Supplemental Document

Building Construction Sets by Tiling Grammar Simplification

Abstract

In this document we describe an approach for CSG-based extraction of r-similar building blocks. We start with a surface decomposition into r-symmetric components and convert them into exchangeable, rigid, 3D building blocks allowing manual assembly of shape variations. We then give detailed instructions on how to reproduce the printable building block for several of our test models. Finally, we provide additional simplification results using various cost function parameterizations.

Algorithm summary

We extract the geometric neighborhoods of the slippable and rigid parts of the model separately, using different strategies for each type. The rigid neighborhoods can be enclosed by a finite number of spheres and cylinders with radius r. The one-slippable neighborhoods are also cylindrical, but we extract the two-slippable parts of the model as the difference between the models and the remaining building blocks.

Rigid pieces

Each non-slippable building block in our segmentation is defined by a set of corner points, with r-neighborhoods matching the r-neighborhoods of all equivalent microtiles. Hence, the r-neighborhood of each microtile is contained inside a set of spheres (around corner features) and cylinders (along line features) by definition. This allows us to use a union of elementary shapes to cut out the microtile together with its r-neighborhood from the input shape using CSG operations. Similarly, by definition of r-symmetry each surface point at distance not greater than r to any geometric edge with both ends inside the microtile has an equivalent surface point in all microtile instances. Therefore, the geometry inside these volumes is guaranteed to be identical (equivalent) for all microtiles of a given type.

We need to take special care for overlapping r-neighborhoods of neighboring tiles when extracting building blocks that need to be put together after manufacturing. Intersecting r-neighborhoods of instances of the same tile class are present even in the simple example in **Figure 1.**

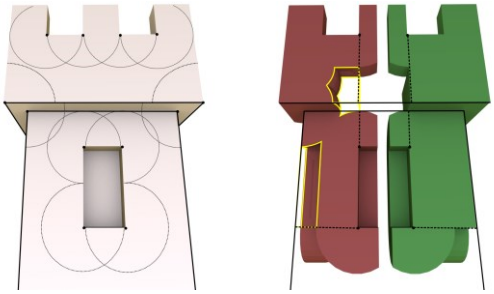


Figure 1: The input model (left), some of its corner features, their r-neighborhoods and all visible r-slippable edges (black). We extend the r-neighborhoods of rigid pieces (right) along outgoing long edges to avoid concavities (outlined in yellow) that are likely to complicate the assembly of the manufactured pieces.

To resolve overlapping neighborhoods of tiles we modify the equivalence relation that defines our elementary building blocks. Initially microtiles are classified according to r-neighborhoods of corner features. We construct a new decomposition by splitting classes of microtiles if they have different neighbors at distance less or equal than 2r. This allows to classify r-neighborhoods of corner points as well as intersections of these neighborhoods into equivalence classes.

At this point, it remains to separate pairs of corners and edges with intersecting r-neighborhoods at each tile boundary consistently. A general solution to this problem is to construct a 3D Voronoi diagram and split along its edges. For our tests, we constructed Voronoi regions only at corner points. We then

extended the regions along edges using the convex hull of the Voronoi regions at both end points of each edge (see **Figure 2**). If pairs of neighborhoods constructed in this way had intersections, we assigned the intersecting volume to the smaller neighborhood. It is important to note that using Voronoi regions to separate r-neighborhoods of corner points allows to preserve all planar cuts that split rigid geometry through the middle of geometric edges shorter than r.

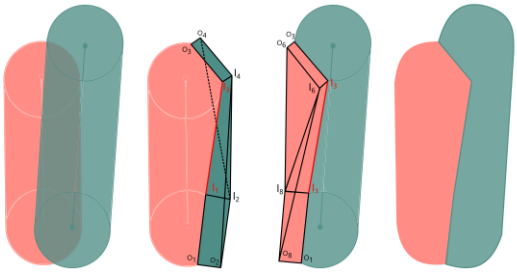


Figure 2: Our naive strategy for separating overlapping r-neighborhoods in 2D. From left to right: the initial situation, the subtracted polygons (second, third image), and the resulting cut.

Slippable pieces

After we have removed all rigid r-neighborhoods in the previous step, we can extract one-slippable tiles by intersecting the modified input object with a set of their enclosing cylinders. The difference of the input shape and the remaining microtiles consists of all two-slippable building blocks attached to the volume enclosed by the shape.

In order to separate the individual planar pieces from the inner volume, one can intersect them with their enclosing voxels. A more efficient approach is to intersect with a single bounding box aligned with the plane of the two-slippable region. Because the rigid and one-slippable pieces are subtracted beforehand, it is not necessary to compute a more accurate enclosing volume.

Separating the two-slippable parts from the interior is not always necessary (see **Figure 4**), but it increases the likelihood that the pieces can be assembled after manufacturing. We parameterized the CSG tree for the slippable building blocks to allow three types of cutting as illustrated in **Figure 4**. For our experiments, we preferred to merge one- and two-slippable microtiles together. This resulted in fewer building blocks and did not prevent assembly for our test models.

To increase shape variability we separate slippable pieces along symmetric pairs of global planar cuts. Each pair of planes defines an "inner" and "outer" region of the input model. From the intersections of these regions we obtain a general, not necessarily rectilinear, grid. We split the geometry in different grid cells by adding an intersection with the respective cell volume to the CSG-tree used to extract the final pieces.

Some pairs of symmetric planes isolate a region consisting entirely of slippable geometry, e.g. the middle section of the tower in **Figure 4**. If this is the case, valid shape variations can be obtained by "stretching" or "shrinking" the slippable building blocks.

Instructions

This section provides information on how to reproduce our results using the automatically generated scripts and our test models.

Download or install (OpenSCAD, 2014). This tool is needed to execute the required CSG operations on the input model and generate triangle meshed for each building block. We used version 2014.03.

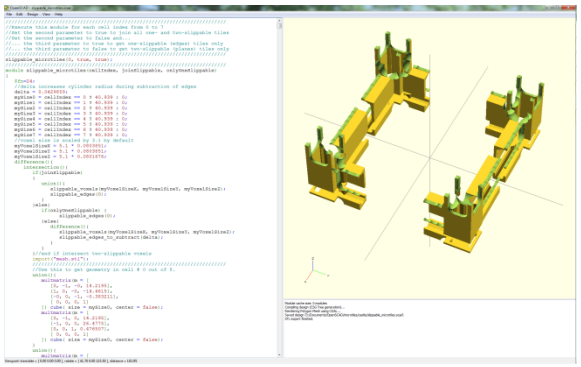


Figure 3: Building blocks in OpenSCAD

The supplemental materials are split into directories for each model. The necessary files in each of the folders are named

- `mesh.stl` - this file contains the input model.
- `rigid_microtiles.scad` - this is a script that generates each of the rigid building blocks.
- `slippable_microtiles.scad` - this script generates the slippable building blocks.

The scripts have to be executed several times to generate all pieces. An OpenSCAD script is executed by selecting

`Design -> Compile and Render (CGAL)`.

Note that this operation can take several hours and requires a lot of memory (up to 4GB per running script). After OpenSCAD generates the triangle mesh (see Figure 3) it can be saved with

`File -> Export -> Export as STL....`

The exact amount of executions varies for the different models, and there are more specific instructions in the beginning of the script. It is only necessary to change the index of the tile for each execution in order to generate each building block (as used in the paper).

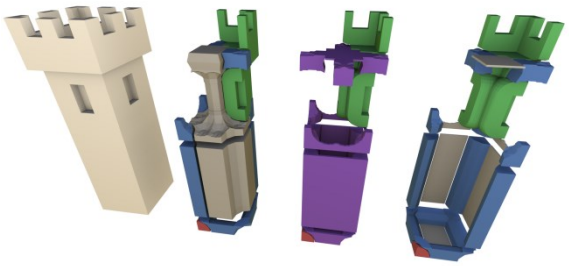


Figure 4: CSG modes for extraction of two-slippable building blocks. Please see the paper for more details.

Changing the additional parameters in `slippable_microtiles.scad` can be used if a different CSG mode for slippable tiles extraction is desired (see Figure 4Figure 4). The default parameters will merge one- and two-slippable building blocks together (the purple decomposition). Alternatively, one can generate them separately as shown in the rightmost CSG mode in **Figure 4**. The scripts currently do not allow for easy generation of the first type of building blocks. We included the generated meshes for all of the CSG modes for the tower model.

For convenience we also include `.blend` files (see (Blender Foundation, 2014)) that contain enough copies of the building block to construct the input model. The meshes are also oriented in a favorable way for 3D printing. From Blender, all objects can be selected and exported to ready-to-print `.stl` files.

Before printing, we scaled the building blocks for the tower model by a factor of 12, the castle and house models by a factor of 10 and the bridge model by a factor of 3. Note that it may be necessary to print the sets of building blocks in more than one runs, if the printer build platform is not large enough.

Bibliography

Blender Foundation. (2014). *Blender*. Retrieved 2014, from Blender: http://www.blender.org/

OpenSCAD. (2014). *OpenSCAD*. Retrieved 2014, from OpenSCAD: http://www.openscad.org

Weights

Redundancy	Classes	Overlap	#Cuts
2	1	1	2

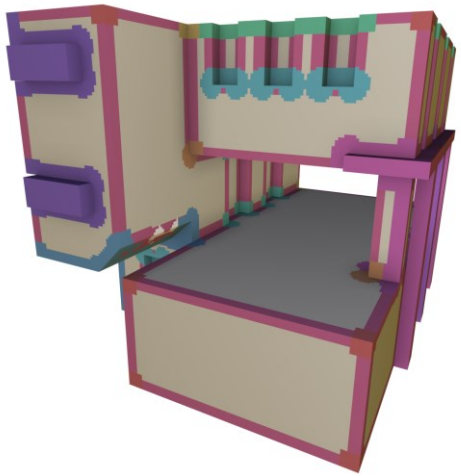
Initial Costs

Redundancy	Classes	Overlap	Cuts
0.02	1	0.14	100

Optimized Cost = 0.62

Redundancy	Classes	Overlap	Cuts
0.1	0.38	0.04	0

#T before	#T after	#C before	#C after
251	77	42	16



Weights

Redundancy	Classes	Overlap	#Cuts
4	1	1	2

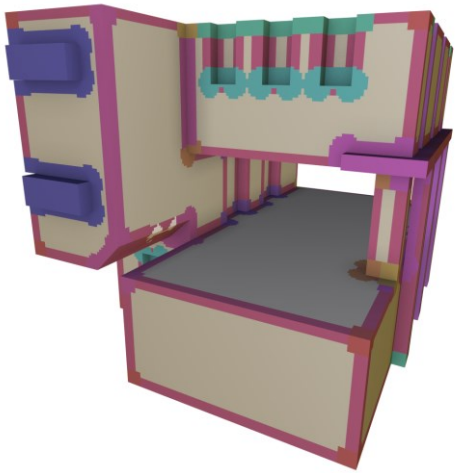
Initial Costs

Redundancy	Classes	Overlap	Cuts
0.02	1	0.14	100

Optimized Cost = 0.64

Redundancy	Classes	Overlap	Cuts
0.05	0.40	0.04	0

#T before	#T after	#C before	#C after
251	98	42	17



Weights

Redundancy	Classes	Overlap	#Cuts
12	1	1	2

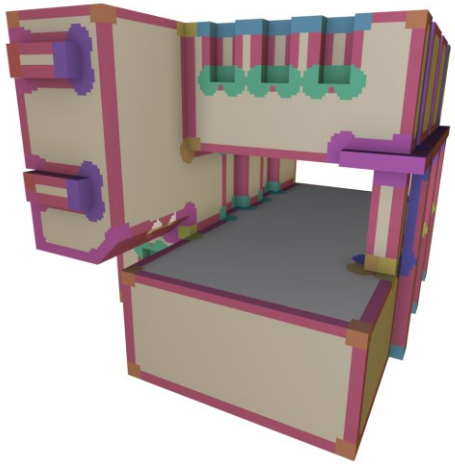
Initial Costs

Redundancy	Classes	Overlap	Cuts
0.02	1	0.14	100

Optimized Cost = 0.95

Redundancy	Classes	Overlap	Cuts
0.04	0.48	0.04	0

#T before	#T after	#C before	#C after
251	109	42	20



Weights

Redundancy	Classes	Overlap	#Cuts
16	1	1	2

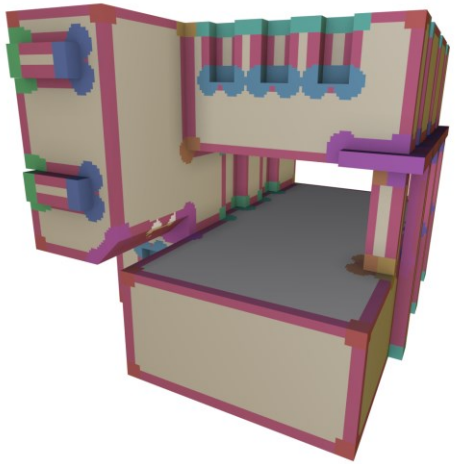
Initial Costs

Redundancy	Classes	Overlap	Cuts
0.02	1	0.14	100

Optimized Cost = 0.95

Redundancy	Classes	Overlap	Cuts
0.03	0.43	0.01	0

#T before	#T after	#C before	#C after
251	117	42	18



Weights

Redundancy	Classes	Overlap	#Cuts
<= 1.75	1	1	2

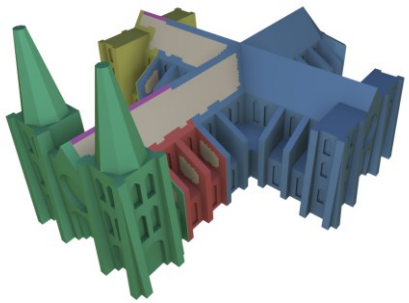
Initial Costs

Redundancy	Classes	Overlap	Cuts
0.06	1	0.25	100

Optimized Cost = 0.88

Redundancy	Classes	Overlap	Cuts
0.58	0.01	0	0

#T before	#T after	#C before	#C after
2042	7	309	4



Weights

Redundancy	Classes	Overlap	#Cuts
1.85	1	1	2

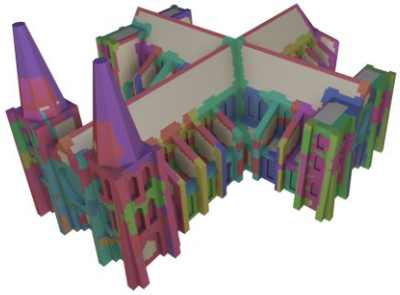
Initial Costs

Redundancy	Classes	Overlap	Cuts
0.06	1	0.25	100

Optimized Cost = 0.79

Redundancy	Classes	Overlap	Cuts
0.23	0.26	0.20	0

#T before	#T after	#C before	#C after
2042	163	309	49



Weights

Redundancy	Classes	Overlap	#Cuts
2	1	1	2

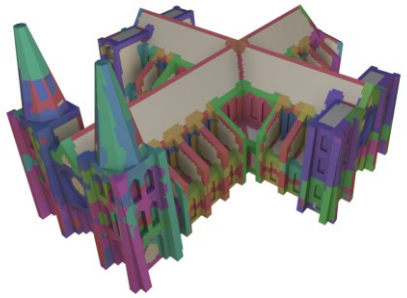
Initial Costs

Redundancy	Classes	Overlap	Cuts
0.06	1	0.25	100

Optimized Cost = 0.93

Redundancy	Classes	Overlap	Cuts
0.23	0.24	0.24	0

#T before	#T after	#C before	#C after
2042	230	309	73



Weights

Redundancy	Classes	Overlap	#Cuts
4	1	1	2

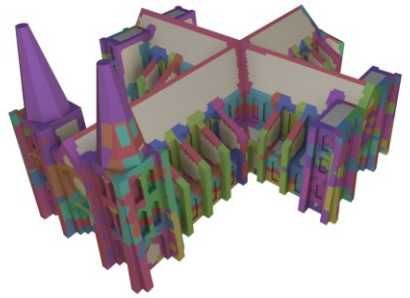
Initial Costs

Redundancy	Classes	Overlap	Cuts
0.06	1	0.25	100

Optimized Cost = 1.15

Redundancy	Classes	Overlap	Cuts
0.16	0.27	0.25	0

#T before	#T after	#C before	#C after
2042	364	309	86



Weights

Redundancy	Classes	Overlap	#Cuts
1	1	1	1

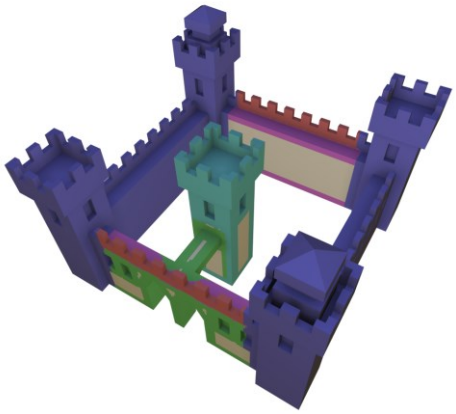
Initial Costs

Redundancy	Classes	Overlap	Cuts
0.04	1	0.26	1200

Optimized Cost = 0.53

Redundancy	Classes	Overlap	Cuts
0.44	0.09	0	0

#T before	#T after	#C before	#C after
716	7	61	5



Weights

Redundancy	Classes	Overlap	#Cuts
4	1	1	1

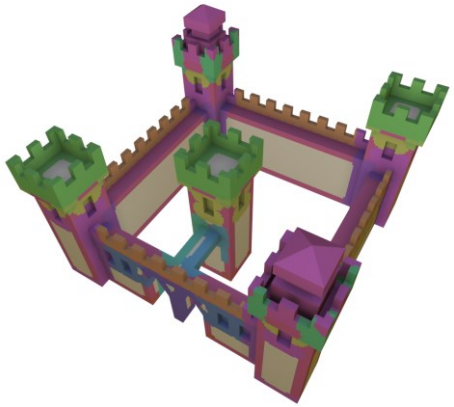
Initial Costs

Redundancy	Classes	Overlap	Cuts
0.04	1	0.26	1200

Optimized Cost = 1.05

Redundancy	Classes	Overlap	Cuts
0.18	0.26	0.07	0

#T before	#T after	#C before	#C after
716	77	61	15



Weights

Redundancy	Classes	Overlap	#Cuts
8	1	1	1

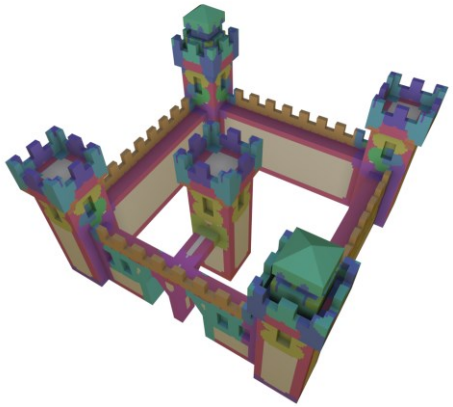
Initial Costs

Redundancy	Classes	Overlap	Cuts
0.04	1	0.26	1200

Optimized Cost = 1.4

Redundancy	Classes	Overlap	Cuts
0.1	0.34	0.23	0

#T before	#T after	#C before	#C after
716	176	61	21



Weights

Redundancy	Classes	Overlap	#Cuts
12	1	1	1

Initial Costs

Redundancy	Classes	Overlap	Cuts
0.04	1	0.26	1200

Optimized Cost = 1.99

Redundancy	Classes	Overlap	Cuts
0.08	0.75	0.22	0

#T before	#T after	#C before	#C after
716	355	61	46

