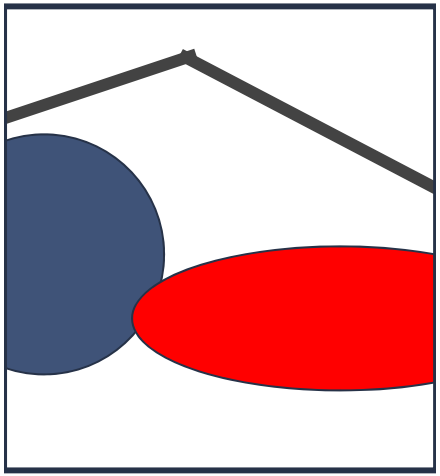
Računarska grafika
(2OER7O02)

Odsecanje (*Clipping*) objekata

Predavanja

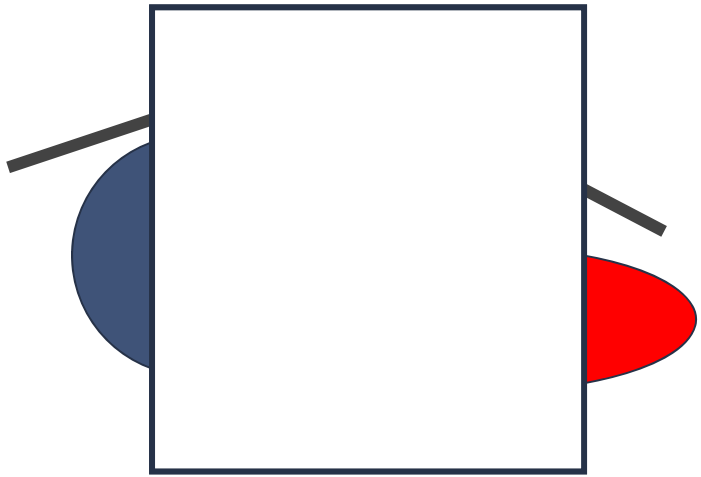
Isecanje (odsecanje, clipping)

- **Odsecanje** (Clipping) - uklanjanje svih delova primitiva izvan prozora kroz koji se slika posmatra.



Pokrivanje (covering, shielding)

- **Pokrivanje** (Covering, Shielding) - uklanjanje delova primitiva unutar prozora. Ova dva postupka su međusobno suprotni.

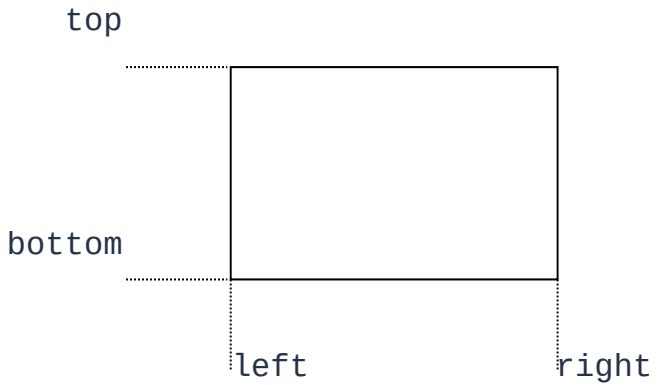


Algoritmi za odsecanje

- Algoritmi za odsecanje se dele prema sledećim kriterijumima:
  - prema **primitivi** koja se odseca (proizvoljna tačka, linija, poligon, krug, elipsa)
  - prema **prozoru** za odsecanje (clipping window) – (pravougaoni prozor, paralelan, konveksni poligon, proizvoljan poligon).

Isecanje tačke

- Jednostavan algoritam selektivnog postavljanja piksela
  - Algoritam rešava problem odsecanja proizvoljne tačke izvan pravougaonog prozora. Prozor je pravougaoni sa ivicama koje su paralelne osama i koje pripadaju prozoru.



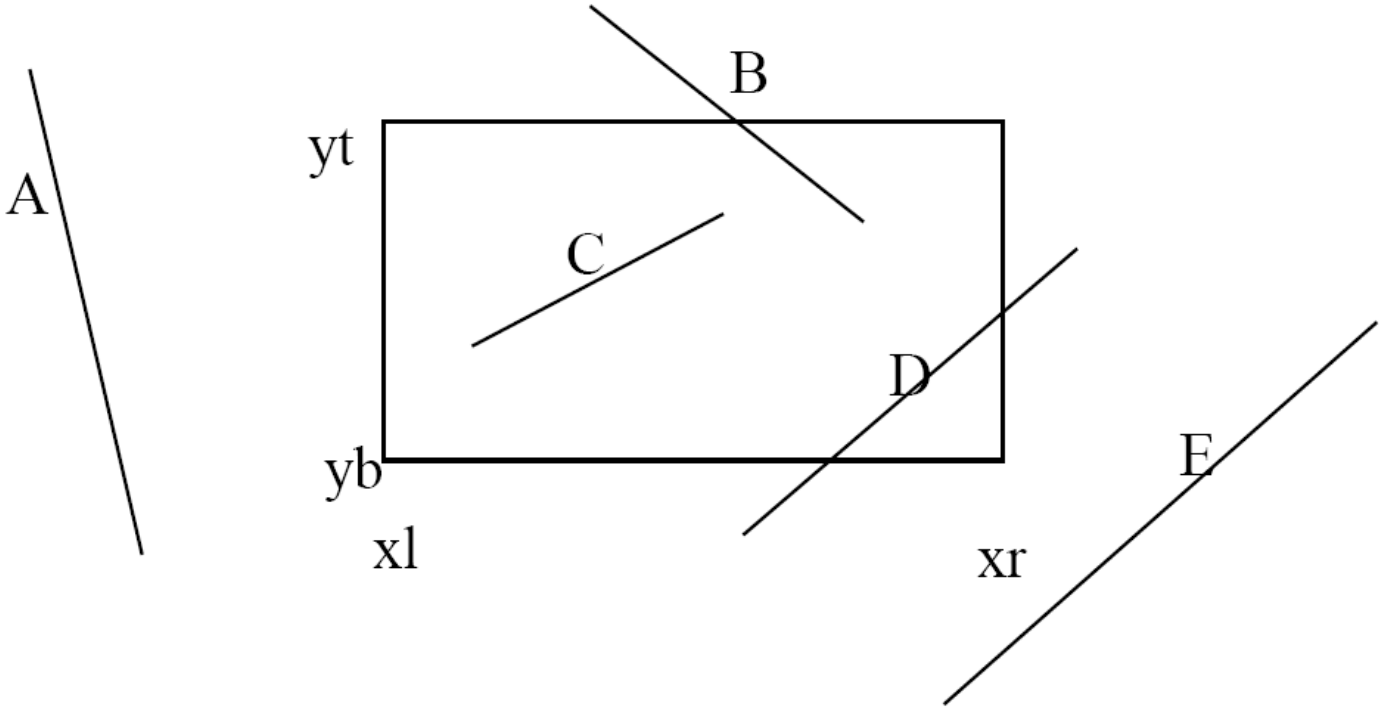
Isecanje tačke

```
void SelSetPixel(CDC* pDC, CRect w, CPoint p,
   COLORREF v)
{
  if ((w.left <= p.x) && (p.x <= w.right) &&
  (w.top >= p.y) && (p.y >= w.bottom))
      WritePixel(pDC,p.x,p.y,v)
}
```

Isecanje tačke - nedostaci

- Ovaj metod nije efikasan jer koristi dosta **if** naredbi.
- Bolje rešenje je posmatrati cele objekte, a ne njegove sastavne tačke (piksele).

Odsecanje linija



Algoritmi za odsecanje linija

- Aloritam “grube sile”
- Cohen-Sutherland algoritam
- Cyrus-Back algoritam

Algoritam grube sile

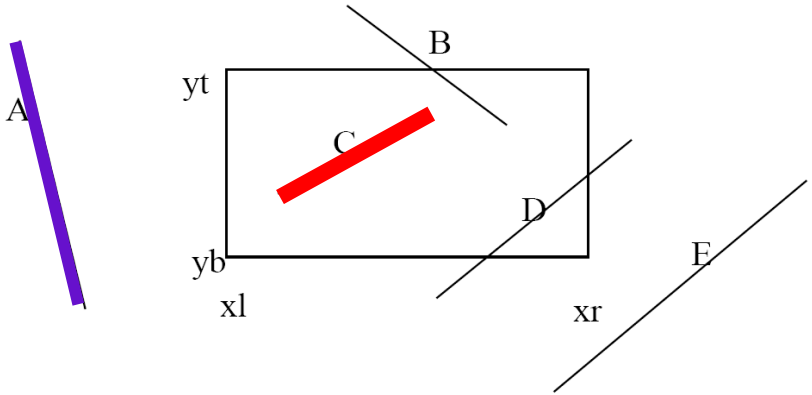
- Računaju se preseci linija sa linijama regiona za odsecanje i onda se testira šta je unutar, a šta van...
- **Nedostaci**: neefikasnost, sporost

Cohen-Sutherland algoritam

- **Cohen-Sutherland**-ov algoritam efikasno rešava problem odsecanja linija izvan pravougaonog prozora.
- Osnovna ideja algoritma je da se najpre pokuša da se linija u celini prihvati ili odbaci. Ukoliko se ne uspe i prvom koraku, određuje se presek linije i produžene ivice prozora pa se ponovo pokušava prihvatanje ili odbacivanje preostalog dela linije.

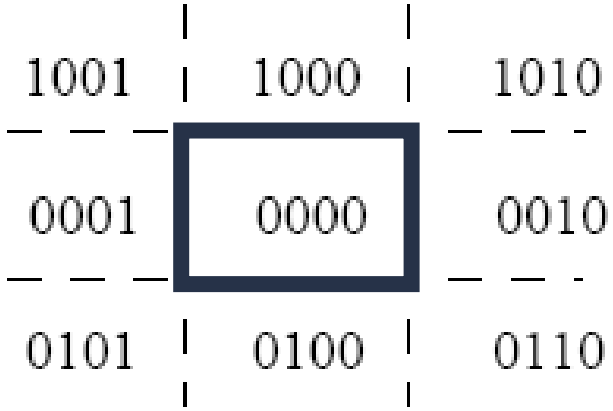
Cohen-Sutherland algoritam

- Posmatraju se krajnje tačke linija.
- Uslov **trivijalnog prihvatanja**:
  - Obe krajnje tačke su unutar regiona za odsecanje.
- Uslov **trivijalnog odbacivanja**:
  - Obe krajnje tačke su levo od x_l ili desno od x_r ili iznad y_t ili ispod y_b.
- U svim ostalim slučajevima neophodno je izračunavanje preseka.



Cohen-Sutherland algoritam

1. Ivice prozora se produže tako da se cela slika podeli u 9 oblasti.



Cohen-Sutherland algoritam

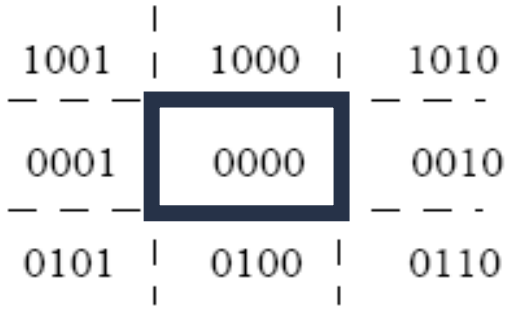
2. Svakoj oblasti se pridružuje 4-bitni položajni kod (Outcode): $b_3b_2b_1b_0$, gde svaki bit označava jednu oblast:

   b_3 - iznad,

   b_2 - ispod,

   b_1 - desno,

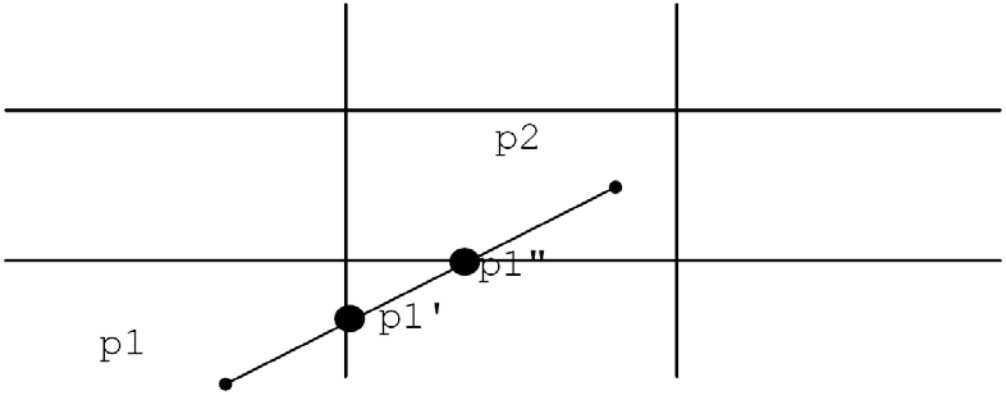
   b_0 – levo



$b_3b_2b_1b_0$

Cohen-Sutherland algoritam

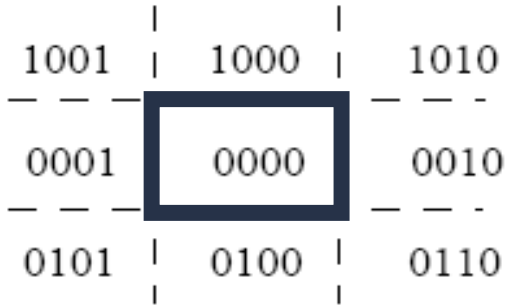
3. Za krajnje tačke linije p1 i p2 određuju se položajni kodovi c1 i c2



Cohen-Sutherland algoritam

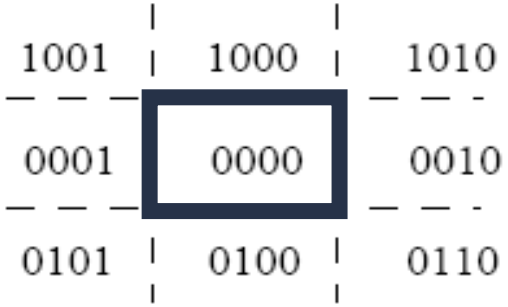
4. Ako su i c1 i c2 jednaki 0 tada je:

   (c1 | c2)==0 $\Rightarrow$ linija se **trivijalno prihvata** jer se nalazi u vidnom polju prozora.



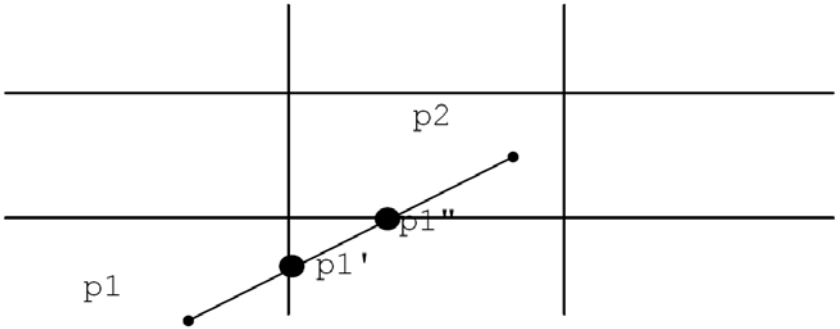
Cohen-Sutherland algoritam

5. Ako c1 i c2 imaju barem jedan zajednički bit
(c1 & c2)!=0 $\Rightarrow$ linija se **trivijalno odbacuje**



Cohen-Sutherland algoritam

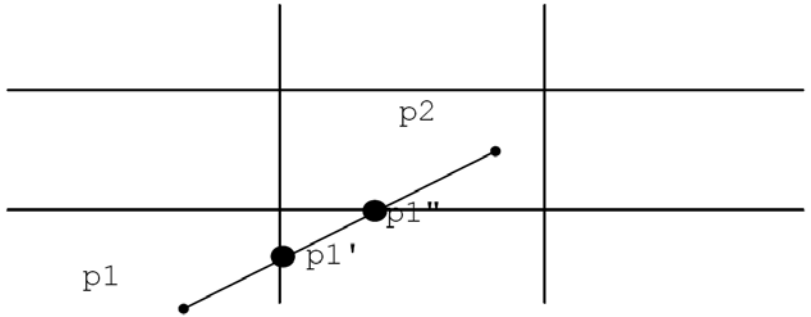
6. Za preostale linije ispituje se da li je tačka koja nije u prozoru (c!=0) levo, desno, iznad ili ispod prozora, te se nalazi presek linije koja se odseca sa odgovarajućom produženom ivicom prozora.



Cohen-Sutherland algoritam

7. Krajnja tačka linije se premešta u presečnu tačku (P1→P1') i izračunava joj se položajni kod.

8. Ide se na korak 4.



```
typedef unsigned int outcode;
enum { TOP = 0x8, BOTTOM = 0x4, RIGHT = 0x2, LEFT = 0x1 };

void CohenSutherlandLineClipAndDraw( CDC* pDC, double x0, double y0, double x1,
   double y1, double xmin, double xmax, double ymin, duble ymax, COLORREF value)
{
outcode outcode0, outcode1, outcodeOut;
boolean accept = FALSE, not_done = TRUE;

outcode0 = CompOutCode( x0, y0, xmin, ymin, xmax, ymax );
outcode1 = CompOutCode( x1, y1, xmin, ymin, xmax, ymax );
while (not_done) {
  if( !(outcode0|outcode1) ) {   /* Trivijalno prihvatanje */
     accept = TRUE;
     not_done = FALSE;
  }
  else if( outcode0&outcode1)    /* Trivijalno odbacivanje */
     not_done = FALSE;
  else {
```

```
double x, y;
outcodeOut = outcode0 ? outcode0 : outcode1;
/* Nalazenje presecne tacke */
if ( outcodeOut & TOP ) {
   x = x0 + (x1-x0)*(ymax - y0)/(y1 - y0);
   y = ymax;
}
else if (outcodeOut & BOTTOM) {
   x = x0 + (x1-x0)*(ymin - y0)/(y1 - y0);
   y = ymin;
}
else if (outcodeOut & RIGHT) {
   y = y0 + (y1-y0)*(xmax - x0)/(x1 - x0);
   x = xmax;
}
else {
   y = y0 + (y1-y0)*(xmin - x0)/(x1 - x0);
   x = xmin;
}
```

```
  if ( outcodeOut == outcode0 ) {
     x0 = x;
     y0 = y;
     outcode0 = CompOutCode( x0, y0, xmin, xmax, ymin, ymax);
  }
  else {
     x1 = x;
     y1 = y;
     outcode1 = CompOutCode( x1, y1, xmin, xmax, ymin, ymax);
  }
 }
}
if ( accept ) {
   /** Crtanje linije **/
   BresenhamLine( pDC, x0, y0, x1, y1, value );
}
```

```
outcode CompOutCode(double x, double y, double xmin, double xmax,
    double ymin, double ymax)
{
     outcode code = 0;
     if ( y > ymax )
        code |= TOP;
     else if ( y < ymin )
        code |= BOTTOM;
     if ( x > xmax )
        code |= RIGHT;
     else if ( x < xmin )
        code |= LEFT;
     return code;
}
```

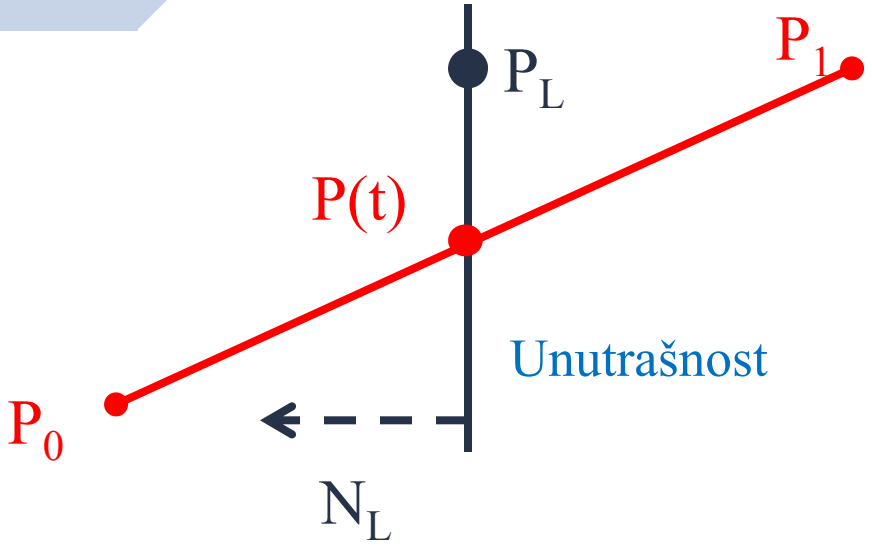
Cyrus-Beck algoritam

- Optimizuje se postupak nalaženja preseka

1. Startuje se od parametarske jednačine linije:
   - $P(t) = P_0 + (P_1 - P_0)\, t$
2. Izabrati tačke koje pripadaju ivicama regiona za kliping i izračunati normale za svaku ivicu
   - P_L, N_L

Cyrus-Beck algoritam

3. Naći t tako da je:
   - $N_L \bullet [P(t) - P_L] = 0$



a) Zameniti P(t) u jednačini

b) Rešiti jednačinu po t

- $t = N_L \bullet [P_0 – P_L] \ / \ -N_L \bullet [P_1 - P_0]$

Cyrus-Beck algoritam

$P(t) = P_0 + (P_1 - P_0)\, t$ (1)

$N_L \bullet [P(t) - P_L] = 0$ (2) zameniti (1) u (2)

$N_L \bullet [P_0 + (P_1 - P_0)\, t - P_L] = 0$ (3) distribuirati skalarni proizvod

$N_L \bullet [P_0 - P_L] + N_L \bullet [P_1 - P_0]\, t = 0$ (4) rešiti po t

$t = N_L \bullet [P_0 - P_L] / -N_L \bullet [P_1 - P_0]$

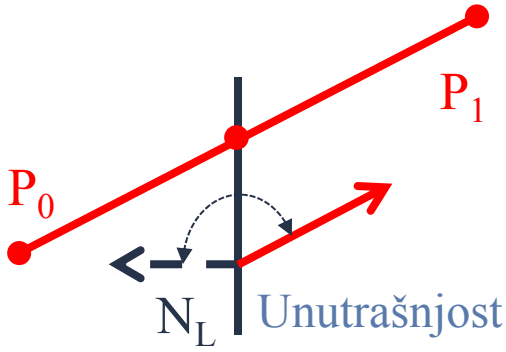
da bi postojalo rešenje, imenilac mora da bude različit od 0; N_L može da bude 0 samo greškom, $[P_1 - P_0]$ je 0 samo ako je $P_1 = P_0$, a ako je $N_L \bullet [P_1 - P_0] = 0$, ivica i linija su paralelne (pa se ne seku ili se poklapaju, tj. ide se na sledeći korak)

Cyrus-Beck algoritam

6. Izračunati t za preseke linije sa svim ivicama regiona za kliping.
7. Odbaciti sve preseke sa (t < 0) i (t > 1)
8. Preostale preseke klasifikovati kao:
   - **Potencijalno ulazne** (Potentially Entering (PE))
   - **Potencijalno izlazne** (Potentially Leaving (PL))

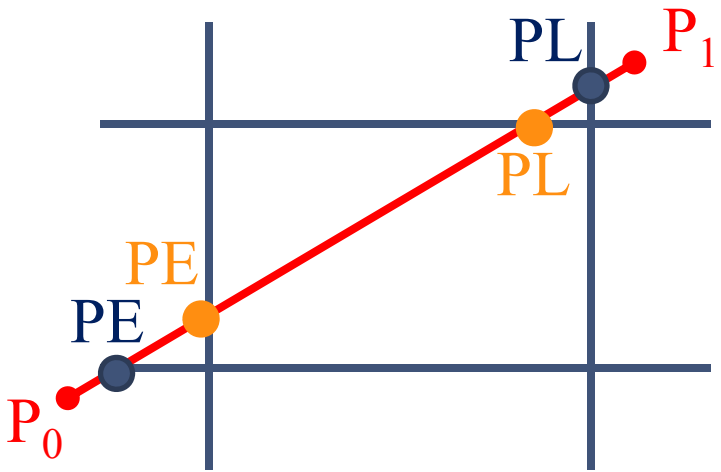
$N_L \bullet [P_1 - P_0] > 0 \Rightarrow PL$

$N_L \bullet [P_1 - P_0] < 0 \Rightarrow PE$



Cyrus-Beck algoritam

- Naći PE sa najvećim t
- Naći PL sa najmanjim t
- Nacrtati liniju između ove dve tačke



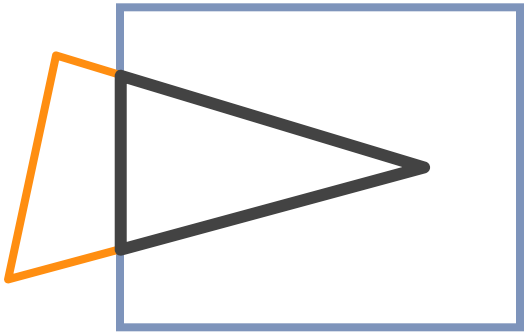
Poređenje

- **Cohen-Sutherland**
  - Neefikasan je kada se radi o ponavljanju odsecanja.
  - Najbolje rezultate daje kada se najveći broj linija može trivijalno prihvatiti ili odbaciti.
- **Cyrus-Beck**
  - Efikasno izračunavanje t-preseka.
  - Izračunavanje (x,y) tačaka za kliping se vrši samo jedanput.
  - Algoritam ne razmatra trivijalno prihvatanje i odbacivanje.
  - Najbolji je kada postoji dosta linija za kliping.
- Postoji i optimizovani Cyrus-Beck algoritam, poznat kao **Liang-Barsky** algoritam

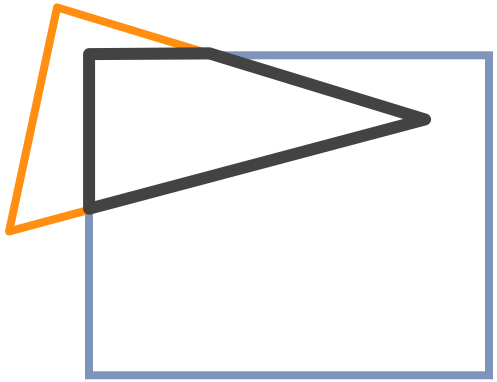
Odsecanje poligona

- Mnogo je složenije od odsecanja linija
  - Ulaz: poligon
  - Izlaz: originalni poligon, novi poligon, ili ništa...

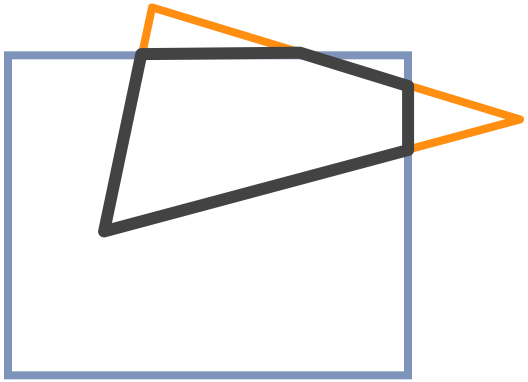
Odsecanje trougla



trougao ⇨ trougao



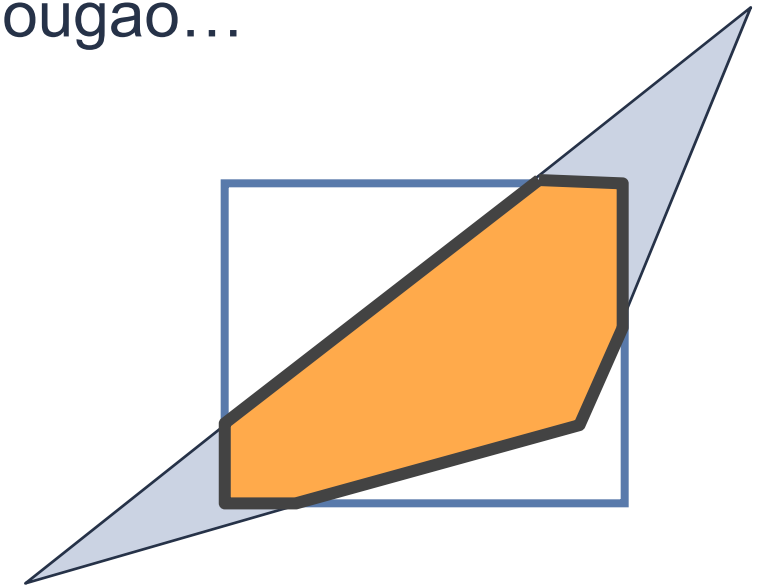
trougao ⇨ četvorougao



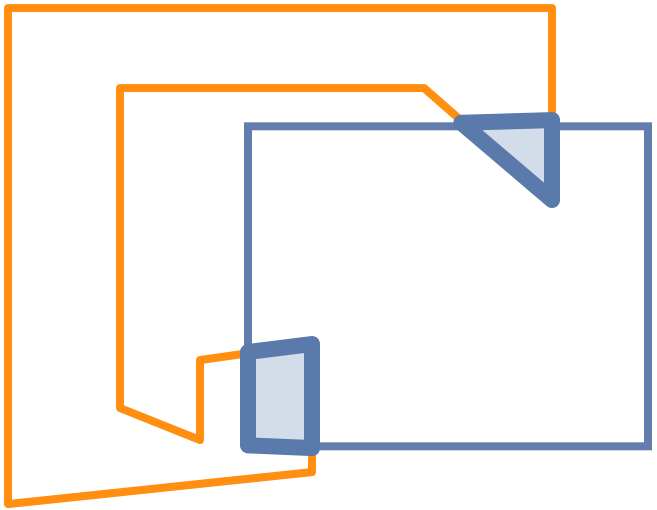
trougao ⇨ 5-gon

Odsecanje trougla

- Sedmougao…



Odsecanje proizvoljnog poligona



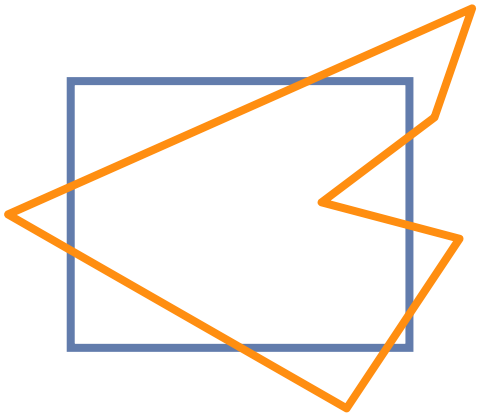
konkavni poligon ⇨ više poligona

Algoritmi za odsecanje poligona

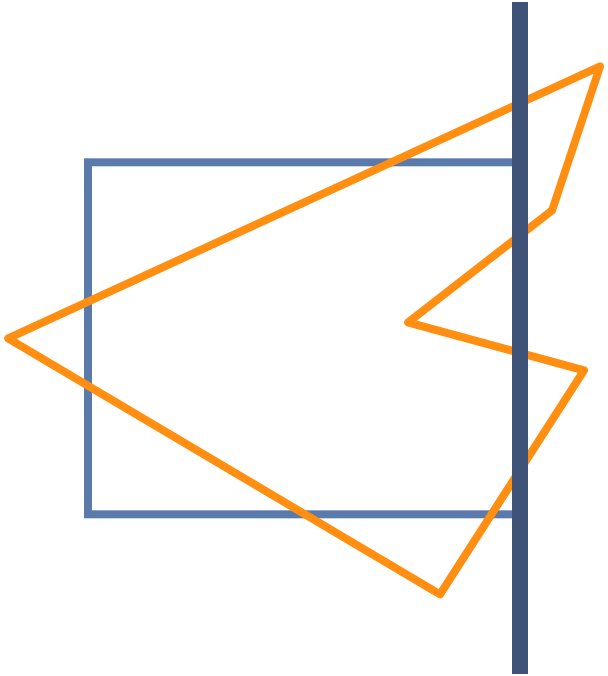
- **Sutherland-Hodgman** algoritam
- **Weiler-Atherton** algoritam

Sutherland-Hodgeman algoritam

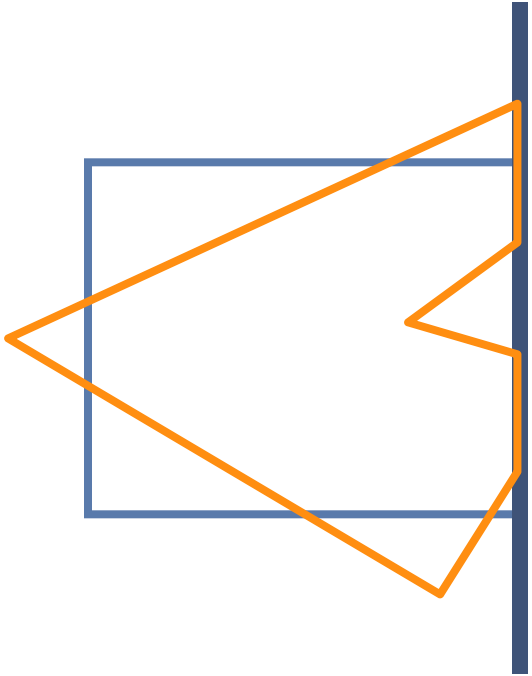
- Osnovna ideja:
  - Vrši se isecanje poligona u odnosu na neku od ivica koje se posmatraju kao granice poligona.
  - Postupak se nastavlja za sve ivice regiona za kliping.



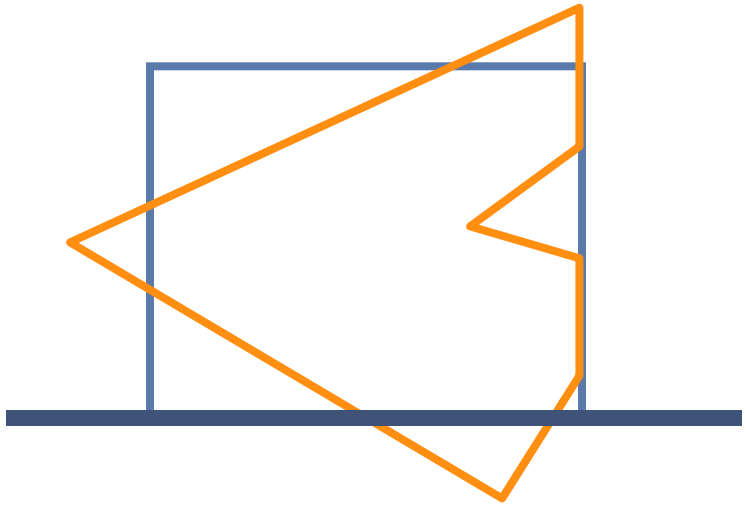
Sutherland-Hodgeman algoritam



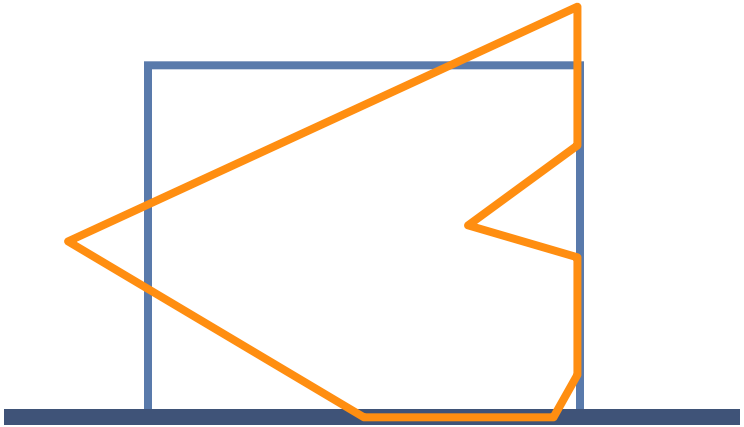
Sutherland-Hodgeman algoritam



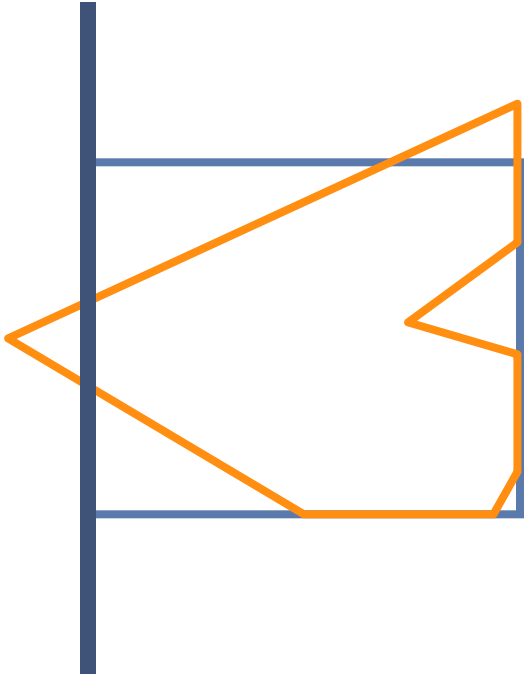
Sutherland-Hodgeman algoritam



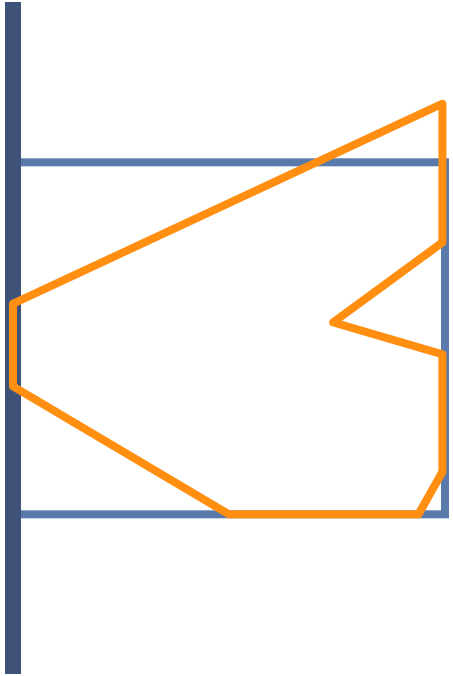
Sutherland-Hodgeman algoritam



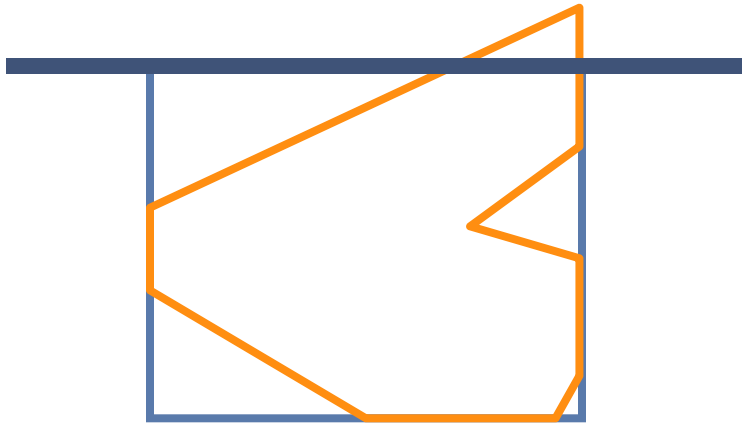
Sutherland-Hodgeman algoritam



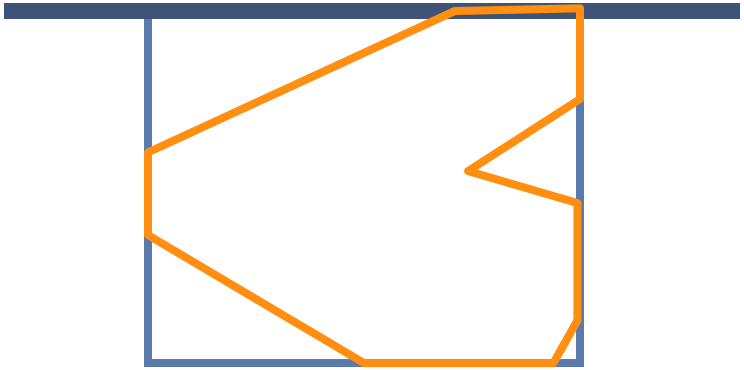
Sutherland-Hodgeman algoritam



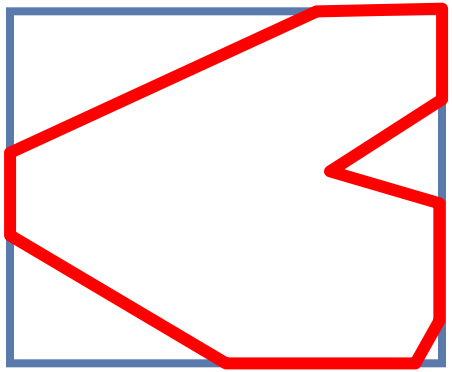
Sutherland-Hodgeman algoritam



Sutherland-Hodgeman algoritam



Sutherland-Hodgeman algoritam



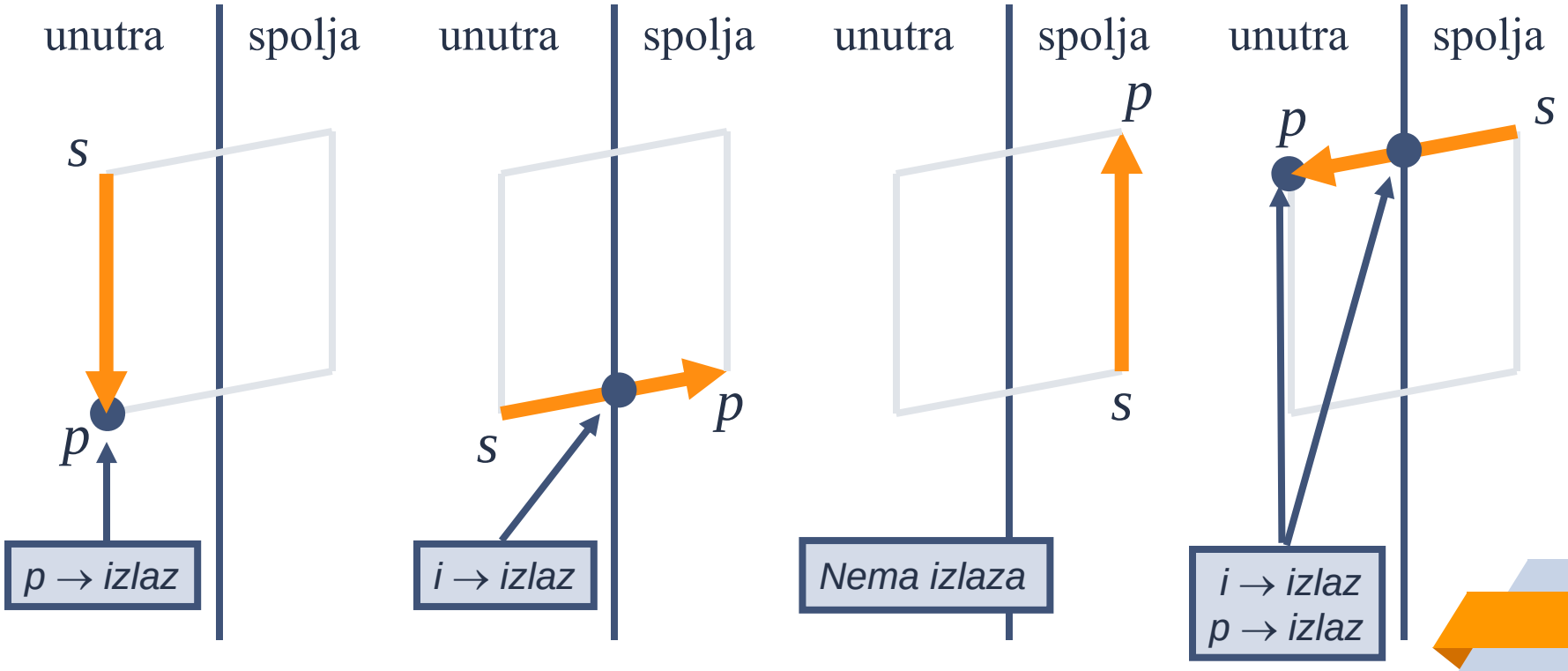
Sutherland-Hodgeman algoritam

- Ulaz/izlaz za algoritam:
  - Ulazna lista: uređena lista temena poligona
  - Izlazna lista: lista temena novog poligona koja se sastoji od nekih starih i nekih novih temena (najčešće)

Sutherland-Hodgeman algoritam

- Oznake:
  - ***p*** - tačka za koju se trenutno proverava da li treba da ide u izlaznu listu.
  - ***s*** - tačka iz prethodne iteracije.

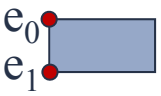
Sutherland-Hodgeman algoritam

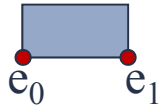


Sutherland-Hodgeman algoritam

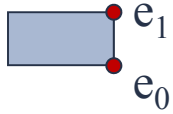
- Četiri slučaja:
  - ***s*** i ***p*** su unutra
    - Dodati ***p*** u izlaznu listu
    - ***s*** je već u listi
  - ***s*** je unutra a ***p*** spolja
    - Naći tačku preseka ***i***
    - Dodati ***i*** u izlaznu listu
  - ***s*** i ***p*** su spolja
    - Ništa se ne dodaje u izlaznu listu
  - ***s*** je spolja a ***p*** je unutra
    - Naći tačku preseka ***i***
    - Dodati ***i*** i ***p*** u izlaznu listu

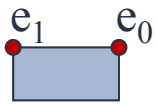
```
typedef struct Vertex{
        float x,y;
        } Vertex
typedef Vertex edge[2];
typedef Vertex VertexArray[MAX]; /*MAX je deklarisana konst.*/

Vertex* PolygonClipping(Vertex *inputVertexArray){
   int inLength;
   edge[0] = Top_Left_Vertex_of_Clipping_Window; /*Leva ivica*/
   edge[1] = Bottom_Left_Vertex_of_Clipping_Window;
   inLength = Size_of(inputVertexArray);
   SutherlandHodgmanPolygoClip (inputVertexArray,
        outVertexArray, inLength, outLength, edge);
   OutputToInput(inLength, inVertexArray, outLength, outVertexArray);

   edge[0] = Bottom_Left_Vertex_of_Clipping_Window; /*Donja ivica*/
   edge[1] = Bottom_Right_Vertex_of_Clipping_Window;
   SutherlandHodgmanPolygoClip (inputVertexArray,
        outVertexArray, inLength, outLength, edge);
   OutputToInput(inLength, inVertexArray, outLength, outVertexArray);
```





```
    edge[0] = Bottom_Right_Vertex_of_Clipping_Window; /*Desna ivica*/
    edge[1] = Top_Right_Vertex_of_Clipping_Window;

    SutherlandHodgmanPolygoClip (inputVertexArray,
         outVertexArray, inLength, outLength, edge);
    OutputToInput(inLength, inVertexArray, outLength, outVertexArray);


    edge[0] = Top_Right_Vertex_of_Clipping_Window;   /*Gornja ivica*/
    edge[1] = Top_Left_Vertex_of_Clipping_Window;

    SutherlandHodgmanPolygoClip (inputVertexArray,
         outVertexArray, inLength, outLength, edge);
    OutputToInput(inLength, inVertexArray, outLength, outVertexArray);

    return(outVertexArray);
}
```





```
void SutherlandHodgmanPolygoClip(
    Vertex *inVertexArray ,        /*Ulazna lista temena*/
    Vertex *outVertexArray,        /*Izlazna lista temena*/
    int *inLength,                 /*Broj temena u ulaznoj listi */
    int *outLength,                 /*Broj temena u izlaznoj listi */
    Vertex *clipBoundary)          /*Ivica kliping poligona*/
{
    Vertex s,p,i;
    int j;
    *outLength = 0;
    /*Krecemo od poslednjeg temena u listi inVertexArray*/
    s = inVertexArray[inLength-1];
    for (j = 0; j < inLength; j++){
       p = inVertexArray[j];
        if (Inside(p,clipBoundary)){      /*Slucajevi 1 i 4*/
          if (Inside(s, clipBoundary)){
           Output(p, outLength, outVertexArray);  /*Slucaj 1*/
```

```
else {                                                  /*Slucaj 4*/
            Intersect(s, p, clipBoundary, i);
            Output(i, outLength, outVertexArray);
            Output(p, outLength, outVertexArray);
            }
         }
         else {                                        /*Slucajevi 2 i 3*/
            if (Inside(s, clipBoundary)){              /*Slucaj 2*/
                 Intersect(s, p, clipBoundary, i);
                 Output(i, outLength, outVertexArray);
             }
                                                       /*Slucaj 3*/
        s = p;     /*priprema sledece iteracije*/
     }
}      /*SutherlandHodgmanPolygonClip*/
```

```
void OutputToInput(int *inLength, Vertex *inVertexArray, int *outLength, Vertex
   *outVertexArray )
{
        if ((*inLength == 2) && (*outLength == 3)){
                inVertexArray[0].x = outVertexArray[0].x;
                inVertexArray[0].y = outVertexArray[0].y;
               /*Prva dva temena su ista*/
                if (outVertexArray[0].x == outVertexArray[1].x)
                {
                        inVertexArray[1].x = outVertexArray[2].x;
                        inVertexArray[1].y = outVertexArray[2].y;
                }
                else
                {
                        inVertexArray[1].x = outVertexArray[1].x;
                        inVertexArray[1].y = outVertexArray[1].y;
                }
                *inLength = 2;
        }
```

```
        else
      /* postavi outVertexArray kao inVertexArray u sledecoj iteraciji*/
        {
                *inLength = *outLength;
                for (i = 0; i < outLength; i++)
                {
                        inVertexArray[i].x = outVertexArray [i].x;
                        inVertexArray[i].y = outVertexArray [i].y;
                }
        }
}
```

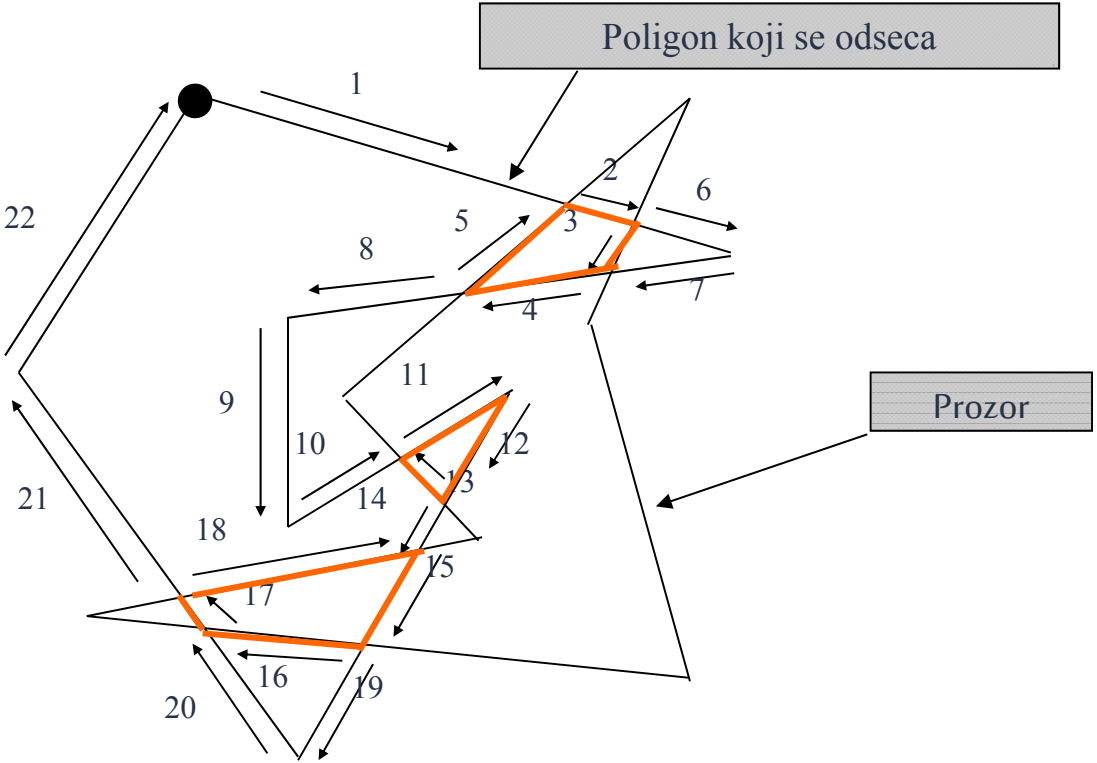
```
boolean Inside(point testVertex, point *clipBoundary)
{
  if (clipBoundary[1].x > clipBoundary[0].x)   /*donja ivica*/
    if (testVertex.y >= clipBoundary[0].y) return TRUE;
  if (clipBoundary[1].x < clipBoundary[0].x)  /*gornja ivica*/
   if (testVertex.y <= clipBoundary[0].y) return TRUE;
  if (clipBoundary[1].y > clipBoundary[0].y)  /*desna ivica*/
    if (testVertex.x <= clipBoundary[1].x) return TRUE;
  if (clipBoundary[1].y < clipBoundary[0].y)  /*leva ivica*/
    if (testVertex.x >= clipBoundary[1].x) return TRUE;
  return FALSE;
}
```

```
void Intersect(point first, point  second, point  *clipBoundary, point  *intersectPt) {
  if (clipBoundary[0].y == clipBoundary[1].y) { /*horizontalna ivica*/
          intersectPt->y = clipBoundary[0].y;
          intersectPt->x = first.x +(clipBoundary[0].y-first.y)*
          (second.x-first.x)/(second.y-first.y);
   }
    else {                                          /*vertikalna ivica*/
           intersectPt->x = clipBoundary[0].x;
           intersectPt->y = first.y +(clipBoundary[0].x-first.x)*
                   (second.y-first.y)/(second.x-first.x);
          }
}


Void Output(Vertex newVertex, int* outLength, Vertex outVertexArray);
{
        (*outLength)++;
        outVertexArray[*outLength-1].x = newVertex.x;
        outVertexArray[*outLength-1].y = newVertex.y;
}
```

Weiler-Atherton algoritam

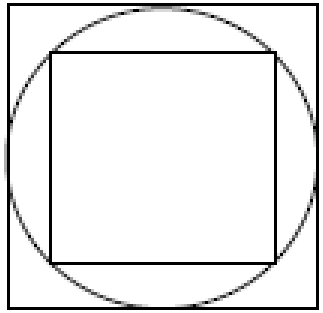
- Osnovna ideja:
  - Algoritam rešava problem odsecanja proizvoljnog poligona izvan prozora, koji je proizvoljan poligon. Praktično, određuje se presek dva proizvoljna poligona. Oba poligona mogu biti i konkavna, a takođe, mogu sadržati i rupe. Algoritam kreće od proizvoljnog temena zadatog poligona koji se odseca, u pravcu kazaljke časovnika (temena su uređena na ovaj način). Prati se ivica poligona koji se odseca sve do preseka sa ivicom poligona-prozora. Ako ivica "ulazi" u prozor, nastavlja se praćenje ivice poligona koji se odseca. Ako ivica "izlazi" iz prozora, skreće se "udesno" i nastavlja ivicom prozora, na način kao da je on sada poligon za odsecanje, a originalni poligon za odsecanje sada prozor. Tačke preseka se pamte da bi se obezbedilo da se svi putevi pređu tačno jednom.

Weiler-Atherton algoritam



Odsecanje kruga i elipse

- Algoritam:
  1. Vrši se trivijalno prihvatanje ili odbacivanje celog kruga (elipse)



Odsecanje kruga i elipse

2. Deli se krug (elipsa) na kvadrante i vrši se trivijalno prihvatanje ili odbacivanje kvadranata.
3. Nalaze se presečne tačke kruga (elipse) sa regionom za odsecanje i vrši se iscrtavanje dela kruga (elipse).

PITANJA