Non-Uniform Tile Wave Function Collapse

Piepenbrink, Rolf; Bidarra, Rafael

DOI
10.1109/CoG64752.2025.11114084

Licence
Dutch Copyright Act (Article 25fa)

Publication date
2025

Document Version
Final published version

Published in
Proceedings of the IEEE 2025 Conference on Games, CoG 2025

Non-Uniform Tile Wave Function Collapse

Rolf Piepenbrink, Rafael Bidarra
Computer Graphics and Visualization Group
Delft University of Technology
Delft, The Netherlands
Email: rpiepenbrink@proton.me, r.bidarra@tudelft.nl

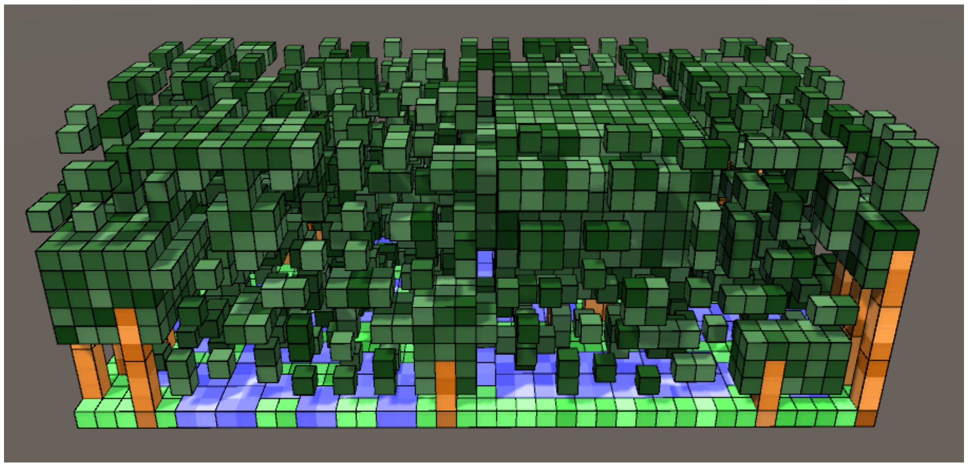


Fig. 1: A three-dimensional forest terrain generated with nutWFC.

***Abstract*—Procedural Content Generation methods enable the creation of varied content algorithmically. One such method is Wave Function Collapse (WFC), a tile-based local constraint solver commonly applied to texture, map and level generation for grid-based content; it is able to create varied output from the same set of rules, usually derived from an input sample. However, a glaring limitation of WFC is that it only operates on tiles of the same shape and size. We propose Non-Uniform Tile Wave Function Collapse (nutWFC), an extension of WFC that supports multi-cellular tiles with varying shapes and sizes, so-called Non-Uniform Tiles (NUTs). Familiar examples of such tiles can be found in LEGO® and Tetris. The algorithm guarantees NUT shape and size preservation even under WFC's Overlapping Model in three dimensions. We show that nutWFC is a super-set of WFC that harmonizes strict NUT shape and size constraints with WFC's output diversity without significant performance penalties. We illustrate the expressive power of nutWFC with a few results that explore the advantages of NUTs and would therefore not be feasible with WFC.**

***Index Terms*—Wave Function Collapse, non-uniform tiles, procedural content generation, constraint solving**

I. Introduction

WFC is a local constraint solver that has been a topic of great research interest since its introduction in 2016 by Gumin [1]. The power of WFC lies with its ability to quickly generate diverse grid-based output from the same input: either i) a set of tiles and a set of adjacency constraints, specifying which tiles may be adjacent, or ii) a grid-based texture example from which the tile set and constraints are inferred.

Most approaches deploy WFC on rectangular or cuboid tiles of the same size, spanning the same number of cells in a single grid; the tiles must thus be uniform. However, much can be gained from tiles with varying shapes, spanning multiple cells, which we call Non-Uniform Tiles (NUTs). NUTs allow WFC to enter currently unexplored domains where tile variety, shape and size preservation is crucial. This can include brick-by-brick construction of LEGO® models, Tetris-inspired textures [2] and varied three-dimensional terrains. Unfortunately, standard WFC cannot achieve this, due to the following limitations:

1) it is unable to represent and handle NUTs and cannot, therefore, guarantee the preservation of a NUT's diverse shape and size.
2) even if this obstacle were overcome, no mechanism is in place for representing and operating on NUT adjacency constraints either. Contrary to constraints between uniform tiles, NUT adjacency constraints are ambiguous: there can be multiple configurations that satisfy such constraints.
3) WFC can learn from a grid-based input only if it assumes that *every tile* occupies a uniform grid pattern (e.g. 1×1 or 3×3); however, it cannot distinguish NUTs within the input (by their own non-uniform nature) which, in turn, prevents WFC from preserving NUTs' properties.

To overcome these issues and provide access to the creative freedom and expressiveness of such new domains, we present Non-Uniform Tile Wave Function Collapse (nutWFC), an extension of WFC capable of supporting NUT sets. Most no-

2025 IEEE Conference on Games (CoG) | 979-8-3315-8904-2/25/$31.00 ©2025 IEEE | DOI: 10.1109/COG64752.2025.11114084

tably, nutWFC can guarantee NUT shape and size preservation under WFC's Overlapping Model. This means combining the essential WFC capacity of learning adjacencies from an input example with the consistent use of only the tiles included in the given NUT set. As a reminder, under the Overlapping Model, standard WFC learns allowed tile adjacencies by scanning with a sliding window over the input grid.

The nutWFC algorithm is the main contribution of this paper. It was recently developed as a part of our earlier work Expressive Wave Function Collapse [3], where it was shown to truly be a super-set of standard WFC. The foundation of the underlying structure of NUTs lies with uniquely identifying its single-cellular elements. In other words, by disabling this identifier uniqueness within a NUT, standard WFC emerges.

In this work, we show how nutWFC supports NUTs and how it can be applied under WFC's Overlapping Model. To support the above claims, we illustrate the application of nutWFC to terrain and architectural structure generation, based on dedicated NUT tilesets.

II. Related Work

WFC's release by Gumin in 2016 attracted significant attention from researchers and artists alike [1]. Gumin's WFC repository currently features dozens of variations, including adaptations, optimizations, generalizations and implementations to name a few [4]–[8]. WFC can also be found in commercial games, such as Bad North [9], Townscaper [10] and Caves of Qud [11]. While WFC rests on the same foundations as Merrell's Model Synthesis published years prior [12], WFC had a larger impact, especially among game developers [7]. Nonetheless, Merrell's contributions remain an important source of inspiration.

Several researchers have addressed tiles spanning multiple cells for WFC. Stålberg's Bad North features modules that can span multiple cells in a grid [9]. Stålberg acknowledges the useful properties of such large modules, such as allowing for smooth transitions. To solve the issue of formulating the numerous tile configurations, he derives the modules' adjacency constraints based on their vertices at the edges of the module: if they match, adjacency is allowed.

Moreover, Newgas' Tessera [6] and Piepenbrink's Expressive Wave Function Collapse [3] tackle multi-cellular tiles by subdividing them into uniquely identified single-cellular components. By enforcing constraints between these elementary components, the tile's structure is maintained. Newgas performs this method on multi-cellular tiles, which he calls *big tiles* through the addition of pre- and postprocessing. Big tiles consist of a connected arrangement of single-cellular components called *cubes*. In order to determine the adjacency constraints between big tiles, Newgas opts for manually painting the big tiles' exposed faces.

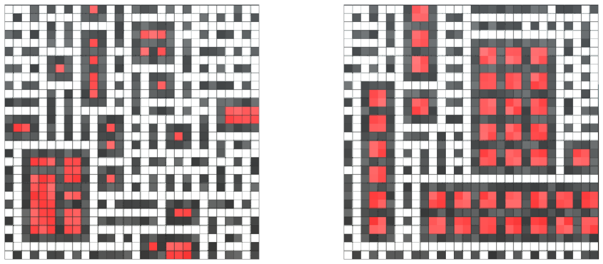
Two tiles are allowed to be adjacent if their colored edges match. However, specifying only these constraints is insufficient to rule out the overlap of non-rectangular big tiles. If the colors of two edges match, that does not mean that the other components — which must be collapsed into as well due to the structure of the tile — do not overlap. Consequently, this method is prone to conflict and is likely to often run into the same problems, as the irregularity of the tiles increases. Although a valid tiling may be found, it will be at the high cost of frequent conflict handling. Alternatively, one could choose to paint the edges so that overlap is reduced. However, determining this manually, as Tessera requires, can be an increasingly challenging task for complex big tile configurations.

While Newgas' approach is effective for the Simple Tiled Model, it largely lacks the capabilities for operating on the Overlapping Model because it does not learn big tile adjacency constraints from a grid-based input. Instead, Tessera requires manual user annotations to make them explicit. We can therefore conclude that his method is strongly compromised for use under the Overlapping Model.

Piepenbrink, in turn, generalizes Newgas' approach to multi-cellular tiles of any shape, where the adjacency constraints between the tiles are expressed in terms of the tile's single-cellular components. Rather than employing (manually) colored edges to infer constraints, Piepenbrink extrapolates adjacency constraints on multi-cellular tiles in two ways. The first method adheres to the same principles described in his earlier work with Bidarra [2]. Given an adjacency constraint between two multi-cellular tiles for a given direction, his algorithm computes the positions of two tiles such that (i) two single-cellular components are adjacent in the corresponding direction, and (ii) no intersection of single-cellular components of the two tiles is present. From such a position, the adjacency constraints on single-cellular components are obtained.

In its second method, Expressive Wave Function Collapse [3] features learning adjacency constraints from an input example model. Since this input does not explicitly define the composition of its multi-cellular tiles, the NUT set used in the input is also expected as a secondary input. This enables identifying the single-cellular components in the input, thereby allowing for inference of adjacency constraints. Through these two methods, multi-cellular tiles are not restricted to rectangles (i.e. the shape of their bounding box), and may be an arbitrary composition of connected single-cellular components. For the Simple Tiled Model, Expressive Wave Function Collapse utilizes both methods just described; for the Overlapping Model it uses the second method only.

An orthogonal problem to that approached here regards the generation of large patterns or structures spanning multiple grid cells in the output, but *always based on a uniform tileset*. Karth et al. [13], for example, proposed a combination of WFC with a a vector-quantizing variational autoencoder (VQ-VAE) to generate game maps based on one input map. Once trained, this approach is able to output a variety of maps, possibly containing several large structures. However, the WFC algorithm itself, which is applied in latent space to a grid of integer indexes, operates on a simple grid with a uniform tileset. Bateni et al. [14] propose a modified WFC to improve the resemblance between input and output, based on a context-sensitive heuristic for choosing the next tile to


(a) Output generated by WFC (left), not preserving the shape and size of the red square, and by nutWFC (right), preserving it. Color tone variations are added to reinforce each individual cell's character

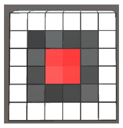

(b) The input used for the red square example.

Fig. 2: An example portraying the effect of uniquely identifying atoms of the red center NUT.

collapse. This heuristic is shown to perform better than that in Gumin's original algorithm. Again, they always use either uniform (1×1) tiles or overlapping (3×3) patterns on a grid.

From a different perspective, Langendam and Bidarra's miWFC features stamp-like functionality [15]. Here, the designer is able to manually create a group of tiles, called a *template*, and later reuse it, by placing instances on the grid. One of the main advantages of this approach is that it offers the user control on the design process. This is akin to how NUTs can be perceived: a pattern of small tiles collapsed at once. Likewise, this notion extends to Alaka and Bidarra's HSWFC, following a hierarchical approach for the WFC tile set [16]. Both approaches share in our goal of better capturing the semantics of patterns and offering a tighter design control [17].

III. Non-Uniform Tile fundamentals

There are plenty of domains where multi-cellular tiles with shape and size preservation are required. However, WFC is currently unable to cope with such heterogeneous tile sets. Consider the example shown in Figure 2 and suppose the central two-by-two red square is a tile whose shape and size must be preserved. With WFC, each single cell of the tile is considered equal, rendering it unable to preserve the tile. With nutWFC the red square NUT in the input remains intact. This notion extends to arbitrarily shaped tiles.

In this paper we introduce nutWFC, a major WFC extension capable of supporting NUTs, whose shapes and sizes must preserved. To achieve this, two challenges must be overcome: i) storing NUT information at cell level and ii) expressing adjacency constraints between NUTs in a form usable for the WFC algorithm.

nutWFC solves the former challenge through splitting a tile into uniquely identified single-cellular units, which we call *atoms* (see Subsection III-A). To overcome the second challenge of NUT adjacency ambiguity (see Figure 3), we

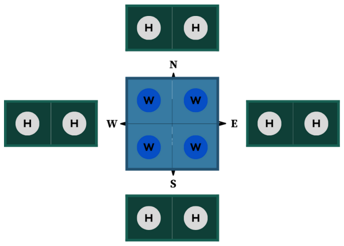


Fig. 3: NUT adjacency constraint ambiguity shown in 2D. The "H" tile can be adjacent to the "W" tile in multiple ways per direction (i.e. N(orth), E(ast), S(outh) and W(est).

apply the aforementioned mechanism on the input grid, so that the adjacency of two NUTs can be expressed in terms of adjacency of their atoms (see Subsection III-B). As we will show, this can be done in the initialization phase of the Overlapping Model, enabling compatibility with WFC with minimal changes.

For the remainder of this section, we use the following notation:

- For a given vector $\mathbf{v}$, the *i-th* element of the vector is referenced with $\mathbf{v}_i$.
- To reference properties of entities, we use dot notation. The extent of a given entity x is represented as Δ. For instance, to reference the *i-th* element of x's extent, we write $x.\Delta_i$.
- Matrix cells will be referenced following array notation, i.e. the element at cell x, y of a matrix M is denoted as $M[x, y]$.
- Referring to grid cells is done similarly, where the cell at coordinate $\mathbf{c}$ is denoted as $G[\mathbf{c}]$.
- To represent a cardinal direction $\hat{d}$, we use unit vectors and say that $\hat{d} = \pm\hat{\mathbf{e}}_k$, where k corresponds to the cardinal direction's axis.

A. NUT preservation through tile atomization

The purpose of tile atomization is to split a NUT into a set of atoms. This allows for differentiation among those single-cellular units of the NUT, and opens the doors to WFC compatibility. Before we discuss NUT properties, we introduce a formal definition of NUTs:

Definition 1 *(Non-Uniform Tile).* A *Non-Uniform Tile (NUT)* is a 3D tile that may span an arbitrary number of connected grid cells, and has an arbitrary shape and size that must be preserved. The shape is defined by a connected arrangement of uniquely identified single-cellular units called *atom*s, each with a relative position within the NUT. The size of the NUT, also referred to as *extent*, is the extent of the axis-aligned bounding box containing those atoms. Formally, a NUT n has an extent $n.\Delta$, and set of atoms $n.A = \{a_0, \ldots, a_n\}$.

From this definition, a NUT is an arrangement of atoms, whose relative positions within the NUT can be mapped to absolute coordinates within a 3D grid, based on four observations:

1) Cells are discrete and uniform within a grid.
2) Each cell contains at most one atom.
3) All atoms are single-cellular; they cannot overlap.
4) Each atom has a relative position within its NUT.

Since the atoms are single-cellular and cells are discrete, we can place the atoms according to their relative positions in a grid without any atoms overlapping. Each atom's coordinate within this grid is called the *atom coordinate* and is used to represent the atom's position. We denote the atom coordinate of an atom a as $a.c$. Since each cell can contain at most one atom, the combination of an atom's associated NUT's identifier and atom coordinate is unique. As a consequence, we can now distinguish between atoms within the same tile, which is crucial for NUT placement during the main loop of nutWFC. For ease of reference, we also assign a unique integer identifier to each atom. We denote the identifier of an atom a as $a.id$. Therefore, we can express a NUT in terms of the atoms within its extent, which brings us to the same domain as WFC's grid, with all its properties and advantages, including its handling of adjacency constraints.

Step one for NUT preservation is its atomization, yielding the building blocks that enable shape and size preservation. Per Definition 1, a NUT's atom arrangement, and thus its shape and size, must be preserved. We achieve this through specifying constraints between its atoms. In the grid of atoms we mentioned earlier, atoms can be neighbors of one another in a certain direction. We refer to that neighboring property as *atom adjacency*:

Definition 2 *(Atom adjacency).* Two atoms a_i and a_j are said to be *adjacent* in direction d if a_j is in a_i's neighboring cell in direction d. Atom adjacency between those atoms is denoted as $a_i \sim_d a_j$.

Whenever a NUT is placed (or collapsed) in the grid, its atoms must always occur in the same arrangement. Essentially, these can be expressed as adjacency constraints among the NUT's atoms, which we refer to as *inter-atom adjacency constraints*. These constraints will enforce that collapsing one of the atoms of a NUT requires collapsing all of its other atoms as well, thereby preserving shape and size. This follows from the observation that an atom a_i with an inter-atom adjacency constraint with an atom a_j in a direction d is not allowed to be adjacent to any other atoms in that direction. Thus, the atoms of a NUT are tightly bonded together. Since these relations are known beforehand, the constraints are determined in the initialization phase. These inter-atom adjacency constraints are fully compatible with WFC, since the constraints are expressed in terms of adjacency constraints between single-cellular units.

B. Adjacency between NUTs

With an individual NUT's properties preserved and compatibility with WFC achieved, we can turn into adjacency between NUTs. As mentioned, NUT adjacency is ambiguous and not directly compatible with WFC's tile adjacency. To overcome this, we have to clearly define what NUT adjacency entails. Because we expressed each NUT in terms of its differentiated atoms, we can use them to define NUT adjacency:

Definition 3 *(NUT adjacency).* Two NUTs n_i and n_j are said to be *adjacent* in a given direction d, if at least one atom of n_i is adjacent (see Definition 2) to at least one atom in n_j in direction d.

From this definition, the ambiguity becomes clear: there may be multiple configurations of two NUTs in which their atoms are adjacent. To express that two NUTs may be adjacent, we formulate NUT adjacency constraint in Definition 4.

Definition 4 *(NUT adjacency constraint).* A *NUT adjacency constraint* is a constraint stating that two NUTs may be adjacent along a given direction d, following Definition 3.

With this knowledge in mind, we can now describe how these constraints are learned from the input.

IV. Core nutWFC algorithm

A. Grid atomization: learning from input

WFC traditionally learns the tile set and adjacency constraints from input examples, typically a pixel texture. During the learning phase, the input is scanned to derive both the tile set and the adjacency constraints.

However, when that input is build up of NUTs, the NUT set is evidently prior to the input itself, rather than derived from it. We, therefore, argue that it is a reasonable requirement to pass along, as input, both the NUT set and one (or more) grid example(s) using those NUTs. By ensuring that each NUTs' atoms are unique, we are guaranteed to find a valid configurations of NUTs in the provided input.

Certainly, standard WFC cannot cope with such a heterogeneous tileset anyway; but how can nutWFC learn from such an input? The answer to this is given by the notion of *grid atomization*, analogous to the mechanism behind tile atomization, described in Subsection III-A.

Input grid atomization essentially consists of determining a NUT's atom identifier for each of its cell. For this, the cells in the input grid G can be mapped to a matrix, G', in which each cell's value is the atom identifier of the NUT at that atom coordinate. For this, grid atomization follows a greedy sliding window approach, incrementally iterating over all cells of the input grid, starting at its zero-index. For each cell with coordinate $\mathbf{c}$ in grid G, we overlay a NUT n_i with extent $n_i.\Delta$ and atoms $n_i.A$ such that all atoms $a_j \in n_i.A$, with their respective atom coordinates $a_j.\mathbf{c}$ cover cells $\mathbf{c} + a_j.\mathbf{c}$.

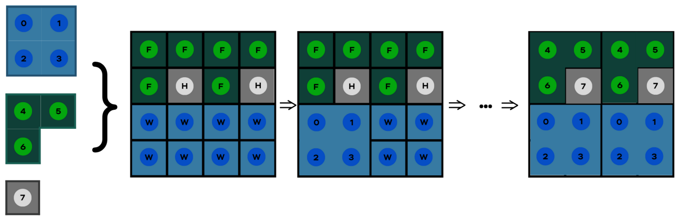


Fig. 4: Input grid atomization: given a set of atomized NUTs (left), we overlay and slide each NUT over the grid, to find a fitting NUT and associate its atom identifiers to the cells in the grid.

Algorithm 1 Pattern Adjacency Inference

```
1:  $D = \{d_0, \dots, d_k\}$                    ▷ Set of directions
2:  $P = p_0, \dots, p_n$ ▷ Set of patterns obtained form the input
3:  for each $p_i$ in $P$ do
4:      for each $p_j$ in $P$ do
5:          for each $d$ in $D$ do
6:              if IsCompatible($p_i$, $p_j$, $d$) then
7:                  StorePatternAdjacency($p_i$, $p_j$, $d$)
8:                  StorePatternAdjacency($p_j$, $p_i$, $-d$)
9:              end if
10:         end for
11:     end for
12: end for
```

We say that a NUT *fits* in the input grid at a given coordinate $\mathbf{c}$, if for all atoms a_j of NUT n_i, the atom's identifier equals the value stored in grid G at coordinate $\mathbf{c} + a_j.\mathbf{c}$. Figure 4 depicts a high-level illustration of this mapping. By starting from the $\mathbf{0}$ coordinate in G and selecting cells incrementally thereafter, we align the NUTs with this coordinate.

After each successful NUT fitting check on the input grid G, we map its atoms as well onto the atomized grid G'. For this, we store the identifier $a_i.id$ of an atom a_i at $\mathbf{c} + a_i.\mathbf{c}$.

Once the input has been atomized, the adjacency constraints between its atoms (and their respective NUTs) can be derived for the Overlapping Model, as described next.

B. Patterns for the Overlapping Model

The key for deriving and capturing nutWFC adjacency constraints between NUTs lies in the notion of a *pattern*, a small square portion of the atomized grid G'. A kernel w with extent $w.\Delta$ is used to obtain the patterns from the atomized grid G'. For this, the kernel slides cell-by-cell over the entire grid: at each position, one pattern is registered, and kept in set $P = \{p_0, \dots, p_n\}$. From the definition of the Overlapping Model, an adjacency constraint between two patterns depends on how they overlap, hence the minimal size of the kernel is $\mathbf{2}$.

Pattern adjacency constraints are obtained by overlapping any two such patterns along a given direction $\hat{d} = \pm\hat{\mathbf{e}}_k$, and checking their compatibility, as shown in Algorithm 1. Let p_i and p_j be two patterns obtained as described above. Each pattern aggregates a square of atoms, with their identifiers. Consequently, each atom has a position within the pattern, which we refer to as a *pattern coordinate*. Evaluating whether two patterns are compatible (see line 6 of Algorithm 1), and their NUTs may therefore be adjacent for a given direction $\hat{d}$, involves the following procedure:

1) Position patterns p_j and p_i such that they fully overlap. That is, each pattern coordinate $\mathbf{c_i} \in p_i.C$ maps to a pattern coordinate $\mathbf{c_j} \in p_j.C$.
2) Translate p_j by $\hat{d}$; not all pattern coordinates overlap anymore.
3) For each pair of overlapping pattern coordinates $(\mathbf{c_i}, \mathbf{c_j'})$, check whether the value at $\mathbf{c_i}$ equals the value at $\mathbf{c_j'}$.
4) If all assertions pass, p_j and p_i are said to be *compatible* along direction $\hat{d}$.

One of the advantages of this use of NUTs in nutWFC is that a small kernel of size $\mathbf{2}$ can convey much more information than with standard WFC, due to NUT atomization: the patterns obtained through input processing contain atom identifiers rather than tiles. Since each atom belongs to a concrete NUT, collapsing a pattern on the output grid, and thereby the atoms in said pattern, will enforce that all other atoms of those NUTs (possibly outside of the pattern extent) must be collapsed as well. In other words, atomization facilitates maintaining the shape and semantics explicitly desired for each NUT.

In addition, patterns containing only atoms of the same NUT become redundant, since those atom relations are already enforced through the inter-atom adjacency constraints, maintained in the atom adjacency matrix. The larger a NUT is, the fewer patterns are required to represent it (relative to its size). While this sounds counter-intuitive, it makes sense: the purpose of a pattern is to contain information on how two (or more) NUTs may be adjacent in a given context. For a large NUT, only patterns at its perimeter contain information on how it touches others NUTs, all other internal patterns can be ignored.

C. nutWFC for the Overlapping Model

The basic nutWFC algorithm uses the features and properties described above when extending the standard WFC algorithm, as shown in Algorithm 2.

During observation, the cell with the lowest entropy is selected and then collapsed. However, since the domain of the Overlapping Model is unionized — operating on both patterns and atoms — collapsing works slightly differently. At runtime, we must know which cells are collapsed to which atom *within* the pattern. Therefore, upon collapsing a cell into a pattern, we assign it the atom identifier at pattern coordinate $\mathbf{0}$. Subsequently, a new collapse wave propagates, where all other atoms belonging to that atom's NUT are also collapsed. However, a new challenge arises with this step: immediately collapsing cells into atoms, rather than according to patterns, results in a lack of pattern data. To overcome this, we precompute at initialization stage, the table of which patterns correspond to which atom identifiers. Then, at run-time we easily retrieve from this table, for each collapsed cell, which patterns correspond to their atom identifiers. In this way, we safely restore the correspondence between atom identifiers and patterns, and avoid including in the pattern adjacency matrix any patterns containing only atoms belonging to the same NUT.

D. Pattern elimination at the grid's edges

While the inter-atom adjacency constraints ensure that a NUT's atoms are all properly laid out on the grid, this is not always guaranteed at the edges of the grid. There, upon propagating the collapse of all those atoms, it could happen that some of them would fall outside of the grid, yielding an 'incomplete NUT' behind, thus violating the NUT shape and size preservation requirement.

The atoms that could cause that violation are known before run-time. Therefore, during the initialization phase of nutWFC, a cleanup step is performed on each grid cell, eliminating from it any (potential) patterns whose associated NUTs do not entirely fit within the grid. Definition 5 formulates this for a given position in a grid for one of a NUT's atoms. It is required to be specific here, due to the different ways a Non-Uniform Tile (NUT) can be laid on a grid. With this definition, elimination can be explained for the Overlapping Model.

Definition 5 *(Allowed NUT).* A NUT n with extent $n.\Delta$ and atoms $n.A$ is said to be allowed for an atom $n.a_i \in n.A$ with atom coordinate $a_i.\mathbf{c}$ on a grid G with extent $G.\Delta$ at grid coordinate $\mathbf{c}$ if the following criteria are met:

1) (NUT within bounds) $\mathbf{c} - a_i.\mathbf{c} \geq \mathbf{0} \wedge \mathbf{c} - a_i.\mathbf{c} + n.\Delta \leq G.\Delta$
2) (free) If requirement 1 holds, assert that for each n's atom coordinates $a_j.\mathbf{c} \in n.A$ grid cell $G[\mathbf{c} + a_j.\mathbf{c}]$ may be collapsed into atom a_j.

Pattern elimination has six steps:

1) Given a cell at coordinate $\mathbf{c}$, consider it to correspond to the pattern atom coordinate relative to the pattern p, which has extent $p.\Delta$.
2) For each pattern cell coordinate $p.\mathbf{c}$ where $\mathbf{0} \leq p.\mathbf{c} < p.\Delta$, assert that the NUT associated with the atom a in the pattern at $p.\mathbf{c}$ is allowed for atom a and cell $\mathbf{c} + p.\mathbf{c}$ in the grid.
3) Disallow the pattern if at least on of its atoms results in a disallowed NUT in the step prior and exclude the disallowed pattern from the cell's wave.
4) Edge case: consider an atom resulting in a disallowed NUT and the grid positions of all other atoms associated to the same NUT. If all of the cells at those positions were already collapsed, the NUT would not be placed at all. The properties of a NUT that will not be placed cannot be violated. This means that the pattern should thus still be allowed.
5) If a cell's wave was changed, add it to the propagation queue.
6) Perform propagation once all cells at the grid's edges have been processed.

Algorithm 2 nutWFC for the Overlapping Model

```
1: Initialize()
2: while not all cells are collapsed do
3:     Observe()
4:     Collapse()
5:     CollapseNUTsInPattern()
6:     Propagate()
7: end while
```

E. Conflict handling

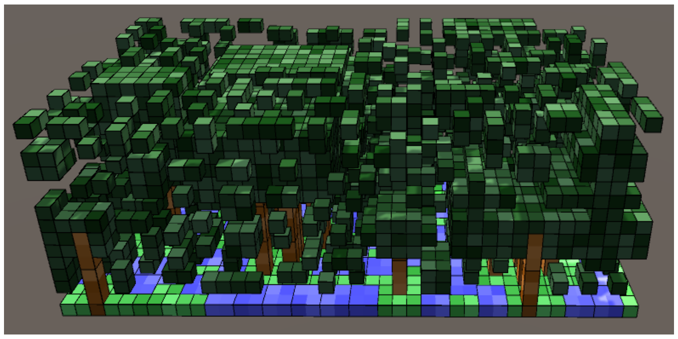
When a cell runs out of options, a conflict is encountered and WFC would fail, even though a solution could exist in the search space. By exploring this search space, the likelihood of finding a solution increases. Backtracking, for example, is guaranteed to return a solution should one exist, as it can traverse the whole search space [6], [15]. However, backtracking can only guarantee this at the cost of efficiency. We therefore opt for a method that employs save states. This method works by periodically creating save points that consist of the wave and a grid containing the collapsed atoms (nutWFC's Overlapping Model requires both). When a conflict is encountered, i.e. when a cell runs out of options, nutWFC is reverted to a previous save point to try again from there. The use of save points favors efficiency due to its lower time complexity, but cannot guarantee that a solution will always be found should it exist.

V. Results and Discussion

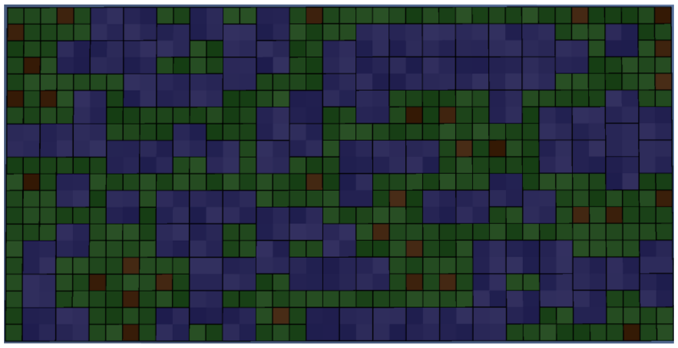
In this section, we showcase the results of the following two experiments. **Experiment 1** features generating a forest consisting of trees, grass and wide water bodies — an under-explored application of WFC. Its purpose is to illustrate how NUTs can help strengthen creative freedom compared to standard WFC. **Experiment 2** illustrates how a simple two-story hut can be generated consisting of a variety of NUTs. Its goal is to highlight how NUTs can be used to support repeating NUT patterns while allowing for diversity. A more comprehensive discussion of performance can be found in Chapter 7 of Piepenbrink's earlier work [3]. To reinforce a NUT's individual atoms, purely cosmetic color variations have been added to the atoms displayed in the figures.

A. Forest terrain experiment

The results of Experiment 1 are shown in Figures 1 and 5. They illustrate how control (by means of the input patterns) and diversity (by combining many allowed adjacencies) are harmonized, resulting in more expressiveness. For instance, one can easily control the minimum height of the tree trunks with NUTs, while maintaining variety of the tree canopies, as indicated by the tree leaves in Figures 1 and 5a. Moreover, the

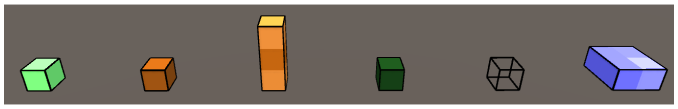


(a) An elevated rear view of the forest.

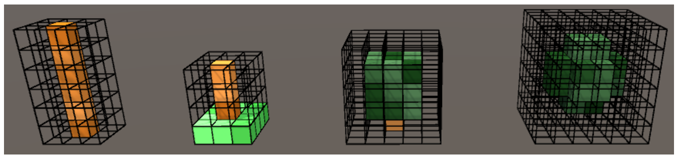


(b) A bottom view of the forest.

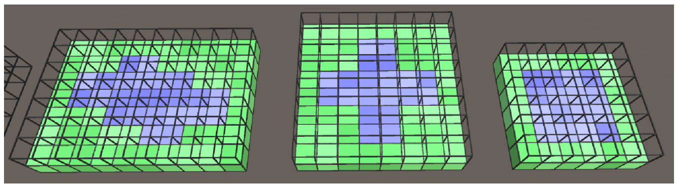
Fig. 5: The results of the forest terrain experiment. Grid size (w,h,d): 40,15,20.



(a) The NUT set used in the forest experiment. From left to right: grass, root, trunk, leaf, void, water.

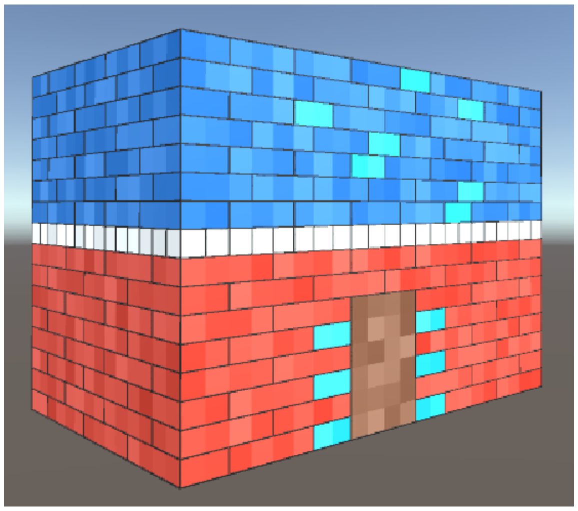


(b) The patterns used for generating the trees.

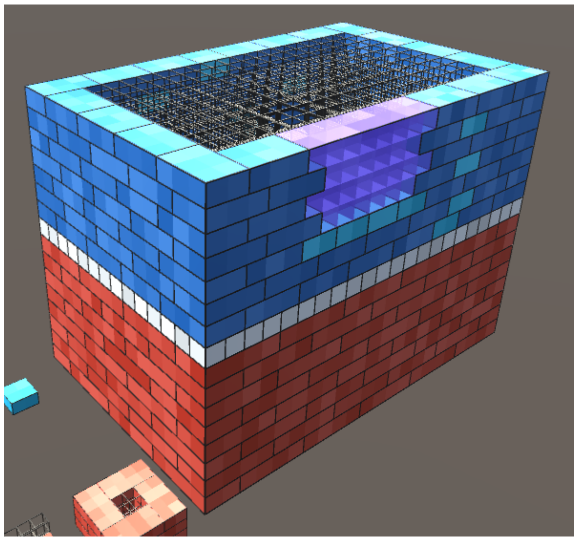


(c) The patterns used for generating water and grass.

Fig. 6: The input used for forest generation (Experiment 1).



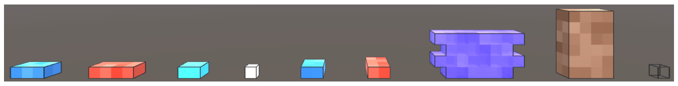
(a) A front view of the hut.



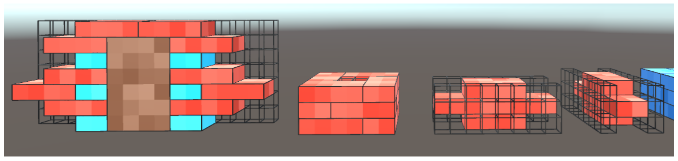
(b) A rear view of the hut.

Fig. 7: The results of the hut experiment. Grid size (w,h,d): 22,16,14.

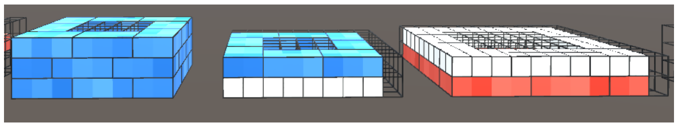
experiment shows how NUTs offer more control over the scale and coarseness of free form elements, without compromising variety. For example, by varying the tree base pattern in Figure 6b, one can control the minimum distance between trees. Furthermore, the tree heights illustrate how, with atomization, simple input grids can be used to generate more structured representations, which can be tweaked by slightly changing the NUT set as preferred. In addition, the water bodies shown in Figure 5b support this claim, where utilizing a two-by-two NUT for composing the bodies steers the generation in line with the intention of generating a body of water without thin streams or deltas. Attempting to achieve this result with standard WFC would be extremely challenging as larger NUTs are being used, certainly without larger sliding windows under the Overlapping Model, due to WFC's lack of unique atom identification. With nutWFC, this is not an issue: a small window of size **2** suffices for this variety (see Figure 5), based on a small set of inputs (see Figure 6).



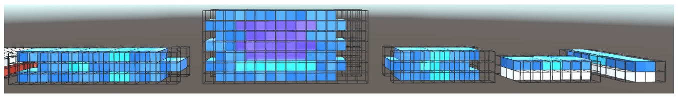
(a) The NUT set used in the hut experiment. From left to right: b312, b412, b212, b111, b213, b214, window, door, void.



(b) The patterns used for generating the ground floor walls with a door.



(c) The patterns used for combining the two floors.



(d) The patterns used for generating the second floor with a window.

Fig. 8: The input used for hut generation (Experiment 2).

B. Hut experiment

The results of Experiment 2 are shown in Figure 7. In this experiment, we favored control over freedom. The brick patterns are repeated according to the input given (see Figure 8), placing them brick-by-brick, which is very different from simply applying a texture. The red bottom floor also features a door, whose shape and size are preserved correctly. This shows that nutWFC can guarantee NUTs semantics upon placement, unlike previous variants of WFC. Likewise, the blue top floor illustrates nutWFC capabilities of synthesizing different brick patterns and creating new combinations of brick configurations. This floor also includes a large window with an irregular shape (see Figure 7b), reinforcing the claim that NUT shapes may have arbitrary forms. This experiment featured a larger set of patterns, as shown in Figure 8, in order to include two distinct floors in the output.

VI. Conclusion

Despite its popularity, the WFC algorithm is unable to work with heterogeneous tile sets, consisting of tiles of disparate shapes and sizes. We presented Non-Uniform Tile Wave Function Collapse (nutWFC), a major extension of WFC that introduces this possibility, through the notion of Non-Uniform Tile (NUT), a rigid agglomeration of distinct one-cellular atoms.

We described how nutWFC extends WFC to work under its Overlapping Model, while preserving NUT shape and size in all generated output. In addition, we also showed how WFC can be seen as a particular case of nutWFC, in which all atoms of each NUT have the same identifier.

Among nutWFC's most innovative and attractive features, we exemplified (i) the easier and more intuitive fine-tuning provided by individual input patterns, which help specify and preserve their intended semantics; (ii) the much richer and nuanced input enabled by nutWFC allows for a larger expressive power. We therefore believe nutWFC has the potential to impact new areas of content generation, in particular in a mixed-initiative context, empowering the incremental expression of a designer's intent. For this reason, we plan to further investigate how such interactive facilities can best profit from the procedural core of nutWFC, developed for our current prototype implementation.

References

[1] M. Gumin, "Wave Function Collapse," https://github.com/mxgmn/WaveFunctionCollapse, 2016, accessed: 2024-08-15.

[2] R. Piepenbrink and R. Bidarra, "How much Tetris can Wave Function Collapse put up with?" Delft University of Technology, Delft, The Netherlands, Tech. Rep. CGV-24-1, May 2024. [Online]. Available: http://graphics.tudelft.nl/Publications-new/2024/PB24

[3] R. Piepenbrink, "Expressive Wave Function Collapse," Master's thesis, Delft University of Technology, August 2024. [Online]. Available: https://resolver.tudelft.nl/uuid:cc11b0d7-82b5-48e5-adc2-d5033a6ab661

[4] I. Karth and A. M. Smith, "WaveFunctionCollapse is constraint solving in the wild," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*, 2017, pp. 1–10.

[5] ——, "WaveFunctionCollapse: Content generation via constraint solving and machine learning," *IEEE Transactions on Games*, vol. 14, no. 3, pp. 364–376, 2021.

[6] A. Newgas, "Tessera: A practical system for extended WaveFunctionCollapse," in *Proceedings of the 16th International Conference on the Foundations of Digital Games*, 2021, pp. 1–7.

[7] ——, "Infinite modifying in blocks," https://www.boristhebrave.com/2021/11/08/infinite-modifying-in-blocks/, 2021, accessed: 2024-08-15.

[8] Marian42, "Wave Function Collapse - an algorithm for generating random structures," https://marian42.de/article/wfc/, 2020, accessed: 2024-08-16.

[9] O. Stålberg. (2018) EPC2018 - Wave Function Collapse in Bad North. Youtube. [Online]. Available: https://www.youtube.com/watch?v=0bcZb-SsnrA

[10] ——, "Townscaper," https://store.steampowered.com/app/1291340/Townscaper/, 2021, accessed: 2024-08-15.

[11] F. Games, "Caves of Qud," https://www.cavesofqud.com/, 2015, accessed: 2024-08-15.

[12] P. Merrell and D. Manocha, "Model synthesis: A general procedural modeling algorithm," *IEEE Transactions on Visualization and Computer Graphics*, vol. 17, no. 6, pp. 715–728, 2010.

[13] I. Karth, B. Aytemiz, R. Mawhorter, and A. M. Smith, "Neurosymbolic map generation with VQ-VAE and WFC," in *Proceedings of the 16th International Conference on the Foundations of Digital Games*, 2021, pp. 1–8.

[14] B. Bateni, I. Karth, and A. Smith, "Better resemblance without bigger patterns: Making context-sensitive decisions in WFC," in *Proceedings of the 18th International Conference on the Foundations of Digital Games*, 2023, pp. 1–11, accessed: 2024-08-16.

[15] T. S. Langendam and R. Bidarra, "miWFC - designer empowerment through mixed-initiative Wave Function Collapse," in *Proceedings of the 17th International Conference on the Foundations of Digital Games*, 2022, pp. 1–8. [Online]. Available: https://publications.graphics.tudelft.nl/papers/45

[16] S. Alaka and R. Bidarra, "Hierarchical Semantic Wave Function Collapse," in *Proceedings of the 18th International Conference on the Foundations of Digital Games*, 2023, accessed: 2024-08-16. [Online]. Available: https://publications.graphics.tudelft.nl/papers/67

[17] ——, "Mixed-initiative generation of virtual worlds - a comparative study on the cognitive load of WFC and HSWFC," in *Proceedings of the 19th International Conference on the Foundations of Digital Games*, 2024, accessed: 2025-04-16. [Online]. Available: https://publications.graphics.tudelft.nl/papers/23