RAČUNARSKA GRAFIKA

Oznaka predmeta: RG

Predavanje broj: 07

Nastavna jedinica: Popunjavanje. Odsecanje.

Nastavne teme: Popunjavanje konveksnog poligona. Implementacija. Popunjavanje proizvoljnog poligona: algoritam, problemi. Popunjavanje oblasti. Tipovi oblasti. Popunjavanje poplavom (flood-fill). Popunjavanje oivičene oblasti. Iterativni algoritam. Selektivno postavljanje piksela. Cohen-Sutherland algoritam. Sutherland-Hodgman-ov algoritam. Weiler-Atherton-ov algoritam. Algoritam Liang-Barsky.

Predavač: prof. dr Perica S. Štrbac, dipl. ing.

Literatura:

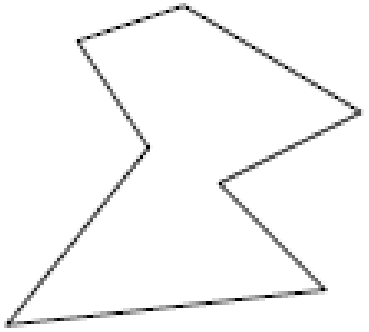
James D. Foley, Andries van Dam, Steven K. Feiner, John F. Hughes: "Computer Graphics: Principles and Practice", 2nd ed. in C, Addison-Wesley, 1996.

Popunjavanje

- Dva načina definisanja površine koju treba popuniti:
  - vektorsko
    - površina je poligon - poznate su koordinate svakog njegovog temena
    - može se "izračunati" koji su pikseli unutar poligona
      - Popunjavanje vektorski zadate površine
        - » **izračunato popunjavanje** ili **popunjavanje poligona** (*pre-computed* ili *poligon filling*)
  - rastersko
    - površina je oblast opšteg oblika i postoji jedino u memoriji
    - popunjavanje počinje od zadate tačke u unutrašnjosti oblasti i širi se na okolinu
      - Popunjavanje rasterski definisane površine
        - » **popunjavanje oblasti** (*region filling*)

Popunjavanje konveksnog poligona

- Ograničenje
  - zahteva se konveksnost poligona barem u pravcu jedne od osa 2D sistema
- Ovde se radi sa konveksnošću u X pravcu
  - slično bi bilo raymatranje za konveksnost u Y pravcu



- Uvodi se xval-vektor čiji je indeks y-koordinata, a vrednosti elemenata su odgovarajuće vrednosti x-koordinate
- Algoritam:
  - 1. Postaviti sve elemente xval na (-1)
  - 2. Generisati poliliniju
    pri tome za piksel sa koordinatama (x,y) proveravati xval[y]:

    a) ako je xval[y]<0 onda xval[y]=x

    b) ako je xval[y]>=0 onda
    popuniti sve piksele između (x,y) i (xval[y] , y)

Implementacija

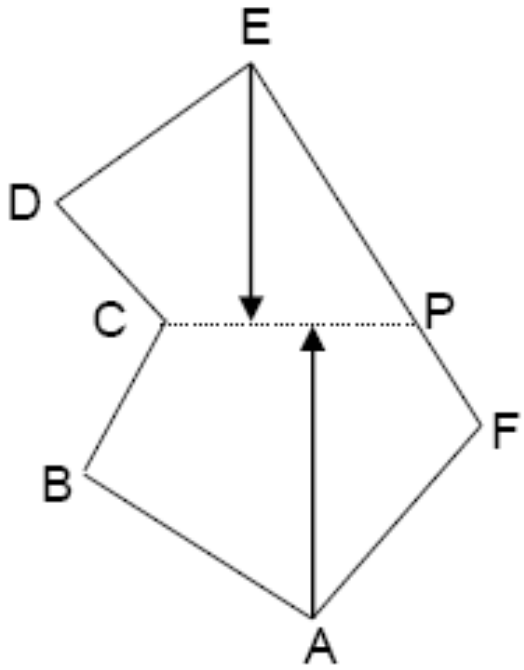
- Realizacija:
  - 1. pre crtanja potrebno je inicijalizovati xval pozivom setXValue
  - 2. kada se zahteva crtanje popunjenog poligona, iz procedure za crtanje linijskog segmenta polilinije (npr. Bresenhamovim algoritmom) poziva se procedura pixelFill umesto setPixel

```
int xval[MaxY];
void setXValue()
{
  for( int y=0; y<MaxY; y++) xval[y]=-1;
}
void pixelFill(int x, int y, int value);
{
  int x0,x1,i;
  if ( xval[y]>=0 )
  {
    x0 = min(x,xval[y]);
    x1 = max(x,xval[y]);
    for(i = x0; i<=x1; i++) setPixel(i,y,v);
  }
  xval[y] = x;
}
```

Efekat popunjavanja

- Za poligon sa slike,
  ukoliko se crtanje počne
  od temena C u smeru (C→D),
  popunjavanje počinje od E prema F
  i traje do tačke P;

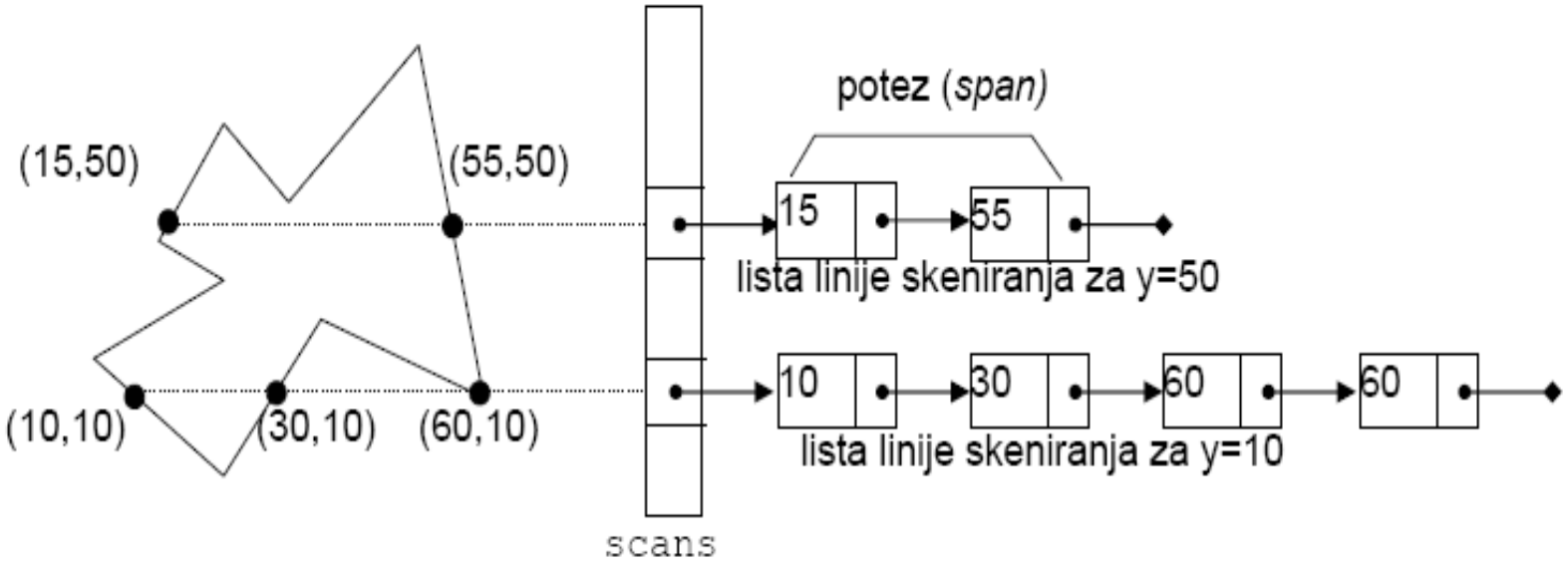
  od P do A nema popunjavanja,
  pa ponovo počinje od tačke A do tačke C



- Napomena:
  - kada bi procedura za crtanje linije uvek crtala liniju CD u smeru (C→D), efekat bi bio upravo kako je i opisano
  - to najčešće nije slučaj, pa je i efekat nešto drugačiji

Popunjavanje proizvoljnog poligona

- Pretpostavka:
  - ivica (granica) i unutrašnjost se popunjavaju istom vrednošću

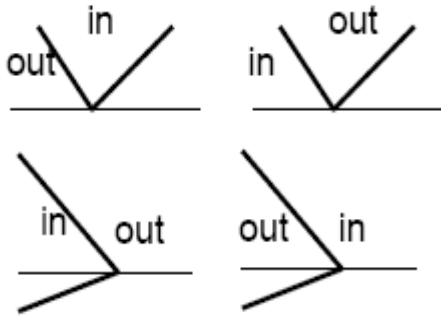


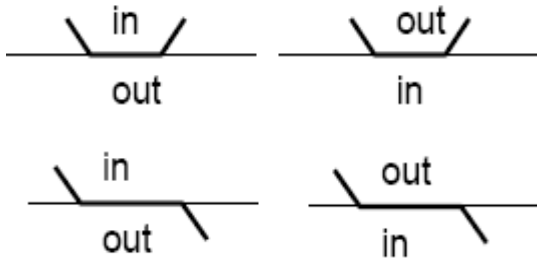
Algoritam

- Svaka y-vrednost definiše jednu horizontalnu liniju skeniranja (*scan line*)
- Svakoj liniji skeniranja odgovara jedan pokazivač u vektoru scans
- Ivice poligona se "obilaze" redom
  - modifikovanim Bresenham-ovim algoritmom za crtanje linije
- Izračunat (x,y) par koji leži na ivici (liniji) daje jedan element u listi na koju pokazuje scans[y]
- Liste x vrednosti se formiraju u rastućem poretku
- Sukcesivni elementi, i to neparan-paran, formiraju poteze
  - horizontalne linijske segmente (*span*)
- Potezi pripadaju unutrašnjosti oblasti i oni se prikazuju

Problemi

- Postoje sledeći problemi sa linijama skeniranja
- Linija skeniranja prolazi kroz teme poligona
  - ako su oba susedna temena sa iste strane: sve je u redu, kritično teme ulazi dva puta u listu
  - ako su susedna temena sa različitih strana: korekcija, kritično teme ulazi samo jedanput u listu
- Linija skeniranja prolazi kroz horizontalnu ivicu poligona
  - sama horizontalna ivica se preskače, ali se naknadno crta linija koja joj odgovara
  - ako su oba susedna temena sa iste strane: sve je u redu, oba kritična temena ulaze u listu
  - ako su susedna temena sa različitih strana: korekcija, samo jedno kritično teme ulazi u listu





Popunjavanje oblasti

- Oblast je grupa susednih, povezanih piksela
- Za oblast je karakteristično
  - definisana je rasterski - vrednošću piksela u video memoriji
  - nije definisana vektorski - poligonom koji je ograničava
- Popunjavanje oblasti je moguće samo na uređajima koji poseduju memoriju u kojoj je smeštena slika.
- Potrebna je funkcija za očitavanje vrednosti proizvoljnog piksela:

  int getPixel(int x, int y);

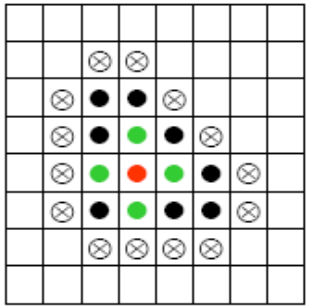
- Uobičajeno, oblast se definiše na jedan od dva načina:
  - svi pikseli koji pripadaju oblasti imaju datu vrednost
  - pikseli koji su na ivici oblasti imaju datu vrednost

Tipovi oblasti(1)

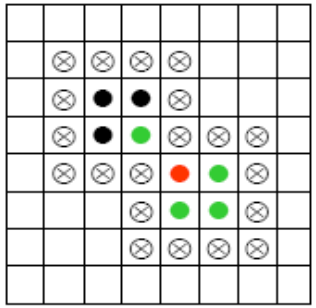
- Klasifikacija oblasti prema načinu definisanja
- Oblast definisana unutrašnošću (*interior-defined*)
  - svi pikseli u oblasti imaju vrednost old_value, i nema piksela na granici oblasti sa tom vrednošću
    - algoritmi "poplave" (*flood-fill*) rade nad ovakvim oblastima da postave njihove piksele na vrednost new_value
- Oblast definisana granicom (*boundary-defined*)
  - pikseli na granici oblasti imaju vrednost boundary_value, dok pikseli u oblasti imaju proizvoljnu drugu vrednost
  - algoritmi "oivičenog popunjavanja" rade nad ovakvim oblastima da postave sve njihove piksele na vrednost new_value

Tipovi oblasti(2)

- Klasifikacija oblasti prema povezanosti piksela
- Četvoro-susedna oblast (*4-connected*)
  - od proizvoljnog piksela u oblasti do svih drugih u oblasti može se stići sekvencom horizontalnih i vertikalnih pomeraja za 1 piksel (ortogonalno kretanje)
- Osmo-susedna oblast (*8-connected*)
  - od proizvoljnog piksela u oblasti do svih drugih u oblasti može se stići sekvencom horizontalnih, vertikalnih i dijagonalnih pomeraja za 1 piksel



Četvorosusedna oblast ●
Osmosusedna granica ⊗



Osmosusedna oblast ●
Četvorosusedna granica ⊗

- Granica 4-susednih oblasti 8-susedna, a 8-susednih oblasti je 4-susedna

Popunjavanje poplavom (flood-fill)

- Jednostavni rekurzivni algoritam za
  - 8-susednu oblast
  - definisanu unutrašnjošću (old_val)
  - koji postavlja unutrašnjost na new_val

```
void floodFill8(int x, int y, int new_val, int old_val)
{
  if ( getPixel(x,y) == old_val )
  {
    setPixel(x,y,new_val);
    floodFill8(x-1,  y,new_val,old_val);
    floodFill8(x+1,  y,new_val,old_val);
    floodFill8(x,  y-1,new_val,old_val);
    floodFill8(x,  y+1,new_val,old_val);
    floodFill8(x-1,y-1,new_val,old_val);
    floodFill8(x-1,y+1,new_val,old_val);
    floodFill8(x+1,y-1,new_val,old_val);
    floodFill8(x+1,y+1,new_val,old_val);
  }
}
```

Popunjavanje oivičene oblasti

- Jednostavan rekurzivni algoritam za
  - 4-susednu oblast
  - definisanu granicom (boundary_val)
  - koji postavlja celu unutrašnjost na new_val

```
void boundaryFill4(int x, int y, int new_val, int boundary_val)
{
      if ((getPixel(x,y)!= boundary_val) &&
          (getPixel(x,y)!= new_val))
      {
        setPixel(x,y,new_val);
        boundaryFill4(x-1,   y, new_val, boundary_val);
        boundaryFill4(x+1,   y, new_val, boundary_val);
        boundaryFill4(x  , y-1, new_val, boundary_val);
        boundaryFill4(x  , y+1, new_val, boundary_val)
      }
}
```

Iterativni algoritam(1)

- Uvodi se kružni bafer queue veličine Qmax tačaka
  - indeksi qfirst (ukazuje na prvu tačku u baferu) i qlast (ukazuje na poslednju tačku u baferu)
  - procedure za pristup baferu insert(stavljanje tačke) i remove(uzimanje tačke) iz bafera
- Za 4-susednu oblast definisanu unutrašnjom vrednošću old_val:

```
const int Qmax = 800;
Point queue[Qmax];
int qfirst,qlast;

void insert(int x, int y)
{
    qlast = (qlast + 1) % Qmax;
    queue[qlast].x = x;
    queue[qlast].y = y;
}

void remove( int &x, int &y)
{
    x = queue[qfirst].x;
    y = queue[qfirst].y;
    qfirst = (qfirst + 1) % Qmax;
}
```

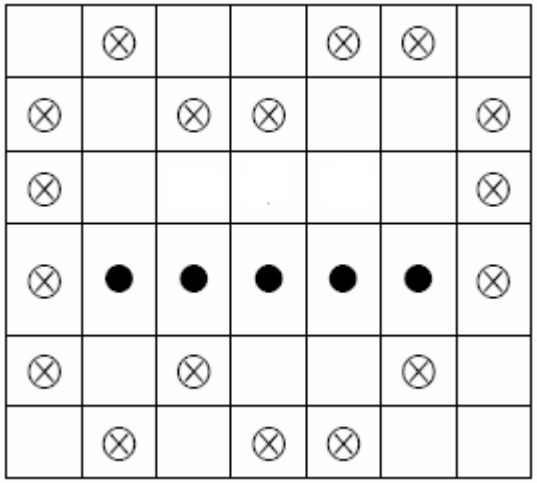
Iterativni algoritam(2)

```
void setPixQ( int x, int y, int new_val, int old_val ){
  if ( getPixel(x,y)== old_val )  {
    setPixel(x,y,new_val);
    insert(x,y);
  }
}
void forestFireFill4(int x, int y, int new_val, int old_val){
  qfirst = 0; qlast = Qmax-1; setPixQ(x,y);
  do  {
    remove(x,y);
    setPixQ(x-1,  y);
    setPixQ(x+1,  y);
    setPixQ(x  ,y-1);
    setPixQ(x  ,y+1);
  }
  while( qfirst!=((qlast+1) % Qmax) );//dok qfirst ne
                                      //prestigne qlast
}
```

- U datoj implementaciji nije rešen problem prepunjavanja bafera.
- Procedura ima efekat "šumskog požara" (*forest fire*)
- Manja potrošnja memorije i brži je od rekurzivnog algoritma.

Ubrzan iterativni algoritam

- Algoritam se zasniva na popunjavanju (horizontalnih) redova piksela
- Crta se horizontalna linija kroz datu tačku sve do granica u oba pravca
- Krajnje tačke linije se stavljaju u red čekanja
    - ne stavlja se u red svaka popunjena tačka
- Posledice: smanjuje se red čekanja i procedura se ubrzava



Ubrzan iterativni algoritam

```
const int Qmax = 50;
int qfirst,qlast;

struct linija3V { int leftx; int rightx; int y; };

linija3V queue[Qmax];


void insert( int leftx, int rightx, int y ) {
  qlast = (qlast+1) % Qmax;
  queue[qlast].leftx  =  leftx;
  queue[qlast].rightx =  rightx;
  queue[qlast].y      =  y;
}
void remove( int &fromx, int &tox, int &y ) {
  fromx =  queue[qfirst].leftx;
  tox   =  queue[qfirst].rightx;
  y     =  queue[qfirst].y;
  qfirst =  (qfirst +1) % Qmax; }
```

Ubrzan iterativni algoritam

```
void hLine(int x, int y, int &leftx, int &rightx,
           int old_val, int new_val)
{
  int rx, lx;
  rx = x;
  while ( (getPixel(rx,y)== old_val) && (rx <= Max_X) )
  {
    setPixel(rx,y,new_val);     rx = rx+1;
  }
  lx =  x - 1;
  while ( (getPixel(lx,y) == old_val) && (lx >= 0) )
  {
    setPixel(lx,y,new_val);     lx = lx-1;
  }
  leftx = lx+1; rightx = rx-1;
}

int findOld( int x, int tox, int y, int old_val) {
  int bx;
  bx = x;
  while ((getPixel(bx,y) != old_val) && (bx <= tox)) { bx++; }
  return bx;
}
```

Ubrzan iterativni algoritam

```
void setLine(int fromx, int tox, int y,
             int old_val, int new_val){
  int nextx; int leftx; int rightx;

  nextx= findOld(fromx,tox,y,old_val);
  while ( nextx <= tox )
  {
    hLine(nextx,y,leftx,rightx,old_val,new_val);
    insert(leftx,rightx,y);
    nextx = findOld(nextx,tox,y,old_val);
  }
}

void speedFill( int x, int y, int old_val, int new_val ){
  int fromx,tox,leftx,rightx;
  qfirst =  0;    qlast  =  Qmax-1;
  hLine(x,y,leftx,rightx,old_val,new_val);
  insert(leftx,rightx,y);
  do {
        remove(fromx,tox,y);
        setLine(fromx,tox,y+1,old_val,new_val)
        setLine(fromx,tox,y-1,old_val,new_val)
  } while ( qfirst != (qlast + 1) % Qmax); }
```

Flood fill 1/3

```
#pragma comment( lib, "opengl32.lib")
#pragma comment( lib, "glu32.lib")
#pragma comment( lib, "glut32.lib")

#include <GL/glut.h>

float yellow[3] = {1.0,1.0,0.0};

float red[3]    = {1.0,0.0,0.0};

float blue[3]   = {0.0,0.0,1.0};

void setPixel(int x, int y, float r, float g, float b)
{
     glColor3f( r,g,b );
     glBegin(GL_POINTS);
          glVertex2i(x,y);
     glEnd();
}
```

Flood fill 2/3

```
void floodFill4(int x, int y,
                float rnew, float gnew, float bnew,
                float rold, float gold, float bold)
{
  float c[3];
  glReadPixels(x,y,1,1,GL_RGB,GL_FLOAT,&c[0]);

  if ( c[0] == rold && c[1] == gold && c[2] == bold)
  {
    setPixel(x,y,rnew,gnew,bnew);
    floodFill4(x-1,  y, rnew,gnew,bnew, rold,gold,bold);
    floodFill4(x+1,  y, rnew,gnew,bnew, rold,gold,bold);
    floodFill4(x  ,y-1, rnew,gnew,bnew, rold,gold,bold);
    floodFill4(x  ,y+1, rnew,gnew,bnew, rold,gold,bold);
  }
}
```

Flood fill 3/3

```
void displayCB(void)
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3fv(yellow);
  glBegin(GL_POLYGON);
       glVertex2i(10,10);
       glVertex2i(10,100);
       glVertex2i(70,100);
       glVertex2i(70,10);
  glEnd();
  floodFill4(50,60,
             red[0],red[1],red[2],
             yellow[0],yellow[1],yellow[2]);
  glFlush();
}

void keyCB(unsigned char key, int x, int y)
{
  if( key == 'q' ) exit(0);
}
```

Flood fill 3/3

```
int main(int argc, char *argv[])
{

  int win;

  glutInit(&argc, argv);

  glutInitDisplayMode(GLUT_RGB);
  glutInitWindowSize(500,500);
  win = glutCreateWindow("Flood fill");

  glClearColor(0.0,0.0,0.0,0.0);
  gluOrtho2D(0,500,0,500);

  glutDisplayFunc(displayCB);
  glutKeyboardFunc(keyCB);
  glutMainLoop();

  return 0;
}
```

Odsecanje

- Odsecanje ili isecanje(*Clipping*)
  - uklanjanje svih delova primitiva izvan prozora kroz koji se slika posmatra
- Pokrivanje (*Covering*, *Shielding*)
  - uklanjanje delova primitiva unutar prozora

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

Odsecanje

- Najčešće se koriste sledeći 2D algoritmi odsecanja:
  - odsecanje tačaka
  - odsecanje linija (pravolnijskih segmenata)
  - odsecanje zatvorenih kontura (poligona)
  - odsecanje krivulja
  - odsecanje teksta
- Prozor odsecanja se definiše koordinatama krajnjih dijagonalnih tačaka prozora:

$$(X_{min}, Y_{min}) \text{ i } (X_{max}, Y_{max})$$

- Tako definisane koordinate odgovaraju normalizovanom kvadratu, gde se vrednosti koordinata x i y kreću od 0 do 1 ili od -1 do 1
- Ako su x koordinate prozora odsecanja Xmin i Xmax, a y koordinate Ymin i Ymax, onda sledeće nejednakosti moraju biti zadovoljene da bi tačka (X,Y) bila unutar prozora odsecanja (ako ivice pripadaju prozoru odsecanja):

```
Xmin <= X <= Xmax

Ymin <= Y <= Ymax
```

- Ako barem jedna od ove 4 nejednakosti nije zadovoljena, tačka je izvan prozora odsecanja.

Selektivno postavljanje piksela

- Algoritam rešava problem odsecanja proizvoljne tačke izvan prozora odsecanja
- Umesto setPixel, primitive se iscrtavaju pozivima setPixelInRect
- Ako je prozor odsecanja pravougaoni, ivice paralelne osama, a pripadaju prozoru:

```
struct Window {
  int left; int right; int bottom; int top;
};

void setPixelInRect( Window w, Point p, int v)
{
  if ( (w.left <= p.x) && ( p.x <= w.right) &&
       (w.top  >= p.y) && ( p.y >= w.bottom) )
     setPixel(p.x,p.y,v);
}
```

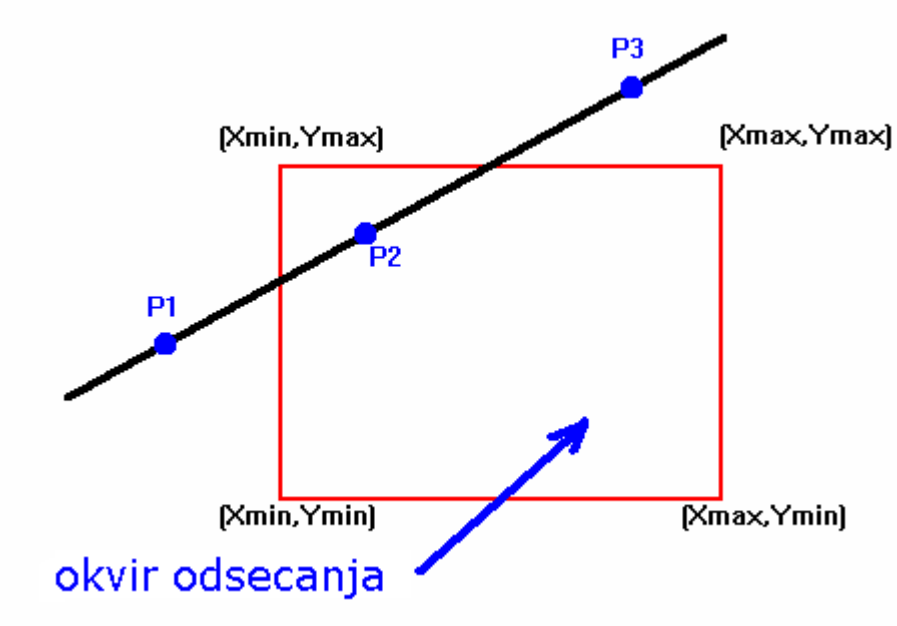
- Ako je prozor proizvoljan poligon:

```
void setPixelInPolygon(Polygon py, Point p , int v)
{ if inside(p,py) setPixel(p.x,p.y,v); }
```

- Metod je neefikasan jer se za svaki piksel vrši ispitivanje

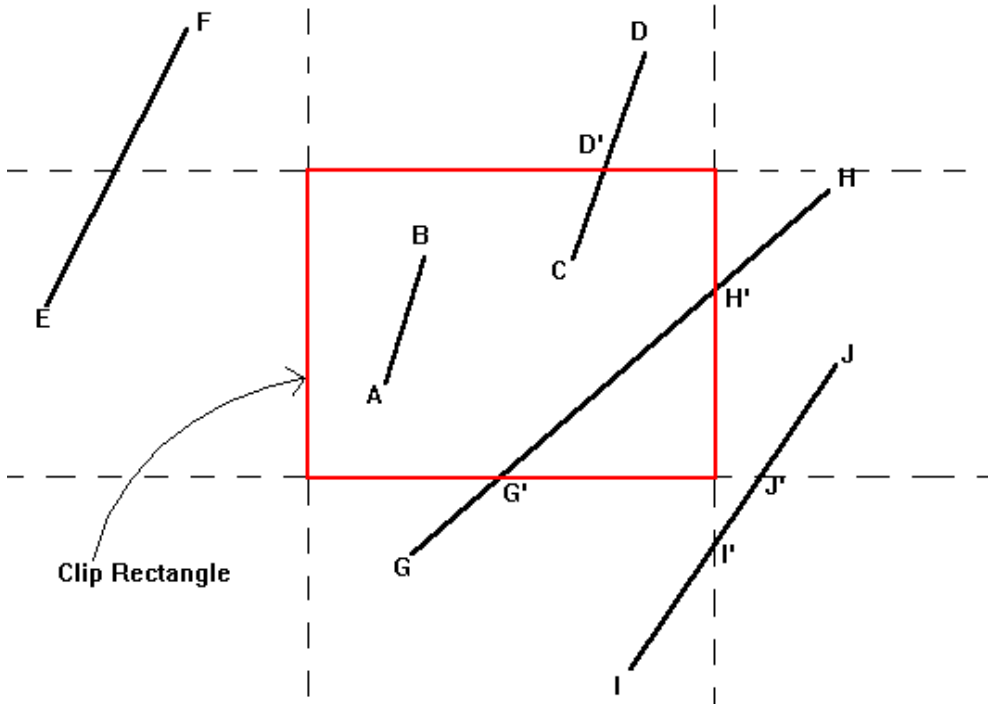
Odsecanje linije

- Kod odsecana linije, ne posmatraju se sve tačke koje joj pripadaju, nego samo krajnje tačke.
- Ako su obe krajnje tačke unutar prozora isecanja, cela linija je vidljiva (trivijalno).
- Ako je jedna krajnja tačka unutar a druga izvan prozora odsecanja, onda linija seče prozor i treba izračunati tačku preseka.
- Ako su obe krajnje tačke izvan prozora, potrebni su dodatni proračuni da bi se utvrdila da li je deo linije vidljiv.



Odsecanje linije

- Za netrivijalne slučajeve razvijeni su algoritmi za odsecanje linije:
  - Algoritam Cohen-Sutherland
  - Sutherland-Hodgman-ov algoritam
  - Weiler-Atherton-ov algoritam
  - Algoritam Liang-Barsky



Cohen-Sutherland algoritam

- Algoritam selSetPixelInRect primenjen na liniju je podjednako spor:
  - kada se radi o linijama koje se delimično odbacuju
  - kada se radi o linijama koje se u celini prihvataju ili odbacuju
- Cohen-Sutherland-ov algoritam efikasno rešava problem odsecanja linija izvan pravougaonog prozora
- Osnovna ideja algoritma
  - da se prvo pokuša da se linija u celini:
    - prihvati

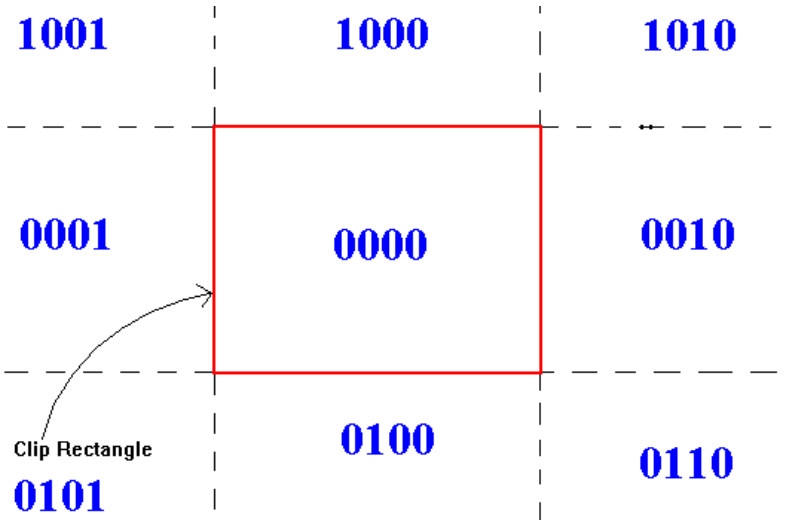
    ili
    - odbaci
  - ukoliko se ne uspe i prvom koraku, određuje se presek linije i produžene ivice prozora
  - ponovo se pokušava prihvatanje ili odbacivanje preostalog dela linije

Algoritam Cohen-Sutherland

1. Parovi krajnjih tačaka se prvo provere da li se trivijalno odbacuju ili prihvataju pomoću binarnih regiona
2. Ako se tako ne može utvrditi vidljivost linije, linija se deli na dva segmenta po ivici prozora
3. Iterativno se testiraju segmenti dok se ne dođe do segmenta koji je ceo vidljiv ili se trivijalno odbacuje.

- Trivijalna rešenja se dobiju logičkim operacijama sa bitovima koji predstavljaju regione.
  - Ako su obe krajnje tačke vidljive (OR krajnjih tačaka == 0000): trivijalno vidljivo.
  - Ako su obe krajnje tačke u istom delu, koji je izvan prozora (AND krajnjih tačaka != 0000): trivijalno nevidljivo.
- Ako su obe krajnje tačke u različitim regionima, algoritam traži jednu od tačaka koja je izvan prozora.
    - Zatim se izračunava tačka preseka linije sa pravcem koji prolazi kroz ivicu prozora
    - Tako izračunata tačka preseka se uzima kao nova krajnja tačka.
- Algoritam se ponavlja dok se ne dođe do trivijalnog rešenja.

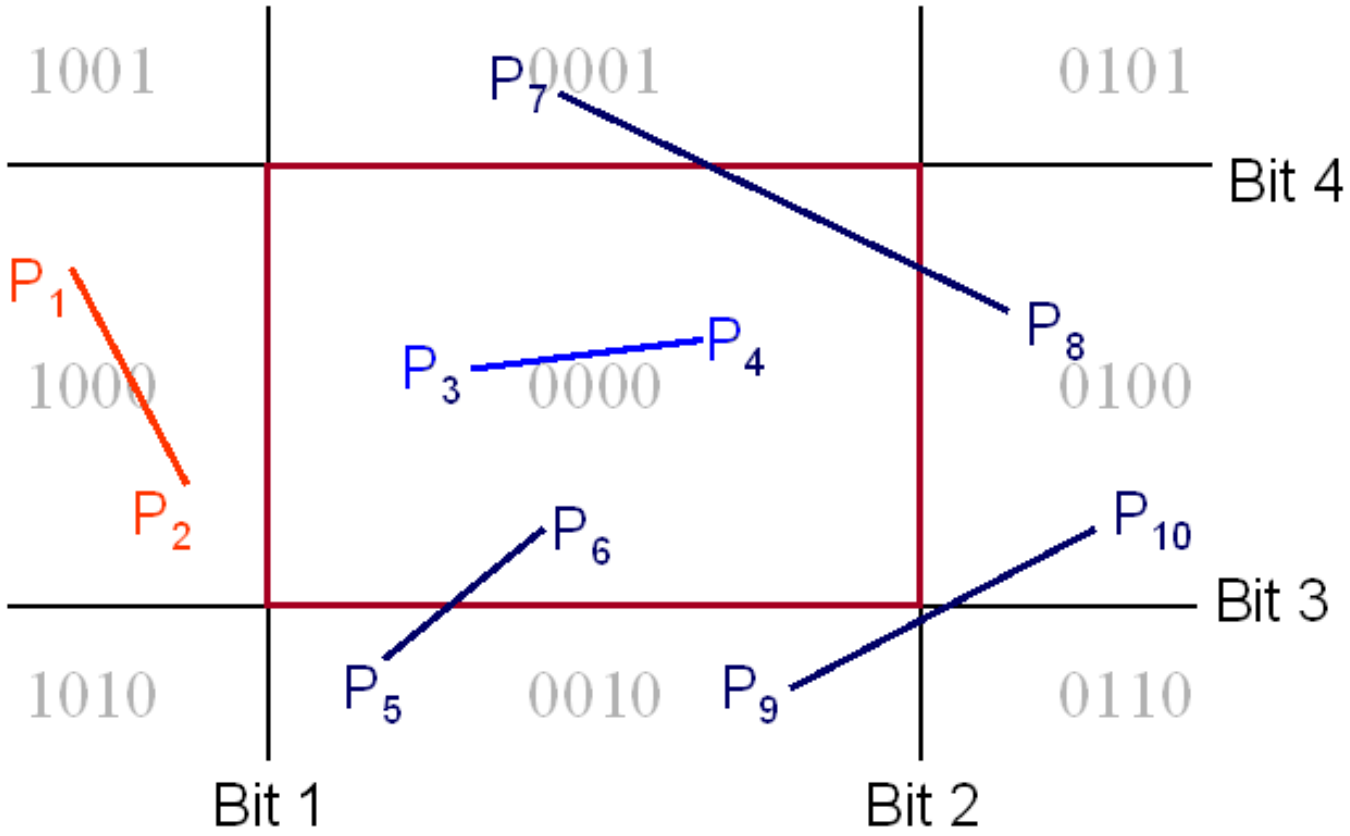
Algoritam Cohen-Sutherland



- Binarni regioni ABCD = goredoledesnoLevo
  - Bit 3: iznad gornje ivice Y > Ymax
  - Bit 2: ispod donje ivice Y < Ymin
  - Bit 1: desno od desne ivice X > Xmax
  - Bit 0: levo od leve ivice X < Xmin

Primer

- Izvršiti klasifikaciju linija koristeći AND (nevidljivo) i OR (vidljivo).



Primer

- Vidljivo: OR == 0000
- Nevidljivo: AND != 0000

- Linija P1P2

  1000 ili 1000 = 1000 – ?

  1000 i 1000 = 1000 – nevidljivo

- Linija P3P4

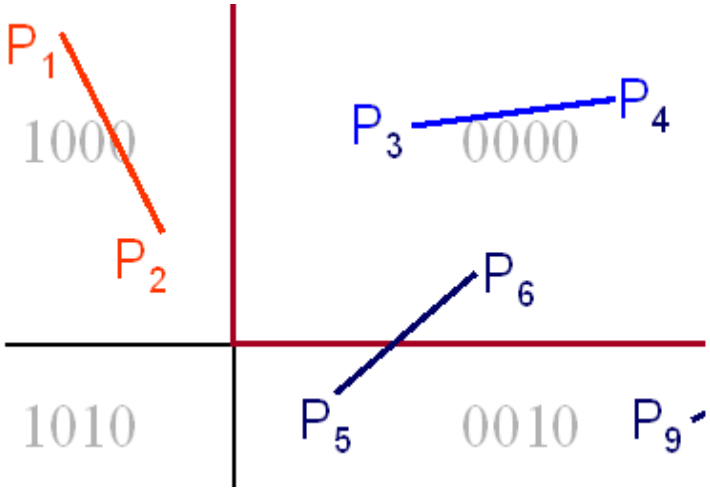
  0000 ili 0000 = 0000 – vidljivo

  0000 i 0000 = 0000 - ?

- Linija P5P6

  0010 ili 0000 = 0010 – ?

  0010 i 0000 = 0000 – ?



Algoritam (1)

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

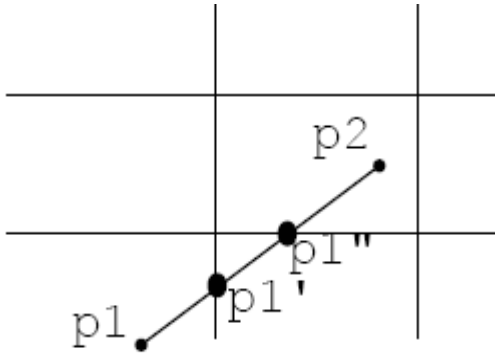
1001	1000	1010
0001	0000	0010
0101	0100	0110

2. Svakoj oblasti se pridružuje 4-bitni položajni kod (o*utcode*): $b_3b_2b_1b_0$-gde svaki bit označava jednu oblast:
   - b_3-iznad
   - b_2-ispod
   - b_1-desno
   - b_0-levo
3. Za krajnje tačke linije p1 i p2 određuju se položajni kodovi c1 i c2
4. Vrši se ispitivanje:
   - a) Ako su i c1 i c2 jednaki 0 tada je (c1|c2)==0 →linija se trivijalno prihvata
   - B) Ako c1 i c2 imaju barem 1 zajednički bit (c1&c2)!=0→linija se cela odbacuje

Napomena: ovim se ne odbacuju sve linije koje su u celini van prozora (npr. c1==0100, c2==0010, a nema tačke na duži C1C2 čiji je kod 0000)

Algoritam (2)

5. 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
6. Krajnja tačka linije se premešta u presečnu tačku (P1→P1')
7. Ide se na korak 3



- Presečna tačka P se određuje na sledeći način (u slučaju da linija seče levu ivicu prozora):

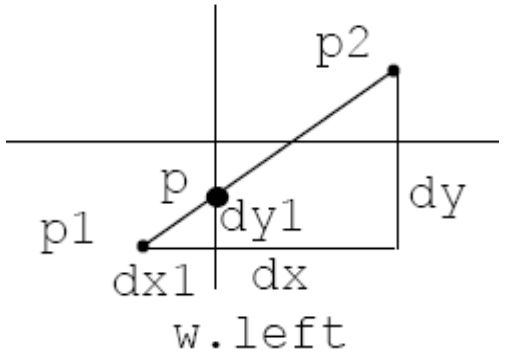
  p(w.left, p1.y+dy1)

  dy=p2.y-p1.y

  dx=p2.x-p1.x

  dx1=w.left-p1.x

  dy1:dx1=dy:dx => dy1=(dy/dx)dx1



Kod (1)

- Definišu se položajni kodovi:

```
const int LeftCode=1, RightCode=2, BottomCode=4, TopCode=8;
```

- Definiše se tip ivice prozora:

```
enum WindowEdge {LeftEdge, RightEdge, BottomEdge, TopEdge};
```

- Uvodi se pomoćna funkcija outCode
  - vraća položajni kod tačke p u odnosu na prozor w:

```
int outCode(Window w, Point p)
{
  int code;
  code = 0;
  if      (p.x > w.right ) code = RightCode;
  else if (p.x < w.left  ) code = LeftCode;
  if      (p.y > w.top   ) code = code + TopCode;
  else if (p.y < w.bottom) code = code + BottomCode;
  return code;
}
```

Kod (2)

- Procedura wCross određuje presek *p* linije ***l*** sa ivicom ***e*** prozora ***w***:

```
void wCross( Window w, WindowEdge e, Line l, Point &p )
{
  float dx, dy, dx1, dy1;
  dx = l.b.x - l.a.x;
  dy = l.b.y - l.a.y;
  switch( e )
  {
    case LeftEdge:   p.x = w.left;     dx1 = p.x-l.a.x;
                     p.y = l.a.y + (int)(dy*dx1/dx);      break;
    case RightEdge:  p.x = w.right;    dx1 = p.x-l.a.x;
                     p.y = l.a.y + (int)(dy*dx1/dx);      break;
    case BottomEdge: p.y = w.bottom;   dy1 = p.y -l.a.y;
                     p.x = l.a.x + (int)(dx*dy1/dy);      break;
    case TopEdge:    p.y = w.top;      dy1 = p.y -l.a.y;
                     p.x = l.a.x + (int)(dx*dy1/dy);      break;
  }
}
```

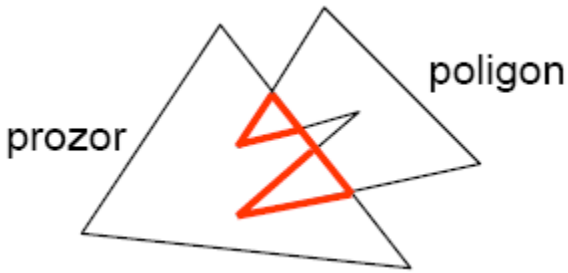
Kod (3)

- Procedura *clip* odseca delove linije *l* izvan prozora *w* i obaveštava da li se linija (makar i samo njen deo) prihvata (accept):

```
bool clip(Window w, Line &l){
  int c1, c2, c;
  Point p;
  bool accept = true;
  c1 = outCode(w, l.a);
  c2 = outCode(w, l.b);
  while ( ((c1!=0) || (c2!=0)) && accept ) //bar jedna tacka je van
  {
    if ((c1 & c2) != 0) accept = false; //linija van prozora
    else
    {
      if (c1 != 0)  c = c1;
      else          c = c2;
      if ((c & LeftCode) != 0)          wCross(w, LeftEdge,  l, p);
        else if ((c & RightCode) != 0)  wCross(w, RightEdge, l, p);
          else if ((c & TopCode) != 0)  wCross(w, TopEdge,   l, p);
            else                        wCross(w, BottomEdge,l, p);
      if (c == c1){
        l.a = p; c1 = outCode(w, l.a);
      }
      else{
        l.b = p; c2 = outCode(w, l.b);
      }
    }
  } return accept;}
```

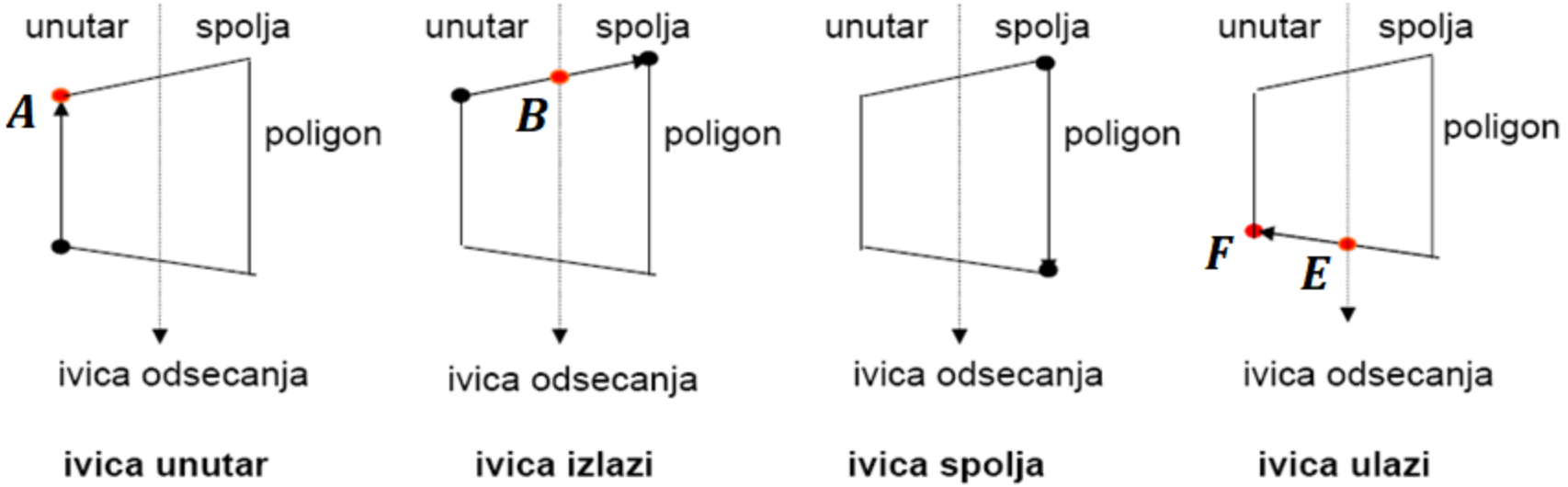
Sutherland-Hodgman-ov algoritam

- Algoritam rešava problem odsecanja proizvoljnog poligona izvan prozora, koji je konveksan poligon
- Strategija "podeli-pa-vladaj":
  - rešavaju se elementarni problemi čija kombinacija rešava ceo problem
- Elementaran problem:
  - odsecanje poligona u odnosu na jednu ivicu za odsecanje
- Ivica za odsecanje je prava – produžena ivica prozora
- Sukcesivno odsecanje svim ivicama za odsecanje daje konačni rezultat
- Rezultat predstavljaju delovi poligona koji pripadaju prozoru, zatvoreni ivicama prozora
  - za razliku od rezultata koji bi se dobio primenom Cohen-Sutherland algoritma (ukoliko bi prozor bio pravougaoni)
- Problem: spajanje rezultantnih poligona



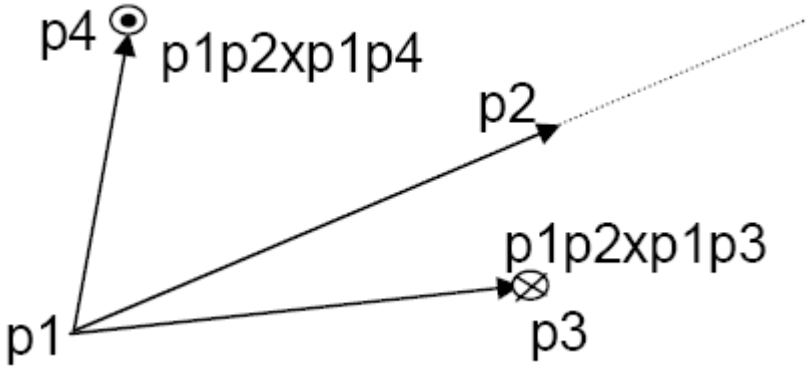
Algoritam (1)

- Polazi se od niza temena (vrhova) poligona
  $v_1, v_2, \ldots v_n$;
  - ivice poligona su od v_i do v_{i+1}, za i=1,…,n-1 i poslednja od v_n do v_1
- Odsecanje prema ivici ***i*** generiše drugu (izlaznu) seriju temena koja definišu novi poligion
- Kreće se od ivice (v_n,v_1), pa do (v_{n-1},v_n)
- Ispituje se odnos između susednih temena i ivice odsecanja: 0 , 1 ili 2 temena se dodaju u izlaznu listu
- Postoje 4 slučaja koji se analiziraju (u izlaznu listu dodaju se: ***A, B, E i F***)



Algoritam (3)

- Prozor je orijentisan u smeru kazaljke na satu, tako da je unutrašnjost definisana kao desna strana ivice odsecanja.
- Test unutrašnjosti se zasniva na vektorskom proizvodu vektora ivice odsecanja ***v*** i vektora ***w*** od početne tačke ivice odsecanja do ispitivane tačke
- Ako su oba vektora u XoY ravni desnog koordinatnog sistema:
  - ako je vektor proizvoda u smeru pozitivne Z ose tada je posmatrana tačka spolja (p4)
  - ako je vektor proizvoda u smeru negativne Z ose tada je posmatrana tačka unutar (p3)



- Tačka je unutar ako je magnituda (z komponenta) vektorskog proizvoda negativna
- Vektorski proizvod dva vektora **v**x**w** u XY ravni ima magnitudu : $v_x w_y - v_y w_x$
- Tačka p je sa unutrašnje strane ivice odsecanja p1p2 ako je:

$$(p2.x-p1.x)(p.y-p1.y) - (p2.y-p1.y)(p.x-p1.x)<0$$

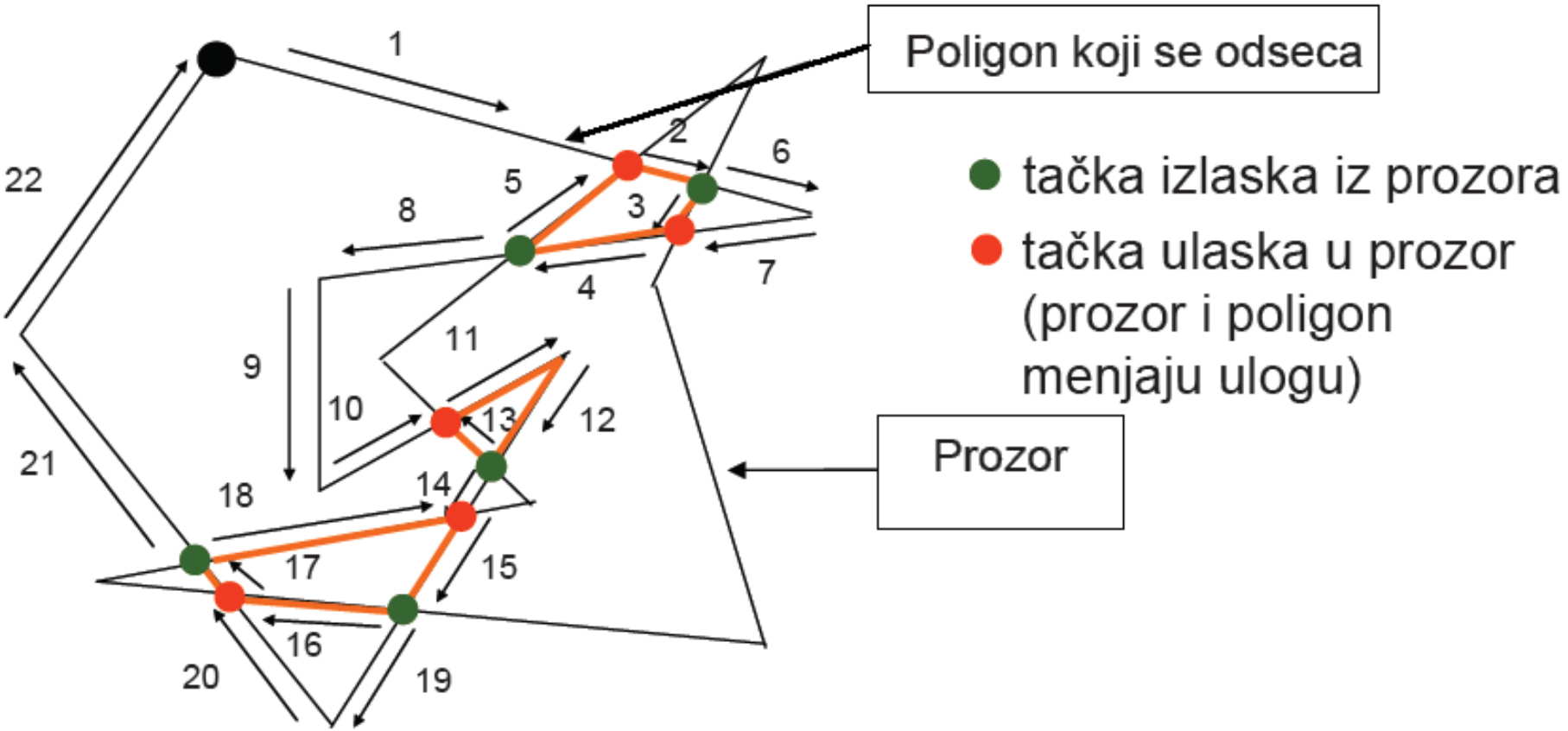
$$\det \begin{vmatrix} x & y & z \\ v_x & v_y & v_z \\ w_x & w_y & w_z \end{vmatrix}$$

Weiler-Atherton-ov algoritam

- Algoritam rešava problem odsecanja
  - proizvoljnog poligona izvan prozora gde je prozor proizvoljan poligon
- Praktično, određuje se presek dva proizvoljna poligona
- Oba poligona mogu biti i konkavna, a takođe, mogu sadržavati i rupe
- Algoritam
  - kreće se 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 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 jedan put

Weiler-Atherton-ov algoritam

- Tačke izlaska iz prozora se pamti da bi se od nje nastavilo posle zatvaranja putanje
- Tačke ulaska se pamte da bi se u njima zatvorile putanje (da se izbegnu beskonačne petlje)



Algoritam Liang-Barsky

- Algoritam Liang-Barsky koristi parametarsku jednačinu linije i nejednakosti koje opisuju granice prozora odsecanja da bi se odredili preseci između linije i prozora odsecanja.
- Pomoću tih preseka se može znati koji deo linije je vidljiv
- Algoritam je efikasniji od algoritma Cohen-Sutherland.

- Koraci
  1. Predstaviti segmente linije u parametarskoj formi
  2. Izvesti jednačine za testiranje da li je tačka unutar prozora
  3. Izračunati nove vrednosti parametara za vidljivi deo segmenta, ako takav postoji
  4. Prikazati vidljivi deo segmenta linije

Algoritam Liang-Barsky

- Prikaz segmenta linije u parametarskom obliku:

$x = x_1 + (x_2 - x_1) * u = x_1 + dx * u, \quad 0.0 < u < 1.0$

$y = y_1 + (y_2 - y_1) * u = y_1 + dy * u, \quad 0.0 < u < 1.0$

  - za $u = 0.0 \Rightarrow (x_1, y_1)$
  - za $u = 1.0 \Rightarrow (x_2, y_2)$

sada je potrebno zadovoljiti nejednačine:

$$Xmin <= x_1 + dx*u <= Xmax$$

$$Ymin <= y_1 + dy*u <= Ymax$$

- Prethodno se može napisati u obliku:

$$p_k * u <= q_k, \quad k = 1, 2, 3, 4$$

gdje je:

$p_1 = -dx$	$q_1 = -(Xmin - x_1)$	Levo
$p_2 = dx$	$q_2 = Xmax - x_1$	Desno
$p_3 = -dy$	$q_3 = -(Ymin - y_1)$	Dole
$p_4 = dy$	$q_4 = Ymax - y_1$	Gore

Algoritam Liang-Barsky

- Sada su slučajevi:
  1. $p_k=0$, paralelnost sa stranicama
     1. $q_k<0$, nalazi se van granica
     2. $q_k\geq 0$, nalazi se unutra
  2. $p_k!=0$, linija seče "prozor"
     1. $p_k<0$
     2. $p_k>0$

  - Može se izračunati takav *u* da se odredi presek stranice i linije

    $u = q_k/p_k$;
  - Za svaku liniju može se izračunati vrednost parametra u_1, u_2, i odrediti deo linije koji leži unutar prozora

    u_1 za $p_k<0$ $r_k=q_k/p_k$ pri čemu $u_1=\max\{0,r_1,r_2,r_3,r_4\}$

    u_2 za $p_k>0$ $r_k=q_k/p_k$ pri čemu $u_2=\min\{1,r_1,r_2,r_3,r_4\}$

    ako je $u_1>u_2$ linija je van prozora,

    inače linija seče prozor, presečne tačke su određene iz u_1, u_2

Primer 1/6

```
#pragma comment( lib, "opengl32.lib")
#pragma comment( lib, "glu32.lib")
#pragma comment( lib, "glut32.lib")

#include <GL/glut.h>

float yellow[3] = {1.0,1.0,0.0};
float red[3]= {1.0,0.0,0.0};
float blue[3]= {0.0,0.0,1.0};

void ShowLine(double x1, double y1, double x2, double y2)
{
  glBegin(GL_LINES);
    glVertex2i(x1,y1);
    glVertex2i(x2,y2);
    glEnd();
  glFlush();
}

double xwmin=50;
double ywmin=50;
double xwmax=200;
double ywmax=200;
```

Primer 4/6

```
bool LiangBarsky (double Xmin, double Xmax, double Ymin, double Ymax,
double &x1, double &y1, double &x2, double &y2) {
    double u1 = 0.0, u2 = 1.0, dx = x2-x1, dy = y2-y1, p,q,r;
    for(int edge=0; edge<4; edge++){
        switch(edge){
          case 0: p = -dx;    q = -(Xmin - x1); break;
          case 1: p =  dx;    q =  (Xmax - x1); break;
          case 2: p = -dy;    q = -(Ymin - y1); break;
          case 3: p =  dy;    q =  (Ymax - y1); break;
        }
        if(p==0 && q<0) return false;   //pralelna, van
        r = q/p;
        if(p<0){
            if(r>u2) return false;          // van
            else if(r>u1) u1=r;             // odsecanje
        } else if(p>0) {  // da je samo if(p>0) bio bi malo sporiji kod
            if(r<u1) return false;       // van
            else if(r<u2) u2=r;          // odsecanje
        }
    }
    double xtemp = x1 + u1*dx; double ytemp = y1 + u1*dy;
             x2 = x1 + u2*dx;            y2 = y1 + u2*dy;
             x1 = xtemp;                 y1 = ytemp; return true; //OK
}
```

Primer 5/6

```
double x1=10;
double y1=10;
double x2=350;
double y2=220;

void displayCB(void)
{
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3fv(yellow);
  glBegin(GL_POLYGON);
       glVertex2i(xwmin,ywmin);
       glVertex2i(xwmin,ywmax);
       glVertex2i(xwmax,ywmax);
       glVertex2i(xwmax,ywmin);
  glEnd();
  glColor3fv(red);
  double x1t=x1,y1t=y1,x2t=x2,y2t=y2; //ako budete napisali i svoju fLB
  if(LiangBarsky(xwmin,xwmax,ywmin,ywmax,x1t,y1t,x2t,y2t))
    ShowLine(x1t,y1t,x2t,y2t);
  glFlush();
}
```

Primer 6/6

```
void keyCB(unsigned char key, int x, int y)
{
  if( key == 'q' ) exit(0);
  if( key == 's' ) {ShowLine(x1,y1,x2,y2);};
}

int main(int argc, char *argv[])
{
  glutInit(&argc, argv);

  glutInitDisplayMode(GLUT_RGB);
  glutInitWindowSize(500,500);
  int win = glutCreateWindow("Liang-Barski");

  glClearColor(0.0,0.0,1.0,0.0);
  gluOrtho2D(0,500,0,500);
  glutDisplayFunc(displayCB);
  glutKeyboardFunc(keyCB);

  glutMainLoop();
  return 0;
}
```