# Estimarea Hartii de Adancime Folosind Transformata Census si Distanta Hamming

Student: Moldovan Paul Andrei

Grupa: 30235

Indrumator: Muresan Mircea Paul

# Contents

# 1. Transformata Census (CT)

Transformata Census este un operator care asociaza pentru fiecare pixel dintr-o imagine grayscale un sir binar care codifica daca pixelul curent are intensitate mai mica decat vecinii sai, cate o valoare pentru fiecare vecin.

$$C(P) = \bigotimes_{[i,j]\in D} \xi(P, P + [i,j])$$

Simbolul $\bigotimes$ reprezinta concatenarea valorilor, D este fereastra cu care parcurgem imaginea iar P pixelul din centrul ferestrei. Transformata este definite de:

$$\xi(P, P + [i,j]) = \begin{cases} 1, if\ P > P + [i,j] \\ 0, \quad otherwise \end{cases}$$

Un exemplu pentru transformata census:

$$\begin{array}{|c|c|c|} \hline 124 & 74 & 32 \\ \hline 124 & 64 & 18 \\ \hline 157 & 116 & 84 \\ \hline \end{array} \longrightarrow \begin{array}{|c|c|c|} \hline 1 & 1 & 0 \\ \hline 1 & x & 0 \\ \hline 1 & 1 & 1 \\ \hline \end{array} \longrightarrow 11010111$$

Fig.1.1 Exemplu transformata Census

Transformata census converteste diferentele relative ale intensitatilor la 0 sau 1 in vectori de biti. Forma ferestrei pentru transformata census poate sa fie atat patrata cati si dreptunghiulara. Diferite dimensiuni avand rezultate diferite.

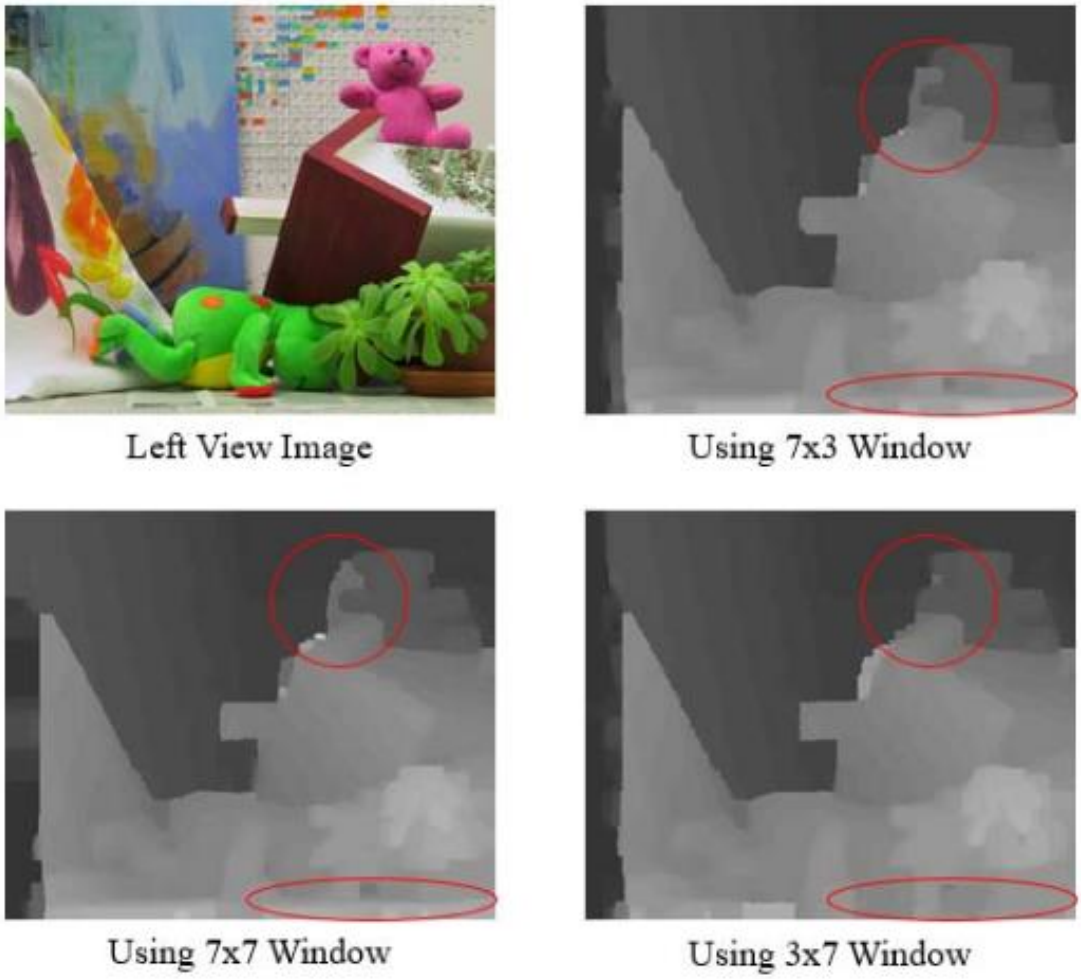



Fig.1.2 Rezultat harta de adancime folosind transformata Census pe ferestre de diferite dimensiuni

## 2. Distanta Hamming

Distanta Hamming dintre doua siruri de lungime egala reprezinta numarul de pozitii cu valori diferite dintre cele doua siruri. Ea masoara numarul minim de substitutii necesare pentru a schimba un sir in celalalt. De exemplu pentru doua siruri de biti **1011101** și **1001001** distanta Hamming este 2.

Pentru a folosii distanta Hamming impreuna cu transformata Census comparam rezultatele transformatei a doi pixel bit cu bit.

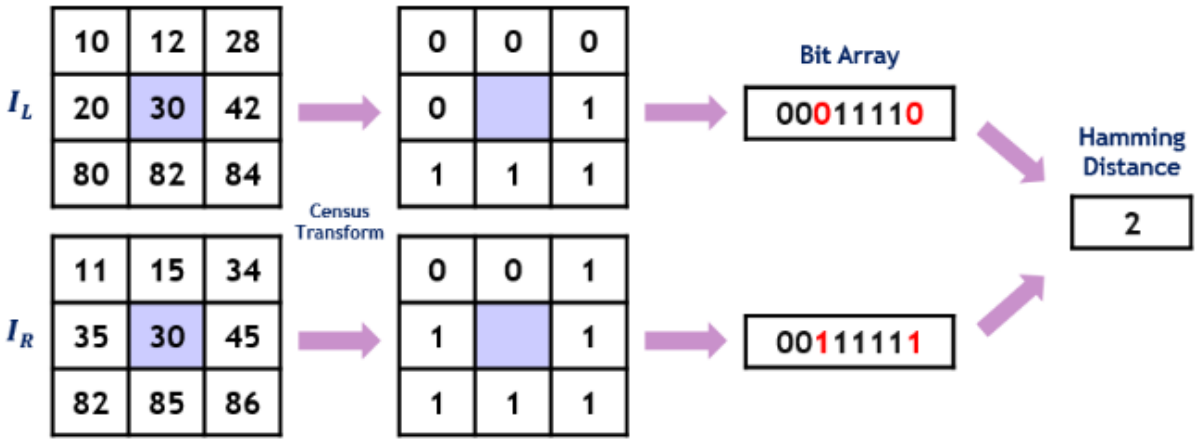



Fig.2.1 Exemplu distanta Hamming pe doi pixeli dupa folosirea transformatei Census

## 3. Harta de disparitate (disparity map/depth map)

***disparity*** =
*position of* ***selected*** *Right block – position of* ***closest matching*** *Left block*

Diferenta d = pl – pr a doua puncte(pixeli) corespunzatoare unei imagine se numeste disparitate. Aceasta arata miscarea pixelilor intre doua imagini stereo. Pentru calcularea ei avem nevoie de pozitia pixelului in prima imagine (imaginea stanga, pl) si pozitia pixelului din a doua imaginea (imaginea dreapta, pr). De exemplu pentru un pixel cu pozitia (60, 30) in imaginea stanga si pozitia (40, 30) in imaginea dreapta disparitatea va fi: 60 – 40 = 20. Deci disparitatea ne da diferenta dintre pozitia pixelului in cele doua imagini.

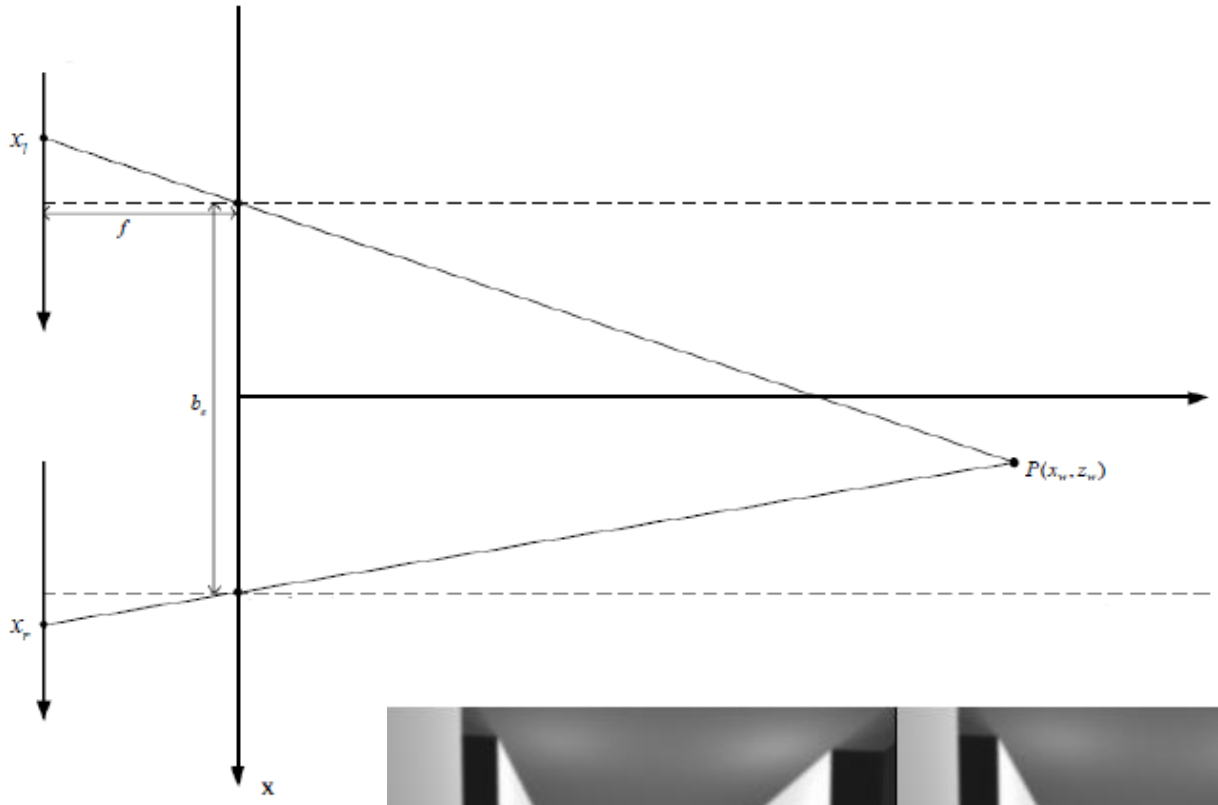
Cautarea adancimii obiectele intr-o imagine este la fel ca si cautarea disparitatii, distanta fiind invers proportionala disparitatii. Adancimea folosind disparitatea este data de formula urmatoare:

$$z_w = -\frac{fb_s}{x_l - x_r}$$

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} - \begin{pmatrix} b_{11} & \cdots & b_{1n} \\ \vdots & \ddots & \vdots \\ b_{m1} & \cdots & b_{mn} \end{pmatrix} = \begin{pmatrix} (a-b)_{11} & \cdots & (a-b)_{1n} \\ \vdots & \ddots & \vdots \\ (a-b)_{m1} & \cdots & (a-b)_{mn} \end{pmatrix}$$

[m x n array for Right block] [m x n array for Left block] [m x n array containing the difference in each pixel]

Imaginea urmatoare reprezinta doua camere (stanga si dreapta) care incearca sa gasesca distanta unui punct P fata de ele.




Dupa cum se vede in aceasta imagine obiectele mai deschise arata distanta mai mica si obiectele mai inchise arata o distanta mai mare.

Ne folosim de harta de disparitate pentru a calcula harta de adancime.

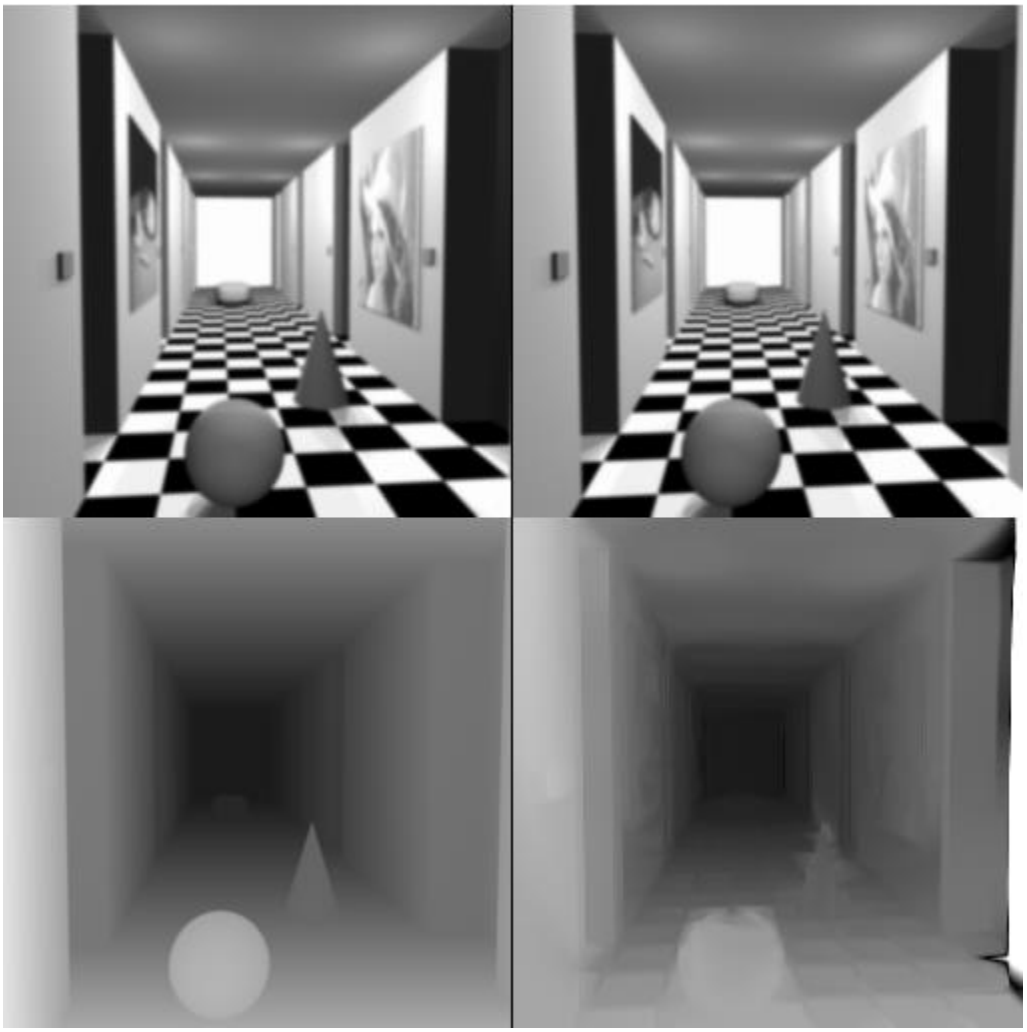


Fig.3.1 Rezultat depthmap (disparity map)

# 4. Rezultate Experimentale

Imaginile originale (left/right) si rezultatul (disparity map).

Fig.4.1 Sample test

Rezultatele mele in urma aplicarii algoritmului cu ferestre de diferite dimensiuni si cu doua distance, Hamming si Manhattan folosite pentru a calcula eroarea minima:

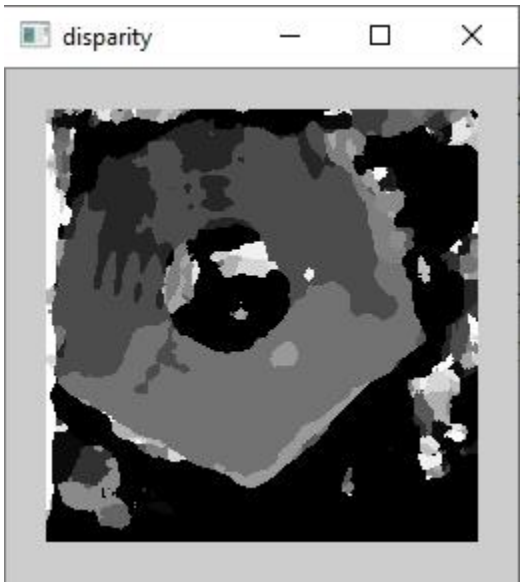



Fig.4.2 Hamming cu fereastra 3x3

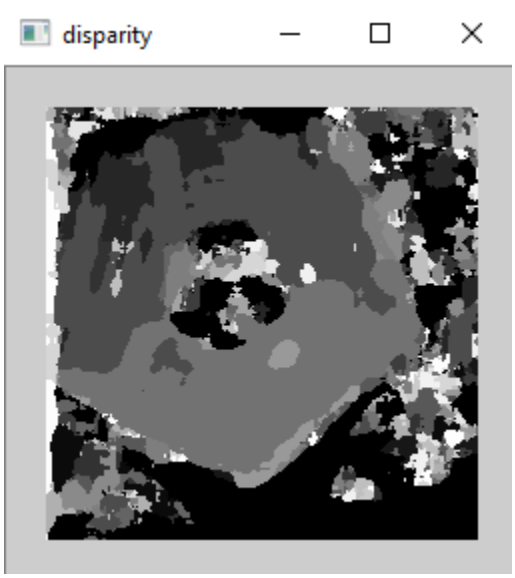



Fig.4.3 Hamming cu fereastra 7x7

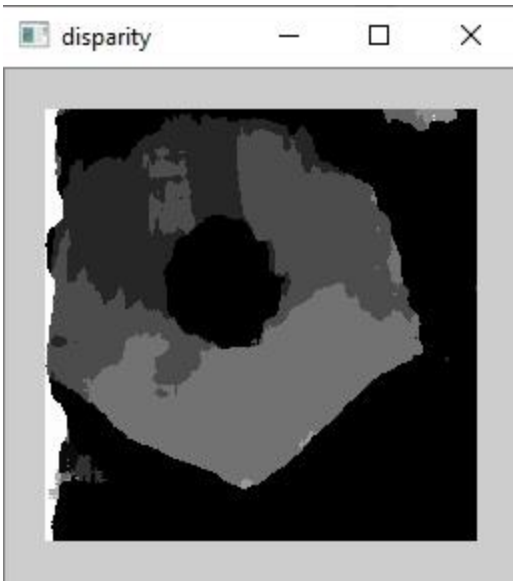



Fig.4.4 Manhattan cu fereastra 11x11


Fig.4.5 Manhattan cu fereastra 7x7

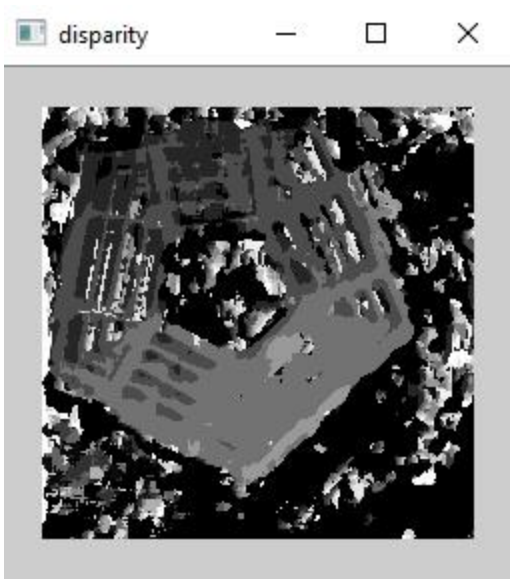



Fig.4.6 Manhattan cu fereastra 3x3

Rezultate in urma aplicarii transformatei Census pentru ferestre de diferite dimensiuni:

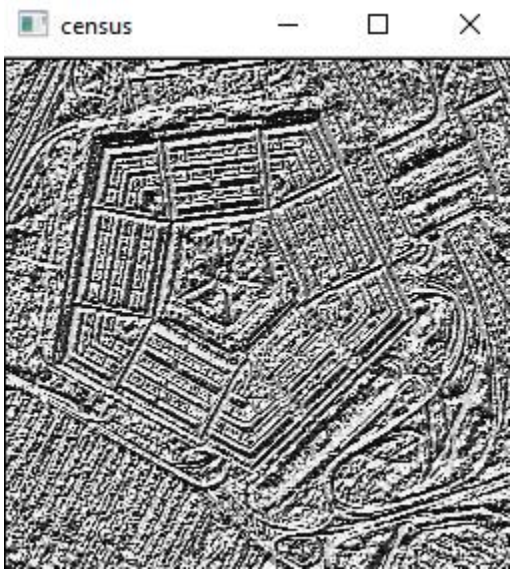



Fig.4.7 Transformata Census 3x3

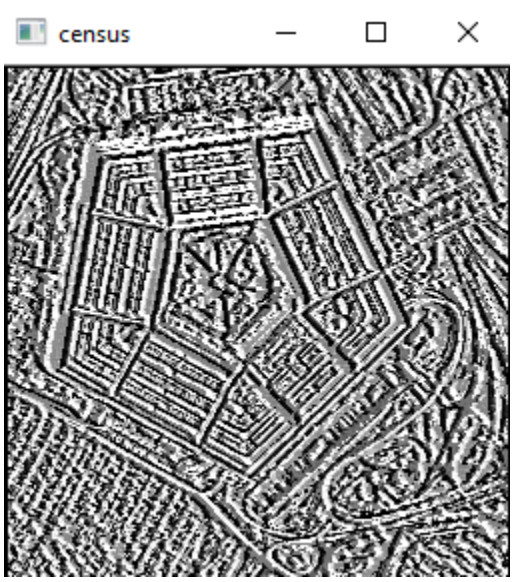



Fig.4.7 Transformata Census 7x3


Fig.4.7 Transformata Census 11x11

# 5. Concluzii

In acest proiect am testat si experimentat cu mapa de adancime implementata folosindu-ma de transformata Census si diferite distance. Pentru imagine mari algoritmul mi se pare foarte costisitor pentru ca timpul de executie este destul de lung. Aceasta metoda necesita optimizare multa pentru diferite conditii.

Am ales sa fac acest proiect pentru ca vreau ca urmare sa estimez adancimea folosind retele neuronale convolutionale care poate sa fie folosita pentru multe lucruri cum ar fi recunoastearea actiunilor.

# 6. Anexa

## 1. Transformata Census

```
uchar* census_transform(Mat img, int h, int v) {

    imshow("original img", img);
    Mat imgTemp = Mat::zeros(img.rows, img.cols, CV_8UC1);  //imagine plina de 0

    unsigned int census = 0;
    int v1 = v / 2;
    int h1 = h / 2;

    //calculez size la data
    uchar* data_;
    int pixelBitSize = v * h - 1;   //calcuzel cati biti o sa fie pe pixel, pt 3x3 => 8biti/pixel pentru ca ignoram pixelul din centru
    int totalPixels = img.rows*img.cols*pixelBitSize;   //numarul total de biti pentru toata imaginea
    int dataSize = (totalPixels + sizeof(uchar) - 1) / sizeof(uchar); //size-ul la data
    data_ = new uchar[dataSize]; //vector de uchar unde salvezi bitii
    memset(data_, 0, dataSize * sizeof(uchar)); //setez tot vectorul 0

    //printf("%d %d %d\n\n", pixelBitSize, totalPixels, dataSize);

    int count = 0;
    int pixel_pos, bit_pos, bit_pos_start;
    uchar val, center_val;
```

Bitii pentru fiecare pixel ii salvez intr-un vector de uchar. Pentru asta am nevoie sa calculezi cati biti o sa fie in total. Calculezi numarul de biti/pixel (pixelBitSize), care este dat de numarul de vecini din fereastra. Dupa numarul total de biti pentru toti pixelii din imagine (totalPixels).

```
//parcurg imaginea pixel cu pixel incepand de la v1/h1 astfel ca sa nu am pading(sa nu iasa kernelul din imagine)
for (int y = v1; y < img.rows - v1; y++) {
    for (int x = h1; x < img.cols - h1; x++) {

        census = 0;
        count = 0;

        pixel_pos = y * img.cols + x;
        bit_pos_start = pixel_pos * pixelBitSize; //pixel bit size

        center_val = img.at<uchar>(y, x);
        //parcurg kernelul
        for (int i = -v1; i <= v1; i++) {
            for (int j = -h1; j <= h1; j++) {

                if (i == 0 && j == 0) continue; //skip center

                int w_x = max(min(x + j, img.cols-1), 0);   //window x
                int w_y = max(min(y + i, img.rows-1), 0);   //window y

                census <<= 1;   //shift cu 1 la stanga (census = census << 1)
                bit_pos = bit_pos_start + count++;  //pentru pixelul curent
                val = img.at<uchar>(w_y, w_x);

                if (val < center_val) { // ...
                    census += 0;
                }
                else {
                    int shift = bit_pos % sizeof(uchar);
                    data_[bit_pos / sizeof(uchar)] |= (1 << shift);
                    //sau logic cu 1 shiftat pentru pozitia curenta

                    census += 1;
                }
            }
        }
        //printf("%d %d %d\n", bit_pos, data_[bit_pos / sizeof(uchar)], census);

        imgTemp.at<uchar>(y, x) = census;
    }
}
//imshow("census", imgTemp);
return data_;
```

Parcurg imaginea pixel cu pixel si salvez in data stringul de pixi pentru fiecare pixel si de asemenea construiesc o imagine rezultat pentru a vizualiza efectul transformatei Census pe imagine. Calculez pozitia de start pentru bitii pixelului curent si pentru fiecare vecin incrementez count cu 1 => pozitia urmatoare. Parcurg fereastra, sar peste pixelul din centru ei, pentru el facem sirul de biti si in data_ pentru pozitia curenta fac sau logic cu 1 shiftat cu shift pozitii.

## 2. Distanta Hamming

```
int getDistance(uchar *data_, int y1, int x1, int y2, int x2, int h, int v, int cols, int rows){
    int v1 = v / 2;
    int h1 = h / 2;
    int pixelBitSize = v * h - 1;

    int bit_pos_start1 = (y1 * cols + x1) * pixelBitSize;
    int bit_pos_start2 = (y2 * cols + x2) * pixelBitSize;

    int dis = 0;
    int count = 0;
    for (int i = -v1; i <= v1; i++){
        for (int j = -h1; j <= h1; j++){

            int bit_pos1 = bit_pos_start1 + count;
            int bit_pos2 = bit_pos_start2 + count;

            count++;
            int bit_val1 = getBitVal(data_, bit_pos1);
            int bit_val2 = getBitVal(data_, bit_pos2);
            printf("%d %d %d %d\n", bit_pos1, bit_pos2, bit_val1, bit_val2);
            dis += bit_val1 != bit_val2;    //pt fiecare bit diferit distanta++
        }
    }
    return dis;
}
```

Calculez pozitia de start pentru bitii la amandoi pixeli si pentru fiecare bit incrementez dis pentru fiecare valori de pe aceeas pozitie diferite.

```
int getDistance2(uchar *data_left, uchar *data_right, int y1, int x1, int y2, int x2, int h, int v, int cols, int rows) {
    int v1 = v / 2;
    int h1 = h / 2;
    int pixelBitSize = v * h - 1;

    int bit_pos_start1 = (y1 * cols + x1) * pixelBitSize;
    int bit_pos_start2 = (y2 * cols + x2) * pixelBitSize;

    int dis = 0;
    int count = 0;
    for (int i = -v1; i <= v1; i++) {
        for (int j = -h1; j <= h1; j++) {

            int bit_pos1 = bit_pos_start1 + count;
            int bit_pos2 = bit_pos_start2 + count;

            count++;
            int bit_val1 = getBitVal(data_left, bit_pos1);
            int bit_val2 = getBitVal(data_right, bit_pos2);
            //printf("%d %d %d %d\n", bit_pos1, bit_pos2, bit_val1, bit_val2);
            dis += bit_val1 != bit_val2;
        }
    }
    return dis;
}
```

Pentru doua imagini diferite.

```
int getBitVal(uchar *data, int pos) {
    uchar elem = data[pos / sizeof(uchar)];
    int shift = pos % sizeof(uchar);
    elem = (elem >> shift);
    return elem & 1;
}
```

Pentru a luat bitul de la pozitia dorita shiftez la dreapta cu pozitia calculate pentru uchar.

## 3. Disparity Map

```
void computeDispMap2(Mat left_img, Mat right_img, int ndisp, int w, int w2) {

    int k = w / 2;
    Mat disp_map = Mat(left_img.rows, left_img.cols, CV_8UC1);

    //calculez transformatele pentru cele doua imagini
    uchar *data_left = census_transform(left_img, w2, w2);
    uchar *data_right = census_transform(right_img, w2, w2);

    int dis = 0;
    float cost = 0;
    long min_cost = 0;
    int w_x, w_y, w_x2;
    //parcug imaginile
    for (int y = ndisp; y < left_img.rows  - ndisp; y++) {
        for (int x = ndisp; x < left_img.cols - ndisp; x++) {

            min_cost = 1e9; //10^9, asignez o valoare foarte mare pentru costul minim
            dis = 0;
            //pentru fiecare disparitate(diferenta)
            for (int d = x-ndisp; d <= x; d++) {    //ndisp = disparitatea maxima

                cost = 0;
                for (int i = -w; i <= w; i++) {         //parcurg kernelul
                    for (int j = -w; j <= w; j++) {

                        w_x = x + j;                //x img1
                        w_y = y + i;                //y img1
                        w_x2 = j + d;               //x2 img2

                        //daca nu e in imagine
                        if (w_x < 0 || w_y < 0 || w_x >= left_img.cols || w_y >= left_img.rows || w_x2 < 0 || w_x2 >= left_img.cols) continue;

                        //calculez distanta Hamming dintre cele doua puncte
                        cost += getDistance2(data_left, data_right, w_y, w_x, w_y, w_x2, w2, w2, left_img.cols, left_img.rows);
                        //cost += abs(left_img.at<uchar>(w_y, w_x) - right_img.at<uchar>(w_y, w_x2));
                    }
                }
                //daca costul calculat e < min_cost atunci devine min cost si avem o noua disparitate d
                if (cost < min_cost) {
                    min_cost = cost;
                    dis = d;
                }
            }
            disp_map.at<uchar>(y, x) = 3*abs(x - dis) * (255. / ndisp); //formula pentru inversul disparitatii (*3 pentru ca imaginea sa fie mai deschisa)
        }
    }

    imshow("disparity", disp_map);
    waitKey(0);
}
```

Parcurg imaginea si pentru disparitate de la [x-ndisp, x] calculez costul (eroarea minima) in functie de care asignez depth-ul in noua imagine.

# 7. Bibliografie


https://opencv.org/

http://vision.middlebury.edu/stereo/data/scenes2014/

https://github.com/mbaird/stereo-disparity-map

https://www.jstage.jst.go.jp/article/transinf/E100.D/11/E100.D_2017EDP7052/_pdf

http://www.apsipa.org/proceedings_2016/HTML/paper2016/277.pdf

http://www.cs.cornell.edu/~rdz/Papers/ZW-ECCV94.pdf

https://aikiddie.wordpress.com/2017/05/24/depth-sensing-stereo-image/