Filtros Morfológicos de Imágenes Binarias

Ejercicio N° 2

Objetivos	• Diseño y construcción de sistemas con procesamiento concurrente • Diseño y construcción de sistemas con acceso distribuido • Encapsulación de Threads y Sockets en TDAs • Protección de los recursos compartidos
Instancias de Entrega	**Entrega 1:** clase 6 (20/09/2016). **Entrega 2:** clase 8 (04/10/2016).
Temas de Repaso	• Threads en C++11 • Clases en C++11
Criterios de Evaluación	• Criterios de ejercicios anteriores • Ausencia de secuencias concurrentes que permitan interbloqueo • Ausencia de condiciones de carrera en el acceso a recursos • Buen uso de Mutex, Condition Variables y Monitores para el acceso a recursos compartidos

Índice

Introducción

El filtrado morfológico es un procesamiento de imagen basado en las formas locales de estas.
A pesar de ser herramientas sumamente simples cumplen un rol esencial en las etapas previas y posteriores de otros algoritmos más avanzados como la detección de contornos y blobs.

Descripción

Se implementarán dos operaciones elementales de filtros: la dilatación y la erosión. Para darle soporte a filtros más complejos y útiles como filtros de apertura y de cierre se implementará un mecanismo de encadenado de filtros.

Los filtros procesaran imágenes binarias que serán representadas en ASCII con el carácter **.** como un **0** y con **#** como un **1**.

Todos los filtros toman como entrada la imagen inicial o fuente y un fragmento de imagen conocido como patrón o estructura (*structuring element*) . Como se usa este patrón depende de cada filtro.
Para la simplificación del trabajo todos los patrones tendrán la misma cantidad de filas que de columnas y esas cantidades serán un número impar. En otras palabras los patrones serán 1x1, 3x3, 5x5, 7x7.

En la pagina web
https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm se muestran con mayor detalle los conceptos de patron, superposicion, coincidencia, dilatacion y erosion.

A continuación hay un breve resumen de esos conceptos pero se aconseja leer el la página web mencionada para tener un entendimiento más amplio.

Superposición del patrón o estructura

Dada una imagen, se dice que el patrón coincide totalmente (fit) en el bit (i,j) si *centramos* al patron en esa posicion y vemos que en todas las posición de los bits 1 del patrón hay un bit 1 en la imagen fuente.
Se dice que el patron coincide parcialmente (intersect) si hay al menos 1 bit en común.

Por ejemplo, sea el patrón 3x3

```
###
###
###
```

Y sea la imagen 6x6:

```
###...
###...
###...
```

Solo en la posición (fila 2, columna 2) tenemos una coincidencia total mientras que en las posiciones (5,1), (5, 2), (5, 3), (6,1), (6,2) y (6,3) son las únicas posiciones en donde no hay ninguna coincidencia. En el resto de las posiciones hay una coincidencia parcial.

Filtro de dilatación

Dada una imagen binaria fuente, por cada posición (i,j) se *centra* el patrón o estructura y si hay una *coincidencia parcial* se emite un **bit 1** en la posición (i,j) de la imagen destino.

Para mostrar un ejemplo, sea la estructura de 3x3:

```
.#.
###
.#.
```

y sea la imagen fuente de 6x6:

```
......
.#....
......
......
...#..
......
```

el resultado nos da una imagen final de 6x6:

```
.#....
###...
.#....
...#..
..###.
...#..
```

Nótese como los **bits 1** originales se expandieron o dilataron. De ahí el nombre del filtro.

En los casos bordes se tiene:

Siendo la imagen fuente de 6x6

```
#....#
......
..#...
......
......
#....#
```

y usando la misma estructura, la imagen final de 6x6 es:

```
##..##
#.#..#
.###..
..#...
#....#
##..##
```

Nótese como el patrón o estructura se aplicó parcialmente en las esquinas pero en ningun momento la imagen fue agrandada: sigue siendo de 6x6.

Filtro de erosion

El filtrado de erosión viene a funcionar de forma inversa al de dilatación. Por cada posición (i,j) en la imagen fuente se *centra* el patrón o estructura y se emite un **bit 1** en (i,j) de la imagen destino si hay una *coincidencia total*:

Para mostrar un ejemplo, sea la estructura de 3x3:

```
.#.
###
.#.
```

y sea la imagen fuente de 6x6:

```
.#....
###...
.#....
..####
..####
..####
```

el resultado es una imagen final de 6x6:

```
......
.#....
......
......
...##.
......
```

Nótese solo como los **bits 1** en las coordenadas (fila 2, columna 2), (5, 4) y (5, 5) de la imagen fuente son los únicos bits que al centrar y superponer el patrón tienen una coincidencia total y por ende son los unicos **bits 1** que son copiados a la imagen destino.

Una interpretación de este filtrado es que remueve los contornos, de ahí el nombre de erosión.

En los casos bordes el patrón o estructura *nunca* tiene una coincidencia:

Siendo la imagen fuente de 6x6

```
##..##
#.#..#
.###..
..#...
#....#
##..##
```

y usando la misma estructura, la imagen final de 6x6 es:

```
......
......
..#...
......
......
......
```

Encadenado de filtros: filtros de apertura y de cierre

Los filtros de dilatación y erosión pueden ser combinados para generar filtros más interesantes como los filtros opening (apertura) y closing (cierre).

El filtro de opening consiste en aplicar un filtro erosión seguido de uno de dilatación usando la misma estructura o patrón.

Para mostrar un ejemplo, sea la estructura de 3x3:

```
###
###
###
```

y sea la imagen fuente de 10x6:

```
..........
..###.###.
..#######.
..###.###.
...#......
......#...
```

el resultado (parcial) luego de la erosión es:

```
..........
..........
...#...#..
..........
..........
..........
```

y luego de la dilatación tenemos:

```
..........
..###.###.
..###.###.
..###.###.
..........
..........
```

Nótese como el filtro de opening hace que islas de **bits 1s** se separen entre sí si están unidas apenas por unos pocos **bits 1**.

Por el otro lado, el filtro de closing aplica un filtro de dilatación seguido por uno de erosión:

Sea la imagen fuente de 10x10:

```
..........
.###..###.
.###..###.
.###..###.
..........
..........
.###..###.
.###..###.
.###..###.
..........
```

el resultado (parcial) luego de la dilatación es:

```
##########
##########
##########
##########
##########
##########
##########
##########
##########
##########
```

y luego de la erosión tenemos:

```
..........
.########.
.########.
.########.
.########.
.########.
.########.
.########.
.########.
..........
```

Nótese como las 4 iniciales islas de **bits 1s** se unieron en una única isla, de ahí el nombre del filtro.

Procesamiento multihilo

Con el fin de mejorar el rendimiento del sistema cada filtro aplicado usará **N** hilos: la imagen fuente será dividida en **N** grupos de filas y cada hilo procesara un grupo.
Es posible que la cantidad de filas de una imagen no sea múltiplo de **N** en cuyo caso uno de los hilos (a criterio de usted) deberá hacerse cargo del procesamiento de las filas restantes.

En el caso de encadenar múltiples filtros los **N** hilos procesaran la imagen fuente usando el primer filtro para obtener una imagen resultado parcial. Recién cuando la imagen parcial esté completa y *los **N** hilos hayan terminado* se continúa con el siguiente filtro repitiendo los mismos pasos: los **N** hilos procesaran la imagen parcial con el segundo filtro para producir una segunda imagen parcial y continuarán el ciclo hasta obtener eventualmente la imagen final.

Formato de Línea de Comandos

El formato de la línea es
`./tp` **`<numero de hilos> <filtro 1> <patron 1> <filtro 2> <patron 2> …`**

Donde **`<numero de hilos>`** es el número de hilos que se usarán en total *incluyendo al hilo principal*. Un valor de **1** significa que todo el procesamiento lo hará el hilo principal sin lanzar ningun otro hilo adicional.

El parámetro **`<filtro n>`** consta de un carácter y representa el tipo de filtro a ejecutar: los posibles valores son **d** (filtro de dilatación) y **e** (filtro de erosión).
A continuación del parámetro **`<filtro n>`** viene el patrón o estructura **`<patron n>`** que será usado por el n-ésimo filtro. El formato del parámetro **`<patron n>`** es el mismo formato en que se representan las imágenes (véase el formato en la sección Entrada y Salida Estándar).

La línea de comando debe soportar hasta n filtros donde solo el primer filtro (y su patrón) son argumentos obligatorios.

Códigos de Retorno

Codigo de error 1 si los parametros son invalidos, 0 en otro caso.

Entrada y Salida Estándar

El programa leerá de la entrada estándar la imagen binaria a procesar. El formato de esta es:
<ancho> <alto>
<imagen>

Dónde **<ancho>** y **<alto>** son dos números en texto con las dimensiones de la imagen y donde **<imagen>** es la imagen en si representada con dos caracteres ASCII: **.** para el **bit 0**, **#** para el **bit 1**.

Vale aclarar que la cantidad de espacios entre los numeros **<ancho>** y **<alto>** asi como entre estos y la **<imagen>** es *como mínimo 1, pero pueden ser más*, incluso pueden llegar a ser tabs y saltos de línea. Lo mismo aplica a la separación de los caracteres dentro de **<imagen>**. *Se aconseja que investigue* el manipulador **std::ws** y el **operador <<** de la clase **std::istream**.

Por ejemplo los siguientes 3 archivos son imágenes equivalentes a pesar de la diferencia de espacios:

a)
```
3 3
###
###
###
```

b)
```
3 3 ### ### ###
```

c)
```
3       3      #########
```

El programa deberá escribir por salida estándar la imagen final con el mismo formato usado en la entrada estándar solo que la cantidad de espacios y saltos de línea *no* son arbitrarios:

- Un único espacio entre **<ancho>** y **<alto>**
- Seguido de **<alto>** hay un salto de línea **\n** y luego los caracteres de **<imagen>**
- Los caracteres de **<imagen>** vienen todos juntos con un salto de línea **\n** por cada fila
- El final del archivo *no* incluye un salto de línea **\n**.

Ejemplo (los saltos de línea ser resaltaron para una mayor compresión)

```
3 5\n
###\n
###\n
###\n
```

```
###\n
###
```

Ejemplos de Ejecución

Ejemplo 1

Dado el archivo stdin:

```
6 6
......
.#....
......
......
...#..
......
```

Al ejecutar:
`./tp 1 d “3 3 .#. ### .#.”  < stdin`

El resultado por salida estándar es:

```
6 6
.#....
###...
.#....
...#..
..###.
...#..
```

Ejemplo 2

Dado el archivo stdin:

```
10     6
..........       ..###.###.
..########.    ..###.###.
...#......     ......#...
```

Al ejecutar:
`./tp 2 e “3 3 ### ### ###” d “3 3 ######     ###”`

La salida estandar es:

```
10 6
..........
..###.###.
```

```
..###.###.
..###.###.
..........
..........
```

Restricciones

La siguiente es una lista de restricciones técnicas exigidas por el cliente:

1. El sistema debe desarrollarse en ISO C++11.
2. Debe haber *por lo menos una estructura jerárquica* (herencia de clases) y al menos *un método virtual puro*.
3. Las clases que correspondan y que usted considere *deben no ser copiables* y deben implementar *el constructor por movimiento y el operador asignación por movimiento*. Al menos una clase debe cumplir con esos requisitos.

Referencias

[1] Ejemplo de referencia: http://en.wikipedia.org
[2] Filtros Morfologicos
https://www.cs.auckland.ac.nz/courses/compsci773s1c/lectures/ImageProcessing-html/topic4.htm