URBANFLOW: DESIGNING COMFORTABLE OUTDOOR AREAS

Daoming Liu
Florian Rist
Dominik L. Michels

Visual Computing Center
KAUST
Campus, Building 1, Thuwal 23955, KSA
{daoming.liu,florian.rist,dominik.michels}@kaust.edu.sa

ABSTRACT

Design decisions in urban planning have to be made with particular carefulness as the resulting constraints are binding for the whole architectural design that follows. In this context, investigating and optimizing the airflow in urban environments is critical to design comfortable outdoor areas as unwanted effects such as windy areas and the formation of heat pockets have to be avoided. Our *UrbanFlow* framework enables interactive architectural design allowing for decision making based on simulating urban flow. Compared to real-time fluid flow simulation, enabling interactive architecture design poses an even higher computational efficiency challenge as evaluating a design by simulation usually requires hundreds of time steps. This is addressed based on a highly efficient Eulerian fluid simulator in which we incorporate a unified porosity model which is devised to encode digital urban models containing objects such as buildings and trees. *UrbanFlow* is equipped with an optimization routine enabling the direct computation of design adaptations improving livability and comfort for given parameterized architectural designs. To ensure convergence of the optimization process, instead of the classical Navier–Stokes equations, the Reynolds-averaged Navier–Stokes equations are solved as this can be done on a relatively coarse grid and allows for the decoupling of the effects of turbulent eddies which are taken into account using a separate turbulence model. As we demonstrate on a real-world example taken from an ongoing architectural competition, this results in fast convergence of the optimization process which computes a design adaptation avoiding heat pockets as well as uncomfortable windy areas. *UrbanFlow* exploits the power of the GPU running on a single desktop computer as it is widely available in most architectural and urban planning firms. We also provide a plugin which enables its use with the *Rhinoceros 3D* software widely used in computational design and architecture.

Keywords: Fluid Simulation, Inverse Problems, Numerical Optimization, Outdoor Areas, Urban Design.

1 INTRODUCTION

Urban planning deals with the design of settlements, building groups, city districts and public spaces in general. It can be understood as an early stage of architectural design or its preceding process. Design decisions have to be made with particular carefulness as the resulting constraints are binding for the whole architectural design that follows. To avoid bad decision making, carefully investigating potential implications is absolutely critical. In this regard, architects and urban planners have a vast interest in understanding and controlling the airflow within and around the particular design area as well as in its surrounding neigh-

borhood. This understanding can be used to reduce the energy consumption of a building in hot and cold climate (Feng et al. 2019, Al-Saadi and Shaaban 2019), improve the outdoor comfort (Adamek et al. 2017), and reduce building costs by controlling wind loads (Thordal, Bennetsen, and Koss 2019, Sharma, Mittal, and Gairola 2018). Saving energy is of paramount importance since the building sector is currently responsible for about 40% of the global carbon emissions (Lucon et al. 2014). Improving natural ventilation not only reduces the energy consumption of a building (Chen, Tong, and Malkawi 2017, Irving and Clements-Croome 2005), on a larger scale it also plays a key role in ensuring sufficient ventilation in urban areas (Kaseb et al. 2020, Ramponi et al. 2015) and creating a livable outdoor environment. The effect of global warming is amplified in cities by the formation of heat pockets (Zhongming et al. 2021). Optimizing the airflow can help to mitigate this effect, and to create a livable and comfortable outdoor environment.

Air flow in an urban environment has to be considered as highly complex, and a designer can usually not rely on intuition to successfully utilize it. Computational fluid dynamics (CFD) can provide the necessary information to allow the designer to take informed decisions and improve the design. This information is already needed in early design phases, because fundamental decisions about the layout and morphology of the design are difficult to change in later design phases. Current simulation tools do not fit the specific needs of the architects in these early design phases. Compared to real-time fluid flow simulation, enabling interactive architecture design poses an even higher computational efficiency challenge as evaluating a design by simulation usually requires hundreds of time steps.

We present *UrbanFlow*, an interactive simulation and optimization framework which can be integrated into the design workflow right from conceptual design on. It is seamlessly integrated into a computer-aided design (CAD) system often used by architects, allows for interactive design as well as automated optimizations. It further includes a porosity model as it is often overlooked that a city does not only consist of buildings, but important porous structures like trees as well. At the core of *UrbanFlow* is our simulator designed as an Eulerian fluid solver. From a technical perspective, we had to address several points to enable the interactive flow simulation in urban environments. First, simply representing buildings as solid objects implies significant limitations as solid-wall boundary conditions for every object have to be enforced. Moreover, in an interactive design mode, the coefficient matrix for the pressure projection equation has to be reconstructed whenever the geometric configuration of a building is edited. As conjugate gradient (CG) preconditioners usually depend on the coefficient matrix, additional re-computations would be required substantially reducing the efficiency. Also, we would like to point out that some buildings are not purely solid objects dependent on their specific architecture potentially containing open design elements. Consequently, we propose to represent buildings as porous material instead of regular solid objects, and a high-level porosity model is constructed based on a widely-used empirical model for trees. For the urban flow simulation, an extra drag force term is used instead of enforcing the solid-wall boundary conditions. This way, the porosity effect of buildings can be easily included. More important, it can decouple the dependence between the coefficient matrix of the pressure projection equation of the CG preconditioner as well as the geometrical configuration of buildings. Therefore, the efficiency of the urban flow simulation can be improved using precomputation. However, although the CG preconditioner can be precomputed, its application process cannot be precomputed and may be quite time-consuming for some complicated preconditioners. This work explores an approximate inverse preconditioner that has not been widely used in graphics literature so far. Compared to other commonly used preconditioners, it shows superior balance between convergence rate and parallel-computing costs.

Our fluid solver supports arbitrary building shapes efficiently in order to facilitate a creative design process. In general, we need to know for each grid cell if it is empty or belongs to an object within the urban environment. Our building surfaces are geometrically represented as closed polygon meshes. Consequently, we adopt a classical even-odd rule which counts how many times a ray starting at the specific point crosses boundaries.The whole urban scene is discretized using a set of cells. For each cell we store a porosity value as a new unified way to encode urban models containing elements such as buildings and trees, as well as empty regions. Cut-cells can also be naturally handled by averaging the porosity values of their vertices.

Motivated by this novel perspective, we further enable a new design pattern in which the porosity values are encoded by color and directly set by the designer. This way, the users could perform the design very easily in 2D, e.g., by painting. In parallel, the geometrical configurations of buildings and trees could be easily obtained by decoding the color of each grid cell without any explicit geometric calculations. Computational costs are further reduced and do not depend on the complexity of the building and tree shapes. This idea could also be easily extended to 3D for a given user-interaction interface which allows to directly access the 3D grid cells.

Next to enabling interactive architectural design, we further propose an optimization routine which directly computes slight design adaptations improving livability and comfort for given parameterized building layouts while maintaining the overall concept and structure of the design. The optimization routine employs a gradient descent procedure to compute the design adaptations which poses the challenges to avoid oscillations within the optimization process and to ensure convergence within a reasonable number of iterations. This is not easily possible when using the classical formulation of the Navier–Stokes (NS) equations due to the turbulent nature of the resulting flow. Instead, the Reynolds-averaged NS (RANS) equations allow for a decoupling of the effects of turbulent eddies which are taken into account using a separate turbulence model resulting in fast convergence of the optimization.

Our specific contributions are as follows. (1) A unified porosity model is devised to encode digital urban models containing objects such as buildings and trees. (2) An Eulerian urban flow simulator is presented showing high performance due to the pre-computation of the pressure projection equation's coefficient matrix and its CG preconditioner. (3) An approximate inverse preconditioner is explored and evaluated for the flow simulation showing excellent balance between convergence rate and computational costs. (4) An optimization procedure based on solving RANS equations is devised improving urban designs as demonstrated on a real-world example from an ongoing architectural competition. (5) We provide a plugin to couple our framework with the *Rhinoceros 3D* (*Rhino*) software which is widely used in computational design and architecture, and exploit the power of parallel hardware by implementing our framework on the graphics processing unit (GPU).

2 RELATED WORK

Eulerian Fluid Simulation. At least since the seminal work of Stam (Stam 1999), the simulation of fluids is an established research field within computer graphics. Back then, Stam proposed a semi-Lagrangian-type scheme to handle the nonlinear advection term within the NS equations based on Chorin's (Chorin et al. 1990) framework. Unconditionally stable fluid simulations could be achieved which are very attractive for graphics applications but suffer from severe numerical diffusion. Subsequent approaches have been proposed to overcome this limitation including the Mac-Cormack scheme (Selle, Fedkiw, Kim, Liu, and Rossignac 2008) which is adopted in our work, vorticity confinement (Zhang, Bridson, and Greif 2015), the advection-reflection scheme (Zehnder, Narain, and Thomaszewski 2018), and bidirectional mapping (Qu et al. 2019). It is also worthy to mention that it has been claimed (Rando, Muñoz, and Patow 2016) that a full NS-based model would be too complicated to efficiently compute urban flow for which reason a light weight Lattice Boltzmann method has been employed for 2D urban flow simulations. In contrast, our work devises a fast NS equation solver for 3D urban flow.

The most time-consuming operation within the numerical integration of the NS equations is usually the pressure projection step in which a large scale pressure Poisson equation set has to be solved. Consequently, significant effort has been invested to accelerate this step. In general, iterative solvers are much more preferred than direct methods, since they behave much better in terms of computation time and memory costs for such large scale linear systems. Among many iterative methods, Krylov subspace approaches (Saad 2003) such as preconditioned conjugate gradient (PCG) methods are widely adopted in graphics literature as the coefficient matrix of the pressure Poisson equations is usually symmetric and positive definite. The convergence rate of such PCG methods depends heavily on the efficacy of their preconditioners. The

list of popular preconditioners include Jacobi, symmetric successive over-relaxation (SSOR), incomplete Cholesky factorization (Foster and Fedkiw 2001, Selle, Rasmussen, and Fedkiw 2005), as well as multigrid methods (McAdams et al. 2010, Aanjaneya et al. 2019). In our work, a kind of an approximate inverse preconditioner not widely used before in graphics is explored and evaluated against these popular preconditioners.

Optimizing Designs based on Fluid Simulation. Among others, shape optimization based on fluid simulation is an important topic with several potential applications. Plenty of work has been conducted in the CFD community, especially for the automobile and airplane industries, and became recently popular for closed indoor environment research (Liu, Zhang, Xue, Zhai, Wang, Wei, and Chen 2015). Topology optimization also shares some similarities, and we refer to the review paper of Deaton and Grandhi (Deaton and Grandhi 2014) for more details. Within the graphics community, Du et al. (Du et al. 2020) recently worked on design optimization for fluidic devices considering Stokes flow. This can be treated as the first step in this direction as claimed in their paper. Our work will target at the more challenging turbulent flow and the architecture design application in particular.

Computational Architectural Design. This field is of interest to many researchers within the graphics and CAD communities. The textbook of Pottmann et al. (Pottmann et al. 2008) provides an introduction to the underlying concepts and applications. For more recent advances in this field we refer to a survey (Pottmann, Eigensatz, Vaxman, and Wallner 2015) and a recent publication (Gavriil, Guseinov, Pérez, Pellis, Henderson, Rist, Pottmann, and Bickel 2020). However, most of this prior contributions were focused on the form/shape design in contemporary architecture. We further explore the optimization of designs by incorporating physical simulation, and in particular the air flow simulation within complex urban models.

3 FAST URBAN FLOW SIMULATION

Incompressible Navier-Stokes Equations. The incompressive NS equations are given by

$$\nabla \cdot \mathbf{u}' = 0\,, \tag{1}$$

$$\frac{\partial \mathbf{u}'}{\partial t} + (\mathbf{u}' \cdot \nabla)\mathbf{u}' = -\frac{1}{\rho_{\mathsf{air}}}\nabla p' + \nu \nabla^2 \mathbf{u}'\,, \tag{2}$$

in which $\mathbf{u}'$ denotes the velocity, p' the pressure, ν the kinematic viscosity of the air, and ρ_{air} the density of the air. The zero divergence condition in Eq. (1) ensures incompressibility of the air flow. Eq. (2) is also called momentum equation.

Reynolds-averaged Navier–Stokes Equations. In the industrial design practice, especially for turbulent flow, the incompressible NS equations were rarely solved in a direct fashion as this requires a high grid resolution to capture the smallest eddy scale (i.e., Kolmogorov scale, around $Re^{3/4}$ for Reynolds number Re). Instead, the Reynolds-averaged NS (RANS) equations are widely used and implemented as a mainstream method in commercial CFD software (Manceau 2021). RANS equations can be solved on a relative coarse grid with the effect of turbulent eddy modelled with an additional turbulence model. Moreover, in our optimization routine (see Section 4.2) we observe that the directly solved incompressible NS model can produce serious osculations not achieving convergence. RANS is a physical approximation of the original NS model derived via a Reynolds decomposition, Reynolds averaging and the Boussinesq relation. The mathematical formulations of the time-dependent RANS (Hanjalić and Launder 2020) are given by Eq. (1) and

$$\frac{\partial \mathbf{u}}{\partial t} + (\mathbf{u} \cdot \nabla)\mathbf{u} = -\frac{1}{\rho_{\mathsf{air}}}\nabla p + (\nu + \nu_t)\nabla^2 \mathbf{u} + \frac{1}{\rho_{\mathsf{air}}}\mathbf{f_d}\,, \tag{3}$$

$$\mathbf{u} = \mathbf{u}' - \tilde{\mathbf{u}}, \quad p = p' - \tilde{p}\,, \tag{4}$$

in which $(\mathbf{u}, p)$ denotes the mean (i.e., time-averaged) part of $(\mathbf{u}', p')$, and $(\tilde{\mathbf{u}}, \tilde{p})$ denotes the fluctuating part according to the Reynolds decomposition. Motivated by the widely used drag force model for

trees (Kenjereš and ter Kuile 2013, Kang, Kim, Kim, Choi, and Park 2017), we have included a drag force $\mathbf{f_d} = -\rho_{\mathsf{air}}\, C_d\, \mathrm{LAD}\, |\mathbf{u}|\, \mathbf{u}$, in which C_d denotes the drag coefficient which depends on the leaf roughness, the leaf area density *LAD* denotes the total one-side leaf area per unit volume, $\mathbf{u}$ denotes the wind velocity and $|\mathbf{u}|$ its magnitude. A high-level drag force model for buildings can be constructed if LAD is replaced with the term $a\ ((1-\phi)/(\phi+e))^b$. Here, ϕ denotes the porosity coefficient of the building and $e = 1^{-10}$ is a small number to avoid zero-division. The coefficients $a = 0.62$ and $b = 2.5$ have been experimentally validated.

Turbulence Model. Intuitively, the eddy viscosity term ($\nu_t \gg \nu$) in the RANS model is used to smooth the mean flow. This is why it has been called "laminarization" in the original paper (Jones and Launder 1972) of the popular $k-\varepsilon$ turbulence model which considers the turbulent kinetic energy k and the turbulent kinetic energy dissipation rate ε. We have extensively tested this model in our numerical experiments and observed serious numerical instability problems. Hence, we further implemented its alternative variant, the $k-\omega$ turbulence model (Wilcox 2006) (small-eddy frequency $\omega = \varepsilon/(C_\mu k)$ with a model parameter C_μ) given by

$$\frac{\partial k}{\partial t} + (\mathbf{u}\cdot\nabla)k = P_k - C_\mu k\omega + (\nu + \sigma^*\nu_t)\nabla^2 k\,, \tag{5}$$

$$\frac{\partial \omega}{\partial t} + (\mathbf{u}\cdot\nabla)\omega = 2\alpha|S|^2 - \beta\omega^2 + (\nu + \sigma\nu_t)\nabla^2\omega\,, \tag{6}$$

$$P_k = 2\nu_t|S|^2\,, \qquad S = (\nabla\mathbf{u} + (\nabla\mathbf{u})^{\mathsf{T}})/2\,, \tag{7}$$

$$\nu_t = \frac{k}{\tilde{\omega}}, \quad \tilde{\omega} = \max\{\omega,\ C_{\mathrm{lim}}|S|/(C_\mu/2)\}\,. \tag{8}$$

The mean strain-rate tensor is denoted by S. The values of k and ω on the inlet boundary are calculated as $k = 1.5\,(I\,U_{\mathrm{ref}})^2$ and $\omega = L^{-1}C_\mu^{-0.25}k^{0.5}$, in which I denotes the turbulence intensity, and U_{ref} and L denote the reference speed and length scale respectively. The model's coefficients are shown in Table 1.

Table 1: Coefficients within the $k-\omega$ model.

C_μ	α	β	σ	σ^*	C_{lim}
0.09	0.52	0.0708	0.5	0.6	7/8

Discretization and Numerical Integration. While the $k-\omega$ model shows improved numerical robustness, it still limits the use of a larger time step (around 1/10) compared to the case without using a decoupled turbulence model. To support a stable simulation with a larger time step, we propose the following viscosity limiter, which is derived via the von Neumann stability analysis on the diffusion term of Eq. (3):

$$\tilde{\nu_t} = \min\left\{\nu_t,\ \ 2\Delta t(\Delta x^2 + \Delta y^2 + \Delta z^2)^{-1} - \nu\right\}. \tag{9}$$

In *UrbanFlow*, the time-dependent RANS (or called U/T RANS) equations share the same mathematical form with NS except an extra eddy-viscosity parameter, then can be solve with fast simulation methods developed in computer graphics literature. In this work, we adopt the Eulerian Fluid simulation framework with a uniform staggered Cartesian grid. The components (u,v,w) of the velocity field $\mathbf{u}$ are located at the face center of each grid cell while pressure p and external drag force $\mathbf{f_d}$ stored at the center of each grid cell. A time splitting method is used to handle the time derivative term of the equation, and all other terms are temporally integrated one by one. The Mac-Cormack-based semi-Lagrangian scheme is adopted to handle the nonlinear advection term, and the pressure Poisson equation is solved by the conjugate gradient algorithm (CG) with an approximate inverse (AI) preconditioner. The additional turbulence model equations are discretized with a forward difference scheme for the time derivative term, central difference scheme for the Laplacian diffusion term, and upwind scheme for the advection term. Please note, that the turbulence equations have been solved after the projection step so that the incompressive velocity data can be used.

Approximate Inverse Preconditioner. GPU parallelization is exploited to boost up *UrbanFlow*'s performance. One of the most important parts is to handle the preconditioned CG algorithm that is adopted to iteratively solve the large scale pressure Poisson equation in each time step. However, the modified incomplete Cholesky decomposition preconditioner (widely used in serial code) cannot be easily parallelized, because it involves forward and backward substitutions to solve the triangular matrix equation which are inherently serial procedures (Chu, Zafar, and Yang 2017). The algebraic and geometric multigrid preconditioners are too complicated for our application. The simple Jacobi preconditioner is straightforward for the parallel implementation on the GPU, but it only comes with slight improvements on the convergence rate compared to the naive CG algorithm. A kind of preconditioner that was not widely used before in graphics literature is explored in *UrbanFlow*. It was firstly proposed by Ament at al. (Ament et al. 2010) and was called incomplete Poisson (IP) preconditioner in their original article. Later on, a slightly modified formulation was rigorously derived from the first order Neumann series approximation of the SSOR preconditioner by Helfenstein and Koko (Helfenstein and Koko 2012). When solving a system $\mathbf{Ax} = \mathbf{b}$, the basic idea of such a preconditioner is to find a direct approximating matrix $\mathbf{M}^{-1}$ of $\mathbf{A}^{-1}$, instead of the approximating matrix $\mathbf{M}$ of $\mathbf{A}$. It turned out that it can be applied as simple as the Jacobi preconditioner through one matrix-vector multiplication while almost equally good results compared to other more advanced preconditioners. This is very attractive for our work. Its formulation is as follows:

$$\mathbf{M}^{-1} = \mathbf{K}^{\mathsf{T}}\mathbf{K}, \tag{10}$$

in which the first-order and second-order approximations of K are

$$\mathbf{K} = \sqrt{2-\omega}\bar{\mathbf{D}}^{-1/2}(\mathbf{I} - \mathbf{L}\bar{\mathbf{D}}^{-1}), \tag{11}$$

$$\mathbf{K} = \sqrt{2-\omega}\bar{\mathbf{D}}^{-1/2}(\mathbf{I} - \mathbf{L}\bar{\mathbf{D}}^{-1} + (\mathbf{L}\bar{\mathbf{D}}^{-1})^2), \tag{12}$$

where $0 < \omega < 2$, $\bar{\mathbf{D}} = \mathbf{D}/\omega$, and $\mathbf{L}$ and $\mathbf{D}$ denote the lower triangle part and the diagonal part of the matrix $\mathbf{A}$, respectively, which can be easily calculated. The incomplete formulation of the matrix $\mathbf{M}^{-1}$ can be calculated by keeping the same sparsity as in the matrix $\mathbf{A}$. In our implementation, all these sparse matrices are stored in the compressed sparse row (CSR) format. In addition, the user's interactive editing of the urban model is decoupled from $\mathbf{A}$, and thus all these calculations of the preconditioner matrix $\mathbf{M}^{-1}$ can be precomputed for our simulation. Even in other applications where it cannot be precomputed, it still can be efficiently calculated since it is straightforward to be parallelized on the GPU.

Porosity Model vs. Solid-wall Boundary Conditions. Setting up appropriate boundary conditions is critical for the fluid simulation and can be very tricky. In this work, we use the Dirichlet condition to directly set the velocity for the inlet boundary, while for the outlet boundary we use the Neumann boundary condition to ensure that the derivative of the velocity (with respect to the normal direction of the outlet boundary) is equal to zero: $\partial\mathbf{u}/\partial\mathbf{n} = 0$. No-slip solid wall boundary conditions are enforced over the solid wall boundary: $\mathbf{u} = \mathbf{u}_{\text{solid}}$. In the pressure projection step, the pressure in the outlet cell is set to be zero, and a ghost pressure value in the inlet and solid wall cell is calculated by a Neumann boundary condition on the fly:

$$\partial p/\partial\mathbf{n} = 0. \tag{13}$$

We would like to particularly discuss more about the handling of the boundary condition of buildings which is very important in our application. If we treat it as a solid wall, besides the fact that its porosity effect cannot be included, extra inefficiency will also be caused by the solid-wall boundary condition enforced over the surfaces of the buildings. As described above, the wall boundary condition needs to be specially handled, and the geometry configuration of buildings will be constantly changed for interactive urban design. Moreover, the coefficient matrix $\mathbf{A}$ of the pressure Poisson equation set depends on the buildings' geometrical configuration since a ghost pressure value needs be calculated for the solid cells nearby the solid-wall

boundary according to Eq. (13). It turned out that each time the user edits the urban model, the coefficient matrix $\mathbf{A}$ and its preconditioner have to be recomputed. When the porosity model is used to handle buildings, an extra drag force will be enforced for each building cell and no solid-wall boundary conditions are needed anymore. Since the coefficient matrix $\mathbf{A}$ does not depend on the drag force term, this problem is solved. The coefficient matrix $\mathbf{A}$ and its preconditioner can be precomputed to improve the computational efficiency.
We use the forward Euler method to numerically solve the drag force step:

$$\frac{\partial \mathbf{u}}{\partial t} = \frac{1}{\rho_{\mathsf{air}}}\mathbf{f_d} = -C_d a((1-\phi)/(\phi+e))^b U\mathbf{u}\,, \tag{14}$$

$$\mathbf{u}_{n+1} = (1-\Delta t C_d a((1-\phi)/(\phi+e))^b U)\mathbf{u}_n\,, \tag{15}$$

in which the constraint $(1-\Delta t C_d a((1-\phi)/(\phi+e))^b U) \geq 0$ is enforced and any negative values will be replaced by zeros. It is clear that this drag force term damps the velocity field and for the zero-porosity case (i.e., solid cell), the speed is always ensured to be zero.

Urban model resolving. To efficiently handle the buildings, trees, and the boundary setting of the simulation domain, we use the marked-cell method to assign a label value to each grid cell to show its type. The possible types for boundary cells include inlet, outlet, and solid wall, which can be easily set. The possible types for internal cells include buildings, trees and air, while its setting will take some effort because we need to resolve the urban model first and the urban model will be constantly edited by the designers in the interactive mode or keep evolving in the automatic optimization mode.
A two-pass strategy is adopted. For a given geometry object (building or tree) with arbitrary shape (its surface is represented as a triangle mesh), an axis-align bounding box (AABB) is generated in the first pass. It can be easily judged if some grid cell $\mathbf{c}$ is inside its AABB or not (we use cell center points in our implementation for simplicity). If grid cell $\mathbf{c}$ is inside its AABB, we use the ray-casting algorithm and the odd-even rule to determine if this grid cell is inside this object or not in the second pass. The ray is used to calculate its intersection points with all the surface mesh triangles of this building if any. We count the number of intersection points, and an odd number means that this grid cell is inside this object. The intersection of ray and triangle is calculated with the classical Möller–Trumbore algorithm (Möller and Trumbore 1997).
The grid cells can be classified into three types: Inside-architecture cells, outside-architecture cells, and the cut-cells by the surfaces of the architectures. Particular treatment needs to be done with the cut-cells to better support sub-cell resolution for architecture surfaces, continuous optimization computation, simplicity and efficiency (Du et al. 2020). In this paper, we propose to use the volume ratio outside architecture surface as the porosity values of the cut-cells. Different from the plane fitting method using least square algorithm based on the signed distance calculated on the quadrature points in cut-cells adopted from Du et al. (Du et al. 2020), we adopt an alternative method where a uniform subdivision to the desired resolution is performed in the cut-cells, and the ratio of the subdivision grid points outside the architectures is used as the volume ratio approximation instead of the linear plane approximation. This is illustrated in Figure 1.

4 TOWARDS SIMULATION-INSPIRED DESIGN

4.1 Interactive Design

Because of the fact that most of the urban designers are not so familiar with fluid simulation, an interface tool with popular urban design software needs to be developed where the designers can easily run the flow simulation in the way that they are familiar with.
We developed a plugin to use *UrbanFlow* with *Rhino* widely used in computational architecture. This supports designers to perform the interactive design of urban models directly in a software system they

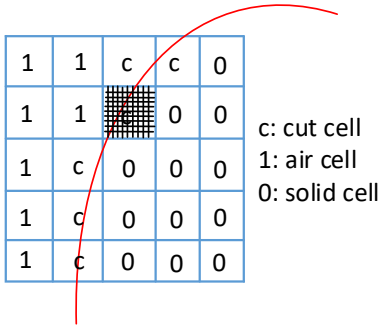

Figure 1: Cut-cell treatment: The red curve denotes the solid/building boundary, porosity of 0 denotes the cell inside and porosity of 1 denotes the cell outside the object.

are familiar with. An initial urban model can be automatically generated from the procedural urban model plugin or just a non-procedural urban model can be imported, and then designers perform edits. Whenever the designer would like to know about the flow distribution under some geometrical configuration, the urban flow simulation can be called via the developed urban flow plugin. Then, the geometry information of the urban model will be transferred to our fast simulator, and the simulation results are transferred back to *Rhino* for visualization/rendering. Besides the urban flow plugin, a simple procedural urban model plugin has been developed that can produce random urban models for testing.

4.2 Automatic Design Optimization

Objective Function. Optimizing air flow is the main focus of this work and several objective functions have been defined for different specific applications. Generally, the objective function is defined as

$$L(\mathbf{u}) = \sum_i ||F_i(\mathbf{u}) - F_i(\mathbf{u_t})||_2, \quad \mathbf{u} = \Phi(\theta), \tag{16}$$

in which $\mathbf{u_t}$ denotes the desired velocity field and F_i denotes a function of the velocity field, Φ denotes the forward simulation operator, and θ denotes the design parameter. The index i corresponds to a specific target region.

Gradient-descent. The standard gradient-descent optimization algorithm is implemented in this work. One key issue is the computation of the gradient of objective functions with respect to the design parameters. A simple strategy is used in this paper, where the well-designed fast forward simulation is used to calculate the gradient information. Its mathematical formulation is given by

$$\theta^* := \mathsf{argmin}_\theta \{L(\theta)\}, \tag{17}$$

$$\theta^{n+1} = \theta^n + \lambda \frac{\partial L}{\partial \theta}, \quad \frac{\partial L}{\partial \theta_i} \approx \frac{L(\theta_i + \varepsilon) - L(\theta_i)}{\varepsilon}, \tag{18}$$

in which θ denotes a vector with N design parameters θ_i, L denotes the objective function, and λ denotes the step size. The partial derivative with respect to the design parameter θ_i is calculated by forward differences, where a small increment ε is added to the specific parameter. In addition, the wind direction perturbation is an important factor that needs to be taken into account especially for outdoor ventilation.

Discussion. This work is mainly targeted at the early stage urban design application where buildings are usually represented with simple shapes and small number of design parameters are needed. Therefore, we

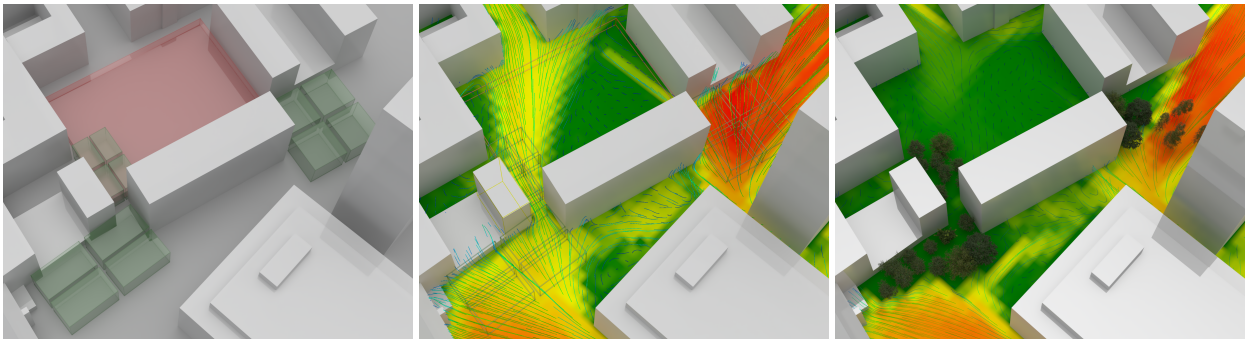

Figure 2: Simulation of the urban flow at the windy public "Neumarkt" square in Bielefeld, Germany (middle). The outdoor comfort could be improved by planting trees in strategic locations which are determined using our system. The area of interest (red) and possible plantation locations (green) are shown (left). A possible landscaping following the optimized porosity is illustrated (right).

adopt the light-weight numerical method based on forward simulation to calculate gradient instead of the more complicated advanced methods such as adjoint method, auto-differentiation, etc. These advanced methods will take extra cost by themselves and thus are more suitable for the applications with large number of design parameters.

5 IMPLEMENTATION

All results presented in this paper are computed on a workstation with an Intel Xeon CPU E5-2699, an NVIDIA TITAN V GPU, and 256 GB main memory. The flow simulation solver, urban model handling, and the generation and extraction of the streamlines are implemented in C++ and parallelized using CUDA. In the interactive mode, they are compiled into a dynamic link library (DLL) that is called by the *Rhino* interfaces/plugins which are implemented in C# and *RinoCommon*. The calculation of the matrix condition number is done with *MATLAB*. The optimization routine is also implemented with our in-housed code in C++ and CUDA, and no pre-existing packages are used. The visualization/rendering is mainly done with *Rhino*.

6 CASE STUDIES FOR URBAN DESIGN APPLICATIONS

We have carefully validated our fluid simulator including its porosity model and the choice of the AI preconditioner. This can be found in the appendix.

6.1 Improving Wind Comfort by Trees Planting

Figure 2 (left) shows the digital urban model of the "Neumarkt" square in the city of Bielefeld in Germany. This square is known as an extremely windy place within the city as strong winds are coming from southwest (bottom left in our illustrations). First, we run an urban flow simulation using real wind data obtained from the Global Wind Atlas (http://globalwindatlas.info/). A uniform grid with the resolution of $100 \times 100 \times 30$ is used and each grid cell contains a volume of $3.5 \times 3.5 \times 3.5$ m^3. The outflow boundary condition is enforced over right and top boundaries. Over the inflow (left and bottom boundaries), a vertical wind inflow with a logarithmic profile $u_z = u_* / \kappa \ln(z/z_0)$ is enforced, in which u_z denotes the speed at altitude z, $\kappa \approx 0.41$ the dimensionless Von Kármán constant, z_0 the roughness length of 0.5 m, and $u_* = 0.53$ ms^{-1} the friction speed calculated by data-fitting with real meteorological data of Bielefeld.
As shown in Figure 2 (middle), uncomfortable wind speeds are present on the square. A natural question is

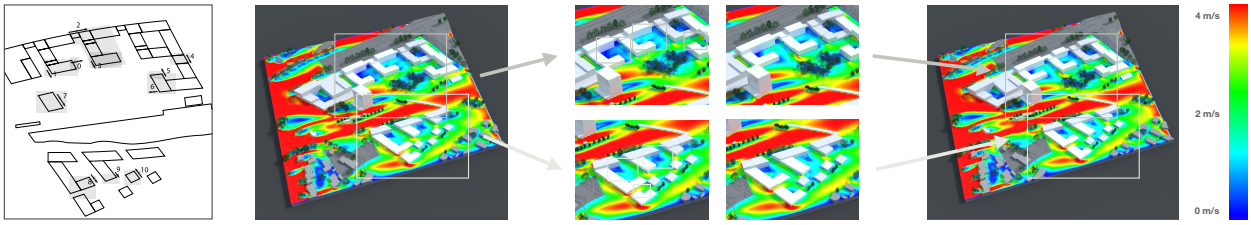

Figure 3: Illustration of *UrbanFlow*'s optimization process: After the identification of DOFs by a professional architect familiar with the initial design (left), a forward simulation is carried out to identify target areas in which the wind speed should be improved (second and third images from the left). After the gradient descent method converges, which computes the design adaptations, the effect of the new design is shown (fourth and fifth image from the left). It avoids heat pockets without compromising the overall outdoor comfort. The wind speed is illustrated using a color map. Please note, that low wind speeds (highlighted in blue) correspond to potential heat pockets.

if we can overcome this issue by planting trees. After some interactive operations shown in our supplemental video, we have been able to identify a design solution of planting trees that can clearly improve the windy square problem. The urban flow simulation results for the improved situation after planting trees is shown in Figure 2 (right). To test the robustness of this design solution under perturbations of the main wind direction, two more simulations with $\pm 10^{\circ}$ wind direction perturbations have been carried out as shown in Figure 5. It can be observed that the right-bottom corner of the square can always be well improved, while the left-top corner is more complicated.

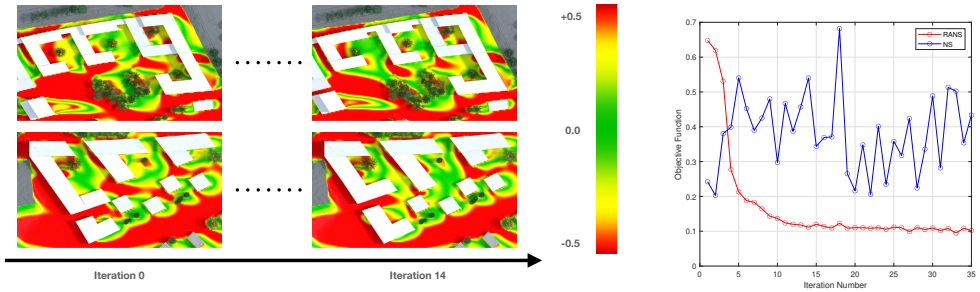

Figure 4: Illustration of the result of the optimization routine (left) and the development of the objective function during the optimization process (right). The deviation from the target wind speed of 0.55 m/s is illustrated using a color map.

6.2 Avoiding Heat Pockets in Early Design Stages

We demonstrate the practical usability of *UrbanFlow*'s optimization routine by contributing to an ongoing architectural competition based on an urban design by *BLAUWERK*, a well established architectural firm in Munich specialized on residential and office buildings, and *bauchplan*, a landscape architectural firm mostly working on designing public spaces. Recently, these firms have jointly worked on a design for a new residential quarter on a 44 000 m^2 area in Nürnberg-Gebersdorf in Germany. One of the design objectives given to the architects has been to prevent the formation of heat pockets in high density areas with low ventilation. The particular challenge is to compute small design adaptations in order to increase outdoor ventilation in a way that the formation of heat pockets is avoided while at the same time no uncomfortable windy areas should be created. Please note, that in 2022 this design draft was awarded a third price in an international competition organized by the *WBG Nürnberg GmbH* in cooperation with the *Bavarian State Ministry of Housing, Building and Transport* and the *Bavarian State Ministry for the Environment and*

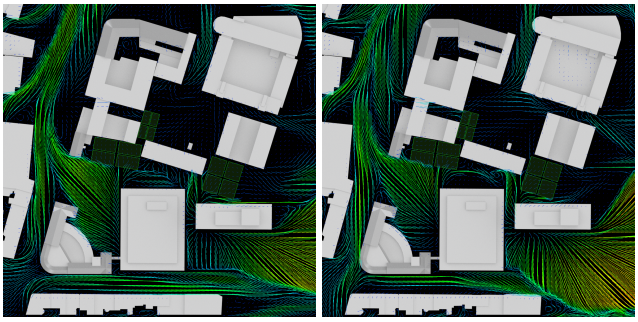

Figure 5: Test of the tree planting solution against perturbations of the main wind direction (30°). Simulations using wind directions of 20° (left) and 40° are shown.

Consumer Protection underlying the importance not change the visual appearance too much. In order to keep the spirit and the visual impression of the design and do not change its key properties too much (e.g., the overall amount of office space), only small adaptations should be made. The design is shown in Figure 3 in which the critical areas where heat pockets are expected are surrounded with rectangular frames (second image from the left). An architect who is familiar with this design systematically identified eleven degrees of freedom (DOFs) which can be easily changed in the design draft. These DOFs are shown in Figure 3 (left). Moreover, maximum values for the adaptation of all DOFs have been defined to ensure physical constraints so that, e.g., buildings cannot collide or overlap. An objective function

$$L(\tilde{\mathbf{u}}) = \sum_{i=1}^{N} (\tilde{u_i} - u_t)^2 \tag{19}$$

has been defined in which $N = 6$ denotes the number of target regions under consideration in which heat pockets or inconvenient windy area can occur. Please note, that among these six target regions we do not only consider regions in which heat pockets can occur but also regions which are potentially exposed to inconveniently high wind speeds. This is especially important in this application as strong wind blow from the west side because there is mostly undeveloped land. For illustration, the six target regions are surrounded with rectangular frames in Figure 3 (middle). In Eq. (19), $\tilde{\mathbf{u}}$ denotes the vector of all average wind speeds $\tilde{u_i}$ (each corresponding to the i-th target region) and u_t denotes the ideal average wind speed which we set to 0.55 m/s which is sufficient to avoid heat pockets without compromising outdoor comfort. Driven by this objective function, the average wind speed should automatically approach the ideal wind speed u_t. However, please note that the average wind speed of some regions can naturally not converge to the ideal wind speed. Hence, the value of the objective function does not have to be zero when converged. In each iteration of the optimization process, the wind field is computed using a forward simulation using a uniform grid resolution of $450 \times 300 \times 40$ and a cell volume of $1 \times 1 \times 1$ m^3. A time step of 0.2 s is applied for these forward simulation and about 2 000 time steps are computed per forward simulation. The gradient descent algorithm then applies an optimization step size of $\lambda = 1.0$ and the small increment is set to $\varepsilon = 0.1$.
The optimization process is illustrated in Figure 4 which shows that the gradient descent optimizer converges after 14 iterations using the RANS solver. According to Eq. (18), we have to compute 12 (i.e.,number of parameters plus one) forward simulations in each iteration of the gradient descent routine. This results in a computation time of about 2 hours for all 14 iterations on a regular single workstation. In Figure 4 (right), we have also shown what happens if we use the standard NS formulation instead of RANS which only results in oscillations not leading to convergence of the optimization routine. The overall result of the optimization process is shown in Figure 3 demonstrating a significant improvement of the initial design with just minor adaptations necessary. The formation of heat pockets is avoided where possible without compromising outdoor comfort and maintaining the nature of the design. After obtaining the design solution, forward simulations with small perturbations of the main wind direction are carried out to verify robustness.

7 DISCUSSION, LIMITATIONS, AND FUTURE WORK

In this contribution, we have demonstrated how fluid flow simulation can contribute to intelligent decision making in architectural design. This is based on our *UrbanFlow* framework which combines an efficient Eulerian fluid simulator with a unified porosity model to encode digital urban models containing objects such as buildings and trees. *UrbanFlow* has been used in the context of two real-world design challenges improving the wind comfort on a square in Bielefeld and for avoiding heat pockets in a current project in Nürnberg. While the wind comfort in the first application has been improved by planting trees based on design decisions obtained by running our flow simulator, the optimization routine of *UrbanFlow* has been applied in the second application resulting in the computation of a design adaptation without a human in the loop. These practical examples showcase the usefulness of simulation-inspired design decisions in architecture and urban planning. While the optimization routine keeps the overall structure on the initial design, from a technical perspective, key objective of the design such as the amount of office space, are not strictly enforced as hard constraints. Overcoming this limitation by extending our work with a constraint solver is left for future work. In general, we consider *UrbanFlow* as a first step towards the integration of physics-simulations in the design process of architects opening up an interesting avenue of future work including applications such as designing public areas in a way that noise in minimized.

ACKNOWLEDGEMENTS

The authors are thankful to Helmut Pottmann for his advice and domain expertise. Torsten Hädrich provided his reference code which is gratefully acknowledged. This work has been partially supported by KAUST baseline funding (grant BAS/1/1679-01-01). The constructive suggestions and comments from the anonymous reviewers that improved the manuscript are gratefully acknowledged.

A VALIDATION OF THE FLUID SIMULATOR

As a standard benchmark example, we use a Kármán vortex street example simulating the stream around a cylinder to validate the accuracy of our solver, and the following widely-used analytical formula valid for $250 < Re < 2.5 \cdot 10^5$ is used as the ground truth:

$$St = 0.198\,(1 - 19.7/Re)\,, \qquad f = St \cdot U/L\,, \tag{20}$$

in which St denotes the Strouhal number, Re denotes the Reynolds number, and U denotes wind speed, f is the frequency of the vortex shedding, L is the characteristic length and equal to the diameter of the cylinder (https://thermopedia.com/content/1247/). The calculated results are shown in the Figure 6. For this simulation, the bottom boundary is set to be inlet, the top boundary is outlet, and both side boundaries are solid walls. The grid resolution is 512×768, $\Delta x = \Delta y = 2 \cdot 10^{-3}$ m, $\Delta t = 1 \cdot 10^{-4}$ s, the diameter of the circular obstacle is 0.046 m. The inlet wind speed varies from 2 m/s to 20 m/s, thus the corresponding Reynolds number ranges from $5.86 \cdot 10^3$ to $5.86 \cdot 10^4$. As shown in Figure 6 our numerical results match the theoretical prediction very well.

B VALIDATION OF THE POROSITY MODEL

According to its classical definition, porosity can be quantified by a parameter $\phi = 1 - V_s/V$, where V is the total volume and the V_s is the non-empty volume. Please note, that this definition assumes that gas communicates between the pores and the surrounding volume. In practice, this means that the pores must not be closed cavities.

Trees are canonical examples for porous objects. A common model for the computation of the drag force for tree objects is given by $\mathbf{f}_d = -\rho_{air} C_d \mathrm{LAD} |\mathbf{u}| \mathbf{u}$ with leaf area density (LAD). For buildings, LAD can, e.g.,

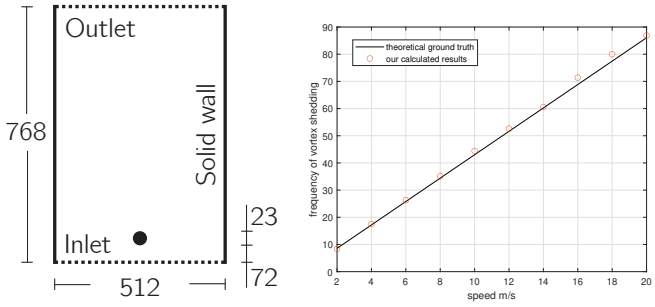

Figure 6: Kármán vortex street benchmark test case, frequency of vortex shedding vs. wind speed, geometric and simulation configuration(left).

be replaced by $a((1-\phi)/\phi)^b$ with $a = 0.62$ and $b = 2.5$. A drag force coefficient of $C_d = 1.0$ corresponds to a smooth surface.

We set up a validation experiment shown in Figure 7. Form bottom to top, wind a blowing with a speed of 2 m/s and 5 m/s for Test 1 and Test 2 respectively. only the middle part of the bottom boundary is set to be inlet, its left and right parts are set to be outlet. Both, the side boundaries are set to be no-slip solid boundary. V_{out} measures the averaged wind speed of the top outlet boundary. The different porosity configurations are done by changing the radius of the circular obstacles. The left sub-figure of Figure 7 shows the dependence of V_{out} on the porosity. In this simulation, a grid resolution of 512×640, $\Delta x = \Delta y = 1$ m, and $\Delta t = 0.1$ s are used. The results obtained using our porosity model are validated against the results calculated by using no-slip boundary conditions which are treated as the ground truth. It can clearly be observed that our porosity matched the ground truth very well.

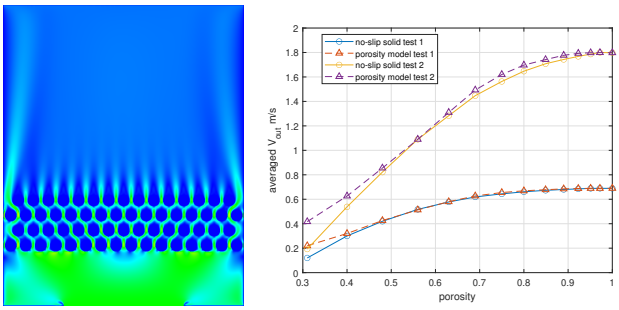

Figure 7: Benchmark test validating the porosity model.

C EFFICIENCY OF THE AI PRECONDITIONER

We compare the AI preconditioner against popular preconditioners including Jacobi, SSOR, incomplete Cholesky factorization (IC) and modified incomplete Cholesky factorization (MIC). Since all these pre-conditioners can be precomputed for our simulation, we only evaluate their application process. The AI preconditioner only requires one matrix-vector multiplication in its application process as the Jacobi pre-conditioner in each iteration, and is straightforward to parallelize. It is well known that the popular MIC preconditioner cannot be easily parallelized because of its inherently serial back substitutions. Therefore, we will mainly compare their convergence rate via measuring the condition number of the preconditioned matrix $\mathbf{M}^{-1}\mathbf{A}$ as shown in Table 2, in which AI1 and AI2 denote the AI preconditioners using first and

Table 2: Comparison of different CG preconditioners.

	CG	Jacobi	AI1	AI2
K	2.3501e+06	2.3463e+06	9.5582e+05	6.0811e+05

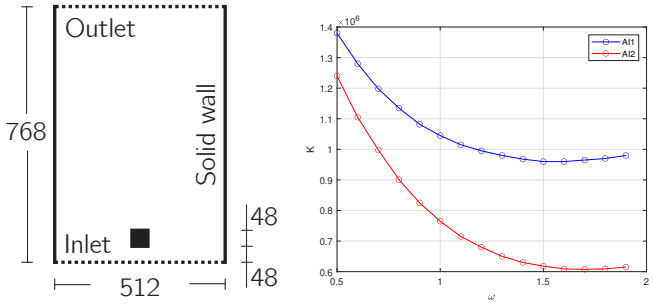

Figure 8: Illustration of the dependence of the condition number on ω using the approximate inverse preconditioner.

second order approximation, and their condition number depends on the value of ω as show in Figure 8. The optimal value of ω is between 1.6 and 1.7.

The calculation of the condition numbers for SSOR, IC, and MIC involves the inverse matrix calculation of a large scale matrix. It is too time and too memory expensive to be calculated in our workstation for the current resolution. Thus, we perform another sparser simulation (128×192) where SSOR, IC, and MIC are included in the comparison of the condition number shown in Table 3.

Table 3: Comparison of different CG preconditioners using a sparser grid resolution (128×192).

	CG	Jacobi	AI1	AI2
K	$1.4516 \cdot 10^5$	$1.4422 \cdot 10^5$	$5.8736 \cdot 10^4$	$3.7385 \cdot 10^4$
	SSOR	IC	MIC	
K	$1.2234 \cdot 10^4$	$2.1118 \cdot 10^4$	$4.7955 \cdot 10^3$	

The condition number of the matrix $\mathbf{M}^{-1}\mathbf{A}$ is defined as the ratio of its maximum and minimum eigenvalues. The lower condition number of an arbitrary symmetric positive definite matrix is, the faster the CG algorithm converges. Although the AI preconditioner does not have the best convergence rate, it has the best balance between convergence rate and the parallel computational cost of each iteration. We employ the converge condition

$$(\mathbf{M}^{-1}\mathbf{r}) \cdot \mathbf{r} / \|\mathbf{b}\|^2 < 10^{-5}, \tag{21}$$

in which $\mathbf{r}$ denotes the residual. CG without any preconditioner converged after 350 iterations which took 0.62 s. Using the Jacobi preconditioner, CG converged after 330 iterations which took 0.55 s. Using AI1, only 5 iterations are required for convergence which took 0.01 s. AI1 and AI2 show comparable performance. In these times comparisons, we do not include other preconditioners such as SSOR because they need backward substitutions which is suboptimal for the parallelization (Chu, Zafar, and Yang 2017).

D SUPPLEMENTAL VIDEO

Please use the following (publicly accessible) YouTube link to watch the video.

https://youtu.be/FVNx3KzM13o

REFERENCES

Aanjaneya, M., C. Han, R. Goldade, and C. Batty. 2019. "An Efficient Geometric Multigrid Solver for Viscous Liquids". *Proceedings of the ACM on Computer Graphics and Interactive Techniques* vol. 2 (2), pp. 1–21.

Adamek, K., N. Vasan, A. Elshaer, E. English, and G. Bitsuamlak. 2017. "Pedestrian level wind assessment through city development: A study of the financial district in Toronto". *Sustainable cities and society* vol. 35, pp. 178–190.

Al-Saadi, S. N., and A. K. Shaaban. 2019. "Zero energy building (ZEB) in a cooling dominated climate of Oman: Design and energy performance analysis". *Renewable and Sustainable Energy Reviews* vol. 112, pp. 299–316.

Ament, M., G. Knittel, D. Weiskopf, and W. Strasser. 2010. "A parallel preconditioned conjugate gradient solver for the Poisson problem on a multi-GPU platform". In *2010 18th Euromicro Conference on Parallel, Distributed and Network-based Processing*, pp. 583–592. Institute of Electrical and Electronics Engineers.

Chen, Y., Z. Tong, and A. Malkawi. 2017. "Investigating natural ventilation potentials across the globe: Regional and climatic variations". *Building and Environment* vol. 122, pp. 386–396.

Chorin, A. J., J. E. Marsden, and J. E. Marsden. 1990. *A mathematical introduction to fluid mechanics*, Volume 168. Springer.

Chu, J., N. B. Zafar, and X. Yang. 2017. "A Schur complement preconditioner for scalable parallel fluid simulation". *ACM Transactions on Graphics (TOG)* vol. 36 (4), pp. 1.

Deaton, J. D., and R. V. Grandhi. 2014. "A survey of structural and multidisciplinary continuum topology optimization: post 2000". *Structural and Multidisciplinary Optimization* vol. 49 (1), pp. 1–38.

Du, T., K. Wu, A. Spielberg, W. Matusik, B. Zhu, and E. Sifakis. 2020. "Functional optimization of fluidic devices with differentiable stokes flow". *ACM Transactions on Graphics (TOG)* vol. 39 (6), pp. 1–15.

Feng, W., Q. Zhang, H. Ji, R. Wang, N. Zhou, Q. Ye, B. Hao, Y. Li, D. Luo, and S. S. Y. Lau. 2019. "A review of net zero energy buildings in hot and humid climates: Experience learned from 34 case study buildings". *Renewable and Sustainable Energy Reviews* vol. 114, pp. 109303.

Foster, N., and R. Fedkiw. 2001. "Practical animation of liquids". In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 23–30.

Gavriil, K., R. Guseinov, J. Pérez, D. Pellis, P. Henderson, F. Rist, H. Pottmann, and B. Bickel. 2020. "Computational design of cold bent glass façades". *ACM Transactions on Graphics (TOG)* vol. 39 (6), pp. 1–16.

Hanjalić, K., and B. Launder. 2020. "Eddy-Viscosity Transport Modelling: A Historical Review". In *50 Years of CFD in Engineering Sciences*, pp. 295–316. Berlin, Springer.

Helfenstein, R., and J. Koko. 2012. "Parallel preconditioned conjugate gradient algorithm on GPU". *Journal of Computational and Applied Mathematics* vol. 236 (15), pp. 3584–3590.

Irving, S., and D. J. Clements-Croome. 2005. *Natural ventilation in non-domestic buildings*. Chartered Institution of Building Services Engineers.

Jones, W. P., and B. E. Launder. 1972. "The prediction of laminarization with a two-equation model of turbulence". *International journal of heat and mass transfer* vol. 15 (2), pp. 301–314.

Kang, G., J.-J. Kim, D.-J. Kim, W. Choi, and S.-J. Park. 2017. "Development of a computational fluid dynamics model with tree drag parameterizations: Application to pedestrian wind comfort in an urban area". *Building and Environment* vol. 124, pp. 209–218.

Kaseb, Z., M. Hafezi, M. Tahbaz, and S. Delfani. 2020. "A framework for pedestrian-level wind conditions improvement in urban areas: CFD simulation and optimization". *Building and Environment* vol. 184, pp. 107191.

Kenjereš, S., and B. ter Kuile. 2013. "Modelling and simulations of turbulent flows in urban areas with vegetation". *Journal of Wind Engineering and Industrial Aerodynamics* vol. 123, pp. 43–55.

Liu, W., T. Zhang, Y. Xue, Z. J. Zhai, J. Wang, Y. Wei, and Q. Chen. 2015. "State-of-the-art methods for inverse design of an enclosed environment". *Building and Environment* vol. 91, pp. 91–100.

Lucon, O., D. Ürge-Vorsatz, A. Z. Ahmed, H. Akbari, P. Bertoldi, L. F. Cabeza, N. Eyre, A. Gadgil, L. D. Harvey, and Y. Jiang. 2014. "Buildings". vol. IPCC Fifth Assessment Report (AR5).

Manceau, R. 2021. *Industrial codes for CFD*. Poitiers, France.

McAdams, A., E. Sifakis, and J. Teran. 2010. "A Parallel Multigrid Poisson Solver for Fluids Simulation on Large Grids.". In *Symposium on Computer Animation*, pp. 65–73.

Möller, T., and B. Trumbore. 1997. "Fast, minimum storage ray-triangle intersection". *Journal of Graphics Tools* vol. 2 (1), pp. 21–28.

Pottmann, H., A. Asperl, M. Hofer, and A. Kilian. 2008. *Architectural geometry*. Bentley Institute Press.

Pottmann, H., M. Eigensatz, A. Vaxman, and J. Wallner. 2015. "Architectural geometry". *Computers & graphics* vol. 47, pp. 145–164.

Qu, Z., X. Zhang, M. Gao, C. Jiang, and B. Chen. 2019. "Efficient and conservative fluids using bidirectional mapping". *ACM Transactions on Graphics (TOG)* vol. 38 (4), pp. 1–12.

Ramponi, R., B. Blocken, B. Laura, and W. D. Janssen. 2015. "CFD simulation of outdoor ventilation of generic urban configurations with different urban densities and equal and unequal street widths". *Building and Environment* vol. 92, pp. 152–166.

Rando, E., I. Muñoz, and G. Patow. 2016. "Interactive Low-Cost Wind Simulation for Cities". In *Proceedings of the Eurographics Workshop on Urban Data Modelling and Visualisation*, UDMV '16, pp. 57–63. Goslar, DEU, Eurographics Association.

Saad, Y. 2003. *Iterative methods for sparse linear systems*. SIAM.

Selle, A., R. Fedkiw, B. Kim, Y. Liu, and J. Rossignac. 2008. "An unconditionally stable MacCormack method". *Journal of Scientific Computing* vol. 35 (2), pp. 350–371.

Selle, A., N. Rasmussen, and R. Fedkiw. 2005. "A vortex particle method for smoke, water and explosions". In *ACM SIGGRAPH 2005 Papers*, pp. 910–914.

Sharma, A., H. Mittal, and A. Gairola. 2018. "Mitigation of wind load on tall buildings through aerodynamic modifications". *Journal of Building Engineering* vol. 18, pp. 180–194.

Stam, J. 1999. "Stable fluids". In *Proceedings of the 26th annual conference on Computer graphics and interactive techniques*, pp. 121–128.

Thordal, M. S., J. C. Bennetsen, and H. H. H. Koss. 2019. "Review for practical application of CFD for the determination of wind load on high-rise buildings". *Journal of Wind Engineering and Industrial Aerodynamics* vol. 186, pp. 155–168.

Wilcox, D. C. 2006. *Turbulence Modelling for CFD, 3rd edition*. CA, DCW industries La Canada.

Zehnder, J., R. Narain, and B. Thomaszewski. 2018. "An advection-reflection solver for detail-preserving fluid simulation". *ACM Transactions on Graphics (TOG)* vol. 37 (4), pp. 1–8.

Zhang, X., R. Bridson, and C. Greif. 2015. "Restoring the missing vorticity in advection-projection fluid solvers". *ACM Transactions on Graphics (TOG)* vol. 34 (4), pp. 1–8.

Zhongming, Z., L. Linong, Z. Wangqiang, and L. Wei. 2021. "AR6 Climate Change 2021: The Physical Science Basis". vol. IPCC Sixth Assessment Report (AR6).

AUTHOR BIOGRAPHIES

DAOMING LIU is a Postdoctoral Researcher working with Professor Dominik L. Michels and Professor Helmut Pottmann at the KAUST Visual Computing Center. He obtained his Ph.D. in Communication and Information Systems from the University of the Chinese Academy of Sciences. During his Ph.D. studies, he spent one year working in the Transport Phenomena Group at TU Delft. Before joining KAUST, he worked at the ShanghaiTech University as a Research Assistant Professor. His main fields of research are physics-based simulation and geometric computing, and their applications in various fields, such as in computer graphics, scientific visualization, computational design and digital fabrication, and robotics. His past research was centered around fluid flow phenomena and thin solid structures.

FLORIAN RIST is a Research Scientist at the KAUST Visual Computing Center. He is a founding member of the Center for Geometry and Computational Design at TU Vienna and currently holds a double affiliation as a KAUST Research Scientist and as a Senior Scientist at the Institute of Arts and Design at TU Vienna. He studied at TU Munich and at the Tokyo University of Technology, and is a licensed architect and member of the Chamber of Architects in Bavaria, Germany. He worked as an architect at various well-known international architectural firms, including Wöhr Heugenhauser Architects in Munich and Gerber Architects in Dortmund. His main fields of research are computational and parametric design, complex free form geometry in architecture, and digital and computational fabrication.

DOMINIK L. MICHELS is an Associate Professor of Computer Science and Applied Mathematics at KAUST, and the Principal Investigator of the KAUST Computational Sciences Group. Together with his team he develops principled computational methods for the accurate and efficient simulation of natural phenomena solving practically relevant problems in scientific and visual computing. Previously, he joined Stanford University heading the High Fidelity Algorithmics Group within the Max Planck Center for Visual Computing and Communication. Prior to this, he did postdoctoral studies in Computing and Mathematical Sciences at the California Institute of Technology, and studied at the University of Bonn from where he received a B.Sc. in Computer Science and Physics, a M.Sc. in Computer Science, and a Ph.D. from the Faculty of Mathematics and Natural Sciences.