysis and machine intelligence* 39.12 (2017), pp. 2481–2495.

[89] Iro Armeni, Ozan Sener, Amir R. Zamir, Helen Jiang, Ioannis Brilakis, Martin Fischer, and Silvio Savarese. "3D Semantic Parsing of Large-Scale Indoor Spaces." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[90] Hugues Thomas, Charles R Qi, Jean-Emmanuel Deschaud, Beatriz Marcotegui, François Goulette, and Leonidas J Guibas. "Kpconv: Flexible and deformable convolution for point clouds." In: *Proceedings of the IEEE/CVF international conference on computer vision*. 2019, pp. 6411–6420.

[91] Daniel Scharstein and Richard Szeliski. "A taxonomy and evaluation of dense two-frame stereo correspondence algorithms." In: *International journal of computer vision* 47 (2002), pp. 7–42.

[92] Xiaolong Wang. "Learning and Reasoning with Visual Correspondence in Time." PhD thesis. Pittsburgh, PA: Carnegie Mellon University, 2019.

[93] Tong Qin, Peiliang Li, and Shaojie Shen. "Vins-mono: A robust and versatile monocular visual-inertial state estimator." In: *IEEE Transactions on Robotics* 34.4 (2018), pp. 1004–1020.

[94] Vladyslav Usenko, Nikolaus Demmel, David Schubert, Jörg Stückler, and Daniel Cremers. "Visual-inertial mapping with non-linear factor recovery." In: *IEEE Robotics and Automation Letters* 5.2 (2019), pp. 422–429.

[95] Bruce D Lucas and Takeo Kanade. "An iterative image registration technique with an application to stereo vision." In: *IJCAI'81: 7th international joint conference on Artificial intelligence*. Vol. 2. 1981, pp. 674–679.

[96] Jiayi Ma, Xingyu Jiang, Aoxiang Fan, Junjun Jiang, and Junchi Yan. "Image matching from handcrafted to deep features: A survey." In: *International Journal of Computer Vision* 129 (2021), pp. 23–79.

[97] Yuki Ono, Eduard Trulls, Pascal Fua, and Kwang Moo Yi. "LF-Net: Learning local features from images." In: *Advances in neural information processing systems* 31 (2018).

[98] Xuelun Shen, Cheng Wang, Xin Li, Zenglei Yu, Jonathan Li, Chenglu Wen, Ming Cheng, and Zijian He. "RF-Net: An end-to-end image matching network based on receptive field." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2019, pp. 8132–8140.

[101] Jeffrey Pennington, Richard Socher, and Christopher D. Manning. "GloVe: Global Vectors for Word Representation." In: *Empirical Methods in Natural Language Processing (EMNLP)*. 2014, pp. 1532–1543.

[102] Aude Oliva and Antonio Torralba. "Modeling the shape of the scene: A holistic representation of the spatial envelope." In: *International journal of computer vision* 42 (2001), pp. 145–175.

[103] Sivic and Zisserman. "Video Google: A text retrieval approach to object matching in videos." In: *Proceedings ninth IEEE international conference on computer vision*. IEEE. 2003, pp. 1470–1477.

[104] Gabriella Csurka, Christopher Dance, Lixin Fan, Jutta Willamowski, and Cédric Bray. "Visual categorization with bags of keypoints." In: *Workshop on statistical learning in computer vision, ECCV*. Vol. 1. 1-22. Prague. 2004, pp. 1–2.

[105] Florent Perronnin and Christopher Dance. "Fisher kernels on visual vocabularies for image categorization." In: *2007 IEEE conference on computer vision and pattern recognition*. IEEE. 2007, pp. 1–8.

[106] Florent Perronnin, Jorge Sánchez, and Thomas Mensink. "Improving the fisher kernel for large-scale image classification." In: *Computer Vision–ECCV 2010: 11th European Conference on Computer Vision, Heraklion, Crete, Greece, September 5-11, 2010, Proceedings, Part IV 11*. Springer. 2010, pp. 143–156.

[107] Jorge Sánchez, Florent Perronnin, Thomas Mensink, and Jakob Verbeek. "Image classification with the fisher vector: Theory and practice." In: *International journal of computer vision* 105 (2013), pp. 222–245.

[108] Hervé Jégou, Matthijs Douze, Cordelia Schmid, and Patrick Pérez. "Aggregating local descriptors into a compact image representation." In: *2010 IEEE computer society conference on computer vision and pattern recognition*. IEEE. 2010, pp. 3304–3311.

[109] Relja Arandjelovic and Andrew Zisserman. "All about VLAD." In: *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*. 2013, pp. 1578–1585.

[110] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "ImageNet Classification with Deep Convolutional Neural Networks." In: *Advances in Neural Information Processing Systems*. Vol. 25. Curran Associates, Inc., 2012.

[111] Relja Arandjelovic, Petr Gronat, Akihiko Torii, Tomas Pajdla, and Josef Sivic. "NetVLAD: CNN architecture for weakly supervised place recognition." In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 5297–5307.

[112] Farzana Anowar, Samira Sadaoui, and Bassant Selim. "Conceptual and empirical comparison of dimensionality reduction algorithms (pca, kpca, lda, mds, svd, lle, isomap, le, ica, t-sne)." In: *Computer Science Review* 40 (2021), p. 100378.

[113] Mayur Datar, Nicole Immorlica, Piotr Indyk, and Vahab S Mirrokni. "Locality-sensitive hashing scheme based on p-stable distributions." In: *Proceedings of the twentieth annual symposium on Computational geometry*. 2004, pp. 253–262.

[114] Yoseph Linde, Andres Buzo, and Robert Gray. "An algorithm for vector quantizer design." In: *IEEE Transactions on communications* 28.1 (1980), pp. 84–95.

[115] Herve Jegou, Matthijs Douze, and Cordelia Schmid. "Product quantization for nearest neighbor search." In: *IEEE transactions on pattern analysis and machine intelligence* 33.1 (2010), pp. 117–128.

[116] Patrick Wieschollek, Oliver Wang, Alexander Sorkine-Hornung, and Hendrik P. A. Lensch. "Efficient Large-Scale Approximate Nearest Neighbor Search on the GPU." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016.

[117] Martin Aumüller, Erik Bernhardsson, and Alexander Faithfull. "ANN-Benchmarks: A benchmarking tool for approximate nearest neighbor algorithms." In: *Information Systems* 87 (2020), p. 101374.

[118] Yury A Malkov and Dmitry A Yashunin. "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs." In: *IEEE transactions on pattern analysis and machine intelligence* (2018).

[119] Jon M Kleinberg. "Navigation in a small world." In: *Nature* 406.6798 (2000), pp. 845–845.

[120] Ben Harwood and Tom Drummond. "Fanng: Fast approximate nearest neighbour graphs." In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2016, pp. 5713–5722.

[121] Cong Fu and Deng Cai. "Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph." In: *arXiv preprint arXiv:1609.07228* (2016).

[122] Jie Ren, Minjia Zhang, and Dong Li. "Hm-ann: Efficient billion-point nearest neighbor search on heterogeneous memory." In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 10672–10684.

[123] Jeff Johnson, Matthijs Douze, and Hervé Jégou. "Billion-scale similarity search with GPUs." In: *IEEE Transactions on Big Data* (2019).

[124] Sebastian Borgeaud, Arthur Mensch, Jordan Hoffmann, Trevor Cai, Eliza Rutherford, Katie Millican, George Bm Van Den Driessche, Jean-Baptiste Lespiau, Bogdan Damoc, Aidan Clark, et al. "Improving language models by retrieving from trillions of tokens." In: *International conference on machine learning*. PMLR. 2022, pp. 2206–2240.

[125] Hui Wang, Wan-Lei Zhao, and Xiangxiang Zeng. "Large-Scale Approximate k-NN Graph Construction on GPU." In: *arXiv preprint arXiv:2103.15386* (2021).

[126] Shulin Zeng, Zhenhua Zhu, Jun Liu, Haoyu Zhang, Guohao Dai, Zixuan Zhou, Shuangchen Li, Xuefei Ning, Yuan Xie, Huazhong Yang, et al. "DF-GAS: a Distributed FPGA-as-a-Service Architecture towards Billion-Scale Graph-based Approximate Nearest Neighbor Search." In: *Proceedings of the 56th*

Annual IEEE/ACM International Symposium on Microarchitecture. 2023, pp. 283–296.

[127] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. "High-resolution image synthesis with latent diffusion models." In: *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*. 2022, pp. 10684–10695.

[128] Ben Poole, Ajay Jain, Jonathan T Barron, and Ben Mildenhall. "Dreamfusion: Text-to-3d using 2d diffusion." In: *arXiv preprint arXiv:2209.14988* (2022).

[129] Ruoshi Liu, Rundi Wu, Basile Van Hoorick, Pavel Tokmakov, Sergey Zakharov, and Carl Vondrick. "Zero-1-to-3: Zero-shot one image to 3d object." In: *Proceedings of the IEEE/CVF International Conference on Computer Vision*. 2023, pp. 9298–9309.

IMAGES

[57] Mike Switzerland Mike Lehmann. *RedCat 8727*. CC BY-SA 2.5 https://creativecommons.org/licenses/by-sa/2.5. 2006. URL: https://de.wikipedia.org/wiki/Datei:RedCat_8727.jpg.

[58] Eadweard Muybridge. *Muybridge race horse gallop*. Public domain. 1904.

[59] Daniel L. Lu. *Ouster OS1-64 lidar point cloud of intersection of Folsom and Dore St, San Francisco*. CC BY 4.0 https://creativecommons.org/licenses/by/4.0/. 2019. URL: https://de.wikipedia.org/wiki/Datei:Ouster_OS1-64_lidar_point_cloud_of_intersection_of_Folsom_and_Dore_St,_San_Francisco.png.

[82] Fabian Groh. *Realtime Multi-Kinect 3D Object Video*. 2013.

[99] Ugrashak. *An up-close picture of a curious male domestic shorthair tabby cat*. CC BY-SA 4.0 https://creativecommons.org/licenses/by-sa/4.0. 2017. URL: https://en.m.wikipedia.org/wiki/File:An_up-close_picture_of_a_curious_male_domestic_shorthair_tabby_cat.jpg.

[100] Christian Mehlführer. *Bradypus variegatus*. CC BY 2.5 https://creativecommons.org/licenses/by/2.5. 2007. URL: https://de.wikipedia.org/wiki/Datei:MC_Drei-Finger-Faultier.jpg.

DECLARATION

Ich erkläre hiermit, dass ich die zur Promotion eingereichte Arbeit mit dem Titel: *Efficient Processing and Learning on Unstructured Data* selbständig verfasst, nur die angegebenen Quellen und Hilfsmittel benutzt und wörtlich oder inhaltlich übernommene Stellen als solche gekennzeichnet habe. Ich erkläre, dass die Richtlinien zur Sicherung guter wissenschaftlicher Praxis der Universität Tübingen (Beschluss des Senats vom 25.5.2000) beachtet wurden. Ich versichere an Eides statt, dass diese Angaben wahr sind und dass ich nichts verschwiegen habe. Mir ist bekannt, dass die falsche Abgabe einer Versicherung an Eides statt mit Freiheitsstrafe bis zu drei Jahren oder mit Geldstrafe bestraft wird.

Tübingen, 2024

Fabian Groh

COLOPHON

This document was typeset using the typographical look-and-feel `classicthesis` developed by André Miede and Ivo Pletikosić. The style was inspired by Robert Bringhurst's seminal book on typography "*The Elements of Typographic Style*". `classicthesis` is available for both LaTeX and LyX:

https://bitbucket.org/amiede/classicthesis/

Part II

APPENDIX: PAPERS ON EFFICIENT PROCESSING AND LEARNING ON UNSTRUCTURED DATA

A

MULTI-VIEW CONTINUOUS STRUCTURED LIGHT SCANNING

Multi-view continuous structured light scanning

Fabian Groh, Benjamin Resch, and Hendrik P.A. Lensch

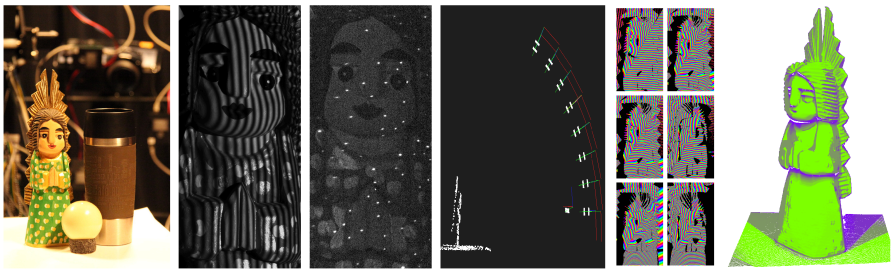
German Conference on Pattern Recognition (GCPR), 2017 [1]

Multi-View Continuous Structured Light Scanning

Fabian Groh, Benjamin Resch, Hendrik P. A. Lensch

University of Tübingen

Abstract. We introduce a highly accurate and precise multi-view, multi-projector, and multi-pattern phase scanning method for shape acquisition that is able to handle occlusions and optically challenging materials. The 3D reconstruction is formulated as a two-step process which first estimates reliable measurement samples and then simultaneously optimizes over all cameras, projectors, and patterns. This holistic approach results in significant quality improvements. Furthermore, the acquisition time is drastically reduced by relying on just six high-frequency sinusoidal captures without the need of phase unwrapping, which is implicitly provided by the multi-view geometry.



(a) (b) (c) (d) (e) (f)

Fig. 1: For objects (a) with optically challenging properties, shifted sinusoidal patterns (b) are captured from multiple cameras and projectors. On top of the phase information, highly accurate features (c) are generated to perform a precise bundle adjustment (d). With the multi-view continuous phase signals (e) the reliable camera-projector pairs for every surface point are estimated by validating multi-view consistency (f). Finally, we optimize the depth over all available information.

1 Introduction

In order to capture a complete model and to provide a good coverage of the visible surface area of any real-world object, a multi-view approach is neces-

sary. In particular, these multi-view recordings are relevant in the context of object recognition, industrial inspection and material acquisition where also the illumination is varied. In all applications, it is beneficial to acquire a detailed geometry model to establish precise correspondences between the different views. In this paper, we propose an active multi-view, multi-projector shape acquisition system.

Multi-view acquisition systems typically consist of multiple cameras that are either distributed at fixed positions [21], or are movable [1] around an object, or get extended with projectors to perform active structured light (SL) scanning alongside reflection measurements [24]. While our proposed method is developed to be used inside such an illumination system for material acquisition, it purely focuses on achieving high-quality active SL geometry reconstruction in the general case of multiple viewpoints. In this case, each projector-pattern produces a continuous SL signal that is captured by every camera as a two-dimensional projection. The main idea is that a surface point is located exactly where all SL signals from different viewpoints align, provided that the surface is visible from the respective camera-projector pair and the signal is not corrupted by material properties like specular- or inter-reflections.

To surpass the projector resolution, which is the most limiting factor of SL scanning, continuous coding methods [20] are suggested. We use traditional phase shifting methods (PS), where phases are recovered from projecting at least three shifted sinusoidal patterns [23]. Phase shifting methods have been proven to deliver high-quality results with sub-pixel accuracy, even in optically difficult areas [5, 6, 14, 17]. We fully integrate phase shifting into our multi-view reconstruction pipeline with two benefits. The multi-view approach further improves the accuracy, and we can eliminate the need for explicit phase unwrapping by a multi-view consistency validation. Thus, our method just needs to capture the highest frequency patterns. Real world objects are often composed out of different materials with varying illumination profiles, which make it necessary to capture the phase shifting patterns in high dynamic range (HDR) sequences to achieve precise results in all areas.

Since our optimization requires correct camera and projector poses, a very accurate calibration is required. We perform a precise online bundle adjustment (BA) on actively marked corresponding points. Therefore, we propose to combine the HDR phase shifting results with a fast LDR projector pixel identification scheme to get highly accurate sub-pixel correspondences for a structure from motion reconstruction.

The presented results feature a geometry accuracy far beyond the pixel resolution of both the involved cameras and projectors.

2 Related Work

3D Reconstruction: Three-dimensional shape estimation from real world objects is still a very challenging task and an active field of research. In general, methods can be classified into two major categories: namely passive and active. In

passive approaches, the scene is just captured from at least two viewpoints. The most prominent representatives are multi-view stereo systems. However, most of these techniques depend heavily on finding good salient point correspondences between the views. Thus, they have problems in optically challenging and textureless areas that often results in sparse reconstructions. A comparison and evaluation can be found in [22].

Active techniques overcome this issue by establishing correspondences between camera and projector pixels for each scene point with active illumination patterns. Besides laser scanning [4], projector-based structured light pattern are a well-studied technique [7,20]. The projected patterns establish correspondences on the object, in the way that every point has a corresponding code-word, which can be recovered by analyzing the differences in the captured images with the projected patterns. Salvi et al. [20] compare state of the art structured light patterns.

Structured light scanning in the appearance of global illumination, interreflections or challenging materials, e.g. specular reflections, can still be an issue. High frequency patterns can cope with that kind of problems [5,6,17]. Gupta et al. [14] propose to combine different Gray code patterns logically in an ensemble. This method achieves remarkable results. Nevertheless, the used patterns just provide a discrete coding, and in consequence are not able to surpass the projector resolution on their own. Hence, they applied the same technique to a phase shifting method as Micro PS [15].

In traditional approaches, structured light patterns need to be temporally unwrapped by projecting coarse-to-fine patterns consecutively to get a unique correspondence for each point. The disadvantage is the number of additional patterns that are dependent on the projector resolution. In the context of phase shifting methods, this is referred to as phase unwrapping [20], where each frequency band needs to be captured by at least three shifts. Multiple methods are addressing this issue by reducing the amount of necessary patterns [20], e.g. Embedded PS [16], and Micro PS [15]. Our approach replaces phase-unwrapping by exploiting the multi-view setup.

Multi View Structured Light Scanning: For the purpose of phase unwrapping in a stereo camera plus projector system, Garcia and Zakhor [11] present a method that performs a correspondence labeling in the projector domain via loopy belief propagation. Afterwards, missing absolute phases are estimated in the camera domain from neighboring pixels.

Binary structured light scanning from uncalibrated viewpoints has been proposed by several approaches: with a laser pointer [8], for multiple viewpoints [9, 13], and for multiple projectors in a one-shot setting [10]. In a setting with multiple cameras Young et al. [25] suggest using viewpoint-coded structured light to mimic the temporal encoding. In recent years, using multiple cameras and multiple projectors to capture geometry alongside photometric data in self-calibrating systems has been demonstrated by the work of Aliaga et al. [2,3] and Weinmann et al. [24]. Aliaga et al. utilize the projectors as additional virtual

cameras. While this is beneficial in settings with few cameras, it makes their approach even more dependent on the projector resolution. For a denser surface, they perform an up-sampling scheme by warping the photometric captures onto the geometrical model [2, 3]. Weinmann et al. [24] suggest utilizing overlapping areas of the projected binary codes from multiple projectors to overcome low projector resolutions. The final resolution of the surface area is directly dependent on the overlapping alignment of the projector pixel on the scene.

In contrast, our method utilizes traditional single phase shifting to achieve continuous signals in the scene [23]. Thus, it is possible to optimize directly on the continuous signals from multiple camera-projector pairs to estimate the surface. Further, we introduce a novel multi-view consistency validation that substitutes phase unwrapping, relies solely on high-frequencies patterns at no additional cost, and reliably handles occlusions. All calculations can be done for each 3D point separately without neighboring information. Additionally, we propose to enhance active sparse bundle adjustment with the continuous signal for better multi-view precision.

3 Multi-View Depth Optimization

Our reconstruction pipeline makes use of phase shifting patterns for two steps: Once, to establish active correspondence to perform a highly accurate bundle adjustment to estimate the camera and projector locations, and, second, for optimizing the 3D geometry.

For the actual depth estimation, at first, a viewpoint consistency validation is completed by coarsely sampling potential depth values. This step performs an explicit multi-view phase unwrapping by utilizing geometrical constraints. The step identifies for each pixel which camera-projector pairs provide valid samples. Subsequently, all points are further optimized solely on the reliable phase signal to achieve highly precise and accurate depth information. Figure 1 shows examples of the specific steps.

3.1 Multi-View HDR Continuous SL Calibration

This section describes our combination of the HDR phase scanning with an active sparse bundle adjustment (SBA). Initially, we calculate the phase responses for a horizontal and vertical projector pattern direction. Afterwards, sub-pixel accurate feature points are generated by utilizing phase responses. These precisely positioned feature points establish exact correspondences between all cameras and all projectors suitable for a high-precision SBA.

We perform this calibration online during the capturing process to estimate all extrinsic and also intrinsic parameters for a moving camera setup. In a fixed system the same calibration could also be performed as a pre-processing step.

HDR Phase Scanning: We use traditional phase shifting [5, 20] for the x and y directions of the projectors respectively. Given a linearized projector, a set of

N shifted sine patterns $L_{n,x}$ in direction x, period T is generated:

$$L_{n,x}(x,y) = 0.5\,\cos\left(x\,\omega + \theta_n\right) + 0.5, \tag{1}$$

with $\omega = {}^{1}/_{T}$ and $\theta_n = {}^{n2\pi}/_{N}$.

Once successively projected and captured by the camera the phase vector $\boldsymbol{u} = [o, c, s]$ (offset, cosine, sine) can be recovered for a single frequency given the captured intensity responses $\boldsymbol{r} = [r_0, r_1, \ldots, r_n]$ (see [16]):

$$\boldsymbol{u} = \underset{u}{\operatorname{argmin}} \|\boldsymbol{r} - A\boldsymbol{u}\|_2\,, \quad \text{with } A \coloneqq \begin{bmatrix} 1 & \cos(\theta_0) & -\sin(\theta_0) \\ 1 & \cos(\theta_1) & -\sin(\theta_1) \\ \vdots & \vdots & \vdots \\ 1 & \cos(\theta_n) & -\sin(\theta_n) \end{bmatrix}. \tag{2}$$

The phase ϕ is obtained by $\phi = \tan^{-1}\left({}^{s}/_{c}\right)$.

Since our goal is to recover the shape of objects with optically challenging materials, the acquisition with a HDR sequence is essential, for which we use the algorithm provided by Granados et al. [12].

Representing Phase Information: All phase differences and interpolations in this work are computed on the respective sine and cosine part, not on the angle. Consequently, for two phase responses a and b, we represent the norm as:

$$\|a - b\|_{\Phi} \coloneqq \sqrt{\left(\cos\left(a\right) - \cos\left(b\right)\right)^2 + \left(\sin\left(a\right) - \sin\left(b\right)\right)^2}\,, \tag{3}$$

and for linear interpolation:

$$lerp(a, b; \lambda)_{\Phi} \coloneqq \begin{bmatrix} (1-\lambda) * \cos(a) + \lambda \cdot \cos(b) \\ (1-\lambda) * \sin(a) + \lambda \cdot \sin(b) \end{bmatrix}. \tag{4}$$

While this approximation introduces errors on larger intervals, on small scale they perform close to the optimum. This is sufficient, since higher differences are thresholded and interpolation is most likely performed on very small scale. Especially on GPUs, this not only avoids unnecessary branching when dealing with phase values, but also enables the utilization of the texture hardware.

Active Sparse Bundle Adjustment: After capturing the phases patterns for each viewpoint, a sparse set of randomly sampled projector pixels are projected temporally onto the scene using a binary encoding to enumerate these pixels. This is very robust and done in LDR with high gain to be much faster than a single capture of a HDR sequence.

For the selected projector pixels the analytic phase is known. An initial camera pixel position for these feature points is obtained as the center of mass of the detected spots. Afterwards the the sub-pixel location is refined by the Nelder-Mead procedure [18] to match the analytically given phase to the measured phase in the camera image. This optimization procedure yields very accurate correspondences and subsequently more precise results in the SBA. For the SBA we use the work from [19].

3.2 Multi-View Depth Optimization

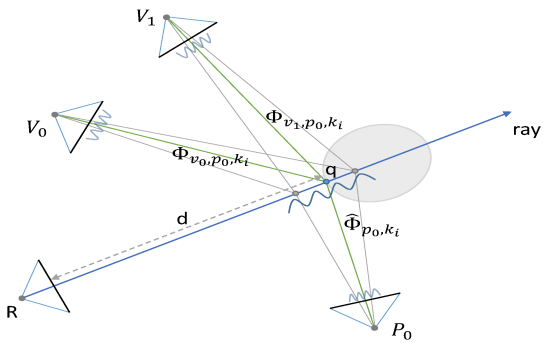


Fig. 2: General concept: Points along a ray are projected into all view-projector pairs. Each point is evaluated based on the phase differences between the obtained and the expected phase values for all patterns.

The general concept of our shape reconstruction is to recover the depth of a surface point by a multi-view optimization, i.e. combining the information of all cameras, projectors and patterns. Tracing a ray into the scene from an arbitrarily chosen reference camera system R, we are able to project any point q of depth d along the ray into all camera-views V and projectors P to get the respective phase responses $\Phi_{v,p,k}$ as well as the expected projector phases $\hat{\Phi}_{p,k}$ for all patterns K. This procedure is illustrated in Figure 2. For an actual surface point the phase differences $\|\Phi_{v,p,k}(d) - \hat{\Phi}_{p,k}(d)\|_{\Phi}$ are required to be minimal. However, we need to ensure to only operate on correct phase signals, since occlusion and optical properties could induce corrupted phase information. Hence, the depth reconstruction can be formulated as an optimization problem over all views, projectors and patterns:

$$d = \underset{d}{\operatorname{argmin}} \sum_{p \in P} \sum_{v \in V} \sum_{k \in K} \Omega_{v,p,k}(d) \, \|\Phi_{v,p,k}(d) - \hat{\Phi}_{p,k}(d)\|_{\Phi} \,, \tag{5}$$

where $\Omega_{v,p,k}(d)$ is our multi-view consistency validation that evaluates to 1 if the phase information is estimated to be reliable and 0 otherwise.

Multi-view Consistency Validation: This step identifies which phase responses are reliable. For that reason, correct phase responses need to be distinguished from false positives. These wrongly predicted phase signals occur very often in a system that only relies on high frequency patterns. On the other side, phase responses can also be truly wrong or corrupted by optical influences such as inter-reflections (false negatives). As neither of those signals should be used

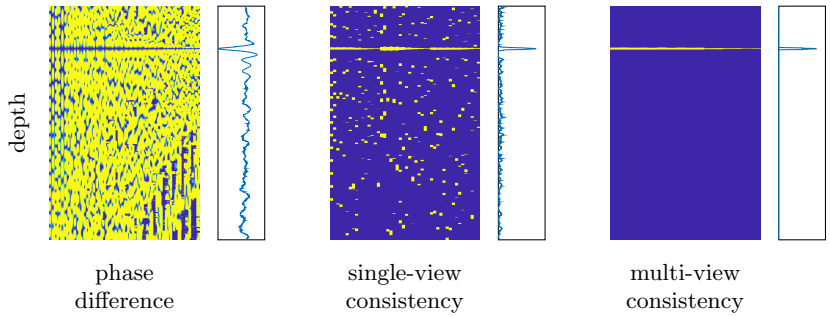


Fig. 3: Evaluation for an example point along a ray (depth) with the error/consistency for all camera-projector-pattern phase signals (horizontal) and the sum of each depth. Our multi-view consistency scheme is able to determine the contributing camera-projector pairs reliably and eliminates all false candidates.

for the optimization, it is necessary to keep track of such events to not falsely exclude cameras or projectors.

We utilize the geometrical setup to perform a consistency check. The acceptance measure of a single pattern is defined as:

$$\hat{\Omega}_{v,p,k}(d) \coloneqq \begin{cases} 1, & \text{if } \|\Phi_{v,p,k}(d) - \hat{\Phi}_{p,k}(d)\|_{\Phi} < \theta \\ 0, & \text{otherwise} \end{cases}, \tag{6}$$

where θ is the allowed threshold difference for a phase value to still be acceptable.

Further, for the correct depth all patterns from the same view-projector pair need to be consistent as well: $\hat{\Omega}_{v,p,k_0}(d) \wedge \cdots \wedge \hat{\Omega}_{v,p,k_{K-1}}(d)$ (single-view consistency).

Finally, we consider the fact that for any point in the scene it is not possible to have disjunctive subsets of valid view-projector pairs: If a projector is valid in one view it cannot be occluded in the other views; and vice versa. If a view is able to receive the signal from one projector this view cannot be occluded for the other projectors. Firstly, the contributing projectors are determined by evaluating the overlap of valid views. Subsequently, a view is accepted if it is valid for at least half of the contributing projectors. In that way, we allow for some optical disturbances. Nonetheless, corrupted phase responses are still excluded from the final optimization. The validation for a single point at different depth is demonstrated in Figure 3.

Coarse Approximation and Multi-view-based Phase Unwrapping: For the purpose of finding points close to a surface, the number of consistent phases correlates to a good alignment. Thus, performing multi-view consistency validation along the depth substitutes phase unwrapping.

With relaxing the consistency threshold θ, it is possible to coarsely sample the depth along the ray to approximate a good initial alignment.

Fine Depth Estimation: Our fine depth estimation takes the result of the coarse approximation as seed and performs a simple binary search on Equation 5. We compute the variance σ of the phase differences for the final depth as an additional measure of fitness.

4 Experimental Hardware Setup

All experiments in this paper have been captured in a light stage setting, with two Point Grey Grasshopper3 (GS3-U3-120S6C-C) 12 MP cameras and three Viewsonic Pro9000 1080p LED projectors. The two cameras are moved along an arc at about 1 m distance around the scene to different positions. They are treated as independent cameras in our system. The projectors have to be placed outside the light stage. Otherwise, they would occlude lightpaths for the reflectance measurement. Thus, their distance is roughly 1.40 m, and the scene is only covered by a subset of the projected area for each projector. The projector resolution on the scene is only about 500 µm, whereas the camera has a resolution of about 80 µm on the object. The setting is shown in Figure 5.

For the structured light scanning, we use two shifted sinusoidal patterns in horizontal as well as vertical direction respectively with a period length of 8 pixels and three shifts. Hence, we need to capture 6 HDR sequences per projector. We chose to capture the scenes from 8 positions along the two-camera-arc with an angle of about 10 degrees apart (vertical) and with the three projectors to the side (horizontal). Altogether, we optimize on 48 camera-projector pairs with two patterns each which result in 96 phases signals. For the demonstration of the results, we always choose the lowest camera viewpoint as the reference view. The global threshold for acceptable points is 8 camera-projector pairs and only phase differences below $\theta = \pi/6$ are considered acceptable. The coarse estimation is calculated in 100 µm steps.

Since all work is done on a per point basis, the problem is highly parallelizable. The optimization is carried out on the GPU requiring just a few seconds of computation time for a whole scene.

5 Results

In this section, we demonstrate our results on three objects that are shown in Figure 5. The angel is carved out of wood and painted in different colors. The small gold plates are a specular alloy. The mug consists of a plastic top and a rubber middle part with embossed details. In between and at the ground it consists of brushed metal. The ball is a good target since it features specular reflection as well as subsurface scattering; Furthermore, the diameter is known for ground truth evaluation.

Firstly, we demonstrate that our coarse depth estimation finds correct regions and respective visibilities. The colored point clouds are shown in Figure 4. Bright green areas display that all camera-projector pairs are reliable, whereas the darker regions indicate a drop of supported pairs. Keep in mind that the

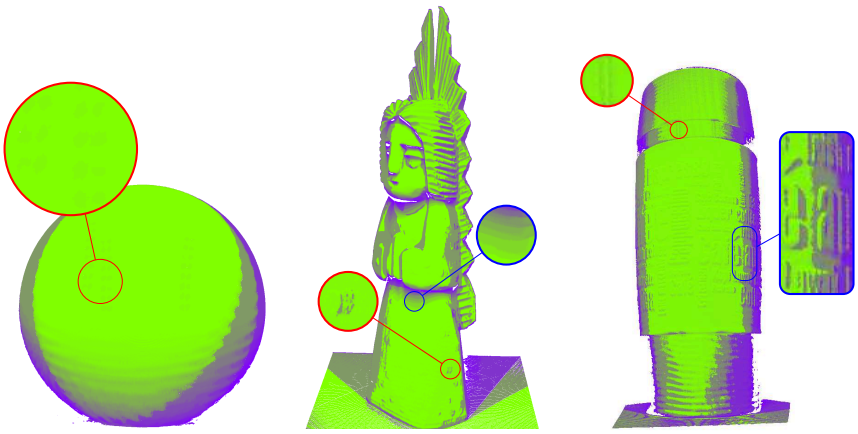


Fig. 4: Pointclouds of the estimated depth with respect to a reference camera. The color encodes the number of contributing camera-projector pairs: green (all) to purple (few). Changes are due to excluding unreliable phase information. Examples are given for specular reflections (red) and occlusions (blue).

color does not signal a change in the geometry. Most of the differences are due to occlusion of different projectors, or cameras (blue). The small, isolated areas on the ball as well as on the body of the angel are due to highly specular reflections in some of the camera-projector patterns (red). At the lower mirroring part of the mug, we have inter-reflection with the ground. Nonetheless, most of the shape up front and on the ground is recovered at high precision. Especially on the ground of the mug, many phase signals are dropped due to inconsistency introduced by the interreflections. This shape would definitively benefit from a more azimuthal camera setting. There are also some inaccuracies in the very dark top area due to high gain used to accelerate the capturing.

For the evaluation of our fine optimization step, we follow the work of Weinmann et al. [24]. The reconstructed point cloud of the cue ball is mapped against a sphere and normalized with respect to the specified diameter of 6.02 cm. We need to stress that we do not perform any removal of outliers, except defining an ROI to crop out the mounting device of the ball. We vary the set of used points dependent on the number of at least visible camera-projector pairs and accepted variance in the result of the fine optimization step. The numbers are reported in Table 1 as RMSE values in µm. These numbers indicate that the number of pairs is an important factor for achieving high accuracy.

Comparing the numbers with Weinmann et al., they achieved an RMSE of 23.3 µm with 51 cameras and 10 projectors. We achieve an RMSE error of down to 13.9 µm for the points visible from almost all cameras and projectors. Even

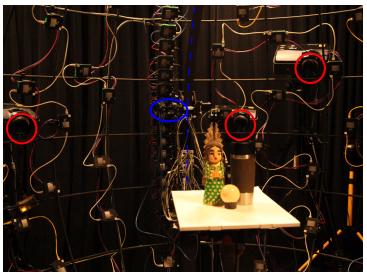


Fig. 5: The test objects in our experimental hardware setup. (Blue: Cameras on the arc. Red: Projectors)

	Cam/Proj-Pairs			
σ^2	8	15	30	45
$\pi/12$	25.2	22.1	18.0	15.0
$\pi/18$	24.9	21.9	17.7	15.0
$\pi/24$	21.9	20.3	17.2	14.9
$\pi/36$	16.1	15.2	14.1	13.9

Table 1: Results of the cue ball evaluation (RMSE values in µm).

when we include less reliable points, the numbers are still comparable, but with much fewer required cameras and projectors. Furthermore, we need to capture only 6 images per projector, whereas their approach would be on 44 for the same projector resolution.

6 Conclusion

We propose an accurate and precise multi-view phase scanning method for robust 3D reconstruction that is able to handle occlusions and optically challenging materials with e.g. subsurface scattering and specular reflections. We demonstrate that optimizing over all cameras, all projectors and all patterns simultaneously improves the overall accuracy significantly. Nonetheless, we only rely on capturing the highest frequency phase shifting patterns. Phase unwrapping with lower frequencies is substituted by a multi-view consistency validation. The final optimization considers the phase of all available SL responses which have been judged to be reliable. The quality of the reconstructed results clearly depends on the number of reliable camera-projector pairs, which strongly indicates that the method fully exploits the available multi-view information for improved accuracy.

While we demonstrate that our combined multi-view and multi-projector approach is able to cope with subsurface scattering and specular reflections, areas of very strong inter-reflections that affect almost all projectors still pose a challenge. The proposed framework is easily extendable with other continuous structured light patterns or even a mixture of different patterns for potentially more reliable detection of correct camera-projector pairs.

Acknowledgement.

This work was supported by the German Research Foundation (DFG): SFB 1233, Robust Vision: Inference Principles and Neural Mechanisms, TP 2.

References

1. Ajdin, B., Finckh, M., Fuchs, C., Hanika, J., Lensch, H.: Compressive higher-order sparse and low-rank acquisition with a hyperspectral light stage. Citeseer (2012)
2. Aliaga, D.G., Xu, Y.: Photogeometric structured light: A self-calibrating and multi-viewpoint framework for accurate 3d modeling. In: Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on. pp. 1–8. IEEE (2008)
3. Aliaga, D.G., Xu, Y.: A self-calibrating method for photogeometric acquisition of 3d objects. IEEE transactions on pattern analysis and machine intelligence 32(4), 747–754 (2010)
4. Blais, F.: Review of 20 years of range sensor development. Journal of Electronic Imaging 13(1) (2004)
5. Chen, T., Lensch, H.P., Fuchs, C., Seidel, H.P.: Polarization and phase-shifting for 3d scanning of translucent objects. In: 2007 IEEE Conference on Computer Vision and Pattern Recognition. pp. 1–8. IEEE (2007)
6. Chen, T., Seidel, H.P., Lensch, H.P.: Modulated phase-shifting for 3d scanning. In: Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on. pp. 1–8. IEEE (2008)
7. Davis, J., Ramamoorthi, R., Rusinkiewicz, S.: Spacetime stereo: A unifying framework for depth from triangulation. In: Computer Vision and Pattern Recognition, 2003. Proceedings. 2003 IEEE Computer Society Conference on. vol. 2, pp. II–359. IEEE (2003)
8. Furukawa, R., Kawasaki, H.: Dense 3d reconstruction with an uncalibrated stereo system using coded structured light. In: Computer Vision and Pattern Recognition-Workshops, 2005. CVPR Workshops. IEEE Computer Society Conference on. pp. 107–107. IEEE (2005)
9. Furukawa, R., Kawasaki, H.: Uncalibrated multiple image stereo system with arbitrarily movable camera and projector for wide range scanning. In: 3-D Digital Imaging and Modeling, 2005. 3DIM 2005. Fifth International Conference on. pp. 302–309. IEEE (2005)
10. Furukawa, R., Sagawa, R., Kawasaki, H., Sakashita, K., Yagi, Y., Asada, N.: One-shot entire shape acquisition method using multiple projectors and cameras. In: Image and Video Technology (PSIVT), 2010 Fourth Pacific-Rim Symposium on. pp. 107–114. IEEE (2010)
11. Garcia, R.R., Zakhor, A.: Consistent stereo-assisted absolute phase unwrapping methods for structured light systems. IEEE Journal of Selected Topics in Signal Processing 6(5), 411–424 (2012)
12. Granados, M., Ajdin, B., Wand, M., Theobalt, C., Seidel, H.P., Lensch, H.P.: Optimal hdr reconstruction with linear digital cameras. In: Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. pp. 215–222. IEEE (2010)
13. Gühring, J.: Dense 3d surface acquisition by structured light using off-the-shelf components. In: Photonics West 2001-Electronic Imaging. pp. 220–231. International Society for Optics and Photonics (2000)
14. Gupta, M., Agrawal, A., Veeraraghavan, A., Narasimhan, S.G.: Structured light 3d scanning in the presence of global illumination. In: Computer Vision and Pattern Recognition (CVPR), 2011 IEEE Conference on. pp. 713–720. IEEE (2011)
15. Gupta, M., Nayar, S.K.: Micro phase shifting. In: Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on. pp. 813–820. IEEE (2012)
16. Moreno, D., Son, K., Taubin, G.: Embedded phase shifting: Robust phase shifting with embedded signals. In: 2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR). pp. 2301–2309. IEEE (2015)

17. Nayar, S.K., Krishnan, G., Grossberg, M.D., Raskar, R.: Fast separation of direct and global components of a scene using high frequency illumination. In: ACM Transactions on Graphics (TOG). vol. 25, pp. 935–944. ACM (2006)
18. Nelder, J.A., Mead, R.: A simplex method for function minimization. The computer journal 7(4), 308–313 (1965)
19. Resch, B., Lensch, H.P., Wang, O., Pollefeys, M., Hornung, A.S.: Scalable structure from motion for densely sampled videos. In: CVPR (2015)
20. Salvi, J., Fernandez, S., Pribanic, T., Llado, X.: A state of the art in structured light patterns for surface profilometry. Pattern recognition 43(8), 2666–2680 (2010)
21. Schwartz, C., Sarlette, R., Weinmann, M., Klein, R.: Dome ii: A parallelized btf acquisition system. In: Rushmeier, H., Klein, R. (eds.) Eurographics Workshop on Material Appearance Modeling: Issues and Acquisition. pp. 25–31. Eurographics Association (Jun 2013), http://diglib.eg.org/EG/DL/WS/MAM/MAM2013/025-031.pdf
22. Seitz, S.M., Curless, B., Diebel, J., Scharstein, D., Szeliski, R.: A comparison and evaluation of multi-view stereo reconstruction algorithms. In: 2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06). vol. 1, pp. 519–528. IEEE (2006)
23. Srinivasan, V., Liu, H.C., Halioua, M.: Automated phase-measuring profilometry: a phase mapping approach. Appl. Opt. 24(2), 185–188 (Jan 1985), http://ao.osa.org/abstract.cfm?URI=ao-24-2-185
24. Weinmann, M., Schwartz, C., Ruiters, R., Klein, R.: A multi-camera, multi-projector super-resolution framework for structured light. In: 3D imaging, modeling, processing, visualization and transmission (3DIMPVT), 2011 international conference on. pp. 397–404. IEEE (2011)
25. Young, M., Beeson, E., Davis, J., Rusinkiewicz, S., Ramamoorthi, R.: Viewpoint-coded structured light. In: Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on. pp. 1–8. IEEE (2007)

B

FLEX-CONVOLUTION

Flex-Convolution: Million-scale point-cloud learning beyond grid-worlds

Fabian Groh, Patrick Wieschollek, and Hendrik P.A. Lensch
Asian Conference on Computer Vision (ACCV), 2018 [2]

Project side with video: https://uni-tuebingen.de/en/136528
Code: https://github.com/cgtuebingen/Flex-Convolution
Updated pre-print version: https://arxiv.org/abs/1803.07289

Flex-Convolution
Million-Scale Point-Cloud Learning Beyond Grid-Worlds

Fabian Groh[1( )], Patrick Wieschollek[1,2], and Hendrik P. A. Lensch[1]

[1] University of Tübingen, Tübingen, Germany
fabian.groh@uni-tuebingen.de
[2] Max Planck Institute for Intelligent Systems, Tübingen, Germany

Abstract. Traditional convolution layers are specifically designed to exploit the natural data representation of images – a fixed and regular grid. However, unstructured data like 3D point clouds containing irregular neighborhoods constantly breaks the grid-based data assumption. Therefore applying best-practices and design choices from 2D-image learning methods towards processing point clouds are not readily possible. In this work, we introduce a natural generalization *flex-convolution* of the conventional convolution layer along with an efficient GPU implementation. We demonstrate competitive performance on rather small benchmark sets using fewer parameters and lower memory consumption and obtain significant improvements on a million-scale real-world dataset. Ours is the first which allows to efficiently process 7 million points *concurrently*.

1 Introduction

Deep Convolutional Neural Networks (CNNs) shine on tasks where the underlying data representations are based on a regular grid structure, e.g., pixel representations of RGB images or transformed audio signals using Mel-spectrograms [11]. For these tasks, research has led to several improved neural network architectures ranging from VGG [24] to ResNet [9]. These architectures have established state-of-the-art results on a broad range of classical computer vision tasks [29] and effortlessly process entire HD images (∼2 million pixels) within a single pass. This success is fueled by recent improvements in hardware and software stacks (*e.g.* TensorFlow), which provide highly efficient implementations of layer primitives [15] in specialized libraries [6] exploiting the grid-structure of the data. It seems appealing to use grid-based structures (*e.g.* voxels) to process higher-dimensional data relying on these kinds of layer implementations. However, grid-based approaches are often unsuited for processing irregular point clouds and unstructured data. The grid resolution on equally spaced grids poses

Electronic supplementary material The online version of this chapter (https://doi.org/10.1007/978-3-030-20887-5_7) contains supplementary material, which is available to authorized users.

C. V. Jawahar et al. (Eds.): ACCV 2018, LNCS 11361, pp. 105–122, 2019.
https://doi.org/10.1007/978-3-030-20887-5_7

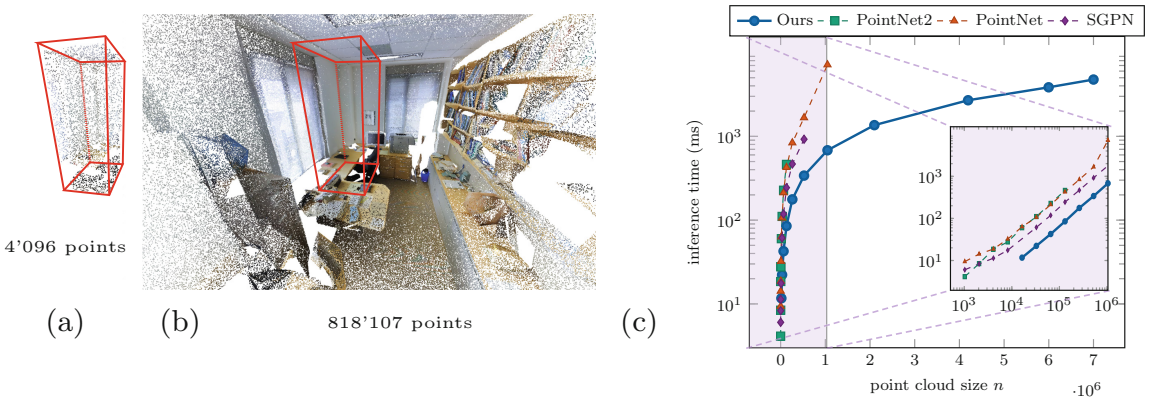


Fig. 1. Processing full-resolution point clouds is an important ingredient for successful semantic segmentation. Previous methods [17,19,28] subsample small blocks (a), while ours (b) processes the entire room and can (c) handle inputs up to 7 Million points in a single forward-pass with the *same* accuracy. Previous methods *could* handle at most 1 Million points – but training is not feasible on today's hardware.

a trade-off between discretization artifacts and memory consumption. Increasing the granularity of the cells is paid by higher memory requirements that even grows exponentially due to the curse of dimensionality.

While training neural networks on 3D voxel grids is possible [16], even with hierarchical octrees [20] the maximum resolution is limited to 256^3 voxels—large data sets are currently out-of-scope. Another issue is the discretization and resampling of continuous data into a fixed grid. For example, depth sensors produce an arbitrarily oriented depth map with different resolution in x, y and z. In Structure-from-Motion, the information of images with arbitrary perspective, orientation and distance to the scene—and therefore resolution—need to be merged into a single 3D point cloud. This potentially breaks the grid-based structure assumption completely, such that processing such data in full resolution with conventional approaches is infeasible by design. These problems become even more apparent when extending current data-driven approaches to handle higher-dimensional data. A solution is to learn from unstructured data directly. Recently, multiple attempts from the PointNet family [17,19,28] amongst others [10,14,25] proposed to handle *irregular* point clouds directly in a deep neural network. In contrast to the widely successful general purpose 2D network architectures, these methods propose very particular network architectures with an optimized design for very specific tasks. Also, these solutions only work on rather small point clouds, still lacking support for processing million-scale point cloud data. Methods from the PointNet family subsample their inputs to 4096 points per $1\,\mathrm{m}^2$ as depicted in Fig. 1. Such a low resolution enables single object classification, where the primary information is in the global shape characteristics [30]. Dense, complex 3D scenes, however, typically consist of millions of points [2,8]. Extending previous learning-based approaches to *effectively* process larger point clouds has been infeasible (Fig. 1(c)).

Inspired by commonly used CNNs architectures, we hypothesize that a simple convolution operation with a small amount of learnable parameters is advantageous when employing them in deeper network architectures—against recent trends of proposing complex layers for 3D point cloud processing.

To summarize our main contributions: (1) We introduce a novel convolution layer for arbitrary metric spaces, which represents a natural generalization of traditional grid-based convolution layers along (2) with a highly-tuned GPU-based implementation, providing significant speed-ups. (3) Our empirical evaluation demonstrates substantial improvements on large-scale point cloud segmentation [2] *without* any post-processing steps, and competitive results on small benchmark sets using fewer parameters and less memory.

2 Related Work

Recent literature dealing with learning from 3D point cloud data can be organized into three categories based on their way of dealing with the input data.

Voxel-based methods [16,18,20,30] discretize the point cloud into a voxel-grid enabling the application of classical convolution layers afterwards. However, this either loses spatial information during the discretization process or requires substantial computational resources for the 3D convolutions to avoid discretization artifacts. These approaches are affected by the curse of dimensionality and will be infeasible for higher-dimensional spaces. Interestingly, ensemble methods [22,26] based on classical CNNs still achieve state-of-the-art results on common benchmark sets like ModelNet40 [30] by rendering the 3D data from several viewing directions as image inputs. As the rendered views omit some information (*i.e.* occlusions) Cao *et al*. [4] propose to use a spherical projection.

Graph-based methods are geared to process social networks or knowledge graphs, particular instances of unstructured data where each node locations is solely defined by its relation to neighboring nodes in the absence of absolute position information. Recent research [13] proposes to utilize a sparse convolution for graph structures based on the adjacency matrix. This effectively masks the output of intermediate values in the classical convolution layers and mimics a diffusion process of information when applying several of these layers.

Euclidean Space-based methods deal directly with point cloud data featuring absolute position information but *without* explicit pair-wise relations. PointNet [17] is one of the first approaches yielding competitive results on ModelNet40. It projects each point *independently* into some learned features space, which then is transformed by a spatial transformer module [12] – a rather costly operation for higher feature dimensions. While the final aggregation of information is done effectively using a max-pooling operation, keeping all high dimensional features in memory beforehand is indispensable and becomes infeasible for larger point clouds by hardware restrictions. The lack of granularity during features aggregation from local areas is addressed by the extension PointNet++ [19] using "mini"-PointNets for each point neighborhood across different resolutions and later by [28]. An alternative way of introducing a structure in point clouds

relies on kD-trees [14], which allows to share convolution layers depending on the kD-tree splitting orientation. Such a structure is affected by the curse of dimensionality can only fuse point pairs in each hierarchy level. Further, defining splatting and slicing operations [25] has shown promising results on segmenting a facade datasets. Dynamic Edge-Condition Filters [23] learn parameters in the fashion of Dynamic Filter-Networks [7] for each single point neighborhood. Note, predicting a neighborhood-dependent filter can become quickly expensive for reasonably large input data. It is also noted by the authors, that tricks like BatchNorm are required during training.

Our approach belongs to the third category proposing a natural extension of convolution layers (see next section) for unstructured data which can be considered as a scalable special case of [7] but allows to evaluate point clouds and features more efficiently "in one go" – without the need of additional tricks.

3 Method

The basic operation in convolutional neural networks is a discrete 2D convolution, where the image signal[1] $I \in \mathbb{R}^{H \times W \times C}$ is convolved with a filter-kernel w. In deep learning a common choice of the filter size is $3 \times 3 \times C$ such that this mapping can be described as

$$(w \circledast f)[\ell] = \sum_{c \in C} \sum_{\tau \in \{-1,0,1\}^2} w_{c'}(c, \tau) f(c, \ell - \tau), \tag{1}$$

where $\tau \in \{-1, 0, 1\}^2$ describes the 8-neighborhood of ℓ in regular 2D grids. One usually omits the location information ℓ as it is given implicitly by arranging the feature values on a grid in a canonical way. Still, each pixel information is a *pair* of a feature/pixel value $f(c, \ell)$ and its location ℓ.

In this paper, we extend the convolution operation $\circledast$ to support irregular data with real-valued locations. In this case, the kernel w needs to support arbitrary relative positions $\ell_i - \tau_i$, which can be potentially unbounded. Before discussing such potential versions of w, we shortly recap the grid-based convolution layer in more detail to derive desired properties of a more generic convolution operation.

3.1 Convolution Layer

For a discrete $3 \times 3 \times C$ convolution layer such a filter mapping[2]

$$w_{c'} : C \times \{-1, 0, 1\}^2 \to \mathbb{R}, \quad (c, \tau) \mapsto w_{c'}(c, \tau) = \sum_{\tau' \in \{-1,0,1\}^2} 1_{\{\tau = \tau'\}} w_{c,c',\tau'} \tag{2}$$

[1] $c \in C$ represents the RGB, where we abuse notation and write C for $\{0, 1, \ldots, C - 1\} \subset \mathbb{N}$ as well.

[2] 1_M is the indicator function being 1 iff $M \neq \emptyset$.

is based on a lookup table with 9 entries for each (c, c') pair. These values $w_{c,c',\tau'}$ of the box-function $w_{c'}$ can be optimized for a specific task, *e.g.* using back-propagation when training CNNs. Typically, a single convolution layer has a filter bank of multiple filters. While these box functions are spatially invariant in ℓ, they have a bounded domain and are neither differentiable nor continuous wrt. τ by definition. Specifically, the 8-neighborhood in a 2D grid always has exactly the same underlying spatial layout. Hence, an implementation can exploit the implicitly given locations. The same is also true for other filter sizes $k_h \times k_w \times C$.

Processing irregular data requires a function $w_{c'}$, which can handle an *unbounded* domain of arbitrary—potentially real-valued—relations between τ and ℓ, besides retaining the ability to share parameters across different neighborhoods. To find potential candidates and identify the required properties, we consider a point cloud as a more generic data representation

$$P = \left\{ (\ell^{(i)}, f^{(i)}) \in L \times F \mid i = 0, 1, \ldots, n-1 \right\}. \tag{3}$$

Besides its value $f^{(i)}$, each point cloud element now carries an *explicitly* given location information $\ell^{(i)}$. In arbitrary metric spaces, *e.g.* Euclidean space $(\mathbb{R}^d, \|\cdot\|)$, $\ell^{(i)}$ can be real-valued without matching a discrete grid vertex. Indeed, one way to deal with this data structure is to *voxelize* a given location $\ell \in \mathbb{R}^d$ by mapping it to a specific grid vertex, *e.g.* $L' \subset \alpha\mathbb{N}^d, \alpha \in \mathbb{R}$. When L' resembles a grid structure, classical convolution layers can be used after such a discretization step. As already mentioned, choosing an appropriate α causes a trade-off between rather small cells for finer granularity in L' and consequently higher memory consumption.

Instead, we propose to define the notion of a convolution operation for a set of points in a local area. For any given point at location ℓ such a set is usually created by computing the k nearest neighbor points with locations $\mathcal{N}_k(\ell) = \{\ell'_0, \ell'_1, \ldots, \ell'_{k-1}\}$ for a point at ℓ, e.g. using a kD-tree. Thus, a generalization of Eq. (1) can be written as

$$f'(c', \ell^{(i)}) = \sum_{c \in C} \sum_{\ell' \in \mathcal{N}_k(\ell^{(i)})} \tilde{w}(c, \ell^{(i)}, \ell') \cdot f(c, \ell'). \tag{4}$$

Note, for point clouds describing an image Eq. (4) is equivalent[3] to Eq. (1). But for the more general case we require that

$$\tilde{w}_{c'} : C \times \mathbb{R}^d \times \mathbb{R}^d \to \mathbb{R}, \quad (c, \ell, \ell') \mapsto \tilde{w}(c, \ell, \ell') \tag{5}$$

is an *everywhere* well-defined function instead of a "simple" look-up table. This ensures, we can use $\tilde{w}$ in neighborhoods of arbitrary sizes. However, a side-effect of giving up the grid-assumption is that $\tilde{w}$ needs to be differentiable in both ℓ, ℓ' to perform back-propagation during training.

[3] By setting $\mathcal{N}_9(\ell) = \{\ell - \tau | \tau \in \{-1, 0, 1\}^d\}$ and $\tilde{w}_{c'}(c, \ell^{(i)}, \ell') = w_{c'}(c, \ell^{(i)} - \ell')$.

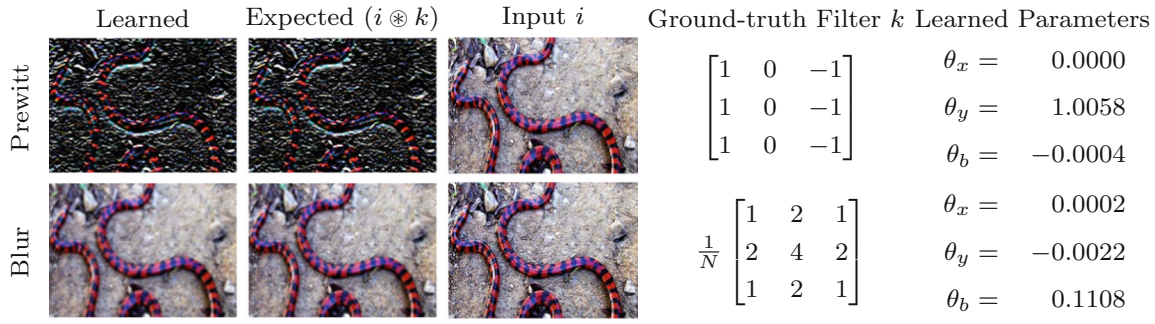


Fig. 2. Results on a toy dataset for illustration purposes. The special-case $w(x, y) = \theta_x(x - x_0) + \theta_y(y - y_0) + \theta_{b_c}$ of Eq. (6) is trained to re-produce the results of basic image operations like Prewitt or Blur.

While previous work [19,23] exert small neural networks for $\tilde{w}$ as a workaround inheriting all previously described issues, we rely on the given standard scalar product as the natural choice of $\tilde{w}$ in the Euclidean space with learnable parameters $\theta_c \in \mathbb{R}^d, \theta_{b_c} \in \mathbb{R}$:

$$\tilde{w}(c, \ell, \ell' \,|\, \theta_c, \theta_{b_c}) = \langle \theta_c, \ell - \ell' \rangle + \theta_{b_c}. \tag{6}$$

This formulation can be considered as a linear approximation of the lookup table, with the advantage of being defined everywhere. In a geometric interpretation $\tilde{w}$ is a learnable linear transformation (scaled and rotated) of a high-dimensional Prewitt operation. It can represent several image operations; Two are depicted in Fig. 2.

Hence, the mapping $\tilde{w}$ from Eq. (6) exists in all metric spaces, is *everywhere* well-defined in c, ℓ, ℓ', and continuously differentiable wrt. to *all* arguments, such that gradients can be propagated back even through the locations ℓ, ℓ'. Further, our rather simplistic formulation results in a significant reduction of the required trainable parameters and retains translation invariance. One observed consequence is a more stable training even *without* tricks like using BatchNorm as in [23]. This operation is parallel and can be implemented using CUDA to benefit from the sparse access patterns of local neighborhoods. In combination with a minimal memory footprint, this formulation is the first being able to process millions of irregular points simultaneously – a crucial requirement when applying this method in large-scale real-world settings. We experimented with slightly more complex versions of flex-conv, *e.g.* using multiple sets of parameters for one filter dependent on local structure. However, they did not lead to better results and induced unstable training.

3.2 Extending Sub-sampling to Irregular Data

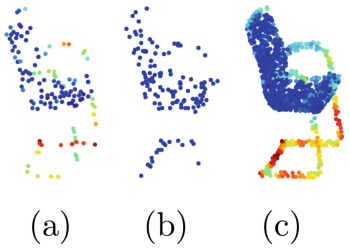


Fig. 3. IDISS (a) against random sub-sampling (b) for an object (c) with color-coded density.

While straightforward in grid-based methods, a proper and scalable sub-sampling operation in unstructured data is not canonically defined. On grids, down-sampling an input by a factor 4 is usually done by just taking every second cell in each dimension and aggregating information from a small surrounding region. There is always an implicitly well-defined connection between a point and its representative at a coarser resolution.

For sparse structures this property no longer holds. Points being neighbors in one resolution, potentially are not in each other's neighborhood at a finer resolution. Hence, it is even possible that some points will have no representative within the next coarser level. To avoid this issue, Simonovsky *et al.* [23] uses the VoxelGrid algorithm which inherits all voxel-based drawbacks described in the previous sections. Qi *et al.* [19] utilizes Farthest point sampling (FPS). While this produces sub-samplings avoiding the missing representative issue, it pays the price of having the complexity of $\mathcal{O}(n^2)$ for *each* down-sampling layer. This represents a serious computation limitation. Instead, we propose to utilize inverse density importance sub-sampling (IDISS). In our approach, the inverse density ϕ is simply approximated by adding up all distances from one point in ℓ to its k-neighbors by $\phi(\ell) = \sum_{\ell' \in \mathcal{N}_k(\ell)} \|\ell - \ell'\|$.

Sampling the point cloud proportional to this distribution has a computational complexity of $\mathcal{O}(n)$, and thereby enables processing million of points in a very efficient way. In most cases, this method is especially cheap regarding computation time, since the distances have already been computed to find the K-nearest neighbors. Compared to pure random sampling, it produces better uniformly distributed points at a coarser resolution and more likely preserves important areas. In addition, it still includes randomness that is preferred in training of deep neural networks to better prevent against over-fitting. Figure 3 demonstrates this approach. Note, how the chair legs are rarely existing in a randomly sub-sampled version, while IDISS preserves the overall structure.

4 Implementation

To enable building complete DNNs with the presented flex-convolution model we have implemented two specific layers in TensorFlow: *flex-convolution* and *flex-max-pooling*. Profiling shows that a direct highly hand-tuned implementation in CUDA leads to a run-time which is in the range of regular convolution layers (based on cuDNN) during inference.

4.1 Neighborhood Processing

Both new layers require a known neighborhood for each incoming point. For a fixed set of points, this neighborhood is computed once upfront based on an efficient kD-tree implementation and kept fixed. For each point, the k nearest neighbors are stored as indices into the point list. The set of indices is represented as a tensor and handed over to each layer.

The *flex-convolution* layer merely implements the convolution with continuous locations as described in Eq. (6). Access to the neighbors follows the neighbor indices to lookup their specific feature vectors and location. No data duplication is necessary. As all points have the same number of neighbors, this step can be parallelized efficiently. In order to make the position of each point available in each layer of the network, we attach the point location ℓ to each feature vector.

The *flex-max-pooling* layer implements max-pooling over each point neighborhood individually, just like the grid-based version but without subsampling.

For subsampling, we exploit the IDISS approach described in Sect. 3.2. Hereby, flex-max-pooling is applied before the subsampling procedure. For the subsequent, subsampled layers the neighborhoods might have changed, as they only include the subsampled points. As the point set is static and known beforehand, all neighborhood indices at each resolution can be computed on-the-fly during parallel data pre-fetching, which is neglectable compared to the cost of a network forward+backward pass under optimal GPU utilization.

Upsampling (*flex-upsampling*) is done by copying the features of the selected points into the larger-sized layer, initializing all other points with zero, like zero-padding in images and performing the flex-max-pooling operation.

Table 1. Profiling information of diverse implementations with 8 batch of 4096 points with $C' = C = 64$ and 9 neighbors using a CUDA profiler.

Method	Timing		Memory	
	Forward	Backward	Forward	Backward
flex-convolution (pure TF)*	1829 ms	2738 ms	34015.2 MB	63270.8 MB
flex-convolution (Ours)	24 ms	265 ms	8.4 MB	8.7 MB
flex-convolution (TC [27])	42 ms	-	8.4 MB	-
grid-based conv. (cuDNN)	16 ms	1.5 ms	1574.1 MB	153.4 MB
flex-max-pooling (Ours)	1.44 ms	15 us	16.78 MB	8.4 MB

4.2 Efficient Implementation of Layer Primitives

To ensure a reasonably fast training time, highly efficient GPU-implementations of flex-convolution and flex-max-pooling as a custom operation in TensorFlow are required. We implemented a generic but hand-tuned CUDA operation, to ensure optimal GPU-throughput. Table 1 compares our optimized CUDA kernel against a version (pure TF) containing exclusively existing operations

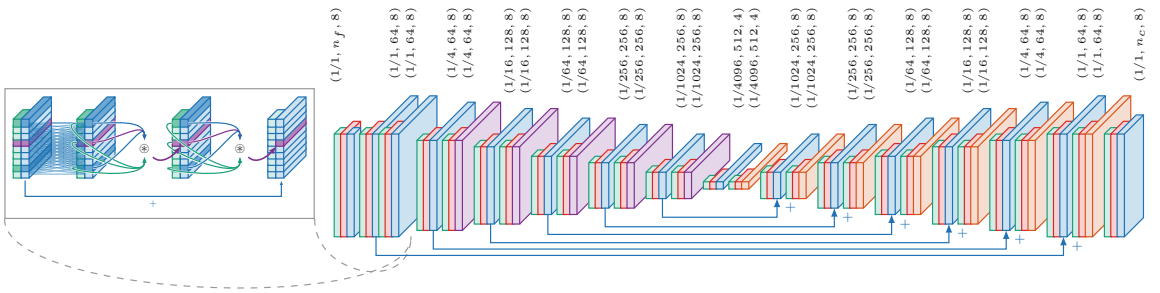


Fig. 4. Network architecture for semantic 3D point cloud segmentation. The annotations (a, d_f, k) represent the spatial resolution factor a (*i.e.* using $a \cdot n$ points) and feature length d_f with n_f input features and n_c classes. The used neighborhood size is given by k. In each step, the position information ▫ and neighborhood information ▫ is required besides the actual learned features. After flex-convolution layers ▫ , each downsampling step ▫ (flex-max-pool) has a skip-connection to the corresponding decoder block with flex-upsampling layer ▫.

provided by the TensorFlow framework itself and its grid-based counterpart in cuDNN [6] using the CUDA profiler for a *single* flex-convolution layer on a set of parameters, which fits typical consumer hardware (Nvidia GTX 1080Ti). As the grid-based convolution layer typically uses a kernel-size of $3 \times 3 \times C$ in the image domain, we set $k = 9$ as well – though we use $k = 8$ in all subsequent point cloud experiments. We did some experiments with a quite recent polyhedral compiler optimization using TensorComprehension (TC) [27] to automatically tune a flex-convolution layer implementation. While this approach seems promising, the lack of supporting flexible input sizes and slower performance currently prevents us from using these automatically generated CUDA kernels in practice.

An implementation of the flex-convolution layer by just relying on operations provided by the TensorFlow framework requires data duplication. We had to spread the pure TensorFlow version across 8 GPUs to run a *single* flex-convolution layer. Typical networks usually consist of several such operations. Hence, it is inevitable to recourse on tuning custom implementations when applying such a technique to larger datasets. Table 1 reveals that the grid-based version (cuDNN) prepares intermediate values in the forward pass resulting in larger memory consumption and faster back-propagation pass—similar to our flex-max-pooling.

4.3 Network Architecture for Large-Scale Semantic Segmentation

With the new layers at hand, we can directly transfer the structure of existing image processing networks to the task of processing large point clouds. We will elaborate on our network design and choice of parameters for the task of semantic point cloud segmentation in more detail. Here, we draw inspiration from established hyper-parameter choices in 2D image processing.

Our network architecture follows the SegNet-Basic network [3] (a 2D counterpart for semantic image segmentation) with added U-net skip-connections [21]. It has a typical encoder-decoder network structure followed by a final point-wise soft-max classification layer. To not obscure the effect of the flex-convolution layer behind several other effects, we explicitly do *not* use tricks like Batch-Normalization, weighted soft-max classification, or computational expensive pre- resp. post-processing approaches, which are known to enhance the prediction quality and could further be applied to the results presented in the Sect. 5.

The used architecture and output sizes are given in Fig. 4. The encoder network is divided into six stages of different spatial resolutions to process multi-scale information from the input point cloud. Each resolution stage consists of two ResNet-blocks. Such a ResNet block chains the following operations: 1×1-convolution, flex-convolution, flex-convolution (compare Fig. 4). Herewith, the output of the last flex-convolution layer is added to the incoming feature following the common practice of Residual Networks [9]. To decrease the point cloud resolution across different stages, we add a flex-max-pooling operation with subsampling as the final layer in each stage of the encoder. While a grid-based max-pooling is normally done with stride 2 in x/y dimension, we use the flex-max-pooling layer to reduce the resolution n by factor 4. When the spatial resolution decreases, we increase the feature-length by factor two.

Moreover, we experimented with different neighborhood sizes k for the flex-convolution layers. Due to speed considerations and the widespread adoption of 3×3 filter kernels in image processing we stick to a maximal nearest neighborhood size of $k = 8$ in all flex-convolution layers. We observed no decrease in accuracy against $k = 16$ but a drop in speed by factor 2.2 for 2D-3D-S [2].

The decoder network mirrors the encoder architecture. We add skip connections [21] from each stage in the encoder to its related layer in the decoder. Increasing spatial resolution at the end of each stage is done via flex-upsampling. We tested a trainable flex-transposed-convolution layer in some preliminary experiments and observed no significant improvements. Since pooling irregular data is more light-weight (see Table 1) regarding computation effort, we prefer this operation. As this is the first network being able to process point clouds in such a large-scale setting, we expect choosing more appropriate hyper-parameters is possible when investing more computation time.

5 Experiments

We conducted several experiments to validate our approach. These show that our flex-convolution-based neural network yields competitive performance to previous work on synthetic data for single object classification ([30], 1024 points) using fewer resources and provide some insights about human performance on this dataset. We improve single instance part segmentation ([31], 2048 points). Furthermore, we demonstrate the effectiveness of our approach by performing semantic point cloud segmentation on a large-scale real-world 3D scan ([2], 270 Mio. points) improving previous methods in both accuracy and speed.

5.1 Synthetic Data

To evaluate the effectiveness of our approach, we participate in two benchmarks that arise from the ShapeNet [5] dataset, which consists of synthetic 3D models created by digital artists.

Table 2. Classification accuracy on ModelNet40 (1024 points) and 256 points*.

Method	Accuracy	#params.
PointNet [17]	89.2	1'622'705
PointNet2 [19]	90.7	1'658'120
KD-Net [14]	90.6	4'741'960
D-FilterNet [23]	87.4	345'288
Human	64.0	-
Ours	90.2	346'409
Ours (1/4)	89.3	171'048

ModelNet40 [30] is a single object classification task of 40 categories. We applied a smaller version of the previously described encoder network-part followed by a fully-connected layer and a classification layer. Following the official test-split [17] of randomly sampled points from the object surfaces for object classification, we compare our results in Table 2. Our predictions are provided from by a single forward-pass in contrast to a voting procedure as in the KD-Net [14]. This demonstrates that a small flex-convolution neural network with significant fewer parameters provides competitive results on this benchmark set. Even when using just 1/4th of the point cloud and thus an even smaller network the accuracy remains competitive. To put these values in a context to human perception, we conducted a user study asking participants to classify point clouds sampled from the official test split. We allowed them to rotate the presented point cloud for the task of classification without a time limit. Averaging all 2682 gathered object classification votes from humans reveals some difficulties with this dataset. This might be related to the relatively unconventional choice of categories in the dataset, *i.e.* plants and their flower pots and bowls are sometimes impossible to separate. Please refer to the Supplementary for a screenshot of the user study, a confusion matrix, saliency maps and an illustration of label ambiguity.

ShapeNet Part Segmentation [31] is a semantic segmentation task with per-point annotations of 31963 models separated into 16 shape categories. We applied a smaller version of the previously described segmentation network that receives the (x, y, z) position of 2048 points per object. For the evaluation, we follow the procedure of [25] by training a network for per category. Table 3 contains a comparison of methods using only point cloud data as input. Our method demonstrates an improvement of the average mIoU while being able to process a magnitude more shapes per second. Examples of ShapeNet part segmentation are illustrated in Fig. 5. These experiments on rather small synthetic data confirm our hypothesis that even in three dimensions simple filters with a small amount of learnable parameters are sufficient in combination with deeper network architectures. This matches with the findings that are known from typical CNN architectures of preferring deeper networks with small 3×3 filters. The resulting smaller memory footprint and faster computation time enable processing more points in reasonable time. We agree with [25,28] on the data labeling issues.

Table 3. ShapeNet part segmentation results per category and mIoU (%) for different methods and inference speed (on a Nvidia GeForce GTX 1080 Ti).

	Airpl.	Bag	Cap	Car	Chair	Earph.	Guitar	Knife	Lamp	Laptop	Motorb.	Mug	Pistol	Rocket	Skateb.	Table	mIoU	shapes/ sec
Kd-Network [14]	80.1	74.6	74.3	70.3	88.6	73.5	90.2	**87.2**	81.0	94.9	57.4	86.7	78.1	51.8	69.9	80.3	77.4	n.a.
PointNet [17]	83.4	78.7	82.5	74.9	89.6	73.0	91.5	85.9	80.8	95.3	65.2	93.0	81.2	57.9	72.8	80.6	80.4	n.a.
PointNet++ [19]	82.4	79.0	87.7	77.3	**90.8**	71.8	91.0	85.9	83.7	95.3	71.6	94.1	81.3	58.7	76.4	82.6	81.9	2.7
SPLATNet3D [25]	81.9	83.9	88.6	**79.5**	90.1	73.5	91.3	84.7	**84.5**	96.3	69.7	95.0	81.7	59.2	70.4	81.3	82.0	9.4
SGPN [28]	80.4	78.6	78.8	71.5	88.6	**78.0**	90.9	83.0	78.8	95.8	**77.8**	93.8	**87.4**	60.1	**92.3**	**89.4**	82.8	n.a.
Ours	**83.6**	**91.2**	**96.7**	**79.5**	84.7	71.7	**92.0**	86.5	83.2	**96.6**	71.7	**95.7**	86.1	**74.8**	81.4	84.5	**85.0**	**489.3**

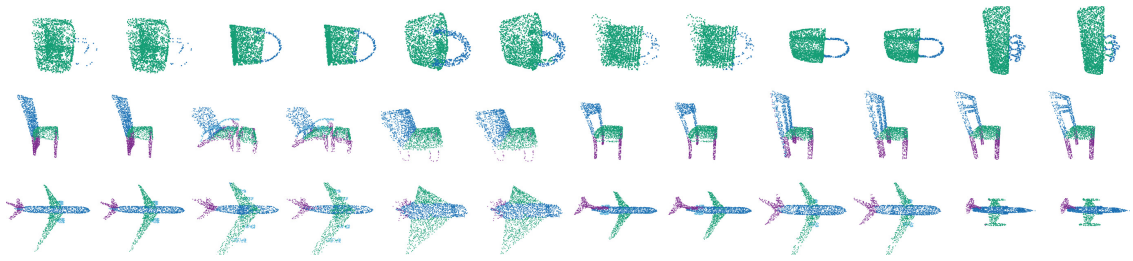


Fig. 5. Our semantic segmentation results on ShapeNet (ground-truth (left), prediction (right)) pairs. Please refer to the supplementary for more results at higher resolution.

5.2 Real-World Semantic Point Cloud Segmentation

To challenge our methods at scale, we applied the described network from Sect. 4.3 to the 2D-3D-S dataset [2]. This real-world dataset covers 3D scanning information from six square kilometers of several building complexes collected by a Matterport Camera. Previous approaches are based on sliding windows, either utilizing hand-crafted feature, *e.g.* local curvature, occupancy and point density information per voxel [1,2] or process small sub-sampled chunks PointNet [17], SGPN [28] (4096 points, Fig. 1). We argue, that a neural network as described in Sect. 4.3 can learn all necessary features directly from the data – just like in the 2D case and at *full* resolution.

An ablation study on a typical room reveals the effect of different input features f. Besides neighborhood information, providing only constant initial features $f = 1$ yields 0.31 mAP. Hence, this is already enough information to perform successful semantic segmentation. To account for the irregularity in the data, it is however useful to use normalized position data $f = (1, x, y, z)$ besides the color information $f = (1, x, y, z, r, g, b)$ which increases the accuracy to 0.39 mAP resp. 0.50 mAP. Our raw network predictions from a single inference forward pass out-performs previous approaches given the same available information and approaches using additional input information but lacks precision in categories like beam, column, and door, see Table 4. Providing features like local curvature besides post-processing [2] greatly simplify detecting these kinds of objects. Note, our processing of point clouds at full resolution benefits the handling of smaller objects like chair, sofa and table.

Consider Fig. 6, the highlighted window region in room A is classified as wall because the blinds are closed, thus having a similar appearance. In room B, our network miss-classifies the highlighted column as "wall", which is not surprising as both share similar geometry and color. Interestingly, in room C our network classifies the beanbag as "sofa", while its ground-truth annotation is "chair". For more results please refer to the accompanying video.

Training is done on two Nvidia GTX 1080Ti with batch-size 16 for two days on point cloud chunk with 128^2 points using the Adam-Optimizer with learning-rate $3 \cdot 10^{-3}$.

To benchmark inference, we compared ours against the author's implementations of previous work [17,19,28] on different point clouds sizes n. Memory requirements limits the number of processed points to at most 131k [19], 500k [28], 1Mio [17] points (highlighted region in Fig. 1). We failed to get meaningful performance in terms of accuracy from these approaches when increasing $n > 4096$. In contrast, ours – based on a fully convolutional network – can process up to 7 Mio. points concurrently providing the *same* performance during inference within 4.7 s. Note, [17] can at most process 1 Mio. points within 7.1 s. Figure 1 further reveals an exponential increase of runtime for the PointNet family [17,19,28], ours provides significant faster inference and shows better utilization for larger point clouds with a linear increase of runtime.

Table 4. Class specific average precision (AP) on the 2D-3D-S dataset. (‡) uses additional input features like local curvature, point densities, surface normals. (*) uses non-trivial post-processing and (**) a mean filter post-processing.

	Table	Chair	Sofa	Bookc.	Board	Ceiling	Floor	Wall	Beam	Col.	Wind.	Door	mAP
Armeni *et al.* [2]*	46.02	16.15	6.78	54.71	3.91	71.61	88.70	72.86	66.67	**91.77**	25.92	54.11	49.93
Armeni *et al.* [2]‡	39.87	11.43	4.91	**57.76**	3.73	50.74	80.48	65.59	68.53	85.08	21.17	45.39	44.19
PointNet [17]*	46.67	33.80	4.76	n.a.	11.72	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.	n.a.
SGPN [28]*	46.90	40.77	6.38	47.61	11.05	79.44	66.29	**88.77**	**77.98**	60.71	**66.62**	**56.75**	54.35
Ours	66.03	51.75	15.59	39.03	43.50	87.20	96.00	65.53	54.76	52.74	55.34	35.81	55.27
Ours**	**67.02**	**52.75**	**16.61**	39.26	**47.68**	**87.33**	**96.10**	65.52	56.83	55.10	57.66	36.76	**56.55**

Limitation. As we focus on static point cloud scans ours is subject to the same limitations as [17,19,25,28], where neighborhoods are computed during parallel data pre-fetching. Handling dynamic point clouds, *e.g.* completion or generation, requires an approximate nearest-neighborhood layer. Our prototype implementation suggests this could be done *within* the network. For 2 Million points it takes around 1 s which is still faster by a factor of 8 compared to the used kd-Tree, which however has neglectable costs being part of parallel pre-fetching.

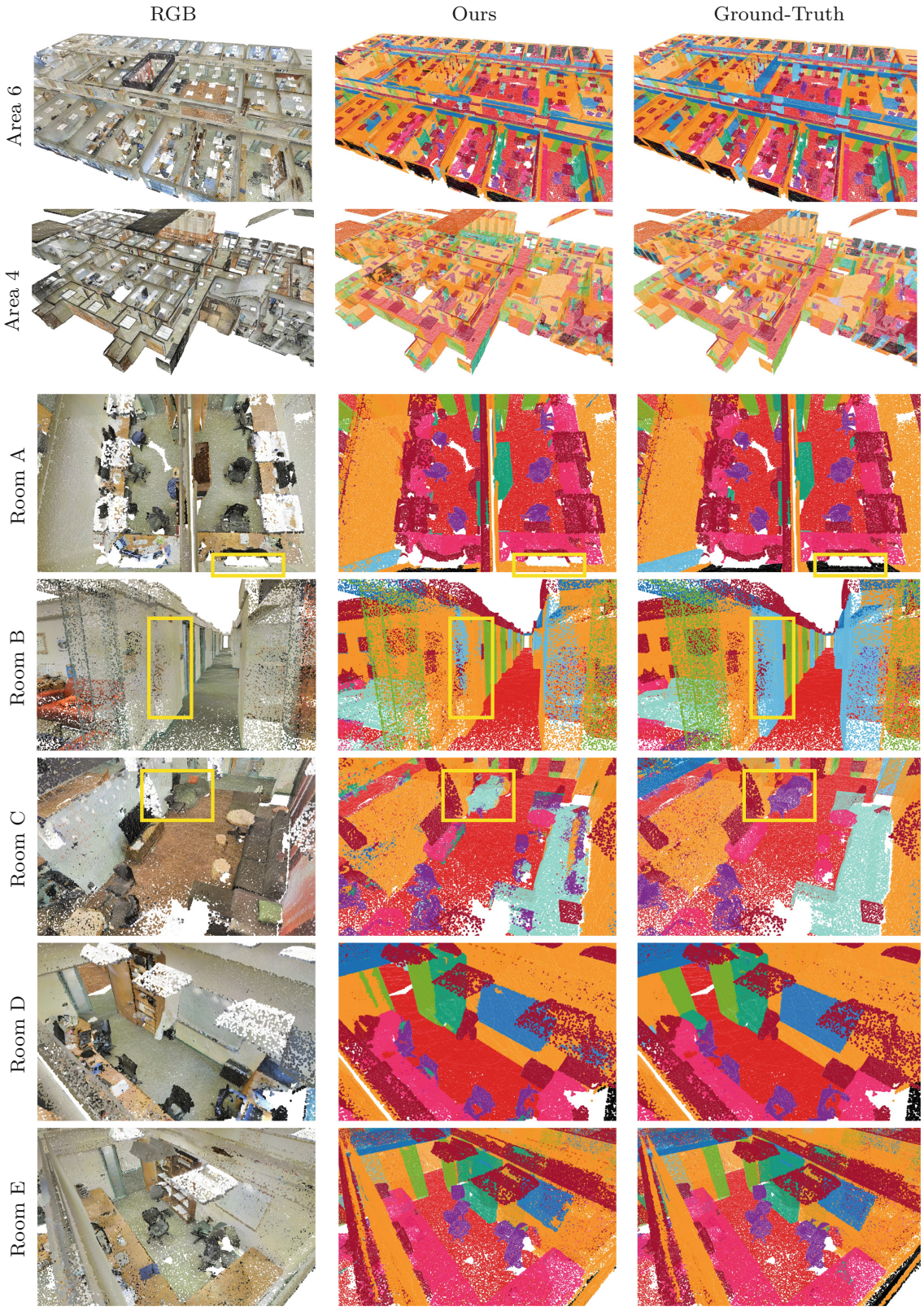


Fig. 6. Semantic point cloud segmentation produced as raw outputs of our proposed network from the held-out validation set. In this point-based rendering, surfaces might not be illustrated as opaque.

6 Conclusion

We introduced a novel and natural extension to the traditional convolution, transposed convolution and max-pooling primitives for processing irregular point sets. The novel sparse operations work on the local neighborhood of each point, which is provided by indices to the k nearest neighbors. Compared to 3D CNNs our approach can be extended to support even high-dimensional point sets easily. As the introduced layers behave very similar to convolution layers in networks designed for 2D image processing, we can leverage the full potential of already successful architectures. This is against recent trends in point cloud processing with highly specialized architectures which sometimes rely on hand-crafted input features, or heavy pre- and post-processing. We demonstrate state-of-the-art results on small synthetic data as well as large real-world datasets while processing millions of points concurrently and efficiently.

Acknowledgment. This work was supported by the German Research Foundation (DFG): SFB 1233, Robust Vision: Inference Principles and Neural Mechanisms, TP 01 & 02.

References

1. Armeni, I., Sax, A., Zamir, A.R., Savarese, S.: Joint 2D-3D-semantic data for indoor scene understanding. arXiv e-prints, February 2017
2. Armeni, I., et al.: 3D semantic parsing of large-scale indoor spaces. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
3. Badrinarayanan, V., Kendall, A., Cipolla, R.: SegNet: a deep convolutional encoder-decoder architecture for image segmentation. IEEE Trans. Pattern Anal. Mach. Intell. (PAMI) **39**(12), 2481–2495 (2017)
4. Cao, Z., Huang, Q., Karthik, R.: 3D object classification via spherical projections. In: International Conference on 3D Vision (3DV), pp. 566–574. IEEE (2017)
5. Chang, A.X., et al.: ShapeNet: an information-rich 3D model repository. Technical report arXiv:1512.03012 [cs.GR], Stanford University — Princeton University — Toyota Technological Institute at Chicago (2015)
6. Chetlur, S., et al.: cuDNN: efficient primitives for deep learning. CoRR (2014)
7. De Brabandere, B., Jia, X., Tuytelaars, T., Van Gool, L.: Dynamic filter networks. In: Advances in Neural Information Processing Systems (NIPS) (2016)
8. Groh, F., Resch, B., Lensch, H.P.A.: Multi-view continuous structured light scanning. In: Roth, V., Vetter, T. (eds.) GCPR 2017. LNCS, vol. 10496, pp. 377–388. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-66709-6_30
9. He, K., Zhang, X., Ren, S., Sun, J.: Identity mappings in deep residual networks. In: Leibe, B., Matas, J., Sebe, N., Welling, M. (eds.) ECCV 2016. LNCS, vol. 9908, pp. 630–645. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46493-0_38
10. Hermosilla, P., Ritschel, T., Vázquez, P.P., Vinacua, À., Ropinski, T.: Monte Carlo convolution for learning on non-uniformly sampled point clouds. arXiv preprint arXiv:1806.01759 (2018)
11. Hershey, S., et al.: CNN architectures for large-scale audio classification. In: IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), pp. 131–135. IEEE (2017)

12. Jaderberg, M., Simonyan, K., Zisserman, A., Kavukcuoglu, K.: Spatial transformer networks. In: Cortes, C., Lawrence, N.D., Lee, D.D., Sugiyama, M., Garnett, R. (eds.) Advances in Neural Information Processing Systems (NIPS), pp. 2017–2025. Curran Associates, Inc., Red Hook (2015)
13. Kipf, T.N., Welling, M.: Semi-supervised classification with graph convolutional networks. In: International Conference on Learning Representations (ICLR) (2017)
14. Klokov, R., Lempitsky, V.: Escape from cells: deep Kd-networks for the recognition of 3D point cloud models. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV), pp. 863–872, October 2017
15. Lavin, A., Gray, S.: Fast algorithms for convolutional neural networks. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 4013–4021 (2016)
16. Maturana, D., Scherer, S.: VoxNet: a 3D convolutional neural network for real-time object recognition. In: International Conference on Intelligent Robots and Systems (2015)
17. Qi, C.R., Su, H., Mo, K., Guibas, L.J.: PointNet: deep learning on point sets for 3D classification and segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017)
18. Qi, C.R., Su, H., Niessner, M., Dai, A., Yan, M., Guibas, L.J.: Volumetric and multi-view CNNs for object classification on 3D data. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2016)
19. Qi, C.R., Yi, L., Su, H., Guibas, L.J.: PointNet++: deep hierarchical feature learning on point sets in a metric space. In: Guyon, I., et al. (eds.) Advances in Neural Information Processing Systems (NIPS), pp. 5099–5108. Curran Associates, Inc., Red Hook (2017)
20. Riegler, G., Ulusoy, A.O., Bischof, H., Geiger, A.: OctNetFusion: learning depth fusion from data. In: International Conference on 3D Vision (3DV), October 2017
21. Ronneberger, O., Fischer, P., Brox, T.: U-Net: convolutional networks for biomedical image segmentation. In: Navab, N., Hornegger, J., Wells, W.M., Frangi, A.F. (eds.) MICCAI 2015. LNCS, vol. 9351, pp. 234–241. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24574-4_28
22. Sfikas, K., Pratikakis, I., Theoharis, T.: Ensemble of PANORAMA-based convolutional neural networks for 3D model classification and retrieval. Comput. Graph. **71**, 208–218 (2017)
23. Simonovsky, M., Komodakis, N.: Dynamic edge-conditioned filters in convolutional neural networks on graphs. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) (2017). https://arxiv.org/abs/1704.02901
24. Simonyan, K., Zisserman, A.: Very deep convolutional networks for large-scale image recognition. CoRR (2014)
25. Su, H., et al.: SPLATNet: sparse lattice networks for point cloud processing. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2530–2539 (2018)
26. Su, H., Maji, S., Kalogerakis, E., Learned-Miller, E.G.: Multi-view convolutional neural networks for 3D shape recognition. In: Proceedings of the IEEE International Conference on Computer Vision (ICCV) (2015)
27. Vasilache, N., et al.: Tensor comprehensions: framework-agnostic high-performance machine learning abstractions (2018)
28. Wang, W., Yu, R., Huang, Q., Neumann, U.: SGPN: similarity group proposal network for 3D point cloud instance segmentation. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 2569–2578 (2018)

29. Wieschollek, P., Schölkopf, M.H.B., Lensch, H.P.A.: Learning blind motion deblurring. In: International Conference on Computer Vision (ICCV), October 2017
30. Wu, Z., et al.: 3D ShapeNets: a deep representation for volumetric shapes. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), pp. 1912–1920 (2015)
31. Yi, L., et al.: A scalable active framework for region annotation in 3D shape collections. ACM Trans. Graph. (SIGGRAPH ASIA) **35**(6), 210 (2016)

C

FISHERVECTOR

Backpropagation training for fisher vectors within neural networks

Patrick Wieschollek*, Fabian Groh*, and Hendrik P.A. Lensch

arXiv preprint arXiv:1702.02549, 2017 [3]

** shared first authorship.*

Backpropagation Training for Fisher Vectors within Neural Networks

Patrick Wieschollek* Fabian Groh* Hendrik P.A. Lensch
University of Tübingen

Abstract

Fisher-Vectors (FV) encode higher-order statistics of a set of multiple local descriptors like SIFT features. They already show good performance in combination with shallow learning architectures on visual recognitions tasks. Current methods using FV as a feature descriptor in deep architectures assume that all original input features are static. We propose a framework to jointly learn the representation of original features, FV parameters and parameters of the classifier in the style of traditional neural networks. Our proof of concept implementation improves the performance of FV on the Pascal Voc 2007 challenge in a multi-GPU setting in comparison to a default SVM setting. We demonstrate that FV can be embedded into neural networks at arbitrary positions, allowing end-to-end training with backpropagation.

1. Introduction

Many fundamental computer-vision problems rely on extracting multiple meaningful local *rigid* descriptors like SIFT [12], GIST [13] or SURF [1] features from a single RGB image. Classifying a combination of these features with methods such as Multi-Layer-Perceptron (MPL) or linear Support-Vector-Machines (SVM) often result in good performance [3].

Local features as a compressed image representation are an important factor in visual recognition tasks. Methods like deep *neural networks* (NN) are widely used to *learn* good feature representations without hand-engineering effort. Applied to datasets of large amounts of labeled examples they define state-of-the art results in various computer vision tasks and even surpass human performance [4].

Enriched features, which additionally encode higher-order statistics of the underlying distribution, further improve the classification results. Prominent examples are VLAD [9] and Fisher-Vectors (FV) [14], where the latter is based on a Gaussian-Mixture-Model (GMM) fitted to the data and encodes information relative to each Gaussian component.

*Indicates equal contribution.

The combination of (convolutional) NN and FV using shallow architectures recently became popular [24, 11, 5, 3, 2]. In a nutshell, these methods approach a minimization of the empirical risk R_{emp}, depending on static input features $x \in \mathbb{R}^D$, a feature transformation $f_W(\cdot)$ with parameters $W \in \mathbb{R}^{D\times D}$, a kernel F with parameter $\xi \in \mathbb{R}^{D'}$ and classifier parameters $\theta \in \mathbb{R}^{D'}$ solving

$$(W^\star, \xi^\star, \theta^\star) := \arg\min_{W,\xi,\theta} R_{\text{emp}}(F_\xi(f_W(x)), \theta). \quad (1)$$

All previous methods perform a greedy-wise optimization of these parameters one after another. Usually these steps are: Training neural network parameters $\hat{W}$ for feature extraction on some loss functions, learning GMM components $\hat{\xi}$ on these fixed features $f_{\hat{W}}(x)$ using *expectation-maximization* and finally optimizing a linear SVM to obtain $\hat{\theta}$. Empirical results of this sequential approach already improves performance compared NN. However, it seems reasonable to share information between these optimization steps in the fashion of deep neural networks to *jointly* solve problem (1) in W, ξ, θ.

Therefore, we propose a neural-network-like batch-wised back-propagation training for optimizing W, ξ, θ together, with a strong theoretical support of [23]. Unfortunately, directly tackling Eq. (1) in a joint optimization approach comes at the price of handling a huge amount of input data. A single image is described by T SIFT features, $T > 8 \cdot 10^4$. For 5k Images from the Pascal Voc 2007 data set this results in approx. 204GB, in contrast to a single 4128-dimensional FV per image. Fortunately, exploiting multi-GPU and sampling approaches makes it possible to compute all necessary parameter update rules in reasonable time.

Our main contributions in this paper can be summarized as follows:

- The proposed architecture includes FV, GMM, and normalization as neural network modules, which enables end-to-end learning in the fashion of classical neural networks.
- We provide the first multi-GPU accelerated implementation for FV computation in large-scale classification tasks.

– We introduce feature learning from FV classification including all back-propagation rules to update feature representation.
– The proposed method includes a re-formulation of supervised GMM parameters learning which satisfies GMM constraints naturally using batches.

The remainder of this paper is organized as follows: After delimiting our work from related methods in this field in Section 2, we describe the Fisher-Vector encoding that is used for learning features in Section 3. Section 4 contains details for the back-propagation pass and the learning environment. We evaluate the proposed method in Section 5 and provide concluding remarks in the last Section 6.

2. Related Work

The interest of re-using activation values of NN-layers as mid-level features for training additional classifiers in combination with Fisher-Vector and VLAD became popular in recent work. Even for large-scale problems, methods like sparse Fisher-Vector-Coding [11] yield state-of-the-art results in generic object recognition and scene classification [5]. Thereby, computing a FV is usually done as an independent step decoupled from the feature learning process. A kind of stacking of Fisher-Vector layer in deep neural networks was applied to the ILSVRC-2010 challenge with impressive results in [20]. Still, their training method is a greedy layer-by-layer training without back-propagation training steps, which disconnects FV from the already trained layers in the network. Another work considering Fisher-Vectors as a pre-processing step followed by dimensionality reduction methods and a multi-layer-perceptron was recently discussed in [16], again without back propagating updates. In addition, there is no reason for extracted mid-level features from a neural network being the best choice in a completely different classification method. The goal of sharing gradient information between the classifier and Fisher-Vector parameters was first addressed in [21] by adapting the underlying GMM parameters from the classifier loss information. Their algorithm samples GMM parameter gradients from all inputs. To guarantee a decreasing loss they determine the optimal update by line-search. From the computational aspect their approach is limited to learn the Fisher-Kernel only due to the enormous amount of gradient data, which has to be calculated.

Armed with the knowledge of the classifier loss, one might ask how the points of the data set should move to allow the classifier to better separate classes. Common approaches do not incorporate this information shared between the feature learning stage of original feature representation and the classifier. Although it is not possible to shift data points directly, this mapping can be approximated by training $f_W(\cdot)$ in the feature learning stage. This gradient propagation backwards through the FV layer closes the gap between the current *one-direction* usage of FV and deep neural networks. However, any back-propagation through the FV layer ends up with a non-trivial mapping into more than half a million elements. Currently, we are only aware of batch-wise methods to tackle this issue.

Using current methods comes along with several additional drawbacks like the requirement of multiple independent pipelines and limiting the solution space when optimizing (1) and discarding any relevant information like $\nabla\theta$ from the classifier.

Applying these methods to normalized inputs falls into the class of learning methods, where a SVM seeks for the optimal separation hyperplane reducing the theoretical upper-bound for the expected risk R_{ex}, which depends on the radius of the sphere containing all data points (see Theorem 2.1 in [23]). Our work builds on previous attempts without violating this theorem and, thereby, obtaining a strong theoretical basis.

Overall, it is preferable to fully embed FV *within* deep architectures to enable deep *end-to-end* learning approaches without decoupled stages as illustrated in Figure 1 for optimizing the complete set of parameters simultaneously.

3. Background

Before deriving the update rules for the Fisher-Vector layer, we shortly recap the analysis and motivation of Fisher-Vectors to be self contained. A comprehensive essay and more details can be found in [22].

Fisher Vectors Let $\mathcal{X}$ be a set of features

$$\mathcal{X} = \{x_1, x_2, \ldots, x_T\}, \quad x_j \in \mathbb{R}^D$$

for a single image. In computer vision tasks $\mathcal{X}$ typically represents local image descriptors, e.g., 128-dimensional SIFT features. We denote the probability density function as $c_\lambda(\cdot)$ which corresponds to the observations $\mathcal{X}$. The observation $\mathcal{X}$ implies a score function

$$G_\lambda^{\mathcal{X}} = \nabla_\lambda \ln c_\lambda(\mathcal{X}),$$

with dimension only depending on the number of parameters, not the size of the sample. Let F_λ be the Fisher information matrix of c_λ, a natural kernel (see [8]) for measuring the similarity between two samples X, Y is given by

$$K(X,Y) = \left(G_\lambda^X\right)^T F_\lambda^{-1} G_\lambda^Y,$$
$$F_\lambda = \mathbb{E}_{x\sim c_\lambda}\left[\nabla_\lambda \ln c_\lambda(x) \left(\nabla_\lambda \ln c_\lambda(x)\right)^T\right].$$

Learning a classifier with kernel $K(X,Y)$ is equivalent to learning a linear classifier on the Fisher Vectors $\mathcal{F}_\lambda^X =$

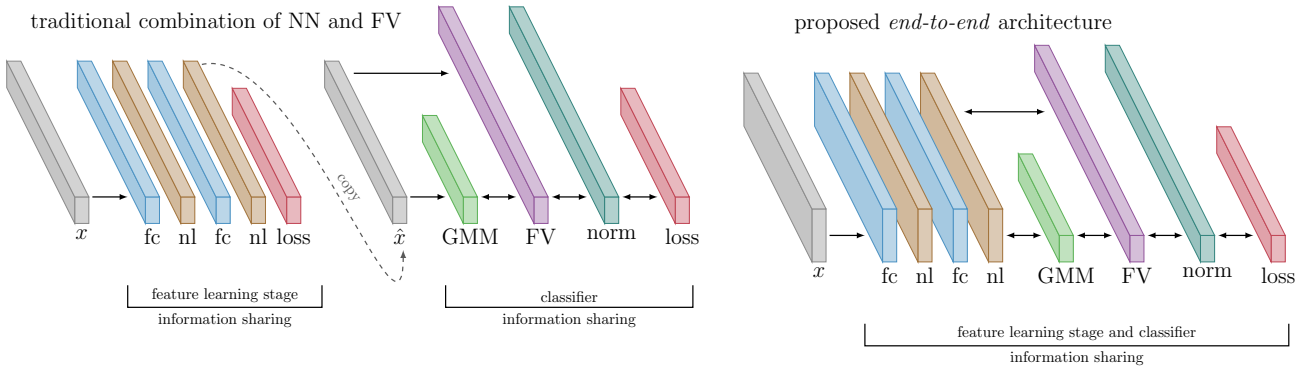


Figure 1: Difference between the traditional combination of deep neural networks (left) with fully-connected layers (fc), non-linearities (nl), normalization (norm) and FV compared to our proposed combination (right). Our approach fully integrates the GMM and FV layer enabling end-to-end with batched back-propagation.

$L_\lambda G_\lambda^X$, where $F_\lambda^{-1} = L_\lambda^T L_\lambda$ is the Cholesky decomposition of F_λ^{-1}. Although, a natural choice is c_λ to be a Gaussian-Mixture-Model (GMM) as a weighted sum of K Gaussians, an alternative is a hybrid Gaussian-Laplacian [10].

Assuming diagonal covariance matrices $\Sigma_k = \mathrm{diag}(\sigma_k^2)$, $0 < \sigma_k^2 \in \mathbb{R}^D$ for efficient computations, learning a GMM with K components gives $(2D+1)K$ tunable parameters

$$\lambda_1 \ldots, \lambda_K, \sigma_1, \ldots, \sigma_K \in \mathbb{R}^D, \mu_1, \ldots, \mu_K \in \mathbb{R}^D,$$

where λ_k denotes the prior probability of Gaussian $\mathcal{N}(\cdot; \mu_k, \Sigma_k)$. For practical purposes all necessary computations are done in the log-domain using the logarithm of the multivariate normal distribution. For a fixed x_t we abbreviate $c_j := \mathcal{N}(x_t; \mu_j, \sigma_j^2)$. The soft assignment or *posterior probability* for arbitrary but fixed $x_t \in X$ and a GMM with K components is given by

$$\gamma_k(x_t) = \frac{\lambda_k \, \mathcal{N}(x_t; \mu_k, \sigma_k^2)}{\sum_{j=1}^K \lambda_j \, \mathcal{N}(x_t; \mu_j, \sigma_j^2)} \in [0, 1],$$

which can be computed using "logsumexp-trick" to avoid explicit computation of c_j. Computing

$$\mathcal{F}_{\lambda_k}^X = \frac{1}{T\sqrt{\lambda_k}} \sum_{t=1}^T \left(\gamma_k(x_t) - \lambda_k\right) \tag{2}$$

$$\mathcal{F}_{\mu_k}^X = \frac{1}{T\sqrt{\lambda_k}} \sum_{t=1}^T \left(\gamma_k(x_t) \left(\frac{x_t - \mu_k}{\sigma_k}\right)\right) \tag{3}$$

$$\mathcal{F}_{\sigma_k^2}^X = \frac{1}{T\sqrt{\lambda_k}} \sum_{t=1}^T \left(\gamma_k(x_t) \frac{1}{\sqrt{2}} \left(\frac{(x_t - \mu_k)^2}{\sigma_k^2} - 1\right)\right) \tag{4}$$

and concatenating these K results $\mathcal{F}_{\lambda_k}^X \in \mathbb{R}, \mathcal{F}_{\mu_k}^X, \mathcal{F}_{\sigma_k^2}^X \in \mathbb{R}^D$ into a $(2D+1)K$ dimensional vector leads to the

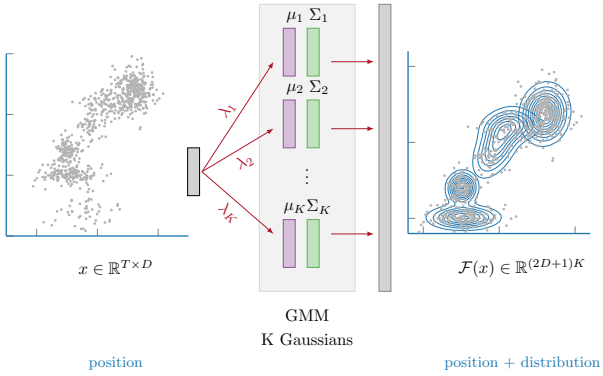


Figure 2: The FV encodes the data distribution fitted to a GMM in addition to the position of the feature.

Fisher-Vector representation ([15, 17]):

$$\mathcal{F} := \left(\mathcal{F}_{\lambda_1}^X, \ldots, \mathcal{F}_{\lambda_K}^X, \mathcal{F}_{\mu_1}^X, \ldots, \mathcal{F}_{\mu_k}^X, \mathcal{F}_{\sigma_1^2}^X, \ldots, \mathcal{F}_{\sigma_k^2}^X\right),$$

which is usually classified by a shallow architecture, *e.g.* a linear SVM. All calculus of vectors should be understood component-wise. This scheme is illustrated in Figure 2. Note, that $\mathcal{F}$ has a fixed dimension independent of T. As described in [22] eq. (2), (3) and (4) can be computed efficiently using pre-computed terms $S_k^p = \sum_t \gamma_k(x_t) {x_t}^p$ for $x_t \in \mathcal{X}, p = 0, 1, 2$. In our prototype implementation we were able to exploit the massively parallel nature of this step, which consists of several reductions.

Finally, a function composition of *power-normalization*

$$x \mapsto \mathrm{sign}(x) \, |x|^\alpha, \quad \alpha \in (0, 1] \tag{5}$$

and *L2-normalization*

$$x \mapsto x \, \|x\|_2^{-1} \tag{6}$$

applied to $\mathcal{F}$ improves the performance [17] of the SVM+FV combination.

Backpropagation A neural network is usually a composition of functions $F := F_n \circ F_{n-1} \circ \cdots \circ F_1$ represented as stacked layers:

$$F_j \colon \mathbb{R}^{d_1} \to \mathbb{R}^{d_2}, \quad X_{j-1} \mapsto X_j := F_j(W_j, X_{j-1}),$$

where $X_{j-1} \in \mathbb{R}^{d_1}$ is the input to the j-th layer and W_j is a collection of tunable parameters. Given X_0, the objective of the training phase for a neural network is to minimize a loss function $L(x, F(x))$ wrt. $W_{j=1,\dots,n}$ most often using stochastic gradient descent optimization.

Given partial derivatives wrt. X_j it is possible to compute the partial derivatives wrt. X_{j-1}, W_j using the chain rule

$$\frac{\partial F_j}{\partial W_j} = \frac{\partial}{\partial W} F_j(W_j, X_{j-1}) \frac{\partial F_{j+1}}{\partial X_j} \tag{7}$$

$$\frac{\partial F_j}{\partial X_{j-1}} = \frac{\partial}{\partial X} F_j(W_j, X_{j-1}) \frac{\partial F_{j+1}}{\partial X_j} \tag{8}$$

as described in [18].

4. Method

In the following, we describe all necessary steps for the Fisher-Vector layer to integrate this layer into backpropagation training.

Fisher-Vector update rules: Since equations (2)-(4) are differentiable in $(\lambda_j, \mu_j, \sigma_j^2)_{j=1,\dots,K}$, these parameters can be optimized using the gradients $\frac{\partial}{\partial x} \ln c_j$, $\frac{\partial}{\partial \mu} \ln c_j$, $\frac{\partial}{\partial \sigma^2} \ln c_j$ and the derivatives of FV wrt. $\ln c_j$ by exploiting the chain-rule.

Elementary calculation yields all derivatives of (2), (3), (4) wrt. to the GMM parameters and x, namely[1]

$$\frac{\partial \mathcal{F}^X_{\lambda_k}}{\partial \lambda_s}, \frac{\partial \mathcal{F}^X_{\mu_k}}{\partial \lambda_s}, \frac{\partial \mathcal{F}^X_{\sigma_s^2}}{\partial \lambda_k}, \frac{\partial [\mathcal{F}^X_{\mu_k}]_d}{\partial [\mu_s]_e}, \frac{\partial \left[\mathcal{F}^X_{\sigma_k^2}\right]_d}{\partial [\mu_s]_e}, \frac{\partial [\mathcal{F}^X_{\mu_k}]_d}{\partial [\sigma_s^2]_e}, \frac{\partial [\mathcal{F}^X_{\sigma_k}]_d}{\partial [\sigma_s^2]_e},$$
$$\frac{\partial \mathcal{F}^X_{\lambda_k}}{\partial [x]_e}, \frac{\partial [\mathcal{F}^X_{\mu_k}]_d}{\partial [x]_e}, \frac{\partial [\mathcal{F}^X_{\mu_k}]_d}{\partial [x]_e}.$$

See the appendix for the formulas. We use gradient descent to update parameters in the backward step.

One part of the gradient for the feature learning stage is given by

$$\frac{\partial}{\partial [x]_e} \left[\mathcal{F}^X_{\mu_k}\right]_d = \frac{1}{\sqrt{\lambda_k}} \left(\frac{\partial \gamma_k(x_t)}{\partial [x]_e} [\alpha_k(x_t)]_d + \delta_{ed} \left[\frac{\gamma_k(x)}{\sigma_k}\right]_d \right),$$

[1] Consult our prototype implementation of symbolic derivatives in the supplemental material.

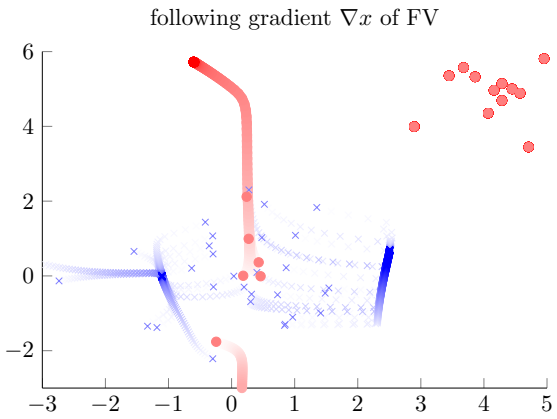


Figure 3: Although, it is not possible to separate both classes using the Fisher-Kernel, learning a transformation of x facilitates the classification of points. Here, each data point x is shifted along the FV-gradient (darker means later in the training process). After the optimization the points are clearly separable.

where

$$\frac{\partial}{\partial x_i} \gamma_k = \gamma_k(x_i) \left(-\beta_k(x_i) + \sum_{n=1}^{K} \beta_n(x_i) \gamma_n(x_i) \right), \tag{9}$$

$$\beta_k(x_i) := \frac{x_i - \mu_k}{\sigma_k^2} \tag{10}$$

and Kronecker-$\delta_{ab} := (a{=}{=}b)$. Therefore, any update will push descriptor x_i into the direction of the k-th Gaussian weighted by the posterior $\gamma_k(x_i)$ and $\frac{1}{\sigma_k^2}$. The effect of this gradient $\frac{\partial}{\partial [x]_e} \left[\mathcal{F}^X_{\mu_k}\right]_d$ is illustrated in Figure 3 for a 2D dataset. There, we shifted each input following the FV-gradient. Notice, that Eq. (9) has a connection to *inverse-variance weighting*.

Satisfying the GMM parameter constraints: It is crucial to not violate the GMM parameter assumptions $\sigma_k^2, \lambda_k > 0, \sum_k \lambda_k = 1$ when applying the steepest descent rule. The authors [21] proposed to re-normalize the weights $\lambda_{j=1,\dots,K} = \overline{\lambda_j} (\sum_k \overline{\lambda_k})^{-1}$ in each iteration to satisfy constraints for λ_j. Instead, we internally model each weight λ_k as a sum of sigmoid-functions:

$$\lambda_j(\nu_j) = \frac{[1 + \exp(-\nu_j)]^{-1}}{\sum_\ell [1 + \exp(-\nu_\ell)]^{-1}} \in (0, 1) \tag{11}$$

and represent σ_k^2 as

$$\sigma_k^2(\zeta_j) = \varepsilon + \exp(\zeta_j) > 0 \tag{12}$$

for some $\nu_j, \zeta_j \in \mathbb{R}$ and $\varepsilon > 0$, which allows to optimize objective (1) in an unconstrained setting in $\lambda, \mu, \sigma^2, x$ and

provides a natural way to satisfy all GMM parameter constraints $\sigma_k^2 > \varepsilon, \lambda_j \in [0,1], \sum_\ell \lambda_\ell = 1$, without numerical issues or projection steps during gradient descent.

Design of SVM gradient information: Let $(x_i, y_i)_{i=1,\dots,N}$ some training data consisting of feature $x_i \in \mathbb{R}^D$ with label $y_i \in \{-1, +1\}$. The back-propagation process starts from the SVM layer, the standard C-SVM:

$$\min_{\theta,b} \frac{1}{\|\theta\|_2^2} + \frac{C}{N} \sum_{i=1}^{N} \max\{0,\ 1 - y_i(\langle \theta, x_i \rangle + b)\} \quad (13)$$

with regularization constant $1/C$ and hinge loss. The update information from (13) is sparse and only forces changes after a long training period, whenever the rare event occurs that a batch contains a miss-classified point. hence, using the non-differential hinge loss for back-propagation one ends up with a sub-gradient-method. Although, the quadratic hinge loss as in [21] is smoother, it also creates sparse gradients.

Switching to the quadratic loss $(1 - y_i(\langle \theta, x_i \rangle + b))^2$ for back-propagating a dense gradient information would move the data points to the margin (see Figure 4), which is unsuitable, since these clusters cannot be represented by Gaussian with diagonal covariance matrices. We use $-y\theta^T$ to shift all data towards the correct "side" away from the decision boundary – detached from the actual SVM formulation for classifying these points.

Normalization layer: The post-processing of FV, a function composition of (5) and (6), which refer to *normalization layer* can be expressed as

$$\phi(x) = \frac{\operatorname{sign}(x)|x|^\alpha}{\| \operatorname{sign}(x)|x|^\alpha \|_2}. \quad (14)$$

Its derivative $\frac{\partial F_j}{\partial X_{j-1}} = \phi'(X_{j-1}) \frac{\partial F_{j+1}}{\partial X_j}$ for $\alpha = 0.5$ is given as

$$\phi'(x) = \frac{2}{\|x\|_1} (1_D - \frac{\hat{x}\hat{x}^T}{\hat{x}^T\hat{x}}) \mathbb{1}_{x \neq 0} \quad (15)$$

for $\hat{x} = \operatorname{sign}(x)\sqrt{|x|}$ and identity matrix 1_D.

Initialization: Learning the GMM is done by k-means initialization until the convergence of the log-likelihood. Initializing the feature learning stage is the tricky part. When using a random transformation W of the PCA-projected SIFT vectors x , the algorithm suffers from a bad initialization. Therefore, we revert these transformations to have exactly the same starting input for the Fisher layer by using features $\tilde{x}$ satisfying $x = \tanh(W\tilde{x} + b) \in [-1, +1]$ in our method. The parameter W is initialized using *Xavier* [7] initialization.

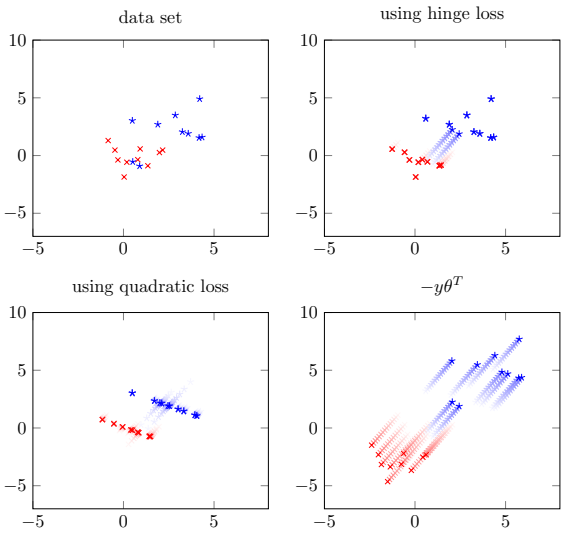


Figure 4: The update-effect of different loss (sub-) gradients in the SVM layer: The hinge loss produces sparse gradients. On the other hand a GMM with coordinates-independence assumption is not capable of representing the result of the quadratic loss. We propose to directly use $-y\theta^T$, which simply shifts each point in the correct direction away from the classification boundary.

5. Implementation and Evaluation

We now describe our prototype CPU/GPU-based implementation, the used datasets and the experimental evaluation of our approach. We evaluate the effectiveness of training additional parameters within the parameter W in our experiments.

5.1. Real-World Dataset and Feature extraction

The publicly available Pascal Voc 2007 (Voc07) dataset [6] comprises 20 classes of objects to be recognized, split up into 5011 images for training and 4952 images for testing for each class.

As the initial fixed features representation we use dense SIFT features extracted at multiple scales from OpenCV normalized to $[-1, 1]$. Each of these 128 dimensional local features is projected onto the first 64 principal components similar to [21], [17]. We observed no decrease of performance when representing each image by randomly selected 10^4 features per image instead of approx. $9 \cdot 10^4$ similar to [21], which reduces the storage costs from 204GB to 25GB. Hence, in our approach each back-propagation needs to compute more than 25GB derivatives per epoch.

5.2. Implementation Details

The implemented architecture consists of multiple layers, which process the input features in batches of 24 images

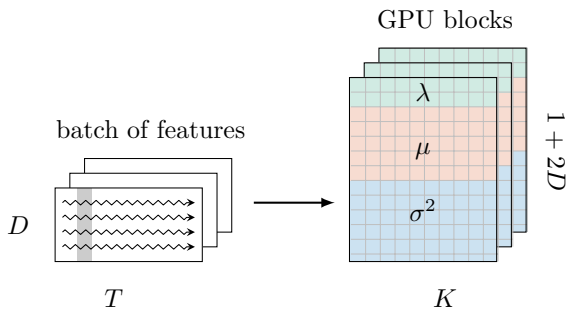


Figure 5: The feature set of each image is distributed to one GPU thread block. The GPU block computes the respective calculus for each feature in the set one after another.

for fast enough updates. Each image described by T PCA-transformed 64 dimensional features is used to compute a 4128 dimensional Fisher-Vector, which was normalized as described in section 3. The underlying GMM contains 32 Gaussians. We trained the SVM (13) using stochastic dual coordinate ascent (SDCA) with $C := N$, which gives superior performance than applying PEGASOS [19] in the primal.

All remaining parameter updates are done by SGD with learning rate $\eta = 10^{-4}$. In contrast, [21] determines an optimal step size in each update.

5.3. GPU Implementation

The main challenge of a batch-wise end-to-end Fisher-Vector implementation is the highly non-linear mapping of each FV-batch back to feature and GMM space. Furthermore, the computation of those derivatives has a high complexity of $\mathcal{O}(K^2D^2T)$. Thus, a fast GPU implementation is indispensable to train the classifier in a feasible amount of time.

Our approach is well suited to take advantage of GPU parallism. There are two levels of granularity in our implementation. The first is by processing the images in parallel. In this fashion, each set of features is assigned to one block of threads. The second is by utilizing a block layout, which is shaped like the derivatives themselves. Hence, it is possible to process all elements of a feature in parallel while sweeping over the complete set of features. This procedure ensures that global memory access is kept at a minimum, while all computations are performed with on-chip memory. Figure 5 shows a scheme of this approach. For the implementation we used CUDA and a setting of four NVIDIA Titan X GPUs. The limiting factors in our GPU approach are the number of available registers and the size of shared memory.

task	timing		speedup of GPU
	Matlab	GPU	
$\mathcal{F}^X$	9.06s	20ms	$\times$453
$\frac{\partial}{\partial\lambda,\mu,\sigma^2}\mathcal{F}^X$	18.9h	10.79s	$\times$6306
$\frac{\partial}{\partial x}\mathcal{F}^X$	1.9h	2.89s	$\times$2367
sum	19.1h	13.71s	$\times$5015

Table 1: Timing comparison of vectorized MATLAB version and GPU implementation for a single batch of 24 images.

Timings For computing the FV and respective derivatives, a comparison between our MATLAB and GPU implementation is shown in Table 1. The highly vectorized MATLAB code is running on a i5-2500 CPU with 3.30GHz. In this experiment, the performance is measured on one CPU core compared to one GPU. The batch size is 24 with 10^4 features each. The reported GPU timings also include all necessary memory transfers between host and device system. Furthermore, both variants compute dense vectors and have no criteria to omit data.

We allow for concurrent Kernel execution, which adapts better for the available resources. The performance for different batch sizes is illustrated in Figure 6, where we use all four GPUs as well as all four cores of the i5 CPU.

In summary, the MATLAB implementation with four cores takes more than 4.7 hours to compute a complete forward and backward step of the Fisher layer for batches of size 24. Our multi-GPU implementation reduces the amount of computation time to less than 5.2 seconds. Note, that for the PASCAL VOC 2007 challenge a complete sweep over the training data comprises 208 batches of 24 images, which still results in about 18 minutes runtime per epoch.

5.4. Experimental Results

To ensure fair evaluation we tested our methods against the baseline method of only training the SVM classifier. To compare our method against [21], we re-implemented their algorithm up to batch-wise updates. As an evaluation metric, we use the *average precision* (AP), corresponding to the area under the precision-recall curve.

In detail, our training process consists of different phases. First, the initial training (with at most 15 epochs, $\approx$ 3000 iterations) of SVM was done. We observed the convergence of the SVM within this phase for all 20 experiments, i.e., the duality gap is less than 0.01. This initial training guarantees already good performance as reported in the first column of Table 2, ie. only θ was learned. Note, that our initial training already significantly outperforms the reported base-line results from [21].

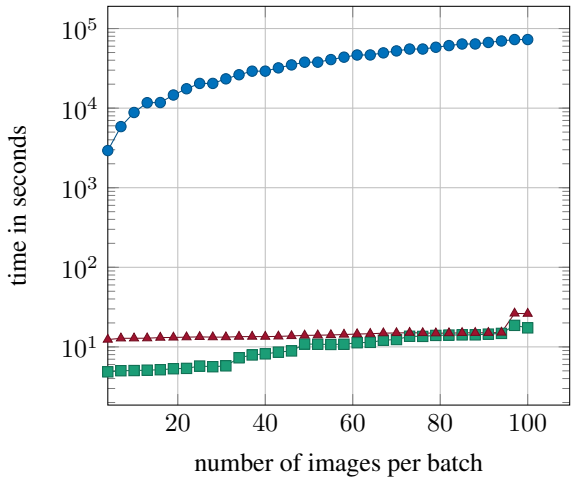


Figure 6: Timing for CPU(—●—), async. GPU (—■—) and sync. GPU (—▲—) for a complete forward and backward batch-computation of the Fisher layer. Smaller is better. Hence, all four GPUs are saturated for 24 images per batch.

	AP (when optimizing parameters)		
class	θ	$\theta, \lambda_k, \mu_k, \sigma_k^2$	$\theta, \lambda_k, \mu_k, \sigma_k^2, (W, b)$
aeroplane	74.1	75.2 (+1.1)	**77.3 (+3.2)**
bicycle	55.1	55.7 (+0.6)	**59.5 (+4.4)**
bird	39.6	40.8 (+1.4)	**41.9 (+2.3)**
boat	68.2	68.5 (+0.3)	**69.8 (+1.6)**
bottle	24.2	24.8 (+0.6)	**25.0 (+0.8)**
bus	56.4	56.9 (+0.5)	**59.0 (+2.6)**
car	74.5	74.9 (+0.4)	**77.8 (+3.3)**
cat	50.2	50.8 (+0.6)	**53.8 (+3.6)**
chair	49.9	50.6 (+0.7)	**51.8 (+1.9)**
cow	30.1	32.2 (+1.1)	**34.3 (+4.2)**
dining table	42.1	43.1 (+1.0)	**44.5 (+2.4)**
dog	34.3	35.0 (+0.7)	**38.2 (+3.9)**
horse	75.3	75.2 (-0.1)	**76.8 (+1.5)**
motorbike	55.1	55.3 (+0.2)	**57.9 (+2.8)**
person	81.1	81.3 (+0.2)	**82.6 (+1.5)**
pottedplant	23.6	24.7 (+1.4)	**27.5 (+3.9)**
sheep	37.7	38.9 (+1.2)	38.9 (+1.2)
sofa	48.9	49.8 (+0.9)	**51.1 (+1.2)**
train	75.6	75.9 (+0.3)	**77.8 (+2.2)**
tvmonitor	46.8	46.7 (-0.1)	**49.5 (+2.7)**
mAP	52.1	52.8 (+0.7)	**54.7 (+2.6)**

Table 2: Results on the PascalVoc2007-Database, when optimizing different parameter sets.

From this, we start to evaluate further training of GMM parameters $(\lambda_k, \mu_k, \sigma_k^2)_{k=1,\dots,K}$ and θ. Training the Fisher kernel alone slightly improves previous performance, at the price of much higher computation time. The corresponding mean AP is comparable to [21]. Starting again from the strong initial training, we now trained all parameters from the GMM and SVM including the feature mapping of our linear layer with Tanh activation. Allowing the first layer to update parameters W and b makes the solution process more flexible allowing the fitting of $\tanh(Wx+b)$ better to the GMM parameters.

Our approach of optimizing all parameters increases the gain of training only GMM parameters by more than three times from 52.8 (+0.7) to 54.7 (+2.6). Remarkably, the total computational effort increases only by factor 1.2, compared to the methods of [21].

6. Conlusion and Outlook

We introduce feature learning in combination with Fisher-Vectors in the fashion of neural networks, which paves the way for a wider range of applications for Fisher-Vectors. Our interpretation of the Fisher-Kernel as a module with a forward and backward pass allows end-to-end training-architectures to benefit from this data distribution encoding scheme. Analogously to the huge impact of GPU implementations in deep learning methods, we expect further progress for GPU-accelerated FV implementations in combinations with other methods.

We believe that this approach enables several future directions. One interesting idea might be the embedding of Fisher-Vector modules in deep neural networks at arbitrary positions. Extracted features from convolution filters can be pooled applying the Fisher layer instead of a fully connected layer. We are planning to add our implementation to popular deep learning frameworks like Caffe or mxnet. Another interesting way of using Fisher-Vectors, is to combine our approach with the work of [20] to train stacked Fisher layers in deep architecture.

Currently, the success of FV depends on robust downsampling approaches like PCA. A batch-wise replacement would facilitate a large-scale end-to-end pipeline including Fisher-Vector training. Solving challenges like pre-training a Gaussian mixture model in a batch-wise online mode would help to realize a fully embedded Fisher layout in a classical back-propagation training.

7. Appendix

The derivatives of the Fisher-Vector layer are given by:

$$\frac{\partial}{\partial \lambda_s} \mathcal{F}^X_{\lambda_k} = \sum_{t=1}^{T} \frac{\delta_{sk}(\gamma_k - \lambda_k) - 2\gamma_s\gamma_k}{2\lambda_s\sqrt{\lambda_k}}$$

$$\frac{\partial}{\partial \lambda_s} \mathcal{F}^X_{\mu_k} = \sum_{t=1}^{T} \frac{\gamma_k \alpha_k\,(\delta_{sk} - 2\gamma_s)}{2\lambda_s\sqrt{\lambda_k}}$$

$$\frac{\partial}{\partial \lambda_k} \mathcal{F}^X_{\sigma_s^2} = \sum_{t=1}^{T} \frac{\gamma_k(\alpha_k^2 - 1)(\delta_{ks} - 2\gamma_s)}{2\lambda_s\sqrt{2\lambda_k}}$$

$$\frac{\partial}{\partial\,[\mu_s]_e} \mathcal{F}^X_{\lambda_k} = \sum_{t=1}^{T} \frac{\gamma_k(\delta_{ks} - \gamma_s)\alpha_s}{[\sigma_s]_e\sqrt{\lambda_k}}$$

$$\frac{\partial}{\partial [\mu_s]_e}\left[\mathcal{F}^X_{\mu_k}\right]_d = \sum_{t=1}^{T} \frac{\gamma_k\left((\delta_{ks}-\gamma_s)\,[\alpha_s]_e\,[\alpha_k]_d - \delta_{sk}\delta_{ed}\right)}{[\sigma_s]_e\,\sqrt{\lambda_k}}$$

$$\frac{\partial}{\partial [\mu_s]_e}\left[\mathcal{F}^X_{\sigma^2_k}\right]_d = \sum_{t=1}^{T} \frac{\gamma_k\left([\alpha_s]_e\left((\delta_{ks}-\gamma_s)([\alpha_k^2]_d - 1) - 2\delta_{de}\delta_{sk}\right)\right)}{[\sigma_s]_e\,\sqrt{2\lambda_k}}$$

$$\frac{\partial}{\partial [\sigma^2_s]_e}\mathcal{F}^X_{\lambda_k} = \sum_{t=1}^{T} \frac{\gamma_k(\delta_{ks}-\gamma_s)\left([\alpha_s^2]_e - 1\right)}{2\,[\sigma^2_s]_e\,\sqrt{\lambda_k}}$$

$$\frac{\partial}{\partial [\sigma^2_s]_e}\left[\mathcal{F}^X_{\mu_k}\right]_d = \sum_{t=1}^{T} \frac{\gamma_k\,[\alpha_k]_d\left((\delta_{sk}-\gamma_s)\,[\alpha_s^2-1]_e - \delta_{sk}\delta_{ed}\right)}{2\,[\sigma^2_s]_e\,\sqrt{\lambda_k}}$$

$$\frac{\partial}{\partial [\sigma^2_s]_e}\left[\mathcal{F}^X_{\sigma_k}\right]_d = \sum_{t=1}^{T} \frac{\gamma_k\left((\delta_{ks}-\gamma_s)\,[\alpha_s^2-1]_e\cdot[\alpha_k^2-1]_d - 2\delta_{ks}\delta_{ed}\,[\alpha_k]_d^2\right)}{2\,[\sigma^2_s]_e\,\sqrt{2\lambda_k}}$$

$$\frac{\partial}{\partial [x]_e}\mathcal{F}^X_{\lambda_k} = \frac{1}{\sqrt{\lambda_k}}\frac{\partial \gamma_k(x)}{\partial [x]_e}$$

$$\frac{\partial}{\partial [x]_e}\left[\mathcal{F}^X_{\mu_k}\right]_d = \frac{1}{\sqrt{\lambda_k}}\left(\frac{\partial \gamma_k}{\partial [x]_e}\,[\alpha_k]_d + \delta_{ed}\left[\frac{\gamma_k(x)}{\sigma_k}\right]_d\right)$$

$$\frac{\partial}{\partial [x]_e}\left[\mathcal{F}^X_{\sigma^2_k}\right]_d = \frac{1}{\sqrt{2\lambda_k}}\left(\frac{\partial \gamma_k(x)}{\partial [x]_e}\,[\alpha_k^2-1]_d + 2\delta_{ed}\gamma_k(x)\,[\beta_k(x_t)]_d\right).$$

We abbreviate $\gamma_\ell(x_t), \alpha_\ell(x_t) = \frac{x_t-\mu_\ell}{\sigma_\ell}$ by dropping the argument. Note, most expressions are sums of dyadic products. Therefore, all gradients are build up on the gradients of the soft-assignment function γ_k which can be precomputed:

$$\frac{\partial}{\partial \lambda_s}\gamma_k = \gamma_k\left(\frac{\delta_{ks}}{\lambda_k} - \frac{\gamma_s}{\lambda_s}\right)$$
$$\frac{\partial}{\partial \mu_s}\gamma_k = \gamma_k(\delta_{ks}-\gamma_s)\left(\frac{x-\mu_s}{\sigma_s^2}\right)$$
$$\frac{\partial}{\partial \sigma^2_s}\gamma_k = \gamma_k(\delta_{ks}-\gamma_s)\left(\frac{(x-\mu_s)^2}{2\sigma_s^4} - \frac{1}{2\sigma_s^2}\right).$$

References

[1] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool. Speeded-up robust features (surf). *Comput. Vis. Image Underst.*, 110(3):346–359, June 2008. 1

[2] S. Bell, P. Upchurch, N. Snavely, and K. Bala. Material recognition in the wild with the materials in context database. *CoRR*, abs/1412.0623, 2014. 1

[3] M. Cimpoi, S. Maji, and A. Vedaldi. Deep convolutional filter banks for texture recognition and segmentation. *CoRR*, abs/1411.6836, 2014. 1

[4] D. C. Ciresan, U. Meier, and J. Schmidhuber. Multi-column deep neural networks for image classification. *CoRR*, abs/1202.2745, 2012. 1

[5] M. Dixit, S. Chen, D. Gao, N. Rasiwasia, and N. Vasconcelos. Scene classification with semantic fisher vectors. June 2015. 1, 2

[6] M. Everingham, L. Gool, C. K. Williams, J. Winn, and A. Zisserman. The pascal visual object classes (voc) challenge. *Int. J. Comput. Vision*, 88(2):303–338, June 2010. 5

[7] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In *In Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS10). Society for Artificial Intelligence and Statistics*, 2010. 5

[8] T. Jaakkola and D. Haussler. Exploiting generative models in discriminative classifiers. In *In Advances in Neural Information Processing Systems 11*, pages 487–493. MIT Press, 1998. 2

[9] H. Jegou, M. Douze, C. Schmid, and P. Perez. Aggregating local descriptors into a compact image representation. In *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*, pages 3304–3311, June 2010. 1

[10] B. Klein, G. Lev, G. Sadeh, and L. Wolf. Fisher vectors derived from hybrid gaussian-laplacian mixture models for image annotation. *CoRR*, abs/1411.7399, 2014. 3

[11] L. Liu, C. Shen, L. Wang, A. van den Hengel, and C. Wang. Encoding high dimensional local features by sparse coding based fisher vectors. In Z. Ghahramani, M. Welling, C. Cortes, N. Lawrence, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 1143–1151. Curran Associates, Inc., 2014. 1, 2

[12] D. G. Lowe. Distinctive image features from scale-invariant keypoints. *Int. J. Comput. Vision*, 60(2):91–110, Nov. 2004. 1

[13] A. Oliva and A. Torralba. Modeling the shape of the scene: A holistic representation of the spatial envelope. *Int. J. Comput. Vision*, 42(3):145–175, May 2001. 1

[14] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, June 2007. 1

[15] F. Perronnin and C. Dance. Fisher kernels on visual vocabularies for image categorization. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–8, June 2007. 3

[16] F. Perronnin and D. Larlus. Fisher vectors meet neural networks: A hybrid classification architecture. June 2015. 2

[17] F. Perronnin, J. Snchez, and T. Mensink. Improving the fisher kernel for large-scale image classification. In *Proceedings of the 11th European Conference on Computer Vision: Part IV*, ECCV'10, pages 143–156, Berlin, Heidelberg, 2010. Springer-Verlag. 3, 4, 5

[18] D. E. Rumelhart, G. E. Hinton, and R. J. Williams. Neurocomputing: Foundations of research. chapter Learning Representations by Back-propagating Errors, pages 696–699. MIT Press, Cambridge, MA, USA, 1988. 4

[19] S. Shalev-Shwartz, Y. Singer, and N. Srebro. Pegasos: Primal estimated sub-gradient solver for svm. In *Proceedings of the 24th International Conference on Machine Learning*, ICML '07, pages 807–814, New York, NY, USA, 2007. ACM. 6

[20] K. Simonyan, A. Vedaldi, and A. Zisserman. Deep fisher networks for large-scale image classification. In C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 163–171. Curran Associates, Inc., 2013. 2, 7

[21] V. Sydorov, M. Sakurada, and C. H. Lampert. Deep fisher kernels - end to end learning of the fisher kernel gmm parameters. June 2014. 2, 4, 5, 6, 7

[22] J. Snchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the fisher vector: Theory and practice.

International Journal of Computer Vision, 105(3):222–245, 2013. 2, 3

[23] V. Vapnik and O. Chapelle. Bounds on error expectation for support vector machines. *Neural Comput.*, 12(9):2013–2036, Sept. 2000. 1, 2

[24] D. Yoo, S. Park, J. Lee, and I. Kweon. Fisher kernel for deep neural activations. *CoRR*, abs/1412.1628, 2014. 1

D

GGNN: GRAPH-BASED GPU NEAREST NEIGHBOR SEARCH

GGNN: Graph-based GPU nearest neighbor search

Fabian Groh, Lukas Ruppert, Patrick Wieschollek, and Hendrik P.A. Lensch

IEEE Transactions on Big Data, 2022 [4]

Updated pre-print version: https://arxiv.org/abs/1912.01059.

GGNN: Graph-based GPU Nearest Neighbor Search

Fabian Groh, Lukas Ruppert, Patrick Wieschollek, and Hendrik P.A. Lensch

Abstract—Approximate nearest neighbor (ANN) search in high dimensions is an integral part of several computer vision systems and gains importance in deep learning with explicit memory representations. Since PQT [1], FAISS [2], and SONG [3] started to leverage the massive parallelism offered by GPUs, GPU-based implementations are a crucial resource for today's state-of-the-art ANN methods. While most of these methods allow for faster queries, less emphasis is devoted to accelerating the construction of the underlying index structures. In this paper, we propose a novel GPU-friendly search structure based on nearest neighbor graphs and information propagation on graphs. Our method is designed to take advantage of GPU architectures to accelerate the hierarchical construction of the index structure and for performing the query. Empirical evaluation shows that GGNN significantly surpasses the state-of-the-art CPU- and GPU-based systems in terms of build-time, accuracy and search speed.

Index Terms—Nearest neighbor searches, Graph and tree search strategies, Information retrieval, Approximate search, Similarity search, Big data.

1 Introduction

Approximate nearest neighbor (ANN) search plays a crucial and long-standing role in various domains, including databases, computer vision, autonomous vehicles, personalized medicine, and machine learning. Since collecting large amounts of data became easier, the creation of a scalable and efficient data structure for retrieving similar items has become an active research topic. Despite all recent advances, the only available method for guaranteed retrieval of the exact nearest neighbor in high dimensions is still exhaustive search, due to the curse of dimensionality [4]. Even when leveraging modern hardware, it remains impractical to perform an exhaustive search over billions of high-dimensional data points. Instead, most popular methods relax the problem by searching for an entry that is likely to be the nearest neighbor, accepting a minimal loss in accuracy.

Besides designing a very fast GPU-based approximate kNN query algorithm, we address the efficient construction of a hierarchical graph-based search structure. Building time becomes more and more important in dynamically growing or changing datasets for on-the-fly analysis, e.g. in correspondence and feature matching for tracking and object recognition in videos, for newly embedded vectors of neural networks similar to DEEP1B [5], or in recommender systems [6]. Our graph-construction explicitly determines the true k nearest neighbors of each point in the dataset with high probability. Solving this for-all task is highly relevant in n-body problems like salient point estimation, kernel-density computation, cluster analysis, retrieval of more robust prototypes, or feature matching in embeddings.

To keep up with the scale of data that is produced day by day, modern approaches use index structures that are heavily tailored towards exploiting the massive parallelism of GPUs [1], [2], [7], [8] or custom hardware [9], as opposed to previous CPU-based methods [10], [11], [12], [13], [14], [15], [16], [17].

The quality of the recall heavily depends both on the choice of the search structure and the executed search (query) itself. Structures based on quantization or hashing/binning schemes [1], [2], [11], [12], [13], [14], [15], [16], [17], [18], [19] can be built efficiently, but typically suffer from relatively low recall rates as enumerating and visiting neighboring cells is exhaustive in high dimensions. Better recall rates are recently achieved by graph-based methods [8], [20], [21], [22], [23], [24], [25], [26], [27]. Existing methods for constructing effective search-graphs, e.g. [20], update varying-sized edge lists sequentially. They are highly dependent on global memory synchronization and difficult to parallelize effectively beyond a few cores. Hence, their construction times are not scaling well and are measured in hours or even days [19], [27].

Given a precomputed graph structure, a query traverses the edges of the graph to decrease the distance to a query point. It needs to compute the distances to all neighbors of the currently investigated node, move to the next-best point and store which points have already been visited. All decisions are made locally and independent for each query, rendering the query algorithm an ideal candidate for parallelization. However, a parallel implementation needs to carefully address a number of issues to be efficient. For one, the available memory bandwidth for loading each vector to compare with might become a limiting factor. Here, Optane memory [27] or GPUs [3] offer a solution. Secondly, on GPUs one needs to cope with the limited amount of available per-thread or per-block memory when storing the list of visited points.

Hence, we propose a GPU-friendly query design on thread-block level with high on-chip resource utilization through a fully parallel multi-purpose cache and a fixed number of neighbors per point. Further, we introduce a novel technique for very fast graph construction that utilizes the fast parallel query algorithm

- *During the work of this paper, all authors were member of the Computer Graphics group at the University of Tübingen, Germany.*
- *Patrick Wieschollek and Fabian Groh are now with Amazon, Tübingen, Germany. This work has been done prior to joining Amazon.*

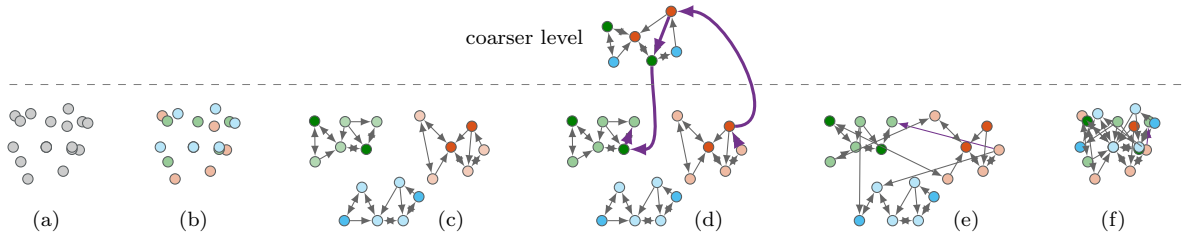


Fig. 1: Illustration of the bottom-up construction of the kNN-graph for a dataset (a). After random parititioning (b), a kNN-graph is constructed for each partition (c). A selection of nodes builds a coarser kNN-graph (d), which is used to propagate links between partitions. These links are used to merge several partitions (e) into a final kNN-graph (f).

to iteratively merge multiple sub-graphs together. This bottom-up construction scheme is sketched in Figure 1. Hierarchical graphs are used as a global optimization substitution to overcome gaps in local connectivity, in particular during construction. Additionally, our local symmetric linking approach reduces the amount of redundant links in the graph to circumvent negative impacts of overflows in the memory-limited caches.

Our approach is also inherently suited for batch processing by already generating multiple sub-graphs at the same time. By forgoing the final merge of several sub-graphs into one combined graph, one can easily obtain independent yet effective sub-graphs for parts of large-scale datasets which would otherwise not fit in memory. These can easily be processed even on multiple GPUs at the same time where each query then needs to be executed once on each shard. This simple yet effective method allows for optimal multi GPU utilization.

To summarize our main contributions, we propose a method for extremely fast approximate kNN-search on GPUs for high-dimensional data. We focus not only on fast query time, but also on a very efficient construction of the index structure, while still achieving high recall rates of the true nearest neighbours.

As evidenced by our empirical evaluation, the presented scheme outperforms existing approaches concerning both the construction as well as the query time. At the same time, the recall rate is consistently high and can be traded in for even faster construction or query time. We present a multi-GPU scheme that is capable of achieving above 99% recall even for common benchmark datasets with billions of high-dimensional entries.

2 Related Work

A large amount of literature exists on designing structures that accelerate a nearest neighbor search. Besides traditional approaches [10], [28], most popular techniques rely either on data quantization in clusters [1], [2], [11], [12], [13], [14], [15], [16], [17], [18], [19] or building neighborhood graphs [8], [22], [23], [24], [25], [26], [27]. To achieve peak performance, most of these methods compute a compressed representation for each entry as large datasets will not fit into fast memory. There exist several strategies to compute such a compression. While hashing methods [29], [30], [31] produce compact binary codes, quantization-based methods reuse centroids by assigning each data point a unique identifier based on the centroid to which they belong. It has been empirically shown that quantization methods are more accurate than various hashing methods [10], [11].

Quantization methods for nearest neighbor search using clustering methods were popularized by Jégou *et al.* [11] while originally being introduced in [32]. Such index structures, like IVFADC [11], partition the high-dimensional search space into disjoint Voronoi cells described by a set of centroids obtained by Vector Quantization (VQ) [33]. The idea has been extended later by Babenko *et al.* [15], where the high-dimensional vector-space is factored in orthogonal subspaces. Hereby, each vector is assigned to a centroid independently for each subspace according to a separate codebook that resides there. Wieschollek *et al.* [1] proposed a hierarchical representation of the codebook besides demonstrating superior performance using a GPU. Johnson *et al.* [2] ported IVFADC [11] to the GPU in combination with a fast GPU-based implementation for k-selection, *i.e.*, returning the k lowest-valued elements from a given list (a crucial part of quantization based methods). They are the first employing multi-GPU parallelism by replication and sharding. Their work forms the library "FAISS". Eventually, Chen *et al.* [18] proposed a GPU based method RobustiQ overcoming the memory limitations of FAISS by extending the idea of Line Quantization from [1] in a hierarchical fashion. Still, the reported distances are only an approximation of the true distance.

All hashing and quantization-based indexing schemes share the same problem that they partition the space into cells. While the containing cell for a query might be found very efficiently, the exact nearest neighbor might be across the boundary to one of the neighboring cells. Determining and visiting all neighboring cells in high dimension is a problem severely limiting these approaches.

kNN-graph based methods are another way to accelerate the query process. Our presented approach belongs to this category. The main idea is to link each point from the search space to k of its nearby points. Each query will start at a random guess in the dataset. Then, the guess itself is refined by replacing it with a better point from the k linked neighbor points. Chen *et al.* [23] propose a fast divide and conquer strategy for computing such kNN-graphs. Done *et al.* [22] introduced NN-descent for using kNN-graphs to accelerate NN-search. Hereby, each point maintains a list of its own nearest neighbors and points where itself is considered as a nearest neighbor. This has been later extended [24] to make use of MapReduce. EFANNA as a multiple hierarchical index structure uses a truncated KD-tree to build a kNN-graph [25].

In the ideal case, a kNN-graph augmented with additional links could guarantee that for an arbitrary starting point the NN-descent will converge to the correct solution. Computing such a graph with additional links at scale is not practicable. Therefore, several methods exist to at least approximate such a graph [8], [26]. Fu *et al.* [26] introduce NSG as an approximation. To reduce the overall number of edges, their optimization tries to lower the out-degree individually per node. Their method can scale beyond multiple cores, outperforming a GPU approach [2] on a benchmark dataset. Harwood *et al.* [8] suggest an alternative approach for

constructing such a graph. Starting from a fairly dense graph, they remove "shadowed edges", which are redundant when considering traversing paths during a query. They showed promising results using the GPU but only on rather small datasets as their build time is rather high. Malkov *et al.* [20] construct a hierarchical graph structure to accelerate the nearest neighbor search.

For graph-based methods, memory throughput is often the limiting factor. Ren *et al.* [27] propose an optimized construction of HNSW-style search graphs on out-of-core datasets using heterogeneous memory. Using fast Optane memory allows them to quickly query even billion-scale datasets at high accuracy. Zhao *et al.* [3] propose GPU-based "search on graph" (SONG) using index structures built using either HNSW [20] or NSG [26], achieving significant speedup over CPU-based queries in most cases.

Our method is most similar to HNSW [20] with its hierarchical construction, but it is very carefully tuned for optimal parallelism of construction tasks and also differs during the query.

3 Background

In this section, we formally introduce the approximate nearest neighbor (ANN) problem statement and used notation.

3.1 Nearest Neighbor Search

The nearest neighbor problem retrieves a point $x^\star$ from a dataset $\mathcal{X} = \{x_1, \ldots, x_n\}$ that has the smallest distance to a query q. For the sake of simplicity, here, we assume an Euclidean space ($\mathcal{X} \subset \mathbb{R}^d, q \in \mathbb{R}^d$) and Euclidean distances ($\|\cdot\|_2$). The nearest neighbor $x^\star \in \mathcal{X}$ of q therefore is defined as

$$x^\star = \arg\min_{x \in \mathcal{X}} \|q - x\|_2 \,. \tag{1}$$

Similarly, the k-nearest neighbor search retrieves the k closest entries from $\mathcal{X}$ for a given query. As finding the exact nearest neighbor might be costly, we may accept points in $\mathcal{X}$ which are close to q and therefore deliver an approximate solution to Eq. (1).

3.2 KNN Graph

In a kNN-graph, each point x form the dataset $\mathcal{X}$ represents one node in the graph G. Further, we define $\mathcal{N}_x \subseteq \mathcal{X}$ as a local neighborhood of x with k elements and defer the details on how to construct $\mathcal{N}_x$ to the next section. The edges E of the graph are then defined as (x, y) where $y \in \mathcal{N}_x$. Note that the resulting graph is a directed graph $G = (\mathcal{X}, E)$, where $E = \{(x, y) \mid x \in \mathcal{X}, y \in \mathcal{N}_x\}$ and $(x, y) \in E \not\Rightarrow (y, x) \in E$.

One greedy algorithm to find the nearest neighbor for a query point q is *NN-descent* [22]. Starting from an initial guess $x \in \mathcal{X}$, the distance between q and each neighboring point $y \in \mathcal{N}_x$ is computed. If any $y \in \mathcal{N}_x$ is closer to q than x, the guess x is replaced by the closest point from $\mathcal{N}_x$ to q. This process iterates until no point in $\mathcal{N}_x$ has a smaller distance to q than x. However, as the current x might not provide an edge into the right search direction, this greedy algorithm might get stuck in a local minimum on a pure kNN-graph.

3.2.1 Common Pitfalls

Since the NN-descent is a greedy search, it offers no guarantee on finding the exact solution. Some of the reasons are listed below:

Connectivity: As a kNN-graph is a directed graph, y might be directly connected to x, being its nearest neighbor, hence $y \in \mathcal{N}_x$. But this does not imply that the inverse link exists ($y \in \mathcal{N}_x \not\Rightarrow x \in \mathcal{N}_y$). Therefore, the construction of an augmented (diversified) kNN search-graph has to deal with synchronizing outgoing and incoming (inverse) edges.

Gaps in high-dimensional spaces: As each point is only linked to a finite number of local neighbors, there exist pathological cases (even in 2D), where close-by points are not directly connected at all. Such a case is illustrated in Figure 3. Due to the gap, the true nearest neighbor will not be found. Computing an idealized monotonic relative neighborhood graph [26] (MRNG) would avoid this issue, but requires a strongly varying connectivity not suitable for parallel approaches – besides its additional computational burden.

Degree of nodes: There exists a trade-off when choosing the cardinality of $\mathcal{N}_x$ for any $x \in \mathcal{X}$. Having fewer edges amplifies the previously described issues, but also reduces the number of necessary comparisons at each step. Conversely, having many edges allows the greedy search to escape from local neighborhoods but increases the cost for each iteration.

4 GPU-Based Nearest Neighbor Search

Searching for the k nearest neighbors for multiple queries, given some graph structure, is the central operation for various applications and also the main building block for our graph construction (see Section 5). Achieving high recall rates at very short times is the primary goal of our parallel GPU-based search-algorithm on kNN-graph structures. The algorithm can, within bounds, be tuned for quality or speed, offering a solution to a broad range of use-cases, accommodating those requiring high recall rates as well as real-time algorithms with reduced quality constraints.

The main idea is to highly parallelize the search by utilizing one thread-block per query. In contrast to the naïve solution of a query per thread, the main advantage is to bundle enough on-chip resources that all meta-data can be kept on very fast memory. This is of utmost importance, since graph-based methods still need to load a large amount of neighborhood information as well as vectors from global memory for distance computations. In addition, the thread-block approach guarantees high memory bandwidth through very efficient coalesced memory accesses. It also allows to perform more computation in parallel since common tasks like distance computations or the maintenance of priority queues and visited lists can be parallelized. Furthermore, individual queries that vary in runtime can be scheduled independently.

4.1 GPU-based Search with Backtracking

Assuming a diversified (see Section 5.2) kNN-graph with given starting points $\mathbf{s} \subset \mathcal{X}$, for a query $q \in \mathbb{R}^d$, the simple greedy downhill search with backtracking from Algorithm 1 is executed. First, the cache structure (see Section 4.1.1) is initialized (`init`) with the query point and a slack factor τ (see Section 4.1.2). Then, the starting points $\mathbf{s}$ are introduced to the cache by `fetch`, which computes their distances to the query q and adds them to the priority queue ($prioq$). As starting points $\mathbf{s}$, we use the nodes on the top-layer of our hierarchical graph (see Section 5.4).

Until there are no more unexplored points in the priority queue, all neighbors $\mathcal{N}_a$ of the closest not yet visited point a are being fetched to the cache. Essentially, performing a depth-first search in the graph. Our cache structure manages a sorted list of the k closest points ($best$) to the query which are observed during the traversal. Hence, the $best$ list is returned as the result of the search.

Algorithm 1: Basic query with backtracking.

Function `query` (G, $\mathbf{s}$, q, τ, k) **:**
  Input: search-graph G with start points $\mathbf{s}$, query point q, slack factor τ (see Section 4.1.2)
  Output: k closest points to q
  $cache$.`init` (q, τ) `/* (see Section 4.1.1) */`
  $cache$.`fetch` ($\mathbf{s}$)
  while $(a \leftarrow cache$.`pop` `()` $) \neq \emptyset$**:**
    $cache$.`fetch` ($\mathcal{N}_a$) `/* (see Section 3.2) */`
  return $cache.best$

4.1.1 Caching on GPUs

The centerpiece of our fast GPU-based kNN search is a multi-purpose cache (see Algorithms 1 and 2). One of the major deficits of GPUs is their limited on-chip memory (*register* and *shared memory*[1]) that can be accessed with low latency.

In particular, kNN-graph algorithms have a strong demand for highly dynamic structures like unpredictably growing lists of potential points to visit or of already visited points. These meta-data are perused and modified frequently during the query.

The cache's memory layout is shown in Figure 2. It consists of three major parts:

1) *best*: A sorted list of the points closest to the query and their distances.
2) *prioq*: A priority-queue that manages points to be visited as a distance-sorted ring-buffer.
3) *visited*: A ring-buffer that caches indices of already visited points in a first in first out (FIFO) fashion.

All three parts are handled in *shared memory*. The combined length is a multiple of the assigned threads, hence, every thread has multiple work items.

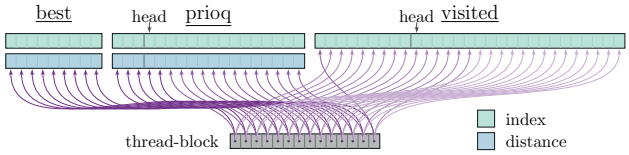


Fig. 2: Our cache consists of a *best* list, *prioq* (priority queue) and a *visited* list. It resides entirely within shared memory. The *best* list and *prioq* both contain indices and distances and are sorted by distance. The *visited* list contains only indices to preserve memory. The *prioq* and the *visited* list are implemented as ring buffers. This allows elements to be easily `pop`ped from the front of the *prioq* and to overwrite old elements in the *visited* list once it is full. The cache is accessed and maintained in parallel by the entire thread-block.

The main methods of our cache are shown in Algorithm 2 and described in the following paragraphs. Note that for ease of readability synchronization barriers preventing race conditions are omitted. We refer to our published open source code[2] for more information. For the purpose of visualization, *shared memory* variables are denoted with an underline, while normal variables can be assumed to live in *register* space.

Fetch: Given a list of point proposals $\mathbf{p}$, first, known elements are removed from the proposal list. Each proposal is compared in parallel against the working items of each thread and removed in case of a match. The remaining proposals are processed iteratively. The distance to the query point is computed in parallel (`dist`).

Algorithm 2: Cache Operations.

Function `fetch` ($\mathbf{p}$) **:**
  Input: Point proposals $\mathbf{p}$.
  parfor $p_{c_i} \in \{\underline{best}, \underline{prioq}, \underline{visited}\}$ **:**
    for $p \in \underline{\mathbf{p}}$ **:**
      if $p_{c_i} = p$ **:**
        remove p from $\underline{\mathbf{p}}$
  for $p \in \underline{\mathbf{p}}$ **:**
    $d \leftarrow$ `dist` (p) `/* Compute in parallel. */`
    if `criteria`(d) **:**
      `push` (p, d)

Function `push` (p, d) **:** `/* Parallel insertion */`
  Input: Point proposal p with distance d.
  parfor $p_{c_i}, d_{c_i} \in \{\underline{best}, \underline{prioq}\}$ **:**
    `/* Read` p_{c_i} `and` d_{c_i} `and sync. */`
    if $d_{c_i} \geq d$ **:**
      if `is_not_end`(c_i) **:**
        $\underline{p_{c_{i+1}}} \leftarrow p_{c_i}$
        $\underline{d_{c_{i+1}}} \leftarrow d_{c_i}$
      if `is_begin`(c_i) *or* $\underline{d_{c_{i-1}}} < d$ **:**
        $\underline{p_{c_i}} \leftarrow p$
        $\underline{d_{c_i}} \leftarrow d$

Function `pop` `()` **:** `/* Single-threaded routine */`
  Output: Returns head of priority queue.
  $p, d \leftarrow \underline{prioq_{head}}$
  if *not* `criteria`(d)**:**
    return *empty*
  `/* Remove point from` *prioq*`. */`
  $\underline{prioq_{head}} \leftarrow$ *empty*
  `move_head_forward`$(head_{prioq})$
  `/* Add point to` *visited* `list. */`
  $\underline{visited_{head}} \leftarrow p$
  `move_head_forward`$(visited_{head})$
  return p

Function `criteria` (d) **:**
  return $d \leq d_{best_k} + \xi$ `/* (see Section 4.1.2) */`

Every thread reads its respective vector element(s) from memory and computes the element-wise result. A subsequent reduction produces the complete distance. For parallel primitives like reduction, we use NVIDIA's CUB[3] library for highly optimized implementations. The parallel coalesced loading and processing is highly effective in terms of GPU utilization. Points that are inside our `criteria` are `push`ed to the cache structure.

Push: For a given pair of index p and distance d, the `push` method performs a parallel insertion into the distance-sorted lists of *best* and *prioq*. To perform the insertion, all items with index c_i that have a distance which is greater than d are temporarily copied into the thread-block's registers and are written back into the subsequent index c_{i+1} unless the end of the list or ring-buffer is reached. The new element is inserted where the item to the left is closer than the new item, or at the beginning if the new item is closer than all previous items. Consequently, a point that is eligible for *best* and *prioq* will be inserted in both lists simultaneously.

1. https://docs.nvidia.com/cuda/
2. https://github.com/cgtuebingen/ggnn
3. https://nvlabs.github.io/cub/

Since *prioq* is a ring-buffer, the logical beginning and end positions are dependent on the head position. Hence, index computations on the physical borders need to be wrapped around.

Pop: This single-threaded routine returns the current head of the priority queue and manages the ring-buffers. Our `criteria` is monotonically decreasing over the course of a query (inspect Section 4.1.2). Hence, if the head of the *prioq* violates the `criteria`, we are able to safely terminate the query. In the valid case, the point is removed from the *prioq* and added to the *visited* list at the head position. Both head pointers are moved one step forward. Finally, the point is returned.

4.1.2 Stopping Criterion

On high-dimensional data, algorithms that solely rely on greedy downhill search will quickly get stuck in local minima. On the other hand, complete backtracking might visit the entire dataset.

A common stopping criterion, e.g. [20], is to terminate the search once the *best* list cannot be improved by adding the closest not yet visited element, i.e., once $d > d_{\text{best}_K}$, where d is the distance of the new element and d_{best_K} is the distance of the last element in the *best* list. To achieve higher recalls, the *best* list needs to be extended (K needs to be increased), potentially by hundreds of elements.

We propose an efficient approximation by adding an adaptive, monotonically decreasing slack ξ to the distance of the k-closest neighbor: $d_{\text{best}_K} \approx d_{\text{best}_k} + \xi$, where $k \ll K$, that allows us to limit the size of the *best* list to only the k closest neighbors which are to be queried (or at least 10). Instead of explicitly tracking the distance of further neighbors, the slack accounts for a safety margin to ensure that the path to the closest neighbor will likely be found, e.g. Figure 3. The search will terminate once

$$d > d_{\text{best}_k} + \xi, \quad \xi = \tau \cdot \min\{d_{\text{best}_1}, d^+_{\text{nn}_1}\}. \tag{2}$$

The slack factor τ controls the size of the safety margin. d_{best_1} is query-specific and relates the margin to the currently best found match while $d^+_{\text{nn}_1}$ correlates with the density of the given database nodes. Specifically, $d^+_{\text{nn}_1}$ provides a global limit (to catch outliers) that is calculated during graph construction. It denotes the maximum distance to the closest neighbor across all points within the currently processed subset of the graph $\mathcal{S}$:

$$d^+_{\text{nn}_1} = \max_{x \in \mathcal{S}} \left\{ \min_{y \in \mathcal{N}_x} \|x - y\|_2 \right\}, \text{where } \mathcal{S} \subseteq \mathcal{X}. \tag{3}$$

Using our stopping criterion keeps the *best* list at a small size and by that reduces the amount of shared memory required for performing the query. As demonstrated by Figure 3, our stopping criterion allows for high efficiency even at high accuracy.

Typical values for τ are above 0 and below 2. Larger values improve accuracy with diminishing returns.

5 Approximate Symmetric Nearest Neighbor Graph Construction

The construction of proper CPU-based kNN search-graphs is performed typically in a sequential procedure on global graphs. Edges are either attached (e.g., [20]), or pruned (e.g., [8], [21], [26]) one after the other. During the process, the edge-lists per node tend to vary heavily ranging from completely empty to complete lists of all possible points. While these global optimization approaches are able to produce high quality graph structures on CPUs, for fast GPU-based construction, they are impractical.

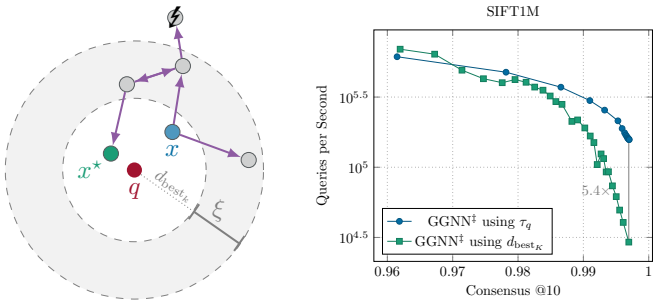


Fig. 3: Allowing a slack ξ makes it possible to escape a local minimum $x \in \mathcal{X}$, eventually reaching the solution $x^\star \in \mathcal{X}$ for a given query $q \in \mathbb{R}^d$. Points outside of this boundary are unlikely to provide helpful links and are discarded to reduce computational cost. (right) Comparison of using our stopping criterion (τ_q) to using a growing *best* list to terminate queries. With the *best* list based approach, high recalls can only be achieved in conjunction with high memory usage which slows down the query.

We propose a parallel graph construction based on parallel merging of hierarchical kNN search-graphs as sketched in Figure 1. By partitioning the task of constructing the search graph into small, parallelizable tasks, we are able to efficiently use the massive parallism offered by GPUs. In the following, we present our parallel graph construction process, followed by details on the hierarchical query process, and the graph-diversification and refinement by symmetric linking. Finally, we look at how to best perform the final queries and multi-GPU configurations.

5.1 Building the Hierarchical kNN-Graph

Our construction process (Algorithm 3) builds the kNN search-graph bottom-up by recursively merging smaller search-graphs.

To `init`ialize the process, we logically partition the entire dataset $\mathcal{X}$ into small batches of size s, e.g. 32. These represent hierarchical search-graphs of height 1. We initialize the neighbors within the bottom layer by first performing the `merge` operation which initializes the k outgoing edges per point with each point's nearest neighbors. Then, the `sym` operation replaces up to k_{sym} outgoing links in an effort to approximate an undirected graph, details in Section 5.2.

In order to merge the individual sub-graphs, we first `select` s points from the groups of g graphs which each are to be merged. These selected points now form batches representing the new top layer. There, we perform the same operations as with the bottom-layer before. We `merge`, i.e. query for the nearest neighbors to fill the outgoing edge list and perform graph-diversification (`sym`). As the nearest neighbor of any point might be found in any sub-graph, edges across batch boundaries will be introduced. These steps now repeat top-down until the entire hierarchical search-graph is interconnected.

Whenever the bottom layer is reached, we also save the distance to the first nearest neighbor d_{nn_1} for each point and compute the mean and max over the dataset (`stats`). This allows for an easy adaption of our stopping criterion to every dataset without any precomputation. Initially, the maximum is prone to outliers in the still coarse nearest neighbor graph. Therefore, during construction, we substitute the maximum distance to the closest neighbor $d^+_{\text{nn}_1}$ in Eq. 2 with the mean distance $\bar{d}_{\text{nn}_1}$:

$$\bar{d}_{\text{nn}_1} = \frac{1}{|\mathcal{S}|} \sum_{x \in \mathcal{S}} \left\{ \min_{y \in \mathcal{N}_x} \|x - y\|_2 \right\}, \text{where } \mathcal{S} \subseteq \mathcal{X}. \tag{4}$$

Algorithm 3: Parallel graph construction.

Function `BuildGraph` ($\mathcal{X}$, L, k, τ_{build}) **:**
  Input: Dataset $\mathcal{X}$, number of layers L, number of neighbors per point k, slack factor τ_{build}
  Result: search structure `graph`
  `graph` $\leftarrow$ `init` ($\mathcal{X}$, L, k, τ_{build})
  for $l_{top} \leftarrow 0$ **to** $L-1$**:**
    if $l_{top} > 0$**:**
      `graph.select` (l_{top})
    for $l_m \leftarrow l_{top}$ **to** 0**:**
      `graph.merge` (l_{top}, l_m)
      `graph.sym` (l_m)

Function `graph.merge` (l_{top}, l_m) **:**
  parfor $z \in$ `graph.layer` (l_m) **:**
    `/* Use brute force if` $l_{top} = l_m$. `*/`
    $b_z \leftarrow$ `graph.query` (z, l_{top}, l_m)
  `graph.layer` (l_m) $\leftarrow b$
  if $l_m = 0$**:**
    `graph.stats` ()

Function `graph.sym` (l_i) **:**
  parfor $z \in$ `graph.layer` (l_i) **:**
    `graph.sym_links` (z)

Algorithm 4: Hierarchical query.

Function `graph.query` (z, l_{top}, l_m) **:**
  Input: query point z from layer l_m, start layer l_{top}, and end layer l_m
  Output: k closest points to z
  $cache$.`init` (z, τ)
  $\mathbf{s} \leftarrow$ `getTopSegment` (z, l_{top})
  $cache$.`fetch` ($\mathbf{s}$)
  for $l_i \leftarrow l_{top} - 1$ **to** l_m**:**
    $cache$.`transform` (l_i)
    while $(a \leftarrow cache$.`pop` `()` $) \neq \emptyset$**:**
      $cache$.`fetch` ($\mathcal{N}_a$)
  return $cache.best$

For selecting the points for the top layer, weighted reservoir sampling with d_{nn_1} as weights is used with the method of [34].

As all construction tasks can be performed in parallel for each point in each layer while requiring small, limited amounts of memory each, the construction can be performed quite quickly when exploiting the massive parallelism offered by GPUs.

Hierarchical kNN Graph Query: In a top layer, the `merge` operation can simply perform brute-force search between the s points per batch. With each lower layer, the number of batches grows with the factor g. To perform efficient searches for nearest neighbors in lower layers we perform a hierarchical query (Algorithm 4), which layer by layer utilizes the already merged upper layer to find entry points into the batches on the lower layers, and otherwise behaves just like the previously presented simple query (Section 4.1). The indices of the closest points found on the upper layer are then `transform`ed into the indices of the same points on the next lower layer and the search continues until the target layer l_m is reached. While distances to already visited points can be reused, their neighborhood on the lower layers changes. Therefore, we allow all points to be visited again after switching between layers.

As in HNSW [20], the hierarchy allows us to bridge gaps in the connectivity as multiple entry points from the top layer may approach the query on the bottom layer from different sides.

5.2 Graph Diversification

In order to reach any point from any direction, every edge of the graph, in principle, should be undirected. However, when every point should at least know its k_{nn} nearest neighbors, the total number of edges per point x would vary significantly for an undirected graph as the number of points which have x as their nearest neighbor will heavily depend on the local geometry.

In order to obtain a regular representation and to allow for simple parallelized traversal with constant workload per step, our graph only consists of exactly k directed (*i.e.*, outgoing) edges. We logically split the outgoing edges into at least $k_{\text{nn}} = k/2$ true nearest neighbors and up to $k_{\text{sym}} = k/2$ inverse links that are used to approximate an undirected graph.

As the number of representable inverse links is limited, one has to determine which of the inverse links are necessary. Harwood and Drummond [8] introduce the concept of shadowed links which are made redundant by the query's greedy exploration strategy. Optimizing the entire graph to remove all potentially shadowed edges requires a complex global optimization. The same holds for the construction of monotonic relative neighborhood graphs [26].

Our approach to graph diversification is to explicitly search for missing inverse links by querying for each point z from all its k_{nn} nearest neighbors $x_i \in \mathcal{N}_z^{nn}$ within a small search radius. Where a direct inverse link from x_i back to z exists, the query can trivially be skipped. When there is no path within the allowed range, the link is added (e.g., Figure 4). The searches are executed in parallel over all points $z \in \mathcal{X}$, allowing for faster construction.

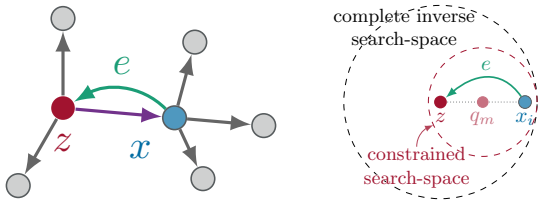


Fig. 4: Maintaining symmetric links. If there is no easily found connection from $x \in \mathcal{N}_z^{nn}$ back to $z \in \mathcal{X}$, the edge e is added to allow propagating nearest neighbor information between z and x. (right) Instead of considering the full search radius when probing whether to insert symmetric links, we constrain the maximum distance of points to be visited to the smaller sphere centered around the midpoint q_m.

The detailed symmetric linking operation is shown in Algorithm 5. Each point z launches queries for itself, starting from its k_{nn} nearest neighbors x_i. During the query, only the k_{nn} nearest neighbors of each visited point $\mathcal{N}_{x_i}^{nn}$ and potential already inserted symmetric/inverse links at that point $\mathcal{N}_{x_i}^{sym}$ are considered by the query as all others may still be overwritten. If a path back to z is found, the search continues with z's next neighbor. Otherwise, a link to z is inserted at the closest point to z encountered during the search which still has free capacities in its inverse link list.

To avoid race conditions on the insertions, we use atomics to keep track of the inverse link list sizes. On average, less than $k/4$ inverse links are necessary this way. If no candidate is found, the link is ignored. In our setting, this is, however, a very rare case

Algorithm 5: Symmetric linking query.

Function `graph.sym_links(z)`:
  Input: symmetric query point z
  for each $x_i \in \mathcal{N}_z^{nn}$:
    $cache$.`init(z, τ)`
    $cache$.`fetch(`x_i`)`
    while $(a \leftarrow cache$.`pop()`$) \neq \emptyset$:
      $\mathbf{p} \leftarrow \mathcal{N}_a^{nn} \cup \mathcal{N}_a^{sym}$
      if $z \in \mathbf{p}$:
        skip x_i
      $cache$.`fetch(`$\mathbf{p}$`)`
    for each $p \in cache.best$:
      if $|\mathcal{N}_p^{sym}| < k_{sym}$:
        $\mathcal{N}_p^{sym} \leftarrow \mathcal{N}_p^{sym} \cup \{z\}$
        break

and z might potentially still be reachable through either through hierarchy or multiple start points.

5.2.1 Additional Symmetric Linking Constraint

As demonstrated in Figure 4, the usual search-space would be defined depending on z and x_i. However, in order to generate a search structure that is able to find a path for all potential query points, we additionally constrain it on the worst-case query that still needs to find a path to z. Hence, a link is required to bridge the gap for a worst-case query on the midpoint q_m. Note, a path to x_i is given since it is directly known from z. For symmetric linking, only this search space is relevant, which is the circle defined by the center q_m and distance to x_i, but still encloses z. In practice, we set $q_m = z + 0.4 \cdot (x_i - z)$. This is slightly less conservative, but decreases the number of necessary symmetric links.

5.3 Graph Refinement

The initial graph construction might not always produce perfect results at the first trial. The reason for this is to some extent the large merge factor g that results when constructing search-graphs of low height that need to span over massive datasets.

In our experiments we found using a height of $L = 4$ for the hierarchical search-graph to provide the best performance. To create a single graph from the initially n/s disjoint sub-graphs consisting of single batches of size s (usually, $s = 32$), we need to merge groups of $g = \sqrt[L-1]{n/s}$ sub-graphs on each layer, where each new sub-graph's top-layer contains s points sampled from these g sub-graphs (see Section 5.1). While for $n = 256$, just 2 graphs need to be merged together each, for $n = 10^6$, about 32 graphs each need to be merged per layer. The higher g, the fewer entry points into the lower level exist, limiting the success rate of the initial hierarchical queries, as there are only s/g per merged sub-graph. For $g > s$, some points can only be reached after sub-graphs are bridged by symmetric linking.

To improve the graph quality, refinement steps, as described in Algorithm 6, can be performed which simply repeat the merging and symmetric linking steps throughout the search-graph. While refinement steps would also be beneficial during the actual sub-graph merging, we observed that performing them in the end is sufficient for the overall quality of the search structure.

Algorithm 6: Parallel graph refinement.

Function `RefineGraph(graph,` l_{top}`)`:
  Input: search structure `graph`, top layer l_{top}
  Result: refined search structure `graph`
  for $l_m \leftarrow l_{top} - 1$ **to** 0 :
    `graph.merge(`l_{top}, l_m`)`
    `graph.sym(`l_m`)`

5.4 Query Considerations

For our approach, during graph construction, it is crucial to perform a hierarchical query, which searches nearest neighbors layer by layer, due to the fact that the search index is not yet fully merged. Here, the coarse-to-fine methodology is able to bridge gaps between sub-graphs that are not yet connected and provides multiple routes to the area of interest. Starting from multiple spread-out points increases the quality significantly. In particular, if not only the 1-NN is of importance but also the neighbors at the far end of k. However, for a converged search-graph, this effort is no longer necessary as all points are interlinked well.

Searching in flat index structures does not have the hierarchical overhead. For example, Li *et al.* [21] start their query always from the centroid in a flat graph. Nevertheless, as the number of points grows, the number of hops to the centroid grows as well. In addition, with a single starting point, any goal point will be approached from just one direction. The search might be prone to gaps or linking problems inside the search-graph.

In our design, the bottom layer is actually a flat graph that has all points included. When searching the finished search-graph for nearest neighbors of novel points, it turned out to be more efficient in practice to skip the intermediate layers and continue the search directly on the bottom layer after exploring the s start points that make up the top layer. This allows to significantly cut back on the search iterations performed as otherwise the search would need to iterate on each intermediate layer. In practice, compared with the hierarchical approach, we did not observe any loss in recall but a significant saving in time.

5.5 Multi-GPU

The parallel construction and search algorithm can be extended to multiple GPUs. Johnson *et al.* [2] distinguish between two types of multi-GPU parallelism: *Replication* and *Sharding*. *Replication* copies the entire dataset $\mathcal{X}$ to multiple GPUs to parallelize the query process even further. The queries $\mathcal{Q}$ are divided equally among the available GPUs, resulting in linear speedups.

Sharding subdivides the dataset $\mathcal{X}$ into smaller, independent shards. This allows us to process billion-scale datasets which would otherwise not fit into GPU memory. When necessary, shards can also be swapped into main memory or onto disk. All GPUs construct the search-graphs for their shards in parallel. Since each shard has lower complexity due to the reduced number of points, super-linear speedups *w.r.t.* the same quality can be achieved.

The downside is that all shards need to be processed for each query. When querying multiple shards per GPU, we swap already processed shards for new ones in the background to hide memory latencies. Still, swapping shards puts a limit on the minimum run time per query batch. As some additional overhead, the individual query results per shard need to be merged. Within each GPU, we use a parallel radix sort. The final result across the GPUs is computed on the CPU using an efficient n-way merge.

6 Empirical Evaluation

In the following, we evaluate the performance of the proposed approach and its individual components on several publicly available benchmark datasets and report qualitative and quantitative results in terms of timings and accuracy.

Datasets. We ran experiments on datasets of varying sizes and dimensionality generated in different application contexts: SIFT1M [11] and SIFT1B [12] containing SIFT vectors of dimension 128, DEEP1B [5] with 1 billion 96-dimensional feature vectors encoding entire images, NyTimes [35], [36] and GloVe 200 [35], [37], which are two skewed and clustered datasets containig random projections of word embeddings compared using cosine-similarity, and finally the 960-dimensional dataset GIST [11]. Details on the datasets and construction details for our search graphs can be found in Table 4.

Hardware. We use a machine with 8x NVIDIA Tesla V100 and 2x Intel Xeon Gold 5218, where we usually use just a single GPU, except for SIFT1B and DEEP1B, where we use all 8 GPUs. We publish results for additional GPUs in the supplemental.

Performance Metrics. When comparing results to other approaches, one has to carefully look at the employed metric. The query performance is typically measured in recall (R@k). For quantization-based methods (e.g., [11]), the recall is understood as the ratio of queries which return the true first nearest neighbor among the first k results:

$$R@k = \frac{\left|\mathcal{N}_q^{\text{gt}}(1) \cap \mathcal{N}_q(k)\right|}{\left|\mathcal{N}_q^{\text{gt}}(1)\right|}. \tag{5}$$

For methods with exact distance computations – such as graph-based methods – the recall is instead understood as the overlap between the first k results and the first k true nearest neighbors. For disambiguation, we refer to it as consensus (C@k):

$$C@k = \frac{\left|\mathcal{N}_q^{\text{gt}}(k) \cap \mathcal{N}_q(k)\right|}{\left|\mathcal{N}_q^{\text{gt}}(k)\right|}. \tag{6}$$

As our algorithm uses exact distance calculations, the true nearest neighbor is either reported as the first element in an answer or not found at all. Therefore, we report only R@1 (=C@1).

Batch Size. The number of queries executed per batch is an important factor for GPU-based queries, which we analyze in Figure 8b. In our experiments, we measure the time for executing all 10000 queries (except 1000 queries for GIST [11]) in a batch and report the total execution time divided by the number of queries as µs/query or plot it as queries per second.

6.1 Performance Comparisons

Evaluation is performed on the default configuration GGNN, a faster, less accurate configuration GGNN‡, and a 8-GPU configuration GGNN^{8} (see Table 4 for parameters).

As reference, we use several recent CPU-based methods [20], [21], [26], where we executed the single-threaded reference implementations on an Intel Core i7-9700K. We further compare against the GPU-based SONG [3] and mimic their approach of querying the bottom layer of a prebuilt HNSW [20] graph using our query as "GGNN-HNSW". Using our query is even faster. In addition, it performs very similar to querying a similarly sized graph constructed using our method in significantly less time. Detailed performance comparisons are shown in Figure 5 and Tables 1, 2, and 3. In the tables, we compare against previously published results, where GPU-based methods [1], [2], [8], [18]

TABLE 1: Million-scale performance comparison.

Approach	τ_q	SIFT1M [11] Query time µs/query ↓	Recall @1 ↑	Recall @10 ↑	Recall @100 ↑
LOPQ [13]		51 100	0.51	0.93	0.97
IVFPQ [11]		11 200	0.28	0.70	0.93
FLANN [10]		5320	0.97	-	-
HM-ANN [27]		855	0.9995	-	-
		236	0.99	-	-
PQT [1]		20	0.51	0.83	0.86
FAISS [2]		20	0.80	0.88	0.95
PQFPGA [9]		20	0.88	0.94	0.97
FANNG [8]		1.3	0.95	-	-
		0.8	0.90	-	-
GGNN	0.65	5.8	**0.9997**	-	-
	0.42	1.5	0.99	-	-
	0.30	0.7	0.95	-	-
	0.20	**0.5**	0.90	-	-
Brute Force (1x V100)		381.4	1.0	-	-

TABLE 2: Performance comparison on SIFT1B [12]. Our method GGNN^{8} used 8 GPUs. RobustiQ [18] used 2 GPUs. Otherwise, single CPUs or GPUs were used. Our method achieves perfect recall@1. GGNN^{8} is limited by the host to device memory bandwidth and could otherwise perform the query with 99% recall@1 in just 24.5µs/query (see Section 6.3.1). The smaller GGNN^{8‡} can be queried much faster, since it iterates over fewer shards and fits on the GPUs in one piece.

Approach	τ_q	SIFT1B [12] Query time µs/query ↓	Recall @1 ↑	Recall @10 ↑	Recall @100 ↑
LOPQ [13]		8000	-	0.20	-
IVFPQ [11]		74 000	0.08	0.37	0.73
Multi-D-ADC [15]		1603	0.33	0.80	0.98
HM-ANN [27]		1014	0.99	-	-
PQT [1]		150	0.14	0.35	0.57
FAISS [2]		17.7	-	0.37	-
PQFPGA [9]		20	-	0.55	-
RobustiQ [18]		33	0.33	0.76	0.90
GGNN^{8}	0.56	61.6	**1.0**	-	-
	0.50	46.4	0.999	-	-
	0.38	42.7	0.99	-	-
GGNN^{8‡}	0.61	7.1	0.99	-	-
	0.42	**3.9**	0.90	-	-
Brute Force (8x V100)		≈30 000	1.0	-	-

TABLE 3: Performance comparison on DEEP1B [5]. Our method GGNN^{8} used 8 GPUs. FAISS [2] used 4 GPUs. RobustiQ [18] used 2 GPUs. Otherwise, single CPUs were used. GGNN^{8} is limited by the host to device memory bandwidth to at least 233µs/query on the 10k queries set (see Section 6.3.1). To show the influence of τ_q and to compare with future hardware configurations, we report times without the currently necessary upload as GGNN^{8*}.

Approach	τ_q	DEEP1B [5] Query time µs/query ↓	Recall @1 ↑	Recall @10 ↑	Recall @100 ↑
Multi-D-ADC [15]		1065	0.37	0.74	0.92
HM-ANN [27]		1142	**0.99**	-	-
FAISS [2]		13.3	0.45	-	-
RobustiQ [18]		30	0.38	0.75	0.89
GGNN^{8*}	1.50	27.6	**0.99**	-	-
	0.44	8.3	0.95	-	-
	0.36	**7.2**	0.90	-	-

TABLE 4: Datasets and search graph construction details. For some datasets, we also demonstrate the construction-query trade-off with a faster construction version (‡). All configurations shown here are capable of reaching at least 99% recall@1.

Dataset			Search Graph Configuration						Construction Time and Graph Size			
Dataset	#Points	#Dim.	Config.	k	s	l	τ_b	r	GPU		Time	Size
SIFT1M [11]	1 000 000	128	GGNN‡	24	32	4	0.5	2	1x V100		9.2 s	95 MiB
			GGNN	40	32	4	0.5	2			21.8 s	158 MiB
NyTimes [35], [36]	290 000	256	GGNN‡	40	32	4	0.3	0			13.4 s	46 MiB
			GGNN	96	64	4	0.3	0			39.3 s	113 MiB
GloVe 200 [35], [37]	1 183 514	200	GGNN	96	64	4	0.5	2			6.0 min	450 MiB
GIST [11]	1 000 000	960	GGNN	96	64	4	0.4	2			11.9 min	382 MiB
SIFT1B [12]	1 000 000 000	128	GGNN^{8‡}	20	32	4	0.5	2	8x V100	16 shards	33.4 min	75 GiB
			GGNN^{8}	40	32	4	0.5	2		128 shards	78.6 min	152 GiB
DEEP1B [5]	1 000 000 000	96	GGNN^{8}	24	32	4	0.5	2		32 shards	47.5 min	90 GiB

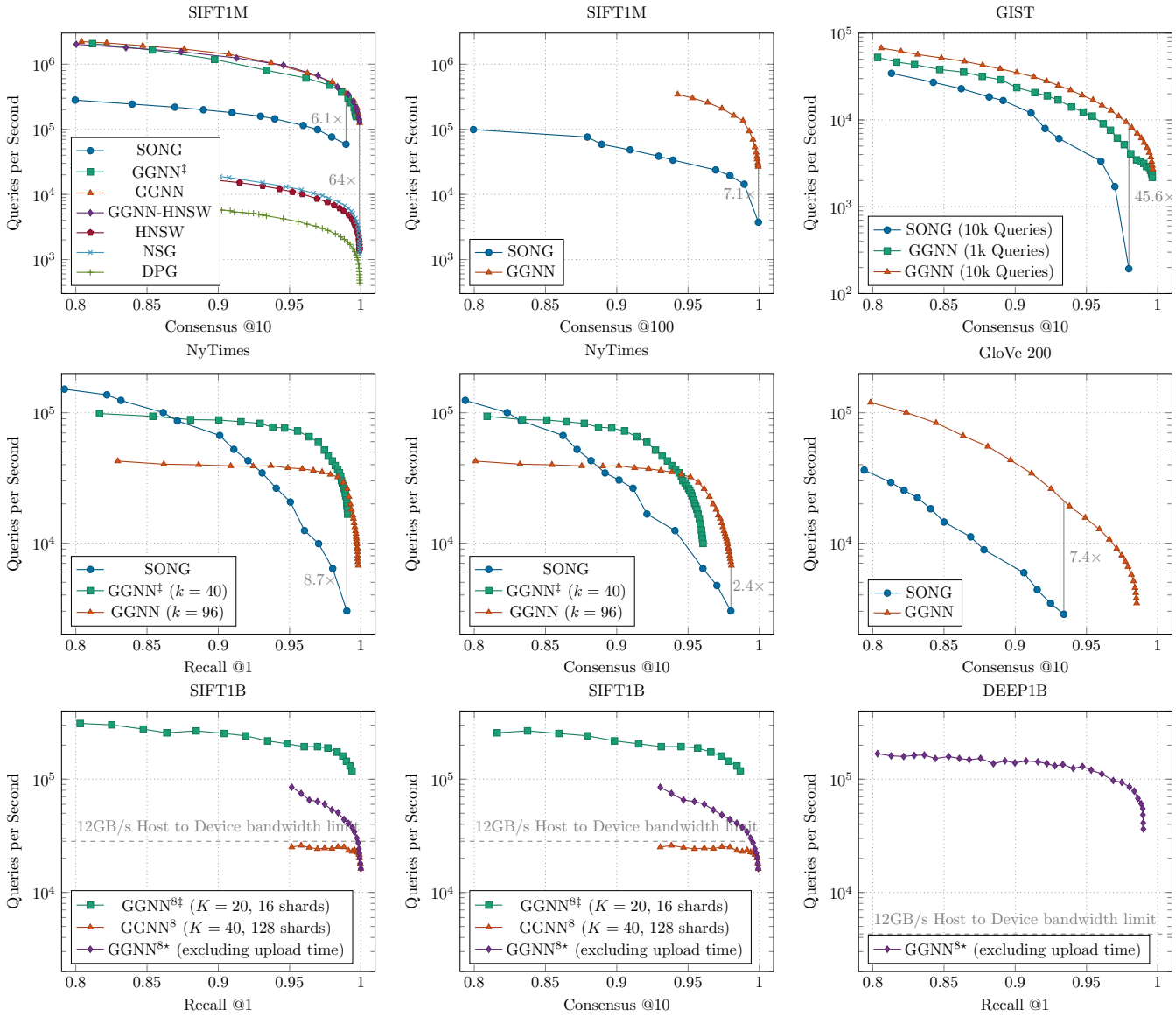


Fig. 5: Performance comparison across various datasets and top-k queries. Results in the top-right corner are preferable. We compare mainly against SONG [3], using the values from their paper and using the same GPU. On SIFT1M, we additionally compare against state-of-the-art single-core CPU-based methods [20], [21], [26] and include a configuration "GGNN-HNSW", where we use our method to query the graph built by HNSW [20], like SONG [3] does. Our query significantly outperforms SONG and achieves similar results as a query on a similarly-sized search-graph constructed in much less time using our method. High performance is also achieved when querying for the 100 nearest neighbors or more complicated datasets like GloVe 200 or very high dimensional datasets like GIST. On the rather small NyTimes dataset, our graph construction appears to be non-optimal but the highly efficient query still manages to outperform SONG by a large factor. Using 8 GPUs, even billion-scale datasets like SIFT1B or DEEP1B can be queried at rates around 100k queries per second with near-perfect recalls.

used NVIDIA GTX Titan X GPUs, and [9] used an Arria 10 GX1150 FPGA. Besides being significantly faster on all datasets (around 1µs per query on SIFT1M), our method can also achieve very high recall rates, close to perfect.

6.2 Search-Graph Construction

In the following, we inspect several aspects of the construction process, including the time spent on the individual construction steps, the influence of the dataset size, the resulting graph quality in terms of nearest neighbors per point, and how additional construction time can be traded in for even faster queries.

6.2.1 Construction Time

Construction times and index sizes of our method are listed in Table 4. For comparison, FAISS [2] report construction times between 4 and 24 hours for DEEP1B, yet reaches less than 50% R@1. HM-ANN [27] report construction times around 100 hours for effective billion-scale search-graphs. Using our hierarchical GPU-based graph-merge algorithm, 99% R@1 on DEEP1B and even perfect 100% R@1 SIFT1B can be reached with fast queries while the construction takes less than 2 hours.

6.2.2 Construction Time Composition

Figure 6 shows the time spent on the individual graph-construction operations with 2 refinement iterations. The majority of the construction time is spent on the `merge` operation, especially those merges which involve the bottom layer, where the merge operation searches for the k nearest neighbors for all points in the dataset.

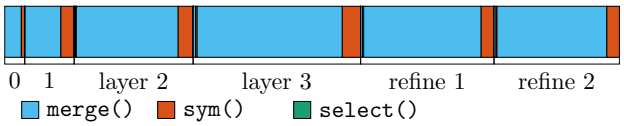


Fig. 6: Split-up of the construction time for the SIFT1M search graph. Most time is spent on the `merge` operation (86.92%), some time on the `sym` operation (13.06%) and almost no time on `select`ing the points for the upper layers (0.02%).

6.2.3 Dataset Size

Efficient graph construction needs to scale well with the size of the dataset. As demonstrated in Figure 7, the construction time scales almost linearly with the size of the dataset, $\approx O(n^{1.077})$. This can be explained by most of the construction time being spent on independently determining the neighbors of each bottom-level point (see Section 6.2.2). This almost linear behavior is significantly better than the complexity of brute-force kNN construction $O(n^2)$.

6.2.4 For-all Queries

The fast construction of the kNN search graph from scratch solves another interesting problem on the side, namely computing the k nearest neighbors for all points in the dataset, as frequently encountered in n-body problems. When we construct the full hierarchy, the final merging step effectively searches for the k neighbors of all points. In Table 5, we demonstrate the quality of the for-all problem considering the consensus. Referring to Figure 7, this for-all problem again shows almost a linear behavior. In case of sharding, though, all points would need to be tested once against every shard. As the number of shards linearly depends on the number of points one approaches $O(n^2)$ complexity again with a tiny constant though, e.g. 16 shards for the SIFT1B dataset.

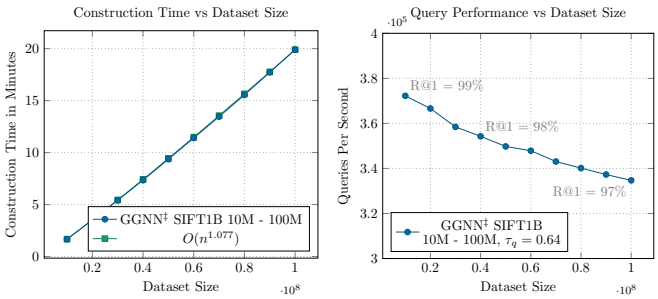


Fig. 7: Construction times and queries per second for increasing subsets of SIFT1B (up to 100M) on a single GPU. The construction time almost linearly depends on the dataset size. With increasing dataset size, the query performance suffers slightly in both execution time and accuracy for a constant slack factor τ_q.

TABLE 5: Accuracy of the nearest neighbors for each point in the dataset, as contained within our search graph.

Method	Dataset	C@1 ↑	C@10 ↑
GGNN‡	SIFT1M	0.996	0.981
GGNN		0.999	0.998

6.2.5 The Construction-Query Trade-off

Our graph construction algorithm and the query can be tuned for speed or accuracy. The build time is controlled by the slack factor τ_b and by how many refinement iterations r are carried out. The longer the construction time, the more precise the resulting graph. A quickly assembled graph will have worse edges and a query might need to visit more nodes until fulfilling the stopping criterion. In Figure 8a, the trade-off between build time and query time is visualized for fixed recall rates.

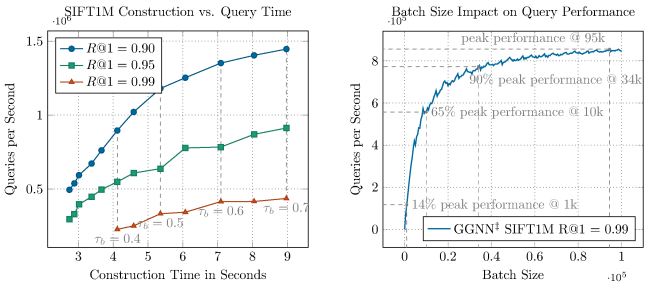
6.3 Query Behaviour

The runtime and the quality of a query can be controlled by adapting the stopping criterion through the slack factor τ_q. As can be seen e.g. in Table 1, by increasing τ_q, a larger safety margin is considered during query, resulting in better recall rates at the cost of visiting more points and consequently longer query time.

It is instructive to look at the behaviour of the query over time. In Figure 9, the initial distance of the query to the starting point, i.e. the closest point on the top layer, is drastically reduced already after very few iterations. Here, an iteration stands for fetching the best point from the priority queue and calculating the distance to all its neighbors. Most time is spent to carefully explore the neighborhood around the true nearest neighbor. Quickly after locating the best candidates, the query is terminated by the stopping criterion to prevent unnecessary iterations. This effect does not only show up in the small subset of plotted query distances, but also statistically in the histogram over all queries.

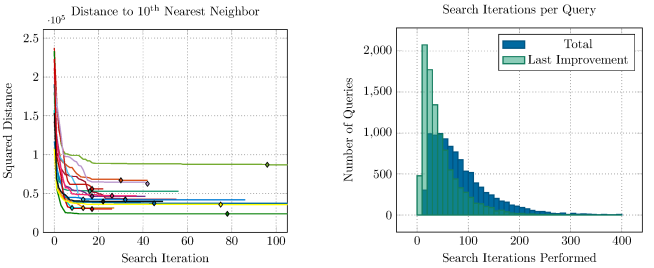
6.3.1 Out of GPU-memory Queries

When querying on billion-scale datasets, the amount of necessary GPU-memory will quickly exceed the available memory even on an 8 NVIDIA Tesla V100 system. While SIFT1B together with the low-profile graph GGNN^{8‡} still fits on the GPUs, with the GGNN^8 graph, it exceeds the available memory by 2 shards (4.2 GiB). On DEEP1B, the dataset alone already surpasses the available memory, and with GGNN^8 an additional 28 GiB per GPU need to be loaded per query pass. This results in a bound that is defined



(a) Varying construction time. **(b)** Varying batch size.

Fig. 8: Query performance depends both on the quality of the search-graph and the number of queries executed simultaneously. Queries with the same accuracy can be executed faster if more time is spent on construction (a). Here, no refinement is performed ($r = 0$) to visualize the influence of τ_b on the initial construction. Generally, having a few refinement iterations can further boost query performance. In terms of query size (b), high performance is already reached for a few thousand queries, while further simultaneous queries yield even better performance with a peak at 95k queries.



(a) Development of the 10^{th} best query point as the query progresses. **(b)** Iterations performed total, and until the last improvement is found.

Fig. 9: Looking at the behavior of several queries (a), one can see that the general area is found very quickly by the search graph. Afterwards, the local neighborhood is explored and shortly after finding the last improvement (indicated by the marker), the query is terminated by the stopping criterion. The effectiveness of the stopping criterion also shows in the statistics (b). The distribution of total iterations closely follows the distribution of iterations necessary to make the final improvement, only lagging behind by a handful of iterations.

by the host to device memory throughput ($\approx$ 12 GiB/s [38]), *i.e.*, for 10k queries, this results in a minimum of 35µs/query on SIFT1B and 233µs/query on DEEP1B. These bounds are shown in Figure 5. Note that this loading time can be hidden completely by performing an overall costlier query, e.g. using more queries, increased τ_q, or larger caches. For easy comparison with future approaches on devices with sufficient memory, we also report the computation time without the memory transfers.

7 Conclusion

Our proposed parallel GPU-based search algorithm is surpassing all state-of-the-art algorithms in terms of speed by a large factor and represents the fastest ANN query technique while maintaining recall rates above 99%. The efficient query algorithm further accelerates the parallel construction and merging of sub-graphs. Our hierarchical construction scheme creates high-quality kNN graphs with graph-diversification links for very efficient traversal. It is easily deployed in multi-GPU systems to cope with very large-scale datasets. It is the first ANN search method capable of constructing effective search structures (R@1 $\geq$ 0.99) for billion-scale datasets in less than an hour. For million-scale datasets, search-graphs of sufficient quality can often be constructed within seconds, allowing for quick nearest-neighbor searches also on intermediate data structures as a part of larger GPU-based algorithms, e.g. in machine learning applications.

Currently, our method computes the exact distance to all visited high-dimensional points. As the number of distance calculations dominates the query time, a compressed representation of the data vectors could lead to further acceleration.

Acknowledgments

This work has been partially funded by the Deutsche Forschungsgemeinschaft (DFG, German Research Foundation) under Germany's Excellence Strategy – EXC number 2064/1 – Project number 390727645 and SFB 1233, TP 02 – Project number 276693517. It was supported by the German Federal Ministry of Education and Research (BMBF): Tübingen AI Center, FKZ: 01IS18039A.

References

[1] P. Wieschollek, O. Wang, A. Sorkine-Hornung, and H. Lensch, "Efficient large-scale approximate nearest neighbor search on the gpu," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 2027–2035. 1, 2, 8

[2] J. Johnson, M. Douze, and H. Jégou, "Billion-scale similarity search with gpus," *IEEE Transactions on Big Data*, 2019. 1, 2, 7, 8, 10

[3] W. Zhao, S. Tan, and P. Li, "Song: Approximate nearest neighbor search on gpu," in *2020 IEEE 36th International Conference on Data Engineering (ICDE)*, 2020, pp. 1033–1044. 1, 3, 8, 9

[4] R. Weber, H.-J. Schek, and S. Blott, "A quantitative analysis and performance study for similarity-search methods in high-dimensional spaces," in *Proceedings of the 24rd International Conference on Very Large Data Bases (VLDB)*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 1998, p. 194–205. 1

[5] A. Babenko and V. S. Lempitsky, "Efficient indexing of billion-scale datasets of deep descriptors," *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 2055–2063, 2016. 1, 8, 9

[6] J. Bobadilla, F. Ortega, A. Hernando, and A. Gutiérrez, "Recommender systems survey," *Knowledge-based systems*, vol. 46, pp. 109–132, 2013. 1

[7] W. Chen, J. Chen, F. Zou, Y.-F. Li, P. Lu, Q. Wang, and W. Zhao, "Vector and line quantization for billion-scale similarity search on gpus," *Future Generation Computer Systems*, vol. 99, p. 295–307, Oct 2019. 1

[8] B. Harwood and T. Drummond, "Fanng: Fast approximate nearest neighbour graphs," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 5713–5722. 1, 2, 5, 6, 8

[9] J. Zhang, S. Khoram, and J. Li, "Efficient large-scale approximate nearest neighbor search on opencl fpga," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 4924–4932. 1, 8, 10

[10] M. Muja and D. G. Lowe, "Scalable nearest neighbor algorithms for high dimensional data," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 36, 2014. 1, 2, 8

[11] H. Jégou, M. Douze, and C. Schmid, "Product quantization for nearest neighbor search," *IEEE Transactions on Pattern Analysis and Machine Intelligence (PAMI)*, vol. 33, no. 1, pp. 117–128, 2011. 1, 2, 8, 9

[12] H. Jégou, R. Tavenard, M. Douze, and L. Amsaleg, "Searching in one billion vectors: Re-rank with source coding," in *2011 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2011, pp. 861–864. 1, 2, 8, 9

[13] Y. Kalantidis and Y. Avrithis, "Locally optimized product quantization for approximate nearest neighbor search," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. Columbus, Ohio: IEEE, June 2014. 1, 2, 8

[14] T. Ge, K. He, Q. Ke, and J. Sun, "Optimized product quantization for approximate nearest neighbor search," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE Computer Society, June 2013. 1, 2

[15] A. Babenko and V. S. Lempitsky, "The inverted multi-index," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. IEEE, 2012, pp. 3069–3076. 1, 2, 8

[16] A. Babenko and V. Lempitsky, "Tree quantization for large-scale similarity search and classification," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 4240–4248. 1, 2

[17] A. Babenko and V. Lempitsky, "Additive quantization for extreme vector compression," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2014, pp. 931–938. 1, 2

[18] W. Chen, J. Chen, F. Zou, Y.-F. Li, P. Lu, and W. Zhao, "Robustiq: A robust ann search method for billion-scale similarity search on gpus," in *Proceedings of the International Conference on Multimedia Retrieval (ICMR)*, 2019, pp. 132–140. 1, 2, 8

[19] Y. Matsui, T. Yamasaki, and K. Aizawa, "Pqtable: Fast exact asymmetric distance neighbor search for product quantization using hash tables," *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1940–1948, 2015. 1, 2

[20] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE transactions on pattern analysis and machine intelligence*, 2018. 1, 3, 5, 6, 8, 9

[21] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin, "Approximate nearest neighbor search on high dimensional data-experiments, analyses, and improvement," *IEEE Transactions on Knowledge and Data Engineering*, 2019. 1, 5, 7, 8, 9

[22] W. Dong, C. Moses, and K. Li, "Efficient k-nearest neighbor graph construction for generic similarity measures," in *Proceedings of the 20th international conference on World wide web*. ACM, 2011, pp. 577–586. 1, 2, 3

[23] J. Chen, H.-r. Fang, and Y. Saad, "Fast approximate knn graph construction for high dimensional data via recursive lanczos bisection," *Journal of Machine Learning Research*, vol. 10, no. Sep, pp. 1989–2012, 2009. 1, 2

[24] T. Warashina, K. Aoyama, H. Sawada, and T. Hattori, "Efficient k-nearest neighbor graph construction using mapreduce for large-scale data sets," *IEICE Transactions*, vol. 97-D, pp. 3142–3154, 2014. 1, 2

[25] C. Fu and D. Cai, "Efanna: An extremely fast approximate nearest neighbor search algorithm based on knn graph," *arXiv preprint arXiv:1609.07228*, 2016. 1, 2

[26] C. Fu, C. Xiang, C. Wang, and D. Cai, "Fast approximate nearest neighbor search with the navigating spreading-out graph," *Proceedings of the VLDB Endowment*, vol. 12, no. 5, pp. 461–474, 2019. 1, 2, 3, 5, 6, 8, 9

[27] J. Ren, M. Zhang, and D. Li, "Hm-ann: Efficient billion-point nearest neighbor search on heterogeneous memory," in *34th Conference on Neural Information Processing Systems (NeurIPS 2020)*, November 2020. 1, 2, 3, 8, 10

[28] J. H. Friedman, J. L. Bentley, and R. A. Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Trans. Math. Softw.*, vol. 3, no. 3, pp. 209–226, Sep. 1977. 2

[29] M. Datar, N. Immorlica, P. Indyk, and V. S. Mirrokni, "Locality-sensitive hashing scheme based on p-stable distributions," in *Proceedings of the Twentieth Annual Symposium on Computational Geometry*, ser. SCG '04. New York, NY, USA: ACM, 2004, pp. 253–262. 2

[30] A. Andoni and P. Indyk, "Near-optimal hashing algorithms for approximate nearest neighbor in high dimensions," *Commun. ACM*, vol. 51, no. 1, p. 117, Jan. 2008. 2

[31] S. Korman and S. Avidan, "Coherency Sensitive Hashing," *Proceedings of the IEEE International Conference on Computer Vision (ICCV)*, pp. 1607–1614, Nov. 2011. 2

[32] D. S. Kim and N. B. Shroff, "Quantization based on a novel sample-adaptive product quantizer (sapq)." *IEEE Transactions on Information Theory*, vol. 45, no. 7, pp. 2306–2320, 1999. 2

[33] Y. Linde, A. Buzo, and R. Gray, "An algorithm for vector quantizer design," *IEEE Transactions on Communications*, vol. 28, no. 1, pp. 84–95, January 1980. 2

[34] P. S. Efraimidis and P. G. Spirakis, "Weighted random sampling with a reservoir," *Information Processing Letters*, vol. 97, no. 5, pp. 181 – 185, 2006. 6

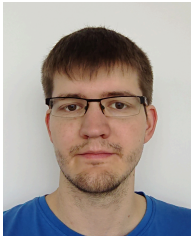
[35] M. Aumüller, E. Bernhardsson, and A. Faithfull, "Ann-benchmarks: A benchmarking tool for approximate nearest neighbor algorithms," 2018. 8, 9

[36] D. Dua and C. Graff, "UCI machine learning repository," 2017. [Online]. Available: http://archive.ics.uci.edu/ml 8, 9

[37] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Empirical Methods in Natural Language Processing (EMNLP)*, 2014, pp. 1532–1543. 8, 9

[38] Z. Jia, M. Maggioni, B. Staiger, and D. P. Scarpazza, "Dissecting the nvidia volta gpu architecture via microbenchmarking," *arXiv preprint arXiv:1804.06826*, 2018. 11

Fabian Groh received his diploma in computer science in 2013 from Ulm University, Germany. He is working towards a PhD degree at the computer graphics group of the University of Tübingen. His primary research topics are processing and learning on unstructured data in large-scale settings by utilizing GPU-based designs. In 2020 he joined Amazon Tübingen.



Lukas Ruppert received his master of science in computer science in 2019 from University of Tübingen. He is working on his Ph.D. in computer graphics with a focus on physically based rendering.

Patrick Wieschollek received his master of science in 2014 from Friedrich-Schiller University, Germany. He wrote his PhD thesis at the computer graphics group of the University of Tübingen and Max Planck Institute for Intelligent Systems in Tübingen. His primary research topic is developing data-driven approaches for multi-frame and unstructured data in large-scale settings. In 2019 he joined Amazon Tübingen.

Hendrik Lensch Hendrik P. A. Lensch holds the chair for computer graphics at Tübingen University. He received his diploma in computers science from the University of Erlangen in 1999. He worked as a research associate at the computer graphics group at the Max-Planck-Institut für Informatik in Saarbrücken, Germany, and received his PhD from Saarland University in 2003. Hendrik Lensch spent two years (2004-2006) as a visiting assistant professor at Stanford University, USA. From 2009 to 2011 he was a full professor at the Institute for Media Informatics at Ulm University, Germany. In his career, he received the Eurographics Young Researcher Award 2005 and was awarded an Emmy Noether Fellowship by the German Research Foundation (DFG) in 2007. His research interests include 3D appearance acquisition, computational photography, machine learning, global illumination and image-based rendering, and massively parallel programming.

GGNN: Graph-based GPU Nearest Neighbor Search

Fabian Groh, Lukas Ruppert, Patrick Wieschollek, and Hendrik P.A. Lensch

Appendix A
Hierarchical Construction Example

To visualize the individual steps in the construction of the hierarchy, Figure 1 shows an example with growth rate $g = 3$ where layer 1 has already been constructed by merging 9 bottom layer batches into 3 sub-graphs of height 2. To merge the individual sub-graphs further, first, s points are selected equally from the current top layers of the g sub-graphs, creating a new top layer (`select`). Starting from the top, layer by layer, a hierarchical query for the k nearest neighbors of all points in the current layer is performed which can access all g sub-graphs via the new top layer. (On the new top layer, the k nearest neighbors can again be determined efficiently using brute-force.) The kNN lists within the layer are then updated using the result of these queries (`merge`). Afterwards, symmetric links are inserted (`sym`). Once all layers are processed, the g sub-graphs form one consistent search-graph, where each point can potentially reach all others.

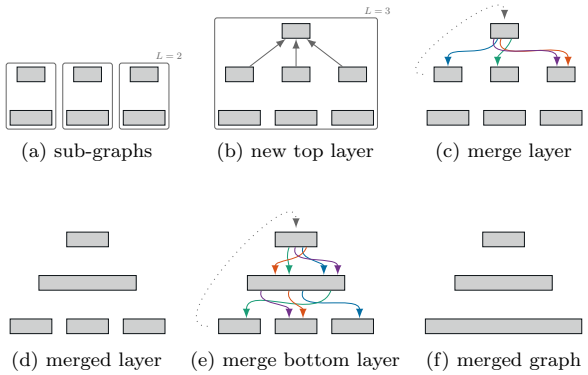


Fig. 1: Bottom-up-top-down construction of our graph hierarchy. a) Disjoint batches/sub-graphs provide a few sampled points which are fused into a common new top layer b). For the top layer, a symmetric kNN-graph is constructed by brute force. c) The next finer layer is fused by carrying out hierarchical kNN-searches from the top which will find and link neighborhoods across sub-graphs, d) establishing one consistent kNN-layer. This process of e) finding the nearest neighbors for all points in a layer and f) fusing this layer across the sub-graphs is recursively applied until all initial shards are merged.

- *During the work of this paper, all authors were member of the Computer Graphics group at the University of Tübingen, Germany.*
- *Patrick Wieschollek and Fabian Groh are now with Amazon, Tübingen, Germany. This work has been done prior to joining Amazon.*

Appendix B
Parallelization of Graph-based methods

Graph-based methods receive impressive recall numbers even for high-dimensional datasets, e.g. HNSW [1]. Nonetheless, there is a caveat to it. Every comparison between two points needs to load the complete vector from memory. Therefore, graph-based methods require a huge memory throughput, that gets especially prominent when considering parallelization. In contrast, quantization-based methods are already utilizing the huge advantages in memory throughput offered by GPUs, *e.g.*, 900 GB/s NVIDIA Tesla V100 vs. 76.8 GB/s Intel Xeon E5-2650 v4. Figure 2 demonstrates the limits of CPU-based parallelization approaches. Hence, our work focuses on enabling graph-based kNN structures on GPUs, where we do not have the luxury of cheap global or sequential update steps, excessive resources, nor simple dynamic neighborhood lists.

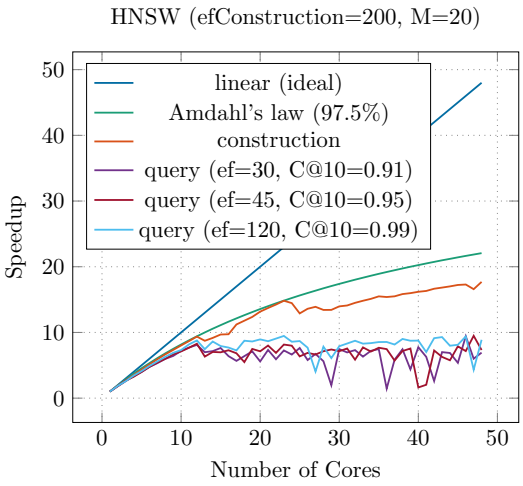


Fig. 2: Parallelization of the HNSW algorithm on two Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz (2x12 cores with 24 threads each). First, all 12 physical cores on the first CPU were used, then the 12 cores of the second CPU were added. Finally, the logical cores were added. Speedup is computed as the execution time on a single core divided by the parallelized time. The trendline using Amdahl's law has been fitted optimistically.

TABLE 1: Query configurations for the NVIDIA Tesla V100 (CUDA compute capability 7.0).

Dataset	$best$	$prioq$	$visited$	max. iter.	block size	registers	shared memory	occupancy
SIFT1M	10	54	384	400	32	56	2200‡ / 2232	0.5
	100	28	896	1000	64	48	4796	0.625
NyTimes	10	22	2016	2000	128	51	8536‡ / 8760	0.5625
GloVe 200	10	22	2016	2000	128	51	8760	0.5625
GIST	10	22	2016	2000	128	56	8740	0.5625
SIFT1B DEEP1B	10	54	384	400	32	56	2200‡ / 2232 2200	0.5

APPENDIX C
CHOOSING PARAMETERS

Our method can be tuned for optimal performance using various parameters. In the following, we provide guidelines on choosing good parameters for the construction and query process.

C.1 Choosing Search-Graph Construction Parameters

Our search graph construction process can be tuned by the following parameters:

$k = |\mathcal{N}_x|$ selects the number of neighbors, i.e., outgoing links, per point in the graph and is best kept low to prevent over-exploration during query. For highly skewed or very high-dimensional datasets, large neighborhoods seem necessary to bridge gaps and achieve high recalls.

s is the segment size in which points are grouped and nearest neighbors are determined in a brute-force fashion to fill the initial neighbor lists. As a result, $s > k/2$ is a requirement. Simultaneously, it is also the size of the top-layer which is used as the set of starting points for each query. Using multiples of 32 makes good use of the GPU's resources which always executes 32 threads simultaneously in a *warp*.

l chooses the number of layers in the hierarchy. Larger hierarchies can bridge wider gaps in the dataset more easily, but don't help much in the case of individual outliers. The larger l, the longer the construction will take. Empirically, we found $l = 4$ to be the ideal choice for most datasets.

r chooses the number of refinement iterations which repeat the merging and symmetric linking steps performed during construction. For most datasets, using $r = 2$ can significantly boost query performance, while additional refinements do not result in further improvements of query performance. For the NyTimes dataset, we did not observe any improvement, most likely due to its comparably small size.

C.1.1 Graph Size

In some applications, the size of the search graph can be a limiting factor. Especially for larger datasets, it can become problematic to fit the dataset and the search graph onto the GPU. In such cases, we resort to conventional sharding.

To determine the precise memory requirements, one must first determine the exact integer growth factor g and the resulting number of points on the upper layers N_{upper}. There are $N_{\text{upper}} = s\sum_{i=0}^{l-2} g^i$ points in the upper layers of the hierarchy where the growth factor $g \approx \sqrt[l-1]{\frac{N}{s}}$ may be rounded up or down such that the bottom-level segment size $s_0 \approx \frac{N}{g^{l-1}}$ is closest to s except to ensure that $s_0 > k$.

To determine the memory required by the search-graph, one can then simply sum up the following:

- $(N + N_{\text{upper}}) \cdot k \cdot 4$ `Bytes` for the graph's edge list.
- $2 \cdot N_{\text{upper}} \cdot 4$ `Bytes` for translating higher-level indices into the level below and into the base-level indices.
- $2 \cdot 4$ `Bytes` for $d^{+}_{\text{nn}_1}$ and $\bar{d}_{\text{nn}_1}$.

As an example, for SIFT1M with $k = 24$, the graph takes the following amount of memory: ($g = 32$)

$$(1033824 \cdot 24 + 2 \cdot 33824 + 2) \cdot 4\,\texttt{B} \;\approx\; 94.9\,\texttt{MiB}$$

During construction, additional temporary memory is required:

- $N \cdot k \cdot 4$ `Bytes` for temporary edge lists.
- $N \cdot 4$ `Bytes` for random numbers used for selection.
- $N \cdot (k_{sym} + 1) \cdot 4$ `Bytes` for temporary inverse link lists.
- $N \cdot 4$ `Bytes` for the nearest neighbor distance per point.
- temporary storage for reduction using CUB

As not all of these are used simultaneously, about $N \cdot (k + 1) \cdot 4$ `Bytes` suffice in practice, which is roughly as much as is needed for the graph itself.

C.2 Choosing Query Parameters

To make the most use of the GPU's resources, the parameters of the query kernel can be carefully tuned. The query uses *registers* to cache the query vector for efficient distance computations, whereas our cache structure resides in *shared memory*.

As discussed in the paper, we can afford to use $best$ lists as short as the number of neighbors to be searched for. However, we always search for at least 10 neighbors. This comes at negligible additional cost for queries for fewer neighbors but keeps the stopping criterion stable.

When aiming for high accuracy, it turned out that large $visited$ lists are of utmost importance, as most time is spent to carefully explore the local neighborhood of a query. Being able to enumerate each visited point makes sure that the query does not perform any cycles. We also limit the maximum number of iterations performed by the query such that the visited list is not exceeded by more than a few points at most.

Concerning the $prioq$, we found that using larger priority queues is only beneficial up to a point and often very short priority queues perform surprisingly well, allowing us to only cache as little as 32 or 64 distances in our queries.

Finally, the *block size* can be tuned to achieve higher *occupancy* on the GPU when more registers are needed, e.g., for high-dimensional datasets, or more shared memory, when using a large cache. The ideal values will vary somewhat depending on the used GPU but small block sizes (e.g., 32) perform better during the reduction-based distance computation, especially for lower-dimensional (e.g., 128) datasets. For more costly queries, however, larger block sizes may be necessary to achieve higher occupancy which allows to hide latencies when accessing data in the GPU's main memory.

C.2.1 Used Query Parameters

The detailed query parameters used for evaluating the performance of our method on each dataset can be found in Table 1. This includes the GPU-specific register and shared memory usage, as well as the resulting occupation on the device.

We use the same parameters across the more complicated datasets. For SIFT1M, a cheaper query suffices. The query for GIST needs a few more registers to store the high-dimensional query vector. The queries for NyTimes and GloVe 200 have some shared memory overhead due to the cosine-similarity distance metric. There is almost no overhead due to the dataset size, so even the queries on SIFT1B and DEEP1B can use the same configuration as the query on SIFT1M with the same resulting register and shared memory usage. (There is a minor overhead when dealing with billion-scale datasets due to the necessity of 64-bit indices when addressing base-vectors or neighbors in the graph.) In all cases, our query reaches an occupancy of at least 50%, which allows to effectively hide memory-access latencies.

APPENDIX D
PERFORMANCE ON DIFFERENT GPUS

In the main paper, we only show results using the NVIDIA Tesla V100, which is a high-end datacenter GPU. Figure 3 demonstrates the performance of our method on a wide range of GPUs, including the low-end GT 1030. All GPUs used the same query configuration shown in Table 1. On the GT 1030 and GTX 1080 Ti, increasing the blocksize to 64 resulted in slightly faster queries.

On all GPUs, our method performs equally well in terms of quality but the number of queries per second depends on the memory bandwidth and computational power that the GPU is able to provide. E.g., with about 5% of the V100's memory bandwidth, the GT 1030 is also only able to provide about 5% of the query performance. The construction time is similarly impacted, as shown in Table 2, which also shows the memory bandwidth of the compared GPUs.

Finally, the consistent query quality across the GPUs also demonstrates the stability of the search-graph construction process, despite its largely independent construction and random selection steps. While the shown results are each based on different, newly constructed search-graphs, their consensus at the fixed slack factor (τ_q) intervals is almost identical.

TABLE 2: Construction times for the GGNN search-graph for SIFT1M on various GPUs.

GPU	Memory Bandwidth	Construction Time
RTX 3090	936 GB/s	21.7 s
Tesla V100	900 GB/s	21.8 s
Titan RTX	672 GB/s	26.7 s
RTX 2080 Ti	616 GB/s	29.3 s
GTX 1080 Ti	484 GB/s	59.1 s
GTX Titan X	336 GB/s	102.1 s
GT 1030	48 GB/s	586.8 s

APPENDIX E
QUERY STARTING POINTS

Every graph based search needs to start on at least one point in the graph. Our algorithm uses all the points at the top layer as **starting points**. These points are well connected in the graph due to the construction design, which is of particular interest during construction itself. However, In a well converged flat search-graph, the actual starting points are not that important in terms of quality. The option to start from the **centroid** point is proposed by Li *et al*. [2]. We conducted an experiment to compare the different methods. Additionally, we also test the performance when a search starts at the furthest point from the centroid, which is denoted as **worst**. The centroid or worst point is cheaply determined during construction with no measurable overhead. The experiment is performed on a lower spec NVIDIA GTX Titan X GPU, compared to the main paper. Table 3 shows the results for the 1, 2, and 5 million point subsets of the SIFT1B dataset. While c@10 is on par for all query start point options, we see some improvements in terms of speedup when using our **starting points**.

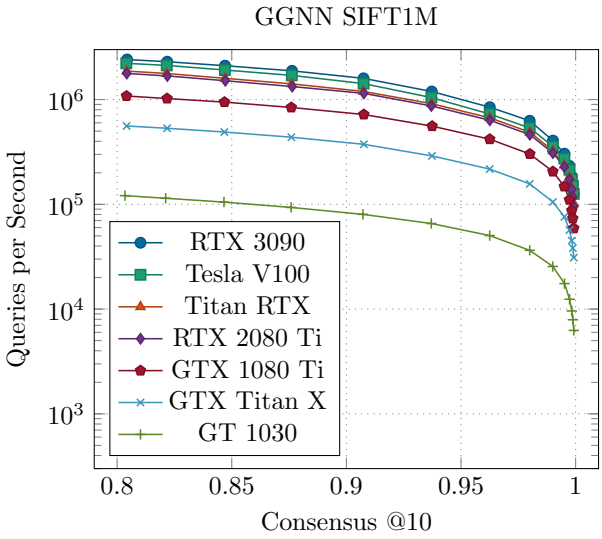


Fig. 3: Performance comparison across different GPUs.

TABLE 3: Comparison of query start options for different slack factors τ and subsets of the SIFT1B. Execution times are evaluated on an NVIDIA GTX Titan X GPU.

τ	starting points		centroid point		worst point		SIFT1B
	c@10	us/query	c@10	us/query	c@10	us/query	subset
0.6	0.983	7.4	0.983	8.5	0.983	8.5	1M
	0.978	8.1	0.978	9.3	0.978	9.3	2M
	0.969	8.8	0.968	10.3	0.969	10.3	5M
0.7	0.993	12.2	0.994	13.7	0.993	13.7	1M
	0.990	13.2	0.990	14.9	0.990	14.8	2M
	0.985	14.0	0.984	16.1	0.985	17.0	5M

APPENDIX F
IMPACT OF PARTITIONING

In our experiments, we are using all datasets as they are, i.e. without any argumentation or pre-analysis. All points in the dataset are read in as they are laid out in the files (*constant*). Our algorithm splits data into multiple different partitions throughout the construction process. When using multi-GPU variants, further partitions are introduced by sharding. Table 4 demonstrates the effect of constructing the search-graph on a randomly *shuffled* dataset to mix up the points grouped together in each otherwise deterministic partition. The experiment is conducted on on the 1 million subset of SIFT1B with 4 shards. The mean μ and standard

deviation σ over the consensus@10 are computed across 100 iterations. While σ is increased in the *shuffled* case, μ is on par with the constant read in, which is expected due to our randomized selection process for upper-level points.

TABLE 4: Analysis of partitioning impact: Mean μ and standard deviation σ are computed on the 1M subset of SIFT1B with 4 shards over 100 iterations.

τ	c@10 μ	c@10 σ	
0.6	0.9956	1.8e-5	constant
	0.9959	2.0e-4	shuffled
0.7	0.9989	1.4e-5	constant
	0.9988	9.5e-5	shuffled

APPENDIX G
IMPACT OF THE SYMMETRIC LINKING CONSTRAINT AND REFINEMENT

To analyze the impact of our proposed symmetric linking constraint introduced in Section 5.2.1 and search-graph refinement introduced in Section 5.3, we inspect the query performance on SIFT1M for several configurations.

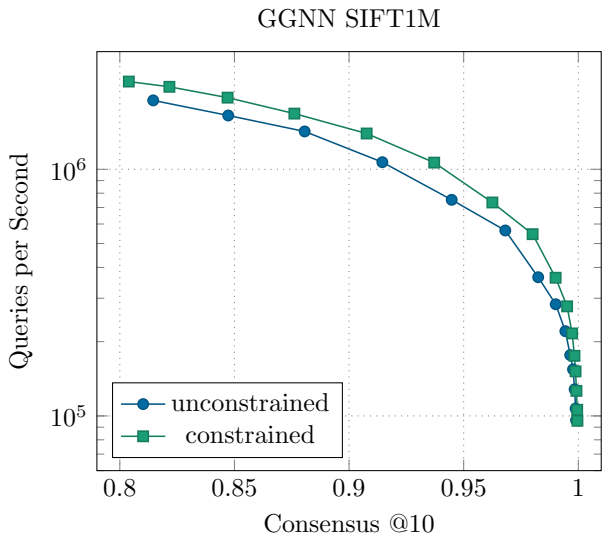


Fig. 4: Comparing query performance using symmetric linking without and with the additional constraint introduced in Section 5.2.1.

We inspect query performance without and with the additional symmetric linking constraint introduced in section 5.2.1 in Figure 4. Search-graph construction took 21.2 seconds without the constraint and 22.1 seconds with the constraint. Without the additional constraint, symmetric linking took about 250 ms more, but merging took about 1.1 seconds less construction time.

These results confirm that constraining the "symmetric" query for reverse paths from known nearest neighbors results in the addition of better links for graph diversification. With the additional constraint in place, following queries explore a richer set of candidate points, which takes slightly more time to explore but also results in increased accuracy. This is part of what allows our method to achieve (close to) perfect recall rates even on billion-scale datasets. At the same time, more queries can be processed per second at the same accuracy.

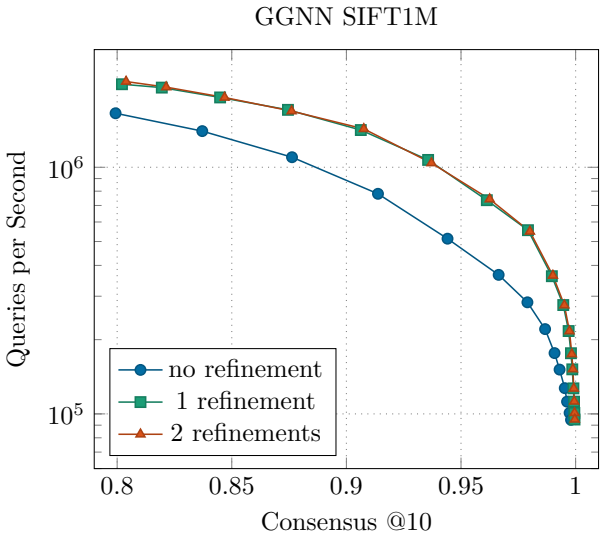


Fig. 5: Query performance at different levels of refinement.

When inspecting the query performance at different levels of refinement as shown in Figure 5, we see that query performance is increased significantly after the first refinement iteration. Refining the search-graph a second time can further boost query performance by a small amount in terms of both speed and accuracy. Further refinements result in no noticeable changes in query performance. Therefore, we generally perform two refinement iterations during search-graph construction to achieve optimal query performance while keeping construction time to a minimum.

REFERENCES

[1] Y. A. Malkov and D. A. Yashunin, "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs," *IEEE transactions on pattern analysis and machine intelligence*, 2018. 1

[2] W. Li, Y. Zhang, Y. Sun, W. Wang, M. Li, W. Zhang, and X. Lin, "Approximate nearest neighbor search on high dimensional data-experiments, analyses, and improvement," *IEEE Transactions on Knowledge and Data Engineering*, 2019. 3