视频处理与显示技术

Assignment4

三维景深

3D-DOF

姓名:许海鸣

学号:06118113

东南大学电子科学与工程

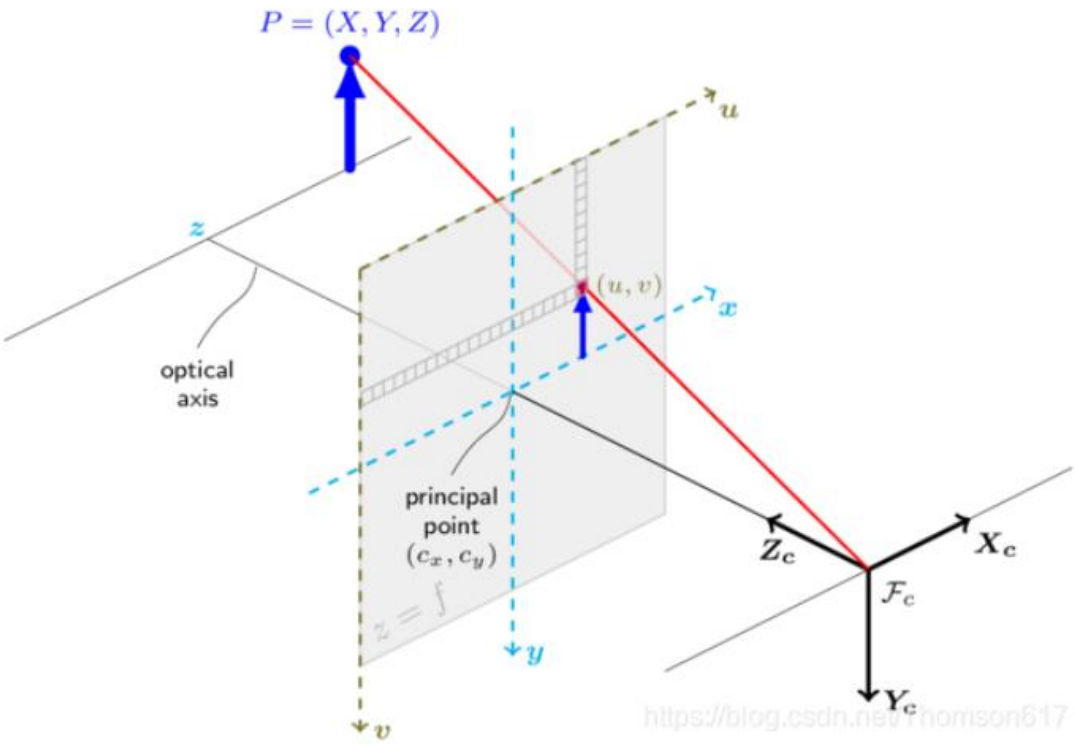
2021-10-26

一 背景简介：

立体视觉匹配 Stereo Matching，在计算机视觉领域是一个重要且悠久的研究方向。从最早的关于视觉的研究中，研究人员就发现我们人类建立立体视觉是基于左右眼观察到的表现差异来感知深度的。这种视觉差异，简称为视差。同时，我们从日常生活就能感知到，物体水平运动的幅度与我们离物体的距离成反比。由此从两眼的输入，再经过我们大脑的处理，人类自然的就获得立体感。

双目摄像头就是基于人眼视觉成像原理而来的，通过计算摄影学的理论，我们就可以通过左右摄像图来获取视差图。由对极几何理论，通过简单的运算我们就能获取景深。

尽管人们已经对场景结构产生视差所相关的基本物理和几何原理都已清楚，但是快速，高精度的通过双目图像建立视差图仍是有挑战性的工作。



二 基本流程：

实际过程中，我们先需要对摄像头进行校正 Rectify，这对能否成功的进行后续计算工作影响十分巨大。校正主要分为畸变校正和立体校正。进行校正之后，我们需要对摄像机参数进行标定 Calibration，获取摄像头的内参与外参，内参反应平面与像素的转换，涉及摄像头焦距，像素密度，光心参数,生成一个内参本征矩阵 E。外参反应摄像机坐标与世界坐标系间旋转和平移关系， Rotation 和 Translation.

像素坐标系 ⟷ 归一化坐标系 ⟷ 摄像机坐标系 ⟷ 世界坐标系

$$\begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} \alpha & 0 & c_x \\ 0 & \beta & c_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} \Leftarrow \begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \frac{1}{z_c} \begin{bmatrix} f & 0 & 0 & 0 \\ 0 & f & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} \Leftarrow \begin{bmatrix} x_c \\ y_c \\ z_c \\ 1 \end{bmatrix} = \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

整理得： 像素坐标系 ⟷ 世界坐标系

$$z_c \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & c_x & 0 \\ 0 & f_y & c_y & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \begin{bmatrix} R & t \\ 0^T & 1 \end{bmatrix} \begin{bmatrix} x_w \\ y_w \\ z_w \\ 1 \end{bmatrix}$$

注：注意非齐次到齐次的变换。

由于提供了处理过的图像，上述步骤可以跳过，接下来就到了如何计算视差图：

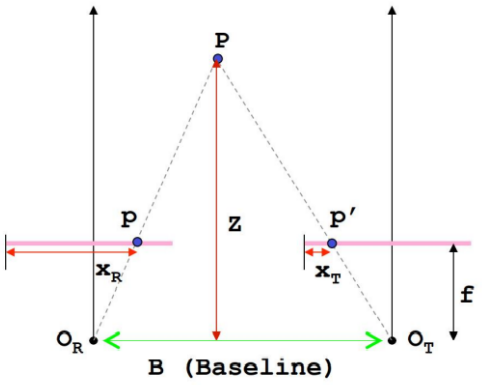
1：匹配代价计算 Cost Computation 这一步是最重要的

2：代价聚合 Cost Aggregation

3：视差计算 Disparity Computation

4：视差优化 Disparity Refinement 可以优化，实际效果还取决匹配的结果

计算完视差图后，我们就通过几何关系，利用公式计算景深。



$$Z = \frac{b \cdot f}{x_R - x_T} = \frac{b \cdot f}{d}$$

虚拟视点重建 View Synthesis

虚拟视点重建比较简单，原理就是根据视差或者景深数据，生成双目间任意视点的图像。做法就是利用左图，对每一个像素，同时加上 0%~100%的视差值，就可以生成虚拟视点图像了。重建的效果，取决于立体匹配的效果。

三 实验思路：

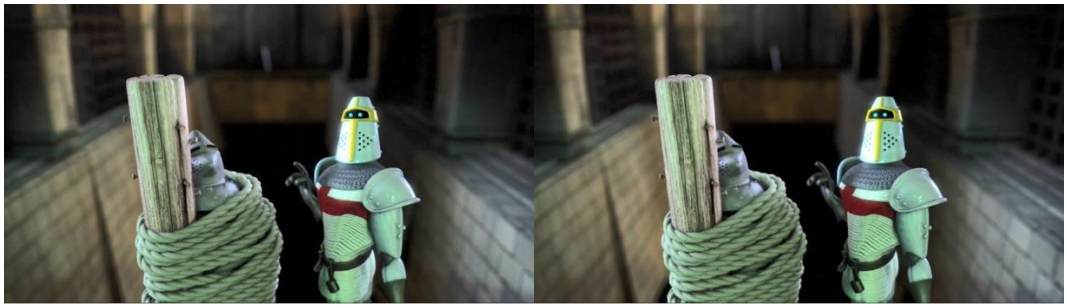
代价匹配--| 匹配算法------OpenCv 自带 SGBM 算法-----→参数调整
--| 优化方法----|--左右一致性检查（交叉验证）LRC-checking
|--边缘优化 边缘检测+滤波
景深计算--|利用公式计算

三维重建--|利用视差图重建

四 实验过程：

1. 素材准备

Rhine_Valley.bmp



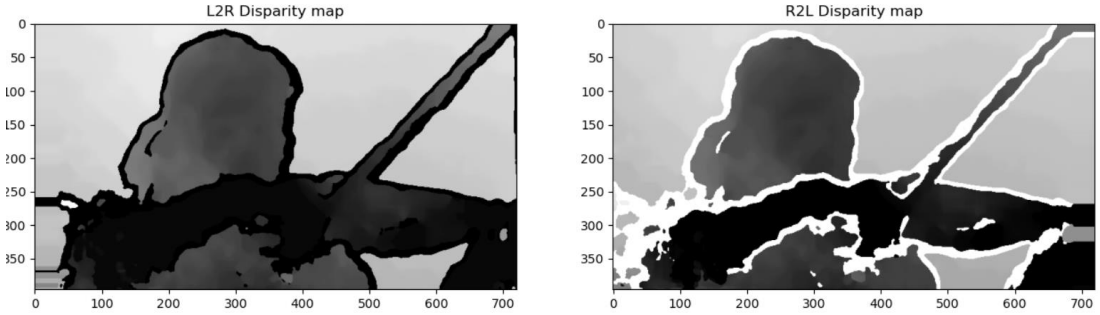
Knight_Quest.bmp
利用画图软件，将左右目图像分开

2. 代价匹配计算 以 Rhine_Valley 为例

我使用了 OpenCV 自带的 SGBM 算法，首先我们需要对 SGBM 参数配置，具体参数意义就不详细说明了，调整方法就是看第一步视差计算的效果以及其泛用性。

```
windowsize=17
channel=1
numdisparities=16*2
stereoL = cv.StereoSGBM_create(0,numdisparities,windowsize,8*channel*windowsize**2,32*channel*windowsize**2,-1,15,0,10,1,1)
disparity_L = stereoL.compute(imgL1,imgR1)
stereoR = cv.ximgproc.createRightMatcher(stereoL)
disparity_R = abs(stereoR.compute(imgR1,imgL1))
```

计算结果如图 可以看到在图像边缘处存在黑边-1 或白边 32



这些地方主要就是由于遮挡 Occlusion 产生的。参数调整过后的实际匹配效果还是不错的。期间还进行了无效深度带填充，空洞填充与中值滤波。

```
for i in range(H):
    for j in range(numdisparities):
        disparity_L[i][0:numdisparities]=disparity_L[i][numdisparities+1]
        disparity_R[i][W-numdisparities:W+1]=disparity_R[i][0]
disparity_L=filterLayer(disparity_L)
disparity_R=filterLayer(disparity_R)
title1='L2R Disparity map'
title2='R2L Disparity map'
show_double(disparity_L,disparity_R,title1,title2)
```

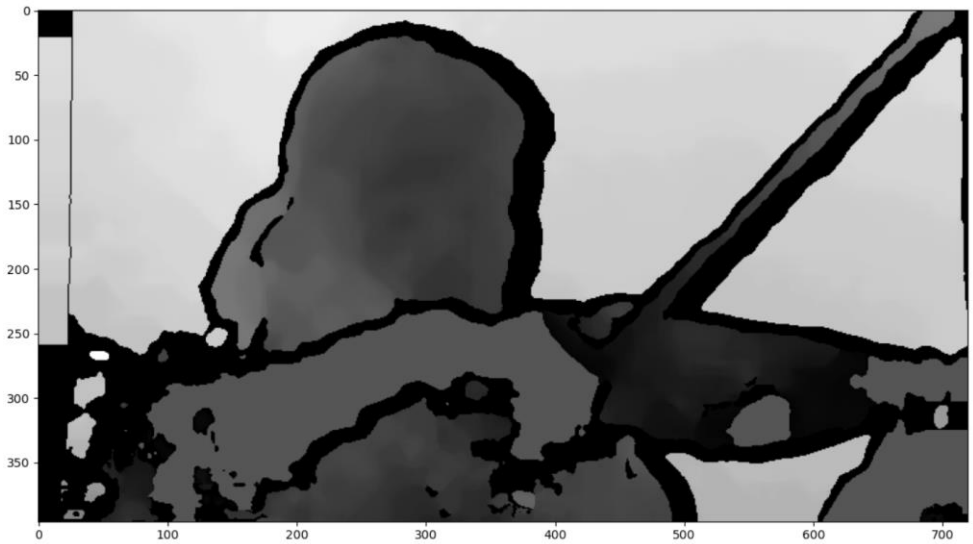
3. 优化处理，

交叉检测

一方面就是 LRC-checking，主要思想就是根据左右两幅输入图像，分别得到左右两幅视差图。对于左图中的一个点 p，求得的视差值是 d1，那么 p 在右图里的对应点应该是(p-d1)，(p-d1)的视差值记作 d2。若|d1-d2|>threshold，p 标记为遮挡点。

具体算法如下 threshold 取 1

```
Occlusion=np.zeros_like(disparity_L)
for i in range(H):
    for j in range(W):
        dispL=disparity_L[i][j]
        if j-dispL>=W:
            Occlusion[i][j]=0
            continue
        dispR=disparity_R[i][j-round(dispL)]
        if dispL==0 and dispR==0:
            Occlusion[i][j]=10
            continue
        if abs(dispL-dispR)<1:Occlusion[i][j]=disparity_L[i][j]
        else:Occlusion[i][j]=0
#show(Occlusion)
```

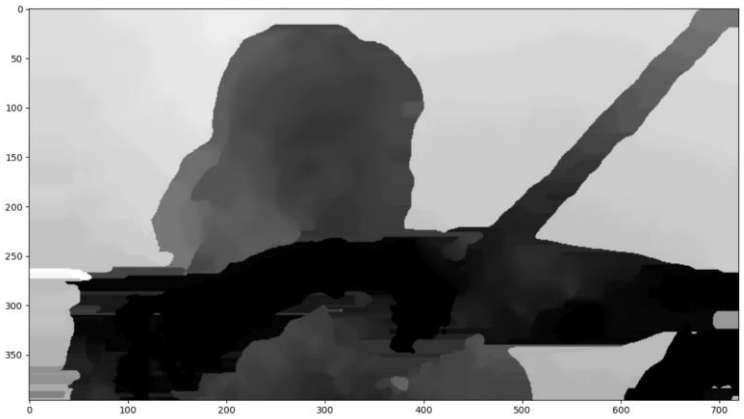


Occlusion 计算图，当然我做了些处理，由于我对图像整体/16，所以在 Occulsion 检测时，我排除了两图对应点都为 0 的像素。

处理 Occlusion 不匹配点，即为这些点赋值，方法是对于一个遮挡点 p，分别水平往左和往右找到第一个非遮挡点，记作 pl、pr。点 p 的视差值赋成 pl 与 pr 的视差值中较小的那一个。 即被遮挡的部分就是背景点。

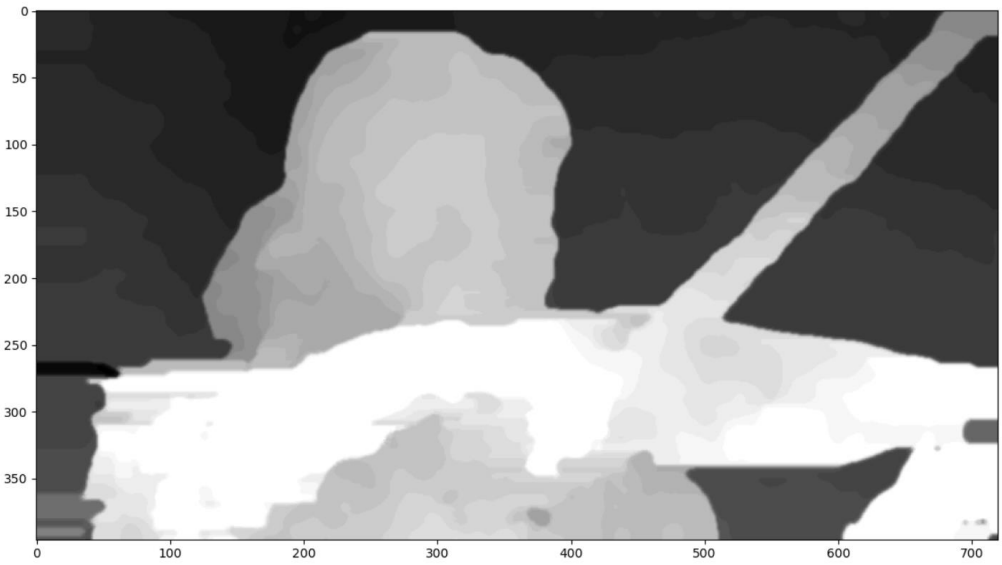
算法如下：

```
for i in range(H):
    for j in range(W):
        if Occlusion[i][j]>0:
            DisparityL_LRC[i][j]=disparity_L[i][j]
            continue
        else:
            pl=0
            pr=0
            left=right=1
            flagL=flagR=1
            while(flagL):
                if j-left<=0:
                                flagL=0
                                pl=1000
                                break
                pl=disparity_L[i][j-left]
                if pl>0:flagL=0
                else:left=left+1
            while(flagR):
                if j+right==W:
                                flagR=0
                                pr=1000
                                break
                pr=disparity_L[i][j+right]
                if pr>0:flagR=0
                else:right=right+1
        if Occlusion[i][j]==0:DisparityL_LRC[i][j]=min(pl,pr)
show(DisparityL_LRC)
```



LRC-checking 后的 Disparity Map

修改填充判据，可以使效果更好，主要修改了边缘填充的值，判断需要填充前景还是背景。

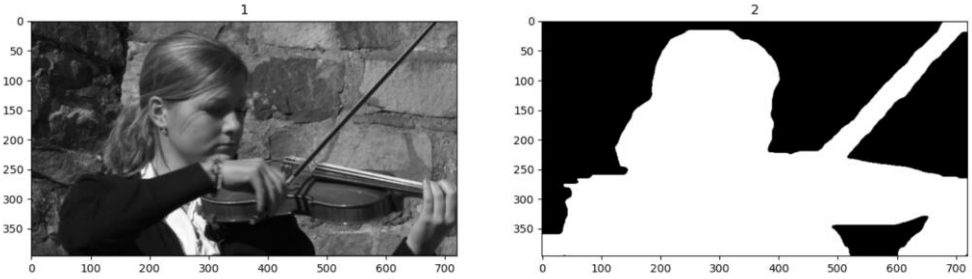


优化过后的边缘填充

可以看到边缘包围是缩小了，但是还是有些多余的地方，需要进一步优化。

进一步的边缘优化

可以看到，我们获取的视差图比实际的图像还多一些，主要是因为边缘遮挡。
优化的目的主要就是让这些地方，运动趋势减小，使得运动过程不是简单平移，而是带有透视变幻的移动，使得立体感增强。



第一步获取掩膜 Mask
代码如下：

```
DisparityL_LRC=cv.GaussianBlur(DisparityL_LRC,(5,5),1)
DisparityL_LRC=cv.medianBlur(DisparityL_LRC.astype(np.uint8),5)
for i in range(H):
    for j in range(1,W):
        if DisparityL_LRC[i][j]>200:
            DisparityL_LRC[i][j]=DisparityL_LRC[i][j-1]
    for k in range(1,W):
        if DisparityL_LRC[i][W-k-1]>200:
            DisparityL_LRC[i][W-k-1]=DisparityL_LRC[i][W-k]
DMask=DisparityL_LRC.copy()
DMask[DMask<=20]=0
DMask[DMask>20]=255
DMask=255-DMask
show_double(imgL,DMask,'1','2')
DMasked=cv.bitwise_and(DMask,imgL)
show(DMasked)
```

在 Mask 范围内对图像做边缘检测，然后排除错误的边缘信息（靠左右统计，梯度与阈值比较缺点）

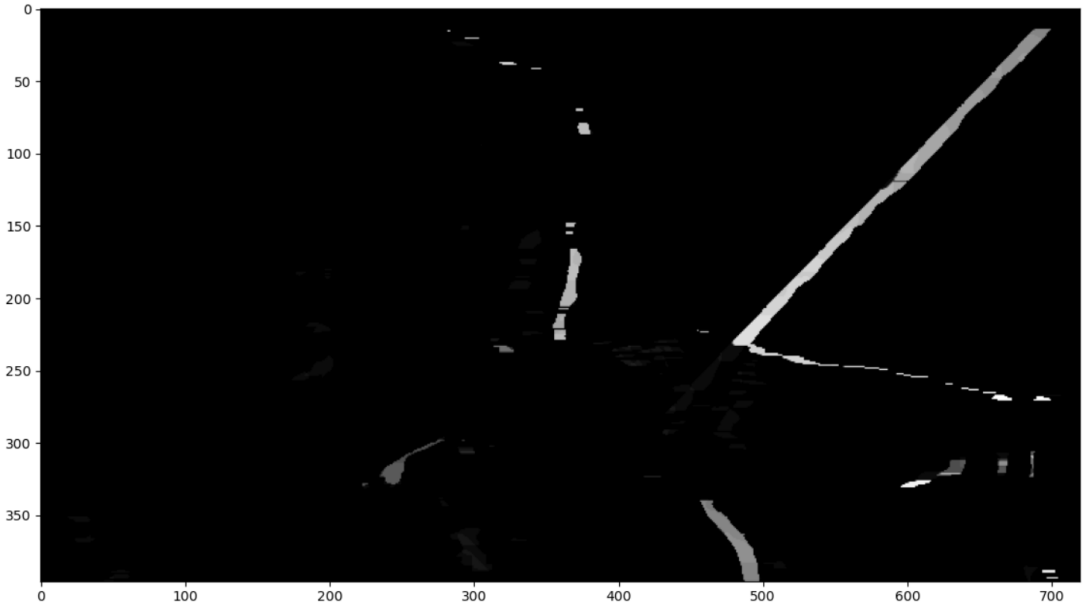
边界信息

处理过后的边界信息（定位出左边发生错误边缘）

然后对这些边缘像素，在左右指定个像素内做了均值滤波。

代码如下：

```
sobel_x = cv.Sobel(DMasked, -1, 1, 0, ksize=3)
sobel_y = cv.Sobel(DMasked, -1, 0, 1, ksize=3)
edge=cv.addWeighted(sobel_x,0.5,sobel_y,0.5,0)
Disp_B=DisparityL_LRC.copy()
edge[edge<70]=0
edge[edge>=70]=255
for i in range(1,H-1):
    for j in range(1,W-1):
        if edge[i][j]==0:continue
        else:
            if abs(sum(DMask[i][j-5:j-1])-sum(DMask[i][j+1:j+5]))>100:edge[i][j]=0
show_double(DisparityL_LRC,edge,"disp","edge")
ws=10
for i in range(0,H):
    for j in range (ws,W-2*ws):
        if edge[i][j]==0:continue
        else:DisparityL_LRC[i][j:j+ws]=np.median(DisparityL_LRC[i][j+ws:j+2*ws])
DisparityL_LRC=max(DisparityL_LRC.flatten())-DisparityL_LRC
```



边缘优化差异

此时，显示出的前景

4. 虚拟视点重建

根据视差或者景深数据，生成双目间任意视点的图像。做法就是利用左图，对每一个像素，同时加上 0%~100%的视差值，就可以生成虚拟视点图像了，并生成 GIF(见附件)

实际上，最好是用景深，视差数据反推回去，重建视点图。

代码如下:

```
##########################3D-construction#########################
visual_Synthesis=np.zeros_like(imgC)
number=50
for i in range(number):
    print(i)
    for j in range(H):
        for k in range(W):
            disp_VS=DisparityL_LRC[j][k]
            if k+round(i*1/number*disp_VS)>=W:visual_Synthesis[j][k]=imgC[j][k]
            else:visual_Synthesis[j][k]=imgC[j][k-round(i*1/number*disp_VS)]
    cv.cvtColor(visual_Synthesis,cv.COLOR_BGR2RGB)
    cv.imwrite("C:/Users/PC/Desktop/assignment4/Rhine_valley/output/"+str(i)+".bmp",visual_Synthesis)
##############################gif#################################################

with imageio.get_writer(uri='test.gif',mode='I',fps=24) as writter:
    for i in range(number):
        writter.append_data(imageio.imread(f'C:/Users/PC/Desktop/assignment4/Rhine_valley/output/{i}.bmp'))
    for i in range(number):
        writter.append_data(imageio.imread(f'C:/Users/PC/Desktop/assignment4/Rhine_valley/output/{50-1-i}.bmp'))
with imageio.get_writer(uri='test1.gif',mode='I',fps=24) as writter:
    for i in range(25):
        writter.append_data(imageio.imread(f'C:/Users/PC/Desktop/assignment4/Rhine_valley/output/{i}.bmp'))
    for i in range(25):
        writter.append_data(imageio.imread(f'C:/Users/PC/Desktop/assignment4/Rhine_valley/output/{25-1-i}.bmp'))
```

重建出的中间某一帧图像

重建出的右目图像

五 思考与总结

传统的立体匹配算法，搭配左右一致性检测，效果已经在可以接受的范围内了，若要做到十分精准的前后景区分，只有不断的增加条件去约束，不断的调整参数。所以，算法泛化性不会高。

虽然我没有使用神经网络来做这次的作业，但是在本次作业中，不段的调试传统算法的参数，增加约束的过程相当的耗费时间，使得我深深的感受到，神经网络的优越性。神经网络依赖其大量的输入数据，训练出来的网络具有一定的泛化能力的，且神经网络只需要输入与输出，只要通过训练网络内核的权重就可以将网络部署到实际生活中。这为工程人员提供解决问题的新思路，尽管训练的过程，需要依赖于大量的计算资源，但事实证明这样的开销是值得的。