Image Formation

Image Formation

- First photograph due to Niepce – 1822

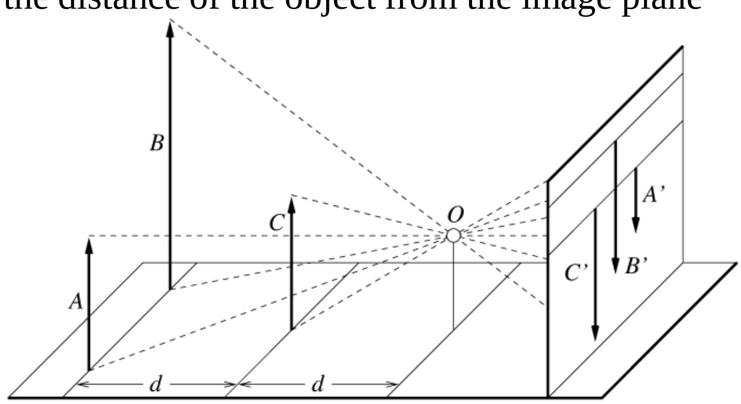
- Now: various films and digital (CCD, CMOS)
- “Spectrum” of parameters

The Camera's Job

- Basically, the job of the camera (no matter what the format) is *mapping* the 3D world onto a 2D plane
  - Yes, even a 3D camera does this…it just does it twice
- The operation is called a "projection"
- There are two projections that we study/utilize
  - Perspective
  - Orthographic

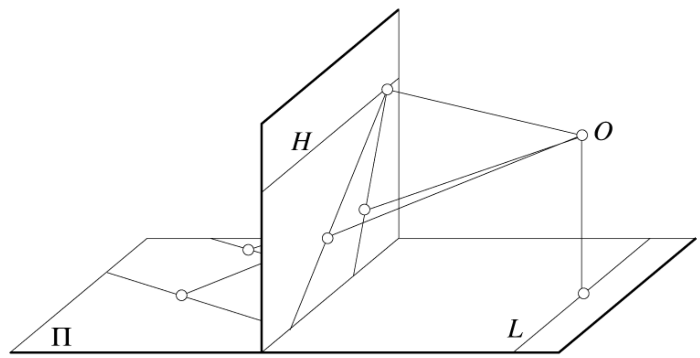
Perspective Projection

- Size of the object on the image plane is dependent on the distance of the object from the image plane



Perspective Projection

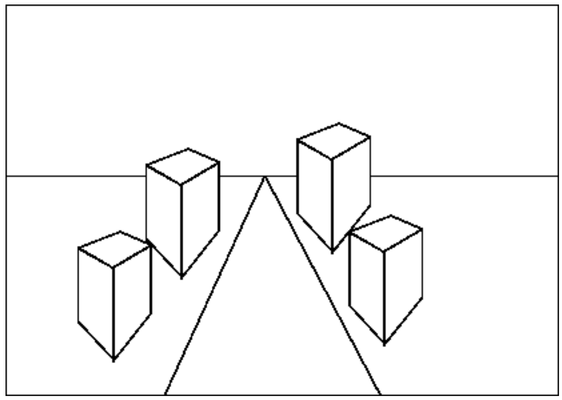
- Parallel lines in the scene intersect at the horizon on the image plane



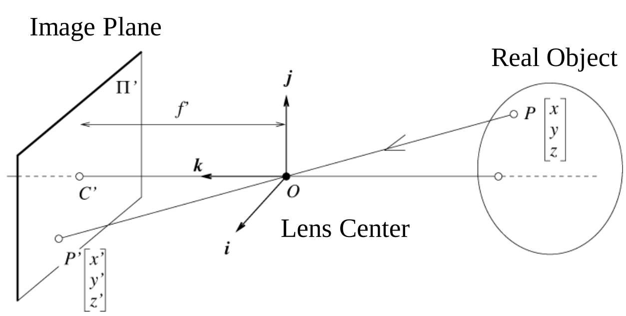
Vanishing Points

- Sets of parallel lines meet at a different points (for a given direction)
  - The *vanishing point* for this direction
- Sets of parallel lines on the same plane lead to *collinear* vanishing points.
  - The line is called the *horizon* for that plane
- An easy way to spot poorly faked images

Vanishing Points



Equation of Perspective Projection



- **(x, y, z) -> (f x/z, f y/z, -f) (by considering similar triangles – simple geometry)**
- **We ignore the 3rd coordinate since all image points are in the image plane**
- **Multiple real objects will map to the same image point**
- **Why is that last point important?**

Accuracy

- Most of computer vision is geared towards recognizing an object within a scene
  - For these applications, general knowledge of the perspective projection is enough
- Some applications use computer vision to make measurements
  - For these applications accuracy is required
  - Therefore, we must calibrate the camera system (lens, image plane)

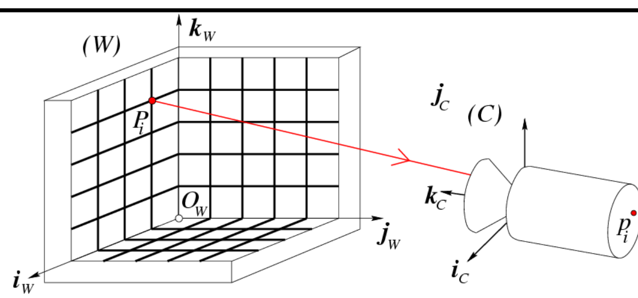
Camera Parameters

- Optical axis relative to the image plane
- Angle between optical axis and image plane
- Focal length of the lens
- Size of the pixels in the image plane
- Position of camera in real world
- Orientation of camera in real world

Camera Calibration

- Through the use of appropriate target scenes and test set ups all these parameters can be derived
- Once derived, images can be compensated based on parameters thus creating a measurement device

Camera Calibration

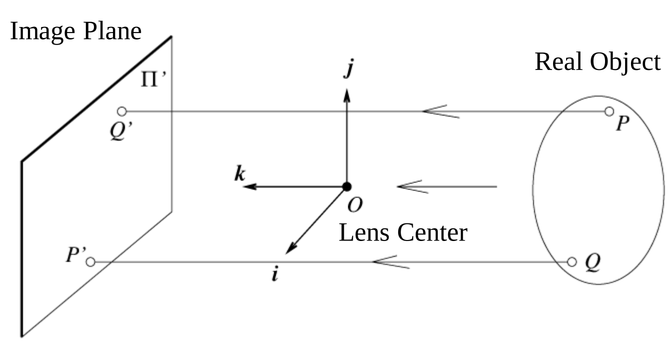


- **We know where the grid points are (relative to one another)**
- **We know where the camera is (relative to the grid points)**
- **Thus, we know where the [image of] the grid points should lie in the image plane**
- **We can create calibration factors based on where they should lie and where they actually lie in the image plane**
- **Search web for "Roger Tsai" and "Camera Calibration" for details**
  - **It's very math-intensive**

Orthographic Projection

- If the camera is far from the objects relative to the depth (height?) of the objects
  – i.e. distance from scene objects to camera is constant (flat scenes)
  – Provides an approximation of the perspective projection
  – Sometimes useful for simplifying various algorithms where depth is not a concern
    - Overhead aerial (air to ground) applications

Orthographic Projection



Orthographic Projection

- Aerial image

Lens

- For our purposes the lens will
  - Provide a means of focus
  - Provide a means for more efficient light ray capture
- We won't go into the mathematics of lens design here – that's better covered in a physics course

Image Representation

- Two dimensional array of values
  - Typically byte or integer (unsigned)
  - When performing operations you must do range checking
- Each array location is referred to as a ***pixel***
  - Short for "picture element"
- Height and width of the array determine the image's spatial resolution
- Bit depth of each pixel determines the ***intensity resolution***
  - 8-bit image – each pixel is in the range of 0..255
  - If it's greater than 8 you must scale it prior to display as most monitors can only display 256 gray levels
  - 24-bit color images are merely three 8-bit color images "stacked" together

File Storage

- There are many image file formats in use today
  - jpg –JPEG
  - png – portable network graphics
  - tiff – tagged image file format
  - j2k – JPEG 2000
  - raw – no meta-data, just image data
  - gif – graphics interchange format
  - bmp – bitmap

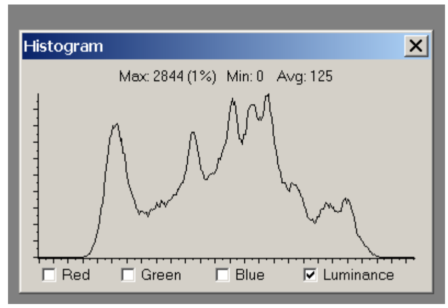
File Storage

- Differences include
  - Compressed vs. uncompressed
  - Inclusion of image meta-data
  - Inclusion of device meta-data
  - Inclusion of processing meta-data
  - Allowable bit depth
  - etc.

Preprocessing

Intensity Histogram

- Histogram
  - Distribution of pixel intensities
  - One dimensional array of integers
  - Size of the array is directly related to the pixel resolution

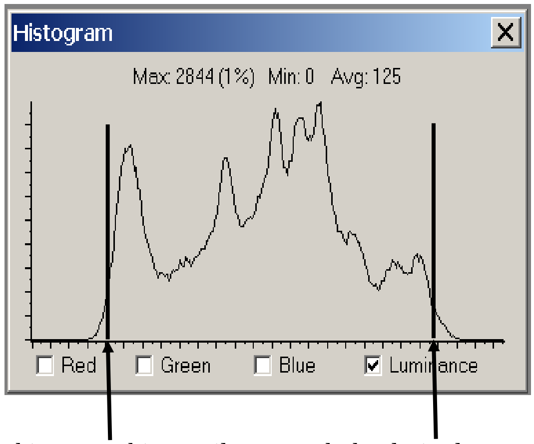


Contrast Enhancement

- ***Histogram Stretch***
  - Compute the image histogram
  - Specify dark and bright cut-off points
    - This is usually done percentiles of the distribution – then converted to actual intensity cutoffs
    - Alternatively, may be specified as two fixed intensity cutoffs
  - For each pixel I(i, j) compute:

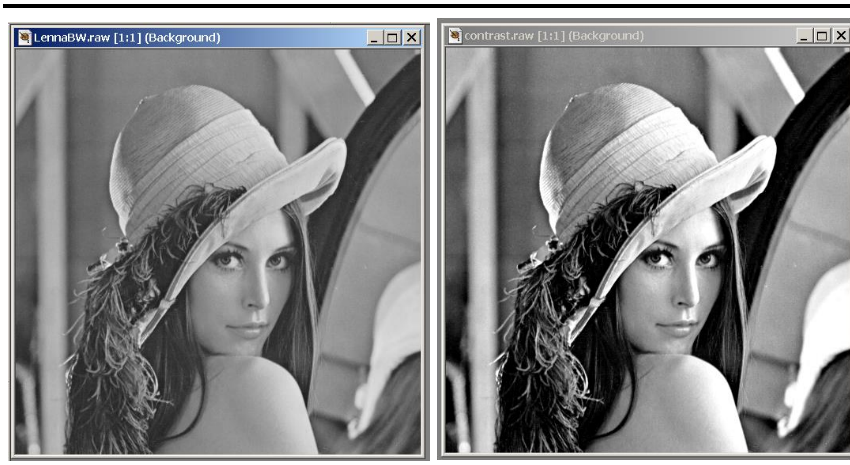
$$I'(i,j) = \left(I(i,j) - cutoff_{dark}\right) \bullet \frac{255}{(cutoff_{bright} - cuttoff_{dark})}$$

Percentiles to Intensity Cutoffs

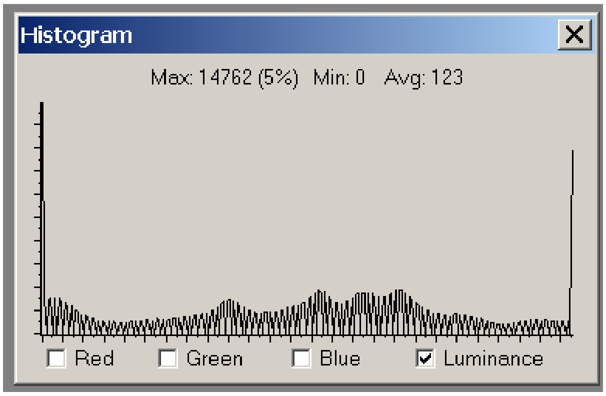


Count pixels in histogram bins until you reach the desired percentile values to get the two cutoff points

Contrast Enhancement

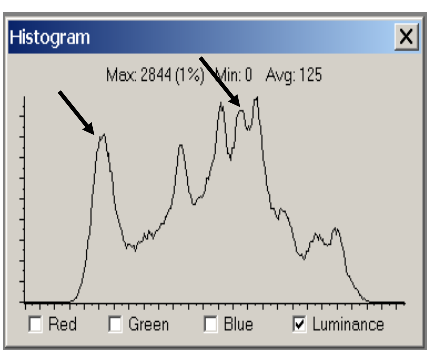


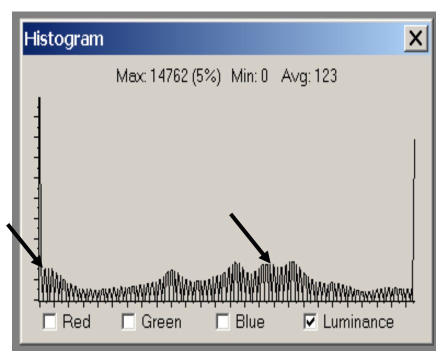
Resultant Histogram



Note: The scale on the vertical axis has changed due to the display program
The high frequency bumps are due to the multiply operation

Before/After Comparison





Contrast Enhancement

- Why do we want to do this?
- Because contrast enhancement will bring out some features and suppress others
- The problem is that it's not easy to control
- Basically, it's an operation that is more for human consumption than computer vision
- We study it as an introduction to the histogram

Contrast Enhancement

- ***Binarize/Threshold***
  - Choose a threshold τ
  - Perform the following [parallel] operation on every pixel

$$\textbf{\textit{if}} \quad \begin{matrix} I(i,j) > \tau \\ I(i,j) \le \tau \end{matrix} \quad \textbf{\textit{then}} \quad \begin{matrix} I'(i,j) = 255 \\ I'(i,j) = 0 \end{matrix}$$

  - The result is a 1-bit image (represented in 8-bits for display)
  - The problem is selecting the proper value of τ

Binarize/Threshold

Noise Reduction

- Median Filter
  - Define a neighborhood around every pixel
    - Should be odd dimensions (but not absolutely necessary)
  - For every pixel *I(i, j)*
    - Numerically sort the pixel values in the neighborhood
    - Replace *I(i, j)* with the median (middle) value of the sorted neighborhood
  - The result is the removal of "salt and pepper" artifacts
  - Note that this operation is parallel in that all pixels perform their operation simultaneously
    - So, how do you do this on a sequential machine?

Median Filter

Little spots

Noise Reduction

- Outlier Filter
  - Define a neighborhood around every pixel
    - Should be odd dimensions (but not absolutely necessary)
  - For every pixel *I(i, j)*
    - Compute the average pixel value of the neighborhood, $\overline{n(i, j)}$
    - If $|I(i, j) - \overline{n(i, j)}| >$ *threshold* then replace *I(i, j)* with the neighborhood average
    - Result will be sensitive to the selected threshold
  - The result is the reduction of "salt and pepper" artifacts
  - Note that this operation is parallel in that all pixels perform their operation simultaneously

Outlier Filter

Little spots

What happens at the edges?

- Since the outlier filter uses a symmetrical neighborhood around each pixel you have to do something about the edges and corners
- One option is to not process the edges and corners
  - This is a problem if the neighborhood is large
- One option is to use a truncated neighborhood
  - This makes implementation difficult due to the heterogeneous processing
- One option is to reflect the edges and corners

Enlarging by Reflection

Assume a 3x3 kernel

Original image

A larger kernel requires additional reflection

238	238	237	234	227	223	216	216
238	238	237	234	227	223	216	216
229	229	227	224	220	225	221	221
205	205	212	221	220	225	220	220
177	177	192	213	207	212	217	217
164	164	180	211	208	209	215	215
190	190	194	220	212	210	219	219
190	190	194	220	212	210	219	219

Why Reflect?

- Why not just pad with 0 (or some other value)?
- Because it alters the results
- We'll see how critical this is later when we look at edge detection

Homework – due beginning of class next week

- Write a program to:
  - Read an image file (BMP or PNG format will be supplied)
  - Separate the image into three components (red-green-blue)
  - On the green component:
    - Perform a histogram stretch operation using the 10^{th} and 90^{th} percentile bins as cutoff points
      - Compute the mean of the resultant image
    - Perform a binarization operation using a threshold value of 128
      - Compute the mean of the resultant image
    - Perform a 3x3 median filter operation with reflection image padding
      - Compute the mean of the resultant image
    - Perform a 3x3 outlier filter operation using a threshold value of 50
      - Compute the mean of the resultant image
    - Perform an Otsu optimal threshold operation
      - Compute the mean of the resultant image
  - Print out the mean value for each operation. For the Otsu algorithm, print out the threshold you calculated.
  - Write the results of each operation to an output image file (BMP, PNG)

Notes on homework

- Make sure you start from the original input image for each of the four output processes
- When writing images to a file use the RGB format where all three components are set to the modified green component, that is

  output_red = output_green = output_blue = modified_green
- DO NOT USE JPEG OR ANY OTHER LOSSY COMPRESSION FILE FORMATS (lossless JPEG is fine)

Reading

- See syllabus