# Computer Vision

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

# Obiective

- Accesarea valorilor pixelilor
- Scanarea unei imagini cu pointeri
- Scanarea unei imagini cu iteratori
- Masurarea eficientei prin timpi de executie
- Operatii aritmetice aplicate la imagini
- Definirea de regiuni de interes

# Generalitati imagini

- O imagine este o matrice de valori numerice
- Fiecare element din matrice este un pixel
- In imaginile fara culori (alb-negru) pixelii sunt valori unsigned 8-bit
  - 0 = negru
  - 255=alb
- Pentru imaginile color sunt necesare 3 astfel de valori pentru a reprezenta canalele culorilor primare rosu, verde si albastru (RGB)
  - Matricea este formata din triplete de valori

# Accesarea valorilor pixelilor

- Trebuie sa specificam numerele corespunzatoare liniilor si coloanelor
  - Daca este alb-negru se obtine un numar
  - Daca este color se obtin 3 valori numerice
- Consideram imaginea de mai jos

# Adaugarea de zgomot la o imagine

- Alegem la intamplare un numar de pixeli din imagine si ii modificam in alb

```
void zgomot(Mat &image, int n)
{
    for (int k=0; k<n; k++)
    {
        int i = rand()%image.rows;
        int j = rand()%image.cols;
        if (image.channels() == 1) // imagine alb-negru
            image.at<uchar>(i, j)= 255;
        else
            if (image.channels() == 3)
            { // imagine color
                image.at<Vec3b>(i, j)[0]= 255;
                image.at<Vec3b>(i, j)[1]= 255;
                image.at<Vec3b>(i, j)[2]= 255;
            }
    }
}
```

- Apelul:

```
zgomot(poza, 5000);
```

# Adaugarea de zgomot la o imagine

- Variabila poza este de tip Mat.
- Dupa apelarea metodei zgomot, afisam poza in fereastra.
- Rezultatul:

# Adaugarea de zgomot la o imagine

- Pentru accesul la elemente am folosit metoda `at(int x, int y)`.
- Acestei metode trebuie sa i se precizeze insa tipul elementelor avute.
  - Nu realizeaza conversii.
- `Vec3b` reprezinta tipul de vector cu `3 uchar` definit de OpenCV pentru a specifica 3 canale cu valori 8-bit.
- Valoarea dintre [ ] reprezinta numarul canalului.

```
void zgomot(Mat &image, int n)
{
    for (int k=0; k<n; k++)
    {
        int i = rand()%image.rows;
        int j = rand()%image.cols;
        if (image.channels() == 1) // imagine alb-negru
            image.at<uchar>(i, j)= 255;
        else
            if (image.channels() == 3)
            { // imagine color
                image.at<Vec3b>(i, j)[0]= 255;
                image.at<Vec3b>(i, j)[1]= 255;
                image.at<Vec3b>(i, j)[2]= 255;
            }
    }
}
```

# Adaugarea de zgomot la o imagine

- Cand se cunoaste de la inceput tipul matricei, se poate folosi `Mat_<uchar>` - sau de alt tip decat uchar
  - Este o subclasa a lui Mat

```
Mat_<uchar> im = poza;//poza de tip Mat
im(25, 32) = 0;// linia 25 si coloana 32
```

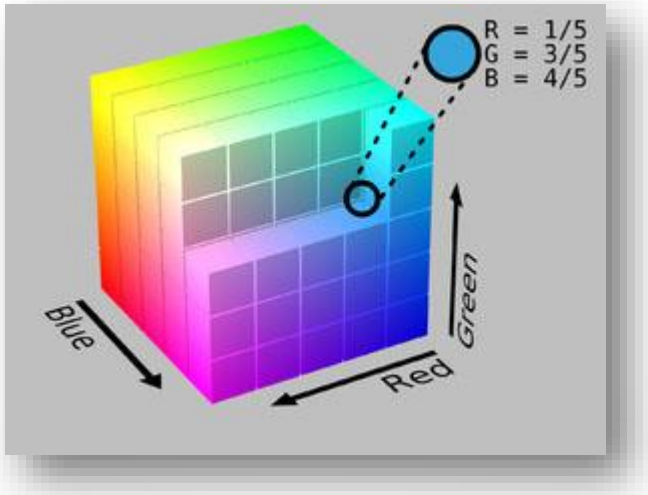
- Aceasta modalitate de accesare a pixelilor produce acelasi efect cu `at(...)`

# Scanarea unei imagini cu pointeri

- Vom reduce in continuare numarul de culori dintr-o imagine.
- Cum fiecare valoare dintr-un canal al unei poze color este un 8-bit uchar, numarul total de culori este 256x256x256
  - Adica peste 16 milioane de culori
- Pentru a reduce complexitatea dintr-o analiza, este adesea util sa reducem numarul de culori.
- O modalitate de reducere este prin subdiviziunea spatiului RGB in cuburi de marimi egale.

# Reducerea numarului de culori

- Daca reducem numarul de culori in fiecare dimensiune de 8 ori, obtinem un total de 32x32x32.
- Fiecare culoare din imaginea originala este inlocuita de o culoare in noua imagine care corespunde valorii din centrul cubului la care apartine.



# Reducerea numarului de culori - algoritm

- N = factorul de reducere
- Fiecare pixel din fiecare canal se imparte la N (impartire intreaga).
- Se multiplica apoi rezultatul cu N
- Se adauga N/2 pentru a obtine pozitia centrala a intervalului intre 2 multipli de N
- Se vor obtine astfel un total de 256/N x 256/N x 256/N culori.

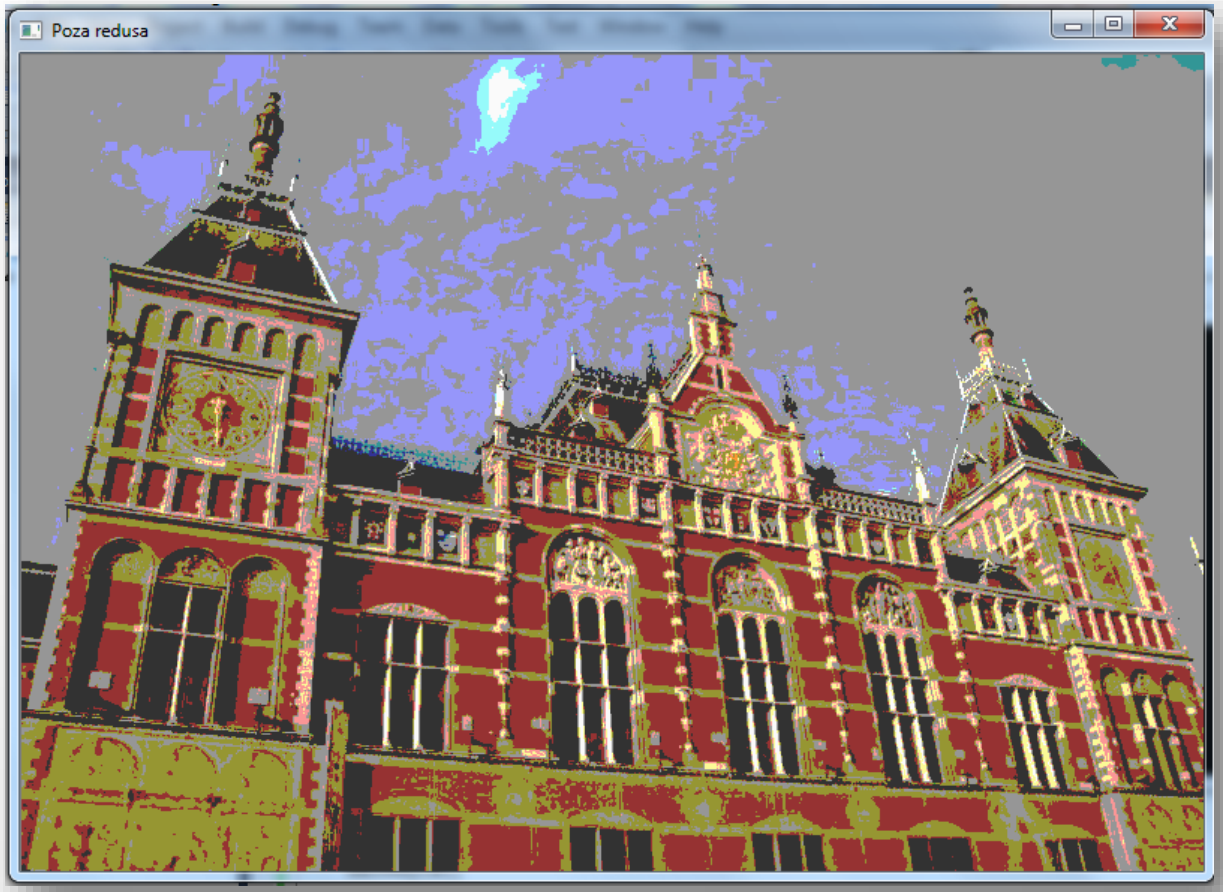
# Reducerea numarului de culori

- Metoda primeste ca argumente imaginea (adresa ei) si un factor de reducere
- Modificarea o facem direct pe imagine
- Apelul din main: `reducereCulori(poza, 100);`

```
void reducereCulori(Mat &image, int div)
{
    int nl= image.rows; // numarul de linii
    // numarul total de elemente pe linie
    int nc= image.cols * image.channels();
    for (int j=0; j<nl; j++)
    {
        // luam adresa unei linii j
        uchar* data= image.ptr<uchar>(j);
        for (int i=0; i<nc; i++)
            data[i]= (data[i]/div)*div + div/2;
    }
}
```

# Reducerea numarului de culori

- Dupa afisarea imaginii, rezultatul este urmatorul:



# Reducerea numarului de culori

- O imagine de lungime L si inaltime h ocupa L x h x 3 uchar.
- Unele imagini au insa extra pixeli care sunt utilizati doar de unele procesoare.
- Valorile acestor extra pixeli sunt insa ignorati.
- Atributele **`cols`** si **`rows`** ale obiectului de tip Mat contin numarele de coloane si linii ale imaginii.
- Atributul **`step`** al obiectului de tip Mat contine lungimea in numarul de bytes a unei linii.
- Numarul de canale este dat de metoda `channels()`: 1 pt grayscale si 3 pt color
- Metoda **`total()`** da numarul total de pixeli.

# Alte modalitati de reducerea a numarului de culori

- Metoda utilizata anterior:

```
data[i]= (data[i]/div)*div + div/2;
```

- Alta metoda de a calcula aceleasi valori :

```
data[i]= data[i] – data[i]%div + div/2;
```

  - Este ineficienta pentru ca citeste de doua ori valoarea pixelilor
- Alta optiune mai eficienta foloseste operatori pe biti
  - Restrangem factorul de reducere la o putere a lui 2, adica div = 2^n.
  - Aplicam masking la primii n biti:

```
uchar mask= 0xFF<<n;
data[i] = (data[i]&mask)+div/2;
```

# Reducerea numarului de culori – input si output

- Pentru a pastra si imaginea initiala intacta, trimitem ca argument catre metoda si un argument de iesire.
- Pentru a crea o imagine identica, folosim metoda `clone()`.

```
void reducereCulori(Mat &image, Mat &rez, int div)
{
    rez.create(image.rows, image.cols, image.type());//cream o imagine cu aceleasi dimensiuni si tip
    int nl= image.rows; // numarul de linii
    // numarul total de elemente pe linie
    int nc= image.cols * image.channels();
    for (int j=0; j<nl; j++)
    {
        // luam adresa unei linii j
        uchar* data_in = image.ptr<uchar>(j);
        uchar* data_out = rez.ptr<uchar>(j);
        for (int i=0; i<nc; i++)
            data_out[i]= (data_in[i]/div)*div + div/2;
    }
}
```

# Reducerea numarului de culori – input si output

- Se poate apela cu numele unei poze de intrare, una de iesire si factorul de reducere.
- Poate fi apelata si cu imaginea de intreare data ca argument de intrare si iesire, caz in care metoda este identica cu cea anterioara.
- Metoda `create()` creeaza o imagine continua, adica fara extra pixeli.

# Scanare eficienta a imaginilor continue

- Daca metoda isContinuous() intoarce true, imaginea nu are extra-pixeli.
- Eficienta se obtine astfel din realizarea unui singur loop in loc de 2 imbricate.

```
void reducereCulori(Mat &image, int div)
{
    int nl= image.rows; // numarul de linii
    // numarul total de elemente pe linie
    int nc= image.cols * image.channels();
    if(image.isContinuous())
    {//fara extra pixeli
        nc = nc*nl;
        nl = 1;
    }
    for (int j=0; j<nl; j++)
    {
        // luam adresa unei linii j
        uchar* data= image.ptr<uchar>(j);
        for (int i=0; i<nc; i++)
            data[i]= (data[i]/div)*div + div/2;
    }
}
```

# Scanare a unei imagini cu iteratori

- Iteratorii pot fi declarati in doua moduri:
- `MatIterator_<Vec3b> it;`
- `Mat_<Vec3b>::iterator it;`
- Daca vrem sa incepem cu a doua linie, putem initializa iteratorul la image.begin<Vec3b>() + image.rows.
  - Daca vrem sa ne mutam cu pasi mai mari, putem pune it+=10 pentru a procesa pixelii din 10 in 10.

```
void reducereCulori2(Mat &image, int div)
{
    // obtinem iteratorul la pozitia initiala
    Mat_<Vec3b>::iterator it= image.begin<Vec3b>();
    // si pozitia finala
    Mat_<Vec3b>::iterator itend= image.end<Vec3b>();
    // loop peste toti pixelii
    for ( ; it!= itend; ++it)
    {
        (*it)[0]= ((*it)[0]/div)*div + div/2;
        (*it)[1]= ((*it)[1]/div)*div + div/2;
        (*it)[2]= ((*it)[2]/div)*div + div/2;
    }
}
```

# Masurarea timpilor de procesare

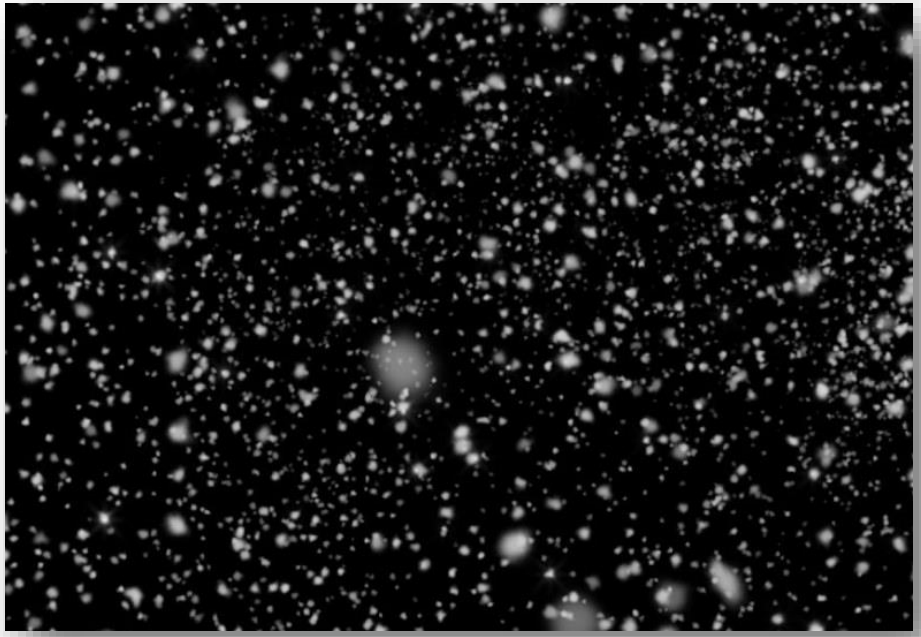
- In procesarea de imagini, timpul este adesea crucial
- Metode utilizate:
- getTickCount()
  - Numarul de cicluri de ceas de la un anumit eveniment.
- getTickFrequency()
  - Numarul de cicluri pe secunda
- Pentru a masura secundele necesare unui proces:
- `double t = (double)getTickCount();`
- `// procesul pentru care facem masuratoarea...`
- `t = ((double)getTickCount()-t)/getTickFrequency();`

# Masurarea timpilor de procesare

- Pentru o masura cat mai corecta, este bine sa se faca mai multe executii (10 sau chiar 30) si sa se calculeze media timpilor.
- Masurati eficienta privitor la timp pentru diversele modalitati de reducere de imagini prezentate anterior.
- Includeti masurarea timpului pentru toate programele urmatoare din cadrul acestui curs.

# Operatii aritmetice aplicate la imagini

- Imaginile pot fi combinate in diverse moduri
- Sunt matrice, deci pot fi adunate, scazute, inmultite, impartite
- Pe langa imaginea considerata initial, luam o a doua imagine care are exact aceleasi dimensiuni si tip



# Operatii aritmetice aplicate la imagini

- Putem utiliza functia `add`, sau, daca vrem sa folosim si ponderi, `addWeighted`
- `addWeighted(poza1,0.7,poza2,0.9,0.,rez);`
- Primele doua poze sunt date, poza `rez` este cea obtinuta.

# Operatii aritmetice aplicate la imagini

- `// c[i]= a[i]+b[i];`
- `add(imageA,imageB,resultC);`

- `// c[i]= a[i]+k;`
- `add(imageA,cv::Scalar(k),resultC);`

- `// c[i]= k1*a[1]+k2*b[i]+k3;`
- `addWeighted(imageA,k1,imageB,k2,k3,resultC);`

- `// c[i]= k*a[1]+b[i];`
- `scaleAdd(imageA,k,imageB,resultC);`

# Definirea de regiuni de interes

- Pozele se pot combina si daca nu au aceleasi dimensiuni
- In acest caz, in poza mai mare se defineste o regiune de interes de dimensiunea pozei mici
- Regiunea de interes este de forma unui dreptunghi
- Pe langa poza initiala, o consideram si pe urmatoarea:

# Definirea de regiuni de interes

- Dimensiunile celor doua poze considerate sunt:
- Poza initiala (mare): 789 x 545
- batman (mica): 300 x 168
- Punctul de inceput al dreptunghiului trebuie sa fie ales astfel incat sa incapa poza mica in cea mare
  - Altfel, avem eroare in cadrul rularii
  - Se poate face o verificare in prealabil

```
//poza1 contine poza cea mare
Mat poza1 = imread("D:/pic.jpg");
//batman contine poza cu sigla
Mat batman = imread("D:/batman.jpg");
//noua imagine pozaROI reprezinta o parte decupata din
//imaginea mare care are exact dimensiunile pozei mici
Mat pozaROI = poza1(Rect(470,5,batman.cols,batman.rows));
// Se unesc cele doua poze care au aceleasi dimensiuni
addWeighted(pozaROI,1.0,batman,0.3,0.,pozaROI);

namedWindow("Poza cu sigla", CV_WINDOW_AUTOSIZE);
//afisam poza initiala in fereastra
imshow("Poza cu sigla", poza1);
```

# Definirea de regiuni de interes

- Modificarile aduse la regiunea de interes se aplica direct in poza mare initiala
- Rezultatul:


-

# Definirea de regiuni de interes

- O modalitate de a defini regiunea de interes este cea care utilizeaza un dreptunghi (Rect)
  - Punctul din cadrul sau indica pozitia de stanga sus
- Alta modalitate presupune definirea de regiuni:
- ```
  Mat pozaROI = poza1(Range(180, 180 + batman.cols),
                      Range(45, 45 + batman.rows));
  ```
- Copierea imaginii mici “batman” in “pozaROI” se poate face si cu:
- `batman.copyTo(pozaROI);`
- La aceasta nu se mai folosesc insa ponderi
  ```

# Proiecte 1/3

- (1p) Realizati o interfata grafica in care se poate face o reducere a numarului de culori dintr-o imagine incarcata de pe calculator.
- Factorul de reducere trebuie sa se poata stabili de catre utilizator prin intermediul interfetei, de preferat printr-un slider.
- Termen: 29 oct

# Proiecte 2/3

- (2p) Stabiliti 2 foldere cu cel putin 3 imagini fiecare care au toate aceleasi tip si dimensiuni.
- Realizati o interfata grafica in care se citesc imagini in 2 paneluri, fiecare din acestea cu cate un buton de next asociat.
- Fiecare panel incarca poze din un folder corespunzator din cele doua initiale.
- Adaugati un panel in care se combina cele doua imagini din panelurile initiale
- Introduceti in interfata si o modalitate de a introduce ponderile, de preferat prin slider.
- Optiune de salvare a imaginii rezultate
- Termen: 5 nov

# Proiecte 3/3

- (2p) Ca si la proiectul precedent, avem posibilitatea sa introducem 2 poze, de data aceasta de oriunde de pe calculator.
- Cele doua imagini pot avea dimensiuni diferite.
- Prima se introduce in cea de a doua la o pozitie (x, y) care se alege de catre utilizator.
- Se face verificarea daca prima poza incape in cea de a doua.
  - Inclusiv daca incape de la pozitia data de utilizator
  - Termen 5 nov

# Exemplu proiect realizat de un student