1. Color, Linear space

4.1 NEWTON’S SUMMARY DRAWING of his experiments with light. Using a point source of light and a prism, Newton separated sunlight into its fundamental components. By reconverging the rays, he also showed that the decomposition is reversible.

From Foundations of Vision, by Brian Wandell, Sinauer Assoc., 1995

Source: Freeman @ MIT
Grassman’s Laws

- For color matches:
  - symmetry: \( U = V \iff V = U \)
  - transitivity: \( U = V \) and \( V = W \) \(\implies\) \( U = W \)
  - proportionality: \( U = V \iff tU = tV \)
  - additivity: if any two (or more) of the statements \( U = V \), \( W = X \), \( (U+W)=(V+X) \) are true, then so is the third

- These statements are as true as any biological law. They mean that additive color matching is linear.

Source: Freeman @ MIT
Color matching functions for a particular set of monochromatic primaries

Source: Freeman @ MIT
1.A. RGB Space

- An RGB color space is any additive color space based on the RGB color model.
Skin color detection in RGB space.
1. RGB Space (cont.)

- Skin detection


- Have a look of the Demo..
1.B. CIE XYZ color space

- Commission Internationale d'Eclairage, 1931
- “…as with any standards decision, there are some irritating aspects of the XYZ color-matching functions as well…no set of physically realizable primary lights that by direct measurement will yield the color matching functions.”
- “Although they have served quite well as a technical standard, and are understood by the mandarins of vision science, they have served quite poorly as tools for explaining the discipline to new students and colleagues outside the field.”

Source: Freeman @ MIT
1.B. CIE XYZ color space (cont.)

CIE XYZ: Color matching functions are positive everywhere, but primaries are imaginary. Usually draw x, y, where $x = X/(X+Y+Z)$

$y = Y/(X+Y+Z)$

Foundations of Vision, by Brian Wandell, Sinauer Assoc., 1995

Source: Freeman @ MIT
A qualitative rendering of the CIE (x,y) space. The blobby region represents visible colors. There are sets of (x, y) coordinates that don’t represent real colors, because the primaries are not real lights (so that the color matching functions could be positive everywhere).
1.B. CIE XYZ color space (cont.)

Figure 6.8 The volume of all visible colors in CIE XYZ coordinate space is a cone whose vertex is at the origin. Usually it is easier to suppress the brightness of a color, which we can do because to a good approximation perception of color is linear, and we do this by intersecting the cone with the plane $X + Y + Z = 1$ to get the CIE xy space shown in Figures 6.9 and 6.10.

Source: Freeman @ MIT
2. Nonlinear color space

- HSV (popular)
- Lab (uniform color space)
2.A. HSV Space

- HSV space is a transformation of RGB space that can describe colors in terms more natural to an artist. The name HSV stands for hue, saturation, and value.

\[
n = \begin{cases} 
0 & \text{if } \max = \min \\
60^\circ \times \frac{g-b}{\max - \min} + 0^\circ & \text{if } \max = r \text{ and } g \geq b \\
60^\circ \times \frac{g-b}{\max - \min} + 360^\circ & \text{if } \max = r \text{ and } g < b \\
60^\circ \times \frac{b-r}{\max - \min} + 120^\circ & \text{if } \max = g \\
60^\circ \times \frac{r-g}{\max - \min} + 240^\circ & \text{if } \max = b
\end{cases}
\]

\[
v = \max
\]

\[
s = \begin{cases} 
0, & \text{if } \max = 0 \\
\frac{\max - \min}{\max} = 1 - \frac{\min}{\max}, & \text{otherwise}
\end{cases}
\]
2.B Lab Space

- A Lab color space is a color-opponent space with dimension L for luminance and a and b for the color-opponent dimensions, based on nonlinearly-compressed CIE XYZ color space coordinates.
- Lab is now almost universally the most popular uniform space.
The forward transformation

\[ L = 116f(Y/Y_n) - 16 \]
\[ a = 500[f(X/X_n) - f(Y/Y_n)] \]
\[ b = 200\sqrt[3]{f(Y/Y_n) - f(Z/Z_n)} \]

where
\[ f(t) = t^{1/3}, \text{ for } t > 0.008856 \]
\[ f(t) = 7.787t + 16/116 \text{ otherwise} \]

Here \( X_n, Y_n \) and \( Z_n \) are the CIE XYZ tristimulus values of the reference white point.
OpenCV have a nice implementation for color conversion:

```c
# Convert between color spaces:
cvCvtColor(src, dst, code); // src -> dst
```

- `code = CV_<X>2<Y>`
- `<X>/<Y> = RGB, BGR, GRAY, HSV, YCrCb, XYZ, Lab, Luv, HLS`
- e.g.: `CV_BGR2GRAY`, `CV_BGR2HSV`, `CV_BGR2Lab`
Correspondence

- Matching points across images important for:
  - object identification (instance recognition)
  - object (class) recognition
  - pose estimation
  - stereo (3-d shape)
  - motion estimate
  - stitching together photographs into a mosaic
  - ....

Source: Freeman @ MIT
Correspondence using window matching

- Points are highly individually ambiguous...
- More unique matches are possible with small regions of image, i.e. image patches.

Source: Freeman @ MIT
Correspondence using window matching

Criterion function:

canline

error

disparity

Source: Freeman @ MIT
Sum of Squared (Pixel) Differences

\[ w_L \text{ and } w_R \text{ are corresponding } m \text{ by } m \text{ windows of pixels.} \]

\[ \text{We define the window function:} \]

\[ W_m (x, y) = \{ u, v \mid x - \frac{m}{2} \leq u \leq x + \frac{m}{2}, \, y - \frac{m}{2} \leq v \leq y + \frac{m}{2} \} \]

\[ \text{The SSD cost measures the intensity difference as a function of disparity:} \]

\[ C_r (x, y, d) = \sum_{(u, v) \in W_m (x, y)} [I_L (u, v) - I_R (u - d, v)]^2 \]
Even when the cameras are identical models, there can be differences in gain and sensitivity.
The cameras do not see exactly the same surfaces, so their overall light levels can differ.
For these reasons and more, it is a good idea to normalize the pixels in each window:

\[
\bar{I} = \frac{1}{|W_m(x,y)|} \sum_{(u,v) \in W_m(x,y)} I(u,v)
\]

Average pixel

\[
||I||_{W_m(x,y)} = \sqrt{\sum_{(u,v) \in W_m(x,y)} [I(u,v)]^2}
\]

Window magnitude

\[
\hat{I}(x,y) = \frac{I(x,y) - \bar{I}}{||I - \bar{I}||_{W_m(x,y)}}
\]

Normalized pixel

Source: Freeman @ MIT
Optical Flow:

2D velocity field describing the apparent motion in the images.

Source: J. J. Gibson, and Black
Problem: The compute of Optical Flow

Source: Black @ Brown
Optical flow constraint equation

**Differential approach:**

Brightness should stay constant as you track motion

\[ I(x + u \delta t, y + v \delta t, t + \delta t) = I(x, y, t) \]

1st order Taylor series, valid for small \( \delta t \)

\[ I(x, y, t) + u \delta t I_x + v \delta t I_y + \delta t I_t = I(x, y, t) \]

Constraint equation

\[ u I_x + v I_y + I_t = 0 \]

“BCCE” - Brightness Change Constraint Equation
Assume a single velocity for all pixels within an image patch

$$E(u, v) = \sum_{x,y \in \Omega} \left( I_x(x, y)u + I_y(x, y)v + I_t \right)^2$$

Solve with:

$$\begin{bmatrix} \sum I_x^2 & \sum I_x I_y \\ \sum I_x I_y & \sum I_y^2 \end{bmatrix} \begin{pmatrix} u \\ v \end{pmatrix} = -\begin{pmatrix} \sum I_x I_t \\ \sum I_y I_t \end{pmatrix}$$

On the LHS: sum of the 2x2 outer product tensor of the gradient vector

$$\left( \sum \nabla I \nabla I^T \right) \vec{U} = -\sum \nabla II_t$$

Source: Freeman @ MIT
The gradient constraint:

\[ I_x u + I_y v + I_t = 0 \]

\[ \nabla I \cdot \vec{U} = 0 \]

Defines a line in the \((u,v)\) space

Normal Flow:

\[ u_\perp = -\frac{I_t \nabla I}{|\nabla I| \cdot |\nabla I|} \]
Barber pole (aperture problem)
Good Features to Track

\[
\begin{bmatrix}
\sum I_x^2 & \sum I_x I_y \\
\sum I_x I_y & \sum I_y^2
\end{bmatrix}
\begin{bmatrix}
u \\
v
\end{bmatrix} =
\begin{bmatrix}
\sum I_x I_t \\
\sum I_y I_t
\end{bmatrix}
\]

\[A \quad u = \quad b\]

When is This Solvable?
- A should be invertible
- A should not be too small due to noise
  - eigenvalues \(\lambda_1\) and \(\lambda_2\) of A should not be too small
- A should be well-conditioned
  - \(\lambda_1 / \lambda_2\) should not be too large (\(\lambda_1 =\) larger eigenvalue)

Both conditions satisfied when \(\min(\lambda_1, \lambda_2) > c\)
Harris corner detector

Auto-correlation matrix

\[
\begin{bmatrix}
\sum_{(x_k, y_k) \in W} (I_x(x_k, y_k))^2 & \sum_{(x_k, y_k) \in W} I_x(x_k, y_k)I_y(x_k, y_k) \\
\sum_{(x_k, y_k) \in W} I_x(x_k, y_k)I_y(x_k, y_k) & \sum_{(x_k, y_k) \in W} (I_y(x_k, y_k))^2
\end{bmatrix}
\]

- **Auto-correlation matrix**
  - captures the structure of the local neighborhood
  - measure based on eigenvalues of this matrix
    - 2 strong eigenvalues => interest point
    - 1 strong eigenvalue => contour
    - 0 eigenvalue => uniform region

- **Interest point detection**
  - threshold on the eigenvalues
  - local maximum for localization
Selecting Good Features

\[ \lambda_1 \text{ and } \lambda_2 \text{ are large} \]
Selecting Good Features

Source: Freeman @ MIT
Selecting Good Features

Source: Freeman @ MIT
Dense Flow
Two approach to compute optical flow

- Horn-Schunck and Lucas-Kanade methods work only for small motion.
- If object moves faster, the brightness changes rapidly, 2x2 or 3x3 masks fail to estimate spatiotemporal derivatives.
- Pyramids can be used to compute large optical flow vectors.
Pyramids are needed for large motion

Lucas-Kanade without pyramids

Fails in areas of large motion

Source: Shafique @ UCF
Pyramids are needed for large motion

Lucas-Kanade with Pyramids

Source: Shafique @ UCF
Pyramid Searching


Source: Irani & Basri
function a=pyramidFlow(im1, im2, ainit, iters, levels)
    % build image pyramids
    % divide a1 and a4 by 2 for each level
    % starting with coarse level
    % warp im2 by current flow
    % estimate flow
    % project flow to next level
    % repeat to finest level
Gaussian Pyramid

High resolution  Low resolution

Source: Irani
Gaussian Pyramid

Source: Irani
An Isotropic Gaussian Filter

- The picture shows a smoothing kernel proportional to

\[ g(x, y) = \frac{1}{2\pi\sigma^2} \exp\left( -\frac{(x^2 + y^2)}{2\sigma^2} \right) \]

- To smooth image, convolve with this filter.
- Demo: gaussianScript3.m for denoising.

Source: Black, Ponce and Forsyth
Gaussian Pyramid

\[ G_0 = \text{Image} \]
\[ G_1 = (G_0 \ast \text{gaussian}) \downarrow 2 \]
\[ G_2 = (G_1 \ast \text{gaussian}) \downarrow 2 \]
\[ G_3 = (G_2 \ast \text{gaussian}) \downarrow 2 \]
\[ G_4 = (G_3 \ast \text{gaussian}) \downarrow 2 \]

Source: Irani and Black
Aliasing

- Why smooth before sub-sampling?
  - Can’t shrink an image by taking every second pixel
  - If we do, characteristic errors appear. The common phenomenon:
    - Wagon wheels rolling the wrong way in movies
    - Checkerboards misrepresented in ray tracing
Resample the checkerboard by taking one sample at each circle.
- In the case of the top left board, new representation is reasonable.
- Top right also yields a reasonable representation.
- Bottom left is all black (dubious) and bottom right has checks that are too big.
Affine Flow

\[
E(a) = \sum_{x, y \in I} (\nabla I^T u(x; a) + I_t)^2
\]

\[
u(x; a) = \begin{bmatrix} u(x; a) \\ v(x; a) \end{bmatrix} = \begin{bmatrix} a_1 + a_2 x + a_3 y \\ a_4 + a_5 x + a_6 y \end{bmatrix}
\]
Linear Basis

- Can be thought as just another set of linear basis functions!

\[
\mathbf{u}(\mathbf{x}; \mathbf{c}) = \sum_{j=1}^{n} a_j \mathbf{b}_j(\mathbf{x})
\]


Source: Black @ Brown
Affine Transformation

\[
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}^* = \begin{bmatrix}
  a_1 & a_2 \\
  a_4 & a_5 \\
  0 & 0
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix} + \begin{bmatrix}
  a_3 \\
  a_6
\end{bmatrix}
\]

Homogeneous coordinates

Linear transformation

Translation

Source: Black @ Brown
What can be represented?

What does this do?

\[
\begin{bmatrix}
  x' \\
  y' \\
  1
\end{bmatrix} =
\begin{bmatrix}
  1 & 0 & a3 \\
  0 & 1 & a6 \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]
What can be represented?

What does this do?

\[
\begin{bmatrix}
x' \\
y' \\
1
\end{bmatrix} = \begin{bmatrix}
\cos \theta & -\sin \theta & 0 \\
\sin \theta & \cos \theta & 0 \\
0 & 0 & 1
\end{bmatrix} \begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
\]
What can be represented?

What does this do?

\[
\begin{bmatrix}
x' \\
y' \\
1
\end{bmatrix} = \begin{bmatrix} 2 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}
\]
What can be represented?

What does this do?

\[
\begin{bmatrix}
x' \\
y' \\
1
\end{bmatrix} =
\begin{bmatrix}
1 & 0 & 0 \\
2 & 1 & 0 \\
0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
x \\
y \\
1
\end{bmatrix}
\]
Composition and inversion

- Apply two affine transforms successively.
- Represent affine transformations as matrices
  \[ x^* = A(BI) = (AB)x \]

- Inversion
  \[ x = (AB)^{-1}x^* \]
Transforming images

- Affine transformation is applied to image coordinates $x, y$

$$I' = I(A[x, y, 1]^T)$$

How do we do this in Matlab? What are the issues?
Matlab code to generate image coordinates

```
>> [y,x]=meshgrid(1:10, 1:10)

Get the image coordinates:

y =
    1     2     3     4     5     6     7     8     9    10
    2     3     4     5     6     7     8     9    10
    3     4     5     6     7     8     9    10
    4     5     6     7     8    10
    5     6     7     8    10
    6     7     8    10
    7     8    10
    8    10
    9    10
    10

x =
    1     1     1     1     1     1     1     1     1     1
    2     2     2     2     2     2     2     2     2     2
    3     3     3     3     3     3     3     3     3     3
    4     4     4     4     4     4     4     4     4     4
    5     5     5     5     5     5     5     5     5     5
    6     6     6     6     6     6     6     6     6     6
    7     7     7     7     7     7     7     7     7     7
    8     8     8     8     8     8     8     8     8     8
    9     9     9     9     9     9     9     9     9     9
   10    10    10    10    10    10    10    10    10    10
```

Source: Black @ Brown
Forwards Warp

- Difficult to use due to the discrete raster in digital image.

![Image at t]

Contributes to 4 pixels.

Source: Black @ Brown
Backwards Warp

Image at $t$

Contributions from 4 pixels.

Source: Black @ Brown
Backwards Warp

Image at t

Image at t+1

Contributions from 4 pixels – bi-linear interpolation

Every pixel at time t+1 defined.

Source: Black @ Brown
Interpolation

- Possible interpolation filters:
  - nearest neighbor
  - bilinear
  - bicubic (interpolating)

- Needed to prevent “jaggies”
- When iteratively warping, always *compose the warps* and warp the original image

Source: Szeliski, Fleet, and Black
Matlab function: interp2

- **INTERP2** 2-D interpolation
- **ZI = interp2(X,Y,Z,XI,YI)**
  - Interpolates to find ZI, the values of the underlying 2-D function Z at the points in matrices XI and YI.
  - Matrices X and Y specify the points at which the data Z is given.
  - Out of range values are returned as NaN.
Warping images

Example warps:

- Translation
- Rotation
- Aspect
- Affine
- Perspective
- Cylindrical

Source: Black @ Brown
Fun House Mirror Simulation

Source: http://www.ele.uri.edu/~hanx/MirrorSimulation.htm
Funny warp

Source: http://www.ele.uri.edu/~hanx/MirrorSimulation.htm
Warping images

```matlab
function warpim = warpImage(image, a)
    warpim = zeros(size(image));
    [y, x] = meshgrid(1:size(image, 2), 1:size(image, 1));
    % find the center of the image
    % compute the new pixel locations x2 and y2
    warpim = interp2(y, x, image, y2, x2, 'linear');
    % fix NaNs
    ind = find(~(warpim > 0 & warpim < 256));
    warpim(ind) = 0.0;
```
Affine flow

Motion (flow) between frames

\[
\begin{bmatrix}
  u \\
  v \\
  1
\end{bmatrix} =
\begin{bmatrix}
  a_1 & a_2 & a_3 \\
  a_4 & a_5 & a_6 \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]

Transformation of pixels

\[
\begin{bmatrix}
  x' \\
  y' \\
  1
\end{bmatrix} =
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix} +
\begin{bmatrix}
  a_1 & a_2 & a_3 \\
  a_4 & a_5 & a_6 \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]

Use when transforming pixels

\[
\begin{bmatrix}
  x' \\
  y' \\
  1
\end{bmatrix} =
\begin{bmatrix}
  a_1 + 1 & a_2 & a_3 \\
  a_4 & a_5 + 1 & a_6 \\
  0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
  x \\
  y \\
  1
\end{bmatrix}
\]
Optimization

\[ E(\mathbf{a}) = \sum_{x,y \in R} (I_x u + I_y v + I_t)^2 \]

\[
\begin{bmatrix}
    u \\
    v \\
    1
\end{bmatrix} =
\begin{bmatrix}
    a1 & a2 & a3 \\
    a4 & a5 & a6 \\
    0 & 0 & 1
\end{bmatrix}
\begin{bmatrix}
    x \\
    y \\
    1
\end{bmatrix}
\]

\[ E(\mathbf{a}) = \sum_{x,y \in R} (I_x a_1 x + I_x a_2 y + I_x a_3 + I_y a_4 x + I_y a_5 y + I_y a_6 + I_t)^2 \]
Optimization

\[
E(\mathbf{a}) = \sum_{x,y \in \mathbb{R}} \left( I_x a_1 x + I_x a_2 y + I_x a_3 + I_y a_4 x + I_y a_5 y + I_y a_6 + I_t \right)^2
\]

Differentiate wrt the \( a_i \) and set equal to zero.

\[
\begin{bmatrix}
\Sigma I_x^2 x^2 & \Sigma I_x^2 xy & \Sigma I_x^2 x & \Sigma I_x I_y x^2 & \Sigma I_x I_y xy & \Sigma I_x I_y x \\
\Sigma I_x^2 xy & \Sigma I_x^2 y^2 & \Sigma I_x^2 y & \Sigma I_x I_y xy & \Sigma I_x I_y y^2 & \Sigma I_x I_y y \\
& & & \ddots & & \\
\end{bmatrix}
\begin{bmatrix}
a_1 \\
a_2 \\
a_3 \\
a_4 \\
a_5 \\
a_6 \\
\end{bmatrix}
= 
\begin{bmatrix}
-\Sigma I_x I_t x \\
-\Sigma I_x I_t y \\
-\Sigma I_y I_t x \\
-\Sigma I_y I_t y \\
-\Sigma I_y I_t \\
-\Sigma I_y I_t \\
\end{bmatrix}
\]
function a=basicFlow(im1, im2, ainit, iters)
    % warp im1 by current flow parameters
    % compute derivatives
    % build structure tensor
    % solve for motion parameters
    % (exclude boundary pixels and non-overlapping pixels from the analysis) – mark them with nan’s
    % update current flow parameters
    % repeat
Coarse to Fine Pyramid Searching

function a = pyramidFlow(im1, im2, ainit, iters, levels)

  % build image pyramids
  % divide a3 and a6 by 2 for each level
  % starting with coarse level
  % warp im1 by current flow
  % estimate flow
  % project flow to next level
  % repeat to finest level

Source: Black @ Brown