2D Shape Synthesis

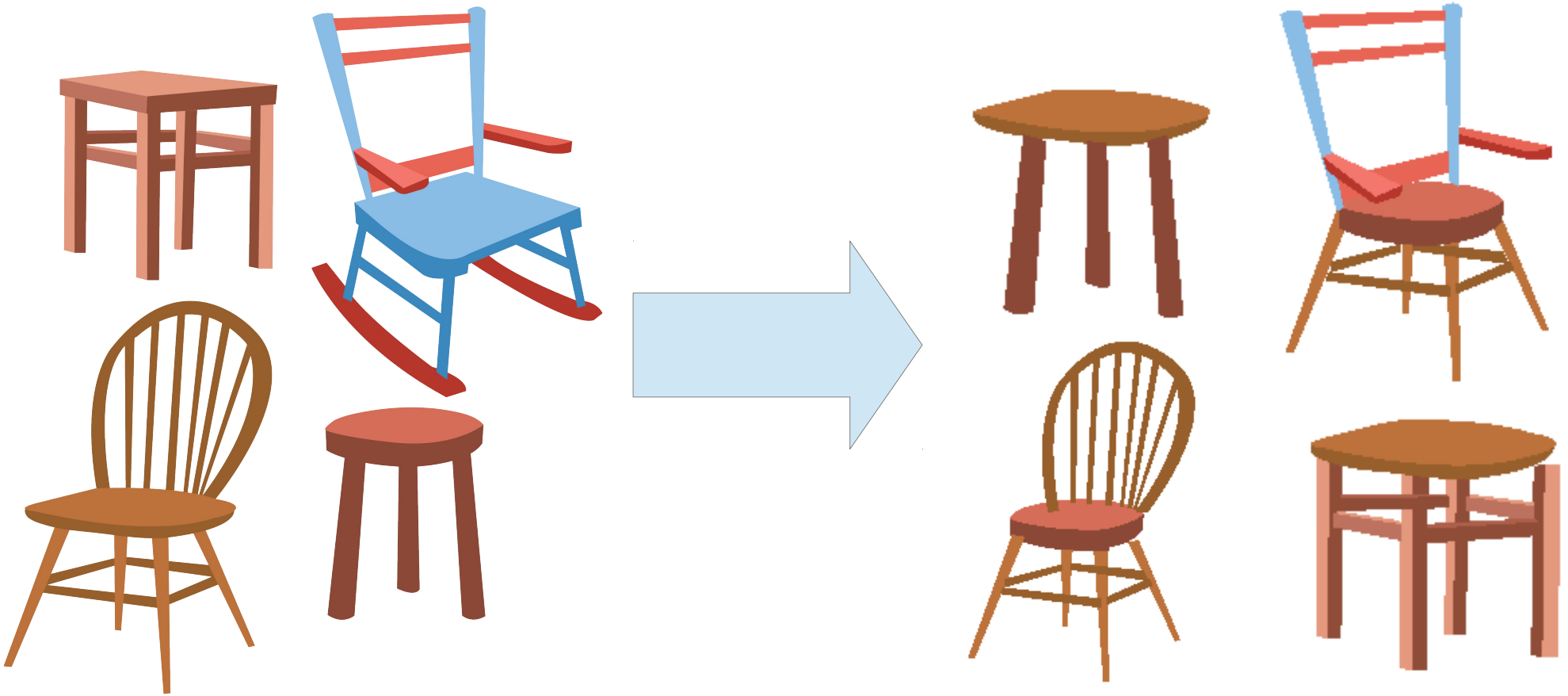
Shape Synthesis

Generating new figures/patterns/layouts using some criteria (e.g. similarity to base inputs).

- Generating new 3D figures
- Generating new surfaces
- Generating new doodles

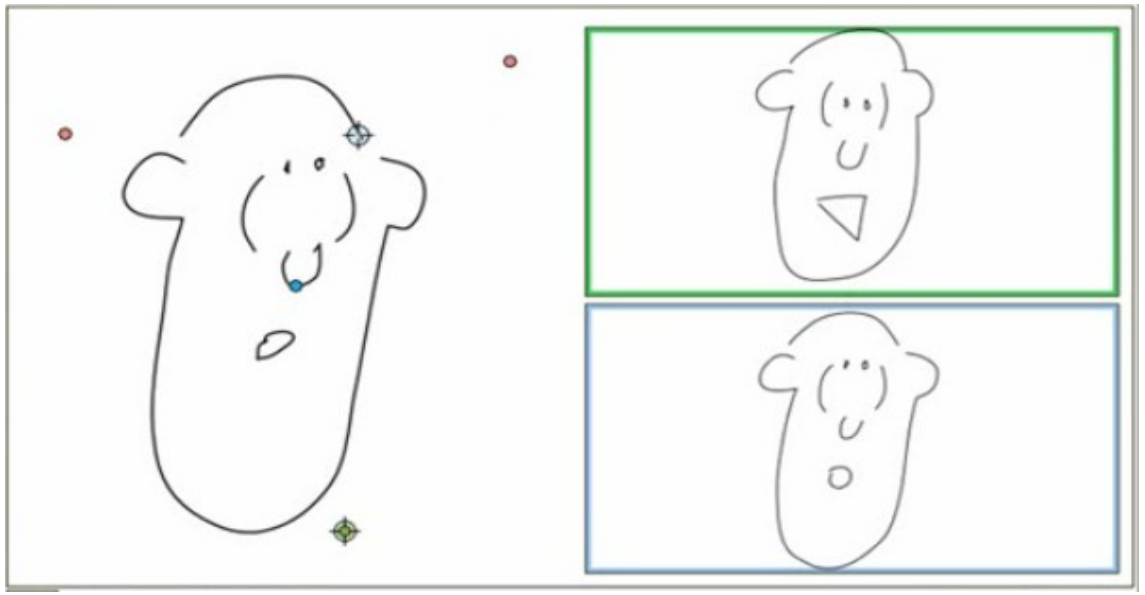
The problem

Given a set of 2D figures that have a similar structure, generate a new set of unique figures that preserve this structure. We consider both doodles and images.



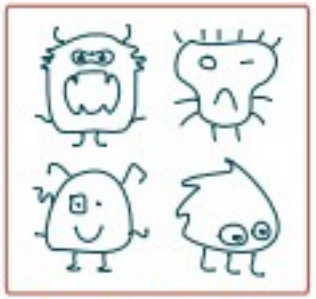
Latent doodle space '06

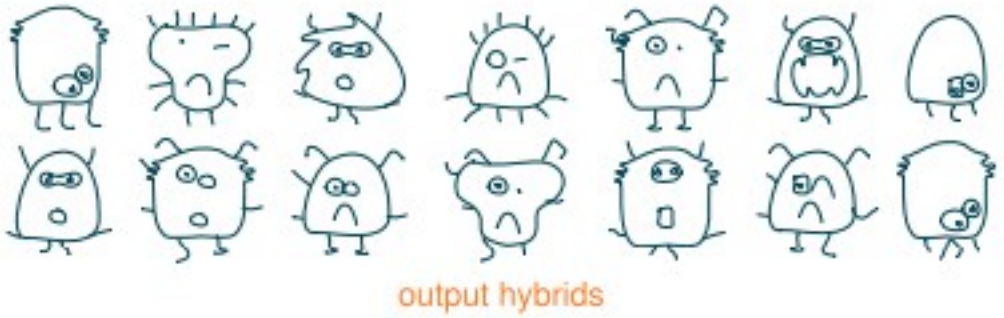
- Tool for creating new doodles.
- Creates new doodles based on interpolation of parts.
- Getting mismatches.



Hurtut doodle hybrids '13

- Synthesizing doodles
- Uses “inclusion tree” to describe inputs, as a result requires parts to be enclosed in each other.



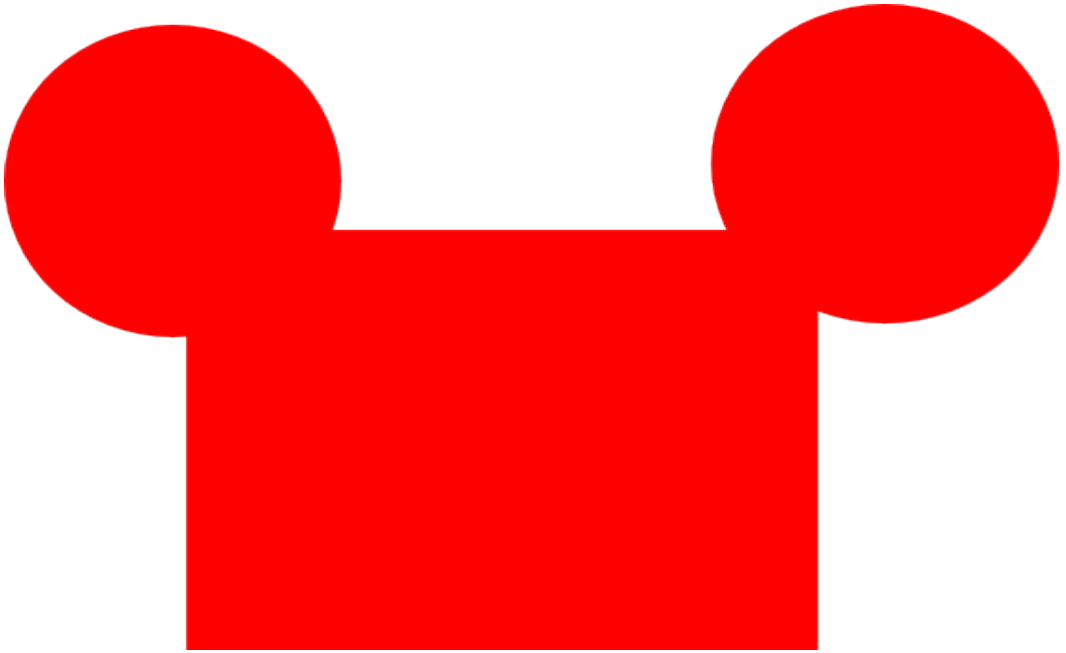


Our approach

Small tech details

- Inputs to the generator are given as SVG files.

```
<svg>
 <g id="layer1" fill="#f00">
  <rect id="rect2988" height="245.71" width="337.14" y="432.36" x="234.29"/>
  <path id="path2985" d="m700 393.79a92.857 92.857 0 1 1 -185.71 0 92.857 92.857 0 1 1 185.71 0z"/>
  <path id="path2993" d="m317.14 403.79a90 91.429 0 1 1 -180 0 90 91.429 0 1 1 180 0z"/>
 </g>
</svg>
```



At the very beginning, we parse each of the files to get information about each element. For each of the elements we also calculate some properties that we will need later. All components are polygonized.

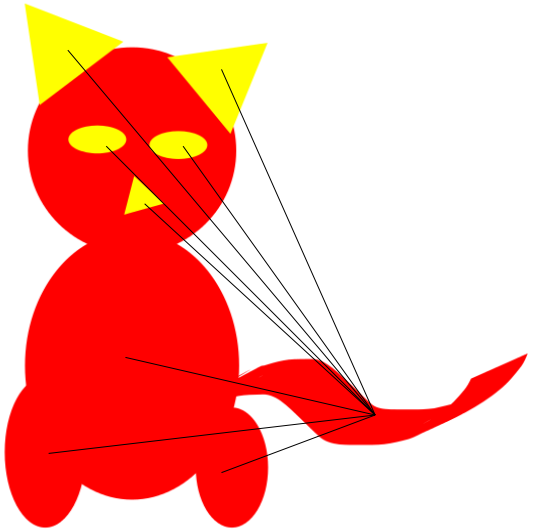
Splitting the problem

1) Matching two figures with each other – we want to get a correspondence between the parts so that we know what can be replaced with what.

2) Synthesis – using matchings we got generate new figures.

First idea

- Evaluate match quality between every pair of elements in two figures.
- One contributor to quality is similarity of their vectors to other objects.



- Afterwards select a resulting matching either greedily, or by Hungarian algorithm. We stop when the match quality gets low.

Turned out to give a lot of mismatches :(

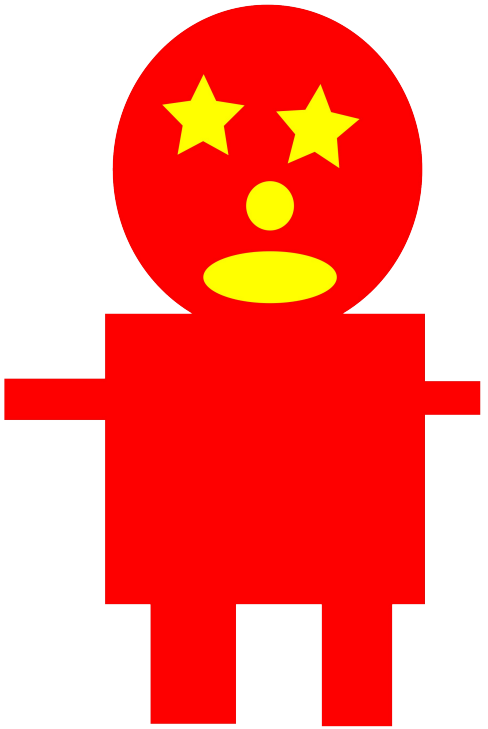
Making shape structured

1) Node – simplest object (polygon)

2) Nest – collection of Nodes, all of which are enclosed inside a root Node.

Making shape structured

3) Group – group of connected Nests.



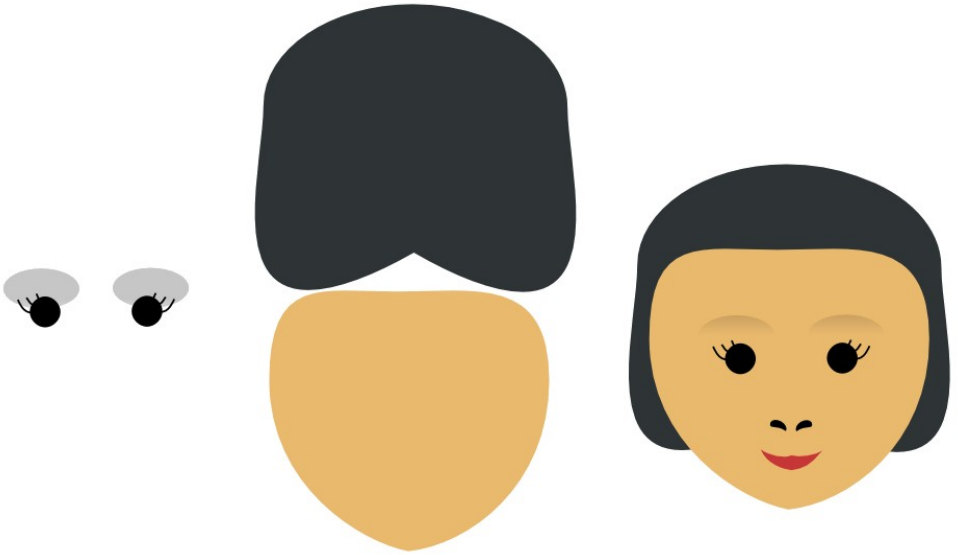
4) Graph – entire content. It can have several Groups in it.

Building the structure

When we initially parse the SVG, we created Nodes for all elements we find. After that we're partitioning them into nests/groups.

Special case

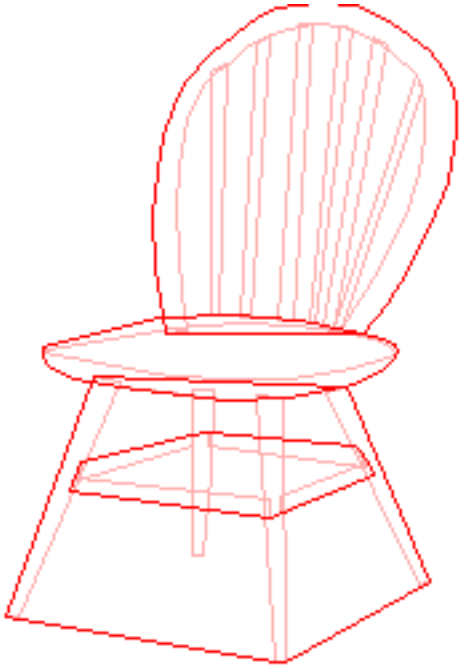
- We can have an element that is enclosed in two other elements:



- We decide this case by the visibility order – whichever of the two was displayed later gets the element enclosed in it.

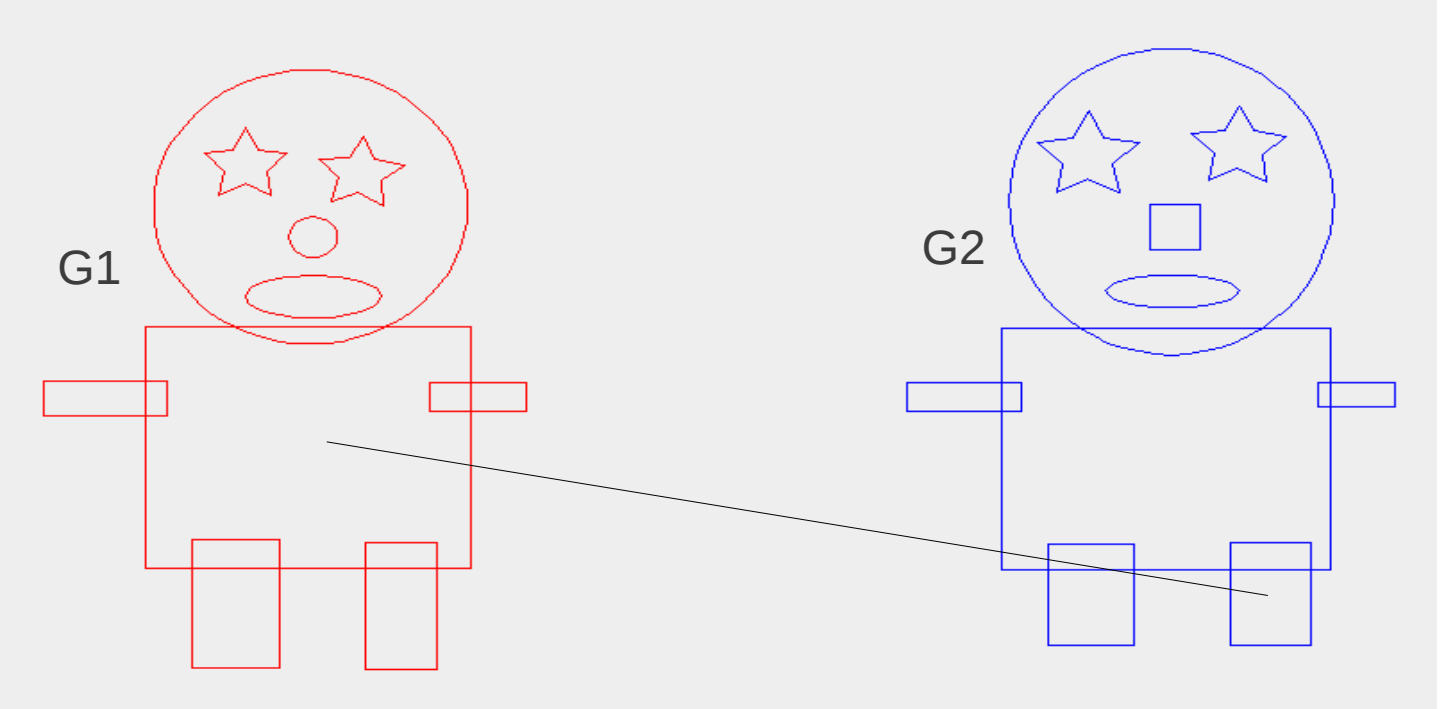
<g> tag

- Some elements can be grouped using a <g> tag.
- In this case, we unite these elements into one Node.
- Originally a convex hull of all united elements was used for purposes of intersection or matching. However, this caused some problems at synthesis step, so we consider intersection happening if one of sub-elements gives us intersection.



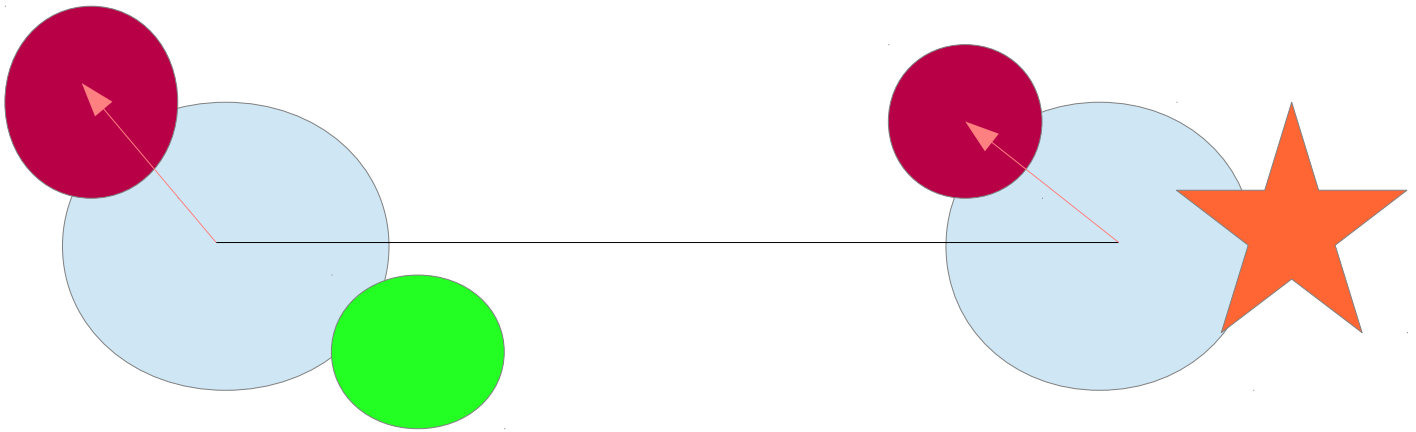
Matching groups

- We try all pairs of nests from G1 and G2 as our initial match.
- We try to expand each such match.



Matching groups

- For each matched pair we look at their neighbors, and attempt to match them. To do this, we evaluate a match quality. If match quality is larger than some threshold, we add a new matched pair, add will repeat the same procedure to it recursively.



Match quality

- Match quality for any two nodes is a weighted energy function of several parameters that we are using to measure similarity.

$$M = \frac{k_1 w_{angle} + k_2 w_{distance} + k_3 w_{area} + k_4 w_{context}}{k_1 + k_2 + k_3 + k_4}$$

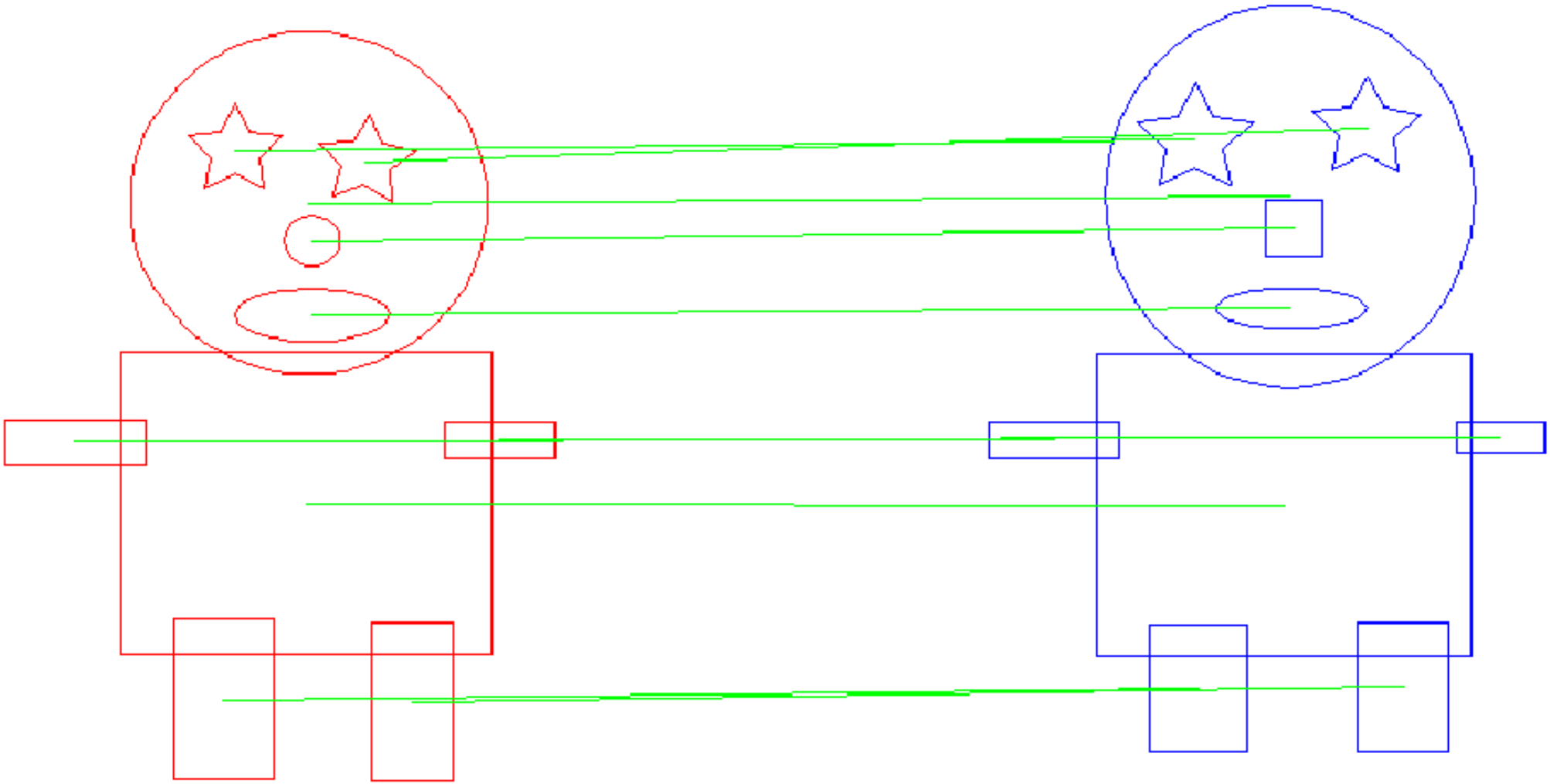
Matching nests

- Happens if root elements were matched.
- Look at each pair <f1, f2> of elements (f1 from the first figure, f2 from the second one) and evaluate its match.
- Energy function uses the same parameters as before. Angle of f is defined as angle from centroid of f's root to centroid of f.

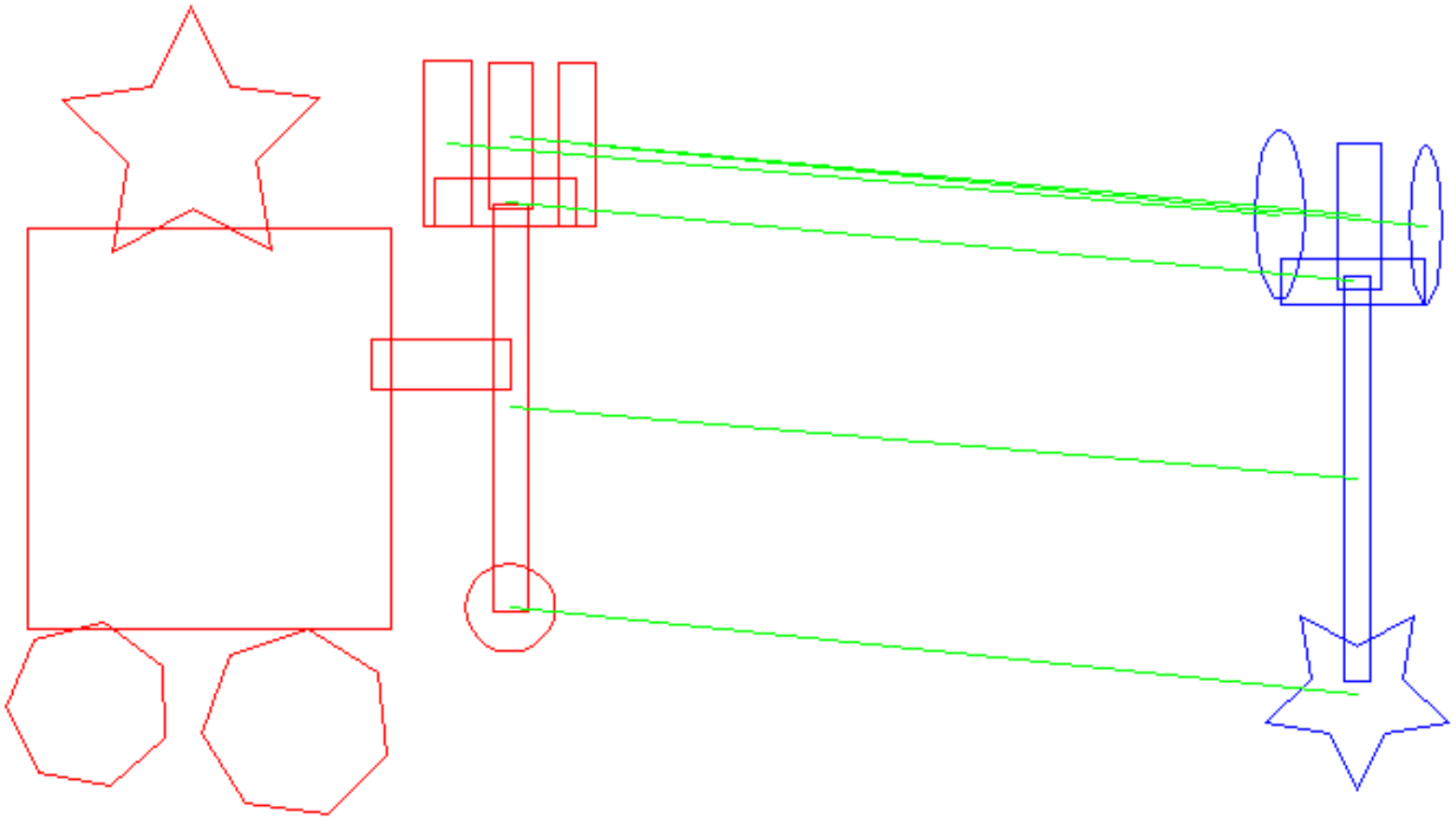
Matching graphs

- Currently we consider different groups to be unrelated. Hence, we just look at pairs of matched groups and process them greedily.
- Alternatively, we can also take positioning of groups into account.

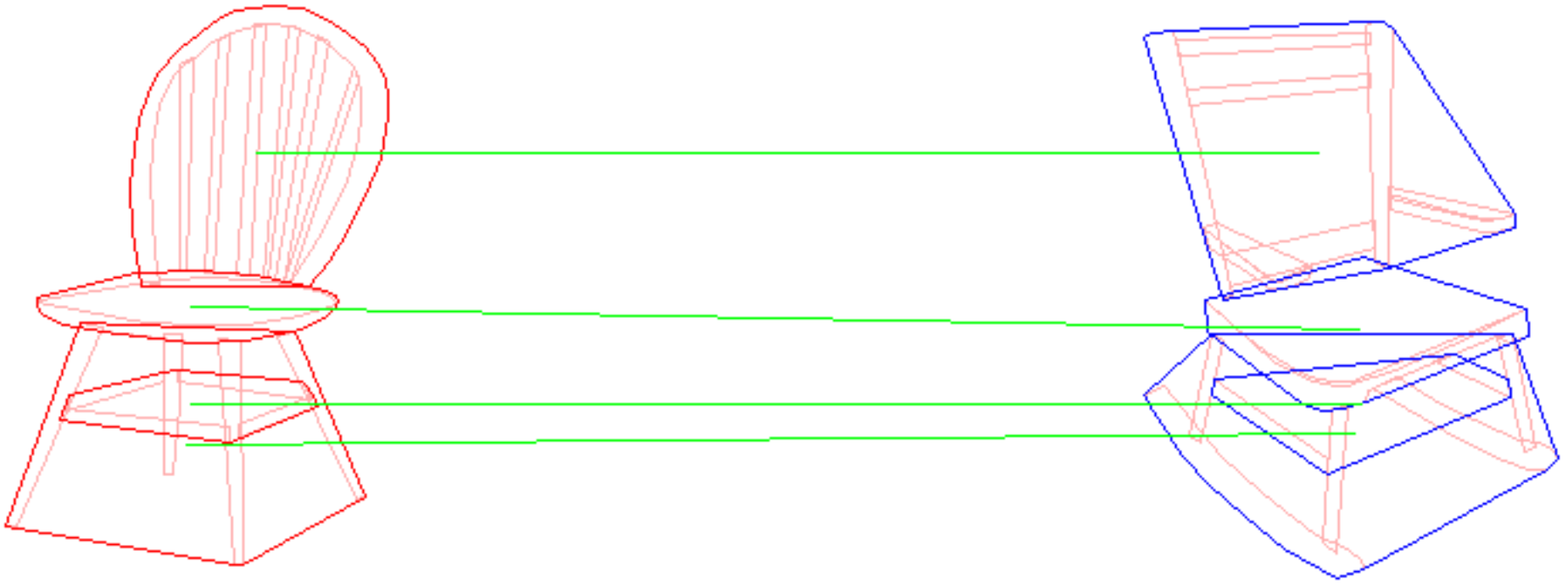
Resulting match



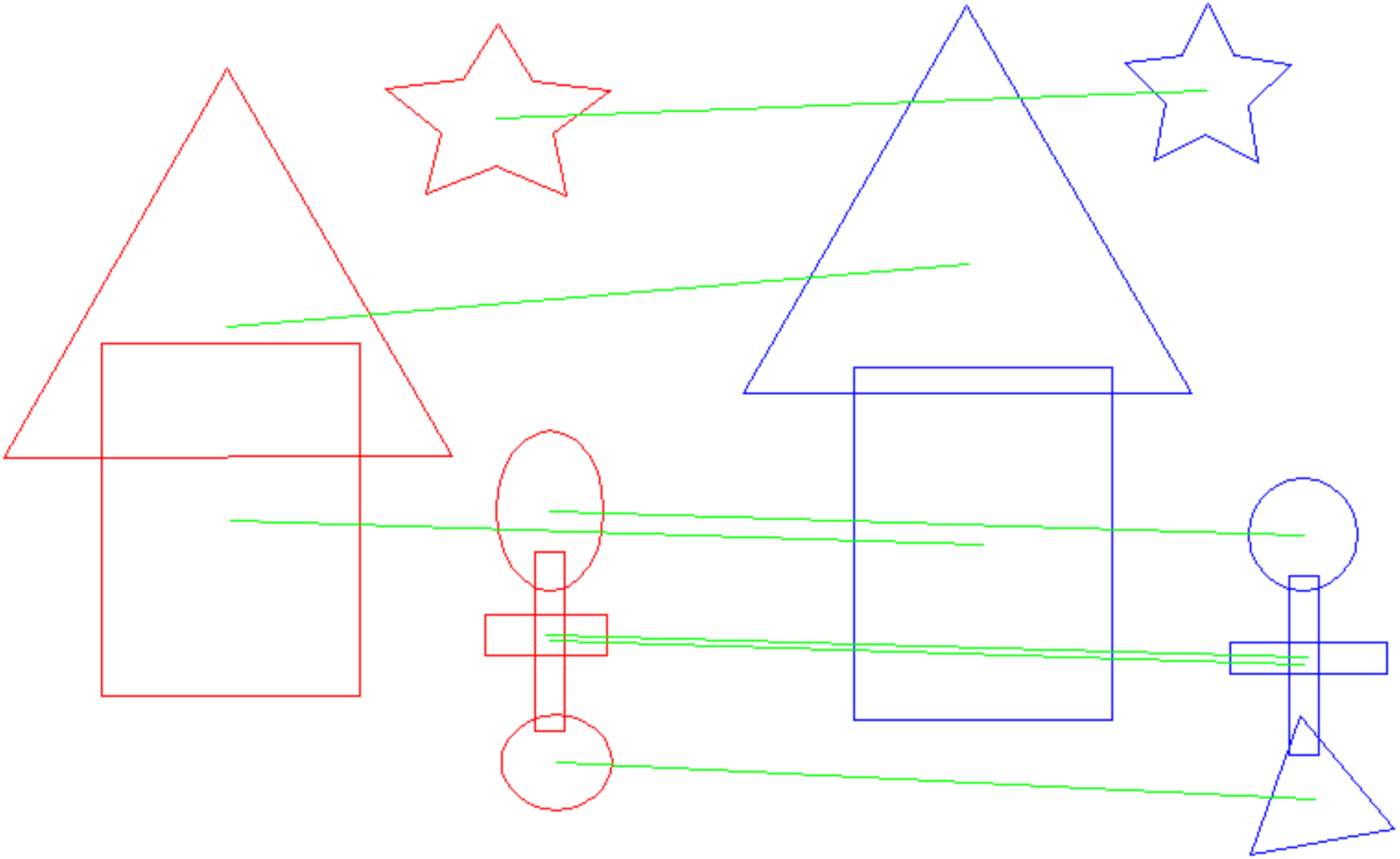
Resulting match



Resulting match

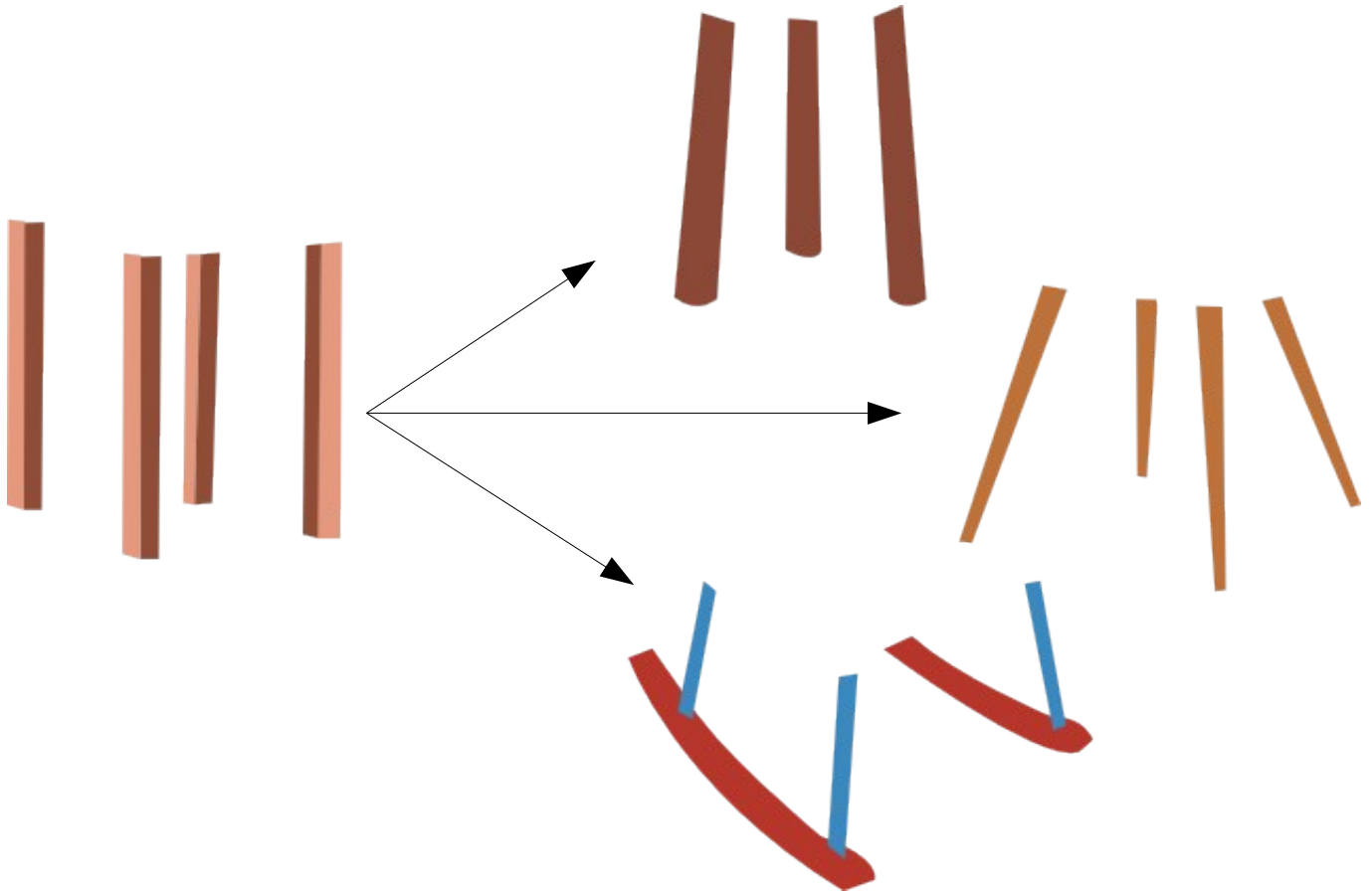


Resulting match



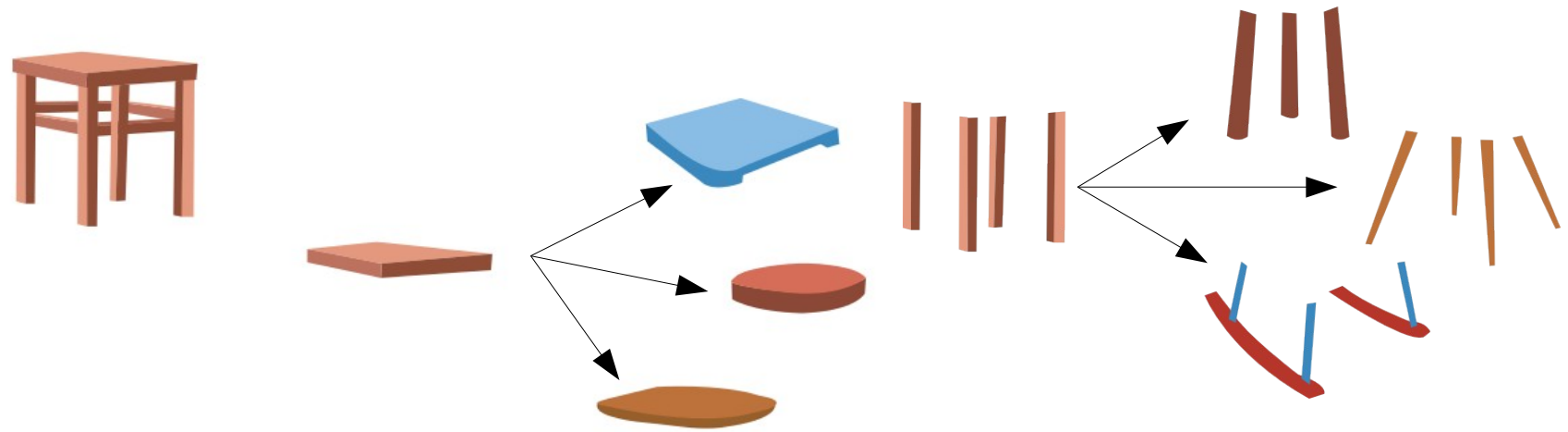
Generation

- After we're done with the matching, we compile a list of possible replacements for each part of the figure.



Generation

- For each generated figure, we pick one of the samples as the “base”, and for each part of the “base” pick a random replacement (or do no replacement at all).



Generation

- When we want to replace part A with some part B, we place part B so that its centroid lies where centroid of part A used to be.
- However, resulting shape can become “damaged” as a result.
- It is important, that figures that were intersecting, are still intersecting; and the figures that were nested, are still nested.

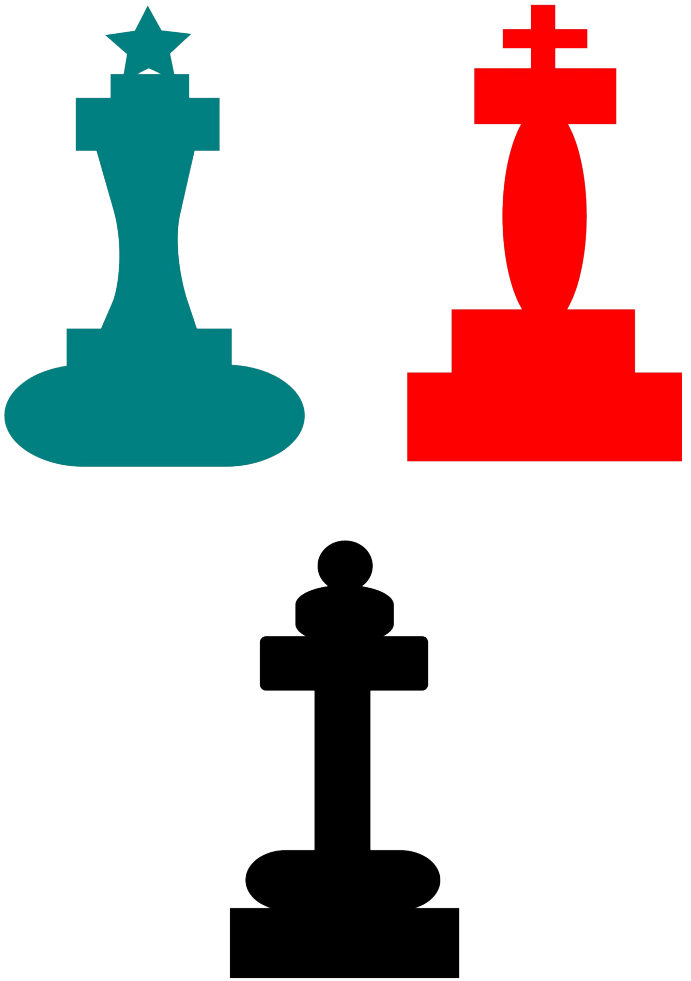
Loosing connectivity

- Head became disconnected from the body.

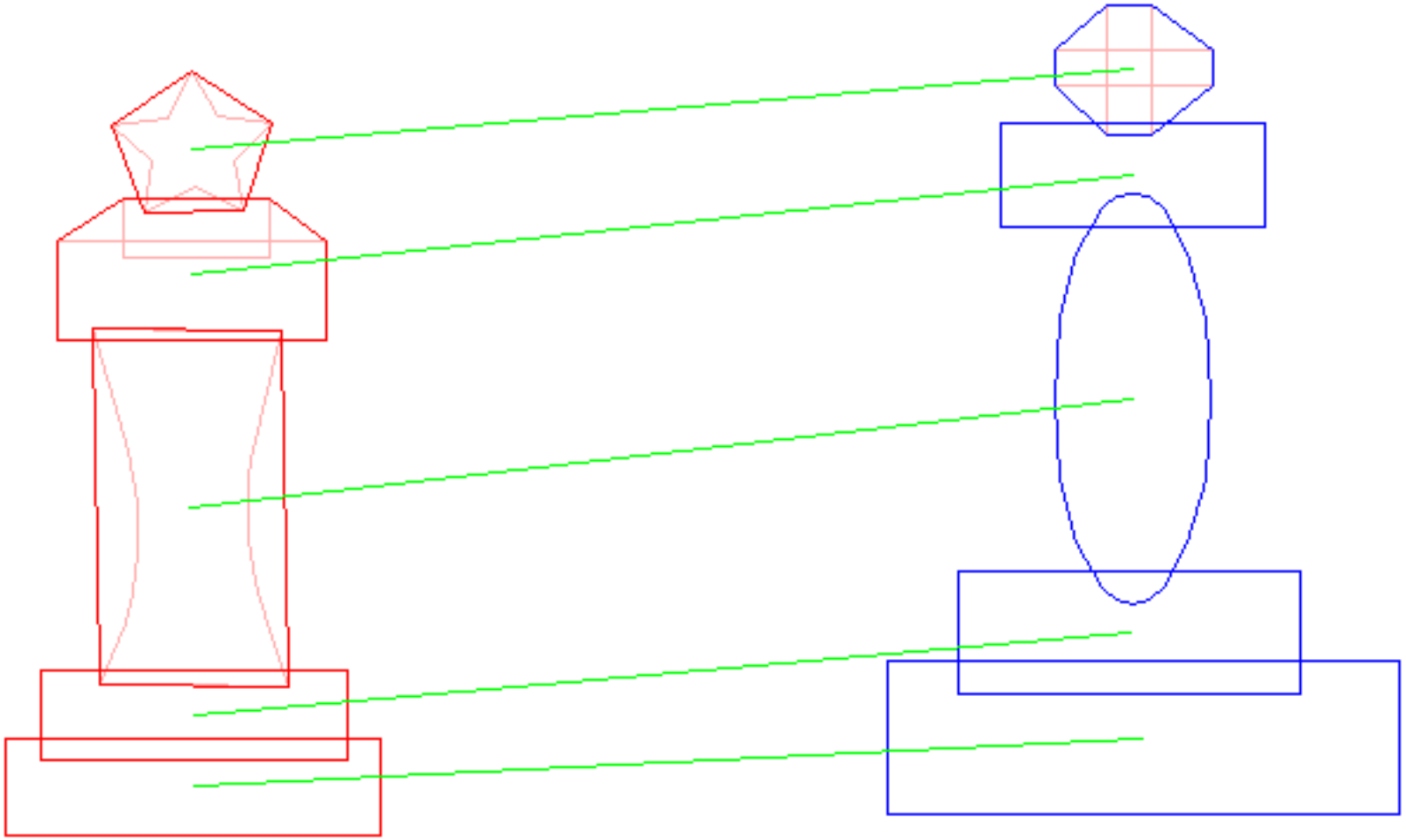
For each part check to see if it still intersects it neighbors. If not, move the neighbor closer so that the intersection happens.

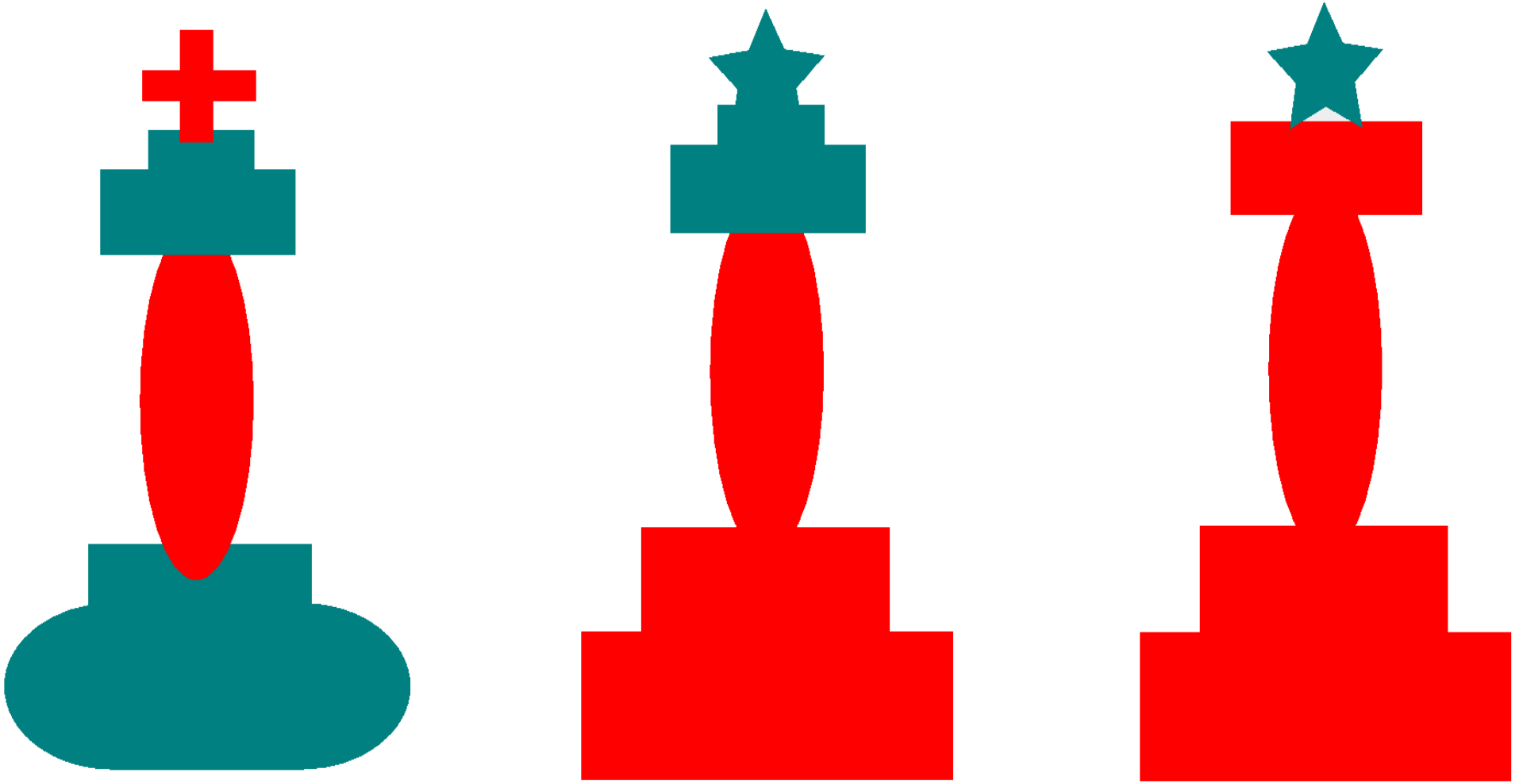
Some results

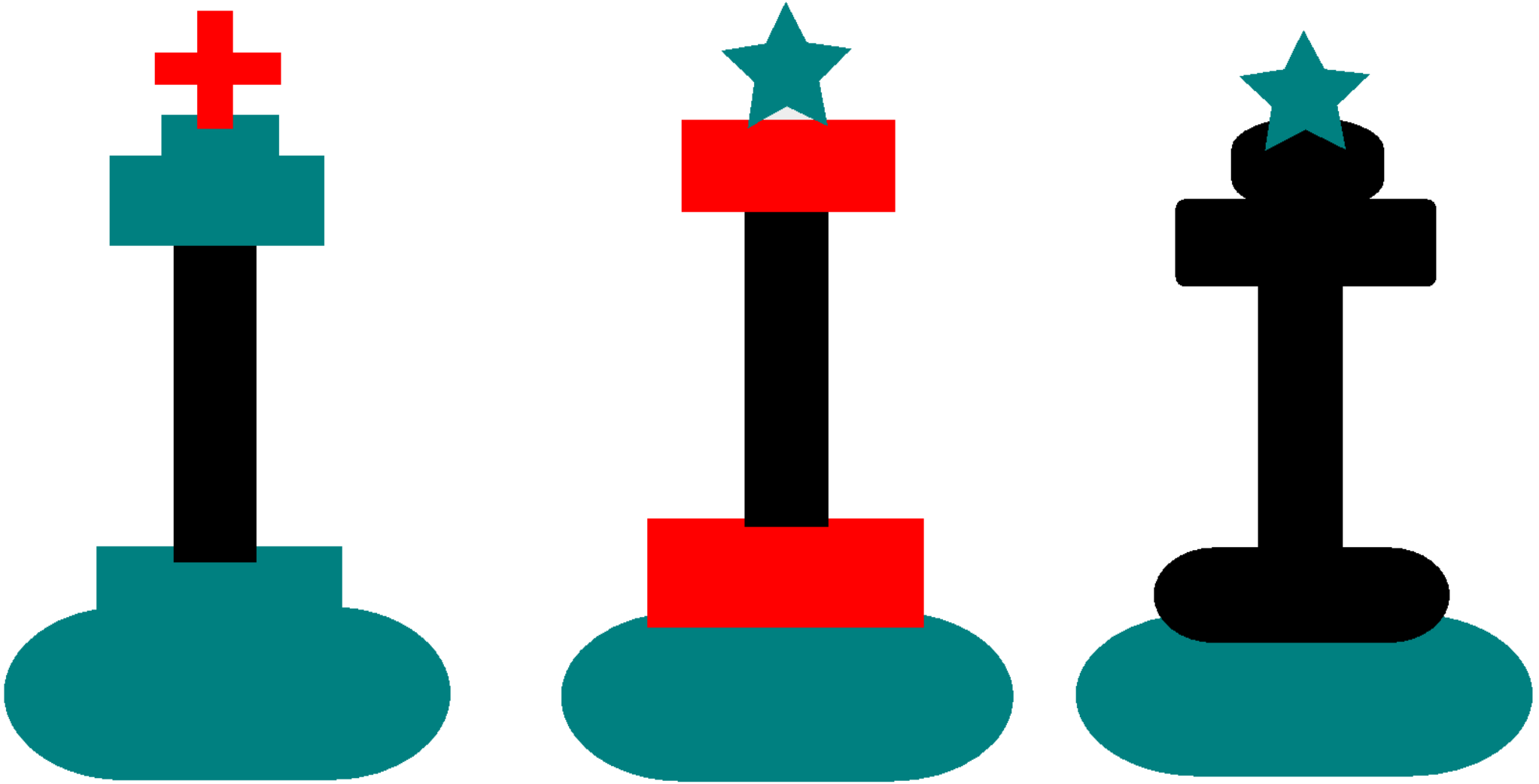
Chess doodles



Matching



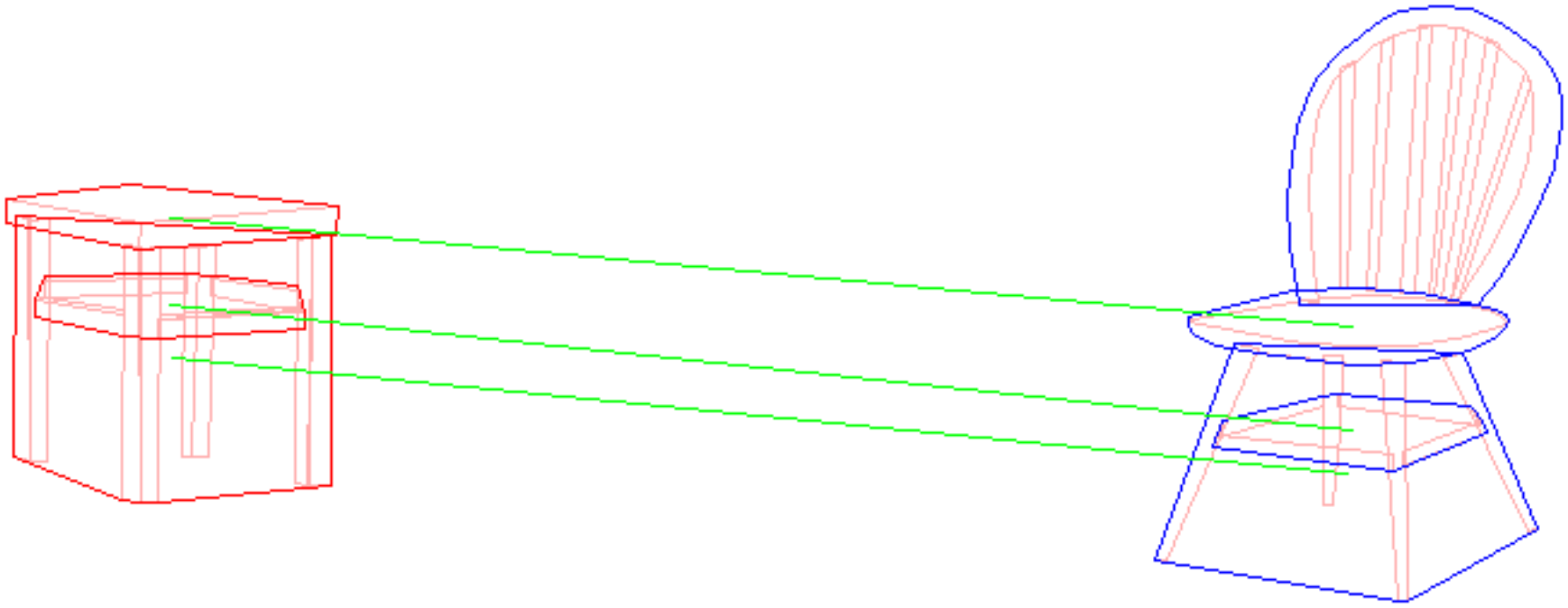




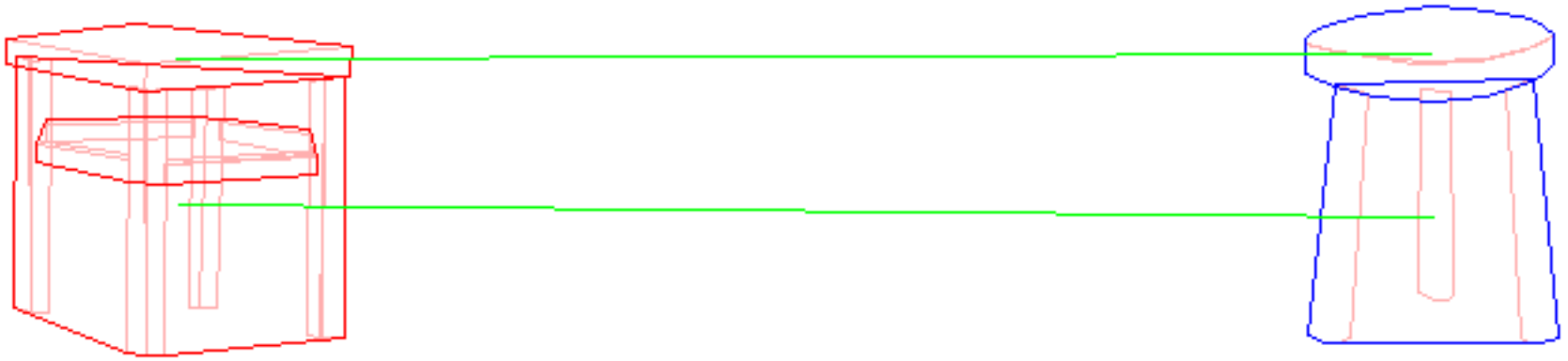
Chairs data set

Input set

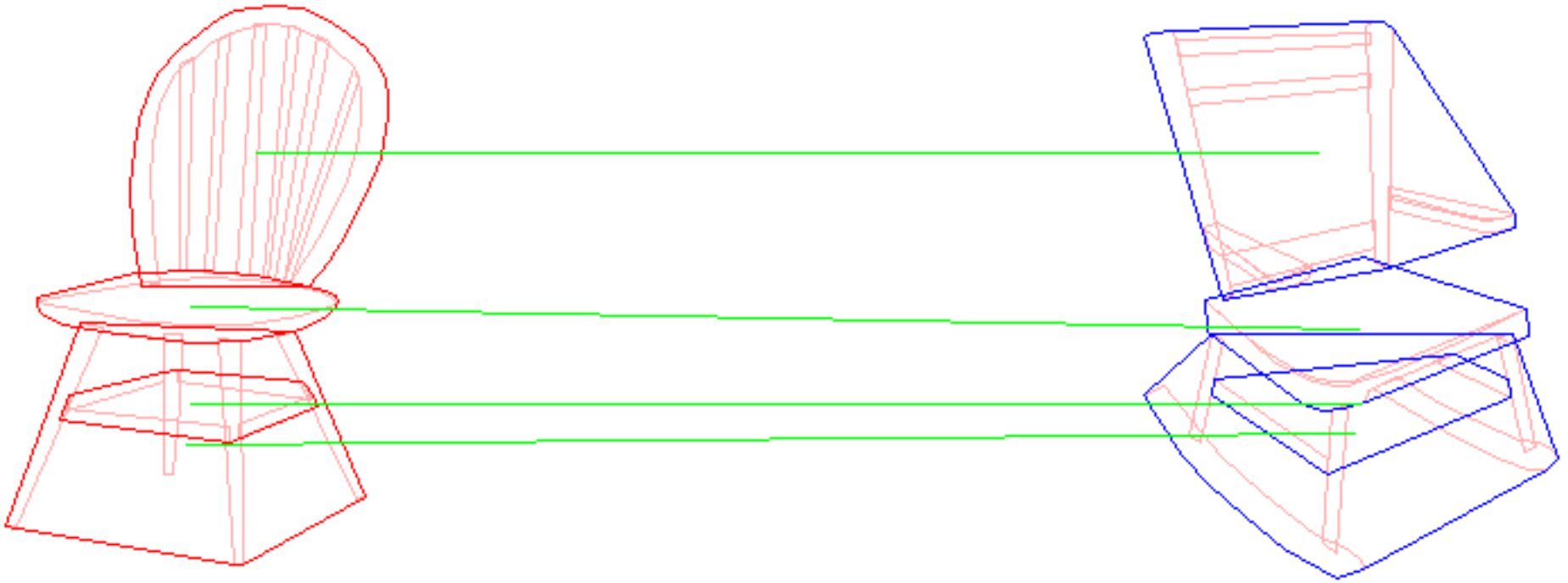
Matching



Matching



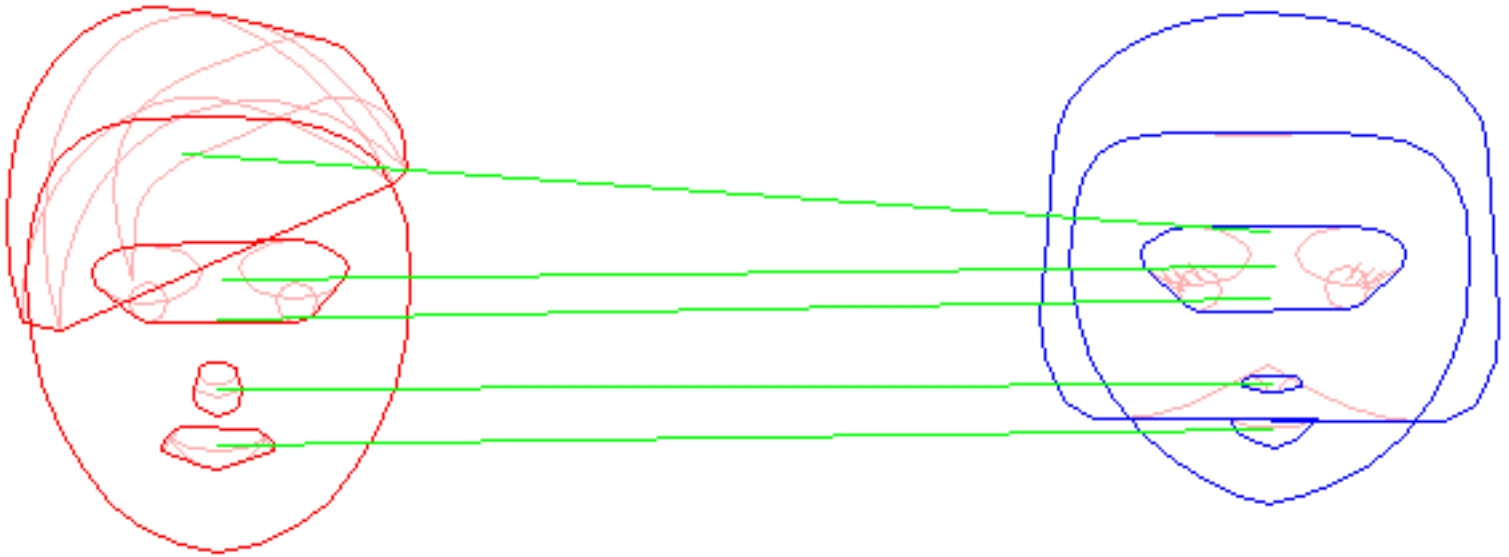
Matching

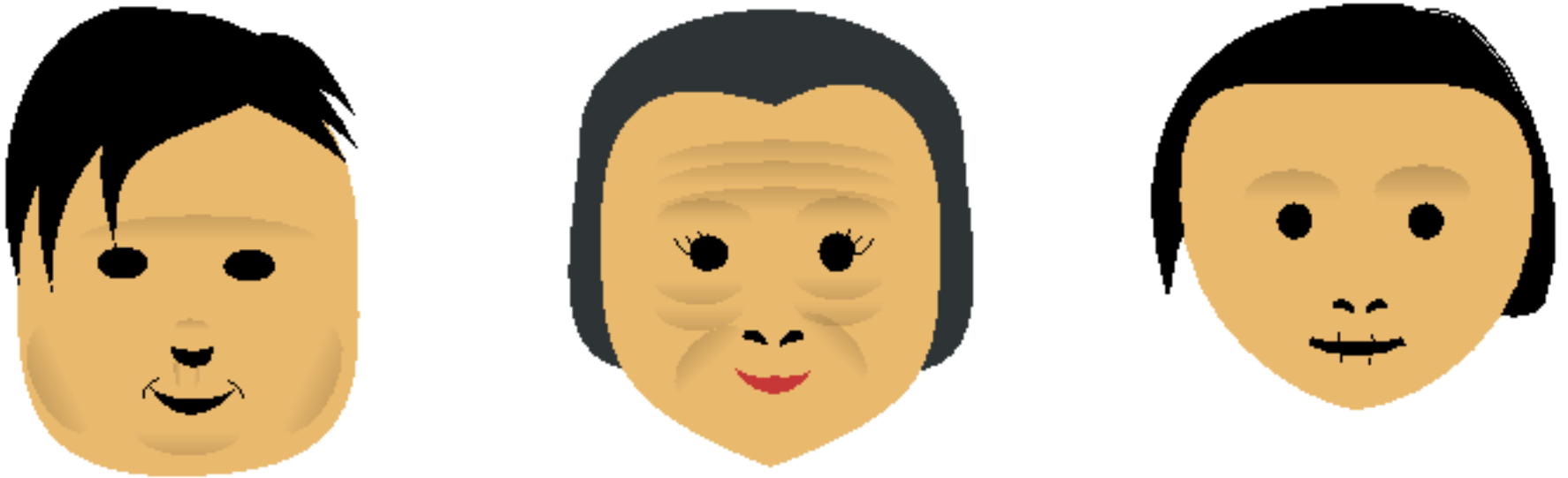


Generated chairs

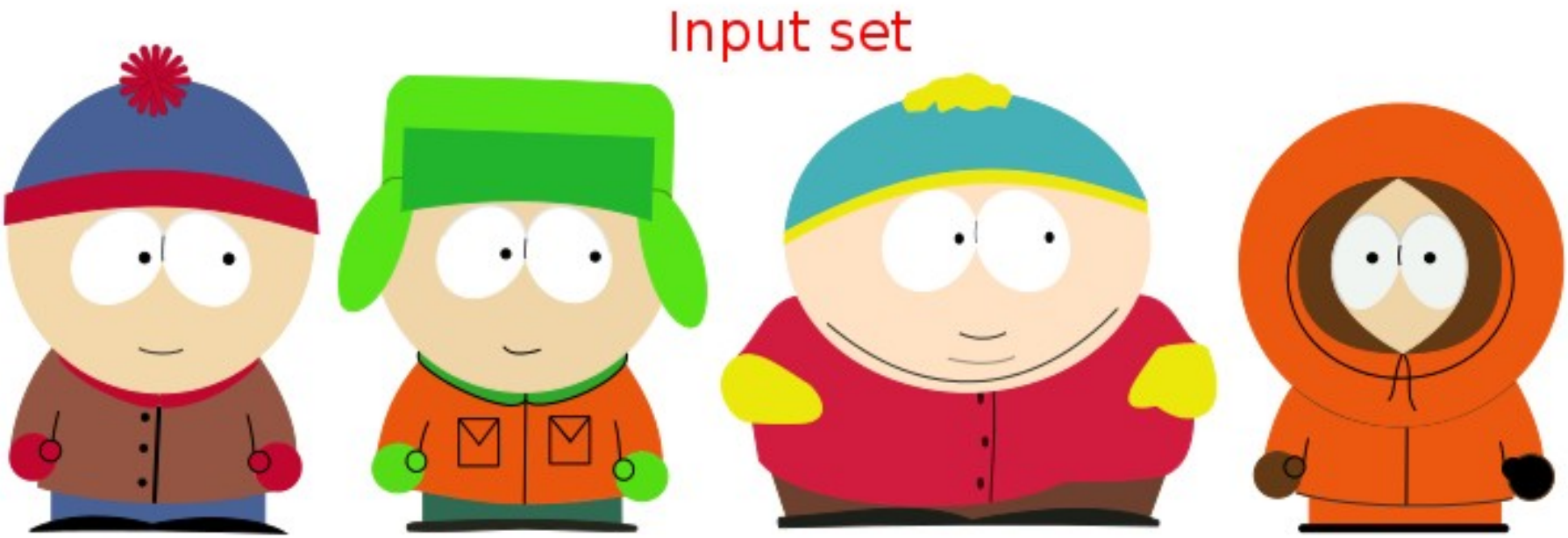
Faces data set

Matching

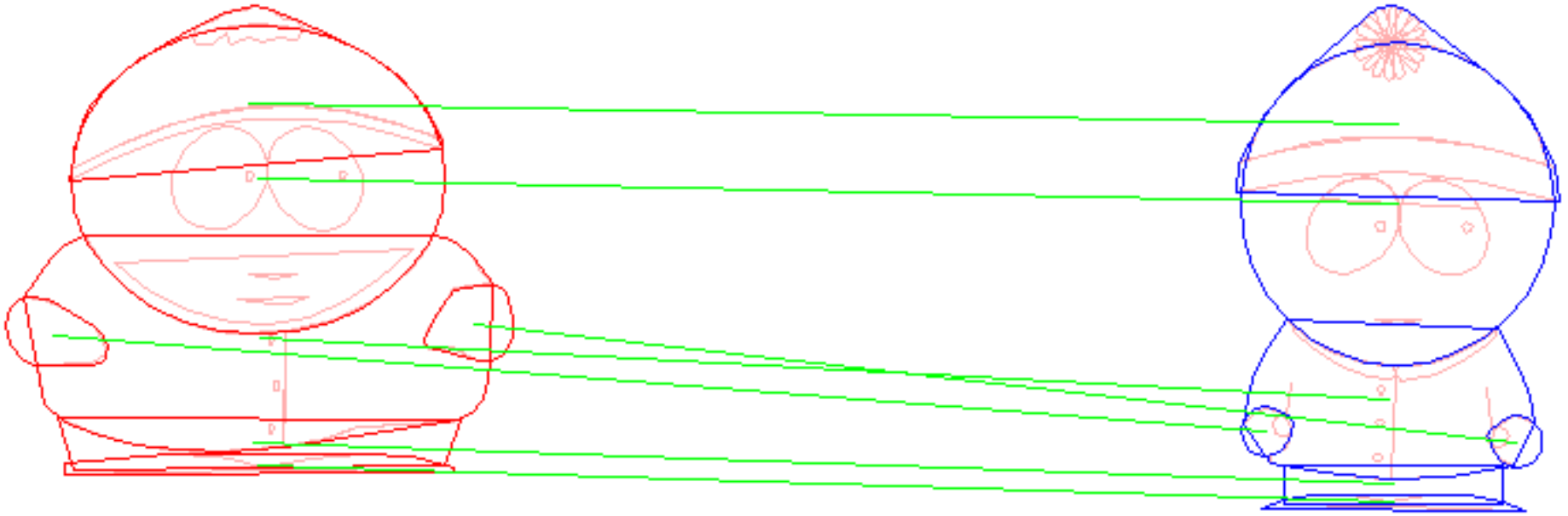




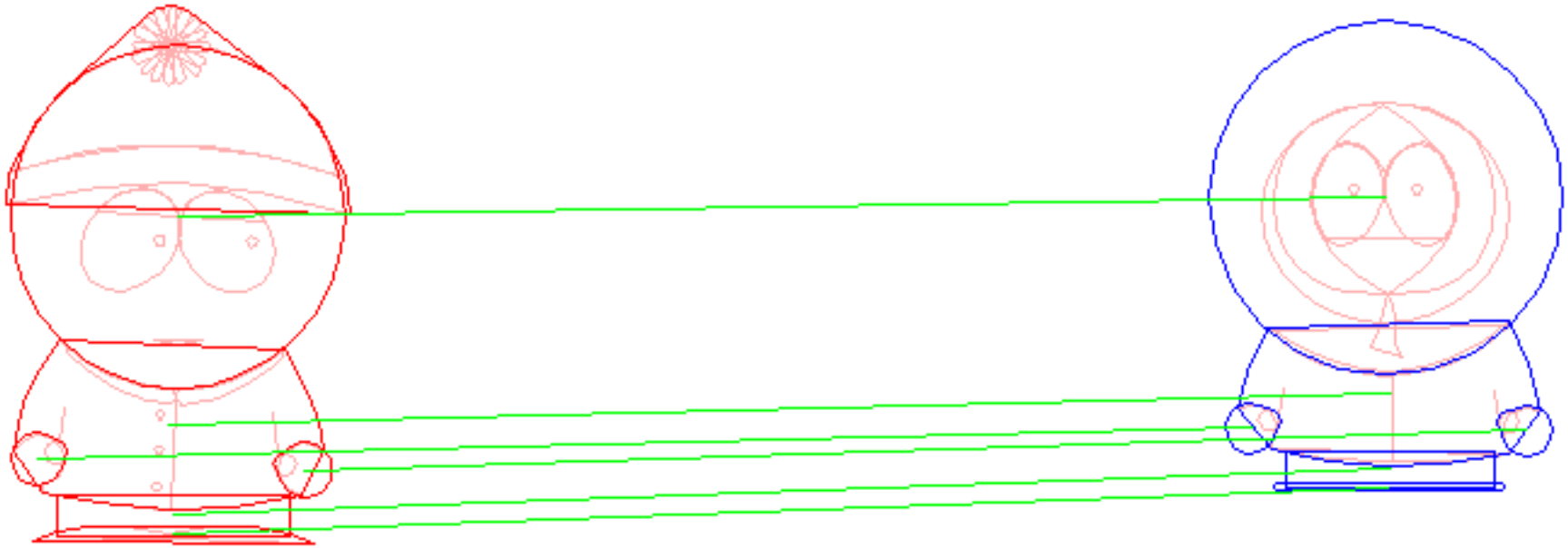
South Park data set



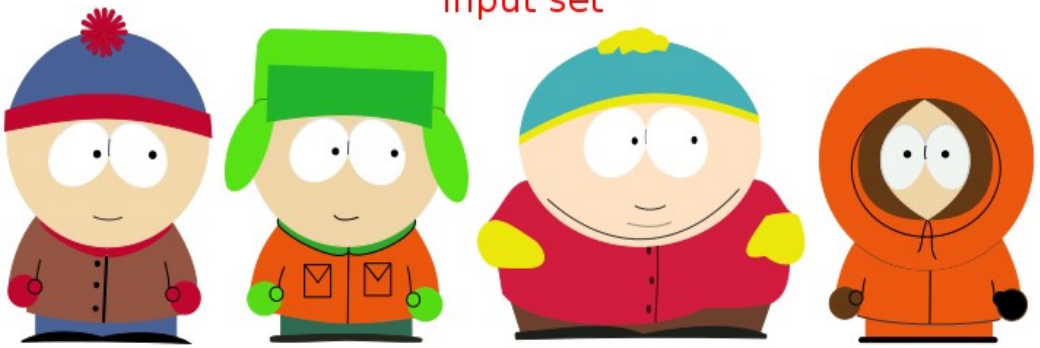
Matching

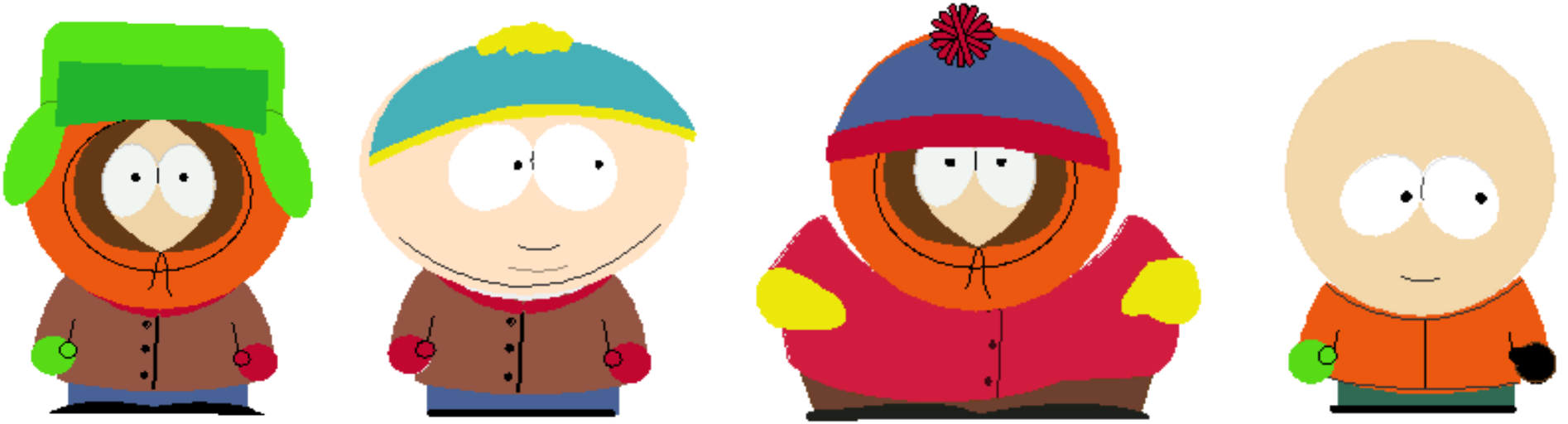


Matching

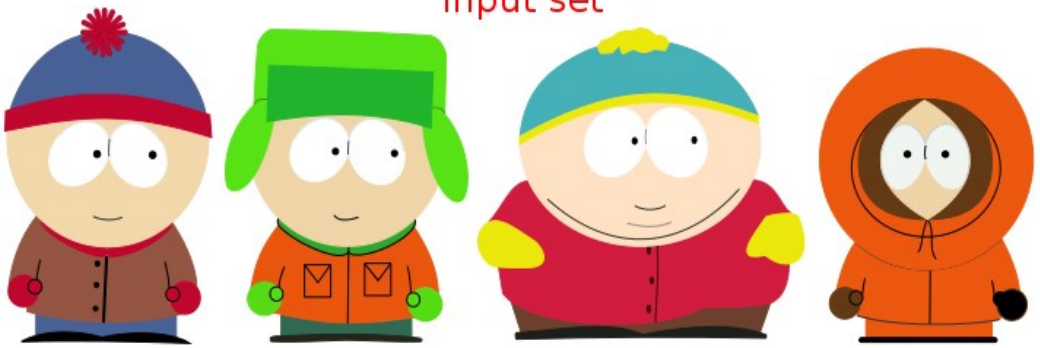


Input set





Input set



Limitations and extensions

- Understanding perspective
- Dealing with figures that have inappropriate shape for replacement.

- Remove the need to restructure SVG's.