

Lecture 4 - Stereo Reconstruction

- Lect 3 的内容主要关于，如何利用稀疏的对应关系，从两个或多个视角进行 3D 重建
- 本章内容主要关于稠密立体重建

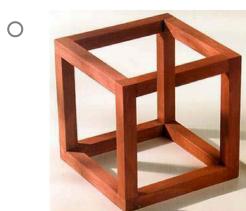
Sparse reconstruction VS Dense reconstruction

在 3D 重建技术中，稀疏重建和稠密重建是两种目标不同的重建策略

1. 稀疏重建主要关注图像中的特征点（如角点、边缘点等），通过一些特征提取算法（如 SIFT、SURF、ORB）等，将这些特征点从图像中检测出来，然后通过匹配和 Triangulation 的方法计算出对应的三维坐标；稀疏重建的数据量小、计算速度比较快、而且点云比较稀疏，无法完整的描述物体的表面细节和纹理信息。主要的应用包括初步的场景结构理解、相机定位（如视觉 SLAM 中的定位模块）
2. 稠密重建旨在恢复完整的 3D 场景，包括物体的表面细节和纹理信息，利用图像的全部像素，通过深度估计、立体匹配、多视角立体（VMS）等算法，生成高密度的三维点云或网格模型；数据量大、计算复杂度高，但是点云的细节较为丰富，适合精细化的三维建模和渲染

4.1 Preliminaries

- 我们如何从 image 还原 3D 呢？
 - 一般来说，我们在考虑这个问题的时候，会遇到下面几种情况
 - Occlusion (遮挡)



- 当一个物体被另一个物体部分或完全遮住，导致其在图像中部分可见或完全不可见，会导致视觉错觉
- Parallax (视差)



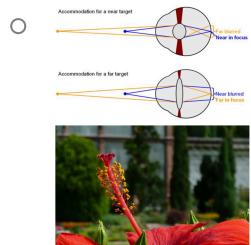
- 由于观察者位置的变化导致物体在视野中的相对位置发生变化的现象
- 例如图中，路灯和月亮与其在水面的倒影，看起来外观差不多，但其景深肯定是差很多的

■ Perspective (透视)



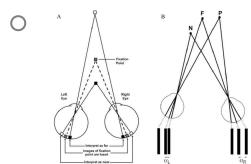
- 物体有时在二维图像中会表现出三维空间的深度感，最基础的便是“近大远小”的规则

■ Accommodation (调节)



- 在人类视觉系统中，通过肌肉改变晶状体的形状，从而聚焦不同距离的物体
- 而在计算机视觉中，我们需要模拟这种调节焦距的机制，从而感知或生成不同焦点的图像

■ Stereopsis (立体视觉)



- 立体视觉是指人类利用双眼视差感知深度的能力
- 双眼看到的图像略有不同，大脑将这些差异整合，形成对三维空间的感知
- 在CV中，立体视觉计数通过从2个或多个视角，匹配对应特征点，计算视差，从而还原三维结构 (triangulation)

● Binocular VS Monocular

- 在历史上和自然界，基本不存在只有1只眼睛的生物（有一些生物具有单只感光点，不完全算作眼睛），一些单只眼睛的人只在神话或动画当中

出现



Stalk-Eyed Fly



Stereoscopic Rangefinder

- 由 Triangulation 的原理可知，当我们从 two-view 中还原立体信息时，两个 view 之间的距离越远，还原效果越好
- 这样的技术也被用于军事中，例如立体测距仪

经过研究，两只眼睛是能够健全地还原 3D 信息的最低配置，这也是为什么大多数生物最后进化的结果是两只眼睛

- Two-View Stereo Matching



Left Image

Right Image

Disparity Map

- 目标：从两幅图中计算出每一对像素之间的 **disparity**，进而生成视差图(disparity map)
- 可以看到，这里给出的两张图不是完全静态的，甚至不是同一时间拍摄的，但是大部分场景是可以还原的

由于这里 **disparity** 和 **parallax** 都翻译为视差，但是实际定义不太一样，因此用英文表示

Disparity (视差)

指同一场景中的物体在左右两个摄像机（人眼）的图像中出现的水平位置的差异

在立体视觉系统中，计算对应点在左、右图像中的位置差（通常以像素为单位），这个差值就成为 **disparity**

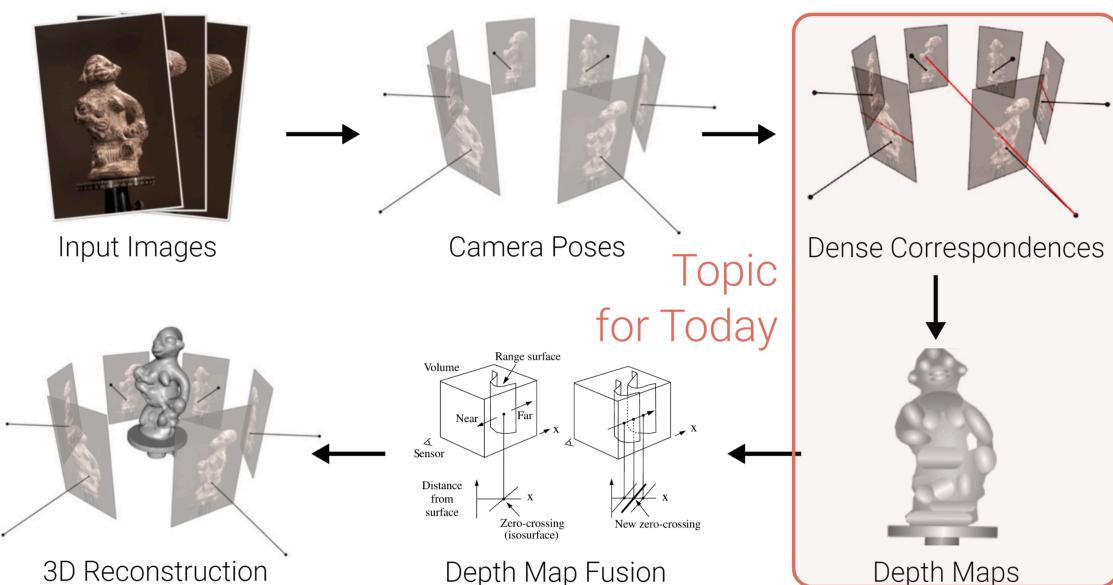
通过计算整个图像的视差，可以得到视差图 (disparity map)

Depth (深度)

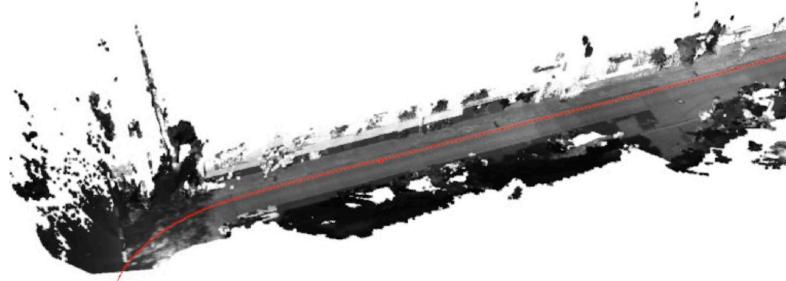
视差和深度之间存在反比关系：视差越大，物体离摄像机越近；视差越小，物体离摄像机越远，用公式表示为： $Z = \frac{f \times B}{d}$ ，其中 Z 为 depth、 f 为焦距、 B 为左右摄像机（或人眼）的基线距离， d 为视差

景深(Depth of Field)

是指图像中清晰成像的前后景物之间的距离范围

- 目标
 - 从静态（或部分静态的）two-view 中构建一个稠密 3D 模型
- Pipeline
 1. Calibrate cameras intrinsically and extrinsically (参见 Lect 3.1)
 2. Rectify images (using given calibration)
 3. Compute disparity map for reference image.
 4. Remove outliers using consistency/occlusion test.
 5. Obtain depth from disparity using camera calibration.
 6. Construct 3D model, e.g. via volumetric fusing and meshing.(Lect 8.4)
- The entire pipeline to 3D reconstruction
 - 

- 首先，一组输入的图像，然后利用 Lect 3 中提到的 Incremental Bundle Adjustment 来确定每个相机的 pose，并且确定了哪些相机是彼此相邻的，从而计算出 dense 的对应关系，使用 Epipolar geometry 来分析每一组相邻的图像，从而得到 $\frac{N}{2}$ 组深度图（假设总共有 N 个图像），并且用一些融合算法来将这些深度图处理为一个完整连贯的 3D 模型，甚至在此基础上提取出网格结构

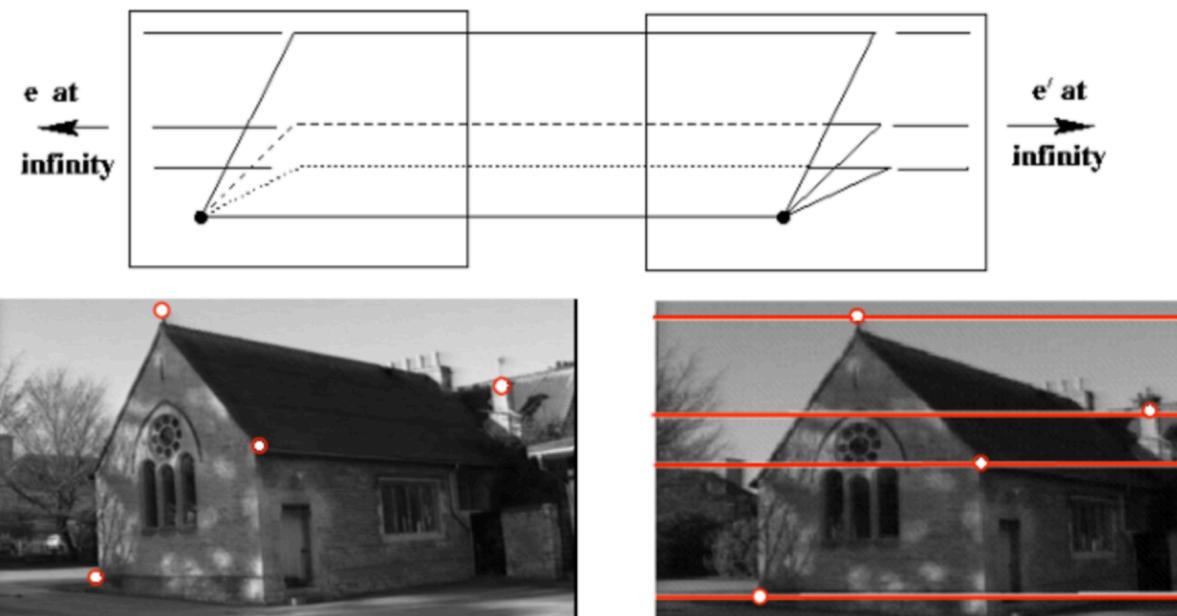


- 上图是一个很古老的，稠密重建的成果例子 (camera 的位置贯穿图中的红色线条)

4.1.1 Epipolar Geometry

这里不再对 Epipolar Geometry 的定义做回顾，如果有需要请查看 [Lect 3 的3.2.1小节](#)

4.1.2 Image Rectification

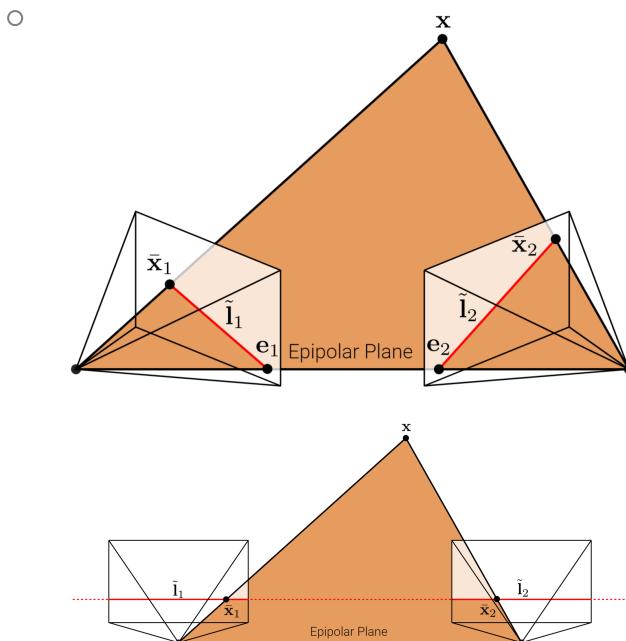


- 如果两个 view 刚好是从同一角度拍摄的 (也即它们的成像平面是平行的) , 该怎么办?
 - 可以想象到, 如果我们建立 Epipolar 平面, 那么基线 (baseline) 将不会穿过成像平面, 极点 (epipoles) 会趋向于无穷远的两侧, 所有的极线 (epipolar lines) 也会是平行的
 - 因此, 和正常建立了 Epipolar 平面的做法一样, 我们只需要按行扫描图像来寻找特征点匹配即可 (由于不需要考虑斜线, 也即不需要遍历各种角度, 实际上这种做法还是很快捷高效的)

这个做法听起来很理想，我们甚至都不需要建立 Epipolar 结构，只需要按行扫描两幅图即可——但是在现实中，我们实际上很难得到完全平行的两个 view

由于匹配算法对于 Epipolar line 的精度要求很高——可能目标像素相差一行，结果就会完全不一样；因此，摄像、机械的误差会带来很大的问题，我们需要以像素级别的精确度或亚像素级别的精确度为目标

- 那么如果图像不是按照我们理想的设置建立的，该怎么办？



- 有一个办法：我们可以以穿过相机中心的重垂线（与地面垂直）为轴，旋转相机平面，将它们映射到与基线平行的公共平面上，这个过程称为 **Rectification**
 - 由于这个旋转是以相机中心为轴的，因此我们不需要知道 3D 结构就可以做这一变换操作
- 我们怎么使双极线 (epipolar lines) 保持水平？
 - 我们先来看一个简单的情况，假设相机参数矩阵、校正矩阵、旋转矩阵都等于单位矩阵 \mathbf{I} ，并且两个 image 之间的平移向量，仅仅是 x 轴方向上的 t 个单位距离
 - 也即令 $\mathbf{K}_1 = \mathbf{K}_2 = \mathbf{R} = \mathbf{I}$, $\mathbf{t} = (t, 0, 0)^T$
 - 在这个例子中，基础矩阵可以由下式计算
 - \$\$

$$\tilde{\mathbf{E}} = [\mathbf{t}] \times \mathbf{R} = [t] = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & -t \\ 0 & t & 0 \end{bmatrix}$$

– 根据Lect3当中所讲的，这里我们其实应该由 $\bar{x}_i \times \mathbf{K}^T$ 得到对应的校正值，

$$\begin{aligned} \bar{x}_2^T \tilde{\mathbf{E}} \bar{x}_1 &= \bar{x}_2^T \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & -t \\ 0 & 0 & 0 & 0 & t & 0 \\ 0 & 0 & 0 & -t & 0 & 0 \end{pmatrix} \bar{x}_1 \\ &= \bar{x}_2^T \begin{pmatrix} 0 & 0 & 0 & 0 & 0 & -t \\ 0 & 0 & 0 & 0 & t & 0 \\ 0 & 0 & 0 & -t & 0 & 0 \end{pmatrix} \bar{x}_1 = t - t = 0 \end{aligned}$$

– 其中 y_i 表示 x_i 向量的第2个分量值 – 因此，我们得到了 $y_1 = y_2$ 的结论，

- 那么我们怎么进行这种 Rectification 操作呢？

- 我们需要计算得到 rectifying 旋转矩阵 $\mathbf{R}_{rect} = (\mathbf{r}_1, \mathbf{r}_2, \mathbf{r}_3)^T$ ，其中

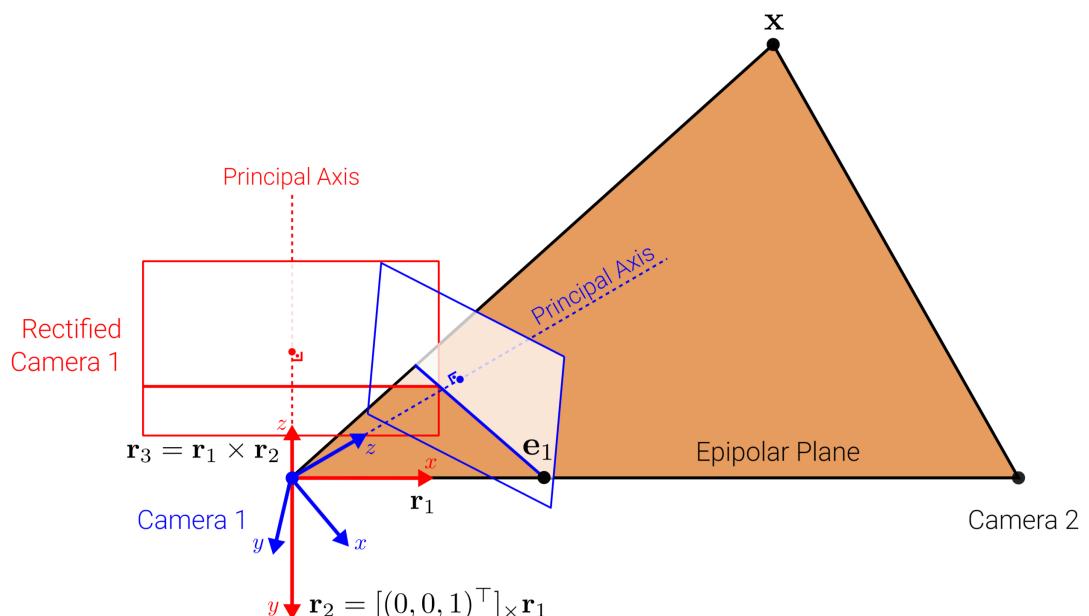
$$\mathbf{r}_1 = \frac{-\mathbf{R}^T \mathbf{t}}{\|\mathbf{R}^T \mathbf{t}\|_2}, \quad \mathbf{r}_2 = [(0, 0, 1)^T] \times \mathbf{r}_1 / \|[(0, 0, 1)^T] \times \mathbf{r}_1\|_2, \quad \mathbf{r}_3 = \mathbf{r}_1 \times \mathbf{r}_2$$

- Rectification Algorithm

1. 计算 $\tilde{\mathbf{E}}$ ，将其分解出 \mathbf{t} 和 \mathbf{R} ，然后按照上面的公式构建 \mathbf{R}_{rect}
2. 将第一个 image 中的像素按照 $\tilde{x}'_1 = \mathbf{K} \mathbf{R}_{rect} \mathbf{K}_1^{-1} \bar{x}_1$ 扭曲
3. 将第二个 image 中的像素按照 $\tilde{x}'_2 = \mathbf{K} \mathbf{R}_{rect} \mathbf{R}^T \mathbf{K}_2^{-1} \bar{x}_2$ 扭曲

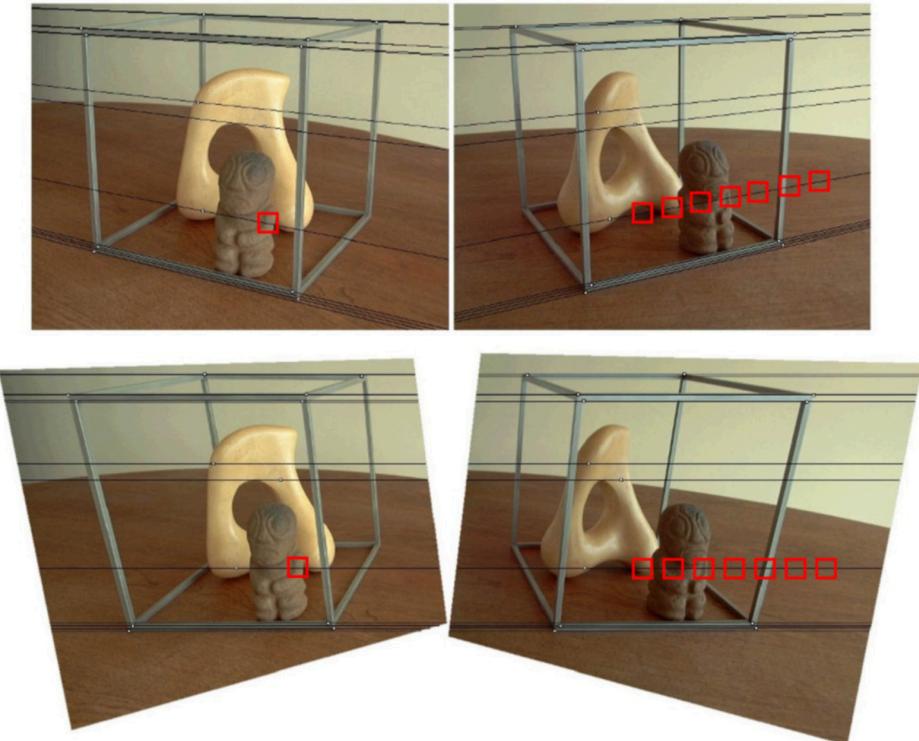
- 注意

- \mathbf{K} 是一个共享的矩阵，它必须相等，当然我们可以主观的选择，一般会选择和其中一个 calibration 矩阵相近的矩阵（例如令 $\mathbf{K} = \mathbf{K}_1$ ）
- 在实际应用中，我们一般不会以原始图像为主体去计算虚拟图像（Rectification 后的图像），因为主题旋转后总有一些像素是会丢失的；因此，我们实际用到的其实是这个公式 inverse 后的，并且在欠缺原始像素时，采用一些插值算法作为补充



- 上图是一个 Rectification 应用的例子

- 首先计算 x 轴（它一定与基线平行），然后对应的计算 y 轴和 z 轴的位置，也即按照 \mathbf{r}_1 、 \mathbf{r}_2 、 \mathbf{r}_3 的顺序，计算 \mathbf{R}_{rect} ，并且应用旋转变换，从而得到新的映射平面



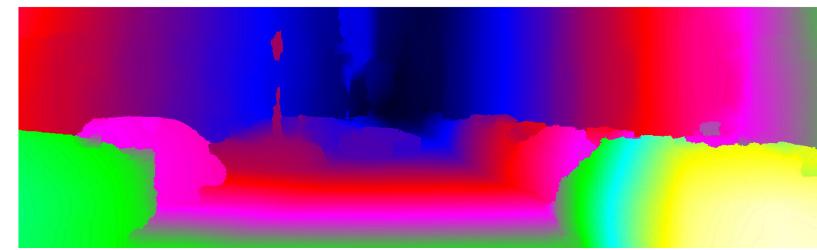
- ○ 这是应用 Rectification 前后的 image 示意图，可以看到对应特征都在同一行对应的像素上

4.1.3 Disparity to Depth

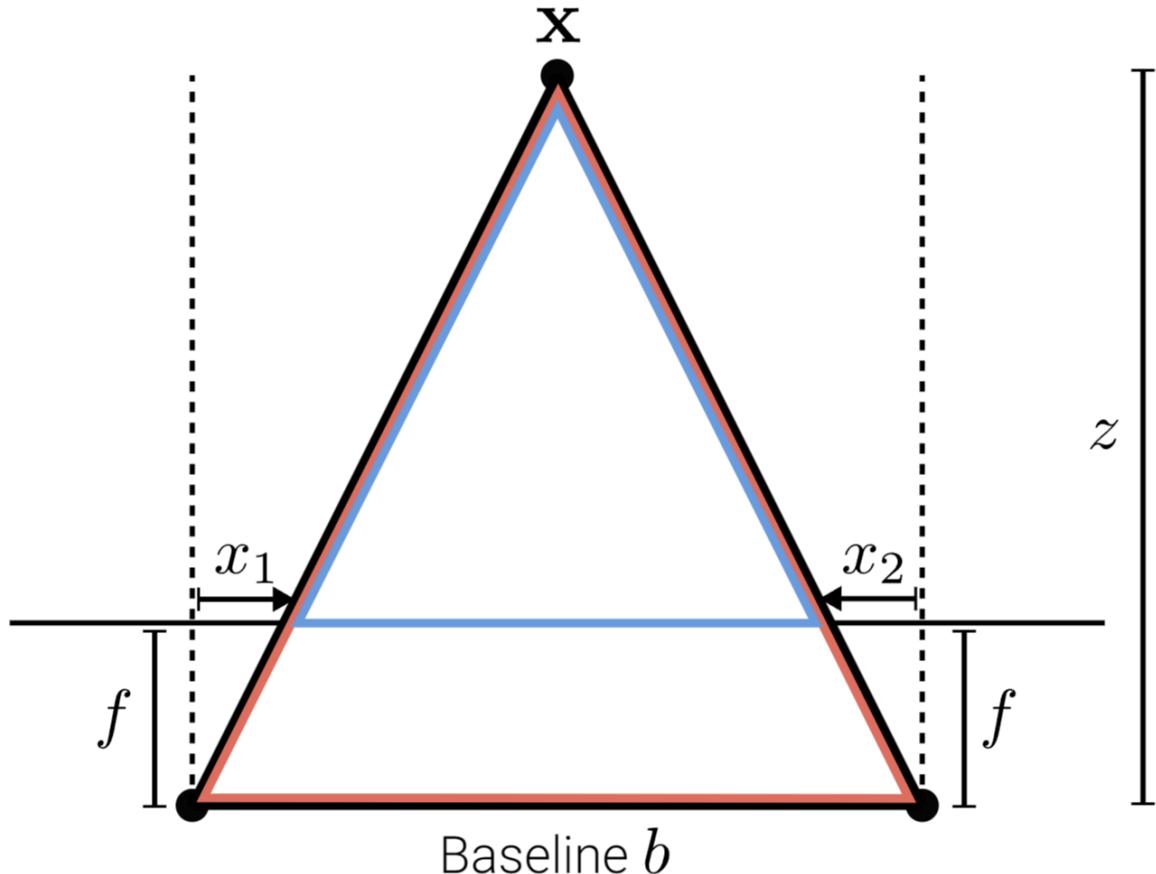
- Disparity 评估



- ■ 如果你仔细比对上面两张图，可以发现相机的位置发生了很小的变化，而对应的点在远处移动的距离很小，在近处移动的距离却很大，这种现象称为 disparity（视差）
 - 通过这种规律，我们可以估计像素点的深度



- 而这种方法要求两幅变换的图是水平变化的，因此它也是 Rectification 的一个应用
- 具体怎么从评估的 disparity 中还原深度呢？



- 上图是一个鸟瞰图 (BEV)，因此我们只画出 x 轴和 z 轴
- 由于两个 image 都位于 epipolar 平面，因此左边和右边的光线一定是会相交于 3D 点的
- 假设 disparity 是 $d = x_1 - x_2$ (注意，以图中来说 x_2 其实是负值)
- 根据图中的三角关系，我们可以得到等式 $\frac{z-f}{b-d} = \frac{z}{b}$
- 其中 $\frac{z-f}{b-d}$ 表示蓝色三角形中的比例， $\frac{z}{b}$ 表示橘色三角形中的比例，比值表示三角形的高比底边的一半
- 从而可以得到 $zb - fb = zb - zd$, 也即 $z = \frac{fb}{d}$
- 因此，我们可以得到 depth z 是与视差 d 成反比的

这里可以得到一个事实——估计误差 Δz 与 z 的值成正比，也即越近的物体估计越准，越远的物体估计越不准，证明详见这一小节的 exercise

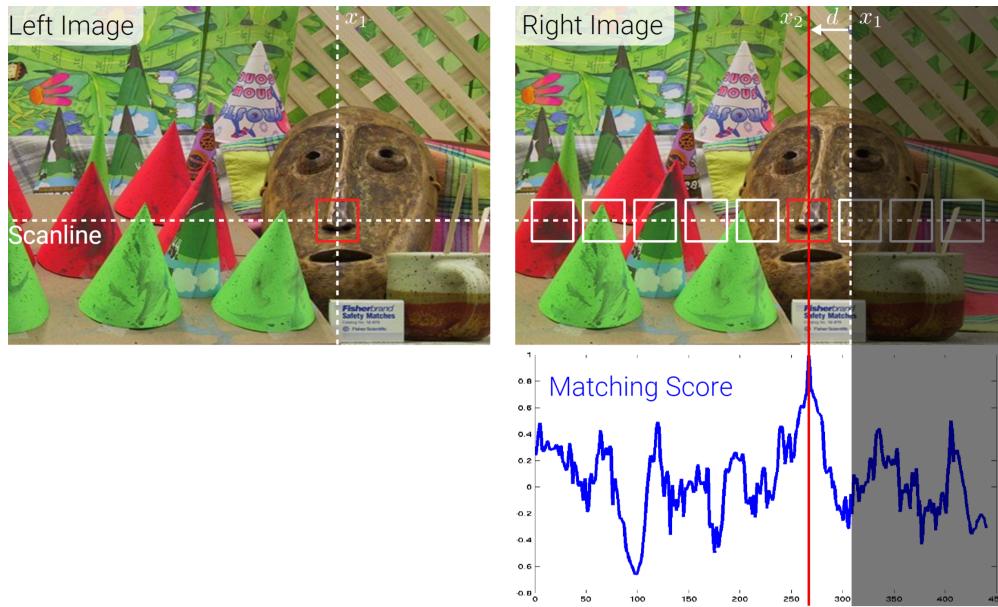
4.2 Block Matching

- 当我们做完 Rectification 后，要逐行扫描 image 来找到特征匹配的位置，但是我们如何判断两个位置是匹配的呢？或者说，我们认为它们应该是“相似的”，如何定义这种“相似性”呢？

- Three images of a squirrel's tail. The first two images show the tail from different angles, with red boxes highlighting specific regions for comparison. The third image shows the squirrel's head and tail, with a red box highlighting a region on the tail.
- 实际上，只是比对两个像素，即使是人眼也无法区分是否表示同一物体——因此，我们一般会取一个 patch，然后以这个 patch 为单位来扫描

- Similarity Metrics





- 实际过程中，我们会采用一些相似度的 metric 算法来评估两幅 image 之间的相似性，并取相似性最大点的 patch (当然，我们不排除有歧义的情况，也即有多个最大值相等的情况)

注意，一般来说，我们将视野较左的 image 作为 left image，并且以 left image 为中心，考虑 right image 的 disparity disparity 的值本身是正值，但是 right image 对应点需要往负半轴寻找



- 令 patch 为 $K \times K$ 个像素大小，并且左、右 image 中的 patch 分别用向量 $\mathbf{w}_L, \mathbf{w}_R \in R^{K^2}$ 表示（也即一个 K^2 长度的向量，分量的位置都一一对应）
- 对于 metric 算法，我们肯定希望这两个向量越相似越好
- 有很多的 similarity metric 算法，其中最常用的两种：
- Zero Normalized Cross-Correlation (ZNCC)
- \$\$

$$\text{ZNCC}(x,y,d) = \frac{(\bar{w}_L(x,y) - \bar{w}_L(x-d,y))^T (\bar{w}_R(x,y) - \bar{w}_R(x-d,y))}{\|(\bar{w}_L(x,y) - \bar{w}_L(x-d,y))\|^2 \|(\bar{w}_R(x,y) - \bar{w}_R(x-d,y))\|^2}$$

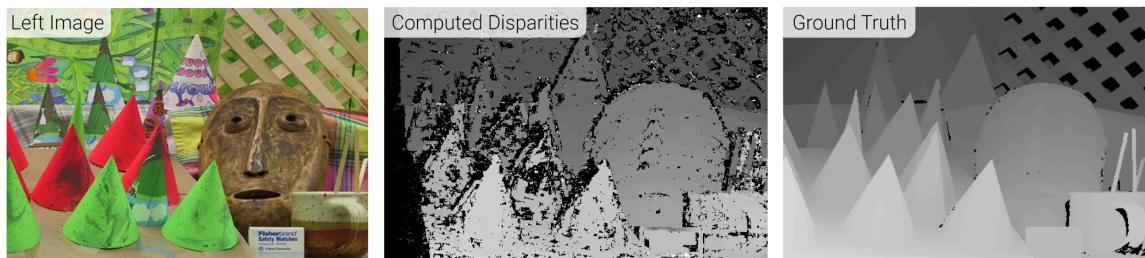
— **Sum of squared differences (SSD) ** —

$$\text{SSD}(x,y,d) = \|(\bar{w}_L(x,y) - \bar{w}_R(x-d,y))\|^2$$

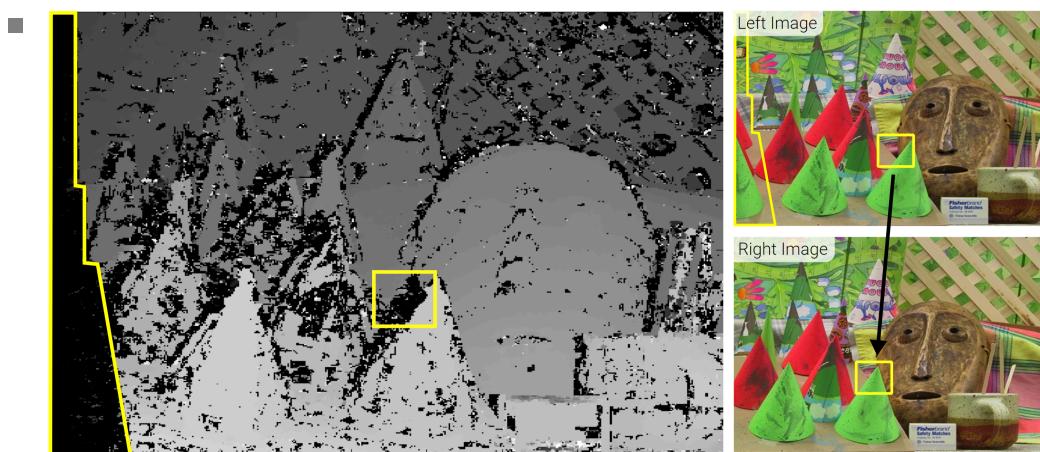
> ** ZNCC ** (零均值归一化互相关) 适用于光照和对比度变化较大的场景

其余的 metric 算法不在这里给出，可以参考文献：Hirschmuller and Scharstein: Evaluation of Stereo Matching Costs on Images with Radiometric Differences. TPAMI, 2009.

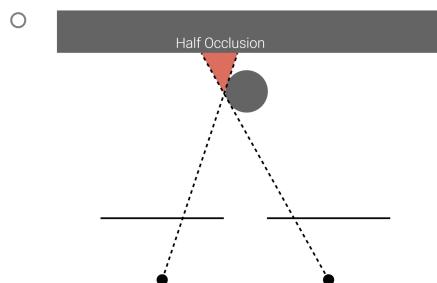
- Block Matching 算法过程



- ■ 首先选择一个 disparity 范围 $[0, D]$
- 对于所有的
- Block Matching 中的几个问题
 - Half Occlusions



- 我们可以看到，生成的 depth 图中，除了噪声，还有一些伪影的区域
 - 左侧的黑色区域：这部分区域在 right image 中是不可见的
 - 中间的黑色区域：这部分区域在 right image 中被遮挡了
- 这些只有一个 image 中可见的区域，我们称为半遮挡 (half occlusions) 区域



- Assumption Violations
 - 到目前为止，我们的计算都建立在一个假设上——right image 的对应点都可以由 left image 的坐标减去 disparity d 来得到
 - 但是实际上，这个假设非常容易被打破——它只适用于与相机平面平行的物体平面，因此这个假设称为 fronto-parallel 假设



- 可以看到图中的红色 patch，它是一个不满足 fronto-parallel 假设的区域，左侧 patch 中黑色的线条在右侧 patch 中被压缩了，因此它并不能直接通过减去 disparity 来得到

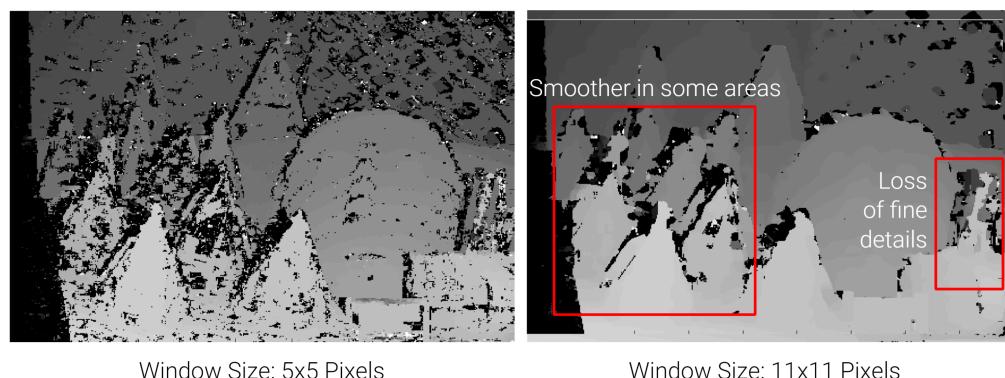
- Disparity Discontinuities



- ○ 可以看到，从 left image 到 right image 过程中，图中黄色的纸条和背景的布袋的移动距离并不一样——当 disparity 不连续的时候，patch 内的像素也不会以同样的 d 值偏移

- Effect of Window Size

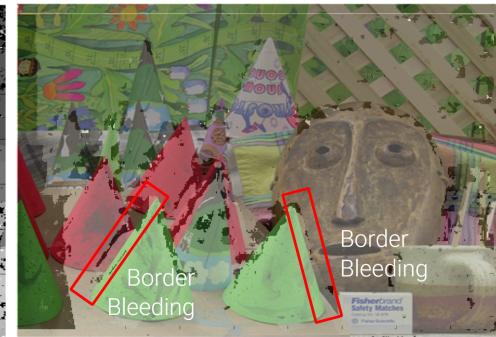
- 由上面的几个问题，我们可以考虑，选择 patch 的大小对于 depth 图的生成效果也会有很大的影响



- ○ 由上图可以看到，如果 patch_size 选择得较小，则噪声会比较严重，但是细节比较丰富；如果 patch_size 选择得较大，则图像会比较平滑，但是细节丢失比较严重

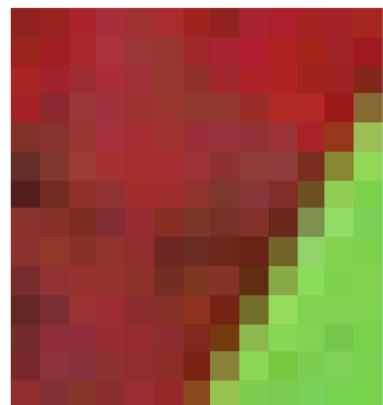


Window Size: 5x5 Pixels



Window Size: 11x11 Pixels

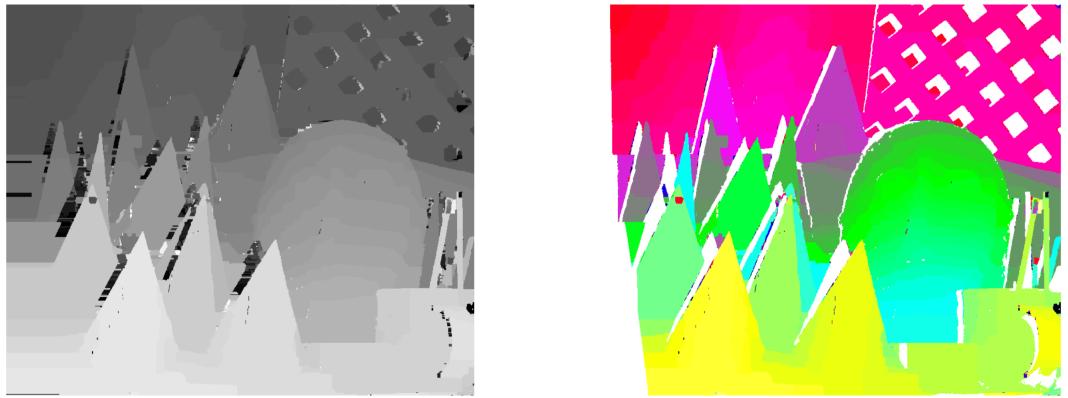
- - 我们可以注意一个细节——图重圈出的区域，前景的边界并不明显——前景的部分像素出现了“出血状”的伪影，渗透进了后景的区域，我们将这个现象称为 Border Bleeding；
 - 当选择的 patch_size 过大，并且违反了 fronto-parallel 假设时，就会出现这个现象



- - (这里的区域对应上一张图左侧的红色圈区域)
 - 在这里给出的 patch 比较中，红绿边界是主要特征——我们的算法可以很容易计算出它的优势，并将它所对应的 disparity 值作为主导值——然而，当我们想计算红色的背景区域时，它并没有足够的纹理特征使得我们计算它对应的 disparity，因此算法会认为它也是边界的一部分，从而给予其一样的 disparity，从而将其视作前景的一部分
 - 这个原因，导致了 border bleeding 现象的出现

- Left-Right Consistency Test

- 到此，我们感觉 Block Matching 并不是很好用——它似乎有很多问题
- 不过对于 Block Matching，还有一种方法来修正这些错误——左右一致性检查 (Left-Right Consistency Test)



- - 我们不仅以 left image 为基准生成 depth 图, 还以 right image 为基准生成 depth 图, 并且遍历每一个像素点的 disparity——如果某一点的 disparity 在两个 depth 图中都差不多 (之差小于一个 threshold) , 那么我们认为这点的 disparity 就是这样; 如果差别很大, 那么我们认为这点的 disparity 不对——将其标为无效, 或者应用其它匹配策略, 重新进行匹配
 - 噪声和 half-occlusions 的情况都可以被左右一致性检查算法检测到, 并解决; 但是它会带来额外的开销

4.3 Siamese Networks

- 4.2 节介绍了 hand-crafted 的特征检测以及 similarity metric
 - 然而, 这些 hand-crafted 的方法没有将所有的几何相关性、辐射不变性、遮挡模式都考虑进去
 - 现实需要考虑的情况对于人工来说太过于复杂
 - 一篇文献工作提出——我们可以将 Matching 问题考虑为图像分类问题

Left patch	Right patch	Label
		Good match
		Bad match
⋮	⋮	

- 方法概述
 - 训练 CNN patch-wise 基于 ground truth disparity 图 (例如 [KITTI Stereo](#))

2015)

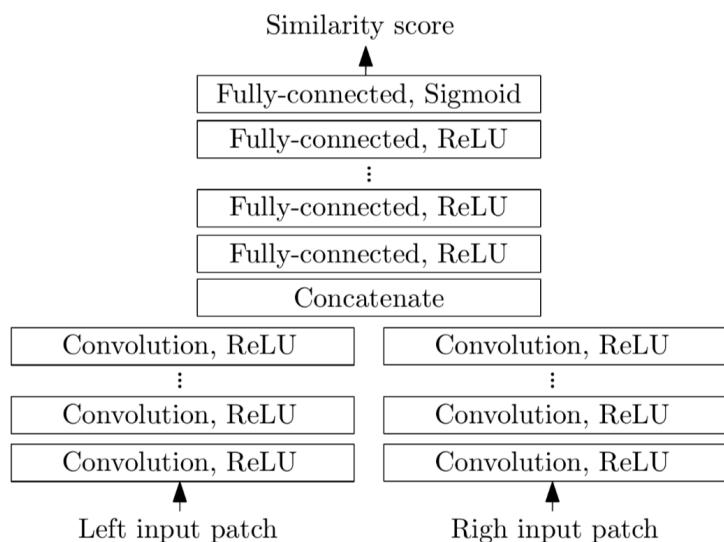


- 利用训练好的模型来为每一对 image 计算特征
- 找到每个像素的最大值 (winner-takes-all) 或者使用全局优化算法

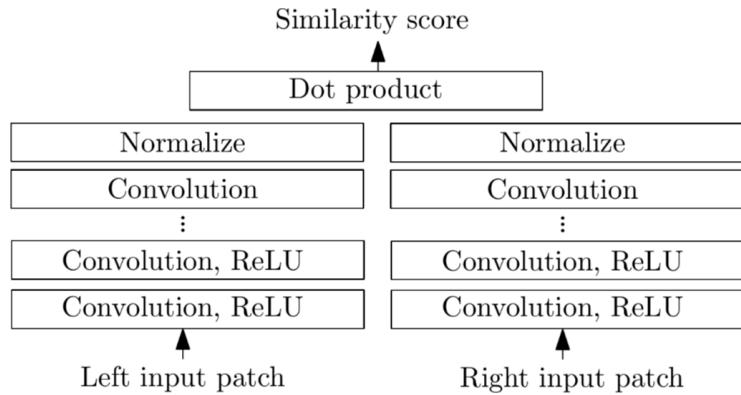
值得一提的是，这个神经网络的思想至今仍然可以使用，因为它只是用于分类 disparity 的 patch，具有很高抽象层次的应用，而 end-to-end 的方法将在最后一小节讲述

- Siamese Network Architecture

- Learned Similarity



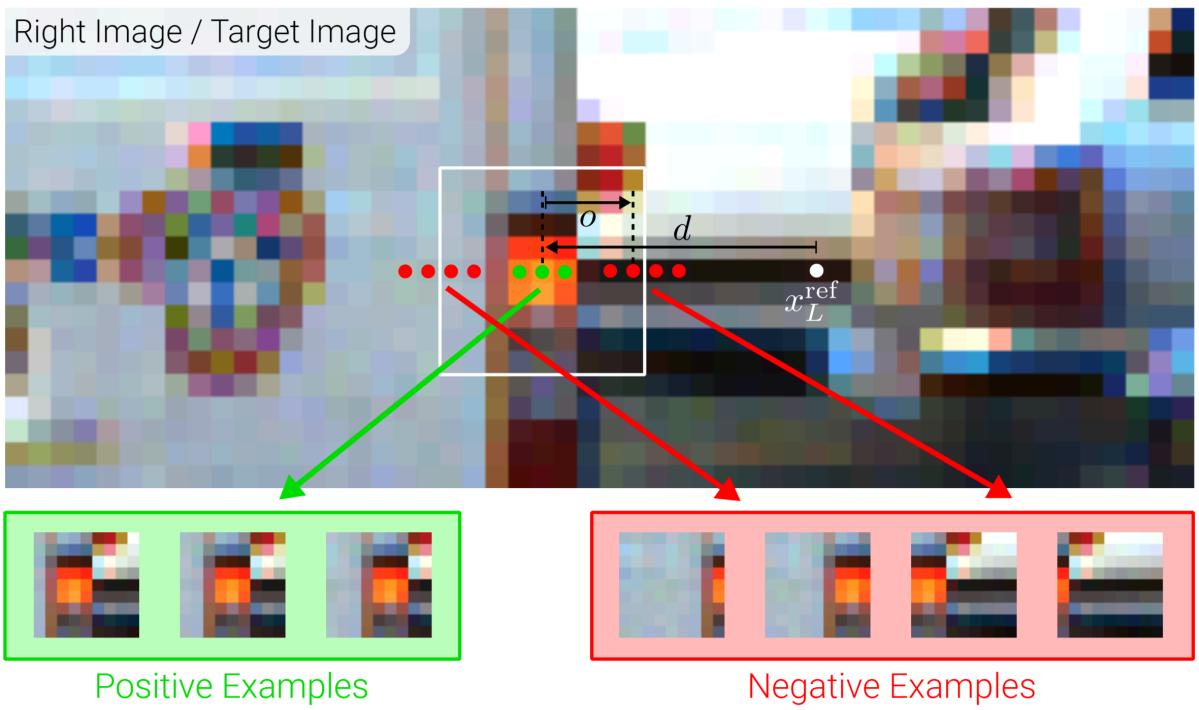
- left image 和 right image 的 patch 分别输入多层卷积来获取特征，然后将它们的特征级联在一起，输入全连接的 MLP (Multi-Layer Perceptron)
 1. 可能更具潜力
 2. 运行很慢
 - Cosine Similarity



- ○ 由于 MLP 是前一个架构的主要计算量的结构，因此我们直接将其去除，而是将 left、right image 卷积得到的特征标准化后，利用点积，计算出 similarity 分数
 1. 功能必须完成繁重的工作
 2. 运行快速

- Training

- 训练集由若干 patch 的 3 元组组成
- $(\mathbf{w}_L(\mathbf{x}_L^{ref}), \mathbf{w}_R(\mathbf{x}_R^{neg}), \mathbf{w}_R(\mathbf{x}_R^{pos}))$
- $\mathbf{w}_L(\mathbf{x}_L)$ 指的是 left image 中以 $\mathbf{x}_L = (x_L, y_L)$ 为中心的 patch
- $\mathbf{w}_R(\mathbf{x}_R)$ 指的是 right image 中以 $\mathbf{x}_R = (x_R, y_R)$ 为中心的 patch
- Negative example
- $\mathbf{x}_R^{neg} = (x_L^{ref} - d + o_{neg}, y_L^{ref})$
- Offsets o_{neg} ，取值于 $U(\{-N_{hi}, \dots, N_{low}, N_{low}, \dots, N_{hi}\})$
- Positive example
- $\mathbf{x}_R^{pos} = (x_L^{ref} - d + o_{pos}, y_L^{ref})$
- Offsets o_{pos} ，取值于 $U(\{-P_{hi}, P_{hi}\})$
- 注意，这里 d 表示一个像素的真正 disparity (由 ground truth 提供)
- 一般来说， $P_{hi} = 1, N_{low} = 3, N_{hi} = 6$
- 这里采用称为 **hard negative mining** 的方法思想
- 为了使机器能够学习的更好，我们希望构造一组更加有难度的例子，也即目标和真实值更接近的样本，因此这里我们会在距离正确样本较小范围内取 positive 样本、在较大范围内取 negative 样本



这里要注意，pos 和 neg 的 offset 取值范围不能重合，事实上，最好它们的界限有一定距离

我们旨在让分类器遇到对应的情况是，认为 positive 的样本都是可以接受的、正确的；negative 的样本都是错误的

- Loss Function

- Hinge Loss

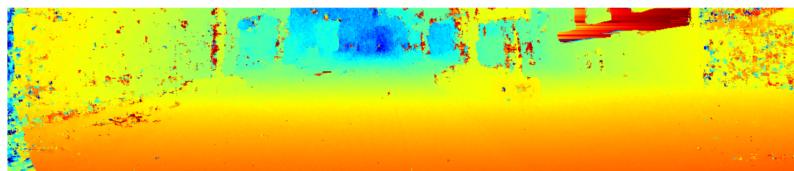
- \$\$

$$L = \max(0, m + s_{-} - s_{+})$$

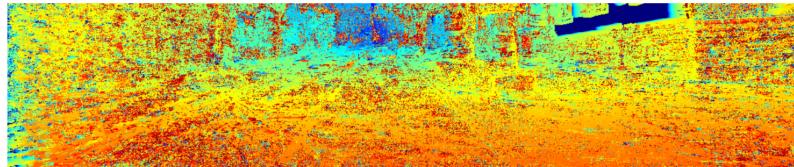
s_{-} / s_{+} 是网络对于 negative/positive 样本的分数 – 当 pos



Left Input Image

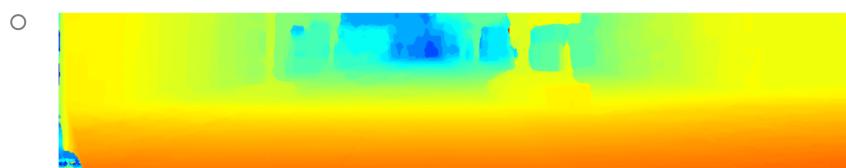


Siamese Network

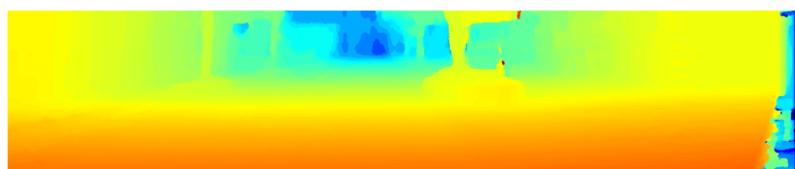


Standard Block Matching

- 上图展示了 Siamese 网络和标准的 Block Matching 运行结果的对比
 - 如果我们使用一种全局优化算法，那么就会得到更好的识别结果，甚至更好的 disparity map，而这正是我们下一小节要讨论的内容



Left Disparity Map



Right Disparity Map



Left-Right Consistency Test

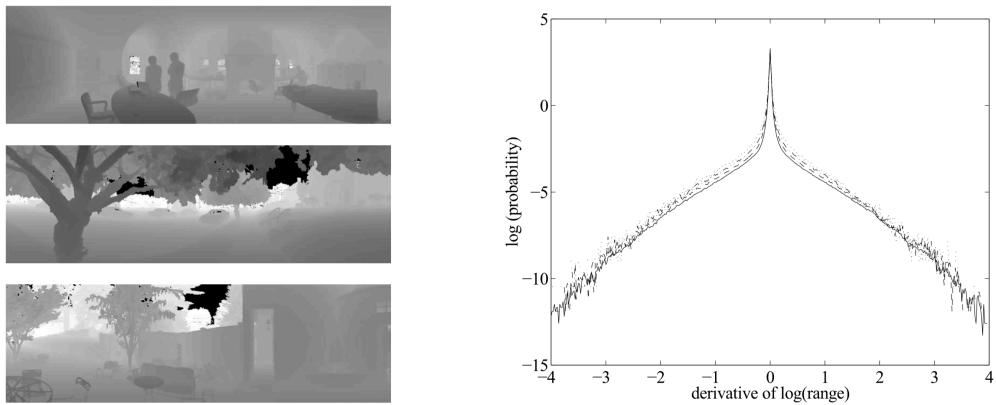
- 原始的 Siamese 网络的运行情况，是在带有 CUDA 的 Lua/Torch7 上运行的，采用 Nvidia GTX Titan GPU，使用 5 个小时训练 (4500 万个样本、16 epochs、128 batch_size)
- 在 1 对 image 上推理花费 6/100 秒

4.4 Spatial Regularization

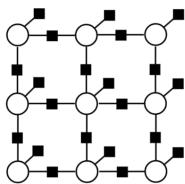
- 我们之前提到的方法，都是基于 local 的 matching，也即选取一个 patch，然后用某种方法（比如 Siamese 网络）计算另一个 image 上所有的 similarity 分数，然后用 winner-takes-all 的思想来选取一个分数最高的，或者匹配成本最低的
- 但是这种 local 的 matching 方法有哪些局限性呢？



- - 当出现很少的纹理或者几乎无纹理的 patch，例如图中近乎纯白的 patch，我们很难将其匹配到对应的位置
 - 当出现重复情况时，我们也很难进行正确的匹配，因为它们彼此之间的相似度就很高
 - 当出现遮挡情况时，左右 image 对应的 patch 的 similarity 分数也不会很高，这种时候很有可能被别的看起来很类似的 patch 错误地匹配
- 因此，我们想用一种全局优化的方式来取代这种局部的匹配
- Spatial Regularization
 - 先来观察一个现实世界存在的特点



- 上图是利用红外测量的现实世界的 depth 图
- 可以看到，depth 的变化（灰度值的变化）在同一个物体的范围内是十分缓慢的，只有在物体不连续的位置会突然有飞跃式的变化
- Stereo MRF
- 指定一个定义在网格上的循环 Markov 随机场
-



- 如上图所示的网格，其中圆圈表示一个像素，方块表示一个 constrain；之前的 local 匹配中，像素（或者说 patch）之间是没有 constrain 的，而我们现在的目的就是给像素之间定义一种 constrain
- 我们的全局优化目标是最小化“能量”，如下式，其中
- p 表示概率，由于它和右侧的指数部分成正比，因此我们想要最小化右侧花括号内的式子，也即最大化概率
-

$$p(\mathbf{D}) \propto \exp\left\{-\sum_i \psi_{data}(d_i) - \lambda \sum_{i-j} \psi_{smooth}(d_i, d_j)\right\}$$

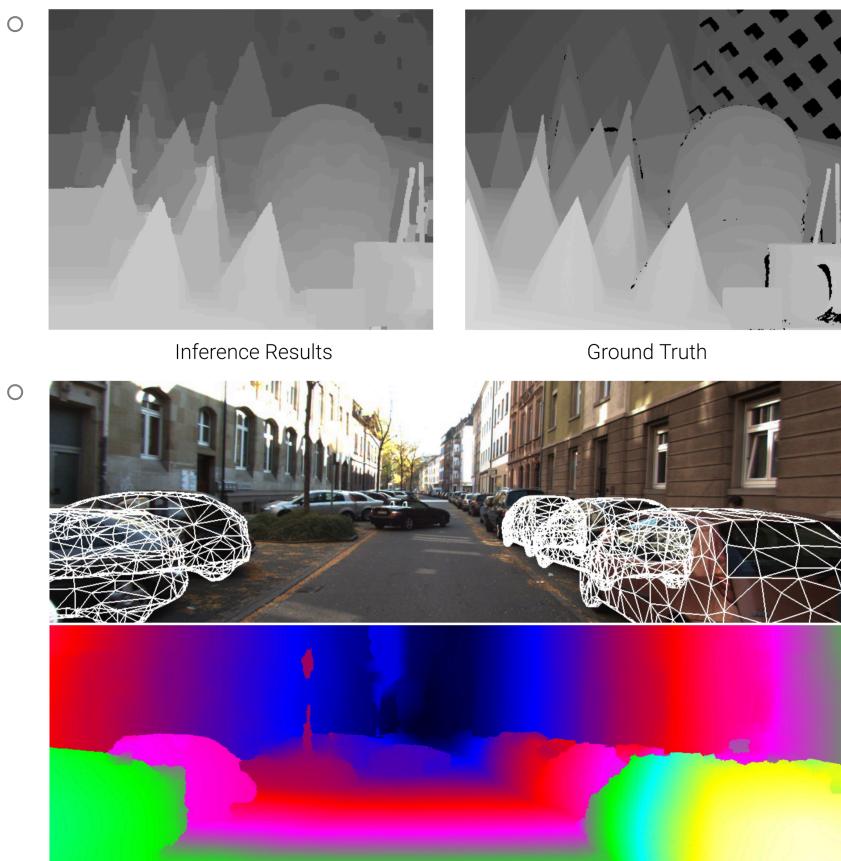
- $i-j$: 与像素 i 相邻的 4 个像素 j
- $\psi_{data}(d)$: 匹配成本（可以是 similarity score 的倒数或相反数）
- $\psi_{smooth}(d, d')$
- Potts: $\psi_{smooth}(d, d') = [d \neq d']$
- Truncated l_1 : $\psi_{smooth}(d, d') = \min(|d - d'|, \tau)$
- $\psi_{smooth}(d, d')$ 表示像素 d 和它的相邻像素 d' 之间的平滑程度，我们有上述两种表示方法

- Potts 的方法显然比较的“简单粗暴”，因此我们多半选择第二种，尤其是它还考虑到了 disparity 差的值
- 其中 τ 指的是 disparity 差值的置信度
- 然后我们利用 belief propagation 来解决这个 MRF

MRF 将在 Lect 5 中详细讲述

$[d \neq d']$ 是一种指示性函数，可以理解为它计算其中等式/不等式的 bool 值，当等式/不等式成立时，值为 1；否则为 0
belief propagation 也将在 Lect 5.7 中详细讲述

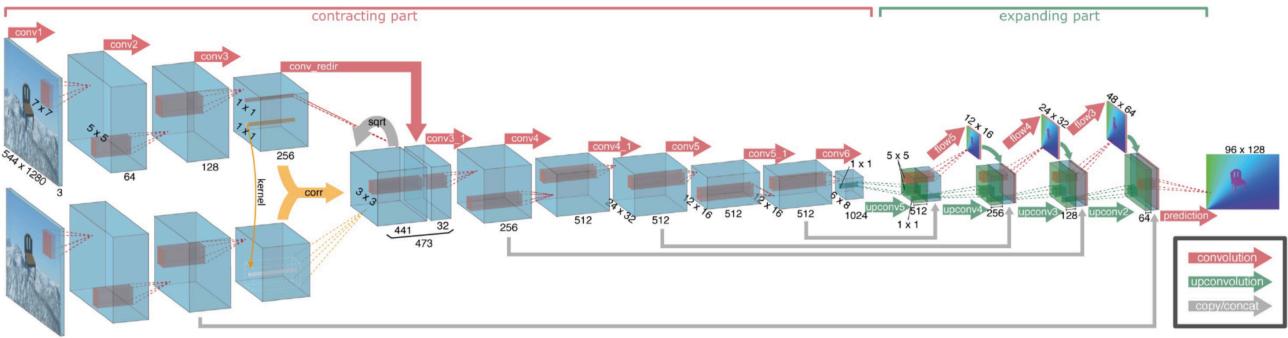
- 最后来看一下，应用这种全局优化后的效果



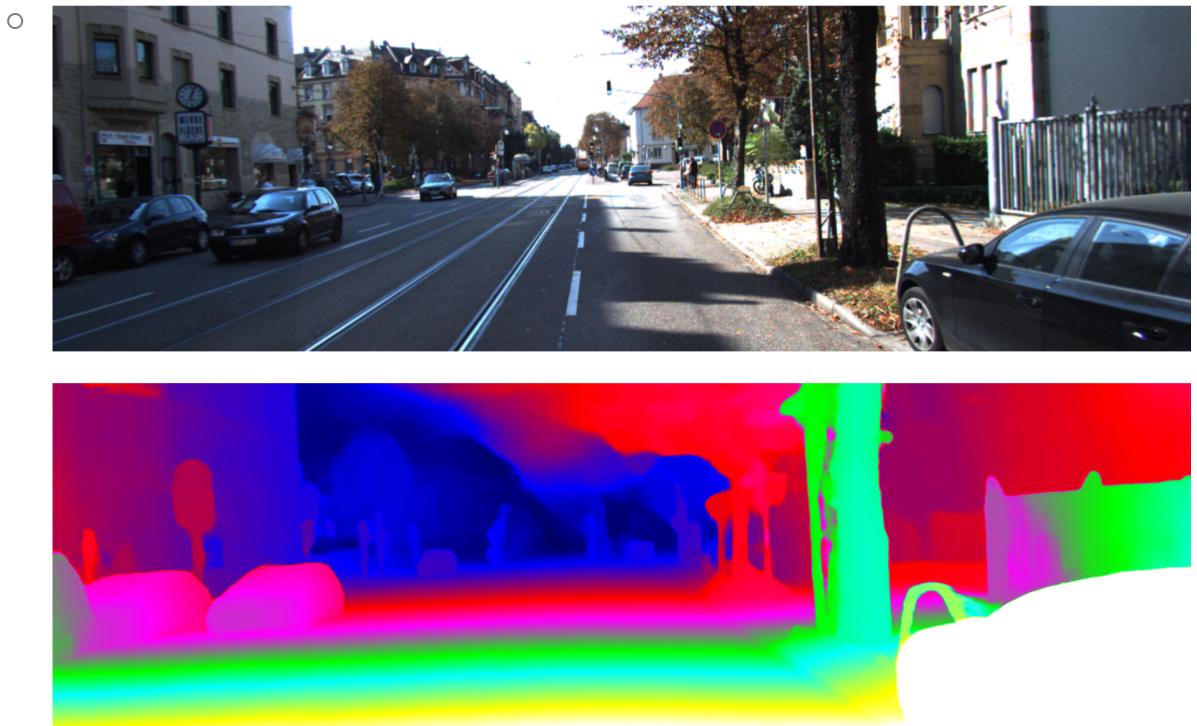
4.5 End-to-End Learning

4.5.1 DispNet

Mayer et al.: A Large Dataset to Train Convolutional Networks for Disparity, Optical Flow, and Scene Flow Estimation. CVPR, 2016.

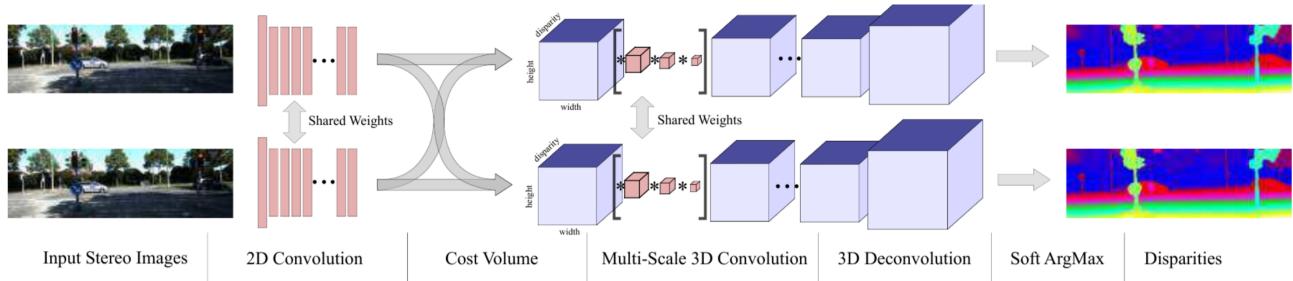


- DispNet 是对于立体还原的首个 End-to-End 的深度神经网络
- 将两个 image 输入后，不断卷积、下采样提取特征，然后采用 correlation layer 将两个特征级联在一起，继续卷积、下采样，之后到设定最小 patch 之后再开始上采样；其中采用许多类似 U-Net 的结构——skip-Connections 来保留细节
 - 它使用了一种类似 U-Net 的结构，通过 skip-Connections 来保留细节
 - 没有使用全局优化算法，但是在 Correlation layer 的部分实际上可以借鉴传统还原的方法，例如采用 Siamese 网络，可以使得模型效果有轻微提升
 - DispNet 使用了多尺度的 loss 函数（像素级别的 disparity 错误），从简单任务到复杂任务的学习过程
- Synthetic Datasets
 - KITTY 使用了立体激光雷达来测量真实的 depth 数据作为 ground truth
 - 但是 KITTY 这样的数据集收集的人力成本巨大
 - KITTY 主要是面向自动驾驶场景的
 - 当时令人非常惊讶的是——这篇文献的工作者们自己利用 3D 引擎渲染了许多数据集用于训练
 - Flying Things
 - Monkaa
 - 由于是 3D 引擎渲染的，depth 信息自然可以作为副产物输出，这样的获取是非常容易的，也即标注的成本是非常低的
 - 他们在大型的 3D 模拟的数据集上做了预训练，并在相对来说体积较小的真实数据集上做了微调；
 - 这种方法在当时是极具有创新性的
- Result on KITTI Dataset



4.5.2 GC-Net

Kendall, Martirosyan, Dasgupta and Henry: End-to-End Learning of Geometry and Context for Deep Stereo Regression. ICCV, 2017



- 紧接着 DispNet 之后的工作是 GC-Net，它用了一个甚至更简单的思想取得了比 DispNet 更好的效果
- 将两个 image 输入后，首先经过共享的 2D 卷积，然后再进行多尺度的 3D 卷积、反卷积，最后经过 softmax 来输出 disparity
 - 主要思想：计算每个像素的 disparity 的匹配成本体积，然后在这个体积上应用 3D 卷积
 - 利用下式将学习到的匹配成本 $c_\theta(d)$ 转换为 disparity
 - $d^* = \text{softmax}(-c_\theta(d)) \cdot d$
 - 比 DispNet 有更好的效果，但是更多的内存需求（因为要应用 3D 卷积）

4.5.3 Stereo Mixture Density Networks

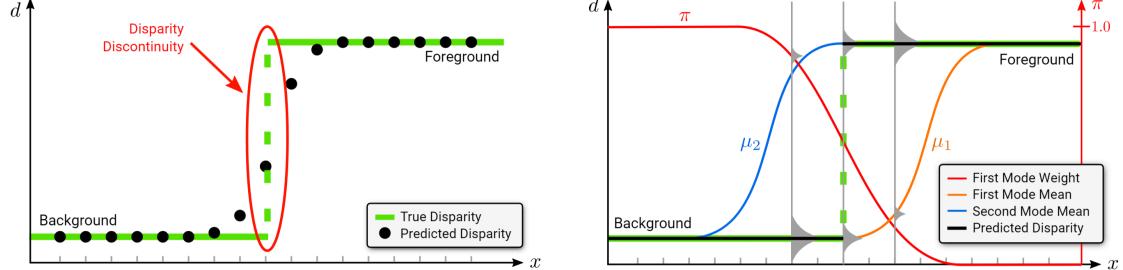
Tosi, Liao, Schmitt and A. Geiger: SMD-Nets: Stereo Mixture Density Networks. CVPR, 2021



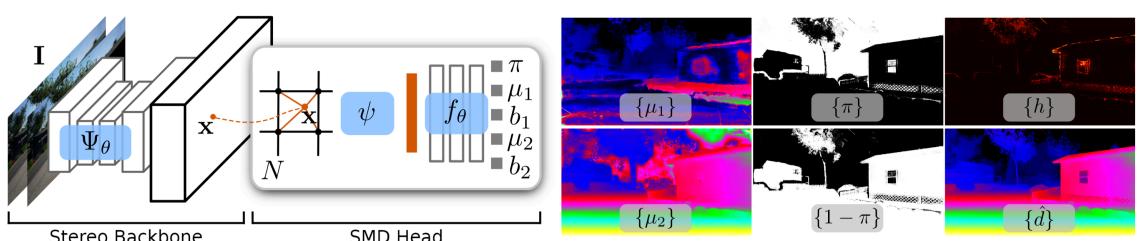
- 由于基于神经网络的回归具有平滑的属性，GC-Net 在 disparity 不连续处出现“飞行像素”一样的出血效果，它的边界不是很清晰，如图



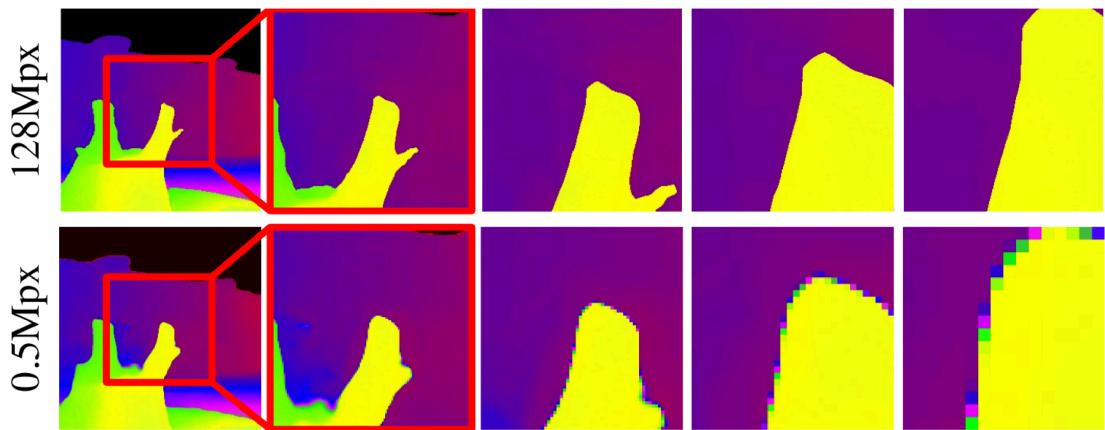
- 而我们接下来介绍的一种，新的思想方法——立体混合密度神经网络，它可以解决这种问题，拥有更清晰的边界、更高的空间分辨率，为什么呢？
- Stereo Mixture Density-Nets



- 左图：由于基于神经网络的回归具有平滑的特性，它在遇到“飞跃式”的 disparity 变化时，总是会有偏差，如图所示
- 右图：SMD-Nets 预测双峰 (Laplacian) 混合分布 (灰色)，可以准确捕获接近深度不连续的不确定性



- SMD-Nets 并不会输出固定分辨率的图像，它在最后加了一个 head 层，称为 SMD Head；它不断在图像中查询连续 disparity 处，利用双线性插值得到缺失的位置的特征，并利用 MLP 去继续预测这部分特征值，从而补充原本固定分辨率的确实位置



- 可以看到，利用这种思想，当我们输入的图像分辨率很高时，输出也可以对应的有很高的分辨率
■ (第一行为 SMD-Nets，第二行为传统的立体回归神经网络)