

СОФИЙСКИ УНИВЕРСИТЕТ
„СВ. КЛИМЕНТ ОХРИДСКИ“



Отсичане

ТЕМА №16

Съдържание

Тема 16: Отсичане

- Екранно отсичане
- Отсичане на отсечка
- Алгоритъм на Коеен-Съдърленд
- Отсичане на многоъгълник

Екранно отсичане

Проблем

Проблем за решаване

- Растеризирането на примитиви е в рамките на правоъгълна зона
- Цял екран, прозорец или част от прозорец
- Няма нужда да се растеризира извън тази зона

Цел на отсичането (на англ. clipping)

- Да няма растеризиране извън видимата зона
- Да ускори растеризирането

Ускорение

- Не се растеризират целите примитиви
- Растеризира се само видимата им част
- Някои примитиви дори и не се рисуват

Особености

Отсичащи прави

- Почти винаги те са хоризонтални и вертикални:
 $x = \textit{const}$ или $y = \textit{const}$
- Отсичането е в правоъгълник

Координати

- Почти винаги те са цели числа
(измерват разстояние в пиксели)

Ограничение

- Правоъгълна зона на отсичане
- Ограничена от прави, успоредни на координатните оси:

$$x_{min} \leq x \leq x_{max}$$

$$y_{min} \leq y \leq y_{max}$$

- Отсичане, приложимо и за векторна графика
- Отсичане на отсечки и многоъгълници

Отсичане на отсечка

Алгоритми за отсичане

Много алгоритми за отсичане на отсечки:

- Наивен алгоритъм (даже са два броя)
- На Коеен-Съдърленд (Cohen-Sutherland)
- На Лианг-Барски (Liang-Barsky)
- На Никол-Лий-Никол (Nicholl-Lee-Nicholl)
- На Сайръс-Бек (Cyrus-Beck)

Наивни алгоритми

Първи наивен алгоритъм

- Растеризира се нормално
- Всеки пиксел се проверява за видимост

Особености

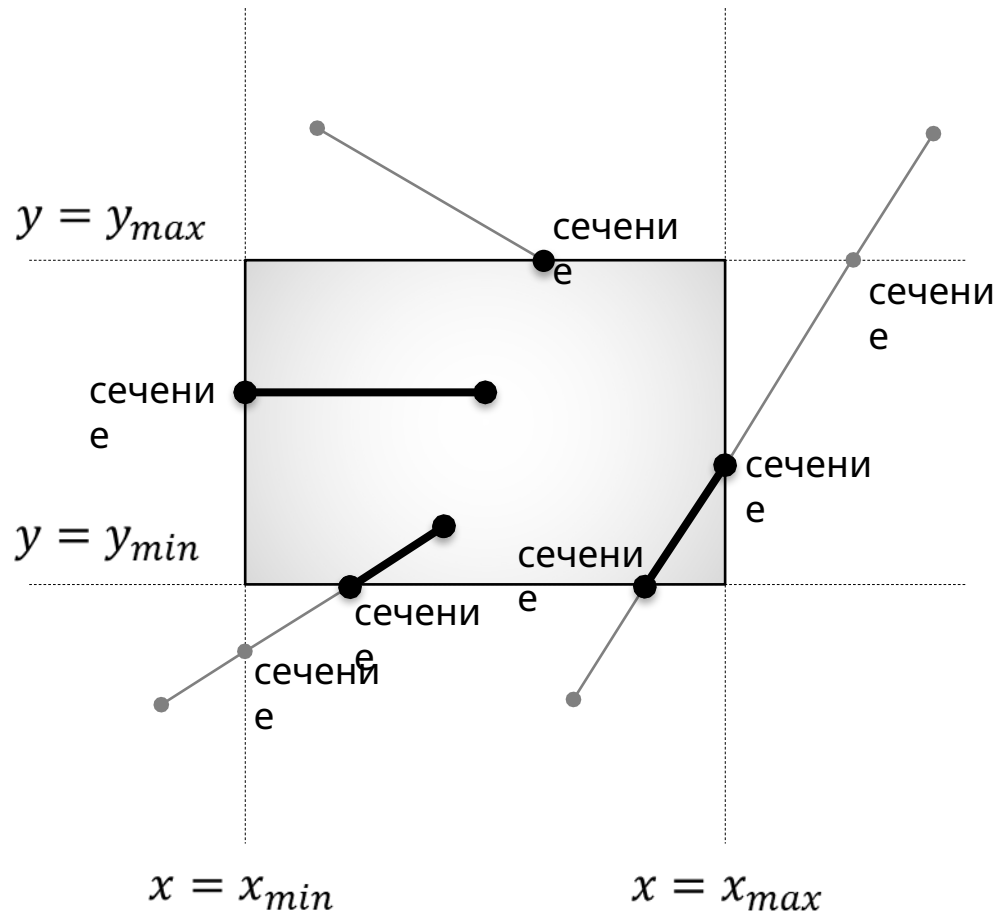
- Проверката за един пиксел е бърза
- Работи за всички примитиви
- Многооого излишни изчисления

Втори наивен алгоритъм

- Отсечките се секат от ограничителните линии
- Премахват се точките, които са външни
- Ако остават две точки – те са краища на видимата отсечка – иначе е или изцяло видима, или изцяло невидима

Особености

- Не зависи от дължината на отсечката
- Все пак прави излишни изчисления



Алгоритъм на Коеен-Съдърленд

Обща идея

Обща идея

- Разделя се равнината на 9 области с кодове
- Всеки край на отсечка получава код според това в коя област се намира
- По двата кода се разбира дали отсечката е вътрешна, външна или пресичаща видимата зона
- Ако е пресичаща, кодът указва как да се скъси
- След скъсяване остатъкът ѝ пак се проверява

Кодиране на областите

- Използва се 4-битова маска
- Девет области с уникален код, центърът е 0000
- Разпределението на битовете е по избор

Примерно ето така

- Бит 0 = 1, ако $y < y_{min}$
- Бит 1 = 1, ако $y > y_{max}$
- Бит 2 = 1, ако $x < x_{min}$
- Бит 3 = 1, ако $x > x_{max}$

0110	0010	1010
	--1-- --0--	
-1-- -0-- 0100	0000	1000
	---0 ---1	0--- 1---
0101	0001	1001

Свойства на кода

Дадено

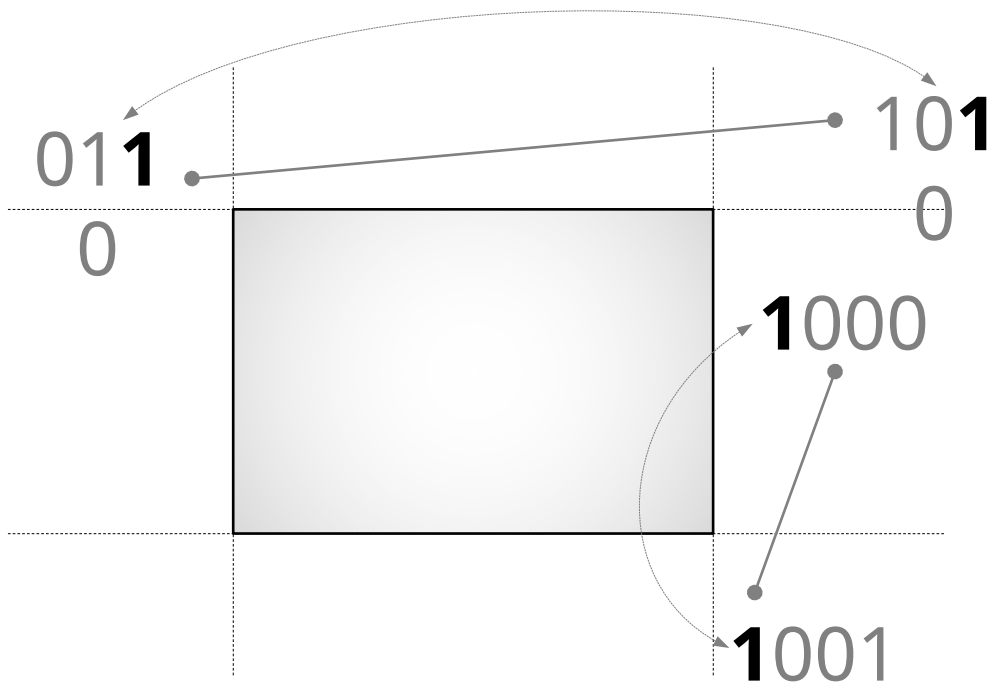
- Краищата на отсечка са P и Q с кодове p и q

Бързи проверки с битови операции

- Ако $p|q = 0$ отсечката е вътрешна
($p = 0$ и $q = 0 \Rightarrow P$ и Q са във видимата зона \Rightarrow цялата отсечка е видима)
- Ако $p \& q \neq 0$ отсечката е външна
(p и q имат общ ненулев бит $\Rightarrow P$ и Q са едновременно отвъд някоя права \Rightarrow цялата отсечка е невидима)

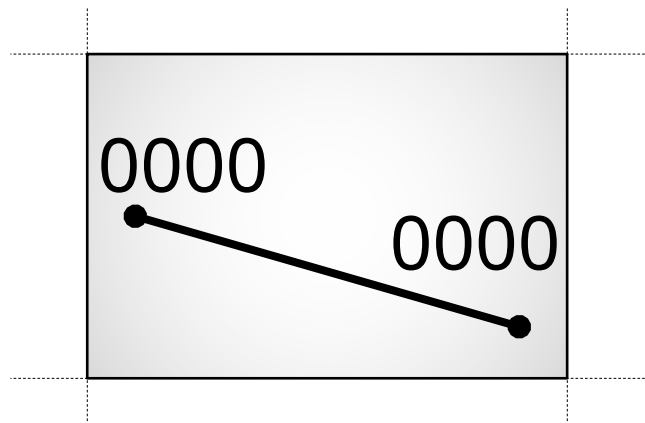
Изцяло външна отсечка

- двата ѝ кода имат поне по един общ ненулев бит



Изцяло вписана отсечка

- и двата кода на краищата ѝ са 0000



После какво?

- Краищата имат кодове без общи ненулеви битове
- Отсечката потенциално пресича видимата зона

Разделя се на две части, такива че:

- Едната част е изцяло външна и отпада от раз
- За другата част си проверяват пак кодовете

Как се прави това?

- Единият край има ненулев код (защо? 1m бонус)
- Избира се ненулев бит и се изрязва спрямо него
- Така се получават две отсечки
- Едната е външна и се забравя веднага
- Другата се тества пак с $p|q = 0$ и $p\&q \neq 0$

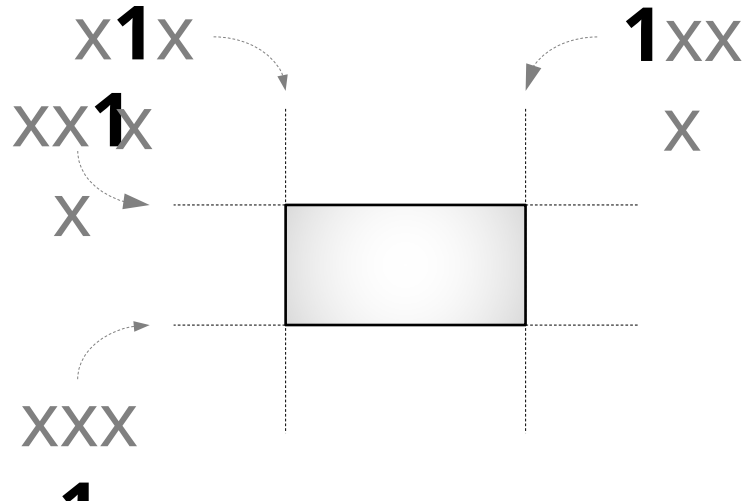
Резултат

- С краен брой стъпки се отсича отсечката до видимата зона

Изрязване спрямо бит

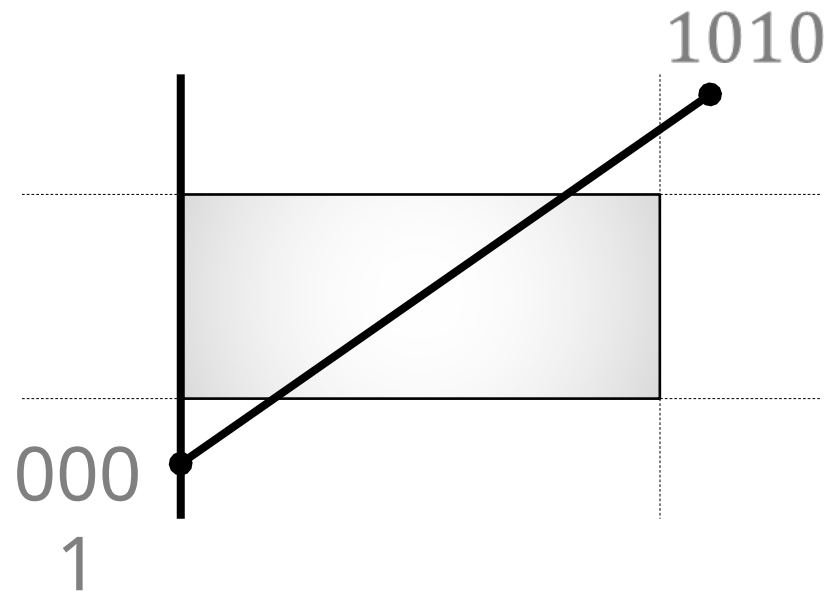
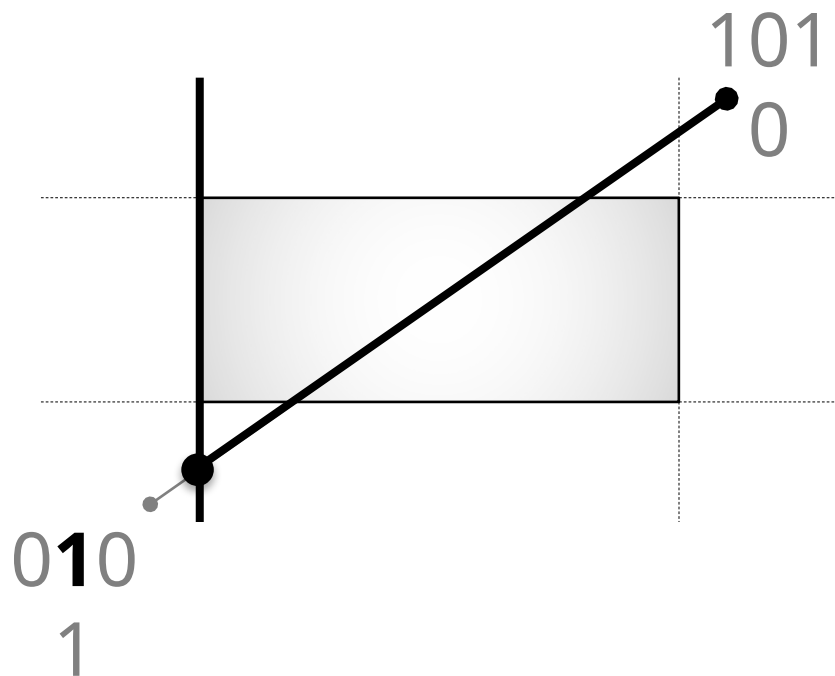
Определяне на правата за отсичане

- Всеки бит е за зона извън дадена права
- По тази права се отсича



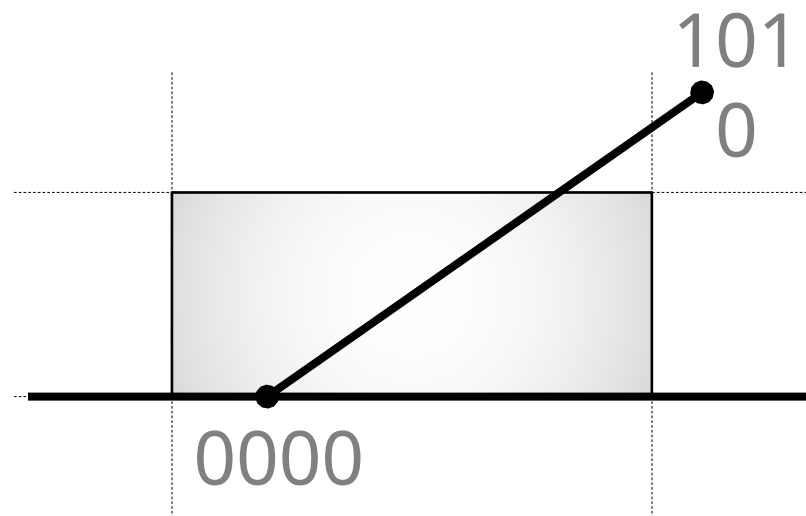
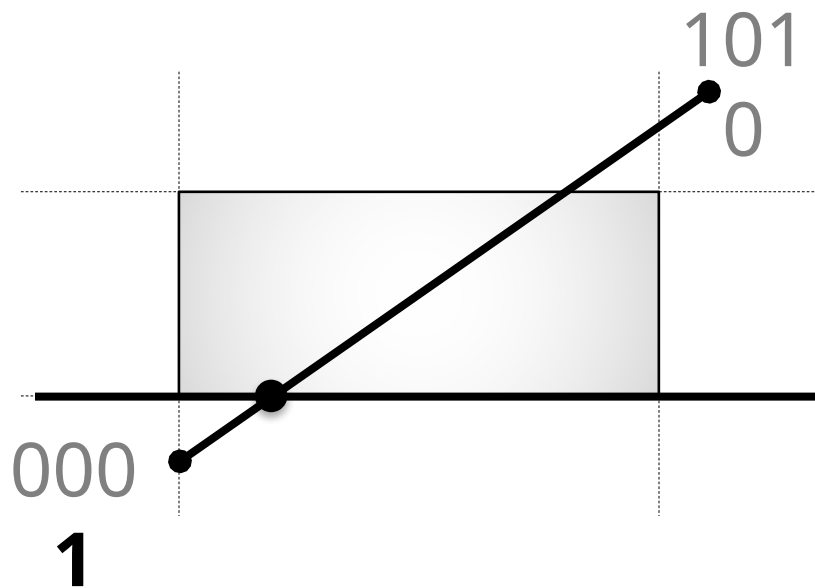
Примери за отсичане

- По избран бит се отсича по съответната му права



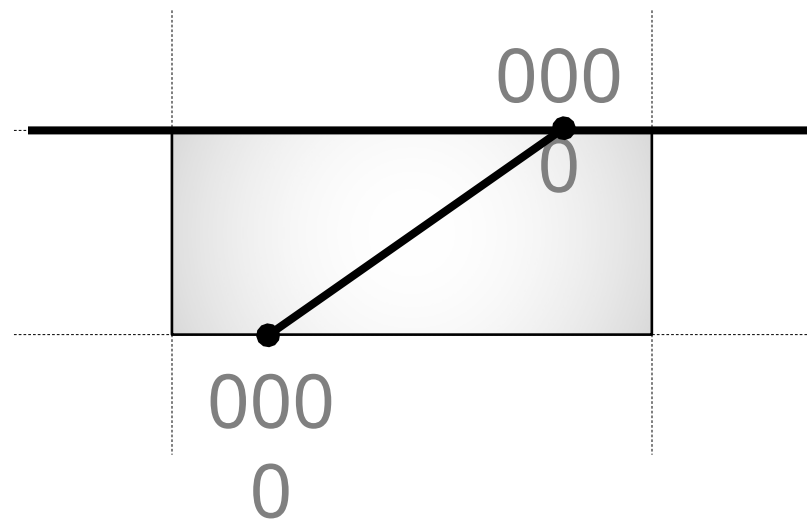
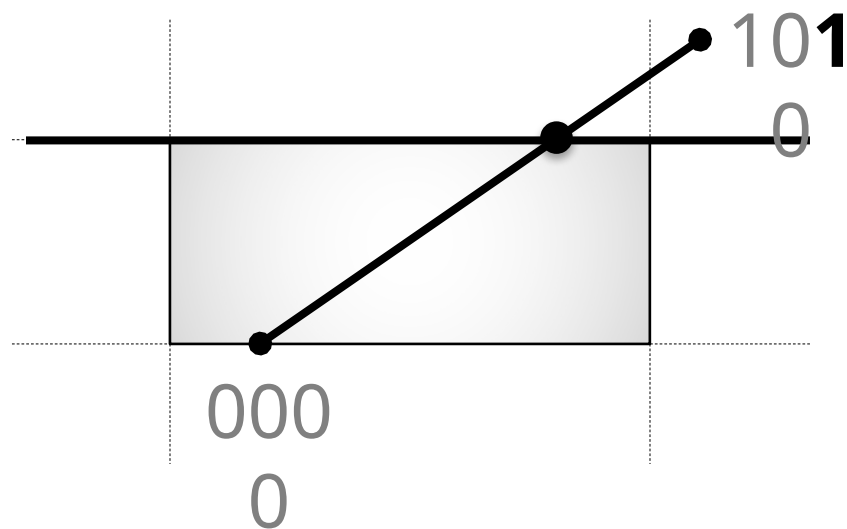
Второ отсичане

- Отсича се по следващия ненулев бит



Следващо отсичане

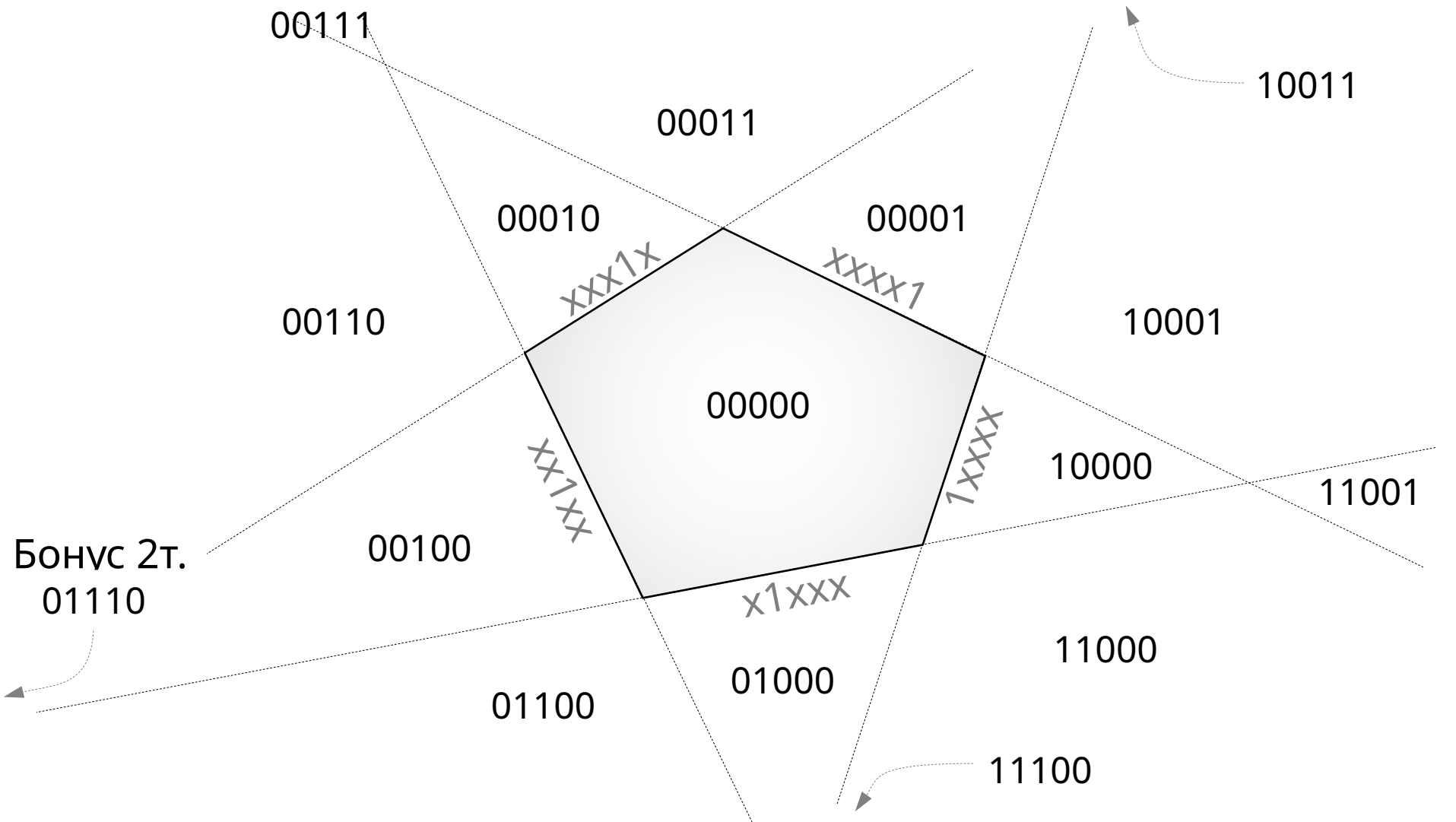
- Сега и от другия край на отсечката
- До получаване на $p|q = 0$ или $p \& q \neq 0$



Многоъгълно изрязване

Приложение на алгоритъма за многоъгълна зона

- Толкова битови числа, колкото ъгълен е многоъгълникът
- Отново централната зоната е с нулев код 000...0
- Броят зони зависи от броя стени и от ориентацията им

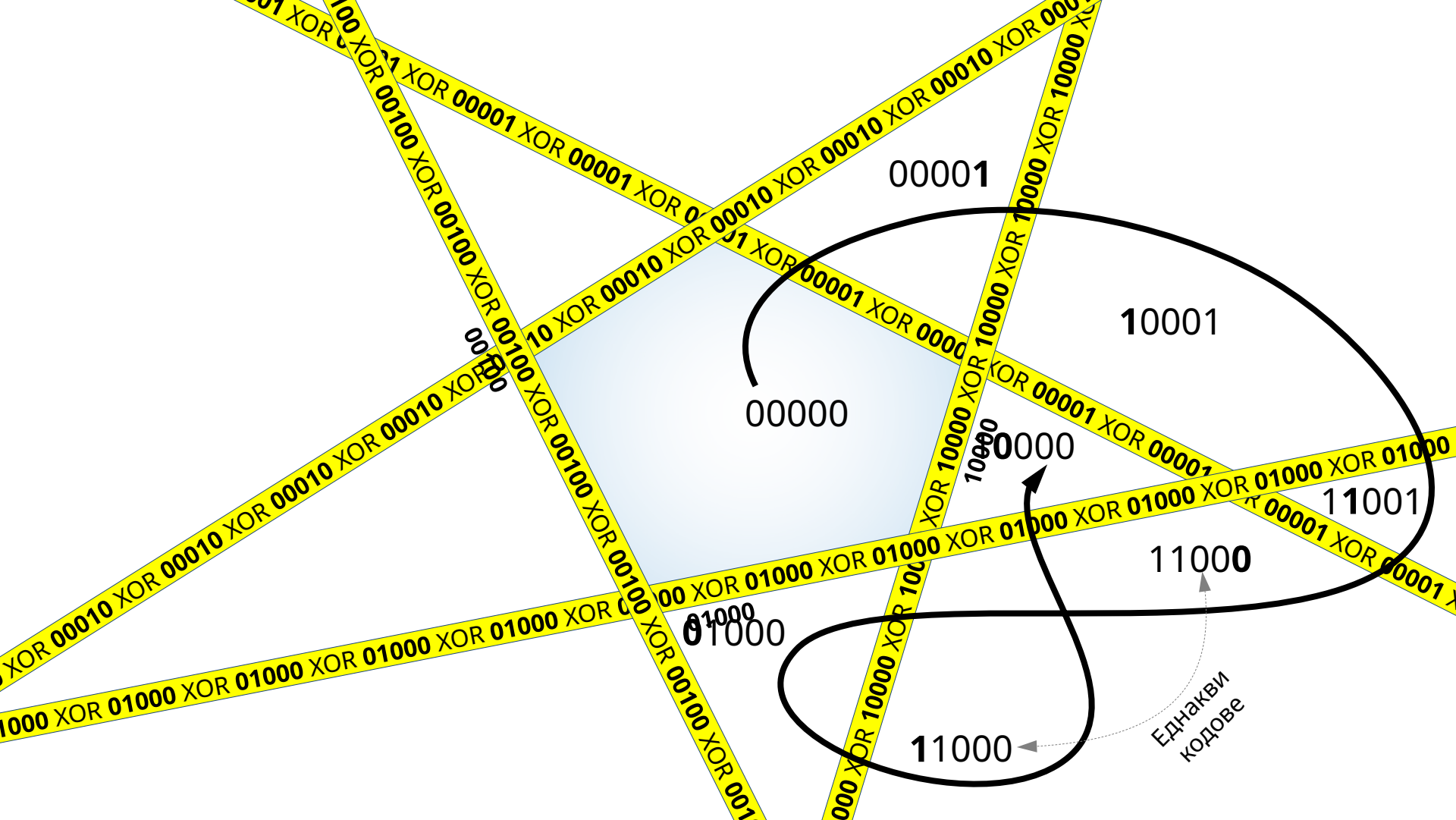


Свързаност на зоните

- Кодове на две съседни зони се различават по един бит
- Този бит е на правата между тях

Обхождане на зони

- Тръгване от централната зона с код **0**
- При всяко пресичане на права се *xor*ва съответният бит, т.е. **0** \rightarrow **1** и **1** \rightarrow **0**
(Ако сме умни, ще минем без **1** \rightarrow **0**. Защо? Още 1 точка бонус)



В 3D

Алгоритъм на Коеен-Съдърленд за многостени

- Изрязване на отсечка от многостен
- Всеки бит определя равнина, а не права

Пример

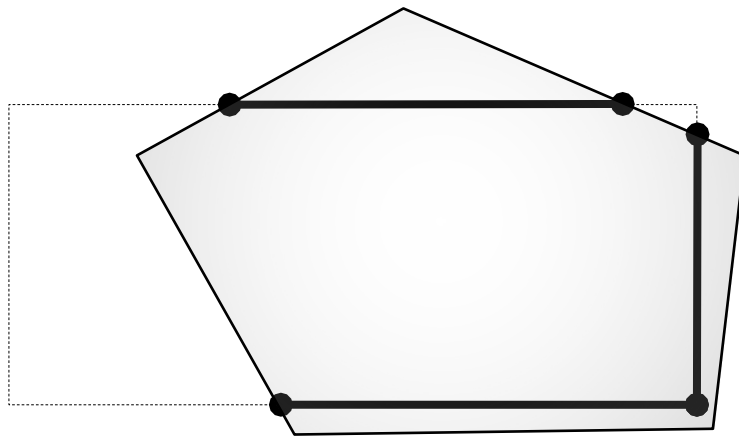
- За изрязване спрямо куб ще трябва 6-битови маски

Отсичане на многоъгълник

Отсичане на многоъгълник

Основна разлика

- Отсичането може да доведе до включването на отсечки, които не са по контура на многоъгълника



Много алгоритми за отсичане на многоъгълник

- На Съдърленд-Ходжман (Sutherland-Hodgman)
- На Вайлер (Weiler)
- На Лиан(г)-Барски (Liang-Barsky)
- На Мейлот (Maillot)
- На Вати (Vatti)
- На Грайнер-Хорман (Greiner-Hormann)

Алгоритъм чрез отсичане на отсечка по Коеен-Съдърленд

Първичен алгоритъм

Вариант на алгоритъм

- Чрез отсичане на отсечка по Коеен-Съдърленд

Особености

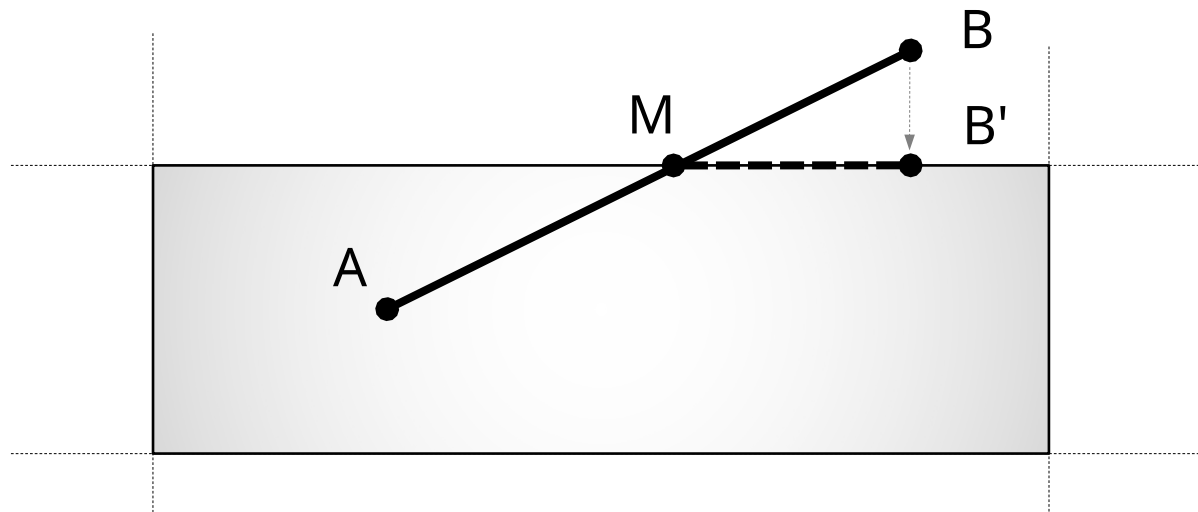
- Лесен за реализация
- Може да доведе до дублирани върхове
- Може да доведе до контурни точки
- Може да доведе до фалшиви страни

Основна идея на алгоритъма

- Страните на многоъгълника се обхождат една по една
- Всяка се отсича чрез алгоритъма на Кoen-Sьдърленд
- Отсечените части не се изтриват, а се долепят до съответната отсичаща права

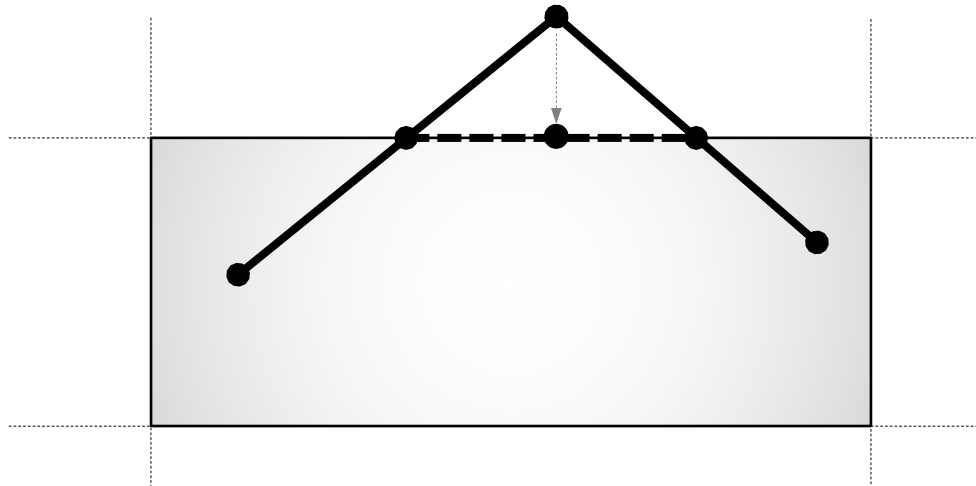
Обработване на отсечките

- AM се запазва, а MB се свлича до MB'



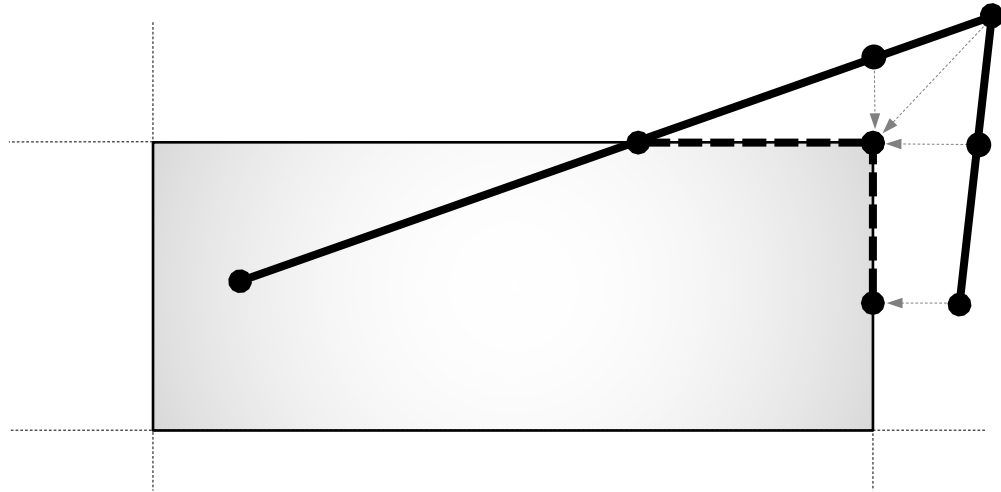
Получаване на контурни точки

- Могат да се елиминират в допълнителен пас (ако е необходимо)



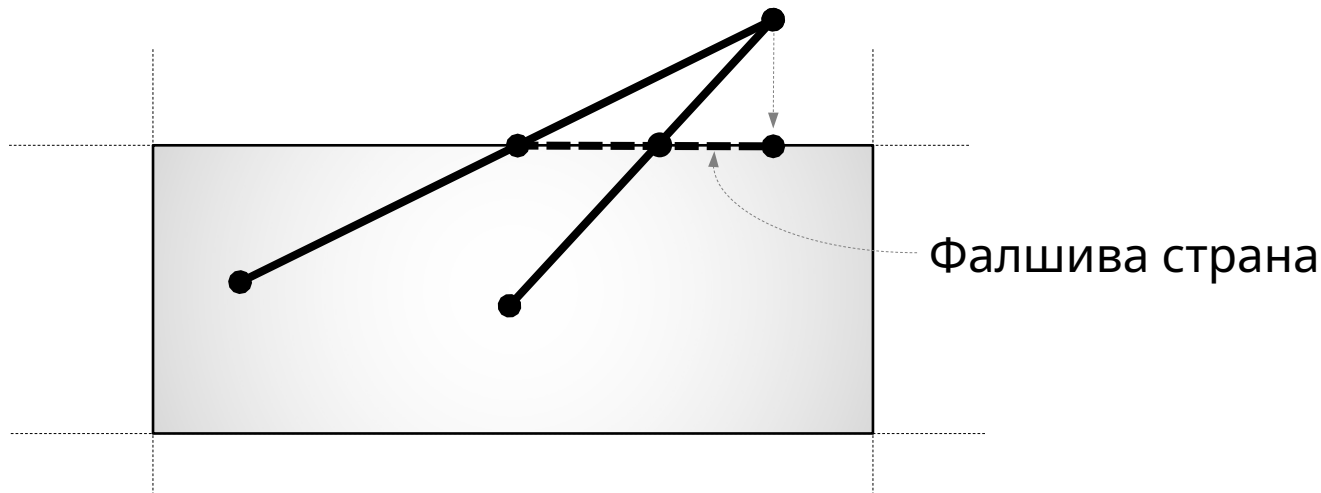
Получаване на дублирани върхове

- Могат да се елиминират в допълнителен пас (ако е необходимо)



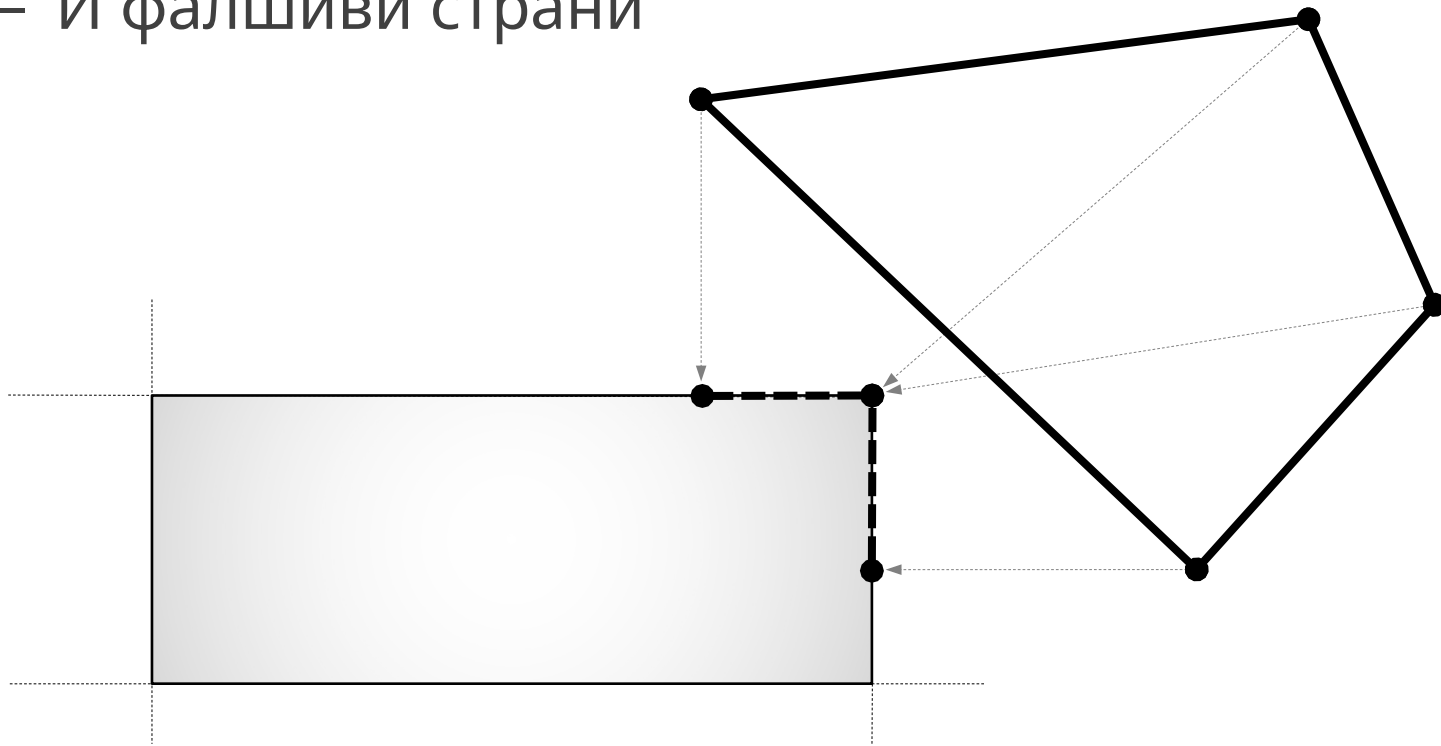
Получаване на фалшиви страни

- Могат да се елиминират в допълнителен пас (ако е необходимо)



Гъдел при външни многоъгълници

- Само контурни и дублирани точки
- И фалшиви страни



Алгоритъм на Съдърленд-Ходжман

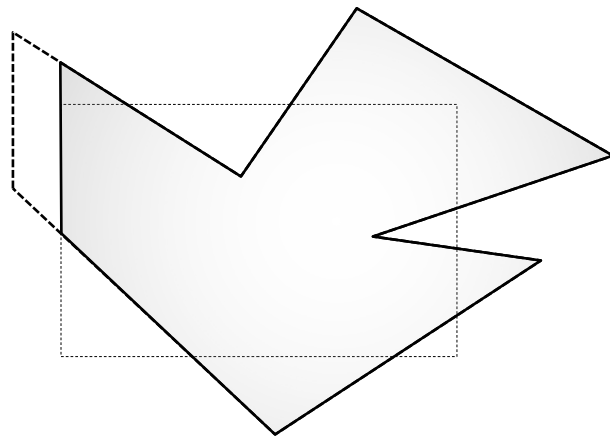
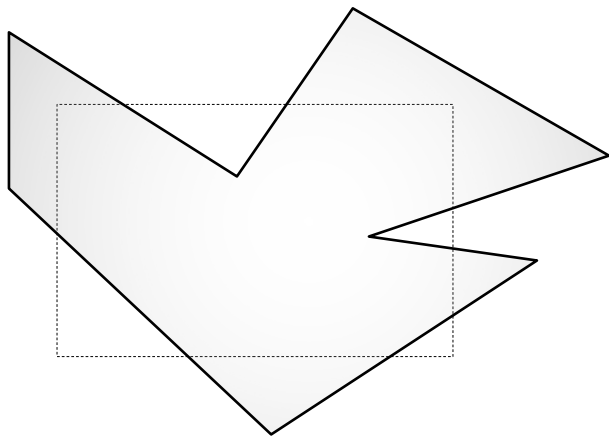
Основна идея

Основна идея на алгоритъма

- Отсича се многоъгълник по страна на видимата зона
- После по друга страна и т.н.
- Нужно е различаване вътрешност-външност на двойка точки

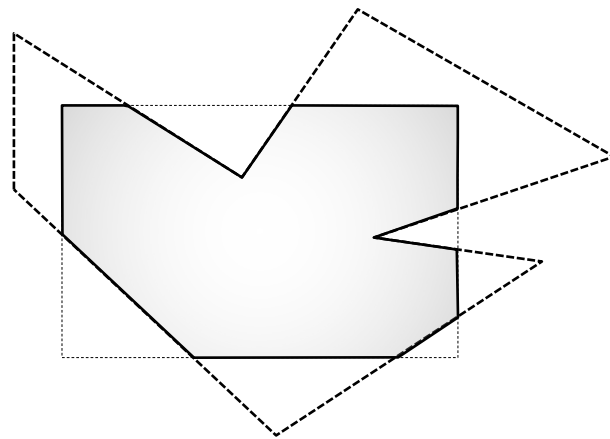
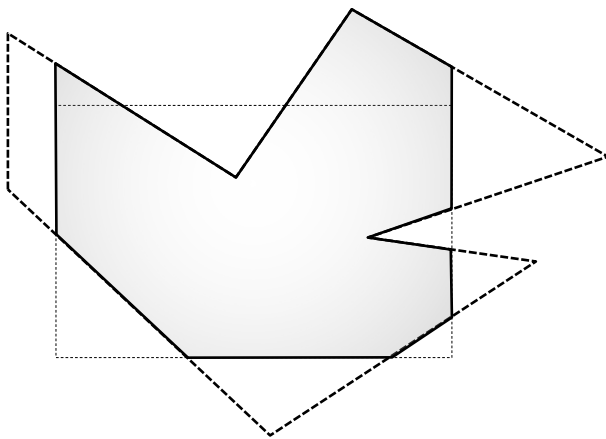
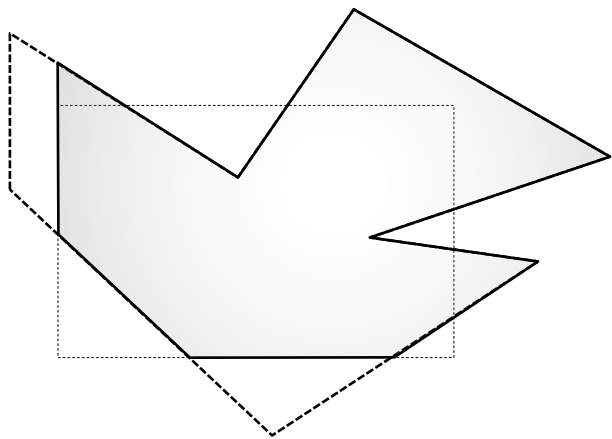
Първа стъпка

- Отсичане отляво (например)



Отсичане по другите страни

- Долу, дясно, горе
- В произволен ред

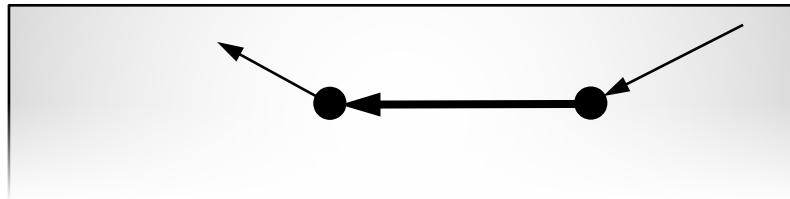


Алгоритъм на отсичане

- Обхождат се страните на многоъгълника
- За всяка страна се определя дали влиза или излиза от видимата зона
- По време на обхождането се генерират върховете на отсечения многоъгълник
- Използват се четири прости правила

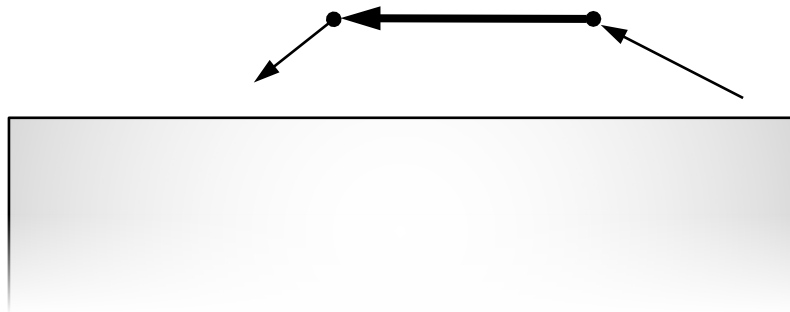
Двете точки са вътре

- И двете участват в резултата



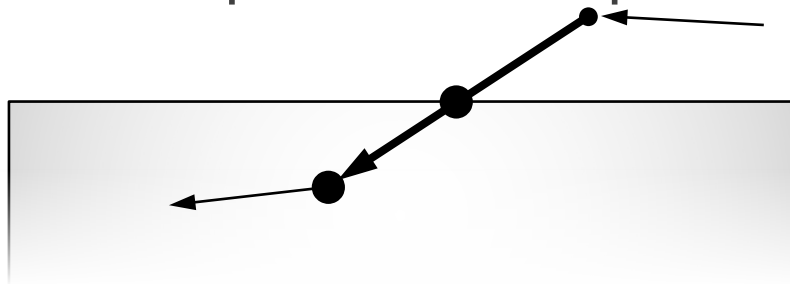
Двете точки са вън

- Нито една не участва в резултата



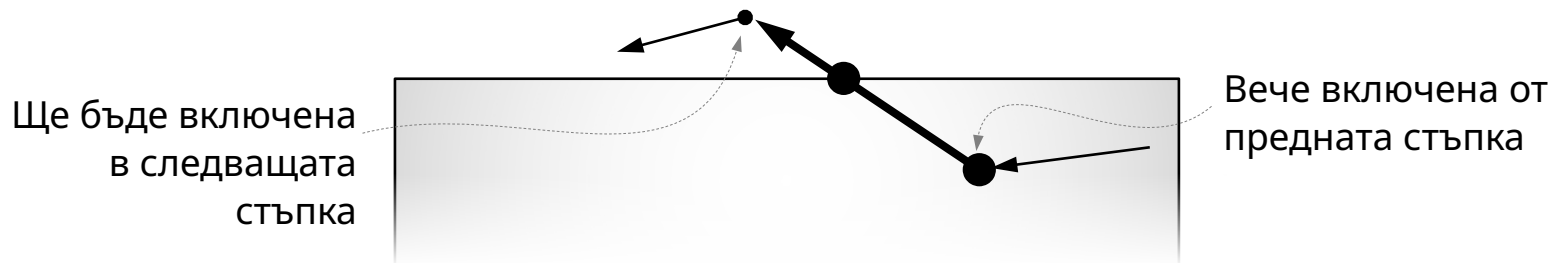
Влизаме във видимата зона

- Участват вътрешната и пресечната



Излизаме от видимата зона

- Включва се само пресечната точка



Въпроси?

Повече информация

LUKI	стр. 86-109
AGO2	стр. 69-109
ALZH	гл. 4.4-4.5
KLAW	стр. 59-65
LASZ	стр. 122-128
MORT	стр. 296-299

А също и:

- Polygon Clipping

<http://www.codeguru.com/cpp/misc/misc/graphics/article.php/c8965/Polygon-Clipping.htm>

Край