Supplementary Material for *Computational Design of High-level Interlocking Puzzles*

This supplementary material is composed of three parts. The first part presents implementation details about our shape optimization in Section 6 of the paper. The second part illustrates the kernel disassembly graph of 3D puzzle results shown in the paper. The last part presents details about our user study in Section 7 of the paper.

1 Shape Optimization and Voxelization

This section provides algorithmic details for the shape optimization and voxelization presented in our paper. The pseudocode of our algorithm is presented below.

Our method has two steps: transformation step and shape optimization step. Our approach has to iterate between these two steps n times in order to converge to a good result. In the transformation step, our algorithm scales and transforms the mesh to minimize the energy $E_{\texttt{voxel}}$. We use the function `TransformationOpt` $(\mathbf{V})$ for this problem. In the shape optimization step, we optimize the mesh's vertices to reduce the voxel energy $E_{\texttt{voxel}}$ while preserving the shape energy $E_{\texttt{shape}}$. The main function of this step is `MeshOpt` $(\mathbf{V}, \mathbf{S}, \{\mathbf{R}_i\})$. The shape preservation energy we use is from the paper [Sorkine and Alexa 2007], which requires us to compute series of rigid transformations $\{\mathbf{R}_i\}$ before optimizing the mesh's vertices. The function `RigidTransformation` $(\mathbf{V}, \mathbf{V}_0)$ is aimed for this purpose. We iterate the shape optimization and computation of the rigid transformation m times to acquire an actual shape preservation energy $E_{\texttt{shape}}$. Lastly, the mesh needs to be deformed into a shell $\mathbf{S}$ which is generated by using the function `ShellGeneration` $(\boldsymbol{t}_{i+1}, w_{i+1}, \mathbf{V})$.

ALGORITHM 1: Pseudocode for shape optimization and voxelization

```
Function shape_optimization_voxelization ($\mathbf{V}_0$)
    $i = 0$;
    $\mathbf{V} = \mathbf{V}_0$;
    for $i < n$ do
        $\boldsymbol{t}, w$ = TransformationOpt($\mathbf{V}$);
        $\mathbf{S}$ = ShellGeneration($\boldsymbol{t}_{i+1}, w_{i+1}, \mathbf{V}$);
        $j = 0$;
        for $j < m$ do
            $\{\mathbf{R}_i\}$ = RigidTransformation($\mathbf{V}, \mathbf{V}_0$);
            $\mathbf{V}$ = MeshOpt($\mathbf{V}, \mathbf{S}, \{\mathbf{R}_i\}$);
        end
    end
    return $(\mathbf{V}, \boldsymbol{t}, w)$;
end
```

Parameters:

(1) n: the number of iterations of optimization both voxel transformation and shape optimization

(2) m: the number of iterations of conducting the as-rigid-as possible method [Sorkine and Alexa 2007].

1.1 `TransformationOpt` $(\mathbf{V})$

We uniformly sample the variables' space of $(\boldsymbol{t}, w)$. Among them, we find the optimal $(\boldsymbol{t}, w)$ with the lowest energy $E_{\texttt{voxel}}$. The key for computing this energy is to calculate the volume of each voxel (intersection with the mesh). Rather than computing the exact intersection shape, we uniformly sample points inside the voxel and count those that are inside the mesh. This number gives a rough approximation of the voxel's volume. To further accelerate the query of whether a point is inside a given mesh, we utilize the method "fast winding number" from paper [Barill et al. 2018]. Note that by carefully choosing the sampling density, we could reuse some sampling points to reduce computational cost.

1.2 `RigidTransformation` $(\mathbf{V})$

We follow the optimization scheme from the paper [Sorkine and Alexa 2007]. For each local iteration, it computes a rigid transformation $\{\mathbf{R}_i\}$ from the initial shape $\mathbf{V}_0$ to the deformed shape $\mathbf{V}$ by minimizing the following energy:

$$\mathbf{R}_i = \min_{\mathbf{R}_i} \sum_{j \in N(i)} |(\mathbf{v}_i - \mathbf{v}_j) - \mathbf{R}_i(\mathbf{v}_i^0 - \mathbf{v}_j^0)|^2 \tag{1}$$

where i goes through the vertices of the input shape, $N(i)$ is the 1-ring neighbour of vertex i, $\mathbf{v}_i$ is the vertices' position of the deformed shape, and $\mathbf{v}_i^0$ is the vertices' position of the initial shape.

The paper [Sorkine and Alexa 2007] provides a fast implementation of computing the rotation matrix $\{\mathbf{R}_i\}$ by utilizing the singular value decomposition. Please refer to their paper for more details.

1.3 `MeshOpt` $(\mathbf{V}, \mathbf{S}, \{\mathbf{R}_i\})$

After computing the rigid transformation $\{\mathbf{R}_i\}$, we formulate the shape energy as:

$$E_{\texttt{shape}} = \sum_i \sum_{j \in N(i)} |(\mathbf{v}_i - \mathbf{v}_j) - \boldsymbol{R}_i(\mathbf{v}_i^0 - \mathbf{v}_j^0)|^2 \tag{2}$$

We first compute derivatives of $E_{\texttt{shape}}$ and $E_{\texttt{voxel}}$ with respect to V. We then solve the optimization by using L-BFGS method.

2 Kernel Disassembly Graph

For each puzzle result shown in the paper, we provide the 3D models of the puzzle pieces, as well as the kernel disassembly graph, as supplementary data. In the following, we show an example kernel disassembly graph for a level-5 CUBE puzzle in Figure 1. Each node in the kernel disassembly graph represents a puzzle configuration. The green node indicates the initial puzzle configuration (i.e., the root node) and the red node indicates a target node where a subassembly has been taken out. The blue line shows the shortest path to take out the first subassembly which defines the puzzle's level of difficulty. Please refer to Figure 3 and our provided supplementary data for more complicated graphs.

3 User Study

In our user study, we recruited 8 participants, 4 males and 4 females, between 19 to 33 years old. For each participant, we briefly introduced the concept of our high-level interlocking puzzle and then informed him/her the task of solving a given puzzle by translating the puzzle pieces. After each participant tried to solve the puzzle (i.e. disassemble the puzzle into individual pieces) within 15 minutes, we recorded whether he/she solved the puzzle successfully as well as

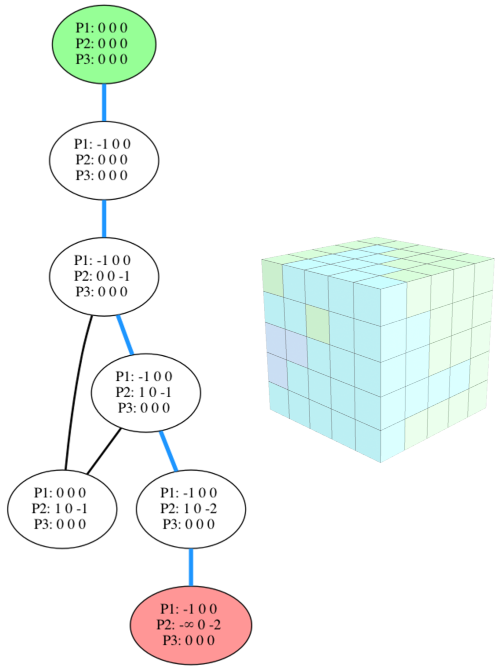

Figure 1: *(Left) Kernel disassembly graph of (right) a level-5* CUBE.

Model	Smooth Appearance	Resolution	E	K	L	G_N	G_E	G_T
Cube_L4	No	5x5x5	1	3	4	5	4	1
Cube_L8	No	5x5x5	1	3	8	11	11	1
Cube_L16	No	5x5x5	1	4	16	64	82	14
Sofa	No	7x8x6	0	4	8	307	637	91
Owl	Yes	5x5x5	1	3	7	8	7	1

Table 1: *Statistics of the puzzles for our user study. The labels in the 3rd to 9th columns refer to the voxelization resolution, number of hole voxels E, number of puzzle pieces K, level of difficulty L, number of nodes (G_N), edges (G_E), and target nodes (G_T) in the kernel disassembly graph.*

the time that he/she solved the puzzle. A questionnaire was required to be filled in after playing puzzles for each participant, which asked each participant to select the most attractive puzzle and the hardest puzzle among the given puzzles. We presented 5 puzzles to each participant including 3 CUBE 5x5x5 puzzles, SOFA and OWL; see Table 1 and Figure 2.

Result. The statistics of our user study is presented in Table 2. The slash in Table 2 means the user cannot solve the given puzzle in 15 minutes. There are three findings in our user study:

- *Level of difficulty.* We find that the average solving time increases with the level of difficulty. Specifically, if the level of difficulty is not that large (e.g. less than 8), the difference of solving time is not that significant. Moreover, when the level of difficulty increases to 16, 4 of our participants cannot solve CUBE_L16 within 15 minutes and the rest 4 participants' solving time is 7.021 minutes on average. All the participants claim that the hardest one among the five puzzles is CUBE_L16, which confirms that level of difficulty is an effective criterion to measure how challenging a puzzle is for users.
- *Kernel disassembly graph size.* By comparing the data of the CUBE_L8 and SOFA, the level of difficulty may not be the only criterion to measure the degree of complexity to solve a puzzle. These two puzzles have the same level-of-difficulty and similar resolution, but the average solving time of these 2 puzzles shows significant difference, 0.352 minutes for CUBE_L8 and 1.871 minutes for SOFA, respectively. We infer that the difference of kernel disassembly graph size leads to the difference of solving time for these 2 puzzles. The kernel disassembly graph of SOFA contains 307 nodes, 637 edges and 91 target nodes while that of CUBE_L8 has only 11 nodes, 11 edges and 1 target node; see Figure 3. The bigger kernel disassembly graph means that the user has to put more effort to find a disassembly plan in a larger puzzle configuration space.

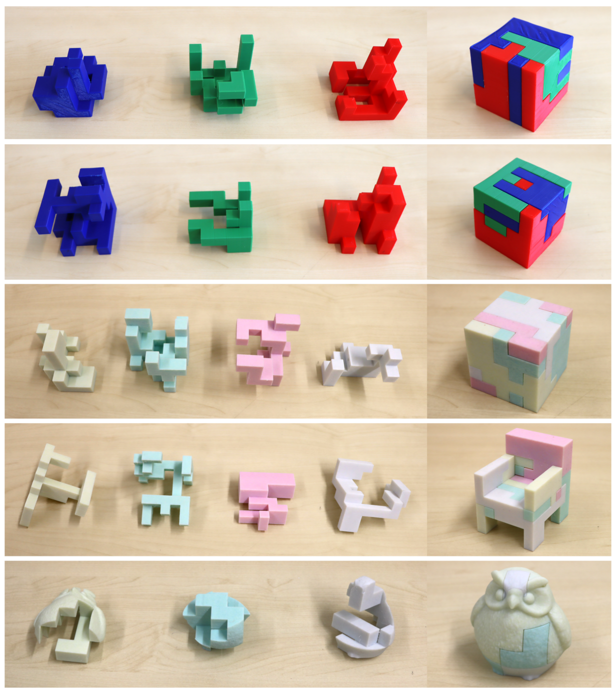

Figure 2: *Puzzles for our user study. From top to bottom,* CUBE_L4, CUBE_L8, CUBE_L16, SOFA, *and* OWL.

- *Puzzle appearance.* From the questionnaires filled in by the participants, 7 of 8 participants select OWL as the most attractive puzzle because of the fancy puzzle appearance. From the perspective of users, they may not only focus on the degree of complexity to solve the puzzle but also the fascinating puzzle appearance, which confirms the necessity of our approach to design puzzles with smooth appearance. By comparing the data of CUBE_L8 puzzle and OWL, which have the similar level and disassembly graph size, it shows significant difference of solving time, averagely 0.352 and 2.410 minutes, respectively. According to the feedback of the participants, the smooth appearance of puzzles may make it difficult to recognize blocking relationship among the puzzle pieces, therefore slowing down the puzzle solving process.

References

BARILL, G., DICKSON, N., SCHMIDT, R., LEVIN, D. I., AND JACOBSON, A. 2018. Fast winding numbers for soups and clouds. *ACM Trans. on Graph. (SIGGRAPH) 37*, 4, 43:1–43:12.

SORKINE, O., AND ALEXA, M. 2007. As-rigid-as-possible surface modeling. In *Proceedings of EUROGRAPHICS/ACM SIGGRAPH Symposium on Geometry Processing*, 109–116.

	Cube_L4 (mins)	Cube_L8 (mins)	Cube_L16 (mins)	Sofa (mins)	Owl (mins)	The most attractive puzzle	The hardest puzzle
User 1	0.250	0.350	/	1.567	1.833	Owl	Cube_L16
User 2	0.200	0.450	/	2.250	3.250	Owl	Cube_L16
User 3	0.350	0.333	/	2.667	3.250	Owl	Cube_L16
User 4	0.383	0.417	6.500	4.250	1.133	Cube_L16	Cube_L16
User 5	0.250	0.183	9.417	0.750	2.133	Owl	Cube_L16
User 6	0.416	0.500	/	0.833	3.167	Owl	Cube_L16
User 7	0.300	0.333	5.500	1.050	2.783	Owl	Cube_L16
User 8	0.233	0.250	6.667	1.600	1.733	Owl	Cube_L16
Average Solving Time (mins)	0.298	0.352	7.021	1.871	2.410		

Table 2: *Statistics of our user study. The 2nd column to 6th column present the puzzle solving time of each puzzle in minutes. Specifically, slash means the user cannot solve the given puzzle within 15 minutes. The 7th and 8th columns show the most attractive puzzle and the hardest puzzle selected by each participant. The last row provides the average time to solve each puzzle.*

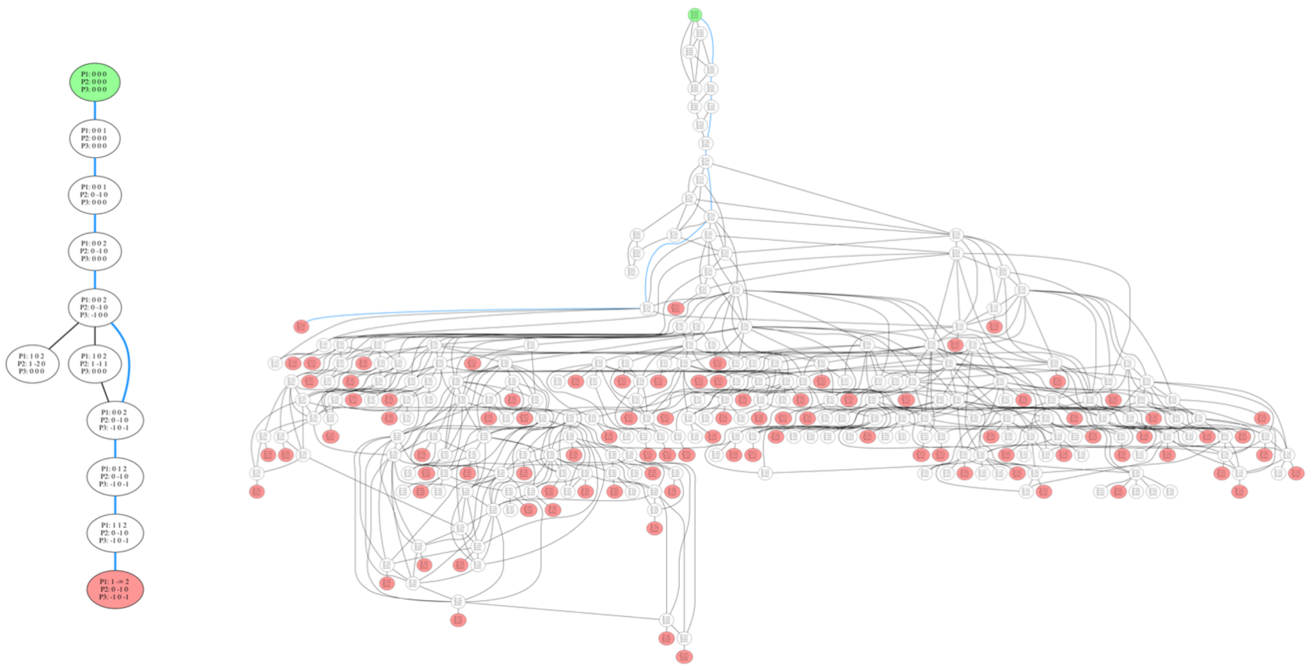

Figure 3: *Left: Kernel disassembly graph of* CUBE_L8. *Right: Kernel disassembly graph of* SOFA. CUBE_L8 *and* SOFA *are both level-8 puzzles but the kernel disassembly graph size of* SOFA *is much bigger than that of* CUBE_L8.