Figure 1: Crazy eye. TODO: attribution.

1 Computer Vision Introduction and Fundamentals

Attribution: much of the text below is inspired by slides widely shared, re-purposed, and re-mixed in the computer vision teaching community. We mention some of these in the preface, but the list is probably not complete. Yet we are grateful to all who contributed to this mindshare.

Motivation

To act sensibly in the world, robots need to infer knowledge about the world from their sensors. One of the most affordable *and* richest sensors are **cameras**. They are actually amazing devices, having the ability to measure precise angles in space. Unfortunately, understanding camera images is not easy: despite the apparent ease by which we (humans) seem to understand what we see, replicating this in a computer has been fiendishly hard. Since the sixties, when "solve computer vision" was famously assigned as an undergraduate summer project, researchers have tried to tackle this problem. However, since 2012 the emergence of **deep learning** has led to incredible progress. Perception for robotics has been following closely behind, and we live in an exciting age where long-standing problems in robot vision are being conquered.

1.1 What is Computer Vision?

Computer Vision (CV) is in some sense the inverse of Computer Graphics. Graphics pipelines take 3D models, where they are in space, and their material properties, and then produce pleasing images. This should be very familiar to the gamers among you, and graphics processing units (GPUs) excel at those computations. But Computer Vision goes the other way: it takes images and tries to recover the models, their geometry, and their properties. This is very hard, as in the projection information about the 3D world has been destroyed and "mushed together" from 3D into 2D. As an

Figure 2: Outdoor scene about which we could ask a variety of questions. TODO: attribution.

aside, a third field of interest is *Computational Photography,* which is all about transforming images to images.

As a discipline, CV interacts with a lot of other fields. There are technical sub-disciplines of CV that include Image Processing, Geometric Reasoning, Image and Object Recognition, and Deep Learning as applied to CV. Then there some fundamental fields that help CV researchers in solving these sub-problems, such as optics, geometry, computational photography, statistics, and machine learning. And CV finds applications in Robotics, HCI, medical fields, neuroscience, etc...

To introduce the problem a bit more, consider the image of an outdoor scene in Figure 2, about which we could ask a variety of questions. These questions might include, bur are not limited to:

1. What kind of scene is this?
2. Where are the cars in this image?
3. How far is the building?
4. When was this image taken?
5. Where was this image taken?

In other words, we would like to develop algorithms that can take the raw numerical representation of the image (of which you see a rendered representation in the figure) and make sense of that data, answering all or some of the questions above. Note that in this image the date is overlaid on the image in the bottom right, so these algorithms should include the ability to read! Some of

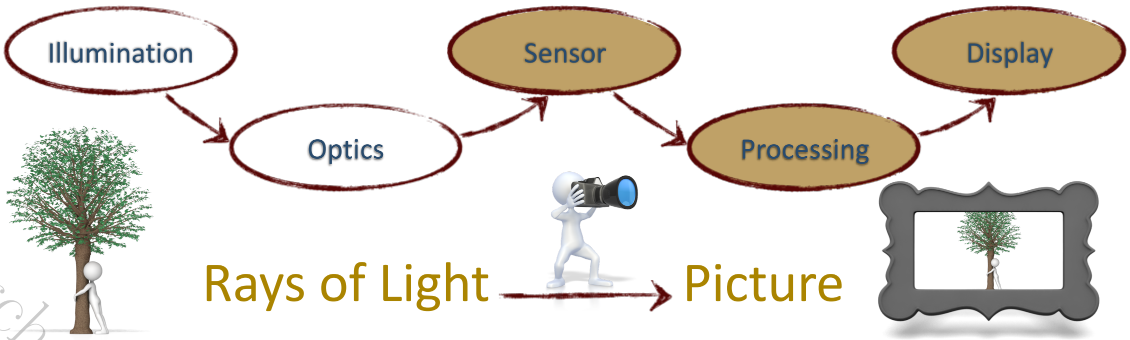

Figure 3: The image acquisition pipeline. Image by Irfan Essa.

the other questions are much harder to answer, however. Also, images are only one type of visual representation that CV considers: others are video, stereo imagery (including stereo video), multi-view imagery in general, multispectral imagery, and even information coming from depth sensors - especially in robot vision.

Yet, despite our own brains seeming to solve this task with *apparent* ease creating computer vision algorithms that work as expected is *really* hard. Introspection is a dangerous activity for computer vision researchers, because we have very little understanding about how our brain solves this problem. It has been said numerous time that "vision [in animals] is an amazing feat of natural intelligence"[1]: neural circuitry devoted to the task of vision in humans takes up a large fraction (approximately a third) of the human brain, and even more in other species. Interestingly, cuttlefish and octopuses also devote large fractions of their brains to the display of visual information. In either case, the large amount of neural machinery devoted to this seems to be roughly proportional to the amount of visual information that is sensed (or displayed) and which has to be digested into actionable information for the animal. And, while we have some sense of the computations that go in the eye and on the information right after the sensing stage, the function of higher-level brain structures is still poorly understood.

While scientific inquiry human and animal vision has al long history, the field of *computer* vision itself started around the sixties, including the venerable summer project assigned by Minsky at MIT. Initially it was very geometry driven, partly because computational resources where scarce. A large amount of progress was made in the nineties and 2000s with the influx of ideas from machine learning, which started to leverage data to do better at some tasks. This took off in a big way in the 2010s with the advent of deep learning, which leveraged the computational graphics pipelines (GPUs!) to really take advantage of large-scale data such as internet image collections.

1.2 Applications of CV

Computer vision algorithms have found wide application and have in some cases transformed entire industries.

TBD: please refer to the slides.

1.3 Images as 2D arrays

[1]This is quoted in so many CV lecture slides it is hard to track down who to attribute it to.

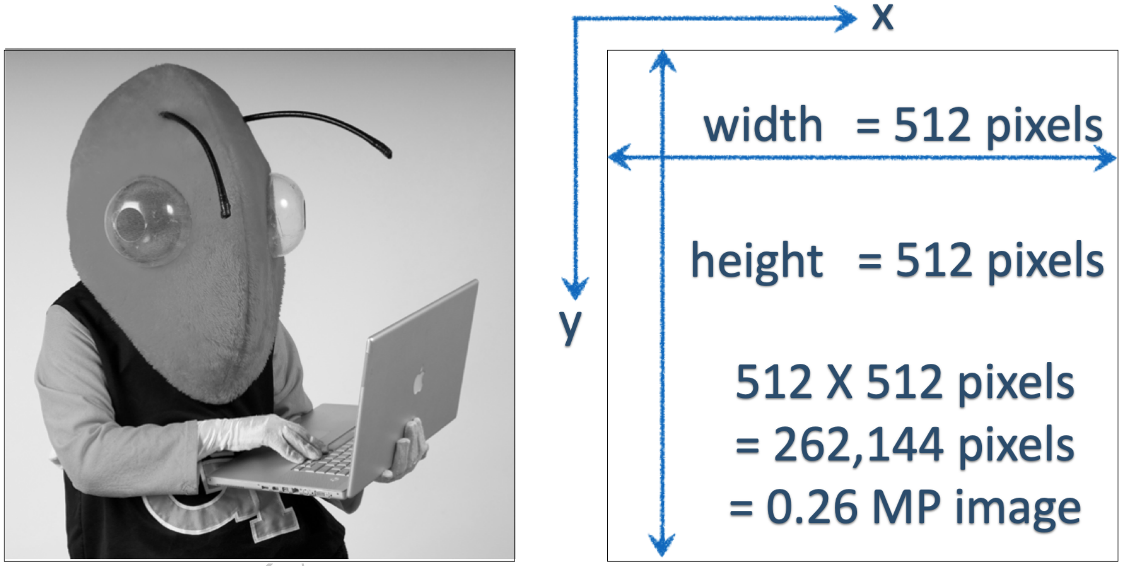

Figure 4: Black and white image, stored in a 512 by 512 array of bytes, making for a quarter megapixel image. Image by Irfan Essa.

The image acquisition pipeline for a typical camera is illustrated in Figure 3. The light reflected of the scene goes through the camera's optics after which is collected by the sensor. After further processing, which typically includes considerable "improvement" by the camera manufacturer, the image is stored in a binary format. The processing these days can involve considerable algorithmic improvements and, on modern smartphones, it could be that the final stored image is derived from a "burst" of images taken by the sensor. Finally, the image can be displayed on a screen or projected, which can involve more tweaking of the actual image.

Grayscale digital images are stored as 2D array of **values**. An example is shown in Figure 4. This is a 512 by 512 **resolution** image, making for a quarter megapixel image. The resolution is typically given as $w \times h$, i.e. image width by image height. People use the term "megapixel" or MP as a shorthand for a million pixels. Example sensor resolutions are 640×480 (VGA, 0.3MP), 1280×720 (0.9MP), or 3840×2160 (8.3MP). The **aspect ratio** is the ratio of width to height, e.g., VGA is 4:3, and another important aspect ratio is 16:9.

Each of the elements of the array is a picture element or **pixel** which stores an intensity value, typically between 0 and 255, i.e., one byte or 8 bits. Sometimes, in high-end cameras or when images are stored in RAW format, 10, 12, or 16 bits are used as well. We speak of the **bit-depth**. We say that that the value 0 corresponds to black, whereas 255 corresponds to pure white. In reality, however, the exposure time and aperture of the camera determine the dynamic range of light intensities the sensor will pick up. Hence it is more correct to say that the value 0 corresponds to "the light intensity was not registered by the sensor", and the value 255 corresponds to "the sensor was overexposed in this area".

An image can also be visualized as a 2D function, which is a sampled representation of a continuous intensity field $I(x, y)$. We typically use x, y when referring to continuous coordinates and i, j when referring to discrete coordinates. When using continuous coordinates, typically $x$is left to right and y increases downwards. Note that different conventions are used in different APIs, e.g., in MATLAB the discrete i refers to the row and j refers to the column. And OpenGL uses a y-axis

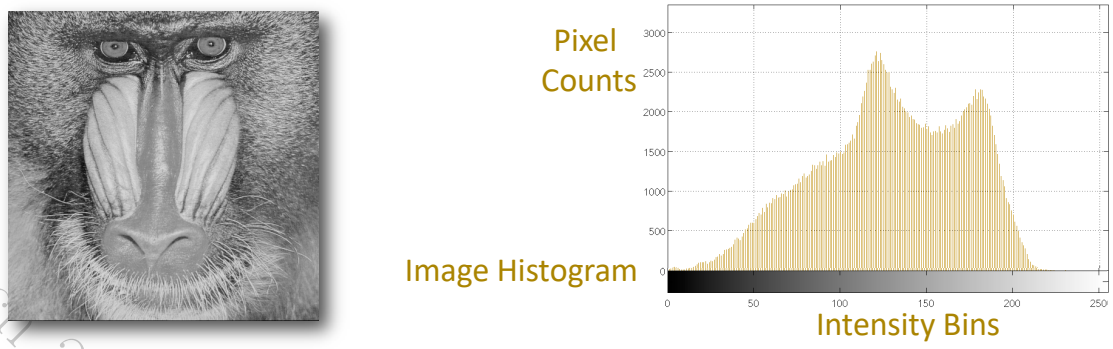

Figure 5: An image histogram is one way to display global image statistics.

Figure 6: Color images are stored as three channels: R(ed), G(reen), and (B)lue.

which points upwards, not downwards. Careful attention is thus needed when reading literature around computer vision and/or coding up image processing algorithms.

One way to analyze images is through image statistics, e.g., the mean or median intensity value. A more fine-grained device is the image histogram, illustrated in Figure 5, which tabulates the number of pixel for a given intensity value. The particular example shows that there are no under- or over-exposed pixels, and there are two peaks, probably corresponding to the bright and more grey areas of the animal pictured.

Color images are stored as three separate arrays corresponding to the red (R), green (G), and blue (B) channel. We speak of an **RGB image**. In (very) expensive cameras this can literally correspond to three separate sensors that receive light filtered through a red, green, and blue filter, respectively. However, in almost all consumer cameras on the market today, and certainly in all smartphone cameras, the color is created by an array of color filters glued on top of the sensor, called a **Bayer pattern**. Hence, the color signal is captured at a lower resolution than the actual sensor resolution, and the full resolution color image is estimated from that signal in a process called **debayering** or **demosaicing**.

1.4 Basic Image Processing

Image processing refers to algorithms that work on raw pixel data or information that is derived from it. We will not replicate an entire image processing course here, but will discuss two broad categories here, which are

1. Per-pixel image processing
2. Area-based image processing

The latter category will be discussed in the next section. The first category can be expressed in terms of a simple functional relationship, e.g., **contrast** enhancement simply multiplies the pixel values with a constant a,

$$g(x) = af(x)$$

where x refers to a pixel location, $f(x)$ is the source image, and $g(x)$ is the target image. A **brightness** adjustment, on the other hand, is just adding a constant:

$$g(x) = f(x) + b$$

Hence, you can see that the two "master knobs" of any image editing program, brightness and contrast, simply implement a linear transformation of the data:

$$g(x) = af(x) + b$$

One important caveat applies: because the image values are clipped at 0 and 255, respectively, this is not *truly* a linear transformation. However, some image editing software does all the math in signed floating point, and only does the clipping at the end, when saving a file, which gives one temporarily relief of this non-linear clipping. In fact, it is common practice to rescale all intensity values to the domain $[0, 1]$ upon reading, and work in a bit-depth independent manner throughout the entire workflow.

Per-pixel image processing can be non-linear as well. For example, **gamma correction** is an operation frequently used to adapt images for human consumption and/or correct for display non-linearities. It has a single parameter γ, hence its name:

$$g(x) = f(x)^{\gamma}$$

Another example of a nonlinear operation is **histogram equalization**, which computes and then flattens the histogram of pixel values in an image.

Finally, **image arithmetic**, e.g., adding, subtracting, multiplying and dividing images are all per-pixel operations. Again careful attention is required to the clipping of the intensity values. Image arithmetic is frequently used in image compositing, where an **alpha matte** encodes the transparency of objects, and compositing of a foreground F on a background B can be done by some clever mixing. An example is shown in Figure 7. Note that in these workflows all images are handled in bit-depth independent floating point intensity, ranging from 0.0 to 1.0, as discussed above. In that context, the alpha matte also ranges from 0.0 (completely transparent) to 1.0 (completely opaque), and if you look carefully you can see that at the boundary of the foreground object, the values are actually somewhere between the two, indicating "mixed pixels".

Figure 7: Image compositing, frequently used in the special effects industry. Top to bottom, left to right we have the alpha matte a, the pre-multiplied foreground image aF, the pre-multiplied background image $(1-a)B$, and the final composite $aF + (1-a)B$.

1.5 Image Filtering

TBD. Refer to the slides.

Summary

We briefly summarize what we learned in this section:

1. Computer Vision defined
2. Applications of CV are plentiful! (TBD)
3. Images are 2D arrays of pixel values
4. Basic image processing: contrast, intensity, histogram eq., arithmetic
5. Image filtering: convolution (linear) and non-linear (median) (TBD)