Learning Three-Dimensional Flow for Interactive Aerodynamic Design

NOBUYUKI UMETANI, Autodesk Research
BERND BICKEL, IST Austria

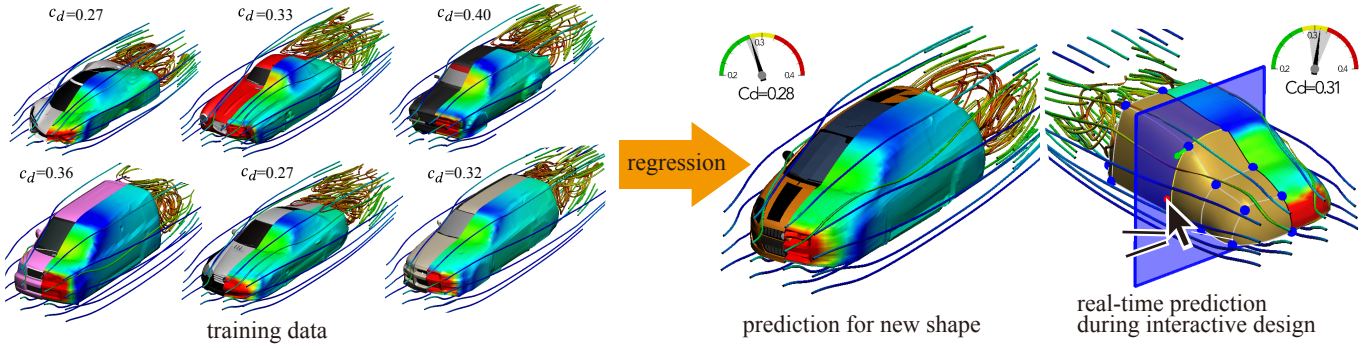

Fig. 1. Our interactive design system allows users to interactively design and optimize a free-form 3D shape while the pressure on the surface (color-coded) and the velocity field (illustrated as stream lines) are accurately predicted in real-time. Our data-driven model is based on a novel representation for 3D shapes and can be robustly trained from a set of exemplars.

We present a data-driven technique to instantly predict how fluid flows around various three-dimensional objects. Such simulation is useful for computational fabrication and engineering, but is usually computationally expensive since it requires solving the Navier-Stokes equation for many time steps. To accelerate the process, we propose a machine learning framework which predicts aerodynamic forces and velocity and pressure fields given a three-dimensional shape input. Handling detailed free-form three-dimensional shapes in a data-driven framework is challenging because machine learning approaches usually require a consistent parametrization of input and output. We present a novel PolyCube maps-based parametrization that can be computed for three-dimensional shapes at interactive rates. This allows us to efficiently learn the nonlinear response of the flow using a Gaussian process regression. We demonstrate the effectiveness of our approach for the interactive design and optimization of a car body.

CCS Concepts: • **Computing methodologies** → **Gaussian processes**; *Real-time simulation*; *Physical simulation*; • **Applied computing** → *Engineering*;

Additional Key Words and Phrases: machine learning, fluid simulation, Gaussian process, parameterization

ACM Reference Format:
Nobuyuki Umetani and Bernd Bickel. 2018. Learning Three-Dimensional Flow for Interactive Aerodynamic Design. *ACM Trans. Graph.* 37, 4, Article 89 (August 2018), 10 pages. https://doi.org/10.1145/3197517.3201325

Authors' addresses: Nobuyuki Umetani, Autodesk Research, Toronto, Canada, n.umetani@gmail.com; Bernd Bickel, IST Austria, Vienna, Austria, bernd.bickel@ist.ac.at.

1 INTRODUCTION

Computational Fluid Dynamics (CFD) has become an indispensable component in the field of engineering and computational fabrication for design evaluation and optimization. Successful applications include improving the aerodynamics of cars and airplanes, refining heating, ventilation and air conditioning in architectural planning, or the optimization of casting and injection molding in manufacturing. Many of these applications would benefit from an interactive design approach. However, analyzing a design is usually computationally expensive, as it requires repetitively solving a system with many degrees of freedom over many time steps. Although modern design tools allow the user to edit the shape interactively, computation times often prohibit the interactive visualization, exploration, and optimization of shapes and their surroundings based on CFD simulations. This problem is extremely challenging, even when real-time simulation capabilities are available, i.e., the simulation of individual time-steps at real-time rates, because evaluating a design in simulation may require many time steps.

In this paper, we suggest a novel approach to achieve real-time CFD performance based on a *data-driven* method. While CFD generally refers to fluid simulation for engineering purposes, we limit the scope to the evaluation of aerodynamic forces and flow around an object given that it is moving at a constant speed in a single-phase fluid (e.g., air flow around cars or airplanes). Our approach enables drastically faster evaluation by constructing a reduced model between input and output based on pre-existing data. Our model takes a 3D shape as the input, and outputs a drag coefficient, fluid pressure on the surface and velocity field around the shape. The nonlinearly and globally coupled relationship between input and output is learned using the Gaussian Process (GP).

A key challenge in applying machine learning to CFD problems is the parameterization of the input and output. As most machine learning approaches take fixed dimensional vectors as input and

output, we need to parameterize (i.e., represent as fixed dimensional variables) the input 3D shape and output pressure field on the shape and velocity field around the shape. To address this problem, we present a novel approach for parameterizing the 3D shape and volumetric field around it for machine learning based on the PolyCube map [Tarini et al. 2004]. The PolyCube map represents arbitrary 3D shapes by deforming a set of cubes. By deforming the space around the cubes, we also achieve a natural parameterization of the field around arbitrary 3D shapes. Our PolyCube-based shape and field parameterization is extremely efficient and can be done in real-time with a small distortion.

Our proposed representation features a fixed-length parameterization of the shape and field, which allows a GP to easily learn and predict their relationship. As a demonstration, we constructed an interactive aerodynamic design tool where the user can edit a shape while getting real-time feedback on pressure distribution on the surface and the air flow around the shape. Our tool also visualizes reliability of the prediction as machine learning is unreliable when input is very different from ones in the training data. The main contributions of this work are:

- A novel, computationally efficient parameterization of 3D shapes and volumetric fields based on the PolyCube map.
- A regression framework for three-dimensional flow around an object.
- An interactive interface for aerodynamically efficient design.

2 RELATED WORK

Reduction techniques for fluid simulation. Model reduction techniques perform simulations in a lower dimensional representation of the original simulation domain, thereby decoupling the grid resolution from the simulation complexity. In the computer graphics community, Treuille et al. [2006] introduced model reduction for the real-time simulation of incompressible fluids. Later, this technique was extended to modular tiles that can be assembled during the runtime to adapt the domain and simulate novel fluid configurations [Wicke et al. 2009]. For moving solids, model reduction can also be conducted on a moving grid [Cohen et al. 2010]. To enable the reduced simulation of fluid flow around a deformable object, Stanton et al. [2013] suggest a nonlinear Galerkin projection approach to construct nonlinear bases. The subspace integration is accelerated by the optimized cubature scheme [Kim and Delaney 2013]. Eigenmodes of the Laplacian operator are used to construct bases of the reduced model [De Witt et al. 2012]. Since a model reduction approach solves the Navier-Stokes equation step-by-step, it requires the running of a simulation for some time to obtain the overall fluid behavior. In contrast, our approach directly learns the results, circumventing the use of a Navier-Stokes solver during the runtime. Stanton et al. [2014] use a state graph to synthesize animated fluid motion from examples, but it is unclear how this can be applied to a high-dimensional input such as a free-form shape.

Machine learning for fluid simulation. Recently, accelerating fluid simulations using machine learning has gained significant attention Tompson et al. [2016] accelerate the pressure projection in Eulerian fluid simulations using a neural network. Forces on Lagrangian particles can be efficiently approximated using a regression forest [Ladický et al. 2015]. Detailed splashing effects can be added to Lagrangian particles using neural networks to model the regression of splash formation [Um et al. 2017]. Chu et al. [2017] use a Siamese network to learn a distance metric between fine and coarse simulation to synthesize the details of smoke. Bonev et al. [2017] suggest a data-driven technique to map changing simulation conditions such as user interactions to a reduced representation based on space-time deformations. In addition, a network is trained to refine the deformations and represent details present in the initial training data. While these approaches either accelerate the computation of individual time steps or add detail as a post-process, our approach directly generates a time-averaged velocity field and pressure field.

Data-driven aerodynamics. There are few studies that use machine learning to predict aerodynamic forces for real-world applications. Pteromys [Umetani et al. 2014] constructs a relationship between paper airplane shapes and aerodynamic forces based on recorded flight trajectories of a large set of airplanes. The proposed model is limited to assemblies of free-form planar shapes. Addressing this limitation, OmniAD [Martin et al. 2015] models aerodynamic forces on an object from arbitrary directions using spherical harmonics and learns model parameters from falling motions captured with a single camera. It also features the prediction of model parameters of low-dimensional parameterized shapes. Schultz et al. [2017] present a sampling scheme to capture nonlinear simulation response by interpolating samples in the a low-dimensional parameterized design space. Baqué et al. [2018] predict pressure distribution on free-form 3D objects using graph convoluation. Our approach learns aerodynamic forces of various free-form 3D objects, in addition to the time-averaged velocity field around the object.

3D shapes in machine learning. When feeding a 3D shape into a machine learning framework, a central question is how to parameterize the shape, i.e., how to express it as a set of variables. Parameterization of free-form 3D shapes is a non-trivial task as the shapes can have different resolutions, orientations, structures, and topologies. This is significantly different from other domains, such as natural language processing or vision, where text or image datasets frequently come with a common parameterization [Xu et al. 2016]. For measuring shape similarity, many shape descriptors, such as 3D shape histograms, were proposed. However, since the representation only parameterizes the shape as a distribution of features, it loses a significant amount of information concerning the original shape, making it unsuitable for our regression problem.

Recently, a number of deep neural network architectures have been proposed for 3D objects. These explore multiple views of a 3D shape [Su et al. 2015] or voxel-based representations [Wang et al. 2017; Wu et al. 2014] as inputs. Instead of using descriptors, these approaches take a more direct representation of a three-dimensional shape as an input. However, these implicit shape representation approaches are not desirable for our regression problem since parameters change discontinuously with respect to the continuous shape change. PointNet [Qi et al. 2016] explicitly encodes the 3D geometry using a set of randomly distributed points on the surface. However, it is difficult to discretize volumetric fields consistently with such a grid-less point-based discretization.

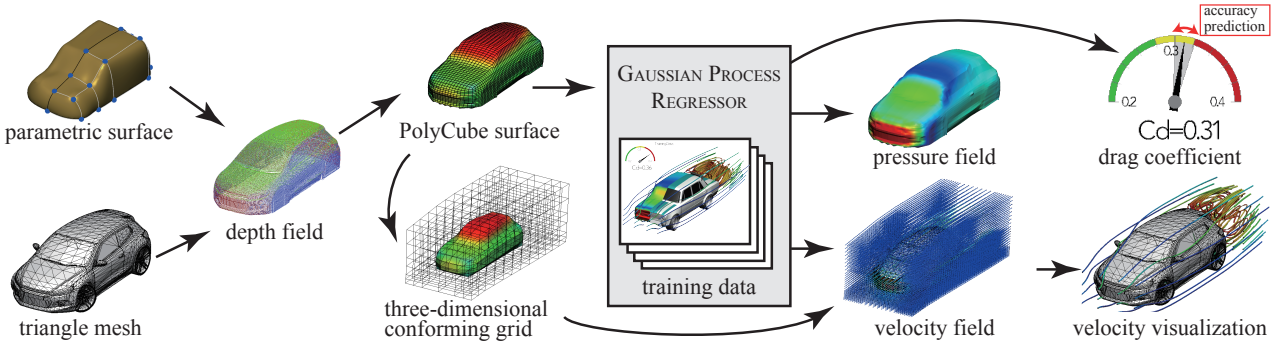

Fig. 2. Overview of our method.

Addressing this problem, our shape representation is based on deforming a template mesh. While Umetani [2017] deforms a cube surface to parameterize a 3D shape, we further extend this approach by deforming a PolyCube. The PolyCube approach allows us to parameterize a wider variation of shapes (see Fig. 10) and to construct a hexahedral mesh outside the shape (see Fig. 5). As years of studies show, PolyCube mapping is suitable to parameterize a wide range of three-dimensional shapes [Fang et al. 2016; Huang et al. 2014; Tarini et al. 2004]. While these works focus on offline parameterization quality optimization, our focus is real-time computation for generating machine learning input shapes and output fields.

3 OVERVIEW

User interface. Our machine learning framework takes a shape $\mathcal{S}$ and outputs drag coefficient c_d, a velocity field $\vec{v}$ around the shape and a pressure field p on the shape. The drag coefficient shows the magnitude of drag force with respect to the projected frontal area and is the most important indicator of aerodynamic efficiency. The drag coefficient c_d takes a value between 0.2 and 0.6 for typical cars (e.g., 0.57 for a Hummer H2 and 0.30 for a Toyota Corolla). The predicted drag coefficient is shown with a possible error range because the prediction via machine learning is not perfect for unknown inputs (see Fig. 2-top rightmost). These predicted values are time-averaged as they dynamically fluctuate over time. The pressure field is visualized as a color contour on the object's surface and the velocity field is visualized with streamlines.

We do not have specific restrictions on the input shape representation, which we demonstrate using two popular shape representations: triangle meshes and boundary representation (B-Rep) models, whose surfaces are defined by patches of parametric surfaces. These shapes can be edited freely with standard shape editing operations (e.g., free-form deformation, changing the position of control points, etc.). Topology change is allowed in the input, i.e., the user can change the topology of the B-Rep by inserting an edge inside a patch or by adding a control point to an edge. During the editing, the tool continuously updates the visualizations.

Computational pipeline. Figure 2 illustrates an overview of the runtime computation. We first convert the input shape into a set of depth images that are taken from multiple orthogonal orientations. From the depth images, we construct a PolyCube-based representation (see Sec. 4), to parameterize the shape and velocity and pressure fields. The pressure regressor estimates pressures at the grid points on the shape and the velocity regressor estimates velocity at grid points around the shape. We construct such regressors based on the Gaussian Process (see Sec. 5).

4 PARAMETERIZATION OF 3D SHAPE AND FIELD

4.1 Shape Parameterization for the Regression Problem

In theory, state-of-the-art machine learning techniques such as deep neural networks have the capability to approximate arbitrary multivariable functions [Hornik 1991]. However, in practice, a huge amount of training data is required to represent a high-dimensional highly nonlinear function while avoiding over-fitting. How well the regression predicts outputs for previously unseen data (i.e., the generalization error) depends heavily on the properties of the input and output data. Three factors are usually desired:

(fix) The input and output vectors for regression should have *fixed dimensions*.

(compact) The dimension of the input and output vectors should be *as small as possible* to avoid redundancy.

(linear) The relationship between input and output should be *as linear as possible* to avoid complicated models, which are difficult to train from a small amount of data.

Table 1. Comparison of shape and field parameterizations.

		(fix)	(compact)	(linear)
3D SHAPE	***our parameterization***	○	○	○
	B-Rep	×	○	○
	triangle mesh	×	○	○
	multi-view projection	○	×	×
	voxel	○	×	×
	SDF on a Cartesian grid	○	×	○
VOLUMETRIC FIELD	***our parameterization***	○	○	○
	tetrahedra mesh	×	○	○
	values on a Cartesian grid	○	○	×

Unfortunately, these desirable features are difficult to satisfy simultaneously. Table 1 compares common parameterization (representation) approaches of a 3D shape and field. Most shape editing tools operate with boundary representation (B-Rep) models or triangle meshes. However, the number of parameters changes as the topology of these representations is modified; for example, by inserting new edges and points or by subdividing or reconnecting the triangle mesh. Representing shapes using density values on multi-view projections or a three-dimensional Cartesian grid approach alleviates this problem, but induces significant nonlinearity, making the regression difficult. As illustrated in Figure 3-top, a continuous change in the shape results in abrupt changes of the velocity and density at grid points which are passed by the boundary. The volumetric and surface field representations suffer from a similar nonlinearity problem when parameterized on a Cartesian grid. Parameterizing the shape using the signed distance field (SDF) instead of a density function reduces this problem. However, the dimension of the parameters is still large as we express the shape using a 3D field, even though the input shape is locally two-dimensional. Representing the volumetric field with a tetrahedral mesh for various shapes is difficult without changing its topology.

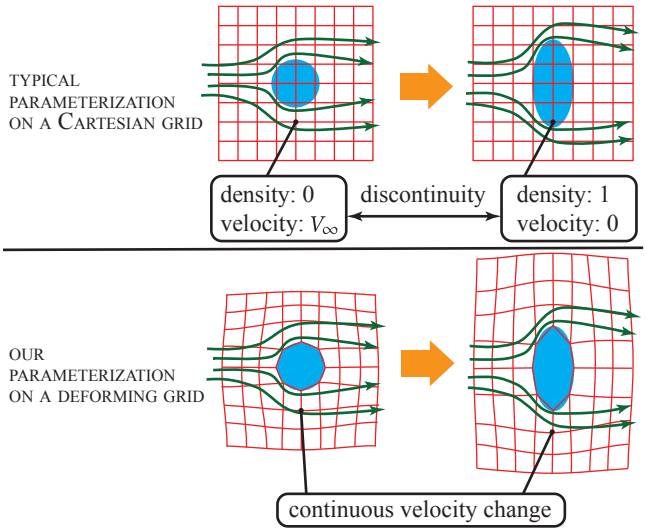

Fig. 3. Top: Small changes to a parameterized shape and field on a Cartesian grid induce discontinuities. Bottom: We deform the grid continuously to reduce nonlinearities.

Our solution is to deform the grid continuously on and around the shape (see Figure 3-bottom). The deformed grid shares the interface with the input shape. Hence, the boundary does not cross grid points, reducing the nonlinearity in the parameterization. The deformation of the entire volumetric grid is uniquely determined by the surface grid. Hence, if we specify the surface grid, we can define the volumetric field just by giving the values at the grid points. In the following section, we explain how we construct such a grid based on the PolyCube map.

4.2 Construction of the PolyCube Grid

The parametrization of the 3D shape is based on the work [Umetani 2017] in which we convert the input shape into GPU-computed depth surfaces and we then project the template mesh in the static normal directions to the depth surface. The key difference is the use of PolyCube instead of a single cube for the template mesh to parameterize a wider range of shapes by choosing the PolyCube close to the inputs. In this paper, we define a PolyCube as a set of axis-aligned unit cubes that are connected face-to-face. Such a PolyCube is manually modeled with an interface similar to MineCraft [Mojang 2009]. Aside from continuous shape and field representation, another important advantage of the cube representation is that we can easily construct a grid to parameterize the field outside the input shape by filling additional cubes outside the input cubes (see Figure 4-a). Different to the original PolyCube map [Tarini et al. 2004], we hierarchically construct the mapping by projecting the mesh in the normal direction similar to the Normal Mesh [Guskov et al. 2000]. This allows us parameterizing a concave shape with a lower number of variables — instead of storing three-dimensional coordinate values for all the vertices we store a scalar height map — and fewer parameters come in handy for machine learning. We also show our hierarchical construction of the parameterization benefits the reguralization of the machine learning (see Sec. 5.3).

Point classification. We start with a single, user-provided PolyCube model as an approximation of the various input shapes. We name the grid generated from an input and exterior cube as a level-0 grid. The level-0 grid has three types of points: corner points $\mathcal{P}_c$, surface points $\mathcal{P}_s$ and exterior points $\mathcal{P}_e$. These points are classified based on how many input cubes belong to the points (i.e., the valence). If the valence is an odd number, the point is a corner point that is located at the corner of the input PolyCube. On the other hand, the exterior points have a valence of zero as they do not belong to the input cube. If the valence is non-zero and an even number, since we do not construct a grid inside the shape, the point is a surface point that is located on an edge or face of the input PolyCube. For corner and surface points, we define the normal directions by averaging the normals of the input PolyCube faces where these points belong.

Corner point placement. We first project the corner points onto the object's surface. We start initializing the location of corner points by translating and scaling the input PolyCube in the axes directions such that the bounding boxes become the same. Then, we project the corner points onto the input shape by finding the nearest intersection point.

Surface and exterior point placement. Given the positions of the corner points, we move the surface points and exterior points to the locations linearly interpolated from the corner points (see Figure 4-d). For the surface points, which are located on the faces or the edges of the input PolyCube, we apply two-dimensional Mean Value Coordinates (MVC) [Floater 2003] to move them together with the corner points. As a consequence, the surface points on the edge of the PolyCube move as the linear interpolation of the two endpoints with the ratio of their distances. A surface on a face of the input PolyCube moves linearly with respect to the corner points that

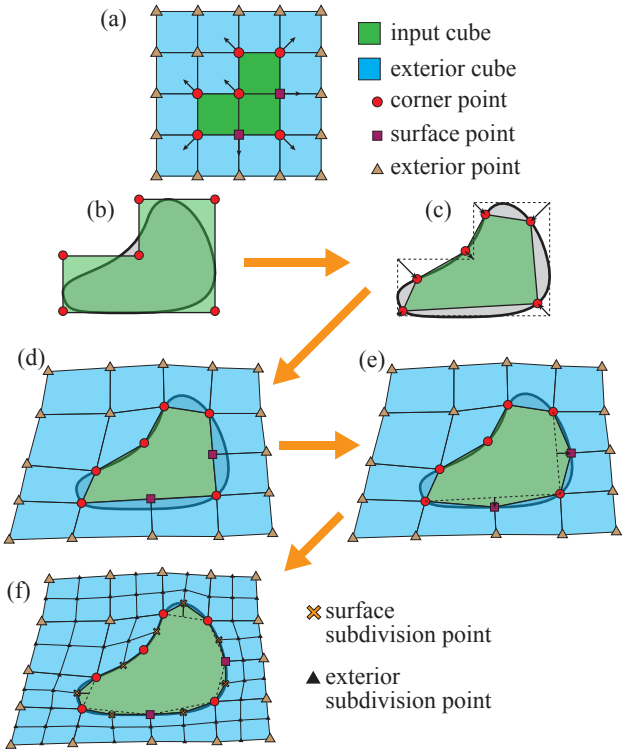

Fig. 4. Our construction scheme of a PolyCube parameterization.

outline the face. For exterior points, we use the 3D extension of MVC [Ju et al. 2005] to move the points linearly with respect to the position of all the corner points. The MVC determines the weight that transfers the movement of corner point p_c to a movement of an exterior point and the surface points. These weights are dependent only on the initial PolyCube construction, thus we only need to compute them once. After the linear interpolation, we project the surface point onto the object surface. The set of projection heights $\mathbf{H}_s$ for all the surface points specifies how much these points are moved toward the normal direction, encoding the shape at the coarsest level together with the positions of the corner points $\vec{\mathbf{P}}_c$.

Subdivision. So far, we have constructed a coarse grid such that the boundary between the input and exterior cubes is shared with the input shape. We subdivide the exterior cube to construct further detailed parameterization of the input shape and the output field (see Fig. 4-f). Although we can selectively choose which exterior cubes are subdivided, in this paper, we uniformly subdivide the exterior cubes such that all cubes have eight children. New points are introduced at the middle points of the edges, the center of the four corners of the faces, and the cell center of the parent grid. If a face or edge subdivision point is on the boundary facing the interior, we project it to the input shape in its normal direction. The normal direction is computed as the normal on the input PolyCube at the corresponding location. We denote the normal projection heights at the surface subdivision points as $\mathbf{H}_{ss}^{i}$, where i is the subdivision level. Fig. 5 shows an example of the subdivision.

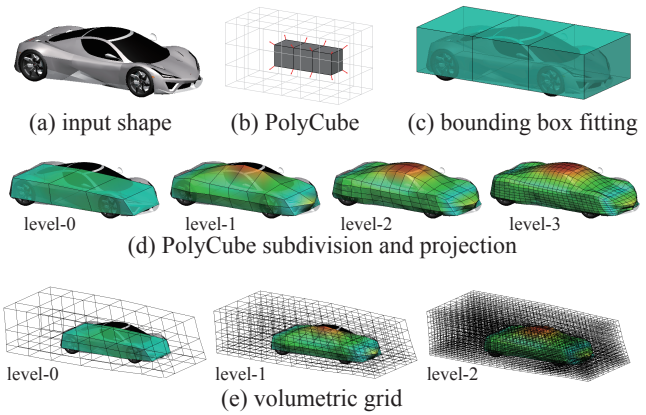

Fig. 5. Three-dimensional example of the subdivision of a level-0 grid. The color visualizes the accumulated normal projection heights up to the current level.

5 MACHINE LEARNING

5.1 Data for Regression

So far, we have explained our parameterization framework that encodes shape and field in fixed dimensional vectors while also avoiding excessive nonlinearities between them. In this section, we explain how we construct the regressor to approximate the output parameter vector from the input parameter vector.

Dimensional analysis. We start with four parameters as inputs: shape $\mathcal{S}$, incoming wind velocity V_∞, mass density ρ, and viscosity μ. We first reduce these four parameters into two using dimensional analysis. The dimensional analysis determines how physical phenomena can be parameterized with a minimum number of independent non-dimensional variables. Non-dimensionalization is convenient for machine learning of simulations because it reduces the number of parameters without loss of accuracy. Furthermore, we can reduce the amount of training data by avoiding redundant, i.e., dynamically similar, examples. OmniAD [Martin et al. 2015] uses the Buckingham Π theory to construct a model of aerodynamic forces on 3D objects. Similarly, our problem is parameterized by the Reynolds number $\mathrm{Re} = \rho L V_\infty / \mu$ and the non-dimensionalized (i.e., normalized) shape $\mathcal{S}'$ which is uniformly scaled such that the representative length becomes one. The scaling of the shape can be done in the parameter space by scaling corner point positions $\vec{\mathbf{P}}_c$ and projection heights $\mathbf{H}$. Here, we define the representative length as the diagonal length of the bounding box of the input shape $\mathcal{S}$. From these non-dimensionalized input parameters, we obtain non-dimensionalized velocity and pressure values as $\vec{v}' = \vec{v}/V_\infty$ and $p' = p/(\rho V_\infty^2)$. We can easily compute the final velocity and pressure values by scaling them with V_∞ and ρV_∞^2, respectively.

Input and output vectors. In Section 4, we explained our polycube parameterization. Since our problem setting (i.e, a single object inside a constant flow) is invariant to translation and the grid moves together with the object, we can remove the translation component from the input parameters. We move the corner points such

that the averaged position of the corner points rests at the origin. The input parameter vector becomes a concatenation of the scaled and translated corner point positions, and projection heights of the surface points $\mathbf{x} = \{\vec{\mathbf{P}}_c, \mathbf{H}_s,\ \mathbf{H}^1_{ss},\ \mathbf{H}^2_{ss},\ \ldots\}$. We have three regressors: for drag coefficient, velocity, and pressure. Thus, the level of subdivision can be chosen differently for each of them. The machine learning outputs a vector concatenating the drag coefficient and the nondimentionalizeds values at all grid points for velocity $\mathbf{v} = \{\vec{v}'_0,\ \vec{v}'_1,\ \ldots\}$ and pressure $\mathbf{p} = \{p'_0,\ p'_1,\ \ldots\}$.

5.2 Gaussian Process

We choose Gaussian Process (GP) regression for inferring the CFD simulation data. The advantages of a GP in our context is four folds: (i) The GP can express a highly nonlinear relationship between input and output from a small number of training samples compared to the dimension of inputs ($T \ll M$). (ii) The GP is a probabilistic model that interpolates values from the training data probabilistically, i.e., we can obtain the *confidence* for the regression result aside from the regressed output itself. (iii) We can optimize its hyperparameters directory from the training data to increase the accuracy. (iv) We can handle noise in the training data. Thus, it is robust to the over-fitting problem.

Leaving the comprehensive details of GP to a textbook [Rasmussen and Williams 2005], we only briefly explain its key concepts. Here, we rely on common notations in statistic such as $\mathbb{E}(\cdot)$ for the expected value and $p(\cdot)$ for the probability. Suppose we would like to predict a scalar value y^* from a high-dimensional input $\mathbf{x}^* \in \mathbb{R}^M$ given the set of training pairs $\mathcal{D} = \{(y^n, \mathbf{x}^n)\}_{n=1}^N$, where the expected value of the output is zero $\mathbb{E}[y] = 0$. We denote the outputs in the training data concatenated with the unknown output as $\mathbf{Y} = \{y^1, \ldots, y^N, y^*\}$. The GP assumes that a set of outputs $\mathbf{Y}$ follows a multivariate Gaussian distribution $p(\mathbf{Y}) = \mathcal{N}(0, \mathbf{K})$, where $\mathbf{K}$ denotes the $(N+1) \times (N+1)$ covariance matrix. The desired solution is inferred from the conditional probability distribution of the output y^* given the known training data $p(y^*|\mathcal{D})$.

Each element of the covariance matrix $\mathbf{K}$ is defined as the expected value of the multiplication of two outputs $K^{ij} = \mathbb{E}[y^i y^j]$, where $1 \le i, j \le (N+1)$. Intuitively, the diagonal entry K^{ii} provides information on the variance of the output y^i, while the off-diagonal elements of the covariance matrix K^{ij} gives the magnitude of correlation between the output y^i and y^j (i.e., the similarity of the y^i and y^j). The GP regression models an element of the covariance matrix from the corresponding inputs using the kernel function $K^{ij} = k(\mathbf{x}^i, \mathbf{x}^j)$. Our choice of the kernel function is

$$k(\mathbf{x}^i, \mathbf{x}^j) = \theta_s \exp\left\{-\sum_{k=1}^{M} \theta_k (x^i_k - x^j_k)^2\right\} + \theta_n \delta^{ij}. \tag{1}$$

The first term in (1) defines the similarity of the two inputs $\mathbf{x}^i$ and $\mathbf{x}^j$ in terms of the similarity of the outputs y^i and y^j. The hyperparameter θ_s is a positive uniform scaling factor, while $\boldsymbol{\theta} = \{\theta_1, \ldots, \theta_M\} > 0$ are positive scaling factors for the each dimension of the input. This squared exponential covariance function is one of the most favored choices of kernel functions in statistics because it can be shown such a kernel corresponds to linear regression with infinite number of basis functions. The second term of (1) models Gaussian white noise with variance $\theta_n > 0$ assumed in the output. These hyperparameters $\theta_s, \boldsymbol{\theta}, \theta_n$ are optimized to agree with the dataset (see Sec. 5.3).

Given the covariance matrix $\mathbf{K}$, which defines how much each elements are relevant, we can compute the conditional probability of y^* under the training data $\mathcal{D}$ is given as

$$p(y^*|\mathcal{D}) = \mathcal{N}(\mathbf{w}^*\mathbf{y}, k(\mathbf{x}^*, \mathbf{x}^*) - \mathbf{w}^*\mathbf{k}^*), \tag{2}$$

$$\mathbf{w}^* = \mathbf{k}^*\mathbf{K}^{-1}_{\mathcal{D}} \tag{3}$$

$$\mathbf{k}^* = \{k(\mathbf{x}^1, \mathbf{x}^*), \ldots, k(\mathbf{x}^N, \mathbf{x}^*)\}^T \tag{4}$$

The $\mathbf{K}_{\mathcal{D}}$ is the $N \times N$ covariance matrix in the dataset $K^{ij}_{\mathcal{D}} = k(\mathbf{x}^i, \mathbf{x}^j)$.

We apply GP regression to the regression of the drag coefficient c_d. As the GP requires the output value to have a zero mean, we offset the drag coefficients with their average from the training data $y = c_d - \tilde{c}_d$, where $\tilde{c}_d$ means the average of the c_d from the training examples. How much the regression result is trustworthy is visualized to the user as the range of $\mu \pm \sigma$ where the probability to fall in that range is about 69% based on a normal distribution.

$$c_d(\mathbf{x}^*) = \mathbf{w}^*\mathbf{c}_d + \omega^*\tilde{c}_d, \quad where \quad \omega^* = 1 - \sum_{n=1}^{N} w^*_n. \tag{5}$$

Here $\mathbf{c}_d$ is a vector concatenating c_d values in the training data.

The GP regression scheme can be seen as interpolation from the training outputs with weight $\mathbf{w}^*$. Here we used the same weight to interpolate the training data for the pressure field and velocity field regression

$$\mathbf{p}(\mathbf{x}^*) = \mathbf{w}^*\mathbf{P} + \omega^*\tilde{\mathbf{p}}, \tag{6}$$

$$\mathbf{v}(\mathbf{x}^*) = \mathbf{w}^*\mathbf{V} + \omega^*\tilde{\mathbf{v}}, \tag{7}$$

where $\mathbf{P}$ and $\mathbf{V}$ are matrices stacking training data in rows. The weight $\mathbf{w}^*$ is optimized for the drag coefficient and reusing them for pressure and velocity regression is probably not the optimal. However, it is much computationally efficient compared to optimizing the weight for every element of the velocity and pressure. Moreover, the result shows the weight have much better accuracy compared to the naïve approach (see Sec.6). We leave the rigorous weight optimization for the pressure and velocity as a future work.

5.3 Hyperparameter Optimization

Given $\boldsymbol{\theta} = \{1, \ldots, 1\}$ and $\theta_n = 0$, the above interpolation scheme becomes equivalent to a radial based function (RBF) network, which measures the difference between inputs as a function of uniform Euclidean distance. When the dimension of the input M is very large, the naïve Euclidean distance suffers the *curse of dimensionality*, where the distances to the example inputs fail to give information of the closeness of the outputs. To alleviate this problem, we optimize $\boldsymbol{\theta}$ to emphasize elements of $\mathbf{x}$ that is relevant to the output. We cannot optimize all the elements of $\boldsymbol{\theta}$ as its dimension is much higher than the number of the training sample without assuming smoothly varying parameters (i.e., regularization). The smoothness distribution of the parameter over the shape is reasonable because the positions of the neighboring nodes have strong correlations and possibly the similar influence to the outputs. Our proposed PolyCube

representation naturally supports this approach. We directly specify the weights at the coarse subdivision level $\boldsymbol{\theta}_c$ and obtain the smooth weights for the remaining nodes in the finer subdivision level by interpolation. The interpolation scheme is based on the subdivision scheme in Sec. 4.2 – we interpolate the weight by averaging the two end points of an edge if the fine point is on the edge, and by averaging four corner points of a quadrant if the fine points are in the quadrant.

We denote $\boldsymbol{\Theta}$ for all the hyperparameters – uniform kernel weight θ_s, the weights for coarse nodes $\boldsymbol{\theta}_c$ and noise variance θ_n. We optimize these hyperparameters using the maximum likelihood method such that we have maximum probability of the training outputs under the optimized parameter as

$$\boldsymbol{\Theta}_{optimum} = \arg\max_{\boldsymbol{\Theta}} \log p(\mathbf{y}|\boldsymbol{\Theta}) = -\mathcal{L}(\boldsymbol{\Theta}), \tag{8}$$

$$\mathcal{L}(\boldsymbol{\Theta}) = \frac{1}{2}\log|\mathbf{K}_{\mathcal{D}}(\boldsymbol{\Theta})| + \frac{1}{2}\mathbf{y}^T\mathbf{K}_{\mathcal{D}}^{-1}(\boldsymbol{\Theta})\mathbf{y} + \frac{N}{2}\log(2\pi), \tag{9}$$

$$\text{where } \theta_s > 0,\ \boldsymbol{\theta}_c > 0,\ \theta_n > 0. \tag{10}$$

As the gradient of the log-likelihood in eq. (9) can be analytically computed [Rasmussen and Williams 2005], we solve this constraint optimization problem using the projected gradient decent method.

6 RESULT

Training data generation. We start with describing the car aerodynamics example we created using our framework. We then prepare the input shapes from the "car" category of ShapeNet [Chang et al. 2015] (see Figure 1-left). We manually modify the shape to remove the side mirrors, spoilers and tires. Then, we solve the Navier-Stokes equation in a highly detailed spatial grid for many time steps until we accurately obtain the time-averaged velocity and pressure fields. The degrees of freedom in these simulations range from 600 k to 700 k. We simulate 10 seconds of air flow and average the results of the last 4 seconds to obtain the training data. Each simulation takes approximately 50 minutes. Note that our machine learning approach does not depend on the fluid solver. In our example, the fluid flow is simulated using our in-house finite element Navier-Stokes solver with the k-epsilon turbulence model and SUPG stabilization [Zienkiewicz et al. 2013]. The simulation runs on a tetrahedral mesh that conforms to the boundary and we adaptively refined the mesh around the surface and the back of the car to resolve the boundary layer and separated vortices resulting from non-slip boundary condition. We used realistic physics parameters for the car simulation with the car driving in air at 72 km/h speed (Re = 5×10^6).

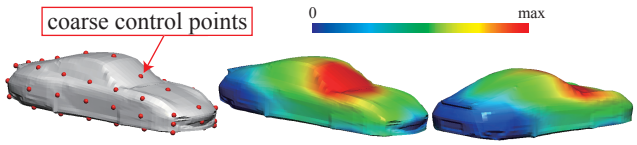

Fig. 6. Optimized weight $\boldsymbol{\theta}$ in the computation of the kernel. We set the control points at the nodes on the coarse level (left) and smoothly interpolate their values (middle and right). The weight illustrates how relevant the underlying shape is to the prediction.

Figure 5 illustrates the PolyCube and the resulting grid. This model is built on a PolyCube with three cubes assuming sedan or wagon-type cars. To encode the shape of the car, we choose level-4 subdivision for the input shape $M = 3698$, level-4 subdivision for the surface pressure discretization (3682 degrees of freedom), and level-3 for the velocity on the spatial volumetric grid (88494 degrees of freedom). The pressure and velocity values are sampled at the grid points, whose location can be computed from the input parameters. We prepare 889 pairs of input shapes, and the output simulation data consists of the drag coefficient value c_d, pressure and velocity field data computed with the CFD simulation. All training data is included in the supplemental material*.

Training. We optimize the hyperparameters according to the maximum likelihood method in (8). The hyperparameter optimization is the most time-consuming training step, taking approximately 30 minutes. With the optimized hyperparameters, the GP does not require expensive computations aside from the computation and inversion of the kernel matrix, which is done only once. We trained and ran the real-time demonstration program, as shown in the accompanied video, on a MacBook Pro 2014 model with a 3.0 GHz Core i7 CPU using the Intel MKL Library™. Fig. 6 shows the color-coded resulting optimized spatially-varying weights $\boldsymbol{\theta}$ in the kernel computation. Intuitively, the weights reflect the importance of specific input positions in predicting the output. In our example, the weight is high on the windshield and the A-pillar, while low around the front grill and the rear side. This is reasonable because the boundary layer grows and separates around the windshield and the air is stagnant at the front and back of the car. We also have analyzed the impact of the hyperparameter optimization. Fig. 8-right shows a comparison of the error with and without weight optimization. The unoptimized weight scenario is equivalent to a naïve RBF interpolation where all the weights are the same. With hyperparameter optimization, we can observe a significant improvement in accuracy, with an error decrease of about 68% (see Fig. 8).

Performance. We show our runtime aerodynamic design tool in Figure 1 and the accompanying supplemental video. While the user is interactively modifying the shape by moving the control points of B-Rep or the free-form deformation, the system consistently maintains 10 > frames per second for all shape-editing operations. By showing this drag force in real-time, the user can design aerodynamically-efficient shapes based on interactive trial-and-error shape exploration. The memory footprint for the three regressors (c_d, pressure and velocity) combined is 270MB.

Accuracy. To examine the accuracy of our model, we conducted 9-fold cross-validations, i.e., we split all the data into nine equal-sized subgroups and executed the cross-validation five times. For each cross-validation, we use one subgroup for validation and the rest for training. Fig. 7-left shows the distribution of the magnitude of errors with respect to the predicted standard distribution σ^*. This shows that when the confidence is low (i.e., the standard deviation is high) the error tends to be large. Fig. 7-right shows the distribution of the errors divided by the standard deviation. When the errors

*http://www.nobuyuki-umetani.com/publication/mlcfd_data.zip

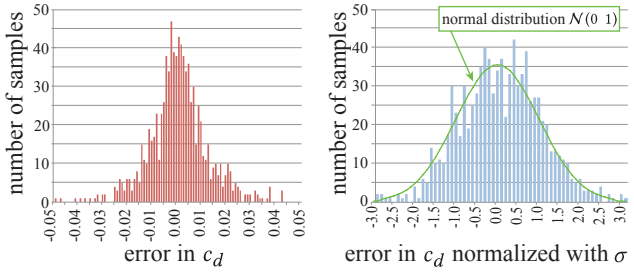

Fig. 7. (Left) Distribution of the magnitude of errors with respect to the predicted standard distribution. (Right) Histogram of the errors divided by the predicted standard distribution.

are normalized by their standard deviation, they follow the normal distribution $\mathcal{N}(0, 1)$.

Fig. 8 compares the accuracy of the drag coefficient with other regression approaches including linear regression, random forest, GP with optimized uniform weight, and GP without hyperparameter optimization ($\boldsymbol{\theta} = 1$). Fig. 8 shows the standard deviation of the errors of the drag coefficient with respect to the size of the training set for various regression strategies including linear regression with L_2 reguralizer, random forest, GP with optimized uniform weight, GP without hyperparameter optimization ($\boldsymbol{\theta} = 1$) and neural networks with and without the 50% dropout [Srivastava et al. 2014] . For the regression forest method, we used the implementation in the scikit-learn library with its default parameters. The neural network we used here is a perceptron with one hidden layer that has ten neurons with the logistic activation function. We trained the neural network until convergence (sum of squared error was less than 10^{-6} times the initial error). We observed that the neural network regression significantly performed worse in validation than the others because of the overfitting problem as the number of the training data ($\sim$ 0.8k) is much smaller than the dimension of the inputs ($\sim$ 3.7k). The dropout alleviated the overfitting problem to some extent, but the result was still worse than the other regressors. We are surprised to find the linear model is very competitive. Thanks to our PolyCube parameterization, we managed to make the input and output relationship close to linear. We include a Python script for linear least square fitting in the supplemental material*.

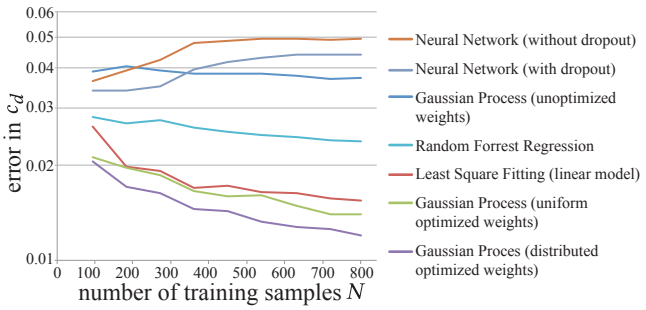

Fig. 8. Convergence of the standard variation of the errors in the drag coefficient prediction with respect to the training size for different regression strategies.

Compared to the other approaches, our regression achieves the highest accuracy in predicting the c_d value. The standard deviation of the error is about ± 0.012, which relates to a relative error of less than $\pm 3.4\%$ when normalized with an average drag coefficient of the example $\tilde{c}_d = 3.5$. According to an extensive study by SAE International, the standard deviation of the c_d values obtained from ten large different wind tunnel testing facilities throughout the world is $\pm 2.2\%$ and the difference can be up to 5% [Buchheim et al. 1983; Hucho 1998]. Allowing the assumption that the offline simulation would perfectly reproduce reality, our learned flow accuracy would be comparable to that of a wind tunnel. However, in practice the offline simulation is not perfectly accurate, leaving us with an unknown error relative to the true real-world behaviour. Our goal in this paper is to show that we can efficiently reproduce the time-consuming offline simulation result. Because no dataset of wind tunnel experiments with many different car shapes is publicly available, we leave the physical validation of the accuracy as future work.

We also evaluated the errors in the pressure and velocity fields. Fig. 9 shows the distribution of errors in the velocity and pressure fields obtained from the cross-validation. The horizontal axis shows the mean squared errors for the velocity and pressure values, while the vertical axis shows the frequency of the errors. Even in the case of the largest error in the validation, we did not observe a significant difference in the velocity and pressure fields compared to the ground truth. This graph also suggests that the distributed weight optimized for c_d improves the accuracy of the velocity and pressure fields prediction in comparison with the naïve weight $\boldsymbol{\theta} = 1$.

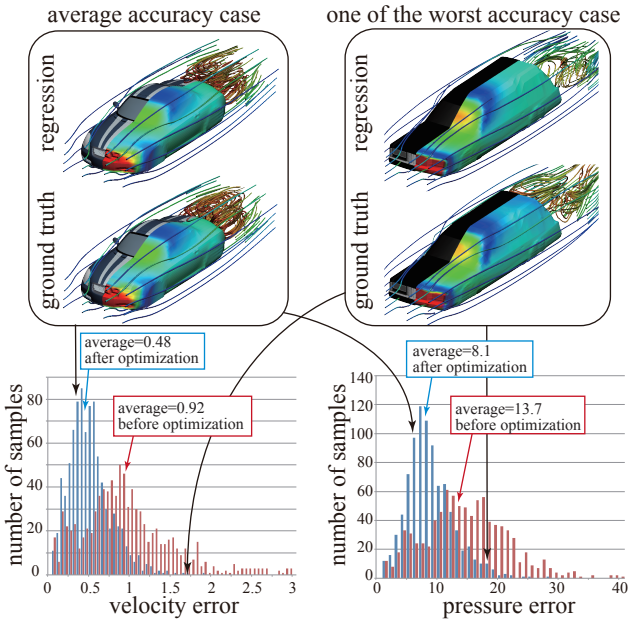

Fig. 9. (Top) Comparisons between ground truth velocity and pressure fields and regression results for one average accuracy case and one worst accuracy case. (Bottom) Frequency of the squared norm error of the velocity field (left) and pressure field (right).

More specifically, mean squared errors averaged over the example decreased 47% for velocity and 41% for the pressure. In terms of the maximum absolute value of the errors in the velocity and the pressure fields, their averaged value over the examples are 8.80 m/s and 126 Pa, respectively, using the optimized weight. In the supplemental material, we include about one hundred comparisons between ground truth and machine learning results.

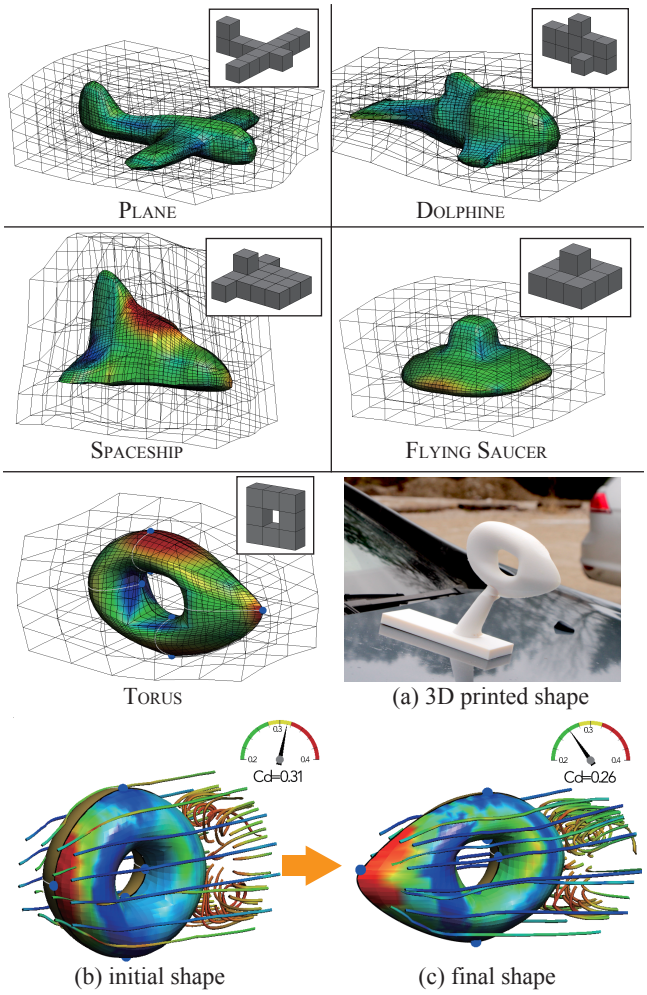

Fig. 10. Examples of our PolyCube 3D shape and field parameterization for a various shapes. The resulting aerodynamically efficient shape can be 3D printed and attached to a car (bottom-rightmost).

Parameterization of more complex shapes. Our shape and field parameterization technique is not limited to car-like, almost concave shapes. We can parameterize a wide variety of shapes starting from a PolyCube that approximates the shapes. Figure 10 shows examples of our shape parameterization including an airplane, a dolphin, a flying saucer, a spaceship, and a torus. While the previous technique [Umetani 2017] fits a cube into the shape, our PolyCube-based technique can achieve better quality (i.e., less bias) in the parameterization for concave shapes. Note that for the torus example (Fig. 10-bottom top left), there is no way to continuously fit a cube into this genus-1 shape, thus the previous technique [Umetani 2017] results in a failure.

For the torus model, we also construct a machine learning CFD model based on 50 training examples. These example shapes are chosen randomly from the recording of the user's modeling sequence. For the torus model, we assume that the shape is used as a car-top sign, so we set the dimension of the torus about 10 cm in diameter and 4 cm in thickness. We encourage the readers to look at the supplemental material for the interactive modelling sequence of the car-top sign while getting the feedback from the real-time CFD estimation. Fig. 10-(b) and -(c) show the CFD simulation results on the initial torus shape and final torus shape after the editing. Using our system, the torus has 15% less drag coefficient (from 0.31 to 0.26) compared to the original undeformed torus shape based on the simulation. The bottom rightmost image in Fig. 10-(a) shows the resulting aerodynamically-efficient torus shape we 3D printed.

7 LIMITATIONS AND FUTURE WORK

Currently, the user needs to manually specify a PolyCube as a rough representation of the input shape. Automatically generating such a PolyCube representation would further reduce the user's workload. We currently use a single regression model which is built on a single PolyCube. An interesting avenue for future work would be to explore the construction of multiple regression models for different PolyCube configurations and switch them during the user's interaction. This would require developing a reliable framework to determine decision criteria and methods for choosing and switching between PolyCube representations.

In this work, we constructed a parameterization by projecting points of a subdivided PolyCube onto the input shape in predetermined directions. While this parameterization can handle a wide range of shapes similar to the input PolyCube, it is still challenging to parameterize a highly concave shape or very sharp and thin features (e.g., side mirrors) without excessive distortion of the grid. We are planning to increase the representation capability by adaptively changing the resolution of the input template PolyCube in regions in which we know a priori that high curvatures might occur.

We believe our approach could be extended to a wide range of other complex physical phenomena, which we plan to investigate in the future. Since our approach parameterizes the shape and associated 3D field, we can parameterize boundary conditions and the solution of many partial differential equations such as the thermal fluid equation, supersonic fluid equation, or electromagnetic fluid equations. Finally, we think it would be exciting to extend our method to include a state graph and/or a temporal frequency decomposition of the field. As an application, we are particularly interested in biomechanical simulation (e.g., the simulation of flow inside a heart or blood vessels), such that doctors can give more reliable diagnostics based on preexisting simulation data.

8 CONCLUSION

We have presented a technique to use to interactively predict how fluid flows around a three-dimensional object given its shape. Our

primary contribution is a novel interactive PolyCube representation to parameterize the input shape and represent a field on and around the shape. Using that parameterization, we successfully constructed a regression model using Gaussian Process. Our method is fast enough to be integrated into existing interactive shape modeling tools. With the real-time feedback, the user can manually optimize the aerodynamics of an object. We evaluated the accuracy of our method by cross-validation, confirming a good match of our results to ground truth simulation.

ACKNOWLEDGEMENTS

We thank anonymous reviewers and Ryan Schmidt for their comments and advice. We appreciate the assistance from Rin Ishikawa for the production of the supplemental video. This project has received funding from the European Research Council (ERC) under the European Union's Horizon 2020 research and innovation programme (grant agreement No 715767 – MATERIALIZABLE).

REFERENCES

Pierre Baqué, Edoardo Remelli, François Fleuret, and Pascal Fua. 2018. Geodesic Convolutional Shape Optimization. *CoRR* abs/1802.04016 (2018). arXiv:1802.04016 http://arxiv.org/abs/1802.04016

B. Bonev, L. Prantl, and N. Thuerey. 2017. Pre-computed Liquid Spaces with Generative Neural Networks and Optical Flow. *ArXiv e-prints* (April 2017). arXiv:cs.GR/1704.07854

R. Buchheim, R. Unger, P. Jousserandot, E. Mercker, F. K. Schenkel, Y. Nishimura, and D. J. Wilsden. 1983. Comparison Tests Between Major European and North American Automotive Wind Tunnels. In *SAE Technical Paper*. SAE International. https://doi.org/10.4271/830301

A. X. Chang, T. Funkhouser, L. Guibas, P. Hanrahan, Q. Huang, Z. Li, S. Savarese, M. Savva, S. Song, H. Su, J. Xiao, L. Yi, and F. Yu. 2015. ShapeNet: An Information-Rich 3D Model Repository. *ArXiv e-prints* (Dec. 2015). arXiv:cs.GR/1512.03012

Mengyu Chu and Nils Thuerey. 2017. Data-Driven Synthesis of Smoke Flows with CNN-based Feature Descriptors. *Transaction on Graphics (SIGGRAPH)* 36(4) (Apr 2017), 14.

Jonathan M Cohen, Sarah Tariq, and Simon Green. 2010. Interactive fluid-particle simulation using translating Eulerian grids. In *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. ACM, 15–22.

Tyler De Witt, Christian Lessig, and Eugene Fiume. 2012. Fluid Simulation Using Laplacian Eigenfunctions. *ACM Trans. Graph.* 31, 1, Article 10 (Feb. 2012), 11 pages. https://doi.org/10.1145/2077341.2077351

Xianzhong Fang, Weiwei Xu, Hujun Bao, and Jin Huang. 2016. All-hex Meshing Using Closed-form Induced Polycube. *ACM Trans. Graph.* 35, 4, Article 124 (July 2016), 9 pages. https://doi.org/10.1145/2897824.2925957

Michael S. Floater. 2003. Mean Value Coordinates. *Comput. Aided Geom. Des.* 20, 1 (March 2003), 19–27. https://doi.org/10.1016/S0167-8396(02)00002-5

Igor Guskov, Kiril Vidimče, Wim Sweldens, and Peter Schröder. 2000. Normal Meshes. In *Proceedings of the 27th Annual Conference on Computer Graphics and Interactive Techniques (SIGGRAPH '00)*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 95–102. https://doi.org/10.1145/344779.344831

Kurt Hornik. 1991. Approximation Capabilities of Multilayer Feedforward Networks. *Neural Netw.* 4, 2 (March 1991), 251–257. https://doi.org/10.1016/0893-6080(91)90009-T

Jin Huang, Tengfei Jiang, Zeyun Shi, Yiying Tong, Hujun Bao, and Mathieu Desbrun. 2014. ℓ1Basedd Construction of Polycube Maps from Complex Shapes. *ACM Trans. Graph.* 33, 3, Article 25 (June 2014), 11 pages. https://doi.org/10.1145/2602141

Hucho. 1998. *Aerodynamics of road vehicles : from fluid mechanics to vehicle engineering*. Society of Automotive Engineers, Warrendale, PA.

Tao Ju, Scott Schaefer, and Joe Warren. 2005. Mean Value Coordinates for Closed Triangular Meshes. *ACM Trans. Graph.* 24, 3 (July 2005), 561–566. https://doi.org/10.1145/1073204.1073229

Theodore Kim and John Delaney. 2013. Subspace Fluid Re-simulation. *ACM Trans. Graph.* 32, 4, Article 62 (July 2013), 9 pages. https://doi.org/10.1145/2461912.2461987

L'ubor Ladický, SoHyeon Jeong, Barbara Solenthaler, Marc Pollefeys, and Markus Gross. 2015. Data-driven Fluid Simulations Using Regression Forests. *ACM Trans. Graph.* 34, 6, Article 199 (Oct. 2015), 9 pages. https://doi.org/10.1145/2816795.2818129

Tobias Martin, Nobuyuki Umetani, and Bernd Bickel. 2015. OmniAD: Data-driven Omni-directional Aerodynamics. *ACM Trans. Graph.* 34, 4, Article 113 (July 2015), 12 pages. https://doi.org/10.1145/2766919

Mojang. 2009. Minecraft. (2009).

Charles Ruizhongtai Qi, Hao Su, Kaichun Mo, and Leonidas J. Guibas. 2016. PointNet: Deep Learning on Point Sets for 3D Classification and Segmentation. *CoRR* abs/1612.00593 (2016). arXiv:1612.00593 http://arxiv.org/abs/1612.00593

Carl Edward Rasmussen and Christopher K. I. Williams. 2005. *Gaussian Processes for Machine Learning (Adaptive Computation and Machine Learning)*. The MIT Press.

Adriana Schulz, Jie Xu, Bo Zhu, Changxi Zheng, Eitan Grinspun, and Wojciech Matusik. 2017. Interactive Design Space Exploration and Optimization for CAD Models. *ACM Trans. Graph.* 36, 4, Article 157 (July 2017), 14 pages. https://doi.org/10.1145/3072959.3073688

Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. 2014. Dropout: A Simple Way to Prevent Neural Networks from Overfitting. *Journal of Machine Learning Research* 15 (2014), 1929–1958. http://jmlr.org/papers/v15/srivastava14a.html

Matt Stanton, Ben Humberston, Brandon Kase, James F. O'Brien, Kayvon Fatahalian, and Adrien Treuille. 2014. Self-refining Games Using Player Analytics. *ACM Trans. Graph.* 33, 4, Article 73 (July 2014), 9 pages. https://doi.org/10.1145/2601097.2601196

Matt Stanton, Yu Sheng, Martin Wicke, Federico Perazzi, Amos Yuen, Srinivasa Narasimhan, and Adrien Treuille. 2013. Non-polynomial Galerkin Projection on Deforming Meshes. *ACM Trans. Graph.* 32, 4, Article 86 (July 2013), 14 pages. https://doi.org/10.1145/2461912.2462006

Hang Su, Subhransu Maji, Evangelos Kalogerakis, and Erik Learned-Miller. 2015. Multi-view Convolutional Neural Networks for 3D Shape Recognition. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV) (ICCV '15)*. IEEE Computer Society, Washington, DC, USA, 945–953. https://doi.org/10.1109/ICCV.2015.114

Marco Tarini, Kai Hormann, Paolo Cignoni, and Claudio Montani. 2004. PolyCube-Maps. *ACM Trans. Graph.* 23, 3 (Aug. 2004), 853–860. https://doi.org/10.1145/1015706.1015810

Jonathan Tompson, Kristofer Schlachter, Pablo Sprechmann, and Ken Perlin. 2016. Accelerating Eulerian Fluid Simulation With Convolutional Networks. *CoRR* abs/1607.03597 (2016). arXiv:1607.03597 http://arxiv.org/abs/1607.03597

Adrien Treuille, Andrew Lewis, and Zoran Popović. 2006. Model Reduction for Real-time Fluids. *ACM Trans. Graph.* 25, 3 (July 2006), 826–834. https://doi.org/10.1145/1141911.1141962

K. Um, X. Hu, and N. Thuerey. 2017. Liquid Splash Modeling with Neural Networks. *ArXiv e-prints* (April 2017). arXiv:cs.GR/1704.04456

Nobuyuki Umetani. 2017. Exploring Generative 3D Shapes Using Autoencoder Networks. In *SIGGRAPH Asia 2017 Technical Briefs (SA '17)*. ACM, New York, NY, USA, Article 24, 4 pages. https://doi.org/10.1145/3145749.3145758

Nobuyuki Umetani, Yuki Koyama, Ryan Schmidt, and Takeo Igarashi. 2014. Pteromys: Interactive Design and Optimization of Free-formed Free-flight Model Airplanes. *ACM Trans. Graph.* 33, 4, Article 65 (July 2014), 10 pages. https://doi.org/10.1145/2601097.2601129

Peng-Shuai Wang, Yang Liu, Yu-Xiao Guo, Chun-Yu Sun, and Xin Tong. 2017. O-CNN: Octree-based Convolutional Neural Networks for 3D Shape Analysis. *ACM Trans. Graph.* 36, 4, Article 72 (July 2017), 11 pages. https://doi.org/10.1145/3072959.3073608

Martin Wicke, Matt Stanton, and Adrien Treuille. 2009. Modular Bases for Fluid Dynamics. *ACM Trans. Graph.* 28, 3, Article 39 (July 2009), 8 pages. https://doi.org/10.1145/1531326.1531345

Zhirong Wu, Shuran Song, Aditya Khosla, Xiaoou Tang, and Jianxiong Xiao. 2014. 3D ShapeNets for 2.5D Object Recognition and Next-Best-View Prediction. *CoRR* abs/1406.5670 (2014). http://arxiv.org/abs/1406.5670

Kai Xu, Vladimir G Kim, Qixing Huang, Niloy Mitra, and Evangelos Kalogerakis. 2016. Data-driven shape analysis and processing. In *SIGGRAPH ASIA 2016 Courses*. ACM, 4.

Olek C Zienkiewicz, Robert L Taylor, and P. Nithiarasu. 2013. *The Finite Element Method for Fluid Dynamics, Seventh Edition* (7 ed.). Butterworth-Heinemann. http://amazon.com/o/ASIN/1856176355/