

# COALESCE: Component Assembly by Learning to Synthesize Connections

Kangxue Yin  
Simon Fraser University

Zhiqin Chen  
Simon Fraser University

Siddhartha Chaudhuri  
Adobe Research, IIT Bombay

Matthew Fisher  
Adobe Research

Vladimir Kim  
Adobe Research

Hao Zhang  
Simon Fraser University

## Abstract

We introduce COALESCE, the first data-driven framework for component-based shape assembly which employs deep learning to synthesize part connections. To handle geometric and topological mismatches between parts, we remove the mismatched portions via erosion, and rely on a joint synthesis step, which is learned from data, to fill the gap and arrive at a natural and plausible part joint. Given a set of input parts extracted from different objects, COALESCE automatically aligns them and synthesizes plausible joints to connect the parts into a coherent 3D object represented by a mesh. The joint synthesis network, designed to focus on joint regions, reconstructs the surface between the parts by predicting an implicit shape representation that agrees with existing parts, while generating a smooth and topologically meaningful connection. We demonstrate that our method significantly outperforms prior approaches including baseline deep models for 3D shape synthesis, as well as state-of-the-art methods for shape completion.

## 1. Introduction

Shape assembly by composing component parts has been the dominant paradigm for data-driven 3D modeling and design exploration [29, 10, 3, 47, 34]. The component assembly gathers parts from a shape collection and composes them into a target shape. To date, most research effort has been devoted to finding compatible parts for the assembly [2, 17, 20, 52, 19, 23, 38]. However, these parts still need to be properly aligned and then connected or joined in a natural and coherent fashion, ideally resulting in a high-quality final shape that is suitable for fabrication, simulation, high-resolution rendering, and seamless texturing.

Compared to the higher-level and human-centric tasks, such as part selection [3, 38] and assembly exploration [47], geometrically aligning and joining 3D parts is a much more tedious and unnatural practice for human modelers. A full automation of this low-level task is highly desirable. How-

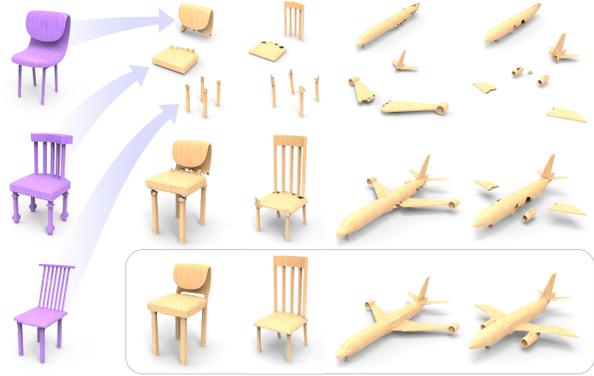


Figure 1: Our deep learning framework, COALESCE, assembles 3D shapes by a learning-based approach to synthesize part connections. Given a set of unaligned, roughly cut parts with possibly mismatched joint regions (top row: all parts come from *different* shapes), the alignment module of our network can learn to arrange these parts into a plausible configuration (middle row). However, this still leaves gaps and geometric/topological mismatches at the joints. After eroding the input parts in advance, our joint synthesis module creates connecting regions, which are stitched to the input to create seamless assemblies (bottom row).

ever, due to the large variability exhibited by the geometry and topology of the shape parts, heuristic approaches such as hole filling and stitching are hardly general and reliable, as they must make varying assumptions about part placement, feature alignment and boundary geometry, or resort to user interactions [21, 7, 15, 49, 26, 37, 35]. Further, nearly all these analytical methods (except that of Huang et al. [15]) are primarily designed for organic shapes with smoothness priors, and do not handle man-made shapes with sharp corners, edges, and rich structural variations.

In this paper, we introduce COALESCE, the first *data-driven* framework for component assembly by learning to *align* shape parts *and synthesize* the connections between them. We stipulate that in the absence of any mathematical formulation of how to connect parts, the most general and

effective way to handle the immensely rich part connection varieties is to *learn* how parts are connected from a large variety of 3D shapes. In addition, the most viable solution to handle geometric or topological mismatches between two parts is to remove the mismatched portions, via an *erosion* process, and then rely on a *learned joint synthesis* step to produce a natural and plausible part connection.

A defining feature of our learning-based part connection framework is that it focuses on *joint* synthesis, rather than learning to reconstruct whole shapes. With proper part alignment, which is also learned, the space of part joints possesses a higher level of predictability compared to that of the whole shapes, making joint synthesis a comparably easier learning problem. Another novel feature of our method, compared to previous works on hole filling and shape completion, is the “retreat (erode) before advance” connection mechanism. It can be regarded as a means of normalization to handle the wide variety of possible part connections.

The main contributions of our work include:

- We tackle an important problem for assembly-based shape modeling: how to best connect parts, which may come from different 3D models and possess varying degrees of geometric and structural incompatibilities.
- An erosion-based part connection mechanism to help reconcile geometric and topological mismatches between parts, and potential part misalignment.
- A novel joint synthesis method to connect input parts into a plausible 3D object, via an implicit surface. As shown in the leftmost chair assembly in Figure 1, our network is able to connect parts into a novel structure which is absent from the parts’ shape origins.

Our networks are trained on ShapeNet part dataset [48]. We demonstrate the performance of our “learning to connect” framework on a variety of 3D shapes, both qualitatively and quantitatively, through comprehensive comparative studies. Our results exhibit clear superiority in both overall visual quality of the assembled meshes and feature preservation with respect to the original mesh parts.

## 2. Related work

The literature on modeling shapes from parts is extensive. Hence we mainly discuss geometric stitching with analytical approaches, assembly-based modeling from given parts, and modern neural shape synthesis methods.

**Part stitching.** Several papers have studied the problem of stitching parts together by connecting a pair of boundary edge loops, where the principal objective is smoothness of the connection. Yu et al. [49] proposed gradient-domain Poisson stitching: after establishing correspondences between the loop vertices, a target continuous gradient field

is set up in the adjacent mesh regions. Sharf et al. [37] proposed a “Soft-ICP” algorithm, which iteratively and non-rigidly brings two overlapping patches (one adjacent to each loop) together until they smoothly blend with each other. Lin et al. [26] create a sketch-guided implicit field to connect two boundary loops. The use of an implicit field to generate a connecting surface is similar to our approach, but the synthesis is entirely guided by user input. Schmidt and Singh [35] join a boundary edge loop to a target surface by creating consistent local parametrizations on the surface as well as in the polygon bounded by the loop. The field-guided registration of Huang et al. [15] has a goal more similar to ours. However, the method is predicated on the existence of an underlying smooth transition between the parts that can be detected by geometric analysis.

In contrast to all of these methods, our approach *learns* how to repair geometrically and topologically dissimilar junctions, generates both low and high curvature regions as indicated by training data, and co-aligns the assembled parts while accounting for substantial cutting error.

**Assembly-based modeling.** A variety of methods studied ways to construct shapes from pre-existing parts. The pioneering work of Funkhouser et al. [10] presented tools for searching for database shapes partially matching a user-provided proxy, cutting out a matching part using “intelligent scissors”, and pasting the part into a new model. The majority of subsequent assembly-based methods focused on retrieving and organizing compatible parts to facilitate or automate assembly [22, 2, 17, 19, 39, 47, 9, 18, 38, 34, 36, 44, 6]. These methods did not place particular emphasis on repairing or synthesizing the connections between parts, especially for man-made shapes. Often, parts were simply placed in contact, or slightly overlapped, to appear visually connected, although Kraevoy et al. [22], Chaudhuri et al. [2] and Kalogerakis et al. [19] did create smooth transitions between organic parts using methods described above.

The SCORES system of Zhu et al. [53] shares some aspects of our method: it attempted to realign a crude input arrangement of parts in order to resolve incompatibilities and misalignments using a recursive neural network. However, this work focused on *structural* repair. Only affine transforms were applied to input parts, and no stitching was performed at joints. Duncan et al. [7] presented a method to deform corresponding joint regions in a shape collection to resemble each other, so that parts could be seamlessly interchanged. However, this required substantial modifications to the input parts, and forced all joints between parts from the same two semantic categories to look identical.

In general, complicated joints, especially in man-made shapes, remained an open problem.

**Generative models of 3D geometry.** In contrast to the assembly-based methods, which *reuse* existing shape parts,

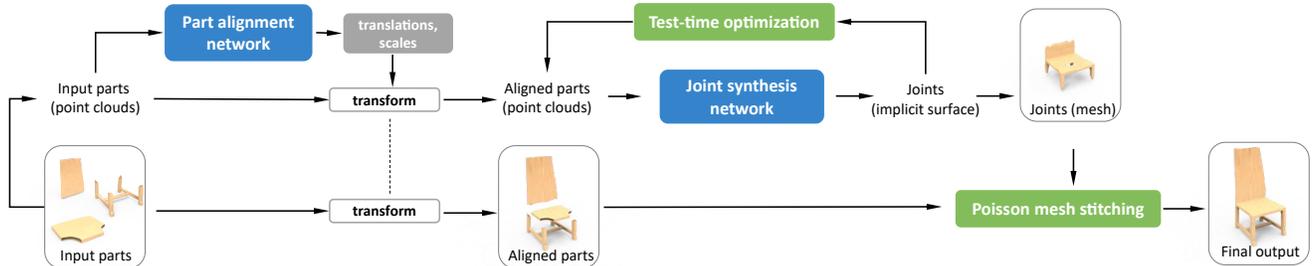


Figure 2: An overview of COALESCE, our component-wise shape assembly. Input mesh parts, with the joint areas eroded to create room for joint synthesis, are first transformed by a part alignment network. Then, a joint synthesis network is employed to synthesize a joint region as an implicit field between each pair of aligned parts. These fields are then polygonized and finally Poisson stitching is applied to the input parts and the synthesized joints to produce the final mesh output. Note: dashed lines indicate that the set of part transforms is shared between the point cloud (top) and mesh (bottom) branches.

substantial recent research has focused on *synthesizing* the part geometry itself. Neural generative models employ various representations for the output shape, including point clouds [1], voxel grids [45, 51], octrees [41], deformations of a template mesh [12], 3D embeddings of 2D atlases [13], or combinations of the above [31]. Recent techniques model shapes as continuous implicit functions, where the network learns to map a 3D point to an occupancy value [28] or a clamped signed distance field [32]. We employ a variant of these methods [4] in our joint synthesis, since they demonstrate high visual quality and can capture complex topological variations of shapes. Since these implicit functions are defined everywhere in the 3D volume, it makes it easier for us to incorporate them into consistency losses that ensure the synthesized joint aligns with the input geometry, without expensive closest point sampling. Some methods explicitly model part layout in addition to generating part geometry [16, 25, 40, 24, 11, 46, 30], but none of them explicitly focus on joints between parts.

Neural generative models have been commonly used in encoder-decoder architectures; they encode input that is partial or of another modality, such as images or depth scans, and output full 3D shapes. The output is evaluated via reconstructive [8] or discriminative [43] losses to train the model. Hence, a naive solution to joint synthesis could employ a shape completion approach [5], where the system would encode unstitched parts, and decode the entire shape. Unfortunately, this approach does not preserve the input parts or provide the highest quality joints since much of the network capacity is allocated to representing the entire shape, as we show in our experiments. In contrast, our method focuses neural synthesis specifically on the joint sections, but in a part- and global shape-aware way.

### 3. Method

Given a set of input parts extracted from *different* objects, our network automatically aligns them and synthesizes plausible joints, connecting the parts into a coherent

mesh object. Our pipeline is visualized in Figure 2.

The input to our pipeline is a set of labeled parts, where each part is represented by a triangle mesh. We uniformly sample points on the input parts and normalize the resulting point clouds. We then erode the joint regions on each part to ensure that there is enough room for joint synthesis and to avoid the structural mismatch between crudely segmented parts. Specifically, for each point on an input part, we compute the distance to the segmentation boundary and if it is less than a threshold  $\tau$ , the point is excluded from the part.

The alignment network takes normalized parts as inputs and predicts translation and scaling to create structurally sound composition of parts (Section 3.1). Next, the joint synthesis network takes the aligned parts as inputs and synthesizes connections between them as an implicit surface (Section 3.2). Note that implicit representation is especially suitable for joint synthesis, since the network is able to focus on the joint area and allocate its capacity to the geometry near the joints. To make the output implicit surface closely conform to the input part boundaries, we jointly refine the predicted spatial transformations and the weights of the joint synthesis network to minimize a part-joint matching loss. After that, the implicit surface produced by the joint synthesis network is discretized into a voxel grid and converted into a mesh using marching cubes [27].

Finally, we develop a Poisson-based approach, a variant of Poisson blending, to seamlessly stitch the synthesized joint meshes with the original meshes of the aligned input parts. The details of the approach is described in the supplementary material.

#### 3.1. Part alignment network

Given a set of input parts from different objects, we sample points for each of them and normalize the resulting point clouds. After that, we use a part alignment network to assemble them into a shape by applying predicted translation and scaling to each part. As shown in Figure 3(a), the part alignment network contains PointNet++[33] networks that

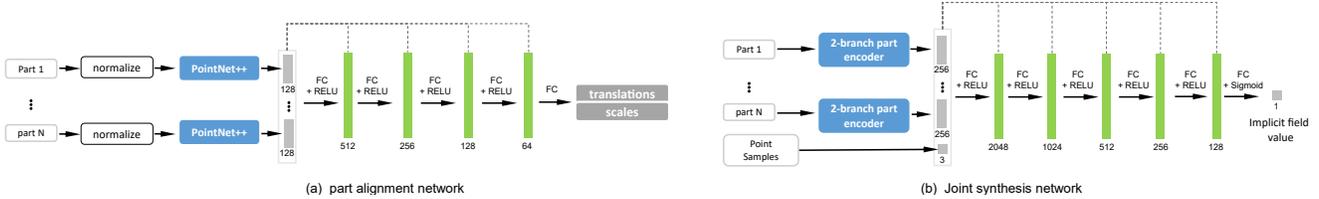


Figure 3: The architectures of our part alignment network (a), and our joint synthesis network (b), where the input parts are represented by point clouds. Dashed lines denote skip connections.

compute shape features for input parts. Given the features of parts, we use a fully connected regression network with skip connections to produce the spatial transformation that aligns the parts. The features for missing object parts are simply set as zero vectors. We train the part alignment network with sets of randomly disturbed parts belonging to the same object. Our loss function is the Earth Mover’s Distance between the translated and scaled parts and the ground truth shape. Details of training process and the architecture of the PointNet++ used in the part alignment network is provided in the supplementary material.

### 3.2. Joint synthesis network

Our joint synthesis network is a core component of the framework. It takes point clouds of the aligned parts as input and reconstructs the joints as an implicit surface that connects the parts into a plausible object. We show in Figure 3(b) the architecture of our joint synthesis network. The network is designed as an encoder-decoder network, where each input part is assigned a two-branch point cloud encoder. The first branch of the part encoder is a PointNet++ network for extracting overall shape features from the part. The second branch is another PointNet++ network for extracting localized shape features from the points that are close to the joint area. We represent each part with 2048 points, and consider the 512 points that are closest to the joint area as the input to the second branch for localized shape features. The details of the two PointNet++’s used in the two-branch part encoder is provided in the supplementary material.

The network uses an IM-NET [4] to reconstruct an implicit surface for the joint. Given an object category, for each semantic part segment, we use the two-branch part encoder to extract a feature vector from the part. If a part does not exist in an object, we use a zero vector instead. The feature vectors of  $N$  parts are then concatenated into a single vector used by the IM-NET decoder.

The implicit decoder requires point samples in the 3D space surrounding the shape we aim to reconstruct. When we sample points with inside/outside status with respect to the ground-truth shape for training the IM-decoder, to make the network focus on the joint area, we densely sample points around joints and sparsely sample points away

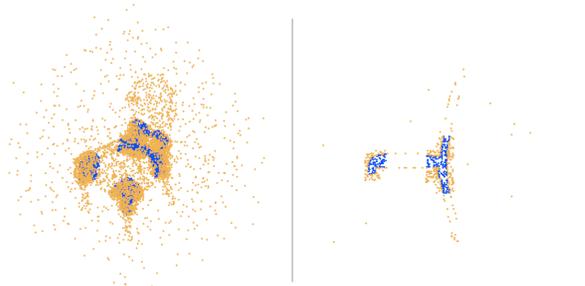


Figure 4: Visualization of our point sampling for IM-NET. Dark yellow dots denote points outside the shape, blue dots denote points inside the shape.

from joints as shown in Figure 4. Specifically, we use 16384 point samples for each shape: 80% samples are within the joint volume after 5-step volumetric inflation in a  $256^3$  voxel grid, 10% are points outside the shape but tightly close to the surface of the shape, and the remaining 10% are random samples that are outside the shape. By sampling training points for IM-decoder in this way, we only force the IM-decoder to reconstruct the joint area rather than the entire object.

The loss function of the joint synthesis network is defined by two terms:

$$L_{\text{joint}} = L_{\text{mse}} + \alpha L_{\text{match}}$$

where  $\alpha$  is a balancing weight set to 0.2 in all experiments. The first term  $L_{\text{mse}}$  is the original loss term in IM-NET which is defined as:

$$L_{\text{mse}} = \frac{1}{|S|} \sum_{p \in S} |f(p) - F(p)|^2$$

where  $S$  is the set of point samples,  $f(p)$  is the output value from our network for point sample  $p \in S$ .  $F(p)$  is the ground-truth label of point  $p$ .

The term  $L_{\text{match}}$  is for encouraging the synthesized joint to match the open boundaries of input parts. To define  $L_{\text{match}}$ , we first select a set of 1024 points  $\mathcal{N}(J)$  from all input parts that are close to the joint area. If the implicit surface tightly matches the open boundaries of the input parts, for each point  $p \in \mathcal{N}(J)$ , if we move it along its normal direction, the expected implicit function value on its new

position is 0(outside). If we move it in the inverse direction of its normal direction, the expected implicit function value is 1(inside). Thus we define  $L_{\text{match}}$  as:

$$L_{\text{match}} = \frac{1}{2|N(J)|} \sum_{p \in N(J)} |f(p + \lambda n)|^2 + |f(p - \lambda n) - 1.0|^2$$

where  $n$  is the unit normal vector of  $p$ , and  $\lambda$  is a small constant with a default value of 0.005.

The joint synthesis network is trained by segmented 3D objects and their parts. To make room for the network to learn to synthesize the joint connecting input parts, we erode the parts along the segmentation boundaries. As discussed in section 1, another effect of the erosion is helping to eliminate topological mismatches among input parts from different objects when given new parts. As the point cloud encoders and the IM-decoder require different learning rates for training, we first pre-train the two-branch point cloud encoders for 100 epochs by pairing it with a fully connected decoder [1] for reconstructing 3D shape parts. After that, we pair the pretrained point cloud encoders with the IM-decoder, and train the IM-decoder with the loss function  $L_{\text{joint}} = L_{\text{mse}}$  for 80 epochs. Finally, we train the encoders and the IM-decoder together with the loss function  $L_{\text{joint}} = L_{\text{mse}} + \alpha L_{\text{match}}$  for another 80 epochs. More details on our training procedures can be found in the supplementary material.

### 3.3. Test-time optimization

It is critical that the synthesized joints align closely with the open boundaries of the parts to create a realistic model. While our feed-forward joint synthesis network attempts to maximize boundary consistency, to further improve results we employ test-time optimization to explicitly minimize any boundary discontinuities. In this phase, we jointly optimize the scaling factors  $\{s\}$ , translations  $\{t\}$  of parts, and the parameters  $W$  of the IM-decoder in the synthesis network for a given test input. We define the objective function for this optimization as an  $L_1$  version of the part-joint matching term:

$$h(\{s\}, \{t\}, W) = \frac{1}{2|N(J)|} \sum_{p \in N(J)} |f(p + \lambda n)| + |f(p - \lambda n) - 1.0|$$

where  $N(J)$  is a set of 1,024 points close to the joint,  $n$  is the unit normal at  $p$ , and  $\lambda$  is a small constant set to 0.005.

Minimizing the objective function  $h$  with respect to  $\{s\}$ ,  $\{t\}$  requires a significantly different step size compared to minimizing the objective function with respect to network parameters  $W$ . This motivated our choice to alternately optimize for part alignments and joint synthesis. We first fix the parameters of the IM-decoder and optimize  $\{s\}$ ,  $\{t\}$  by minimizing the objective function  $h$  with an Adam optimizer with step size 0.002. After that, we fix  $\{s\}$ ,  $\{t\}$  and optimize the parameters of the IM-decoder  $W$  with step size 0.0001. We keep repeating the above two steps for a

fixed number of iterations (25, in our implementation). As we show in Section 4.2, the test-time optimization refines the positions and scales of the parts and the decoding result such that they are matched more tightly.

### 3.4. Poisson mesh stitching

After test-time optimization, the joint region represented by implicit function is converted into a polygon mesh by Marching Cubes. A Poisson-based approach is developed to seamlessly stitch the synthesized mesh with the aligned parts. The Poisson mesh stitching has three phases: (i) finding corresponding loops between the input parts and the synthesized joints, (ii) removing redundant regions from the joint mesh and (iii) Poisson mesh blending. Please find the description of these steps in the supplementary material.

## 4. Experiment and results

We evaluate the utility of COALESCE for the part assembly problem. In the following, we discuss our data pipeline, ablation studies for various design choices, and comparisons to alternative approaches. More visual results can be found in the supplementary material. Unless stated otherwise (e.g., in ablations), we use the same network architecture with the same hyper-parameters and the same Poisson mesh stitching post-process for all the experiments.

### 4.1. Dataset

We use three categories of objects from the segmented shapes provided by Yi et al. [48]: 3,746 chairs, 2,690 airplanes, and 184 mugs. We split each category into training (80%) and test (20%) sets. We then create eroded part-wise point clouds as described in Section 3. For our erosion process we set the erosion radius to  $\tau = 0.05$  of the bounding box diameter for chairs and mugs and  $\tau = 0.025$  for airplanes. Since there are very few mugs, we train the joint synthesis network for the mug dataset for 80 more epochs.

### 4.2. Ablation study

**Qualitative study.** As shown in Figure 5, the inputs to our part alignment network are crudely segmented parts from *different* objects. Figure 5b shows that the results of our network may be inaccurate. Similarly, the geometry created by the forward pass through the joint synthesis network might not align tightly with the input parts, e.g., the last row. Our test-time optimization addresses these issues and further refines the alignment and the joint geometry (Figure 5c) to produce the final result (Figure 5d).

A naive approach to assemble parts from different objects is to learn to align them directly, without erosion or joint synthesis. We train a part alignment network that directly aligns input parts to get the assembly, and show the results in Figure 5e. We see several artifacts emerging from

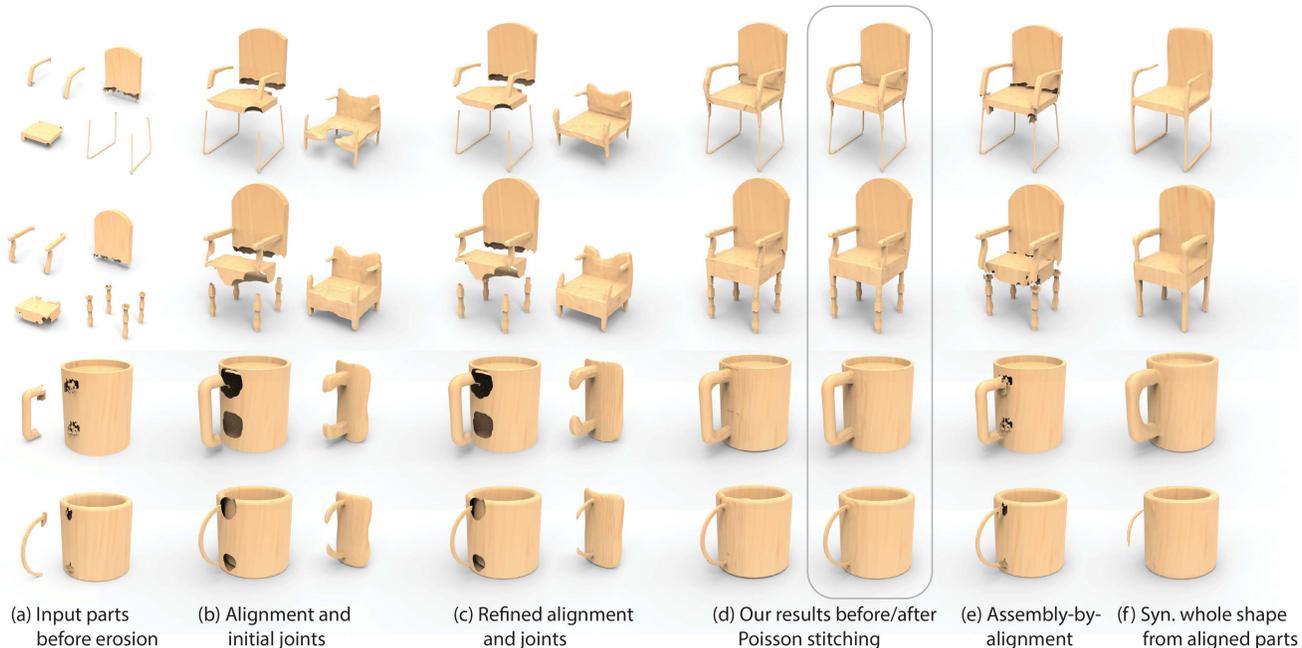


Figure 5: We evaluate the significance of different components of our framework by ablation. Given the input parts (a), we show results that only involve part alignments (b), refined alignment with test-time optimization (c), and the effect of Poisson stitching (d). We also demonstrate the disadvantages of naive approaches, such as assembling input parts without any joint synthesis (e), or directly synthesizing the entire shape (f) from aligned parts.

this naive pipeline. First, there could be a topological or geometric mismatch, such as armrests of a chair falling outside the range of the seat (see second row) or the cup of a mug having holes that are larger than the size of the handle (see last row). Second, segmented parts might have imperfect boundaries (see first row). Note how our erosion and synthesis approach solves these issues (Figure 5d).

Another alternative is to treat this as a shape completion problem, which is commonly addressed with encoder-decoder architectures. We train a neural network that encodes point clouds of aligned parts, and then decodes them into a full shape via IM-decoder (with the identical architecture to our joint synthesis network). As we can see in Figure 5f, the results of this approach might not preserve the geometry of the input parts and often have more artifacts near joints due to the absence of joint focus.

**Quantitative study.** In addition to the qualitative ablation study in Figure 5, we have done a quantitative ablation study as shown in Table 1. In the study, we feed the eroded input parts from the same test shape to our network, and measure the chamfer distance and light field distance (LFD) between the results of our method and ground-truth test shapes. The quantitative evaluation results show that the test-time optimization module and the Poisson blending module can improve the quality of the network output.

|                        | Chamfer Dis. ( $\times 10^3$ ) |              |              | Light Field Dis. |              |              |
|------------------------|--------------------------------|--------------|--------------|------------------|--------------|--------------|
|                        | plane                          | chair        | mug          | plane            | chair        | mug          |
| Synthesize whole shape | 0.164                          | 0.738        | 0.804        | 3265.2           | 3054.3       | 1117.8       |
| Before test-time opt.  | 0.073                          | 0.225        | 0.297        | 1148.3           | 1251.6       | 374.3        |
| After test-time opt.   | 0.071                          | 0.191        | 0.300        | 1072.9           | 951.5        | 276.9        |
| After Poisson blending | <b>0.067</b>                   | <b>0.181</b> | <b>0.274</b> | <b>895.4</b>     | <b>894.6</b> | <b>196.2</b> |

Table 1: Quantitative ablation study. We compute the errors of our method against ground-truth shapes on different pipeline stages including before and after test-time optimization, and after Poisson blending. For a comparison, we also evaluate the result of a network that directly synthesizes the entire shape from the aligned parts.

**Part erosion.** To show the necessity of part erosion, we train a neural network to synthesize joints from *uneroded* parts, with the same architecture and configuration as our network described in Section 3. We compare the result of directly synthesizing joints from uneroded parts with our result in Figure 6. Note that the variant without erosion cannot successfully resolve the mismatches at part junctions.

### 4.3. Comparison to Baselines

Although there are no prior works on learning joint synthesis, we can compare to shape completion methods, such as AtlasNet [13] and PCN [50]. We directly use the publicly available source codes of these methods and train the networks on individual categories following the provided

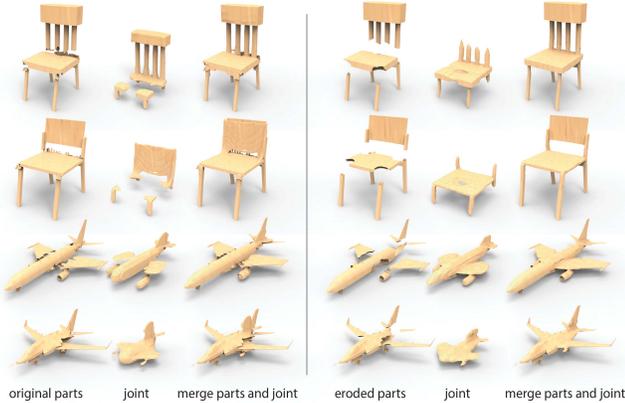


Figure 6: Comparison of two networks that synthesize part connections from original parts before erosion (left) and from eroded parts (right). Incorporating erosion better resolves geometric and topological mismatches.

instructions. We also tested a customized version of AtlasNet that only outputs surfaces in the joint regions, similar to our joint synthesis network. We denote this version of AtlasNet as AtlasNet\* in Figure 7h and Tables 2, 3, 4.

We also consider two voxel-based shape completion baselines that can be applied to our problem: 3DCNN and 3DResNet. In the two baselines, we use architectures that perform shape completion akin to image translation tasks. See the supplementary material for more details. Unlike encoder-decoder architectures such as AtlasNet and PCN, these methods do not have a bottleneck layer, and they only modify input in a relatively local manner. Thus, they do not have the challenge of reconstructing the entire shape from the latent code, and might be better at preserving the geometry of the original parts. When testing, we use marching cubes [27] to obtain the meshes from the output voxels.

The qualitative comparison of our method to the baselines is provided in Figure 7. The input parts in Figure 7 are from different objects (Figure 7a), and naively aligning them (Figure 7b) results in a mesh with a large number of boundary artifacts. We feed eroded parts (Figure 7c) as input to our joint synthesis method and various baselines. Our method produces high quality seamless meshes (Figure 7d). Voxel translation techniques (3DCNN, 3DResNet) suffer from discretization artifacts and often produce poor joints (Figure 7e,f). Encoder-decoder architectures (AtlasNet, PCN) tend to significantly change the input geometry (Figure 7g,i). While modified AtlasNet\* preserves the original geometry by construction, the synthesized joints do not seamlessly merge with the rest of the geometry (Figure 7h).

To quantitatively evaluate our method, we take eroded input parts from the same test shape, and use that test shape as the ground truth. We report Chamfer distance (*lower* is better) and light field distance (LFD: *lower* is better) as error metrics in Table 2. We can see that our results have the

|           | Chamfer Distance ( $\times 10^3$ ) |              |              | Light Field Distance |              |              |
|-----------|------------------------------------|--------------|--------------|----------------------|--------------|--------------|
|           | plane                              | chair        | mug          | plane                | chair        | mug          |
| AtlasNet  | 0.117                              | 0.402        | 0.885        | 2402.1               | 2482.4       | 2227.9       |
| AtlasNet* | <b>0.067</b>                       | 0.205        | 0.277        | 1049.4               | 1316.8       | 455.2        |
| PCN       | 0.095                              | 0.308        | 0.506        | -                    | -            | -            |
| 3DCNN     | 0.151                              | 0.268        | 0.440        | 2464.6               | 1446.2       | 615.8        |
| 3DResNet  | 0.141                              | 0.262        | 0.421        | 2086.2               | 1313.2       | 426.9        |
| Ours      | <b>0.067</b>                       | <b>0.181</b> | <b>0.274</b> | <b>895.4</b>         | <b>894.6</b> | <b>196.2</b> |

Table 2: Quantitative results of different methods when the input parts are from the same object and aligned with respect to the ground truth positions. Chamfer Distance is multiplied by  $10^3$ . Light Field Distance cannot be applied to PCN since PCN generates point clouds rather than meshes.

|           | Chamfer Distance ( $\times 10^3$ ) |              |              | Light Field Distance |               |              |
|-----------|------------------------------------|--------------|--------------|----------------------|---------------|--------------|
|           | plane                              | chair        | mug          | plane                | chair         | mug          |
| AtlasNet  | 0.200                              | 0.525        | 0.948        | 3462.5               | 2691.9        | 2240.4       |
| AtlasNet* | 0.092                              | 0.258        | 0.288        | 1716.5               | 1517.9        | 485.5        |
| PCN       | 0.173                              | 0.489        | 0.583        | -                    | -             | -            |
| 3DCNN     | 0.199                              | 0.330        | 0.495        | 2896.6               | 1563.2        | 655.4        |
| 3DResNet  | 0.179                              | 0.319        | 0.485        | 2265.7               | 1414.0        | 449.2        |
| Ours      | <b>0.080</b>                       | <b>0.222</b> | <b>0.287</b> | <b>1302.2</b>        | <b>1039.9</b> | <b>198.7</b> |

Table 3: Quantitative results of different methods when the input parts are warped by a sine function.

|           | Chamfer Distance ( $\times 10^3$ ) |              |              | Light Field Distance |               |              |
|-----------|------------------------------------|--------------|--------------|----------------------|---------------|--------------|
|           | plane                              | chair        | mug          | plane                | chair         | mug          |
| AtlasNet  | 0.313                              | 0.666        | 1.394        | 4158.8               | 3035.8        | 2843.1       |
| AtlasNet* | 0.109                              | 0.255        | 0.302        | 2264.6               | 1835.1        | 737.2        |
| PCN       | 0.274                              | 0.512        | 0.989        | -                    | -             | -            |
| 3DCNN     | 0.219                              | 0.314        | 0.486        | 3682.1               | 1570.3        | 719.8        |
| 3DResNet  | 0.177                              | 0.304        | 0.476        | 2362.7               | 1403.7        | 505.9        |
| Ours      | <b>0.077</b>                       | <b>0.190</b> | <b>0.292</b> | <b>1247.9</b>        | <b>1063.0</b> | <b>231.9</b> |

Table 4: Quantitative results of different methods when the input parts are randomly scaled and translated.

lowest errors in both measures. Although the customized AtlasNet\* has errors close to ours in terms of Chamfer distances, it does not have the same visual quality as our results, which is more evident from LFD errors.

To evaluate the robustness of our method and the baselines to part misalignment, we warp the test input parts and the ground-truth shape along the y-axis with a sine transform:  $y \leftarrow y + 0.02 \sin(4\pi z + f)$ , where  $y$  and  $z$  are the coordinates of a 3D point  $(x, y, z)$ ,  $f$  is a per-shape random number sampled uniformly from  $[-\pi, \pi]$ , and 0.02 is the constant amplitude. We report the quantitative performance of our method and the baselines in Table 3.

To evaluate the robustness of our method and the baseline to translation and scaling of the test shape, we randomly scale and translate each of the test input parts, with the uniformly-sampled scale in the range  $[0.9, 1.1]$ , and uniformly-sampled displacement in the range  $[-0.04, 0.04]^3$ . Table 4 shows results under random scaling and translation. These results demonstrate that our method

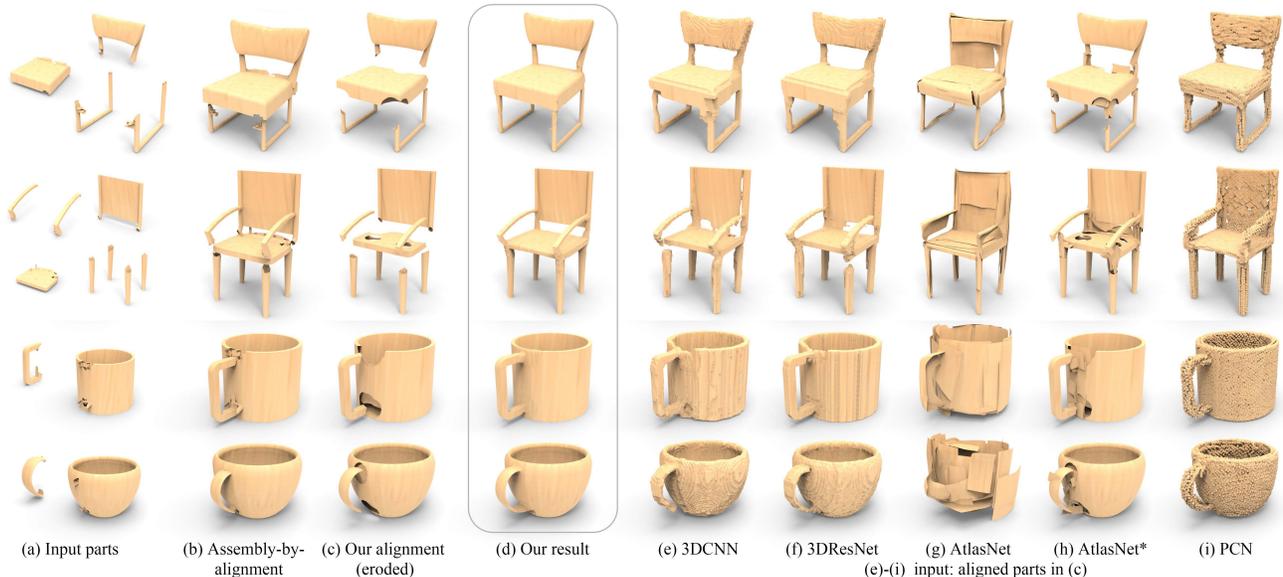


Figure 7: Qualitative comparisons with baseline methods, which took the parts aligned by our method (c) as input.

is the most robust to various kinds of part misalignment.

#### 4.4. Generalizability

We also tested our network on the chair category of the COSEG [42] dataset. We do not retrain our network to fit the dataset. Instead, we use the network trained on the chair category from ShapeNet, and test it on COSEG to demonstrate how well it generalizes to novel datasets. Figure 8 shows some results to demonstrate such generalizability.

#### 5. Discussion and Future Work

We presented COALESCE, the first neural network-based pipeline for aligning and seamlessly connecting parts in a 3D shape assembly. In interactive modeling scenarios, it is hard to perfectly align parts, and even harder (and often impossible) to make topologically and geometrically compatible cuts in adjacent parts for the purpose of seamless joining. COALESCE erodes crudely cut joint regions to prepare them for connections, corrects alignment errors in the assembly, and generates seamless connecting patches filling in the eroded joints, accounting for geometric and topological mismatches. Experiments show that our approach significantly outperforms alternatives, and is a promising addition to the assembly-based modeling toolbox, applying modern neural synthesis techniques to address a need largely neglected by prior work.

Our method is only a first step and still has several limitations, which suggest ripe avenues for future work. First, like most learning-based methods for shape analysis, COALESCE is still category-specific, and relies on predictable, labeled part layouts and joint locations. Also, our method has been trained on parts aligned to a canonical orientation

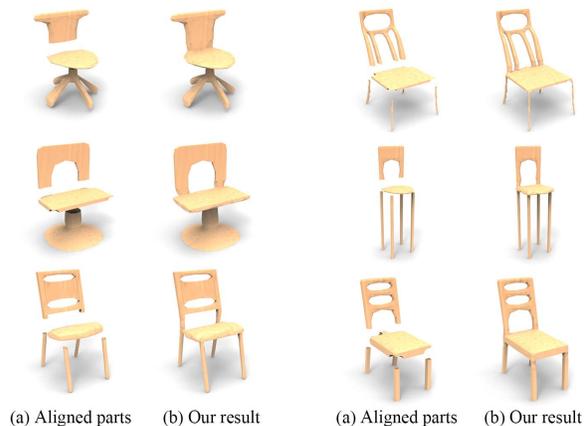


Figure 8: Qualitative results of our method trained on chairs from ShapeNet but tested on chairs from COSEG.

and compatibly scaled. Furthermore, we used heuristic erosion of the input parts, but we believe it is possible to extend this approach to learn a natural erosion operation.

**Acknowledgements** This research was supported in part by grants from NSERC and gifts from Adobe Research. We thank members of SFU GrUVi lab and Adobe Research for helpful discussions. Part of this work was done while Kangxue was an intern at Adobe Research.

## References

- [1] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas. Learning representations and generative models for 3d point clouds. In *ICML*, 2018. 3, 5, 12
- [2] S. Chaudhuri, E. Kalogerakis, L. Guibas, and V. Koltun. Probabilistic reasoning for assembly-based 3d modeling. *ACM Trans. Graph.*, 30(4):35:1–35:10, 2011. 1, 2
- [3] S. Chaudhuri and V. Koltun. Data-driven suggestions for creativity support in 3D modeling. *ACM Trans. Graph.*, 29(6):183:1–9, 2010. 1
- [4] Z. Chen and H. Zhang. Learning implicit fields for generative shape modeling. In *CVPR*, pages 5939–5948, 2019. 3, 4
- [5] A. Dai, C. R. Qi, and M. Nießner. Shape completion using 3d-encoder-predictor cnns and shape synthesis. In *CVPR*, 2017. 3
- [6] A. Dubrovina, F. Xia, P. Achlioptas, M. Shalah, R. Groskot, and L. Guibas. Composite shape modeling via latent space factorization. In *ICCV*, 2019. 2
- [7] N. Duncan, L.-F. Yu, and S.-K. Yeung. Interchangeable components for hands-on assembly based modelling. *ACM Trans. Graph.*, 35(6):234:1–234:14, 2016. 1, 2
- [8] H. Fan, H. Su, and L. J. Guibas. A point set generation network for 3d object reconstruction from a single image. In *CVPR*, 2017. 3
- [9] L. Fan and P. Wonka. A probabilistic model for exteriors of residential buildings. *ACM Trans. Graph.*, 35(5):155:1–155:13, 2016. 2
- [10] T. Funkhouser, M. Kazhdan, P. Shilane, P. Min, W. Kiefer, A. Tal, S. Rusinkiewicz, and D. Dobkin. Modeling by example. *ACM Trans. Graph.*, 23(3):652–663, 2004. 1, 2
- [11] L. Gao, J. Yang, T. Wu, H. Fu, Y.-K. Lai, and H. Zhang. SDM-NET: Deep generative network for structured deformable mesh. *ACM Trans. Graph.*, 38(6), 2019. 3
- [12] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry. 3D-CODED: 3D correspondences by deep deformation. *ECCV*, 2018. 3
- [13] T. Groueix, M. Fisher, V. G. Kim, B. C. Russell, and M. Aubry. Atlasnet: A papier-mâché approach to learning 3d surface generation. *CVPR*, 2018. 3, 6, 12
- [14] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 12
- [15] H. Huang, M. Gong, D. Cohen-Or, Y. Ouyang, F. Tao, and H. Zhang. Field-guided registration for feature-conforming shape composition. *ACM Trans. Graph. (SIGGRAPH Asia)*, 31(6):Article 179, 2012. 1, 2
- [16] H. Huang, E. Kalogerakis, and B. Marlin. Analysis and synthesis of 3D shape families via deep-learned generative models of surfaces. In *SGP*, 2015. 3
- [17] A. Jain, T. Thormählen, T. Ritschel, and H.-P. Seidel. Exploring shape variations by 3D-model decomposition and part-based recombination. In *Eurographics*, 2012. 1, 2
- [18] P. Jaiswal, J. Huang, and R. Rai. Assembly-based conceptual 3D modeling with unlabeled components using probabilistic factor graph. *Comput. Aided Des.*, 74(C):45–54, 2016. 2
- [19] E. Kalogerakis, S. Chaudhuri, D. Koller, and V. Koltun. A probabilistic model for component-based shape synthesis. *ACM Trans. Graph. (SIGGRAPH)*, 31(4), 2012. 1, 2
- [20] V. G. Kim, W. Li, N. J. Mitra, S. Chaudhuri, S. DiVerdi, and T. Funkhouser. Learning part-based templates from large collections of 3D shapes. In *SIGGRAPH*, 2013. 1
- [21] V. Kraevoy, D. Julius, and A. Sheffer. Shuffler: Modeling with interchangeable parts. In *Pacific Graphics*, 2007. 1
- [22] V. Kraevoy, D. Julius, and A. Sheffer. Model composition from interchangeable components. In *Pacific Graphics*, pages 129–138, 2007. 2
- [23] H. Laga, M. Mortara, and M. Spagnuolo. Geometry and context for semantic correspondences and functionality recognition in man-made 3d shapes. *ACM Trans. Graph.*, 32(5):150, 2013. 1
- [24] J. Li, C. Niu, and K. Xu. Learning part generation and assembly for structure-aware shape synthesis. *arXiv preprint arXiv:1906.06693*, 2019. 3
- [25] J. Li, K. Xu, S. Chaudhuri, E. Yumer, H. Zhang, and L. Guibas. GRASS: Generative Recursive Autoencoders for Shape Structures. *ACM Trans. Graph.*, 36(4):52:1–52:14, 2017. 3
- [26] J. Lin, X. Jin, C. C. Wang, , and K.-C. Hui. Mesh composition on models with arbitrary boundary topology. *IEEE Trans. Visualization & Computer Graphics*, 14:653–665, 2008. 1, 2
- [27] W. E. Lorensen and H. E. Cline. Marching cubes: A high resolution 3D surface construction algorithm. In *ACM Trans. Graph. (SIGGRAPH)*, volume 21, pages 163–169. ACM, 1987. 3, 7, 10, 12
- [28] L. Mescheder, M. Oechsle, M. Niemeyer, S. Nowozin, and A. Geiger. Occupancy networks: Learning 3D reconstruction in function space. In *CVPR*, 2019. 3
- [29] N. Mitra, M. Wand, H. Zhang, D. Cohen-Or, and M. Bokeloh. Structure-aware shape processing. In *Eurographics State-of-the-art Report (STAR)*, 2013. 1
- [30] K. Mo, P. Guerrero, L. Yi, H. Su, P. Wonka, N. Mitra, and L. Guibas. StructureNet: Hierarchical graph networks for 3D shape generation. In *SIGGRAPH Asia*, 2019. 3
- [31] S. Muralikrishnan, V. G. Kim, M. Fisher, and S. Chaudhuri. Shape unicode: A unified shape representation. *CVPR*, 2019. 3
- [32] J. J. Park, P. Florence, J. Straub, R. Newcombe, and S. Lovegrove. DeepSDF: Learning continuous signed distance functions for shape representation. In *CVPR*, 2019. 3
- [33] C. R. Qi, L. Yi, H. Su, and L. J. Guibas. PointNet++: Deep Hierarchical Feature Learning on Point Sets in a Metric Space. In *NeurIPS*, 2017. 3
- [34] D. Ritchie, S. Jobalia, and A. Thomas. Example-based authoring of procedural modeling programs with structural and continuous variability. *Comp. Graph. For.*, 37(2):401–413, 2018. 1, 2
- [35] R. Schmidt and K. Singh. Drag-and-drop surface composition. Technical Report CSRG-604, University of Toronto, 2009. 1, 2
- [36] N. Schor, O. Katzier, H. Zhang, and D. Cohen-Or. CompoNet: Learning to generate the unseen by part synthesis and composition. In *ICCV*, 2019. 2
- [37] A. Sharf, M. Blumenkrants, A. Shamir, and D. Cohen-Or. SnapPaste: An interactive technique for easy mesh composition. In *Pacific Graphics*, 2006. 1, 2

- [38] M. Sung, H. Su, V. G. Kim, S. Chaudhuri, and L. Guibas. ComplementMe: Weakly-supervised component suggestions for 3D modeling. *ACM Trans. Graph. (SIGGRAPH Asia)*, 2017. 1, 2
- [39] J. Talton, L. Yang, R. Kumar, M. Lim, N. Goodman, and R. Měch. Learning design patterns with Bayesian grammar induction. In *UIST*, pages 63–74, 2012. 2
- [40] H. Wang, N. Schor, R. Hu, H. Huang, D. Cohen-Or, and H. Huang. Global-to-local generative model for 3d shapes. In *SIGGRAPH Asia*, page 214. ACM, 2018. 3
- [41] P.-S. Wang, Y. Liu, Y.-X. Guo, C.-Y. Sun, and X. Tong. Adaptive O-CNN: A Patch-based Deep Representation of 3D Shapes. *ACM Transactions on Graphics (SIGGRAPH Asia)*, 37(6), 2018. 3
- [42] Y. Wang, S. Asafi, O. Van Kaick, H. Zhang, D. Cohen-Or, and B. Chen. Active co-analysis of a set of shapes. *SIGGRAPH Asia*, 31(6):165, 2012. 8
- [43] J. Wu, C. Zhang, T. Xue, W. T. Freeman, and J. B. Tenenbaum. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. In *NeurIPS*, 2016. 3
- [44] R. Wu, Y. Zhuang, K. Xu, H. Zhang, and B. Chen. PQ-NET: A generative part seq2seq network for 3d shapes. In *CVPR*, 2020. 2
- [45] Z. Wu, S. Song, A. Khosla, F. Yu, L. Zhang, X. Tang, and J. Xiao. 3D ShapeNets: A deep representation for volumetric shapes. In *CVPR*, 2015. 3
- [46] Z. Wu, X. Wang, D. Lin, D. Lischinski, D. Cohen-Or, and H. Huang. Sagnet: Structure-aware generative network for 3d-shape modeling. *SIGGRAPH Asia*, 38(4):91:1–91:14, 2019. 3
- [47] K. Xu, H. Zhang, D. Cohen-Or, and B. Chen. Fit and diverse: Set evolution for inspiring 3D shape galleries. *ACM Trans. Graph.*, 31(4):57:1–10, 2012. 1, 2
- [48] L. Yi, V. G. Kim, D. Ceylan, I.-C. Shen, M. Yan, H. Su, C. Lu, Q. Huang, A. Sheffer, and L. Guibas. A scalable active framework for region annotation in 3D shape collections. *SIGGRAPH Asia*, 35(6):1–12, 2016. 2, 5, 11
- [49] Y. Yu, K. Zhou, D. Xu, X. Shi, H. Bao, B. Guo, and H.-Y. Shum. Mesh editing with poisson-based gradient field manipulation. *ACM Trans. Graph.*, 23(4):644–651, 2004. 1, 2, 11
- [50] W. Yuan, T. Khot, D. Held, C. Mertz, and M. Hebert. Pcn: Point completion network. In *3DV*, 2018. 6, 12
- [51] M. E. Yumer and N. J. Mitra. Learning semantic deformation flows with 3D convolutional networks. In *ECCV*, 2016. 3
- [52] Y. Zheng, D. Cohen-Or, and N. J. Mitra. Smart variations: Functional substructures for part compatibility. *Comp. Graph. For.*, 32(2), 2013. 1
- [53] C. Zhu, K. Xu, S. Chaudhuri, R. Yi, and H. Zhang. SCORES: Shape composition with recursive substructure priors. *ACM Trans. Graph.*, 37(6), 2018. 2
- [54] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networks. In *ICCV*, 2017. 12

## A. Supplementary material

### A.1. Qualitative results

We show visual results on more examples of chairs in Figure 10, 11, 12, more mugs in 13, 14, and more airplanes in Figure 15, 16.

### A.2. Poisson mesh stitching

We use a Poisson-based approach to combine the synthesized joints with the meshes of the aligned parts to create a seamless final model. We first discretize the implicit function for the joints into a voxel grid of resolution  $128^3$  and convert it into a mesh using Marching Cubes [27]. The Poisson mesh stitching between this mesh and the mesh of the aligned parts has three phases: (i) finding corresponding loops between the input parts and the synthesized joints, (ii) removing redundant regions from the joint mesh and (iii) Poisson mesh blending.

As described in the method section of the main paper, we erode the part meshes to avoid potential topological mismatches. When eroding the part meshes, the open boundaries of the eroded parts are given as directed loops as shown in Figure 9 (a) and (c). To blend them with the synthesized joint mesh, corresponding loops in the joint mesh

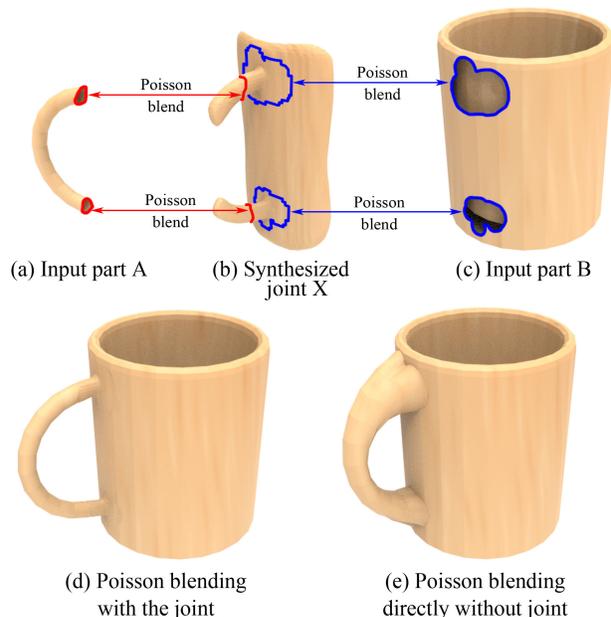


Figure 9: Poisson blending with synthesized joints. (b) shows the synthesized joint mesh for two input parts A and B. Corresponding loops are shown for the handle (red) and for the body (blue). (d) shows the output of our full pipeline which blends the parts with the synthesized joint region. (e) shows the result of naively blending the loops in A and B without the synthesized joint.

are needed. We employ a simple approach based on nearest neighbor search to establish these correspondences as follows.

Let us denote the loop in the input part as  $S$  and the loop in the synthesized joint as  $T$ . The joint is denoted as  $X$ . The first step is to define a metric to measure the distance between a point in  $S$  and a point in  $X$ . We consider both euclidean distance and normal consistency and define the distance as

$$D(\mathbf{p}, \mathbf{q}) = \|\mathbf{p} - \mathbf{q}\|_2 \cdot (2 - \mathbf{n}_p \mathbf{n}_q^T)$$

where  $\mathbf{n}_p$  is the surface normal of point  $p$ .

We randomly sample a point in  $S$  as initial point, and find its closest vertex in  $X$  as the first point in  $T$ . Then we keep adding new points to  $T$  iteratively. The algorithm is as follows.

---

**ALGORITHM 1:** Finding corresponding loop

---

```

Input: circular array  $S$  that contains the vertices
forming the loop in the input part;
Input: vertices  $X$  and undirected edges  $E$  of the joint;
Input: distance metric  $D$ ;
Output: queue  $T$  that contains the vertices from  $X$ 
forming the loop in the joint;
Initialize  $T = \emptyset$ ;
 $r =$  a random integer;
initS =  $S[r]$ ;
initT = the closest point to initS in  $X$ , according to  $D$ ;
 $T.push(initT)$ ;
 $r = r+1$ ;
nextS =  $S[r]$ ;
nextT = initT;
while nextS  $\neq$  initS do
     $N =$  neighbor vertices of nextT in  $X$  according to
     $E$ ;
    nextT = the closest point to nextS in  $N$ , according
    to  $D$ ;
    if nextT  $\neq T.head$  then
         $T.push(nextT)$ ;
    end
     $r = r+1$ ;
    nextS =  $S[r]$ ;
end
if  $T.head = T.tail$  then
     $T.pop()$ ;
    Return  $T$ ;
else
    Return None;
end

```

---

Note that there is a chance that the algorithm cannot find a loop, i.e., returning “None”. In our implementation, we

try several different initial points in  $S$  to find a loop. If they all fail, we ignore this loop and move to the next loop in the input parts. In addition, the returned loop may contain several smaller sub-loops. In this case, we only use the largest sub-loop.

After finding corresponding loops as shown in Figure 9 (b), there are redundant regions of the synthesized joint mesh which need to be removed before we can do Poisson mesh blending. Since the loops are directed, after splitting the joint mesh using the loops, each component can be marked as “inside the part” or “outside the part”. We remove the components that are detected as “inside the part”.

After the above two steps, we use an existing Poisson mesh blending algorithm [49] to merge the synthesized joint mesh with the aligned part meshes. The Poisson blending can operate directly on the input part meshes which ensures faithful preservation of the original mesh detail without surface resampling and remeshing. Note that after blending, there might be seams between the boundary of the joint and the boundary of the parts. We generate a ring of triangles to fill the gap between any two corresponding loops to obtain a seamless final model.

Figure 9 (e) shows the result of blending part A to part B directly without the joint. Without the joining mesh, the ends of the handle are swollen to fit the irregular open boundaries of the body. Compared to directly blending the input parts, a much better stitching result is obtained by using our synthesized joint mesh to connect the parts, as shown in Figure 9 (d).

### A.3. Details of part preprocessing

We use a part-wise point cloud representation in our neural processing. We create it by first sampling 16384 points on each shape using Poisson-disk sampling, segmenting the resulting point set into parts [48], and randomly sampling 2048 points on each part with uniform probability. Each resulting point cloud is normalized so that its centroid is at the origin and its bounding box has unit diameter. To ensure that parts can be merged together, we eliminate topological mismatches and poor segmentation boundaries by eroding the inputs around the joints. For each point, we compute the distance to the segmentation boundary and if it is less than a threshold  $\tau$ , the point is excluded. As an additional benefit, erosion also enables us to train the joint synthesis network in a strongly-supervised fashion, since the network has to complete the eroded regions in a training shape. Here, erosion puts the training and testing scenarios on a common footing by forcing both to process similarly eroded parts, instead of trying to synthetically simulate real-world cutting and joining errors for training.

#### A.4. Details of network architecture

In this section, we provide detailed architectures of the PointNet++’s used in our part alignment network and joint synthesis network. Recall that in our part alignment network, we use a PointNet++ to compute the shape features for input parts. We denote this PointNet++ as PointNet++ A. Also, in our joint synthesis network, each input part is assigned a two-branch point cloud encoder. The first branch of the part encoder is a PointNet++ network for extracting overall shape features from the parts, and we denote this PointNet++ as PointNet++ B. The second branch is another PointNet++ network for extracting localized shape features from the points that are close to the joint area, and we denote this PointNet++ as PointNet++ C.

A set abstraction layer of PointNet++ is denoted as  $SA(M, r, N, [l_1, \dots, l_d])$  where  $M$  is the number of local patches,  $r$  is the radius of balls that bound the patches,  $N$  is the number of sample points selected in each patch,  $[l_1, \dots, l_d]$  are the widths of fully-connected layers used in local PointNet.

The architectures of the three PointNet++ encoders are as follows.

PointNet++ A: Input  $\rightarrow SA(256, 0.2, 128, [64, 64, 128]) \rightarrow SA(128, 0.4, 128, [128, 128, 128]) \rightarrow SA(1, \text{inf}, \text{inf}, [128, 128, 128]) \rightarrow \text{feature}$

PointNet++ B: Input  $\rightarrow SA(256, 0.1, 128, [64, 64, 128]) \rightarrow SA(128, 0.2, 128, [128, 128, 128]) \rightarrow SA(1, \text{inf}, \text{inf}, [128, 128, 128]) \rightarrow \text{feature}$

PointNet++ C: Input  $\rightarrow SA(256, 0.05, 128, [32, 32, 64]) \rightarrow SA(128, 0.1, 128, [64, 64, 128]) \rightarrow SA(1, \text{inf}, \text{inf}, [128, 128, 128]) \rightarrow \text{feature}$

#### A.5. Details of network training

The part alignment network is trained for 200 epochs with batch size 8. We use an Adam optimizer with learning rate 0.001. The training time for each data category is provided in table 5.

|                         | Chair | Airplane | Mug |
|-------------------------|-------|----------|-----|
| Part alignment network  | 3.0   | 2.1      | 0.1 |
| Joint synthesis network | 32.7  | 17.4     | 1.4 |

Table 5: Training time (hours) on an RTX 2080 Ti GPU.

As the point cloud encoders and the IM-decoder in our joint synthesis network require different learning rates for training, we first pre-train the two-branch point cloud encoders for 100 epochs by pairing it with a fully connected decoder [1] for reconstructing 3D shape parts. The learning rate starts at  $10^{-3}$  and is halved after every 20 training

epochs, until reaching  $1.25 \times 10^{-4}$ . We pair the pretrained point cloud encoders with the IM-decoder, and train the IM-decoder with the loss function  $L_{\text{joint}} = L_{\text{mse}}$  for 80 epochs, with learning rate  $10^{-4}$ . To reduce training time and get more robust output, we train the IM-decoder in a coarse-to-fine manner. First, the network is trained by downsampled point samples (2k samples). Then, we double the number of samples every 20 epochs, until the number reaches 32k. Finally, we train the encoders and the IM-decoder together with the loss function  $L_{\text{joint}} = L_{\text{mse}} + \alpha L_{\text{match}}$  for another 80 epochs. The default batch size is 1. The optimizer is Adam.

In an early stage of our experiments, we noticed that the joint synthesis network did not work well in predicting connections for chair back with bars. After a close examination of the chair dataset, we found this is because only about 10% of chairs have bars in their back. Therefore, we did a simple data augmentation for the chair dataset by duplicating the examples with back bars by 4 times (We did the same augmentation for all methods in our comparison experiments). After the data augmentation, our joint synthesis network works pretty well for chairs with back bars. For example, the second chair in the teaser of the paper, and the second row in Figure 10.

#### A.6. Details of baseline networks

We consider two voxel-based shape completion baselines that can be applied to our problem: 3DCNN and 3DResNet. In the two baselines, we use architectures that perform shape completion akin to image translation tasks. Unlike encoder-decoder architectures such as AtlasNet [13] and PCN [50], these methods do not have a bottleneck layer, and they only modify input in a relatively local manner. Thus, they do not have the challenge of reconstructing the entire shape from the latent code, and might be better at preserving the geometry of the original parts. The 3DCNN and 3DResNet have similar architectures, but the latter uses residual skip connections. 3DCNN has 12 layers that gradually downsamples the input voxels to  $16^3$  and upsamples to the output resolution; 3DResNet mimics the ResNet [14] structure for image translation used in CycleGAN [54]: we simply replace 2D convolutions with 3D convolutions. The resolution of input and output voxels is  $128^3$  due to the hardware (GPU memory) limitations. We train the two baseline networks on individual categories for 1 million iterations each. When testing, we use marching cubes [27] to obtain the meshes from the output  $128^3$  voxels.



Figure 10: More examples of chairs.



Figure 11: More examples of chairs.

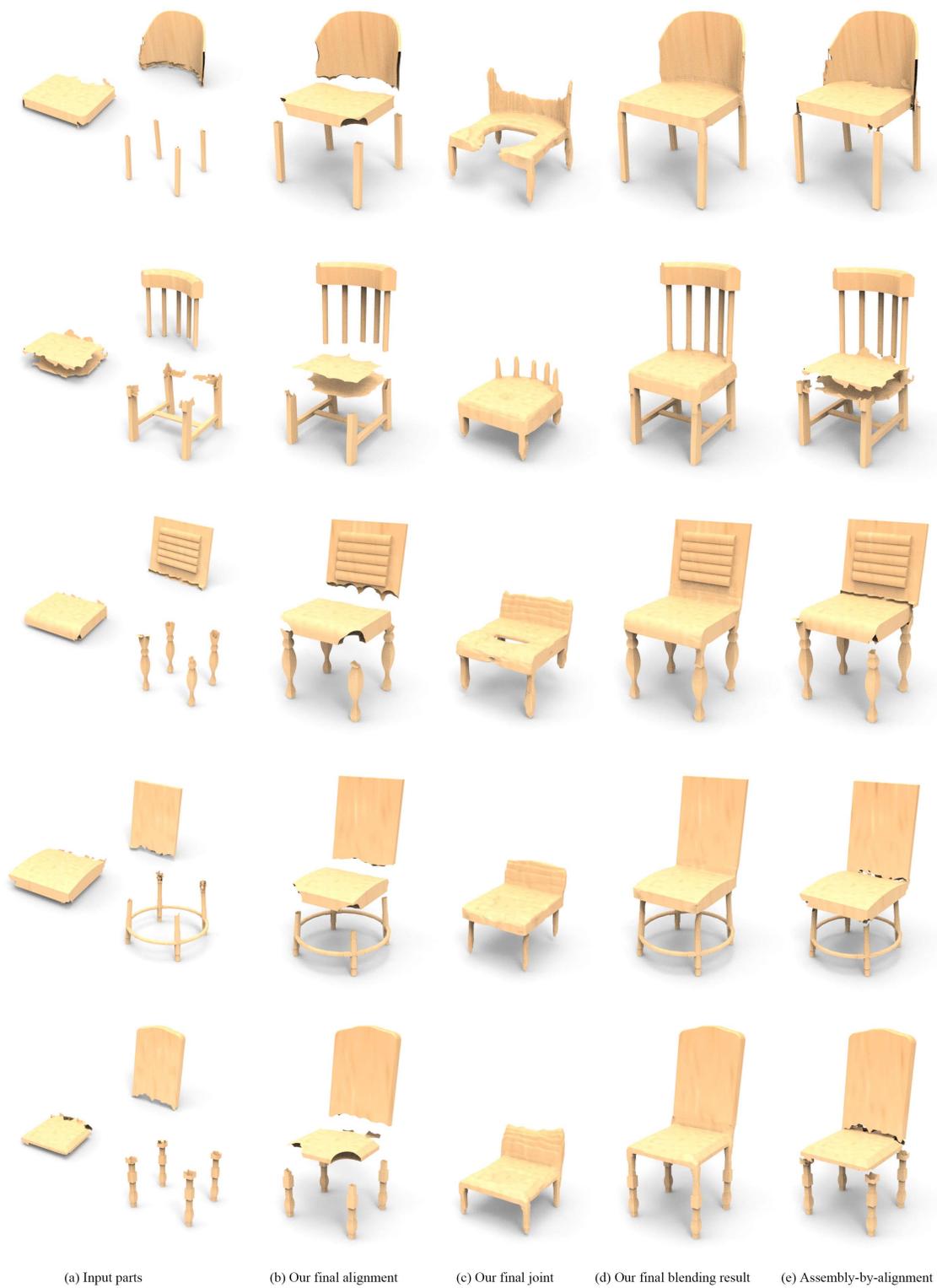


Figure 12: More examples of chairs.



Figure 13: More examples of mugs.



Figure 14: More examples of mugs.



Figure 15: More examples of airplanes.



Figure 16: More examples of airplanes.