Thesis for the Degree of Licentiate of Engineering

Applications of Finite Element Simulation and Mixed Reality in Architecture and Built Environment

Carl Lundholm

Division of Applied Mathematics and Statistics
Department of Mathematical Sciences
Chalmers University of Technology and University of Gothenburg
Gothenburg, Sweden 2019

Applications of Finite Element Simulation and Mixed Reality in Architecture and Built Environment
Carl Lundholm

Department of Mathematical Sciences
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg, Sweden
Tel: +46 (0)31 772 1000

Author e-mail: carlun@chalmers.se

Typeset with LaTeX.
Department of Mathematical Sciences
Printed in Gothenburg, Sweden 2019

Cover image: Holograms visualized with Microsoft's mixed reality glasses HoloLens. The holograms represent problem data and solutions to finite element formulations of a Laplace problem and an advection–diffusion problem. The green grid represents the computational mesh. The arrows represent an air flow. The green cloud represents a gas source. The spheres represent the gas concentration in the room. The problems have been defined and solved in the author's office with the PDE-solving mixed reality app HoloFEM.

Applications of Finite Element Simulation and Mixed Reality in Architecture and Built Environment

Carl Lundholm

Department of Mathematical Sciences
Chalmers University of Technology and University of Gothenburg

Abstract

This thesis presents applications in architecture and built environment that are based on the finite element method and mixed reality. The work consists of two main components: two projects presented in three papers. The first project (presented in Paper I) concerns an application in urban design, thus built environment on a *larger* scale. In this project, a model for efficiently evaluating configurations of buildings is presented along with numerical results. The key ingredient for making the model efficient is the use of cut finite element methods on overlapping meshes. Although the project in its current state does not involve mixed reality, there is a natural extension to it by considering virtual or mixed reality user interfaces. The second project (presented in Paper II and Paper III) is about a mixed reality software application (app) with potential use in HVAC-simulations (heating, ventilation, and air conditioning), thus built environment on a *smaller* scale. The app allows a user to interactively define and solve problems governed by differential equations in mixed reality, where the real world surroundings are used as input data and the differential equations are solved with the finite element method.

Keywords: augmented reality, CutFEM, FEniCS, finite element method, HoloLens, mixed reality, multimesh, multi-objective optimization, view computation, virtual reality

List of appended papers

Paper I Christian Valdemar Hansen, Anders Logg, Carl Lundholm
Multi-objective optimization of wind and view in urban design
Preprint

Paper II Anders Logg, Carl Lundholm, Magne Nordaas
Solving Poisson's equation on the Microsoft HoloLens
Published in Proceedings of VRST'17

Paper III Anders Logg, Carl Lundholm, Magne Nordaas
Solving differential equations in mixed reality
Preprint

My contribution to the appended papers:

Paper I: Part of modeling. Part of implementation and simulation of wind model. Implementation and simulation of view 2D model and optimization.

Paper II: Part of modeling. Implementation of treatment of problem data (except meshing) and assembly of the linear system. Design and implementation of visualization of holograms representing problem data and solution.

Paper III: Part of modeling. Extending the existing implementation of the mixed reality application from Poisson problems to time-dependent advection–diffusion problems.

Acknowledgements

I like to thank my supervisor Anders Logg for his ingenuity, support, and many contributions to our joint work. I also wish to thank Magne Nordaas and Christian Valdemar Hansen for collaboration and discussions. Thanks to members of the FEniCS-community: August Johansson, Benjamin Kehlet, Jørgen Dokken, Simon Funke, and many more. I like to thank CREAM Architects, Michael Patriksson, Ann-Brith Strömberg, and Stig Larsson for valuable input in discussions about the work presented in this thesis. Thanks to other faculty members, colleagues, and friends at the department for lectures, seminars, discussions, problem-solving sessions, lunches, social gatherings, and other fun stuff.

Thanks to non-departmental friends for a wide range of activities and discussion topics, warmth and friendship. Final thanks to my family for constant support and love, and for always welcoming me back to countryside weekends on Orust.

Carl Lundholm
Gothenburg, March 2019

Contents

Introduction

The finite element method has been applied in areas related to architecture and built environment ever since development started around the 1950s with the classical civil engineering application in structural analysis. It has during the decades since then become a valuable tool for engineers. See, e.g., [1, 2]. The application of mixed reality in architecture and built environment does not go back as long, but has seen an increase during the last years, since hardware development has started to catch up with the technology in terms of size and cost. See, e.g., [3, 4, 5, 6]

In this thesis the two concepts are combined. Two applications in architecture and built environment that make use of the finite element method and mixed reality are presented in three papers.

The first application, presented in Paper I, presents work in which *cut* finite element methods on overlapping meshes are used to aid in the design of an urban system, thus being an application in built environment on a larger exterior multi-building scale. The current work on this application does not involve mixed reality, but there exists a natural extension to it by considering a virtual or mixed reality user interface that allows a user to freely move around buildings to test and evaluate different configurations.

The second application, presented in Paper II and Paper III, concerns the development of a software application (app) for Microsoft's mixed reality glasses HoloLens. The app lets a user define and solve problems governed by partial differential equations using the real world geometry as spatial input to define a computational mesh for the finite element method which is used to solve the equations. Problem data and solutions are represented by holograms in a mixed reality environment. The app is aptly named HoloFEM and can be used for simple HVAC-simulations (heating, ventilation, and air conditioning), thus being an application in built environment on a smaller interior single-building scale.

The core of the thesis is the three appended papers. They are written and presented with the idea of being relatively self-contained, meaning that necessary theory is presented in them. The only prerequisite is some familiarity with the finite element method. For related works and literature, the reader is referred to the introduction and references sections of the appended papers. This introduction is concluded by short presentations of the appended papers.

Paper I *Multi-objective optimization of wind and view in urban design* presents a model for evaluating configurations of buildings in 2D. Two factors are taken into consideration when evaluating a configuration: the wind around the buildings, and the view from the buildings. The wind model is based on a cut finite element method for overlapping meshes. It is applied by encapsulating each

building in its own mesh and also having a background mesh of the domain without any buildings. This set-up only requires initial mesh generations and allows the same meshes to be used for evaluation of a very large number of building configurations, thus avoiding costly remeshing for new configurations. The view models, a simple one for 2D settings, and a more elaborate one for 3D settings, relate to concepts such as isovists and using different view weights depending on what type of object is seen. The models are used to define measures for wind and view that can be used to evaluate a configuration of buildings. These measures are used to formulate a multi-objective optimization problem that is implemented and solved for 2D settings.

Paper II *Solving Poisson's equation on the Microsoft HoloLens*, published in Proceedings of VRST'17, gives a brief presentation of the first version of the app. This version of the app can only handle Poisson problems.

Paper III *Solving differential equations in mixed reality* contains a far more elaborate presentation and description of the app. Here, the app has also been improved and extended to handle time-dependent advection–diffusion problems.

References

[1] K. K. GUPTA and J. L. MEEK, "A brief history of the beginning of the finite element method," *International Journal for Numerical Methods in Engineering*, vol. 39, pp. 3761 – 3774, 11 1996.

[2] L. T. Tenek and J. Argyris, *A brief history of FEM.* Dordrecht: Springer Netherlands, 1998, pp. 17–25. [Online]. Available: https://doi.org/10.1007/978-94-015-9044-0_2

[3] X. Wang and M. A. Schnabel, *Mixed reality in architecture, design and construction.* Springer, 2009.

[4] A. Bendinger, "New approaches in urban simulation and city planning using virtual reality tools," in *1º Congreso Internacional Ciudad y Territorio Virtual, Barcelona, 3, 14 y 15 Septiembre 2004*, 2004, pp. 67–74.

[5] V. Heuveline, S. Ritterbusch, and S. Ronnas, "Augmented Reality for Urban Simulation Visualization," *Preprint Series of the Engineering Mathematics and Computing Lab*, vol. 0, no. 16, 2011.

[6] X. Wang, M. J. Kim, P. E. D. Love, and S.-C. Kang, "Augmented Reality in built environment: Classification and implications for future research," *Automation in Construction*, vol. 32, pp. 1–13, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0926580512002166

Paper I

Multi-objective optimization of wind and view in urban design

Christian Valdemar Hansen, Anders Logg, Carl Lundholm

Abstract

We present methodology for evaluating settlement layouts. For a given settlement layout consisting of a number of buildings arbitrarily positioned on a piece of land, in the present study an island situated off the west coast of Sweden, the methodology allows for evaluation of the wind around the buildings and the view experienced from the buildings. The computation of wind is based on multimesh finite element methods. This allows each building to be embedded in a boundary-fitted mesh which can be placed freely in a fixed background mesh. The computation of view is based on a weighted and quantitative measure which can be efficiently computed by rasterization. The models for wind and view are used to formulate an optimization problem for which the solution is position and orientation of buildings in a landscape. Optimization of 2D settlement layouts is performed.

Keywords: urban design, settlement layout, multi-objective optimization, wind simulation, wind comfort, Stokes equations, finite element method, CutFEM, overlapping meshes, multimesh, FEniCS, view computation, visibility analysis, viewshed, isovist,

1 Introduction

When designing settlement layouts, architects need to take a large number of variables into consideration, such as economic interests, connections to infrastructure (roads, water and electricity), experienced quality of view, wind conditions, and many more, see, e.g., [1, 2, 3, 4, 5]. In this study, we examine how to evaluate and improve arbitrary settlement layouts by efficiently computing wind patterns around buildings and evaluating the view from them. The aim is to provide architects with a guiding computational tool that can be used as part of an iterative design process. The study is based on a challenging urban design problem presented in [6], where houses are to be placed on the island "Lilla Fjellsholmen" which is located off the west coast of Sweden.

There are several examples of urban CFD simulations, see, e.g., [7, 8, 9, 10, 11, 12]. A central issue when constructing a computational tool for urban design is that the tool should be able to quickly evaluate a multitude of suggested configurations, either as part of an optimization loop or as part of a manual (artistic) iterative design process. Standard numerical methods for flow computations require that a mesh or grid is generated around buildings, ground and other objects. Generating such a body-fitted mesh is a costly procedure and even more so when a large number of different meshes must be created, one for each configuration of the buildings. Instead, we use *multimesh* finite element methods, see, e.g., [13, 14, 15]. Multimesh finite element methods allow a problem to be posed on a collection of meshes that may overlap arbitrarily and which together define the computational domain, see Figure 1 for an example.

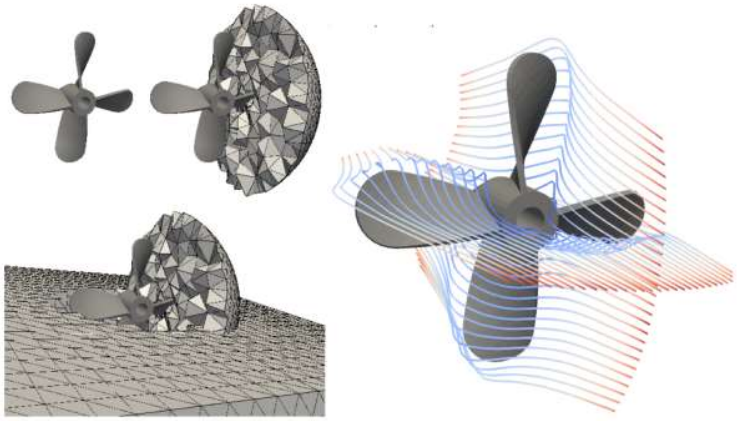

Figure 1: A collection of arbitrarily overlapping meshes definining the computational domain. Here, the surroundings of a 3D propeller immersed in water. Images courtesy of [16].

For an interesting application in shape optimization, see [17]. The overlapping meshes may be moved freely relative to one another, allowing a multitude of configurations without costly mesh generation. However, this flexibility comes at a price. If the finite element method is not carefully designed, certain configurations may lead to very ill-conditioned systems, low accuracy and even blow-up. Another concern is that these methods require integration over cut cells and interfaces, resulting in challenging computational geometry problems, when intersections and quadrature points must be computed efficiently and robustly.

For the computation of view, we construct quantitative measures for both 3D and 2D settings that can be used to evaluate the view for any given design. In addition to standard visibility analysis tools such as isovists and viewsheds, see, e.g., [18, 19, 20, 21], the 3D view measure may not only take into account how much surface and space that is seen, but also *what* is seen. This is done by incorporating object weights in the measure, allowing air, water, ground, buildings, herbage and other objects to have different influence on the view, an idea also presented by [22]. The use of these object weights is also the reason for using the term 'view' instead of, e.g., 'visibility'. The 3D view measure may be computed efficiently by rasterization. The 2D view measure is a binary version of the 3D measure that does not use object weights. This makes it equivalent to a simplified isovist.

Optimization in urban design and architecture is already in use, see, e.g., [23, 24, 25]. Our approach to multi-objective optimization is relatively basic and straightforward, and it is only numerically tested for 2D settlement layouts. The optimization in itself is not intended to be an essential contribution to optimization in urban design, but rather a suggestion or an example of how the wind and view models may be used to efficiently evaluate and improve settlement layouts.

In the remainder of this paper, we first present the models for wind and view, and how to use them in optimization in Section 2. Results are presented in Section 3, and discussed in relation to the methods in Section 4. We present our conclusions, discuss current limitations, and future work in Section 5.

2 Methods

We here present an overview of the methodology used to compute wind and view for the application under consideration: the design of a settlement layout (placement of houses) on a small island off the west coast of Sweden.

2.1 Computation of wind

To model the wind over the island and houses, a cut finite element method for Stokes equations on overlapping meshes is used. It is important to recall that Stokes equations are not *physically* suitable for modelling wind, since they describe flows with low Reynold's number, whereas wind may very well be turbulent. So why not use, e.g., the more physically appropriate Navier-Stokes equations instead? The reason is that we want to use cut finite element methods on overlapping meshes to avoid costly remeshing when testing several settlement layouts with respect to wind. At the moment, the most well-studied and advanced flow model for such methods is Stokes equations.

The method used is presented in detail by [16], but we give a brief description of it here: For a bounded domain $\Omega \subset \mathbb{R}^d$ with boundary $\partial\Omega$, the strong problem formulation for Stokes equations reads: Find the velocity $\mathbf{u} : \Omega \to \mathbb{R}^d$ and the pressure $p : \Omega \to \mathbb{R}$ such that

$$\begin{cases} -\Delta \mathbf{u} + \nabla p = \mathbf{f} & \text{in } \Omega, \\ \nabla \cdot \mathbf{u} = 0 & \text{in } \Omega, \\ \mathbf{u} = \mathbf{0} & \text{on } \partial\Omega, \end{cases} \tag{1}$$

where $\mathbf{f} : \Omega \to \mathbb{R}^d$ is a given right-hand side. To obtain a cut finite element formulation for Stokes equations on overlapping meshes, we start by considering two bounded domains $\widehat{\Omega}_0$ and $\widehat{\Omega}_1$, called predomains. Let $\Omega_0 := \widehat{\Omega}_0 \setminus \widehat{\Omega}_1$ and $\Omega_1 := \widehat{\Omega}_1$, with boundaries $\partial\Omega_0$ and $\partial\Omega_1$, respectively. We define the solution domain by $\Omega := \Omega_0 \cup \Omega_1$ and the joint boundary between Ω_0 and Ω_1 by $\Gamma := \partial\Omega_0 \cap \partial\Omega_1$. See Figure 2 for an example.

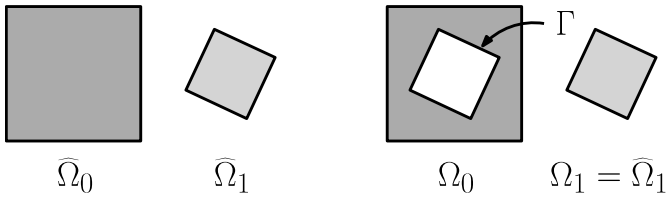

Figure 2: *Left*: Predomains $\widehat{\Omega}_0$ and $\widehat{\Omega}_1$. *Right*: Partition of solution domain Ω into Ω_0 and Ω_1 with intermediate boundary Γ.

To obtain a finite element formulation, the predomains, $\widehat{\Omega}_0$ and $\widehat{\Omega}_1$, are tessellated to create the meshes $\widehat{\mathcal{K}}_{h,0}$ and $\widehat{\mathcal{K}}_{h,1}$, respectively, where h denotes mesh size. We call $\widehat{\mathcal{K}}_{h,0}$ the background mesh, and $\widehat{\mathcal{K}}_{h,1}$ the overlapping mesh. See Figure 3.

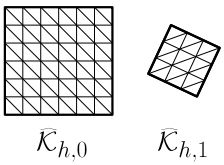

Figure 3: Background mesh $\widehat{\mathcal{K}}_{h,0}$ and overlapping mesh $\widehat{\mathcal{K}}_{h,1}$ resulting from a triangulation of the predomains.

The following subdomains will also be useful. For $i = 0, 1$,

$$\Omega_{h,i} := \bigcup_{K \in \widehat{\mathcal{K}}_{h,i}, \overline{K} \cap \Omega_i \neq \emptyset} K, \quad \omega_{h,0} := \bigcup_{K \in \widehat{\mathcal{K}}_{h,0}, \overline{K} \cap \overline{\Omega}_1 = \emptyset} K \tag{2}$$

For an example of the domains defined in (2), see Figure 4.

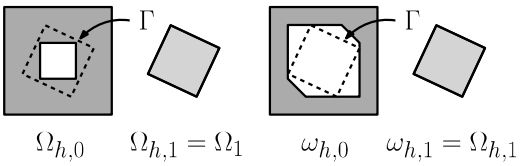

Figure 4: Subdomains $\Omega_{h,i}$ and $\omega_{h,i}$, for $i = 0, 1$. Subdomains with $i = 0$ are slightly darker than those with $i = 1$.

The desired finite element space may then be constructed in three steps:

1. For $i = 0, 1$, define a product space $\widehat{\mathbf{V}}_{h,i} \times \widehat{Q}_{h,i}$ for each mesh, $\widehat{\mathcal{K}}_{h,i}$, where $\widehat{\mathbf{V}}_{h,i}$ and $\widehat{Q}_{h,i}$ are finite element spaces for the velocity and pressure, respectively.

2. Consider the restriction of these spaces to $\Omega_{h,i}$ defined by

$$\mathbf{V}_{h,i} \times Q_{h,i} := \widehat{\mathbf{V}}_{h,i}|_{\Omega_{h,i}} \times \widehat{Q}_{h,i}|_{\Omega_{h,i}}, \tag{3}$$

3. The desired space for the finite element formulation is defined by

$$\mathbf{V}_h \times Q_h := \bigoplus_{i=0}^{1} \mathbf{V}_{h,i} \times Q_{h,i}. \tag{4}$$

Note that functions in $\mathbf{V}_h \times Q_h$ will technically have have two different parts on $\Omega_{h,0} \cap \Omega_1$, namley one part from each $\mathbf{V}_{h,i} \times Q_{h,i}$, for $i = 0, 1$. However, the convention when referring to, evaluating, or visualizing functions in $\mathbf{V}_h \times Q_h$ is to always pick the part from the highest mesh in the hierarchy, unless otherwise specified. Note that this convention allows functions in $\mathbf{V}_h \times Q_h$ to be discontinuous on Γ even though the functions in $\widehat{\mathbf{V}}_{h,i} \times \widehat{Q}_{h,i}$, for $i = 0, 1$, are not. In this application, Taylor-Hood elements are used to ensure the stability of the solution, i.e., continuous piecewise polynomials of degree two and degree one for the velocity and the pressure, respectively. The cut finite element formulation for Stokes equations on two overlapping meshes reads: Find $(\mathbf{u}_h, p_h) \in \mathbf{V}_h \times Q_h$ such that

$$
\begin{aligned}
& (D\mathbf{u}_h, D\mathbf{v})_{\Omega_0} + (D\mathbf{u}_h, D\mathbf{v})_{\Omega_1} \\
& - (\langle \partial_{\mathbf{n}} \mathbf{u}_h \rangle, [\mathbf{v}])_\Gamma - ([\mathbf{u}_h], \langle \partial_{\mathbf{n}} \mathbf{v} \rangle)_\Gamma \\
& + \beta h^{-1} ([\mathbf{u}_h], [\mathbf{v}])_\Gamma + ([D\mathbf{u}_h], [D\mathbf{v}])_{\Omega_{h,0} \cap \Omega_1} \\
& - (\nabla \cdot \mathbf{u}_h, q)_{\Omega_0} - (\nabla \cdot \mathbf{u}_h, q)_{\Omega_1} + ([\mathbf{n} \cdot \mathbf{u}_h], \langle q \rangle)_\Gamma \\
& - (\nabla \cdot \mathbf{v}, p_h)_{\Omega_0} - (\nabla \cdot \mathbf{v}, p_h)_{\Omega_1} + ([\mathbf{n} \cdot \mathbf{v}], \langle p_h \rangle)_\Gamma \\
& + h^2 (\Delta \mathbf{u}_h - \nabla p_h, \Delta \mathbf{v} + \nabla q)_{\Omega_{h,0} \setminus \omega_{h,0}} \\
= \quad & (\mathbf{f}, \mathbf{v})_{\Omega_0} + (\mathbf{f}, \mathbf{v})_{\Omega_1} - h^2 (\mathbf{f}, \Delta \mathbf{v} + \nabla q)_{\Omega_{h,0} \setminus \omega_{h,0}},
\end{aligned}
\tag{5}
$$

for all $(\mathbf{v}, q) \in \mathbf{V}_h \times Q_h$. Here, $(\cdot, \cdot)_\Omega$ is the $L^2(\Omega)$-inner product, $D\mathbf{v} = (\nabla \mathbf{v}_1, \ldots, \nabla \mathbf{v}_d)$, $\langle v \rangle = (v_0 + v_1)/2$ is the average of v on Γ (v_i is the limit of v on Ω_i when approaching Γ, for $i = 0, 1$), $\partial_{\mathbf{n}} \mathbf{v} = (\nabla \mathbf{v}_1 \cdot \mathbf{n}, \ldots, \nabla \mathbf{v}_d \cdot \mathbf{n})$, $\mathbf{n}$ is the unit normal to Γ exterior to Ω_1, $[v] = v_1 - v_0$ is the jump in v on Γ, β is a stability parameter, h is the mesh size, $\Omega_{h,0} \cap \Omega_1$ is the covered part of all the background cells that are cut by Γ, and $\Omega_{h,0} \setminus \omega_{h,0}$ is all the background cells that are cut by Γ. Recall from theory that all differential operators are to be understood in the generalized sense. For more details on the method, analysis, and numerical results, see [16].

For the application studied in this work, the background mesh is also referred to as the air mesh, since it is a discretization of a region containing air. The overlapping meshes are also referred to as house meshes, since they contain the houses.

The wind model is implemented in Python using the open-source finite element library FEniCS, see [26], whose multimesh-functionality provides easy-to-use tools for implementing cut finite element methods on overlapping meshes. The development of FEniCS-multimesh is ongoing work and currently it is not as robust in 3D as in 2D. Therefore, after some experimentation in 3D, we chose to retreat to a 2D setting for the wind model. Wind in a real world setting is obviously a 3D phenomenon, but as already mentioned, to have a physically accurate wind model is not the main goal here. At the moment, we have to settle

with 2D Stokes instead of 3D Navier-Stokes, due to how far the work with cut finite element methods on overlapping meshes has come. A good thing is that the difference between FEniCS-implementations in 2D and 3D usually is small, simplifying extensions to higher dimensions. However, one difference for this application is that in 3D, the surface of the island, where the houses are placed, will be part of the boundary of the solution domain. Thus one might have parts of house meshes protruding from the air mesh. This situation can be avoided in 2D by viewing the island and houses from above, which also seems natural for this application, and extending the solution domain beyond the boundary of the island. The background air mesh may then be made independently of the island. However, we have chosen to embed the boundary of the island in the air mesh to simplify visual validation of house locations and for aesthetic reasons. We are now ready to formulate the algorithm for obtaining the wind over the island and houses by solving (5).

Algorithm 1 Wind model

1: Geometries for the island and houses are imported. See Figure 5.
2: Meshes are generated using the geometries. See Figure 6.
3: The house meshes are placed inside the air mesh. See Figure 7.
4: The linear system of equations, resulting from (5), is assembled and boundary conditions are applied to it: inlet and outlet for the air mesh, and no-slip for the house boundary of the house meshes.
5: Background cells intersecting the hole in house meshes are located and marked as covered. Corresponding degrees of freedom are inactivated, preventing wind inside houses (That would be wrong!). See Figure 8.
6: The modified linear system of equations is solved.

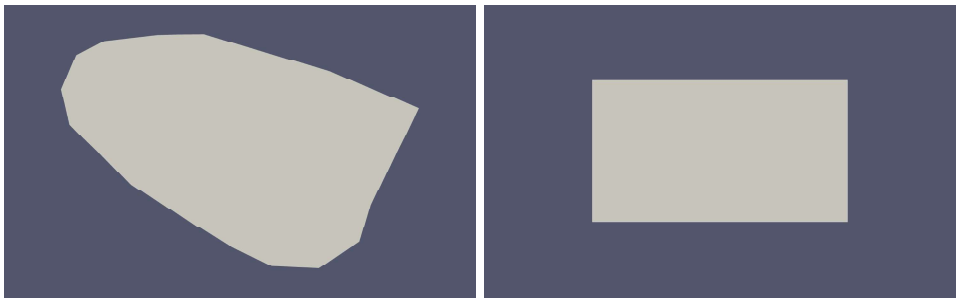

Figure 5: Geometries for island (*left*) and house (*right*) represented by polygons.

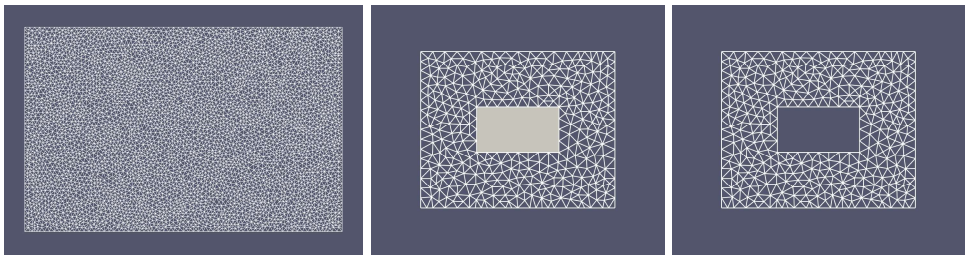

Figure 6: Meshes generated from geometries. *Left*: Air mesh with embedded boundary of island geometry. *Middle*: House mesh around house geometry. *Right*: House mesh without house geometry. Note the house-shaped hole in the mesh.

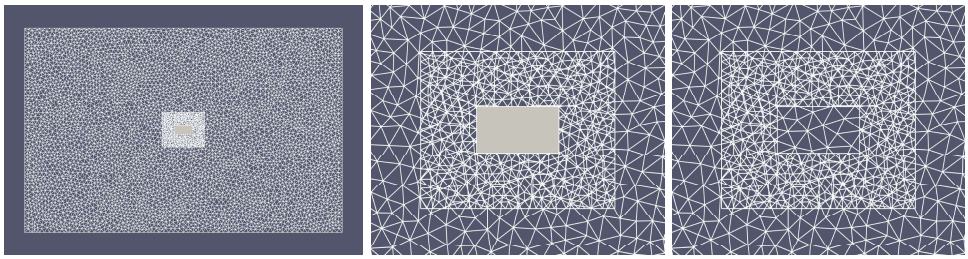

Figure 7: Placement of overlapping house mesh in background air mesh with and without house geometry. Note the presence of background mesh cells in the hole of the overlapping mesh.

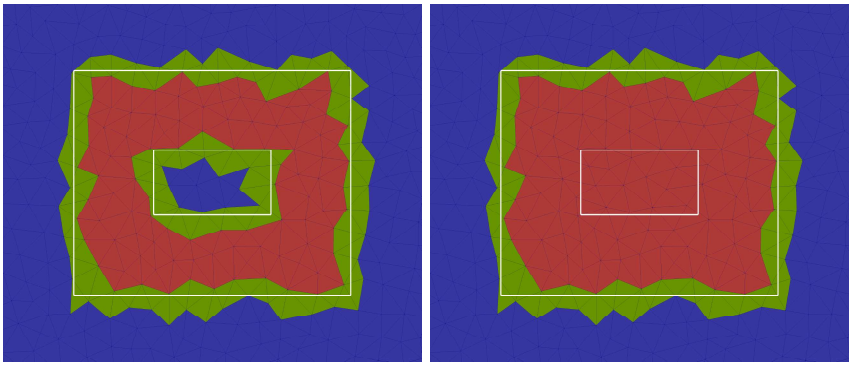

Figure 8: Classification of background mesh cells before (*left*) and after (*right*) locating and marking cells intersecting the hole of the overlapping mesh outlined in white. Uncovered cells are blue, cut cells are green, and covered cells are red.

2.2 Computation of view

Looking into literature, it is well known that the view from a real estate can affect the value of the property. However, the impact of view on real estate prices has not been researched as much as other factors. Examples from literature, that discuss the value of view are [27, 28, 29]. However, these papers classify the view based on subjective human opinion. In [28] they classify view by different types: ocean, lake or mountain view, where some of them are further classified into four subclasses from full ocean view to partial ocean view. They then estimate how big impact the different types of view have on the price. Instead of a purely subjective classification, we in this paper present a novel and quantitative measure, denoted V, for evaluating the view from a location such as the window of a building. The measure gives a value between zero and one, zero being the worst possible view and one the best possible view. We present a more elaborate measure for 3D settings and a quite basic measure for 2D settings, starting with the former.

2.2.1 View 3D

It is interesting to first consider the worst and the best cases. For this application, the worst case would occur if the view is nothing else than another house, see illustration in Figure 9. Thus in this case the view should evaluate to $V \approx 0$. The best case, $V = 1$, would be a view consisting entirely of sea and/or sky, see Figure 10.

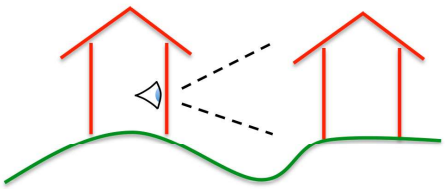

Figure 9: Worst case: $V \approx 0$.

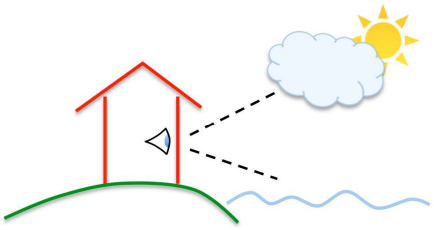

Figure 10: Best case: $V = 1$.

Examining again the worst case, the distance to a neighboring house should influence the value of the view. In general, the negative impact of objects on the view should decrease by the distance.

Our proposed measure of view is expressed as an integral over the integration domain $\omega = \omega_\phi \times \omega_\theta$ of size $|\omega|$, where ω represents the viewed surface:

$$V := \frac{1}{|\omega|} \iint_{\omega} \sigma(\phi, \theta) \, \mathrm{d}\phi \, \mathrm{d}\theta \ . \tag{6}$$

Here, the function σ is the view density. It depends on the angles ϕ and θ, and it should take a value between zero and one. In the present study, we have used the following definition:

$$\sigma(\phi, \theta) := \begin{cases} 1, & \text{if water or sky,} \\ 2S\left(w(\phi, \theta)\frac{l(\phi,\theta)}{L}\right) - 1, & \text{otherwise,} \end{cases} \tag{7}$$

where S is the Sigmoid function, $S(t) = (1 + e^{-t})^{-1}$; $w(\phi, \theta)$ is the specific weight for the object viewed at (ϕ, θ), and is set to be 0.1 for house, and 0.7 for ground; $l(\phi, \theta)$ is the distance to the nearest object viewed at (ϕ, θ); and L is a calibration parameter. Based on the premise that $\sigma(\phi, \theta) = 0.9$ for a house viewed at (ϕ, θ) and placed at the horizon approximately 5 kilometers away, the parameter L is found to be 0.217 km. As the size of objects decreases almost exponentially with respect to the distance from the camera, $S(t)$ has the largest impact on objects nearby. The reason for choosing $2S(t) - 1$ instead of, e.g., the simpler $1 - e^{-t}$, (both are 0 for $t = 0$, and both go to 1 when $t \to \infty$), is because of the slower growth of the former.

The measure defined in (6) computes the view independent of cardinal direction. However, in northern countries like Sweden, a southward view is often weighted higher than a northward view because of the sunlight. Thus, to obtain a view formula which can depend on the cardinal direction, we let $\theta \in [0, 2\pi]$ be the angle in the horizontal plane. Let southward be $\theta = 0$, consequently $\theta = \pi$ is northward. Assuming that one would weight the southward view three times higher as the northward view, the cardinal direction weight function could be expressed by

$$D(\theta) = 1 + \frac{1}{2}\sin(\theta - 3\pi/2) \ . \tag{8}$$

For a 360° horizontal view valuation, we introduce $D(\theta)$ in (6). We thus define the following modified measure of view that takes values in $[0, 1]$:

$$V_{360} := \frac{1}{|\omega|} \int_0^{2\pi} D(\theta) \int_{\omega_\phi} \sigma(\phi, \theta) \, \mathrm{d}\phi \, \mathrm{d}\theta \ . \tag{9}$$

Given the mesh(es) constructed for a 3D wind simulation, it is possible to visualize the view from a given point. In practice we do this by rendering an image from the 3D mesh with the use of the rasterization rendering technique. This technique goes back to [30, 31, 32]. The rasterization algorithm projects the triangles from the 3D mesh onto a 2D image.

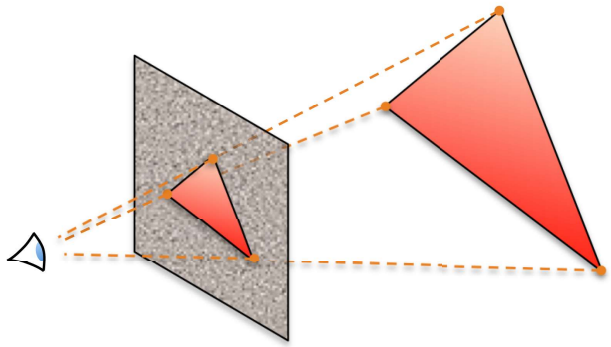

Figure 11: Projection of triangles from the 3D mesh onto the 2D image.

The projecting principle is sketched in Figure 11, where the vertices that are mapped onto the 2D image plane are used to check which pixels the triangle intersects. The naive idea is to loop through all the pixels in the image and check if they are inside the projected triangle or not. The efficiency of this approach depends on the size of the triangles. To account for small triangles, one may optimize the search by only checking pixels that lie inside the bounding box of the triangle. In Figure 12, the blue rectangle around the triangle illustrates the bounding box for which the corner coordinates are rounded off in order to include whole pixels. This can be optimized further by not checking all the pixels inside the bounding box, but starting in the pixel which contains the top-point of the triangle. Then go stepwise down and check the pixels to the left and right of the reference point. If we already visited one or more pixels which are inside the projected triangle and then comes to a pixel which is not in the triangle, the search stops in that direction. The search algorithm is illustrated with the green lines in Figure 12. There are several ways to optimize this, see [31] for further reading. Looping over all the triangles to do the projections one by one can cause two or more projected triangles to overlap. To decide which one that should be shown in the image, we have to look at the distances to them. The distances to the triangles that are already shown in the image are stored in a two dimensional array with the same dimension as the image. Thus only the triangle with the shortest distance may be shown in the image.

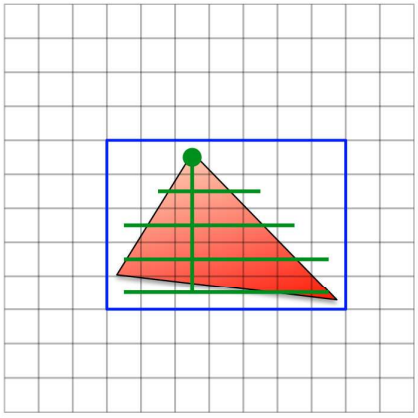

Figure 12: To check which pixels lie inside the triangle, we start from the top-point of the triangle (the green dot) going down one step at a time to check if the pixels to the left and right are contained in the triangle.

The rasterization algorithm is implemented in C++ and by the use of the SWIG interface compiler it is possible to access the rasterization algorithm from a Python script. The rasterization algorithm needs as input a list of FEniCS (.xml) meshes, where the first mesh in the list should be the background air mesh, and the rest overlapping house meshes. With the scene set, the algorithm needs to know the size of the image, both the size in pixels and the real size measured in the same units as the meshes. Also the position and direction of the camera, and the distance between the camera and the image are needed. With these inputs, the algorithm generates an image and a matrix $\mathbf{S}$ of the same size as the image that contains a value for σ for each pixel. The view V may thus be computed from $\mathbf{S}$. If we want to compute the 360° view from (9), the domain ω is assumed to be either a cylinder or a sphere. As the rasterization algorithm generates flattened images, (9) cannot be used directly. However, (9) can be approximated by

$$V_{360} \approx \sum_{n=0}^{N-1} \frac{D\left(2\pi n/N\right)}{N} V_n \ , \tag{10}$$

where $N \geq 3$ is the number of images, and V_n is the view valuation for image number n, i.e., V_n is evaluated in the direction with angle $2\pi n/N$ (recall 0 being south), and an image width of $2d\tan(\pi/N)$, where d is the distance between the camera and the image. It is important that no images are overlapping and that they, when joined together and projected onto the horizontal plane, form the boundary of a convex regular polygon with N edges.

2.2.2 View 2D

We now present a simple view measure for 2D settings. Start by considering the projection of a settlement layout onto the horizontal plane. It does no longer make sense to consider the influence of ground, since the houses are immersed in it. Instead we let the view be either clear, taking $\sigma = 1$, or blocked by another house, taking $\sigma = 0$. The analogous 2D measure of (6), for some view point, is defined by

$$V^{2D} := \frac{1}{|\omega_\theta|} \int_{\omega_\theta} \sigma(\theta) \, \mathrm{d}\theta. \tag{11}$$

Due to the binary view density σ, (11) consists of clear and blocked circle sectors, see Figure 13. The angle of a blocked circle sector is inversely proportional to the distance between the view point and the blocking house(s).

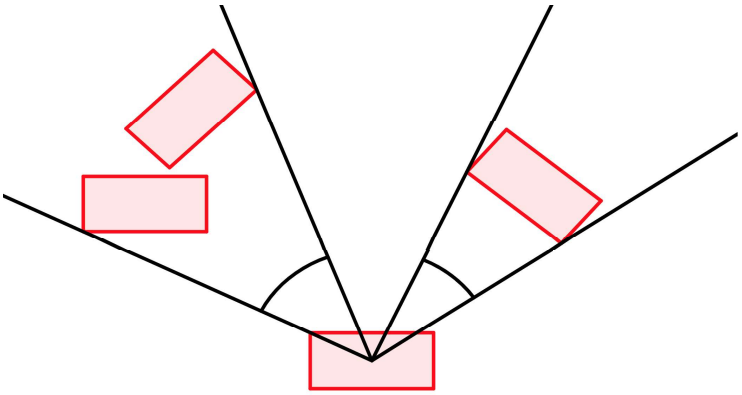

Figure 13: View point taken as the midpoint of a house, resulting in two blocked circle sectors. Note that two or more houses may give rise to a single blocked circle sector.

Letting $\omega_\theta = [0, 2\pi)$, i be a house index, and taking the view point to be the midpoint of house i, we may define the 360° view from house i by

$$V^{2D}_{360,i} := 1 - \frac{1}{2\pi} \sum_{j \in B_i} b_{ij}, \tag{12}$$

where B_i is the index set of blocked circle sectors for house i, and b_{ij} is the angle in radians of blocked circle sector j for house i. Note that (12) is equivalent to a simplified isovist.

2.3 Multi-objective optimization

The models for wind and view can be used to quantitatively evaluate a settlement layout. We may thus formulate a multi-objective optimization problem for finding the optimal placement of buildings in a landscape, such as houses on an island, with respect to the wind conditions around the buildings and the view from the buildings. Let $d = 2, 3$, be the spatial dimension of the settlement layout, and H be an index set for the houses. The variables for the problem are the position and orientation of the houses:

$$\mathbf{x} = (\dots, x_{i1}, \dots, x_{id}, \theta_i, \dots), \quad i \in H, \tag{13}$$

where $x_{i1}, \dots, x_{id}$ are the Cartesian coordinates for the midpoint of house i, and θ_i is the angle of rotation in the horizontal plane for house i around its midpoint. To define an objective function, measures for both wind and view are needed. From construction, the output from the wind model is a function (velocity and pressure as a function of space) and not a number that says how good the wind is, as opposed to the view model. The natural mathematical way of measuring functions is to use norms. Thus, a measure of wind in a region Ω could be the $L^\infty(\Omega)$-norm of the velocity. In practice it can be more convenient to use an L^p-norm with a large p instead of the L^∞-norm, since this makes the objective function smoother. After trial and error testing in 2D settings, $p = 100$ was chosen for this study. Since wind can cause noise inside a house and damage the outside, we assume that it is desirable to have as weak wind as possible around all the houses. Hence we choose to minimize the wind around the house with the strongest surrounding wind. The collective measure of wind for a settlement layout is thus defined by

$$f_W := \max_{i \in H} \{ \|\mathbf{u}_h\|_{L^{100}(\Omega_i)} \}. \tag{14}$$

where $\mathbf{u}_h$ is the velocity field from (5), and Ω_i is the region of the overlapping house mesh for house i. The reason for choosing this region is simply that it is an already existing and easy accessible data structure for a surrounding region of a house, but one could of course work with other regions as well. For the view, we assume that it is desirable to have as good view as possible from all the houses. Hence, and for consistency with (14), we choose to minimize the complement of the 360° view from the house with the worst view. The collective measure of view for a settlement layout is thus defined by

$$f_V := \max_{i \in H} \{ 1 - V_{360,i} \}, \tag{15}$$

where $V_{360,i}$ is the 360° view for house i, defined by (9) for 3D and (12) for 2D. One may now use the two measures (14) and (15) to construct an objec-

tive function. In this study, we have taken it to be a convex combination of the two measures. This is more generally known as linear scalarization with positive weights and minimizing such an objective function is a sufficient but not necessary condition for Pareto optimality of the solution according to, e.g., [33]. The problem constraints are of two types: "stay on island"-constraints, meaning that the houses must be placed on the island; and "keep distance to neighbors"-constraints meaning that the houses may not be too close to each other. We denote the former by s_i, for $i \in H$, and the latter by k_{ij}, for $i, j \in H$ with $i < j$. Note that this is because the distance between houses i and j is the same as between houses j and i, and that a house cannot keep a distance to itself. The multi-objective constrained problem for optimizing a settlement layout is thus

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \alpha f_W(\mathbf{x}) + (1-\alpha) f_V(\mathbf{x}) \\ \text{subject to} \quad & s_i(\mathbf{x}) \geq 0,\ i \in H, \\ & k_{ij}(\mathbf{x}) \geq 0,\ i, j \in H, i < j, \end{aligned} \tag{16}$$

where the coefficient $\alpha \in [0, 1]$ is called the wind weight. Taking $\alpha = 0$ gives a pure view optimization problem, and $\alpha = 1$ pure wind.

The implementation is done with the Python package scipy.optimize, using the modified Powell algorithm as optimization method. By testing the available methods in scipy.optimize for 2D settings, this was simply the method that worked best. The Powell method is also gradient-free, which seems to be suitable considering the black box nature of the objective function in (16). However, constraints cannot be supplied to the Powell method in scipy.optimize, so they have to be added to the objective function, resulting in an unconstrained optimization problem. The reformulation of (16) as an unconstrained problem was taken to be

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \alpha f_W(\mathbf{x}) + (1-\alpha) f_V(\mathbf{x}) \\ & + p_s \sum_{i \in H} \min(0, s_i(\mathbf{x}))^2 + p_k \sum_{\substack{i,j \in H \\ i<j}} \min(0, k_{ij}(\mathbf{x}))^2, \end{aligned} \tag{17}$$

where p_s and p_k are penalty parameters.

3 Results

We present simulation results from the wind model in 2D, Section 3.1; the 3D view model, Section 3.2; the 2D view model, Section 3.3; and optimization of 2D settlement layouts, Section 3.4. Sections 3.1 and 3.3 are kept short, since Section 3.4 naturally contains results from the wind model in 2D and the 2D view model.

3.1 Wind 2D

In Figure 14, the velocity, $\mathbf{u}_h$ from (5), for a 2D problem with one overlapping house mesh is shown. The velocity is visualized with glyphs, colored and scaled by magnitude. The house is shown in red

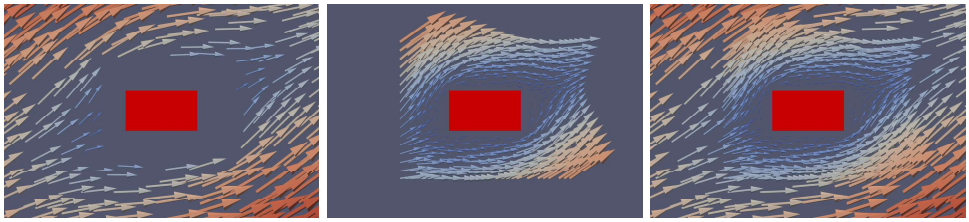

Figure 14: *Left*: Part of $\mathbf{u}_h$ from background air mesh. *Middle*: Part of $\mathbf{u}_h$ from overlapping house mesh. *Right*: Complete $\mathbf{u}_h$, i.e., from both meshes.

3.2 View 3D

To test the 3D view model, houses are arbitrarily placed on the island as seen in Figure 15. The computed view from the yellow dot in Figure 15 for four different directions is shown in Figure 16–19.

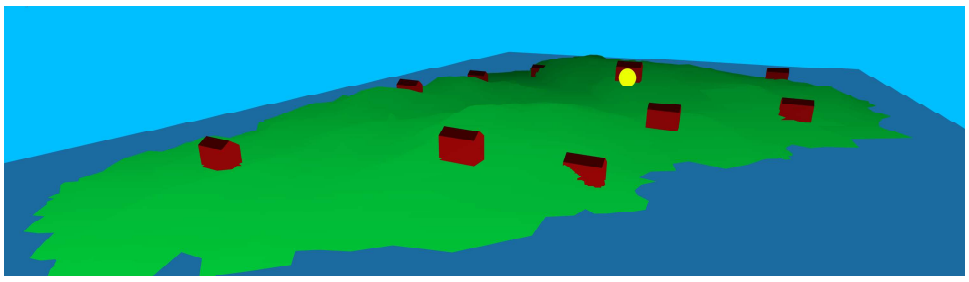

Figure 15: Overview of the island with ten houses. The yellow dot represents the position of the camera for the view computation.

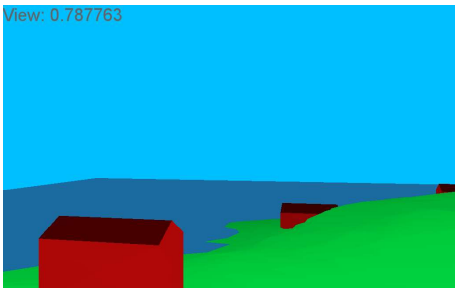

Figure 16: Fair view: $V = 0.79$

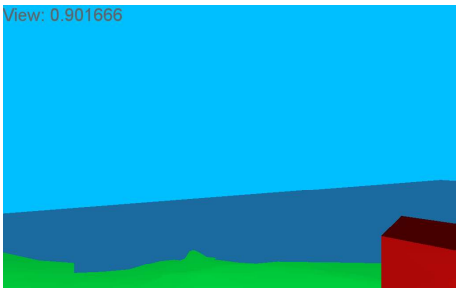

Figure 17: Good view: $V = 0.90$

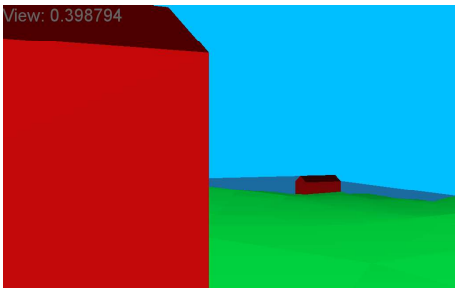

Figure 18: Poor view: $V = 0.40$

Figure 19: Bad view: $V = 0.02$

The 360° horizontal view measure V_{360}, defined by (10), depends on the cardinal direction. For 32 computed rasterizations from the yellow dot in Figure 15, $V_{360} = 0.68$, if south is in the direction of the neighboring house; and $V_{360} = 0.73$, if south is in the opposite direction.

3.3 View 2D

In Figure 20, three different 2D settlement layouts with seven houses and different values for the collective measure of view f_V, defined by (15), are shown.

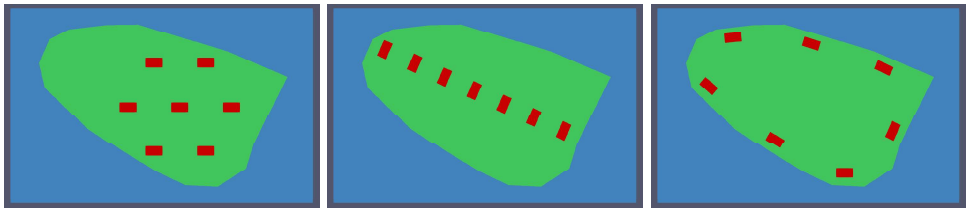

Figure 20: *Left*: Hexagonal layout with $f_V = 0.3064$. *Middle*: Principal line layout with $f_V = 0.1880$. *Right*: Coastal layout with $f_V = 0.1674$.

3.4 Optimization 2D

We present optimization results from solving problem (17) for 2D settlement layouts with seven houses, using the Powell method in scipy.optimize. The optimiziations are of three types: Pure wind ($\alpha = 1$), pure view ($\alpha = 0$), and mixed ($0 < \alpha < 1$). For all three types, three different initial settlement layouts are used: hexagonal, principal line, and coastal. They are shown in Figure 20. The penalty parameters, chosen by trial and error, are $p_s = 10^8$ and $p_k = 10^3$. For the wind simulations, seven identical copies of a house mesh are used for the houses. Together with the air mesh they constitute the multimesh hierarchy. See Table 1 for mesh data.

Mesh data	Air mesh	House mesh	Multimesh
Number of vertices	5242	390	7972
Number of edges	15453	1072	22957
Number of cells	10212	682	14986

Table 1: Mesh data for background air mesh, house mesh, and multimesh hierarchy (background air mesh + 7 overlapping house meshes).

The system size is 69830, since Taylor-Hood elements are used. On a MacBook Pro with a 3.1 GHz Intel Core i7 processor, the average time for an evaluation of f_W is 5.3s, and for f_V 0.0086s (averaged over 100 simulations). The evaluation of f_W thus costs approximately 600 times more than that of f_V.

For pure wind optimization, three different inflow wind directions are used: horizontal (from left to right), vertical (from bottom to top), and diagonal (from lower left corner to upper right corner). The inflow wind speed is 1. Numerical results from pure wind optimization in 2D are shown in Table 2.

Wind \ Layout	Hexagonal	Principal line	Coastal
Horizontal	1.1266 (1.3490) 3363	1.0631 (1.5981) 8307	1.0370 (1.3934) 12574
Vertical	1.1191 (1.2135) 1723	0.8633 (2.0162) 10119	1.1101 (1.3331) 6062
Diagonal	0.9923 (1.5870) 13800	1.2298 (2.1427) 3185	1.2423 (1.3917) 2982

Table 2: Optimized values of the collective measure of wind f_W from *pure wind* optimization. Starting values of f_W are shown in parentheses. Number of evaluations of the objective function are shown below.

Layouts from pure wind optimization in 2D are shown in Figure 21 – 23. The velocity $\mathbf{u}_h$ from (5) is visualized with glyphs, colored and scaled by magnitude. From left to right, the initial layout was hexagonal, principal line, and coastal, respectively.

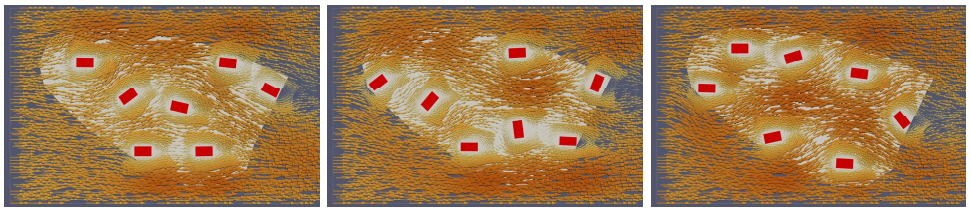

Figure 21: Layouts from pure wind optimization with *horizontal* wind.

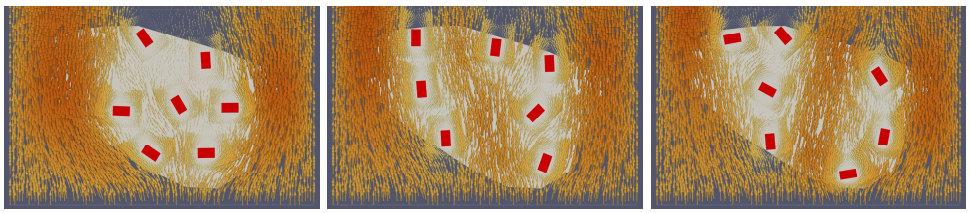

Figure 22: Layouts from pure wind optimization with *vertical* wind.

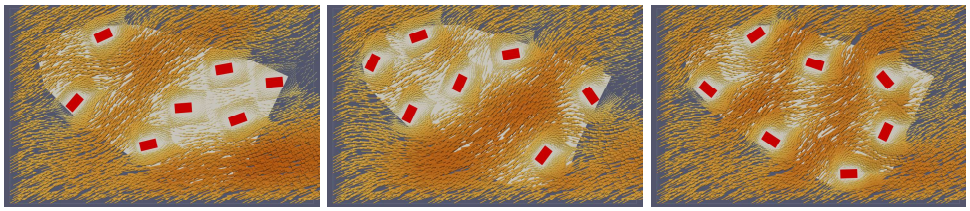

Figure 23: Layouts from pure wind optimization with *diagonal* wind.

Simulation results from pure view optimization in 2D are shown in Table 3 and Figure 24.

Hexagonal	Principal line	Coastal
0.1513 (0.3064)	0.1619 (0.1880)	0.1486 (0.1674)
5142	6582	5040

Table 3: Optimized values of the collective measure of view f_V from *pure view* optimization for three initial layouts. Starting values of f_V are shown in parentheses. Number of evaluations of the objective function are shown below.

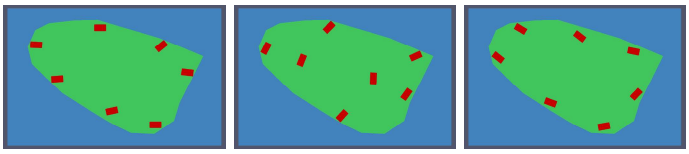

Figure 24: Layouts from pure view optimization. From left to right, the initial layout was hexagonal, principal line, and coastal, respectively

For mixed optimization, only the diagonal inflow wind direction and five values for α are used. Numerical results from mixed optimization in 2D are shown in Table 4 – 6.

Layout \ α	0.1	0.3	0.5	0.7	0.9
Hexagonal (1.5870)	1.3207	1.1926	1.2735	1.1916	1.1837
Principal line (2.1427)	1.2634	1.2065	1.1642	1.1585	1.1559
Coastal (1.3917)	1.2553	1.2240	1.2285	1.2529	1.2702

Table 4: Optimized values of the collective measure of wind f_W from *mixed* optimization. Starting values of f_W are shown in parentheses.

Layout \ α	0.1	0.3	0.5	0.7	0.9
Hexagonal (0.3064)	0.1647	0.1842	0.2011	0.1810	0.2340
Principal line (0.1880)	0.1584	0.1681	0.1757	0.1898	0.1969
Coastal (0.1674)	0.1529	0.1655	0.1729	0.1726	0.1926

Table 5: Optimized values of the collective measure of view f_V from *mixed* optimization. Starting values of f_V are shown in parentheses.

Layout \ α	0.1	0.3	0.5	0.7	0.9
Hexagonal	2984	5355	3001	5501	3552
Principal line	3855	5282	3861	6317	5248
Coastal	2480	4177	3718	3143	3803

Table 6: Number of evaluations of the objective function for mixed optimization.

Layouts from mixed optimization in 2D are shown in Figure 25 – 27. From left to right, the wind weight $\alpha = 0.1, 0.3, 0.5, 0.7, 0.9$, respectively

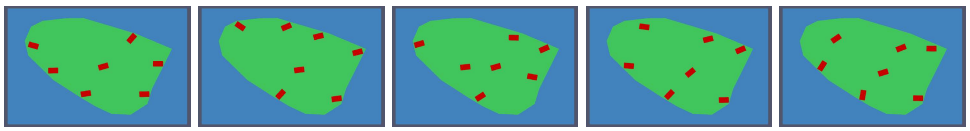

Figure 25: Layouts from mixed optimization with *hexagonal* initial layout.

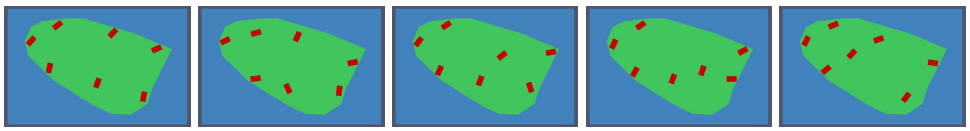

Figure 26: Layouts from mixed optimization with *principal line* initial layout.

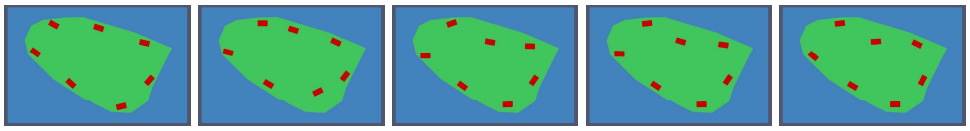

Figure 27: Layouts from mixed optimization with *coastal* initial layout.

4 Discussion

Here we comment on methods and results, starting with the wind model. Figure 14 and Figure 21 – 23 show the smooth transition of $\mathbf{u}_h$ between meshes. This is good, since overlapping meshes should be a behind the scenes technique that does not affect the appearance of the solution. Especially, we do not want the existing discontinuity on the boundary between the meshes to show up as a seam in the solution. The glyphs used to visualize $\mathbf{u}_h$ in Figure 14 and Figures 21 – 23 obviously reveal the existence and extent of overlapping house meshes. But this is because the overlapping house meshes have a finer mesh size than the background air mesh, see Figure 7, and that the glyphs are placed uniformly over the nodal points in the meshes, resulting in a higher glyph density on the house meshes. This could be nice, since it makes it easier to see the boundaries between meshes and verify that the transition of the solution looks smooth enough.

Concerning the 3D view model, Figure 16–19 show some views from the selected location. Figure 16 shows a fair view with a lot of sea and sky but also three houses and a part of the island. Therefore the view valuation becomes $V = 0.79$. To get a better view there should be as much sea or sky as possible, fewer houses, and less ground. This can be seen in Figure 17, where $V = 0.90$. Examples of views from the other end of the view scale can be found in Figure 18 and 19. In Figure 19 one almost only sees a neighboring house, resulting in a very low view value. In [29] Table 1, previous studies on the impact of view on real estate prices are listed. Every study classifies the view differently. Summarizing these studies, view from a waterfront increases the value of a house by $127\% - 147\%$, full water view by $32\% - 68\%$, and simple/partial water view by $6\% - 10\%$. It is hard to directly compare these numbers to our view computations. But comparing the full waterfront view in Figure 17 with the partial sea view in Figure 18, the waterfront view ($V = 0.90$) is 125% higher than the partial sea view ($V = 0.40$).

For the 2D view model, Figure 20, shows that f_V is larger when the houses are lumped together (hexagonal with $f_V = 0.3064$), than when spread out (coastal $f_V = 0.1674$). This is what to be expected from the construction of the view measure, since seeing more house should result in poorer view, and seeing less house in a better. When the houses are placed closely, but on a line, f_V is also relatively low (principal line with $f_V = 0.1880$). This is because the view from a house consists of at most two other houses, behind which the rest of the houses are hidden.

The optimization results show that there are a lot of local minima to our optimization problem. For pure wind optimization, Table 2 shows that f_W decreases for all nine cases. It also shows that the starting values of f_W are

significantly higher ($f_W > 2$) for principal line layout with vertical and diagonal wind. This is probably because a lot of wind is forced in between the closely placed houses, resulting in high surrounding wind speeds, which also is to be expected. From Table 2, the optimized values of f_W seem to get smaller when more evaluations of the objective function are used in the optimization. The optimizations that resulted in the three smallest values all used more than 10000 evaluations. Figure $21 - 23$ show that it can be beneficial for the houses to align themselves with the wind and to lie in the wake of other houses, similar to how birds fly in V formation. This is especially evident in the middle layout of Figure 22 ($f_W = 0.8633$), and the left layout in Figure 21 ($f_W = 0.9923$).

For pure view optimization, Table 3 shows that f_V decreases to similar values for all three cases. The starting value of f_V for hexagonal initial layout is significantly larger than its optimized value. This is in contrast to the other initial layouts, for which starting and optmized values are similar. This suggest that principal line and coastal already are good layouts with respect to view. Comparing Figure 24 with Figure 20, this seems to be true since the left case goes from hexagonal to a coastal layout, and the right case shows that the coastal layout barely changes. The middle case seems to be a compromize between principal line and coastal layout, with some houses at the coast and others along the principal line of the island.

For mixed optimization, we would expect f_W to decrease when α increases. Table 4 shows that for hexagonal, the values are generally decreasing; for principal line, the values are strictly decreasing, and for coastal the values remain quite constant (actually increasing somewhat for $\alpha > 0.3$). In Figure 27 the layouts do not change much, suggesting that the optimization process gets stuck in a smaller valley of the optimization landscape when using coastal layout, leading to the quite constant values of f_W. Considering f_V for mixed optimization, we would expect it to increase, when α increases. Table 5 shows that for hexagonal and coastal, the values are generally increasing (almost strictly for coastal), and for principal line the values are strictly increasing. For the layouts from mixed optimization, we would expect them to have characteristics of pure view layouts, i.e., more spread out, for small α. As α increases, we would expect to see less of these and more characteristics of pure wind layouts, i.e., alignment with wind and lying in wakes. In Figure 25, characteristics of pure wind layouts seem to be present in all cases, but to be more apparent for larger α. In Figure 26, the behavior of the layouts seems to be as expected, i.e., going from a spread out coastal layout to one where houses lie in wakes of other houses and are aligned with the diagonal wind, when α increases. As already mentioned, the behavior in Figure 27, suggests the entrapment in a local minimum valley.

5 Conclusions

We have presented a generic framework for evaluating and optimizing settlement layouts with respect to wind and view. The framework allows multiple configurations to be computed and evaluated with relative ease. The current proof-of-concept implementation has several limitations that will be addressed in future work. These limitations and extensions fall into three different categories: modelling, multimesh software, and ease of use.

Concerning modelling, the use of Stokes equations in 2D to model air flow between buildings has two important limitations: Stokes flow is not physically appropriate to model air flow, and real world buildings are 3D. Future work would therefore extend the wind model to more advanced flow equations in 3D. For the computation of view, a more extensive comparison with real world data, in order to calibrate functions and weights, could be a topic for further research. Maybe in a similar fashion to [22]. Although the current basic optimization seems to work fairly well, there is a lot of potential for future studies and improvements. We mainly think of the definition of the objective function, and the choice of algorithm for solving the optimization problem.

Regarding multimesh software, we noticed during this study that some particular configurations of buildings in 3D resulted in numerical instabilities. This is likely due to bugs or untreated corner cases in the computational geometric framework of the FEniCS multimesh implementation. This issue is also the focus of ongoing work.

To be a useful tool in an iterative architectural process, the framework must not only be efficient and robust. It must also be easy to use. Future work will also consider the creation of a user-friendly interface. In particular, it would be highly relevant to consider the creation of virtual or mixed reality interfaces to our framework.

Acknowledgements

This work was supported by the profile area *Virtual Cities* as part of *Building Futures*, an Area of Advance at Chalmers. We are also grateful to CREAM Architects in Gothenburg for ideas, feedback and inspiration during this project; and Professor Michael Patriksson, Chalmers University of Technology and University of Gothenburg, for consultation and guidance regarding optimization.

References

[1] M. Batty, "Exploring isovist fields: Space and shape in architectural and urban morphology," *Environment and Planning B: Planning and Design*, vol. 28, no. 1, pp. 123–150, 2001.

[2] B. Blocken, W. D. Janssen, and T. V. Hooff, "Environmental Modelling & Software CFD simulation for pedestrian wind comfort and wind safety in urban areas : General decision framework and case study for the Eindhoven University campus," *Environmental Modelling and Software*, vol. 30, pp. 15–34, 2012.

[3] J. Hang, Y. Li, M. Sandberg, R. Buccolieri, and S. Di Sabatino, "The influence of building height variability on pollutant dispersion and pedestrian ventilation in idealized high-rise urban areas," *Building and Environment*, vol. 56, pp. 346–360, 2012.

[4] X. Shi, Y. Zhu, J. Duan, R. Shao, and J. Wang, "Assessment of pedestrian wind environment in urban planning design," *Landscape and Urban Planning*, vol. 140, pp. 17–28, 2015.

[5] V. Jaillot, F. Pedrinis, S. Servigne, and G. Gesquière, "A generic approach for sunlight and shadow impact computation on large city models," in *25th International Conference on Computer Graphics, Visualization and Computer Vision 2017*, 2017, pp. 10—-pages.

[6] G. Johansson, F. Karlén, and M. Stark, "Lilla Fjellsholmen, varsamt byggande i en unik skärgårdsmiljö," Master's thesis, Chalmers University of Technology, Gothenburg, Sweden, 2014.

[7] A. Baskaran and A. Kashef, "Investigation of air flow around buildings using computational fluid dynamics techniques," *Engineering Structures*, vol. 18, no. 11, pp. 861–875, 1996.

[8] B. Blocken, J. Carmeliet, and T. Stathopoulos, "CFD evaluation of wind speed conditions in passages between parallel buildings—effect of wall-function roughness modifications for the atmospheric boundary layer flow," *Journal of Wind Engineering and Industrial Aerodynamics*, vol. 95, no. 9, pp. 941–962, 2007.

[9] B. Blocken, T. Stathopoulos, J. Carmeliet, and J. L. M. Hensen, "Application of computational fluid dynamics in building performance simulation for the outdoor environment: an overview," *Journal of Building Performance Simulation*, vol. 4, no. 2, pp. 157–184, 2011.

[10] V. Heuveline, S. Ritterbusch, and S. Ronnas, "Augmented Reality for Urban Simulation Visualization," *Preprint Series of the Engineering Mathematics and Computing Lab*, vol. 0, no. 16, 2011.

[11] B. Blocken, "Journal of Wind Engineering 50 years of Computational Wind Engineering : Past, present and future," *Jnl. of Wind Engineering and Industrial Aerodynamics*, vol. 129, pp. 69–102, 2014.

[12] S. Ingelsten, A. Mark, F. Edelvik, A. Logg, and M. Österbring, "Urban CFD-Simulation Using Point Cloud Data," in *Proceedings of NSCM-26: the 26th Nordic Seminar on Computational Mechanics*, 2016.

[13] A. Hansbo, P. Hansbo, and M. G. Larson, "A finite element method on composite grids based on nitsche's method," *ESAIM: Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique*, vol. 37, no. 3, pp. 495–514, 2003.

[14] A. Massing, M. G. Larson, A. Logg, and M. E. Rognes, "A Stabilized Nitsche Overlapping Mesh Method for the Stokes Problem," *Numerische Mathematik*, pp. 1–29, 2014.

[15] A. Johansson, B. Kehlet, M. G. Larson, and A. Logg, "MultiMesh Finite Element Methods I : Solving PDEs on Multiple Intersecting," *Computer Methods in Applied Mechanics and Engineering*, no. September, 2018.

[16] A. Johansson, M. G. Larson, and A. Logg, "High order cut finite element methods for the Stokes problem," *Advanced Modeling and Simulation in Engineering Sciences*, vol. 2, no. 1, pp. 1–23, 2015.

[17] J. S. Dokken, S. W. Funke, A. Johansson, and S. Schmidt, "Shape optimization using the finite element method," no. 251237, pp. 1–24, 2018.

[18] M. Benedikt, "To take hold of space: isovists and isovists fields," *Environment and Planning B: Planning and Design*, vol. 6, no. February, pp. 47–65, 1979.

[19] P. P.-j. Yang, S. Y. Putra, and W. Li, "Viewsphere : A GIS-based 3D visibility analysis for urban design evaluation," no. November, 2007.

[20] D. Nutsford, F. Reitsma, A. L. Pearson, and S. Kingham, "Personalising the viewshed : Visibility analysis from the human perspective," *Applied Geography*, vol. 62, pp. 1–7, 2015.

[21] D. Poerwoningsih, Antariksa, A. S. Leksono, and A. W. Hasyim, "Integrating Visibility Analysis in Rural Spatial Planning," *Procedia - Social and Behavioral Sciences*, vol. 227, pp. 838–844, jul 2016.

[22] D. Fisher-Gewirtzman, "Integrating ' weighted views ' to quantitative 3D visibility analysis as a predictive tool for perception of space," *Environment and Planning B: Urban Analytics and City Science*, 2018.

[23] M. Bruno, K. Henderson, and H. M. Kim, "Multi-Objective Optimization in Urban Design," pp. 102–109, 2010.

[24] I. Keough and D. Benjamin, "Multi-Objective Optimization in Architectural Design," pp. 1–8, 2010.

[25] S. Cajiot, N. Schüler, M. Peter, A. Koch, and F. Maréchal, "Interactive optimization for the planning of urban systems," 2017.

[26] A. Logg and G. N. Wells, "DOLFIN: Automated Finite Element Computing," *ACM Transactions on Mathematical Software*, vol. 37, no. 2, 2010.

[27] M. R. M. Hussain, I. Tukiman, I. H. Zen, and F. M. Shahli, "The Impact of Landscape Design on House Prices and Values in Residential Development in Urban Areas," *APCBEE Procedia*, vol. 10, pp. 316–320, 2014.

[28] E. D. Benson, J. L. Hansen, A. L. Schwartz, and G. T. Smersh, "Pricing Residential Amenities: The Value of a View," *The Journal of Real Estate Finance and Economics*, vol. 16, no. 1, pp. 55–73, 1998.

[29] S. C. Bourassa, M. Hoesli, and J. Sun, "What's in a View?" *Environment and Planning A*, vol. 36, no. 8, pp. 1427–1450, 2004.

[30] E. E. Catmull, "A Subdivision Algorithm for Computer Display of Curved Surfaces." Ph.D. dissertation, The University of Utah, 1974.

[31] J. Pineda, "A Parallel Algorithm for Polygon Rasterization," in *In Proceedings of Siggraph '88*, 1988, pp. 17–20.

[32] P. S. Heckbert, "Fundamentals of Texture Mapping and Image Warping," University of California at Berkeley, Berkeley, CA, USA, Tech. Rep., 1989.

[33] R. T. Marler and J. S. Arora, "Survey of multi-objective optimization methods for engineering," pp. 91–95, 2007.

Paper II

Solving Poisson's equation on the Microsoft HoloLens

Anders Logg, Carl Lundholm, Magne Nordaas

Abstract

We present a mixed reality application (HoloFEM) for the Microsoft HoloLens. The application lets a user define and solve a physical problem governed by Poisson's equation with the surrounding real world geometry as input data. Holograms are used to visualise both the problem and the solution. The finite element method is used to solve Poisson's equation. Solving and visualising partial differential equations in mixed reality could have potential usage in areas such as building planning and safety engineering.

Keywords: HoloLens, Poisson's equation, finite element method, FEniCS

1 Introduction

We develop an application, called HoloFEM, for solving Poisson's equation with the finite element method (FEM) using Microsoft's mixed reality glasses HoloLens [1]. The aim is to set up and solve a partial differential equation (PDE) in the real world geometry surrounding the HoloLens user, and then visualise the computed solution on top of the real surroundings in mixed reality.

Partial differential equations are used to model many physical processes, such as fluid flow, heat transport and electromagnetic fields to name a few. We consider here the Poisson equation, which serves a prototypical example of a PDE. This equation models steady-state diffusion and electrostatic potential. The Poisson problem is mathematically formulated as: Find the solution $u : \Omega \to \mathbb{R}$ such that

$$\begin{cases} -\Delta u = f & \text{in } \Omega, \\ u = g & \text{on } \partial\Omega, \end{cases} \tag{1}$$

where $\Omega \subset \mathbb{R}^3$ is the solution domain, Δ is the Laplace operator, f is a given source, g is a given boundary value of the solution, and $\partial\Omega$ is the boundary of Ω.

Since various PDEs often show up in modelling and engineering problems, there could be a potential use for mixed reality PDE-solving software that allows a user to define a problem, compute a solution, and study it on the spot. Say for example that we would like to know how a dangerous substance spreads in a room after a leak has sprung. The heat equation could be used as a simplistic model for describing this situation, potentially making applications like HoloFEM useful in building planning and safety engineering.

Augmented, mixed, and virtual reality are already used in engineering, architecture, and design, see for example [2, 3], but to our knowledge there is currently no other mixed reality PDE-solving software.

2 Technical description

The workflow of the application HoloFEM has three main stages. The first one is the meshing stage, where a computational mesh is generated from the environment. This is followed by the simulation stage, where the mathematical problem is formulated and solved. Finally, the solution is visualised in the last stage.

2.1 Meshing

The Microsoft HoloLens can scan the user's surroundings and extract a discrete representation of the geometry, in the form of a surface mesh. This mesh is not adequate for numerical computations – firstly, it is a surface mesh, while we need a volume mesh to solve (1), and secondly, the mesh quality is rather poor.

Instead, the main planes are extracted from the surface mesh. These will represent the walls, the floor and the ceiling of the room in which the user is located. From this geometric representation, the computational volume mesh is constructed. The procedure so far is summarised in Figure 1. The steps in the mesh generation are shown in Figure 2.

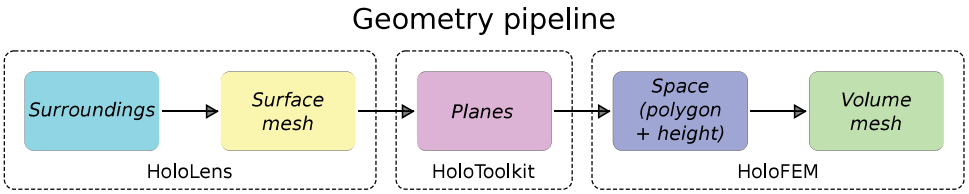

Figure 1: The geometry pipeline shows the steps in going from a spatial scan of the surroundings to a volume mesh.

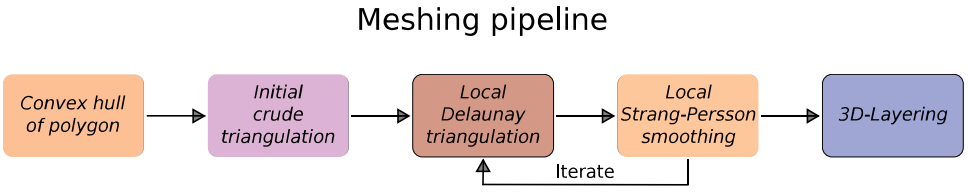

Figure 2: The meshing pipeline demonstrates the generation of a volume mesh from a space, i.e., the last arrow in the geometry pipeline.

2.2 Simulation

The user may configure the problem parameters by placing point sources in the surroundings and setting boundary conditions on the walls, floor, and ceiling of the room. This is done in a similar fashion to how holograms are usually placed with the HoloLens.

When the user is satisfied with the problem specification, the problem is discretised with the finite element method. The FEniCS form compiler [4, 5] is used to generate code for the finite element assembly, see Figure 3. This means that this part of the program can easily be generalised to handle other PDEs. The assembled sparse linear system can be solved with standard techniques, e.g., preconditioned Krylov subspace methods.

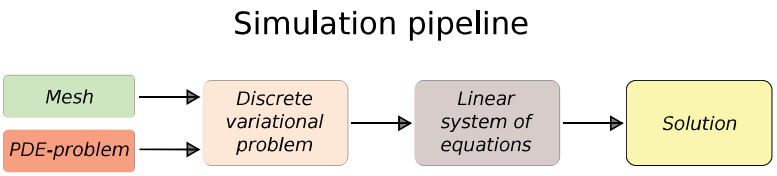

Figure 3: The simulation pipeline outlines how a mesh and PDE-problem are used to obtain a solution with FEM.

2.3 Visualisation

The visualisation of the numerical solution takes advantage of the mixed reality technology of the HoloLens. Holograms are placed at nodal points of the computational mesh, or at the cell centers, and are superimposed on the real world background. This simplistic approach suffices for the current prototype. More sophisticated approaches could be developed specific to the engineering applications considered.

3 Demo overview

In its current state the application HoloFEM works mainly with voice commands. A user, wearing the HoloLens, starts by saying "Scan room". This initiates the scanning phase in which the surroundings are scanned by looking around the room. During the scanning phase a surface mesh is produced that shows what surfaces have already been scanned, see the left part of Figure 4. After the scanning phase is completed, the approximative geometric representation of the room (a prism with polygonal base) is automatically created. A tetrahedral mesh is then generated with the voice command "Generate mesh". In the right part of Figure 4, parts of a space and a volume mesh are displayed.

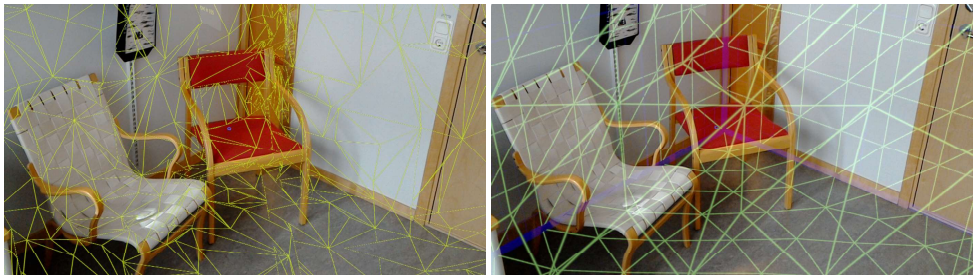

Figure 4: Meshes. *Left*: Surface mesh (yellow) of the surroundings. *Right*: Space (blue) and generated volume mesh (green).

When the mesh has been generated, the user may define additional problem data. The voice command for defining a source is "Create source". This places a source in front of the user. The voice command for defining boundary conditions is "Set boundary value". This sets the solution to be zero on the wall the user is looking at. In Figure 5, visualised problem data are shown.

Figure 5: Problem data represented by holograms. *Left*: Source (fire ball) placed in room. *Right*: Zero value boundary conditions (ice patches) on wall.

Once the desired problem data have been defined it is time to solve the problem. The voice command for that is "Solve problem". After the problem has been solved, the solution is automatically visualised. The solution values at the nodal points of the volume mesh are represented by spheres. The size of a sphere is proportional to the solution value. The spheres are also coloured according to an RGB-scale, where blue represents low values and red high values. See Figure 6 for a visualised solution.

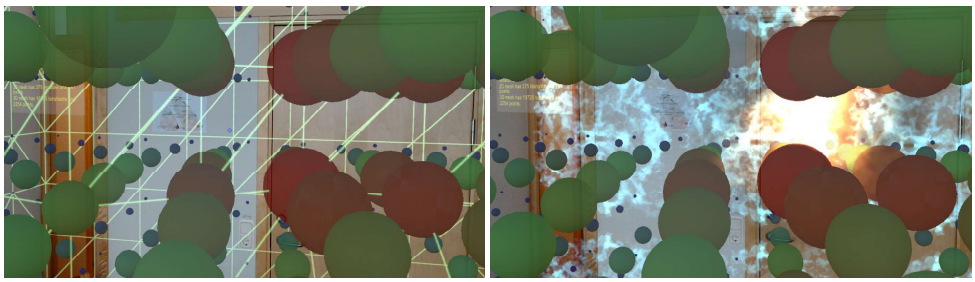

Figure 6: Solution represented by spheres. *Left*: Solution and tetrahedral mesh used in simulation. *Right*: Solution together with problem data used in simulation.

4 Conclusions

We have developed an application for solving and visualising a PDE model with the Microsoft HoloLens. A user wearing the HoloLens can scan the surroundings, define a mathematical model and see the numerical solution superimposed on the real world, all within a matter of seconds. This has potential applications in building planning and safety engineering. Future development includes extension to other PDE models and more sophisticated visualisation.

References

[1] Microsoft. Microsoft HoloLens Homepage. Date accessed: 2017-09-14. [Online]. Available: https://www.microsoft.com/en-us/hololens

[2] A. Bendinger, "New approaches in urban simulation and city planning using virtual reality tools," in *1º Congreso Internacional Ciudad y Territorio Virtual, Barcelona, 3, 14 y 15 Septiembre 2004*. Centre de Política de Sòl i Valoracions, 2004, pp. 67–74.

[3] V. Heuveline, S. Ritterbusch, and S. Ronnas, "Augmented Reality for Urban Simulation Visualization," *Preprint Series of the Engineering Mathematics and Computing Lab*, vol. 0, no. 16, 2011.

[4] A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells, "FFC: the FEniCS form compiler," *Automated Solution of Differential Equations by the Finite Element Method*, pp. 227–238, 2012.

[5] A. Logg, K.-A. Mardal, and G. Wells, *Automated solution of differential equations by the finite element method: The FEniCS book*. Springer Science & Business Media, 2012, vol. 84.

Paper III

Solving differential equations in mixed reality

Anders Logg, Carl Lundholm, Magne Nordaas

Abstract

We present a system for solving differential equations in mixed reality, implemented as an application on the Microsoft HoloLens. The system allows a user to interactively simulate physical phenomena in real-world geometries, by scanning the surroundings and automatically creating a finite element discretization (mesh) for simulation. The system also provides an intuitive interface for specification of boundary conditions and source terms (via voice commands), as well as interactive simulation and visualization. All computations are carried out on the HoloLens and simulation results are visualized as holograms. The current implementation, named HoloFEM, demonstrates the application of mixed reality simulation for the time-dependent advection–diffusion equation. The model is applicable to the simulation of indoor climate (temperature and air quality), which has implications for interactive HVAC (heating, ventilation, and air conditioning) simulation.

Keywords: mixed reality, augmented reality, PDE, FEM, FEniCS, HoloLens, HVAC, advection–diffusion

1 Introduction

Mixed reality merges virtual reality with the real world by creating an environment where real and virtual objects may interact. This has a lot of potential in both industry and education since interaction with 3D objects is a natural way for us to learn and gain understanding. The flexibility in development and design of virtual objects is also high, and the production costs are usually lower than those of the real world counterparts, especially when making copies. It is therefore not surprising that the interest in mixed reality (and similar technologies) has increased over the last years as the required hardware has become smaller and less expensive. For literature, see, e.g., [1, 2].

In the current work, we present a software application (app) developed by us for Microsoft's mixed reality glasses HoloLens [3], see Figure 1. The app is named HoloFEM and it allows a user to define and solve a physical problem in the real-world space where the user is located. Both problem data and solution are represented by holograms, providing an interactive problem-solving environment in mixed reality. The physical problems we consider are governed by partial differential equations (PDEs) and hence also referred to as PDE-problems. A basic ingredient in the formulation of a PDE-problem is always the solution domain Ω in which the PDE should hold. HoloFEM uses the real world surroundings as spatial input to define the solution domain. It is then up to the user to define additional problem data, such as boundary conditions, before solving the problem. The PDE-problems are solved numerically with the finite element method (FEM); hence the name HoloFEM.

The first version of HoloFEM, presented in [4], allowed a user to define and solve a Poisson problem which was used to model a physical system at equilibrium. In the current work, we extend the initial version of HoloFEM in several ways. In addition to a number of improvements to the user interface of the holographic problem-solving environment, the model has been extended to the time-dependent domain and may thus be used to simulate also transient problems. Furthermore, advective transport is now taken into account by solving a multiphysics system to first compute the flow of air in a room and then using the computed flow as input to the time-dependent advection–diffusion equation.

The advection–diffusion equation can be used to model how the concentration of a substance changes in space and over time as a result of macroscopic (advective) and microscopic (diffusive) transport forces. An application could be in the study of how the design of a ventilation system affects the efficiency of removing a toxic gas from a room. Such a set-up falls into the category of heating, ventilation, and air conditioning (HVAC). HVAC simulations are of importance when designing buildings and there exist various software packages with specialized support for HVAC-simulations [5, 6, 7]. The use of mixed

reality in architecture and built environment is not new [2, 8, 9], but to our knowledge, HoloFEM is at the moment the only system that combines PDE with mixed reality.

The current scope of the HoloFEM application is restricted by the hardware limitations of the HoloLens, since all computations are performed on the HoloLens itself. This results in a loss of physical accuracy in comparison with more advanced PDE-solving software. The current version of the HoloFEM application should thus be viewed as a proof-of-concept of combining mixed reality with PDE, rather than as an engineering tool for design of HVAC systems.

Another area of application for mixed reality is education [1, 10, 11, 12]. Since HoloFEM lets a user be immersed in a holographic environment representing a PDE-problem and its solution, it could aid in the intuitive understanding of PDEs, e.g., how variations in data effect the solution. HoloFEM therefore has the potential of being an engaging supplementary tool in areas such as engineering education and mathematics communication.

Figure 1: The Microsoft HoloLens.

2 Mathematical model

We want to model the flow of air and the concentration of a gaseous substance in a physical domain such as an office space. The model may not be too advanced, since we want to simulate the system on the HoloLens which has limited computational power. Restrictions and simplifying assumptions must therefore be made.

To model the air flow in a room, we use potential flow, thus restricting the flow to be irrotational. This is a reasonable assumption for the situation at hand since what we want is basically a flow from one wall to another. To compute a flow potential we use Laplace's equation, thus also restricting the flow to be incompressible and divergence-free. To represent the real-world room, we consider a domain $\Omega \subset \mathbb{R}^3$ with boundary $\partial\Omega$. We consider a partition of $\partial\Omega$ into three parts denoted Γ_{in}, Γ_{out}, and Γ_{wall}, where Γ_{in} is inlet, Γ_{out} is outlet, and Γ_{wall} denotes walls, floor and ceiling (impenetrable). The Laplace problem for the flow potential is thus formulated as: Find the flow potential $u : \Omega \to \mathbb{R}$ such that

$$\begin{cases} -\Delta u = 0 & \text{in } \Omega, \\ u = u_{\text{in}} & \text{on } \Gamma_{\text{in}}, \\ u = u_{\text{out}} & \text{on } \Gamma_{\text{out}}, \\ n \cdot \nabla u = 0 & \text{on } \Gamma_{\text{wall}}, \end{cases} \tag{1}$$

where $\Delta = \nabla \cdot \nabla$ is the Laplace operator, u_{in} and u_{out} are given potentials such that $u_{\text{in}} > u_{\text{out}}$, and n is the outward pointing normal vector to the boundary. Hence, we define the potential flow field $\beta : \Omega \to \mathbb{R}^3$ by

$$\beta := -\nabla u. \tag{2}$$

Note that the boundary condition for Γ_{wall} ensures that there is no flow β through the walls, since $n \cdot \beta = -(n \cdot \nabla u) = 0$.

To model the concentration of a gas in the same room, we use the time-dependent advection–diffusion equation which describes how the concentration changes in space and time as a result of diffusion and advection, where the advective velocity field is taken to be β. We consider a final time T and the same domain Ω with the same boundary partition Γ_{in}, Γ_{out}, and Γ_{wall} as for the Laplace problem (1). We assume that advective transport dominates diffusive transport in the normal direction at the inlet and outlet. We also assume that the air flowing into the room through the inlet does not contain any gas, that there is no transport of gas through the walls, and at initial time $t = 0$ there is no gas in the room. The time-dependent advection–diffusion problem is thus

formulated as: Find the concentration $c : \Omega \times (0, T] \to \mathbb{R}$ such that

$$\begin{cases} \dfrac{\partial c}{\partial t} - \nabla \cdot (D \nabla c) + \beta \cdot \nabla c = f & \text{in } \Omega \times (0, T], \\ c = 0 & \text{on } \Gamma_{\text{in}} \times (0, T], \\ n \cdot \nabla c = 0 & \text{on } \Gamma_{\text{out}} \cup \Gamma_{\text{wall}} \times (0, T], \\ c = 0 & \text{in } \Omega \times \{0\}, \end{cases} \tag{3}$$

where $\frac{\partial c}{\partial t}$ is the partial time-derivative of c, D is the diffusivity, β is the velocity field defined by (2), f is a source, and n is the outward pointing normal vector to the boundary. To motivate the choice of boundary conditions in (3), we consider the total flux q of the gaseous substance given by

$$q = q_{\text{diff}} + q_{\text{adv}} = -D \nabla c + \beta c, \tag{4}$$

The total net flux of gas through the boundary is $n \cdot q$. By making suitable choices of D, u_{in}, and u_{out}, the assumption that advection dominates diffusion in the normal direction at the inlet and outlet may be realized. This assumption together with the boundary conditions in problem (3) and the Laplace problem (1) give

$$n \cdot q|_{\Gamma_{\text{in}}} = (n \cdot q_{\text{diff}} + n \cdot q_{\text{adv}})|_{\Gamma_{\text{in}}} \approx n \cdot q_{\text{adv}}|_{\Gamma_{\text{in}}} = (n \cdot \beta) c|_{\Gamma_{\text{in}}} = 0, \tag{5}$$

$$n \cdot q|_{\Gamma_{\text{out}}} = -D(n \cdot \nabla c)|_{\Gamma_{\text{out}}} + n \cdot q_{\text{adv}}|_{\Gamma_{\text{out}}} = n \cdot q_{\text{adv}}|_{\Gamma_{\text{out}}}, \tag{6}$$

$$n \cdot q|_{\Gamma_{\text{wall}}} = -D(n \cdot \nabla c)|_{\Gamma_{\text{wall}}} + (n \cdot \beta) c|_{\Gamma_{\text{wall}}} = 0 + 0. \tag{7}$$

This means that there is no advective flux of gas through the inlet so no gas enters the room there. Gas is however allowed to leave the room by diffusion, but by the assumption that advection dominates diffusion in the normal direction at the inlet, the diffusive flux of gas through the inlet is negligible. This assumption can be further strengthened by placing the gas source(s) sufficiently far away from the inlet. The only transport of gas through the outlet is due to advection, thus also relying on the dominance of advection over diffusion in the normal direction to be a good description of the system. There is no diffusive flux nor any advective flux of gas through the walls.

Another reason for the choice of boundary conditions is that they are convenient to use in FEM. We solve the Laplace problem (1) approximatively by using FEM with continuous piecewise linear elements. For the time-dependent advection–diffusion problem (3), we use FEM in space with continuous piecewise linear elements and Crank–Nicholson in time. The current implementation does not make use of any additional stabilization (like SUPG). Thus, the stability of the solution relies on the balance of not letting advection dominate diffusion too much which is typically not a problem for the application at hand.

3 Algorithm and implementation

Application development for the HoloLens relies on two main tools: the Unity game engine [13] and the integrated development environment Microsoft Visual Studio [14]. Projects are created in Unity and then exported to Visual Studio where programming in C# takes place. Finally, the projects are uploaded to the HoloLens. The process may be summarized by the development pipeline

$$\boxed{\text{Unity}} \rightarrow \boxed{\text{Visual Studio}} \rightarrow \boxed{\text{HoloLens}}.$$

The current version of HoloFEM was developed using Unity 5.5.0 and Microsoft Visual Studio 2017. Applications can also be run on the HoloLens emulator, allowing quick testing during development. Many of the figures in this section contain screenshots from the HoloLens emulator environment (figures with black background).

In the following, we explain the algorithms behind and the implementation of the HoloFEM system. In Section 3.1 and 3.2, we first describe how the real-world surroundings are used to generate a mesh for FEM. Section 3.3 deals with the definition of additional problem data that are used to complete the specification of the PDE-problems. In Section 3.4 and 3.5, we explain how the discrete PDE-problems are assembled and solved. Finally, we describe how the solutions are visualized in Section 3.6.

3.1 Spatial input

The HoloLens provides functionality for scanning the surroundings and then representing the geometry by a surface mesh. Holograms projected by the HoloLens may then appear to interact with real world objects when in fact they interact with the surface mesh. See Figure 2.

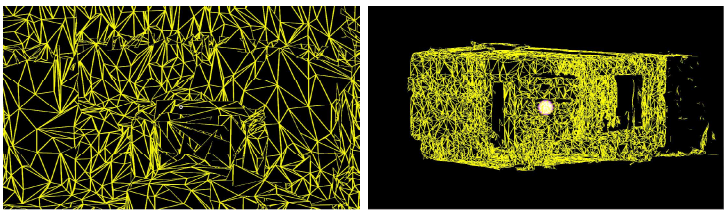

Figure 2: Surface mesh representing the surroundings seen from the inside and outside of a room.

The surface mesh is used to compute confining planes. This is done with the freely available package HoloToolkit from Microsoft. Confining planes are shown in Figure 3. The confining planes are in turn used to compute a geometry that approximates the real world room. We call this geometry a *space* and the computation of it is done by HoloFEM itself. Planes with a *horizontal* normal vector (usually stemming from walls) are used to compute a polygon. The lines where these planes intersect are projected onto the horizontal plane, defining the vertices of the polygon. Planes with a *vertical* normal vector (usually two that stem from floor and ceiling) are then used to define an extrusion of the horizontal polygon in the vertical direction. The resulting space is a polygonal prism, defined by a polygonal base and a height. See Figure 4.

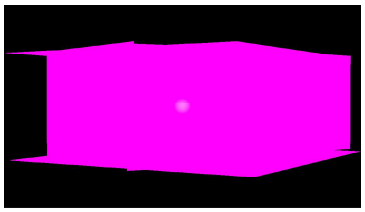

Figure 3: Confining planes seen from the outside of a room.

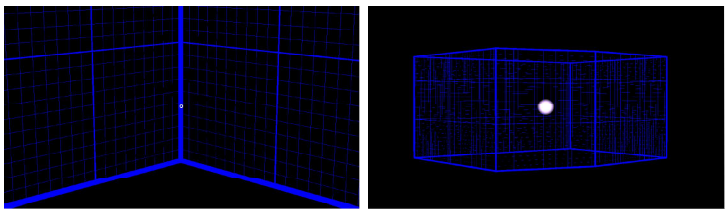

Figure 4: Approximating space as seen from the inside and outside of a room.

A tetrahedral volume mesh is then generated with the space as input. This is also done by HoloFEM. See Figure 5. A more detailed explanation of the mesh generation is given in Section 3.2.

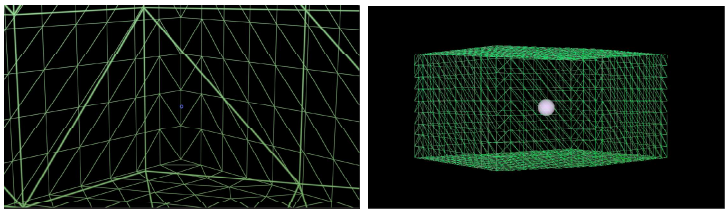

Figure 5: Tetrahedral volume mesh seen from the inside and outside of a room. No full tetrahedra are shown, instead triangles on the boundary of tetrahedra are shown if they belong to the boundary of the mesh or if they are intersected by a sphere centered around the user.

The process so far may be summarized in the geometry pipeline shown in Figure 6.

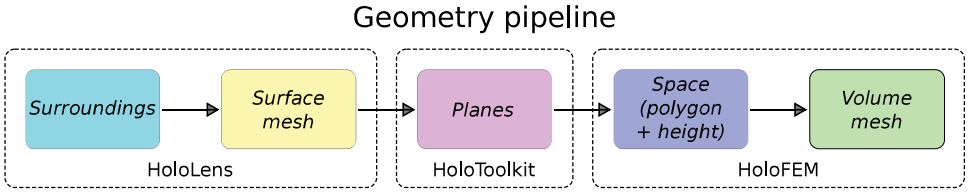

Figure 6: The geometry pipeline shows the steps in going from a spatial scan of the surroundings to a volume mesh.

3.2 Mesh generation

A core component of FEM is the computational mesh of the solution domain. For HoloFEM, the solution domain is an approximation of the real world room where the Laplace problem (1) and the time-dependent advection–diffusion problem (3) are to be solved. From the surroundings, a space approximating the real world room is computed as explained in Section 3.1. This space is used as the starting point for meshing. In the same fashion as the space is computed by extruding a polygon in the vertical direction to form a prism, the mesh is computed by first generating a 2D triangular mesh and then adding layers in the vertical direction.

To generate the 2D triangular mesh, a simple algorithm for Delaunay triangulation is performed. The algorithm first computes the convex hull of the base polygon of the space using a Graham-Scan [15]. An initial mesh can then be created by simply connecting the first (arbitrary) vertex of the convex hull with all other vertices. Vertices are then sampled from a uniform equilateral grid in the interior of the convex hull, one vertex at a time. For each such vertex, it is checked whether the vertex is contained in one of the existing triangles. If so, the triangle is split into three new triangles by connecting the new vertex to all three vertices of the triangle. If the vertex happens to be located on an edge (or very close), the edge is split into two edges, as well as the neighboring triangles. After each addition of a vertex, the standard Delaunay in-circle test is performed and non-Delaunay edges are flipped. This ensures that the mesh remains Delaunay in each step of the triangulation. The step-wise Delaunay triangulation is illustrated in Figure 7.

After the Delaunay triangulation has been computed, a local Persson–Strang smoothing [16] is performed. This procedure is based on viewing the vertices as points connected by springs sitting on each edge of the mesh. By making the natural length of each spring slightly longer than the initial length, the

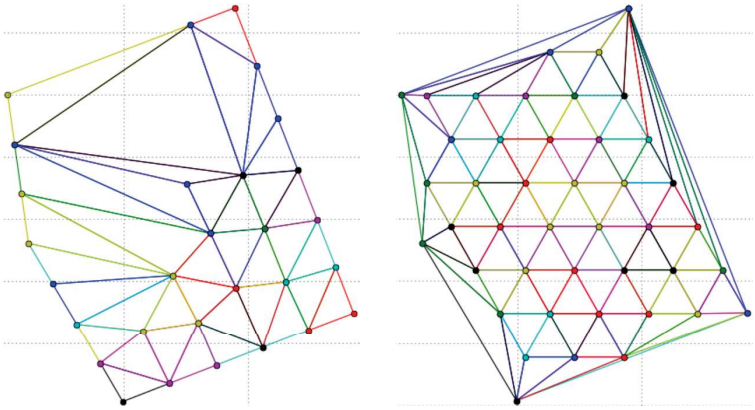

Figure 7: Delaunay triangulation of the convex hull of a scanned space.

vertices will be pushed towards the boundary of the mesh. After a few time steps, a new Delaunay triangulation is performed using the new positions of the vertices as seed points, and the Persson–Strang smoothing is repeated. This procedure of first generating a Delaunay mesh using the current vertex positions and then letting the point–spring system tend to equilibrium, is repeated until convergence. The mesh thus obtained is typically of very good quality (triangles close to equilateral). Finally, triangles generated on the outside of the base polygon (but inside of the convex hull) are removed from the triangulation.

To generate the 3D tetrahedral mesh, copies of the 2D triangular mesh are stacked parallel exactly above each other with equal distance in the height direction. In two adjacent layers, every triangle in one layer will have an exact copy of itself in the other layer. Two such triangles define a prism with a triangular base and such a prism may be partioned into three tetrahedra. The tetrahedra will have at least one vertex in both triangles, thus connecting the layers and defining a 3D tetrahedral mesh. Figure 8 shows a 2D mesh generated by the Persson–Strang smoothing and the corresponding layered 3D mesh discretizing a scanned space.

The mesh generation algorithm is typically very fast (ca. 1 s) relative to the overall simulation time, but sometimes fails to converge due to rounding errors and the algorithm being implemented in single precision on the HoloLens.

3.3 Problem definition

A PDE-problem consists of a PDE and problem data. For the discrete FEM formulation of the PDE-problem, the problem data consist of a mesh and all the given functions such as source terms and boundary conditons. Once a

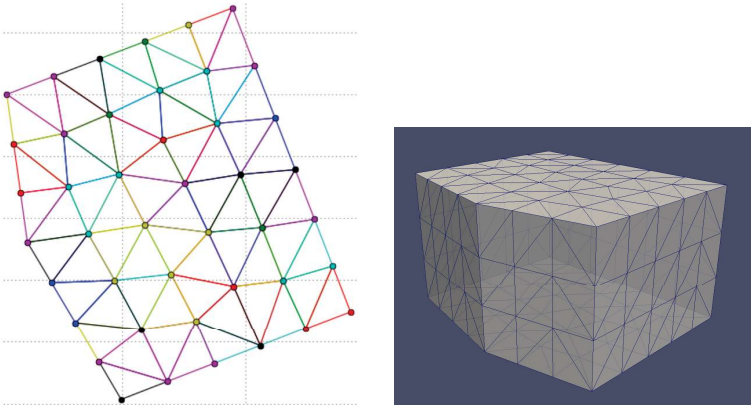

Figure 8: (Left) A 2D mesh generated by the Delaunay/Persson–Strang smoothing algorithm. (Right) The corresponding layered 3D mesh.

computational mesh has been generated, the user needs to define additional problem data, which is done via voice commands.

The additional problem data needed for the Laplace problem (1) are boundary conditions defining inlet and outlet for air flow in a room. By using the Möller–Trumbore algorithm for ray-triangle intersection [17], we compute which of the boundary triangles of the mesh that are intersected by the HoloLens gaze. The centroid of this triangle is used as the center of a disc with a predefined radius. All boundary nodal points of the mesh that lie inside this disc are computed and the corresponding degrees of freedom in the linear system of equations are later assigned the given boundary values. Boundary nodal points not belonging to an inlet or outlet are automatically treated as belonging to wall boundary regions. The inlet and outlet regions (discs) are visualized by holograms. See Figure 9.

Figure 9: Holograms representing the boundary regions (Γ_{in} and Γ_{out}) and the source term (f).

The additional problem data needed for the time-dependent advection–diffusion problem (3) are boundary conditions, velocity field and sources. The

boundary conditions are defined simultaneously when defining boundary conditions for the Laplace problem (1). The velocity field is automatically computed from the solution to the Laplace problem. This is explained in Section 3.4. Sources are placed with center points $\bar{\mathbf{x}} \in \mathbb{R}^3$ at the midpoint of the line segment between the user and the cursor. The source terms are gaussian functions in space $f : \mathbb{R}^3 \to \mathbb{R}$ defined by

$$f(\mathbf{x}) = a \exp(-b|\mathbf{x} - \bar{\mathbf{x}}|^2), \tag{8}$$

where $a, b > 0$ are constants chosen such that the source terms return large values close to the center points and decrease rapidly towards zero away from them. A particle system designed to look like a gas cloud is used to represent sources with holograms. See Figure 9 for a close-up and Figure 10 for an overview of holograms representing problem data.

The structural treatment of boundary conditions and sources in the code is very similar. When defining a boundary condition or a source, a specific C#-method is called. Two objects are created by the method: the first contains the specified problem data that will be used in the assembly of the linear system, and the second is a hologram representing the problem data. Both objects are stored in specific lists that contain all the objects of the same type created so far. The C#-method that does this for sources is presented in Figure 11.

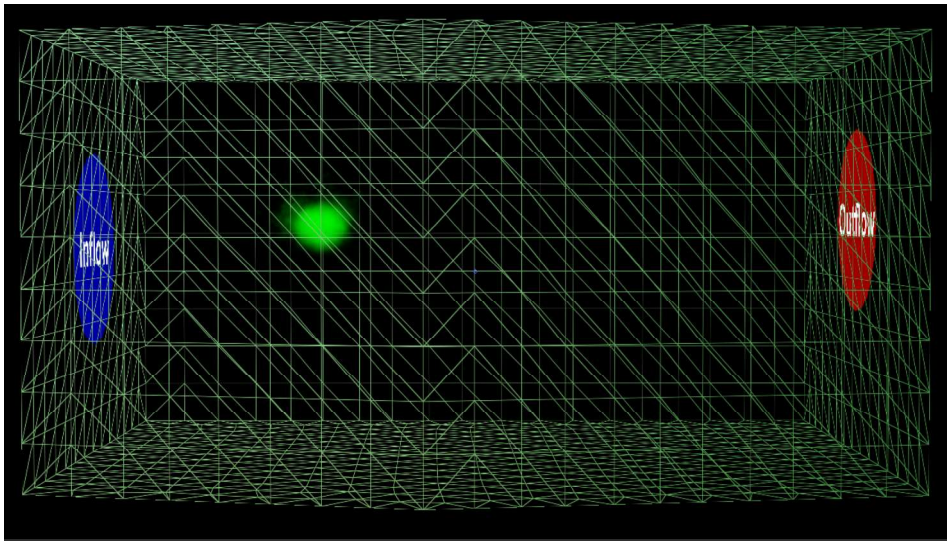

Figure 10: Holograms representing problem data for the PDE-problems (1) and (3) seen from outside a room. The domain is represented by a tetrahedral volume mesh (green grid), the gas source by a green cloud, and the inlets and outlets by blue and red discs, respectively.

```
// Create source
public void CreateSource()
{
    // Get current gaze origin
    Vector3 gazeOrigin = GazeManager.Instance.GazeOrigin;

    // Get current hit position
    Vector3 hitPosition = gazeOrigin;
    hitPosition = GazeManager.Instance.HitPosition;

    // Compute source position
    Vector3 sourcePosition = 0.5f*(gazeOrigin + hitPosition);

    // Add the source to the list of sources
    Sources.Add(new GaussianSource(sourcePosition));

    // Print source location
    SpatialPrinting.Print("Creating source at "
                          + sourcePosition.ToString());

    // Visualize source
    GameObject visualSource =
        GameObject.Instantiate(Resources.Load("GasSource")) as GameObject;
    Transform transform = visualSource.GetComponent<Transform>();
    transform.Translate(sourcePosition);

    // Add game object to visualization list
    VisualSources.Add(visualSource);
}
```

Figure 11: Code snippet containing HoloFEM's C#-method `CreateSource()`. The method is called when defining a source. Rows starting with "//" are comments explaining the code.

3.4 Finite element discretization

After problem data have been defined, a linear system of equations $AU = b$ has to be assembled and solved. Here, A is the system matrix, U is the solution vector for the degrees of freedom of the finite element solution, and b is the right-hand side vector.

The finite element method with continuous piecewise linear elements is used to obtain linear systems of equations for spatially discretized versions of problems (1) and (3). The assembly of the systems is partially automated by using the FEniCS Form Compiler (FFC) [18]. The input to FFC is a file containing

code written in the domain specific language UFL [19]. UFL is used to declare finite element discretizations of variational forms in FEniCS [20]. An example of UFL code declaring variational forms for the stationary advection–diffusion equation is shown in Figure 12. The output from FFC is a C++ header file containing functions for computing the local contributions to A and b. The functions take problem data as input and return the local element tensor (matrix or vector). The use of FFC makes application development more convenient and also makes it easier to extend HoloFEM to solve other types of PDE-problems. Note that since FFC generates C++ code (not C#), some minor manual editing is necessary to include the generated code in the HoloFEM C# application. The assembled systems are solved with the preconditioned conjugate gradient method, implemented in C# as part of HoloFEM.

```
# Stationary advection-diffusion equation

# Define element
element = FiniteElement("P", tetrahedron, 1)

# Define functions
c = TrialFunction(element)
v = TestFunction(element)
u = Coefficient(element)  # flow potential, -grad(u) = flow field
f = Coefficient(element)  # source

# Define constant(s)
D = Constant(tetrahedron) # diffusivity

# Define bilinear and linear forms
a = D*inner(grad(c), grad(v))*dx + dot(-grad(u), grad(c))*v*dx
L = f*v*dx
```

Figure 12: Code snippet containing UFL-code for the stationary advection–diffusion equation.

3.5 Time discretization

For a partition $0 = t_0 < t_1 < \ldots < t_{n-1} < t_n < \ldots < t_N = T$ of the time interval $[0, T]$, we use the Crank-Nicholson method to obtain a temporal discretization of the time-dependent advection–diffusion problem (3). This results in a time-stepping scheme where a linear system $AU_n = b_n$ has to be solved at every discrete time t_n. The system matrix A does not depend on time. Hence, it is sufficient to assemble it once initially and reuse it during the time-stepping to increase computational efficiency. The right-hand side vector b_n depends on the solution at the previous time U_{n-1} so it has to be updated. Pseudocode for solving the time-dependent advection–diffusion problems as outlined in this section is shown in Algorithm 1. The actual solver implementation in C# for HoloFEM is shown in Figure 13 – 14.

Algorithm 1 Solving the discretized time-dependent PDE-problem

Input: Problem data (mesh, boundary conditions, sources, etc.)
Output: Solution stored as time-series

1: Initialize system matrix A and right-hand side vector b
2: Assemble matrix A using code generated by FFC and problem data
3: Apply boundary conditions to A
4: Initialize time t, solution vector U, and time-series list U_{list}
5: **while** $t_n < T$ **do**
6: Assemble vector b_n using code generated by FFC and problem data
7: Apply boundary conditions to b_n
8: Solve system $AU_n = b_n$ using the conjugate gradient method
9: Add U_n to the list U_{list}
10: Update t_n

```
public static List<Vector> SolveTransientAdvDiff(AdvectionDiffusionProblem problem)
{
    // Initialize system matrix and RHS vector
    int vs = problem.Mesh.Points.Count;
    SpatialPrinting.Print("System size is " + vs);
    Matrix A = new Matrix(vs, vs);
    Vector b = new Vector(vs);

    // Create assembler object
    Assembler AO = new Assembler(problem.Mesh,
                                 problem.Sources,
                                 problem.FlowPotential,
                                 problem.Diffusivity);

    // Assemble system matrix
    AO.AssembleTransientAdvectionDiffusion3DP1_LHSMatrix(A, problem.TimeStep);

    // Apply boundary conditions to system matrix
    DirichletBC BO = new DirichletBC(problem.AdvectionDiffusionDBCs);
    BO.Apply(A);

    // Create solver object
    PCG SO = new PCG();
    SO.SetOperator(A);

    // Initialize solution vector and use it as initial data, i.e., zero everywhere
    Vector U = new Vector(vs);
    U.Zero();

    // Initialize list for solution vectors
    List<Vector> Ulist = new List<Vector> { U.Copy() };
```

Figure 13: Code snippet containing the *first* half of HoloFEM's solver for the time-dependent advection-diffusion equation implemented as a C#-method. Rows starting with "//" are comments explaining the code. See Figure 14 for the *second* half.

```
    // Initialize list for solution vectors
    List<Vector> Ulist = new List<Vector> { U.Copy() };

    // Start time-stepping
    float t = problem.TimeStep;
    while (t <= problem.FinalTime)
    {
        // Reset RHS vector
        b.Zero();

        // Assemble RHS vector b using, i.a., solution at previous time
        AO.AssembleTransientAdvectionDiffusion3DP1_RHSVector(b, U, problem.TimeStep, t);

        // Apply boundary conditions to RHS vector
        BO.Apply(b);

        // Reset solution vector and solve system
        U.Zero();
        SO.Solve(U, b);

        // Store solution at current time
        Ulist.Add(U.Copy());

        // Update time
        t += problem.TimeStep;
    }

    return Ulist;
}
```

Figure 14: Code snippet containing the *second* half of HoloFEM's solver for the time-dependent advection-diffusion equation implemented as a C#-method. Rows starting with "//" are comments explaining the code. See Figure 13 for the *first* half.

3.6 Solution visualization

The solutions are represented with holograms. The negative gradient of the solution to the Laplace problem is computed and a predefined number of 3D arrows are equally distributed at the centroids of tetrahedra in the mesh. The arrows point in the direction of the negative gradient at the point where they are placed. They are also scaled such that larger arrows represent larger magnitudes of the gradient. See Figure 15.

The solution to the advection–diffusion problem is represented by spheres. The spheres are placed at nodal points of the mesh and colored according to the solution value at that point. They are also scaled such that larger spheres represent larger values of the solution. In Figure 16 – 17 solution spheres at two

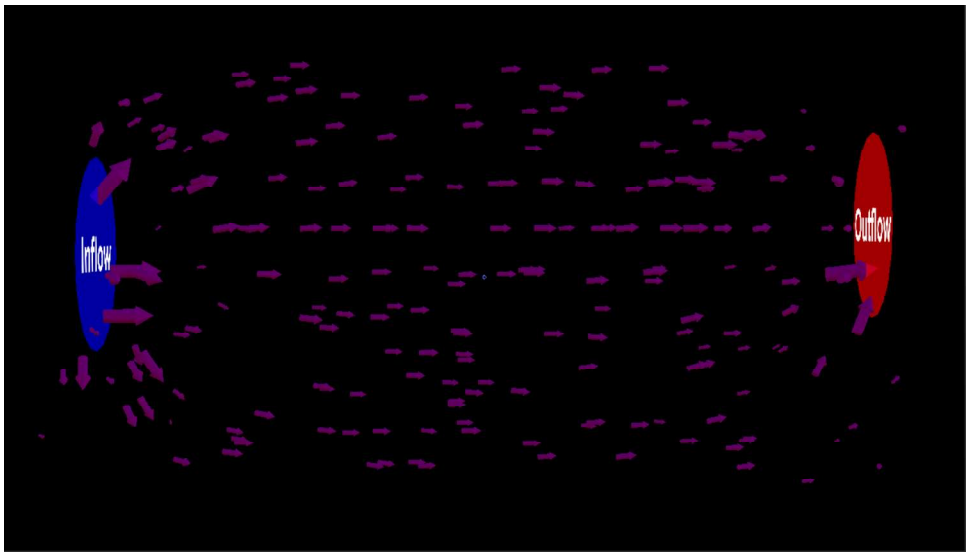

Figure 15: Holograms representing the potential flow obtained from solving a discrete Laplace problem. The flow goes from inlet (blue) to outlet (red).

times are shown. In Figure 16, the animation of the time-dependent solution has just started. In Figure 17, we see the system at a later time when equilibrium has been reached. Note that the spheres indicate an increase of the concentration in time around the gas source (green cloud). Also note the slight shift to the right as a result of the flow from the inlet (blue) to the outlet (red).

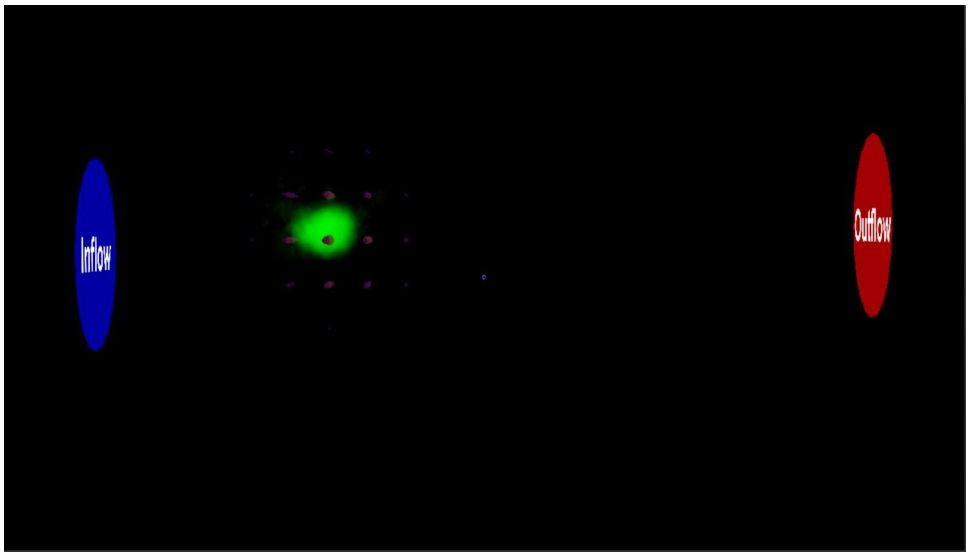

Figure 16: Holograms representing the solution to a discrete time-dependent advection–diffusion problem at an *early* time.

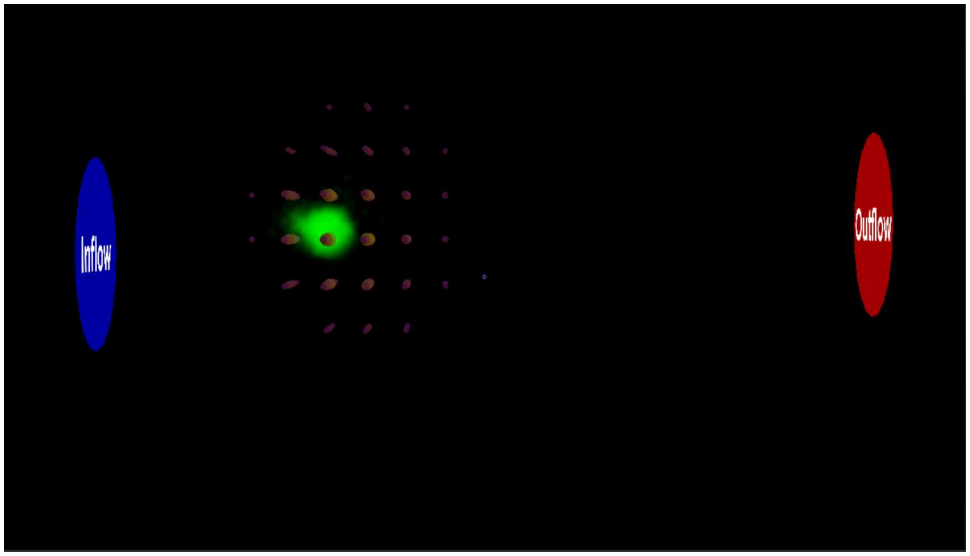

Figure 17: Holograms representing the solution to a discrete time-dependent advection–diffusion problem at a *later* time.

4 Demonstration of the HoloFEM application

User input is mainly supplied to HoloFEM by voice commands. When HoloFEM is launched, the user is greeted by a welcome message on a holographic information display. First the real world room needs to be scanned. This is done by speaking "Scan room" and then looking around. The HoloLens then successively computes the overlaying surface mesh that represents the surroundings, visualizing it as the scanning progresses. After a successful scan, a geometrical representation of the scanned space is computed as described above and then visualized as a grid. See Figure 18.

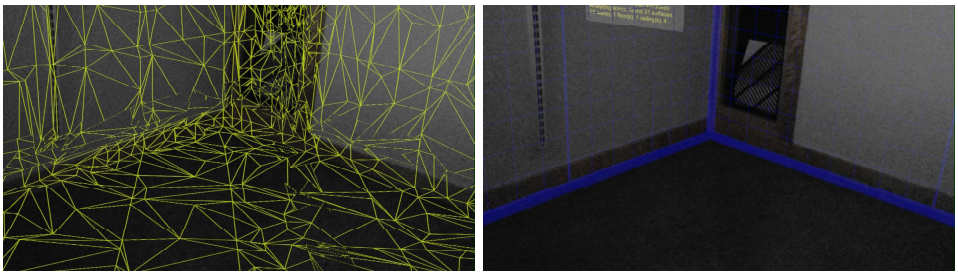

Figure 18: Surface mesh to the left and resulting space to the right.

The next step is to generate a computational volume mesh for FEM. By speaking the voice command "Generate mesh", the mesh generation is initiated. After successful meshing, the volume mesh is visualized in the room. The volume mesh in Figure 19 consists of 2106 nodal points. The number of degrees of freedom in the linear systems is the same, since FEM with piecewise linear elements is used.

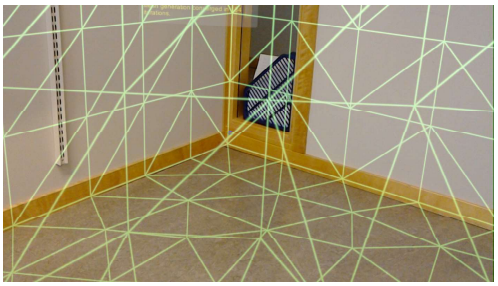

Figure 19: Computational volume mesh generated from the space in Figure 18.

The inlets and outlets of air flow in the room can only be placed on the boundary of the volume mesh, i.e., the walls, floor, and ceiling of the real world room. By looking at a desired location and speaking "Set inflow" or "Set outflow", one defines boundary conditions for inlet or outlet, respectively. See Figure 20.

At this point, a complete discrete Laplace problem has been defined. The voice command to assemble and solve it, thus obtaining

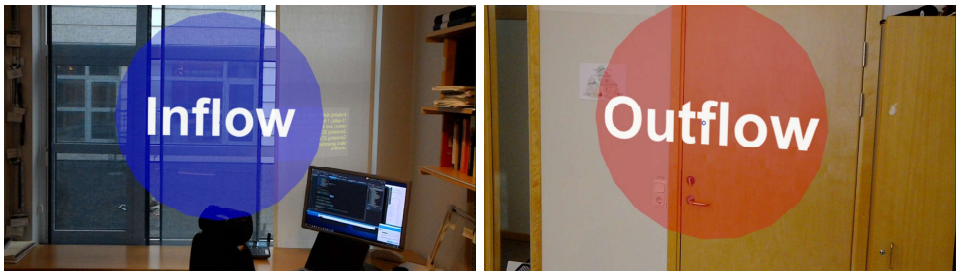

Figure 20: Holograms representing boundary conditions for inlet and outlet.

the air flow in the room, is "Compute flow". Next, 3D arrows representing the air flow are automatically visualized, where larger arrows represent a larger magnitude of the flow. See Figure 21. The total time for assembly, solving, and visualization for the Laplace problem in this demonstration is about 3 s.

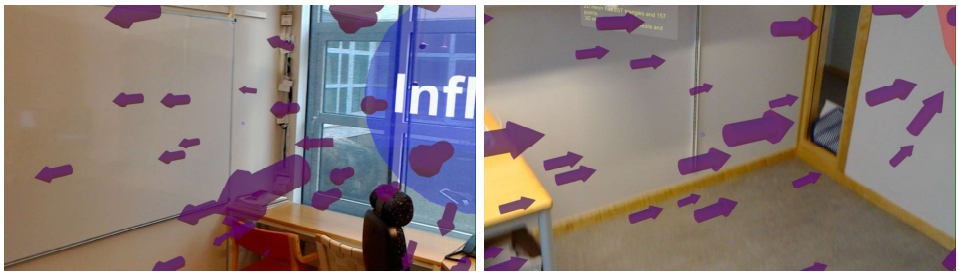

Figure 21: Air flow represented by 3D arrows.

The flow is used as problem data for defining an advection–diffusion problem (3). To make the definition complete, sources are also needed. They are automatically placed at half the distance between the cursor and the user by speaking "Create source". Green gas clouds are used to represent sources. See Figure 22.

Now a complete advection–diffusion problem is defined. The solution is the concentration of a gaseous substance in the room. To assemble and solve the stationary problem, thus computing the concentration when the system has reached equilibrium, one speaks "Solve stationary problem". Once the problem is solved, spheres representing the solution values at the nodal points of the mesh are automatically visualized. The spheres are scaled and colored according to the solution values so that larger spheres represent larger values. The total time for assembly, solving, and visualization for the stationary problem in this demonstration is about 4 s.

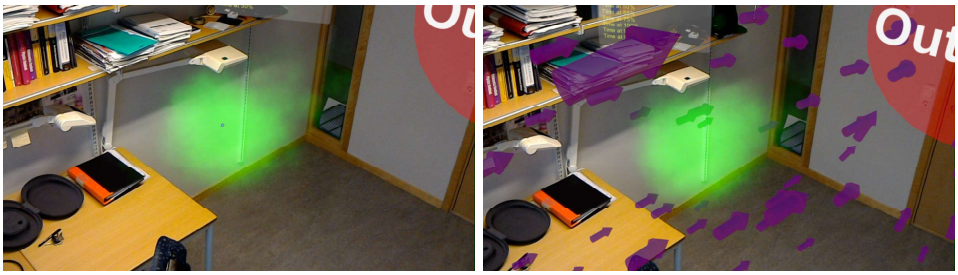

Figure 22: Gas cloud representing a source without and with holograms for air flow.

To assemble and solve the time-dependent problem one speaks “Solve time-dependent problem”. During the solve process, the computed concentration in the room at different times is stored in a time series. Once the problem has been solved it is up to the user how to visualize it. By speaking “Animate time-dependent solution” an animation of the concentration’s evolution in time is started. By speaking “Plot time-dependent solution”, “Forwards in time” and ”Backwards in time”, a user may freely inspect the concentration in the room at various times. See Figure 23 and 24. The total time for assembly and solving for the time-dependent problem in this demonstration is about one minute, when using 60 time steps. This (of course) amounts to 1 s per frame so real-time animation is not possible. However, by separating the computation and visualization, the solution may be smoothly animated at a higher frame rate.

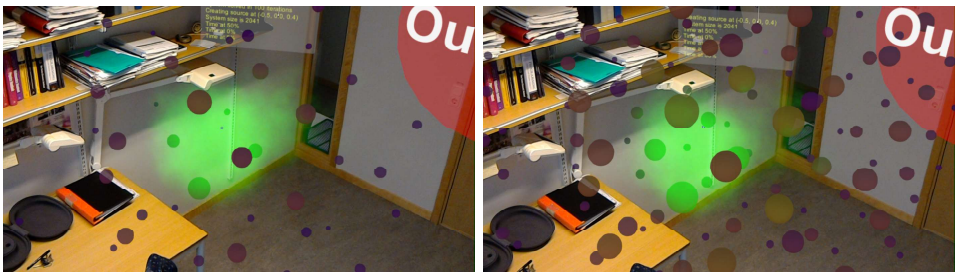

Figure 23: Solution spheres to a time-dependent advection–diffusion problem at an early and a later time with an *active* gas source.

The solution spheres in Figure 23 indicate an increase of concentration in time around the gas source between the two times. Note that they show a shift in the concentration towards the outlet due to the air flow.

Figure 24: Solution spheres to a time-dependent advection–diffusion problem at an early and a later time with a *deactivated* gas source. The gas cloud hologram is left visible for reference.

The solution spheres in Figure 24 show what happens with the concentration when the gas source is deactivated, i.e., stopping the inflow of gas to the room. The spheres decrease in size between the two times and are moved towards the outlet, indicating that gas leaves the room because of the air flow. See also Figure 25, showing holograms used to represent problem data and solutions to the finite element formulations of the Laplace problem (1) and the advection–diffusion problem (3).

Figure 25: Holograms representing data and solution to PDE-problems without and with the computational mesh used visible (in the office space of one of the authors).

5 Conclusions and outlook

We present a mixed reality app for defining and solving PDE-problems in real world surroundings. The current state of the app serves mainly as a proof-of-concept since PDE-problems of interest in industry and real world applications often are of a higher complexity than the model problems presented here. A way for the app to tackle more complex problems is to do all demanding computations on an external server, thus relieving the HoloLens of this burden. The idea would be to collect spatial data, define a PDE-problem, send the data to an external server for processing, and then receive solution data for visualization once the external computations are done.

References

[1] C. Hughes, C. Stapleton, D. E Hughes, and E. Smith, "Mixed Reality in Education, Entertainment, and Training," *IEEE computer graphics and applications*, vol. 25, pp. 24–30, 2005.

[2] X. Wang and M. A. Schnabel, *Mixed reality in architecture, design and construction.* Springer, 2009.

[3] Microsoft. Microsoft HoloLens Homepage. Date accessed: 2019-03-12. [Online]. Available: https://www.microsoft.com/en-us/hololens

[4] A. Logg, C. Lundholm, and M. Nordaas, "Solving Poisson's Equation on the Microsoft HoloLens," in *Proceedings of the 23rd ACM Symposium on Virtual Reality Software and Technology*, ser. VRST '17. New York, NY, USA: ACM, 2017, pp. 87:1—-87:2. [Online]. Available: http://doi.acm.org/10.1145/3139131.3141777

[5] SimScale. HVAC Simulation Software in the Cloud with SimScale Homepage. Date accessed: 2019-02-28. [Online]. Available: https://www.simscale.com/hvac/

[6] SimulationX. HVAC - SimulationX Homepage. Date accessed: 2019-02-28. [Online]. Available: https://www.simulationx.com/industries/applications/hvac.html

[7] Autodesk. Building HVAC Simulation — CFD Software — Autodesk Homepage. Date accessed: 2019-02-28. [Online]. Available: https://www.autodesk.com/solutions/simulation/building-hvac

[8] X. Wang, M. J. Kim, P. E. D. Love, and S.-C. Kang, "Augmented Reality in built environment: Classification and implications for future research," *Automation in Construction*, vol. 32, pp. 1–13, 2013. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0926580512002166

[9] J. Milovanovic, G. Moreau, D. Siret, and F. Miguet, "Virtual and Augmented Reality in Architectural Design and Education An Immersive Multimodal Platform to Support Architectural Pedagogy," 2017.

[10] D. Liu, C. Dede, R. Huang, and J. R. Richards, "Virtual, Augmented, and Mixed Realities in Education," in *Smart Computing and Intelligence*, 2017.

[11] Pearson. Pearson Mixed Reality portfolio Homepage. Date accessed: 2019-03-04. [Online]. Available: https://www.pearson.com/uk/web/pearsontq/our-training-services/immersive-learning/augmented-and-mixed-reality.html

[12] HoloGroup. HoloGroup Education in Mixed Reality Homepage. Date accessed: 2019-03-04. [Online]. Available: https://holo.group/en/education/

[13] Unity. Unity Homepage. Date accessed: 2019-03-05. [Online]. Available: http://unity3d.com/

[14] Microsoft. Microsoft Visual Studio Homepage. Date accessed: 2019-03-05. [Online]. Available: https://visualstudio.microsoft.com/

[15] R. L. Graham, "An efficient algorithm for determining the convex hull of a finite planar set," *Information Processing Letters*, vol. 1, no. 4, pp. 132–133, 1972. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0020019072900452

[16] P.-O. Persson and G. Strang, "A simple mesh generator in MATLAB," *SIAM review*, vol. 46, no. 2, pp. 329–345, 2004.

[17] T. Möller and B. Trumbore, "Fast, Minimum Storage Ray-Triangle Intersection," *Journal of Graphics Tools*, vol. 2, no. 1, pp. 21–28, 1997.

[18] A. Logg, K. B. Ølgaard, M. E. Rognes, and G. N. Wells, "FFC: the FEniCS form compiler," *Automated Solution of Differential Equations by the Finite Element Method*, pp. 227–238, 2012.

[19] M. Alnæs, A. Logg, K. Ølgaard, M. Rognes, and G. Wells, "Unified form language: A domain-specific language for weak formulations of partial differential equations," *ACM Transactions on Mathematical Software*, vol. 40, no. 2, 2014.

[20] A. Logg, K.-A. Mardal, and G.-N. Wells, *Automated solution of differential equations by the finite element method*, 2012, vol. 84 LNCSE.