Rethinking Tolerance through Interactive and Complex Architectural Systems

Zehra Delerel[1], Funda Tan Bayram[2]

ORCID NO: 0009-0005-6383-0639 [1], 0000-0001-2345-6789[2]

[1] Gebze Technical University, Institute of Graduate Studies, Department of Architecture, Kocaeli, Türkiye
[2]Gebze Technical University, Faculty of Architecture, Department of Architecture, Kocaeli, Türkiye

This study redefines tolerance within computational design, shifting it from a margin of error toward an interactive, uncertainty-driven, and context-sensitive design strategy. Two custom Python scripts developed in Rhino 3D demonstrate how tolerance operates across dual-wall systems responsive to attractor points. Code 1 produces spatial contrast and topological tension through opposing wall behaviors, while Code 2 generates coherence and porosity through synchronized responses. The comparative analysis reveals that tolerance functions on multiple levels: uncertainty (randomized attractors and micro-variation), bounded variability (extrusion ranges between 15–60 units), behavioral contrast (Code 1), behavioral coherence (Code 2), permeability (aperture scaling), and semantic flexibility (4-bit threshold-based encoding). Scenario testing confirmed these distinctions: Code 1 generated greater variability and contrast across runs, while Code 2 exhibited more consistent and homogeneous fields. These findings resonate with McVicar's (2016) definition of tolerance as a "range of opportunity," showing that identical inputs can yield divergent spatial outcomes. The results demonstrate that tolerance is not a passive technical allowance but an active design agent embedded within coding logic. By transforming uncertainty into structured variability, tolerance enables architectural surfaces to evolve beyond optimized façades into dynamic, data-rich, and temporally adaptive systems. The study highlights the capacity of digital environments to act as design participants, where codes and their generative ranges interact with context to produce new spatial possibilities. Future research should extend this approach through broader software platforms, larger datasets, and physical prototyping, enabling tolerance to be examined at more comprehensive experimental and applied levels.

Keywords: Computational design, Complexity, Interactive architecture, Parametric surfaces, Tolerance.

Received: 30.06.2025
Accepted: 26.09.2025

Corresponding Author:
zehra.delerel2017@gtu.edu.tr

Delerel, Z. & Tan Bayram, F. (2025). Rethinking tolerance through interactive and complex architectural systems. *JCoDe: Journal of Computational Design, 6(2)*, 211-234. https://doi.org/10.53710/jcode.1728523

Interaktif ve Karmaşık Mimari Sistemler Üzerinden Toleransı Yeniden Düşünmek

Zehra Delerel[1], Funda Tan Bayram[2]

ORCID NO: 0009-0005-6383-0639 [1], 0000-0001-2345-6789[2]

[1] Gebze Teknik Üniversitesi, Lisansüstü Eğitim Enstitüsü, Mimarlık Anabilim Dalı, Kocaeli, Türkiye
[2]Gebze Teknik Üniversitesi, Mimarlık Fakültesi, Mimarlık Bölümü, Kocaeli, Türkiye

Bu çalışma, tolerans kavramını hesaplamalı tasarım bağlamında yeniden ele almakta; onu hata payı olmaktan çıkarıp etkileşim, belirsizlik ve bağlamsal duyarlılığa dayalı üretken bir stratejiye dönüştürmektedir. Rhino 3D'de geliştirilen iki özel Python betiği, çekim noktalarına yanıt veren ikili duvar sistemleri üzerinden toleransın farklı biçimlerde nasıl işlediğini ortaya koymuştur. Kod 1, yüzeyler arasında karşıtlık ve topolojik gerilim üretirken; Kod 2, senkronize davranışlarla daha bütünlüklü ve geçirgen alanlar oluşturmuştur. Karşılaştırmalı analiz, toleransın çok katmanlı işlediğini göstermektedir: belirsizlik (rastlantısal çekim noktaları, mikro-varyasyon), sınırlandırılmış değişkenlik (15–60 birimlik ekstrüzyon aralıkları), davranışsal karşıtlık (Kod 1), davranışsal uyum (Kod 2), geçirgenlik (açıklık ölçekleri) ve anlamsal esneklik (4-bitlik kodlama). Senaryo testleri bu katmanları doğrulamış; Kod 1 daha yüksek çeşitlilik ve kontrast, Kod 2 ise daha tutarlı ve homojen davranışlar üretmiştir. Böylece toleransın, McVicar'ın (2016) tanımladığı gibi, "fırsat aralığı" sunduğu ve aynı girdilerin farklı mekânsal çıktılar doğurabildiği gösterilmiştir. Sonuçlar, toleransın yalnızca teknik bir kısıt değil, kodlama süreçlerine gömülü üretken bir tasarım aktörü olduğunu ortaya koymaktadır. Bu bağlamda tolerans, mimari yüzeylerin optimize edilmiş cephelerden öteye geçerek veri ve bağlamla şekillenen, tepki verebilen ve zamansal olarak evrilen sistemlere dönüşmesini sağlamaktadır. Çalışma, dijital ortamın tasarımda aktif bir rol üstlenebileceğini ve toleransın gelecekte daha kapsamlı, deneysel ve uygulamalı düzeylerde mekânsal etkileşimlerle sınanabileceğini ortaya koymaktadır.

Teslim Tarihi: 30.06.2025
Kabul Tarihi: 26.09.2025

Sorumlu Yazar:
zehra.delerel2017@gtu.edu.tr

Delerel, Z. & Tan Bayram, F. (2025). Interaktif ve karmaşık mimari sistemler üzerinden toleransı yeniden düşünmek. *JCoDe: Journal of Computational Design, 6(2)*, 211-234. https://doi.org/10.53710/jcode.1728523

Anahtar Kelimeler: Hesaplamalı tasarım, Karmaşıklık, İnteraktif mimarlık, Parametrik yüzeyler, Tolerans.

1. INTRODUCTION

The concept of tolerance in architecture has historically been associated with engineering precision, fabrication errors, and dimensional deviations. However, with the advent of computational and parametric design thinking, tolerance carries the potential to become a generative strategy that embraces uncertainty, openness, variability, and systemic adaptability within design processes. As Banihashemi et al. (2025) argue, parametric design offers critical points in the formation of design knowledge by establishing generative mechanisms and enabling the production of variants.

In this expanded sense, tolerance facilitates a shift from deterministic solutions toward probability-based systems, transforming the role of the designer from that of an absolute controller into a mediator of parametric thresholds. As Fattahi Tabasi et al. (2024) note, design strategies grounded in tolerance and parametric thresholds allow parametric design to develop its own methods for producing novel solutions and alternatives. Parametric design, thus acknowledged as a new approach in architectural practice, provides a unique methodological framework for generating diverse design outcomes and alternatives.

This shift—abandoning the pursuit of absolute precision at the core of computational design and instead embracing tolerance—acknowledges material, environmental, and human variability as inseparable elements of both design and production. Applied in this way, tolerance leads to the design of spaces open to personalization and reinterpretation. Aesthetically, this move signifies a departure from the pursuit of impossible precision toward an aesthetic that embraces variation and deviation (Kolarevic, 2014).

Surfaces, as the fundamental components of spatial articulation, can also be reconsidered within this context. Building envelopes—particularly façades—that surround everyday environments are often designed either with an emphasis on aesthetics or performance. In both cases, contemporary design practice increasingly aims to achieve “the most optimal façade formation.” Current research even attempts to situate aesthetic preference within measurable frameworks through multi-objective optimization and expert systems, simultaneously

enhancing daylight performance and aesthetic perception in façade design (Yi, 2009).

This raises the following question: can a surface, beyond merely being "optimal," also enable interaction, personalization, and reinterpretation through a tolerance-oriented design approach? In this regard, the present study aims to reframe the architectural concept of tolerance through computational tools. Instead of seeking singular optimized solutions, the computational process developed here is structured to generate variations that are body-referenced, interactive, tolerance-adaptive, and complexity-based.

The research investigates how interactive surfaces can be designed as variable, contextual, and responsive systems using two custom Python scripts developed in the Rhino 3D environment. These scripts produce dual-wall parametric systems in which extrusion and aperture values respond to attractor points, thereby generating topological variability.

In this context, the study demonstrates how tolerance can operate not merely as a technical constraint but as a performative, interactive, and context-sensitive design principle in architecture.

2. THEORETICAL BACKGROUND: INTERACTIVE ARCHITECTURE, COMPLEXITY, AND TOLERANCE

Interactive architecture presents a new design paradigm that moves away from producing static environments toward creating systems responsive to parameters such as environment, time, and uncertainty. This approach establishes a framework in which flows themselves become design decisions, enabling multiple possibilities rather than singular solutions. While flow constitutes the primary aim of interaction design, stability remains a fundamental goal of architecture—two objectives that converge (McCullough, 2005). Similarly, Fox and Kemp (2009) define interactive architecture as a hybrid between embedded computation (intelligence) and its physical counterpart (kinetics), which adapts through human–environment interaction. Oosterhuis (2007) further emphasizes that interactive architecture is not merely passively responsive but forms a system of two-way communication that requires two active participants.

The design and production stages of interactive architecture, being dependent on parameters, can be associated with parametric design. As Oxman (2017) states, parametric design develops diversified, discoverable, and context-based strategies for generative and performative design tied to specific goals. When such strategies of variation intersect with interactivity, differentiations emerge. For Oxman (2017), differentiation in architecture can be understood as the local specialization of a repetitive formation. The purpose of specializing part of a regular system is to produce new formal, functional, performative, and structural features as well as material behaviors.

Flows move beyond fixed conditions and instead generate real-time scenarios, thereby opening new fields of design. These openings are closely tied to the concept of tolerance. According to McVicar (2016), tolerance in design refers to a margin of opportunity that enables the negotiation of constructional difficulties—opportunities that are immediate and unanticipated. Bates and Sergison (1999) similarly describe tolerance as a technological framework that allows components of a system to be used in unconventional ways.

Rather than operating within temporally fixed states, interactive environments produce emergent, unexpected, and complex conditions that, in turn, generate perceptible fields of experience. As Yücel and Ökten (2020) argue, “The most significant factor obstructing the body’s need for experience is the pursuit of comfort and ease.” By contrast, the unpredictable conditions produced through tolerance create a form of disorder that is inherently more practical. For De Certeau (2008), space is defined as a practiced place.

Everyday life is directly tied to time and is continuous. Tolerance is not only limited to the moment of construction but can also be reproduced in alternative scenarios emerging within ongoing flows. Each temporal state thus yields a new condition of tolerance. Jules Moloney (2009) asserts that rather than a fixed design, the outcome is the kinetic system itself, within which countless permutations unfold over time. The temporality, variability, and generative potential of interaction produce new forms of productive tolerance. Through the integration of such variables, the concept of articulation becomes interchangeable with tolerance itself.

Two practical forms of tolerance may arise. First, tolerance can be anticipated and accommodated through predesigned parameters controlled by the designer. Second, it may emerge unpredictably from the actions of bodies within everyday life. Both scenarios remain open to the unexpected: the first allows for generative deviations and new algorithmic formulations, while the second permits disappearance and spatial reconstitution depending on bodily engagement.

When tolerance manifests in unexpected moments, it may result in flawed outputs. While such outcomes are often perceived as undesirable or structural defects in contemporary practice, Venturi (2005) counters this view: “A structure without flaw has no virtue, for it is contrast that supports meaning.” Oppositions create tensioned spatial relationships between bodies and architecture, giving rise to new interactive scenarios of experience.

In everyday life, the triad of body–space–technology is continuously experienced, and this interaction can be inscribed into spatial geometries. In this context, Michael Fox (2009) identifies the growing driving force behind interactive architecture as stemming from the increasingly technology-mediated interactions between humans and the built environment. The built environment today does not merely host human actions it becomes reconfigurable through them.

Architecture’s adaptive capacity lies in rethinking the relationships between body and space. Understanding space not solely through geometric definitions, but also through the disproportionate, perceptual, and experiential relationships the body establishes with it, reveals the potential for surfaces to transform into a new form of skin one that is dynamic rather than passive. Park et al. (2011) regard such skins as interactive, kinetic surfaces integrating physical and content layers.

Digitally augmented surfaces that respond to stimuli such as light, sound, or motion are no longer simply façades or interior partitions; they become responsive, communicative, and transformative living textures. In this regard, Marcos Novak’s (2001) concept of transactive intelligence is crucial for conceptualizing the potential of such surfaces. Transformability and variability foster complex interactions. According to Bundy (2007), increasing environmental complexity necessitates the

development of computational methods informed by complexity theory—a concern shared across disciplines. The importance here lies in the everyday nature of complex systems and their theoretical relevance to architecture.

In interactive systems, actions and reactions define systemic relationships. These interactions function through positive and negative feedback loops. While actions are initiated by human bodies and experiences, reactions may be architectural—or vice versa. In such scenarios, tolerance becomes dynamic and continuously mutable. The system logic here aligns with Norbert Wiener's mathematical models of feedback, where positive loops amplify change and negative loops suppress it. As Jaskiewicz (2013) argues, the design process should not be understood as producing a fixed final product but as an intervention within an evolving complex system. In this way, the designer's role shifts toward establishing systems that can accommodate and respond to variation—revealing systemic authorship capacity.

Contemporary parametric modeling approaches predominantly focus on formal variation and geometric control; however, they lack a framework that addresses tolerance as a dynamic threshold—one that can generate both constraints and creative opportunities. Theoretically, parametric design, by applying both generative and logical approaches, replaces the manual development of design alternatives with a design logic (Tabadkani, Banihashemi et al., 2018). Generative algorithmic modeling draws from both relational and productive modeling. As Nasir and Kamal (2023) emphasize, the term "algorithm" in this context underlines that objects are generated through algorithms, and their subsequent outputs for later design stages are likewise produced algorithmically.

Moreover, the complex behaviors of interactive and responsive systems must be reconciled with the real-time capacities of computational environments. Current models rarely address how surfaces can negotiate environmental interaction and algorithmic variability simultaneously within a generative system logic. This study fills this gap by repositioning tolerance as a data-driven, performative, and interactive design parameter. In doing so, tolerance is rendered both computationally measurable and analyzable through custom algorithmic tools. This approach moves away from deterministic

control, instead proposing a design framework that encodes systemic uncertainty and adaptability as integral qualities of architecture.

3. METHODOLOGY: DEVELOPMENT OF THE PARAMETRIC SCRIPTS

This research adopts a computational design methodology (Step 1, Framework and Data Setup) grounded in the principles of parametric modeling, data-driven variation, and tolerance-based adaptation. The dual-wall system is selected as the design focus, with attractor-based spatial conditions defining the primary inputs. Within this framework, randomized attractors introduce tolerance to spatial uncertainty, while the normalization of distance (t) and its processing through a sigmoid function (k parameter) establish tolerance bands and smooth sensitivity ranges (Figure 1).

Building on this foundation, two custom Python scripts were developed within the Rhino 3D environment (Figure 1). Code 1 encodes a dual-wall extrusion logic with opposing responses (left vs. right), while Code 2 implements a dual-wall aperture transformation with symmetric behaviors. At this stage, opposing wall logic defines a behavioral tolerance field, min–max extrusion ranges (15–60) set generative tolerance zones, and micro-variation (±0.012) introduces computational tolerance through controlled uncertainty. Aperture scaling further embeds permeability tolerance. The third stage focuses on exploring how these scripts generate local adaptation to attractors, behavioral contrast across wall surfaces, and semantic encoding of spatial conditions. Tolerance manifests as the system's capacity to accommodate uncertainty and variability while maintaining structural coherence. Iterative runs allow the observation of topological variability and emergent spatial effects.

Finally, a comparative evaluation was conducted between the two scenarios. Criteria such as spatial adaptability, performative potential, and semantic depth were defined to assess the outcomes. The analysis revealed that tolerance operates at multiple levels: as uncertainty (randomized attractors, micro-variation), as bounded variability (extrusion ranges), as behavioral contrast (opposing wall logic), and as semantic flexibility (thresholds) (Figure 1).

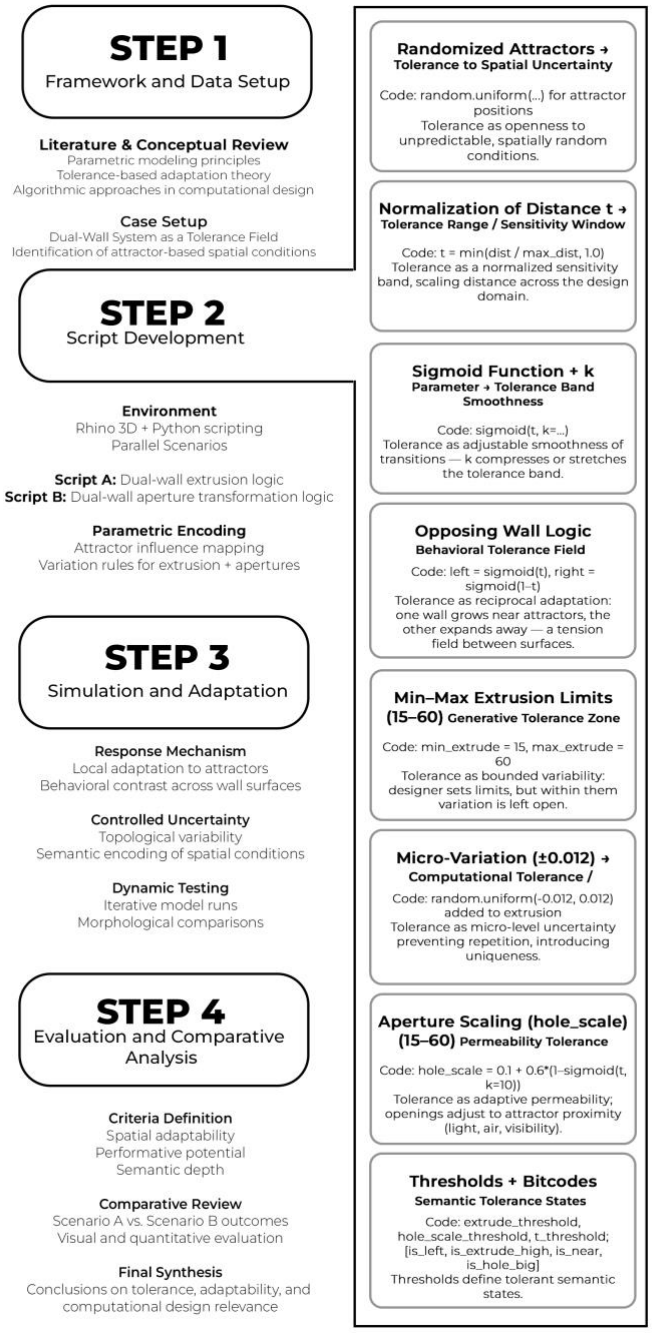


Figure 1: Methodological schema (Generated by the authors).

The synthesis of these findings underscores the relevance of tolerance-based adaptation in computational design and highlights its potential to articulate spatial systems that are both geometrically responsive and semantically encoded.

4. FINDINGS

In this study, tolerance is redefined as a generative design principle. Within this framework, two custom Rhino Python scripts were developed to examine how tolerance operates on architectural surfaces through attractor-based variation, micro-scale uncertainty. The following section presents the models and scenarios generated, demonstrating how computational tools articulate responsiveness, ambiguity, and performative differentiation in architecture.

4.1 Code 1: Parallel Opposing Response

Code 1 establishes a computational system that generates parametric variations between two parallel walls based on the positions of attractor points. Its core strategy lies in orchestrating opposing responses from the two walls: as one wall volumetrically expands when approaching an attractor, the other reduces its growth as it moves away. This reciprocal behavior produces a topographical tension field between the two surfaces. As outlined in the theoretical framework, such reciprocity resonates with Wiener's (2019) notion of feedback loops, where positive responses amplify while negative ones suppress change, thereby situating tolerance as a dynamic mediator rather than a fixed constraint. The interpretation of this behavior goes beyond formal differentiation: it demonstrates how tolerance operates as a field condition that mediates between contrasting spatial logics, producing adaptable yet coherent configurations.

The originality of Code 1 lies in its integration of tolerance into dual-wall dynamics, where variability is generative design condition. Unlike conventional parametric models that often rely on uniform attractor responses, this system explicitly encodes reciprocity and divergence as a design principle. tolerance aligns with McVicar's (2016) view of a margin of opportunity and with Bates and Sergison's notion of unconventional use, since the bounded extrusion and aperture ranges create spaces for variation within control. By positioning tolerance as both a spatial mediator and a computational driver, Code 1 contributes a novel framework for reading and constructing architectural surfaces that are simultaneously adaptive, differentiated, and semantically encoded.

The primary intention of this script is to demonstrate the presence and legibility of the tolerance concept within a computational framework, using a dual-wall system inspired by responsive façade examples. In this setup, tolerance is embodied in the attractors' spatial positions and their influence on dynamic extrusion and aperture scaling. These behavioral differences between the two walls produce interactive spatial conditions. The attractor points, randomized and embedded within the space between the walls serve as spatial inputs that impact the system's behavior. Tolerance is positioned as spatial uncertainty: attractor placement acts as a variable input that challenges determinism and ensures adaptability. The randomness simulates unpredictable real-life conditions and reveals the potential for design to evolve through such uncertainty.

Extrusion values are developed along the normal surface and bounded by predefined minimum and maximum values (ranging from 15 to 60 units) (Figure 3). These limits define the extent of designer control yet they are intentionally left open to variability. As theorized, such bounded ranges function as tolerance zones producing unique configurations through attractor influence. These ranges create generative tolerance zones, where each attractor-driven variation results in a unique configuration. In this sense, tolerance is defined by the range of variations that operate within these limits (Figure 2). This reflects tolerance as bounded variability, where designer-defined thresholds create room for controlled openness.

For each grid cell, the shortest distance to any attractor point is calculated and normalized relative to the overall grid dimensions, resulting in a scalar value t. A t value closer to 0 indicates proximity to an attractor, while a value closer to 1 indicates greater distance (Figure 3). This distance serves as the foundation for all subsequent variations and determines the computational sensitivity of the spatial system.

The script processes it through a sigmoid function, which compresses t between 0 and 1, generating smooth transitions rather than abrupt changes. This process demonstrates the difference between deterministic and probabilistic design. The same distance value can produce varying outcomes depending on the sharpness of the sigmoid curve (Figure 2).

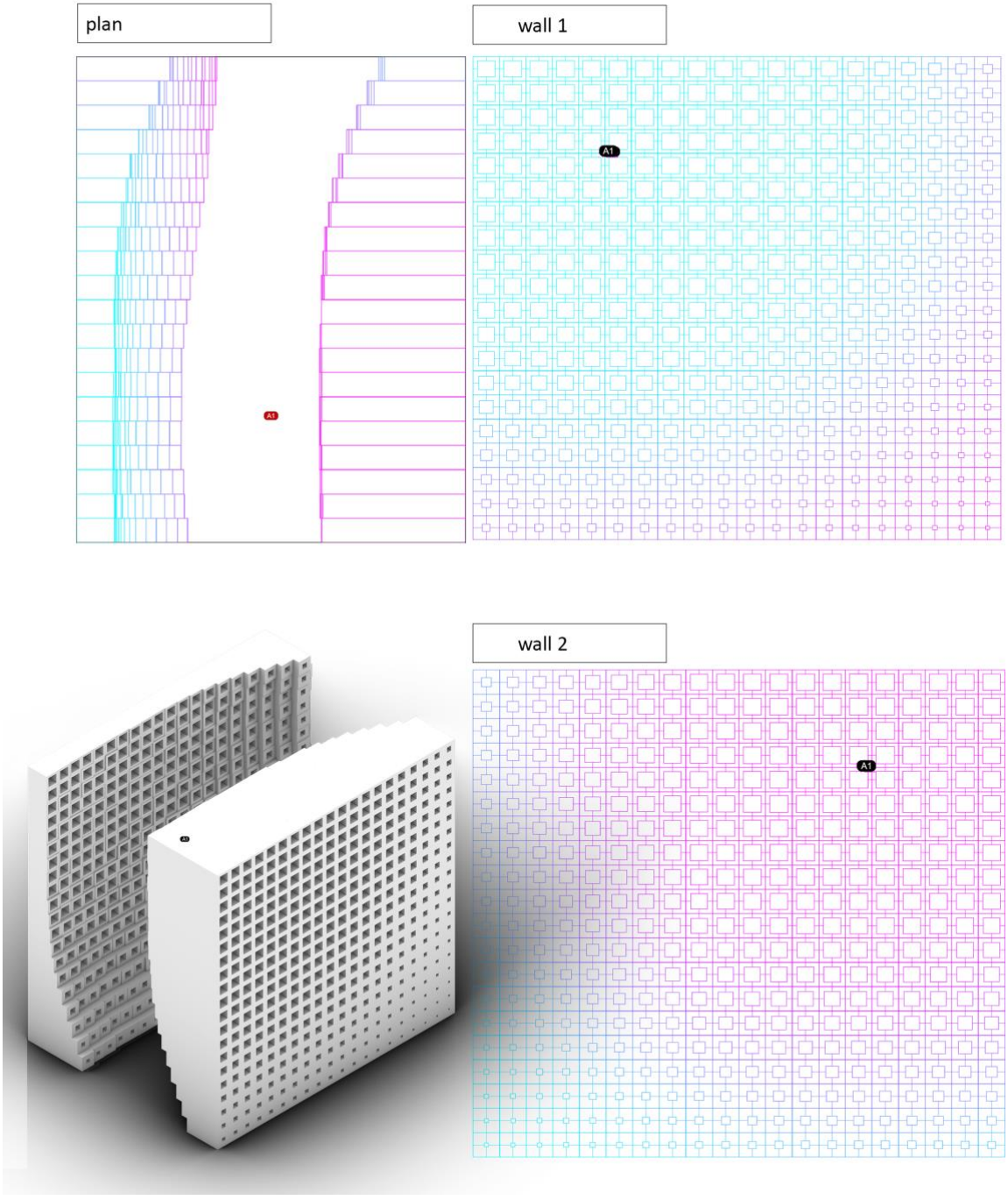


Figure 2: Code 1 3d geometry (Generated by the author).

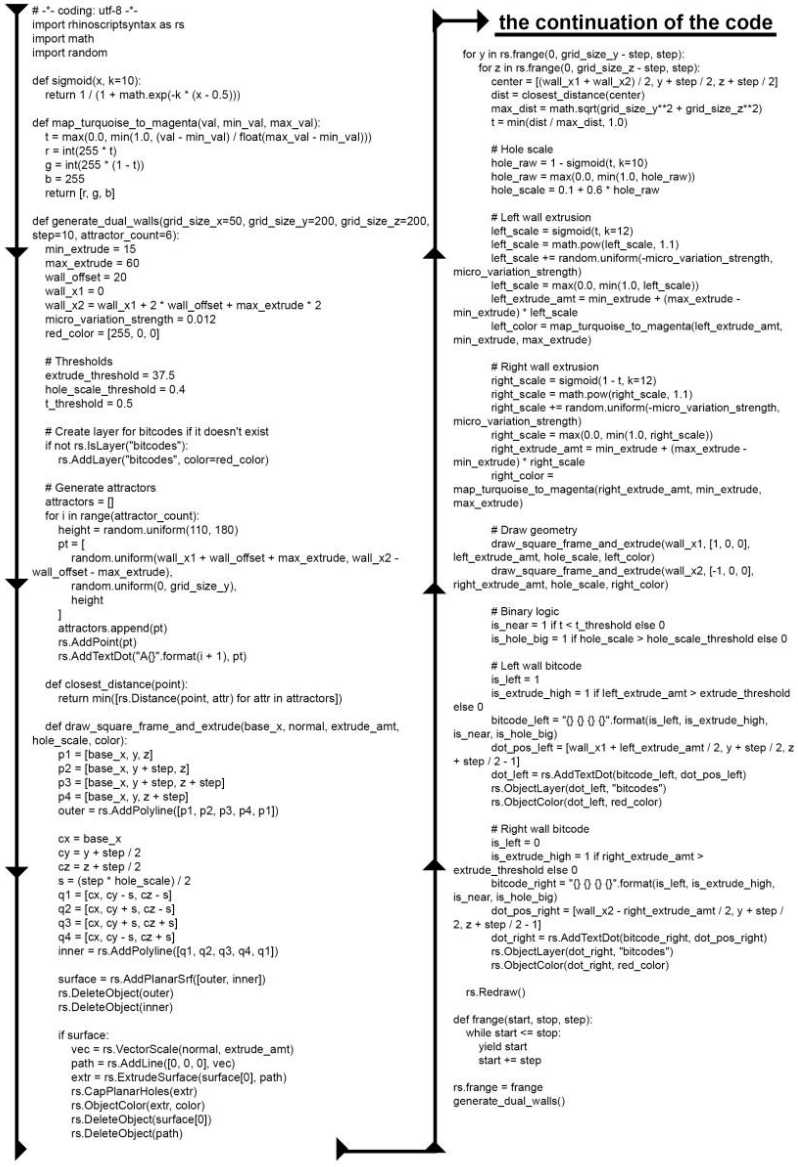
In the script, the sigmoid function is defined as:

$$(x, k) = \frac{1}{1+e-k(x-0.5)} \quad (1)$$

where x is the normalized distance (0–1) and k controls curve sharpness: higher k yields sharper transitions, lower k smoother gradients. This shows tolerance as adjustable thresholds and shifting sensitivities rather than fixed limits (Equation 1).

The extrusion logic of the two walls is intentionally inverse: Left Wall: Extrusion magnitude is directly proportional to sigmoid(t). As the cell approaches the attractor, its extrusion increases. Right Wall: Extrusion is determined by sigmoid(1-t). This wall expands more as it moves away from the attractor (Figure 3). This opposing wall logic operationalizes tolerance as behavioral contrast, encoding divergence as a generative design principle.

Although the system behavior is primarily defined at the level of the walls, an additional layer of algorithmic complexity is introduced: small-scale randomness (±0.012) is added to the extrusion values. These micro-variations break the determinism of the pattern and encode computational tolerance into the system. Such micro-level uncertainty prevents the formation of strictly repetitive units, enhancing the uniqueness of the surface. Each extruded unit also includes a central aperture (hole), with its scale calculated using the formula. This ensures that apertures are larger in regions closer to the attractors and smaller farther away. These perforations can serve as performance metrics—affecting spatial permeability, light transmission, or environmental integration (Figure 2).



```
# -*- coding: utf-8 -*-
import rhinoscriptsyntax as rs
import math
import random

def sigmoid(x, k=10):
    return 1 / (1 + math.exp(-k * (x - 0.5)))

def map_turquoise_to_magenta(val, min_val, max_val):
    t = max(0.0, min(1.0, (val - min_val) / float(max_val - min_val)))
    r = int(255 * t)
    g = int(255 * (1 - t))
    b = 255
    return [r, g, b]

def generate_dual_walls(grid_size_x=50, grid_size_y=200, grid_size_z=200,
step=10, attractor_count=6):
    min_extrude = 15
    max_extrude = 60
    wall_offset = 20
    wall_x1 = 0
    wall_x2 = wall_x1 + 2 * wall_offset + max_extrude * 2
    micro_variation_strength = 0.012
    red_color = [255, 0, 0]

    # Thresholds
    extrude_threshold = 37.5
    hole_scale_threshold = 0.4
    t_threshold = 0.5

    # Create layer for bitcodes if it doesn't exist
    if not rs.IsLayer("bitcodes"):
        rs.AddLayer("bitcodes", color=red_color)

    # Generate attractors
    attractors = []
    for i in range(attractor_count):
        height = random.uniform(110, 180)
        pt = [
            random.uniform(wall_x1 + wall_offset + max_extrude, wall_x2 -
wall_offset - max_extrude),
            random.uniform(0, grid_size_y),
            height
        ]
        attractors.append(pt)
        rs.AddPoint(pt)
        rs.AddTextDot("A{}".format(i + 1), pt)

    def closest_distance(point):
        return min([rs.Distance(point, attr) for attr in attractors])

    def draw_square_frame_and_extrude(base_x, normal, extrude_amt,
hole_scale, color):
        p1 = [base_x, y, z]
        p2 = [base_x, y + step, z]
        p3 = [base_x, y + step, z + step]
        p4 = [base_x, y, z + step]
        outer = rs.AddPolyline([p1, p2, p3, p4, p1])

        cx = base_x
        cy = y + step / 2
        cz = z + step / 2
        s = (step * hole_scale) / 2
        q1 = [cx, cy - s, cz - s]
        q2 = [cx, cy + s, cz - s]
        q3 = [cx, cy + s, cz + s]
        q4 = [cx, cy - s, cz + s]
        inner = rs.AddPolyline([q1, q2, q3, q4, q1])

        surface = rs.AddPlanarSrf([outer, inner])
        rs.DeleteObject(outer)
        rs.DeleteObject(inner)

        if surface:
            vec = rs.VectorScale(normal, extrude_amt)
            path = rs.AddLine([0, 0, 0], vec)
            extr = rs.ExtrudeSurface(surface[0], path)
            rs.CapPlanarHoles(extr)
            rs.ObjectColor(extr, color)
            rs.DeleteObject(surface[0])
            rs.DeleteObject(path)
```

the continuation of the code

```
    for y in rs.frange(0, grid_size_y - step, step):
        for z in rs.frange(0, grid_size_z - step, step):
            center = [(wall_x1 + wall_x2) / 2, y + step / 2, z + step / 2]
            dist = closest_distance(center)
            max_dist = math.sqrt(grid_size_y**2 + grid_size_z**2)
            t = min(dist / max_dist, 1.0)

            # Hole scale
            hole_raw = 1 - sigmoid(t, k=10)
            hole_raw = max(0.0, min(1.0, hole_raw))
            hole_scale = 0.1 + 0.6 * hole_raw

            # Left wall extrusion
            left_scale = sigmoid(t, k=12)
            left_scale = math.pow(left_scale, 1.1)
            left_scale += random.uniform(-micro_variation_strength,
micro_variation_strength)
            left_scale = max(0.0, min(1.0, left_scale))
            left_extrude_amt = min_extrude + (max_extrude -
min_extrude) * left_scale
            left_color = map_turquoise_to_magenta(left_extrude_amt,
min_extrude, max_extrude)

            # Right wall extrusion
            right_scale = sigmoid(1 - t, k=12)
            right_scale = math.pow(right_scale, 1.1)
            right_scale += random.uniform(-micro_variation_strength,
micro_variation_strength)
            right_scale = max(0.0, min(1.0, right_scale))
            right_extrude_amt = min_extrude + (max_extrude -
min_extrude) * right_scale
            right_color =
map_turquoise_to_magenta(right_extrude_amt, min_extrude,
max_extrude)

            # Draw geometry
            draw_square_frame_and_extrude(wall_x1, [1, 0, 0],
left_extrude_amt, hole_scale, left_color)
            draw_square_frame_and_extrude(wall_x2, [-1, 0, 0],
right_extrude_amt, hole_scale, right_color)

            # Binary logic
            is_near = 1 if t < t_threshold else 0
            is_hole_big = 1 if hole_scale > hole_scale_threshold else 0

            # Left wall bitcode
            is_left = 1
            is_extrude_high = 1 if left_extrude_amt > extrude_threshold
else 0
            bitcode_left = "{} {} {} {}".format(is_left, is_extrude_high,
is_near, is_hole_big)
            dot_pos_left = [wall_x1 + left_extrude_amt / 2, y + step / 2, z
+ step / 2 - 1]
            dot_left = rs.AddTextDot(bitcode_left, dot_pos_left)
            rs.ObjectLayer(dot_left, "bitcodes")
            rs.ObjectColor(dot_left, red_color)

            # Right wall bitcode
            is_left = 0
            is_extrude_high = 1 if right_extrude_amt >
extrude_threshold else 0
            bitcode_right = "{} {} {} {}".format(is_left, is_extrude_high,
is_near, is_hole_big)
            dot_pos_right = [wall_x2 - right_extrude_amt / 2, y + step /
2, z + step / 2 - 1]
            dot_right = rs.AddTextDot(bitcode_right, dot_pos_right)
            rs.ObjectLayer(dot_right, "bitcodes")
            rs.ObjectColor(dot_right, red_color)

    rs.Redraw()

def frange(start, stop, step):
    while start <= stop:
        yield start
        start += step

rs.frange = frange
generate_dual_walls()
```

Figure 3: Code 1 Rhinoscript (Generated by the authors).

4.2 Code 2: Synchronously Diverging Surfaces

Code 2 represents the second scenario of a dual-wall system generated through attractor-based parametric variation. In this case, both the left and right walls exhibit identical behavior: they increase in volume and reduce aperture size as they move away from the attractor points. While maintaining synchronized parametric responses across both surfaces, the code embeds tolerance into the system through micro-variations and semantic encodings. Code 2 frames tolerance as

synchronized adaptation, demonstrating how coherence and regularity can themselves serve as generative margins within parametric systems.

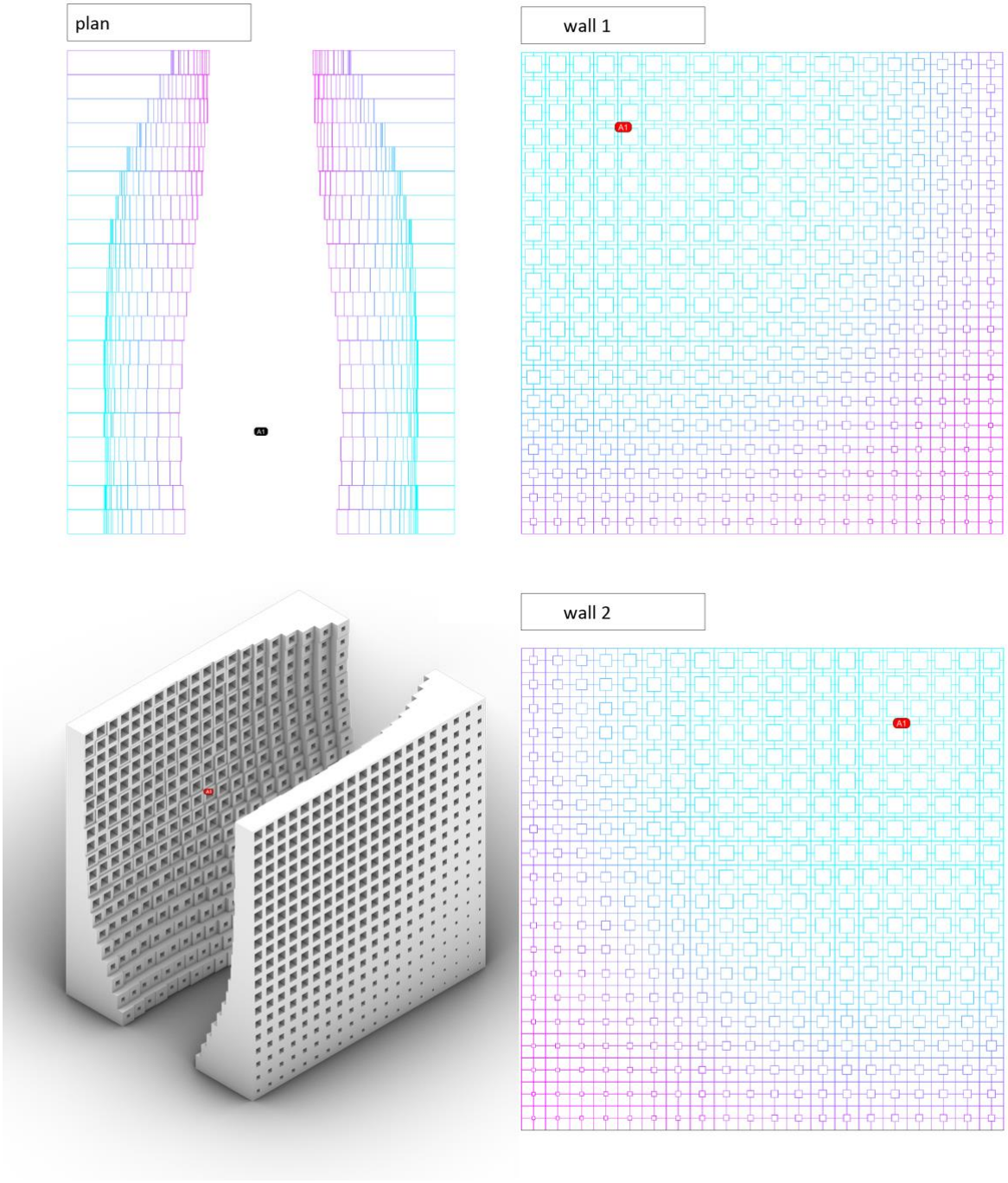


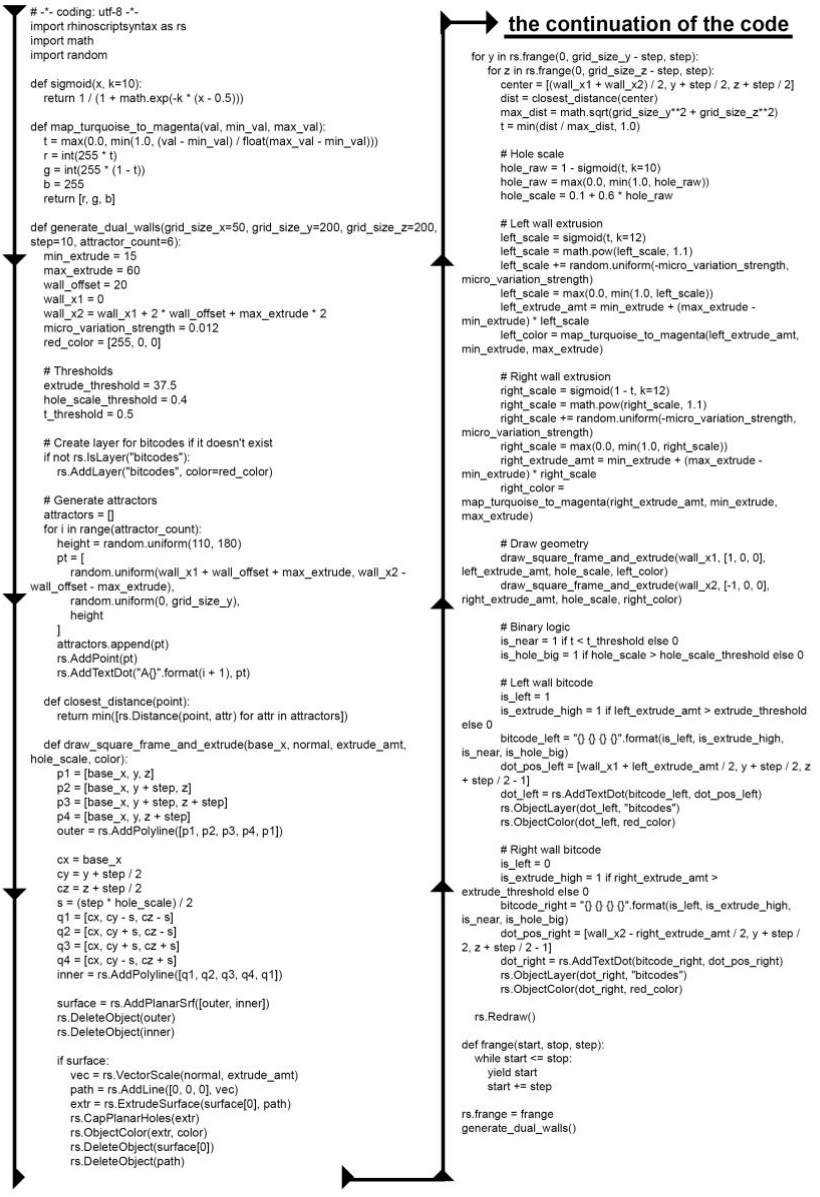
Figure 4: Code 2 Rhinoscript and 3d geometry (Generated by the author).

The architectural generation process unfolds through the following steps: The foundational structure of Code 2 mirrors that of Code 1. The script begins by constructing a three-dimensional grid, within which grid sizes along the x, y, and z axes define the bounds of the design domain and its zones of tolerance (Figure 5). Attractors are placed randomly within the space between the two walls, and their positions establish spatial intervals that trigger morphological transformation. As in the theoretical discussion of flows, these attractors embody spatial uncertainty, functioning as unpredictable inputs that generate openings for variation (McCullough, 2005).

As in Code 1, the grid serves as the base from which extrusion occurs along the surface normal, with each cell's shortest distance to the nearest attractor normalized into a scalar value .This value is processed through a sigmoid function to generate performative parameters such as extrusion height and aperture size, where tolerance operates as a sensitivity band: the sharpness coefficient k determines how smoothly or abruptly surfaces adapt to attractor proximity, framing tolerance as a tunable threshold of responsiveness. Unlike Code 1, both walls in Code 2 share the same extrusion logic, establishing a synchronized field of variation in which they expand simultaneously as they move away from attractors. This coordinated response embeds tolerance as bounded coherence, demonstrating adaptability through repetition and alignment rather than divergence, while added micro-variations prevent determinism and encode tolerance as micro-level uncertainty.

The relationship to hole scale mirrors that of Code 1, where aperture size increases as proximity to the attractor decreases. This inverse relationship creates context-sensitive porosity, shaped by spatial proximity to attractor fields (Figure 4). In this sense, permeability tolerance is inscribed in the surface logic, producing adaptable openings that mediate between light, air, and visibility—aligning with

the notion of interactive skins as dynamic, reconfigurable membranes (Park et al., 2011).



```
# -*- coding: utf-8 -*-
import rhinoscriptsyntax as rs
import math
import random

def sigmoid(x, k=10):
    return 1 / (1 + math.exp(-k * (x - 0.5)))

def map_turquoise_to_magenta(val, min_val, max_val):
    t = max(0.0, min(1.0, (val - min_val) / float(max_val - min_val)))
    r = int(255 * t)
    g = int(255 * (1 - t))
    b = 255
    return [r, g, b]

def generate_dual_walls(grid_size_x=50, grid_size_y=200, grid_size_z=200,
step=10, attractor_count=6):
    min_extrude = 15
    max_extrude = 60
    wall_offset = 20
    wall_x1 = 0
    wall_x2 = wall_x1 + 2 * wall_offset + max_extrude * 2
    micro_variation_strength = 0.012
    red_color = [255, 0, 0]

    # Thresholds
    extrude_threshold = 37.5
    hole_scale_threshold = 0.4
    t_threshold = 0.5

    # Create layer for bitcodes if it doesn't exist
    if not rs.IsLayer("bitcodes"):
        rs.AddLayer("bitcodes", color=red_color)

    # Generate attractors
    attractors = []
    for i in range(attractor_count):
        height = random.uniform(110, 180)
        pt = [
            random.uniform(wall_x1 + wall_offset + max_extrude, wall_x2 -
wall_offset - max_extrude),
            random.uniform(0, grid_size_y),
            height
        ]
        attractors.append(pt)
        rs.AddPoint(pt)
        rs.AddTextDot("A{}".format(i + 1), pt)

    def closest_distance(point):
        return min([rs.Distance(point, attr) for attr in attractors])

    def draw_square_frame_and_extrude(base_x, normal, extrude_amt,
hole_scale, color):
        p1 = [base_x, y, z]
        p2 = [base_x, y + step, z]
        p3 = [base_x, y + step, z + step]
        p4 = [base_x, y, z + step]
        outer = rs.AddPolyline([p1, p2, p3, p4, p1])

        cx = base_x
        cy = y + step / 2
        cz = z + step / 2
        s = (step * hole_scale) / 2
        q1 = [cx, cy - s, cz - s]
        q2 = [cx, cy + s, cz - s]
        q3 = [cx, cy + s, cz + s]
        q4 = [cx, cy - s, cz + s]
        inner = rs.AddPolyline([q1, q2, q3, q4, q1])

        surface = rs.AddPlanarSrf([outer, inner])
        rs.DeleteObject(outer)
        rs.DeleteObject(inner)

        if surface:
            vec = rs.VectorScale(normal, extrude_amt)
            path = rs.AddLine([0, 0, 0], vec)
            extr = rs.ExtrudeSurface(surface[0], path)
            rs.CapPlanarHoles(extr)
            rs.ObjectColor(extr, color)
            rs.DeleteObject(surface[0])
            rs.DeleteObject(path)
```

the continuation of the code

```
    for y in rs.frange(0, grid_size_y - step, step):
        for z in rs.frange(0, grid_size_z - step, step):
            center = [(wall_x1 + wall_x2) / 2, y + step / 2, z + step / 2]
            dist = closest_distance(center)
            max_dist = math.sqrt(grid_size_y**2 + grid_size_z**2)
            t = min(dist / max_dist, 1.0)

            # Hole scale
            hole_raw = 1 - sigmoid(t, k=10)
            hole_raw = max(0.0, min(1.0, hole_raw))
            hole_scale = 0.1 + 0.6 * hole_raw

            # Left wall extrusion
            left_scale = sigmoid(t, k=12)
            left_scale = math.pow(left_scale, 1.1)
            left_scale += random.uniform(-micro_variation_strength,
micro_variation_strength)
            left_scale = max(0.0, min(1.0, left_scale))
            left_extrude_amt = min_extrude + (max_extrude -
min_extrude) * left_scale
            left_color = map_turquoise_to_magenta(left_extrude_amt,
min_extrude, max_extrude)

            # Right wall extrusion
            right_scale = sigmoid(1 - t, k=12)
            right_scale = math.pow(right_scale, 1.1)
            right_scale += random.uniform(-micro_variation_strength,
micro_variation_strength)
            right_scale = max(0.0, min(1.0, right_scale))
            right_extrude_amt = min_extrude + (max_extrude -
min_extrude) * right_scale
            right_color =
map_turquoise_to_magenta(right_extrude_amt, min_extrude,
max_extrude)

            # Draw geometry
            draw_square_frame_and_extrude(wall_x1, [1, 0, 0],
left_extrude_amt, hole_scale, left_color)
            draw_square_frame_and_extrude(wall_x2, [-1, 0, 0],
right_extrude_amt, hole_scale, right_color)

            # Binary logic
            is_near = 1 if t < t_threshold else 0
            is_hole_big = 1 if hole_scale > hole_scale_threshold else 0

            # Left wall bitcode
            is_left = 1
            is_extrude_high = 1 if left_extrude_amt > extrude_threshold
else 0
            bitcode_left = "{} {} {} {}".format(is_left, is_extrude_high,
is_near, is_hole_big)
            dot_pos_left = [wall_x1 + left_extrude_amt / 2, y + step / 2, z
+ step / 2 - 1]
            dot_left = rs.AddTextDot(bitcode_left, dot_pos_left)
            rs.ObjectLayer(dot_left, "bitcodes")
            rs.ObjectColor(dot_left, red_color)

            # Right wall bitcode
            is_left = 0
            is_extrude_high = 1 if right_extrude_amt >
extrude_threshold else 0
            bitcode_right = "{} {} {} {}".format(is_left, is_extrude_high,
is_near, is_hole_big)
            dot_pos_right = [wall_x2 - right_extrude_amt / 2, y + step /
2, z + step / 2 - 1]
            dot_right = rs.AddTextDot(bitcode_right, dot_pos_right)
            rs.ObjectLayer(dot_right, "bitcodes")
            rs.ObjectColor(dot_right, red_color)

    rs.Redraw()

def frange(start, stop, step):
    while start <= stop:
        yield start
        start += step

rs.frange = frange
generate_dual_walls()
```

Figure 5: Code 2 Rhinoscript (Generated by the author).

4.3: Code 1 vs. Code 2 Matris – Response Logics and Tolerance Behavior

The comparative structure of Code 1 and Code 2 illustrates how interactive architecture can generate distinct design strategies through system-based responses. Within the scope of this study, it has been

discovered that the everyday spatial scenarios emerging between geometric complexity and attractors are, in fact, generated through calculable tolerance ranges. Both scripts are grounded in a shared set of base parameters—wall orientation, extrusion magnitude, attractor proximity, and hole scale—which define specific threshold intervals. These intervals constitute the framework of generative tolerance; however, the behavioral characteristics are embedded within the scripts themselves. Since the extrusion direction is defined by the designer's intervention, the orientation and behavior of the two scripts differ accordingly.

Code 1 produces oppositional surface behaviors in response to attractor points, establishing a spatial field of tension through the contrasting tendencies of the two walls. This duality resonates directly with Venturi's (2005) emphasis on the productivity of imperfection and the meaning-generating power of contrast. In this context, surfaces engage conceptually, forming a reciprocal relationship. In contrast, Code 2 proposes a synchronized system wherein both walls react in parallel as they move away from the attractors.

The key distinction between the two scripts lies in the extrusion logic. In Code 1, the left wall responds with increasing extrusion as sigmoid(t), while the right wall behaves inversely through sigmoid (1 - t), creating an inherent "dialectical" interaction within the system. Design parameters such as aperture and opacity are derived from this contrast. In Code 2, both walls share the same extrusion function, sigmoid(t), revealing the possibility of co-evolving response patterns within interactive systems. The attractor proximity parameter introduces divergent behavior in Code 1, with one wall expanding while the other contracts. This dynamic reflects McVicar's (2016) definition of generative tolerance as a range of opportunity that allows different responses to the same input. In Code 2, the same proximity results in lower extrusion and larger apertures across both walls, generating a permeable scenario in which interactive surfaces respond collectively to contextual conditions (Table 1).

Feature	Code 1: Parallel Opposing Response	Code 2: Parallel Synchronous Divergence
Response Orientation	Contrasting variation – surface behaviors are inversely oriented relative to the attractor.	Synchronous variation – both surfaces respond similarly by increasing as they move away from the attractor.
Extrusion Value Calculation	The left wall uses sigmoid(t), while the right wall uses sigmoid (1 - t) to generate opposite behaviors.	Both walls use sigmoid(t) to produce identical directional behavior.
Proximity to Attractor	Divides the system: proximity increases volume on one wall while decreasing it on the other.	Proximity results in low extrusion and large apertures on both walls.
Aperture Generation (Hole Scale)	hole_scale = 0.1 + 0.6 * (1 - sigmoid(t)); apertures are larger near attractors.	Uses the same equation; apertures increase near attractors, leading to lower opacity.
Micro Variation and Bit Encoding	±0.012 random variation in extrusion adds micro uncertainty	The same level of variation is applied appears synchronously on both walls for matching units.

Table 1: Code 1 and Code 2 Rhinoscripts Matris (Generated by the author).

The scenarios illustrate the distinct spatial behaviors generated by the two codes. In the scenarios of Code 1 (1.1–1.4), attractor points are reassigned randomly at each execution, leading to opposing responses between the walls: as one wall expands volumetrically when approaching an attractor, the other contracts as it moves away. This produces varying topological tension fields and generates diversity across scenarios. In contrast, the scenarios of Code 2 (2.1–2.4) follow a synchronized logic, with both walls expanding or contracting simultaneously in relation to attractor positions. As a result, the surfaces exhibit more coherent, continuous, and homogeneous behaviors (Figure 6).

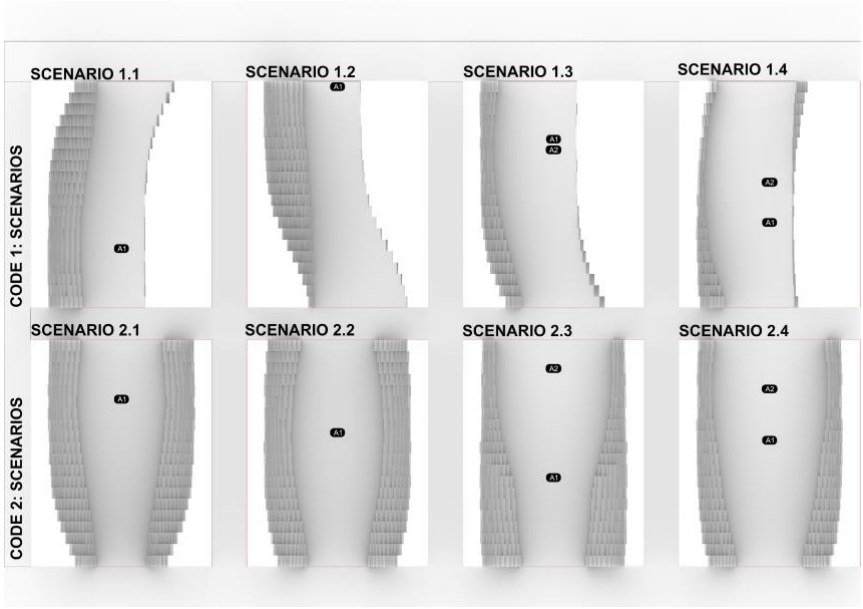


Figure 6: Codes Scenarious (Generated by the author).

5. RESULTS

This study redefines the concept of tolerance within the context of computational design, transforming it into a productive strategy that embraces uncertainty, openness, diversity, and system-level adaptability. The aim is to position tolerance not merely as a technical allowance but as a computational language capable of generating new spatial potentials. The two custom-developed Rhino Python scripts demonstrate this potential by integrating generative tolerance ranges into a performative, interactive, and context-sensitive design method.

The process moves away from singular and deterministic understandings of architecture, instead proposing a design logic based on attractor-referenced, tolerance-enabled variations. Through these two scripts, different surface behaviors were tested, showing how architectural elements can evolve from static components into dynamic, data-driven spatial interfaces. As these surfaces begin to generate semantic responses, their behaviors emerge from within the system itself. Rather than producing fixed outputs, the scripts embody a probabilistic and open-ended system capable of generating new architectural conditions.

Everyday spaces are inherently complex systems, shaped by instantaneous and interactive situations. For such systems to be translated into computationally designed environments, the concept of tolerance must define a new set of operational design conditions. This study rearticulates the concept of tolerance through the role of coding in the design process and situates it in relation to the position of the designer. The digital environment, with its capacity for simulation and production, emerges as an active agent within design. In this framework, design is no longer confined to a single moment in the designer's mind, but unfolds as a process that can be reproduced over time and tested under varying conditions. Codes and the ranges of variation they generate come alive through interaction with the environment, giving rise to dynamic spatial possibilities. In this process, spatial interaction acquires a tangible dimension, creating a new interactive domain between the digital and the physical. The findings demonstrate that digital and computational design environments are positioned as integral and generative actors within architectural design.

The study presents a limited scope as it has been conducted within a specific digital platform (Rhino 3D) and through two custom Python scripts. Broader implementation across different software environments, larger datasets, or physical prototyping could increase the generalizability of this approach. Future research may focus on integrating tolerance with spatial interactions, testing the behaviors of geometry and the reproducibility, variability, and temporality of constructed spaces, thereby enabling a more comprehensive examination of the concept at both experimental and applied levels.

Conflict of Interest Statement

The manuscript is entitled "Rethinking Tolerance through Interactive and Complex Architectural Systems" has not been published elsewhere and that it has not been submitted simultaneously for publication elsewhere.

Author contribution

The authors' contributions to this study are as follows: The first author contributed 60% to the development of the concept, data analysis, manuscript writing, and overall execution of the study. The second

author contributed 40% to the conceptual phase, script development, and the revision process. These percentages accurately reflect the extent of each author's contribution to the work.

References

Banihashemi, S., Assadimoghadam, A., Hajirasouli, A., LeNguyen, K., & Mohandes, S. R. (2024). Parametric design in construction: a new paradigm for quality management and defect reduction. International *Journal of Construction Management*, 1-18. https://doi.org/10.1080/15623599.2024.2447653

Banihashemi, S., Tabadkani, A., & Hosseini, M. R. (2018). Integration of parametric design into modular coordination: A construction waste reduction workflow. *Automation in Construction*, 88, 1-12. https://doi.org/10.1016/j.autcon.2017.12.026

Bates, S., & Sergison, J. (1999). *Teaching and learning: The practice of architecture*. Academy Editions.

Bundy, A. (2007). Computational thinking is pervasive. *Journal of Scientific and Practical Computing, 1*(2), 67-69. https://www.pure.ed.ac.uk/ws/portalfiles/portal/408398/Computational_Thinking_is_Pervasive.pdf

De Certeau, M. (2008). *The practice of everyday life* (S. Sert, Trans.). Dost Kitabevi Publications (Original work published 1980).

Fox, M., & Kemp, M. (2009). *Interactive architecture*. Princeton Architectural Press.

Jaskiewicz, T. J. (2013). *Towards a methodology for complex adaptive interactive architecture* [Doctoral dissertation, Technische Universiteit Delft]. *TU Delft Repository.* https://repository.tudelft.nl/islandora/object/uuid:a81827c5-7d65-4cc7-9fab-20fab3a14c30

Kolarevic, B. (2014). Why we need architecture of tolerance. *Architectural Design, 84*(1), 128–132. https://doi.org/10.1002/ad.1712

Kas, O. (2004). *Towards a new kind of building: A designer's guide for nonstandard architecture* [Doctoral dissertation, Technische Universiteit Delft]. *TU Delft Repository.* https://repository.tudelft.nl/islandora/object/uuid:291a8d50-c45a-4e20-8ad8-b9d6426d3f05

McCullough, M. (2004). *Digital ground: Architecture, pervasive computing, and environmental knowing*. MIT Press.

McVicar, M. T. (2016). *Precision in architectural production* [Doctoral dissertation, Cardiff University]. *ORCA Cardiff University Repository*. https://orca.cardiff.ac.uk/id/eprint/97224/

Moloney, J. (2009). Kinetic architectural skins and the computational sublime. *Leonardo, 42*(1), 65–70. https://doi.org/10.1162/LEON.2009.42.1.65

Nasir, O., & Kamal, M. A. (2023). Exploring the role of parametric architecture in building design: An inclusive approach. *Facta Universitatis, Series: Architecture and Civil Engineering*, *21*(1), 95–114. https://doi.org/10.2298/fuace230114007n

Novak, M. (2004, May 5). Marcos Novak interview [Interview by A. Ludovico]. Neural. http://www.neural.it/english/marcosnovak.htm

Oosterhuis, K. (2007). *Towards a new kind of building: A designer's guide for non-standard architecture*. NAi Publishers.

Oxman, R. (2017). Thinking difference: Theories and models of parametric design thinking. *Design Studies*, 52, 4–39. https://doi.org/10.1016/j.destud.2017.06.001

Park, J., Moere, A. V., & Tomitsch, M. (2011). The role of physicality in the design of ambient information systems. In P. Campos et al. (Eds.), *Human-Computer Interaction – INTERACT 2011* (pp. 151–167). Springer. https://doi.org/10.1007/978-3-642-23774-4_11

Tabadkani, A., Banihashemi, S., & Hosseini, M. R. (2018). Daylighting and visual comfort of oriental sun responsive skins: A parametric analysis. *Building Simulation, 11*(4), 663–676. https://doi.org/10.1007/s12273-018-0433-0

Venturi, R. (2005). *Complexity and contradiction in architecture* (E. Şener, Trans.). Şevki Vanlı Mimarlık Vakfı Publications. (Original work published 1966)

Wiener, N. (2019). *Cybernetics or control and communication in the animal and the machine*. MIT Press (Original work published 1948).

Yi, Y. K. (2019). Building facade multi-objective optimization for daylight and aesthetical perception. *Building and Environment, 156*, 178–190. https://doi.org/10.1016/j.buildenv.2019.04.002

Yücel, G., & Ökten, A. (2020). *Space and body: Spatial experience in everyday life*. İletişim Publications.