An FPGA acceleration of a level set segmentation method

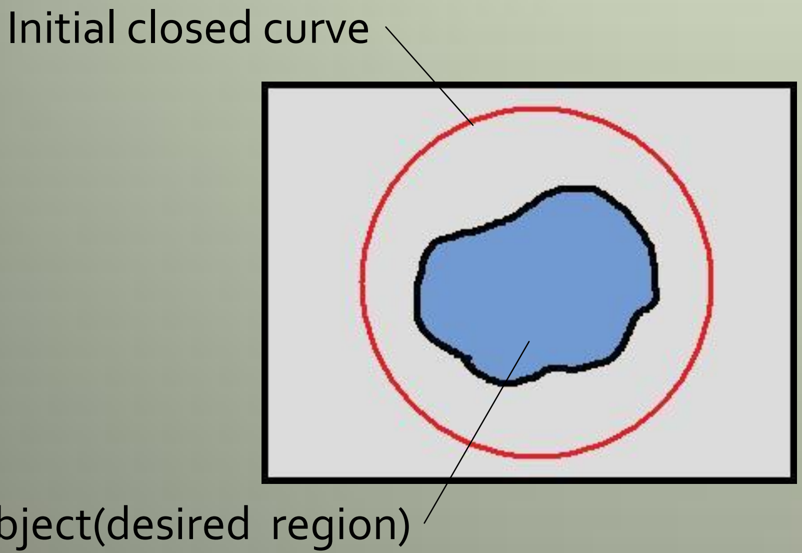
Haruhisa Tsuyama, Tsutomu Maruyama
University of Tsukuba

Purpose of our Research

- The level set method is a numerical technique for tracking interfaces, and often used for image segmentation.
- The purpose of our research is to realize the real-time processing of a level set method on an FPGA.

What is the level set method?

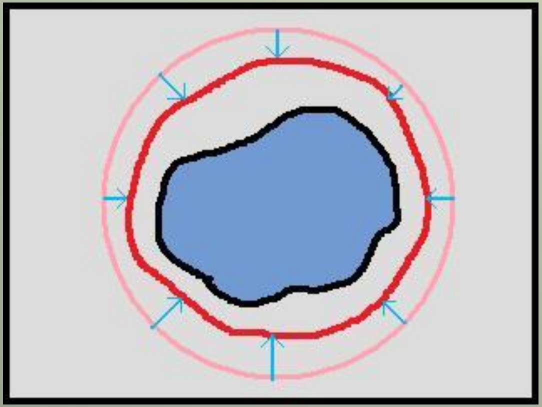
- In the level set method, a closed curve is defined first, and then the curve is gradually evolved in order to detect desired region.



Blue region is the target object, and red circle is the initial closed curve.

What is the level set method?

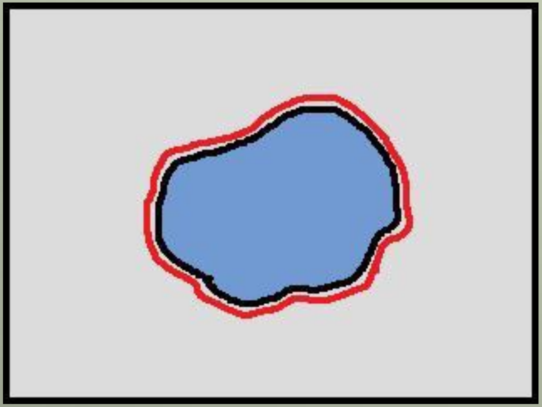
- In the level set method, a closed curve is defined first, and then the curve is gradually evolved in order to detect desired region.



The closed curve is gradually evolved toward the target object.

What is the level set method?

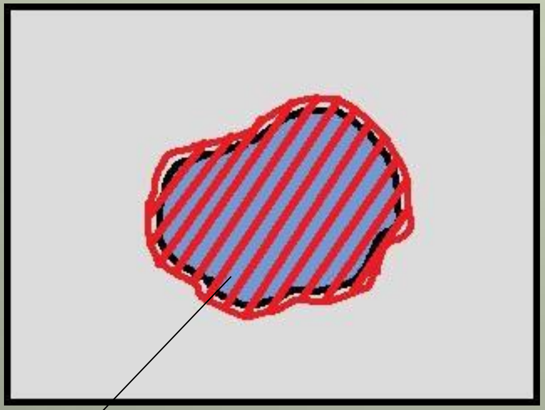
- In the level set method, a closed curve is defined first, and then the curve is gradually evolved in order to detect desired region.



Finally, the closed curve stops at the border of the object.

What is the level set method?

- In the level set method, a closed curve is defined first, and then the curve is gradually evolved in order to detect desired region.



Finally, the closed curve stops at the border of the object.

Detected region

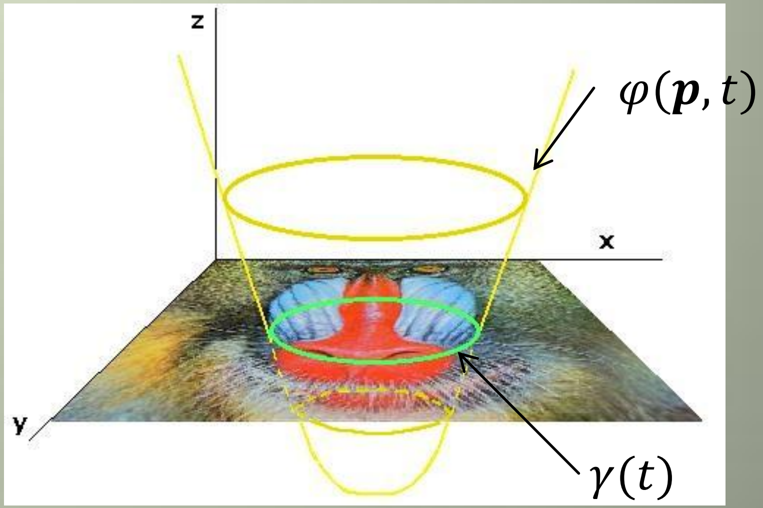
What is the level set method?

- A three-dimensional auxiliary function is used to develop the closed curve.
- The auxiliary function satisfies the following condition.

$$\gamma(t) = \{\boldsymbol{p} | \varphi(\boldsymbol{p}, t) = 0\}.$$

  - $\gamma(t)$ is closed curve.
  - $\varphi(\boldsymbol{p}, t)$ is auxiliary function.

  Where $\boldsymbol{p}$ is a position vector and t is time.



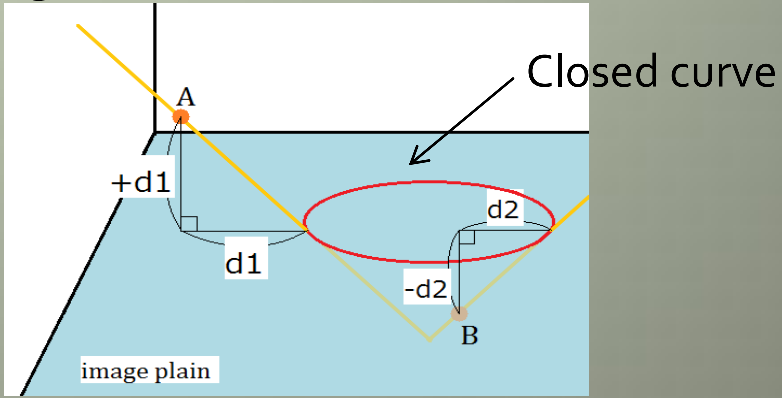
- The closed curve is given as the cross section of the three-dimensional function and the plane of $z = 0$.

What is the level set method?

- In general, the initial value of the auxiliary function is given by the following equation.

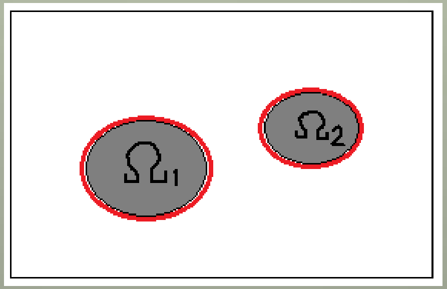
$$\varphi(\boldsymbol{p}, t = 0) = \pm d\,,$$

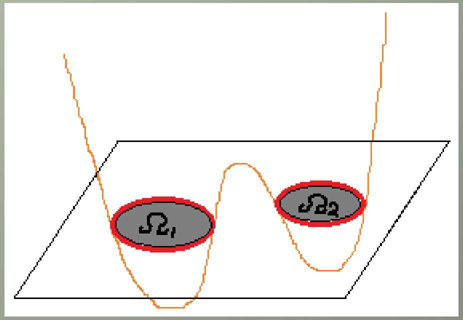
  where d is distance from the point $\boldsymbol{p}$ to $\gamma(t = 0)$ (the closed curve at the time $t = 0$).
- Positive sign is used if the point $\boldsymbol{p}$ is outside the closed curve, and negative sign is used if the point $\boldsymbol{p}$ is inside the closed curve.



What is the level set method?

- The closed curve is evolved on the plane of $z = 0$ by evolving the three-dimensional auxiliary function in three-dimensional space.
- By using the three-dimensional auxiliary function, it becomes possible to follow the topology changes, such as breaking into two, merging two shapes into one.





The evolution of auxiliary function

- The evolution of the auxiliary function is calculated by differentiating the closed curve $\gamma(t)$ with respect to time t.
- The closed curve is represented as $\gamma(t) = \{\boldsymbol{p} | \varphi(\boldsymbol{p}, t) = 0\}$. The following equation is given by differentiating $\gamma(t)$ with respect to t.

$$\varphi_t + \nabla\varphi \cdot \boldsymbol{p}'^{(t)} = 0,$$

Where $\varphi_t = \frac{\partial \varphi}{\partial t}, \nabla\varphi = \left(\frac{\partial}{\partial x}, \frac{\partial}{\partial y}\right)$.

The evolution of auxiliary function

- Let $\vec{n}$ be an outward directed normal to the curve, and the auxiliary function evolves in the direction of $\vec{n}$ with the speed F.
- Then $\vec{n}$ and F can be rewritten as follows

$$\vec{n} = \frac{\nabla\varphi}{|\nabla\varphi|}, F = \boldsymbol{p}'^{(t)} \cdot \vec{n}.$$

- With these equations and the former equation:

$$\varphi_t + \nabla\varphi \cdot \boldsymbol{p}'^{(t)} = 0,$$

we can obtain the equation:

$$\varphi_t + F|\nabla\varphi| = 0.$$

- The auxiliary function is evolved according to this equation.

The evolution of auxiliary function

- The speed function F has to be slowed down to zero on the edges of the desired regions.
- A function $k_I(x, y)$ which becomes smaller around the edges is introduced, and F is multiplied by $k_I(x, y)$:

$$F' = k_I(x, y) \times F.$$

- By using F' as a threshold, the evolution of the auxiliary function can be slowed down around edges.

The evolution of auxiliary function

- In general, $k_I(x, y)$ is given by the following equation:

$$k_I(x, y) = \frac{1}{1 + |\nabla G_\sigma * I(x, y)|}$$

or,

$$k_I(x, y) = e^{-|\nabla G_\sigma * I(x,y)|}.$$

  - Where $|\nabla G_\sigma * I(x, y)|$ is the brightness gradient obtained after applying a Gaussian filter of variance σ.

- Around the edges, $k_I(x, y)$ becomes smaller because the brightness gradient becomes bigger.

The evolution of auxiliary function

- To calculate the evolution of the auxiliary function, the variables in the auxiliary function are discretized as follows:

$$\varphi(\boldsymbol{p}, t) = \varphi(ih, jh, n\Delta t).$$

  - Where i, j, n are integers, and $h, \Delta t$ are step sizes of $\boldsymbol{p} = (x, y)$ and t.

- Then the equation:

$$\varphi_t + F|\nabla\varphi| = 0$$

can be rewritten as

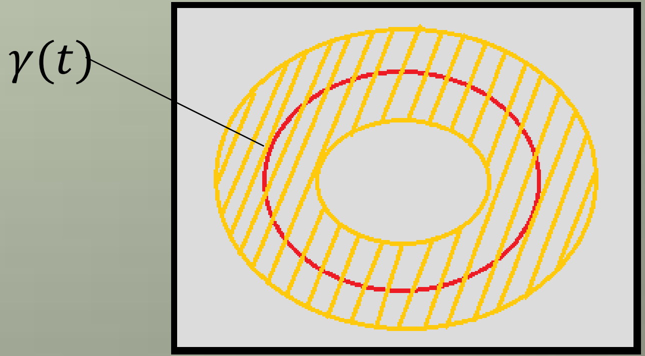
$$\varphi_{ij}^{n+1} = \varphi_{ij}^{n} - \Delta t F\left|\nabla_{ij}\varphi_{ij}^{n}\right|.$$

  - With this equation, the auxiliary function in time $n + 1$ can be obtained from the auxiliary function in time n.

A method to reduce the computational complexity

- The computational complexity to evolve the auxiliary function over all image is very high.
- To reduce computational complexity, narrow band method was proposed.
- In the narrow band method, the auxiliary function is evolved only around the closed curve.
- But, it is hard to implement it on hardware because of the irregular memory access required in the narrow band method.

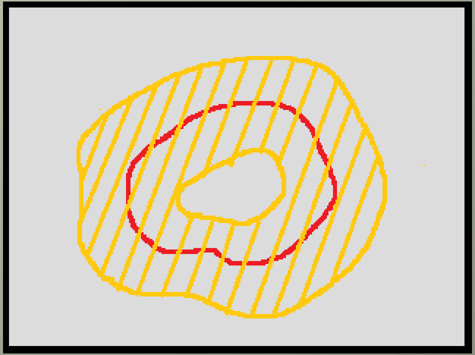
The auxiliary function is evolved only in orange region (narrow band).



A method to reduce the computational complexity

- The computational complexity to evolve the auxiliary function over all image is very high.
- To reduce computational complexity, narrow band method was proposed.
- In the narrow band method, the auxiliary function is evolved only around the closed curve.
- But, it is hard to implement it on hardware because of the irregular memory access required in the narrow band method.

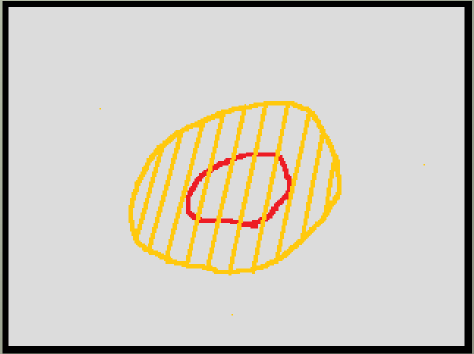
The shape of narrow band is changed according to the closed curve.



A method to reduce the computational complexity

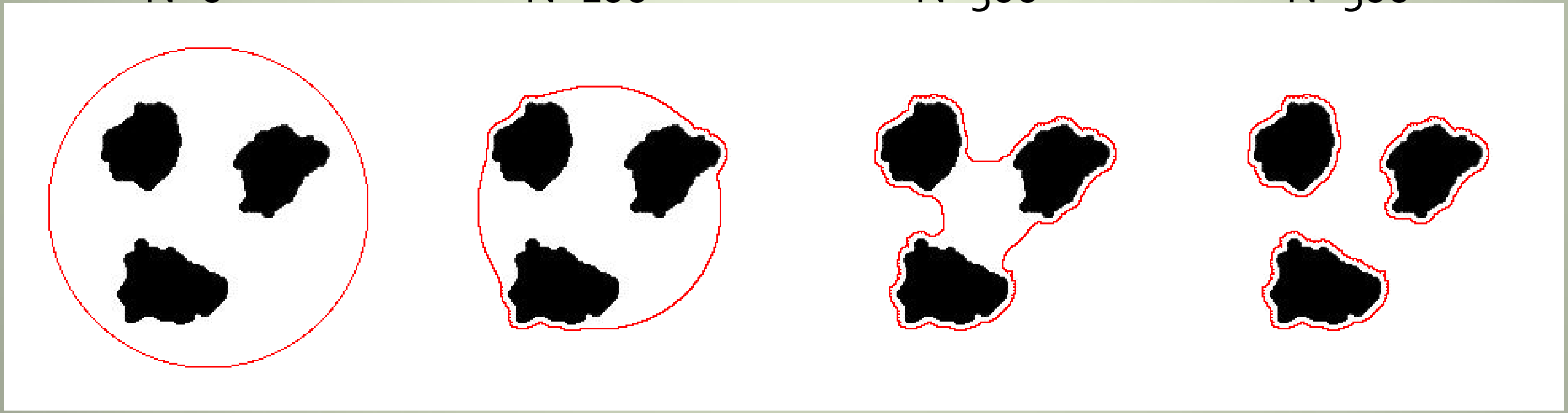
- The computational complexity to evolve the auxiliary function over all image is very high.
- To reduce computational complexity, narrow band method was proposed.
- In the narrow band method, the auxiliary function is evolved only around the closed curve.
- But, it is hard to implement it on hardware because of the irregular memory access required in the narrow band method.

The shape of narrow band is changed according to the closed curve.

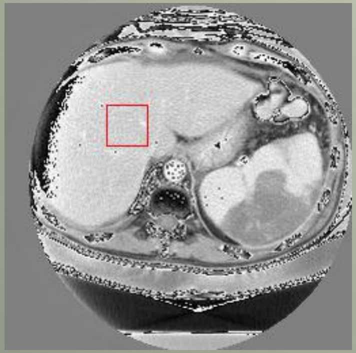


Result of the narrow band method

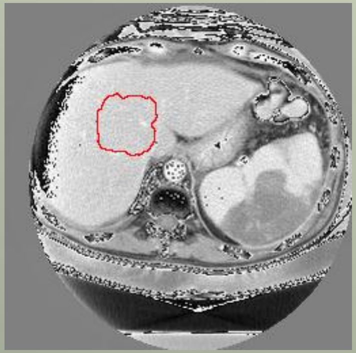
N=0 N=100 N=300 N=500



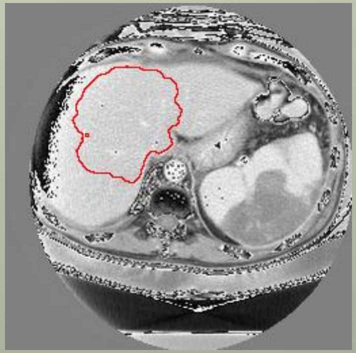
N=0



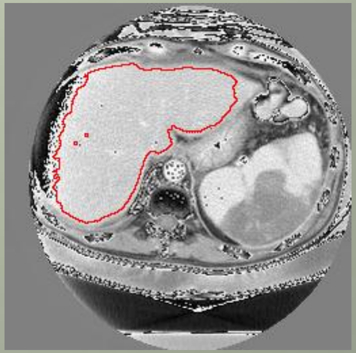
N=50



N=200



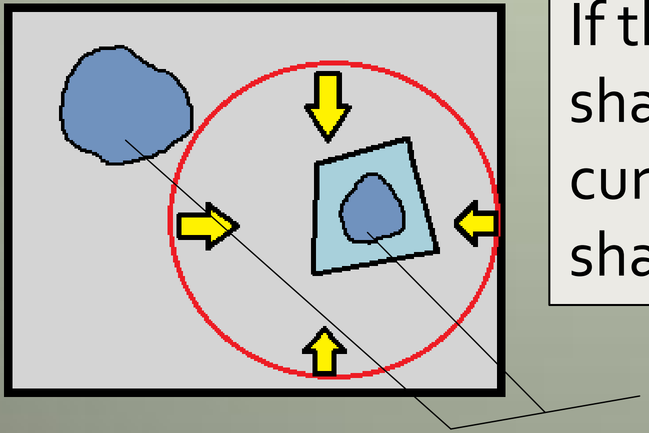
N=500



N is the number of repetition in processing.

Problems of the narrow band method

1. Its computational complexity is still high.
   - Even with the narrow band method, it is hard to achieve real-time processing.
2. Some shapes may not be detected according to the starting point of the closed curve.



If the closed curve is grown inwards, the shapes which is (1) out of the closed curve, or (2) surrounded by other shapes cannot be detected.

Our approach

- We propose a new level set algorithm to solve these problems.
- Our algorithm is designed so as to allow deep pipelining on hardware systems, and able to detect all objects in the image.

Our algorithm

- In our algorithm, the following two changes are added to the original algorithm.
- First, F is fixed to 1. (F is the evolving speed of the auxiliary function).

With this change, the equation:

$$\varphi_{ij}^{n+1} = \varphi_{ij}^{n} - \Delta t F' \left| \nabla_{ij} \varphi_{ij}^{n} \right|$$

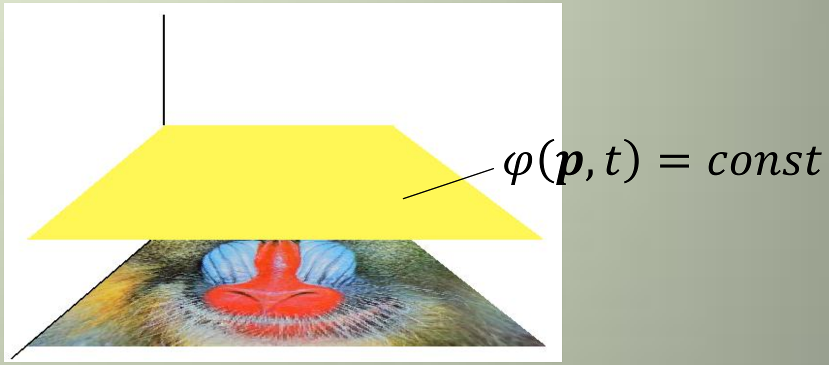
can be rewritten as

$$\varphi_{ij}^{n+1} = \varphi_{ij}^{n} - \Delta t k_{I_{ij}} \left| \nabla_{ij} \varphi_{ij}^{n} \right|.$$

Our algorithm

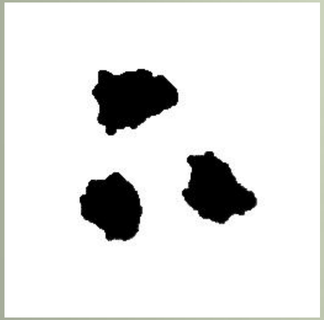
- Second, the initial auxiliary function is given as a flat plane.

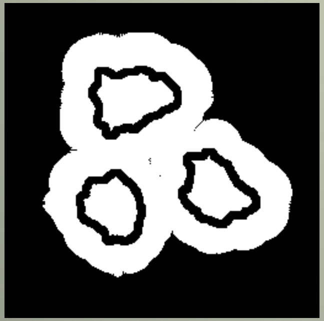
$$\varphi(\boldsymbol{p}, t = 0) = const.$$

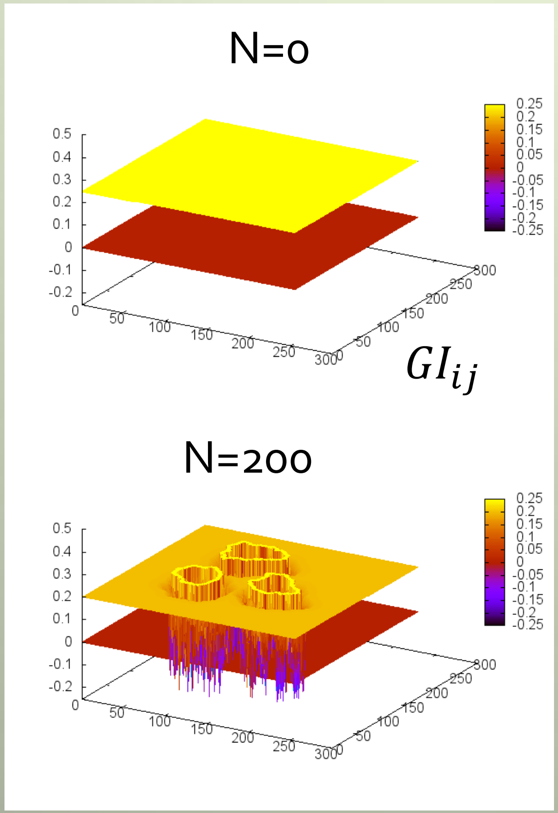


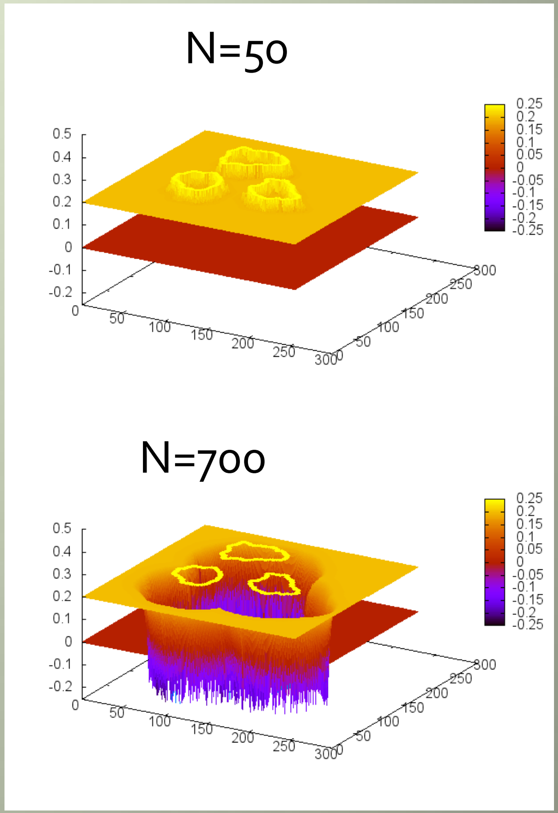
- The auxiliary function is evolved over all image.
  - In the narrow band method, the auxiliary function is evolved only around the closed curve
- The closed curves are automatically generated while the auxiliary function is evolved.

The auxiliary function in our algorithm


input image


output image





The yellow plane is the initial auxiliary function and the red plane is the image.
In our algorithm, the regions with $\varphi > 0$ correspond to the borders of the objects, and the region with $\varphi < 0$ correspond to the objects, or their background.

The procedure of our algorithm

- Step 1.

Apply the Gaussian filter to the input image as follows:

$$GI_{ij} = I_{ij} * G_\sigma ,$$

where G_σ is a Gaussian filter of variance σ and GI_{ij} is the image after applying the Gaussian filter.

The procedure of our algorithm

- Step2.

  Calculate $k_{I_{ij}}$ from its four neighbor pixels.

$$I_{x_{ij}} = \frac{\left(GI_{i+1,j} - GI_{i-1,j}\right)}{2}, I_{y_{ij}} = \frac{\left(GI_{i,j+1} - GI_{i,j-1}\right)}{2}$$

$$\left|\nabla_{ij} I_{ij}\right| = \sqrt{{I_{x_{ij}}}^2 + {I_{y_{ij}}}^2}$$

$$k_{I_{ij}} = \frac{1}{1 + \left|\nabla_{ij} I_{ij}\right|} \quad \text{or} \quad k_{I_{ij}} = e^{-\left|\nabla_{ij} I_{ij}\right|}$$

- In case of color images, I_x and I_y are calculated as $I_x \leftarrow I_{R_x} + I_{G_x} + I_{B_x}$ and $I_y \leftarrow I_{R_y} + I_{G_y} + I_{B_y}$.

The procedure of our algorithm

- Step3.

  Calculate $|\nabla_{ij}\varphi_{ij}^{n}|(n \geq 1)$ from its four neighbor pixels.

$$\varphi_{x_{ij}}^{n} = \frac{\left(\varphi_{i+1,j}^{n} - \varphi_{i-1,j}^{n}\right)}{2}$$

$$\varphi_{y_{ij}}^{n} = \frac{\left(\varphi_{i,j+1}^{n} - \varphi_{i,j-1}^{n}\right)}{2}$$

$$|\nabla_{ij}\varphi_{ij}^{n}| = \sqrt{{\varphi_{x_{ij}}^{n}}^{2} + {\varphi_{y_{ij}}^{n}}^{2}}$$

The procedure of our algorithm

- Step4.

  Calculate φ_{ij}^{n+1} by the equation:

$$\varphi_{ij}^{n+1} = \varphi_{ij}^{n} - \Delta t k_{I_{ij}} \left| \nabla_{ij} \varphi_{ij}^{n} \right|.$$

- When $n = 0$, φ_{ij}^{n+1} is calculated by the following equation because $\left| \nabla_{ij} \varphi_{ij}^{0} \right| = 0$ at this time.

$$\varphi_{ij}^{n+1} = \varphi_{ij}^{n} - \Delta t k_{I_{ij}}$$

The procedure of our algorithm

- Step 5.

  Repeat step3(calculate$|\nabla_{ij}\varphi_{ij}^{n}|$) and step4(Calculate φ_{ij}^{n+1}) until the number of repetition reaches to the fixed number N, or all evolution for $\varphi > 0$ stop.

The procedure of our algorithm

Summary

- Step 1.

  Apply the Gaussian filter to the input image.

- Step 2.

  Calculate $k_{I_{ij}}$ from its four neighbor pixels.

- Step 3.

  Calculate $|\nabla_{ij}\varphi_{ij}^{n}|(n \geq 1)$ from its four neighbor pixels.

- Step 4.

  Calculate φ_{ij}^{n+1} by the equation:

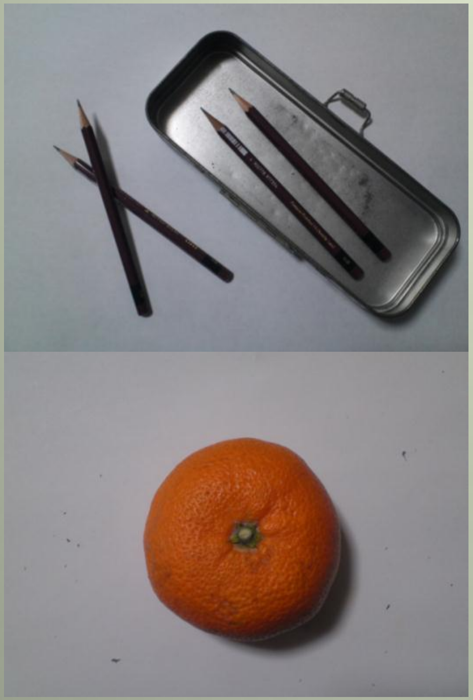
$$\varphi_{ij}^{n+1} = \varphi_{ij}^{n} - \Delta t k_{I_{ij}} \left|\nabla_{ij}\varphi_{ij}^{n}\right|.$$

- Step 5.

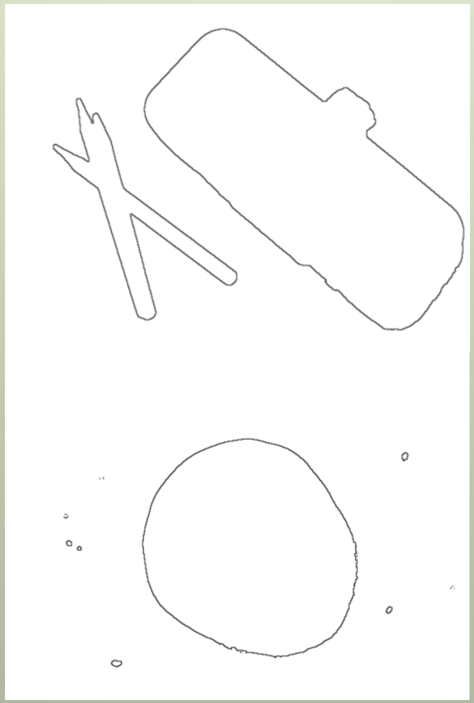
  Repeat step3 and step4 until the end condition is satisfied.

Comparison of our algorithm and the original algorithm

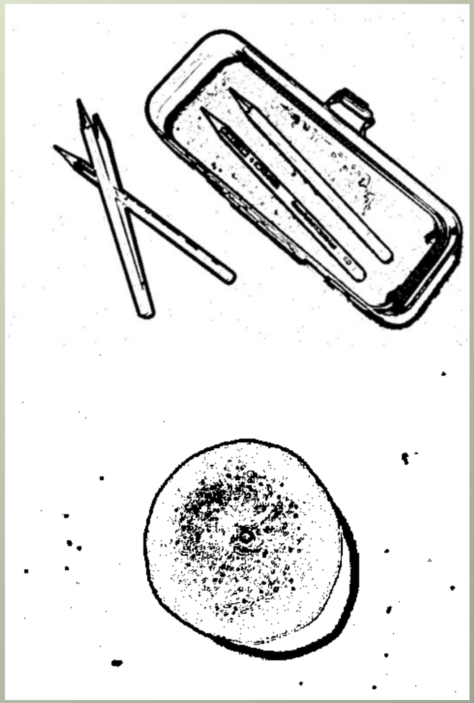
input image



original algorithm



our algorithm

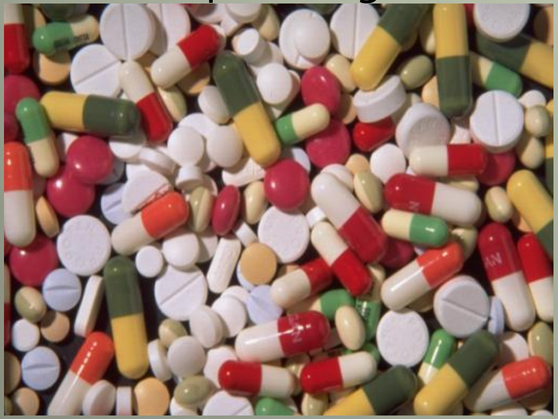


Our algorithm can detect the shapes of the objects surrounded by other objects, but too many segments are detected on the objects.

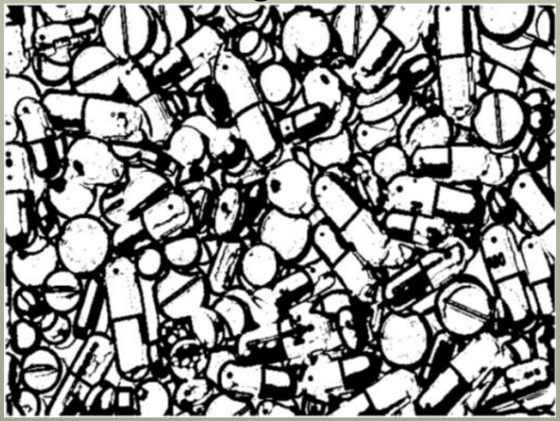
Difference by the number of repetition

- By increasing number of repetition, small segments can be eliminated.
- But, it may fail to detect the boundaries with the small change of the brightness.

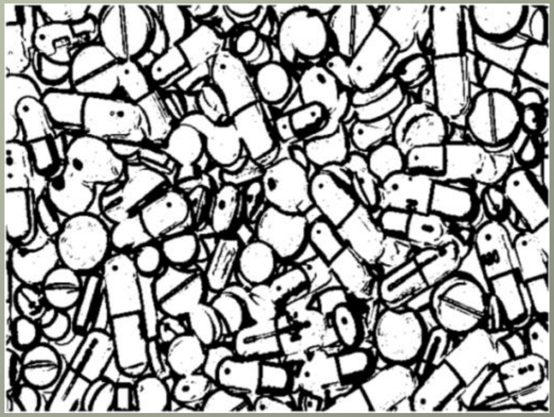
input image



N=50

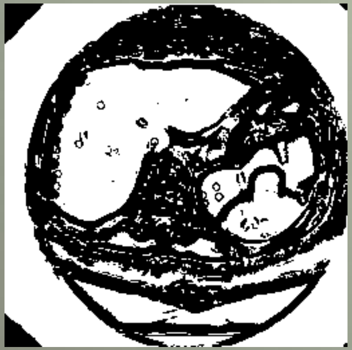


N=1000

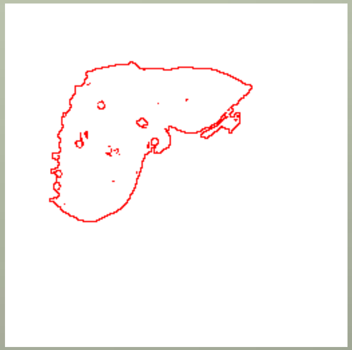


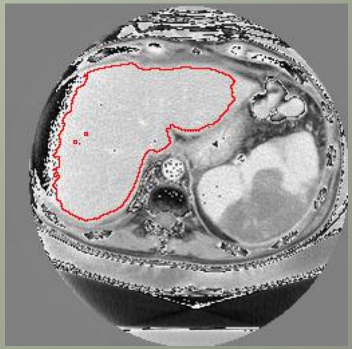
Obtaining the same result as the original method

- With our algorithm, almost same result as the original level set method can be obtained by specifying a background or desired region afterward (as being done in the original method before starting the computation).


The result of our algorithm

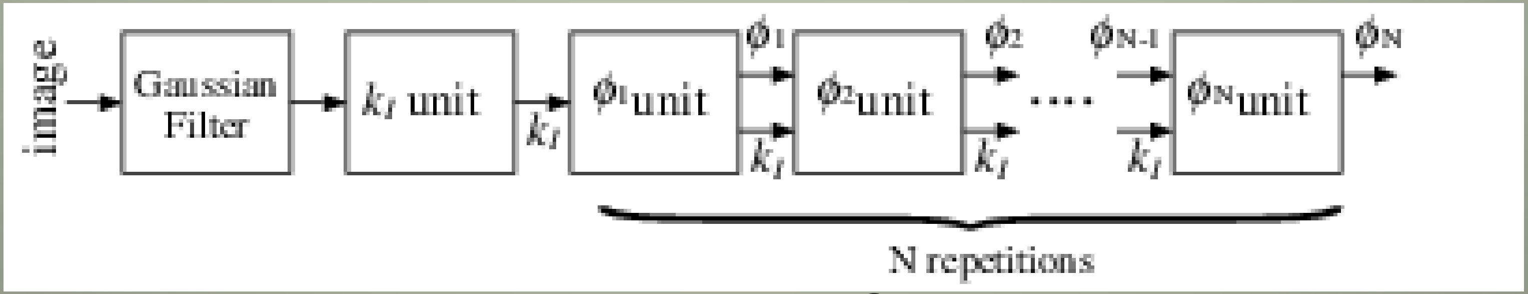
Specifying desired region $\longrightarrow$




The result of original algorithm

Hardware implementation

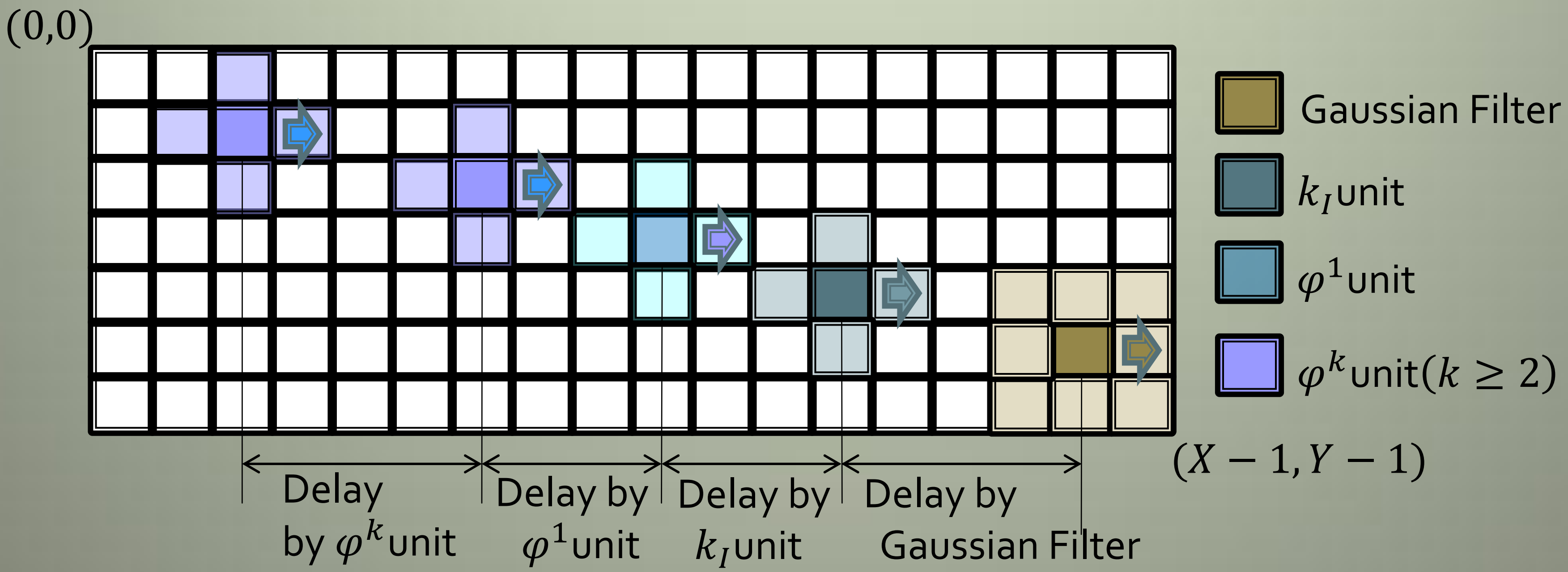
- The steps in our algorithm can be grouped as follows.
  - 1. Apply a Gaussian filter.
  - 2. Calculate $k_{I_{ij}}$.
  - 3. Calculate $\varphi_{ij}^{n+1} = \varphi_{ij}^{n} - \Delta t k_{I_{ij}} \left| \nabla_{ij} \varphi_{ij}^{n} \right|$ repeatedly.
- The system can be constructed by three types of units.



A block diagram of our circuit.

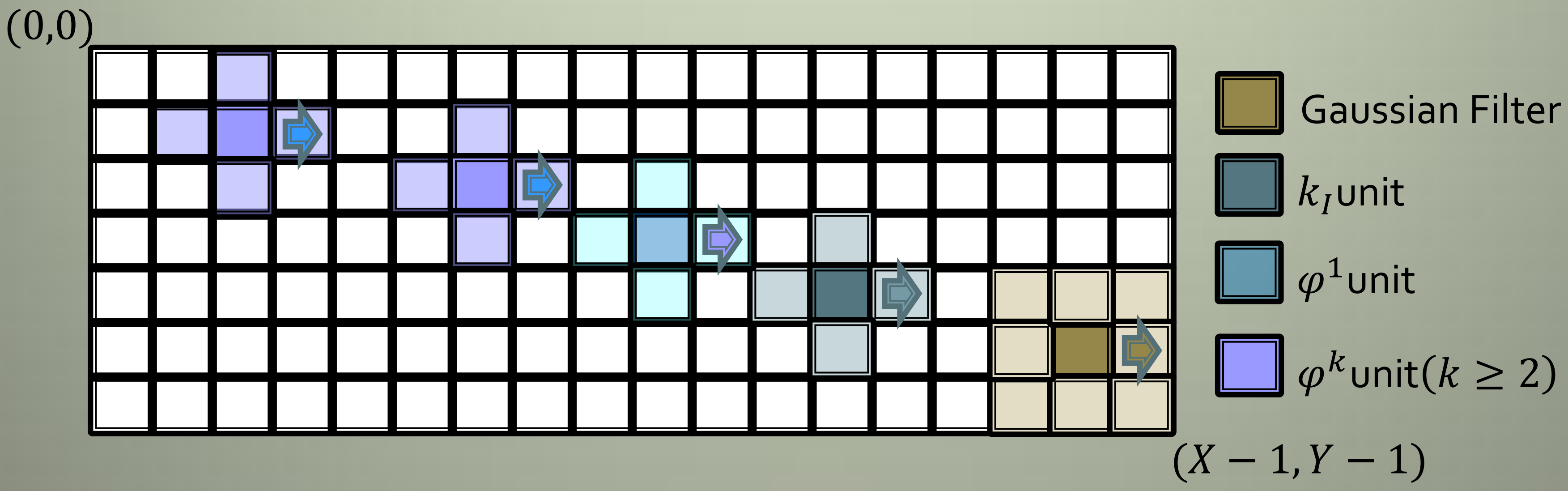
Pipeline processing

- All units scan the image from $(0,0)$ to $(X-1, Y-1)$.
- Each unit can calculate its output using only four (eight) neighbor values (the outputs of its previous unit).

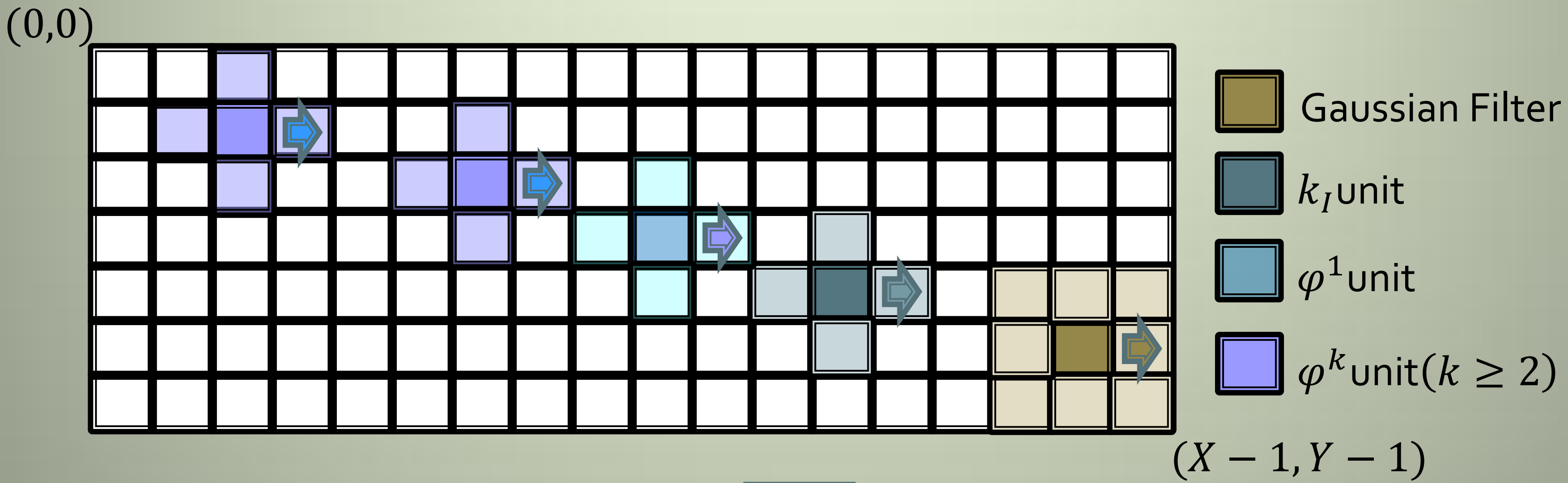


Pipeline processing

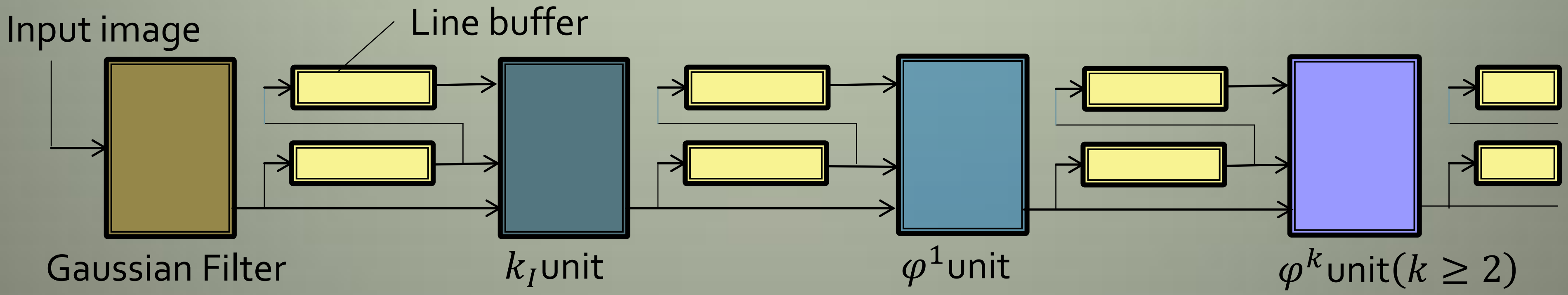
- These pixels can be processed on the deeply pipelined circuit.



Pipeline processing

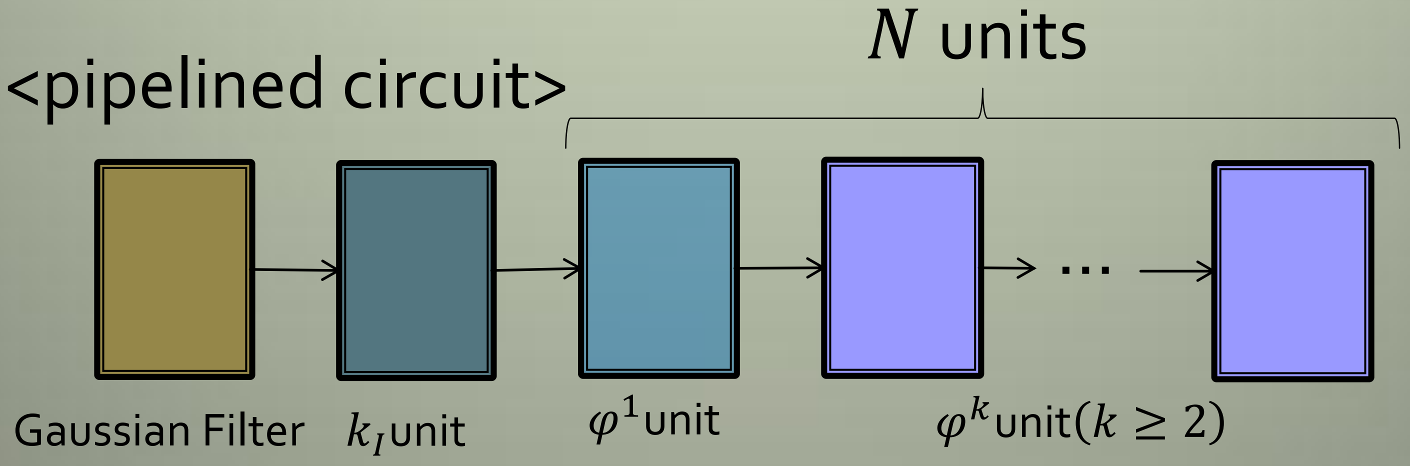


<pipelined circuit>



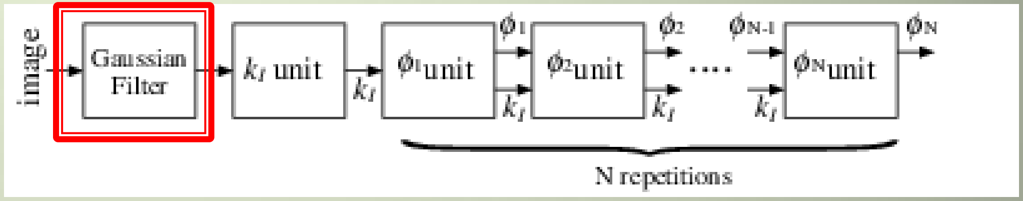
Pipeline processing

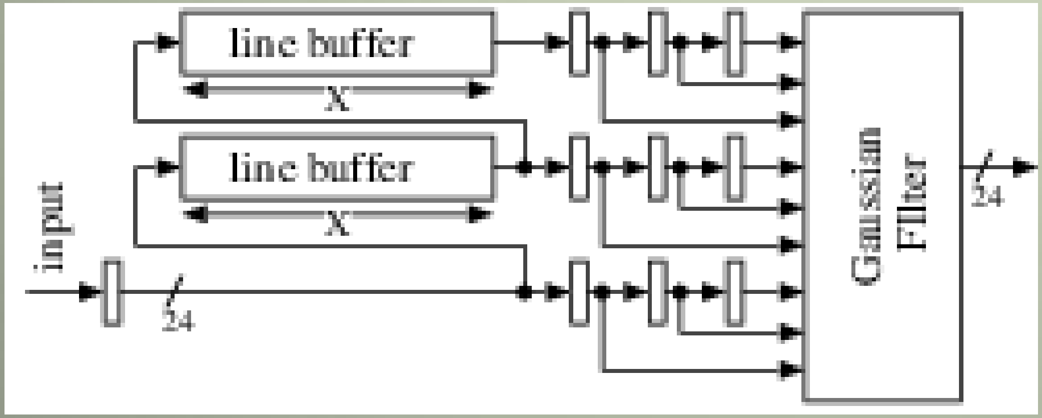
- If the number of φ unit is N (which means that the repetition number is N) , the pipelined circuit processes $N+2$ pixels in parallel.



Details of each unit

- Gaussian Filter

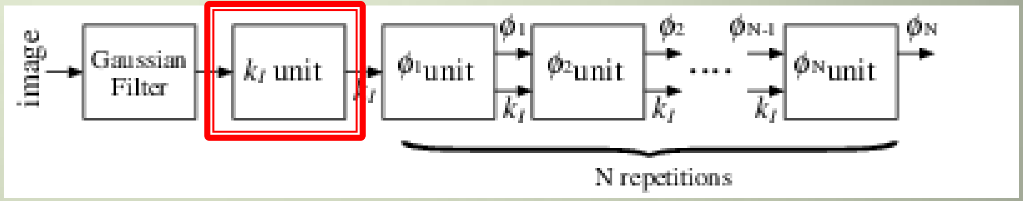


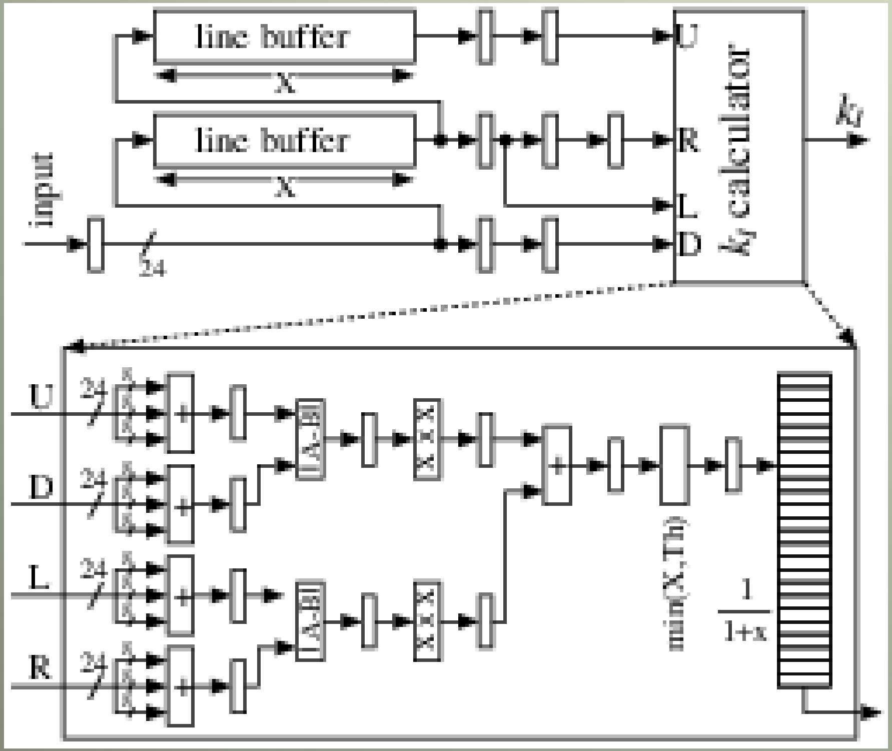


Nine pixels on three lines are held on the register array.
Two line buffers are used to supply the data of the three lines.

Details of each unit

- k_I unit

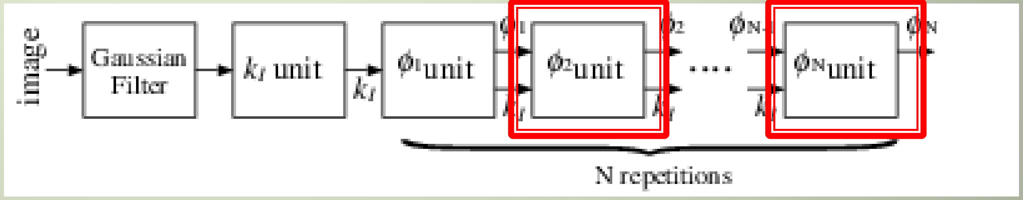


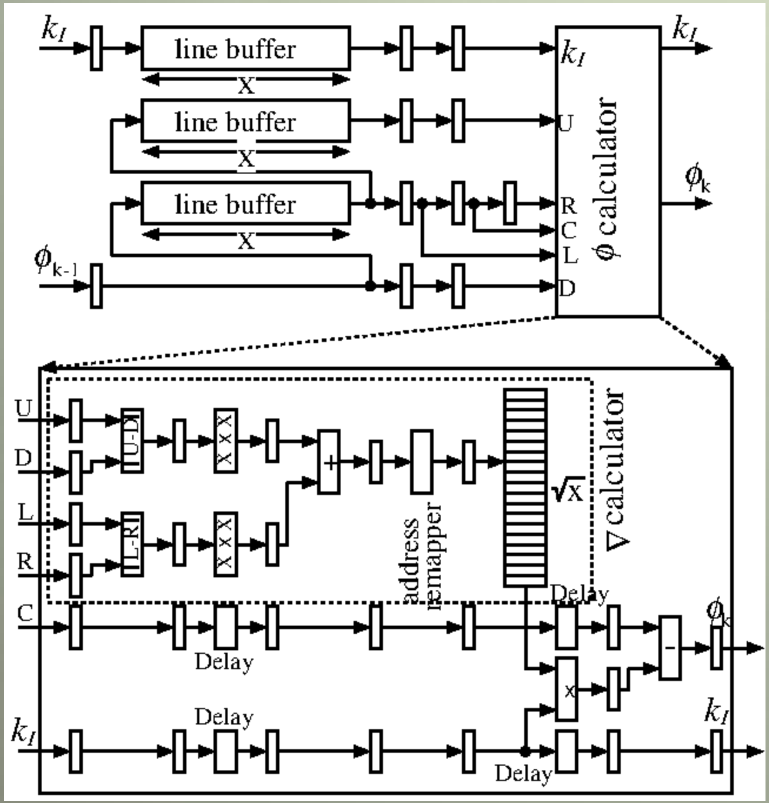


k_I is calculated using a register array which holds its four neighbor pixels. Two line buffers are used. An array is used to calculate $\frac{1}{1+x}$.

Details of each unit

- φ unit





φ is also calculated using a register array which holds its four neighbor pixels.
Two line buffers are also used to supply the data of the three lines, and another is used to control delays.
An array is used to calculate $\sqrt{x}$.

Experimental results

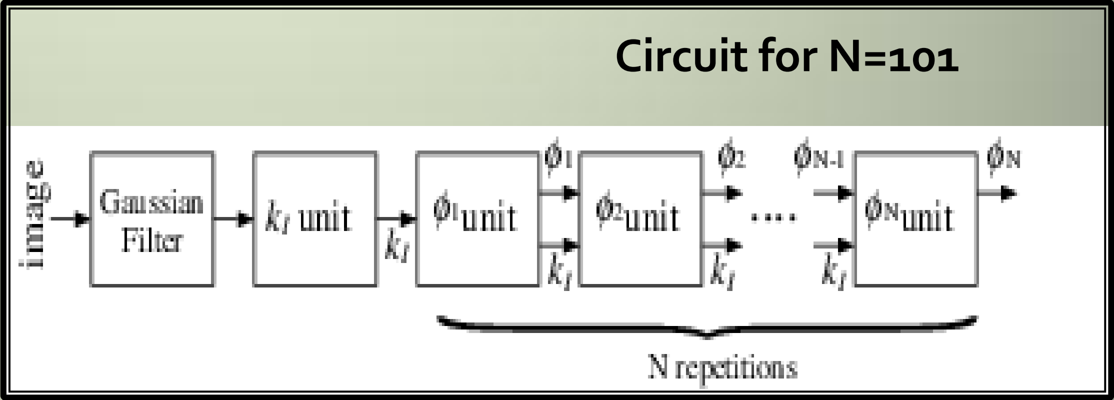
- We have implemented the circuit for $N = 31, 51, 101$ on Xilinx XC4VLX160.
  - Target image size is 640×480.
  - Operating frequency is 252.4MHz.
- The square and multiply operations are implemented using LUTs (in order to avoid multiplier bottleneck).

N		31	51	101
LUTs(K)		19.8	32.8	64.9
BRAMs		67	107	207
Performance (fps)	one frame	769	739	674
	throughput	822	822	822

Experimental results

- By running our circuit m times, we can obtain $\varphi^k (k > N)$.

INPUT IMAGE



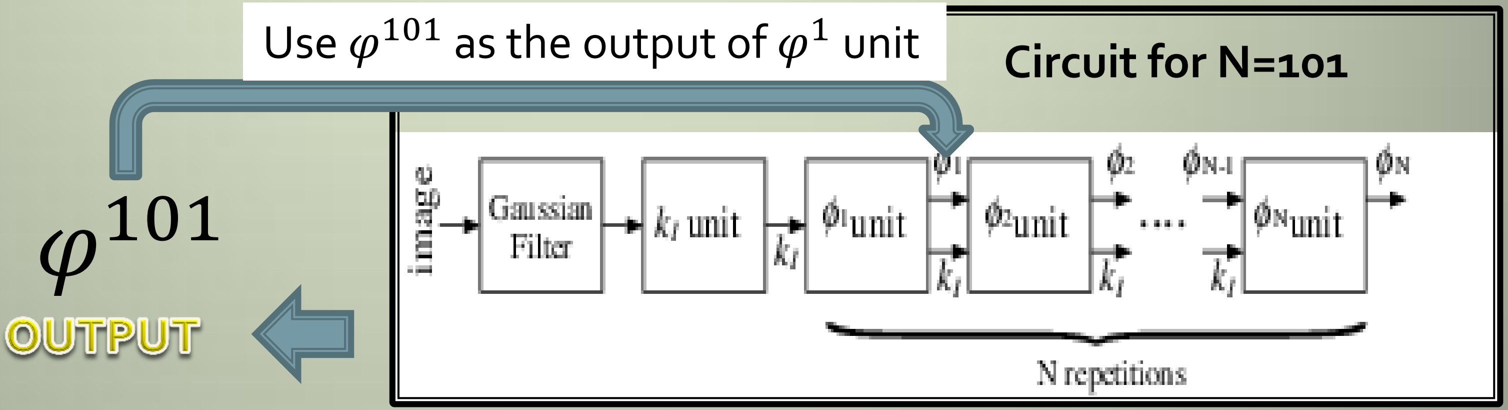
φ^{101}

OUTPUT

- When the image size is 640×480, up to $N = 2201$ can be calculated in real-time(30 fps) by running our circuit 22 times.
  - By running the circuit m times, the performance becomes one-mth.

Experimental results

- By running our circuit m times, we can obtain $\varphi^k (k > N)$.

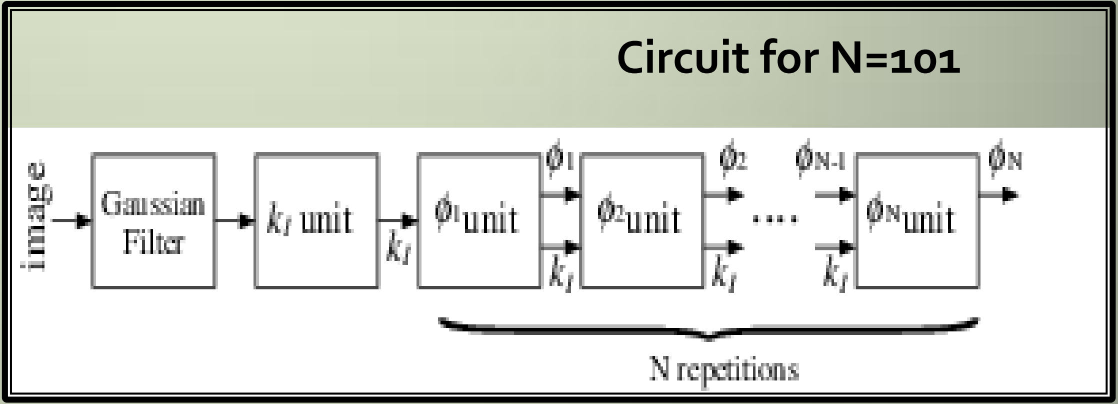


- When the image size is 640×480, up to $N = 2201$ can be calculated in real-time(30 fps) by running our circuit 22 times.
  - By running the circuit m times, the performance becomes one-mth.

Experimental results

- By running our circuit m times, we can obtain $\varphi^k (k > N)$.

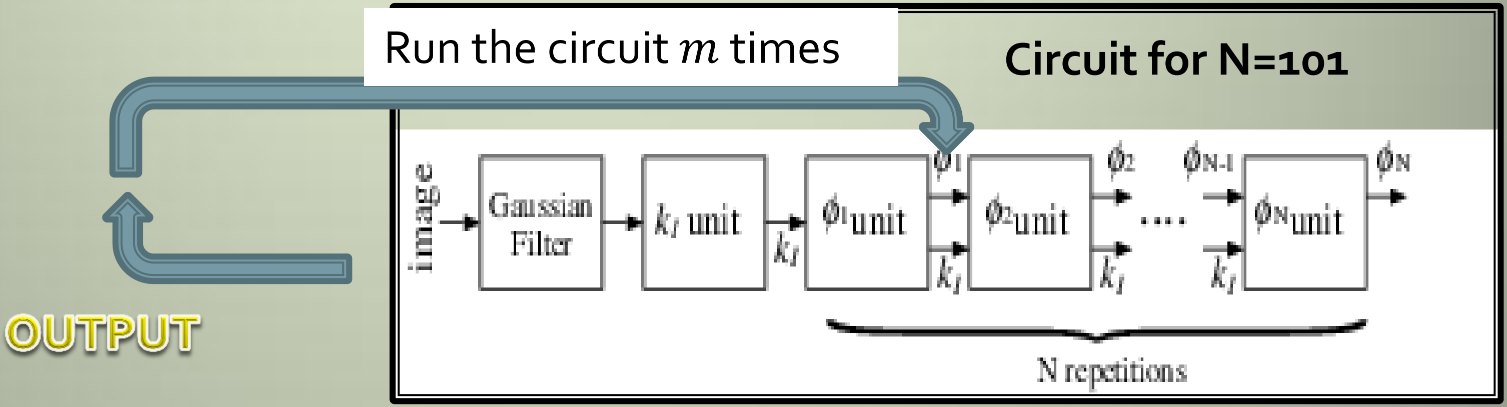
$$\varphi^{101+(101-1)}$$



- When the image size is 640×480, up to $N = 2201$ can be calculated in real-time(30 fps) by running our circuit 22 times.
  - By running the circuit m times, the performance becomes one-mth.

Experimental results

- By running our circuit m times, we can obtain $\varphi^k (k > N)$.



- When the image size is 640×480, up to $N = 2201$ can be calculated in real-time(30 fps) by running our circuit 22 times.
  - By running the circuit m times, the performance becomes one-mth.

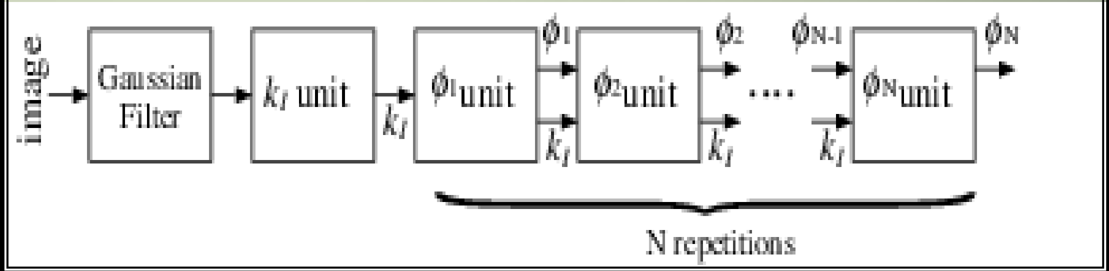
Experimental results

- By running our circuit m times, we can obtain $\varphi^k (k > N)$.

$$\varphi^{101+(m-1)(101-1)}$$

OUTPUT

Circuit for N=101



- When the image size is 640×480, up to $N = 2201$ can be calculated in real-time(30 fps) by running our circuit 22 times.
  - By running the circuit m times, the performance becomes one-mth.

Experimental results

- When the image width is X and $N + 2$ pixels are processed in parallel on the pipelined circuit, the number of block RAMs becomes as follows.

  When $X \leq 1024$

  $$7 + 2(N - 1),$$

  when $X > 1024$

  $$13 + \frac{7(N-1)}{2}.$$

- When the image size is 1920×1080 and $N = 101$, 363 block RAMs are required.
- If we want to implement this circuit on a small FPGA, we should implement the circuit for $N = 51$ and run it twice. Then, the number of block RAMs becomes 188.(However the performance becomes half.)

Conclusions

- We have described a new level set algorithm and its FPGA implementation.
- With our algorithm, it becomes possible to detect all objects in the image, which is difficult for previous level set algorithms.
- In addition, we have shown that real-time processing is possible even with a small size FPGA.