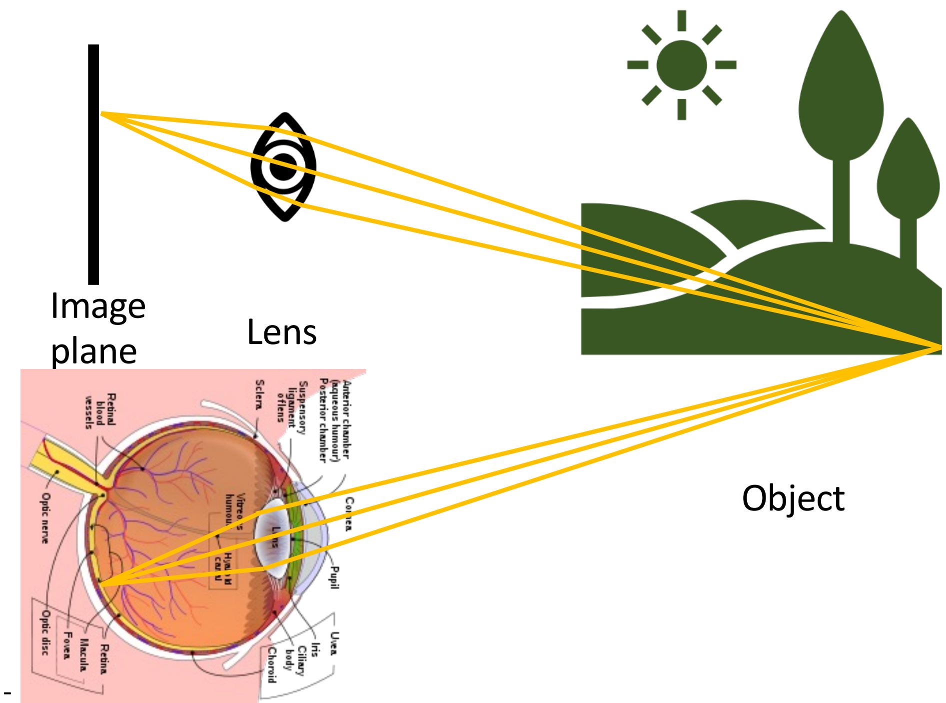
# CS440/ECE448 Lecture 14 Computer Vision: Image Formation

Mark Hasegawa-Johnson, 2/2024





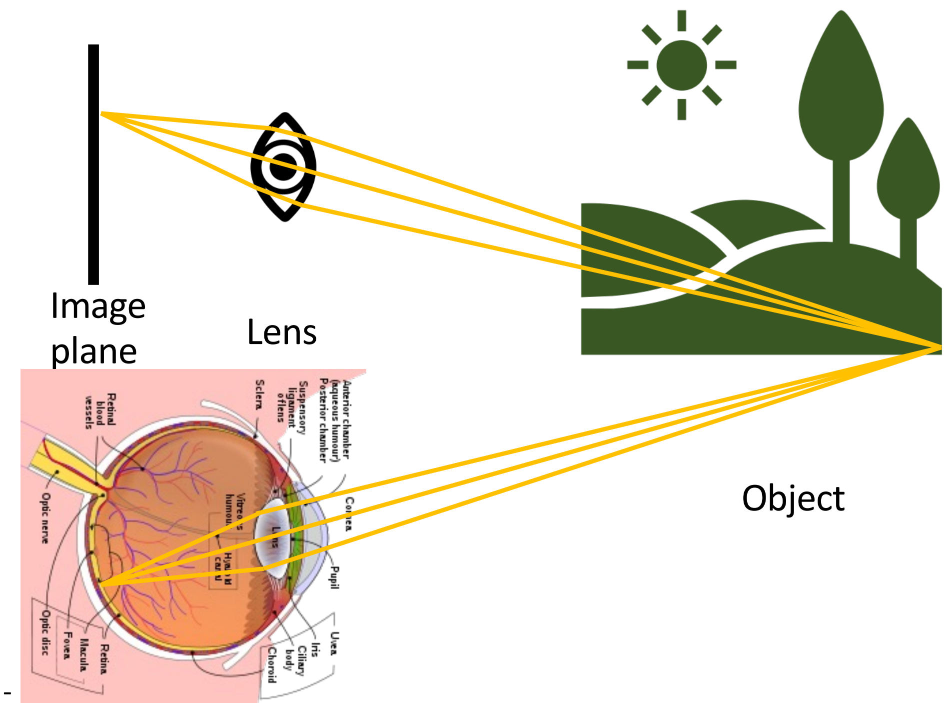

By Rhcastilhos. And Jmarchn. - Schematic_diagram_of_the_human_eye_with_English_annotations.svg, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=1597930

# Outline

- Space
  - Pinhole camera equations
  - Vanishing point
- Color
  - Structure of the eye
  - RGB displays
  - Color features: YPrPb
- Edges
  - Things that look like edges
  - Edge detection: the difference-of-Gaussians filter

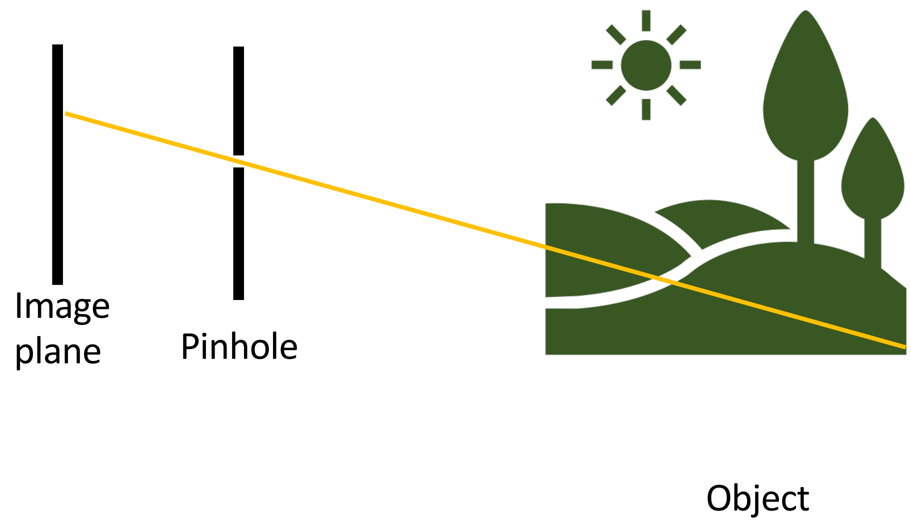
# Lenses and focus

- The lens in your eye collects light.
- Light that passes directly through the center of the lens is not bent.
- Light that passes above center is bent back toward center, and vice versa, so that it can all be collected in the same point on the image plane.



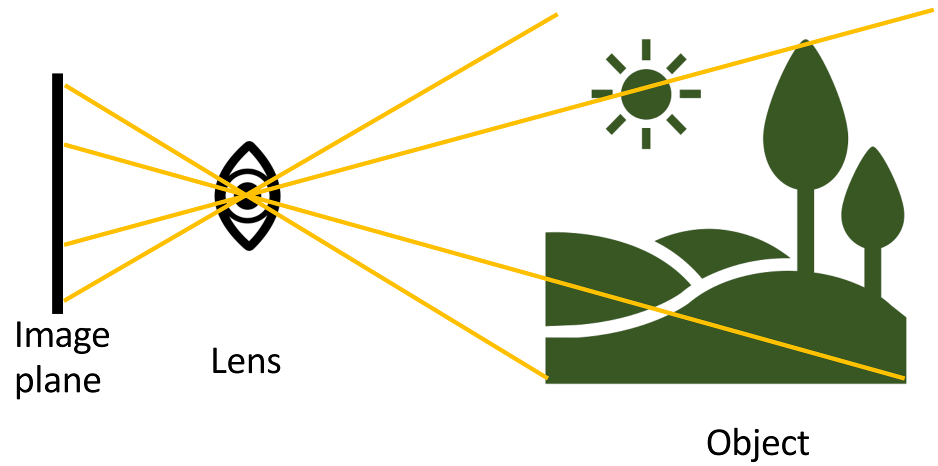
# The “pinhole camera” approximation

- A “pinhole camera” is a camera that only allows light through a very small hole.
- Disadvantage: A pinhole camera gets much less light than a lens (because the hole is smaller).
- Advantage: A pinhole camera focuses on all objects, at every distance, simultaneously.



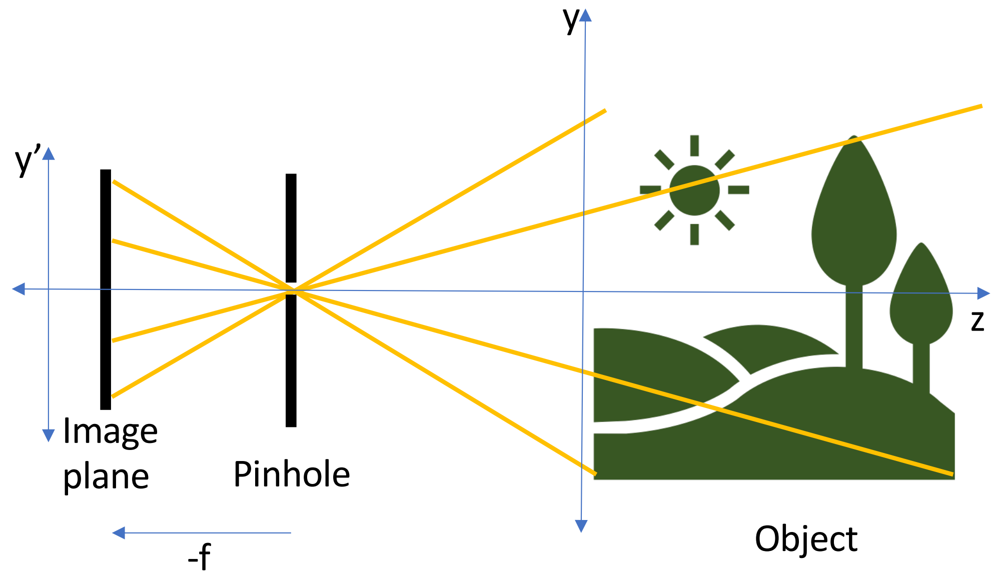
# Converting a 3D world to a 2D picture

- Different spots in the real world are projected onto different points in the image plane.
- Light that passes through the center of the lens is not bent.
- Therefore, we can use the pinhole camera approximation to analyze the relationship between real world position (x,y,z) and position on the image plane (x',y').



# The pinhole camera equations

- Define the origin (0,0,0) to be the pinhole.
- Define (x,y,z) as position of the object: x is horizontal (into the slide), y is vertical (upward), z is away from the camera.
- Define (x',y') as the position on the image plane where the light strikes (upside down).
- Define f as the distance from the pinhole to the image plane.



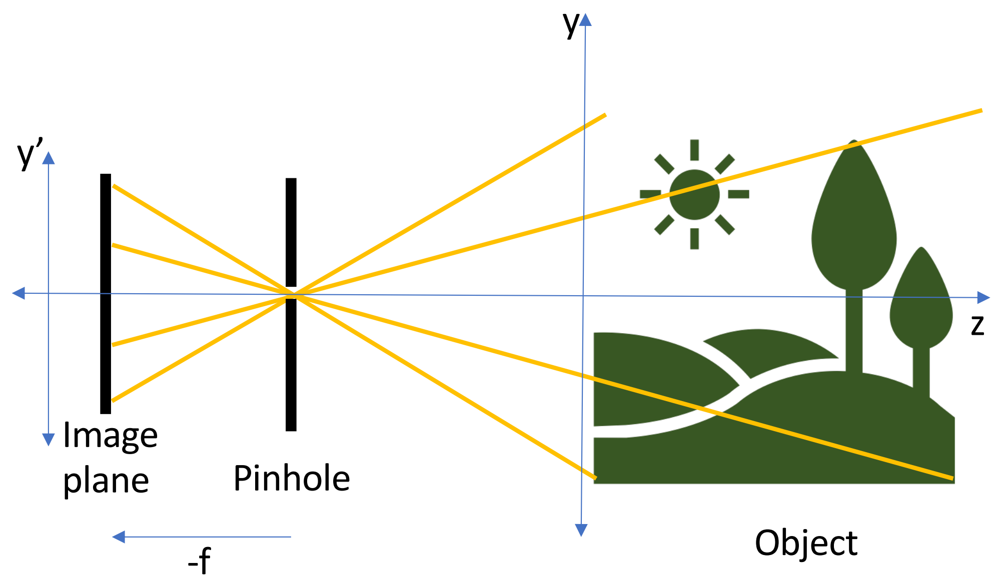
# The pinhole camera equations

- These are similar triangles! So

$$\frac{x'}{x} = \frac{y'}{y} = \frac{-f}{z}$$

- Solving for (x',y'), we get the pinhole camera equations:

$$\frac{x'}{f} = -\frac{x}{z}, \qquad \frac{y'}{f} = -\frac{y}{z}$$



# Outline

- Space
    - Pinhole camera equations
    - Vanishing point
- Color
    - Structure of the eye
    - RGB displays
    - Color features: YPrPb
- Edges
    - Things that look like edges
    - Edge detection: the difference-of-Gaussians filter

# Vanishing point

- When you take a picture, lines that are parallel in the real world appear to converge.
- The point toward which they converge is called the vanishing point. It lies on the horizon.
- The “horizon” is a line in the image, where a plane parallel to the ground passes through the pinhole.

# Vanishing point

- Recall the pinhole camera equations:

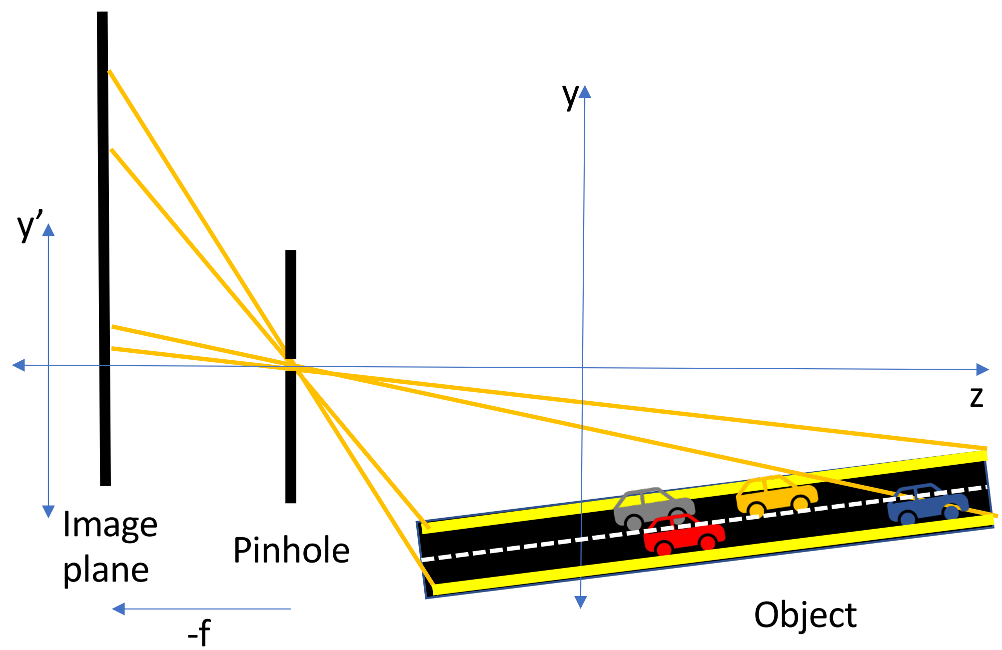
$$\frac{x'}{f} = -\frac{x}{z}, \qquad \frac{y'}{f} = -\frac{y}{z}$$

- Suppose we have a couple of lines:

Line 1: $x_1 = az + c_1, y_1 = bz + d_1$

Line 2: $x_2 = az + c_2, y_2 = bz + d_2$

These are parallel lines, so they have the same slopes, $a$ and $b$.



# Vanishing point

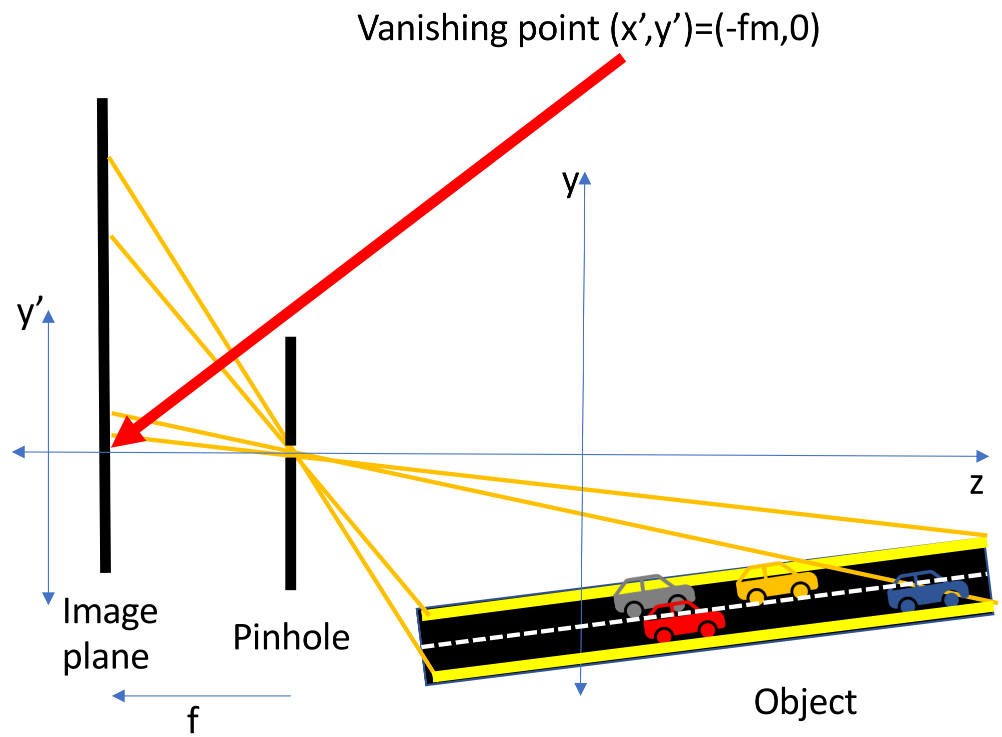
- Plug equations for the lines into the pinhole camera equations:

$$\frac{x_1'}{f} = -\frac{az + c_1}{z}, \qquad \frac{y_1'}{f} = -\frac{bz + d_1}{z}$$

$$\frac{x_2'}{f} = -\frac{az + c_2}{z}, \qquad \frac{y_2'}{f} = -\frac{bz + d_2}{z}$$

- As $z \rightarrow \infty$, the two lines converge to the vanishing point, which depends only on the ***slope*** of the lines, not on their shift:

$$(x', y') = (-fa, -fb)$$



# Vanishing point

$(-fa, -fb)$:

- Notice that, if the lines are on a flat plane (b=0; e.g., the ground), then the vanishing point is at the eye level of the camera (y'=0).
- The line y'=0 is therefore called the "horizon."
- If the lines are not on a flat plane, they would not converge at y'=0.

# Quiz

Try the quiz!

https://us.prairielearn.com/pl/course_instance/147925/assessment/2398230

# Outline

- Space
  - Pinhole camera equations
  - Vanishing point
- Color
  - Structure of the eye
  - RGB displays
  - Color features: YPrPb
- Edges
  - Things that look like edges
  - Edge detection: the difference-of-Gaussians filter

# Color spaces: RGB

- Every natural object reflects a continuous spectrum of colors.
- However, the human eye only has three color sensors:
  - Red cones are sensitive to lower frequencies
  - Green cones are sensitive to intermediate frequencies
  - Blue cones are sensitive to higher frequencies

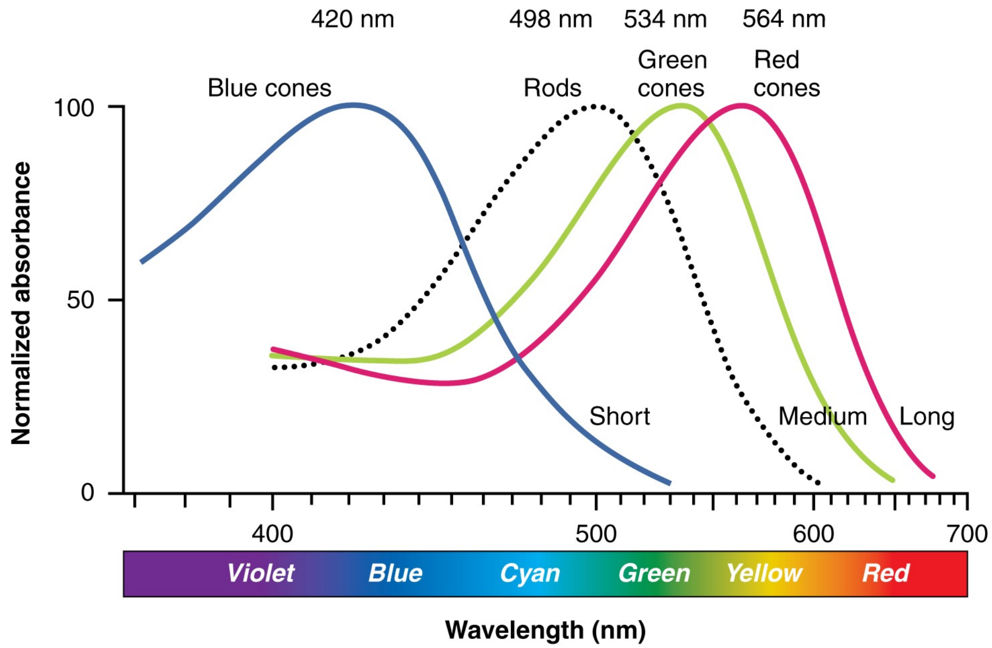
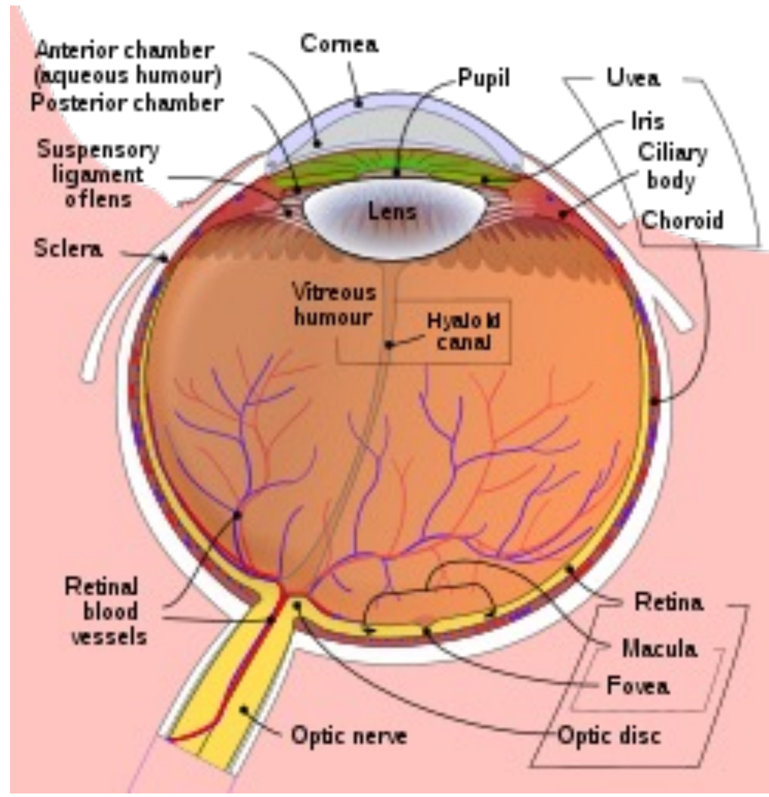



Illustration from Anatomy & Physiology, Connexions Web site. [http://cnx.org/content/col11496/1.6/](http://cnx.org/content/col11496/1.6/), Jun 19, 2013.

# Structure of the eye

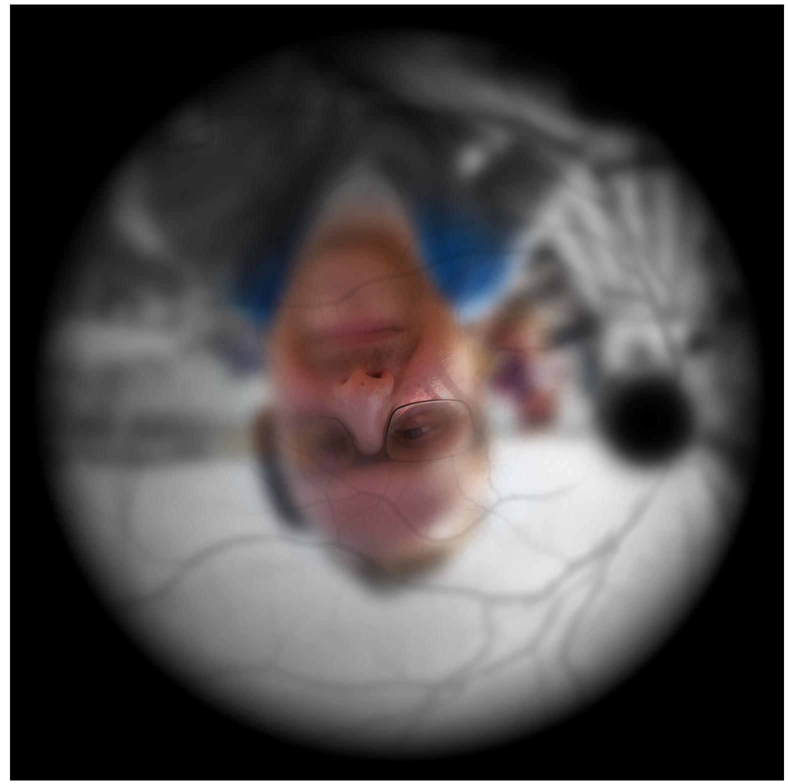
- Cones (color-sensitive cells) are located in only a small area, close to the fovea
- Rods (black-and-white cells) are spread more widely.



# Structure of the eye

- Because we only have cones in the center of the eye, we can only actually see color in the center.
- The colors that you believe you see, in the periphery of your vision, are being filled in from memory by your pre-conscious visual processes (optic nerve and striate cortex).

Illustration of image as 'seen' by the retina independent of optic nerve and striate cortex processing.



# Outline

- Space
  - Pinhole camera equations
  - Vanishing point
- Color
  - Structure of the eye
  - RGB displays
  - Color spaces: YPrPb, HSV
- Edges
  - Things that look like edges
  - Edge detection: the difference of Gaussians filter

# Color spaces: RGB

- Every natural object reflects a continuous spectrum of colors.
- However, the human eye only has three color sensors:
  - Red cones are sensitive to lower frequencies
  - Green cones are sensitive to intermediate frequencies
  - Blue cones are sensitive to higher frequencies

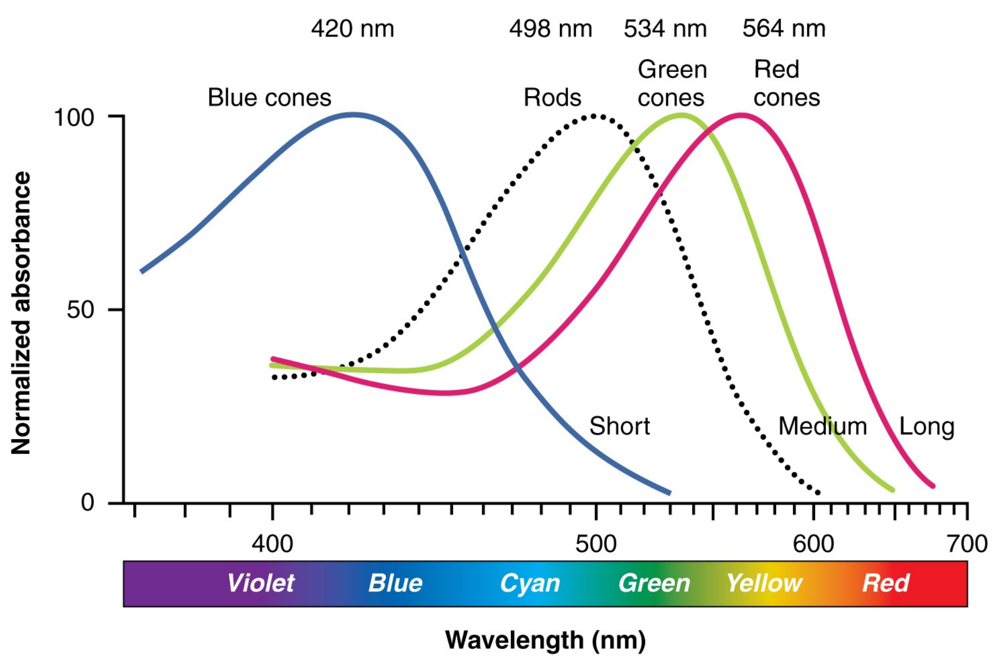
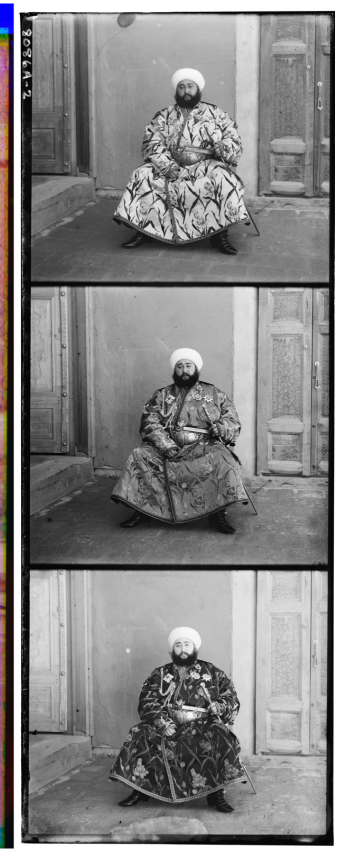



Illustration from Anatomy & Physiology, Connexions Web site. [http://cnx.org/content/col11496/1.6/](http://cnx.org/content/col11496/1.6/), Jun 19, 2013.

# Color spaces: RGB

A photograph of Mohammed Alim Khan (1880–1944), Emir of Bukhara, taken in 1911 by Sergey Prokudin-Gorsky using three exposures with blue, green, and red filters.

- By activating LED or other display hardware at just three discrete colors (R, G, and B), it is possible to fool the human eye into thinking that it sees a continuum of colors.
- Therefore, a so-called "color" camera is really three different black-and-white photographs:
  - R(x',y') is the brightness of red light at position (x',y')
  - G(x',y') is brightness of green.
  - B(x',y') is brightness of blue.



By Sergei Prokudin-Gorskii - Taken from the Library of Congress' website and converted from TIFF to PNG.TIFF file from LOC, Public Domain, https://commons.wikimedia.org/w/index.php?curid=1470606

# Outline

- Space
  - Pinhole camera equations
  - Vanishing point
- Color
  - Structure of the eye
  - RGB displays
  - Color features: YPrPb
- Edges
  - Things that look like edges
  - Edge detection: the difference-of-Gaussians filter

# Color features: Luminance

- The "grayscale" image is often computed as the average of R, G, and B intensities, i.e., $I[x', y'] = \frac{1}{3}(R(x', y') + G(x', y') + B(x', y'))$
- The human eye, on the other hand, is more sensitive to green light than to either red or blue.
- The intensity of light, as viewed by the human eye, is well approximated by the standard ITU-R BT.601:

$$Y(x', y') = 0.299R(x', y') + 0.587G(x', y') + 0.114B(x', y')$$

- The signal $Y(x', y')$ is called the **luminance** of light at pixel $(x', y')$.

# Color features: YPrPb

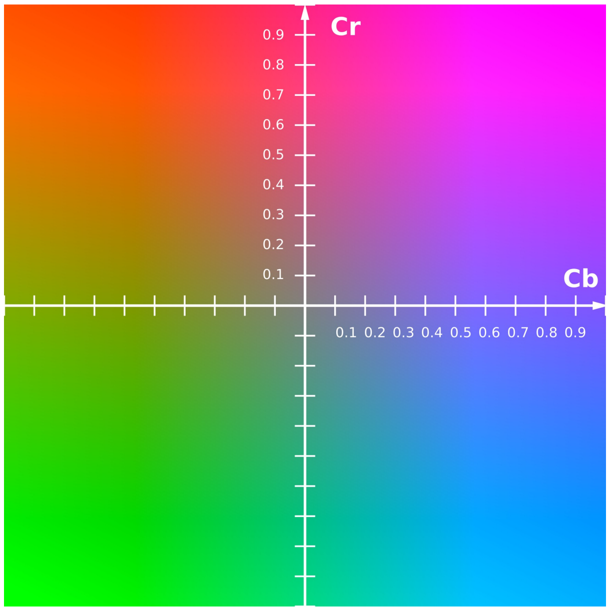
- The human eye is much more sensitive to spatial variation in luminance (brightness) than to spatial variation in chrominance (color)
- For this reason, the JPG image coding standard represents luminance, $Y(x',y')$, at twice the spatial resolution of chrominance:
  - First, JPG converts (R,G,B) into (Y,Pr,Pb), where Pr and Pb represent the "degree of redness" and "degree of blueness."
  - Second, JPG downsamples $Pr(x',y')$ and $Pb(x',y')$, so that they have ½ as many rows and ½ as many columns as $Y(x',y')$.
- For computer vision, we can use the same logic: represent Pr and Pb at half the resolution that we use for luminance.

# Color features: Chrominance

- Chrominance = color-shift of the image.
- We measure $P_R$=red-shift, and $P_B$=blue-shift, relative to luminance (luminance is sort of green-based, remember?)
- We want $P_R(x', y')$ and $P_B(x', y')$ to describe only the color-shift of the pixel, not its average luminance.
- We do that using

$$\begin{bmatrix} Y \\ P_B \\ P_R \end{bmatrix} = \begin{bmatrix} \vec{v}_Y \\ \vec{v}_B \\ \vec{v}_R \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

Where $sum(\vec{v}_R) = sum(\vec{v}_B) = 0$.



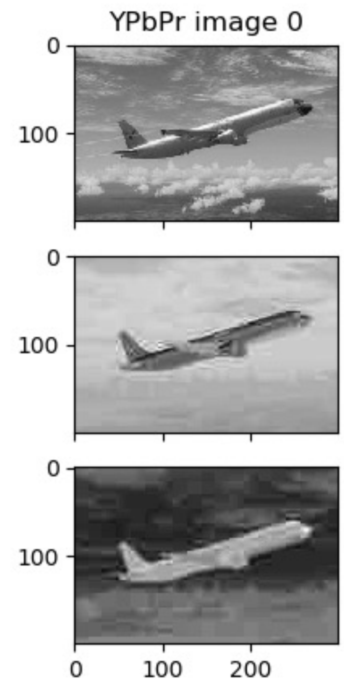

Cr and Cb, at Y=0.5
Simon A. Eugster, own work.

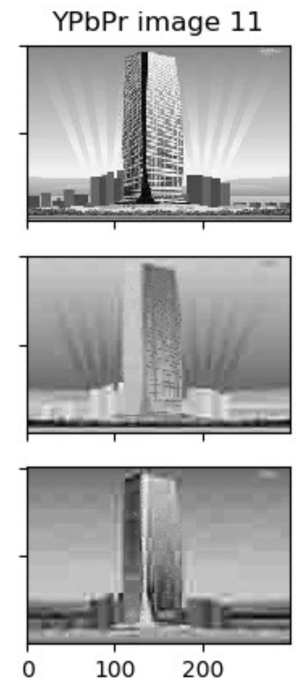
# Color features: Chrominance

$$\begin{bmatrix} Y \\ P_B \\ P_R \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.114 \\ -0.168736 & -0.331264 & 0.5 \\ 0.5 & -0.418688 & -0.081312 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

gives $sum(\vec{v}_R) = sum(\vec{v}_B) = 0$. You don't need to memorize those numbers, but you should know that

- Y = weighted average(R,G,B)
- Pr = (1/2) (R – weighted average(G,B))
- Pb = (1/2) (B – weighted average(R,G))



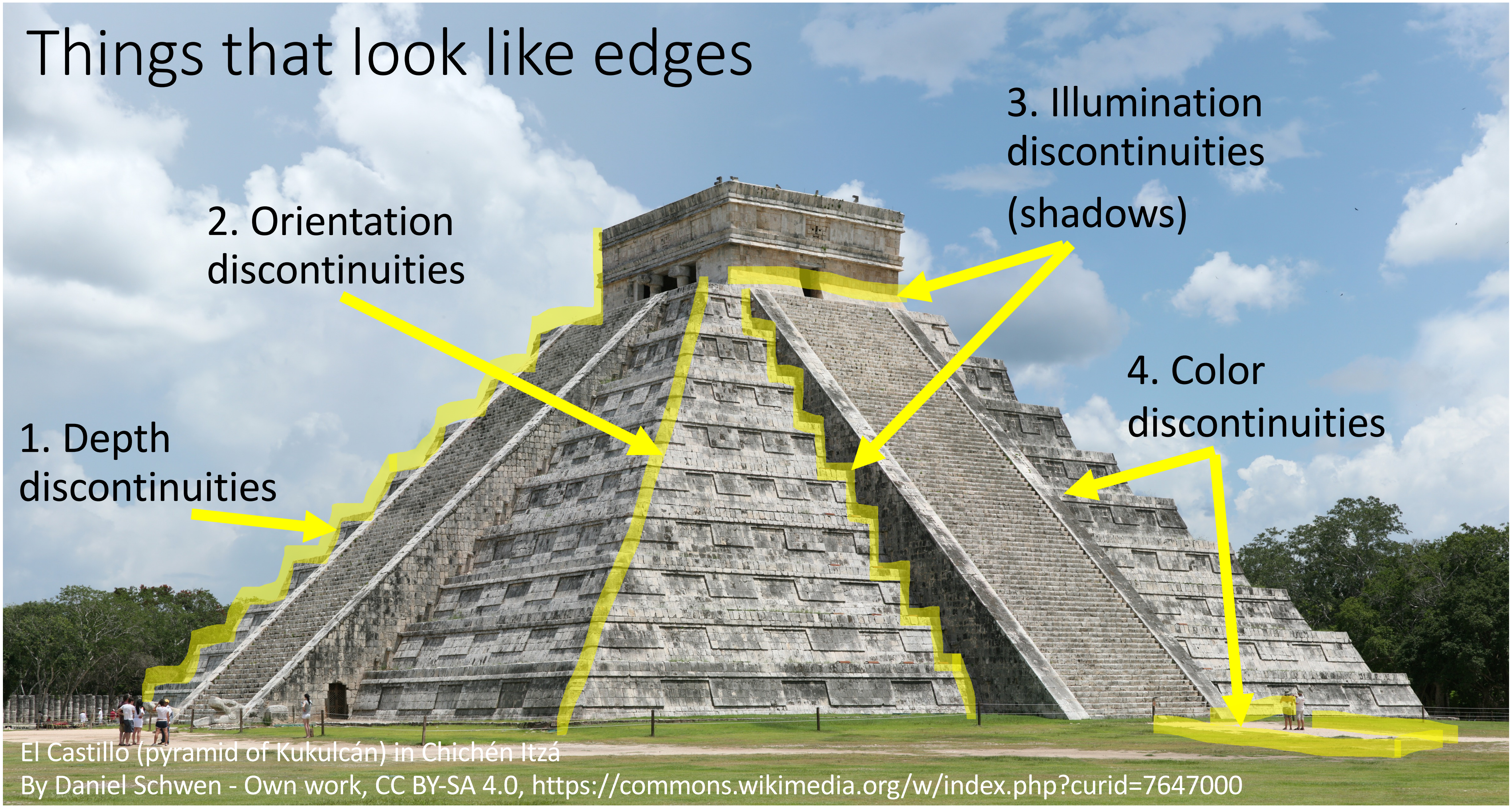



# Color features: Chrominance

- Some images are obviously red! (e.g., fire, or wood)
- Some images are obviously blue! (e.g., water, or sky)
- Average(Pb)-Average(Pr) should be a good feature for distinguishing between, for example, ”fire” versus “water”

# Outline

- Space
  - Pinhole camera equations
  - Vanishing point
- Color
  - Structure of the eye
  - RGB displays
  - Color features: YPrPb
- Edges
  - Things that look like edges
  - Edge detection: the difference-of-Gaussians filter


Things that look like edges
1. Depth discontinuities
2. Orientation discontinuities
3. Illumination discontinuities (shadows)
4. Color discontinuities
El Castillo (pyramid of Kukulcán) in Chichén Itzá
By Daniel Schwen - Own work, CC BY-SA 4.0, https://commons.wikimedia.org/w/index.php?curid=7647000

# Edge detection by subtraction

Subtract neighboring pixels, to compute an "image gradient:"

X gradient:

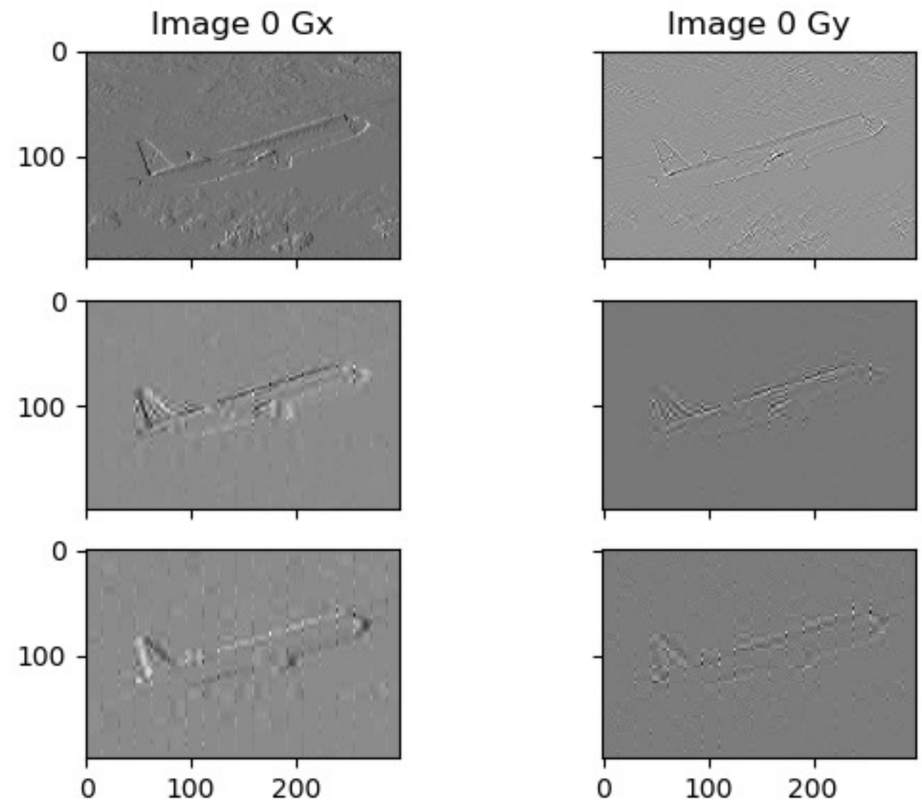
$$G_x(x', y') = \frac{(Y(x'+1, y') - Y(x'-1, y'))}{2} \approx \frac{\partial Y}{\partial x'}$$

Y gradient:

$$G_y(x', y') = \frac{(Y(x', y'+1) - Y(x', y'-1))}{2} \approx \frac{\partial Y}{\partial y'}$$

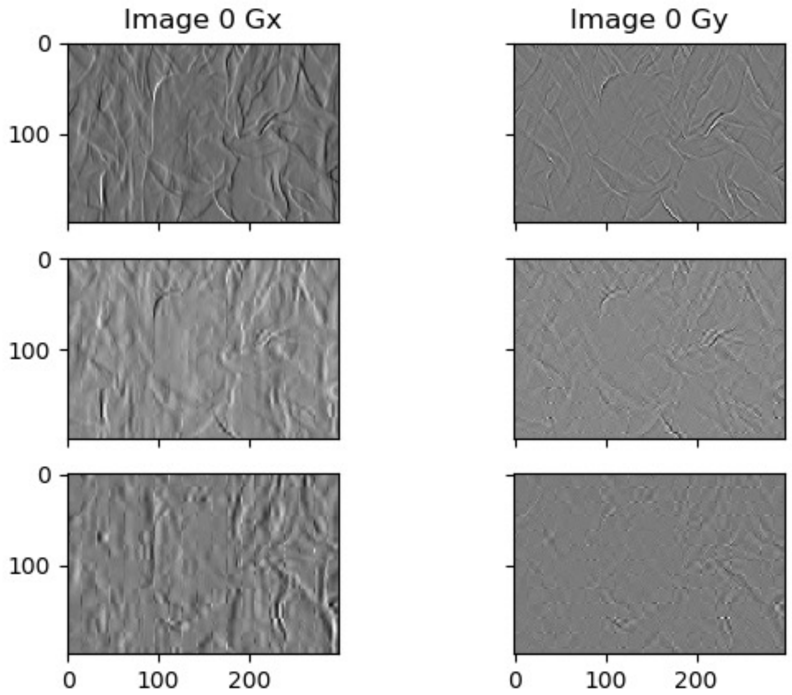
Magnitude of the edge:

$$G(x', y') = \sqrt{G_x^2 + G_y^2} \approx \sqrt{\left(\frac{\partial Y}{\partial x'}\right)^2 + \left(\frac{\partial Y}{\partial y'}\right)^2}$$



# A problem with "edge detection by subtraction"

It tends to exaggerate noise.



# Solving the noise problem

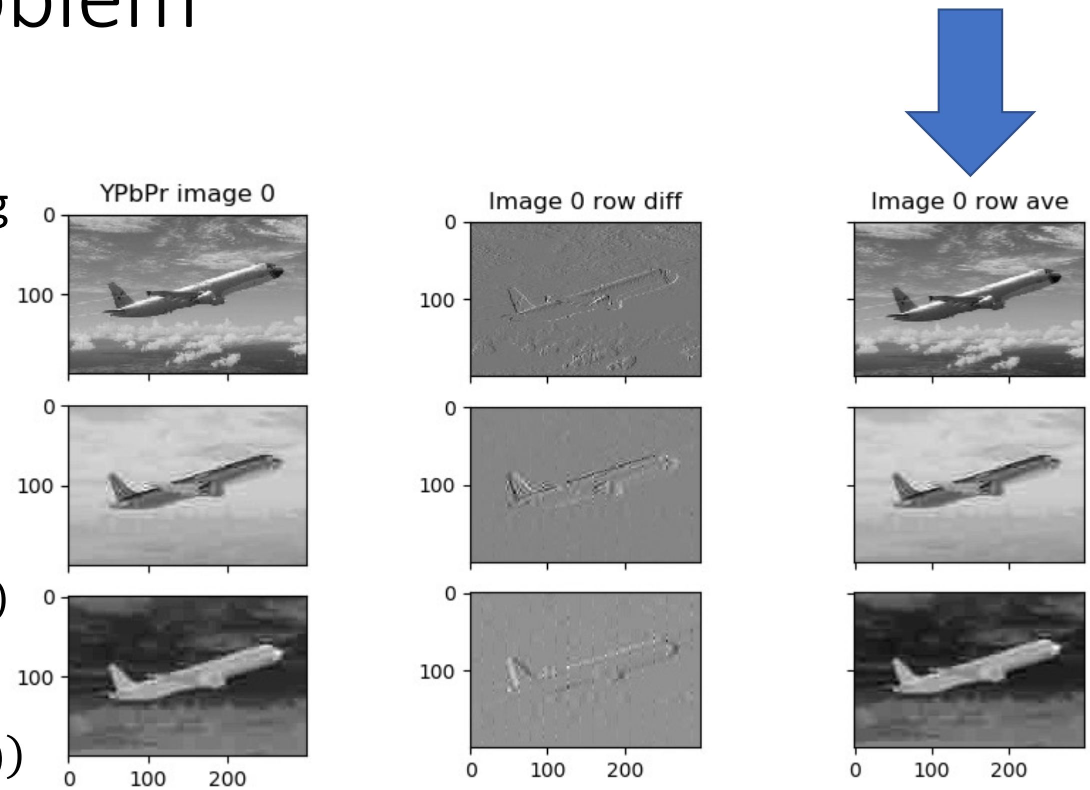
We can solve the noise problem by smoothing the image first, then taking the difference.

Smoothing is done by taking a local average, e.g.,

$$S(x',y') = \sum_{m=-5}^{5}\sum_{n=-5}^{5}\left(\frac{1}{(11)^2}\right)Y(x'-m,y'-n)$$

$$G_x(x',y') = \frac{(S(x'+1,y') - S(x'-1,y'))}{2}$$

$$G_y(x',y') = \frac{(S(x',y'+1) - S(x',y'-1))}{2}$$



# Gaussian blur

Weighted averaging is a little better than unweighted averaging. We just need to make sure that the weights add up to one. For example:

$$S(x',y') = \sum_{m=-5}^{5}\sum_{n=-5}^{5} h(m,n)Y(x'-m,y'-n)$$

$$h(m,n) = \frac{1}{2\pi\sigma^2} e^{-\left(\left(\frac{m}{\sigma}\right)^2+\left(\frac{n}{\sigma}\right)^2\right)}$$

$$G_x(x',y') = \frac{(S(x'+1,y') - S(x'-1,y'))}{2}$$

$$G_y(x',y') = \frac{(S(x',y'+1) - S(x',y'-1))}{2}$$

The Gaussian blur filter $h(m,n)$ results in different degrees of smoothness of $S(x',y')$, depending on how we set the hyperparameter $\sigma$ (the StDev):

# Gaussian blur

Weighted averaging is a little better than unweighted averaging. We just need to make sure that the weights add up to one. For example:

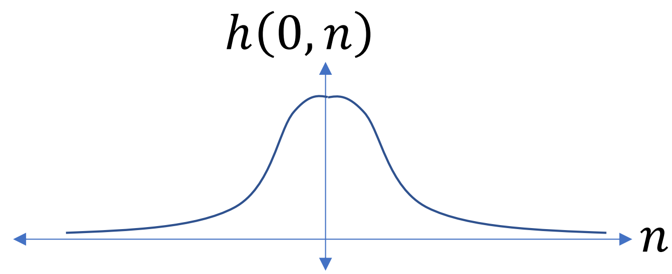
$$S(x',y') = \sum_{m=-5}^{5}\sum_{n=-5}^{5} h(m,n)Y(x'-m,y'-n)$$

$$h(m,n) = \frac{1}{2\pi\sigma^2} e^{-\left(\left(\frac{m}{\sigma}\right)^2+\left(\frac{n}{\sigma}\right)^2\right)}$$

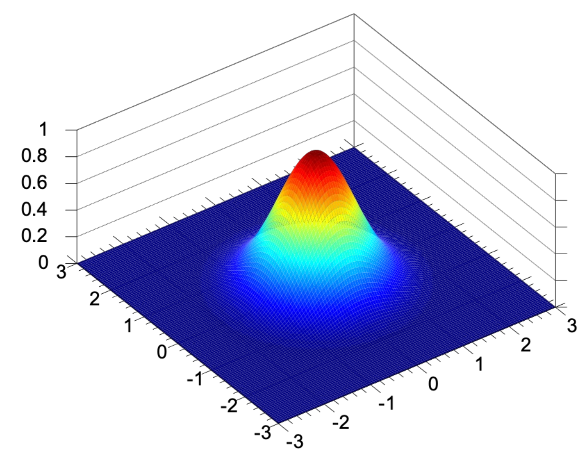
$$G_x(x',y') = \frac{(S(x'+1,y') - S(x'-1,y'))}{2}$$

$$G_y(x',y') = \frac{(S(x',y'+1) - S(x',y'-1))}{2}$$

The Gaussian blur filter $h(m,n)$ looks kind of like this (plotted as a function of just $n$, for the coordinate $m = 0$):




Plotted in 2D, it looks kind of like this:



# Difference of Gaussians

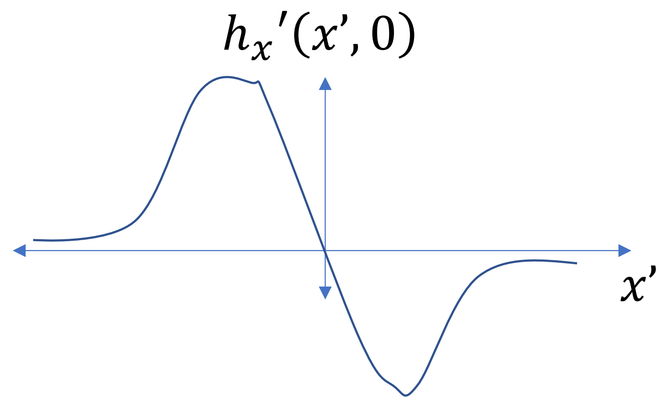
A "difference-of-Gaussians" filter is created by subtracting two Gaussian-blur filters, like this:

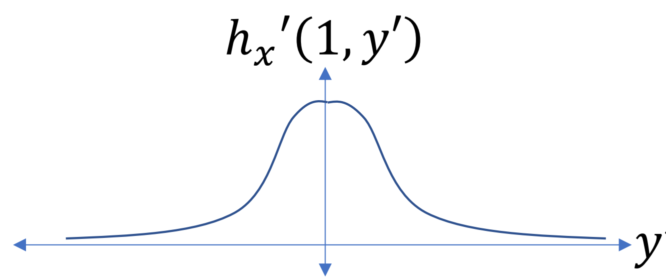
$$h(m,n) = \frac{1}{2\pi\sigma^2} e^{-\left(\left(\frac{m}{\sigma}\right)^2+\left(\frac{n}{\sigma}\right)^2\right)}$$

$$h_x{}'(x',y') = \frac{(h(x'+1,y') - h(x'-1,y'))}{2}$$

$$h_y{}'(x',y') = \frac{(h(x',y'+1) - h(x',y'-1))}{2}$$

A "difference-of-Gaussians" filter looks kind of like this:



# Difference of Gaussians

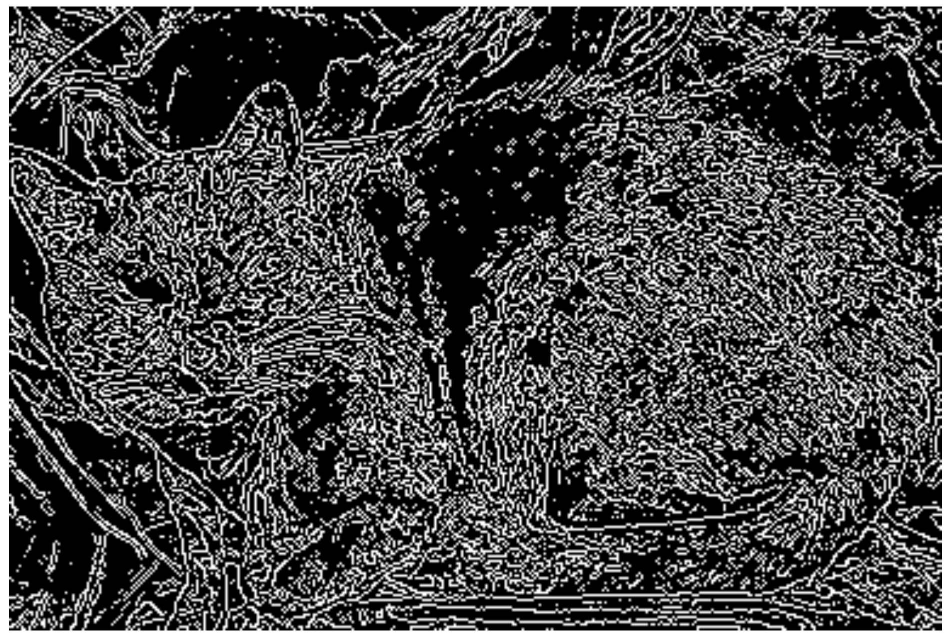
If we pre-compute the difference-of-Gaussians filters, then we can combine the weighted-average and the subtraction into just one operation, to save computation:

$$G_x(x',y') = \sum_{m=-5}^{5}\sum_{n=-5}^{5} h_x{}'(m,n) Y(x'-m, y'-n)$$

$$G_y(x',y') = \sum_{m=-5}^{5}\sum_{n=-5}^{5} h_y{}'(m,n) Y(x'-m, y'-n)$$

… so we never need to compute the smoothed image, $S(x',y')$, at all. We just go directly from the luminance, $Y(x'-m, y'-n)$, to the edge detection.

The difference-of-Gaussians filters, $h_x{}'(m,n)$ and $h_y{}'(m,n)$, detect more or less edges, depending on how we set the hyperparameter $\sigma$. Here is $\sqrt{G_x^2 + G_y^2}$, thresholded to make it black and white:



By Overremorto - Own work, Public Domain, https://commons.wikimedia.org/w/index.php?curid=10581259

# Conclusions

- Pinhole camera equations tell you the relationship between the position on the image, (x',y'), and the position in the real world, (x,y,z). In particular, they tell you why parallel lines seem to converge at the vanishing point.
- The eye has two types of light sensors: cones (see color, only near the center), and rods (see black & white, near the periphery). The real world has all colors, but we can fool the eye by stacking up three images (R, G, and B). The YPrPB color space separates luminance (Y = average(R,G,B)) from chrominance (Pr=R-average(G,B), Pb=B-average(R,G)).
- Edges are caused by discontinuities of depth, orientation, illumination, and color. It's useful to detect where such things happen! Subtracting pixels is noisy, so use a difference-of-Gaussians filter instead.