

# Computer Vision

Catalin Stoean  
catalin.stoean@inf.ucv.ro  
<http://inf.ucv.ro/~cstoean>

# Obiective

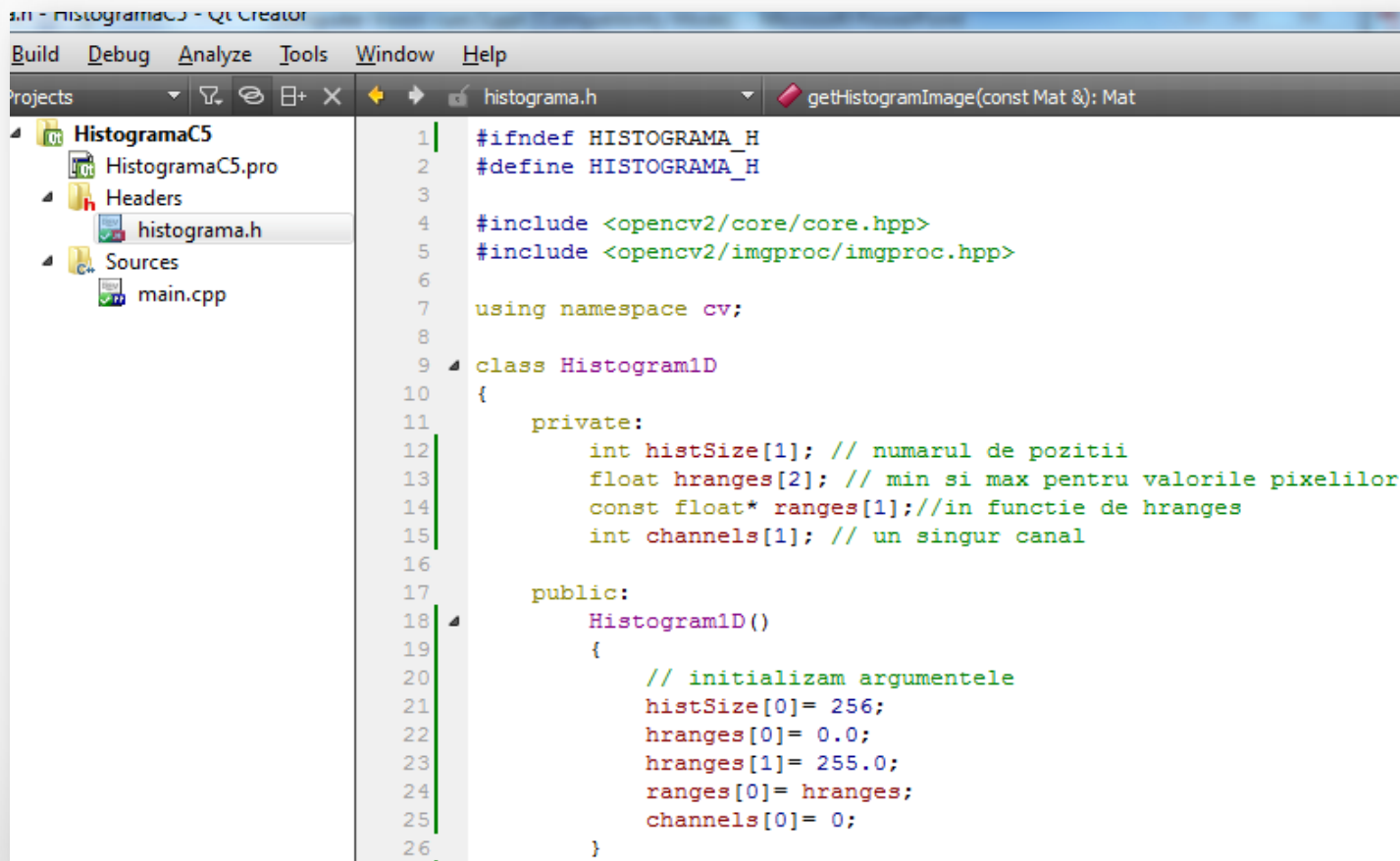
- Calcul histograme pentru imagini
- Modificarea imaginilor cu tablouri look-up
- Egalizarea histogramei
- Proiectarea inapoi a histogramei pentru a detecta un anumit continut din imagine
- Folosirea algoritmului mean-shift pentru a detecta obiecte
- Similaritatea dintre imagini folosind histogramele lor
-

# Ce este histograma?

- Histograma
  - caracterizeaza continutul unei imagini
  - Ajuta la detectarea obiectelor dintr-o imagine
  - Se pot identifica texturi cu ajutorul lor
- Intr-o imagine alb-negru (un singur canal) fiecare pixel are o valoare de la 0 (negru) la 255 (alb)
- O histograma este un tabel care da numarul de pixeli care au aceeasi valoare intr-o imagine
- Histograma pentru o imagine alb-negru va avea 256 de valori:
  - Pozitia 0 contine numarul de pixeli care au valoarea 0
  - Pozitia 1 contine numarul de pixeli care au valoarea 1
  - ...

# Calcul histograme pentru imagini

- Facem o clasa pentru histograma unei poze alb-negru:



```
1 | #ifndef HISTOGRAMA_H
2 | #define HISTOGRAMA_H
3 |
4 | #include <opencv2/core/core.hpp>
5 | #include <opencv2/imgproc/imgproc.hpp>
6 |
7 | using namespace cv;
8 |
9 | class Histogram1D
10 | {
11 |     private:
12 |         int histSize[1]; // numarul de pozitii
13 |         float hranges[2]; // min si max pentru valorile pixelilor
14 |         const float* ranges[1]; // in functie de hranges
15 |         int channels[1]; // un singur canal
16 |
17 |     public:
18 |         Histogram1D()
19 |         {
20 |             // initializam argumentele
21 |             histSize[0]= 256;
22 |             hranges[0]= 0.0;
23 |             hranges[1]= 255.0;
24 |             ranges[0]= hranges;
25 |             channels[0]= 0;
26 |         }
```

# Calcul histograme pentru imagini

- MatND este o clasa care manipuleaza matrice N-dimensionale si este folosita pentru a reprezenta histograme.
- Exista mai multe definitii pentru calculul histogramei care sa aiba si alti parametri
  - Pot fi consultate in documentatia OpenCV

```
// Calculeaza histograma pentru o imagine alb-negru
MatND getHistogram(const Mat &image)
{
    MatND hist;
    // Calculam histograma
    calcHist(&image,
    1, // histograma unei singure imagini
    channels, // canalul folosit
    Mat(), // nu se foloseste mask
    hist, // histograma rezultata
    1, // histograma 1D
    histSize, // numarul de pozitii
    ranges // intervalele
    );
    return hist;
}
```

# main.cpp

- Citim o imagine in format alb-negru (punem un al doilea argument 0 la `imread`).
- Calculam histograma si retinem rezultatul in `histo`.
- Afisam valorile histogramei.

```
main.cpp main(): int
1 #include <iostream>
2 #include <opencv2/core/core.hpp>
3 #include <opencv2/highgui/highgui.hpp>
4 #include <opencv2/imgproc/imgproc.hpp>
5 #include <opencv2/video/tracking.hpp>
6
7 #include "histograma.h"
8
9 using namespace std;
10 using namespace cv;
```

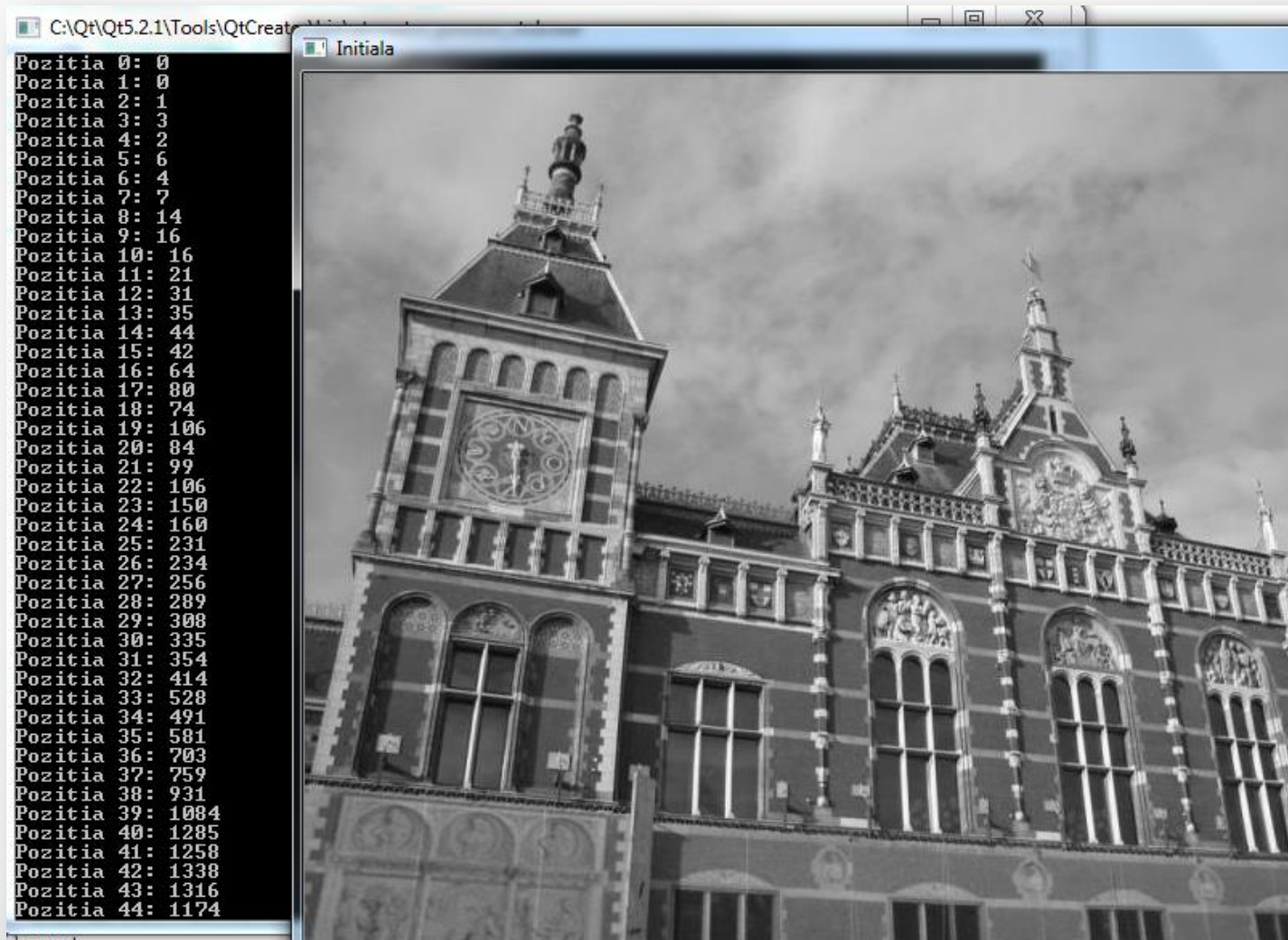
```
int main()
{
    Mat poza = imread("D:/pic.jpg",
0); // deschidem poza in alb-negru

    namedWindow("Initiala");
    imshow("Initiala", poza);

    // Obiectul histograma
    Histogram1D h;
    // Calculam histograma
    MatND histo = h.getHistogram(poza);

    // Se pot afisa valorile histogramei
    for (int i = 0; i < 256; i++)
        cout << "Pozitia " << i << ": " << histo.at<float>(i) << endl;
```

# Valori histograma



# Histograma desenata

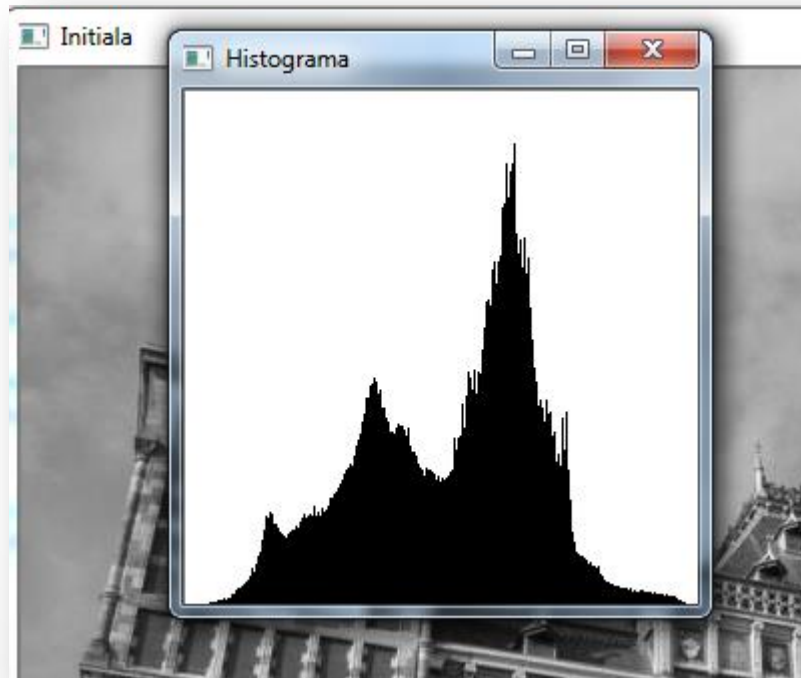
- Adaugam metoda getHistogramImage la clasa Histogram1D.

```
//Intoarce imaginea unei histograme
Mat getHistogramImage(const Mat &image)
{
    MatND hist= getHistogram(image);
    // Aflam valorile min si max
    double maxVal=0;
    double minVal=0;
    minMaxLoc(hist, &minVal, &maxVal, 0, 0);
    // Imaginea in care sa afisam histograma
    //alba, de 256 x 256 pixeli
    Mat histImg(histSize[0], histSize[0], CV_8U,Scalar(255));
    // Punem cel mai inalt punct la 90% din 256
    int hpt = static_cast<int>(0.9*histSize[0]);
    // Desenam o linie verticala pentru fiecare pozitie
    for( int h = 0; h < histSize[0]; h++ )
    {
        float binVal = hist.at<float>(h);
        int intensity = static_cast<int>(binVal*hpt/maxVal);
        // functia urmatoare deseneaza o linie intre 2 puncte
        line(histImg,Point(h,histSize[0]), Point(h,histSize[0]-intensity), Scalar::all(0));
    }
    return histImg;
}
```



# main.cpp

```
// Afisam histograma ca imagine  
namedWindow("Histograma");  
imshow("Histograma", h.getHistogramImage(poza));
```

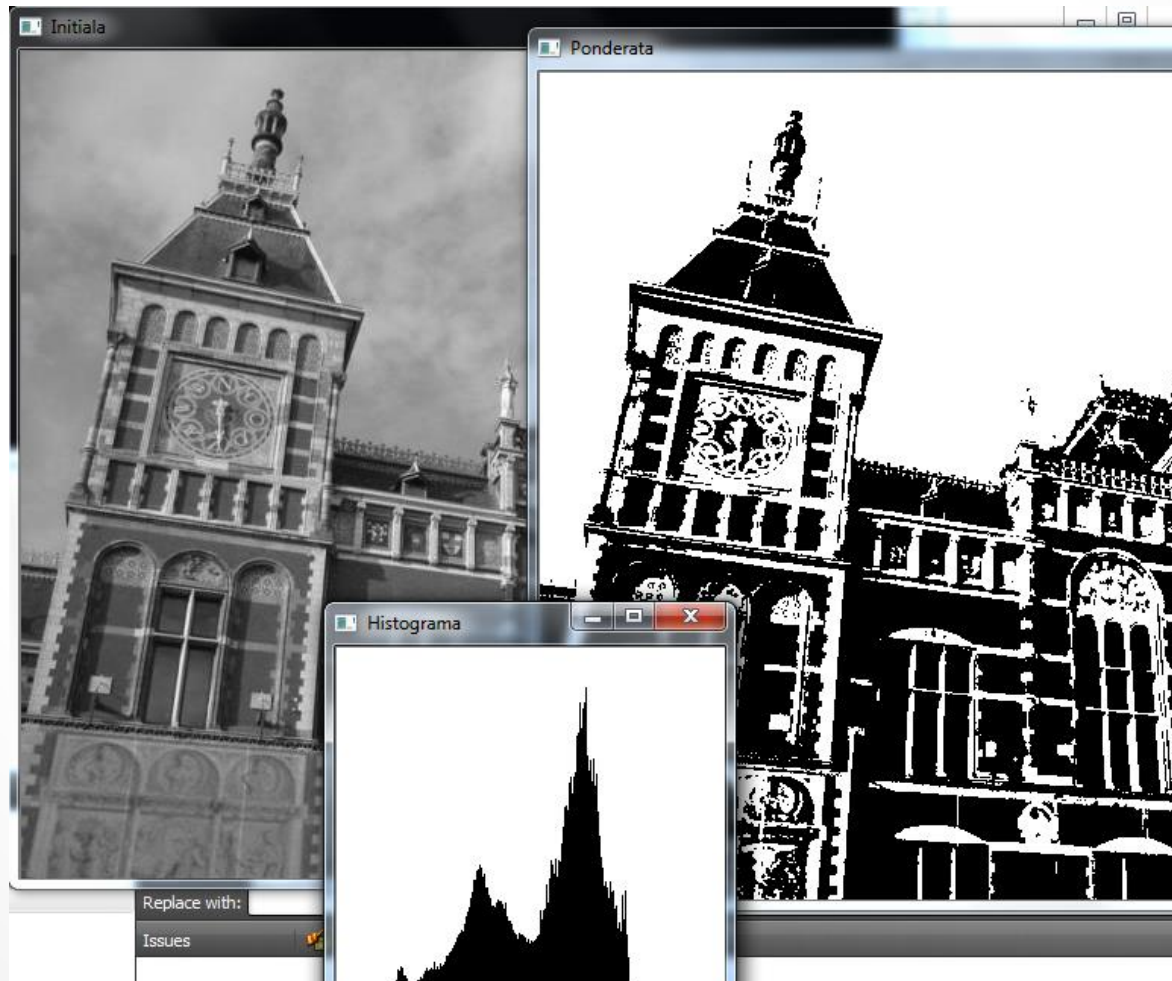


# Thresholding

- Este cea mai simpla metoda de segmentare
- Din histograma putem observa intuitiv in ce zona se gasesc cei mai multi pixeli.
- Putem selecta contururile celor mai importante sectiuni din imaginea initiala (segmentare) prin o buna stabilire a acestor praguri.

```
Mat thresholded;  
threshold(poza, thresholded, 120, 255, THRESH_BINARY);  
  
namedWindow("Ponderata");  
imshow("Ponderata", thresholded);
```

# Thresholding



# Thresholding - optiuni

- Exista mai multe optiuni de a aplica ponderarea:
- `THRESH_BINARY` (cea utilizata anterior)

$$\text{dst}(x, y) = \begin{cases} \text{maxval} & \text{if } \text{src}(x, y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- `dst` vine de la destinatie, `src` de la sursa, `thresh` este pragul, `maxval` este al patrulea parametru din functia `threshold`.
- Daca intensitatea unui pixel este mai mare decat pragul (`thresh`), noua intensitate devine `MaxVal`.
  - In cazul anterior, totul a fost facut negru
- Altfel, 0 (alb)
-

# Thresholding - optiuni

- Exista mai multe optiuni de a aplica ponderarea:
- THRESH\_BINARY\_INV

$$\text{dst}(x, y) = \begin{cases} 0 & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{maxval} & \text{otherwise} \end{cases}$$

- Daca intensitatea unui pixel este mai mare decat pragul (thresh), noua intensitate devine 0.
  - Altfel, maxval
- THRESH\_TRUNC

$$\text{dst}(x, y) = \begin{cases} \text{threshold} & \text{if } \text{src}(x, y) > \text{thresh} \\ \text{src}(x, y) & \text{otherwise} \end{cases}$$

- Intensitatea maxima devine pragul daca sursa este peste prag, altfel se mentine la fel

# Thresholding - optiuni

- THRESH\_TOZERO

$$\text{dst}(x,y) = \begin{cases} \text{src}(x,y) & \text{if } \text{src}(x,y) > \text{thresh} \\ 0 & \text{otherwise} \end{cases}$$

- Daca pixelul sursa are intensitatea peste prag, atunci ramane la fel, altfel devine 0.

- THRESH\_TOZERO\_INV

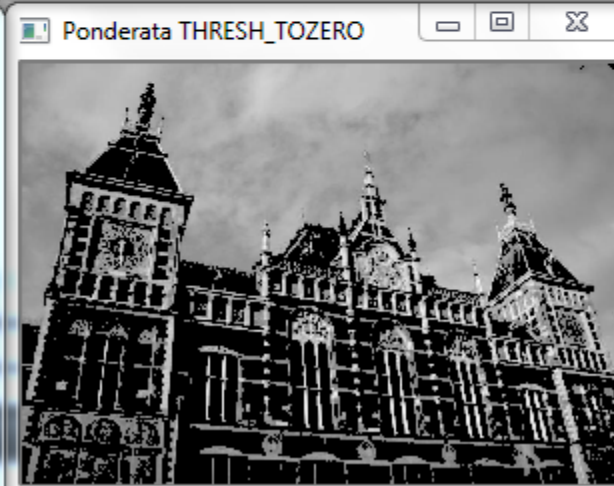
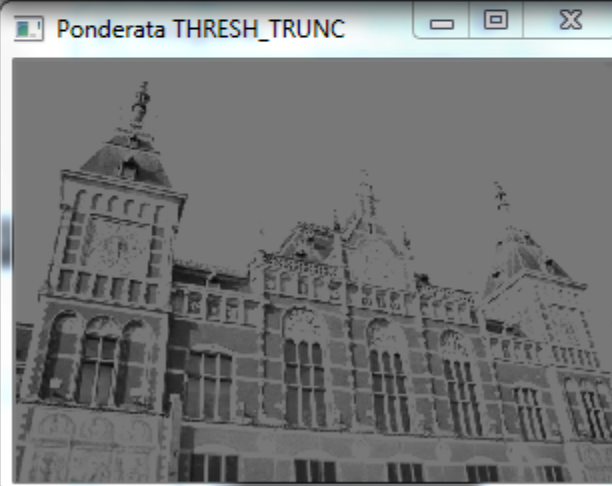
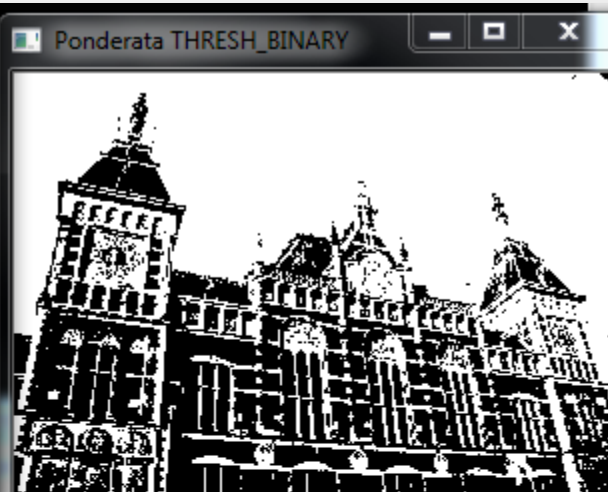
$$\text{dst}(x,y) = \begin{cases} 0 & \text{if } \text{src}(x,y) > \text{thresh} \\ \text{src}(x,y) & \text{otherwise} \end{cases}$$

- Daca pixelul sursa are intensitatea peste prag, atunci devine 0, altfel ramane la fel.

# Thresholding - optiuni

```
void multipleThreshold(Mat &poza, int type)
{
    string rez;
    rez = "Ponderata ";
    Mat thresholded;
    switch(type)
    {
        case 0://THRESH_BINARY
            rez += "THRESH_BINARY";
            threshold(poza, thresholded, 120, 255, THRESH_BINARY);
            break;
        case 1://THRESH_BINARY_INV
            rez += "THRESH_BINARY_INV";
            threshold(poza, thresholded, 120, 255, THRESH_BINARY_INV);
            break;
        case 2://THRESH_TRUNC
            rez += "THRESH_TRUNC";
            threshold(poza, thresholded, 120, 255, THRESH_TRUNC);
            break;
        case 3://THRESH_TOZERO
            rez += "THRESH_TOZERO";
            threshold(poza, thresholded, 120, 255, THRESH_TOZERO);
            break;
        case 4://THRESH_TOZERO_INV
            rez += "THRESH_TOZERO_INV";
            threshold(poza, thresholded, 120, 255, THRESH_TOZERO_INV);
            break;
        default://
            cout<<"Sunt numai 5 posibilitati: 0-4";
            break;
    }
    namedWindow(rez, WINDOW_NORMAL);//cu WINDOW_NORMAL se permite redimensionarea ferestrei
    resizeWindow(rez, 300, 210);
    imshow(rez, thresholded);
}
```

# Thresholding





# Histograma color

- Avem nevoie de 3 canale.
- Definim marimile pentru ele.

```
class ColorHistogram {
private:
    int histSize[3]; //cate o histograma pentru fiecare canal
    float hranges[2]; // min si max pentru valorile pixelilor
    const float* ranges[3];
    int channels[3]; //3 canale
public:
    ColorHistogram()
    {
        // Initializam argumentele pentru histograma color
        histSize[0]= histSize[1]= histSize[2]= 256;
        hranges[0]= 0.0;
        hranges[1]= 255.0;
        ranges[0]= hranges; // toate canalele au aceleasi intervale
        ranges[1]= hranges;
        ranges[2]= hranges;
        channels[0]= 0; // cele trei canale
        channels[1]= 1;
        channels[2]= 2;
    }
}
```

# Histograma color

- Diferenta principala fata de metoda folosita la poza alb-negru este ca avem acel parametru cu valoarea 3 in loc de 1.
- Histograma rezultata va fi tridimensionala.
- Rezultatul ne ajuta pentru a face comparatii intre histograme, detectare de obiecte, etc.

```
MatND getHistogram(const Mat &image)
{
    MatND hist;
    calcHist(&image,
            1, // histograma unei singure imagini
            channels, // canalul folosit
            Mat(), // nu se foloseste mask
            hist, // histograma rezultata
            3, // histograma 3D
            histSize, // numarul de pozitii
            ranges // intervalele pentru pixeli
            );
    return hist;
}
```

# Histograma color

- Matricea MatND contine  $3 \cdot 256$  elemente.
- O modalitate mai economica de a reprezenta histogramele este folosind structura SparseMat
  - Aceasta nu foloseste multa memorie
  - Omite valorile nule

```
SparseMat getSparseHistogram(const Mat &image)
{
    SparseMat hist(3, histSize, CV_32F);

    calcHist(&image, 1, channels, Mat(), hist, 3, histSize, ranges);
    return hist;
}
```

# Modificarea imaginilor cu tablouri look-up

- Analizand distributia pixelilor dintr-o histograma, putem modifica si chiar imbunatati o imagine.
- Un tablou look-up este o **functie** care defineste o modalitate de transformare a valorilor pixelilor in noi valori.
- Se foloseste o metoda LUT care are urmatoarele argumente:
  - O imagine de intrare (Mat)
  - Un tablou look-up (tot Mat)
  - Imaginea rezultat (Mat)

# Modificarea imaginilor cu tablouri look-up

- Rezultatul este o imagine noua unde valorile pentru intensitatile noi sunt schimbate conform tabloului look-up.
- Definim (in dreapta jos) un tablou look-up care sa inverseze intensitatile pixelilor: 0 devine 255, 1->254 etc.

```
Mat applyLookUp(const Mat& image, const Mat& lookup)
{
    // imaginea de iesire
    Mat rez;

    LUT(image, lookup, rez);
    return rez;
}
```

```
// Cream un tablou look-up pentru inversarea imaginii

Mat lut(1, // 1 dimensiune
256, // 256 de valori
CV_8U); // tip uchar

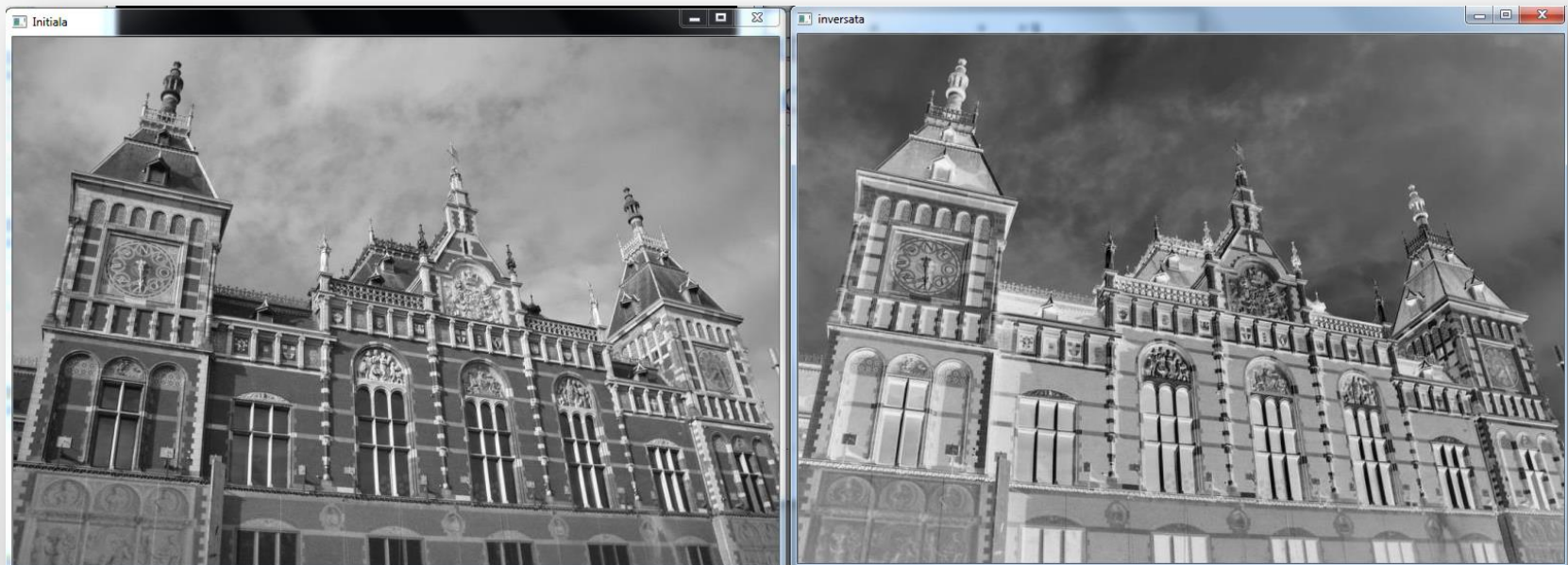
for (int i = 0; i < 256; i++)
    lut.at<uchar>(i) = 255 - i;

Mat pozaNoua = h.applyLookUp(poza, lut);

namedWindow("inversata");
imshow("inversata", pozaNoua);
```

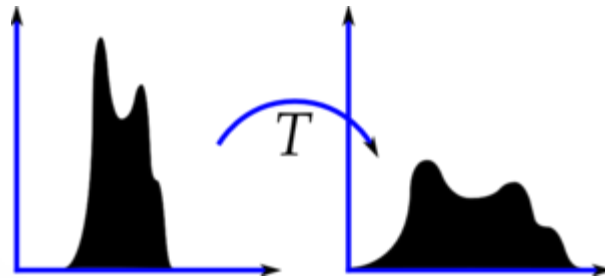
# Modificarea imaginilor cu tablouri look-up

- Am adaugat metoda `applyLookUp` in clasa `Histogram1D` si cealalta bucata de cod in `main.cpp`

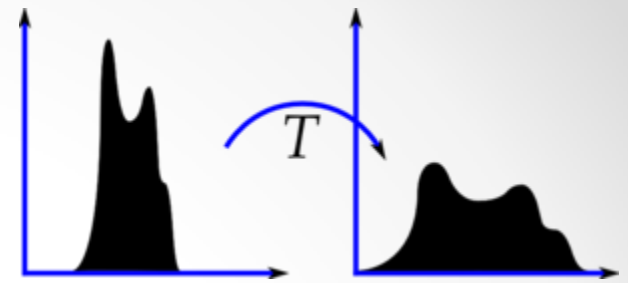


# Imbunatatirea contrastului

- Se poate *stretch* o imagine pentru a atinge un contrast mai bun.
- Pixelii din imaginile foarte luminoase au intensitati foarte ridicate.
- Dar o imagine reusita are intensitati similare pe toata plaja de posibilitati.



# Imbunatatirea contrastului



- Se detecteaza cea mai din stanga (imin) si cea mai din dreapta valoare cu intensitatea diferita de 0 in histograma.
- Valorile sunt remapate astfel incat pozitiile de pana la imin sunt facute 0, iar cele mai mare decat imax 255.
- Intensitatile  $I$  dintre imin si imax sunt remapate liniar dupa formula:
  - $i = 255 * (i - \text{min}) / ((\text{imax} - \text{imin}) + 0.5)$



# Imbunatatirea contrastului

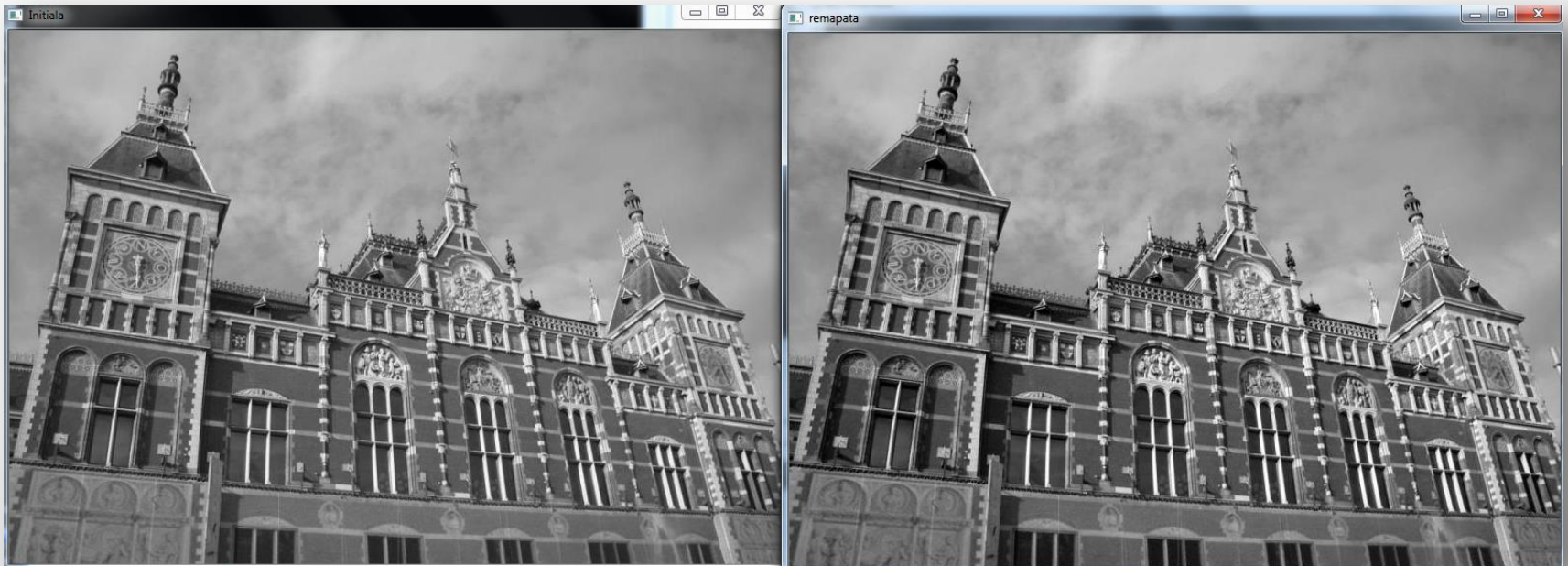
```
Mat strange(const Mat &image, int minValue=0)
{
    // Calculam histograma
    MatND hist = getHistogram(image);
    // gasim extremitatea din stanga a histogramei
    int imin = 0;
    while(imin < histSize[0] && hist.at<float>(imin) < minValue)
        imin++;

    // gasim extremitatea din dreapta a histogramei
    int imax = histSize[0]-1;
    while(imax >= 0 && hist.at<float>(imax) <= minValue)
        imax--;

    // Cream tabloul lookup
    Mat lookup(1, 256, CV_8U);
    for (int i = 0; i < 256; i++)
    {
        if (i < imin) lookup.at<uchar>(i) = 0;
        else
            if (i > imax) lookup.at<uchar>(i) = 255;
            else
                lookup.at<uchar>(i) = static_cast<uchar>(255.0*(i-imin)/(imax-imin)+0.5);
    }
    // Aplicam tabloul lookup la imaginea initiala
    Mat rez;
    rez = applyLookup(image,lookup);
    return rez;
}
```

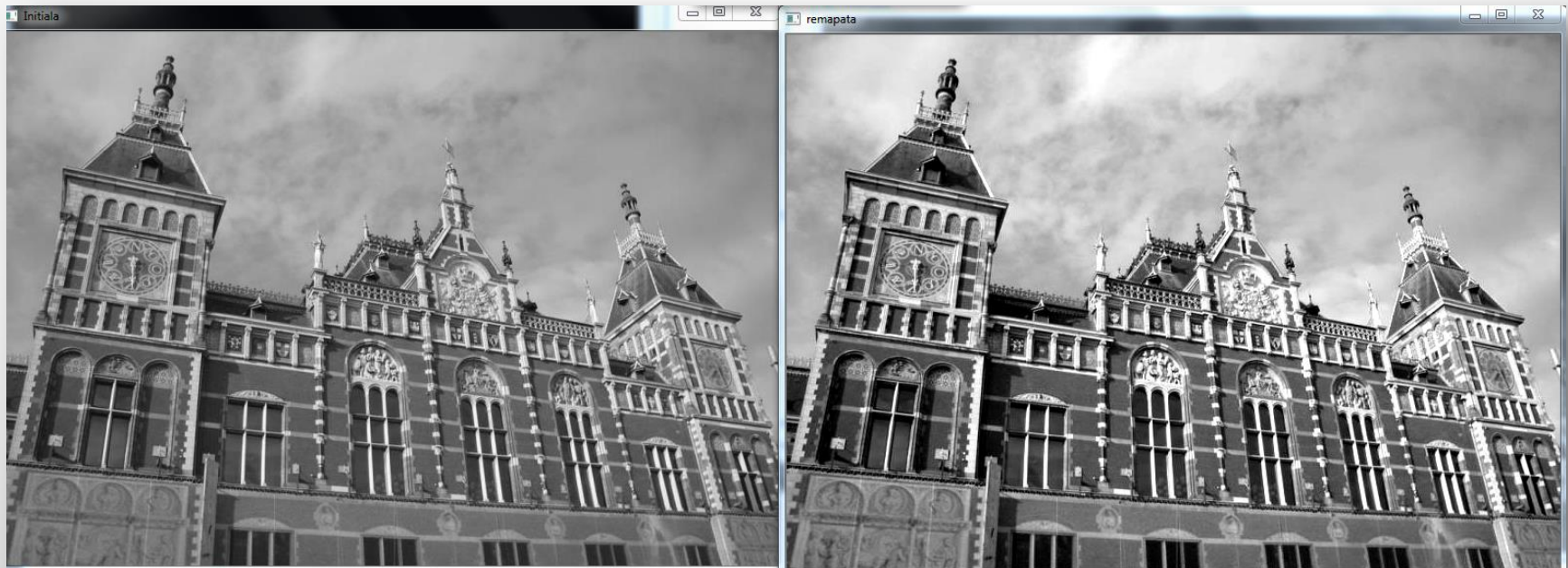
# Imbunatatirea contrastului

```
// ignora pozitiile de inceput si sfarsit cu mai putin de 100 pixeli - imbunatateste contrastul.  
Mat stransa= h.strange(poza,100);  
  
namedWindow("remapata");  
imshow("remapata", stransa);
```



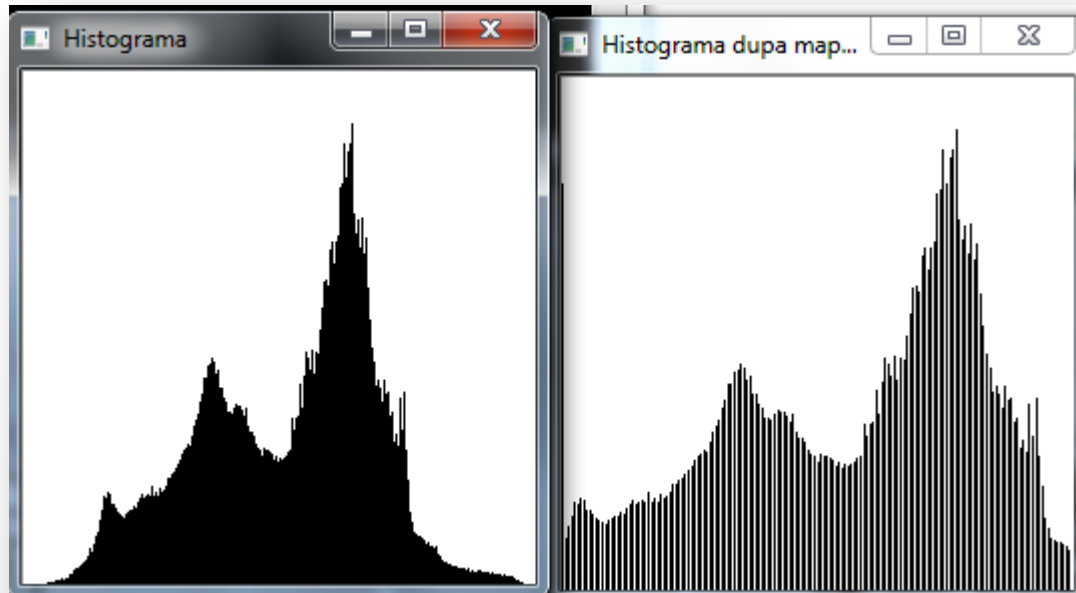
# Imbunatatirea contrastului

- In exemplul de mai jos pragul a fost ridicat la 600.
- Se poate observa si mai clar cum partile deschise in poza initiala devin si mai deschise in dreapta, iar cele inchise -> si mai inchise.



# Imbunatatirea contrastului

```
//si asa se transforma histograma  
namedWindow("Histograma dupa mapare");  
imshow("Histograma dupa mapare", h.getHistogramImage(stransa));
```



# Egalizarea histogramei

- Pentru a realiza o buna acoperire a intensitatilor tuturor pixelilor posibili, avem o solutie simpla oferita de OpenCV:
  - `equalizeHist(imagineSursa, imagineDestinatie)`

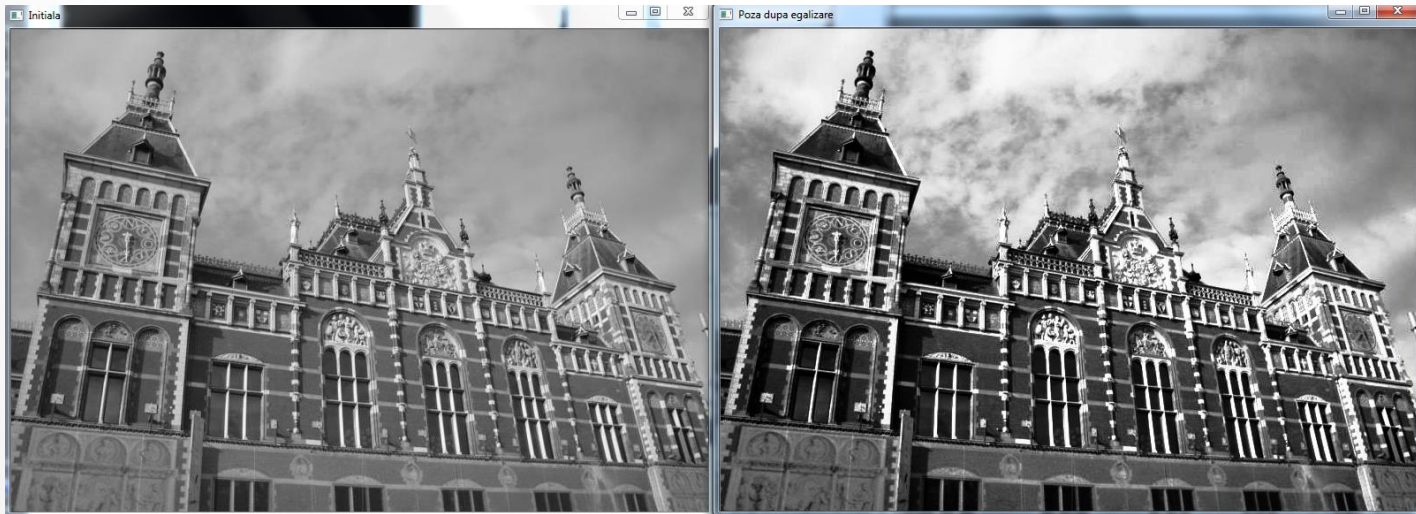
```
//egalizarea histogramei
Mat equalize(const Mat &image)
{
    Mat rez;
    equalizeHist(image, rez);
    return rez;
}
```

```
Mat pozaEgalizare = h.equalize(poza);

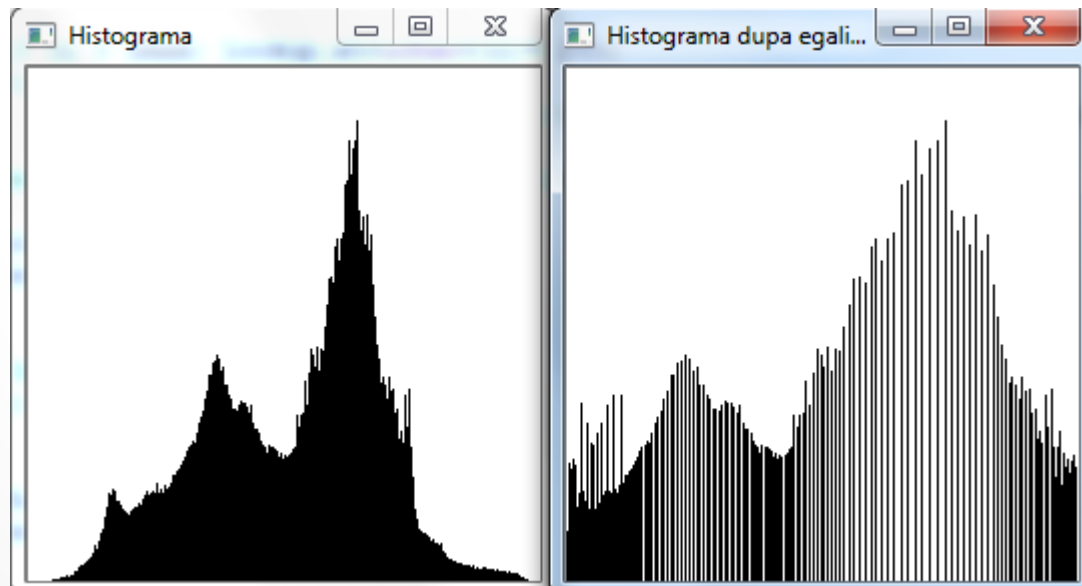
namedWindow("Poza dupa egalizare");
imshow("Poza dupa egalizare", pozaEgalizare);

//si asa se transforma histograma
namedWindow("Histograma dupa egalizare");
imshow("Histograma dupa egalizare",
h.getHistogramImage(pozaEgalizare));
```

# Egalizarea histogramei



# Egalizarea histogramei



# Proiectarea inapoi a histogramei pentru a detecta un anumit continut din imagine

- Histograma unei sectiuni dintr-o imagine (un obiect, o textura) poate fi utilizata pentru ca apoi sa obtinem probabilitatea ca un pixel dat sa apartina acelui/acelei obiect/texturi.
- Selectam o regiune de interes (ROI) dintr-o imagine pentru ca apoi sa cautam in imaginea completa unde se mai pot identifica regiuni asemanatoare.





# Cum desenam un dreptunghi pe o poza

- Desenarea unui dreptunghi pe o poza ne ajuta sa identificam ROI.
- Avand `poza` citita anterior in program:

```
rectangle(poza, Rect(355, 360, 50, 110),  
          Scalar(0, 0, 255), //culoarea  
          2, //grosimea liniilor  
CV_AA); //tipul liniei = anti-aliasing  
  
namedWindow("Initiala");  
imshow("Initiala", poza);
```

# Proiectarea inapoi a histogramei pentru a detecta un anumit continut din imagine

- Folosim clasa Histogram1D definita anterior pentru a obtine histograma imaginii date.
- Realizam normalizarea histogramei pentru a obtine doar valori in  $[0, 1]$ , adica probabilitati ca un pixel cu o anumita intensitate sa apartina imaginii.
- Proiectarea inapoi a histogramei consta in inlocuirea valorilor fiecarui pixel dintr-o imagine de intrare cu valoarea probabilitatii corespondenta din histograma normalizata.
  - Fireste, se poate lua histograma unei imagini diferite decat cea in care se face apoi verificarea.

# Proiectarea inapoi a histogramei pentru a detecta un anumit continut din imagine

- Realizam o noua clasa pentru gasirea de obiecte similare intr-o imagine folosind proiectarea inapoi pornind de la histograma unuei ROI.

```
class ObjectFinder
{
    private:
        float hranges[2];
        const float* ranges[3];
        int channels[3];
        float threshold;
        MatND histogram;
        SparseMat shistogram;
        bool isSparse;

    public:
        ObjectFinder()
        {
            threshold = 0.1f;
            isSparse = false;
            ranges[0] = hranges; // toate canalele au aceleasi margini
            ranges[1] = hranges;
            ranges[2] = hranges;
        }
}
```

# Proiectarea inapoi a histogramei pentru a detecta un anumit continut din imagine

- Adaugam cateva metode publice pentru stabilire de valori pentru parametri.

```
void setThreshold(float t)
{
    threshold = t;
}

float getThreshold()
{
    return threshold;
}

//realizam si normalizarea histogramei daca aceasta este de tip MatND
void setHistogram(const MatND& h)
{
    isSparse= false;
    histogram= h;
    normalize (histogram, histogram, 1.0);
}

//realizam si normalizarea histogramei daca aceasta este de tip SparseMat
void setHistogram(const SparseMat& h)
{
    isSparse= true;
    shistogram= h;
    normalize (shistogram, shistogram, 1.0, NORM_L2);
}
```

# Proiectarea inapoi a histogramei pentru a detecta un anumit continut din imagine

```
// cautam pixelii care apartin histogramei
Mat find(const Mat& image)
{
    Mat result;
    hranges[0]= 0.0;
    hranges[1]= 255.0;
    channels[0]= 0;
    channels[1]= 1;
    channels[2]= 2;

    if (isSparse)// apelam functia potrivita in functie de tipul histogramei
        calcBackProject(&image,
            1,          // o imagine
            channels,   // vector care specifica ce dimensiuni ale histogramei apartin la ce canale de imagine
            shistogram, // histogram utilizata de tip SparseMat
            result,    // imaginea proiectata inapoi rezultata
            ranges,    // intervalele pentru fiecare dimensiune
            255);      // factorul de scalare care trimite valorile
                    // din [0, 1] catre {0, ..., 255} prin inmultirea cu 255

    else//histograma de tip MatND
        calcBackProject(&image, 1, channels, histogram, result, ranges, 255);

    // Aplicam threshold pentru a obtine o imagine binara si a ameliora aspectul imaginii rezultat
    if (threshold > 0.0)//trebuie sa punem cv:: ca sa stie de unde sa ia metoda
        cv::threshold(result, result, 255*threshold, 255, THRESH_BINARY);
    return result;
}
```

# Proiectarea inapoi a histogramei pentru a detecta un anumit continut din imagine

```
//regiuni de interes
Mat pozaROI, pozaROIColor;
pozaROI= poza(Rect(355,360,50,100)); // o fereastră din poza

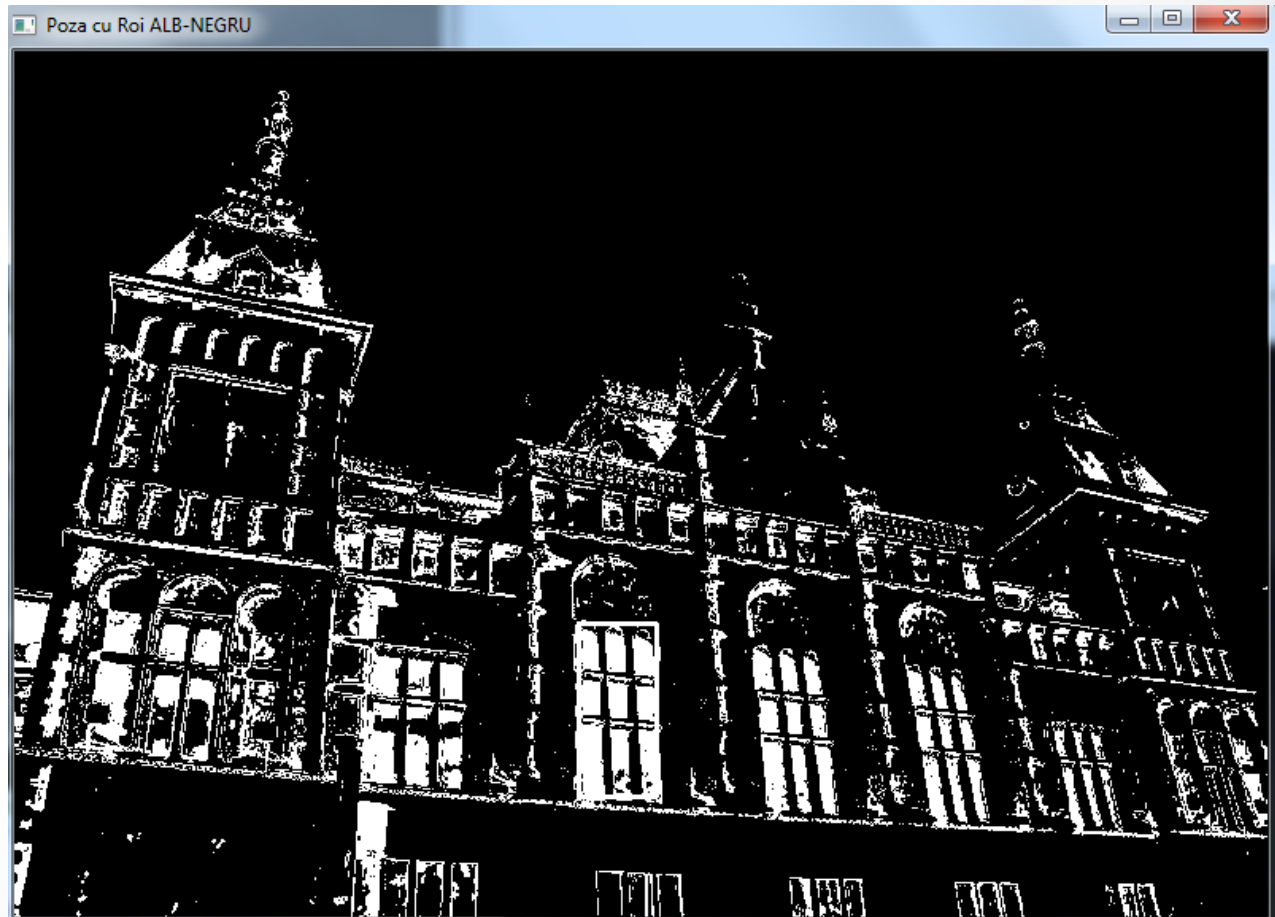
rectangle(poza, Rect(355,360,50,110), Scalar(0,0,255), 2, CV_AA);
namedWindow("Initiala");
imshow("Initiala", poza);
```

- Am selectat ca ROI fereastra din imagine.



# Proiectarea inapoi a histogramei pentru a detecta un anumit continut din imagine

- Se observa ca a identificat cu o buna precizie ferestrele, in special pe cea folosita ca ROI



# Proiectarea inapoi a histogramei pentru a detecta un anumit continut din imagine

- Insa a gasit si alte sectiuni din imagine ca asemanatoare.
- Motivul acestor erori mari este dat de faptul ca s-a lucrat pe o imagine alb-negru.
  - Sunt multe nuante din alte culori care la transformare au ajuns sa semene cu gri-ul de la fereastra cladirii
- Vom face in continuare proiectarea inapoi pornind de la poza colorata.

```
//poza color
ColorHistogram hc;
Mat pozaColor = imread("D:/pic.jpg");
//pozaColor = hc.reducereCulori(pozaColor, 30);
pozaROIColor = pozaColor(Rect(355,360,50,100)); // fereastra cladirii

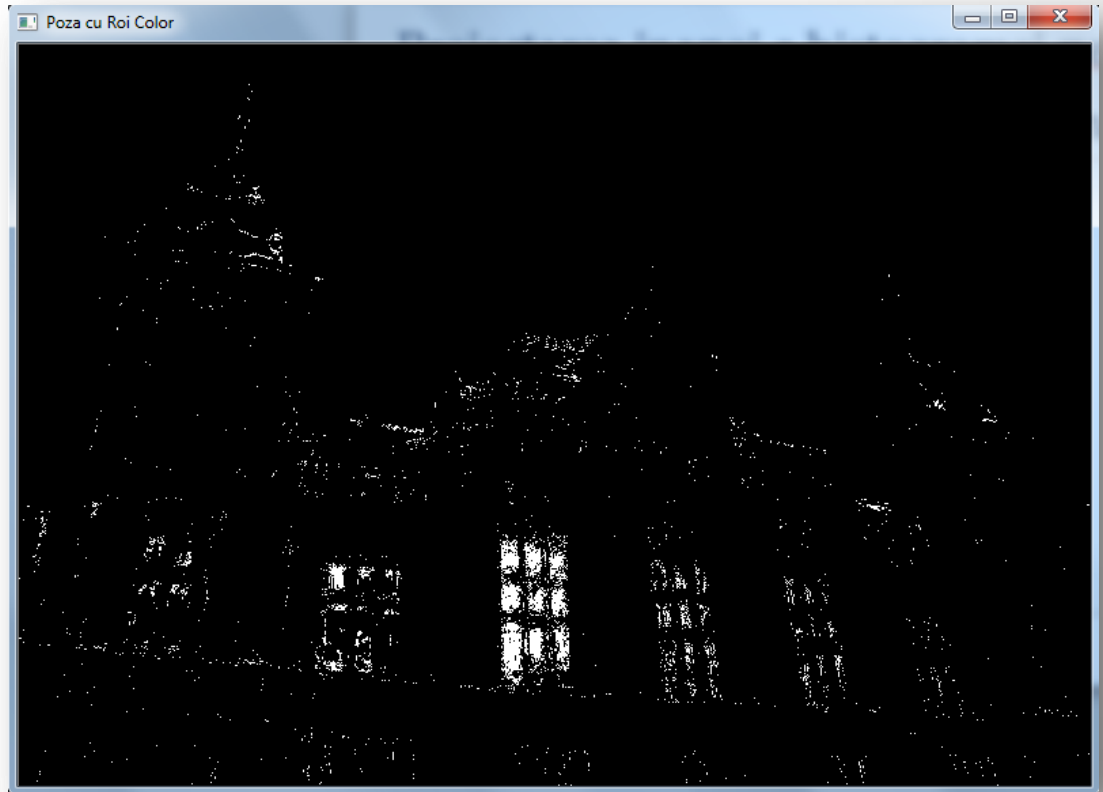
Mat rezColor;
MatND histColor = hc.getHistogram(pozaROIColor);

ObjectFinder obFindColor;
obFindColor.setHistogram(histColor);
obFindColor.setThreshold(0.01f);
rezColor = obFindColor.find(pozaColor);
```



# Proiectarea inapoi a histogramei pentru a detecta un anumit continut din imagine

- In imagine:  
fereastra  
obtinuta cand s-a  
utilizat o  
histograma color
- Valoarea  
parametrului  
prag poate fi  
modificata  
pentru a se  
atinge o mai  
buna precizie

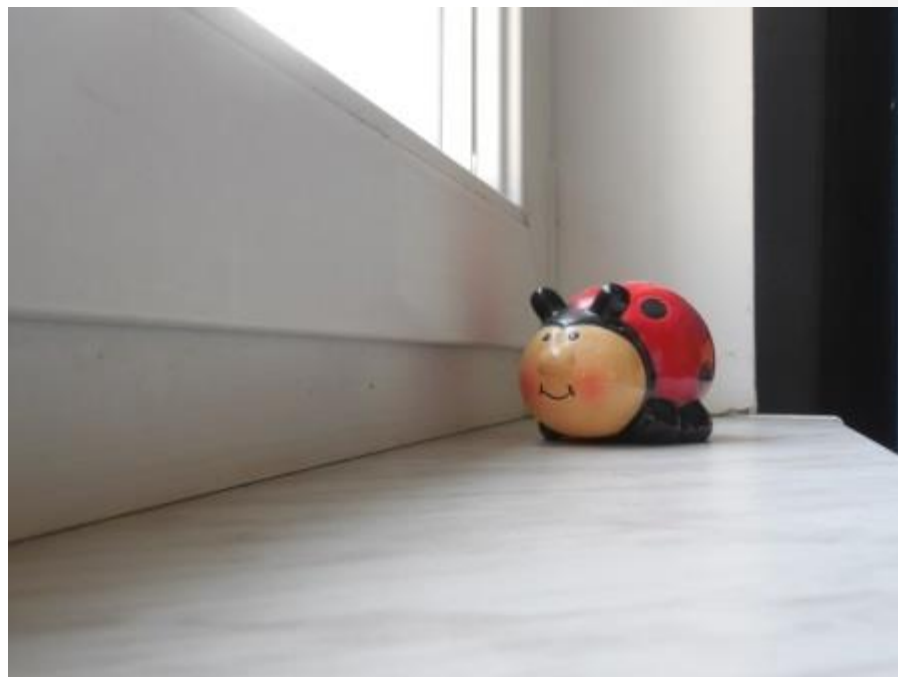


# Folosirea algoritmului mean-shift pentru a detecta obiecte

- Vom utiliza harta de probabilitate calculata anterior cu metoda **find** astfel:
- Intr-o imagine este identificat obiectul cu ROI
- Se calculeaza harta de probabilitate pentru a identifica locatia sa
- Luam o imagine noua in care apare obiectul mutat pentru a il gasi.
- Pentru aceasta vom folosi metoda **mean shift**.

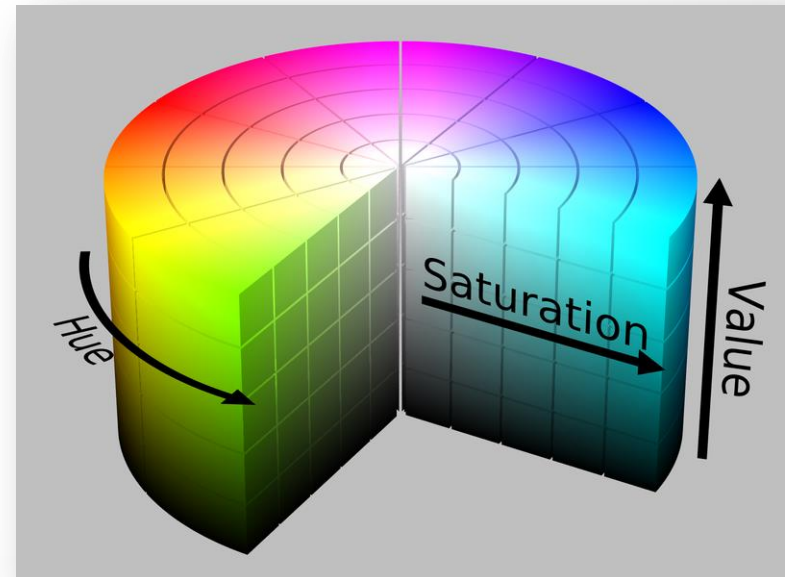
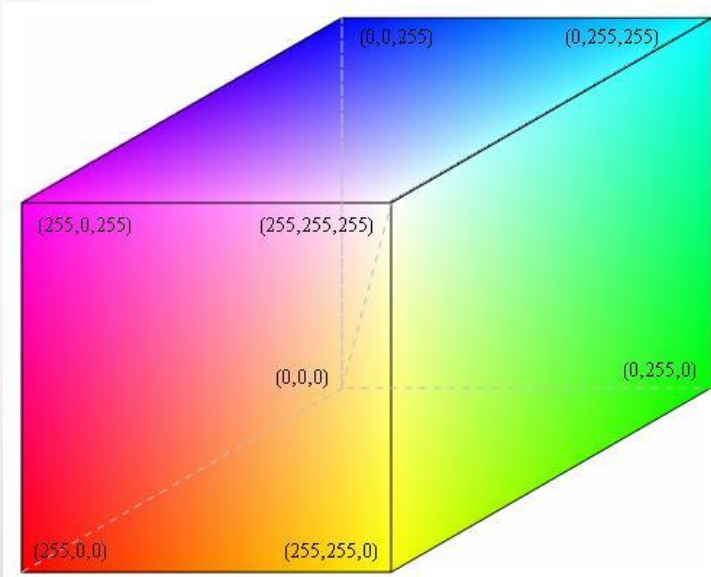
# Folosirea algoritmului mean-shift pentru a detecta obiecte

- Pornim de la imaginea din stanga si alegem ca obiect fața buburuzei.
- Incercam sa o identificam apoi in poza din dreapta.

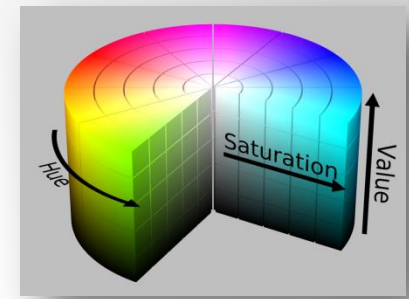


# Folosirea algoritmului mean-shift pentru a detecta obiecte

- Vom utiliza canalul HUE al spatiului de culori HSV.
- Va trebui
  1. Sa convertim imaginea intr-una HSV
  2. Sa extragem canalul hue
  3. Sa calculam histograma 1D a ROI



# HSV



- In HSV avem o reprezentare cilindrica fata de cubul din RGB.
- In HSV se incepe de la rosu primar (0), se merge spre verde (120), apoi albastru (240) si inapoi la rosu (360)
  - toate culori primare la acele limite.
- Centrul cilindrului este gri.
- HSV vine de la HUE (culoare), saturatie (umbra, nivel de gri) si valoare (luminozitate)
- Spatiul culorilor din HSV seamana cu modul in care oamenii percep culorile.

# Folosirea algoritmului mean-shift pentru a detecta obiecte

- Convertim imaginea intr-una HSV
- Extragem canalul hue
- Calculam histograma 1D a ROI
- Totul este scris in main.cpp

```
Mat image= imread("D:/pic1.jpg");//buburuza 1
// Definim ROI
Rect r = Rect(775,370,110,110);//fata buburuzei 1
Mat imageROI= image(r);
rectangle(image, r, Scalar(0,0,255));

// Display image
namedWindow("Poza 1");
imshow("Poza 1",image);

// Calculam histograma HUE
int minSat=60;//parametru care poate fi schimbat

ColorHistogram hc;
MatND colorhist= hc.getHueHistogram(imageROI, minSat);
```



# Metoda getHueHistogram

- Metoda este adaugata in cadrul clasei ColorHistogram
  - Tip public

```
// Calculeaza histograma 1D Hue cu mask.
// Sursa BGR este convertita la HSV
// Pixelii cu saturatie mica sunt ignorati
MatND getHueHistogram(const Mat &image, int minSaturation=0)
{
    MatND hist;
    // Convertim la HSV
    Mat hsv;
    cvtColor(image, hsv, CV_BGR2HSV);
    Mat mask;
    if (minSaturation > 0)
    {
        // Se impart cele 3 canale in 3 imagini
        vector<Mat> v;
        split(hsv, v);
        // Eliminam pixelii cu saturatie scazuta
        threshold(v[1], mask, minSaturation, 255, THRESH_BINARY);
    }
    // Pregatim argumentele pentru o histograma 1D hue
    hranges[0]= 0.0;
    hranges[1]= 180.0;
    channels[0]= 0; // the hue channel
    // Compute histogram
    calcHist(&hsv,
    1, // histogram of 1 image only
    channels, // the channel used
    mask, // binary mask
    hist, // the resulting histogram
    1, // it is a 1D histogram
    histSize, // number of bins
    ranges // pixel value range
    );
    return hist;
}
```

# Folosirea algoritmului mean-shift pentru a detecta obiecte

- Afisam mai jos cea de a doua imagine folosind doar canalul de saturatie caruia ii aplicam si un thresholding.

```
// A doua imagine
image= imread("D:/pic2.jpg");

// Convertim la spatiul HSV
Mat hsv;
cvtColor(image, hsv, CV_BGR2HSV);

// Impartim imaginea
vector<Mat> v;
split(hsv,v);
//v[0] = hue
//v[1] = saturatie
//v[2] = valoare

// Eliminam pixelii cu saturatie scazuta - folosim v[1]
threshold(v[1],v[1],minSat,255,THRESH_BINARY);
namedWindow("Saturatie");
imshow("Saturatie",v[1]);
```





# Gasirea de obiecte folosind un singur canal

- Adaugam in clasa `ObjectFinder` inca o metoda find in care sa poate fi modificate limitele, canalele si numarul de canale.

```
Mat find(const Mat& image, float minValue, float maxValue, int *_channels, int dim)
{
    Mat result;
    hranges[0]= minValue;
    hranges[1]= maxValue;
    for (int i=0; i<dim; i++)
        channels[i]= *_channels[i];
    if (isSparse) // apelam functia potrivita in functie de tipul histogramei
        calcBackProject(&image,
            1, // o imagine
                channels, // vector care specifica ce dimensiuni ale histogramei apartin la ce canale de imagine
                shistogram, // histogram utilizata de tip SparseMat
                result, // imaginea proiectata inapoi rezultata
                ranges, // intervalele pentru fiecare dimensiune
                255); // factorul de scalare care trimite valorile
                // din [0, 1] catre {0, ..., 255} prin inmultirea cu 255

    else//histograma de tip MatND
        calcBackProject(&image, 1, channels, histogram, result, ranges, 255);

    // Threshold back projection to obtain a binary image
    if (threshold>0.0)
        cv::threshold(result, result, 255*threshold, 255, THRESH_BINARY);
    return result;
}
```

# Folosirea algoritmului mean-shift pentru a detecta obiecte

- Cand saturatia unei culori este scazuta, informatia hue devine nestabila, neconcludenta.
  - Pentru culori cu saturatie scazuta, componentele B, G si R sunt aproape egale.
- In consecinta, componentele hue ale culorilor cu saturatie scazuta sunt ignorate.
- Pentru aceasta, folosim un parametru minSat si pixelii cu saturatie sub acest nivel sunt ignorati.
- Operatorul bitwise\_and elimina pixelii cu saturatie scazuta inainte de a apela meanShift.

# Folosirea algoritmului mean-shift pentru a detecta obiecte

- Inapoi in main.cpp.
- De la locatia initiala a dreptunghiului ( $r$ ), algoritmul meanShift va actualiza pozitia
  - noul dreptunghi va fi desenat cu verde.

```
Mat rez;

// Proiectarea inapoi a histogramei HUE
int ch[1]={0}; //vectorul are un singur element, care este 0, pt HUE

// Proiectarea inapoi a histogramei HUE
finder.setThreshold(-1.0f);
rez= finder.find(hsv,0.0f,180.0f,ch,1);
// Eliminam pixelii cu saturatie scazuta
bitwise_and(rez,v[1],rez); //calculeaza conjunctia pe biti
namedWindow("Rezultatul Hue and");
imshow("Rezultatul Hue and",rez);

rectangle(image, r, Scalar(0,0,255));

TermCriteria criteria(TermCriteria::MAX_ITER,10,0.01);
cout << "meanshift= " << cv::meanShift(rez,r,criteria) << endl;

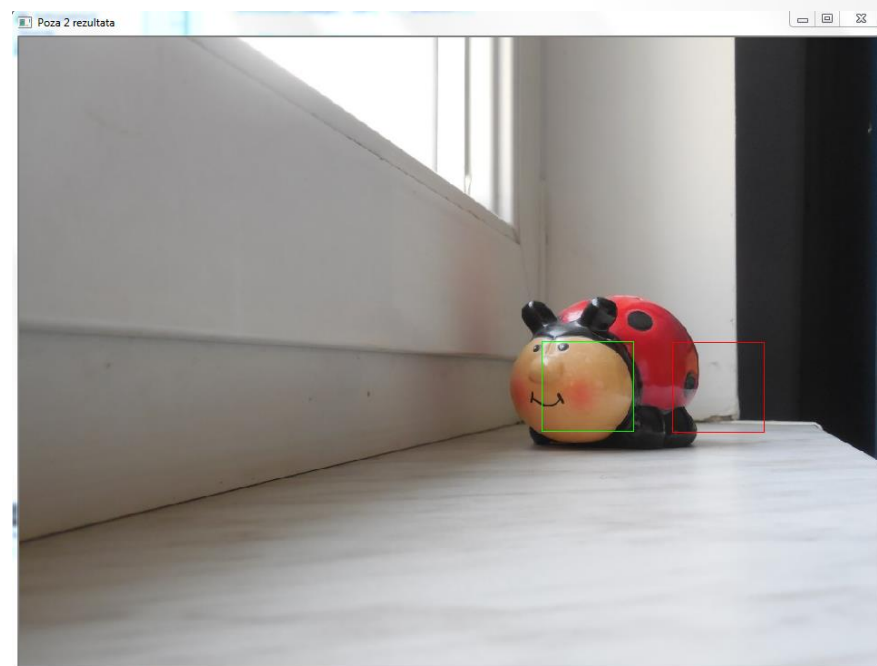
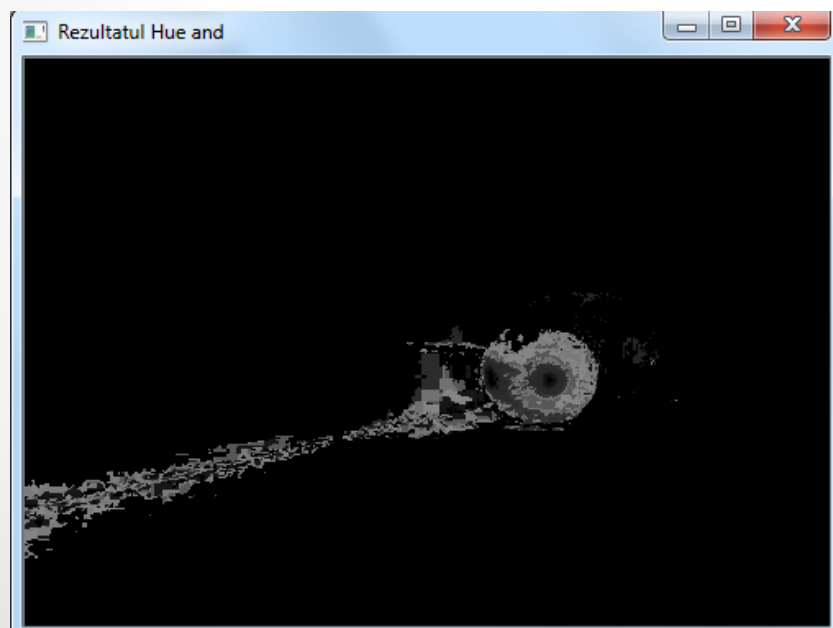
rectangle(image, r, Scalar(0,255,0));

// Afisam poza 2
namedWindow("Poza 2 rezultata");
imshow("Poza 2 rezultata",image);
```

# Algoritmul mean shift

- Algoritmul mean shift este o procedura iterativa care localizeaza maximele locale ale unei functii de probabilitate.
- Mijlocul dreptunghiului se muta de la o locatie la alta pana cand centrul converge catre un punct stabil.
- Doua criterii pentru oprire sunt utilizate:
  - Un numar maxim de iteratii
  - O valoare de plasare a centrului dreptunghiului sub care pozitia este considerata ca a fi convex deja catre un punct stabil.
- Metoda meanShift intoarce numarul de iteratii dupa care algoritmul s-a oprit.

# Folosirea algoritmului mean-shift pentru a detecta obiecte



# Similaritatea dintre imagini folosind histogramele lor

- Pentru a compara 2 imagini folosind histogramele lor  $H_1$  si  $H_2$  se pot folosi 4 metrici distincte
- **Corelatia (CV\_COMP\_CORREL )**

$$d(H_1, H_2) = \frac{\sum_I (H_1(I) - \bar{H}_1)(H_2(I) - \bar{H}_2)}{\sqrt{\sum_I (H_1(I) - \bar{H}_1)^2 \sum_I (H_2(I) - \bar{H}_2)^2}}$$

$$\bar{H}_k = \frac{1}{N} \sum_J H_k(J)$$

- $N$  este numarul de pozitii din histograma
- O valoare mai mare corespunde la o potrivire mai buna.
- Potrivire perfecta da valoarea 1, nepotrivire totala: -1.
- Valoarea 0 inseamna lipsa corelatiei.

# Similaritatea dintre imagini folosind histogramele lor

- **Chi-patrat (CV\_COMP\_CHISQR)**

$$d(H_1, H_2) = \sum_I \frac{(H_1(I) - H_2(I))^2}{H_1(I)}$$

- O valoare mai mica reprezinta o potrivire mai buna
- Potrivire perfecta este 0, nepotrivirea duce la numere foarte mari.
- **Intersectia (CV\_COMP\_INTERSECT)**

$$d(H_1, H_2) = \sum_I \min(H_1(I), H_2(I))$$

- O valoare mai mare semnifica o buna potrivire.

# Similaritatea dintre imagini folosind histogramele lor

- Distanța Bhattacharyya  
(**CV\_COMP\_BHATTACHARYYA**)

$$d(H_1, H_2) = \sqrt{1 - \frac{1}{\sqrt{H_1 H_2 N^2}} \sum_I \sqrt{H_1(I) \cdot H_2(I)}}$$

- Valori mici reprezintă potriviri bune
- Potrivirea perfectă întoarce valoarea 0
- Nepotrivirea totală întoarce 1
- Pentru unele comparații (intersecția, de exemplu) este utilă o normalizare anterioară.



# Similaritatea dintre imagini folosind histogramele lor

```
class ImageComparator
{
private:
    Mat reference;
    Mat input;
    MatND refH;
    MatND inputH;
    ColorHistogram hist;
    int div;
public:
    ImageComparator()
    {
        div = 32;
    }

    // Folosim un factor de reducere a culorilor
    // Facem compararea pe imagini cu factorul de reducere aplicat
    void setColorReduction( int factor)
    {
        div= factor;
    }

    int getColorReduction()
    {
        return div;
    }

    void setReferenceImage(const Mat& image)
    {
        reference= hist.reducereCulori(image,div);
        refH= hist.getHistogram(reference);
    }
}
```

In main.cpp

```
Mat pozaColor = imread("D:/pic2.jpg");

Mat pozaColor2 = imread("D:/pic.jpg");

namedWindow("Poza 1", WINDOW_NORMAL);
resizeWindow("Poza 1", 300, 210);
imshow("Poza 1", pozaColor);

namedWindow("Poza 2", WINDOW_NORMAL);
resizeWindow("Poza 2", 300, 210);
imshow("Poza 2", pozaColor2);

ImageComparator c;
c.setReferenceImage(pozaColor);
cout<<"Iata cat sunt de similare: "<<c.compare(pozaColor2);
```

# Similaritatea dintre imagini folosind histogramele lor

```
double multipleComparison(const Mat& image, int type)
{
    input= hist.reducereCulori(image,div);
    inputH= hist.getHistogram(input);




    string rez;
    rez = "Compar folosind ";
    double compRez = 0;

    switch(type)
    {
        case 0://CV_COMP_CORREL
            rez += "CORREL";
            compRez = compareHist(refH, inputH, CV_COMP_CORREL);
            break;//CV_COMP_BHATTACHARYYA
        case 1://CV_COMP_CHISQR
            rez += "CHISQR";
            compRez = compareHist(refH, inputH, CV_COMP_CHISQR);
            break;
        case 2://CV_COMP_INTERSECT
            rez += "INTERSECT";
            compRez = compareHist(refH, inputH, CV_COMP_INTERSECT);
            break;
        case 3://CV_COMP_BHATTACHARYYA
            rez += "BHATTACHARYYA";
            compRez = compareHist(refH, inputH, CV_COMP_BHATTACHARYYA);
            break;
        default://
            cout<<"Sunt numai 4 posibilitati: 0-3";
            break;
    }
    cout<<rez<<": ";
    return compRez;
}
```

In main.cpp

```
for(int i = 0; i < 4; i++)
    cout<<c.multipleComparison(pozaColor2, i)<<endl;
```

# Similaritatea dintre imagini folosind histogramamele lor

Comparatii	Corelatia	Chi-patrat	Intersectia	Bhattacharyya
	1	0	786432	0
	0.991442	251399	715280	0.119236
	0.113655	3.11409e+007	93611	0.764144

# Proiecte 1/4

1. Realizati un program cu GUI care sa permita utilizatorului sa incarce o imagine si sa ii aplice o segmentare folosind thresholding. Utilizatorul trebuie sa poata alege din cele 5 tipuri de thresholding si sa tunifice valorile celor 2 praguri prin slidere.
2. Realizati un program cu GUI care sa permita utilizatorului sa incarce o imagine in format alb-negru si sa ii calculeze si afiseze ca imagine histograma.

# Proiecte 2/4

3. Realizati un program cu interfata grafica ce presupune alegerea unei imagini color si desenarea celor 3 histograme care se pot obtine din poza. Toate componentele se vor pune in aceeasi fereastră.
4. Realizati un program cu interfata grafica in care sa se introduca o imagine si pentru aceasta sa se aplice un tablou lookup pentru a o modifica. Fereastră afiseaza ambele imagini.

# Proiecte 3/4

5. Realizati un program cu GUI care sa permita utilizatorului sa incarce o imagine si sa ii amelioreze contrastul prin diferite metode care sa poata fi selectate de catre utilizator:
  - Strangerea imaginii cu un parametru ce poate fi stabilit
  - Egalizarea imaginii
6. Realizati un proiect cu interfata grafica in care sa se poata desena cu ajutorul mouse-ului linii dreptunghiuri sau cercuri. Forma se alege prin intermediul unor butoane radio.

# Proiecte 4/4

7. Sa se aleaga o regiune de interes (ROI) dintr-o imagine si sa se aplice o reducere de culori doar la acea subsectiune din imagine. Imaginea se incarca prin GUI si la fel se pot da valorile pentru pozitia ROI.
8. Faceti o aplicatie in care sa se poata stabili o imagine de comparat si apoi sa se dea o cale catre un folder cu poze si sa se gaseasca si afiseze poza care seamana cel mai mult cu cea initiala.