Robotics
Image sensors

Tullio Facchinetti
<tullio.facchinetti@unipv.it>

Tuesday 13th December, 2016

http://robot.unipv.it/toolleeo

Image and science

the visual observation is of paramount importance
in all scientific activities

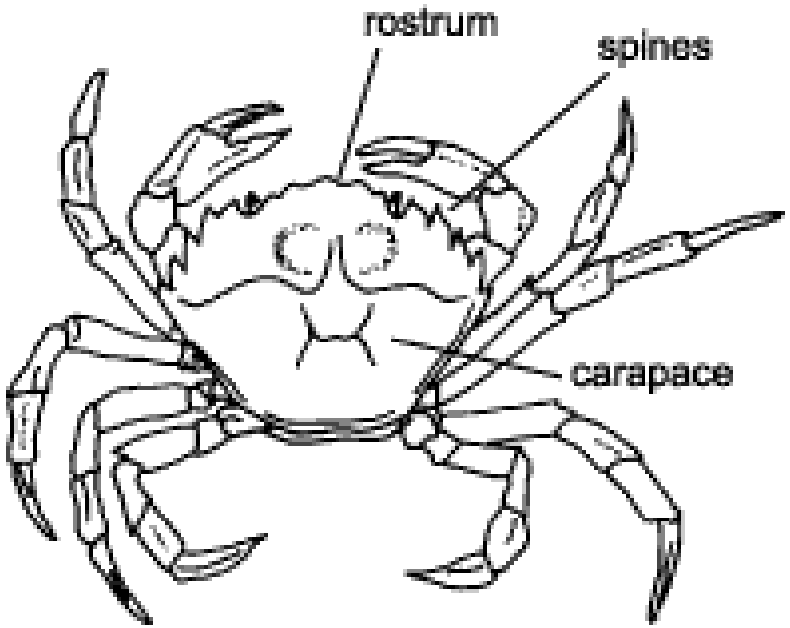
in the past, in the absence of automatic image processing, the only way to represent something was the verbal description or making pictures

there were no automatic tools to qualitatively describe an object, a process or a result

a nice example is made by the biology, which include ecology, zoology, botany, etc.

Describing animals and vegetables

before the advent of image acquisition and processing



KEY FEATURES

- 5 developed spines on both sides of the carapace near the eyes
- Colour varies. Underside & legs may be red/orange or green
- Broad triangular carapace, deeply sculpted on top
- 4th walking leg has no swimming "paddle"
- Can grow to 9 cm carapace width, common size is between 5-7 cm

Image processing

some applications that opened the door to the use of image processing in the scientific domain are

- astronomy: to measure the position of stars
- photogrammetry: to obtain the shape, position and orientation of objects
- particles physics: to discover new particles from the analysis of images acquired during experiments

in few year, the use of image processing became ubiquitous

The image processing nowadays

the factors that contributed to the wide adoption of image processing techniques are:

- powerful and cheap computing system
- established image processing methods and algorithms
- standard multimedia hardware (e.g. GPUs) and software (libraries, e.g. OpenCV) components
- a technology available to every scientist or engineer

> the application of image processing techniques encompasses almost all possible scientific fields

Applications of image processing

typical applications are:

- detection and counting objects
- classification (biometry, surveillance, etc.)
- reconstruction of 3D environments (mapping, etc.)
- human feedback in remote operations (surgery, drone piloting, etc.)
- multimedia
- control of autonomous robots

the image processing is one of the most versatile sensory technique, providing the richest kind of information

Applications of image processing

Applications of image processing

Applications of image processing

Images in computer science

there are some different disciplines in the IT domain that involve the use of images:

- **visual computing**: multimedia processing, often integrating image, audio and video processing
- **computer graphics**: generation of images starting from information regarding the objects in the scene
- **image processing**: extraction of features associated to objects captured in the image

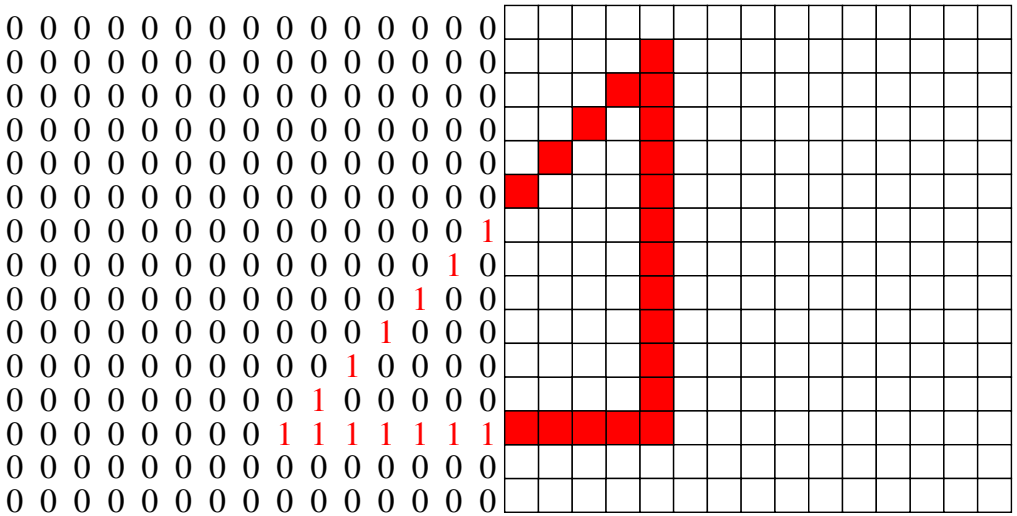
Involved research disciplines

there are several scientific domains that are very important for the image processing:

- optics (lens and other optical effects)
- solid state physics (interaction between photons and materials)
- micro-electronics (design and manufacturing of sensing chips)
- computer architecture (for fast computing)
- algebra (rotations, translations, and other operations)
- graph theory (in image processing)
- numerical analisys (e.g., optimization algorithms)
- etc.

Images and image sensors

- the image is composed by a matrix of colored points, called **pixels** (picture elements)
- each pixel is represented by its coordinate and its color



resolution

the amount of pixels in a given area, often expressed as the number of pixels on the two axis

Image sensors

features of an image sensor:

- different resolutions available, from 5 Mpxl of common cameras to 39+ Mpxl of expensive devices
- frame rates of 500+ frame-per-second (FPS)
- not limited to the visible spectrum (infrared, ultraviolet, X-ray)

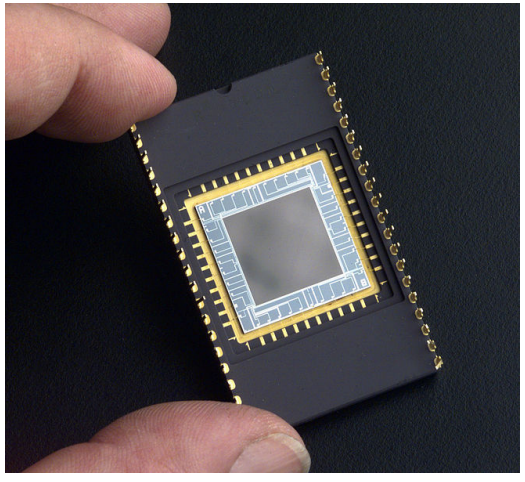
there are two types of sensors available to capture an image:

- CCD – Charged Coupled Device
- CMOS – Complementary Metal Oxid Semiconductor

all sensors are based on a matrix of photo-sensors,
but they have different design technologies
and (therefore) different features

The CCD sensor

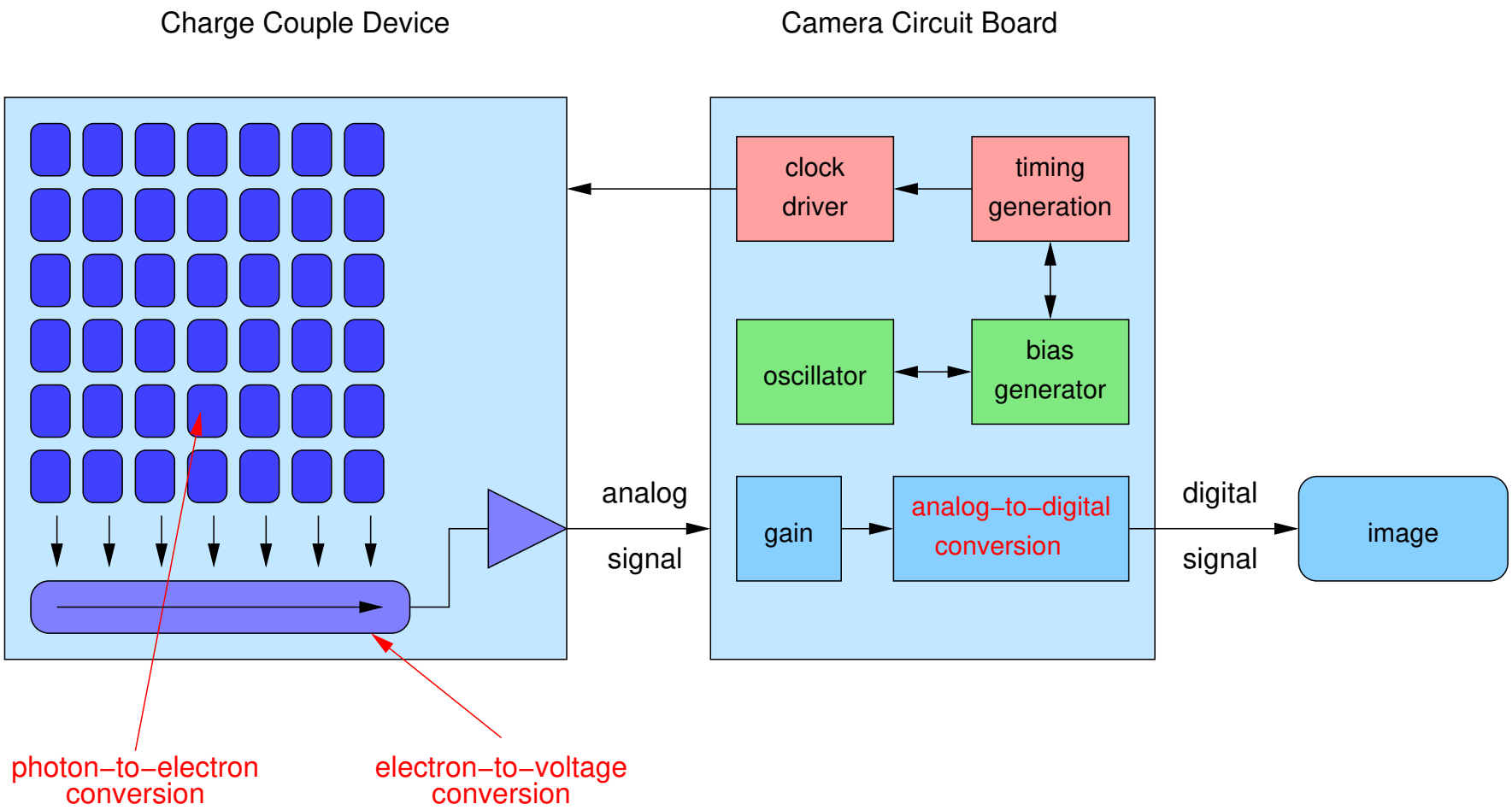
CCD = Charged Coupled Device



source: Wikipedia

in the CCD technology, the output of the transducer is an analog signal, which is processed by a decoupled logic component

The CCD technology



the acquisition requires two steps:

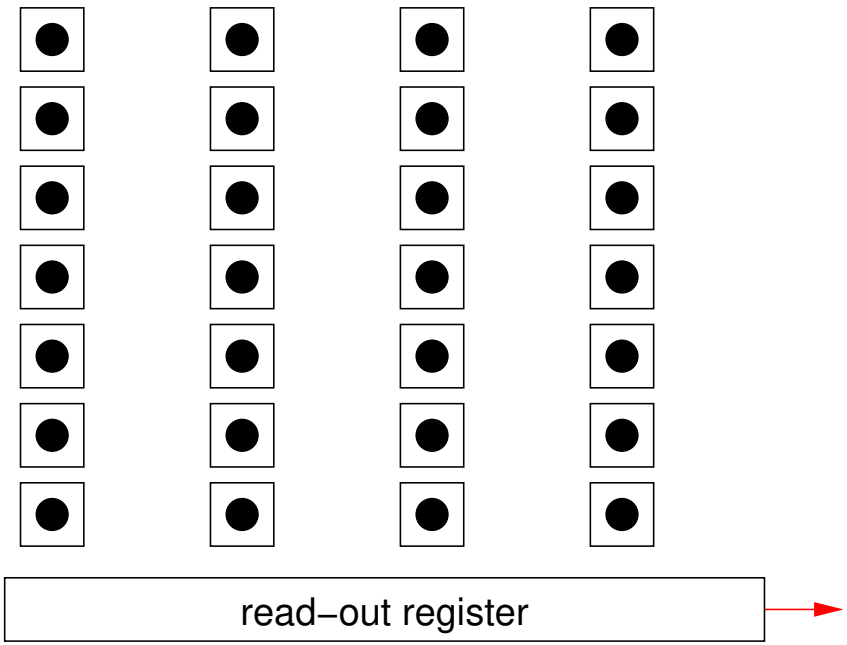
exposure (accumulation) phase:

- each photoactive area collects electric charges due to the absorption of photons
- the charge is proportional to the intensity of the light

readout phase:

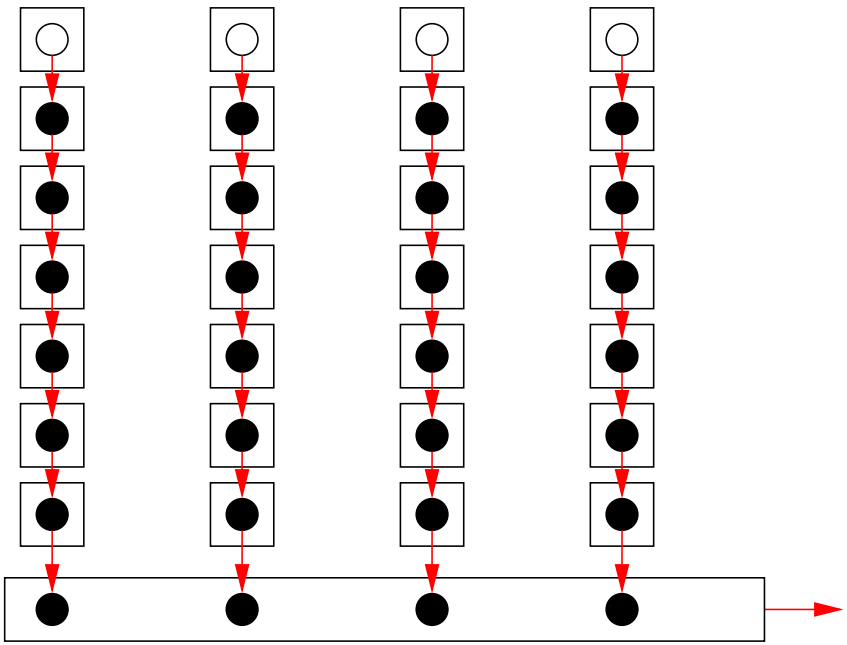
- the charges are sequentially carried out from the component
- the charge is translated into a voltage and properly amplified

Full-Frame Transfer



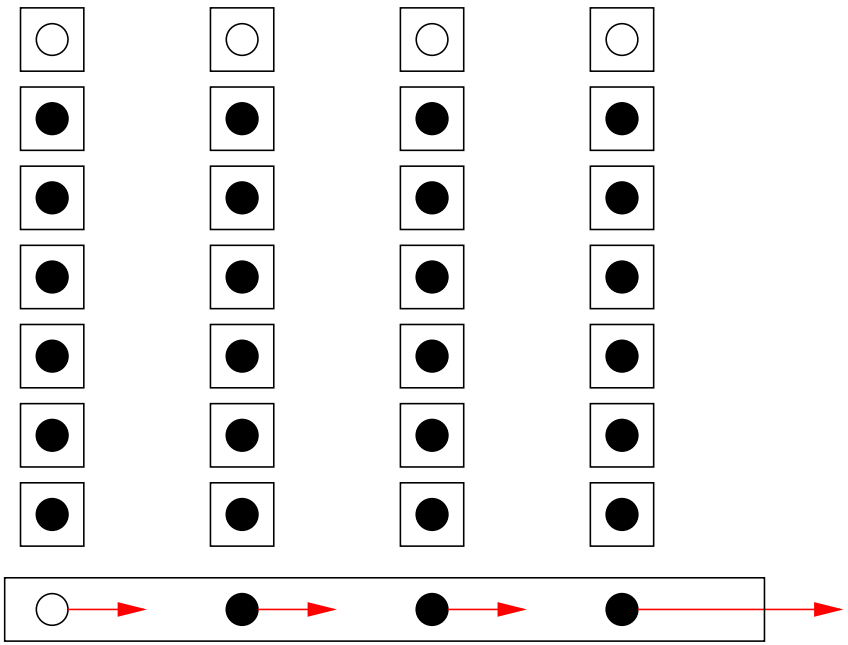
- the sensing area is a matrix of sensing elements

Full-Frame Transfer



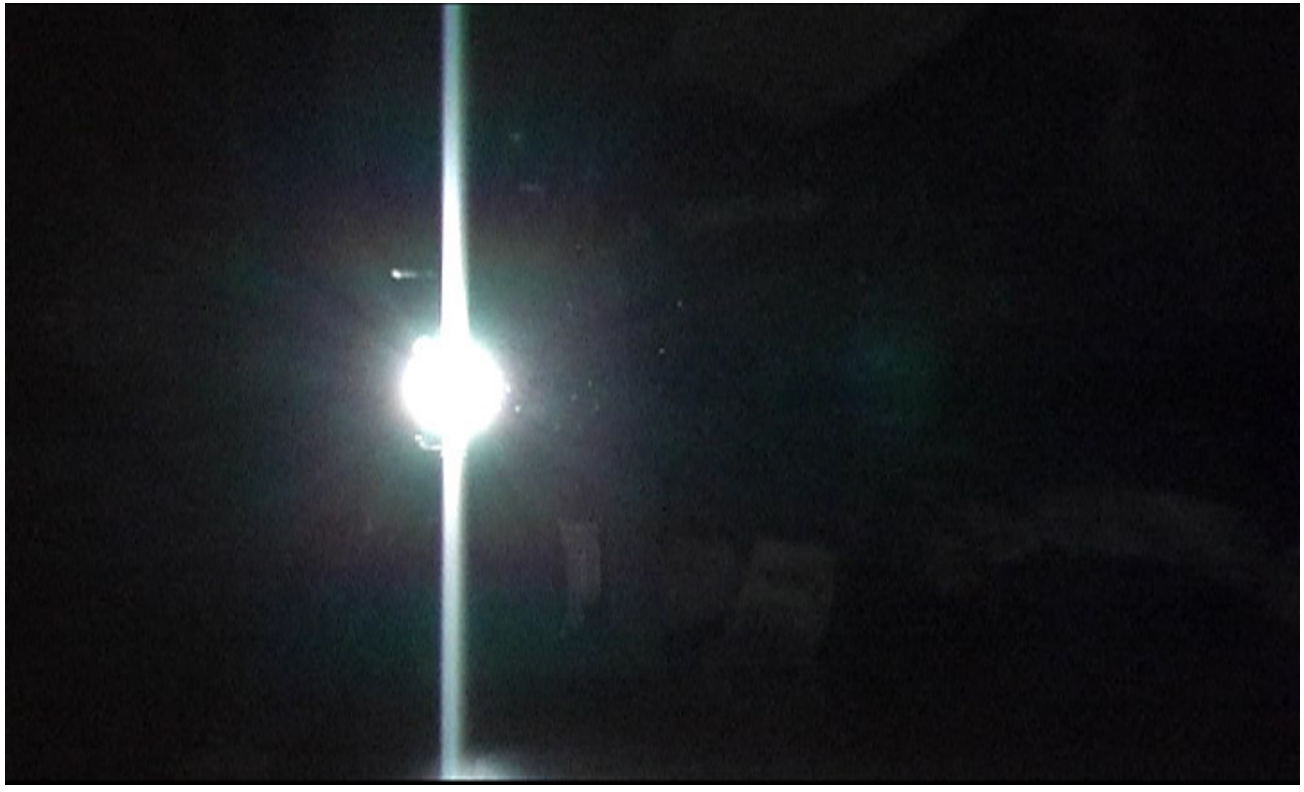
- an horizontal readout shift register receives the charges row-by-row
- during this phase, the sensing elements are still exposed to the light

Full-Frame Transfer



- each readout may take around 65μs per row (strongly technology dependent, though)

Full-Frame Transfer: smearing effect



source: Wikipedia

Full-Frame Transfer: smearing effect

- it is due to the collection of charges during the vertical shift
- to eliminate it, a fast vertical shifting would be needed

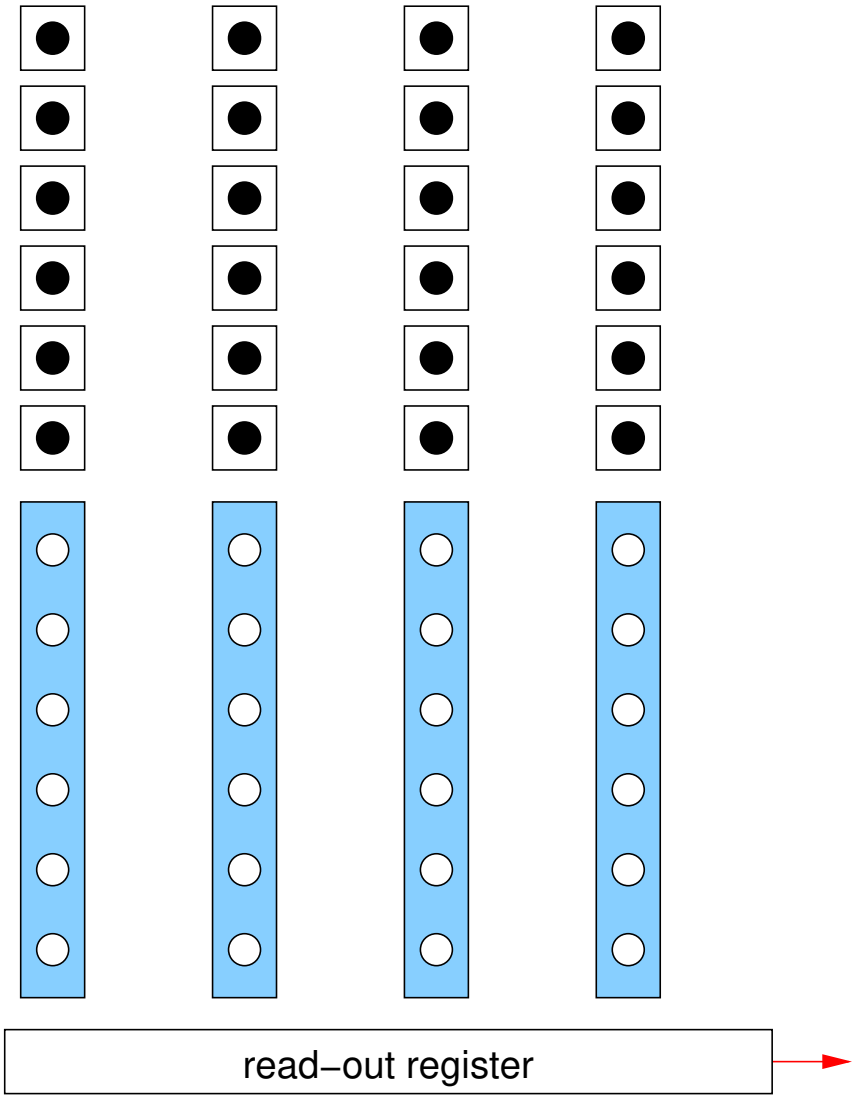
  however...

- a fast vertical shifting would require a faster readout
- and a fast readout implies a faster reading of the amount of charge

  however...

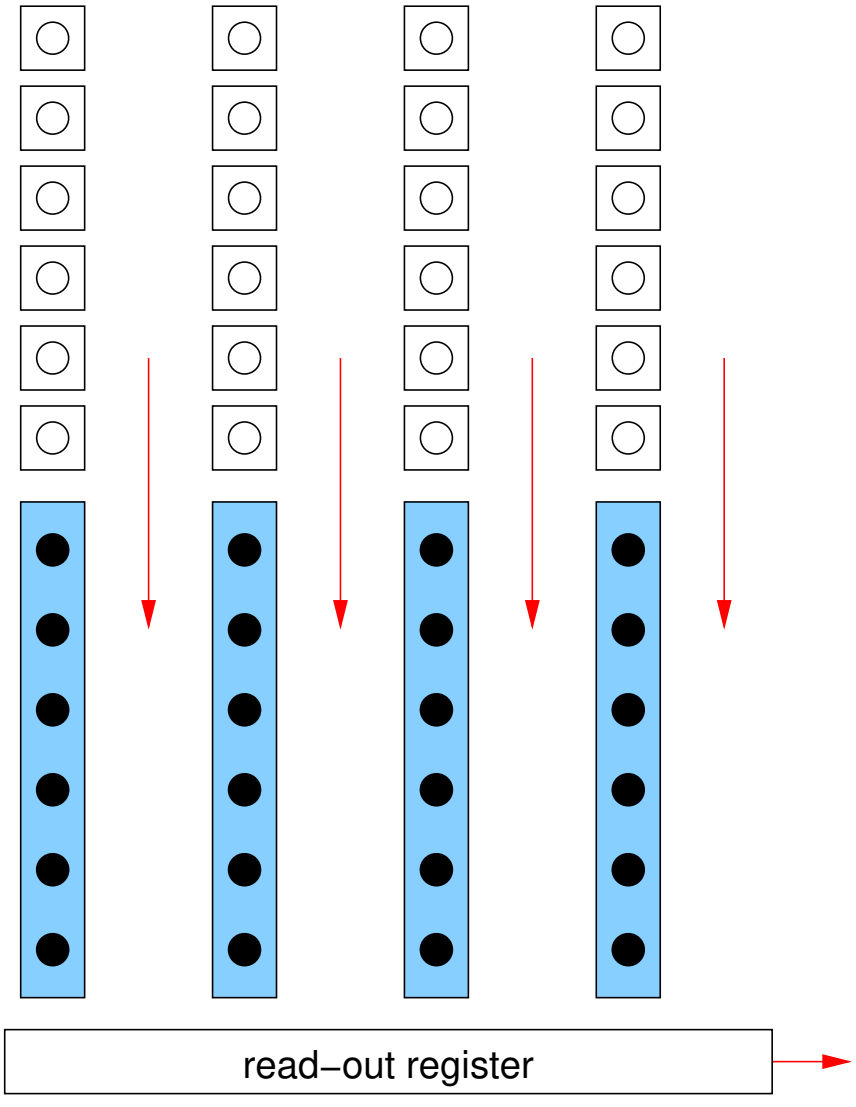
- a short charge collection time leads to higher measurement errors and noise

Frame Transfer



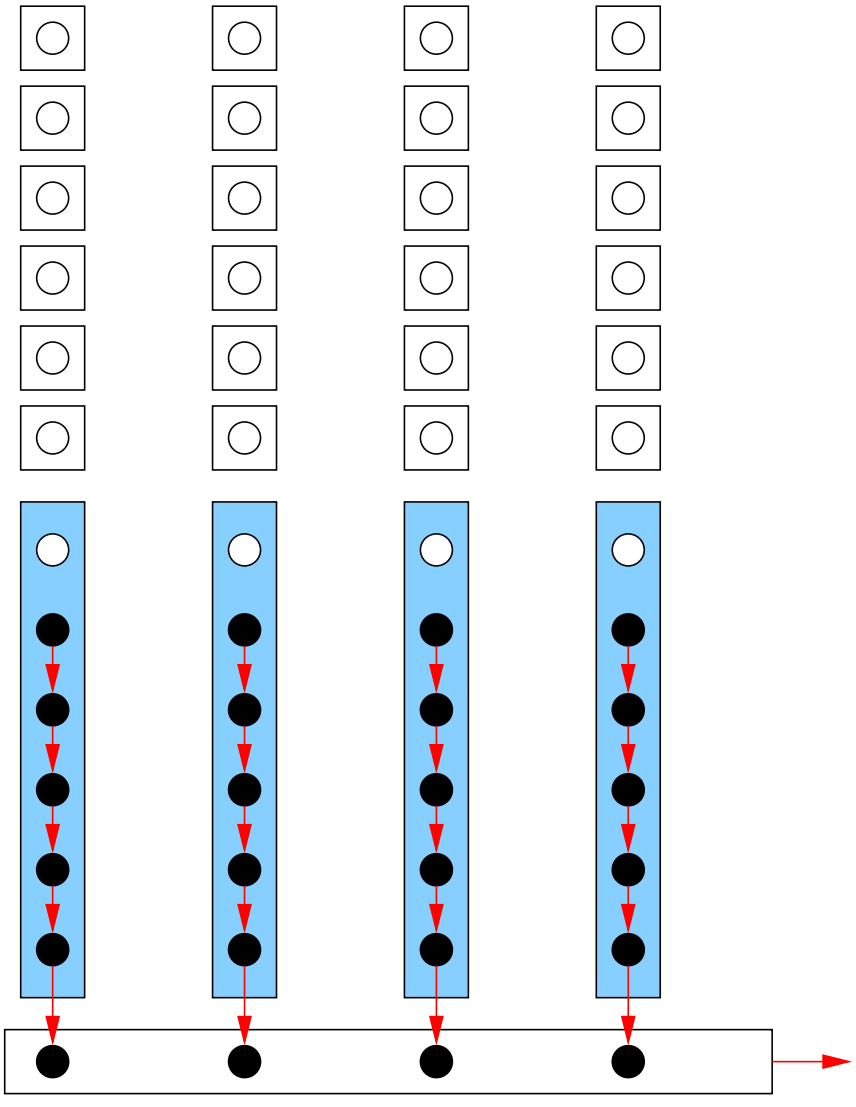
- there is a double area of cells
- the above area is the sensing element
- the lower area is a shielded memory that receives the charge from sensors

Frame Transfer



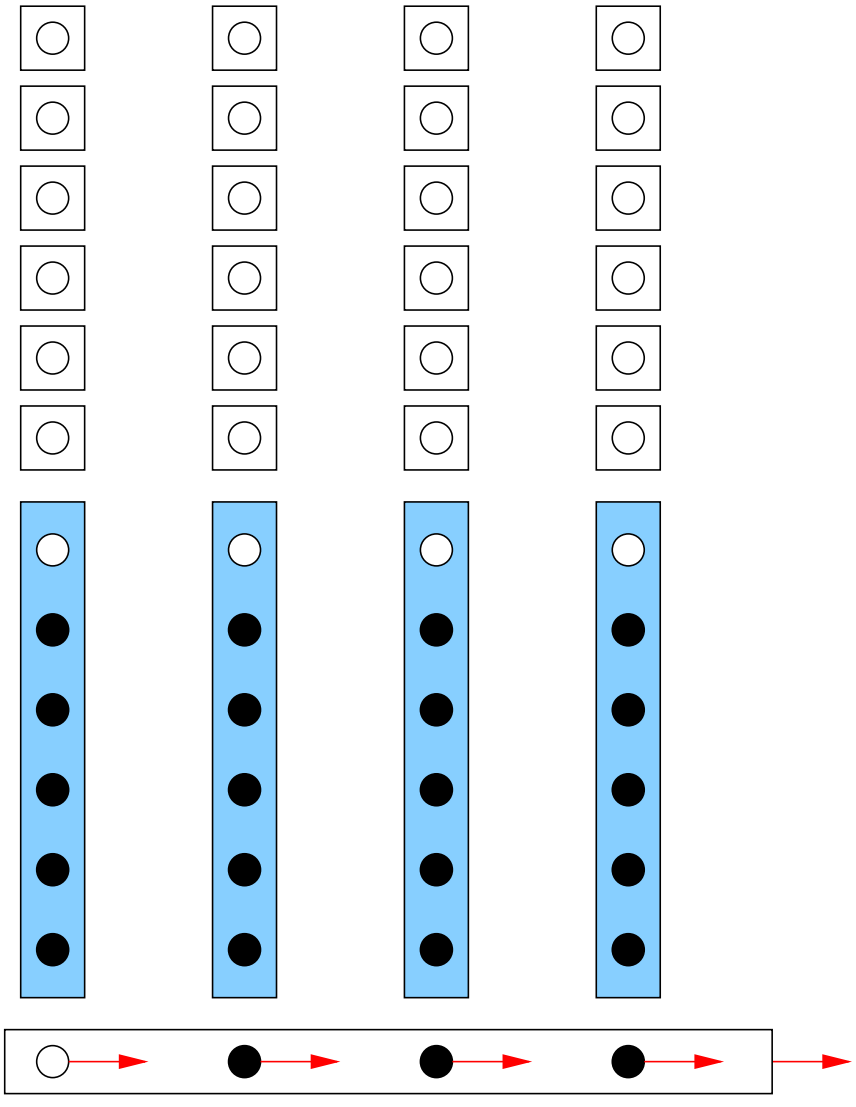
- first, the whole frame is transferred in a dark area, protected by incident light
- this operation takes place every 20ms
- it may require around 500μs

Frame Transfer



- the vertical shifting brings each line of charges into the readout register
- even thought the sensors continue to accumulate charges, these latter are transferred with no interference

Frame Transfer



- the serial readout is done row-by-row
- each readout takes around $65\mu s$ per row

Frame Transfer

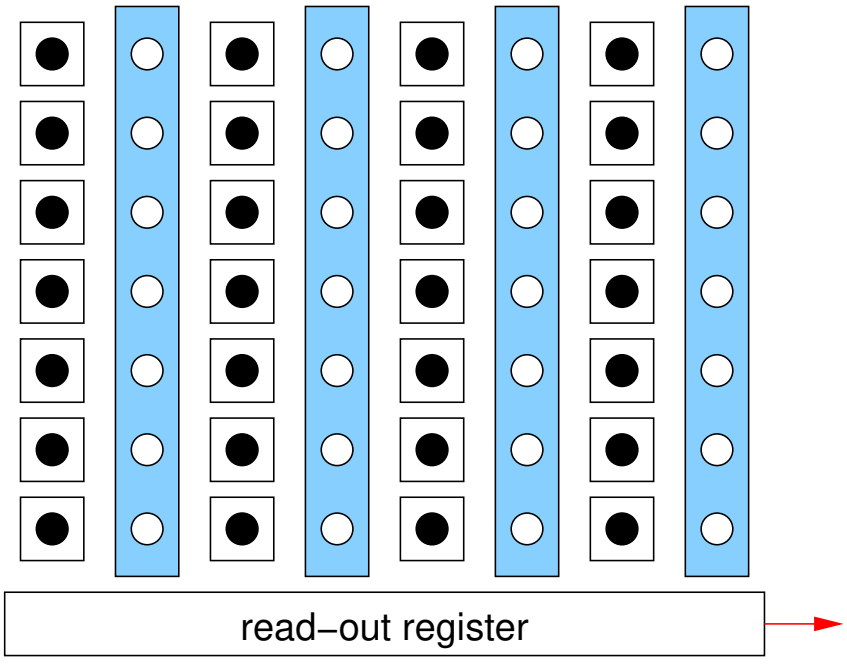
pros

- fill factor (percentage of area available for the sensing elements) close to 100%
- for sufficiently long exposure times, there is no need of electro-mechanical shutter, which is obtained electronically

cons

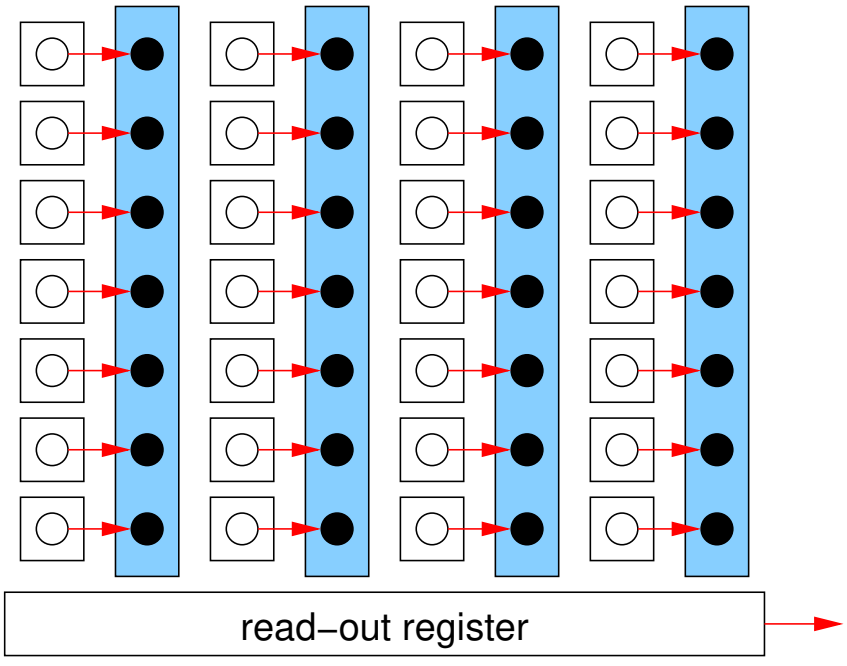
- requires twice of the silicon surface w.r.t. the Full Frame Transfer
- not suitable for very short exposure times

Interline Transfer



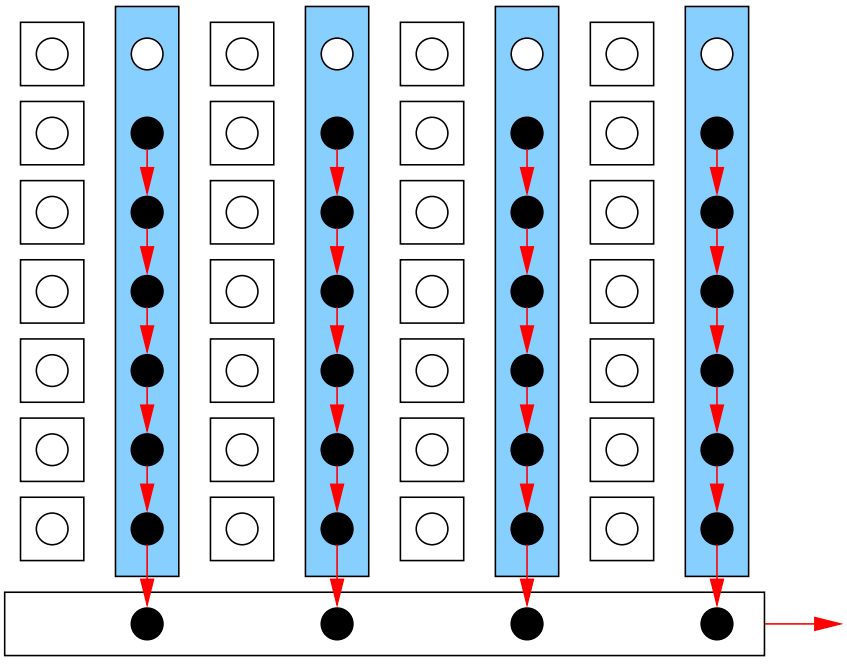
- the columns are interleaved with vertical shift registers
- the sensing area (fill factor) is reduced

Interline Transfer



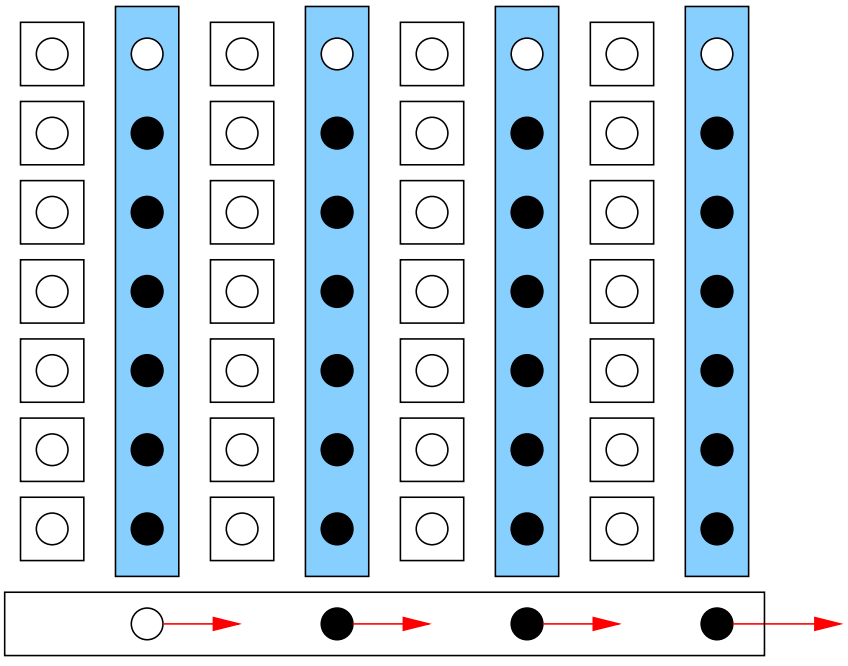
- initially, the charge is transferred into the vertical shift registers
- this transfer can take around $2.5\mu s$

Interline Transfer



- the charges are shifted row-by-row in the horizontal readout register
- there is no interference from newly collected charges

Interline Transfer



- the serial readout is done row-by-row
- this operation takes around $65\mu s$ per row

Interline Transfer

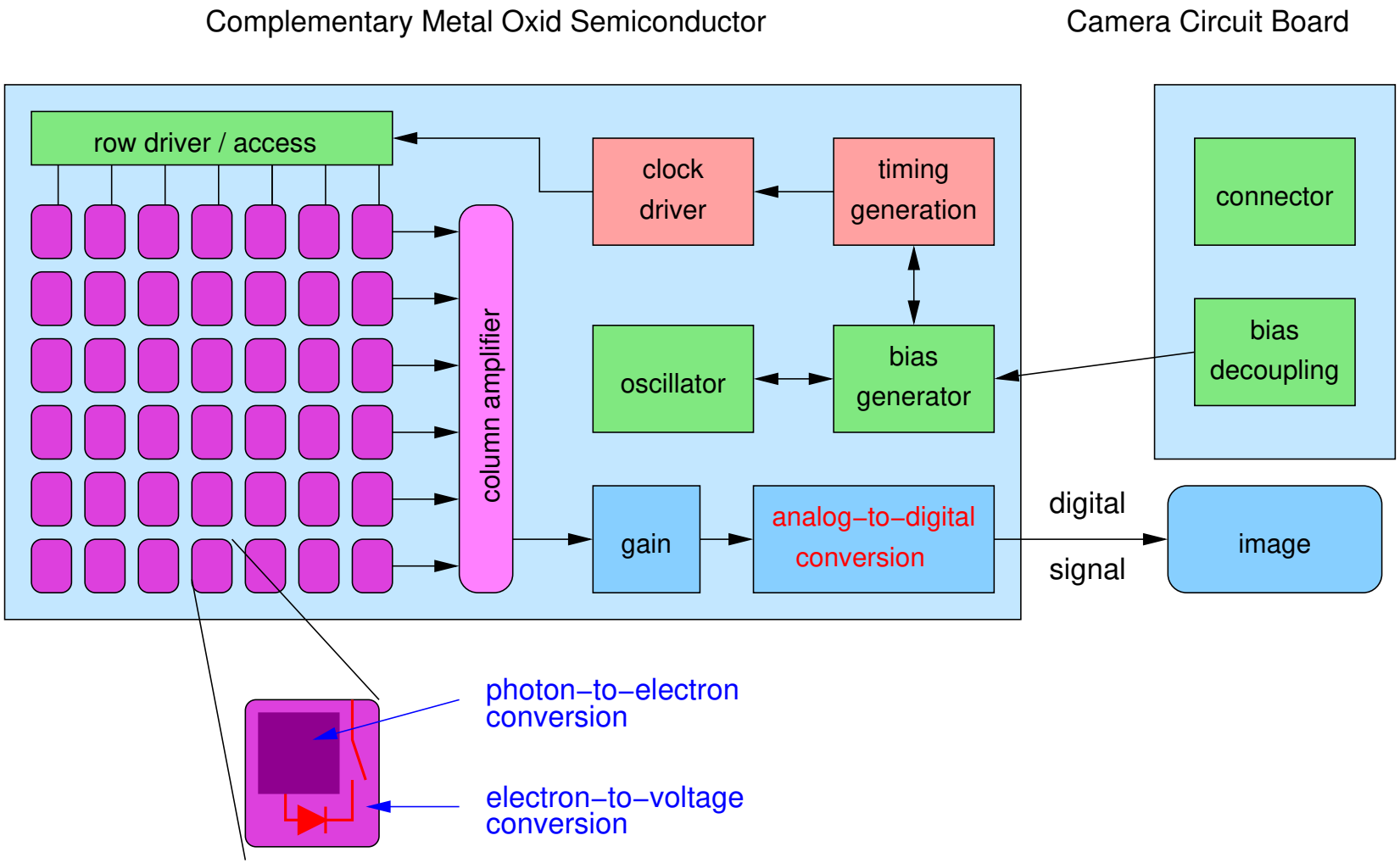
cons:

- reduced fill factor w.r.t. Full Frame and Frame Transfer technologies

pros:

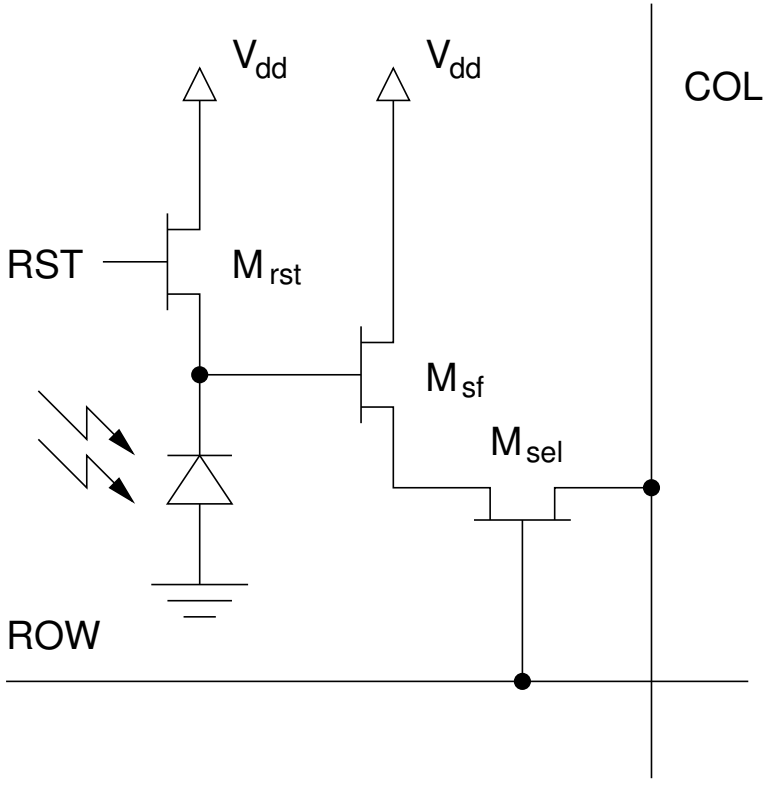
- no need of electromechanical shutter
- during the years, the fill factor increased from 50% up to 90%

The CMOS technology



The architecture of one sensing element

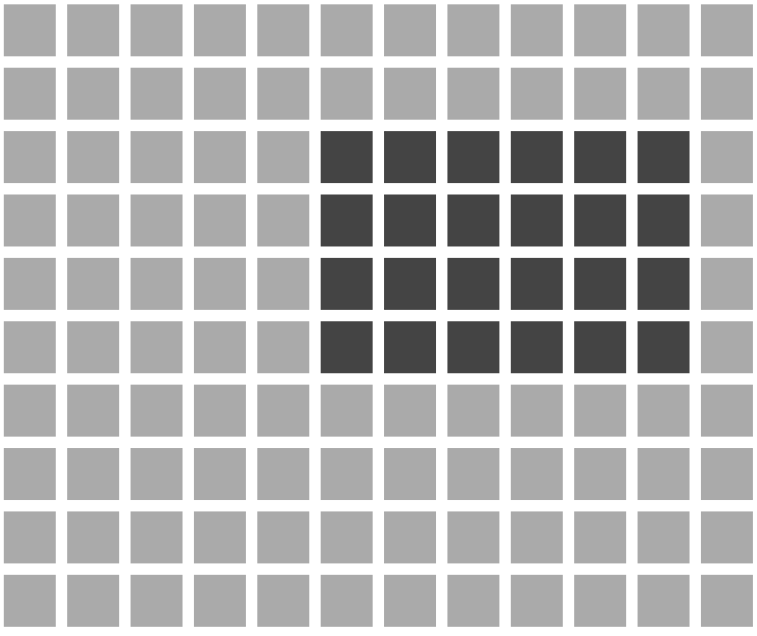
this is the circuit composing a CMOS sensing cell using 3 transistors



- photo-diode
- M_{rst} is the reset transistor
- M_{sf} is the amplifier in source follower configuration that collects the charges
- M_{sel} is the row selector

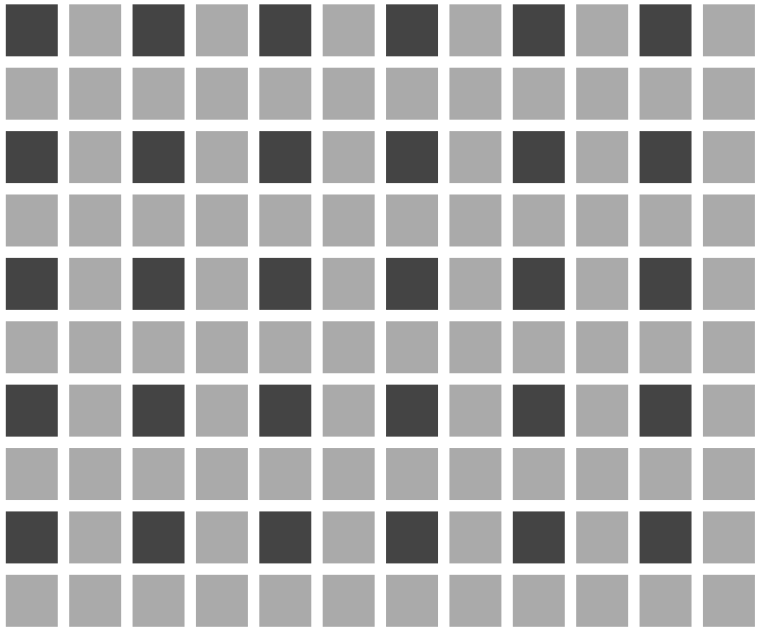
nowadays, there are technologies using 4, 5 or 6 transistors per sensing element

Image scanning: windowing



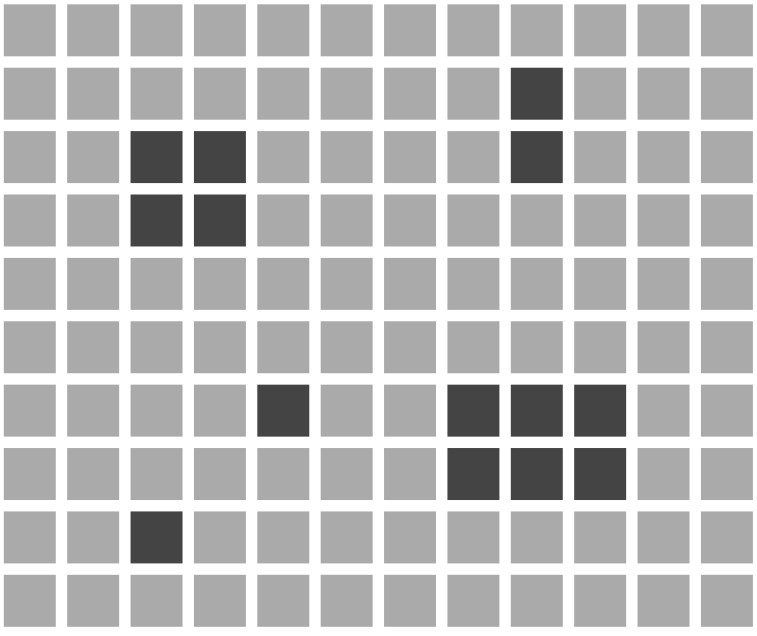
- a rectangular window can be read
- position and size of the window are arbitrarily selectable

Image scanning: subsampling



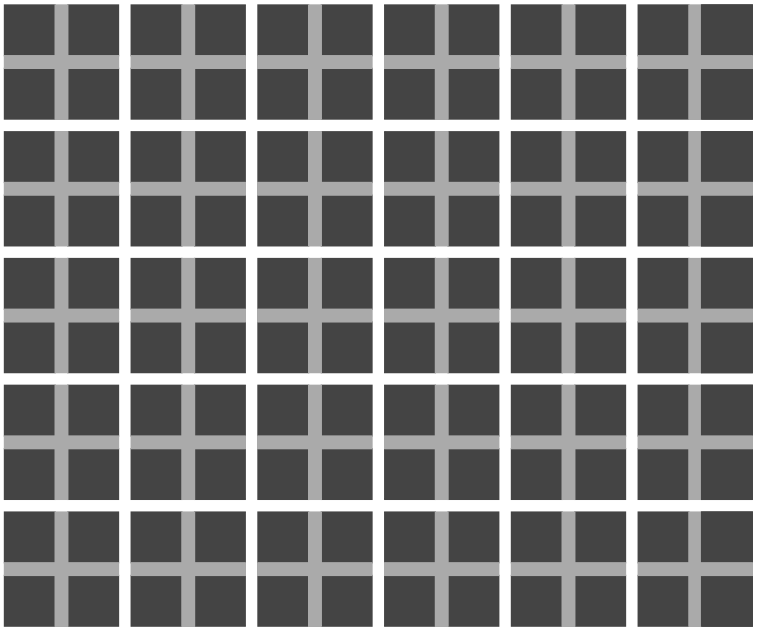
- pixels are discarded with a regular pattern during the acquisition
- automatic reduction of the resolution
- lower resolution $\longrightarrow$ less pixels to read $\longrightarrow$ higher frame-rate

Image scanning: random access



- each single pixel can be accessed
- saturated pixels can be individually reset

Image scanning: binning



- a set of adjacent pixels can be grouped into a bigger "aggregated" pixel
- this method can be also used with CCDs

CCD vs CMOS

- CMOS has a lower quality w.r.t. CCD, especially in case of low illumination
- CMOS requires less power (up to 100 times less)
- the CCD technology is preferred in high-end applications (scientific and military domains), where quality is preferred to costs and power consumption
- the CMOS technology allows faster acquisitions, so that it can be used for video recording
- CMOS allows direct access to sub-areas of the image
- CMOS is cheaper than CCD thanks to the integration of circuits

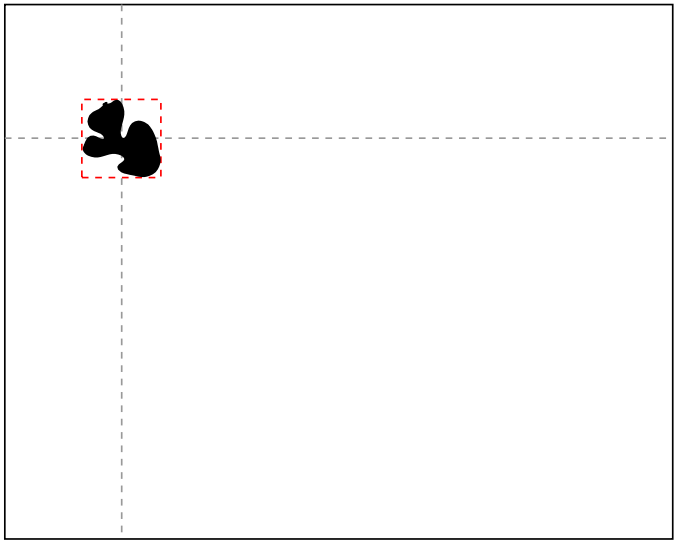
Effect of the sampling noise

- the wave-length of the visible spectrum is in the range 400-750 nm
- in 8-12 Mpxl sensors, the photo-sensor has size of around 5-6 μm
- the size of the photo-sensor is close to the wave-length
- the reduction of the photo-sensor size reduces the number of captured photons

$\rightarrow$ the amount of noise increases (the S/N ratio is reduced)

$\rightarrow$ a larger sensing area achieves less noise and better performance with poor illumination

Some basic objectives of image processing



$\{(x_i, y_i)\}$ is the set of pixels belonging to the object of interest in the image

in robotics some useful basic information are the **center of mass** and the **bounding box**

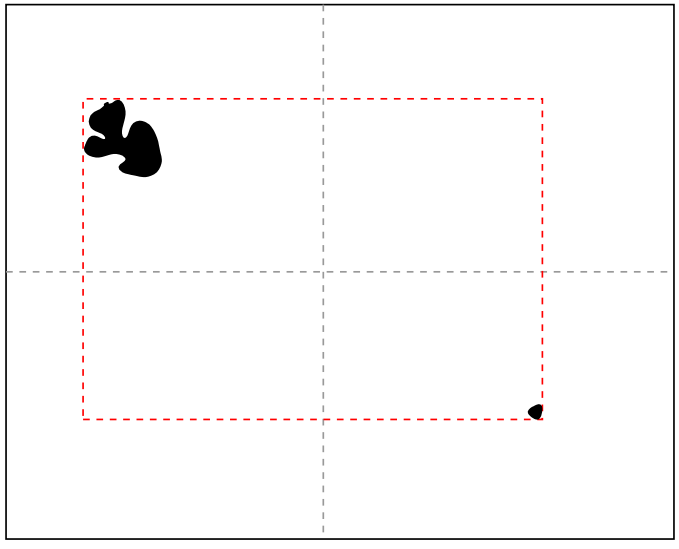
center of mass

$$x_m = \frac{\sum_{i=1}^{n} x_i}{n} \qquad y_m = \frac{\sum_{i=1}^{n} y_i}{n}$$

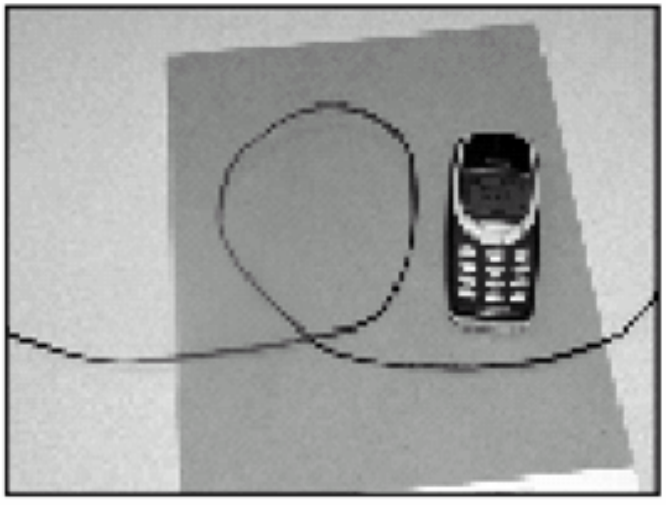
bounding box

lower-left $= (\min\{x_i\}, \min\{y_i\})$

upper-right $= (\max\{x_i\} \max\{y_i\})$

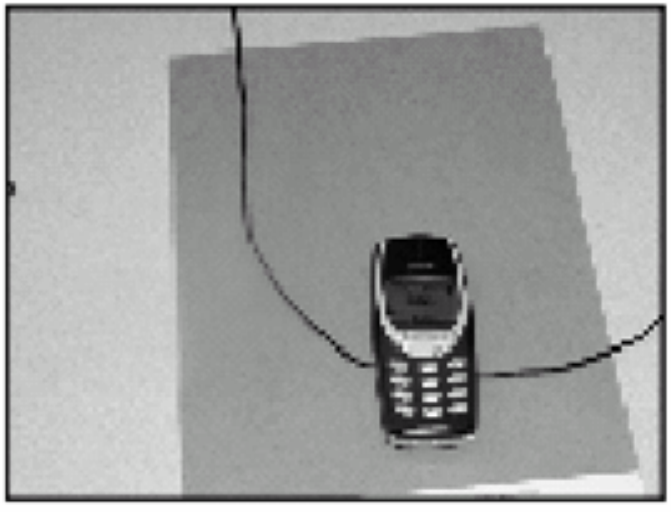


Segmentation

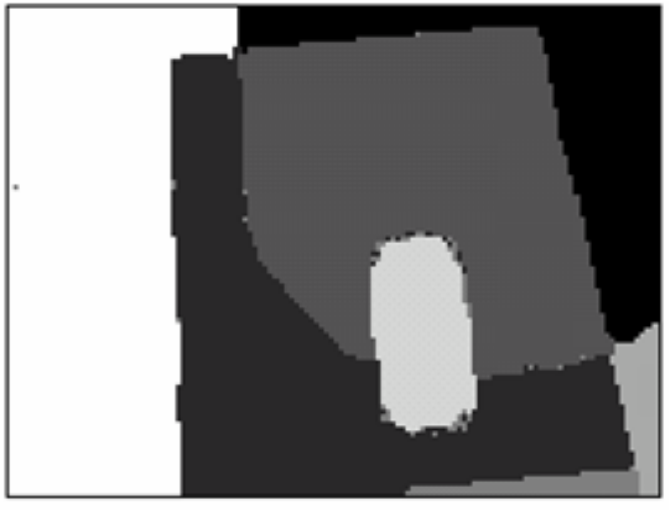


(a)

(b)

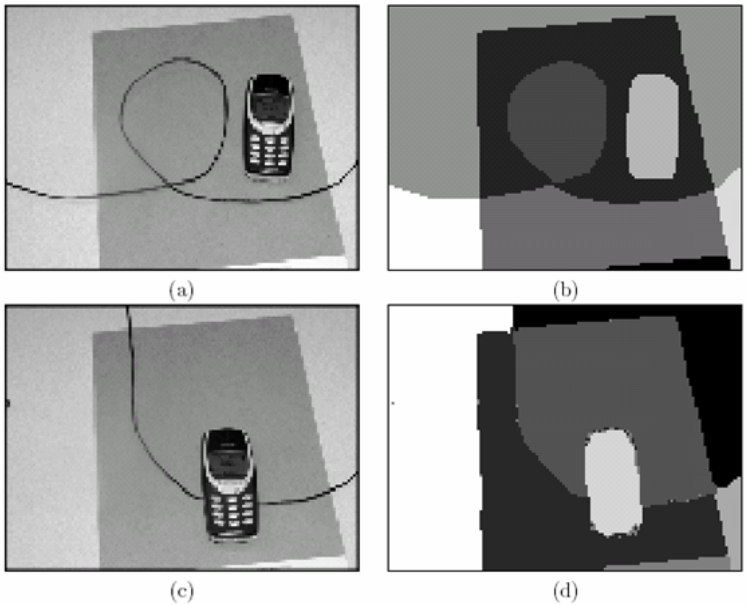


(c)



(d)

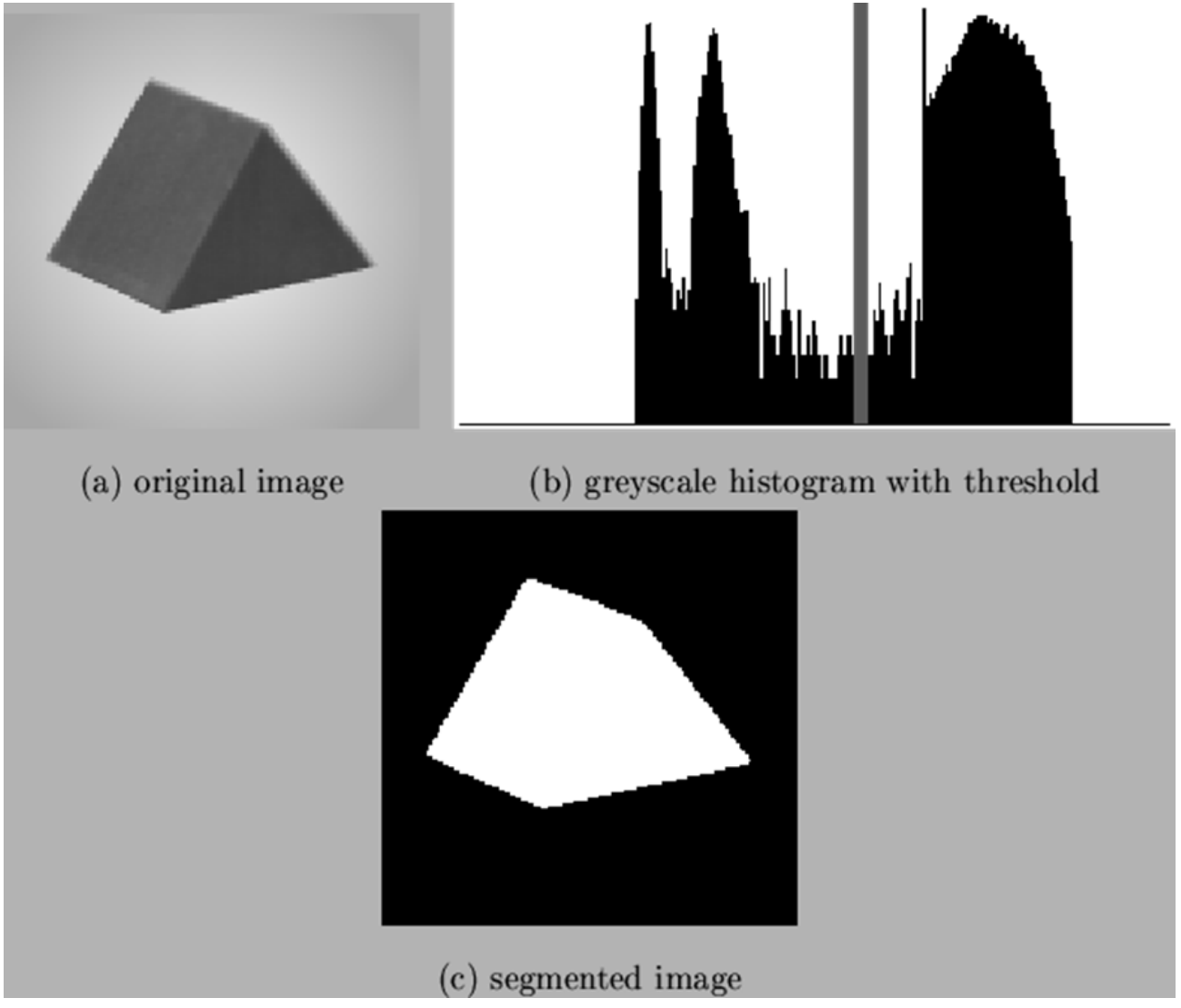
Segmentation



- very common method to extract features (objects or shapes) from an image
- based on the identification of regions having similar characteristics (intensity, color, etc.)

- distinct adjacent regions have significantly different characteristics
- the result is a set of regions or contours
- heavy computation in case of several regions, so often is implemented in hardware
- the efficiency can be improved in case additional information about the features to extract are available

Thresholding



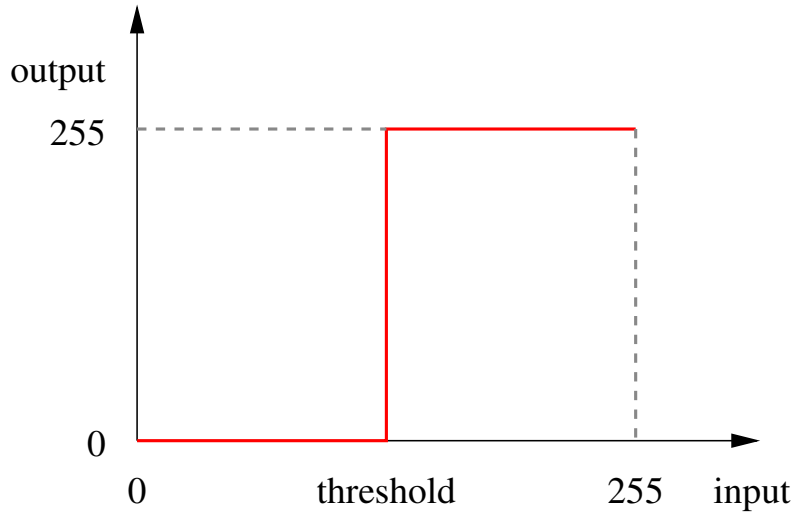
(a) original image

(b) greyscale histogram with threshold

(c) segmented image

Thresholding

- the thresholding is a simplified type of segmentation
- it can be used in case of simple images, i.e., made by few objects or colors
- in the simplest case, pixels are divided into foreground and background
- the thresholding operation is done by applying the following function:



RGB

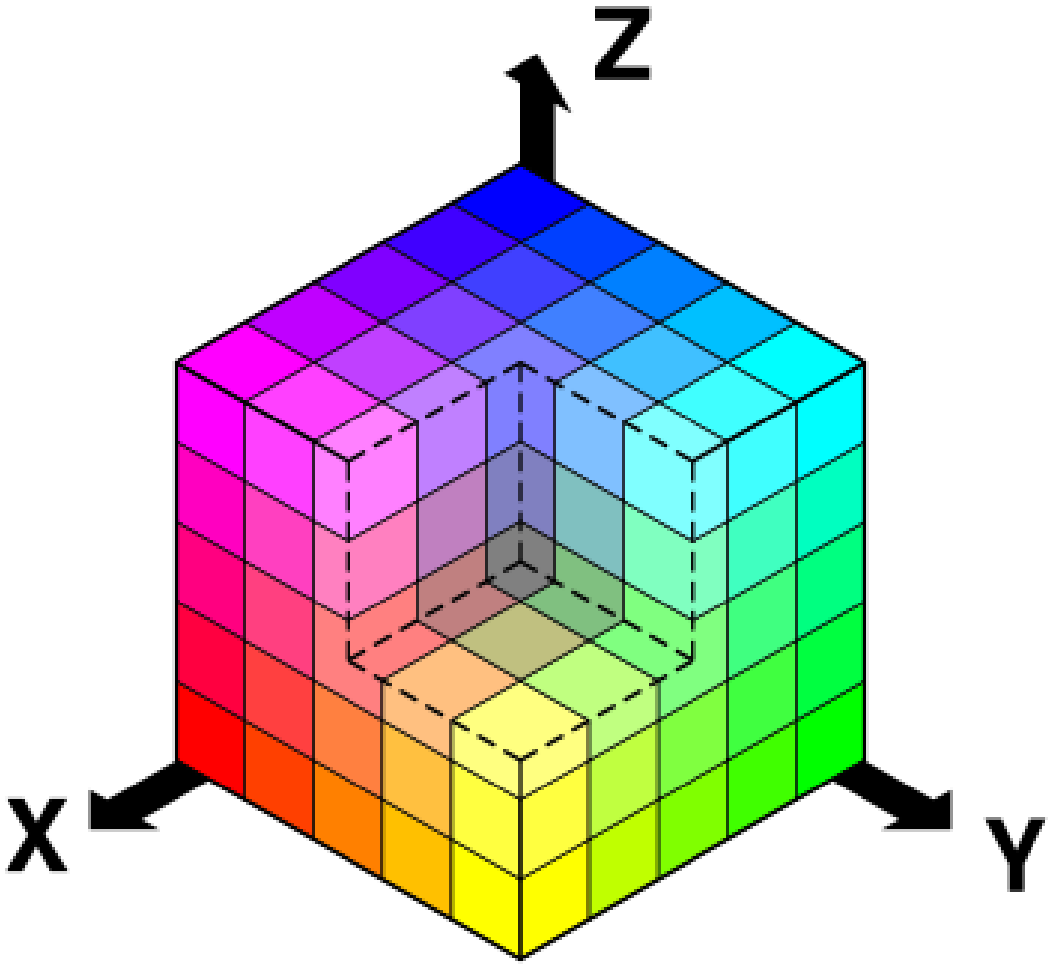
the RGB coding represents a colored pixel using a triple of numbers associated to the Red (R), Green (G) and Blue (B) components

the RGB is an additive coding that was born to display color images on cathod tubes TVs

typical coded values are in the following ranges:

- range $[0 \dots 1]$
- range $[0 \dots 100]$
- range $[0 \dots 255]$ (one byte per color)
- in hexadecimal representation [0x00 ... 0xff], e.g. used in HTML

The RGB space



YUV

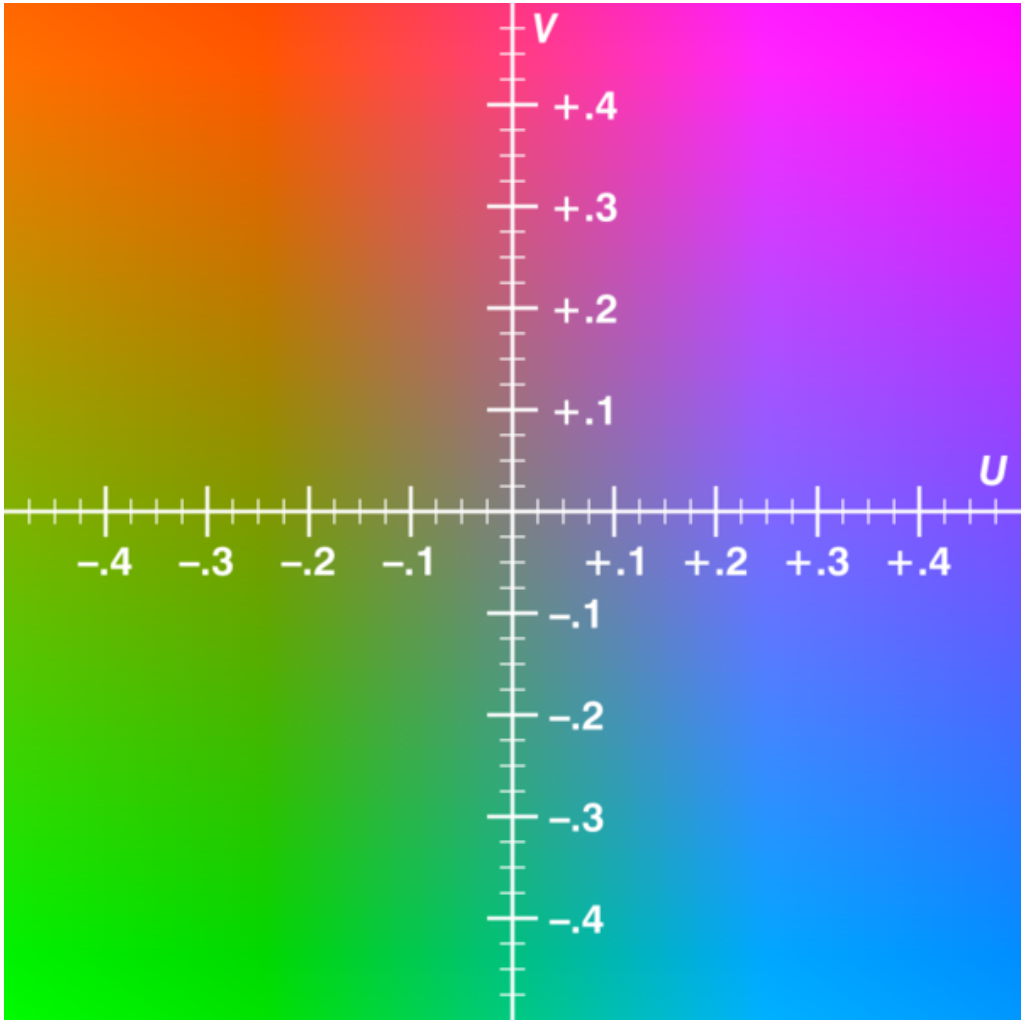
it is a model based on luminance and chrominance

- Y (luma) defines the luminosity
- the U-V components contain the information on the color

it is possible to transform a RGB to a YUV representation using the following transformation:

$$\begin{bmatrix} Y \\ U \\ V \end{bmatrix} = \begin{bmatrix} 0.299 & 0.587 & 0.144 \\ -0.14713 & -0.28886 & 0.436 \\ 0.615 & -0.51498 & -0.10001 \end{bmatrix} \begin{bmatrix} R \\ G \\ B \end{bmatrix}$$

YUV: the U-V plane



the plane is represented with $Y = 0.5$

A common low efficient approach to segmentation

2 threshold values for each pixel (6 thresholds in total):

```
if ((Y >= Y_LowerThresh) && (Y <= Y_UpperThresh) &&
    (U >= U_LowerThresh) && (U <= U_UpperThresh) &&
    (V >= V_LowerThresh) && (V <= V_UpperThresh))
  pixel-color = color-class;
```

- the previous code fragment uses 6 comparisons instructions per pixel
- to evaluate 32 distinct colors up to 192 operations are required

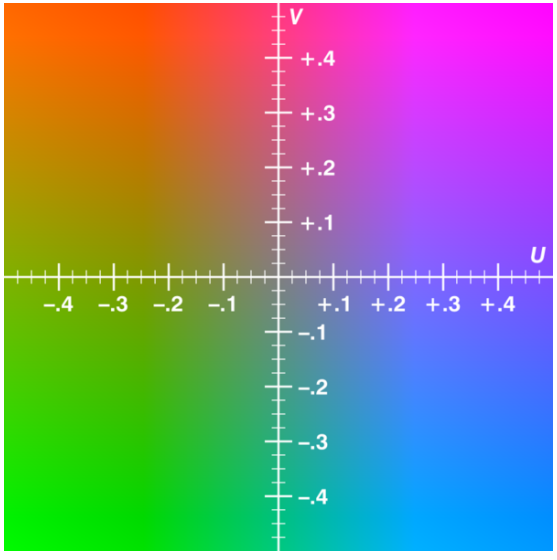
the execution can be very inefficient in modern architectures with pipelines and speculative execution

An efficient segmentation approach

- based on the quantization of the range of variation of colors
- let's use a quantization made by 10 levels

e.g., the "orange" color can be described by the following arrays:

```
YClassOrange[ ] = {0,1,1,1,1,1,1,1,1,1};
UClassOrange[ ] = {1,1,1,0,0,0,0,0,0,0};
VClassOrange[ ] = {0,0,0,0,0,0,0,1,1,1};
```



An efficient segmentation approach

considering the previous arrays:

```
YClassOrange[ ] = {0,1,1,1,1,1,1,1,1,1};
UClassOrange[ ] = {1,1,1,0,0,0,0,0,0,0};
VClassOrange[ ] = {0,0,0,0,0,0,0,1,1,1};
```

to check whether the pixel identified by the color (1,0,9) is orange it suffices to check the boolean value

```
isOrange = YClassOrange[1] & UClassOrange[0] & VClassOrange[9]
         = 1 & 1 & 1 = 1
```

where `isOrange` is "1" if the pixel is classified as orange

NOTE: arrays are indexed in C notation; e.g., the *ClassOrange arrays are indexed from 0 to 9

Assigning one class over n

in general, to distinguish n different colors, a total amount of n AND bit-wise operations and $3n$ arrays will be required

for example, the "green" color class is defined by

```
YClassGreen[ ] = {0,1,1,1,1,1,1,1,1,1};
UClassGreen[ ] = {1,1,1,0,0,0,0,0,0,0};
VClassGreen[ ] = {1,1,1,0,0,0,0,0,0,0};
```

so that to check the pixel having colors (1,0,9) the following test is used:

```
isGreen = YClassGreen[1] & UClassGreen[0] & VClassGreen[9]
        = 1 & 1 & 0 = 0
```

Parallelizing of the assignment

it is possible to pack several class codes into a single bit mask

```
YC1assOrange[ ] = {0 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 ,1 };
UClassOrange[ ] = {1 ,1 ,1 ,0 ,0 ,0 ,0 ,0 ,0 ,0 };
VClassOrange[ ] = {0 ,0 ,0 ,0 ,0 ,0 ,0 ,1 ,1 ,1 };

YClassGreen[ ]  = { 0, 1, 1, 1, 1, 1, 1, 1, 1, 1};
UClassGreen[ ]  = { 1, 1, 1, 0, 0, 0, 0, 0, 0, 0};
VClassGreen[ ]  = { 1, 1, 1, 0, 0, 0, 0, 0, 0, 0};

YClass[ ]       = {00,11,11,11,11,11,11,11,11,11}
UClass[ ]       = {11,11,11,00,00,00,00,00,00,00}
VClass[ ]       = {01,01,01,00,00,00,00,10,10,10}
```

- most significant bit $\rightarrow$ orange
- least significant bit $\rightarrow$ green

Parallelizing of the assignment

to assign a color class to the (1,0,9) pixel the same bit-wise operation is required:

```
ClassMask = YClass[1] & UClass[0] & VClass[9]
```

leading to

&	00	01	10	11
00	00	00	00	00
01	00	01	00	01
10	00	00	10	10
11	00	01	10	11

classMask is “10” if the pixel is orange, “01” if green, and “00” if some other color

Parallelizing of the assignment

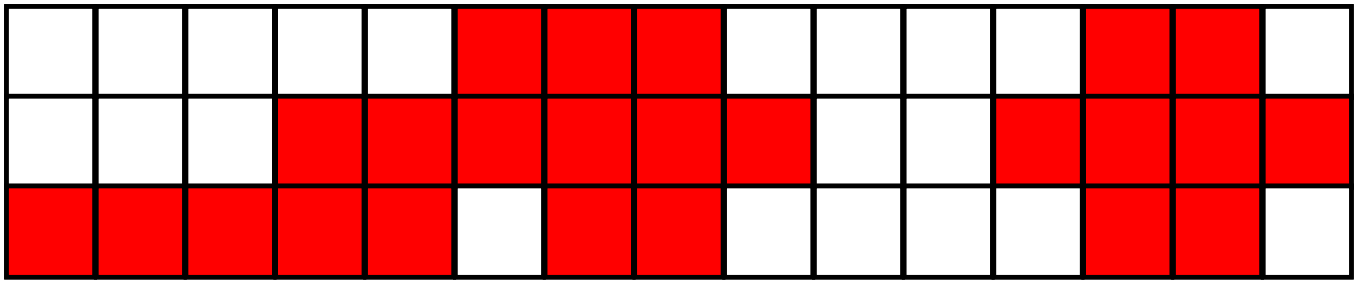
- up to 32 color classes can be packed within an array of 32-bit integers
- the color class can be determined using two AND operations only

```
YClass[ ] = {00,11,11,11,11,11,11,11,11,11}
UClass[ ] = {01,01,01,00,00,00,00,10,10,10}
VClass[ ] = {00,00,00,01,01,01,00,10,10,10}
```

the small size of the data structure allows very efficient operations thanks to the use of cache memory

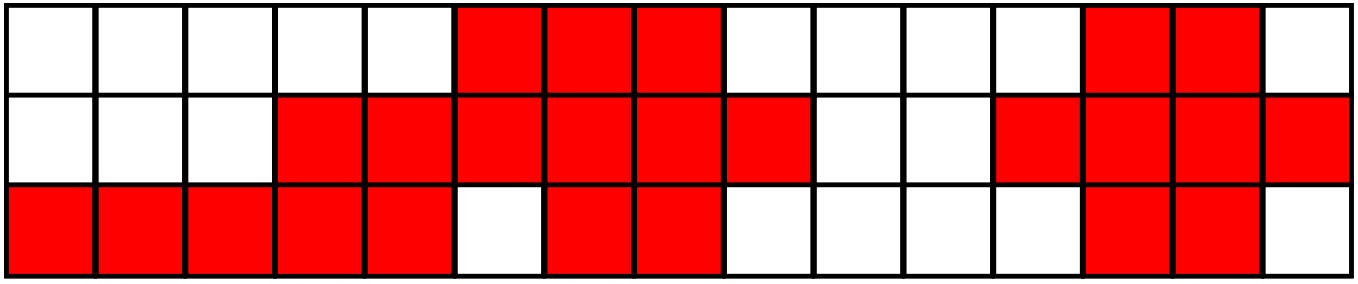
Grouping of pixels

- once pixels are classified, they must be grouped
- adjacent pixels belong to the same object



horizontal lines of pixels can be encoded using the Run Length Encoding method (RLE)

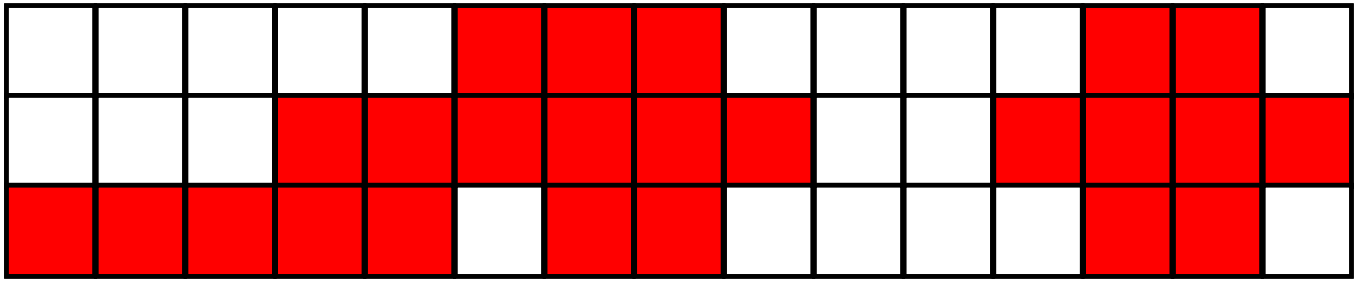
Run Length Encoding compression



- the compression takes place row-by-row
- horizontal sequences of pixels (runs) are identified
- the length of the each run is stored in each row

the RLE works well for long sequences of pixels having the same color, otherwise the size of the image may increase

Run Length Encoding compression



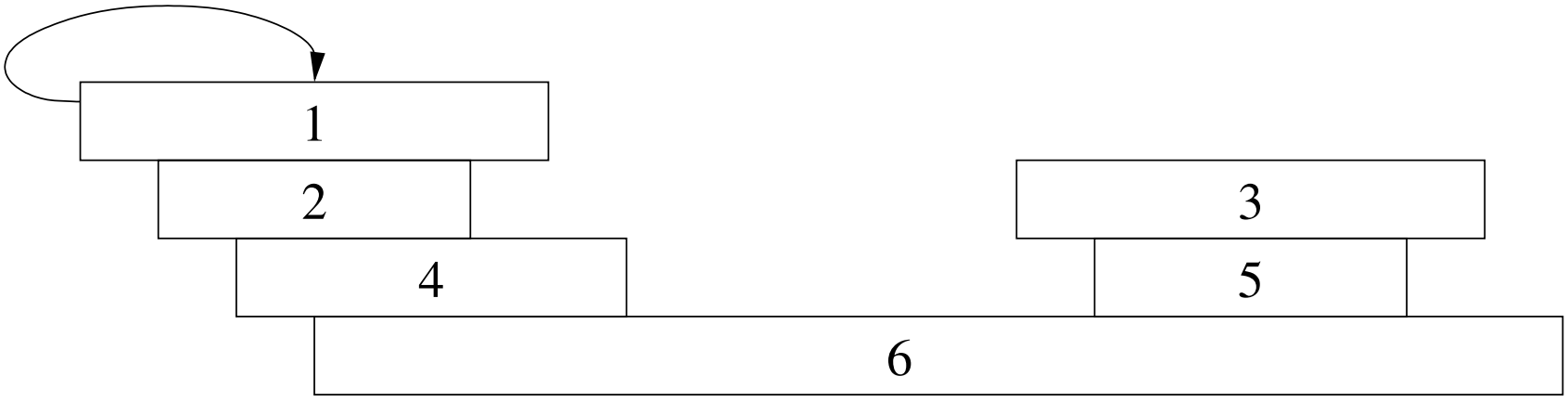
the above 3 rows are compressed as follows (by default, the coding start with white pixels in each row)

```
5 3 4 2 1
3 6 2 4
0 5 1 2 4 2 1
```

Grouping of pixels

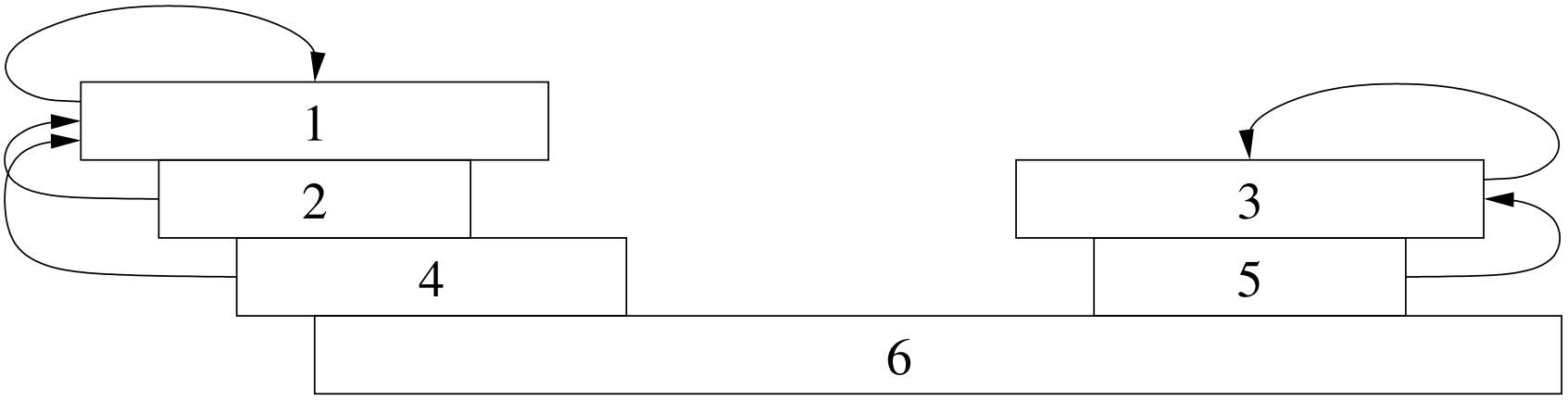
- the technique is suitable for robotics applications consisting of images having few color variations
- the merging considers vertical adjacencies, since the horizontal adjacency is done by the RLE compression
- runs are vertically grouped based on the 4-connectivity
- runs are grouped while object features (center of mass, bounding box, etc.) are collected

Find-union and merging algorithm



- parsing the first row, the run 1 is classified as the root of a tree
- the pointer of a root node points to itself

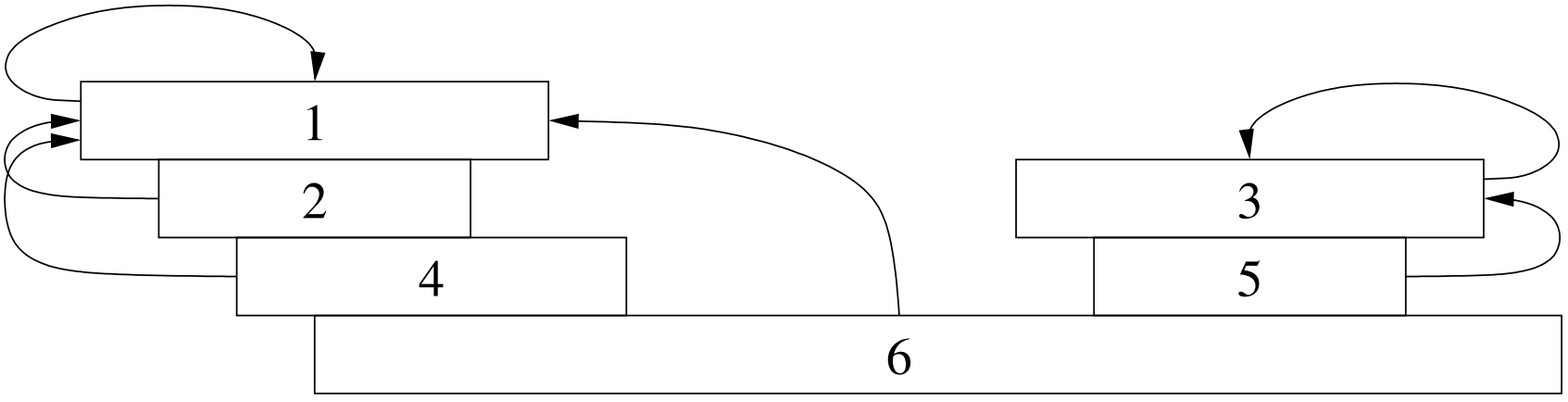
Find-union and merging algorithm



parsing the next rows:

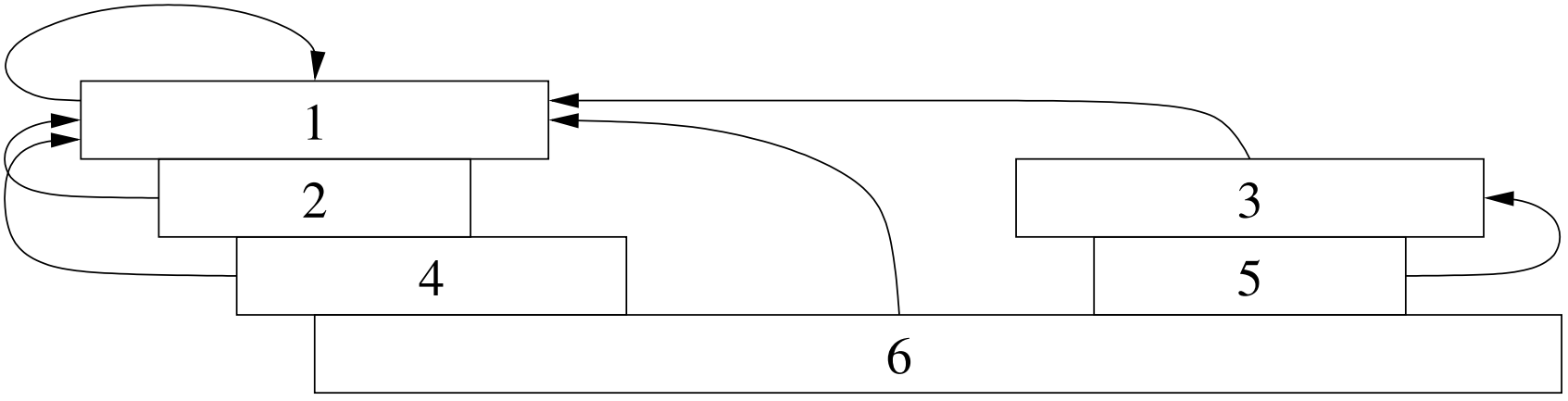
- run 2 is linked to run 1
- run 3 is found to be another root
- run 4 is linked to run 2, and its pointer is linked to the root of 2, which is 1
- run 5 is directly linked to 3

Find-union and merging algorithm



- when run 6 is considered (from left to right), it is found to be connected to run 4
- therefore, it is linked to the root of run 4, which is run 1

Find-union and merging algorithm



- afterwards, run 6 is found to be also connected to run 5
- since run 5 has a distinct root w.r.t. the root of 4, the pointer of run 3 (root of run 5) is linked to the first found root (run 1)

in this way, different trees are merged

Find-union and merging algorithm

```
 1: i = 0; k = 0
 2: for each row i
 3:   for each run k
 4:     if i == 0 then
 5:       k is the root of a new tree
 6:     else
 7:       if k is linked to at least one run in row i-1
 8:         if k is linked to more than one run in row i-1
 9:           find the roots of the 2 runs
10:           merge (union) of the two roots
11:           if k is linked to more than 2 runs
12:             go to line 9
13:         else
14:           the root of k is set equal to those of run in row i-1
15:       else k is root of a new tree
16:     save the information of the tree
17:     k = k + 1
18:  i = i + 1
```

Performance of the algorithm

- resolution : 640×480
- frame rate : 30Hz
- processor : Intel Pentiun III (700 MHz)
- processor utilization : around 60%