A. Supplementary material

A.1. Qualitative results

We show visual results on more examples of chairs in Figure 2, 3, 4, 5, more mugs in Figure 6, 7, 8, 9, and more airplanes in Figure 10, 11.

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 [4]. 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 1 (a) and (c). To blend them with the synthesized joint mesh, corresponding loops in the joint mesh

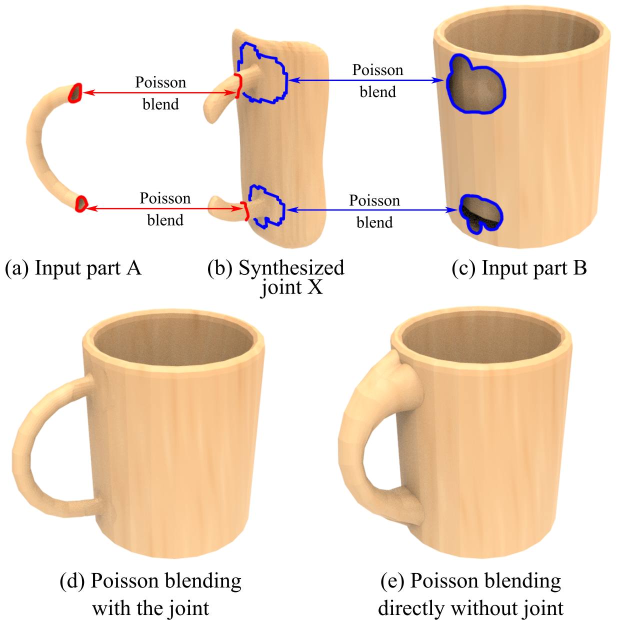

Figure 1: Poisson blending with synthesized joints. (b) shows the synthesized joint mesh for two input parts A and B. Corresponding loops are shown for the handle (red) and for the body (blue). (d) shows the output of our full pipeline which blends the parts with the synthesized joint region. (e) shows 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

$$D(\boldsymbol{p}, \boldsymbol{q}) = ||\boldsymbol{p} - \boldsymbol{q}||_2 \cdot (2 - \boldsymbol{n_p}\boldsymbol{n_q}^T)$$

where $\boldsymbol{n_p}$ is the surface normal of point $\boldsymbol{p}$.

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 1 (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 [6] 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 1 (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 1 (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 [5], 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 τ, 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 $SA(M, r, N, [l_1, ..., l_d])$ 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, $[l_1, ..., l_d]$ 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, inf, inf, [128, 128, 128]) $\rightarrow$ 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, inf, inf, [128, 128, 128]) $\rightarrow$ 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, inf, inf, [128, 128, 128]) $\rightarrow$ 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 1.

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

Table 1: 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 1.25×10^{-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 2.

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 [2] and PCN [7], 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 [3] structure for image translation used in CycleGAN [8]: 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 [4] to obtain the meshes from the output 128^3 voxels.

References

[1] P. Achlioptas, O. Diamanti, I. Mitliagkas, and L. Guibas. Learning representations and generative models for 3d point clouds. In *ICML*, 2018. 2

[2] 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

[3] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *CVPR*, 2016. 3

[4] 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. 1, 3

[5] 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

[6] 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. 2

[7] W. Yuan, T. Khot, D. Held, C. Mertz, and M. Hebert. Pcn: Point completion network. In *3DV*, 2018. 3

[8] J.-Y. Zhu, T. Park, P. Isola, and A. A. Efros. Unpaired image-to-image translation using cycle-consistent adversarial networkss. In *ICCV*, 2017. 3

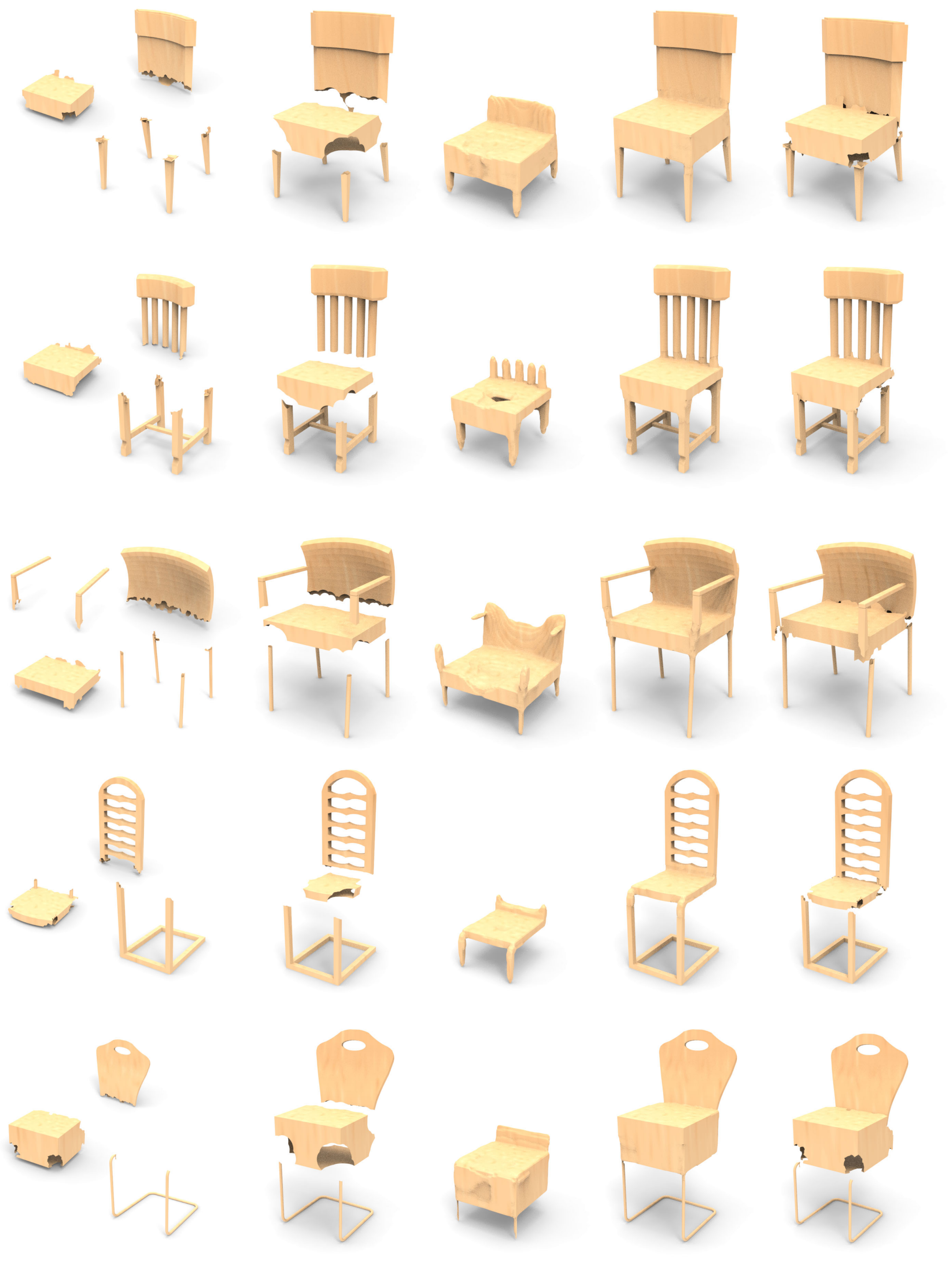

Figure 2: More examples of chairs.

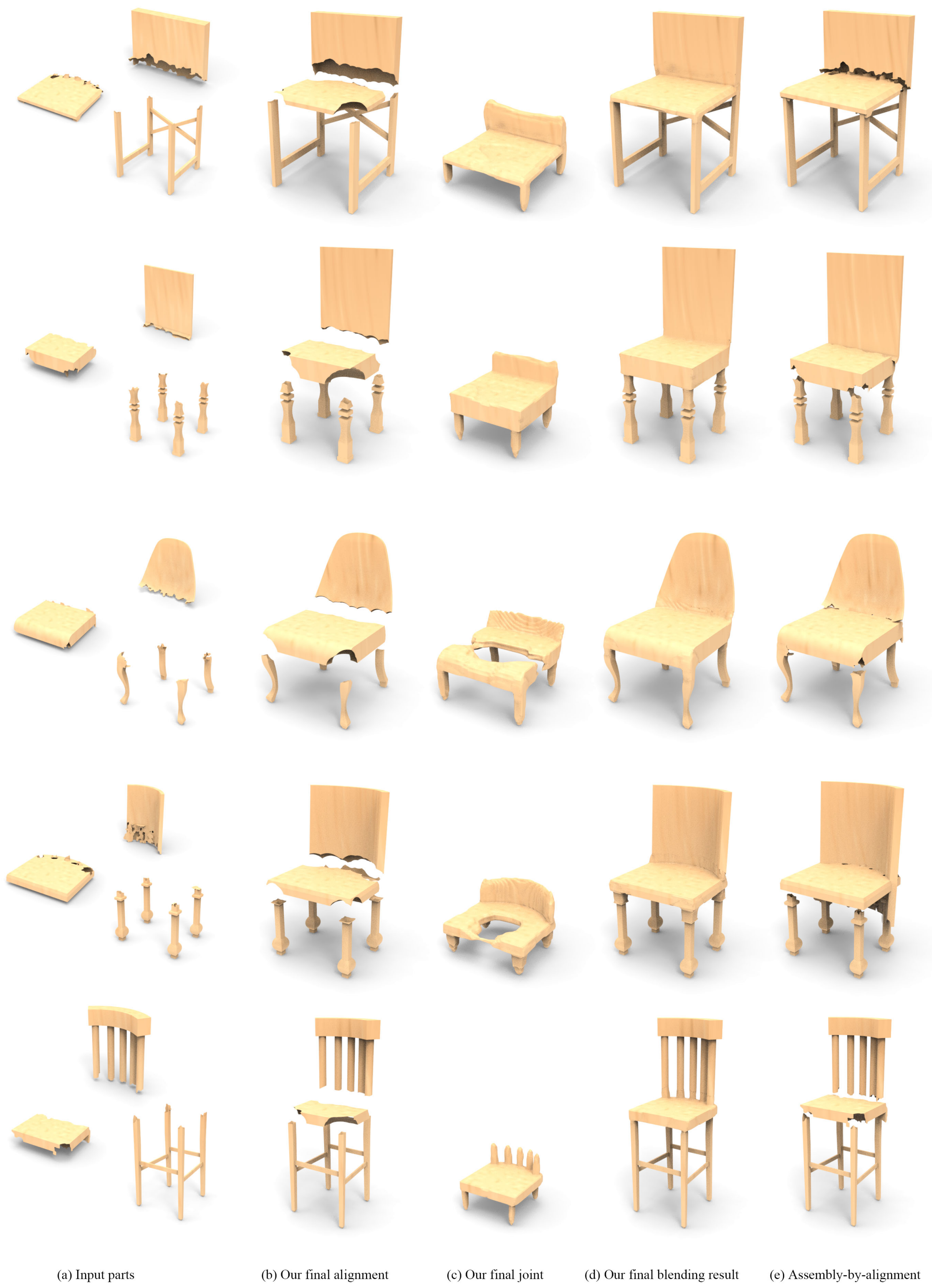

Figure 3: More examples of chairs.

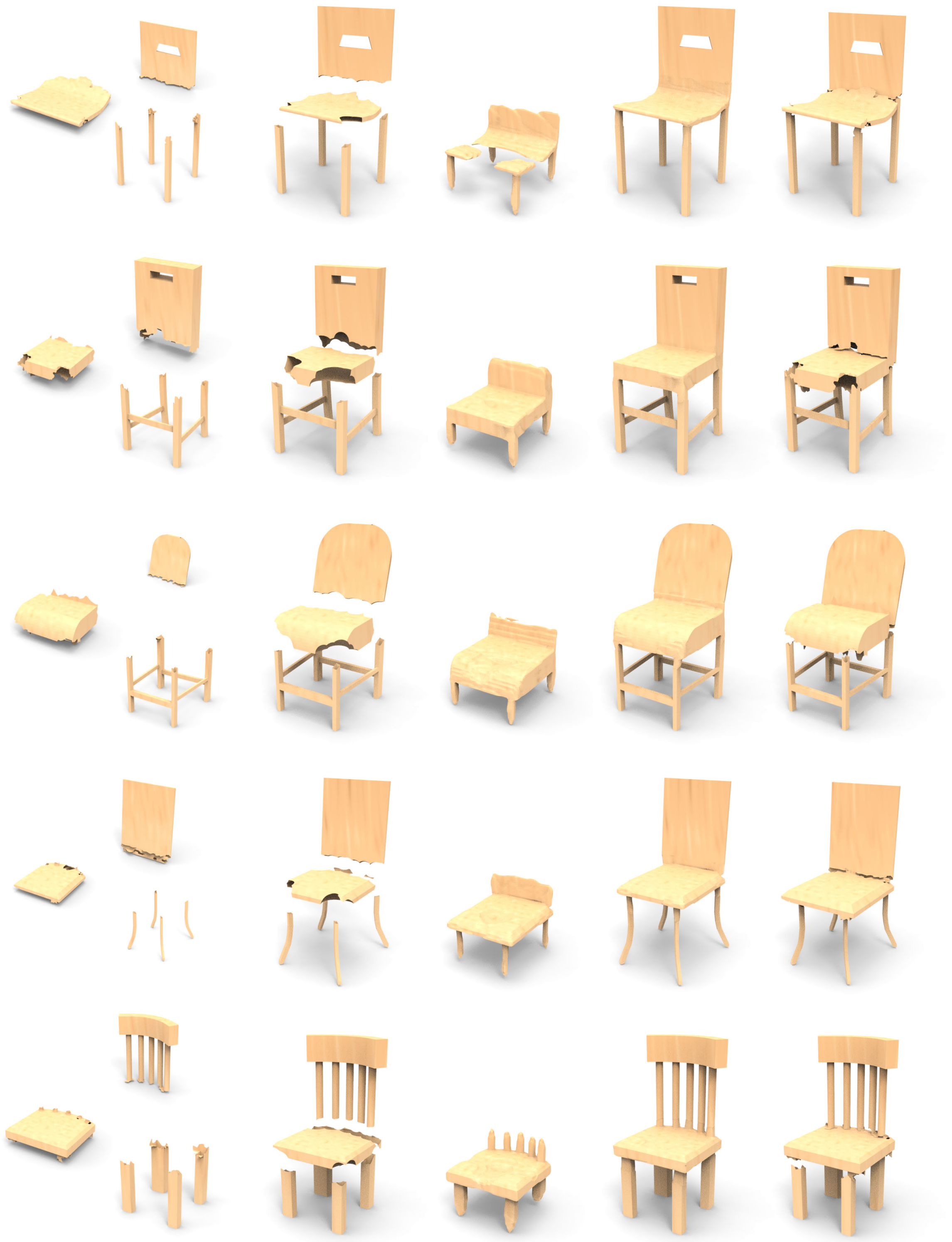

Figure 4: More examples of chairs.

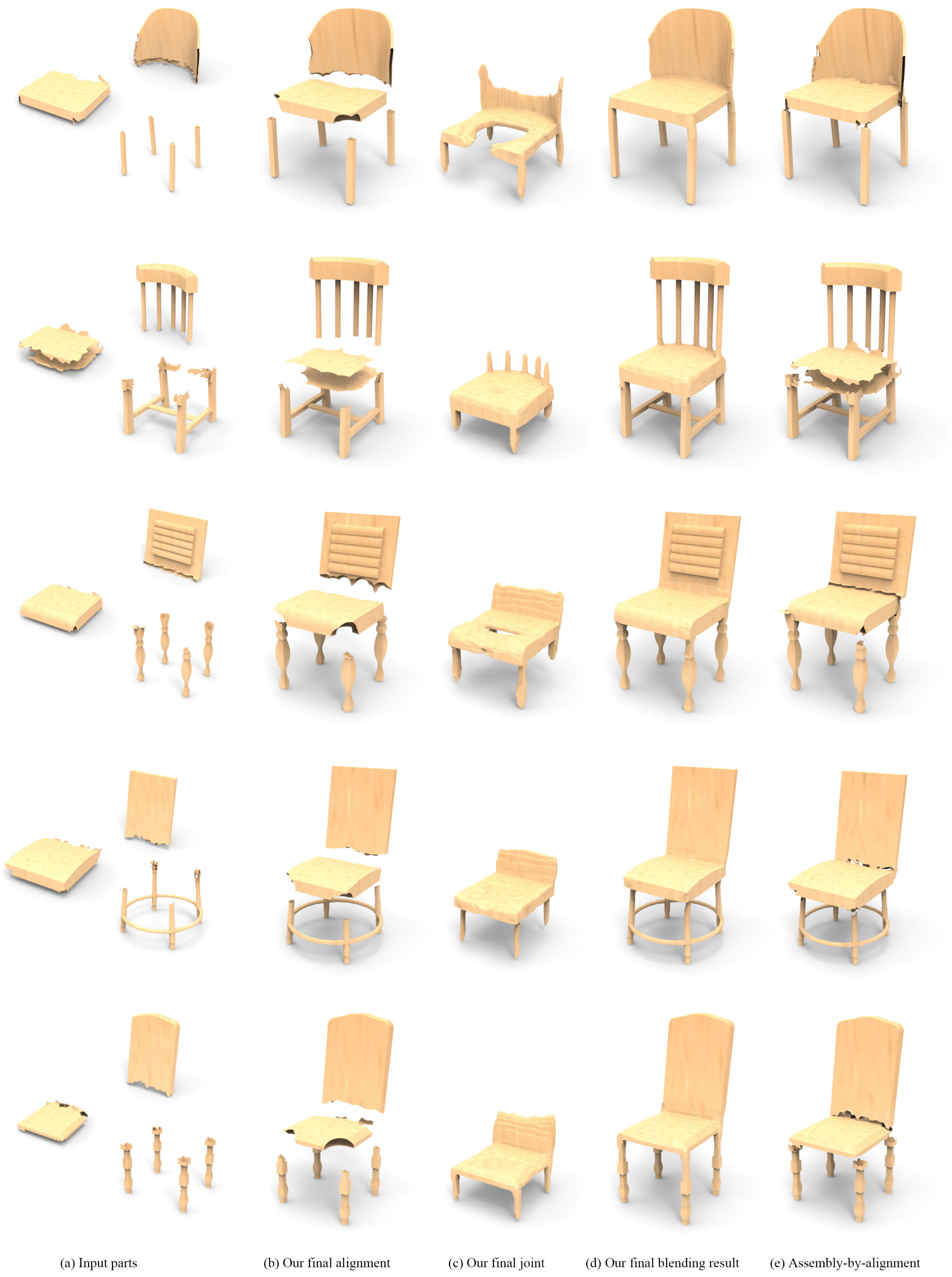

Figure 5: More examples of chairs.

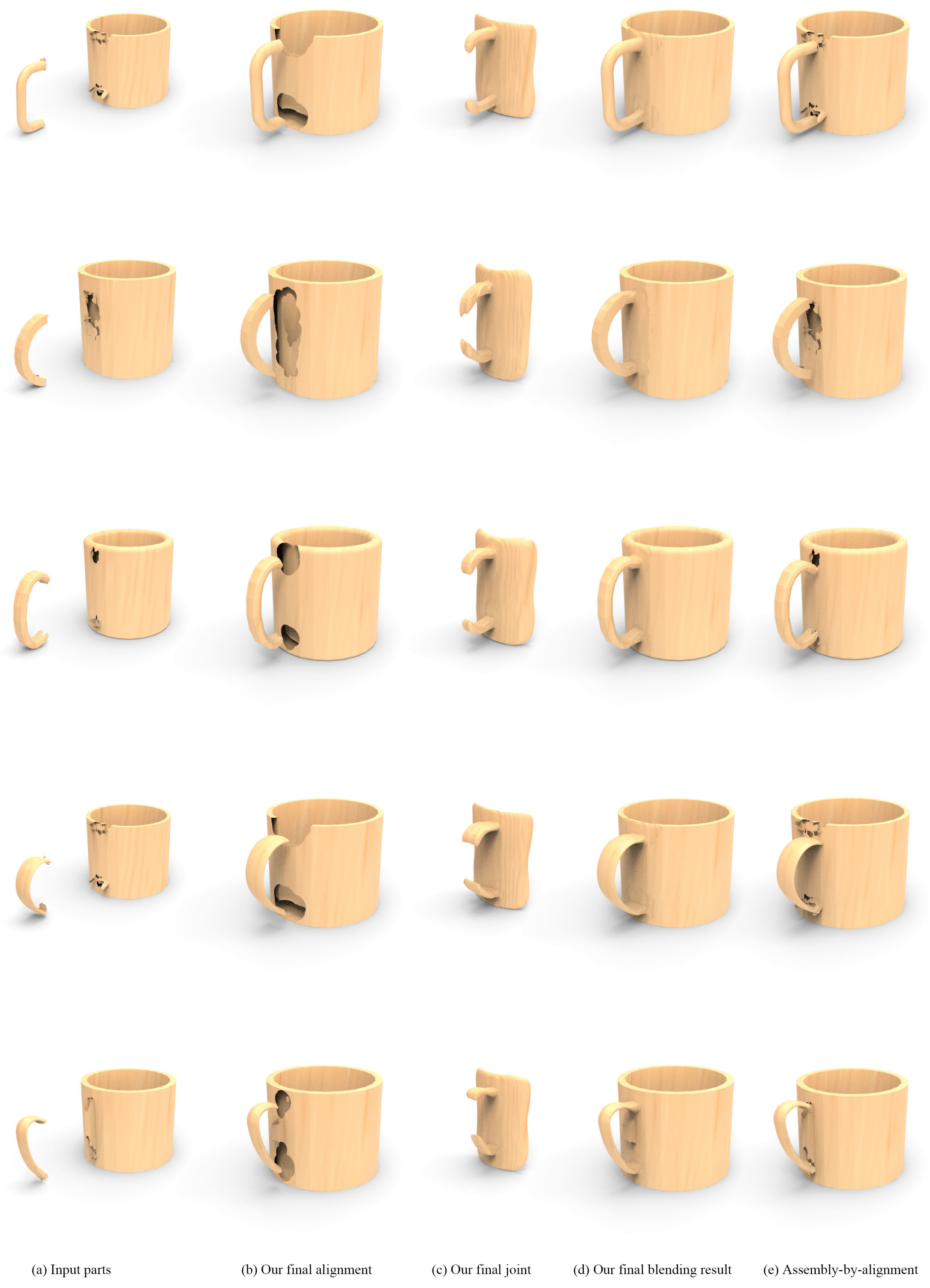

Figure 6: More examples of mugs.

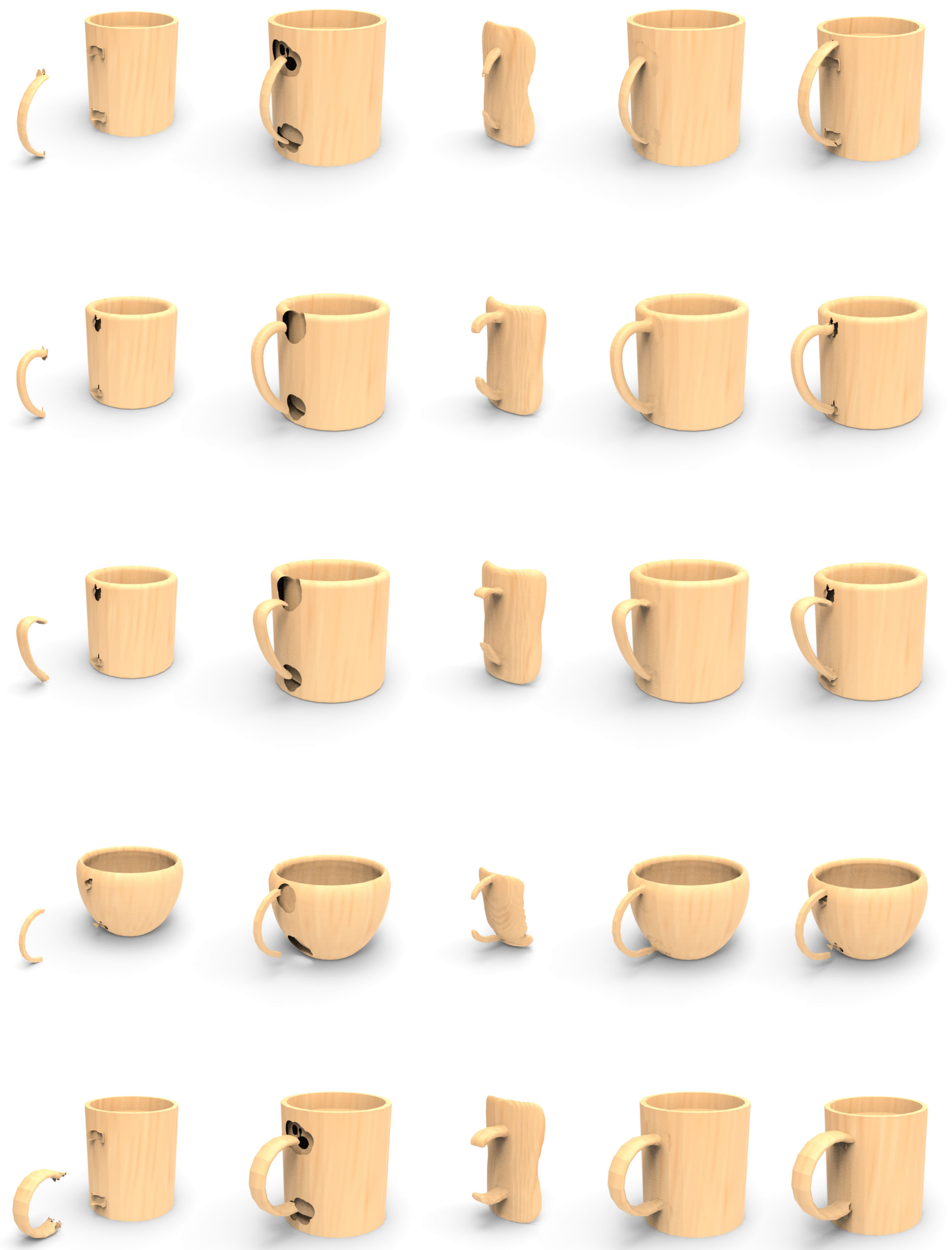

Figure 7: More examples of mugs.

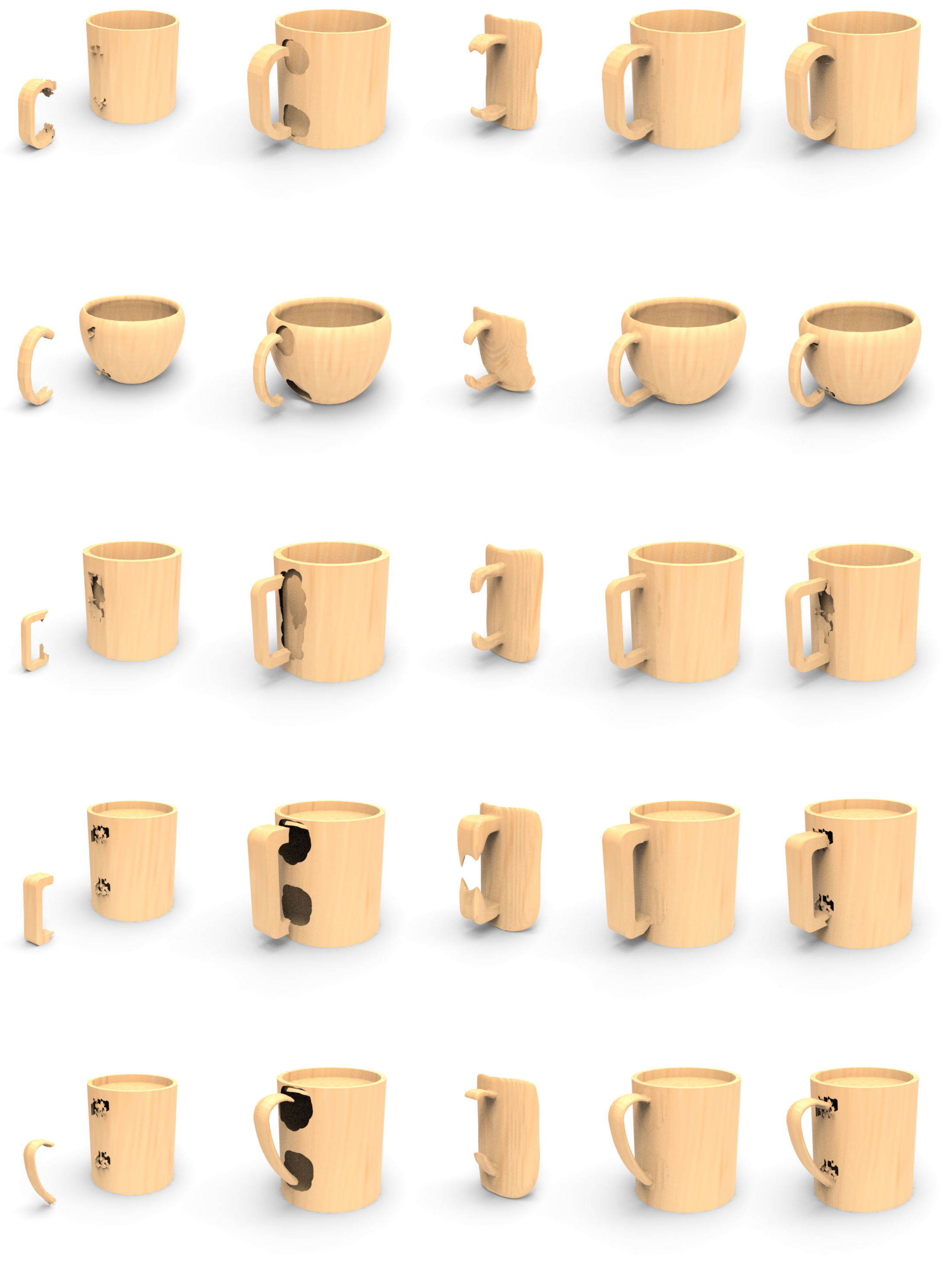

Figure 8: More examples of mugs.

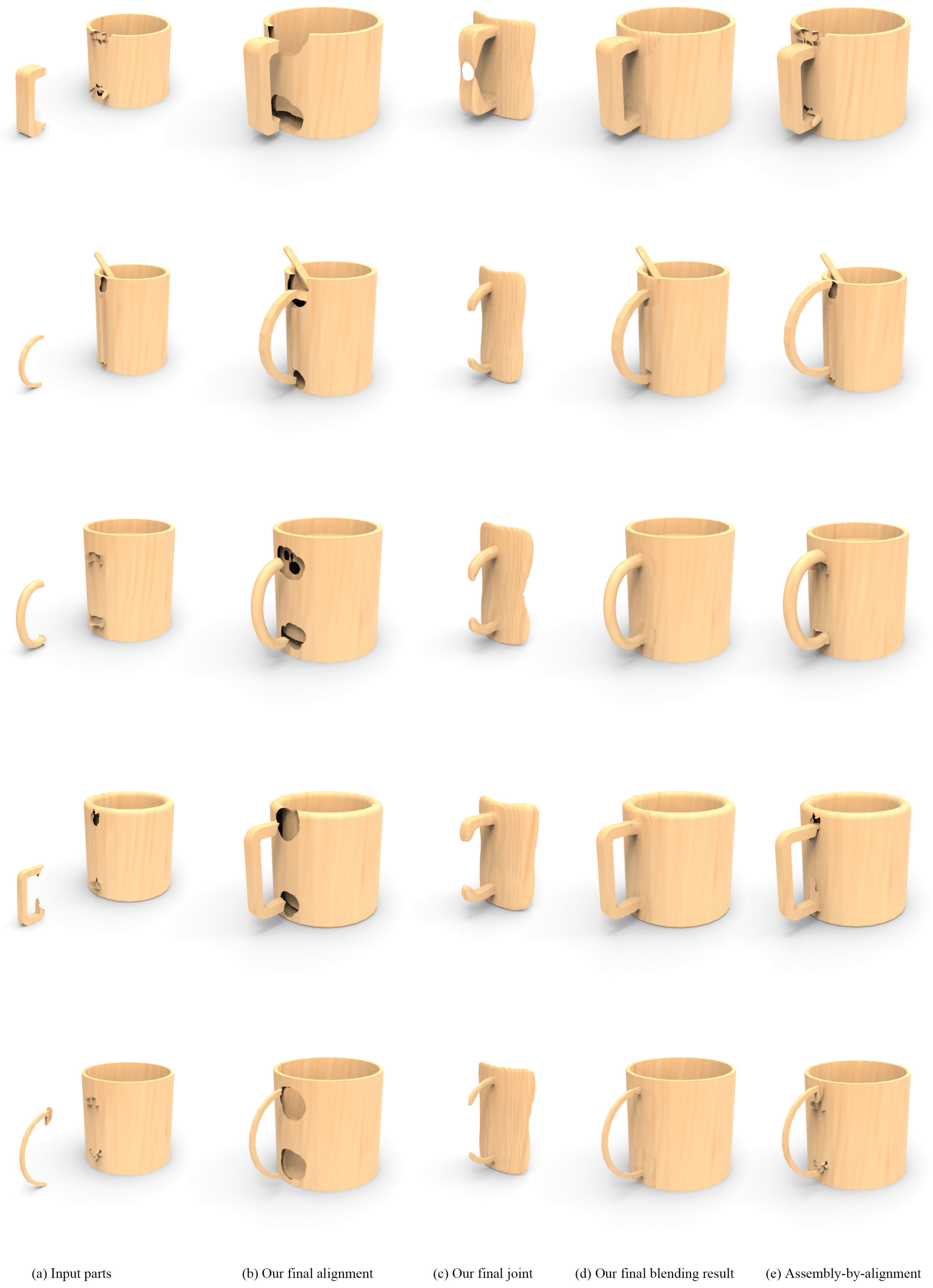

Figure 9: More examples of mugs.

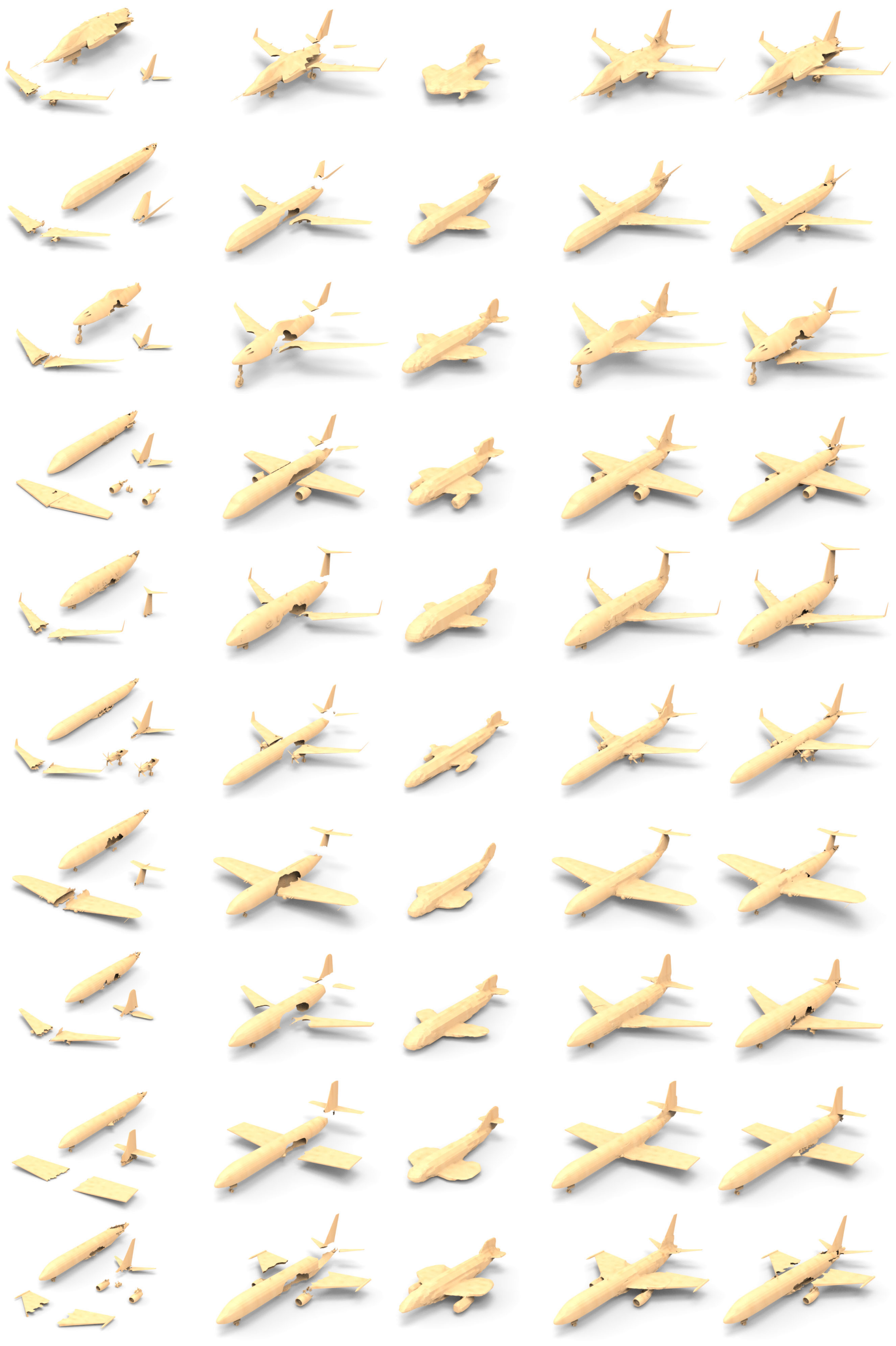

Figure 10: More examples of airplanes.

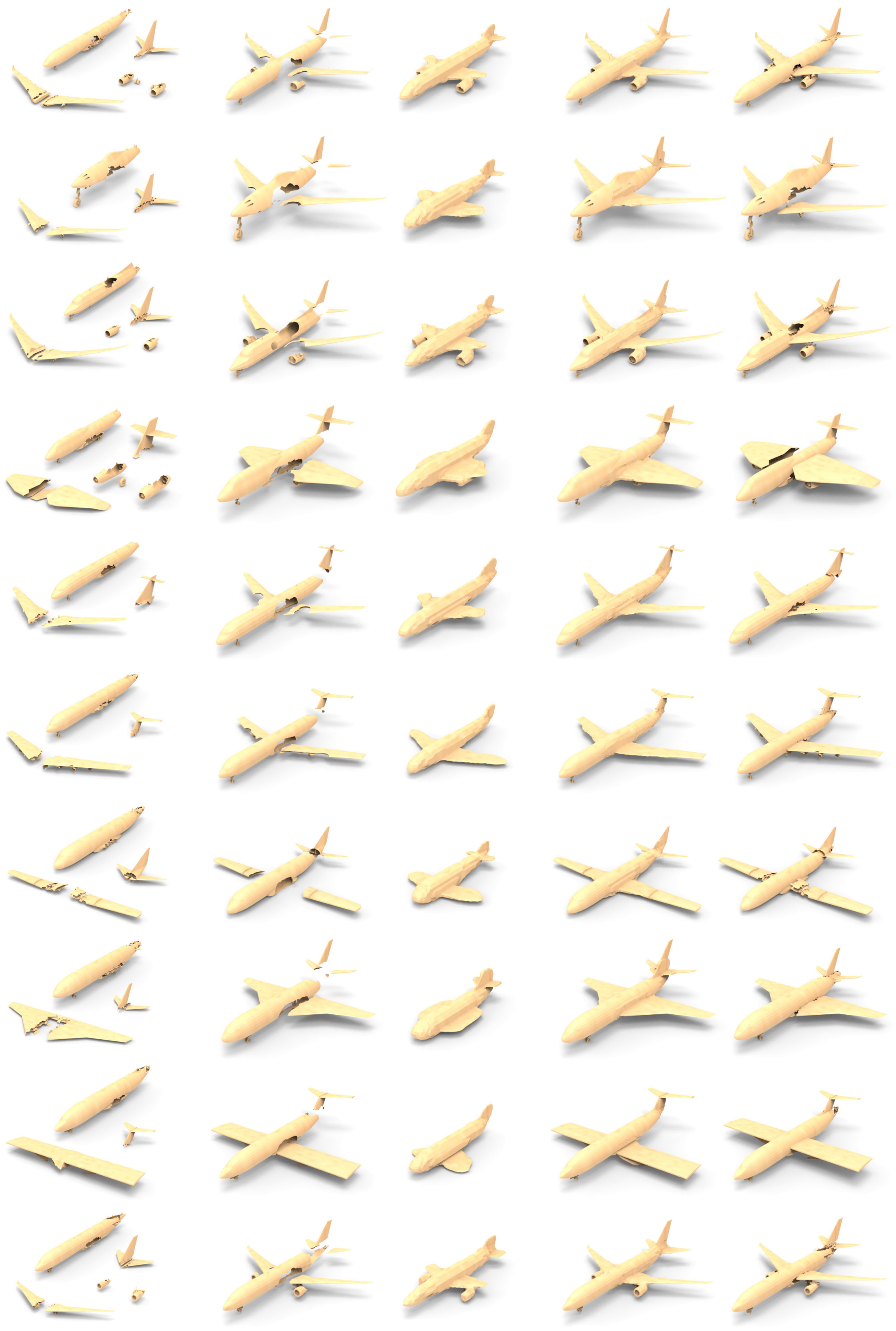

Figure 11: More examples of airplanes.