ARCADE

an interactive playground for immersive topology optimization

Aragón, Alejandro M.; Algra, Hendrik J.

DOI
10.1007/s00158-025-04152-2

Publication date
2025

Document Version
Final published version

Published in
Structural and Multidisciplinary Optimization

Structural and Multidisciplinary Optimization (2025) 68:251
https://doi.org/10.1007/s00158-025-04152-2

EDUCATIONAL PAPER

ARCADE: an interactive playground for immersive topology optimization

Alejandro M. Aragón[1] · **Hendrik J. Algra**[2]

Received: 14 April 2025 / Revised: 17 August 2025 / Accepted: 10 September 2025

Abstract
Topology optimization (TO) has found applications across a wide range of disciplines but remains underutilized in practice. Key barriers to broader adoption include the absence of versatile commercial software, the need for in-depth knowledge of the methodology from the user, and high computational demands. Additionally, challenges such as ensuring manufacturability, tuning hyper-parameters, and integrating subjective design elements like esthetics further hinder its widespread use. Emerging technologies like augmented reality and virtual reality offer transformative potential for TO. By enabling intuitive, gesture-based human–computer interactions, these immersive environments bridge the gap between human intuition and computational processes. They provide the means to integrate subjective human judgment into optimization workflows in real time, creating a paradigm shift toward interactive and immersive design. Here, we introduce the concept of immersive topology optimization (ITO) as a novel design paradigm that leverages augmented reality for TO. By incorporating real-time human interaction during the optimization process, and the subsequent visualization of the design in its intended target location, ITO has the potential to reduce lead times, enhance manufacturability, and improve design integration. To demonstrate this ITO design paradigm, we present ARCADE: Augmented Reality Computational Analysis and Design Environment. Developed in Swift for the Apple Vision Pro mixed reality headset, ARCADE enables users to define, manipulate, and solve structural compliance minimization problems within an augmented reality setting. We provide the source code in Swift for the optimization procedure. While initially demonstrated for minimizing compliance, our framework could also be extended to other disciplines, paving the way for a new era of interactive and immersive computational design.

Keywords Augmented reality · Immersive topology optimization · Human-informed optimization · Apple Vision Pro · visionOS

1 Introduction

Topology optimization (TO) (Bendsøe and Sigmund 2004) has successfully been applied to a myriad of applications spanning many disciplines, including automotive (Yang and Chahande 1995), construction (Vantyghem et al. 2020), chip manufacturing (Delissen et al. 2022), biomechanical (Koper et al. 2020), nanomechanical sensing (Høj et al. 2021), and spacecraft (Guibert et al. 2024), among others. However, the use of TO as a structural design tool is currently more a niche than a standard, as a more widespread adoption in industry is currently hindered by many obstacles. One major reason is the lack of TO implementations in commercial software that extend beyond classic compliance minimization problems. Implementing a general topology optimization problem necessitates a high degree of software flexibility, which is currently lacking in software packages. Furthermore, it requires a highly specialized analyst profile with knowledge on computational modeling and optimization. Setting up the problem involves not only implementing the appropriate formulation—i.e., the objective function and its corresponding sensitivity analysis formulation—but also deciding on the type of material interpolation and filtering functions, the definition of appropriate constraints, and the

Responsible Editor: Josephine Voigt Carstensen.

✉ Alejandro M. Aragón
a.m.aragon@tudelft.nl

[1] Faculty of Mechanical Engineering, Delft University of Technology, Mekelweg 2, 2628 CD Delft, Zuid-Holland, The Netherlands

[2] Faculty of Civil and Environmental Engineering, Technion Israel Institute of Technology, 3200003 Haifa, Israel

Published online: 19 November 2025

tuning of the optimization hyper-parameters. All these may influence considerably the final design obtained. Although attempts have been made to alleviate some of these hurdles (e.g., via machine learning (Ha and Carstensen 2024)), achieving a satisfactory optimization result is often as much an art as a science. Partly to blame is also the inherent time-consuming nature of the TO design process, which usually requires solving large systems of equations (describing one or multiple physical phenomena) hundreds of times (Yano et al. 2021). Although there are methods to accelerate this process, for instance by means of surrogate (or meta-) models such as Kriging or machine learning (Kudela and Matousek 2022), it is not uncommon to set the design resolution (i.e., the mesh size) to the extent that it strains computers for weeks to obtain a final design.

Another important factor that has received less attention is the trial-and-error process of fine-tuning the final design. Once an optimized design is obtained, challenges may arise during implementation: The design may not be manufacturable (e.g., due to overhang angles), fail to meet certain requirements, or require modifications due to newly available information that needs to be integrated into the design process. While some of these aspects have been addressed, such as incorporating manufacturability constraints (Langelaar 2016) or by generating a structure composed of predefined structural components (Zhang et al. 2016), accounting for subjective factors such as esthetics remains challenging (Loos et al. 2022). Such aspects are often difficult, if not impossible, to quantify numerically, making it difficult to systematically integrate them into the design process. However, experienced analysts can often quickly assess undesirable characteristics of a particular structure after just a few iterations, highlighting the need for a means to *steer the optimizer* in a different direction as needed, and in real time. To that aim, many works have proposed means to interact with the topology optimization process in real time. Notable examples include interactive applications in 2-D (Aage et al. 2013) and 3-D (Nobel-Jørgensen et al. 2015), an educational game (Nobel-Jørgensen et al. 2016), and interactive combined topology and shape optimization (Nguyen et al. 2020). Interactive topology optimization has also been explored for controlling local feature size (Ha et al. 2023), for including subjective preferences of the designer (Li et al. 2023), and for infill design targeting additive manufacturing (Schiffer et al. 2024). In order to take this concept to the next level, and overcome other challenges associated with TO procedures in general, recent advancements in computer technology can be of great assistance.

Over the past decades, computers have undergone an extraordinary transformation, both in terms of computing power and their widespread availability. Advances in microchip technology, with an ever-growing number of transistors, have led to an exponential increase in processing capabilities, making devices smaller, faster, and more energy-efficient. Once confined to bulky desktops, computers now fit seamlessly into our daily lives through a wide range of devices, from smart watches and phones, to tablet, laptop, and desktop computers. The way that people interact with these devices has also changed over time. Alternatives to the traditional mouse and keyboard combination, such as touchscreens and voice commands, are now ubiquitous. There has also been an uptake in enabling users to physically interact with the digital world by using augmented reality (AR) and virtual reality (VR). Undoubtedly, there is great potential in applying these technologies to topology optimization. From a conceptual perspective, AR and VR integrate the designer's gestures with digital tools, enhancing their ability to work seamlessly with computing systems and therefore closing the gap between human input and machine response.

We envision a future where AR and VR revolutionize design paradigms, enabling a fully immersive and interactive approach that seamlessly integrates human intuition and judgment into the design process in real time. We have coined the term immersive topology optimization (ITO) to describe this concept. By incorporating human expertise into the optimization workflow and visualizing designs in their intended real-world contexts, ITO has the potential to significantly reduce lead times from conceptualization to manufacturing. A recent step in this direction is the work of Li et al. (2025), who integrated their preference-based topology optimization (Li et al. 2023) with a VR headset. Their VR-BESO design tool allows users to sculpt an initial model within a VR environment and subsequently perform TO using the bi-directional evolutionary structural optimization (BESO) methodology, thereby incorporating user-provided features into the design process. In their setup, sculpting is carried out with a Meta Quest 3 headset and handheld controllers, while a high-performance personal computer is used for the computationally intensive topology optimization and smoothing of the optimized design. The sculpting and optimization steps are performed sequentially, so users interact with the process through repeated sculpting–optimization cycles rather than steering the optimization during execution as carried out by the applications described earlier. This approach nevertheless provides valuable iterative feedback and demonstrates a compelling integration of VR and structural optimization. At the same time, immersive VR environments can pose challenges, including cybersickness, visual and muscular fatigue, acute stress, and mental overload (LaViola 2000; Souchet et al. 2023), as well as reduced spatial awareness that may increase accident risk (Tseng 2023). Beyond health and safety considerations, and to broaden adoption, one avenue could be the integration of visualization and optimization on a single device and the removal of handheld controllers (e.g., through hand

gestures). All these limitations are addressed in the framework described herein.

Apple's Vision Pro mixed reality headset was unveiled on June 5, 2023, marking a significant step forward in immersive spatial computing. By unlocking new possibilities for interaction, visualization, and productivity within mixed reality environments, the advent of this device has allowed us to take a significant step toward realizing our vision of immersive topology optimization. Herein, we introduce ARCADE, which stands for Augmented Reality Computational Analysis and Design Environment. ARCADE is a computational framework *fully developed* in the Swift programming language for the Apple Vision Pro headset. The ARCADE app allows users to define, manipulate, and solve structural (compliance) optimization problems within an immersive AR setting. Interaction is performed exclusively through hand gestures—specifically tap and pinch-and-drag. Users can modify boundary conditions and material volume fraction during optimization, enabling real-time steering of the design. Because the Vision Pro functions as a standalone computer, all processing and visualization occur directly on the device. We also provide the Swift source code for the topology optimization. By leveraging augmented reality, this interactive environment allows users to design within the intended physical context, making real-time adjustments as desired.

2 The ARCADE application

ARCADE is our proof-of-concept application that implements immersed topology optimization. Developed for the Apple Vision Pro mixed reality headset, ARCADE solves a compliance minimization problem in an augmented reality environment. The Apple Vision Pro is a standalone device equipped with an Apple R1 chip for real-time sensor processing and an M2 chip for general computation. The latter features an 8-core CPU, a 10-core GPU, a 16-core neural engine, and 16 GB of unified memory. This hardware configuration enables topology optimization to be performed directly on the headset, without the need for external computing resources, and supports *real-time* interactive responsiveness during the optimization process. While we do not perform high-resolution, full-scale optimization interactively, low-resolution designs can be generated almost instantaneously. The implemented TO formulation follows that of the educational paper by Ferrari and Sigmund (2020)—in fact, we translated their code into the Swift programming language. Therefore, we will not delve into the topology optimization formulation but rather focus mostly on the interactive aspects of the application.

Apple Vision Pro uses advanced sensors and cameras to track hand and eye movements in real-time. Therefore, we leverage spatial gesture recognition built into visionOS for all interactions within the app. A quick thumb-to-index finger pinch is known as a *tap* gesture, and it is used to select or activate an item, similar to a mouse left click. A *pinch-and-drag* gesture is a sustained thumb-to-index finger pinch that is used to grab and move objects or user interface (UI) elements by dragging them through space, akin to a mouse click-and-drag. For both gestures alike, the object to which the gesture is applied is whatever object the user is looking at.

The ARCADE application itself has two main components; it consists of a 2-D window with tabs in a navigation menu, where all aspects of the structural compliance problem can be defined, as well as a 3-D rendering of the domain for each corresponding tab. In this section, we describe each of the workflow steps, in an order that mirrors the tabs in the app.

2.1 Defining the domain

To define the domain we tap the "Computational domain" menu. The domain, rendered in 3-D as a colored box, appears *floating* in front of the user (see Fig. 1).

Users can manipulate the dimensions, position, and rotation of the design domain through the previously described *pinch-and-drag* gestures. When simply looking at one of the domain boundaries shown by the 3-D rendering, users can intuitively pull or push the boundaries inward or outward when pinching. Alternatively, the sliders in the 2-D window can be dragged in a similar fashion. Additionally, both the position and orientation of the 3-D rendering in space can be manipulated by performing such a gesture while looking at one of the arrows that define the origin of the reference Cartesian system (in Fig. 1, these can be seen behind the 3-D rendered shape).

2.2 Defining boundary conditions

Tapping on "Boundary conditions" (see Fig. 2) allows us to define clamped regions of the boundary (i.e., regions with prescribed zero displacement) and boundary tractions. The rendering of the computational design domain changes to a volumetric representation of faces, edges, and corners, each of which can have a different boundary condition. The selection is hierarchical, so prescribing boundary conditions to a surface also selects its edges and vertices. Likewise, prescribing boundary conditions to an edge also selects its vertices. Noteworthy, since these two types of boundary conditions are mutually exclusive, only one kind will be active for each geometric entity. Geometric entities without an assigned boundary condition represent regions with zero traction.

The app provides two buttons with default boundary conditions, one for a cantilever beam problem (see Fig. 2b) and

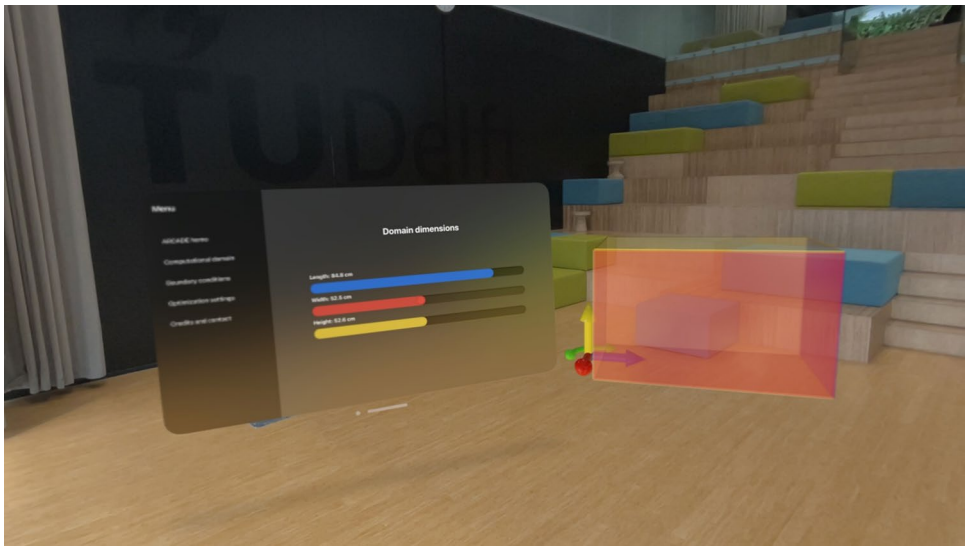

Fig. 1 The definition of the design domain takes place in the "Computational domain" tab. The domain can be resized via the sliders in the window, or simply by pinching and dragging each of the three colored planes of the rendered domain. The three orthogonal arrows represent the Cartesian coordinate system, and pinch-and-drag gestures on the arrows move the domain in space. The domain can also be rotated around the vertical yellow axis by pinching and dragging the green circular arrow

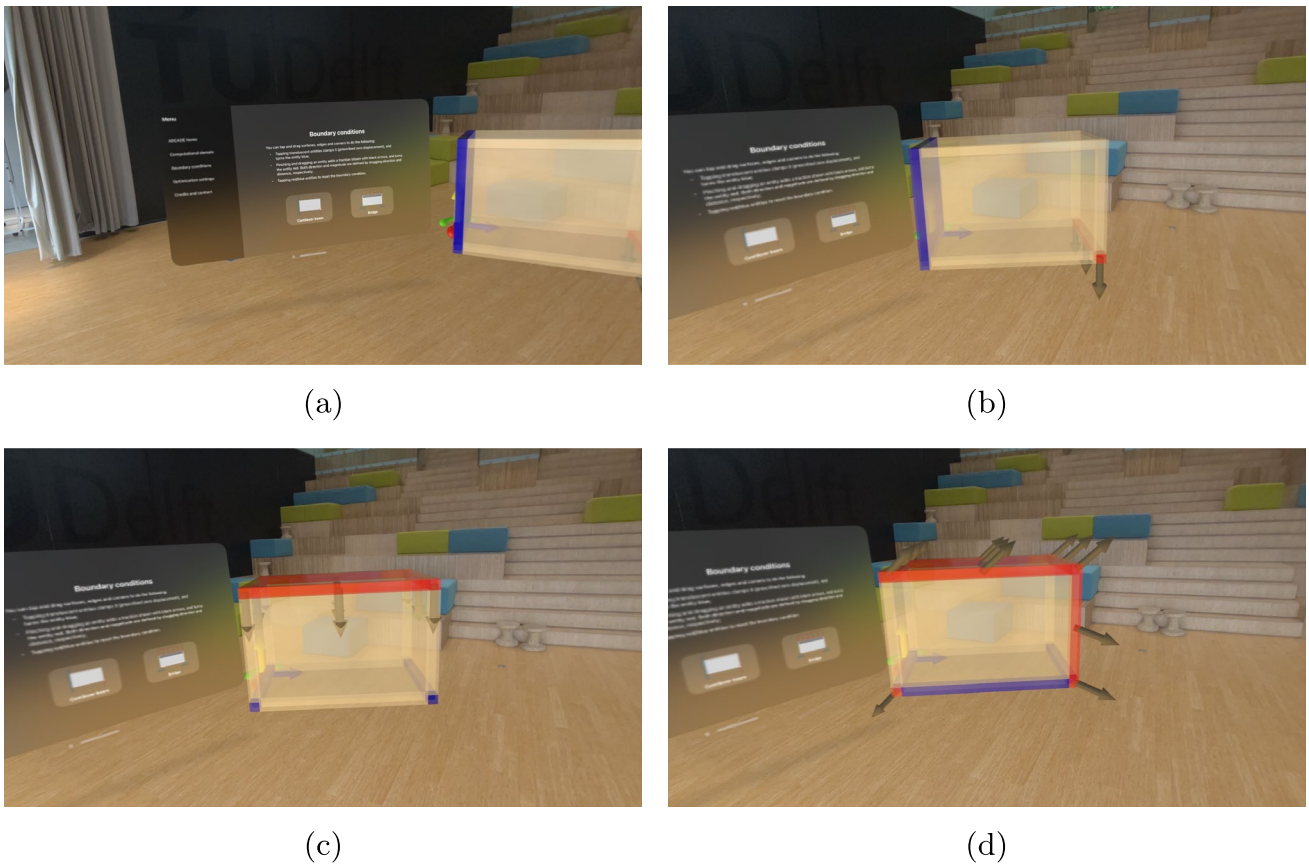

Fig. 2 Boundary conditions are defined in the "Boundary conditions" tab (**a**). Buttons with default boundary conditions are provided for a cantilever beam (**b**) and for a bridge (**c**). Any other boundary conditions can be defined (**d**). Clamped regions of the boundary, shown in blue, can be defined by simply tapping a vertex, edge, or an entire surface. Boundary tractions (colored in red with black arrows) are defined using pinch-and-drag gestures. Both traction direction and magnitude are defined by the dragging direction and dragging distance, respectively. Tapping a geometric entity with a prescribed boundary condition clears the boundary condition (the entity becomes translucent again)

one for a bridge problem (see Fig. 2c). However, the user can define an arbitrary set of boundary conditions. Simply tapping a geometric entity where a boundary traction is prescribed removes it, and tapping it again clamps it (the entity turns blue). The assigned clamped entity can also be removed simply by tapping it again. Pinching and dragging an entity defines a traction in the dragged direction and with magnitude proportional to the dragged distance. Figure 2d shows an example where different tractions are prescribed to the top surface, the right edges, and the bottom-left vertex. We have also provided the app the ability to show warnings when there are no enough supports—which would lead to a singular stiffness matrix during finite element analysis—and when there are no prescribed tractions.

2.3 Defining and running the optimization

After defining the computational domain dimensions and location in space, and the boundary conditions, we tap on the "Optimization settings" tab (see Fig. 3). Settings include the maximum number of iterations, the target volume fraction (as a percentage of the computational domain), and the finite element mesh size. These can be changed by pinching and dragging the sliders. Changing the element size updates the number of voxels in the finite element discretization. Two toggle buttons add the ability to hide void elements during the optimization and to use an iterative solver instead of a direct solver (both toggles are set to true by default).

When satisfied with the settings, the user taps the button to start the immersed topology optimization. Throughout the optimization process, changes in the design are rendered in real time and objective function values are shown in the "Output" section as a function of iterations. The app is interactive, so during the optimization, the user can go back to the "Boundary conditions" tab to make changes that take effect immediately. Behind the scenes, changing boundary tractions simply modifies the force vector—i.e., the right-hand side of the system of linear equations that is solved. From an implementational perspective, changing clamped regions takes more effort as it necessitates finding the new set of degrees of freedom whose displacement is prescribed to zero. Finally, feedback sounds have been added to indicate each optimization step (Rubik's cube sound) and for the completion of the optimization. Figure 4 illustrates the optimization of the cantilever beam problem.

At the end of the optimization, the final design is always rendered as a fully solid design, unloading elements with a density below 0.5 (see Fig. 5). Notice that the final design remains fixed in space so that we can walk around to inspect it from any angle.

2.4 Topology optimization formulation and implementation

As mentioned earlier, our topology optimization formulation is essentially a translated and modified version of the educational 3-D TO code by Ferrari and Sigmund (Ferrari and Sigmund 2020). As the original code relied on built-in MATLAB functions that are not available in Swift (such as `imfilter` for the filtering procedure), similar functionalities were implemented using Apple's framework. In the case of `imfilter`—used in the original MATLAB code to perform a 3-D filtering operation on the vector of design densities—we used Apple's `Accelerate` framework to perform a similar filtering procedure. In our implementation, we perform 3-D filtering as a sequence of 2-D convolutions. We note that, while we opted for zero-padding at boundaries, the MATLAB `imfilter` function also supports other types of padding. Since Swift has a lower level of abstraction, the translation significantly increased the number of coding

Fig. 3 Settings that define the optimization are set in the "Optimization settings" tab. These include the maximum allowed number of iterations, the target volume fraction of material within the computational design domain, and the finite element mesh size used for the discretization. The toggles allow the removal of voids during optimization and to use an iterative solver (both set to true by default)

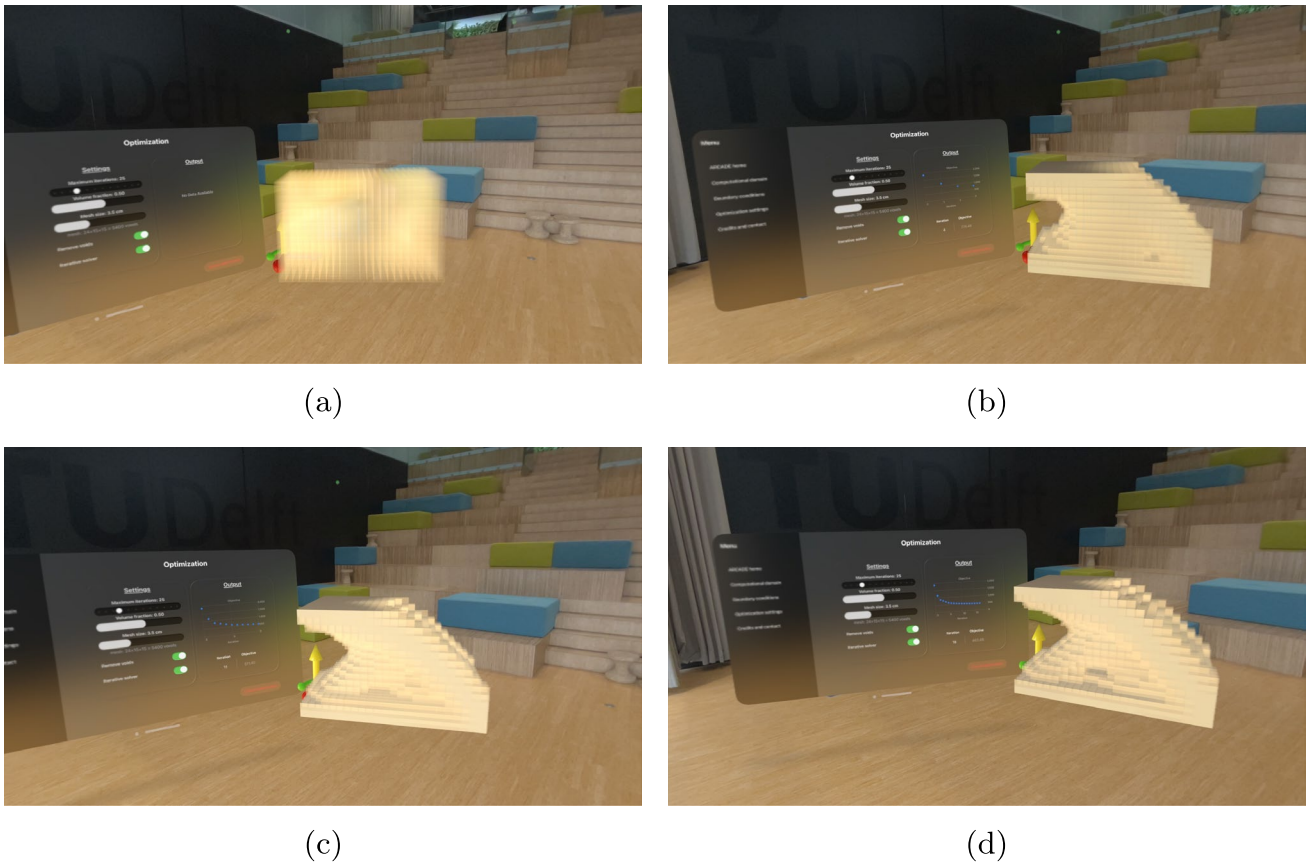

Fig. 4 During the optimization process, which starts by pressing the `Start optimization` button, changes to the design are visualized in real time. Objective function values as a function of iterations are displayed in the floating 2-D menu. The app is interactive so that changes to boundary conditions and other parameters can take place during the design process, steering the optimization toward a new local minimum

lines beyond the reference 125 lines. Additionally, a different node numbering convention was adopted, which greatly simplified the identification of unique faces, edges, and vertices, and the subsequent definition of boundary conditions. The source code of our Swift implementation is given in the Appendix.

3 Swift implementation details

The goal of this section is to provide insight into ARCADE's application structure. The operating system of the Apple Vision Pro is visionOS (Apple 2024b), so the technical terms used in this chapter often refer to concepts and components specific to its application developer environment.

3.1 User interface

The floating menu that appears when opening the application is generated using a `Window` from the `SwiftUI` framework. The window contains a `NavigationSplitView` that defines the different tabs of the menu. Switching between tabs loads new content, and asynchronously instructs the `DomainManager` (discussed below) to load or unload its 3-D content.

3.2 Rendering 3-D content

In visionOS, unbounded content in a user's surroundings—such as the design domain in our application—is managed within an `ImmersiveSpace` (Apple Inc. 2024a). This space acts as an invisible coordinate system that allows for the dynamic addition or removal of 3-D content. Upon launching the application, an `ImmersiveSpace` is created with its local coordinate system centered on the user's initial position. The `ImmersiveSpace` remains active throughout the session, allowing the user to move freely while ensuring that rendered 3-D content remains fixed in its designated position within the space.

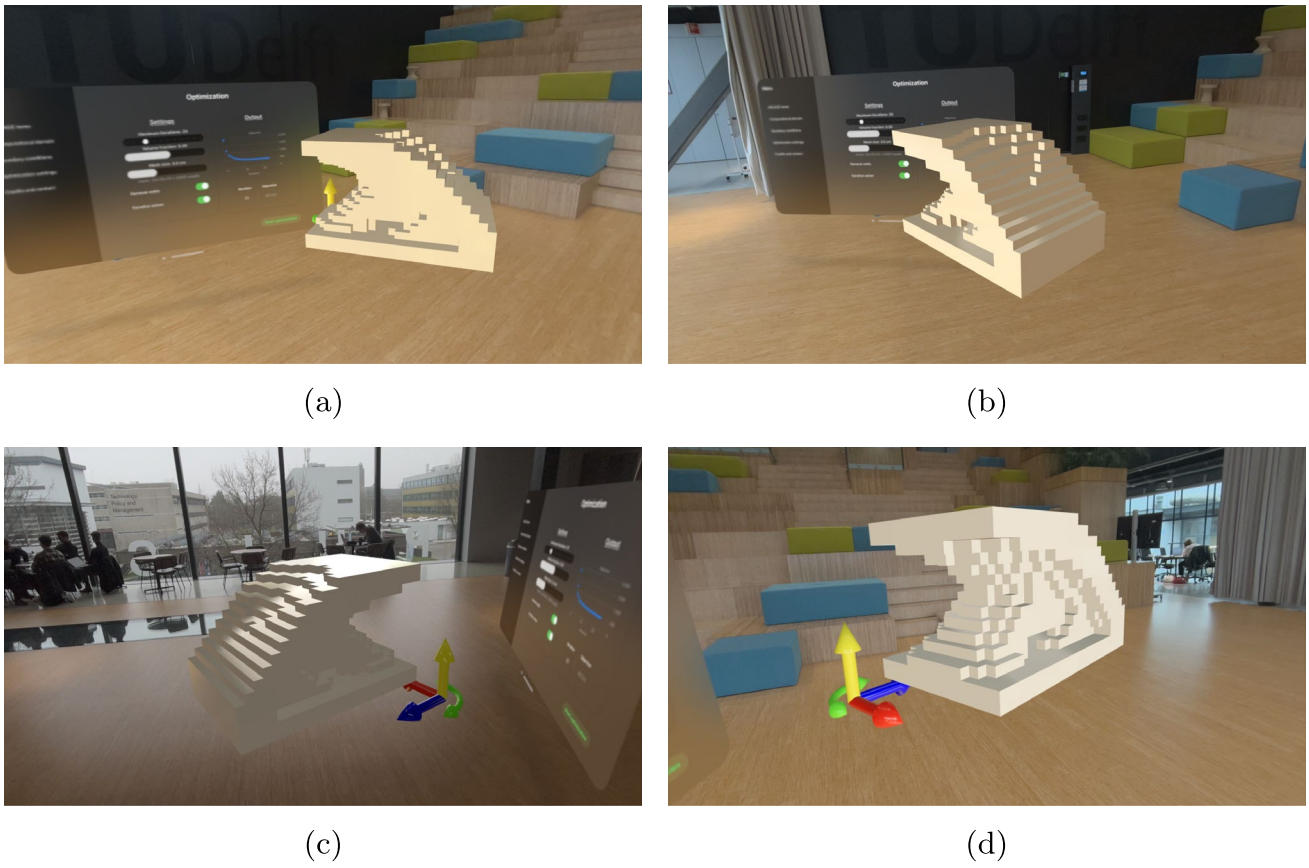

Fig. 5 The final result of an optimization procedure fully renders all elements with a density above a predefined threshold. The design stays still in space so it can be inspected from any angle

In order to load and unload 3-D content, an empty `Entity` from the `RealityKit` framework is added to the `ImmersiveSpace` during its initialization. During the flow of the application, all 3-D content is added or removed by making it a child of this `Entity`.

The logic behind loading and unloading different domain renderings is taken care of by the `DomainManager` class. The 3-D content itself is produced by three other classes, namely the `ResizableDomain` (for reshaping the domain in the "Computational domain" tab), `BCDomain` (for setting boundary conditions in the "Boundary conditions" tab), and `MeshDomain` (for rendering the finite element mesh during optimization). Each of these classes keep the state (properties in Swift) related to the content that they render. For example, the `ResizableDomain` object stores the actual dimensions of the domain, and the `MeshDomain` the finite element mesh size. Furthermore, the gestures that allow the user to interact with the 3-D content are also defined in these classes.

3.3 Running the optimization

Everything related to the finite element analysis and optimization is managed by the `SimulationManager` class. When the user starts the simulation from the 2-D menu, this class starts running the iterative topology optimization code on a separate thread. This allows the user to keep using the application while the optimization is in progress. A very useful property of the `SwiftUI` framework is that objects can become `Observable`, which enables a *reactive programming model* where UI components automatically update when the underlying data of `Observable` objects change. For example, as the topology optimization progresses, the density vector variable stored by the `SimulationManager` changes. This prompts the `DomainManager.MeshDomain` to update the 3-D rendering of the mesh. Furthermore, changes made in the optimization tab of the 2-D menu directly change the stored variables of `SimulationManager`. As a result, any adjustments to

the volume constraint or iteration limit take effect instantly during the ongoing optimization.

3.4 Performance

Regarding performance, relying solely on `RealityKit` introduced significant limitations in scalability. In the first implementation of `ARCADE`, voxel visualization was handled exclusively using individual `ModelEntity` instances per voxel. While this approach is simple to implement and tightly integrated with `RealityKit`, it incurs substantial CPU-side overhead due to memory allocation, scene graph management, and entity instantiation. This overhead prevents real-time performance when visualizing beyond approximately 10,000 voxels. This can be visualized in Fig. 6, where for the first set of green curves (labeled `ModelEntity`) the visualization stage dominates the total runtime, in contrast to the relatively fast topology optimization computations (the figure shows the time for 10 iterations of the iterative loop in the appendix). Furthermore, these performance measurements—captured via `os_signpost` instrumentation while running the code on the Vision Pro device—reflect only CPU-side execution time and do not account for additional delays introduced by `RealityKit`'s internal scene synchronization and GPU pipeline, leading to even greater perceived latency. As of the current versions of visionOS and `RealityKit`, it is not possible to render per-voxel opacity in a single `ModelEntity` using Apple's built-in framework features.

To enable more efficient mesh rendering, we implemented another mesh visualization approach in Swift using `RealityKit`, this time encapsulated in a `VoxelMesh` class. This approach constructs a custom mesh from a list of voxels by replicating cube geometries and merging their vertex, normal, and index data into a single `MeshResource` generated from a `MeshDescriptor`. The use of the `MeshDescriptor` API facilitates efficient GPU uploading and reuse. Because per-voxel opacity cannot be updated dynamically, in this second implementation we simply recreate the mesh after each optimization iteration, rendering only the subset of voxels whose density exceeds a specified threshold. The system relies on a single `ModelEntity` and integrates seamlessly with `RealityKit`'s entity-component architecture. It is capable of scaling well beyond 100,000 voxels while supporting real-time updates in the immersive environment of Apple Vision Pro. As illustrated in Fig. 6, the second set of orange curves (labeled `VoxelMesh`) demonstrates the significantly improved performance of this approach (see Fig. 7 for an optimization with a voxel size of 1 cm). In this configuration, the finite element analysis dominates the total runtime.

3.5 App availability

We have made `ARCADE` freely available on the visionOS App Store. As the writing of this article, **ARCADE** can run in visionOS versions 2.0-2.2 and visionOS 26.

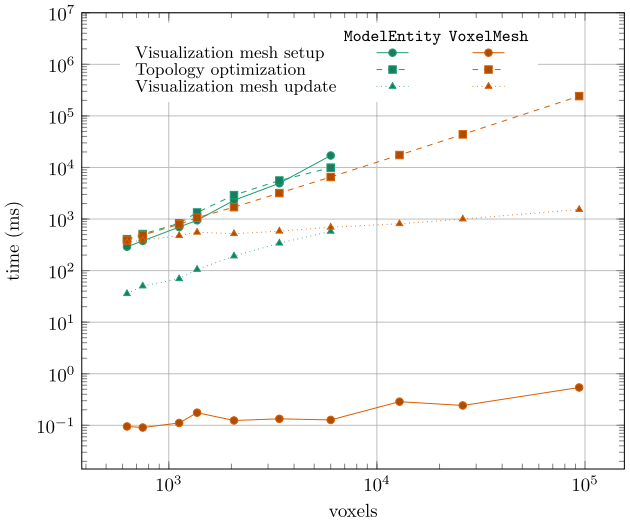

Fig. 6 Performance graph, showing current limitations with visualization in `RealityKit`. The curves labeled "Topology optimization" involve 10 iterations of the iterative loop (see appendix). The measurements were obtained using Apple Instruments while running the code on the Apple Vision Pro device

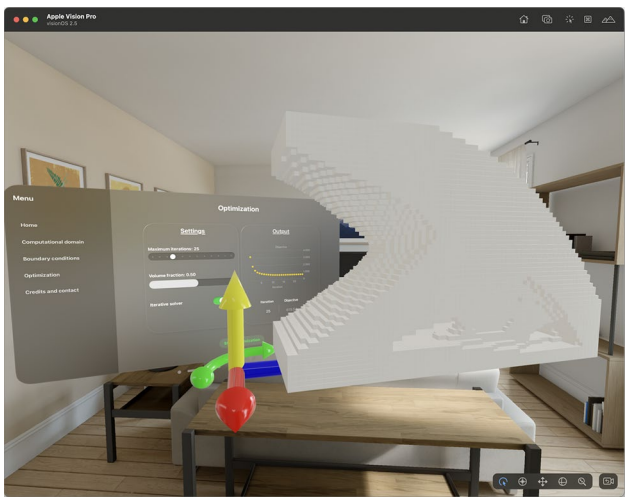

Fig. 7 Optimized design with a mesh size of 1 cm obtained in Apple Simulator after 25 iterations

3.6 Code availability

We have included the code for topology optimization in `Swift` in the appendix, which is essentially a translated version of the 125-line MATLAB code by Ferrari and Sigmund (2020).

4 Discussion

Immersed topology optimization in an augmented reality environment has the potential to shift design paradigms. By incorporating subjective human factors early in the design process, visualizing the design in its intended context, and enabling real-time adjustments to the optimization, this technology has the potential to accelerate the design process toward a final prototype. `ARCADE` represents our endeavor to make a contribution in that direction.

By allowing users to use hand gestures to set up the domain, prescribe boundary conditions, and solve a topology optimization problem, `ARCADE` reduces the learning curve associated with traditional modeling tools. This feature may in turn make topology optimization more accessible to users that do not particularly have a strong background in mechanics, e.g., interior designers, potentially enabling them to focus on subjective aspects such as esthetics.

Many improvements to `ARCADE` can be made and are left for future work. While in our first implementation users can interact with the optimization process, making changes to boundary conditions and other parameters in real time, we plan to make other aspects of the design process interactive. For instance, we plan to change the size of the domain in real time, which would change the entire finite element mesh and therefore the corresponding structure of the linear system of equations that is solved behind the scenes.

Further advancements can also be made to the implementation of the topology optimization methodology. While in this iteration of the app we wrote the code exclusively in the Swift programming language, where the formulation implementation is simply a translation of the vanilla MATLAB code of Ferrari and Sigmund (2020) (see the appendix), future versions could use other solvers and/or leverage external libraries written in other languages. We already consider to use a preconditioned conjugate gradient iterative approach, which has been shown to reduce the computational cost related to solving equilibrium by an order of magnitude (Amir and Sigmund 2011). A logical next step would be to setup *non-design* void and/or solid regions, which may require additional custom-defined gestures.

Even though this first iteration focuses solely on AR, we see strong potential to port the framework to VR environments, possibly outside Apple's ecosystem. However, since the `ARCADE` app was written entirely in Swift and relies heavily on Apple-specific frameworks such as `RealityKit` and `ARKit`, porting it to other platforms would likely require a complete rewrite using a different set of tools and possibly a different language. We note, however, that while Swift was originally developed by Apple, it was open-sourced starting with version 2.2 under the Apache License 2.0, which may help facilitate porting efforts.

Further improvements regarding visualization can also be made. In the future, we plan to develop a custom Metal[1]-based rendering pipeline with GPU instancing and per-voxel opacity control, enabling real-time visualization of extremely fine meshes with more than a million voxels. However, as of visionOS 26, Apple does not yet support user-defined Metal shaders within `CustomMaterial`, which limits the integration of fully custom rendering logic in `RealityKit`'s material system. Once such functionality becomes available, we plan to implement per-instance data buffers and GPU-based opacity filtering via Metal shaders, enabling high-performance, visually rich rendering within immersive environments on Apple Vision Pro.

All in all, the enhanced accessibility, expedited design process, and improved decision-making capabilities that arise from a seamless user interaction within an immersive design environment hold significant potential. `ARCADE` represents our initial step in that direction. It is our hope that future studies will shed light on the extent to which our app reduces the learning curve, enhances user experience, and improves design efficiency when compared to conventional topology optimization tools. It is also our hope that `ARCADE` serves as a source of inspiration for further developments. For instance, the augmented reality setting in which the optimization takes place can elevate existing concepts that integrate human knowledge into optimization processes to the next level. We hope that this innovative design paradigm will be embraced by the optimization community, facilitating the transition of our current research from an educational app to a valuable tool in the design toolkit of both researchers and industry professionals alike.

[1] Metal is Apple's low-level, high-performance graphics and compute framework. It allows developers to interface directly with the GPU on Apple devices for rendering graphics and performing data-parallel computations.

Appendix: Topology optimization code in Swift

```
/// Starts the optimization. Functions setupFilter and setupFEA must be executed first.
func startOptimizationLoop() throws {

    // Retrieve data from the SimulationManager class

    var loop = self.iteration

    // unwrap data (they would be 'nil' if not initialized)
    let h =  self.h!
    let Hs = self.Hs!
    let XYZ = self.dimensionsFEA!
    let ijKf32 = self.indicesKentries32!
    let dV = self.dV!
    let cMat = self.cMat!
    var x = self.x!
    let totalDofs = self.totalDofs!
    let act = self.act!
    let Ke0 = self.Ke0!
    let Ke0Array = self.Ke0Array!
    let totalElements = self.totalElements!

    // retrieve data from unwrapped variables
    let nelx = XYZ.0
    let nely = XYZ.1
    let nelz = XYZ.2
    let dHs = Hs

    // retrieve constants form SimulationManager (no unwrapping needed)
    let betaCnt = self.betaCnt
    let penalCnt = self.penalCnt
    let move = self.move
    var varPenal = self.penal
    var varBeta = self.beta

    // set up some variables
    var ch = 1.0
    var dsK: [Double] = Array(repeating: 0, count: self.totalElements!)

    // Initialize arrays for physical density, 'old' density, and global displacement
    let xOld: [Double] = Array(repeating: 0, count: act.count)

    print("- Starting optimization loop")
    repeat {

        let id = SignpostUtils.begin(name: "Topology single iteration")

        // Increase loop counter
        loop += 1

        // unwrap within the loop to allow for interactive changes to Dirichlet boundary conditions
        let zeros = self.zeros!
        let nonzeros = self.nonzeros!


        // 1) COMPUTE PHYSICAL DENSITY FIELD

        // Reshape 1D density vector to 3D matrix, and apply the filtering procedure.
        // This is done by doing a 3D convolution with the filter kernel 'h'.
        let xTilde3D : [[[Double]]] =
        convolve3D(
            signal: reshapeTo3DArray(x, toShape: (nelz, nely, nelx))!,
            kernel: h
        )

        // Explicitly divide result by Hs
        let xTilde3D_div = div3D(A: xTilde3D, B: Hs)

        // Convert 3D -> 1D using flattening operation
```

```
let xTilde : [Double] = flatten3D(xTilde3D_div)

// Reshape density vector to only contain active elements. This is done by extracting the values of xTilde based
// on the act vector (which contains indices of active elems).
let xPhys = act.map { xTilde[$0] }

// Obtain difference between xPhys new and xOld vectors
let diff: [Double] = zip(xPhys, xOld).map { $0 - $1 }

// Compute the error
ch = euclideanNorm(diff) / Double(totalElements);


// 2) SET UP & SOLVE THE EQUILIBRIUM EQUATIONS

// result of SIMP(rho_filtered) stored in vector
let sK0 : [Double] = xPhys.map {
    self.Emin + pow($0, penal) * (self.E0 - self.Emin)
}

var sKf =  sK0.flatMap{ v in
    Ke0Array.map {$0 * v}
}

// Derivative of penalization, with respect to physical density.
for i in act {
    dsK[i] = -penal * ( E0 - Emin ) * pow(xPhys[i], penal-1)
}

for i in zeros {
    sKf[i] = 0
}
for i in nonzeros {
    sKf[i] = 1.234
}

/// Assemble Kf
let Kf: SparseMatrix_Double =
SparseConvertFromCoordinate(Int32(totalDofs), Int32(totalDofs), sKf.count, 1, // row/col count, block count/size
                            SparseAttributes_t(),               // attributes
                            ijKf32[0], ijKf32[1], sKf)          // row [i's], column [j's], data

defer {
    SparseCleanup(Kf)
}

var U =  Array(repeating: 0.0, count: totalDofs)
var F = self.F!

if self.iterativeSoler {

    let preconditioner = SparseCreatePreconditioner(SparsePreconditionerDiagScaling, Kf)

    defer {
        SparseCleanup(preconditioner)
    }

    func reportStatus(msg: UnsafePointer<CChar>) {
        print("CG: ", String(cString: msg), terminator: "")
    }

    var throwException: Bool = false

    F.withUnsafeMutableBufferPointer { bPtr in
        U.withUnsafeMutableBufferPointer { xPtr in
            let b = DenseVector_Double(count: Int32(totalDofs), data: bPtr.baseAddress!)
            let x = DenseVector_Double(count: Int32(totalDofs),data: xPtr.baseAddress!)

            let status = SparseSolve(
                SparseConjugateGradient(SparseCGOptions(reportError: reportStatus, maxIterations: 10000,
```

```
                                                                atol: 1e-2, rtol: 1e-2, reportStatus: reportStatus,)),
                Kf, b, x, preconditioner)

            if status != SparseIterativeConverged {
                throwException = true
                print("Failed to converge. Returned with error \(status).")
            }
        }
    }
    if throwException {
        throw FailedToConvergeError(message: "Failed to converge.")
    }

} else {

    /// Factorize K
    let factorization_Kf = SparseFactor(SparseFactorizationQR, Kf)
    defer {
        SparseCleanup(factorization_Kf)
    }

    U = Array(self.F!)

    // solve the linear system. It starts with F-values in U,
    // and ends with displacement values in U .
    U.withUnsafeMutableBufferPointer { xPtr in
        let x = DenseVector_Double(count: Int32(totalDofs),
                                   data: xPtr.baseAddress!)
        SparseSolve(factorization_Kf, x)
    }
}


/// Compute compliance & volume
let C = zip(U, self.F!).map { $0 * $1 }.reduce(0, +)

let V = xPhys.reduce(0, +)/Double(totalElements)

/// Print output
print(
    "\(String(format: "%3d", loop)): " +
    "Compliance: \(String(format: "%10.3e", C)) " +
    "Vol: \(String(format: "%.2g", V))"
)


// 3) COMPUTE SENSITIVITIES

/// Compliance derivative
var UU = [[Double]](repeating: [], count: cMat.count)
for (i, row) in cMat.enumerated() {
    UU[i] = row.map { index in
        U[index]
    }
}
UU = UU.transpose()
let UUKe0 = multiply2DMatrices(UU, Ke0)!  // uu 750 x 24

/// Perform element-wise multiplication and sum
let t1 = zip(UUKe0, UU).map { zip($0, $1).map(*) }
                       .map { $0.reduce(0, +) }
let dC = zip(dsK, t1).map { $0 * $1 }

// reshape compliance sensitivities into a 3D matrix, and perform convolution with h
let conv_dC = convolve3D(signal: div3D(A: reshapeTo3DArray(dC, toShape: (nelz, nely, nelx))!, B: dHs), kernel: h)

let dCC = flatten3D(conv_dC)

// the same procedure is done for the volume constraint (filter & divide by dHs)
let dVV =
```

```
        flatten3D(convolve3D(signal: div3D(A: reshapeTo3DArray(dV, toShape: (nelz, nely, nelx))!, B:dHs), kernel: h))


        // 4) UPDATE DESIGN VARIABLES AND APPLY CONTINUATION

        let xT: [Double] = act.map { x[$0] }        // Active Elements
        let xU: [Double] = xT.map { $0 + move }     // Upper bound
        let xL: [Double] = xT.map { $0 - move }     // Lower bound

        // constant part in resizing rule
        let dCact: [Double] = act.map { dCC[$0] }   // Active Compliance Sensitivities
        let dVact: [Double] = act.map { dVV[$0] }   // Active Volume Constraint Sensitivities
        let dCbydV: [Double] = zip(dCact, dVact).map { sqrt(-$0/$1) }

        let ocP: [Double] = zip(xT, dCbydV).map { $0 * $1  }

        // initial estimate for LM (top3D125.m line 109)
        var l = [0, (ocP.reduce(0, +) / Double(ocP.count)) / volfrac ]

        // Optimality Criteria rule
        while (l[1] - l[0])/(l[1] + l[0]) > 1e-4 {
            let lmid = 0.5 * (l[0] + l[1])
            let comparison : [Double] =
            compareArrays(larger: true,
                          A: Array(repeating: 0.0, count: ocP.count),
                          B: compareArrays(larger: true,
                                           A: xL,
                                           B: compareArrays(larger: false,
                                                            A: Array(repeating: 1.0, count: ocP.count),
                                                            B: compareArrays(larger: false,
                                                                             A: ocP.map {$0 / lmid},
                                                                             B: xU)!)!)!)!
            for el in act {
                x[el] =  comparison[el]
            }
            if x.reduce(0, +) / Double(x.count) > volfrac {
                l[0] = lmid
            } else {
                l[1] = lmid
            }
        }

        // Apply continuation on parameters
        varPenal = applyContinuation(val: varPenal, valCnt: penalCnt, loop: loop)
        varBeta  = applyContinuation(val: varBeta, valCnt: betaCnt, loop: loop)
        self.penal = varPenal
        self.beta = varBeta

        SignpostUtils.end(name: "Topology single iteration", signpostID: id)

        // The below {} code is added to the FEA to let it communicate with the app
        DispatchQueue.main.async {

            // change is made to published variable. for some reason this must happen on main thread...
            // hence the DispatchQueue.main.async above!
            self.iteration = loop
            self.dvec = xPhys
            self.iteration = loop
            self.objectives.append(C)
        }

    } while ch > self.tolerance && loop < self.iterationLimit && self.appState == .simulating

    if ch < self.tolerance {
       print("Objective below tolerance!")
    }

    if loop >= self.iterationLimit {
       print("Exceeded iterl limit!")
    }

    // prepare for next simulation
    self.appState = .idle
}
```

Author contributions Conceptualization: A. M. Aragón; Code development: A. M. Aragón and H. J. Algra; Writing - original draft preparation: H. J. Algra; Writing - review and editing: A. M. Aragón and H. J. Algra; Funding acquisition: A. M. Aragón; Supervision: A. M. Aragón.

Funding The authors acknowledge the High Tech Visibility Boost grant by the faculty of Mechanical Engineering at TU Delft.

Data availability The core optimization routine used in the application is provided in the appendix of this paper. The app itself was developed

using Apple's visionOS SDK and integrates this routine into an interactive spatial computing environment for topology optimization. The application has been made freely available through the visionOS App Store. Additional information is available from the corresponding author upon request.

Declarations

Conflict of interest On behalf of all authors, the corresponding author states that there is no conflict of interest.

Replication of results All results in this paper are generated by `ARCADE`. To replicate the results, readers are encouraged to download the app from the Vision App Store.

References

Aage N, Nobel-Jørgensen M, Andreasen CS et al (2013) Interactive topology optimization on hand-held devices. Struct Multidisc Optim 47(1):1–6. https://doi.org/10.1007/s00158-012-0827-z

Amir O, Sigmund O (2011) On reducing computational effort in topology optimization: how far can we go? Struct Multidisc Optim 44(1):25–29. https://doi.org/10.1007/s00158-010-0586-7

Apple Inc. (2024a) Immersive spaces—SwiftUI documentation. https://developer.apple.com/documentation/swiftui/immersive-spaces. Accessed 26 Nov 2024

Apple Inc. (2024b) visionOS—apple developer. https://developer.apple.com/visionos/. Accessed 7 Nov 2024

Bendsøe M, Sigmund O (2004) Topology optimization theory, methods and applications, 1st edn. Springer. https://doi.org/10.1007/978-3-662-05086-6

Delissen A, Boots E, Laro D et al (2022) Realization and assessment of metal additive manufacturing and topology optimization for high-precision motion systems. Addit Manuf 58:103012. https://doi.org/10.1016/j.addma.2022.103012

Ferrari F, Sigmund O (2020) A new generation 99 line MATLAB code for compliance topology optimization and its extension to 3D. Struct Multidisc Optim 62(72):2211–2228. https://doi.org/10.1007/s00158-017-1884-0

Guibert A, Aaron K, Daimaru T et al (2024) Multiphysics level-set topology optimization of a rover chassis for extreme cold environments. AIAA SCITECH Forum. https://doi.org/10.2514/6.2024-2233

Ha DQ, Carstensen J (2023) Human-informed topology optimization: interactive application of feature size controls. Struct Multidisc Optim 66:59. https://doi.org/10.1007/s00158-023-03512-0

Ha D, Carstensen J (2024) Automatic hyperparameter tuning of topology optimization algorithms using surrogate optimization. Struct Multidisc Optim 67:157. https://doi.org/10.1007/s00158-024-03850-7

Høj D, Wang F, Gao W et al (2021) Ultra-coherent nanomechanical resonators based on inverse design. Nat Commun 12:5766. https://doi.org/10.1038/s41467-021-26102-4

Koper D, Leung C, Smeets L et al (2020) Topology optimization of a mandibular reconstruction plate and biomechanical validation. J Mech Behav Biomed Mater. https://doi.org/10.1016/j.jmbbm.2020.104157

Kudela J, Matousek R (2022) Recent advances and applications of surrogate models for finite element method computations: a review. Soft Comput 26(24):13709–13733. https://doi.org/10.1007/s00500-022-07362-8

Langelaar M (2016) Topology optimization of 3D self-supporting structures for additive manufacturing. Addit Manuf 12:60–70. https://doi.org/10.1016/j.addma.2016.06.010

LaViola JJ (2000) A discussion of cybersickness in virtual environments. SIGCHI Bull 32(1):47–56. https://doi.org/10.1145/333329.333344

Li Z, Lee TU, Xie YM (2023) Interactive structural topology optimization with subjective scoring and drawing systems. Comput Aided Des 160:103532. https://doi.org/10.1016/j.cad.2023.103532

Li Z, Lee TU, Xie YM (2025) Interactive 3D structural design in virtual reality using preference-based topology optimization. Comput Aided Des 180:103826. https://doi.org/10.1016/j.cad.2024.103826

Loos S, Svd W, Graaf N et al (2022) Towards intentional aesthetics within topology optimization by applying the principle of unity-in-variety. Struct Multidisc Optim 65(7):185. https://doi.org/10.1007/s00158-022-03288-9

Nguyen TT, Bærentzen JA, Sigmund O et al (2020) Efficient hybrid topology and shape optimization combining implicit and explicit design representations. Struct Multidisc Optim 62(3):1061–1069. https://doi.org/10.1007/s00158-020-02658-5

Nobel-Jørgensen M, Aage N, Nyman Christiansen A et al (2015) 3D interactive topology optimization on hand-held devices. Struct Multidisc Optim 51(6):1385–1391. https://doi.org/10.1007/s00158-014-1214-8

Nobel-Jørgensen M, Malmgren-Hansen D, Bærentzen JA et al (2016) Improving topology optimization intuition through games. Struct Multidisc Optim 54(4):775–781. https://doi.org/10.1007/s00158-016-1443-0

Schiffer G, Schmidt MP, Pedersen CBW et al (2024) Interactive infill topology optimisation guided by user drawn patterns. Virtual Phys Prototyp 19(1):e2361864. https://doi.org/10.1080/17452759.2024.2361864

Souchet AD, Lourdeaux D, Pagani A et al (2023) A narrative review of immersive virtual reality's ergonomics and risks at the workplace: cybersickness, visual fatigue, muscular fatigue, acute stress, and mental overload. Virtual Reality 27(1):19–50. https://doi.org/10.1007/s10055-022-00672-0

Tseng WJ (2023) Understanding physical breakdowns in virtual reality. In: Extended abstracts of the 2023 CHI conference on human factors in computing systems. Association for Computing Machinery, New York, NY, USA, CHI EA '23. https://doi.org/10.1145/3544549.3577064

Vantyghem G, Corte WD, Shakour E et al (2020) 3D printing of a post-tensioned concrete girder designed by topology optimization. Autom Constr 112:103084. https://doi.org/10.1016/j.autcon.2020.103084

Yang R, Chahande A (1995) Automotive applications of topology optimization. Struct Multidisc Optim 9:245–259. https://doi.org/10.1007/BF01743977

Yano M, Huang T, Zahr MJ (2021) A globally convergent method to accelerate topology optimization using on-the-fly model reduction. Comput Methods Appl Mech Eng 375:113635. https://doi.org/10.1016/j.cma.2020.113635

Zhang S, Norato J, Gain A et al (2016) A geometry projection method for the topology optimization of plate structures. Struct Multidisc Optim 54:1173–1190. https://doi.org/10.1007/s00158-016-1466-6