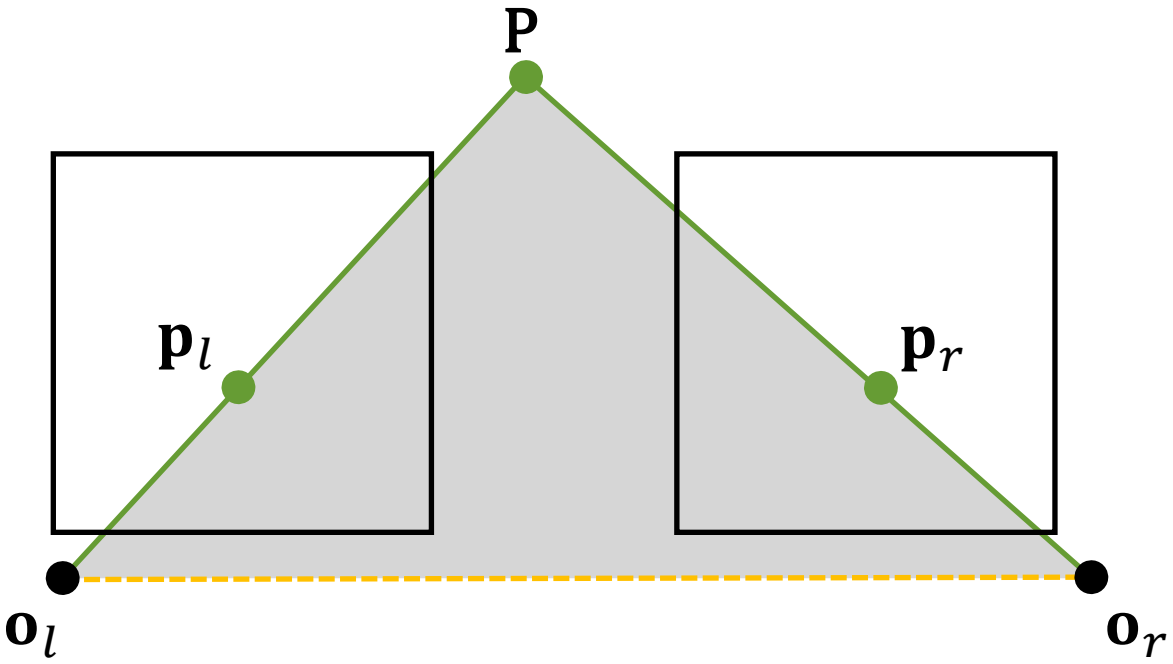
图像矫正



$$\mathbf{R} = \mathbf{I}_{3\times3}$$

$$\mathbf{T} = (B, 0, 0)^{\mathrm{T}}$$

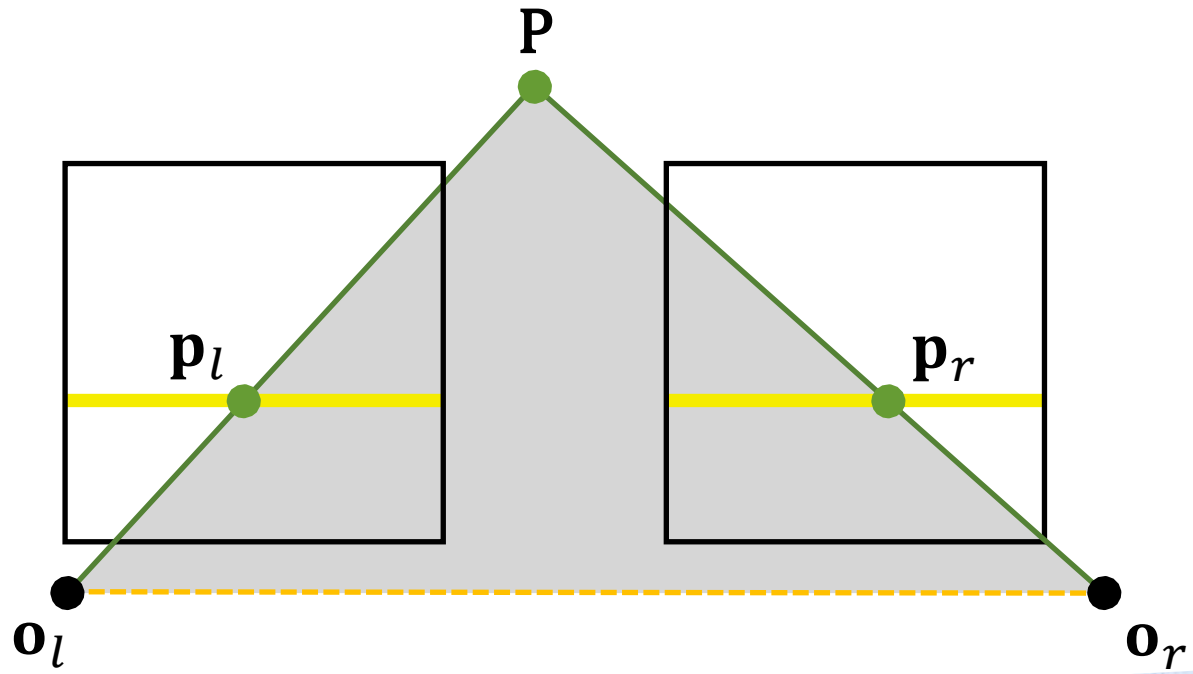
$$\mathbf{p}_r^{\mathrm{T}}\mathbf{E}\mathbf{p}_l = 0$$

$$(x_r, y_r, 1)[\mathbf{T}_\times](x_l, y_l, 1)^{\mathrm{T}} = 0$$

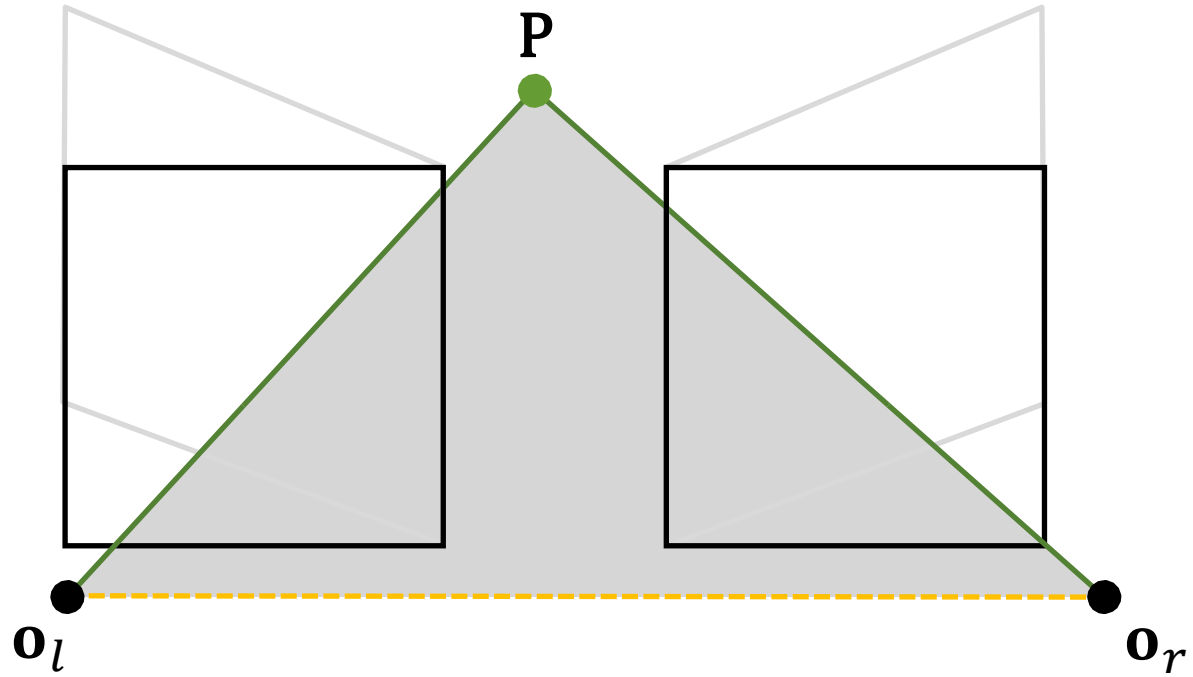
展开并化简

$$y_r = y_l$$

对应点位于相同一条水平直线上



平行的像平面可以简化立体匹配问题



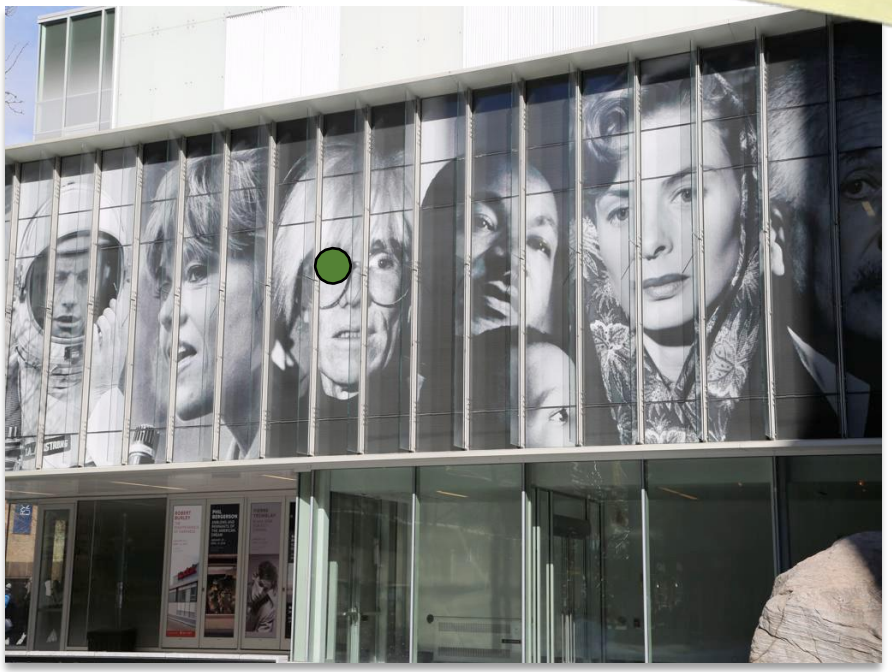
将图像重投影到与基线平行的公共平面上

对每幅图像应用单应变换

原始
图像对

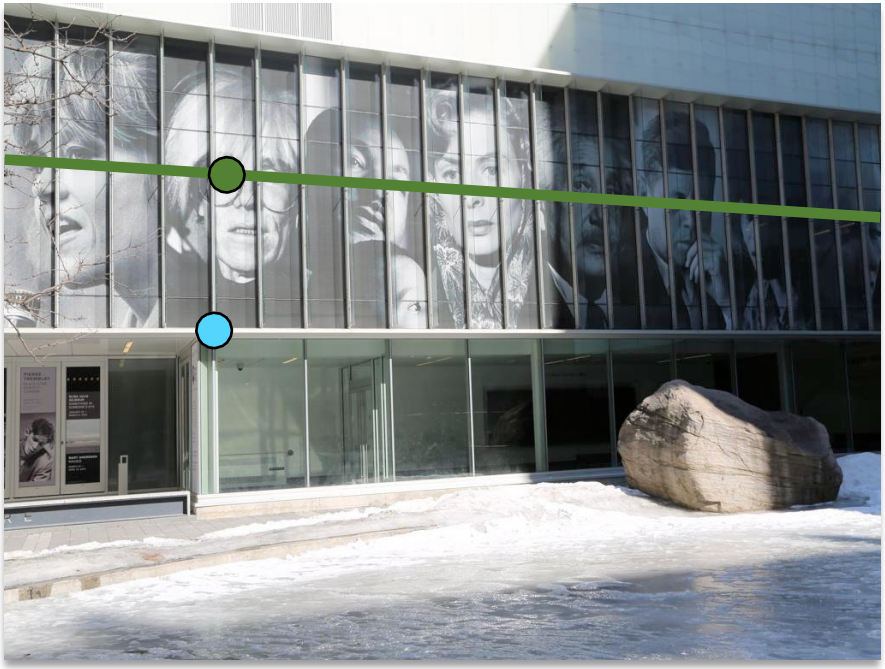
原始
图像对

原始
图像对

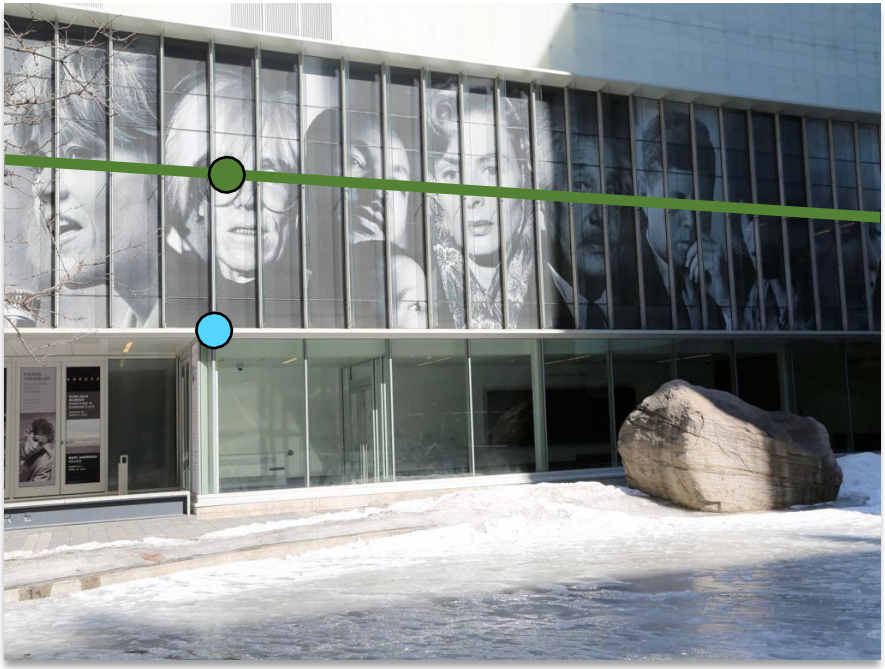


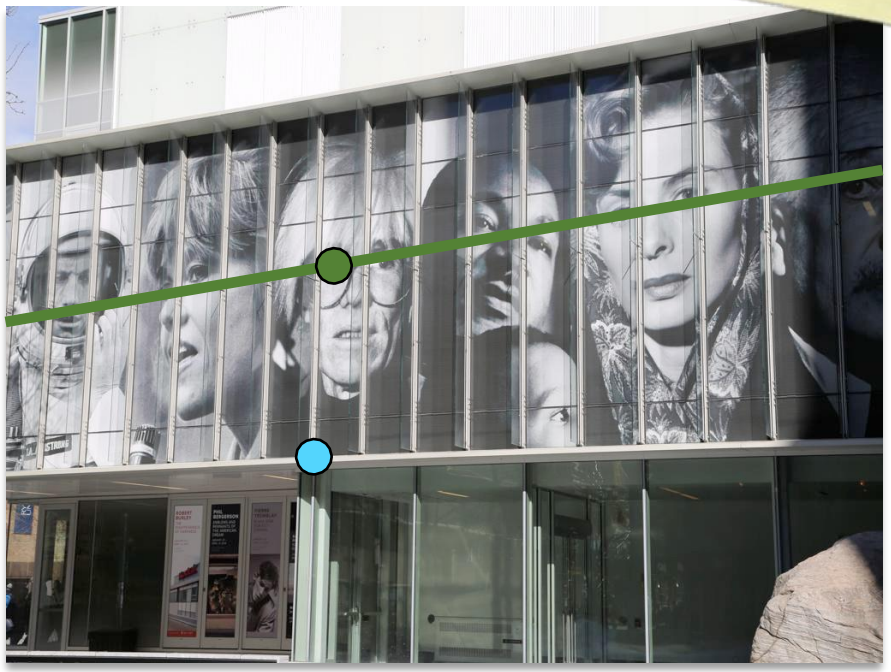
原始
图像对

原始
图像对

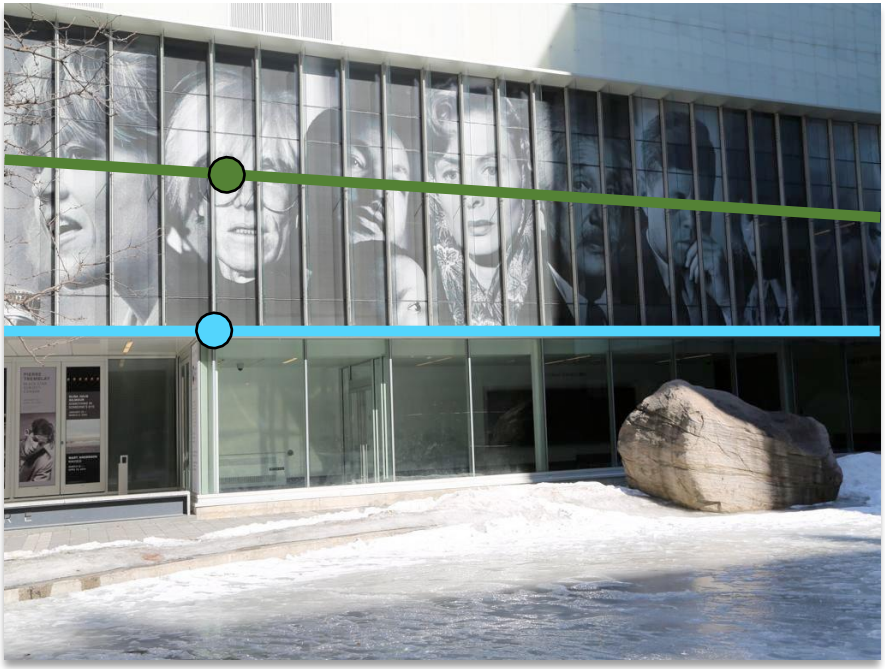


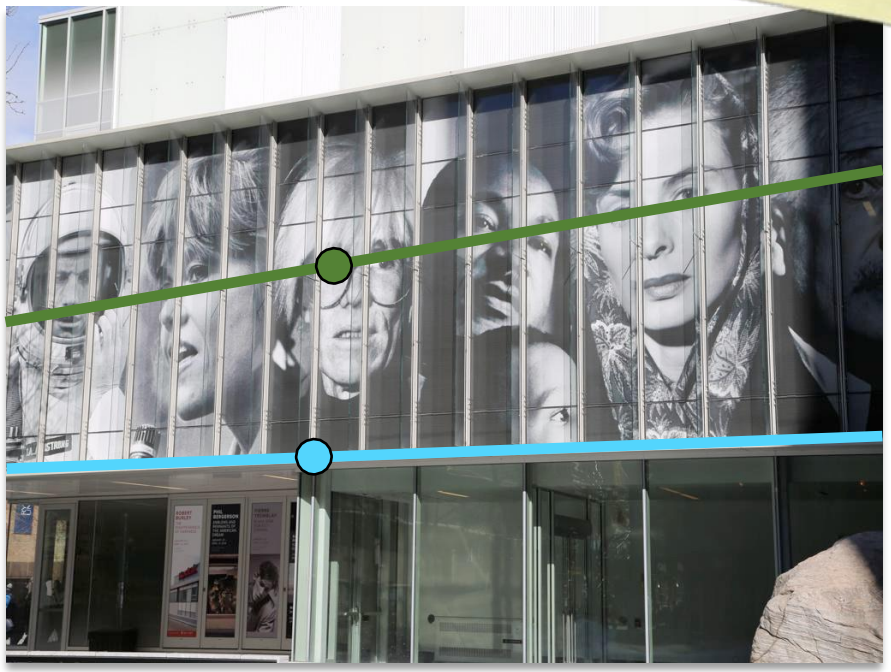
原始
图像对



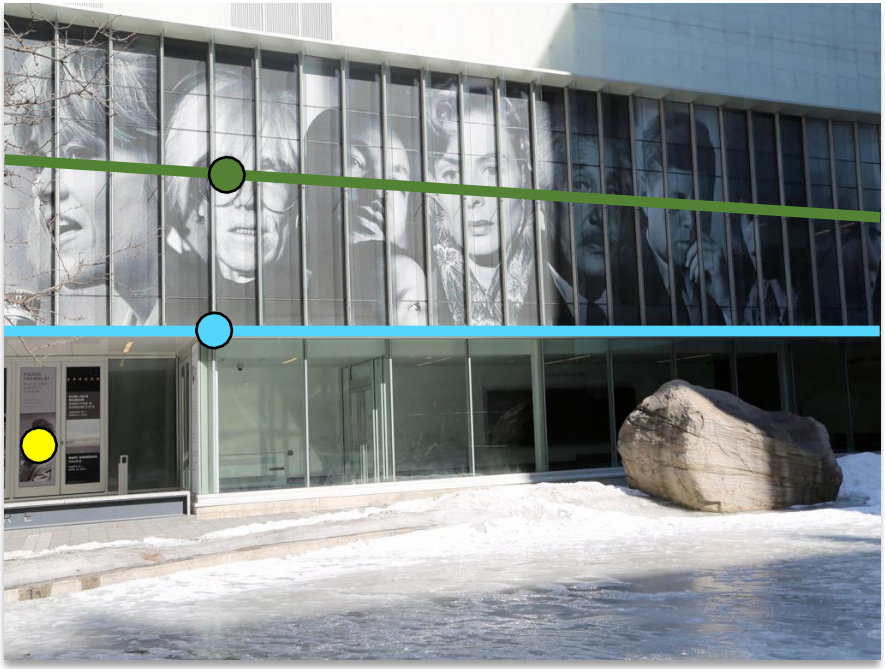


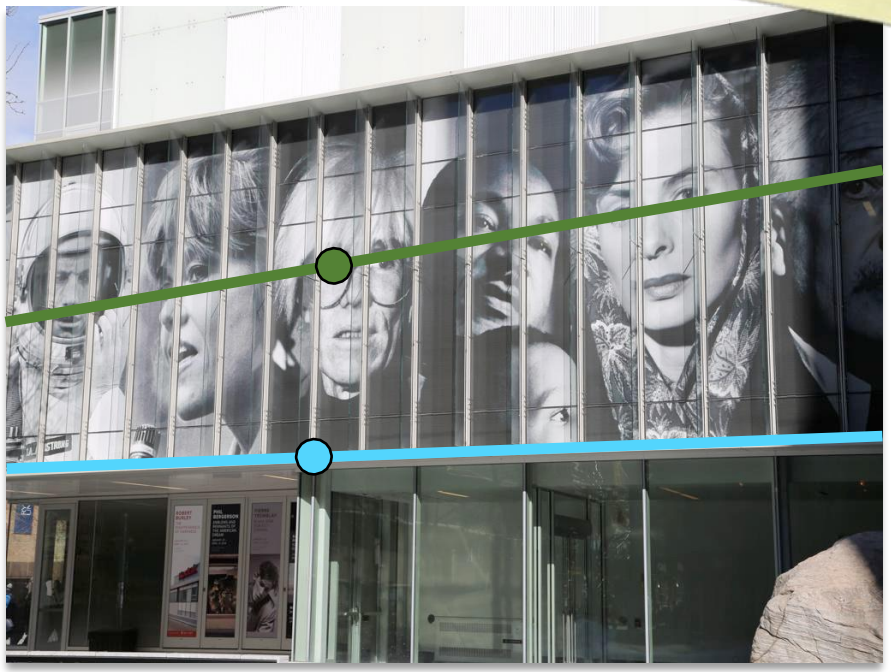
原始
图像对



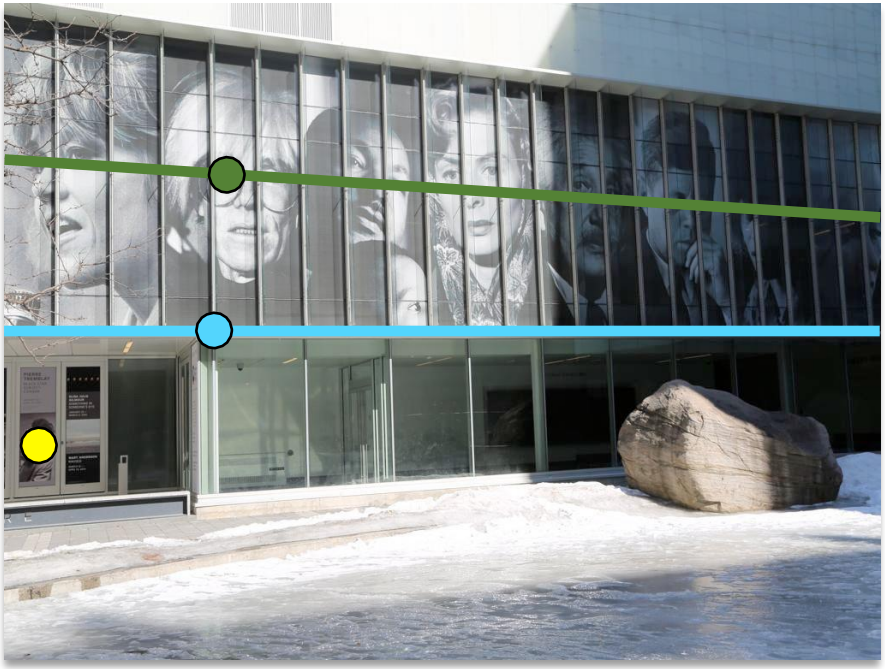


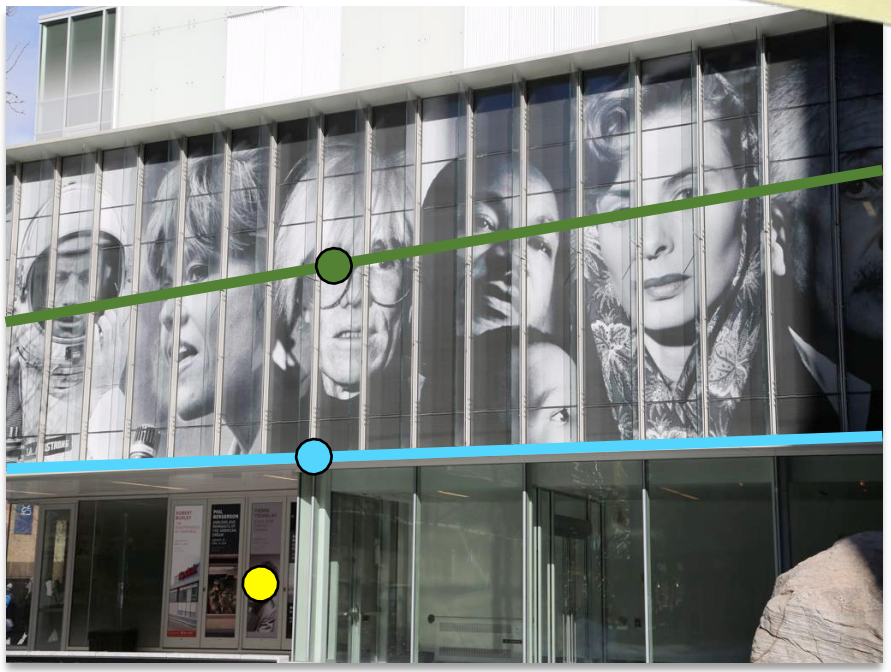
原始
图像对



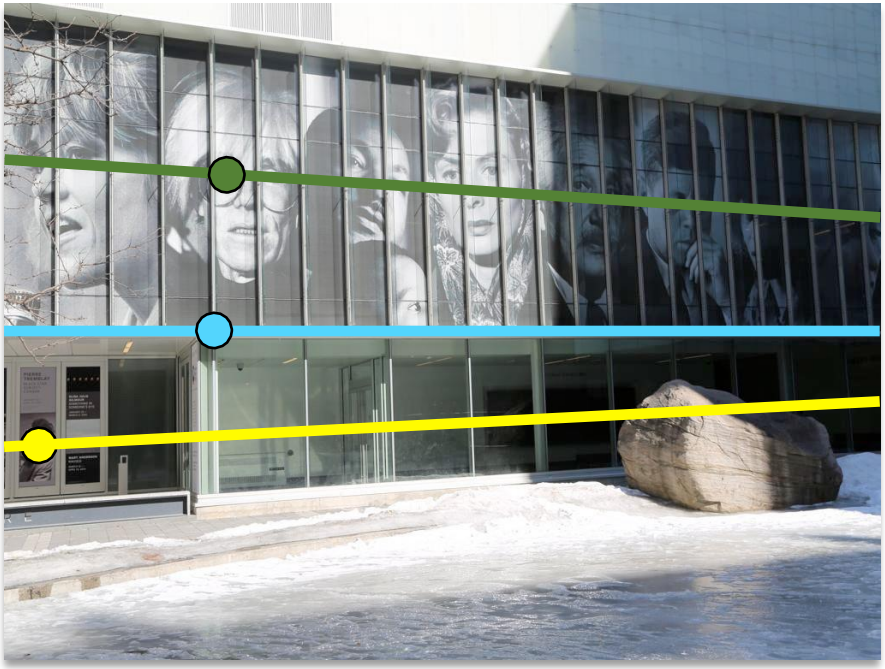


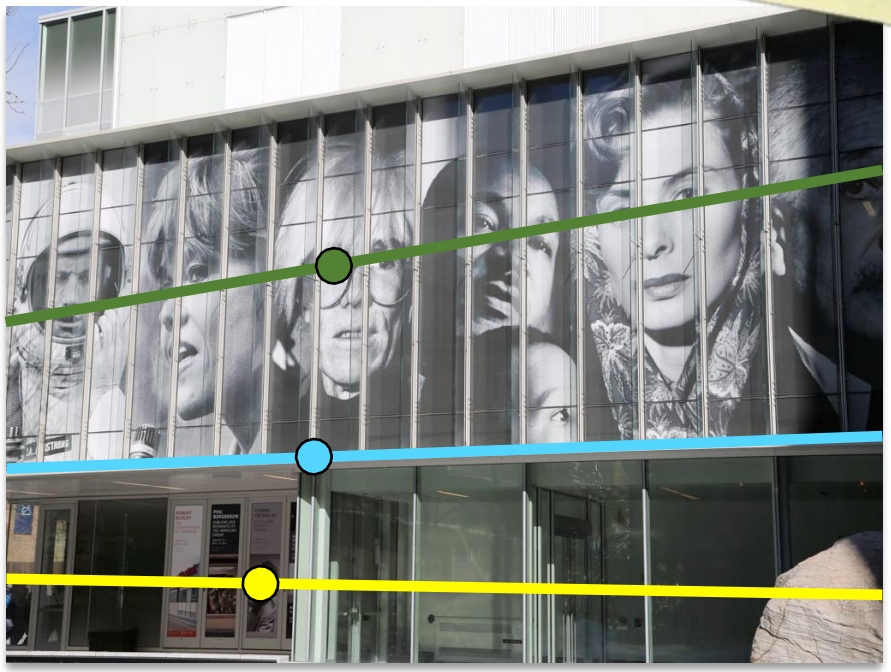
原始
图像对





原始
图像对





矫正后
图像对

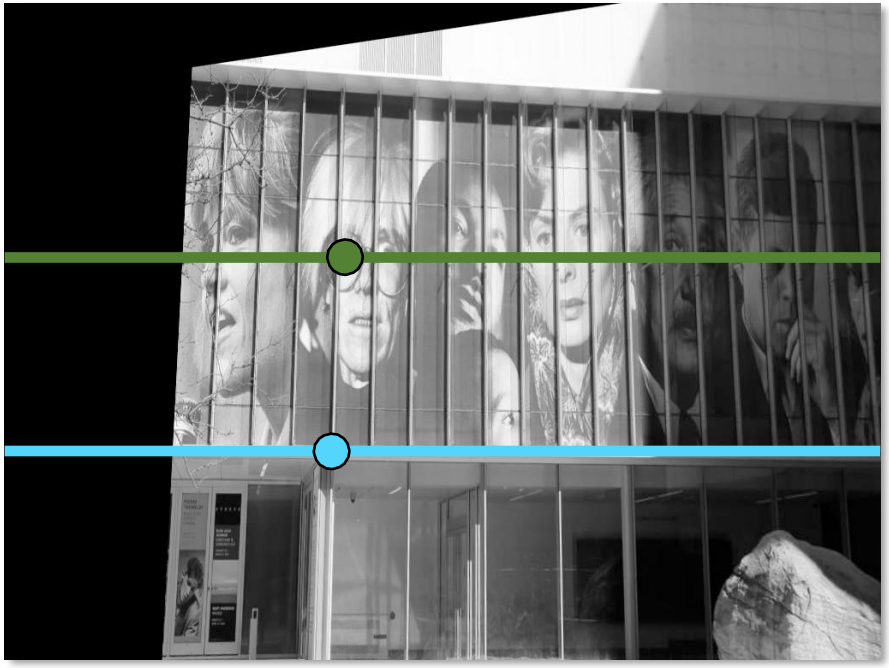
矫正后
图像对

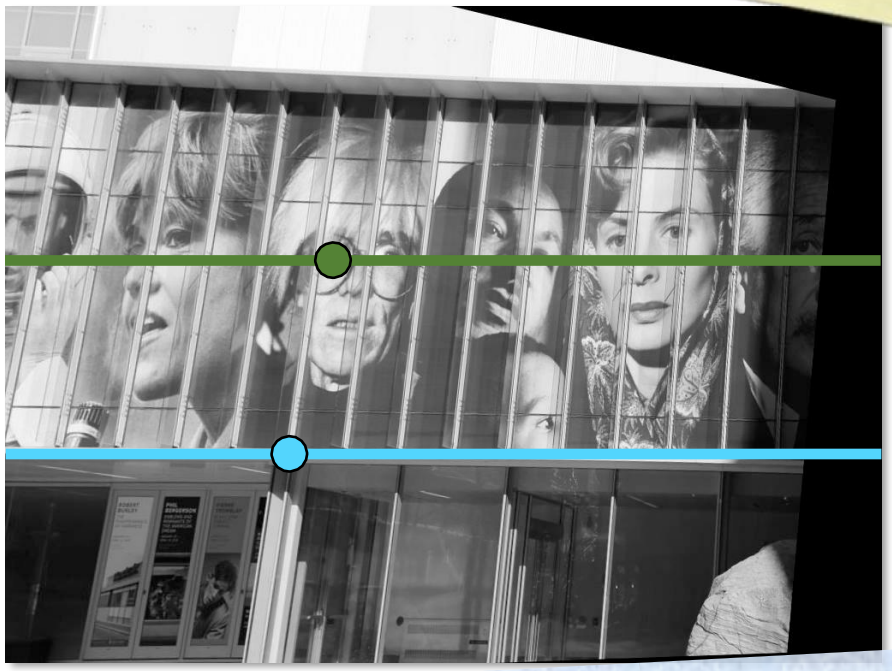
对应点在同一条扫描线上

矫正后
图像对

对应点在同一条扫描线上

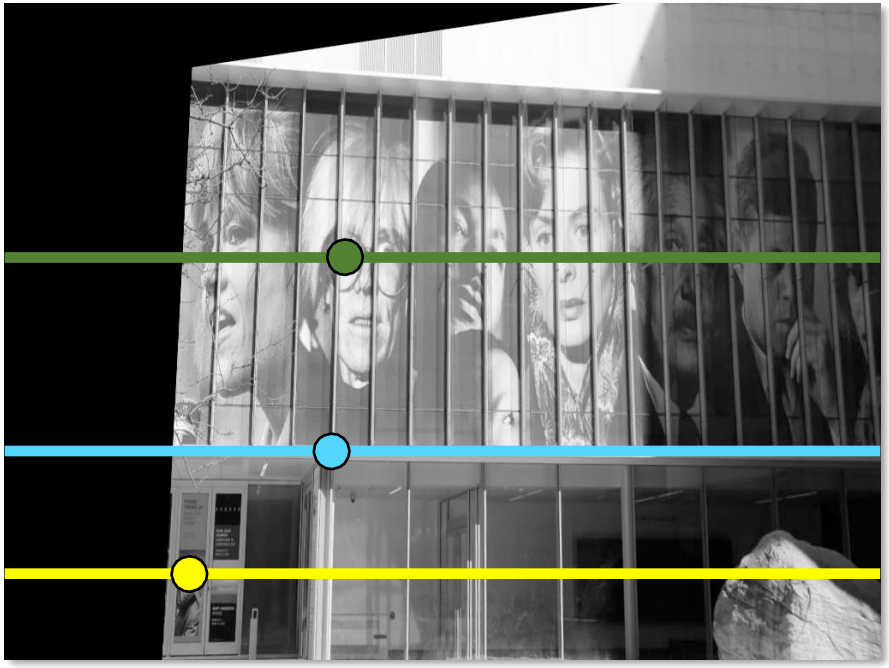
矫正后
图像对





对应点在同一条扫描线上

矫正后
图像对



对应点在同一条扫描线上

Python时间

图像矫正

```
# Load stereo image pair and convert to grayscale
I1 = cv2.imread('left.png', cv2.IMREAD_GRAYSCALE)
I2 = cv2.imread('right.png', cv2.IMREAD_GRAYSCALE)

# Find the keypoints and descriptors with SIFT
sift = cv2.SIFT_create()
kp1, des1 = sift.detectAndCompute(I1, None)
kp2, des2 = sift.detectAndCompute(I2, None)

# Visualize keypoints
I1_sift = cv2.drawKeypoints(I1, kp1, None,
   flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
I2_sift = cv2.drawKeypoints(I2, kp2, None,
   flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

I1_I2_sift = np.concatenate((I1_sift, I2_sift), axis=1)
cv2.imshow('Image SIFT keypoints', I1_I2_sift)
```

```
# Load stereo image pair and convert to grayscale
I1 = cv2.imread('left.png', cv2.IMREAD_GRAYSCALE)
I2 = cv2.imread('right.png', cv2.IMREAD_GRAYSCALE)

# Find the keypoints and descriptors with SIFT
sift = cv2.SIFT_create()
kp1, des1 = sift.detectAndCompute(I1, None)
kp2, des2 = sift.detectAndCompute(I2, None)

# Visualize keypoints
I1_sift = cv2.drawKeypoints(I1, kp1, None,
   flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
I2_sift = cv2.drawKeypoints(I2, kp2, None,
   flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

I1_I2_sift = np.concatenate((I1_sift, I2_sift), axis=1)
cv2.imshow('Image SIFT keypoints', I1_I2_sift)
```

```
# Load stereo image pair and convert to grayscale
I1 = cv2.imread('left.png', cv2.IMREAD_GRAYSCALE)
I2 = cv2.imread('right.png', cv2.IMREAD_GRAYSCALE)

# Find the keypoints and descriptors with SIFT
sift = cv2.SIFT_create()
kp1, des1 = sift.detectAndCompute(I1, None)
kp2, des2 = sift.detectAndCompute(I2, None)

# Visualize keypoints
I1_sift = cv2.drawKeypoints(I1, kp1, None,
   flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
I2_sift = cv2.drawKeypoints(I2, kp2, None,
   flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

I1_I2_sift = np.concatenate((I1_sift, I2_sift), axis=1)
cv2.imshow('Image SIFT keypoints', I1_I2_sift)
```

```
# Load stereo image pair and convert to grayscale
I1 = cv2.imread('left.png', cv2.IMREAD_GRAYSCALE)
I2 = cv2.imread('right.png', cv2.IMREAD_GRAYSCALE)

# Find the keypoints and descriptors with SIFT
sift = cv2.SIFT_create()
kp1, des1 = sift.detectAndCompute(I1, None)
kp2, des2 = sift.detectAndCompute(I2, None)

# Visualize keypoints
I1_sift = cv2.drawKeypoints(I1, kp1, None,
   flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
I2_sift = cv2.drawKeypoints(I2, kp2, None,
   flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

I1_I2_sift = np.concatenate((I1_sift, I2_sift), axis=1)
cv2.imshow('Image SIFT keypoints', I1_I2_sift)
```

关键点

```
# Load stereo image pair and convert to grayscale
I1 = cv2.imread('left.png', cv2.IMREAD_GRAYSCALE)
I2 = cv2.imread('right.png', cv2.IMREAD_GRAYSCALE)

# Find the keypoints and descriptors with SIFT
sift = cv2.SIFT_create()
kp1, des1 = sift.detectAndCompute(I1, None)
kp2, des2 = sift.detectAndCompute(I2, None)

# Visualize keypoints
I1_sift = cv2.drawKeypoints(I1, kp1, None,
   flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
I2_sift = cv2.drawKeypoints(I2, kp2, None,
   flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

I1_I2_sift = np.concatenate((I1_sift, I2_sift), axis=1)
cv2.imshow('Image SIFT keypoints', I1_I2_sift)
```

特征描述子

```
# Load stereo image pair and convert to grayscale
I1 = cv2.imread('left.png', cv2.IMREAD_GRAYSCALE)
I2 = cv2.imread('right.png', cv2.IMREAD_GRAYSCALE)

# Find the keypoints and descriptors with SIFT
sift = cv2.SIFT_create()
kp1, des1 = sift.detectAndCompute(I1, None)
kp2, des2 = sift.detectAndCompute(I2, None)

# Visualize keypoints
I1_sift = cv2.drawKeypoints(I1, kp1, None,
   flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)
I2_sift = cv2.drawKeypoints(I2, kp2, None,
   flags=cv2.DRAW_MATCHES_FLAGS_DRAW_RICH_KEYPOINTS)

I1_I2_sift = np.concatenate((I1_sift, I2_sift), axis=1)
cv2.imshow('Image SIFT keypoints', I1_I2_sift)
```

寻找
对应点

```
# Match keypoints in both images
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees
flann = cv2.FlannBasedMatcher(index_params, {})
matches = flann.knnMatch(des1, des2, k=2)

# Keep good matches: calculate distinctive image features
good, pts1, pts2 = [], [], []

for i, (m, n) in enumerate(matches):
    if m.distance < 0.7 * n.distance:
        good.append([m])
        pts1.append(kp1[m.queryIdx].pt)
        pts2.append(kp2[m.trainIdx].pt)

keypoint_matches = cv2.drawMatchesKnn(I1, kp1, I2, kp2, good, None,
    flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
cv2.imshow('Keypoint matches', keypoint_matches)
```

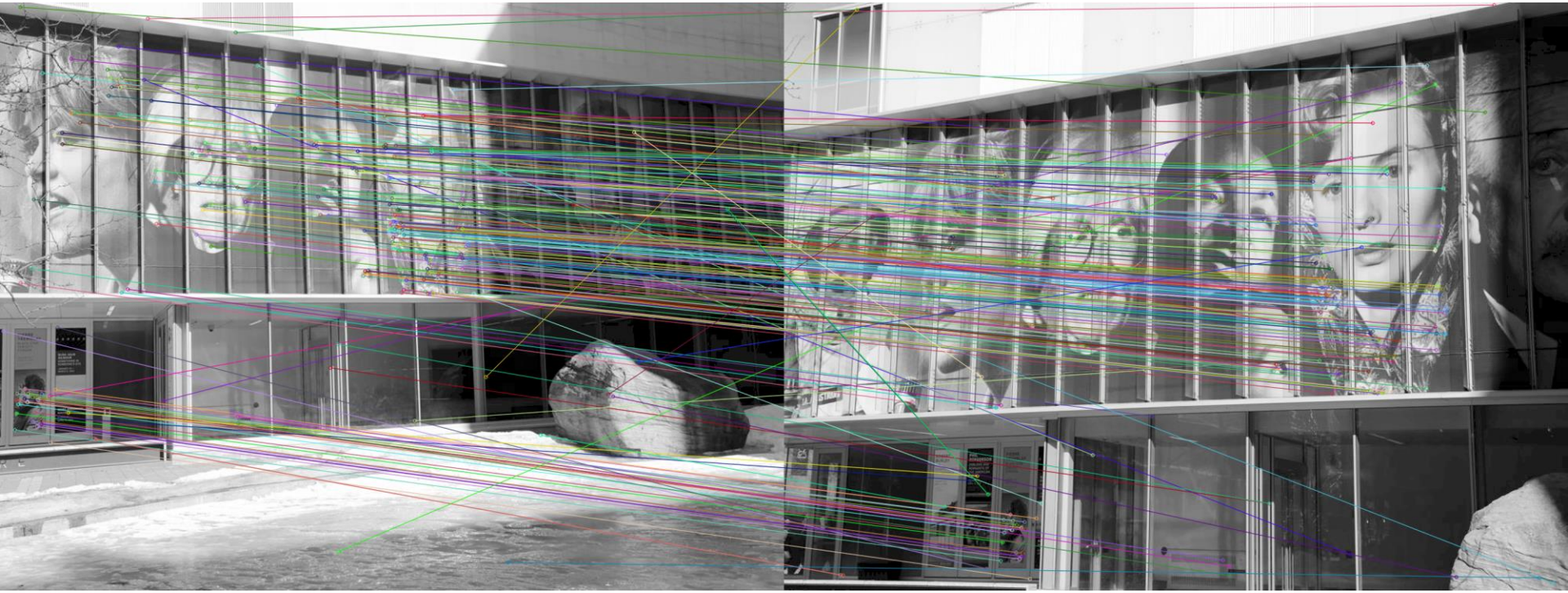
```
# Match keypoints in both images
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
flann = cv2.FlannBasedMatcher(index_params, {})
matches = flann.knnMatch(des1, des2, k=2)

# Keep good matches: calculate distinctive image features
good, pts1, pts2 = [], [], []

for i, (m, n) in enumerate(matches):
    if m.distance < 0.7 * n.distance:
        good.append([m])
        pts1.append(kp1[m.queryIdx].pt)
        pts2.append(kp2[m.trainIdx].pt)

keypoint_matches = cv2.drawMatchesKnn(I1, kp1, I2, kp2, good, None,
    flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
cv2.imshow('Keypoint matches', keypoint_matches)
```

```
# Match keypoints in both images
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
flann = cv2.FlannBasedMatcher(index_params, {})
matches = flann.knnMatch(des1, des2, k=2)

# Keep good matches: calculate distinctive image features
good, pts1, pts2 = [], [], []

for i, (m, n) in enumerate(matches):
    if m.distance < 0.7 * n.distance:
        good.append([m])
        pts1.append(kp1[m.queryIdx].pt)
        pts2.append(kp2[m.trainIdx].pt)

keypoint_matches = cv2.drawMatchesKnn(I1, kp1, I2, kp2, good, None,
    flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
cv2.imshow('Keypoint matches', keypoint_matches)
```

```
# Match keypoints in both images
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
flann = cv2.FlannBasedMatcher(index_params, {})
matches = flann.knnMatch(des1, des2, k=2)

# Keep good matches: calculate distinctive image features
good, pts1, pts2 = [], [], []

for i, (m, n) in enumerate(matches):
    if m.distance < 0.7 * n.distance:
        good.append([m])
        pts1.append(kp1[m.queryIdx].pt)
        pts2.append(kp2[m.trainIdx].pt)

keypoint_matches = cv2.drawMatchesKnn(I1, kp1, I2, kp2, good, None,
    flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
cv2.imshow('Keypoint matches', keypoint_matches)
```

```
# Match keypoints in both images
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
flann = cv2.FlannBasedMatcher(index_params, {})
matches = flann.knnMatch(des1, des2, k=2)

# Keep good matches: calculate distinctive image features
good, pts1, pts2 = [], [], []
for i, (m, n) in enumerate(matches):
    if m.distance < 0.7 * n.distance:
        good.append([m])
        pts1.append(kp1[m.queryIdx].pt)
        pts2.append(kp2[m.trainIdx].pt)

keypoint_matches = cv2.drawMatchesKnn(I1, kp1, I2, kp2, good, None,
    flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
cv2.imshow('Keypoint matches', keypoint_matches)
```

```
# Match keypoints in both images
FLANN_INDEX_KDTREE = 1
index_params = dict(algorithm=FLANN_INDEX_KDTREE, trees=5)
flann = cv2.FlannBasedMatcher(index_params, {})
matches = flann.knnMatch(des1, des2, k=2)

# Keep good matches: calculate distinctive image features
good, pts1, pts2 = [], [], []

for i, (m, n) in enumerate(matches):
    if m.distance < 0.7 * n.distance:
        good.append([m])
        pts1.append(kp1[m.queryIdx].pt)
        pts2.append(kp2[m.trainIdx].pt)

keypoint_matches = cv2.drawMatchesKnn(I1, kp1, I2, kp2, good, None,
    flags=cv2.DRAW_MATCHES_FLAGS_NOT_DRAW_SINGLE_POINTS)
cv2.imshow('Keypoint matches', keypoint_matches)
```



```
# Calculate the fundamental matrix for the cameras
pts1 = np.float32(pts1)
pts2 = np.float32(pts2)

fundamental_matrix, inliers = cv2.findFundamentalMat(
    pts1, pts2, cv2.FM_RANSAC,
    ransacReprojThreshold=0.9, confidence=0.99
)

# Select only inlier points
pts1 = pts1[inliers.ravel() == 1]
pts2 = pts2[inliers.ravel() == 1]
```

计算
基础矩阵

```
# Calculate the fundamental matrix for the cameras
pts1 = np.float32(pts1)
pts2 = np.float32(pts2)

fundamental_matrix, inliers = cv2.findFundamentalMat(
    pts1, pts2, cv2.FM_RANSAC,
    ransacReprojThreshold=0.9, confidence=0.99
)

# Select only inlier points
pts1 = pts1[inliers.ravel() == 1]
pts2 = pts2[inliers.ravel() == 1]
```

```
# Calculate the fundamental matrix for the cameras
pts1 = np.float32(pts1)
pts2 = np.float32(pts2)

fundamental_matrix, inliers = cv2.findFundamentalMat(
    pts1, pts2, cv2.FM_RANSAC,
    ransacReprojThreshold=0.9, confidence=0.99
)

# Select only inlier points
pts1 = pts1[inliers.ravel() == 1]
pts2 = pts2[inliers.ravel() == 1]
```

```
# Calculate the fundamental matrix for the cameras
pts1 = np.float32(pts1)
pts2 = np.float32(pts2)

fundamental_matrix, inliers = cv2.findFundamentalMat(
    pts1, pts2, cv2.FM_RANSAC,
    ransacReprojThreshold=0.9, confidence=0.99
)

# Select only inlier points
pts1 = pts1[inliers.ravel() == 1]
pts2 = pts2[inliers.ravel() == 1]
```

```
# Calculate the fundamental matrix for the cameras
pts1 = np.float32(pts1)
pts2 = np.float32(pts2)

fundamental_matrix, inliers = cv2.findFundamentalMat(
    pts1, pts2, cv2.FM_RANSAC,
    ransacReprojThreshold=0.9, confidence=0.99
)

# Select only inlier points
pts1 = pts1[inliers.ravel() == 1]
pts2 = pts2[inliers.ravel() == 1]
```

```
# Stereo rectification (uncalibrated variant)
h1, w1 = img1.shape
h2, w2 = img2.shape
_, H1, H2 = cv2.stereoRectifyUncalibrated(
      pts1, pts2, fundamental_matrix, imgSize=(w1, h1)
)

# Rectify the images
I1_rect = cv2.warpPerspective(I1, H1, (w1, h1))
I2_rect = cv2.warpPerspective(I2, H2, (w2, h2))

# Visualize rectified images
I1_I2_rect = np.concatenate((I1_rect, I2_rect), axis=1)
cv2.imshow('Rectified images', I1_I2_rect), cv2.waitKey(0)
```

矫正图像

```
# Stereo rectification (uncalibrated variant)
h1, w1 = img1.shape
h2, w2 = img2.shape
_, H1, H2 = cv2.stereoRectifyUncalibrated(
    pts1, pts2, fundamental_matrix, imgSize=(w1, h1)
)

# Rectify the images
I1_rect = cv2.warpPerspective(I1, H1, (w1, h1))
I2_rect = cv2.warpPerspective(I2, H2, (w2, h2))

# Visualize rectified images
I1_I2_rect = np.concatenate((I1_rect, I2_rect), axis=1)
cv2.imshow('Rectified images', I1_I2_rect), cv2.waitKey(0)
```

```
# Stereo rectification (uncalibrated variant)
h1, w1 = img1.shape
h2, w2 = img2.shape
_, H1, H2 = cv2.stereoRectifyUncalibrated(
    pts1, pts2, fundamental_matrix, imgSize=(w1, h1)
)

# Rectify the images
I1_rect = cv2.warpPerspective(I1, H1, (w1, h1))
I2_rect = cv2.warpPerspective(I2, H2, (w2, h2))

# Visualize rectified images
I1_I2_rect = np.concatenate((I1_rect, I2_rect), axis=1)
cv2.imshow('Rectified images', I1_I2_rect), cv2.waitKey(0)
```

```
# Stereo rectification (uncalibrated variant)
h1, w1 = img1.shape
h2, w2 = img2.shape
_, H1, H2 = cv2.stereoRectifyUncalibrated(
      pts1, pts2, fundamental_matrix, imgSize=(w1, h1)
)

# Rectify the images
I1_rect = cv2.warpPerspective(I1, H1, (w1, h1))
I2_rect = cv2.warpPerspective(I2, H2, (w2, h2))

# Visualize rectified images
I1_I2_rect = np.concatenate((I1_rect, I2_rect), axis=1)
cv2.imshow('Rectified images', I1_I2_rect), cv2.waitKey(0)
```