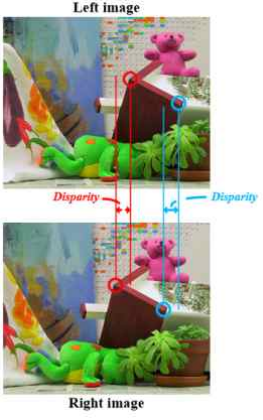
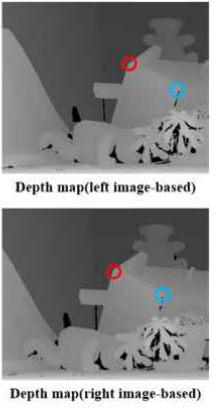
Stereo Matching

스테레오 정합
2017104034 최시원

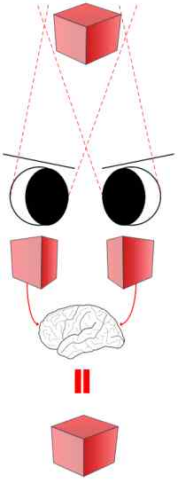
Stereo matching?

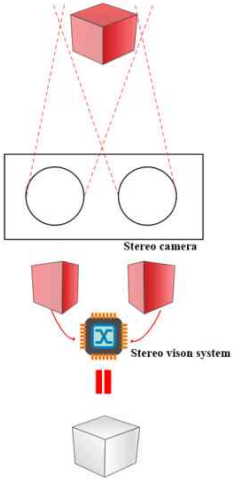




- 스테레오 정합 : reference image에서, 한 점에 대해 동일한 점을 target image에서 찾는 과정을 의미함.
- 두 이미지는 left, right image로 생각될 수 있음. 한쪽을 reference로 하면 반대쪽이 target이 됨.
- 두 이미지 사이에는 시차(disparity)가 존재함. 스테레오 정합은 이 '시차'를 계산하는 과정이다.
- 결과 영상은 흑백으로 표현된다. 어두움과 밝음이 존재하는데, 어두움은 시차 값이 작다는 의미이고 밝음은 시차 값이 크다는 의미이다. 따라서 각각 멀고 가까움을 나타낸다.

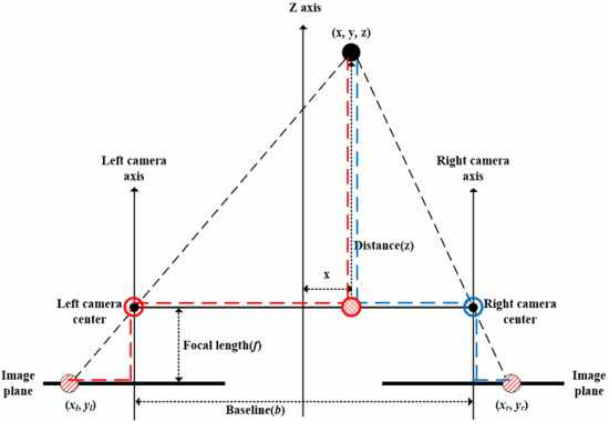
Stereo vision





- 사람의 시각 시스템을 모방한 기술. 시차(disparity)라는 개념을 설명하기 위함.
- 인간의 눈은 미간의 간격만큼 서로 떨어져 있다. 이 격차로 인해 물체를 보았을 때 각 눈에서 보이는 시점의 차이, 즉 시차disparity가 생긴다.
- 이 시차disparity는 가까이 있는 것에선 크게, 멀리 있는 것에선 작게 나타난다. 이것이 물체를 3차원으로 인식하게 만든다.
- 마찬가지 매커니즘으로 stereo vision은 stereo camera를 통해 입력된 2차원의 좌/우 영상의 x축 위치 차이를 통해 시차disparity를 계산함으로써 3차원 거리 정보를 획득한다.

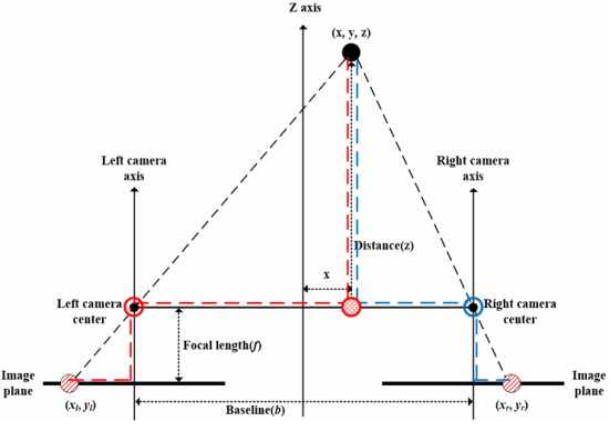
Stereo vision



- 3차원 거리 정보는 시차disparity, 초점거리focal length, 베이스라인baseline 세가지 요소를 통해 획득 가능하다.

- 시차disparity : 좌/우 영상에서 동일하게 나타나는 픽셀에 대한 x축 위치 차이
- 초점 거리focal length : 이미지 평면과 카메라 렌즈의 거리
- 베이스라인baseline : 좌/우 카메라의 간격(인간의 미간 거리)

Stereo vision



- 3차원의 (x, y, z)좌표를 가진 점은 좌, 우 카메라렌즈의 중심을 거쳐 이미지 평면에 상이 맺힌다.
- 이때, 좌측과 우측 이미지 평면에 맺힌 상의 차이를 자세히 보면(빨간색 빗금) 좌측은 left camera axis와 비교하여 왼쪽, 우측은 right camera axis와 비교하여 비교적 덜 오른쪽에 맺힘을 볼 수 있다.
- 이 차이가 곧 x값(disparity)으로 나타남을 알 수 있다. 이걸로 시차를 구한다.
- 3차원 거리정보 요소 셋중 나머지 둘은 고정된 상수이므로, 이 시차를 정확히 구하는것이 3차원 거리 정확성에 영향을 미칠 것이다.

Stereo matching

```
#이미지의 크기를 알아냄
rows = leftImg.shape[0]
cols = leftImg.shape[1]

# 좌우 DSI 크기만큼 0을 채움
# 이게 결과물이 됨
leftDSI=np.zeros((rows,cols))
rightDSI=np.zeros((rows,cols))

#대각선이 아닌 방향으로 한 칸 움직일 때의 코스트
costPerBlock = 21

#커널은 여기서 지정
#3*3 크기
#지금은 Gaussian filter로 설정
kernel = [0.0625, 0.125, 0.0625,
          0.125, 0.25, 0.125,
          0.0625, 0.125, 0.0625]
```

- 이미지의 크기를 구하고, 이 크기를 가진 0으로 채운 2차원 배열을 생성한다. 결과물을 저장할 배열이다.
- costPerBlock은 DP 과정에서 행이나 열 방향으로 이동할 때 부여하는 정합 비용matching cost 값이다. 이는 곧 픽셀간 비유사도를 나타낸다. 이것이 작은 값일 수록 픽셀간 유사도가 크다는 것을 나타낸다.
- bluring 처리를 할 때 쓸 Gaussian filter이다.

Stereo matching

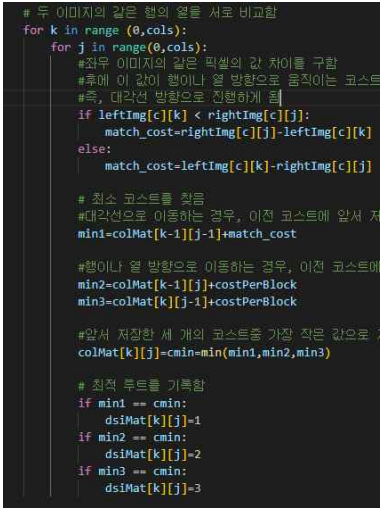
```
# 행 단위로 반복
for c in range (0,rows):
    print("current row :",c)
    # Dytnamic programming에 의해 최적 루트를 저장하기 위한 배열
    dsiMat=np.zeros((cols,cols))
    # 코스트를 저장할 배열, 좌우 이미지의 c행의 각 열을 비교
    colMat=np.zeros((cols,cols))

    # 첫째 행, 첫째 열은 무조건 해당 방향으로 한칸씩 이동해야 최적이므로
    # 원점에서 멀어진 거리 * 블록당 코스트를 저장해 둠
    for i in range(0,cols):
        colMat[i][0] = i*costPerBlock
        colMat[0][i] = i*costPerBlock
```

```
[[    0.    21.    42. ... 14028. 14049. 14070.]
 [   21.     0.     0. ...     0.     0.     0.]
 [   42.     0.     0. ...     0.     0.     0.]
 ...
 [14028.     0.     0. ...     0.     0.     0.]
 [14049.     0.     0. ...     0.     0.     0.]
 [14070.     0.     0. ...     0.     0.     0.]]
```

- 이미지의 모든 행에 대해 반복한다.
- dsiMat는 DP에 의해 나온 최적 루트를 저장한다
- colMat는 코스트를 저장할 배열이다. 특정 c행 기준으로 좌/우 이미지의 각 열을 비교하여 코스트를 저장한다
- 첫 행, 열에서는 무조건 해당 방향으로 한칸씩 이동해야 하므로 블록당 코스트로 초기화한다.
- 그 결과가 좌측이다. 이 2차원 배열은 특정 c행에서 좌측 이미지의 k열과 우측 이미지의 j열 간의 코스트를 의미한다.

Stereo matching



```
# 두 이미지의 같은 행의 열을 서로 비교함
for k in range (0,cols):
    for j in range(0,cols):
        #좌우 이미지의 같은 픽셀의 값 차이를 구함
        #후에 이 값이 행이나 열 방향으로 움직이는 코스트
        #즉, 대각선 방향으로 진행하게 됨
        if leftImg[c][k] < rightImg[c][j]:
            match_cost=rightImg[c][j]-leftImg[c][k]
        else:
            match_cost=leftImg[c][k]-rightImg[c][j]

        # 최소 코스트를 찾음
        #대각선으로 이동하는 경우, 이전 코스트에 앞서 저
        min1=colMat[k-1][j-1]+match_cost

        #행이나 열 방향으로 이동하는 경우, 이전 코스트에
        min2=colMat[k-1][j]+costPerBlock
        min3=colMat[k][j-1]+costPerBlock

        #앞서 저장한 세 개의 코스트중 가장 작은 값으로
        colMat[k][j]=cmin=min(min1,min2,min3)

        # 최적 루트를 기록함
        if min1 == cmin:
            dsiMat[k][j]=1
        if min2 == cmin:
            dsiMat[k][j]=2
        if min3 == cmin:
            dsiMat[k][j]=3
```

- 두 이미지의 특정 c행에서의 열을 비교한다
- 좌측 이미지는 k열, 우측 이미지는 j열이다. 기준 이미지가 좌측임을 의미한다.
- 좌우 이미지의 픽셀값의 차이를 match_cost에 저장한다.
- 이후 이를 통해 colMat에서 대각선으로 이동하는 경우, 행이나 열 방향으로 이동하는 경우를 저장한다.
- 대각선으로 이동하는 경우는 두 이미지의 비교 열을 가리키는 k, j 인덱스가 같이 변함을 의미한다. 행이나 열은 둘중 하나만 바뀜을 의미한다.
- 셋중 가장 작은 코스트를 값으로 정하고, 이를 dsiMat에 최적 루트로 기록한다.

Stereo matching

```
# 끝까지 도달했으면 이제 되짚어가면서 DSI에 값을 저장함
i=cols-1
j=cols-1

while i > 0 and  j > 0:
    if dsiMat[i][j] == 1:
        leftDSI[c][i]=np.absolute(i-j)
        rightDSI[c][j]=np.absolute(j-i)
        i=i-1
        j=j-1
    elif dsiMat[i][j] == 2:
        leftDSI[c][i]=0
        i=i-1
    elif dsiMat[i][j] == 3:
        rightDSI[c][j]=0
        j=j-1
```

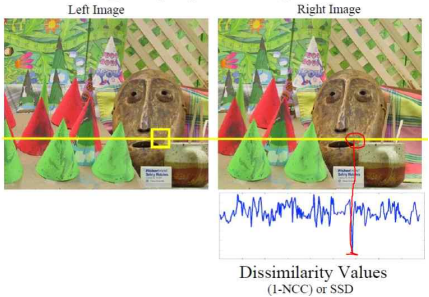
- 이후 dsiMat에 저장된 1, 2, 3의 값을 토대로 거꾸로 되짚어가며 DSI에 값을 저장한다.
- 1인 경우 대각선으로 움직인 경우이고, 이는 임의의 c행에서 k열과 j열의 픽셀차이가 k열만 옮기거나 j열만 옮겼을때에 비해 상대적으로 작다는 의미이다
- 거꾸로 2의 경우 k열이 움직였을 때, 3의 경우 j열이 움직였을 때의 값이고, 각각 좌측/우측 이미지를 의미하므로 leftDSI, rightDSI에 저장한다.

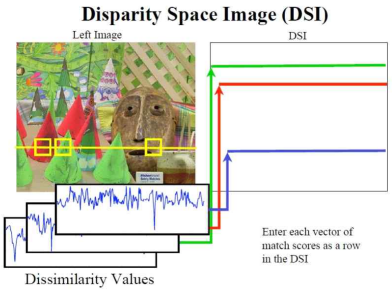
Stereo matching

```
# 완성된 DSI의 중간중간 구멍이 있는 부분은 필터를 통해 bluring 시킴
blurDisp=np.zeros((rows,cols)) # bluring 시킨 DSI
for i in range(0,rows):
    for j in range(0,cols):
        filterValues = [0]*9
        index = -1
        for _i in range(i-1, i+2):
            for _j in range(j-1, j+2):
                index +=1
                if _i < 0 or _i > rows-1:
                    continue
                if _j < 0 or _j > cols-1:
                    continue
                #커널에 입력할 값 수집
                filterValues[index] = rightDSI[_i][_j]
        blurDisp[i][j] = filtering(filterValues, kernel)
```

- 이후 완성된 DSI의 중간중간 구멍이 있는 부분을 gaussian filter를 이용해 bluring시킨다.
- 이를 blurDisp에 저장한다.
- 이후 이 둘을 출력한다. 좌측 영상을 기준으로 잡았기 때문에 DP만 적용한 것을 rightDSI로 삼아 출력한다.

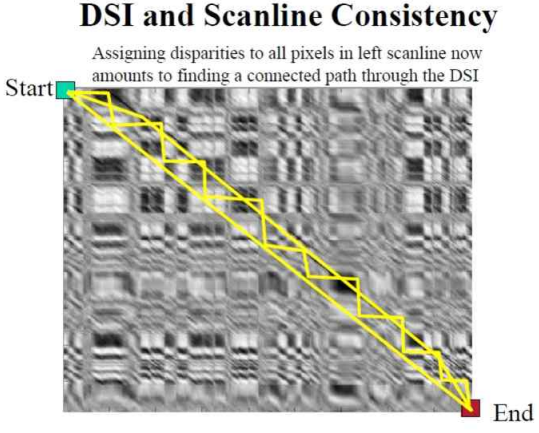
Disparity Space Image (DSI)





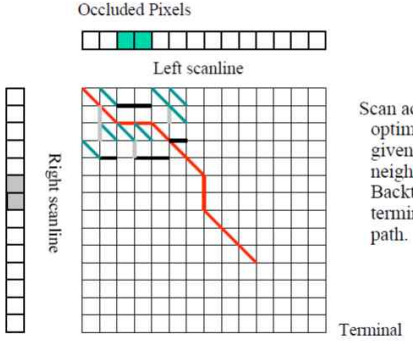
- 각 행마다 DSI가 구해지며, 이미지의 모든 행에서 구해진다.
- 특정 행을 c라고 하면, 좌측 이미지와 우측 이미지의 c행의 k, j 열에서의 픽셀값 차이를 구하여 만들어진다.
- 픽셀값의 차이로 구한 비유사도 값의 그래프이다. 우리가 보기에 동일한 지점에서 비유사도 값이 최하점인 것을 확인할 수 있다.
- 이러한 DSI를 모든 행에 대하여 구한다.

Disparity Space Image (DSI)



- 특정 행에서 구해진 DSI의 한 예시이다.
- 좌측 이미지의 k열을 세로축, 우측 이미지의 j열을 가로축이라고 따졌을 때, k=0~col, j=0~col까지의 모든 범위의 픽셀을 각각 비교한다.
- 비교과정에서, 픽셀값의 차이를 match_cost 변수에 저장한다. 이것과 행, 열 방향으로 이동하는 것과의 최소값을 구한다.
- 행, 열 방향으로 이동하는 경우는 특정 k와 j의 비유사도 값이 큰 편임을 의미한다.
- 이 과정을 통해 dsiMat에 1, 2, 3이 각각 방향에 따른 값으로 저장된다.

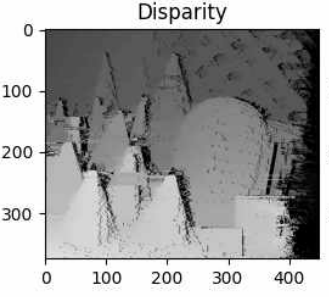
Disparity Space Image (DSI)

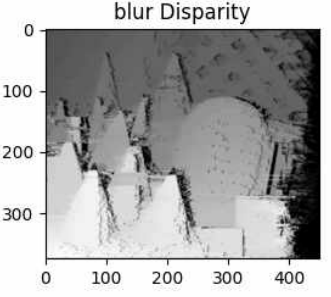


- 그렇게 비유사도 값이 최소가 되는, 최단 경로를 찾는다. 이것이 해당 c행에서 픽셀들의 3차원 깊이 값이 된다. 시차 값을 나타낼 수 있는 것이다.

- DSI를 leftDSI, rightDSI에 저장한다. 모든 c행에 대해서 이 과정이 끝나면 하나의 깊이 영상 depth map이 만들어지게 된다.

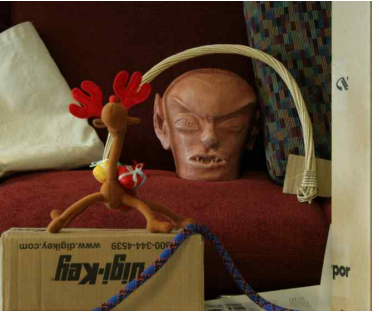
결과물

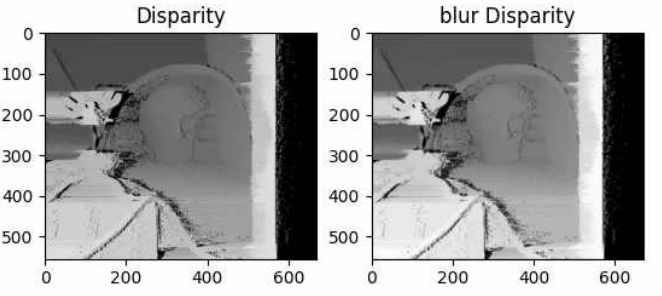




- middleburry 데이터 셋 2003년 cones
- https://vision.middlebury.edu/stereo/data/scenes2003/newdata/cones/

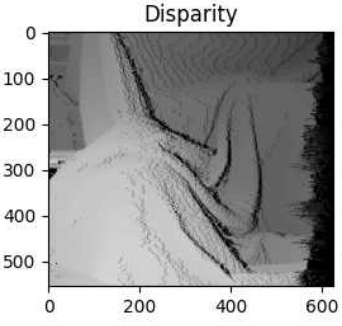
결과물

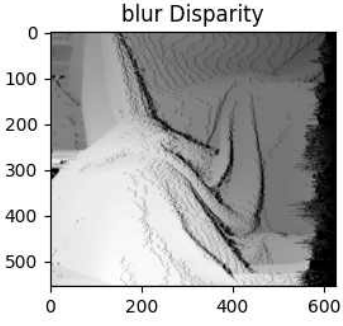




- middleburry 데이터 셋 2005년 reindeer
- https://vision.middlebury.edu/stereo/data/scenes2005/FullSize/Reindeer/

결과물





- middleburry 데이터 셋 2006년 Cloth3
- https://vision.middlebury.edu/stereo/data/scenes2006/FullSize/Cloth3/