A Extended Gallery

This appendix presents additional visualizations and intuitive descriptions that complement the examples in the *Gallery* section. We include mirror rooms, translation prisms, and thickened singularity visualizations. Figure 8 shows mirror rooms formed by reflecting rays across the walls of triangles. Figure 9 shows two translation prisms [2], which are closed 3-manifolds obtained by taking our flat surface rooms and identifying their floors with their ceilings through a vertical translation, so the height coordinate becomes periodic. Figure 10 shows how singularities in our flat surface visualizations can be rendered as vertical cylinders.

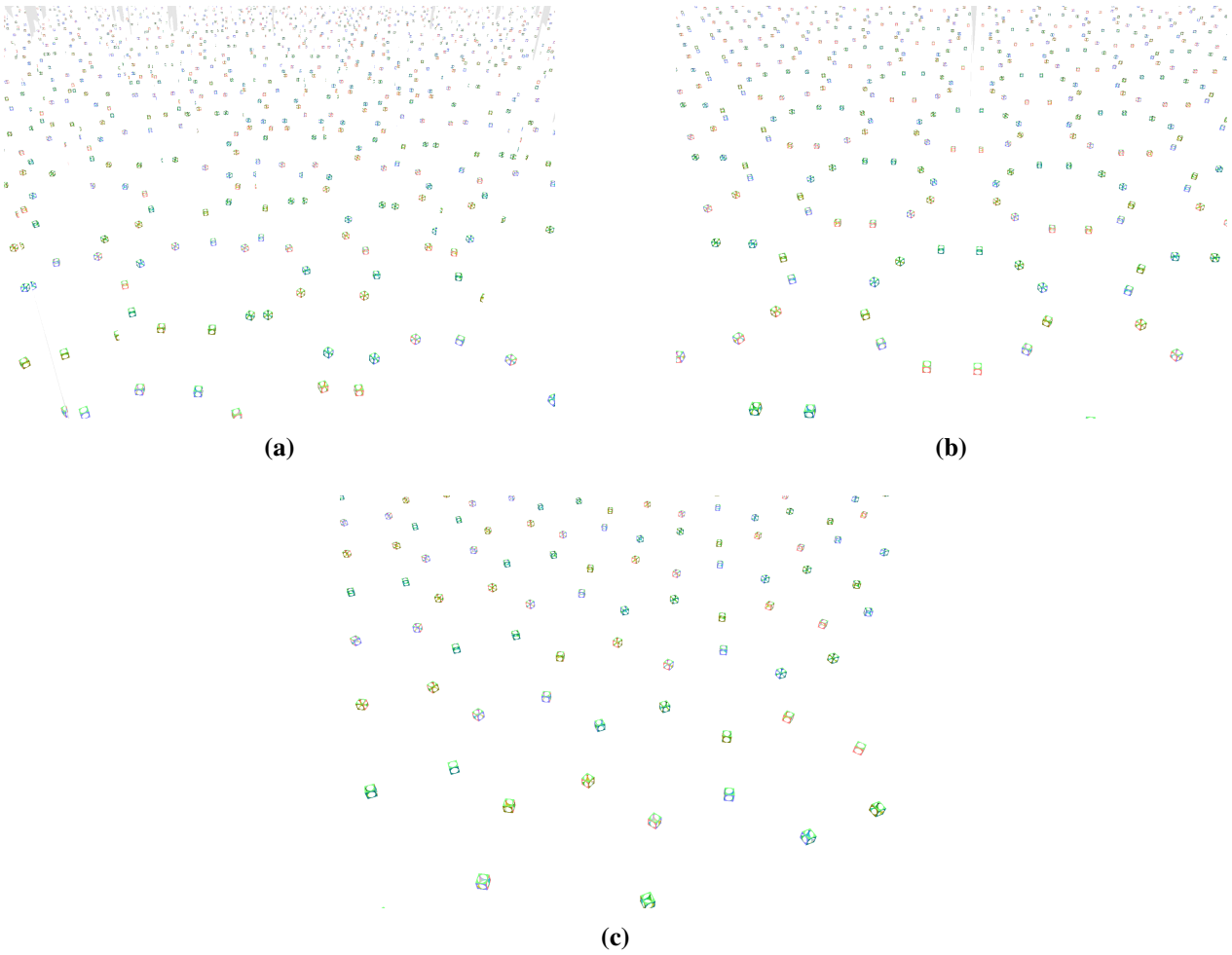
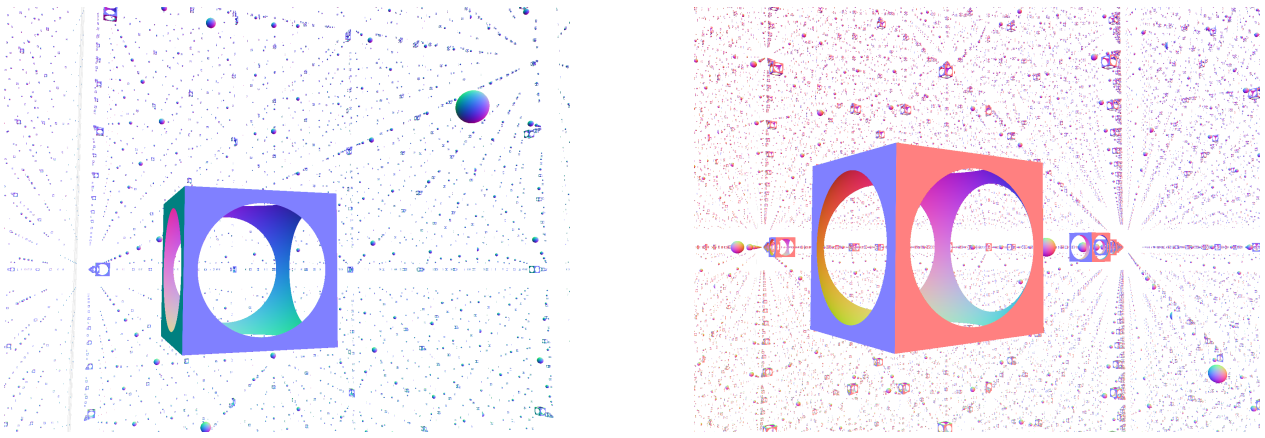


Figure 8: *Top-down view in mirror rooms with very high walls and whose floor plans are triangles. (a) A triangle with a nearly irrational angle. (b) A* 30°-60°-90° *triangle. (c) An equilateral triangle.*

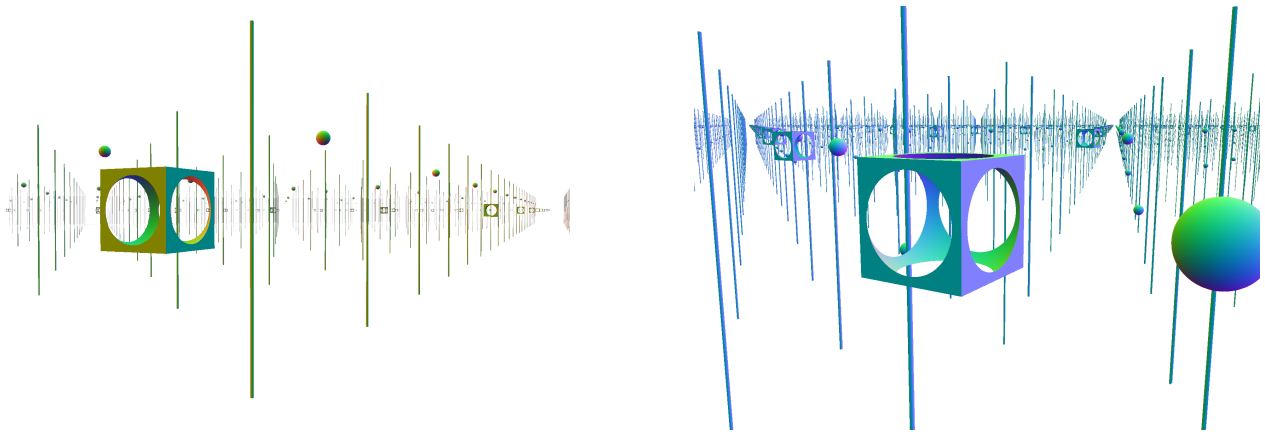
B More on Signed-Distance Functions

A signed distance function (SDF) is a scalar field $d(\vec{p})$ that implicitly encodes a shape by assigning to each point $\vec{p}$ its shortest signed distance to the surface of the object so that the shape itself is recovered exactly as the zero-level set $d(\vec{p}) = 0$. SDFs for many primitive shapes, such as spheres, cubes, cylinders, and tori, are well-known and serve as modular building blocks. More complex and organic shapes can be constructed by combining and deforming these building blocks using different numerical operations involving the component SDFs, and so that resulting zero-level set matches the intended composite shape. For example,



(a) **(b)**

Figure 9: *Visualizations of translation prisms over the L translation surface and the double pentagon.*



(a) **(b)**

Figure 10: *Visualizations of singular points represented as cylinders in our flat surface renderings.*

the fundamental set-theoretic operations of union, intersection, and difference translate directly into pointwise min, max, and negation on the SDFs, forming the basis of constructive solid geometry (CSG):

Operation	*Formula*	*Effect*
Union	$d_\cup(\vec{p}) = \min(d_1(\vec{p}), d_2(\vec{p}))$	join two objects
Intersection	$d_\cap(\vec{p}) = \max(d_1(\vec{p}), d_2(\vec{p}))$	keep overlap only
Difference	$d_\Delta(\vec{p}) = \max(d_1(\vec{p}), -d_2(\vec{p}))$	subtract second object from first

If a scene consists of N shapes modeled by SDFs $d_1, d_2, \cdots, d_N$, the scene itself is then modeled as one SDF that results from combining all the shapes in the scene:

$$d_{\text{scene}}(\vec{p}) = \min_{i=1}^{N} d_i(\vec{p}).$$

For more intricate geometries, practitioners rely on techniques such as voxel-based distance field generation, mesh-to-SDF conversion algorithms, or, more recently, neural implicit networks trained to approximate continuous SDFs directly from data. We refer the interested reader to the excellent survey [17]. Lastly, for

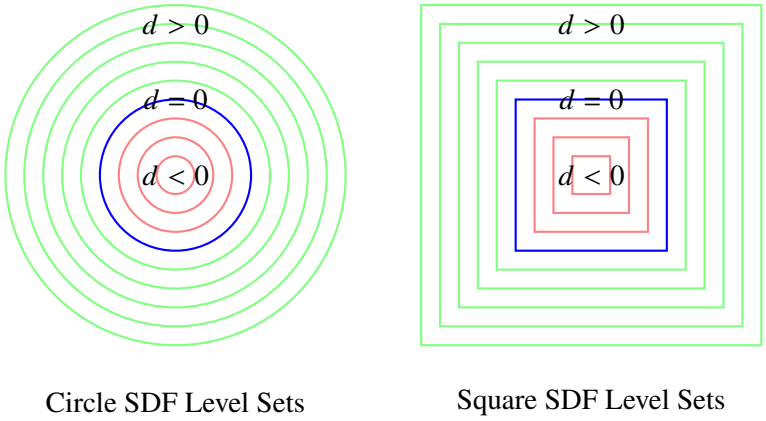


Figure 11: *Two-dimensional signed distance function level sets. Each contour shows points at fixed distance d from the surface: red for $d < 0$, blue for $d = 0$, and green for $d > 0$.*

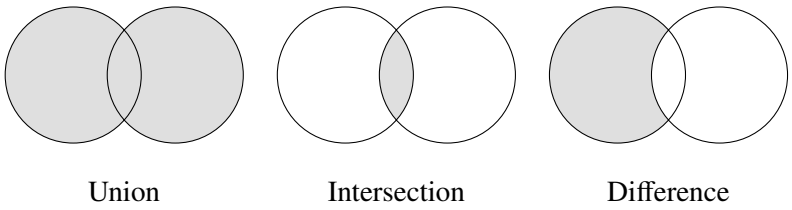


Figure 12: *The three fundamental CSG operations:* Union *combines objects,* intersection *preserves the overlapping regions, and* difference *subtracts one object from another.*

a comprehensive catalog of hand-crafted primitives, deformation operators, and ready-to-use GLSL code implementations, we refer the reader to Inigo Quílez's distance function compendium [16].

C Qualitative Experience and Educational Value

In our own experience, and in observing users during outreach, we have collected a few consistent observations. First, many abstract properties of flat surfaces come to life in ways that contradict everyday Euclidean expectations: walking straight ahead and, without visual breaks, finding oneself in an identical copy of the room brings the Pac-Man analogy to life, while circling a singularity and finding the room unexpectedly rotated (because the cone angle is not 2π) is surprising the first time it is experienced. We also notice that, after only a few minutes, users start spotting periodic orbits by sight, as periodic orbits pop out as low-level patterns, whereas aperiodic ones appear more visually noisy. Lastly, our simulated mirror rooms are as intriguing as the real ones: when a ray bounces off a side, it extends into another copy of the room that is one reflection away, so reflected objects march off toward vanishing points and shrink at the correct rate, giving an immediate sense of optical "infinity." In short, the simulation lets newcomers absorb new concepts by experiencing the phenomena firsthand, not by reading definitions or staring at static diagrams.

D Magnified and Rotated Torus Figure

Figure 13 is a magnified view of the torus in Figure 2 from a different angle. The geometry is untouched, and the torus is only enlarged and rotated. The apparently "undersized" diameter of the torus might appear uncanny at first, yet that scale is the faithful result of rolling the original rectangle into a torus.

E Software Design Considerations

Our software design choices aimed to balance ease of use, the potential for broad adoption, and flexibility by leveraging well-established metaphors and tools:

- *Usability*: For end-users, interaction with the simulation is modeled after familiar 3D video game environments. Navigation uses standard first-person and free-camera controls, drawing on well-established, time-tested schemes that users already know or can quickly pick up. Also, the simulation is delivered as a web application which runs directly in modern browsers without requiring any additional installations, plugins, or specific operating systems.
- *Adoption*: The implementation uses standard and widely adopted technologies in the computer graphics and mathematical visualization communities:
  - Ray marching for rendering signed distance functions, a common technique for visualizing implicit geometries and non-Euclidean spaces.
  - OpenGL shaders for efficient GPU-accelerated rendering.
  - Three.js, a popular WebGL-based framework that is accessible, well-documented, and supported by a large community.
- *Generality.* Our implementation consists of a compact ray marching kernel and a boundary-identification layer that supports flat spaces defined by edge identifications. Instead of requiring developers to hand-code SDFs directly, users can specify walls, gluing rules, and scene assets using a JSON file, and our system then synthesizes a shader that incorporates these specifications. Each rendered scene in our app is described by a JSON file, making it easy to modify or add examples that fit our framework.

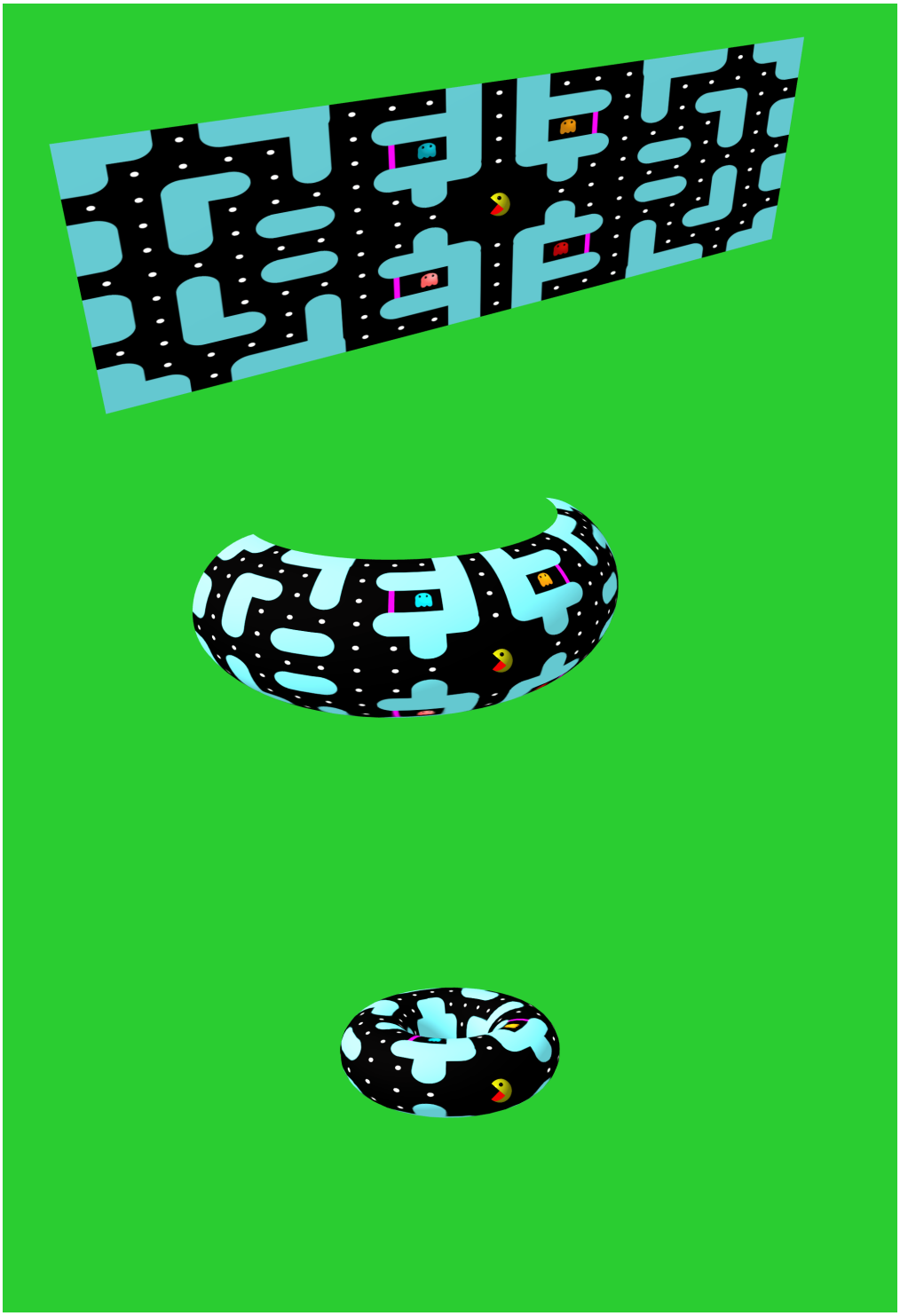


Figure 13: *A magnified version of Figure 2.*