Final Report

Shaoxuan YUAN

August 2024

1 Introduction

In the previous work, we have been able to create LEGO® models using the brick set. Next, we will focus on its connectivity and stability to make sure it can be built in real world. On top of that, we will also enlarge our brick set by adding some curved bricks and plates, which will make the boundary smoother. Some popular LEGO® models are constructed in different orientations to improve the quality of design. We will try to create a computational method to solve this problem.

2 Related Work

In recent years, Liu et al. [2] analyse the forces and moment of a LEGO® model and transfer it into an optimization problem. What is more, Lambrecht [1] create a strategy to construct LEGO® sculptures in different orientations. And Zhou et al. [3] display a method to align the input shapes with grid points. We will follow these researches to optimize our generated LEGO® models and step up to generate more complex LEGO® models in multi-orientations.

3 Connectivity Optimization

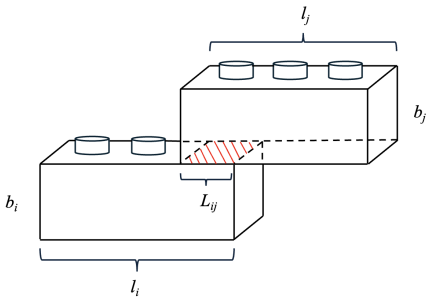

Figure 1: The connected structure

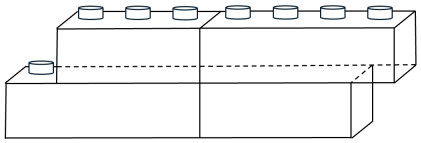

Figure 2: Inefficient structure

Connectivity is important because well-connected models tend to have high stability. In this part, we will introduce an optimization method to improve the connectivity of our generated LEGO® model. Our target is using less bricks to create more connections. First, we associate each brick with a binary variable x_i. Let us consider a structure with two bricks, denoted by b_i and b_j; see Figure 1. We associate each such structure with a value, denoted by $\mathcal{E}_{ij}$. We define L_{ij} as the length of intersection part of b_i and b_j. We also define l_i and l_j as the length of b_i and b_j respectively.

$$\mathcal{E}_{ij} = x_i x_j \frac{L_{ij}}{l_i + l_j} \tag{1}$$

The sum of the length of two bricks reflects the number of the bricks we use for using longer ones tends to reduce the number of bricks. The length of the intersection part of two bricks reflects the potential of generating connections since smaller intersection tends to have more positions to connect other structures.

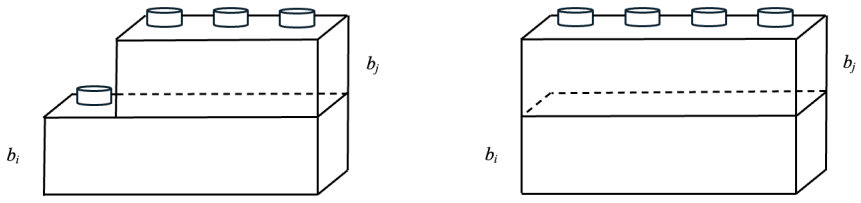

Figure 3: Case 1 (left) and Case 2 (right)

Let us consider a special structure; see Figure 2. This kind of structure is not efficient to create more connections. Based on this observation, we will add a factor to reduce the number of such structure. For each two bricks b_i and b_j, we define T_{ij} to reflect the impact of such structure.

- *Case 1.* If only one pair of their ends is aligned; see Figure 3 case 1:

$$T_{ij} = x_i x_j \tag{2}$$

- *Case 2.* If both of their ends are aligned; see Figure 3 case 2:

$$T_{ij} = 2x_i x_j \tag{3}$$

- *Case 3.* If none of their ends are aligned:

$$T_{ij} = 0 \tag{4}$$

We can simply prove that T_{ij} contributes more than $\mathcal{E}_{ij}$ in case 1 and case 2 because $\mathcal{E}_{ij}$ is impossibly larger than $\frac{x_i x_j}{2}$.

$$\mathcal{E}_{ij} = x_i x_j \frac{L_{ij}}{l_i + l_j} \leq x_i x_j \frac{min(l_i, l_j)}{l_i + l_j} \leq x_i x_j \frac{min(l_i, l_j)}{2min(l_i, l_j)} = \frac{x_i x_j}{2} \tag{5}$$

Then, we construct our objective function:

$$\mathcal{C} = \sum_{i<j} \mathcal{E}_{ij} + \sum_{i<j} T_{ij} \tag{6}$$

We will add several constraints into the connectivity-optimization model.

- *Non-empty-position.* We should make sure that all empty positions are filled. We should let the number of empty positions, denoted by P, be equal to the total length of bricks:

$$P = \sum_{i=1} x_i l_i \tag{7}$$

- *Non-conflicts.* We will define $\mathcal{M}$ as a conflict set. For each two bricks b_i and b_j, $(b_i,b_j) \in \mathcal{M}$ if they are at the same height and have intersection. We should make sure there are no conflict bricks:

$$\sum_{(b_i,b_j)\in\mathcal{M}} x_i x_j = 0 \tag{8}$$

- *Using least unit bricks.* Because unit bricks can create fewer connections, occupy the least position and make the search space larger, we prefer using the least unit bricks. To achieve this, we will optimize the model to get the least number of unit bricks before connectivity optimization and add it into connectivity-optimization model as a constraint:

$$\sum_{l_i=1} x_i = \min_{l_i=1} |b_i| \tag{9}$$

Finally, we apply an IP solver to minimize our objective function and obtain some well-connected LEGO® models; see Figure 4.

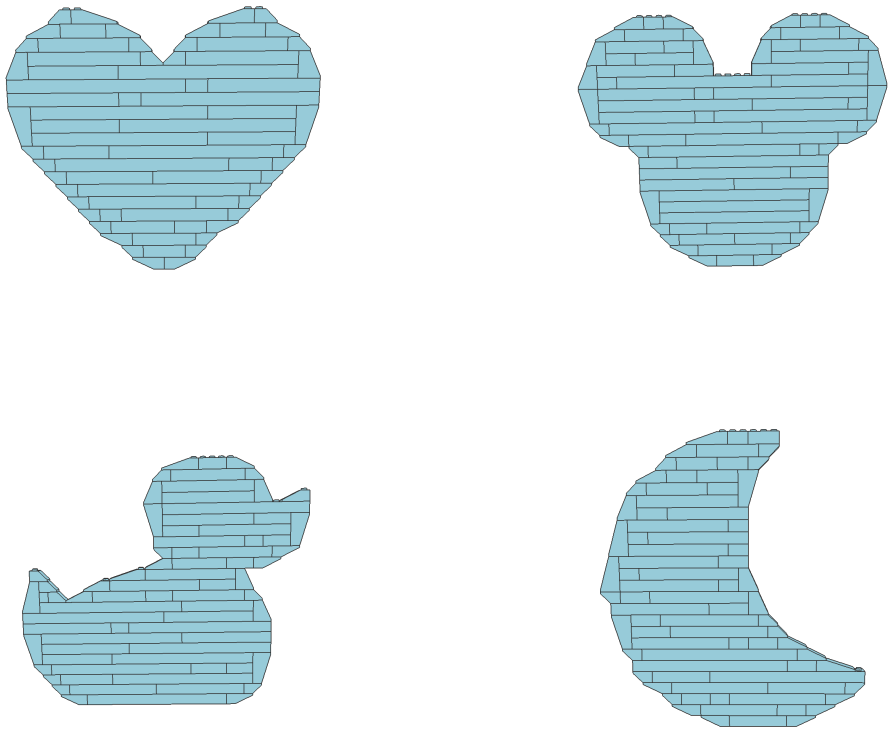

Figure 4: Connectivity Optimization

4 Stability Analysis

Stability is also an important factor. In this part, we will focus on the stability of generated LEGO® models. This problem can be transferred into a a rigid body equilibrium problem at the 2D level. We apply Pymunk to create rigid models according to our generated LEGO® layouts and simulate real-world situation. We also use Pygame to make this analysis visible. If the models are not stable, the simulated models will collapse. Our method can determine the stability of our LEGO® models within seconds. There are some results of stability analysis; see Figure 5.

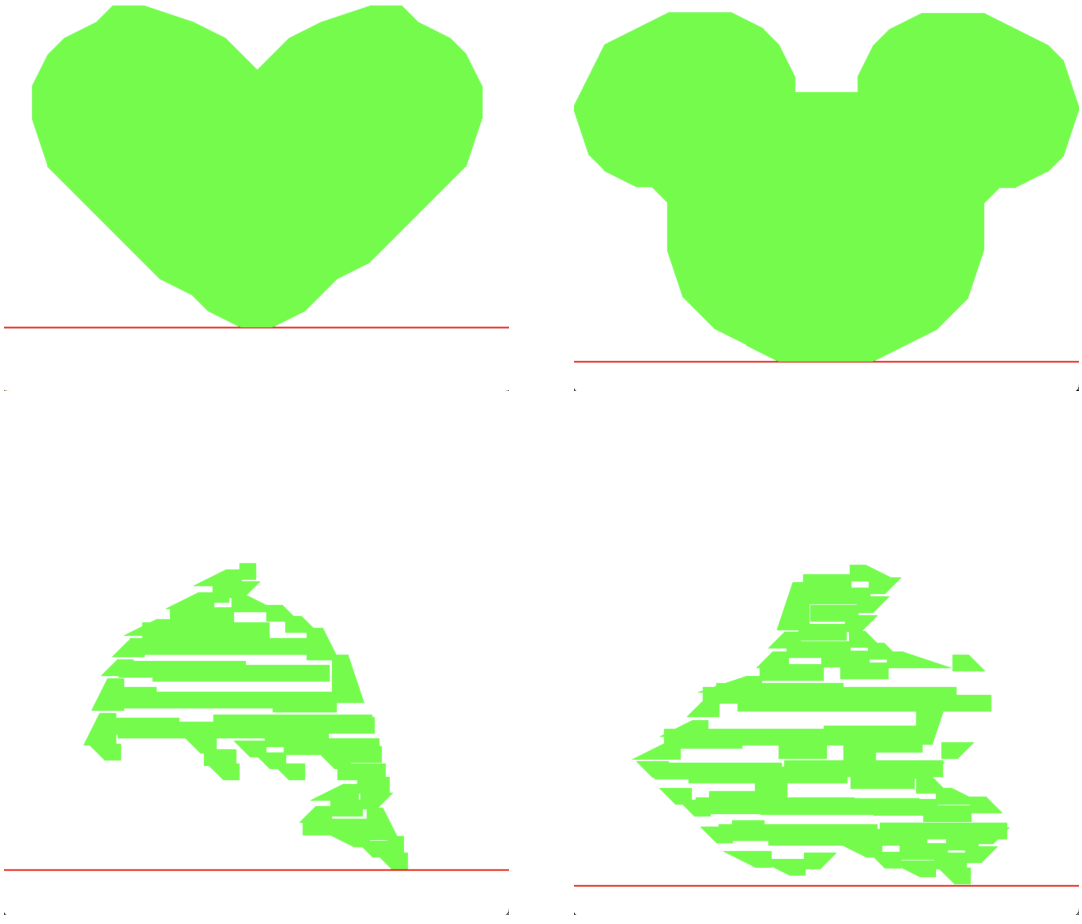

Figure 5: Stability Analysis

5 Enlarged Brick Set

Based on the previous research, we will use a larger brick set, which includes curved bricks and plates, to construct LEGO® models; see Figure 6.

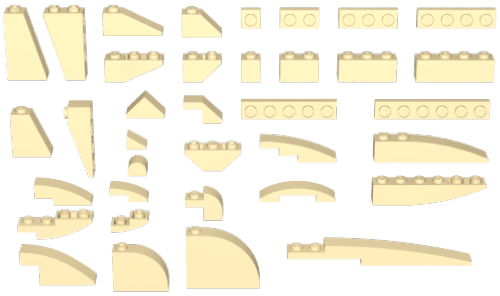

Figure 6: Enlarged brick set

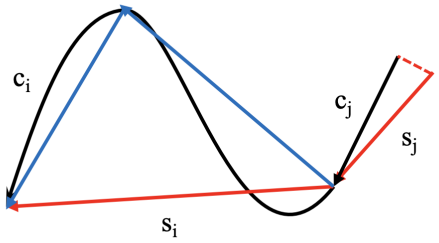

Figure 7: Convex detection

Compared with previous method, we will add a new factor, denoted by $\mathcal{L}_s$, to our objective function to improve the smoothness of generated models. We define $\mathcal{S}$ as a connection set. The edge tuple $(s_i, s_j) \in \mathcal{S}$ if s_i and s_j have a same endpoint and the same direction. These two edges also need to meet an another condition. We first project s_i and s_j onto the contour and obtain two edges, denoted by c_i and c_j. The length of c_i and c_j should not exceed the length of s_i and s_j much, otherwise this factor will wipe out many details to get better smoothness; see Figure 7, blue lines can be added to $\mathcal{S}$ while red lines cannot. Then, for each $(s_i, s_j) \in \mathcal{S}$, we calculate the difference in the angle between s_i, s_j and the horizontal direction, denoted by a_{ij}.

$$\mathcal{L}_s = \sum_{(s_i,s_j)\in\mathcal{S}} x_i x_j a_{ij} \tag{10}$$

Finally, we add $\mathcal{L}_s$ to our previous objective function and set a weight w_s for it.

$$\mathcal{W} = w_d\mathcal{L}_d + w_v\mathcal{L}_v + w_s\mathcal{L}_s \tag{11}$$

We can use this objective function to obtain LEGO® models with better smoothness. The results will be shown in following section.

6 Grid Alignment

To make graph optimization more effective, we apply grid alignment to relocate some feature points to their nearest grid points. First, we detect feature points from input shapes and obtain sharp points and the end points of nearly vertical and horizontal lines. Then, we apply global alignment and local alignment to the contour successively.

- *Global alignment.* After detecting feature points on the contour, we aim to minimize the sum of distance between feature points and their nearest grid points. In this step, we only consider scaling and translation. We can transfer this problem to a least square problem. After obtaining the scaling and translation, we apply them to the whole contour.

- *Local alignment.* In this step, we aim to relocate the feature points to their nearest grid points with least distortion. We will apply slight rotation, scaling and translation to the contour segments between each pair of feature points p_i and p_{i+1} and drag them to the nearest grid points t_i and t_{i+1}. To achieve this, we set a as a rotation angle, s_x and s_y as scaling, d_x and d_y as translation and form an affine transform matrix T_i:

$$T_i = \begin{bmatrix} s_x cos(a) & -s_x sin(a) & d_x \\ s_y sin(a) & s_y cos(a) & d_y \\ 0 & 0 & 1 \end{bmatrix} \tag{12}$$

Next, we can form two equations and use them to express s_x, s_y, d_x and d_y with a.

$$T_i \begin{bmatrix} (p_i)_x \\ (p_i)_y \\ 1 \end{bmatrix} = \begin{bmatrix} (t_i)_x \\ (t_i)_y \\ 1 \end{bmatrix} \tag{13}$$

$$T_i \begin{bmatrix} (p_{i+1})_x \\ (p_{i+1})_y \\ 1 \end{bmatrix} = \begin{bmatrix} (t_{i+1})_x \\ (t_{i+1})_y \\ 1 \end{bmatrix} \tag{14}$$

Then, we construct our objective function to show the shape distortion:

$$Loss = a^2 + s_x^2 + s_y^2 \tag{15}$$

Finally, we use gradient descent to solve this optimization problem and apply each T_i to the contour.

After applying grid alignment, we can find that the contour becomes more smooth and it will be more efficient in graph optimization. The images below show the results of grid alignment; see Figure 8.

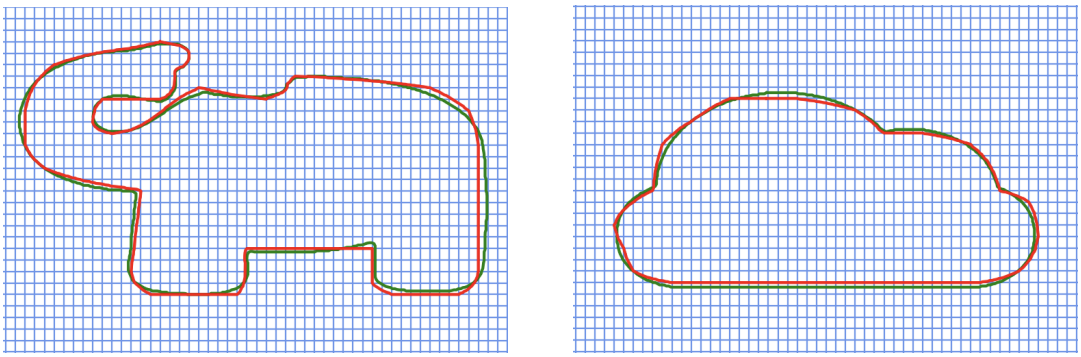

Figure 8: Grid alignment

7 Results

Below are some generated LEGO® models using our enlarged brick set. Smooth optimization and grid alignment have applied to all results; see Figure 9.

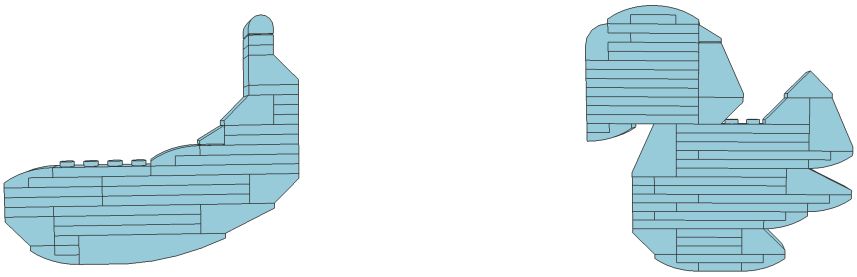

Figure 9: Generated models

8 Future Research

In the future, we will consider constructing a LEGO® model in different orientations. In that case, slope and curved bricks are allowed to be both sides of a LEGO® model. This will make models more smooth and expressive.

References

[1] Lambrecht, B. Voxelization of boundary representations using oriented lego plates. *University of California, Berkeley* (2006).

[2] Liu, R., Deng, K., Wang, Z., and Liu, C. Stablelego: Stability analysis of block stacking assembly. *arXiv preprint arXiv:2402.10711* (2024).

[3] Zhou, M., Ge, J., Xu, H., and Fu, C.-W. Computational design of lego® sketch art. *ACM Transactions on Graphics (TOG) 42*, 6 (2023), 1–15.