Morfologia3D

Autor: Daniel Wilczak

Wprowadzenie, operacje morfologiczne.

Trójwymiarowym obrazem binarnym (lub krótko **bitmapą**) będziemy określać taki obraz trójwymiarowy, w którym każdy piksel ma kolor albo czarny, albo biały. W rozpoznawaniu i analizie obrazów binarnych bardzo często stosuje się tak zwane operacje (filtry) morfologiczne. Standardowo filtry te są zdefiniowane bardzo ogólnie. My, na potrzeby niniejszego zadania, zawężimy nieco ich definicje.

Sąsiadem piksela **p** jest każdy inny piksel bitmapy, który styka się z nim pełną ścianą. Piksele stykające się tylko wierzchołkami lub krawędziami nie są sąsiadami. Na przykład, w bitmapie o rozmiarze **7x7x7** indeksowanej od zera, piksel **p=(3,3,3)** ma dokładnie sześciu sąsiadów

(2,3,3), (4,3,3), (3,2,3), (3,4,3), (3,3,2), (3,3,4).

Jeżeli piksel leży na brzegu bitmapy, to sąsiedztwo zawężamy do pikseli, które są w bitmapie.

Poniżej omówimy kolejno filtry morfologiczne, które będą musieli Państwo zaimplementować:

Inwersja
: lub prościej negacja - polega na zamianie pikseli czarnych na białe i białych na czarne.

Erozja
: przez piksel brzegowy obrazu rozumiemy piksel czarny, którego jednym z sąsiadów jest piksel biały. Operacja erozji polega na tym, że najpierw lokalizowane są wszystkie piksele brzegowe w danym obrazie, a następnie ich kolor jest ustawiany na biały.

Dylatacja
: jest w pewnym sensie operacją odwrotną do erozji. W wyniku dylatacji wszystkie białe piksele, które sąsiadują z pikselem czarnym, mają zmieniony kolor na czarny.

Zerowanie
: wszystkie piksele ustawiane są na kolor biały

Uśrednianie
: dla każdego piksela **p** obrazu sprawdzamy liczbę sąsiadujących z nim pikseli białych i czarnych. Jeśli ma on więcej niż trzech sąsiadów w kolorze **k**, to nowym kolorem piksela **p** jest kolor **k**. W przeciwnym razie kolor piksela nie zmienia się. Zwracam Państwa uwagę na to, że sąsiadów rozważamy w oryginalnym obrazie, nie w obrazie częściowo uśrednionym!

Sposób przesłania rozwiązania.

Całość implementacji musi być zawarta w pliku **Morfologia.cpp**. Nie może on zawierać funkcji **main**. Do systemu BaCa przesyłamy pojedynczy plik **Morfologia.cpp**.

Szczegółowe wymagania dotyczące implementacji klas.

Dany jest plik nagłówkowy Morfologia.h, który będzie dostępny w czasie kompilacji i testowania Państwa rozwiązań, a jego zawartość to:

```
#ifndef __MORFOLOGIA_H__
#define __MORFOLOGIA_H__

class Bitmapa
{
public:
  virtual unsigned sx() const = 0;
  virtual unsigned sy() const = 0;
  virtual unsigned sz() const = 0;

  virtual bool& operator()(unsigned x, unsigned y, unsigned z) = 0;
  virtual bool operator()(unsigned x, unsigned y, unsigned z) const
= 0;

  virtual ~Bitmapa(){}
};

class Przeksztalcenie
{
public:
  virtual void przeksztalc( Bitmapa& ) = 0;
  virtual ~Przeksztalcenie() {}
};

#endif
```

Państwa zadaniem jest napisanie kilku klas dziedziczących z klas **Bitmapa** oraz **Przeksztalcenie**.

Plik z rozwiązaniem **Morfologia.cpp** musi włączać plik nagłówkowy **Morfologia.h**

```
#include "Morfologia.h"
```

Należy zaimplementować:

przeładowany operator<<
: jego zadaniem jest wypisanie bitmapy (**Bitmapa**) do podanego strumienia (format wyjścia i przykład użycia będą podane w teście jawnym)

klasę BitmapaExt
: jest to klasa pochodna od klasy **Bitmapa**. Ma ona za zadanie reprezentować bitmapy w wybrany przez Państwa sposób. Ponadto klasa **BitmapaExt**:

- musi definiować wszystkie metody wirtualne z klasy **Bitmapa**
- musi dostarczać konstruktor przyjmujący trzy liczby typu **unsigned** określające rozmiary bitmapy w kolejności: **zakresX, zakresY, zakresZ**. Konstruktor tworzy bitmapę w jednolitym białym kolorze.
- musi definiować metody wirtualne **sx,sy,sz**, które zwracają podane w konstruktorze rozmiary bitmapy, czyli **zakresX, zakresY, zakresZ**,
- musi definiować dwa przeładowane operatory zwracające kolor piksela o podanych współrzędnych **(x,y,z)**. **Piksele są indeksowane od zera!**
- musi poprawnie zarządzać zasobami (konstruktory kopiujące, operatory przypisania, destruktory, jeśli tylko są potrzebne).

klasy pochodne od klasy Przeksztalcenie:

- **Inwersja**,
- **Erozja**,
- **Dylatacja**,
- **Zerowanie**,
- **Usrednianie**.

Każda klasa z powyższej listy musi zaimplementować metodę

void przeksztalc(Bitmapa&);

Metoda **przeksztalc** w klasie **X** musi poprawnie realizować filtr morfologiczny o nazwie **X** - zgodnie z opisem podanym na początku zadania.

klasę ZlozeniePrzeksztalcen

jest to klasa pochodna od **Przeksztalcenie**. Nowo stworzony obiekt klasy **ZlozeniePrzeksztalcen** ma realizować filtr identycznościowy czyli taki, który nie modyfikuje bitmapy podanej jako argument metody **przeksztalc**.
Klasa **ZlozeniePrzeksztalcen** ma dodatkowo definiować metodę

void dodajPrzeksztalcenie(Przeksztalcenie* p);

Wykonanie **przeksztalc** na rzecz obiektu klasy **ZlozeniePrzeksztalcen** polega na:

- przekształcaniu bitmapy wszystkimi filtrami, które zostały dodane do tego obiektu za pomocą metody **dodajPrzeksztalcenie**
- filtry są stosowane w takiej samej kolejności, w jakiej zostały dodane do obiektu klasy **ZlozeniePrzeksztalcen** za pomocą metody **dodajPrzeksztalcenie**