Data Structures
for
Image Representation

Array versus Matrix Operations

- Consider two 2 x 2 images

$$\begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \text{ and } \begin{bmatrix} b_{11} & b_{12} \\ b_{21} & b_{22} \end{bmatrix}$$

- Array Product is:

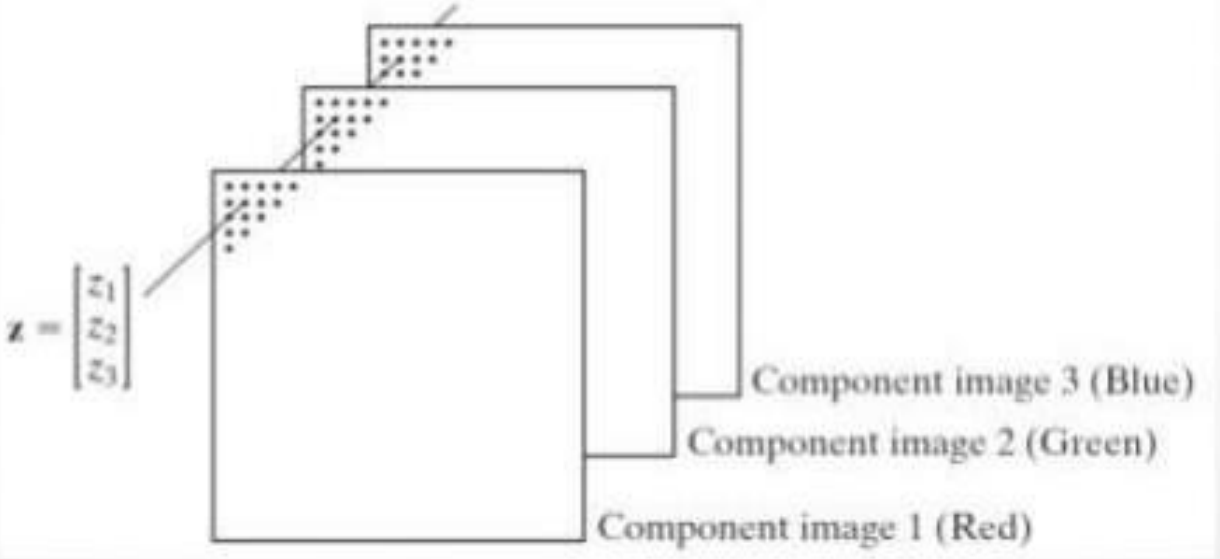
$$\begin{bmatrix} a_{11}b_{11} & a_{12}b_{12} \\ a_{21}b_{21} & a_{22}b_{22} \end{bmatrix}$$

- Matrix Product is:

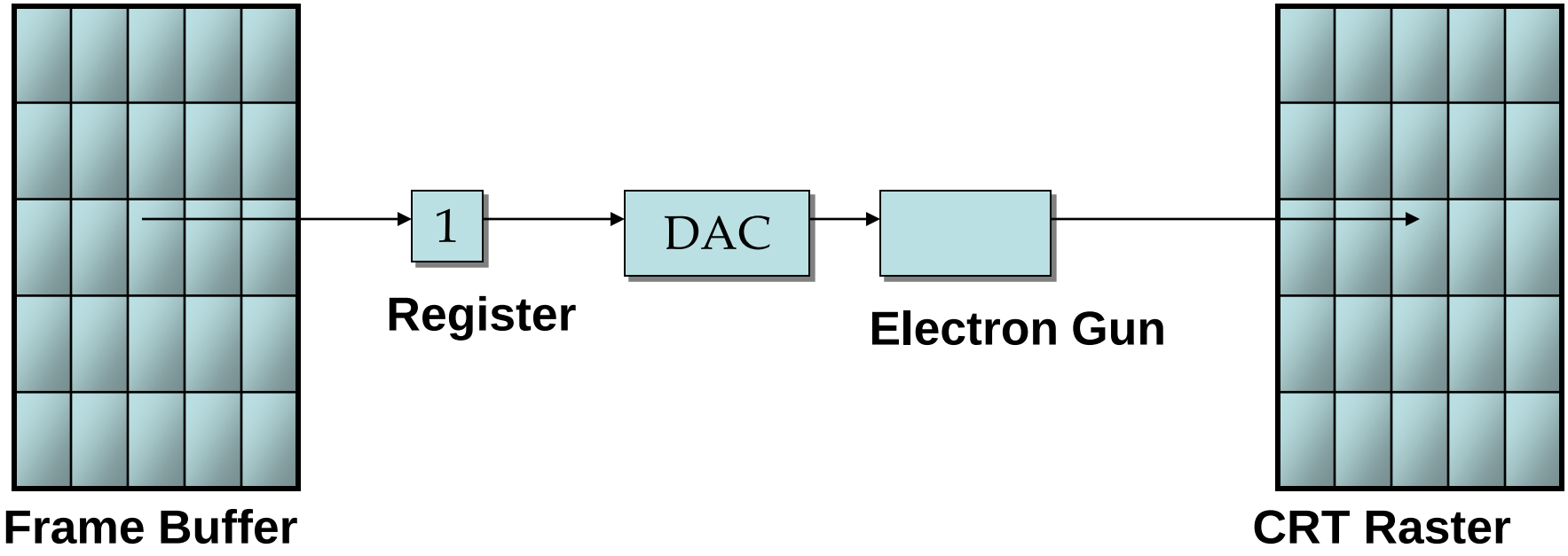
$$\begin{bmatrix} a_{11}b_{11} + a_{12}b_{21} & a_{11}b_{12} + a_{12}b_{22} \\ a_{21}b_{11} + a_{22}b_{21} & a_{21}b_{12} + a_{22}b_{22} \end{bmatrix}$$

Vector and Matrix Operations

- Color images are formed in RGB color space by using red, green, and blue component images.
- *Each* pixel of an RGB image has 3 components, which can be organized in the form of a *column vector.*

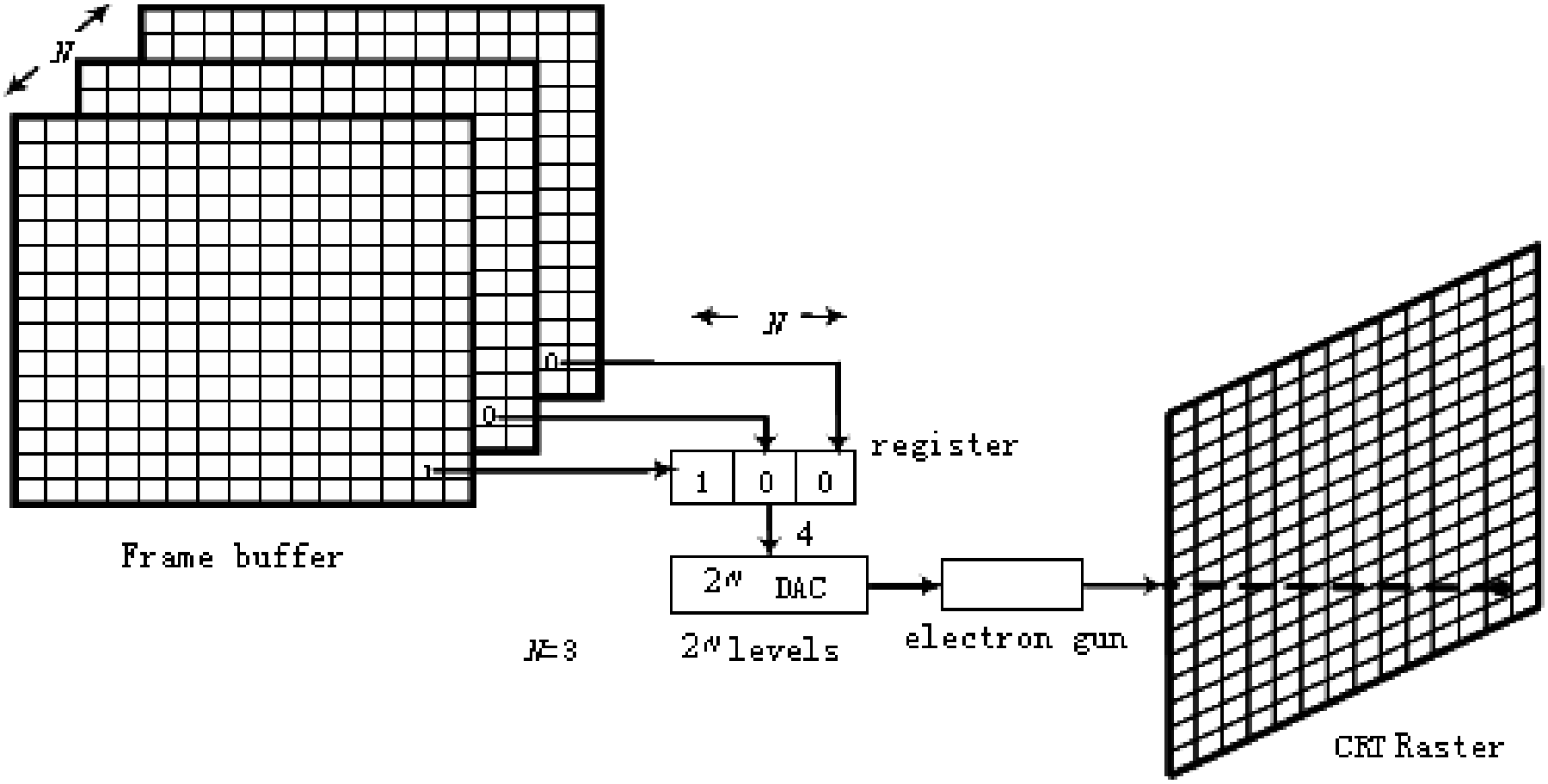


One bit plane frame buffer

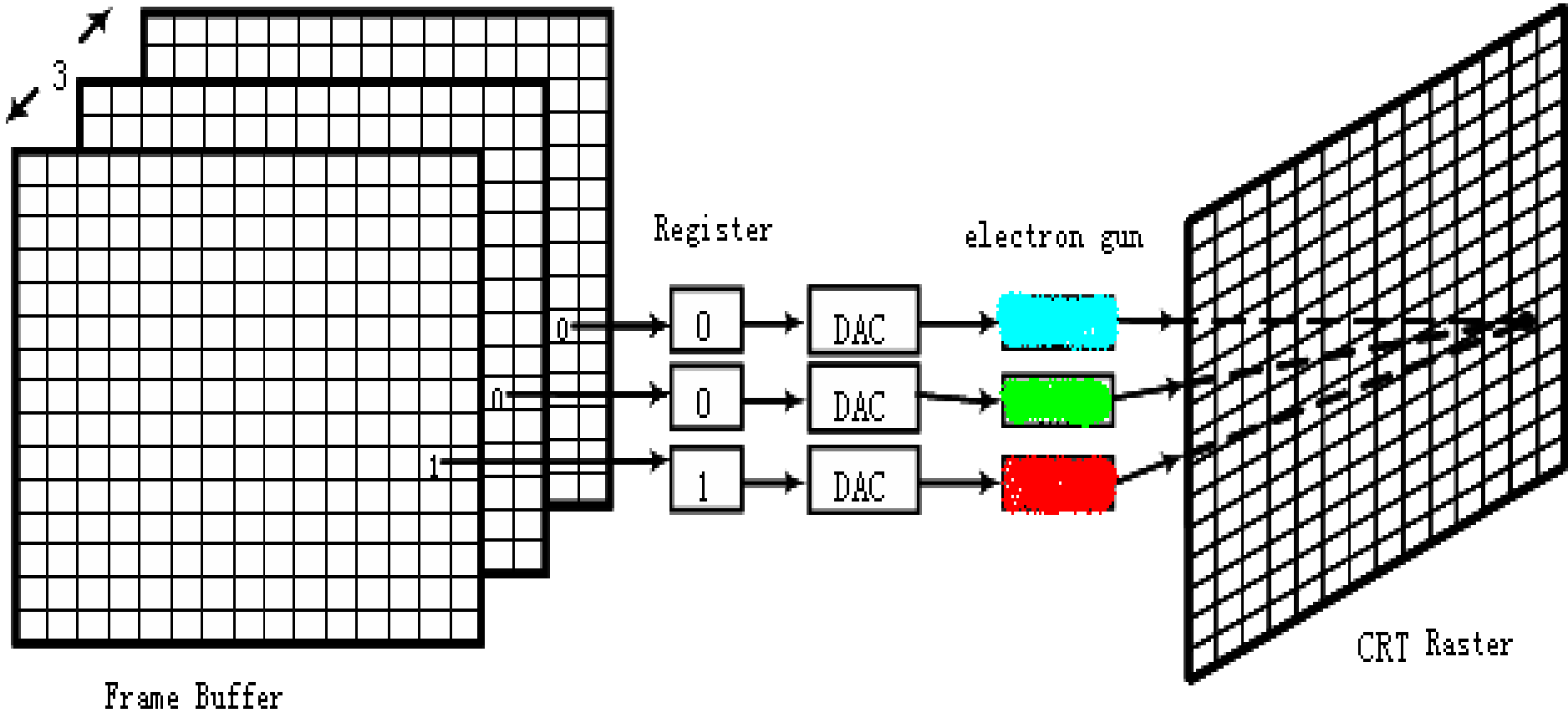


A single-bit-plane(1 bit per pixel) black-white frame buffer raster CRT graphics device

N bit gray level frame buffer



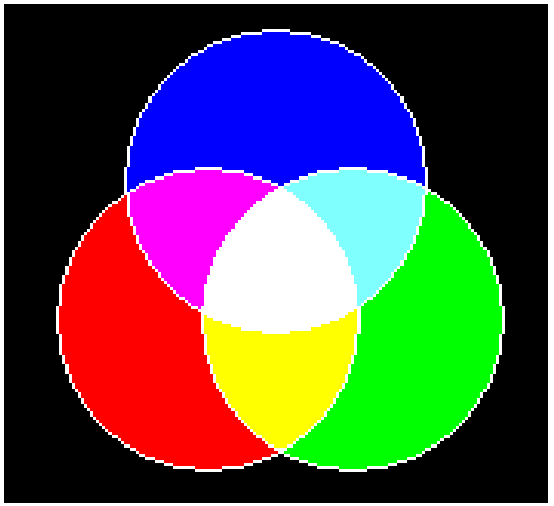
Simple color frame buffer

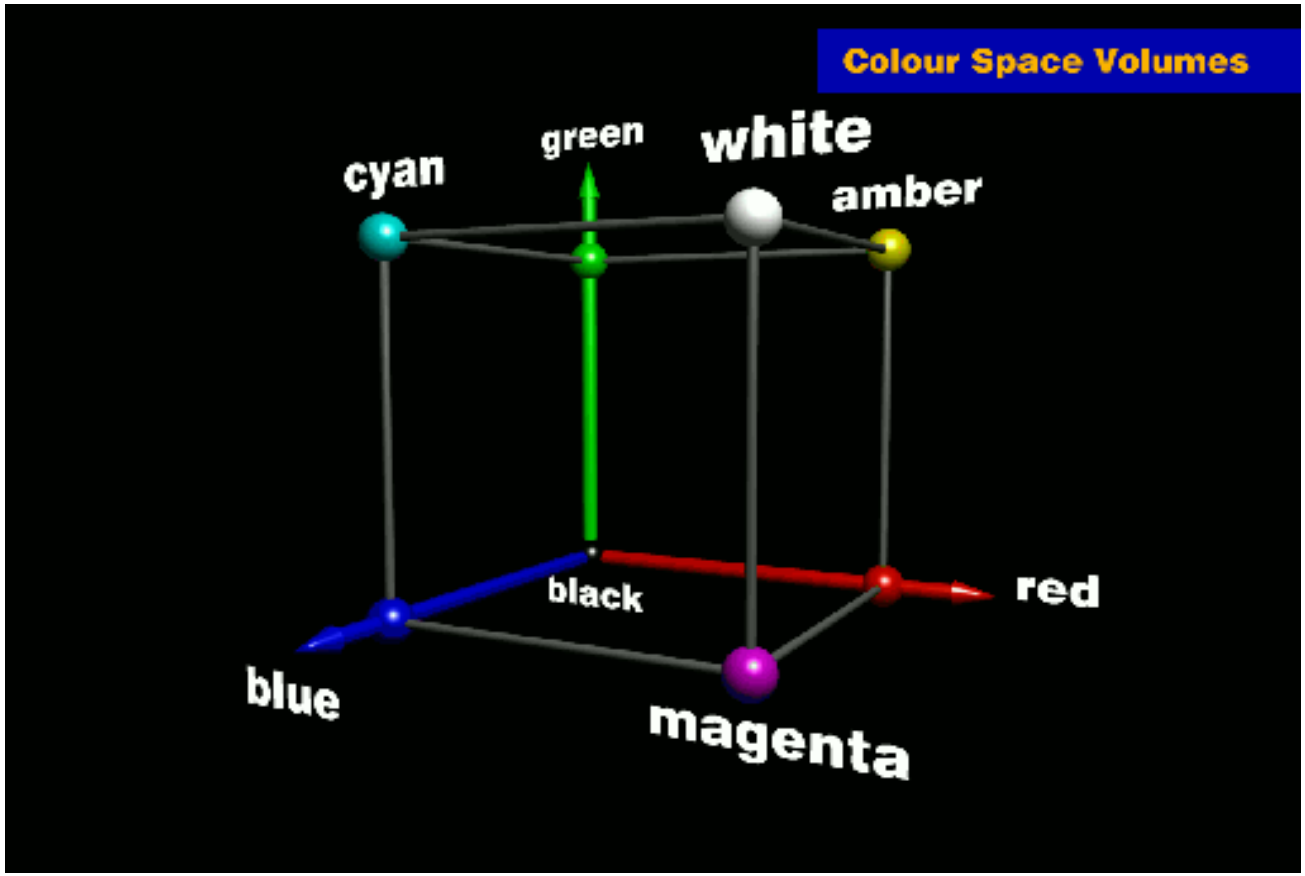


3 Bit plane frame buffer color combinations

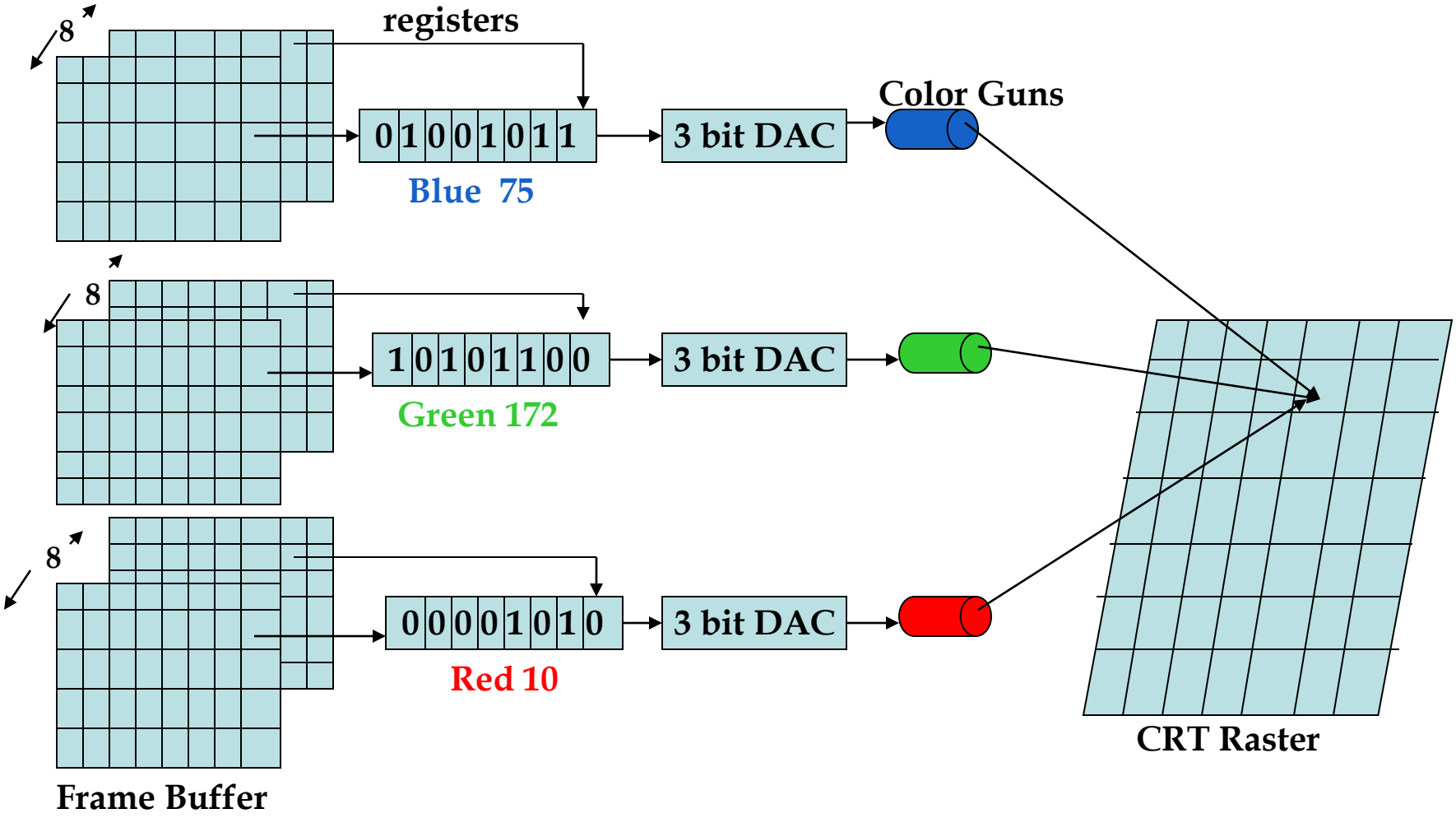
	Red	Green	Blue
Black	0	0	0
Red	1	0	0
Green	0	1	0
Blue	0	0	1
Yellow	1	1	0
Cyan	0	1	1
Magenta	1	0	1
White	1	1	1

RGB Color Space





A 24 Bit plane color frame buffer

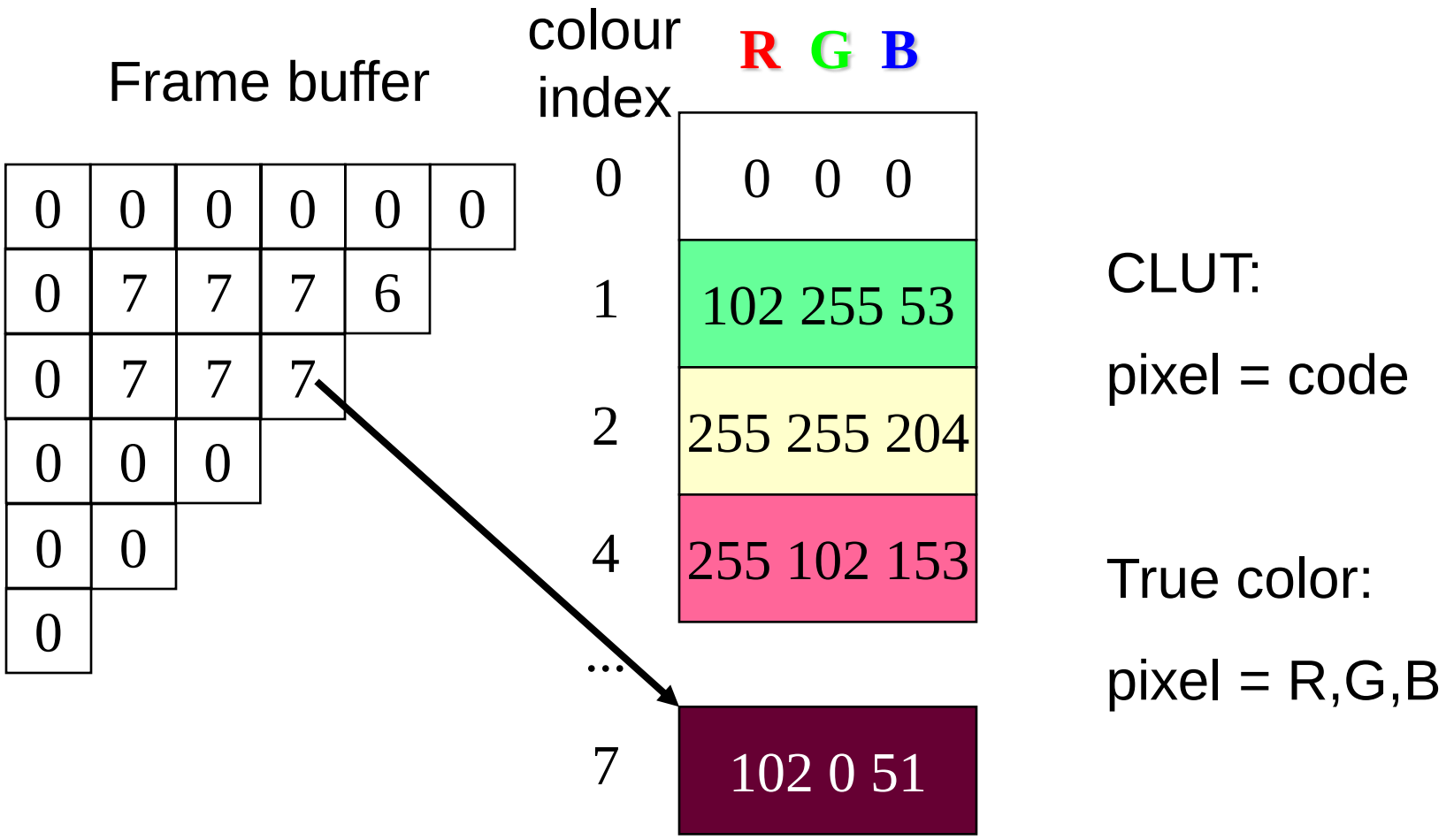


Color Image Generation Techniques

- Direct coding
  - RGB values directly stored for each pixel
  - Memory intensive

- Color lookup table - a compromise technique
  - Each pixel points to a location in a table
  - Saves space

Color Lookup Table



CLUT:

pixel = code

True color:

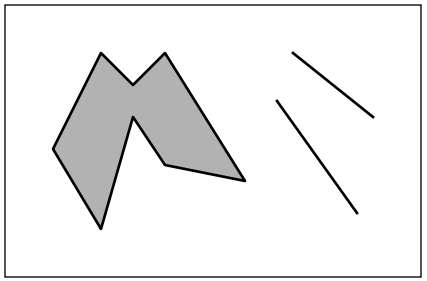
pixel = R,G,B

Color Map Look-Up Tables

- If color depth = *b* bits,
- and each LUT entry is *w* bits wide
- Then the system **can display 2^w colors, any 2^b at one time**

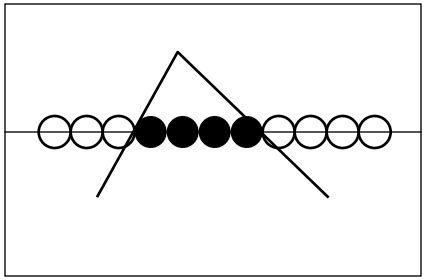
Data Structures

- **On-the-fly scan conversion**



using geometry and visual attributes

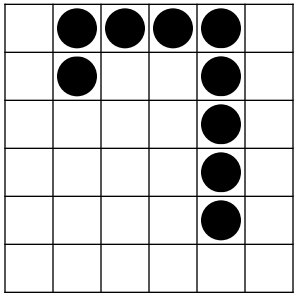
- **Run-length encoding**



.....○3●4○4.....

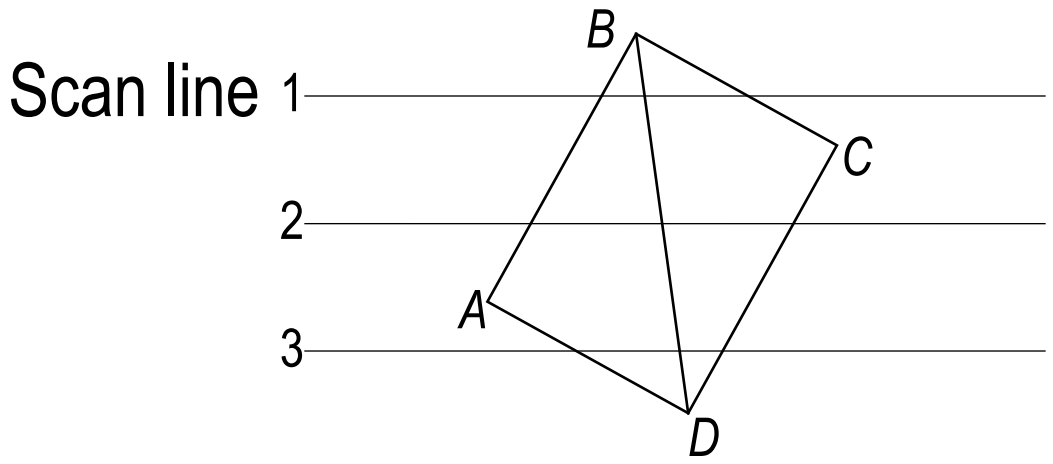
intensity	run-length

- **Cell organization**



- **Frame buffer**

On-the-fly Scan Conversion



Scan line

1	2	3
BC ← b	*BC*	*BC*
BA	*BA* ← b	*BA*
BD ← e	*BD*	*BD* ← b
CD	*CD* ← e	*CD*
AD	*AD*	*AD* ← e

1	2	3
BA ← b	*BA* ← b	*BA*
BC	*BC*	*BC*
BD ← e	*BD*	*BD* ← b
CD	*CD* ← e	*CD*
AD	*AD*	*AD* ← e

1	2	3
BD ← b	*BD* ← b	*BD* ← b
BA	*BA*	*BA*
BC ← e	*BC*	*BC*
CD	*CD* ← e	*CD*
AD	*AD*	*AD* ← e

The active edge list may be lengthy.

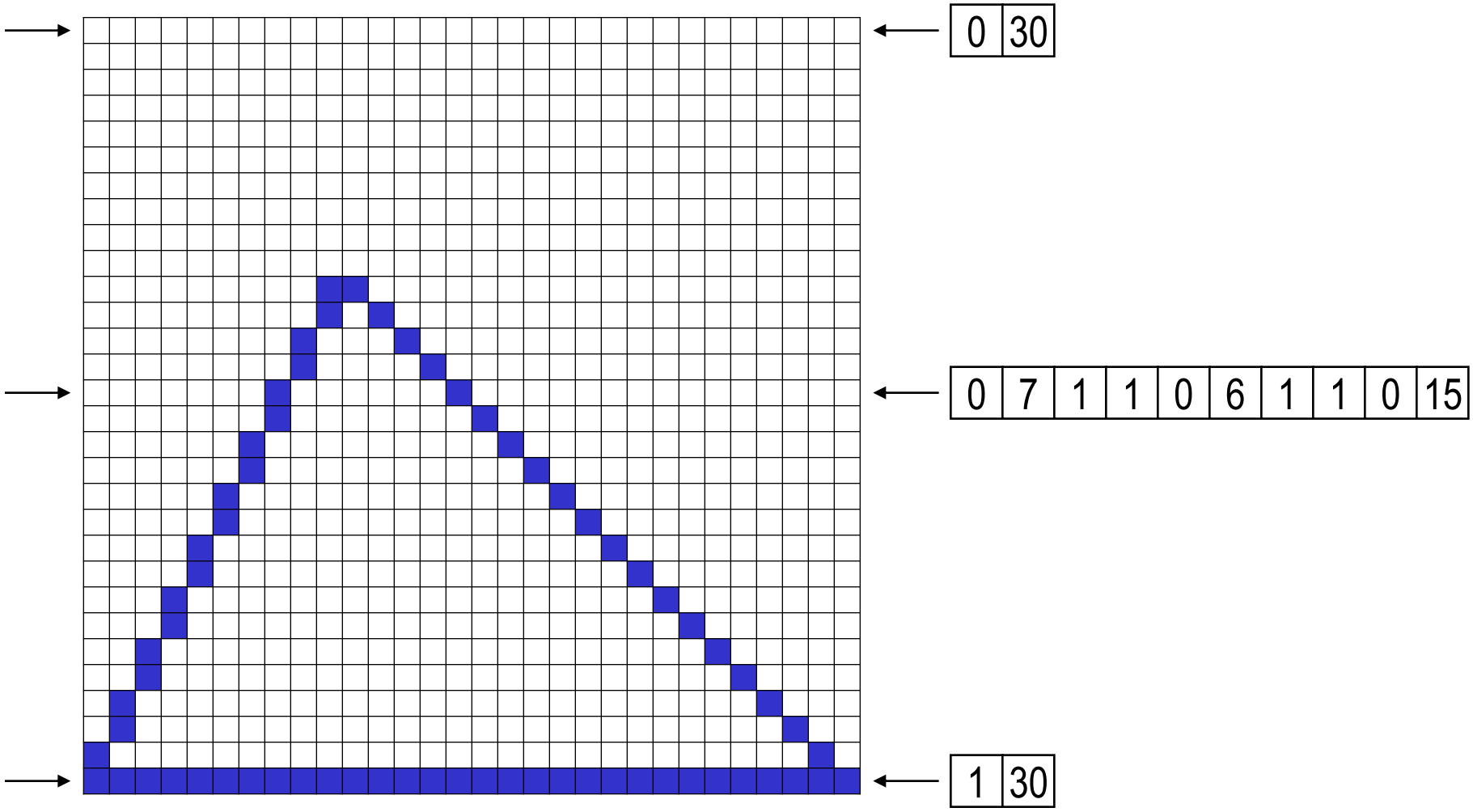
∴ Need Sorting !!!

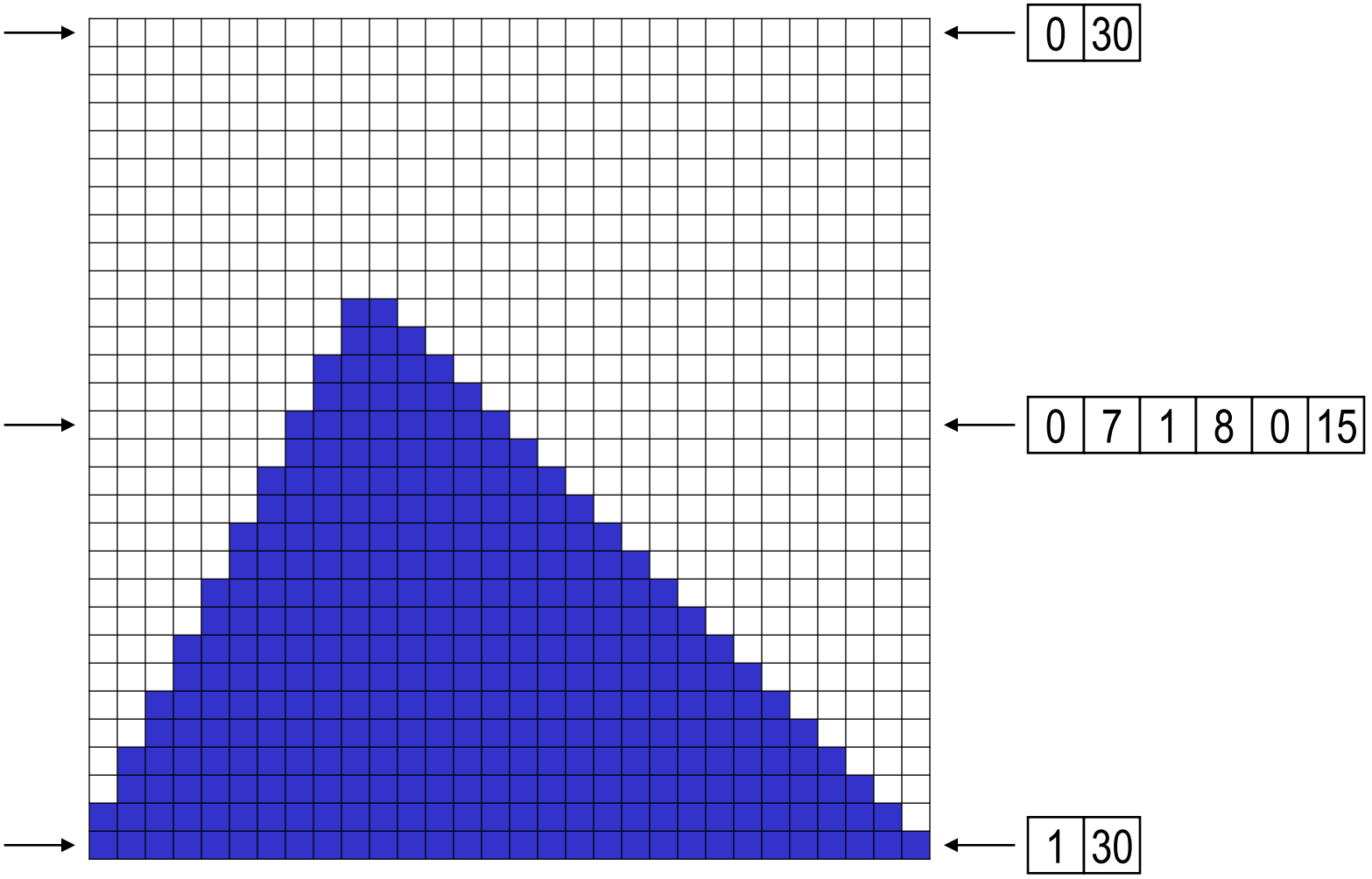
Run-length Encoding

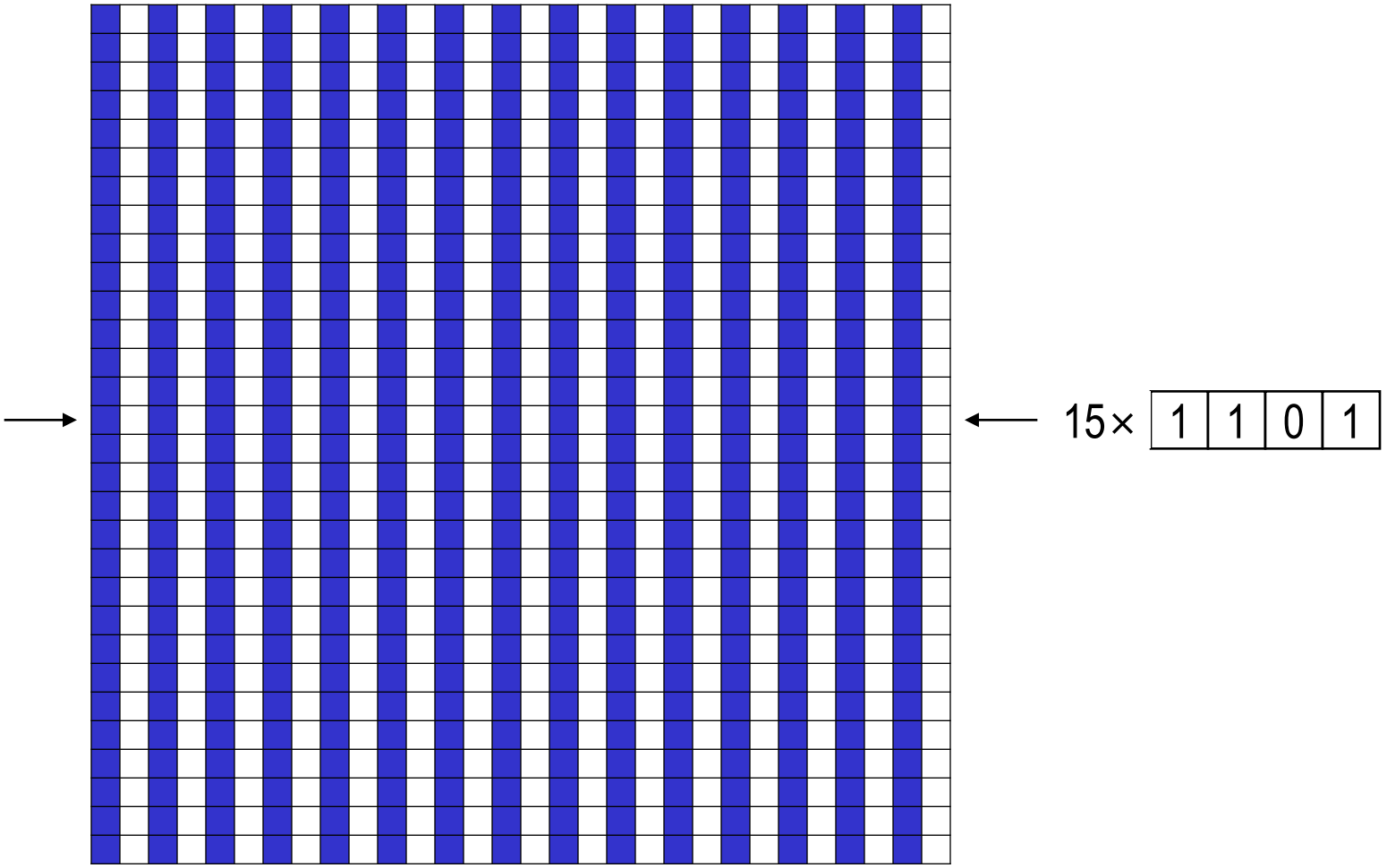
Intensity	Run length

Color :

R	G	B	Run length




0 30
0 7 1 8 0 15
1 30

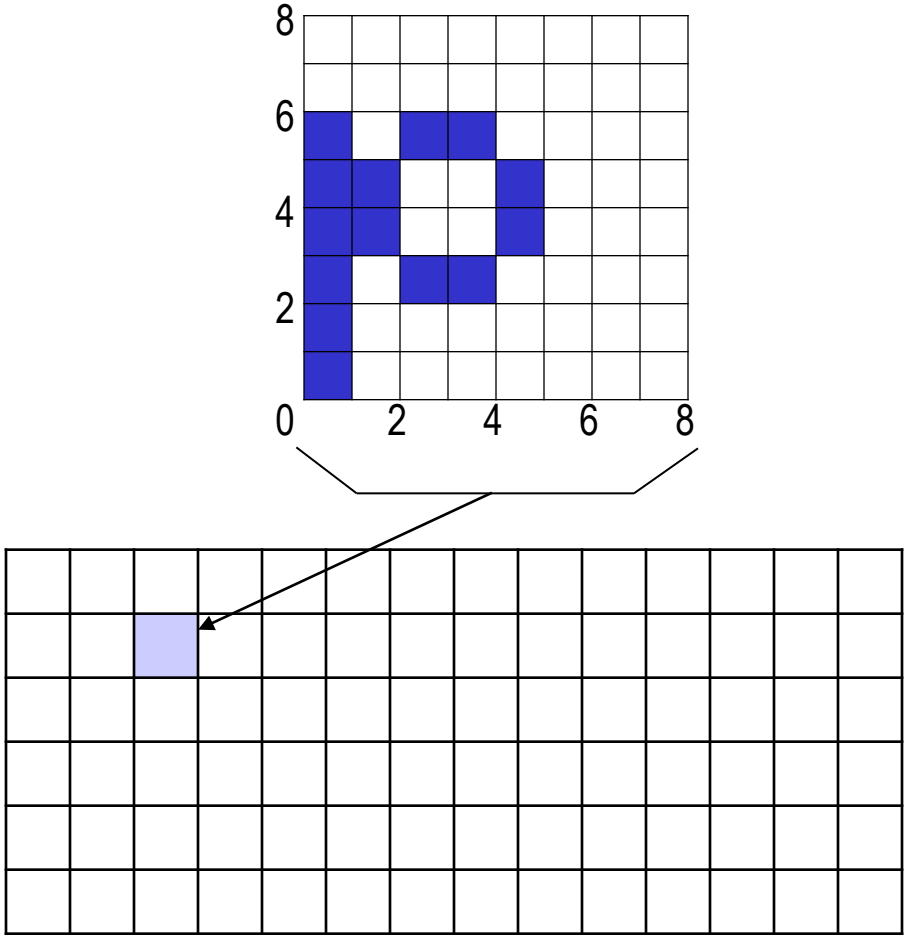

15×
1
1
0
1

Run-Length Encoding

- ✓ Data compression up to 10:1
- ✓ Saves memory
- ✓ Improves transmission time
- ✓ Useful for animated pictures

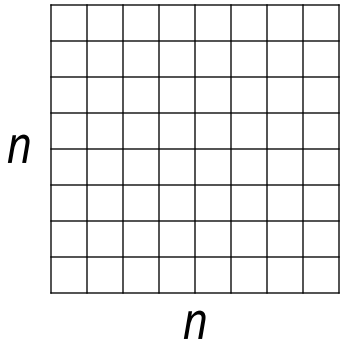
- Time-consuming to update
- Runs are stored in sequence
- Addition /deletion of lines or text is difficult
- Overhead required to encode/decode pictures
- For short runs storage requirement is higher- up to 2:1

Cell Encoding



Good for repeating patterns !!!

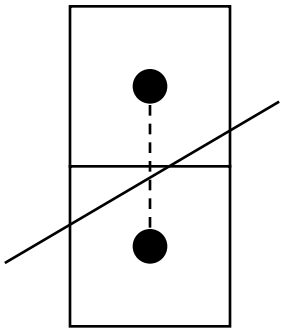
- Representing drawings



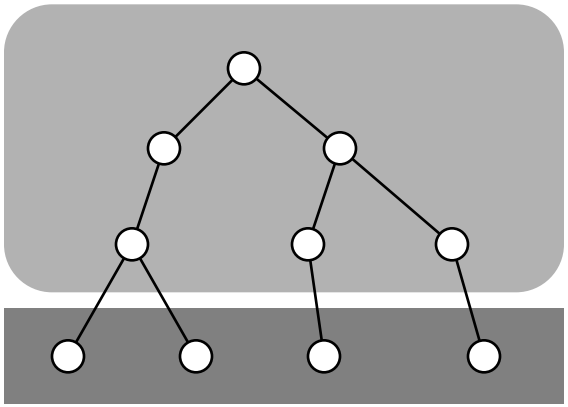
2^{n^2} patterns !!!

∴ Looks impossible for representing drawings

- A drawing consists of a list of line segments.



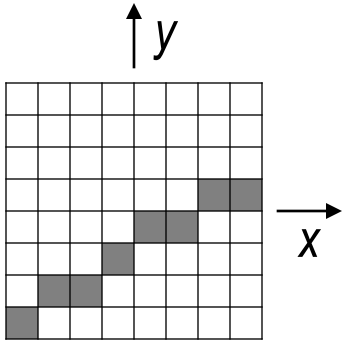
For $0 \le \text{slope} \le 1$
2^{n-1} patterns !!!

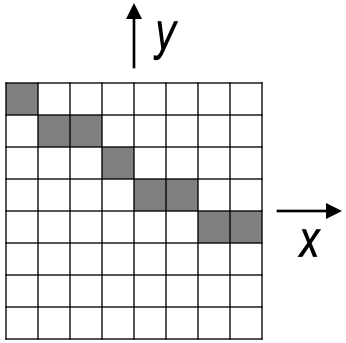


← operations

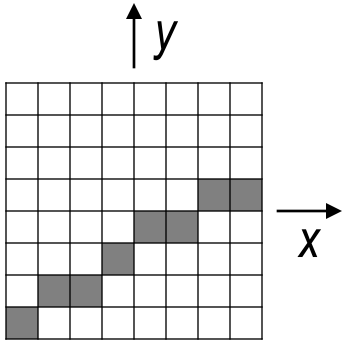
← basic primitives
||
line segment

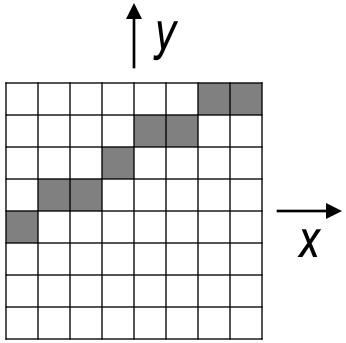
Reflection



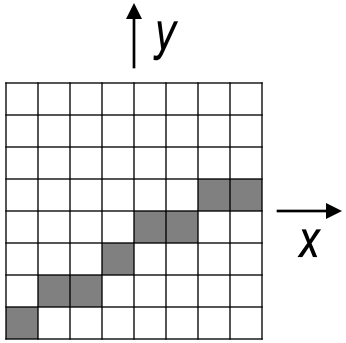


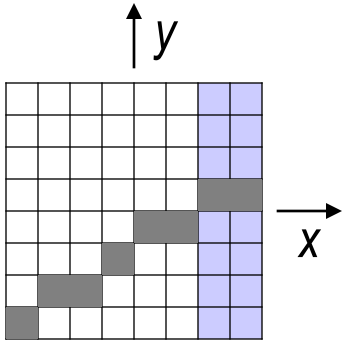
Translation

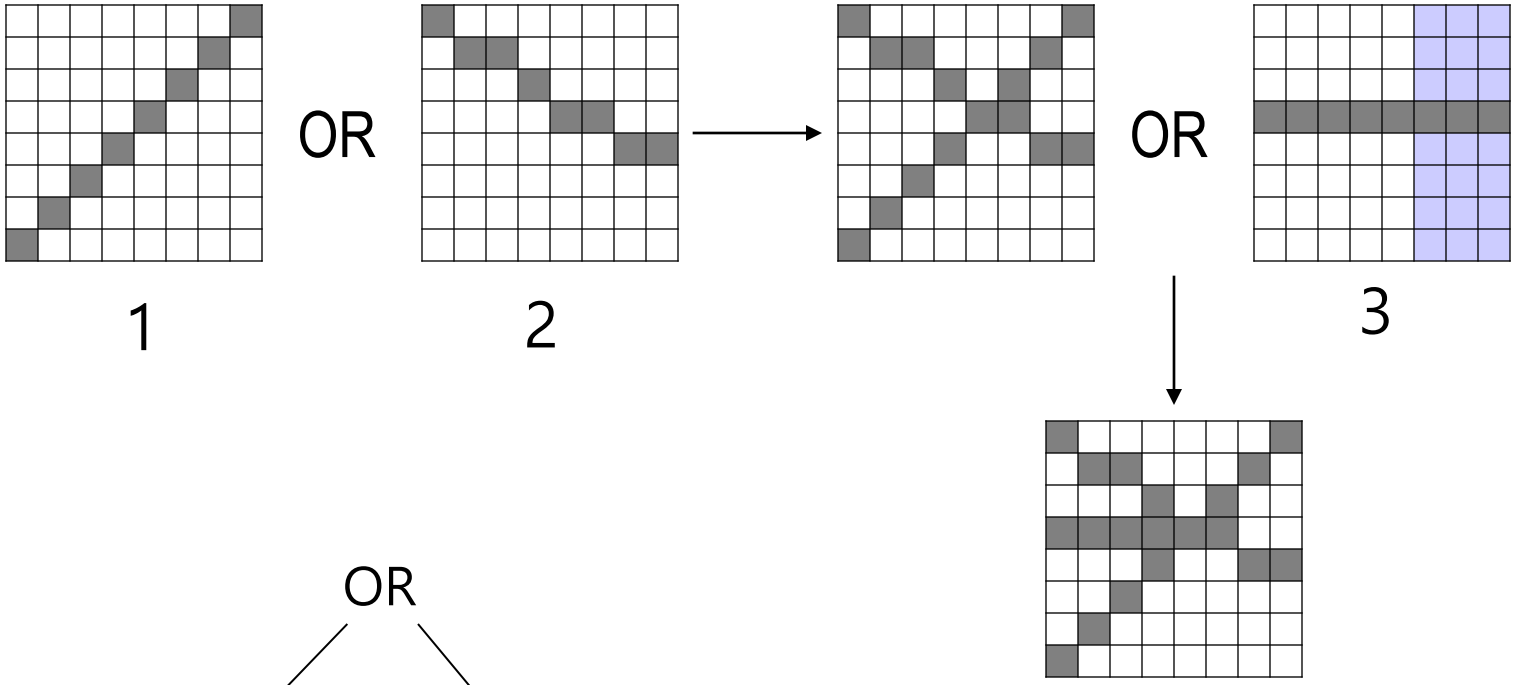


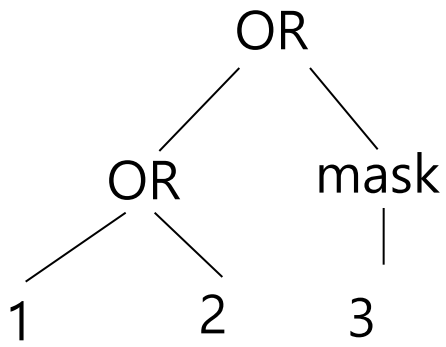


Masking

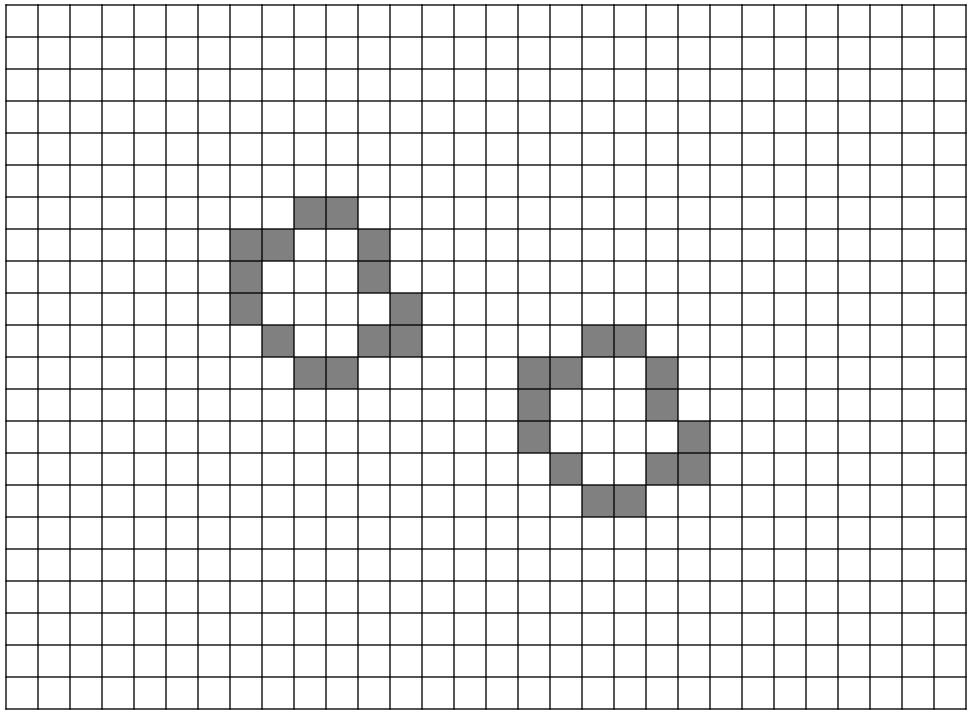





OR
OR
1
2
3


OR
OR
mask
1
2
3

Frame Buffer

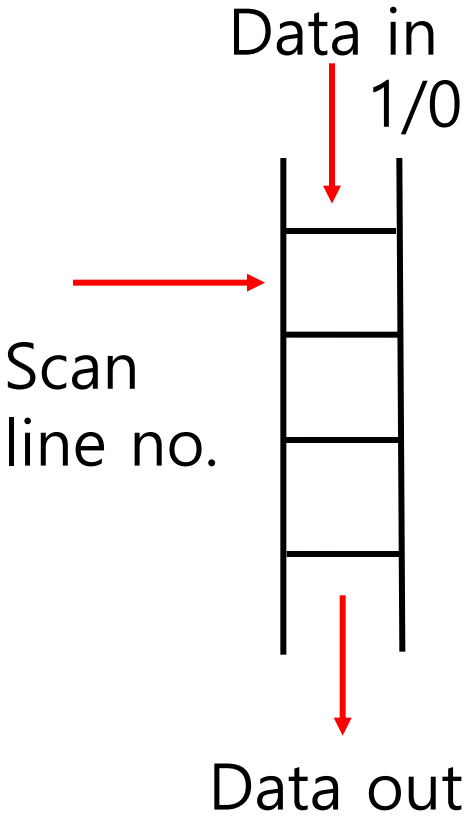


Memory - worst

Speed - best

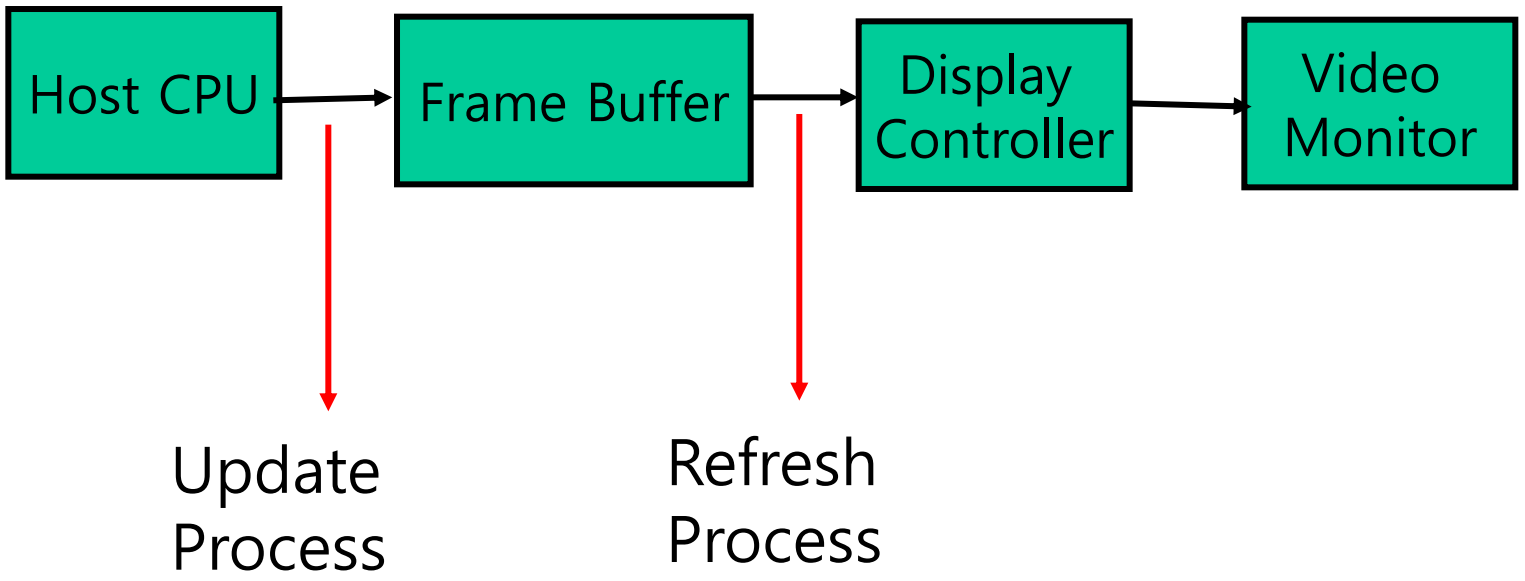
- Initial investment for scan conversion
- The complexity of a picture is independent of the speed of display processor.

Frame Buffer

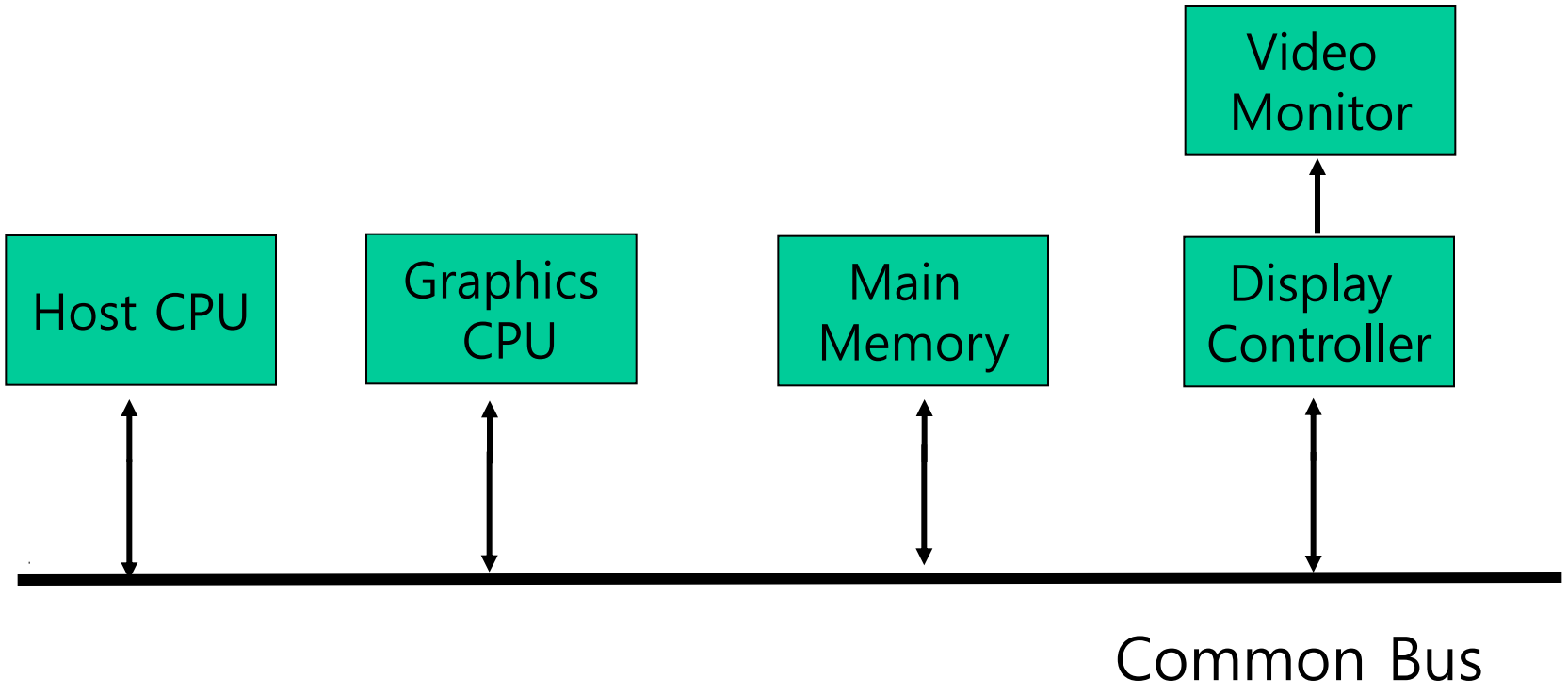


- Can use RA semiconductor memory or rotating memory- disk /drum
- Most common –shift registers
  - FIFO
- One register per pixel

Frame Buffer Graphics System

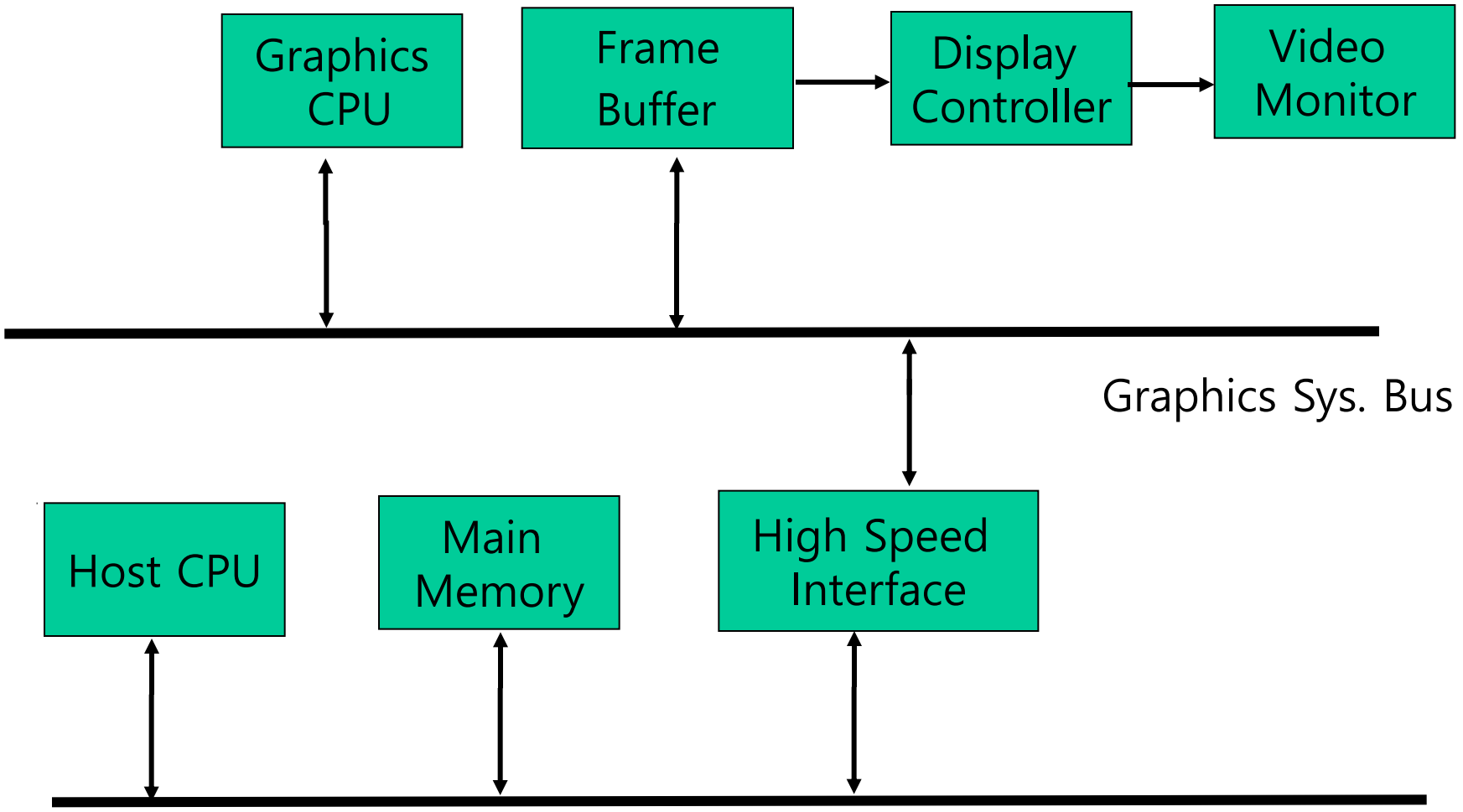


Frame Buffer Architecture (A)



- Contention occurs at memory
- Reduces overall performance

Frame Buffer Architecture (B)



- Efficient, System performance improves
- Extra architecture

Data Structure in Image Analysis:

- Levels of Image Data Representation
- Traditional Image Data Structures
  - Matrices
  - Chains
  - Topological Data Structures
  - Relational Structures
- Hierarchical Data Structures
  - Pyramids
  - Quad trees

IMAGE DATA REPRESENTATION:

1. The aim of computer visual perception is to find a relation between an input image and models of the real world.
2. During the transition from the raw input image to the model, image information becomes denser with increasing use of semantic knowledge about the interpretation of image data and more.
3. Several levels of visual information representation are defined between the input image and the model;
    * Intermediate representations (data structures)
    * Algorithms used for the creation of representations and introduction of relations between them
4. The representations can be stratified into four abstract levels

Levels of Image Data Representation:

Iconic images - lowest level; images containing original data; integer matrices with data about pixel brightness.

e.g., outputs of pre-processing operations (e.g., filtration or edge sharpening) used for highlighting some aspects of the image

Segmented images - parts of the image are joined into groups that probably belong to the same objects.

It is useful in application domain during image segmentation;

Easier to deal with noise & blur associated with inaccurate image data.

Geometric representations - hold knowledge about 2D and 3D shapes.

The quantification of a shape is very difficult but very important.

Relational models - give the ability to treat data more efficiently and at a higher level of abstraction.

Example - counting planes standing at an airport using satellite images

-A priori knowledge position of the airport (e.g., from a map)

-Relations to other objects (e.g., to roads, lakes, urban areas)

-Geometric models- planes under searching

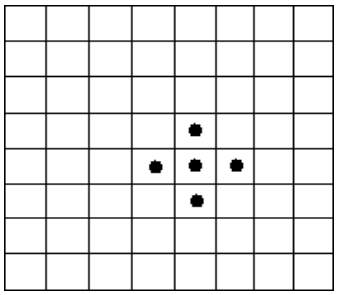
Data Structures for Image Analysis

- Traditional Image Data Structures
- Hierarchical Data Structures

Traditional Image Data Structures

- Matrices, Chains, Graphs, Tables

i) Ratio image – removes the effect of illumination variation



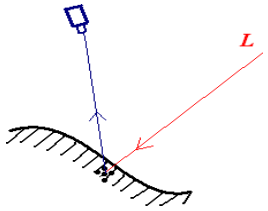
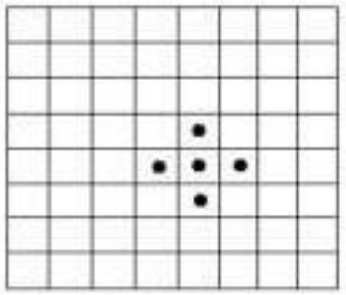


Image model:

$$I(i, j) = LR(i, j)$$

Brightness ratio between adjacent pixels:

$$r(i,j) = \frac{I(i,j+1)}{I(i,j)} = \frac{LR(i,j+1)}{LR(i,j)} = \frac{R(i,j+1)}{R(i,j)}$$



$$\text{or} \quad r(i,j) = \max\left\{\frac{I(i,j+1)}{I(i,j)}, \frac{I(i,j)}{I(i,j+1)}\right\} = r_{0,1}$$

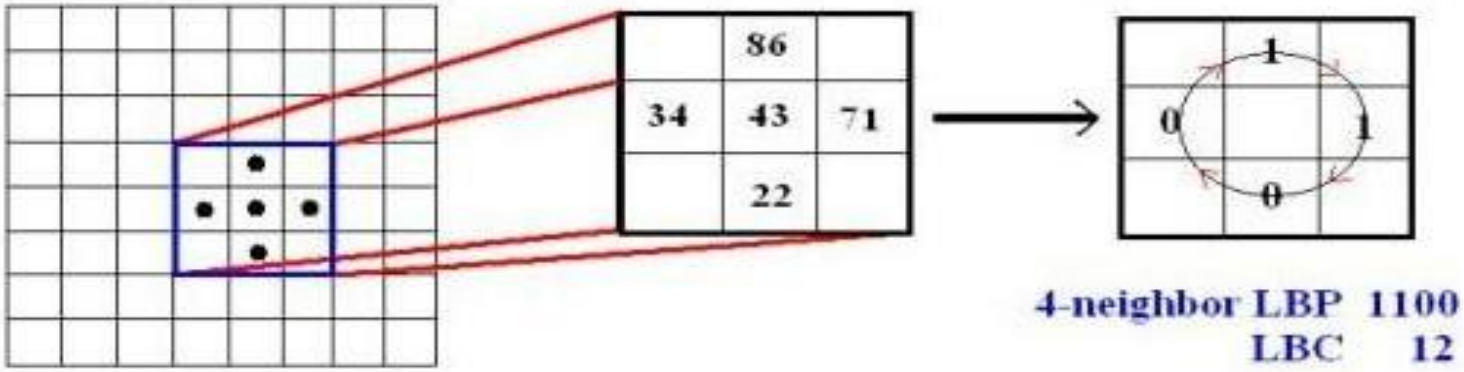
$$r(i,j) = \frac{1}{4}(r_{0,1} + r_{0,-1} + r_{1,0} + r_{-1,0})$$

Grayscale image

Ratio image

(ii) Local binary coding (LBC) image

Local binary pattern (LBP)



For 4-neighbor, the range of LBC is 0 - 16

For 8-neighbor, the range of LBC is 0 - 255

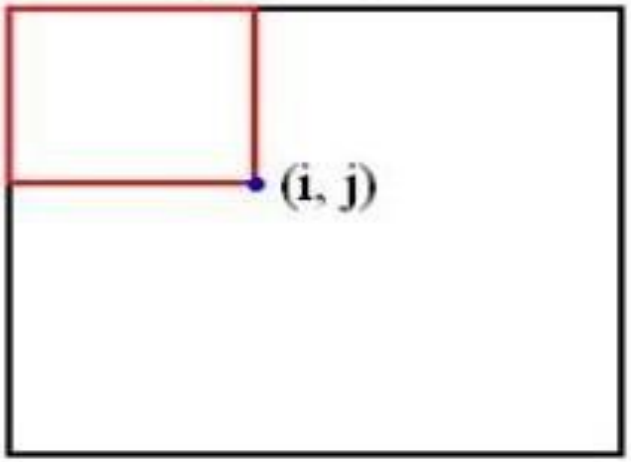
4-neighbor LBC

8-neighbor LBC

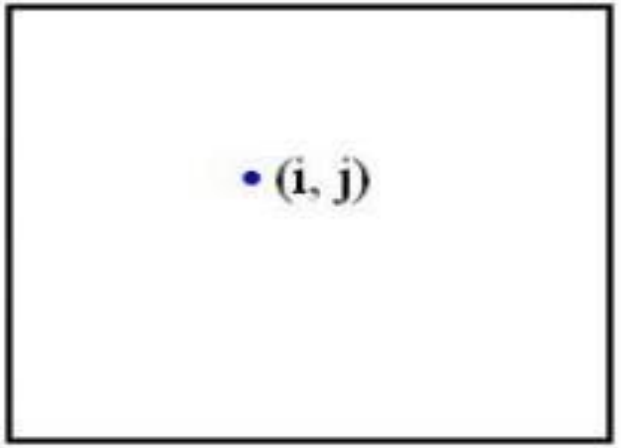
(iii) Integral image

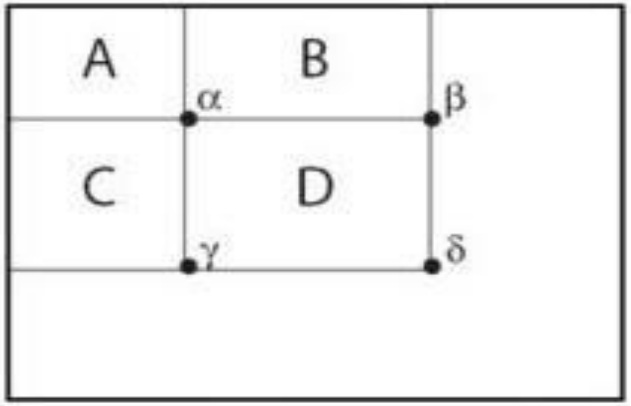
$$ii(i,j) = \sum_{k=0}^{i} \sum_{l=0}^{j} f(k,l)$$

Input image f



Integral image ii





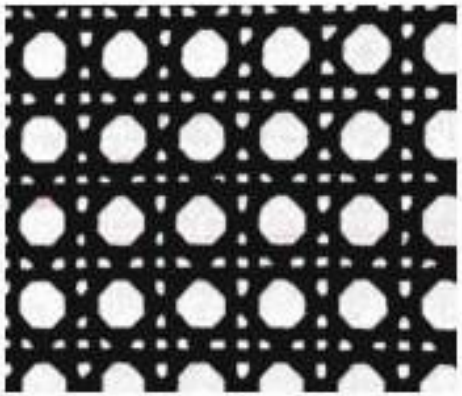
The sum of pixel values within D

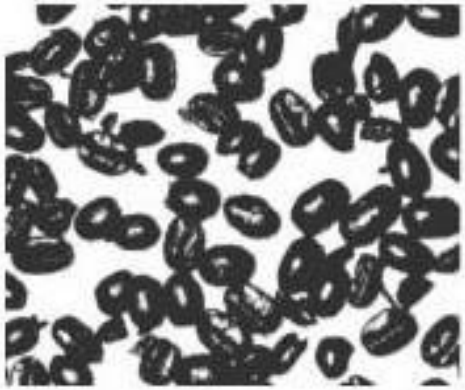
$$D_{sum} = ii(\delta) + ii(\alpha) - ii(\beta) - ii(\gamma)$$

(iv) Co-occurrence matrix (or Spatial gray-level dependence matrix)

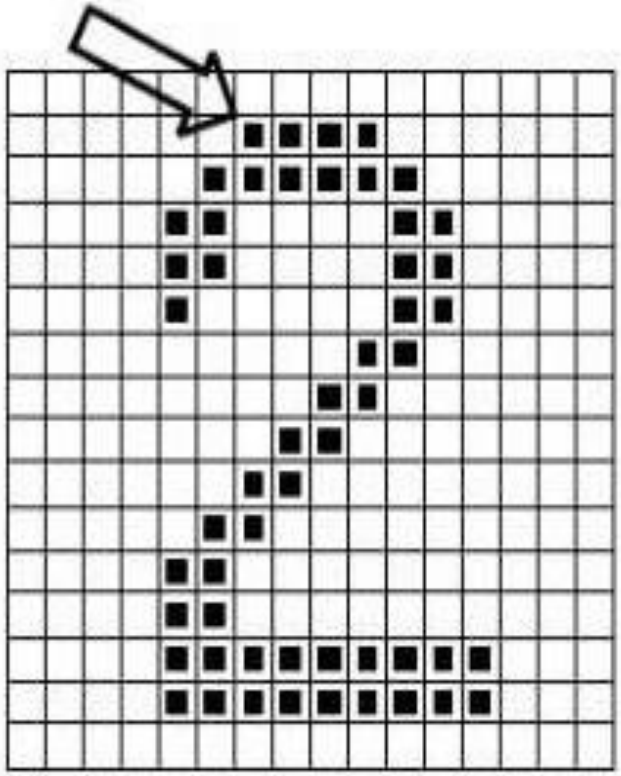
Texture analysis

Texture: closely interwoven elements





(i) **Chain code:** for description of object borders



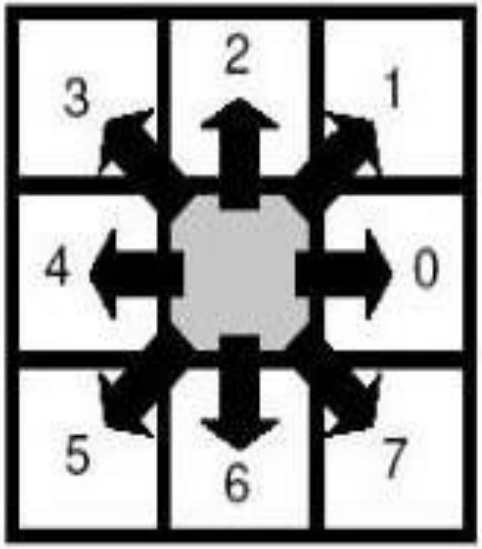
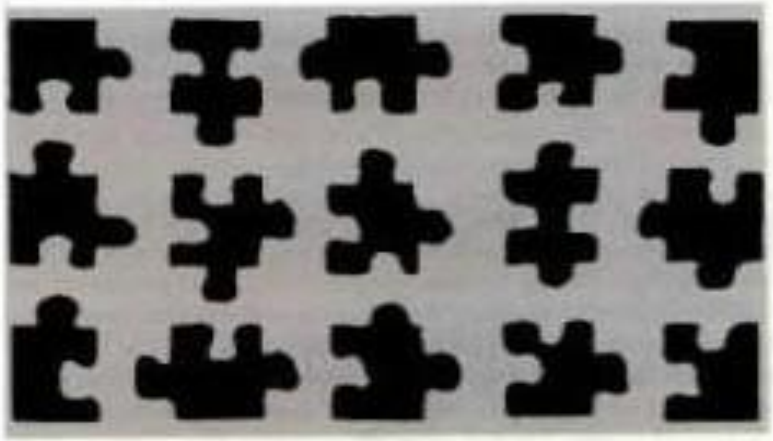


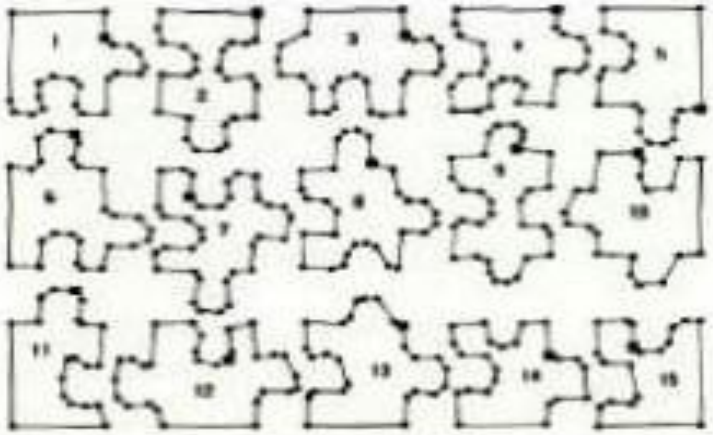
Figure 4.3: An example chain code; the reference pixel starting the chain is marked by an arrow: 00077665555 556600000006444444422211111122 34445652211.

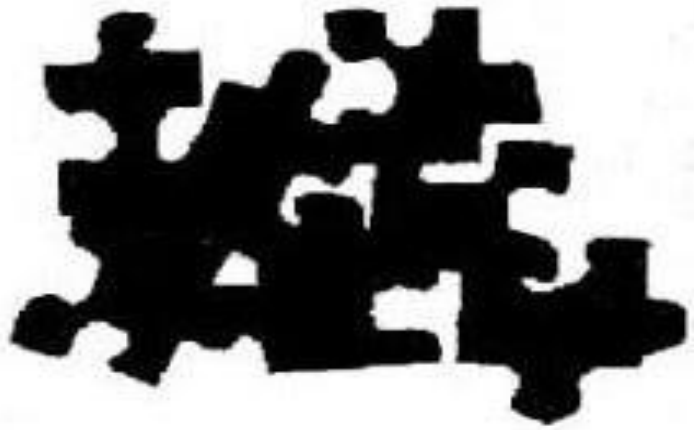
(ii) Attributed strings

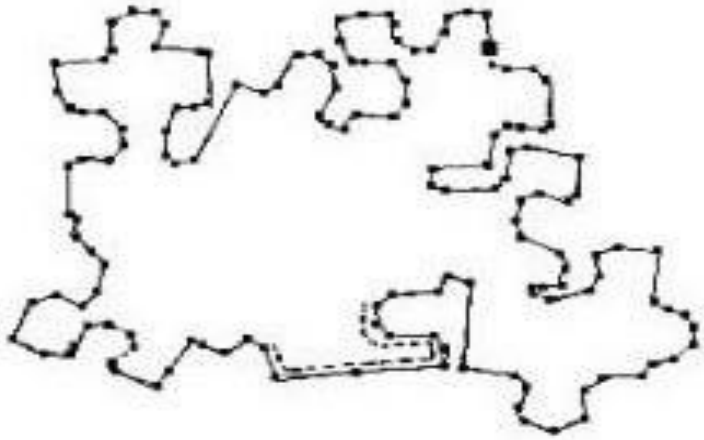
$$S = (p_1, l_1, \theta_1)(p_2, l_2, \theta_2) \cdots (p_n, l_n, \theta_n)$$

where p : polyline, l : length, θ : direction









String matching

Scene string: $S^S = (p_1, l_1, \theta_1)^S (p_2, l_2, \theta_2)^S \cdots\cdots (p_m, l_m, \theta_m)^S$

Model string: $S^M = (p_1, l_1, \theta_1)^M (p_2, l_2, \theta_2)^M \cdots (p_n, l_n, \theta_n)^M$

Calculate the cost of transforming S^S to S^M

The larger the cost the larger the difference between the two strings, and in turn the larger the difference between the two shapes described by the strings.

String transformation through editions

Types of edition: **insert, delete, change**

Define the cost functions of editions

$$\mathrm{Cost}_{\mathrm{insert}}[(p_j, l_j, \theta_j)^M], \quad \mathrm{Cost}_{\mathrm{delete}}[(p_i, l_i, \theta_i)^S]$$

$$\mathrm{Cost}_{\mathrm{change}}[(p_i, l_i, \theta_i)^S, (p_j, l_j, \theta_j)^M]$$

The total cost of string transformation

$$\mathrm{Cost}_{\mathrm{transform}} = \sum_i \mathrm{cost}_i$$

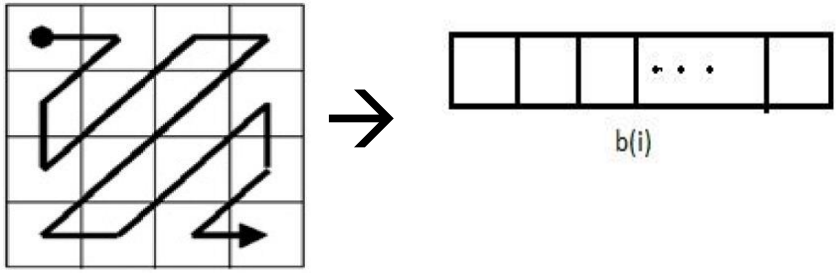
Object recognition by

$$S_*^M = \arg\min_i \mathrm{Cost}_{\mathrm{transform}}(S_i^M)$$

SCAN

It is **order** in which each element of the array is accessed exactly once

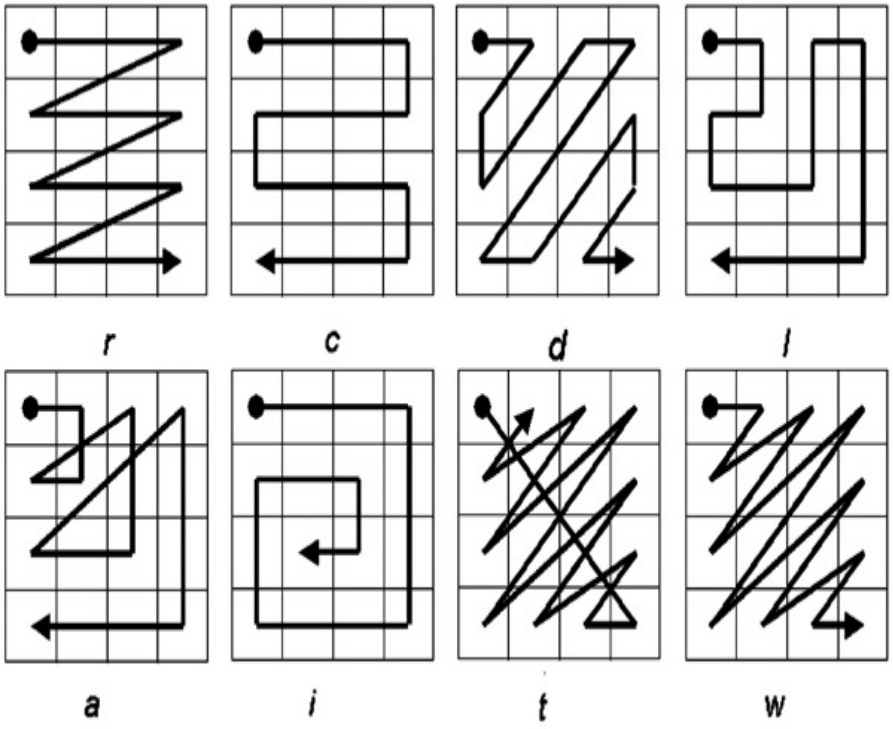
It represents a family of formal **language-based** 2-D spatial **accessing approaches**



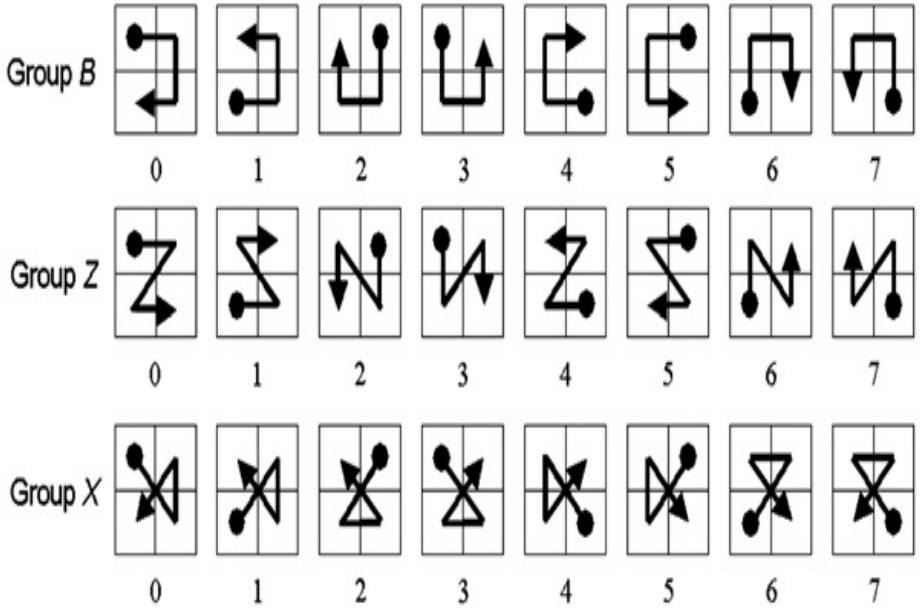
SCAN language has

1) a set of basic **scan patterns**
2) a set of **transformations** of scan patterns, and
3) a set of **rules** to recursively compose simple scan patterns to obtain complex scan patterns.

SCAN…

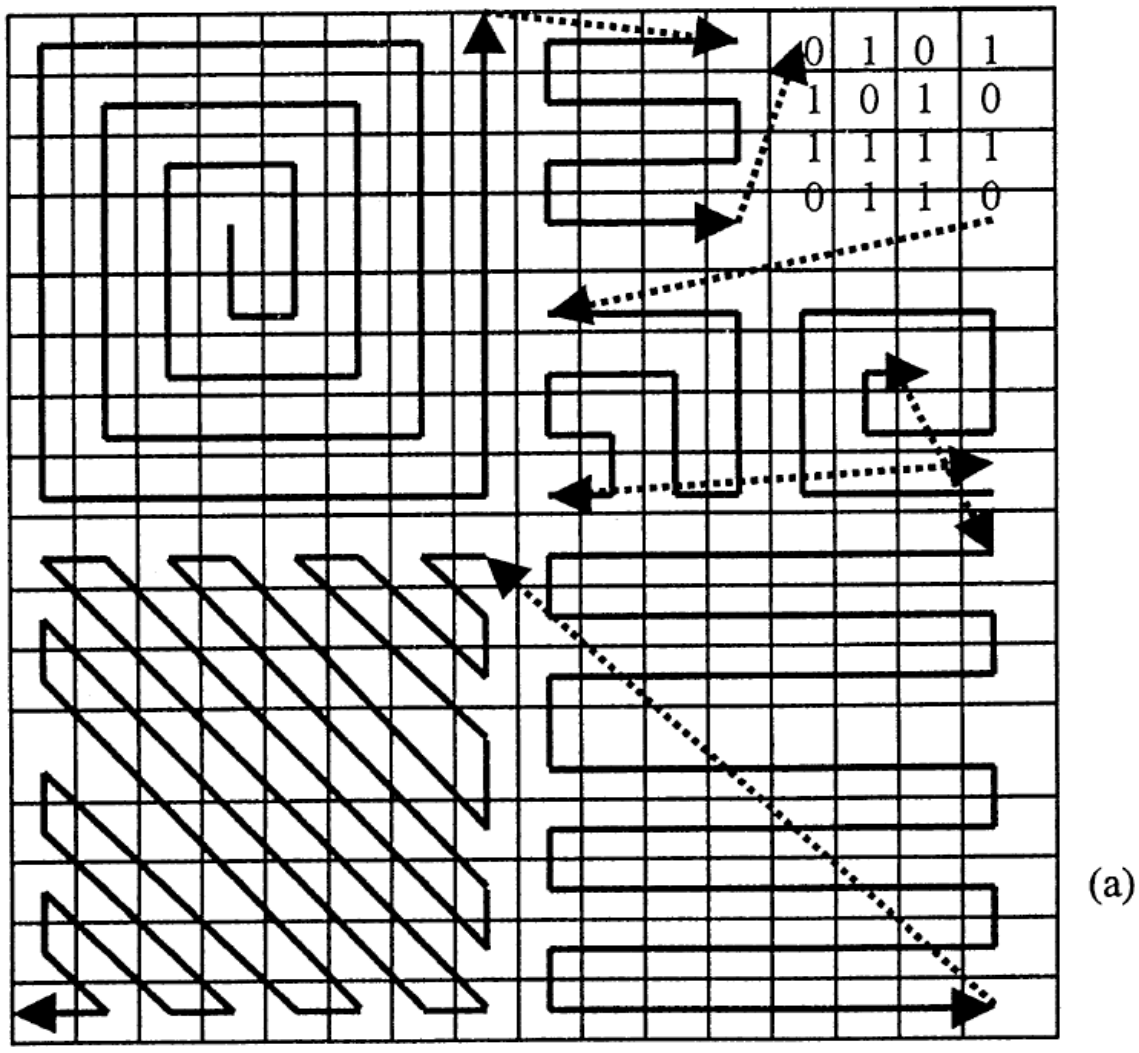


Basic Scan Patterns

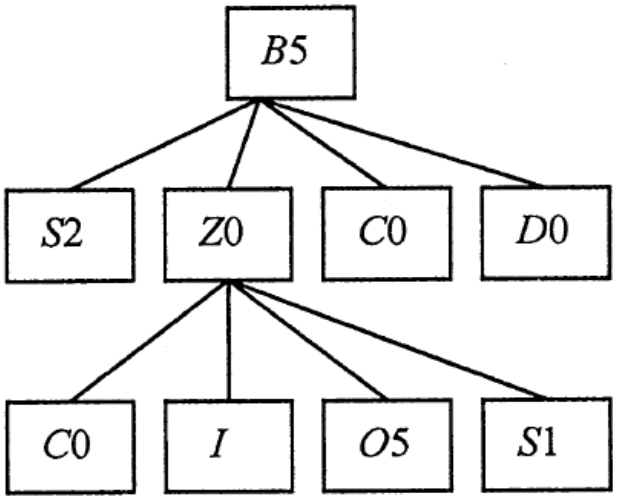


Transformations with partition

SCAN…



(a)



(b)

4.3. Hierarchical data structures

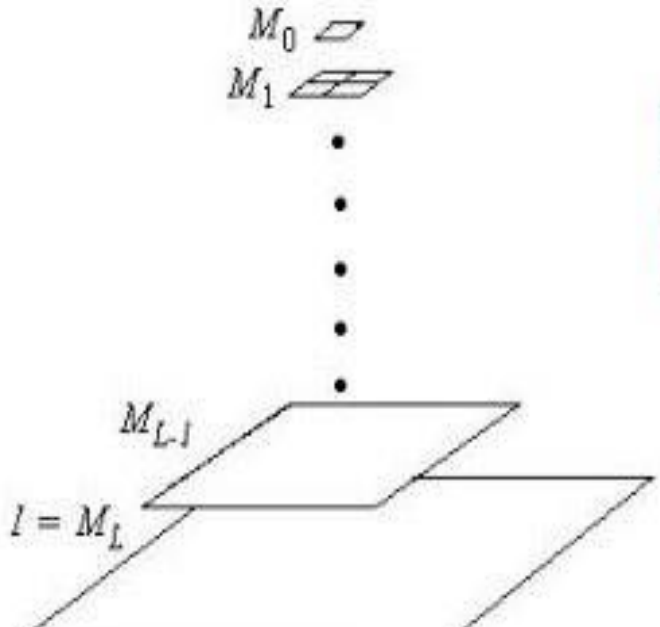
4.3.1. Pyramids

Matrix pyramid: a sequence of images

$$\{M_L, M_{L-1}, \cdots, M_0\}$$

M_{i-1} is derived from M_i by reducing the resolution by 1/2

M_0 corresponds to one pixel only



Multigrid processing

Tree pyramid: Let 2^L : the size of an image

$P_k = \{(k,i,j) \mid i,j \in [0, 2^k - 1]\}$, $k \in [0, L]$:

the set of nodes at level k

$F : P_k \to P_{k-1}$: mapping the nodes in P_k to those in P_{k-1}. $F(k,i,j) = (k-1,\ i \text{ div } 2,\ j \text{ div } 2)$

div: whole-number division(?)

$V : P \to Z$ defines the values of nodes, e.g., average, maximum, minimum

Z: a set of brightness levels

Leaf nodes have pixel brightness values

e.g., $k=0, \Rightarrow P_0=\{(0,i,j)\mid i,j\in[0,0]\}=\{(0,0,0)\}$

$k=1, \Rightarrow i,j\in[0,2^k-1]=[0,1]$

From $P_k=\{(k,i,j)\mid i,j\in[0,2^k-1]\}$

$P_1=\{(1,i,j)\mid i,j\in[0,1]\}$

$=\{(1,0,0),(1,0,1),(1,1,0),(1,1,1)\}$

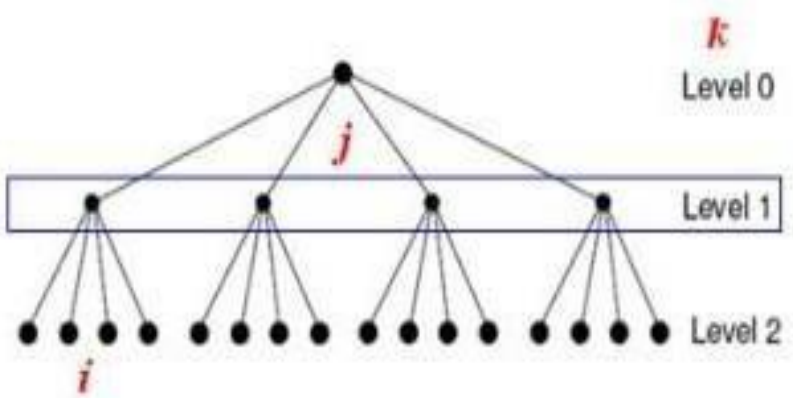
From $F(k,i,j)=(k-1,\ i \text{ div } 2,\ j \text{ div } 2)$

$F(1,0,0)=(0,0,0),$

$F(1,0,1)=(0,0,0),$

$F(1,1,0)=(0,0,0),$

$F(1,1,1)=(0,0,0)$



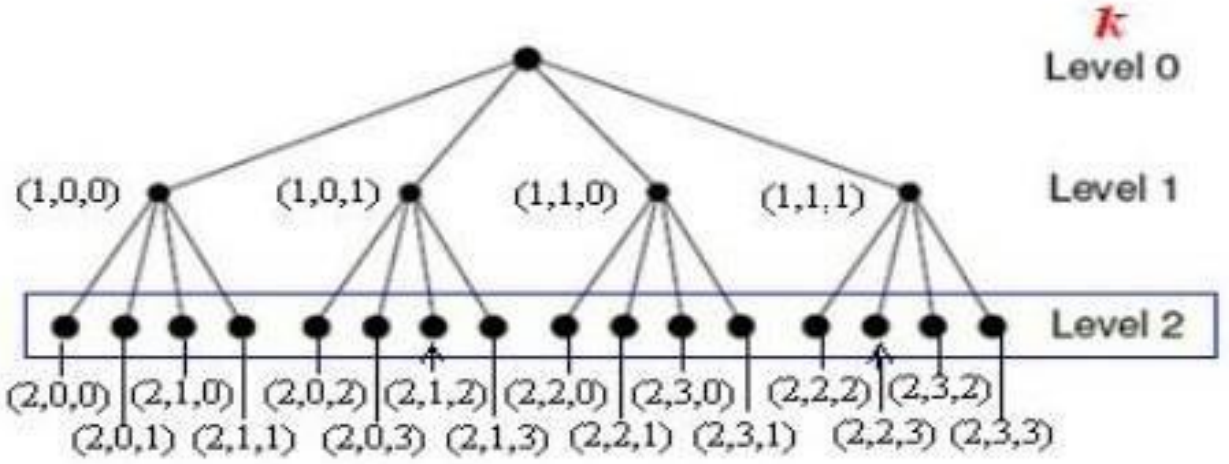
$k = 2, \Rightarrow i, j \in [0, 2^k - 1] = [0, 3]$

From $P_k = \{(k, i, j) \mid i, j \in [0, 2^k - 1]\}$

$$\Rightarrow P_2 = \{(2,0,0),(2,0,1),(2,0,2),(2,0,3),(2,1,0),(2,1,1),$$
$$(2,1,2),(2,1,3),(2,2,0),(2,2,1),(2,2,2),(2,2,3),$$
$$(2,3,0),(2,3,1),(2,3,2),(2,3,3)\}$$

$F(k, i, j) = (k-1,\ i \text{ div } 2,\ j \text{ div } 2)$

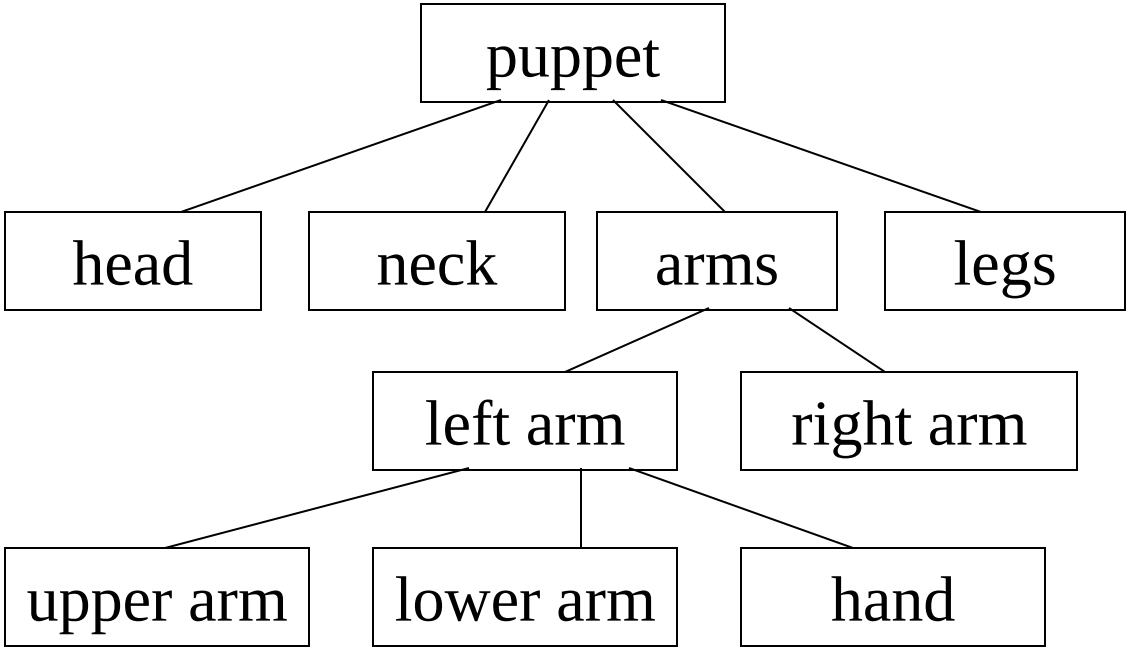
e.g., $F(2,1,2) = (1,0,1)$, $F(2,3,1) = (1,1,1)$



Hierarchical modeling 1

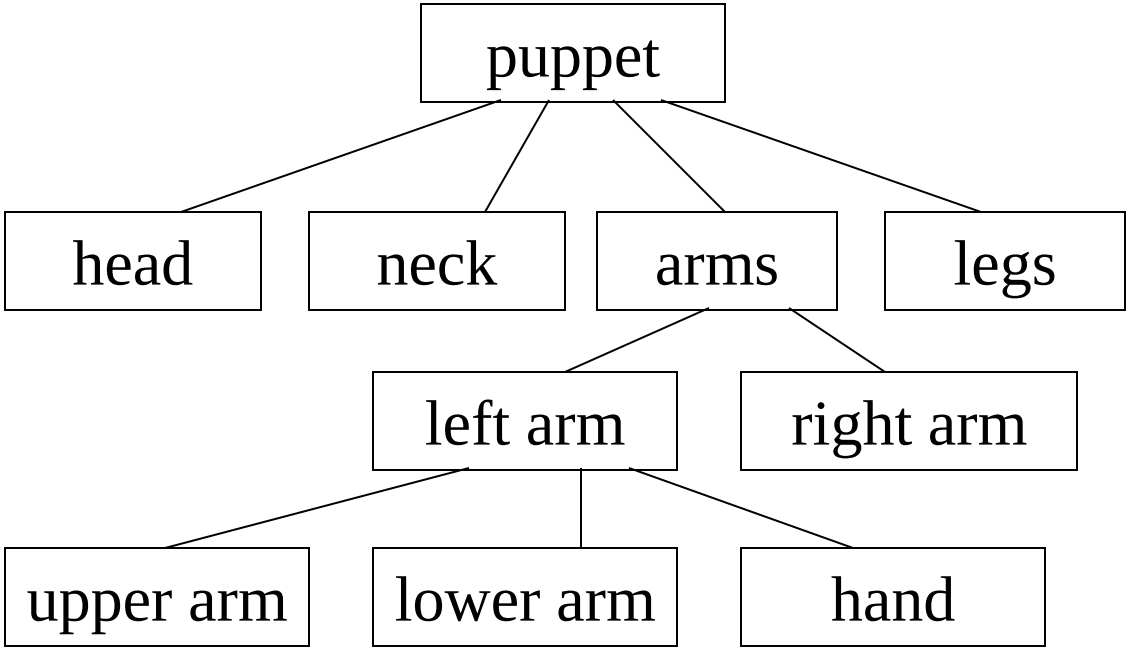
ComplexObject::= Composition of Objects

Object::= SimpleObject or ComplexObject



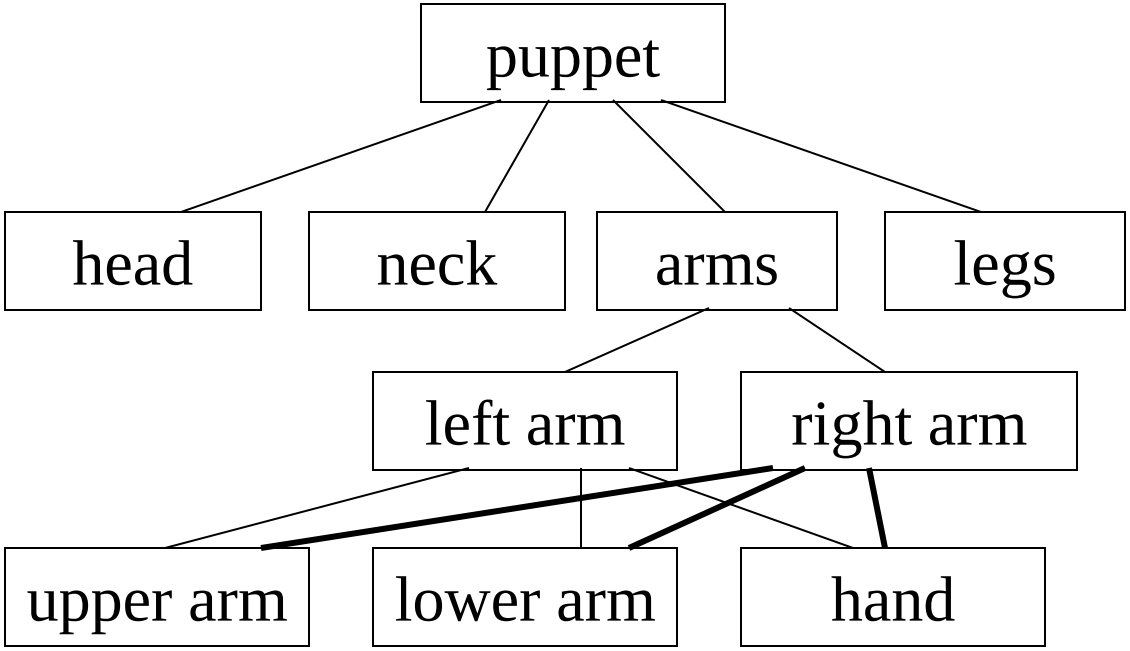
Hierarchical modeling 2

Hierarchical model: tree structure …



Hierarchical modeling 3

Hierarchical model: tree structure or directed, acyclic graph



Hierarchical modeling 4

Scene graph:

Leaf: primitive object

geometric object, possibly parametrised

Composite node: instruction for composition (usually union)

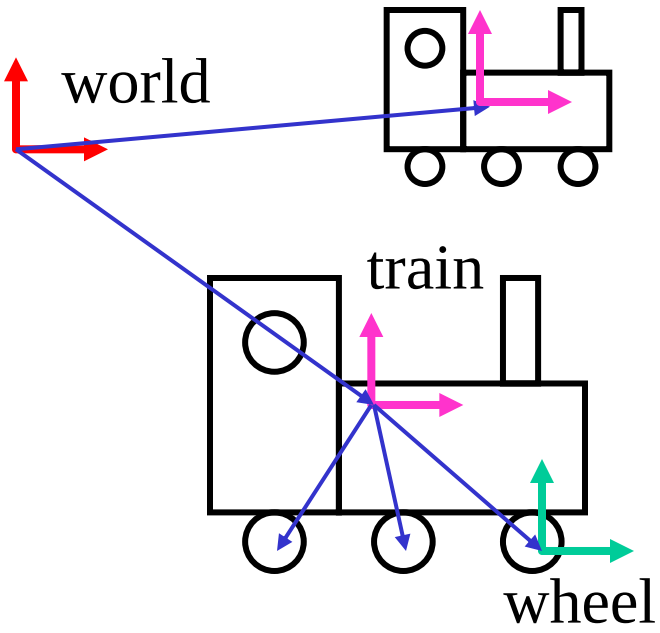
Leaves, nodes and/or edges: transformations + other data

Implementation: data (read model, generic engine) or

code (translate into statements & calls)

Hierarchical modeling 5

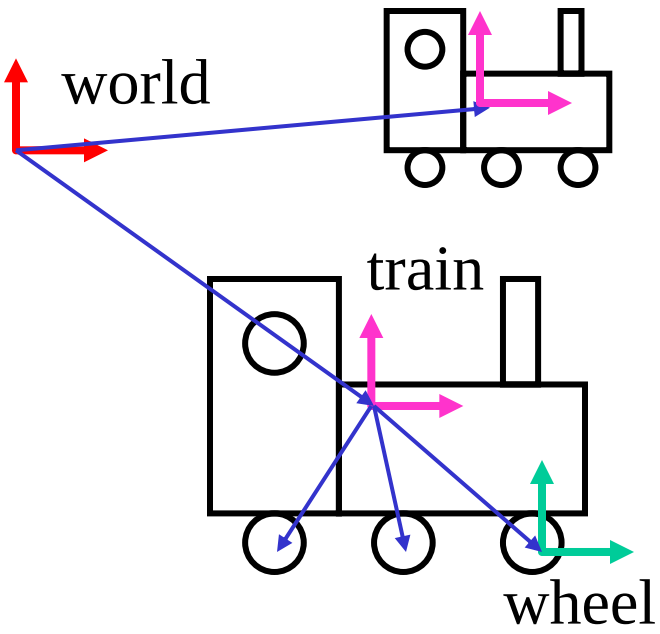
Local coordinates: coordinates of node



world coordinates
train coordinates
wheel coordinates

Hierarchical modeling 6

Implementation: Many variations are possible



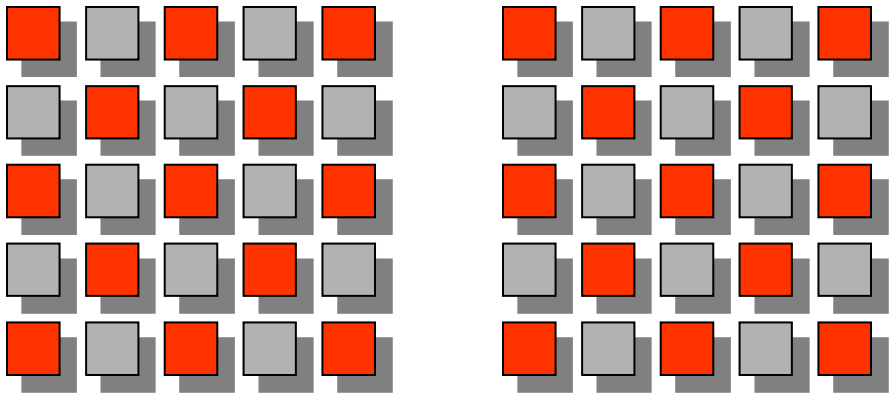

```
DrawWorld
    DrawTrain(P1, S1);
    DrawTrain(P2, S2);

DrawTrain(P, S);
    Translate(P); Scale(S);
    DrawChimney(); …
    DrawWheel(W1); DrawWheel(W2); DrawWheel(W3);
    Scale(1/S); Translate(-P);

DrawWheel(W);
    Translate(W); DrawCircle(radius); Translate(-W);
```

Hierarchical modeling 7

Use structure of model to structure implementation:

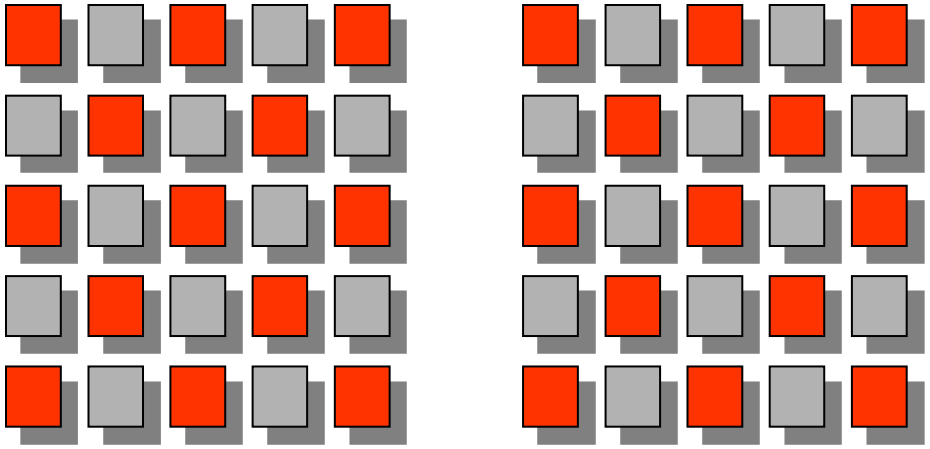
- Hierarchy (use classes, procedures and functions)
- Regularity (use loops)
- Variation (use conditions)

How to code for this picture?



Hierarchical modeling 8

- Hierarchy
(use classes, procedures and functions)
- Regularity (use loops)
- Variation (use conditions)



```
DrawTwoGrids
    DrawGrid;
    Translate(7,0);
    DrawGrid;

DrawGrid
    for i := 1 to 5 do
    for j := 1 to 5 do
        DrawCell(i, j, (i+j) mod 2 = 0);

DrawCell(x, y, use_red)
    SetColor(dark_grey);
    DrawRect(x+0.2,y, 0.7, 0.7);
    if use_red then SetColor(red)
       else SetColor(light_grey);
    DrawRect(x, y+0.2, 0.7, 0.7);
```