2017

Computational methods and software for the design of inertial microfluidic flow sculpting devices

Daniel James Stoecklein
Iowa State University

Computational methods and software for the design of inertial microfluidic flow sculpting devices

by

Daniel Stoecklein

A dissertation submitted to the graduate faculty

in partial fulfillment of the requirements for the degree of

DOCTOR OF PHILOSOPHY

Major: Mechanical Engineering

Program of Study Committee:
Baskar Ganapathysubramanian, Major Professor
Ludovico Cademartiri
Jonathan Claussen
Nicole Nastaran Hashemi
Shankar Subramaniam

Iowa State University

Ames, Iowa

2017

DEDICATION

To my lovely wife, Suzy, and our three dogs, Weebie, Indy, and Lotte.

And to my Mom, Dad, Brother, and Sister.

I owe my joy in learning to you.

TABLE OF CONTENTS

LIST OF FIGURES

ACKNOWLEDGEMENTS

I would first like to thank my wife, Suzy, for putting up with my time as a graduate student and being incredibly supportive at every step. You have shouldered my setbacks, and celebrated every success. I owe this accomplishment entirely to you.

I am especially grateful to Dr. Baskar Ganapathysubramanian for the opportunity to work on this exciting research, as well as the time and patience afforded to me over the last few years. I'm also indebted to Dr. Yu Xie, on whose shoulders I have stood in my time as a graduate student; Dr. Chueh-Yu Wu, whose experiments and expertise contributed to much of my work; and Keegan Owsley, who was the architect and primary author of the software "uFlow".

I am quite fortunate in the composition of my committee, and appreciate their insight: Dr. Ludovico Cademartiri, Dr. Jonathan Claussen, Dr. Nicole Nastaran Hashemi, and especially Dr. Shankar Subramaniam, who has fostered and encouraged my affinity for teaching. This fortune extends to collaborators: Dr. Dino Di Carlo, who initiated this project and continues to push it new heights, and Dr. Soumik Sarkar, along his student, Kin Gwn Lore, who have been a joy to work with. Additionally, I am grateful to Dr. Terrence Meyer for initially taking me on and helping to point me in the right direction.

I would also like to acknowledge the various high school and undergraduate students who I have worked with and mentored: Katherine Abrams, Jan Borchers, Matthew Fulford, Jonathan Le, Triet Ly, Julia Meyer, Jesse Trujillo, Nadab Wubshet; along with Marc Pedersen from Valley High School. I am particularly grateful to Michael Davies, who was indispensable in this work, and whose career I will watch with great interest.

Finally, I would like to thank my lab mates, who made many hours spent in front of a computer bearable (the Baskar group is too large to list here). I learned a great deal from them, and will endeavor to pass on this camaraderie, and cooking recipes, to future peers.

This work was supported by the National Science Foundation (NSF) grant NSF-1306866.

ABSTRACT

The ability to sculpt inertially flowing fluid via bluff body obstacles has enormous promise for applications in bioengineering, chemistry, and manufacturing within microfluidic devices. However, the computational difficulty inherent to full scale 3-dimensional fluid flow simulations makes designing and optimizing such systems tedious, costly, and generally tasked to computational experts with access to high performance resources. The goal of this work is to construct efficient models for the design of inertial microfluidic flow sculpting devices, and implement these models in freely available, user-friendly software for the broader microfluidics community. Two software packages were developed to accomplish this: "uFlow" and "FlowSculpt". uFlow solves the forward problem in flow sculpting, that of predicting the net deformation from an arbitrary sequence of obstacles (pillars), and includes estimations of transverse mass diffusion and particles formed by optical lithography. FlowSculpt solves the more difficult inverse problem in flow sculpting, which is to design a flow sculpting device which produces a target flow shape. Each piece of software uses efficient, experimentally validated forward models developed within this work, which are applied to deep learning techniques to explore other routes to solving the inverse problem. The models are also highly modular, capable of incorporating new microfluidic components and flow physics to the design process. It is anticipated that the microfluidics community will integrate the tools developed here into their own research, and bring new designs, components, and applications to the inertial flow sculpting platform.

CHAPTER 1. INTRODUCTION

The directed use of fluid inertia in microscale flows has seen a dramatic rise in popularity within the last two decades across disciplines in bioengineering, chemistry, materials science, and manufacturing. Although inertial fluid flow[1] has been studied in the fluids community for more than half a century (Segré and Silberberg, 1961), modern computational resources have made understanding and exploiting the non-linear effects of this flow regime increasingly tractable, while breakthroughs in microscale manufacturing processes enabled the low-cost exploration of microfluidic devices designs and their applications. Two physical characteristics of inertial flow have driven the resurgence of interest within the microfluidics community: control over finite-sized particles in flow, and control over the fluid itself. Many of the particle-based applications have leveraged the same self-focusing effect that Segre and Silberberg discovered in 1961, where particles migrate to stable focusing positions within the cross-section of a straight channel, despite the lack secondary flow within the fluid (Segré and Silberberg, 1961; Di Carlo et al., 2007; Di Carlo, 2009). However, if the channel geometry is altered, the inertia within the fluid will induce complex secondary flows, which will perturb both larger particles in flow and the fluid itself. The particles will eventually return to stable focusing positions, but the fluid has undergone a more permanent transformation. This dissertation is centered on the latter concept of manipulating the structure of inertially flowing fluid by the careful placement of obstacles in flow.

Early use of this type of fluid flow deformation was primarily for the mixing of fluids, with grooves or herringbone patterns in the channel floor, or curved and serpentine channels, inducing chaotic advection at low Re (Liu et al., 2000; Stroock et al., 2002; Howell et al., 2004;

[1] In this work, the inertial flow regime is defined by $1 < Re < 100$, with the Reynolds number being the ratio of inertial to viscous forces, defined as $Re = \frac{UD_H}{\nu}$ for fluid velocity U and kinematic viscosity ν, and channel hydraulic diameter D_H.

Sudarsan and Ugaz, 2006). Patterned channels were also employed as a method of hydrodynamic focusing, forming sheathed flow into a continuous stream with a circular cross-section - aiding in the fabrication of functionalized microfibers (Howell et al., 2008; Daniele et al., 2015; Sharifi et al., 2016), for example. While useful, these operations had limited capability to purposely deform flow to a desired cross-sectional shape. Amini et al. (2013) introduced the concept of using a sequence of bluff-body obstacles to deterministically deform the fluid in a programmable manner. The key idea in this method is that each obstacle - typically a cylindrical pillar spanning the height of the microchannel - is placed far enough apart from neighboring pillars in order to prevent cross-talk. If the Reynolds number is low enough to prevent time-dependent effects (e.g., periodic motion or vortex shedding), the deformation from each pillar becomes an independent operator on the cross-sectional shape of the fluid. Hence, the net deformation from a sequence of pillars becomes a deterministic and programmable system, represented by a set of nested flow operations acting on an arbitrary inlet flow configuration. This process, known as "flow sculpting", was quickly adopted by the microfluidics community to create tailored fibers (Nunes et al., 2014), particles (Paulsen et al., 2015), and separate interstitial particles from co-flowing streams Sollier et al. (2015), to cite a few example applications.

A useful feature that made flow sculpting (computationally) tractable is that an obstacle's flow deformation can be pre-computed and stored as a 2-dimensional advection map, which contains information on how fluid displaces laterally as it moves past the obstacle. This relegates the heavy computational workload of solving the full Navier-Stokes equations over a set of 3-dimensional domains to a one-time task. Following this, a library of pre-computed advection maps can be rapidly accessed for fast simulations of arbitrary sequences of flow-deforming obstacles. This machinery directly solves the forward model of flow sculpting, which is predicting how fluid will deform based on an inlet flow and sequence of pillars. The inverse problem is more complex, however: determining the necessary pillar sequence and inlet flow condition which produces a desired fluid flow shape. The difficulty in the inverse problem arises from the non-linear nature of inertial flow deformations, combined with a large combinatorial selection of pillars, to make a highly corrugated, multi-modal, diverse, and richly featured design space.

The goal of this work is to exploit the concept of programmable pillar sequences to easily and rapidly design flow sculpting devices, solve the inverse problem of determining a device design for a desired fluid flow shape, and enable the broader microfluidics community to leverage these advances for their own research needs. This is accomplished by pursuing four primary objectives: (1) demonstrate the utility and rich design space of flow sculpting by designing and fabricating useful pillar program designs; (2) develop an efficient formulation of the forward model to enable widespread adoption and understanding of flow sculpting; (3) solve the inverse problem of flow sculpting; and (4) implement the solution to the inverse problem in a user-friendly framework for non-expert users in research and industry. The subsequent chapters in this dissertation are based on published (or submitted/in preparation) journal papers.

This dissertation is organized as follows. Chapter 2 describes manual exploration of the flow sculpting design space, and demonstrates a wide variety of fluid shapes achievable through hierarchical design. Chapter 3 directly tackles the inverse problem in flow sculpting by utilizing the genetic algorithm, and shows the successful optimization of micropillar sequence designs. Chapter 4 introduces the FlowSculpt software, a freely available application that solves the inverse problem for automated design of flow sculpting devices, and is capable of running on most modern computers. Chapter 5 describes the use of deep learning as another route to solving the inverse problem, with fast feedback showing enormous potential for rapid design. Chapters 6 describes in detail another software, uFlow, outlining its models for visualizing inertial flow physics. Chapter 7 shows the current state-of-the-art of the FlowSculpt software, with drastically improved optimization runtime and full-channel asymmetric design. Chapter 8 concludes with a look at continuing work on the tools and methods developed thus far, and future directions for flow sculpting.

CHAPTER 2. MICROPILLAR SEQUENCE DESIGNS FOR FUNDAMENTAL INERTIAL FLOW TRANSFORMATIONS

A paper accepted by *Lab on a chip*

Daniel Stoecklein, Chueh-Yu Wu, Keegan Owsley, Yu Xie,

Dino Di Carlo, and Baskar Ganapathysubramanian

Abstract

The ability to control the shape of a flow in a passive microfluidic device enables potential applications in chemical reaction control, particle separations, and complex material fabrication. Recent work has demonstrated the concept of sculpting fluid streams in a microchannel using a set of pillars or other structures that individually deform a flow in a predictable precomputed manner. These individual pillars are then placed in a defined sequence within the channel to yield the composition of the individual flow deformations - and ultimately complex user defined flow shapes. In this way, an elegant mathematical operation can yield the final flow shape for a sequence without experiment or additional numerical simulation. Although these approaches allow for programming complex flow shapes without understanding the detailed fluid mechanics, the design of an arbitrary flow shape of interest remains difficult, requiring significant design iteration. The development of intuitive basic operations (i.e. higher level functions that consist of combinations of obstacles) that act on the flow field to create a basis for more complex transformations would be useful in systematically achieving a desired flow shape. Here, we show eight transformations that could serve as a partial basis for more complex transformations. We initially used an in-house, freely available custom software (uFlow) which allowed us to arrive at these transformations that include making a fluid stream

concave and convex, tilting, stretching, splitting, adding a vertex, shifting, and encapsulating another flow stream. The pillar sequences corresponding to these transformations were subsequently fabricated and optically analyzed using confocal imaging - yielding close agreement with uFlow-predicted shapes. We performed topological analysis on each transformation, characterizing potential sequences leading to these outputs and trends associated with changing diameter and placement of the pillars. We classify operations into four sets of sequence-building concatenations: stacking, recursion, mirroring, and shaping. The developed basis should help in the design of microfluidic systems that have a phenomenal variety of applications, such as optofluidic lensing, enhanced heat transfer, or new polymer fiber design.

2.1 Introduction

The ability to control the cross-sectional shape of a fluid stream is enabling for a variety of microfluidic applications, from optimizing mixing for reactions and heat transfer (Sudarsan and Ugaz, 2006) to developing tunable optical component such as fluid lenses (Mao et al., 2007), to fabricating shaped polymer fibers (Nunes et al., 2014). The most common methods by which fluid parcels have been diverted across a channel cross-section to shape a stream make use of asymmetric structured channels (e.g. grooves, chevrons) in Stokes flow (Stroock et al., 2002)) or curved channels with finite inertia (Sudarsan and Ugaz, 2006; Liu et al., 2000; Di Carlo, 2009). Grooved channels have been widely used to mix flows (Howell, Jr. et al., 2005) and more recently combinations of grooves have been combined to shape flows (Boyd et al., 2013) in a programmable manner.

Fluid inertia also allows for the cross-stream motion of fluid parcels in curving or structured channels. Inertial fluid flow ($1 < Re < 100$, where the Reynolds number, $Re = \rho U D_H/\mu$, is the ratio of inertial to viscous forces in the flow, with fluid density ρ, viscosity μ, downstream velocity U, and hydraulic diameter D_H) has recently seen increased interest in the microfluidics community, with various strategies for manipulating particles emerging from equilibrium inertial focusing effects and the action of inertial lift forces on particles (Segré and Silberberg, 1961; Di Carlo, 2009). Fluid streams within a channel can also be deformed by the presence of particles (Amini et al., 2012), curvature (Sudarsan and Ugaz, 2006), or channel structure

(Amini et al., 2013). Liu et al. (2000) employed a serpentine channel design which efficiently used chaotic advection for mixing fluids. This most common approach used Dean flows, which are induced secondary flows through curved channels, to deform the flow field and increase interfacial area for mixing. Dean flow can also act in superposition with inertial lift forces to reduce the number of stable particle focusing positions (Di Carlo, 2009). A remarkable example of fluid flow engineering via Dean flow was the creation of a tunable optofluidic lens (Mao et al., 2007). But Dean flow is limited in its ability to precisely deform a particular region of the fluid - that is, the entire flow cross-section is manipulated. Particle-induced convection and Dean flow-based schemes - while useful for mixing - are difficult to adapt to arbitrary flow sculpting. However, recent developments in geometry-induced secondary flows under finite inertia conditions show promise for novel strategies in microfluid engineering with local flow stream perturbations (Amini et al., 2013).

The inertial flow around a cylindrical pillar yields secondary flows within a channel cross-section that are localized to the pillar location. The flow deformation around a simple geometry in this inertial regime has been only recently investigated in microfluidic systems, due to the general assumption of Stokes flow creating fore-aft symmetry in flow deformations around an object (Di Carlo, 2009). Experimental work has made use of flow separation and recirculations that form downstream of obstacles, where the fore-aft symmetry is broken in inertial flow around a geometry placed in the cross section of the fluid stream (Sudarsan and Ugaz, 2006; Chiu, 2007). More recently, microscale cylinders (pillars) have been used to induce secondary flow (separate from the flow separation) and this has been used in tandem with inertial focusing in order to reposition particle streams at high-throughput (Chung et al., 2013). A single cylindrical pillar at a variety of lateral positions in a microchannel has been found to induce a set of local fundamental deformations in inertial flow around each pillar (Amini et al., 2013). One notable aspect of the inertial flow deformation past a micropillar is the saturation of the turning motion of the flow within a limited distance downstream ($<\sim$6 pillar diameters), which leads to the idea of programmability. Lightweight, pre-computed advection maps for each pillar can be stored and rapidly accessed, with the net advection from a sequence of pillars being easily composed from their representative maps.

Using this method, and with appropriate inter-pillar spacing, pillars placed sequentially in a microfluidic channel can be can be easily simulated with little new computation (Amini et al., 2013). Useful subsequences can be labelled as functions, which can act on fluids alone or concatenate in order to tailor the flow as desired. These 'pillar programs' can open up an extensive library of associated net deformations for future microfluidic applications, such as using shifted fluid streams to improve heat transfer from local hotspots, creating new methods of cell sorting in bio-medical diagnostic 'lab-on-a-chip' devices, enhancing reactions at the interface between fluids by increasing surface area of streams, or changing the cross-section shape of polymer precursors for novel material design (Nunes et al., 2014) .

The complicated phase space on offer per micropillar was initially described through dimensional analysis using three non-dimensional groups: the Reynolds number, the channel aspect ratio h/w, and normalized pillar diameter D/w, where h, w, and D are the channel height, width, and pillar diameter. A dataset of 12,400 transformations with varying pillar diameter, offset, channel height, and flow Reynolds number was generated through distributed HPC resources by Diaz-Montes et al. (2014). Changing the pillar position and diameter was observed to have a tremendous effect on the fluid transformation with rich variety of shapes, but the vast number of permutations for sequences from this large dataset made for a daunting phase space of pillar programs. Therefore, a reduced, discretized framework was created for a preliminary foray into understanding what is possible with pillar programming. In order to quickly explore the potential operations allowed by this new method of microfluid engineering, we introduce a lightweight program (uFlow, www.biomicrofluidics.com/software.php) that uses a database of high precision 3D simulations based on single-pillar deformations. We first utilize this easily extensible program to develop intuition with the most fundamental transformations (one to three pillars per sequence), which we eventually combine to design more complex, hierarchically designed transformations. This facilitates rapid investigation of pillar programs without requiring access to high performance computational power. A set of eight fluid transformations, chosen for potential applications in microfluidics, are initially identified as target transformations. This first set of functions helps to illustrate the impact of pillar programming, in addition to signaling the potential complexity of a more complete library of

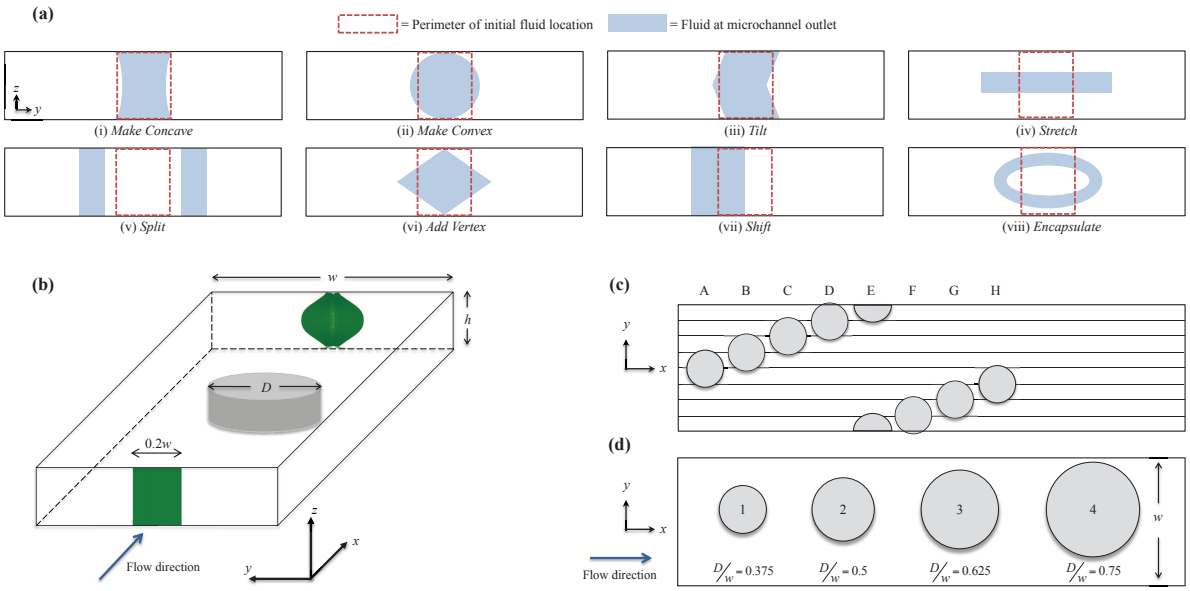

Figure 2.1 (a) Objective transformations. (b) Schematic of microchannel, showing a cross–section of the fluid elements at the inlet (middle 20% being tracked), and a sample deformation induced by a single pillar at the center of the channel. (c) Discretized simulation scheme lettered indices for locations space $0.125w$ (A-H), and (d) pillar diameters shown with indices 1-4, labelled with true pillar diameter and drawn to scale.

programs. Upon obtaining these novel transformations through this software exploration, we validated the designs by fabricating microfluidic devices and conducting confocal imaging. This work presents specific sequences used to create the target operations, along with some of the basic intuition, tools, and analysis that emerged from the process of finding and understanding these operations.

2.2 Objective

8 fundamental transformations were identified to be potentially useful in microfluidic applications, and are illustrated in figure 2.1. Unless specified otherwise, all transformations will be acting on a fluid region of interest 1/5 the width of the channel, and spanning the entire height (figure 2.1(b)). The objective transformations are the following:

- '*Make Concave*' (figure 2.1(a-i)) and '*Make Convex*' (figure 2.1(a-ii)) give curvature to the fluid interface for potential microscale optofluidic applications, or as a basic tool for hierarchical design by collapsing or expanding the fluid at different sections.
- '*Tilt*' (figure 2.1(a-iii)) is among the most basic transformations, and could be utilized for minor deformations in hierarchical design, or as a simple folding mechanism for enhanced mixing.
- '*Stretch*' (figure 2.1(a-iv)) flattens the fluid region, and its ability to collapse the whole shape could be useful in program design, as many transformations will have weaker effects on the fluid elements near the microfluid channel walls. Control over the thickness of the fluid via *stretch* will aid more complex fiber design.
- '*Split*' (figure 2.1(a-v)) has applications for fiber design and could lead to simultaneous engineering of separate streams in the microfluid channel.
- '*Add Vertex*' (figure 2.1(a-vi)) stands apart from most of the transformations due to its sharp vertices at the midsection of the fluid. Such angles could help define a more robust interlocking structure, or hint to more complex polygonal transformations.
- '*Shift*' (figure 2.1(a-vii)) is a very powerful manipulation of the fluid flow, with potential to isolate and further engineer fluid regions of interest. A program that could laterally shift streams will be a boon to mixing, heat transfer, solution reactions, and other far-reaching applications.
- '*Encapsulate*' (figure 2.1(a-viii)) seeks to envelope a fluid stream with the primary, central stream. It could be used to create a polymer sheath, induce coaxial solution reactions, or for general mixing applications.

2.3 Method

2.3.1 Computational

The phase space for this initial study is limited to manipulating the non-dimensional pillar diameter, D/w, and lateral location, y/w. The position was defined by eight equally spaced

offsets in a microchannel ($h/w = 0.25$), while pillar diameters were restricted to $D/w = 0.375$, $D/w = 0.5$, $D/w = 0.625$, $D/w = 0.75$. The velocity field for each pillar case was computed by solving the Navier-Stokes equations using the finite element method, incorporating streamline-upwind/Petrov-Galerkin (SUPG) (Brooks and Hughes, 1982) and pressure-stabilizing/Petrov-Galerkin (PSPG) (Tezduyar et al., 1992) terms to stabilize the numerical solution. No-slip boundary conditions are applied on the lateral walls and the surface of the pillar. Due to symmetry of the flow field, the simulation is only performed on the top half of the channel with a mirror boundary condition of velocity components on the centerline of the channel. We assume that the fluid is incompressible and at a steady state, and that the velocity profile is fully developed at the inlet. The pressure drop between the inlet and outlet is fixed and is determined to match the flow rate. The inlet velocity profile and pressure drop are calculated with the method used in (Jaeger et al., 2012), where the inlet velocity profile is predicted by solving the 2D cross-section flow with a guessed pressure gradient from the Hagen-Poisseuille law, and then corrected by linearly adjusting the pressure gradient to give the correct flow rate. For each configuration, fluid parcels were tracked through their respective velocity field to produce an advection map, which describes the displacement of each parcel as it travels down the channel. Stream deformation is characterized by marking those fluid parcels that belong to a stream of interest, and looking up the motion of those particles in the advection map. Advection maps are concatenated by using a parcel's deformed location to lookup the displacement in the following map, and summing displacements down the pillar sequence. The result is a new advection map comprising of the net effect of the entire pillar sequence.

2.3.2 uFlow

A lightweight utility "uFlow" was created (and is freely available online[¶]) that composes advection maps in real-time using the graphical processor (GPU) available on most modern consumer computing hardware. The software presents a simple graphical user interface that allows the user to place pillars into a straight channel and mark regions of the flow using a pallette of colors. Feedback is provided immediately in the form of cross-sectional views of the fluid between each pillar and at the end of the channel.

In order to provide realtime feedback, uFlow takes advantage of some properties of advection maps that make them amenable to the highly parallelized computing provided by the GPU. Briefly, the pre-computed advection maps for each pillar are loaded as OpenGL floating point textures, with x, y, and z displacements stored in the blue, red, and green channels, respectively. A render-to-texture framebuffer object of the same size as the advection maps is created to store the output. A custom fragment shader is executed for each "pixel" in the framebuffer, corresponding to a single fluid parcel in the advection map. The fragment shader performs two fast texture lookups using the GPU's texture fetching hardware, and combines two advection maps into a single map corresponding to the net parcel displacement. The result is stored into a texture, and can be used as the input advection map for a subsequent pillar. In this way, the net displacement for each parcel is accumulated as it travels down the channel.

For visualization, a custom vertex shader performs a texture lookup into the advection map, and uses this information to deform particle locations. The deformed particles are rendered onto the user's screen, and optionally exported to a high-resolution png. This entire loop occurs in realtime on typical laptop hardware.

2.3.3 Fabrication

The microfluidic devices for verifying transformations were fabricated using conventional soft photolithography. The microchannels with sequence of pillars were designed to be 200 μm by 50 μm, and standard photolithography was utilized to produce corresponding mold from a silicon master spin-coated with KMPR 1050 (MicroChem Corp.). Sylgard 184 Elastomer Kit (Dow Corning Corporation) was used to replicate the polydimethylsiloxane (PDMS) devices from the mold. PDMS base was mixed with a curing agent with a ratio of 10:1, and poured into the molds placed in the petri dishes. The petri dishes were put in a vacuum to remove bubbles and then in an oven until fully cured. The PDMS devices were peeled from the mold and punched using a pin vise (Technical Innovations, Inc.) to create inlet and outlet holes. The PDMS device and a thin glass slide were activated by air plasma (Plasma Cleaner, Harrick Plasma) and bonded together to enclose the microchannel. To help visualizing the channel,

[0¶] www.biomicrofluidics.com/software.php

Rhodamine B (Sigma-Aldrich) was infused into the channels to permeate the PDMS layer, and washed after one night.

2.3.4 Confocal Imaging

The confocal images were obtained using a Leica inverted SP1 confocal microscope at the California NanoSystem Institute. For each transformation, the bonded microfluidic device was taped tightly on the stage of microscopy and its three inlets were connected to three syringes set on three separate syringe pumps (Harvard Apparatus PHD 2000) by PEEK tubing (Upchurch Scientific Product No. 1569). The middle syringe included fluorescent isothiocyanate dextran 500kDa (5 μM, Sigma-Aldrich) dissolved in deionized water to help with visualization of the deformed stream, and the other two were filled only with deionized water. The volume flow rate of each pump is proportional to area of the flow stream at the microchannel inlet, with a total volume flow rate of 150 μL/min. The images were taken perpendicular to the flow direction at a downstream location of at least four times the pillar diameter from the last pillar. Six images were taken and averaged to get a final image for each case to avoid random noise.

2.4 Results

Figure 2.2(a-h) shows pillar sequences found to engineer the fluid into each objective output (not shown to scale). There are two indices for each pillar (above the channel schematic and within the pillars). The naming convention for pillar program schematics is as follows: the lettered index above the figure represents the pillar's position in the channel, with eight possible positions (see figure 2.1(c)). The numbered index placed inside of the pillar denotes its diameter, with four numbers corresponding to four diameters: $1 = 0.375w$, $2 = 0.5w$, $3 = 0.625w$, and $4 = 0.75w$ (see figure 2.1(d)). $Re = 20$ for all conditions, and the channel aspect ratio $h/w = 0.25$.

We divide the pillar programs into two categories: fundamental and hierarchically designed transformations. The fundamental transformations are created using simple programs with three or fewer pillars, and can be leveraged in sequence in order to create the more complex hierarchically designed transformations.

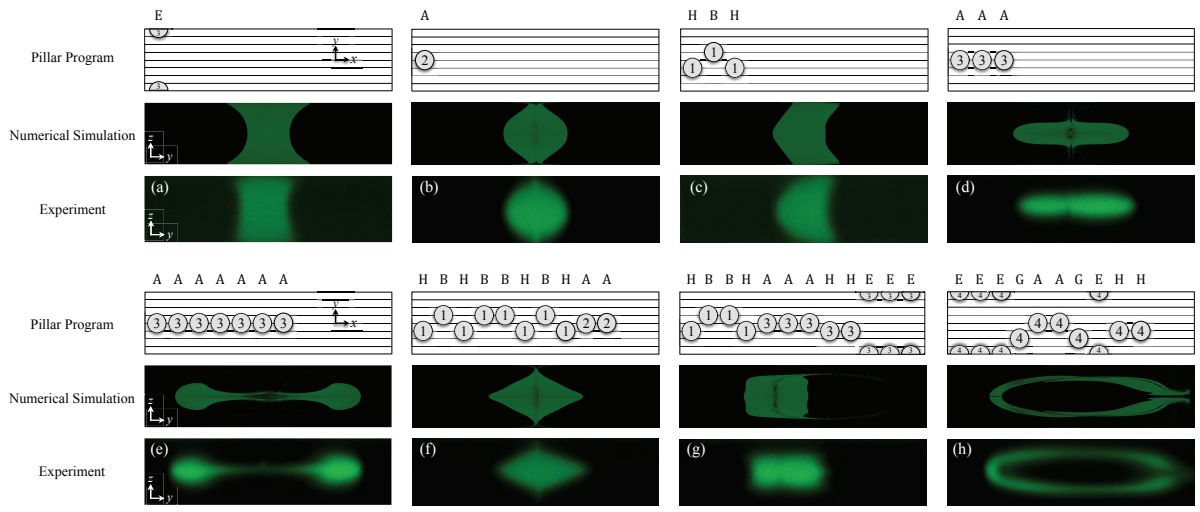

Figure 2.2 Fundamental transformations '*Make Concave*' (a), '*Make Convex*' (b), '*Tilt*' (c), '*Stretch*' (d), and Hierarchical transformations '*Split*' (e), '*Add Vertex*' (f), '*Shift*' (g) and '*Encapsulate*' (h). The figures show the pillar sequence schematic on top, the numerical prediction as an output from uFlow in the middle, and the experimental validation on the bottom.

2.5 Discussion

2.5.1 Pillar Sequence Concatenation

The exploration of pillar sequences and a subsequent search for target transformations was enabled by an understanding of how transformations can be used in sequence to appropriately sculpt flow. Because of the deterministic mapping of fluid elements upstream to downstream of the pillar sequences, the sequences themselves can be 'concatenated' end to end to create more complex programs. We identified four classes of concatenation, presented in order of simplicity:

- stacking
- recursion
- mirroring
- shaping

Stacking operations simply repeat the location and diameter of the prior pillar exactly, as seen in figure 2.3(a). It is typically seen to create a more exaggerated transformation of the previous

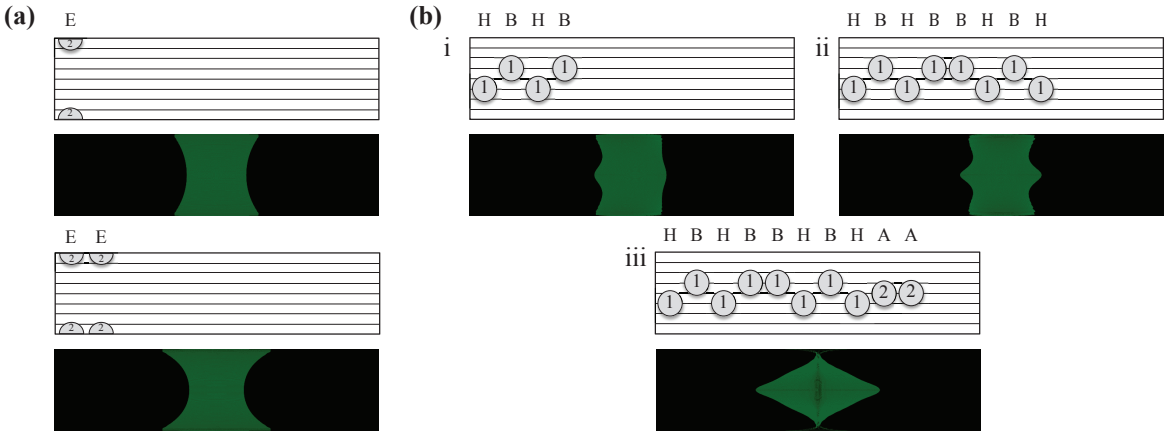

Figure 2.3 (a) Stacking demonstration with *make concave*. (b) Creation of *add vertex* via recursion (i), mirroring (ii), and shaping (iii).

pillar. A recursive operation is like stacking, but instead of a single pillar, a subsequence of pillars is repeated, as seen in the first pillar program of figure 2.3(b-i). A sequence of two pillars at positions H and B is repeated, thereby eliciting a more pronounced vertex on the left side of the midsection. Mirroring builds on recursion, except each location of the repeated subsequence is "mirrored" across the microchannel's longitudinal centerline, as demonstrated in figure 2.3(b-ii). Mirroring can be useful creating symmetric transformations from asymmetric functions, allowing shapes to have more complex edges. A clear example of this is the overall *add vertex* transformation: mirroring is used to form vertices, while each pillar deformation is based on secondary flows that contain no obvious sharp angles. Finally, shaping is simply an arbitrary concatenation of any subsequence to a different one, provided it is not stacking, recursive, or mirroring. An example is seen in figure 2.3(b-iii), where the known effect of *make convex* is used to create the net shape of *add vertex*. Shaping is a critical tool for discovering new transformations by building on a foundation of fundamental operations, for the user can attempt to sculpt a fluid packet in a way defined by an operation already seen, thus further defining the hierarchy of pillar programming.

2.5.2 Analysis of *Make Concave/Make Convex*

Make concave is formed by a basis of two half-pillars at the channel wall of diameters $D/w = \{0.375, 0.5, 0.625\}$ (figure 2.4). A diameter of $0.75w$ induces sharp corners in the fluid geometry, and is not effective in creating a smooth concave shape. Changing the pillar diameter changes the curvature of the fluid element, with a varying radius of curvature across the middle 1/3 of the contour as plotted in figure 2.4(b). Note the change in behavior with pillar diameters larger than $D/w = 0.375$; the 'longer' sequences tended to force more fluid to the edges of the channel, and created a more linear contour at the convex shape's midsection, thus increasing the radius of curvature for the majority of the shape. While it seems that the largest space to work while maintaining a truly convex shape is with $D/w = 0.375$, mass diffusion effects may be magnified by difficulty manufacturing smaller diameter pillars. Figure 2.2(a) shows a poor reproduction of the numerical prediction even with the modest pillar size of $D/w = 0.625$, suggesting the need for incorporating a diffusion model into the computational framework.

Make convex is a qualitative counterpart in output and program to *make concave*, and primarily uses a single pillar with diameters $D/w = \{0.375, 0.5, 0.625\}$. $D/w = 0.75$ results in an irregular hexagon at the output and therefore not satisfying the target operation requirements. Tunable convex and concave fluid shapes could provide a straight-channel alternative to Dean flow for the application of optofluidics, in addition to novel polymer shapes, or fluid focusing techniques. The average change in radius of curvature per number of pillars of similar diameter used is shown in figure 2.4(d). Again, it should be noted that the curvature shown is an average of the curvature for the middle 1/3 of the contour, as the profile is parabolic in nature, and becomes linear near the edge.

2.5.3 Analysis of *Stretch/Split*

The *stretch* transformation is accomplished by placing pillars on the center position of the channel. Though larger diameters can stretch the fluid element more rapidly - therefore requiring fewer pillars for a desired output - it is possible to overshoot the goal of a stretching deformation and move into the *split* transformation. *Stretch* is another example of how a

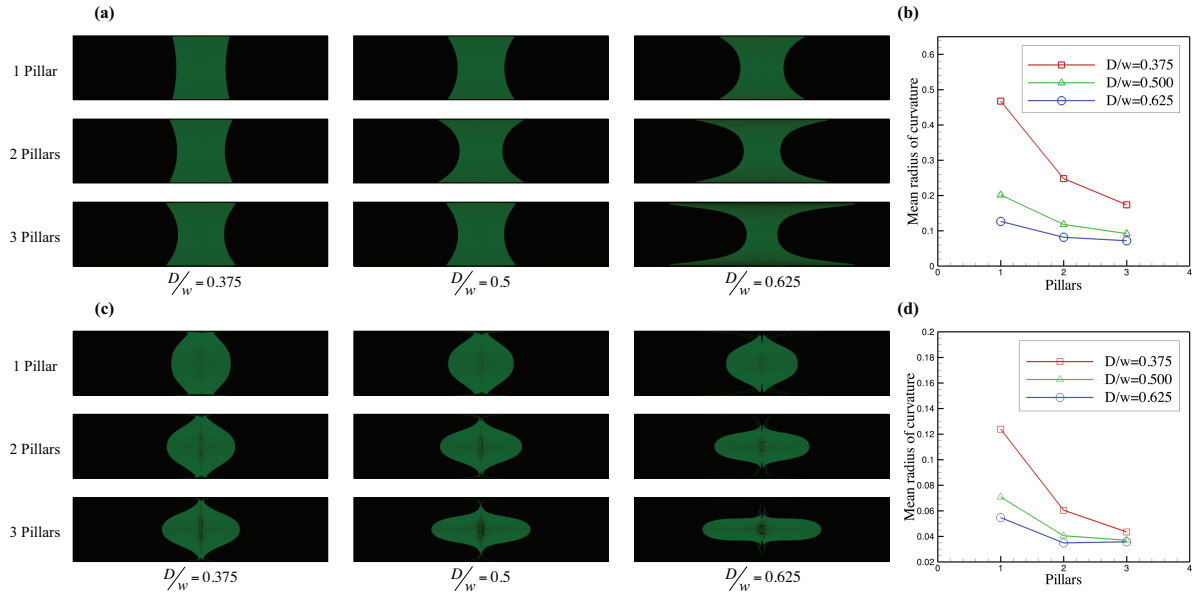

Figure 2.4 Illustrations of (a) *make concave* and (c) *make convex* per pillar type through 3 pillars, and changes in mean radius of curvature vs. number of similar pillars in sequence for (b) the *make concave* and (d) *make convex* operations.

transformation can be found through different pillar sequences. Figure 2.5(a) shows how a 5 pillar sequence with pillars of $D/w = 0.5$ (figure 2.5(a-i)) can be effectively recreated using a 3 pillar sequence with pillars of $D/w = 0.625$ (figure 2.5(a-ii)). Attempting to create this same transformation via 3 pillar sequence with $D/w = 0.75$ results in the beginning of the *split* transformation (figure 2.5(a-iii)), but still meets the goal of the *stretch* transformation. The variety of sequences for similar transformations shows the richness of the phase space, in addition to emphasizing the future need for numerically optimizing for minimal pillar sequences in order to mitigate diffusion effects. Figure 2.5(b) illustrates the effect of increasing the pillar diameter on the change in fluid width, per number of pillars in the sequence. The *split* transformation (figure 2.2(e)) is an example of how pillar program recursion can produce an entirely new operation. Placing at least seven pillars of $D/w = 0.625$ results in a separation of the fluid element into two roughly circular elements with diminishing tails on either side of the channel. Adding an eighth pillar begins to distort the separated elements until they begin to wrap back into the center of the channel, beginning a potential mixing scheme.

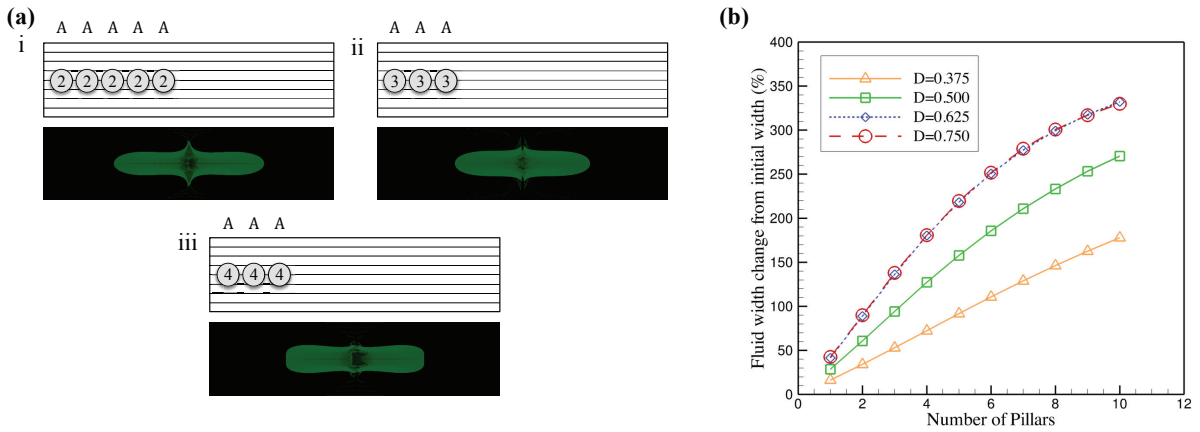

Figure 2.5 (a) Multiple routes to similar *stretch* transformations. Note that three pillars of $D/w = 0.625$ achieves roughly the same output as five pillars of $D/w = 0.5$. Moving to a $D/w = 0.75$ begins the *split* transformation sooner, in addition to flattened ends. (b) (Please view in color) The effect of changing diameter and increasing numbers of central pillars on the stretched width of the central fluid element. Note that $D/w = 0.75$ and $D = 0.625$ are functionally identical for this effect.

2.5.4 Analysis of *Tilt*

The *tilt* program shown in figure 2.2(c) creates a 38 ° tilt, with an internal angle of 104 °. *Tilt* forms the basis for several hierarchical operations, but there are very few ways to create *tilt* with the reduced framework. The sequence uses pillar diameters of $D/w = 0.375$, but moving to $D/w = 0.5$ removes the vertex in the transformation. Similarly, moving outward in lateral placement also removes the vertex. This motivates the future analysis of a larger dataset with fine-grained pillar diameters and locations, which would allow for sensitivity analysis to determine the degree of error allowed in manufacturing pillar sequences.

2.5.5 Analysis of *Add Vertex*

Different programs were found to arrive at the *add vertex* transformation, but the sequence shown in figure 2.3(b-iii) contains the sharpest vertices. There are two primary components to this sequence: the creation of an irregular decagon with vertices protruding at the channel centerline, and the use of the *make convex* program to complete the final shape. The irregular decagon is formed by mirroring a modified 38 ° *tilt* sequence that adds a vertex to one side

of the fluid structure, which results in two vertices symmetric about the channel's y-axis, as shown in figure 2.3(b-ii). The *make convex* program, 2 pillars with $D/w = 0.5$, is concatenated onto the previous eight pillars to finish the transformation. An alternate route is to decrease the mirrored sequence from four to two pillars. This creates a 5 pillar program, but alters the output edges to become more smooth and convex (see figure 2.6(b)).

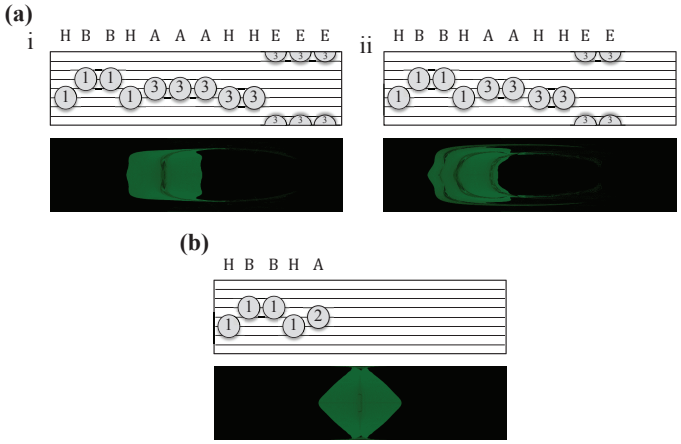

Figure 2.6 (a) 10% *shift* (i) and 20% *shift* (ii). Note that the program for 20% *shift* is concatenated onto the 10% sequence, for a total of 22 pillars. (b) Alternative 5 pillar approach to *add vertex*, demonstrating the same recursion/mirroring/shaping methods as the 10 pillar version.

2.5.6 Analysis of *Shift*

The initial strategy for shifting a fluid element was to begin with a *stretch* program and then force the fluid to either side of the channel with pillars to the left or right of the centerline. Thin 'tails' are formed in the most basic *shift* sequences, as seen in figure 2.6(a-i). These tails are a natural consequence of the secondary flow at the center of the channel, which - coupled with the no-slip boundary condition - results in an incomplete pinching effect at the wall/fluid interface. To marginalize this effect, the *add vertex* program was used to 'pre-treat' the collapsed fluid element, therefore minimizing the fluid at the wall/fluid interface that would end up creating the tails. Several methods were found for the *shift* transformation, with lateral migration of the central column varying from 10% to 20% of the full channel width (see figure 2.6(a-ii)), and varying degrees of 'squareness' in the final output. The basic transformation as

originally outlined seeks to preserve the shape at the microchannel inlet, and the programs shown demonstrate the best outcome to this effect.

2.6 Conclusions

We are able to identify a set of fundamental fluid transformations using rationally chosen sequences of pillars. This is made possible by quickly prototyping different pillar sequences in silico, and identifying those sequences that have the desired effect on the fluid shape. The identified transformations have potential applications in optofluidics, shaped fiber and particle fabrication, and mixing. The transformations also provide some intuition on the types of fluid deformations that can be performed using pillars in a channel.

The framework utilized on this study, although applied here only to pillars positioned in a straight channel, can be used to analyze the flow deformation resulting from the concatenation of arbitrary geometries, provided that the fluid is allowed to fully develop in between. Other geometries for consideration include channel expansion regions, curved segments, walls, asymmetric pillars, and channel bifurcations. The analysis can also be expanded to include a range of Reynolds numbers and channel aspect ratios, which were held fixed for this study.

Presently, the effect of diffusion on the fluid is ignored. Large numbers of pillars - and therefore longer channels - along with more complex fluid deformations will enhance mass diffusion further downstream, resulting in a blurred departure from uFlow's predicted transformation. Accordingly, mass diffusion will limit the size of structures that can be formed using this method, though diffusion of species across streamlines may be important for many applications. The current framework can be extended to account for diffusion by applying a simple local diffusion model to a fluid parcel as it traverses the channel. Techniques for enabling this analysis in real-time are under investigation.

Although the present study only follows fluid parcels as they traverse the channel, it is possible in principle to apply the same analysis to finite-sized particles suspended in the fluid flow, which may cross streamlines. Although it is more expensive to compute advection maps for finite-sized particles, this computation is performed offline, and does not affect the speed of the analysis.

Building off this initial work, future designers of continuous flow fluidic systems should be able to achieve a desired output behavior without significant rounds of experimental trial and error, paving the way for systems of increased complexity to address biomedical, materials fabrication, and industrial heat and mass transport problems.

CHAPTER 3. OPTIMIZATION OF MICROPILLAR SEQUENCE DESIGNS FOR FLUID FLOW SCULPTING

A paper accepted by *Physics of Fluids*

Daniel Stoecklein, Chueh-Yu Wu, Donghyuk Kim,
Dino Di Carlo, and Baskar Ganapathysubramanian

Abstract

Inertial fluid flow deformation around pillars in a microchannel is a new method for controlling fluid flow. Sequences of pillars have been shown to produce a rich phase space with a wide variety of flow transformations. Previous work has successfully demonstrated manual design of pillar sequences to achieve desired transformations of the flow cross-section, with experimental validation. However, such a method is not ideal for seeking out complex sculpted shapes as the search space quickly becomes too large for efficient manual discovery. We explore fast, automated optimization methods to solve this problem. We formulate the inertial flow physics in microchannels with different micropillar configurations as a set of state transition matrix operations. These state transition matrices are constructed from experimentally validated streamtraces for a fixed channel length per pillar. This facilitates modeling the effect of a sequence of micropillars as nested matrix-matrix products, which have very efficient numerical implementations. With this new forward model, arbitrary micropillar sequences can be rapidly simulated with various inlet configurations, allowing optimization routines quick access to a large search space. We integrate this framework with the genetic algorithm and showcase its applicability by designing micropillar sequences for various useful transformations. We computationally discover micropillar sequences for complex transformations that are sub-

stantially shorter than manually designed sequences. We also determine sequences for novel transformations that were difficult to manually design. Finally, we experimentally validate these computational designs by fabricating devices and comparing predictions with the results from confocal microscopy.

3.1 Introduction

The physics of inertial fluid flow deformation at the microscale has seen a surge of theoretical and experimental interest in the past decade(Amini et al., 2014). Use of the inertial flow regime ($1 < Re < 100$, with the Reynolds number $Re = \frac{\rho V D_H}{\mu}$, where ρ, V, and μ are the fluid density, average downstream velocity, and viscosity, and D_H the hydraulic diameter) in the microfluidics community contrasts the previously held notion that most practically useful flows were in the Stokes regime(Stone et al., 2004) ($Re \to 0$), though the effects of inertial fluid flow has been observed since the 1960s(Segré and Silberberg, 1961). New applications have since emerged, with inertial focusing in particular, leading to new methods for high-throughput cytometry(Bhagat et al., 2008; Oakey et al., 2010; Mach and Di Carlo, 2010; Goda et al., 2012; Hur et al., 2011; Hansson et al., 2012; Ciftlik et al., 2013).

More recently, fluid sculpting via inertial flow has been demonstrated through so-called "pillar programming" or "flow sculpting"(Amini et al., 2013; Stoecklein et al., 2014). See Fig 3.1 for a general schematic of pillar programming. In pillar programming, pillars spanning the height of a microchannel induce secondary flow which can be used in sequence with additional downstream pillars to generate a net deformation to the fluid. The idea of programmability comes from the spacing micropillars far enough apart to allow for their individual deformations to saturate. Thus, the flow sculpted by one pillar is used as an input to another pillar, without concern for cross-talk or time dependent effects. This allows for a powerful computational shortcut for future simulations, as the deformation for particular micropillar configurations need be computed only once, with the resulting fluid displacement forming a 2D "advection map". Net deformation for a sequence of micropillars can then be rapidly simulated using the pre-computed advection maps for each micropillar in the sequence. This obviates solving Navier-Stokes equations over large 3D domains for entire microfluidic devices.

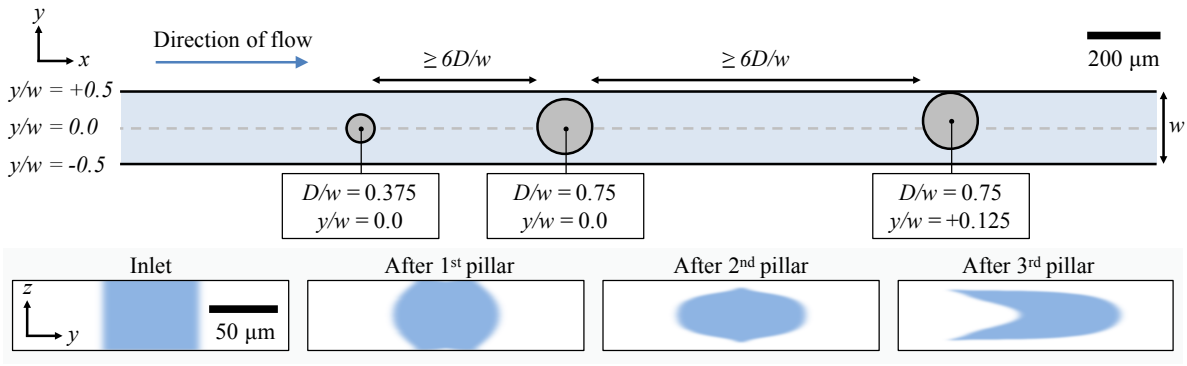

Figure 3.1 Schematic of a micropillar sequence (drawn to scale). Note the varying inter-pillar spacing based on the size of the preceding pillar, allowing the turning motion from each pillar's deformation to saturate before the flow arrives at the next pillar. All lengths are normalized to the microchannel width, w. Below are cross-sectional images of sculpted fluid based on the sheathed flow shown in the inlet image and the micropillar sequence as illustrated. The predictions come from the simulation method described in this work ($Re = 20$).

Amini et al. (2013) determined numerically that an inter-pillar spacing of 6 pillar diameters is sufficient to prevent upstream flow effects of one pillar from interacting with a previous pillar's deformation. In experiments, they use a spacing of 10 pillar diameters to ensure the pillar deformations are completely isolated. Therefore, each pillar in a designed microfluidic device will extend the length of the microchannel by a fixed amount, and its pillar programming simulation is valid only in the flow conditions for the pre-computed advection map.

With a coarse but broad design space for initial exploration, a set of widely varied transformations was found and experimentally validated using this method for forward simulation(Stoecklein et al., 2014). The designs used 1 to 10 micropillars, each chosen from the set of configurations shown in Fig 3.2. A user-friendly GPU based software platform "uFlow" was developed in tandem with this work and made freely available, enabling any researcher with modest computing power to manually test various micropillar sequence designs for their own purposes. uFlow is built on top of computationally expensive solutions to the Navier-Stokes equations(Stoecklein et al., 2014), but as a standalone product it uses lightweight, pre-computed advection maps, relieving the end-user of time consuming calculations. Using this framework, manual pillar programming has been successfully employed for shaping polymer precursors for

streams(Nunes et al., 2014) and particles(Paulsen et al., 2015; Wu et al., 2015), reducing inertial flow focusing positions (Chung et al., 2013), and solution transfer around particles (Sollier et al., 2015).

Such manual exploration of flow deformations will quickly run up against the combinatorial phase space of possible pillar combinations(Stoecklein et al., 2014). For example, consider creating pillar sequences where each pillar is chosen from a set of 32 possible pillars of varying size and transverse location(Diaz-Montes et al., 2014) (see Fig 3.2). For a 10-pillar sequence, there are $32^{10} \approx 10^{15}$ possible pillar sequences, in addition to multiple inlet fluid flow configurations. It becomes non-trivial (in terms of time and effort) to manually design a micropillar sequence by searching this phase space. Even if manual design is attempted, a more efficient sequence with fewer pillars might be possible. This is important in microfluidic devices, as the number of pillars in a sequence determines the length of the channel, and therefore the footprint of the device itself. Longer channels and more pillars increase the pressure required for operation, which can introduce device flexure, and exacerbate transverse mass diffusion that increases with fluid distance traveled (Ismagilov et al., 2000). Decreasing the channel length can not only reduce the pressure drop to improve the performance of the device, but also provide more space downstream for applications, for example, complex 3D shaped particle fabrication or particle separation and solution exchange (Nunes et al., 2014; Paulsen et al., 2015; Sollier et al., 2015; Wu et al., 2015). Therefore, an optimization scheme is needed to enhance the capability of pillar programming while still remaining accessible to, and easily implementable by, an interested researcher with modest computing hardware. Although pillar diameter will play a role in the required pressure to drive flow through a micropillar sequence, we have determined that microchannel length has a greater overall impact on pressure drop (more information available in Appendix A). Hence, the first priority in optimization is to find a flow shape that matches the target. This is the motivation for the work presented here. Secondary to this is the minimization of micropillar sequence length (determined primarily by the number of micropillars needed), which contributes advantages in terms of both reduced pressure drop and space for applications. Tertiary goals, such as optimizing for the effects of pillar diameter on pressure drop, are currently not taken into consideration for optimization.

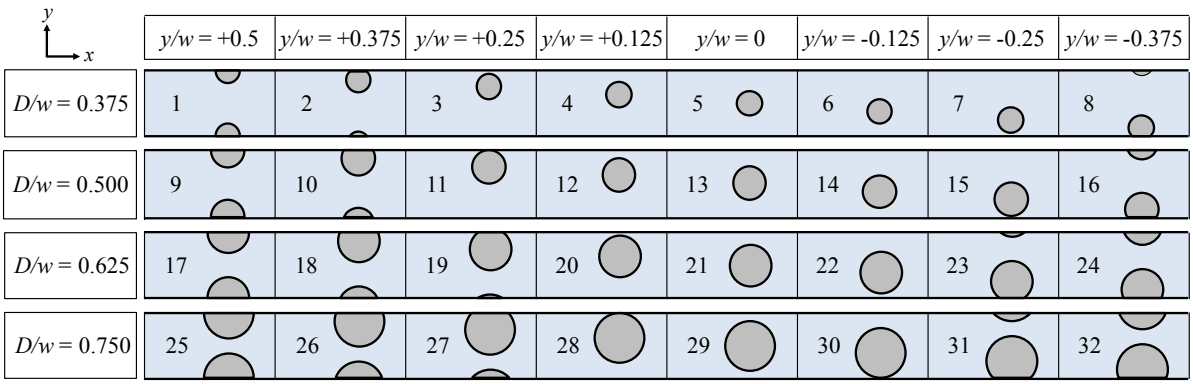

Figure 3.2 Set of previously used micropillar configurations (Amini et al. (2013); Stoecklein et al. (2014); Diaz-Montes et al. (2014)). Each pillar spans the height of the microchannel. Note the periodic boundary condition, for which a pillar being placed so close to the microchannel wall will have the merged volume protrude from the opposite wall. For this paper, each pillar configuration of diameter D/w and offset y/w has an integer index, which is shown next to each depiction of the micropillar.

We show two steps toward a framework for automated pillar sequence design: the development of an efficient forward model for rapid evaluation of arbitrary micropillar sequences, and the formulation of the design problem as an optimization problem. To accomplish these steps, we first convert the per-pillar advection maps to sparse transition matrices, which store fluid state displacement information as transition probabilities across states. These matrices can be multiplied together to quickly propagate net deformation across many pillars. This reduction of complex three-dimensional deformation simulations to numerically efficient matrix-matrix multiplication subsequently enables effectively pairing with a genetic algorithm for optimal design (Mott et al., 2009) of micropillar sequences. To demonstrate the effectiveness of this platform, we show the design and experimental validation of optimized micropillar sequences for previously discovered deformations, as well as novel designs which have no previously known pillar sequences.

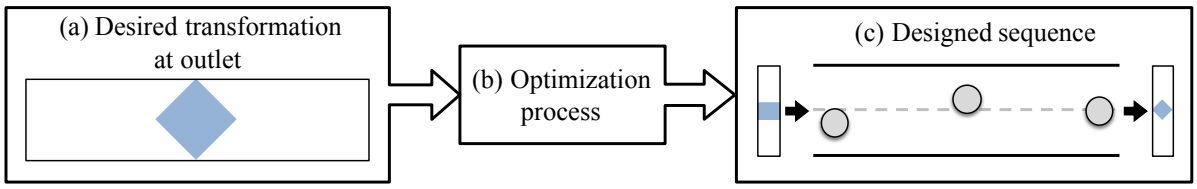

Figure 3.3 Illustration of the design problem, where a micropillar sequence is desired that will deform fluid into the given fluid flow shape (a). Some optimization routine (b) must determine a micropillar sequence (c) that yields a flow shape that closely matches the given fluid flow shape.

3.2 Problem formulation

The goal is to develop a computational framework that accepts a desired transformation as an input, and produces a micropillar sequence design for the nearest possible match to this transformation as an output (see Fig. 3.3). For this work, the simulation and fabrications parameters will remain within the constraints of pillar programming as previously defined.

3.2.1 Forward model

The design problem potentially requires evaluation of many thousands of different pillar sequences. We refer to each such evaluation as solving the forward model (in contrast to the design problem). A forward model must therefore be accurate and fast. We begin by creating advection maps from streamtrace data for various pillar configurations, which is computed from our validated in-house finite element Computational Fluid Dynamics (CFD) framework ((Amini et al., 2013; Stoecklein et al., 2014; Diaz-Montes et al., 2014)). We then translate the displacement information from the maps into transition matrices, which are an effective method for fast and accurate pillar program simulation.

3.2.1.1 CFD data to advection maps to transition matrices

We start with a dataset of 3D velocity fields calculated for a set of individual micropillar configurations at $Re = 20$ (See Fig 3.2). The Navier-Stokes equations are solved (using the finite element method) in a domain that spans six diameters upstream and downstream of the

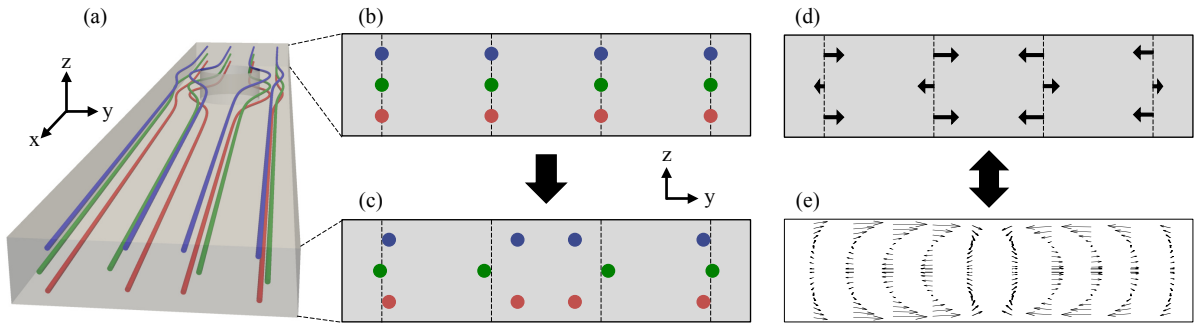

Figure 3.4 (a) Streamtraces through a 3D velocity field are used to form advection maps by comparing inlet (b) and outlet (c) positions of infinitesimal massless, neutrally buoyant particles (in this case, $N_Y = 4$, $N_Z = 3$ particles are used). (d) The resulting advection map shows the net secondary flow for a particular pillar configuration and flow condition, which has resolution determined by the number of particles used. (e) Shows a representative realistic 2D advection map.

pillar. A detailed description of this procedure is provided in our earlier work(Stoecklein et al., 2014; Amini et al., 2013) and is not repeated here for the sake of brevity. Each 3D velocity field is contracted to a 2D advection map by streamtracing uniformly distributed, neutrally buoyant particles through the velocity field (see Fig. 3.4(a)).

For enhanced numerical efficiency, our idea is to make pillar programming amenable to using the BLAS (Basic Linear Algebra Subprograms) libraries within the CPU and/or GPGPU (General Purpose computation on Graphics Processing Units). BLAS libraries are freely available software that are architecture-aware, utilize memory optimally, and are specifically pipelined and tuned for fast matrix-matrix operations. These libraries are platform agnostic, and are usually efficiently implemented on most CPUs and GPUs. Adapting pillar programming to such matrix-matrix operations allows for utilization of a growing repertoire of highly efficient optimization methods.

We convert the displacement information contained in an advection map into a sparse "transition matrix". This is accomplished by first discretizing the advection map into a finite set of N cells, and subsequently identifying where each cell advects to (or is displaced to). Consider an advection map constructed by using $N = N_Y \times N_Z$ particles that are uniformly distributed across the cross-section (with N_Y particles along the y direction, and N_Z particles

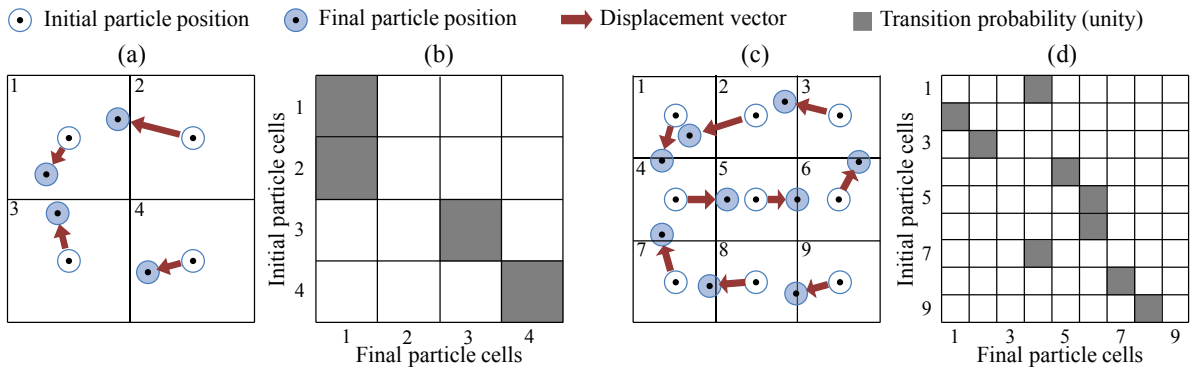

Figure 3.5 (a) Mapping of a 4 particle advection map with $N_Y = 2$, $N_Z = 2$ to a 4×4 transition matrix (b). Each row corresponds to one cell. For each row, the column that is darkened represents the destination cell. Increasing the number of particles in the advection map results in a more resolved transition map as shown in (c) and (d).

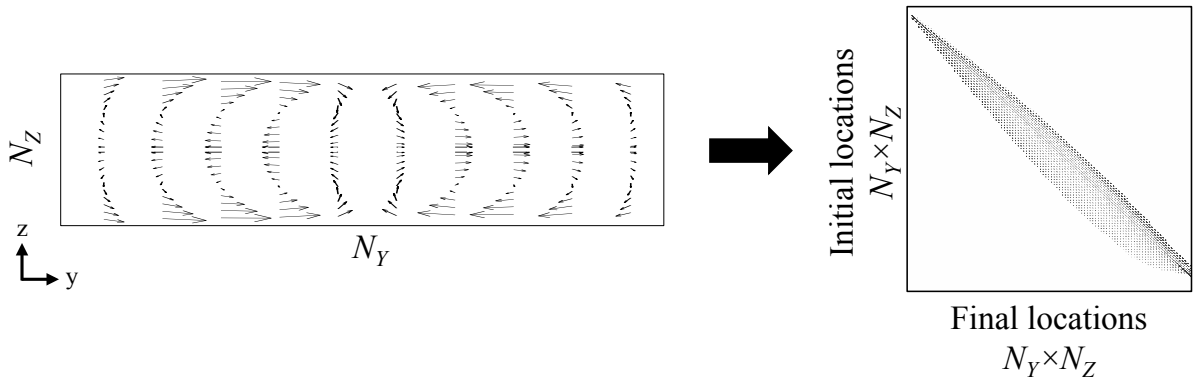

Figure 3.6 Advection maps converted to transition matrices. This matrix is sparse. The figure shows the non-zero entries of the matrix in black.

along the z direction). We discretize the channel cross-section into $N = N_Y \times N_Z$ cells, with each cell center displacement given by the corresponding particle displacement from the advection map. This is illustrated in Fig. 3.5(a) and Fig. 3.5(c).

The transition matrix is a square matrix with N rows and columns. Each row of a transition matrix accounts for the displacement information of one fluid cell. Specifically, a row stores an individual cell's displaced location index (i.e. the index of the new cell where this cell lands) as indicated by the advection data. Examples of such transition matrices are illustrated in Fig. 3.5(b) and Fig.3.5(d). These examples show *forward advection*, which transition fluid from

cells at the channel inlet to the outlet. Otherwise, *reverse advection* will show the history of fluid cells at the outlet, pulling them forward from the inlet. Both of these approaches result in a sparse matrix that is easily stored and manipulated. Fig. 3.6 illustrates the sparsity of a representative transition matrix.

Once the transition matrices for individual pillar transformations have been computed, the net deformation caused by an arbitrary sequence of pillars is easily computed as the matrix product (in sequence) of the corresponding transition matrices. This is schematically shown in Fig. 3.7. In this example, the inlet flow shape is represented as a vector μ_{inlet}. The fluid is transformed by a sequence of three pillars, which have transition matrices P_1, P_2, and P_3, respectively. Then, the outlet flow shape μ_{outlet} is given by

$$\mu_{outlet} = \mu_{inlet} P_1 P_2 P_3 \tag{3.1}$$

Thus, the prediction of fluid flow shapes from arbitrary micropillar sequences has been reduced from computationally expensive CFD to simple sparse matrix multiplication[1]. The key advantages of the formalism are the speed with which flow shapes can be predicted (see Fig. 3.8(a)), and the subsequent simplicity of integrating this with a multitude of optimization frameworks. Furthermore, sparse matrices require far less memory than a dense matrix of similar size, since only the non-zero values are stored (along with their locations in the matrix). In our case, note that each cell is displaced to a single cell (a binary transition). Thus, the framework optimally deploys sparsity with the number of non-zero values in each row equal to 1. This reduces computational overhead. In the next section, we explore the accuracy of this transition matrix representation as a function of discretization.

3.2.2 Implementation

Consider a transition matrix constructed from a discretization of the microchannel cross-section into $N_Y \times N_Z$ uniform cells. The cells have dimensions $\frac{w}{N_Y} \times \frac{h}{N_Z}$, but are effectively represented as points in the transition matrix. As such, subtle behavior of streamtraces will be truncated to displacements that align with the discretized representation. For example,

[1]This conceptual framework can be readily extended to account for multiple fluid species by creating separate inlet vectors for each fluid type.

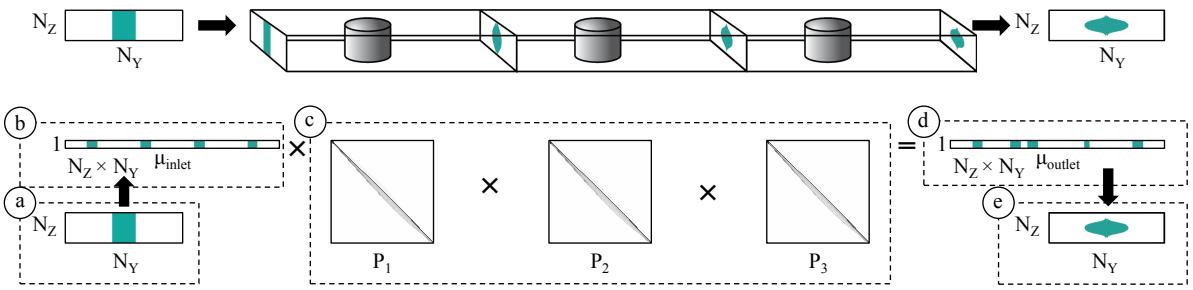

Figure 3.7 Illustration of how transition matrices P_1, P_2, and P_3 can be used to simulate the net deformation from three individual micropillars. First, the inlet flow condition (a) is reshaped to a row vector μ_{inlet} (b) with a length matching the dimension of the square transition matrix. This vector is then multiplied by the product of the transition matrices (c), which forms an outlet row vector μ_{outlet} (d). This can then be reshaped into the original microchannel dimensions (e), giving the sculpted fluid flow shape.

fluid movements too small to leave their immediate cell space will not register in the transition matrix (see cell 1 in Fig. 3.5(a)). Similarly, movement just large enough to arrive in a new cell will be considered as completely displaced to this cell center (see cell 2 in Fig 3.5(a). The precision of the transition matrix is therefore limited by the level of discretization of the cross-section into cells. Increasing the number of cells (larger N_Y, N_Z) accounts for smaller displacements, thereby leading to a more accurate mapping onto the transition matrix (see Fig. 3.5(c)). However, larger discretization results in longer computational times. Fig. 3.8(a) plots the time needed for a single matrix multiplication of transition matrices constructed with increasing cell discretization. A doubling of discretization (along both N_Y and N_Z) increases the computational time by a factor of around 5 in Matlab running on a 2.0 GHz 8-Core Intel E5 2650. The 801×101 and 1601×201 based discretizations take about 2 milliseconds, and 10 milliseconds, respectively, which result in reasonable run times for the optimization framework.

We next explore the representation error that these discretizations produce, especially as a function of nested matrix-matrix products. Note that experimental and mass diffusion constraints (please see Appendix A) limit the maximum number of pillars in a sequence to 10. We evaluate the representation error by comparing the predictions of nested matrix products with those of the advection maps for arbitrary pillar sequences of increasing complexity. Error is

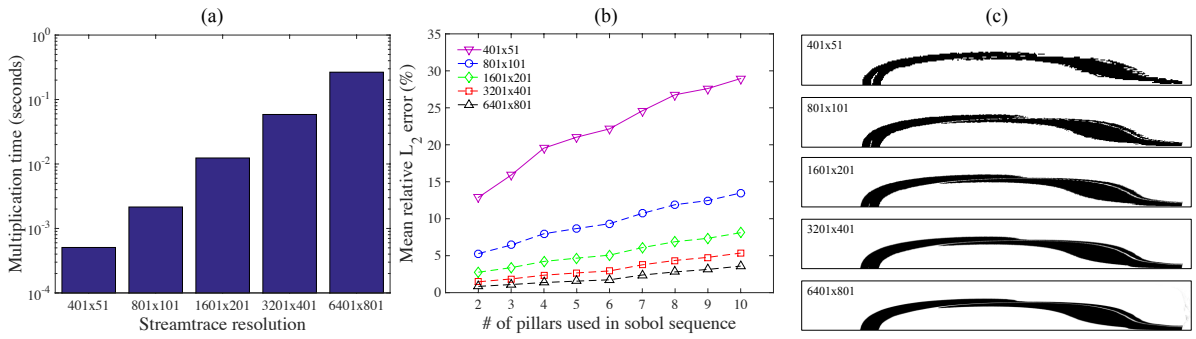

Figure 3.8 (a) Log-scale time for the sparse matrix-matrix product of two random sparse matrices, averaged over 1,000 multiplications, per resolution of transition matrix. (b) Comparison of truncation error vs number of matrix multiplications, averaged over a 100 sample sobol sequence for each number of possible pillars. (c) A 10-pillar outlet flow shape for different levels of streamtrace discretization. Note that these images only show the top half of the microchannel cross-section.

computed by first converting the transition matrices back into 2D advection maps, and comparing these to the streamtraced advection maps. Thus, for an arbitrary streamtraced advection map $\mathcal{A}$ and matrix-reconstructed map $\mathcal{A}_\mathcal{P}$:

$$\text{error} = \frac{\|(\mathcal{A}_\mathcal{P} - \mathcal{A})\|_2}{\|\mathcal{A}\|_2} \tag{3.2}$$

We create many sequences of pillars and compute the error. These sequences are constructed using a quasi-random sobol number generator (which is a low discrepancy generator) for maximal coverage of the phase space of pillar combinations. The mean error over 100 realizations is plotted in Fig. 3.8(b). From these plots, we chose to utilize a $N_Y = 801, N_Z = 101$ discretization for subsequent analysis, as it provides a good balance between accuracy and computational overhead. We finally validate this discretization by comparing the prediction of the transition matrix framework with experimental confocal images (see Section IV B,C) of three flow transformations. Fig. 3.9 illustrates this promising comparison.

3.3 Design problem

With an efficient pathway for quickly evaluating arbitrary sequences of micropillars, we formulate the problem of identifying micropillar sequences for user-defined transformations.

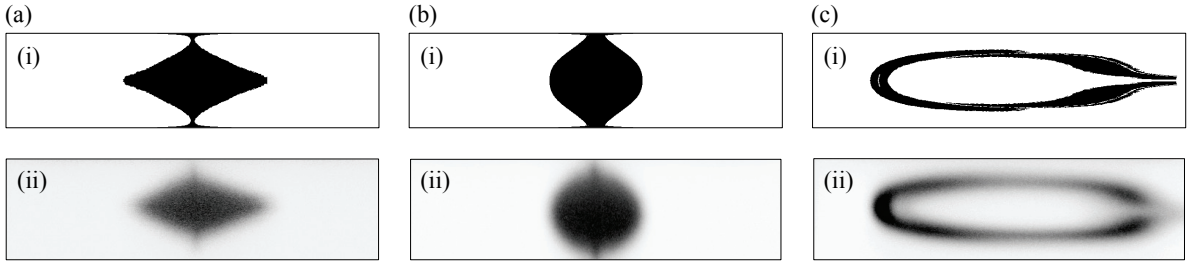

Figure 3.9 Flow deformations *add vertex* (a), *make convex* (b), and *encapsulate* (c) as predicted by matrix multiplication (i), with experimental confocal images (ii).

We formulate the design problem as an optimization problem. Specifically, given a target outlet shape I_{Target}, we define a cost functional $C(I_{Target}, I_{Test}(\mathbf{s}))$ that returns a scalar value representing how closely a test shape, $I_{Test}(\mathbf{s})$, matches the target shape. $I_{Test}(\mathbf{s})$ is the output transformation of an arbitrary sequence of pillars $\mathbf{s} \equiv \{s_1, s_2, ..., s_k\}$. Each pillar, s_i, is chosen from a set of possibilities which are defined in Fig. 3.2. The optimization problem is defined as

$$\underset{\mathbf{s}\equiv\{s_1,s_2,...,s_k\}}{\operatorname{argmin}} \quad C(I_{Target}, I_{Test}(\mathbf{s})) \tag{3.3}$$

where the pillar sequence $\mathbf{s}$ that minimizes the fitness function is found.

There are several approaches to solve this optimization problem. Here, we choose to utilize a gradient free, evolutionary optimization strategy. The rationale behind this choice is motivated by the following:

1. The problem formulation as defined here is inherently discrete. This is due to the underlying experimental constraints. Fabrication of arbitrarily sized pillars at continuous locations within practical tolerance and cost is not viable. Effective manufacturing tolerances benefit from a well defined set of micropillar configurations, rather than a continuous space of micropillar diameters and offsets. Simulating a significant subset of this infinite library is also not practical. Nevertheless, the discrete set from which each pillar is chosen makes the possible search space countably large, hence, precluding an exhaustive search.

2. Though it is difficult to illustrate the phase space a 10-pillar sequence offers, a cost function (developed later in this work) evaluated for 2-pillar sequences (with each pillar chosen from 32 possible configurations as shown in Fig. 3.2) is shown in Fig. 3.10. This is a highly corrugated surface, with one global optimum but many nearly identical local minima. This precludes the utilization of gradient based methods, and instead suggests the applicability of stochastic, multistart methods that can explore the phase space efficiently.

3. The discrete problem of finding a sequence of matrices such that their product matrix has a desired structure is a variant of the minimum length generator sequence problem (Even and Goldreich, 1981)which has been shown to be a NP-hard problem. This necessitates the utilization of (meta)heuristic optimization methods for efficient solutions.

These issues naturally suggested using gradient-free metaheuristic evolutionary search algorithms. We specifically use the genetic algorithm (GA) to locate optimal sequences. GAs are well suited to multi-modal, highly corrugated solution spaces, especially when the cost function is not easily adapted to gradient-based methods(Mohammadi and Pironneau, 2004; Giles and Pierce, 2000; Holland, 1992).

3.3.1 Genetic Algorithm

The GA has been used with substantial success in fluid mechanics with applications ranging from efficient macro- to nano-fluid heat exchanger design(Foli et al., 2006; Wang et al., 2011; Yang et al., 2014), bluff body flow control(Milano and Koumoutsakos, 2002), indoor airflow geometry(Xue et al., 2013), and aircraft design(Ahuja et al., 2014). Though there are many different flavors and modifications of the GA, the basic algorithm is generally as follows:

1. Initialize a population of randomly generated points in the search space. This is the first generation of points. The points are encoded (usually as a binary bitstring) and are called "chromosomes".

2. Evaluate each chromosome of the current generation using a fitness function (cost function) specific to the problem.

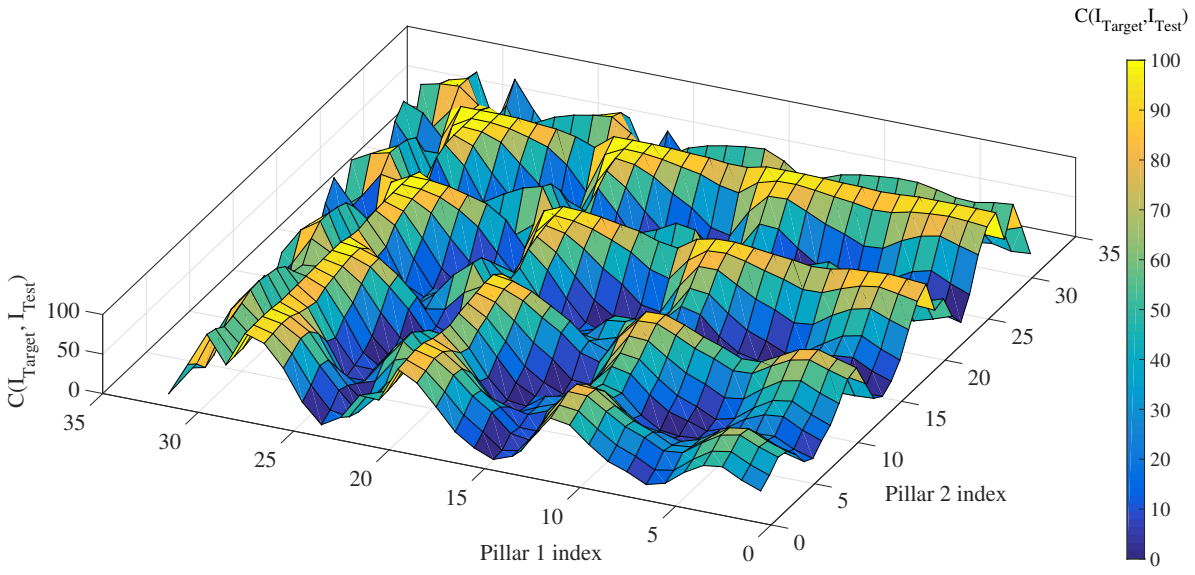

Figure 3.10 This figure shows the design space for a 2-micropillar target fluid flow shape. The topology will change depending on the target, and quickly become far more complicated in n-dimensions for n-micropillar designs. Note the multi-modal, corrugated nature of this simple 32×32 space.

3. Select chromosomes based on their fitness to create a new generation of individuals through crossover and mutation methods. Good fitness is rewarded by increased chance of selection, which will propagate genetic material to the next generation. There are three basic methods of creating the next generation:

   - Crossover combines the genetic material in individuals' chromosomes (design parameters) to create offspring that should retain some of the traits of the parents. A crossover rate determines the proportion of each new generation that should come from crossover.
   - Mutation randomly alters bits within a chromosome. This introduces random variation into the population, which can bring candidate solutions out of local optima.
   - Top-performing chromosomes in the population are chosen as Elites, whose chromosomes are preserved in the next generation. This retains the optimal solution throughout the GA.

4. Repeat steps 2-3 until some set of termination criteria are met, which typically include the following:

    - Stall generation limit: If mean or optimal fitness does not improve for a set number of generations, terminate the GA.
    - Generation limit: If the number of generations reaches this value, terminate the GA.
    - Stall time: if the GA runtime meets this value, terminate the GA

From the final generation, a single optimal solution is selected by choosing the most-fit individual (chromosome). For this work, we formulate the GA as a minimization problem, making lower fitness more desirable. Because GAs deploy a population of potential solutions distributed over the design space, they are less prone to getting stuck in shallow local minima. GAs are an inherently stochastic method, so we repeat each optimization multiple times (10 times) to consider statistical significance of results and attempt to reliably explore the phase space.

There are two primary design choices for implementing the GA: chromosome design and choice of the fitness function. The chromosome used here simply consists of pillar sequences for a fixed number of pillars, with each chromosome bit having an integer value as shown in Fig 3.2. So, the first generation is a set of completely random pillar sequences of a fixed length. During the evaluation stage of the GA, the previously described forward model creates a fluid flow shape for each chromosome, which is then compared to the target image in the fitness function.

The choice of the fitness function critically determines the success of the optimization procedure. Since the primary goal of this exercise is to capture the overall shape of the transformation, the fitness function should be a measure of the topology of the shape. Extensive numerical experiments suggested that the (image) correlation coefficient (defined below) gives substantially better results than several standard pixel based norms, while remaining competitive with respect to computational speed. The correlation function is a per-pixel measure, pinning flow shape searches to the regions that the target image has fluid within. However, given the non-linear nature of inertial flow deformations, there is not guarantee that a desired

flow shape can be found in any given region - if at all. Future work could incorporate more complex fitness functions.[2] The correlation coefficient r, is defined as

$$r(I_{Target}, I_{test}) = \frac{\sum_{i}^{N_Y}\sum_{j}^{N_Z}(I_{test} - \overline{I_{test}})(I_{Target} - \overline{I_{Target}})}{\sqrt{(\sum_{i}^{N_Y}\sum_{j}^{N_Z}(I_{test} - \overline{I_{test}})^2)(\sum_{i}^{N_Y}\sum_{j}^{N_Z}(I_{Target} - \overline{I_{Target}})^2)}} \tag{3.4}$$

This measure is extensively used to determine how similar two images are. Identical images would result in $r = 1$, while comparing images with precisely complementary pixels would result in $r = -1$. In the latter case, the result is still desirable, as the overall shape has been achieved using fluid in complementary inlets. That is, the "empty" co-flow inlets have been shaped to align with the target fluid locations in the desired fluid flow shape. The user would simply need to "flip" the inlet flow configuration in order to create their desired microfluidic device. We can treat both optimal values of r equally by squaring its value. Thus, the fitness function C takes the form

$$C = f(I_{Target}, I_{test}) \tag{3.5}$$

$$C = 100 \times (1 - r(I_{Target}, I_{test})^2) \tag{3.6}$$

We achieved better convergence by pre-processing I_{Target} and $I_{test}(\mathbf{s})$ before evaluating the fitness function. Specifically, a low-pass filter applied to both I_{Target}, and $I_{test}(\mathbf{s})$ emphasized the large scale (topological) features of the shapes while de-emphasizing the fine scale features. This is shown in Fig. 3.11. When used with the correlation function, images similar in shape will generally have a better cost functional than without the low-pass filter.

3.4 Experimental Verification of Designed Sequences

3.4.1 Fabrication

All simulations and experiments follow the pillar configurations and flow conditions used in our previous work(Stoecklein et al., 2014), with a microchannel of height h, width w, mi-

[2] Considering the importance of cost function selection with regards to optimization algorithms, we emphasize that it will be valuable to investigate additional cost functions (specifically wavelet and Fourier transform based multiscale feature functions).

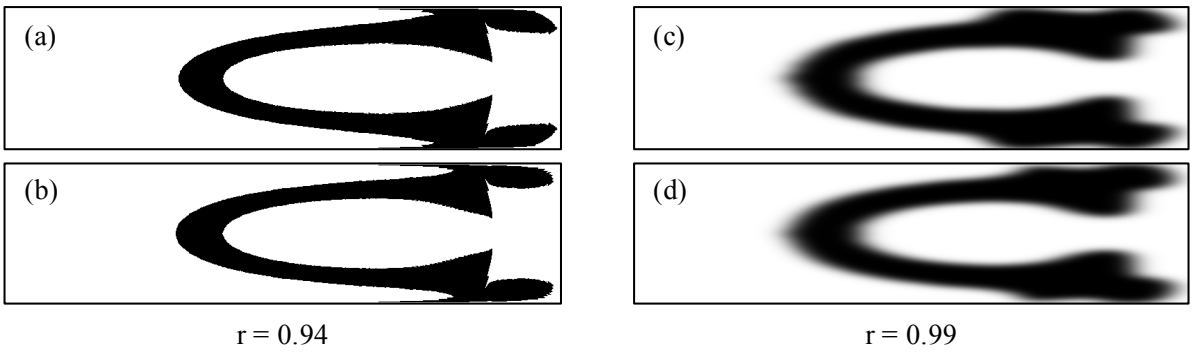

Figure 3.11 Outlet images of 6-pillar (a) and 4-pillar (b) sequences as predicted by transition matrices. Note that despite their overall similarity, minute discrepancies throughout the shape result in a correlation coefficient of $r = 0.94$. When pre-processed by a low-pass filter, as shown in (c) and (d), the correlation coefficient $r = 0.99$ relays a more useful comparison of the bulk shapes.

crochannel height-to-width aspect ratio $h/w = 0.25$, and flow conditions $Re = 20$. The micropillar configurations are shown in Fig 3.2, with indices $1 - 32$ corresponding to different combinations of normalized pillar diameter $D/w = \{0.375, 0.5, 0.625, 0.75\}$, and normalized offset from the center of the channel $y/w = \{0.5(\times 2), 0.375, 0.25, 0.125, 0, -0.125, -0.25, -0.375\}$. The offset location of $y/w = 0.5(\times 2)$ consists of two half-pillars, located at both sides of the channel. In fabricated devices, each pillar was spaced approximately 10 pillar diameters apart to ensure deformation saturation. For verification of the optimized designs, microfluidic chips (200μm x 50μm) incorporating the designed pillar sequences were fabricated using soft photolithography. The molds corresponding to the channel design were fabricated from a silicon master spin-coated with KMPR 1050 (MicroChem Corp.) and then patterned by standard photolithography. Polydimethylsiloxane (PDMS) base and curing agent (Sylgard 184 Elastomer Kit, Dow Corning Corporation) were mixed at a ratio of 10 to 1, poured onto the molds in petri dishes, put in a vacuum to remove bubbles, and cured in an oven to replicate the structure of the microchannels. The PDMS devices were peeled from the mold and punched with holes at the inlet and outlet, and bonded with a glass slide to enclose the microchannel after activation using air plasma (Plasma Cleaner, Harrick Plasma). The PDMS devices were filled with Rhodamine B (Sigma-Aldrich), which infused into the PDMS to visualize the channel walls.

3.4.2 Confocal Imaging

Confocal images of the fluid flow deformation for optimized designs were taken downstream of the fabricated pillars using a Leica inverted SP1 confocal microscope at the California NanoSystems Institute. For each design, three syringes on separate syringe pumps (Harvard Apparatus PHD 2000) were connected to the inlets of the microchannel using PEEK tubing (Upchurch Scientific Product No. 1569). For visualization of the deformed stream, the middle flow stream contained fluorescein isothiocyanate dextran 500kDa (5 μM, Sigma-Aldrich) while the side streams contained deionized water. The total volume flow rate was 150 μL/min, with the flow rate of each stream proportional to its cross-sectional area before the first pillar in the design. The confocal images in the cross-sectional plane were taken at least four times of pillar diameter downstream of the last pillar when the flow was fully developed (about 10 minutes after starting to pump). For each measurement, random noise was eliminated by averaging six images to arrive at a final image.

3.5 Results

3.5.1 Improved efficiency of existing designs

Three of the hierarchical designs from our earlier manual design work (Stoecklein et al., 2014) were selected to demonstrate the potential to create more efficient pillar sequences for existing transformations. These are the *encapsulate, shift*, and *add vertex* transformations (see Fig. 3.13(a-c,i-ii)). All three deform the same inlet flow configuration shown in Fig. 3.7(a) to entirely different flow shapes. The *encapsulate* transformation envelopes a co-flow fluid stream with the primary central stream. One application of *encapsulate* is seen in a similar shape devised in (Nunes et al., 2014), which formed a new cross-section of a fabricated micropolymer. *Shift* translates the entire fluid stream across the channel. Solution transfer(Sollier et al., 2015), enhanced heat transfer, and mixing are examples where *shift* may see application. The *add vertex* transformation stands apart due to its sharp vertices at the midsection of the fluid, which may defy expectations of pillar deformation. *Add vertex* could be the start of interlocking fluid structures, and hints to more complex polygonal fluid flow shapes.

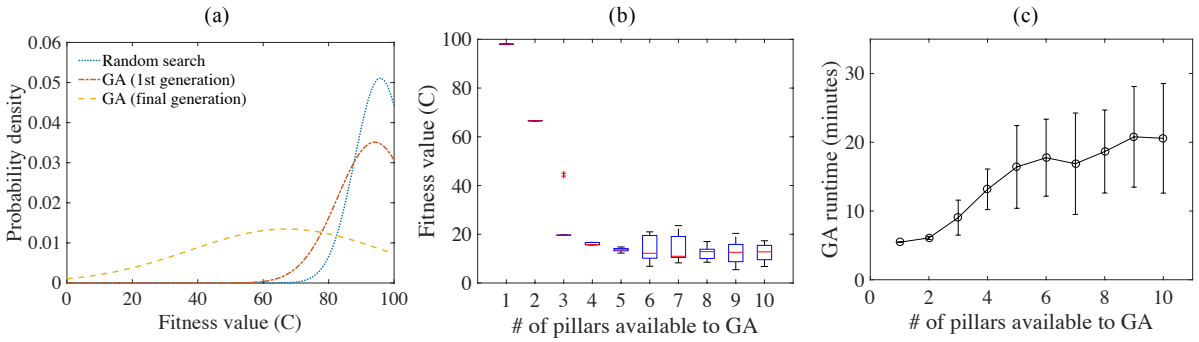

Figure 3.12 (a) Probability density functions for fitness function evaluations for a random search, and the first and last generations of a genetic algorithm. The random search is based on 10,000 randomly generated sequences with 10 pillars, while the genetic algorithm evolves from an initially random population of 100 10-pillar sequences. Here, the target flow shape was *encapsulate*. The optimal GA solution had a fitness value $C = 6.76$, while the best random solution was $C = 21.56$. (b) Boxplots of optimal fitness values for 10 genetic algorithm trials per number of pillars used in the algorithm chromosomes. Note that the spread of fitness values tends to widen with a larger number of pillars available to the GA, which corresponds to the increasingly complex design space being searched. (c) Mean runtime for the genetic algorithm based on the low-pass post processed fitness function (see Fig. 3.11). Error bars are the standard deviation for 10 trials per number of pillars available to the genetic algorithm.

GA optimization found multiple designs for each target transformation, with the shortest sequences and simulated flow shapes shown in Fig. 3.13(e-f,i-ii). See Fig. 3.2 for pillar configurations corresponding to the indices in the sequence figures. The new designs resulted in 33-60% improvement to sequence length while maintaining the overall desired shape and location in the microchannel. These designs were fabricated and the pillar transformations were evaluated using confocal images. The experimental results agree well with the predictions as seen in Fig. 3.13(d-f,iii). The improvement by computational optimization also reduced device footprint and pressure by approximately the same percentage in experiments.

All simulations were performed using a Matlab implementation of the sparse matrix operations. We deployed the parallel GA routine available in Matlab on computing clusters. GA parameters included a population size of 100, crossover rate of 0.8, mutation rate of 0.2, and 5 elites. Example behavior of the algorithm is shown in Fig. 3.12, showing optimum fitness

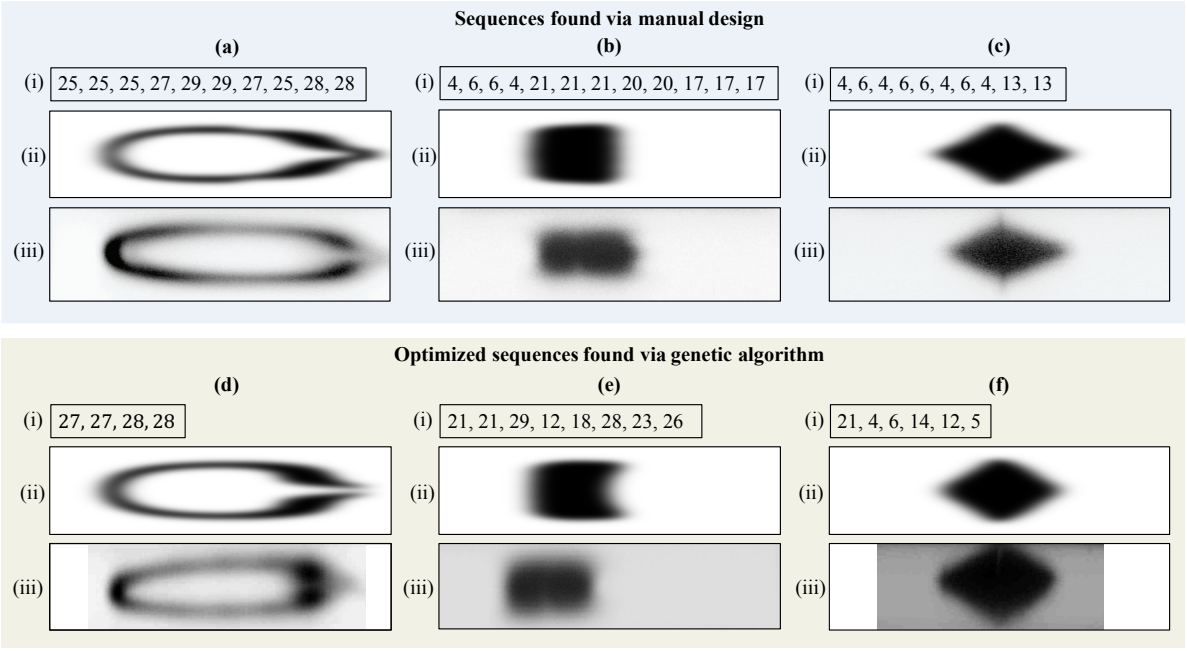

Figure 3.13 Optimization of the 10 pillar *encapsulate* (a), 12 pillar *shift* (b), and 10 pillar *add vertex* (c) transformations resulted in 4, 8, and 6 pillar optimized sequences (d-f). Pillar indices for each sequence (i) are found in Fig. 3.2. Pre-processed transition matrix simulations are seen in (ii), and confocal images of deformed fluid from fabricated devices are shown in (iii).

functions and runtime for the optimization of the *encapsulate* flow shape. We see good support for using GAs by comparing their evolved generation's fitness functions to a computationally equivalent random searches. The GA will evolve 100 initially random pillar sequences for approximately 100 generations (with an upper limit of 200), so we evaluated 10,000 randomly generated pillar sequences. Fig 3.12(a) shows fits of normal distributions to the results of these searches, with the GA results having clear improvement over the random search. The final generation for the GA found an optimum fitness of $C = 6.76$, while the random search's best fitness was $C = 21.56$. The GA typically converges on the same optimal shape for smaller design spaces (see Fig 3.12(b), for # of pillars = 1-3). Searches using pillar sequences with a length in excess of 5 pillars find a variety of local optima, which is unsurprising given the larger, more complex search space. Runtime was as much as 30 minutes, but on average the total run of 10 trials for 10 pillar sequence lengths took about 24 hours on a 2.0 GHz 8-Core

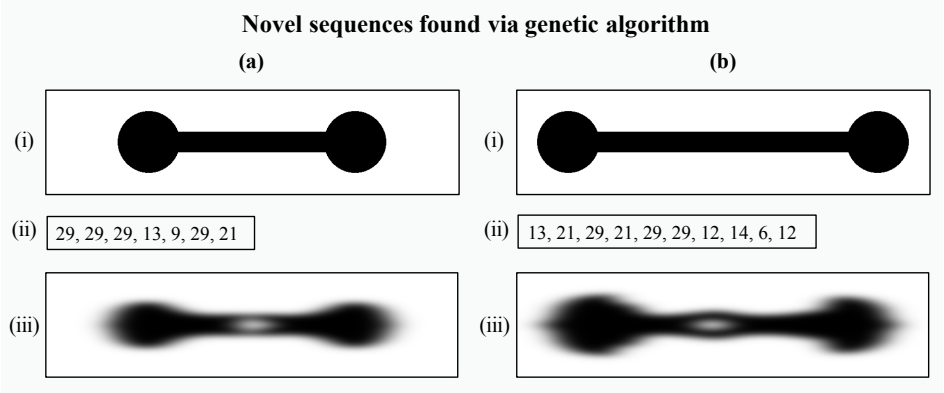

Figure 3.14 Optimization targeting novel "dumbbell" shapes (a,i) and (b,i), with genetic algorithm solution sequences (ii) and post-processed flow predictions (iii). For shape (b,iii), the inlet flow shape spans a width of $\frac{w}{3}$. Pillar indices can be found in Fig. 3.2.

Intel E5 2650 CPU.

3.5.2 Novel designs

We next applied the framework to design novel flow transformations. These transformations change the shape of the central fluid into shapes akin to a dumbbell. The dumbbell shape was motivated by experimental particle fabrication research, where polymer precursors are shaped by a micropillar sequence and subsequently polymerized by UV light. Use of a shaped mask allows for a single section of flow to be polymerized, rather than the entire stream as in (Paulsen et al., 2015). Unlike circular or ellipsoid shapes, the dumbbell offers a more complex geometry that could be applied to create particles that align along one axis in a flow(Uspal et al., 2013). The targets used here differ with respect to the width of the dumbbell, which will show the sensitivity in the choice of a desired shape.

The GA was run with several different inlet configurations in order to allow for additional fluid. Results are in Fig. 3.14, with the wider dumbbell shape requiring an inlet flow shape spanning $\frac{w}{3}$.

3.6 Summary

We have devised and validated an efficient forward model for simulating pillar programming via simple matrix multiplication, and implemented the model into a parallel GA. The simple nature of the new forward model is valuable for its speed and applicability. Optimization routines can now incorporate flow shape prediction directly into a computational pipeline. We have shown how the GA successfully used this new framework by optimizing for known flow shapes, and producing new micropillar sequence designs for novel flow shapes. Future work on design for pillar programming could pursue faster and more exhaustive searches by tailoring the process to more specialized hardware (e.g., the GPU), and investigating new fitness functions.

Overall, this work completes the cycle moving from a user-defined flow shape of interest to a physical implementation of a channel design that achieves this flow shape. Such an approach opens up the computer-aided design and manufacturing of shaped polymer fibers(Nunes et al., 2014) and particles(Paulsen et al., 2015) for a range of applications. Additional uses in directing mass and heat transfer, and transferring solutions for automation of biological sample preparation should also benefit. In comparing to experimental and numerical iteration in which the full Navier-Stokes equations are solved for a set of complex channel geometries, our approach achieves orders of magnitudes improvements in time to result, making previously intractable problems that were not even attempted now possible.

CHAPTER 4. AUTOMATED DESIGN FOR MICROFLUID FLOW SCULPTING: MULTI-RESOLUTION APPROACHES, EFFICIENT ENCODING, AND CUDA IMPLEMENTATION

A paper accepted by *Journal of Fluids Engineering*

Daniel Stoecklein, Michael Davies, Nadab Wubhset,
Jonathan Le, and Baskar Ganapathysubramanian

Abstract

Sculpting inertial fluid flow using sequences of pillars is a powerful method for flow control in microfluidic devices. Since its recent debut, flow sculpting has been used in novel manufacturing approaches such as microfiber and microparticle design, flow cytometry, and biomedical applications. Most flow sculpting applications can be formulated as an inverse problem of finding a pillar sequence that results in a desired fluid transformation. Manual exploration and design of pillar sequences, while useful, have proven infeasible for finding complex flow transformations. In this work, we extend our automated optimization framework based on genetic algorithms (GA) to rapidly design micropillar sequences that can generate arbitrary user-defined fluid flow transformations. We design the framework with the following properties: (a) a parameter encoding that respects locality to ensure fast convergence, and (b) a multi-resolution approach that accelerates convergence while maintaining accuracy. The framework also utilizes GPU architecture via NVIDIA's CUDA for function evaluations. We package this framework in a user friendly and freely available software suite that enables the larger microfluidics community to utilize these developments. We also demonstrate the framework's capability to rapidly design arbitrary fluid flow shapes across multiple microchannel aspect ratios.

4.1 Introduction

Manipulating fluid flow in microchannels using a sequence of cylindrical obstacles (pillars) has been recently shown to be a powerful method for passive fluid flow control (Amini et al., 2013; Stoecklein et al., 2014). Previous methods of microfluidic fluid flow control were generally applied to fluid mixing (Amini et al., 2013), and used chaotic flow transformations with limited capability to purposely sculpt flow (Lepchev and Weihs, 2010; Zhou et al., 2015), or required active control (Wang et al., 2007). This new technique, known as "pillar programming", uses pillars that span the height of a microchannel to deform fluid. The basis of pillar programming is the fore-aft asymmetry in fluid streamlines around a single pillar in inertial flow, with $1 < Re < 100$ (where $Re = \frac{UD_H}{\nu}$ for a fluid kinematic viscosity ν, average downstream velocity U, and channel hydraulic diameter D_H). The inertial force present in the fluid causes it to be deformed as it moves past a pillar, unlike Stokes flow ($Re \approx 0$, a flow regime typical of microfluidic devices), where the fluid will return to its location in the microchannel cross-section. Furthermore, for $Re < 100$, fluid flow around a confined pillar is still highly laminar and predictable (Amini et al., 2013). Thus, pillars of varying diameter and position will each produce a unique and time-invariant flow deformation for similar fluid flow conditions. Pillars can then be placed in a sequence, with each pillar's individual deformation acting as a building-block toward a net deformation to the fluid at the microchannel outlet (see Fig. 4.1 for an overview of a pillar programming microfluidic device).

If micropillars in a sequence are placed far enough apart in the streamwise direction to suppress possible cross-talk (spacing $\geq 6D/w$ for a pillar diameter D and microchannel width w (Amini et al., 2013)), the fluid flow deformation from each pillar can be independently simulated and used as a nested function. In other words, each pillar accepts the deformed flow shape from the previous pillar as an input for their own deformation. This provides a useful computational shortcut for simulating arbitrary sequences of micropillars, as time-invariant deformations for single micropillars need only be simulated once. Thus, a library of pre-computed fluid flow deformations for various micropillar configurations and flow conditions can be assembled from computationally difficult simulations for the Navier-Stokes equations

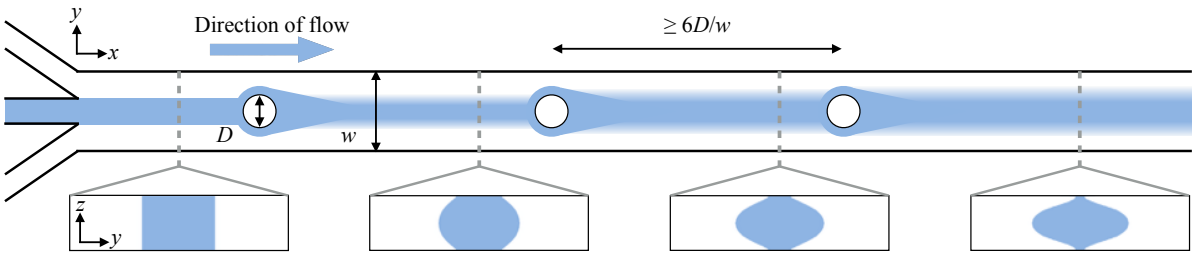

Figure 4.1 Illustration of a 3-pillar flow sculpting device with a 3-channel inlet. The middle channel contains the fluid that is "sculpted" by the micropillar sequence (colored blue). In this sequence, each pillar has the same normalized diameter D/w, and is at the same location y/w in a microchannel of height h. With sufficient inter-pillar spacing ($\geq 6D/w$, for $Re \leq 40$), the fluid deformation from each pillar has time to saturate before reaching the next pillar. Cross-sectional images showing the inlet flow condition and fluid deformations after each pillar are shown below the microchannel. These images are created by the same simulation method as by Stoecklein et al. (2016), which is used in this work as well.

over a 3-D domain for each micropillar. These simulations form state transition matrices, which relay the displacement information of the fluid within the cross-section of the microchannel from inlet to outlet of each pillar domain (Stoecklein et al., 2016) (see Appendix A B). Subsequently, a fluid flow simulation for a microfluidic device consisting of an arbitrary sequence of pillars is well represented by piecing together each pillar's pre-computed individual deformation, rather than simulating the device's entire 3-D microchannel domain. Such computation is remarkably fast on modest consumer hardware, with typical runtime being well under 1 second for pillar sequences of reasonable length (Stoecklein et al., 2016). This method of simulation has been experimentally validated in microfluidic devices in (Amini et al., 2013; Stoecklein et al., 2014, 2016).

Previous work incorporated this nested representation into a fast, user-friendly, and cross-platform utility for manual design called uFlow (Stoecklein et al., 2014). With an experimentally identified set of 4 micropillar diameters and 8 lateral positions in the channel (for a total of 32 different micropillar configurations, which are easily manufacturable), a variety of possible fluid flow shape transformations have been predicted and experimentally validated for an inlet configuration (Stoecklein et al., 2014). These manually designed micropillar sequences

have since been used to tailor polymer fiber cross-sections (Nunes et al., 2014), create 3-D particles (Paulsen et al., 2015; Wu et al., 2015; Paulsen and Chung, 2016), and for solution transfer around particles in flow (Sollier et al., 2015). Such applications are still quite fundamental and broad in their scope, and the full potential of pillar programming in research and industry remains untapped. However, the immense phase space for pillar combinations and inlet configurations makes manual design of complex fluid structures practically difficult and time consuming. For example, in choosing from the 32 different pillar configurations as used in (Stoecklein et al., 2014), a 10-pillar sequence has approximately $32^{10} \sim 10^{15}$ different permutations. Hence, the sheer number of possible flow shapes via pillar programming is well beyond manual exploration. This motivates the need for a computational framework that solves the inverse problem, which is: given a desired fluid flow shape, what is the micropillar sequence and inlet design that produces the closest possible match?

Stoecklein et al. (2016) tackled the inverse problem in pillar programming by using an off-the-shelf Genetic Algorithm (GA)[1], which has been previously used to optimize geometry for fluid flow applications (Foli et al., 2006; Lee et al., 2010; Ahuja et al., 2014; Lyutov et al., 2015). The encoding of the design variables (a "chromosome" in the GA) was a set of integers, each of which acted as an index representing a micropillar configuration (i.e., a micropillar of some diameter and position for set flow conditions and channel geometry). The GA uses a cost function, referred to as a fitness function, with which to evaluate chromosomes during optimization. Stoecklein et al. (2016) used a correlation function to compare flow shapes created by the GA to a target fluid flow shape. The inlet flow design was manually specified by the user and held constant during optimization.

Though this work was successful in optimizing pillar sequences for known transformations, we identified computational limitations that preclude widespread use. These can be categorized into three issues: long execution time, the need for user-supplied inlet design, and use of proprietary software. A single pillar sequence design search took as long as 24 hours. The requirement for manual inlet flow design meant that multiple user-guided searches were often necessary for a single image. That is, an initial guess at what the flow should be at the

[1]Stoecklein et al. (2016) discuss the rationale of using a meta-heuristic, evolutionary approach to this problem

inlet frequently had to be altered, especially since the fluid can undergo considerable lateral displacement. Hence, multiple iterations on the supplied inlet configuration are often required before a satisfactory result is found (see Fig. 4.2(a)). Additionally, the framework used Matlab's built-in GA code. Beyond being commercial software, Matlab uses interpreted code which is generally slower in execution than compiled code such as C++. Reducing GA runtime would have several benefits, the simplest of which is ease of use. Moreover, with faster operation comes the opportunity to repeat searches and use larger populations, which will more effectively search the design space. These issues motivated the development of a powerful, robust, and fast framework for pillar programming design that vastly improves on these problem areas and provides a roadmap to future enhancements. There are five key contributions in this work: use of multi-resolution transition matrices, design of a real-valued GA chromosome, incorporating both inlet and pillar sequence design into the framework, adapting the fitness function to NVIDIA's CUDA, and the development of a freely available, open source utility for automated pillar programming design.

4.2 Problem definition and baseline GA

The forward model (defined in (Stoecklein et al., 2016)), which is used to formulate the inverse problem, uses sparse transition matrices to efficiently store and compute per-pillar fluid displacement information (see Appendix B). A sequence of pillars is represented as a nested product of the appropriate transition matrices. Transition matrices can be multiplied with each other per some arbitrary sequence, forming a new transition matrix for the net displacement of fluid passing through said sequence of pillars. This matrix is multiplied with a row vector representing the fluid states at the inlet in order to simulate the fluid states (flow shape) at the outlet. Mathematically, for an inlet image (vectorized as μ_0) and a sequence of k transition matrices, $\{M_i\}_{i=1}^{k}$, a simulated output image (vectorized as I_{sim}), is found by:

$$I_{sim} = \mu_0 \prod_{i=1}^{k} M_i \tag{4.1}$$

We limit the possible values of pillar diameter and location to $D/w = [0.375, 0.5, 0.625, 0.75]$ and $y/w = [-0.375 : 0.125 : 0.5]$ (where $y/w = 0.5$ is two half-pillars on either side of the

channel), for a total of 32 different pillar configurations (as in Stoecklein et al. (2014, 2016)). This defines a set $\mathcal{P} = \{M[1] \dots M[32]\}$ from which each pre-computed transition matrix in a sequence M_i can be selected.

The inverse problem in pillar programming is to find some sequence of pillars, $\{M_i\}_{i=1}^{k}$, and a corresponding inlet design, μ_0, such that the resulting flow deformation matches a desired flow shape. Given this forward model, we formulate the inverse problem as an unconstrained optimization problem:

$$\min f(I_T, \mu_0 \prod_{i=1}^{k} M_i) \tag{4.2}$$

$$\text{s.t. } M_i \in \mathcal{P}$$

Where f is a functional that quantifies the difference between the target image I_T and I_{sim}. This design space has been shown to be highly diverse, with a wide variety of possible fluid flow shapes (Stoecklein et al., 2014, 2016). Not only do many possible fluid flow shapes exist, but there are also multiple micropillar sequences that will produce practically similar flow shapes (Stoecklein et al., 2014). However, some solutions (micropillar sequences) for a given fluid flow shape may be less desirable, as longer fluid flow sequences - which require longer microchannels - can be problematic. Longer microchannels require higher pressures to drive flow, potentially leading to flexure in the channel, which will in turn alter geometry and therefore the sculpted flow shape. Moreover, longer channels allow for additional transverse mass diffusion in the fluid streams, which will blur and distort the sculpted flow shape. Finally, smaller micropillar sequences require less of a physical footprint on a lab-on-a-chip, which makes for more simple fabrication and multiplexing/parallelization, and provides room for additional microscale components. Consequently, it is often worthwhile to allow for fewer micropillars during optimization, rather than targeting only the complex spaces that come with longer micropillar sequences.

The finite cardinality of $\mathcal{P}$ provides a discrete set of choices. The inverse problem will therefore have a discrete input space. This, coupled with the corrugated, non-convex phase space, makes continuous approaches impractical. On the other hand, evolutionary heuristics such as the GA are well suited for such a problem (Holland, 1992). The GA operates by

evolving an initially random population of candidate solutions through a process of selection, crossover, and mutation, until some set of termination conditions are met. Each candidate solution is represented in the GA as a chromosome: a sequence of bits that is evaluated using a problem-specific fitness function. Selection is a mechanism for choosing pairs of chromosomes in the population that will crossover segments of their genetic material, creating offspring for the next generation that share traits from their parents. Chromosomes with good fitness are more likely to be selected to pass on their genetic code, while those with poor fitness tend to be washed out of the population. Those not selected for crossover still have a chance of being mutated. Mutation randomly alters bits of a chromosome, which can kick candidate solutions out of local optima into new areas of the search space. There is also an option to tag high-performing chromosomes as "elites". If elites are used in a GA, the most-fit individuals within each generation pass on to the next generation untouched. This preserves the current best solution over the course of the GA. The process of selection, crossover/mutation, and evaluation will repeat for many generations with the overall population fitness improving. Termination criteria typically include a measure of convergence, e.g., some number of stall generations for which the best fitness in the population does not improve, as well as a generation cap. The final generation will contain a most-fit chromosome, which is the optimized solution for the problem.

In the current work, we seek to carefully improve the computational performance of our existing GA framework (Stoecklein et al., 2016) and extend its functionality in a user-friendly way. We will refer to the framework in (Stoecklein et al., 2016), which uses Matlab's built-in genetic algorithm, as a *baseline* for comparison in terms of speed, accuracy, and scope. The *baseline GA* was formulated with a chromosome that used a sequence of integer values 1-32 as indices for a set of pre-computed transition matrices. The inlet flow design was user-defined. Each pillar sequence design involved executing the *baseline GA* through a user-defined number of allowed pillars in a sequence, repeating each search at least 10 times (see Algorithm 1). The search was repeated in order to attempt sufficient coverage of the design space in using this stochastic evolutionary heuristic algorithm. Users employing the GA for flow sculpting would tend to increase the number of repetitions, especially for larger pillar sequences in the

chromosome, to make sure their search gives statistically meaningful results (i.e., it is clear that nothing significantly better can be found for the given search parameters).

Algorithm 1 Baseline GA Framework.

Inputs:
I_T = Target flow shape image
μ_0 = Inlet flow design
$N_{P,start}$ = Initial pillar sequence length
$N_{P,stop}$ = Final pillar sequence length
Outputs:
$M_{i,Opt}$ = Optimal pillar sequence
$I_{sim,Opt}$ = Optimal flow shape image
f_{Opt} = Optimal fitness value

procedure Baseline
$f_{Opt} = \infty$
for $N_P = N_{P,start}$ **to** $N_{P,stop}$ **do**
    for $N_{GA} = 1$ **to** 10 **do**
        $f, M_i, I_{sim} = GA(I_T, \mu_0, N_P)$
        if $f < f_{Opt}$
            $f_{Opt} = f$
            $M_{i,Opt} = M_i$
            $I_{sim,Opt} = I_{sim}$
return $M_{i,Opt}, I_{sim,Opt}, f_{Opt}$

function $GA(I_T, \mu_0, N_P)$
Create random population of chromosomes
while not terminated **do**
    Calculate f(population)
    Select crossover, mutation, and elite chromosomes
    Perform crossover and mutation on population
Select most-fit f, M_i, I_{sim}
return f, M_i, I_{sim}

The fitness function was defined for a target image I_T and GA-produced image I_{sim}, each of dimensions $N_Y \times N_Z$ pixels, as

$$f = 100 \times (1 - r(I_T, I_{sim})^2) \tag{4.3}$$

$$r(I_{sim}, I_T) = \frac{\sum_i^{N_Y} \sum_j^{N_Z} (I_{sim} - \overline{I_{sim}})(I_T - \overline{I_T})}{\sqrt{(\sum_i^{N_Y} \sum_j^{N_Z} (I_{sim} - \overline{I_{sim}})^2)(\sum_i^{N_Y} \sum_j^{N_Z} (I_T - \overline{I_T})^2)}} \tag{4.4}$$

The value of the correlation coefficient r varies between 1 (for exactly similar images) and -1 (for precisely complementary images), both of which are acceptable. We square the value of r, thereby treating both values equally. We formulate our GA as a minimization problem, meaning that perfectly matching flow shapes give $f = 0$ (optimal fitness), and worse matches tend toward $f = 100$.

The baseline GA (Stoecklein et al., 2016) post-processed flow shape images generated by the forward model (along with the target image) with a low-pass filter based on the Fast Fourier Transform (FFT). This was done to compare coarse features of the targets, such as overall shape and size. Small discrepancies between shapes are generally irrelevant, given that real-world flow shapes blur due to mass diffusion. Use of the low-pass FFT mimics this blur, making practically similar shapes effectively the same. However, we find that this step is not necessary for a successful GA search, as the correlation function used in f still effectively compares flow shapes without the need for post-processing. This results in a decrease in fitness function execution time, but a degradation in reported optimal fitness due to the "raw" output of the transition matrix method being compared to what is likely a smoothly drawn user-supplied target shape.

Other parameters used in the baseline GA include a population size of 100 (5 elites), 200 maximum generations per GA search, scattered crossover, and tournament selection. Separate searches were conducted for a single target image using different numbers of pillars in the chromosome. Specifically, the framework swept through a user-specified number of pillars $N_{P,start}$ to $N_{P,stop}$, and repeated the entire search 10 times (see Algorithm 1). We match these parameters in our comparisons to the baseline.

4.3 Design of a robust, accelerated optimization framework

4.3.1 Use of multi-resolution simulations

(Stoecklein et al., 2016) showed how higher resolution streamtraces through the 3-D pillar domain contribute to a more accurate representation of the sculpted fluid. In the conversion to transition matrices, less information from the fluid advection is lost due to a more refined

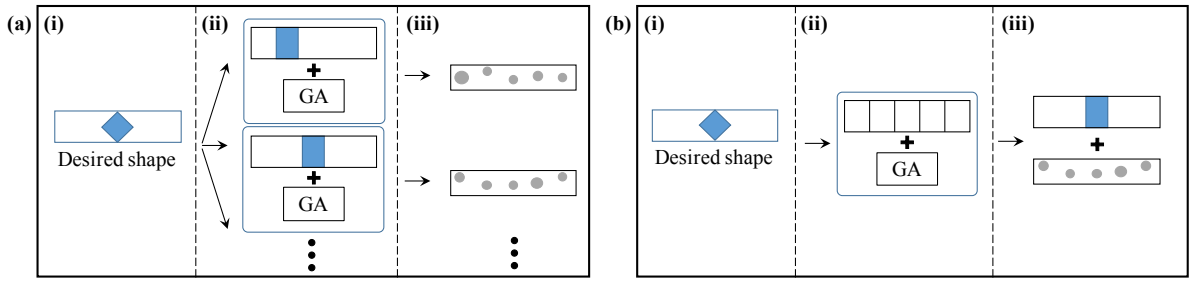

Figure 4.2 A comparison of the workflow necessary for pillar programming design based on the baseline GA framework (a), and the one presented in this paper (b). Previously, the user needed to specify both a desired flow shape (a)(i) and the inlet flow configuration (a)(ii). This could require multiple user-guided iterations on inlet configurations before a solution is found (a)(iii). Now that the inlet is a part of the GA chromosome the user simply supplies the GA framework their desired image (b)(i) and the number of possible channels at the microchannel inlet (b)(ii). The solution chromosome (b)(iii) contains the optimized inlet design as well as the pillar sequence.

grid onto which the fluid displacement is mapped. Nonetheless, a coarse streamtrace resolution can still relay useful information for bulk fluid movement. More importantly, smaller transition matrices require fewer matrix algebra operations, which reduces runtime.

We leverage this idea in two ways. First, we performed a study to determine the coarsest streamtrace sampling that maintains parity in accuracy with the baseline resolution as used in (Stoecklein et al., 2016). Then, we test a multi-resolution GA that uses approximate (but extremely fast) low resolution transition matrices in conjunction with high resolution matrices during the GA. The smaller, low resolution data (M_L) guide the population to several basins of attraction. From there, high resolution transition matrices (M_H) continue the search with greater respect for detail in the optimized flow shape. Thus, a single GA is modified during evolution as in Algorithm 2. This is an example of the so-called "multi-scale" GA, which has been used to improve the rate of convergence (Kim et al., 2008; Babbar and Minsker, 2006). We recognize that the choice of generation switch and resolution is a complete study unto itself, and defer an exhaustive study to a future publication. Here, we demonstrate the possibilities that even non-optimized values of resolution and generation switch produce in terms of significant computational gains.

Algorithm 2 Multi-resolution GA

Inputs

Gen_{switch} =Generation at which $M_L \rightarrow M_H$

Parameters:

Gen = current generation

function Multi-Res GA($I_T, \mu_0, N_P, M_L, M_H$)
while not terminated **do**
    if $Gen < Gen_{switch}$
        use $f = f(M_L)$ {Fast, approximate}
    Else
        use $f = f(M_H)$ {Slow, accurate}
    Calculate f(population)
    Select crossover, mutation, and elite chromosomes
    Perform crossover and mutation on population
Select most-fit f, M_i, I_{sim}
return f, M_i, I_{sim}

4.3.2 Expanding the design space to include inlet design

The chromosome used in the baseline GA (Stoecklein et al., 2016) contained information pertaining only to a pillar sequence. However, the complete forward model for simulated fluid flow must include some inlet flow design. We define this as number of channels at the inlet, and which channels contain the fluid being sculpted (see Fig. 4.1). As such, for a more complete search of the design space using the previous chromosome, the user would specify different inlet conditions for multiple design searches (see Fig. 4.2). This is an inefficient method of design which is exacerbated by the number of possible inlet configurations and the need for *a priori* knowledge of the design space. A user may end up running dozens of GAs, yet still miss an optimal solution. We address this by encoding the design of the inlet into the chromosome, thereby allowing the GA itself to tune the complete pillar geometry and fluid flow design *in-silico*.

We utilize a simple scheme for encoding the inlet into the chromosome. We specify a fixed number of channels that join together at the inlet, the limit of which will usually depends on microfluidic fabrication capability (e.g., Fig. 4.1 shows 3 channels being joined at the inlet). The GA then allocates the same number of "bits" in the chromosome to correspond with the

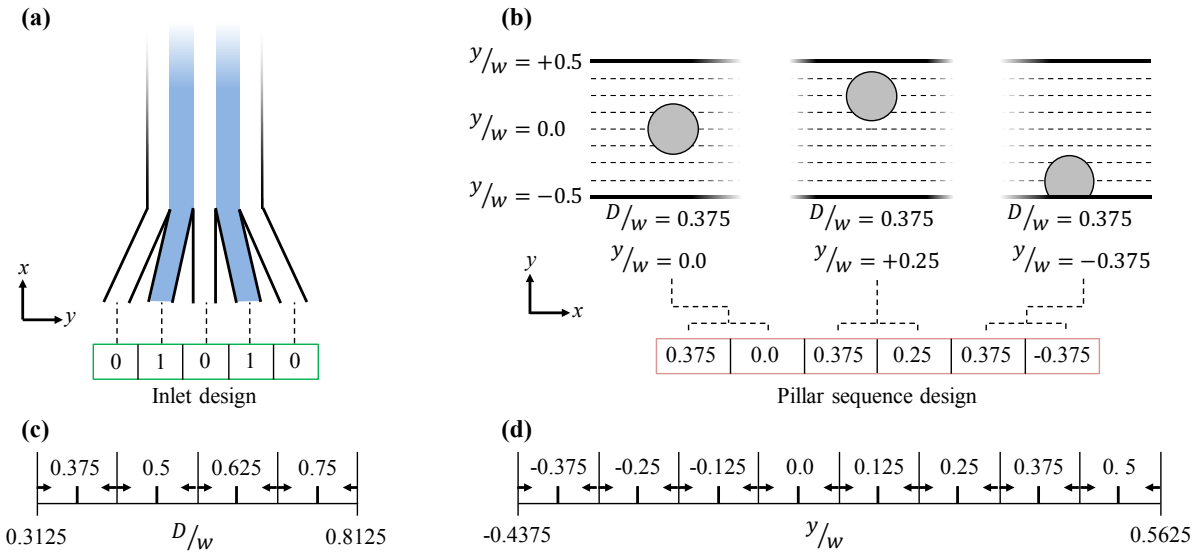

Figure 4.3 (a) Schematic for the inlet design portion of the chromosome, where the number of values in the chromosome represent the number of channels being joined at the inlet. The values are binary, where 0 represents non-tracked fluid and 1 is the fluid of interest being sculpted. (b) The $\mathbb{R}$-chromosome design uses two values for each pillar: diameter and location. (c-d) These values are given equal probability by their bounds in the GA, and interpreted in real time to select the nearest available pre-computed transition matrix. The entire chromosome for the design as shown would be [0, 1, 0, 1, 0, 0.375, 0.0, 0.375, 0.25, 0.375, -0.375].

channels at the microchannel inlet. These bits are limited to binary values (0 or 1) in the GA, and constrained so that their sum is never less than one or equal to the total number of channels (thus ruling out "empty" and "full" solutions).

When the fitness function evaluates a chromosome, an inlet vector is created with ones or zeroes in the fractions of the channel corresponding to each channel bit. For example, a 5-channel inlet can be represented in the chromosome as [0, 1, 0, 1, 0], with two-fifths of the inlet having tracked fluid separated by an "empty" central channel (see Fig. 4.3(a)). As the GA evolves its population, it will tune the inlet-specific portions of the chromosome for optimal fitness. Thus, the inlet design becomes an output of the GA framework.

4.3.3 Improving the pillar sequence chromosome design

We also change how the pillar sequence is encoded as a chromosome. Previously, integer values were used as proxies for a fixed set of micropillar configurations. That is, an arbitrary mapping of the possible configurations to the integer set was performed, and these integers were used as indices in the chromosome. For example, a chromosome of [1, 7, 31, 25] represented a 4-pillar sequence, with each integer referring to a micropillar of some diameter and position in the channel. This standard choice of representation results in an encoding that does not respect *locality*. That is, the baseline GA chromosome introduces arbitrary jumps in the design space, as incrementing an integer value may alter a pillar's design by a large change in diameter, position, or both. This could slow down convergence in the GA, as changes caused by mutation or crossover would have non-linear sensitivity. For example, if a particular chromosome is very close to a good solution, the only necessary change may be modifying a single pillar's diameter. In using integer proxies, there is a chance that mutation or crossover will alter both the pillar's diameter and position. In contrast, splitting up each pillar's diameter and position into separate entries improves the odds that a selection event will alter only the required parameter. Thus, using an encoding based on individual representation of pillar diameter and position necessarily doubles the chromosome length, but ensures locality of perturbations. This new chromosome design will be referred to as the $\mathbb{R}$-chromosome, and the previous, integer-proxy design, the $\mathbb{Z}$-chromosome.

To implement the $\mathbb{R}$-chromosome, we use two floating point values to describe a single pillar in the chromosome: one for its diameter, and one for its lateral position in the channel (see Fig. 4.3(b)). When the GA evaluates a population, the chromosome parameters are interpreted in real-time to translate from real-valued pillar configurations to their nearest possible design (per the supplied pre-computed transition matrices). In order to give the pillar configurations equal likelihood of selection during random processes, the bounds and interpretation code are created based on the intervals in the sampling (see Fig. 4.3(c,d)). This has the added advantage of being able to choose from arbitrary micropillar configurations and refine the set of transition matrices available to the GA without altering the chromosome. For example, some number of generations in a GA search could have a small, coarsely discretized selection of micropillar diameters and locations. Then, the GA can have an on-the-fly switch to a larger selection of pillar configurations with new chromosome bounds, possibly biased toward currently-favored pillar sizes and locations.

4.3.4 Optimized, open source GA framework

Our long term intention is to provide a lightweight, platform-agnostic, and freely available design framework that will enable wider utility by the microfluidics community. To accomplish this, we constructed a dependency-free, open-source GA application for pillar programming from the ground up in C++. Currently, the application runs on Windows, Linux, and OS X platforms, with an intuitive Graphical User Interface (GUI) designed using Python's Tkinter (see Fig. 4.4(a) for an image of the GUI). Although it is not necessary for use, the GUI allows users to draw their own fluid flow shape targets, load images, set GA parameters, and receive an output microfluidic device design without learning the command line interface. The application is available at www.me.iastate.edu/bglab/software.

The codebase contains a fundamental GA class that provides basic functionality. Top level applications that use the GA can be modularly added with minimal need for altering the framework. Moreover, almost every part of the algorithm can be customized. Users can modify the core GA to include more advanced features, such as thread-parallel evaluation of population members, or customized selection methods.

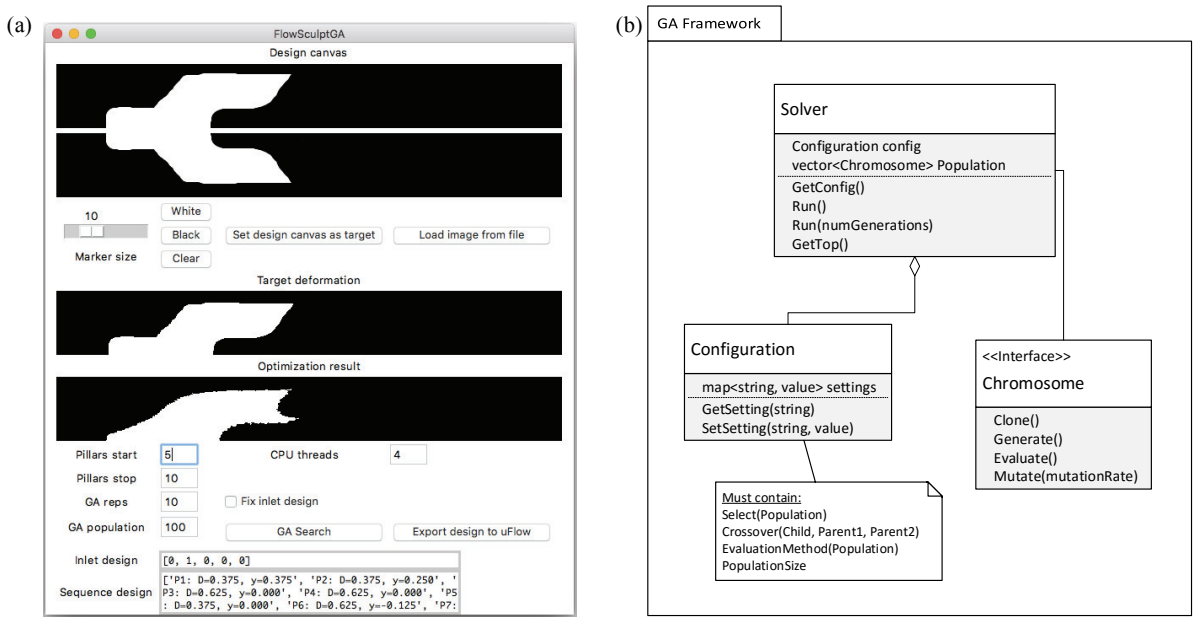

Figure 4.4 (a) Graphical User Interface (GUI) for using the GA framework. Note the drawing canvas (top of GUI) with enforced channel symmetry, GA parameters (lower left), and inlet/sequence design for a microfluidic device (bottom). (b) UML Class diagram of custom GA Framework.

At the core of our GA framework is a single class called Solver. Solver contains all functions and information needed to run the GA (see Fig. 4.4(b) for the class hierarchy). The Solver contains a configuration map which contains the settings for the GA. This includes which methods to use for selection/crossover, population size, fitness function evaluation, and other custom settings. If no methods are specified, the framework will fill in default methods. There are three basic methods used by the GA: selection, crossover, and mutation. The default selection method is tournament, which chooses parents for crossover by selecting the most-fit out of k random population members. The crossover method is required to be implemented by the user. This is done purposefully to provide greater flexibility in how the chromosome is represented. Finally, the default mutation rate of 5% randomly alters the population's chromosomes. All of these functions to rely to some degree on random information to be fed to the system. For this, an SIMD (Single Instruction, Multiple Data) oriented Fast Mersenne Twister (SFMT) library is included in the codebase to generate random data (Saito and Matsumoto, 2008). This generator has a large period for random data, reducing the likelihood of repeat information.

The Chromosome interface is intended to be subclassed by client software implementing the required methods. That is, users should be able to encode their chromosome design into a custom subclass which will be evaluated by the Solver. A subclass of Chromosome and a provided allocation and crossover methods are the minimum required components needed to use the framework. For this application, the subclass of Chromosome is our implementation of a pillar sequence with an inlet.

4.3.5 Using CUDA for the fitness function evaluation

Massive growth in Graphics Processing Unit (GPU) parallel computing capability in the last decade lends itself well to the execution of the GA. While the Central Processing Unit (CPU) can perform arithmetic at high speed (given appropriate problem size), it is still a serial process. Distributing the fitness function across multiple CPU cores will scale with the number of cores available to the user, which is typically below double digits for a normal user (without access to HPC resources). GPUs, on the other hand, have evolved to include many hundreds of arithmetic units, though they are each considerably slower than a modern CPU core. Smartly parallelizing a fitness function across a GPU's many processors can increase optimization speed, which will have a greater impact as the GPU continues to add more processors with each hardware iteration. Furthermore, replacing an existing computer's GPU device is a far simpler and cost-effective upgrade than replacing the CPU, which often requires a complete hardware overhaul. By enabling computation of the fitness function on the GPU, we allow users multiple routes to fast design.

We leverage CUDA, NVIDIA's language for General Purpose computing on Graphics Processing Units (GPGPU), to distribute our fitness function across many processors on consumer grade hardware (NVIDIA GTX 980). More specifically, we use cuSPARSE, a library within CUDA specifically designed and optimized to handle sparse matrix operations on the GPU. The process of random initialization, selection, and evolution (crossover/mutation) is handled by a host (CPU) process written in C++, while the fitness function's sparse-matrix operations are handled by the GPU. The GA evaluates the fitness functions serially using cuSPARSE. In the future, we anticipate that the evaluations themselves be parallelized by the use of CUDA

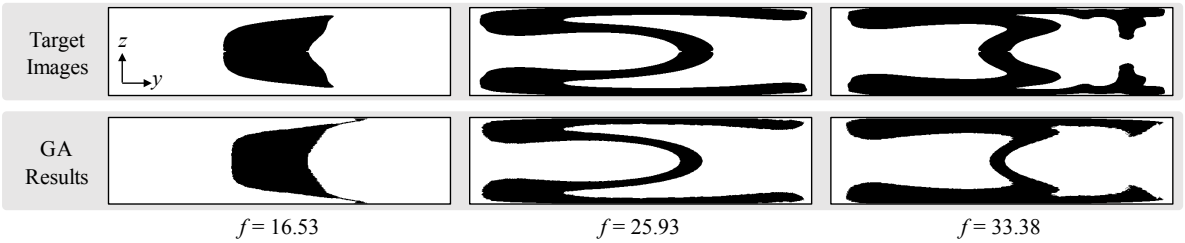

Figure 4.5 Examples of randomly generated target images and results from the GA framework (without using FFT post-processing in the fitness function), with their fitness values. Sculpted fluid is shown in black. The target images were all made with the same inlet design of [0, 0, 1, 0, 0] (see Fig. 4.3(a)).

streams, which will allow multiple cuSPARSE calls to be active on the GPU at a single time. This will help realize the full potential of the GPU, as fitness functions can be evaluated in parallel. Moving these operations to the GPU should also see improvement to fitness function runtime with subsequent generations of NVIDIA hardware.

4.4 Results and discussion

Results are shown using 6 different test cases which are designed to demonstrate the speed and scope of the developed framework, along with comparisons with the baseline framework.

Case 1: Transition matrix resolution. Compares the use of different resolution transition matrices to the baseline resolution ($N_Y = 801$, $N_Z = 101$) in terms of speed and accuracy.

Case 2: Multi-resolution GA. Tests the multi-resolution fitness function, which switches matrix resolution after 50 generations in the GA, for GA performance and accuracy.

Case 3: $\mathbb{R}$-chromosome vs. $\mathbb{Z}$-chromosome. Tests the $\mathbb{R}$-chromosome against the $\mathbb{Z}$-chromosome for GA performance and accuracy.

Case 4: Inlet design. Uses the $\mathbb{R}$-chromosome with multi-inlet target images to test the GA's ability to tune the inlet to match target flow shapes of arbitrary inlet design.

Case 5: Custom GA. Compares the low dependency, freely available GA created for this work against Matlab's GA.

Case 6: Demonstration of the modified GA. Uses all modifications of the GA for complex fluid flow shapes.

For these studies, target images have been generated using pre-computed transition matrices making an exact match in pillar sequence possible. The images have each been blurred and thresholded into binary images, giving them slightly larger area, more contiguous shapes, and smoother edges. This was done to simulate the nature of user-supplied designs[2]. The fitness value, f, will tend to vary within the range of 15-45 for a successful result. Fig. 4.5 shows examples of the randomly generated target images with corresponding solutions from the GA, along with their fitness value. A fitness value of $f = 16.53$, as seen in the first set of images in Fig. 4.5, shows excellent agreement with the target image. But even twice this value of $f = 33.38$ shows a decent result, with most of the discrepancies lying with high level features that would be unlikely to make a difference in fabricated devices.

Results using Matlab were run on an Intel i7-4790 CPU @ 3.60 GHz, while the custom framework used an NVIDIA GTX 980. Transition matrices were created using experimentally validated methods in (Amini et al., 2013; Stoecklein et al., 2014, 2016).

4.4.1 Case 1: Transition matrix resolution

The purpose of this study is to determine the smallest transition matrix resolution with accuracy close to the base line of $N_Y = 801$, $N_Z = 101$. A set of target flow shape images were created using quasi-random Sobol sequences for the pillar designs. 10 fluid flow shapes were selected from 5-10 pillar sequence lengths (10 from 5-pillar sequences, an additional 10 from 6-pillar sequences, etc.), for a total of 60 images of increasing complexity. A single inlet flow configuration of [0, 0, 1, 0, 0] (corresponding to the chromosome design) was used in order to allow for comparisons to the previous GA framework, which had no capability to tune inlet design. The target images were searched for using transition matrices of resolution $N_Y = [101 : 100 : 801]$. The microchannel aspect ratio was fixed at $h/w = \frac{1}{4}$, and since

[2]Because of post-processing of the target images, it is very unlikely that the GA will find the "true" solution (i.e., the pillar sequence used to generate a target image). However, considering that fluid flowing through a fabricated micropillar device will encounter some blur or aberrations depending on mass diffusion and manufacturing tolerances, the precision necessary for a successful result is not strict, and ultimately up to the user.

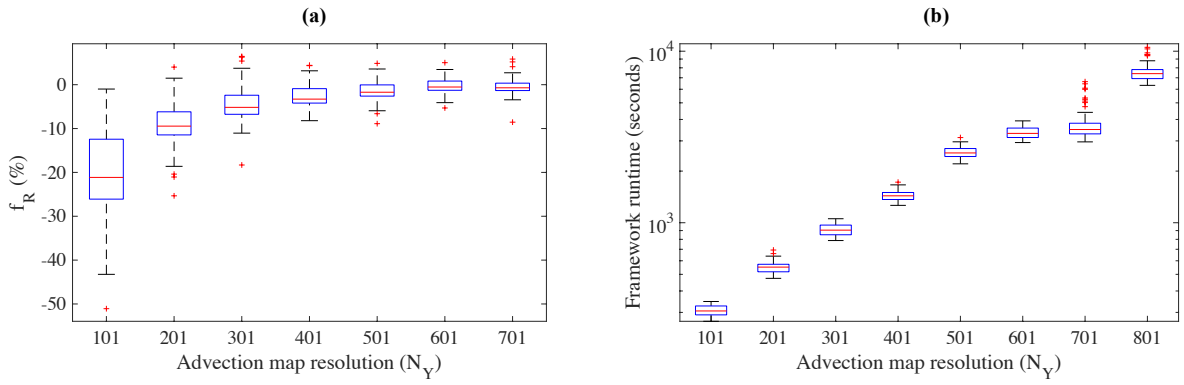

Figure 4.6 Results for case 1, which tested different transition matrix resolutions for accuracy and speed. 60 different flow shape images were generated using transition matrices from sequences determined by the quasi-random Sobol set, and a fixed inlet configuration. Each target image was blurred and thresholded to mimic a realistic user-supplied design. Framework accuracy (a) and runtime (b) are shown for each transition matrix size, for all 60 target images. Note that the fitness function accuracy (a) has been normalized relative to the baseline ($N_Y = 801$).

the simulation data is restricted to the top-half of the microchannel (due to symmetry in the microchannel and pillar geometry), $N_Z = \lceil \frac{N_Y}{8} \rceil$. The baseline and updated frameworks in case 1 both used Matlab's built-in GA. In all searches, a population size of 100 and generation limit of 200 were enforced, the GA swept 5-10 pillar sequences, and repeated each sequence size 10 times (for a total of 60 GA searches per image; per Algorithm 1, $N_{P,start} = 5$, $N_{P,stop} = 10$, and $N_{GA} = 10$). To compare GA accuracy to the baseline streamtrace resolution of $N_Y = 801$, we define a relative fitness f_R:

$$f_R = 100 \times \frac{f_{GA} - f_{Baseline}}{f_{Baseline}} (\%) \tag{4.5}$$

Results for case 1 are shown in Fig. 4.6. Fig. 4.6(a) shows how transition matrix resolution affects GA framework accuracy as measured relative to the baseline, as measured by f_R (equation 5). Accuracy comparable to the baseline is retained until $N_Y = 601$, though optimal performance is still competitive at $N_Y = 401$. We hypothesize that despite a considerable loss in accuracy (as much as 50% for $N_Y = 101$, see Fig. 4.6(a)), smaller matrices can be useful in a multi-resolution framework (see earlier discussion). We validate this hypothesis in case 2.

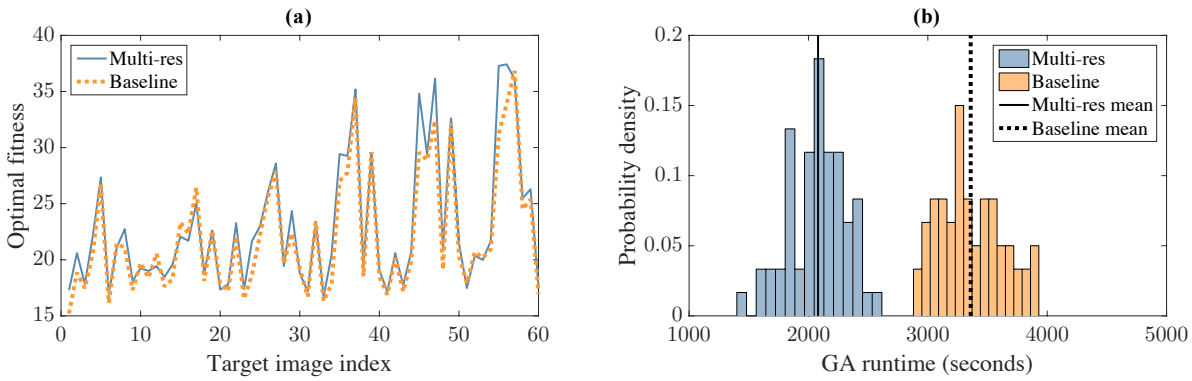

Figure 4.7 Results for case 2. (a) Use of a mutli-resolution fitness function ($N_Y(M_L) = 51$, $N_Y(M_H) = 601$) resulted in similar optimal fitness to the baseline of $N_Y = 601$. (b) Use of $N_Y = 51$ transition matrices for the first 50 generations resulted in substantially decreased runtime.

A GA framework runtime comparison is shown in Fig. 4.6(b). Resolutions of $N_Y = 701$ and $N_Y = 601$ have similar accuracy to the baseline of $N_Y = 801$, but $N_Y = 601$ has a runtime improvement of 50% over the baseline. Going forward, the GA framework for pillar programming can cut computational effort in half by using $N_Y = 601$, without an effective loss in accuracy to the baseline. Accuracy within 10% of the baseline is still obtainable in utilizing a streamtrace resolution of $N_Y = 401$, for which runtime reduces by a factor of 4.

4.4.2 Case 2: Multi-resolution GA

This study tested the multi-resolution fitness function as outlined in Algorithm 2, using $N_Y(M_H) = 601$ and $N_Y(M_L) = 51$ (not shown in case 1). The parameter $Gen_{switch} = 50$, which matches the stall generation limit (a termination condition). Results are shown in Fig. 4.7. As mentioned previously, the choice of Gen_{switch}, $N_Y(M_L)$, and $N_Y(M_H)$ lends itself to another study entirely, with a parameter study and adaptive switching being likely directions. The results for this particular study show that the multi-resolution GA is a powerful method for accelerating convergence without sacrificing accuracy.

Coupled with the results from case 1, we can now give users several options weighing runtime against precision with respect to fine-scale flow shape features. On one hand, using transition

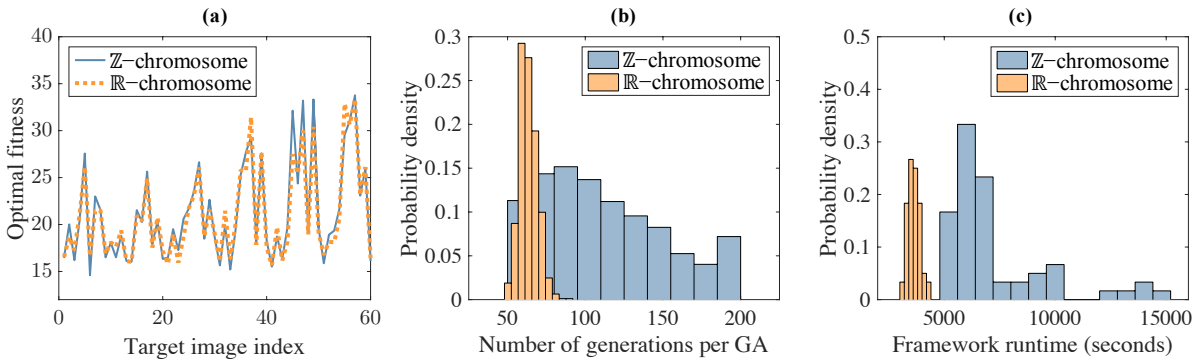

Figure 4.8 Results for case 3, which tests the ℝ-chromosome against the integer proxy ℤ-chromosome. The inlet design is fixed in the chromosome, thus isolating the test to the choice of pillar sequence encoding. (a) The ℝ-chromosome results in comparable accuracy in optimal fitness for each target image. Here, the fitness for the best result from each GA framework search is reported for all 60 images. (b) Use of the ℝ-chromosome shows faster convergence in the GA with a mean of 63 generations per GA. Note that the ℤ-chromosome, with a mean of 111 generations, had a significant proportion of GA searches terminate at the generation limit of 200. (c) Fewer generations translates into less framework runtime, with a median of ≈7,000 seconds for the ℤ-chromosome, and ≈3,600 seconds for the ℝ-chromosome.

matrices only of $N_Y = 401$ results in a fast but possibly less accurate result. On the other hand, using $N_Y = 601$ will give more accurate results to the target flow shape, but at twice the execution time. Using multi-resolution matrices offers a middle ground between the two, and could even be used to accelerate the $N_Y = 401$ resolution further. We anticipate offering these choices to users in future releases of the custom C++ framework.

4.4.3 Case 3: ℝ-chromosome vs. ℤ-chromosome

The same set of 60 target flow shape images from case 1 were used to compare the ℤ-chromosome to the ℝ-chromosome. From Section 4.3.3, the ℤ-chromosome used a set of integer values as indices in a look-up table for each pillar configuration, which leads to highly non-linear design space due to the jumps that will occur with changing pillar diameters or locations. The ℝ-chromosome, on the other hand, splits each pillar in the chromosome design into two real-values: one each for diameter and location. We hypothesized that this will translate

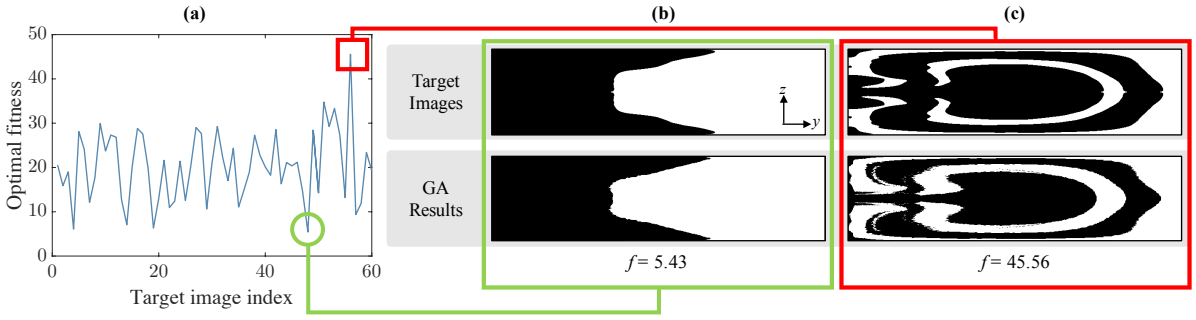

Figure 4.9 Results for case 4, which tested the GA's ability to design the inlet flow configuration. (a) Optimal fitness results for 60 target flow shapes with randomly generated pillar sequences and inlet flow configurations. Sculpted fluid is shown in black. The best match by fitness, with $f = 5.43$, is shown in (b), while the worst match, with $f = 45.56$, is shown in (c). Although the discrepancies between the target fluid shape and the GA result in (c) are clearly discernible, the overall design has still been found.

into faster GA convergence due to more smoothly varying neighborhoods in the input space. For this study, the baseline transition matrix resolution of $N_Y = 801, N_Z = 101$ was used for both chromosome types. The inlet was fixed for both chromosomes, meaning the GA need only design the pillar sequence. Results are shown in Fig. 4.8. Despite the increased chromosome size and complexity (which can lead to a larger number of generations needed for convergence), there is no meaningful change in search capability per target image (Fig. 4.8(a)). Use of the $\mathbb{R}$-chromosome shows vast improvement in GA convergence (Fig. 4.8(b)). Where the previously used $\mathbb{Z}$-chromosome repeatedly ran into the hard limit of 200 generations, the $\mathbb{R}$-chromosome converged by 88 generations on average, with none of the GAs requiring more than 100 generations. This reduces runtime by nearly 50% (Fig. 4.8(c)). In fact, the distribution of generations required for the convergence shifts from skewed shape to a more normal distribution. Overall, these results fall in line with our hypothesis of a faster GA by using a more physically modelled chromosome.

4.4.4 Case 4: Arbitrary inlet design

To test for arbitrary inlet design, a new set of 60 target flow shape images were generated with the same scheme as case 1, except the Sobol sequences now include values for a 5-channel

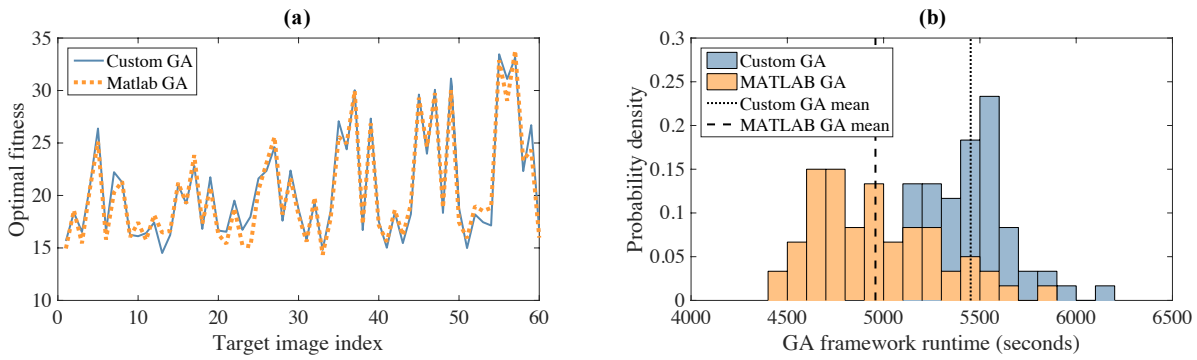

Figure 4.10 Result for case 5, which compares performance of the custom GA to the Matlab implementation. (a) Comparison of optimal fitness values for each of the 60 target images, which are identical to those used in case 1. Unlike case 1, the portion of the chromosome which governs inlet design was accessible to the GA, thus making the problem more difficult. Note that the custom GA accuracy matches, and in some cases exceeds that of the Matlab GA. (b) Runtime distributions for the custom GA and Matlab GA. The high complexity of the problem compared to case 1 (designing the inlet as well as the pillar sequence) requires additional generations for convergence, which increases runtime.

inlet design. Therefore, these target flow shapes have varying amounts of fluid and locations of the target flow shape. This is meant to test the GA framework's ability to design inlet configurations. Here, the baseline framework has no means of effective comparison, as each image would require multiple user-guided GA searches. Results are in Fig. 4.9, showing fitness values for each target image in Fig. 4.9(a), and examples of the best and worst fitness in Fig. 4.9(b,c). Allowing the GA to design the inlet for the microfluidic device in this manner is highly successful, with even the worst reported fitness resulting in an overall viable design ($f = 45.56$, Fig. 4.9(c)). However, the increased complexity of the design leads to longer framework runtime, as seen in case 5 (Fig. 4.10). The median runtime of approximately 3,600 seconds for a fixed inlet (as in case 2, Fig. 4.8) increases by nearly 40% to roughly 5,000 seconds. Nonetheless, given the inlet design iterations that this new chromosome design avoids, using the GA to design the fluid flow inlet is by far the more desirable option.

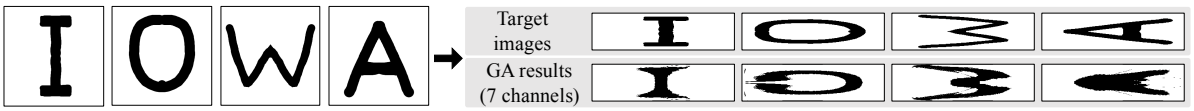

Figure 4.11 Results for case 6, which uses the new GA framework for hand-drawn flow shapes of the letters that spell "IOWA", for microchannel aspect ratio $h/w = 0.25$.

4.4.5 Case 5: Custom GA

To compare our custom GA implementation against Matlab's GA, we use the test images from case 1. Each framework used the $\mathbb{R}$-chromosome, with the inlet being designed by the GA. The custom GA performed fitness function calculation using CUDA on an NVIDIA GTX 980, while Matlab used an Intel i7-4970 @ 3.60 GHz. We compare runtime and fitness values in Fig. 4.10. Fitness values for the custom framework match, and in some cases even improve upon, the values found using Matlab's GA. We do not see an improvement in runtime by moving the fitness function to the GPU, which is not entirely unexpected. The matrices being used are very sparse (as many non-zero values as rows in the matrix), and their size does not make for particularly difficult computation on the CPU. Despite this, GPU runtime is competitive with the (currently) modern CPU used here. Moreover, future development to the custom framework could enable multiple CUDA streams to distribute the fitness functions across many GPUs. This would allow for generation of fitness functions to be evaluated simultaneously, which would accelerate the framework and make larger population sizes more practical.

4.4.6 Case 6: Demonstration of the modified GA

In case 6, letters from the English alphabet were used as targets. Unlike all other test images used in this work, the letters were hand-drawn, making their existence in the design space completely unknown. Due to the top-bottom symmetry in sculpted flow shapes, we are only able to design for symmetric letters. Since the flow physics are entirely contained within the transition matrices, future optimization problems could incorporate new geometry that allows for top-bottom asymmetry, such as half-pillars, or "steps" in the microchannel. Here, we drew the capitalized four letters in "IOWA", choosing the axis of symmetry in the

microchannel to align with similar axes in the letters. The images were resized for 3 different microchannel aspect ratios: $h/w = [0.25, 0.5, 1.0]$, where h is the height of the microchannel. In searching across microchannel aspect ratios, we expect the designed shapes to have considerably different styles of deformation. Amini et al. (2013) established that changing the microchannel aspect ratio will introduce new modes of fluid deformation, with some modes being unique to particular aspect ratios. The GA searches used 9 channels for $h/w = 1.0$ and 7 channels for $h/w = [0.25, 0.5]$. Results for $h/w = 0.25$ are in Fig. 4.11, and results for $h/w = 0.5, 1.0$ are in Appendix B. Matching fluid flow shapes are found for each letter, despite the unique flow deformations available across different channel aspect ratios. This is a highly useful result, as manual design would require memorization of the various modes for each aspect ratio in addition to combinatorial difficulty. All results show that while the framework is effective in designing for bulk fluid displacement and overall shape, subtle features are still difficult to optimize for. As an example, the hollow portion of the letter "A" is never truly found, though the outline of the letter does not seem difficult. This indicates room for improvement on the fitness function itself, which remains a simple correlation function. Promising avenues include using elliptic Fourier representations of shape, wavelet representation or using methods from image classification like SIFT (Scale Invariant Feature Transforms).

4.5 Conclusion

The concepts of inertial microfluidics and flow sculpting are increasingly being used by the microfluidics community for a diverse set of technologically and socially relevant applications. However, manual exploration and design of pillar sequences have proven infeasible for generating complex flow transformations necessary for these applications. We formulate the flow sculpting problem as an optimization problem to enable automated design of pillar sequence design that results in a desired fluid flow transformation. In this work, we developed an automated optimization framework for efficient lab-on-a-chip design. This provides a valuable resource to the microfluidics community to leverage new understanding of inertial fluid flow.

We detail four computational modifications that improved the computational efficiency and functionality of the optimization framework. This includes a parameter encoding (i.e.,

chromosome) that respects parameter locality ensuring fast convergence, a multi-resolution approach that accelerates convergence while maintaining accuracy, extending the chromosome to include inlet design, and enabling the use of GPU architecture via NVIDIA's CUDA for function evaluations. We package this framework into a user friendly, freely available software suite that enables the larger microfluids community to utilize these developments. We showcase the computational improvements using a series of case studies and conclude with non-trivial designs.

Future avenues of research include designing more sophisticated fitness functionals. For example, users may not have a preference for location, size, or orientation of the sculpted fluid flow shape. This suggests a fitness function that is invariant to translation, scale, or rotation. In addition, the use of simple transition matrices makes the addition of user-created 2-D advection maps trivial. As long as new transition matrices can be effectively represented in the chromosome, entirely new microchannel geometry (e.g., half pillars, steps, curved regions) can be easily added to the GA toolkit for microfluidic device design. In summary, the framework we have developed not only supplants previous design optimization efforts, but provides a roadmap for improvements to speed and functionality for future flow sculpting applications.

CHAPTER 5. DEEP LEARNING FOR FLOW SCULPTING: INSIGHTS INTO EFFICIENT LEARNING USING SCIENTIFIC SIMULATION DATA

A paper accepted by *Scientific Reports*

Daniel Stoecklein, Kin Gwn Lore, Michael Davies,

Soumik Sarkar, and Baskar Ganapathysubramanian

Abstract

A new technique for shaping microfluid flow, known as flow sculpting, offers an unprecedented level of passive fluid flow control, with potential breakthrough applications in advancing manufacturing, biology, and chemistry research at the microscale. However, efficiently solving the inverse problem of designing a flow sculpting device for a desired fluid flow shape remains a challenge. Current approaches struggle with the many-to-one design space, requiring substantial user interaction and the necessity of building intuition, all of which are time and resource intensive. Deep learning has emerged as an efficient function approximation technique for high-dimensional spaces, and presents a fast solution to the inverse problem, yet the science of its implementation in similarly defined problems remains largely unexplored. We propose that deep learning methods can completely outpace current approaches for scientific inverse problems while delivering comparable designs. To this end, we show how intelligent sampling of the design space inputs can make deep learning methods more competitive in accuracy, while illustrating their generalization capability to out-of-sample predictions.

5.1 Introduction

As the availability and power of modern computing resources has increased, so has interest in the field of inverse problems in science and engineering (Neto and Neto, 2013; Vogel, 2002). In particular, ill-posed inverse problems for which there is no analytical solution or certainty of a unique solution, but have a tractable forward model, are now more easily solved with modest computing hardware. An example of such a physical system is a recently developed method of fluid flow manipulation called flow sculpting. Flow sculpting uses sequences of bluff-body structures (pillars) in a microchannel to passively sculpt inertially flowing fluid (where $1 < Re < 100$, for Reynolds number $Re = \frac{UD_H}{\nu}$, with fluid velocity U, viscosity ν, and channel hydraulic diameter D_H). Specifically, fluid flowing in the inertial regime past a pillar shows broken fore-aft symmetry in the pillar-induced deformation, which laterally displaces the cross-sectional shape of the fluid in some way depending on the channel and pillar geometry, and the fluid flow conditions (described by Re)(Amini et al., 2013). By arranging pillars in a sequence within a microchannel, the fluid will experience individual deformations from each pillar, resulting in an overall net deformation at the end of the sequence (see Fig. 5.1 for an illustration of flow sculpting). With sufficient spacing between each pillar in a sequence, the deformation from one pillar will saturate before the flow reaches the following pillar. Therefore, the deformation caused by a single pillar can be viewed as an independent operation on the fluid flow shape, enabling a library of pre-computed deformations to predict the sculpted flow shape for a given pillar sequence. Work by Stoecklein et al. demonstrated this forward model with user-guided manual design in a freely available utility "uFlow"[1] to create a wide variety of useful flow shapes(Stoecklein et al., 2014).

Flow sculpting via pillar sequences has since been applied to problems in biological and advanced manufacturing fields. For example, polymer precursors can create shaped microfibers and particles (Nunes et al., 2014; Paulsen et al., 2015; Paulsen and Chung, 2016; Wu et al., 2015), and a pillar sequence can shift fluid streams away from cells in flow (Sollier et al., 2015). Although novel in their application, these use cases utilize simple micropillar sequence designs

[1] www.biomicrofluidics.com/software.php

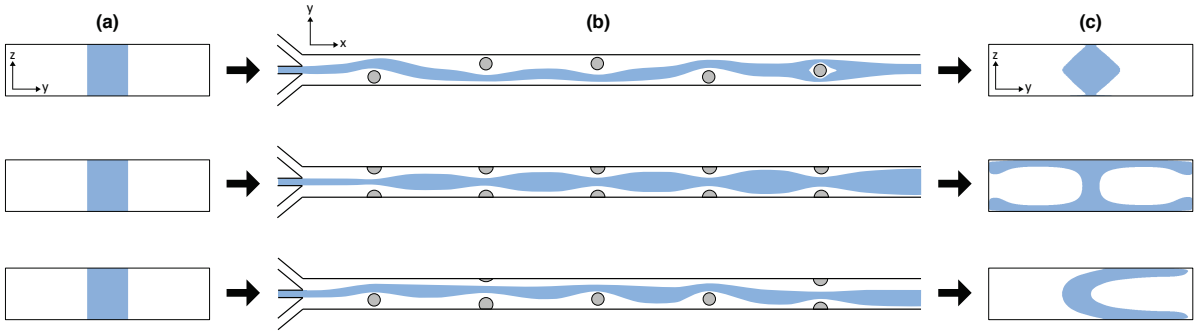

Figure 5.1 Illustration of three different flow sculpting devices that modify the same inlet fluid flow. (a) For a given inlet flow configuration (shown here in a cross-sectional view with the middle-fifth of the channel containing the sculpted fluid, colored blue), arbitrary sequences of pillars (b) (shown as top-down views of three different microchannels) can purposely sculpt the cross-sectional shape of fluid, yielding a net deformation (c) at the outlet of the channel.

(e.g., forming an encapsulating stream, or shifting fluid to one side of a microchannel). More complex fluid flow shapes could lead to powerful new technologies in the aforementioned fields, for example: fabricated microparticles could be designed for optimal packing efficiency, or to focus with directed orientation to specific locations within a microchannel for improved on-chip cytometry(Uspal et al., 2013); porous hydrogel could be designed to reduce wound healing time(Griffin et al., 2015), or study cell growth and chemotaxis (Fiorini et al., 2016). These are perhaps more obvious applications of flow sculpting, but as more disciplines and industries are exposed to the technique, new possibilities are expected to abound.

However, microfluidic device design via trial-and-error is tedious, and requires user intuition with the design space. Choosing from many sizes and locations for individual pillars, in addition to their sequential arrangement in a microchannel, produces an enormous combinatorial space of fluid flow transformations. It has also been shown that the space is multi-modal, with many micropillar sequences creating similar fluid flow shapes, making design optimization non-trivial. Clearly, these features (a large *many-to-one* design space with non-linear fluid transformations) are endemic to many engineering inverse problems, with flow sculpting being a good representative. Thus, manual design of micropillar sequences is generally impractical for most of its intended users, which includes researchers in fields such as advanced manufac-

turing, biology, bio-sensing, healthcare, pharmaceuticals, and chemistry (Amini et al., 2014; Zhang et al., 2016; Lu et al., 2016). This drives the need for an automated solution to the inverse problem: designing a micropillar sequence that produces a desired fluid flow shape. To date, there are two automated approaches in literature: heuristic optimization via the Genetic Algorithm (GA)(Stoecklein et al., 2016, 2017a) and deep learning via trained Convolutional Neural Networks (CNN)(Lore et al., 2015). While the GA capably optimized existing microfluidic devices and explored novel flow shapes, there exist a few drawbacks to its use. GAs require well-crafted cost functions specific to different problems, necessitating that the user have knowledge of programming and optimization. The GA is also a stochastic method, with no guarantee of finding global optima using a finite number of searches. For flow sculpting, this leads to excessive runtime (as much as 2 h), which makes swift design iterations difficult (Stoecklein et al., 2017a). On the other hand, the application of deep learning shown by Lore et al. operated with an extremely quick time-to-result ($\approx$ 1 s), but lacked in accuracy comparable to the GA(Lore et al., 2015). We seek to carefully improve and demonstrate the capabilities of deep learning in a comprehensive manner for flow sculpting with this work. Our study also reveals fundamental insights into training strategies for machine learning of engineered systems, which aids in providing best practices for analogous engineering inverse problems.

A key motivation in the use of deep learning for design in flow sculpting is the ability to generalize patterns and associations between input and output data into a globally effective mapping (Bengio et al., 2007). That is, a properly trained deep neural network (i.e., with appropriate model structure and regularization, sufficient training data, and carefully chosen hyperparameters) does not strictly memorize the relationship between local regions in the input space to their corresponding regions in the output space, but creates a function that maps input data to the output space based on the learned hierarchical features and rules of the underlying physical system (i.e., rote memorization vs. actual learning and generalization). This allows a trained neural network to operate in as-yet unseen regions of the input space. This is especially of interest for flow sculpting, as a completely exhaustive search for tailored fluid flow shapes is computationally infeasible. This is owed to the combinatorial complexity in the design of a pillar sequence. Previous work has performed effective design in choosing

among 32 possible pillar configurations (four pillar diameters and eight lateral locations in the channel), using 1-10 total pillars in a sequence (Stoecklein et al., 2014). Thus, a fluid flow shape created from a 10-pillar sequence exists among $32^{10} \approx 10^{15}$ possible combinations, making this a difficult combinatorial problem. On the other hand, a trained neural network could map a large portion of the design space from a significantly smaller sample. The rapid feedback in using a deep learning tool - generally a matter of seconds - also makes for a user-friendly experience. Moreover, the design space will continue to grow as additional microfluidic components are added to the flow sculpting toolset, for example: half-pillars, steps, and curved channels can also be used to deterministically sculpt fluid flow. Thus, it is important to have a design utility that maintains parity with new flow sculpting tools without requiring extensive re-training and fine tuning, which aligns with the strengths of deep learning.

We speculate that an impediment to performance in deep learning techniques for design in flow sculpting[2] is the *many-to-one* design space. That is, there may be many solutions (pillar sequences) that produce a desired fluid flow shape. Consider the set all possible pillar sequences $\boldsymbol{s_i}$ as the space $\mathcal{S}$, and their corresponding fluid flow shapes $\boldsymbol{o_i}$ as the space $\mathcal{O}$, with a forward model f that maps a specific realization $\boldsymbol{s} \in \mathcal{S}$ to $\boldsymbol{o} \in \mathcal{O}$, i.e. $f : \mathcal{S} \rightarrow \mathcal{O}$. A deep neural network attempts to construct an approximation to $g = f^{-1}$, with the inverse function g mapping $g : \mathcal{O} \rightarrow \mathcal{S}$. During training, a deep neural network that has been shown a pillar sequence and fluid flow shape pair ($\boldsymbol{s_1}$, $\boldsymbol{o_1}$) may be trained on another flow shape, $\boldsymbol{o_2}$, that is extremely similar to $\boldsymbol{o_1}$ and thus occupying the same space in $\mathcal{O}$. However, the pillar sequence $\boldsymbol{s_2}$ that produces $\boldsymbol{o_2}$ could be entirely different from $\boldsymbol{s_1}$, and therefore very far apart in $\mathcal{S}$. Hence, the *many-to-one* mapping $f : \mathcal{S} \rightarrow \mathcal{O}$ could make effective training quite difficult.

Appropriate selection of training data and an understanding of what constitutes "good" training remain open challenges in modern applications of machine learning (Zeiler and Fergus, 2014). However, unlike traditional problems in machine learning - image classification or speech and handwriting translation, for example, where training data attempt to sample an unbounded and highly variable space - the domain of flow sculpting is finite (though extremely large).

[2]While our focus is clearly on the flow sculpting problem, the issues raised here would tend to appear in most inverse problems.

Furthermore, the space of sculpted flows presents a natural metric (i.e., binary images with sculpted flow and co-flow) that enables efficient characterization of the data space. This offers a unique opportunity to explore how domain knowledge and the choice of sampling can influence high-level decision making in a deep learning model. Similar critical scientific optimization problems, such as robotic path planning, material processing, or design for manufacturing can benefit from the insight on intelligent sampling gained here. We explore a sampling method for choosing training data known as High Dimensional Model Representation (HDMR) (Rabitz et al., 1999), and analyze the space $\mathcal{O}$ using dimension reduction via Principal Component Analysis (PCA). Our analysis includes a parameter study on training set size, along with several out-of-sample studies to demonstrate deep learning's capability to generalize for this complex *many-to-one* problem. We also test the hypothesis that a training set with a more uniform distribution in $\mathcal{O}$ will lead to a more accurate model.

5.2 Results and Discussion

5.2.1 Flow Sculpting Physics

The concept and implementation of inertial fluid flow sculpting via pillar sequences has been previously investigated by the work of Amini et al. (Amini et al., 2013, 2014) and Stoecklein et al. (Stoecklein et al., 2014, 2016, 2017a). We will briefly elaborate on the generalities of flow sculpting to clarify the application for this work. Most microfluidic devices typically employ low-Re fluid flow, known as Stokes flow, such that $Re \approx 0$ (Squires and Quake, 2005) (in this case, the characteristic length that defines Re is the microchannel hydraulic diameter). This flow regime is generally achieved via small length scales and low flow rates, and is highly laminar, easily controlled, and well predicted. One consequence of Stokes flow is that fluid flowing past an obstacle exhibits no inertial effects. Therefore, in Stokes flow past an obstacle in a microchannel, fluid will return to its original location in the channel cross-section, with no apparent displacement. However, by increasing Re, inertia in the fluid begins to break this symmetry around the ostacle, inducing a deformation in the fluid as it flows past the pillar. This effect was first shown in a confined microchannel by Amini et al., who determined that, in using

a pillar as an obstacle, post-pillar time dependent effects do not appear until $Re > 100$ (Amini et al., 2013). Hence, flow conditions of $1 < Re < 100$ define the operating regime for which fluid flow past a bluff-body obstacle will exhibit laminar behavior, but is predictably deformed in a time-independent manner. The nature of the deformation will depend on pillar diameter, location in the channel, flow physics (Re), and channel geometry (aspect ratio, for a rectangular channel). The idea of flow sculpting is to leverage single-pillar flow deformations as independent operations on the cross-sectional fluid flow structure, and piece multiple pillars together to create a more complex net deformation at the end of the pillar sequence. An illustration of inertial flow sculpting is shown in Fig 5.1, where three different sequences of five pillars each deform the same inlet flow pattern into distinct flow shapes at their respective outlets.

A necessary condition for flow sculpting is that pillars must be placed far enough apart such that each individual pillar's flow deformation saturates before the fluid arrives at the subsequent pillar, preventing cross-talk between two pillars. The prevention of cross-talk enables a powerful computational shortcut for simulating flow sculpting devices, as complex flow fields for single-pillar configurations - which require significant computational effort to acquire - need only be solved once before their subsequent use in simulating a pillar sequence. That is, rather than solving the Navier-Stokes equations for fluid flow within an entire multi-pillar device, pre-computed solutions for the individual pillars that comprise the device can be assembled in real-time for an accurate prediction of sculpted flow. Provided that the microchannel dimensions and Reynolds number for each pre-computed flow deformation - stored as a 2-D "advection map" - match, an arbitrary arrangement of sufficiently spaced pillars can be well predicted by the concatenation of their respective maps. This informs the creation of a pre-computed library of advection maps for a variety of pillar configurations, which can be easily utilized with modest computing power for flow sculpting design. Additional details for the implementation of the forward model is in the Methods section.

5.2.2 Deep Learning Framework

Deep learning, a branch of machine learning, emphasizes the use of multiple levels of abstraction of data. It achieves this by using hyper-parametrized model structures composed of

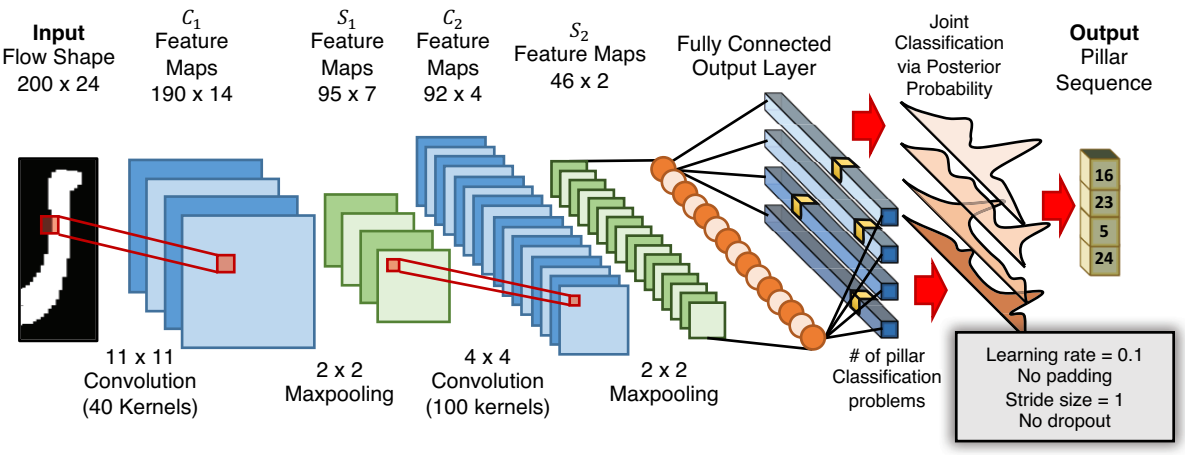

Figure 5.2 Schematic of the Convolutional Neural Network (CNN) used in this work. The input image on the left consists only of pixels from the top-half of a microchannel simulation, due to the top-bottom symmetry implied by the pillars spanning the full channel height. Note the posterior distributions in the joint classification layer which determine the predicted pillar sequence via maximum a posteriori probability estimate.

nonlinear transformations of data. (Deng and Dong, 2014; Bengio et al., 2013). Deep learning methods have been an attractive option to researchers for data dimensionality reduction (Hinton and Salakhutdinov,), collaborative filtering (Salakhutdinov et al., 2007), feature learning (Coates et al., 2011), topic modeling (Salakhutdinov and Hinton, 2009), and classification problems (Larochelle and Bengio, 2008). Recently, deep learning has also gained traction as a tool for data-driven decision making systems. Examples of its wide applicability across scientific disciplines include steady flow approximation (Guo et al., 2016) and early detection of combustion instability (Akintayo et al.,) for engineering; low-light image enhancement for autonomous mobile systems (Lore et al., 2017); policy learning and reward estimation (Lore et al., 2016) for decision and support; high-throughput plant phenotyping (Singh et al., 2016) and modeling specificity of DNA-protein binding (Zeng et al., 2016) for biology and plant sciences; and in high-energy physics, researchers have used deep learning to search for exotic particles (Baldi et al., 2014) and modes for Higgs Boson decay (Sadowski et al., 2014).

We make use of a Convolutional Neural Network (Krizhevsky et al., 2012) framework and pose the inverse problem in flow sculpting as a simultaneous multi-class classification prob-

lem. A schematic of this framework is shown in Fig. 5.2, with the images as inputs and their corresponding pillar sequences matching to a fixed number of classifier nodes at the output. Note that for the deep learning framework, which is intended to solve the inverse problem, by building an approximation of the map $g : \mathcal{O} \rightarrow \mathcal{S}$ it is implied that the fluid flow shapes in $\mathcal{O}$ become inputs for the CNN, while pillar sequences in $\mathcal{S}$ are the output. Convolutional neural networks use feature maps to preserve spatially local correlations within an image, therefore exploiting the 2D structure of input data to improve detection, classification, and prediction over typical neural networks. These feature maps are obtained by convolving filters, which comprise an early hidden layer in the neural network, over the entire image. The feature maps then undergo a process called maxpooling, which is a method of downsampling by overlapping sequential filters such that important information is retained from the feature maps while reducing the dimensionality of the system for subsequent layers. Our CNN repeats this process over smaller filters before feeding the reduced mapping into a fully connected output layer, which operates as a classifier for a pillar sequence corresponding to the flow shape. Every pillar in the output sequence is classified using integer values 1-32, choosing from 32 possible types of pillars. Each type of pillar is defined size and location: four diameters ($D/w = \{0.375, 0.5, 0.625, 0.75\}$ for pillar diameter D and channel width w), and eight different locations in the channel ($y/w = \{-0.5, -0.375 - 0.25, -0.125, 0.0, 0.125, 0.25, 0.375, 0.5\}$, where $y/w = 0.0$ is the center of the channel). This library is created by the same experimentally validated simulation technique used by Stoecklein et al. (Stoecklein et al., 2016, 2017a) and Lore et al. (Lore et al., 2015), which is explained in more detail within the methods section of this work. We use this CNN in several experiments outlined below.

5.2.3 Principal Component Analysis

We utilize Principal Component Analysis (PCA) to characterize the space $\mathcal{O}$. PCA is a method of linear dimensionality reduction that creates an orthogonal basis within a dataset's high-dimensional space, which optimally explains the variance of every observation in the dataset(John Aldo Lee, 2007). Once this basis is constructed, every observation from the original dataset can be shown to be a linear combination of each high-dimensional axis ("principal

component") of the new basis. Importantly, the principal components are ordered in terms of variance explained within the original dataset, i.e., the 1st principal component explains the most variance within a dataset, the 2nd principal component explains the second-most variance, and so on. PCA aids in high dimensional data visualization, as the degree to which an observation uses each principal component provides a mapping onto the newly constructed high-dimensional basis. This means that the original high-dimensional dataset can be projected onto an n-D orthogonal space ("PCA-space"), where the coordinates of a projected observation from the dataset (flow shape images, for this work) shows how that data aligns with an axes' principal component. Therefore, flow shape images that project to nearby regions in the PCA-space are similarly explained by the principal components of those axes, implying that they are similar flow shapes. Conversely, points far apart in the PCA-space imply that the compared fluid flow shapes are more dissimilar.

An example of PCA projection of a training set of flow sculpting image is shown in Fig. 5.3. The statement above concerning proximity in the PCA-space is demonstrated, where a target image is selected near the top of the 3-D grouping. All other fluid flow shapes are compared to this target image using the Pixel Match Rate (PMR), a measure of image similarity discussed below which compares the number of matching pixels of two images (a higher PMR means two images are more similar). For this projection, approximately 44% of the variance in the high dimensional data is explained by the first three principal components. While this does not explain all variation in the data, it is clear from Fig. 5.3 that flow shapes which project to similar locations in the PCA-space are also similar in their full dimensionality, and vice versa. We use the PCA-space as a visually interpretable representation of $\mathcal{O}$, and utilize this to reason about the performance of our deep learning tool, and perform $\mathcal{O}$-sampling, discussed below.

5.2.4 High-Dimensional Sampling

Intuitively, an effective tool for design in fluid flow sculpting should have good coverage of the space $\mathcal{O}$. However, knowledge of this space is difficult to come by, as the combinatorial possibilities are immense (for a pillar sequence size n_s, the number of possible combinations $n_c = 32^{n_s}$). Additionally, the inlet fluid flow design - which dictates fluid flow shape as much

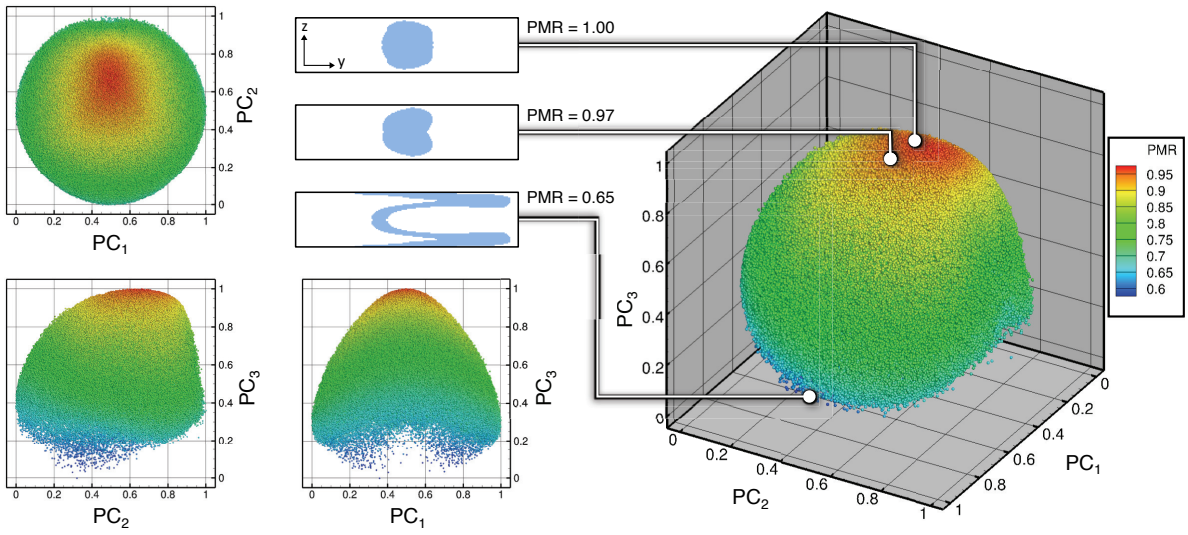

Figure 5.3 Visualization of the space $\mathcal{O}$ via PCA. This illustration shows the projection of a training set of 150,000 images (chosen by random sampling of $\mathcal{S}$) onto its PCA-space, with the first three principal components as the axes PC_1, PC_2, and PC_3, and every point corresponds to an image in the dataset. Each point (image) is colored with a measure of similarity, the Pixel Match Rate (PMR), to a single image in the set (arbitrarily chosen as the image with the largest PC_3 coordinate). A high PMR (red) means the flow shape is more similar to the selected image, while a low PMR (blue) is less similar. Inserted are the selected target flow shape image (top) and two other sample images, each with their respective PMR value to the target image and indicated location in the PCA-space.

as a pillar sequence - is entirely arbitrary, depending on the number of channels joining at the microchannel inlet and their respective flow rates. The current implementation of deep learning for the flow sculpting problem holds the inlet design constant, and trains for pillar sequences of fixed length. Thus, while an ideal training set has completely uniform coverage of $\mathcal{O}$, and despite the speed of the forward model, generating and making practical use of such a set for each length of pillar sequence and inlet configuration is computationally infeasible. And, as previously mentioned, the toolset in flow sculpting will likely grow to include curved channels, symmetry-breaking half-pillars and steps, and multi-fluid optimization (Xie et al., 2016). Therefore, intelligent sampling on the space $\mathcal{S}$ is still a priority. Previous methods of sampling attempted uniform coverage of $\mathcal{O}$ via uniform sampling of $\mathcal{S}$, through uniform-random and quasi-random Sobol sampling of n_s-dimensional spaces in $\mathcal{S}$ for pillar sequences of length n_s(Lore et al., 2015).

Here, we apply a technique for multivariate representation known as High-Dimensional Model Representation (HDMR). In HDMR, a high-dimensional input variable $\mathbf{x} = (x_1, x_2, \ldots, x_n)$ with a model output $f(\mathbf{x})$ is expressed as a finite hierarchical expansion:

$$f(\mathbf{x}) = f_0 + \sum_i f_i(x_i) + \sum_{1\le i<j\le n} f_{ij}(x_i, x_j) + \sum_{1\le i<j<k\le n} f_{ijk}(x_i, x_j, x_k) + \ldots + \sum_{1\le i_1<\ldots<i_l\le n} f_{i_1 i_2 \ldots i_l}(x_{i_1}, x_{i_2}, \ldots, x_{i_l}) + \ldots + f_{12\ldots n}(x_1, x_2, \ldots, x_n) \quad (5.1)$$

Where l^{th} order component functions $f_{i_1 i_2 \ldots i_l}(x_{i_1}, x_{i_2}, \ldots, x_{i_l})$ give outputs of f evaluated for contributions by l-order sampling of $\mathbf{x}$. That is, f_0 is a scalar value as the mean response to $f(\mathbf{x})$, $f_i(x_i)$ represents each variable x_i independently (holding all other variables constant), and so on. $f_{12\ldots n}(x_1, x_2, \ldots, x_n)$ contains residual contributions for permutations of all variables x_n. In many cases, a high dimensional system can be well described using up to the 2^{nd} order terms(Rabitz et al., 1999):

$$f(\mathbf{x}) \approx f_0 + \sum_i f_i(x_i) + \sum_{1\le i<j\le n} f_{ij}(x_i, x_j) \quad (5.2)$$

We use this expression to hierarchically sample $\mathcal{S}$, choosing k points along each chosen dimension of $\mathbf{x}$ such that we limit the training set size to be comparable to the pseudo-random and quasi-random sampling used by Lore et al. (Lore et al., 2015), which comprises of 150,000 $(\boldsymbol{s}, \boldsymbol{o}) \in (\mathcal{S}, \mathcal{O})$. The pseudo-random, uniformly distributed sampling is accomplished via *mt19937ar* Mersenne twister in MATLAB (Matsumoto and Nishimura, 1998), while the quasi-random sampling was accomplished via Sobol sequences (Sobol et al., 2011), which seek to maximally separate the sampled points in a deterministic manner in order to prevent clustering or gaps. We neglect the mean response term in the HDMR expansion, as there is no effective "mean" pillar. Since the 1st and 2nd order terms sample a limited number of points in each dimension (choosing one of 32 pillar configurations at each point), there is a choice as to what pillars will be chosen as constant for the rest of the sequence. We desire statistical metrics on the use of HDMR for a fixed k, so we randomly pick k points in the i^{th} dimension.

For each pillar sequence that is predicted during testing, we use the forward model to create a corresponding flow shape $\hat{\boldsymbol{o}}$ and then compare that to the target image $\boldsymbol{o_t}$ from the testing set (see Supplementary Fig. S1). Thus, while the classification error during training is based on label error (that is, correct prediction of the pillar sequence), we judge model performance on whether the trained neural network effectively solves the inverse problem. The metric for performance is a per-pixel norm, the Pixel Match Rate (PMR)[3], which is defined for a target image $\boldsymbol{o_t}$ and a predicted image $\hat{\boldsymbol{o}}$ as:

$$PMR = 1 - \frac{||\boldsymbol{o_t} - \hat{\boldsymbol{o}}||_1}{|\boldsymbol{o_t}|} \tag{5.3}$$

Our goals are to determine the effectiveness of HDMR in sampling flow sculpting's input space $\mathcal{S}$, and we use PCA to both characterize these results and test sampling of the flow shape space $\mathcal{O}$ as a method of improving performance. To accomplish this, we conducted three main experiments: (1) a study on training set size, (2) out-of-sample testing using $\mathcal{O}$-sampling, which selects training and testing data using PCA, and (3) uniform output sampling, which uses PCA

[3]The PMR can vary from 0 (no pixels in common between prediction and target) and 1.0 (a perfect match), but a threshold for a successful result is up to the microfluidic practitioner. For example, if the goal is to simply displace bulk fluid without care for the overall shape, a PMR greater than 0.8 may be suitable. However, if the user requires fine details in the prediction to match the target, a PMR of 0.9 and above is an appropriate threshold. More generally, a PMR of 0.85 is a good result.

for the creation of training data with a uniform distribution on $\mathcal{O}$. The dimension of the pillar sequence space $\mathcal{S}$ corresponds to the number of pillars used to sculpt flow in a microfluidic device, and in practice would depend on fabrication/design constraints. Typical designs have used between 1-10 pillars, in an attempt to mitigate issues with transverse mass diffusion and increased pressure requirements (Stoecklein et al., 2016). For this work we elected to keep a constant pillar sequence size of $n_s = 7$, as this provides a combinatorially complex space (≈34 billion pillar sequences in choosing from 32 different pillars), while remaining computationally tractable. For the flow shape space, $\mathcal{O}$, we match the dimensionality of 200×24 images used by Lore et al. (Lore et al., 2015).

5.2.4.1 Training set size study

The purpose of this study is to directly compare HDMR to previously used methods for sampling $\mathcal{S}$, and observe accuracy-scaling trends with training set size. The testing set used was of 10,000 flow shapes randomly sampled from $\mathcal{S}$. Performance of uniform-random (RNG), quasi-random Sobol, and HDMR sampling with a training set size of 150,000 is shown in Fig. 5.4(a). As the quasi-random Sobol sampling does not perform significantly differently from uniform random sampling, we will drop Sobol sampling from future analyses, and compare only uniform random sampling with HDMR sampling. A clear shift in the median PMR of roughly 2.5% from 0.79 (RNG) to 0.81 (HDMR) is observed, along with a more left-skewed distribution in favor of high PMR values. In Fig. 5.4(b), the training size study shows that although the randomly generated HDMR sets have greater variance in data, there is a perceivable increase in performance with larger training set size. RNG sampling shows tighter variance, but no discernible improvement with larger training sets.

We also analyze performance by computing the Shannon entropy of the posterior distribution at the decision (top) layer of a CNN model (as shown in Fig. 5.2) for a test example during the inference process. Entropy was computed for the best and worst performing training sets for HDMR and RNG sampling with a training size of 150,000 images to show contrast between the sampling methods and their performance. Results are shown in Fig. 5.4(c,d), where models trained with HDMR sampling exhibit significantly reduced entropy in their predictions, even

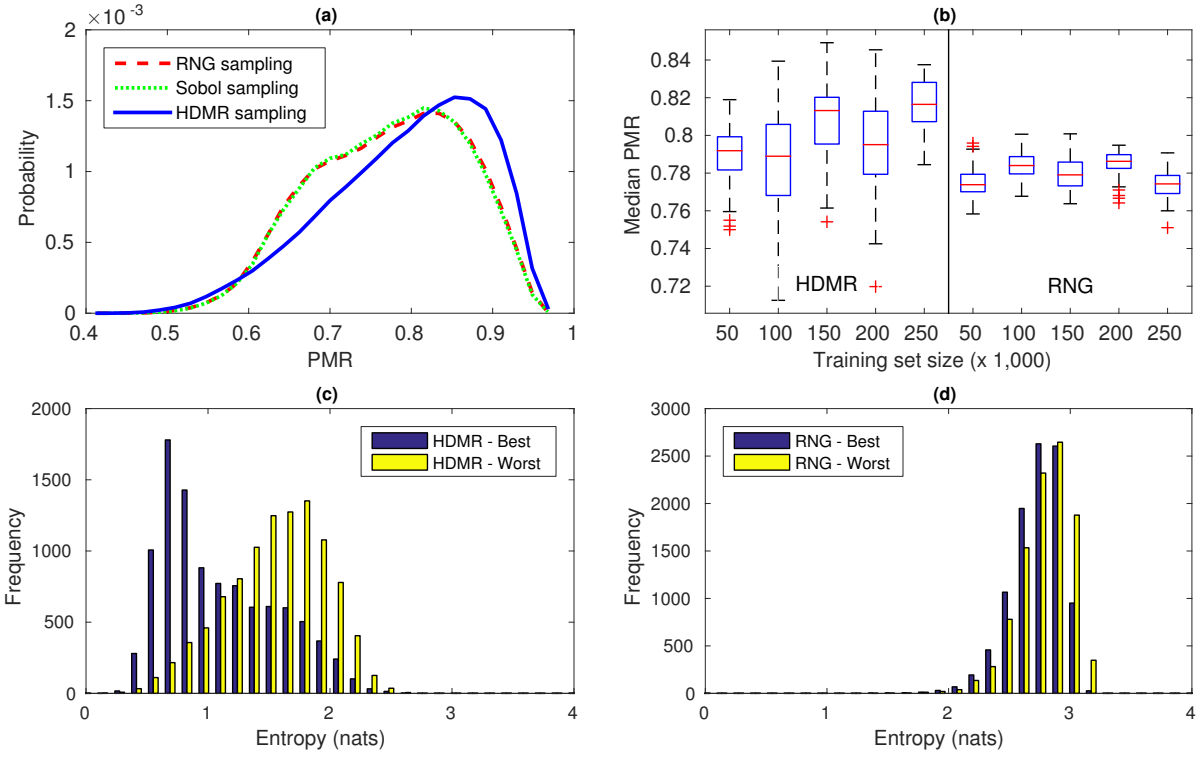

Figure 5.4 (a) Performance of uniform-random (RNG), quasi-random (Sobol), and HDMR sampling on $\mathcal{S}$ for the creation of training sets of 150,000 flow shapes, with a uniform-random sampled test set of 10,000 flowshapes. The classification was for 7-pillar sequences. (b) Performance of HDMR sampling and pseudo-random (RNG) sampling vs. training set size, from 50,000 to 250,000 images in increments of 50,000, with the same testing set as in (a). Error bars indicate standard deviation calculated from 50 randomly generated training sets. Note that while RNG sampling can easily generate additional "random" sets of training data, randomly selecting values for k sampling indices in HDMR could result in clustering that greatly impairs performance. The distribution of posterior entropy for the CNN models in testing performance of the best and worst (c) HDMR and (d) RNG training sets are shown for training set size of 150,000. HDMR sampling shows far lower entropy, even with decreased performance, indicating strong model confidence and significant disturbance rejection capability.

in the case of the worst-performing HDMR model (mean entropy of 1.09 and 1.55 nats for the best and worst HDMR sampling, compared to mean values of 2.73 and 2.79 nats for the best and worst RNG sampling). A low entropy posterior suggests that the model has a higher confidence in decision-making (in this case, classification of appropriate pillar configuration given the desired flow shape). This also signifies that the corresponding training data set is representative enough with respect to the test sample. At the same time, a model that is able to consistently produce low entropy posteriors generally has a better noise and disturbance rejection capability. This can be argued from the fact that there is a low chance of switching classification decision (when it is made based on the maximum a posteriori, MAP principle) under small disturbances either numerically or in the input. Hence, such a machine learning model is typically preferred in most applications.

5.2.4.2 Out-of-sample study

To study deep learning's ability to make predictions beyond the training data, we used a 2-D PCA projection to sample from given training data through either *interior-hole* $\mathcal{O}$-sampling (implying interpolation within the training data) or *exterior-chord* $\mathcal{O}$-sampling (implying extrapolation outside of the training data), and used these data as test sets (see Fig. 5.5(a,b)). Since the fluid flow shapes in a given region of PCA-space are similar, eliminating these images from the training set leaves a complementary set of considerably different flow shapes for the neural network to learn from. Thus, for the trained model to succeed, it must truly "learn" the underlying features of the physical system (as opposed to "memorization" of the input/output mapping). For each out-of-sample type (*interior-hole* and *exterior-chord*), we sampled 50 random locations and created complementary sets of training/testing data for both HDMR and RNG sampled training data. In both scenarios, the trained models clearly have some successful predictions of pillar sequences for target flow shapes outside of the given training data (see Fig. 5.5(c,d)). This is a clear demonstration of deep learning's ability to integrate learned features with a mapping that is useful for out-of-sample predictions. Combined distributions of performance are shown in Fig. 5.6, which relay a general reduction in performance compared to in-sample testing from the prior training set size study (Fig. 5.4). However, for both types

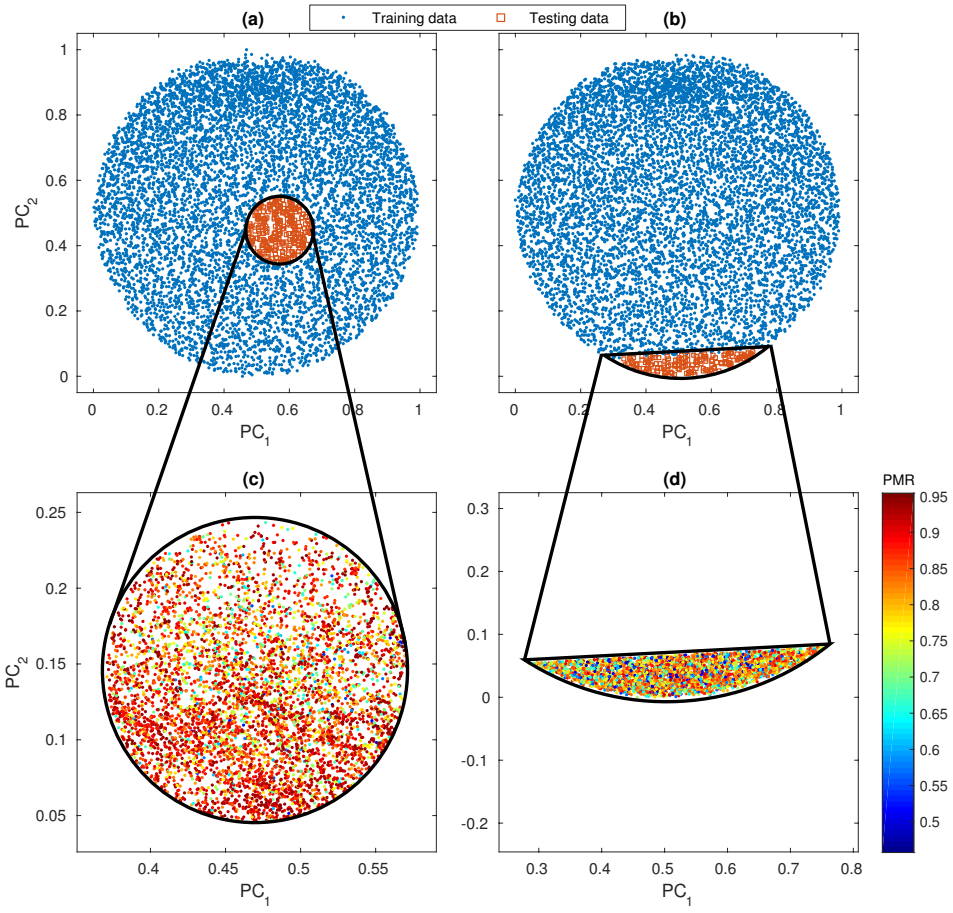

Figure 5.5 Illustration of out-of-sample testing using (a) *interior-hole* $\mathcal{O}$-sampling and (b) *exterior-chord* $\mathcal{O}$-sampling. In each case, data is extracted from a training set based on its projection onto the 2-D PCA-space, leaving a substantial gap in image space coverage where the trained model is then tested. Despite this deficiency, performance is not necessarily hindered, as there are still successful predictions from the trained model as shown in the example (c) *interior-hole* and (d) *exterior-chord* test sets. Points in (c) and (d) with a higher PMR value (and therefore better prediction) are red, while a lower PMR value is blue.

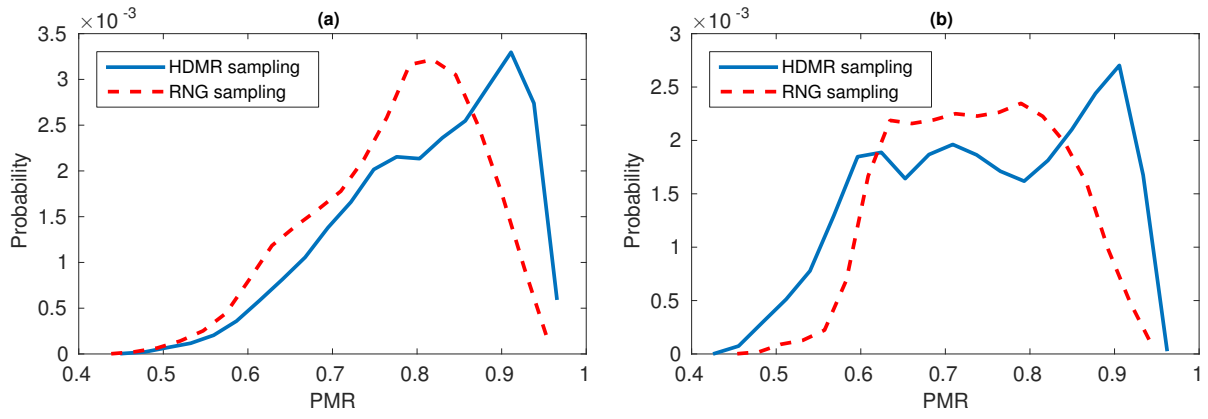

Figure 5.6 Distributions of out-of-sample results using (a) interior-hole and (b) exterior-chord sampling for test data extraction from the training set. Note that although performance is generally shifted from the in-sample testing of the training set size study, there are still successful predictions above 85%, especially for HDMR.

of out-of-sample testing, use of HDMR for training data generation shows a clear increase in successful predictions for which PMR > 0.85. These results imply that while effective coverage of the design space is important for deep neural network training data, the manner by which the data is sampled also plays a crucial role.

5.2.4.3 Uniform output sampling study

Based on the many-to-one nature inherent to this problem, we hypothesized that a training set that is more uniformly distributed in $\mathcal{O}$ will have better performance in testing due to less redundancy in similar images during training. That is, for a training set that is particularly clustered in $\mathcal{O}$, two similar images with disparate pillar sequences may "undo" whatever is learned from the other, resulting in a less effective model overall. Conversely, a training set that is well distributed in $\mathcal{O}$ would have more unique images, perhaps mimicking a one-to-one design space, and create better predictions. We tested this hypothesis by assembling 50 randomly generated 150,000 image datasets into a single 7.5 million image dataset. The corresponding PCA projection of this dataset is constructed and sampled such that each image is isolated within an n-D hypercube of a radius defined by a uniform discretization of $\mathcal{O}$ (see Fig. 5.7(a,b) for a demonstration of this process). This results in a method for selection of

training data that is more uniformly distributed in $\mathcal{O}$ (See Fig. 5.7(c,d) for examples of such a dataset).

We performed uniform $\mathcal{O}$-sampling for both RNG and HDMR super datasets, and tested the resulting ≈150,000 image training set with the same data as in the first study, and found no significant change in performance. The median PMR values in testing was 0.79 and 0.81 for RNG and HDMR generated data, respectively, with no significant change in variance. This follows with results from the out-of-sample study, as it was shown that a clearly non-uniform distribution of training data in $\mathcal{O}$ could still make successful predictions.

5.3 Conclusion

There is a large variety of science and engineering problems with a tractable forward problem simulation process and/or a large database which can benefit from a powerful machine learning framework to solve seemingly intractable inverse problems. However, as such applications of machine learning are still at a nascent stage, important questions related to model choice, training data generation, decision problem formulation, have to be answered to achieve significant impact. In this context, we have shown that intelligent sampling of training data in deep learning models for flow sculpting can greatly improve performance, with the capacity to raise the ceiling on training set size for additional improvement. We also demonstrated deep learning's capability to effectively make predictions well outside of the trained scope, with additional improvements in out-of-sample predictions through HDMR sampling. Finally, we tested the hypothesis that a training set made artificially uniform in the training data's image space leads to a more effectively trained model, but found no evidence in support of this claim. In fact, this result meshes well with the out-of-sample testing, where the deep learning model was still capable of successful predictions despite a substantial gap in the image space of the training data. With improved accuracy, high model confidence, and potential for further growth, deep learning will see immediate use for design in flow sculpting, while analogous problems in scientific design can benefit from the lessons of its application here.

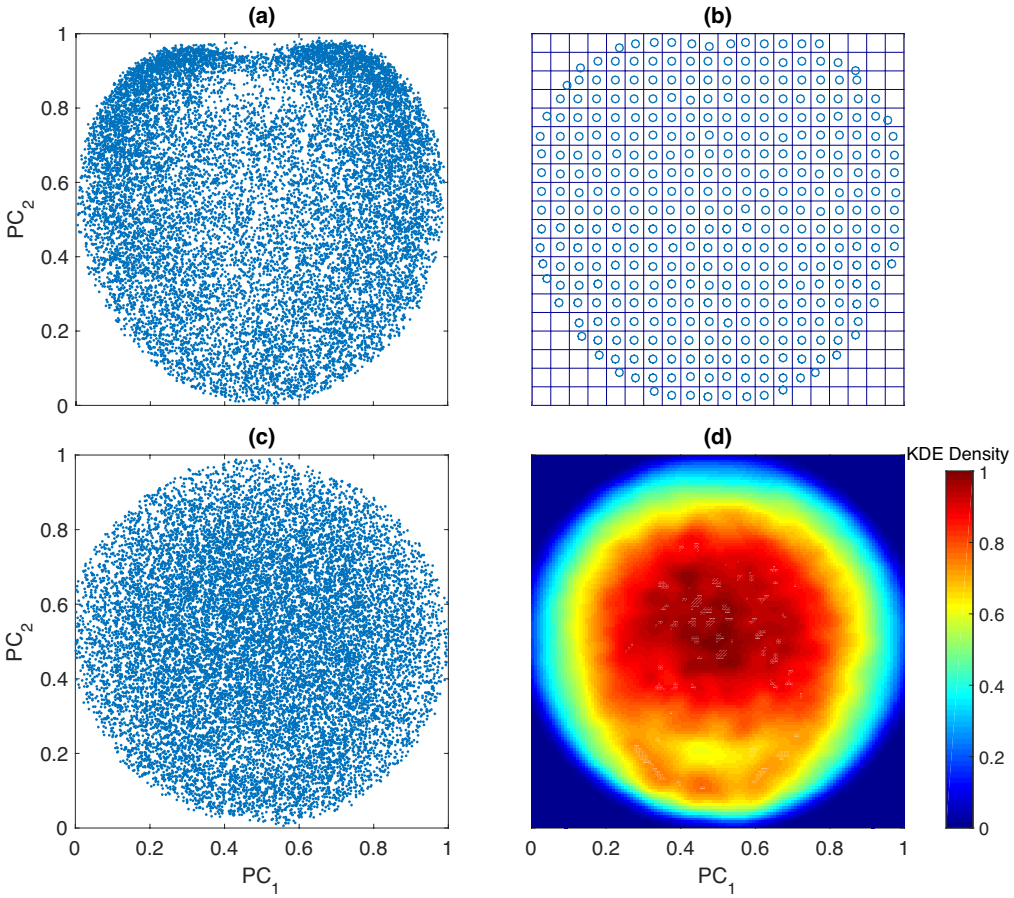

Figure 5.7 Demonstration of uniform *output* sampling. (a) An arbitrary distribution of points in n-D PCA-space can be uniformly sub-sampled (b) by isolating sampled points within individual n-cubes, the size of which depends on the level of discretization. By composing 50 sets of 150,000 images randomly sampled from $\mathcal{S}$ (for a total of 7.5 million images), *output* sampling in 4-D PCA-space is used to create a more uniform set of 150,000 images (c), as seen in (d) with an applied bivariate Kernel Density Estimator (KDE) on the resulting set.

5.4 Methods

5.4.1 Forward Model via Computational Fluid Dynamics

The forward model of simulating a fluid flow shape given a sequence of micropillars follows the experimentally validated method used by Stoecklein et al. (Stoecklein et al., 2016, 2017a), which we briefly outline here. A library of pre-computed deformation maps are created for a set of pillar diameters and locations in the channel, with a constant channel aspect ratio (height/width) and matching flow physics (Re). To create a single map, a 3-D velocity field is simulated for a given pillar configuration using an experimentally validated finite element framework. The 3-D domain extends well beyond the pillar ($\geq 6D$) to allow for the structure-induced deformation to saturate within the domain(Amini et al., 2013). Then massless, neutrally buoyant fluid elements are traced through the velocity field to determine the 2-D displacement of fluid in the cross-section of the channel from inlet-to-outlet, defining an advection map for a single fluid deformation. An advection map informs the creation of a square transition matrix, which has of a row and column for every discretized cross-section location in the streamtrace. For every starting location of a fluid element in the inlet cross-section, there is a single value of unity in a column indicating 100% likelihood of displacement to that location in the outlet cross-section.

A fluid flow shape at the inlet can be represented as a vector of fluid states in the channel cross-section, $\boldsymbol{\mu_0}$, with zeroes for untracked fluid and ones for the fluid being sculpted. This inlet vector can then be multiplied by the transition matrix to create an outlet vector, $\boldsymbol{o}$, which represents the deformed fluid flow shape at the outlet. By using binary values for transition probabilities (i.e., 0 or 1), the matrices are extremely sparse, leading to very efficient computation. On modest computing hardware, a single pillar can be simulated in roughly 3 milliseconds(Stoecklein et al., 2016). Any number of transition matrices can be multiplied together, each created from an arbitrary pillar configuration, ultimately creating a net transition matrix which produces the deformed shape for the entire sequence. In general, any kind of geometry for flow physics can be implemented within the flow sculpting forward model, e.g., curved channels, steps, non-pillar structures, and multi-fluid flow (Xie et al., 2016). However,

for the overall prediction to be valid, the flow physics (defined by Re) must match for each pre-computed deformation, and the microchannel geometry (defined by the channel aspect ratio) must either also match for each new transition matrix, or additional deformations must be computed to stitch together the step from one channel geometry to another.

5.4.2 Deep Learning Implementation

The CNN used consists of 2 convolutional layers (40 kernels of size 11×11 for the first layer, 100 filters of size 4×4 for the second layer), each pooling layer followed by downsampling by 2×2 maxpooling, and one fully-connected layer with 500 hidden units (see Fig. 5.2). The loss function was a negative log-likelihood as defined in Lore et al. (Lore et al., 2015), and the learning rate was 0.1. Training was performed on sample sizes from 50,000 to 250,000 (with an additional 20,000 validation samples), while testing was performed on 10,000 samples generated through random sampling of $\mathcal{S}$. The framework as described above was implemented in the Theano platform (Theano Development Team, 2016), and performed on NVIDIA K20 Tesla GPUs. All experiments were deployed on Iowa State University's High Performance Computing cluster Cyence, with post-processing and analysis conducted on an NVIDIA Geforce GTX Titan Black.

CHAPTER 6. UFLOW: SOFTWARE FOR RATIONAL ENGINEERING OF SECONDARY FLOWS IN INERTIAL MICROFLUIDIC DEVICES

This chapter includes a manuscript titled "uFlow: software for rational engineering of secondary flows in inertial microfluidic devices", and is under review in *Lab on a chip* authored by Daniel Stoecklein, Keegan Owsley, Chueh-Yu Wu, Dino Di Carlo, and Baskar Ganapathysubramanian

Abstract

Approaches to abstract and modularize models of fluid flow in microfluidic devices have the potential to enable predictive and rational engineering design of microfluidic circuits with rapid designer feedback. The shape of co-flowing streams has become of particular importance for new developments in high-throughput microscale manufacturing, in addition to biological and chemical research. The cross-sectional distribution of fluid elements is deformed due to channel structures and velocity gradients under the combined effects of diffusion and transverse advection both in Stokes flow and inertial conditions. However, the computational difficulty that comes with solving the Navier-Stokes equations in complex geometries makes design in this space unintuitive, time-consuming, and costly. To mitigate these issues, and with focus on inertial flows, we have efficiently embedded fluid flow operations previously relegated to high-performance computing into a freely available, user-friendly, and cross-platform framework to bring the utility of flow sculpting to the broader community. This software, called "uFlow", enables real-time manual design and exploration of microfluidic devices, includes both advection and diffusion effects, and enables fast visualization of 3D particles fabricated via stop flow lithography or transient liquid molding. Advanced numerical routines give instant

access to several thousand flow transformations. We showcase uFlow's design models, describe their implementation and usage, and integrate them with real world examples of cutting-edge microfluidic particle fabrication processes.

6.1 Introduction

Enabled by electronic design automation software, modularity and abstraction have been critical concepts that fueled a revolution in producing complex electronic circuits to perform information processing tasks. Similar approaches can aid in the design of microfluidic devices, where software can store pre-computed operations of microfluidic components, giving non-expert researchers instant access to complex physics models. An important aspect of these software packages is the ability to make rapid prediction of the outputs of the entire system when modifications are made in modular components and their arrangement. For example, design of monolithic microfluidic devices has been facilitated by hydraulic circuit analysis. However, unlike with electron flow, for many materials fabrication or diagnostic applications engineers aim to also control the fluid elements within the cross-section of a channel. Software that abstracts the Navier-Stokes and convection-diffusion equations, focusing only on the input-output transfer functions while tracking the cross-sectional flow structure, could benefit the next generation of microfluidic designers. Only recently has there been effort in this direction for mixing flows and shaping cross-sectional flow shape (Mott et al., 2006; Howell et al., 2008; Amini et al., 2013).

Over the last two decades, the microfluidics community has developed multiple approaches for the passive manipulation of a fluid's cross-sectional material composition at the microscale. Liu et al. (2000) used serpentine channels to induce mixing, in which a curving channel was the modular element that led to turning fluid streamlines with finite inertia. Stroock et al. (2002) demonstrated a more straightforward method for mixing by embedding herringbone structures as modular elements in the wall of a microchannel. This approach allowed for mixing from the Stokes flow regime ($Re < 1$) to inertial flows ($Re < 100$) with qualitatively similar results. Howell et al. (2004) showed how Dean flow in curved channels can achieve efficient mixing, and Sudarsan and Ugaz (2006) enhanced this method with sudden channel expansions. For more

precise control over the fluid, Howell et al. (2008) extended the idea of patterned microchannel surfaces with the use of grooves for hydrodynamic focusing. However, each of these methods typically employ a small set of flow operations that leverage chaotic advective forces, limiting their potential for arbitrary user-designed flow shapes. Amini et al. (2013) introduced the idea of using sequenced microstructures for hierarchically designed fluid flow deformation and cross-sectional shaping in the inertial regime. This technique for flow manipulation, dubbed "flow sculpting", has dramatically enhanced the level of control microfluidic researchers can exert on microfluid streams.

Flow sculpting uses a sequence of bluff body microstructures, each acting as independent operators, to deform the cross-sectional structure of inertially flowing fluid. To maintain independence between the flow deforming operators, there are some restrictions on the design of a flow sculpting device. The primary restriction is that the microstructures - cylindrical pillars are commonly used - must be spaced far enough apart in the microchannel to allow for an individual pillar's flow deformation, i.e., advection map (Mott et al., 2006; Stoecklein et al., 2014) (see Fig. 6.2 in section 6.3.1), to saturate before the fluid arrives at the subsequent pillar. The flow deformation should also have no time-dependent effects. These two features prevent cross-talk between pillars, and enable an elegant computational shortcut for simulating flow sculpting devices: the fluid flow deformation for any structure can be independently simulated ahead of time, and since we are only concerned with the transformation of flow after passing the pillar, 2D advection maps of the cross-sectional fluid flow shape can be sequentially composed for *real-time* simulation of fluid flow deformation. This is especially useful for the inertial flow regime, as accurate Computational Fluid Dynamics (CFD) simulations that include convective terms to account for inertia are computationally expensive to solve, making their use in design costly, and beyond the reach of researchers who are without access to High Performance Computing (HPC) resources. On the other hand, rapid use of pre-computed results from complex CFD simulations makes exploratory investigations more feasible, and previously difficult design problems tractable.

In 2014, a simple tool utilizing the aforementioned computational scheme, "uFlow", was released as an entry point for researchers to explore inertial flow sculpting and design their

own microfluidic devices. With a library of 32 pre-computed flow deformations for pillars of different size and location, uFlow was shown to produce a wide variety of sculpted fluid flow shapes (Stoecklein et al., 2014), which has since enabled researchers to use flow sculpting for novel applications in advanced manufacturing and biological sciences: Nunes et al. (2014) fabricated shaped microfibers with tailored cross-sections by sculpting a UV-polymer pre-cursor; in an extension of this technique, Paulsen et al. (2015) used stop-flow lithography to fabricate 3D particles, with Wu et al. (2015) following up with a rapid manufacturing approach for 3D microparticles. Such processes could be used to create functionalized particles for cell capture and diagnostics (Bong et al., 2015; Chen et al., 2016) or accelerated wound healing (Do et al., 2015; Griffin et al., 2015), encoded hydrogel particles (Lee et al., 2014; Appleyard et al., 2011), photonic crystals (Galisteo-López et al., 2011), and self-assembling hydrogels (Gurkan et al., 2012), for example. Sollier et al. (2015) used flow sculpting to shift fluid away from larger particles in flow, thus introducing a high-throughput approach to purify samples, or collect reusable reagents.

As a utility for flow sculpting design, uFlow is intended to democratize the search for useful fluid flow shapes, and enable microfluidic researchers to access this powerful flow manipulation technique without incurring computational costs associated with high-volume CFD analysis. Since its debut, we have incorporated a number of features into uFlow to bring the user from simple point-and-click assembly of flow sculpting devices, through post-processing their design with high-resolution images and Computer Aided Design (CAD) models, to advanced modeling of mass diffusion and particle fabrication via UV-polymerization. uFlow also allows users to integrate their own flow-deforming microstructures as fluid structure operators, making the design of a flow sculpting device more modular and customizable. uFlow can be freely accessed as a compiled executable for Windows and Linux systems[1], and its source code is publicly available[2]. Here we describe the implementation of uFlow which will aid other researchers who wish to adopt the framework to address other microfluidic design problems of interest, and add functionality. We anticipate that uFlow will assist the broader microfluidics community for

[1] www.biomicrofluidics.com/software.php
[2] www.bitbucket.com/baskargroup/uflow.git

design, to enable rapid visual feed back for fluid mechanics education, and to help accelerate the maturation and adoption of microfluidic technologies.

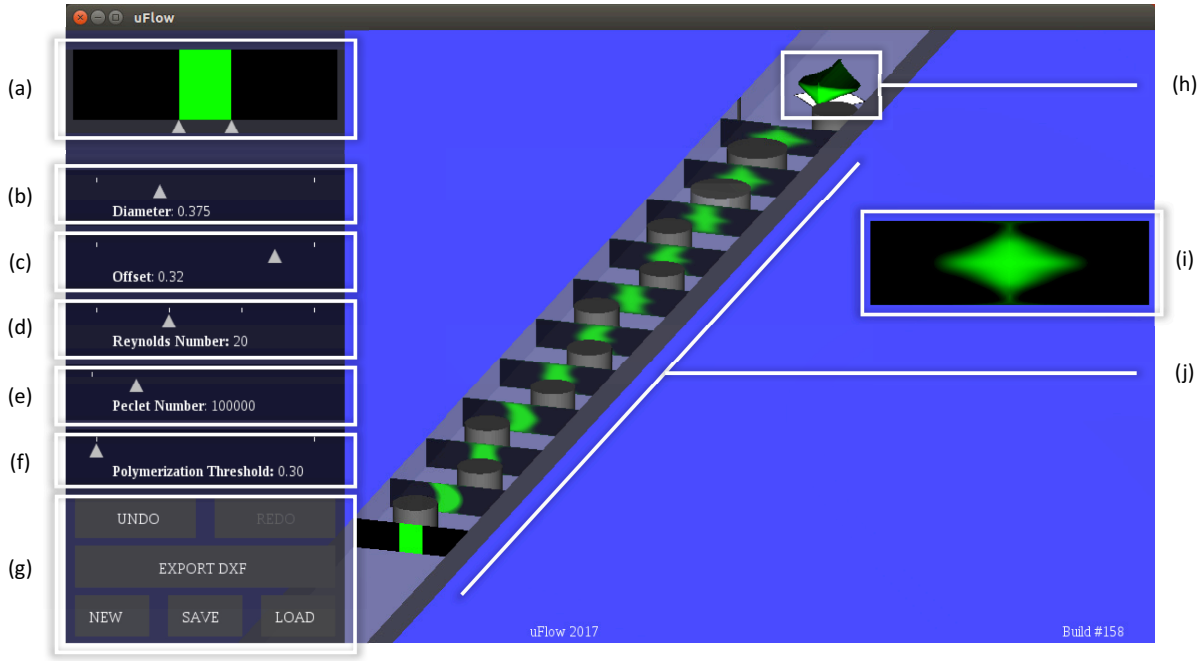

Figure 6.1 uFlow Graphical User Interface (GUI). uFlow's GUI has a number of controls for user input, and a visually responsive display of the simulated flow sculpting device: (a) Inlet flow pattern design of the channel cross-section with colored regions indicating different regions of the flow; (b,c) pillar diameter and location for the next pillar, or for altering a selected pillar; (d) Reynolds number for the flow sculpting device; (e) Peclét number for the flow sculpting device; (f) polymerization threshold for microparticle rendering; (g) undo/redo, device saving/loading/export; (h) microparticle rendering area; (i) net flow deformation visualization (can be selected to save a high-resolution image); (j) pillar sequence area (for placement/adjustment of pillars).

6.2 uFlow Overview

The three primary physics models available in the uFlow library (dubbed "uFlowlib") are those of fluid advection, transverse mass diffusion, and microparticle fabrication visualization, all of which are accessible through its Graphical User Interface (GUI) (See Fig. 7.4). Advection map compression is also implemented via principal component analysis (PCA), embedding several thousand advection maps within uFlow, with interpolation giving access to a practi-

cally infinite set of flow deformations. uFlowlib accelerates these models using the Graphics Processing Unit (GPU) in order to make real-time exploration possible.

GPUs contain a highly parallelized hardware architecture in order to accelerate tasks that would otherwise require considerable runtime on Central Processor Units (CPUs), whose general purpose architecture is the typical workhorse of scientific computation (Kalivarapu and Winer, 2010). Using GPUs in this manner is the result of a peculiar evolution of hardware, as GPUs were originally built for a specialized purpose: rendering visually rich 2D/3D graphics on a display for entertainment or visualization. To create increasingly complex visuals, GPU architecture began incorporating additional numbers of small processors that operate in parallel, to the point where modern commodity GPUs have many hundreds, and in some cases, thousands, of processors that collectively achieve performance greatly exceeding that of a comparably-priced CPU (Kalivarapu and Winer, 2010). Many researchers have observed a natural fit for accelerating computation on these "supercomputer-on-a-chip" devices, with General Purpose computation using Graphics Processing Units (GPGPU) becoming a new frontier of Computational Science and Simulation (Galoppo et al., 2005; Owens et al., 2006; Du et al., 2012).

We adapt the computation of flow sculpting to GPUs in a platform-agnostic manner by using the Open Graphics Library (OpenGL), which is a cross-platform programming interface for GPUs that is typically utilized in 2D/3D rendering for visualization. uFlow itself is written in the Python language, while OpenGL routines are programmed in the Graphics Library Shader Language (GLSL), and sent to the GPU through the Python module "Pyglet"[3]. Most of uFlow's library makes use of *shaders*, which are small programs that run in parallel on the GPU and can use data stored in *textures* which reside in GPU memory (analogous to arrays in CPU memory). By aligning flow sculpting simulations with computational routines endemic to the GPU, we have created an extremely fast and responsive framework for exploration and design. In the next two sections, we detail physics models and utilities of uFlowlib, along with their implementation and usage in the software.

[3] https://www.pyglet.org/

6.3 uFlow physics models

6.3.1 Advection

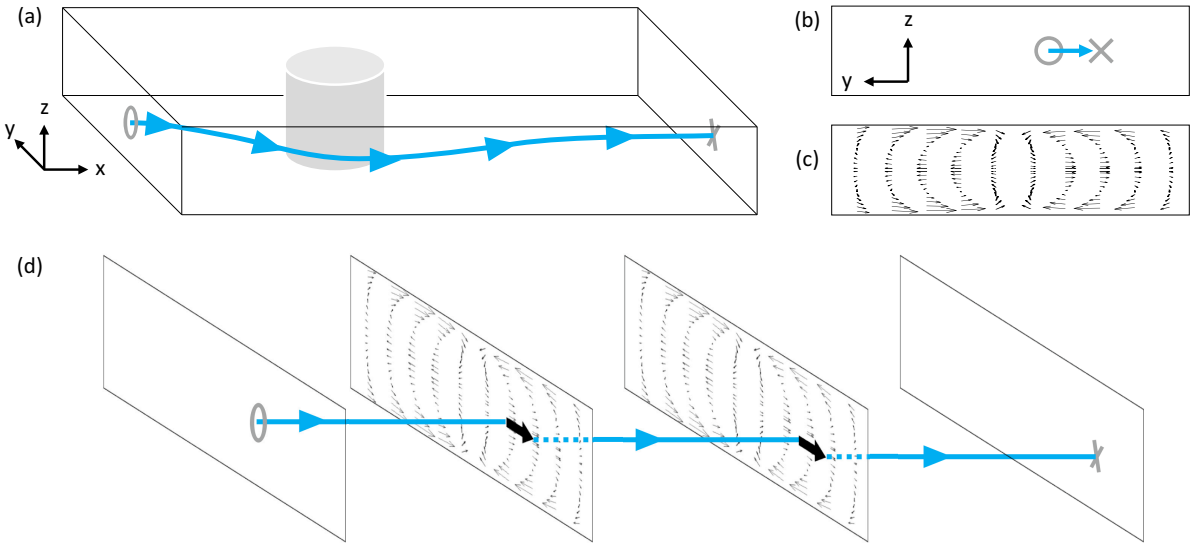

Figure 6.2 Illustration of the creation of advection maps, and how they are used in flow sculpting. (a) Fluid flow is simulated within a 3D domain for a single pillar and a given fluid flow condition (Reynolds number Re), producing a 3D velocity field. Infinitesimal, massless particles are streamtraced through the velocity field, with the goal of determining each fluid element's (particle) lateral displacement (b) in the microchannel cross-section. This is performed for an ensemble of particles across the entire cross-sectional area, with their collective displacements informing a single advection map for that pillar's geometry and the given flow physics. The advection map can then be used in place of the 3D velocity field for the simulation of a flow sculpting device, where (d) transverse fluid displacement is computed for a single fluid element by sampling the advection map at the fluid's current location in the channel cross section. This process of marching through advection maps can be repeated for an arbitrary number of pillars or other microchannel modules, provided that each 3D simulation has matching boundary geometry and flow physics.

6.3.1.1 Background.

The most basic operation of uFlow is to simulate the convection-driven net fluid flow deformation induced by a user-designed pillar sequence. Previous work of Amini et al. (2013) and Stoecklein et al. (2014) explored the nature of steady-state inertial flow past cylindrical

bluff-bodies (pillars) in microfluidic channels, with overall lateral displacement due to net inertial convection yielding a so-called "advection map" for a single pillar and fluid flow condition, Re[4]. An advection map is a 2D vector field describing the lateral displacement of fluid in the microchannel cross-section, acting as a computational shortcut for simulating the full CFD flow deformation for a single pillar. By collapsing the streamtrace through a 3D velocity field to a 2D map, the net flow deformation from a sequence of pillars in a flow sculpting device is well-simulated in an extremely rapid manner by simply traversing from map to map (see Fig. 6.2). In fact, the flow transformation induced by any arbitrary microchannel geometry (e.g., grooves, curved channels, ridged channels) can be represented as an advection map and added to a library of pre-computed deformations, any of which are readily placed into a flow sculpting device simulation. However, there are several requirements for this to work. First, the channel geometry and flow physics must match at the inlet and outlet of each microfluidic "module". One cannot append an advection map for which $Re = 40$ to a device which uses maps computed for $Re = 20$, nor mix-and-match microchannel height-to-width aspect ratios, without a connecting channel that matches the boundary conditions at the outlet to the preceding module, and at the inlet of the following module. Second, the 3D streamtrace must completely capture the flow-induced deformation within the 3D domain, i.e., flow displacement should saturate before fluid exits the domain. Finally, the flow should not exhibit any time-dependent effects.

6.3.1.2 Implementation.

uFlow uses custom full-color RGBA 32-bit floating point textures to store advection map data, which describes flow displacement on the $y-z$ plane (see Fig. 6.2(a)) for a discretized grid of $N_Y \times N_Z$ fluid element locations. The red and green channels store (dy, dz) displacement data at their respective locations in the microchannel cross-section. Because each displacement is computed independently in the 3D streamtrace, displacing fluid in uFlow (now represented as pixels) is an embarassingly parallel operation, and well suited to concurrent shader programs

[4]In this work, we define the Reynolds number - which is the ratio of inertial to viscous forces - as $Re = \frac{UD_H}{\nu}$, with fluid kinematic viscosity ν and average velocity U, and microchannel hydraulic diameter D_H.

that execute on the GPU. To simulate flow deformation from a microfluidic module (e.g., a pillar), uFlow uses the following process:

1. A set of fluid "particles" are distributed uniformly across the 2D domain (microchannel cross-section) as a set of `GL_POINTS` (an OpenGL primitive, describing a single point in 1D, 2D, or 3D space). The points will collectively describe the fluid as it is transformed via flow sculpting.

2. All fluid particles are colored and rendered as pixels in parallel by a *fragment* shader, coloring each point based on which inlet stream they are located in at the microchannel inlet.

3. After a pillar - or some microfluidic part - is placed in the microchannel, a *vertex* shader concurrently samples the advection map for that part (stored in the GPU as a 32-bit 2D texture) on a uniform grid, computes the displacement (using hardware-accelerated bilinear interpolation), and stores the net displacement in a new 2D texture.

4. This texture - now storing the net flow displacement - is sampled by another vertex shader, transforming all fluid particles to their updated locations in parallel.

5. A fragment shader then colors each pixel based on its initial location at the inlet.

Steps 1 and 2 are part of uFlow's initialization, step 3 only occurs when a new microfluidic part is added to the flow sculpting device, and steps 4-5 are processed for each frame rendered (at 60 frames per second). Currently, uFlow contains a library of pillars spanning the height of their channel with a channel aspect ratio of 4:1, width:height, so each transformation is symmetric about the horizontal channel midline. Therefore, we only compute advection in the top half-channel cross-section, and mirror the result about the channel midline.

6.3.1.3 Usage.

Using uFlow through its Graphical User Interface (GUI) is a simple game-like process (see Fig. 7.4). Once the program is started, most interaction is through the mouse, which can select

a pillar diameter and Reynolds number from the left-pane of uFlow (Fig. 7.4(b, d)), and place the pillar within the rendered microchannel itself in a selected location (Fig. 7.4(j)). A view of the flow device's outlet cross-section is then shown in color in the upper-right portion of the window. More complex deformations can be created by placing additional pillars as desired, although several practical limitations - beyond GPU memory to store deformations - should be considered in creating a flow sculpting device: First, transverse mass diffusion will blur deformed shapes, the effects of which become exacerbated with longer channels asssociated with longer sequences of pillars (diffusion is discussed in greater detail in the following section). Second, an increased number of obstacles and a lengthened microchannel will require higher pressure at the inlet to drive flow. High pressure in a microfluidic device can be impractical to implement, and introduce flexure (or damage) to the walls of the device itself, especially if it has been fabricated from softer polymers such as Polydimethylsiloxane (PDMS) (Sollier et al., 2011). Finally, it is generally desirable to minimize flow sculpting's footprint in a lab-on-a-chip device. In addition to the above concerns, a smaller device with fewer flow sculpting parts will allow more room for additional microfluidic components (e.g., polymerization, sensors, or visual interrogation), and have fewer points of failure during fabrication.

The flow streams and pillar properties are all dynamically adjustable in uFlow with real-time feedback to the design. The inlet flow stream widths (Fig. 7.4(a)) can be changed dynamically by clicking and dragging on the small triangles at the lower edge of a stream, a menu to change a stream's color can be opened by right clicking a stream, and new flow streams can be added by double-clicking within the inlet flow pattern design. Because the fluid particles are colored in each rendered frame, adjustments to the inlet flow pattern are immediately reflected in real-time in the deformed flow shape. Similarly, as uFlow stores each microfluidic part's contribution to the net flow deformation in memory, changes made to previously placed parts (e.g., resizing a previous pillar's diameter, or changing its location in the channel) will instantly update the deformed flow shape. The 3D view of the flow sculpting device can be rotated (holding in the right-mouse button and moving the mouse) moved along the channel (holding the left-mouse button and dragging the mosue), or zoomed in/out (using the scroll wheel on the mouse). An image of the deformed fluid is shown after each individual pillar,

allowing the user to see the change in the fluid flow shape as each pillar sculpts it. This can be informative for iterative design, and help educate on chaotic effects of changes early in the sequence. Notably for applications in material fabrication, different regions of sculpted flow can be functionalized with designed properties by doping inlet flow streams with functional materials, such as biotin or superparamagnetic particles (Wu et al., 2015).

The net deformation can be output as a high-resolution portable network graphics (PNG) format image, and the device design itself can be saved in two different forms: uFlow's custom .dev format, which can be loaded into other instances of uFlow to recreate the design; and a CAD .dxf format (see Fig. 7.4(g)). The .dxf format, which can be opened by many CAD programs, is intended to aid in device fabrication, and contains a top-down view of the entire flow sculpting device with pillars correctly spaced to avoid cross-talk.

6.3.2 Diffusion

6.3.2.1 Background.

The effects of mass diffusion have generally been neglected in previous flow sculpting work, as it was assumed that high Peclét numbers ($Pe = \frac{L_c U_c}{D}$, for characteristic length L_c and velocity U_c, and diffusivity coefficient D, showing the relative dominance of the rate of advection vs. diffusion in mass transport) would be employed in their use cases. However, the practical effects of diffusion cannot be neglected even at relatively high Pe, especially given the lower velocity regions near channel boundaries. For flow sculpting in particular, work done by Stoecklein et al. (2014) relied upon mass diffusion to produce several flow transformations. Hence, we aimed to include the effects of diffusion in uFlow's toolkit in a manner that was compatible with the modular assembly of microfluidic components.

In principle, the flow physics of a fluid parcel in inertial flow is determined by the advection-diffusion equation. We track the Lagrangian motion (i.e., pure advection) of the fluid parcel using the previously described advection maps. By using a frame of reference that is moving with the particle, we can consider how diffusion occurs in the direction lateral to the streamline direction. This diffusion in the cross-sectional $y - z$ plane is modeled using Fick's law:

$$\frac{\partial c}{\partial t} = D\nabla^2 c \tag{6.1}$$

We utilize the analytical solution for 2D isotropic diffusion from a point source, computing the concentration $c(t)$ from an initial mass M:

$$c(t) = M\delta(\xi)\delta(\eta)\left(\frac{1}{4\pi Dt}\right)e^{-(\xi^2+\eta^2)/4Dt} \tag{6.2}$$

Where (ξ, η) represent a local coordinate system in the $y-z$ plane, with $(\xi = 0, \eta = 0)$ representing the center of the point source (pixel) being diffused, and the Dirac-delta functions $\delta(\xi)\delta(\eta)$ place the initial concentration of mass M at the center of this pixel. Use of this solution is reasonable since we consider pixel-width streamlines.

uFlow's advection routines operate on non-dimensional velocities, normalized to a characteristic length L_c. We can make equation (6.2) non-dimensional by introducing dimensionless time as $t^* = \frac{tU_c}{L_c}$ and $D = \frac{L_c U_c}{Pe}$, and substituting into the term Dt:

$$Dt = \left(\frac{L_c U_c}{Pe}\right)\left(\frac{t^* L_c}{U_c}\right) \tag{6.3}$$

$$Dt = \frac{L_c^2 t^*}{Pe} \tag{6.4}$$

uFlow's current library of advection maps use $L_c = 1$, simplifying the non-dimensional diffusive length into $Dt = \frac{t^*}{Pe}$. However, the 2D domain over which this diffusion is to act (the microchannel cross section) has a non-uniform time-of-flight $\tau(y, z)$, as fluid moves at different velocities depending on its distance from the boundaries (where velocity goes to zero due to the no-slip boundary condition; see Fig. 6.3(a)) and the reduced cross-sectional area in moving past a pillar. We therefore create a spatially-dependent effective diffusivity $D_\tau(y, z) = D\tau(y, z)$, and utilize the same time-step $t^* = 1$ for the entire domain. The end result is a gaussian blur for each pixel of the form:

$$c(y, z, t^*) = M\delta(\xi)\delta(\eta)\left(\frac{1}{4\pi D_\tau(y,z)t^*}\right)e^{-(\xi^2+\eta^2)/4D_\tau(y,z)t^*} \tag{6.5}$$

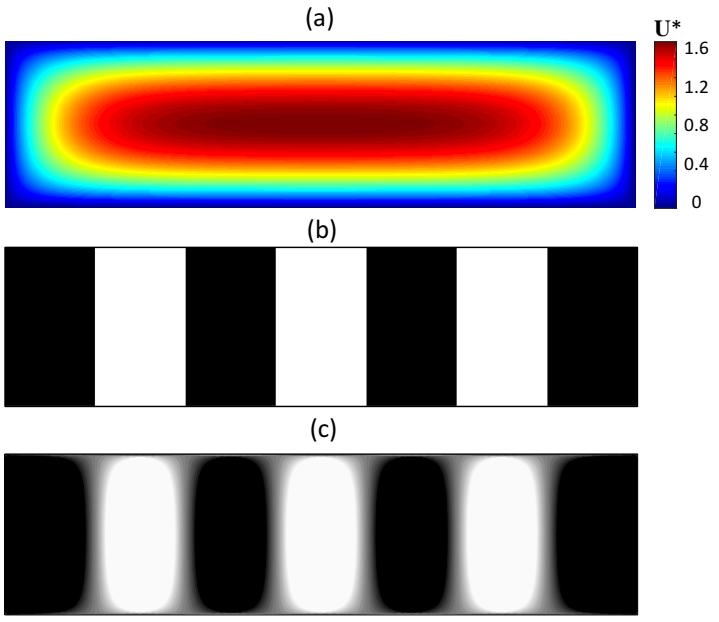

Figure 6.3 Diffusion in uFlow. (a) A parabolic velocity profile $\mathbf{U}^*$ for a rectangular channel is calculated using Spiga and Morino's analytical solution (Spiga and Morino (1994)), which is used with a fixed channel length to calculate a non-dimensional fluid flow time-of-flight τ. The scalar field $\tau(x, y)$ is then used to create a spatially dependent diffusivity coefficient $D_\tau(x, y)$, which is used with a single time step t^* in the fundamental solution to Fick's law to blur the fluid flow. This is demonstrated for a non-advected inlet flow pattern (b), which is diffused for two equivalent pillar lengths (c) with $Pe = 10^5$.

and in non-dimensional form, using a spatially dependent $Pe(y, z) = \frac{\tau(y,z)}{D}$:

$$c(y, z, t^*) = M\delta(\xi)\delta(\eta)\left(\frac{Pe(y, z)}{4\pi L_c^2 t^*}\right) e^{\frac{-Pe(y,z)(\xi^2+\eta^2)}{4L_c^2 t^*}} \tag{6.6}$$

Although a unique time-of-flight as a function of cross-section position and pillar component could be utilized in this scheme, we approximate D_τ by assuming that the bulk of a fluid element's time-of-flight will be spent in the open channel, and calculate $\tau(y, z)$ from an analytical solution for the velocity field considering laminar flow in a rectangular duct (Spiga and Morino, 1994).

This diffusive blur is applied after each pillar deforms the cross-sectional flow shape, with the diffused result becoming the input for the next pillar.

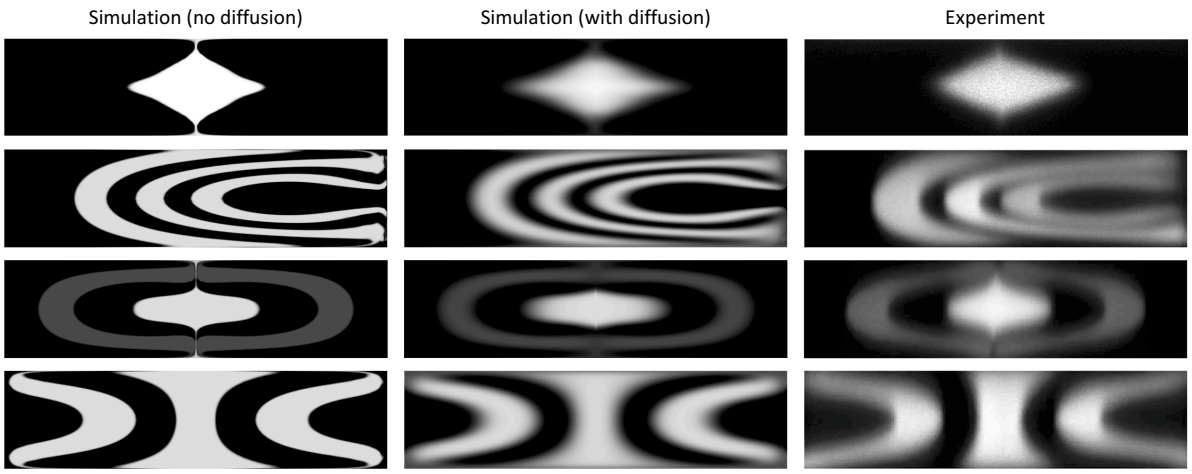

Figure 6.4 Examples of uFlow's simulated diffusion compared to confocal images from experiments. Shown are simulations with no effective diffusion ($Pe = 10^7$, left), moderate diffusion ($Re = 20, Pe = 10^5$, middle), and confocal images from flow sculpting devices with matching Pe ($Pe = 10^5$, right).

6.3.2.2 Implementation.

Diffusion is implemented in uFlow by considering the diffused mass M as the RGB color profile at each pixel, and embedding the previously defined gaussian function within the advection-coloring fragment shader. First, a time-of-flight is calculated based on a series sum from Spiga and Morino (Spiga and Morino, 1994), and used with the known inter-pillar spacing (Amini et al., 2013) ($10L_c$) to find a non-dimensional scalar field $\tau(x, y)$. This field is then used with equation (6.6) to compute the color profile at a point. To diffuse a single point, a fragment shader samples an 8×8 neighborhood of pixels, and calculates how each neighbor will affect the single point under consideration based on equation (6.6). The point is then colored by summing each neighbor's diffused contribution and its own color, advected from the inlet streams. This sum is normalized by the number of points sampled, allowing diffusion to work near the boundaries where the 8×8 blur operation only samples from the smaller number of points within the fluid domain.

Despite simplifying the physics, this diffusion model replicates experimental behavior well. Fig. 6.3 shows how the time-of-flight field $\tau(y, z)$, based on the velocity field for laminar duct

flow, diffuses a co-flow of streams in a straight channel in the expected "hourglass" shape (Ismagilov et al., 2000). In Fig 6.4, uFlow's diffusion model is compared to confocal images taken from flow sculpting devices with $Pe = 10^5$ (fabricated using the techniques outlined in Ref. (Amini et al., 2013)), alongside non-diffusive simulations ($Pe = 10^7$) to demonstrate the difference. While showing qualitative accuracy, there is further scope for improvement of incorporating the effect of diffusion. Future work will include per-pillar (or generic microfluidic part) time-of-flight information, as advection maps use only two of the four available channels in an RGBA texture to store a (dy, dz) vector field, leaving room for time-of-flight data.

6.3.2.3 Usage.

The diffusion operation automatically updates with each frame render, so adjustments to the Peclét number in the GUI (see Fig. 7.4(e)) will change the visualized diffusive blur in real-time. If a non-diffusive image is desired (advection only), the device's Peclét number can be increased to the point where diffusive effects are no longer present. Practically, we see this for $Pe \geq 10^7$.

6.3.3 Microparticle Fabrication Visualization

6.3.3.1 Background.

Flow sculpting has been used with stop flow lithography (Paulsen et al., 2015) and transient liquid molding (Wu et al., 2015) to fabricate shaped millimeter- and micrometer-scale particles. The basic idea is to shape an inlet stream containing a crosslinkable polymer precursor, and polymerize the shaped stream using a UV light source with a defined mask shape once the flow is stopped (see Fig. 6.5(a)). This creates a 3D polymer microparticle with a flow-sculpted shape in one orientation, and a mask-defined shape in an orientation normal to that. The mask - generally manufactured as a standard photolithography mask - has high precision for smaller features, allowing for essentially any 2D design to form the shape of the microparticle in the UV-light direction. The cross-sectional flow shape, on the other hand, is limited by

what is possible through inertial flow sculpting[5]. However, both shapes will be affected by an extended diffusive process during the stopped-flow portion of fabrication ($\approx$ 1s), which should be anticipated during the design of both shapes and minimized or accounted for in software as much as possible for a more faithful recreation of the target 3D geometry (Wu et al., 2015).

6.3.3.2 Implementation.

uFlow uses an indirect approach for rendering the expected particle shape, without computing the actual particle geometry as an intermediate step. The rendering algorithm uses a *signed distance function* (SDF) representation of the particle in combination with a *ray marching* scheme (Hart, 1996), discussed in detail in Appendix C. The result of this algorithm is a very fast renderer that produces an image of the predicted particle shape without computing the particle geometry as an intermediate step.

6.3.3.3 Usage.

To visualize the fabricated microparticle, uFlow needs a sculpted flow shape and a shape for the orthogonal optical mask that selectively exposes the shaped flow stream to UV light. By default, uFlow selects the central fluid stream as the polymer pre-cursor, while sheath flow surrounding the central stream is the same monomer but without the photoinitiator (making viscosity uniform throughout the channel). Different streams can be chosen to be polymerized by altering a sequence of booleans in the `polymerized_streams` variable within the main script, `uflow.py`. The mask shape is created by the user as a 256×256 black-and-white image "mask.png", and placed in uFlow's top-level directory (a rectangular UV-mask is visible in the floor of the visualized microchannel in Fig. 7.4(h)).

While purely advective flow would retain all photoinitiator within the sculpted polymer precusor shape, diffusion - naturally present to some degree in most flows - will bring some

[5]Although uFlow itself is capable of generating user-designed flow shapes through manual design, an automated optimization tool called "FlowSculpt" has been developed for finding pillar sequences for desired flow shapes (Stoecklein et al., 2017a). FlowSculpt uses the genetic algorithm to identify a solution to the inverse problem with minimized error, and can export designs as .dev to uFlow itself for flow visualization. Additional methods for fast automated design such as deep learning (Lore et al., 2015; Stoecklein et al., 2017b), are also being explored.

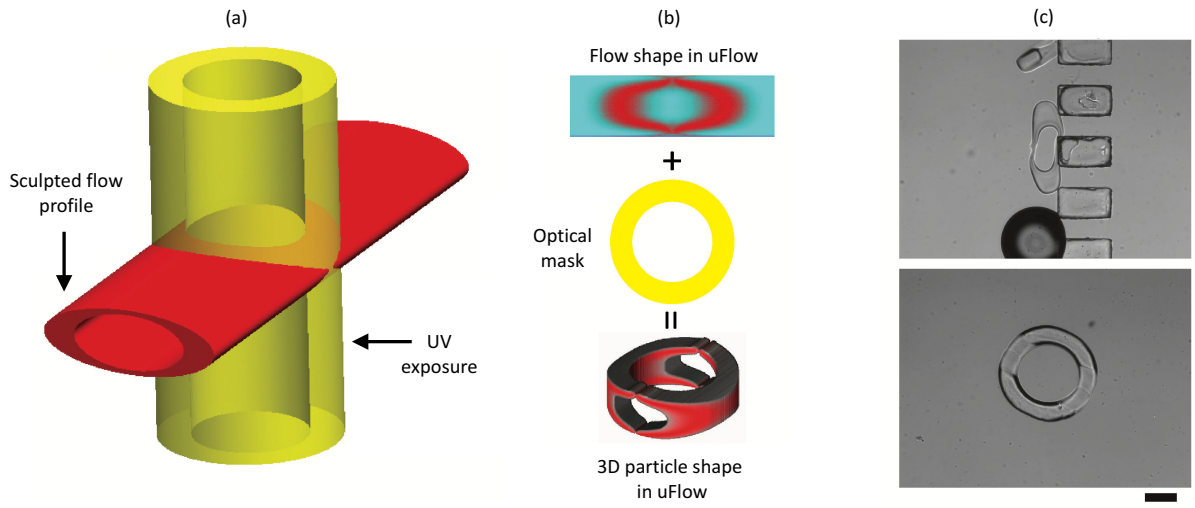

Figure 6.5 Microparticle fabrication via Transient Liquid Molding (TLM) and visualization in uFlow. (a) Flow sculpting can be used with TLM to fabricate tailored 3D microparticles. A pillar sequence shapes a UV-crosslinked polymer precursor into some user-designed shape in the flow direction (red), while a mask is placed over the channel in the post-pillar sequence region (yellow). When the flow stops, a UV light shines on the mask, polymerizing the sculpted flow to have the mask's shape in the UV light orientation, and the flow sculpted shape in the flow direction. (b) 3D particle estimation of a porous particle in uFlow, with a flow shape, optical mask (defined in mask.png), and the resulting 3D particle visualization. (c) TLM fabrication of the porous particle designed in (b), shown with its flow direction face (top image) and mask-defined face (bottom). uFlow's prediction and the experiment show good agreement. The scale bar is $100\mu m$.

amount of photoinitator into neighboring regions. To take the effects of diffusion on the fabricated particle into account, uFlow has a user-adjustable "polymerization threshold" value (see Fig. 7.4(f)), which determines the UV-curable volume fraction required to achieve solidification. This threshold is also affected by the energy in the UV light, which will depend on intensity and exposure time (this is discussed in more detail in Ref. (Wu et al., 2015)). By adjusting both the Peclét number and polymerization threshold, one changes the degree to which the fluid shape expands and neighboring fluids diffuse into the final polymerized particle shape.

6.4 Advanced compression routines

The utility of uFlow is predicated on quick access to advection maps derived from full-scale solutions to the Navier-Stokes equations. However, the intent of its users, whether to create a particular flow shape, direct fluid to a particular region of the microchannel cross-section, or some other discipline-specific goal, cannot be fully anticipated when preparing a useful set of advection maps. We therefore seek to provide as large of a design space as possible.

uFlow allows access to a continuous set (i.e., infinite set) of pillar diameters and locations by compressing a discrete dataset of pre-computed advection maps using Principal Component Analysis (PCA) (John Aldo Lee, 2007). PCA can compress a set of n high-dimensional observations $\{O_1, O_2, \ldots, O_n\}$, each of dimension m, by finding a new *basis* of "principal components" $\{PC_1, PC_2, \ldots, PC_n\}$ in the high-dimensional space which maximally explains the variance among all observations in the dataset. Once the high-dimensional basis has been constructed, each of the dataset's original observations can be recreated as $\hat{O}_i$ from a linear combination of the new basis vectors PC_j (which are also of dimension m):

$$\hat{O}_i = \sum_{j=1}^{n} \alpha_{i,j} PC_j \tag{6.7}$$

where the coefficient $\alpha_{i,j}$ is an observation-specific low-dimensional mapping, indicating how the observation O_i aligns with the direction of the principal component PC_j. This mapping is generated by simply projecting an observation onto the PCA basis, thus embedding the high-dimensional observation in the low-dimensional PCA space.

PCA orders the principal components by the variance explained in the original dataset: PC_1 explains the most variance of all principal components, PC_2 the second-most, and so on. Therefore, a small subset of all n available principal components can be used to reconstruct any of the original data to a nominal degree of accuracy. This can be determined by calculating the cumulative proportion of variance explained by the chosen subset of principal components, and comparing it the total variance in the dataset. This produces an effective cumulative "energy content", E, which, for a principal component's variance-explained λ_j is found for a reduced number of n_R principal components by:

$$E(n_R) = \frac{1}{E_T} \sum_{j=1}^{n_R} \lambda_j \tag{6.8}$$

where E_T, the total cumulative energy content of the dataset, is found by:

$$E_T = \sum_{j=1}^{n} \lambda_j \tag{6.9}$$

A dataset is well represented using the basis formed by n_R principal components as $E(n_R) \rightarrow 1$. A tradeoff can then be made between complete reconstruction of the original data (which requires all n principal components, in which case you may as well use the original data), and using a subset of the first $n_R \ll n$ principal components to reconstruct data to an acceptable degree. This is a form of data compression, as one only needs to store n_R principal components and the original data's projected mapping in the low-dimensional space (m arrays of n_R floating point values).

For uFlow, the high-dimensional dataset being compressed are a collection of 3,224 advection maps, calculated for $Re = \{10, 20, 30, 40\}$ from 31 different normalized pillar diameters of $D/w = [0.2 : 0.02 : 0.8]$ and 26 normalized pillar locations $y/w = [0.0 : 0.02 : 0.5]$ for channel width w (locations from $-0.5 < y/w < 0$ are mirrored on the GPU, and do not require storage). These maps were created using an in-house streamtracing code (there are many freely available visualization software packages that contain effective streamtracing tools, such as Paraview), tracing through 3D velocity fields created on HPC resources using an experimentally validated Finite Element Method framework (Diaz-Montes et al., 2014). Such a large dataset may not be easily stored and transferred across computers, or easily recalled for computation on the GPU, but its fine sampling is well-suited to PCA compression and interpolation. We found that splitting the dataset's dy, dz displacement data into separate datasets made for more efficient PCA compression, likely due to differences in variance between vertical and horizontal velocities being difficult to explain with the same linear basis. In using a reduced number of $n_R = 300$ principal components (from a possible $n = 3,224$), we are able to reconstruct any of the original observations with a cumulative energy content of $E(n_R) = 99.89\%$. Therefore, uFlow need only store the first 300 principal components (as 2D RGBA textures) and each ob-

servation's low-dimensional mapping to effectively recreate any of the original 3,224 advection maps, with data compressed to $\approx 9.3\%$ of its original size.

Beyond storing the original data in a compressed format, PCA allows for interpolation within the low-dimensional mapping in order to create advection maps not contained in the original dataset (given a smoothly varying low-dimensional embedding). This is achieved by linearly interpolating a desired pillar's low-dimensional mapping from its nearest neighbors within the original dataset, and using the new mapping with equation (6.7) and the dataset's principal components to construct a new advection map.

We tested PCA interpolation by reconstructing non-original advection maps across the $Re = 20$ domain, and comparing them to natively-streamtraced advection maps using the L_2-norm: $error = \|\hat{O} - O\|_2 / \|O\|_2$. We found that over 98% of the reconstructed data had less than 5% error, with a mean reconstruction error of 1.76%, and a maximum error of 10% (See Supplemental Information). With such low error overall, we are confident that PCA interpolation can be used in uFlow to access a practically infinite set of pillar configurations, provided the number of pillars in a single sequence stays within the limits of practical fabrication (i.e., fewer than 30 pillars) to avoid significant error propagation. PCA interpolation in uFlow is currently limited to pillar geometry and location only, as the sampling of Reynolds numbers is likely too coarse for accurate interpolation.

The PCA library is implemented in uFlow as the `OGLPCAFactory` class, which reads from a stored set of 300 principal components and the original dataset's dy, dz low-dimensional mappings (stored as text files). The linear reconstruction in equation (6.7) is encoded into a set of fragment shaders to reconstruct a requested advection map on-the-fly on the GPU, with no discernible latency compared to recalling a stored advection map.

6.5 Conclusions

uFlow gives users with modest computing hardware access to a practically infinite set of high performance computational fluid flow simulations, capable of real-time design and exploration, and complete with estimations of diffusion effects and visualization of fabricated microparticles using various masks. uFlow also enables users to incorporate new microfluidic components to

sculpt flow of their own design, and shows how they can be efficiently stored for quick access on the GPU.

The primary modes of uFlow's functionality - flow sculpting and visualization - are accessible through the GUI, while the new advection map storage is found in the publicly available uFlow code. Our goal is to enable the broader microfluidics community - be they researching diagnostic technology, advanced manufacturing, or other new applications of flow sculpting - to easily integrate complex flow physics into their research and technology designs.

In uFlow, a path is provided for users to conceive of microfluidic components to sculpt fluid, convert their pre-computed 3D velocity data to high-speed advection maps, and explore each component's impact on the flow sculpting design space. With the addition of diffusion modeling and an expansive, easily extensible library of pillar configurations, many previous major shortcomings have been ameliorated. By including microparticle fabrication visualization, uFlow is now reaching into a set of post-flow-sculpted operations and analyses which can have immediate impact in microparticle design and optimization. Additional post-processing of sculpted flow, such as concentration gradient calculations, or voxelized particle geometries, are logical next steps for this new feature space. Future directions for uFlow's physical models and routines include asymmetric, full-channel flow sculpting by using pillars of variable height, herringbone structures, or steps in the channel; integration with automated design software like FlowSculpt (Stoecklein et al., 2017a); tracing and predicting the paths of finite size particles; and additional tools to streamline the process of implementing new microfluidic components in the uFlow framework.

CHAPTER 7. FLOWSCULPT: SOFTWARE FOR EFFICIENT DESIGN OF INERTIAL FLOW SCULPTING DEVICES

This chapter includes a manuscript titled "FlowSculpt: software for efficient design of inertial flow sculpting devices", in preparation for submission to *Lab on a chip* journal, authored by Daniel Stoecklein, Michael Davies, Joseph de Rutte, Chueh-Yu Wu, Jesse Trujillo, Dino Di Carlo, and Baskar Ganapathysubramanian

Abstract

Flow sculpting is a powerful method for passive flow control, capable of engineering inertially flowing microfluidic streams using a sequence of bluff-body structures. A variety of cross-sectional flow shapes can be designed through this method, useful for the manufacturing of tailored microfibers and particles, or enhancing bioengineering and chemistry applications. However, the inverse problem - designing a device that produces a target fluid flow shape - remains challenging due to the complex, diverse, and enormous design space. Current approaches to design such microfluidic devices are still in nascent stages of development. State-of-the-art computational utilities have considerable runtime (on the order of hours), which impedes their widespread adoption. Moreover, the design space is generally limited to top-bottom symmetric flow shapes due to the bluff-body structures available in current libraries (pillars) spanning the height of the channel. In this work, we present an extremely fast simulation method for flow sculpting, which integrates into our design framework to provide a 34× reduction in runtime. We also introduce symmetry-breaking flow deformations with the use of partial-height pillars. The framework is deployed freely as a cross-platform application, "FlowSculpt". We detail its implementation and usage, and discuss the addition of enhanced search operations, which

enable users to more easily design flow shapes that replicate their input drawings. With FlowSculpt, the microfluidics community can now quickly design flow shaping microfluidic devices on modest hardware, and integrate these complex physics into their research toolkit.

7.1 Introduction

Shaping inertial microfluidic flow using sequences of pillar structures has recently gained traction in the microfluidic community as a method for passive fluid flow control. As laminar fluid flow with finite inertia ($1 < Re < 100$, with the Reynolds number $Re = \rho U D_h/\mu$, for fluid density ρ, average velocity U, viscosity μ, and channel hydraulic diameter D_h) moves past a pillar in a microchannel, induced net secondary flow alters the cross-sectional fluid shape in a time-invariant deformation (Amini et al., 2013). If pillars are placed sufficiently far apart in a sequence to prevent cross-talk, their deformations act independently on the fluid flow shape, allowing for the entire sequence to be rapidly simulated as a nested set of pre-computed operations (Amini et al., 2013). This technique of intelligently placing obstructions in a microchannel to produce a desired deformation in inertially flowing fluid is called "flow sculpting".

Flow sculpting enables new levels of passive flow control for a wide range of applications at the microscale. Tailored fluid flow can be used in lab-on-a-chip environments to redirect flow in a microchannel (Sollier et al., 2015), produce fibers with 3D cross-sections (Nunes et al., 2014) and particles at millimeter (Paulsen et al., 2015) and micrometer (Wu et al., 2015) scales, for example. A useful aspect of flow sculpting is the ease with which devices leveraging these physics can be simulated, due in large part to each pillar in a sequence acting as an independent building block toward an overall net deformation. This allows for a library of pre-computed flow deformations, known as *advection maps*, to be rapidly concatenated for real-time device simulation, as demonstrated in the freely available software "uFlow" (Stoecklein et al., 2014). However, the inverse problem in flow sculpting - determining the design of a flow sculpting device that produces a target fluid flow shape - is challenging for a number of reasons, among which is the complex, diverse, and enormous combinatorial design space. Consider a simple library of 32 pre-computed deformations as used in uFlow by Stoecklein et al. (2014). These

32 deformations were simulated from various pillar geometries, consisting of four diameters of $d/w = \{0.375, 0.5, 0.625, 0.75\}$ for a channel of width w, and eight locations spanning the width of the channel, at $Re = 20$ and channel aspect ratio $h/w = 0.25$. Complex fluid flow structures were demonstrated for sequences of 10 pillars in length (Stoecklein et al., 2014), of which there are of $32^{10} \approx 10^{15}$ possible pillar sequence configurations, many of which lead to similar flow shapes. Combined with the choice of how the fluid flow shape is configured at the microchannel inlet (called the *inlet flow pattern*, seen in Fig. 7.1(a)), manually searching through flow sculpting's large design space for a particular flow deformation quickly becomes impractical. In fact, an analogous problem of choosing some set of non-linear state transformations to match a desired net transformation[1] has been shown to be NP-hard (nondeterministic polynomial time hard) (Even and Goldreich, 1981). Hence, an automated - and possibly heuristic optimization - routine is required to make design accessible to the broader microfluidics community.

Heuristic based searching methods have proven effective in solving inverse problems with discrete, non-differentiable, and multimodal design spaces (Giles and Pierce, 2000; Mohammadi and Pironneau, 2004). In particular, the Genetic Algorithm (GA) has been shown to successfully optimize system geometry and operating conditions for fluid flow problems (Mott et al., 2009; Cortes-Quiroz et al., 2009; Müller et al., 2004; Ivorra et al., 2006), including flow sculpting (Stoecklein et al., 2016, 2017a). The GA is based on the mechanisms of natural evolution, where a population of candidate solutions strive toward an objective, while randomly mutating and exchanging design parameters according to stochastic selection and genomic modification processes, all attempting to maximize the population's *fitness* (Goldberg, 1989). This technique was first employed for flow sculpting by Stoecklein et al. (2016) to optimize pillar sequences for known fluid flow transformations, and search for arbitrary hand-drawn flow shapes. However, there were considerable limitations: the application utilized commercial software (MATLAB) for deployment, users were required to specify the inlet design, the objective (fitness) function was a simple pixel-to-pixel image comparison, and runtime was less than ideal at ≈24 h

[1]The inverse problem in flow sculpting as defined in this work is in fact more difficult than that of Ref (Even and Goldreich, 1981), as the target here is a scalar-valued *result* of an unknown (and perhaps impossible) transformation, rather than just the transformation itself. That is, one does not necessarily know what state the flow must begin in, nor the route to their desired shape - they only know how they want the flow to appear at the end of a pillar sequence.

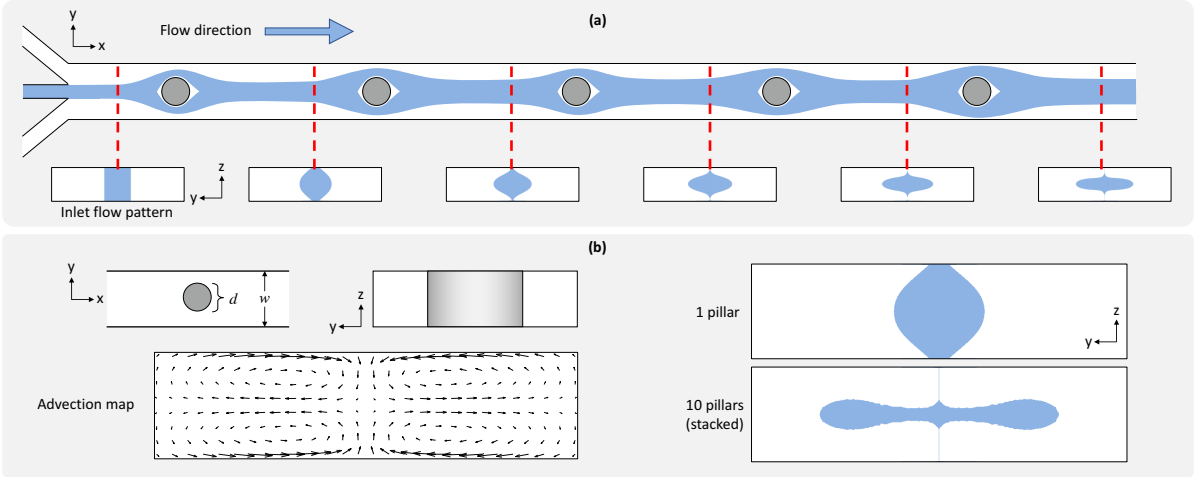

Figure 7.1 Overview of flow sculpting. (a) A sequence of obstacles (pillars) deforms an *inlet flow pattern* (sculpted flow is colored blue), with each obstacle contributing its own flow deformation to the overall net deformation at the outlet. In this illustration, the inlet flow pattern is a central stream of a width $w/5$ for a microchannel width w. (b) To accelerate flow sculpting device simulations, *advection maps* are created for a library of pillar geometries (shown is the advection map and flow deformation for a pillar of diameter $d/w = 0.5$ and lateral location $y/w = 0.0$), making whole-device simulation a matter of sampling stored 2D vector fields. Note the top-bottom symmetry in the advection map and flow deformations, due to the pillar spanning the height of the channel. For this figure, the channel aspect ratio is $h/w = 0.25$ and $Re = 20$.

per search. A 2nd generation GA framework for flow sculpting was created (Stoecklein et al., 2017a), improving on most of these shortcomings with a freely available software "FlowSculpt". The FlowSculpt software included a customized GA framework that allowed for GA-designed inlet flow patterns, a Graphical User Interface (GUI), and a reduced runtime of ≈2 h. Yet, there remains considerable scope for improvement in the capability of the FlowSculpt platform. Namely, (1) the design space available in FlowSculpt is still limited to the symmetric design present in flow sculpting's debut, due to the continued use of pillars spanning the height of the microchannel; (2) design optimization runtime is quite lengthy at 2 h, making rapid design iterations tedious and less approachable; and (3) the use of a single *per-pixel* objective function greatly limits the design space, and often necessitates repeated design iterations.

The symmetry of fluid flow shapes is a natural result of the deformation induced by pillars which span the height of the channel, since there is mirror symmetry across the horizontal midline of the channel. Paulsen and Chung proposed a novel method of obviating this limitation for the fabrication of non-spherical particles (Paulsen and Chung, 2016): mismatched densities of the sculpted fluid and sheath flow would be given time for gravitationally-driven settling to break symmetry before polymerizing the fluid into particles. This method comes with several clear drawbacks. The relegation of asymmetric design to gravity (and time) removes a great deal of control over the final outcome of the shape, and appends a relatively complex simulation to the comparatively trivial advection map computation in predicting the asymmetric shape being produced. It also confines asymmetric design to stopped fluid flow, which, while useful for fabricating particles, prevents application to continuous on-chip processes in fabrication or analysis. In addition, the use of mismatched fluid stream densities places constraints on the materials used for design, which may not be optional for microfluidics researchers.

In this work, we seek to ameliorate all previously described shortcomings to the FlowSculpt software. To break mid-plane symmetry, we use partial-height pillars. By using partial-height pillars, we achieve deformations with both non-mirror symmetric vertical and lateral displacement of fluid, allowing for a complex set of new flow transformations. We couple this asymmetric design component to the FlowSculpt software with a new library of advection maps. We showcase fully asymmetric flow sculpting by using FlowSculpt to design letters of

the Roman alphabet. We also present a 3rd generation of the FlowSculpt framework, focusing on new objective functions, significant runtime improvements, and a rebuilt Graphical User Interface (GUI). We discuss a new fitness evaluation method which drastically increases the speed of the forward model, enabling researchers with extremely modest computing power to use FlowSculpt. In addition, we discuss other flexibility improvements such as the ability to export computer aided design (CAD) models for device fabrication, and modifications which allow this GA approach to be even more effective in searching the vast pillar design space.

7.2 FlowSculpt Overview and Methods

Solving the inverse problem in flow sculpting usually requires HPC resources that preclude widespread adoption by microfluidics researchers with modest computing power. Instead, we take advantage of the previously described computational shortcut using advection maps to embed a fast *forward model* within an evolutionary algorithm (the GA) to make automated flow sculpting design easily accessible via the FlowSculpt software. Binary executables for FlowSculpt on Windows, MacOS, and Linux, as well as its source code, are freely available at www.flowsculpt.org. In the following sections, we describe FlowSculpt's optimization process, forward model implementation, and GUI usage.

7.2.1 Genetic Algorithm Optimization

FlowSculpt uses Genetic Algorithm (GA) optimization to design flow sculpting devices which produce flow shapes closely matching a user-provided *target flow shape*. We quickly outline the GA here. The GA is initialized by randomly generating a population of candidate solutions known as *chromosomes*, each of which contains a standalone design for the problem of interest. The population of chromosomes is then evaluated using a "fitness function" (objective function), which returns a scalar value representing the fitness of each chromosome, with higher fitness being more desirable. Fitness is used to determine the next generation of chromosomes by a stochastic *selection* process, which chooses parents to exchange random components of their design variables via *crossover* to form offspring for the next generation. Higher fitness is rewarded with a greater chance of selection, thus propagating genetic material associated with

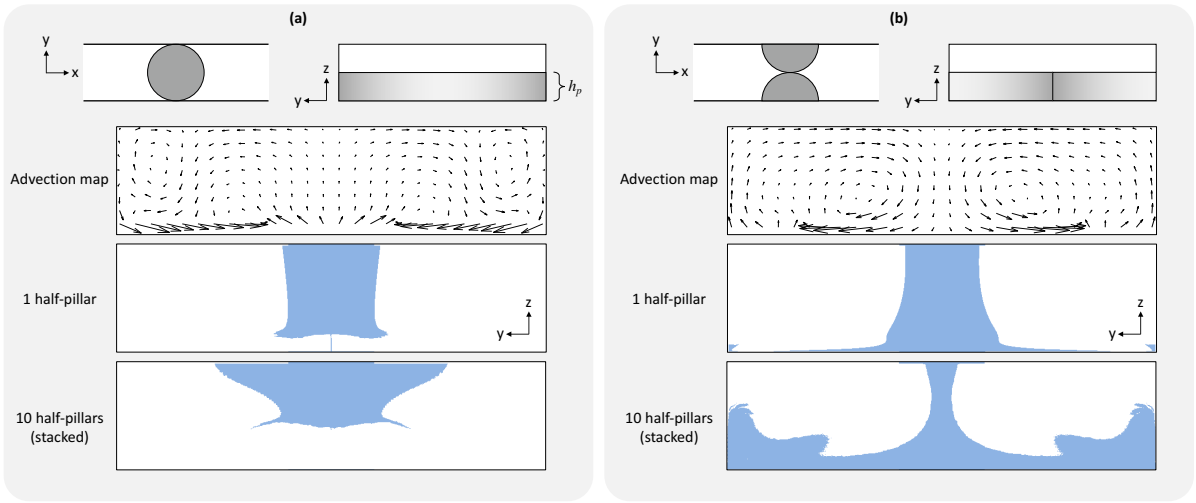

Figure 7.2 Examples of asymmetric fluid flow transformations from *half-pillar* obstacles. (a) A full-width half-pillar placed at $y/w = 0.0$ will drive fluid in the center of the channel upward, resulting in flow being displaced to the upper-half of the microchannel. (b) Placing full-width half-pillars at $y/w = 0.5$ (with periodic pillar geometry) results in a flow transformation complementing that of $y/w = 0.0$, with fluid being forced into the lower-half of the microchannel.

successful design throughout the population; conversely, chromosomes with poor fitness will tend to have their designs washed out of the population. The best-performing chromosomes will also be chosen as "elites", which are passed on untouched to the next generation, thus preserving a running optimum solution. Another evolutionary mechanism that occurs after fitness evaluation is *mutation*, which randomly selects chromosomes to have their genetic material perturbed by some random amount, and placed in the next generation. Mutation introduces genetic variation beyond what is allowed through selection/crossover operators, since crossover only exchanges genetic material already present in the population's gene pool. The population will then step forward, repeating the processes of evaluation, selection, crossover, and mutation to create a new generation of chromosomes. This repeats until some termination criteria are met. After the GA terminates, the best-performing chromosome is output as the solution. As the GA is a stochastic process, finding the global optimum is not guaranteed; therefore, it is important to re-run the GA some number of times in order to attempt statistical exploration of the space.

There are two essential components of the GA that must be implemented to solve an inverse problem: a chromosome containing problem-specific design parameters as genetic material, and a fitness function to evaluate each chromosome during the GA's evolutionary step. We utilize the same chromosome design used previously in Ref. (Stoecklein et al., 2017a), which encodes a flow sculpting device design as a binary-valued inlet flow pattern, and a fixed-length, real-valued pillar sequence with separated diameter and location values for each pillar. To account for half-pillars in the chromosome, we add an extra real-valued component to each pillar's design to select the pillar height[2]. For a fitness function, we emulate the forward model for predicted flow deformation used by Stoecklein et a. (Stoecklein et al., 2017a), which used transition matrices derived from pre-computed advection maps to transform a discretized set of binary fluid states within the cross-section of the channel. However, we significantly improve on the speed of this model by using one-to-one *index maps* (synonymous with lookup tables), which are discussed later in this paper. To compute fitness, we continue the use of an image-comparison correlation metric, and implement additional fitness functions by including image post-processing functionality to FlowSculpt, also discussed later.

7.2.2 Forward Model and Fitness Function

The forward model used in FlowSculpt is based on pre-computed advection maps, which are then converted to index maps for fast flow deformations on the Central Processing Unit (CPU). Once a chromosome's flow deformation has been computed using the forward model, a comparison is made between this flow shape and the target flow shape, yielding the chromosome's fitness value. Below, we describe how advection maps are created and used within the forward model, and discuss FlowSculpt's various fitness functions.

7.2.2.1 Advection Map Generation.

Advection maps are 2D vector fields describing the net lateral displacement of fluid flowing through a 3D microchannel domain. Previously, FlowSculpt used advection maps created

[2]Because the top-half of the channel is completely open in half-pillar geometries, larger diameters of $d/w = \{0.875, 1.0\}$ can be utilized without drastically increasing pressure drop. We therefore constrain the chromosomes to prevent full-height pillars from attaining these larger, infeasible diameters.

from simulated 3D domains containing pillars spanning the height of the microchannel, with normalized diameter d/w and location y/w for microchannel width w. In theory, any structure inducing secondary flow which conforms to the constraints of flow sculpting can be used. These constraints are necessary for accurate predictions of sculpted flow, and enumerated here:

1. Each structure-induced flow deformation must be completed before the fluid begins to interact with neighboring pillars (i.e., no cross-talk).

2. The structure-induced flow deformation must not vary in time (e.g., no periodic motion or shed vortices).

3. For a set of obstacles to be used in a flow sculpting sequence, each neighboring obstacle's domain must match flow conditions (Re) and channel geometry (h/w).

These rules for flow sculpting allow a broad set of usable flow physics and geometry. For this work, we chose to maintain the same microchannel dimensions $h/w = 0.25$ (for channel height h and width w) and matching flow physics of $Re = 20$ (with Re defined by the hydraulic diameter of the channel, D_h). To generate advection maps, we solved the 3D Navier-Stokes equations for a set of pillar geometries using our experimentally validated in-house Finite Element Method (FEM) software (Stoecklein et al., 2014, 2016). Each simulated 3D domain has $\geq 2d/w$ of empty channel spacing upstream of the pillar, and $\geq 8d/w$ spacing downstream, which allows flow deformation to saturate before exiting the domain (Amini et al., 2013). The resulting 3D velocity fields are then streamtraced with neutrally buoyant, infinitesimal particles, which gives the 2D fluid displacements for each advection map.

Until now, FlowSculpt has used flow-deforming pillars of a height $h_p/h = 1.0$, thus confining the design space of sculpted cross-sectional flow shapes to have symmetry across the horizontal mid-plane (See Fig. 7.1(b)). In this work, we break mid-plane symmetry by adding partial-height pillars for which $h_p/h = 0.5$ to FlowSculpt's library of flow deformations. We therefore simulated two different classes of pillar geometry: *full-pillar* ($h_p/h = 1.0$) and *half-pillar* ($h_p/h = 0.5$). For each class of pillar, we simulated 3D flow fields for pillar locations spanning $y/w = [-0.5 : 0.125 : 0.5]$, with $y/w = 0.0$ being the center of the microchannel.

Pillar diameters for the full-pillars were $d/w = \{0.375, 0.5, 0.625, 0.75\}$, while the half-pillars used the same set, appended with larger diameters $d/w = \{0.875, 1.0\}$. These larger-diameter pillars force a considerable amount of fluid into the top-half of the channel as it flows past the pillar, creating a set of powerful asymmetric flow transformations yet-unseen in flow sculpting's toolkit. For example, whereas previous flow transformations could shift fluid laterally to one side of a microchannel, full-width half-pillars can direct fluid downward toward the channel floor (Fig. 7.2(a)) or upward toward the top of the channel (Fig 7.2(b)). This kind of flow transformation could be useful for enhancing the efficiency of sensors embedded in the microchannel (Bailey et al., 2015; Selmi et al., 2015), for example. We also checked the deformation saturation length for half-pillar simulations, and found that they followed the same guidelines of a spacing $\geq 8d/w$ downstream as with full-pillars.

One caveat to the use of half-pillars is increased FlowSculpt runtime, which comes from two different mechanisms: flow shape image size and GA chromosome complexity. In using neutrally buoyant co-flows with pillars spanning the height of the microchannel ($h_p/h = 1.0$), flow shapes are completely represented by simulating only one half of the channel cross-section (top or bottom), from which the entire flow shape can be made via mirroring across a symmetry plane running through the microchannel. Thus, numerical optimization need only operate in the originally simulated half-channel with $h_p/h = 1.0$. However, breaking mid-plane symmetry - e.g., by use of half-pillars - requires *full-channel* simulation, effectively doubling the computational workload for both flow deformation simulations and subsequent processing within a fitness function. Beyond this, we have increased the complexity of the GA's chromosome by adding a new parameter for pillar height, h_p/h, thereby introducing half-pillar counterparts to the existing 32 pillar configurations, and adding another 16 configurations for $d/w = \{0.875, 1.000\}$. This nearly triples the single-pillar design space from 32 to 80 configurations, which makes flow sculpting more capable in its design, but more difficult as a combinatorial problem. With this in mind, a faster fitness function is quite helpful, as the use of larger GA population sizes and additional repetitions - which can aid in dealing with a larger design space - become computationally feasible.

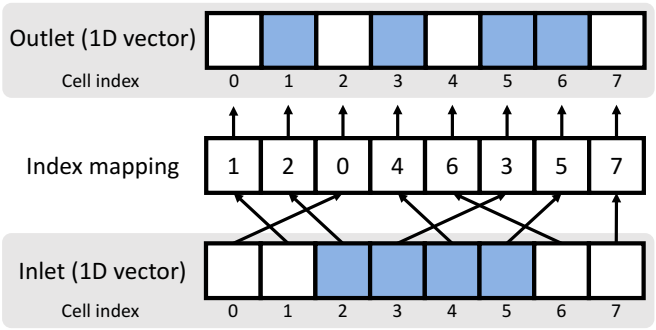

Figure 7.3 Diagram depicting the operation performed by an index mapping on a 1D vector of eight binary fluid states (colored blue to show tracked fluid) from inlet to outlet.

7.2.2.2 Fast Index Maps.

Computing the flow deformation caused by each flow sculpting component starts with a costly CFD simulation, which is reduced to a 2D advection map, and subsequently used to deform some inlet flow pattern. Stoecklein et, al. (Stoecklein et al., 2016) presented a method of further reducing the complexity of flow shape simulation by discretizing the microchannel cross-section into a set of cells representing fluid elements as binary states (either 1 for sculpted flow, or 0 for co-flow). Advection maps are then converted into sparse *transition matrices*, which describe how fluid will transition from one discretized cell to another (Stoecklein et al., 2016, 2017a). Simulating a flow sculpting device then becomes a matter of linear algebra, with matrix-matrix multiplication producing a *net* transition matrix, which can be multiplied by a 1D vector representing the inlet flow pattern, producing a 1D vector representing the outlet fluid states. The vector can be reshaped into a 2D matrix, and processed as an image, with each fluid cell being a pixel.

This approach made for easy portability to CPU or GPU architecture, and fast implementation in optimization routines. However, a tradeoff is made between speed and accuracy based on the number of fluid cells in the discretized cross-section. For a coarse discretization of fewer - but larger - cells, the number of linear algebra operations are minimized, thus making for a faster forward model. However, there will be a more significant loss of information in the streamtraced fluid displacements, as the characteristic size of the cells determines the degree to which more subtle fluid flow displacement is captured in the transition matrix. On the

other hand, a high resolution discretization with smaller fluid cells will retain more information in the streamtrace, but at the cost of additional operations during flow sculpting simulation. Stoecklein et al. (2016) conducted a study of discretization resolution by comparing information degradation vs. speed for the advection maps used in FlowSculpt, and found that a resolution of $N_Y = 801$, $N_Z = 201$ gives a satisfactory tradeoff between accuracy and speed. But even at higher resolutions, some multiple streamlines may start in unique cells at the 3D domain inlet, and end up in the same cell at the outlet. This causes fluid to accumulate in some cells, leaving gaps throughout the flow shape image (see Appendix B). While the bulk fluid motion and overall sculpted shape is accurately portrayed, this source of error can make comparisons to target fluid flow shapes more difficult.

To mitigate image degradation caused by artificial loss of mass, transition matrices were formed via "reverse advection", which performs streamtracing from outlet-to-inlet. That is, instead of pushing fluid forward along a streamline from a set of cells at a 3D domain inlet, a streamline from outlet-to-inlet looks at where fluid originates from at the inlet, and pulls it forward to the outlet. This yielded more visually complete flow deformations, as there is a uniform distribution of flow displacement information at the outlet for each transformation (see Appendix B). In this construction, each output cell has a single source cell (which may coincide with other cells' reverse streamtraces).

With these conditions, transition matrices can be collapsed down into a one-to-one *index mapping*. We represent index mappings as a function of 3D domain outlet cell index that maps to an inlet index. Let $A = \{a_i\}$ be an input cross-section flattened into a 1D array, and $B = \{b_j\}$ be the output cross-section. We can define a mapping $f : \mathbb{N} \to \mathbb{N}$ that relates an output cell index j in B to an input cell index i in A. This means a transition for a given mapping and input cross-section is performed by computing

$$b_j = a_{f(j)}$$

For all b_j in the outlet cross-section. The function f is one-to-one since each output index has exactly one input index. Figure 7.3 shows this mapping visually.

Index mappings gives us a method for computing the forward simulation equivalent to sparse

matrix-matrix multiplication, but with far fewer CPU operations, as the computation is now a matter of memory access, rather than arithmetic calculation. To use an index mapping, we find the index mapping f for each transition matrix and store it in memory at runtime. Instead of performing matrix-matrix multiplication, which in general has a worst case runtime complexity greater than $\mathcal{O}(n^2)$, (Yuster and Zwick, 2005) the index mapping method is essentially a lookup table, with a runtime complexity of $\mathcal{O}(1)$ (Cormen et al., 2001).

7.2.2.3 Fitness Function.

Previously, flow sculpting design optimization has computed fitness using a correlation function $r(\mathbf{I}_{sim}, \mathbf{I}_T)$, which compares a chromosome's simulated flow shape image $\mathbf{I}_{sim}$ to a target flow shape $\mathbf{I}_T$, each of size $N_Y \times N_Z$:

$$r(I_{sim}, I_T) = \frac{\sum_{i}^{N_Y}\sum_{j}^{N_Z}(I_{sim} - \overline{I_{sim}})(I_T - \overline{I_T})}{\sqrt{(\sum_{i}^{N_Y}\sum_{j}^{N_Z}(I_{sim} - \overline{I_{sim}})^2)(\sum_{i}^{N_Y}\sum_{j}^{N_Z}(I_T - \overline{I_T})^2)}} \tag{7.1}$$

The correlation function r is a *per-pixel* comparison, producing a measure of distance between the two images I_{sim} and I_T, with a value of $r = 1.0$ meaning that both images match pixel-to-pixel[3]. This kind of measure is useful if the goal is to design a flow sculpting device for a target flow shape of a certain size, orientation, and location within the microchannel cross-section.

On the other hand, if the designer's goal is to find the desired shape regardless of its location, using the correlation function will require many optimization runs, each time translating the target image to some new location within the cross-section. Similarly, the goal could be to sculpt a shape with a certain topology, with little regard to the precise shape of the sculpted flow, or to displace fluid generally to some target region in the channel cross-section, with no regard at all to its resulting shape. Indeed, as we cannot fully anticipate the needs of every researcher using the flow sculpting platform, we have taken steps to make open-ended design problems easily approached in FlowSculpt. To allow designs for which flow shape location

[3] $r = -1.0$ implies that both images are perfectly complementary, which is still a suitable design if the inlet flow pattern's channel design is inverted. Hence, the fitness computed in FlowSculpt squares the value computed by r: $\mathit{fitness} = r(\mathbf{I}_{sim}, \mathbf{I}_T)^2$.

is not important, we have included a translation invariant fitness function. We have also provided hooks within the FlowSculpt code to easily access the Open Source Computer Vision Library (OpenCV) (Bradski, 2000), which has many image processing routines, thus making the creation of customized fitness functions a more simple task. These implementations are detailed next.

Translation Invariant Fitness Function. To provide a translation invariant fitness function, which allows to search for a target flow shape at any translated position within the channel, we compute the Fast Fourier transform (FFT) of the target I_T and chromosome-produced simulated image I_{sim}. We then exploit the "shift" property of the Fourier transform, whereby the translation of an image will be reported as a phase shift in frequency space, while the magnitude of the transforms will remain the same (Srinivasa Reddy and Chatterji, 1996). By comparing the magnitude of each image's FFT we introduce translation invariance to the measure of fitness, thereby searching for a fluid flow shape without regard to where it is located within the microchannel cross-section. In FlowSculpt, this is accomplished by using the previously described correlation function r on the difference between the images' magnitudes in Fourier space, which are computed using FFTW (Frigo and Johnson, 2005):

$$\mathit{fitness} = r(|\mathcal{F}(I_{sim})|, |\mathcal{F}(I_T)|) \tag{7.2}$$

The use of a translation invariant fitness function comes at a price, as FFT computation greatly increases chromosome fitness evaluation time. Moreover, including translation invariance complicates and expands the design space, making GA convergence more difficult. There is no getting around the increased runtime due to heavier computational load, but a larger population size and additional GA repetitions can help alleviate difficulty exploring an expanded design space.

Custom Fitness Functions via OpenCV. We have separated the implementation of fitness functions from FlowSculpt's primary routines as a standalone source file `fsfitness.cpp`, and included the use of OpenCV's image processing operations with several examples.This allows users to specialize their flow sculpting design optimization based on domain knowledge

and problem-specific criteria. For example, when attempting to design porous shapes, a user may desire only that the shape matches a particular topology (e.g., porous "holes" within a fluid structure), without requirements on the precise location, size, and orientation of the shape within the channel. In this case, a pixel-by-pixel measure (such as the correlation function r) would be debilitating, as it requires optimal fluid flow shapes to precisely match the target image, thus pinning the search space based on the user's intuition (and artistic skill). On the other hand, a more qualitative measure such as the Euler number for binary shapes (Pratt, 1991; Horn, 1986), which returns a scalar value for the number of objects in the image minus the holes in each of the objects), would provide a completely different search landscape that preserves the desired topology of the target flow shape in optimal designs. We have partitioned the implementation of fitness functions from the rest of the FlowSculpt framework, and provide several templates for custom fitness function design. As an additional aid to researchers designing their own fitness functions, we have also added support for the OpenCV library. OpenCV is a freely available computer vision library that contains a multitude of techniques for image processing and comparison. We anticipate that future publications will explore OpenCV methods for flow sculpting, but researchers can begin utilizing it in their own work immediately.

7.2.3 Graphical User Interface

We have embedded the FlowSculpt software (written in C++) within a GUI using the cross-platform application framework Electron[4], which makes use of Javascript, HTML, and CSS. The GUI, shown in Fig. 7.4, is freely available at www.me.iastate.edu/bglab/software, and can be easily installed on Windows, MacOS, and Linux operating systems.

The top section of the GUI is dedicated to user input for the target flow shape, with a drawable canvas and modifiable drawing tool. Within this section, there is a "Half-Pillars" selection, choosing whether their design should be symmetric about the cross-section midplane (using only full-pillars), or asymmetric (using both full- and half-pillars). In the former case, anything drawn in the top-half of the canvas will be automatically mirrored in the lower-half, and vice versa; in the latter case, the entire canvas is open to shape design.

[4]https://github.com/electron/electron

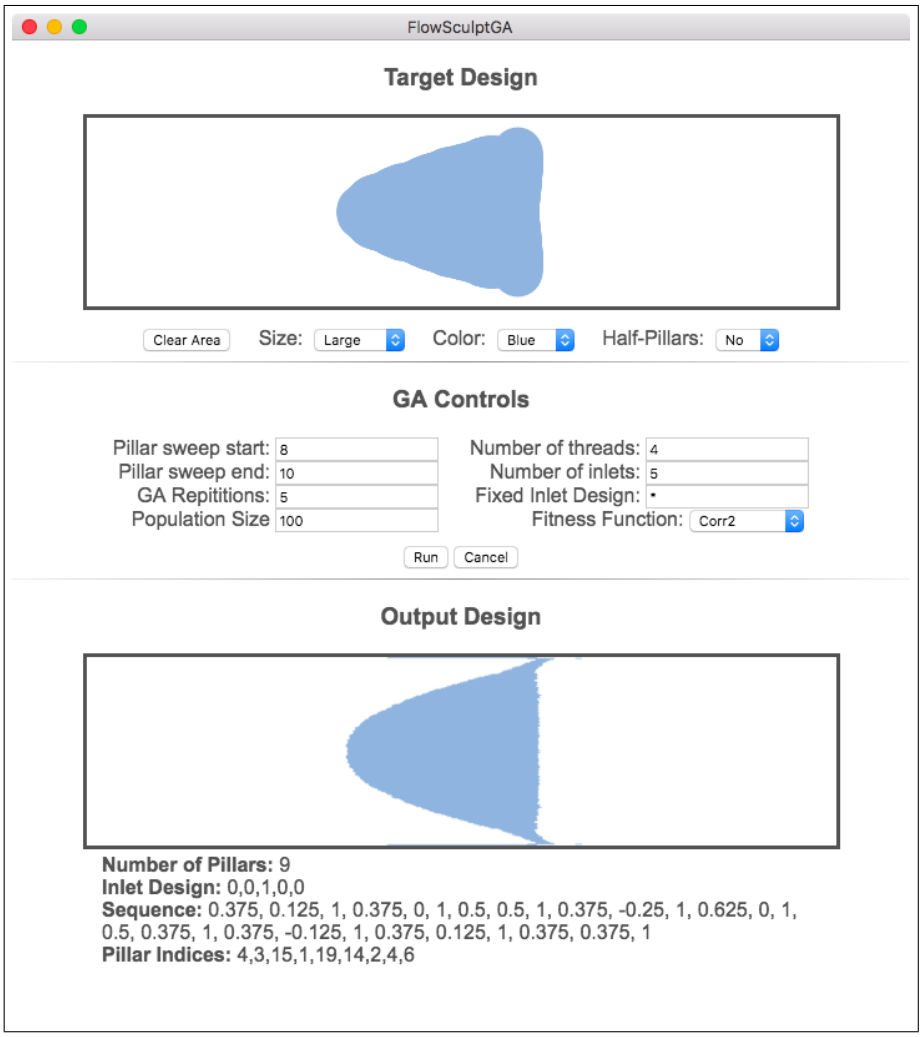

Figure 7.4 The graphical user interface (GUI) for FlowSculpt, created using Electron. The top section (Target Design) contains a drawing canvas for the user-designed target flow shape, with adjustable marker size and color (blue/white for sculpted flow/co-flow). By default, FlowSculpt will use pillars spanning the height of the channel, forcing symmetric design within the canvas (i.e., drawing in the top-half will be mirrored in the lower-half, and vice versa). By choosing to use half-pillars, the entire canvas can by drawn on. The middle section (GA Controls) configures parameters for FlowSculpt's optimization.

The middle section of the GUI configures runtime parameters for FlowSculpt: Pillar sweep start/stop will choose the allowable number of pillars; GA repetitions determines how many times the GA is repeated per pillar sequence length; Population size controls the GA population; Number of threads chooses how many threads (i.e., CPU cores) to use during parallel evaluation; Number of inlets limits how many channels the GA can use to create an inlet flow pattern; Fixed inlet design allows the user to force a particular inlet flow pattern (e.g, [0, 0, 1, 0, 0]); Translation invariance forces the GA to use the FFT-based translation invariant fitness function described within this paper. The default values for each configuration are sufficient to provide a quick result within a well-resolved design space (1-10 pillars), but users are encouraged to modify these values to ensure a more exhaustive search that is tuned to their experimental constraints. Within this section are buttons to run FlowSculpt on the target design, and cancel the run.

The bottom section of the GUI gives the resulting design from FlowSculpt optimization, along with the number of pillars used, inlet flow pattern, and pillar sequence. The pillar sequence design can also be exported to the open source computer-aided design (CAD) software "OpenSCAD"(Ravindran, 2017), which can aid in integrating the flow sculpting device with larger microfluidic devices, or used to export the design for photolithography fabrication as a standalone device.

7.3 Results and Discussion

We have tested the FlowSculpt framework in two campaigns: (1) benchmark testing and (2) arbitrary flow shape design. The benchmarks are intended to showcase the speed of the index map forward model and demonstrate the tradeoffs involved with translation invariant design. For arbitrary flow shapes, we designed several asymmetric flow shapes using FlowSculpt with half-pillars, targeting letters of the Roman alphabet (which we have no reason to believe should necessarily exist in the flow sculpting design space).

7.3.1 Benchmark Tests

We recorded results for benchmark images with several test cases to evaluate the benefit of the modifications to FlowSculpt. All tests were run on a system with a 6-core (12 thread) Core i7 4960X processor and 32GB of ram running Ubuntu 16.04. The cases we ran were the following:

Case Study 1: Index Map Runtime We performed a speed comparison of FlowSculpt using the index map forward model vs. matrix multiplication.

Case Study 2: Translation Invariant Design We tested the relative speed of the new GA with and without an FFT pass for translation invariance.

Case Study 3: Translation Invariant Runtime We created a new set of benchmark images to evaluate the benefits of translation invariance.

7.3.1.1 Case Study 1: Index Map Runtime.

A set of 60 randomly generated target flow shapes were used to compare FlowSculpt's forward model using index maps vs. matrix-multiplication, as used in Refs. (Stoecklein et al., 2016, 2017a). These flow shapes are the same set used in Ref. (Stoecklein et al., 2017a). The purpose of this test was to compare the use of transition matrices (using matrix-matrix multiplication) to the index maps introduced in the present work. We used the same parameters for both GA setups aside from the modified fitness evaluation. GA parameters were a population size of 100, sweeping from 5 to 10 pillars with 10 repetitions of the GA at each step (for a total of 60 GAs per flow shape search). For all runs in case 1, we used 801×101 (half-channel) resolution cross-sections, and fixed the inlet flow pattern. It should be noted that these tests - using full-pillars and a fixed inlet - represent the simplest, most constrained search possible. The use of half-pillars (which double the image size) or free inlet design (which greatly expands the design space) will necessarily increase FlowSculpt's runtime.

Results are shown in Fig. 7.5. The speedup from the use of index maps is significant, with no practical difference in accuracy. Where the matrix-matrix multiplication based framework required, on average, $\approx$5000 s (1.4 h) per image search, the index map based framework required

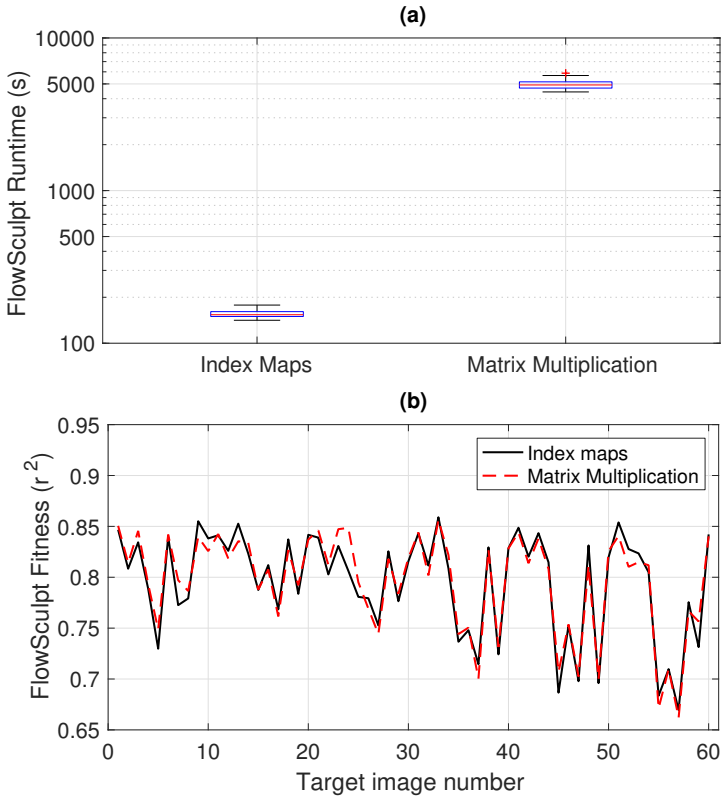

Figure 7.5 Results from case study 1. (a) Runtime comparison of the FlowSculpt framework between the use of index maps and matrix multiplication, showing a reduction of over 2 orders of magnitude by using index maps. (b) Optimal fitness comparison of index map and matrix multiplication based on 60 randomly generated target flow shapes, showing comparable results.

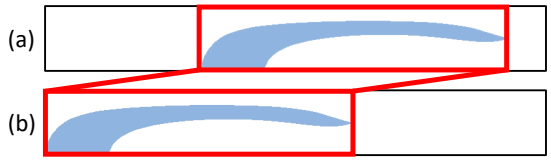

Figure 7.6 Method of translating benchmark images for testing translation invariant search (note: these are half-channel cross-sections, as used within FlowSculpt). **(a)** Shows the original benchmark image with first and last column identified. **(b)** Shows the artificially translated image.

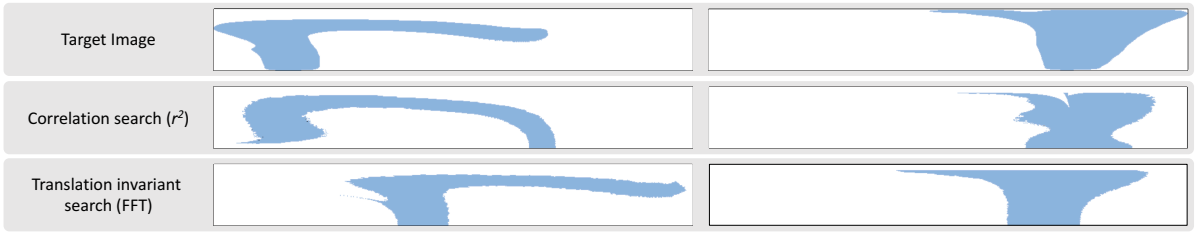

Figure 7.7 Results from case study 2, comparing how the GA performed the per-pixel correlation fitness to FFT-based translation invariance. Two test images are shown. The top row contains the target images. The middle and bottom rows show the best result from the GA using the correlation and translation invariant fitness functions, respectively.

≈145 s (2.5 m), on average. This is approximately a 34× reduction in runtime, with the same quality of results. Such drastic improvements in forward model runtime are critical for optimization problems, as they allow for overall optimization routines to favor coverage of the search space with little to no deference to runtime. In FlowSculpt, this means a single flow shape search can use hundreds of GA repetitions per pillar sequence length, in the same time that only 10 repetitions were previously used. A faster forward model also provides additional breathing room in the fitness function pipeline to implement computationally costly methods, such as the translation invariant FFT post-processing presented in this work.

7.3.1.2 Case Study 2: Translation Invariant Design.

To test the use of the FFT's shift property for translation invariant searches in FlowSculpt, we artificially translate each of the 60 target flow shapes from case study 1 as depicted in

Fig. 7.6, where a flow shape produced by the forward model is translated to the left edge of the channel cross-section. By translating flow shapes produced by the forward model, we can test the FFT-based translation invariant fitness function against targets with known solutions. In this case study, we allow FlowSculpt to design the inlet flow pattern.

The translation invariant search successfully found matching flow shapes for each altered target image, while the per-pixel correlation searches were pinned to the shifted target within the channel, with poorly matching results. Fig 7.7 shows two example target flow shapes with a comparison of the results from the use of only the correlation function, and with use of the FFT-based translation invariant search. When FlowSculpt uses the correlation function only, it capably matches the horizontal location of the target flow shape, but fails to accurately capture details of the shape. On the other hand, the FFT-based translation invariant searches show well-resolved flow shapes, though at different locations within the channel cross-section.

These results are very encouraging, not only for the immediate use of translation invariant design, but additional post-processing operations within FlowSculpt's fitness function. Despite the considerably expanded design space, FlowSculpt was still able to effectively design both the inlet flow pattern and pillar sequences for known transformations.

7.3.1.3 Case 3: Speed comparison with translation invariance.

We ran an additional test on the FFT-based translation invariant fitness function to analyze FlowSculpt runtime and GA convergence. In this case study, we use FlowSculpt's FFT and non-FFT fitness functions on the original set of 60 target flow shapes from case study 1 in order to isolate differences in performance to the FFT-based function's difficulty in searching a translation invariant design space. That is, both fitness functions are capable of finding exact solutions to these target flow shapes, but the FFT-based fitness function will have a more difficult search.

Fig. 7.8(a) shows a comparison of FlowSculpt runtime with and without translation invariance, while Fig 7.8(b) shows the number of generations needed for GA convergence, across the entire test set. The use of the FFT increases FlowSculpt runtime by $\approx$3.5$\times$, with an average FlowSculpt runtime of $\approx$ 442 s with FFT vs. $\approx$ 145 s without. As expected due to the more di-

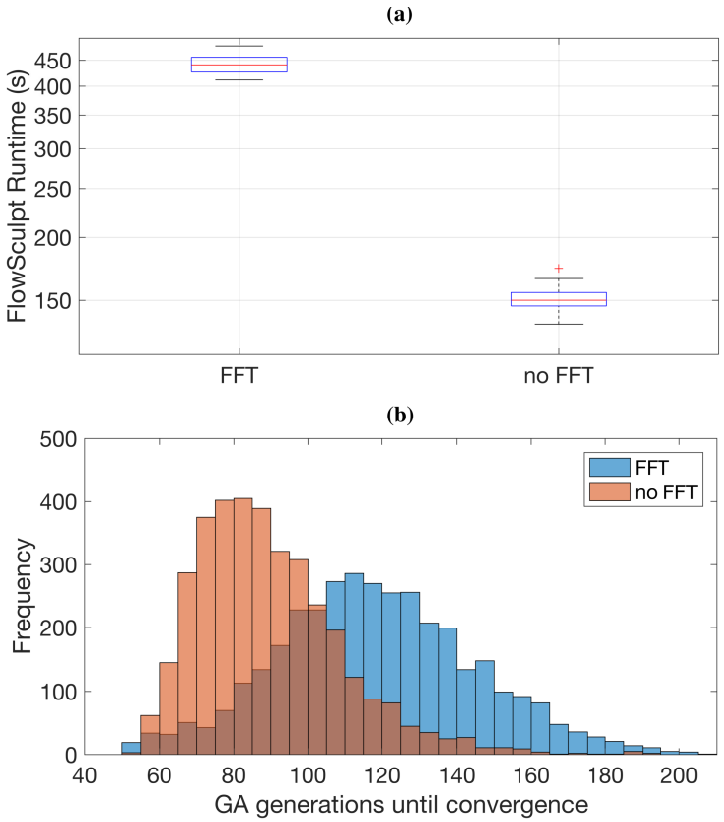

Figure 7.8 Results from case study 3, a runtime comparison of FlowSculpt using the translation invariant (FFT) and per-pixel correlation (no FFT) fitness functions. (a) Using the FFT substantially increases FlowSculpt's runtime, from an average of ≈ 145 s to ≈ 442 s. (b) A translation invariant search space requires more generations within a GA to converge, which also contributes to the increased runtime.

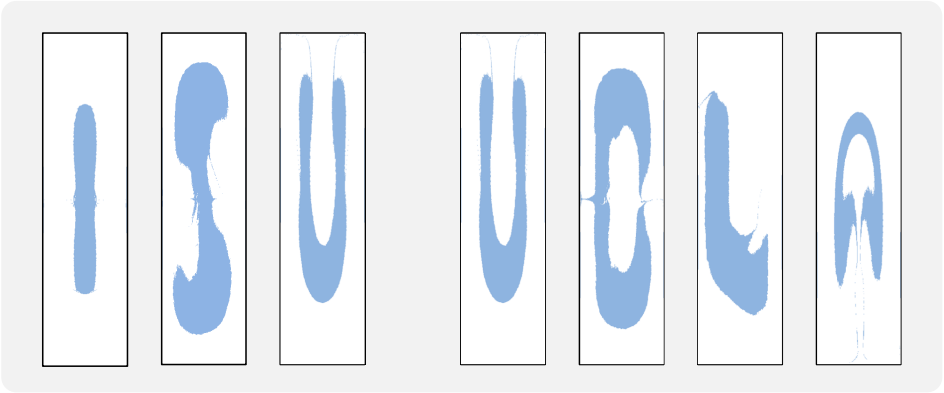

Figure 7.9 By incorporating half-pillar configurations into FlowSculpt's GA chromosome, a wealth of asymmetric designs are now possible. Here, the letters in "ISU" and "UCLA" are found with a channel aspect ratio $h/w = 0.25$, and shown as rotated shapes for clarity. The device designs were fabricated, and sculpted flow imaged using confocal microscopy.

verse design space in translation invariant search, FlowSculpt's GAs required more generations to converge, with an average of 120 generations until convergence with FFT vs. 89 generations without FFT.

Despite the increase in runtime in using the FFT-based fitness function when compared to the correlation function, consider that its use avoids a campaign of manually-translated target flow shapes. In fact, for these test images, a user need only attempt 4 such manual translations before their campaign runtime exceeds FlowSculpts FFT-based translation invariant design, which searches a continuously translated space in an automated fashion.

7.3.2 Arbitrary flow shape design

We show how half-pillar designs fare in arbitrary design via FlowSculpt by searching for device designs which create the capitalized letters in "UCLA" and "ISU", and validate the resulting designs (Fig 7.9(b)). For such arbitrary searches where there is no guarantee of finding a matching flow shape, iterative design is often required. This iterative process would begin with a user-provided target flow shape to serve as an initial guess. The FlowSculpt framework will attempt to match this shape, but if the solution is not sufficient for the user's application, this domain response can be used as an indication of what is possible, and inform the second

FlowSculpt search. For the designs shown in Fig 7.9, we used the per-pixel correlation metric, as we desired the letters to exist in a similar region of the microchannel cross-section, rather than a scattered set of locations, which may be produced using the translation invariant FFT search.

7.4 Conclusion

FlowSculpt provides an intuitive and easy to use graphical user interface for researchers to leverage when designing fluid deformations. It allows users to design a target flow shape graphically, and gives a rich set of options for search optimization including the number of threads to use for parallel evaluation, a number of pillars allowed, and translation invariant design. The GUI is freely available as a user-friendly, cross-platform app, which we hope sees widespread adoption by the microfluidics community. We believe FlowSculpt will help usher in a new generation of microfluidic devices for flow engineering, whether to advance microscale manufacturing processes, enhance existing bio-sensing platforms, or devise new bioengineering technologies. FlowSculpt's cross-platform architecture, combined with its newfound computational efficiency, should allow users with extremely modest computing hardware to engage in flow sculpting design optimization.

With this work, we not only present a highly functional application for design, but a significant step forward in its evolution. The design elements we have added to the FlowSculpt framework vastly increase its design space through both additional microfluidic components (half-pillars) and advanced fitness functions to modify the design space (FFT-based translation invariance and OpenCV functions). Future work will likely continue along these routes, incorporating new geometries such as curved channels for Dean flow induced flow deformation, and more specialized fitness functions for unique flow shapes. We anticipate further development on user-provided microfluidic parts, allowing easier integration of FlowSculpt with researchers' existing microfluidics-based projects. Beyond this, FlowSculpt itself can be brought into other regimes of hierarchical design, such as tailored 3D particles via stop-flow lithography (Paulsen et al., 2015) or transient liquid molding (Wu et al., 2015), enabling a new class of complex microfluidic engineering.

CHAPTER 8. CONCLUSIONS AND FUTURE WORK

8.1 CONCLUSIONS

Computational methods and utilities have been developed to provide the microfluidics community with knowledge and intuition of inertial fluid flow sculpting, and enable them to easily and rapidly integrate this exciting set of flow physics into their own research. The uFlow software contains thousands of complex numerical simulations, accessible at the click of a button, and provides an expressive teaching tool for instructing researchers, students, industry, and laypeople on the potential of flow sculpting. It also provides visually useful estimations not only of convective flow deformation, but of transverse mass diffusion, and 3D particles fabricated using stop-flow lithography or transient liquid molding. This functionality was demonstrated with the design and experimental validation of fundamental and hierarchical flow shapes, diffusion estimations, and polymerized 3D particles. The FlowSculpt application is a more intensive and directed utility for solving the inverse design problem, and has proven capable of optimizing microfluidic devices to deform fluid into a user-specified flow shape.

In tandem with this software, numerical routines were implemented for efficient storage and rapid access to a library of thousands of pre-computed advection maps. Additionally, an extremely fast, experimentally validated forward model for simulating flow sculpting devices was developed. This model was easily pipelined into the genetic algorithm and deep learning frameworks to solve the inverse problem, and will likely be adapted to other optimization routines as the flow sculpting library grows.

With this work, user-friendly approaches to understanding, designing, and optimizing microfluidic devices for inertial flow sculpting are freely available for public use.

8.2 FUTURE WORK

Flow sculpting presents a number of avenues for future research, with the present work only beginning to explore what is possible. Immediate lines of inquiry can be categorized in the following ways: (1) extensions to the uFlow and FlowSculpt software; (2) further characterization and analysis of inertial flow physics; and (3) adapting the design tools from this work to new problems in microfluidics.

8.2.1 EXTENSIONS TO UFLOW/FLOWSCULPT

The computational platforms of uFlow and FlowSculpt are easily extended to include new types of microfluidic components. Implementing curved channels, which have already seen wide use in microfluidic devices, would add Dean flow operators to the library of flow deformations, and allow flow sculpting devices to occupy a more compact space on a lab-on-a-chip. Different channel aspect ratios have been shown to produce completely different sets of flow transformations, and should be made available for design. Additional top-bottom symmetry-breaking geometries, such as variable-height pillars or backward-facing steps, should be pursued as well. Beyond simple additions to the library of advection maps, both softwares could be combined into a one-stop-shop for inertial flow sculpting. Future updates should seek to enhance user intuition with the inertial flow design space, and add pathways for integrating home-grown microfluidic components to the toolkit. The availability and ubiquity of the software should also grow - perhaps offering lightweight versions embedded within an internet browser, and an alternative high-performance version for those with access to large computational resources.

8.2.2 EXPLORING INERTIAL FLOW PHYSICS

Despite many decades of study, there remain a number of unexplored aspects of fundamental inertial physics. Pertaining to this work, a comprehensive characterization of obstacle-induced flow deformations could define an optimized set of pillar geometries, tuned to maximize the variety of flow deformations accessible to FlowSculpt in a constrained and efficient data set. Available pillars could be biased toward easier manufacturability, avoiding pillar designs that

are more sensitive to fabrication errors changing the mode of flow deformation. In addition, where the current library of pre-computed deformations are uniformly distributed in pillar diameter and location, a reduced set of configurations could more efficiently cover all possibe flow deformations with less redundancy.

Pillar sequences could also be designed with the purpose of undoing other inertial effects within a channel, thereby maintaining the appearance of Stokes flow in the net flow deformation, but retaining the high-throughput associated with moderate Re. In addition, non-Newtonian and multi-phase flows could be investigated as separate branches of fluid flow sculpting (not necessarily inertial). These would be especially challenging given their relative complexity compared to the Newtonian fluids investigated in this work, but such research could lead to a completely new set of applications.

8.2.3 DESIGN FOR OTHER FLOW PROBLEMS

Although this work focused only on the fluid itself, inertial effects are also commonly used to control larger particles in flow. This has typically been done by allowing particles to focus from a random location within the channel to one of several pre-defined cross-sectional locations (depending on the channel geometry). In some cases, secondary flow has been used via Dean flow or pillar structures to reduce the number of focusing positions, but this still requires some length of a straight channel to allow the particles to fully focus. If particle deflection from a bluff-body obstacle over some length of channel can be well-represented as an advection map, then many of the design tools developed in this work could be used to steer particles of a particular size to a certain location. This would be a more difficult problem, but it would have immediate utility for much of the microfluidics community.

APPENDIX A. NUMERICAL EXPERIMENTS COMPARING PRESSURE DROP IN A CHANNEL WITH AND WITHOUT OBSTACLES

While the primary objective in flow sculpting optimization is to most closely match a target fluid flow shape, a secondary objective is to minimize the required pressure to drive flow, which can create complications when the device is in use (e.g., flexure in the channel). This motivates the reduction of pillar sequence length, rather than the size of the pillars used, as each pillar in a sequence necessarily increases the microchannel length. Even in the situation where the pillar's contribution to pressure drop is especially large (D/w = 0.750), its use is still preferable to due to an otherwise unobtainable transformation. That is, the goal of matching the target flow shape - which is enabled by maximizing the number of different fluid deformations - takes preference over pressure drop minimization. A set of numerical simulations are shown here to demonstrate that the microchannel length is indeed the dominant contribution to pressure drop, rather than the pillar obstacle, for the pillar library used in this work. For all simulations, the channel aspect is $h/w = 0.25$, $Re = 20$, and pressure is compared non-dimensionally as $p^* = \frac{pD_H}{\mu U}$ for the dimensional pressure p, channel hydraulic diameter D_H, and fluid viscosity μ and average velocity U.

Fig A.1 shows pressure fields from solution of the Navier-Stokes equation (numerically solved using an experimentally validated finite element framework) on two different 3D domains of the same channel length, one empty channel and one with a pillar ($D/w = 0.750$). By looking at this largest pillar size, we see the worst-case scenario for pressure requirements to drive flow for the library of pillar diameters in this work. Here, the introduction of the pillar has effectively doubled the pressure drop. However, as shown in Fig A.3, the resulting deformation is unique for the available pillar diameters, and cannot be replicated with smaller pillar diameters.

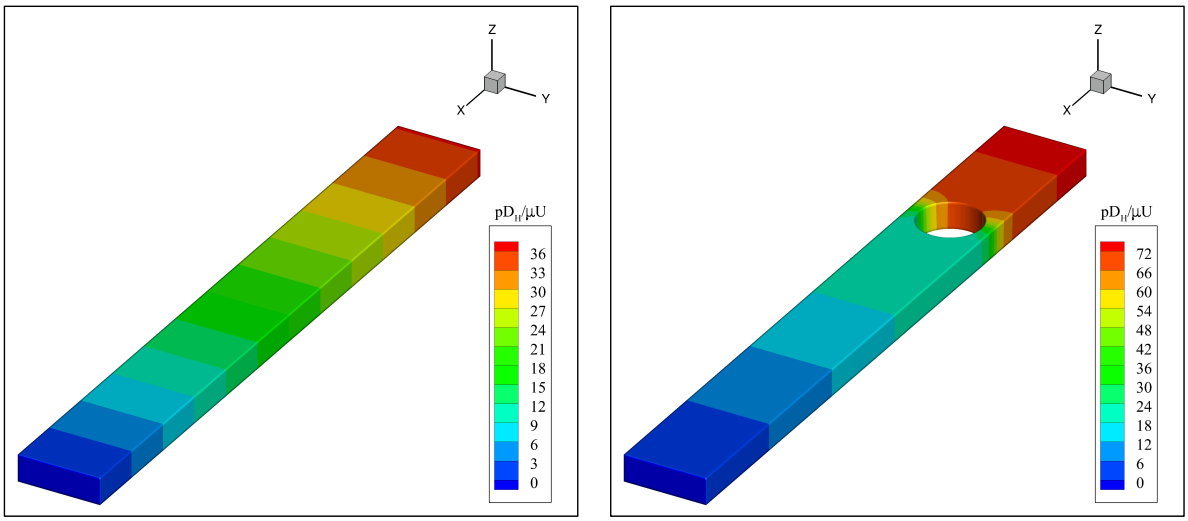

Figure A.1 Visual comparison of the pressure field contour for an empty channel (left) and a channel containing a pillar of diameter $D/w = 0.750$ (right).

For smaller pillars of $D/w = \{0.375, 0.500, 0.625\}$, the pressure drop increase is less substantial. We measure the impact of pillar on the channel pressure drop by $\frac{\Delta P^*_{empty} - \Delta P^*_{pillar}}{\Delta P^*_{pillar}}$, as shown in Fig A.2. For these smaller pillar sizes, the contribution to pressure drop from the pillar ranges from 12%-33%, indicating that the length of the channel is largely responsible for the pressure drop. Thus, minimizing the channel length (rather than pillar diameter) is the preference here.

One could try to minimize pressure drop by re-creating a fluid flow deformation with a larger sequence of smaller pillars. This is a futile endeavor, as the increased channel length will significantly increase the required pressure. Fig 4.3 shows a series of fluid deformations for all pillar diameters, with each pillar being placed at the center of the channel. In each case of a fluid transformation being replicated with a larger sequence of smaller diameter pillars, the required pressure drop is greater. Moreover, some fluid deformations are inaccessible with smaller pillars. For example, all of the D/w = 0.750 fluid deformations are unique within this set. Thus, a pillar with $D/w = 0.750$ cannot be substituted with a sequence of any smaller pillars (unless the fluid flow shape details that make these deformations unique are trivial to the end-user).

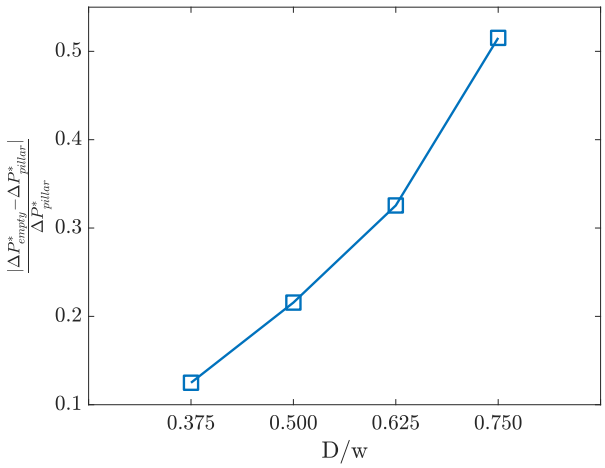

Figure A.2 Proportion of pressure drop due to the presence of pillars of difference sizes. Note that although the largest diameter $D/w = 0.750$ contributes more than half of the difference in pressure drop, its unique deformation makes it non-expendable in a fully-informed flow deformation library (see Fig A.3).

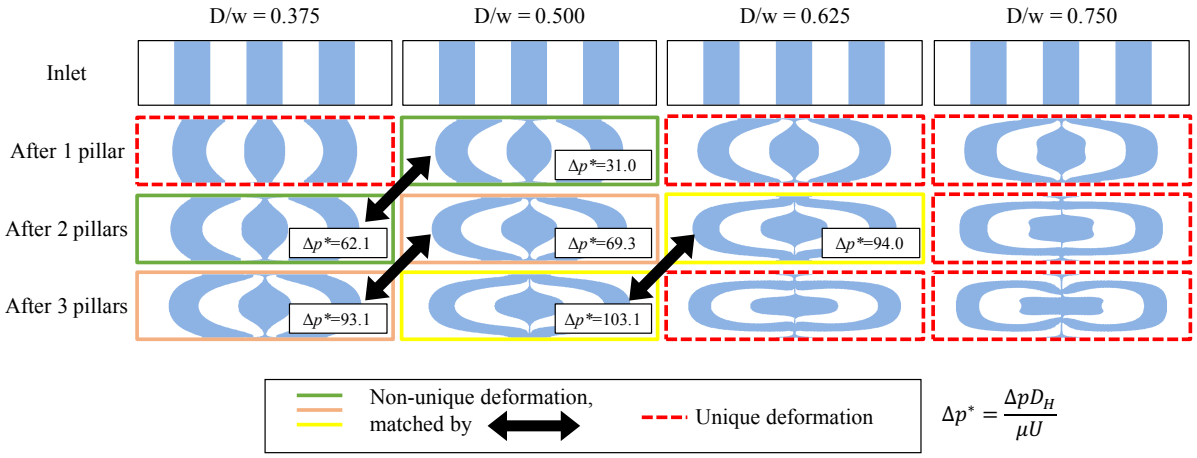

Figure A.3 Visual comparison of fluid deformations from pillar sequences of constant diameter (by column), with each pillar located at $y/w = 0.0$. Several transformations (solid highlights in green, orange, and yellow) can be created across pillar sizes, using different numbers of pillars (matching deformations shown with double arrows). Corresponding non-dimensional pressure drop is inset with each fluid deformation. Other fluid flow transformations are effectively unique within this set (dashed red highlights), and cannot be recreated with any other pillar diameters shown here.

APPENDIX B. THE FORWARD MODEL: FORWARD AND REVERSE ADVECTION, AND SPARSE TRANSITION MATRICES

Transition matrices show how particles move from inlet-to-outlet. For this work, the stream-traces that form transition matrices are performed using reverse advection. This is done to avoid an issue with forward advection, which may displace multiple fluid elements to the same location (column) in the transition matrix. Such a matrix has "blank" elements at the outlet, where there is no apparent fluid (see Fig. B.1(a)) . Reverse advection ensures a uniform and complete distribution of displacement information at the outlet, which translates to a more contiguous and accurate flow shape image (see Fig. B.1(b)).

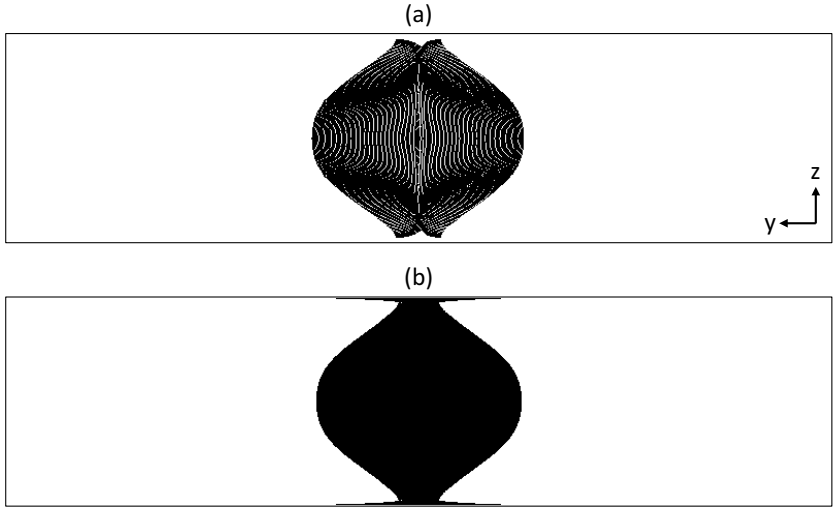

Figure B.1 Demonstration of how the use of (a) forward advection and (b) reverse advection affects the quality of flow shape images created using the forward model described in this work. For clarity, sculpted flow is visualized as black pixels. Note that although the overall fluid shape is represented in either case, interstitial pockets of "empty" fluid cells present in the forward advection image are not present for reverse advection. This is due to the uniform distribution of displacement information in reverse advection maps.

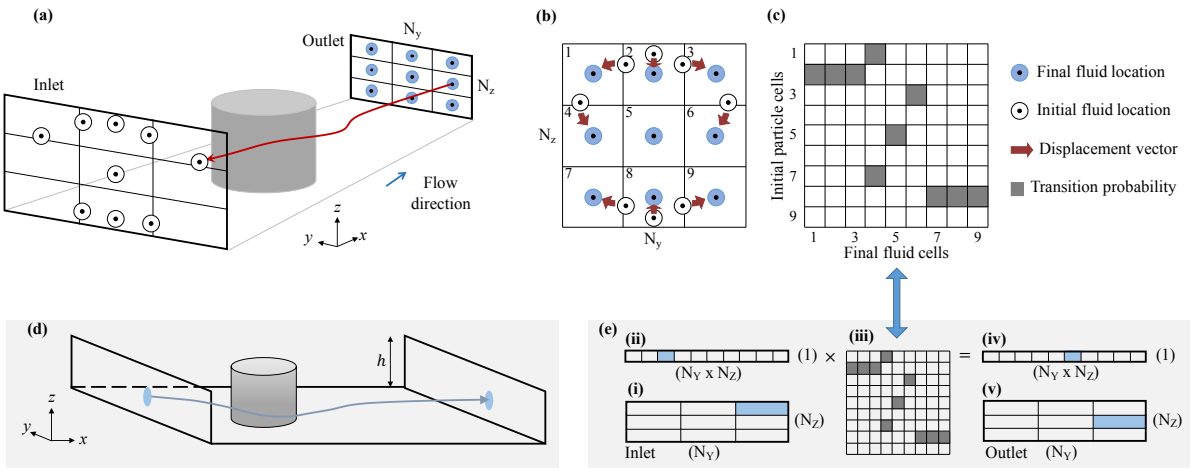

Figure B.2 (a) A simple illustration of how reverse advection is used to create an advection map (b) that contains uniformly distributed information for fluid displacement at the microchannel *outlet*. The advection map can then be converted into a column-stochastic transition matrix (c), for which every row and column represents a fluid element in the 2-D cross-section shown in (a) for the inlet and outlet, respectively. Thus, the displacement for a fluid element that would otherwise be calculated by streamtracing through a 3-D domain (d) can be computed using only matrix multiplication (e). The inlet fluid states are discretized in the same way as the transition matrix (e)(i), and then reshaped to form a row vector (e)(ii). This is then multiplied by a transition matrix (e)(iii), which produces the outlet fluid states as a row vector (e)(iv). This can then be reshaped into the 2-D representation of the domain (e)(v).

The use of transition matrices allows for the forward model in flow sculpting to be ported to any computational platform capable of performing linear algebra. Work done in chapter 7 makes even this advantage obsolete, but the index maps used there are still formed from transition matrices. The steps to create and use a transition matrix for flow sculpting are outlined in Fig. B.2.

APPENDIX C. RAY MARCHING SCHEME FOR 3D PARTICLE VISUALIZATION

Ray marching (Hart, 1996) is used with signed distance functions (SDFs) to render the intersection of the the extruded flow shape and the UV optical mask-defined shape. An SDF $d(\mathbf{x})$ returns the shortest distance from any location, $\mathbf{x}$, in 3D space to the surface of a 3D volume. We are interested in the intersection of two 3D volumes, which can be found by taking the maximum value of two SDFs calculated for a single point:

$$d_{int}(\mathbf{x}) = max(d_1(\mathbf{x}), d_2(\mathbf{x})) \tag{C.1}$$

Ray marching uses this distance to determine which on-screen pixels comprise the surface of the microparticle, and assign proper color and lighting to the object. The overall microparticle rendering algorithm is shown in Fig. C.1, and enumerated here:

1. Set a bounding box for the microparticle, then use it to determine which screen pixels may possibly contain the particle. Rendering will only happen for screen pixels that include this domain.

2. For each pixel that may contain the particle, produce a ray (vector) $\mathbf{r}$ that projects into the domain in the direction normal to computer screen (i.e., a vector in the direction of the viewer).

3. Initialize a pixel depth to the boundary of the bounding box.

4. Iterate the following loop for each pixel that overlays the bounding box:

   (a) Calculate the distance to the microparticle $d_{int}(\mathbf{x})$ at the current location $\mathbf{x}$, computed from pixel screen coordinates and depth.

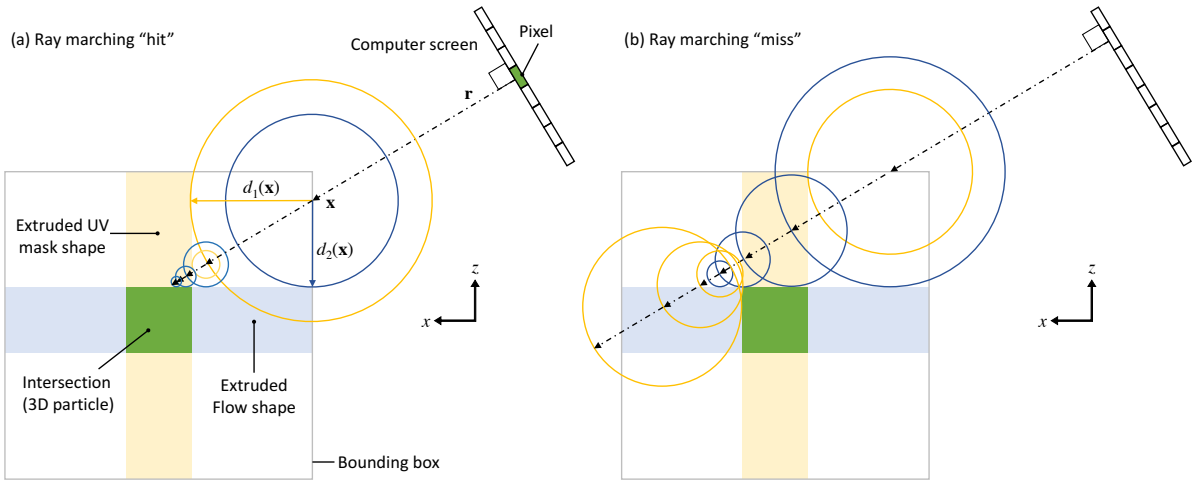

Figure C.1 A side-on illustration of how ray marching uses signed distance functions (SDFs) to visualize the intersection of two volumes. Each volume (shown here as two rectangular paths, with the intersection colored in green) has an associated SDF: $d_1(\mathbf{x})$ for the mask-defined volume, and $d_2(\mathbf{x})$ for the flow shape volume, for a point $\mathbf{x}$ in 3D space. For a set of pixels which cover the bounding box, a ray $\mathbf{r}$ is cast in a direction normal to the computer screen to a depth of the bounding box in 3D space. The SDFs d_1, d_2 are computed at this point, and the depth of $\mathbf{r}$ is increased by $d_{int} = max(d_1(\mathbf{x}), d_2(\mathbf{x}))$. This process of computing $d_{int}(\mathbf{x})$ and incrementing $\mathbf{r}$ by the result continues until either (a) $d_{int} < 10^{-6}$, whereby the ray has hit the volume and the pixel is colored, or (b) the ray exits the bounding box without hitting the volume, and the pixel is not colored. Note that at each point $\mathbf{x}$, it is the larger of d_1, d_2 which determines the distance to the intersection of the two volumes. Hence, $d_{int} = max(d_1, d_2)$.

(b) Increase the depth of this pixel along its ray $\mathbf{r}$ by the distance returned by $d_{int}(\mathbf{x})$.

(c) If the pixel depth exits the bounding box, we break from the loop without coloring the pixel. In the case, although the pixel depth moved the distance from the previous depth to the microparticle surface, the ray direction $\mathbf{r}$ did not intersect the microparticle volume.

(d) At the new location, if $d(\mathbf{x}) < tol$ for a tolerance $tol = 1 \times 10^{-6}$, the current pixel position is on the surface of the particle. We break from the loop, and calculate the pixel's color and lighting.

5. To color the pixel appropriately, compute the surface normal $\mathbf{N} = \nabla d_{int}(\mathbf{x})$ using finite differences.

6. The surface normal $\mathbf{N}$ and pre-defined light source direction $\mathbf{L}$ are used with the diffuse ambient light coefficient A to compute the surface intensity coefficient I_D via the Lambertian reflectance model (Blinn, 1977):

$$I_D = \mathbf{N} \cdot \mathbf{L} + A \tag{C.2}$$

7. Multiply the intensity coefficient by the color of the fluid stream at $\mathbf{x}$ to produce the color for the current pixel at the surface of the microparticle.

While most of the particle rendering algorithm is done on the GPU, SDFs are actually well suited to CPU calculation. We use the `scipy.ndimage` python module to pre-compute SDFs as fields $\mathbf{d_1}$, $\mathbf{d_2}$ for the two particle-defining shapes from the UV-mask and flow sculpting device, both extruding into the microchannel (we assume that the UV light will be exactly perpendicular to the channel flow direction during fabrication). These fields are then sent to the GPU as 2D textures, with the x-component of the field being unnecessary since each volume is simply an extruded 2D shape.

APPENDIX D. ADVECTION MAP LIBRARY COMPRESSION VA PRINCIPAL COMPONENT ANALYSIS

Principal Component Analysis (PCA) is used to compress advection map data to $\approx 9.3\%$ of its original size, and to interpolate new maps that were not a part of the original data. Here, the error introduced through this compression method is shown by comparing reconstructed data to its original source.

Each advection map is a set of $N_Y \times N_Z$ displacements in the $y-z$ plane, stored as (dy, dz) floating point values. The dy and dz data for all maps are compressed via PCA as separate datasets, as the displacements have different variance between the two directions, which is harder to capture if they are analyzed as one dataset.

If we represent an original advection map as the observations O, and the reconstructed advection map as $\hat{O}$, we can compare the loss of information (relative error) from compression with:

$$error = \frac{\|\hat{O} - O\|_2}{\|O\|_2} \tag{D.1}$$

We performed this operation for all of the compressed data, which is shown as four separate heat maps (one for each Reynolds number, $Re = \{10, 20, 30, 40\}$) in Fig D.1. We found that over 98% of the reconstructed data had less than 5% error, with a mean reconstruction error of 1.76%, and a maximum error of 10%. In the case of the 10% error, this map was for $Re = 10$, and $D/w = 0.2$, making the displacement magnitudes quite small, and therefore making the (relative) error larger.

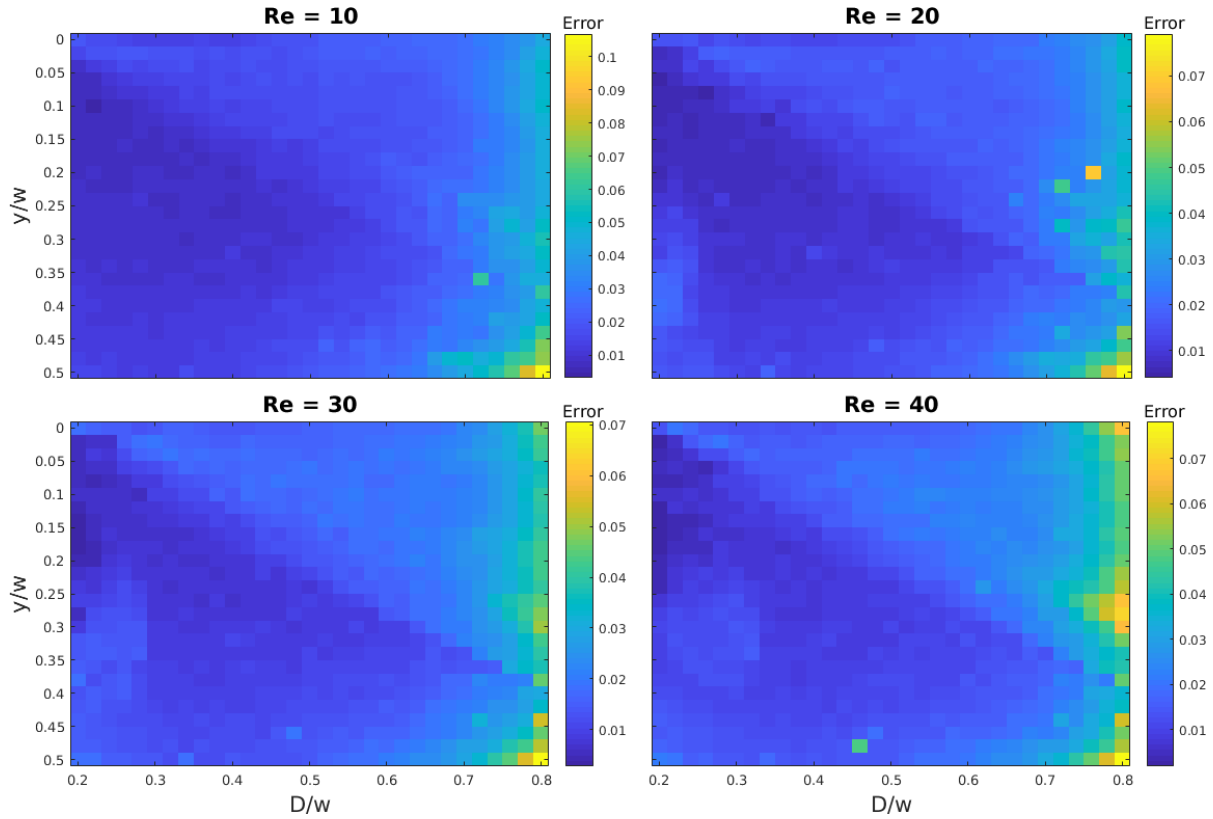

Figure D.1 Error plots for $Re = \{10, 20, 30, 40\}$ at a microchannel height/width aspect ratio $h/w = 0.25$ for pillar diameters $D/w = [0.2 : 0.02 : 0.8]$ and locations $y/w = [0.0 : 0.02 : 0.5]$.

BIBLIOGRAPHY

Ahuja, V., Hartfield, R. J., and Shelton, A. (2014). Optimization of hypersonic aircraft using genetic algorithms. *Applied Mathematics and Computation*, 242:423–434.

Akintayo, A., Lore, K. G., Sarkar, S., and Sarkar, S. Prognostics of combustion instabilities from hi-speed flame video using a deep convolutional selective autoencoder. *International Journal of Prognostics and Health Management*, 7 (2016).

Amini, H., Lee, W., and Di Carlo, D. (2014). Inertial microfluidic physics. *Lab on a Chip*, 14:2739–2761.

Amini, H., Sollier, E., Masaeli, M., Xie, Y., Ganapathysubramanian, B., Stone, H. A., and Di Carlo, D. (2013). Engineering fluid flow using sequenced microstructures. *Nature Communications*, 4(May):1826.

Amini, H., Sollier, E., Weaver, W. M., and Di Carlo, D. (2012). Intrinsic particle-induced lateral transport in microchannels. *Proceedings of the National Academy of Sciences*, 109(29):11593–11598.

Appleyard, D. C., Chapin, S. C., Srinivas, R. L., and Doyle, P. S. (2011). Bar-coded hydrogel microparticles for protein detection: synthesis, assay and scanning. *Nature Protocols*, 6(11):1761–1774.

Babbar, M. and Minsker, B. S. (2006). Groundwater Remediation Design Using Multiscale Genetic Algorithms. *Journal of Water Resources Planning and Management*, 132(5):341–350.

Bailey, M. R., Pentecost, A. M., Selimovic, A., Martin, R. S., and Schultz, Z. D. (2015). Sheath-flow microfluidic approach for combined surface enhanced Raman scattering and electrochemical detection. *Analytical Chemistry*, 87(8):4347–4355.

Baldi, P., Sadowski, P., and Whiteson, D. (2014). Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5:4308.

Bengio, Y., Courville, A., and Vincent, P. (2013). Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(8):1798–1828.

Bengio, Y., LeCun, Y., et al. (2007). Scaling learning algorithms towards ai. *Large-scale kernel machines*, 34(5).

Bhagat, A. A. S., Kuntaegowdanahalli, S. S., and Papautsky, I. (2008). Enhanced particle filtration in straight microchannels using shear-modulated inertial migration. *Physics of Fluids (1994-present)*, 20(10):–.

Blinn, J. F. (1977). Models of light reflection for computer synthesized pictures. *SIGGRAPH Comput. Graph.*, 11(2):192–198.

Bong, K. W., Kim, J. J., Cho, H., Lim, E., Doyle, P. S., and Irimia, D. (2015). Synthesis of Cell-Adhesive Anisotropic Multifunctional Particles by Stop Flow Lithography and Streptavidin-Biotin Interactions. *Langmuir*, 31(48):13165–13171.

Boyd, D. A., Shields, A. R., Howell, P. B., and Ligler, F. S. (2013). Design and fabrication of uniquely shaped thiol-ene microfibers using a two-stage hydrodynamic focusing design. *Lab Chip*, 13:3105–3110.

Bradski, G. (2000). Todo. *Dr. Dobb's Journal of Software Tools*.

Brooks, A. N. and Hughes, T. J. (1982). Streamline upwind/petrov-galerkin formulations for convection dominated flows with particular emphasis on the incompressible navier-stokes equations. *Computer methods in applied mechanics and engineering*, 32(1):199–259.

Chen, C. L., Mahjoubfar, A., Tai, L.-C., Blaby, I. K., Huang, A., Niazi, K. R., and Jalali, B. (2016). Deep Learning in Label-free Cell Classification. *Scientific reports*, 6(August 2015):21471.

Chiu, D. (2007). Cellular manipulations in microvortices. *Analytical and Bioanalytical Chemistry*, 387(1):17–20.

Chung, A. J., Pulido, D., Oka, J. C., Amini, H., Masaeli, M., and Di Carlo, D. (2013). Microstructure-induced helical vortices allow single-stream and long-term inertial focusing. *Lab on a chip*, 13(15):2942–9.

Ciftlik, A. T., Ettori, M., and Gijs, M. A. M. (2013). High throughput-per-footprint inertial focusing. *Small*, 9(16):2764–2773.

Coates, A., Ng, A. Y., and Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning. In *International Conference on Artificial Intelligence and Statistics*, pages 215–223.

Cormen, T. H., Stein, C., Rivest, R. L., and Leiserson, C. E. (2001). *Introduction to Algorithms*. McGraw-Hill Higher Education, 2nd edition.

Cortes-Quiroz, C. A., Zangeneh, M., and Goto, A. (2009). On multi-objective optimization of geometry of staggered herringbone micromixer. *Microfluidics and Nanofluidics*, 7(1):29–43.

Daniele, M. A., Boyd, D. A., Mott, D. R., and Ligler, F. S. (2015). 3D hydrodynamic focusing microfluidics for emerging sensing technologies. *Biosensors and Bioelectronics*, 67:25–34.

Deng, L. and Dong, Y. (2014). Foundations and trends in signal processing. *Signal Processing*, 7:3–4.

Di Carlo, D. (2009). Inertial microfluidics. *Lab on a Chip*, 9:3038–3046.

Di Carlo, D., Irimia, D., Tompkins, R. G., and Toner, M. (2007). Continuous inertial focusing, ordering, and separation of particles in microchannels. *Proceedings of the National Academy of Sciences*, 104(48):18892–18897.

Diaz-Montes, J., Xie, Y., Rodero, I., Zola, J., Ganapathysubramanian, B., and Parashar, M. (2014). Federated computing for the masses–aggregating resources to tackle large-scale engineering problems. *Computing in Science Engineering*, 16(4):62–72.

Do, A.-V., Khorsand, B., Geary, S. M., and Salem, A. K. (2015). 3D Printing of Scaffolds for Tissue Regeneration Applications. *Advanced Healthcare Materials*, 4(12):n/a–n/a.

Du, P., Weber, R., Luszczek, P., Tomov, S., Peterson, G., and Dongarra, J. (2012). From CUDA to OpenCL: Towards a performance-portable solution for multi-platform GPU programming. *Parallel Computing*, 38(8):391–407.

Even, S. and Goldreich, O. (1981). The minimum-length generator sequence problem is NP-hard. *Journal of Algorithms*, 2(3):311–313.

Fiorini, F., Prasetyanto, E. A., Taraballi, F., Pandolfi, L., Monroy, F., López-Montero, I., Tasciotti, E., and De Cola, L. (2016). Nanocomposite Hydrogels as Platform for Cells Growth, Proliferation, and Chemotaxis. *Small*, pages 4881–4893.

Foli, K., Okabe, T., Olhofer, M., Jin, Y., and Sendhoff, B. (2006). Optimization of micro heat exchanger: CFD, analytical approach and multi-objective evolutionary algorithms. *International Journal of Heat and Mass Transfer*, 49:1090–1099.

Frigo, M. and Johnson, S. G. (2005). The design and implementation of FFTW3. *Proceedings of the IEEE*, 93(2):216–231. Special issue on "Program Generation, Optimization, and Platform Adaptation".

Galisteo-López, J. F., Ibisate, M., Sapienza, R., Froufe-Peréz, L. S., Blanco, A., and López, C. (2011). Self-assembled photonic structures. *Advanced Materials*, 23(1):30–69.

Galoppo, N., Govindaraju, N. K., Henson, M., and Manocha, D. (2005). LU-GPU: Efficient algorithms for solving dense linear systems on graphics hardware. *Proceedings of the ACM/IEEE 2005 Supercomputing Conference, SC'05*, 2005(c).

Giles, M. B. and Pierce, N. A. (2000). An Introduction to the Adjoint Approach to Design. *Flow, Turbulence and Combustion*, 65:393–415.

Goda, K., Ayazi, a., Gossett, D. R., Sadasivam, J., Lonappan, C. K., Sollier, E., Fard, a. M., Hur, S. C., Adam, J., Murray, C., Wang, C., Brackbill, N., Di Carlo, D., and Jalali, B. (2012). High-throughput single-microparticle imaging flow analyzer. *Proceedings of the National Academy of Sciences*, 109(29):11630–11635.

Goldberg, D. (1989). *Genetic Algorithms in Search, Optimization, and Machine Learning*. Artificial Intelligence. Addison-Wesley.

Griffin, D. R., Weaver, W. M., Scumpia, P. O., Di Carlo, D., and Segura, T. (2015). Accelerated wound healing by injectable microporous gel scaffolds assembled from annealed building blocks. *Nature materials*, 14(7):737–744.

Guo, X., Li, W., and Iorio, F. (2016). Convolutional neural networks for steady flow approximation. In *Proceedings of the 22nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining*.

Gurkan, U. A., Tasoglu, S., Kavaz, D., Demirel, M. C., and Demirci, U. (2012). Emerging technologies for assembly of microscale hydrogels. *Advanced Healthcare Materials*, 1(2):149–158.

Hansson, J., Karlsson, J. M., Haraldsson, T., Brismar, H., van der Wijngaart, W., and Russom, A. (2012). Inertial microfluidics in parallel channels for high-throughput applications. *Lab on a Chip*, 12:4644–4650.

Hart, J. C. (1996). Sphere tracing: A geometric method for the antialiased ray tracing of implicit surfaces. *The Visual Computer*, 12(10):527–545.

Hinton, G. and Salakhutdinov, R. Reducing the dimensionality of data with neural networks. *Science*, 313(5786).

Holland, J. H. (1992). Genetic algorithms. *Scientific American*, 267:66–71.

Horn, B. P. (1986). *Robot Vision*. McGraw-Hill.

Howell, P. B., Golden, J. P., Hilliard, L. R., Erickson, J. S., Mott, D. R., and Ligler, F. S. (2008). Two simple and rugged designs for creating microfluidic sheath flow. *Lab on a chip*, 8(7):1097–1103.

Howell, P. B., Mott, D. R., Golden, J. P., and Ligler, F. S. (2004). Design and evaluation of a Dean vortex-based micromixer. *Lab on a chip*, 4(6):663–669.

Howell, Jr., P. B., Mott, D. R., Fertig, S., Kaplan, C. R., Golden, J. P., Oran, E. S., and Ligler, F. S. (2005). A microfluidic mixer with grooves placed on the top and bottom of the channel. *Lab Chip*, 5:524–530.

Hur, S. C., Henderson-MacLennan, N. K., McCabe, E. R. B., and Di Carlo, D. (2011). Deformability-based cell classification and enrichment using inertial microfluidics. *Lab on a Chip*, 11:912–920.

Ismagilov, R. F., Stroock, A. D., Kenis, P. J. a., Whitesides, G., and Stone, H. A. (2000). Experimental and theoretical scaling laws for transverse diffusive broadening in two-phase laminar flows in microchannels. *Applied Physics Letters*, 76(17):2376.

Ivorra, B., Hertzog, D. E., Mohammadi, B., and Santiago, J. G. (2006). Semi-deterministic and genetic algorithms for global optimization of microfluidic protein-folding devices. *International Journal for Numerical Methods in Engineering*, 66(2):319–333.

Jaeger, R., Ren, J., Xie, Y., Sundararajan, S., Olsen, M., and Ganapathysubramanian, B. (2012). Nanoscale surface roughness affects low reynolds number flow: Experiments and modeling. *Applied Physics Letters*, 101(18):184102.

John Aldo Lee, M. V. (2007). *Nonlinear dimensionality reduction*. Springer.

Kalivarapu, V. K. and Winer, E. H. (2010). Performance of Hardware Accelerated Particle Swarm Optimization with Digital Pheromones on Dissimilar Computing Platforms. *13th AIAAISSMO Multidisciplinary Analysis Optimization Conference*, pages 1–19.

Kim, D. S., Jung, D. H., and Kim, Y. Y. (2008). Multiscale Multiresolution Genetic Algorithm with a Golden Sectioned Population Composition. *International Journal for Numerical Methods in Engineering*, 74(3):349–367.

Krizhevsky, A., Sutskever, I., and Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In *NIPS*.

Larochelle, H. and Bengio, Y. (2008). Classification using discriminative restricted boltzmann machines. In *Proceedings of the 25th international conference on Machine learning*, pages 536–543. ACM.

Lee, J., Bisso, P. W., Srinivas, R. L., Kim, J. J., Swiston, A. J., and Doyle, P. S. (2014). Universal process-inert encoding architecture for polymer microparticles. *Nature Materials*, 13(5):524–529.

Lee, Y.-T., Ahuja, V., Hosangadi, A., and Ebert, M. (2010). Shape Optimization of a Multi-Element Foil Using an Evolutionary Algorithm. *ASME J. Fluids Eng.*, 132(5):051401.

Lepchev, D. and Weihs, D. (2010). Low Reynolds Number Flow in Spiral Microchannels. *ASME J. Fluids Eng.*, 132(7):071202.

Liu, R. H., Stremler, M. a., Sharp, K. V., Olsen, M. G., Santiago, J. G., Adrian, R. J., Aref, H., and Beebe, D. J. (2000). Passive Mixing in a Three-dimensional Serpentine Microchannel. *Journal of Microelectromechanical Systems*, 9(2):190–197.

Lore, K. G., Akintayo, A., and Sarkar, S. (2017). LLNet: A deep autoencoder approach to natural low-light image enhancement. *Pattern Recognition*, 61:650–662.

Lore, K. G., Stoecklein, D., Davies, M., Ganapathysubramanian, B., and Sarkar, S. (2015). Hierarchical feature extraction for efficient design of microfluidic flow patterns. In *Proceedings of The 1st International Workshop on Feature Extraction: Modern Questions and Challenges, NIPS*, pages 213–225.

Lore, K. G., Sweet, N., Kumar, K., Ahmed, N., and Sarkar, S. (2016). Deep value of information estimators for collaborative human-machine information gathering. *International Conference on Cyber-physical Systems (ICCPS). Vienna, Austria.*

Lu, M., Ozcelik, A., Grigsby, C. L., Zhao, Y., Guo, F., Leong, K. W., and Huang, T. J. (2016). Microfluidic hydrodynamic focusing for synthesis of nanomaterials. *Nano Today*, 11(6):778–792.

Lyutov, A. E., Chirkov, D. V., Skorospelov, V. A., Turuk, P. A., and Cherny, S. G. (2015). Coupled Multipoint Shape Optimization of Runner and Draft Tube of Hydraulic Turbines. *ASME J. Fluids Eng.*, 137(11):111302.

Mach, A. J. and Di Carlo, D. (2010). Continuous scalable blood filtration device using inertial microfluidics. *Biotechnology and Bioengineering*, 107(2):302–311.

Mao, X., Waldeisen, J. R., Juluri, B. K., and Huang, T. J. (2007). Hydrodynamically tunable optofluidic cylindrical microlens. *Lab Chip*, 7(10):1303–1308.

Matsumoto, M. and Nishimura, T. (1998). Mersenne twister: A 623-dimensionally equidistributed uniform pseudo-random number generator. *ACM Trans. Model. Comput. Simul.*, 8(1):3–30.

Milano, M. and Koumoutsakos, P. (2002). A Clustering Genetic Algorithm for Cylinder Drag Optimization. *Journal of Computational Physics*, 175(1):79–107.

Mohammadi, B. and Pironneau, O. (2004). Shape Optimization in Fluid Mechanics. *Annual Review of Fluid Mechanics*, 36(1):255–279.

Mott, D. R., Howell, P. B., Obenschain, K. S., and Oran, E. S. (2009). The Numerical Toolbox: An approach for modeling and optimizing microfluidic components. *Mechanics Research Communications*, 36(1):104–109.

Mott, D. R., Howell, Jr., P. B., Golden, J. P., Kaplan, C. R., Ligler, F. S., and Oran, E. S. (2006). Toolbox for the design of optimized microfluidic components. *Lab on a Chip*, 6(4):540.

Müller, S. D., Mezić, I., Walther, J. H., and Koumoutsakos, P. (2004). Transverse momentum micromixer optimization with evolution strategies. *Computers & Fluids*, 33(4):521 – 531.

Neto, F. D. M. and Neto, A. J. d. S. (2013). *An Introduction to Inverse Problems with Applications*. Springer.

Nunes, J. K., Wu, C. Y., Amini, H., Owsley, K., Di Carlo, D., and Stone, H. A. (2014). Fabricating shaped microfibers with inertial microfluidics. *Advanced Materials*, 26:3712–3717.

Oakey, J., Applegate, R. W., Arellano, E., Di Carlo, D., Graves, S. W., and Toner, M. (2010). Particle focusing in staged inertial microfluidic devices for flow cytometry. *Analytical Chemistry*, 82(9):3862–3867.

Owens, J. D., Luebke, D., Govindraju, N., Harris, M., Kruger, J., Lefohn, A. E., and Purcell, T. J. (2006). A Survey of General Purpose Computation on Graphics Hardware. *Computer Graphics Forum*, 26(1):80–113.

Paulsen, K. S. and Chung, A. J. (2016). Non-Spherical Particle Generation from 4D Optofluidic Fabrication. *Lab on a Chip*.

Paulsen, K. S., Di Carlo, D., and Chung, A. J. (2015). Optofluidic fabrication for 3D-shaped particles. *Nature Communications*, 6:6976.

Pratt, W. K. (1991). *Digital Image Processing*. John Wiley & Son.

Rabitz, H., Alis, Ö., and Al, Ö. F. (1999). General foundations of high-dimensional model representations. *J. Math. Chem.*, 25(2-3):197–233.

Ravindran, P. (2017). Efy dvd: Openscad: Solid 3d modeller for programmers. *Electronics For You*.

Sadowski, P. J., Whiteson, D., and Baldi, P. (2014). Searching for higgs boson decay modes with deep learning. In *Advances in Neural Information Processing Systems*, pages 2393–2401.

Saito, M. and Matsumoto, M. (2008). SIMD-Oriented Fast Mersenne Twister: a 128-bit Pseudorandom Number Generator. In Keller, A., Heinrich, S., and Niederreiter, H., editors, *Monte Carlo and Quasi-Monte Carlo Methods 2006*, pages 607–622. Springer Berlin Heidelberg.

Salakhutdinov, R. and Hinton, G. (2009). Replicated softmax: An undirected topic model. *Advances in Neural Information Processing Systems 22 - Proceedings of the 2009 Conference*, pages 1607–1614.

Salakhutdinov, R., Mnih, A., and Hinton, G. (2007). Restricted Boltzmann Machines for Collaborative Filtering. *Proceedings of the 24th ACM International Conference on Machine Learning (ICML)*, pages 791–798.

Segré, G. and Silberberg, A. (1961). Radial Particle Displacements in Poiseuille Flow of Suspensions. *Nature*, 189(4760):209.

Selmi, M., Echouchene, F., Gazzah, M. H., and Belmabrouk, H. (2015). Flow confinement enhancement of heterogeneous immunoassays in microfluidics. *IEEE Sensors Journal*, 15(12):7321–7328.

Sharifi, F., Patel, B. B., Dzuilko, A. K., Montazami, R., Sakaguchi, D. S., and Hashemi, N. (2016). Polycaprolactone Microfibrous Scaffolds to Navigate Neural Stem Cells. *Biomacromolecules*, 17(10):3287–3297.

Singh, A., Ganapathysubramanian, B., Singh, A. K., and Sarkar, S. (2016). Machine learning for high-throughput stress phenotyping in plants. *Trends in plant science*, 21(2):110–124.

Sobol, I. M., Asotsky, D., Kreinin, A., and Kucherenko, S. (2011). Construction and comparison of high-dimensional sobol generators. *Wilmott Journal*, page 6479.

Sollier, E., Amini, H., Go, D., Sandoz, P., Owsley, K., and Di Carlo, D. (2015). Inertial Microfluidic Programming of Microparticle-laden Flows for Solution Transfer Around Cells and Particles. *Microfluidics and Nanofluidics*, 19(1):53–65.

Sollier, E., Murray, C., Maoddi, P., and Di Carlo, D. (2011). Rapid prototyping polymers for microfluidic devices and high pressure injections. *Lab on a Chip*, 11(22):3752.

Spiga, M. and Morino, G. L. (1994). A symmetric solution for velocity profile in laminar flow through rectangular ducts. *International Communications in Heat and Mass Transfer*, 21(4):469–475.

Squires, T. M. and Quake, S. R. (2005). Microfluidics: Fluid physics at the nanoliter scale. *Reviews of Modern Physics*, 77(3):977–1026.

Srinivasa Reddy, B. and Chatterji, B. N. (1996). An FFT-based technique for translation, rotation, and scale-invariant image registration. *IEEE Transactions on Image Processing*, 5(8):1266–1271.

Stoecklein, D., Davies, M., Wubshet, N., Le, J., and Ganapathysubramanian, B. (2017a). Automated design for microfluid flow sculpting: multiresolution approaches, efficient encoding, and CUDA implementation. *ASME Journal of Fluids Engineering*, 139(3).

Stoecklein, D., Lore, K. G., Davies, M., Sarkar, S., and Ganapathysubramanian, B. (2017b). Deep Learning for Flow Sculpting: Insights into Efficient Learning using Scientific Simulation Data. *Scientific Reports*, 7(April):46368.

Stoecklein, D., Wu, C.-Y., Kim, D., Di Carlo, D., and Ganapathysubramanian, B. (2016). Optimization of micropillar sequences for fluid flow sculpting. *Physics of Fluids*, 28(1):1–21.

Stoecklein, D., Wu, C.-Y., Owsley, K., Xie, Y., Di Carlo, D., and Ganapathysubramanian, B. (2014). Micropillar sequence designs for fundamental inertial flow transformations. *Lab on a Chip*, 14(21):4197–204.

Stone, H., Stroock, A., and Ajdari, A. (2004). Engineering flows in small devices. *Annual Review of Fluid Mechanics*, 36(1):381–411.

Stroock, A. D., Dertinger, S. K. W., Ajdari, A., Mezic, I., Stone, H. a., and Whitesides, G. M. (2002). Chaotic Mixer for Microchannels. *Science (New York, N.Y.)*, 295(January):647–651.

Sudarsan, A. P. and Ugaz, V. M. (2006). Multivortex micromixing. *Proceedings of the National Academy of Sciences of the United States*, 103(19):7228.

Tezduyar, T. E., Mittal, S., Ray, S., and Shih, R. (1992). Incompressible flow computations with stabilized bilinear and linear equal-order-interpolation velocity-pressure elements. *Computer Methods in Applied Mechanics and Engineering*, 95(2):221–242.

Theano Development Team (2016). Theano: A Python framework for fast computation of mathematical expressions. *arXiv e-prints*, abs/1605.02688.

Uspal, W. E., Burak Eral, H., and Doyle, P. S. (2013). Engineering particle trajectories in microfluidic flows using particle shape. *Nature communications*, 4:2666.

Vogel, C. (2002). *Computational Methods for Inverse Problems*. Society for Industrial and Applied Mathematics.

Wang, Y., He, Y.-L., Mei, D.-H., and Tao, W.-Q. (2011). Optimization design of slotted fin by numerical simulation coupled with genetic algorithm. *Applied Energy*, 88(12):4441 – 4450.

Wang, Y., Zhe, J., Dutta, P., and Chung, B. T. (2007). A Microfluidic Mixer Utilizing Electrokinetic Relay Switching and Asymmetric Flow Geometries. *ASME J. of Fluids Eng.*, 129(4):395.

Wu, C.-Y., Owsley, K., and Di Carlo, D. (2015). Rapid Software-Based Design and Optical Transient Liquid Molding of Microparticles. *Advanced Materials*, pages n/a–n/a.

Xie, Y., Wodo, O., and Ganapathysubramanian, B. (2016). Incompressible two-phase flow: Diffuse interface approach for large density ratios, grid resolution study, and 3D patterned substrate wetting problem. *Computers and Fluids*, 141:223–234.

Xue, Y., Zhai, Z. J., and Chen, Q. (2013). Inverse prediction and optimization of flow control conditions for confined spaces using a cfd-based genetic algorithm. *Building and Environment*, 64(0):77 – 84.

Yang, Y.-T., Wang, Y.-H., and Tseng, P.-K. (2014). Numerical optimization of heat transfer enhancement in a wavy channel using nanofluids. *International Communications in Heat and Mass Transfer*, 51:9–17.

Yuster, R. and Zwick, U. (2005). Fast sparse matrix multiplication. *ACM Transactions on Algorithms*, 1(1):2–13.

Zeiler, M. D. and Fergus, R. (2014). Visualizing and Understanding Convolutional Networks. *ECCV*.

Zeng, H., Edwards, M. D., Liu, G., and Gifford, D. K. (2016). Convolutional neural network architectures for predicting dna–protein binding. *Bioinformatics*, 32(12):i121–i127.

Zhang, J., Yan, S., Yuan, D., Alici, G., Nguyen, N.-T., Ebrahimi Warkiani, M., and Li, W. (2016). Fundamentals and Applications of Inertial Microfluidics: A Review. *Lab Chip*, 16:10–34.

Zhou, T., Xu, Y., Liu, Z., and Joo, S. W. (2015). An Enhanced One-Layer Passive Microfluidic Mixer with an Optimized Lateral Structure with the Dean Effect. *ASME J. Fluids Eng.*, 137(c):1–7.