CSGNet: Neural Shape Parser for Constructive Solid Geometry Supplementary Material

1. Supplementary

In this supplementary material, we include the following topics in more detail: a) synthetic dataset creation in the 2D and the 3D case, b) neural network architecture used in our experiments, c) more qualitative results on our test dataset.

1.1. Dataset

Synthetic 2D shapes. We use the grammar described in the Section 4.1 to create our 2D dataset. The dataset is created by randomly generating programs of lengths 3 to 13 following the grammar. While generating these programs we impose additional restrictions as follows: a) Primitives must lie completely inside the canvas, b) Each operation changes the number of ON pixels by at least a threshold set to 10% of sum of pixels in two shapes. This avoids spurious operations such as subtraction between shapes with little overlap. c) The number of ON pixels in the final image is above a threshold. d) The previous rules promotes programs with the $union$ operation. To ensure a balanced dataset we boost the probabilities of generating programs with $subtract$ and $intersect$ operations. Finally we remove duplicates. We only use upright, equilateral triangles and upright squares. Note that locations (L) are discretized to lie on square grid with spacing of 8 units and size (R) are discretized with spacing of 4 units. Figure 1 shows examples from our dataset.

Synthetic 3D shapes. We use the grammar described in the Section 4.1 to create our 3D dataset. While generating shapes we followed a strategy similar to the 2D case. For 3D case, we only use programs of up to length 7 (up to 4 shape primtives and upto 3 boolean operations). Note that the cube and cylinder are upright. The dataset contains $64 \times 64 \times 64$ voxel-grid shapes and program pairs. Also note that locations (L) are discretized to lie on cubic grid with spacing of 8 units, and size (R) and height (H) are discretized with spacing of 4 units.

CSG execution engine. We implemented a CSG engine that reads the instructions one by one. If it encounters a primitive (e.g. `c(32, 32, 16)`) it draws it on an empty canvas and pushes it on to a stack. If it encounters an operation (e.g. `union`, `intersect`, or `subtract`) it pops the top two canvases on its stack, applies the operation to them, and pushes the output to the top of the stack. The execution stops when no instructions remain at which point the top canvas represents the result. The above can be seen as a set of shift and reduce operations in a LR-parser [1]. Figure 2 describes execution procedure to induce programs for 3D shapes.

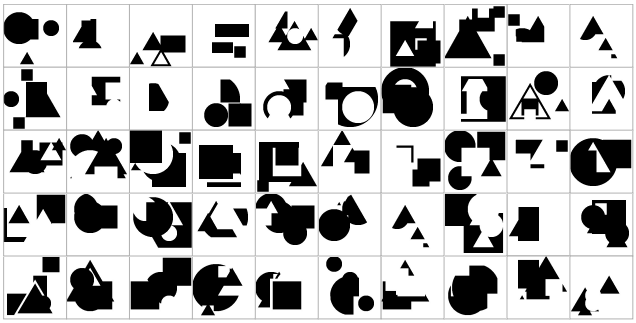

Figure 1. **Random samples from our synthetic 2D dataset.**

Layers	Output
Input image	$64 \times 64 \times 1$
Dropout(Relu(Conv: 3×3, $1 \rightarrow 8$))	$64 \times 64 \times 8$
Max-pool(2×2)	$32 \times 32 \times 8$
Dropout(Relu(Conv: 3×3, $8 \rightarrow 16$))	$32 \times 32 \times 16$
Max-pool(2×2)	$16 \times 16 \times 16$
Dropout(Relu(Conv: 3×3, $16 \rightarrow 32$))	$16 \times 16 \times 32$
Max-pool(2×2)	$8 \times 8 \times 32$
Flatten	2048

Table 1. **Encoder architecture for 2D shapes experiments.** The same architecture is used in all experiments in the Section 4.3.1.

1.2. Network Architecture

Architecture for 2D shape experiments. Table 1 shows the CNN architecture used as the encoder. The input I is an image of size 64×64 and output $\Phi(I)$ is a vector of size 2048. Table 2 describes the architecture used in the decoder. The RNN decoder is based on a GRU unit that at every time step takes as input the encoded feature vector and previous instruction encoded as a 128 dimensional vector obtained

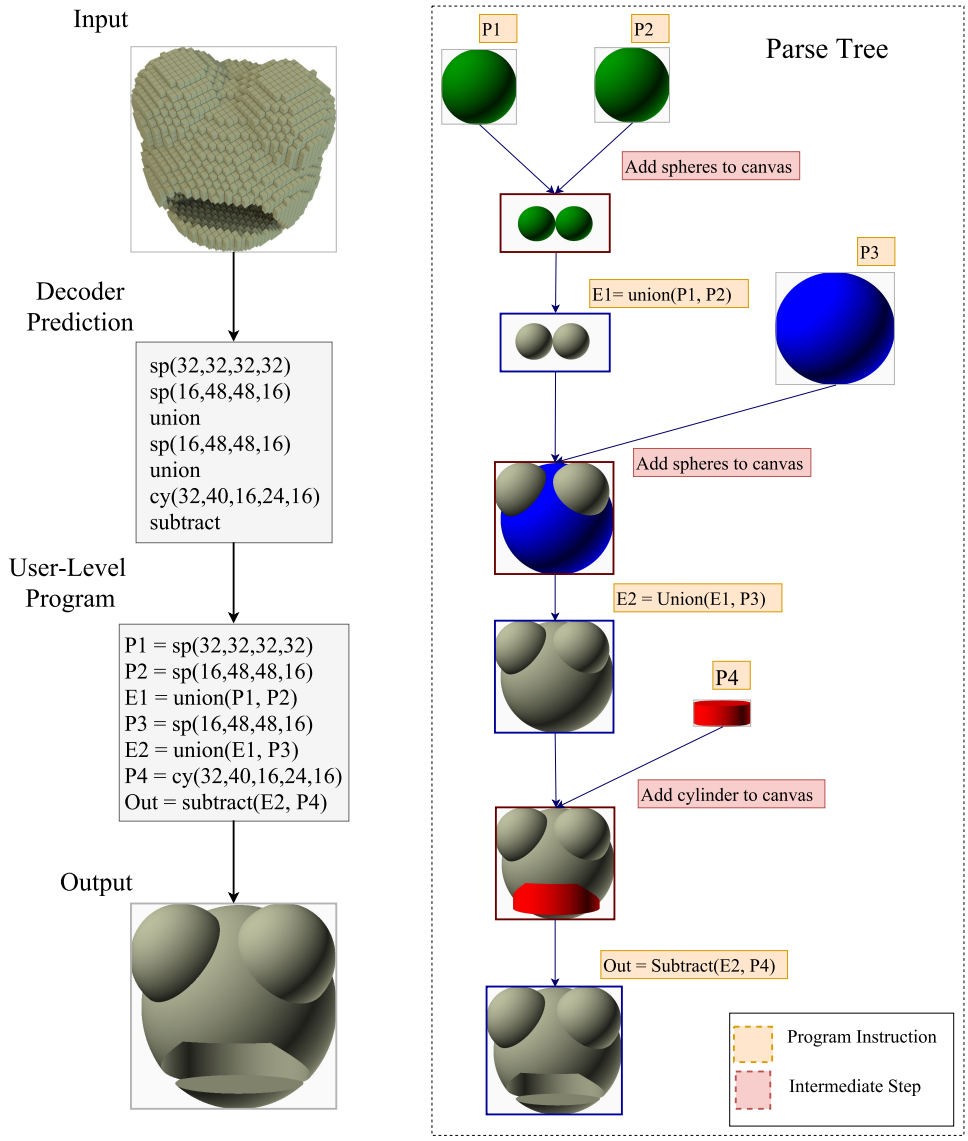

Figure 2. **Detailed execution procedure followed by an induced CSG program in a characteristic 3D case.** The input is a voxel based representation of size $64 \times 64 \times 64$. The RNN decoder produces a program, which can be executed following the grammar described in the Section 1.1, to give the output shown at the bottom. The user-level program is shown for illustration. On the right side is shown a parse tree corresponding to the execution of the program.

by a linear mapping of the 401 dimensional one-hot vector representation. At first time step, the previous instruction vector represents the `START` symbol. Embedded vector of previous instruction is concantenated with $\Phi(I)$ and is input to the GRU. The hidden state of GRU is passed through two dense layer to give a vector of dimension 400, which after `softmax` layer gives a probability distribution over instructions. The output distribution is over 396 different shape primitives, 3 operations (`intersect`, `union` and `subtract`) and a `STOP`. We exclude the `START` symbol from the output probability distribution. Note that the circle, triangle or square at a particular position in the image and of a particular size represents an unique primitive. For example, $c(32, 32, 16)$, $c(32, 28, 16)$, $s(12, 32, 16)$ are different shape primitives.

Architecture for 3D shape experiments. Input to 3D shape encoder (3DCNN) is a voxel grid I of size 64 x 64 x 64 and outputs an encoded vector $\Phi(I)$ of size 2048, as shown in the Table 3. Similar to the 2D case, at every time step, GRU takes as input the encoded feature vector and previous ground truth instruction. The previous ground truth instruction is a 6636-dimensional (also includes the `start` symbol) one-hot vector, which gets converted to a

Index	Layers	Output
1	Input shape encoding	2048
2	Input previous instruction	401
3	Relu(FC $(401 \rightarrow 128)$)	128
4	Concatenate (1, 3)	2176
5	Drop(GRU (hidden dim: 2048))	2048
6	Drop(Relu(FC$(2048 \rightarrow 2048)$))	2048
7	Softmax(FC$(2048 \rightarrow 400)$)	400

Table 2. **Decoder architecture for 2D shapes experiments.** The same architecture is used for all experiments in the Section 4.3.1. FC: Fully connected dense layer, Drop: dropout layer with 0.2 probability. Dropout on GRU are applied on outputs but not on recurrent connections.

Layers	Output
Input Voxel	$64 \times 64 \times 64 \times 1$
Relu(Conv3d: $4 \times 4 \times 4, 1 \rightarrow 32$)	$64 \times 64 \times 64 \times 32$
BN(Drop(Max-pool$(2 \times 2 \times 2)$))	$32 \times 32 \times 32 \times 32$
Relu(Conv3d: $4 \times 4, 32 \rightarrow 64$)	$32 \times 32 \times 32 \times 64$
BN(Drop(Max-pool$(2 \times 2 \times 2)$))	$16 \times 16 \times 16 \times 64$
Relu(Conv3d: $3 \times 3, 64 \rightarrow 128$))	$16 \times 16 \times 16 \times 128$
BN(Drop(Max-pool$(2 \times 2 \times 2)$))	$8 \times 8 \times 8 \times 128$
Relu(Conv3d: $3 \times 3, 128 \rightarrow 256$))	$8 \times 8 \times 8 \times 256$
BN(Drop(Max-pool$(2 \times 2 \times 2)$))	$4 \times 4 \times 4 \times 256$
Relu(Conv3d: $3 \times 3, 256 \rightarrow 256$))	$4 \times 4 \times 4 \times 256$
BN(Drop(Max-pool$(2 \times 2 \times 2)$))	$2 \times 2 \times 2 \times 256$
Flatten	2048

Table 3. **Encoder architecture for 3D shape experiments.** Drop: dropout layer, BN: batch-normalization layer and Drop: dropout layer with 0.2 probability.

Index	Layers	Output
1	Input shape encoding	2048
2	Input previous instruction	6636
3	Relu(FC$(6636 \rightarrow 128)$)	128
4	Concatenate (1, 3)	2176
5	Drop(GRU (hidden dim: 1500))	1500
6	Drop(Relu(FC$(1500 \rightarrow 1500)$))	1500
7	Softmax(FC$(1500 \rightarrow 6635)$)	6635

Table 4. **Decoder network architecture for 3D shapes experiments.** FC: Fully connected dense layer, Drop: dropout layer with 0.2 probability. Dropout on GRU are applied on outputs but not on recurrent connections.

fixed 128-dimensional vector using a learned embedding layer. At first time step the last instruction vector represents the `START` symbol. Embedded vector of previous instruction is concatenated with $\Phi(I)$ and is input to the GRU. The hidden state of GRU is passed through two dense layers to give a vector of dimension 6635, which after Softmax layer gives a probability distribution over instructions. The output distribution is over 6631 different shape primitives, 3 operations (`intersect`, `union` and `subtract`) and a `STOP`. We exclude the `START` symbol from the output probability distribution. Similar to 2D case, $cu(32, 32, 16, 16)$, $cu(32, 28, 16, 12)$, $sp(12, 32, 16, 28)$ are different shape primitives. Table 4 shows details of decoder.

1.3. Qualitative Evaluation

In this section, we show more qualitative results on different dataset. We first show peformance of our CSGNet trained using only Supervised learning on 2D synthetic dataset, and we compare top-10 results from nearest neighbors and and top-10 results from beam search, refer to the Figure 3 and 4. Then we show performance of our full model (using RL + beam search + visually guided search) on CAD 2D shape dataset, refer to the Figure 5 and 6.

References

[1] D. E. Knuttt. On the translation of languages from left to right. 1

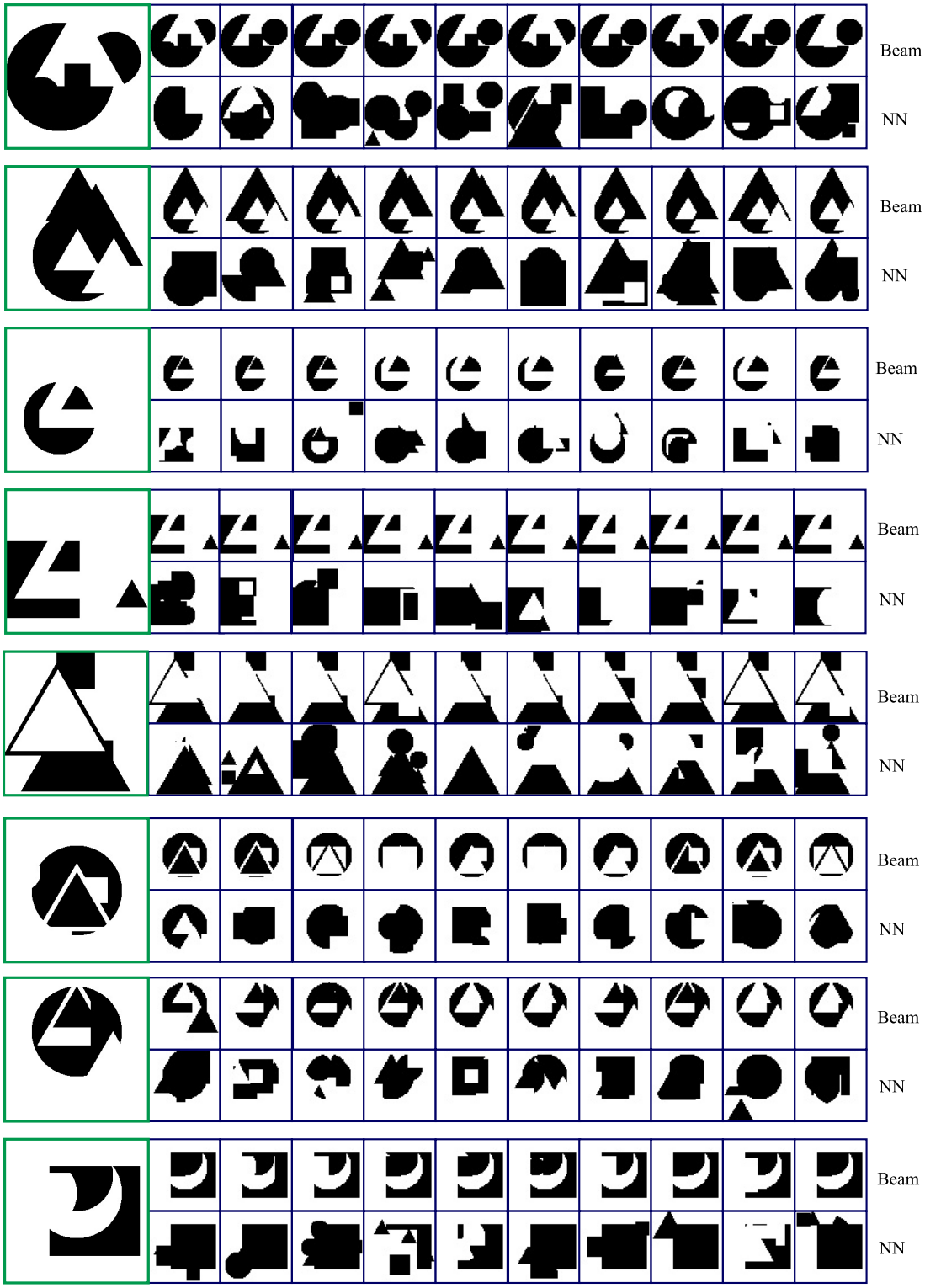

Figure 3. **Qualitative evaluation on 2D synthetic dataset.** In green outline is the groundtruth, top row represent top-10 beam search results, bottom row represents top-10 nearest neighbors.

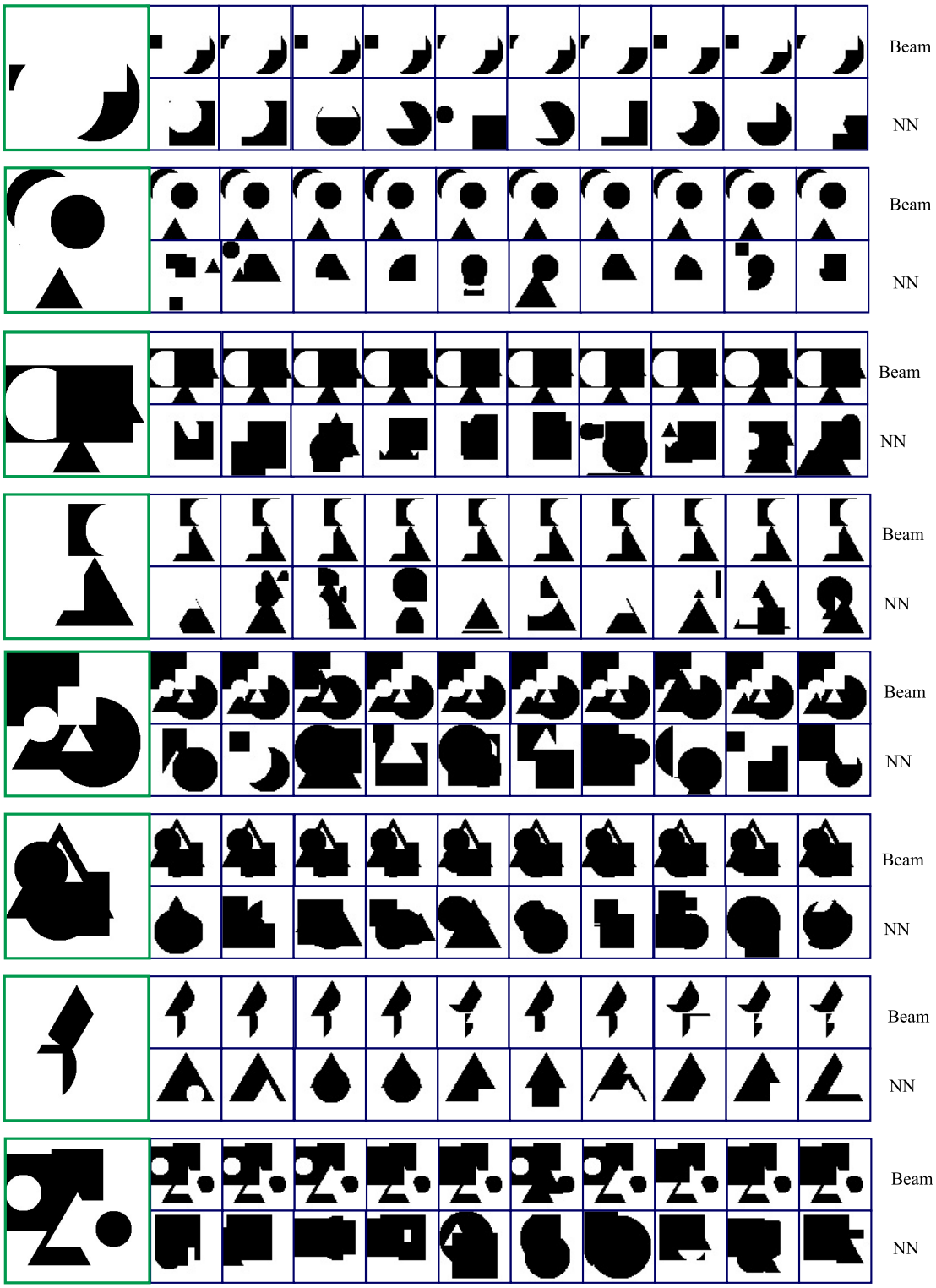

Figure 4. **Qualitative evaluation on 2D synthetic dataset.** In green outline is the groundtruth, top row represent top-10 beam search results, bottom row represents top-10 nearest neighbors.

Figure 5. **Performance of our full model on 2D CAD images**. a) Input image, b) output from our full model, c) Outlines of primitives present in the generated program, triangles are in green, squares are in blue and circles are in red d) Predicted program. s, c and t are shape primitives that represents $square$, $circle$ and $triangle$ respectively, and $union$, $intersect$ and $subtract$ are boolean operations.

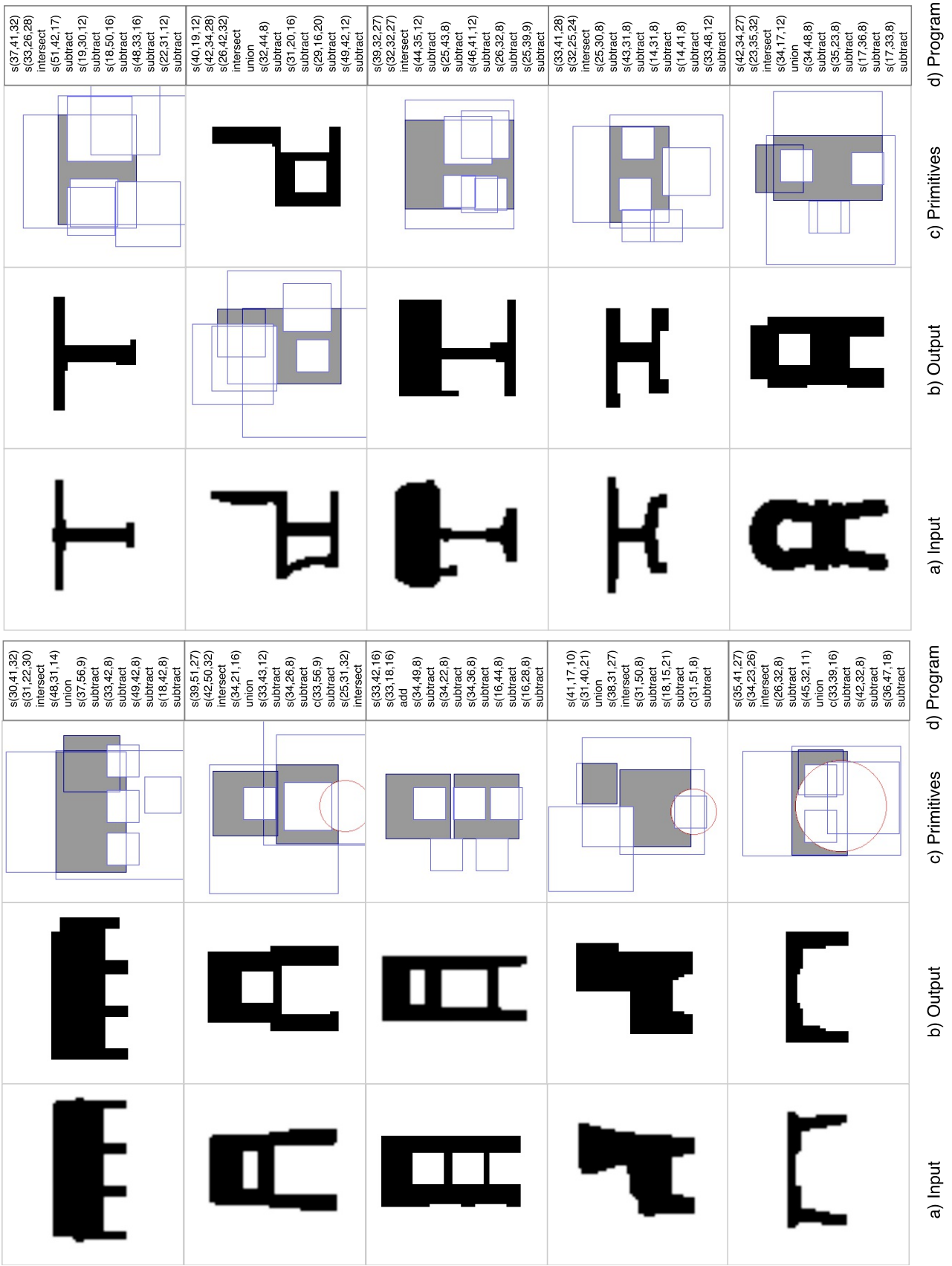

Figure 6. **Performance of our full model on 2D CAD images**. a) Input image, b) output from our full model, c) Outlines of primitives present in the generated program, triangles are in green, squares are in blue and circles are in red d) Predicted program. s, c and t are shape primitives that represents $square$, $circle$ and $triangle$ respectively, and $union$, $intersect$ and $subtract$ are boolean operations.