EUROGRAPHICS 2024 / N.N. and N.N.
(Editors)

Volume 0 (2024), Number 0

PossibleImpossibles: Exploratory Procedural Design of Impossible Structures

Yuanbo Li[1] and Tianyi Ma[1] and Zaineb Aljumayaat[2] and Daniel Ritchie[1]

[1]Brown University, USA
[2]Rhode Island School of Design, USA

Figure 1: *Our method can generate different kinds of structures that appear to be impossible in the 3D space.*

Abstract

We present a method for generating structures in three-dimensional space that appear to be impossible when viewed from specific perspectives. Previous approaches focus on helping users to edit specific structures and require users to have knowledge of structural positioning causing the impossibility. On the contrary, our system is designed to aid users without prior knowledge to explore a wide range of potentially impossible structures. The essence of our method lies in features we call visual bridges *that confuse viewers regarding the depth of the resulting structure. We use these features as starting points and employ procedural modeling to systematically generate the result. We propose scoring functions for enforcing desirable spatial arrangement of the result and use Sequential Monte Carlo to sample outputs that score well under these functions. We also present a proof-of-concept user interface and demonstrate various results generated using our system.*

CCS Concepts

• ***Computing methodologies*** → ***Shape analysis;***

1. Introduction

Impossible structures were initially conceptualized by mathematician Roger Penrose and later gained widespread recognition through the artworks of M.C. Escher during the 20th century. They are characterized by small components that do not appear to violate Euclidean geometry when viewed individually, but when combined, create a structure that is impossible to be physically constructed [PP58, Fou]. These structures have gained recognition in recent years through their incorporation in various applications and designs in games and movies [Ale08, Gam, Nol10].

Creating impossible structures in three-dimensional space requires a combination of technical expertise and artistic talent. Previous research has focused on providing users with tools to edit a single structure and make it impossible [SDG03, WFY*10, OF08, SRC20]. However, as these approaches are goal-directed systems, users need to know the exact structure they want to model before using these systems and they need to have an understanding of the specific structural positions and connections that cause the structure to be deemed impossible.

We present a system that enables non-experts to explore different potential structures that appear impossible in 3D space. The key components of our resulting structures are special features we call *visual bridges*, which are far-apart substructures in 3D space that give the illusion of connectivity in 2D space. Our key insight is that to confuse viewers regarding the three-dimensional depth of the impossible structure, the system must construct a pathway that

connects the separate components of the visual bridges. This pathway serves as evidence that the components of visual bridges are, in fact, far apart in 3D space, despite appearing to be connected in 2D space. Our system then combines the pathway and the visual bridges to create a cycle in 2D space that suggests the components of the visual bridges are both connected and far apart at the same time.

We introduce a procedural language to generate the impossible structure by adding one substructure at a time. A key challenge for our approach is to create a sequence of randomized substructures that projects as a cycle in 2D space. We tackle this challenge in three steps. First, the system chooses visual bridges whose components appear connected in 2D space but separate in 3D space. Then, the system performs a random derivation from the components to introduce randomness and allow the exploration of different potential results. Lastly, the system introduces an algorithm to find a pathway connecting the starting and ending components of the visual bridges.

However, the previous algorithm does not guarantee the creation of a desirable structure. The structural components might overlap or occlude each other, making the impossibility difficult to discern. To control the generation process, we propose scoring functions that enforce desirable spatial arrangements of the structure. We then use Sequential Monte Carlo sampling to steer our procedural model and select outputs that score well under these functions. We demonstrate the effectiveness of our system by generating a variety of impossible structures using different grammars and further develop an interface to facilitate interaction with our model.

In summary, our main contributions are:

- The concept of visual bridges and a taxonomy of such substructures.
- A procedural model for generating impossible structures that utilize visual bridges as starting points.
- A novel algorithm to find randomized pathways between arbitrary components for procedural grammars in 3D space.
- Scoring functions for characterizing visually pleasing impossible structures.

Our code can be found at this link.

2. Related Works

Prior research has investigated impossible structures in both 2D and 3D space. However, all these methods are goal-directed, requiring complicated user input to model specific impossible structures. Additionally, these methods only demonstrate results on simple structures. These two limitations confine the utility of these systems for designers with limited specialized knowledge aiming to produce complicated results akin to scenes in the game Monument Valley [Gam]. Our approach builds on some of the concepts from prior research while adopting a novel strategy to address these challenges.

Modeling and Rendering Impossible Structures: Multiple existing works categorize impossible structures as a subset of view-dependent structures [Rad99], and contribute to methods of comprehending, modeling, and rendering these structures. In the initial stages, research predominantly centered around classification efforts [Cow77, Sug07] and finding the sources of impossibility, such as misinterpretations of depth, background, or surfaces [Ter80, Kul83, Uri01]. Later methods introduce tool sets to help users combine 3D components or transform an existing 3D object to be viewed as an impossible structure in a 2D domain. These approaches typically rely on deforming components, applying linear transformations, or modifying the facades of the input 3D component to achieve the intended outcome [KK01,SDG03,OF08,Elb11, SRC20]. Subsequent approaches simplify inputs to 2D points or normal maps while also improving computation efficiency through advanced optimization algorithms [Sug07, WFY*10, LYY*16]. However, even in these instances, users need to have a clear understanding of the desired connectivity and parallelism of the structure. Our work builds upon these previous concepts by identifying depth misconceptions as a cause of impossibility. But in contrast, our goal is to assist non-expert users in exploring a range of possible impossible structures, without understanding the underlying factors that lead to structural impossibility.

Our approach, inspired by [Ern86,Ern06,LYY*16,SRC20], employs distinct components to create the impossible structures. The concept of the 'shifting gap' from [LYY*16] has been a significant influence in shaping our methodology. However, unlike Lai's method, which dynamically alters an existing structure to hide its disjointed parts from the observer, our technique starts with these independent components as the starting point of our procedural design.

Prior research has predominantly showcased relatively simple results with a single cause of impossibility. This single cause of impossibility can be used to categorize structures into different types (refer to Appendix A for a classification of impossible structures). In contrast, our exploratory approach allows the integration of multiple sources and types of impossibility.

To make impossible structures visually "impossible", it is also important to consider the colors applied to them. Tsuruno proposes the method Mimetic Surface Color and Texture Adjustment (MSCTA), which is designed to produce naturally shaded and appropriately textured 3D impossible objects under physical light sources [Tsu15]. We use this method in visualizing our results.

Procedural Languages: Procedural modeling is widely used to create intricate and detailed 3D objects based on specified grammar rules [MP01,WWSR03,MWH*06,LWW08]. Shape grammars such as CGA [MWH*06] that decompose shapes into other shapes are especially well-suited for generating complex architectural designs. However, shape grammars are not suitable for generating impossible structures that start from visual bridges. This is because visual bridges appear connected in 2D space, so subdividing the 3D space between them (which can be substantial) would result in a clutter of structures hidden behind the visual bridges when projected back into 2D space. In contrast, our procedural language is more similar to L-Systems [PL90], designed for growth processes to explore the space. However, our language differs from L-Systems in terms of termination and branching.

Creating cycles in 2D space within the context of procedural modeling is a challenging task. While there are limited works addressing this issue, one notable approach relies on a top-down decomposition of shape grammars to construct interconnected struc-

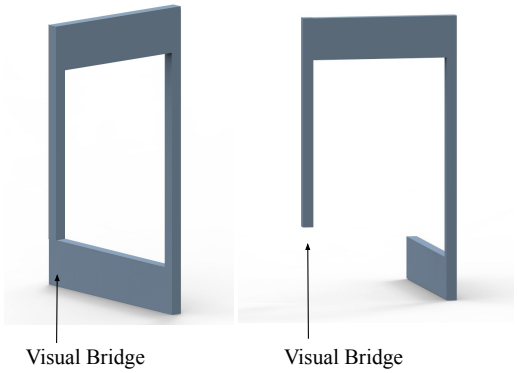

Figure 2: *When the two components of the structure's visual bridge appear to be connected in screen space, the whole structure gives the impression of being impossible (left figure). Otherwise, it would not demonstrate impossibility (right figure).*

tures in 3D space [KK11]. This method starts by drawing a "box" between the target connecting points and then subdividing the space using shape grammars like CGA. However, our procedural language differs significantly from the CGA grammar, and we need to take into account how the components are projected in 2D space during the connecting phase, which is not considered in this method. Another recent approach has employed graph grammars [Roz99] to create cycles in 2D space [Mer23]. This method starts with an existing cycle and performs string replacements at each step. However, our procedural language does not perform string replacements. Furthermore, we start with visual bridges which are separate pieces of components in 3D space instead of a single string. Consequently, we propose a novel approach to generate cycles within procedural modeling.

Guided Procedural Modeling: There's a common desire to generate outcomes that adhere to specific constraints within the context of procedural modeling [MP96, PL90, RMGH15, TLL*11]. Many approaches have been designed to give users control over the process and generate more predictable results [BvmM11, RLGH15]. In our work, we use Sequential Monte Carlo (SMC) [DDFG01], a sampling method to control the output of procedural models by viewing each model as a sample from a probability distribution [SG92].

3. Approach

The process of creating impossible structures begins with understanding the characteristics that define them. Humans have an innate ability to instantly perceive three-dimensional structures from two-dimensional drawings [WFY*10]. However, this ability can pose challenges when encountering impossible structures. Although viewers can easily perceive the localized three-dimensional structure of individual components within the drawing, attempting to view the drawing as a cohesive whole unveils structural inconsistency. This will prompt viewers to realize that the figure is not physically possible.

Our observation is that the key element making this kind of structure appear impossible is the *visual bridges*. A visual bridge contains two separated components carefully positioned to give the impression of connection in the screen space (Figure 2). As viewers are deceived to believe the two components are connected in 3D space, the existence of substructures that connect the two components of the visual bridge in 3D space can subsequently challenge this perception and confuse the viewer.

Our system employs visual bridges as a fundamental entity. It then creates a cycle in 2D space that makes the components of visual bridges appear to be both connected and separated at the same time. The illusion of impossibility arises from the arrangement and combination of all the components that constitute the 2D cycle, regardless of how each component appears individually. Based on this observation, our method (Figure 3) initially generates a proxy of the impossible structure using cuboids. Subsequently, we subdivide and embellish the spaces within these cuboids to enhance the visual appeal of the final result. This design can both simplify our computation and allow artists to apply different decorations to the same proxy to create results satisfying their needs (we show examples of using different decorations on the same proxies in Figure 23).

In the following sections, we start by formally defining the concept of visual bridges (Section 4). Then we describe the procedure for choosing visual bridges as starting points (Section 5.1), performing a random derivation to introduce variation (Section 5.2), and finding a pathway to connect the visual bridges (Section 5.3). After that, we introduce methods to decorate the results (Section 5.4) and a set of scoring functions used to evaluate these structures during generation (Section 6). Additionally, we will explain our use of Sequential Monte Carlo (SMC) inference to sample high-scoring structures (Section 7). Finally, we will demonstrate the interactive system we developed on top of this algorithm and present results with varying features and complexities (Section 8).

3.1. Procedural Language for Impossible Structures

We introduce a procedural language that resembles L-Systems for generating the impossible structures. Essentially, this language adds a new structure component at each execution step. Precisely, this language $\mathcal{L}$ is a tuple

$$\mathcal{L} = < M, \omega, R >$$

where M represents the alphabet, ω stands for the axiom, and R is a set of production rules. The alphabet contains parameterized modules denoted as $M = \{A(P), B(P), \ldots\}$. P are parameters such as translation, rotation, and scaling. In our system, each instance of a module is represented as a cuboid, and we use continuous parameters for the transformations of the modules. The axiom ω is the initial state, which constitutes the visual bridges chosen at Section 5.1. All the production rules are executed with probability and conditions that determine the modules' relationship with either other modules or the viewport. When a rule is executed, we add an instance of the module on the right-hand side of the rule to the structure, without replacing the instance of the module on the left-hand side.

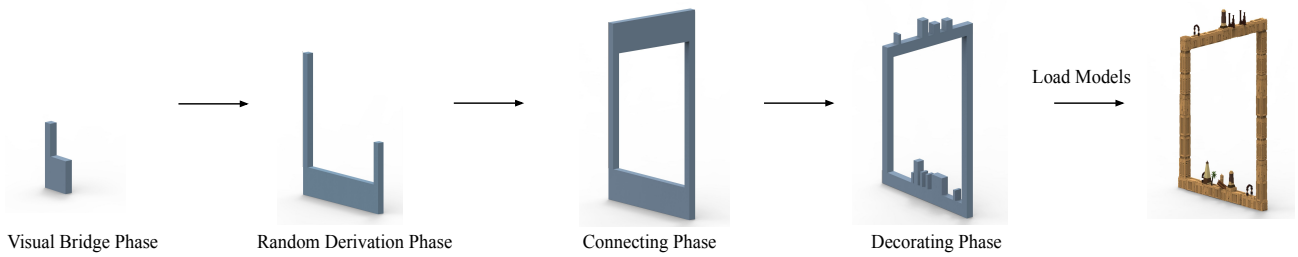

Figure 3: *Generating an impossible structure starts from the visual bridges phase (Section 5.1), and goes through the random derivation phase (Section 5.2), connecting phase (Section 5.3), and decorating phase (Section 5.4).*

The rules take the form:

$$id_1 : A(P) : condition, prob \rightarrow B(P) \in M \quad (1)$$

$$id_2 : B(P) : condition, prob \rightarrow C(P) \in M \quad (2)$$

However, our procedural language differs from an L-System in the following ways:

Termination: We omit termination symbols to avoid introducing the concepts of start and end in the cycle generated. In our language, termination is determined during execution (see Section 5.3). Furthermore, different from L-systems, not only the terminal symbols but also every symbol in our language can be converted to geometry (cuboids, in our system).

Branching: L-Systems often create new branches at each step and execute rules in parallel. But our language is much more conservative when creating new branches. While we do allow the addition of new branches, these branches must also form a cycle that creates the illusion of impossibility. Otherwise, unnecessary branches would only distract viewers from perceiving the impossibility of the structure (we further discuss branching in Section 5.2).

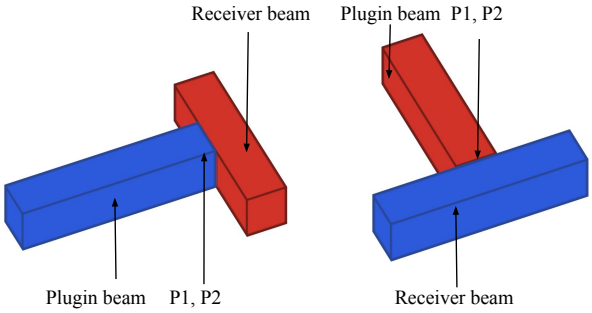

Figure 4: *We introduce terms used in this section. In both figures, the blue beam is the foreground beam and the red beam is the background beam. In the left figure, the blue beam is the plugin beam; in the right figure, the blue beam is the receiver beam. P_1 and P_2 are different points in 3D space that both project to P on screen.*

4. Visual Bridges

In this section, we formally define the concept of visual bridges and discuss their taxonomy. We define a visual bridge as a combination of two distinct components that give the appearance of being connected in the screen space. This concept of finding seemingly connected components in 2D space to create an illusion of impossibility has previously been explored in several studies, such [Ern86, Ern06, LYY*16, SRC20]. However, our approach is different from these methods which focus on high-level concepts (such as the 'shifting gap' in [LYY*16]). We operate on low-level geometries, the visual bridges, to create the impossible structures. We present a detailed taxonomy of visual bridges that has not been seen in previous works. In this section, we will refer to the two components of the visual bridges as "beams," as we represent all components as cuboids. We will further explain how to choose visual bridges and incorporate them into our generation procedural in Section 5.1.

We use $P \in \mathcal{R}^2$ to denote the point where the two components connect on the screen space, and $P_1, P_2 \in \mathcal{R}^3$ as two distinct points in 3D space that both project to P. We let P_1 be the point closer to the camera and we call the beam growing from P_1 as the **foreground beam**, colored in blue. We call the beam growing from P_2 as the **background beam**, colored in red in the following discussion. We call the beams plugged into the other beam the **plugin beam**, and the beam being plugged into as the **receiver beam**. Both the foreground (blue) beam and the background (red) beam can be either the plugin beam or the receiver beam. Figure 4 provides a visualization of these definitions.

In addition to both of their starting points projecting to the same point P on the screen, there is an additional condition for two beams to appear connected: the connecting point of the foreground beam, labeled as P_1, must remain hidden from the viewer (Figure 5). Our insight is that the viewer should not directly perceive the physical connection between the foreground and background beams in 3D space (which does not exist). Instead, the viewer is encouraged to imagine a connection that isn't physically present. In the following subsections, we first discuss the scenarios where two beams extend in the same or different directions and then discuss the allowed rotations for visual bridges.

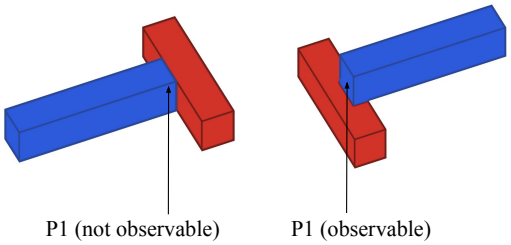

Figure 5: *Hiding the starting point (P_1) of the foreground beam (blue) from the viewer can make the two beams appear connected. In the left figure, P_1 is not observable and the two beams appear to be connected. In the right figure, P_1 is observable and the beams appear not to be connected.*

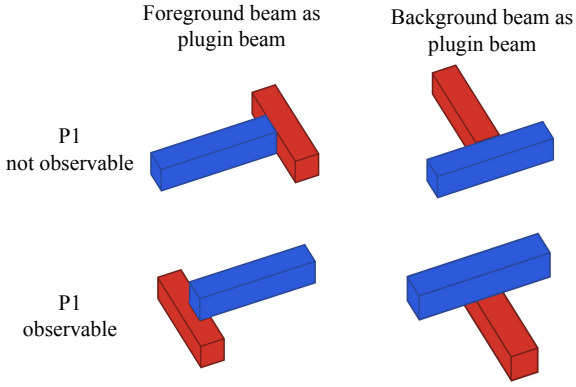

Figure 6: *We show the four combinations of two beams extending in different directions. We categorize the combinations based on which beam is the plugin beam and whether P_1 can be observed. In cases when P_1 cannot be observed, the two beams appear to be connected in 2D space by nature.*

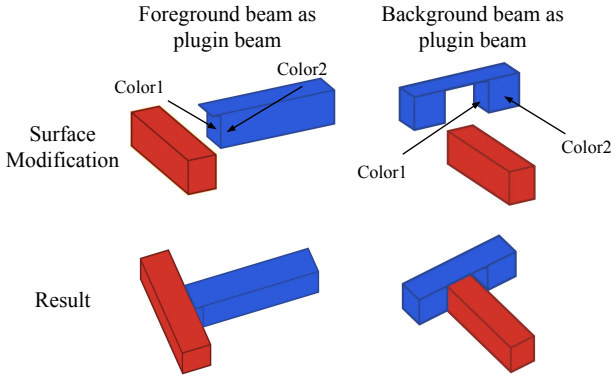

Figure 7: *Surface modification on cases when P_1 is observable. [ADD: We highlight the contour for clear visibility of component edges. The contour is omitted in all our final results.] The first row shows the modification, and the second row shows the result. We show both cases when the blue beam is the plugin beam (left column) and cases when the red beam is the plugin beam (right column).*

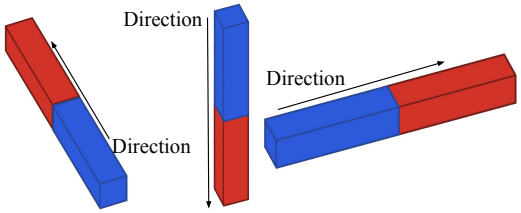

Figure 8: *When two beams extend in the same direction, there is only one type of visual bridge available.*

4.1. Beams Extending in Different Directions

Given two beams extending in different directions, we can identify four distinct combinations of the two beams, as shown in Figure 6. We differentiate them based on two characteristics: whether P_1 is observable and which beam (red or blue) is the plugin beam. In instances where P_1 is not observable, the two beams appear inherently connected when properly positioned. Conversely, for cases where P_1 is observable, we implement surface modifications as detailed below.

Our modification centers around applying the Mimetic Surface Color and Texture Adjustment (MSCTA) algorithm [Tsu15]. This algorithm involves altering the colors of surfaces of varied normal to create the illusion of uniform coloration. In our implementation, we carve out the portion of the foreground (blue) beam that contains point P_1 and then use MSCTA to adjust the color of the carved surface (color1) to match the color of the outer surface (color2) of the beam. This strategy deceives viewers into perceiving both surfaces as identical, thereby concealing the fact that P_1 has been removed (Figure 7).

4.2. Beams Extending in the Same Direction

When both the foreground (blue) beam and the background (red) beam extend in the same direction, there is only one way to hide P_1 from the viewer, resulting in only one type of visual bridge. In Figure 8, we provide some examples of such cases.

4.3. Allowable Beam Rotations

We notice that in instances where P_1 is not observable, we can rotate the plugin beam as long as this rotation does not render P_1 observable. The rotation should occur along directions perpendicular to the receiver beam to ensure that the plugin-receiver status remains unchanged. The rotation is limited to a range of $-\frac{\pi}{2}$ to $\frac{\pi}{2}$ in radians to prevent alignment of the plugin beam with the receiver beam (when rotation = $\frac{\pi}{2}$) or to transform the existing visual bridge into a different type (when rotation $> \frac{\pi}{2}$ or rotation $< \frac{\pi}{2}$). Figure 9 provides visual examples of such rotations.

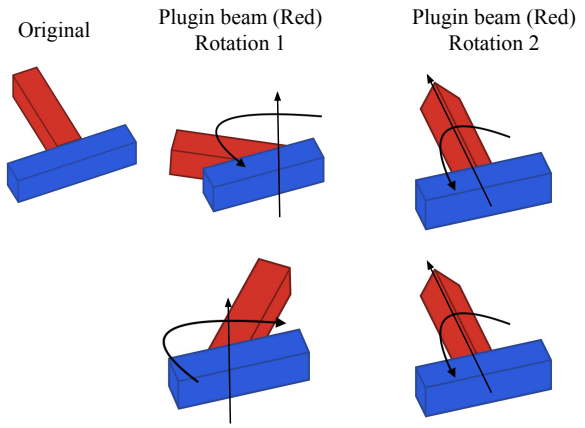

Figure 9: *Potential rotations of the plugin beam (red). The straight arrows serve as indicators of the rotation axes, and the curved arrows show the rotational movement.*

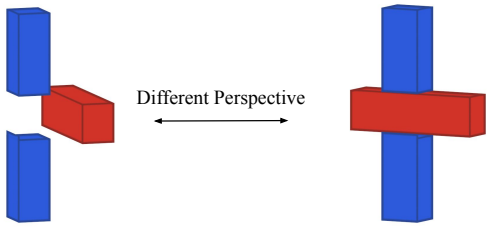

Figure 10: *This method combines two visual bridges by sharing a common component. We present the method from two perspectives. We highlight the contour for clear visibility of component edges. The contour is omitted in all our final results.*

5. Generating Impossible Structures

The preceding section discusses the taxonomy of visual bridges, and this section delves into the process of generating complete impossible structures from the visual bridges. As illustrated in Figure 3, we go through the visual bridge phase, the random derivation phase, and the connecting phase to create a structure that gives the impression of impossibility. After that, we use the decorating phase to make our results visually pleasing.

5.1. Visual Bridge Phase

The visual bridge phase is the first step of the generation process (Figure 3). It provides two procedural components for the entire structure to grow from. We have defined visual bridges as beams in the previous section. In this subsection, we make them procedural components and assign parameters to them.

We start by randomly selecting a point P within a radius r centered on the viewport. Then, we determine two distinct points in world space, P_1 and P_2, both of which project to P on the screen. In our implementation, we use an orthographic camera positioned at coordinates $(5,5,5)$ in world space, with an up vector of $(0,1,0)$ and a look-at point at $(0,0,0)$, which creates an isometric view. This choice can be arbitrary and depends on the desired perspective. Our viewport is an 800×800 square, and we select $r = 100$ to keep P near the center of the screen. We choose P_1 randomly at a distance between 2 and 3 units from the screen and find P_2 based user's target complexity of the structure (see Section 8). Generally, P_2 is positioned 2 to 5 units away from P_1, and increasing this distance results in more complex structures.

The next step involves assigning module types to the components of the visual bridges. We observed that there must be a valid production rule between these components so that when the two components appear connected on the screen, their connection appears appropriate. In our implementation, our grammar provides various combinations of visual bridge types, and our system randomly selects from these options.

Multiple Visual Bridges We also permit the presence of multiple visual bridges within a single structure. In most cases, the system establishes connections between components from different visual bridges as further introduced in Section 5.3. A special scenario occurs when we have two visual bridges with the same module types and directions. In this case, we can combine them to share a common background component as shown in Figure 10. These special visual bridges can result in impossible structures with depth interposition, exampled by the results in Figure 19.

5.2. Random Derivation Phase

The random derivation phase begins with two disconnected components obtained from the visual bridge phase and starts a random derivation of the grammar from both of them to introduce variety as illustrated in Figure 3. The system executes the procedural grammar for a random number of steps that would satisfy the user's desired complexity (see Section 8) and ultimately outputs two sequences of connected procedural components.

We allow the creation of new branches during the random derivation phases but create them with caution. For each new branch introduced, it must form a new cycle. Otherwise, unnecessary branches would only divert viewers' attention away from perceiving the impossibility of the structure. When new branches are created, we randomly pair the new branches at the end of the random derivation phase and establish connections between paired branches in the connecting phase. It is important to note that this pairing may not always be one-to-one. As the grammar executes randomly, it may result in different numbers of branches starting from the foreground and background components. We provide an example of an impossible structure with two branches and its generation process in Figure 22.

5.3. Connecting Phase

The connecting phase is the third step in the process outlined in Figure 3. This phase starts with two disconnected sequences of components and outputs an unadorned impossible structure, which we call a "proxy". We first identify two ending components c_A and c_B

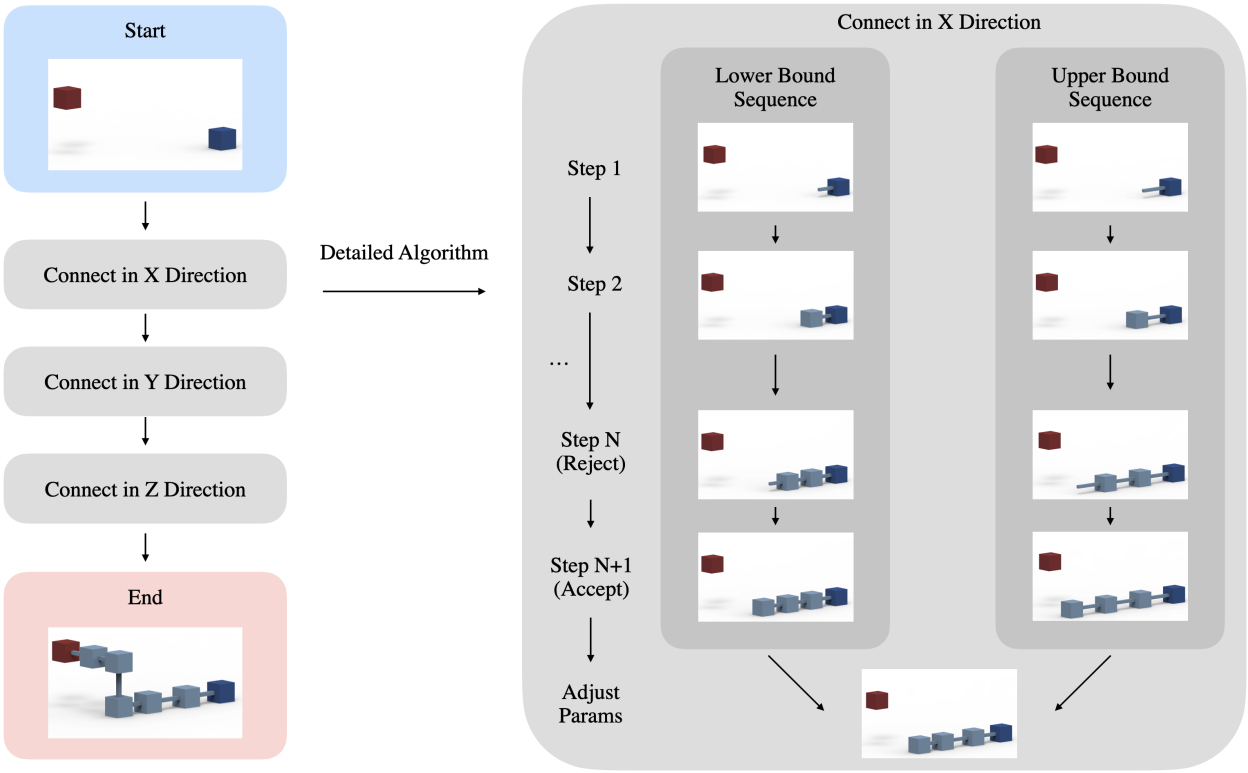

Figure 11: *The procedure for connecting two components in 3D space. In this simplified context, we use a basic grammar comprising two elements: cubes and cuboids, which are governed by two rules - cuboids derive from cubes, and cubes derive from cuboids. The blue cube represents the starting component, while the red cube represents the ending component. In the example, we establish a connection along the X, Y, and Z axes sequentially. We show the detail of connecting in the X direction on the right part of the figure. During the execution, we track both the sum of the lower and upper bound values of all the components along the X-axis in the lower and upper bound sequences. The process continues until the distance between the starting and ending component along the X-axis (denoted as δ_{d_x}) is between the upper and lower sum. Then we check if there exists a production rule for the last object in the current sequence to extend in the Y-axis, which is the next direction to consider. In the provided example, we reject the sequence at step N because the last component (the cuboid) lacks a production rule in the Y-axis. Then we continue to execute the grammar to add a new component and accept the sequence at step N+1 as the newly added component(the cube) has a production rule in the Y-axis. Subsequently, we adjust the parameters of the accepted sequence to precisely match δ_{d_x}. After completing this process for the X-axis, we move on to the other two directions. We obtained our final result when we finished the search for all three directions.*

within the two given sequences and execute our connecting algorithm (Figure 11) to determine a pathway c_1, c_2, ..., c_k that starts from c_A and terminates at c_B. Moreover, there must exist a transition rule from c_A to c_1, c_i to c_{i+1} $\forall\, i < k$, and c_k to c_B.

We denote the distance between c_A and c_B as δ. Our approach tackles this problem separately in three different axes: X, Y, and Z. We denote the direction along an axis as $\hat{d}$, and the distance from c_A to c_B along an axis as $\delta_{\hat{d}}$. Our goal is to find a sequence of components whose combined size in $\hat{d}$ equals $\delta_{\hat{d}}$. Additionally, the last component of this sequence must have a production rule leading to the next direction (or to c_B if we are solving for the last axis). We randomize the order of applying our algorithm along axes X, Y, and Z, and find solutions for each axis sequentially.

When solving for a direction $\hat{d}$, our method prioritizes rules extending in $\hat{d}$. This allows us to efficiently find a solution for $\hat{d}$ without interfering with searches in other directions. However, we do allow the execution of production rules not in $\hat{d}$ when no suitable rules are available (discussed further in Appendix B). Our method involves two major steps. Firstly, we identify a sequence of components whose sum of scales in $\hat{d}$ is close to $\delta_{\hat{d}}$. To do so, we track the lower and upper bounds of the sum of scales of the components along the $\hat{d}$ axis. A potential sequence is identified if $\delta_{\hat{d}}$ falls within this range (proof can be found in Appendix C). Subsequently, we go through an iterative process to gradually adjust the scales of each component, approaching the exact value of $\delta_{\hat{d}}$.

The advantage of this approach is that it does not rigidly constrain the exact number of components. If a sequence encounters difficulties in identifying a suitable production rule for the next direction, introducing a new component could potentially resolve the

problem while still keeping the sequence within the defined scale range. For a more detailed algorithm, please refer to Appendix B. We also provide a validation of our algorithm in Appendix D.

When multiple visual bridges are present within a single structure, we establish connections (method see Section 5.3) between components from different visual bridges in 3D space until only two components remain unconnected. These two remaining components serve as the starting components for the random derivation phase.

5.4. Decorating Phase

The previous subsections focus on generating complete impossible structures, but the results are plain and have simple geometries. In this subsection, we introduce methods to enhance the aesthetics of our results by either subdividing the space within our structures or employing text-guided neural networks. This discussion corresponds to the decorating phase depicted in Figure 3.

Decorations with Procedural Languages: One approach to decorating the impossible structures is to subdivide the space inside it for more complicated designs. The impossibility of the structure is determined by the arrangement of the components, so subdividing spaces inside individual components does not compromise the fundamental impossibility.

Our system can execute various procedural grammars within the components, as long as the outputs are contained in the scope of the components. In our results, we demonstrate structures decorated using CGA (Figure 15). We also execute our own procedural grammar within the components, resulting in structures where smaller impossible structures are nested within larger ones (Figure 17).

Decorations with ControlNet: We also leverage the property that impossible structures can only be perceived as impossible from a specific perspective. This enables us to decorate results in 2D space. In our demonstration, we input 2D images of our results into text control ControlNet as input [ZA23] (model [Zha23]) for further decoration. We showcase both decorations applied to the proxies from the connecting phase and applied to results that have already been decorated using procedural languages (Figure 21).

6. Scoring Functions

The previous sections describe a method for generating complete impossible structures. However, not all of these structures are equally desirable. In this section, we describe a set of scoring functions to identify the most promising structures during the generation process. The scoring function has two main components: structural integrity score and visual harmony score. The structural integrity score evaluates how a newly added component c_i might disrupt the previous structure on the screen space and 3D space (Figure 12). The visual harmony score assesses the overall aesthetics of the structure (Figure 13).

6.1. Structural Integrity score

Occlusion Score: We use occlusion score to ensure that the new component c_i does not obscure essential details of the existing structure. Specifically, if the visual bridges are occluded, the structure loses its inherent impossibility. However, we also noticed that the visual bridges can occupy a relatively large portion of the screen, making it challenging for the newly added components to avoid occlusion entirely. To address this, we assign greater importance to pixels near the 2D position $P(x_p, y_p)$(the starting point of the visual bridge, discussed in Section 4). We use $\mathcal{P}^{c_i}$ to denote the set of pixels covered by c_i (which is the newly added component) and $\mathcal{S}^{oc}$ to represent the occlusion score. We use a Gaussian Function to measure $\mathcal{S}^{oc}$ as shown below:

$$\mathcal{S}^{oc}(c_i) = \alpha \cdot |\mathcal{P}^{c_i}| - \sum_{(x,y)\in\mathcal{P}^{c_i}} exp(-((x-x_p)^2 + (y-y_p)^2))$$

(α is a positive constant that controls the contribution of number of pixels occupied by c_i. In our implementation, we choose α=0.2)

Overlapping Score: We use the overlapping score to ensure that two components do not overlap each other in the 3D space. We use $\mathcal{V}^s$ to denote the 3D space occupied by the existing structure and $\mathcal{V}^{c_i}$ for the space occupied by c_i. We use $\mathcal{S}^{ov}$ to denote the overlapping score.

$$\mathcal{S}^{ov}(c_i) = \begin{cases} 1, & \mathcal{V}^s \bigcap \mathcal{V}^{c_i} = \varnothing \\ 0, & \text{otherwise} \end{cases}$$

Protruding Score: We want to ensure that the c_i does not extend beyond the viewport. We use $\mathcal{P}^{c_i}$ to denote the set of pixels covered by the c_i in screen space and $\mathcal{P}$ to denote all the pixels covered by the viewport. We use $\mathcal{S}^{pr}$ to denote the protruding score.

$$\mathcal{S}^{pr}(c_i) = \begin{cases} 1, & \mathcal{P}^{c_i} \bigcap \mathcal{P} = \mathcal{P} \\ 0, & \text{otherwise} \end{cases}$$

a) Occlusion Score b) Overlapping Score c) Protruding Score

Figure 12: *We show features (in red rectangles) that lead to low integrity scores. Figure a) shows a low occlusion score as the visual bridge is obscured by the foreground structures. Figure b) shows a low overlapping score as the components on the left bottom overlap in 3D space. Figure c) shows a low protruding score as the component on the upper left corner extends beyond the viewport.*

6.2. Visual Harmony Score

The visual harmony score evaluates the aesthetic aspects of the structure (demonstrated in Figure 13). It prevents the structures from leaning towards "extremes," such as having all the procedural components of the same type or crowding all the components into a small space.

a)Probabilistic Consistency Score

b)Density Score

Figure 13: *The importance of the visual harmony score is introduced in this section. Figure a) shows an "extreme" structure with a low probabilistic consistency score. All the components in Figure a) have the same type making the structure uninteresting. Figure b) shows a structure with a low density score clustering in the lower part of the viewport (which is the red box).*

Probabilistic Consistency Score: Our procedural grammar determines a prior probability for each rule to be executed. However, during the execution, the actual results may not strictly align with these probabilities due to factors like the connecting phase (we favor rules in a certain direction) or the Sequential Monte Carlo (SMC) sampling process (introduced in Section 7). In our approach, we prioritize results where the empirical probability doesn't deviate significantly from the initial prior probabilities.

We employ a Markov Chain [Chu60] approach to determine the probabilities of each module type based on the probabilities of the transition rules (for detailed calculations, refer to Appendix E). We represent the prior probabilities of module types as $\pi = (\pi_A, \pi_B, ...\pi_K)$, where $\sum_k \pi_k = 1$.We count the occurrences of objects for each module type present in the existing structures and represent the observed probabilities of component types as $(p_A, p_B, ...p_K)$, where $\sum_k p_k = 1$.

We use KL divergence to measure the dissimilarity between the empirical distribution and the prior distribution at each step, denoted as $\mathcal{S}^{pc}$.

$$\mathcal{S}^{pc}(c_i) = -\sum_k p_k \cdot \log(\frac{p_k}{\pi_k})$$

Density Score: We use the density score to prevent the entire structure from clustering in a small region. We measure such scores in 2D space due to the fact that components far apart in 3D space might still appear very close in 2D space. The density score not only penalizes regions that are overly crowded but also encourages the structure to expand into less populated areas.

To compute the density score, we rasterize c_i onto the screen space and identify its center in the screen space as P_O. We then iterate through a circular region centered at P_O with a radius of r to locate all objects within the neighborhood. For each rasterized object c_k within this neighborhood, we denote its center to P_O distance as d_k and the number of pixels it occupies within the neighborhood as $|\mathcal{P}^k|$. We first calculate a metric for "crowdedness" in the neighborhood, $\mathcal{D}$, by considering both the distance of the objects in screen space and the number of pixels they occupy.

$$\mathcal{D} = \sum_k \alpha \cdot \frac{1}{(d_k)^2} + \beta \cdot |\mathcal{P}^k|$$

Now we define the density score $\mathcal{S}^{de}$ as a decaying function that approaches 0 as $\mathcal{D}$ increases. This score can also become relatively large when c_i is expanding into an empty space.

$$\mathcal{S}^{de}(c_i) = exp(-\alpha' \cdot \mathcal{D} - \beta')$$

(α, β, α', and $\beta'$are positive constants. The former two can be adjusted to balance the contribution of the distance and size factors to $\mathcal{D}$, while the latter two can be adjusted to determine the rate of decay and resulting value when the space is empty. In our implementation, we set $\alpha = 0.2$, $\beta = 0.8$, $\alpha' = 0.5$, and $\beta' = 1.5$.)

6.3. Final Score Computation

We have defined several scoring functions to evaluate different aspects of the impossible structure. Now we show how to combine them to compute a function F that evaluates the effect of the new component c_i. We perform an incrementalized score computation by tracking which components of the score need to be updated in response to a structural change.

The base case of the recurrence, $F(c_0)$, occurs prior to the random derivation phase and after the visual bridges are chosen. In this scenario, $F(c_0) = \mathcal{S}^{pc}(c_0)$ since no components are occluding, overlapping, protruding, or leading to high density. After that, at each step when a new component c_i is added, we update the effect of c_i in terms of occlusion, overlap, protrusion, and density. We also need to check the value of $\mathcal{S}^{pr}(c_i)$ separately each time. At each state, the updated result $\delta F_(c_i)$is:

$$\delta F_(c_i) = \mathcal{S}^{pc}(c_i) - \mathcal{S}^{pc}(c_{i-1}) + \mathcal{S}^{oc}(c_i) + \mathcal{S}^{de}(c_i)$$

Therefore, we have the following relation:

$$F(c_i) = (F(c_{i-1}) + \delta\mathcal{F}_(c_i)) \cdot \mathcal{S}^{ov}(c_i) \cdot \mathcal{S}^{pr}(c_i)$$

7. Inference

Our system uses the scoring functions introduced in Section 6 to search the output space of the procedural model. In particular, we use the Sequential Monte Carlo (SMC) method to focus the search on promising structures. SMC is a method to approximate Bayesian inference: given a prior $p(x)$ and a likelihood $p(y|x)$, it produces samples from the posterior distribution $p(x|y)$. It does this by producing a sequence of distributions $p_0 \ldots p_n$, such that $p_n \approx p(x|y)$. Each distribution p_i is represented by a set of samples, called particles. Given p_0 drawn from system initialization, the algorithm samples an initial set of particles and weights them according to the likelihood. It then samples a new set of particles according to the weights and evolves the particles to p_1 according to the grammar execution [SG92].

In our implementation, the initial state p_0 is given by the system's random choice in the visual bridge phase. After that, in the random derivation phase, the prior is the product of the probabilities of all random choices made during the procedural model. The likelihood is the score function F. Initial particles are sampled by

generating random types of visual bridges. The proposal function executes one more step when the procedural model adds a new component to the structure (please refer to Appendix F for more details).

In the connecting phase, we make modifications to the process. We recognize that the connecting phase is inherently more deterministic than the random derivation phase, as the system tends to favor selecting rules in the target direction instead of making a random choice. During this phase, we continue to evaluate $F(c_i)$ for the newly added component at each step. However, instead of resampling particles, we only remove the particles with $F(c_i)$ smaller than a threshold θ.

During program execution, the likelihood of finding desirable results increases with a larger number of particles. However, a larger number of particles also leads to higher time consumption. We have noticed that this is particularly relevant in cases where the components of visual bridges are distant from each other or when there are a significant number of steps in the random derivation phase. In such scenarios, using more particles becomes necessary to achieve satisfactory outcomes.

Through experimentation, we found that using 1000 SMC particles is adequate for generating satisfactory results for structures containing fewer than 40 random walk steps and 6 units' distance between visual bridge components (which takes about 50-70 connecting steps). This entire process typically takes less than 5 minutes on a 2019 MacBook Pro equipped with an Intel i7 CPU. Our implementation is single-threaded, and further reductions in runtime may be achieved by parallelizing the score computation for different particles. More timing results on structures with varying complexity can be found in Appendix G.

8. Results and User Interactions

Our system can generate a wide range of results, ranging from fundamental and well-known impossible structures (Figure 14) to more intricate and complex ones. We design the system to assist users with limited knowledge of impossible structures in exploring a wide variety of results, and we demonstrate six groups of results showing different features:

- **Desolated Ruins:** In this scenario, our system generates structures depicting abandoned and desolated ruins. This scenario showcases the basic capabilities of our system (Figure 15 and Figure 16).
- **Modern Factories:** In this scenario, our system generates structures depicting modern factories utilizing energy produced from a large sphere's energy synthesis. While the factory itself constitutes an impossible structure, the "energy synthesis" inside the large sphere also forms another impossible structure. This scenario showcases our system's capability to incorporate curves and spheres, as well as execute our language within structural components (Figure 17).
- **Roses and Branches:** In this scenario, our system generates structures depicting roses and plants growing on withered tree branches. This scenario showcases our system's capacity to generate non-axis-aligned impossible structures (Figure 18).
- **Temples:** In this scenario, our system generates structures depicting temples. These temple structures exhibit depth interposition, and their stairs incorporate the characteristic of normal disappearance. This scenario showcases our system's capacity to combine various types of impossibility within a single structure (Figure 19).
- **Dimensional Collapse:** In this scenario, our system generates structures that contain substructures in both 2D and 3D space. We project a portion of the resulting structure, excluding the visual bridges, onto 2D planes. Despite this projection, the resulting structure maintains its sense of impossibility (Figure 20).
- **Guided Impossible Structure:** In this scenario, our system generates a structure featuring the character "EG". While our system primarily focuses on an exploratory design, we demonstrate its ability to generate goal-directed impossible structures by incorporating a scoring function to guide the random derivation phase. Additional details of this process can be found in Figure 24.

User Interface We have developed an interface to facilitate interaction with our system. Users can choose from four predefined scenarios and select a desired complexity level ranging from 1 to 12. The complexity level influences the distances between the foreground and background components of the visual bridges and the number of steps in the random derivation phase (see Appendix H). We also show the corresponding results for different complexity inputs (Figure 15). For users who want more control over the system, we provide advanced options (detailed in Appendix I). The details of our UI system are illustrated in Figure 25. (Refer to the supplemental for a video of an interactive session using this interface, and this link for online demo).

Collaboration with a student artist We collaborated with a student artist from a highly-ranked design school in the US whom we found through social networks. The artist had no prior experience with procedural modeling. We provided the artist with a tutorial on using our system. Additionally, we designed a CGA grammar based on the artist's requests. Afterward, we asked the artist to design a grammar for generating impossible structures and create 3D models as needed. The artist successfully produced results as demonstrated in Figure 15 and Figure 16, and we provide the artist's feedback in Appendix J.

9. Conclusion

We present a novel approach to designing and exploring seemingly impossible structures. Our generation process begins by selecting appropriate visual bridges, followed by random derivation of the grammar and connecting the endpoints of the visual bridges. After the generation process, our system can further enhance the results through 3D space subdivision or application of diffusion models on the 2D results. Furthermore, we propose scoring functions to evaluate structures and employ the Sequential Monte Carlo (SMC) method to explore visually appealing results. We also offer a user interface for interacting with the model and showcasing various results generated by our system.

Our system is designed to assist non-expert users in exploring a wide range of impossible structures. For designers who want to transform a single structure into an impossible one, previous methods that provide tools for structure editing may be more suitable.

Furthermore, our system primarily focuses on creating impossible structures using the disconnected components trick; there are other types of impossible structures such as the 'disappearing space' that we cannot produce (discussed in Appendix A)

Additionally, while generating a variety of results using our built-in grammar is straightforward, authoring new grammar to model entirely new scenes remains a challenge. A promising direction for future research could involve enabling non-expert users to generate a wide array of real-life objects as impossible structures without requiring knowledge of the grammar. One potential approach could involve inverse procedural modeling and grammar inference from input objects, as explored in previous works such as [MM11, RJT18, GJB*20].

Lastly, our work has implications beyond the scope of impossible structures. Our method for generating impossible structures sheds light on creating other view-dependent structures. This involves initially designing components that fulfill certain criteria based on the viewer's viewpoint, followed by assembling the entire structure. Additionally, our algorithm for connecting procedural components, using grammars resembling L-Systems, may also be applied in other procedural modeling works. For instance, we demonstrate its capability to generate cycles in 3D space, a challenging problem in procedural modeling (as detailed in Appendix K).

References

[Ale08] ALEXEEV V.: Impossible world, 2008. URL: https://im-possible.info/english/. 1

[BvmM11] BENEŠ B., ŠAVA O., MĚCH R., MILLER G.: Guided procedural modeling. *Computer Graphics Forum 30*, 2 (Apr 2011), 325–334. doi:10.1111/j.1467-8659.2011.01886.x. 3

[Chu60] CHUNG K.: *Markov Chains with Stationary Transition Probabilities*. Springer Book, 1960. doi:10.1007/978-3-642-49686-8. 9

[Cow77] COWAN T.: Organizing the properties of impossible figures. *Perception 6* (1977), 41–56. 2

[DDFG01] DOUCET A., DE FREITAS N., GORDON N.: *An introduction to sequential monte carlo methods*. Springer Book, 2001. doi:10.1007/978-1-4757-3437-9_1. 3

[Elb11] ELBER G.: Modeling (seemingly) impossible models. *Computer and Graphics 35*, 3 (Jun 2011), 632–638. doi:/10.1016/j.cag.2011.03.015. 2

[Ern86] ERNST B.: *Adventures with Impossible Figures*. Parkwest Pubns; 1st ed. edition, 1986. 2, 4

[Ern06] ERNST B.: *Impossible Worlds: 2 in 1 Adventures with Impossible Objects*. TASCHEN, 2006. 2, 4

[Fou] FOUNDATION E.: M.c escher, the official website. URL: https://mcescher.com/. 1

[Gam] GAME M.: Monument valley panoramic collection. URL: https://www.monumentvalleygame.com/mvpc/. 1, 2

[GJB*20] GUO J., JIANG H., BENES B., DEUSSEN O., ZHANG X., LISCHINSKI D., HUANG H.: Inverse procedural modeling of branching structures by inferring l-systems. *ACM TOG 39*, 5 (Jun 2020), Article 155 : 1–13. doi:/10.1145/3394105. 11

[KK01] KHOH C., KOVESI P.: Animating impossible objects. *Leonardo 34*, 3 (Jun 2001), 197–198. 2

[KK11] KRECKLAU L., KOBBELT L.: Procedural modeling of interconnected structures. *Computer Graphics Forum 30*, 2 (Apr 2011), 335–344. doi:/10.1111/j.1467-8659.2011.01864.x. 3

[Kul83] KULPA Z.: Are impossible figure possible. *Signal Processing 5* (1983), 201–220. 2, 17

[LWW08] LIPP M., WONKA P., WIMMER M.: Interactive visual editing of grammars for procedural architecture. *ACM SIGGRAPH Proceedings* (Aug 2008), Article 102 : 1–10. doi:/10.1145/1399504.1360701. 2

[LYY*16] LAI C., YEUNG S., YAN X., FU C., TANG C.: 3d navigation on impossible figures via dynamically reconfigurable maze. *IEEE Transactions on Visualization and Computer Graphics 22*, 10 (Dec 2016). doi:/10.1109/TVCG.2015.2507584. 2, 4

[Mer23] MERRELL P.: Example-based procedural modeling using graph grammars. *ACM TOG 42*, 4 (Jul 2023), Article 60 : 1–16. doi:/10.1145/3592119. 3

[MM11] MERRELL P., MANOCHA D.: Model synthesis: A general procedural modeling algorithm. *IEEE Transactions on Visualization and Computer Graphics 17*, 6 (Jun 2011), 715–728. doi:/10.1109/TVCG.2010.112. 11

[Mor10] MORTENSEN C.: *Inconsistent geometry. Studies in Logic*. College Publications, London, 2010. 18

[MP96] MĚCH R., PRUSINKIEWICZ P.: Visual models of plants interacting with their environment. *ACM SIGGRAPH Proceedings* (Aug 1996), 397–410. doi:/10.1145/237170.237279. 3

[MP01] MÜLLER P., PARISH Y.: Procedural modeling of cities. *ACM SIGGRAPH Proceedings* (Aug 2001), 301–308. doi:10.1145/383259.383292. 2

[MWH*06] MÜLLER P., WONKA P., HAEGLER S., ULMER A., VAN GOOL L.: Procedural modeling of buildings. *ACM TOG 25*, 3 (Jul 2006), 614–623. doi:10.1145/1141911.1141931. 2

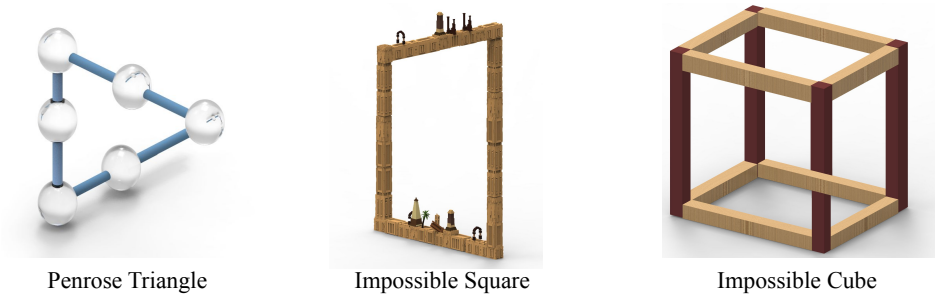

Figure 14: *Our system is capable of producing the most famous impossible structures.*

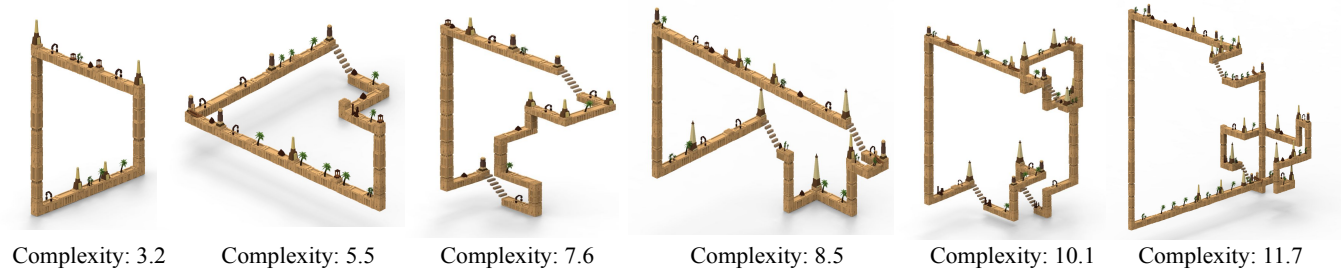

Figure 15: *We collaborated with a student artist to generate impossible structures featuring abandoned and desolated ruins. The artist designed a grammar for generating impossible structures using our system and crafted the 3D models accordingly. We also developed a CGA grammar to decorate the structures according to the artist's specifications.*

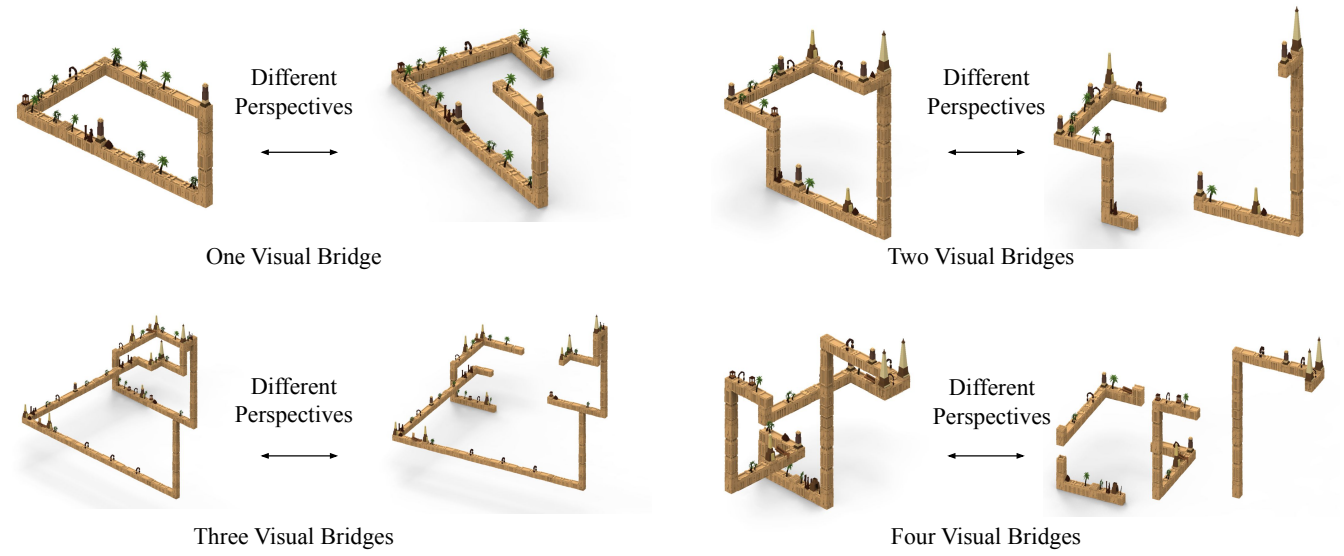

Figure 16: *Results obtained using different numbers of visual bridges and present views of the structures from different perspectives. In scenarios where multiple visual bridges are involved, our system starts with connecting sub-components of these visual bridges as explained in Section 5.3. This can lead to the structure being divided into distinct sections, contingent on which the components are connected. For example, in the "two visual bridges" case: our system chooses to connect the two foreground components during the visual bridge phase, and subsequently connects the two background components in the connecting phase. This generates two separate structures that, when observed from specific vantage points, appear to appear impossible.*

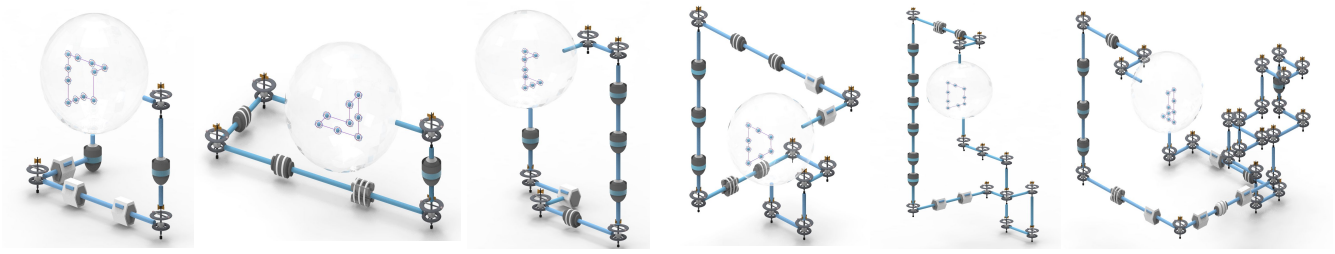

Figure 17: *Results featuring modern factories utilizing energy produced from a large sphere's energy synthesis. Both the factory itself and the "energy synthesis" inside the large sphere form impossible structures.*

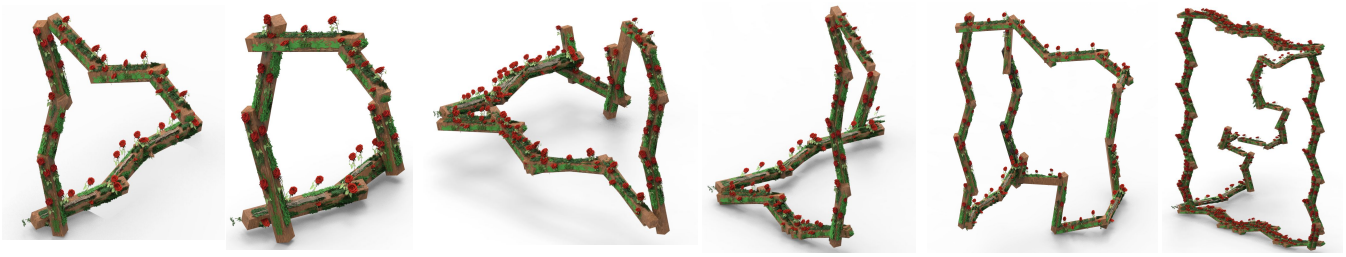

Figure 18: *Results featuring roses and plants growing on withered tree branches. This scenario showcases our system's capacity to generate non-axis-aligned impossible structures.*

Figure 19: *Results featuring temple-like structures. The structure's pillars create an illusion of depth interposition, as certain vertical pillars in the foreground appear to be connected with the horizontal beams in the background. Additionally, when viewed from the right, the temple's staircase seems to consist of three steps, yet when observed from the front, it seems to consist of two steps (the disappearing stairs are modeled based on the method introduced in [SRC20]). These results were generated without leveraging global illumination. The color adjustment algorithm (as explained in Section 4) is only applied to the diffuse reflectance model. Future works may explore the feasibility of implementing color adjustments within global illumination through differentiable rendering.*

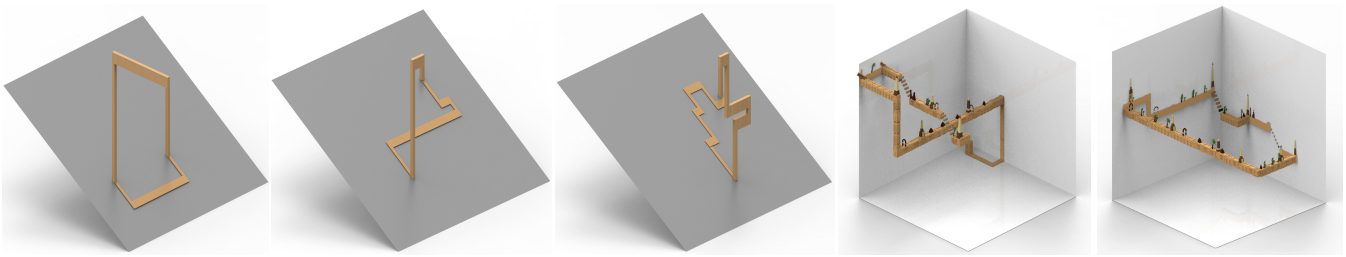

Figure 20: *Results that combine components in both 2D and 3D space. In the decorating phase, we project parts of our structure (does not include the visual bridges) onto 2D planes. Despite this projection, the resulting structure maintains its sense of impossibility.*

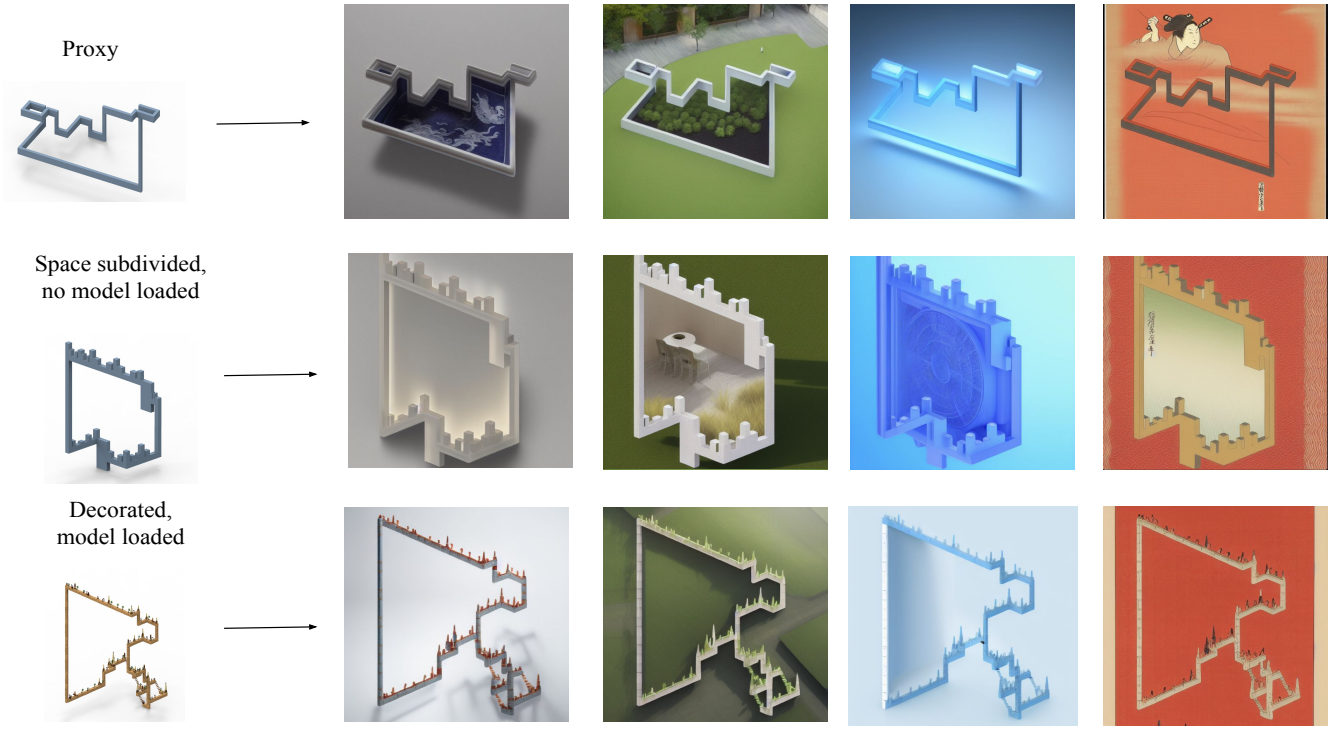

Figure 21: *Results decorated with ControlNet. We show that we can apply decorations to the 2D figure generated from the proxies, the results obtained after subdivision in the decorating phase (Section 5.4) but without loading any model, and the full results. We have chosen four different prompts. From left to right, we are using the following prompts: "porcelain or glass that looks very expensive", "organic, nature-inspired elements of the landscaping", "a high-tech, energy-efficient factory", and "Japanese Ukiyo-e style with a sunset". For each of the prompts, we also include positive prompts such as "high contrast with the background" and "high quality".*

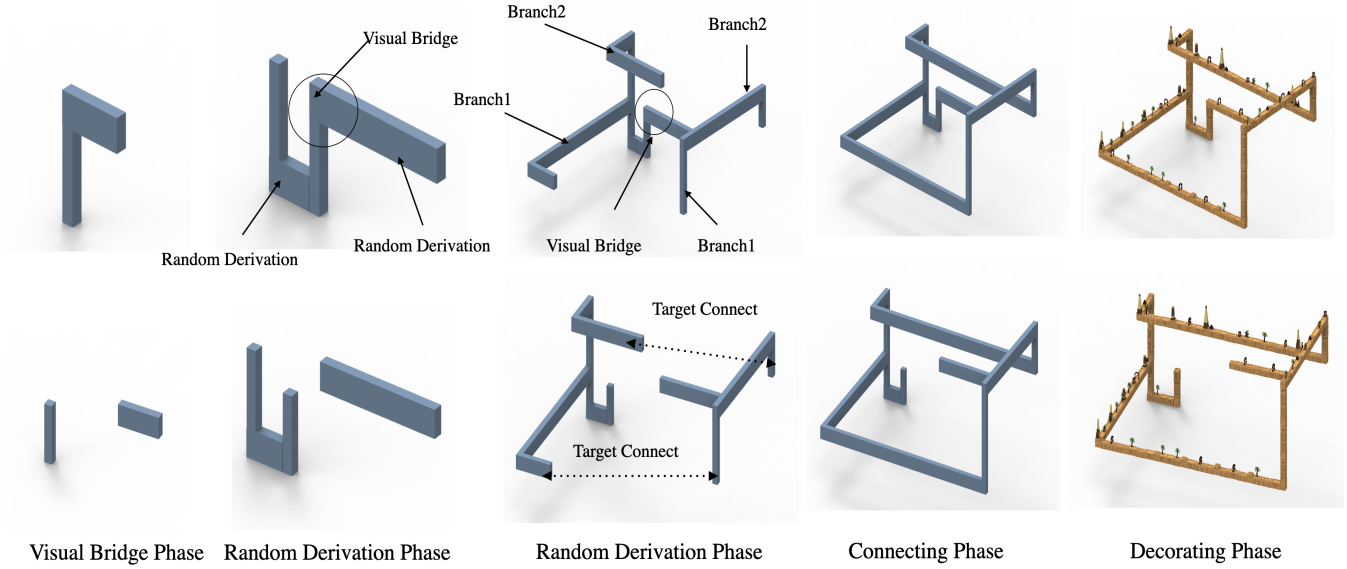

Figure 22: *The procedure for generating impossible structures with branches from two different perspectives. In our process, a new branch is introduced during the random derivation phase (third column). Subsequently, we pair the foreground and background branches and establish a connection for each of them in the connecting phase (fourth column).*

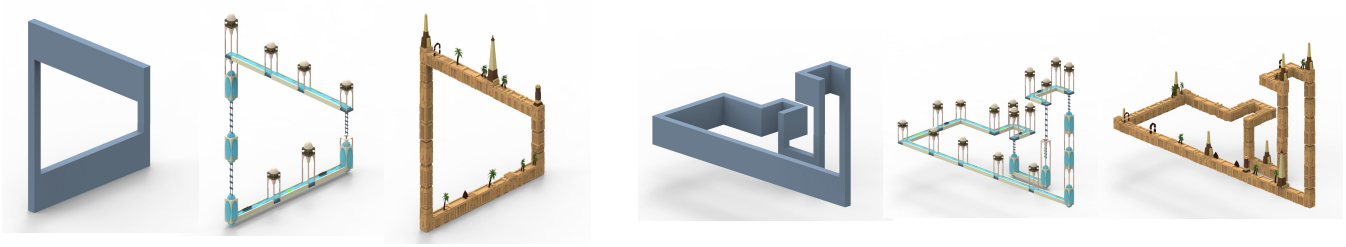

Figure 23: *Results that share the same proxy. In the two groups of figures, we apply different CGA grammars to decorate the same proxy generated. This further allows artists to produce new scenes without knowing the grammar for generating impossible structures.*

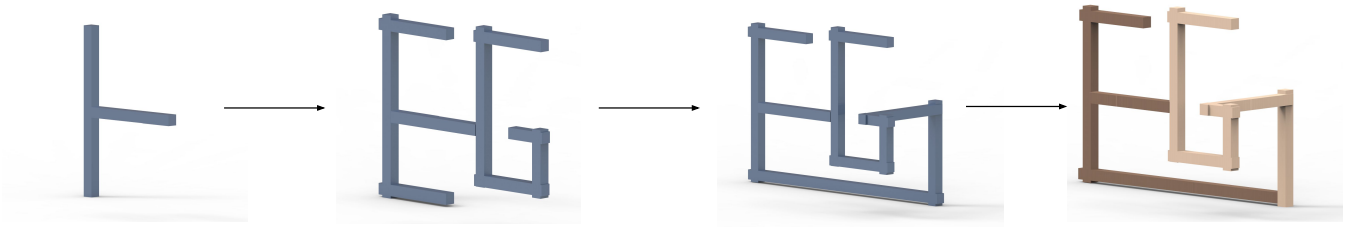

Figure 24: *An example of a goal-directed result showing "EG". To create this result, we rasterize our structure to 2D space during the random walk phase. We employ an extra scoring function that directs the structure to cover pixels forming the intended pattern. This approach is out of the scope of our initial exploratory design.*

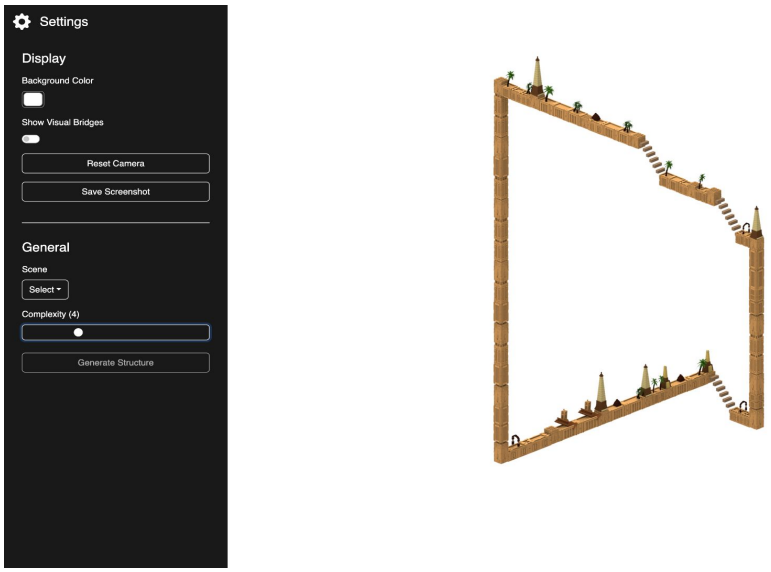

Figure 25: *Our interactive system takes a single numerical input indicating the desired complexity level of the target result (introduced in Appendix H). It assists in exploring a wide range of impossible structures. Refer to the supplemental for a video of an interactive session using this interface, and this online demo.*

[Nol10] NOLAN C.: Inception, 2010. URL: https://en.wikipedia.org/wiki/Inception. 1

[OF08] OWADA S., FUJIKI J.: Dynafusion: A modeling system for interactive impossible objects. *International Symposium on Non-photorealistic Animation and Rendering* (Jun 2008), 65–68. doi:/10.1145/1377980.1377994. 1, 2

[PL90] PRUSINKIEWICZ P., LINDENMAYER A.: *The algorithmic beauty of plants*. Springer-Verlag, Berlin, Heidelberg, 1990. 2, 3

[PP58] PENROSE L., PENROSE R.: Impossible objects: A special type of illusion. *British Journal of Psychology 59*, 1 (Feb 1958), 31–33. doi:/10.1111/j.2044-8295.1958.tb00634.x. 1

[Rad99] RADEMACHER P.: View-dependent geometry. *ACM TOG* (Jul 1999). doi:/10.1145/311535.311612. 2

[RJT18] RITCHIE D., JOBALIA S., THOMAS A.: Example-based authoring of procedural modeling programs with structural and continuous variability. *Computer Graphics Forum 37*, 2 (2018). doi:/10.1111/cgf.13371. 11

[RLGH15] RITCHIE D., LIN S., GOODMAN N. D., HANRAHAN P.: Generating design suggestions under tight constraints with gradient-based probabilistic programming. *Computer Graphics Forum 34*, 2 (Jun 2015), 515–526. doi:10.1111/cgf.12580. 3

[RMGH15] RITCHIE D., MILDENHAL B., GOODMAN N. D., HANRAHAN P.: Controlling procedural modeling programs with stochastically-ordered sequential monte carlo. *ACM TOG 34*, 4 (Jul 2015), 1–11. doi:10.1145/2766895. 3

[Roz99] ROZENBERG G.: *Handbook of Graph Grammars and Computing by Graph Transformation*. Oct 1999. doi:10.1142/4180. 3

[SDG03] SAVRANSKY G., DIMERMAN D., GOTSMAN C.: Modeling and rendering escher-like impossible scenes. *Computer Graphics Forum 18*, 2 (Nov 2003), 173–179. doi:/10.1111/1467-8659.00367. 1, 2

[SG92] SMITH A., GELFAND A.: Bayesian statistics without tears: A sampling-resampling perspective. *The American Statistician 46*, 2 (May 1992), 84–88. doi:/10.2307/2684170. 3, 9

[SRC20] SÁNCHEZ-REYES J., CHACÓN J.: How to make impossible objects possible: Anamorphic deformation of textured nurbs. *Computer Aided Geometric Design* (Feb 2020). doi:/10.1016/j.cagd.2020.101826. 1, 2, 4, 13, 17, 18

[Sug07] SUGIHARA K.: Computer-aided creation of impossible objects and impossible motions. *KyotoCGGT: International Conference on Computational Geometry and Graph Theory* (Jun 2007), 201–212. 2

[Sug20] SUGIHARA K.: Family tree of impossible objects created by optical illusions. *Bridges Conference Proceedings* (2020). 17

[Ter80] TEROUANNE E.: 'impossible' figures and interpretations of polyhedral figures. *Journal of Mathematical Psychology 24* (1980), 370–405. 2

[TLL*11] TALTON J. O., LOU Y., LESSER S., DUKE J., MĚCH R., KOLTUN V.: Metropolis procedural modeling. *ACM TOG 30*, 2 (Apr 2011), Article 11 : 1–15. doi:/10.1145/1944846.1944851. 3

[Tsu15] TSURUNO S.: Natural expression of physical models of impossible figures and motions. *International Journal of Asia Digital Art and Design* (Jan 2015). doi:/10.1111/j.2044-8295.1958.tb00634.x. 2, 5

[Uri01] URIBE D.: A set of impossible tiles. *THE THIRD INTERNATIONAL CONFERENCE MATHEMATICS DESIGN* (2001). URL: https://im-possible.info/english/articles/tiles/tiles.html. 2

[WFY*10] WU T., FU C., YEUNG S., JIA J., TANG C.: Modeling and rendering of impossible figures. *ACM TOG 29*, 2 (Apr 2010), Article 13 : 1–15. doi:/10.1145/1731047.1731051. 1, 2, 3, 17

[WWSR03] WOKNA P., WIMMER M., SILLION F., RIBARSKY W.: Instant architecture. *ACM TOG 22*, 3 (2003), 669–677. 2

[ZA23] ZHANG L., AGRAWALA M.: Adding conditional control to text-to-image diffusion models. *"arXiv preprint"* (2023). doi:/10.48550/arXiv.2302.05543. 8

[Zha23] ZHANG L.: Controlnet model - canny, 2023. URL: https://huggingface.co/lllyasviel/ControlNet-v1-1/blob/main/control_v11p_sd15_canny.pth. 8

Appendix A: Classification of Impossible Structures

Numerous prior works have tackled the classification of impossible structures [Kul83, WFY*10, Sug20]. In this section, we adopt the categorization method introduced by Reyes et al [SRC20]. and classify impossible structures into four types (Figure 26):

1. *Depth interposition:* The apparent depth ordering leads to structural inconsistency. Typical examples are the Necker cube, or a four-bar arrangement.
2. *Depth contradiction:* The propagation of local depth information leads to a structural depth contradiction. Examples are the endless Penrose stairs, or the impossible four-bar.
3. *Disappearing normals:* Apparently, we cannot assign a consistent normal across a face. In the Ernst stairs, the facet F seems horizontal along three edges, but vertical along e. The rhombic Renault logo combines a normal orientation that seems inconsistent and depth contradiction.
4. *Disappearing space:* The silhouette is not closed, as in the impossible fork, also known as devil's fork, and its cognates. If we try to follow the silhouette from the gap between two pins, we go smoothly from background to foreground.

Our approach is capable of generating all variations within categories 1, 2, and 3, along with their combinations. However, when it comes to type 4 structures involving disappearing space, our understanding is that these impossibilities require a distortion of the 3D space. As a result, they are viable exclusively within the confines of 2D space, remaining unattainable within the realm of 3D modeling. This viewpoint is consistent with the observations of Reyes et al. [SRC20] (Section 1.2).

Appendix B: Detailed Connecting Algorithm

In this section, we provide a detailed version of our connecting algorithm discussed in Section 5.3. We define the problem: given two components c_A and c_B and their position $P_A = (x,y,z)$, $P_B = (x',y',z')$ ($P_A \neq P_B$), the objective is to determine a sequence of components, represented as c_1, c_2, ..., c_k, that initiates from the P_A and culminates at P_B. Moreover, there must exist a transition rule from c_A to c_1, c_i to c_{i+1} $\forall\, i < k$, and c_k to c_B.

We denote the distance between c_A and c_B as δ. Our approach tackles this problem separately in three different axes: X, Y, AND Z. For each axis, we denote the direction from c_A to c_B along this axis as $\hat{d}$, and the distance from c_A to c_B as $\delta_{\hat{d}}$. Our goal is to find a sequence of components whose combined size in $\hat{d}$ equals $\delta_{\hat{d}}$. Additionally, the last component of this sequence must have a production rule leading to the next direction (or to c_B if we are solving for the last axis). We randomize the order of axes X, Y, AND Z, and our method finds solutions for each axis sequentially.

For the search in each direction $\hat{d}$, we break down our algorithm into two segments: discovering the sequence of components within the specified range (Algorithm 1) and adjusting the components' parameters (Algorithm 2).

This algorithm 1 finds a path connecting component c_A and component c_B along the x, y, and z directions individually. It starts by randomizing the sequence of the directions. For each direction $\hat{d}$, we use $\delta_{\hat{d}} \in \mathrm{R}$ to denote the target distance for the connection.

Algorithm 1 Finding a Sequence

```
Calculate δ = P_A - P_B, δ ∈ R^3
Random Direction Orders
for d̂ do
    SeqUpBound, SeqLowBound = 0
    while SeqLowBound < δ_d̂ do
        Find next component c_k
        Update(SeqUpBound, SeqLowBound, δ)
        if SeqLowBound > δ_d̂ then
            return False
        if c_k is valid then
            break
     Adjust Parameters
```

The algorithm prioritizes executing rules in $\hat{d}$ and accordingly adjusts SeqUpBound and SeqLowBound if such rules exist. However, when rules in $\hat{d}$ are not feasible, the algorithm adapts by selecting rules in the subsequent directions while concurrently updating the value of δ to account for the change in total target distance. The search process for a direction is concluded either if SeqLowBound surpasses $\delta_{\hat{d}}$ (indicating no available sequence is found) or when three specific conditions below are met (indicating an available sequence is found, proof see Appendix C).

- The cumulative lower bound of the scales of the components in $\hat{d}$ (SeqLowBound) is smaller than $\delta_{\hat{d}}$.
- The cumulative upper bound of the scales of the components in $\hat{d}$ (SeqUpBound) is larger than $\delta_{\hat{d}}$.
- A transition rule to proceed to the next direction (or transition rule to c_B if all directions have been processed).

Algorithm 2 Adjusting Parameters

```
Input: d̂, δ_d̂, SeqLowBound, a sequence S of components
for component c_i ∈ Sequence S do
    Assign scale of c_i as its lower bound defined by the grammar

while SeqLowBound < δ_d̂ do
    Find random component c_i from the sequence
    Denote curScale = current scale of c_i
    Assign a random scale newScale > curScale for c_i.
    Denote newSeqLowBound = SeqLowBound + (newScale -
curScale)
    if newSeqLowBound > δ_d̂ then
        newScale = δ_d̂ - SeqLowBound
        assign newScale to c_i
        break
     update SeqLowBound = newSeqLowBound
```

This algorithm 2 adjusts the parameters for each component to make their sum in $\hat{h}$ exactly equal to $\delta_{\hat{d}}$. We set the initial scales of the components in the $\hat{d}$ to their lower bound defined by the grammar. And then we gradually increase the components' scales to approximate $\delta_{\hat{d}}$. For directions other than $\hat{d}$, the components' scales are randomly assigned.

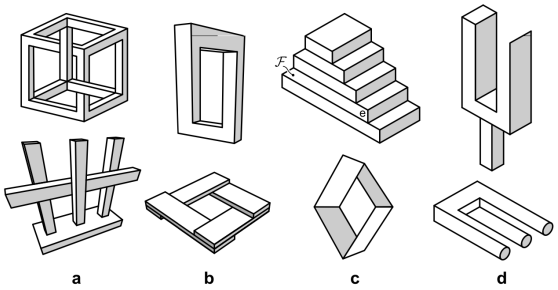

Figure 26: *Earlier research [SRC20, Mor10] has categorized impossible structures into four distinct types: a) Depth interposition, b) Depth contradiction, c) Disappearing normals, and d) Disappearing space. Our system is capable of generating structures falling under categories a), b), and c). Furthermore, we can produce intricate structures by employing combinations of different types. Image from [SRC20]*

Appendix C: Proof of Connecting Conditions

In this section, we show why the three conditions introduced in Section 5.3 are sufficient to guarantee the existence of a valid sequence in $\hat{d}$. To summarize, the conditions are:

- The lower bound of the components' sum is less than the target distance.
- The upper bound of the components' sum is larger than the target distance.
- A transition rule exists to proceed to the next direction.

We frame the problem as outlined below:

Claim:

Let sets $S_1..S_i$ be closed, continuous sets in R, denote as $S_i = [l_i, h_i]$. If $\sum_i^n l_i \leq \delta \leq \sum_i^n h_i$. Then we can always find a sequence $a_1 \in S_1$, $a_2 \in S_2 ... \ a_i \in S_i$ such that $\sum_i^n a_i = \delta$.

Proof:

Let sets $S_1..S_i$ be continuous closed sets, and $S_i = [l_i, h_i]$. We can choose $a_1 = l_1, a_2 = l_2 ... \ a_i = l_i$ such that $\sum_i^n a_i \leq \delta$. Let $\delta' = \delta - \sum_i^n a_i$, and we know $\delta' \geq 0$.

Since $\sum_i^n h_i \geq \delta = \sum_i^n a_i + \delta' \geq \sum_i^n l_i$. We know $\delta' \leq \sum_i^n h_i - l_i$. Since each $[l_i h_i]$ is a continuous space, we can have $\delta' = \varepsilon_1 + \varepsilon_2 + ... + \varepsilon_i$ such that $\varepsilon_i \in [0, h_i - l_i]$.

Appendix D: Validation on Connecting Algorithm

We examined our connecting algorithm (Section 5.3) on connecting two arbitrary procedural components. In our test, we randomly choose two components of different module types and place them in random 3D positions that require 10+ connecting steps to connect between them. Then we execute our connecting algorithm to find a path to connect the two components. Our experiments show a success rate of >40 % in executing all our grammars.

In our implementation, our system operates using 300+ sampling particles (Appendix G), with each independently executing the connecting algorithm. This setup ensures that we can always make a successful connection.

In contrast, we conducted experiments using random execution of procedural grammar for pathfinding under the same setup. Even with 1000 particles, we never observe any successful connection.

Appendix E: Probabilistic Analysis using Markov Chain on Transition Rules

In this section, we will demonstrate how we take a set of modules, denoted as M = A(P), B(P), ..., and a production rule R as input, and then compute the probabilities of each module using a Markov chain. We formulate the problem as below.

Input

- **States:** Given module M = A(P), B(P), ..., we define set of distinct states A, B, C, ... Each state represents a type of module.
- **Transition rules:** For each state X, a set of probabilities denoted as P(Y|X), where Y is another state and P(Y|X) represents the probability of transitioning from state X to state Y. These probabilities sum up to 1 for each X.

Iterative Approach

- **Base case:** We initialize the probabilities with $P(A_O)$=0, $P(B_O)$=0, $P(C_O)$=0 ... We set $P(M_O)$=$P(N_O)$=0.5, in which m,n are module types for the visual bridges.
- **Iterative Process:** At step n, we go through each of the states and calculate the probability of each state Y at step n as $P(Y_{n+1}) = \sum_{i=1}^{N} P(Y|X_i) \cdot P(Y_{i_n})$. We repeat the iterative process for multiple time steps until the probabilities stabilize (in our implementation, we choose 10 as the iterative steps).
- **Validation:** Lastly, we validate that $P(A_n), P(B_n), P(C_n) .... \geq 0$, and $\sum_i P(i_n) = 0$

Output

We output $P(A_n)$, $P(B_n)$, $P(C_n)$... as the final probabilities for each module type.

Appendix F: SMC Implementation

We employ the Sequential Monte Carlo (SMC) method to focus the search on promising structures. In our approach, the prior $p(x)$ is represented by the random selections made by the procedural grammar, while the likelihood $p(y|x)$ corresponds to the scoring function F as detailed in (Section 6). Our system simultaneously tracks and executes k particles in parallel, as outlined in Appendix G. At each step, our system samples a new set of particles based on the likelihood, progressing to the subsequent step.

For our purposes, we use $F(c_i)$ to denote the score of a structure at step i. Considering all k particles, we use $F(c_i^j)$ to represent the score of a particle j at step i. The aggregate score of all k particles at step i is calculated as: $Sum(i) = \Sigma F(c_i^j)$ Therefore, for particle j, the probability of its selection in the next step $i+1$ is given by: $\frac{F(c_i^k)}{Sum(i)}$

Appendix G: Timing

In this section, we will provide statistics on the time consumption of executing our program. Our tests were conducted on a 2019 MacBook Pro equipped with an Intel i7 GPU. Each test was run 5 times (does not include rendering), and we calculated the average results (numbers rounded to 1 decimal place). Our conclusion is that our system is capable of generating complex results within 5 minutes.

Timing				
Complexity C	Number of Steps s	Visual Bridge Distance d	Number of Particles	Time (seconds)
1.0	1	3	400	4.5
3.5	5	3.5	400	6.2
4.2	5	4	400	9.4
4.7	10	3.5	400	10.9
5.3	10	4	600	13.0
7.1	20	4.5	600	23.4
7.8	20	5	600	26.0
9.0	30	5	600	55.8
9.5	30	5.5	600	60.2
10.1	40	5	1000	143.4
10.9	40	6	1000	162.2
11.8	50	5.5	1500	232.1

Appendix H: Complexity Calculation

In this section, we outline how our system takes a single input number (denoted as C, ranging from 1 to 12, C can be a float) to compute parameters within our system. Specifically, we compute the distance d(which is a float) between P_1 and P_2 in 3D space (used in Section 5.1), the number of steps s (which must be an integer) for the random derivation phase (used in Section 5.2), and number of particles (used in Section 7) from the complexity input C.

In our implementation, we place an orthographic camera positioned at coordinates $(5,5,5)$ in world space, with an up vector of $(0,1,0)$ and a look-at point at $(0,0,0)$. Our viewport is an 800×800 square, and we select $r = 100$ to keep P near the center of the screen. We choose P_1 (the position for the foreground component) randomly at a distance between 2 and 3 units from the screen.

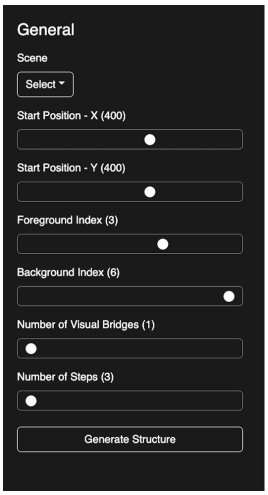

Figure 27: *A detailed view of the advanced options in the user interface.*

In such a setting, we first compute the number of particles used. For complexity smaller than 5, we use 400 particles; for complexity between 5 and 10, we use 600 particles; and for complexity more than 10, we use 1000 particles. Then we compute s (the number of steps) and d (the distance between P_1 and P_2 in units) as below:

$$C^2 + 10 = 2 \times (s + (d-1)^2)$$

The design of this formula allows for various combinations of s and d for a given C. To ensure our flexibility in our parameter selection, we introduce a constant term of 10 on the right-hand side of the equation. When C is relatively small (e.g., C = 1), the foreground and background components can still have some distance between them, for example, 3 units (making s = 1).

When computing values for both s and d, our approach involves first generating a random number for d, and then solving for s based on this randomly chosen d (we round up if s is not an int). This process enables us to achieve different combinations of s and d while adhering to the specified input complexity C.

Appendix I: User Interace

Our system offers advanced user options for controlling the generation process (Figure 27). Users can determine the number of visual bridges and set their positions on the screen and in 3D coordinates. Users can also choose the number of steps in the visual bridge phase (Section 5.2). This feature enables users to tailor the generation process to their individual preferences.

Appendix J: Artist Feedback

Our collaborating artist has shared that understanding the concept of visual bridges is straightforward. The specific technique used to construct these bridges is not crucial as long as the method is

repeatable and the final representation is clear. Creating grammar that begins with visual bridges and progressively incorporates new elements is a feasible approach.

Furthermore, our artist notes that the current results effectively depict impossible structures. However, the results are limited to parametric variations. A richer narrative and more intricate design are desirable. For example, the system could augment these structures by weaving in dynamic elements such as the growth of plants, the construction of houses, and shifts in weather. The creation of such detailed models is a time-intensive process, often spanning several months. Currently, the structures do the most important thing which is to create understandable impossible structures.

Appendix K: Generating Cycles in 3D space

In this section, we illustrate how our connecting algorithm, as introduced in Section 5.3, can be adapted for generating randomized cycles in 3D space. The algorithm takes an initial component, denoted as c_0, from the procedural grammar and a specified number of steps, denoted as s. Its output is a 3D cycle structure. It is important to note that c_0 should have at least two distinct derivation rules.

Starting with the initial component c_0 we select two distinct production rules and execute each rule for s steps. This process leads to two sequences of components, and we c_A and c_B to denote the ending components of two different derivations. Subsequently, we employ our connecting algorithm to establish a 3D connection between c_A and c_B. As a result, we obtain a 3D structure in which c_A and c_B are connected through the connecting algorithm. Additionally, both c_A and c_B are connected to c_0 as they are derived from it. We have a cycle in 3D space.