Interactive, Example-driven Synthesis and Manipulation of Visual Media

Dissertation zur Erlangung des Grades eines
Doktors der Ingenieurwissenschaften der
Fakultät für Mathematik und Informatik der
Universität des Saarlandes

Vorgelegt von

Bernhard Reinert
Weidenalle 61, Haus 6
20357 Hamburg
Deutschland

im Juli 2016 in Saarbrücken

Dekan – Dean
Prof. Dr. Frank-Olaf Schreyer

Kolloquium – Examination

Datum – Date
02. Dezember 2016

Vorsitzender – Chair
Prof. Dr. Matthias Hein

Bericherstatter – Correspondents
Dr. Tobias Ritschel
Dr. Johannes Kopf
Prof. Dr. Hans-Peter Seidel

Akademischer Mitarbeiter – Staff member
Dr. Michael Zollhöfer

Abstract

This thesis proposes several novel techniques for interactive, example-driven synthesis and manipulation of visual media. The numerous display devices in our everyday lives make visual media, such as images, videos, or three-dimensional models, easily accessible to a large group of people. Consequently, there is a rising demand for efficient generation of synthetic visual content and its manipulation, especially by casual users operating on low-end, mobile devices. Off-the-shelf software supporting such tasks typically requires extensive training and in-depth understanding of the underlying concepts of content acquisition, on the one hand, and runs only on powerful desktop machines, on the other hand, limiting the possibility of artistic media generation to a small group of trained experts with appropriate hardware. Our proposed techniques aim to alleviate these requirements by allowing casual users to synthesize complex, high-quality content in real-time as well as to manipulate it by means of simple, example-driven interactions.

First, this thesis discusses a manipulation technique that visualizes an additional level of information, such as importance, on images and three-dimensional surface models by local, non-uniform, and self-intersection-free size manipulations. Second, we propose a technique to automatically arrange and sort collections of images based on the images' shape and a sparse set of exemplar images that builds on a novel distribution algorithm. Along this line, an extension for higher dimensions such as three-dimensional models is presented and the implications of distributions for lower-dimensional projections are discussed. Further, the spectral properties of the distributions are analyzed and the results are applied for efficient, high-quality image synthesis. Finally, we suggest an algorithm to extract deformable, three-dimensional content from a two-dimensional video leveraging a simple limb representation that the user sketches onto a sparse set of key frames.

All methods build on the availability of massively parallel execution hardware, such as *graphics processing units* (GPUs), nowadays built also into cheap mobile devices. By mathematical abstraction, parallelization, and task distribution our algorithms achieve a high efficiency that allows running our methods in real-time on low-end devices.

Kurzzusammenfassung

Die vorliegende Dissertation stellt mehrere neuartige Techniken zur interaktiven, beispielbasierten Synthese und Manipulation visueller Medien vor. Die zunehmende Verbreitung von Wiedergabegeräten macht visuelle Medien – wie Bilder, Videos oder dreidimensionale Oberflächen – einer Vielzahl von (Privat-)Nutzern zugänglich. Folglich gibt es auch im Alltagsbereich einen wachsenden Bedarf an effizienter Synthese und Bearbeitung visueller Medien unter Zuhilfenahme weitverbreiteter, mobiler Endgeräte. Handelsübliche Software, die in der Lage ist, diese Aufgaben umzusetzen, setzt in der Regel langwierige Schulungen sowie ein tiefgreifendes Verständnis der zugrundeliegenden Konzepte zur Inhaltserzeugung voraus. Auf der anderen Seite lässt sich diese Software nur auf leistungsstarken Arbeitsplatzcomputern ausführen, wodurch die Möglichkeit künstlerischer Medienerzeugung auf eine kleine Gruppe gut ausgebildeter Experten mit passender Hardware beschränkt wird. Die im Rahmen dieser Arbeit entwickelten Techniken haben das Ziel, die Hürden für private Endnutzer herabzusetzen, indem sie Gelegenheitsnutzern ermöglichen, komplexe Inhalte von hoher Qualität mit Hilfe von einfachen, beispielbasierten Interaktionen in Echtzeit zu synthetisieren und zu manipulieren.

Zunächst wird eine Manipulationstechnik präsentiert, bei der es darum geht, ein weiteres Level an Informationen, wie z. B. Gewicht, auf Bildern und dreidimensionalen Modellen durch lokale, ungleichförmige und selbstüberschneidungsfreie Größenmanipulationen zu visualisieren. Anschließend stellen wir eine Technik vor, die auf einem neuartigen Verteilungsalgorithmus beruht und Bildersammlungen automatisch anhand der Form der Bilder sowie einer kleinen Menge von Beispielbildern arrangiert. In diesem Kontext stellen wir eine Erweiterung für höhere Dimensionen, wie z. B. dreidimensionale Oberflächen, vor und diskutieren die Implikationen der Verteilungen auf geringer-dimensionale Projektionen. Weiterhin werden die spektralen Eigenschaften der Verteilungen analysiert und zur effizienten, hochqualitativen Bildsynthese eingesetzt. Schließlich schlagen wir einen Algorithmus zur Extraktion deformierbarer, dreidimensionaler Inhalte aus zweidimensionalen Videos vor, der auf einer simplen Repräsentation von Körperteilen beruht, die der Nutzer in einigen wenigen Schlüsselbildern skizziert.

Alle Methoden basieren auf der Verfügbarkeit von hochgradig paralleler Hardware, wie z. B. eine *Grafikkarte* (GPU), die heutzutage auch in günstigen, mobilen Geräten verbaut wird. Durch mathematische Abstraktion, Parallelisierung und Aufgabenverteilung erreichen unsere Algorithmen eine hohe Effizienz, die es erlaubt die Techniken in Echtzeit auf diesen mobilen Geräten auszuführen.

Summary

This thesis proposes several novel techniques for interactive, example-driven synthesis and manipulation of visual media. The numerous display devices in our everyday lives make visual media, such as images, videos, or three-dimensional models, easily accessible to a large group of people. Consequently, there is a rising demand for efficient generation of synthetic visual content and its manipulation, especially by casual users operating on low-end, mobile devices. Off-the-shelf software supporting such tasks typically requires extensive training and in-depth understanding of the underlying concepts of content acquisition, on the one hand, and runs only on powerful desktop machines, on the other hand, limiting the possibility of artistic media generation to a small group of trained experts with appropriate hardware. Our proposed techniques aim to alleviate these requirements by allowing casual users to synthesize complex, high-quality content in real-time as well as to manipulate it by means of simple, example-driven interactions.

First, this thesis discusses a manipulation technique that visualizes an additional level of information, such as importance, on images and three-dimensional surface models by local, non-uniform, and self-intersection-free size manipulations. Second, we propose a technique to automatically arrange and sort collections of images based on the images' shape and a sparse set of exemplar images that builds on a novel distribution algorithm. Along this line, an extension for higher dimensions such as three-dimensional models is presented and the implications of distributions for lower-dimensional projections are discussed. Further, the spectral properties of the distributions are analyzed and the results are applied for efficient, high-quality image synthesis. Finally, we suggest an algorithm to extract deformable, three-dimensional content from a two-dimensional video leveraging a simple limb representation that the user sketches onto a sparse set of key frames.

All methods build on the availability of massively parallel execution hardware, such as *graphics processing units* (GPUs), nowadays built also into cheap mobile devices. By mathematical abstraction, parallelization, and task distribution our algorithms achieve a high efficiency that allows running our methods in real-time on low-end devices.

This thesis starts with an introduction in Chapter 1 that commences the topics, gives an overview of the specific contributions made in the different techniques, and provides an outline of the following chapters. In Chapter 2 relevant related work and further technical background for the subsequent chapters is discussed. Following, Chapter 3 to Chapter 6 present the four main approaches of this thesis. Chapter 7 concludes this thesis and discusses potential directions for future work, both in terms of individual works and combinations thereof. The specific works are listed below.

Homunculus Warping False color coding provides a simple means to depict an additional level of information on a three-dimensional surface or two-dimensional image. However, human color perception differs surprisingly much amongst different viewers, limiting the effectiveness of false color coding. Reversely, human perception of relative extent, area, or volume is remarkably invariant amongst individuals and most naturally relates to importance, nearness, and weight. Conveying importance of specific parts by depicting it at a different size is a classic artistic principle, in particular when importance varies across a domain. One striking example is the *neuronal homunculus*; a human figure where the size of each body part is proportional to the neural density on that part. We propose an approach which enables casual users to create such models starting from an undeformed input model by simple specification of a scalar importance per model part. Our approach changes local size of a two-dimensional image or a three-dimensional surface that, at the same time, minimizes distortion, prevails smoothness, and, most importantly, avoids fold-overs, i. e., collisions. We employ a parallel, two-stage optimization algorithm that scales the shape non-uniformly according to an interactively-defined, sparse importance map and then solves for a similar, self-intersection-free configuration. Our results include a three-dimensional, rendered version of the classic neuronal homunculus but also a range of images and surfaces with different importance maps.

Interactive By-example Design of Artistic Packing Layouts Combining several images into a large collage can be a tedious task requiring a lot of manual effort especially for arbitrarily shaped objects. We propose an approach to "pack" a set of two-dimensional graphical primitives into a spatial layout that follows artistic goals. We formalize this process as a projection from a high-dimensional feature space into a two-dimensional layout space. Our system does not expose the control of this projection to the user in form of sliders or similar interfaces. Instead, we infer the desired layout of all primitives from interactive placement of a small subset of example primitives. To produce a pleasant distribution of primitives with spatial extend, we propose a novel generalization of *Centroidal Voronoi Tessellation* which equalizes the distances between boundaries of nearby primitives. Compared to previous primitive distribution approaches our parallel implementation achieves both, better quality and asymptotically higher speed. A user study evaluates the system's usability and feasibility.

Projective Blue-Noise Sampling Synthesizing realistic digital images requires an approximation of the physically correct, full light transport that is present in the scene, often solved by sampling at specific locations. To get good and fast results the sample pattern should cover the domain uniformly without too much uniformity that can lead to artifacts. We propose projective blue-noise patterns that retain their blue-noise characteristics when undergoing one or multiple projections onto lower-dimensional subspaces. These patterns are produced by extending existing methods, such as dart throwing and Lloyd relaxation, and have a range of applications. For numerical integration, our patterns often outperform state-of-the-art stochastic and low-discrepancy patterns, which have been specifically designed only for this purpose. Our patterns generalize the approach of packing primitives in two-dimensions to arbitrary dimensionality of the primitives, containers, as well as their projection, i. e., they allow to distribute primitives uniformly in three-dimensional space

such that their full-dimensional distributions as well as their two-dimensional projections retain a blue-noise distribution. Finally, for image reconstruction, our method outperforms traditional blue-noise sampling when the variation in the signal is concentrated along one dimension.

Animated 3D Creatures from Single-View Video by Skeletal Sketching Extraction of deformable three-dimensional geometry is not accessible to casual users, as it either requires dedicated hardware or vast manual effort. Inspired by the recent success of semi-automatic, three-dimensional reconstruction from a single image, we introduce a sketch-based extraction technique that allows a fast reconstruction of a dynamic, articulated shape from a single video. We model the shape as a union of generalized cylinders deformed by an animation of their axes, representing the "limbs" of the articulated creature. The axes are acquired from strokes sketched by the user on top of a few key frames. Our method bypasses the meticulous effort required to establish dense correspondences when applying common structure from motion techniques for shape reconstruction. Instead, we produce a plausible shape from the fusion of silhouettes over multiple frames. Reconstruction is performed at interactive rates, allowing interaction and refinement until the desired quality is achieved.

Zusammenfassung

Die vorliegende Dissertation stellt mehrere neuartige Techniken zur interaktiven, beispielbasierten Synthese und Manipulation visueller Medien vor. Die zunehmende Verbreitung von Wiedergabegeräten macht visuelle Medien – wie Bilder, Videos oder dreidimensionale Oberflächen – einer Vielzahl von (Privat-)Nutzern zugänglich. Folglich gibt es auch im Alltagsbereich einen wachsenden Bedarf an effizienter Synthese und Bearbeitung visueller Medien unter Zuhilfenahme weitverbreiteter, mobiler Endgeräte. Handelsübliche Software, die in der Lage ist, diese Aufgaben umzusetzen, setzt in der Regel langwierige Schulungen sowie ein tiefgreifendes Verständnis der zugrundeliegenden Konzepte zur Inhaltserzeugung voraus. Auf der anderen Seite lässt sich diese Software nur auf leistungsstarken Arbeitsplatzcomputern ausführen, wodurch die Möglichkeit künstlerischer Medienerzeugung auf eine kleine Gruppe gut ausgebildeter Experten mit passender Hardware beschränkt wird. Die im Rahmen dieser Arbeit entwickelten Techniken haben das Ziel, die Hürden für private Endnutzer herabzusetzen, indem sie Gelegenheitsnutzern ermöglichen, komplexe Inhalte von hoher Qualität mit Hilfe von einfachen, beispielbasierten Interaktionen in Echtzeit zu synthetisieren und zu manipulieren.

Zunächst wird eine Manipulationstechnik präsentiert, bei der es darum geht, ein weiteres Level an Informationen, wie z. B. Gewicht, auf Bildern und dreidimensionalen Modellen durch lokale, ungleichförmige und selbstüberschneidungsfreie Größenmanipulationen zu visualisieren. Anschließend stellen wir eine Technik vor, die auf einem neuartigen Verteilungsalgorithmus beruht und Bildersammlungen automatisch anhand der Form der Bilder sowie einer kleinen Menge von Beispielbildern arrangiert. In diesem Kontext stellen wir eine Erweiterung für höhere Dimensionen, wie z. B. dreidimensionale Oberflächen, vor und diskutieren die Implikationen der Verteilungen auf geringer-dimensionale Projektionen. Weiterhin werden die spektralen Eigenschaften der Verteilungen analysiert und zur effizienten, hochqualitativen Bildsynthese eingesetzt. Schließlich schlagen wir einen Algorithmus zur Extraktion deformierbarer, dreidimensionaler Inhalte aus zweidimensionalen Videos vor, der auf einer simplen Repräsentation von Körperteilen beruht, die der Nutzer in einigen wenigen Schlüsselbildern skizziert.

Alle Methoden basieren auf der Verfügbarkeit von hochgradig paralleler Hardware, wie z. B. eine *Grafikkarte* (GPU), die heutzutage auch in günstigen, mobilen Geräten verbaut wird. Durch mathematische Abstraktion, Parallelisierung und Aufgabenverteilung erreichen unsere Algorithmen eine hohe Effizienz, die es erlaubt die Techniken in Echtzeit auf diesen mobilen Geräten auszuführen.

Diese Dissertation beginnt mit einer Einführung in Kapitel 1, die die Thematik vorstellt, einen Überblick über die spezifischen Beiträge der unterschiedlichen Ansätze liefert und eine Übersicht über die folgende Kapitel darstellt. In Kapitel 2 werden relevante Arbeiten diskutiert und tiefergehende technische Hintergründe für die folgenden Kapitel vorgestellt. Anschließend stellen Kapitel 3 bis Kapitel 6 die vier Hauptansätze dieser Arbeit vor. Kapitel 7 beschließt diese Dissertation und erörtert mögliche Ansätze für zukünftige Arbeiten, sowohl individuell als auch in Kombination. Im Folgenden werden die spezifischen Arbeiten genauer vorgestellt.

Homunculus Warping Eine Darstellung in Falschfarben stellt eine einfache Möglichkeit dar, ein weiteres Level an Information auf einer dreidimensionalen Oberfläche oder einem zweidimensionalen Bild zu illustrieren. Die menschliche Farbwahrnehmung variiert allerdings erstaunlich stark zwischen verschiedenen Beobachtern, was die Effektivität der Falschfarbdarstellung eingeschränkt. Andererseits ist die menschliche Wahrnehmung von relativer Größe, Fläche oder Volumen auffallend invariant zwischen verschiedenen Individuen und bezieht diese Einheiten naturgemäß auf Wichtigkeit, Nähe und Gewicht. Weiterhin ist die Darstellung spezifischer Bestandteile eines Objektes mit besonderer Bedeutung in einer realitätsfremden Größe ein klassisches, künstlerisches Prinzip, das besonders bei variierender Bedeutung der Bestandteile zur Geltung kommt. Ein eindrucksvolles Beispiel ist der *neuronale Homunkulus*, ein Modell des menschlichen Körpers, dessen Körperteile proportional zur spezifischen neuronalen Dichte skaliert wurden. Unser Ansatz erlaubt es Gelegenheitsnutzern, besagte Modellvariationen aus undeformierten Modellen durch simple Festlegung eines skalaren Gewichtes zu erstellen. Wir erreichen dies durch eine Modifikation der lokale Größe eines zweidimensionalen Bildes oder einer dreidimensionalen Oberfläche, die simultan Verzerrungen minimiert, Gleichmäßigkeit erzielt und vor allem Selbstüberschneidung und -kollisionen vermeidet. Hierzu nutzen wir eine parallele, zweistufige Optimierung, die das Modell ungleichförmig skaliert. Diese Optimierung basiert auf einer interaktiv modifizierbaren, dünn besetzten Gewichtskarte, die zunächst zur Berechnung einer nicht überschneidungsfreien Vorschau dient. Diese wird dann als Zielkonfiguration zur Berechnung einer Konfiguration ohne Selbstüberschneidungen herangezogen. Unsere Ergebnisse beinhalten eine dreidimensionale, gerenderte Version des klassischen, neuronalen Homunkulus aber auch eine Auswahl an anderen Bildern und Oberflächen mit diversen Gewichtskarten.

Interactive By-example Design of Artistic Packing Layouts Die Kombination mehrerer Bilder (sog. Primitive) zu einer ganzheitlichen Kollage stellt einen mühsame Aufgabe dar, die, besonders im Fall beliebig geformter Bilder, mit enormerem Aufwand verbunden ist. Wir stellen einen Ansatz zum Anordnen zweidimensionale Bilder in ein räumliches Layout vor, der künstlerischen Aspekten folgt. Wir formalisieren diesen Prozess als eine Projektion von einem hochdimensionalen Merkmalsraum in einen zweidimensionalen Layoutraum. Hierbei kontrolliert der Nutzer diese Projektion nicht mithilfe von Schiebereglern oder ähnlichen Schnittstellen, sondern durch das interaktive Platzieren einer kleinen Teilmenge von Beispielprimitiven, aus der das gewünschte Layout der übrigen Primitive abgeleitet wird. Um eine ansprechende Verteilung der Primitive mit räumlicher Größe zu erzielen, schlagen wir eine neuartige Verallgemeinerung der sog. *Centroidal Voronoi Tessellation* vor,

die die Abstände zwischen den Rändern der Primitive in alle Richtungen ausgleicht. Im Gegensatz zu vorherigen Verteilungstechniken erreicht unsere parallele Implementierung sowohl eine höhere Qualität als auch eine asymptotisch höhere Geschwindigkeit. Eine Nutzerstudie evaluiert die Nutzbarkeit und Einsetzbarkeit unseres Systems.

Projective Blue-Noise Sampling Die digitale Synthese realistischer Bilder erfordert eine Approximation des physikalisch korrekten, vollständigen Lichttransportes der Szene, die oft durch Abtasten an spezifischen Orten angenähert wird. Um effizient zufriedenstellende Ergebnisse zu erzielen, sollte die Verteilung der Abtastorte die Domäne gleichmäßig abdecken ohne Regelmäßigkeit aufweisen, da diese zu Artefakten führen kann. Wir schlagen sog. *Projective Blue-Noise* Verteilungen vor, die ihre Blue-Noise Eigenschaften auch bei Projektionen in einen oder mehrere, geringer-dimensionale Unterräume beibehalten. Diese Verteilungen werden durch Erweiterungen existierender Methoden, wie z. B. Dart Throwing und Lloyd Relaxation, erreicht und haben eine Vielzahl von Anwendungen. Bei der numerischen Integration übertreffen unsere Muster häufig stochastische Muster und solche mit niedriger Diskrepanz, die speziell zu diesem Zweck entworfen wurden und dem Stand der Technik entsprechen. Unsere Verteilungen verallgemeinern den Ansatz, Primitive im zweidimensionalen Raum zu arrangieren, für eine beliebige Dimensionalät der Primitive, der Container als auch ihrer Projektionen. Mit anderen Worten erlauben sie z. B. Primitive gleichmäßig im dreidimensionalen Raum zu verteilen, so dass sowohl ihre volldimensionale Verteilung als auch ihre zweidimensionalen Projektionen eine Blue-Noise Verteilung darstellen. Schließlich übertrifft unsere Methode bei der Bildrekonstruktion traditionelle Blue-Noise Muster, vor allem falls die Variation des Signals sich hauptsächlich auf eine Dimension konzentriert.

Animated 3D Creatures from Single-View Video by Skeletal Sketching Die Extraktion deformierender, dreidimensionaler Geometrie ist nicht zugänglich für Gelegenheitsnutzer, da dieser Vorgang entweder dedizierte Hardware oder enormen manuellen Aufwand erfordert. Inspiriert durch den kürzlichen Erfolg von halbautomatischen, dreidimensionalen Rekonstruktionen einzelner Bilder anhand von Skizzen, stellen wir eine skizzenbasierte Extraktionsmethode vor, die eine schnelle Rekonstruktion von dynamisch artikulierten Formen aus einem einzelnen Video ermöglicht. Wir modellieren den Umriss des zu rekonstruierenden Objektes als Vereinigung von generalisierten Zylindern, die von einer Animation ihrer Achsen deformiert werden und die Körperteile einer artikulierten Kreatur darstellen. Diese Achsen werden aus Strichskizzen, die der Nutzer auf ein paar wenige Schlüsselbilder der Videosequenz malt, akquiriert. Unsere Methode umgeht die Notwendigkeit dichter Korrespondenzen zwischen den Bildern der Videosequenz, die bei der Rekonstruktion der Form mit Hilfe von Methoden der *Struktur aus Bewegung* benötigt werden. Stattdessen erzeugt unser Ansatz eine plausible Form aus der Fusion von Silhouetten aus mehreren Bildern. Unsere Implementierung erreicht eine interaktive Geschwindigkeit bei der Rekonstruktion, wodurch Interaktion mit den Resultaten und Anpassung des Ergebnisses bis zur gewünschte Qualität ermöglicht werden.

Contents

List of Figures

Chapter 1

Introduction

This thesis proposes several novel techniques for interactive, example-driven synthesis and manipulation of visual media. In this first chapter, we motivate our research (Section 1.1), present our main contributions (Section 1.2) and outline the whole thesis (Section 1.3).

1.1 Background

Nowadays, display devices for digital visual media are easily and universally accessible in our everyday lives. Such devices range from classical computer monitors, used in our homes or at work, over smartphone screens, used in casual situations, to modern head-mounted displays (HMDs) often used for entertainment such as virtual reality (VR). To increase mobility, recently, devices tend to get smaller, effectively reducing their performance and power consumption. Content for these devices can easily be compiled also by casual users using sophisticated and accessible acquisition devices such as cameras or 3D scanners even available in many modern, low-end mobile devices. However, virtual synthesis of adequate content or manipulation of existing content meeting the users' intention and imagination is an intricate task that typically requires high-performance hardware. Additionally, the tasks warrant a substantial level of training from the users as they generally need to understand fundamental concepts of content acquisition to effectively control the parameters of synthetic content creation. Output content can range from two-dimensional images, over image collections to animated, three-dimensional surface models, each implying its distinct set of parameters that need to be controlled and adapted. Consequently, content creation often requires specific training of the users for each of these scenarios.

A wide variety of software is readily available that aims at making media creation as easy as possible and allows for efficient content creation, e. g., Photoshop [Adobe, 2016], Blender [Blender Online Community, 2016], etc. However, realizing sophisticated effects with this software typically requires a considerable amount of familiarization. Commonly, the applications decouple content creation from direct interaction with the content, i. e., users have to pick potentially non-descriptive parameters in an external dialog to realize

their intended effects. Especially for visual media however, the desired appearance can often be achieved through direct manipulation by the user employing appropriate editing tools. Manipulation of the entire model to fit the desired appearance, though, yields a time-consuming and tedious task. These manipulations commonly include large amounts of repetitive work whereas the parameters governing them can be described already by a small subset of the entire manipulations. Providing this subset of example manipulations, the underlying parameters of the full manipulation could be inferred by the software and applied to the full model. Hence, example-driven approaches generalize the appearances of a small set of example to the entire model, potentially leveraging model knowledge to restrict the manipulations to only plausible ones. On the one hand, a major advantage of these approaches is that they unify the required interactions among different domains. They enable synthesis and manipulation tools for different kinds of visual media by simple interactions as they operate on examples, often more intuitive and less involved than explicit parameter adaption. On the other hand, example-based approaches require careful design as they tend to overfit and provide erroneous guesses. Example-based approaches have received a lot of interest and are an active area of research (cf. [Wang et al., 2008; Wei et al., 2009; Garg, Jacobson and Grinspun, 2016]).

Synthesis and manipulation of visual media constitutes a forward problem, i. e., a set of parameters, e. g., three-dimensional geometry and a set of light sources, is provided that governs the final outcome, e. g., the rendered two-dimensional image. Estimation of these parameters given the outcome can be regarded as the inverse or backward problem, e. g., geometry estimation from rendered images. Example-based approaches are one avenue of such inverse problems, as illustrated in this thesis.

Visual media synthesis and manipulation is an intricate task amounting to a high computational complexity. Example-based approaches add an extra layer of complexity through parameter estimation, resulting in a involved and computationally expensive system. Through careful problem formulation many of the problems in media generation allow for massive parallelization that can primarily be exploited by the universally and widespread available *graphics processing units (GPUs)*. Nowadays, such processors are built even into low-end devices enabling utilization also by casual users.

This thesis aims at making content creation more accessible to casual users by introducing efficient and accessible techniques for media creation based on example-based approaches, exemplified by four techniques.

Synthesis of visual media is the task to digitally create artificial visual media such as images, videos, or thre-dimensional models that comply with the users' requirements.

Generating two-dimensional images from three-dimensional scenes is called *rendering* and builds the classical core of computer graphics. The first system to allow for real-time, three-dimensional graphics and text was the Whirlwind Computer in the 1950s [Everett, 1951]. Ever since the quality and speed of rendering has been improved tremendously, in particular by seminal works of Phong [1975], Blinn et al. [1976], Cook et al. [Cook, Carpenter and Catmull, 1987], and many others resulting in state-of-the-art, real-time graphics almost indistinguishable from real photographs. Content creation for such renderings however,

like three-dimensional models and materials, still poses an intricate task that is typically restricted to trained artists requiring vast amounts of manual work. Simplifying this work and enabling content creation for casual users hence has large potential but demands for easier and more intuitive content creation tools. Such tools are an active area of research and many instances have been developed over the years. On the one hand, different tools for editing various kinds of media such as materials [Menzel and Guthe, 2009], colors [Nguyen, Ritschel and Seidel, 2015], as well as three-dimensional models [Sorkine and Alexa, 2007] have been developed but still possess limitations. Editing of three-dimensional models, for example, still most commonly does not handle collisions introducing a demand for more intuitive and lifelike editing tools resulting in feasible manipulations. On the other hand, analyzing the effects of parameter changes on the final rendering requires rapid rendering previews. Handling sophisticated effects such as area light sources however is costly. Hence, there is a demand for better and faster convergence rates in rendering.

The inverse rendering problem constitutes another typical task of visual media synthesis. Here, the three-dimensional information previously projected into the two-dimensional images are to be recovered. Ideally, these methods separate camera and object motion as well as lighting and texturing of the models. Fully automated reconstruction has been a classical task and was introduced by Prazdny [1980] and further investigated by the seminal works of Spetakis et al. [1987], Hartley et al. [2004], and many others. Using images taken from different positions and directions, sophisticated solutions for static geometry reconstruction have been proposed. If enough camera images are available these systems can recreate the full three-dimensional information for typical scenes. The mentioned techniques commonly assume fully rigid geometry with a rigidly transforming camera as the sole dynamic object. In the more realistic case of limited views and deforming geometry, these geometrical solutions however become under-constrained and hard to solve. Particularly, the special case of deforming geometry in a single view has recently attracted a lot of interest, aiming for fully automated methods. If three-dimensional scans are available, sophisticated registration techniques can be employed [Li, Sumner and Pauly, 2008]. In the case of two-dimensional video, fully automated solutions were presented e. g., by Russell et al. [2014], but are limited to sparse reconstructions of short sequences, resulting in a demand to overcome the aforementioned limitations. A recent trend in reconstruction problems is to use minimal user input, e. g., by sketching generalized cylinders [Chen et al., 2013b]. Combining these ideas with deforming geometry allows for dense, three-dimensional reconstructions of two-dimensional videos, i. e., utilizing a set of examples. Here, the solution space should be restricted to only plausible solutions by including knowledge about the model to reconstruct.

Manipulation of visual media Besides synthesis, manipulation of existing visual media to meet the users' intention is of great interest for artists and casual users. Digital media can be replicated and manipulated, enabling non-destructive editing and hence making digital modeling an ideal tool for both, casual as well as experienced users.

One classical manipulation example is layouting of text and images, where the positions of characters, paragraphs, and other elements are manipulated until a desired document layout is acquired. To this end several constraints, such as the document measurements,

line heights, and margins have to be considered. The advent of *letterpress printing* in the 15th century lead to a significant increase in interest and speed with which such layouting problems could be solved. Reordering and -formatting of text and images became a task of simply replacing and inserting certain elements, abolishing the need to recreate the entire document. The digital revolution in the 20th century increased the dissemination even further as it simplified the process of layout development to a level that enabled rapid layout generation even for casual users utilizing easy-to-use software, such as Microsoft Word [Microsoft, 2016]. Due to the high number of constraints of such text layouts the search space for the optimal layout can be pruned extensively, allowing for efficient inference of solutions even on low-performance machines. Similar layouting examples include mosaic generation, a classical art form, or, more generally, the generation of packing layouts optionally following additional constraints. Conversely however, efficient solutions for such problems are much more involved as their computational complexity is disparately higher due to a much larger search space that cannot be pruned as efficiently. One instance of such packing problems particularly covered in this thesis is the packing of images with arbitrary boundaries into arbitrary containers. Besides an even distribution of the elements in the container, ideally having the same distance in all directions, additional constraints govern the macroscopic distribution of the elements, e. g., sorting the elements based on visual features such as brightness. Similar to text layouts, here, the position of the elements can convey certain additional information such as brightness gradients or the like. Controlling these additional information can be tedious and benefits from example-based approaches where the parameters are learned from examples. While the distribution itself can be seen as a forward problem, estimating the parameters for the distribution from examples constitutes an inverse problem. Moving from images to three-dimensional models or even higher dimensions increases the complexity exponentially and demands for even more efficient solutions.

Another instance of media manipulation addresses deformations of two- and especially three-dimensional surfaces. *Sculpture by manipulation* is a classical real-world modeling technique to shape three-dimensional surfaces involving flexible materials such as wax, plaster, or clay. It allows deforming the sculpture and adding or removing certain parts until a desired outcome is achieved, constituting intuitive modeling mechanisms. Moving to the digital world, naturally, in modern three-dimensional modeling software such as Blender [Blender Online Community, 2016] these modeling metaphors are also widely utilized. In contrast to real-world modeling, physical properties, such as collisions of the surfaces, are often ignored in digital modeling as they substantially increase the complexity of the modeling process. Support for these properties has the potential to significantly increase the intuitiveness of modeling tools resulting in an improved user experience. Besides three-dimensional surface manipulation, avoiding collisions is beneficial for many manipulation tools in all dimensions as it is closer to a physically correct behavior. Additionally, manipulations often involve repetitive work that can be overcome using example-based parameter estimations. Such manipulations can be used to generalize example manipulations to the full model, but also from a single model to a collection of models.

Interactivity is a crucial requirement for example-driven approaches such as the ones introduced in this thesis. It relates to both, interactive exploration of design possibilities by the user and interactive performance provided by the machine. Ideally, these tasks are split optimally such that both, the users can freely attain their imagination without overhead and the machine supports this task with a fast response, leveraging the strength of both sides. A key concept in interactive user interfaces is the direct interaction with the media at hand, e. g., by providing subsets of examples. On the one hand, an interactive application allows for easy, fast, and unimpeded exploration as well as navigation of design possibilities by the user. This exploration of design spaces is an inherently interactive task, as it requires a lot of trial-and-error to navigate the possibilities, i. e., users typically conduct multiple iterations until they arrive at their intended outcome. Hence, on the other hand, fast response of the system is essential and interactive frame rates are desirable. Especially in the case of example-driven approaches that might suggest wrong guesses, interactive speed is substantial as it allows rapid error correction by adding more examples. Content synthesis and manipulation with interactive applications enables fast results, exploration of design spaces, and discovery of new effects.

Many of the problems that arise in example-driven approaches and hence also in this thesis can be formulated as optimization problems. For numerous of these problems off-the-shelf algorithms exist that efficiently solve the problem at hand. However, often, these solutions are still too slow for interactive needs and hence yield a high demand for more efficient solutions. Constraints on the optimization commonly can be used to efficiently prune the space of solutions, avoiding unnecessary search for invalid results. Manual reformulation of the problem or a broader analysis of the problems often reveals particular properties that can be exploited to make the optimization more efficient. Such properties include reducing the number of possible solutions, ideally turning the optimization into a convex problem, or changing its dependency structure to allow for parallelization. Due to the universal availability of parallel hardware nowadays, parallelization of algorithms is of particular interest, especially leveraging the massive parallelism of the GPU. Because of inherent scheduling and memory management limitations of these processors, mapping the optimization problems to the GPU to optimally utilize this massive parallelism requires careful algorithm design. Besides parallelization, splitting the computations between the available processors, such as the *central processing unit (CPU)* and the GPU, to exploit their respective strengths is another avenue of runtime optimization explored in this thesis.

Conclusion The observations above suggest three important properties that ideal example-driven algorithms for synthesis and manipulation of visual media should possess:

- Intuitiveness: Simple user interfaces that abstract non-descriptive parameters increase the intuitive operability of the system.
- Plausibility: Plausible generalizations of appearance examples improve the acceptance of the systems' suggestions by the users.
- Speed: Interactive feedback enhances the user experience and reduces fatigue while using the system.

1.2 Contributions

This thesis proposes novel example-based approaches and addresses common limitations with these approaches, exemplified in four different techniques published in [Reinert, Ritschel and Seidel, 2012; Reinert, Ritschel and Seidel, 2013; Reinert et al., 2015; Reinert, Ritschel and Seidel, 2016]. Below the specific contributions of each technique are discussed.

The first approach in Chapter 3 (based on [Reinert, Ritschel and Seidel, 2012]) enables deformations of two- and three-dimensional surfaces by localized, non-uniform size changes. In contrast to previous work in this research area, the results are self-intersection-free. Its specific main contributions are:

- A novel optimization solver to create deformed, self-intersection-free surface models.
- A combination of a fast, parallel implementation for a preview with an offline solver for the final, self-intersection-free solution.

Following, Chapter 4 (based on [Reinert, Ritschel and Seidel, 2013]) presents an approach to interactively pack a set of example images with arbitrary boundaries into a container of arbitrary shape. The margins between each of the image boundaries are equalized and the images itself follow user-prescribed objectives revealing relationships between the images. These user-prescribed objectives are learned from a set of specifically placed example images. In contrast to previous work our system is drift-free and achieves interactive performance for all steps of the pipeline, resulting in the following main contributions:

- An interactive inverse layout approach to infer a user's packing layout intention from a small number of examples.
- A drift-free layout algorithm to evenly distribute primitives with spatial extend in real-time on a GPU.
- A study of packing layout task performance of novice users.

Chapter 5 (based on [Reinert et al., 2015]) introduces projective properties of point patterns as an important feature for the solution to several computer graphics tasks. The chapter provides an in-depth analysis of point patterns in terms of their projective properties and discusses several applications. It generalizes the work on packing layouts of Chapter 4 to arbitrary dimensions and discusses projections onto the screen. The main contributions are:

- Two projective generalizations of algorithms that produce point patterns of arbitrary dimensionality.
- An in-depth comparison of the spectral and projective properties of projective blue-noise patterns to various competing methods.
- A detailed analysis of the influence and effectiveness of the projective properties.

Finally, Chapter 6 (based on [Reinert, Ritschel and Seidel, 2016]) presents an approach to extract animated, three-dimensional geometry from two-dimensional videos. Extraction of dense, deformable three-dimensional geometry from single view video is typically limited to sparse reconstructions of short sequences. Leveraging minimal user input in form of sparse axis sketches in combination with generalized cylinders, our system generates dense and complete reconstructions and presents the following main contributions:

- A parallel tracking algorithm for axis sketches through image sequences.
- A video segmentation consolidation over all frames of a video based on generalized cylinders.
- A three-dimensional generalized cylinders fitting approach leveraging tracked strokes and segmentation masks.

1.3 Outline

This thesis is structured as follows: Chapter 2 discusses additional background and reviews previous work that substantiates our work. Afterwards, Chapter 3 to Chapter 6 present four novel synthesis and manipulation techniques for visual media in detail. More specifically, Chapter 3 presents a self-intersection-free deformation technique for two- and three-dimensional surfaces by localized, non-uniform size changes. Next, Chapter 4 presents a layouting algorithm for images with arbitrary boundaries that allows to interactively and evenly pack and sort a set of images into an arbitrary container image. Here, the layout intention of the user is inferred from a small set of examples. Subsequently, Chapter 5 extends this notion of layouting and packing for sample patterns in arbitrary dimensions. In particular, projective properties that originate e. g., from perspective projections onto a two-dimensional image are analyzed. Further implications for other tasks such as rendering of area light sources and image reconstruction are discussed. Finally, Chapter 6 presents an approach to extract deformable, three-dimensional geometry from uncalibrated two-dimensional videos leveraging minimal user input in form of sketches. This thesis is completed by a conclusion in Chapter 7 that also discusses potential combinations of our approaches and presents promising avenues of future work.

Chapter 2

Previous work

In this chapter, we review some background and related work of the projects and concepts presented in this thesis. First, synthesis of visual media by means of rendering, model reconstruction, as well as animation are examined. Second, we focus on media manipulation by means of surface deformation and example-based approaches. Point patterns, which occur frequently in computer graphics problems, constitute the next topic. Finally, interactivity by means of parallelization and user interfaces is discussed.

2.1 Synthesis of visual media

Synthesis of visual media is the task of creating novel, synthetic content by combining several, distinct components, e. g., three-dimensional models and light-transport to create an image. In particular this thesis is concerned with the synthesis of two-dimensional images from three-dimensional content, a process that constitutes the forward direction of rendering. Conversely, three-dimensional models can be obtained from two-dimensional images or videos by data-driven approaches, e. g., using reconstruction of deformable three-dimensional geometry, generally representing the inverse direction of rendering. Another avenue this section introduces is animation of visual media.

2.1.1 Rendering

Two-dimensional image synthesis, also referred to as *rendering*, is the traditional core of computer graphics and describes the generation of a realistic, two-dimensional image of a three-dimensional scene by modeling the light transport [Goral et al., 1984]. While this thesis is not immediately concerned with improving core rendering, a thorough understanding of the basic principles is inevitable to understand some of the core concepts in sample patterns (cf. Section 2.3) and the following chapters (cf. e. g., Chapter 5). Additionally, this knowledge is helpful for the inverse direction of rendering, i. e., three-dimensional

reconstruction (cf. Section 2.1.2). For realistic image synthesis, the global light transport (*global illumination*) has to be modeled. Every surface sample can potentially interact with all other surface positions possibly multiple times. Formalizing this concept of light transport, the full light interactions in a scene can be expressed by the well-known rendering equation [Kajiya, 1986]. It describes the radiance L_o emitted at location $\mathbf{x} \in \mathbb{R}^3$ of a surface $\mathcal{M} \subseteq \mathbb{R}^3$ in direction $\omega_\mathrm{o} \in \mathbb{S}^2$, with $\mathbb{S}^2$ as the three-dimensional hemisphere, by an integration over all incoming directions. Omitting the wavelength dependency by assuming that all operations are jointly executed on all color channels, it is defined as

$$L_\mathrm{o}(\mathbf{x}, \omega_\mathrm{o}) = L_\mathrm{e}(\mathbf{x}, \omega_\mathrm{o}) + \int_{\mathbb{S}^2} L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i})\, R(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o})\, \langle n(\mathbf{x}), \omega_\mathrm{i} \rangle^+ \mathrm{d}\omega_\mathrm{i}, \tag{2.1}$$

with L_e as the emitted radiance, L_i as the incoming radiance arriving at location $\mathbf{x}$ from direction ω_i, $n(\mathbf{x})$ as the surface normal at $\mathbf{x}$, and $R(\mathbf{x}, \omega_\mathrm{i}, \omega_\mathrm{o}) \in \mathcal{M} \times \mathbb{S}^2 \times \mathbb{S}^2 \to \mathbb{R}^+$as the *bidirectional reflectance distribution function* [Nicodemus, 1965] of the incoming direction ω_i and the outgoing direction ω_o at location $\mathbf{x}$ (Figure 2.1).

The incoming light L_i potentially emanates from other surface locations and is obtained by solving Equation 2.2 for these locations as well, amounting to a large system of non-linear inter-dependent equations. The resulting interactions between all surface locations make an exhaustive evaluation of the integral infeasible. With increasing scene complexity it becomes prohibitively complex to efficiently evaluate the integral of Equation 2.2. To overcome this limitation, several approximation techniques have been proposed [Lafortune and Willems, 1993] that have been improved to allow for efficient rendering [Vorba et al., 2014]. Usually these techniques target offline rendering, but other techniques that enable real-time rendering have been published that typically require some degree of pre-processing as well as approximation and make certain assumptions about the scene [Ritschel et al., 2012; Keller, 1997; Scherzer et al., 2012]. As evaluating the integral of the rendering equation, stated in Equation 2.2, is not feasible for typical scenes due to complex visibility relations, only approximations of the precise result are possible. A classical technique to approximate numerical solutions of an integral is numerical integration, where the result is approximated by sampling the function to integrate with discrete sample points; this approximation is called *Monte Carlo* integration. The integral of Equation 2.2 is replaced by a finite sum over all directional point samples $\mathbf{s} \in S = [0,1)^2$, i. e., it becomes

$$L_\mathrm{o}(\mathbf{x}, \omega_\mathrm{o}) = L_\mathrm{e}(\mathbf{x}, \omega_\mathrm{o}) + \frac{1}{|S|} \sum_{\mathbf{s} \in S} L_\mathrm{i}(\mathbf{x}, \omega_\mathrm{i})\, R(\mathbf{x}, \omega(\mathbf{s}), \omega_\mathrm{o})\, \langle n(\mathbf{x}), \omega(\mathbf{s}) \rangle^+, \tag{2.2}$$

with ω as a spherical unit vector. Evaluation of this sum is often still too expensive for interactive performance and is replaced by some specialized sum that e. g., only samples direct light sources. Typically, the point patterns must possess special properties to perform well in different scenarios, e.g. when sampling an area light source of different size and shape. Ideally, the patterns should be general purpose patterns, like the patterns in Chapter 5 that work well in many scenarios to overcome the need to have specialized patterns for every scenario.

In combination with *importance sampling* [Veach and Guibas, 1997], results with low variance can be obtained in short time. Importance sampling builds on the availability of

additional knowledge about the function that needs to be integrated, e. g., if an environment map has to be sampled. One possibility is to treat this function as a *cumulative distribution function* that can be inverted. With increasing dimensionality of the problem, taking multiple factors, such as light position, wavelength, and others into account, the problem becomes increasingly difficult. The result of this numerical integration heavily depends on the placement of the samples in a sample pattern. A rich set of analysis methods has been developed that allow predicting convergence rates for rendering by discrepancy analysis of such patterns and numerical integration has been investigated extensively [Halton, 1964; Shirley, 1991; Schlömer and Deussen, 2010]. As sample patterns also serve different purposes in computer graphics a comprehensive review for many applications in combination is presented in Section 2.3.

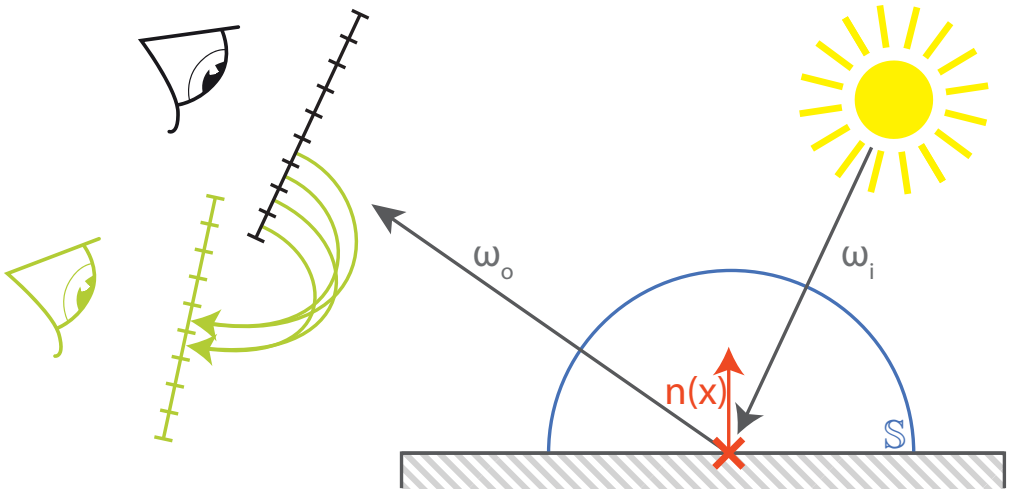

Figure 2.1: *Rendering* integrates the incoming light over all directions ω_i of the hemisphere $\mathbb{S}$ *(blue)* to produce the final result at position $\mathbf{x}$ in direction ω_o. *Image-based rendering*, in contrast, solely reuses the results of the previous image(s) *(black)* to generate the current image *(green)*.

Image-based rendering *Image-based rendering* (IBR) approaches the task of creating an image by solely reusing pixel information from other images. These different images contain information from previous views that in common scenarios presumably are close to the desired current view, but are typically captured from slightly different positions and directions. In this thesis, a variant of IBR is used in Chapter 6 and discussed in more detail as an outlook in Chapter 7. In contrast to traditional rendering described in the previous paragraph, for IBR we do not need to actually solve the rendering equation for each pixel but simply reuse the rendered information of previously received images, making it a computationally affordable method for real-time demands also on mobile hardware. IBR was introduced in the seminal work by Chen et al. [1993] and later improved by Mark et al. [1997] and others. All IBR methods try to invert the flow, i. e., all methods build on the assumption of a known forward flow for each pixel, i. e., for each pixel of the input image we can reconstruct its new position in the output image. The projection of a three-dimensional world position $\mathbf{x}_w \in \mathbb{R}^4$ in homogeneous coordinates into screen-space $\mathbf{x}_s \in \mathbb{R}^4$ given a view

matrix $\mathbf{V}_i \in \mathbb{R}^{4\times4}$ and a projection matrix $\mathbf{P}_i \in \mathbb{R}^{4\times4}$ as

$$\mathbf{x}_w = \mathbf{P}_i\mathbf{V}_i\mathbf{x}_s.$$

For a known projection model, approximate (up to pixel precision) x-, and y-coordinates are implicitly defined by the pixel position for each point. In order to reconstruct the pixels world position only a single depth value needs to be provided. Using an inverse projection, the world position $\mathbf{x}_w$ can be reconstructed from the projected position, i. e.,

$$\mathbf{x}_s = (\mathbf{P}_i\mathbf{V}_i)^{-1}\,\mathbf{x}_w.$$

Now, to calculate the forward flow we simply reconstruct the world position for each pixel and reproject it using the new view marix $\mathbf{V}_o \in \mathbb{R}^{4\times4}$ as well as a potentially new projection matrix $\mathbf{P}_o \in \mathbb{R}^{4\times4}$, e. g., by employing the standard rendering pipeline with depth testing (cf. e. g., [Shreiner et al., 2013]). Since all points are given in homogeneous coordinates, the Cartesian coordinates can be obtained by division by the last element of each vector. Ideally, for each output pixel we would like to know the *backward flow*, i. e., the lookup position in the input image. Obtaining the backward flow is hard due to occlusions and missing regions, leading to potentially multiple or zero solutions per pixel.

2.1.2 Model reconstruction

Reconstructing three-dimensional shape from one or multiple images has been an important area of research in the past decades and remains a challenging task. Especially deforming geometry poses difficult problems that are not easily solvable. Our work in Chapter 6 extends the line of work on user-assisted acquisition of static, three-dimensional geometry from a single view [Chen et al., 2013b] to animated, three-dimensional geometry from multiple video frames.

Three-dimensional geometry is usually acquired using specialized hardware, such as depth sensors [Izadi et al., 2011] or multi-camera setups [Snavely, Seitz and Szeliski, 2006]. When background segmentation is feasible, multiple silhouettes can be combined into a single, three-dimensional object using the visual hull [Matusik et al., 2000]. Sufficiently textured rigid scenes can reliably be acquired using *Structure-from-Motion* (SfM) and enable impressive applications [Snavely, Seitz and Szeliski, 2006] when sufficiently large image collections are available. These algorithms however only reconstruct three-dimensional information for a sparse set of reliably tracked features. Using those features in combination with additional constraints provided by the user, such as symmetry or planarity, high-quality, three-dimensional models can be constructed [Sinha et al., 2008].

If the object class to be reconstructed is known a-priori, specialized template-based solutions for humans from many three-dimensional scans [Allen, Curless and Popović, 2003], faces [Blanz and Vetter, 1999], or animals [Cashman and Fitzgibbon, 2013] have been proposed. Most of these approaches require user interaction in some way, such as defining correspondences by clicking [Allen, Curless and Popović, 2003; Cashman and Fitzgibbon, 2013]. If the video contains a human, for which a template models is available, motion can be captured [Wei, 2010] using automatic or semi-automatic template fitting allowing to

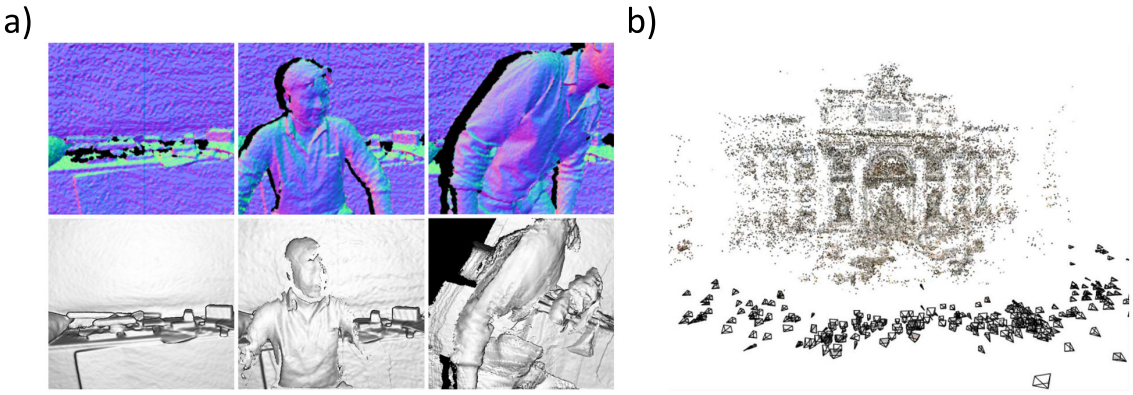

Figure 2.2: *a):* Three-dimensional reconstruction from depth cameras with Kinect Fusion [Izadi et al., 2011], *b):* Three-dimensional reconstruction from multiple photos [Snavely, Seitz and Szeliski, 2006]. *Images courtesy of the publication authors.*

manipulate images [Zhou et al., 2010] or videos [Jain et al., 2010]. The approach described in Chapter 6 goes beyond human shapes, allowing the user to draw and refine arbitrary skeletons unknown a-priori.

Reconstruction of animated, non-rigid three-dimensional models without special hardware poses a challenging, under-constrained problem for which no sophisticated solutions are available. Non-rigid SfM is currently addressed by either assuming that the deformation is a combination of rigid transformations of basis shapes [Bregler, Hertzmann and Biermann, 2000] or basis trajectories [Akhter et al., 2008]. Even if correspondences are given [Garg, Roussos and Agapito, 2013] reconstruction is typically limited to moderately deforming, sphere-like objects and requires long computation time, defying interactive use.

Many works rely on feature tracks that can reliably be tracked and matched throughout long image sequences. For deforming objects these features are clustered into nearly rigid components and their transformations are blended [Russell, Yu and Agapito, 2014] (Figure 2.3, a). Optical flow provides means of calculating the differences between image pairs.

Multi-view three-dimensional reconstruction For multiple views, skeletons, and template models sophisticated systems exist that estimate both, the skeletons and shape, simultaneously [Gall et al., 2009]. In contrast to such approaches, our approach in Chapter 6 does not rely on any a-priori known model or an explicit understanding of the underlying skeletal structure of the creature. Additionally, our algorithm allows for a rich set of deformations, exceeding those of other tracking approaches. While other tracking approaches deform each bone by a single rigid transformation, our limbs commonly aggregate several biological bones allowing for piecewise rigid but also non-rigid motions. This enables tracking of limbs that are otherwise hard to track using a single bone, such as the tail and body of a cheetah or the neck of a giraffe, and abstracts model complexity away. Our system solely relies on the input video in combination with user-defined strokes, enabling three-dimensional reconstruction even for creatures with unknown or no skeleton at all. All

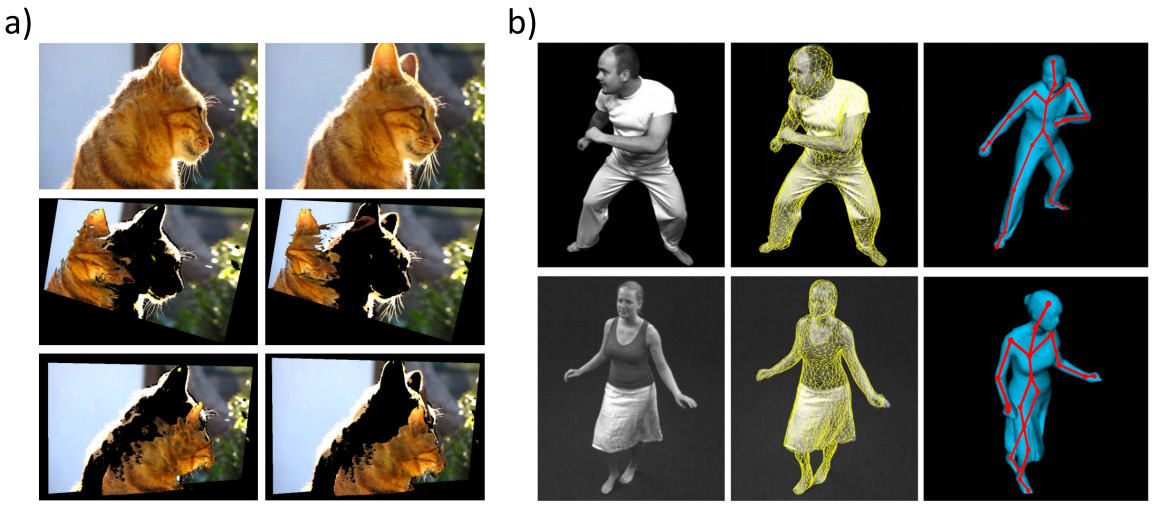

Figure 2.3: *a):* 2.5D reconstruction using Video Pop-Up [Russell, Yu and Agapito, 2014], *b):* Three-dimensional reconstruction using template fitting [Gall et al., 2009]. *Images courtesy of the publication authors.*

video sequences used in Chapter 6 are taken from online video platforms and do not require any calibration steps beforehand, rendering the system useful also for casual users within the assumptions of this thesis.

Single-view three-dimensional reconstruction Creating a three-dimensional model from a single image is an even more challenging task, often addressed using semi-automatic approaches. A classic idea is to assume piecewise planar geometry that is segmented by a user who also specifies the vanishing point [Horry, Anjyo and Arai, 1997]. Zhang et al. [2002] reconstruct a smooth 2.5D patch (equivalent to a depth map) by solving a variational optimization problem that finds a smooth surface that is perpendicular to the viewer at the silhouette and follows several other positional and directional user constraints. Research of human perception has found that the occluding contour or silhouette is a strong cue for the inference of a full shape from its 2D projection [Koenderink, 1984]. Later, the silhouette-based approaches were extended by Prasad et al. [2006] to full 3D patches. Most systems require the user to interactively segment the object in question [Zhang et al., 2002] unless it has been imaged in front of a simple background. A different approach is taken by Hoiem et al. [2005] where foreground, background and up-right labels are assigned to image patches, allowing to infer a simple depth map automatically.

2.1.3 Motion

Finally, besides different spatial dimensions, synthesis of visual media can also relate to temporal, i. e., time-varying aspects such as animations of three-dimensional objects. Explicit modeling of every single frame of these animations is a tedious task and can lead to

salient leaps in the animation if not done carefully. Hence, an important research area is concerned with the (semi-)automatic generation and extraction of animations.

Data-driven animation A popular approach to facilitate animating objects are data-driven techniques that aim at transferring motion from source to target objects. This can be achieved e. g., for a three-dimensional target object, such as an animated camel, with a three-dimensional source object as done by Sumner et al. [2004] (Figure 2.4). Other approaches work on simplified abstracted models such as skeletons, for which motion tracking systems are available. While this approach produces sophisticated results, motion tracking systems or three-dimensional animations of similar objects are usually hard to obtain by casual users. In contrast, other, potentially lower-dimensional animation sources such as videos or images are easily accessible, e. g., on internet video platforms. For this reason, Xu et al. [2008] reconstruct animal motion from a single or low number of images that show multiple animated poses of a walk cycle. Bregler et al. [2002] capture motion of two-dimensional cartoon characters and transfer it to three-dimensional character frames. These approaches work well if enough example frames and/or poses are present, but often only short sequences or sparse image collections are available. To enable smooth animations between these results, interpolation, i. e., *temporal upsampling*, can be used to compute in between frames of an animation.

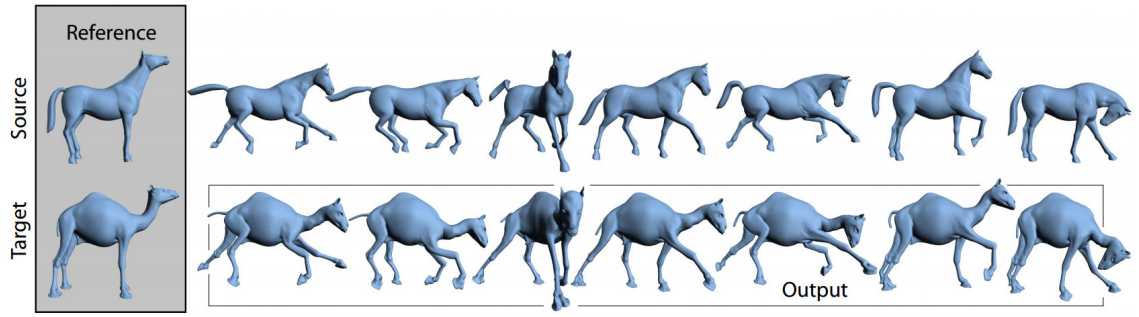

Figure 2.4: Motion transfer from a source model *(upper row)* to a target model *(lower row)* [Sumner and Popović, 2004]. *Image courtesy of the publication authors.*

Temporal upsampling of an animation leverages a sparse set of key frames and provides an efficient and powerful means to achieve both, reduce the amount of work required to produce the animation and generate smooth results. In combination with automated key frame extraction it can be used to constitute full animations and allows for smooth time warping of animations. It requires model knowledge that can be generic, as with constrained velocities between frames, or domain-specific, including model knowledge, e. g., rigidity constraints (cf. Section 2.2.1). Our approach in Chapter 6 includes both, generic as well as model knowledge, and can in many cases reproduce the complicated gait pattern of an entire limb in an animal walk cycle from a single user annotation, including occlusion handling. Further it solves for shape and animation in combination. Favreau et al. use segmentation on videos to extract a small set of key frames that represent the principal components of

animal gait patterns [2004]. A linear combination of these components then constitutes the full animal gait. Their approach allows transferring motion to three-dimensional models, but does not reconstruct the true three-dimensional shape from two-dimensional videos as in our approach. Automatic and semi-automatic creation of animation from examples [Arikan and Forsyth, 2002] or annotations [Arikan, Forsyth and O'Brien, 2003] has been proposed, but requires geometry given a-priori and a skeletal animation hierarchy. Plausible in-between frames can be created from user annotations [Sỳkora, Dingliana and Collins, 2009], but will not reproduce motion details present in the source video. It builds upon user-guided segmentation [Agarwala et al., 2004] of animated objects in video, but to the end of reconstructing an animated three-dimensional shape.

Reconstructing motion from simple annotations, such as sketches, has received less attention than more sophisticated segmentations. In Chapter 6 we build a global motion model [Bergen et al., 1992] from strokes on the fly allowing the user to inspect the resulting three-dimensional shape and animation quality and refine the input.

Optical flow [Brox et al., 2004; Tao et al., 2012; Sanchez Perez, Meinhardt-Llopis and Facciolo, 2013] establishes dense correspondences between frames over time and can be used to transfer results from one frame to adjacent frames. It can be employed to animate objects or transfer edits, but the missing domain-specific knowledge in these generic methods often lead to inferior results compared to specialized methods detailed before.

Physical simulation, as described in Section 2.2.1, is closely connected to temporal upsampling as it can serve as a prior that includes domain-specific knowledge into the upsampling process. It allows to regularize the results obtained by temporal upsampling to only permit plausible deformations and can limit the design space to only these reasonable solutions [Nguyen, 2014].

Our technique in Chapter 6 is a hybrid method that combines data-driven approaches from video feature tracking with model knowledge for temporal upsampling. Generally, motion fits in both categories, synthesis (cf. Section 2.1) and manipulation (cf. Section 2.2) as it, on the one hand, synthesizes motion from examples or other provided data but, on the other hand, also utilizes this data to deform existing surfaces.

2.2 Media manipulation

Media manipulation is concerned with the manipulation of existing, visual content to meet the constraints and intentions of the users. In this thesis we particularly manipulate two- and three-dimensional surfaces and entire collections of such surfaces that create a layout. First, deformations of surfaces of different dimensionality are discussed. Next, as most of the approaches presented in this thesis are example-driven, an overview of example-based approaches is presented.

2.2.1 Model deformation

Deformations of images [Wolberg, 1998] and surfaces [Sederberg and Parry, 1986] have a rich history in computer graphics and pose a classical avenue of visual media manipulation involving prescribed content and deformation models. For three-dimensional models, a wide variety of software (e. g., Blender [2016]) exists that allows to model and manipulate meshes. These applications enable full control of every step along the process of creating detailed three-dimensional models, but many of these steps involve tedious and repetitive manual work. A three-dimensional mesh, representing an object such as an animal, is defined as a complex $C = (V, E)$ with a set of vertices $V = \{\mathbf{v}_1 \ldots, \mathbf{v}_n \in \mathbb{R}^3\}$ and edges $E = \{e_1, \ldots, e_m \in \mathbb{N}^2\}$. Modeling of meshes typically also characterizes sensible manipulations or deformations of existing meshes. A typical way of representing such deformations is explicitly prescribing the positions of a small set of constrained vertices [Welch and Witkin, 1994; Sorkine et al., 2004; Zorin, Schröder and Sweldens, 1997; Witkin and Kass, 1988; Arikan and Forsyth, 2002]. The remaining, unconstrained vertices then should follow these constrained vertices in a plausible way, ideally respecting physical laws.

If rigs [Baran and Popović, 2007] are available for the meshes, off-the-shelf software supports deformations by manipulating the rig vertices that in turn cause the assigned model vertices to move accordingly. Most commonly however, meshes available in three-dimensional model databases, such as Google Warehouse, do not include such rigs and modeling them by hand is involved and requires vast amounts of manual work (Figure 2.5, b). What is more, for some models, rigs might be too restrictive as they only allow approximately rigid deformations. Hence, handling model deformations in an automated and expressive way has the ability to significantly improve the modeling process.

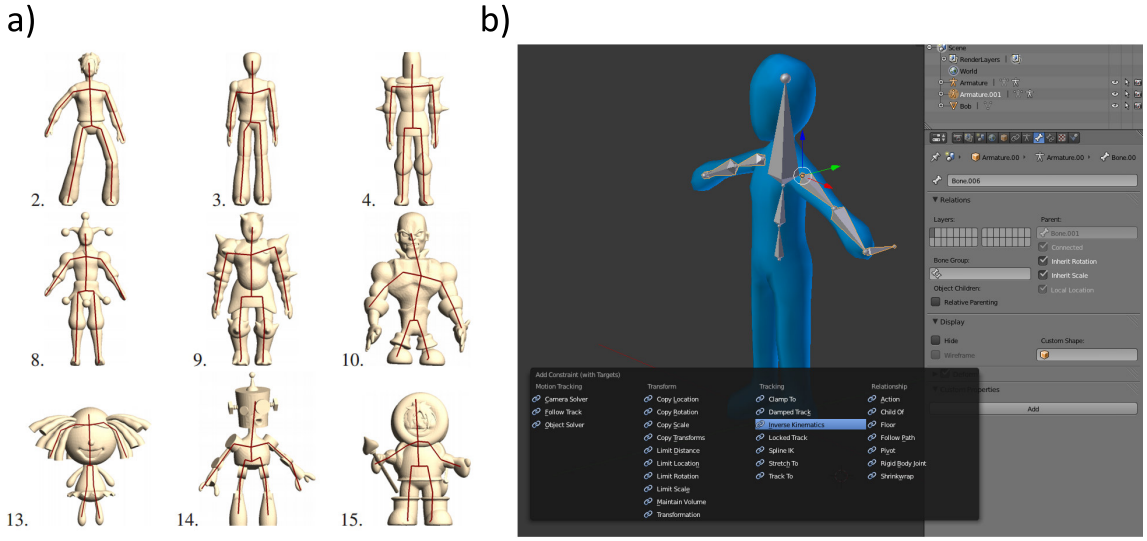

Figure 2.5: Rigging. *a):* Rigs can be automatically constructed for meshes using the technique of [Baran and Popović, 2007]. *b):* The Blender [Blender Online Community, 2016] interface to manually define rigs for a given model allows the manipulation of many parameters. *Images courtesy of the publication authors.*

One solution is to automatically model and assign rigs for existing meshes [Baran and Popović, 2007], but this approach is most commonly restricted to certain classes of known models (Figure 2.5, a). Explicit assignment of physical material properties, such as mass and elasticity, to parts or the entirety of the model is another possibility to accomplish plausible deformations [Popović, Seitz and Erdmann, 2003; McNamara et al., 2004; Wojtan, Mucha and Turk, 2006; Schulz et al., 2014]. Other works have focused on simpler means that do not explicitly relate to physical properties but model plausible model behavior [Botsch and Sorkine, 2008]. Deformations that minimize distortions [Alexa, Cohen-Or and Levin, 2000; Schaefer, McPhail and Warren, 2006; Nealen et al., 2007] have received considerable interest in the past years. The deformations resulting from these methods aim to follow artistic constraints, react smoothly, and locally behave like rotations. In particular *as-rigid-as-possible* (ARAP) surface deformations [Sorkine and Alexa, 2007] are a famous example, where all edges in a neighborhood of a deformed vertex $\widetilde{\mathbf{v}}_i$ should be described by a local rotation of the original, undeformed vertex positions $\mathbf{v}_i$ as close as possible (Figure 2.6, a), i. e.,

$$\widetilde{\mathbf{v}}_i - \widetilde{\mathbf{v}}_j \approx \mathbf{R}_i(\mathbf{v}_i - \mathbf{v}_j), \forall (i,j) \in E,$$

with $\mathbf{R}_i \in \mathbb{R}^{3\times 3}$ as a rotation matrix (cf. Figure 2.7, a). The authors propose an efficient optimization that fixes the positions or rotations in turn, allowing to iteratively obtain sophisticated solutions.

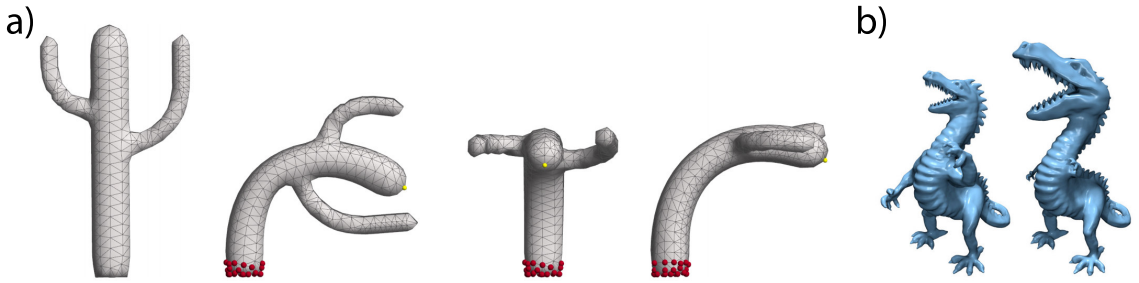

Figure 2.6: *a):* A source model of a cactus *(left)* and an example deformation using the red and yellow handles from different views *(right three images)* [Sorkine and Alexa, 2007]. *b):* Scaling the surface using the technique of [Botsch and Sorkine, 2008]. *Images courtesy of the publication authors.*

In the work presented in Chapter 3 we are interested in particular in local changes of size, where each part of a mesh can be assigned a different size, also known as non-homogeneous scaling. Non-homogenous scaling was addressed in the context of retargeting [Kraevoy et al., 2008], where the bounding box of one object is fit into another one, while trying to locally preserve salient regions and structures, e. g., such that spheres remain spheres. Our approach seeks to do the opposite, as it locally aims at rescaling but has to prevent intersections and minimize distortion. An important aspect in deformations often ignored in interactive applications is to avoid these self-collisions that frequently occur by scaling deformations in free space. The PriMo modeling system [Botsch et al., 2006] is a rare example that allows for local scaling of surface area (Figure 2.6, b), but without an account for intersections that occur for drastic and complex size changes such as the ones we target (cf. Figure 3.1).

The volume-preserving approach of von Funck et al. [2006] produces self-intersection-free vector fields, but is limited to certain types of supported deformations. The approach of Harmon et al. [2011] includes continuous intersection handling into the modeling process that allows for sliding motions, such as the one of cloth on a character, by eliminating space-time interference. Incorporation of internal anatomical structures such as bones, local size changes can be used to model muscularity and other anatomical properties while preventing self-intersections [Saito, Zhou and Kavan, 2015].

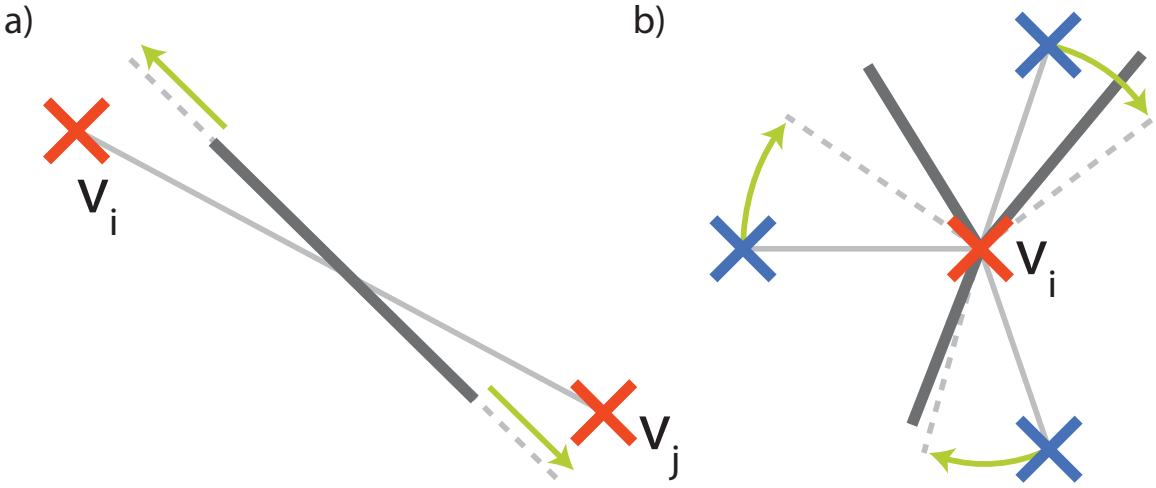

Figure 2.7: Deformation concepts. *a):* Constraint projection as used in [Müller et al., 2007], *b):* ARAP neighborhood rotation as used in [Sorkine and Alexa, 2007]. The figures show vertices *(red and blue crosses)*, previous configurations *(light grey edges)*, new configurations *(dark grey edges)*, corrected configurations *(stippled grey edges)*, and correction vectors *(green arrows)*.

As size changes often induce collisions that need to be resolved, multiple constraints, such as distortion minimization and collision avoidance, have to be optimized simultaneously. Combining multiple constraints in one optimization is a classical task of physical simulation. Physical simulation can be realized using various approaches [Terzopoulos et al., 1987; Nealen et al., 2006]; in this thesis we draw inspiration from one particular approach, coined as *position-based dynamics* [Müller et al., 2007]. Here, several constraints can be defined on the vertices and edges of the complex C to implement physical conditions. For example, all edges try to preserve their rest length as much as possible while other forces, such as gravity and acceleration, try to attract the vertices to new positions. The idea of position-based dynamics is to solve these constraints in turn using a so called constraint projection. Here, the vertices involved in a single constraint are projected to the closest configuration to the previous solution that satisfies the constraints. One example constraint, commonly used in this thesis, is the length constraint, representing distance preservation between adjacent vertices. The projection corrects the positions of both vertices of an edge to satisfy their rest length. For an edge $e = (a,b) \in E$ the correction between its rest length $\|\mathbf{v}_i - \mathbf{v}_j\|_2$ and its current length is distributed evenly to both vertices ($\widetilde{\mathbf{v}}_i$ and $\widetilde{\mathbf{v}}_j$) of an edge in the direction of

the edge, i. e.,

$$\widetilde{\mathbf{v}}_i = \widetilde{\mathbf{v}}_i + \left(\frac{\|\mathbf{v}_i - \mathbf{v}_j\|_2}{\|\widetilde{\mathbf{v}}_i - \widetilde{\mathbf{v}}_j\|_2} - 1 \right) \frac{\widetilde{\mathbf{v}}_i - \widetilde{\mathbf{v}}_j}{2},$$

with $i, j \in \{a, b\}, i \neq j$ (Figure 2.7, b). Other constraints include collision avoidance, detailed in Chapter 3. Inclusion of such collision detection and resolution in physical animation is common, but often ignored for shape manipulation or focus+context visualization. The objective of the approach in Chapter 3 is to achieve a static rest state where all constraints are balanced; the time-dependent change of a deformation subject to certain forces (i. e., of a piece of cloth) and momenta are not important for our application. The edge preservation constraint leads to similar results to those of ARAP surface deformations [Sorkine and Alexa, 2007] but are slower to optimize. They however allow to naturally include multiple other constraints into the optimization that can be weighted differently.

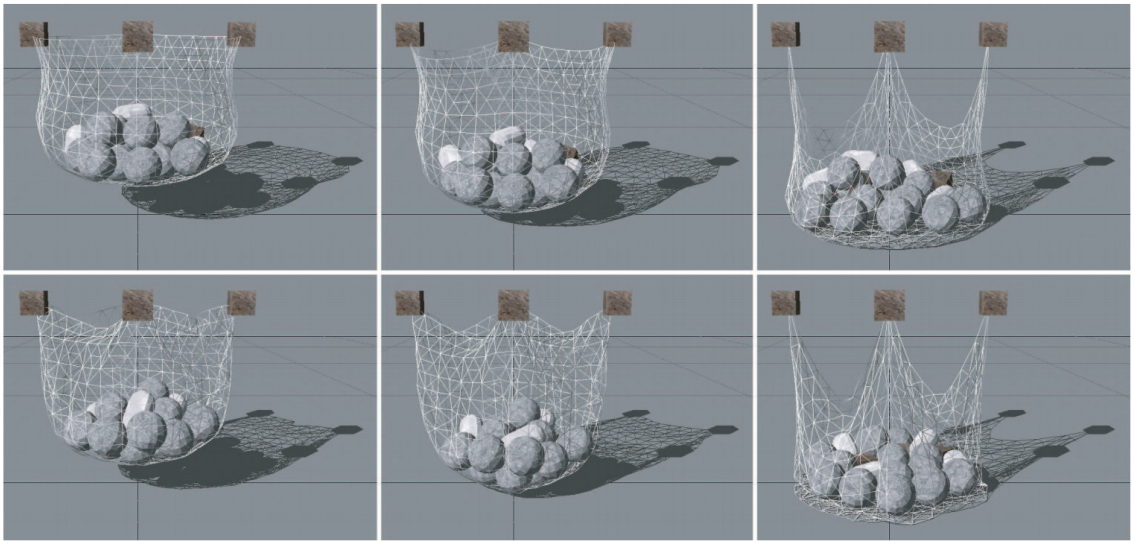

Figure 2.8: Position Based Dynamics. Example deformations showcasing different stiffness parameter configurations [Müller et al., 2007]. *Image courtesy of the publication authors.*

Besides the technical challenges that need to be solved, the visualization community has identified the opportunities provided by local size changes under the label of focus+context visualizations [Leung and Apperley, 1994]. The underlying magnifying glass metaphor was extended to three-dimensional surfaces and volumes [Wang et al., 2005]. Recent works combine all levels of detail in a single image with smooth transitions between the different levels [Hsu, Ma and Correa, 2011]. Different views and the space each one of them should cover in the final image can be defined and the method smoothly interpolates between these views. The result however is view-dependent and it is not clear if and how an extrapolation for different views could be added. Beyond lenses, other deformations can be applied in three dimensions to achieve focus+context. Wang et al. [2008; 2011] deform a three-dimensional surface to achieve focus+context by allowing to deform free space to distort more and minimize distortions of the surface. The problem in Chapter 3 is different in two ways: First, Wang's technique handles all voxels inside of a bounding cube whereas

our technique ignores voxels not occupied by mesh vertices. Second, their focus+context deformation field is intentionally of low frequency (e. g., a single large gradient proportional to the distance from a focus point). In contrast, the solution presented in Chapter 3 can be applied to both, high-frequency deformation, e. g., the crocodile teeth in Figure 3.6 as well as focus+context deformation as used by Wang et al. [2008; 2011], e. g., Figure 3.7, i.

2.2.2 Example-driven approaches

Example-driven approaches try to control manipulations of visual media by a small set of explicitly defined examples. The goal for such systems is to find the right appearance for the remaining, undefined instances by generalizing from the examples to the full model. In this thesis we make use of such approaches throughout most of our techniques, handling different kinds of media. A particular instance of this is three-dimensional model deformation where a sparse set of constrained vertices is defined and all other vertices follow these vertices in a meaningful way, detailed in Section 2.2.1. With size constraints the model can be solved for positions as done in Chapter 3.

Other examples include layout generation from a sparse set of exemplar images. Inferring a layout from sparse user constraints is an instance of semi-supervised learning [Chapelle, Schölkopf and Zien, 2010], in particular semi-supervised dimensionality reduction [Zhang, Zhou and Chen, 2007] where some primitives are labeled (i. e., placed by the user) but most are not. In the most general setting, the problem described in Chapter 4 can be regarded as (inverse) procedural modeling that can be solved with amazing results by approaches such as Metropolis sampling [Talton et al., 2011]. However, this approach is too costly to deliver timely response to the users' interaction. A classical way to reduce the dimensionality of a dataset is to use multi-dimensional scaling [Kruskal and Wish, 1978] or self-organizing maps [Kohonen, 1990] as done e. g., by Nguyen et al. [2015]. While these systems are very good at reproducing the distance matrix with a mixture of different dimensions, they lack the ability to explain the distances by a single, isolated dimension as required by our approach in Chapter 4, necessitating a sparse dimensionality reduction.

As shown in Chapter 5, layout inference and generation can be generalized to higher dimensions. Presentation of these layouts involves projection into lower-dimensional subspaces, such as two-dimensional images. These projective properties pose additional constraints on the layout generation that have to be optimized in combination with other constraints. In computer graphics such projections were studied intensively, especially for shadows (cf. e. g., [Mitra and Pauly, 2009]) to allow optimizing objects such that they produce predefined shadows when seen from multiple distinct viewports. However, arbitrary placement of three-dimensional objects such that they follow certain constraints and possess good distributions in three dimensions as well as in multiple projections (cf. Chapter 5) was not considered.

Figure 2.9: Example-driven approaches. *a)*: Shadow art [Mitra and Pauly, 2009] *b)*: Metropolis procedural modeling [Talton et al., 2011]. *Images courtesy of the publication authors.*

2.3 Point patterns

Point patterns are a key component for solutions to many problems arising in the field of computer graphics. On the one hand, they can be applied for numerical integration to approximate complex integrals, e. g., for rendering and imaging (cf. Section 2.1.1). On the other hand, in the fields of geometry processing and primitive placement, point patterns are used to constitute distinct locations for primitives, such as images or mesh vertices.

Numerical integration was already discussed in Section 2.1.1, hence we will briefly introduce other areas of application and its key requirements in further detail, explain how important pattern properties can be analyzed, and discuss specific pattern generation algorithms. In the following, a point pattern (Figure 2.10, a) of $n \in \mathbb{N}^+$ samples in dimension $d \in \mathbb{N}^+$ is defined as a set $S = \{\mathbf{s}_1, ..., \mathbf{s}_n \in [0,1)^d\}$.

Primitive placement Sample patterns for primitive placement are utilized in geometry processing [Chen et al., 2013a] but also for artistic purposes e. g., in two dimensions for non-photorealistic rendering, such as for stippling [Deussen et al., 2000; Hiller, Hellwig and Deussen, 2003], mosaics [Hausner, 2001; Kim and Pellacini, 2002], or texture synthesis [Lagae and Dutré, 2005].

In particular, the work of Hiller et al. [2003] who distributes primitives in the plane such that they follow a prescribed density, is a similar case of the system described in Chapter 4

that produces distributions that follow rules inferred from the users' interaction with the distribution itself. Properly distributing primitives in a domain has been a challenge for both, technical and aesthetical reasons, and remains demanding especially if interactive performance is crucial.

2.3.1 Pattern Properties

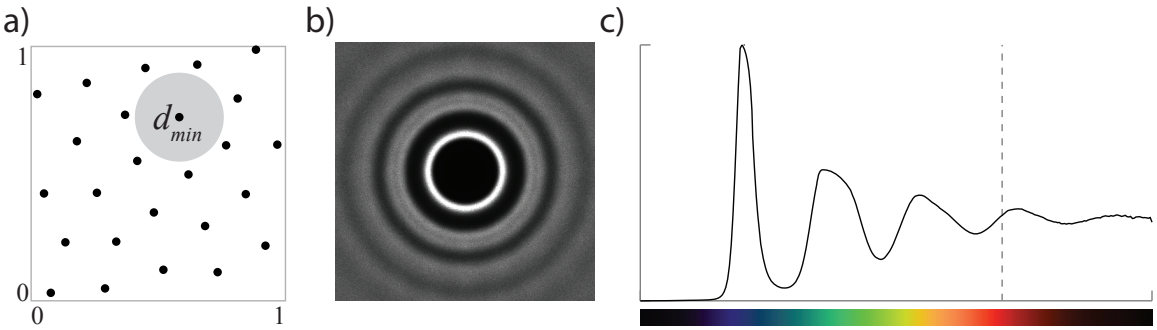

Figure 2.10: *a):* Point patterns with blue-noise characteristics have a minimum distance d_{min} to their closest adjacent point. *b):* The power spectrum of their Fourier transformation can be used to analyze spectral properties. *c):* Radially averaging the power spectrum gives insights about the pattern's quality.

Certain properties of point patterns can be analyzed to assess their usefulness in terms of quality and applicability. Several methods to assess the quality of point patterns have been proposed that allow objectively contrasting the point patterns in terms of different quantities. Most of the analysis is concentrated on two-dimensional patterns but can be generalized to higher dimensions. Additionally, patterns can be analyzed not only in their full-dimensional space but also in lower-dimensional subspaces, yielding insights about important projective properties.

One property is the patterns' progressiveness, i. e., the ability to insert points into existing patterns such that their distribution properties are preserved as much as possible. This enables adaptive insertion of points until a desired result quality is achieved while it progressively increases. Other properties include the so called *stratification*, *Latin hypercube* (LH), or (t,m,s)-*nets* property, which describe the exclusive coverage of certain subspaces of the $[0,1)$-domain, detailed in Section 2.3.2. Patterns can either be calculated on a bounded domain with explicit boundaries or a toroidally wrapped domain which can be interpreted as an infinitely tiled domain in all dimensions.

A key property for good performance of point patterns, both in terms of numerical integration and primitive placement, is a good distribution of the point samples. A direct measure of the pattern's performance for numerical integration is its discrepancy for which the pattern's coverage of the integration domain is investigated. Another tool is to analyze the patterns in terms of their spectral properties, most often using a Fourier transform. Coined by Ulichney [1987], the term *blue-noise* refers to the spectral properties of uniform and isotropic, yet structureless point distributions which implies a certain minimum distance

$d_{min} \in \mathbb{R}$ between the primitives (Figure 2.10, a). The color of noise can be determined by the radially averaged power spectrum of the patterns (plot in Figure 2.10, c). Interpretation as a spectrum of visible light would result in blue light (light spectrum in Figure 2.10, c). Originally developed for the purposes of digital half-toning [Ulichney, 1987], blue-noise distributions have been found useful in many other applications due to their resemblance of the photoreceptor arrangement in the eye's retina [Yellott, 1983].

Even though blue-noise point sets have a very uniform distribution and lack regularity, their application in image synthesis has so far been mostly restricted to image anti-aliasing [Mitchell, 1987; Pharr and Humphreys, 2010]. Reports on their performance for estimating illumination integrals have been controversial [Shirley, 1991; Schlömer, Heck and Deussen, 2011; Marques et al., 2013]. We hypothesize that the main reason for their sub-optimal performance is the poor uniformity in their low-dimensional projections, and aim to produce point sets with both blue-noise and LH properties in Chapter 5. Research in numerical integration has shown that the quality of a pattern can be improved by "latinizing" either the initial values for Lloyd relaxation or the final result [Romero et al., 2006; Saka, Gunzburger and Burkhardt, 2007]. However, no attempt has been made to achieve both LH and blue-noise properties simultaneously.

We will now detail the Fourier analysis and the discrepancy measurement.

Fourier Analysis The most commonly used analysis method for point patterns in computer graphics applications is the Fourier transform. Here, the power spectrum P of a pattern S is computed using the Fourier transform F as

$$P_S(\mathbf{f}) = |F_S(\mathbf{f})|^2 = \frac{1}{n}\left(\left(\sum_{\mathbf{s}\in S}\cos(2\pi\langle\mathbf{f},\mathbf{s}\rangle)\right)^2 + \left(\sum_{\mathbf{s}\in S}\sin(2\pi\langle\mathbf{f},\mathbf{s}\rangle)\right)^2\right),$$

for a frequency $\mathbf{f} \in \mathbb{R}^{+d}$ [Wei and Wang, 2011]. Dependent on the number of points of the pattern, these spectra are typically very noisy and averaging the power spectra of several pattern instances results in a smoothed spectrum that facilitates further analysis. Displaying an excerpt of the lower frequency range of this averaged spectrum as a greyscale image (Figure 2.10, b) allows to visually inspect and analyze the quality of a pattern. Anisotropy and regularity artifacts show up as features in these images (cf. Figure 5.4 – Figure 5.6). Radial statistics of these images, such as average and variance, give insights about the spectral properties of the underlying point patterns (Figure 2.10, c). To facilitate comparisons of spectral analyses among different publications, unification of parameter choices, such as the number of point samples and the frequency range, is crucial. Schlömer et al. developed a standardized way to produce these analyses [Schlömer and Deussen, 2010] also followed in this thesis. Chapter 5 extends this work to patterns of higher dimensions. Recently, the Fourier analysis has been shown to be misleading in terms of minimal sample distance and was improved by operating in the differential domain that is related more closely to actual sample distances [Wei and Wang, 2011]. Although not being directly related to integration error, a Fourier analysis gives adequate insights about the performance of a pattern for rendering applications in many scenarios.

Discrepancy The quality of a pattern for numerical integration can be measured by its *discrepancy* [Shirley, 1991; Mitchell, 1992]. While there are many discrepancy measures, typically the star discrepancy $L_*^\infty \in \mathbb{R}$ is used to assess the pattern quality, defined as

$$L_*^\infty(S) = \sup_{J \in Y} \left| \frac{\#(J,S)}{n} - Vol(J) \right|,$$

with $\#(J,S) \in \mathbb{N}^+$ as the number of points $\mathbf{s} \in S$ in J, $Vol(J) \in \mathbb{R}$ as the volume of $J \subseteq [0,1)^d$ and Y as the set of all d-dimensional subintervals of the form

$$J = \prod_{i=1}^{d} [0, u_i),$$

where $0 \leq u_i \leq 1$. While discrepancy provides good insights about integration error, the box integral of J might be too simplistic for real rendering applications. Consequently, reports on discrepancy measures for rendering performance have been controversial, as the important areas of the integral at hand might as well have a very different shape.

2.3.2 Pattern Generation

In the following we will detail several algorithms to generate point patterns.

Regular sampling The simplest way to get a uniform coverage of the domain $[0,1)^d$ is to produce point patterns on a uniform grid, i. e., placing the points at equidistant locations (Figure 2.11, a). Inter-sample distances can be improved by employing a hexagonal grid or, in higher dimensions, the tightest sphere packing [Lagae and Dutré, 2006]. Uniform patterns require sample counts that are squares of natural numbers, are not progressive, and produce regularity artifacts clearly observable in many applications, such as rendering (cf. Figure 5.14).

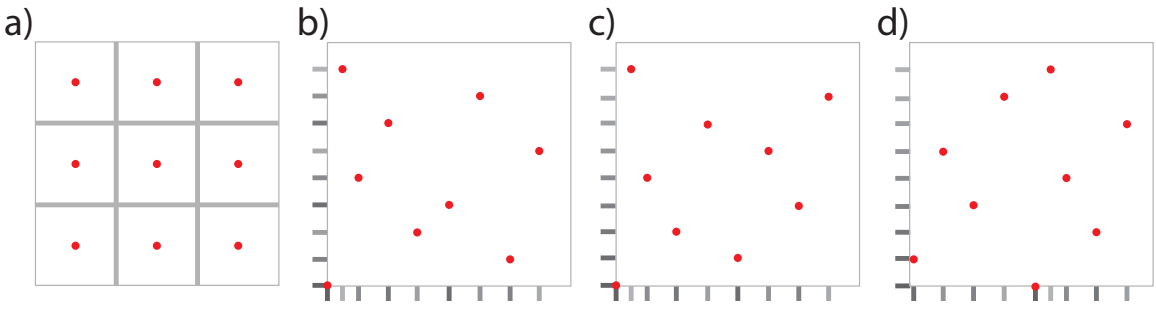

Figure 2.11: *a):* Uniform sampling, *b):* Halton sequence, *c):* Hammersley sequence, *d):* Shuffled Hammersley sequence. The greyscale markers in c – d show the respective order in which the samples were inserted *(dark to bright)*.

Random sampling To overcome the regularity artifacts present in uniform sampling, points can be inserted at random locations. However the white noise power spectrum of these patterns leads to an uneven coverage of the domain, i. e., it produces holes and clusters (Figure 2.12, a).

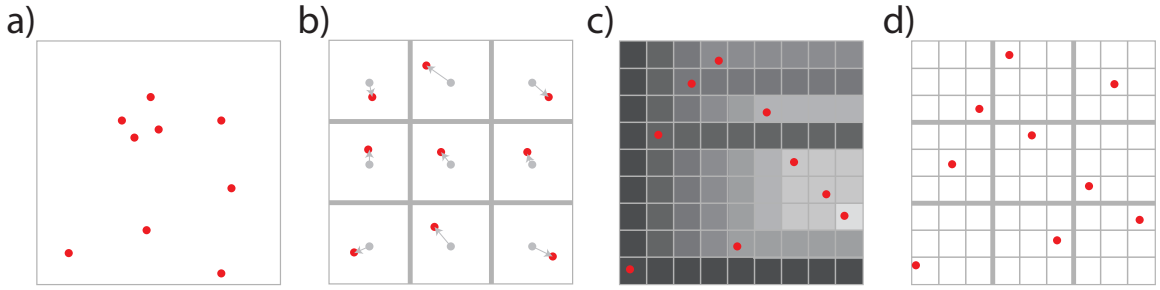

Figure 2.12: *a):* Random sampling, *b):* Jittered sampling, *c):* Latin hypercube sampling, *d):* Multi-jittered sampling. The arrows in *b)* depict the positions of the uniform grid that are randomly displaced. In *c)* the colors show the rows and columns occupied by previous samples *(dark to bright, left to right)*.

Combining uniform and random sampling to produce uniform but irregular point patterns can reduce aliasing [Dippé and Wold, 1985] and noise [Cook, 1986] in Monte Carlo rendering. Such applications have traditionally relied on *stratified* pseudo-random sampling, which places one sample in every stratum of the uniformly subdivided sampling domain [Pharr and Humphreys, 2010]. One instance of this idea is *jittering*, where each sample on a regular grid is displaced randomly by a bounded offset (Figure 2.12, b). Stratification can often increase the error convergence rate of the Monte Carlo estimator over pure random sampling [Mitchell, 1992]. Latin-hypercube, or N-rooks, sampling enforces stratification along each axis of a high-dimensional point set by invalidating each row and column of a grid per sample for all future samples (Figure 2.12, c). Jittered and LH sampling have been combined to produce high-quality numerical integration patterns [Chiu, Shirley and Wang, 1994; Kensler, 2013] (Figure 2.12, d).

Quasi Monte-Carlo sampling *Quasi Monte-Carlo* (QMC) patterns overcome the limitations of uniform sampling using low-discrepancy sequences. The sequences build on the existence of an expansion of an arbitrary number $a \in \mathbb{N}^+$ using a prime base $b \in \mathbb{N}^+$ as

$$a = a_0 \cdot b^0 + a_1 \cdot b^1 + a_2 \cdot b^2 + \cdots + a_n \cdot b^n,$$

where $a_i \in [0, b-1]$ for all $i \in [0, n]$. We can define a function ϕ that constitutes a low-discrepancy sequence for basis b as

$$\phi_b(a) = a_0 \cdot b^{-1} + a_1 \cdot b^{-2} + \cdots + a_n \cdot b^{-n-1}.$$

Using sequences with different values for b – one per dimension – we can now define d-dimensional points. For the special case of $b = 2$ the sequence ϕ_2 is called *Van der Corput* sequence [Halton, 1964]. Several construction patterns involving different choices for the dimensions have been proposed and we will introduce two well-known patterns

for the two-dimensional case. The samples $\mathbf{s}_i \in S = [0,1)^2$ of a so called *Halton* sequence (Figure 2.11, c) can be found as

$$\mathbf{s}_i = (\phi_2(i), \phi_3(i)) . \tag{2.3}$$

The samples of a *Hammersley* sequence (Figure 2.11, b) are defined as

$$\mathbf{s}_i = \left(\frac{i}{n}, \phi_2(i) \right) . \tag{2.4}$$

QMC patterns outperform uniform sampling as they produce lower discrepancies and, as long as all dimensions are produced with low-discrepancy sequences, are progressive. However, they only produce perfect stratification along each axis for point counts that are powers of b and still suffer from regularity artifacts, especially when one axis is sampled equidistantly. In that case the patterns are also not progressive (cf. Hammersley sequence, Equation 2.4). Regularity artifacts can be overcome by randomization but naïve offsetting by a constant would destroy important properties of QMC patterns, such as the (t,m,s)-nets property. Specialized shuffling algorithms can be used that produce randomized results while preserving these properties [Kollig and Keller, 2002]. One example can be seen in Figure 2.11, c and d. This preservation however comes at the cost of lacking randomness, resulting in some remaining regularity artifacts (cf. Figure 5.4).

Low-discrepancy point sets, such as QMC patterns, have seen wide adoption in physically-based rendering, as they are relatively simple to implement and possess excellent stratification and LH properties [Kollig and Keller, 2002; Pharr and Humphreys, 2010; Keller, Premoze and Raab, 2012]. However, as discussed before, most low-discrepancy sampling methods achieve good stratification only for a restricted number of points (e. g., powers of two), depending on the construction method [Keller, Premoze and Raab, 2012], limiting their applicability especially for progressive sampling. The patterns generated by the algorithms in Chapter 5 do not have this limitation and perform on par with or better than such methods for rendering and primitive placement.

Dart throwing One popular way to generate blue-noise point sets is to insert points one by one, while maintaining a minimum inter-sample distance d_{min}. The most common approach for doing this is dart throwing [Cook, 1986], which ensures that points are tightly packed but no closer than a specified minimum distance, producing a so-called *Poisson-disk distribution* (Figure 2.13). This algorithm can be made progressive by adaptively shrinking the minimum distance after a certain number of failed insertion attempts [McCool and Fiume, 1992]. Alternatively, a new point can be inserted at the location farthest from the existing set [Mitchell, 1991; Eldar et al., 1997]. Various enhancements of this basic approach have been proposed recently, regarding its performance [Dunbar and Humphreys, 2006; Jones, 2006; Gamito and Maddock, 2009; Kalantari and Sen, 2011] as well as its ability to produce non-uniform [Wei, 2010; Öztireli and Gross, 2012; Chen et al., 2013a], anisotropic [Li et al., 2010], and multi-class [Wei, 2010] blue-noise point sets.

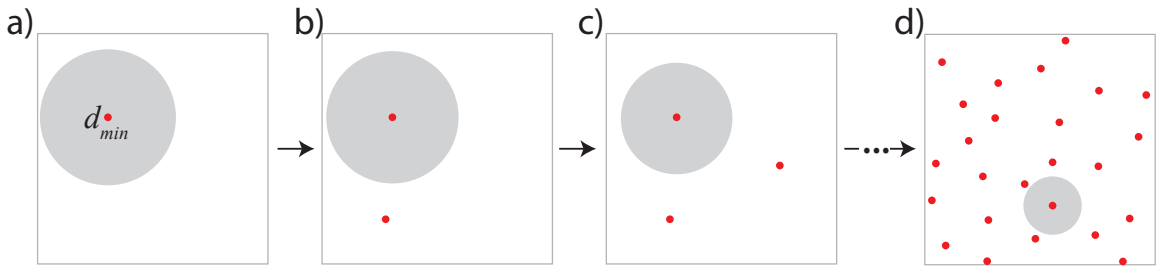

Figure 2.13: Dart throwing: Samples are inserted one by one *(a–c)* obeying a specific minimum sample distance d_{min} that is progressively shrunk with increasing number of points to produce the final pattern *(d)*.

Lloyd relaxation Another approach that produces high-quality blue-noise sample patterns is to start from an initial, e. g., random (cf. Random Sampling), sample set and maximize the minimum inter-point distance using an iterative optimization scheme. The popular Lloyd relaxation algorithm [Lloyd, 1982] (Figure 2.14) is based on *Centroidal Voronoi Tessellation* (CVT). CVT is prone to slight, but visible regularity artifacts and the quality of the resulting patterns has been improved by Balzer et al. [2009] avoiding spurious regular hexagonal patterns. Some further developments have addressed anisotropic sampling [Li et al., 2010] as well as improving efficiency [de Goes et al., 2012; Chen et al., 2012]. In an effort to further improve the performance, flexibility, and spectral properties of blue-noise sampling, various optimization methods have been proposed recently [Fattal, 2011; Schlömer, Heck and Deussen, 2011; Öztireli and Gross, 2012; Heck, Schlömer and Deussen, 2013; Chen et al., 2013a].

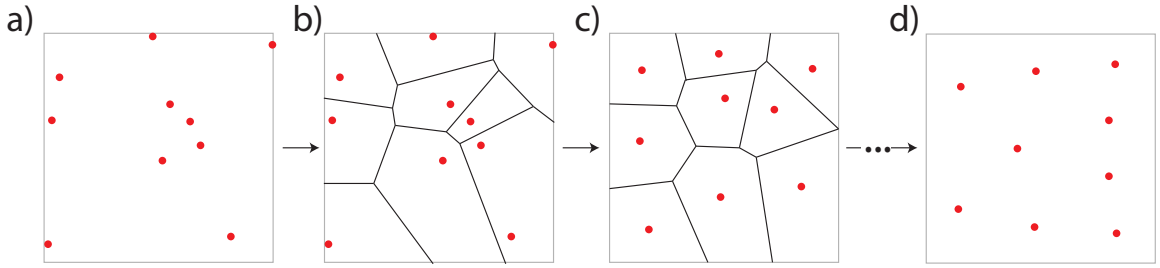

Figure 2.14: Lloyd relaxation: The Voronoi diagram for all samples is computed *(a)* and the samples are moved to the center of mass of their corresponding region *(b)* iteratively *(c)* until convergence *(d)*.

2.3.3 Point Patterns for Primitive Placement

Apart from numerical integration, other works have demonstrated the utility of blue-noise sampling for procedural primitive placement. This application also benefits from the real-time performance of tile-based sampling [Ostromoukhov, Donohue and Jodoin, 2004; Kopf et al., 2006; Ostromoukhov, 2007; Wachtel et al., 2014]. Computational placement of extended primitives in two dimensions that uniformly fill the given space [Hiller, Hellwig

and Deussen, 2003] has applications in automated generation of layouts in print, on screens, and for fabrication. For placing primitives in the plane, the usage of Voronoi tessellation is popular to avoid and resolve collisions [Dalal et al., 2006] and achieve pleasant (temporal) distributions.

The Lloyd relaxation algorithm can be generalized to support samples with arbitrary sizes and shapes by employing a generalized Voronoi diagram (Figure 2.15). This enables efficient, interactive generation of pleasant packing layouts. If shapes are used that cannot be well approximated by ellipsoids, this approach however can lead to drift und unpleasant layouts (cf. Figure 4.5).

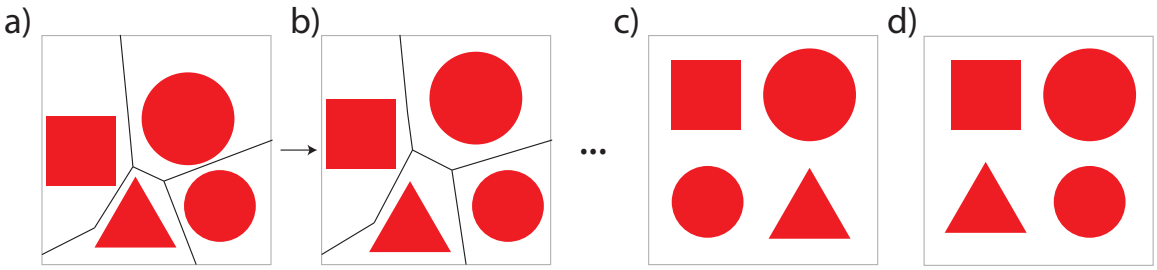

Figure 2.15: Generalized Lloyd relaxation [Hiller, Hellwig and Deussen, 2003]. A generalized Voronoi diagram for all samples is computed *(a)* and the center of mass of each primitive is moved to the center of mass of its corresponding Voronoi region *(b)* iteratively until convergence *(c)*. *d):* A different converged outcome obtained from different initial positions.

In contrast to numerical integration where each point sample constitutes the same entity, for layout generation, every sample is associated with a specific primitive, such as an image, and hence each sample represents a different entity. Consequently, besides analyzing the relationships and distances of adjacent points, as done by most pattern generation algorithms, we can additionally analyze the relationships on a more macroscopic or global level. This view allows us to convey certain relationships between the different primitives, e. g., by clustering similar elements, and has been an active area of research. The results in Figure 2.15, c and Figure 2.15, d, for example, convey different element relationships while their blue-noise characteristics are identical. The parameters for this primitive placement can be difficult to control as noted by Hurtut et al. [2009] and Öztireli et al. [2012], who proposed to transfer the statistics from a source to a target distribution of primitives. Our approach in Chapter 4 is not based on distribution statistics. Instead, we infer high-level rules that describe the intended embedding of a high-dimensional feature space into a low-dimensional medium from user input instead of spatially-invariant statistics of items that cannot express all of the users' intentions. Exploration of high-dimensional spaces of visual features was described by Lasram et al. [2012]. Beyond distribution statistics, grouping for stylization was described by Bezerra et al. [2008]: A layout of a scene is given, and the style of items is made coherent according to an observed grouping. A subset of our approach in Chapter 4 performs the inverse: We are given primitive features (e. g., brightness, shape, size) and want to find a layout.

Apart from the macroscopic positioning, also the smaller-scale relationships of adjacent samples are different due to the spatial extend of the primitives. Optimally placing a set of spatially extended objects into a constraining container, also known as *bin packing*, such as three-dimensional shapes into another three-dimensional shape [Gal et al., 2007] or text into a two-dimensional contour [Xu and Kaplan, 2007; Maharik et al., 2011] is an NP-hard problem, but can be solved with sufficient approximate solutions in practice. Different goals can be formulated to either use up the least space [Lodi, Martello and Vigo, 2002] or to fill the given container evenly [Hiller, Hellwig and Deussen, 2003]. Packing into a container can be one constraint among many, like in our system described in Chapter 4. Packing UV charts [Lévy et al., 2002] is a common technical challenge for surface parametrization. Closest to the objective described in Chapter 4 is the approach of Yu et al. [2011] that arranges furniture in a room according to rules learned from exemplars in a forward procedure, but without assistance from the users to change the layout or to learn from their feedback.

While producing images using projections, e. g., shadows (cf. [Mitra and Pauly, 2009]), has been addressed in computer graphics, no prior work has considered the *spectral* properties of projections of primitive layouts. This becomes important when placing (fabricated) objects in three-dimensional space, i. e., in a physical exhibition or a collaborative virtual environment. Typically employed patterns do not produce layouts with blue-noise characteristics when observed from different viewpoints. The approach in Chapter 5 can reduce clutter and occlusion by optimizing both the spatial and the projected arrangement.

2.4 Interactivity

Interactivity is an essential requirement for any system that tries to keep the user in the loop, i. e., that allows users to iteratively refine the outcome of an algorithm until a desired result is achieved. Interactivity relates to both, a system that the user can intuitively interact with but also a system that is fast enough to present results at interactive rates. The first requirement can be targeted using intuitive *user interfaces* (UIs) that ideally are easily comprehensible also by casual users. The second requirement can be approached with efficient algorithms that achieve *interactive performance*.

2.4.1 Intuitive User Interfaces

User interfaces try to provide the user as much control as possible over algorithms while maintaining a high level of intuitiveness. These two contradicting goals usually cannot be optimized in combination. Hence, several techniques and conventions have been proposed that allow for intuitive operability. For example, a rich set of so-called *widgets* has been developed [Swick and Ackerman, 1988]. These UI elements combine layout and interaction elements such as buttons, sliders, tabs, etc. with interaction metaphors such as drag-and-drop (Figure 2.16, a). Their intuitiveness stems from their utilization amongst many different systems and they are the de-facto standard in UIs. Several widgets are typically combined into a widget collection to allow control of an algorithm, requiring them to be layouted in

an efficient and intuitive way. This assembly is one of the most classic layout problems in desktop publishing and user interfaces. Automated ways to compute them are an open and active area of research [Lok and Feiner, 2001; Jacobs et al., 2003]. Here, the state of the art is based on systems that exploit the regular grid-structure of text layouts, which however does not generalize to arbitrary widget shapes. A limiting factor of these layout elements is their rectangular shape that is even further restricted by constrained column and row dimensions. Distributing elements of different shape still poses a difficult problem for which no optimal solutions are known. A more natural way of distributing the widgets could be to use up the given space more evenly and adaptively distribute the elements based on the content, as done e. g., in information visualization. Here, graph drawing [Harel and Koren, 2002] and specifically word clouds [Bateman, Gutwin and Nacenta, 2008; Strobelt et al., 2012] share challenges such as collision avoidance with this approach. Placement of textual labels by example was considered by Vollick et al. [2007].

Widgets are good at allowing full control over a large range of abstract parameters. In many scenarios, especially when concerned with media generation, these parameters have a direct visual meaning and controlling them with widgets detaches the user from this immediate representation. Arriving at the desired result often requires multiple parameter combinations and many iterations of trial-and-error. A more intuitive way of content creation in these scenarios would be to let the user directly interact with the media at hand. To do this efficiently, the system could learn from the users' examples and interactions to generalize the appearance and estimate the parameters as described in Section 2.2.2. For word clouds, user input has only been included in a forward manner in the ManiWordle system [Koh et al., 2010], where a user can fixate individual word primitives to specific locations. Different from such off-line systems, in Chapter 4 we account for visual features of the primitives themselves (and not only abstract word frequency), learn layout from user feedback, and present new layouts, all on-line, with interactive performance. For layout problems such direct interactions could involve positioning of some elements to their desired location and generalizing the locations of the other, unconstrained elements (cf. Chapter 4).

Another powerful example-based interaction metaphor is sketching, especially popular in the generation of three-dimensional models, where the projections of parts of the models are drawn. The first system to create three-dimensional surfaces from sketches was proposed by Zeleznik et al. [1996]. Later, the "Teddy" system [Igarashi, Matsuoka and Tanaka, 1999] introduced intuitive modeling of free-form, three-dimensional surfaces from simple sketches (Figure 2.16, b). The SmoothSketch system [Karpenko and Hughes, 2006] allows the user to draw the silhouette, from which a smooth surface is created similar to the creation of silhouettes from photos [Prasad and Fitzgibbon, 2006]. A different approach for user annotation is to directly scribble depth inequalities, e. g., for cartoons [Sỳkora et al., 2010] or normals [Wu et al., 2008]. As many real-world objects can be approximated using simple geometric primitives as proxies, like cuboids [Zheng et al., 2012] or generalized cylinders [Chen et al., 2013b] (Figure 2.16, c), some recent systems snap the user sketches directly to a reference photo with interactive feedback. Sketching motion [Guay et al., 2015] is another area where such user interfaces can be useful (Figure 2.16, d). Modeling objects by sweeping a profile along the main axis e. g., a generalized cylinder has been used as an intuitive modeling metaphor for many objects [Choi and Lee, 1990]. The ArtiSketch system

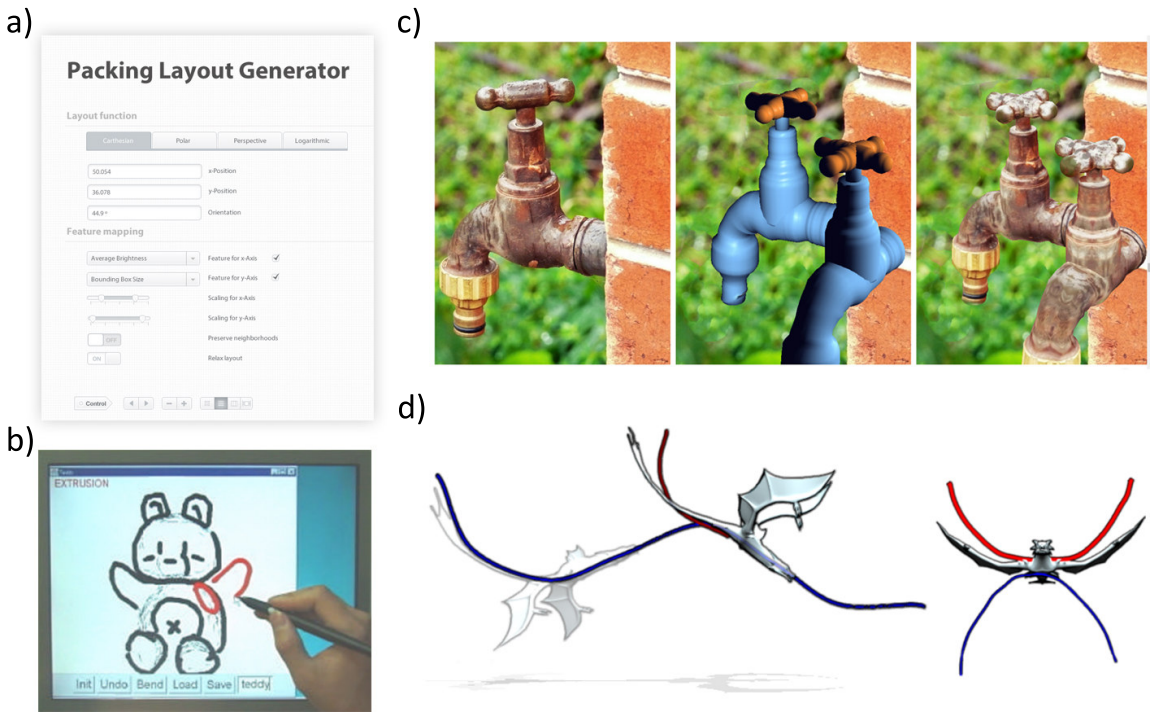

Figure 2.16: *a):* Classical user interfaces consist of a set of widgets. *b):* Sketching can be used as a more intuitive user interface e. g., to model three-dimensional geometry [Igarashi, Matsuoka and Tanaka, 1999], *c:)* to extract geometry from monocular images [Chen et al., 2013b], *d):* or to sketch motion [Guay et al., 2015]. *Images courtesy of the publication authors.*

[Levi and Gotsman, 2013] reconstructs animated surfaces from sketches on video frames in an offline process and demonstrates applications for comic characters. It requires the user to place a skeleton that was designed in an external application and to sketch the outlines of the shape components. Our approach in Chapter 6 combines skeleton and surface estimation in a single interaction metaphor and runs at interactive rates for natural image input of detailed, natural, animated surfaces.

2.4.2 Interactive performance

Interactive performance involves optimization of runtimes of algorithms realizable using different approaches. A trivial solution is to replace the hardware with more powerful machines, which for casual users and mobile devices however often is not an option due to cost and power consumption limitations. Other solutions include complexity reduction of algorithms leveraging typical properties of the problems at hand, e. g., using dynamic programming [Buchanan and Fitzgibbon, 2006], or implementation of specialized data structures that improve efficiency, e. g., hash tables [Cormen et al., 2001]. Reformulation of the problem to reduce its complexity is another possibility. Approximation of the exact solution or presentation of simplified previews provides another avenue of realizing interactive performance. These methods provide solutions close to the final result and the

ultimate, complete solution can then be calculated in an offline or background process, if desired (Chapter 3). Finally, a classical way of improving the runtime of algorithms is to use *parallel computing*. For this approach the problem at hand must be divisible into smaller subproblems that can be solved and combined for the final solution (cf. [Dean and Ghemawat, 2008]). Most of the problems at hand are not trivially parallelizable but have to be reformulated. Many of the problems presented in this thesis allow for parallel execution.

Parallel computing In the last decade the so-called *power-wall* for processors was hit, preventing the CPU to ever become exponentially faster as stated by Moore's law [1998]. To remedy this limitation, instead of faster processors, modern CPUs nowadays possess multiple cores allowing parallel execution of algorithms. This paradigm shift lead to an increased interest in optimizing algorithms by parallelization. Typically, the number of parallel cores per CPU is still low (≤ 8) enabling only a minor degree of parallelism. Many problems, especially those arising in computer graphics, are inherently massively parallel, i. e., can be split up into hundreds of thousands of much smaller subproblems. For such problems an optimized, parallel co-processor is available, the GPU.

Originally designed mainly for rendering computations that often allow for massive, trivial parallelism [England, 1986], GPUs evolved into sophisticated, general purpose co-processors that can be used to speed up a wide variety of algorithms [McCool, Qin and Popa, 2002]. While early programmable GPUs required considerable low-level programming overhead, the interfaces to program the hardware have emerged into an abstract set of tools that enable rapid development, monitoring, and debugging of parallel algorithms. A variety of sophisticated toolsets (APIs) for these tasks is available, such as CUDA [NVIDIA, 2015], OpenGL [Shreiner et al., 2013], OpenCL [AMD, 2015], or DirectX [Luna, 2012], and the field is ever evolving [Kessenich and Sellers, 2016]. To allow for the massively parallel execution of threads, the layout of a GPU is fundamentally different from that of other processors. This layout poses some limitations, prohibiting the trivial transfer of parallel algorithms designed for other processors, such as the CPU. For example, scheduling, an evolved topic for CPUs and operating systems [Tanenbaum, 2007], becomes much more involved on GPUs due of the massive parallelism. Other limitations include expensive synchronization steps with the CPU and inefficient memory management. Most programming interfaces for GPUs come with build-in functionality to remedy these issues to some degree in many cases (e. g., scheduling [NVIDIA, 2012]) but specialized solutions for the tasks at hand are capable of substantially improving the performance. A comprehensive overview of the limitations and potential solutions can be found in the thesis of Steinberger [2013]. While optimization of GPU scheduling poses a potentially beneficial avenue of research, this thesis mostly builds upon operations performed with OpenGL, where the described low-level tasks are handled by the hardware driver. An informed choice of representation can alleviate typical scheduling and memory management problems by reformulating the problems at hand to map well to the GPU layout. Operating on implicit, regular grids with pre-defined, implicit connectivity between adjacent nodes enables problems to be parallelized more efficiently. Many of the problems that arise in this thesis can be formulated to operate on such grids, making usage of the parallel hardware even more beneficial. Aggregation of several results into a combined result is another task that frequently arises

in computer graphics problems. Leveraging pyramidal structures that aggregate multiple results of smaller levels into coarser level results until a coarsest level is reached, maps well to the GPU and has direct hardware support using MIP mapping.

Chapter 3

Homunculus Warping

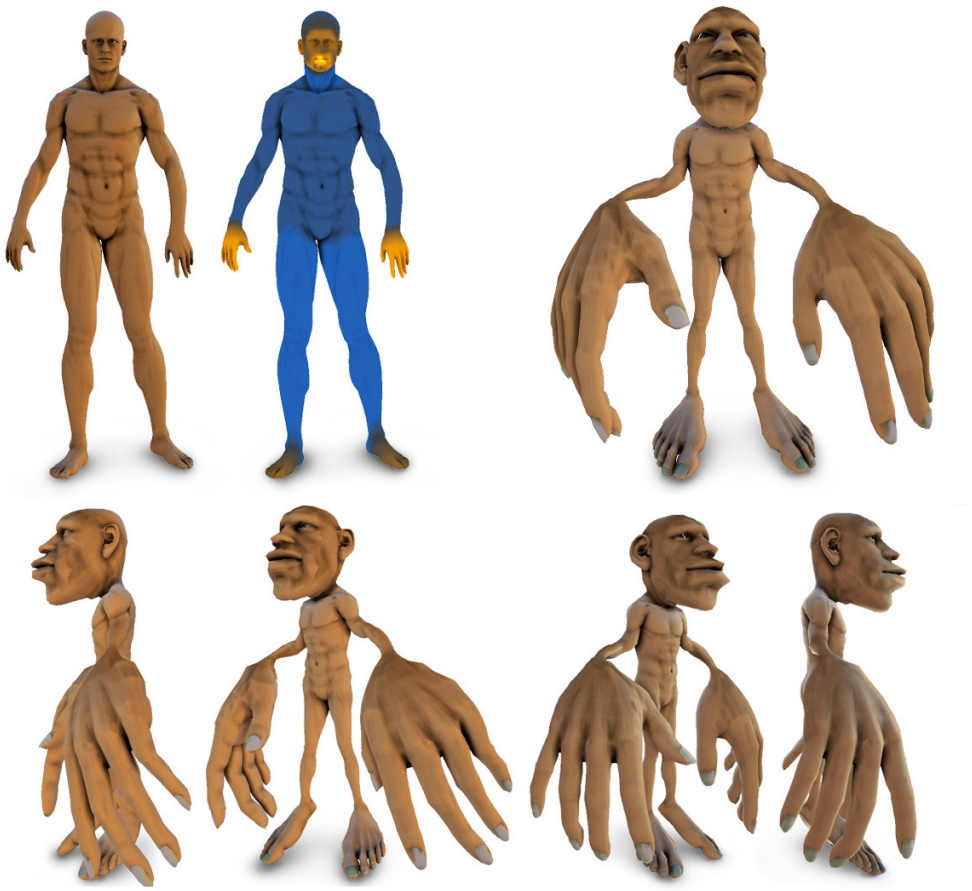

Figure 3.1: *Upper row, left to right:* An undeformed, three-dimensional model, importance defined by false colors on the surface of the model, and the resulting deformed model that resembles the defined importance. *Lower row:* Four additional views of the according Homunculus Warping.

3.1 Introduction

Simple and effective visual representations of complex relations are hard to come by [Tufte, 1997]. While there are many different approaches and diagram types, the problem is to pick one that is suitable for the task at hand and is understood immediately. To clarify the relationship between different data dimensions it is sometimes useful to visualize one dimension on the domain of a different one, e. g., to illustrate the number of inhabitants of a country on a world map. A common way of visualizing this data is to use false color coding, despite all its known shortcomings [Borland and Taylor II, 2007]. The main shortcoming is the complex human color perception that varies considerably among different viewers and even penalizes achromates as well as the conflicts it causes for three-dimensional surfaces in combination with shading.

Figure 3.2: Four examples of local size changes to convey importance: The motorical *(a)* and sensorical *(b)* homunculus (Natural History Museum, London). *c)*: A renaissance painting by Gentile da Fabriano (1370–1427): "Mary Enthroned with the Child, Saints and a Donor" depicting the donors as smaller and less important. *d)*: Miniature from the "Tetraevangelia of Ivan Alexander" (14th century, Bulgaria) depicting the tsar and his wife as more important than his children. *e)*: Limestone engraving from the tomb of Akhenaten (2nd century BC, Cairo Museum, Egypt) where cascading size conveys importance.

Generally perceived differences in size are remarkably invariant under different viewing conditions [Cutting, 1987; Gregory, 1963] especially for familiar objects [Bingham, 1993] e. g., the human body. Hence an alternative to false color coding is to encode the scalar field directly as size.

A prominent contemporary example of this type of visual coding is a didactical concept from neuroscience: the homunculus (Figure 3.2). Here the three-dimensional surface of the human body is locally changed to reflect the density of sensorical and motorical neural density [Penfield and Rasmussen, 1950]. The idea of encoding importance in size however is not a novel idea. It dates back to examples from early renaissance (Figure 3.2, c) where a donor of a painting is depicted smaller, as he is less important than the depicted saint, or even earlier examples like medieval book illustrations (Figure 3.2, d). Considering that the metaphor is as old as art itself, e. g., used in Egypt (Figure 3.2, e), it seems worthwhile to make it accessible for digital media.

In this chapter we introduce an approach to locally deform a domain such as a two-dimensional image or three-dimensional surface such that it conveys importance. The computational challenge is to effectively minimize distortions, such as fold-overs, while fulfilling the goals imposed by the importance field to convey. Besides directly depicting importance, our approach can be used for focus+context visualization [Leung and Apperley, 1994], as a two- or three-dimensional editing metaphor, on its own or for the design of two- and three-dimensional aggregate texture design.

3.2 Approach

An overview of our approach is given in Figure 3.3. The input boundary (Section 3.2.1) is first voxelized (Section 3.2.2) and subsequently the importance is defined on the voxelization (Section 3.2.2). Next, a two-step optimization (Section 3.2.3) is performed to fulfill the importance goals, retain the object appearance, and smoothly avoid self-collisions. Finally, a deformation transfer (Section 3.2.5) from the voxelization back onto the boundary is performed.

3.2.1 Input

Input to our approach is a detailed $(d-1)$-dimensional boundary $\mathcal{B}$ in $\mathbb{R}^d$ and a scalar importance field $\mathcal{F}(\mathbf{x}) : \mathbb{R}^d \to \mathbb{R}^+$ defined on its voxelization. Output is a new boundary $\mathcal{B}'$, where the local size is proportional to the importance, with an intersection-free deformation. In practice we deal with a discrete d-dimensional polygonal mesh $B = (V^{\mathrm{B}}, E^{\mathrm{B}})$ defined by its vertex positions V^{B} and its edges E^{B}.

3.2.2 Voxelization

We perform a voxelization to become independent of the actual underlying boundary and its representation, to achieve a more uniform sampling, to become robust to potential (erroneous) self-intersections present in the input B, and to better reflect the solid nature of the shape inside the boundary B.

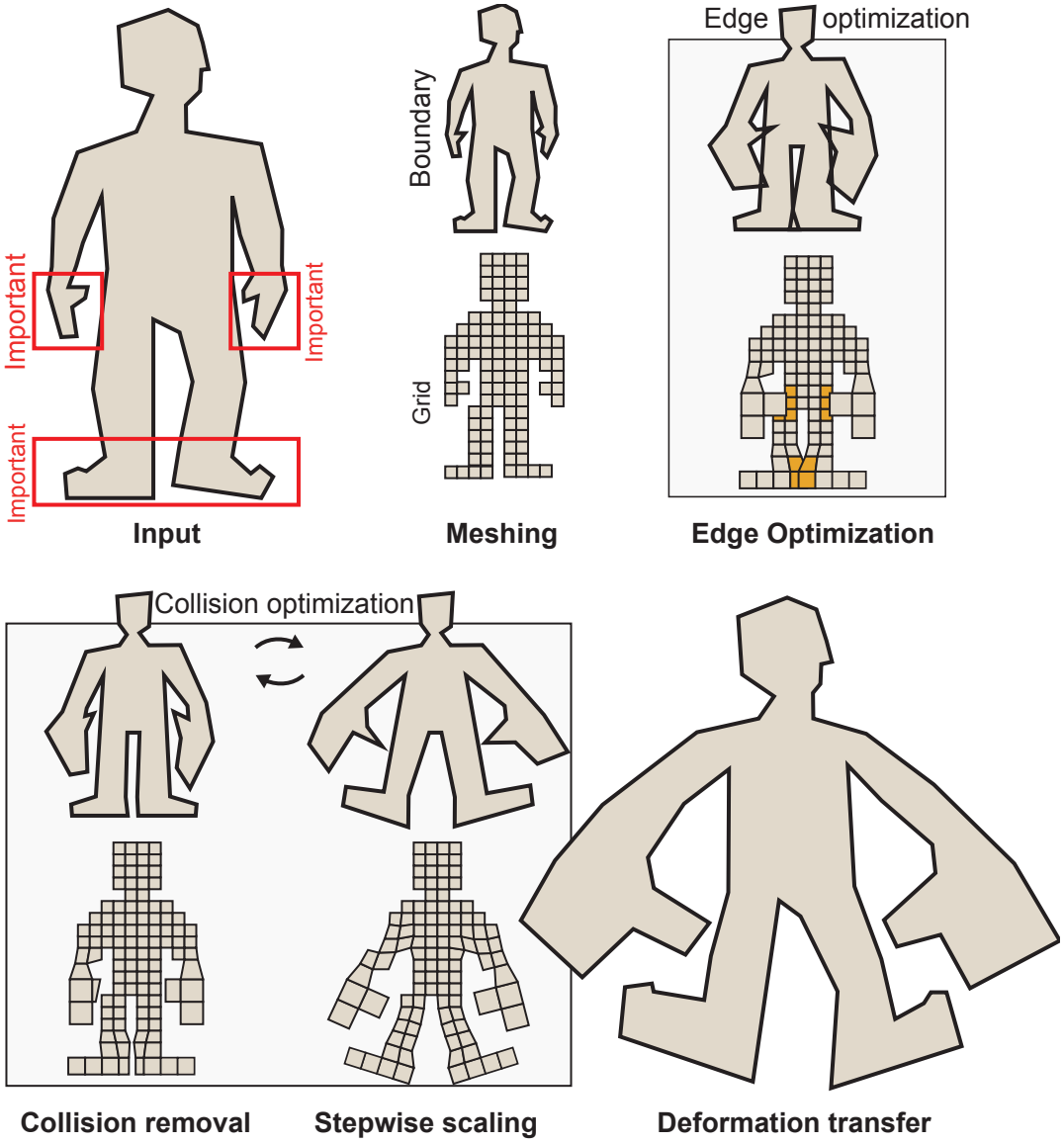

Figure 3.3: Overview of our approach *(Left to right, top to bottom)*. Input is a detailed boundary representation such as a three-dimensional mesh with an annotation of importance. First, the boundary is simplified and voxelized into a regular grid. For didactical purposes the optimization is illustrated both for the the boundary *(top)* and the regular grid actually used in our computation *(bottom)*. The deformation consists of an edge and a collision optimization. The edge part first locally scales the elements proportional to their importance and distributes the distortion by preserving relative coordinates. While the resulting shape is smooth and conveys the importance well, it results in collisions *(yellow cells)*. In the collision part, collisions are removed, which re-introduces distortions. Those distortions are removed by locally preserving edge lengths. This can result in new collisions; hence the collision optimization step is iterated. Finally, the deformation of the grid is propagated onto the original input mesh. Note, how the resulting shape follows the prescribed importance, has no collisions, smoothly distributes collision response over the complex, and has rotated parts *(cf. e. g., the hands)* to avoid collisions.

The first step is to voxelize B into an edge complex $C = (V, E)$ with vertices $V = \{v_1 \ldots, v_n \in \mathbb{R}^d\}$ and edges $E = \{e_1, \ldots, e_m \in \mathbb{N}^2\}$ as well as into a tetrahedral complex $C_t = (V, T)$ with the same vertices V and tertrahedra $T = \{t_1, \ldots, t_o \in \mathbb{N}^4\}$. For images the alpha channel is used to find all occupied pixels, i. e., pixels with an alpha value > 0. To this end, a finite discretization is chosen that partitions space into virtual cells. For each cell that is at least partially inside B, all its edges, its vertices, and its tetrahedra are appended to C and C_t respectively. The scalar field importance values $F = \{f_1, \ldots, f_m \in \mathbb{R}\}$ are assigned to each edge $e \in E$. $\Delta_i \in \mathbb{R}^d$ is defined to represent the i-th edge's difference vector, i. e., $\Delta_i = v_j - v_k$.

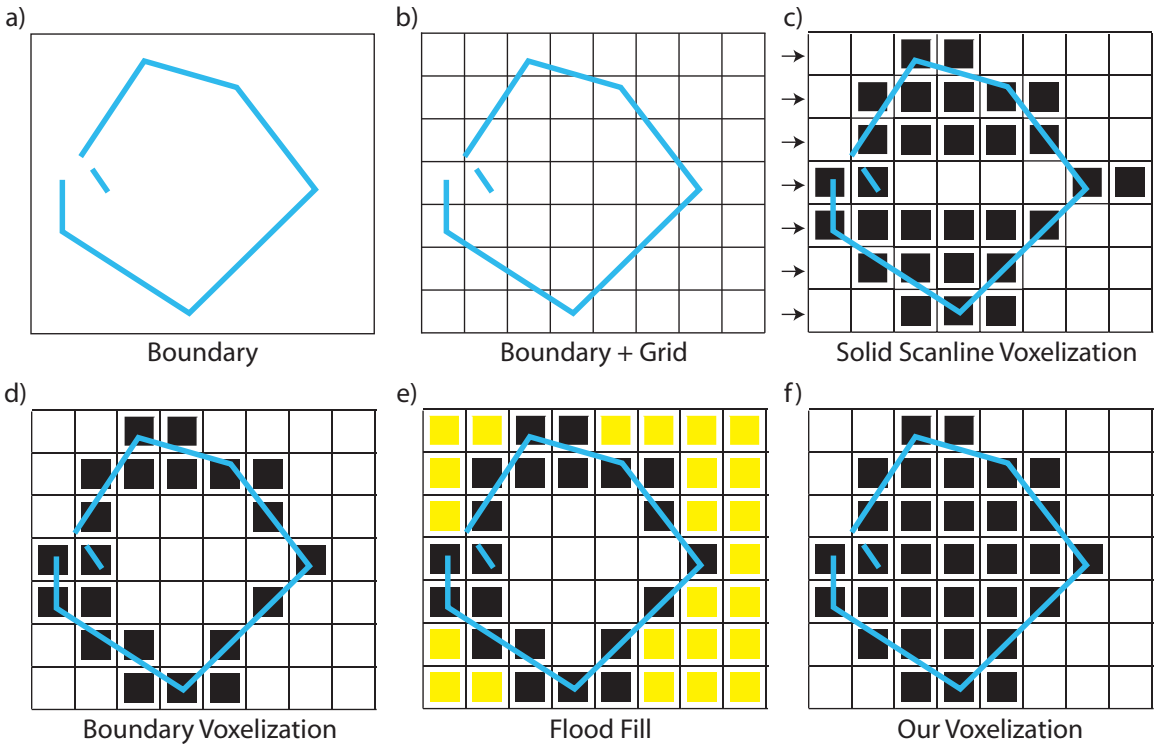

Figure 3.4: Voxelization. Note how our voxelization is more robust to meshing defects.

In practice, B cannot be assumed to be closed manifold as done for many (GPU) voxelizations [Dong et al., 2004]. Instead we perform solid voxelization that tolerates small holes (Figure 3.4, d – e). First, the boundary is voxelized into a grid, by simply rasterizing its triangles. Doing so all cells that are partially covered by B are marked and consequently holes smaller than one cell are closed. Holes larger than a cell, e. g., from non-orientable objects such as cloth lead to shell-like objects without volume. We did not further investigate deformation of such objects and report results restricted to surfaces with holes no larger than one cell. We assume that the cell size is small enough to resolve all details, e. g., to separate the hand and body in Figure 3.3. In a final step a flood fill finds all outside voxels and every voxel that is not marked as being outside is considered to belong to the inside.

3.2.3 Optimization

Our optimization is performed in two steps: The first step finds a mesh configuration that tries to preserve edge directions and set edge lengths according to importance (*edge optimization*). The second step finds a collision-free configuration of the first step's solution which evenly distributes the distortions introduced by collision response over the complex (*collision optimization*).

Edge optimization Here, we seek to find vertex positions V' such that edges in (V', E) have lengths proportional to their importance values F. In other words, for each edge $e_i \in E$ with importance $f_i \in F$ it holds that

$$\Delta_i' = f_i \Delta_i,$$

which leads to a linear system of equations. Please note how this formulation also tries to preserve initial edge directions. Since the importance values f_i are user-defined and thus might not correspond to an actual, viable configuration, in general not all edge lengths can be satisfied simultaneously. Hence the goal becomes to find the minimal sum of errors in a least squares sense, i. e.,

$$\underset{V'}{\operatorname{argmin}} \sum_{i=1}^{m} \|\Delta_i' - f_i \Delta_i\|_2^2. \tag{3.1}$$

The global minimum of this equation can be found by setting its directional derivatives to zero. This step of our pipeline generalizes the approach of Sorkine and Alexa [2007] by adding a per-edge scaling factor.

Collision optimization After the edge optimization, the edges in the complex (V', E) are scaled optimally while reducing distortions, but the complex (V', T) is potentially self-intersecting. A second optimization is used to resolve self-intersections while preserving the appearance of the previous step as much as possible. To this end, collisions need to be detected, resolved, and the resulting deformation needs to be distributed. To fulfill these goals at the same time, each individual goal is iteratively resolved in turn as in Müller et al. [2007]. While the edge optimization strived to preserve edge directions the collision optimization should just maintain edge lengths. Doing so distributes the collision response over the complex since it allows edges to rotate.

Since collision detection is a continuous process only small scalings can be established concurrently; introducing the full scaling of Equation 3.1 at once would lead to many severe collisions which are much harder to resolve. As a solution we propose to introduce only a small portion of the optimal length $l_i = \|\Delta_i'\|_2$ for the i-th edge in multiple iteration steps, i. e., the target length for each edge e_i becomes λl_i with incrementally increasing $\lambda \in [0, 1]$. Doing so will result in much fewer collisions in every step that are much easier to resolve.

For the stepwise scaling of the mesh we solve the equation

$$\|\Delta_i''\|_2 = \lambda l_i$$

for every i. Since the values l_i are derived from a satisfiable configuration of the mesh the resulting system of equations has at least one solution (namely V'). However V'' should be collision-free which introduces distortions to the mesh and turns the system into a potentially overdetermined system of equations that has to be minimized, i. e.,

$$\underset{V''}{\operatorname{argmin}} \sum_{i=1}^{m} \left(\|\Delta_i''\|_2 - \lambda\, l_i \right)^2 \tag{3.2}$$

such that V'' is free of collisions. Since this system involves finding a constrained solution V'' it has to be minimized iteratively. Solving this equation has a two-fold purpose: First it incrementally scales the complex to the designated size and second it distributes the distortions introduced by collision response over the complex.

For higher similarity to the input complex, collisions should not be resolved precisely but rather vertices should keep a certain distance to the boundary. We propose to approximate this behavior by defining for every vertex i a sphere with a radius $r_i \in \mathbb{R}^+$ that equals the minimal distance to every adjacent vertex, i. e., $r_i = \min\{\|v_i'' - v_j''\|_2 | (i,j) \in E\}$. Collision between all vertices and spheres are found using spatial hashing build in every iteration [Teschner et al., 2003]. When a collision of a vertex v_i'' with a sphere surrounding vertex v_j'' is found, the vertex is projected to the shell of the sphere in the direction $v_i'' - v_j''$, i. e., the vertex is corrected by

$$v_i'' += \left(\frac{r_i}{\|v_i'' - v_j''\|_2} - 1 \right) \frac{v_i'' - v_j''}{2}.$$

The same correction is performed for vertex v_j in the opposite direction.

3.2.4 Equation minimization

We minimize Equation 3.1 and Equation 3.2 based on parallel operations (using OpenCL) as follows. Equation 3.1 leads to a linear system $\mathsf{A}\,\mathbf{x} = \mathbf{b}$, where the rows a_i of A express the differences between vertex i and its edge-connected vertices; the values b_i of $\mathbf{b}$ reflect the sum of the signed distances for every coordinate between vertex i and its edge-connected vertices. The matrix A therefore is sparse, symmetric, and positive semi-definite, which allows to use efficient solvers such as the Cholesky decomposition or the Gauss-Seidel method. Although the Cholesky decomposition can solve the system in one step, the iterative approach of the Gauss-Seidel method has some advantages for interactive applications: On the one hand, the system is underdetermined and has three degrees of freedom, i. e., the vertices can be translated to an arbitrary position in space. Using the Cholesky decomposition the complex can freely translate, while the Gauss-Seidel method converges to a solution spatially close to the initial position. On the other hand, the Gauss-Seidel method can efficiently be implemented on parallel hardware which allows for real-time framerates even for very detailed complexes. Moreover, the results of intermediate steps can immediately be visualized which turns the deformation into an animation providing an extra level of information to the user. Because of its iterative nature the Gauss-Seidel method allows for instantaneous changes of the complex' structure, while a Cholesky decomposition would fix the complex and requires additional preprocessing overhead.

Minimizing Equation 3.2 involves solving a constrained system of equations of second degree polynomials. We solve this iteratively using constraint projection as described by Müller et al. [2007], which allows to treat edges independently and admits to handle scaling and collision response in a unified way. Again, since the system of equations only specifies edge lengths, it is underdetermined. This approach is compatible with the solution to Equation 3.1.

Since both our solvers work iteratively they turn the solutions V' and V'' into sequences $(V'_0,\dots,V'_{n_\mathrm{e}})$ and $(V''_0,\dots,V''_{n_\mathrm{c}})$ with $V'_0 = V''_0 = V$, $n_\mathrm{e} \in \mathbb{N}^+$ and $n_\mathrm{c} \in \mathbb{N}^+$ as the number of iterations for Equation 3.1 and Equation 3.2, respectively. We combine both sequences into a single sequence $V^{(i)} : \mathbb{N} \to \left(\mathbb{R}^d\right)^n = (V'_0,\dots,V'_{n_\mathrm{e}},V''_0,\dots,V''_{n_\mathrm{c}})$. The pseudo-code of our approach is given in Figure 3.5. After a user interaction, the last solution V'' is used as the initial guess V_0.

```
V^(0) := V;
// Edge optimization
for t from 0 to n_e − 1
  for i from 0 to n in parallel
    v_i^(t+1) := (b_i + a_ii v_i^(t) − (A v^(t))_i)/a_ii;
// Remember optimal edge length
for all edges e = (i, j)
  l_i := ‖Δ'_i^(n_e)‖_2;
V^(n_e) := V;
// Collision optimization
for t from n_e to n_e + n_c − 1
  λ := (t − n_e)/(n_c − 1);
  // Stepwise scaling
  P := 0;
  for i from 0 to n in parallel
    p_i := v_i^(t);
    for all edges e = (i, j) connected to v_i
      p += ((λ l_i/‖p_i − v_j^(t)‖_2) − 1)(p_i − v_j^(t))/2;
  // Calculate radii
  R^(t) := FLT_MAX;
  for i from 0 to n in parallel
    for all edges e = (i, j) connected to v_i
      r_i^(t) := min(r_i^(t), ‖p_i − p_j‖_2);
  // Handle collisions
  for i from 0 to n in parallel
    for all v_j with ‖p_i − p_j‖_2 < r_i
      p += ((r_i^(t)/‖p_i − p_j‖_2) − 1)(p_i − p_j)/2;
    v'_i^(t+1) := p;
```

Figure 3.5: Pseudo-code of our approach.

Theoretically the increment in edge lengths must not be larger than the minimum distance between two vertices divided by the maximum distance between two vertices of the complex. However this value highly depends on the scalar field F and in practice we empirically found that much larger values for λ are feasible.

3.2.5 Deformation transfer

In a pre-process, the barycentric coordinates $\alpha_i, \beta_i, \gamma_i, \delta_i \in \mathbb{R}$ with

$$\alpha_i + \beta_i + \gamma_i + \delta_i = 1$$

for the containing tetrahedron

$$t_j = (a_j, b_j, c_j, d_j) \in \mathbb{N}^4$$

defined by the vertices v_{a_j}, v_{b_j}, v_{c_j}, and v_{d_j} are computed for every vertex v_i^{B} in B [Müller and Gross, 2004]. Finally, the deformed vertex positions V'' are used to deform the original boundary B. At runtime, the new position is compute as

$$v_i^{\mathrm{B}'} = \alpha_i v''_{a_j} + \beta_i v''_{b_j} + \gamma_i v''_{c_j} + \delta_i v''_{d_j}.$$

Using the barycentric coordinates the scalar field F can also be resampled on the actual boundary B to visualize the importance on it. In practice sometimes higher order upsampling is used [Zollhöfer et al., 2012], but we found the results to be satisfactory.

3.3 Results

Typical outputs of our approach for and for two-dimensional images are shown in Figure 3.6 and for three-dimensional surfaces in Figure 3.7. All results can be manipulated interactively and the solution is found incrementally. A reasonable linear deformation feedback is returned in a fraction of a second before the solution converges in less than a second. Typically, the iterative approach for the edge optimization converges in less than $n_{\mathrm{e}} = 20$ iterations. The collision optimization converges in the order of a few minutes (typically, we set $n_{\mathrm{c}} = 1000$ iterations). The high-frequency (and by that also detailed) importance fields require a high number of voxels, typically $128 \times 128 \times 128$.

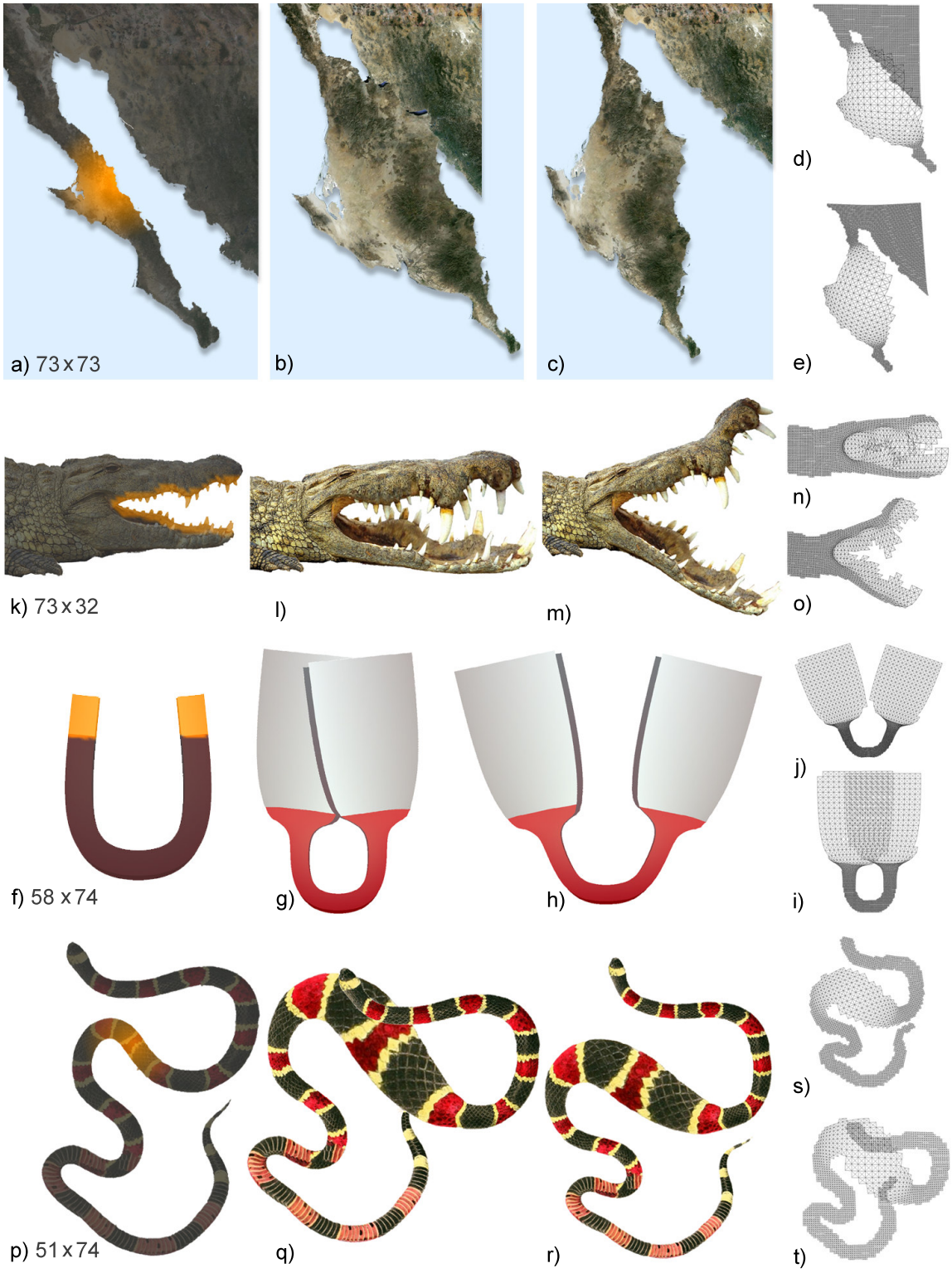

Figure 3.6: Our approach applied to two-dimensional images. For every example the resolution of the discretization is listed. *a)*: Deformation for geographical map data with importance where classic deformation leads to intersections *(b,d)* avoided by our approach *(c,e)*. *k)*: High-frequency deformation on the crocodile scales only the teeth, leading to many collisions *(l,n)*. Our method results in opening the mouth *(m,o)*. As only the teeth grow, the jaw globally and locally bends to make more room for the individual teeth. *f)*: Conveying magnetism on the end of the horseshoe magnet, results in interference *(g,i)* whereas our approach bends the domain to make room *(h,j)*. *p)*: A snake digesting its prey self-intersects when applying classic deformation *(q,t)* that is prevent by our approach *(r,s)*.

Figure 3.7: Results of classic deformation *(Left)* compared to our collision-aware approach *(Right)*. For every example the resolution of the voxelization is listed and a false-color coded importance field is shown. *a)*: Noise sources of an airplane. *b)*: Injury probability of a goal keeper *c)*: Black market price of body parts of an elephant. *d)*: Force field on a robot. *e)*: Pollen concentration of flowers. *f)*: Homunculus. *g)*: Heat field on a street lamp. *h)*: Diluent concentration of bike parts in contact with human body. *i)*: Focus+context visualization of a colon model (cf. [Wang, Lee and Tai, 2008]). *j)*: Comic character deformation.

Chapter 4

Interactive By-example Design of Artistic Packing Layouts

Figure 4.1: Starting from a common layout *(top left)*, the user's objective is inferred from placement of three primitives *(push pins)*, leading to a layout organized vertically by size *(top right)* and after a different placement additionally by brightness horizontally *(bottom)*.

4.1 Introduction

Arranging sets of primitives into a pleasing spatial packing layout that tightly fills the space in two-dimensions is tedious and requires expert skills (Figure 4.2).

While arrangements can serve for recreation and aesthetic purposes, they often seek to convey an underlying message concerning the relation between primitives and serve a didactical purpose. In this work, we propose a system to automate artistic layouts by inferring the user's high-level intentions from the interactions performed. The interactive exploration of different artistic layouts and primitive relations enabled by our system goes beyond static print or display layouts and helps to improve general layouts, such as required for Mind Maps [Buzan, 1991], tag clouds [Bateman, Gutwin and Nacenta, 2008], or any arrangement of graphical, two-dimensional primitives.

Figure 4.2: Packing examples from art. *a):* G. Grohmann: "Recueil de dessins" (1805). *b):* Bulliard: "La Flore Des Environs de Paris" (1776). *c):* Schweizerbart: "Evolution der Tiere" (2001). *d):* U. Gorter: "Whales of the World" (2003). *e):* J. Brickwil: "Natural history of North-Carolina" (1712).

Figure 4.1 shows three steps of a typical interaction using our system: After loading a set of primitives, our system presents a general-purpose layout (Figure 4.1, *left*). To change

this layout, a simple solution would be to expose many sliders that control the importance weight for each feature dimension. Such high-dimensional parameter spaces are hard to navigate for colloquial users and hamper creative exploration. Our system takes a different approach: We offer the user to move primitives to new positions (Figure 4.1, *middle*) and by that to infer the user's intention, leading to a new layout, in this case, where primitives are organized vertically by size. After a second manipulation (Figure 4.1, *right*) the layout is organized by brightness horizontally and by size vertically.

4.2 Overview

Conceptually, our system consists of an infinite loop: First, a *forward* layout step places primitives according to some rules (Section 4.3). When user interaction occurs, an *inverse* layout step (Section 4.4) refines the rules for the forward layout and the loop repeats.

A typical use case of the system is as follows (Figure 4.1): Initially, the user is presented some generic layout of graphical primitives in two dimensions. This layout maps primitives with certain similar features (e. g., brightness, shape, etc.) to similar locations (Section 4.3.1). Primitives are placed in such a way that the average distance of the boundary of nearby primitives (the "gap" between them) has a similar value everywhere (Section 4.3.2). Next, users interactively manipulate this layout by constraining a small number of primitives to particular locations (Section 4.4). This is depicted by a push-pin icon shown next to the constrained primitive. The system infers what features are to be used in the forward layout from these constraints.

4.3 Forward layout

Primitives are $n \in \mathbb{N}$ objects represented as images with a (possibly concave) boundary Ω_i. A typical number of primitives in our examples is between 10 and 200. An additional input to our approach is a per-primitive feature vector $\mathbf{f} \in \mathbb{R}^m$ representing $m \in \mathbb{N}$ distinct features. Features capture visual properties, such as size, shape, brightness, texture, etc. (automatically extracted from the input primitive by image processing) as well as semantic quantities like age, strength, etc. (acquired from an external database).

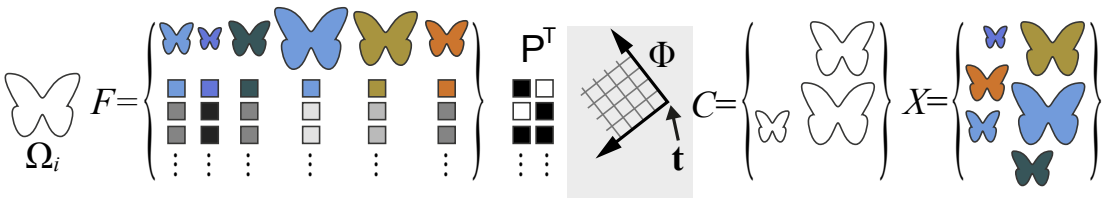

Figure 4.3: Notations of our formalization. Shown is the boundary Ω_i, the sequence of feature vectors F, the feature projection matrix P, the parameter translation vector $\mathbf{t}$, the set of constraints C (cf. Section 4.4), and the final output sequence X.

The sequence of the feature vectors of all primitives is denoted as $F = \{\mathbf{f}_1, \ldots, \mathbf{f}_n \in \mathbb{R}^m\}$. A typical number of features in our examples is 10. Output of our system is a sequence $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n \in \mathbb{R}^2\}$ which contains locations at which primitives are to be placed in the two-dimensional layout space $\mathbb{R}^2$ (Figure 4.3).

Forward layout is performed in two main steps to be explained next: *feature mapping* (Section 4.3.1) and *primitive distribution with spatial extent* (Section 4.3.2).

4.3.1 Feature mapping

Feature mapping reduces high-dimensional features $\mathbf{f} \in \mathbb{R}^m$ to their low-dimensional 2D layout coordinates $\mathbf{x} \in \mathbb{R}^2$ as

$$\mathbf{x} = \phi(\mathsf{P}\mathbf{f} + \mathbf{t}), \tag{4.1}$$

where P is a *feature projection* matrix, $\mathbf{t}$ is a *parameter translation*, and ϕ is a *layout function*, all explained in the next paragraphs.

Feature projection and parameter translation is performed by a tuple $(\mathsf{P}, \mathbf{t})$ as a projection matrix $\mathsf{P} \in \mathbb{R}^{2\times m}$ and a translation vector $\mathbf{t} \in \mathbb{R}^2$ that map feature vectors $\mathbf{f} \in \mathbb{R}^m$ to parameter vectors $\mathbf{p} \in \mathbb{R}^2$ as $\mathbf{p} = \mathsf{P}\mathbf{f} + \mathbf{t}$. P is non-zero only at position (k,l) if feature l is mapped to dimension k. The value at $\mathsf{P}_{k,l}$ gives the factor by which the feature is scaled to create the parameter vector dimension. Per dimension (i. e., row) only one non-zero element (i. e., feature) scaling-factor is present, i. e., only one feature is used per parameter vector dimension. As an example given three features (e. g., size, brightness, anisotropy) the matrix

$$\mathsf{P} = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 0.2 & 0 \end{bmatrix}$$

would select the third feature (anisotropy) with unit scaling as the first dimension and the second feature (brightness) as the second dimension, scaled by 0.2. $\mathbf{t}$ is used to shift the parameter vector along the parameter axis, and can be used for example to move the primitives along the axes in layout space in a Cartesian layout (Figure 4.4).

Layout function The parameter vector $\mathbf{p}$ serves as input to different layout functions $\phi(\mathbf{p}) \in \mathbb{R}^2 \to \mathbb{R}^2$. Such functions map e. g., the first parameter to the x-axis and the second one to the y-axis in a Cartesian layout, or the first parameter to angle and the second to radius in a radial layout function. In practice different layout functions can be used (Figure 4.4). The only requirement for ϕ is that the inverse mapping ϕ^{-1} needs to exist in order to perform the inverse layout (Section 4.4).

Figure 4.4: Isolines of first parameter dimension for different layout functions ϕ.

Single-dimensional case We also support to select only one, or no feature at all, i. e., where P does not have full rank with rows of only zeros. In this case, the missing dimensions in $\mathbf{p}$ are created using Multi-dimensional scaling (MDS) [Cox and Cox, 2008] of all remaining features, i. e., the features with columns that have all zeros in P. The resulting parameter vector $\mathbf{p}$ can then be fed into ϕ as before.

4.3.2 Primitive distribution with spatial extent

Preserving a balanced distance to all adjacent primitives is key to a good layout. The output layout of the feature mapping however can produce arbitrary primitive positions with possibly overlapping primitives that do not necessarily occupy the given layout space evenly. Further, feature mapping only operates on points and has no concept of spatial extent. To distribute primitives with spatial extent we equalize the distances between the boundaries of nearby objects. For primitives of complex shape and varying size, this leads to more pleasant distributions, yet resulting in simple computations that allow for a real-time, GPU implementation.

Boundary Voronoi Tessellation For point primitives, *Centroidal Voronoi Tessellation* (CVT) [Lloyd, 1982] has proven to produce layouts that yield balanced point distances. For our purpose, one option would be to use the extension of Hiller et al. [2003] for general shapes. We implemented this approach but observed unsatisfying results: All but almost ellipsoidal shapes produce unbalanced results that drift (cf. Figure 4.5). This drift arises from that fact that the proposed CVT generalization not explicitly states boundary distances in its objective function but aligns the centroids of the primitive and its Voronoi region resulting in an optimization that rather resembles shape matching [Müller et al., 2005] between the Voronoi region and primitive (Figure 4.7, b).

To achieve an even distance between primitives, we explicitly include the boundary distances in our objective function (Figure 4.7, a). The deviation of a layout X from this equilibrium can be measured by summing the squared distances between each primitive's boundary and its Voronoi region boundary:

$$c(X) = \sum_{i=1}^{n} \int_{\Omega_i^{\mathrm{V}}} \mathrm{distance}_i(\omega, \mathbf{x}_i)^2 \mathrm{d}\omega, \tag{4.2}$$

where Ω_i^{V} is the boundary of the i-th primitive's Voronoi region and $\mathrm{distance}_i(\omega, \mathbf{x}_i) : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ gives the shortest Euclidean distance between ω, a point on the Voronoi region boundary, and the i-th primitive's boundary Ω_i positioned at $\mathbf{x}_i$. The minimum of this cost function $X' = \mathrm{argmin}_X\, c(X)$ yields the optimal solution. This formulation is very similar to Equation 1 in Dalal et al. [2006], except for that only considering the boundary of the Voronoi region in our formulation and not its interior (we also omit the rotational parts as we explicitly are only interested in a translation). In practice, we perform all calculations

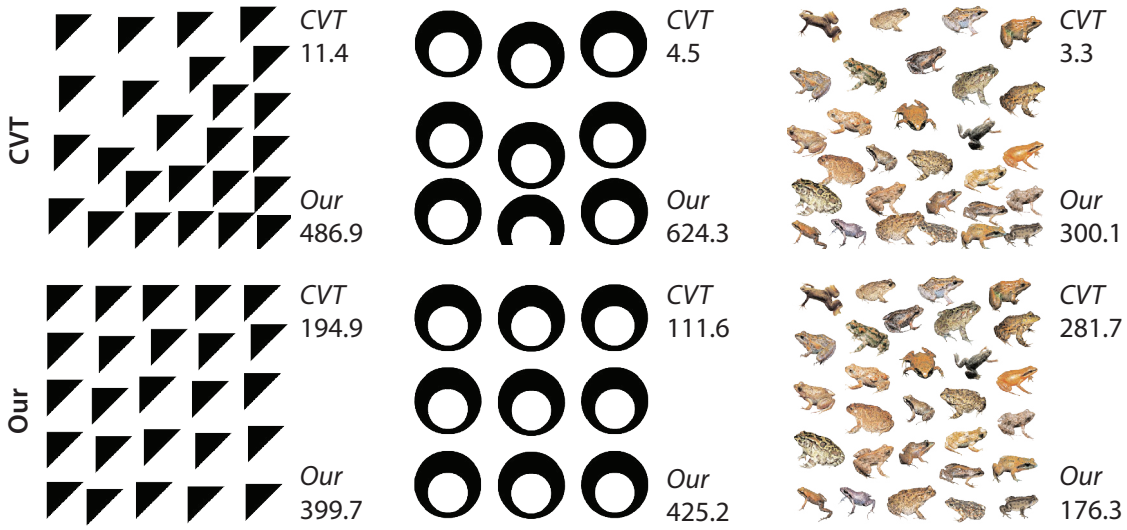

Figure 4.5: Results for extended CVT [Hiller, Hellwig and Deussen, 2003] *(upper row)* and our relaxation *(lower row)*. Adjacent to the layouts the values of the CVT residual and our residual (Equation 4.3) is listed. CVT relaxation results in unbalanced layouts and its residual in each column is lower for CVT relaxation. In contrast our residual is lower for the more balanced layouts, indicating that our residual is more accurate in measuring the quality of a balanced layout.

on a discretized grid, i. e., Equation 4.2 becomes

$$c(X) = \sum_{i=1}^{n} \sum_{\omega \in \Omega_i^{\mathrm{V}}} \mathrm{distance}_i(\omega, \mathbf{x}_i)^2. \tag{4.3}$$

As minimizing Equation 4.3 is NP-hard, finding the global optimum is infeasible. Moreover, we are explicitly not interested in the global optimum of Equation 4.3 as it possibly shuffles all primitive positions to new places, whereas we seek to find a solution that is similar to the one produced by the forward mapping (Section 4.3.1), but balances primitive distances. Hence the global optimization of Equation 4.3 is replaced by a local iterative one that tries to find a small offset for each primitive position individually, given a static Voronoi diagram per iteration. The formula then becomes

$$c_i(\mathbf{x}_i) = \sum_{\omega \in \Omega_i^{\mathrm{V}}} \mathrm{distance}_i(\omega, \mathbf{x}_i)^2. \tag{4.4}$$

This equation could be minimized using image correlation, as done by Dalal et al. [2006], whose complexity is $O(na \log a)$, where n is the number of primitives and a is the total number of pixels of the domain. However, this complexity is prohibitively expensive for our real-time needs. Hence we seek to replaced Equation 4.4 with an approximation.

A first idea could be to replace $\mathrm{distance}_i(\omega, x_i)$ by a Taylor polynomial [Pottmann and Hofer, 2003] of degree one and solve this approximate objective $\hat{c}_i$ instead of Equation 4.4. However c_i is only poorly approximated by $\hat{c}_i$ as the distance between c_i and $\hat{c}_i$ can become

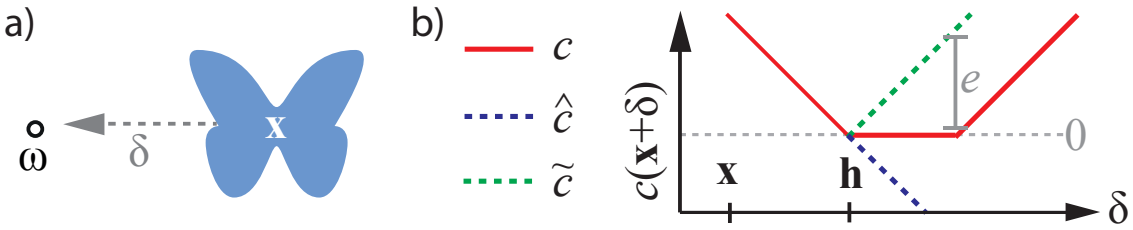

Figure 4.6: *a):* In this 1D example the primitive at $\mathbf{x}$ is shifted by δ. The distance function c and its approximations $\hat{c}$ and $\tilde{c}$ measure the shortest distance to ω. *b):* The distance functions are plotted for an approximation at $\mathbf{x}$. For all values $\delta \leq \mathbf{h}$ the approximations $\hat{c}$ and $\tilde{c}$ exactly conform with c. For values $\delta > \mathbf{h}$ both functions deviate from the correct function, but the error of $\tilde{c}$ is bounded by e.

arbitrarily large (cf. Figure 4.6). Furthermore Taylor polynomials require derivatives, whereas $\text{distance}_i(\omega, x_i)$ is not necessarily continuously differentiable.

We can reformulate Equation 4.4 by expanding distance_i as

$$c_i(\mathbf{x}_i) = \sum_{\omega \in \Omega_i^{\mathrm{V}}} \|\text{closest}_i(\omega, \mathbf{x}_i) - \omega\|_2^2$$

where $\text{closest}_i(\omega, \mathbf{x}_i) : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}^2$ gives the point closest to ω on the boundary Ω_i of the i-th primitive at position $\mathbf{x}_i$. The function $\text{closest}_i(\omega, \mathbf{x}_i)$ can be approximated for a point $\mathbf{x}_i + \delta_i$ as

$$\text{closest}_i(\omega, \mathbf{x}_i + \delta_i) \approx \text{closest}_i(\omega, \mathbf{x}_i) + \delta_i, \tag{4.5}$$

i. e., the closest position to ω on Ω_i at position $\mathbf{x}_i + \delta_i$ is approximately the closest position on Ω_i at position $\mathbf{x}_i$ with an added offset δ_i. This approximation does not involve a derivative and always yields a point on the primitive's boundary. The maximal error is therefore bounded by the maximal distance of two points on the primitive's boundary (cf. Figure 4.6). Using Equation 4.5 the cost for an offset δ_i is given for the primitive positions $\mathbf{x}_i$ as

$$\tilde{c}_i(\mathbf{x}_i + \delta_i) = \sum_{\omega \in \Omega_i^{\mathrm{V}}} \|\text{closest}_i(\omega, \mathbf{x}_i) + \delta_i - \omega\|_2^2 .$$

The optimal δ_i' is found by setting its derivative to zero as

$$\delta_i' = \operatorname*{argmin}_{\delta_i} \tilde{c}_i(\mathbf{x}_i + \delta_i) = \frac{1}{|\Omega_i^{\mathrm{V}}|} \sum_{\omega \in \Omega_i^{\mathrm{V}}} \omega - \text{closest}_i(\omega, \mathbf{x}_i).$$

As an intuition behind this solution, finding the minimum can be regarded as a set of springs located at the Voronoi region boundary that tries to push or pull the primitive into the correct place in its Voronoi region. A single step in our iteration has a complexity of $O(n|\Omega^{\mathrm{V}}|)$ where $|\Omega^{\mathrm{V}}|$ is the total length in pixel of all Voronoi region boundaries.

For the special case of point primitives CVT is equivalent with our problem statement in Equation 4.3 (except for only considering the Voronoi boundary). The Voronoi diagram for overlapping primitives with spatial extend is not well-defined. To overcome this, we

compute the Voronoi diagram using a two-sided distance to the boundaries of the primitives, such that distances increase inside and outside of the boundary, leading to well-defined Voronoi diagrams. The primitives might have interior boundaries (e. g., the circular cutout of the circles in Figure 4.5, b) that should not influence the relaxation. By iterating over the Voronoi boundary the closest point on the primitive boundary by definition is on the outermost boundary. Due to our approximations, in rare cases the converged distributions might still have overlapping primitives.

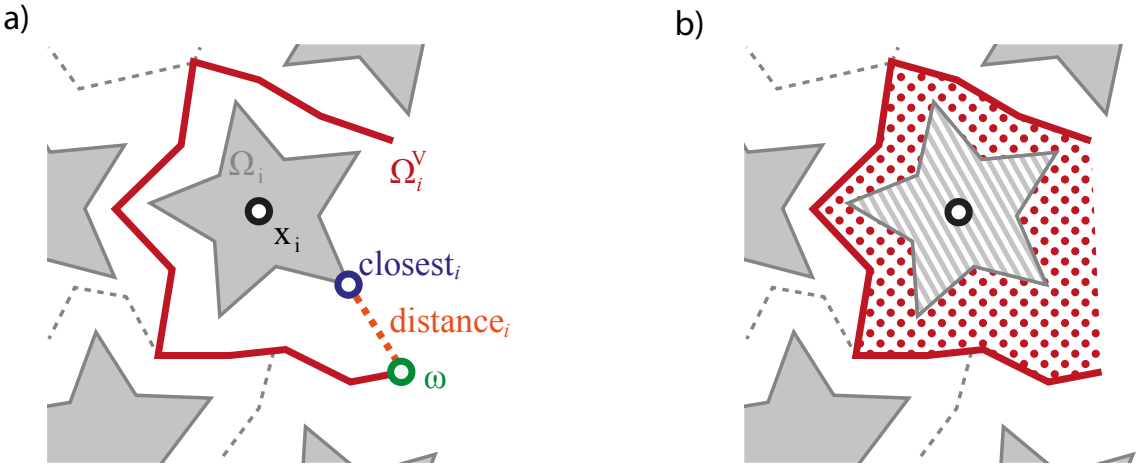

Figure 4.7: Relaxation approaches: *a:)* our new relaxation *(Please see text.)*, *b:)* classical shape matching relaxation, moving the object's *(grey, striped area)* center of mass to its Voronoi region's *(red, dotted area)* center of mass

Implementation We use a GPU to accelerate our relaxation. Two maps are pre-calculated for each primitive: The first one contains the distance of every pixel to the primitive boundary, the latter holds the location of the closest position on the boundary. At runtime we use rasterization [Hoff et al., 1999] in combination with the distance map to create a Voronoi diagram holding the index of the closest primitive at each pixel and an additional map giving the closest position on the primitive's boundary. The calculation of δ_i then becomes a parallel summation over all Voronoi region boundary pixels with a single diagram lookup per pixel to acquire the closest point on the primitive's boundary. The resulting relaxation is at least as efficient as CVT: Typically, we use 30 relaxations in every frame and achieve more than 10 fps for more than 200 primitives including drawing in HD resolution on an Nvidia GTX 680 GPU.

Extensions Arbitrary global boundaries, such as the shape of the butterfly in Figure 4.9, can be handled by removing the pixels of the Voronoi regions that are outside of the global boundary. Parts of the primitives might fall outside of the global boundary. Therefore for each pixel of primitive i outside of the global boundary an offset is added to δ_i in the direction of the closest point on the global boundary, pushing the primitive back inside.

4.4 Inverse Layout

The user can define primitive constraints, i. e., force certain primitives with indices $C = \{c_1, \ldots, c_o \in (1,n)\}$ with $o \in \mathbb{N}^+$ to be located exactly at position $\mathbf{x}_{c_i}$. The inverse layout computes a new feature projection matrix P, a parameter translation vector $\mathbf{t}$, and a new layout function ϕ (as defined in Section 4.3.1) that best "explain" the placement of the o constrained primitives, i. e., given the primitive positions X, their features F, and constraints C we try to find ϕ and $(\mathsf{P}, \mathbf{t})$ (cf. Equation 4.1) minimizing

$$\sum_{a \in C} \|\mathsf{P}\mathbf{f}_a + \mathbf{t} - \phi^{-1}(\mathbf{x}_a)\|. \tag{4.6}$$

Before any user interaction, P and $\mathbf{t}$ are set to zero and ϕ to identity. After a user changed the constraints we try to minimize Equation 4.6. Solving this task is split into two parts: enumeration of all plausible *layout functions* ψ and computation of the optimal *feature mapping* for it. Finally, we choose the layout hypothesis and feature mapping for which a residual function $r(\psi)$ is minimal (*layout selection*). We will explain both steps in the following paragraphs.

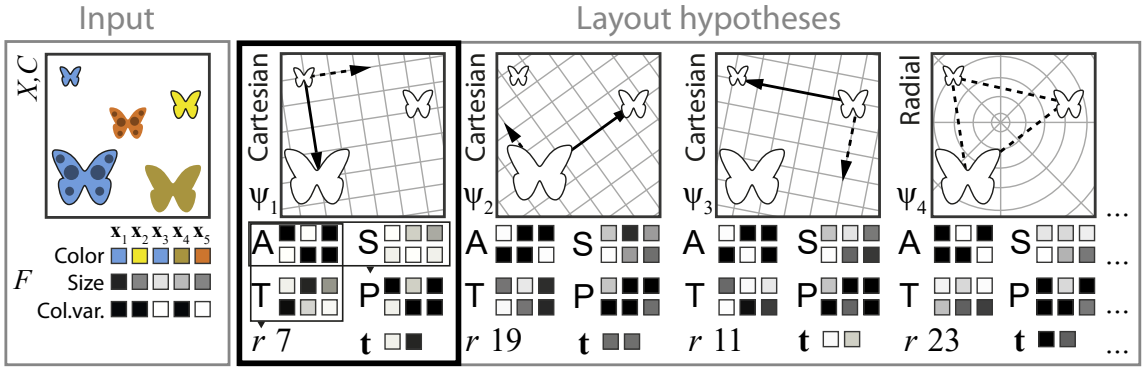

Figure 4.8: Inverse layout. *Input* are the primitives and their positions X, constraints C, and features F. From C several *layout hypotheses* ψ_i are build and the best feature-dimension mapping A and its residual r are calculated. Output is the ψ_i with the lowest cost, its projection matrix P, and the translation vector $\mathbf{t}$ *(ψ_1, black border)*.

Layout functions We restrict our enumeration to a plausible subset of all possible layout functions (the bijections on $\mathbb{R}^2$), which we call *layout hypotheses*. The Cartesian and radial layout hypotheses are built independently as follows.

A *Cartesian* layout function hypothesis is defined by two axes. We assume one axis is the connection between two constraint positions, i. e., for each $a, b \in C$ with $a \neq b$ the first axis is

$$\mathbf{u} = \frac{\mathbf{x}_a - \mathbf{x}_b}{\|\mathbf{x}_a - \mathbf{x}_b\|_2}.$$

The second axis is then chosen orthogonal, i. e., $\mathbf{v} = (-\mathbf{u}_y, \mathbf{u}_x)^\mathsf{T}$ and the layout function becomes

$$\psi(\mathbf{p}) = (<\mathbf{u}, \mathbf{p}>, <\mathbf{v}, \mathbf{p}>)^\mathsf{T},$$

resulting in $o(o-1)/2$ layout hypotheses.

A *radial* layout function hypothesis is defined by a center position $\mathbf{u}$. Our hypothesis assumes that the center position is either located at one of the constraints, i. e., $\mathbf{u} = \mathbf{x}_a$ with $a \in C$ or that its position is the centroid, i. e.,

$$\mathbf{u} = \frac{1}{o} \sum_{a \in C} \mathbf{x}_a,$$

or the center of the bounding box of all constraints, i. e.,

$$\mathbf{u} = \frac{1}{2} \left(\min_{a \in C} x_a + \max_{a \in C} x_a \right).$$

The layout function then is

$$\psi(\mathbf{p}) = (\mathbf{u}_1 + \mathbf{p}_1 \sin \mathbf{p}_2, \mathbf{u}_2 + \mathbf{p}_1 \cos \mathbf{p}_2)^\mathsf{T},$$

resulting in $o+2$ different layout hypotheses.

Feature mapping For a fixed layout hypothesis ψ its residual r and its optimal feature projection matrix P are found as follows: All constraint positions are mapped back to parameter space as $\mathbf{p}_i = \psi^{-1}(\mathbf{x}_i)$, i. e., the inverse of the layout function. Next, we need to find how well a common scalar $\mathsf{S}_{k,l}$ of the k-th dimension-wise difference of the parameter vector could "explain" the differences of the l-th feature. For example, we would like to know how well a common scaling of all butterfly size differences can "explain" the differences of the second layout parameter, which, again, could be the radii of a radial layout or the vertical coordinates in a Cartesian layout. We combine the cost of "explaining" parameter dimension k by feature l using the scaling matrix $\mathsf{S}' \in \mathbb{R}^{2\times m}$ as the residual matrix functional $\mathsf{R} \in \mathbb{R}^{2\times m} \to \mathbb{R}^{2\times m}$ defined as

$$\mathsf{R}_{k,l}(\mathsf{S}') = \sum_{a,b \in C, a \neq b} \left(\mathsf{S}'_{k,l} (\mathbf{f}_a - \mathbf{f}_b)_l - (\mathbf{p}_a - \mathbf{p}_b)_k \right)^2.$$

For each dimension k and feature l the optimal scaling factor is found by setting its derivatives to zero as

$$\mathsf{S}_{k,l} = \operatorname*{argmin}_{\mathsf{S}'} \mathsf{R}_{k,l}(\mathsf{S}') = \frac{\sum_{a,b \in C, a \neq b} (\mathbf{p}_a - \mathbf{p}_b)_k (\mathbf{f}_a - \mathbf{f}_b)_l}{\sum_{a,b \in C, a \neq b} (\mathbf{f}_a - \mathbf{f}_b)_l^2}.$$

Using these scaling factors yields the minimal residual matrix $\mathsf{T} = \mathsf{R}(\mathsf{S})$. Now the assignment of features to dimensions can be found as a solution to the general assignment problem [Martello and Toth, 1987], such that every dimension is assigned to exactly one feature, using the costs from matrix T. The result is an assignment matrix $\mathsf{A} \in \{0,1\}^{2\times m}$ where $\mathsf{A}_{k,l} = 1$, if feature l is mapped to dimension k, and $\mathsf{A}_{k,l} = 0$ otherwise. As T is small, enumerating all $m(m-1)$ possible partial assignments is feasible. The residual of ψ is then given by

$$r = \|\mathsf{A} \circ \mathsf{T}\|_1, \tag{4.7}$$

where $\circ$ denotes the component-wise (i. e., Hadamard) product of two matrices. In other words, the product keeps only the elements of T where A is non-zero. The 1-norm of the resulting matrix then gives the total absolute residual for all dimensions as a single scalar value.

Layout selection A unique and optimal solution to the inverse layout problem can be found for $o \geq 3$. For the hypothesis ψ with the smallest residual r, we set $\phi = \psi$ and $\mathsf{P} = \mathsf{A} \circ \mathsf{S}$, i. e., we keep the best layout function with the best feature projection. The optimal parameter translation vector can then be found as the offset of the centroids, i. e.,

$$\mathbf{t} = \frac{1}{o} \sum_{a \in C} \mathbf{p}_a - \mathsf{P}\mathbf{f}_a.$$

Lower-dimensional case It might not be possible to reliably infer the intended layout from the user constraints, because either not enough constraints are provided ($o < 3$), because of contradicting constraints, or because the constraints are non-unique. If the residual for all layouts is too high, we repeat the above procedure for a single dimension only, producing a matrix with a zero second row. As explained in Section 4.3.1, MDS will then be used to create the second parameter coordinate. If the residual using only this single component is still too high or no hypothesis could be build, MDS is used for both dimensions.

4.5 Results

All results were produced with our system by the authors and by user study participants.

Example Layouts In our main result, starting from an initial distribution users can manipulate the layout and the system tries to infer their layout idea (see Figure 4.10 to Figure 4.12 and captions). The special case of packing into an arbitrarily shaped container can be combined with additional constraints and layout functions, as seen in Figure 4.9.

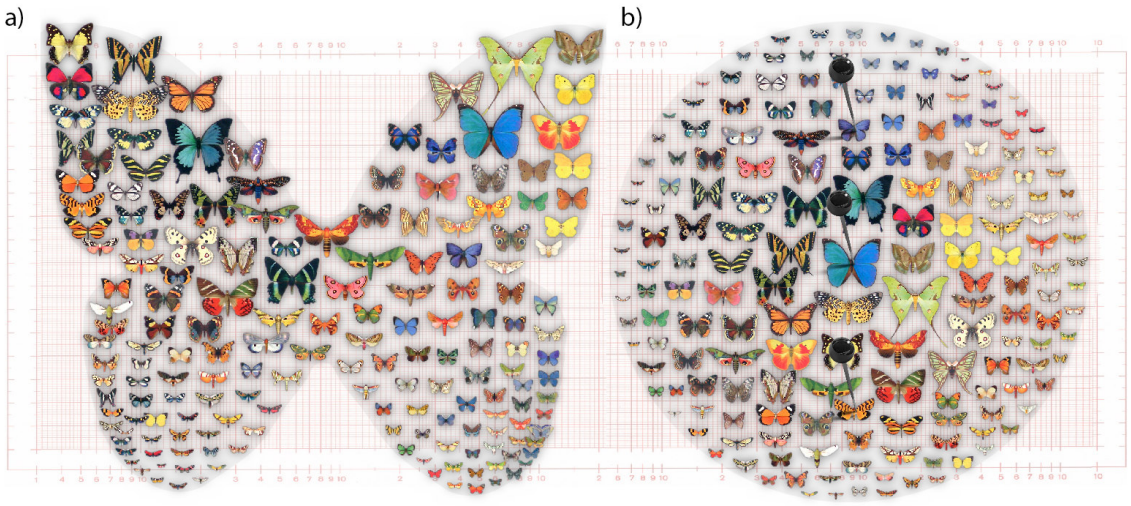

Figure 4.9: *a):* Layout of butterflies inside the boundary of a butterfly. *b):* A radial layout. Radius maps to size and hue to angle.

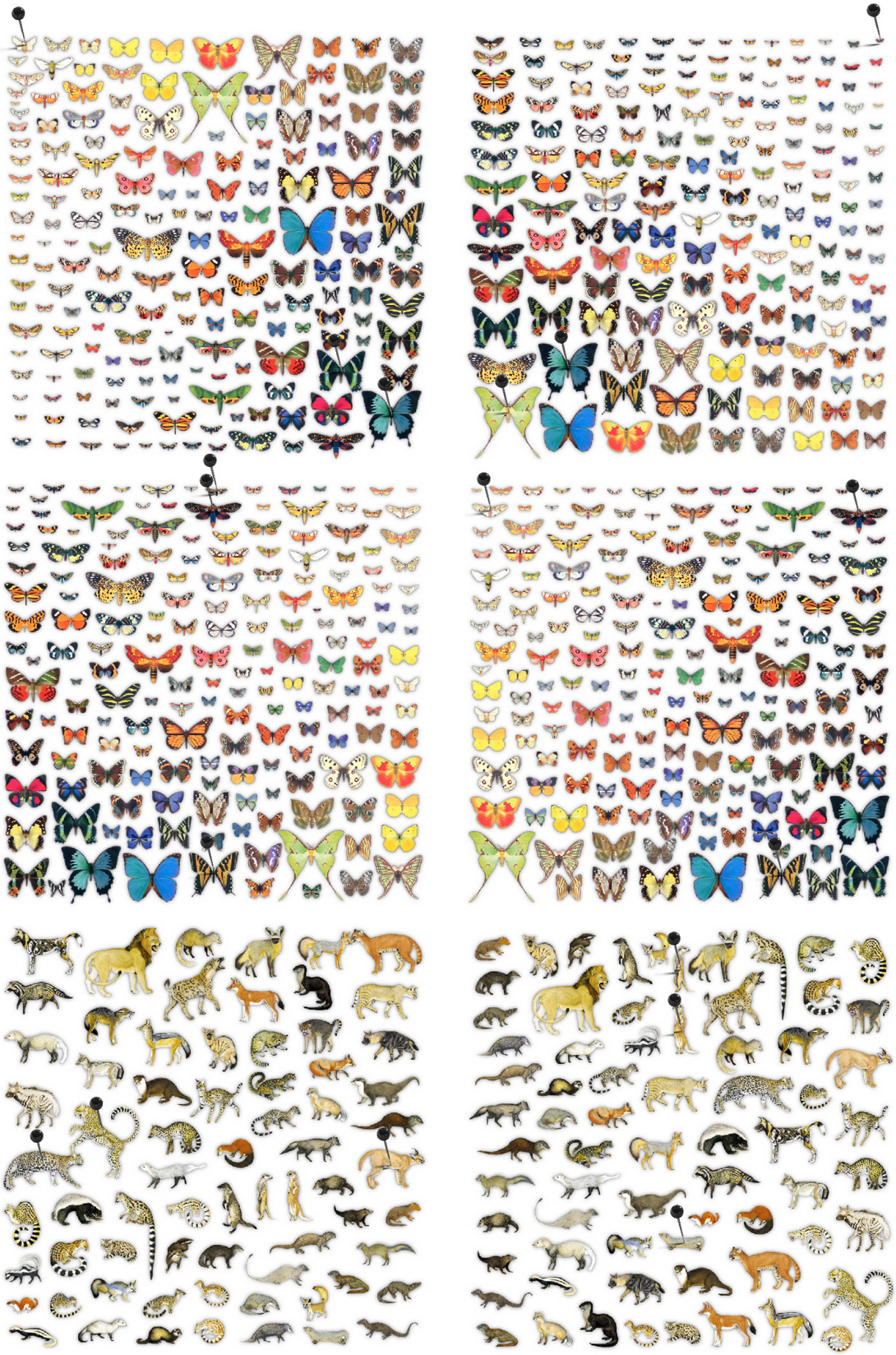

Figure 4.10: Results of our approach *(top to bottom, left to right, layout feature(s) in brackets)*. Butterflies *(brightness, size, shape anisotropy, shape anisotropy plus brightness)*. African carnivores *(color variance, orientation)*.

Figure 4.11: Results of our approach *(top to bottom, left to right, layout feature(s) in brackets)*. Frogs *(brightness)*. Dinosaurs *(brightness)*. Beetles *(size)*. Fishing baits *(hue)*. Bats *(shape anisotropy)*. Cookies *(radial layout by brightness as angle and hue as radius)*.

Figure 4.12: Results of our approach *(top to bottom, layout feature(s) in brackets)*. Sea moluscae *(size plus brightness)*. Whales *(brighness)*.

Finally, our system can be used to interactively explore multi-dimensional datasets, such as the countries in Figure 4.13.

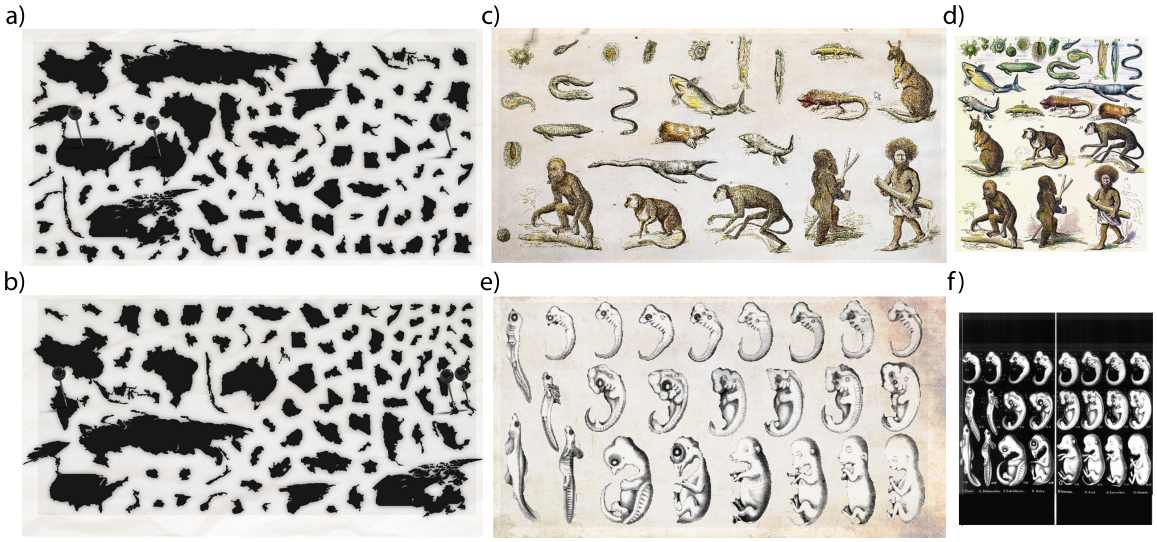

Figure 4.13: Semantic layouts: *a):* Layout by infant mortality rate from the World Fact Book. *b):* Now by population size. *c):* A remix of "The Descent of Men" by Ernst Häckel (1834–1919) seen in *(d)*. *e):* A remix of another layout by Häckel *(f)*.

User Study To design a user study, we first conducted a pilot questioning of two professional artists with a special expertise in packing layout production. According to the artists, besides legal and research issues, the most time-consuming steps are the generation of graphical primitives itself and the final layout. They say that achieving a balanced primitive distance (Section 4.3.2) is a major challenge. Here the artists wish to have "an automated layout tool creating a spatial boundary between primitives" and that these tools were "easier to work with" than currently available software packages. According to the artists, the position of the primitives is governed by their "taxonomic relationship" and by visual features such as brightness, texture, or drawing style but also by arranging them in a "visually pleasing way". The programs used by artists were Adobe Photoshop, Illustrator, InDesign, as well as Corel Draw which all do not offer distribution tools that generate balanced primitive distances in two dimensions.

To assess the usefulness of our system we conducted a user study, in which 15 naïve subjects were asked to produce a layout with three different user interfaces. The interfaces presented were a commercial software ("commercial interface", UI #1), an interface of our software with only our relaxation step (Section 4.3.2) enabled ("relaxation interface", UI #2, Section 4.3.2) and an interface with relaxation (Section 4.3.2) and inverse layout step (Section 4.4) enabled ("our interface", UI #3). For the commercial interface, we let the subjects choose from either Adobe Illustrator CS6 or Microsoft PowerPoint 2010 and rate their expertise in these programs for the presented task after the user study on a scale from 1 (novice) to 5 (expert). The task was to produce a layout that organizes primitives by specific features and has a balanced distance between the primitives. The initial layout was always identical: a fully relaxed MDS layout (cf. Section 4.3). Every session was self-timed until the subject was either satisfied with the generated layout or gave up due to fatigue. The order in which the interfaces were presented to the subjects was randomized for every session. For every session we logged the time spend to produce the final layout

and stored the final layout image result. For the relaxation interface and for our interface we additionally logged the cost (cf. Equation 4.3) of the final layouts. In the subsequent analysis we excluded cases in which the subjects gave up due to fatigue. Three users chose Adobe Illustrator CS6 and rated their expertise with 3.67 on average, the other twelve users chose Microsoft PowerPoint 2010 with an average expertise rating of 3.5. The layout generation took 16:32 minutes on average for the commercial interface with a standard deviation of 8:07 minutes, for the relaxation interface 7:24 minutes on average with a standard deviation of 2:18 minutes and for our interface 1:49 minutes with a standard deviation of 0:52 minutes (Figure 4.14, a). We conclude with statistical significance ($p < 0.0001$, ANOVA test) that our interface results in a speedup of approximately one order of magnitude. The average final residual of the relaxation interface was 2516.47 pixels^2 with a standard deviation of 2944.12 and for our interface we have an average cost of 590.87 pixel^2 with a standard deviation of 129.36. Again we found with statistical significance ($p < 0.027$, ANOVA) that our system results in a quantifiable quality gain.

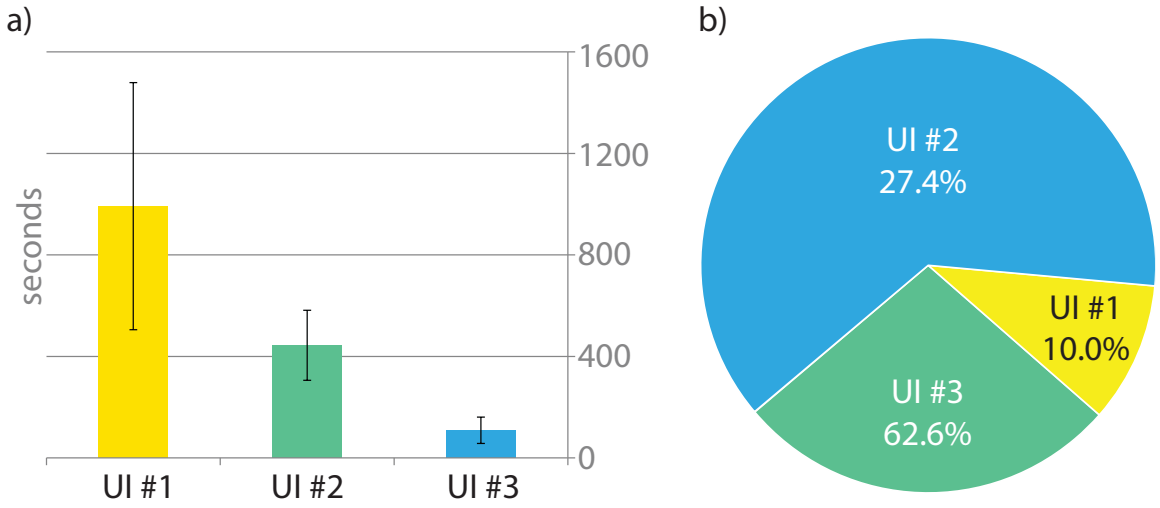

Figure 4.14: Results of the two user studies. *a:*) Average time in seconds spent generating the intended layout with each interface by 15 subjects. Variance is shown as black bars *b:*) Result preference of the layouts of *(a)* by a group of 19 subjects, disjoint from the first group.

In a second user study we asked a group of 19 subjects, disjoint from the first study, to rate the layouts produced by the subjects of the first user study (Figure 4.14, b). The three layouts of each subject of the first user study were presented to the new subjects and they had to pick the layout which they found best accomplished the given task. We found that in 62.63 % of the cases the layout produced by our interface was the preferred one, while 27.37 % preferred the one generated with the relaxation interface and 10 % were in favor of the layout of the commercial interface. We conclude with statistical significance ($p < 0.0004$, binomial between all pairs) that results from our interface are preferred over all alternatives at least by a factor of two. In 69.59% of the cases the subjects' choice correlates with the residual value (cf. Equation 4.3) of the first study.

During our first user study we made the following observations: with the commercial interface users tend to initially cluster primitives with extreme features first. Some users also took the clusters and arranged them in layers, in which primitives of one common feature appeared on the same layer. The hardest task seemed to be achieving a balanced distance; users either used too much or too few space, resulting in unsatisfying results.

Chapter 5

Projective Blue-Noise Sampling

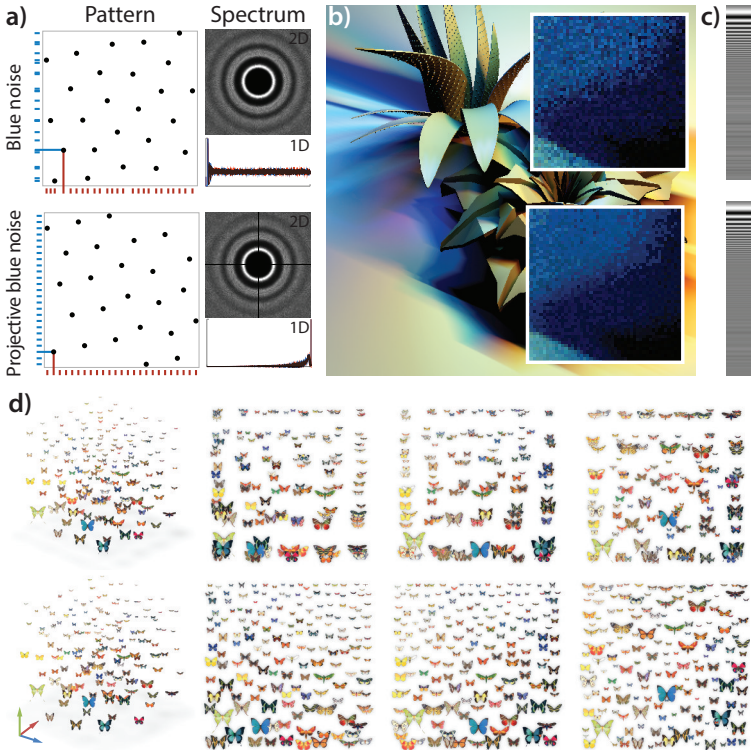

Figure 5.1: Our projective blue-noise distributions *(a, bottom)* have blue-noise spectra in two dimensions as well as in their one-dimensional projections, while classic two-dimensional blue-noise *(a, top)* has an almost white-noise spectrum in one dimension. Applications include Monte Carlo rendering *(b)*, reconstruction *(c)*, as well as placement of primitives *(d)*, such that they are well-distributed both in three dimensions and when projected to two dimensions.

5.1 Introduction

Producing "good" sampling patterns is an important task in many computer graphics applications, including simulation, rendering, image reconstruction, and primitive placement (Chapter 4). But what makes a good pattern depends on the application. For Monte Carlo rendering, low integration error is important [Cook, 1986; Shirley, 1991], which is usually achieved by stratified Latin hypercube or low-discrepancy point sets that have well-distributed low-dimensional projections. For digital half-toning [Ulichney, 1987], stippling [Kopf et al., 2006], and object placement [Hiller, Hellwig and Deussen, 2003], blue-noise point sets with maximized minimal inter-sample distance are preferred, as their structure closely resembles the photoreceptor arrangement in the eye retina [Yellott, 1983]. Such patterns are also desirable for image reconstruction [Dippé and Wold, 1985] where low-discrepancy patterns can lead to spurious aliasing artifacts [Mitchell, 1987]. At the same time, it has been shown that blue-noise patterns are not as competitive for numerical integration [Shirley, 1991]. All these applications call for point sets that are uniformly distributed in the sampling domain, though research in each area has focused on optimizing the distribution for its slightly different definition of uniformity. It has remained an open question whether there exist distributions that meet the requirements of a wide range of applications.

In this chapter we propose *projective blue-noise* point distributions, in an attempt to give a positive answer to the above question. A key property of these distributions is that they retain their blue-noise characteristics when undergoing one or multiple projections to lower-dimensional subspaces (Figure 5.1, a). We show how the classic dart throwing and Lloyd relaxation algorithms can be extended to produce such point sets, and demonstrate the usefulness of their projective blue-noise properties in various applications. For Monte Carlo rendering (Figure 5.1, b) and image reconstruction (Figure 5.1, c), our patterns often outperform existing distributions for functions with variations concentrated along one dimension, where the resulting sampling quality is dominated by a projection of the pattern. Furthermore, while common blue-noise patterns are useful for placing primitives in three-dimensional space, our patterns preserve the good visual distribution when the arrangement is viewed from different angles, i. e., projected to different image planes (Figure 5.1, d).

The rest of this chapter is organized as follows. In Section 5.2 we describe the construction of projective blue-noise patterns and discuss some practical considerations. Section 5.3 analyzes the properties of our point sets and the performance of our construction algorithms. In Section 5.4 we demonstrate the versatility of our method in practical applications, followed by a discussion in Section 5.5.

5.2 Our approach

The basic idea of projective blue noise is to extend existing sampling methods to not only operate in the full d-dimensional sample space, but also in multiple lower-dimensional projective subspaces simultaneously. While classic dart throwing and Lloyd relaxation test

candidates or move points only in the full-dimensional space, our extensions additionally check candidates, respectively move points, in the projection subspaces. We specify all spaces via a set of m projection vectors

$$\mathcal{B} = \{\mathbf{b}_j \in \{0,1\}^d\}_{j=1}^m. \tag{5.1}$$

The vectors $\mathbf{b}_j$ can be used with the Hadamard product (i. e., the element-wise vector multiplication) to project points and vectors onto the (sub)spaces specified by those vectors. Classic, non-projective blue-noise is a special case with $\mathcal{B} = \{\{1\}^d\}$. In our projective extension, for two-dimensional point sets, we use $\mathcal{B} = \{(1,1),(1,0),(0,1)\}$.

In the following subsections, we extend the dart throwing and Lloyd relaxation algorithms to projective blue-noise sampling. For each method, we first review its classic, non-projective variant, before presenting our projective extension.

We create sample patterns $\mathbf{X} = \{x_0,\ldots,x_n\}$ with $n \in \mathbb{N}^+$.

5.2.1 Dart throwing

Classic Dart throwing (Figure 5.2, left) starts with an empty point set and iteratively generates random candidate points $\mathbf{x}$ that are added to the set only if their distance to every other point $\mathbf{x}_i$ is larger than a certain threshold r, called the Poisson-disk radius [Cook, 1986]:

$$\min_{i=1,\ldots,n'} \|\mathbf{x}-\mathbf{x}_i\| > r, \tag{5.2}$$

where $n' < n$ is the number of the previously accepted points. For tiled patterns, the distance is computed on a toroidally wrapped domain. In its most basic form, the algorithm terminates if no new points can be added after a certain number of successive failed attempts. Alternatively, instead of terminating, the radius can be shrunk by a constant factor, which makes the sampling method progressive [McCool and Fiume, 1992]. Our implementation operates on a toroidal domain and also incorporates this shrinkage.

Projective Our dart throwing extension (Figure 5.2, right) accepts a candidate point $\mathbf{x}$ only if its distance to every other point $\mathbf{x}_i$ in the full d-dimensional space and in every projection space is larger than a certain threshold:

$$\min_{i=1,\ldots,n'} \left\|\mathbf{b}_j \circ (\mathbf{x}-\mathbf{x}_i)\right\| > r_j \quad \forall j \in 1,\ldots,m, \tag{5.3}$$

where r_j is the desired radius (i. e., minimum distance) in the j-th space, and $\circ$ is the aforementioned Hadamard product.

Radii Since the distances between points are smaller in lower-dimensional subspaces, the radii for these spaces should be smaller than the ones for higher-dimensional spaces. We derive the radius for a space from the radius of the tightest known lattice sphere packing in

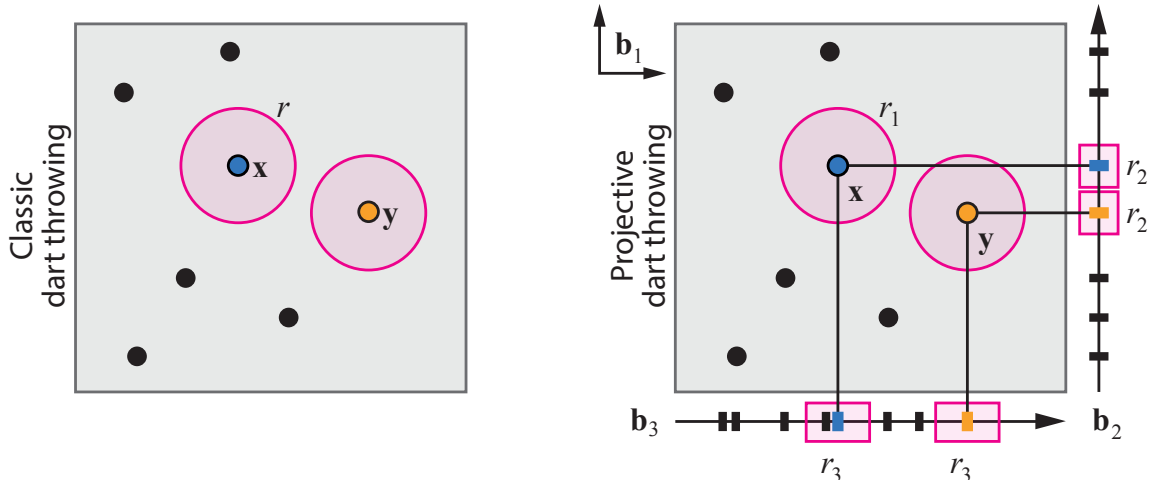

Figure 5.2: Classic dart throwing *(left)* accepts both the orange and the blue candidate points, as no existing point in the set is within the rejection radius r *(pink circle)*. Projective dart throwing *(right)* accepts the orange candidate, but not the blue one, as its projection onto $\mathbf{b}_3$, with radius r_3 *(pink box)* conflicts with an existing point.

the corresponding dimension. For one through four dimensions, these maximum radii are given respectively by [Lagae and Dutré, 2006; Korkin and Zolotarev, 1877]

$$r_1^{\max} = \frac{1}{2n}, \quad r_2^{\max} = \sqrt{\frac{1}{2\sqrt{3}n}}, \quad r_3^{\max} = \sqrt[3]{\frac{1}{4\sqrt{2}n}}, \quad r_4^{\max} = \sqrt[4]{\frac{1}{8n}}.$$

For higher dimensions d, the maximum radius is found by solving $\mathrm{V}_d = \eta_d / n$ for the radius $r_d^{\max}$ of a d-dimensional sphere, where V_d is the sphere volume and η_d is the best lattice disk packing density.

Given the maximum packing radii, we compute the Poisson-disk radii for every dimension as

$$r_j = r \cdot \frac{r_{d_j}^{\max}}{r_d^{\max}}, \tag{5.4}$$

where $d_j = \|\mathbf{b}_j\|_1$ is the dimension of space j and d is the dimension of the full space. The algorithm is now controlled via a single initial parameter $r \in [0,1]$, which we set to $r = 0.15$ for all our experiments.

5.2.2 Lloyd relaxation

Classic The Lloyd optimization algorithm (Figure 5.3, left) constructs a point arrangement that is a Centroidal Voronoi Tessellation (CVT). In a CVT, each point, or site, $\mathbf{x}_i$ is also the center of its associated Voronoi cell – the subset of the domain that is closer to $\mathbf{x}_i$ than to any other site. To obtain such a set $X = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$, Lloyd relaxation minimizes

the cost

$$c(X) = \sum_{i=1}^{n} \int_{\Omega_i} \|\mathbf{x}_i - \mathbf{x}\|^2 \mathrm{d}\mathbf{x}, \tag{5.5}$$

where Ω_i is the i-th cell in the Voronoi tessellation of X. Conceptually, this cost measures how far the sites are from the center of mass of their Voronoi cells. As with dart throwing, for tiled patterns the distances are computed on a toroidally wrapped domain.

The relaxation algorithm starts with a random set of sites X which is refined iteratively in three steps. First, a Voronoi tessellation of X is built, mapping every location in the domain to its closest site. Second, the centroid of every Voronoi cell is computed by averaging the locations in the cell. Finally, every site is moved to the centroid of its associated cell. Practical implementations often discretize the domain into a finite set of locations, turning the integral of Equation 5.5 into a sum.

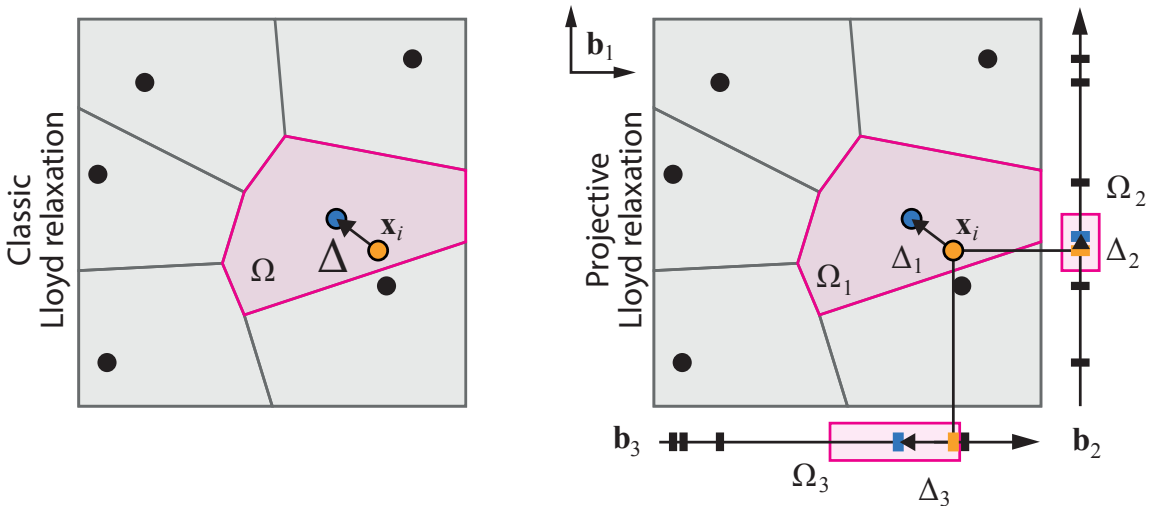

Figure 5.3: Classic Lloyd relaxation *(left)* moves each point $\mathbf{x}_i$ *(orange)* to the center *(blue)* of its Voronoi cell *(pink)*. Our projective extension *(right)* tries to move each point $\mathbf{x}_i$ to the center of its Voronoi cell in the full space $\mathbf{b}_1$ as well as in the two subspaces specified by $\mathbf{b}_2$ and $\mathbf{b}_3$.

Projective In our case, we want to distribute the sites uniformly not only in the full-dimensional space, but also in multiple lower-dimensional projection subspaces (Figure 5.3, right). Similarly to our dart throwing extension, we specify all spaces via a set of m projection vectors $\mathbf{b}_j$. We build m Voronoi tessellations, one for each space, and aim to optimize the sites, such that they coincide with centroids of their corresponding Voronoi cells in all tessellations. Note that for any site, its associated cells are different in the different (sub-)spaces. The cost of each site is computed by summing m weighted norms, and the total cost reads

$$c_{\mathrm{p}}(X) = \sum_{j=1}^{m} w_j \sum_{i=1}^{n} \int_{\Omega_{i,j}} \left\| \phi(b_j \circ \mathbf{x}_i) - \mathbf{x} \right\|^2 \mathrm{d}\mathbf{x}, \tag{5.6}$$

where $\circ$ is the Hadamard product, $\{w_j\}_{j=1}^m$ is a set of scalar projection weights that sum up to one, and $\Omega_{i,j}$ is the Voronoi cell of the i-th site after projection onto the j-th space. Note that the dimension of the points $\mathbf{x}$ above depends on j and ϕ reduces this dimensionality to the relevant dimensions of the subspace.

Our relaxation scheme seeks to minimize Equation 5.6. We start with the same initial point set as the classic Lloyd method, but we perform the optimization steps in m spaces simultaneously as follows: For each site $\mathbf{x}_i$, we compute m correction vectors $\Delta_{i,j}$, one from the Voronoi tessellation in each space. Each vector $\Delta_{i,j}$ would move $\mathbf{x}_i$ to the center of Voronoi cell $\Omega_{i,j}$. These vectors are generally different for the different projections j, and each individual site can be moved to exactly fulfill only one constraint locally. We instead try to partially fulfill all constraints by applying all correction vectors $\Delta_{i,j}$ to site $\mathbf{x}_i$, each scaled by a corresponding weight w_j. After moving all sites, the m Voronoi tessellations are recomputed and the process is iterated.

Note that the projective correction vectors are heuristically chosen and not proven to be optimal as the non-projective vectors are, but work well in practice as shown by our analysis.

Weights Ideally, we want all spaces to have equal importance in the total cost in Equation 5.6. However, since distances between points in lower-dimensional spaces are shorter, the relative contribution of such spaces is smaller than that of higher-dimensional spaces. We equalize all contributions by making each weight w_j inversely proportional to the tightest sphere packing radius in dimension d_j (cf. Section 5.2.1, Weights):

$$w_j = \frac{1/r_{d_j}^{\max}}{\sum_{k=1}^{m} 1/r_{d_k}^{\max}}. \tag{5.7}$$

Additional optimizations Due to the increased number of constraints, the straightforward implementation of the above scheme may converge much slower than the classic Lloyd relaxation algorithm. We propose two enhancements to improve both the speed and the quality of the resulting patterns.

First, in early iterations, the different correction vectors contradict heavily, making the process susceptible to local minima and slow convergence. To remedy this, the weights w_j for the lower-dimensional spaces are faded in linearly from 0 for 50 iterations. This is done for each dimension successively, i. e., first the weights of the $(d-1)$-dimensional subspaces are faded in, then the $(d-2)$-dimensional ones, etc.

Second, the convergence speed can be further increased by exploiting the fact that in one dimension the best point arrangement that maximizes the mutual minimum distance is the regular distribution [Ramamoorthi et al., 2012], which is the global minimum of the Lloyd cost in one dimension (Equation 5.5). Thus, to satisfy a single one-dimensional projection, the pattern can simply be "snapped" to a regular grid along the corresponding axis [Saka, Gunzburger and Burkhardt, 2007]. So instead of building one-dimensional Voronoi tessellations, we directly compute the correction vectors as the differences between the regular grid $\{(i+0.5)/n\}_{i=0}^n$ and the sorted point coordinates along each axis. Note that

this closed-form solution works only for one-dimensional projections, and minimizing the cost in multiple dimensions simultaneously still requires iterative optimization.

5.3 Analysis

In this section we compare the quality of our patterns to existing methods in terms of their spectral and projective properties, Poisson-disk radii and discrepancy. We also compare our method to latinization [Saka, Gunzburger and Burkhardt, 2007] and analyze the convergence and the computational performance of our projective Lloyd relaxation.

5.3.1 Projective analysis

We begin with an analysis of the spectral and spatial properties of our projective patterns. We follow the recommendations of Schlömer et al. [2011] on reporting the spectral properties for two-dimensional point sets, including the critical frequency f_c and anisotropy reference levels in Figure 5.1, Figure 5.4, Figure 5.5, Figure 5.6, Figure 5.7, and Figure 5.8. For the one-, three-, and four-dimensional power spectrum plots we generalize the relevant frequency ranges using the respective $r_d^{\max}$ (see Section 5.2.1, Weights).

2D analysis In Figure 5.4, Figure 5.5, and Figure 5.6 we compare various 2D patterns in terms of their Fourier power spectra as well as their star discrepancy and Poisson-disk radii. The patterns we consider are: (1) regular, (2) Sobol, (3) scrambled Larcher-Pillichshammer (SLP) [Kollig and Keller, 2002], (4) uniform random, (5) jittered, (6) Latin hypercube (LH), (7) multi-jittered (MJ) [Chiu, Shirley and Wang, 1994], (8) correlated multi-jittered (CMJ) [Kensler, 2013], (9) dart throwing, (10) Lloyd relaxation, (11) latinized dart throwing [Saka, Gunzburger and Burkhardt, 2007], (12) latinized Lloyd relaxation [Saka, Gunzburger and Burkhardt, 2007], (13) our projective dart throwing, and (14) our projective Lloyd relaxation. All plotted patterns consist of 25 samples. The spectral power plots have been computed as the average of the periodograms of 100 random instances of each pattern with 3025 samples. The star discrepancy L_*^∞ and Poisson-disk radii ρ (i. e., the normalized global minimum inter-sample distance [Lagae and Dutré, 2008]) have been computed for sample patterns of size 529.

A characteristic feature of two-dimensional blue-noise distributions is their isotropic Fourier power spectrum with a black disk in the center indicating the absence of low-frequency content, surrounded by an energy-peak ring around the principal frequency [Lagae and Dutré, 2008]. Such distributions are produced by maximizing the minimum distance between the points in the set. As discussed in Section 5.2.2, in one dimension the regular distribution achieves the maximum point separation, and its Fourier power spectrum is zero except at frequencies that are multiples of n (the number of points). As a consequence, in Figure 5.4 and Figure 5.5 patterns with good one-dimensional axis projections, such as LH, MJ, as well as the latinized and our projective patterns, have black crosses in their two-dimensional power spectra. This is because the interesting power spectrum features of such distributions

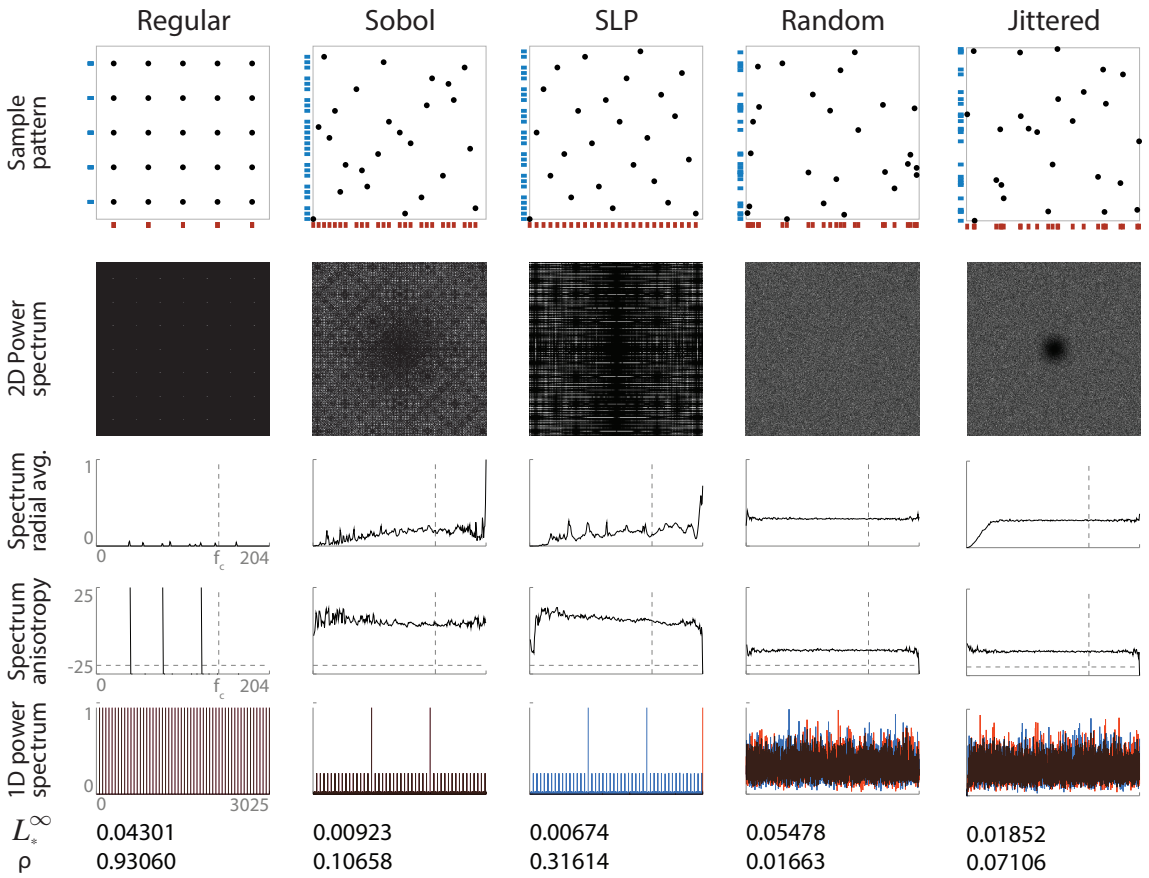

Figure 5.4: Spectral analysis of five different two-dimensional point sets and their one-dimensional axis projections *(shown as red and blue bars and graphs respectively)*, discussed in Section 5.3. Note that the horizontal (frequency) scale is different for the one-dimensional power spectrum plots and the radial average and anisotropy plots. The last rows report the star discrepancy L_*^∞ and Poisson-disk radius ρ of each point set.

occur at different scales in one and two dimensions respectively. For the sake of visual clarity, we clamp the two-dimensional spectrum plots to the frequency range $[0;204]$, and show the one-dimensional projection slices of these spectra as separate plots in the range $[0;3025]$. Note also that the black crosses in the two-dimensional spectra cause a slight artificial increase in anisotropy.

The Sobol and SLP low-discrepancy patterns achieve regular one-dimensional axis projections for sample counts that are powers of two. For other sample counts however, gaps remain along at least one axis. In our 25-sample example in Figure 5.4 and Figure 5.5, only the x-axis of the SLP pattern has a uniform distribution. Furthermore, both methods fail to produce blue-noise two-dimensional spectra.

Uniform random, regular, and jittered sampling all fail to produce high-quality two-dimensional blue-noise distribution and one-dimensional projections. The regular two-dimensional patterns have very poor projection distributions with multiple points sharing the same coordinates on each axis, causing spikes in the one-dimensional power spectra.

Latin hypercube (LH) sampling has good projection properties but poor two-dimensional uniformity.

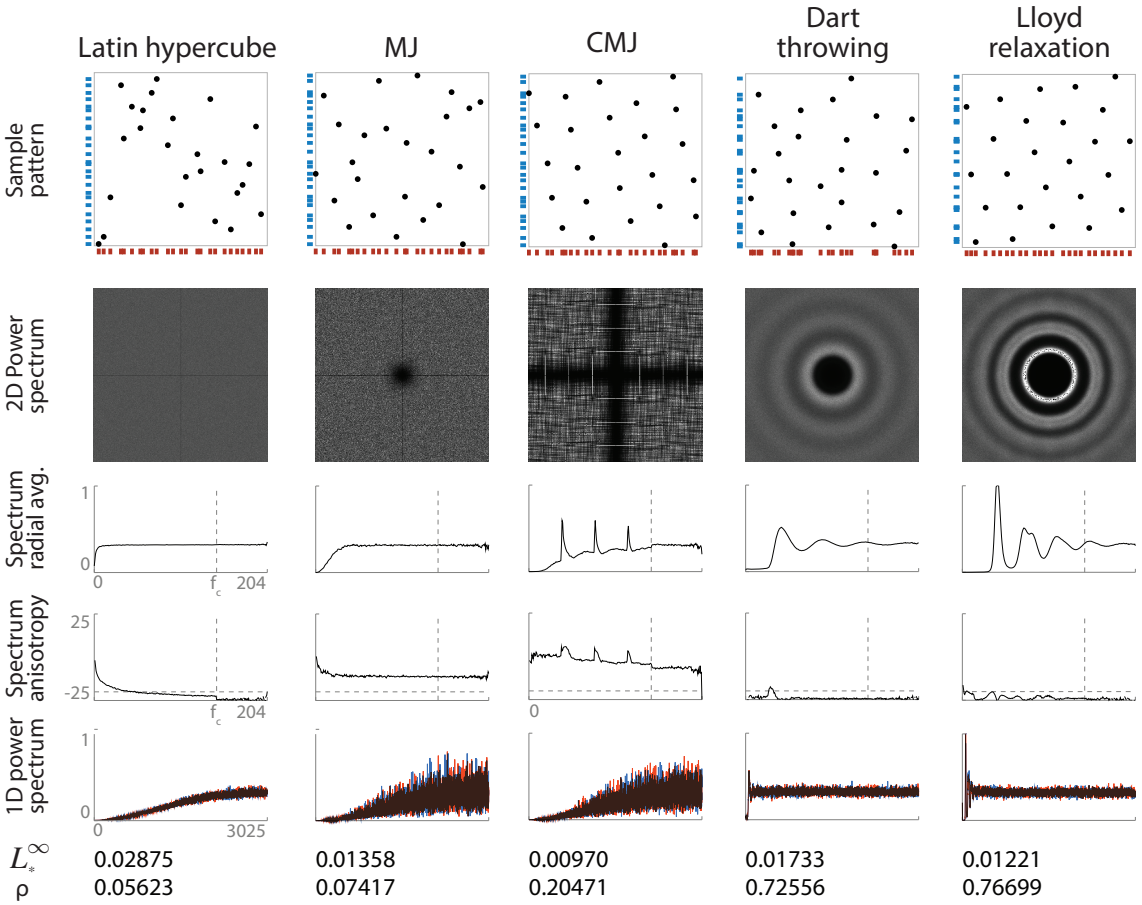

Figure 5.5: Spectral analysis of five different two-dimensional point sets and their one-dimensional axis projections *(shown as red and blue bars and graphs respectively)*, discussed in Section 5.3. Note that the horizontal (frequency) scale is different for the one-dimensional power spectrum plots and the radial average and anisotropy plots. The last rows report the star discrepancy L_*^∞ and Poisson-disk radius ρ of each point set.

Multi-jittered (MJ) sampling [Chiu, Shirley and Wang, 1994], as a combination of jittered and LH sampling, produces a two-dimensional power spectrum that shares the features of both methods. MJ patterns have acceptable projections, but like other jittered patterns do not guarantee a minimal inter-sample distance, which is reflected in the smooth low-frequency ramps in both the two-dimensional and the projected one-dimensional power spectra. CMJ [Kensler, 2013] results in identical one-dimensional spectra but a much less uniform two-dimensional spectrum.

Classic Lloyd relaxation produces excellent blue-noise distributions but only in two dimensions. Latinizing the pattern improves the one-dimensional projections but at the cost of increasing the anisotropy and decreasing the Poisson-disk radius. Our projective Lloyd distribution combines a good two-dimensional spectrum with well-distributed one-dimensional projections, and has a larger Poisson-disk radius than the latinized variant. Improving the

one-dimensional projections of a blue-noise pattern also decreases its discrepancy [Saka, Gunzburger and Burkhardt, 2007], where our projective method once again comes on top of latinization.

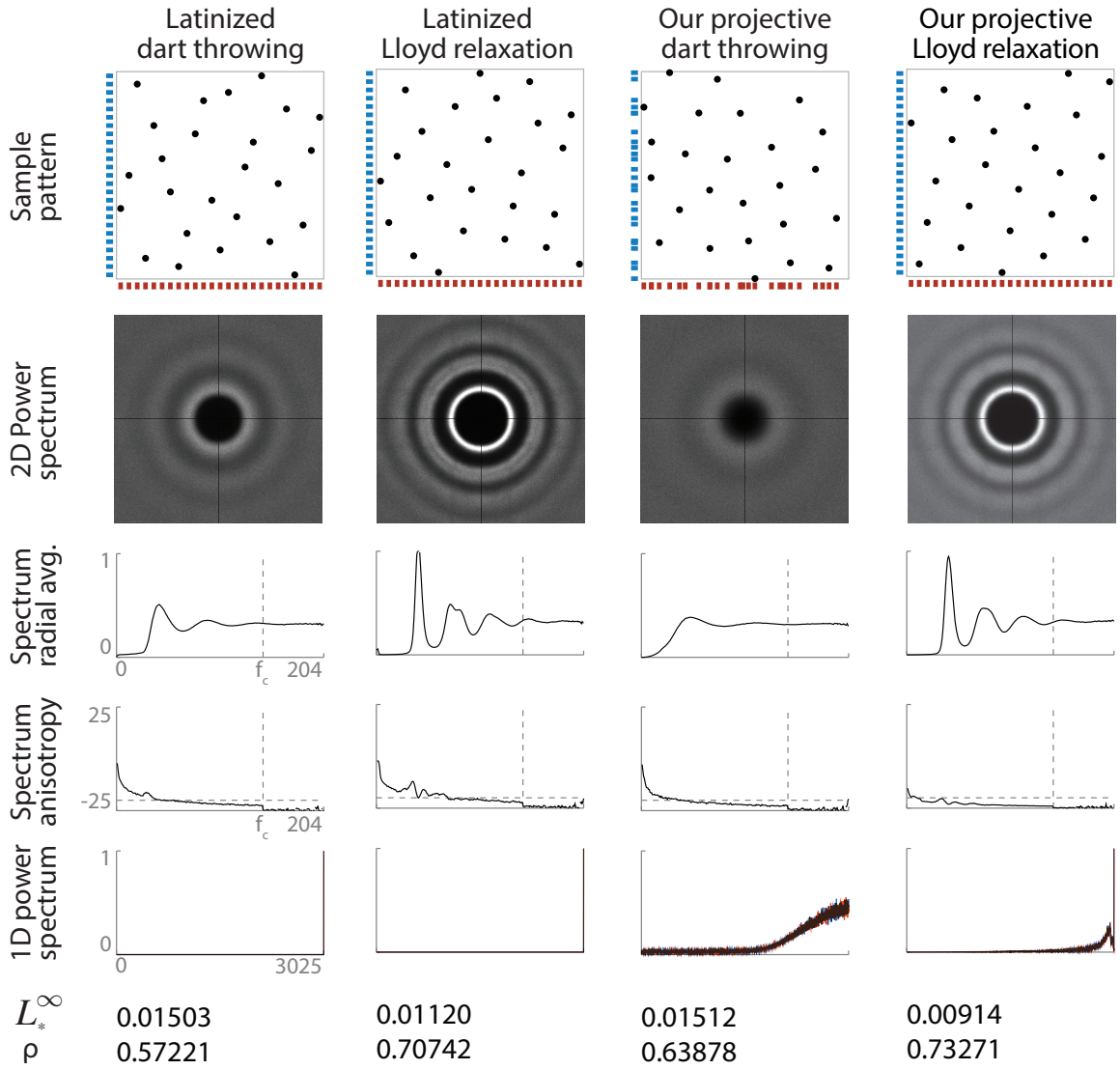

Figure 5.6: Spectral analysis of four different two-dimensional point sets and their one-dimensional axis projections *(shown as red and blue bars and graphs respectively)*, discussed in Section 5.3. Note that the horizontal (frequency) scale is different for the one-dimensional power spectrum plots and the radial average and anisotropy plots. The last rows report the star discrepancy L_*^∞ and Poisson-disk radius ρ of each point set.

Finally, dart throwing produces inferior blue-noise distributions compared to Lloyd relaxation, as confirmed by both the two-dimensional and the projected one-dimensional spectral plots. This is not unexpected, as this method is highly sensitive to the order in which the samples are inserted. The additional constraints introduced by our projective extension further reduce the probability of successfully inserting a candidate point. As a result, the two-dimensional blue-noise spectrum of projective dart throwing suffers more from the

imposed constraints than that of Lloyd relaxation, and some low-frequency components appear – the dark circle in the center of the spectrum is grey, not black. On the other hand, latinized dart throwing produces good projective and non-projective spectra but also higher anisotropy.

Three-dimensional analysis In Figure 5.7 we compare the power spectra of classic three-dimensional blue-noise patterns to the projective distributions produced by our extended Lloyd relaxation. We see that classic, non-projective patterns (top row) have inferior two-dimensional spectral properties and almost white-noise one-dimensional distributions. Our approach (middle and bottom rows) retains good quality in all specified projective subspaces, although this comes at the cost of a slight quality degradation in the three-dimensional spectrum.

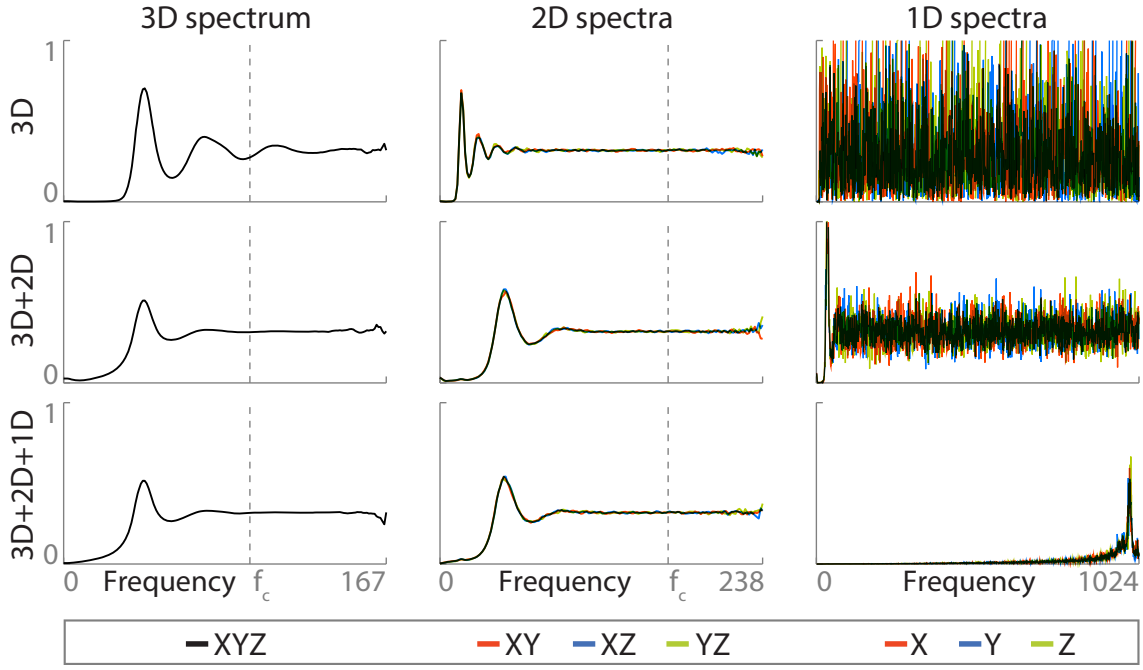

Figure 5.7: The three-, two-, and one-dimensional power spectra *(columns)* of 1,024-sample patterns, averaged from 10 periodograms, produced by our projective Lloyd relaxation with different projections enabled *(rows)*. The spectra of the different same-dimensional projections are color-coded.

Four-dimensional analysis In Figure 5.8 we plot the power spectra of four-dimensional patterns produced by our extended Lloyd relaxation with projective blue-noise properties imposed on all lower-dimensional subspaces. The characteristic blue-noise spectrum shape in all projections demonstrates that our method generalizes to higher dimensions. Note the different frequency scales of each plot, indicating a quality improvement in the corresponding dimension over classic four-dimensional blue noise, as also seen in Figure 5.7. However, imposing this many constraints leads to a more noticeable overall deterioration in

blue-noise quality compared to the three-dimensional case. The noise in the plots is due to the relatively low number of periodograms averaged (ten).

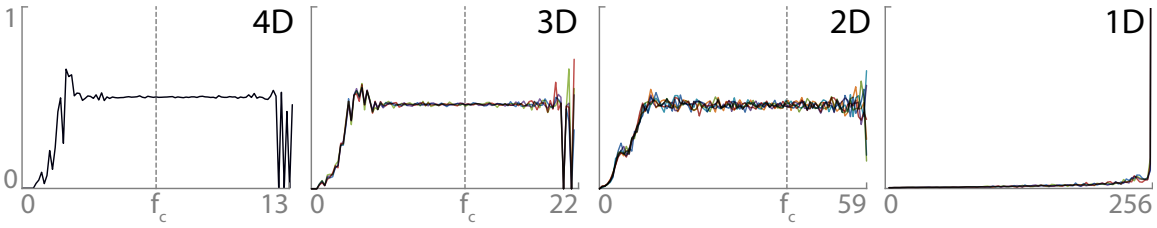

Figure 5.8: The power spectra of a 256-sample 4D projective blue-noise point distribution, averaged from 10 periodograms. Each plot shows the radial averages for all subspaces of the same dimensionality: 1 in four, 4 in three, 6 in two, and 4 in one dimension.

5.3.2 Comparison to latinization

Having analyzed the spectral properties of our projective Lloyd relaxation and the latinization method of Saka et al. [2007] in Section 5.3.1, we now conduct a closer comparison between these two approaches. On a high level, our notion is a non-greedy, high-dimensional generalization of latinization: Instead of first creating a blue-noise pattern and latinizing it as a post-process, we interleave these two steps. Note that while our method handles arbitrary dimensions, latinization only considers 2D patterns and their 1D axis projections. We look at the average projective Poisson-disk radius w.r.t. our subspace-vector $\mathbf{b}$:

$$\tilde{\rho}_{\mathbf{b}} = \frac{1}{n} \sum_{\mathbf{x} \in X} \min_{\mathbf{y} \in X} \|\mathbf{b} \circ (\mathbf{x} - \mathbf{y})\| / d_{\mathbf{b}}^{\max}, \tag{5.8}$$

which is the projective generalization of the average Poisson-disk radius measure [Schlömer, Heck and Deussen, 2011]. The minimal inter-sample distance is normalized by the optimal packing distance $d_{\mathbf{b}}^{\max}$ in subspace $\mathbf{b}$, i. e., $d_{\mathbf{b}}^{\max} = 2r_{\|\mathbf{b}\|_1}^{\max}$. In the following, we will use $\tilde{\rho}$, $\tilde{\rho}_{\mathbf{x}}$, and $\tilde{\rho}_{\mathbf{y}}$ to denote the average radii in two dimensions and in the x- and y-axis projections respectively, corresponding to projections using the vectors $(1,1)$, $(1,0)$ and $(0,1)$.

For patterns with 512 samples, latinized dart throwing and our projective variant achieve $(\tilde{\rho}, \tilde{\rho}_{\mathbf{x}}, \tilde{\rho}_{\mathbf{y}}) = (0.801, 1, 1)$ and $(\tilde{\rho}, \tilde{\rho}_{\mathbf{x}}, \tilde{\rho}_{\mathbf{y}}) = (0.806, 0.917, 0.921)$ respectively. Latinized Lloyd relaxation produces $(\tilde{\rho}, \tilde{\rho}_{\mathbf{x}}, \tilde{\rho}_{\mathbf{y}}) = (0.873, 1, 1)$, whereas our projective variant gives the best balance with $(\tilde{\rho}, \tilde{\rho}_{\mathbf{x}}, \tilde{\rho}_{\mathbf{y}}) = (0.909, 0.998, 0.999)$. In summary, latinization produces perfect one-dimensional stratification, but at the cost of decreasing the uniformity in two dimensions. Our approach optimizes for both measures simultaneously.

The above measures indicate that in two dimensions, latinized dart throwing outperforms our projective variant. However, latinization has a negative effect on the progressiveness of the algorithm, one of the major benefits using this algorithm. We analyze this in Figure 5.9 by comparing the average Poisson-disk radii for classical, latinized, and our projective dart throwing and the Sobol low-discrepancy sequence [Sobol, 1994]. We plot the radii as functions of the first n points in 512-sample sets, where the latinized pattern is computed

from the classical dart-throwing pattern after generating all samples. The plots reveal that our projective dart throwing has the most consistent progressive performance in one and two dimensions, while the latinized variant has good properties only for sample counts close to the maximum. The classical dart throwing performs well in two dimensions, but its one-dimensional projections are consistently poor. In contrast, the Sobol pattern is perfectly latinized for power-of-two sample counts, but its two-dimensional radius is significantly worse than those of the dart throwing patterns.

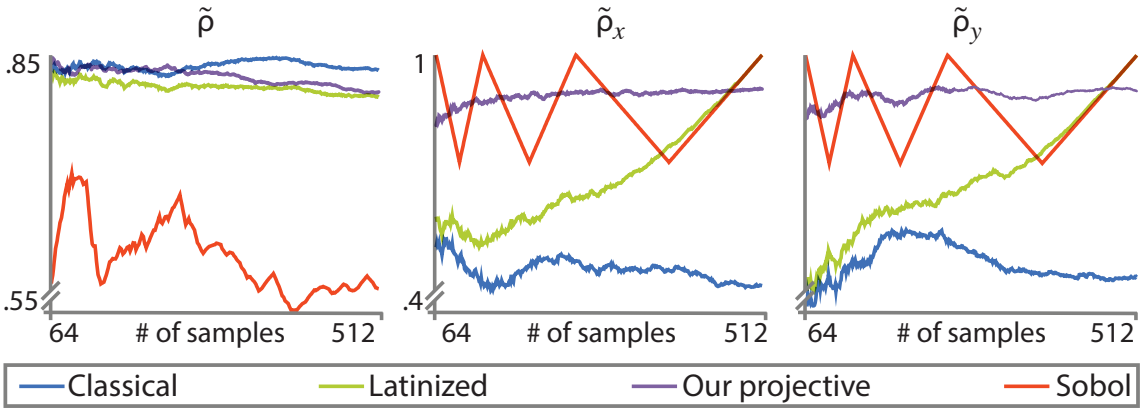

Figure 5.9: Average, generalized Poisson-disk radii *(bigger is better)* in two dimensions and along the x- and y-axis projections *(left to right)* as functions of increasing sample subset size for three dart throwing variants and the Sobol sequence. *Please see Section 5.3.2 for further details.*

5.3.3 Rotation

Our approach supports rotating the canonical coordinate axes, as long as they remain orthogonal. Since our methods operate on a toroidal domain, this can be trivially achieved by optimizing the pattern for the canonical axes (Figure 5.10, a) before rotating it by the desired amount (Figure 5.10, b). The black cross in the resulting power spectrum is oriented according to the rotation angle (Figure 5.10, c). The same approach can also be used in higher dimensions, as the patterns tile in all dimensions.

5.3.4 Sample warping

Monte Carlo rendering and primitive placement often require warping samples according to a specified importance function. In Figure 5.11 we compare the warping quality of our projective Lloyd patterns against other patterns on a thin 2D curve, which can represent e. g. an environment light source for rendering or a path on a (hyper-)plane for primitive placement. To produce the warped patterns, we first create samples as described before, i. e., using a uniform importance. We then warp these samples to follow a new importance function (Figure 5.11, left) using the common approach of tabulating conditional and marginal distributions along the x- and y-axis respectively [Devroye, 1986, p. 555]. Our

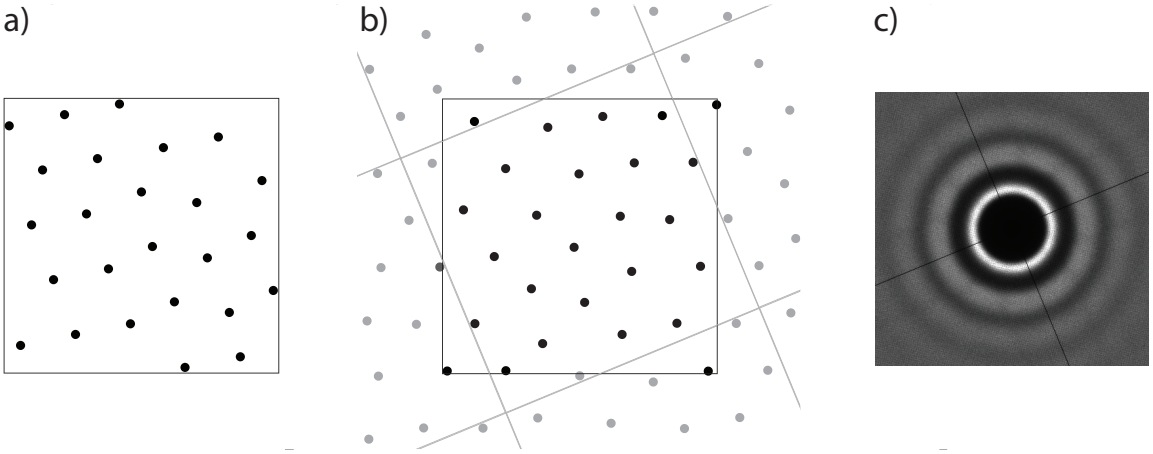

Figure 5.10: Rotation of projection axes. *a):* Original pattern. *b):* Tiled and rotated pattern. *c):* Resulting power spectrum.

pattern achieves the most uniform warped distribution, which in rendering translates to a low integration error. This is due to its well-distributed projections along both axes, a property that no other pattern in the comparison has.

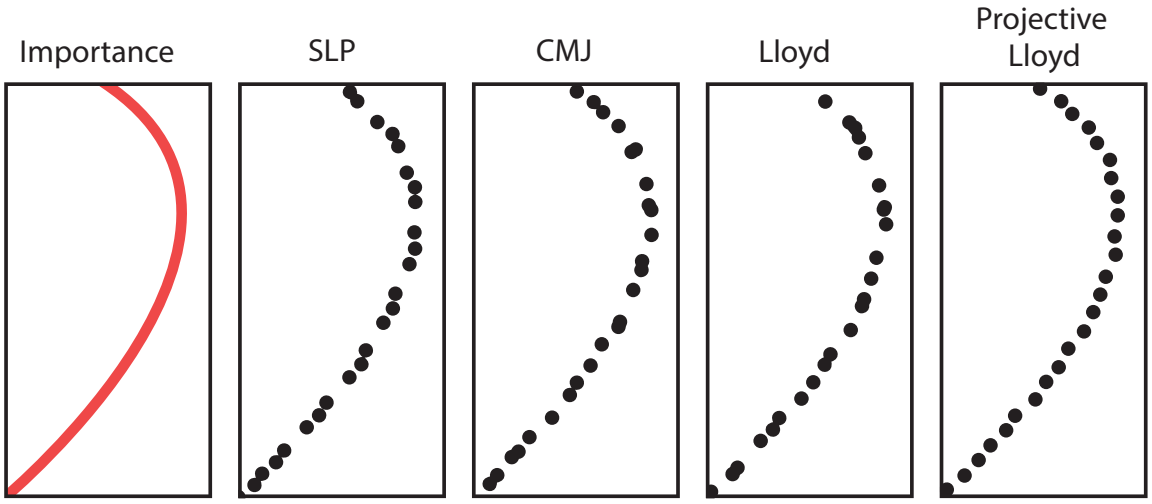

Figure 5.11: Importance-driven warping of different 25-sample 2D patterns. Our projective Lloyd pattern produces the best distribution, thanks to its good projections along both axes.

5.3.5 Lloyd convergence

We now analyze the convergence of different variants of our projective Lloyd optimization on a two-dimensional point set X. In Figure 5.12 we plot the classic two-dimensional cost $c(X)$ (Equation 5.5) and the projective one-dimensional part $c_{\mathrm{p}}(X) - w_1 \cdot c(X)$ of the generalized cost (Equation 5.6), where w_1 is the full (two-dimensional) space weight, over an increasing number of iterations. We consider six variants of our algorithm: classic

non-projective, latinization, as well as variants with and without the weight fading and 1D grid snapping optimizations from Section 5.2.2. A sixth graph shows the effect of latinizing the pattern after 50 optimization iterations.

We see that all non-latinized variants minimize the classic two-dimensional cost, though with slightly different speeds. Our projective variants achieve a final 2D cost only 0.76% worse than that of the classical method (note the small vertical range in the left plot, $[0.16; 0.18]$), while also minimizing the one-dimensional costs. On the other hand, latinization brings excellent projective properties but at the expense of increasing the total cost.

Unsurprisingly, the projective one-dimensional cost is not minimized by the classic Lloyd relaxation. The weight fading increases the projective cost initially, but ultimately achieves a substantially faster convergence. The one-dimensional grid snapping adds a further constant runtime improvement.

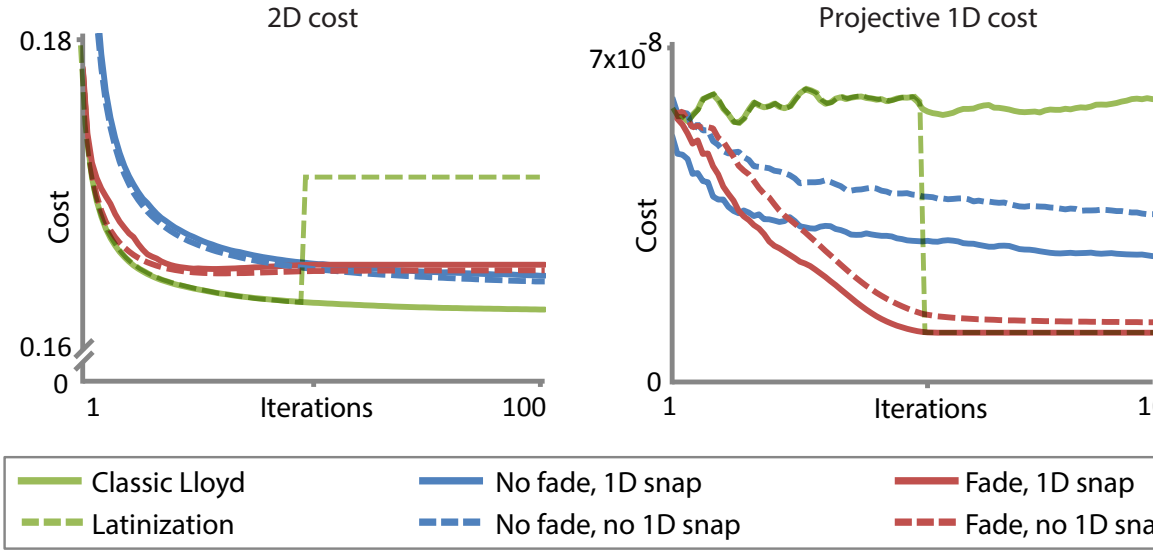

Figure 5.12: Two- and one-dimensional cost plots at different iterations of classic and latinized Lloyd relaxation as well as four different variants of our projective extension for a pattern with 2048 samples (cf. Section 5.3.5). Different weight fading strategies are shown in different colors, while solid and dashed lines denote the utilization of one-dimensional grid snapping (Section 5.2.2).

5.3.6 Performance

The GPU implementation of our projective Lloyd relaxation performs slower than classic Lloyd relaxation by a factor slightly smaller than m (the number of projection spaces). While the performance of the one-dimensional relaxation is improved by the grid snapping optimization, its computational cost is insignificant compared to the higher-dimensional Voronoi tessellations. Our naïve implementation of projective dart throwing is about three times slower than the classic method.

5.4 Applications

In this section we demonstrate the utility of our projective blue-noise patterns in rendering, image reconstruction and primitive placement – applications that have traditionally relied on specialized distributions.

5.4.1 Rendering

To analyze the effect of light source anisotropy, in Figure 5.13, top-left we plot the RMS reference error of three patterns on a simple scene for varying numbers of samples. Classic Lloyd relaxation (green) and SLP (blue) exhibit strong variation in quality. For power-of-two sample counts, SLP is on par with our projective Lloyd pattern (red) which otherwise performs consistently better. We speculate that classic Lloyd relaxation spuriously achieves the same quality as ours when it occasionally produces patterns with good one-dimensional projections. For a square area light source (Figure 5.13, bottom-left), the differences are smaller, though our approach again consistently outperforms classic Lloyd relaxation. Finally, in Figure 5.13 right we plot the RMS error as a function of the light source anisotropy.

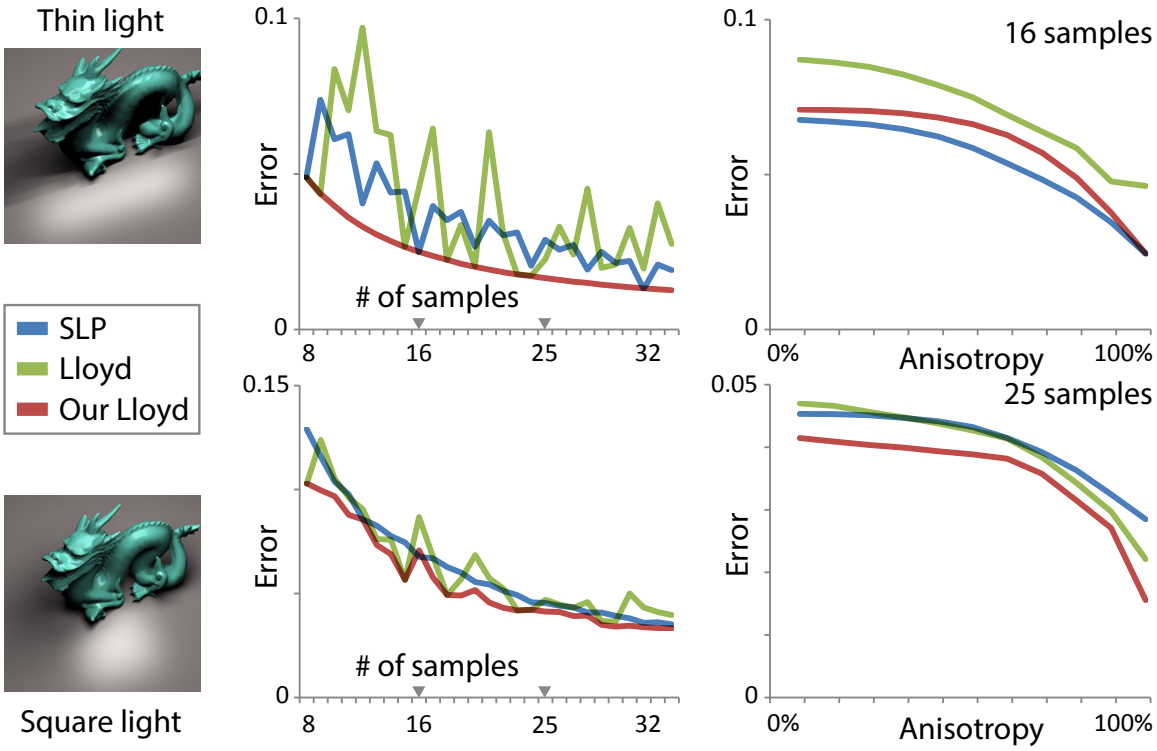

Figure 5.13: RMS error plots of the renderings of a scene using three different sample patterns. We plot the error as a function of sample count *(left plot)* and area light source anisotropy *(right plot)*.

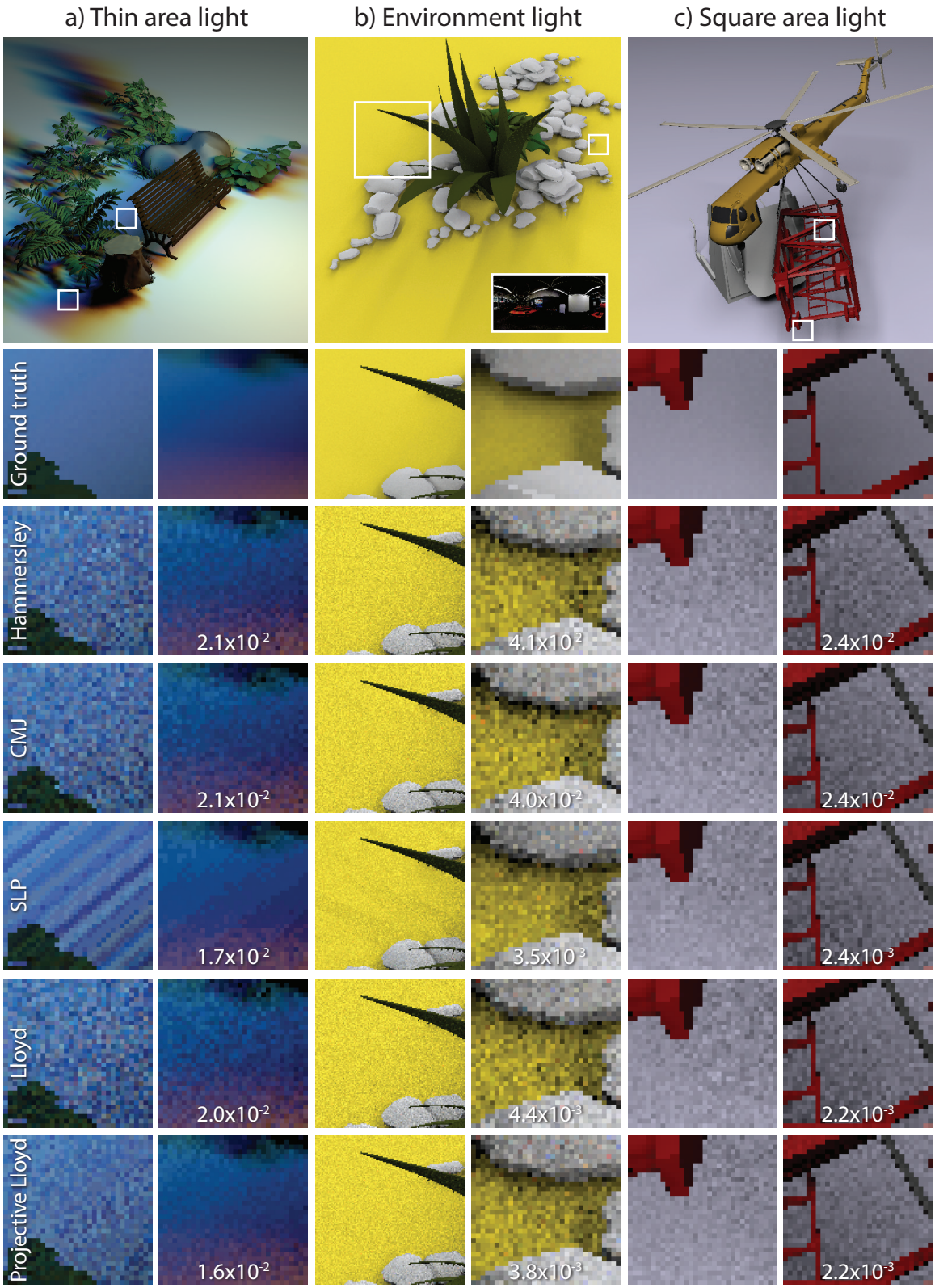

Figure 5.14: Rendering comparison of five sample patterns on three scenes with different light sources, discussed in Section 5.4.1.

In Figure 5.14 we compare several patterns for Monte Carlo rendering of direct illumination from three different types of extended light sources. The light sources used to produce the renderings are (a) an anisotropic area light with a color gradient, 21 samples per pixel (spp); (b) an importance-sampled environment light (see inset in Figure 5.14, b and Section 5.3.4), 25 spp; and (c) a square area light, 25 spp. We decorrelate the pixel estimators for all but the SLP and CMJ patterns (as they employ a specialized scrambling themselves) via Cranley-Patterson rotation [Cranley and Patterson, 1976]. We use (1) Hammersley [Kollig and Keller, 2002], (2) correlated multi-jittered (CMJ) [Kensler, 2013], (3) scrambled Larcher-Pillichshammer (SLP) [Kollig and Keller, 2002], (4) classic Lloyd relaxation (which was slightly better than classical dart throwing), and (5) our projective Lloyd relaxation (which was slightly better than our projective dart throwing) to sample the rectangular domains of the area and environment light sources.

The visual and numerical results in Figure 5.14 indicate that our projective blue-noise patterns outperform all others on thin area lights. SLP is the closest competitor on all scenes and even slightly outperforms our patterns on the environment map scene. However the pattern exhibits pixel correlation which is clearly visible in both the first and the second scenes, as also observed by Kensler [2013]. The environment map in the second scene features thin curved lights that are importance sampled as described in Section 5.3.4. In Figure 5.14, c we observed that the blue-noise patterns perform slightly better than the classical optimal methods also on a square area light source.

5.4.2 Image reconstruction

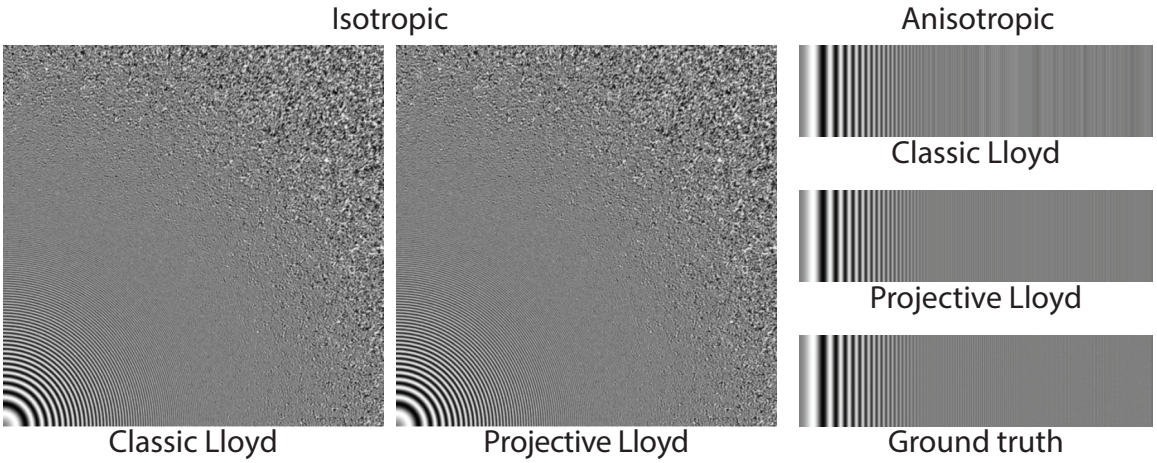

Figure 5.15: Reconstructions of an isotropic *(left two images)* and an anisotropic *(right images)* two-diemsnional zone plate function using a classic blue-noise pattern and our projective blue-noise pattern.

In Figure 5.15 we compare the performance of classic Lloyd relaxation and our projective extension for reconstructing two-dimensional images. We test a two-dimensional zone plate function with a 4-pixel-wide Lanczos filter (512×512 pixels $= 262{,}144$ samples) as well as a modified anisotropic variant (512×16 pixels $= 16{,}384$ samples). On the isotropic zone plate

(left), the two patterns perform similarly. However, our pattern significantly outperforms classic blue noise on the anisotropic variant (right), demonstrating the importance of having well-distributed one-dimensional projections.

5.4.3 Primitive placement

Finally, we demonstrate the utility of our three-dimensional projective blue-noise patterns for primitive placement. Our approach enables arranging objects in three dimensions in a way that is artistically pleasant [Hiller, Hellwig and Deussen, 2003], has semantic structure along its axes [Reinert, Ritschel and Seidel, 2013], and fills multiple two-dimensional projection subspaces uniformly. Figure 5.16 shows one such arrangement where primitives are sorted by size, brightness, and orientation (i. e., direction of the primitive's first principal component) along the x-, y-, and z-axis respectively. The distances in the projective Lloyd cost (cf. Equation 5.6) have been computed on a non-toroidal domain by taking the spatial extent of the objects into account [Reinert, Ritschel and Seidel, 2013]. When looking at the two-dimensional projection along an axis, the distribution remains uniform and shows two aspects of the data, e. g., size and brightness when looking down the z-axis.

5.5 Discussion

Compared to low-discrepancy sequences, our patterns are much more costly to construct. Compared to the cost of common blue-noise sampling approaches, however, the overhead is small and the produced patterns are of almost the same quality in the full-dimensional space. For numerical integration, we have found our projective two-dimensional Lloyd relaxation to perform consistently better than most other patterns in all our experiments. Nevertheless, our patterns can be slightly outperformed by low-discrepancy sequences that achieve perfect one-dimensional stratification for certain sample counts. We have only demonstrated our method for up to four dimensions; however our results indicate that it generalizes to higher dimensions required in full global illumination rendering.

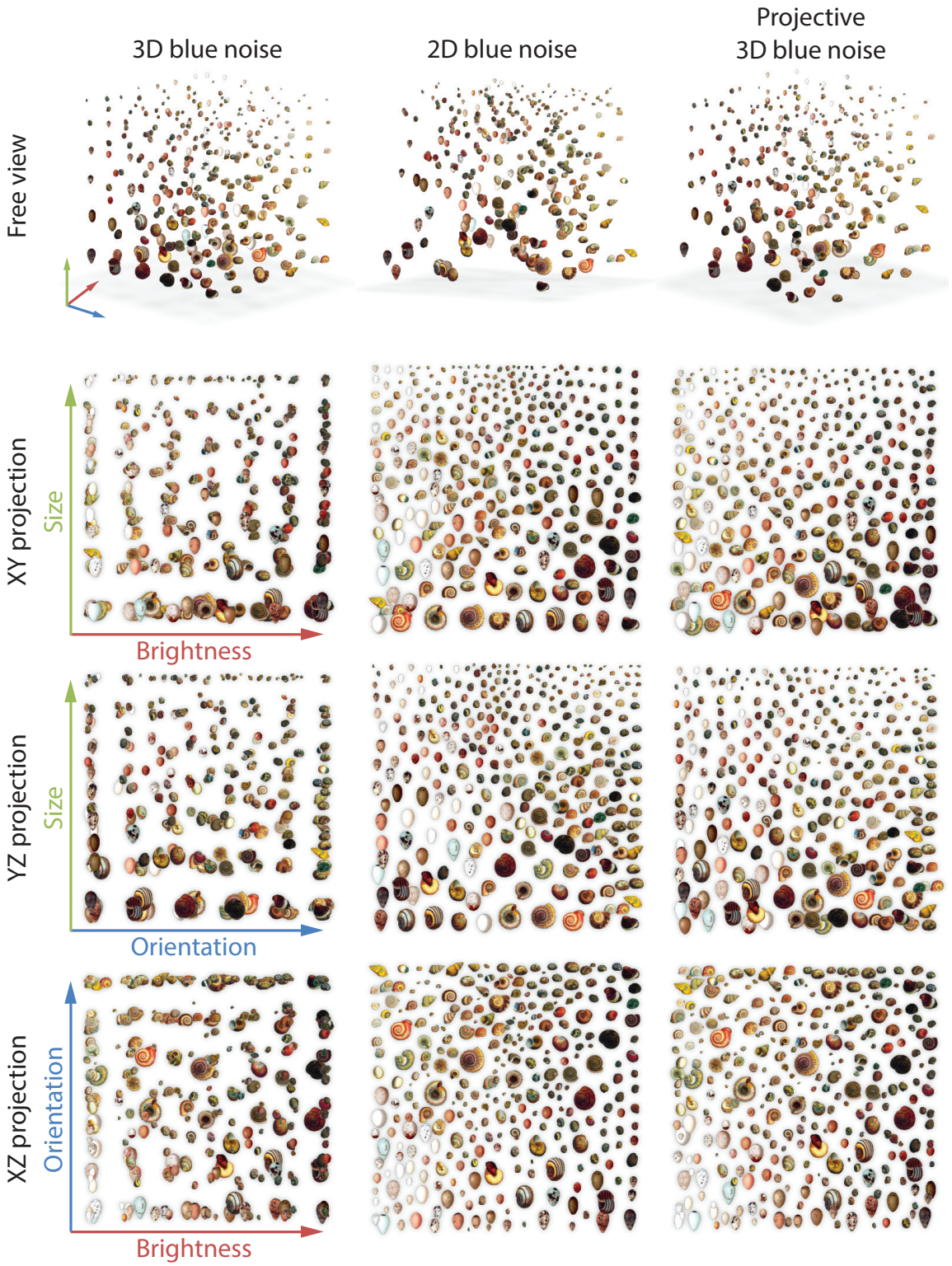

Figure 5.16: Using three-, two-, and our projective three-dimensional blue-noise patterns for distributing primitives ordered by brightness, size, and orientation along the x-, y-, and z-axis, respectively.

Chapter 6

Animated 3D Creatures from Single-View Video by Skeletal Sketching

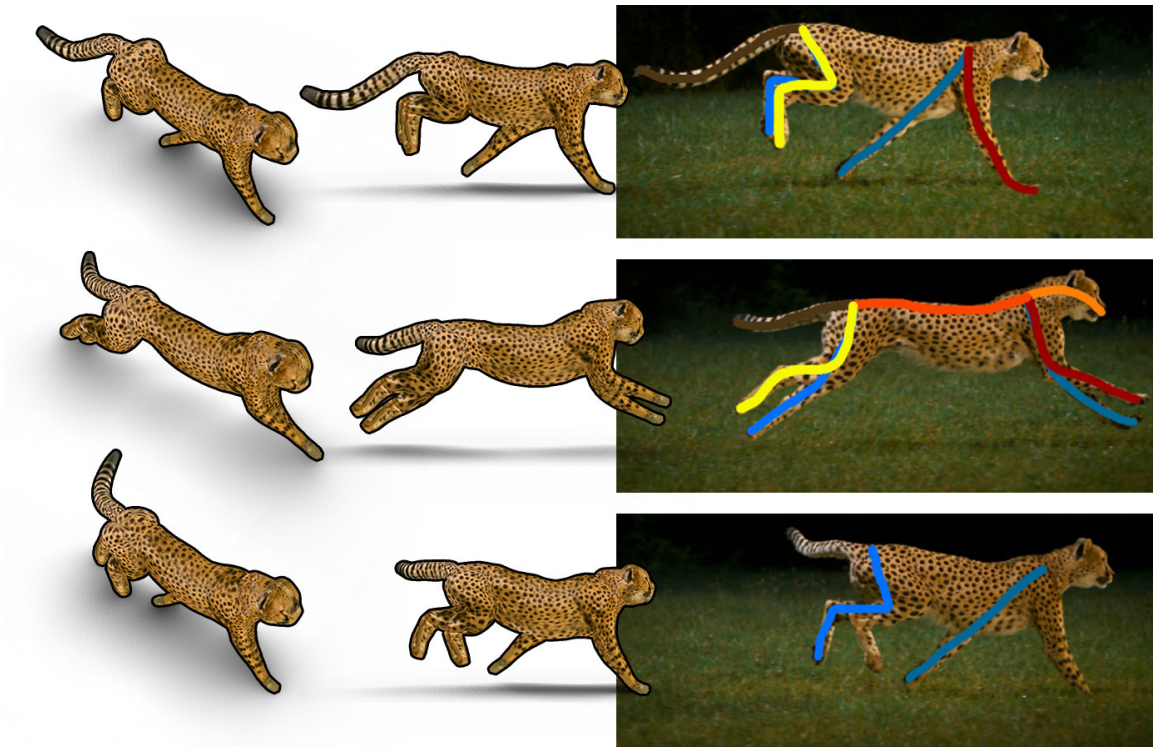

Figure 6.1: Our approach analyzes two-dimensional sketches *(colored lines in video frames of last column)* over a sparse set of key frames *(rows)* of a two-dimensional video *(last column)* to produce an animated and textured, three-dimensional mesh *(left two colums)*.

6.1 Introduction

The acquisition of dynamic three-dimensional content requires either dedicated scanning hardware, such as range sensors [Izadi et al., 2011], multi-camera setups [Matusik et al., 2000], or large amounts of manual effort [Maestri, 2006] often not accessible to casual users. Simpler means to convert a common monocular video into an animated and textured three-dimensional surface would provide new opportunities in display, manipulation, and transmission of visual content.

Multi-view reconstruction, e. g., from a large set of general community images or videos [Snavely, Seitz and Szeliski, 2006], provides an accessible means for the reconstruction of static geometry. Existing approaches reconstruct static models [Debevec, Taylor and Malik, 1996; Sinha et al., 2008] from multiple views using Structure from Motion (SfM) techniques. Extensions for moving or deformable objects [Cootes et al., 1992; Bregler, Hertzmann and Biermann, 2000; Akhter et al., 2008] were proposed, but the challenge to establish reliable correspondences between the different views remains difficult. While for rigid shapes a single linear transformation suffices, non-rigid content requires complicated space-time regularizations.

Inspired by a recent user-assisted reconstruction technique [Chen et al., 2013b], in this chapter we develop a sketch-based acquisition approach for the reconstruction of non-rigid shapes from a single view video. The premise of our approach is that the animated three-dimensional object can be modeled as a union of deforming generalized cylinders. The approach is guided by a user sketching a set of “limb strokes” indicating the axis of a generalized cylinder in a couple of key frames. The core of our method is to propagate this annotation from key frames to all video frames. Our method bypasses the meticulous effort required for specifying dense correspondences used in common SfM techniques. Instead, “limb strokes" are used to fuse the silhouettes of all video frames into a consistent, animated, three-dimensional shape. Combining different silhouettes does not only result in animation, but also resolves shape details that are not revealed in a single frame. The procedure can be performed at interactive rates, allowing the user to refine the result until the desired quality is achieved.

Our results demonstrate extraction of animated, three-dimensional shapes such as animals in motion from video with applications including cloning, reposing, novel view synthesis, texture transfer and three-dimensional printing.

6.2 From skeletal sketches to animated shapes

6.2.1 Overview

Input to our method is an arbitrary video sequence depicting a moving creature and a set of strokes that the user draws on top of a sparse set of key frames. Each stroke is drawn alongside one body part of this creature, representing a deforming “limb” potentially connected to other limbs. Users are free to draw as many strokes as they want but typically

each non-branching body part can be drawn in a single stroke. Every stroke is associated with a label and the user is required to draw consistently labelled strokes in all key frames. Our limb strokes are conceptual and do not necessarily have a biological meaning: As long as a creature's body part can be represented by a single generalized cylinder, i. e., without a branch, it can be drawn with a single stroke. We assume that the limbs observed in all frames deform but exhibit the same connectivity, although not all limb strokes are necessarily entirely observable in all frames. We shall show this rather simple user input suffices to enable the reconstruction of an animated mesh.

The four main components of our approach are stroke tracking, segmentation, cylinder fitting, as well as texturing (Figure 6.2). A simple user interface (Section 6.2.2) provides the necessary input.

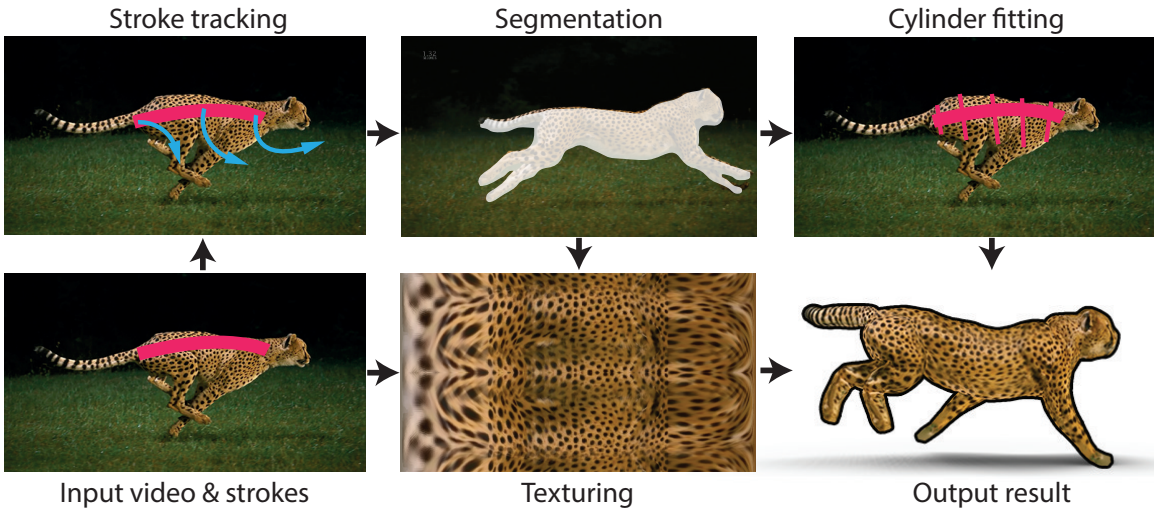

Figure 6.2: An overview of our approach with the dependencies of the individual steps.

Stroke tracking turns the temporally sparse limb strokes defined in a few key frames into temporally dense, coherent strokes for all frames. Section 6.2.5 describes tracking of strokes that form a skeleton over a video sequence. We exploit the connectivity and spatio-temporal structure to track complicated structures such as limbs with occlusion and pose changes over long frame ranges, such that for unoccluded parts often a single stroke is sufficient to be tracked over a typical sequence.

Segmentation uses the temporally densified strokes from tracking and the video to consistently segment foreground from background in all video frames, described in Section 6.2.6. Again, we exploit the space-time structure of the two-dimensional strokes to simplify the problem and make a consistent segmentation over many video frames from sparse input.

Cylinder fitting uses the foreground segmentation in all video frames to fit animated, three-dimensional cylinder geometry around each limb stroke. Section 6.2.7 explains how

to use marching in two-dimensional masks to implement a voting scheme to robustly find cylinder radii and three-dimensional paths.

Texturing finally utilizes the segmentation and video frames to assign texture colors to the animated, three-dimensional cylinders (Section 6.2.8).

6.2.2 User interface

The main user interface elements are the limb strokes suggesting the two-dimensional projection of a non-branching, generalized cylinder-like shape part. Identical limb stroke labels are identified in different frames by the same stroke color. Not all limb strokes need to be drawn in all key frames but the user is required to draw the full skeleton in the first one. The results of the stroke tracking step (Section 6.2.5) are used to complete and display the missing strokes in subsequent frames. The user can review the tracking results at any time using a time slider and insert a new key frame correcting only the inaccurately tracked strokes. Once the user has drawn a stroke in one key frame, for subsequent stroke sketches we display a circle with the maximal limb length found (Figure 6.3, limb length hint) to facilitate sketching of consistent stroke lengths, particularly for (partially) occluded limbs. Each limb stroke comes with a direction state that determines if the stroke points towards or away from the viewer which can be altered with a click and is detailed in Section 6.2.4 and Section 6.2.7.

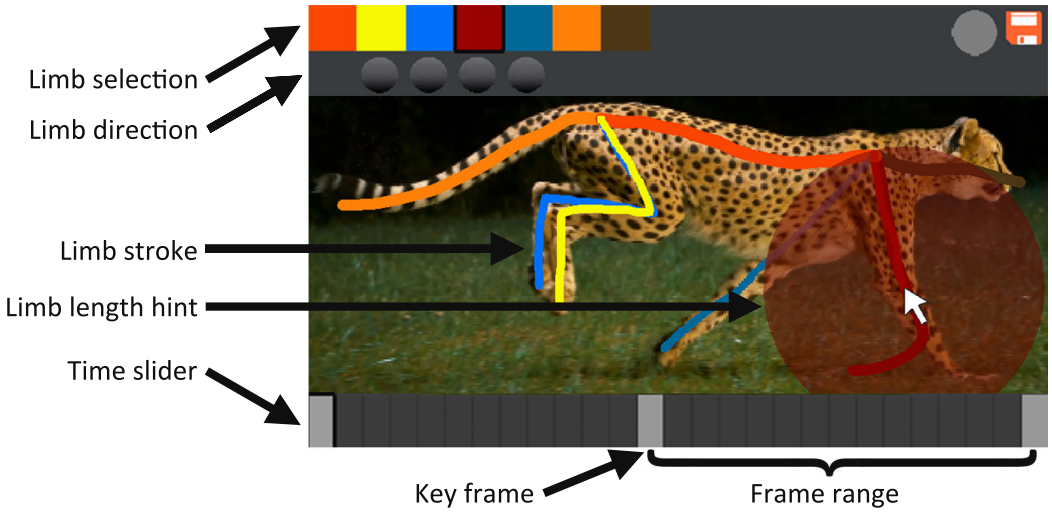

Figure 6.3: Our user interface showing the individual control elements for frame selection and stroke drawing. The functionality of each element is shown in the labels.

6.2.3 Preprocessing

Initially, the video sequence is resampled to a fixed resolution of 512×512 pixels for all steps to speed up the subsequent computations and allow identical parameter settings for all

video sequences. Cylinder fitting (Section 6.2.7) operates on a filtered variant of the scaled video to facilitate video segmentation: a five-times-repeated bilateral filter with a spatial radius of 10 pixels and a range radius of 0.02 (cf. Figure 6.5, a).

6.2.4 Stroke processing

Strokes are sequences of vertices defined at two-dimensional image locations. Each stroke of label $l \in \mathbb{N}^+$ drawn by the user in frame $f \in \mathbb{N}^+$ initially may consist of a varying number of vertices, defined by a set $\hat{X}^{l,f} = \{\hat{\mathbf{x}}_1^{l,f}, \hat{\mathbf{x}}_2^{l,f}, \ldots\}$ with $\hat{\mathbf{x}}_i^{l,f} \in \mathbb{R}^2$. As our subsequent steps require consistent vertex counts in all key frames, the strokes need to be resampled with an approximately equal inter-vertex distance along the original stroke. To this end we employ a linear resampling per label l with a vertex count $n_l \in \mathbb{N}^+$ depending on the maximum stroke length in all frames. A vertex sequence is resampled to a set $X^{l,f} = \{\mathbf{x}_1^{l,f}, \mathbf{x}_2^{l,f}, \ldots, \mathbf{x}_{n_l}^{l,f} \in \mathbb{R}^2\}$ such that $d_{\hat{X}^{l,f}}(\mathbf{x}_i, \mathbf{x}_{i+1}) \approx c, \forall i \in [1, n_l - 1]$ with $d_{\hat{X}^{l,f}}$ as the length along stroke $\hat{X}^{l,f}$. Stroke tracking (Section 6.2.5) uses an approximate inter-vertex distance of $c = 10$ pixels while segmentation (Section 6.2.6) and cylinder fitting (Section 6.2.7) operate on vertices with approximately $c = 1$ pixel distance. Ideally, stroke resampling should take texture features into account; due to limb deformations and occlusions matching those features however is not feasible in this step.

As we require all limb strokes to be connected to other limb strokes, they share common start- and/or end-vertices, resulting in a connection graph of strokes. We arbitrarily define the first vertex of the first limb stroke to be the root vertex, turning the connection graph into a connection tree.

We use a simple, heuristic symmetry detection on the strokes drawn by the user to bootstrap some of the subsequent steps. Two limb strokes drawn in the same key frame that share a connecting stroke vertex are defined to be symmetric if their stroke lengths differ by less than 10% and if the y-coordinates of their outer stroke vertex differ by less than 30 pixels. The limb associated with the stroke of higher label number is heuristically chosen to be the outside limb, further away from the viewer than the inside one. This decision can be flipped for both limbs using the limb direction buttons of the user interface.

6.2.5 Stroke tracking

Input are temporally sparse, two-dimensional strokes in some key frames. Output are dense "space-time strokes", i. e., two-dimensional image locations for every limb stroke vertex in every frame. Space-time strokes are tracked on the *range* between key frames forward and backward independently (cf. Figure 6.3). This potentially contradicting information of two overlapping ranges for each point in time is combined in a blending step.

Without loss of generality, we will next describe the forward tracking of all stroke vertices for one range. Starting from a key frame k (Figure 6.4, a) every frame in the range is tracked sequentially. The current frame is denoted by f whereas the previous frame in forward or backward tracking direction is indexed by $f \pm 1$. For every frame f, new candidate

positions $\mathbf{x}^{f,l}$ for every vertex of every stroke l are generated and combined into stroke candidates $\overline{X}^{f,l}$. For each of the stroke candidates we compute a unary cost $u(\mathbf{x}) : \mathbb{R}^2 \to \mathbb{R}$ encoding appearance constancy and a binary cost $b(\mathbf{x},\mathbf{y}) : \mathbb{R}^2 \times \mathbb{R}^2 \to \mathbb{R}$ encoding kinematic constraints between neighboring vertices of a stroke. Hence we can compute the cost of a candidate stroke $\overline{X}^{l,f}$ as

$$c(\overline{X}^{l,f}) = \sum_{i=1}^{n_l} u(\overline{\mathbf{x}}_i^{l,f}) + \lambda \sum_{i=1}^{n_l-1} b(\overline{\mathbf{x}}_i^{l,f}, \overline{\mathbf{x}}_{i+1}^{l,f}), \tag{6.1}$$

where $\lambda \in \mathbb{R}$ constitutes a relative weight between unary and binary terms. Finally, the best sequence of candidates that globally minimizes Equation 6.1 is selected. We will detail every step in the following.

Candidate generation Candidate vertex positions $\overline{\mathbf{x}}^{l,f}$ for the current frame are generated in a small window around the vertex position $\mathbf{x}^{l,f\pm1}$ of the previous frame (Figure 6.4, b). Two types of candidates are produced: Occluded and non-occluded ones. Non-occluded candidates are placed for every pixel in a search window of 64×64 pixels around the vertex position of the previous frame. Additionally, 100 occluded candidate vertex positions are placed on a regular grid in the identical search window, allowing occluded and non-occluded candidates to potentially use the same position. Non-occluded candidates that fall outside of the image are removed, whereas occluded candidates are allowed to fall outside the image, enabling tracking of strokes partially outside of the image. Each candidate stores its occlusion type for later stages.

Unary cost The unary cost (cf. Equation 6.1) models appearance consistency between frames (filled circles and squares in Figure 6.4). To this end, we compare a neighborhood patch $\mathcal{N}$ of size 32×32 pixels around each non-occluded candidate position to a patch of the same size around the vertex position in previous frames using a sum of squared pixel color distances (SSD), i. e.,

$$u(\mathbf{x}) = \min_{T \in \mathcal{T} \times i \in [1,3]} \sum_{\mathbf{o} \in \mathcal{N}} \| a^{f \pm i}(T(\mathbf{x}+\mathbf{o})) - a^f(\mathbf{x}+\mathbf{o}) \|_2^2,$$

where $a^f(\mathbf{x})$ gives the pixel values at position $\mathbf{x}$ in frame f, $T \in \mathcal{T}$ is single transformation of a set of transformations $\mathcal{T}$ and i is a frame distance detailed below. SSD is used instead of normalized cross correlation (NCC) as the differences in appearance are small between frames and mainly due to deformations rather than illumination changes. Further, SSD can trivially be parallelized whereas NCC requires additional computations.

In particular, we consider *variants* in template and orientation when taking this difference, defined as a set of rigid transformations $\mathcal{T}$. First, the difference to the patch around the three last non-occluded vertex positions is enumerated. Second, we allow for 10 degrees of positive or negative template rotation in seven steps. The unary cost of the candidate is then the minimal difference over all 21 possible variants. These rigid transformations are applied per vertex and their combination allows for a wide variety of non-linear deformations.

Occluded candidates do not have an apparent unary cost as the appearance constancy for occlusions cannot be evaluated directly from the video frames. In order to select an occluded candidate the unary cost should reflect a certain occlusion penalty cost that allows these candidates to be chosen over non-occluded ones that match poorly. To this end we assign a unary cost of 150% of the previous best non-occluded matching cost to each occluded candidate. Combining occlusion states with template comparison makes the system robust against occlusions that occur frequently e. g., on legs, and offers a solution to the well-known "template update"-problem [Matthews, Ishikawa and Baker, 2004].

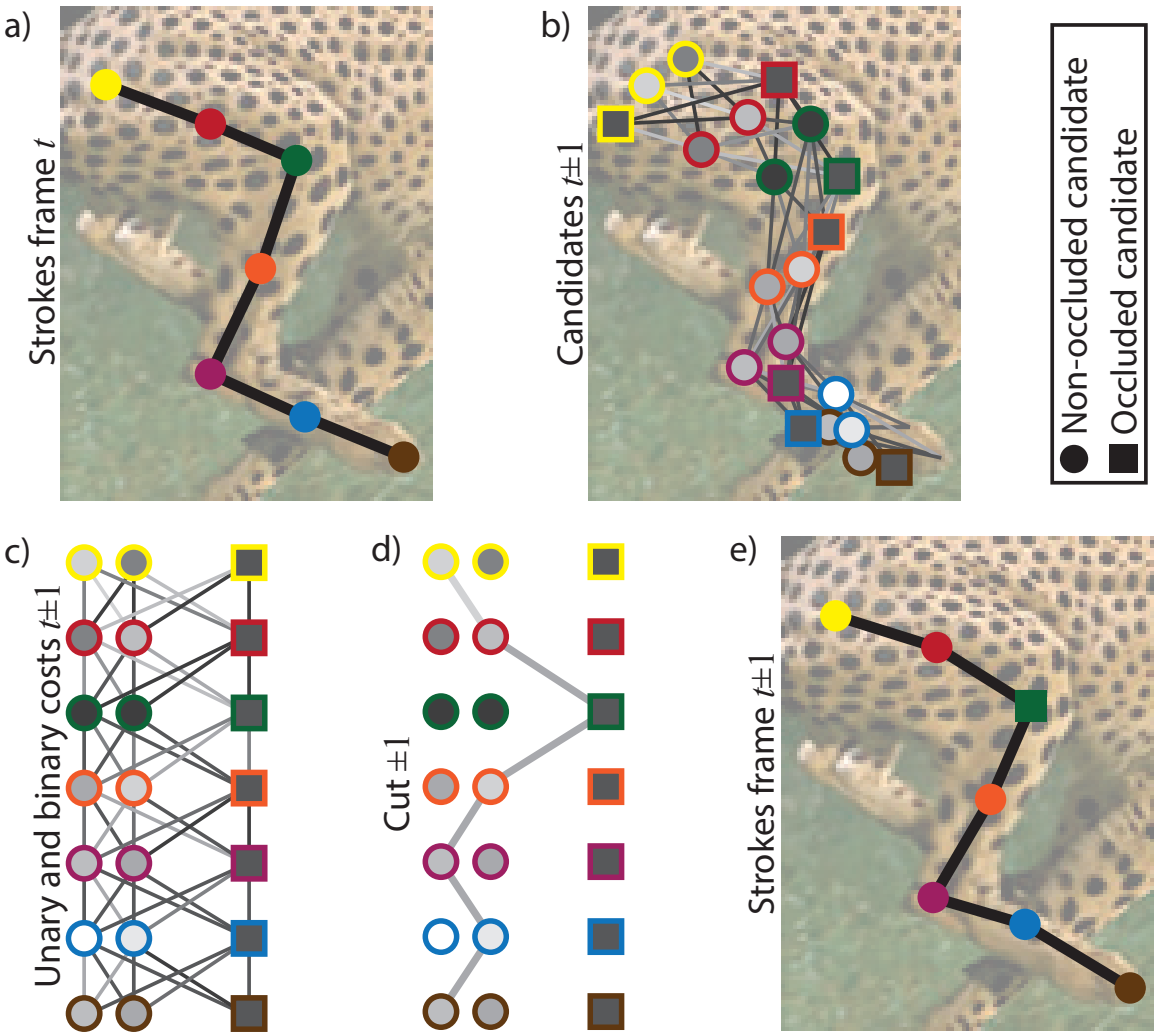

Figure 6.4: Stroke tracking of a stroke from frame t to $t \pm 1$: The solution of frame t initializes candidates for frame $t \pm 1$ *(a and b)*. Non-occluded *(circles)* and occluded *(squares)* candidates are generated for frame $t \pm 1$ *(b and c)*. Costs are visualized as grey values on candidates and on edges for an abstract stroke domain *(b – d)*. The path with lowest cumulative cost is found *(d)* and yields the solution for frame $t \pm 1$ *(e)*. Note that, although being unoccluded, the green marker is chosen from the set of occluded candidates as all non-occluded candidates lead to higher costs.

Binary cost The binary cost (cf. Equation 6.1) consists of two components: *distance* and *direction* preservation (lines in Figure 6.4, b – d), defined as

$$b(\mathbf{x},\mathbf{y}) = \phi(\mathbf{x},\mathbf{y}) + \theta(\mathbf{x},\mathbf{y}).$$

Distance preservation measures the change in length for each candidate pair relative to the original distance in the key frame as

$$\phi(\mathbf{x},\mathbf{y}) = \left| \|\mathbf{x}-\mathbf{y}\|_2 - \|\mathbf{x}^k - \mathbf{y}^k\|_2 \right|,$$

with $\mathbf{x}^k, \mathbf{y}^k$ as the respective vertex position in key frame k. It encodes topology and isometry knowledge put into the system through user interaction and adds user-given semantics not available, e. g., to optical flow. Orientation preservation measures the change in orientation and distance of each candidate pair from the current to the previous frame as

$$\theta(\mathbf{x},\mathbf{y}) = \left\| (\mathbf{x}-\mathbf{y}) - (\mathbf{x}^{f\pm1} - \mathbf{y}^{f\pm1}) \right\|_2,$$

where $\mathbf{x}^{f\pm1}, \mathbf{y}^{f\pm1}$ are the respective vertex positions of the previous frame. It encodes velocity constraints and enforces smooth movements but also penalizes large distance changes between frames.

Candidate selection Candidate selection chooses one candidate out of the occluded and non-occluded candidates for every stroke vertex that globally minimizes the cumulative unary and binary costs of Equation 6.1, i. e., it computes

$$X^{l,f} = \arg\min_{\overline{X}^{l,f} \in C} c(\overline{X}^{l,f}),$$

where C is the set of all possible stroke candidates. Ideally, all combinations of all candidate positions should be evaluated but the sheer amount of candidates makes the optimization prohibitively expensive as the complexity is quadratic in the number of candidates. To enable an efficient optimization we prune the non-occluded candidates down to a count of five constituting the dominant minima of the unary costs. We found that although being an approximation this pruning strategy does not noticeably degrade our result. As distance and direction preservation pose important constraints on our optimization we weight them with a factor of 10 relative to the unary and binary distance cost, i. e., $\lambda = 10$. The optimized result for each frame becomes the initial result governing the candidate generation of the next frame (Figure 6.4, d). Globally optimizing a one-dimensional sequence of labels with unary and binary costs can efficiently be solved using dynamic programming as done e. g., by Buchanan et al. [2006]. In contrast to their approach that regularizes feature tracks over time, we employ dynamic programming to regularize in space, i. e., along our strokes.

In general we do not only want to select candidates for a single stroke but for all strokes of the connection tree in a single frame at once. Without loss of generality we can employ the connection tree to run a global candidate selection on all strokes simultaneously, starting from the tree's leaf vertices up to the root vertex, to get the globally optimal candidate choice.

We apply special treatment to those vertices that are likely to be occluded by other parts. Symmetric limbs flagged to be further away from the viewer (Section 6.2.4) are presumably (partially) occluded by their symmetric counterpart in many frames. Hence our optimization is allowed to only choose from occluded candidates for stroke vertices of these strokes that are closer than 20 pixels away from any other stroke vertex. In some cases these limbs can however be fully occluded in all frames and for such we add the vertex position offsets of the symmetric, inward-facing limb also to the vertex positions of the outward limb.

The spatial connections of the stroke vertices form a Markov chain. Including temporal connections to other frames could allow for global optimization in both, spatial and temporal, directions, resulting in a Markov random field model. As stroke tracking is based on a cost depending not only on the last but all previous frames up to the last key frame candidate selection would become prohibitively expensive. Taking into account all previous frames leads to an excessive number of temporal connections between frames making an optimization of our non-submodular costs too expensive to be optimized with interactive performance.

Blending After all ranges have been tracked, they need to be combined into one consistent sequence. This happens for all forward-backward pairs of ranges between two key frames in isolation. Input to this step are two ranges that overlap in some frames, one from the forward and one backwards in time. Output is a single, consistent track. To preserve motion and intrinsic distances, we blend the orientations of each limb stroke segment relative to its parent using spherical linear interpolation (SLERP) [Shoemake, 1985] with a linear interpolation of the root vertex position. The weight given to a range at each vertex at each point in time decreases with the frame distance to the respective key frame. An alternative way of blending between the two frame ranges could incorporate texture information, i. e., using the unary costs of Equation 6.1. However, the resulting weights would not necessarily be smooth over time and could lead to noticeable jumps in the motion. Computing matching-based weights in a temporally smooth way, hence, remains future work.

6.2.6 Segmentation

As now all strokes are known in all frames, we next segment the foreground of the filtered input video and, in a second step, label the foreground such that every pixel belongs to exactly one stroke.

Foreground segmentation We employ cost-volume filtering [Rhemann et al., 2011] using the previously generated space-time strokes as input. To this end we build two (soft) foreground RGB histograms with 50^3 bins (Figure 6.5, b): one of all pixel colors of the stroke vertices in all frames of the filtered video (colored strokes in Figure 6.5, a) and one build from a set of background vertices in all frames (white stroke in Figure 6.5, a). The background stroke is found as vertices of a slightly offset bounding box of the limb strokes. Softness is achieved using a Gaussian blur with a σ of 2. The cost volume is blurred (we

typically use 2×2 pixels for our resampled video) in space and a binary foreground mask (Figure 6.5, c) is extracted with a threshold at 0.75. We use this slightly higher threshold to allow for larger segmentations that can later be consolidated by the cylinder fitting step. To improve segmentation stability for tracking errors, the last 20 % of the stroke vertices on each terminal limb stroke are skipped. A terminal limb stroke is a stroke with no further child strokes in the connection tree. This helps, as such strokes are susceptible to be drawn too long or are occluded near the ground, e. g., by the grass in Figure 6.1, or can be difficult to segment due to contact shadows. As the user is required to draw the full skeleton in the first key frame the initial histograms resemble a valid color distribution of the object.

Stroke segmentation Each pixel of the foreground segmentation does not belong to all limbs and drawing radius samples from this segmentation directly prevalently leads to overestimation. Hence, in a second step the foreground is partitioned such that every foreground pixel belongs to exactly one stroke. This second segmentation step is done independently in all frames. For every stroke vertex we march the image orthogonal to the stroke until we leave the foreground mask or the image. For every pixel visited during the walk we keep a reference to the closest stroke and draw a line from the stroke to the radius sample with a depth value corresponding to this distance (Figure 6.5, d). This can be done efficiently on a GPU using splatting of sufficiently thickened stroke IDs and z-buffering of distances. Note that this is slightly different from a generalized Voronoi decomposition of the skeleton in terms of Voronoi sites as distances are computed strictly orthogonal to the limb strokes, the marching direction of the cylinders.

6.2.7 Cylinder fitting

Cylinder fitting takes as input the dense, two-dimensional space-time strokes as well as the stroke segmentation. Output is one deforming generalized three-dimensional cylinder for each limb stroke represented as a one-dimensional table of three-dimensional positions (the *path function*), as well as a two-dimensional table of radii for every direction at all stroke positions (the *radius function*). The radius function (Section 6.2.7) and three-dimensional direction (Section 6.2.7) are found independently (Figure 6.6). Finding the correct radius in a single frame can fail due to self-occlusions, especially in the vicinity of limb joints, and segmentation errors. Hence, consolidation of segmentation results of all frames is used to compute consistent radii.

Radius fitting

To account for incautiously drawn terminal limbs of creatures, such as legs or heads, we initially extend every stroke along its start and end direction in two dimensions by an additional 10 % before all further processing. This approach potentially creates samples outside of the segmentation, but the final validation step eliminates invalid samples.

Next, the radius function at every vertex is found independently (Figure 6.6, a and b) before combining them into consistent radius samples along the stroke (Figure 6.6, c). We will now

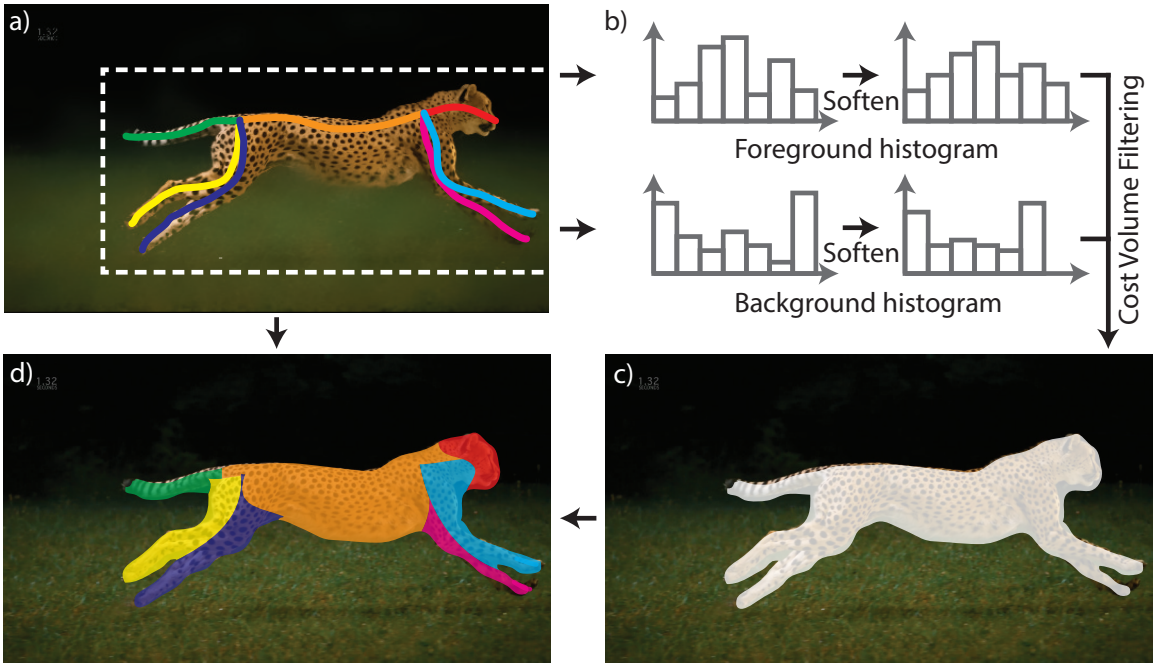

Figure 6.5: Segmentation. *a)*: The foreground *(colored lines)* and background *(white, stippled line)* strokes are shown on top of the filtered video. *b)*: The resulting foregound and background color histograms are built and smoothed, *c)*: and used for cost-volume filtering [Rhemann et al., 2011]. *d)*: Stroke segmentation separates the segmentation results into a single, disjoint segmentation mask per stroke and frame.

describe this procedure for a single vertex. In the work of Chen et al. [2013b] the user has to specify the radii explicitly; in contrast our method automatically combines segmentation results of all video frames to fit consolidated radii. While our approach could benefit from explicitly drawn radii it would make the interface more complicated and especially at limb joints specifying the correct radius is fairly difficult and requires careful drawing. Hence, correcting false radii using additional strokes remains future work.

Marching The full radius function is computed from a *radius sample pair*: One radius in the normal direction of the limb stroke and one in the opposite one. In a subsequent step, these many sample pairs are converted into a full 360-degree radius function. We will describe this procedure for the radius samples drawn from the normal direction without loss of generality, both are performed equally.

Starting from a stroke vertex position we march orthogonal to the stroke direction [Chen et al., 2013b] until we reach a pixel not belonging to the foreground. Each such walk results in a vote for a radius. If on the way we encounter a part belonging to a different stroke we reject this sample. Additionally, radius samples at image boundaries or smaller than a threshold of 3 pixels are rejected. Drawing radii in all frames results in a distribution of different two-dimensional radii votes: one, potentially rejected vote per frame. To obtain a robust radius estimation for every vertex from this distribution, we use the robust 25 %-percentile of the accepted radius samples for the vertex itself, its four neighbor vertices along the stroke

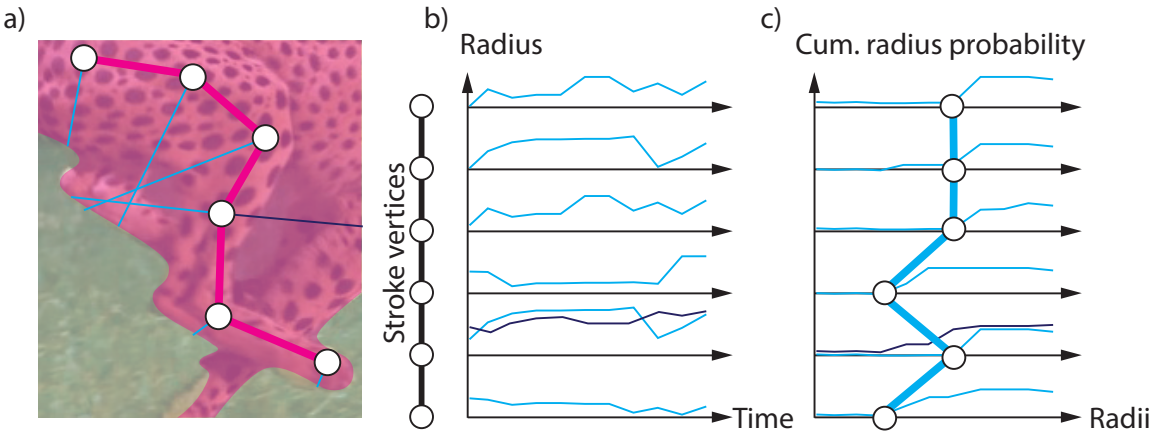

Figure 6.6: Cylinder fitting: For every stroke vertex *(circle)* in each frame orthogonal marching finds the closest background pixel which is then counted as a vote for a radius *(blue line)*. Each frame produces a different radius vote over time, here shown as a radius function of each stroke vertex. Taking a robust minimum finds the effective radius for every stroke vertex, producing a radius function along the stroke *(thick blue vertical line)*. In practice, two radius functions are computed for each vertex *(only shown for one vertex)*.

and vertices of the symmetric limb, if applicable. Percentiles help to resolve false radius samples due to overlapping limbs or erroneous segmentation. If too many radius samples are rejected per vertex (more than 90%), we flag the radius as being invalid. This happens if a limb stroke is stuck inside the body of a creature or if segmentation fails permanently. Assigning a radius for these vertices is left to filtering and in-painting along the stroke.

Filtering and in-painting Next, invalid radius samples are in-painted from nearby valid ones. Also, the radius function is slightly blurred with a σ of 10 % of the total stroke length accounting for noise in the percentile segmentation along the stroke (Figure 6.7, b).

Capping All cylinder ends that are connections between limbs are capped, i. e., the radii at the start and end are blended to 0 in an interval of 20 samples around the ends.

Validation In-painting, blurring, and capping modifies the radius samples, resulting in cylinders that potentially fall outside of the segmentation. Also the robust percentile is prone to a slight overestimation of the correct radius. However, we can be more sure about radii in the key frames. Therefore, we iterate over all key frame foreground masks, and make sure no vertex falls outside this mask in any key frame (Figure 6.7, c).

Densification Finally, the radius pair is converted into a dense radius sample function for all directions at every stroke vertex by fitting an ellipse (Figure 6.8, a). Simply using the average of the normal and counter-normal radii (pink and blue) for both, major and minor, ellipse radii (green and red), yields a circular cross-section (Figure 6.8, b). For many shapes

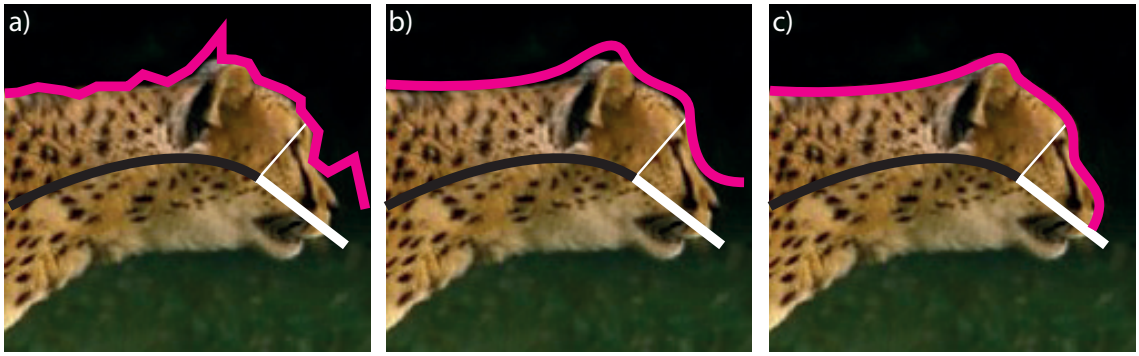

Figure 6.7: *a)*: Estimating the radius pair at every stroke vertex can be noisy and has errors such as at the ends, which are not part of the limb strokes, but added automatically *(white part)*. *b)*: Smoothing is used to reduce noise. Outliers are detected and in-painted along the stroke. *c)*: Finally, the radius function is fitted to the mask in one reference frame to refine the shape, in particular at the end.

such as an animal torso, where the spine is located rather at the border of the cylinders, a compressed ellipse provides a better shape approximation. Thus, we multiply the minor ellipse radius with the ratio of the smaller of the two normal radii divided by the larger one (Figure 6.8, c). As this leads to flat cross sections in case of close to zero minimum normal radii, we average both strategies to obtain our final minor ellipse radius: half the average and half the ratio (Figure 6.8, d).

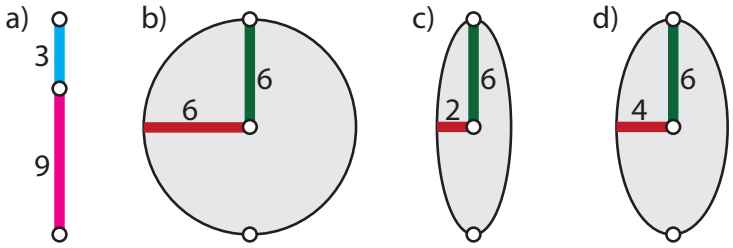

Figure 6.8: Ellipse densification. The final radius function *(d)* is found as the mean radius function of the average *(b)* and a scaled radius function *(c)* of the two radius samples *(a)*.

Path fitting

The x- and y-coordinate of the generalized cylinder path function result from the stroke tracking step (Section 6.2.5). Assuming approximately constant object-to-camera distance throughout the entire video sequence, these coordinates immediately constitute the x- and y-coordinates for each stroke vertex. The missing z-component is found using the kinematic information as follows (cf. Figure 6.9): We detect limb strokes that leave the image plane with the aid of their projected lengths. A limb stroke with a substantially shorter projected length (50 % of its maximum length) likely has rotated outwards of the image plane. For

such frames we use inverse perspective for the known three-dimensional length to compute its out-of-plane rotation. To obtain the final z-values we use a polynomial of degree two that interpolates the start position as well as its gradient and exhibits the correct three-dimensional length. To determine if we have an increasing or decreasing z-component we use the user-defined limb direction flag. A temporal smoothness assumption allows propagating this flag from the sparse set of key frames to all frames: When a limb moves smoothly from an in-plane direction to an out-of plane direction facing the viewer, it is assumed to continue facing the viewer. Finally, limbs that are found to be symmetric get a z-offset depending on their thickness. For this we calculate the average thickness of the lower 50 % of the limbs and use this offset with a factor of 4 to move the limbs in positive or negative z-direction respectively. The previously mentioned limb direction decides on the sign of this offset. This reliably offsets the limbs of animals at hip- or shoulder-type joints and appears to be a suitable heuristic for the animals used in this chapter.

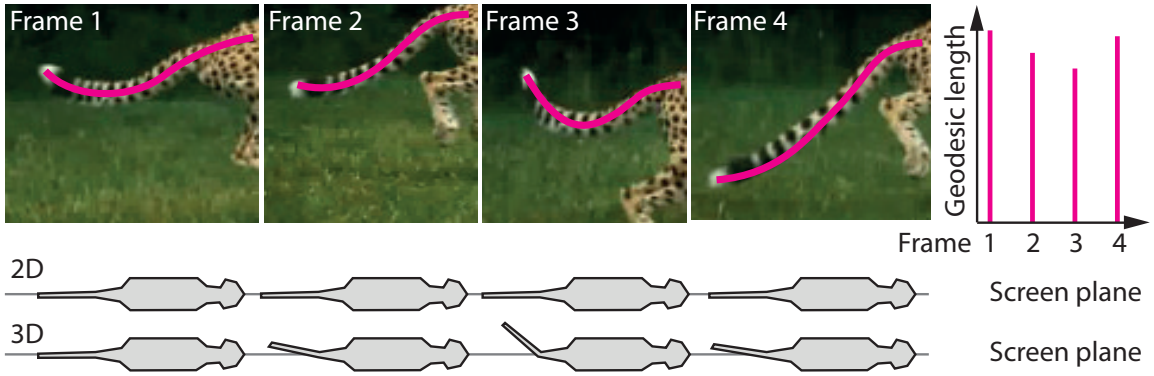

Figure 6.9: The length of each stroke in image space varies for strokes that change their three-dimensional direction such as the tail. To the right we plot the geodesic length for all four frames. We use this information to add three-dimensional out-of-plane rotation of strokes to the two-dimensional shape *(bottom)*.

6.2.8 Texturing

Texturing uses the generalized cylinders as well as the video sequence. It outputs a cylindrical texture for each limb cylinder. Blending textures from different frames has shown to be counter-productive due to slight but apparent shading and deformation changes. Hence, we chose a single frame of the video to obtain the texture for all limbs. We heuristically always pick the first key frame as it contains all limbs by definition. To obtain the texture for each cylinder we project the cylinders vertices into the video frame and calculate a binary reliability. Unreliable pixels are those that fall outside of the foreground segmentation, pixels at grazing angles of the cylinders, and pixels marked as being occluded. These unreliable pixels are filled with the closest reliable pixels using push-pull in-painting [Gortler et al., 1996]. To allow for wrap-around of the in-painting, it is performed on an unwrapped cylinder that respects a toroidal parametrization. Occluded limbs use the texture of their symmetric, non-occluded counterpart for texturing. Texturing uses frames from the original video in full resolution to produce textures with maximum detail.

6.2.9 Implementation

To ensure interactive performance, most steps are implemented using parallel graphics hardware. Candidate selection of the stroke tracking governs the complexity of our algorithm with a running time of 20 to 400 ms per frame depending on the skeleton complexity. All other steps can be parallelized over all stroke vertices and strokes in all frames resulting in a running time of less than 5 ms. The initial tracking of a typical skeleton in a video sequence of 64 frames can be processed in 20 to 30 seconds, whereas the number of frames to be tracked decreases for subsequent key frames. Each video was processed in under 5 minutes. The resulting mesh can be rendered and animated at several hundred frames per second.

6.3 Results

We report qualitative results in form of three-dimensional mesh animations from videos that allow for certain applications as well as quantitative results in terms of a user study, measurement of reconstruction error for synthetic scenes, and a comparison to previous work. On average 4.39% of all strokes in all frames were drawn to reconstruct the sequences; further stroke statistics can be seen in Table 6.1.

Table 6.1: Stroke statistics. Stroke percentage is the ratio between the number of drawn strokes to the number of all strokes in all frames.

Sequence	Frames	Key frames	Strokes	Stroke percentage
Cheetah	64	4	19	4.24 %
Dog	38	3	11	4.13 %
Elephant	45	7	19	4.92 %
Greyhound	88	5	19	3.08 %
Kangaroo	65	4	17	3.26 %
Zebra	47	4	12	4.25 %
Horse	60	5	14	3.33 %
Snake	49	3	3	6.12 %
Camel	60	5	16	3.81 %
Giraffe	29	4	16	6.80 %
Average	54.5	4.4	14.6	4.39 %

Qualitative results Our primary results are three-dimensional mesh animations from skeleton sketches (Figure 6.10 and Figure 6.11). This allows for typical applications such as texture transfer (Figure 6.12), re-posing (Figure 6.13), and cloning (Figure 6.14).

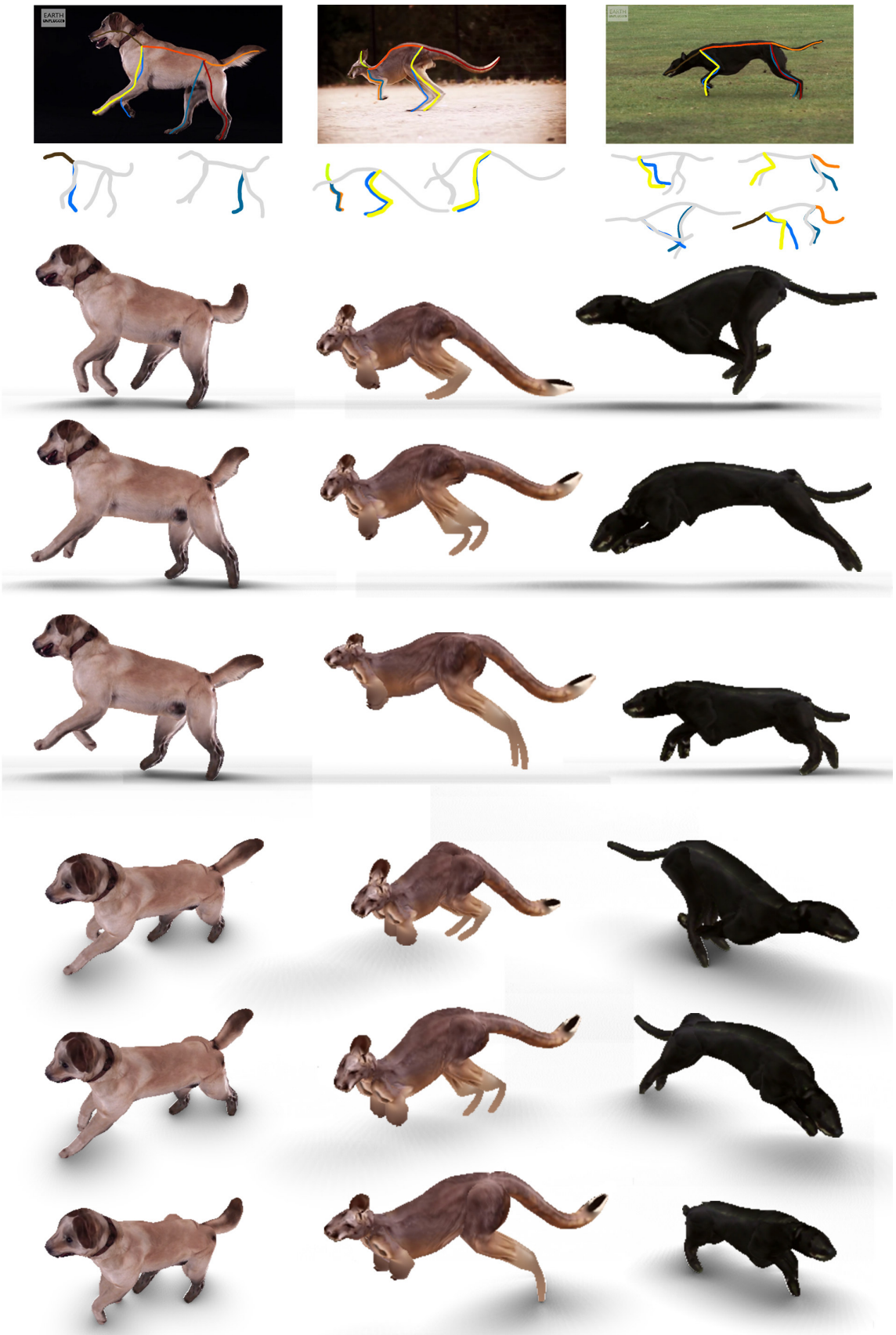

Figure 6.10: Results for different video sequences *(columns)* and their key frame strokes (*colored strokes*). We always show the animated three-dimensional surface at the frame of the video from one view similar to the video and a different one. The total sketching time for such input is below 5 minutes.

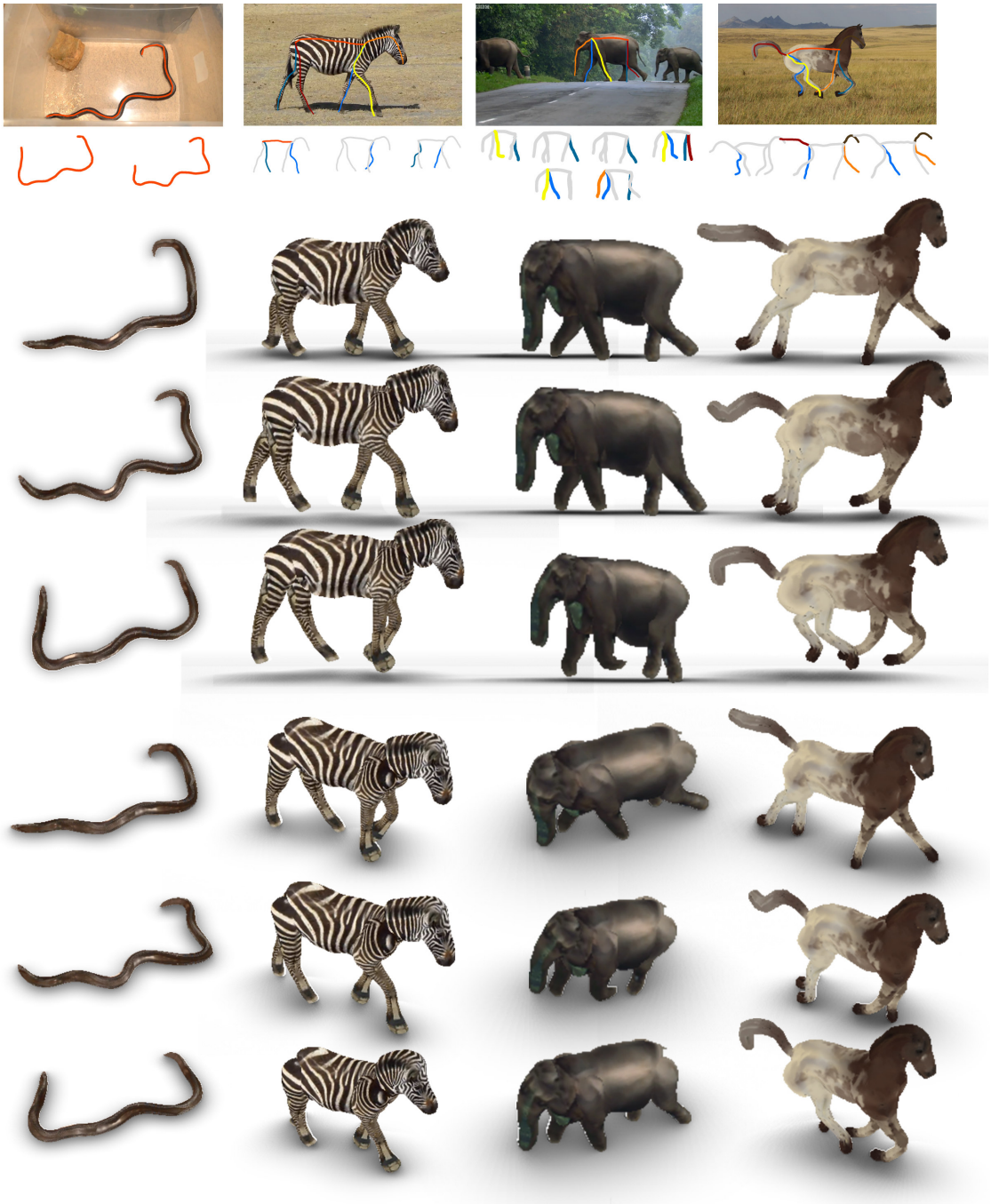

Figure 6.11: Results for different video sequences *(columns)* and their key frame strokes (*colored strokes*). We always show the animated three-dimensional surface at the frame of the video from one view similar to the video and a different one. The total sketching time for such input is below 5 minutes.

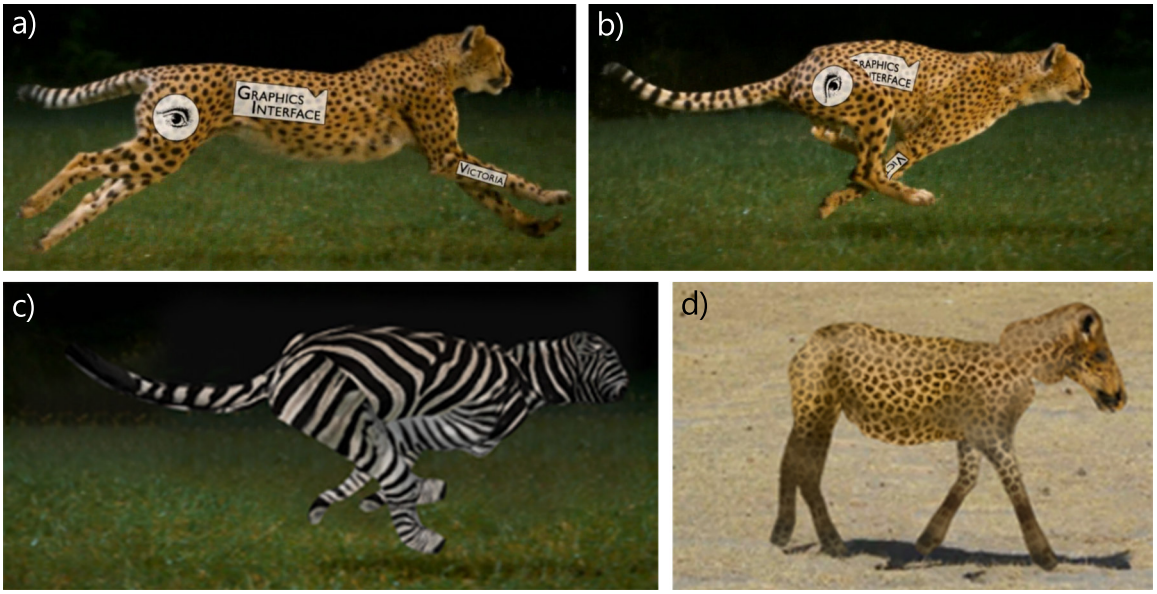

Figure 6.12: Transfer of an overlay texture from one frame *(a)* to another frame *(b)*. Please note how occlusions are realistically handled. *c, d):* Transfers of entirely different textures.

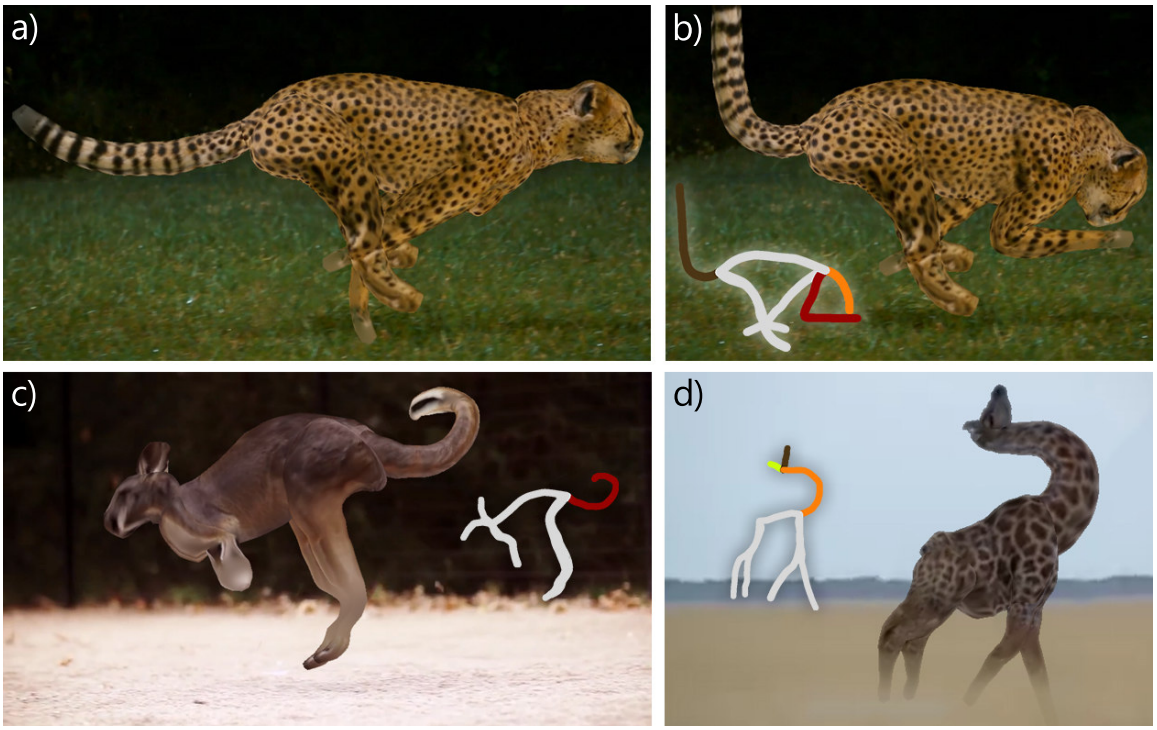

Figure 6.13: *a):* Original frame. *b, c, d):* Posed results using the shown colored strokes. Please not the Kangaroo's tail leaving the image plane as it was drawn shorter *(c)*.

Our output meshes allow to be fed directly to three-dimensional printers such as the MakerBot Replicator 2 (Figure 6.15).

While the quality of our animated meshes does not meet high-quality production needs, we think that they suffice for most casual applications. To obtain high-quality meshes our

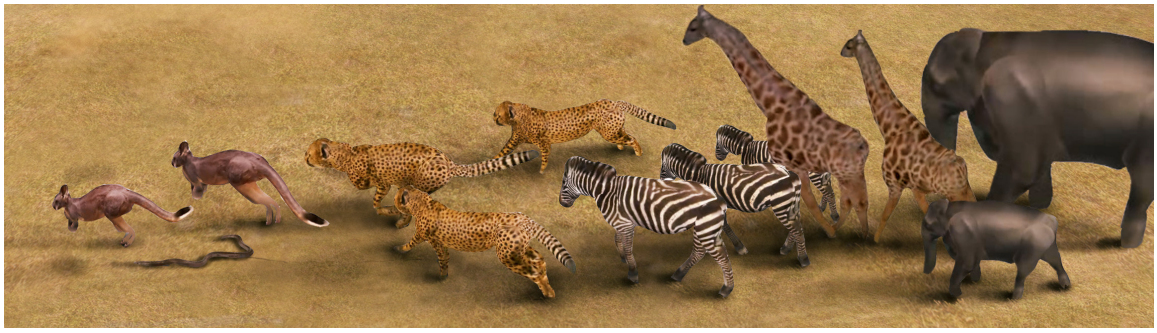

Figure 6.14: Creature cloning (cf. Figure 6.1, Figure 6.10, and Figure 6.11) in new views and new poses.

results can be used as an initial solution for further automatic and manual mesh processing, offering a valuable source for which the vast efforts of tracking as well as shape fitting have already been carried out and only small corrections and shape details need to be added. The methods of Ji et al. [Ji, Liu and Wang, 2010] or Borosan et al. [Borosán et al., 2012] could provide means to generate clean, joint meshes from our parametrization. Our parametrization is flexible enough to propagate changes on the mesh in one frame to all other frames immediately.

Figure 6.15: Three-dimensional printing results of the models from Figure 6.1 and Figure 6.16.

Reconstruction error We rendered animated, three-dimensional meshes of a camel and a horse into a video (Figure 6.16), painted strokes on it, let the system reconstruct the three-dimensional mesh animation, and compared the result to the input in terms of time-averaged intersection-over-union (IoU) in two dimensions with a resolution of 1150×1000 and in three dimensions using a voxelization of size 64^3. The two dimensional and three-dimensional IoU are 63.53% and 85.85% for the horse as well as 58.31% and 83.95% for the camel, respectively. Additionally, we visualized the accumulated error for every vertex of the reconstructed mesh by averaging the distance transform of the rasterization for the two-dimensional and the voxelization for the three-dimensional case. It can be seen that the two-dimensional error is relatively low except for one part of the camel where the radius

was wrongly estimated due to consistent segmentation errors and at the occluded limbs due to imperfect tracking. The three-dimensional error at all limbs is relatively high due to our limb z-offset-heuristic. We found the resulting three-dimensional object to be very similar to the input mesh, even in new views. Some geometric details such as the ears of the camel or the animals' feet details could not be reproduced, as they constitute geometry outside of the scope of generalized cylinders.

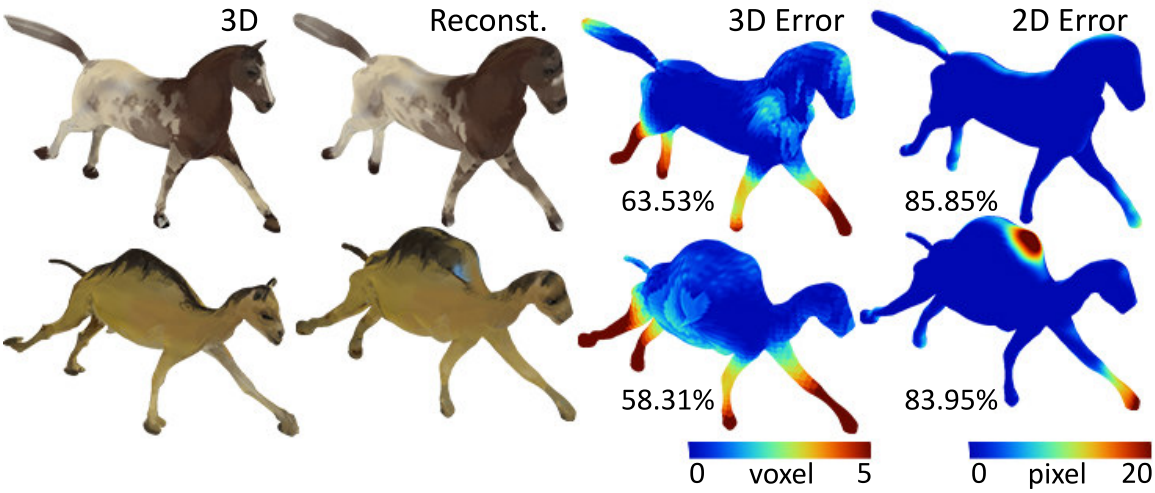

Figure 6.16: We rendered an animated three-dimensional mesh of a horse *(top)* and a camel *(bottom)* to a video, reconstructed the three-dimensional mesh with our system *(middle)*, and compared the outcome to the input using different metrics *(right)*.

User study To evaluate the usefulness of our system we asked nine novice users with previous experience in computer graphics and media production on different levels to use our system and reproduce the results of the first sequence in Figure 6.10. After a short introduction to the system using the sequence in Figure 6.1 the users had at most five minutes to reconstruct the sequence, but most users were satisfied before (3:43 minutes on average). The users used on average 9.5 strokes to reconstruct the sequence which is in agreement to the stroke count used by the authors (cf. Table 6.1). We would argue it does not need a formal control group to see that without substantial training users are not capable to produce models of this quality using any available modeling and animation software such as Blender [Blender Online Community, 2016]. Users were able to produce animations of a quality similar to the animation produced by the authors, indicating the system can be used by non-experts.

Comparison Our stroke tracking approach relies on appearance and regularization terms commonly used in extensively-investigated optical flow methods. An interesting question could be if methods designed to solve a much more general problem already suffice to solve the tracking problem at hand. To this end, we compared our limb stroke tracking method on a running cheetah sequence of 20 frames to state-of-the-art optical flow methods: SimpleFlow [Tao et al., 2012], Brox et al. [2004], TVL1 [Sanchez Perez, Meinhardt-Llopis

and Facciolo, 2013] , and a ground truth for which all strokes were painted manually for all frames.

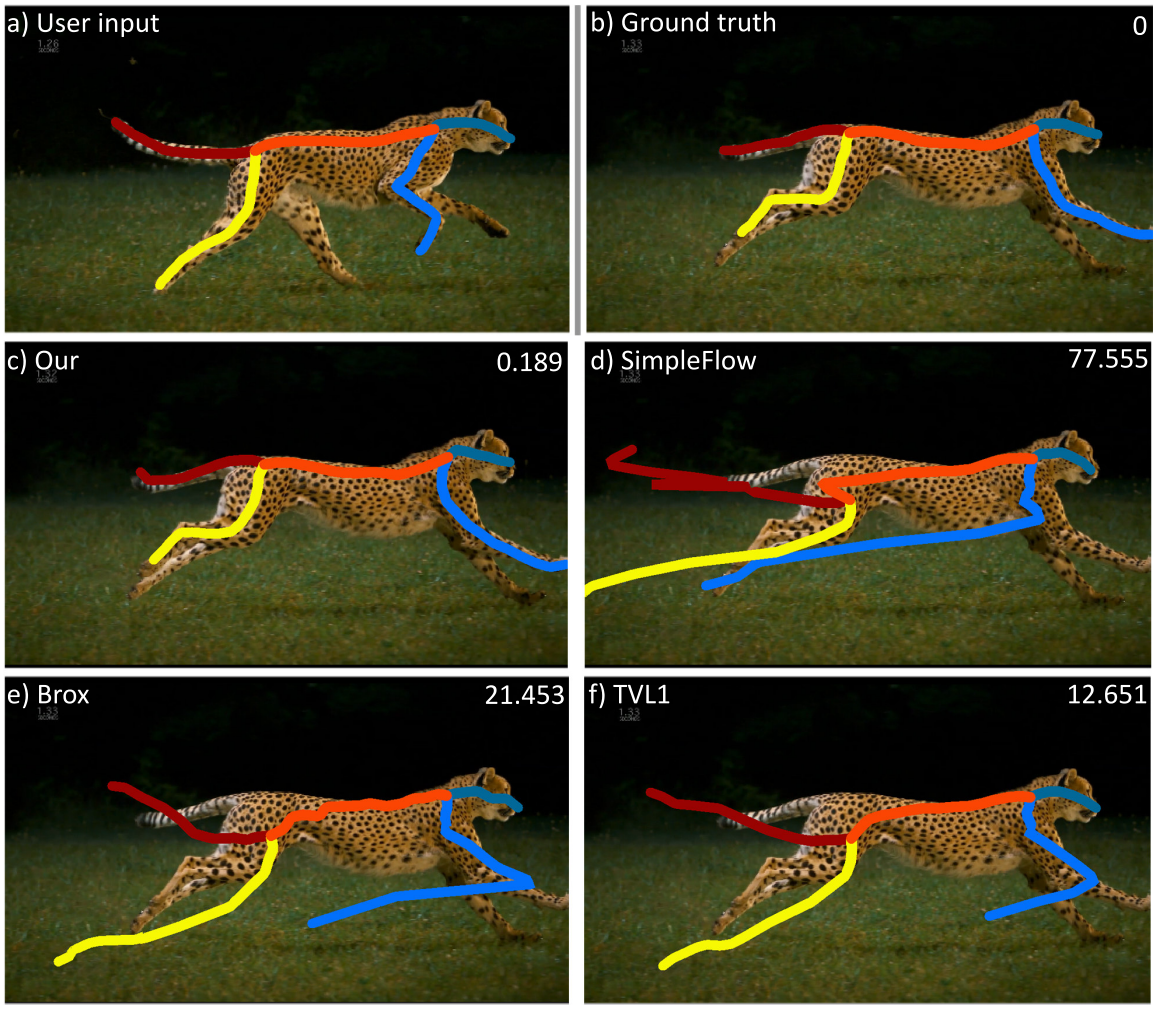

Figure 6.17: Stroke tracking comparison using different optical flow methods and our tracking approach. Our approach outperforms all previous methods.

We optimized the parameters for each method by hand and only report the best results obtained. The strokes are tracked back and forth starting from the same, single key frame for all methods. We use the optical flow offset vectors to track the strokes by accumulating them for every vertex position in every frame. As optical flow generally does not produce offset vectors for occlusions, only strokes that were completely visible in all frames are used for tracking. An error measure is computed as the sum of absolute distances to the ground truth vertex positions. The accumulated error over all frames is 77.6 for SimpleFlow, 21.4 for Brox et al., 12.6 for TVL1, and 0.189 for our approach. Optical flow works well for many vertices in many frames, but the sheer number of vertices and frames will quickly result in a disintegrating skeleton. We believe that the main reason for the limited performance of the investigated optical flow methods is the accumulation of small matching errors. As these methods only compute optical flow between neighboring frames this error accumulates rapidly over longer frame ranges. One way to overcome this issue could be to compute

optical flow not only between neighboring frames but over longer frame ranges. In practice however it is unclear how to choose these ranges to stabilize the tracking on the one hand and allow for limb deformations on the other hand. Optical flow could be used as a prior in our stroke tracking algorithm but would require additional, expensive preprocessing steps and hence remains future work. We conclude that optical flow, which successfully performs a much more general task, might not be the best tool for solving the skeletal correspondence challenge we addressed using our user interface. Additionally, our algorithm allows for interactive performance while the optical flow methods employed, use up to seconds or even minutes to be computed per frame.

To the best of our knowledge no existing template skeleton tracking approach works exclusively on strokes and single-view video, rendering a comparison to these methods unfair and requiring a lot of manual effort.

6.4 Scope and Limitations

Our system is putting the user in the loop to solve an otherwise heavily under-constrained and hard vision problem. We apply several assumptions that might not hold for certain inputs.

Stroke tracking fails if there are not enough features to be tracked on the object or in the presence of severe occlusions. Occluded parts can be mistaken to be non-occluded, in which case tracking starts using erroneous templates for matching. The regularization terms in Equation 6.1 however lead to graceful failures that can be resolved by additional user interaction.

Next, segmentation fails if the fore- and background histograms are too similar and salient edges separating the object from its background are missing. Our method is particularly designed to allow tracking of creatures with arbitrary skeletons for which template models might not be available. While our method might also be applicable to humans, the large body of research on this specific topic leads to results that most likely outperform our method.

Our model allows to track limbs of arbitrary complexity and abstracts model knowledge, but a more rigid model could be beneficial enabling a more realistic motion extraction. We have demonstrated reproduction of sequences where the creature as a whole exhibits approximately image-plane-parallel motion. While our approach is capable of handling out-of-plane motion for isolated limbs, such as tails or trunks, as well as limited camera motion and zooming, support for arbitrary camera motion is missing and constitutes the strongest limitation of our algorithm. Surprisingly however, many videos found in online video platforms depicting animal locomotion satisfied our assumptions, enabling the reconstruction of a wide variety of creatures.

Handling arbitrary camera motion would not only enable support for a larger variety of videos but the extra views add additional knowledge about the true three-dimensional shape of the object to the system. Exploiting these cases in our algorithms remains the most

promising direction for future work. Shape reconstruction using generalized cylinders is a valid assumption in many cases, but fails in case of certain cylindrical cross sections, such as for bodies and fins of fish, or complicated branching structures. Again, in these cases, multiple views can help to find the true shape of the limbs. Nevertheless, many creatures look plausible in views substantially different from the side view (Figure 6.14).

Animals are usually composed of several rigid bones. While on the one hand our approach abstracts this concept away from the user it also leads to a certain non-rigidness on the other hand, where more explicit assumptions could lead to more sophisticated results.

Chapter 7

Conclusion

In Section 7.1, we summarize the major contributions following the four main tracks: Homunculus Warping, Interactive By-example Design of Artistic Packing Layouts, Projective Blue-Noise Sampling, and Animated 3D Creatures from Single-View Video by Skeletal Sketching. Section 7.2 discusses possible future work before Section 7.3 states the take-home message of this thesis.

7.1 Closing Remarks

Synthesis and manipulation of visual media commonly poses a tedious and time-consuming task demanding in-depth knowledge, extensive training, and high-end hardware to be performed effectively. Common tools to accomplish this goal involve recurring interactions cycles and parameter tuning until a satisfying result is achieved. Realization of the desired outcome inevitably depends on the users' intentions and hence keeping the user in the loop and motivated is essential. What is more, without further knowledge many synthesis tasks are heavily under-constrained and only become feasible with the input of the user. Inverse, example-based approaches alleviate these problems by estimating the underlying parameters governing the manipulation and synthesis from a small subset of examples. Making these techniques efficient and accessible is the goal of this thesis. The techniques presented contribute to different kinds of media that needs to be synthesized or manipulated. We exemplify several techniques to create a full system that enables generation and manipulation of images, videos, and three-dimensional models. All parts serve as building blocks of a full system that allows generating visual media. The goals formulated in Section 1.1 were realized as follows: *Intuitiveness* was achieved by leveraging visceral, widget-free user interfaces that build on common interaction techniques such as sketching. We validated our approaches with user studies supporting our arguments. *Plausibility* was accomplished by including informed model knowledge of the specific tasks into the system to constrain the solution space to only plausible solutions. *Speed* was accomplished by reformulation

of sequential tasks as parallel problems that map well to the GPU and allow for efficient, parallel execution.

In Chapter 3 we presented an approach to change the local scale of a surface to convey importance. The resulting shapes are free of self-intersections and avoid distortions. Compared to other visualizations of scalar (importance) fields, size changes are immediately understood by the users and could be employed to illustrate scalar information on complex domains even for naïve observers. This is because a twice-as-large object (cf. e. g., the hands in Figure 3.1) is naturally perceived as being twice as important, whereas there is no color that represents being twice as much as another color. Additionally, unlike colors that are restricted to fixed bounds, there are no limits to size changes. Conversely however, color-coded scalar fields are superior to our approach when qualitative comparisons of locations on the domain are required e. g., finding all locations with the same color label.

Chapter 4 introduced an approach to compile artistic primitive layouts. Technically, we stated the problem as a projection from the space of visual and semantic features into a lower-dimensional layout space combined with a GPU-based relaxation to achieve an equal distance between primitive boundaries. The parameters of this projection are learned from user feedback. Reponses received from artists as well as both parts of our user study show that producing packing layouts is challenging and benefits from computational support. Specifying the underlying parameters by hand involves selection of a layout function (e. g., linear layout), the feature dimensions (e. g., axis direction), a feature mapping (e. g., size to map to the first axis), a feature scaling factor, and an adjustment of the specified layout to the user constraints. All these parameters are automatically inferred by our system. Our widget-free interface cannot replace professional data analysis but augment existing dialogs or serve other purposes. It could, however, be used by novice users, e. g., children in a museum: Such users might not be willing to participate in a formal user dialog involving concepts like “sorting”. However, they might play around and manipulate butterflies and get insights from how our system reacts, without a particular goal in mind, maybe even without knowing they perform man-machine interaction in the first place.

In Chapter 5 we proposed a simple extension to blue-noise sampling that produces patterns with desirable spectral properties when projected onto lower-dimensional subspaces. Our extension is easy to implement and likely orthogonal to many ways of generating patterns, as we demonstrated for two popular methods – dart throwing and Lloyd relaxation. For Monte Carlo rendering we showed that the resulting patterns outperform classic blue-noise patterns and are on par with or better than low-discrepancy patterns which have specifically been designed for the purpose of numerical integration. For primitive placement we showed arrangements with uniform distribution in both three-dimensional space and multiple two-dimensional viewing projections. We believe our approach is a step towards a single, universal, multi-dimensional pattern with a wide range of applications.

Chapter 6 demonstrates a system to acquire mesh animations from sketches in videos. The system uses limb strokes, which we believe are the most basic imaginable input that every casual user knows and produces results at interactive frame rates, allowing to quickly refine the solution until the desired quality is achieved. Previous work on sketch-based acquisition was limited to a single or multiple images of a rigid scene or to specific classes

of animations such as humans. Capturing geometry in multiple frames does not only allow for animated surfaces, but can also resolve ambiguities that could not be eliminated using a single frame. The increase in animated surface quality when adding more images due to new views, new deformations, or both can be inspected visually (cf. Figure 6.10 and Figure 6.11) and quantified by reconstructing rendered images of three-dimensional models with known reference geometry (cf. Figure 6.16).

7.2 Future Work

In the following sections we discuss potential avenues of future research for each component separately (Section 7.2.1) as well as for combinations of the components (Section 7.2.2) and provide a general outlook to promising, open problems (Section 7.2.3).

7.2.1 Individual Future Work

Homunculus Warping The current system's performance is limited by the offline optimization of the collision-free result defying interactive use of the full design pipeline. Hence, increasing its performance is an interesting direction for future work allowing fully interactive focus+context visualizations while abolishing the necessity of a preview. Human perception of area, weight, size, scale, and its dependence on spatial organization of the underlying domain most definitely has an influence on our approach but yet remains to be modeled computationally. Avoiding distortion of salient features such as faces (Figure 3.1) or symmetry is a promising direction for future work. Additionally, considering the articulation by limiting the degrees of freedom of the fundamental distortions could lead to more sophisticated and realistically looking model deformations. However, this would require input meshes with additional information such as skeleton data, defying the usage with most off-the-shelf models. Different applications of the deformations, such as computational body-building [Saito, Zhou and Kavan, 2015] or shape registration, and its implications on the optimization could be explored in further detail. Finally, a perceptual study could lead to a more formal understanding of human task performance when using size versus color-coded information.

Interactive By-example Design of Artistic Packing Layouts The developed system faces the same challenge as other systems making suggestions to users as it might provide undesired results. Mitigating this e. g., by displaying different suggestions is a promising direction for future work. We would like to complement the system by active learning: If a user organizes bright small eggs (cf. Figure 4.1) along the horizontal line, the system could ask if the feature that was meant was "bright" or "small" or both by displaying different propositions. Additionally, only linear arrangements are considered in this work, while there are other methods to reveal relations between the primitives. The system is not yet capable to infer higher-level layouts, such as clustered (e. g., Figure 4.2, d), symmetric (e. g., Figure 4.2, a), or tree-like layouts (e. g., Figure 4.2, c), which would require a more

sophisticated search for the layout function ϕ (Equation 4.1). Using a general combination of different features for the layout, instead of only two distinct features, also remains future work. Generalizing the two-dimensional layouts to primitives of higher dimensionality could spark a wide range of applications. In the current implementation, our approach only operates on bounded domains, but layouts for toroidal domains that tile infinitely in all directions could also be considered. Further, additional degrees of freedom such as rotations, (non-uniform) scaling, affine transformations, or general deformations could be added to the relaxation to facilitate tighter packings. Aside from positional constraints other constraint types, such as how well an individual, single shape matches the local container shape, could be considered [Xu et al., 2014]. Finally, other applications that could benefit from our interactive application such as specialized tilings and floor plan layouts [Peng, Yang and Wonka, 2014] could be investigated further.

Projective Blue-Noise Sampling Extending other sampling algorithms, e. g., capacity-constrained Lloyd relaxation [Balzer, Schlömer and Deussen, 2009] or farthest point optimization [Eldar et al., 1997; Schlömer, Heck and Deussen, 2011], to projective blue-noise and non-linear projections is a pertinent direction of future work. Although not being a limitation of our theoretical formulation, our system is currently constrained to rectangular domains and consequently only allows for (optionally rotated) axis-aligned projections. Generally, support for more elementary intervals of the (t,m,s)-*nets* (cf. Section 2.3.1) could lead to improved discrepancy values and in turn better rendering performance. Extensions to non-orthogonal and non-linear projections are an interesting avenue for further research requiring a more in-depth analysis of how projections into such domains and a toroidal tiling are reciprocally influencing each other. Similar to the packing layouts, additional applications such as floor design and object tilings pose another potential avenue for future work.

Animated 3D Creatures from Single-View Video by Skeletal Sketching The most notable limitation of the current system presumably is the assumption of a static camera (w.r.t. the creature). Although our stable tracking approach relaxed this requirement to a certain degree, the algorithm does not support strong camera motions including observations of the object from drastically different views. Besides this limitation, relaxing other limitations detailed in Section 6.4 remains viable future work towards a unified system to extract arbitrary deformable geometry from video. Additionally, future work could include automatic detection of repetitive motion from a few sketches and a more advanced method for symmetry and orientation detection for ambiguous postures. Further, we would like to relax the requirement of a skeletal model to generalized sketching of animated surface reconstruction hints that specify length, direction, area, volume, fluid flow, compression, torque, and other properties on top of video frames. In combination with a more involved surface estimation, e. g., a global optimization, this would allow to acquire even more challenging animated scenes. Finally, our representation of animated creatures with generalized cylinders could be used to, on the one hand, build a model and database of motion sequences for different creatures and motion objectives. On the other hand, such a database could be used to identify creatures and motion objectives based on the extracted cylinders.

7.2.2 Combinations for Future Work

Several of the approaches introduced in this thesis are not necessarily stand-alone techniques and could potentially benefit from a combination with other algorithms. Homunculus Warping, Interactive By-example Design of Artistic Packing Layouts, Projective Blue-Noise Sampling, and Animated 3D Creatures from Single-View Video by Skeletal Sketching are all handling input and output media of different types and dimensionality with varying design objectives. Combinations of the systems commonly require generalizations of the techniques to other input modalities of different dimensionality, enabling a compound of the techniques into a unified design pipeline. Possible pairwise combinations of the approaches are discussed before a full design pipeline including all approaches is sketched.

The animated meshes obtained in Chapter 6 could serve as input meshes to our Homunculus Warping technique (Chapter 3) allowing edits and deformations of single frames of the mesh animation. Extending Homunculus Warping to mesh animations would enable information visualization of time-varying aspects of the mesh sequences, e. g., the velocity of particular limbs. The specific mesh representation that comes as a result of Chapter 6, i. e., a mesh combined with animated axes, a representation close to a skeleton, can serve as a regularizer of Homunulus Warping, facilitating deformations that respect the viable articulation of the limbs. Leveraging a similar representation also for other meshes, resulting e. g., from the proposed sketching interface of Chapter 3, would enable articulated deformations for arbitrary input meshes. Turning this conjunction around, the collision detection and resolution step of Chapter 3 could be used as an additional regularizer in Chapter 6. Modeling collisions to regularize limb tracking in video is an involved and computationally demanding task probably requiring additional optimizations but would lead to more realistic results and improve the tracking substantially, as it limits the admissible deformation to more plausible results and hence prunes the search space. Leveraging the collisions in the surface estimation step, additionally, could lead to surfaces of higher quality closer to the correct solution.

The insights of the generalization of projective sampling to three or higher dimensions of Chapter 5 could be combined with the packing layouts of Chapter 4. This would lead to a general purpose algorithm to distribute objects of arbitrary dimensionality. In particular, the projective properties of presenting three-dimensional content on a two-dimensional screen pose an interesting and promising direction of future work, partially covered already in Chapter 5. Here, additional constraints such as occlusions and constrained camera positions could be taken into consideration. Combining packing with camera path planning [He, Cohen and Salesin, 1996] would introduce novel applications in visualizations. Including time as an additional dimension, the animated sequences from Chapter 6 could be presented as a collection of deforming, three-dimensional surfaces. Generally, dynamic and deforming primitives would require adjusting primitive positions over time and make up an interesting avenue to investigate further.

The deformation techniques of Chapter 3 could serve as an additional degree of freedom for the packing layouts in Chapter 4. It would allow for tighter or more even packing results.

Finally, all methods could be combined into a single, unified framework that, on the one hand, allows efficient extraction of animated, three-dimensional geometry from uncalibrated, two-dimensional video in combination with efficient and collision-free deformations of such content. On the other hand, the framework would enable to interactively pack and present large collections of such mesh animations on the screen and efficiently render them. All of the steps would be fully guided using spare subsets of examples controlling the parameters of the techniques that optionally can also be influenced directly by expert users.

Figure 7.1: Sketchbook for a cat character. *Image courtesy of Jesse Aclin.*

An example application for this framework could be *character design*. To create a new character, artists draw *character sketches* in different postures, deformations, as well as from different viewpoints (Figure 7.1) to examine different design choices and get an overview of the design space. Creating these sketchbooks is a time-consuming process that requires multiple iterations to converge and could benefit from computational support. Our unified framework could be used to obtain different postures for a character from real video footage (Chapter 6), deform the character to meet the artists intentions (Chapter 3), and render (Chapter 5) as well as arrange (Chapter 4) different styles and views of the character to yield a sketchbook-like character overview (cf. Figure 7.1).

7.2.3 General Outlook

Besides specific and combined directions for future research proposed in the former sections, the techniques presented in this thesis open up interesting general directions for further investigation.

From a user perspective, all methods could be integrated into existing software packages. On the one hand, there are straightforward integrations, e. g., including the relaxation approach of Chapter 4 into layout programs such as Adobe Photoshop [Adobe, 2016] that, so far, only support to arrange images in one dimension. Generally speaking, all software that allows arranging images could benefit from our approach. Another inclusion involves integration of collision detection and response into the modeling process of three-dimensional surfaces with three-dimensional modeling software such as Blender [Blender Online Community, 2016]. While integration of these functionalities can be done in isolation, on the other hand, there are more pervasive integrations of the techniques presented in this thesis. Shifting from the current paradigm of separate, detached user dialogs for many editing tasks in the aforementioned software packages to an example-based paradigm requires a profound redesign of the user interface. Optimally, example-based parameter estimation would complement existing manipulation and synthesis tools as suggested e. g., in Chapter 4. Initially, example-based parameter estimation could be used as a beginner or casual mode to facilitate application of the software by novice users. After each step, the software could display the estimated parameters, on the one hand, to enable fast error correction but, on the other hand, also to instruct the users about the estimated, underlying parameters. Over time recurring users would be able to subconsciously conceive these parameters without explicitly having to learn about them. Ultimately, a competent user would be able to pick the optimal interaction mode for the task at hand. For example, a rapid generation using the example-based approach in combination with minor fine tuning using explicit parameter dialogs could be the fastest way to achieve the goal; or depending on the specific task directly picking the parameters could be faster. Both options have the potential to be superior to the other one, both in terms of quality and convergence speed to a desired result depending on each specific task. Providing users the choice to pick the optimal technique enables efficient software usage by both, novice and expert users, with a smooth transition in between.

Our approaches compute solutions based solely on the current examples provided by the user and the model knowledge built into our system, completely ignoring the history of manipulations of the current and previous sessions by the same and other users. While careful design of the model knowledge has the ability to limit the possibilities to only valid solutions, such approaches fail repeatedly on the same examples. Leveraging the results of previous decisions on the same or similar inputs, our systems could, on the one hand, prioritize manipulations that are more likely to be performed by specific users. This could lead to user-specific profiles that adjust the model knowledge to present plausible results for each individual user more frequently. On the other hand, leveraging manipulations of all users, our system could learn models of plausible or “human” edits and interactively adjust the model knowledge put into the system for all users. This way failures could be compensated over time and implausible manipulations would be decreased in priority.

The presented techniques are applicable by both, expert and casual users. However, enabling casual usage eminently also involves facilitating mobile usage. While generally our techniques are already optimized for efficiency and parallel execution, for mobile hardware distinct limitations have to be considered for tangible applicability. Such limitations include restricted support of specific hardware functionalities and power consumption considerations. If expensive computations have to be executed that do not allow for interactive usage on mobile platforms, remote computing on a powerful server in combination with image streaming is an option. However, such solutions typically introduce latency that has to be compensated on the mobile device. Correcting the delayed image to correspond to the latest view using Image-based Rendering (Section 2.1.1) is an efficient and promising option, but commonly has limitations in image quality demanding for improved rendering techniques.

Finally, in the scope of this thesis, we investigated specific types of media synthesis and manipulation. Other manipulation and synthesis techniques could also benefit from example-based approaches, eventually leading to manipulation software for all kinds of visual media that is entirely established by example-based parameter estimation. Interactions between different kinds of media could lead to a unified framework and facilitate an intuitive transition between the different dimensions. Extending the work to other types of media, especially non-visual media such as sound, could be valuable in rendering also these areas more accessible to casual users.

7.3 Message

This thesis discussed how intricate tasks in manipulation and synthesis of visual media can benefit from more intuitive, example-based approaches that keep the user in the loop. The techniques presented in this thesis target casual users that typically do not perform editing tasks on a regular basis. Such users benefit from more intuitive user interfaces that learn from a small set of examples about the inherent relationships and restrictions in the data. To be effective these approaches require both, interactive performance and intuitive user-interfaces. Unlike other example-based techniques that learn complex models and hence only run offline, in our work we focus on optimization efficiency and achieve interactive performance by reformulation of the problems to run in parallel on GPUs. Potential future work includes adoption of example-driven approaches in cross-domain software packages and combinations with active learning to improve model knowledge. Looking ahead we believe that with the advent of more casual usage especially on mobile machines new intuitive and efficient tools will be required that could benefit from the insights of this thesis.

Bibliography (Own work)

LOCHMANN, G. ET AL.: Real-time Novel-view Synthesis for Volume Rendering Using a Piecewise-analytic Representation. In *Proceedings of VMV*. Bayreuth, Germany: Eurographics Association, 2016

LOCHMANN, G. ET AL.: Real-time Reflective and Refractive Novel-view Synthesis. In *Proceedings of VMV*. Darmstadt, Germany: Eurographics Association, 2014

REINERT, B. ET AL.: Proxy-guided Image-based Rendering for Mobile Devices. *Computer Graphics Forum (Proceedings of Pacific Graphics)* 2016, ISSN 1467–8659

REINERT, B., RITSCHEL, T. AND SEIDEL, H.-P.: Homunculus Warping: Conveying Importance using Self-intersection-free Non-homogeneous Mesh Deformation. *Computer Graphics Forum (Proceedings of Pacific Graphics)* 5 2012 (31)

REINERT, B., RITSCHEL, T. AND SEIDEL, H.-P.: Interactive By-example Design of Artistic Packing Layouts. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 31 2013 (6)

REINERT, B., RITSCHEL, T. AND SEIDEL, H.-P.: Animated 3D Creatures from Single-view video by Skeletal Sketching. *Graphics Interface* 31 2016 (6)

REINERT, B. ET AL.: Projective Blue-Noise Sampling. *Computer Graphics Forum (Presented at EGSR 2016)* 2015

REINERT, B., SCHUMANN, M. AND MÜLLER, S.: Parameter and Configuration Analysis for Non-linear Pose Estimation with Points and Lines. In *Proceedings of the International Conference on Computer Vision Theory and Applications (VISIGRAPP)*. 2012, 271–276

Bibliography

ADOBE: Photoshop CC. Adobe, San Jose: Adobe, 2016

AGARWALA, A. ET AL.: Keyframe-based tracking for rotoscoping and animation. *ACM Transactions on Graphics*, 23 2004 (3), 584–91

AKHTER, I. ET AL.: Nonrigid structure from motion in trajectory space. In *Proceedings of NIPS*. 2008, 41–48

ALEXA, M., COHEN-OR, D. AND LEVIN, D.: As-rigid-as-possible shape interpolation. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. 2000, 157–164

ALLEN, B., CURLESS, B. AND POPOVIĆ, Z.: The space of human body shapes: reconstruction and parameterization from range scans. *ACM Transactions on Graphics*, 22 2003 (3), 587–94

AMD: AMD Accelerated Parallel Processing OpenCL – Programming Guide. 2015

ARIKAN, O. AND FORSYTH, D. A.: Interactive Motion Generation from Examples. In *Proceedings of the 29th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 2002, SIGGRAPH '02, 483–490

ARIKAN, O., FORSYTH, D. A. AND O'BRIEN, J. F.: Motion synthesis from annotations. *ACM Transactions on Graphics*, 22 2003 (3), 402–8

BALZER, M., SCHLÖMER, T. AND DEUSSEN, O.: Capacity-constrained point distributions: a variant of Lloyd's method. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 28 2009 (3)

BARAN, I. AND POPOVIĆ, J.: Automatic Rigging and Animation of 3D Characters. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. New York, NY, USA: ACM, 2007, SIGGRAPH '07

BATEMAN, S., GUTWIN, C. AND NACENTA, M.: Seeing things in the clouds: the effect of visual features on tag cloud selections. In *Proceedings of ACM Hypertext and Hypermedia*. 2008, 193–202

BERGEN, J. R. ET AL.: Hierarchical model-based motion estimation. In *Proceedings of ECCV*. 1992, 237–52

BEZERRA, H. ET AL.: 3D dynamic grouping for guided stylization. In *Proceedings of NPAR*. 2008, 89–95

BINGHAM, G.: Perceiving the size of trees: Form as information about scale. *Journal of Experimental Psychology: Human Perception and Performance*, 19 1993 (6), 1139

BLANZ, V. AND VETTER, T.: A morphable model for the synthesis of 3D faces. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. 1999, 187–4

BLENDER ONLINE COMMUNITY: Blender – A 3D modelling and rendering package. Blender Institute, Amsterdam: Blender Foundation, 2016 ⟨URL: http://www.blender.org⟩

BLINN, J. F. AND NEWELL, M. E.: Texture and Reflection in Computer Generated Images. *Communications of the ACM*, 19 October 1976 (10), 542–547

BORLAND, D. AND TAYLOR II, R.: Rainbow color map (still) considered harmful. *IEEE CG & App.* 2007, 14–17

BOROSÁN, P. ET AL.: RigMesh: Automatic Rigging for Part-based Shape Modeling and Deformation. *ACM Transactions on Graphics*, 31 November 2012 (6), 198:1–198:9

BOTSCH, M. ET AL.: PriMo: Coupled prisms for intuitive surface modeling. In *Proceedings of SGP*. 2006, 11–20

BOTSCH, M. AND SORKINE, O.: On Linear Variational Surface Deformation Methods. *IEEE Transactions on Visualization and Computer Graphics*, 14 January 2008 (1), 213–230

BREGLER, C., HERTZMANN, A. AND BIERMANN, H.: Recovering non-rigid 3D shape from image streams. In *Proceedings of CVPR*. Volume 2, 2000

BREGLER, C. ET AL.: Turning to the masters: motion capturing cartoons. *ACM Transactions on Graphics* 2002

BROX, T. ET AL.: High accuracy optical flow estimation based on a theory for warping. In *ECCV* 2004

BUCHANAN, A. AND FITZGIBBON, A. W.: Interactive Feature Tracking using K-D Trees and Dynamic Programming. In *Proceedings of CVPR*. 2006

BUZAN, T.: Use both sides of your brain : new mind-mapping techniques to help you raise all levels of your intelligence and creativity, based on the latest discoveries about the human brain. New York, N.Y., U.S.A: Dutton, 1991

CASHMAN, T. AND FITZGIBBON, A.: What Shape Are Dolphins? Building 3D Morphable Models from 2D Images. *IEEE PAMI*, 35 2013 (1), 232–44

CHAPELLE, O., SCHÖLKOPF, B. AND ZIEN, A.: Semi-Supervised Learning. 1st edition. The MIT Press, 2010

CHEN, J. ET AL.: Bilateral Blue Noise Sampling. *ACM Transactions on Graphics*, 32 2013a (6), 216:1–216:11

CHEN, S. E. AND WILLIAMS, L.: View Interpolation for Image Synthesis. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. 1993, 279–88

CHEN, T. ET AL.: 3Sweep: Extracting Editable Objects from a Single Photo. *ACM Transactions on Graphics*, 32 2013b (6), 195:1–195:10

CHEN, Z. ET AL.: Variational Blue Noise Sampling. *IEEE Transactions on Visualization and Computer Graphics*, 18 2012 (10), 1784–96

CHIU, K., SHIRLEY, P. AND WANG, C.: Graphics Gems IV. 1994. – chapter Multi-jittered Sampling, 370–4

CHOI, B. AND LEE, C.: Sweep surfaces modelling via coordinate transformation and blending. *CAD*, 22 1990 (2), 87–96

COOK, R. L.: Stochastic sampling in computer graphics. *ACM Transactions on Graphics*, 5 1986 (1), 51–72

COOK, R. L., CARPENTER, L. AND CATMULL, E.: The Reyes Image Rendering Architecture. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 1987, SIGGRAPH '87, 95–102

COOTES, T. F. ET AL.: Trainable method of parametric shape description. *Image and Vision Computing*, 10 1992 (5), 289–94

CORMEN, T. H. ET AL.: Introduction to Algorithms. 2nd edition. McGraw-Hill Higher Education, 2001

COX, A. M. A. AND COX, F. T.: CHAP. MULTIDIMENSIONAL SCALING. INHandbook of Data Visualization. Berlin, Heidelberg: Springer Berlin Heidelberg, 2008, 315–347

CRANLEY, R. AND PATTERSON, T.: Randomization of number theoretic methods for multiple integration. *SIAM Journal on Numerical Analysis*, 13 1976 (6), 904–14

CUTTING, J.: Rigidity in cinema seen from the front row, side aisle. *Journal of Experimental Psychology: Human Perception and Performance*, 13 1987 (3), 323

DALAL, K. ET AL.: A spectral approach to NPR packing. In *Proceedings of NPAR*. 2006, 71–78

DEAN, J. AND GHEMAWAT, S.: MapReduce: Simplified Data Processing on Large Clusters. *Commun. ACM*, 51 January 2008 (1), 107–113

DEBEVEC, P. E., TAYLOR, C. J. AND MALIK, J.: Modeling and Rendering Architecture from Photographs: A Hybrid Geometry- and Image-based Approach. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. 1996, 11–20

DEUSSEN, O. ET AL.: Floating points: A method for computing stipple drawings. In *Computer Graphics Forum*. Volume 19, 2000, 41–50

DEVROYE, L.: Non-Uniform Random Variate Generation. Springer, 1986

DIPPÉ, M. A. AND WOLD, E. H.: Antialiasing through stochastic sampling. *ACM SIGGRAPH Computer Graphics*, 19 1985 (3), 69–78

DONG, Z. ET AL.: Real-time voxelization for complex polygonal models. In *Proceedings of Pacific Graphics*. 2004, 43–50

DUNBAR, D. AND HUMPHREYS, G.: A Spatial Data Structure for Fast Poisson-disk Sample Generation. *ACM Transactions on Graphics*, 25 2006 (3), 503–08

ELDAR, Y. ET AL.: The farthest point strategy for progressive image sampling. *IEEE Transactions on Image Processing*, 6 1997 (9), 1305–15

ENGLAND, N.: A graphics system architecture for interactive application-specific display functions. *IEEE Computer Graphics Applications*, 6 January 1986 (1), 60–70

EVERETT, R. R.: The Whirlwind I Computer. *Managing Requirements Knowledge, International Workshop on*, 0 1951, 70

FATTAL, R.: Blue-noise point sampling using kernel density model. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. Volume 30, Vancouver, 2011, 48

FAVREAU, L. ET AL.: Animal gaits from video. In BOULIC, R. AND PAI, D. K., EDITORS: *Symposium on Computer Animation*. The Eurographics Association, 2004

GAL, R. ET AL.: 3D collage: expressive non-realistic modeling. In *Proceedings of NPAR*. 2007, 7–14

GALL, J. ET AL.: Motion capture using joint skeleton tracking and surface estimation. In *CVPR*. June 2009, 1746–1753

GAMITO, M. N. AND MADDOCK, S. C.: Accurate Multidimensional Poisson-disk Sampling. *ACM Transactions on Graphics*, 29 2009 (1), 8:1–8:19

GARG, A., JACOBSON, A. AND GRINSPUN, E.: Computational Design of Reconfigurables. *ACM Transactions on Graphics* 35 2016 (4)

GARG, R., ROUSSOS, A. AND AGAPITO, L.: Dense variational reconstruction of non-rigid surfaces from monocular video. In *Proceedings of CVPR*. 2013, 1272–9

GOES, F. DE ET AL.: Blue Noise Through Optimal Transport. *ACM Transactions on Graphics*, 31 2012 (6), 171:1–171:11

GORAL, C. M. ET AL.: Modeling the Interaction of Light Between Diffuse Surfaces. In *Proceedings of the 11th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 1984, SIGGRAPH '84, 213–222

GORTLER, S. J. ET AL.: The Lumigraph. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 1996, 43–54

GREGORY, R.: Distortion of visual space as inappropriate constancy scaling. *Nature* 199 1963 (678-91)

GUAY, M. ET AL.: Space-time sketching of character animation. *ACM Transactions on Graphics*, 34 May 2015 (4), 1

HALTON, J. H.: Algorithm 247: Radical-inverse Quasi-random Point Sequence. *Communications of the ACM*, 7 December 1964 (12), 701–702

HAREL, D. AND KOREN, Y.: Drawing graphs with non-uniform vertices. In *Proceedings of Working Conference on Advanced Visual Interfaces*. 2002, 157–166

HARMON, D. ET AL.: Interference-aware geometric modeling. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 30 2011 (6), 137

HARTLEY, R. I. AND ZISSERMAN, A.: Multiple View Geometry in Computer Vision. 2nd edition. Cambridge University Press, 2004

HAUSNER, A.: Simulating decorative mosaics. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. 2001, 573–580

HE, L.-W., COHEN, M. F. AND SALESIN, D. H.: The Virtual Cinematographer: A Paradigm for Automatic Real-time Camera Control and Directing. In *Proceedings of the 23rd Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 1996, SIGGRAPH '96, 217–224

HECK, D., SCHLÖMER, T. AND DEUSSEN, O.: Blue Noise Sampling with Controlled Aliasing. *ACM Transactions on Graphics*, 32 2013 (3), 25:1–25:12

HILLER, S., HELLWIG, H. AND DEUSSEN, O.: Beyond stippling – Methods for distributing objects in the plane. In *Proceedings of Eurographics*. Volume 22, Granada, 2003, 515–522

HOFF, K. I. ET AL.: Fast computation of generalized Voronoi diagrams using graphics hardware. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. 1999, 277–86

HOIEM, D., EFROS, A. A. AND HEBERT, M.: Automatic photo pop-up. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 24 2005 (3), 577–84

HORRY, Y., ANJYO, K.-I. AND ARAI, K.: Tour into the picture: using a spidery mesh interface to make animation from a single image. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. 1997, 225–232

HSU, W.-H., MA, K.-L. AND CORREA, C.: A rendering framework for multiscale views of 3D models. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*. 2011, 131:1–131:10

HURTUT, T. ET AL.: Appearance-guided synthesis of element arrangements by example. In *Proceedings of NPAR*. 2009, 51–60

IGARASHI, T., MATSUOKA, S. AND TANAKA, H.: Teddy: A sketching interface for 3D freeform design. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. 1999, 409–16

IZADI, S. ET AL.: KinectFusion: real-time 3D reconstruction and interaction using a moving depth camera. In *Proceedings of UIST*. 2011, 559–568

JACOBS, C. ET AL.: Adaptive grid-based document layout. *ACM Transactions on Graphics*, 22 2003 (3), 838–847

JAIN, A. ET AL.: Moviereshape: Tracking and reshaping of humans in videos. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 29 2010 (6), 148

JI, Z., LIU, L. AND WANG, Y.: B-Mesh: A Modeling System for Base Meshes of 3D Articulated Shapes. *Computer Graphics Forum* 2010

JONES, T. R.: Efficient generation of Poisson-disk sampling patterns. *Journal of Graphics Tools* 11 2006

KAJIYA, J. T.: The Rendering Equation. In *Proceedings of the 13th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 1986, SIGGRAPH '86, 143–150

KALANTARI, N. K. AND SEN, P.: Efficient Computation of Blue Noise Point Sets through Importance Sampling. *Computer Graphics Forum*, 30 2011 (4), 1215–21

KARPENKO, O. A. AND HUGHES, J. F.: SmoothSketch: 3D free-form shapes from complex sketches. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. Volume 25, 2006, 589–98

KELLER, A.: Instant Radiosity. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, SIGGRAPH '97, 49–56

KELLER, A., PREMOZE, S. AND RAAB, M.: Advanced (Quasi) Monte Carlo Methods for Image Synthesis. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. 2012, 21:1–21:46

KENSLER, A.: Correlated Multi-Jittered Sampling. Pixar Technical Memo 13-01, 2013

KESSENICH, J. AND SELLERS, G.: Vulkan Programming Guide: The Official Guide to Learning Vulkan. Addison Wesley Publishing Company Incorporated, 2016, OpenGL Series

KIM, J. AND PELLACINI, F.: Jigsaw image mosaics. *ACM Transactions on Graphics*, 21 2002 (3), 657–664

KOENDERINK, J. J.: What does the occluding contour tell us about solid shape. *Perception*, 13 1984 (3), 321–30

KOH, K. ET AL.: Maniwordle: Providing flexible control over wordle. *IEEE Transactions on Visualization and Computer Graphics*, 16 2010 (6), 1190–97

KOHONEN, T.: The self-organizing map. *Proceedings of the IEEE*, 78 Sep 1990 (9), 1464–1480

KOLLIG, T. AND KELLER, A.: Efficient multidimensional sampling. In *Proceedings of Eurographics*. Volume 21, Saarbrücken, 2002, 557–63

KOPF, J. ET AL.: Recursive Wang Tiles for Real-time Blue Noise. *ACM Transactions on Graphics*, 25 2006 (3), 509–518

KORKIN, A. AND ZOLOTAREV, G.: Sur les formes quadratiques positives. In *Math. Ann. 11*. 1877, 242–292

KRAEVOY, V. ET AL.: Non-homogeneous resizing of complex models. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*. Volume 27, 2008, 111

KRUSKAL, J. B. AND WISH, M.: Multidimensional scaling. Beverely Hills, California: Sage Publications, 1978, 07 11

LAFORTUNE, E. P. AND WILLEMS, Y. D.: Bi-Directional Path Tracing. In *Proceedings of third International Conference on Computational Graphics and Visualization Techniques (Compugraphics '93)*. 1993, 145–153

LAGAE, A. AND DUTRÉ, P.: A procedural object distribution function. *ACM Transactions on Graphics*, 24 2005 (4), 1442–61

LAGAE, A. AND DUTRÉ, P.: Poisson Sphere Distributions. In *Proceedings of VMV*. Aachen, November 2006, 373–379

LAGAE, A. AND DUTRÉ, P.: A comparison of methods for generating Poisson disk distributions. 27 2008 (1), 114–29

LASRAM, A., LEFEBVRE, S. AND DAMEZ, C.: Procedural texture preview. *Computer Graphics Forum (Proceedings of Eurographics)*, 31 2012, 413–20

LEUNG, Y. AND APPERLEY, M.: A review and taxonomy of distortion-oriented presentation techniques. *ACM Transactions on Computer-Human Interaction*, 1 1994 (2), 126–160

LEVI, Z. AND GOTSMAN, C.: ArtiSketch: A System for Articulated Sketch Modeling. In *Computer Graphics Forum (Proceedings of Eurographics)*. Volume 32, 2013, 235–44

LÉVY, B. ET AL.: Least squares conformal maps for automatic texture atlas generation. In *ACM Transactions on Graphics*. Volume 21, 2002, 362–371

LI, G. ET AL.: Analysis, reconstruction and manipulation using arterial snakes. *ACM Transactions on Graphics*, 29 2010 (6), 152

LI, H., SUMNER, R. W. AND PAULY, M.: Global Correspondence Optimization for Non-rigid Registration of Depth Scans. In *Proceedings of the Symposium on Geometry Processing*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2008, SGP '08, 1421–1430

LLOYD, S.: Least squares quantization in PCM. *IEEE Transactions on Information Theory*, 28 1982 (2), 129–37

LODI, A., MARTELLO, S. AND VIGO, D.: Recent Advances on Two-dimensional Bin Packing Problems. *Discrete Applied Mathematics*, 123 November 2002 (1-3), 379–396

LOK, S. AND FEINER, S.: A survey of automated layout techniques for information presentations. In *Proceedings of Smart Graphics*. 2001, 61–68

LUNA, F.: Introduction to 3D Game Programming with DirectX 11. USA: Mercury Learning & Information, 2012

MAESTRI, G.: Digital Character Animation 3. Pearson Education, 2006

MAHARIK, R. ET AL.: Digital micrography. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 30 2011 (4), 100

MARK, W. R., MCMILLAN, L. AND BISHOP, G.: Post-rendering 3D Warping. In *Proceedings of i3D*. 1997

MARQUES, R. ET AL.: Spherical Fibonacci Point Sets for Illumination Integrals. *Computer Graphics Forum* 32 2013 (8)

MARTELLO, S. AND TOTH, P.: Linear assignment problems. *North-Holland Mathematics Studies*, 132 1987, 259–282

MATTHEWS, I., ISHIKAWA, T. AND BAKER, S.: The template update problem. *IEEE PAMI*, 26 2004 (6), 810–5

MATUSIK, W. ET AL.: Image-based visual hulls. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. 2000, 369–374

MCCOOL, M. AND FIUME, E.: Hierarchical Poisson disk sampling distributions. In *Proceedings of GI*. Vancouver, 1992

MCCOOL, M. D., QIN, Z. AND POPA, T. S.: Shader metaprogramming. In *Proc. ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware*. 2002, HWWS '02, 57–68

MCNAMARA, A. ET AL.: Fluid Control Using the Adjoint Method. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. New York, NY, USA: ACM, 2004, 449–456

MENZEL, N. AND GUTHE, M.: g-BRDFs: An Intuitive and Editable BTF Representation. *Computer Graphics Forum*, 28 2009 (8), 2189–2200

MICROSOFT: Word 2016. Microsoft, Redmond: Microsoft, 2016

MITCHELL, D.: Ray tracing and irregularities of distribution. In *Proceedings of EGWR*. Pisa, 1992, 61–9

MITCHELL, D. P.: Spectrally optimal sampling for distribution ray tracing. *ACM SIGGRAPH Computer Graphics*, 25 1991 (4), 157–64

MITCHELL, D.: Generating antialiased images at low sampling densities. *Computer Graphics (Proceedings of SIGGRAPH)*, 21 1987, 65–72

MITRA, N. J. AND PAULY, M.: Shadow Art. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*. New York, NY, USA: ACM, 2009, SIGGRAPH Asia '09, 156:1–156:7

MOORE, G. E.: Cramming More Components Onto Integrated Circuits. *Proceedings of the IEEE*, 86 Jan 1998 (1), 82–85

MÜLLER, M. AND GROSS, M.: Interactive virtual materials. In *Proceedings of Graphics Interface*. 2004, 239–246

MÜLLER, M. ET AL.: Position based dynamics. *Journal of Visual Communication and Image Representation*, 18 2007 (2), 109–118

MÜLLER, M. ET AL.: Meshless deformations based on shape matching. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. Volume 24, 2005, 471–8

NEALEN, A. ET AL.: FiberMesh: designing freeform surfaces with 3D curves. *ACM Transactions on Graphics*, 26 2007 (3), 41

NEALEN, A. ET AL.: Physically Based Deformable Models in Computer Graphics. *Computer Graphics Forum*, 25 2006 (4), 809–836

NGUYEN, C. H.: Data-driven Approaches for Interactive Appearance Editing. Ph. D thesis, Universität des Sarlandes, 2014

NGUYEN, C. H., RITSCHEL, T. AND SEIDEL, H.-P.: Data-driven Color Manifolds. *ACM Transactions on Graphics* 34 2015 (2)

NICODEMUS, F. E.: Directional Reflectance and Emissivity of an Opaque Surface. *Applied Optics*, 4 Jul 1965 (7), 767–775

NVIDIA: NVIDIA's Next Generation CUDA Compute Architecture: Kepler TM GK110. White paper, 2012

NVIDIA: NVIDIA CUDA Compute Unified Device Architecture – Programming Guide. NVIDIA, 2015

OSTROMOUKHOV, V.: Sampling with Polyominoes. *ACM Transactions on Graphics* 26 2007 (3)

OSTROMOUKHOV, V., DONOHUE, C. AND JODOIN, P.-M.: Fast hierarchical importance sampling with blue noise properties. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. Volume 23, Los Angeles, 2004, 488–95

ÖZTIRELI, A. C. AND GROSS, M.: Analysis and Synthesis of Point Distributions based on Pair Correlation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)* 31 2012 (6)

PENFIELD, W. AND RASMUSSEN, T.: The cerebral cortex of man; a clinical study of localization of function. 1950

PENG, C.-H., YANG, Y.-L. AND WONKA, P.: Computing Layouts with Deformable Templates. *ACM Transactions on Graphics*, 33 July 2014 (4), 99:1–99:11

PHARR, M. AND HUMPHREYS, G.: Physically based rendering: From theory to implementation. Morgan Kaufmann, 2010

PHONG, B. T.: Illumination for Computer Generated Pictures. *Communications of the ACM*, 18 June 1975 (6), 311–317

POPOVIĆ, J., SEITZ, S. M. AND ERDMANN, M.: Motion Sketching for Control of Rigid-body Simulations. *ACM Transactions on Graphics*, 22 October 2003 (4), 1034–1054

POTTMANN, H. AND HOFER, M.: Geometry of the squared distance function to curves and surfaces. In *Visualization and Mathematics III*. Springer, 2003, 223–44

PRASAD, M. AND FITZGIBBON, A.: Single view reconstruction of curved surfaces. In *Proceedings of CVPR*. Volume 2, 2006, 1345–54

PRAZDNY, K.: Egomotion and relative depth map from optical flow. *Biological Cybernetics*, 36 1980 (2), 87–102

RAMAMOORTHI, R. ET AL.: A theory of Monte Carlo visibility sampling. *ACM Transactions on Graphics*, 31 2012 (5), 121

RHEMANN, C. ET AL.: Fast cost-volume filtering for visual correspondence and beyond. In *Proceedings of CVPR*. 2011, 3017–24

RITSCHEL, T. ET AL.: The State of the Art in Interactive Global Illumination. *Computer Graphics Forum*, 31 February 2012 (1), 160–188

ROMERO, V. J. ET AL.: Initial Evaluation of Pure and Latinized Centroidal Voronoi Tessellation for Non-Uniform Statistical Sampling. *Rel. Eng. & Sys. Safety*, 91 2006 (10), 1266–80

RUSSELL, C., YU, R. AND AGAPITO, L.: Video Pop-up: Monocular 3D Reconstruction of Dynamic Scenes. *ECCV*, 2014, 583–598

SAITO, S., ZHOU, Z.-Y. AND KAVAN, L.: Computational Bodybuilding: Anatomically-based Modeling of Human Bodies. *ACM Transactions on Graphics*, 34 July 2015 (4), 41:1–41:12

SAKA, Y., GUNZBURGER, M. AND BURKHARDT, J.: Latinized, improved LHS, and CVT point sets in hypercubes. *International Journal of Numerical Analysis and Modeling*, 4 2007 (3-4), 729–743

SANCHEZ PEREZ, J., MEINHARDT-LLOPIS, E. AND FACCIOLO, G.: TV-L1 Optical Flow Estimation. *Image Processing On Line*, 3 2013, 137–150

SCHAEFER, S., MCPHAIL, T. AND WARREN, J.: Image deformation using moving least squares. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. Volume 25, 2006, 533–540

SCHERZER, D. ET AL.: Pre-convolved Radiance Caching. *Computer Graphics Forum (Proceedings of EGSR)* 4 2012 (31)

SCHLÖMER, T. AND DEUSSEN, O.: Towards a Standardized Spectral Analysis of Point Sets with Applications in Graphics. 2010

SCHLÖMER, T., HECK, D. AND DEUSSEN, O.: Farthest-point optimized point sets with maximized minimum distance. In *Proceedings of HPG*. Vancouver, 2011, 135–142

SCHULZ, C. ET AL.: Animating Deformable Objects Using Sparse Spacetime Constraints. *ACM Transactions on Graphics*, 33 July 2014 (4), 109:1–109:10

SEDERBERG, T. AND PARRY, S.: Free-form deformation of solid geometric models. *ACM SIGGRAPH Computer Graphics*, 20 1986 (4), 151–160

SHIRLEY, P.: Discrepancy as a quality measure for sample distributions. In *Proceedings of Eurographics*. Volume 91, Vienna, 1991, 183–94

SHOEMAKE, K.: Animating rotation with quaternion curves. In *ACM SIGGRAPH Computer Graphics*. Volume 19, 1985, 245–54

SHREINER, D. ET AL.: OpenGL Programming Guide: The Official Guide to Learning OpenGL, Version 4.3. 8th edition. Addison-Wesley Professional, 2013

SINHA, S. N. ET AL.: Interactive 3D architectural modeling from unordered photo collections. *ACM Transactions on Graphics*, 27 2008 (5), 159

SNAVELY, N., SEITZ, S. M. AND SZELISKI, R.: Photo Tourism: Exploring Photo Collections in 3D. *ACM Transactions on Graphics*, 25 2006 (3), 835–846

SOBOL, I. M.: A Primer for the Monte Carlo Method. 1st edition. CRC Press, May 1994

SORKINE, O. AND ALEXA, M.: As-rigid-as-possible surface modeling. In *Proceedings of SGP*. 2007, 109–116

SORKINE, O. ET AL.: Laplacian Surface Editing. In *Proceedings of the 2004 Eurographics/ACM SIGGRAPH Symposium on Geometry Processing*. New York, NY, USA: ACM, 2004, SGP '04, 175–184

SPETSAKIS, M. E. AND ALOIMON, J.: Closed Form Solution to the Structure from Motion Problem from Line Correspondences. In *Proceedings of the Sixth National Conference on Artificial Intelligence*. Volume 2, AAAI Press, 1987, 738–743

STEINBERGER, M.: Dynamic Resource Scheduling on Graphics Processors. Ph. D thesis, Graz Univerity of technology, 2013

STROBELT, H. ET AL.: Rolled-out Wordles: A Heuristic Method for Overlap Removal of 2D Data Representatives. In *Computer Graphics Forum*. Volume 31, 2012, 1135–44

SUMNER, R. W. AND POPOVIĆ, J.: Deformation Transfer for Triangle Meshes. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. New York, NY, USA: ACM, 2004, 399–405

SWICK, R. R. AND ACKERMAN, M. S.: The X Toolkit: More Bricks for Building User-Interfaces or Widgets for Hire. In *USENIX Winter*. USENIX Association, 1988, 221–228

SỲKORA, D., DINGLIANA, J. AND COLLINS, S.: As-rigid-as-possible image registration for hand-drawn cartoon animations. In *Proceedings of NPAR*. 2009, 25–33

SỲKORA, D. ET AL.: Adding depth to cartoons using sparse depth (in) equalities. *Computer Graphics Forum (Proceedings of Eurographics)*, 29 2010 (2), 615–23

TALTON, J. ET AL.: Metropolis procedural modeling. *ACM Transactions on Graphics*, 30 2011 (2), 11

TANENBAUM, A. S.: Modern Operating Systems. Upper Saddle River, NJ, USA: Prentice Hall Press, 2007

TAO, M. ET AL.: SimpleFlow: A Non-iterative, Sublinear Optical Flow Algorithm. In *Computer Graphics Forum (Proceedings of Eurographics)*. Volume 31, 2012, 345–53

TERZOPOULOS, D. ET AL.: Elastically Deformable Models. In *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 1987, SIGGRAPH '87, 205–214

TESCHNER, M. ET AL.: Optimized spatial hashing for collision detection of deformable objects. In *Proceedings of VMV*. 2003, 47–54

TUFTE, E. R.: Visual Explanations: Images and Quantities, Evidence and Narrative. Cheshire, CT, USA: Graphics Press, 1997

ULICHNEY, R.: Digital Halftoning. Cambridge, MA, USA: MIT Press, 1987

VEACH, E. AND GUIBAS, L. J.: Metropolis Light Transport. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, SIGGRAPH '97, 65–76

VOLLICK, I. ET AL.: Specifying label layout style by example. In *Proceedings of UIST*. 2007, 221–230

VON FUNCK, W., THEISEL, H. AND SEIDEL, H.: Vector field based shape deformations. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 25 2006 (3), 1118–1125

VORBA, J. ET AL.: On-line Learning of Parametric Mixture Models for Light Transport Simulation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)* 33 aug 2014 (4)

WACHTEL, F. ET AL.: Fast Tile-based Adaptive Sampling with User-specified Fourier Spectra. *ACM Transactions on Graphics*, 33 2014 (4), 56:1–56:11

WANG, J. ET AL.: Modeling Anisotropic Surface Reflectance with Example-based Microfacet Synthesis. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. New York, NY, USA: ACM, 2008, 41:1–41:9

WANG, L. ET AL.: The magic volume lens: An interactive focus+context technique for volume rendering. In *Proceedings of IEEE VIS*. 2005, 367–374

WANG, Y.-S., LEE, T.-Y. AND TAI, C.-L.: Focus+Context Visualization with Distortion Minimization. *IEEE Transactions on Visualization and Computer Graphics*, 14 2008 (6), 1731–1738

WANG, Y.-S. ET AL.: Feature-Preserving Volume Data Reduction and Focus+Context Visualization. *IEEE Transactions on Visualization and Computer Graphics*, 17 2011, 171–181

WEI, L.-Y. ET AL.: State of the Art in Example-based Texture Synthesis. In *Eurographics '09 State of the Art Reports (STARs)*. Eurographics, March 2009

WEI, L.-Y. AND WANG, R.: Differential domain analysis for non-uniform sampling. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. Volume 30, Vancouver, 2011, 50

WEI, L.: Multi-class blue noise sampling. *ACM Transactions on Graphics*, 29 2010 (4), 79

WELCH, W. AND WITKIN, A.: Free-form Shape Design Using Triangulated Surfaces. In *Proceedings of the 21st Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 1994, SIGGRAPH '94, 247–256

WITKIN, A. AND KASS, M.: Spacetime Constraints. In *Proceedings of the 15th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM, 1988, SIGGRAPH '88, 159–168

WOJTAN, C., MUCHA, P. J. AND TURK, G.: Keyframe Control of Complex Particle Systems Using the Adjoint Method. In *Proceedings of the 2006 ACM SIGGRAPH/Eurographics Symposium on Computer Animation*. Aire-la-Ville, Switzerland, Switzerland: Eurographics Association, 2006, SCA '06, 15–23

WOLBERG, G.: Image morphing: a survey. *The visual computer*, 14 1998 (8), 360–372

WU, T.-P. ET AL.: Interactive normal reconstruction from a single image. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 27 2008 (5), 119

XU, J. AND KAPLAN, C.: Calligraphic packing. In *Proceedings of GI*. 2007, 43–50

XU, P. ET AL.: Global Beautification of Layouts with Interactive Ambiguity Resolution. In *Proceedings of the 27th Annual ACM Symposium on User Interface Software and Technology*. New York, NY, USA: ACM, 2014, UIST '14, 243–252

XU, X. ET AL.: Animating Animal Motion from Still. *ACM Transactions on Graphics (Proceedings of SIGGRAPH Asia)*, 27 2008 (5), 117:1–117:8

YELLOTT, J.: Spectral consequences of photoreceptor sampling in the Rhesus retina. *Science*, 221 1983 (4608), 382–85

YU, L.-F. ET AL.: Make it home: automatic optimization of furniture arrangement. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 30 2011 (4), 86

ZELEZNIK, R. C., HERNDON, K. P. AND HUGHES, J. F.: SKETCH: an Interface for Sketching 3D Scenes. In *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*. 1996, 163–170

ZHANG, D., ZHOU, Z. AND CHEN, S.: Semi-supervised dimensionality reduction. In *Proceedings of SIAM Data Mining*. 2007, 629–34

ZHANG, L. ET AL.: Single-view modelling of free-form scenes. *Journal of Visualization and Computer Animation*, 13 2002 (4), 225–235

ZHENG, Y. ET AL.: Interactive images: Cuboid proxies for smart image manipulation. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 31 2012 (4), 99

ZHOU, S. ET AL.: Parametric reshaping of human bodies in images. *ACM Transactions on Graphics (Proceedings of SIGGRAPH)*, 29 2010 (4), 126

ZOLLHÖFER, M. ET AL.: GPU based ARAP Deformation using Volumetric Lattices. In ANDÚJAR, C. AND PUPPO, E., EDITORS: *Eurographics (Short Papers)*. Eurographics Association, 2012, 85–88

ZORIN, D., SCHRÖDER, P. AND SWELDENS, W.: Interactive Multiresolution Mesh Editing. In *Proceedings of the 24th Annual Conference on Computer Graphics and Interactive Techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 1997, SIGGRAPH '97, 259–268