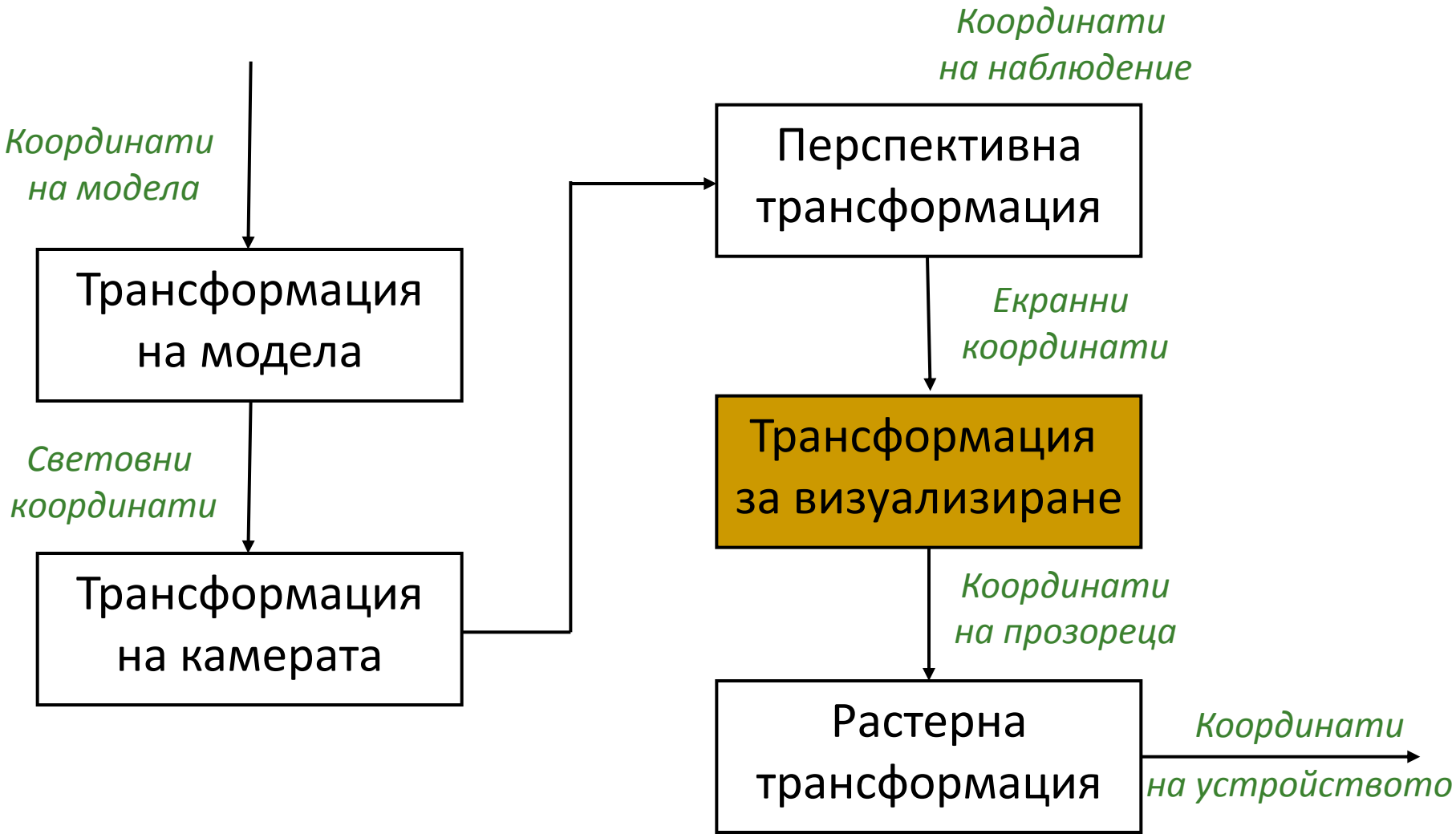
Компютърна графика

Построяване на двумерен изглед

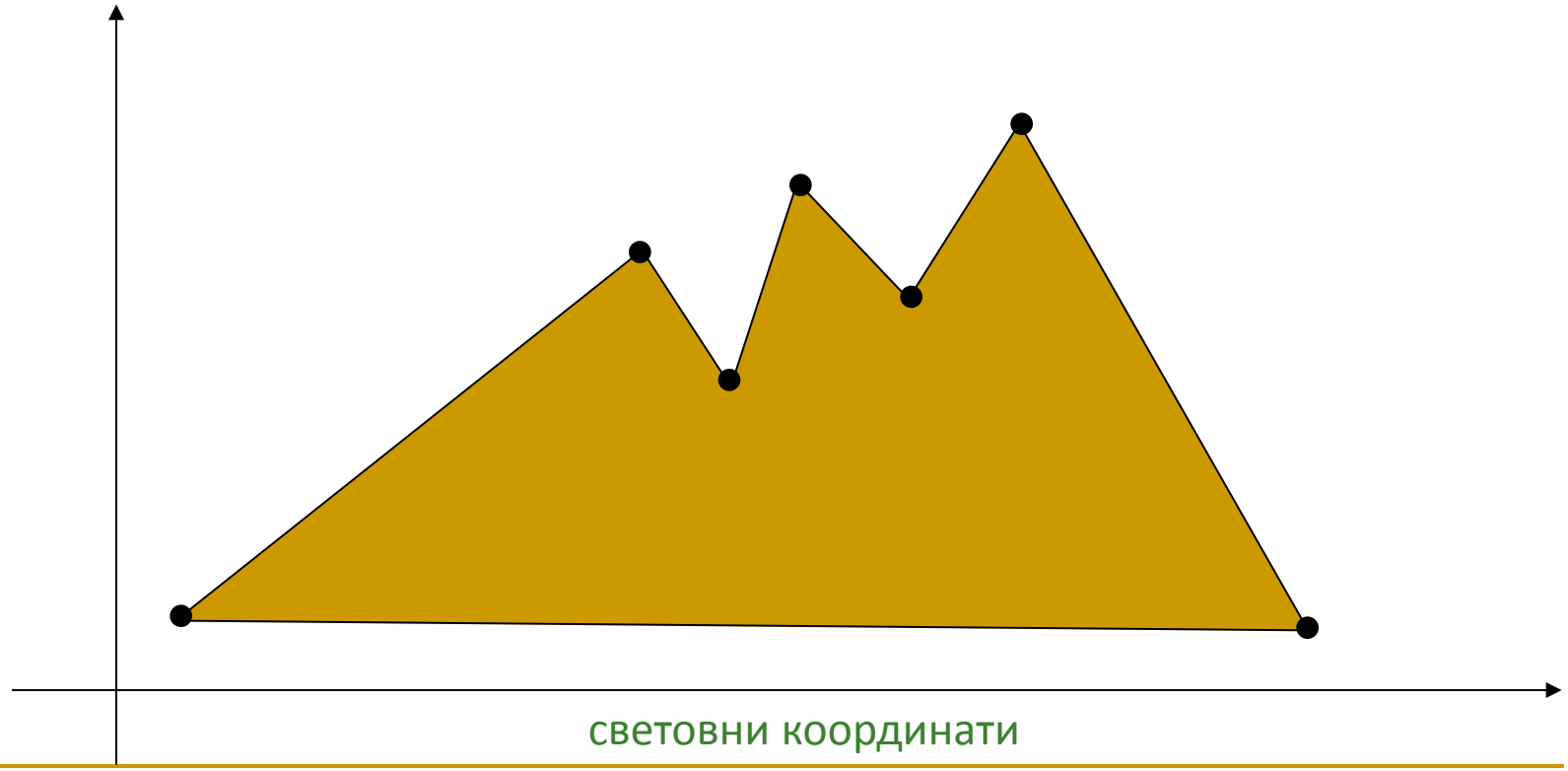
доц. Милена Лазарова, кат. КС, ФКСУ

Основен графичен конвейер



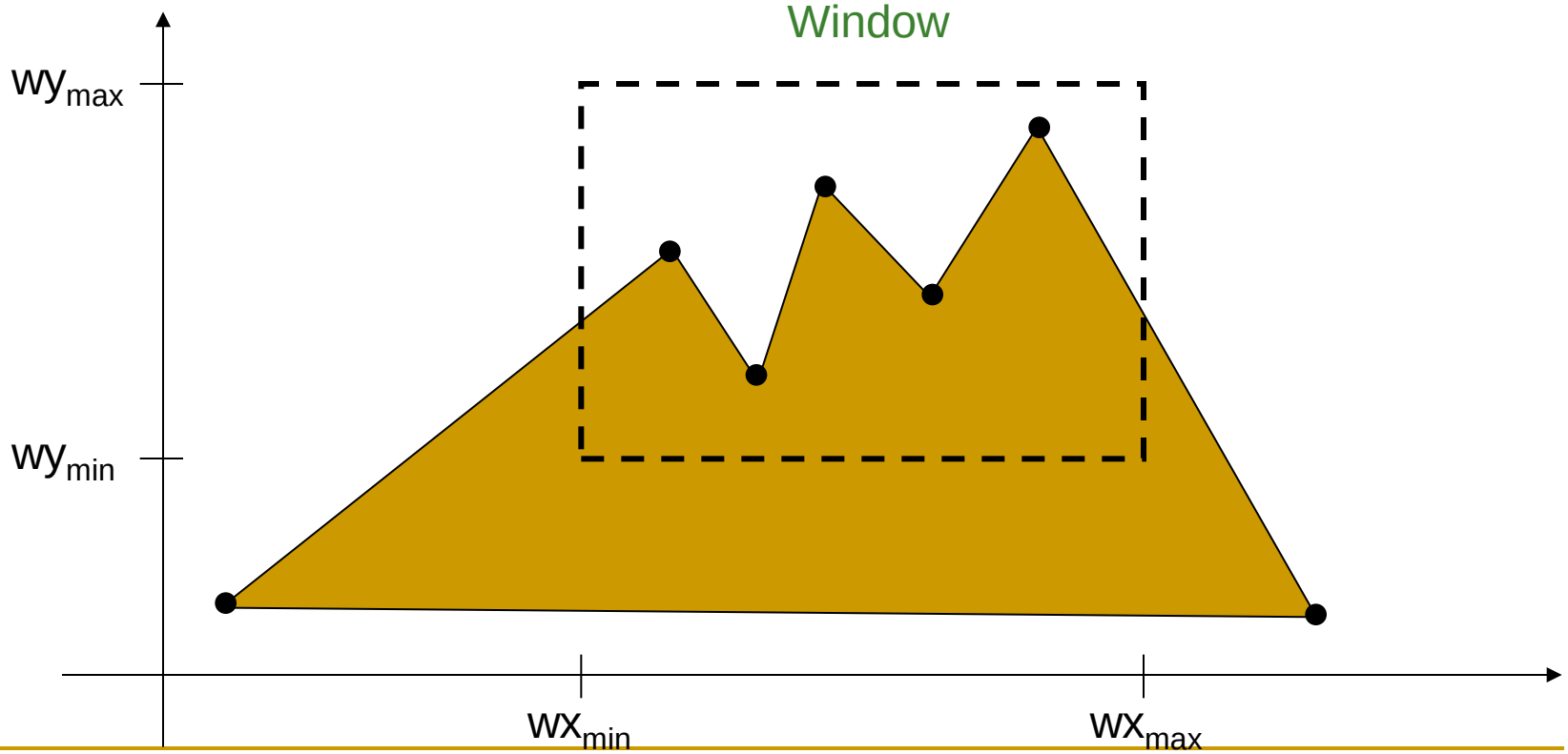
Визуализиране

- Всяка сцена се състои от обекти зададени със световни координати



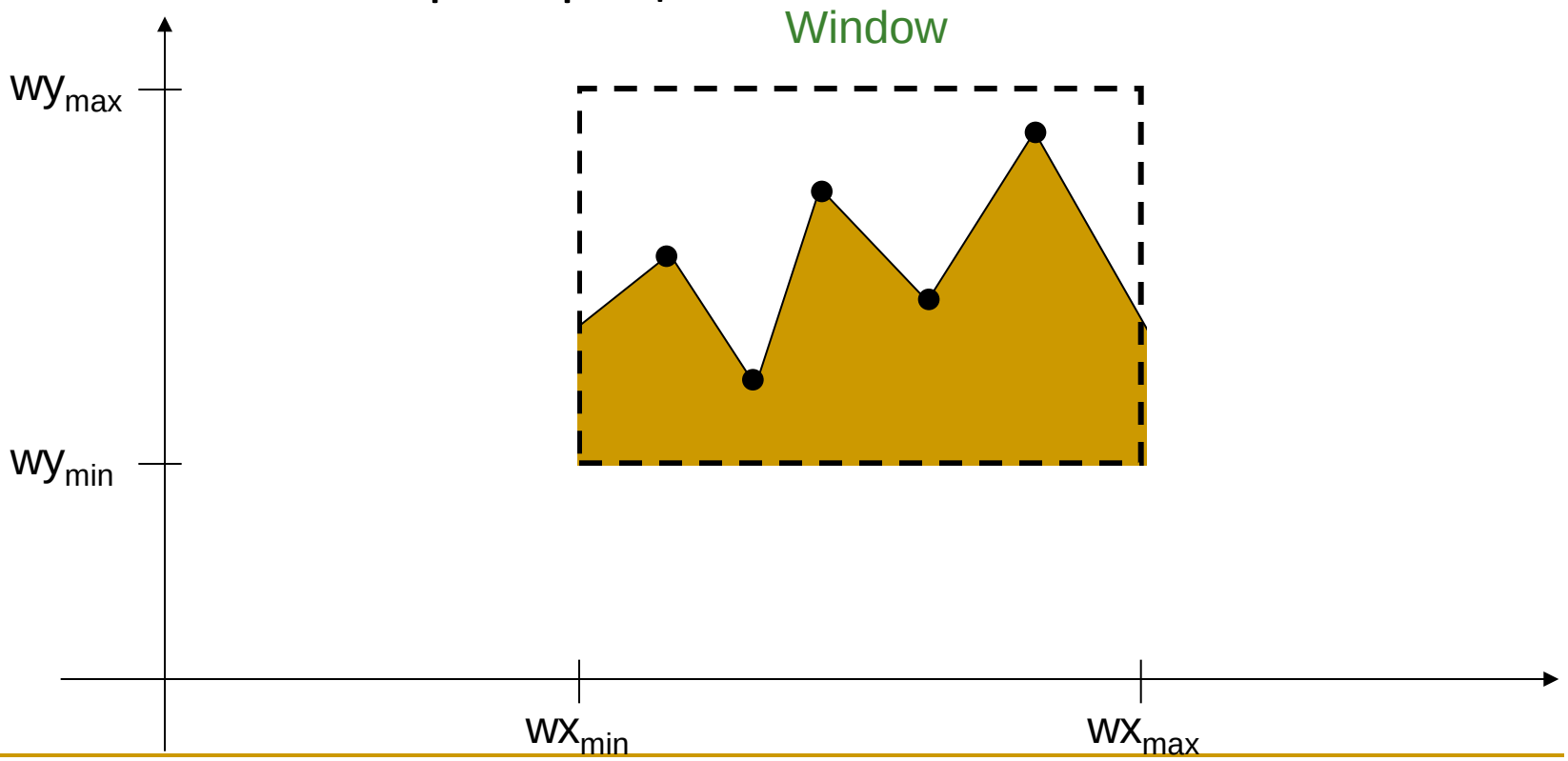
Визуализиране

- При визуализиране на дадена сцена се изобразяват само обектите в определен прозорец



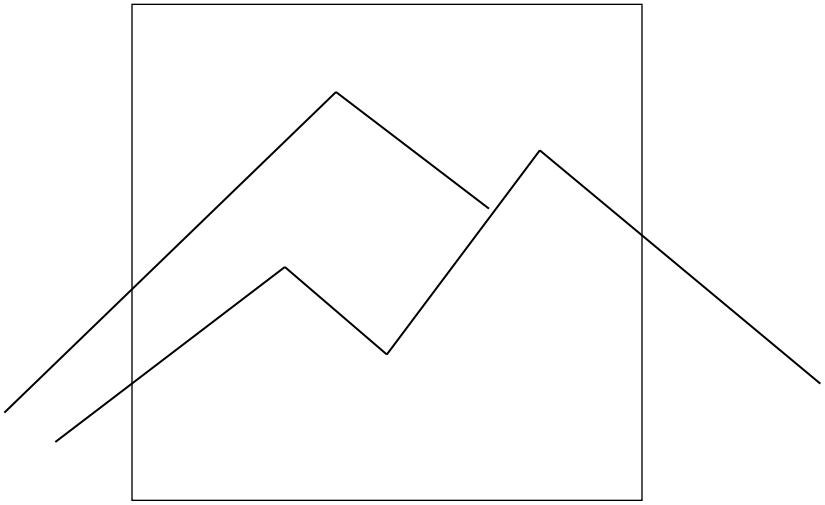
Визуализиране

- За ускоряване на визуализирането се “изрязва” (clip) всичко извън прозореца

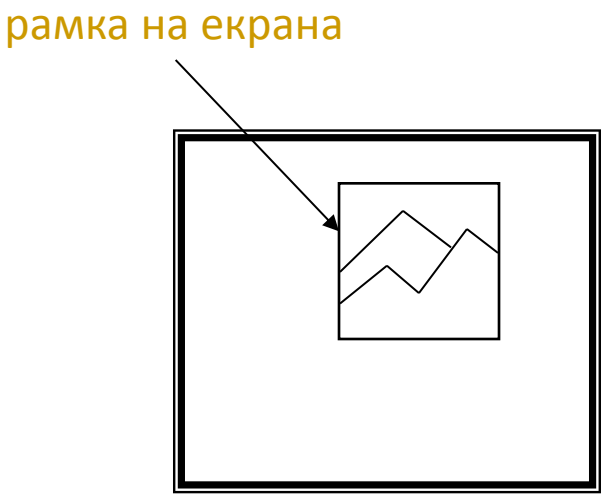


Визуализиране

- Визуализиране (Viewing)
  - изобразяване на обекти от световни координати в екранни координати



световен изглед
(пространство на обектите)



устройство за визуализиране
(пространство на образите)

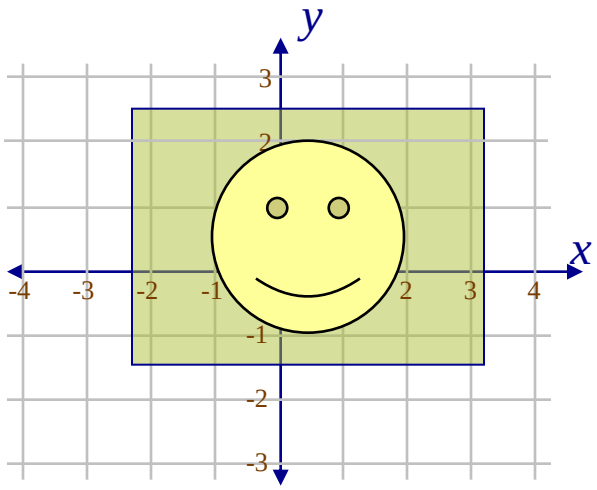
2D визуализиране

- Преобразуване на световния изглед в рамка за визуализиране

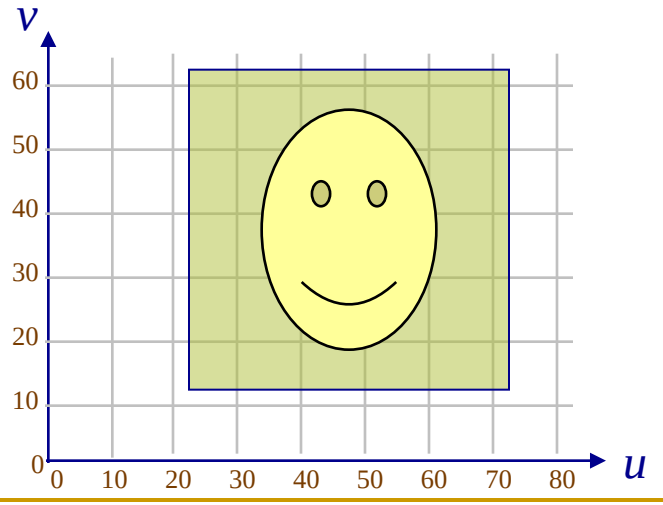
world view → ***screen*** view

- световните координати се трансформират в екранни координати

Window (world)



Viewport (screen)



2D визуализиране

- Допускания
  - координатите на обектите са в пиксели
  - световният прозорец има същата форма и размери, като рамката за визуализиране
  - ако световния прозорец и рамката за визуализиране имат различен коефициент на пропорционалност (aspect ratio) възникват изкривявания
    - aspect ratio = window width / window height

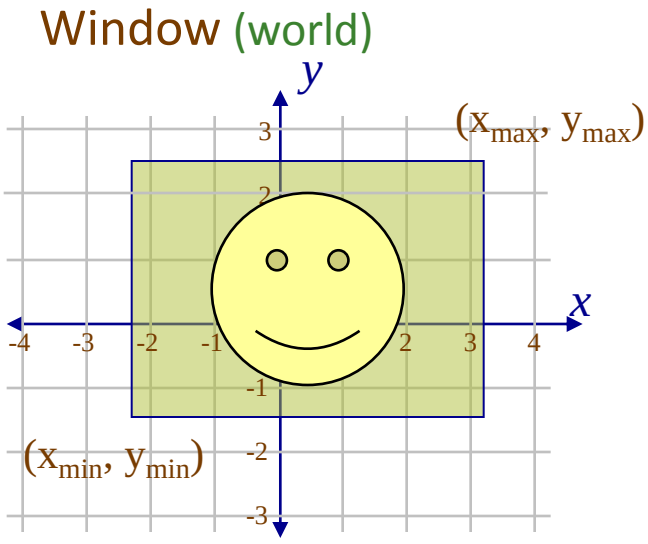
Трансформация Window-to-Viewport

- Позволява да се използват различни координатни системи за ***моделиране*** и ***визуализиране***
- Осигурява независимост от устройствата за визуализиране
  - няма значение дали изходното устройство е голям плотер, принтер или малък екран на дисплея
- Размерите на прозореца на световния изглед и рамката на екрана могат да са различни

- Удобно е тази трансформация да е прозрачна за програмиста

“Window”

- Световен изглед (“world space”, “world coordinates”)
  - областта от света, която се проектира на екрана
  - местоположение, мерни единици, размери и т.н. зависят от конкретното приложение и могат да са специфични за дадена приложна област
    - мерните единици могат да са инчове, сантиметри, футове, метри, километри, нанометри, светлинни години и т.н.
    - обикновено е центриран около началото на координатната система
      - но не е задължително

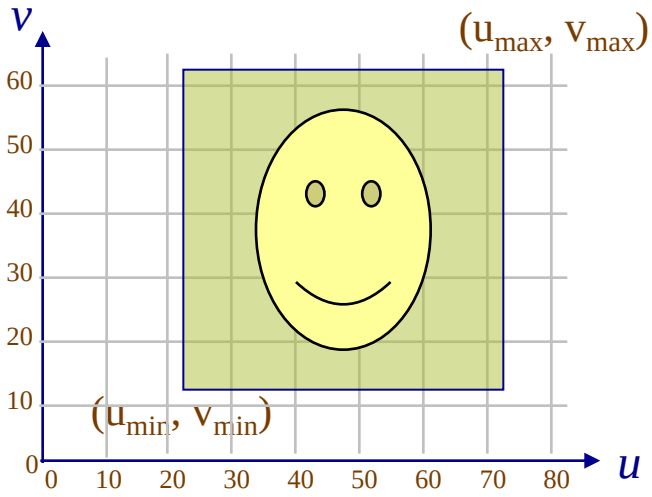


“Viewport”

- Рамка за визуализиране
  - правоъгълник на екрана, в който ще се осъществи визуализирането
    - може да бъде целия екран
      - подразбиращ се случай в OpenGL
    - може да бъде част от екрана
  - задава се с “екранни координати”

- *аналогично на определяне размера на отпечатана снимка (9x13, 10x15)*

Viewport (screen)

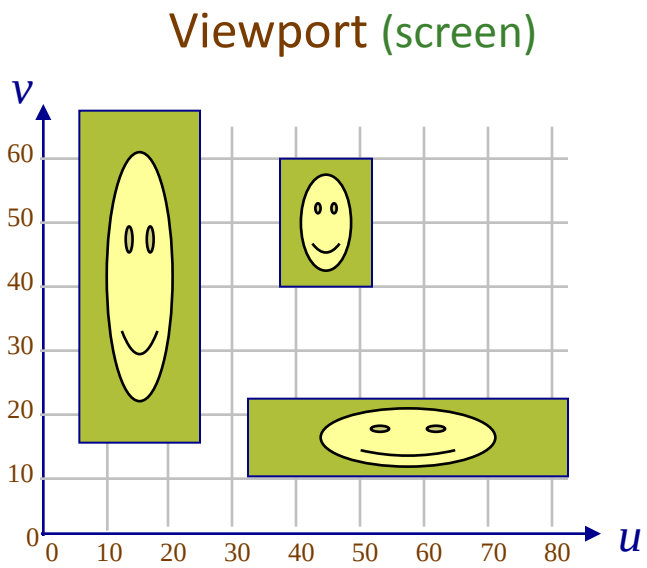


“Viewport”

- Допустимо е да има няколко рамки за визуализиране (multiple viewports)

  могат да съдържат

  - един и същи изглед на даден прозорец
  - различни изгледи на даден прозорец
  - различни изгледи на различни прозорци



2D визуализиране

- **Световни координати** (***world***)
  - представяне на сцена чрез задаване на световни координати на обекти в нея в определена координатна система
- **Световен изглед** (***world window***)
  - част сцената, съдържаща интересуващите ни обекти
- **Прозорец на екрана** (***screen window***)
  - прозорец на екрана, в който се визуализира сцената
- **Рамка за визуализиране** (***viewport***)
  - част от прозореца на екрана, в който се визуализира сцената и обектите в нея

Frame Buffer

Frame Buffer

(например с размери 1280×800)

екранни координати

(координати на пиксели във Frame Buffer)

Window

Frame Buffer

(например с размери 1280×800)

Window

област от екрана, използвана от приложението

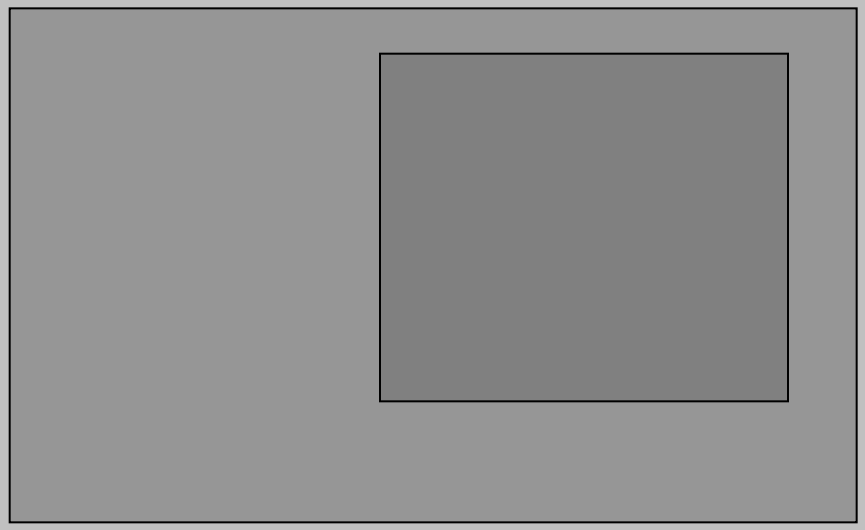
екранни координати

(координати на пиксели във Frame Buffer)

Viewport

Frame Buffer

(например с размери 1280×800)



Viewport

област от прозореца,

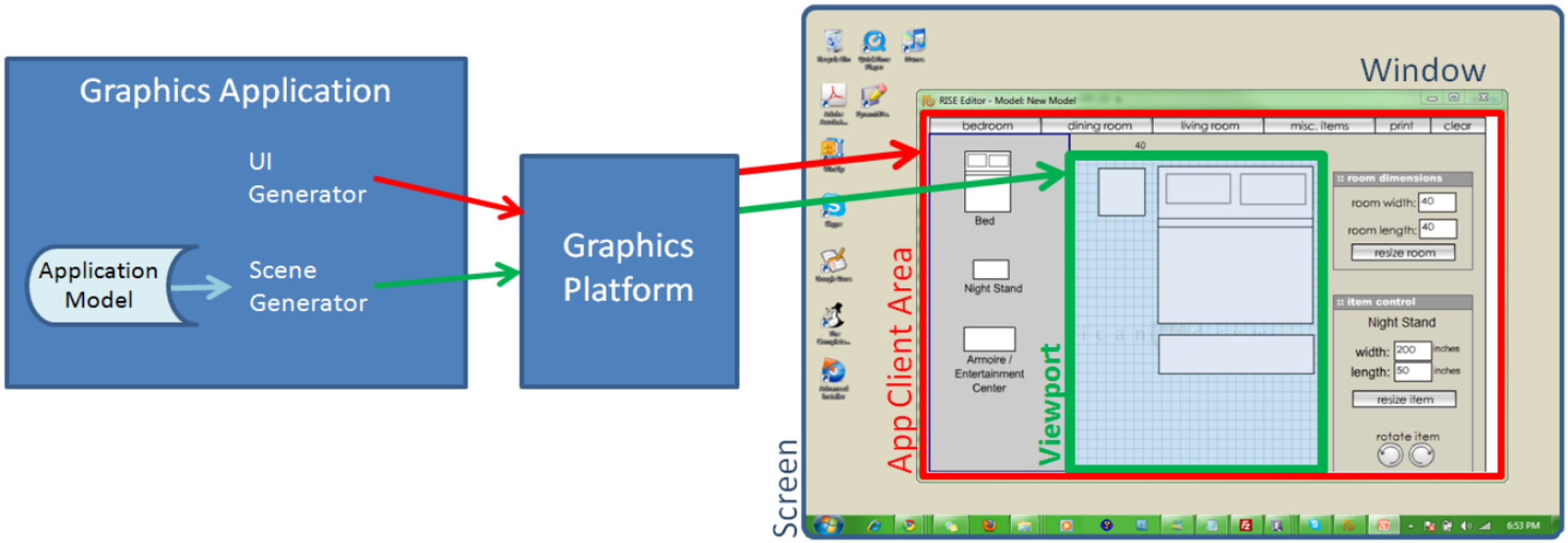
използвана за визуализиране

(например 256×256)

екранни координати

(координати на пиксели във Frame Buffer)

2D визуализиране



2D визуализиране

1) Изграждане на общата сцена чрез моделиращи трансформации над обектите
2) Задаване на прозорец и рамка за визуализиране
3) **Преобразуване от световни координати на прозореца в екранни координати на рамката за визуализиране**
4) Изрязване по прозореца
5) Растеризация на образите в координати на изходното устройство (екранни координати)

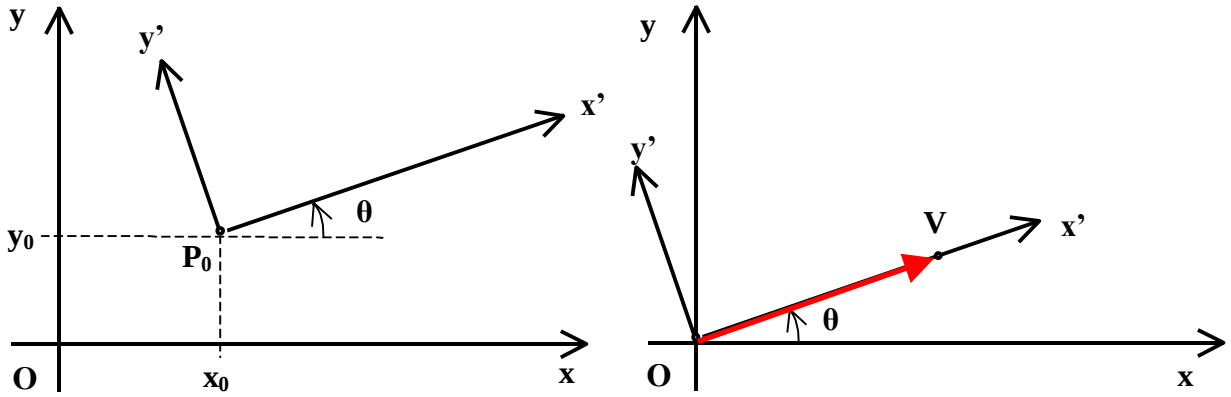
Трансформации между КС

1) Транслация на точка P_0 до точка O с матрица на транслация

$$T(-x_0, -y_0)$$

2) Ротация, така че оста x′ да съвпадне с оста x с матрица на ротация

$$R(-\theta) = \begin{bmatrix} \cos\theta & \sin\theta & 0 \\ -\sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$



Трансформации между КС

- Общата трансформираща матрица е

$$M = R(-\theta).T(-x_0, -y_0)$$

- Определяне на ъгъла на ротация θ
  - може да се зададе явно
  - може да се зададе положителната посока на x′ чрез вектор V (точка от оста x′)

$$\sin\theta = \frac{V_y}{|V|} \qquad \cos\theta = \frac{V_x}{|V|}$$

- Ако се нормализира вектора V
  - единичен вектор $v = \frac{V}{|V|}$
- то $\sin\theta = v_y$

$$\cos\theta = v_x$$

$$\rightarrow \quad R(-\theta) = \begin{bmatrix} v_x & v_y & 0 \\ -v_y & v_x & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Трансформации между КС

- Пример

$$V = (0,1)$$

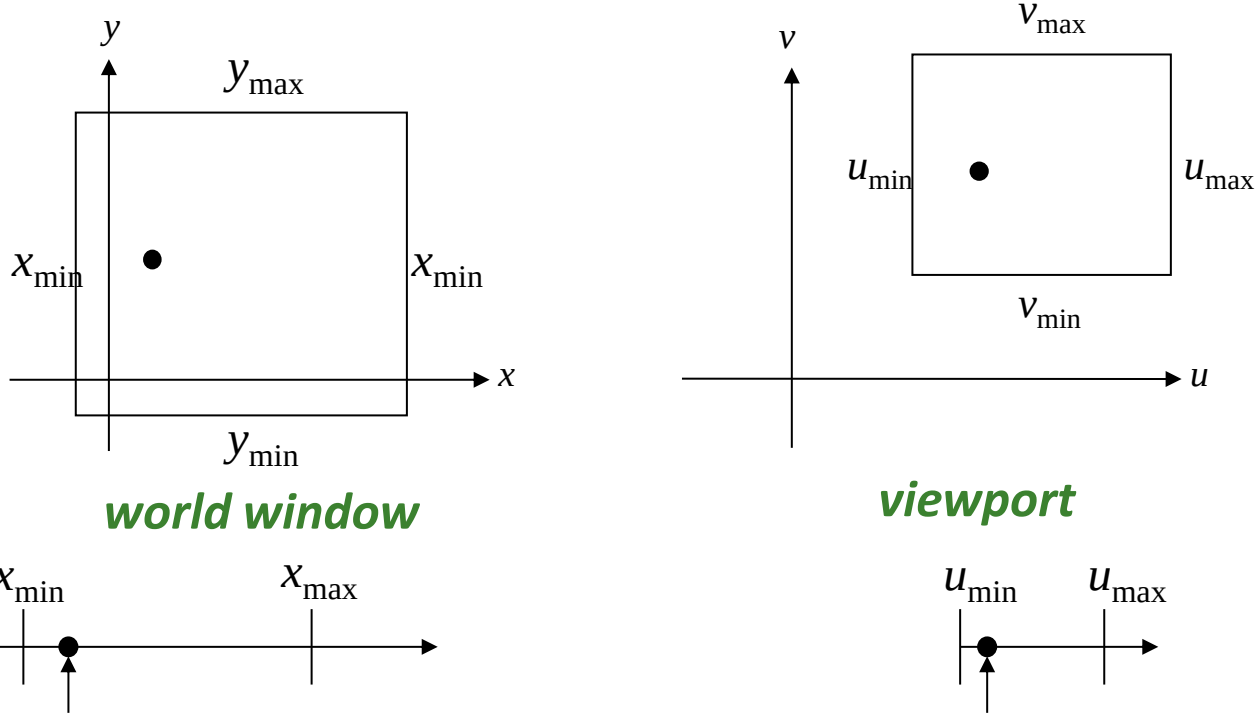
  - т.е. $\theta = 90^0$
  - тогава $R = \begin{bmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{bmatrix}$
  - еквивалентно на

$$R(-90^0)$$

Трансформация Window-to-Viewport

- Световният изглед и рамката за визуализиране трябва да имат еднакъв константен коефициента на пропорционалност за да се избегнат изкривявания
  - абсолютната стойност на височината и ширината на двата прозореца не са от значение, важно е съотношението им
    - трябва да се запазят пропорциите
  - относителните местоположения в световния изглед трябва да съответстват на същите относителните местоположения в рамката на екрана

Трансформация Window-to-Viewport



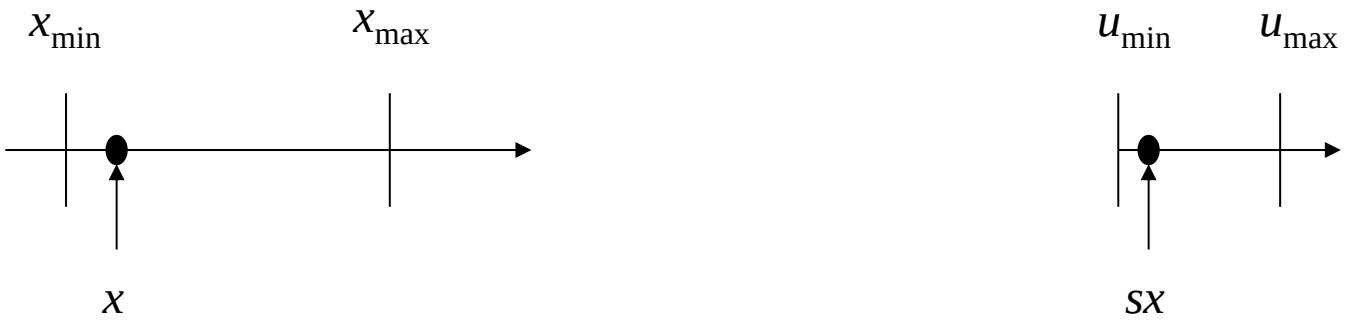
- Същите относителни местоположения означава:

$$\frac{x - x_{min}}{x_{max} - x_{min}} = \frac{sx - u_{min}}{u_{max} - u_{min}}$$

Трансформация Window-to-Viewport

- Определяне на позицията в рамката на екрана (sx) при дадена позиция в световния изглед x:

$$sx = \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} x - u_{\min} + \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} x_{\min}$$



Трансформация Window-to-Viewport

- Определяне на координати в рамката на екрана (sx,sy) при дадени координати в световния изглед (x,y):

$$sx = Ax + C \qquad A = \frac{vr - vl}{wr - wl} \qquad C = vl - Awl$$

$$sy = By + D \qquad B = \frac{vt - vb}{wt - wb} \qquad D = vb - Bwb$$

- При зададени

$$\{x_{\min}, y_{\min}, x_{\max}, y_{\max}\}, \{u_{\min}, v_{\min}, u_{\max}, v_{\max}\}, \{x, y\}$$

могат да се изчислят екранните координати $\{sx, sy\}$

Трансформация Window-to-Viewport

1. Транслация на долния ляв ъгъл на световния изглед в началото на координатната система

$$t_x = -x_{\min} \qquad t_y = -y_{\min}$$

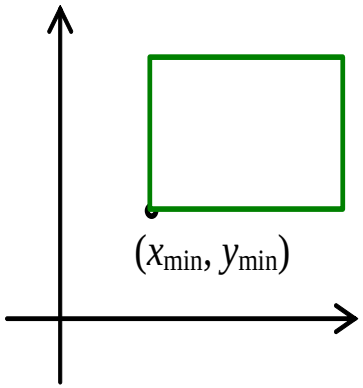
2. Мащабиране на ширината и височината на световния изглед за да съвпадат с размерите на рамката за визуализиране

$$s_x = \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} \qquad s_y = \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}}$$

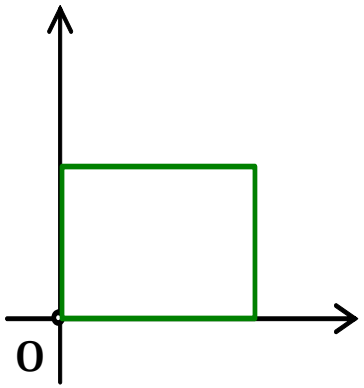
3. Транслация на долния ляв ъгъл от началото на координатната система в долния ляв ъгъл на рамката за визуализиране

$$t_x = u_{\min} \qquad t_y = v_{\min}$$

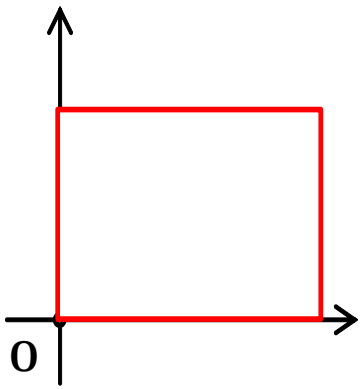
Трансформация Window-to-Viewport



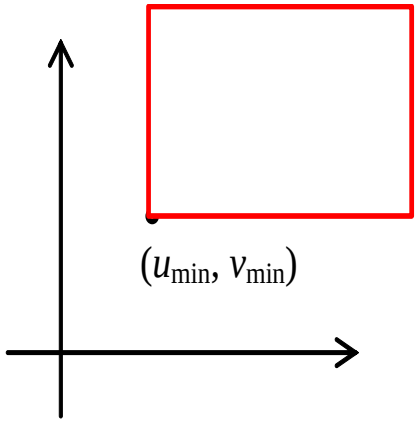
Прозорец в световни координати



Транслиран до началото на КС



Мащабиран до рамката



Преместен в крайна позиция

$$t_x = -x_{\min} \qquad t_y = -y_{\min}$$

$$s_x = \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} \qquad s_y = \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}}$$

$$t_x = u_{\min} \qquad t_y = v_{\min}$$

Трансформация Window-to-Viewport

$$M_{WV} = T(u_{\min}, v_{\min}) S\left(\frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}}, \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}}\right) T(-x_{\min}, -y_{\min})$$

$$= \begin{bmatrix} 1 & 0 & u_{\min} \\ 0 & 1 & v_{\min} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} & 0 & 0 \\ 0 & \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & -x_{\min} \\ 0 & 1 & -y_{\min} \\ 0 & 0 & 1 \end{bmatrix}$$

$$= \begin{bmatrix} \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} & 0 & -x_{\min} \frac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} + u_{\min} \\ 0 & \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} & -y_{\min} \frac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} + v_{\min} \\ 0 & 0 & 1 \end{bmatrix}$$

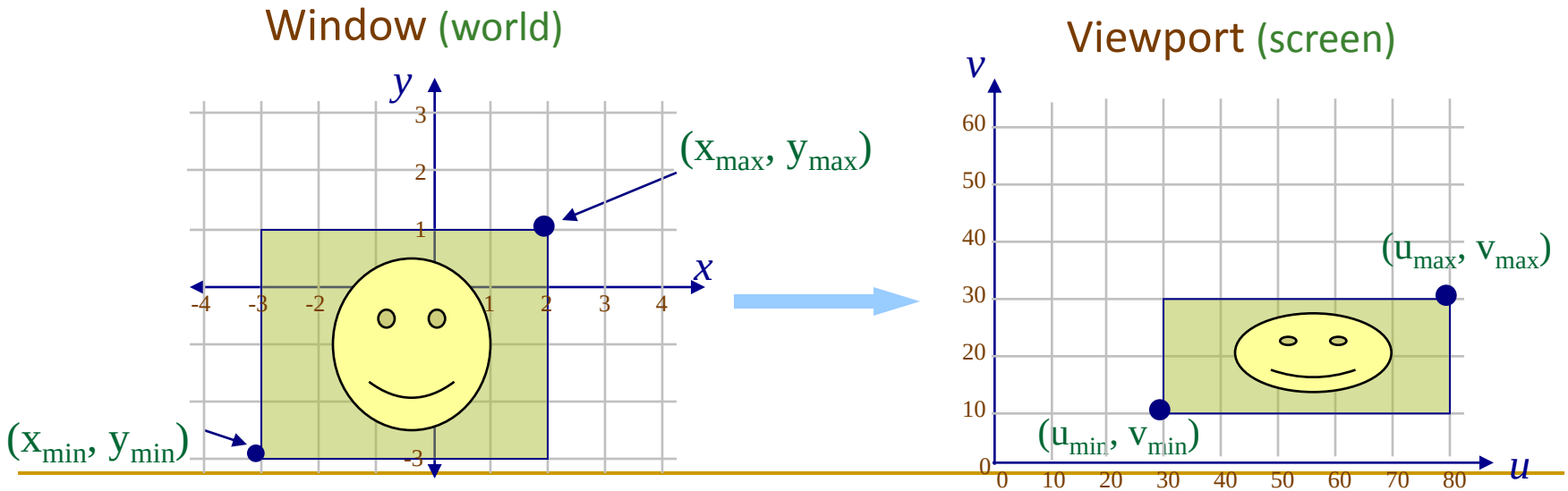
Трансформация Window-to-Viewport

- За точка P със световни координати (x,y) съответната точка P′ в рамката за визуализиране има координати, определени от произведението на общата матрица на трансформацията и световните координати на точката

$$P' = M_{WV} P = \begin{bmatrix} (x - x_{\min}) \dfrac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} + u_{\min} \\ (y - y_{\min}) \dfrac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} + v_{\min} \\ 1 \end{bmatrix}$$

Трансформация Window-to-Viewport

- Пример
  - Световен изглед (Window)
    - $(x_{min}, y_{min}) = (-3, -3)$
    - $(x_{max}, y_{max}) = (2 , 1)$
  - Рамка за визуализиране (Viewport)
    - $(u_{min}, v_{min}) = (30, 10)$
    - $(u_{max}, v_{max}) = (80, 30)$



Трансформация Window-to-Viewport

$$p' = \begin{bmatrix} (x - x_{\min})\dfrac{u_{\max} - u_{\min}}{x_{\max} - x_{\min}} + u_{\min} \\ (y - y_{\min})\dfrac{v_{\max} - v_{\min}}{y_{\max} - y_{\min}} + v_{\min} \\ 1 \end{bmatrix}$$

$$p' = \begin{bmatrix} (x-(-3))\dfrac{80-30}{2-(-3)} + 30 \\ (y-(-3))\dfrac{30-10}{1-(-3)} + 10 \\ 1 \end{bmatrix} = \begin{bmatrix} (x+3)\dfrac{50}{5} + 30 \\ (y+3)\dfrac{4}{4} + 10 \\ 1 \end{bmatrix} = \begin{bmatrix} (x+3)(10) + 30 \\ (y+3)(5) + 10 \\ 1 \end{bmatrix}$$

$$= \begin{bmatrix} 10x + 60 \\ 5y + 25 \\ 1 \end{bmatrix}$$

Трансформация Window-to-Viewport

$$p' = \begin{bmatrix} 10x + 60 \\ 5y + 25 \\ 1 \end{bmatrix}$$

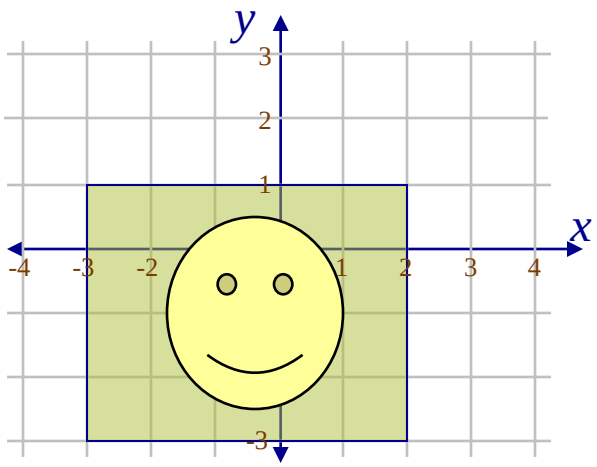
$(x_{min}, y_{min}) = (-3, -3) \rightarrow (30, 10)$

$(x_{max}, y_{max}) = (2, 1) \rightarrow (80, 30)$

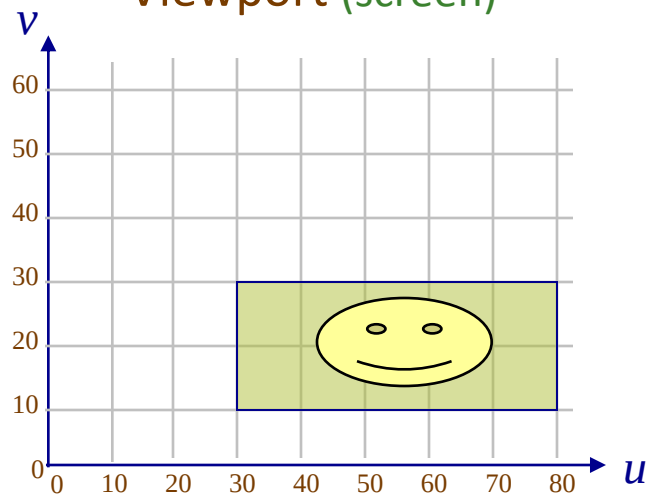
Ляво око $= (-1, -8) \rightarrow (50, 21)$

Връх на главата $= (-0.5, 0.5) \rightarrow (55, 27.5)$

Window (world)

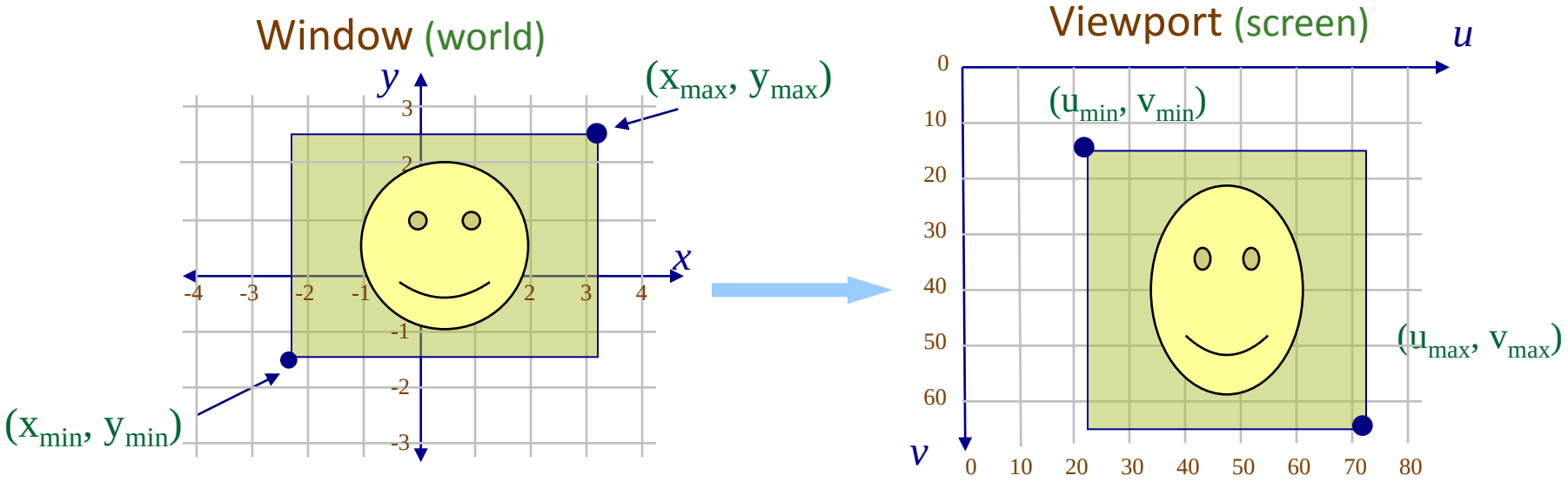


Viewport (screen)



Трансформация Window-to-Viewport

- Ако началото на координатната система на рамката за визуализиране е в горния ляв ъгъл?



Трансформация Window-to-Viewport

- Ако началото на координатната система на рамката за визуализиране е в горния ляв ъгъл
- то

$$M'_{WV}\begin{bmatrix} x \\ y_{\max} \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v_{\min} \\ 1 \end{bmatrix} \qquad M'_{WV}\begin{bmatrix} x \\ y_{\min} \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v_{\max} \\ 1 \end{bmatrix}$$

- докато в предишния случай

$$M_{WV}\begin{bmatrix} x \\ y_{\max} \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v_{\max} \\ 1 \end{bmatrix} \qquad M_{WV}\begin{bmatrix} x \\ y_{\min} \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v_{\min} \\ 1 \end{bmatrix}$$

Трансформация Window-to-Viewport

- Ако началото на координатната система на рамката за визуализиране е в горния ляв ъгъл

$$M_{WV}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v \\ 1 \end{bmatrix} \qquad M'_{WV}\begin{bmatrix} x \\ y \\ 1 \end{bmatrix} = \begin{bmatrix} u' \\ v' \\ 1 \end{bmatrix}$$

в този случай

$$M'_{WV}\begin{bmatrix} x \\ y_{\max} \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v_{\min} \\ 1 \end{bmatrix}$$

в предишния случай

$$M_{WV}\begin{bmatrix} x \\ y_{\max} \\ 1 \end{bmatrix} = \begin{bmatrix} u \\ v_{\max} \\ 1 \end{bmatrix}$$

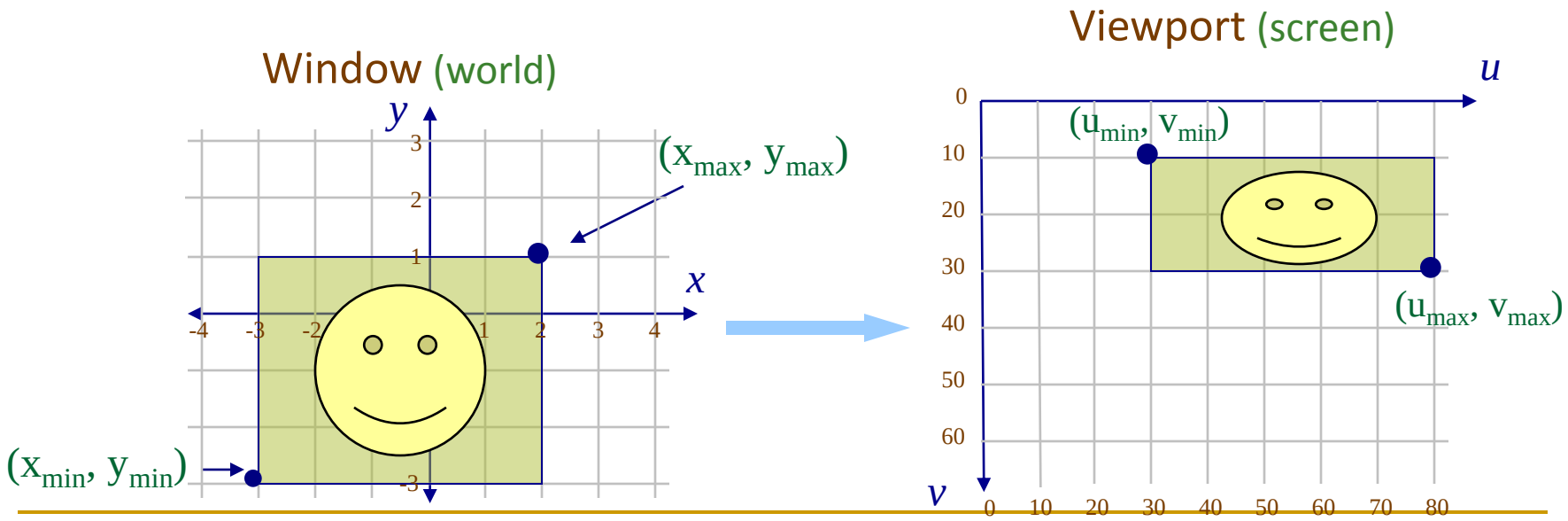
$$u' = u$$

$$v' = v_{\max} + v_{\min} - v$$

$$M'_{WV} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & v_{\max} + v_{\min} \\ 0 & 0 & 1 \end{bmatrix} M_{WV}$$

Трансформация Window-to-Viewport

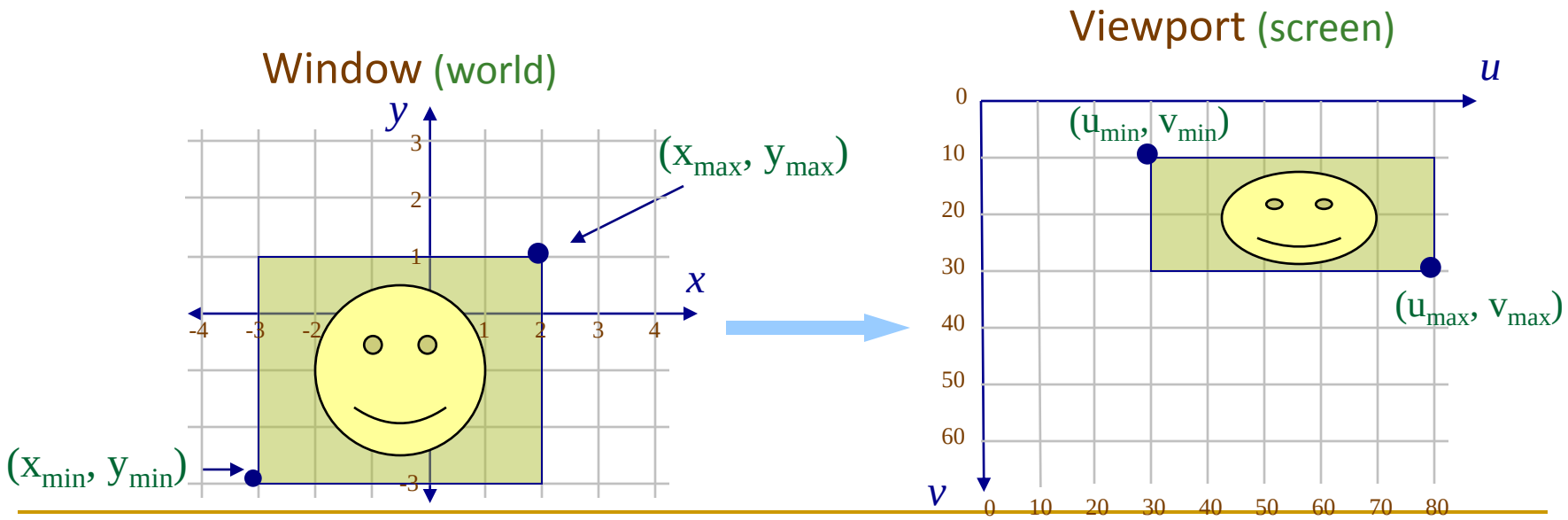
- Пример (КС с ордината v надолу)
  - Световен изглед (Window)
    - $(x_{min}, y_{min}) = (-3, -3)$
    - $(x_{max}, y_{max}) = (2 , 1)$
  - Рамка за визуализиране (Viewport)
    - $(u_{min}, v_{min}) = (30, 10)$
    - $(u_{max}, v_{max}) = (80, 30)$



Трансформация Window-to-Viewport

- Пример (КС с ордината v надолу)

$$p' = \begin{bmatrix} 10x+60 \\ v_{max}+v_{min}-(5y+25) \\ 1 \end{bmatrix} = \begin{bmatrix} 10x+60 \\ 30+10-(5y+25) \\ 1 \end{bmatrix} = \begin{bmatrix} 10x+60 \\ -5y+15 \\ 1 \end{bmatrix}$$



Трансформация Window-to-Viewport

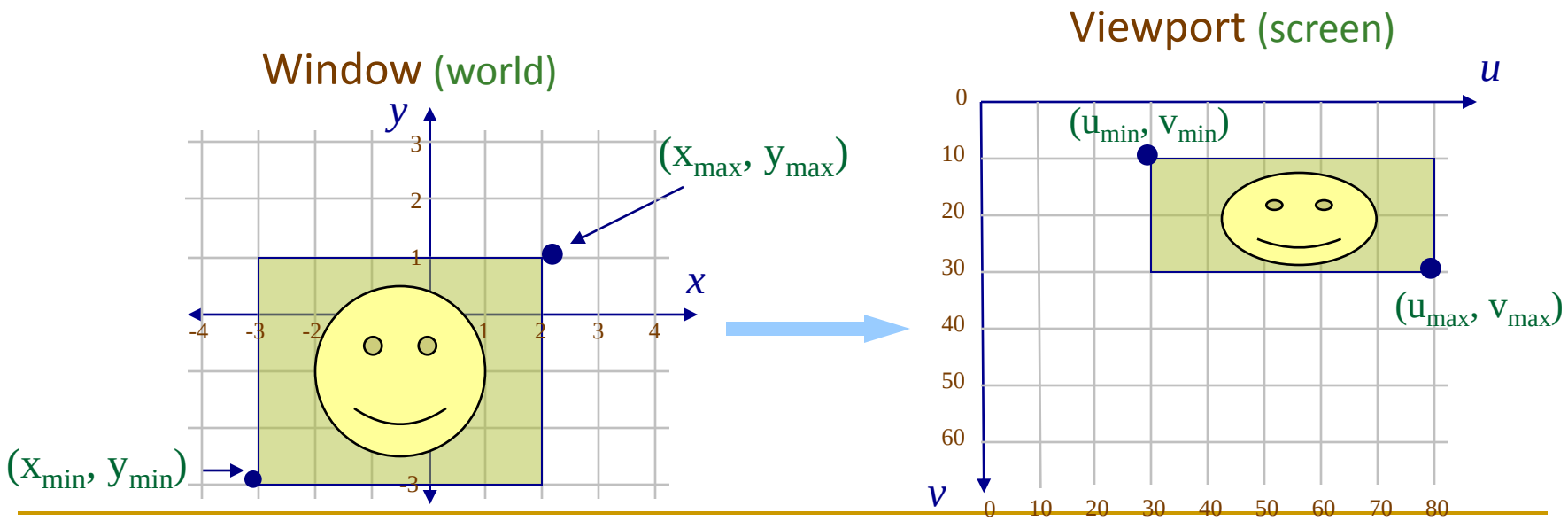
$$p' = \begin{bmatrix} 10x + 60 \\ -5y + 15 \\ 1 \end{bmatrix}$$

$(x_{min}, y_{min}) = (-3, -3) \rightarrow (30, 30)$

$(x_{max}, y_{max}) = (2, 1) \rightarrow (80, 10)$

Ляво око $= (-1, -8) \rightarrow (50, 19)$

Връх на главата $= (-0.5, 0.5) \rightarrow (55, 12.5)$

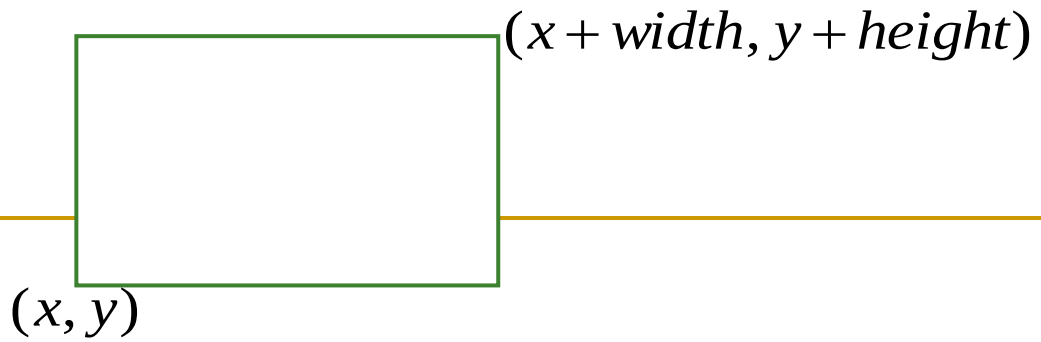


Трансформация Window-to-Viewport

- В OpenGL
  - `gluOrtho2D(left, right, bottom, top)`
    - задава световен изглед
    - дефинира се с две вертикални изрязващи равнини *left* и *right* и две хоризонтални изрязващи равнини *bottom* и *top*
    - по подразбиране световния изглед е (-1,1,-1,1)

Трансформация Window-to-Viewport

- В OpenGL
  - `glViewport(int x, int y, int width, int height)`
    - задава рамка за визуализиране в прозореца за визуализиране
    - *x* и *y* са координатите на долния ляв ъгъл
    - *width* и *height* са размерите на рамката
    - по подразбиране целия прозорец за визуализиране е рамка
    - допустимо е задаването на няколко рамки в прозореца за визуализиране



Трансформация Window-to-Viewport

```
void init() {
   glClearColor(0.3f, 0.0f, 0.0f, 1.0f);
   glMatrixMode(GL_PROJECTION);
   glLoadIdentity();
   gluOrtho2D(-2.0f, 2.0f, 0.0f, 1.0f);    // The world window
} // end init()

void display() {
   glClear(GL_COLOR_BUFFER_BIT);
   float x, y;
   glViewport(0.0f, 0.0f, 200.0f, 200.0f); // The viewport
   glBegin(GL_POINTS);
   for (x = -2.0; x <= 2.0; x += 0.1f) {
     y = exp(-x * x);
     glVertex2f(x, y);
   }
   glEnd();
   glutSwapBuffers();
} // end display()
```

Трансформация Window-to-Viewport

- Приложения
  - ***Pan***
    - преместване на рамката за визуализиране в световния изглед
  - ***Zoom***
    - увеличаване/намаляване на размера на прозореца

Пример

```
#include <fstream>
#include <iostream>
using namespace std;

class Coordinates{
private:
    int left, right, bottom, top;
public:
    Coordinates(int l, int r, int b, int t){
          left = l;
          right = r;
          bottom = b;
          top = t;
    }
    int getLeft(){
          return left;
    }
    int getRight(){
          return right;
    }
    int getBottom(){
          return bottom;
    }
    int getTop(){
          return top;
    }
};
```

Пример

```
void viewPortTransform(Coordinates *worldCoordinates, Coordinates*
    viewportCoordinates, int x, int y){
    int sx = 0 , sy = 0;
    int vr, vl, wr, wl, vt, vb, wt, wb;
    // Get sx
    vr = viewportCoordinates->getRight();
    vl = viewportCoordinates->getLeft();
    wr = worldCoordinates->getRight();
    wl = worldCoordinates->getLeft();
    sx = (x - wl) * (vr - vl) / (wr - wl) + vl;
    // Get sy
    vt = viewportCoordinates->getTop();
    vb = viewportCoordinates->getBottom();
    wt = worldCoordinates->getTop();
    wb = worldCoordinates->getBottom();
    sy = (y - wb) * (vt - vb) / (wt - wb) + vb;
    cout << x << ',' << y << " is transformed into: " << sx << ',' << sy <<
    endl;
}
```

Пример

```
int main(int argc, char** argv )
{
   char * filename = "test.dat";
   int l, r, b, t, numberOfPoints, x, y;
   Coordinates *worldCoordinates, *viewportCoordinates;
   fstream inStream;
   inStream.open(filename, ios ::in);
   if (inStream.fail())
       exit(0);
   inStream >> l; inStream >> r;
   inStream >> b; inStream >> t;
   worldCoordinates = new Coordinates(l, r, b, t);
   inStream >> l; inStream >> r;
   inStream >> b; inStream >> t;
   viewportCoordinates = new Coordinates(l, r, b, t);
   inStream >> numberOfPoints;
   for ( int i = 0; i < numberOfPoints; i++)
   {
     inStream >> x >> y;
     viewPortTransform(worldCoordinates, viewportCoordinates, x, y);
   }
   inStream.close();
   system("pause");
}
```

2D визуализиране

1) Изграждане на общата сцена чрез моделиращи трансформации над обектите
2) Задаване на прозорец и рамка за визуализиране
3) Преобразуване от световни координати на прозореца в екранни координати на рамката за визуализиране
4) **Изрязване по прозореца**
5) Растеризация на образите в координати на изходното устройство (екранни координати)

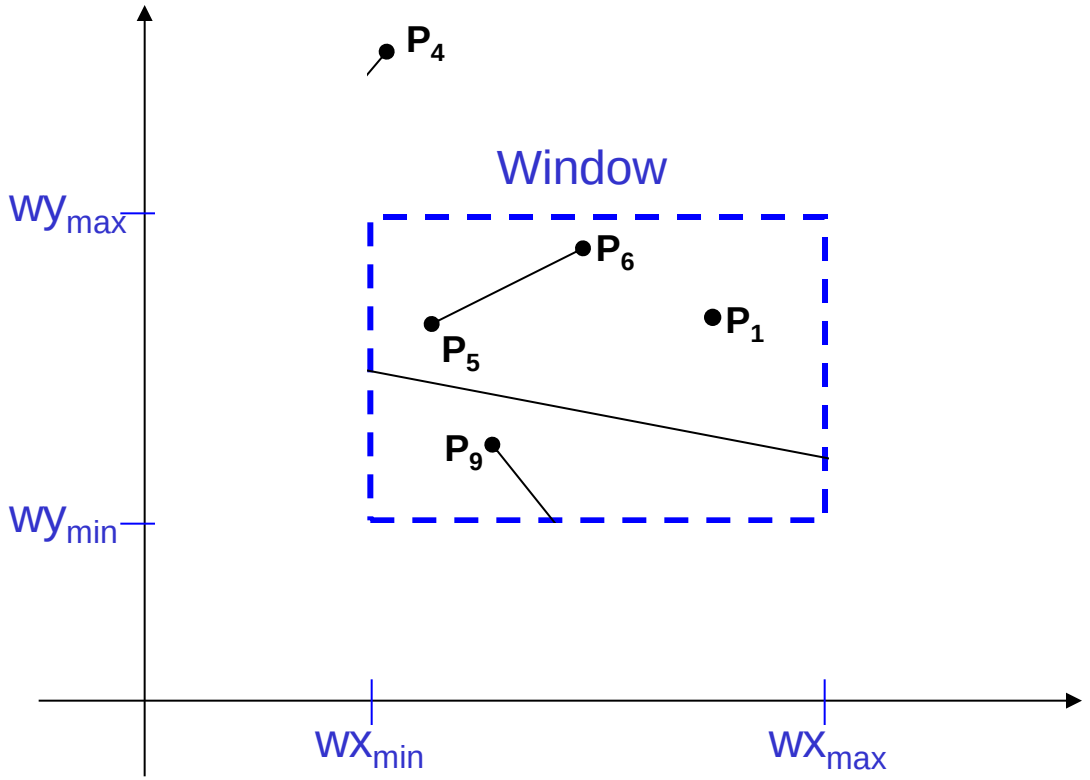
Изрязване

- 2D Clipping
- Процес на терминиране на примитивите, изграждащи обектите в сцената, които са извън рамката за визуализиране
  - стар проблем в компютърната графика
  - старо решение, което все още се използва, защото
    - работи
    - минимизира количеството изчисления
    - ясна идея

Изрязване по правоъгълен прозорец

- Прозорецът е зададен в световни координати
  - (wx_{min}, wy_{min}) и (wx_{max}, wy_{max})
- Задачата е
  - да се определят примитиви и части от примитиви извън прозореца

Изрязване по правоъгълен прозорец

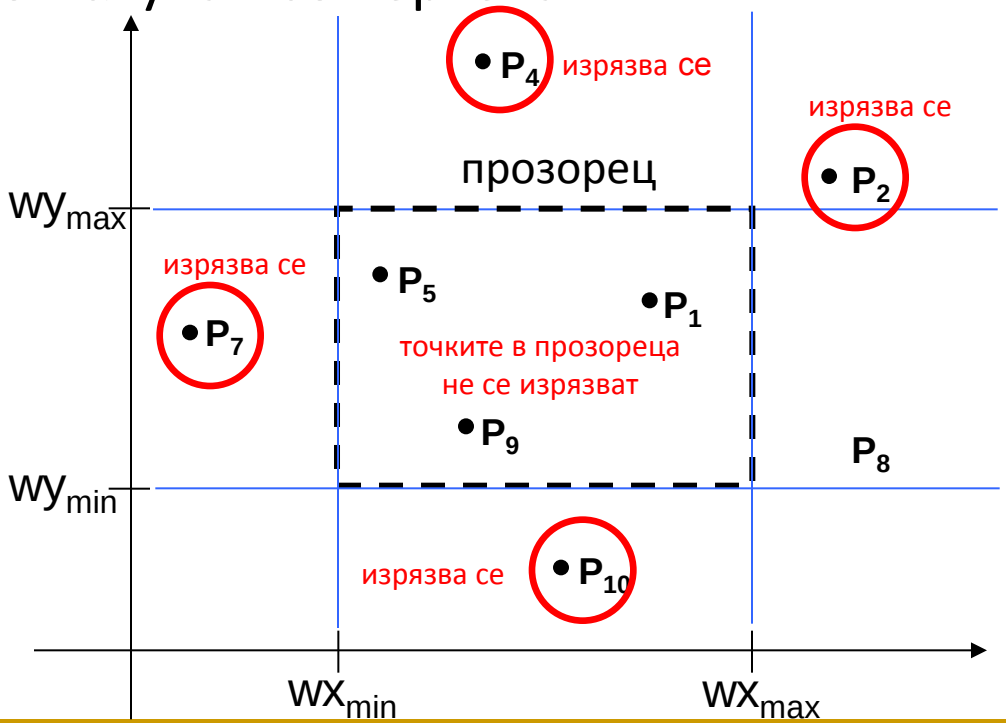


Изрязване на точки

- Точка P с координати (x,y) не се изрязва ако

$$(wx_{min} \le x \le wx_{max}) \text{ AND } (wy_{min} \le y \le wy_{max})$$

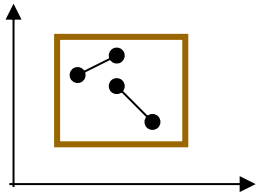
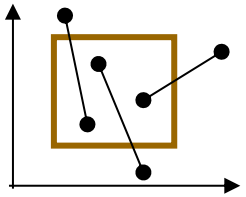
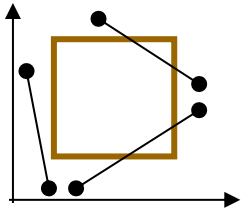
- в противен случай се изрязва



...

Изрязване на отсечки

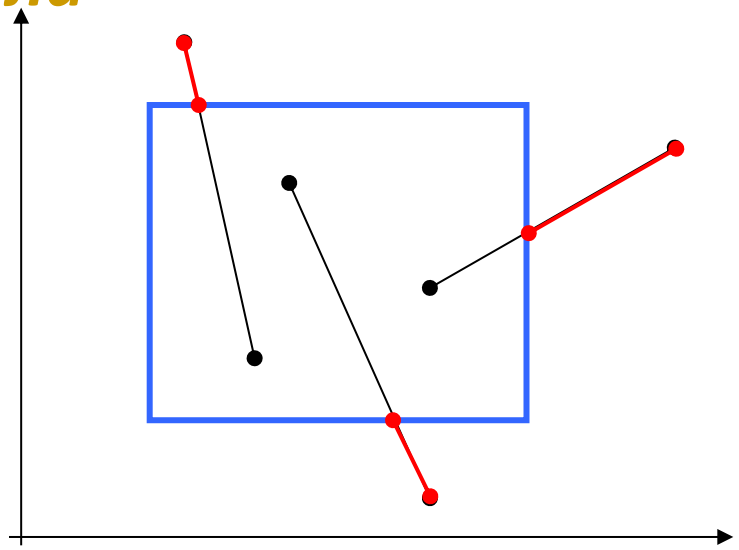
- Проверяват се крайните точки на всяка отсечка дали са вътре или извън прозореца

Ситуация	Решение	Пример
И двете крайни точки са вътре в прозореца	не се изрязва	
Едната крайна точка е вътре в прозореца, другата е извън	изрязва се	
И двете крайни точки са извън прозореца	неопределено	

...

Изрязване на отсечки

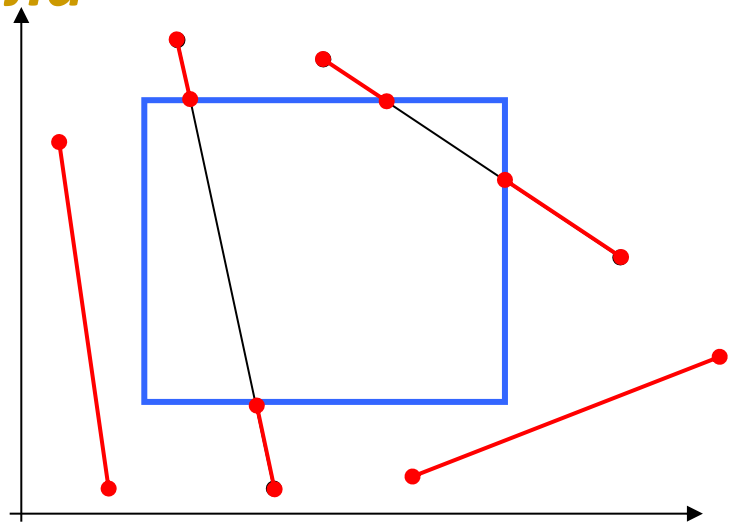
- ***Алгоритъм на "грубата сила"***
  - Brute force line clipping
    - не се изрязват отсечки с две крайни точки вътре в прозореца
    - за отсечки с едната крайна точка вътре в прозореца, другата извън
      - изчислява се пресечна точка (използва се уравнение на права)
      - отсечката се изрязва в тази точка



...

Изрязване на отсечки

- ***Алгоритъм на "грубата сила"***
  - Brute force line clipping
    - за отсечки с двете крайни точки извън прозореца
      - изчислява се пресечна точка с всяка от граничните страни на прозореца
      - отсечката се изрязва съответно в точките на пресичане



- Изчисляването на пресечна точка на прави е изчислително сложно
- Дадена сцена може да съдържа много линии
  - алгоритъмът на "грубата сила" е твърде бавен и неефективен

Изрязване на отсечки

- Пресечна точка на прави линии
  - Нека отсечка е зададена с двете си крайни точки
    - $P_1(x_1, y_1)$ и $P_2(x_2, y_2)$
  - Точката на пресичане на две прави се определя с решаване на система от 2 уравнения с 2 неизвестни

$$\begin{cases} y = m_1 x + b_1 \\ y = m_2 x + b_2 \end{cases}$$

Изрязване на отсечки

- Вертикална права (ръб на прозорец)
  - *например:* $x = wx_{\min}$

$$\left|\begin{array}{l} m = (y_2 - y_1)/(x_2 - x_1) \\ x_0 = wx_{\min} \\ y_0 = y_1 + m(wx_{\min} - x_1) \end{array}\right.$$

- Хоризонтална права (ръб на прозорец)
  - *например:* $y = wy_{\min}$

$$\left|\begin{array}{l} m = (y_2 - y_1)/(x_2 - x_1) \\ x_0 = x_1 + (wy_{\min} - y_1)/m \\ y_0 = wy_{\min} \end{array}\right.$$

Изрязване на отсечки

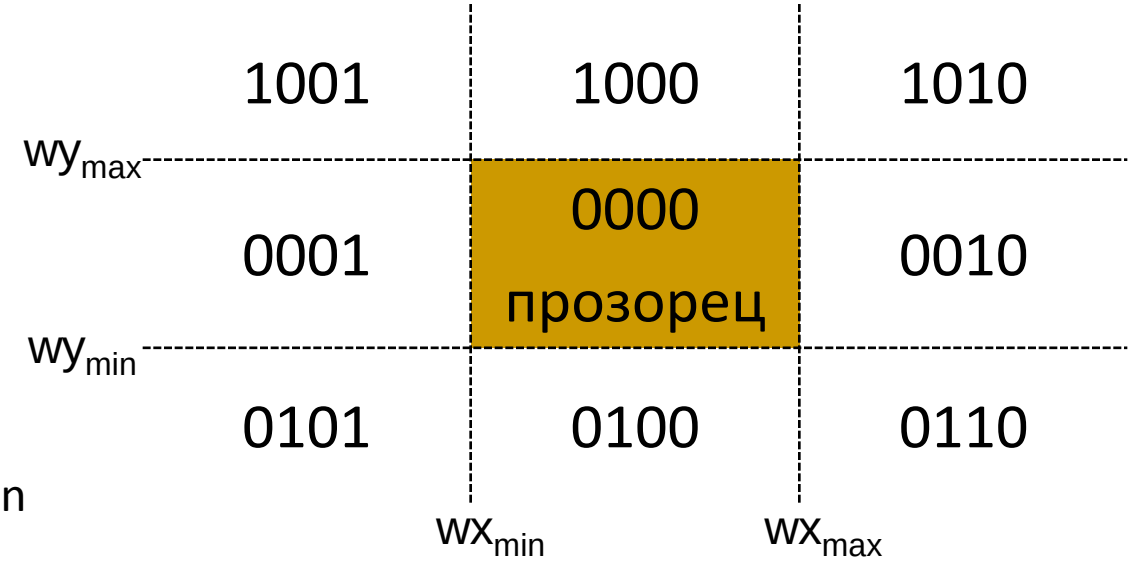
- ***Алгоритъм на Коен-Съдърленд за изрязване на отсечки***
  - Cohen-Sutherland
  - разглеждат се 9 възможни региона
    - всеки регион има уникален 4 битов регионален код
    - регионалните кодове показват позицията на региона спрямо прозореца

Легенда за регионалния код
(LRBT → Left Right Bottom Top)

...

Алгоритъм на Коен-Съдърленд

3	2	1	0
над	под	дясно	ляво



Бит 0 = 1 ако $x<wx_{min}$

Бит 1 = 1 ако $x>wx_{max}$

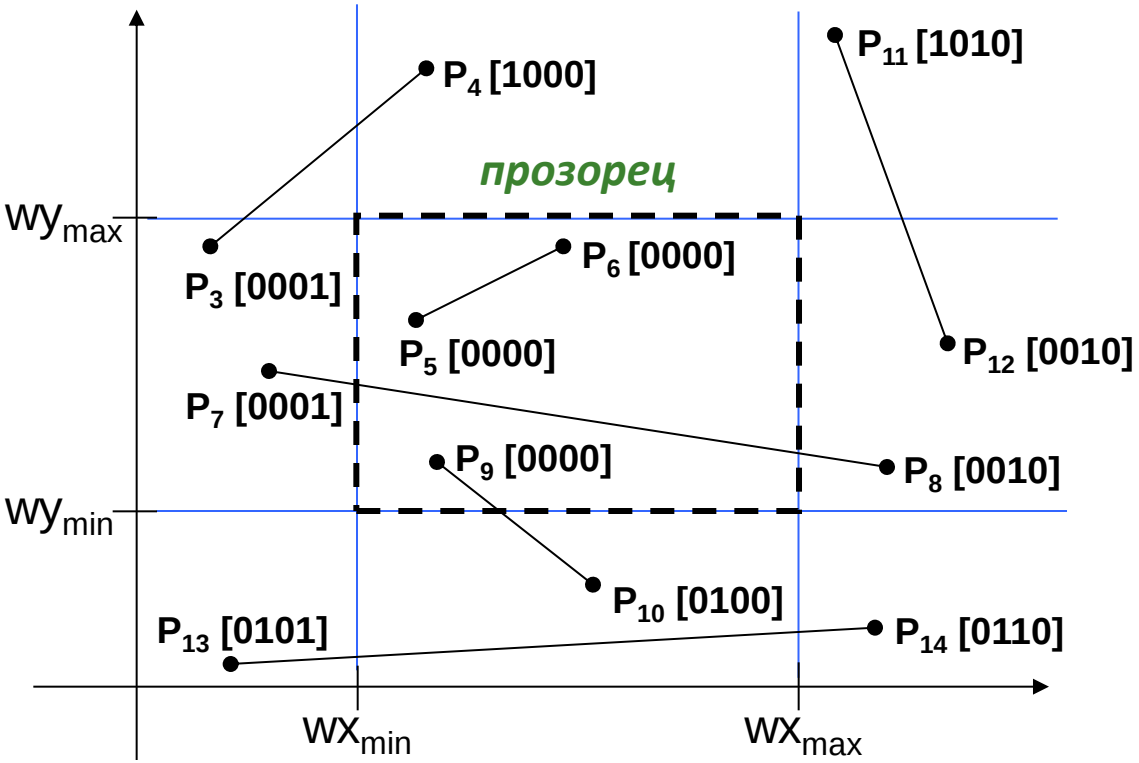
Бит 2 = 1 ако $y<wy_{min}$

Бит 3 = 1 ако $y>wy_{max}$

...

Алгоритъм на Коен-Съдърленд

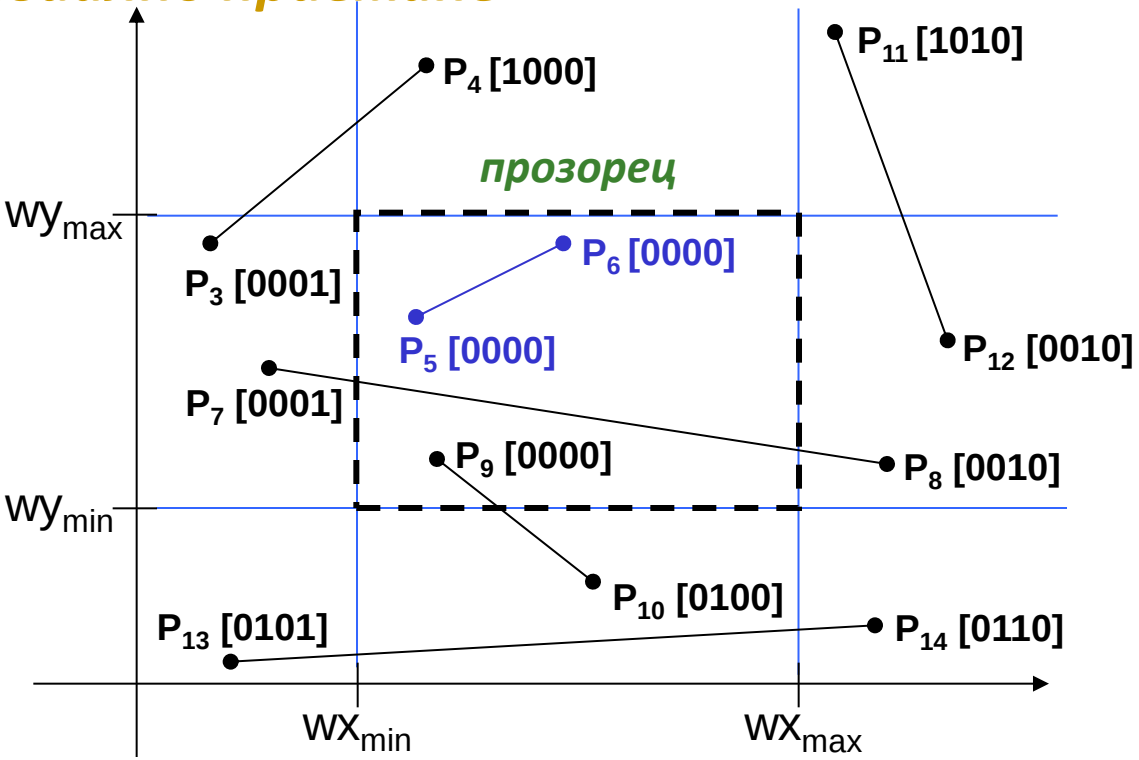
- За всяка крайна точка на отсечка се определя регионален код



...

Алгоритъм на Коен-Съдърленд

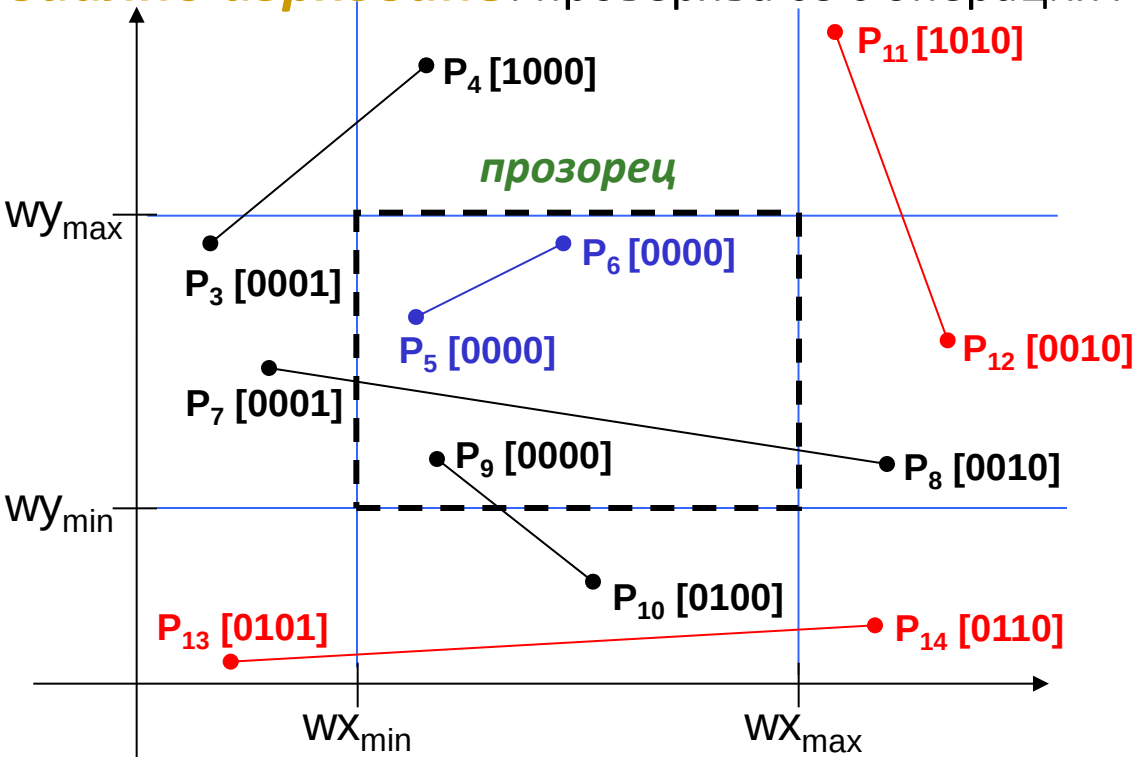
- Отсечките, за които и двете крайни точки имат код [0000] не се изрязват, тъй като напълно се съдържат в прозореца
  - ***тривиално приемане***



...

Алгоритъм на Коен-Съдърленд

- Отсечките, за които кодовете на двете крайни точки имат еднакви битове се изрязват
  - ***тривиално изрязване***: проверява се с операция AND



...

Алгоритъм на Коен-Съдърленд

- За отсечките, за които по кодовете на двете крайни точки не може да се определи, че са напълно вътре или напълно извън прозореца, се определя дали пресичат границите на прозореца
  - за крайните точки извън прозореца се определя пресечна точка с всяка една от границите
    - разглеждат се в определен ред, например лява, дясна, долна, горна (LRBT)
  - изрязва се част от отсечката спрямо дадена граница
  - продължава се с другите граници докато разглежданата отсечка е напълно вътре или напълно извън прозореца

...

Алгоритъм на Коен-Съдърленд

- За да се определи с кои от границите на прозореца да се търси пресечна точка се използват регионалните кодове
  - за да се определи дали отсечка пресича дадена граница на прозорец се сравняват съответните битове на двете крайни точки на отсечката
    - ако единият има стойност 1, а другият стойност 0, то отсечката пресича границата

...

Алгоритъм на Коен-Съдърленд

1) Определят се кодовете на двете крайни точки на отсечката;
2) Проверява се дали отсечката се приема изцяло (кодовете на двете крайни точки са 0000); при тривиално приемане – край;
3) Проверява се дали отсечката се изрязва изцяло (конюнкцията на кодовете на двете крайни точки **не** е 0000); при тривиално изрязване – край;
4) Избира се крайна точка вън от прозореца (с код, различен от 0000) и от кода й се определя ръб на прозореца, пресечен от отсечката, а след това и точка на пресичане;
5) Избраната външна точка се замества с изчислената точка на пресичане, изчислява се кодът й и се преминава към стъпка 2.

...

Алгоритъм на Коен-Съдърленд

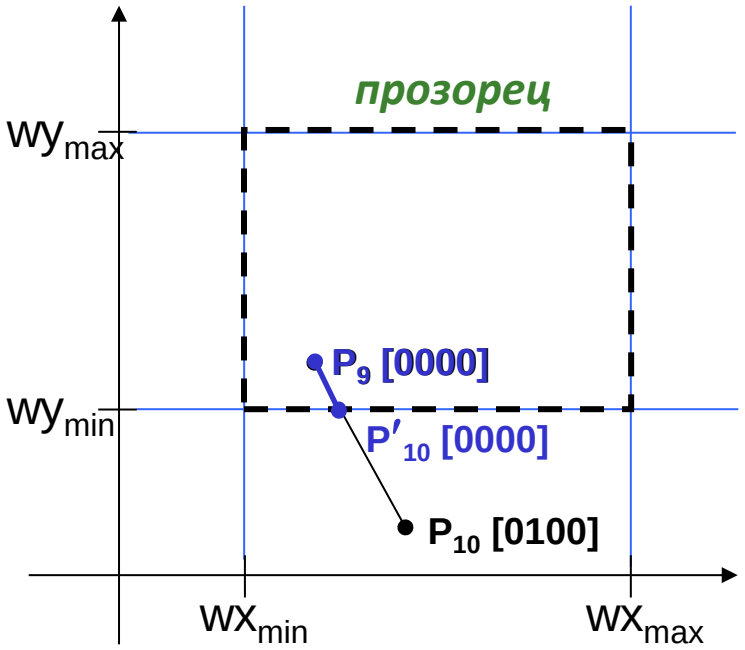
```
int clipSegment(Point2& p1, Point2& p2, RealRect W)
{
     do{
          if(trivial accept) return 1; // Full line survives
          if(trivial reject) return 0; // no portion survives

          if(p1 is outside)
          {
             if(p1 is to the left)  chop against the left edge
             else if(p1 is to the right) chop against the right edge
             else if(p1 is below)  chop against the bottom edge
             else if(p1 is above)  chop against the top edge
          }
          else      // p2 is outside
          {
            if(p2 is to the left) chop against the left edge
                 else if(p2 is to the right) chop against the right edge
            else if(p2 is below)  chop against the bottom edge
            else if(p2 is above) chop against the top edge
          }
     }while(1);
}
```

...

Алгоритъм на Коен-Съдърленд

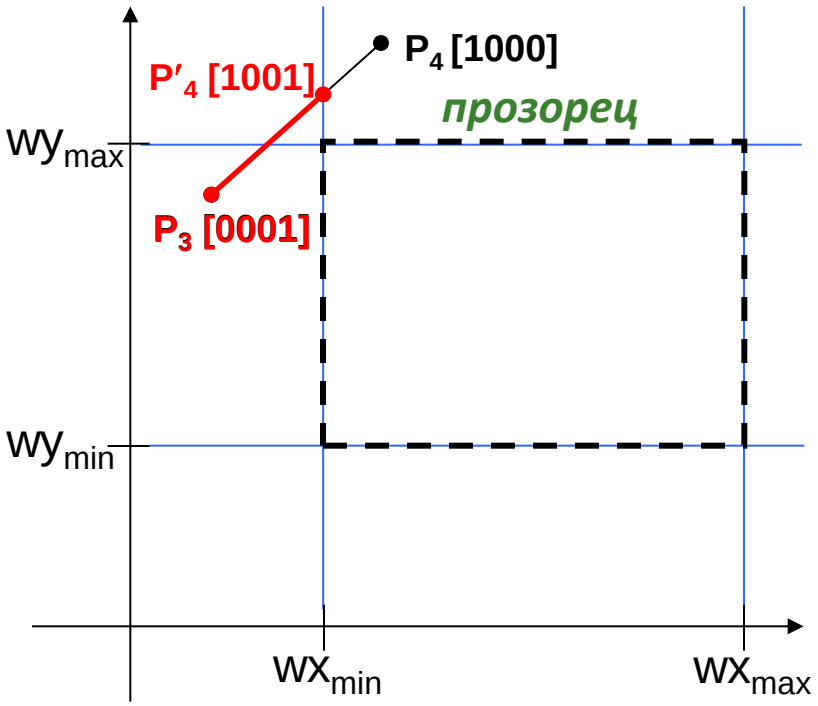
- Пример: разглежда се отсечката от P_9 до P_{10}
  - започва се с точка P_{10}
  - по регионалните кодове на крайните точки се определя, че отсечката не пресича лявата и дясната граница
  - изчисляват се координатите на пресечната точка P'_{10} на отсечката с долната граница на прозореца
  - отсечката от P_9 до P'_{10} е напълно в прозореца и се запазва



...

Алгоритъм на Коен-Съдърленд

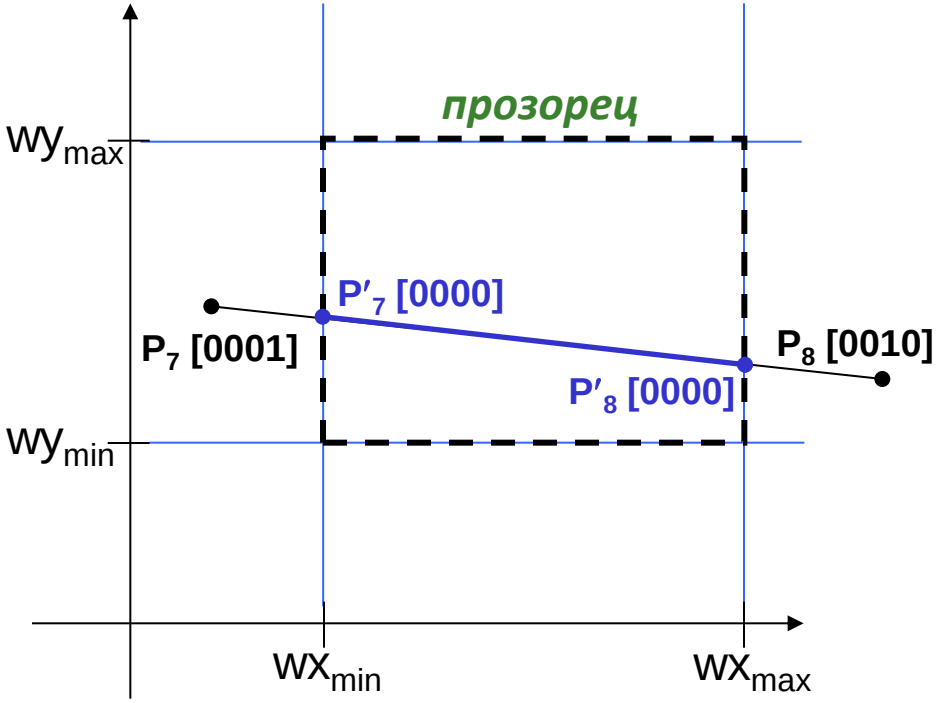
- Пример: разглежда се отсечката от P_3 до P_4
  - започва се с точка P_4
  - по регионалните кодове на крайните точки се определя, че отсечката пресича лявата граница
  - изчисляват се координатите на пресечната точка P'_4 на отсечката с лявата граница на прозореца
  - отсечката от P_3 до P'_4 е напълно извън прозореца и се изрязва



...

Алгоритъм на Коен-Съдърленд

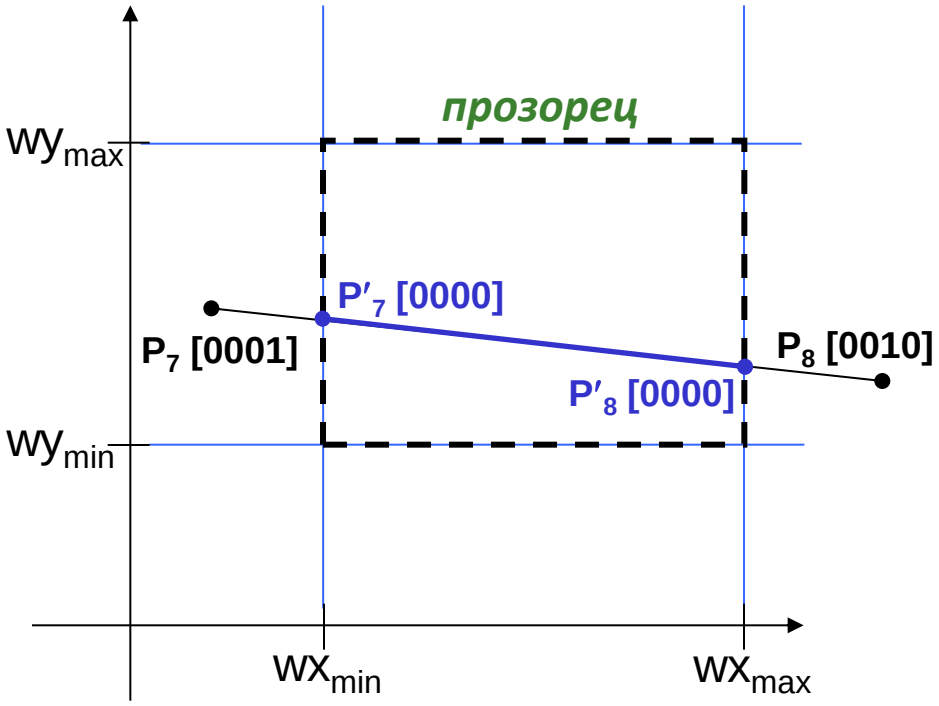
- Пример: разглежда се отсечката от P_7 до P_8
  - започва се с точка P_7
  - по регионалните кодове на крайните точки се определя, че отсечката пресича лявата граница
  - изчисляват се координатите на пресечната точка P'_7 на отсечката с лявата граница на прозореца



...

Алгоритъм на Коен-Съдърленд

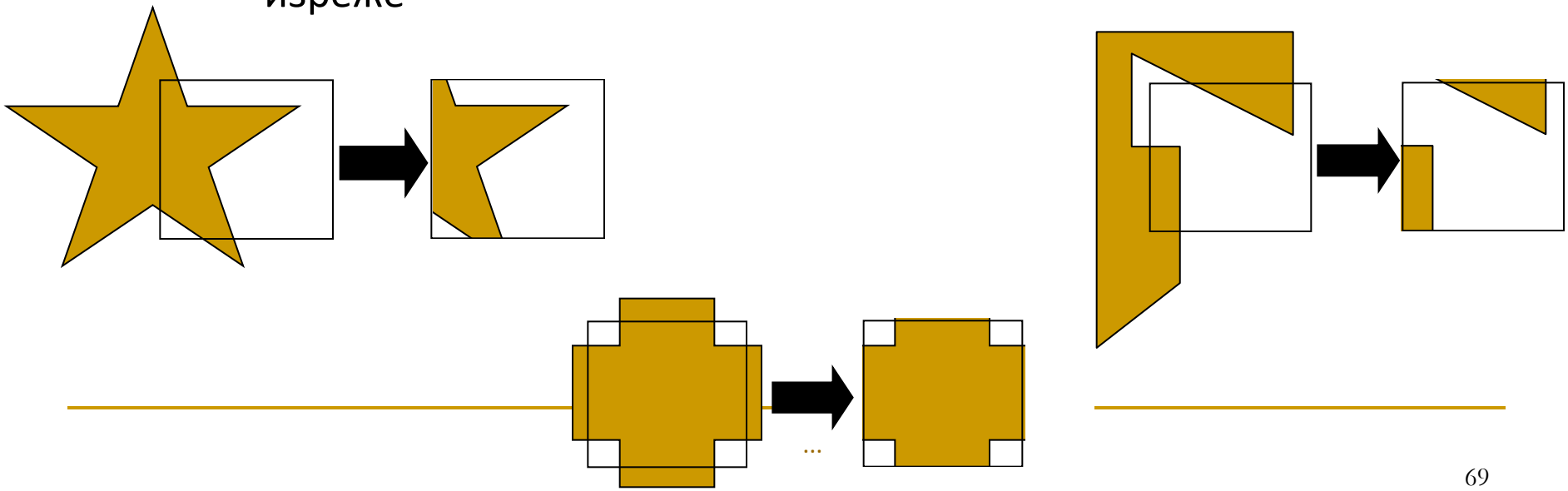
- Пример: разглежда се отсечката от P'_7 до P_8
  - започва се с точка P_8
  - изчисляват се координатите на пресечната точка P'_8 на отсечката с дясната граница на прозореца
  - отсечката от P'_7 до P'_8 е напълно вътре в прозореца и се запазва



...

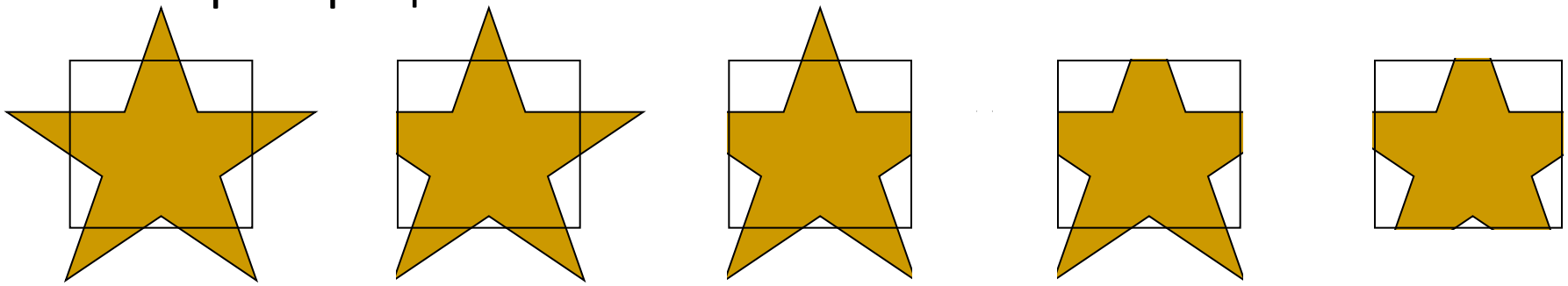
Изрязване по правоъгълен прозорец

- Изрязване на многоъгълници
  - аналогично на линните
  - многоъгълниците трябва да бъдат изрязвани спрямо прозореца на визуализиране
    - трябва да се вземе предвид коя част от многоъгълника ще се изреже



Изрязване по правоъгълен прозорец

- ***Алгоритъм на Съдърленд-Ходжмън за изрязване на многоъгълници***
  - Sutherland-Hodgeman
  - многоъгълникът се изрязва като се разглеждат последователно всяка една от границите на прозореца



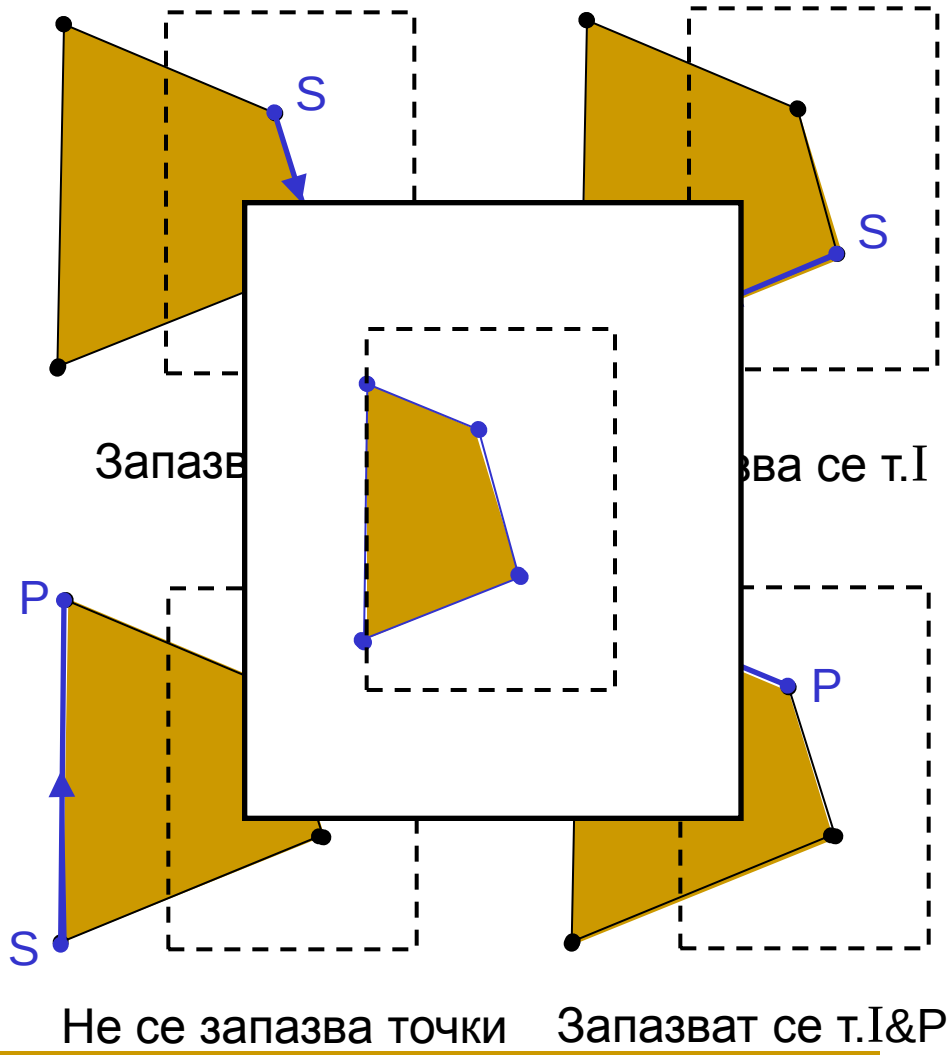
Оригинален обект | Изрязване отляво | Изрязване отдясно | Изрязване отгоре | Изрязване отдолу

Алгоритъм на Съдърленд-Ходжмън

- За изрязване на полигон спрямо дадена граница
  - разглежда се последователно позицията на всеки възел спрямо тази граница
    - възлите вътре в границата се запазват за изрязване спрямо следващата граница
    - възлите извън границата се изрязват
    - ако от точка извън границата се достигне до точка вътре в границата, то се запазва пресечната точка на отсечката с границата
    - ако пресичането е отвън навътре, то се запазват и пресечната точка, и разглеждания възел от полигона

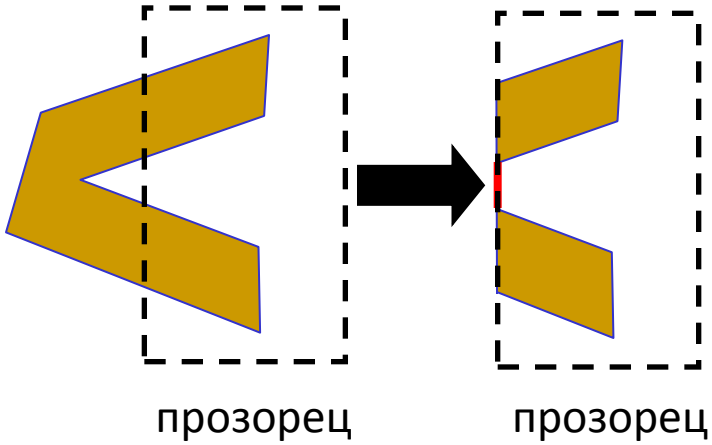
Алгоритъм на Съдърленд-Ходжмън

- Примери
  - Разглежда се и се обработва т.P, а предишната точка е S
  - Запазените точки определят областта, която се изрязва спрямо разглежданата граница



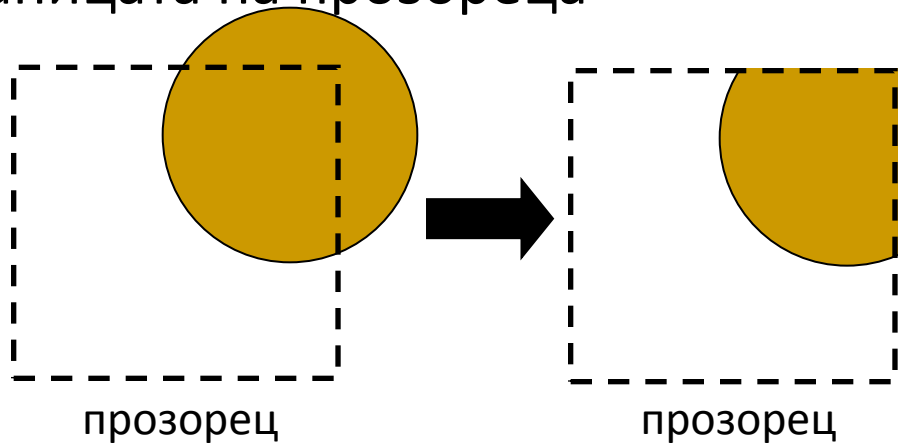
Изрязване по правоъгълен прозорец

- При изрязване на вдлъбнати многоъгълници могат да се получат свързващи линии



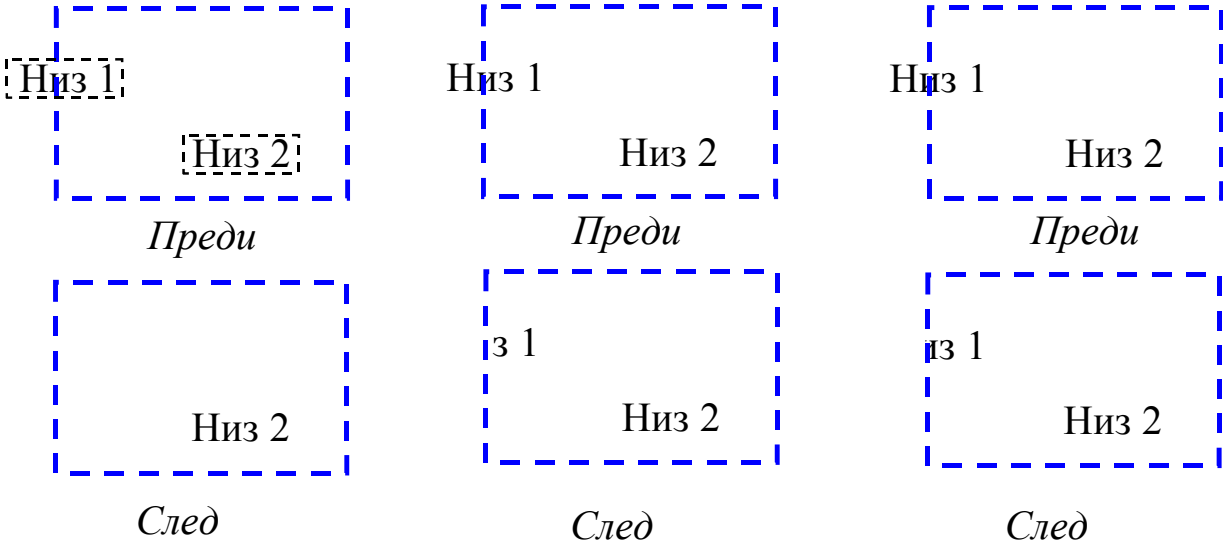
Изрязване по правоъгълен прозорец

- Изрязване на криви
  - най-напред се прави проверка за изцяло изрязване или изцяло приемане с ограждащи правоъгълници
  - изискват се повече изчисления
  - за окръжност: трябва да се намерят двете пресечни точки с границата на прозореца



Изрязване по правоъгълен прозорец

- Изрязване на текст
  - Изрязване на целия низ, ако част от него не се побира в прозореца;
  - Изрязване на целия символ, ако част от него не се побира в прозореца;
  - Изрязване само на тази част от низа, която е извън прозореца



Пример

```
#include <fstream>
#include <iostream>
using namespace std;

class LineCoords {
    private:
        float x1, y1, x2, y2;
    public:
        LineCoords(float a, float b, float c, float d){
            x1 = a; y1 = b; x2 = c; y2 = d;
            cout << "Initial line: (" << x1 << ", " << y1 << ") to (" << x2 <<
                                                    ", " << y2 << ")" << endl;
        }
        float getX1(){return x1;}
        float getY1(){return y1;}
        float getX2(){return x2;}
        float getY2(){return y2;}
        void setX1(float a) {x1 = a;}
        void setY1(float a) {y1 = a;}
        void setX2(float a) {x2 = a;}
        void setY2(float a) {y2 = a;}
};
```

Пример

```
bool* getCode(Coordinates *wCoords, float x, float y){
     bool *code = (bool*)malloc(4 * sizeof (bool));
     code[0] = x < wCoords->getLeft();
     code[1] = y > wCoords->getTop();
     code[2] = x > wCoords->getRight();
     code[3] = y < wCoords->getBottom();
//     cout << code[1] << ',' << code[2];
     return(code);
}

void ClipSegmentTop(float top, LineCoords* line){
     float x1 = line->getX1();  float y1 = line->getY1();
     float x2 = line->getX2();  float y2 = line->getY2();
     float t;
     if (y1 > top) {
          t = (top - y2) / (y1 - y2);
          y1 = top;   x1 = x2 + t * (x1 - x2);
     } else {
          t = (top - y1) / (y2 - y1);
          y2 = top;   x2 = x1 + t * (x2 - x1);
     }
     line->setX1(x1);  line->setY1(y1);
     line->setX2(x2);  line->setY2(y2);
     return;
}
```

Пример

```
void ClipSegmentRight(float right, LineCoords* line){
     float x1 = line->getX1();  float y1 = line->getY1();
     float x2 = line->getX2();  float y2 = line->getY2();
     float t;
     if (x1 > right) {
          t = (right - x2) / (x1 - x2);
          y1 = y2 + t * (y1 - y2);  x1 = right;
     } else {
          t = (right - x1) / (x2 - x1);
          y2 = y1 + t * (y2 - y1);  x2 = right;
     }
     line->setX1(x1);  line->setY1(y1);
     line->setX2(x2);  line->setY2(y2);
     return;
}

void CohenSutherland(Coordinates *wCoords, LineCoords *line){
     float x1 = line->getX1();
     float y1 = line->getY1();
     float x2 = line->getX2();
     float y2 = line->getY2();
     bool *code1, *code2;
     code1 = getCode(wCoords, x1, y1);
     code2 = getCode(wCoords, x2, y2);
```

Пример

```
// --- Trivial accept ---
     bool s = false;
     for (int i = 0; i < 4; i++){ s = s || code1[i] || code2[i]; }
     if (!s){
         cout << "Final line: (" << x1 << ", " << y1 << ") to (" << x2 << ", "
                                              << y2 << ")" << endl << endl;
         return;
     }
// --- Trivial reject ---
     s = false;
     for (int i = 0; i < 4; i++){ s = s || code1[i] && code2[i]; }
     if (s){
         cout << "The line is outside" << endl << endl;
         return;
     }
// --- Clipping is needed ---
     // We only implement top and right clipping!
     if (code1[1] || code2[1]) {
        ClipSegmentTop(wCoords->getTop(), line);
        CohenSutherland(wCoords, line);
     } else // (code1[2] || code2[2])
     {
        ClipSegmentRight(wCoords->getRight(), line);
        CohenSutherland(wCoords, line);
     }
}
```

Пример

```
int main(int argc, char** argv )
{
   char * filename = "test.dat";
   float l, r, b, t, x1, y1, x2, y2;
   int numberOfLines;
   Coordinates *worldCoordinates;
   LineCoords *line;
   fstream inStream;
   inStream.open(filename, ios ::in);
   if ( inStream.fail())
       exit(0);
   // read world coordinates
   inStream >> l;  inStream >> r;  inStream >> b;  inStream >> t;
   worldCoordinates = new Coordinates(l, r, b, t);
   inStream >> numberOfLines;
   // read lines
   for ( int i = 0; i < numberOfLines; i++)
   {
     inStream >> x1 >> y1 >> x2 >> y2;
     line = new LineCoords(x1, y1, x2, y2);
     CohenSutherland(worldCoordinates, line);
      free(line);
   }
   inStream.close();
   system("pause");
}
```

КРАЙ

Следваща тема:

Основни алгоритми за растеризиране