# Blurring Images

Profesor: Veljko Milutinović
Asistent: Jelena Hadži-Purić

Aleksandar Mužina mr14235@alas.matf.bg.ac.rs
Maja Vujović mr14296@alas.matf.bg.ac.rs

Univerzitet u Beogradu, Matematički fakultet

# O algoritmu

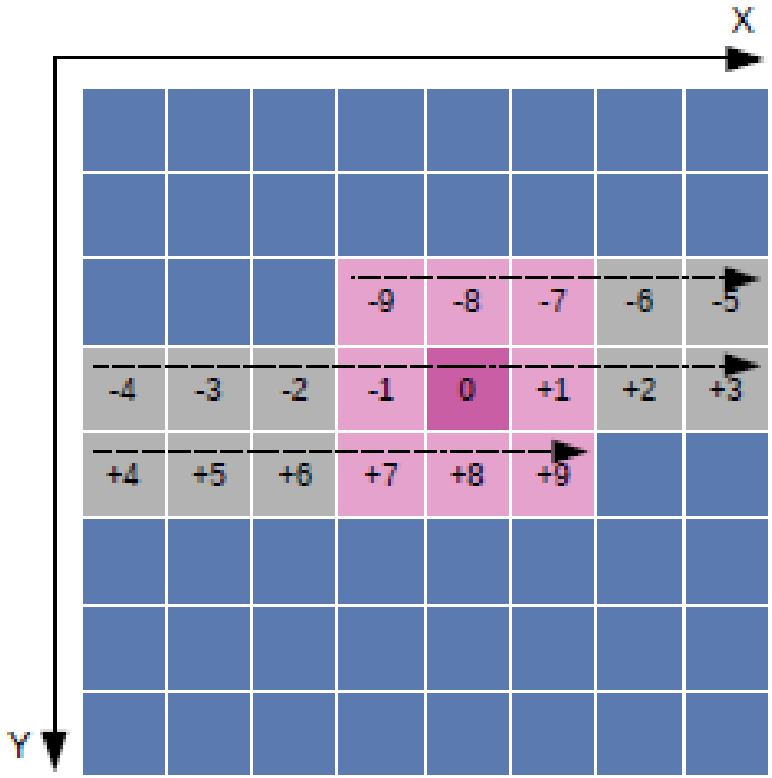
- Zamagljivanje slike (eng. Blurring image) je tehnika koja se koristi u obradi slika.
- Cilj: smanjenje oštrine dela ili cele slike.

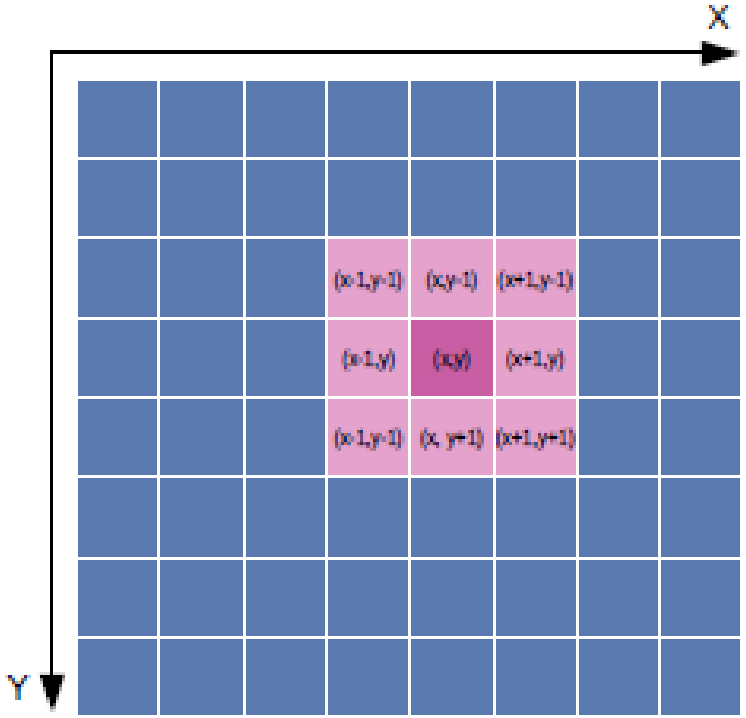
# Upotreba algoritma

- Smanjenje intenziteta ivica objekata
- Efektno razdvajanje pozadine i centra fokusa
- Glatko prelaženje između boja
- Popravljanje slika ″pokvarenih″ piksela
- Sakrivanje identiteta

# Pristup problemu

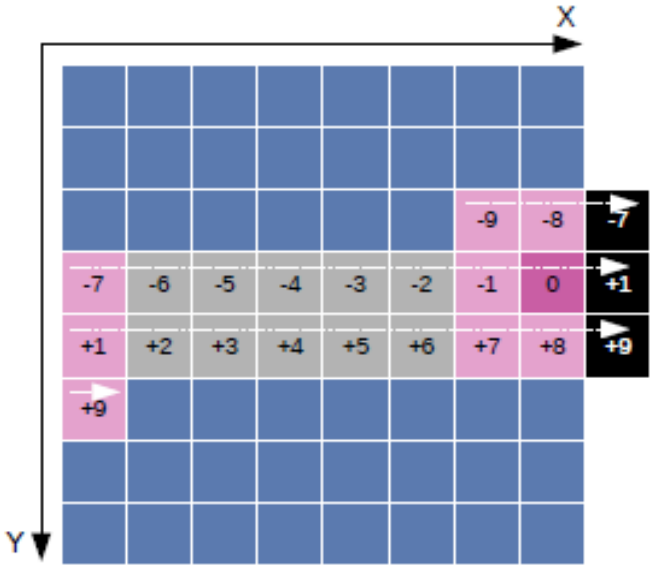
- Slika je predstavljena matricom piksela.
- Uzimanjemo srednju vrednost 9 piksela: trenutne pozicije u matrici i 8 susednih.




Formula za poziciju:
y * širina slike+x

# Detekcija ivice slike




- Važan korak, kako ne bi došlo do uzimanja nevalidnih vredosti piksela.
- Jedno od rešenja: na ivicama slike postavljamo vrednosti piksela na 0.

*(a)* Smoothed Lena without boundary condition *(b)* Smoothed Lena without boundary condition (zoomed) *(c)* Smoothed Lena with boundary condition *(d)* Smoothed Lena with boundary condition (zoomed)

# CPU kod

CPUCode/MeanCpuCode.c

Save | Undo | Redo | Settings

```
#include <math.h>
#include <stdio.h>
#include <stdlib.h>

#include "Maxfiles.h"
#include <MaxSLiCInterface.h>

#include "ppmIO.h"

int main(void){

printf("Loading image.\n");                 //Ucitavanje slike
int32_t *inImage;
int width = 0, height = 0;

loadImage("lena.ppm", &inImage, &width, &height, 1);
int dataSize = width * height * sizeof(int32_t);

int32_t *outImage = malloc(dataSize);       // Alciranje prostora za rezultujucu sliku

printf("Running Kernel.\n");

Mean(width * height, inImage, outImage);
printf("Saving image.\n");

writeImage("lena_mean.ppm", outImage, width, height, 1);    //Ispis rezultujuce slike
printf("Exiting\n");
return 0;
}
```

# Kernel kod (1/2)

EngineCode/src/mean/MeanKernel.maxj

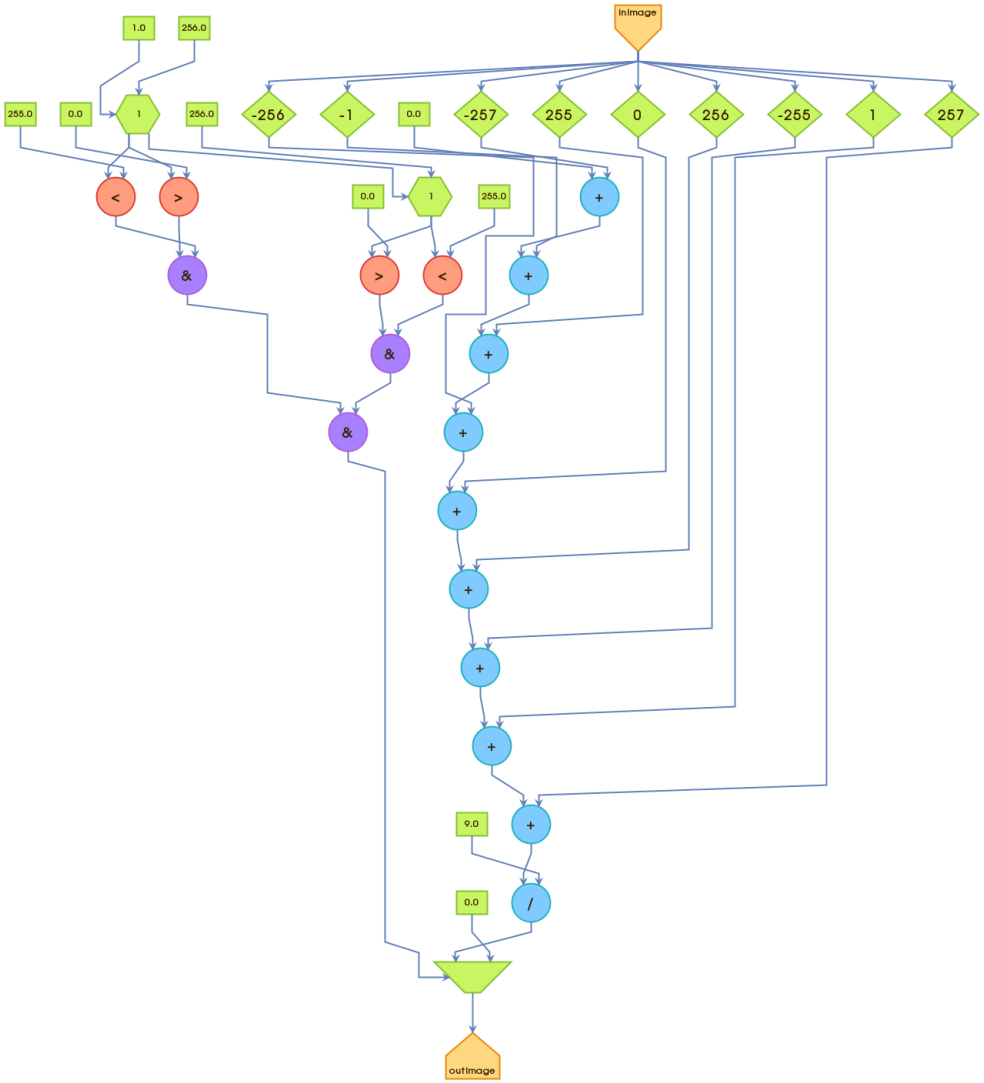
Save | Undo | Redo | Settings

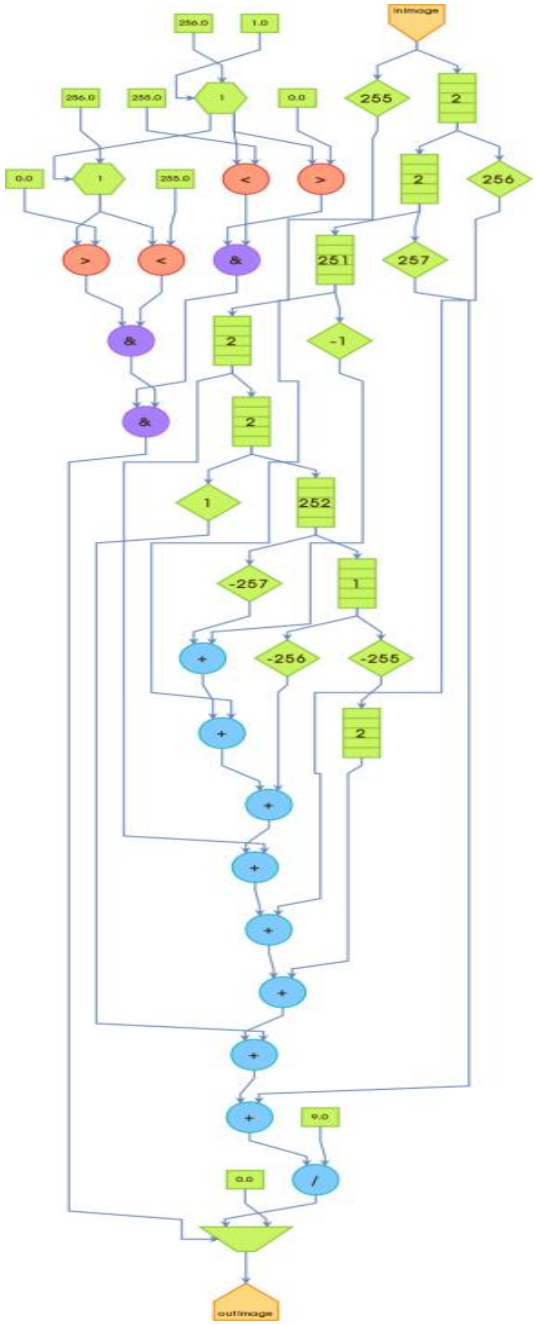
```
1  package mean;
2
3  import com.maxeler.maxcompiler.v2.kernelcompiler.Kernel;
4  import com.maxeler.maxcompiler.v2.kernelcompiler.KernelParameters;
5  import com.maxeler.maxcompiler.v2.kernelcompiler.stdlib.core.CounterChain;
6  import com.maxeler.maxcompiler.v2.kernelcompiler.types.base.DFEVar;
7
8  class MeanKernel extends Kernel {
9      protected MeanKernel(KernelParameters parameters) {
10         super(parameters);
11
12     int height = 256, width = 256;
13
14     DFEVar inImage = io.input("inImage", dfeInt(32));
15     DFEVar Sum = constant.var(0);       //U promenljivoj Sum cuvamo sume, inicijalizujemo je na 0
16
17 // Sabiramo vrednosti 9 susednih polja u pamtimo u Sum
18     for (int x = -1; x <= 1; x++)
19         for (int y = -1; y <= 1; y++)
20             Sum = Sum + stream.offset(inImage, y * width + x);
21
22 // Racunamo srednju vrednost tako sto podelimo sa brojem sabranih vrednosti
23     DFEVar result = Sum / 9;        //Cuvamo srednju vrednost u rezultujucu promenljivu
```

# Kernel kod (2/2)

```
24
25  // Detekcija ivice slike:
26  // Ukoliko je  pitanju prva vrsta ili prva kolona
27  // ili poslednja vrsta ili poslednja kolona, postavljamo 0
28      CounterChain chain = control.count.makeCounterChainMoreBits();
29      DFEVar y = chain.addCounter(height, 1);
30      DFEVar x = chain.addCounter(width, 1);
31      result = ((x > 0 & x < width - 1) & (y > 0 & y < height - 1)) ? result : 0;
32
33      io.output("outImage", result, result.getType());
34
35          }
36
37      }
```

# Grafovi


inImage
1.0
256.0
255.0
0.0
1
-256
-1
-257
255
0
256
-255
257
9.0
outImage


inImage
256.0
1.0
255
2
251
257
-1
252
-257
-256
-255
9.0
outImage

# Hvala na pažnji!