Reconstructing Creative Lego Models

George Tattersall[0000−0003−0636−0157], Dizhong Zhu[0000−0003−4086−7293], William A. P. Smith[0000−0002−6047−0413], Sebastian Deterding[0000−0003−0033−2104] and Patrik Huber[0000−0002−1474−1040]

University of York, York, UK
gedtattersall@gmail.com
{dizhong.zhu,william.smith,sebastian.deterding,patrik.huber}@york.ac.uk

Abstract. Lego is one of the most successful toys in the world. Being able to scan, analyse and reconstruct Lego models has many applications, for example studying creativity. In this paper, from a set of 2D input images, we create a monolithic mesh, representing a physical 3D Lego model as input, and split it in to its known components such that the output of the program can be used to completely reconstruct the input model, brick for brick. We present a novel, fully automatic pipeline to reconstruct Lego models in 3D from 2D images; A-DBSCAN, an angular variant of DBSCAN, useful for grouping both parallel and anti-parallel vectors; and a method for reducing the problem of non-Manhattan reconstruction to that of Manhattan reconstruction. We evaluate the presented approach both qualitatively and quantitatively on a set of Lego duck models from a public data set, and show that the algorithm is able to identify and reconstruct the Lego models successfully.

1 Introduction

Lego is one of the most successful toys in the world today. Its affordances have made it widely recognised as a means of expressing, fostering, and studying creativity [1, 2]. Using Lego models to study human creativity often requires scanning the physical models in some form in order to be able to formally reconstruct

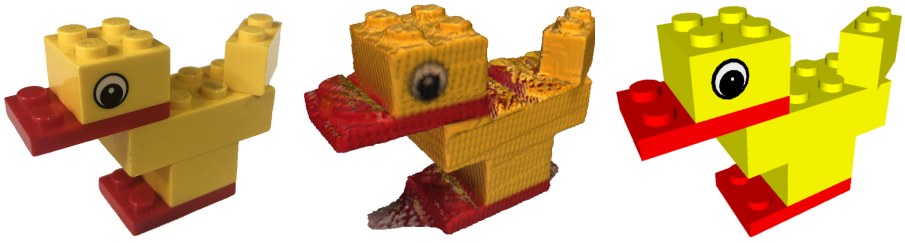

Fig. 1. From left to right: real world Lego duck model, monolithic visual hull of duck from 2D image data, resulting reconstruction of the proposed approach.

and analyse them. One recent example is Ferguson et al. [3, 4]. To explore novel methods for computationally assessing human creativity, they created a "Lego duck task" which involves constructing diverse ducks from the same set of six Lego bricks. Their aim was to use computational formalisations of dissimilarity to assess which ducks were more divergent, novel, and thus, creative. Such computational analysis logically requires a computational representation of the Lego models. To this end, Ferguson et al. created a corpus of duck models, archived as multiview images of the creations [4]. They used a variational auto-encoder on the images to generate neural representations of the ducks. While their results were promising, such a machine learning approach limits possible formal analyses, as the computational representation remains 'opaque' to human inspection, which is why they recommended future work to explore the generation of more transparent, human-readable computational representations of models.

In this paper, we aim to go further than current 3D scanning methods by identifying the arrangement of a set of known Lego components to obtain a complete, brick by brick, 3D reconstruction of the original scanned Lego model. We propose a fully automatic pipeline to reconstruct semantic 3D Lego models from 2D images, comprising the following novel ingredients: angular DBSCAN, a novel variant of the clustering algorithm DBSCAN [5] using angular displacements that is suitable for grouping both parallel and anti-parallel vectors and a method for reducing the problem of non-Manhattan reconstruction to that of Manhattan reconstruction. Figure 1 shows an example model, the reconstructed visual hull, and the output of our proposed method, consisting of a 3D arrangement of the individual blocks that make up the original model.

2 Related Work

Although little research has been undertaken on the programmatic reconstruction of Lego models, there is a large area of research on identifying and reconstructing cuboids and buildings from point clouds. There exists also a small body of work on various mathematics of Lego permutations, Lego construction and Lego detection. No attempts have been made to reconstruct Lego *models* in 3D.

Manhattan-World Reconstruction: Kim and Hilton [6] showed in 2015 a pipeline that detects and reconstructs cuboids from multiple stereo pairs of spherical images, under the assumption of a Manhattan-world. They describe dealing with planes that are too similar to each other, eliminating unreliable planes and plane intersection refinement. They calculate the similarity between sets of parallel planes by: first calculating overlap and difference in position; merging similar planes by taking a parallel bounding box; and finally replacing the original planes with the newly created one. Their plane intersection refinement aims to create perfect cuboids from sets of noisy, Manhattan compliant planes by using intelligently picked tolerances to edit the plane parameters so that they exactly meet at corners to form cuboids.

Li et al. [7] deal with noisy input data in the form of 3D point clouds extracted from aerial photography. They present a *generate and select strategy*, formulated

as a labelling problem, and solve it using a Markov Random Field. Their solution appears to work well for representing inputs as a densely packed set of blocks. In their setup, there is also the problem of missing walls, which is the identification of internal walls missing from the input data, but possibly recoverable from auxiliary data - such as 2D images. They also provide some insight into a solution to the problem of missing walls. They use Hough transforms to identify cuboid faces from their initial input images, which would be impossible to extract via analysis of 3D surfaces alone.

Non-Manhattan-World Reconstruction: With a point cloud as input, Cao and Wang [8] have developed a robust way to find cuboids of any rotation with impressive accuracy. They first seed cubes randomly in the point cloud and then use a Levenberg–Marquardt algorithm [9], paired with their method for calculating distance between each cube and the point cloud to optimise their output. They expand each cube to fit the ground truth as best as it can, paired with a smoothing variable that combines cuboids which it feels should not be disjoint. It seems to work best on inputs which extend out in *all* dimensions, but can still perform well with some changes to their smoothing variable if this condition is not met. Schnabel et al. [10] produced breakthrough research in the area of 3D reconstruction of arbitrary models using their RANSAC [11] approach. Their algorithm runs a single subroutine with 5 different types of shapes, attempting to find areas of the input point cloud where they each fit best. The research stresses that although their approach is fast and robust it does not find these shape proxies for every part of the input surface, leaving holes and overlaps between the disjoint shapes which have been found.

Lego Theory: There is a body of research (Durhuus and Eilers [12], Nilsson [13], Abrahamsen and Eilers [14]) that is focused on investigating Lego under certain heavy constraints such as Manhattan world assumption or known heights and widths of input models, with the aims of investigating the permutations, complexity, entropy and other properties of Lego models. There seems to be no exact formula to follow for computing the number of Lego permutations, especially with non-Mahattan models. The research does, however, show an exponential growth when computing the number of models which can be made using n Lego. Because of this, it is unlikely that any kind of brute force method would be feasible for Lego detection, construction or reconstruction without introducing some constraints on the search space.

Lego Detection: There have been many attempts to recognise and categorise individual Lego and Duplo bricks from 2D images using computer vision, deep learning and machine learning methods, including the work of Mattheij [15], the software of Nguyen [16] and West [17], and in material from Lego Education[1]. These are all able to do well at identifying individual bricks in images with only the target brick and a solid background. However, none of them leverage any advantage that a 3D input can provide, or attempt to process models with multiple

[1] https://education.lego.com/en-us/lessons/ev3-cim/make-a-sorting-machine

connected bricks. Chou and Su [18] attempt to identify entire models as opposed to single bricks. There is no reconstruction involved in their research, however they develop a pipeline using a Microsoft Kinect and RANSAC to extract their target and then use a convolution neural network to identify whether the model is equivalent to another.

Lego Construction: The *Lego Construction Problem* was set by Lego to answer the question: "Given any 3D body, how can it be built from LEGO bricks?" [19]. The Lego Construction Problem looks for ways to construct any 3D input using Lego, so that the surface is similar enough to the original. While the Lego Construction Problem is not entirely applicable to the problem at hand, Kim et al. [19] has some discussion on the difficulties of dealing with the problem's large search space and they evaluate various methods for tackling this, such as using brick colour to aid in detection and ways of reducing the possible number of Lego that can be detected. Peysakhov and Regli [20] feature in the previous paper showing a method of unambiguously representing and abstracting a Lego model as a graph. Their graph representation is a directed graph where each node is a unique Lego, and each edge describes which studs connect Lego.

3 Method

In this section, we present our novel pipeline to reconstruct Lego models in 3D. 3D simplifies representation and provides invariance, modelling the intrinsic structure of our models. The low dimensionality is also crucial for measuring creativity, since we can only hope to capture a few hundred models from a few hundred people, so learning models from this data needs to be highly sample efficient to prevent overfitting.

The underlying idea of our pipeline is to reduce all of the Lego identification requirements to that of the simpler identification of Manhattan compliant meshes. Manhattan compliant Lego models have strict rules on how they can be constructed, leading to a finite number of possible models. By reducing the problem to a Manhattan world we aim to take advantage of this inherent structure to deduce the full reconstruction in an efficient manner. Together with the lack of available training data, this makes the problem an ideal fit for a more classical computer vision approach, rather than learning based methods. Furthermore, our approach allows full control of the types and number of Lego bricks used.

In the first step, we generate the visual hull of a model from the 2D input images. Next, we use any prior knowledge about the colours or patterns on the Lego to extract any information we can about their locations. The next three steps extract face information from the monolithic mesh, use the information to reduce non-Manhattan compliant areas of the mesh to Manhattan ones, and then perform an adaptation of the *Block World Reconstruction* of Kim and Hilton [6] to identify remaining Lego. The final stage fixes any inaccuracies by snapping Lego together as they would be in the real world. An overview of the pipeline is shown in Figure 2. The following sections will describe the steps in detail.

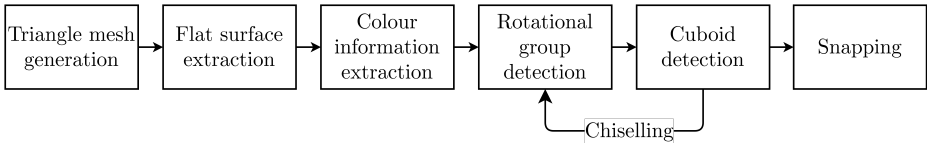

Fig. 2. Flowchart of the proposed pipeline.

3.1 Triangle Mesh Generation

The input to our algorithm consists of a set of 2D RGB images of a physical Lego model, captured from viewpoints that uniformly sample a circle around the vertical Y-axis. Shape-from-X methods based on feature matching, such as structure-from-motion e.g. COLMAP [21, 22], fail on such images due to the lack of texture on the lego bricks. Instead, we assume calibrated cameras and apply a standard space carving method [23, 24] to these images to extract a volumetric visual hull. We convert the volumetric representation to a triangle mesh by applying the marching cubes algorithm (Figure 3 (a)). Finally, on the triangle mesh we apply Laplacian smoothing to reduce the roughness of the surface created by the voxel discretisation (Figure 3 (b)).

3.2 Flat Surface Extraction

To get any kind of meaning from the monolithic mesh, we must extract face information. To do this we perform a discrete differentiation on the monolithic mesh, labelling each triangle of the mesh with the dihedral angles between itself and each of its connecting triangles. From this we find which groups of triangles form flat surfaces by removing any triangles which have one or more dihedral angles over a certain threshold (Figure 3 (c) and (d)). We name the set of all flat surfaces F where each element is a set of triangles which make up a flat surface.

This method holds up very well for bricks due to the input meshes only containing right angles or flat surfaces. The Laplacian smooth used in the pre-processing also assists in ensuring robust identification of flat surfaces.

3.3 Colour Information Extraction

Lego plates are short in height relative to a standard brick. This means that the planar faces of the brick sides may be below the resolution of the generated mesh. If that is the case, we handle plates explicitly using a separate process. To simplify this task, we assume that plates share the same colour and that this is different from the colour of the other bricks.

We extract all triangles from the mesh with the chosen colour, indicating a plate, and compute the bounding box of each cluster of triangles. We can then remove the associated triangles from the mesh and use the found cuboid faces in the *Cuboid Detection* of Section 3.5.

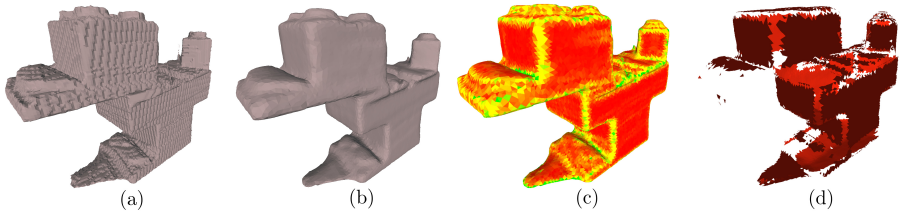

Fig. 3. Left to right: (a) input without any smoothing; (b) input after a Laplacian smooth; (c) mesh differentiation with yellow indicating areas of high curvature, and red indicating areas of low curvature; (d) groups of flat surfaces.

We can also use a similar filter to identify patterned bricks, and to extract the pattern information. This information can later be used to apply textures to appropriate bricks once found.

3.4 Rotational Group Detection

To identify the areas of the monolithic mesh which break the Manhattan assumption we first need to identify a set, P_F, of subsets of F such that each subset, p, contains only flat surfaces which are parallel to each other. We then need to find our set of rotational groups, $\Pi_F = \{(p_0, p_1) | p_0, p_1 \in P_F, p_0 \perp p_1)\}$.

By definition all elements of P_F should be disjoint with all other elements in P_F as well as $\bigcup_{p \in P_F} p$ being equal to F. We can formulate this as a clustering problem on the average normal of elements in F. By doing this we can leverage existing clustering algorithms [25].

Density-based spatial clustering of applications with noise [5] (DBSCAN) is a popular clustering algorithm in cases where the number of clusters to be found is unknown, unlike approaches such as k-means [26]. The fact that cuboids are made up of parallel and perpendicular sides means that DBSCAN clusters should all have a high density where we need them to, e.g. at least two flat surfaces being anti-parallel. This, in addition to its natural noise removal property, makes it a good choice for this application. DBSCAN is often used for clustering points in 3D euclidean space, for which the metric is almost always euclidean distance. While this would cluster together parallel vectors effectively, it would always fail to cluster anti-parallel vectors.

Angular DBSCAN: We subsequently develop *Angular DBSCAN* (A-DBSCAN), using $\left|cos(2\theta + \frac{\pi}{2})\right|$ as the density metric, as opposed to the usual euclidean distance or cosine distance, with the aim of removing the erroneous results produced by vector magnitude and anti-parallelism (Figure 4). We modify the prerequisite definitions of DBSCAN from Schubert et al. [27] as follows:

– A vector V is a 'core vector' if at least $minVecs$ vectors have an angle less than ϵ between them and V, including V.

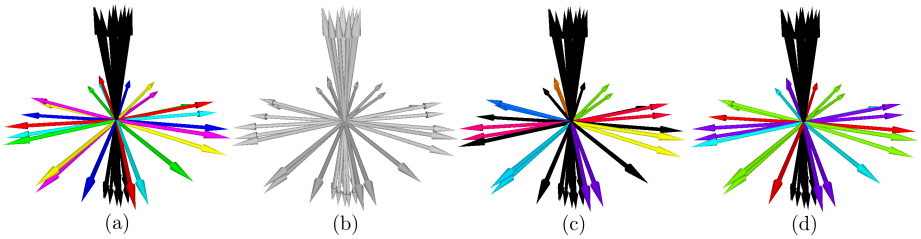

Fig. 4. From left to right: (a) vectors from six Lego with ground truth groupings; (b) vectors with no groupings (input to DBSCAN); (c) vector groupings after euclidean displacement or cosine distance DBSCAN; (d) vector groupings after Angular DBSCAN. $\epsilon = \frac{\pi}{20}$. Each colour represents a cluster.

- A vector W is a 'direct neighbour' of a core vector, V, if vector W has an angle less than ϵ between V and itself.
- A vector W is a 'neighbour' of V if there is a path from V to W, where each vector in the path is a 'direct neighbour' of the previous vector.
- All other vectors are noise.

Here, with *angle between two vectors*, we mean the absolute acute angle, $\left|\cos(2\theta + \frac{\pi}{2})\right|$. The modified algorithm then becomes:

1. Identify the neighbouring vectors of every vector, and identify the core vectors.
2. Identify the connected components of core vectors on the neighbours graph, ignoring all non-core vectors.
3. Assign each non-core vector to a cluster within ϵ range, otherwise assign it to noise.

The output contains clusters of vectors identified using A-DBSCAN which, when mapped back to their original flat surfaces, split F into the elements of P_F. All other vectors will be returned as noise, and their original flat surfaces can be assigned as such.

Angular Pairing: At this point, given our input bricks each have 2 sets of parallel faces which are perpendicular to the Y-axis, $\#P_F \leq 12$. This makes it viable to run a brute force search on P_F, identifying perpendicular pairs, removing them from the search as we progress. The identified pairs make up our rotational groups Π. See Figure 5 for an example.

3.5 Cuboid Detection

Manhattan World Reduction: The first stage of detecting cuboids is to use the previously identified rotational groups to duplicate the monolithic mesh into

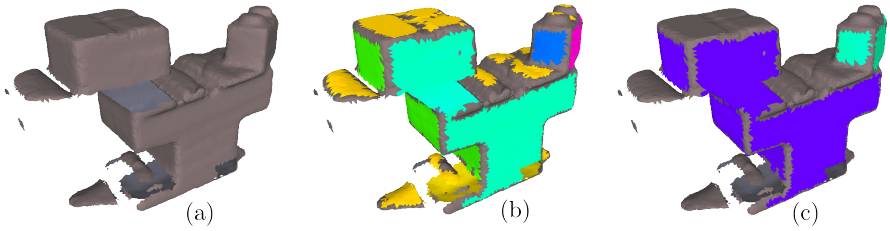

Fig. 5. From left to right: (a) input mesh with red removed; (b) A-DBSCAN clusters by colour; (c) rotation groups Π, by colour.

$\#\Pi$ copies, followed by rotating each copy such that all flat surfaces in any given rotational group are Manhattan compliant within one of the copies. Splitting the meshes also allows the rest of the algorithm to run in parallel.

Brick Face Extraction: To extract the planar faces of our input, we take inspiration from part of Kim and Hilton's *Block world reconstruction* [6] as we act on the now Manhattan aligned mesh sections. In contrast to Kim and Hilton, we can disregard the scene scale parameter used in their approach as we know exactly the sizes we need from the sizes of Lego. In place of their 2D to 3D plane reconstruction methods, the prior steps in this section have already generated Manhattan compliant flat surfaces. If we name the set of flat surfaces in the current rotational group Ω, then for each item, Ω_i, we can look at its normal vector and calculate which global axis it aligns closest to, one of X, Y or Z. We call the identified axis A_i.

To extract plane segments, if we were to simply take a bounding box of each Ω_i, we would be affected by the absence of triangles which were on the connection between faces during the previous mesh differentiation, resulting in a plane segment which would be smaller than the face it is meant to represent. Instead we can compare Ω_i to the original input mesh and dilate Ω_i to rectify the previous erosion caused by removing areas of high curvature. Projecting the dilated Ω_i onto A_i results in a good silhouette of the model's face, from which we can then take a fuller bounding box (see Figure 6). Doing this for all Ω_i gives us a set of axis-aligned plane segments, S_p. Subsequently, we merge similar planes as in Kim and Hilton, reducing the noise caused by rough scans.

Hidden Face Extraction: We define *hidden* faces to be faces of Lego bricks which are hidden in the internal space of the input visual hull, and so cannot be explicitly seen in images or extracted from the input mesh. We can however see faint lines defining these planes from the outside. This occurs when two bricks of the same colour are placed next to each other. In this part, we aim to reconstruct the faces which correspond to these faint lines, to remove ambiguity in our reconstructions.

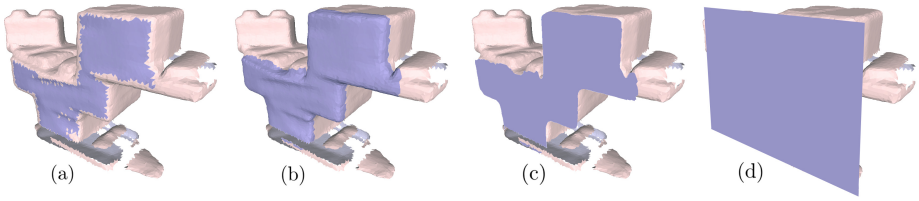

Fig. 6. From left to right: (a) input mesh with flat surface; (b) dilated flat surface against mesh; (c) flat surface projected to the identified axis A_i; (d) flat surface bounding box projected to A_i.

In Li et al. [7], Hough transforms are used to identify lines on the surface of their images, and to then extrapolate a face, perpendicular to the surface on which the line was identified. Instead of using Hough transforms we use Line Segment Detector (LSD) from Grompone von Gioi et al. [28], a more modern approach of line detection, which performs well on low contrast surfaces, has a linear time performance, and requires no parameter tuning.

To find connecting lines between two bricks at the same elevation, we apply LSD to each of the n input images of the model, and filter out any non-vertical lines. Next, we check that the pixels either side of the lines, within a tolerance, have a predominantly similar hue. Finally, to remove detection of corners, we can check that there is a low standard deviation in the hue of the pixels on either side of the line. Once we have our lines in 2D space, we tag them with a colour in the images in which they were found, and then retrieve the 3D information once the images have undergone the visual hull process (see Figure 7 (a)). Because we know that the tagged triangles all represent vertical lines, a 3D line can be created from the minimum and maximum of the y coordinates of the tagged triangles and the average of the x and z coordinates.

Using our identified set of 3D lines, we can find the extracted face which is most consistently closest to the line. Taking the identified face's normal vector, we can extrude a new plane segment from the 3D line in the direction of the normal by 16 units. Finally we perform a merging of new quads which are close to each other or overlapping, replacing two close quads with their bounding box. We chose to extrude by 16 units, as that will be adequate for sides of length 16 and 8, and will allow for 32 unit faces as long as there is a visible line on the other side of the brick which can be merged. Figure 7 depicts the process.

Column Identification and Cuboid Detection: To complete the individual identification of a Lego, $\mathbb{L}$, we first identify pairs of parallel planes from the set of axis-aligned plane segments S_p which have a significant overlap as well as a distance between them equal to either the length or width of $\mathbb{L}$. We next build columns with width and length equal to that of $\mathbb{L}$. Columns are identified by comparing each previously identified X-aligned pairing with each identified Z-

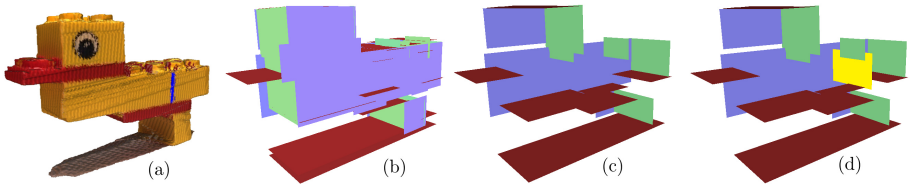

Fig. 7. From left to right: (a) input mesh with detected seam superimposed in blue; (b) extracted faces from input mesh; (c) inside of extracted faces, showing no internal face; (d) extracted hidden face highlighted yellow.

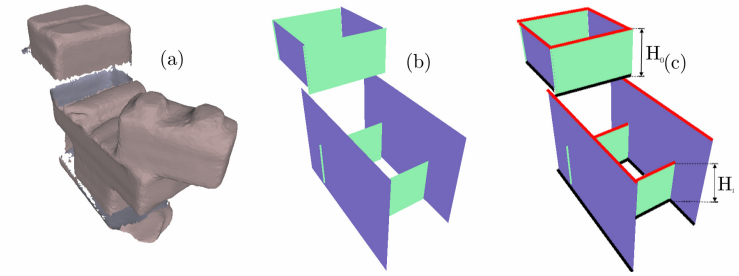

Fig. 8. From left to right: (a) input mesh; (b) identified 2x2 brick columns composed of two green and two purple planes each; (c) red and black edges indicate the maximum and minimum y coordinates of each plane, H_0 and H_1 are the distances between the minimum of the maxima y coordinates and the maximum of minima y coordinate of each column.

aligned pairing. We have a column when: we find two pairings where each plane segment intersects both plane segments of the other pairing, within a tolerance; the width and length of the column equal that of $\mathbb{L}$; $H \geq height(\mathbb{L})$ where H is the distance between the minimum of the maxima y coordinates of each plane and the maximum of the minima y coordinates of all planes in the column Figure 8 depicts this process.

The final step of identifying cuboids is to take each identified column and find where the cuboid (or cuboids) are positioned within the column. Because of the lack of plane information between vertically connected bricks we can only identify top level or bottom level cuboids. We choose to find the top-most cuboid in a column, and to do so we search through all planes which have an identified axis A_i of Y, taking note of all which have a substantial overlap with the cross-section of the column. We then select the top most Y-plane.

At this stage we have five of the six faces of a Lego $\mathbb{L}$ identified, and therefore can extrapolate the final face by copying the top Y plane and subtracting the height of the Lego. Together with the rotation from the A-DBSCAN, $\mathbb{L}$ has now been fully identified, and we can continue with the next Lego.

3.6 Chiselling

To continue identifying the remaining bricks we can *chisel* through the model by removing input mesh information close to identified bricks, and injecting extrapolated bottom planes into the model where the identified top level Lego were found. This can be done recursively until we identify all bricks, or it does not find any new brick on a pass. At the end of the process we can report the exact number of Lego found.

3.7 Snapping

Various inaccuracies occur during the reconstruction, mostly due to inaccuracies in the input mesh. In an effort to improve upon this, we implement a snapping algorithm that loops over each Lego stud, searching for connections of the opposite type, and matching the closest stud as long as it is within $minDist$, where $minDist$ is set to the height of the thinnest Lego.

Doing this produces a graph similar to Peysakhov and Regli [20], connecting Lego by their studs. We can then use this graph to offset the position of each Lego to better snap to each other, while maintaining relative rotations.

4 Evaluation and Results

4.1 Angular DBSCAN

Choosing the correct value for the threshold ϵ is very important. If ϵ is too large, all vectors will be assigned to a single group. While, if ϵ is too small, each vector will be assigned to either its own group or noise, depending on the value of $minVecs$. Since most Lego are cuboid, each group of vectors should be composed of only orthogonal vectors. This sets a maximum ϵ of $\frac{\pi}{4}$ before ϵ covers all possible vectors at least once. To identify the optimal ϵ for the proposed A-DBSCAN on Lego we generated 6 groups of 6 orthogonal vectors to simulate the vectors produced by 6 Lego. We then apply a random rotation around the y-axis to each group to simulated rotations of Lego, and finally we add slight random noise to each vector. We then performed 1000 runs of the A-DBSCAN on these vectors for each value of epsilon between $\frac{\pi}{40}$ and $\frac{\pi}{4}$ in steps of $\frac{\pi}{20}$. We also simulate chiselling by removing vector groups which have been correctly identified, and performing a recursive step on the remaining vectors.

From Figure 9, we can see that any $\epsilon \leq \frac{\pi}{20}$ does not identify nearly enough vectors correctly, while any $\epsilon \geq \frac{\pi}{20}$ shows a recursion depth which we wouldn't expect for groups of vectors this dense. To ensure we have not only grouped vectors together which were grouped together in the input, but also make sure we don't have too few groups, we need to find the ϵ which maximises both the mean percentage of vectors grouped together correctly, as well as the mean recursion depth. We highlight $\epsilon = \frac{\pi}{20}$ as the optimal result, with a mean percentage of vectors group together correctly of 98.37% and mean recursion depth of 1.856 (which implies there will usually be at least 2 groupings). These results are what we expect from this many vectors.

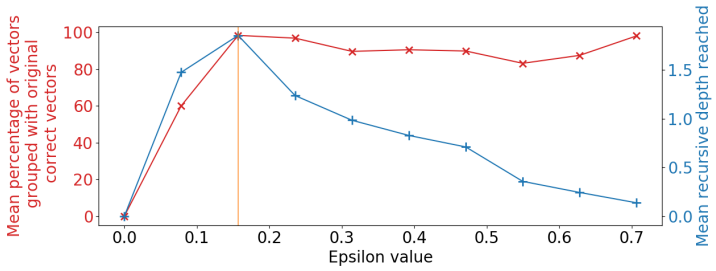

Fig. 9. DBSCAN epsilon tested for percentage of 'successful' groupings and the mean recursive depth reached.

4.2 Lego Reconstruction

Dataset: We evaluate our proposed approach on the data of Deterding et al. [3, 4], which has been collected with the goal of evaluating human creativity using Lego. It consists of a set of 162 Lego ducks, composed of the same 6 Lego. Each model is photographed on a turntable, resulting in 24 images.

Identified Bricks: First, we evaluate the percentage of inputs successfully converted into six unique Lego. Overall, roughly 44.3% of the models were completely reported as being identified correctly. 75% of 2x1 bricks were identified, 73.6% of 2x2 bricks, 84.6% of 2x3 slates, and 55% of 2x4 bricks. 140 out of the 162 ducks were used for the evaluation.

From the above results, we see that the identification of 2x4 bricks is considerably lower than any other Lego. This may be due to the large volume of the brick providing much more space to be affected by various forms of noise in the system. The larger faces of the 2x4 brick, paired with it often being used as a central Lego to a duck (as oppose to a top, bottom or side Lego) mean that it normally has more of its face area obscured by other bricks. If the attached bricks are not correctly identified then it may cause the central 2x4 brick to remain unidentified.

The overall relatively large number of non-perfectly reconstructed models is caused mainly by considerable artifacts of the reconstructed volumetric visual hull, which is an issue on nearly half of the reconstructed meshes. This is due to the test data's scan resolution, as well as the visual hull algorithm's resolution. Improving that would benefit all subsequent steps, and potentially removing the need of using colour information to identify plates (Section 3.3).

Mesh Similarity: Second, we compare the final reconstructed Lego models with the reconstructed visual hull from the 2D input images. We first apply rigid iterative closest point [29] to align the two meshes. Then, for every vertex, v, in our input mesh, of which there are hundreds of thousands, we find the smallest distance between v and the output mesh (a mesh with only hundreds

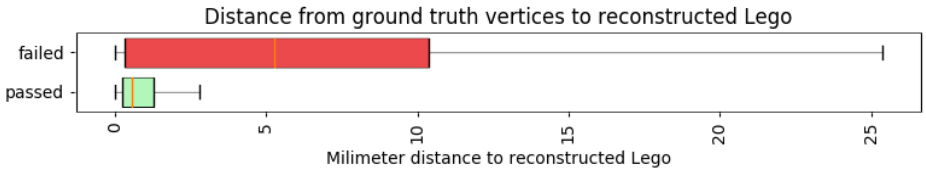

Fig. 10. Summary of vertex-wise distance to ground truth analysis.

of vertices). We then plot the range, interquartile range and mean of the set of ducks where we found six bricks, and the set where we found any less. This is shown in Figure 10. We can see that there is a correlation between the algorithm reporting a pass or fail, as described in the previous subsection, and the distances between input and output meshes being small and large respectively.

For more context, a 1x1 Lego (smaller than any of our Lego) would be 8mm on both of its sides, so when the algorithm reports a pass the output mesh will rarely be more than a 1x1 Lego away from the input, while it can leave gaps of over 5 times as far when it has reported a fail. An advantage of our algorithm being conservative in its searching, only identifying Lego if there is concrete reasoning to do so, as well as the bricks used to build the model being known beforehand, is that it allows us to know when we have passed fairly consistently. These results show that the system should rarely produce a false positive.

Visual Analysis: Finally, we qualitatively analyse the proposed method. Figure 11 shows example outputs from each stage of the algorithm, with the final result. In row (A), we show an example duck with hidden faces between vertically connected bricks. The chiselling was able to successfully identify all blocks, and it works well on most ducks. Row (B) shows a duck with multiple rotational groups, which were successfully detected, and subsequently the *Cuboid detection* is performed on both groups separately. Row (C) shows an example of a 2x2 and a 2x4 brick next to each other, and the proposed method successfully being able to identify and separate both bricks.

While our method performed very well on approximately half of the models in the dataset, there are two main reasons why it did not work well on the others. First, the initial mesh generation does not produce a high enough resolution mesh to capture the sides of plates or small holes in models, or left out large chunks of the model caused by a too large baseline between the 2D input images. Second, some ducks use an unorthodox construction method where the *sides* of plates would be rotated and snapped *between* studs (sometimes called 'pony-ear' or 'pony-leg' connections), which our method is not able to process. An example of such a model is shown in row (D).

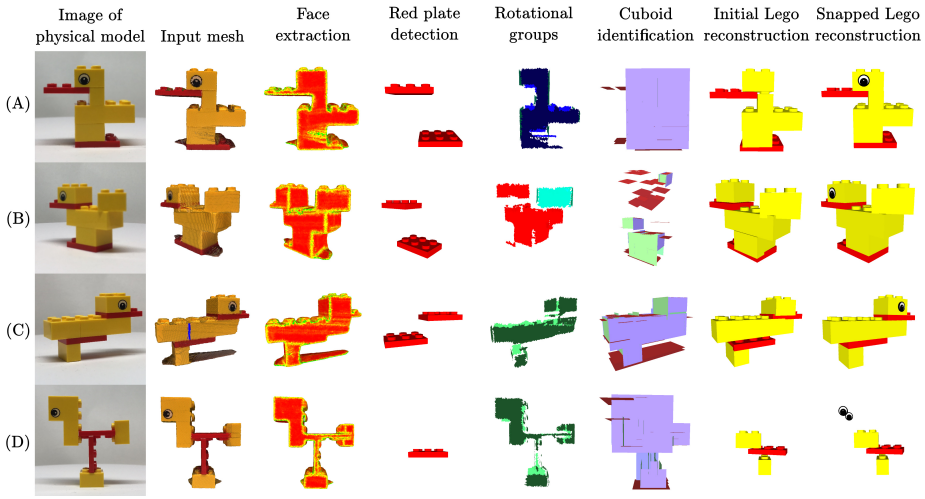

Fig. 11. Example reconstructions. Rows (A)-(C): successful reconstructions, row (D): failed reconstruction with pony-ear connections that our method is not able to process.

5 Conclusion

In this paper, we have demonstrated our novel pipeline that takes in a tightly coupled, non-Manhattan compliant Lego mesh and identifies the exact positions of each Lego used to construct it. We introduced Angular DBSCAN, A-DBSCAN, to reduce the non-Manhattan reconstruction to that of a Manhattan one, as well as a method combining image and mesh data to calculate hidden plane information. To the best of our knowledge, ours is the first piece of work in literature that reconstructs Lego models in 3D, and decomposes them into their exact bricks. The results show large potential for use in practise, for example for studying human creativity, but also in the broader study of 3D shape recognition.

The proposed system is conservative in its approach, only identifying Lego if there is concrete reasoning to do so. This, paired with the fact it takes the bricks used in the models construction, allow it to identify successfully whether or not it has passed in its identification of a model's construction with very high accuracy. The main limitation on what models can be used as input is that the current system cannot deal with models which contain 'pony ears' or 'pony leg' connections. This challenge will be tackled in future work. Finally, future work could be to explore the use of deep learning for parts of our pipeline.

Acknowledgements

This work was supported by the Digital Creativity Labs funded by EPSRC/ AHRC/ Innovate UK (EP/M023265/1), and the PLAYTrack project, supported by a research grant from the LEGO Foundation.

References

1. Gauntlett, D.: The lego system as a tool for thinking, creativity, and changing the world. Lego studies: Examining the building blocks of a transmedial phenomenon (2014) 1–16
2. Pike, C.: Exploring the conceptual space of lego: Teaching and learning the psychology of creativity. Psychology Learning & Teaching **2** (2002) 87–94
3. Deterding, S.: Brains, bricks, ducks, AI: New ways of assessing human and computational creativity. osf.io, doi:10.17605/OSF.IO/2SGM9 (2020)
4. Ferguson, M., Deterding, C.S., Lieberoth, A., Malmdorf Andersen, M., Devlin, S., Kudenko, D., Walker, J.A.: Automatic similarity detection in lego ducks. In: ICCC'20: Eleventh International Conference on Computational Creativity, Association for Computational Creativity (ACC) (2020)
5. Ester, M., Kriegel, H., Sander, J., Xu, X.: A density-based algorithm for discovering clusters in large spatial databases with noise. In Simoudis, E., Han, J., Fayyad, U.M., eds.: Proceedings of the Second International Conference on Knowledge Discovery and Data Mining (KDD-96), Portland, Oregon, USA, AAAI Press (1996) 226–231
6. Kim, H., Hilton, A.: Block world reconstruction from spherical stereo image pairs. Comput. Vis. Image Underst. **139** (2015) 104–121
7. Li, M., Wonka, P., Nan, L.: Manhattan-world urban reconstruction from point clouds. In Leibe, B., Matas, J., Sebe, N., Welling, M., eds.: Computer Vision - ECCV 2016 - 14th European Conference, Amsterdam, The Netherlands, October 11-14, 2016, Proceedings, Part IV. Volume 9908 of Lecture Notes in Computer Science., Springer (2016) 54–69
8. Cao, C., Wang, G.: Fitting cuboids from the unstructured 3d point cloud. In Zhao, Y., Barnes, N., Chen, B., Westermann, R., Kong, X., Lin, C., eds.: Image and Graphics - 10th International Conference, ICIG 2019, Beijing, China, August 23-25, 2019, Proceedings, Part II. Volume 11902 of Lecture Notes in Computer Science., Springer (2019) 179–190
9. Levenberg, K.: A Method For The Solution Of Certain Non-linear Problems In Least Squares. Quarterly of Applied Mathematics **2** (1944) 164–168
10. Schnabel, R., Wahl, R., Klein, R.: Efficient RANSAC for point-cloud shape detection. Comput. Graph. Forum **26** (2007) 214–226
11. Fischler, M.A., Bolles, R.C.: Random sample consensus: A paradigm for model fitting with applications to image analysis and automated cartography. Commun. ACM **24** (1981) 381–395
12. Durhuus, B., Eilers, S.: On the entropy of lego. Journal of Applied Mathematics and Computing **45** (2014) 433–448
13. Nilsson, R.M.: On the number of flat LEGO structures. Master's thesis, University of Copenhagen (2016)
14. Abrahamsen, M., Eilers, S.: On the asymptotic enumeration of LEGO structures. Experimental Mathematics **20** (2011) 145–152
15. Mattheij, J.: Neural nets vs. lego bricks [resources]. IEEE Spectrum **54** (2017) 17–18
16. Nguyen, H.M.: kinect-duplo-sensing: A ROS package for identifying Duplo blocks with the Microsoft Kinect sensor. github.com/cheehieu/kinect-duplo-sensing (2014)
17. West, D.: A high-speed computer vision pipeline for the universal LEGO sorting machine. towardsdatascience.com/a-high-speed-computer-vision-pipeline-for-the-universal-lego-sorting-machine-253f5a690ef4 (2019)

18. Chou, C., Su, Y.: A block recognition system constructed by using a novel projection algorithm and convolution neural networks. IEEE Access **5** (2017) 23891–23900
19. Kim, J.W.: Survey on automated lego assembly construction. In: WSCG 2014: Poster Papers Proceedings: 22nd International Conference in Central Europeon Computer Graphics, Visualization and Computer Vision in co-operation with EUROGRAPHICS Association. (2014)
20. Peysakhov, M., Regli, W.C.: Using assembly representations to enable evolutionary design of lego structures. AI EDAM **17** (2003) 155–168
21. Schönberger, J.L., Frahm, J.M.: Structure-from-motion revisited. In: Conference on Computer Vision and Pattern Recognition (CVPR). (2016)
22. Schönberger, J.L., Zheng, E., Pollefeys, M., Frahm, J.M.: Pixelwise view selection for unstructured multi-view stereo. In: European Conference on Computer Vision (ECCV). (2016)
23. Lorensen, W.E., Cline, H.E.: Marching cubes: A high resolution 3d surface construction algorithm. In Stone, M.C., ed.: Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques, SIGGRAPH 1987, Anaheim, California, USA, July 27-31, 1987, ACM (1987) 163–169
24. Kutulakos, K.N., Seitz, S.M.: A theory of shape by space carving. Int. J. Comput. Vis. **38** (2000) 199–218
25. Xu, D., Tian, Y.: A Comprehensive Survey of Clustering Algorithms. Annals of Data Science **2** (2015) 165–193
26. Lloyd, S.P.: Least squares quantization in PCM. IEEE Trans. Inf. Theory **28** (1982) 129–136
27. Schubert, E., Sander, J., Ester, M., Kriegel, H., Xu, X.: DBSCAN revisited, revisited: Why and how you should (still) use DBSCAN. ACM Trans. Database Syst. **42** (2017) 19:1–19:21
28. von Gioi, R.G., Jakubowicz, J., Morel, J., Randall, G.: LSD: a line segment detector. IPOL Journal **2** (2012) 35–55
29. Besl, P.J., McKay, N.D.: A method for registration of 3-d shapes. IEEE Trans. Pattern Anal. Mach. Intell. **14** (1992) 239–256