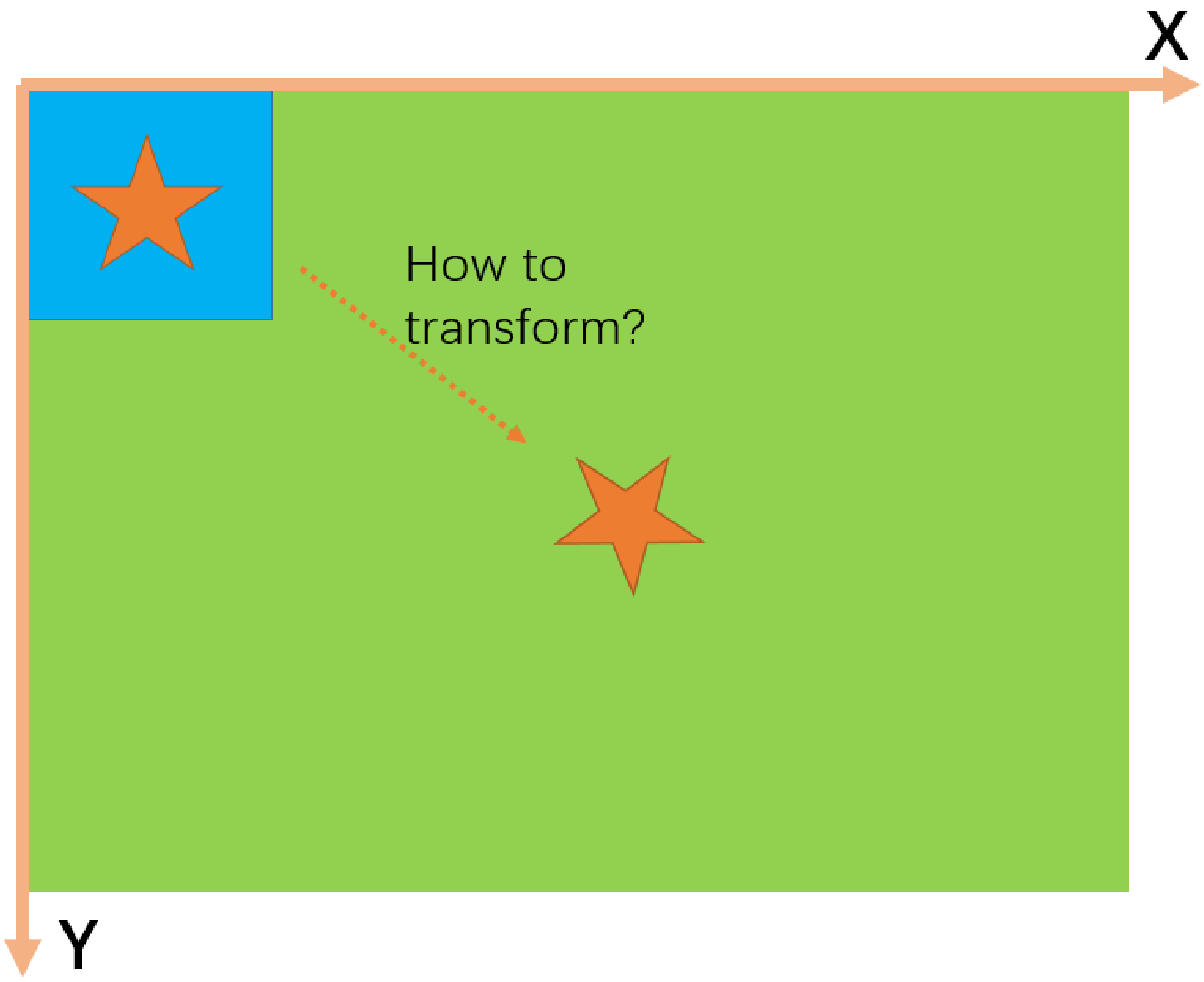
Transforms in shape-based matching

Transform is all we need

Technically speaking, the purpose of shape based matching is simple: find a 3x3 matrix which transforms training image and makes it lie exactly on the similar part of the test image. To make the algorithm better understood, this note explains transforms happened during the train and test phase.

The global frame should be determined first, here we adopt common image frame. At the beginning, the template & test image are aligned to the same top left origin. All mentioned transforms below are referred to the global frame.



Transforms during training

scale and rotate around template image center

$$
\begin{gathered}
T_{init} = \begin{bmatrix} a & b & (1-a)*cx - b*cy \\ -b & a & b*cy - (1-a)*cx \\ 0 & 0 & 1 \end{bmatrix} \\
a = scale * cos(angle) \\
b = scale * sin(angle) \\
cx = templ\_img.cols/2 \\
cy = templ\_img.rows/2
\end{gathered}
$$

The translation part looks strange, because our origin is top left corner. You can check that [cx cy 1] stay unchanged after transform.

crop black margin (translation)

Our template only holds edge points in training images. In test phase, templates' right & bottom borders are not allowed to excel test image's right & bottom borders, otherwise we may visit pixels out of image. Cropping black margin can make templates slide righter and lower.

$$
\begin{gathered}
T_{crop} = \begin{bmatrix} 1 & 0 & -tl_x \\ 0 & 1 & -tl_y \\ 0 & 0 & 1 \end{bmatrix} \\
tl_x = templ[0].tl\_x \\
tl_y = templ[0].tl\_y
\end{gathered}
$$

Translation during testing

Obviously,

$$
T_{test} = \begin{bmatrix} 1 & 0 & match.x \\ 0 & 1 & match.y \\ 0 & 0 & 1 \end{bmatrix}
$$

Transform during ICP2D

Obviously again,

```
cuda_icp::RegistrationResult result = cuda_icp::ICP2D_Point2Plane_cpu(model_pcd, scene);
\\result.transformation_
```

Tracking training image center

It's easy to find angle changes during the whole process. However, imagining translation is not an easy thing due to scaling in the initial transform.

Luckily, training image center is the only point that not affected by scaling. So we can track the center and find out where it goes finally. Well, just multiply the center with all the transforms, we can get:

$$P_{center\_finally} = result.transformation\_ * \begin{bmatrix} cx - tl_x + match.x \\ cy - tl_y + match.y \\ 1 \end{bmatrix}$$

Tracking other points is exactly same as tracking center, though the formula may be not as clean as center_finally.