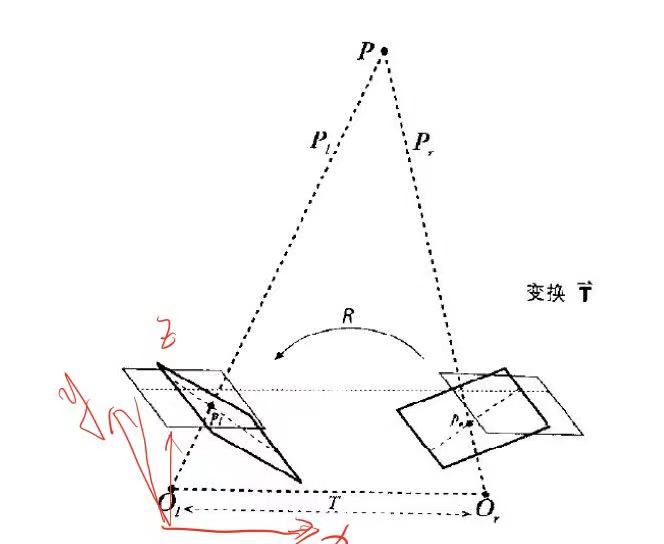
Openmvs



# 极线校正： 将相机在数学上对准到同一观察平面上，使得相机上像素行是严格对齐的， 校正目的：对两幅图像的二维匹配搜索变成一维，节省计算量，排除虚假匹配点

左相机外参Rl， tl， 右相机外参Rr， tr

*openmvs中全部采用的是旋转和光心坐标的方式来表示相机外参p\_c=R\*p\_w+t， p\_c在图像上的坐标，p\_w世界坐标， t平移，相机中心p\_c=[0,0,0]，则p\_w=-R\_invert\*t，p\_w就表示光心坐标C*

## 1.1 首先是X轴的旋转，将X轴旋转到与基线相同的方向，则旋转向量

## 1.2 然后确定新的y轴，只要满足与e1正交即可，选择与旧的主光轴和新的X轴垂直的方向

，这里code中没有tl，平移不会影响旋转之后的方向

## 1.3 新的z轴与新的x轴和y轴互相垂直

坐标转换的旋转矩阵 Rnew=[e1, e2, e3]

左右相机新的旋转矩阵

Rlnew = RnewRlt;

Rrnew = RnewRrt;

校正之后新的基线的计算

1. 先求出未校正之前，右相机光心在左相机坐标系下的坐标
2. 然后在求解左相机光心在右相机校正后的坐标系下的坐标（也等于与右相机校正后的坐标下光心[0, 0, 0]的距离）

（左图到右图的变换）

Tx = t.x，其为负数

# 视差图与深度图转换，矫正后坐标系下的视差图与校正前的坐标系下的深度图的互相转换

## 根据投影方程，先将校正后的视差图转成校正后的深度图

## 2, 写成矩阵形式

## 3, 将校正后的坐标系下的深度图转到校正之前的相机坐标系下

校正之前的相机坐标系下（x, y, z），校正之后的相机坐标系下的坐标（x’,y’,z’）

则有

## 在将校正前相机坐标系下的位置转换成图像坐标可得

## 将上面合并可得校正后坐标系下的视差图（u’， v’， disp）与校正前的坐标系下深度图（u，v, depth）=（X，Y，Z）的转换

# 3 深度估计

1.ComputeDepthMaps

OpenMVS源码中流程（在Scene.h

4. EstimatePointNormals

3. EstimatePointColors

2. FuseDepthMaps

DenseReconstruction

中）

输入：图像，位姿， 输出：depth，点云

数据结构从大到小

DenseDepthMapData -> 所有数据

DepthMapsData-> 用于计算depth maps的类

DepthData ->单帧深度图所需要的数据

其中ComputeDepthMaps中有4个步骤(小写部分都是源码中对应函数的名字) 主要在DepthMapsData对象中在SceneDensify.h文件中。

1. prepare image
2. Select Views
3. Dense Reconstruction Estimate
4. EVT\_PROCESSIMAGE, InitViews
5. EVT\_ESTIMATEDEPTHMAP, extract depth map

ii.i. Patch-Match algorithm EstimateDepthMap

ii.ii. sgm algorithm data.sgm.match

iii. EVT\_OPTIMIZEDEPTHMAP RemoveSmallSegments/GapInterpolation

iiii. EVT\_SAVEDEPTHMAP,

ExportDepthMap/ExportConfidenceMap

/ExportPointCloud

1. Dense Reconstruction Filter
2. EVT\_FILTERDEPTHMAP, data.depthMaps.FilterDepthMap
3. EVT\_ADJUSTDEPTHMAP,
4. EVT\_FAIL return
5. Prepare Image

图片尺寸缩放，相机的内参也进行同样的缩放

=2 \* u, X和Z不变的情况下， 和也要扩大相应的倍数

1. 选择邻域Select Views
2. 搜索该帧的所有邻域帧 data.depthMaps.SelectViews(depthData)
   1. 邻域选择scene.SelectNeighborViews()
   2. 邻域滤波Scene::FilterNeighborViews()
3. 选择该帧的最佳邻域帧用于计算深度 data.depthMaps.SelectViews(data.images, imagesMap, data.neighborsMap)

邻域选择的时候，共有三个条件，分别是

共视点f在两个图像(V,R)的夹角(fV与fR组成的夹角)(夹角设置的阈值为10

度)；

邻域帧R与当前帧V的分辨率是否接近；

共视点在图像中覆盖的面积area 。

参考论文

Multi-View Stereo for Community Photo Collections

夹角的score公式

夹角越接近或者等于该阈值，得分越高，论文中取值为1.6次方，而不是2

邻域帧R与当前帧V的分辨率指的是共视区域的分辨率是否接近

计算方法 空间内共视一点f，在左图中的投影位置为v1，在右图的投影的位置为v2，当f在空间内移动一段距离，在左图中的投影位置为v1’，右图中的投影位置为v2’， SVf = |v1’-v1|, SRf = |v2’-v2|, r=SRf/SVf

代码中直接简化处理f/depth，来等于SVf和SRf，类似disp=fb/depth, 当b为单位长度时，视差的变化值，代码中取值1.6

共视点在图像中覆盖的面积area，当前帧与共视帧的共视点，这些点在共视帧上所占的面积

计算当前帧V和邻域帧R共视所有的共视点，如果该点投影分别在当前帧和邻域帧的图像范围内，则将共视点在当前帧的投影存储起来用于面积的计算。

代码中的计算面积的方法是按比例计算的，先将特征点投影坐标归一化到0-1，然后乘以16，相当于归一化到0-16，然后设置一个16\*16的矩阵，对应位置上设为1，然后计算1的数量除以16\*16，得到共视点在当前帧中覆盖的面积

邻域滤波

对邻域选择时的三个条件分别设置阈值，均满足的话就选择作为邻域

选择该帧的最佳邻域帧

经过上面的计算，每张图片都选出nMaxViews个邻域帧，则构成一个马尔科夫随机场的labeling问题(能量优问题)，即每一个view（node）都有n个邻域（label），优化目标是给每一个node选择一个label使得整体能量最小

该问题的Markov Random Field能量函数：

公式中Y是标签label， X是节点node

UnaryCost:之前计算的score，用平均score归一化后的值

avgScore/score 。view选当前标签（score越大该邻域越合适）的代价， avgScore/score就越小即代价越小则该邻域就越合适

PairwiseCost: 惩罚两个node的label相同的情况即不鼓励任意两个的view相同

area是指两个node之间的覆盖的面积， 指的是xi与li的覆盖面积

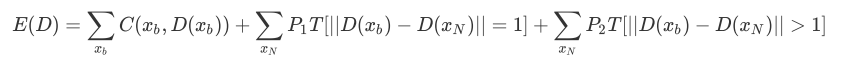
1. 深度估计Dense Reconstruction Estimate
   1. init View

将之前的最佳邻域帧放入队列中，如果没有最佳邻域帧，就将得分最高的前几帧放入计算队列中，然后并把该图片按照共视区域的分辨率比值进行缩放

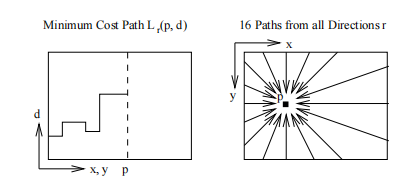
* 1. sgm algorithm data.sgm.match
     1. 极线校正图片，前面推导的Image::StereoRectifyImages（）
     2. 视差图初始化

根据现有的稀疏3D点云，进行三角化先得到深度图

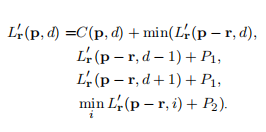
* + - 1. 需要对图像的四个角点进行差值，使得深度图覆盖全张图，四个角点的深度由，角点与相机坐标的原点的射线，与周围几个面的延长面的角点的深度值的加权值depth\_new=（Σscore\*depth）/numPoints=Σ（score/numPoints）\*depth，score是角点单位深度的相机坐标，与相邻面的中心点的单位深度的相机坐标的距离值的倒数。
      2. 计算每个像素点的初始深度值(n/d)x=1 ,x=depth\*k\_inv[u,v,1] 则 depth=（n/d）\*k\_inv[u,v,1]， 其中（n/d）x = 1，是每个平面的方程，然后计算该平面覆盖范围内的像素点的深度值
      3. 根据极线校正的过程反过来计算，得到初始视差图
    1. 代价聚合

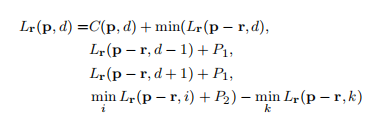


将像素所有视差下的匹配代价进行像素周围所有路径上的一维聚合得到路径下的路径代价值，然后将所有的路径代价相加得到该像素聚合后的匹配代价值，然后win take all

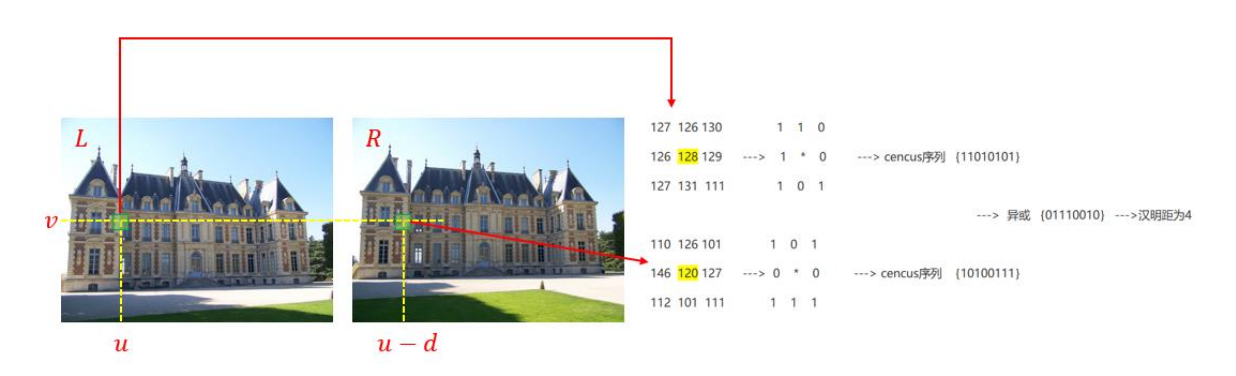


运用动态规划的方法来求具体的点的代价，然后最小代价所对应的视差图就是该像素点最佳的视差图





其中cost函数，openmvs用的是census



* + 1. 深度图优化