第6讲 立体视觉

周文晖

计算机学院

从图像中恢复三维

•D. Marr视觉计算理论：计算机视觉的第三阶段（后期阶段）是获得物体的三维模型表征。

如何从图像中自动计算三维几何？

•换句话说：

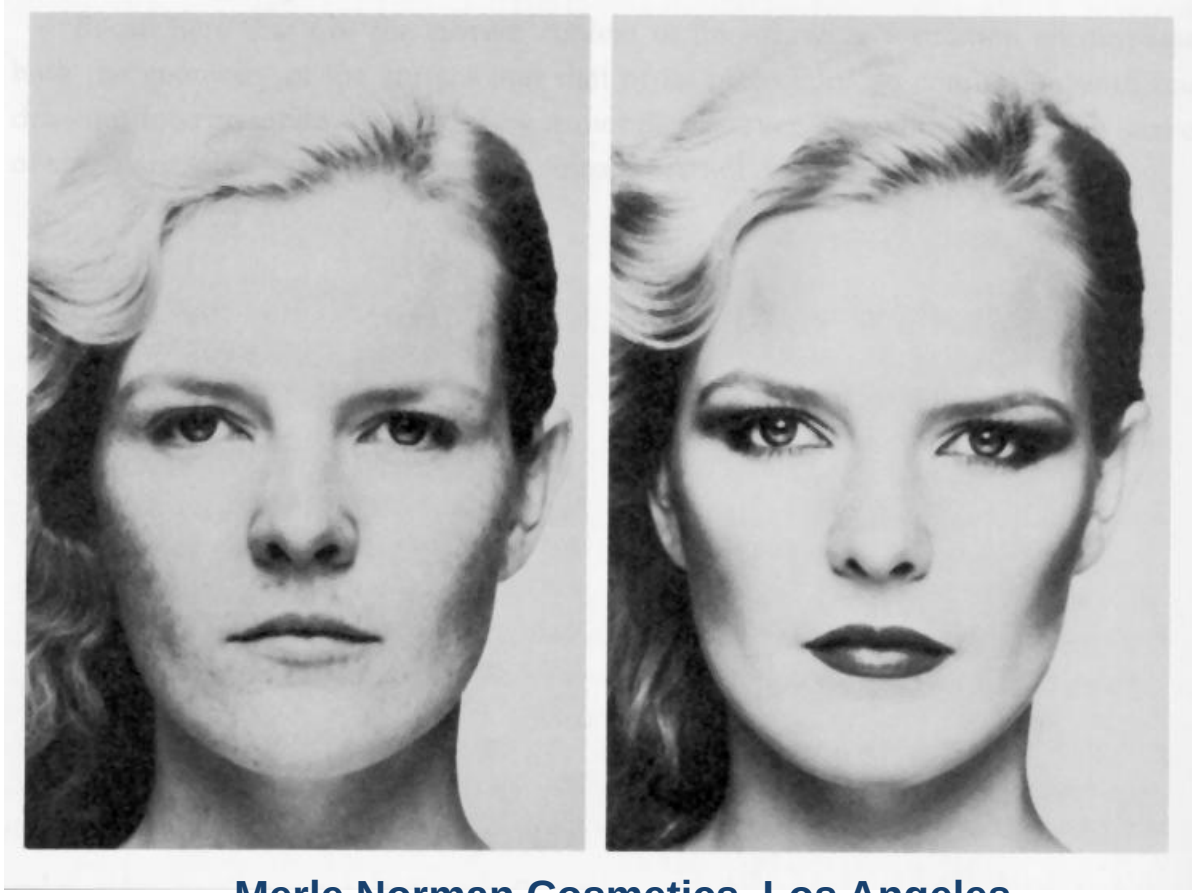
图像中提供了哪些关于三维信息的线索？

令人惊讶的3D壁画艺术

包含了哪些深度线索呢？

三维视觉线索

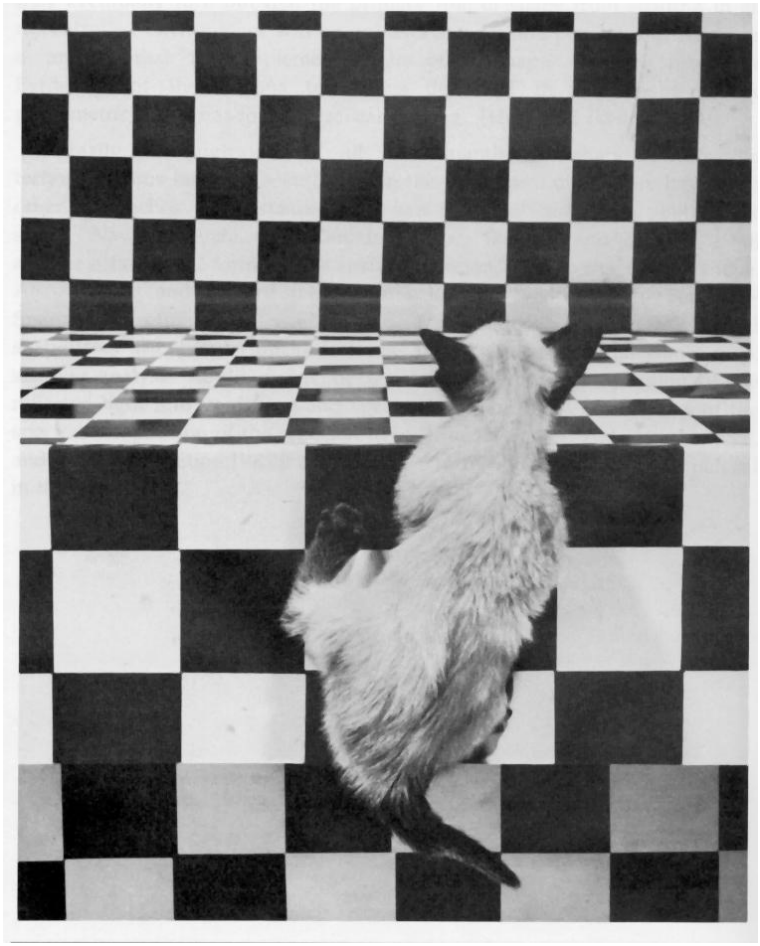
- 阴影



Merle Norman Cosmetics, Los Angeles

三维视觉线索

- 阴影
- **纹理**



***The Visual Cliff*, by William Vandivert, 1960**

三维视觉线索

- 阴影
- 纹理
- **遮挡**

From *The Art of Photography*, Canon

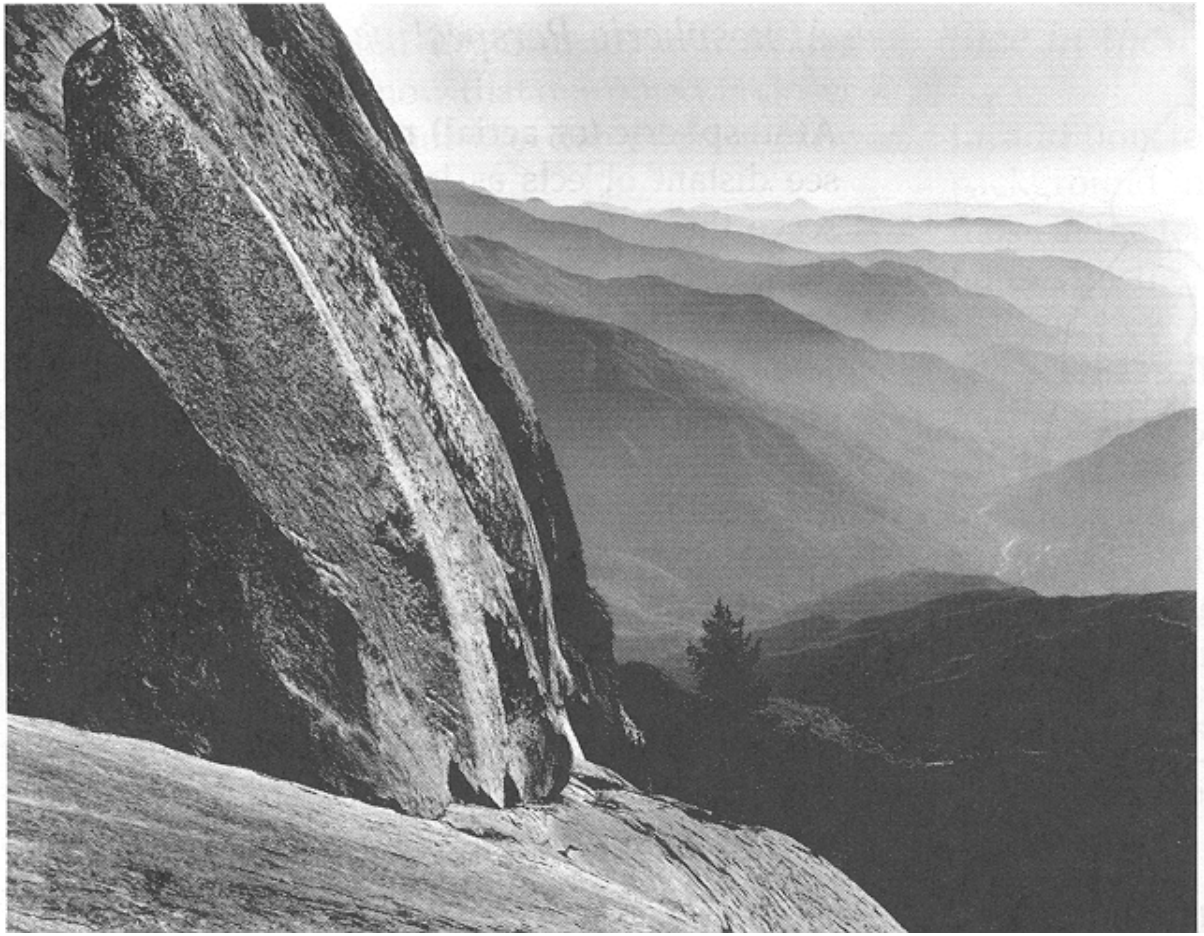
三维视觉线索

- 阴影
- 纹理
- 遮挡
- **运动**



三维视觉线索

- 阴影
- 纹理
- 遮挡
- 运动
- **模糊**



三维视觉线索

- 阴影
- 纹理
- 遮挡
- 运动
- 模糊

- 其他线索
  - 高光
  - 轮廓
  - 对焦
  - ...

Shape From X

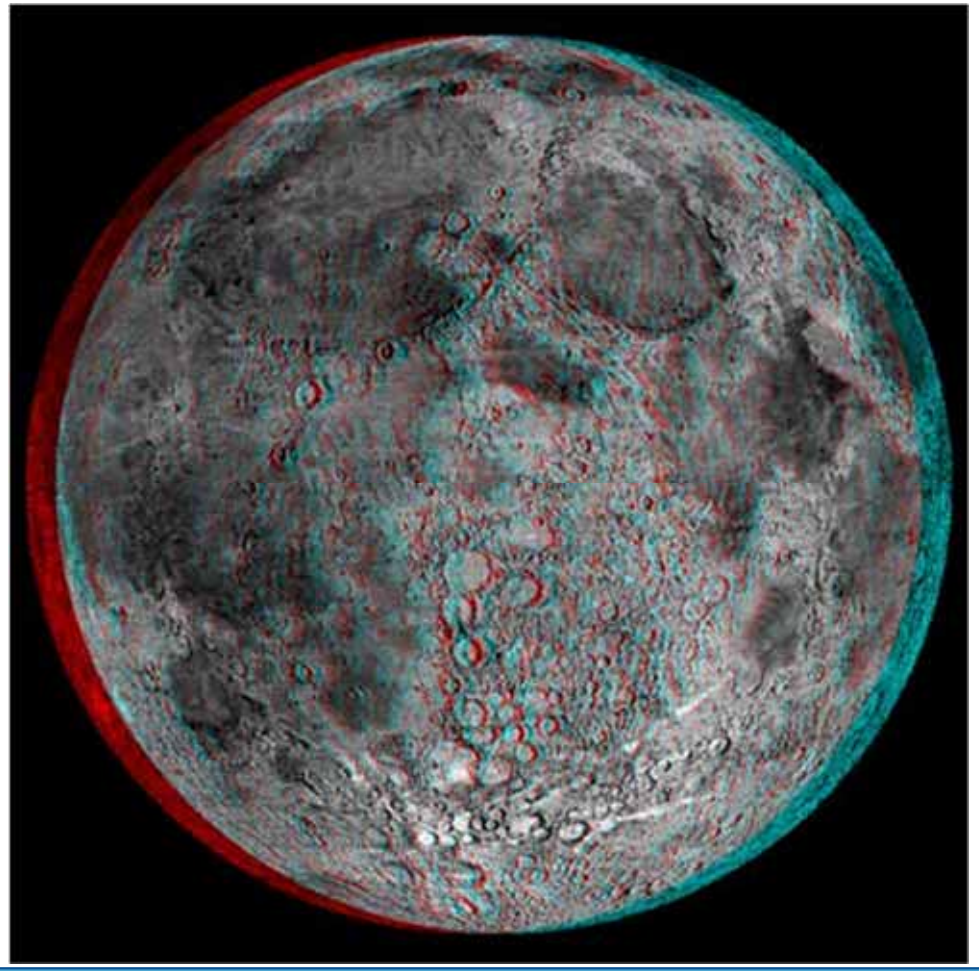
X = 阴影、纹理、遮挡、运动、 ...

三维视觉线索

Two is better than one ?

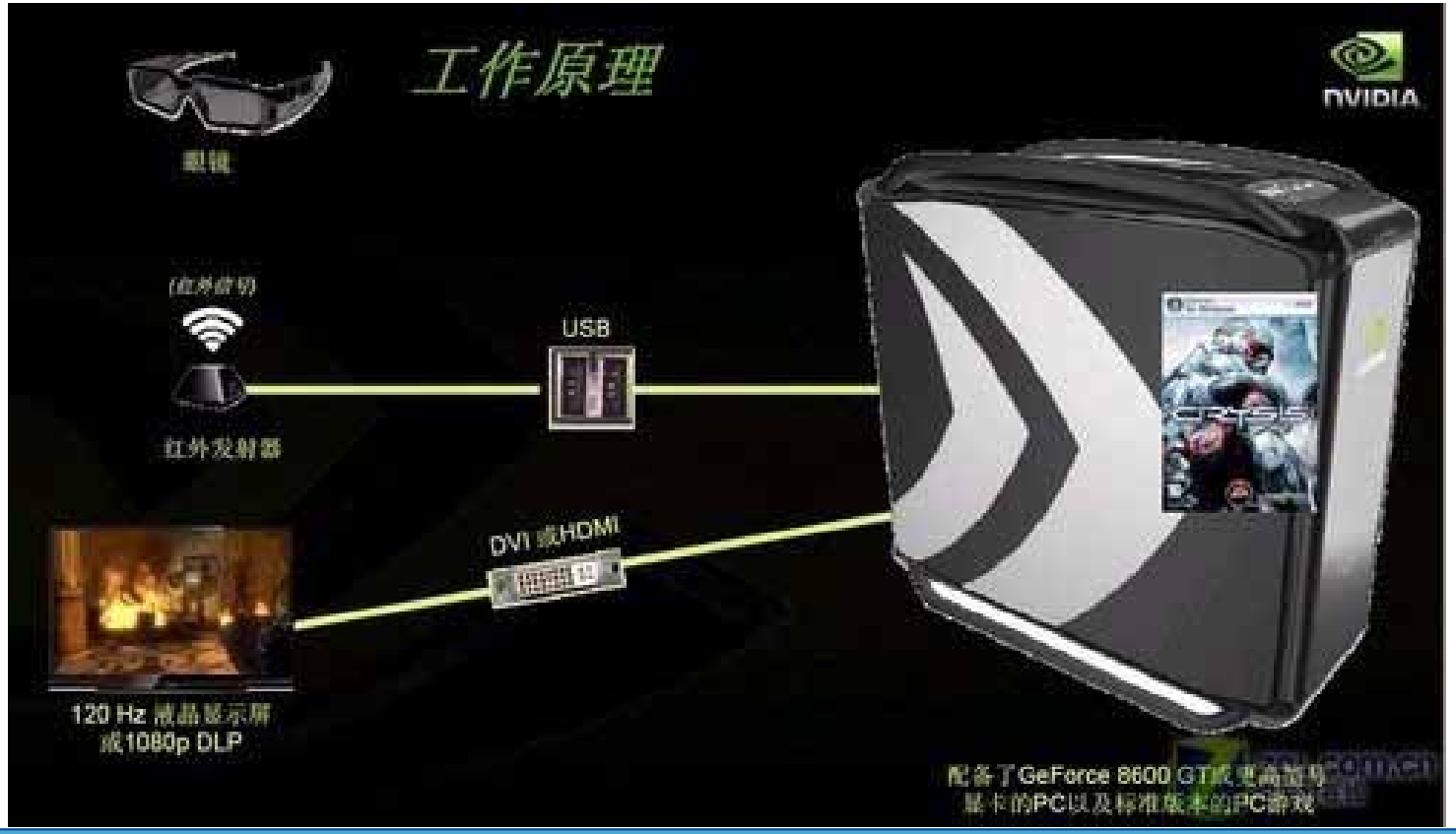
3D电影和3D显示

- 红蓝眼镜
- 偏振光眼镜

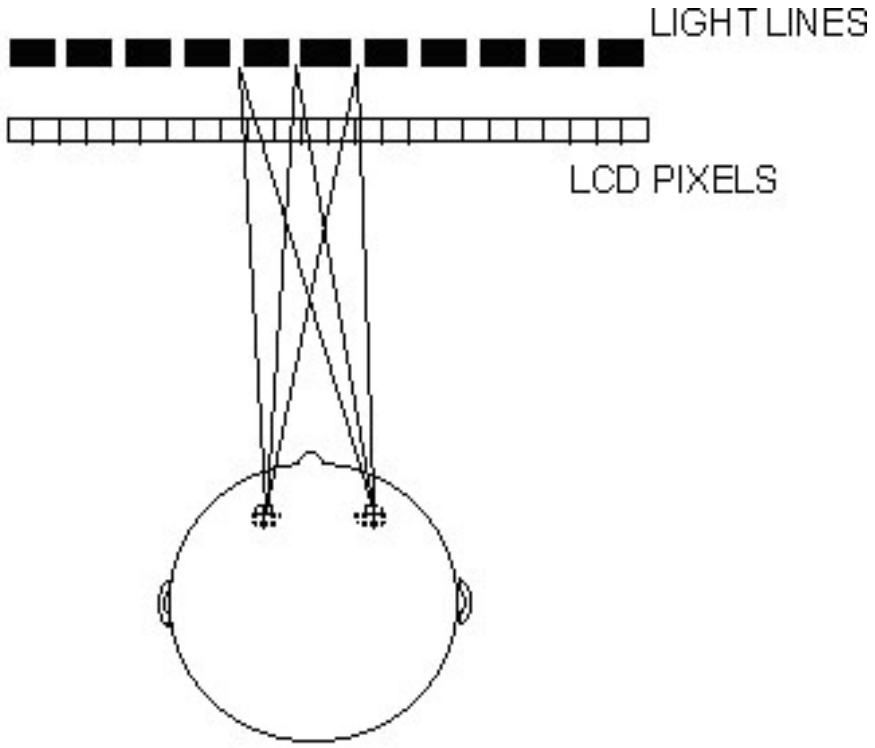


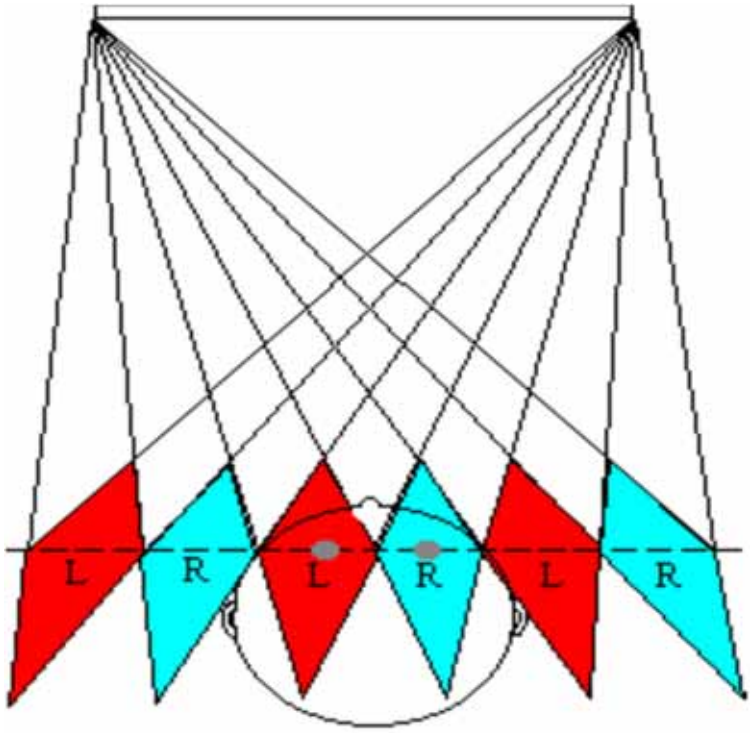
3D电影和3D显示

- 快门眼镜

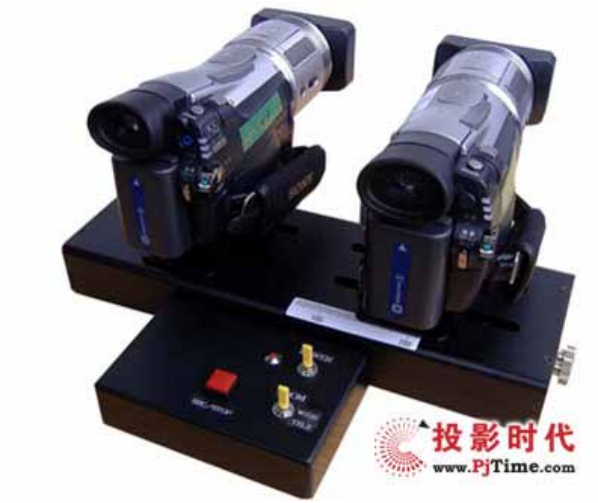


裸眼立体显示系统



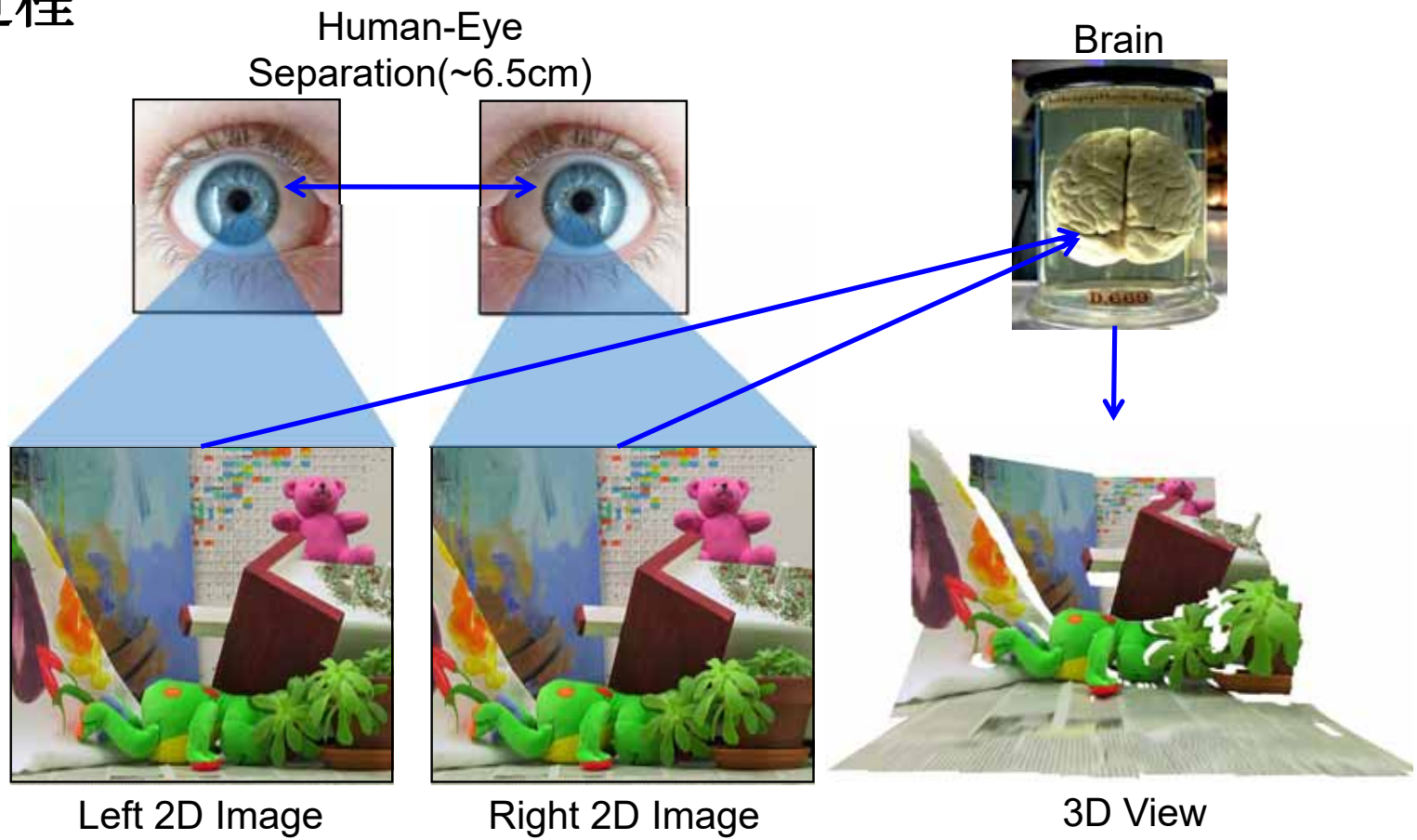


3D拍摄



LG Optimus 3D

3D感知的过程


Human-Eye
Separation(~6.5cm)
Brain
Left 2D Image
Right 2D Image
3D View

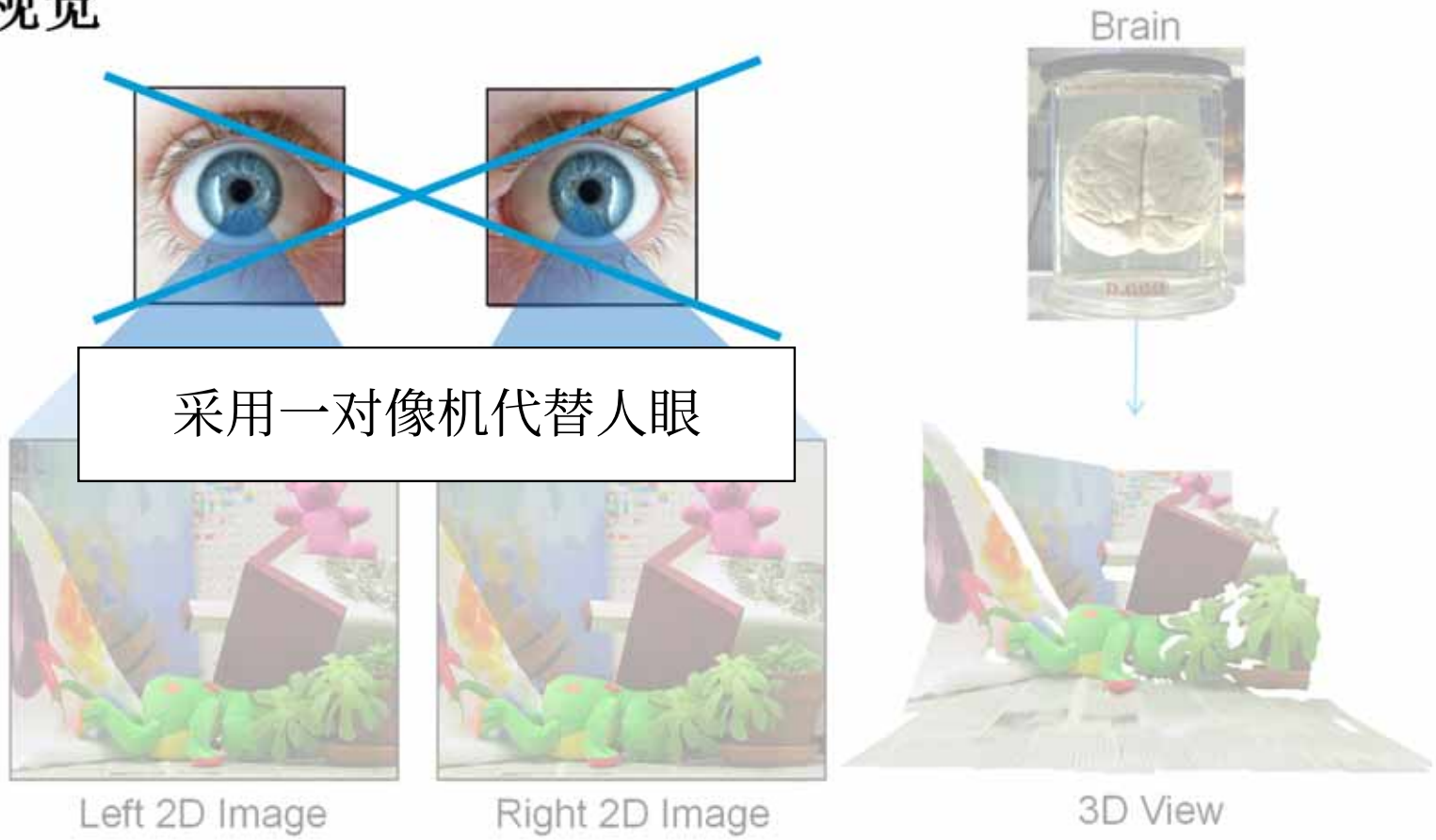
立体图像对的显示

- http://www.well.com/user/jimg/stereo/stereo_list.html

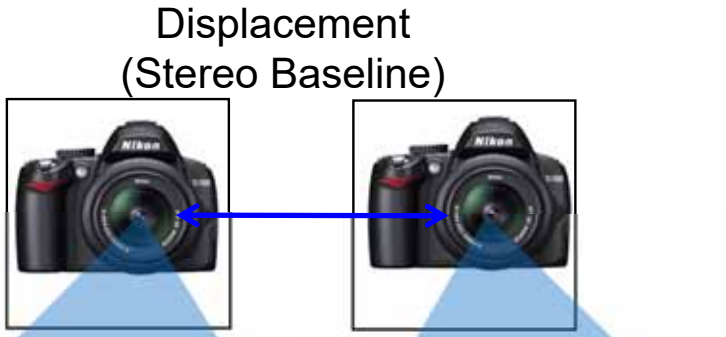
双目如何感受深度？

•左、右图像提供了怎样的深度线索？

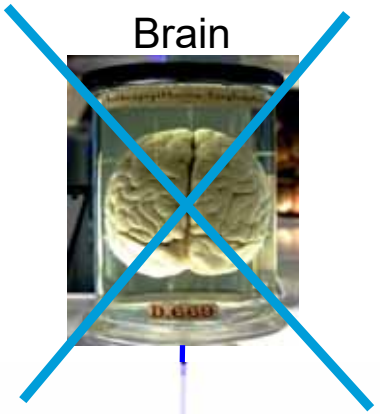
计算立体视觉



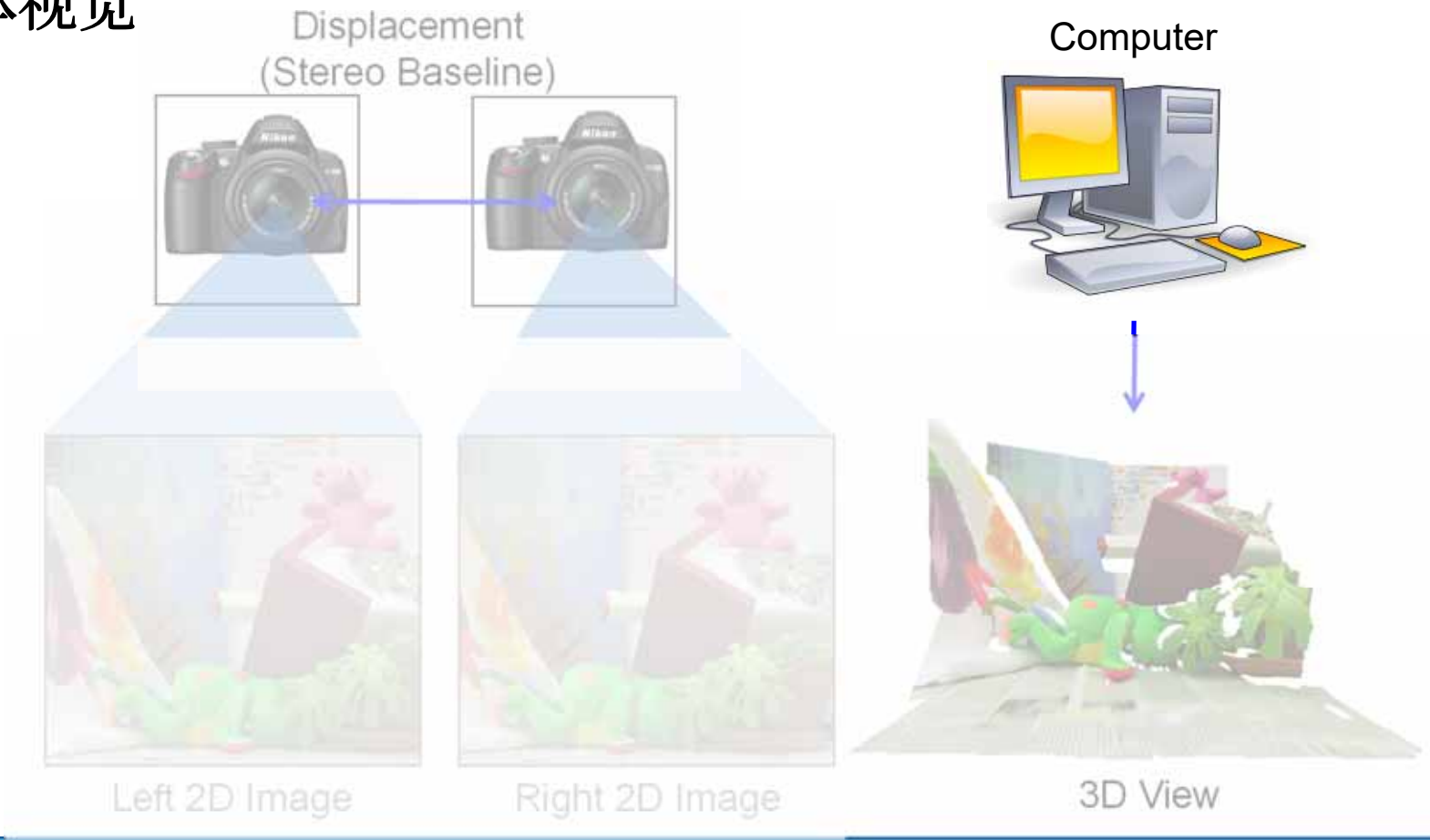
计算立体视觉



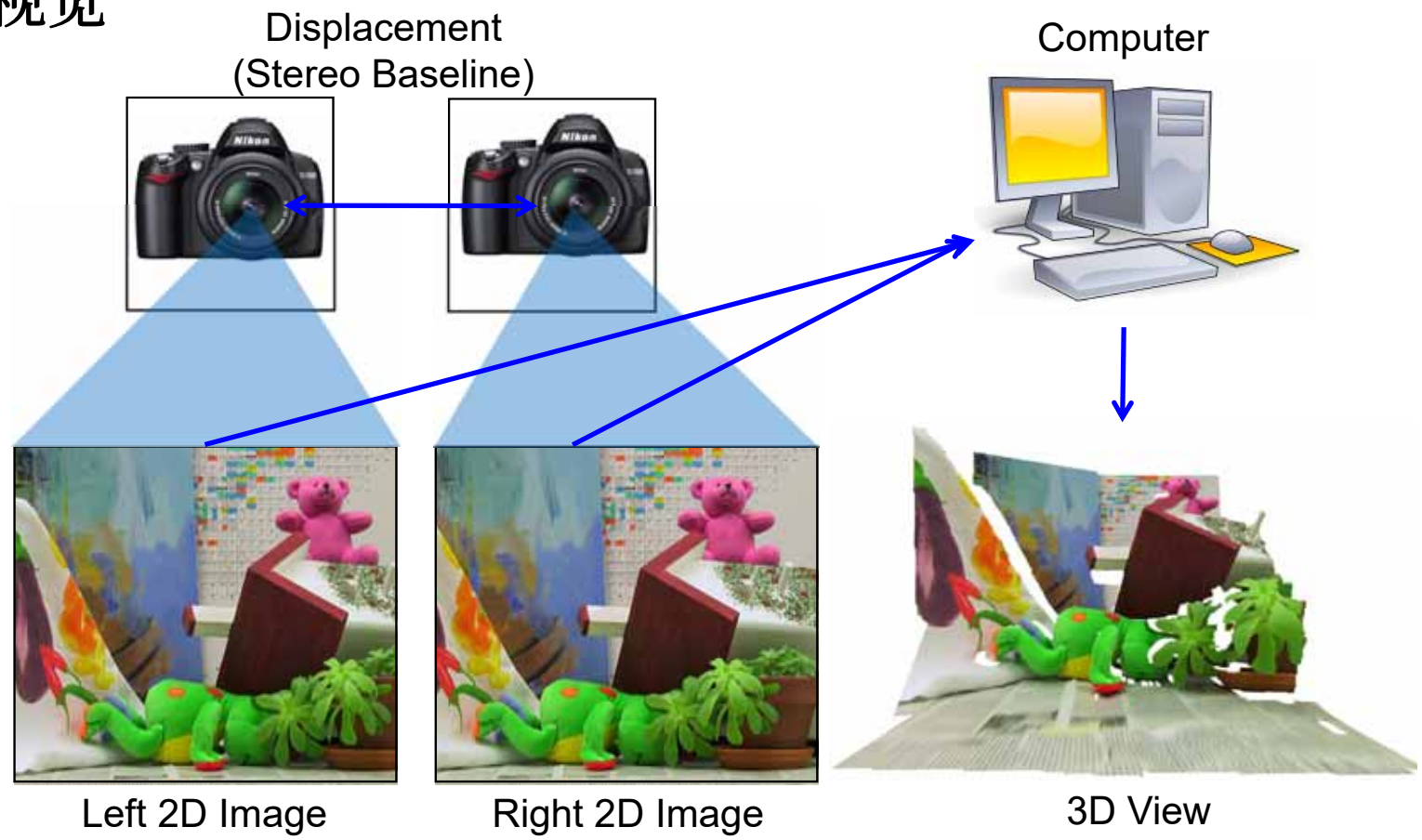
采用一对像机代替人眼



计算立体视觉

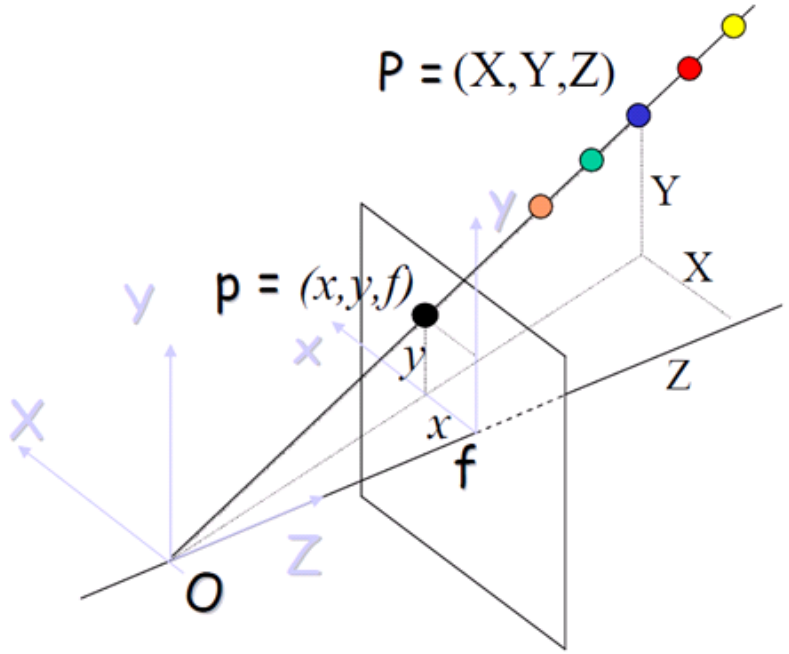

Displacement
(Stereo Baseline)
Computer
Left 2D Image
Right 2D Image
3D View

计算立体视觉


Displacement
(Stereo Baseline)
Computer
Left 2D Image
Right 2D Image
3D View

9.1 双目立体视觉几何

单目视觉

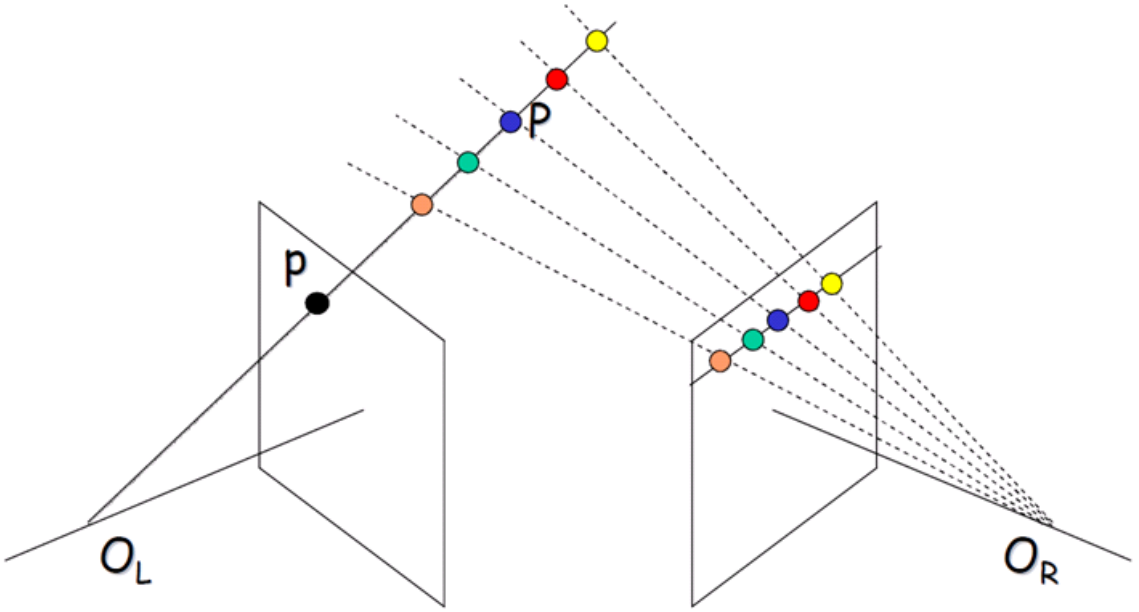


$$x = f\frac{X}{Z} = f\frac{kX}{kZ}$$

$$y = f\frac{Y}{Z} = f\frac{kY}{kZ}$$

- 光线OP上的任意点都投影在图像平面的p点上。
- 深度感知存在歧义

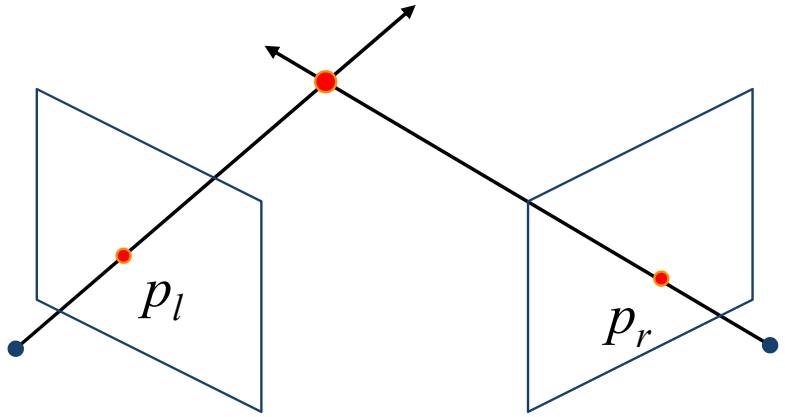
双目视觉



- 第二个像机可解决单目存在的深度歧义
- 可通过三角测量获得深度

三角测量

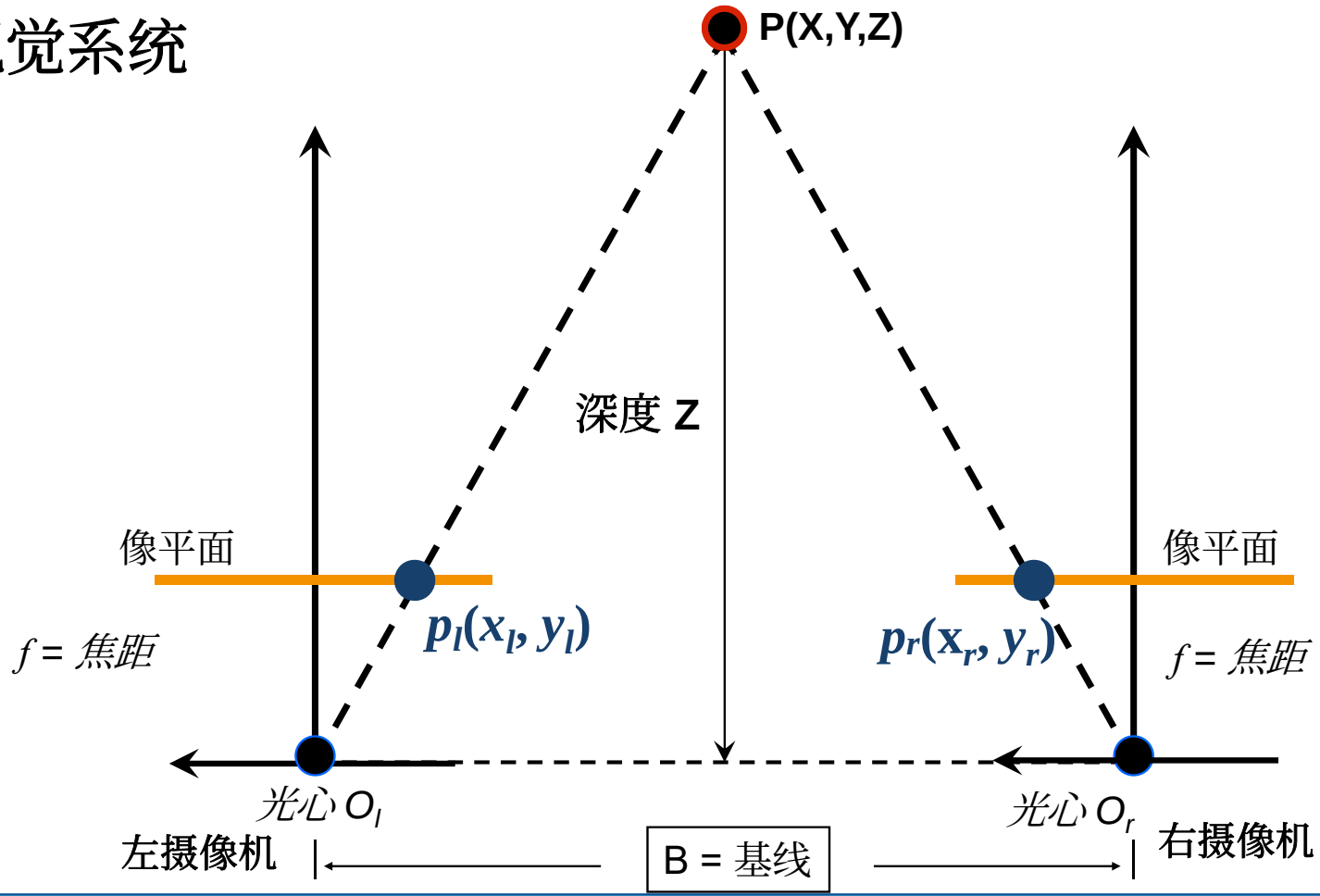
•两直线相交于一点



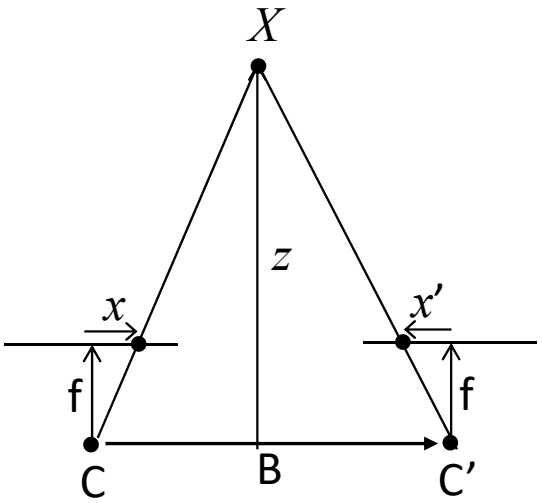
问题：如何根据 p_l 和 p_r 点获取深度？

平行光轴立体视觉系统

- 考虑左、右像机光轴平行的特殊情况



立体视觉基本原理



$$\frac{x - x'}{B} = \frac{f}{z}$$

$$d = x - x'$$

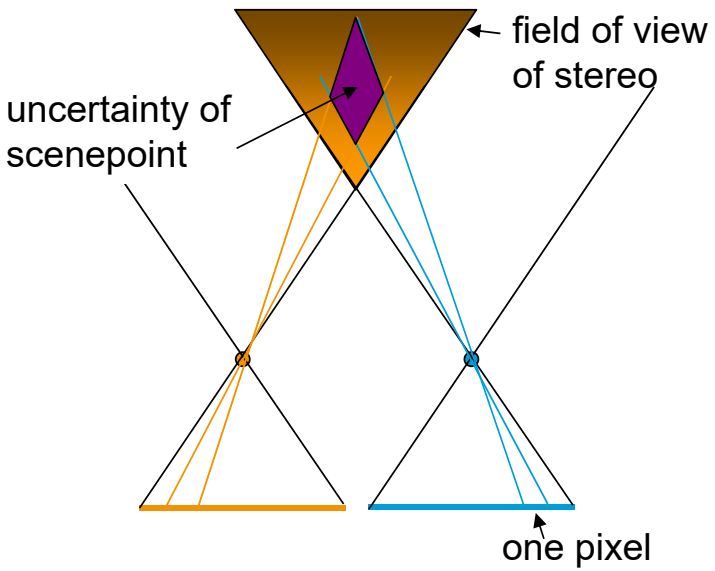
$$z = \frac{B \cdot f}{d}$$

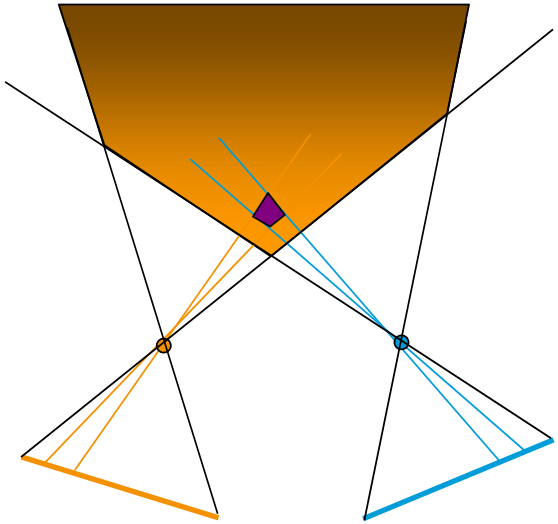
$d = x - x'$ 称为视差

思考：该视差公式隐含了什么条件？

通常的立体视觉系统

•通常的立体视觉系统（包括人类视觉系统）都是采用会聚方式。

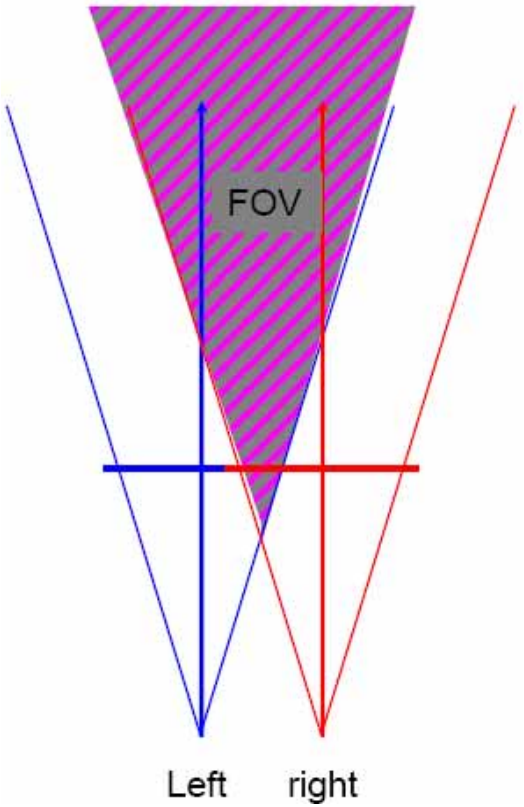




Optical axes of the two cameras need not be parallel

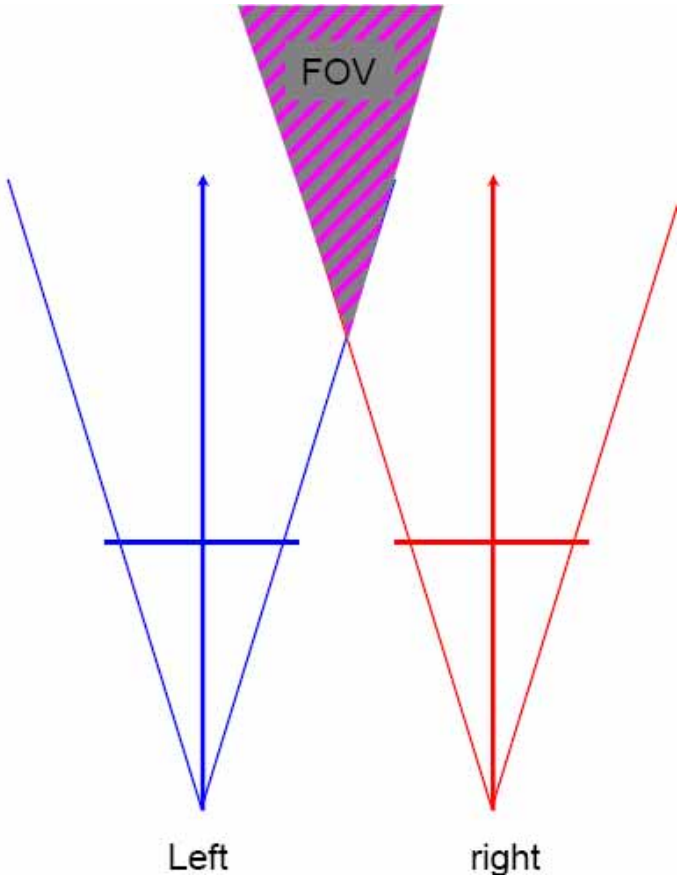
几种立体视觉系统比较

- 平行光轴立体视觉系统
  - 短基线
    - 较大的公共视野区域
    - 深度误差（不确定区域）较大



几种立体视觉系统比较

- 平行光轴立体视觉系统
  - 短基线
    - 较大的公共视野区域
    - 深度误差（不确定区域）较大
  - **长基线**
    - **公共视野区域较小**
    - **深度误差（不确定区域）较小**

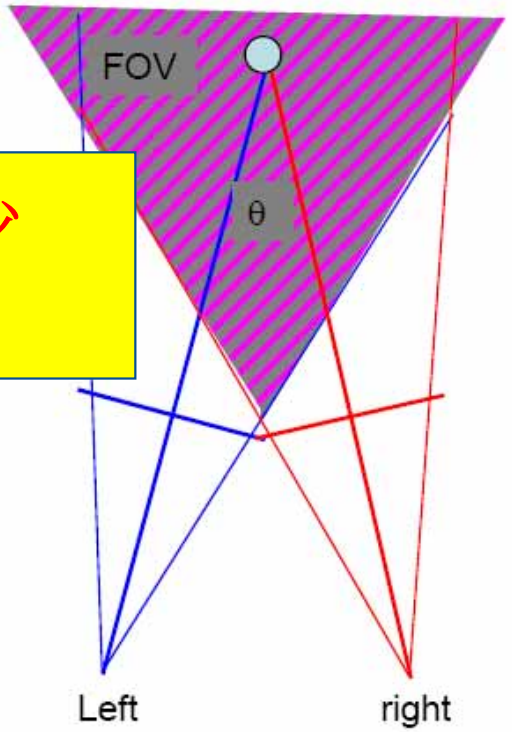


几种立体视觉系统比较

- 平行光轴立体视觉系统
  - 短基线
    - 较大的公共视野区域
    - 深度误差
  - 长基线
    - 公共视野区域较小
    - 深度误差（不确定区域）较小
- 会聚光轴立体视觉系统
  - 会聚角度为ϑ
  - 公共视野区域大

1、基线或聚散度增加导致视野范围减少

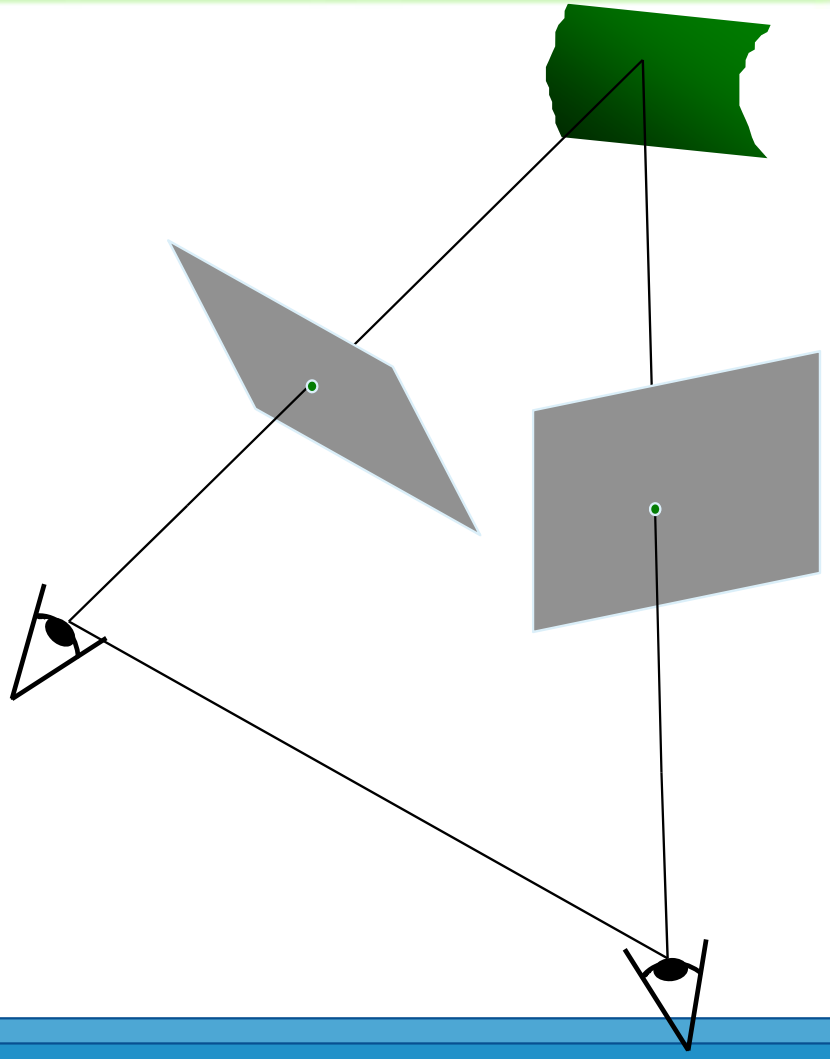
2、基线或聚散度影响三维重建的精度



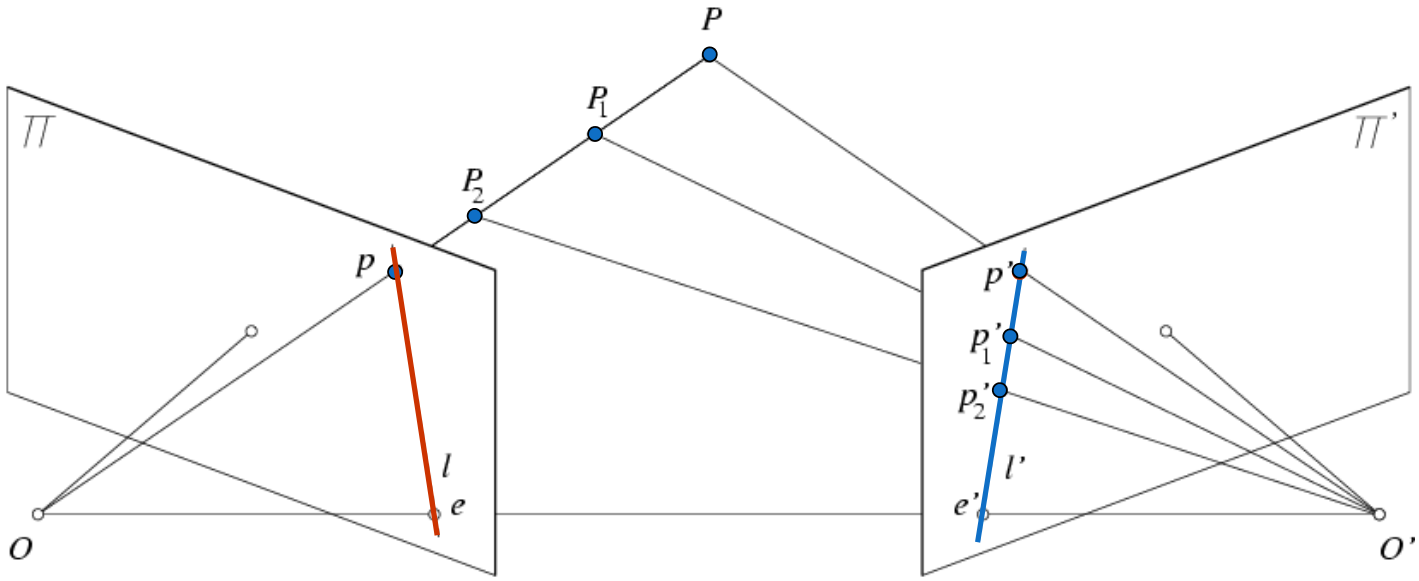
会聚立体视觉系统

- 会聚立体视觉系统能否利用之前的推导结果？

- 立体视觉中重要的几何约束：极线约束



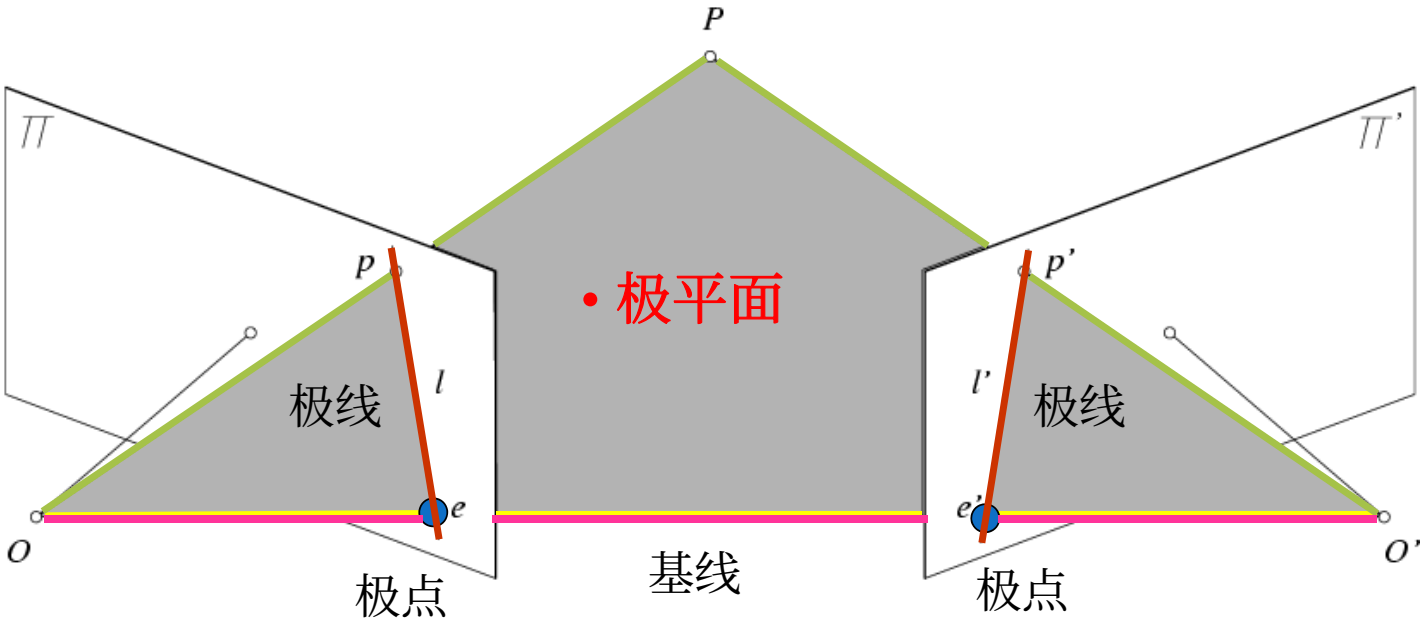
极线约束



•场景点P在视图π中的投影点为像素点p，则在视图π' 中的投影点p'必定满足双目几何约束：

•必定位于图像平面π' 与OPO'平面的交线上。

极线约束



•像素点p和p'是场景点P在两视图上的投影。

•像素点p和p'称为对应点或匹配点。

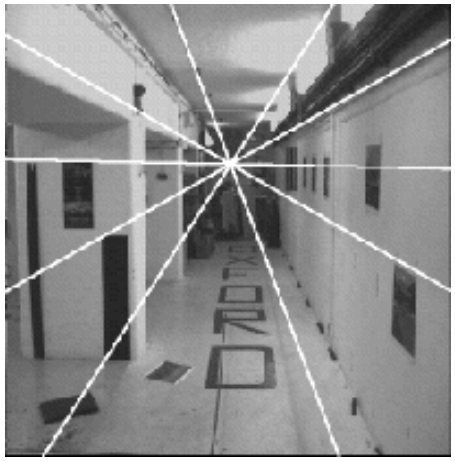
极线约束：对应点必定在极线上！

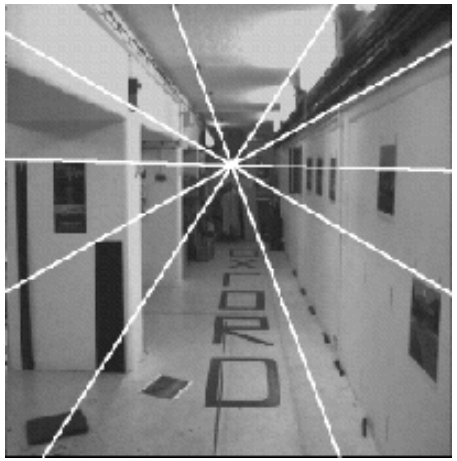
极线几何术语定义

- 基线（Baseline）： 连接像机中心的直线
- 极点（ Epipole ）：基线与图像平面的交点
- 极平面（Epipolar plane）：基线和场景点组成的平面
- 极线（Epipolar line）：极平面与图像平面的交线

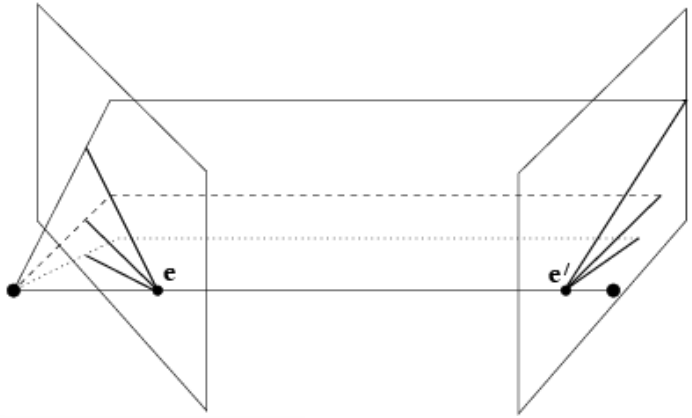
思考：极线约束的意义或用处？

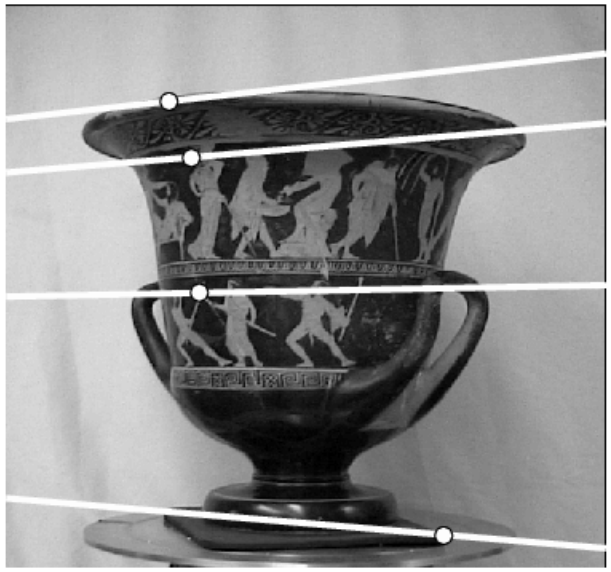
- 所有的极线相交于极点



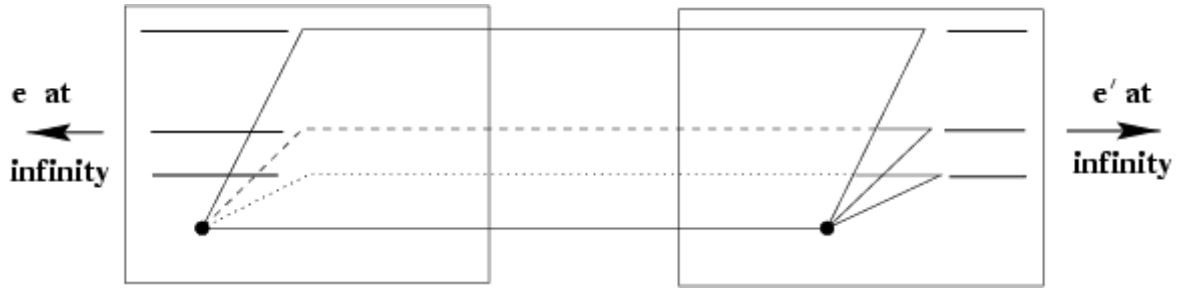


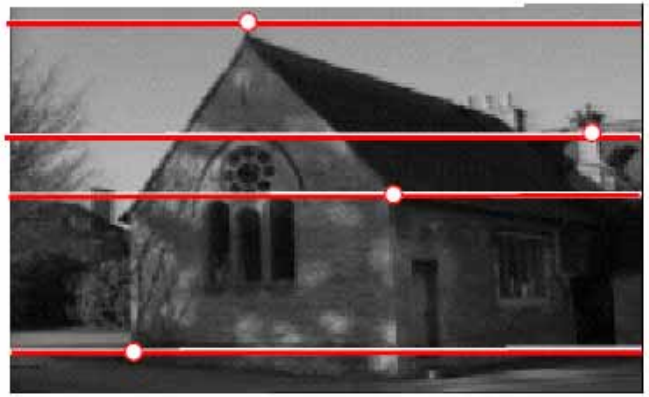
极线约束例



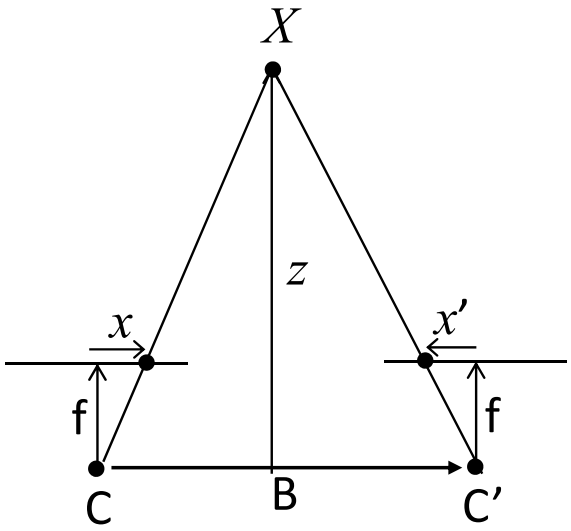


极线约束例





平行光轴的极线约束



- 所有极线水平
- 极点在基线B上的无穷远处

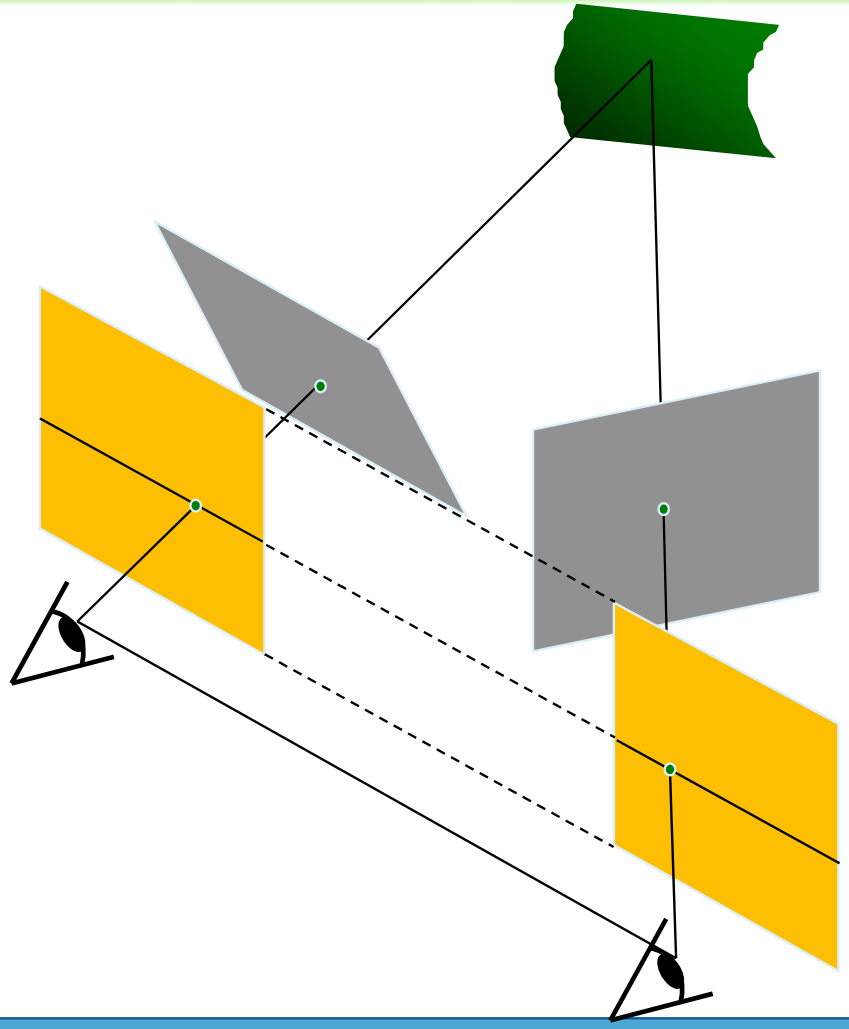
- 左、右图对应极线在同一水平。
- 对应像素点只存在水平坐标差异，不存在垂直坐标差异。
- 即仅有水平视差，垂直视差为0。

- 视差定义：$d = x - x'$

光轴会聚的立体视觉系统

- 光轴会聚的立体视觉系统能否利用平行光轴的推导结果？

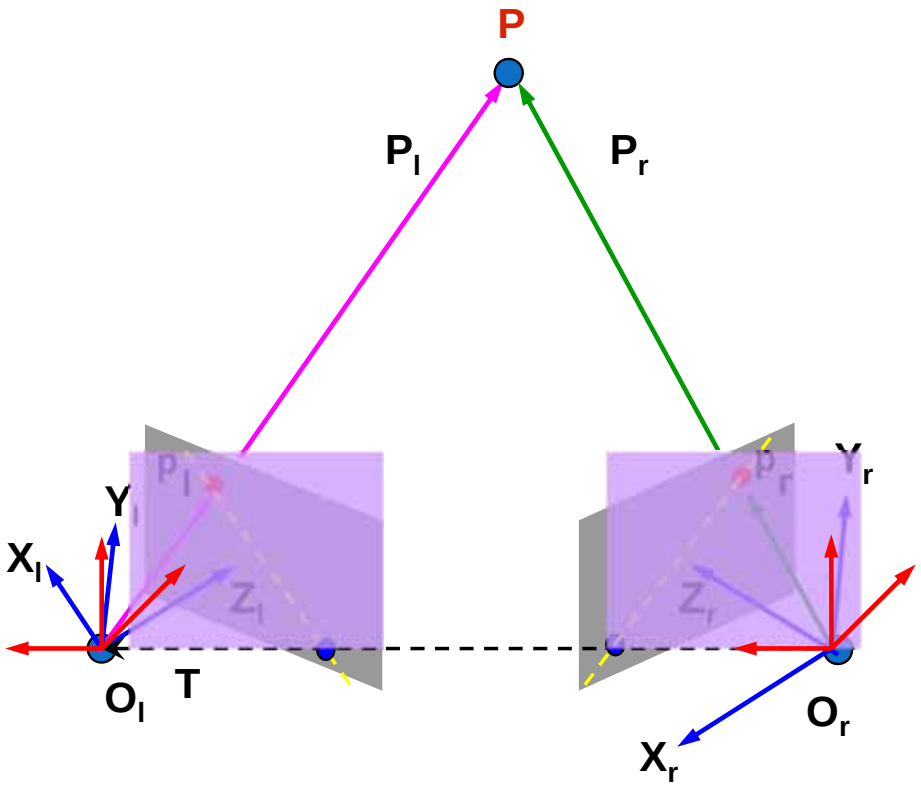
- 立体图像校正
  - 将左、右图像平面都投影到平行于基线的公共平面。
  - 变换后的对应极线处于同一水平线（共线）。




9.2 双目立体图像校正

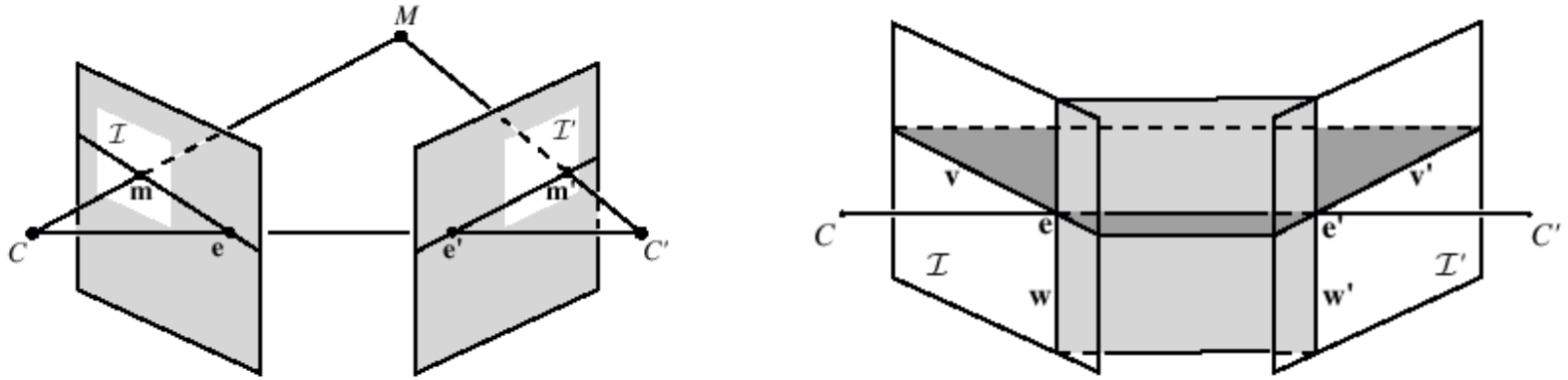
立体图像校正的目的

- 校正的目的：
  - 输入图像通过透视变换使得外极线水平，且共线。
  - 畸变校正，使得成像过程符合小孔成像模型。



立体图像校正步骤

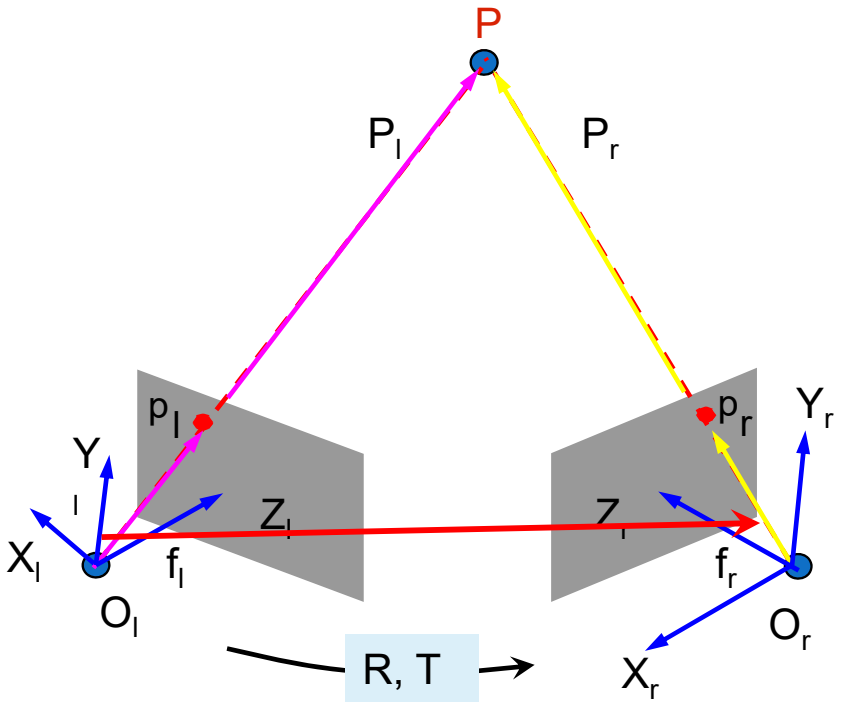
- 校正步骤：
  - 将左右图像平面都投影到平行于基线的公共平面。
  - 图像行像素重采样。
  - 最小化图像畸变。



[Zhang and Loop, MSR-TR-99-21]

立体图像校正

- 立体视觉系统的参数　　（由双目像机标定获得）
  - 内部参数
    - 对于每个像机，其图像坐标系与像机坐标系间的关系。
    - 焦距、光心、畸变系数
  - 外部参数
    - 两像机间的相对位置和方向。
    - 旋转矩阵**R**和平移矢量**T**



立体图像校正

符号定义：

•场景点**P**在左、右像机坐标系下的坐标为：

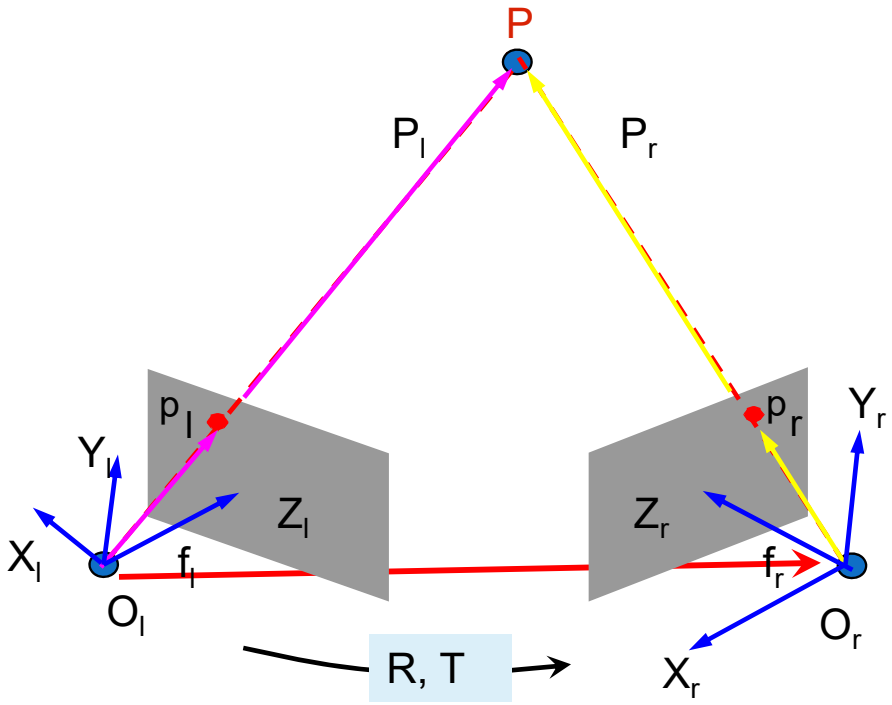
$\mathbf{P}_l = (X_l, Y_l, Z_l)$, $\mathbf{P}_r = (X_r, Y_r, Z_r)$。

•外部参数

•平移矢量 $\mathbf{T}=(\mathbf{O}_r - \mathbf{O}_l)$

•旋转矩阵 $\mathbf{R}$

$$\mathbf{P}_r = \mathbf{R}(\mathbf{P}_l - \mathbf{T})$$



•场景点**P**在左右图像平面上投影：

$\mathbf{p}_l=(x_l, y_l, z_l)$, $\mathbf{p}_r=(x_r, y_r, x_r)$。

•对于所有的像素点有 $z_l = f_l$; $z_r = f_r$

$$\begin{cases} \mathbf{p}_l = \dfrac{f_l}{Z_l}\mathbf{P}_l \\ \mathbf{p}_r = \dfrac{f_r}{Z_r}\mathbf{P}_r \end{cases}$$

立体图像校正

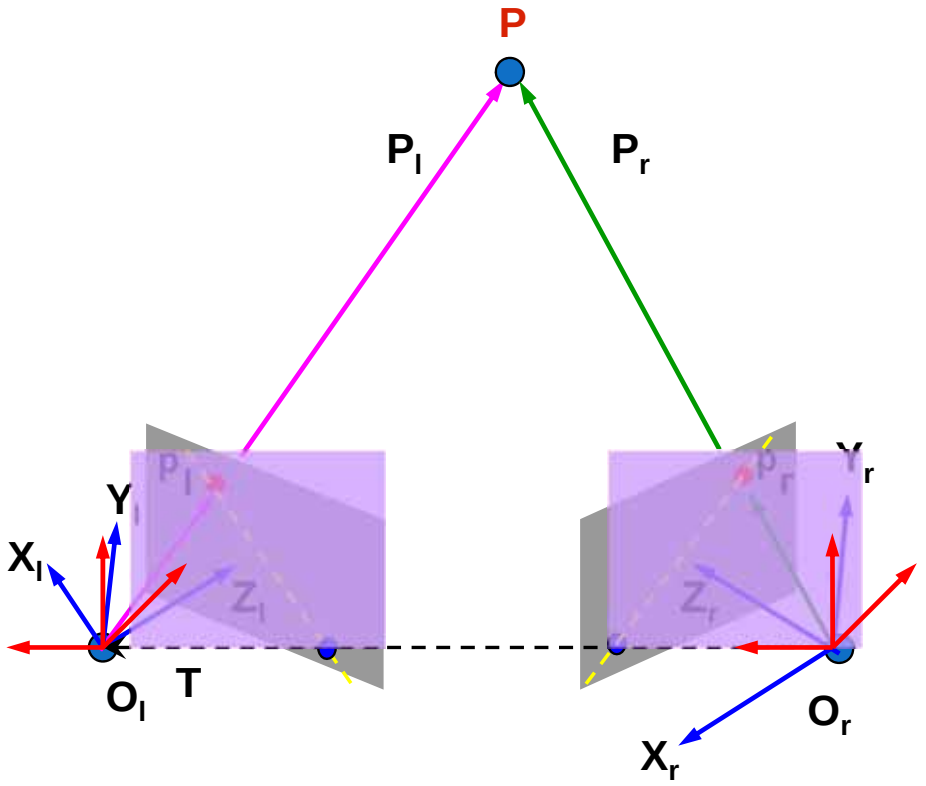
- 同时旋转左右像机使得它们具有相同的x轴。
- 定义左像机的旋转矩阵为$\mathbf{R}_{rect}$。
- 右像机旋转矩阵为 $\mathbf{R}_{rect}\mathbf{R}'$

$\mathbf{R}_{rect}\mathbf{R}'$?

- 立体图像校正后

$$\mathbf{T}' = (B, 0, 0)$$

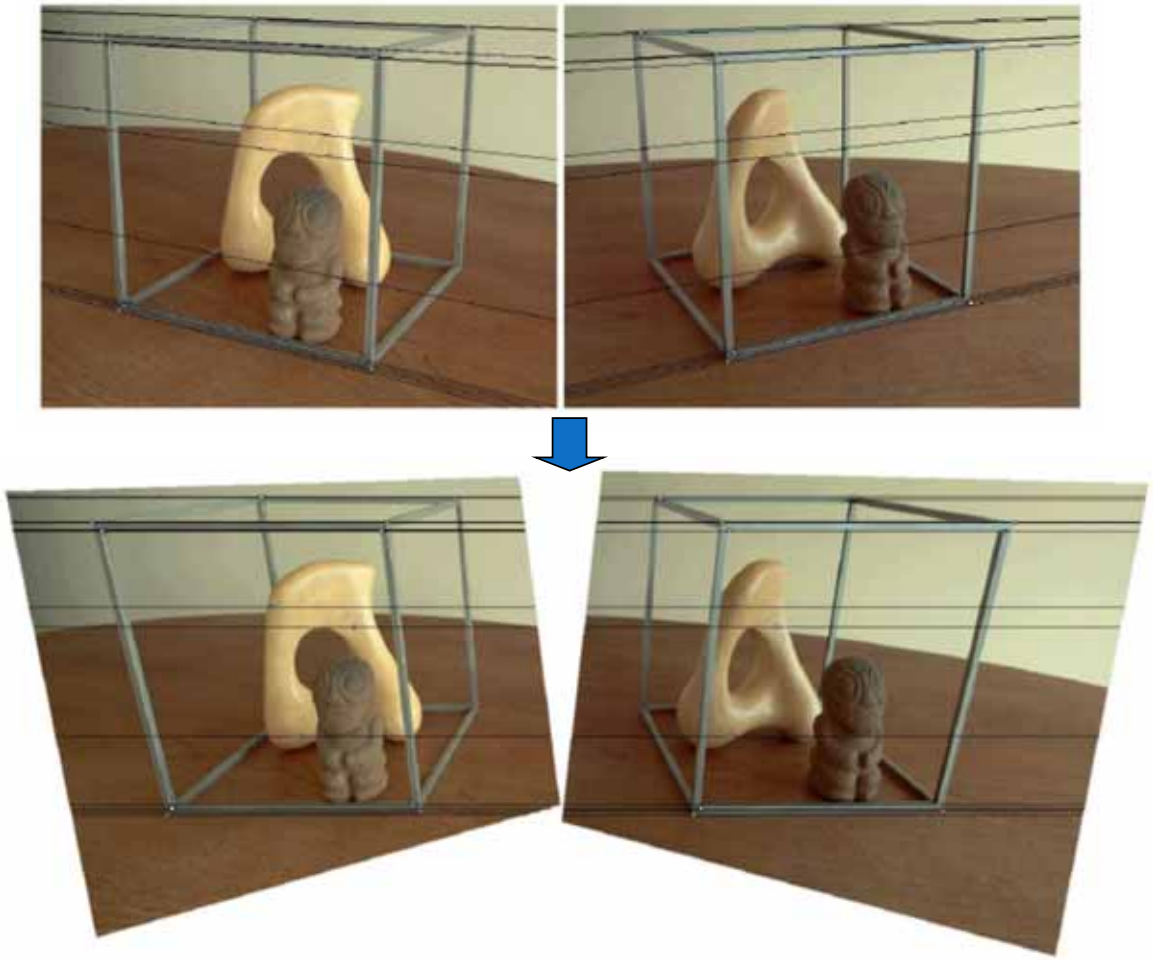
$$\mathbf{P}'_r = \mathbf{P}'_l - \mathbf{T}'$$



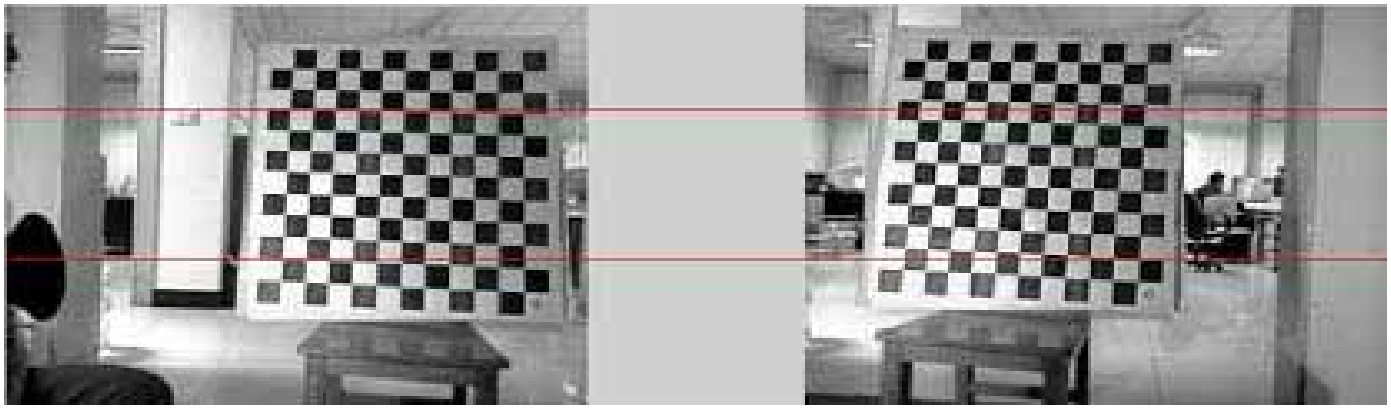
立体图像校正

- 由立体图像标定有，$\mathbf{P}_r = \mathbf{R}\left(\mathbf{P}_l - \mathbf{T}\right)$
- 立体图像校正后，有 $\begin{cases}\mathbf{P}_l' = \mathbf{R}_l\mathbf{P}_l \\ \mathbf{P}_r' = \mathbf{R}_r\mathbf{P}_r\end{cases} \Rightarrow \begin{cases}\mathbf{P}_l = \mathbf{R}_l^{-1}\mathbf{P}_l' \\ \mathbf{P}_r = \mathbf{R}_r^{-1}\mathbf{P}_r'\end{cases}$
- 校正目标：$\mathbf{T}' = \left(B,0,0\right)$，$\mathbf{P}_r' = \mathbf{P}_l' - \mathbf{T}'$
- 即：$\mathbf{R}_r^{-1}\mathbf{P}_r' = \mathbf{R}\left(\mathbf{R}_l^{-1}\mathbf{P}_l' - \mathbf{T}\right) \Rightarrow \mathbf{P}_r' = \mathbf{R}_r\mathbf{R}\mathbf{R}_l^{-1}\mathbf{P}' - \mathbf{R}_r\mathbf{R}\mathbf{T}$
- 可得 $\mathbf{R}_r\mathbf{R}\mathbf{R}_l^{-1} = \mathbf{I} \Rightarrow \mathbf{R}_r = \mathbf{R}_l\mathbf{R}^{-1}$
- 由于R为正交矩阵，有 $\mathbf{R}_r = \mathbf{R}_l\mathbf{R}'$

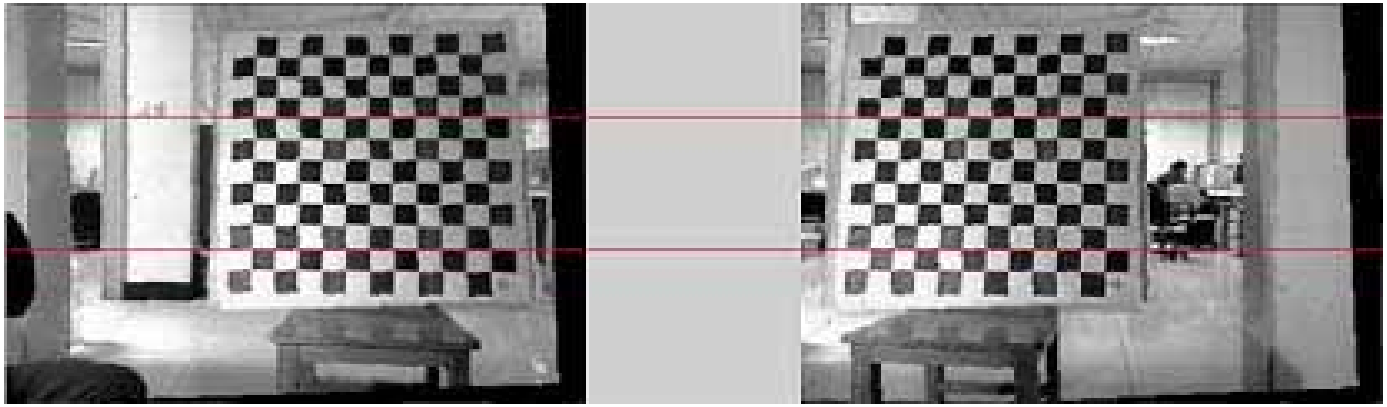
立体图像校正实例



立体图像校正实例



Stereo pairs before rectification

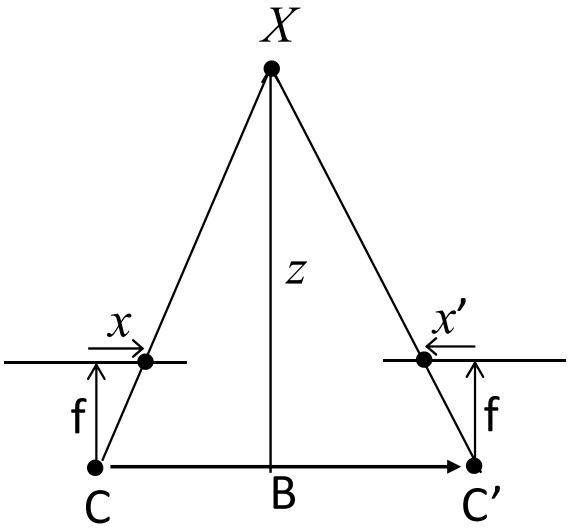


Stereo pairs after rectification

立体图像校正实例

立体匹配——对应点搜索

- 平行光轴的双目立体视觉



$$z = \frac{B \cdot f}{d}$$

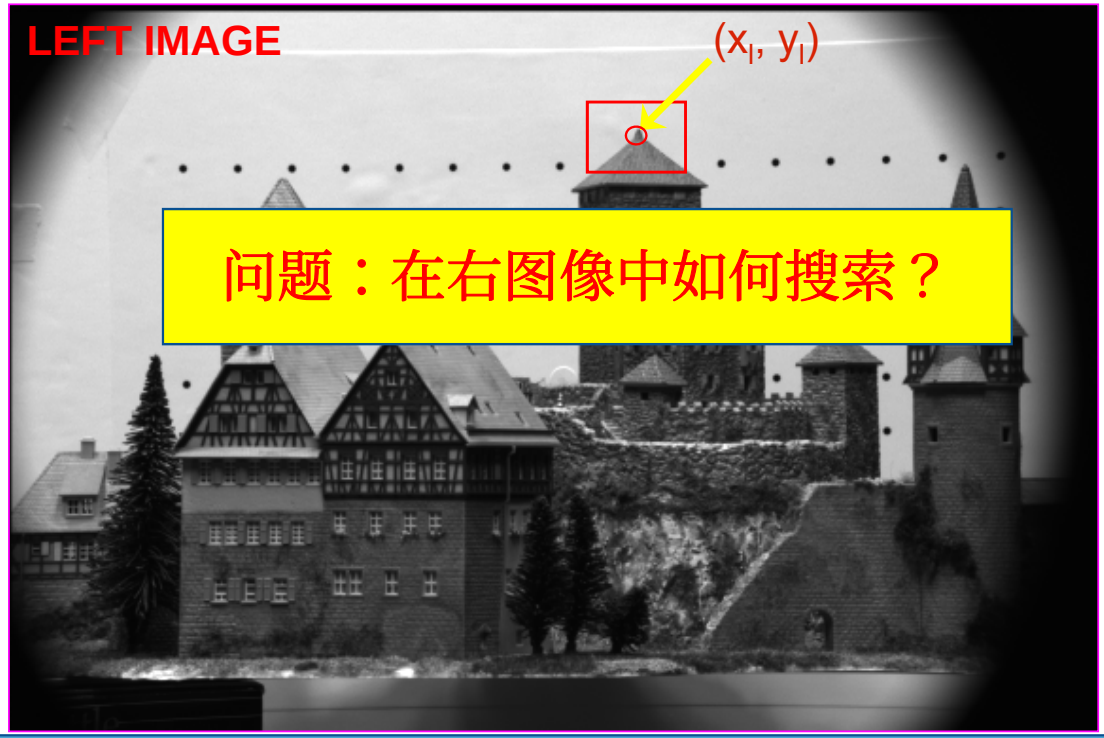
$$d = x - x'$$

问题：如何确定对应像素点？

9.3 立体匹配

立体匹配——对应点搜索

•立体匹配的过程：为左图像的每个像素点(x_l, y_l)，在右图像中搜索对应点。



立体匹配——对应点搜索

- 极线约束的意义：
  - 将对应点搜索由原来的二维平面上搜索变为在极线上的一维搜索。
- 对于校正后的立体图像：
  - 在图像水平方向搜索

立体匹配——对应点搜索

- 比较右图对应极线上的每个像素，寻找最相似的像素作为对应点。即在右图同一水平方向上的视差范围内搜索。

立体匹配的基元

- 匹配基元：参与立体匹配，计算相似测度的基本单元
- 常用的匹配基元：
  - 像素
    - 单个像素存在相似性歧义
    - 需结合一行或整幅图像的所有像素同时完成匹配
  - 局部窗口区域
    - 具有较好的局部独特性
    - 隐含假定：窗口内所有像素应能表征中心像素
  - 特征
    - 具有较好的独特性
    - 稀疏且不均匀分布

立体匹配分类

- 根据立体匹配过程中涉及的像素范围，可分为：
  - **局部立体匹配**
    - 通常以基于局部窗口的立体匹配方法为主。
    - 匹配基元：局部窗口
  - **全局立体匹配**
    - 匹配过程中，求解一行或整幅图像中所有像素的相似测度和最大/最小。
    - 匹配基元：像素

立体匹配分类

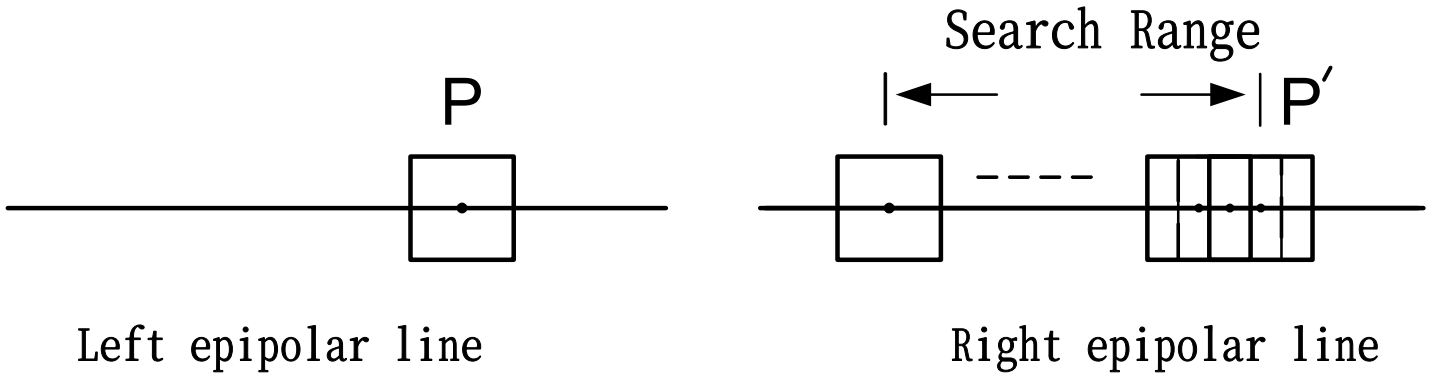
- 根据立体匹配过程中采用的匹配基元，可分为：
  - **致密匹配**
    - 搜索每个像素的对应点，构建致密视差图
    - 匹配基元为像素
  - **稀疏匹配**
    - 仅为特征搜索对应点，构建稀疏的视差图.
    - 匹配基元为特征。

9.3.1 基于局部窗口的立体匹配

基于局部窗口的对应性搜索

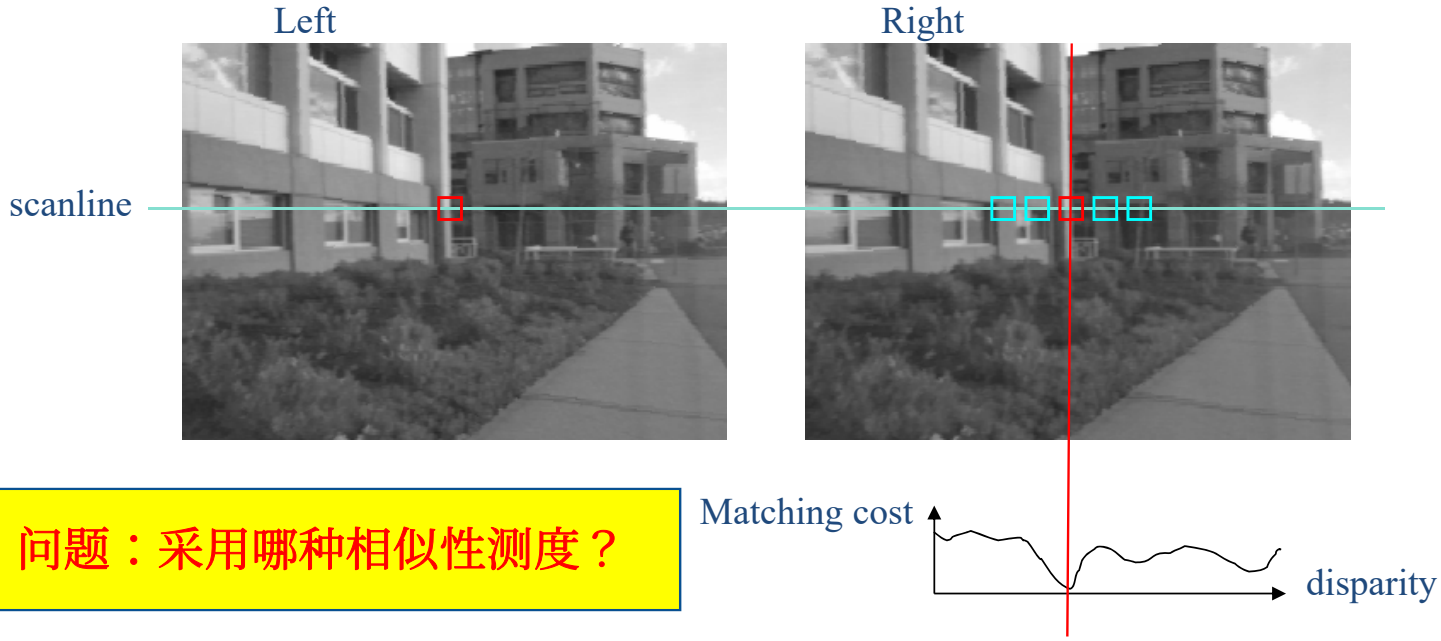
•以基准图的待匹配点为中心创建一个窗口，以在对准图中对应外极线上某一像素点为中心创建同样大小的滑动窗口，窗口内相邻像素的亮度值分布来表征中心像素。

•比较对准图中每个滑动窗口内容与基准图参考窗口内容的相似程度。



基于局部窗口的对应性搜索

对于已校正的双目立体图像对，则在扫描线上搜索。



问题：采用哪种相似性测度？

相似性测度

- 常用的相似性测度包括：
  - 距离测度：L1距离、L2距离、...
  - 相关系数：NCC、ZNCC
  - 非参数化测度：RANK、Census

距离测度

•像素亮度差的绝对值和(Sum of Absolute Differences, SAD)：

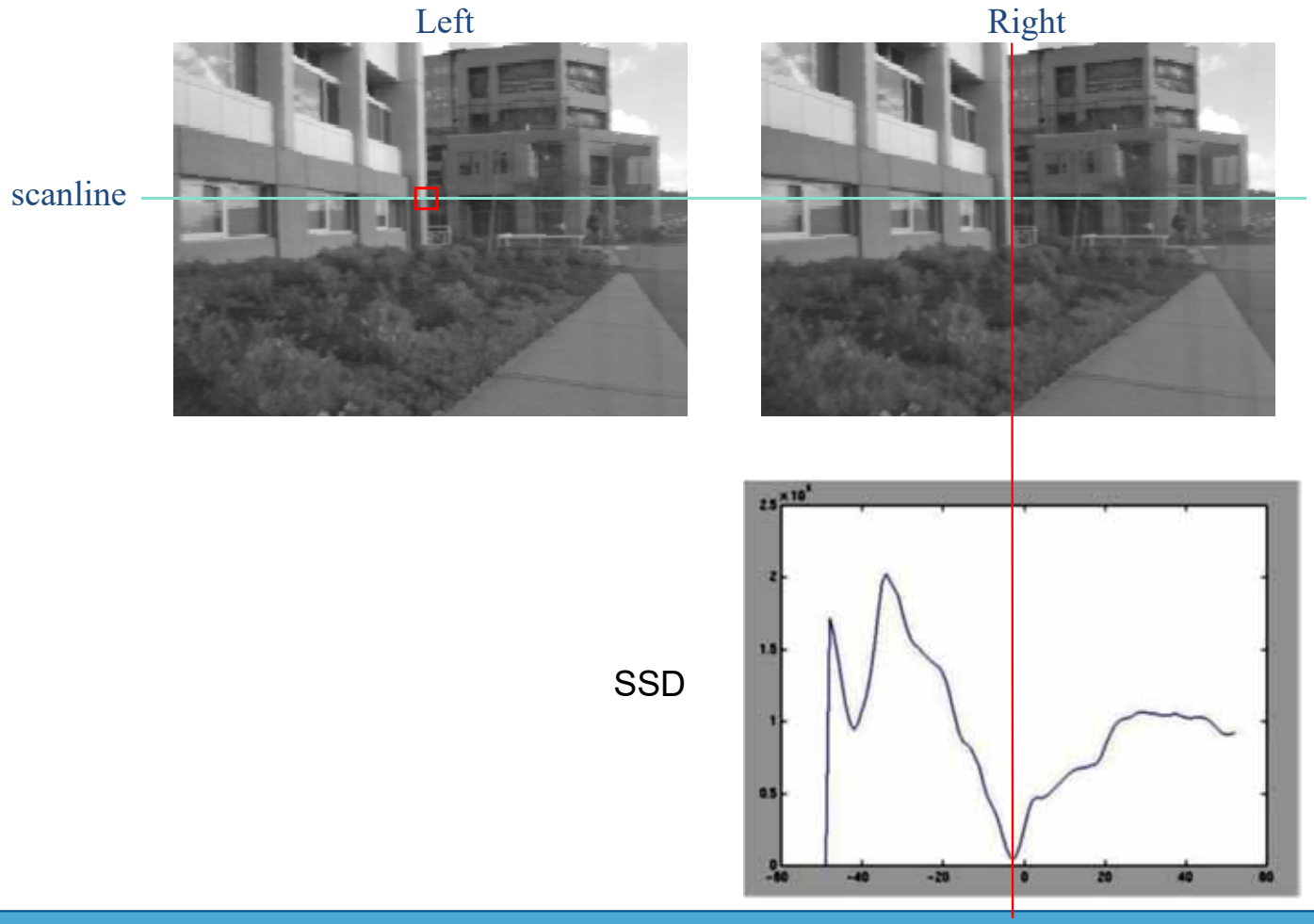
$$C_{SAD} = \sum_{(u,v)\in W_p} \left| I_l(u,v) - I_r(u+d,v) \right|$$

•像素亮度差的绝对平方和(Sum of Squared Differences, SSD)：

$$C_{SSD} = \sum_{(u,v)\in W_p} \left[I_l(u,v) - I_r(u+d,v) \right]^2$$

问题：距离测度越小，相似性越 ？

距离测度



相关系数

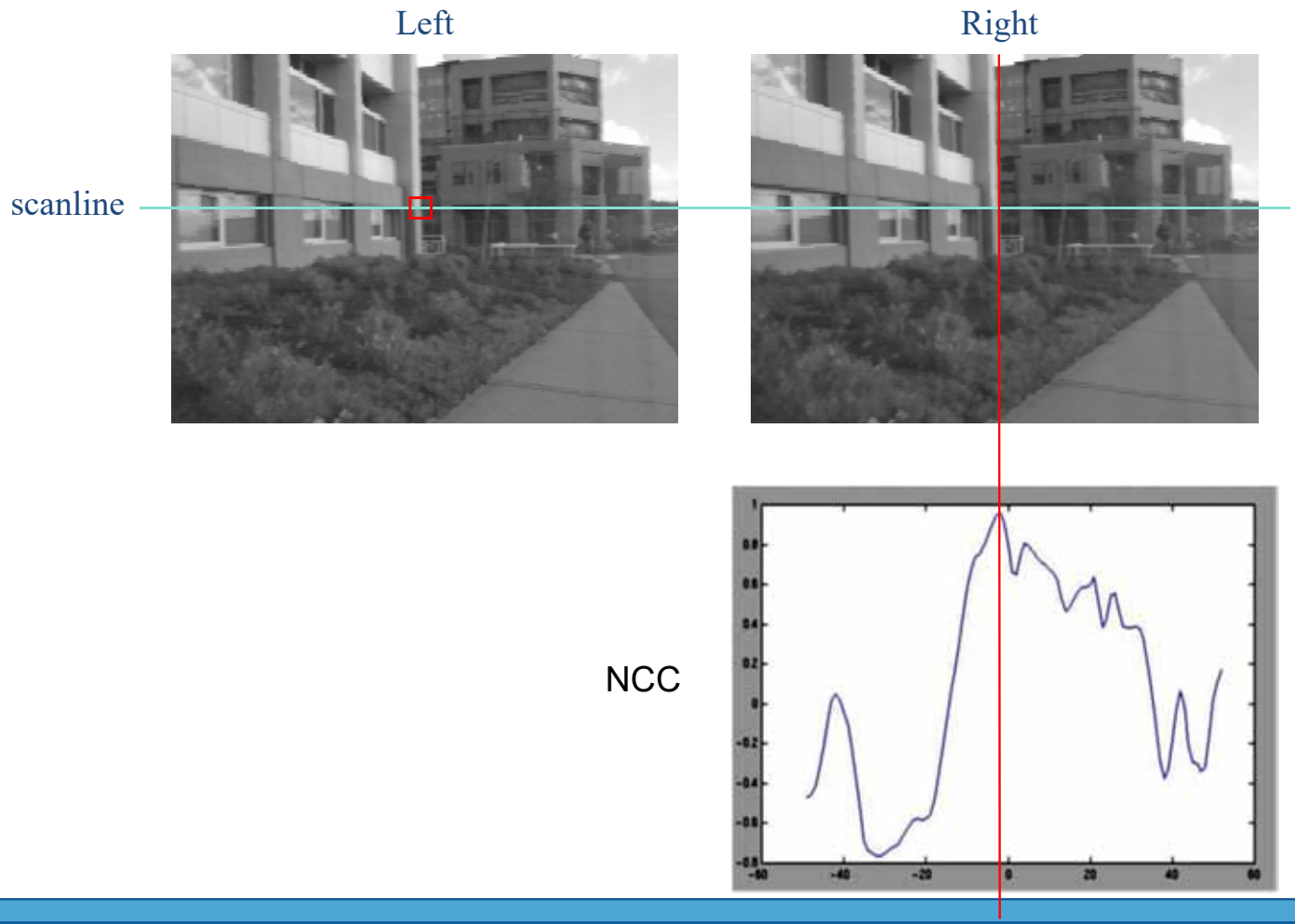
•归一化互相关(Normalized Cross-Correlation, NCC)

$$C_{NCC}=\frac{\sum_{(u,v)\in W_p} I_l(u,v)\cdot I_r(u+d,v)}{\sqrt{\sum_{(u,v)\in W_p} I_l{}^2(u,v)\cdot \sum_{(u,v)\in W_p} I_r{}^2(u+d,v)}}$$

$$C_{ZNCC}=\frac{\sum_{(u,v)\in W_p}\left(I_l(u,v)-\overline{I_l}\right)\cdot\left(I_r(u+d,v)-\overline{I_r}\right)}{\sqrt{\sum_{(u,v)\in W_p}\left(I_l(u,v)-\overline{I_l}\right)^2\cdot\sum_{(u,v)\in W_p}\left(I_r(u+d,v)-\overline{I_r}\right)^2}}$$

问题：相关系数越小，相似性越 ？

相关系数



非参数化测度

•RANK：

$$C_{Rank} = \sum_{(u,v)\in W_p} \left(I_l'(u,v) - I_r'(u+d,v) \right)$$

$$I_k'(u,v) = \sum_{(m,n)\in W_p} I_k(m,n) < I_k(u,v)$$

$I_k'(u,v)$ 统计匹配窗口内像素亮度值小于中心像素的个数。

非参数化测度

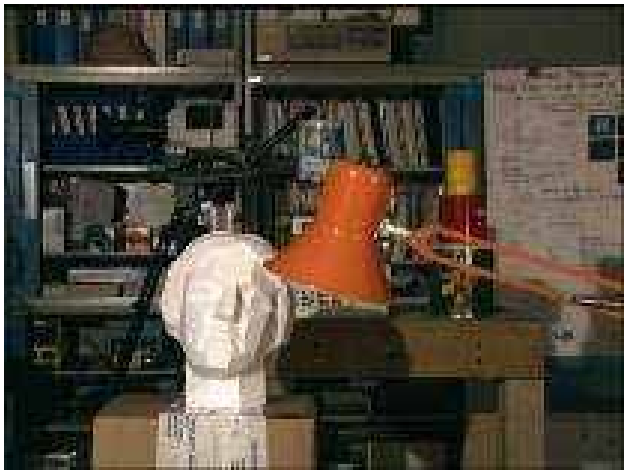
•Census： $$C_{Census} = \sum_{(u,v)\in W_p} HAMMING\left(I_l'(u,v), I_R'(u+d,v)\right)$$

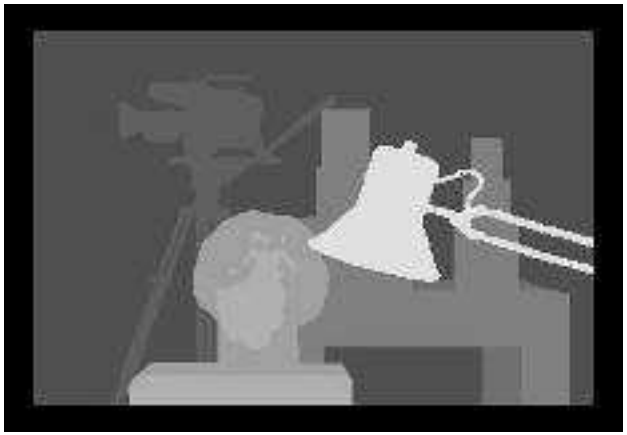
$$I_k'(u,v) = BITSTRING_{(m,n)\in Wp}\left(I_k(m,n) < I_k(u,v)\right)$$

是将匹配窗口内像素按一定顺序映射为二进制位串，如像素亮度值大于中心像素则映射为1，反之为0。然后比较左、右匹配窗口二进制位串的汉明距离。

问题：RANK和Census区别？

立体匹配测试图

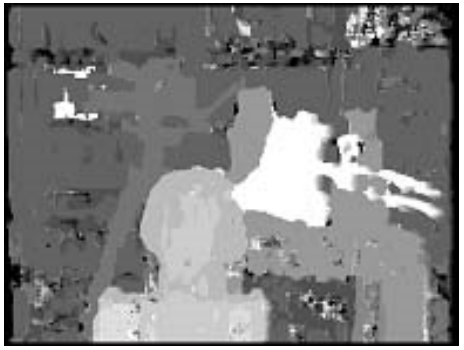




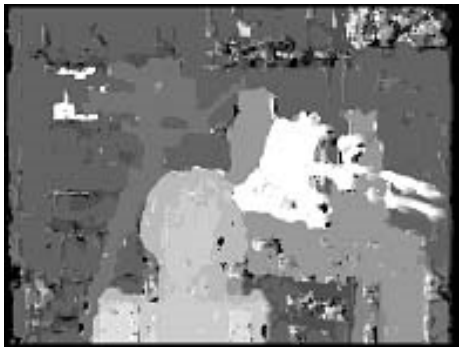
ground truth

问题：视差图灰度级代表的含义？

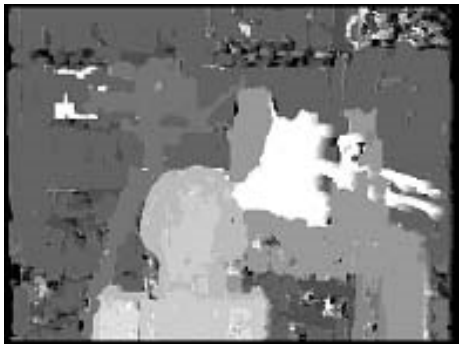
局部窗口立体匹配结果



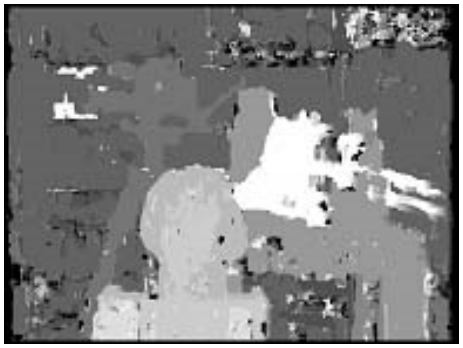
(a) SAD 匹配结果



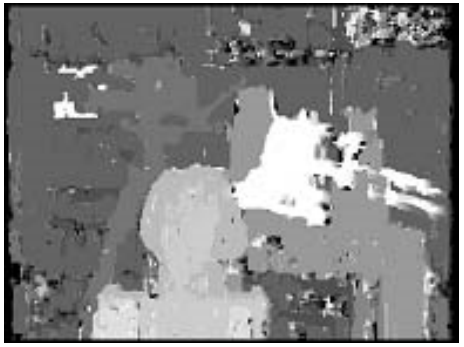
(b) ZSAD 匹配结果



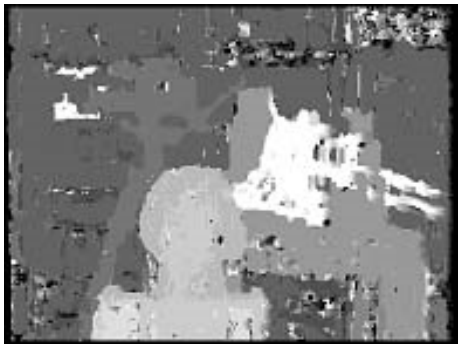
(c) SSD 匹配结果



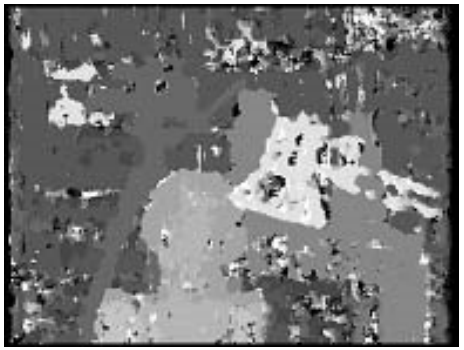
(d) ZSSD 匹配结果



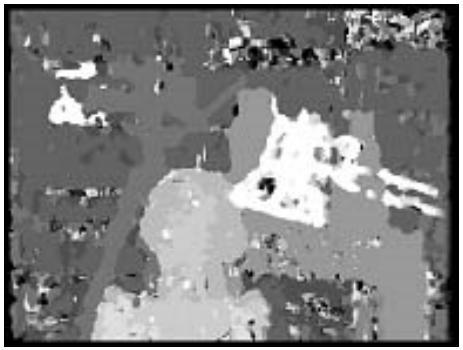
(e) NCC 匹配结果



(f) ZNCC 匹配结果



(g) Rank 匹配结果



(h) Census 匹配结果

区域立体匹配的实现

```
for ( j = line_start; j <= line_end;  j++ ) {
    for ( i=row_start; I <= row_end; i++ ) {    //遍历基准图像素
        for ( d = dpXMin; d <= dpXMax; d++ ) {  //视差搜索
            cov_tmp = 0.0; num = 0;
            for ( n = j-vradius; n <= j+m_vradius; n++) {
                for  ( m = i-hradius; m <= i+hradius; m++) { // 遍历窗口
                    addr = n*width+m;
                    subtmp  = template_img[addr] - shift_img[addr+d];
                    cov_tmp += (subtmp > 0) ? subtmp : -subtmp;
} } } } }
```

区域匹配的特点

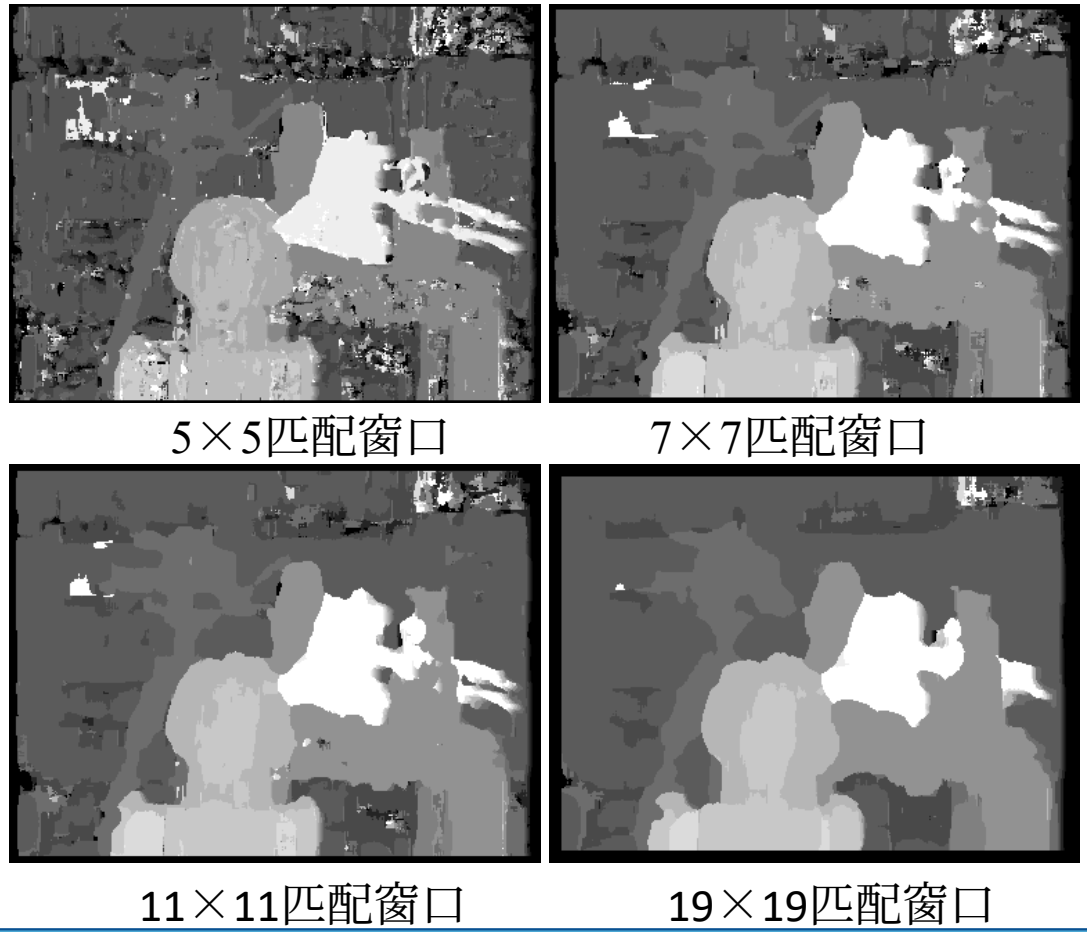
特点：

- 以窗口内像素亮度值的分布特性表征中心像素
  - 单个像素在亮度、色彩上存在歧义
- 隐含条件：匹配窗口内视差平滑
  - 匹配窗口内像素应具有相同的视差值。因此，当匹配窗口跨越深度不连续区域时就会因违背假设而引起误配。
- 匹配窗口大小的选择问题

匹配窗口大小的影响

- 匹配窗口变小
  - 细节丰富
  - 噪声大
  - 视差边缘处好
- 匹配窗口变大
  - 视差更平滑
  - 噪声小
  - 视差边缘处差

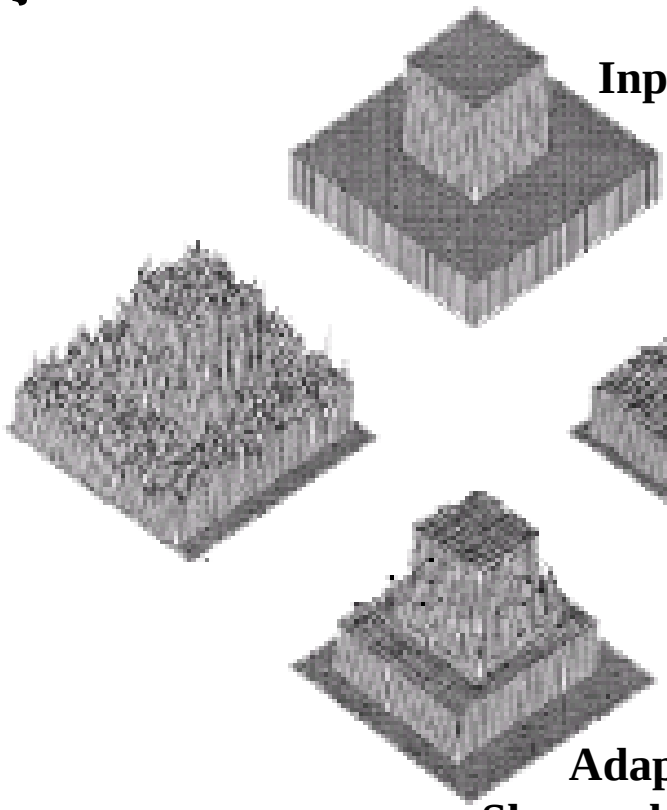
思考：试解释原因？



5×5匹配窗口　　7×7匹配窗口

11×11匹配窗口　　19×19匹配窗口

匹配窗口大小的影响



Input – Ground truth

3×3 window
Too noisy!

7×7 window
Sharp edges
are blurred!

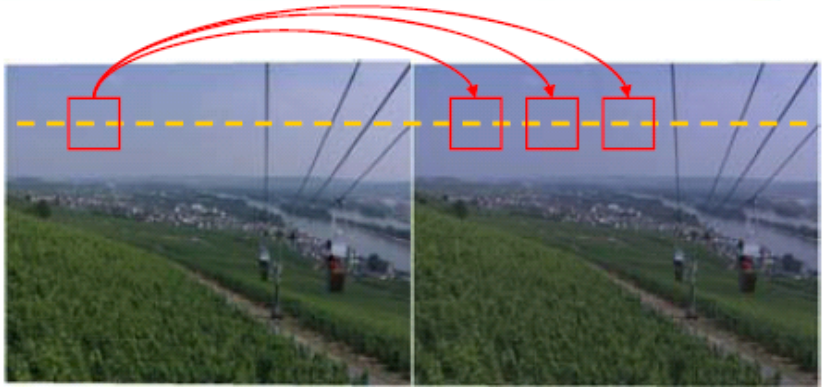
Adaptive window
Sharp edges and less noise

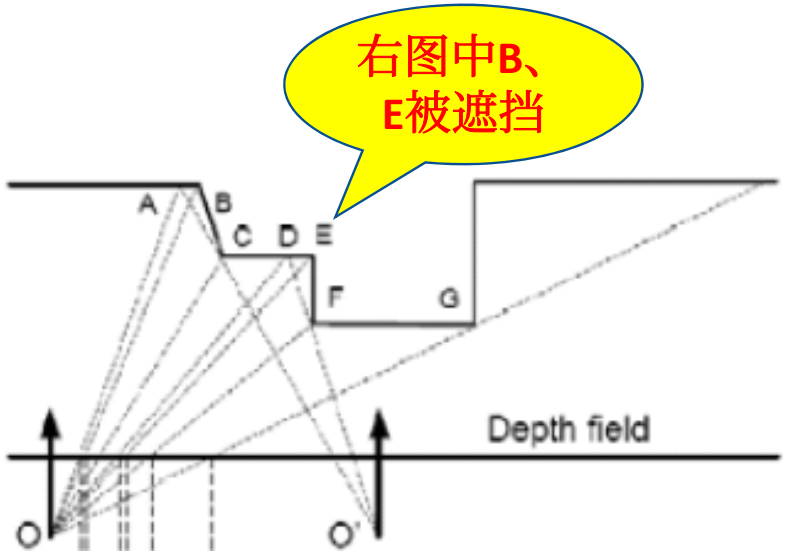
局部窗口匹配的优缺点

- 优点：
  - 容易实现，只需要考虑局部窗口区域
  - 对纹理丰富的区域具有较好匹配性能
  - 速度快，只需考虑有限像素
  - 易于硬件实现，易于流水线实现
- 缺点：
  - 视差不连续、遮挡或边缘区域无法正确估计视差
  - 对重复性纹理、无/弱纹理区域无法准确估计视差

立体匹配面临的挑战

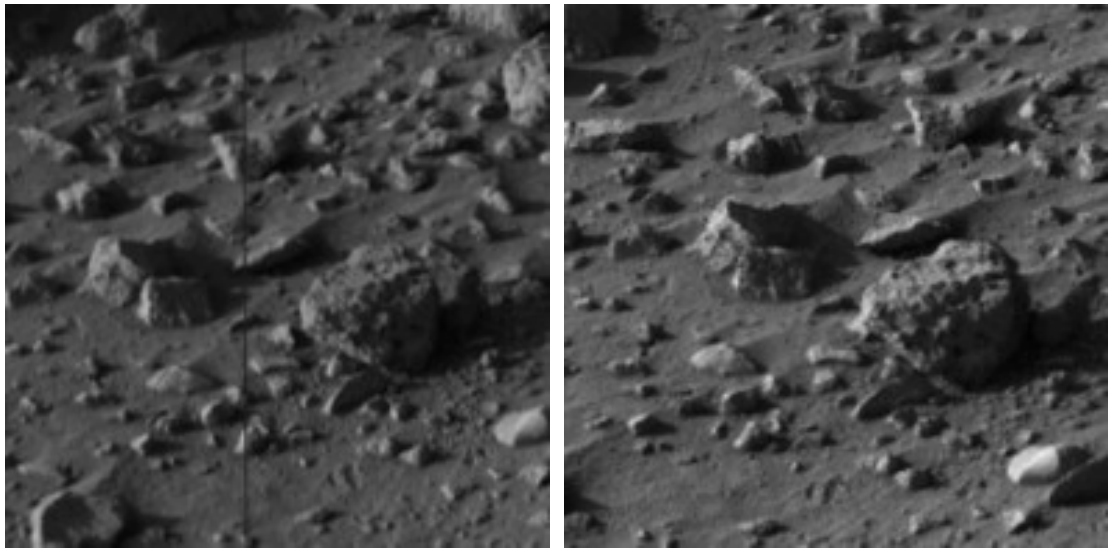
- 即使在测试的标准图像中匹配也并非容易
  - 重复场景
  - 无纹理区域
  - 遮挡



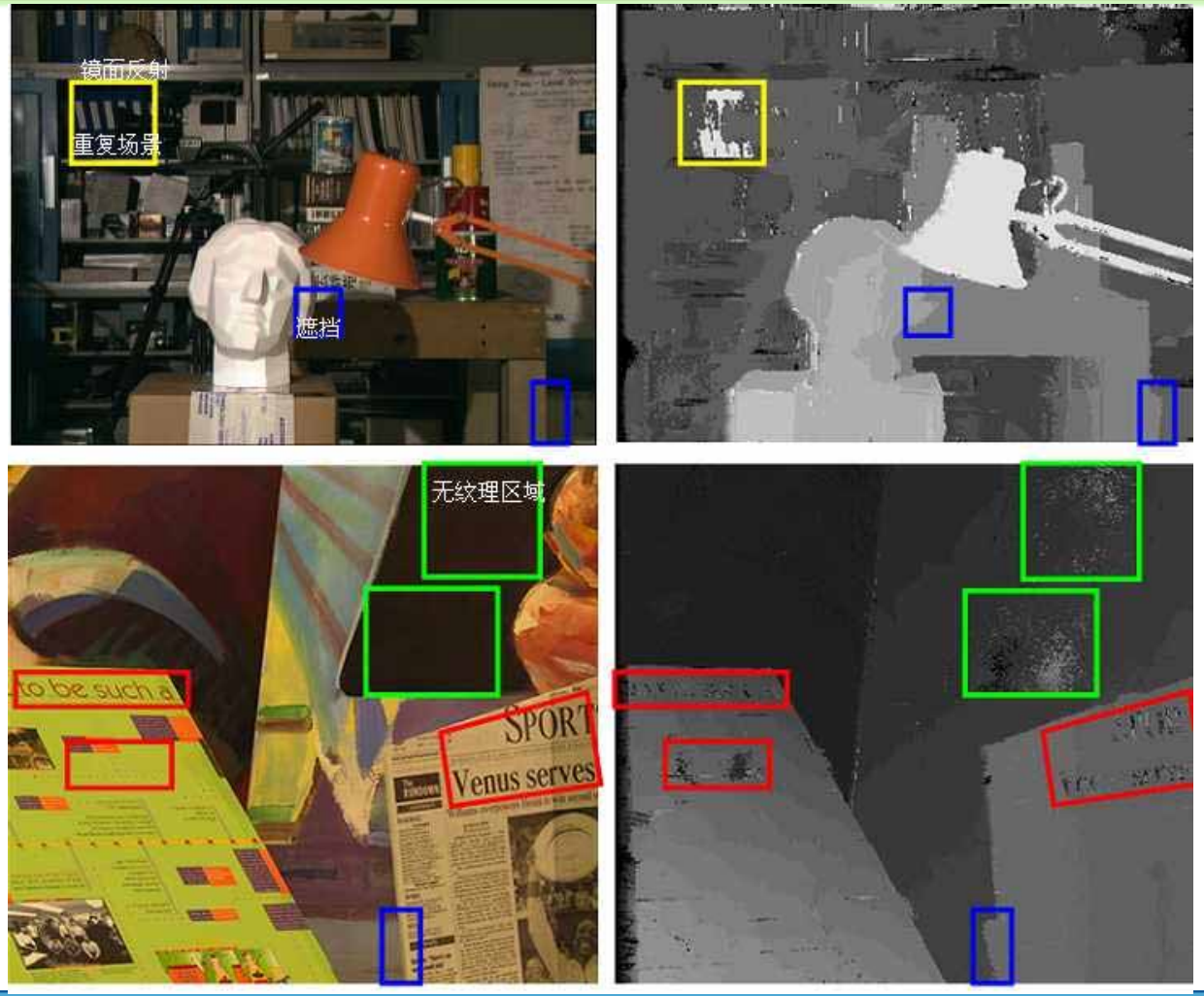


立体匹配面临的挑战

- 场景点投影到两幅图像中并不总是一致的
  - 像机的影响
    - 图像噪声、不同增益、不同对比度等等...
  - 视点的影响
    - 透视畸变
    - 遮挡
    - 镜面反射
    - 尺度、旋转变化



立体匹配面临的挑战



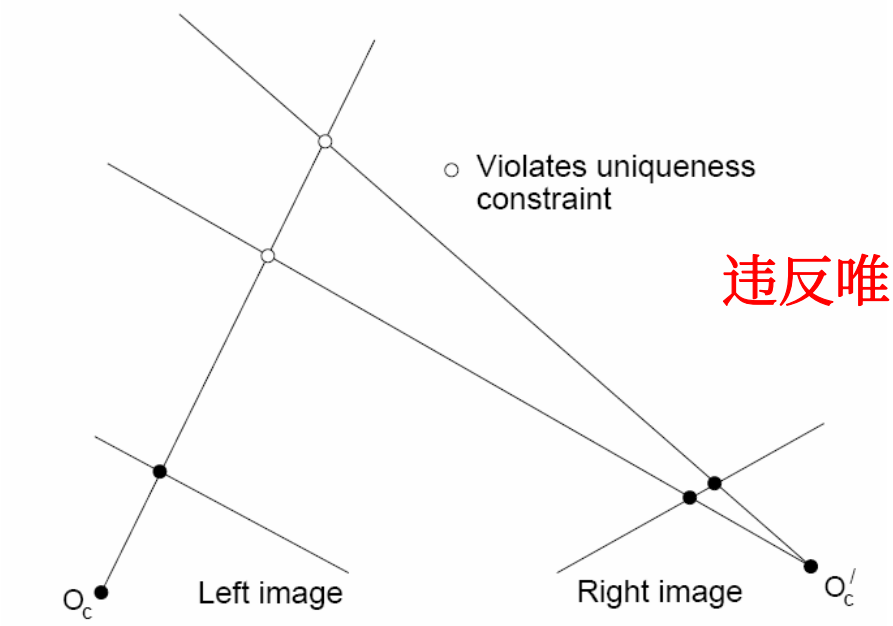
立体匹配约束

- 为克服匹配过程中存在的歧义性，需采用一些常用的匹配约束：
  - **极线约束**：匹配点必须在极线上
  - **相似性约束**：左、右图像的匹配点应具有相似的亮度或颜色。即，假定目标表面符合朗伯漫反射表面。
  - **视差范围约束**：仅在视差搜索内搜索。

思考：视差搜索范围如何确定？

立体匹配约束

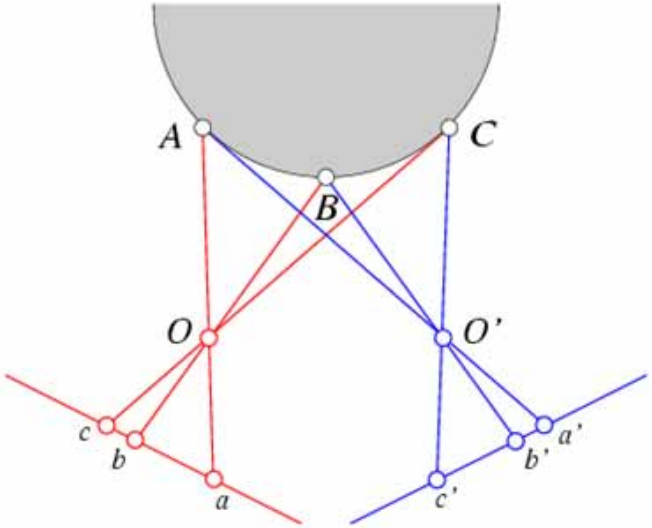
•唯一性约束：一幅图像中的一个像素，在另一幅图像中最多只有一个对应点像素。

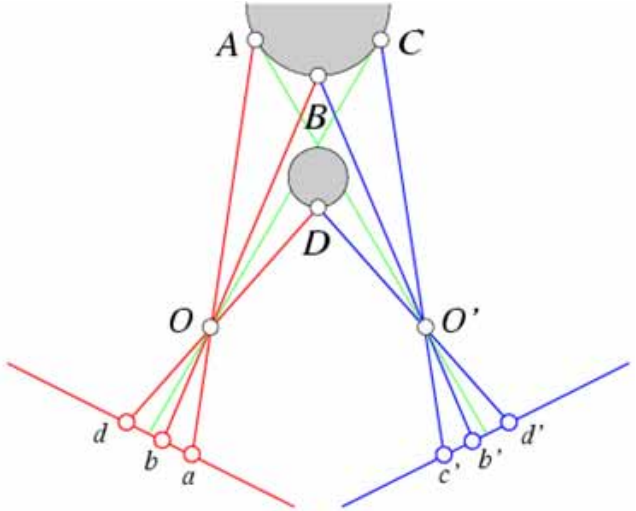


违反唯一性约束了吗?

立体匹配约束

- **顺序约束/单调性约束**：若参考图中A点在B点的左边，则另一幅图像中A点匹配点也在B点匹配点的左边。

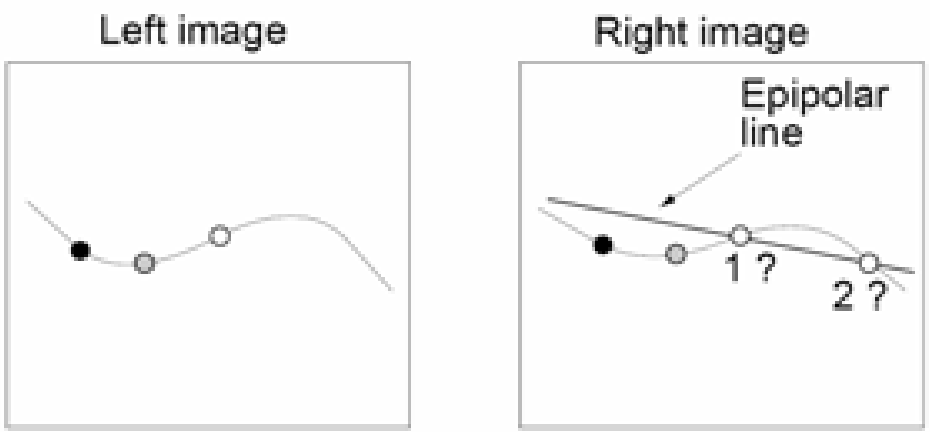




该约束对细小物体不成立

立体匹配约束

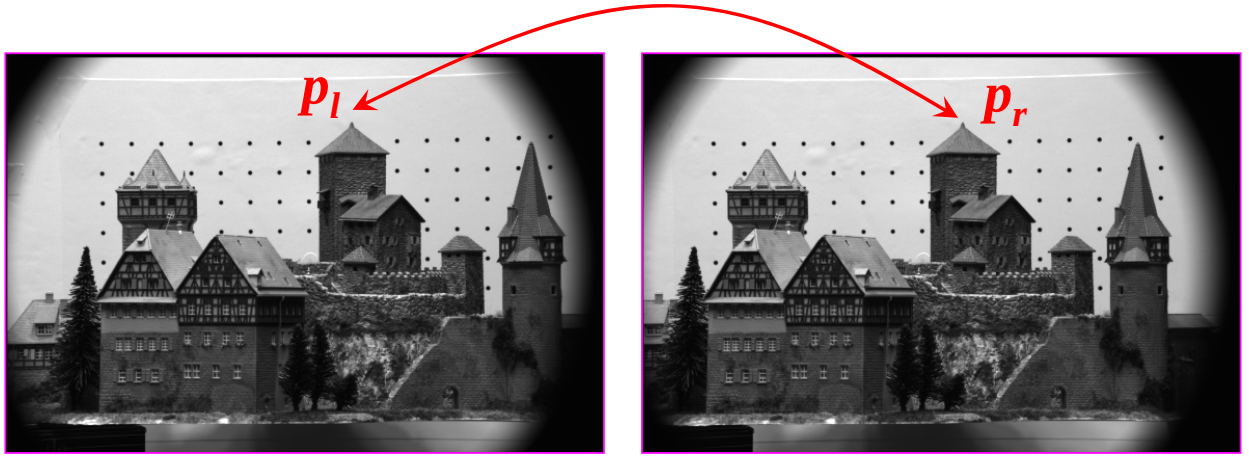
•**平滑性约束/一致性约束：**除了遮挡或视差本身不连续区域外，小邻域范围内视差值变化量应很小或相似。换言之视差曲面应是分段连续的。



Given matches ● and ◎, point ○ in the left image must match point 1 in the right image. Point 2 would exceed the disparity gradient limit.

立体匹配约束

- **互对应约束**：又称左右一致性，若以左图为基准图，左图上一像素点 p_l 的搜索到右图上对应点像素为 p_r ；那么若以右图为基准图，像素 p_r 的对应点也应该是左图上的像素点 p_l。该约束常用于遮挡区的检测。



立体匹配约束小结

- 约束条件
  - 极线约束
  - 相似性约束
  - 视差范围约束

  在区域内

  思考：匹配约束如何实现？

  - 唯一性约束
  - 顺序约束/单调性约束
  - 互对应约束

  应如何实现？

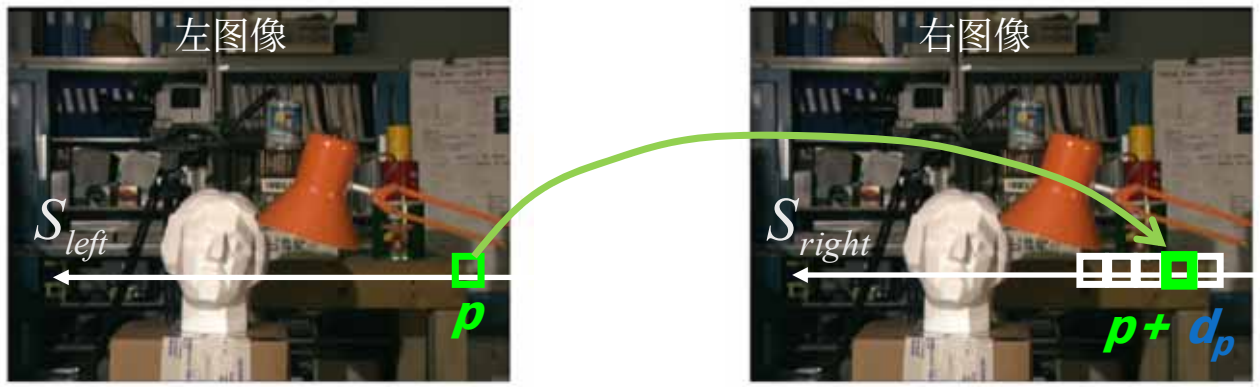
  区域匹配中各像素的对应性搜索相互独立的，而约束要求考虑相邻像素间的视差关系。

9.3.2 全局立体匹配

全局立体匹配

- 根据约束条件作用范围，可分为两大类：
  - **一维优化策略，基于动态规划**
    - 顺序约束和平滑约束优化极线上所有像素点的匹配代价。
    - 该问题可归结为路径规划问题，即寻找视差空间图的最小代价路径。
  - **二维优化策略，基于全局能量函数**
    - 根据贝叶斯理论及马尔可夫随机场理论，立体视觉问题可以转化为求解全局能量最小问题。
    - 全局能量函数为整幅图像所有像素点的匹配代价。

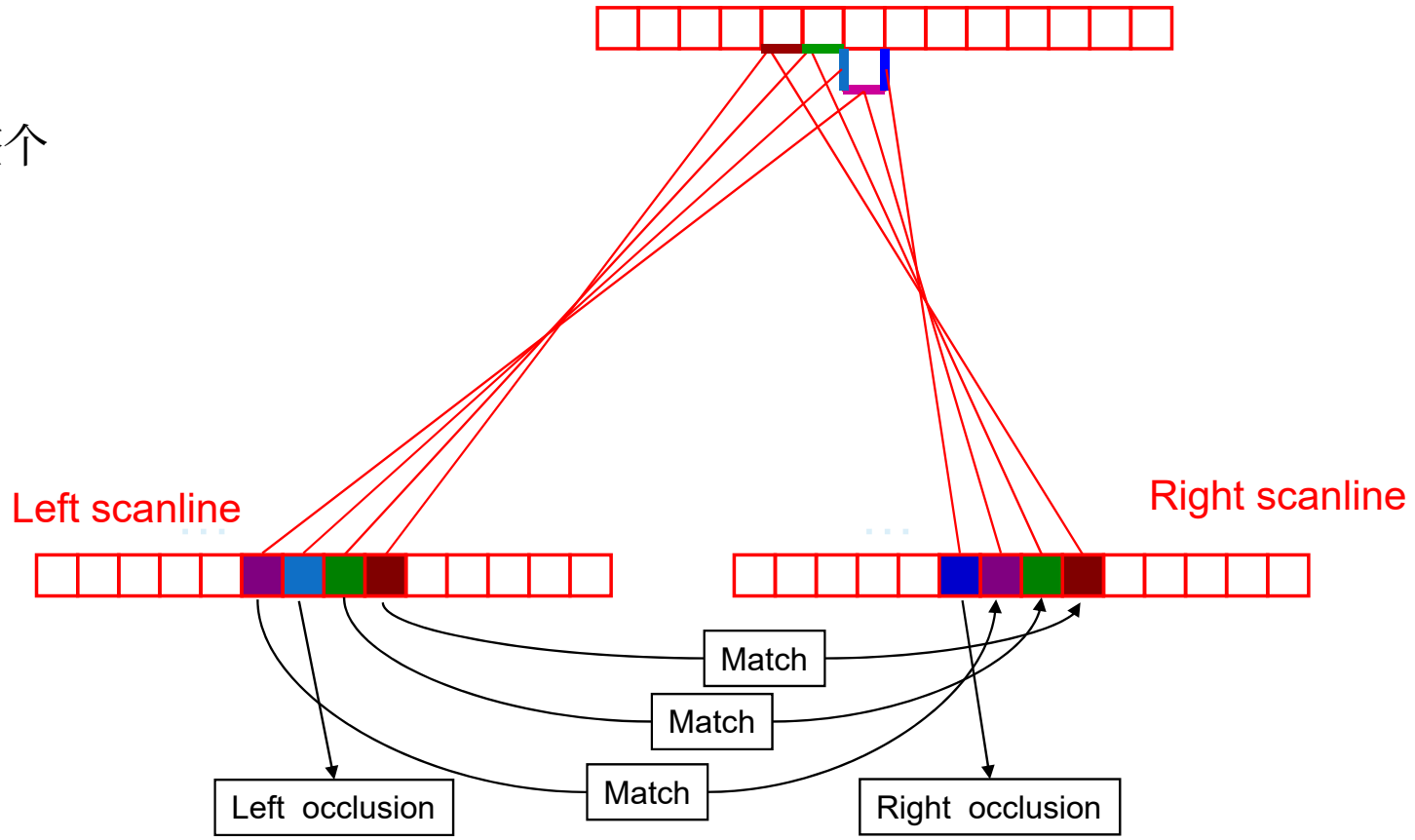
基于动态规划的立体匹配方法



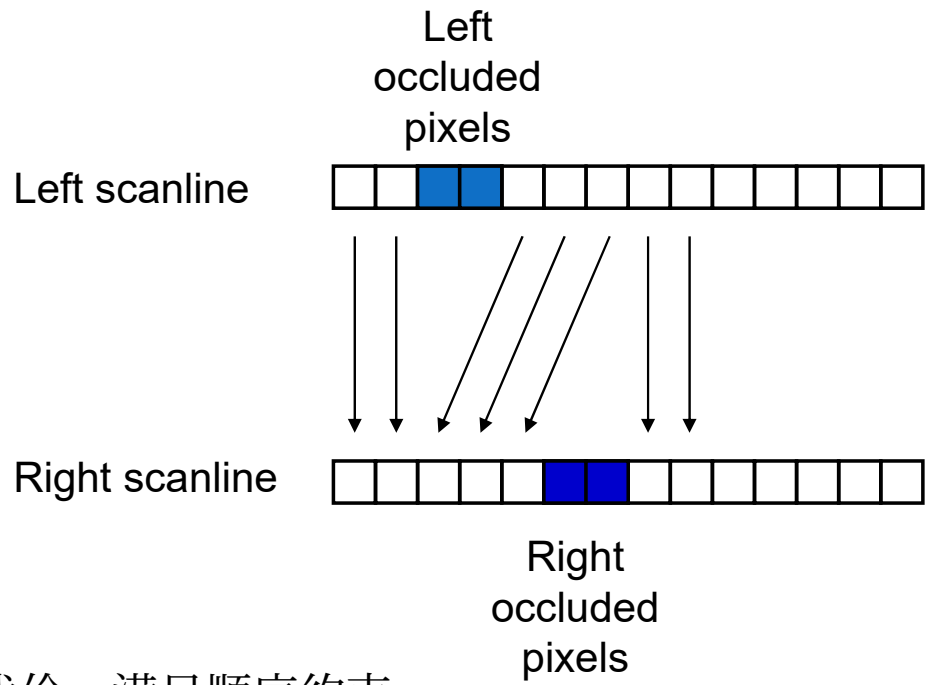
- 基于动态规划的匹配算法并不是孤立地寻找每个像素点的匹配值。
- **优化整条扫描线**，使得该扫描线上所有像素的匹配代价和最小，并满足顺序和平滑约束。
- 不同扫描线独立完成优化。

立体匹配约束

- 匹配过程中关注的是整个扫描线上的所有像素



立体匹配约束

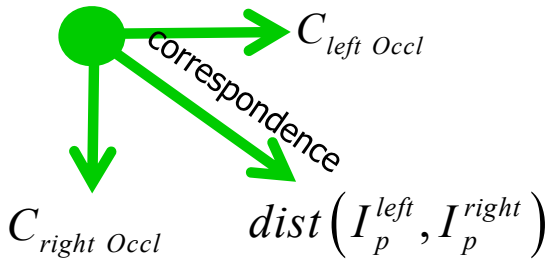


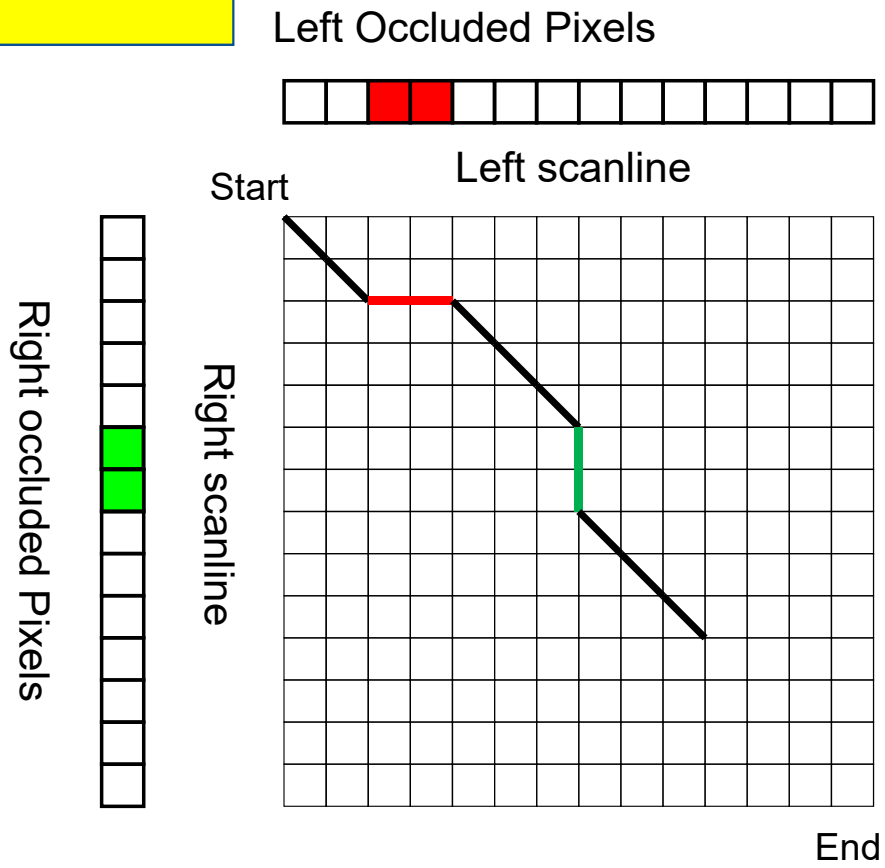
- 三种情况：
  - 连　续——匹配代价，满足顺序约束
  - 左遮挡——无匹配代价
  - 右遮挡——无匹配代价

视差空间

立体匹配问题转换为最优路径的搜索问题

- 动态规划在视差空间中生成一条最优路径。
- 路径生成满足顺序约束
- 遮挡、匹配搜索方向及其代价示意：

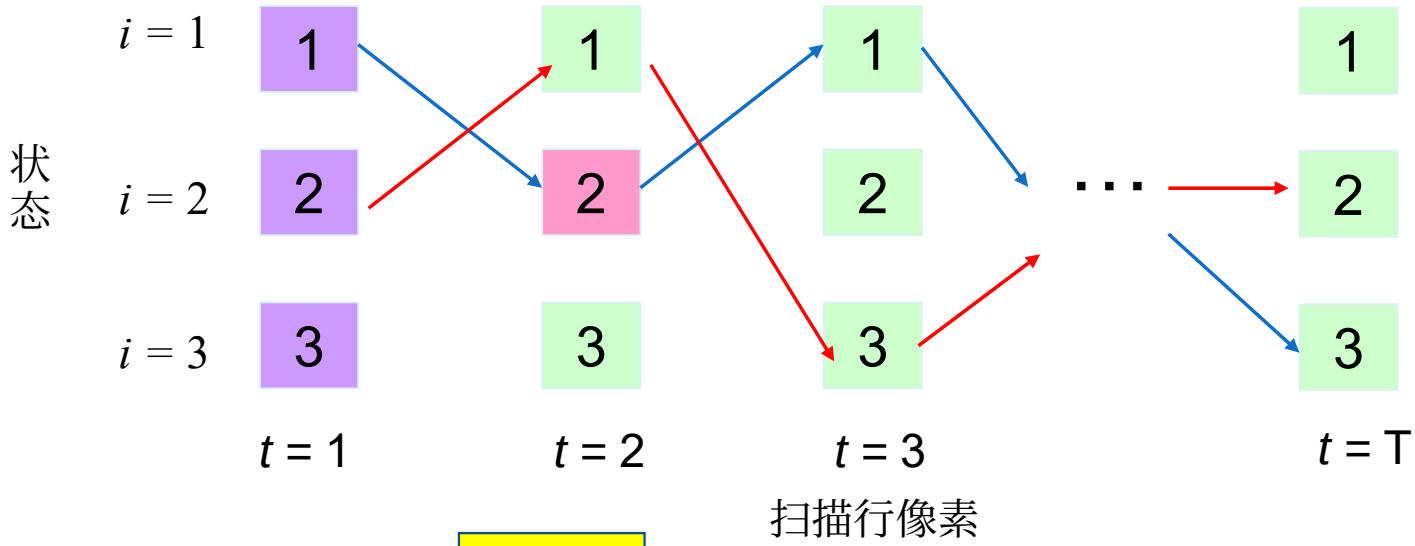




要求：寻找匹配代价和最小的一条路径

动态规划

•动态规划是求解连续决策（最优路径）问题的有效方法。

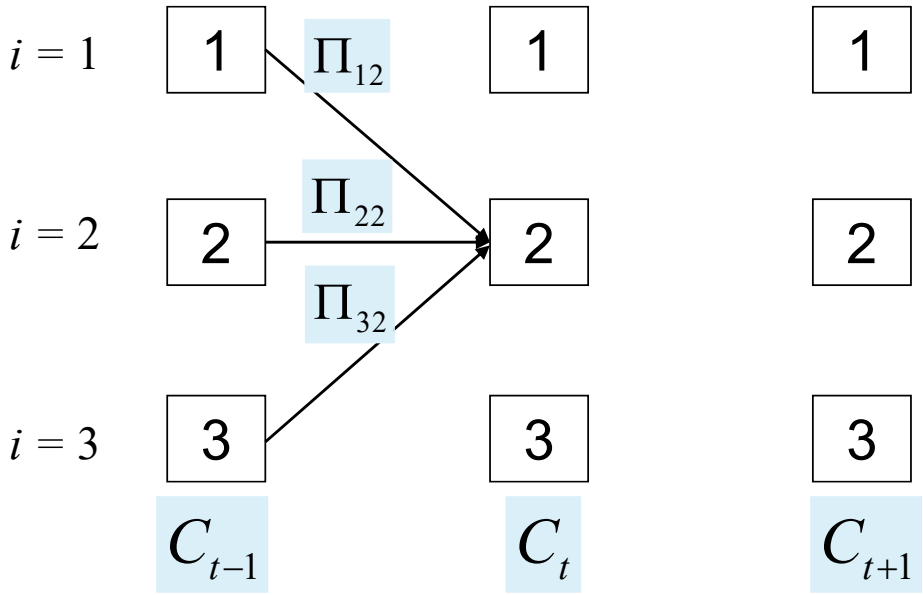


三种状态：

- 匹配
- 左遮挡
- 右遮挡

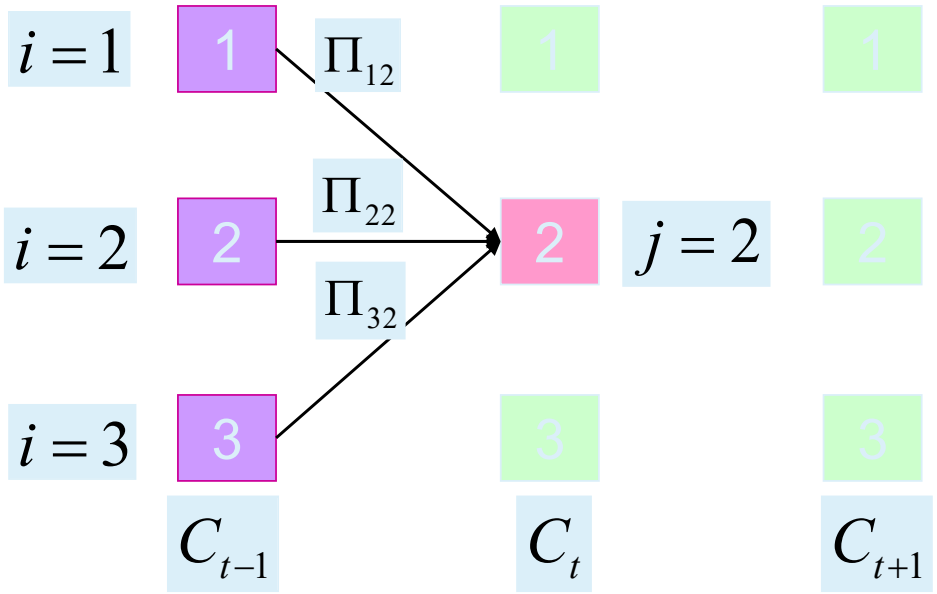
总共有多少条路径？ 3^T

动态规划



- 将整条扫描线匹配过程分解为多阶段决策过程。
- 单个阶段决策的代价为： $\Pi_{ij} = \text{Cost of going from state } i \text{ to state } j$

动态规划



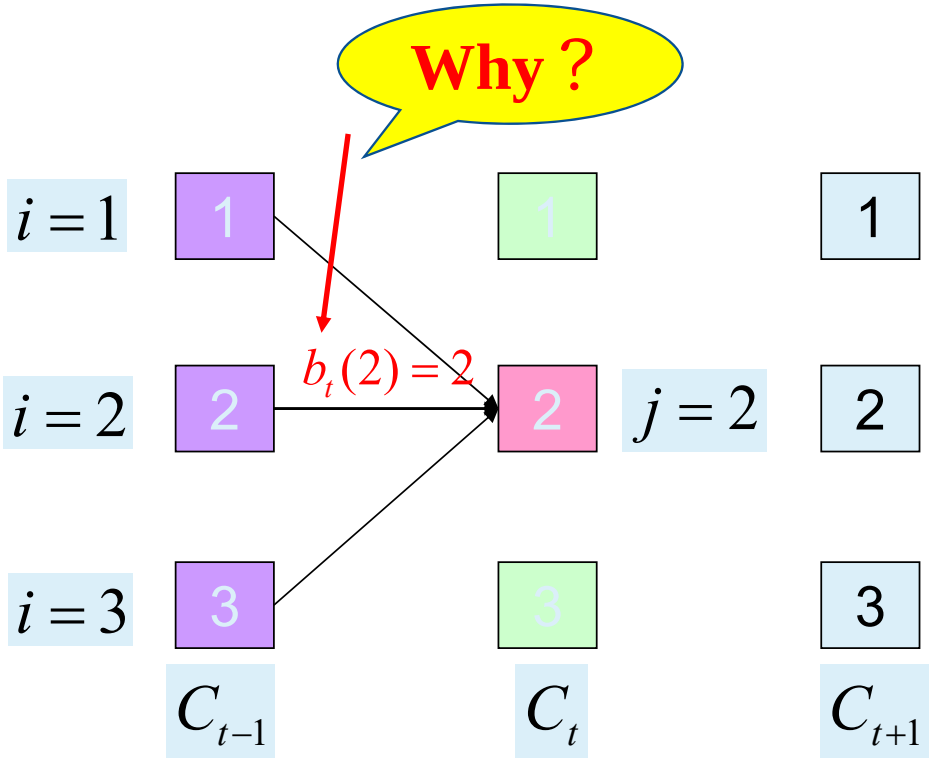
•n阶赋值问题的最优理论（体现动态规划方法）：

$$C_t(j)=\min_i(\Pi_{ij}+C_{t-1}(i))$$

此处最小化是全局概念还是局部概念?

动态规划

- 记录每个节点的父节点



$$C_t(j) = \min_i(\Pi_{ij} + C_{t-1}(i))$$

$$b_t(j) = \arg\min_i(\Pi_{ij} + C_{t-1}(i))$$

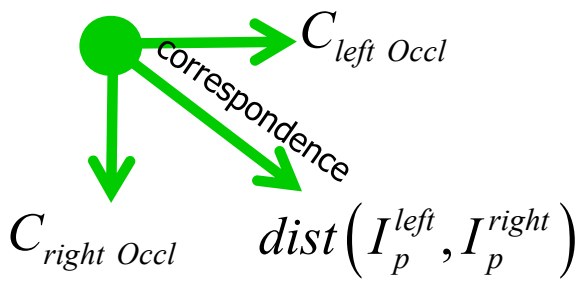
基于动态规划的立体匹配实现

•最优路径搜索的伪代码

```
for(i=1;i≤ N;i++){
   for(j=1;j≤ M;j++){
      min1 = C(i-1,j-1)+c(z_{1,i},z_{2,j});
      min2 = C(i-1,j)+Occlusion;
      min3 = C(i,j-1)+Occlusion;
      C(i,j) = cmin = min(min1,min2,min3);
      if(min1==cmin) M(i,j) = 1;
      if(min2==cmin) M(i,j) = 2;
      if(min3==cmin) M(i,j) = 3;
}}
```

含义？

右遮挡



•注：M(i, j) 用于记录父节点

基于动态规划的立体匹配实现

- 最优路径回溯伪代码

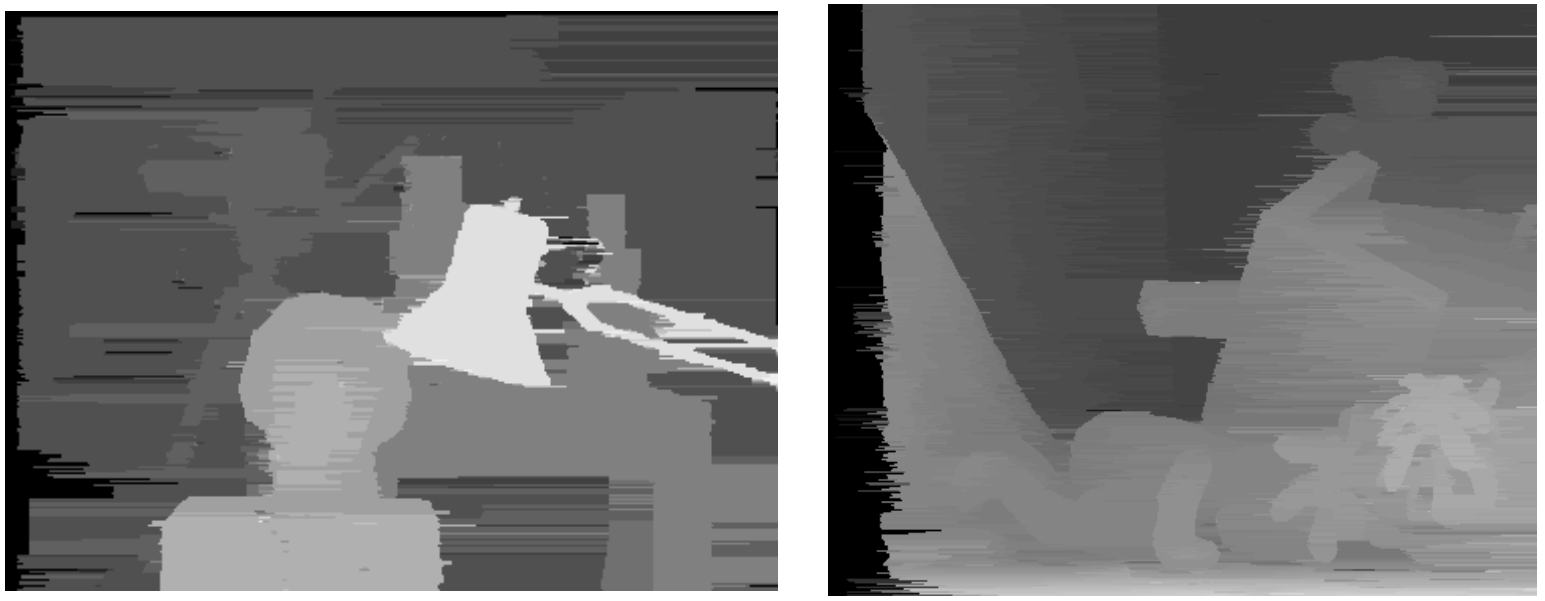
```
p=N;
q=M;
while(p!=0 && q!=0){
   switch(M(p,q)){
      case 1:
         p matches q
         p--;q--;
         break;
      case 2:
         p is unmatched
         p--;
         break;
      case 3:
         q is unmatched
         q--;
         break;
}}
```

遮挡——跳过直到找到下一个匹配点

动态规划方法的局限性

- 优点：
  - 保证了一条扫描线上各像素点的优化匹配。
- 缺点：
  - 缺少扫描线间的强制约束
  - 无法将水平方向和垂直方向的连续性约束有效融合。
  - 局部误差会着扫描线传播。
  - 视差结果图中有着很明显的横纹效应。

动态规划的匹配结果



横纹效应明显

二维优化策略

- 如何评判立体匹配算法性能的优劣?
  - **匹配质量**
    - 为每个像素在另一幅图像中寻找到最佳匹配。
  - **平滑性**
    - 若两像素相邻，则它们的视差（通常）相近。

思路：定义一个全局能量函数，并求解该能量函数最小

全局能量函数

•立体匹配的全局能量函数定义：

$$E(d) = E_d(d) + \lambda E_s(d)$$

数据项: 匹配代价

平滑项: 平滑代价

使得每个像素在另一幅图像中找到一个良好的匹配。

相邻像素（通常）应具有相近的视差。

全局能量函数

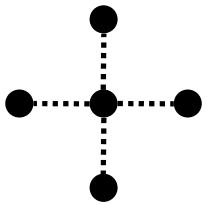
$$E(d) = E_d(d) + \lambda E_s(d)$$

•数据项： $E_d(d) = \sum_{(x,y)\in I} C(x, y, d(x, y))$

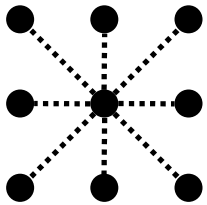
所有像素的匹配代价和

•平滑项： $E_s(d) = \sum_{(p,q)\in\mathcal{E}} V(d_p, d_q)$

邻域像素集



4-connected neighborhood



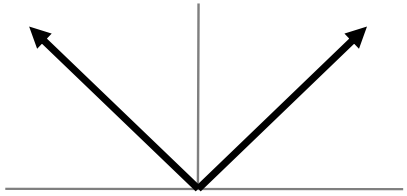
8-connected neighborhood

平滑项函数

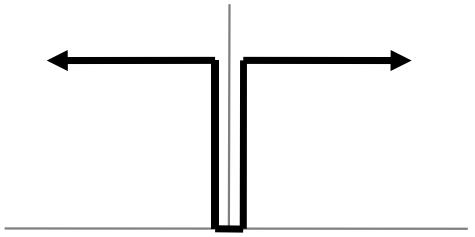
•平滑项：是邻域像素违反平滑性约束的惩罚项。

$$E_s(d) = \sum_{(p,q)\in\mathcal{E}} V(d_p, d_q)$$

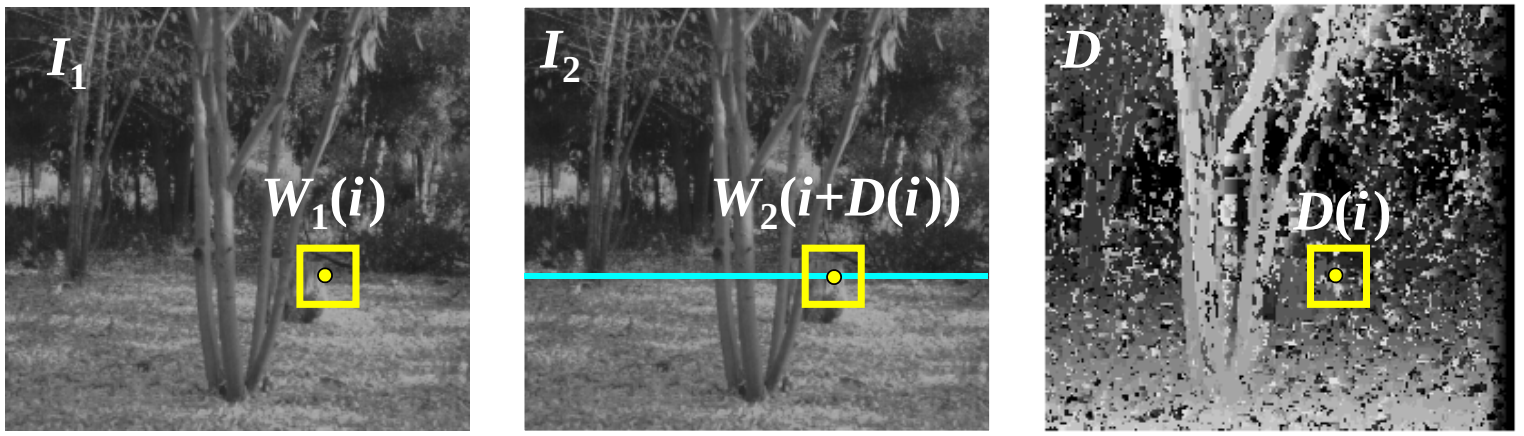
•常用模型函数：　L_1 distance　$V(d_p, d_q) = |d_p - d_q|$



“Potts model”　$V(d_p, d_q) = \begin{cases} 0 & \text{if } d_p = d_q \\ 1 & \text{if } d_p \neq d_q \end{cases}$



全局能量函数举例

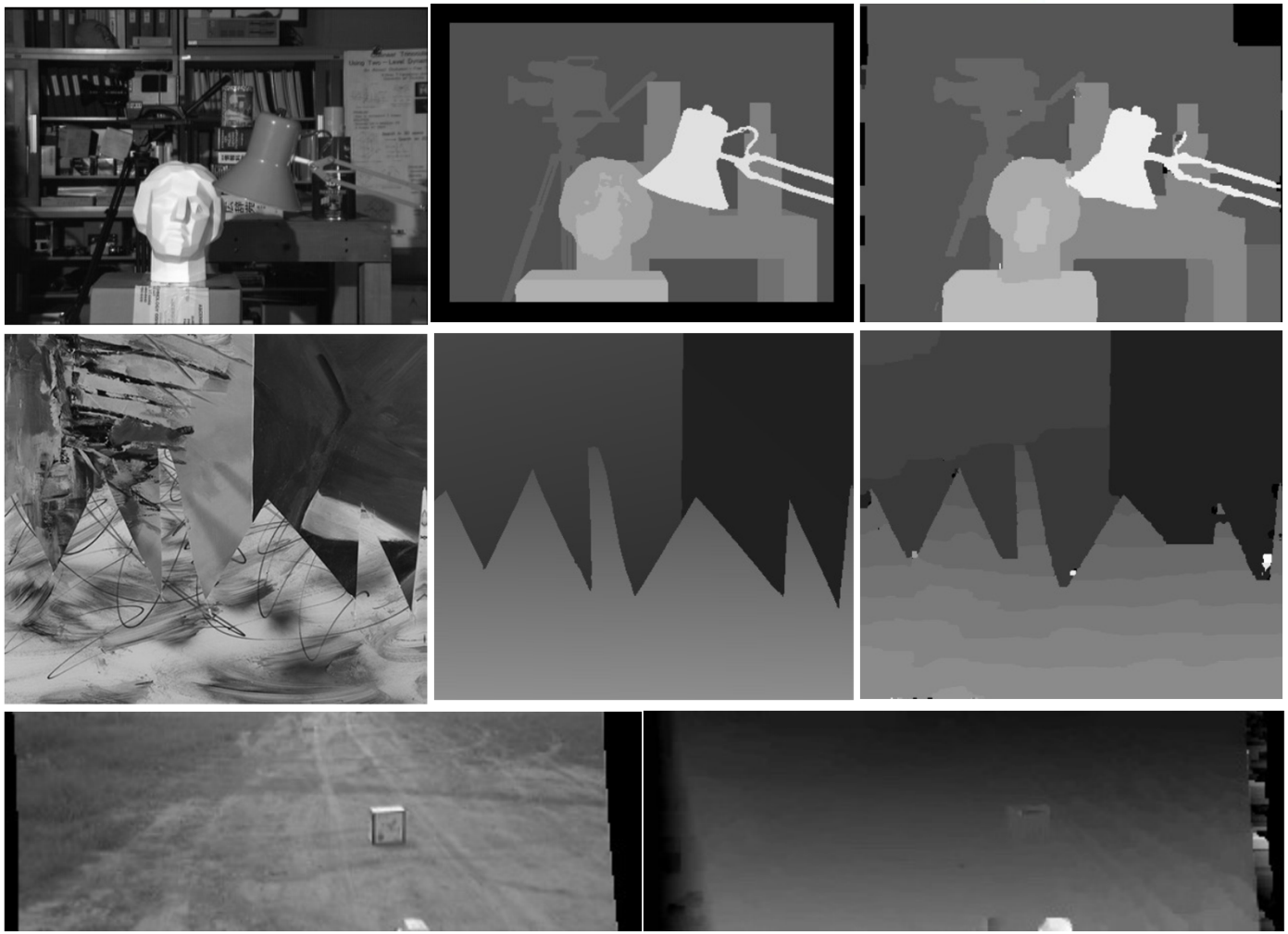


$$E(D) = \underbrace{\sum_{i} \left(W_1(i) - W_2(i + D(i))\right)^2}_{\text{data term}} + \lambda \underbrace{\sum_{\text{neighbors } i,j} \rho\left(D(i) - D(j)\right)}_{\text{smoothness term (Potts Models)}}$$

Y. Boykov, O. Veksler, and R. Zabih, Fast Approximate Energy Minimization via Graph Cuts, PAMI 2001

全局能量函数最小化求解

- 常用能量函数最小求解方法
  - 基于图的求解
    - Graph Cuts
  - 基于概率的求解
    - Belief Propagation（BP）



立体匹配算法测评

Algorithm	Avg. Rank	Tsukuba ground truth nonocc	Tsukuba all	Tsukuba disc	Venus ground truth nonocc	Venus all	Venus disc	Teddy ground truth nonocc	Teddy all	Teddy disc	Cones ground truth nonocc	Cones all	Cones disc
AdaptingBP [17]	2.8	1.11 6	1.37 3	5.79 7	**0.10** 1	0.21 2	**1.44** 1	4.22 4	7.06 2	11.8 4	**2.48** 1	7.92 2	**7.32** 1
DoubleBP2 [35]	2.9	**0.88** 1	**1.29** 1	**4.76** 1	0.13 3	0.45 5	1.87 5	3.53 2	8.30 3	**9.63** 1	2.90 3	8.78 8	7.79 2
DoubleBP [15]	4.9	0.88 2	1.29 2	4.76 2	0.14 5	0.60 13	2.00 7	3.55 3	8.71 5	9.70 2	2.90 4	9.24 11	7.80 3
SubPixDoubleBP [30]	5.6	1.24 10	1.76 13	5.98 8	0.12 2	0.46 6	1.74 4	**3.45** 1	8.38 4	10.0 3	2.93 5	8.73 7	7.91 4
AdaptOvrSegBP [33]	9.9	1.69 22	2.04 21	5.64 6	0.14 4	**0.20** 1	1.47 2	7.04 14	11.1 7	16.4 11	3.60 11	8.96 10	8.84 10
SymBP+occ [7]	10.8	0.97 4	1.75 12	5.09 4	0.16 6	0.33 3	2.19 8	6.47 8	10.7 6	17.0 14	4.79 24	10.7 21	10.9 20
PlaneFitBP [32]	10.8	0.97 5	1.83 14	5.26 5	0.17 7	0.51 8	1.71 3	6.65 9	12.1 13	14.7 7	4.17 20	10.7 20	10.6 19
AdaptDispCalib [36]	11.8	1.19 8	1.42 4	6.15 9	0.23 9	0.34 4	2.50 11	7.80 19	13.6 21	17.3 17	3.62 12	9.33 12	9.72 15
Segm+visib [4]	12.2	1.30 15	1.57 5	6.92 18	0.79 21	1.06 18	6.76 22	5.00 5	**6.54** 1	12.3 5	3.72 13	8.62 6	10.2 17
C-SemiGlob [19]	12.3	2.61 29	3.29 24	9.89 27	0.25 12	0.57 10	3.24 15	5.14 6	11.8 8	13.0 6	2.77 2	8.35 4	8.20 5
SC													0.2 18
D													.32 7
Cos													36 13
OverSegmBP [26]	14.5	1.69 23	1.97 18	8.47 24	0.51 18	0.68 15	4.69 18	6.74 10	11.9 12	15.8 8	3.19 8	8.81 9	8.89 11
SegmentSupport [28]	15.1	1.25 11	1.62 7	6.68 13	0.25 11	0.64 14	2.59 12	8.43 24	14.2 22	18.2 20	3.77 14	9.87 17	9.77 16
RegionTreeDP [18]	15.7	1.39 19	1.64 8	6.85 16	0.22 8	0.57 10	1.93 6	7.42 17	11.9 11	16.8 13	6.31 30	11.9 27	11.8 23
EnhancedBP [24]	16.6	0.94 3	1.74 10	5.05 3	0.35 15	0.86 17	4.34 17	8.11 22	13.3 19	18.5 22	5.09 27	11.1 23	11.0 21

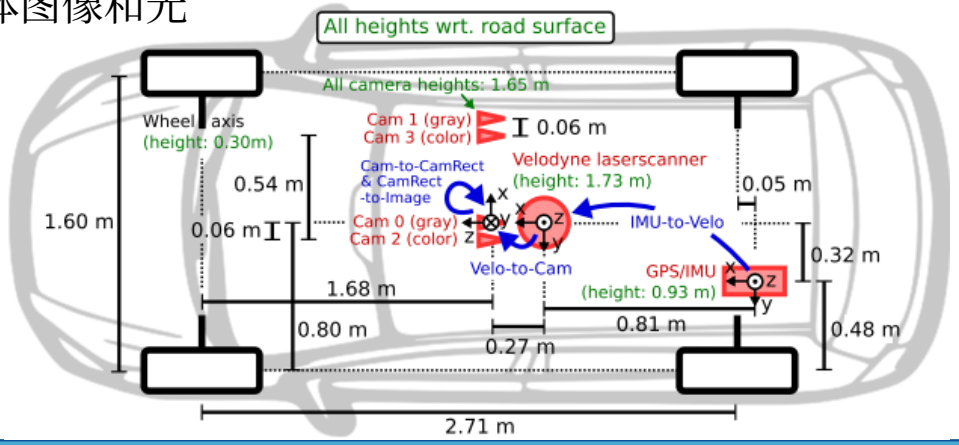
http://vision.middlebury.edu/stereo/eval/

KITTI数据集

http://www.cvlibs.net/datasets/kitti

KITTI数据集由德国卡尔斯鲁厄理工学院和丰田美国技术研究院联合创办，是目前国际上最大的自动驾驶场景下的计算机视觉算法评测数据集。用于评测立体图像（stereo），光流（optical flow），视觉里程计（visual odometry），3D物体检测（object detection）、3D跟踪（tracking）等计算机视觉技术在车载环境下的性能。

KITTI包含市区、乡村和高速公路等场景的真实图像，每张图像中最多达15辆车和30个行人，还有各种程度的遮挡。整个数据集由389对立体图像和光流图，39.2 km视觉里程序列以及超过200k 3D标注的目标图像。



车载传感器：

- 1 Inertial Navigation System (GPS/IMU): OXTS RT 3003
- 1 Laserscanner: Velodyne HDL-64E
- 2 Grayscale cameras, 1.4 Megapixels: Point Grey Flea 2 (FL2-14S3M-C)
- 2 Color cameras, 1.4 Megapixels: Point Grey Flea 2 (FL2-14S3C-C)
- 4 Varifocal lenses, 4-8 mm: Edmund Optics NT59-917

KITTI数据集

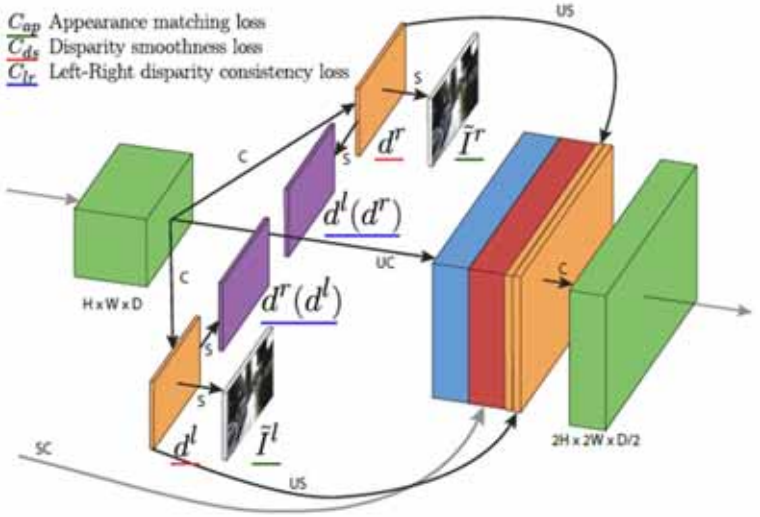
KITTI数据集的典型样本，分为 Road、City、 Residential、 Campus和Person五类。原始数据采集于2011年的5天，共约176GB数据。

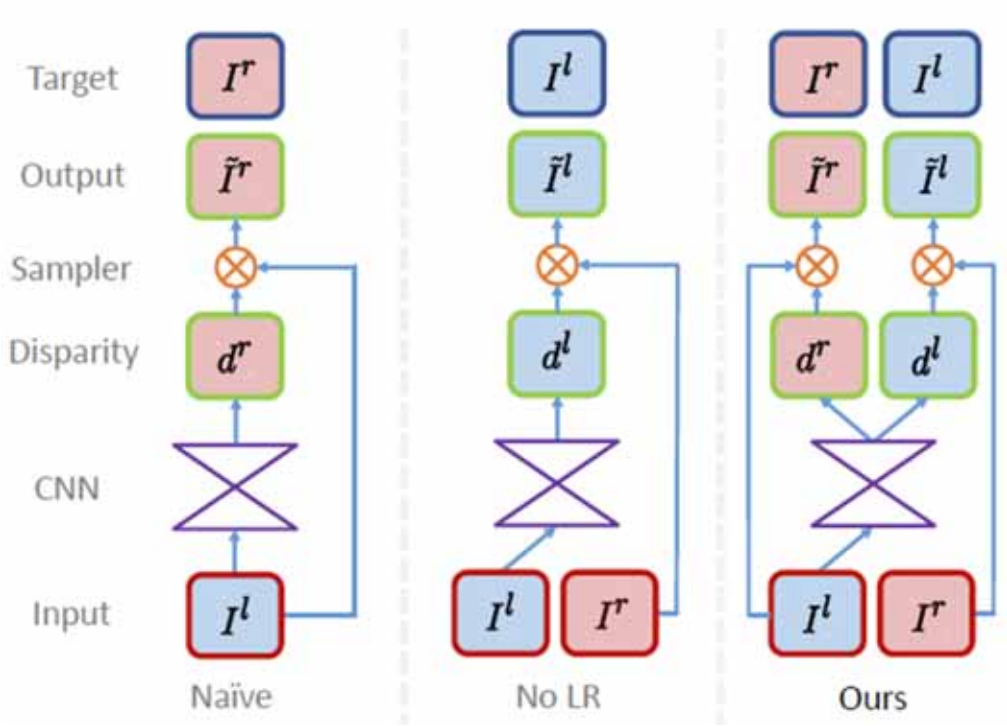
```
date/
├── date_drive/
│   ├── date_drive.zip
│   │   ├── image_0x/   x={0,..,3}
│   │   │   ├── data/
│   │   │   │   └── frame_number.png
│   │   │   └── timestamps.txt
│   │   ├── oxts/
│   │   │   ├── data/
│   │   │   │   └── frame_number.txt
│   │   │   ├── dataformat.txt
│   │   │   └── timestamps.txt
│   │   └── velodyne_points/
│   │       ├── data/
│   │       │   └── frame_number.bin
│   │       ├── timestamps.txt
│   │       ├── timestamps_start.txt
│   │       └── timestamps_end.txt
│   └── date_drive_tracklets.zip
│       └── tracklet_labels.xml
└── date_calib.zip
    ├── calib_cam_to_cam.txt
    ├── calib_imu_to_velo.txt
    └── calib_velo_to_cam.txt
```

U-Net单目深度估计

Clement Godard, OisinMac Aodha, Gabriel J. Brostow. UnsupervisedMonocular Depth Estimation with Left-Right Consistency. CVPR 2017. http://visual.cs.ucl.ac.uk/pubs/monoDepth/

https://github.com/mrharicot/monodepth





U-Net单目深度估计

Encoder：VGG16

"Encoder"						
layer	k	s	chns	in	out	input
conv1	7	2	3/32	1	2	left
conv1b	7	1	32/32	2	2	conv1
conv2	5	2	32/64	2	4	conv1b
conv2b	5	1	64/64	4	4	conv2
conv3	3	2	64/128	4	8	conv2b
conv3b	3	1	128/128	8	8	conv3
conv4	3	2	128/256	8	16	conv3b
conv4b	3	1	256/256	16	16	conv4
conv5	3	2	256/512	16	32	conv4b
conv5b	3	1	512/512	32	32	conv5
conv6	3	2	512/512	32	64	conv5b
conv6b	3	1	512/512	64	64	conv6
conv7	3	2	512/512	64	128	conv6b
conv7b	3	1	512/512	128	128	conv7

```
def build_vgg(self):
    #set convenience functions
    conv = self.conv
    if self.params.use_deconv:
        upconv = self.deconv
    else:
        upconv = self.upconv

    with tf.compat.v1.variable_scope('encoder'):
        conv1 = self.conv_block(self.model_input,  32, 7) # H/2
        conv2 = self.conv_block(conv1,             64, 5) # H/4
        conv3 = self.conv_block(conv2,            128, 3) # H/8
        conv4 = self.conv_block(conv3,            256, 3) # H/16
        conv5 = self.conv_block(conv4,            512, 3) # H/32
        conv6 = self.conv_block(conv5,            512, 3) # H/64
        conv7 = self.conv_block(conv6,            512, 3) # H/128
```

```
with tf.compat.v1.variable_scope('skips'):
    skip1 = conv1
    skip2 = conv2
    skip3 = conv3
    skip4 = conv4
    skip5 = conv5
    skip6 = conv6
```

```
def conv_block(self, x, num_out_layers, kernel_size):
    conv1 = self.conv(x,     num_out_layers, kernel_size, 1)
    conv2 = self.conv(conv1, num_out_layers, kernel_size, 2)
    return conv2
```

```
def conv(self, x, num_out_layers, kernel_size, stride, activation_fn=tf.nn.elu):
    p = np.floor((kernel_size - 1) / 2).astype(np.int32)
    p_x = tf.pad(x, [[0, 0], [p, p], [p, p], [0, 0]])
    return slim.conv2d(p_x, num_out_layers, kernel_size, stride, 'VALID', activation_fn=activation_fn)
```

U-Net单目深度估计

Decoder：

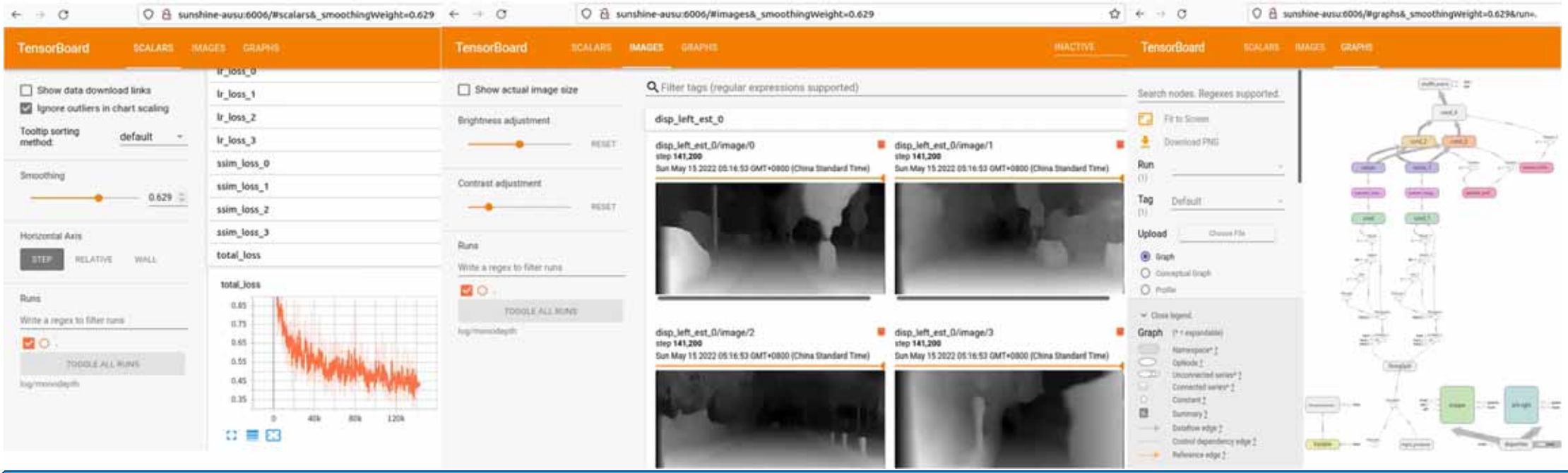
"Decoder"						
layer	**k**	**s**	**chns**	**in**	**out**	**input**
upconv7	3	2	512/512	128	64	conv7b
iconv7	3	1	1024/512	64	64	upconv7+conv6b
upconv6	3	2	512/512	64	32	iconv7
iconv6	3	1	1024/512	32	32	upconv6+conv5b
upconv5	3	2	512/256	32	16	iconv6
iconv5	3	1	512/256	16	16	upconv5+conv4b
upconv4	3	2	256/128	16	8	iconv5
iconv4	3	1	128/128	8	8	upconv4+conv3b
disp4	3	1	128/2	8	8	iconv4
upconv3	3	2	128/64	8	4	iconv4
iconv3	3	1	130/64	4	4	upconv3+conv2b+disp4*
disp3	3	1	64/2	4	4	iconv3
upconv2	3	2	64/32	4	2	iconv3
iconv2	3	1	66/32	2	2	upconv2+conv1b+disp3*
disp2	3	1	32/2	2	2	iconv2
upconv1	3	2	32/16	2	1	iconv2
iconv1	3	1	18/16	1	1	upconv1+disp2*
disp1	3	1	16/2	1	1	iconv1

```
with tf.compat.v1.variable_scope('decoder'):
    upconv7 = upconv(conv7,  512, 3, 2) #H/64
    concat7 = tf.concat([upconv7, skip6], 3)
    iconv7  = conv(concat7,  512, 3, 1)

    upconv6 = upconv(iconv7, 512, 3, 2) #H/32
    concat6 = tf.concat([upconv6, skip5], 3)
    iconv6  = conv(concat6,  512, 3, 1)

    upconv5 = upconv(iconv6, 256, 3, 2) #H/16
    concat5 = tf.concat([upconv5, skip4], 3)
    iconv5  = conv(concat5,  256, 3, 1)

    upconv4 = upconv(iconv5, 128, 3, 2) #H/8
    concat4 = tf.concat([upconv4, skip3], 3)
    iconv4  = conv(concat4,  128, 3, 1)
    self.disp4 = self.get_disp(iconv4)
    udisp4  = self.upsample_nn(self.disp4, 2)

    upconv3 = upconv(iconv4,  64, 3, 2) #H/4
    concat3 = tf.concat([upconv3, skip2, udisp4], 3)
    iconv3  = conv(concat3,   64, 3, 1)
    self.disp3 = self.get_disp(iconv3)
    udisp3  = self.upsample_nn(self.disp3, 2)

    upconv2 = upconv(iconv3,  32, 3, 2) #H/2
    concat2 = tf.concat([upconv2, skip1, udisp3], 3)
    iconv2  = conv(concat2,   32, 3, 1)
    self.disp2 = self.get_disp(iconv2)
    udisp2  = self.upsample_nn(self.disp2, 2)

    upconv1 = upconv(iconv2,  16, 3, 2) #H
    concat1 = tf.concat([upconv1, udisp2], 3)
    iconv1  = conv(concat1,   16, 3, 1)
    self.disp1 = self.get_disp(iconv1)
```

```
def upconv(self, x, num_out_layers, kernel_size, scale):
    upsample = self.upsample_nn(x, scale)
    conv = self.conv(upsample, num_out_layers, kernel_size, 1)
    return conv
def upsample_nn(self, x, ratio):
    s = tf.shape(x)
    h = s[1]
    w = s[2]
    return tf.compat.v1.image.resize_nearest_neighbor(x, [h * ratio, w * ratio])
```

Tensorboard查看训练过程

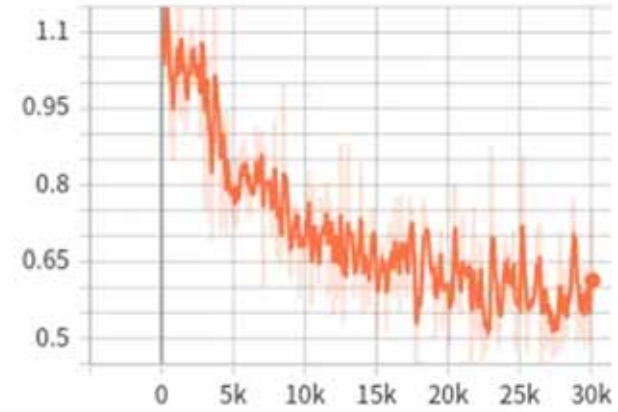
```
Terminal:  Local
(tf1.15.4.py3.8) sunshine@sunshine-ausu:~/MyPrograms/PyCharm/Examples/Tensorflow1.x/MonoDepth_UNet$ tensorboard --logdir=log/monodepth

TensorBoard 1.15.0+nv at http://sunshine-ausu:6006/ (Press CTRL+C to quit)
```



monodepth_main

```
total number of samples: 22600
total number of steps: 141250
number of trainable parameters: 31600072
batch    100 | examples/s: 46.38 | loss: 1.29628
batch    200 | examples/s: 23.63 | loss: 0.93945
batch    300 | examples/s: 38.22 | loss: 1.39224
batch    400 | examples/s: 71.07 | loss: 1.19049
batch    500 | examples/s: 56.66 | loss: 1.03726
batch    600 | examples/s: 33.92 | loss: 0.82971
batch    700 | examples/s: 43.44 | loss: 0.94626
batch    800 | examples/s: 30.11 | loss: 1.18408
batch    900 | examples/s: 43.55 | loss: 0.98870
batch   1000 | examples/s: 48.30 | loss: 1.12154
batch   1100 | examples/s: 30.08 | loss: 0.92985
batch   1200 | examples/s: 39.03 | loss: 1.18913
```

total_loss



disp_left_est_0

disp_left_est_0/image/0
step **30,300**
Sun May 15 2022 01:18:21 GMT+0800 (China Standard Time)

disp_left_est_0/image/1
step **30,300**
Sun May 15 2022 01:18:21 GMT+0800 (China Standard Time)

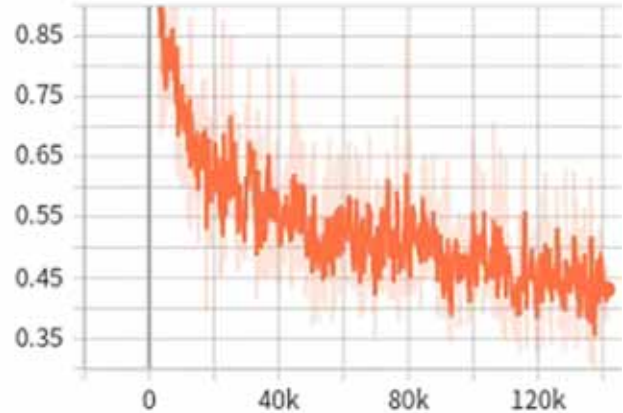
disp_left_est_0/image/2
step **30,300**
Sun May 15 2022 01:18:21 GMT+0800 (China Standard Time)

disp_left_est_0/image/3
step **30,300**
Sun May 15 2022 01:18:21 GMT+0800 (China Standard Time)

Run: monodepth_main

```
batch 140000 | examples/s: 62.21 | loss: 0.54178
batch 140100 | examples/s: 64.62 | loss: 0.39178
batch 140200 | examples/s: 65.39 | loss: 0.45855
batch 140300 | examples/s: 63.90 | loss: 0.37658
batch 140400 | examples/s: 66.53 | loss: 0.47281
batch 140500 | examples/s: 58.32 | loss: 0.44091
batch 140600 | examples/s: 64.27 | loss: 0.40123
batch 140700 | examples/s: 64.51 | loss: 0.41477
batch 140800 | examples/s: 66.72 | loss: 0.43439
batch 140900 | examples/s: 65.24 | loss: 0.56784
batch 141000 | examples/s: 56.24 | loss: 0.49516
batch 141100 | examples/s: 56.33 | loss: 0.38600
batch 141200 | examples/s: 67.18 | loss: 0.50702

Process finished with exit code 0
```

total_loss



disp_left_est_0

disp_left_est_0/image/0
step **141,200**
Sun May 15 2022 05:16:53 GMT+0800 (China Standard Time)

disp_left_est_0/image/1
step **141,200**
Sun May 15 2022 05:16:53 GMT+0800 (China Standard Time)

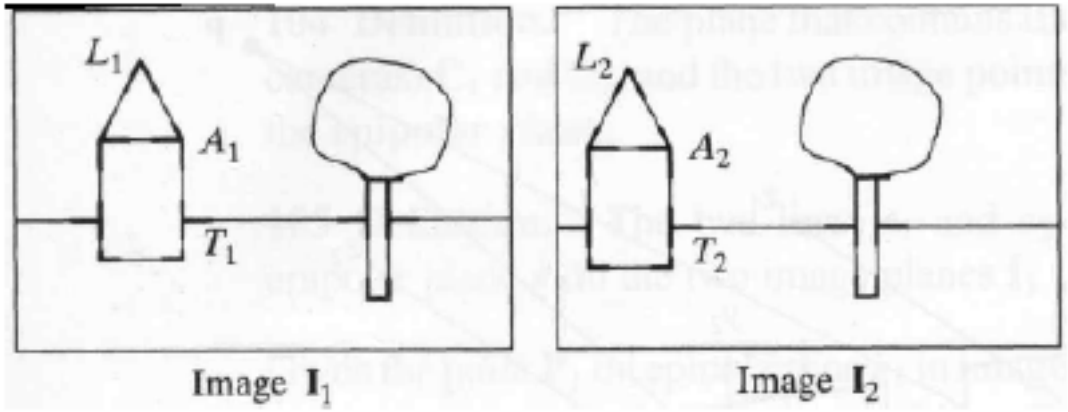
disp_left_est_0/image/2
step **141,200**
Sun May 15 2022 05:16:53 GMT+0800 (China Standard Time)

disp_left_est_0/image/3
step **141,200**
Sun May 15 2022 05:16:53 GMT+0800 (China Standard Time)

9.3.3 基于特征的立体匹配

基于特征的立体匹配

- 主要思想
  - 在左右两幅图像中寻找匹配特征
  - 常用特征有：
    - 边缘点
    - 线段
    - 角点



基于特征的立体匹配

- 匹配算法
  - 在立体图对中抽取特征
  - 定义相似度
  - 利用相似度和极线几何寻找匹配

Inputs:

(1) I_l and I_r

(2) features and their descriptors in both images

(3) the search region in the right image $R(f_l)$ associated with a feature f_l in the left image

For each feature f_l in the left image:

1. Compute the similarity between f_l and each image feature in $R(f_l)$
2. Select the right-image feature f_r, that maximizes the similarity measure.
3. Save the correspondence and disparity $d(f_l, f_r)$

特征匹配和相关匹配比较

•相关匹配方法

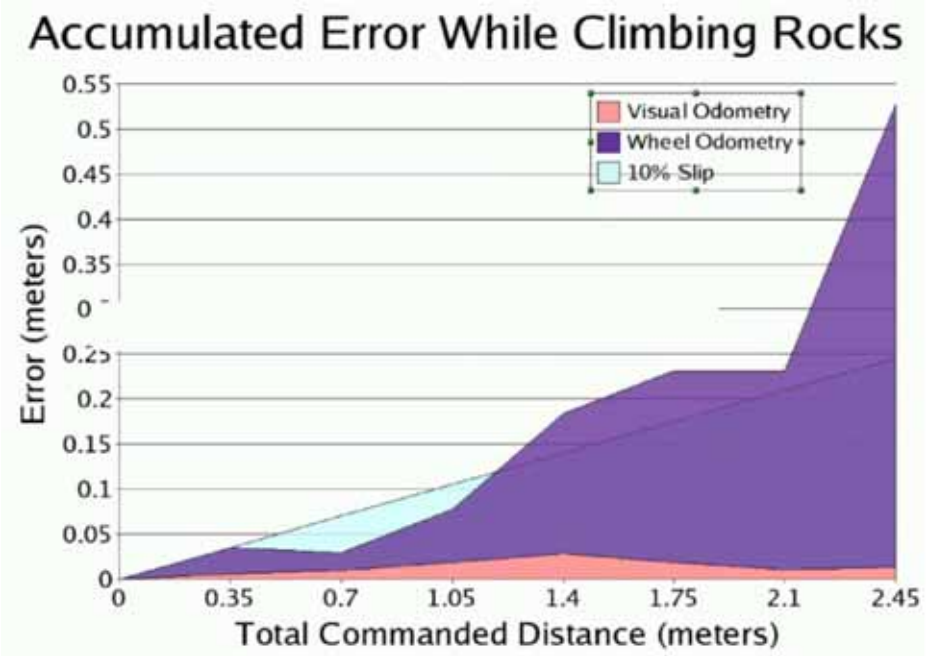
- 易于实现
- 对纹理丰富的图像有良好匹配性能，反之误匹配较多
- 可获得致密视差图（用于表面重建）
- 当视点差异较大时，难以正确匹配，这是由于：
  - –光照方向发生变化
    - –违反了朗伯散射假定

•特征匹配方法

- •运算速度比相关匹配方法快
- •适用于易于提取特征点的场景
- •获得稀疏视差图，适用于视觉导航等应用
- •对亮度变化相对不敏感

特征匹配应用——视觉里程计

- 机器人定位技术是自主导航系统中的关键技术之一，是实现长距离精确导航的前提和保障。
- 传统定位技术采用航位推算法，结合车轮编码器及惯导设备所获得的距离和方向信息来估算自身姿态和位置。
- 车轮编码器无法克服车轮打滑、差动驾驶或制动行驶时引起的计数或测量错误。
- 惯导设备会随时间“漂移”。



Two Years of Visual Odometry on the Mars Exploration Rovers

https://onlinelibrary.wiley.com/doi/pdf/10.1002/rob.20184

美国JPL实验室提出了视觉里程计辅助定位技术，并成功应用于“勇气号”和“机遇号”火星车。

特征匹配应用——视觉里程计

•原理：前、后两帧图像中提取匹配点对，并计算得到同一场景点在前、后两车体坐标系下的三维坐标，则有：

$$\begin{pmatrix} x'_{wi} \\ y'_{wi} \\ z'_{wi} \\ 1 \end{pmatrix} = \begin{pmatrix} \mathbf{R} & \mathbf{t} \\ \mathbf{0}^{\mathrm{T}} & 1 \end{pmatrix} \begin{pmatrix} x_{wi} \\ y_{wi} \\ z_{wi} \\ 1 \end{pmatrix}$$

•通过估计旋转矩阵**R**和平移矢量**t**，从而可获得车体当前帧相对于前一帧的位置和姿态。

•采用欧拉角描述旋转矩阵R

•**优点**：仅需三个角度即可表示旋转矩阵**R**。

•**缺点**：存在万向轴锁问题；由于是非线性求解，增加算法的复杂度。

•**采用四元数表示旋转矩阵R**

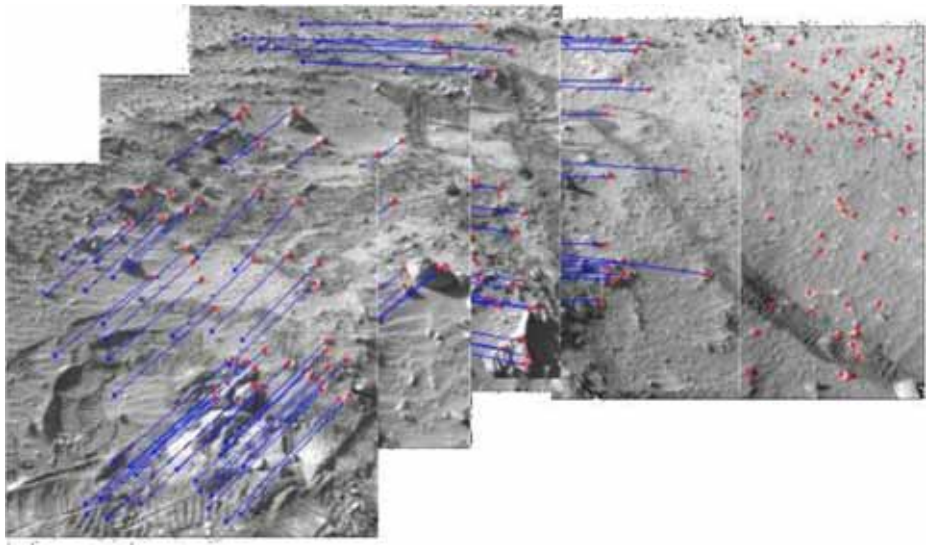
•**四元数的基本形式**：$\mathbf{q} = q_0 + q_1 i + q_2 j + q_3 k$

•旋转矩阵的四元数表示：

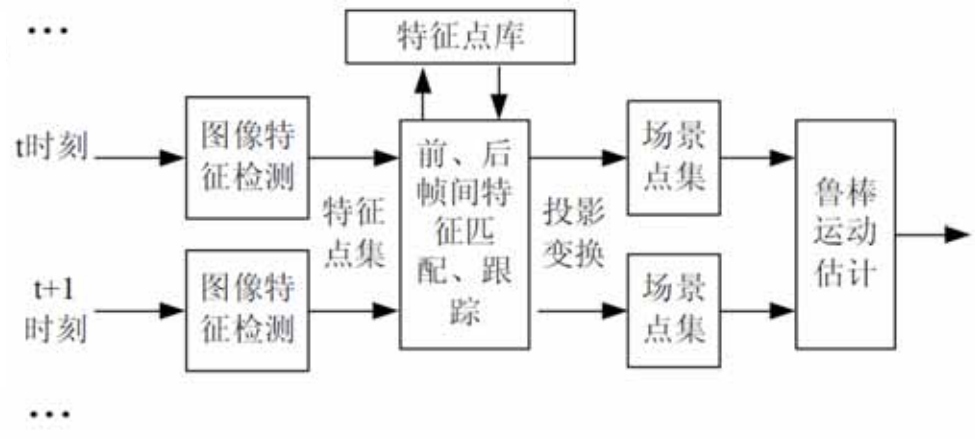
$$R(\mathbf{q}) = \begin{pmatrix} q_0^2 + q_1^2 - q_2^2 - q_3^2 & 2(q_1 q_2 - q_0 q_3) & 2(q_1 q_3 + q_0 q_2) \\ 2(q_1 q_2 + q_0 q_3) & q_0^2 + q_2^2 - q_1^2 - q_3^2 & 2(q_2 q_3 - q_0 q_1) \\ 2(q_1 q_3 - q_0 q_2) & 2(q_2 q_3 + q_0 q_1) & q_0^2 + q_3^2 - q_1^2 - q_2^2 \end{pmatrix}$$

特征匹配应用——视觉里程计

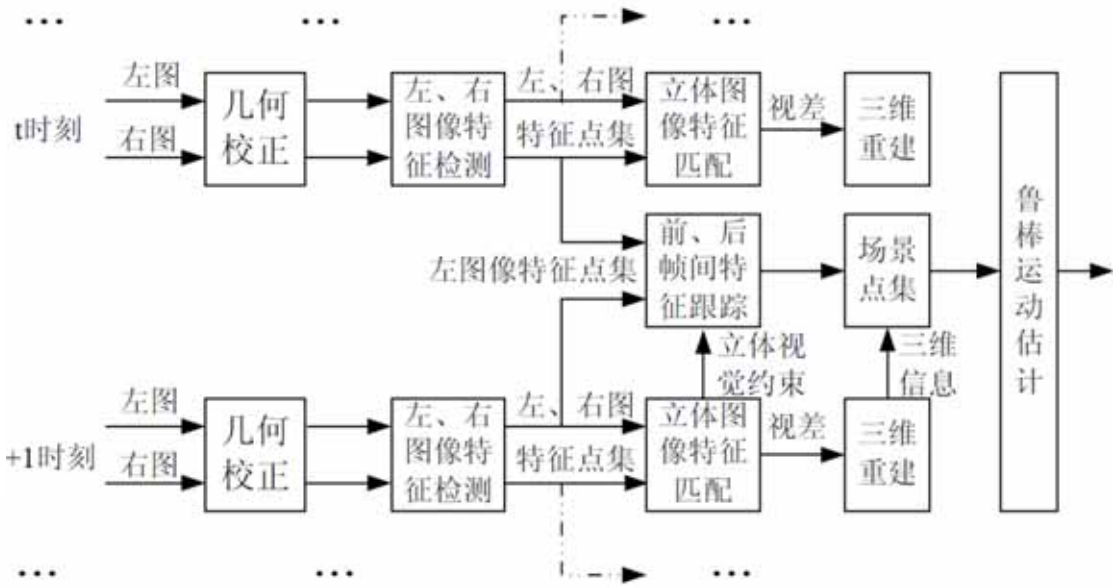
•**基本步骤**：特征点提取、特征点匹配、特征点跟踪、鲁棒的运动估计。



单目视觉里程计

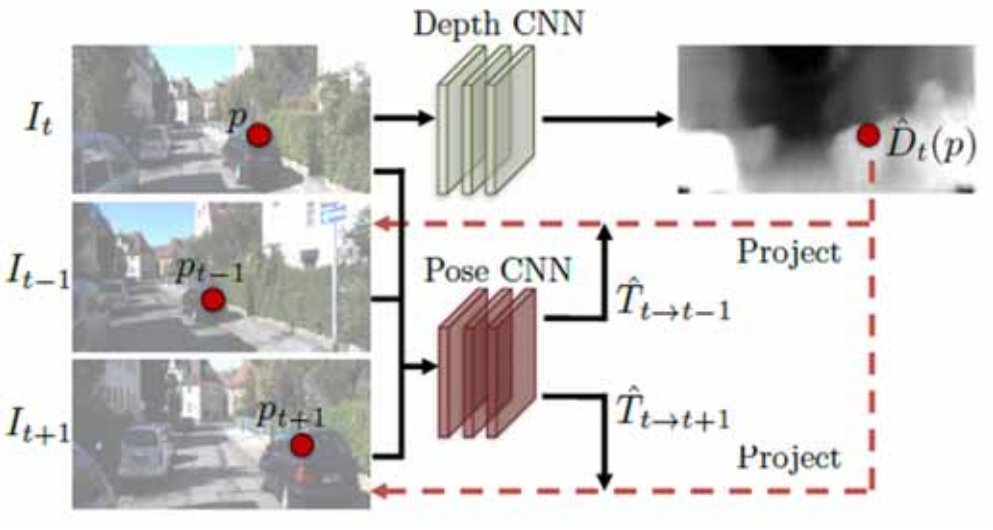


双目视觉里程计



特征匹配应用——视觉里程计

•基于无监督深度学习的视觉里程计



T. Zhou, M. Brown, N. Snavely, and D. G. Lowe, Unsupervised learning of depth and ego-motion from video. CVPR2017.

https://people.eecs.berkeley.edu/~tinghuiz/projects/SfMLearner/

$$\begin{bmatrix} x_i \\ y_i \\ z_i \\ 1 \end{bmatrix} = \begin{bmatrix} \mathbf{R}_{3\times3} & \mathbf{T}_{3\times1} \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} x_{i+1} \\ y_{i+1} \\ z_{i+1} \\ 1 \end{bmatrix}$$

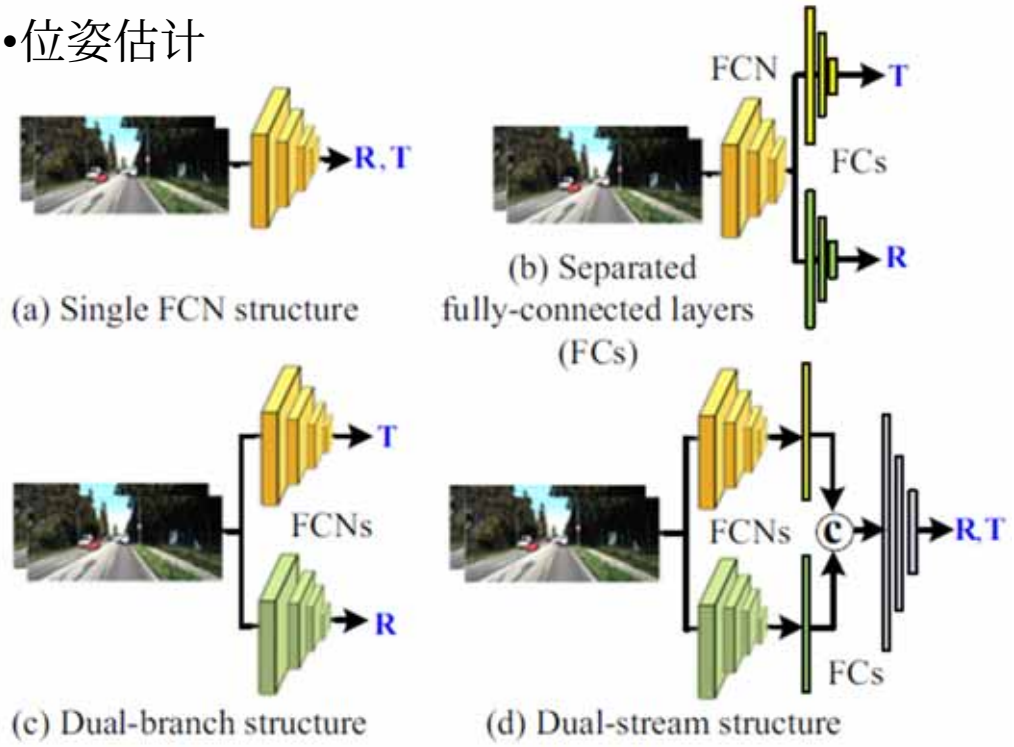
转为视点合成

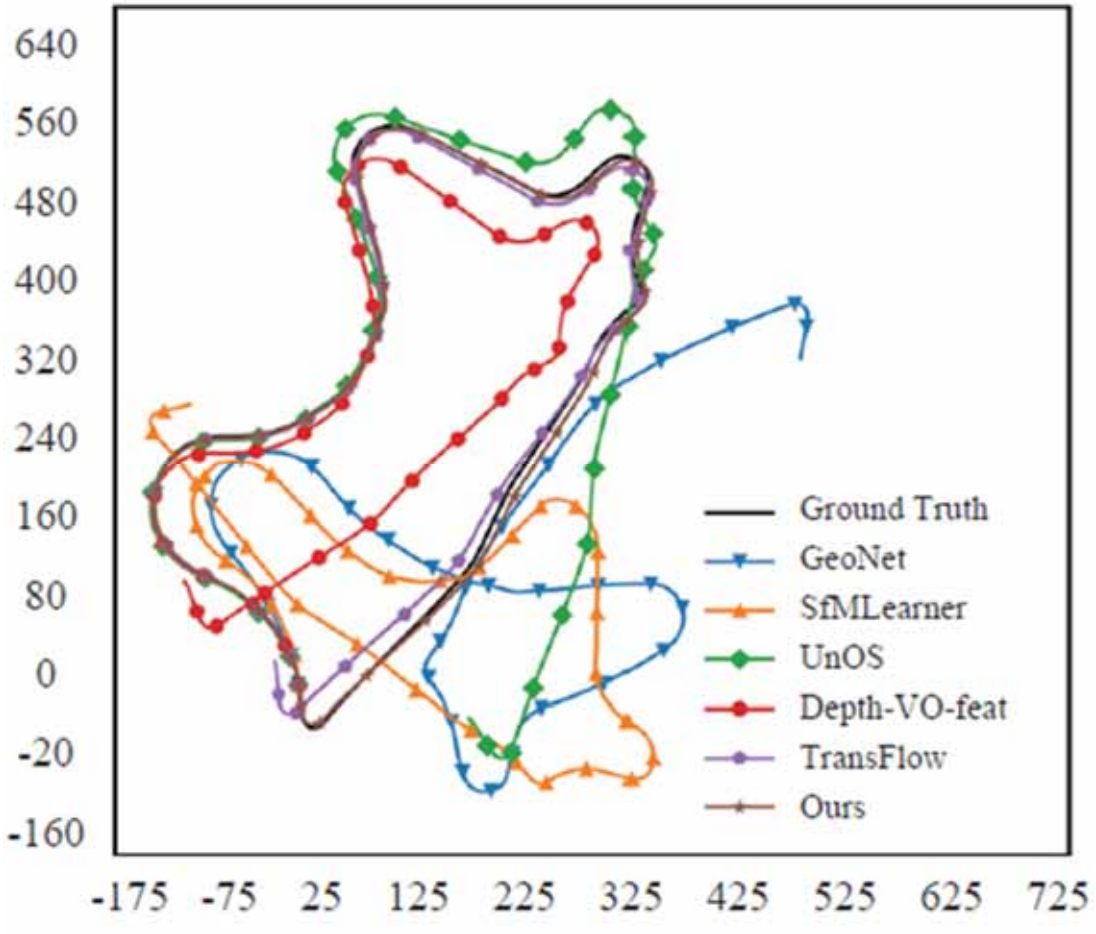
$$z_i \begin{bmatrix} u_i \\ v_i \\ 1 \end{bmatrix} = z_{i+1} \left[\mathbf{K} \mid \mathbf{0}\right]_{3\times4} \mathbf{P}_{i+1\to i} \begin{bmatrix} \mathbf{K}^{-1} \\ \hline \mathbf{0} \end{bmatrix}_{4\times3} \begin{bmatrix} u_{i+1} \\ v_{i+1} \\ 1 \end{bmatrix}$$

$$p_s \sim K\hat{T}_{t\to s}\hat{D}_t(p_t)K^{-1}p_t$$

特征匹配应用——视觉里程计

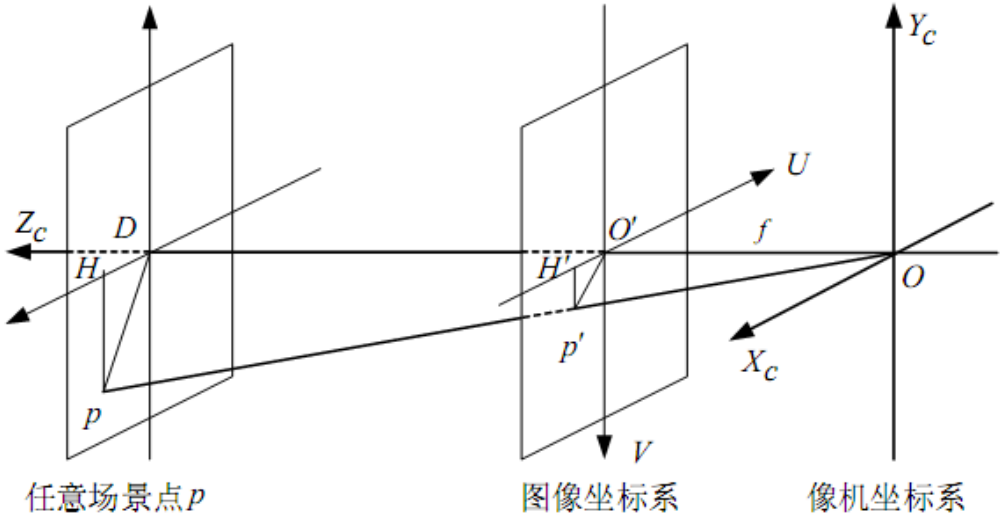
•位姿估计





9.4 三维重建

三维重建的几何关系



$$\frac{OO'}{OD}=\frac{O'H'}{DH}=\frac{Op'}{Op}=\frac{p'H'}{pH},$$

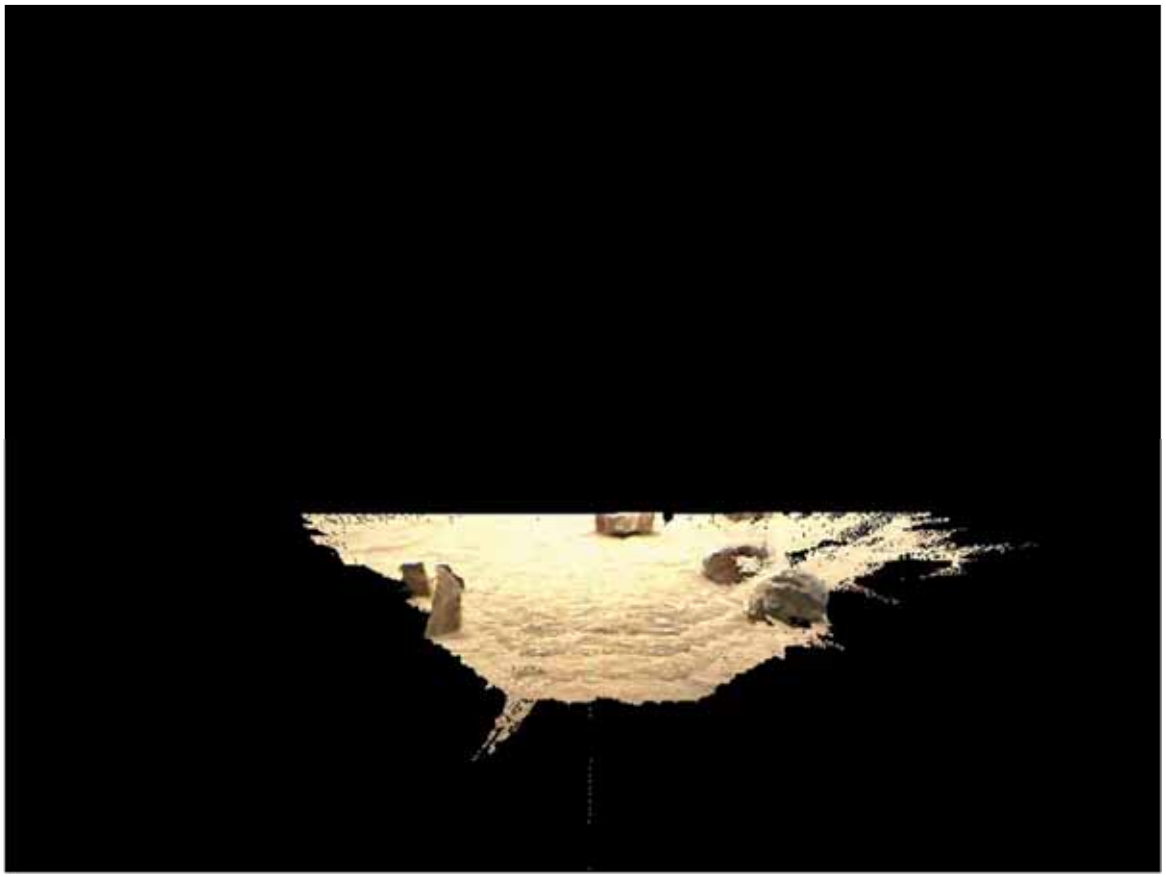
即有：

$$\begin{cases} X_c=\dfrac{b(u-U_0)}{d} \\ Y_c=\dfrac{b(V_0-v)}{d} \\ Z_c=\dfrac{bf}{d} \end{cases}$$

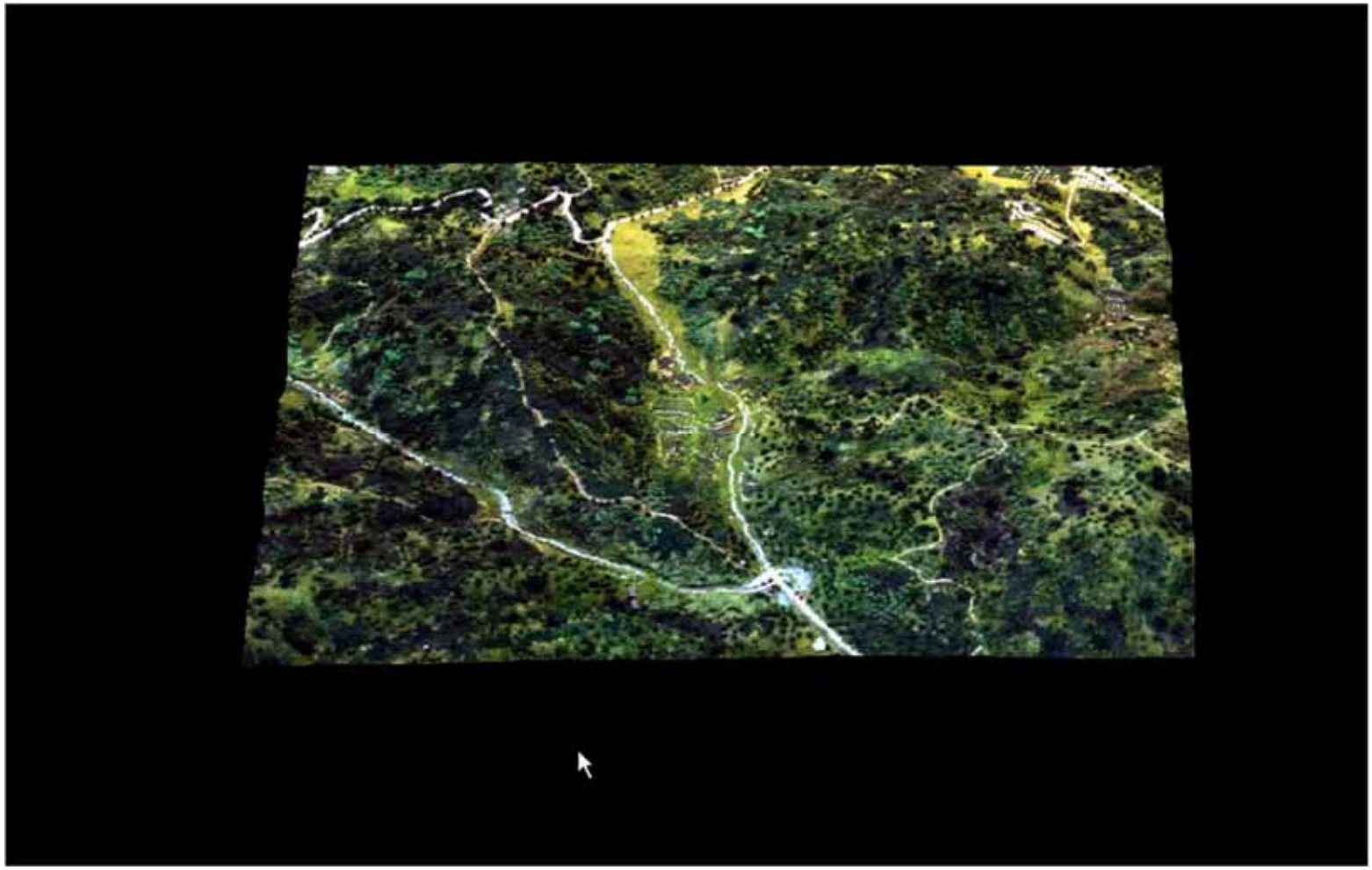
$$\frac{f}{Z_c}=\frac{u-U_0}{X_c}=\frac{v-V_0}{-Y_c}$$

仿真环境双目三维重建

三维点云表示



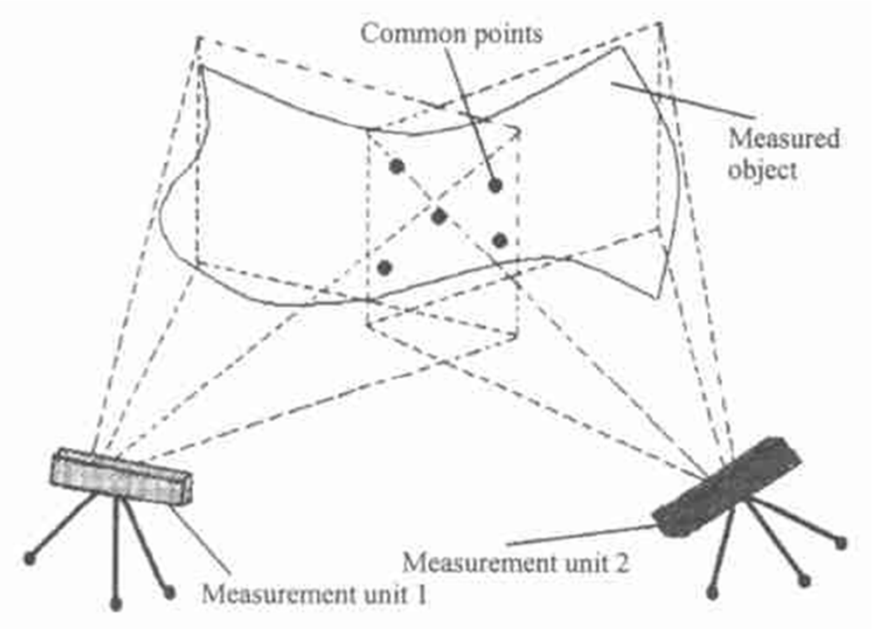
高分辨率双目三维重建



双目像机三维场景拼接

•对于大场景的三维重建，由于系统视场有限，需要对不同视角下的三维模型进行人工拼接。

•三维拼接技术的实质是把在不同的局部坐标系中测量得到的有效数据点云进行坐标变换。

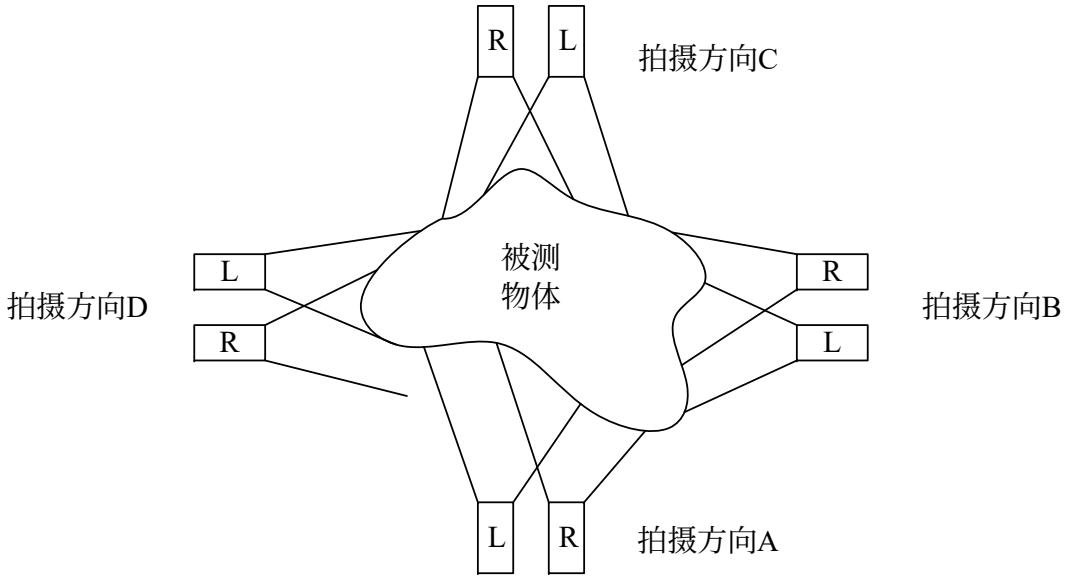


双目像机三维场景拼接

•输入图像必须有足够的特征信息

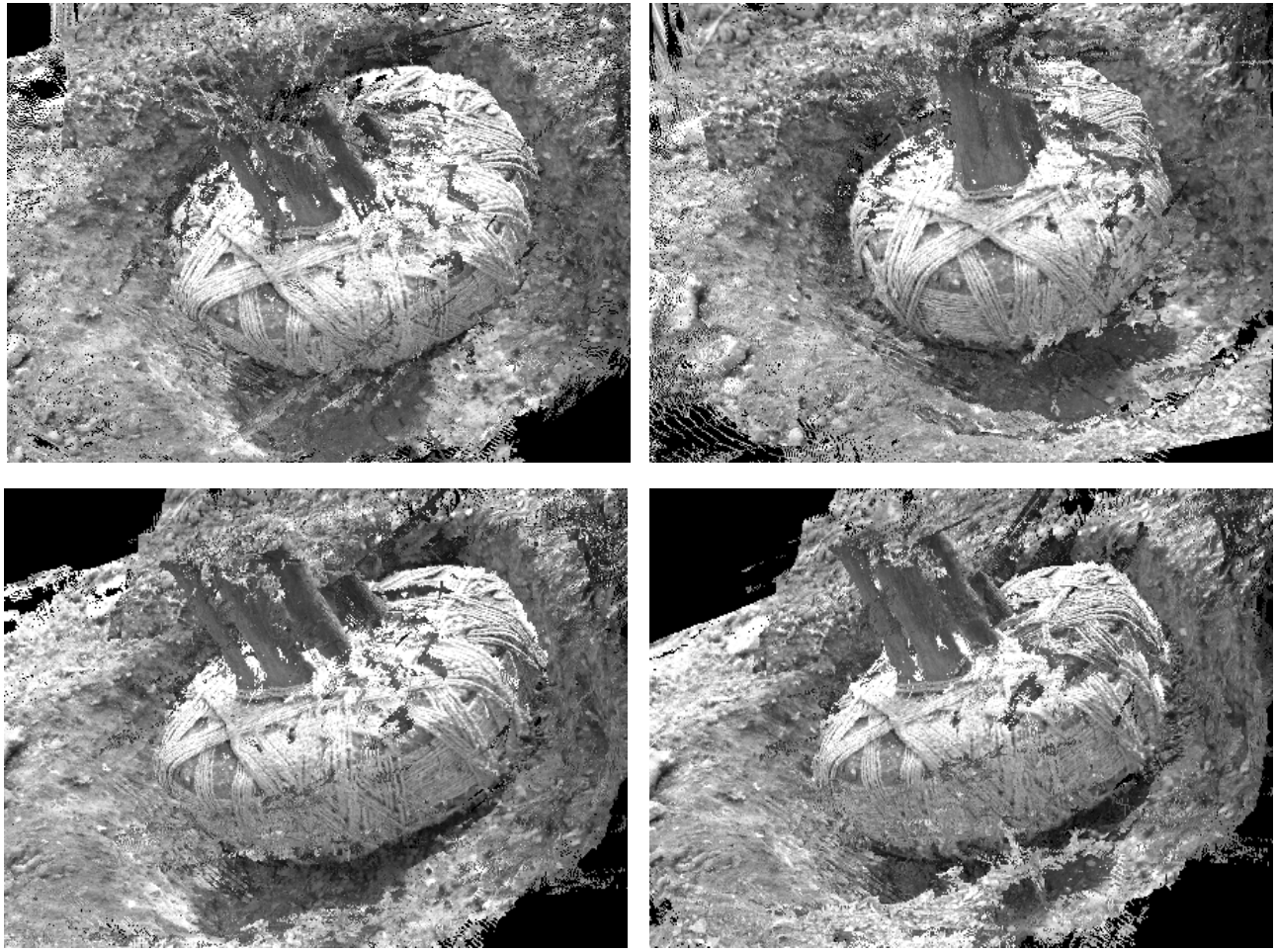
•前后帧之间应保证一定的重叠率，一般选取在60%～90%为佳。

•尺度和旋转变化不易太大。

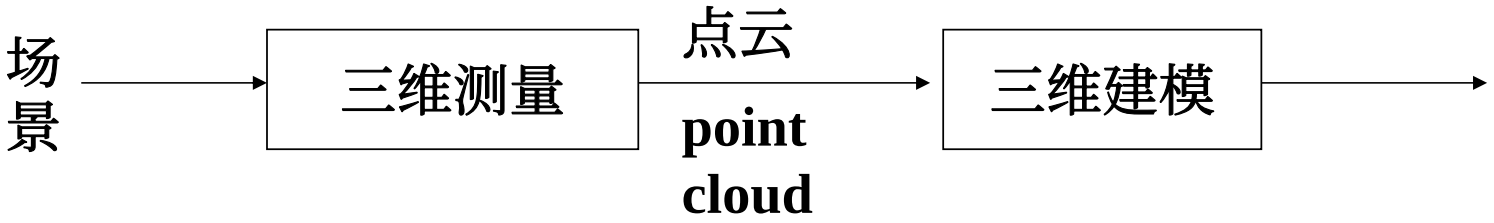


双目像机三维场景拼接

野外树坑



单目像机三维重建



单目像机三维重建

Building Rome in a Day

Video

https://www.cs.cornell.edu/~snavely/bundler/
https://www.cs.cornell.edu/projects/bigsfm/

- S. Agarwal, N. Snavely and I. Simon, et al. Building Rome in a Day. in International Conference on Computer Vision. 2009. Kyoto, Japan.
- J. Frahm, P. Georgel and D. Gallup, et al. Building Rome on a Cloudless Day. in 11th European Conference on Computer Vision. 2010.
- Y. Furukawa, B. Curless and S.M. Seitz, et al. Towards Internet-Scale Multi-View Stereo. CVPR. 2010.

NeRF

Block-NeRF (CVPR2022)

https://waymo.com/intl/zh-tw/research/block-nerf/

UniSim (CVPR2023 Highlight)

https://waabi.ai/unisim/

Gaussian Splatting

杭电7教

杭电10教

写在课程的最后

CCD作为计算机视觉的物质基础
计算机视觉从此获得蓬勃发展

Who are they?

•加拿大物理学家博伊尔（Willard Boyle）和美国科学家史密斯（George E. Smith），因发明数码相机图像感应器“感光半导体电荷耦合器件”(CCD)，2009年10月连同“光纤之父”高锟，荣获诺贝尔物理学奖。

Rome is not built in a day!

A good beginning is half done!