Computer Vision

A simple two-stage model of computer vision:

- **Image processing**
- **Scene analysis**

**Prepare image for scene analysis**
**Build an iconic model of the world**

Feedback (tuning)

---

Computer Vision

The **image processing** stage prepares the input image for the subsequent scene analysis. Usually, image processing results in one or more new images that contain specific information on relevant features of the input image. The information in the output images is arranged in the same way as in the input image. For example, in the upper left corner in the output images we find information about the upper left corner in the input image.

---

Computer Vision

The **scene analysis** stage interprets the results from the image processing stage. Its output completely depends on the problem that the computer vision system is supposed to solve. For example, it could be the number of bacteria in a microscopic image, or the identity of a person whose retinal scan was input to the system. In the following lectures we will focus on the lower-level, i.e., image processing techniques. Later we will discuss a variety of scene analysis methods and algorithms.

---

Digitizing Visual Scenes

Obviously, any 3D point \((x, y, z)\) is mapped onto a 2D point \((x', y')\) by the following equations:

\[
x' = f \cdot x / z \\
y' = f \cdot y / z
\]

---

Actual Camera Geometry

How can we turn a visual scene into something that can be algorithmically processed? Usually, we map the visual scene onto a two-dimensional array of intensities. In the first step, we have to project the scene onto a plane. This projection can be most easily understood by imagining a transparent plane between the observer (camera) and the visual scene. The intensities from the scene are projected onto the plane by moving them along a straight line from their initial position to the observer. The result will be a two-dimensional projection of the three-dimensional scene as it is seen by the observer.
Digitizing Visual Scenes

In this course, we will mostly restrict our concept of images to **grayscale**. Grayscale values usually have a resolution of 8 bits (256 different values), in medical applications sometimes 12 bits (4096 values), or in binary images only 1 bit (2 values).

We simply choose the available gray level whose intensity is closest to the gray value color we want to convert.

Digitizing Visual Scenes

With regard to spatial resolution, we will map the intensity in our image onto a two-dimensional finite array:

\[
\begin{array}{cccc}
[0, 0] & [0, 1] & [0, 2] & [0, 3] \\
[1, 0] & [1, 1] & [1, 2] & [1, 3] \\
\end{array}
\]

So the result of our digitization is a two-dimensional array of discrete intensity values.

Notice that in such a digitized image F[i, j]

- the **first coordinate** i indicates the row of a pixel, starting with 0,
- the **second coordinate** j indicates the column of a pixel, starting with 0.

In an m×n pixel array, the relationship between image (with origin in ints center) and pixel coordinates is given by the equations

\[
x' = j = \frac{n-1}{2} \\
y' = \left( i - \frac{m-1}{2} \right)
\]
Floyd-Steinberg Dithering
The best dithering results are achieved by compensating for the thresholding error at a pixel x by adjusting the intensity of neighboring pixels. Floyd-Steinberg dithering uses the following matrix for this purpose:

\[
\begin{bmatrix}
  x & 7 & 16 \\
  3 & 5 & 1 \\
  16 & 16 & 16
\end{bmatrix}
\]

Levels of Computation
As we discussed before, computer vision systems usually operate at various levels of computation. In the following, we will discuss different levels of computation as they occur during the image processing and scene analysis stages.

We will formalize this concept by means of an operation \( O \) that receives a set of pixels and returns a single intensity value that can be used to determine the value of a pixel in the output image.

We will look at operations at the point level, local level, global level, and object level mapping an input image \( F_A[i, j] \) to an output image \( F_B[i, j] \).

Point Level

Operation type:

\( F_B[i, j] = O_{\text{point}}(F_A[i, j]) \)

This means that the intensity of each pixel in the output image only depends on the intensity of the corresponding pixel in the input image.

Examples for this kind of operation are inversion (creating a negative image), thresholding, darkening, increasing brightness etc.

Local Level

Operation type:

\( F_B[i, j] = O_{\text{local}}(F_A[k, l]; [k, l] \in N[i, j]) \)

Where \( N[i, j] \) is a neighborhood around the position \([i, j]\). For example, it could be defined as:

\( N[i, j] = \{(u, v) \mid |i - u| < 3 \land |j - v| < 3\} \)

Then the neighborhood would include all pixels within a 5\( \times \)5 square centered at \([i, j]\).

So the intensity of each pixel in the output image depends on the intensity of pixels in the neighborhood of the corresponding position in the input image.

Examples: Blurring, sharpening.
Global Level

Operation type:
\[ F_{\text{global}}[i, j] = O_{\text{global}}[F_A[k, l]; 0 \leq k < m, 0 \leq l < n] \]
for an \( m \times n \) image.

So the intensity of each pixel in the output image may depend on the intensity of \textbf{any pixel} in the input image.

Examples: histogram modification, rotating the image

Object Level

The goal of computer vision algorithms usually is to determine properties of an image with regard to specific objects shown in it.

To do this, operations must be performed at the \textbf{object level}, that is, include all pixels that belong to a particular object.

Problem: We must use all points that belong to an object to determine its properties, but we need some of these properties to determine the pixels that belong to the object.

While this seems effortless in biological systems, we will later see that complex algorithms are required to solve this problem in an artificial system.