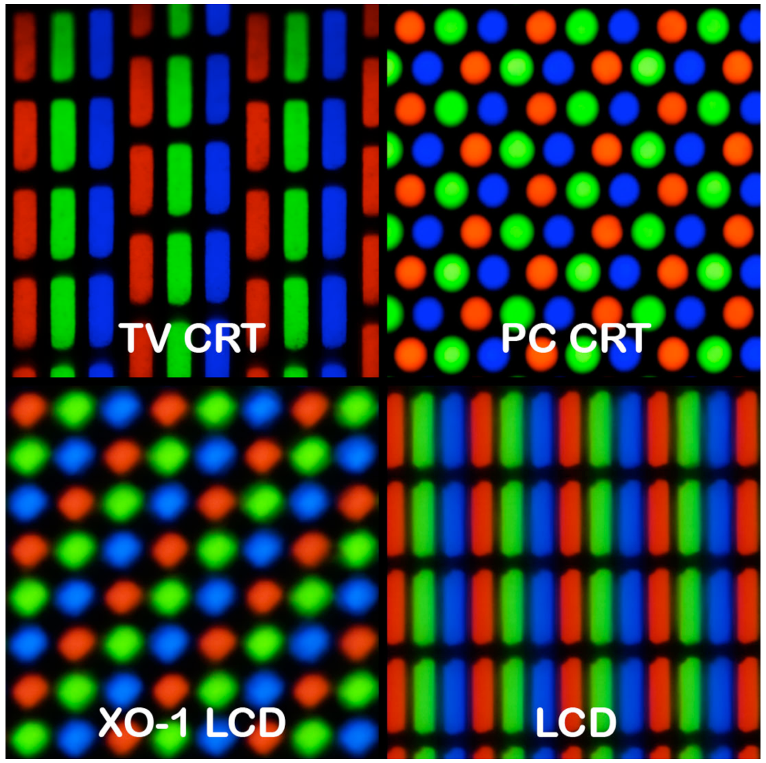
# Computer Graphics 2020

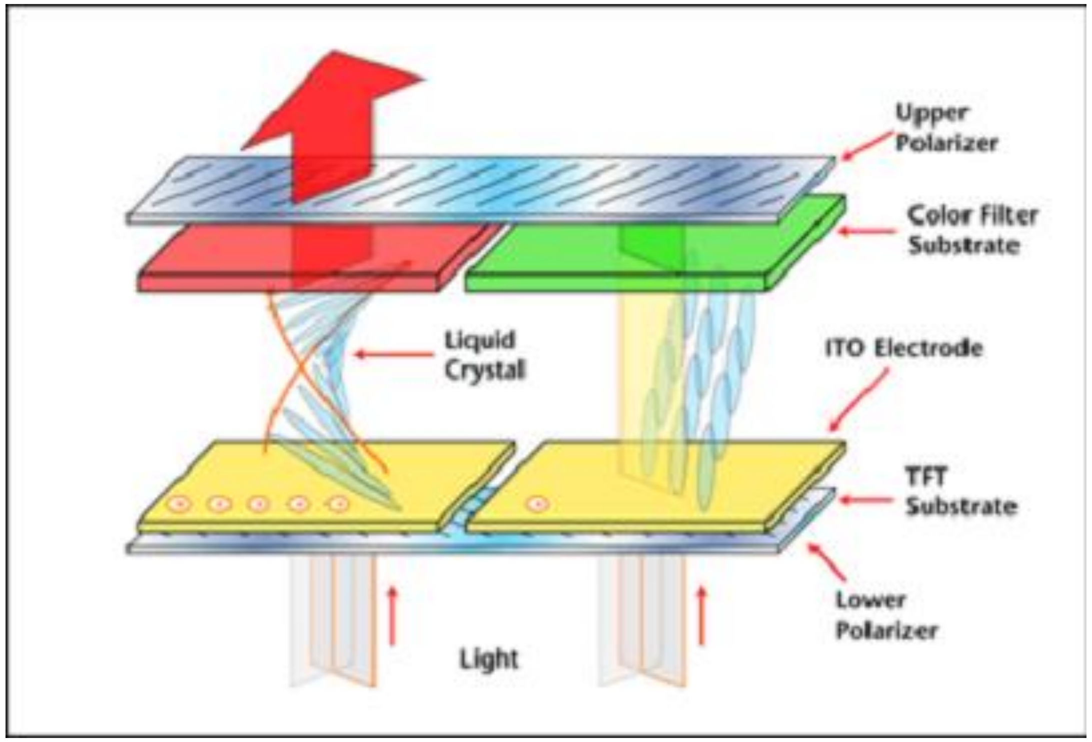
# 2. 2D Graphics Algorithms

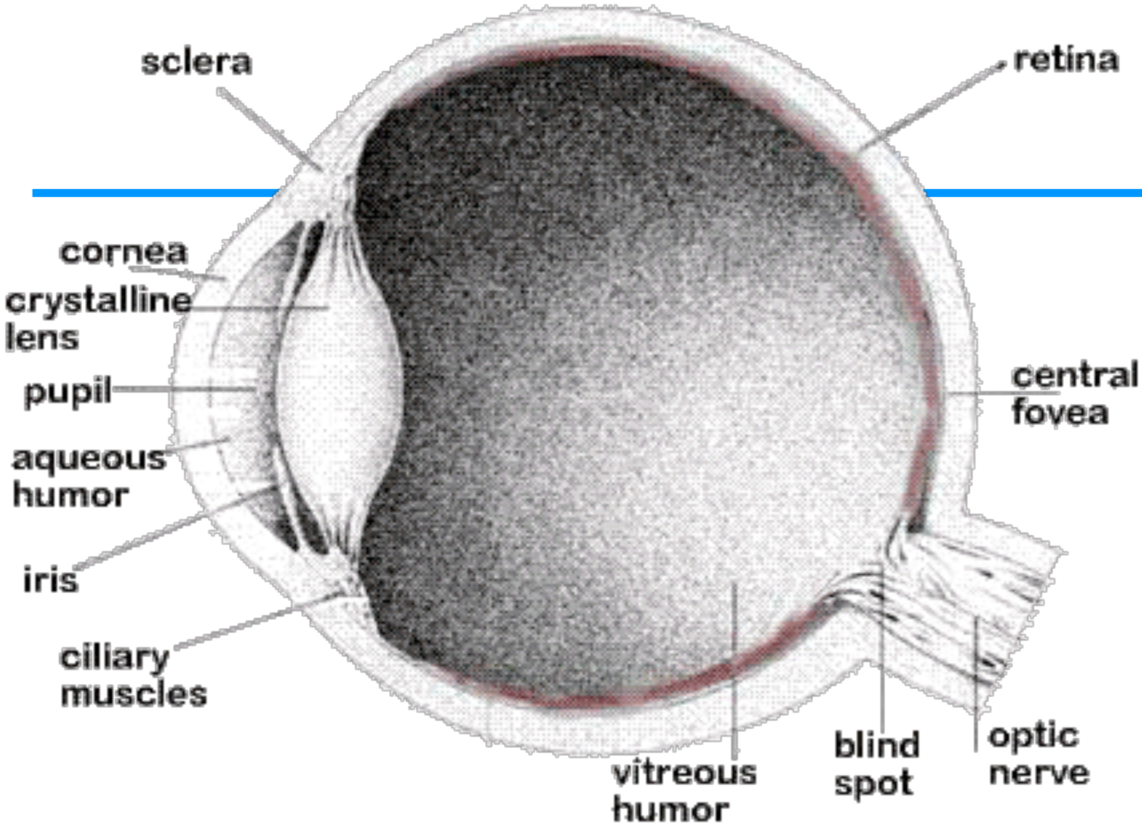
Hongxin Zhang
State Key Lab of CAD&CG, Zhejiang University

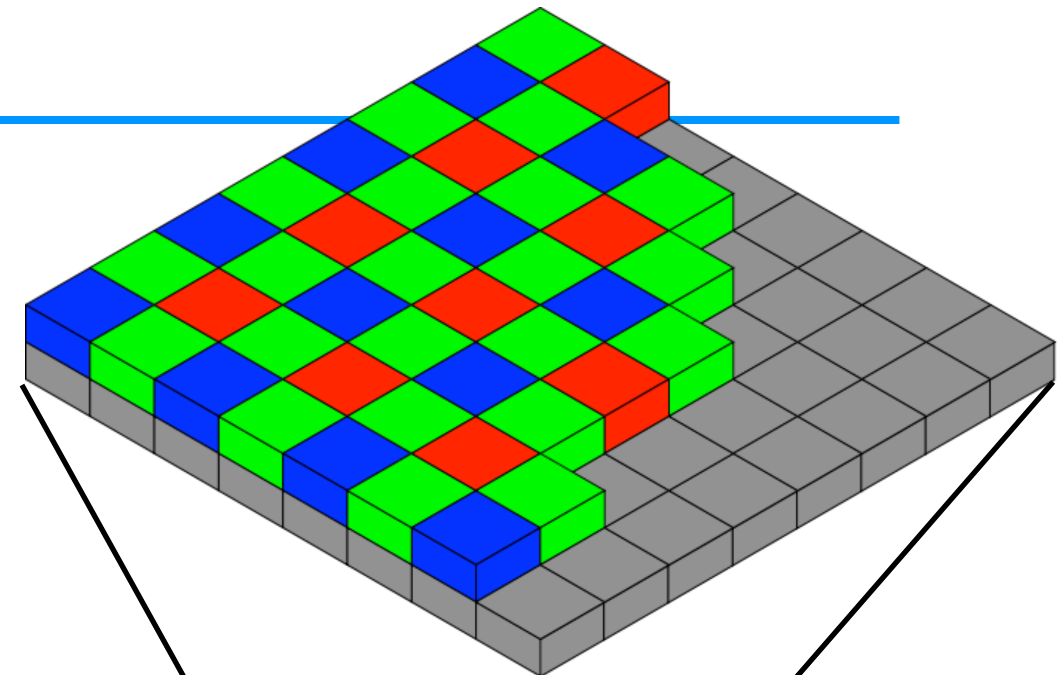
2020-09-21

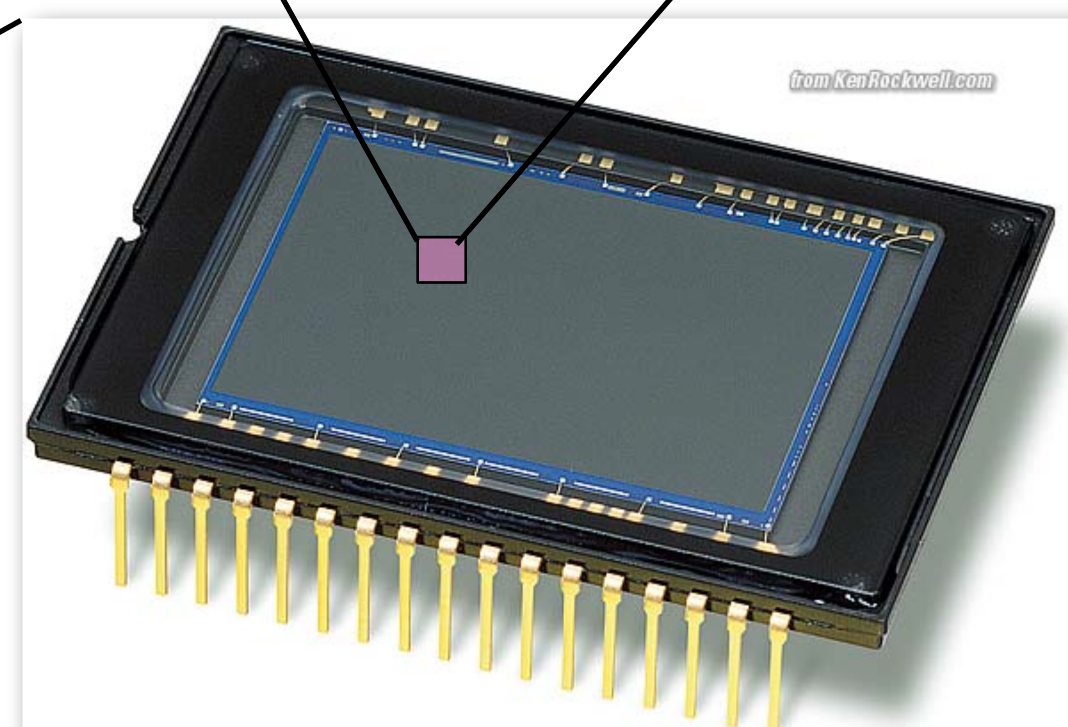
# Screen - Linear Structure



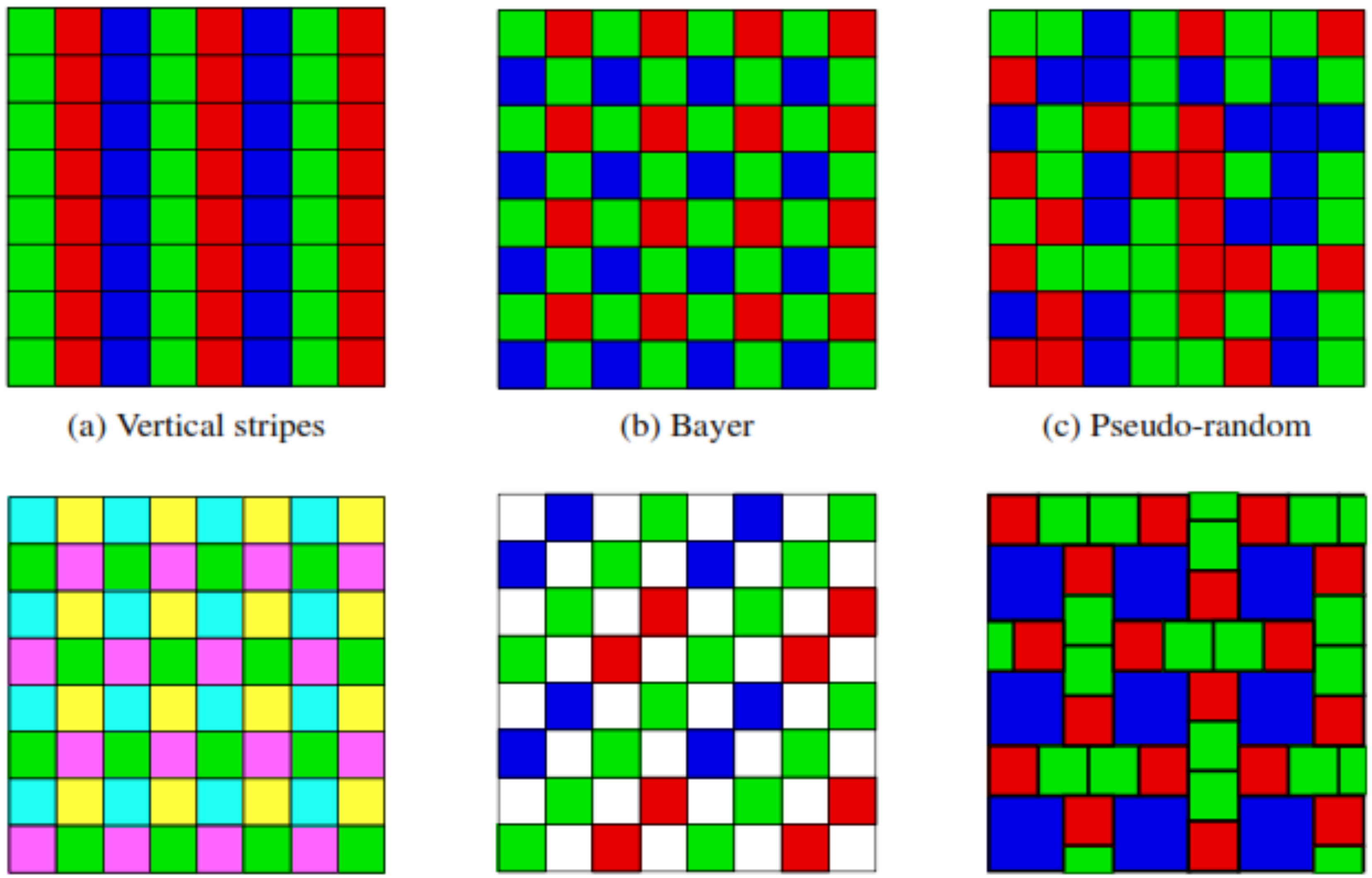



# Nikon D40 Sensors

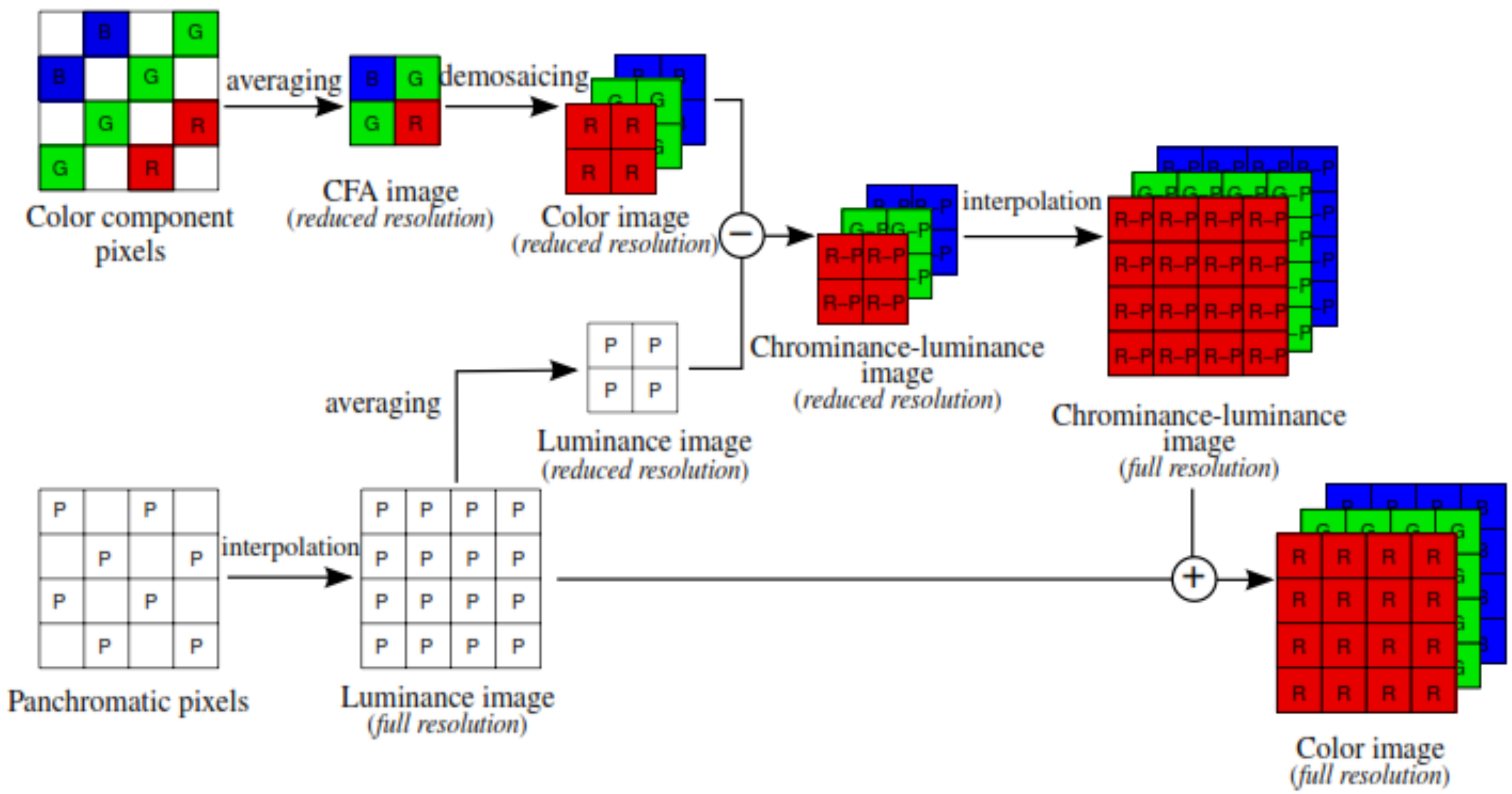
# RGBW Camera Sensor



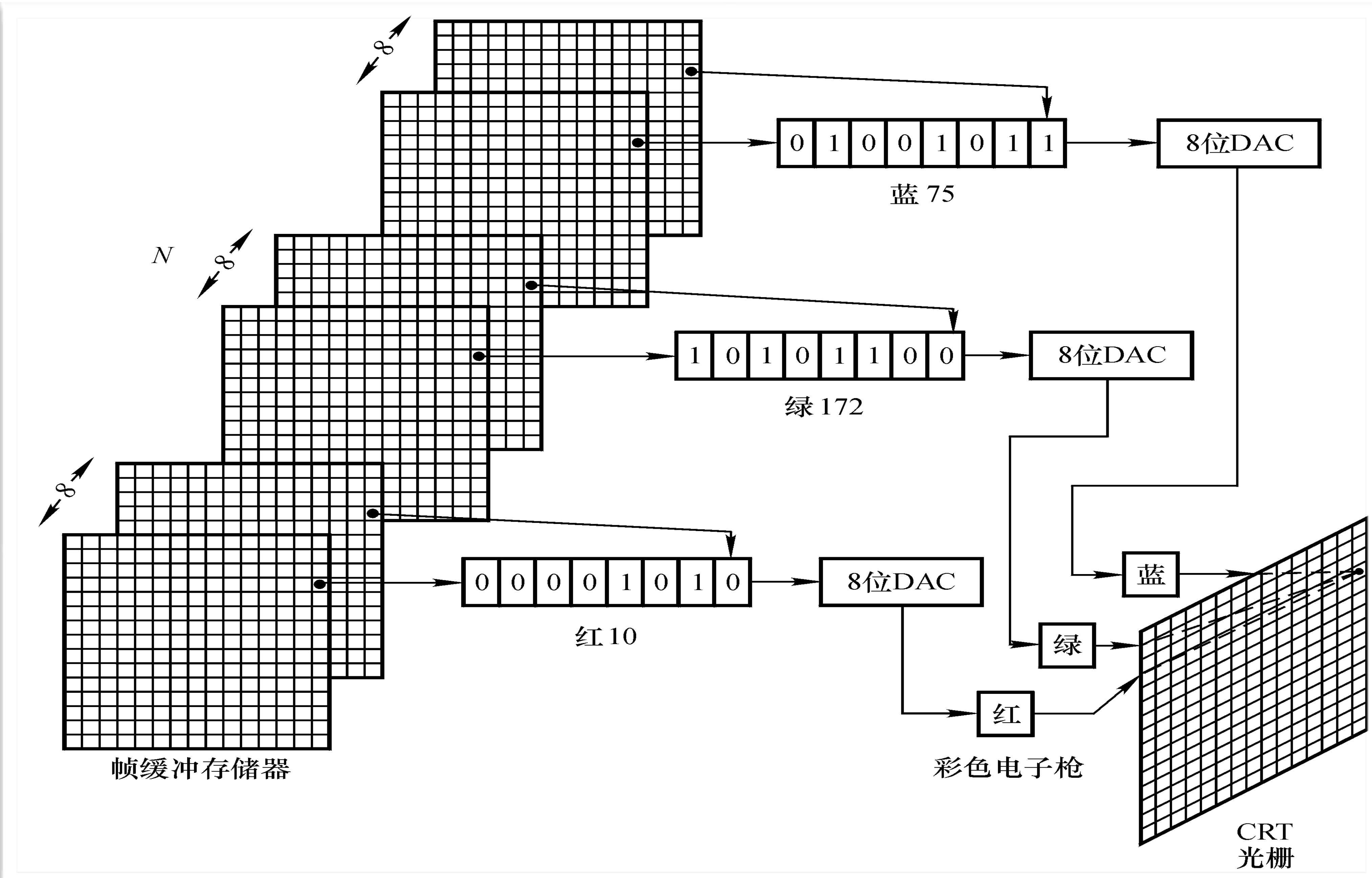
(a) Vertical stripes (b) Bayer (c) Pseudo-random

(d) Complementary colors (e) "Panchromatic", or CFA2.0 (Kodak) (f) "Burtoni" CFA

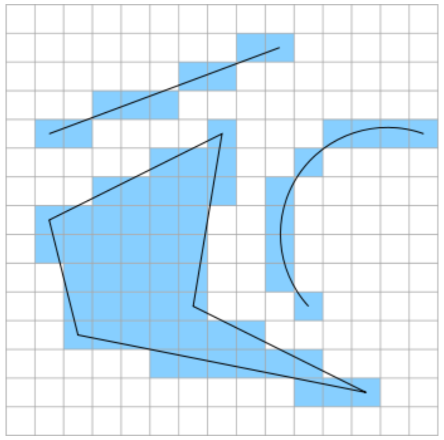
# RGBW Camera Sensor



# Rasterization

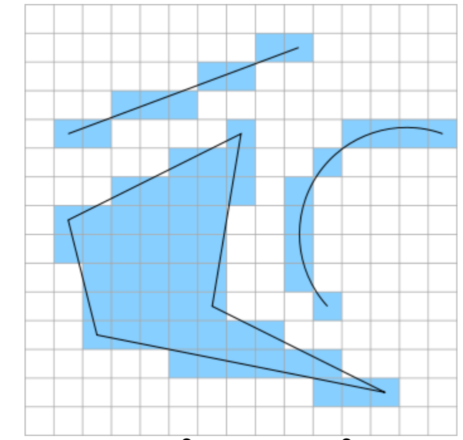


# Rasterization



- The task of displaying a world modeled using primitives like lines, polygons, filled / patterned areas, etc. can be carried out in two steps
  - determine the pixels through which the primitive is visible, a process called Rasterization or scan conversion
  - determine the color value to be assigned to each such pixel.

# Raster Graphics Packages



- The efficiency of these steps forms the main criteria to determine the performance of a display
- The raster graphics package is typically a collection of efficient algorithms for scan converting (rasterization) of the display primitives
- High performance graphics workstations have most of these algorithms implemented in hardware

SDL
Simple Directmedia Layer
OpenGL
DirectX 12

# More …

- Google's New AR OS: Fuchsia

Esher

LunarG … SDK

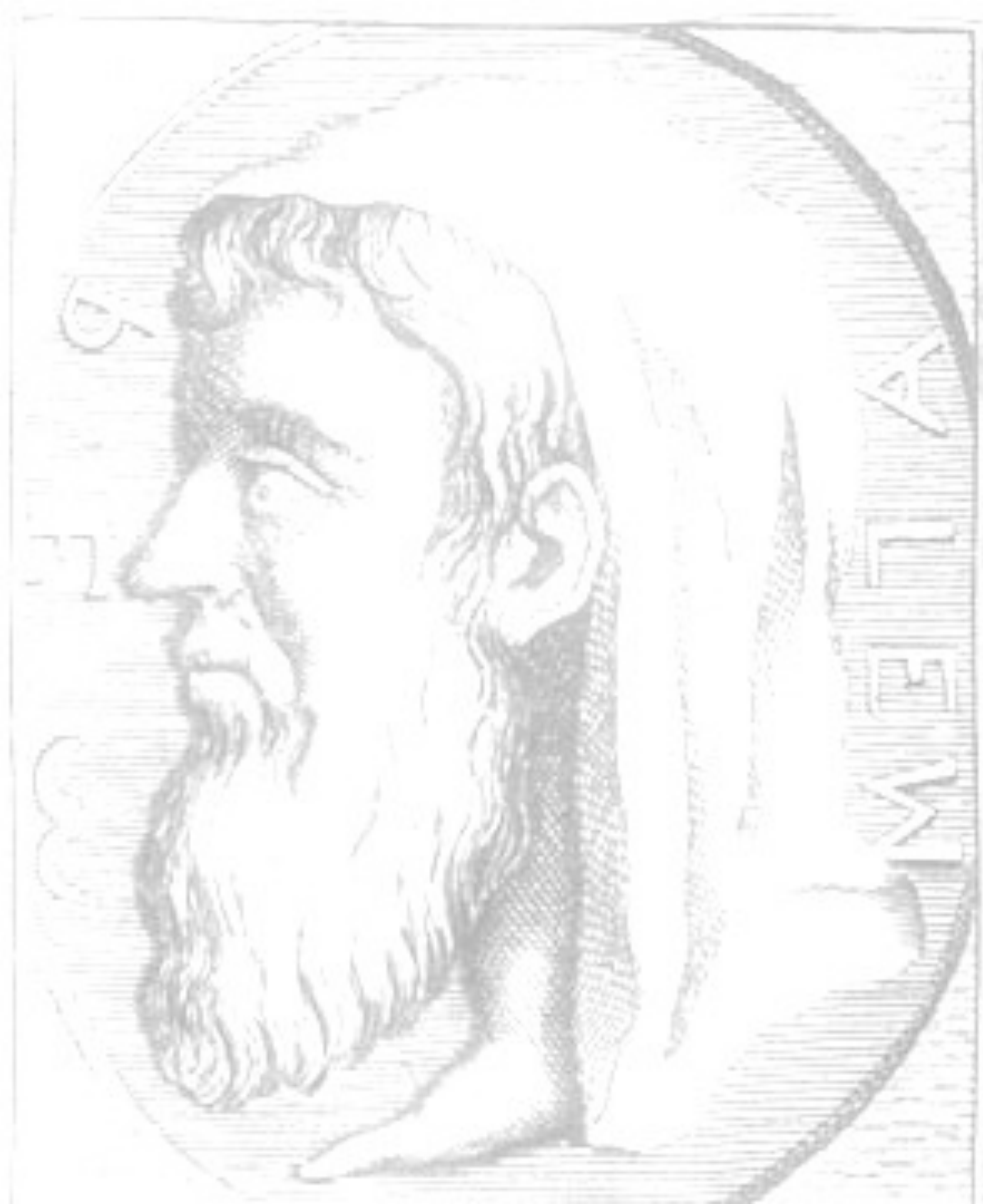
Vulkan

Graphics Hardware

Physically Based Renderer
- Volumetric soft shadows
- Color bleeding
- Light diffusion
- Lens effect

# Why Study these Algorithms?

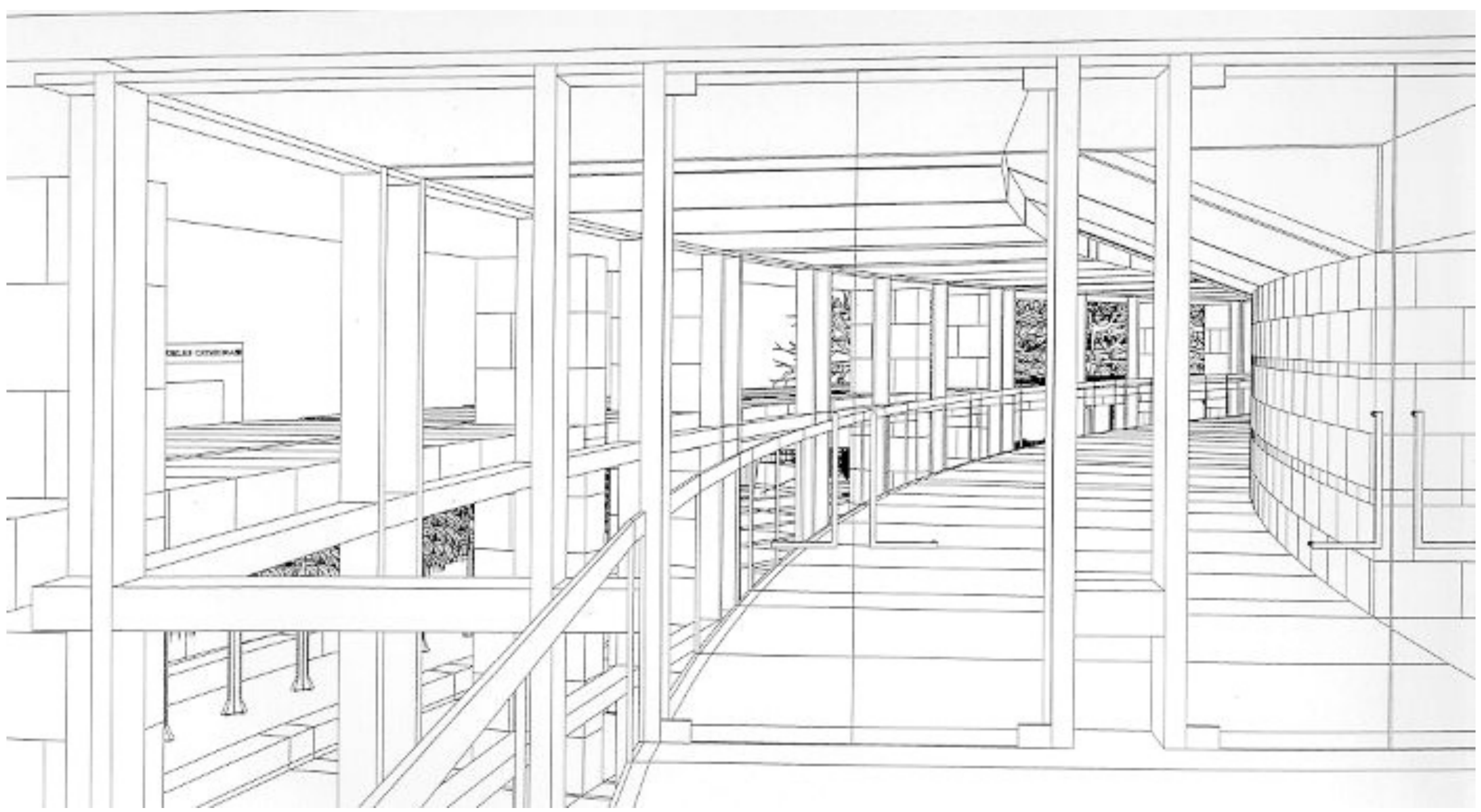
- Some of these algorithms are very good examples of clever algorithmic optimization done to dramatically improve performance using minimal hardware facilities



- Mobile graphics
- Inspiration

# Scan Converting a Line Segment

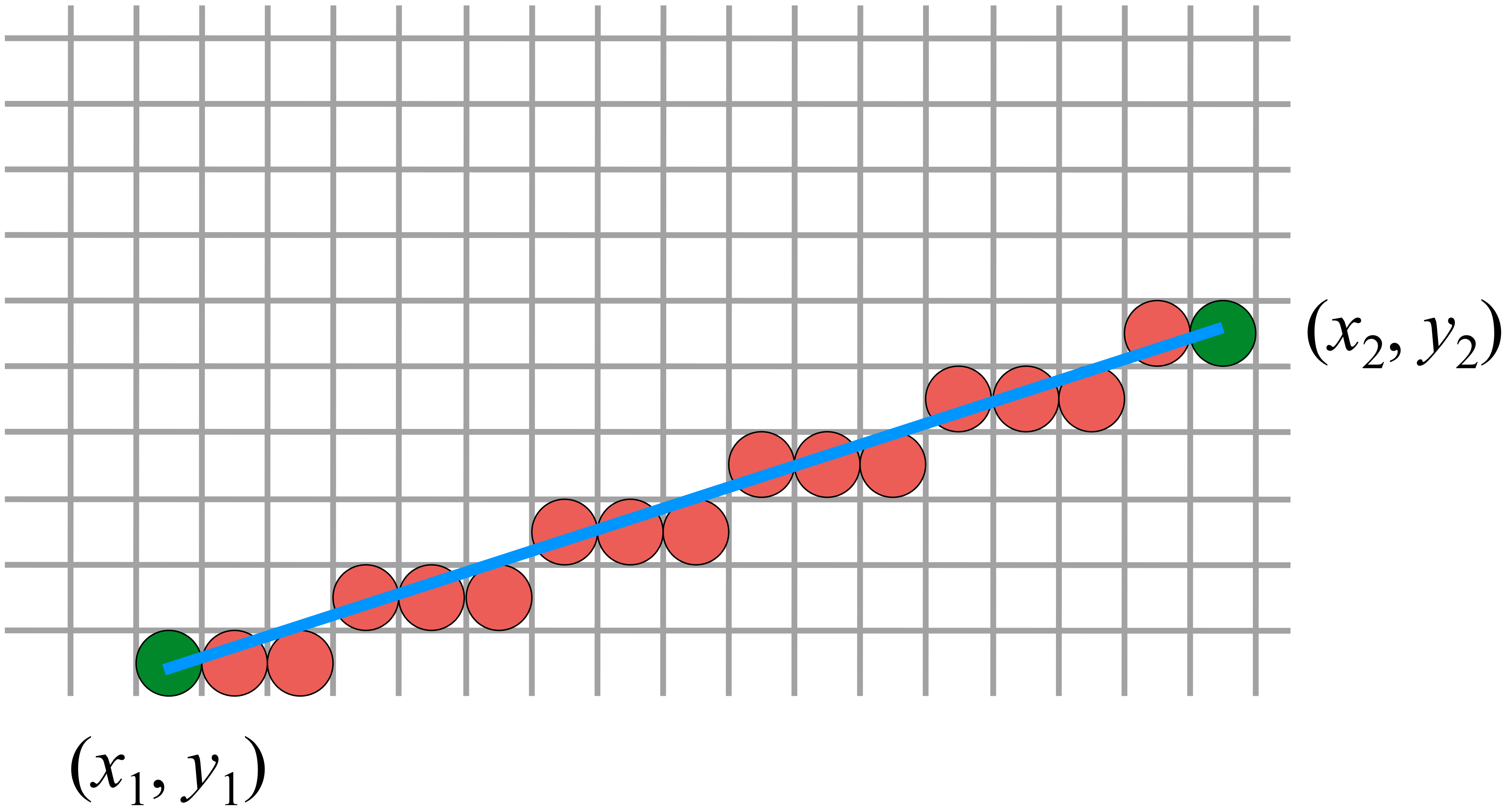
- The line is a powerful element used since the days of Euclid to model the edges in the world.



Given a line segment defined by its endpoints determine the pixels and color which best model the line segment.

# Scan converting lines

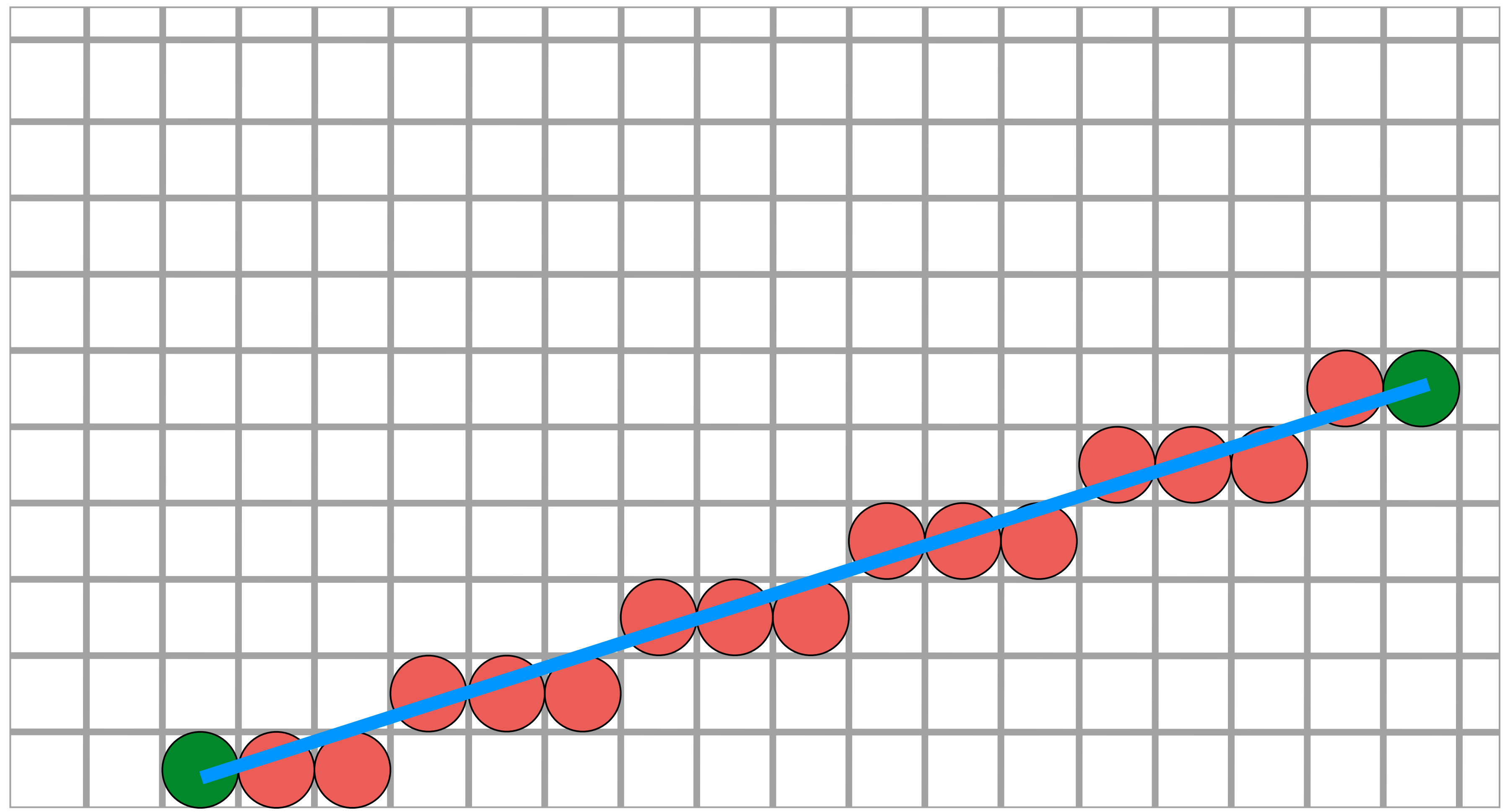
start from $(x_1, y_1)$ end at $(x_2, y_2)$



# Scan converting lines

- Requirements
  - 理想： chosen pixels should lie as close to the ideal line as possible
  - 精确： the sequence of pixels should be as straight as possible
  - 亮度： all lines should appear to be of constant brightness independent of their length and orientation
  - 端点： should start and end accurately
  - 快速： should be drawn as rapidly as possible
  - 变化： should be possible to draw lines with different width and line styles

Question 1: How ?



$(x_1, y_1), (x_2, y_2)$

$y=mx+b$

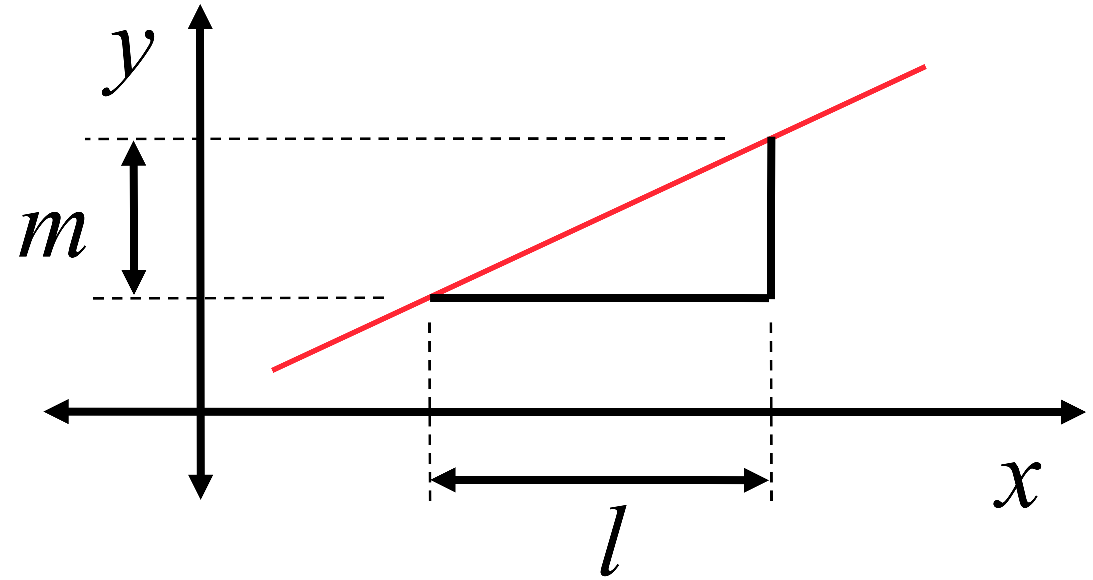
$x_1+1 \Rightarrow y=?$, rounding

$x_1+2 \Rightarrow y=?$, rounding $\rightarrow$ $x_1+i \Rightarrow y=?$, rounding

Question 2: How to speed up?

# Equation of a Line

- Equation of a line is $y - m \cdot x + c = 0$
- For a line segment joining points
- $\boldsymbol{P}(x_1, y_1)$ and $\boldsymbol{Q}(x_2, y_2)$ $\qquad$ *slope* $\qquad m = \dfrac{y2 - y1}{x2 - x1} = \dfrac{\Delta y}{\Delta x}$
- Slope $m$ means that for every unit increment in $x$ the increment in $y$ is $m$ units



# Digital Differential Analyzer (DDA)

- We consider the line in the first octant.
  Other cases can be easily derived.

- Uses differential equation of the line

$$y_i = mx_i + c$$

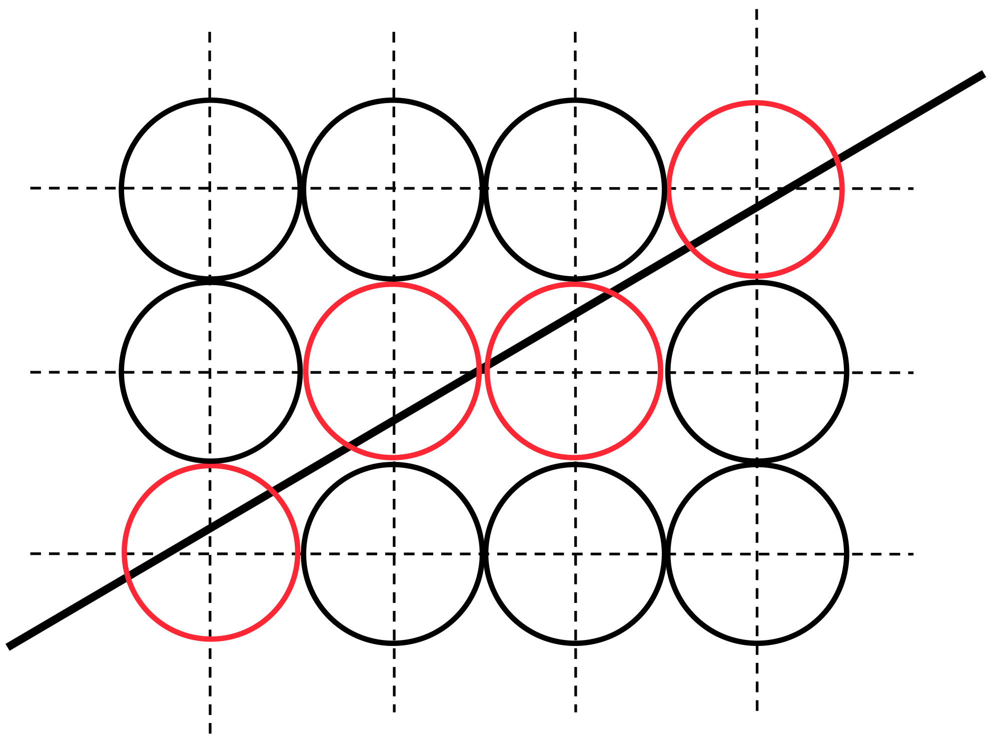
$$where, \quad m = \frac{y2 - y1}{x2 - x1}$$

- Incrementing X-coordinate by 1

$$x_i = x_{i\_prev} + 1$$

$$y_i = y_{i\_prev} + \mathrm{m}$$

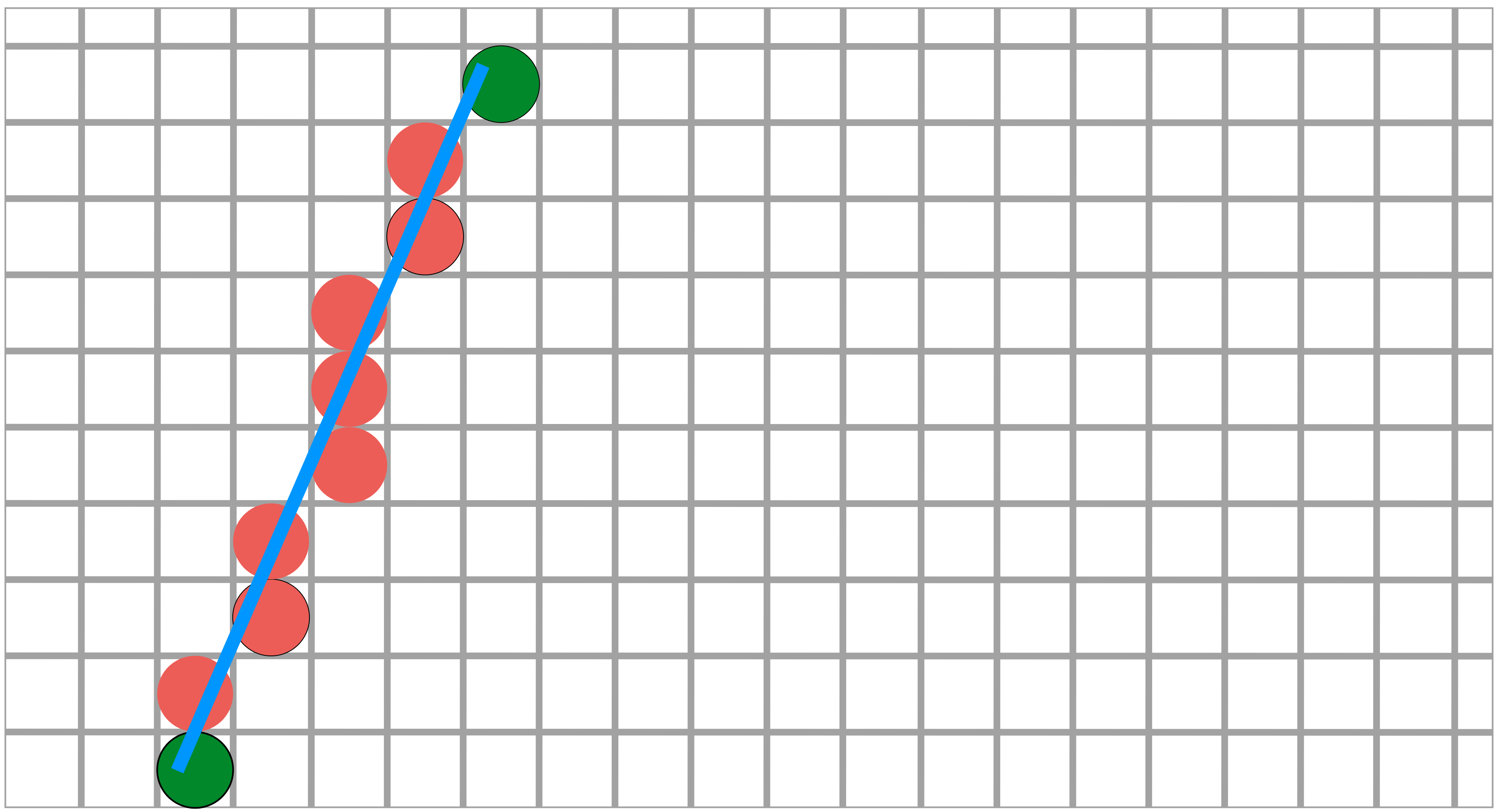
- Illuminate the pixel $[x_i, round(y_i)]$



Discussion1: What technique makes it fast?

Discussion2: Is there any problem in the algorithm?
How to avoid it?

If $\triangle x < \triangle y$



$$y \mathrel{+}= 1;\ x \mathrel{+}= 1/m;$$

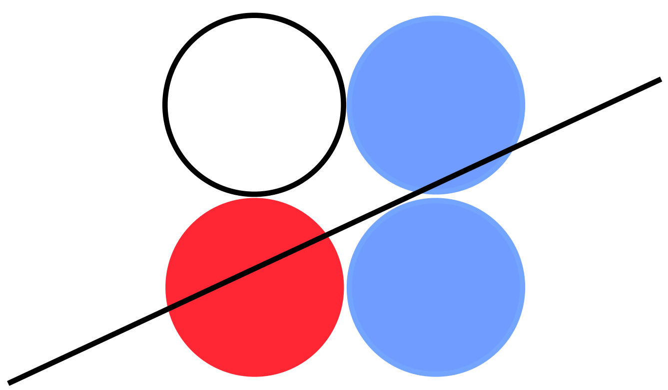
**Divide and conquer!**

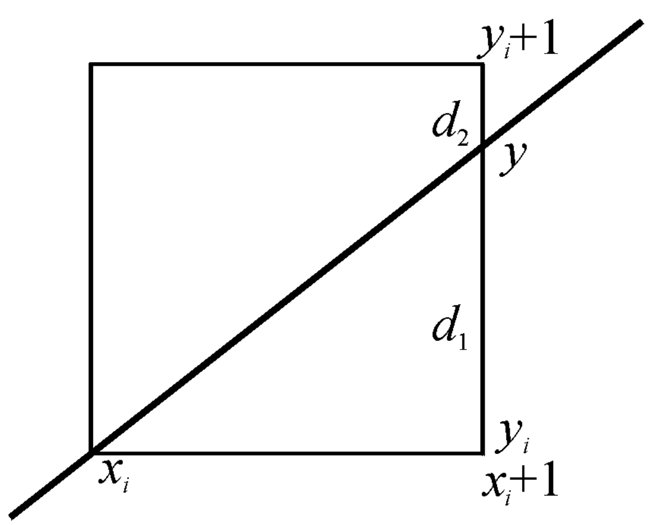
# Digital Differential Analyzer

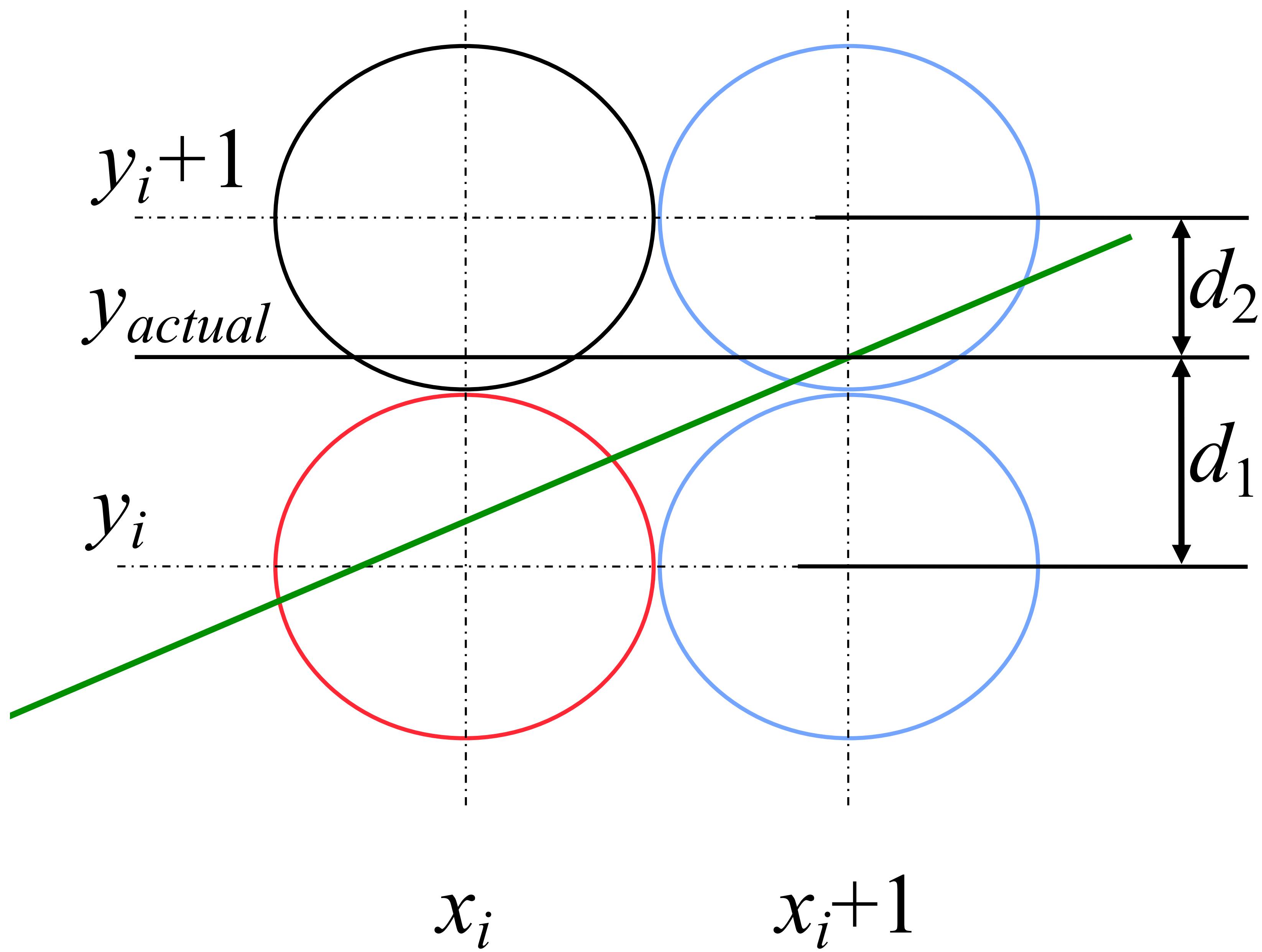
- Digital Differential Analyzer algorithm (a.k.a. DDA)
- Incremental algorithm: at each step it makes incremental calculations based on the calculations done during the preceding step
- The algorithm uses floating point operations.

- An algorithm to avoid this problem is first proposed by J. Bresenham (1937~) of IBM.
- The algorithm is well known as Bresenham's Line Drawing Algorithm (1962, when he was 25).

# Bresenham Line Drawing






$$y_i = mx_i + c$$

$$\textit{where,} \qquad m = \frac{y2-y1}{x2-x1}$$

$d_1 \ge d_2$ ? $\Rightarrow$ $y_{i+1} = y_i$ or $y_{i+1}=y_i+1$

$y = m(x_i+1) + b$

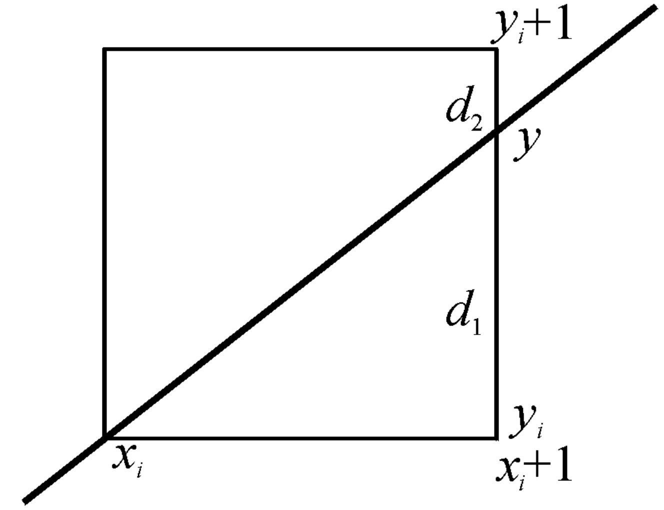
(2.1)

$d_1 = y - y_i$

(2.2)

$d_2 = y_i + 1 - y$

(2.3)




If $d_1 - d_2 > 0$, then $y_{i+1}=y_i+1$, else $y_{i+1}=y_i$

substitute (2.1)、(2.2)、(2.3) into $d_1 - d_2$,

$$d_1 - d_2 = 2y - 2y_i - 1 = 2dy/dx * x_i + 2dy/dx + 2b - 2y_i - 1$$

on each side of the equation, $*$ dx, denote $(d_1 - d_2)$ dx as $P_i$, we have

$$P_i = 2x_i\,dy - 2y_i\,dx + 2dy + (2b-1)dx \qquad (2.4)$$

Because in first octant dx>0, we have sign$(d_1 - d_2)$=sign $(P_i)$

If $P_i > 0$, then $y_{i+1}=y_i+1$, else $y_{i+1}=y_i$

$$P_{i+1} = 2x_{i+1}dy - 2y_{i+1}dx + 2dy + (2b-1)dx,$$ note that $x_{i+1} = x_i + 1$

$$P_{i+1} = P_i + 2dy - 2(y_{i+1} - y_i)\,dx \qquad (2.5)$$

# Bresenham algorithm in first octant

1.Initialization $P_0 = 2\,dy - dx$

2.draw $(x_1, y_1)$, $dx=x_2-x_1$, $dy=y_2-y_1$,

Calculate $P_1=2dy-dx$, $i=1$;

3.$x_{i+1} = x_i + 1$

if $P_i>0$, then $y_{i+1}=y_i+1$, else $y_{i+1}=y_i$;

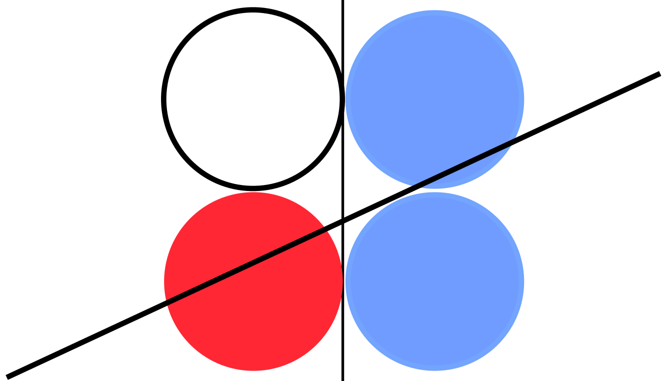
4.draw $(x_{i+1}, y_{i+1})$;

5.calculate $P_{i+1}$:

if $P_i>0$ then $P_{i+1}=P_i+2dy-2dx$,

else $P_{i+1}=P_i+2dy$;

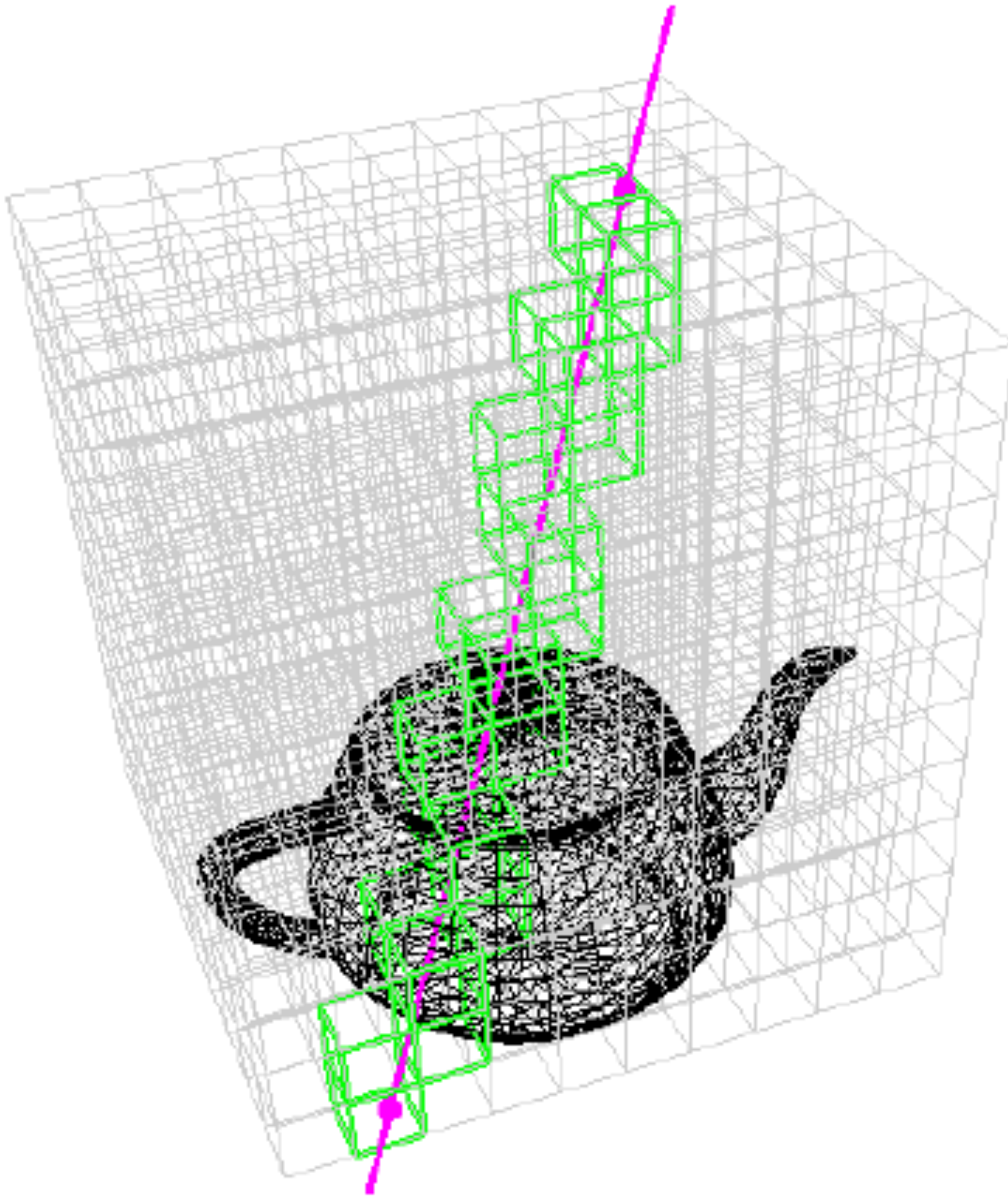
6.$i=i+1$; if $i<dx+1$ then goto 3; else end



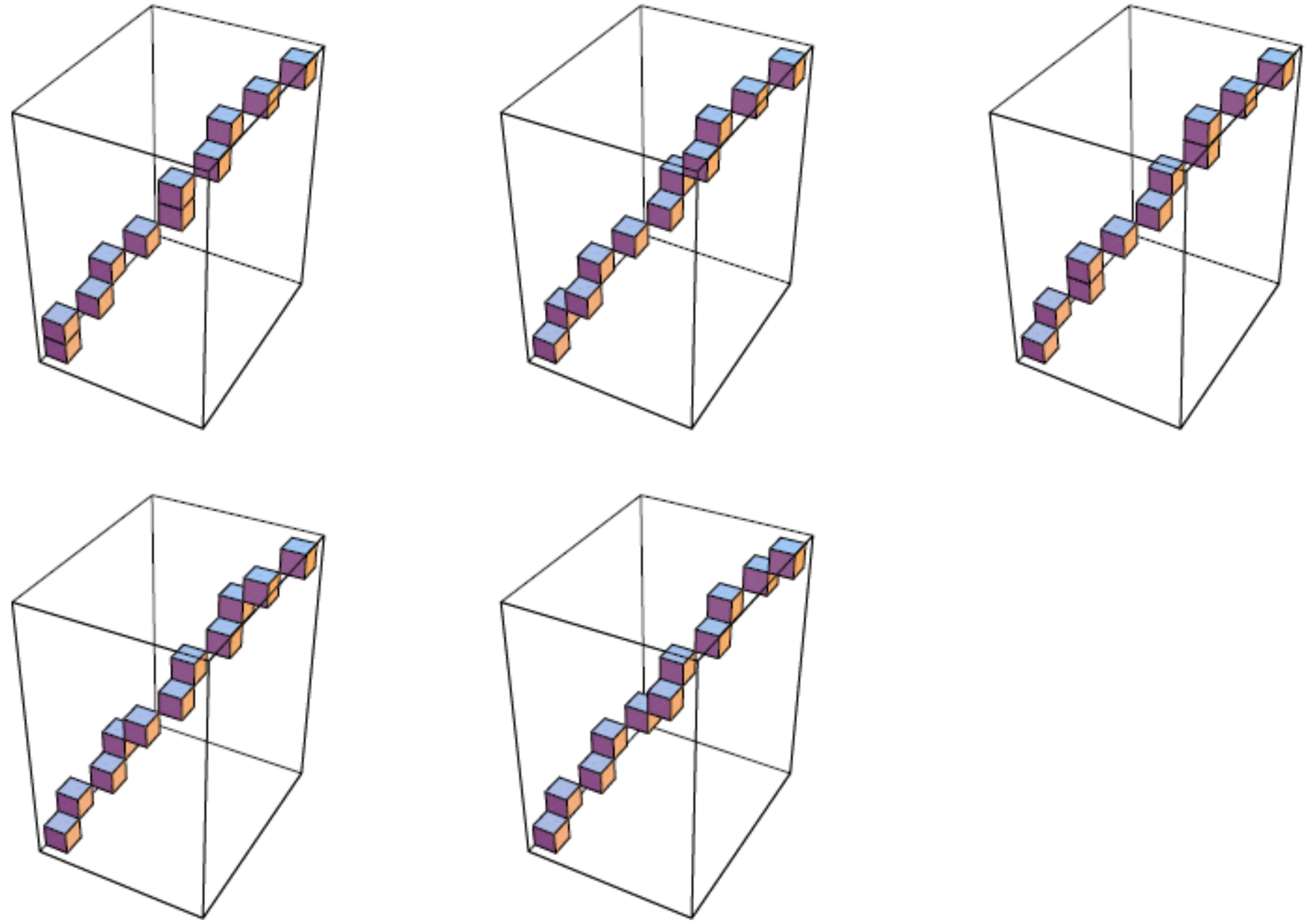
Question 3: Is it faster than DDA ?

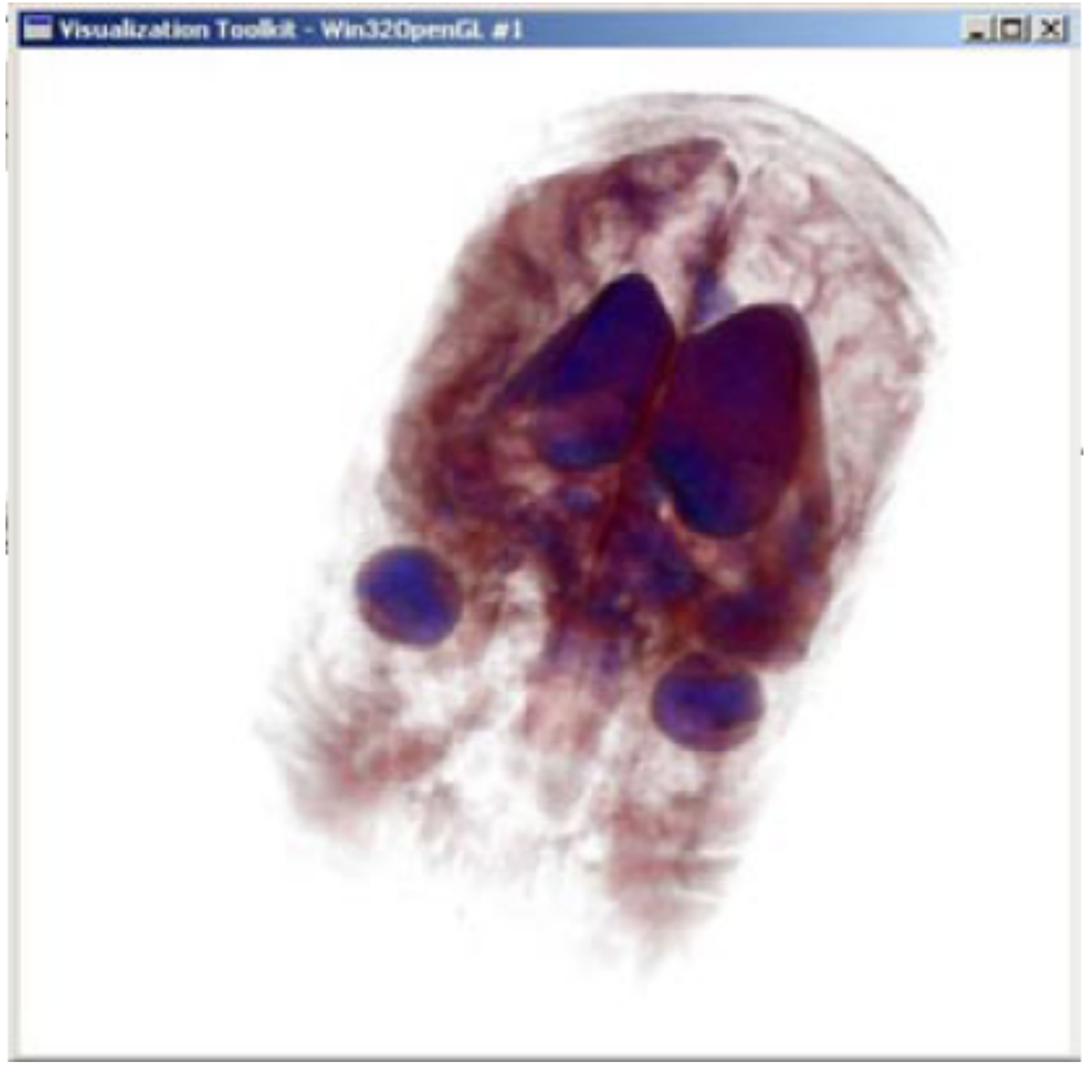
Question 4: What technique ?

# 3D DDA and 3D Bresenham



# 3D DDA and 3D Bresenham algorithm





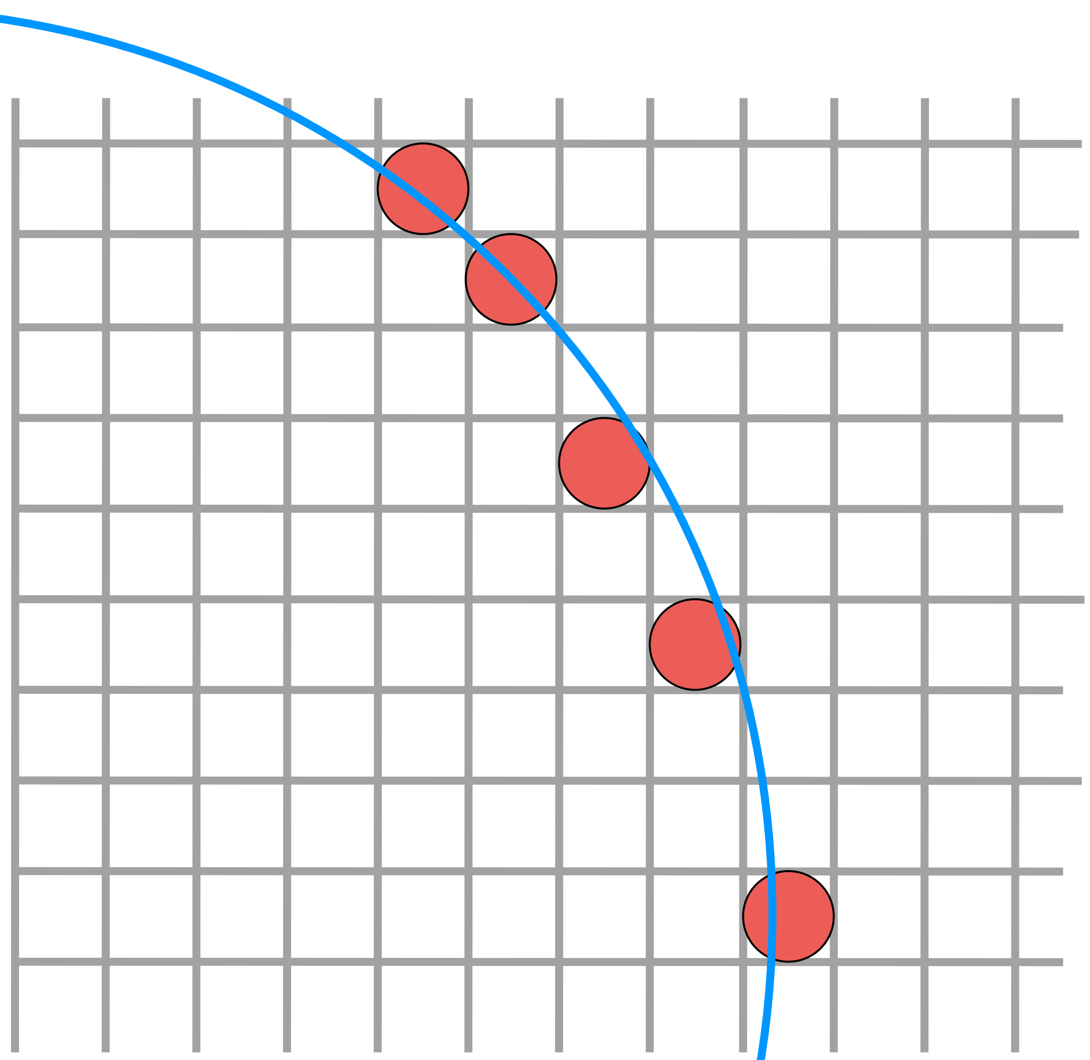
# Scan converting circles

A circle with center $(x_c, y_c)$ and radius $r$:

$$(x-x_c)^2 + (y-y_c)^2 = r^2$$

orthogonal coordinate

$$y = y_c \pm \sqrt{r^2 - (x - x_c)^2}$$

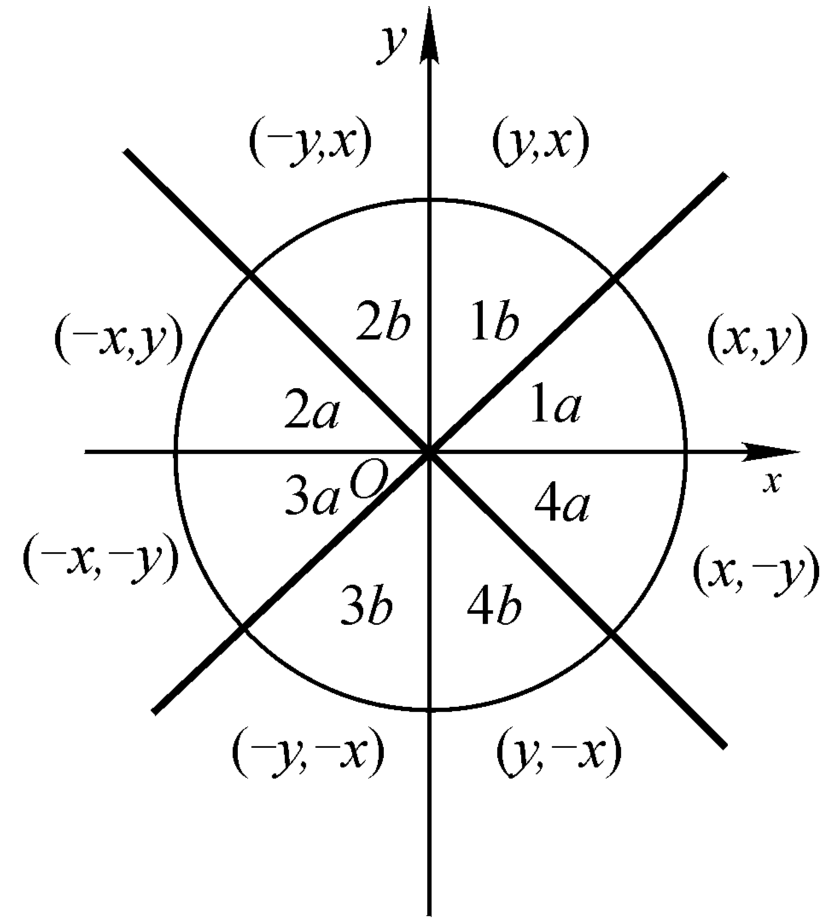


## polar coordinates

$x = x_c + r \cdot \cos\theta$

$y = y_c + r \cdot \sin\theta$

$$x_i = x_c + r \cdot \cos(i * \Delta\theta)$$
$$y_i = y_c + r \cdot \sin(i * \Delta\theta)$$




Can be accelerated by symmetrical characteristic

$$\theta = i * \Delta\theta, \quad i=0,1,2,3,\ldots.$$

Discussion 3 : How to speed up?

$$x_i = r\cos\theta_i$$
$$y_i = r\sin\theta_i$$

$$\begin{aligned} x_{i+1} &= r\cos(\theta_i + \Delta\theta) \\ &= r\cos\theta_i\cos\Delta\theta - r\sin\theta_i\sin\Delta\theta \\ &= x_i\cos\Delta\theta - y_i\sin\Delta\theta \end{aligned}$$

Bresenham Algorithm

# Homework 1

- Bresenham algorithm for drawing circle (deadline: 2019-10-08)
  - Please write down your answer in A4 papers (physical or digital format are both acceptable),
  - and capture it/them by mobile phone camera,
  - finally submit to TA via email (baidu yunpan link, recommended)
  - bonus: implementation (OpenGL, SDL or WebGL)

*Different representations*

$$y = y_c \pm \sqrt{r^2 - (x - x_c)^2} \longrightarrow y = f(x) \quad x \in (x_0,\ x_1)$$

(explicit curve)

$$\begin{aligned} x &= x_c + r \cdot \cos\theta \\ y &= y_c + r \cdot \sin\theta \end{aligned} \longrightarrow \begin{cases} x = x(t) \\ y = y(t) \end{cases} \quad t \in (t_0,\ t_1)$$

(parametric curve)

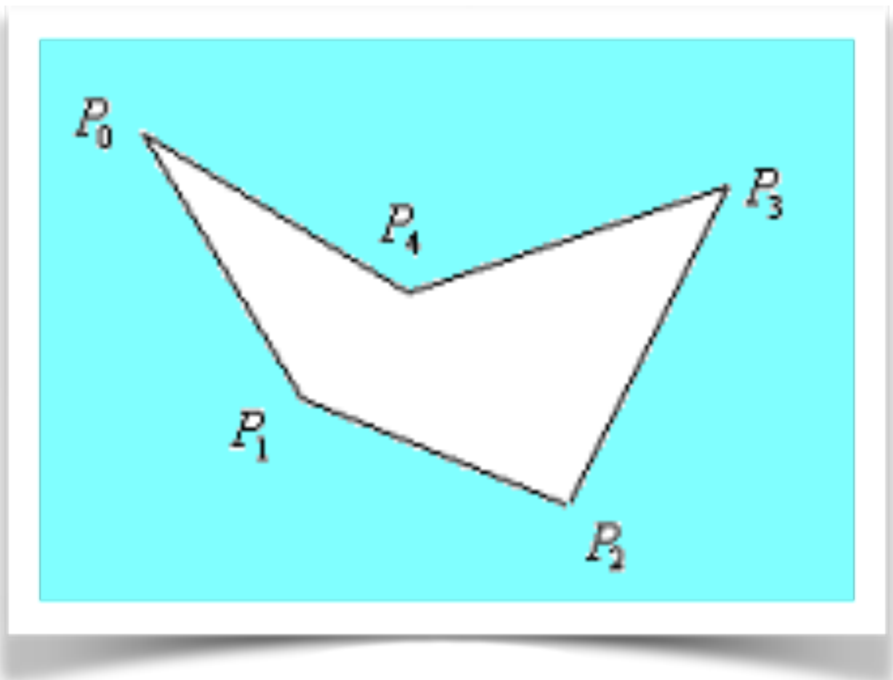
$$(x - x_c)^2 + (y - y_c)^2 = r^2 \longrightarrow g(x,y)=0$$

(implicit curve)

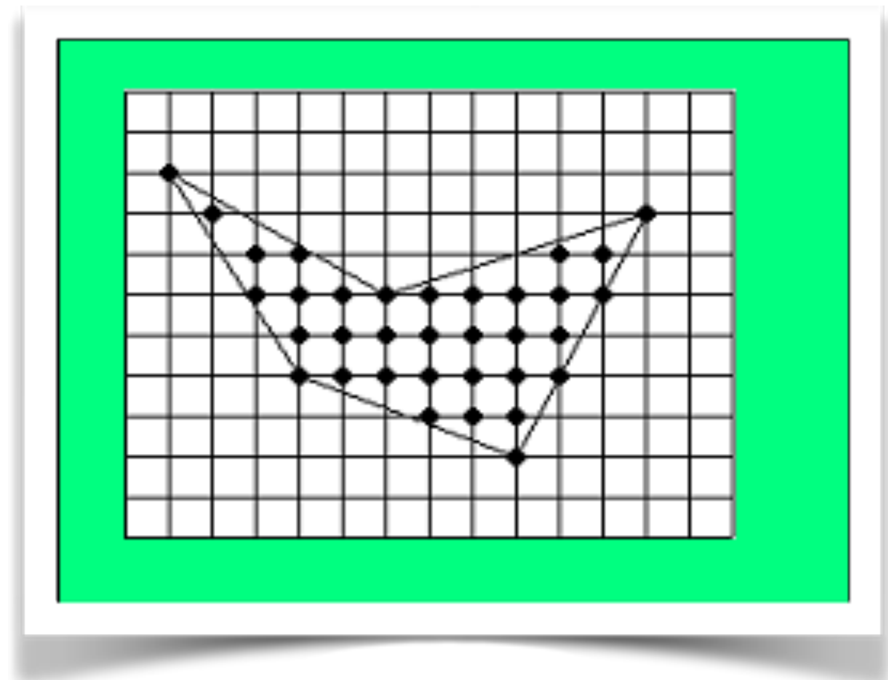
*Discussion 4 : How to display an explicit curve,*
*How to display a parametric curve*

# Polygon filling

- Polygon representation




By vertex



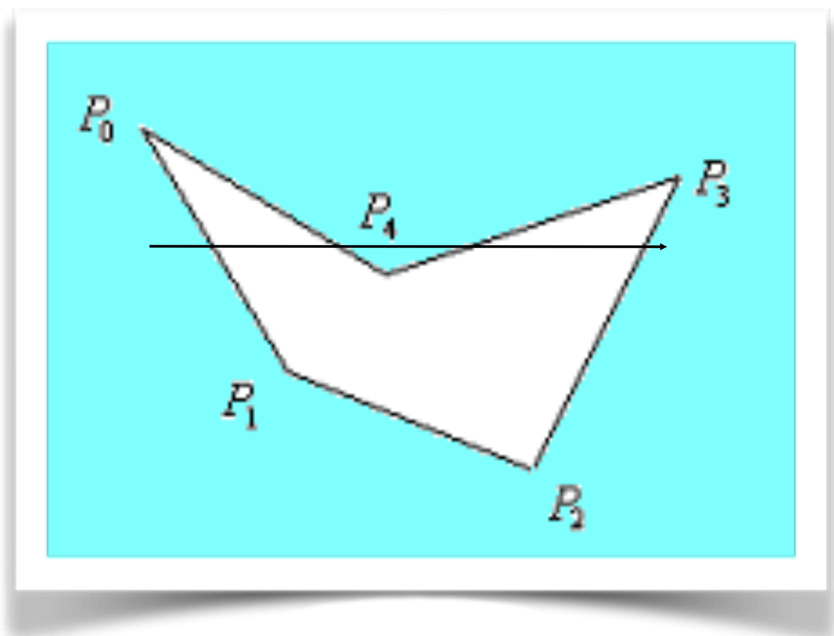
By lattice

- Polygon filling:

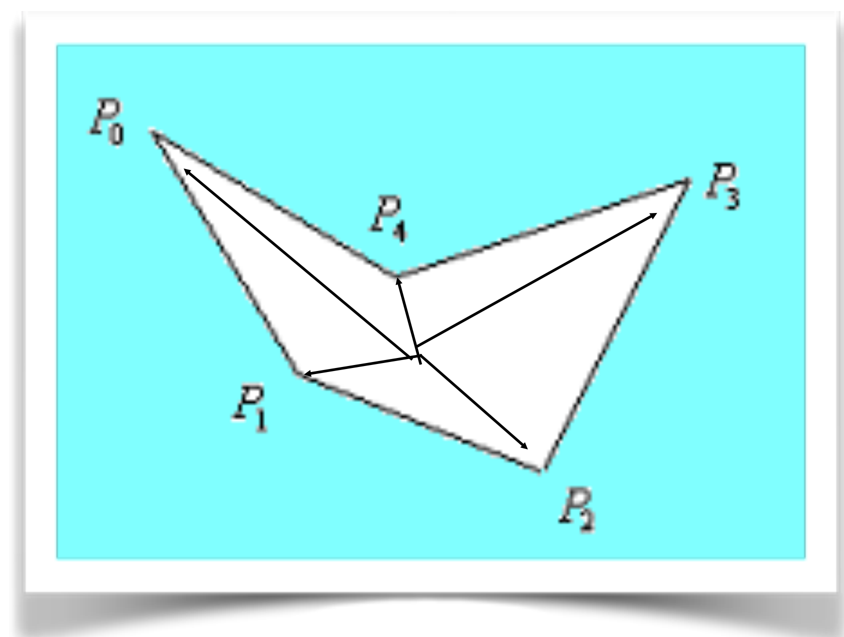
vertex representation → lattice representation

# Polygon filling

- fill a polygonal area → test every pixel in the raster to see if it lies inside the polygon.



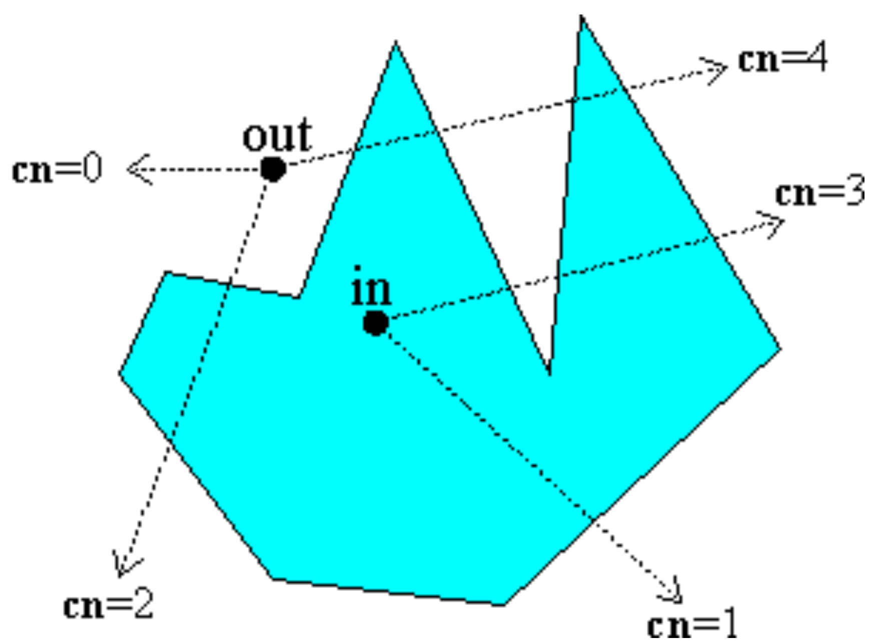

even-odd test

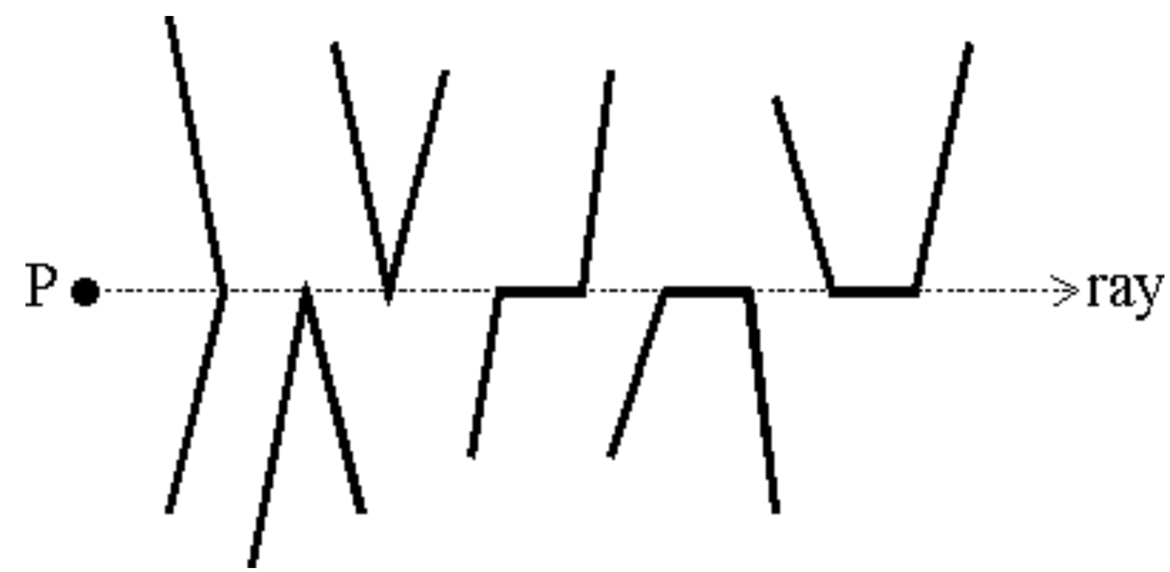



winding number test

Question5: How to Judge…?

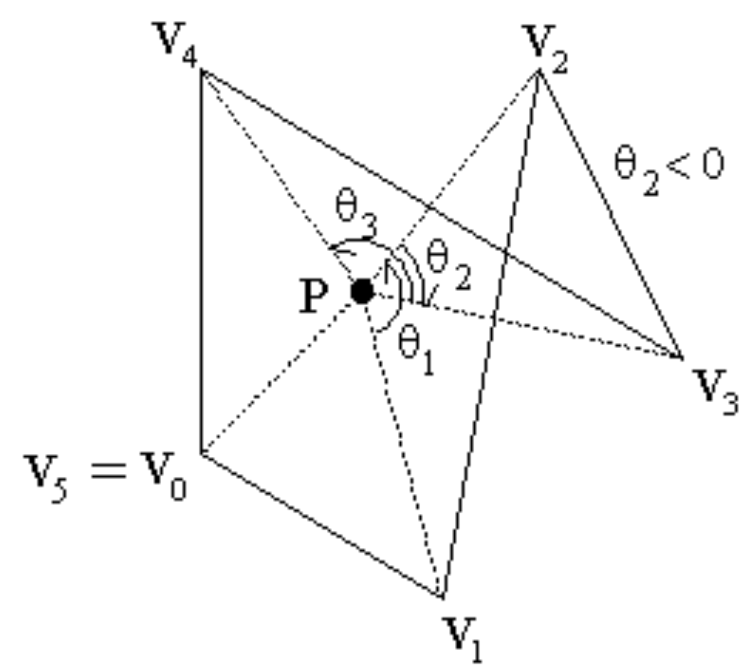
# Inside check




$$\mathbf{wn} = \frac{1}{2\pi}\sum_{i=0}^{n-1}\theta_i$$

$$= \frac{1}{2\pi}\sum_{i=0}^{n-1}\arccos\left(\frac{\mathbf{PV}_i \cdot \mathbf{PV}_{i+1}}{\left|\mathbf{PV}_i\right|\left|\mathbf{PV}_{i+1}\right|}\right)$$




Question6: How to improve …?