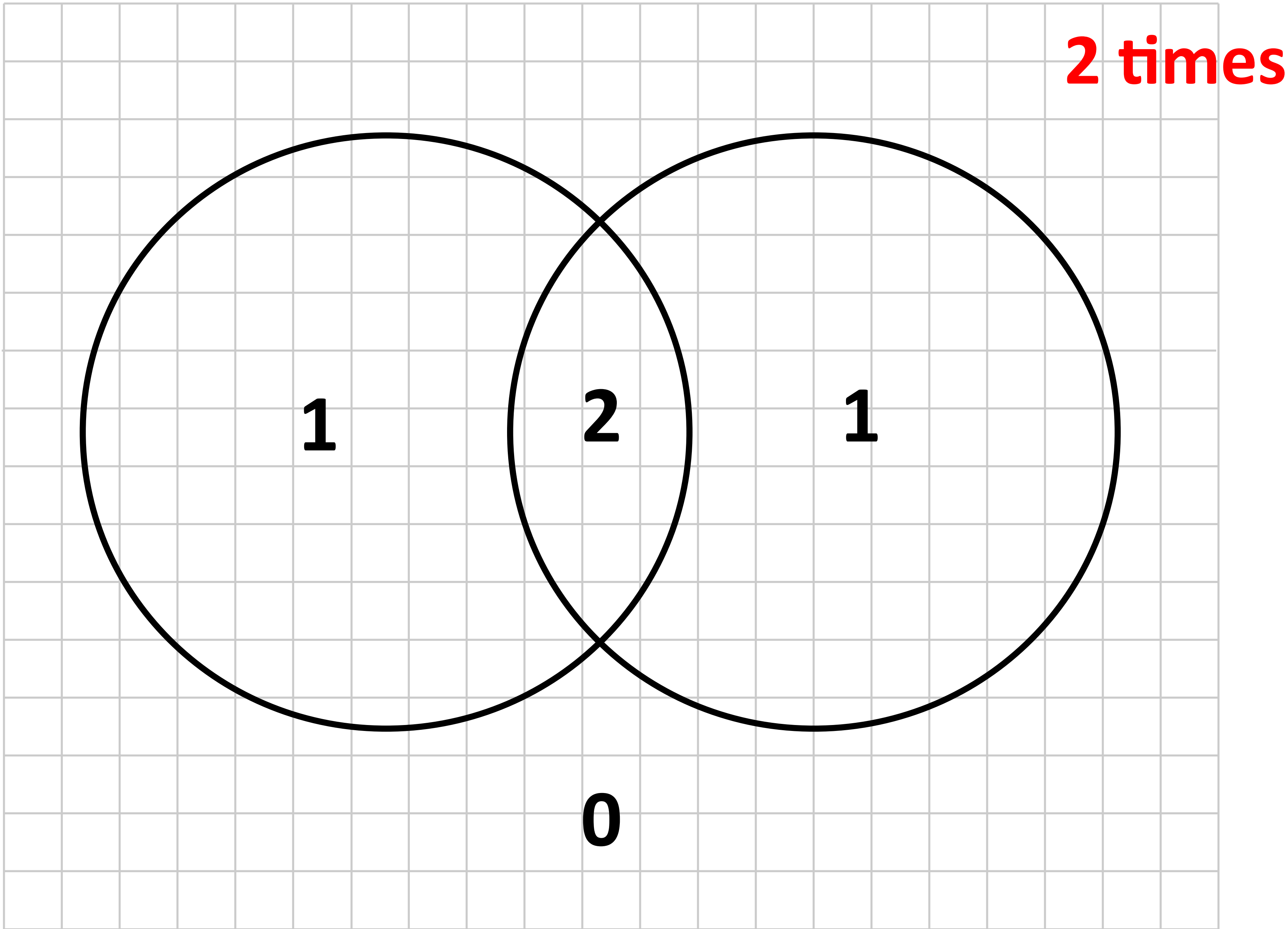
Neural implementation of shape-invariant touch counter based on Euler calculus

Keiji Miura, Tohoku Univ.
Kazuki Nakada, Univ. of Electro-Communications

Topics in Differential Geometry and its Discretizations
Jan 10, 2015

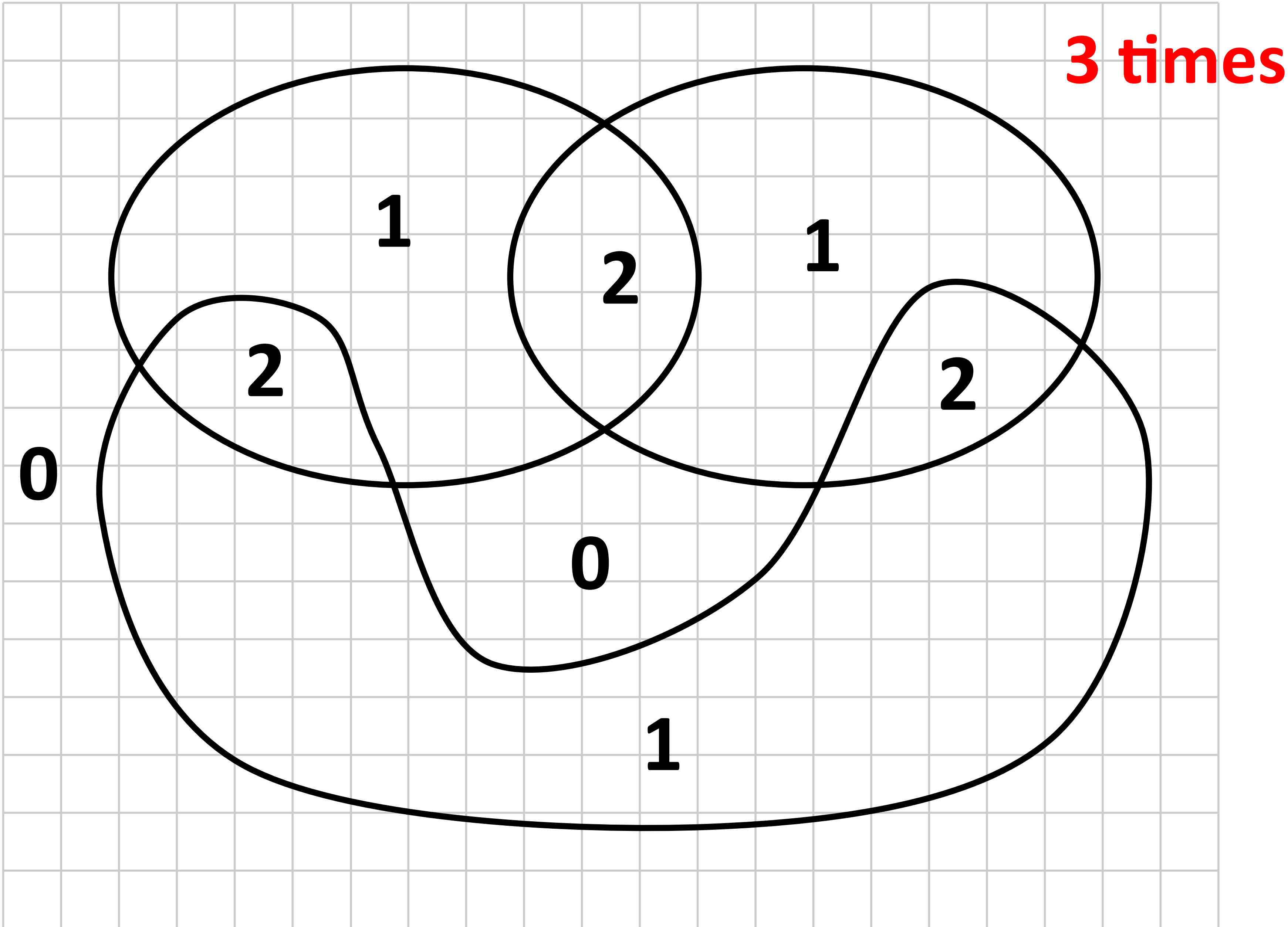
Goal : shape-invariant touch count

(No time resolution)



Goal : shape-invariant touch count

(No time resolution)



Goal : touch count

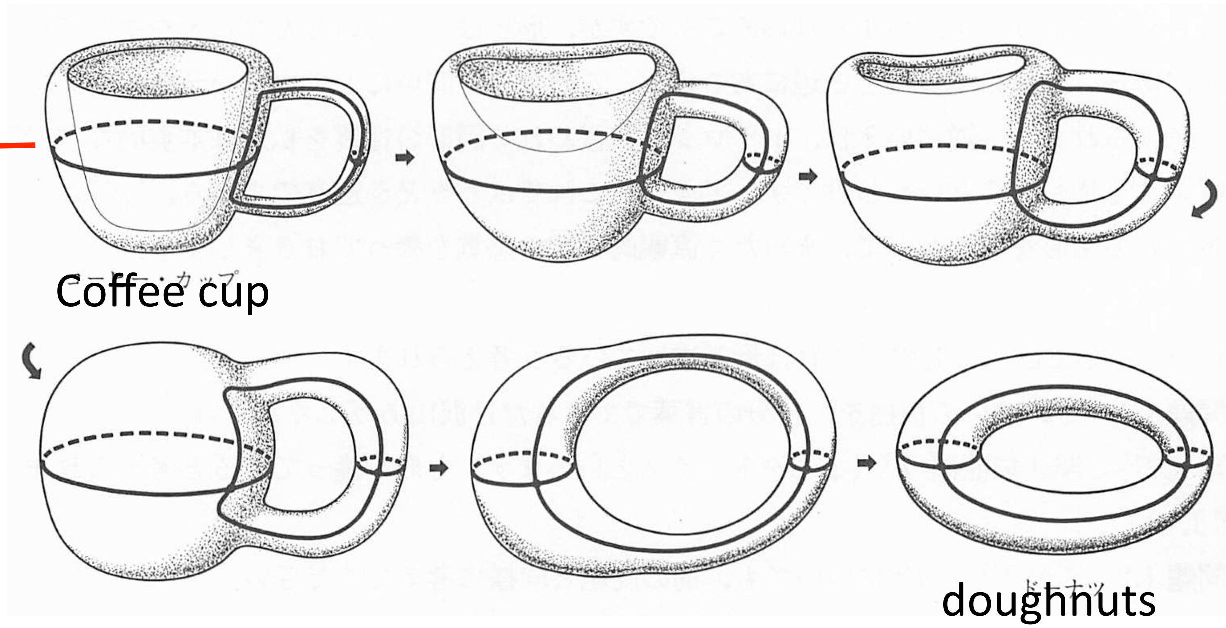
1. Invariance to finger shapes
   → Topology (Euler Calculus, Ghrist group)

2. Speed up
   → Parallelization with neural networks

Topology gives invariants under morphing

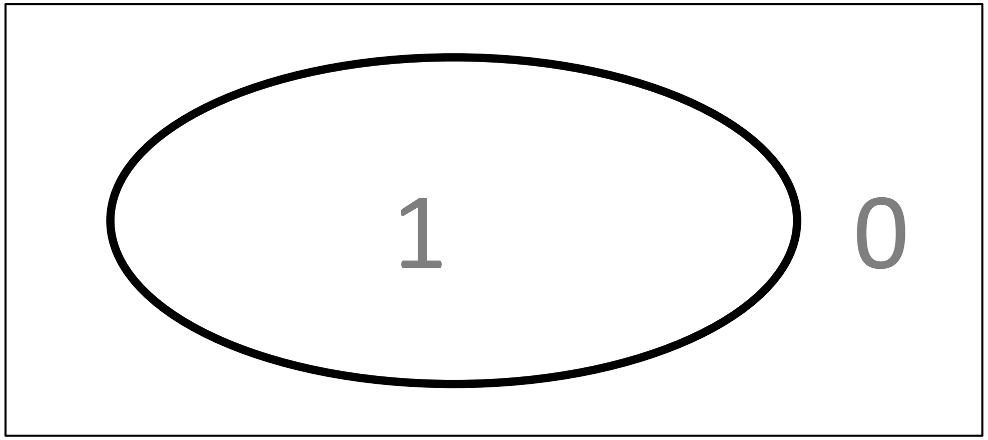
- β_0 = #connected components (in a binary image)
- β_1 = #holes
- $\chi = \beta_0 - \beta_1$

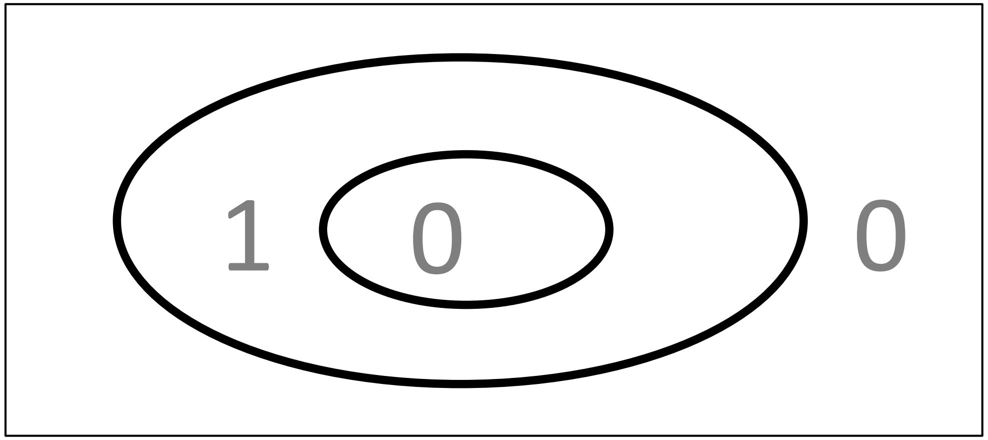
$\beta_0 = 1$
$\beta_1 = 1$
$\chi = 1\text{-}1 = 0$

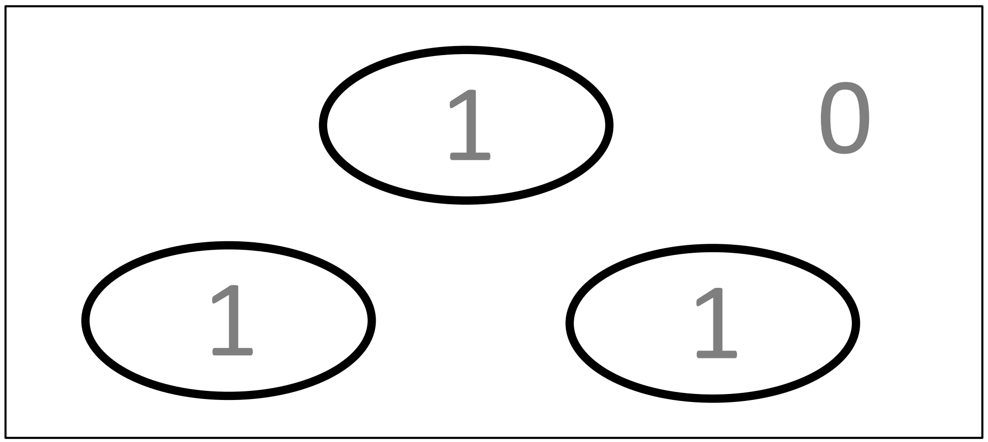


$\beta_0 = 1$
$\beta_1 = 1$
$\chi = 1\text{-}1 = 0$

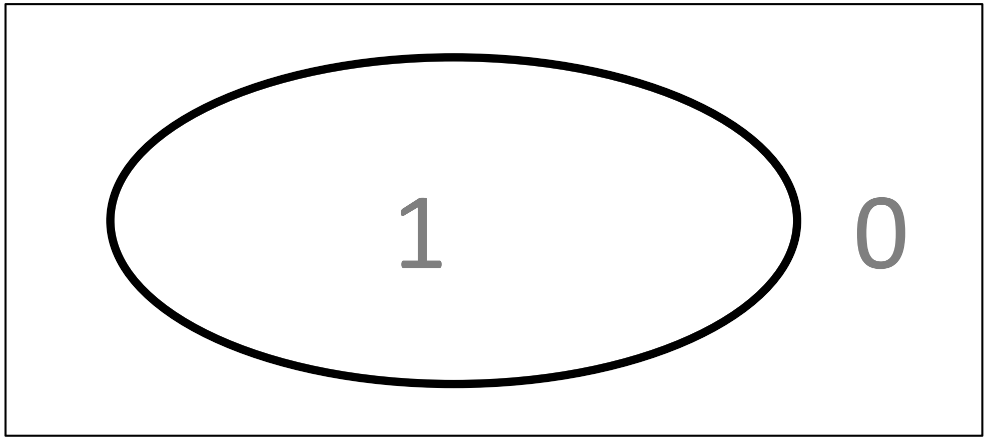
"トポロジー：柔らかい幾何学" written by 瀬山士郎

$\beta_0\Big($  $\Big)=1$

$\beta_0\Big($  $\Big)=1$

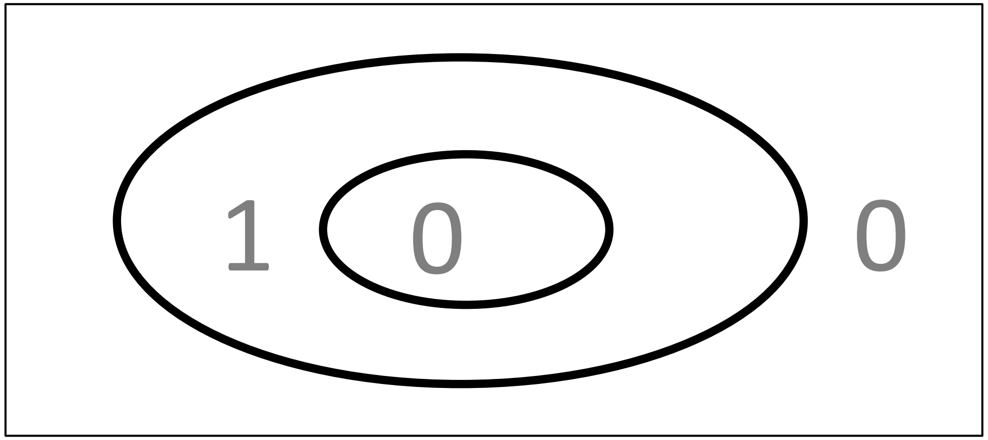
$\beta_0\Big($  $\Big)=3$

$\beta_1($



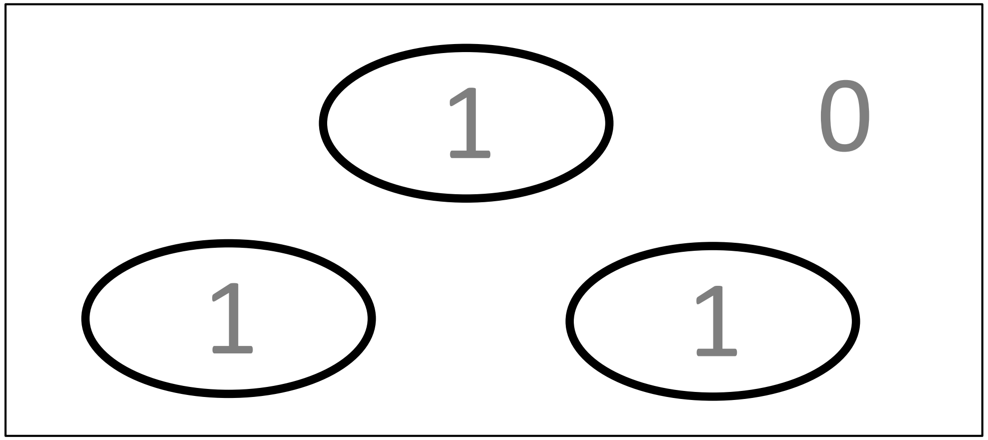
$)=0$

$\beta_1($



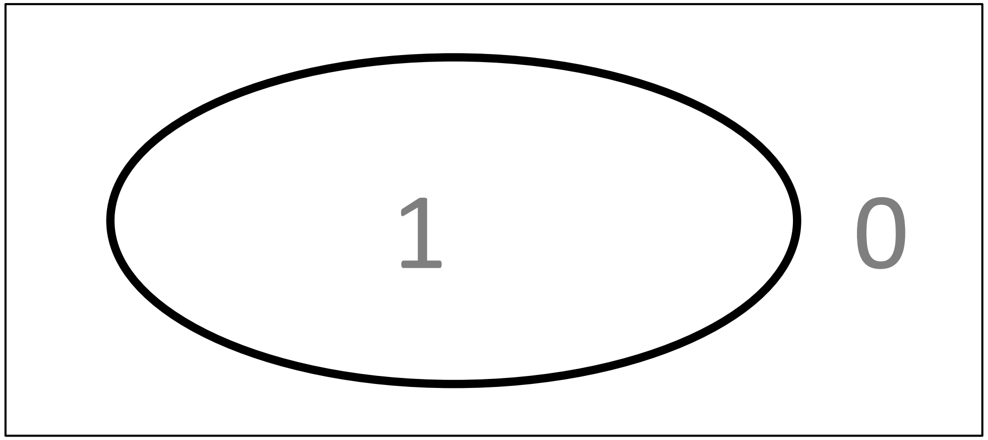
$)=1$

$\beta_1($



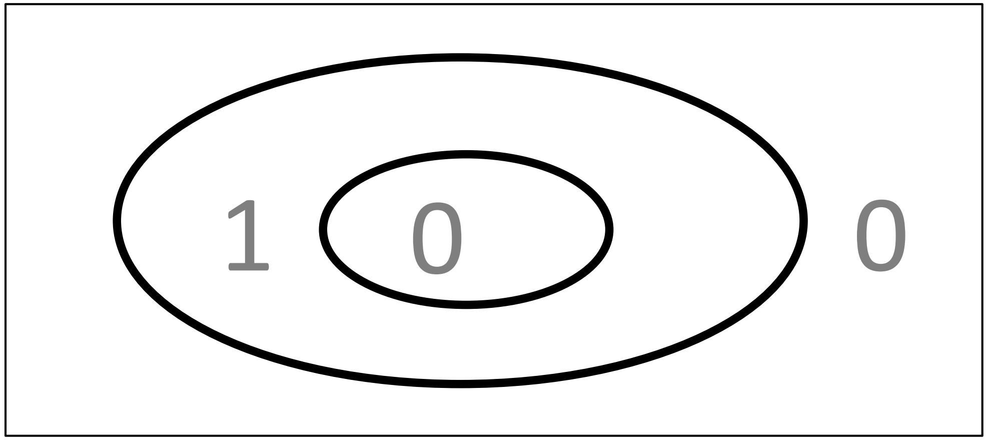
$)=0$

$\chi($



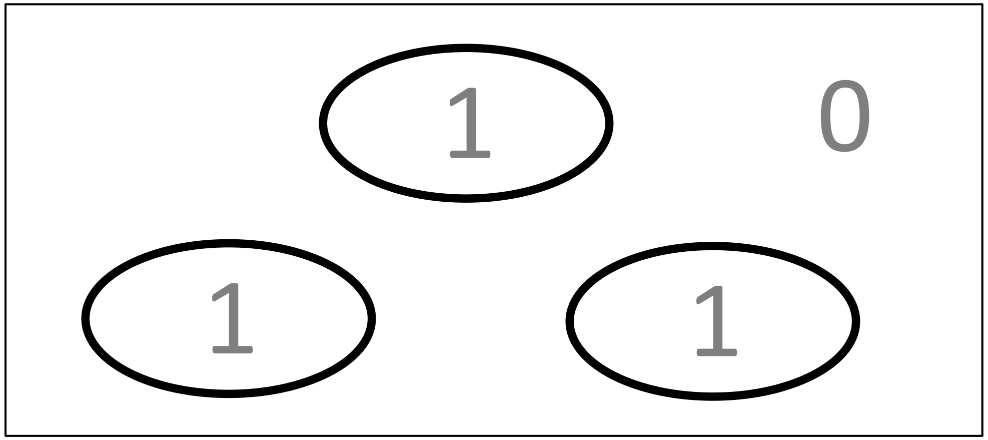
$)=1$

$\chi($



$)=0$

$\chi($

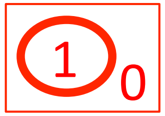


$)=3$

Euler Integral for multivalued image

(Ghrist, 2009-)

Generalized integral:

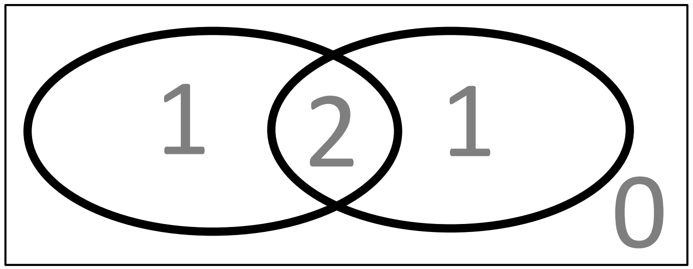
- <u>One touch:</u> $\int(f_1)dx = \chi($ 

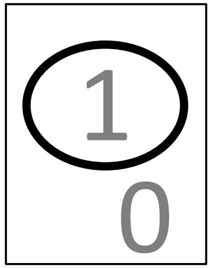
$) = 1$

  cf. $\chi_{\text{binary image}}$:= (#connected components) – (#holes)

- <u>More touches:</u> the integral is defined by a sum of indicator functions (binary images):

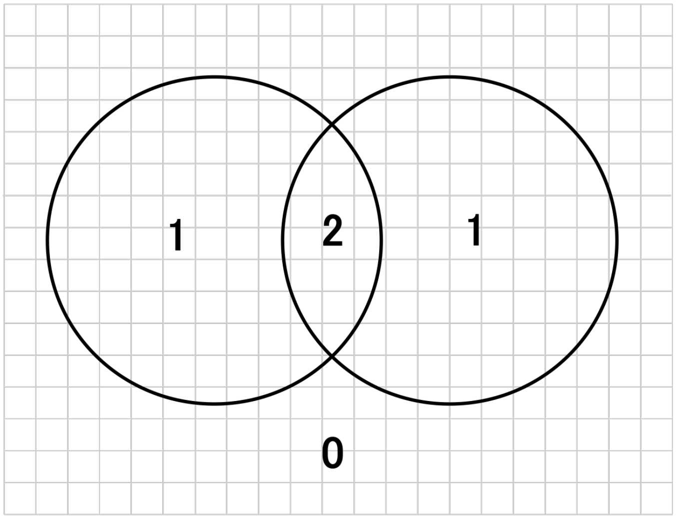
  $\int(\sum f_i)dx \;=\; \sum(\int f_i dx) \;=\;$ Touch number!

$\chi($ 

$)=2\chi($ 

$)=2$

Example 1



$$\fallingdotseq \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

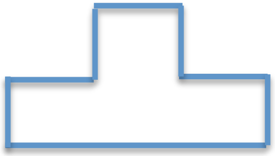
$=$

Height > 0

$+$

Height > 1

Automation with level sets



Decompose into binary indicator functions

Level sets

$$\chi\left(\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}\right) = \chi\left(\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}\right) + \underline{\chi\left(\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}\right)}$$

$$= \beta_0\left(\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}\right) - \beta_0\left(\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 0 & 0 & 0 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}\right) + 1$$

Flip 0 and 1

$$+ \underline{\beta_0\left(\begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}\right) - \beta_0\left(\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 0 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix}\right) + 1}$$

$$= 1 - 1 + 1 + \underline{1 - 1 + 1} + 0 + 0 + \ldots$$

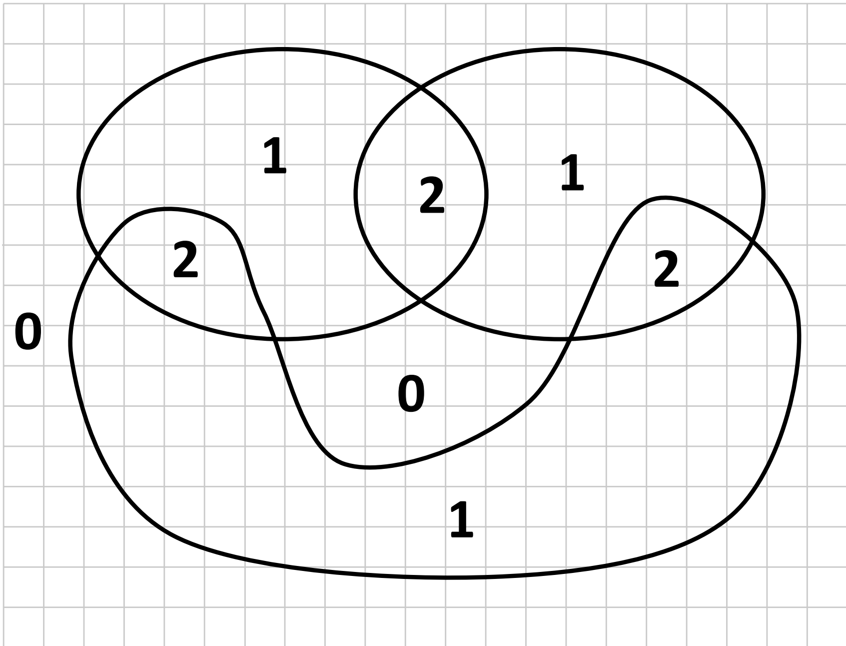
$$= 2.$$

Alexander Duality

Formula:

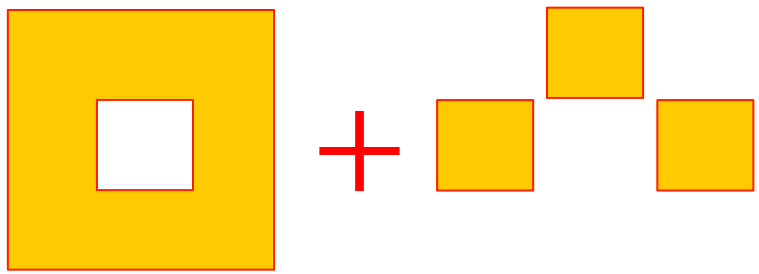
$$\boxed{\chi(h) = \sum_{s=0}^{\infty} \chi(h > s) = \sum_{s=0}^{\infty} [\beta_0(h > s) - \beta_0(h \leq s) + 1]}$$

Example 2



$$\doteqdot \begin{bmatrix} 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 2 & 1 & 0 \\ 0 & 2 & 0 & 2 & 0 \\ 0 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 \end{bmatrix}$$

$=$



Circuit Simulation on Matlab/Simulink

$$\begin{aligned}\chi(h) &= \chi(h>0)+\chi(h>1)+0+0+\ldots\\ &= [\beta_0(h>0)-\beta_1(h>0)]+[\beta_0(h>1)-\beta_1(h>1)]\\ &= [\beta_0(h>0)-\beta_0(h\le 0)+1]+[\beta_0(h>1)-\beta_0(h\le 1)+1] \quad \text{(Alexander Duality)}\\ &= [1-2+1]+[3-1+1]=3\end{aligned}$$

Upper-level module that calls β_0:

a

$$\beta_0(h>0)=\beta_0\left(\boxed{\begin{matrix}0&0&0&0&0\\0&1&1&1&0\\0&1&0&1&0\\0&1&1&1&0\\0&0&0&0&0\end{matrix}}\right)=1$$

b

$$\beta_0(h\le 0)=\beta_0\left(\boxed{\begin{matrix}1&1&1&1&1\\1&0&0&0&1\\1&0&1&0&1\\1&0&0&0&1\\1&1&1&1&1\end{matrix}}\right)=2$$

c

$$\beta_0(h>1)=\beta_0\left(\boxed{\begin{matrix}0&0&0&0&0\\0&0&1&0&0\\0&1&0&1&0\\0&0&0&0&0\\0&0&0&0&0\end{matrix}}\right)=3$$

d

$$\beta_0(h\le 1)=\beta_0\left(\boxed{\begin{matrix}1&1&1&1&1\\1&1&0&1&1\\1&0&1&0&1\\1&1&1&1&1\\1&1&1&1&1\end{matrix}}\right)=1$$

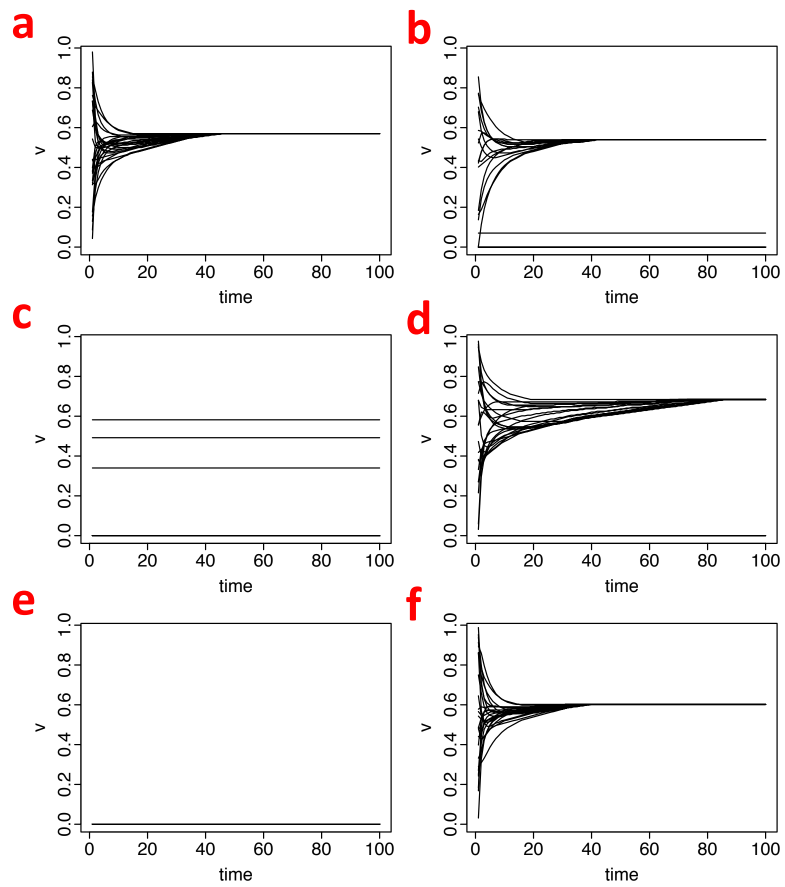
e

$$\beta_0(h>2)=\beta_0\left(\boxed{\begin{matrix}0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\\0&0&0&0&0\end{matrix}}\right)=0$$

f

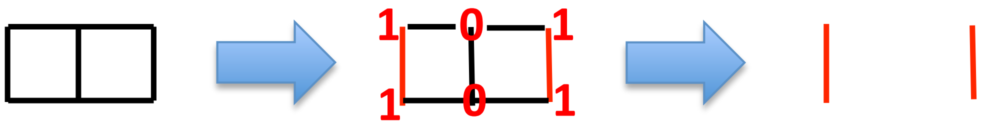
$$\beta_0(h\le 2)=\beta_0\left(\boxed{\begin{matrix}1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\\1&1&1&1&1\end{matrix}}\right)=1$$

Inside β_0: **(recurrent neural network)**



Algorithm to compute β_0(binary image)

Example with 2x3 pixels:



$$\Delta_{ex} = \begin{pmatrix} -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix}$$

- pixel values (0 or 1) determines edges (Hebbian rule)

- Matrix representation of a graph
- -#edge on a node on diagonal

$$\boxed{\Delta v = 0}$$

Parallelizable!

(Next slide)

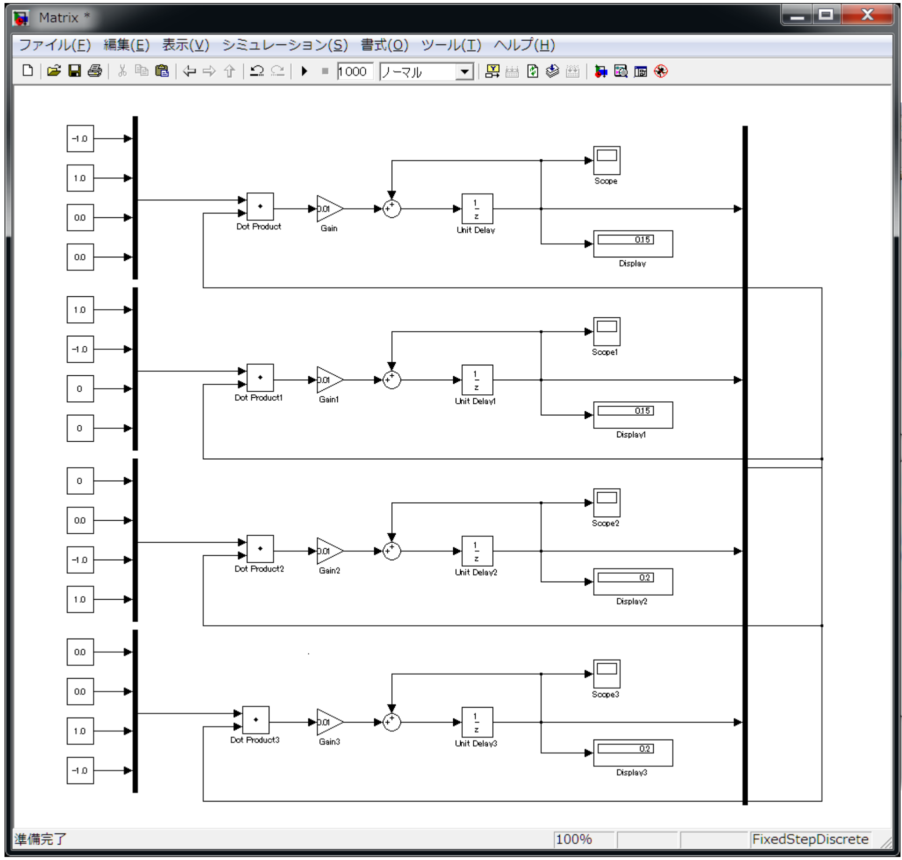
$$v_0 = \begin{pmatrix} a \\ a \\ b \\ b \end{pmatrix}$$

$$\beta_0 = 2$$

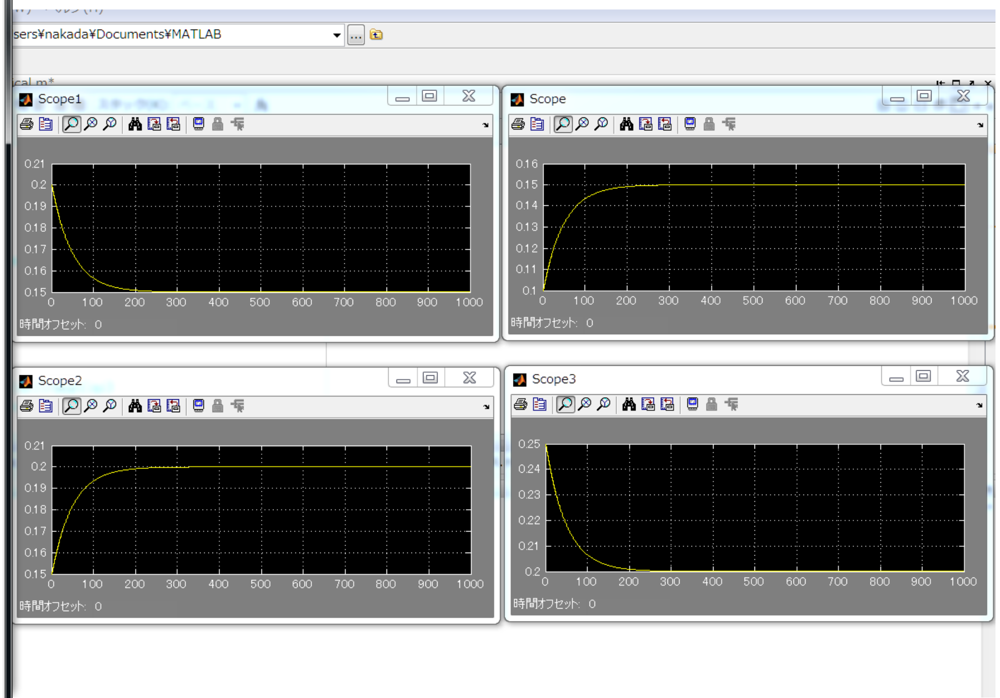
Parallelized Implementation for β_0

$$\Delta_{ex} = \begin{pmatrix} -1 & 1 & 0 & 0 \\ 1 & -1 & 0 & 0 \\ 0 & 0 & -1 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix} \quad \boxed{(I + \alpha\Delta)^n v_0} \quad v_{final} = \begin{pmatrix} a \\ a \\ b \\ b \end{pmatrix}$$

Parallelized!

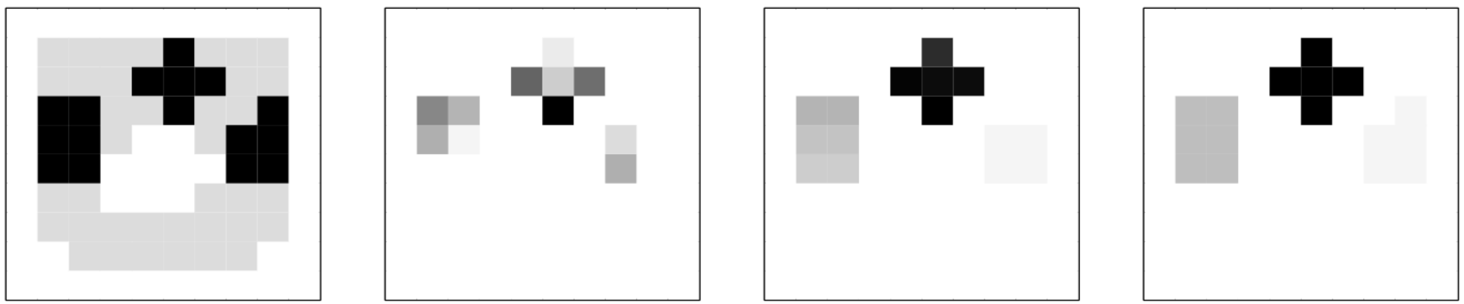


Neurons interact and average their states.

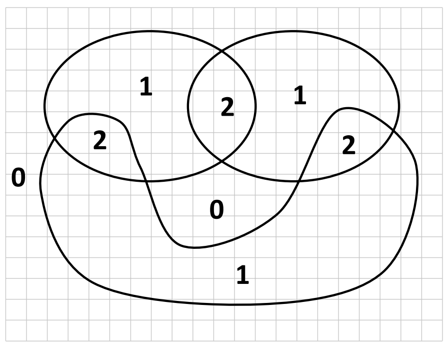


MATLAB HDL Coder → HDL → FPGA

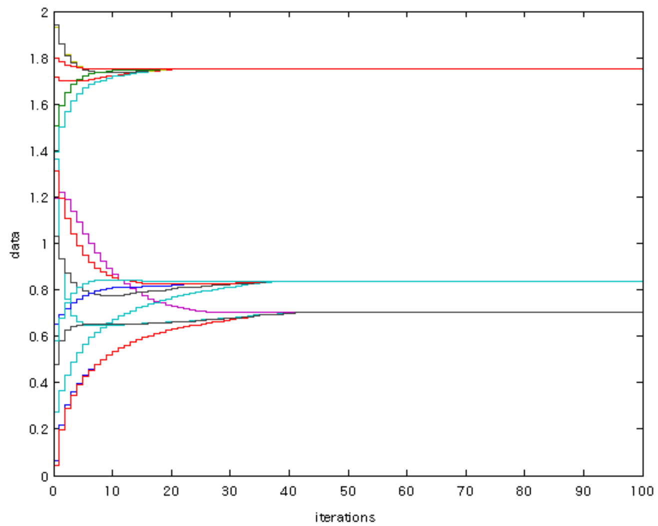
A larger example (10 x 10)



t=0 t=20 t=40



Level set: h=2



Summary

- We propose a parallelized algorithm of shape-invariant touch counter.
  - We use Euler Calculus, a topological method, to realize shape and position invariance.
  - To accelerate, we parallelized the recursive computation of connected component counters, which are elemental sub-modules.

Discussion

- Useful when no time resolution
- The matrix is sparse: O(4N)
- Iterative computation applicable to general β_i
- Integral is unique as it satisfies additivity axiom of measure: $\chi(A \cup B) = \chi(A) + \chi(B) - \chi(A \cap B)$
- Larger problems and parameter tunings for the future works.