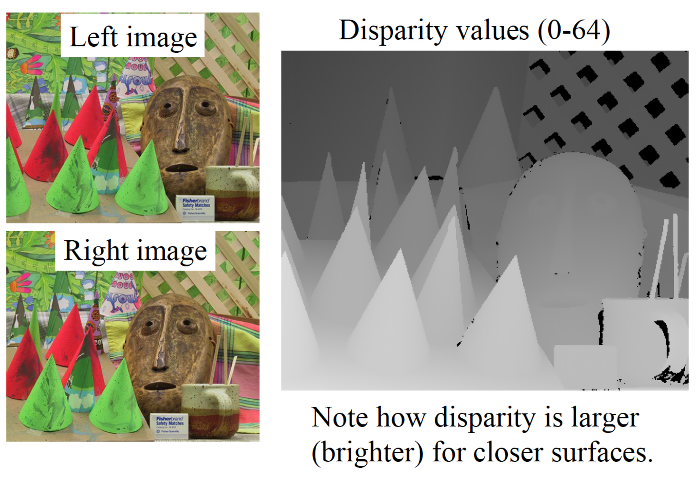
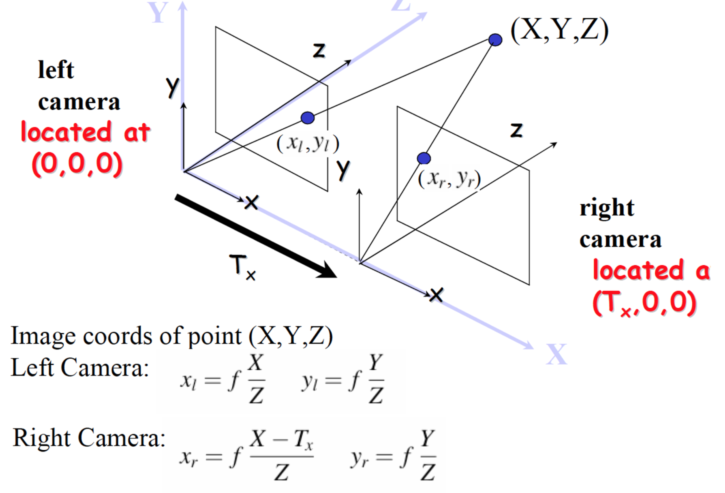
Self-Driving Cars to See Through Fog with GPU-Imaging and Deep Learning

Depth from Stereo Disparity

- We approach the dehazing problem by extracting a depth map, which is used to relate the hazy scene to the clear scene.
- **Stereo vision**: We compute depth using two cameras and observing the "disparity" between corresponding pixels.



a) Disparity from Structured Light



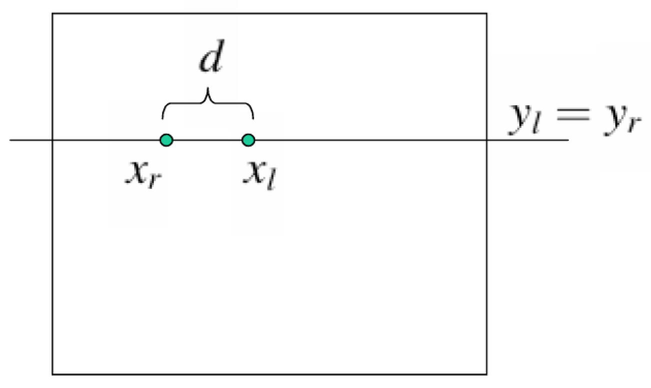
b) Stereo Camera System

Left camera

$$x_l = f\frac{X}{Z} \qquad y_l = f\frac{Y}{Z}$$

Right camera

$$x_r = f\frac{X - T_x}{Z} \qquad y_r = f\frac{Y}{Z}$$



Stereo Disparity

$$d = x_l - x_r = f\frac{X}{Z} - \left(f\frac{X}{Z} - f\frac{T_x}{Z}\right)$$

$$d = \frac{f\,T_x}{Z}$$

depth baseline

$$Z = \frac{f\,T_x}{d}$$

disparity

Important equation!

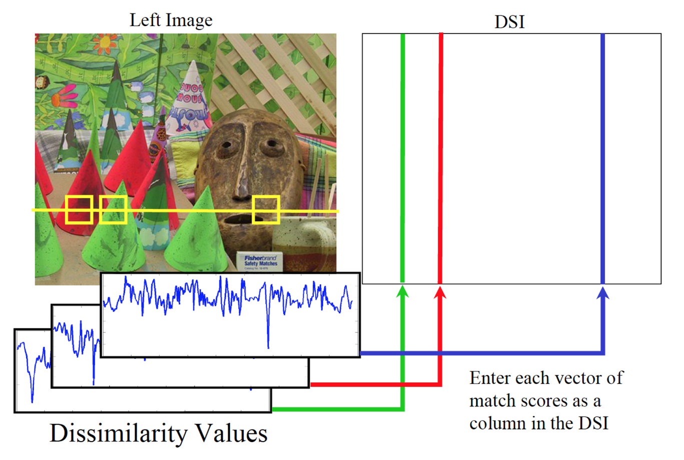
Efficient Disparity Map Extraction

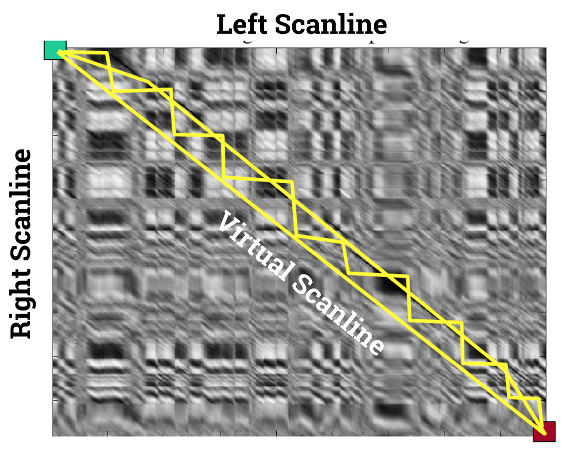
- **Problem:** Given two stereo images, compute the disparity map.
- Define a naive match score (cost) of two pixels using the Normalized Cross-Correlation (NCC). We define a minimum threshold for this match score, α.

$$M(l,r) = 0.5(1 - NCC)$$

$$M(l,r) = 0.5\left(1 - \frac{\sum_{\sigma\in\Omega}(I^l_{p_l+\sigma} - \overline{I^l_{p_l}})\sum_{\sigma\in\Omega}(I^r_{p_r+\sigma} - \overline{I^r_{p_r}})}{\sqrt{\sum_{\sigma\in\Omega}(I^l_{p_l+\sigma} - \overline{I^l_{p_l}})^2\sum_{\sigma\in\Omega}(I^r_{p_r+\sigma} - \overline{I^r_{p_r}})^2}}\right)$$

- We set the two cameras to be level, allowing us to assume an equipolar constraint: each pixel only matches to a pixel on the same row in the corresponding image
- For every row, we can construct a disparity space image (DSI), where every entry DSI[i, j] corresponds to the matching value of IMG[r, i] and IMG[r, j]

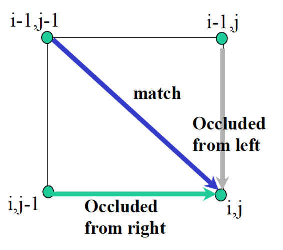




- The disparity values for each row is found by constructing the min cost path through the DSI: backtrack using pointers

$$C(l,r) = \min\begin{cases} C(l-1,r) & +\ \beta \\ C(l-1,r-1) & +\ M(l,r) \\ C(l,r-1) & +\ \beta \end{cases}$$

- Path cannot double back on itself
- Cox et al. (1996) proposed a dynamic programming solution
- Path is always on the far side of the right scanline as the left image always shifted to the left
- Propagate DP F entries past virtual scanline, where F is the max disparity. Reduces DSI memory to F*COLS.



Parameter	Value
Patch Size	5x5
Gaussian Size	7x3
$\sigma_a, \sigma_{a'}$	3.0, 2.0
α	0.7
β, β'	1.0, 1.0
γ	0.25
SW	20
R	3
N	10
Refining Time (100 trials)	0.0419s
Refining FPS	23.8819

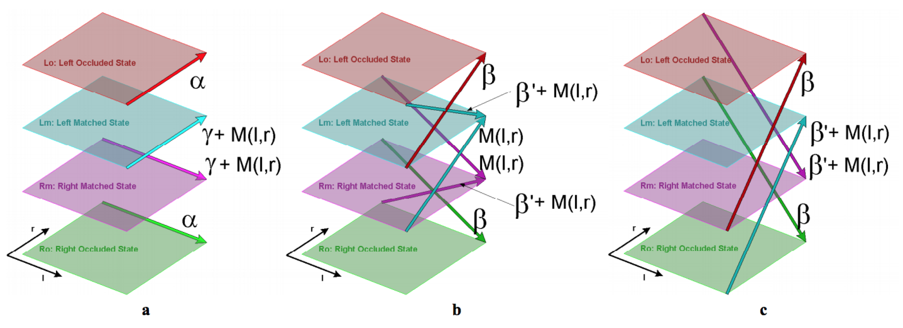
***Not to be confused with scattering parameters**

- DP is parallelizable for real-time applications as each row of the disparity map is calculated independently
- Initial three state DP approach did not produce accurate disparity maps. We utilize a four state DP algorithm proposed by researchers at Microsoft thats builds to the three state model.
- Distinguishes between a left match and a right match. A true match is a consecutive left and right match. Results in less false occlusions.

$$C_{Lo}[l,r] = \min\begin{cases} C_{Lo}[l,r-1] & +\ \alpha \\ C_{Lm}[l,r-1] & +\ \beta \\ C_{Rm}[l,r-1] & +\ \beta \end{cases}$$

$$C_{Lm}[l,r] = M(l,r) + \min\begin{cases} C_{Lo}[l,r-1] & +\ \beta' \\ C_{Lm}[l,r-1] & +\ \gamma \\ C_{Rm}[l,r-1] & \\ C_{Ro}[l,r-1] & +\ \beta' \end{cases}$$

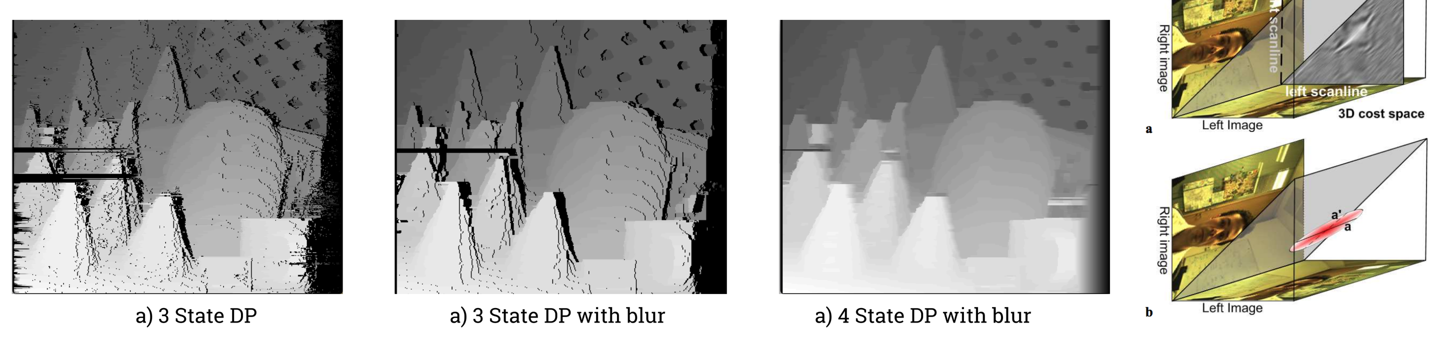
Transitions for Ro and Rm can be determined through symmetry



a) State preserving transitions

b, c) Between state transitions

- We have resolved false occlusions, but because each row is calculated independtly, any DP algorithm produces inconsistent 'streaks'. To remedy this while preserving parallelism, we propogate information across rows by stacking the DSIs and running a Gaussian Blur across the virtual scanline.



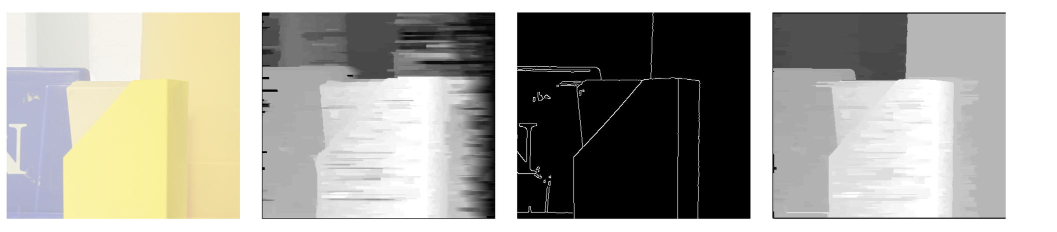
a) 3 State DP a) 3 State DP with blur a) 4 State DP with blur

- Four state DP fails when stereo images are hazy. **We propose a fast O(N), refinement algorithm that greatly improves disparity map quality.**

Algorithm 1 Proposed Refinement Algorithm

```
function REFINE(dismap, img)
    edge ← cannyEdgeDetection(img)
    edge ← dialate(edge)                          ▷ Thicken edges to close gaps
    labels, components ← connectedComponents(edge)   ▷ Iterative Depth First Search
    labels ← labelEdges(labels, edge)             ▷ Ensure all pixels belong to a component
    assignPixelsValues(labels, components)
    for comp in component do
        SW = 20
        histogram ← getHistogram(comp)            ▷ Sliding Window
        ps ← prefixSum(histogram)
        if ps[255]-ps[0] < A then                 ▷ Excludes occlusions, which have value of 0
            continue        ▷ Prevents components with small area from participating, reducing noise
        end if
        window = maxSWSum(histogram, ps, SW)
        mx = findMax(window)         ▷ Max SW Sum first prevents lone peaks from masking true max
        lb = findLeftValley(mx, R, N)
        rb = findRightValley(mx, R, N)     ▷ bin count < mx/R but greater than noise filter, N
        for pixel in comp do                      ▷ Remove incorrect matching
            if dismap.pixel.value not in [lb, rb] then
                dismap.pixel.value = nearest(lb, rb)
            end if
        end for
    end for
end function
```

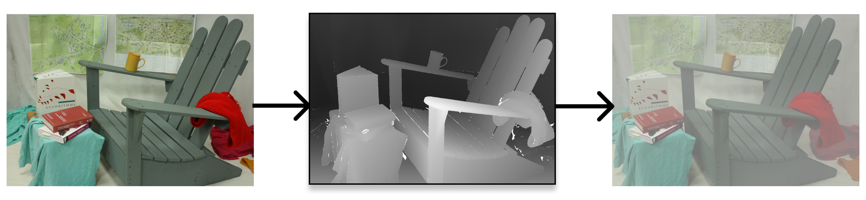
- **Refinement**: We account for 1) discrete histogram peaks (non-max sliding-window is noise) and 2) continuous change in disparity.

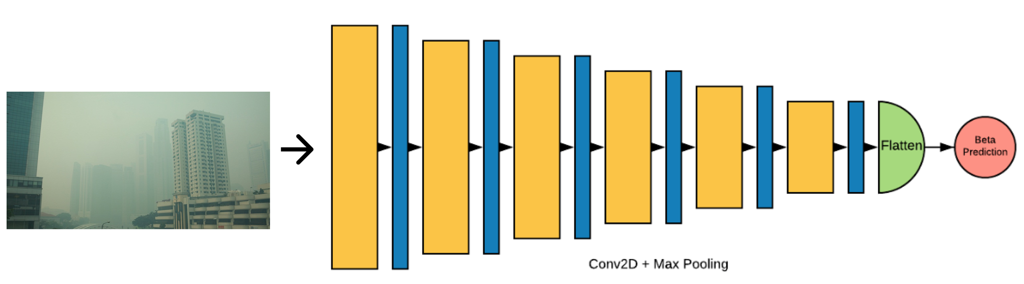

a) Hazy Image b) 4 State DP c) Edge Map d) Refined Disparity (Proposed Algorithm)

Determining Scattering Parameter β

Data:

- No natural haze datasets. Use Atmospheric Scattering Model to generate data.
- Middlebury Datasets 2005, 2014: Contains indoor stereo imagery for depth estimation.
- Given depth, we induce haze in the scenes for $\alpha \in (0.75, 1.0)$, $\beta \in (0, 1.5)$, which was recommened by Cai et al. (Dehazenet)
- 21 scenes, 80 hazy images/scene, 10 128x128 crops/image, totaling 16800 crops





- Regression using Convolutional Neural Network
- Model Input: 128x128 RGB crop of hazy scene
- Model Output: Scattering coefficient, β
- The β predictions will be used to quantify how much airlight to subtract from the hazy scene.

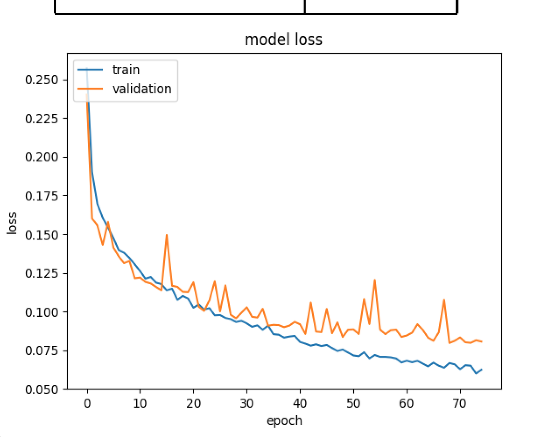
Hyperparameters

activation	ReLU
optimizer	Adam
number of kernels	32
kernel size	3x3
pooling operator	2x2 max
epochs	75
batch size	32
loss	mean absolute error

Layer (type)	Output Shape	Param #
input_1 (InputLayer)	(None, 64, 64, 3)	0
conv2d_1 (Conv2D)	(None, 64, 64, 32)	896
max_pooling2d_1 (MaxPooling2	(None, 32, 32, 32)	0
conv2d_2 (Conv2D)	(None, 32, 32, 32)	9248
max_pooling2d_2 (MaxPooling2	(None, 16, 16, 32)	0
conv2d_3 (Conv2D)	(None, 16, 16, 32)	9248
max_pooling2d_3 (MaxPooling2	(None, 8, 8, 32)	0
conv2d_4 (Conv2D)	(None, 8, 8, 32)	9248
max_pooling2d_4 (MaxPooling2	(None, 4, 4, 32)	0
conv2d_5 (Conv2D)	(None, 4, 4, 32)	9248
max_pooling2d_5 (MaxPooling2	(None, 2, 2, 32)	0
conv2d_6 (Conv2D)	(None, 2, 2, 32)	9248
flatten_1 (Flatten)	(None, 128)	0
dense_1 (Dense)	(None, 1)	129

Total params: 47,265
Trainable params: 47,265
Non-trainable params: 0

Train	0.067
Validation	0.089
Test	0.081



Above: training and validation loss (mean-absolute error) at the 75th epoch and testing MAE.

Below: The graph of training and validation loss over time. Notice past the 75 epoch, validation loss stagnates while training loss continues to decrease, suggesting that the model is beginning to overfit.