# Computer Vision

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

# Identificare contururi, analiza structurala

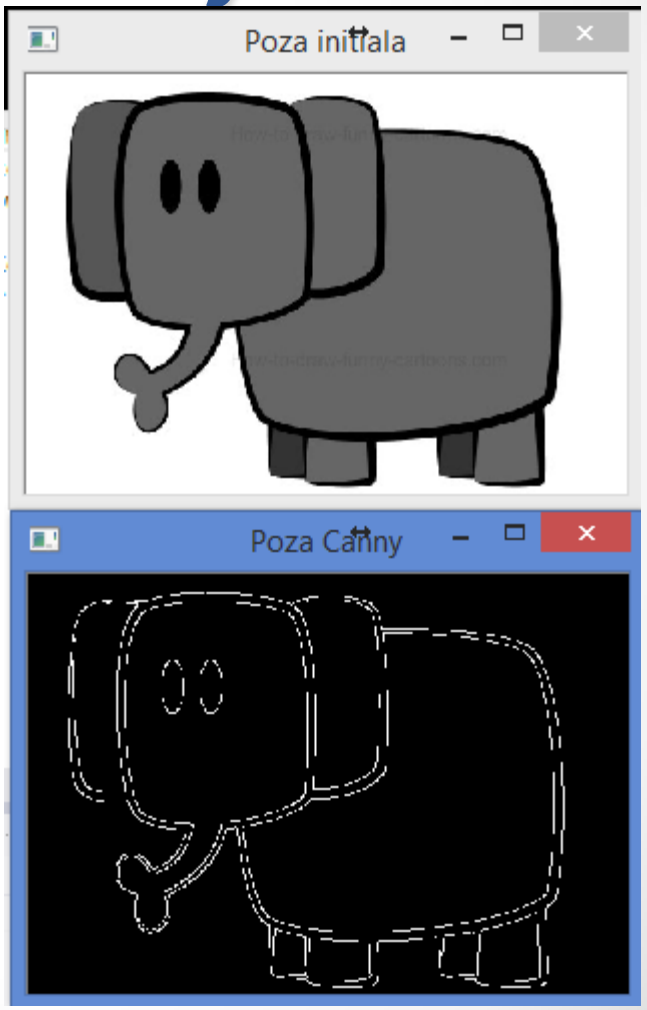
- Detectare margini folosind Canny (inca o metoda ☺)
- Identificare si desenare de contururi
- Cercuri care cuprind poligoane
- Dreptunghiuri care cuprind poligoane
- Ierarhia contururilor detectate
- Compararea a doua forme
- Testare apartenenta punct la poligon

# Detectare margini folosind Canny

- Pentru a extrage contururi, trebuie sa extragem marginile dintr-o imagine
  - Se poate folosi orice metoda prezentata anterior pentru a gasi marginile sau pentru a transforma imaginea intr-una binara, inclusiv threholding binar.
- Algoritmul a fost creat de John F. Canny in 1986.
- Are doua praguri, unul de jos si unul de sus
  - Daca valoarea gradientului unui pixel este mai mare decat valoarea superioara, atunci pixelul curent este considerat a fi parte din margine
  - Daca valoarea gradientului unui pixel este mai mica decat valoarea de jos, atunci acesta este eliminat.
  - Daca valoarea este intre pragul de jos si cel de sus, atunci apare pe o margine doar daca este conectat cu un pixel care ave valoarea peste pragul superior.
- Pentru mai multe detalii despre algoritm, vizitati http://en.wikipedia.org/wiki/Canny_edge_detector

# Detectare margini folosind Canny

```
Mat poza, poza2;

int main()
{
    poza = imread("E:/Test/elefant.jpg", 0);

    Canny( poza, poza2, 100, 200, 3 );

    namedWindow("Poza initiala", CV_WINDOW_AUTOSIZE);
    resizeWindow("Poza initiala", 300, 210);
    imshow("Poza initiala", poza);

    namedWindow("Poza Canny", CV_WINDOW_AUTOSIZE);
    resizeWindow("Poza Canny", 300, 210);
    imshow("Poza Canny", poza2);

    contururi();

    waitKey();
    return 1;
}
```




- La dimensiuni mai mari, imaginea a doua nu are discontinuitati.

- Poza initiala este luata de la how-to-draw-funny-cartoons.com

# Detectare margini folosind Canny

- void Canny(InputArray **image**, OutputArray **edges**, double **threshold1**, double **threshold2**, int **apertureSize**=3, bool **L2gradient**=false )
  - **image** – imaginea de intrare, pe 8 biti, un singur canal (grayscale).
  - **edges** – imaginea rezultata cu marginile detectate din prima.
  - **threshold1** – pragul de jos.
  - **threshold2** – pragul superior.
  - **apertureSize** – marimea aperturii pentru operatorul Sobel().
  - **L2gradient** – data este adevarat, o alta modalitate, mai exacta, de calcul a gradientului este utilizata

# Identificare de contururi

- void findContours(InputOutputArray **image**, OutputArrayOfArrays **contours**, OutputArray **hierarchy**, int **mode**, int **method**, Point **offset**=Point())
  - **image** – o imagine cu un singur canal, de preferat binara. Daca nu este binara, pixelii diferiti de 0 sunt tratati ca 1, ceilalti raman 0
  - **contours** – contururile detectate. Fiecare contur este retinut ca un vector de puncte.
  - **hierarchy** – vector cu atatea elemente cate contururi sunt. Pentru fiecare al i-lea contur (contours[i]),
    - [hierarchy[i][0] , hiearchy[i][1] , hiearchy[i][2] , hiearchy[i][3]] contin informatii despre [urmatorul, anteriorul, primul descendent, parintele] contur.
    - Conturul urmator si anterior se gasesc pe acelasi nivel ierarhic.
    - Daca nu se gaseste alt contur (urmator, anterior, descendent sau parinte), valoarea intoarsa este negativa (-1).

# Identificare de contururi

- void findContours(InputOutputArray **image**, OutputArrayOfArrays **contours**, OutputArray **hierarchy**, int **mode**, int **method**, Point **offset**=Point())
  - **mode** – modul de extragere a contururilor
    - **CV_RETR_EXTERNAL** Intoarce numai contururile din exterior. hierarchy[i][2]=hierarchy[i][3]=-1 (descendentii si parintii) pentru toate contururile.
    - **CV_RETR_LIST** scoate toate contururile fara sa tina cont de ierarhii.
    - **CV_RETR_CCOMP** gaseste toate contururile si le organizeaza pe 2 niveluri. Nivelul superior contine contururile exterioare ale componentelor. Nivelul inferior contine contururile interioare. Daca insa apare un alt contur in interiorul celui interior, acesta este considerat din nou pe nivelul superior.
    - **CV_RETR_TREE** toate contururile cu ierarhie completa.

# Identificare de contururi

- void findContours(InputOutputArray **image**, OutputArrayOfArrays **contours**, OutputArray **hierarchy**, int **mode**, int **method**, Point **offset**=Point())
  - **method** — metoda de aproximare a contururilor
    - **CV_CHAIN_APPROX_NONE** contine toate punctele de contur.
    - **CV_CHAIN_APPROX_SIMPLE** comprima segmentele si lasa numai punctele de final.
    - **CV_CHAIN_APPROX_TC89_L1,CV_CHAIN_APPROX_TC89_KCOS** aplica algoritmul de aproximare a lanturilor Teh-Chin.
  - **offset** – parametru optional prin care fiecare punct de contur este mutat.

# Desenarea contururilor

- void drawContours(InputOutputArray **image**, InputArrayOfArrays **contours**, int **contourIdx**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, InputArray **hierarchy**=noArray(), int **maxLevel**=INT_MAX, Point **offset**=Point() )
  - **image** – Imaginea destinatie.
  - **contours** – Contururile de desenat. Se reprezinta ca vectori de puncte.
  - **contourIdx** – id-ul conturului de desenat.
  - **color** – culoarea contururilor.
  - **thickness** – grosimea liniilor contururilor. Daca este negativa, se deseneaza si interiorul lor.
  - **lineType** – tip de linie (8, 4 sau **CV_AA**).

# Desenarea contururilor

- void drawContours(InputOutputArray **image**, InputArrayOfArrays **contours**, int **contourIdx**, const Scalar& **color**, int **thickness**=1, int **lineType**=8, InputArray **hierarchy**=noArray(), int **maxLevel**=INT_MAX, Point **offset**=Point() )
  - **hierarchy** – Informatie necesara doar daca se doreste desenarea anumitor contururi.
  - **maxLevel** – nivelul maxim pana la care se deseneaza contururi. Are sens numai daca se da hierarchy.
  - **offset** – parametru optional pentru mutarea conturului.

# Identificare si desenare

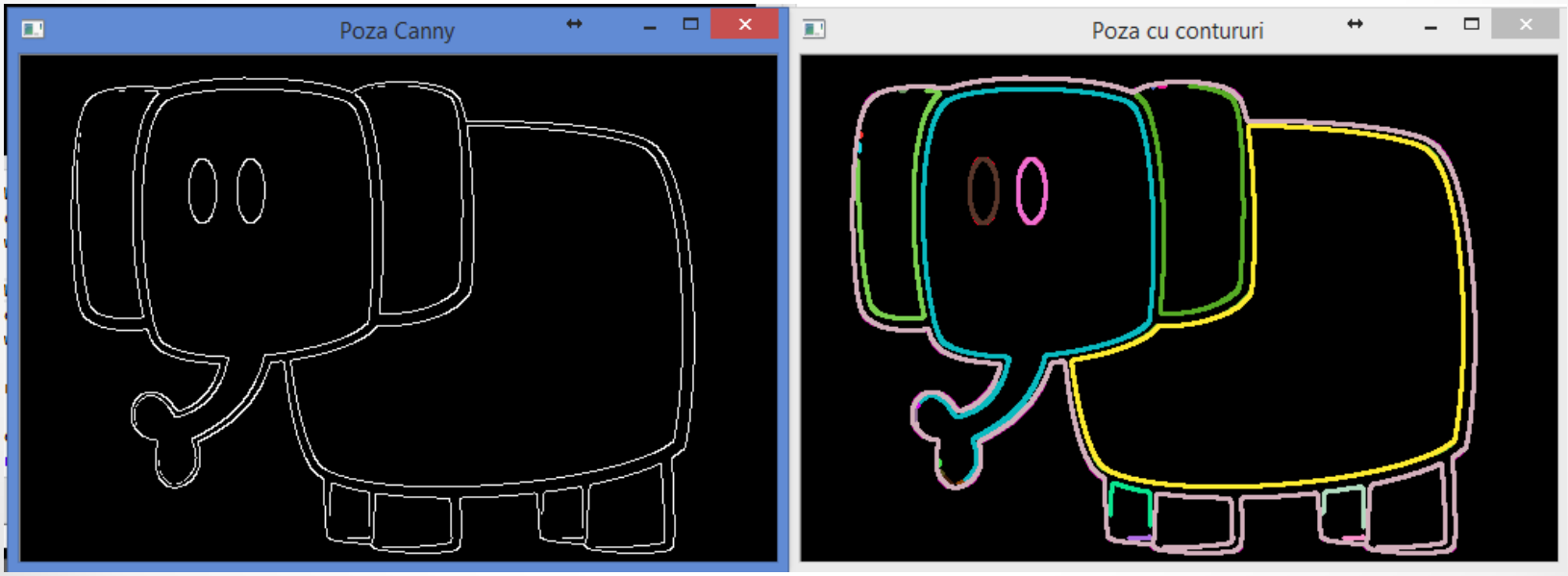
```
Mat poza, poza2;
RNG rng(12345);//Random number generator - valoarea din interior este utilizata pentru initializare
vector<vector<Point> > contours;
vector<Vec4i> hierarchy;

void contururi()
{
    findContours( poza2, contours, hierarchy, CV_RETR_CCOMP, CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );

    Mat pozaCuContururi = Mat::zeros( poza2.size(), CV_8UC3 );
    //in pozaCuContururi desenam contururile detectate unul cate unul, fiecare cu o culoare random
    for( size_t i = 0; i< contours.size(); i++ )
    {
        Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
        drawContours( pozaCuContururi, contours, i, color, 2, 8, hierarchy, 0, Point() );
    }

    namedWindow( "Poza cu contururi", CV_WINDOW_AUTOSIZE );
    imshow( "Poza cu contururi", pozaCuContururi );
}
```

- In main apelam cu

```
contururi();
```

# Identificare si desenare

- De la ce am pornit – se obtinuse din alg. Canny
- Fiecare contur gasit este desenat cu alta culoare



# Contururi poligoane

- void approxPolyDP(InputArray **curve**, OutputArray **approxCurve**, double **epsilon**, bool **closed**)
  - **curve** –vector de puncte 2D (poate fi vector sau Mat)
  - **approxCurve** – Rezultatul aproximarii de acelasi tip cu curve.
  - **epsilon** – parametru care specifica  acuratetea de aproximare. Distanta maxima dintre curba originala si aproximare.
  - **closed** – daca este true, aproximarea se inchide (primul si ultimul nod sunt conectate).

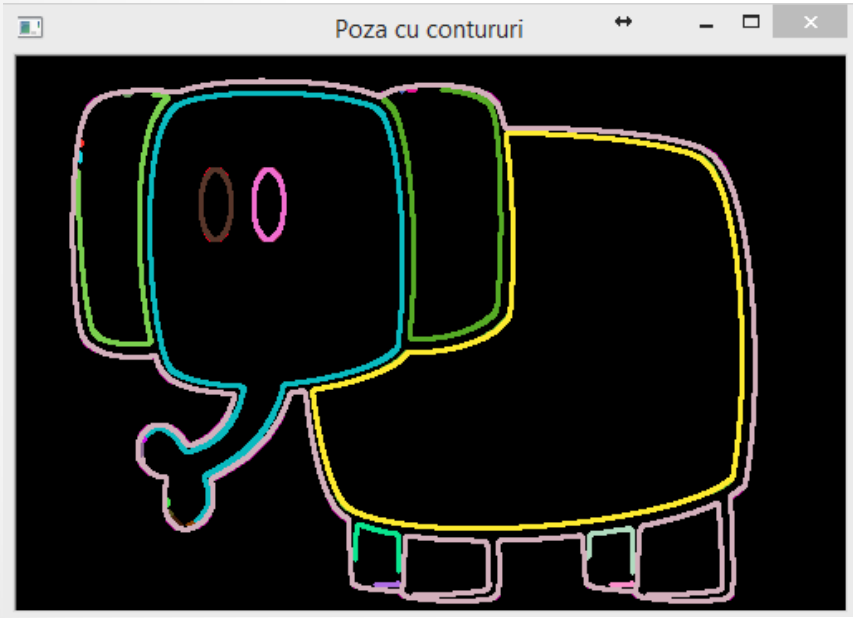
# Contururi poligoane

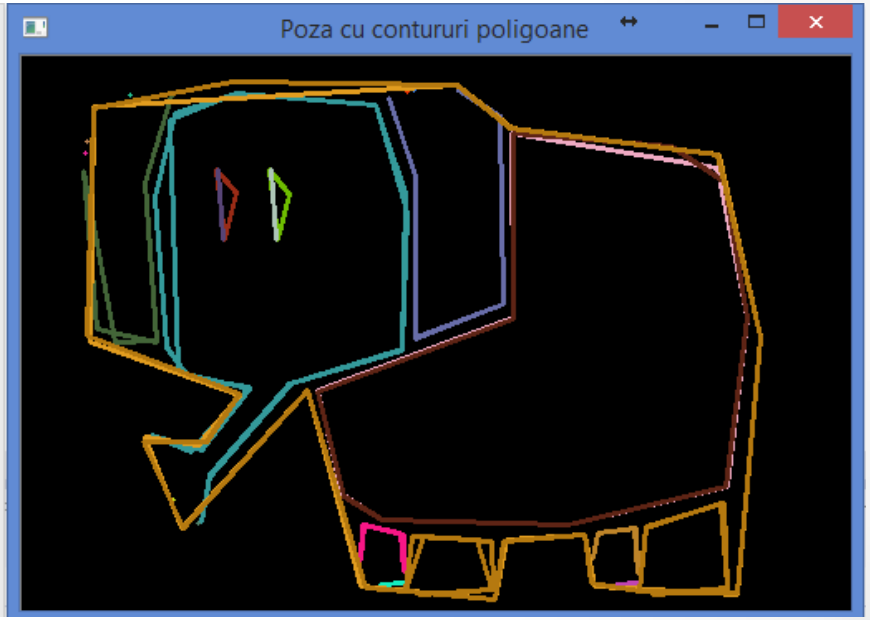
```
void gasesteContururiPoligoane()
{
    vector<vector<Point> > contours_poly( contours.size() );

    for( size_t i = 0; i < contours.size(); i++ )
        approxPolyDP( Mat(contours[i]), contours_poly[i], 10, true );

    Mat pozaCuContururiPoly = Mat::zeros( poza2.size(), CV_8UC3 );
    for( size_t i = 0; i< contours.size(); i++ )
    {
        Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
        drawContours( pozaCuContururiPoly, contours_poly, i, color, 2, 8, hierarchy, 0, Point() );
    }

    namedWindow( "Poza cu contururi poligoane", CV_WINDOW_AUTOSIZE );
    imshow( "Poza cu contururi poligoane", pozaCuContururiPoly );
}
```

- De incercat si alte valori decat 10 la approxPolyDP
- La final, in metoda contururi, adaugam apelul

`gasesteContururiPoligoane();`

# Contururi poligoane



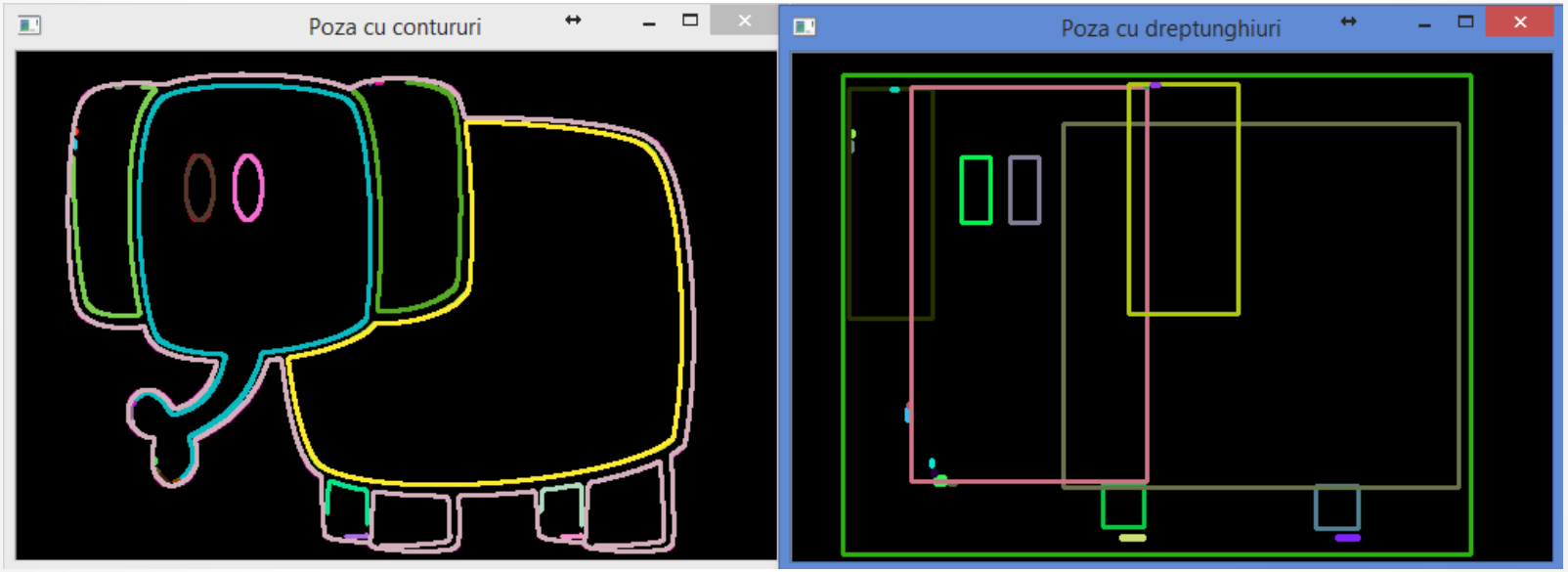
# Dreptunghiuri ce cuprind contururile

- Rect boundingRect(InputArray **points**)
  - **points** – multime de puncte 2D

```
void gasesteContururiDreptunghi()
{
    vector<Rect> boundRect( contours.size() );

    for( size_t i = 0; i < contours.size(); i++ )
        boundRect[i] = boundingRect(contours[i]);

    Mat pozaCuDreptunghiuri = Mat::zeros( poza2.size(), CV_8UC3 );
    for( size_t i = 0; i< contours.size(); i++ )
    {
        Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
        rectangle( pozaCuDreptunghiuri, boundRect[i].tl(), boundRect[i].br(), color, 2, 8, 0 );
    }

    namedWindow( "Poza cu dreptunghiuri", CV_WINDOW_AUTOSIZE );
    imshow( "Poza cu dreptunghiuri", pozaCuDreptunghiuri );
}
```

- Metoda se apeleaza in contururi(), la final.

# Dreptunghiuri ce cuprind contururile

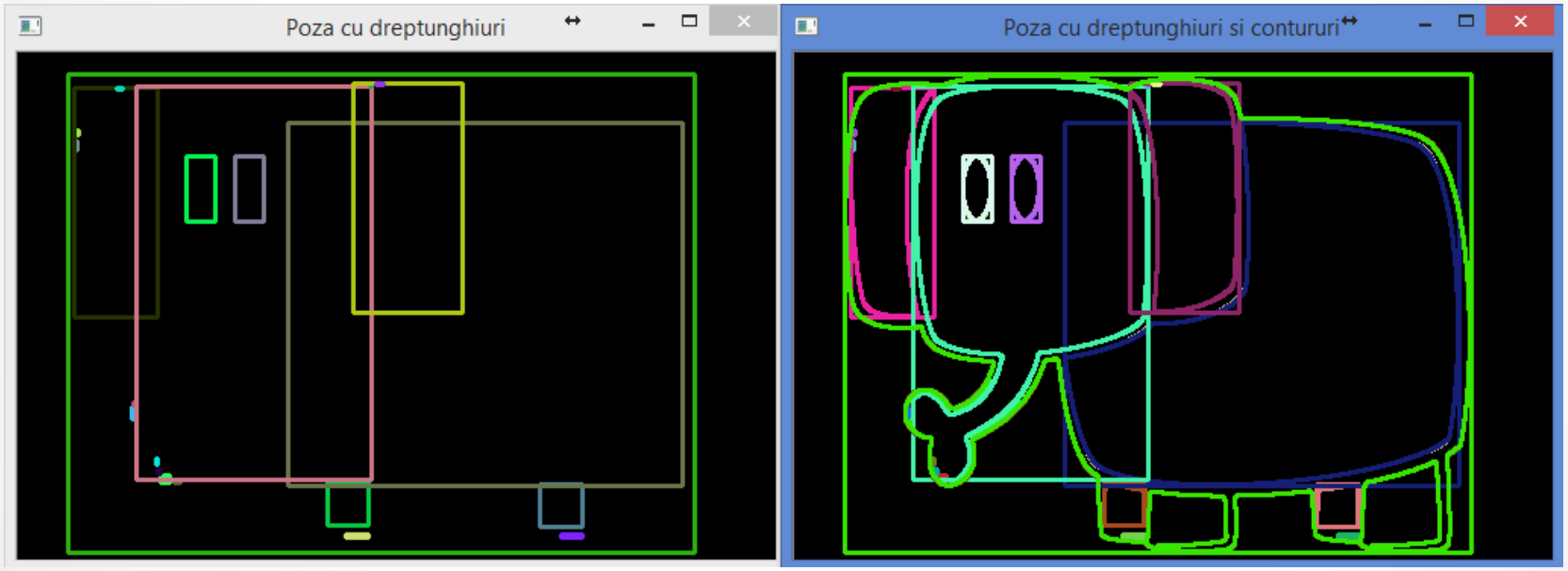


# Dreptunghiuri ce cuprind contururile

- Cuprindem in aceeasi poza si contururile, si dreptunghiurile.

```
void gasesteContururiSiDreptunghi()
{
    vector<Rect> boundRect( contours.size() );

    for( size_t i = 0; i < contours.size(); i++ )
        boundRect[i] = boundingRect(contours[i]);

    Mat pozaCuDreptunghiuriSiContururi = Mat::zeros( poza2.size(), CV_8UC3 );
    for( size_t i = 0; i< contours.size(); i++ )
    {
        Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
        //am adaugat linia de mai jos in plus fata de metoda anterioara
        drawContours( pozaCuDreptunghiuriSiContururi, contours, i, color, 2, 8, hierarchy, 0, Point() );
        rectangle( pozaCuDreptunghiuriSiContururi, boundRect[i].tl(), boundRect[i].br(), color, 2, 8, 0 );
    }

    namedWindow( "Poza cu dreptunghiuri si contururi", CV_WINDOW_AUTOSIZE );
    imshow( "Poza cu dreptunghiuri si contururi", pozaCuDreptunghiuriSiContururi );
}
```

# Dreptunghiuri ce cuprind contururile



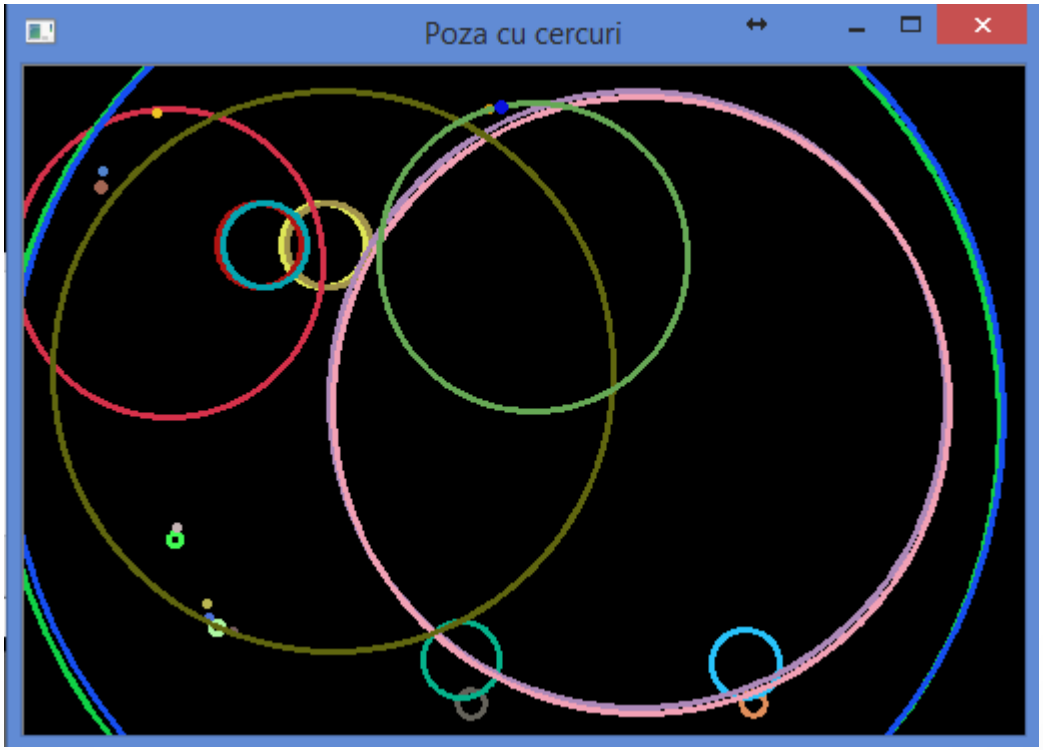
# Cercuri ce cuprind contururile

- void minEnclosingCircle(InputArray **points**, Point2f& **center**, float& **radius**)
  - **points** –vector de puncte 2D (contur)
  - **center** – intoarce centrul cercului.
  - **radius** – intoarce raza cercului.

```
void gasesteContururiCercuri()
{
    vector<Point2f>center( contours.size() );
    vector<float>radius( contours.size() );

    for( size_t i = 0; i < contours.size(); i++ )
        minEnclosingCircle( (Mat)contours[i], center[i], radius[i] );

    Mat pozaCuCercuri = Mat::zeros( poza2.size(), CV_8UC3 );
    for( size_t i = 0; i< contours.size(); i++ )
    {
        Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
        circle( pozaCuCercuri, center[i], (int)radius[i], color, 2, 8, 0 );
    }

    namedWindow( "Poza cu cercuri", CV_WINDOW_AUTOSIZE );
    imshow( "Poza cu cercuri", pozaCuCercuri );
}
```

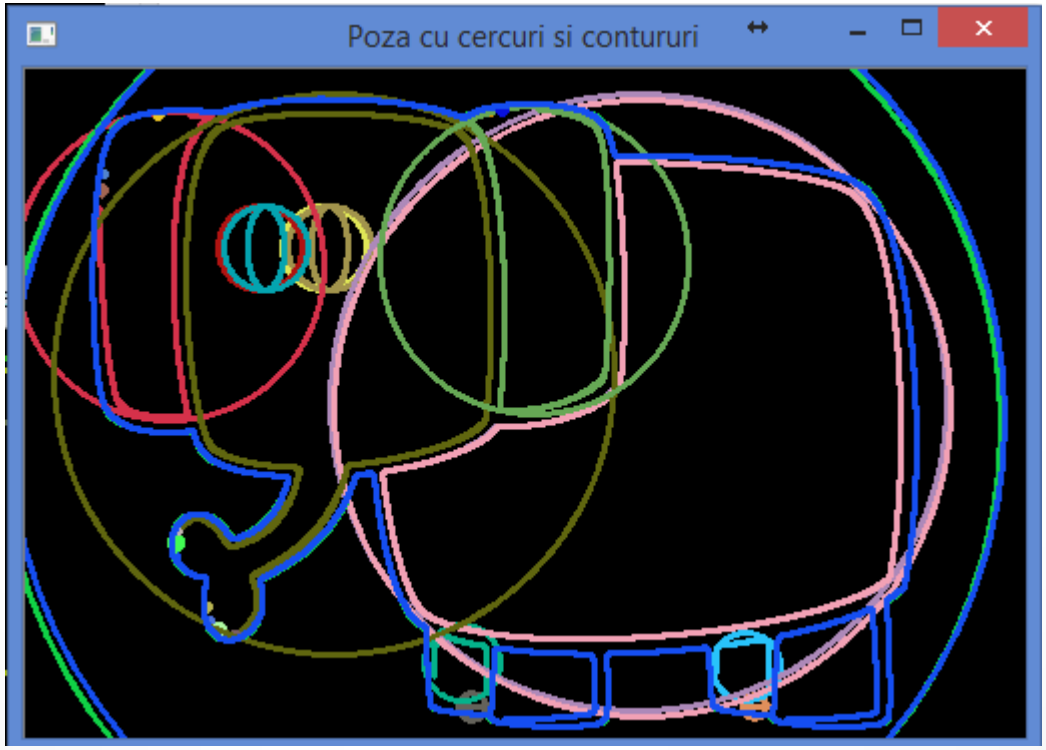
Se apeleaza tot in contururi(), la final.

# Cercuri ce cuprind contururile



# Cercuri ce cuprind contururile

- Analog, se poate obtine poza cu cercuri si contururi.



# Ierarhiile contururilor

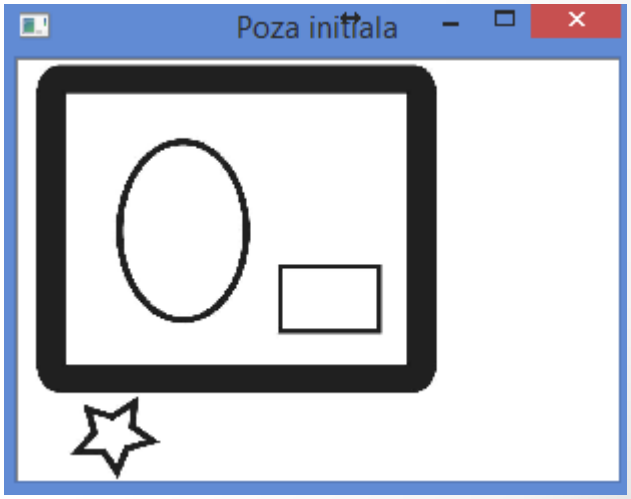
- Am observat ca algoritmul Canny intoarce linii duble pentru fiecare margine.
- findContours gaseste ulterior cate doua contururi pentru fiecare margine.
- Pentru simplitate, utilizam in continuare un simplu threshold pentru o poza noua (stanga jos).


```
//Canny( poza, poza2, 100, 200, 3 );
threshold(poza, poza2, 128, 255, CV_THRESH_BINARY_INV);
```

- Dupa ponderare, trebuie sa se obtina o poza cu fundal negru.
- Dupa caz, poate sa se aplice CV_THRESH_BINARY

# Ierarhiile contururilor

- Vom folosi si o a doua imagine (cea de mai jos) pentru a observa diferite relatii intre contururi.
- Informatiile care vor fi afisate in continuare vor fi
  - Conturul I
  - Urmatorul din acelasi nivel: hierarchy[i][0]
  - Anteriorul din acelasi nivel: hiearchy[i][1]
  - Primul descendent: hiearchy[i][2]
  - Parintele: hiearchy[i][3]]



# Ierarhiile contururilor

```
void gasesteIerarhiile()
{
    Mat pozaCuAdnotari = Mat::zeros( poza2.size(), CV_8UC3 );
    cout<<"Avem "<<contours.size()<<" contururi."<<endl;
    for( size_t i = 0; i < contours.size(); i++ )
    {
        Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
        drawContours( pozaCuAdnotari, contours, i, color, 2, 8, hierarchy, 0, Point() );
        //afisam in partea dreapta a imaginii dependentele fiecarui contur
        //textul care corespunde unui contur va avea aceeasi culoare ca si conturul
        size_t idY = i * 80, idX = poza.size().width - 200;//stabilesc x-ul (acesta ramane fixat) si y-ul
        putText(pozaCuAdnotari, "Contur " + to_string(i), Point(idX, idY + 15), FONT_HERSHEY_SIMPLEX, 0.5, color, 1, 8, false );
        putText(pozaCuAdnotari, "Urmatorul: " + to_string(hierarchy[i][0]), Point(idX, idY + 30), FONT_HERSHEY_SIMPLEX, 0.5, color, 1, 8, false );
        putText(pozaCuAdnotari, "Anteriorul: " + to_string(hierarchy[i][1]), Point(idX, idY + 45), FONT_HERSHEY_SIMPLEX, 0.5, color, 1, 8, false );
        putText(pozaCuAdnotari, "Descendent I: " + to_string(hierarchy[i][2]), Point(idX, idY + 60), FONT_HERSHEY_SIMPLEX, 0.5, color, 1, 8, false );
        putText(pozaCuAdnotari, "Parinte: " + to_string(hierarchy[i][3]), Point(idX, idY + 75), FONT_HERSHEY_SIMPLEX, 0.5, color, 1, 8, false );
    }

    namedWindow( "Poza cu adnotari", CV_WINDOW_AUTOSIZE );
    imshow( "Poza cu adnotari", pozaCuAdnotari );
}
```

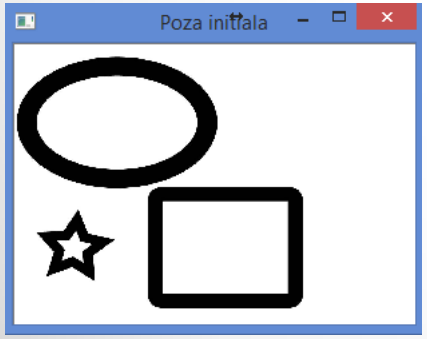
# Ierarhiile contururilor

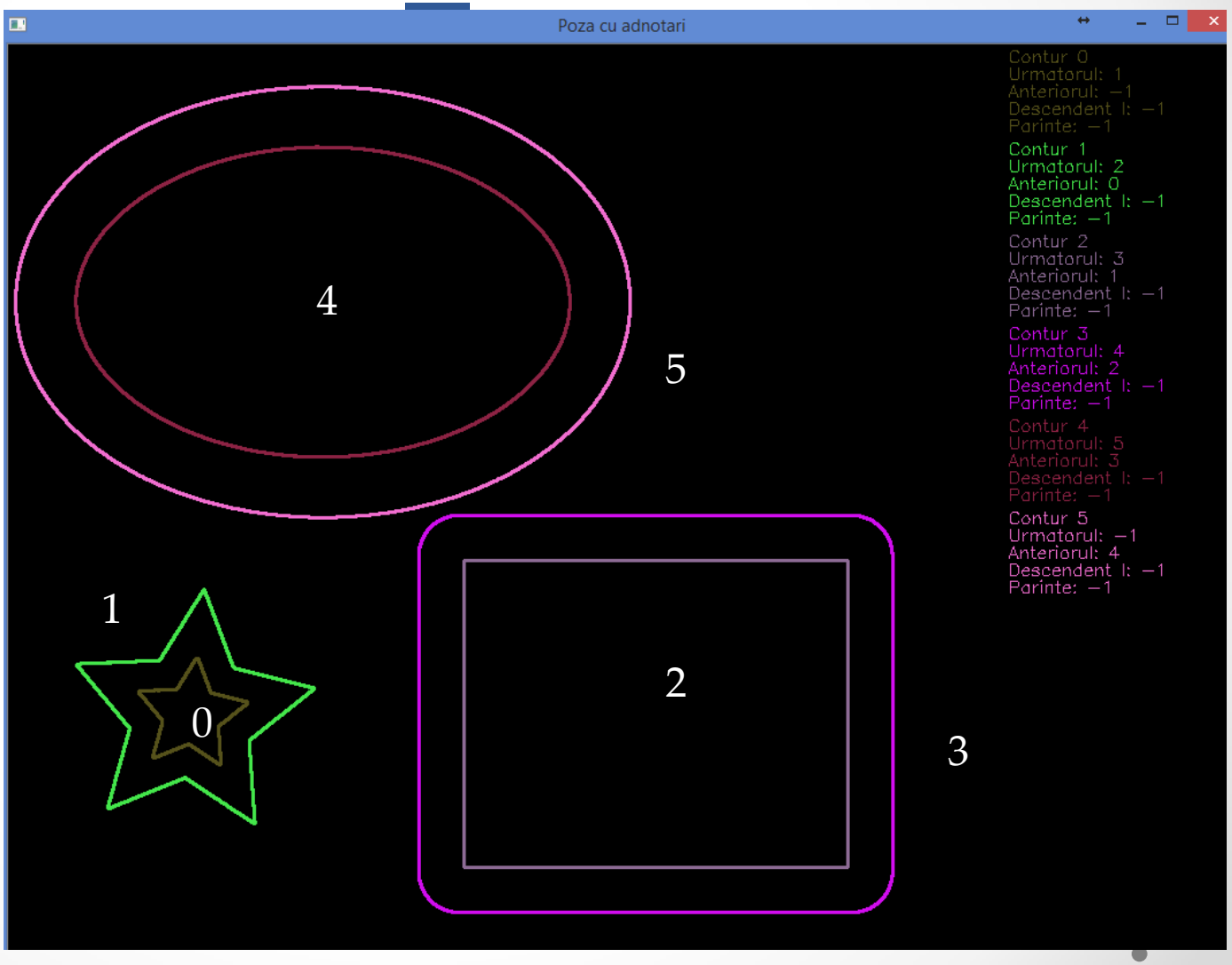
- Apelarea metodei gasestelerarhiile() se face tot din metoda contururi(), la finalul acesteia.
- In metoda contururi(), am folosit pe rand la apelarea findContours RETR_LIST, CV_RETR_EXTERNAL, RETR_CCOMP, CV_RETR_TREE

```
void contururi()
{
    findContours( poza2, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE, Point(0, 0) );
    //RETR_LIST
    //CV_RETR_EXTERNAL
    //RETR_CCOMP
    //CV_RETR_TREE
    cout<<"Avem "<<contours.size()<<" contururi."<<endl;
    Mat pozaCuContururi = Mat::zeros( poza2.size(), CV_8UC3 );
    //in pozaCuContururi desenam contururile detectate unul cate unul, fiecare cu o culoare random
    for( size_t i = 0; i < contours.size(); i++ )
    {
        Scalar color = Scalar( rng.uniform(0, 255), rng.uniform(0,255), rng.uniform(0,255) );
        drawContours( pozaCuContururi, contours, i, color, 2, 8, hierarchy, 0, Point() );
    }

    namedWindow( "Poza cu contururi", CV_WINDOW_AUTOSIZE );
    imshow( "Poza cu contururi", pozaCuContururi );

    gasesteContururiPoligoane();

    gasesteContururiDreptunghi();

    gasesteContururiSiDreptunghi();

    gasesteContururiCercuri();

    gasesteIerarhiile();
}
```

# Ierarhiile contururilor RETR_LIST

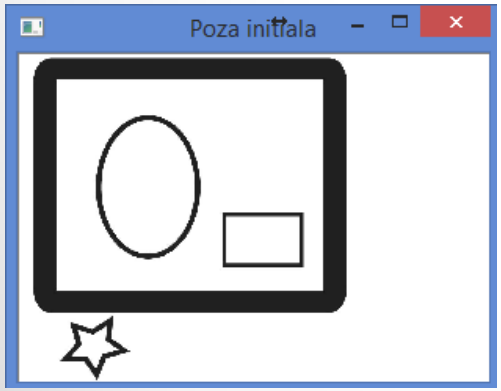
- Cel mai simplu
- Nu are nici descendent, nici parinte

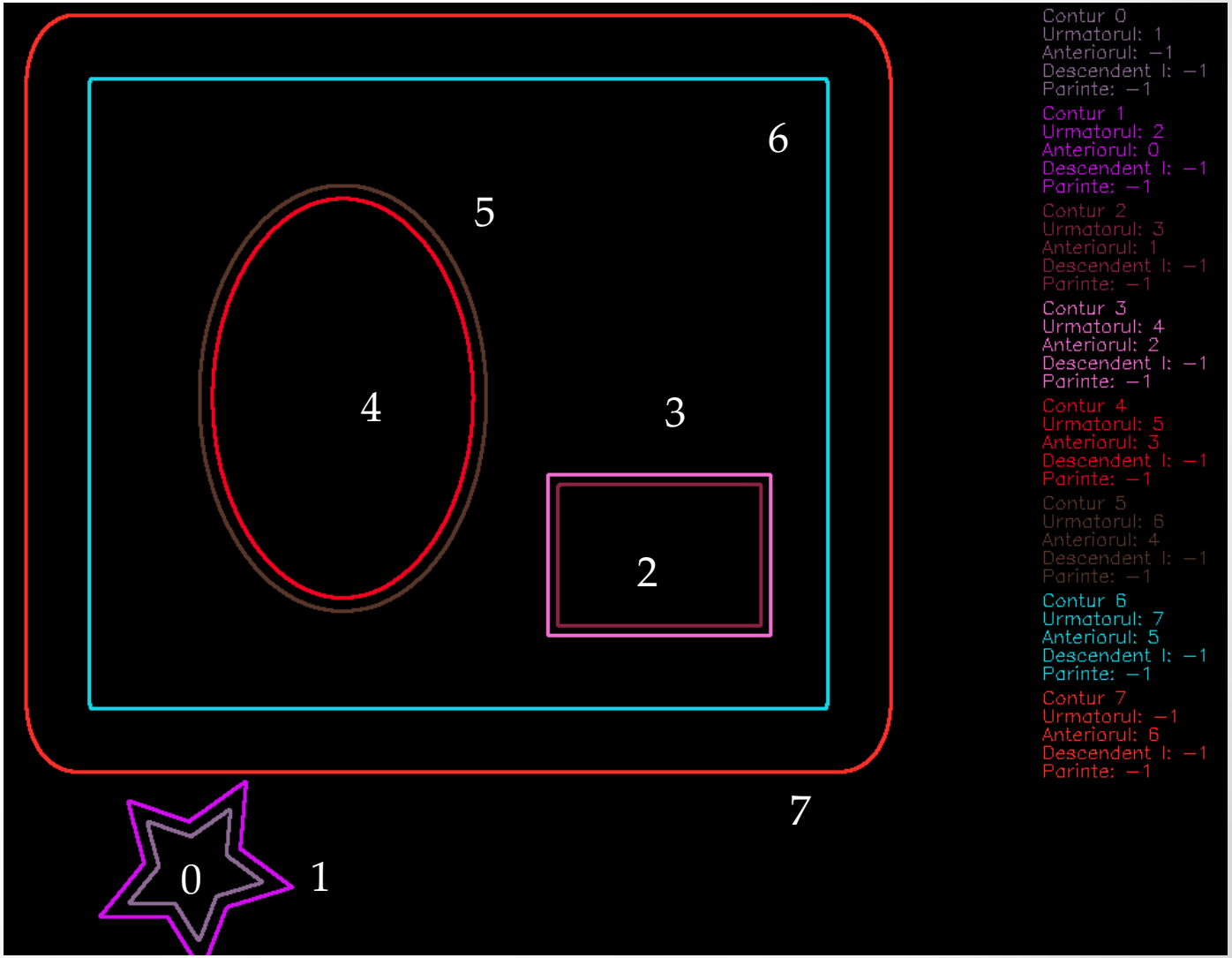


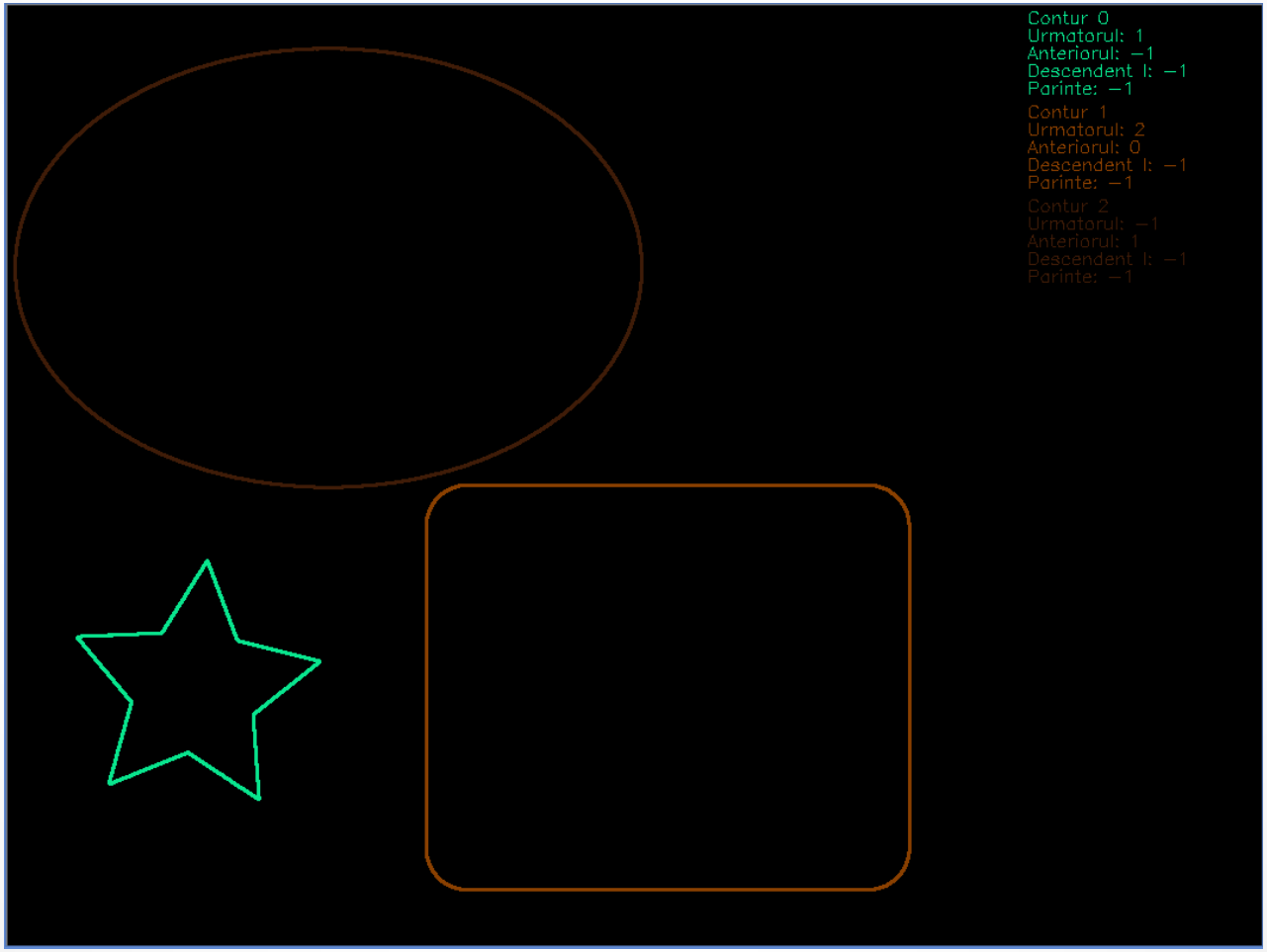
# Ierarhiile contururilor RETR_LIST

- Poza a doua pe care facem calculele.



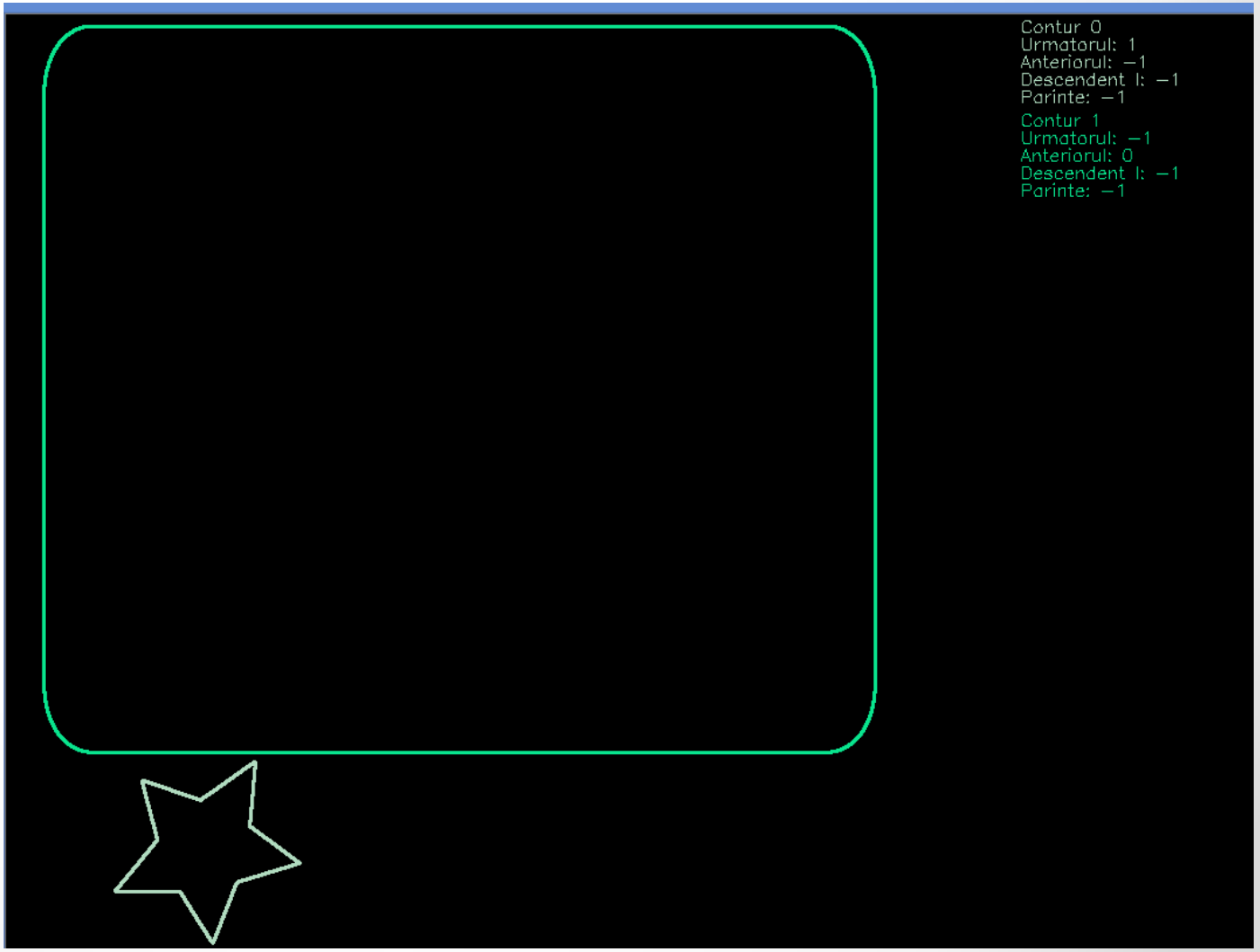



# RETR_EXTERNAL



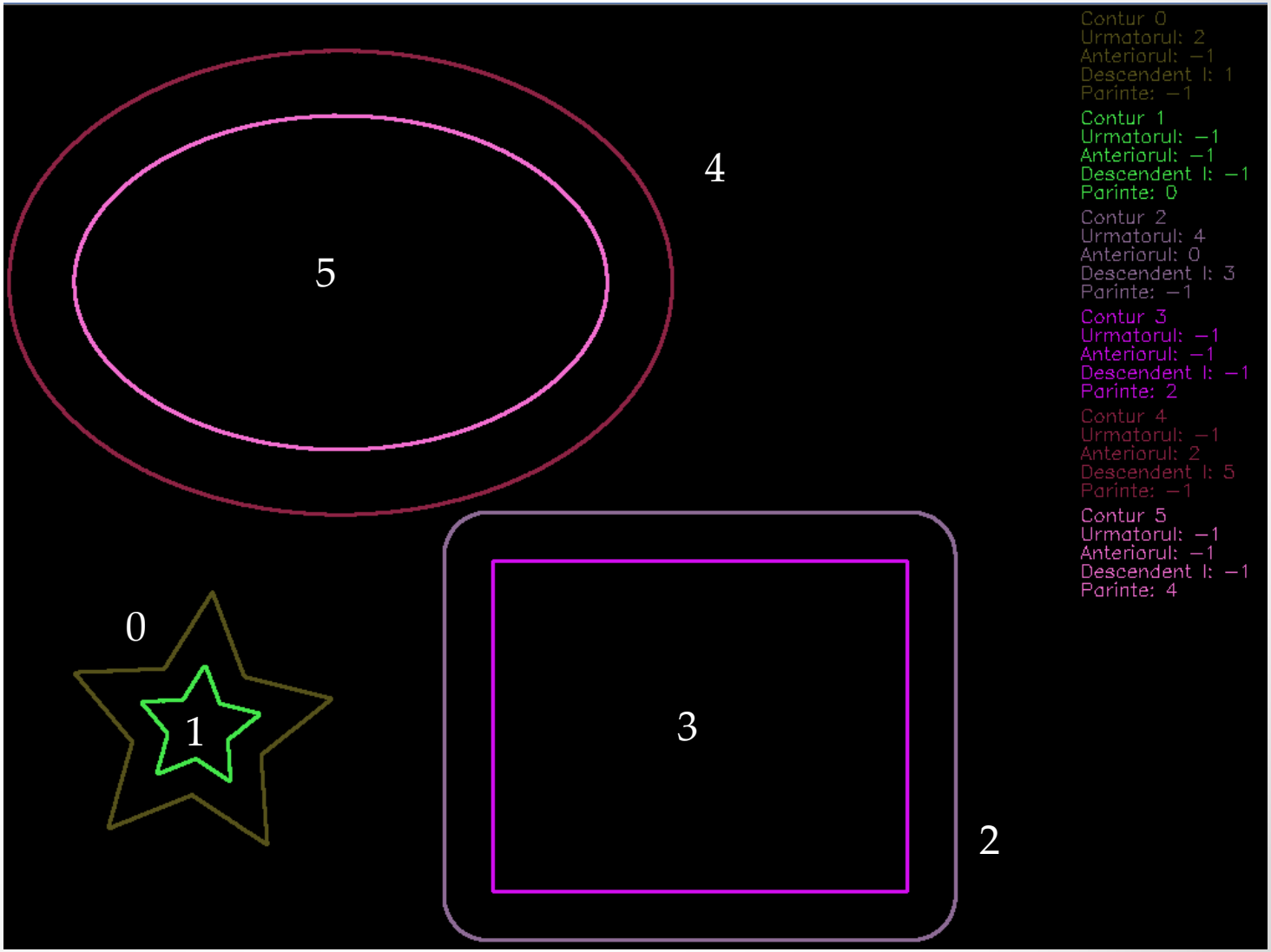
# RETR_EXTERNAL

- Se afiseaza numai contururile exterioare.

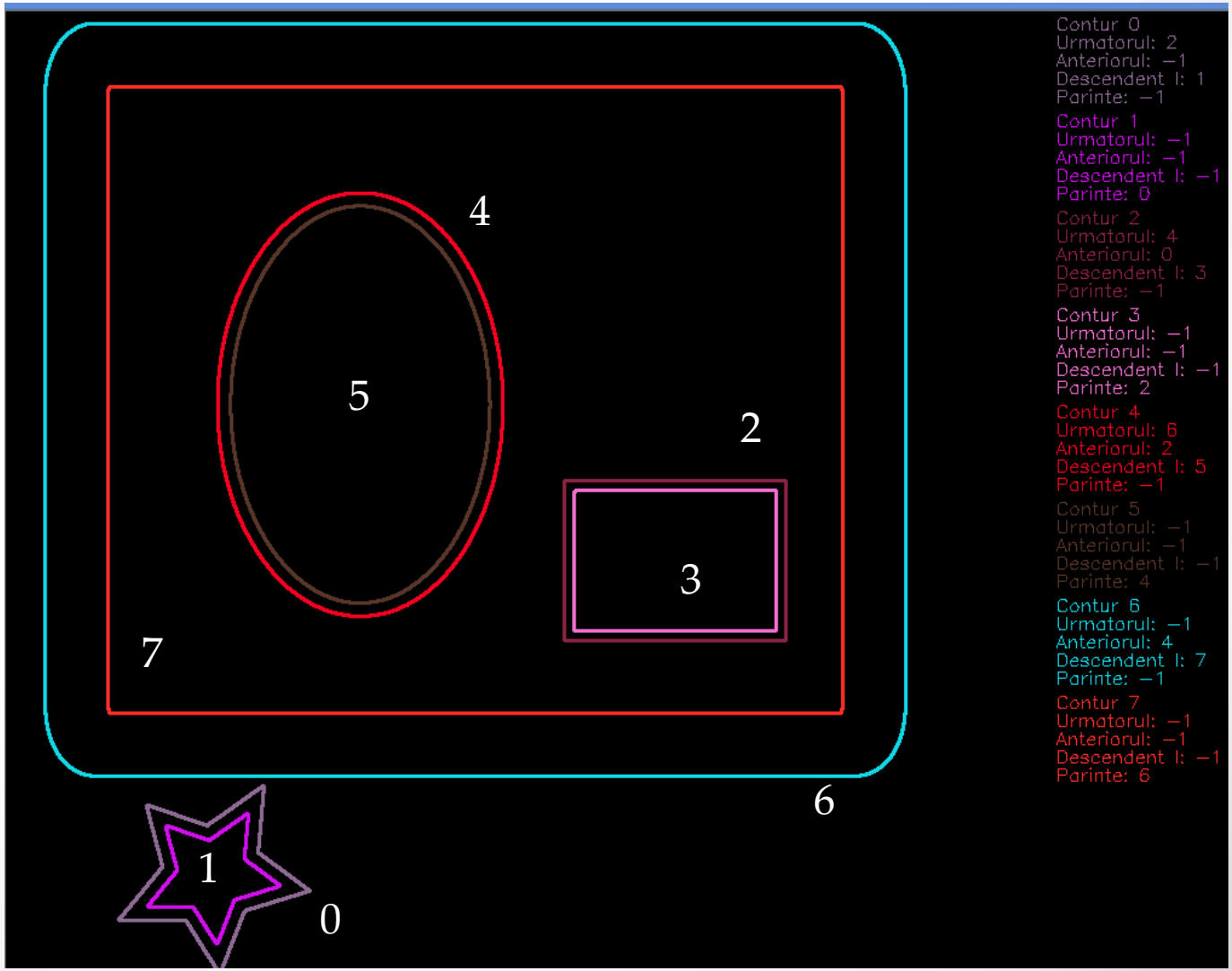


# RETR_CCOMP

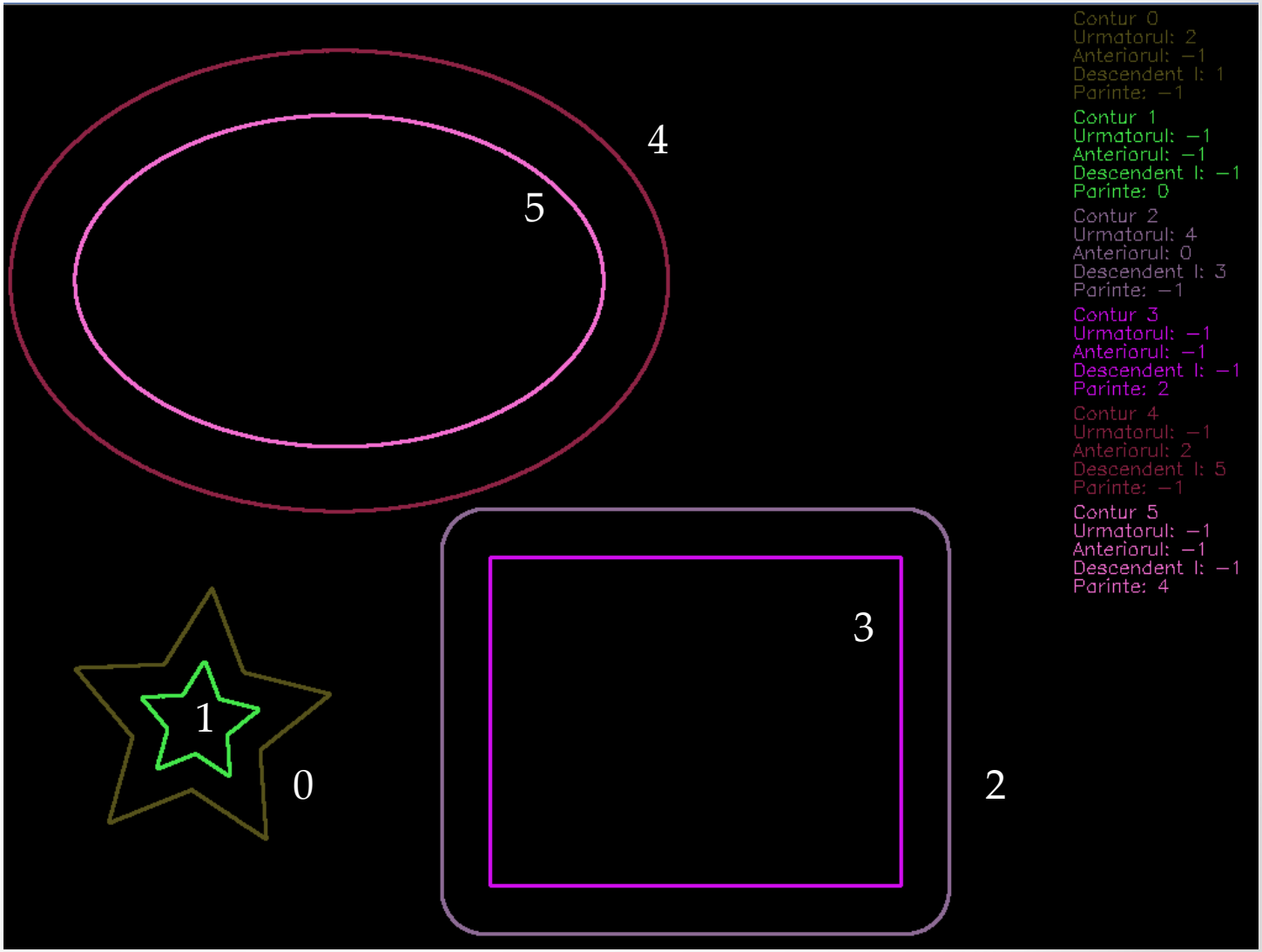
- Sunt doar 2 niveluri de ierarhizare.



# RETR_CCOMP

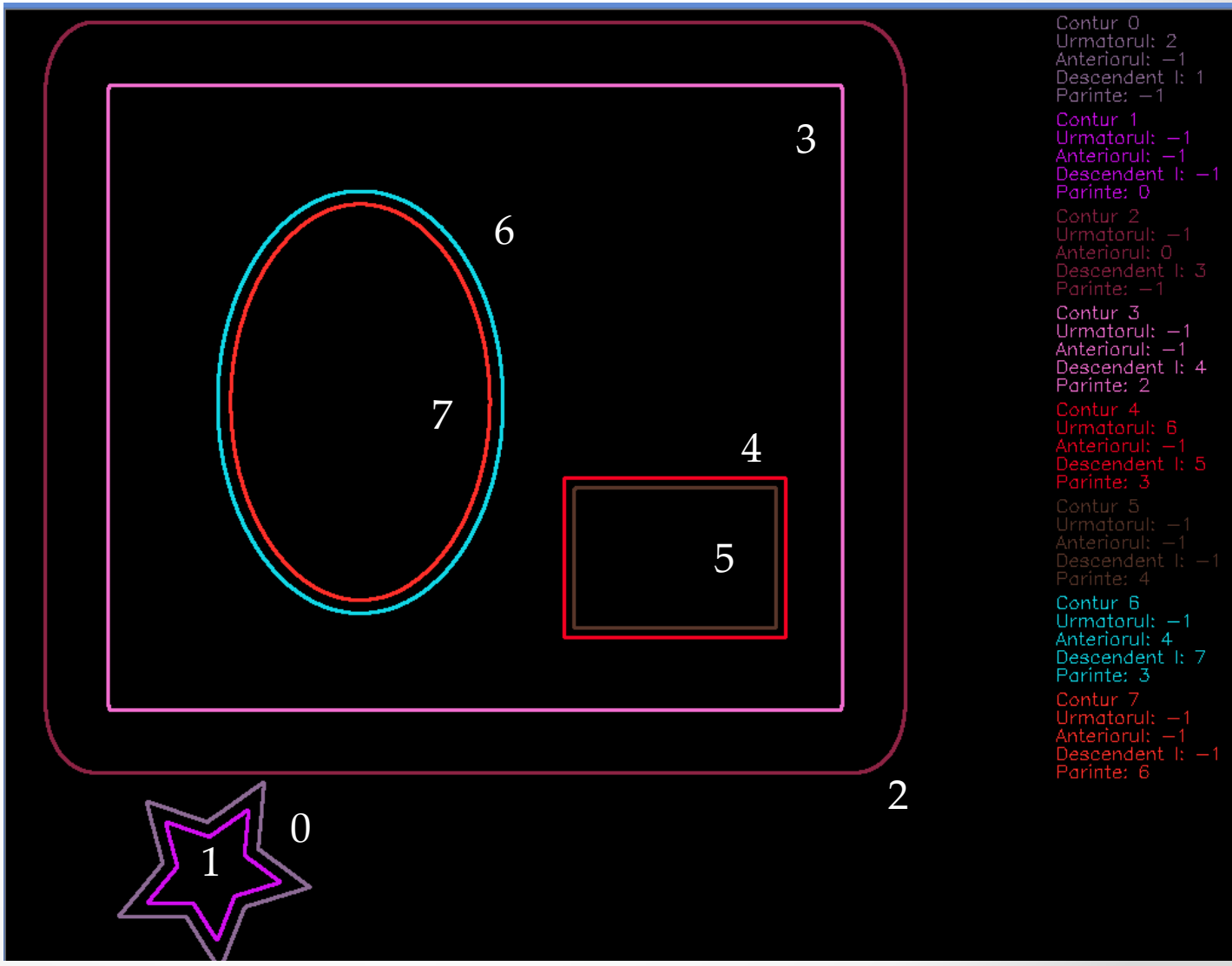

Contur 0
Urmatorul: 2
Anteriorul: -1
Descendent I: 1
Parinte: -1
Contur 1
Urmatorul: -1
Anteriorul: -1
Descendent I: -1
Parinte: 0
Contur 2
Urmatorul: 4
Anteriorul: 0
Descendent I: 3
Parinte: -1
Contur 3
Urmatorul: -1
Anteriorul: -1
Descendent I: -1
Parinte: 2
Contur 4
Urmatorul: 6
Anteriorul: 2
Descendent I: 5
Parinte: -1
Contur 5
Urmatorul: -1
Anteriorul: -1
Descendent I: -1
Parinte: 4
Contur 6
Urmatorul: -1
Anteriorul: 4
Descendent I: 7
Parinte: -1
Contur 7
Urmatorul: -1
Anteriorul: -1
Descendent I: -1
Parinte: 6
4
5
2
3
7
6
1
0

# CV_RETR_TREE


Contur 0
Urmatorul: 2
Anteriorul: −1
Descendent I: 1
Parinte: −1
Contur 1
Urmatorul: −1
Anteriorul: −1
Descendent I: −1
Parinte: 0
Contur 2
Urmatorul: 4
Anteriorul: 0
Descendent I: 3
Parinte: −1
Contur 3
Urmatorul: −1
Anteriorul: −1
Descendent I: −1
Parinte: 2
Contur 4
Urmatorul: −1
Anteriorul: 2
Descendent I: 5
Parinte: −1
Contur 5
Urmatorul: −1
Anteriorul: −1
Descendent I: −1
Parinte: 4
4
5
3
1
0
2

# CV_RETR_TREE

- Acest tip de ierarhizare este cel mai complex, fiindca toate conexiunile sunt reprezentate.



# Compararea a doua forme

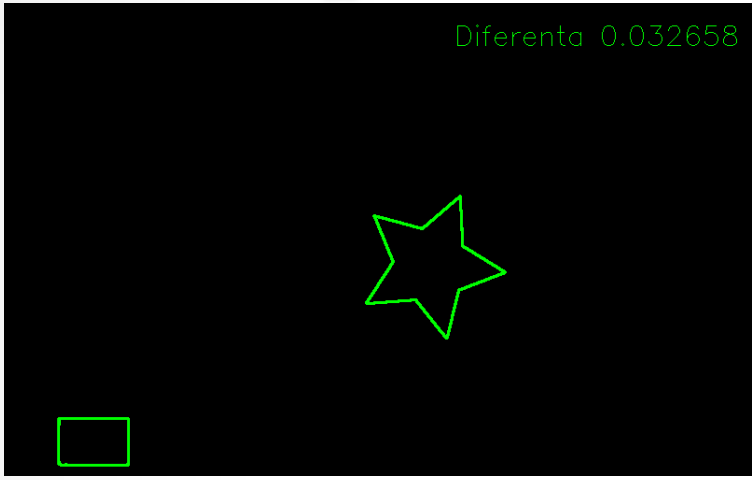
- double matchShapes(InputArray **contour1**, InputArray **contour2**, int **method**, double **parameter**)
  - **object1** – Primul contur sau o imagine grayscale.
  - **object2** – Al doilea contur sau o imagine grayscale.
  - **method** – metida de comparare
    - CV_CONTOURS_MATCH_I1
    - CV_CONTOURS_MATCH_I2
    - CV_CONTOURS_MATCH_I3
      - Pentru detalii legate de ce reprezinta metodele vizitati http://docs.opencv.org/modules/imgproc/doc/structural_analysis_and_shape_descriptors.html?highlight=cvmatchshapes#matchshapes
  - **parameter** – parametru specific metodei, dar deocamdata nu se poate utiliza. Orice valoare ofera acelasi rezultat.

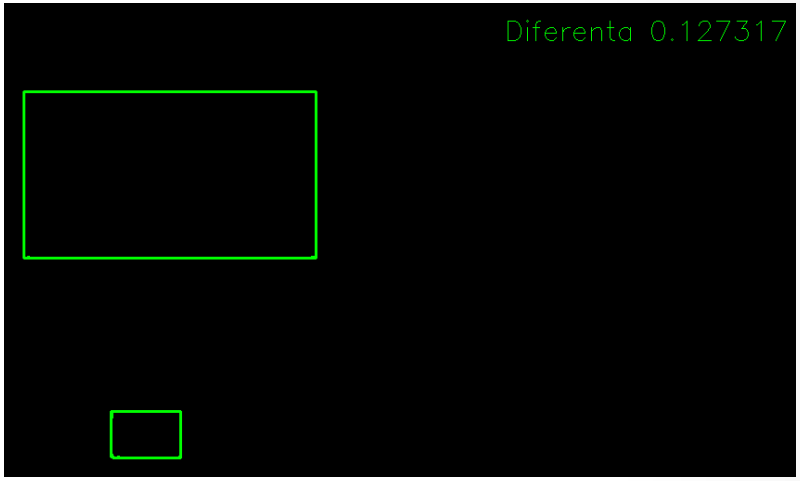
# Compararea a doua forme

- Pentru metoda de mai jos se trimit indicii contururilor de comparat.
- Diferenta este scrisa direct pe imagine.
- Apelarea se face tot din contururi(), deci dupa ce s-au calculat contururile.

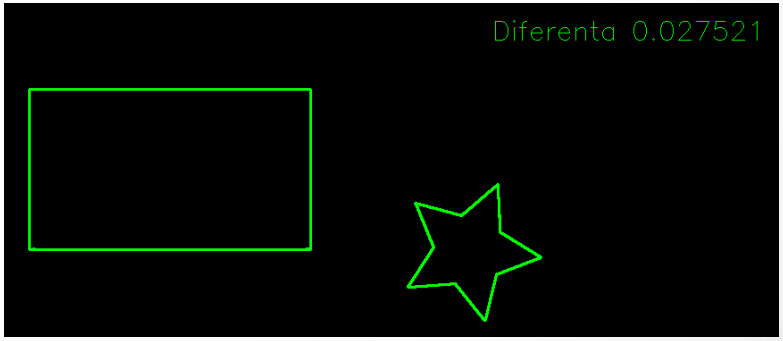
```
void comparContururi(int id1, int id2)
{
    Mat pozaCuAdnotari = Mat::zeros( poza2.size(), CV_8UC3 );
    Scalar color = Scalar( 0, 255, 0);//verde
    drawContours( pozaCuAdnotari, contours, id1, color, 2, 8, hierarchy, 0, Point() );
    drawContours( pozaCuAdnotari, contours, id2, color, 2, 8, hierarchy, 0, Point() );
    putText(pozaCuAdnotari, "Diferenta " + to_string(matchShapes(contours[id1], contours[id2], CV_CONTOURS_MATCH_I1, 1)),
        Point(poza.size().width - 500, 50), FONT_HERSHEY_SIMPLEX, 1, color, 1, 8, false );
    namedWindow( "Poza cu adnotari " + to_string(id1) + " vs. " + to_string(id2), CV_WINDOW_AUTOSIZE );
    imshow( "Poza cu adnotari " + to_string(id1) + " vs. " + to_string(id2), pozaCuAdnotari );
}
```

# Compararea a doua forme




- Cand contururile sunt identice, rezultatul este 0.



# Apartenenta punct la poligon

- double pointPolygonTest(InputArray **contour**, Point2f **pt**, bool **measureDist**)
  - **contour** – conturul de intrare.
  - **pt** – punctul care este testat fata de contur.
  - **measureDist** –
    - Daca este true, functia intoarce distanta cu semn de la punct la cea mai apropiata muchie.
    - Altfel, se testeaza daca punctl este in interiorul conturului sau nu.
      - 1 = interior
      - 0 = pe muchie
      - -1 = exterior
- Poate fi folosita pentru a testa daca centrul cercului care circumscrie un poligon se afla in interiorul altui poligon.

# Proiecte 1/2

1. Faceti un proiect cu GUI care sa permita modificarea celor doi parametri din algoritmul Canny pentru o imagine citita. Imaginea rezultata trebuie sa se poata salva. O varianta mai simpla consta din primirea unui singur parametru, iar al doilea sa se obtina prin dublarea primului.
2. Faceti un proiect cu GUI care sa permita pentru o imagine citita sa se afiseze contururile, poligoanele si/sau cercurile. Se folosesc check box-uri.

# Proiecte 2/2

3. Scrieti un program care sa identifice contururile dintr-o poza si sa verifice daca centrele cercurilor care circumscriu contururile se gasesc in interiorul altor contururi. Contururile sunt verificate 2 cate 2.
4. Gasiti pentru frame-urile unui clip video contururile si afisati-le numai pe cele exterioare.
5. Pentru o imagine de intrare sa afisati numai contururile din interior (care nu mai contin alte contururi).