**目标：**

本章节你需要学习以下内容:

* 我们将学习如何从立体图像创建深度图。

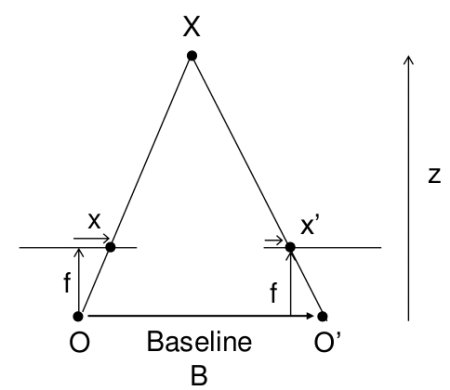
## api：

### stereo = cv.StereoSGBM\_create(minDisparity = min\_disp, numDisparities = num\_disp, blockSize = 16, P1 = 8\*3\*window\_size\*\*2, P2 = 32\*3\*window\_size\*\*2, disp12MaxDiff = 1, uniquenessRatio = 10, speckleWindowSize = 100, speckleRange = 32 )

### disp = stereo.compute(imgL, imgR).astype(np.float32) / 16.0

### points = cv.reprojectImageTo3D(disp, Q)

**基础**

在上一节中我们学习了对极约束的基本概念和相关术语。如果同一场景有两幅图像的话我们在直觉上就可以获得图像的深度信息。下面是的这幅图和其中的数学公式证明我们的直觉是对的。（图像来源 image courtesy） [](https://camo.githubusercontent.com/01e6e76b5ebc0228b0d459d145773e88440ef738bd9d01a6ca6576232d5f90e0/68747470733a2f2f646f63732e6f70656e63762e6f72672f342e302e302f73746572656f5f64657074682e6a7067)

上图包含等效三角形。编写等效方程将产生以下结果：

𝑑𝑖𝑠𝑝𝑎𝑟𝑖𝑡𝑦=𝑥−𝑥′=𝐵𝑓𝑍

x 和 x' 分别是图像中的点到 3D 空间中的点和到摄像机中心的距离。B 是这两个摄像机之间的距离，f 是摄像机的焦距。上边的等式告诉我们点的深度与x 和 x' 的差成反比。所以根据这个等式我们就可以得到图像中所有点的深度图。

这样就可以找到两幅图像中的匹配点了。前面我们已经知道了对极约束可以使这个操作更快更准。一旦找到了匹配，就可以计算出 disparity 了。让我们看看在 OpenCV 中怎样做吧。

**代码实现1.**

|  |  |
| --- | --- |
|  | **import numpy as np import cv2 as cv from matplotlib import pyplot as plt  imgL = cv.imread('./tsukuba\_l.png',0) imgR = cv.imread('./tsukuba\_r.png',0)  *# disparity range is tuned for 'aloe' image pair* window\_size = 3 min\_disp = 16 num\_disp = 112 - min\_disp stereo = cv.StereoSGBM\_create(minDisparity=min\_disp,  numDisparities=num\_disp,  blockSize=16,  P1=8 \* 3 \* window\_size \*\* 2,  P2=32 \* 3 \* window\_size \*\* 2,  disp12MaxDiff=1,  uniquenessRatio=10,  speckleWindowSize=100,  speckleRange=32  )  print('computing disparity...') *# disp = stereo.compute(imgL, imgR).astype(np.float32) / 16.0* disp = stereo.compute(imgR, imgL).astype(np.float32)/16.0 *# 有时候反过来效果更好* plt.imshow(disp,'gray') plt.show()** |

### 效果：

|  |
| --- |
|  |

## 当你熟悉StereoBM时，会有一些参数，可能需要微调参数以获得更好，更平滑的结果。参数：

* **texture\_threshold：过滤出纹理不足以进行可靠匹配**
* **区域斑点范围和大小：基于块的匹配器通常会在对象边界附近产生“斑点”，其中匹配窗口捕获一侧的前景和背景 在另一场景中，匹配器似乎还在桌子上投影的纹理中找到小的虚假匹配项。为了消除这些伪像，我们使用由speckle\_size和speckle\_range参数控制的散斑滤镜对视差图像进行后处理。speckle\_size是将视差斑点排除为“斑点”的像素数。speckle\_range控制必须将值差异视为同一对象的一部分的程度。**
* **视差数量：滑动窗口的像素数。它越大，可见深度的范围就越大，但是需要更多的计算。**
* **min\_disparity：从开始搜索的左像素的x位置开始的偏移量。**
* **uniqueness\_ratio：另一个后过滤步骤。如果最佳匹配视差不足够好于搜索范围中的所有其他视差，则将像素滤出。如果texture\_threshold和斑点过滤仍在通过虚假匹配，则可以尝试进行调整。**
* **prefilter\_size和prefilter\_cap：预过滤阶段，可标准化图像亮度并增强纹理，以准备块匹配。通常，你不需要调整这些。**

## 实例2，参考官方示例，还能够创建3D模型

|  |  |
| --- | --- |
|  | **from** \_\_future\_\_ **import** print\_function  **import** numpy **as** np **import** cv2 **as** cv  ply\_header = **'''ply format ascii 1.0 element vertex %(vert\_num)d property float x property float y property float z property uchar red property uchar green property uchar blue end\_header '''  def** write\_ply(fn, verts, colors):  verts = verts.reshape(-1, 3)  colors = colors.reshape(-1, 3)  verts = np.hstack([verts, colors])  **with** open(fn, **'wb'**) **as** f:  f.write((ply\_header % dict(vert\_num=len(verts))).encode(**'utf-8'**))  np.savetxt(f, verts, fmt=**'%f %f %f %d %d %d '**)   **def** main():  print(**'loading images...'**)  imgL = cv.pyrDown(cv.imread(**'aloeL.jpg'**)) *# downscale images for faster processing* imgR = cv.pyrDown(cv.imread(**'aloeR.jpg'**))   *# disparity range is tuned for 'aloe' image pair* window\_size = 3  min\_disp = 16  num\_disp = 112-min\_disp  stereo = cv.StereoSGBM\_create(minDisparity = min\_disp,  numDisparities = num\_disp,  blockSize = 16,  P1 = 8\*3\*window\_size\*\*2,  P2 = 32\*3\*window\_size\*\*2,  disp12MaxDiff = 1,  uniquenessRatio = 10,  speckleWindowSize = 100,  speckleRange = 32  )   print(**'computing disparity...'**)  disp = stereo.compute(imgL, imgR).astype(np.float32) / 16.0   print(**'generating 3d point cloud...'**,)  h, w = imgL.shape[:2]  f = 0.8\*w *# guess for focal length* Q = np.float32([[1, 0, 0, -0.5\*w],  [0,-1, 0, 0.5\*h], *# turn points 180 deg around x-axis,* [0, 0, 0, -f], *# so that y-axis looks up* [0, 0, 1, 0]])  points = cv.reprojectImageTo3D(disp, Q)  colors = cv.cvtColor(imgL, cv.COLOR\_BGR2RGB)  mask = disp > disp.min()  out\_points = points[mask]  out\_colors = colors[mask]  out\_fn = **'out.ply'** write\_ply(out\_fn, out\_points, out\_colors)  print(**'%s saved'** % out\_fn)   cv.imshow(**'left'**, imgL)  cv.imshow(**'disparity'**, (disp-min\_disp)/num\_disp)  cv.waitKey()   print(**'Done'**)   **if** \_\_name\_\_ == **'\_\_main\_\_'**:  main()  cv.destroyAllWindows() |

### 效果

|  |
| --- |
|  |

## 当然这些程序有点复杂，需要慢慢弄懂