# Képfeldolgozás a gyakorlatban

## Körvonalak kezelése

Szeghalmy Szilvia

# Külső körvonal kinyerése

```
void findContours(InputArray img, OutputArrayOfArrays contours,
                                                    int mode, int method)
```

- img: szürkeskálás, de binárisnak tekintett (0 vs. nem nulla)*
- contours:

```
vector<vector<Point>> contours;
```

- mode:
  - `RETR_EXTERNAL`: a legkülső kontúrokat adja vissza
- method pl:
  - `CHAIN_APPROX_NONE`: minden pontot eltárol
  - `CHAIN_APPROX_SIMPLE`: a lánckód tömörített tárolása, ahol lehet csak a csúcsok kerülnek tárolásra

**Megj.: ha régebbi OpenCV-t használ és InputOutpuArray-t lát típusként, akkor a feldolgozás során megváltozhat a kép.*

# Kontúr rajzolása

```
void drawContours(
    InputOutputArray img,
    InputArrayOfArrays contours,
    int contourIdx,
    const Scalar& color, int thickness=1);
```

`img`: Léteznie kell, férjen ki rá a kontúr!
(az esetlegesen megadott offset-et figyelembe véve)

`contourIdx`: kirajzolandó kontúr indexe a vektoron belül
ha ez -1, akkor minden kontúr kirajzolásra kerül

`thickness`: a kontúr vastagsága
ha -1 (cv::FILLED), akkor telítve lesz kirajzolva

# Feladat: külső kontúr megkeresése

- Olvassa be színes képként valamelyik sajtos képet.
- Konvertálja szürkeskálás képpé (a színes képet őrizze meg).
- Küszöbölje a szürkeskálás képet.
- A findContours függvénnyel nyerje ki a külső körvonalakat.
- Rajzolja ki a színes képre az eredményt, majd jelenítse meg.

- A konzolra írja ki, hogy hány kontúrt talált. Ha nem 1 az érték, akkor végezzen mediánszűrést a küszöbölés után.

Mentse, hamarosan dolgozunk még vele.

# Feladat: külső kontúr megkeresése

- Olvassa fel a dog.jpg képet színesben.
- Alakítsa szürkeskálássá.
- Küszöbölje a képet.

  Ne végezzen semmilyen szűrést.
- Gyűjtse össze a külső kontúrokat. Használjon  CHAIN_APPROX_NONE-t.
- Járja be a kontúrokat és azok mérete alapján döntsön arról, hogy a kutya kontúrjáról vagy zajról van-e szó.

Pl.:

| | |
|---|---|
| pontok száma: | contours[i].size() |
| körvonal hossza: | arcLength(contours[i], True)  //:True - zárt kontúr |
| kontúr területe: | contourArea(contours[i]) |

# Feladat: külső kontúr - tárolási módok

- Olvassa be színesben az *objektumok.png* képet.
- Konvertálja át szürkeskálássá.
- Nyerje ki a külső körvonalakat. Használjon  CHAIN_APPROX_NONE-t.
- Rajzolja rá az eredeti (színes) képre a körvonalakat.
- Járja be a kontúrvektort:
  írja ki a képre az aktuális körvonal méretét:

```
putText(Mat& img, string text, Point point,
            int font, double scale, Scalar color);
```

  - a pont legyen a kontúr 0. pontja
  - a font legyen pl. FONT_HERSHEY_PLAIN
  - a scale legyen 1.0

Módosítsa a tárolást CHAIN_APPROX_SIMPLE-re és figyelje meg a változást.

# Feladat: külső kontúr – Poligon közelítés

- A CHAIN_APPROX_SIMPLE tömöríti a kontúrt, a lánckódra alapozva.
- Alkalmazzunk inkább poligon approximációt, és figyeljük meg a változást.

- Módosítsa a kontúrok bejárására szolgáló ciklust:
  - hozzon létre egy vektort a közelítő poligon csúcsainak tárolására
  - közelítse a kontúrt (approxPolyDP)
  - rajzolja ki a kapott poligont (polylines)
  - A kontúr mérete helyett a poligon méretét jelenítse meg a képen

```
approxPolyDP(vector<Point> contour, vector<Point> approx_contour,
                double eps, bool is_closed);

polylines(Mat& canvas, vector<Point> contour,
                 bool is_closed, Scalar color, int line_width);
```

# Feladat: Sas kivágása és mentése

- Folytassuk a sas.avi videó feldolgozását
- A cél kivágni minden képkockáról a sast tartalmazó képrészt és elmenteni új képként.
- Miután az ég maszkját meghatározta:
  - Invertálja a maszkot (eg maszk -> sas maszk )
  - Dilatálja a maszkot pl. MORPH_ELLIPSE alakú, legalább 15x15-ös struktúraelemmel
  - Keresse meg a külső kontúrokat: RETR_EXTERNAL, CHAIN_APPROX_SIMPLE
  - Járja be a kontúrokat* :
    - Ha nagyobb a területe, mint 1000 pixel,

```
double contourArea(InputArray points);
```

    - akkor határozza meg a kontúr befoglaló téglalapját:

```
Rect boundingRect(InputArray points)
```

    Vágja ki a képről:

```
Mat dest = src( Rect rect );
```

    Mentse el:

```
void imwrite(string filename, InputArray, img);
```

- Ha egyetlen kontúr marad a képen, akkor mellőzhető a bejárás. A 0. kontúr lesz a sas.

# Összes kontúr kinyerése

```
void findContours(InputArray img, OutputArrayOfArrays contours,
                  int mode, int method)
```

- img: szürkeskálás, de binárisnak tekintett (0 vs. nem nulla)

  contours:

  ```
  vector<vector<Point>> contours;
  ```

- mode:
  - `RETR_LIST`: minden (objektum)kontúrt visszaad egy listában, nincs köztük kapcsolat

- method pl:
  - `CHAIN_APPROX_NONE`: minden pontot eltárol
  - `CHAIN_APPROX_SIMPLE`: csak a szükséges pontokat tárolja
  - pl: egy téglalapnál csak a 4 csúcspontot

# Feladat: Összes kontúr kinyerése

- Alakítsa át a korábbi sajtkörvonal kinyerő programot úgy, hogy a belső kontúrokat is rajzolja ki a képre.

  - Olvassa be színes képként valamelyik sajtos képet.
  - Konvertálja szürkeskálás képpé (a színes képet őrizze meg).
  - Küszöbölje a szürkeskálás képet.
  - A findContours függvénnyel nyerje ki az **összes** körvonalat listaként (RETR_LIST).
  - Rajzolja ki a színes képre az eredményt, majd jelenítse meg.

# Összes kontúr kinyerése: hierarchia

```
void findContours(InputArray img, OutputArrayOfArrays contours,
        OutputArray hierarchy, int mode, int method)
```

- img: szürkeskálás, de binárisnak tekintett (0 vs. nem nulla)
  (ha régebbi OpenCV-t használ és InputOutpuArray-t lát az img előtt,
  akkor megváltozhat a feldolgozás során a kép)
- contours:
  `vector<vector<Point>> contours;`
- mode:
  - `RETR_CCOMP`: külső körvonal és egy lyuk képezhet egy komponenst,
    ezek listáját adja vissza (`hierarchy` paraméter is kell)
  - `RETR_TREE`: egy lyukban is lehet újabb objektum azon belül újabb lyuk...
    (`hierarchy` paraméter is kell)
- method pl:
  - `CHAIN_APPROX_NONE`: minden pontot eltárol
  - `CHAIN_APPROX_SIMPLE`: csak a szükséges pontokat tárolja
  - pl: egy téglalapnál csak a 4 csúcspontot

# Kontúr rajzolása

```
void drawContours(
    InputOutputArray img,
    InputArrayOfArrays contours,
    int contourIdx,
    const Scalar& color, int thickness=1, int lineType=8,
    InputArray hierarchy=noArray(),
    int maxLevel=INT_MAX, Point offset=Point() );
```

`img`: Léteznie kell, férjen ki rá a kontúr!
(az esetlegesen megadott offset-et figyelembe véve)

`contourIdx`: kirajzolandó kontúr indexe a vektoron belül
ha ez -1, akkor minden kontúr kirajzolásra kerül

`thickness`: a kontúr vastagsága
ha -1 (cv::FILLED), akkor telítve lesz kirajzolva

`maxLevel`: ha van hierarchia, akkor 0 – adott kontúrt rajzolása,
1 – kontúr és a közvetlen gyerek kontúr(ok) rajzolása
2 – kontúr és a fában alatta lévő összes kontúr rajzolása

# Kontúr hierarchia

```
vector<vector<Point>> contours;
vector<Vec4i> hierarchy;
findContours(img, contours, hierarchy, mode, method);
```

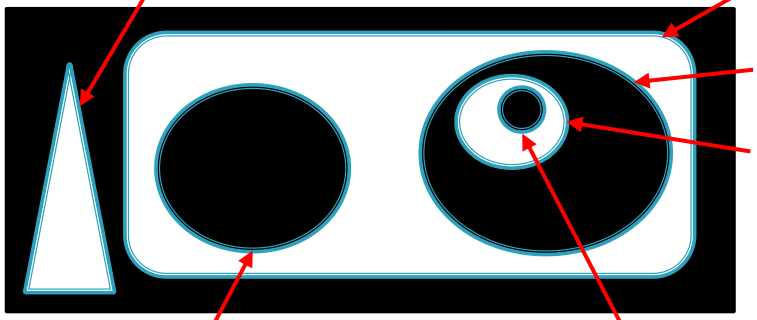
Az i. kontúrhoz `(contour[i])`, az i. hierarchia tartozik `(hierarchy[i])`

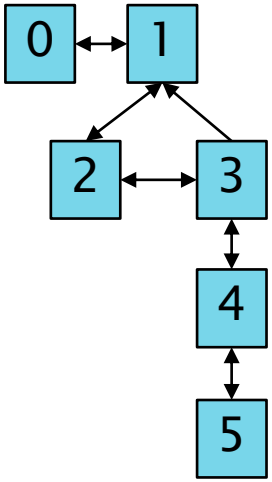
- `hierarchy[i][0]` a következő kontúr indexe (azonos szinten)
- `hierarchy[i][1]` az előző kontúr indexe (azonos szinten)
- `hierarchy[i][2]` gyerek elem indexe
- `hierarchy[i][3]` szülő elem indexe

- ha valamelyik nem létezik, akkor -1 az érték

  pl.: ha az *i.* objektum lyukat tartalmaz, akkor a hierarchy[i][2] != -1

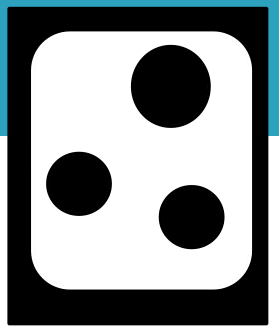
# Kontúr hierarchia (RETR_TREE)




| i | hier[i] [0] (next) | hier[i] [1] (previous) | hier[i] [2] (first child) | hier[i] [3] (parent) |
|---|---|---|---|---|
| 0 | 1 | –1 (nincs előző) | –1 (nincs gyerek) | –1 (nincs szülő) |
| 1 | –1(nincs köv.) | 0 | 2 | –1 (nincs szülő) |
| 2 | 3 | –1 (nincs előző) | –1 (nincs gyerek) | 1 |
| 3 | –1(nincs köv.) | 2 | 4 | 1 |
| 4 | –1(nincs köv.) | –1 (nincs előző) | 5 | 3 |
| 5 | –1(nincs köv.) | –1 (nincs előző) | –1 (nincs gyerek) | 4 |

# Feladat: Mennyi lyukat mértek a sajtba?



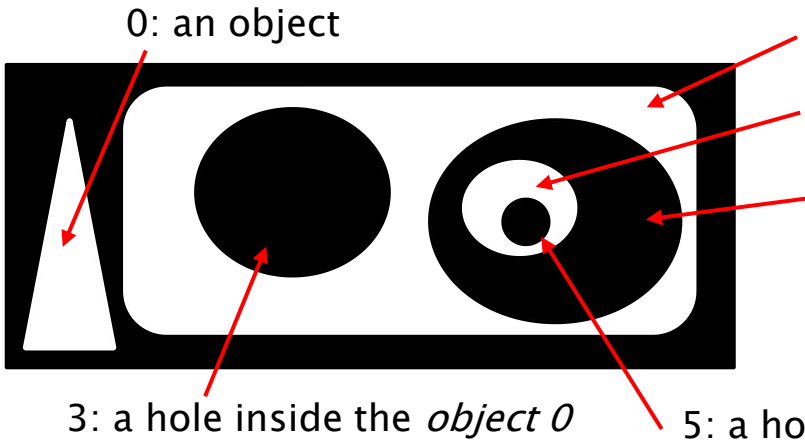
- Olvassa be valamelyik sajtos képet.
- Alakítsa át szürkeskálás képpé
- Küszöbölés segítségével válassza le a fehér hátteret a sajtról
- Kérje le az objektum kontúrokat:

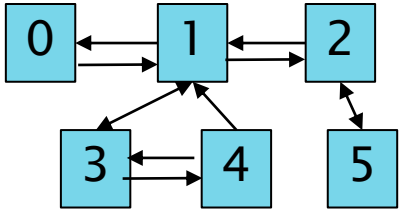
```
vector<vector<Point>> cont;
vector<Vec4i> hier;
findContours(kuszobolt_kep, cont, hier, RETR_TREE, CHAIN_APPROX_NONE);
```

- Határozza meg a sajtszelet területét a lyukak nélkül.
- Határozza meg hány százalék a lyuk a sajtszeletben.
- Ellenőrzésként rajzold is ki.
  - Járja be a kontúrhierarchiát tartalmazó vektort
    - Ha az i. kontúrnak nincs szülőeleme (hier[i][3] == -1), akkor az a sajtlap (esetleg zaj)
    - Különben üreget talált a sajtlapban (esetleg zajt)
  - A kontúr területének lekérése:

```
double contourArea(InputArray contour);
```

# Kontúr hierarchia (RETR_CCOMP)




| contours idx | hier[idx] [0] (next) | hier[idx] [1] (previous) | hier[idx] [2] (first child) | hier[idx] [3] (parent) |
|---|---|---|---|---|
| 0 | 1 | -1 (no previous) | -1 (no child) | -1 (no parent) |
| 1 | 2 | 0 | 3 | -1 (no parent) |
| 2 | -1(no next) | 1 | 5 | -1 (no parent) |
| 3 | 4 | -1(no previous) | -1 (no child) | 1 |
| 4 | -1(no next) | 3 | -1 (no child) | 1 |
| 5 | -1(no next) | -1 (no previous) | -1 (no child) | 2 |

# Feladat: Alátétek

- Olvassa be a gumialatetek3.jpg képet színesben.
- Alakítsa át szürkeskálás képpé.
- Küszöbölés segítségével válassza le az alátéteket a háttérről.
- Kérje le az objektum kontúrokat:

```
vector<vector<Point>> cont;
vector<Vec4i> hier;
findContours(kuszobolt_kep, cont, hier, RETR_CCOMP, CHAIN_APPROX_NONE);
```

- Rajzolja körbe az alátétek külső körvonalát.
  - Járja be a kontúrhierarchiát tartalmazó vektort
    - Ha az i. kontúrnak nincs szülőeleme (hier[i][3] == -1), akkor külső kontúr (alátét vagy zaj)

Megj.: a drawContour-nál hiába adna meg maximum szintet. Azt a TREE-nél vetheti be.

# Feladat: Konvex burok

```
void convexHull (InputArray points, OutputArray hull,
        bool clockwise=false, bool returnPoints=true)
```

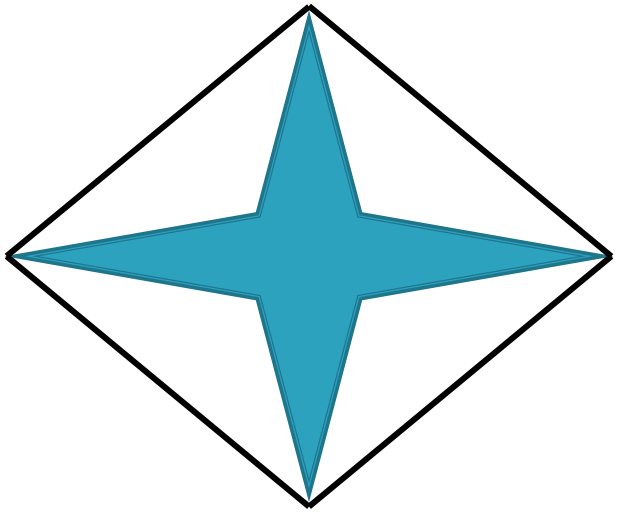
points: általában egy Point típusú elemeket tartalmazó vektor, ezen pontok konvex burkát keressük

hull: a kimeneti halmaz, vector<Point> vagy vector<int> típusú attól függően, hogy a returnPoints paraméter true vagy false-e.

clockwise: a kontúr bejárási iránya

returnPoints:

- true esetén a konvex burok csúcspontjait kapjuk meg
- false esetében a csúcspontok points vektorbeli pozícióját (indexét) kapjuk meg



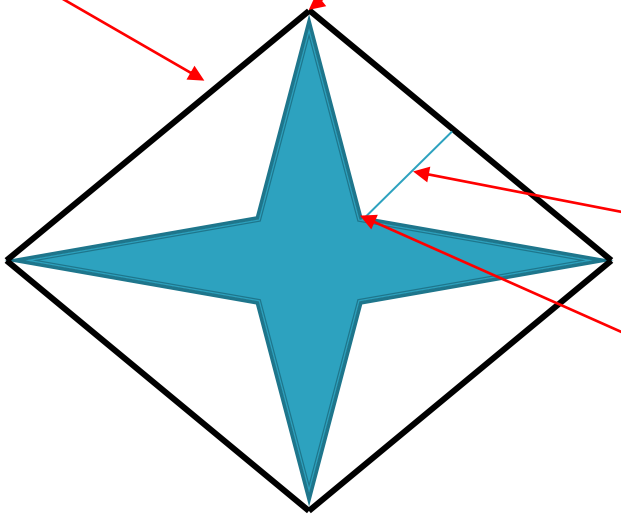
# Feladat: Konvex burok

```
void convexHull (InputArray points, OutputArray hull,
      bool clockwise=false, bool returnPoints=true)
```

- Olvassa be az objektumok.png képet színesben.
- Alakítsa át szürkeskálás képpé ( fekete – fehér kép lesz valójában ).
- Kérje le az objektum **külső** kontúrjait
- Járja be a kontúrokat és mindegyikre
  - rajzolja ki a kontúrt
  - alkalmazza a convexHull függvényt. A returnPoints paraméter legyen true.
  - rajzolja ki a konvex burkot:
    `polylines(img, hull, true, Scalar(0, 255, 0), 2);`

# Csúcspontok és völgypontok

Egy *P* kontúrpont és a konvex burok távolsága alatt a P-hez legközelebbi konvex burok pont és a P Euklideszi távolságát értjük.



# Völgypontok (defect points)

```
convexityDefects(
    InputArray contour,  //pl.: vector<Point> típusú
    InputArray hull,     //pl.: vector<int> típusú
    OutputArray defectPoints);  // vector<Vec4i> típusú
```

- A völgypontok értelmezése: (`Vec4i v`)

  `v[0]: a völgypontot megelőző, kontúrral érintkező pont indexe`

  `v[1]: a völgypont után álló, kontúrral érintkező pont indexe`

  `v[2]: a völgypont indexe`

  `v[3]: a völgypont és a kontúr becsült távolsága pixelben ~ v[3]/256.0`

  Az indexek a kontúr pontjaira hivatkoznak:

  `völgypont: contour[v[2]]`

# Feladat: Konvex burok

- Olvassa be az objektumok.png képet színesben.
- Alakítsa át szürkeskálás képpé ( fekete – fehér kép lesz valójában ).
- Kérje le az objektum külső kontúrjait

- Járja be a kontúrokat és mindegyikre
  - alkalmazza a convexHull függvényt, ezúttal a returnPoints legyen false.
    vector<int> hull2;
    convexHull(contours[i], hull2, false,  false);

- Hívja meg a convexityDefects függvényt:

  vector<Vec4i> defects;
  convexityDefects(contours[i], hull2, defects);
- Járja be a defects vektort és rajzolja ki a völgypontokat.

```
for (Vec4i d : defects){
    //  a contours[d[2]] a rajzolandó pont, ugyanis a d[2] csak indexet tartalmaz.
}
```

```
drawMarker(InputOutputArray srcdest, Point pt, int marker ...)
    srcdest: A kép, amire rajzol.
    pt: A jelölő leendő helye
    marker:  A jelölő típusa (MarkerTypes, pl: MARKER_CROSS)
```

Gyakorlófeladat.
A technika az előzővel azonos.

# Feladat2: Szárnyak kinyerése

- Olvassa be a képet szürkeskálában

  https://arato.inf.unideb.hu/szeghalmy.szilvia/kepfeld/img/szita.png

- Készítsen függvényt a legnagyobb területű szárny körvonalának meghatározására

```
void wing_detecor(const Mat gray, vector<Point>& wing);
```

  - Küszöbölje a képet (th = 100)
  - Kérje le az objektum külső körvonalát

```
vector<vector<Point>> contours;
findContours(kuszobolt_kep, contours, RETR_EXTERNAL, CHAIN_APPROX_NONE);
```

  - Járja be a kontúrokat és keresse meg a legnagyobb területű kontúr indexét

```
double contourArea(contour);
```

  - Mentse el a *wing* paraméterbe. (`wing=contour[max_index]`)

# Konvex burok

```
void convexHull(
    InputArray src,    // Point vektor vagy mátrix
    OutputArray dest, // Point vagy int vektor (az utolsó paramétertől függ)
    int clockwise,    // körüljárási irány
    bool returnPoints);  //true: kontúrpontot ad vissza,
                         //false: kontúrpont indexeit adja vissza
```

- Határozza meg meg a szárny konvex burkát:

```
vector<Point> hull;
convexHull(wing, hull, false, true);
//wing a korábban detektált szárny kontúrja
```

- Rajzolja ki a konvexburkot a következő függvénnyel:

```
polylines(InputOutputArray, InputArrayOfArrays, closed, ...)

InputOutputArray: A kép, amire rajzol.
InputArrayOfArrays: Point vektor (most a hull)
closed: True/False (most True)
```

# Feladat: Völgypontok

- Határozza meg a felső szárnyon lévő völgypontokat (szárnytő, szárnybütyök):
  - Kérje le a konvex burkot úgy, hogy a pontok indexét kapjuk meg.

```
vector<int> hull2;
convexHull(wing, hull2, false, false); //wing a szárny kontúrja
```

a korábban kinyert szárny pontok: vector<Point>

  - Kérje le a völgypontokat leíró vektort:

```
vector<Vec4i> defects;
convexityDefects(wing, hull2, defects);
```

- Járja be a völgypontokat és rajzolja ki azokat a képre, melyek legalább 10 pixel távolságra vannak a szárnytól (d[3]/256.0 – távolság; d[2] az index: wing[d[2]] a pont)

```
drawMarker(InputOutputArray srcdest, Point pt, int marker ...)
    srcdest: A kép, amire rajzol.
    pt: A jelölő leendő helye
    marker:  A jelölő típusa (MarkerTypes, pl: MARKER_CROSS)
```


99 little bugs in the code.
99 little bugs in the code.
Take one down, patch it around.
127 little bugs in the code...