

ИКТ В НОС

Интерактивность

Тема №15

Работа с мишка



Събития и интерактивност

- Събитията са обекти свързани с DOM
- Първоначално разгледани в Тема №6
- Използват се за реализиране на интерактивност

Чрез събития се обработват

- Движения с мишката
- Кликвания с бутоните
- Използване на клавиатурата

Събития на мишката



Движение на мишката

- **mousemove** – движение
- **mouseenter** – навлизане в елемент
- **mouseleave** – излизане от елемент
- **mouseover** – движение над елемент или подчинените му
- **mouseout** – движение навън от елемент или подчинените му

Работа с бутоните

- **mousedown** – натискане на бутон
- **mouseup** – пускане на бутон
- **click** – кликване
- **dblclick** – двойно кликване
- **contextmenu** – кликване с десен бутон за меню

Други събития

- Не са пряко свързани с графика:
 - Събития при влачене на елементи и файлове
 - Събития за контролиране на мултимедия



Обект на събитието

- Всяко събитие се описва с JS обект
- Свойствата му дават информация за събитието

Свойства

- **target** – DOM елемент, където е станало събитието
- **clientX**, **clientY** – координати спрямо прозореца
- **screenX**, **screenY** – координати спрямо екрана
- **buttons** – кои бутони са натиснати
- **altKey**, **ctrlKey**, **shiftKey** – дали са натиснати Alt, Ctrl, Shift



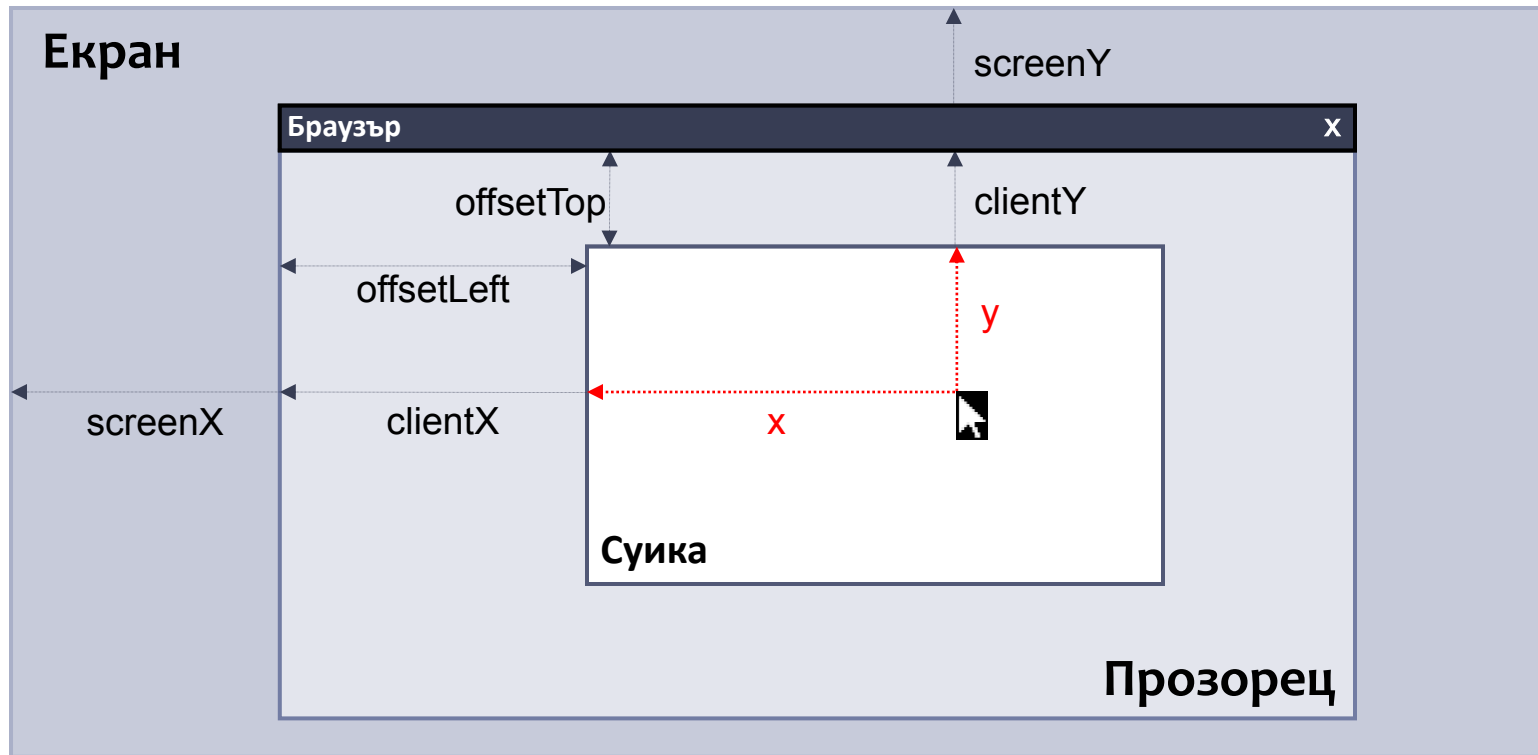
Локални координати

- Имаме графичен елемент canvas
- Искаме координати (x,y) на курсора на мишката спрямо горния ляв ъгъл на обект Suica

$x = \text{clientX} - \text{offsetLeft}$

$y = \text{clientY} - \text{offsetTop}$

$$x = \text{clientX} - \text{offsetLeft}$$
$$y = \text{clientY} - \text{offsetTop}$$



Пример



Празно графично поле

- При движение на мишката показваме координатите ѝ
- Те са спрямо графичното поле
- При движение извън полето не показваме координати

Идея

- Използваме две събития
 - `mousemove` – движение в графичното поле
 - `mouseout` – излизане от графичното поле

Добавяне на слушатели на събития

- В елемента **info** ще се показват координатите
- Всеки Suica обект съдържа WebGL обекта **gl**, който помни в себе си DOM елемента **canvas**, с който има връзка
- Създаваме двата слушателя към **p.gl.canvas**
- Движението на мишката ще се обработва от **mousemove**
- Напускането на графичното поле – от **mouseout**

```
info = document.getElementById('info');
```

```
p = new Suica();
```

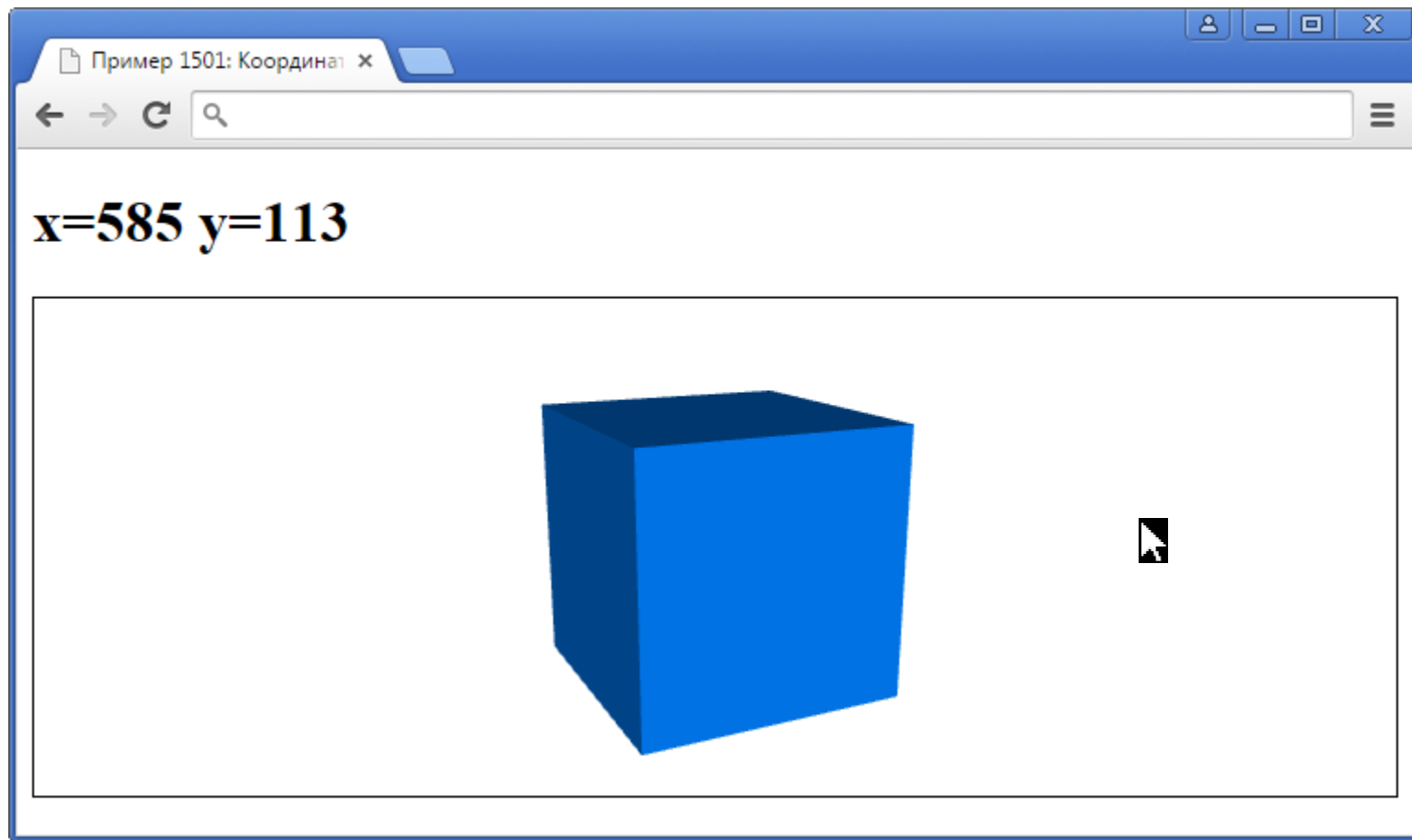
```
p.gl.canvas.addEventListener('mouseout', mouseout, false);
```

```
p.gl.canvas.addEventListener('mousemove', mousemove, false);
```

Обработване на събитията

- От параметъра **event** извличаме координатите на мишката и отместването на графичното поле **target**

```
function mouseMove(event)
{
    var x = event.clientX-event.target.offsetLeft;
    var y = event.clientY-event.target.offsetTop;
    info.innerHTML = 'x='+x+' y='+y;
}
function mouseOut(event)
{
    info.innerHTML = 'Пример 1501:... ';
}
```



ПРОБА

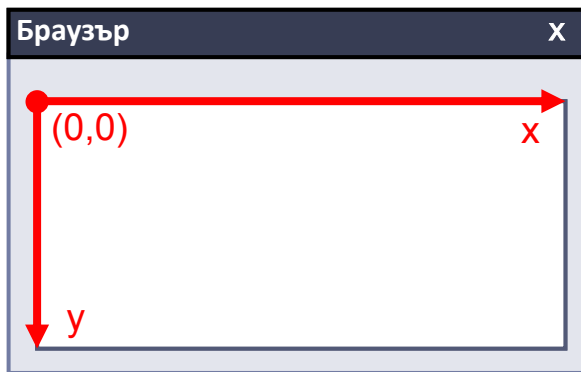
Рисуване с мишка

Графични координати

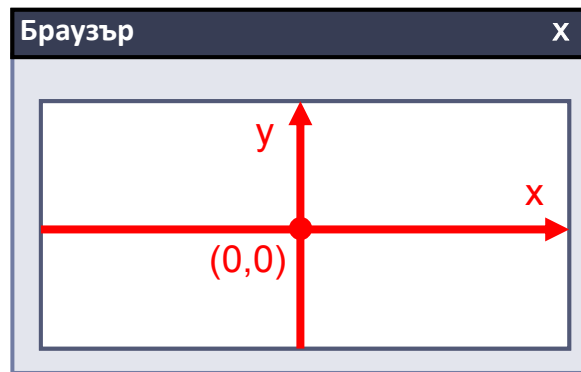


Графични координати

- Координати на обектите в графичното поле
- Разминават се с координатите на мишката
- Най-често $(0,0)$ е центърът, X е надясно, а Y е нагоре



Координати на мишката

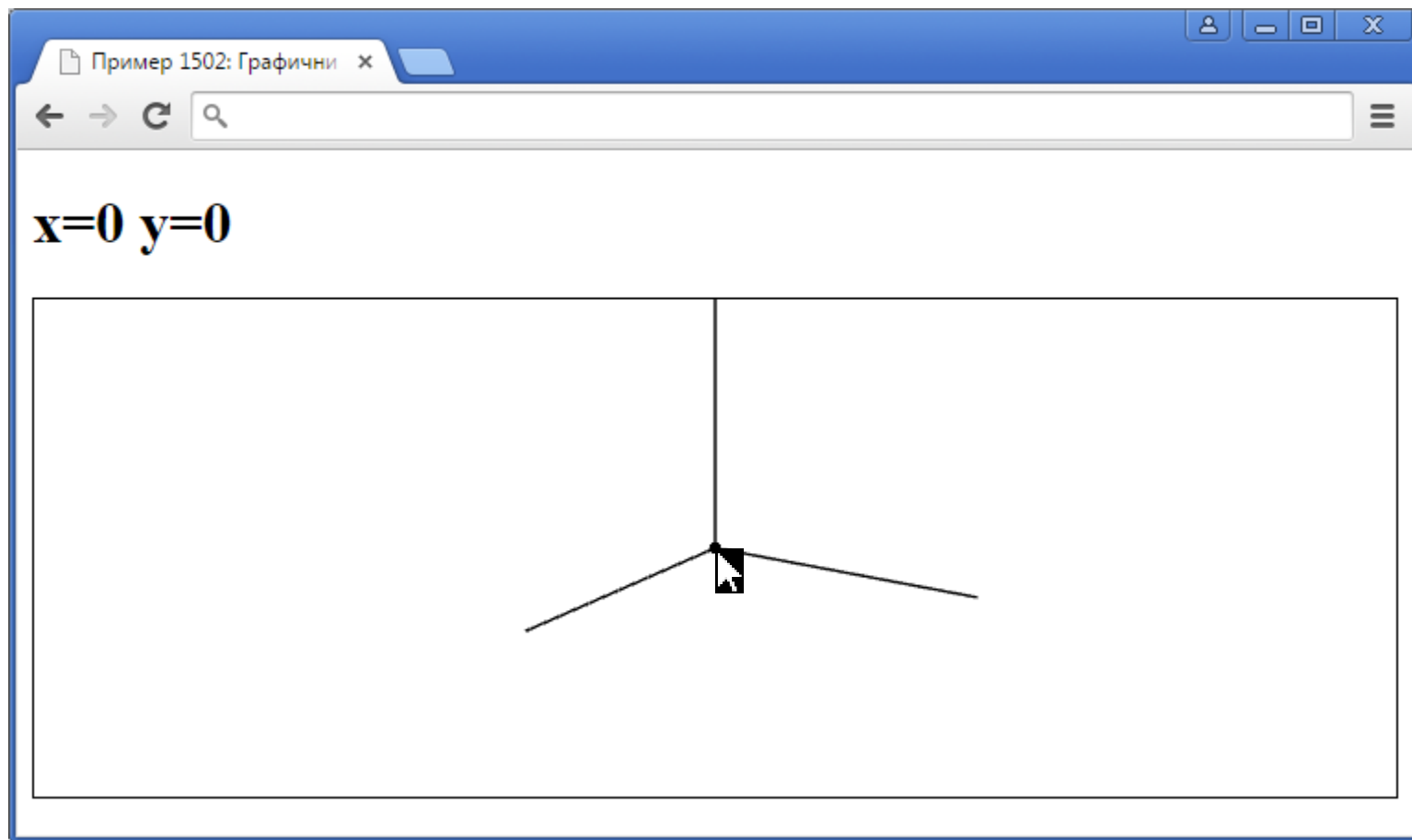


Графични координати

Изчисляване на графични координати

- Отместване на центъра с половината от ширината `offsetWidth` и височината `offsetHeight` на графичното поле
- Координатите по Y трябва да са с обрънат знак

```
var x = event.clientX  
        - event.target.offsetLeft  
        - event.target.offsetWidth/2;  
var y = -(event.clientY  
        - event.target.offsetTop  
        - event.target.offsetHeight/2);
```

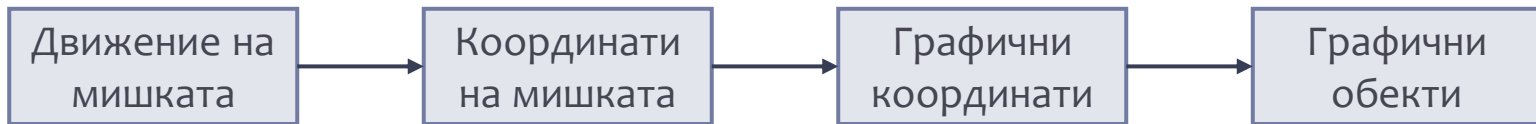


ПРОБА



Рисуване на точки

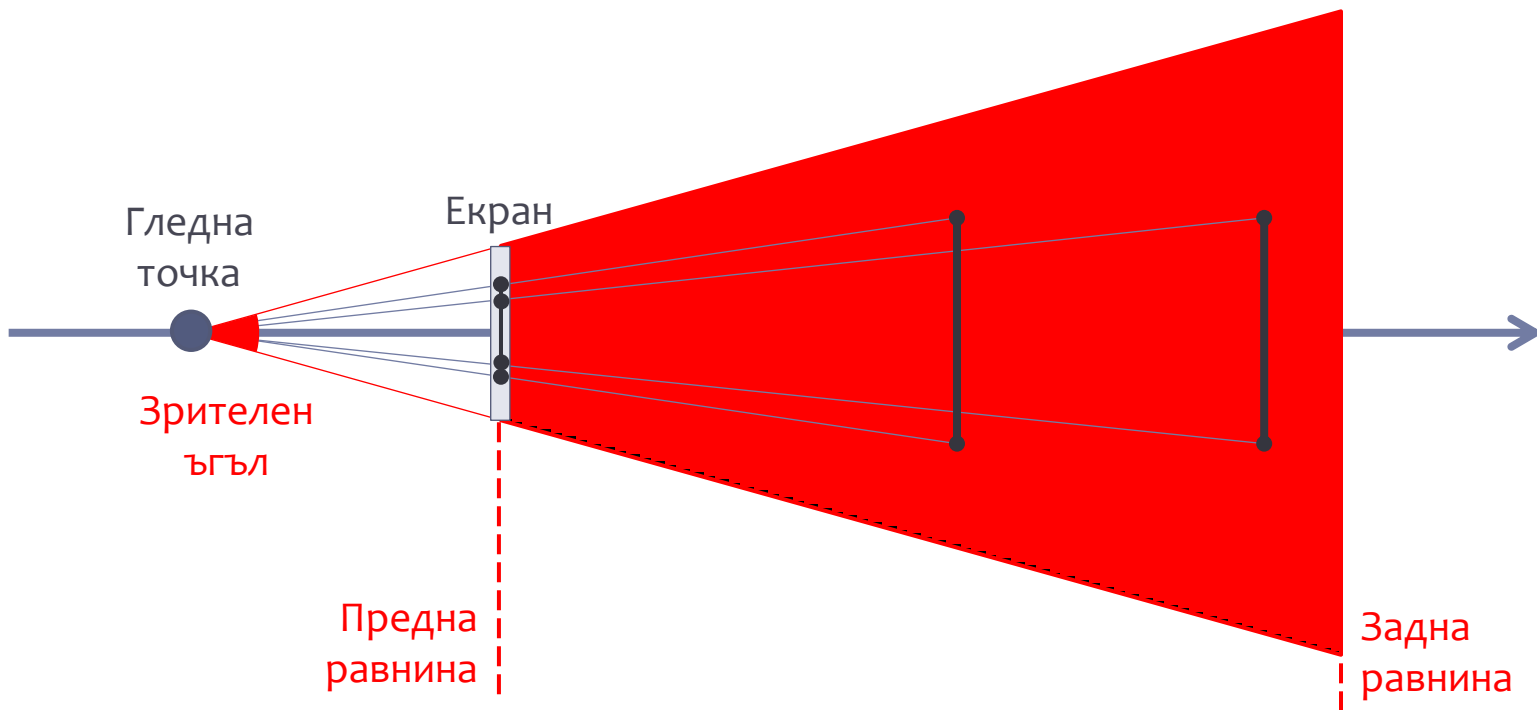
- При движение на мишката тя оставя следа от точки



- Използване на 2D сцена с гледна точка „отгоре“
- Съответствие 1:1 между координатите на мишката и координатите в графичното поле

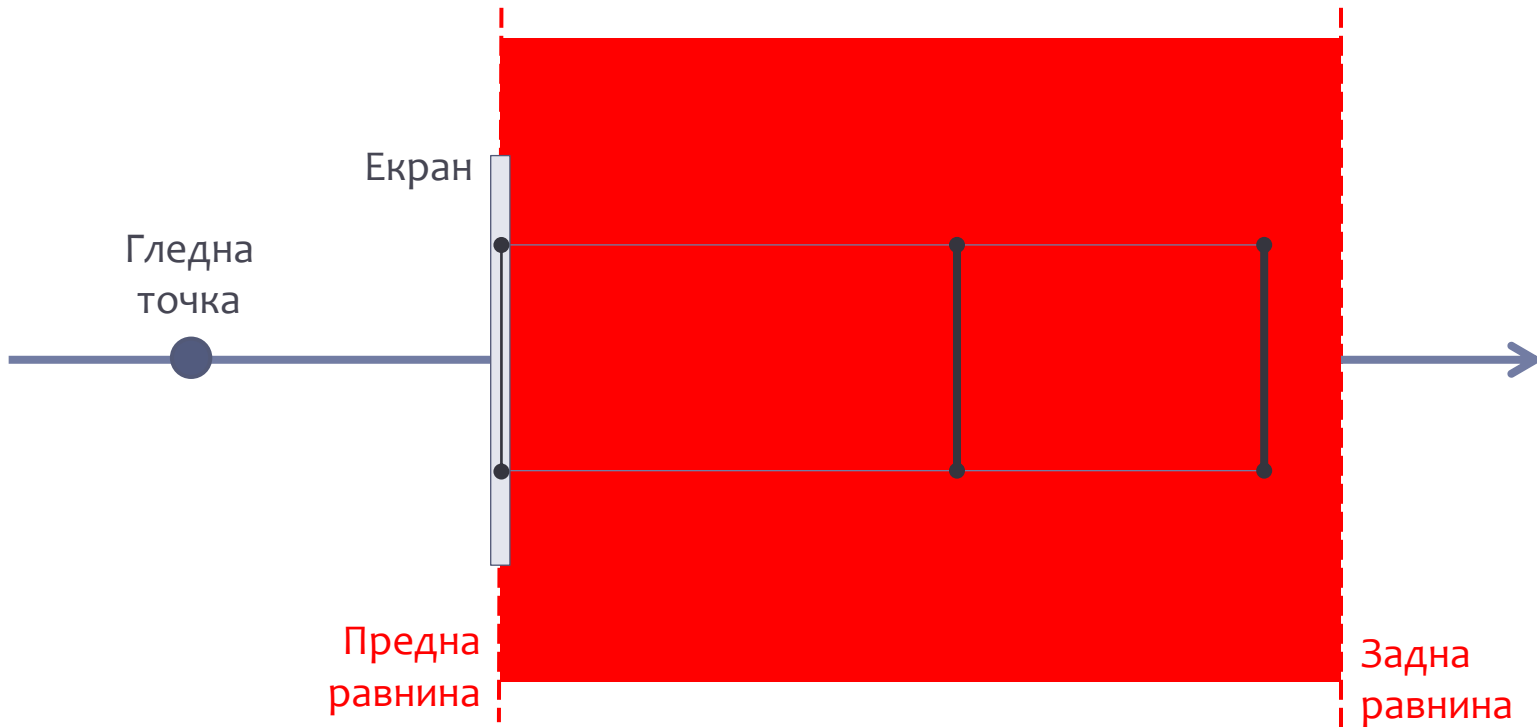
Перспективна проекция

- По-отдалечените графични обекти изглеждат по-малки
- Функция **perspective** (ъгъл, от, до)



Ортографска проекция

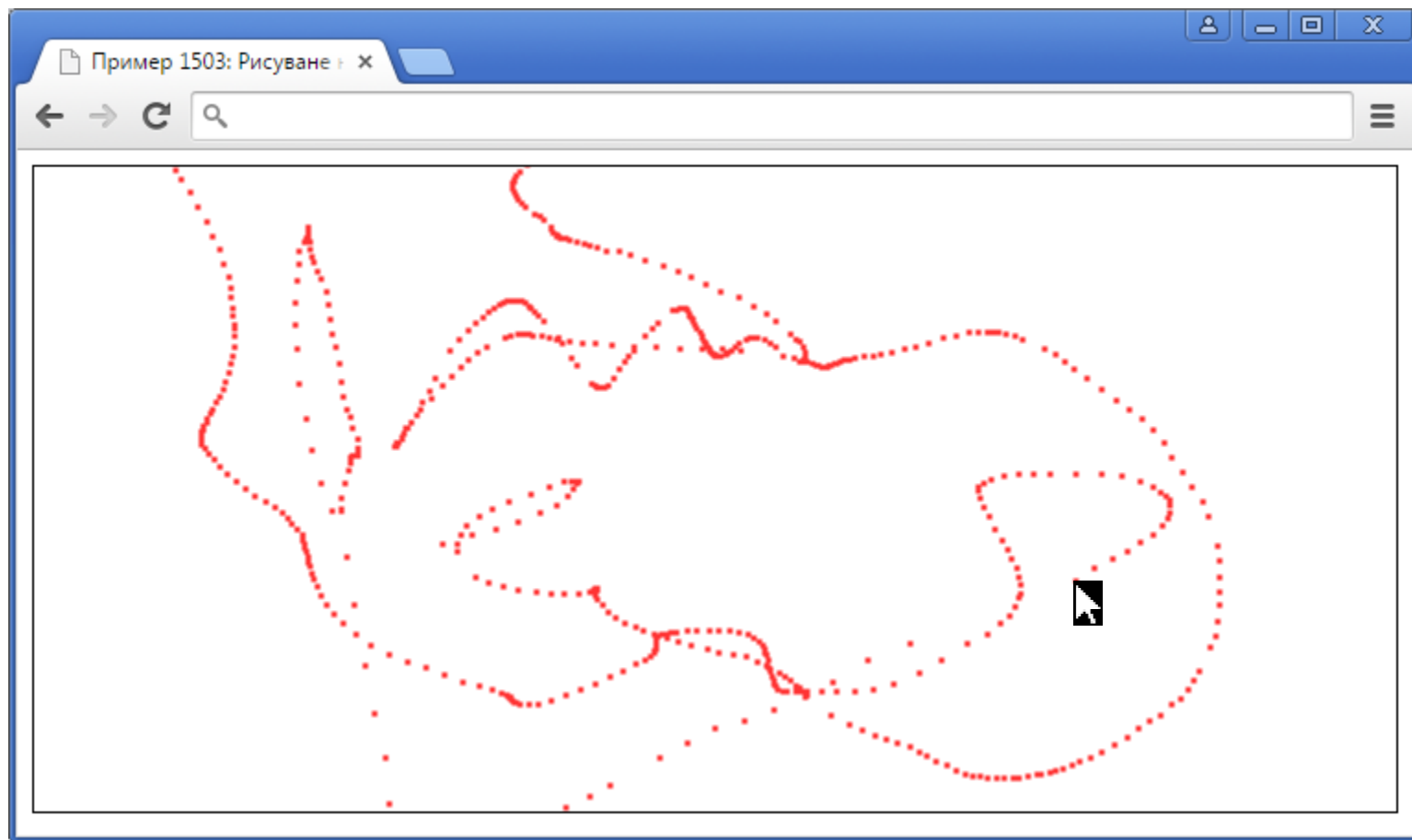
- По-отдалечените графични обекти си запазват размера
- Функция **orthographic** (от, до)



Реализация на рисуване

- За съответствие 1:1 използваме ортографска проекция
- За поглед „отгоре“ използваме **lookAt**, чийто параметри казват: гледаме от $[0,0,1]$ към $[0,0,0]$, а $[0,1,0]$ сочи нагоре
- Намираме графичните координати и създаваме точка

```
orthographic(-2,2);  
lookAt([0,0,1],[0,0,0],[0,1,0]);  
...  
function mouseMove(event)  
{  
    var x = event.clientX - ...;  
    var y = -(event.clientY - ...);  
    point([x,y,0]);  
}
```



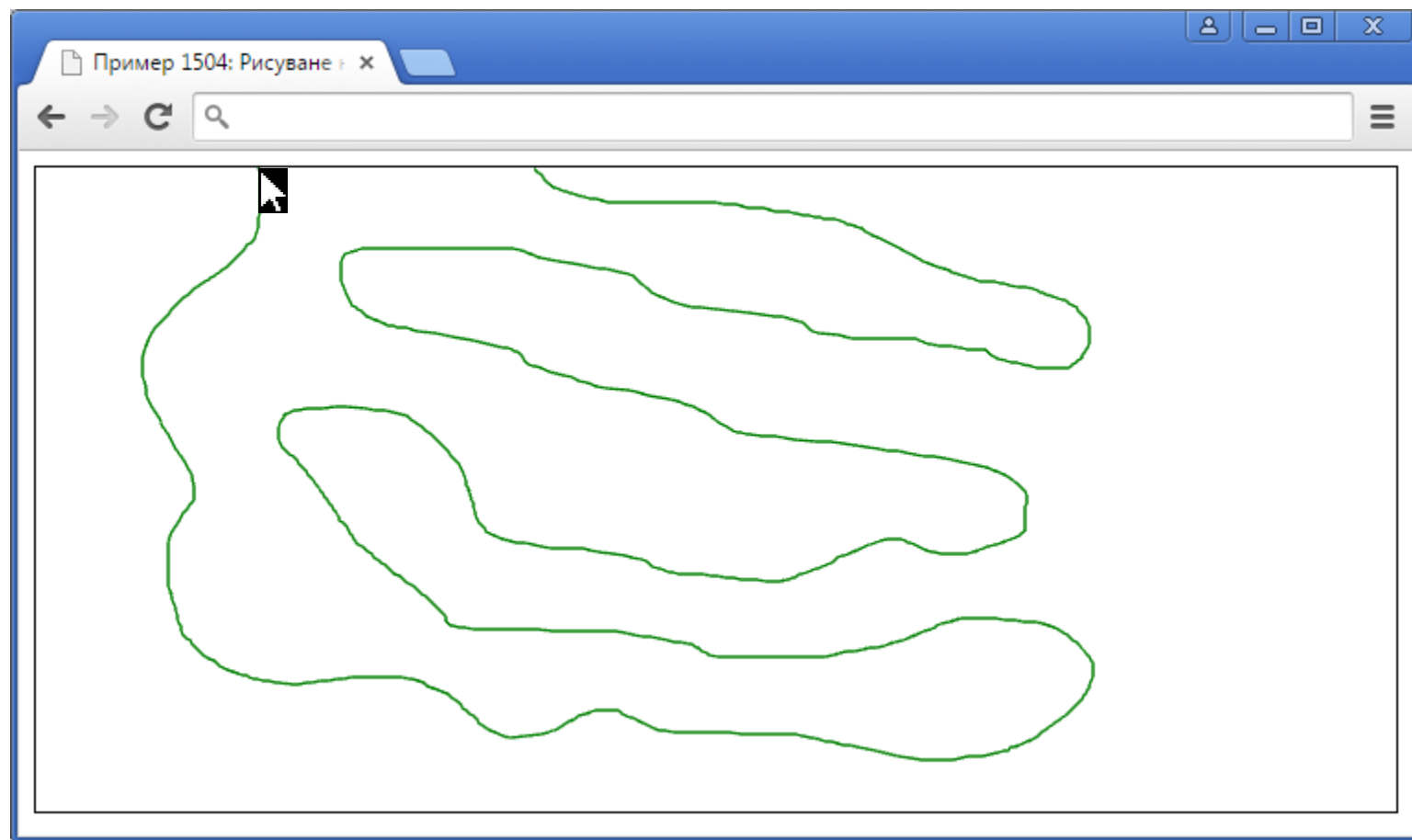
ПРОБА

Рисуване на линии

- Запомняме в **last** последната точка
- Ако в **last** има стойност, рисуваме отсечка
- Ако няма стойност, то сме в началото и все още нямаме две точки, за да направим отсечка

```
var last;

function mouseMove(event)
{
    ...
    if (last) segment(last,[x,y,0]);
    last = [x,y,0];
}
```



ПРОБА

Промяна в рисуването

- Рисуването да започва с натискане на ляв бутон
- Рисуването да свършва с пускането на ляв бутон

Идея

- При натискане влизаме в режим на рисуване
- При пускане излизаме от режима на рисуване
- При движение рисуваме, само ако сме в режим на рисуване

Реализация

- Следим за три събития

```
...addEventListener('mousemove', mouseMove, false);  
...addEventListener('mousedown', mouseDown, false);  
...addEventListener('mouseup', mouseUp, false);
```

- В **mouseUp** излизаме от режим рисуване и забравяме последната запомнена точка

```
draw = false;  
last = undefined;
```

- В **mouseDown**, ако е натиснат левият бутон, влизаме в режим рисуване и запомняме текущата точка

```
if (event.buttons==1)
{
    ...
    draw = true;
    last = [x,y,0];
}
```

- А **mouseMove** е с проверка дали сме в режим на рисуване

```
if (draw)
{
    ...
    if (last) segment(last,[x,y,0]);
    last = [x,y,0];
}
```

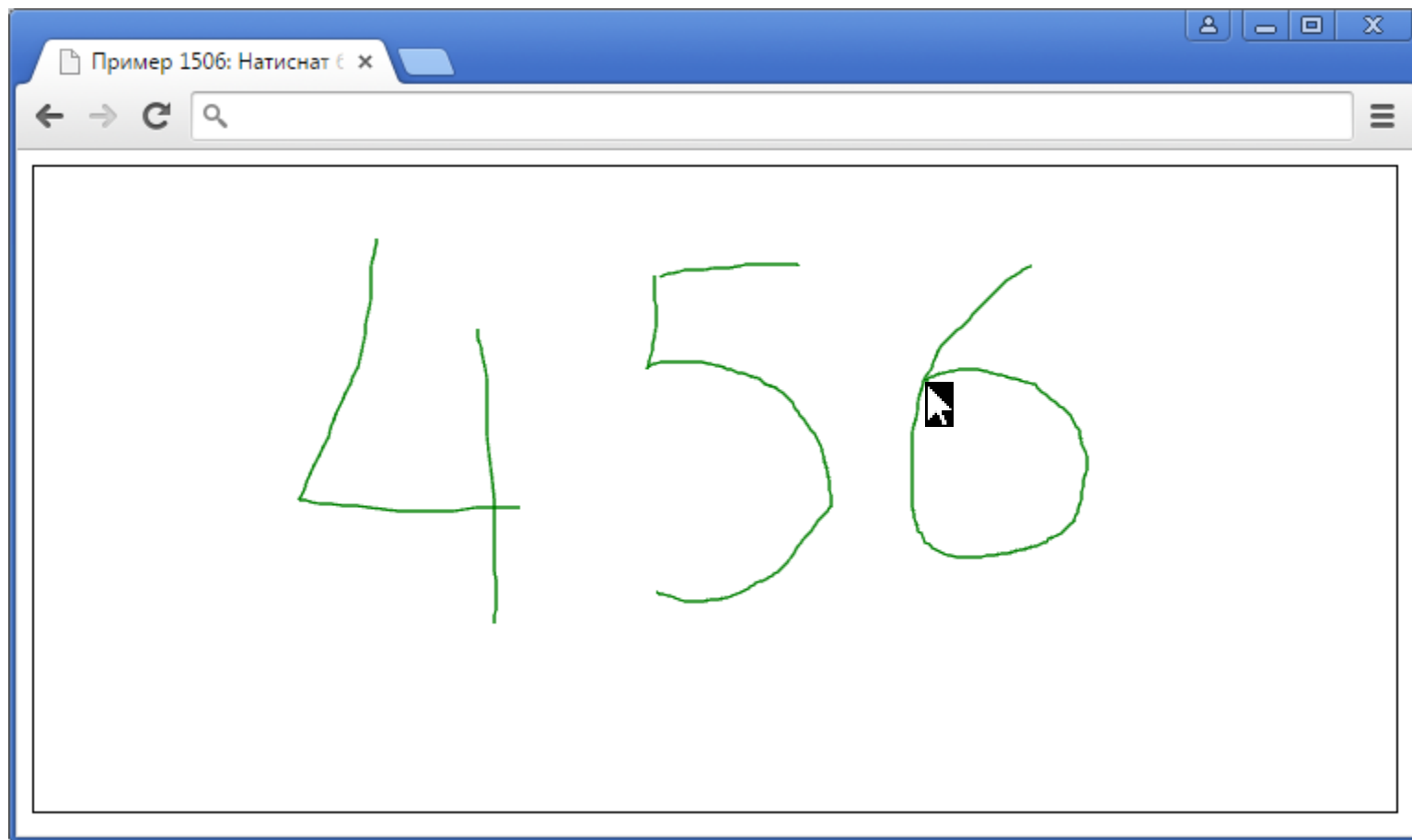


ПРОБА

По-кратък вариант?

- Последната запомнена точка **last** е флаг за рисуване
- Ако има стойност – рисуваме, иначе не рисуваме
- Не се интересуваме от събитията **mousedown** и **mouseup**, а само от **mousemove**

```
if (event.buttons==1)
{
    ...
    if (last) segment(last,[x,y,0]);
    last = [x,y,0];
}
else
    last = undefined;
```



ПРОБА

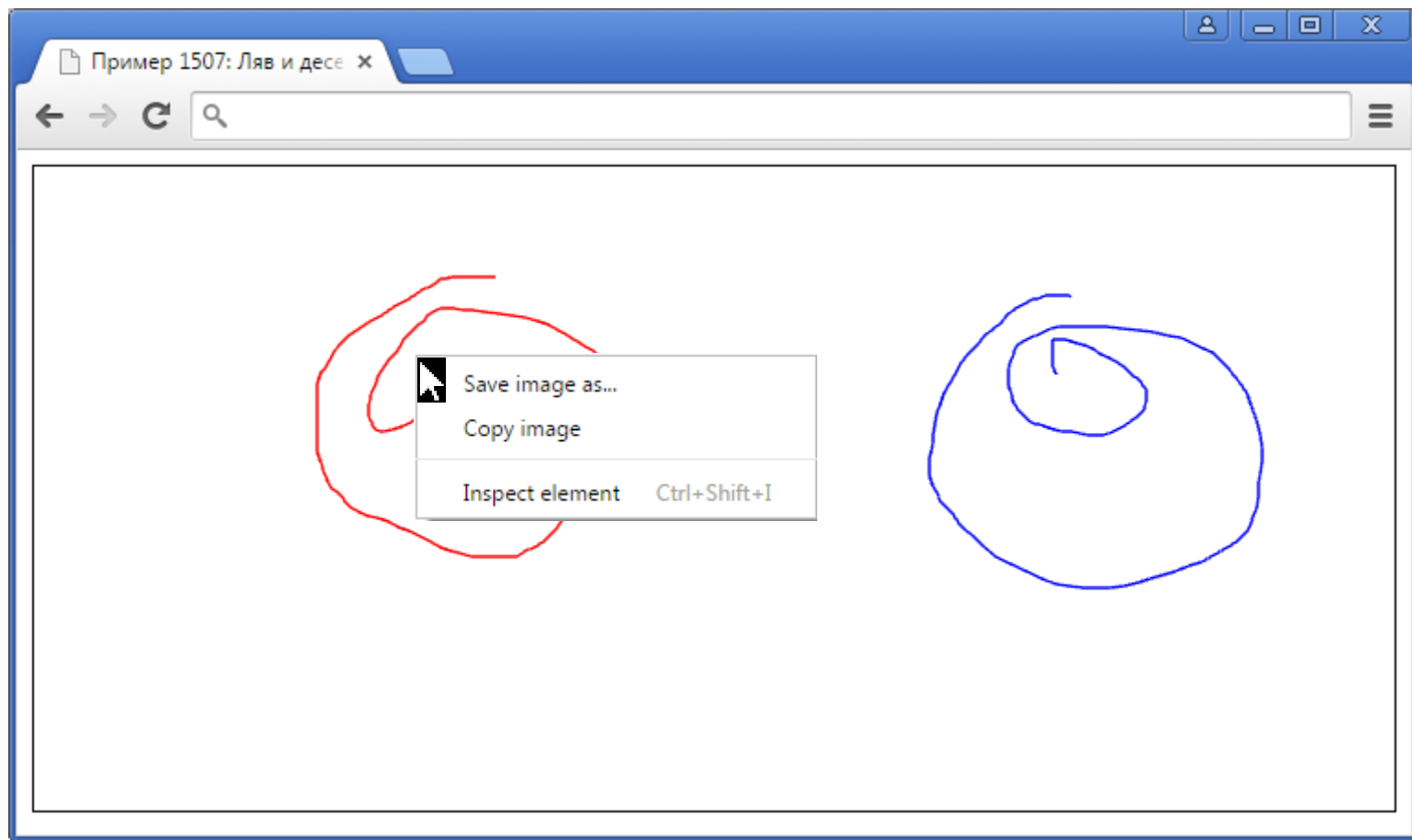
Ляв и десен бутон

- С ляв бутон рисуваме синя линия, с десен – червена линия

Реализация

- Рисуваме при натиснат какъвто и да е бутон
- В **style** помним цвета според кой бутон точно е натиснат

```
if (event.buttons)
{
    ...
    var style = {color:event.buttons==1?[0,0,1]:[1,0,0]};
    if (last) segment(last,[x,y,0]).custom(style);
    last = [x,y,0];
}
```

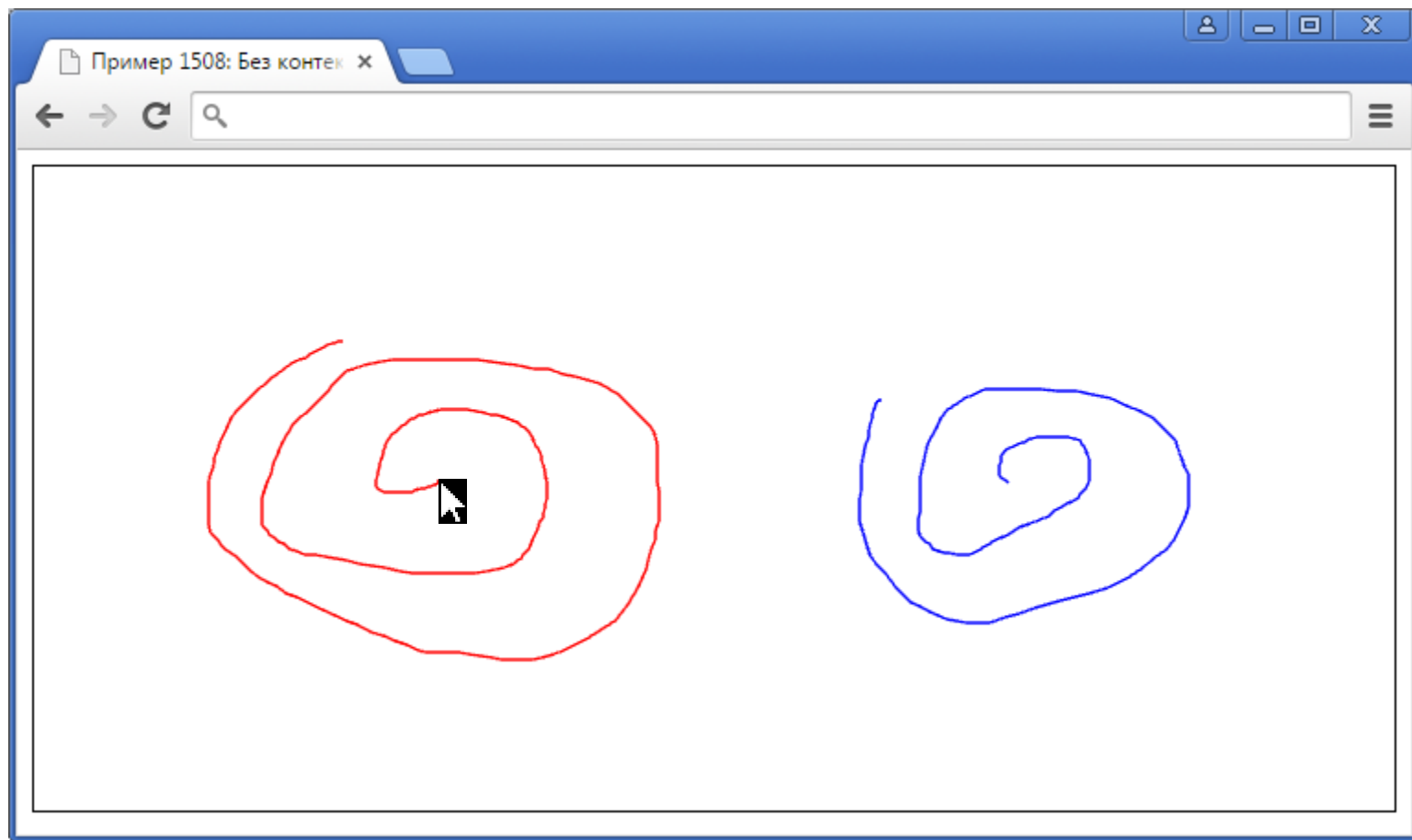


ПРОБА

Контекстно меню

- Проблем – с десен бутон се показва контекстно меню
- За да премахнем показването на контекстното меню, трябва да следим за събитието **contextmenu**
- Когато настъпи, трябва да укажем с метода **preventDefault** да не се извършва подразбиращата се за бутона дейност (в случая това е показването на контекстното меню)

```
...addEventListener('contextmenu', contextMenu, false);  
  
function contextMenu(event)  
{  
    event.preventDefault();  
}
```

ПРОБА

Чертане с мишка



Задача

- С мишката да могат да се правят чертежи
- Елементите са точки, отсечки и окръжности

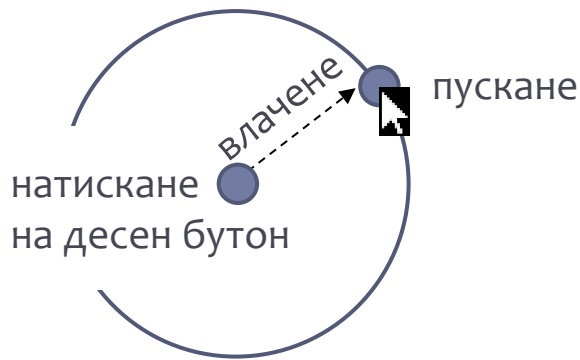
Рисуване на отсечки

- Натискане на бутон в единия край
- Влачене до другия край и пускане на бутона



Рисуване на окръжности

- Натискане на десен бутон в центъра
- Влачене до самата окръжност и пускане на бутона



Рисуване на точки

- Кликване с ляв бутон на мястото на точката



Помощна функция

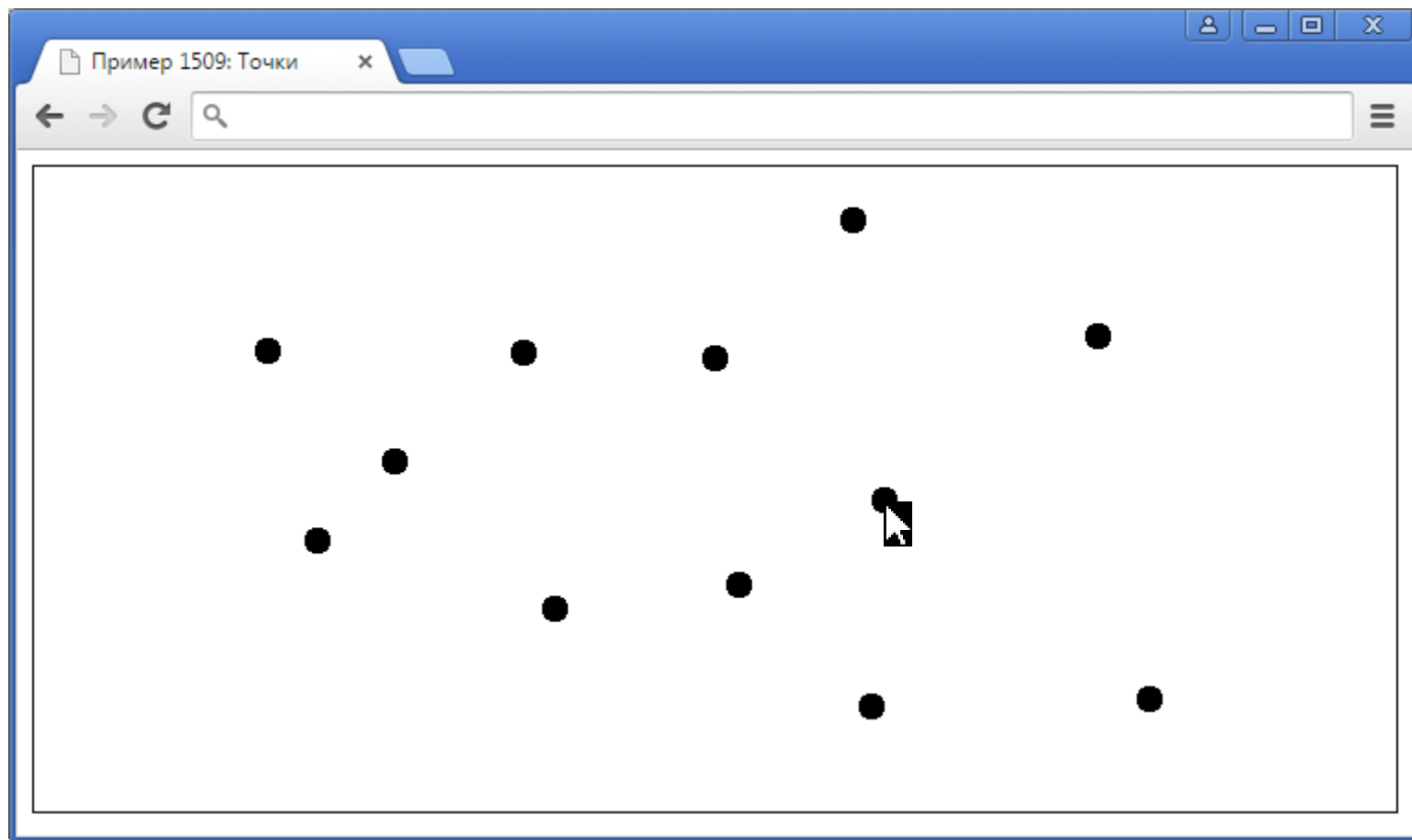
- Функция **mouseXY** изчислява графични координати, които съответстват на координатите на мишката за дадено събитие **event**

```
function mouseXY(event)
{
    var x = event.clientX
        - event.target.offsetLeft
        - event.target.offsetWidth/2;
    var y = -(event.clientY
        - event.target.offsetTop
        - event.target.offsetHeight/2);
    return [x,y,0];
}
```

Чертане на точки

- Улавяме натискането на бутон (който и да е)
- Генерираме обект точка на съответните координати
- За удобство стилът на всички точки е изнесен в глобалната променлива **pointStyle**

```
pointStyle = {color:[0,0,0], pointSize:14.5};  
  
function mouseDown(event)  
{  
    point(mouseXY(event)).custom(pointStyle);  
}
```



ПРОБА

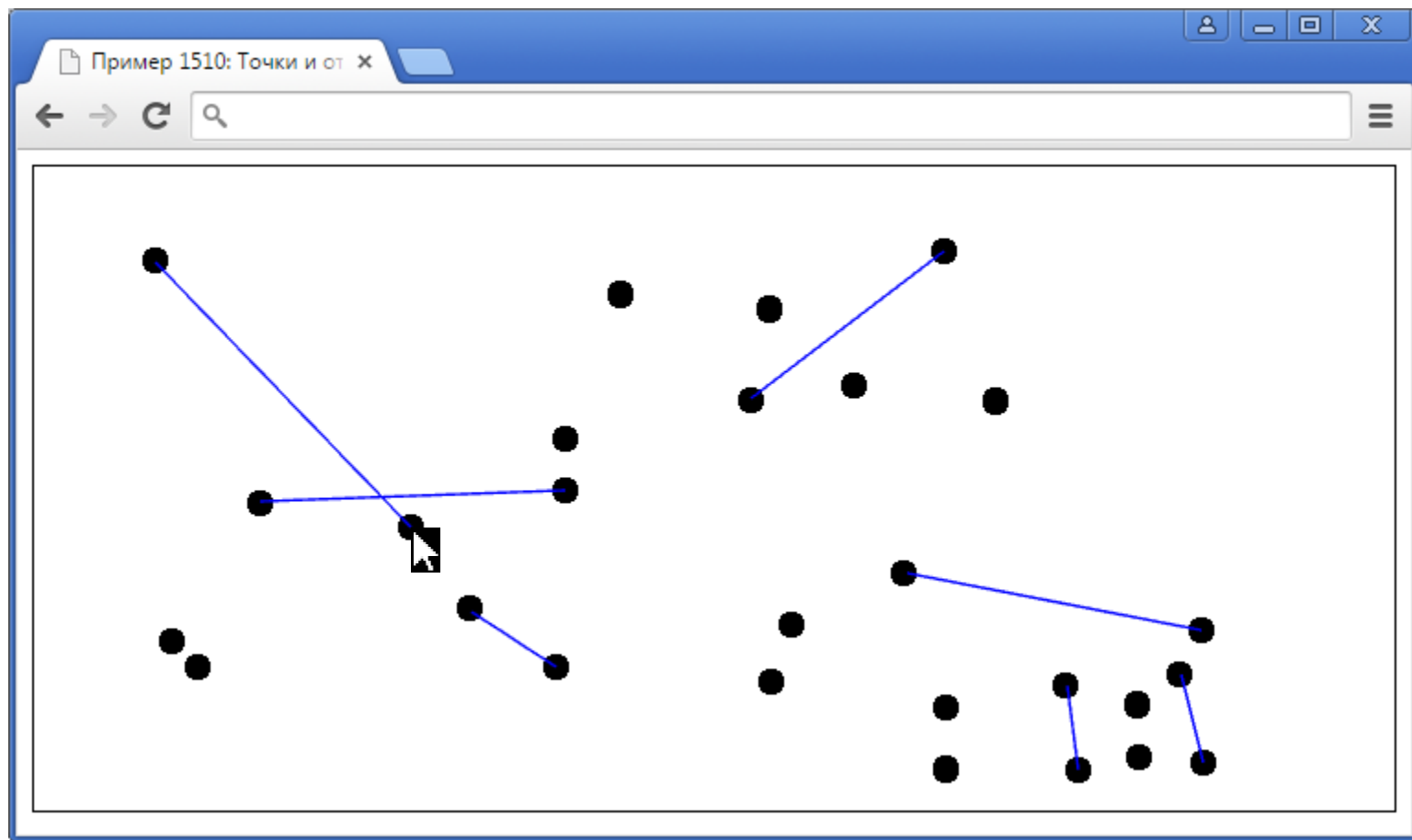
Чертане и на отсечки

- Основен проблем – как да разберем дали да се рисува точка или отсечка
- Решаваме така: при натискане на бутон рисуваме точка
- Това е или самостоятелна точка, или начало на отсечка
- В **pnt** и **obj** са другия край на отсечката и самата отсечка – към момента на натискане на бутон те не съществуват

```
function mouseDown(event)
{
    point(mouseXY(event)).custom(pointStyle);
    pnt = undefined;
    obj = undefined;
}
```


- Отсечката **obj** и крайната ѝ точка **pnt** се генерират при първото движение на мишката с натиснат ляв бутон
- При последващи движения те се актуализират

```
function mouseMove(event)
{
    var pos = mouseXY(event);
    if (event.buttons==1)
    {
        if (!pnt) pnt = point(pos).custom(...);
        if (!obj) obj = segment(pos,pos).custom(...);
        pnt.center = pos;
        obj.to = pos;
    }
}
```

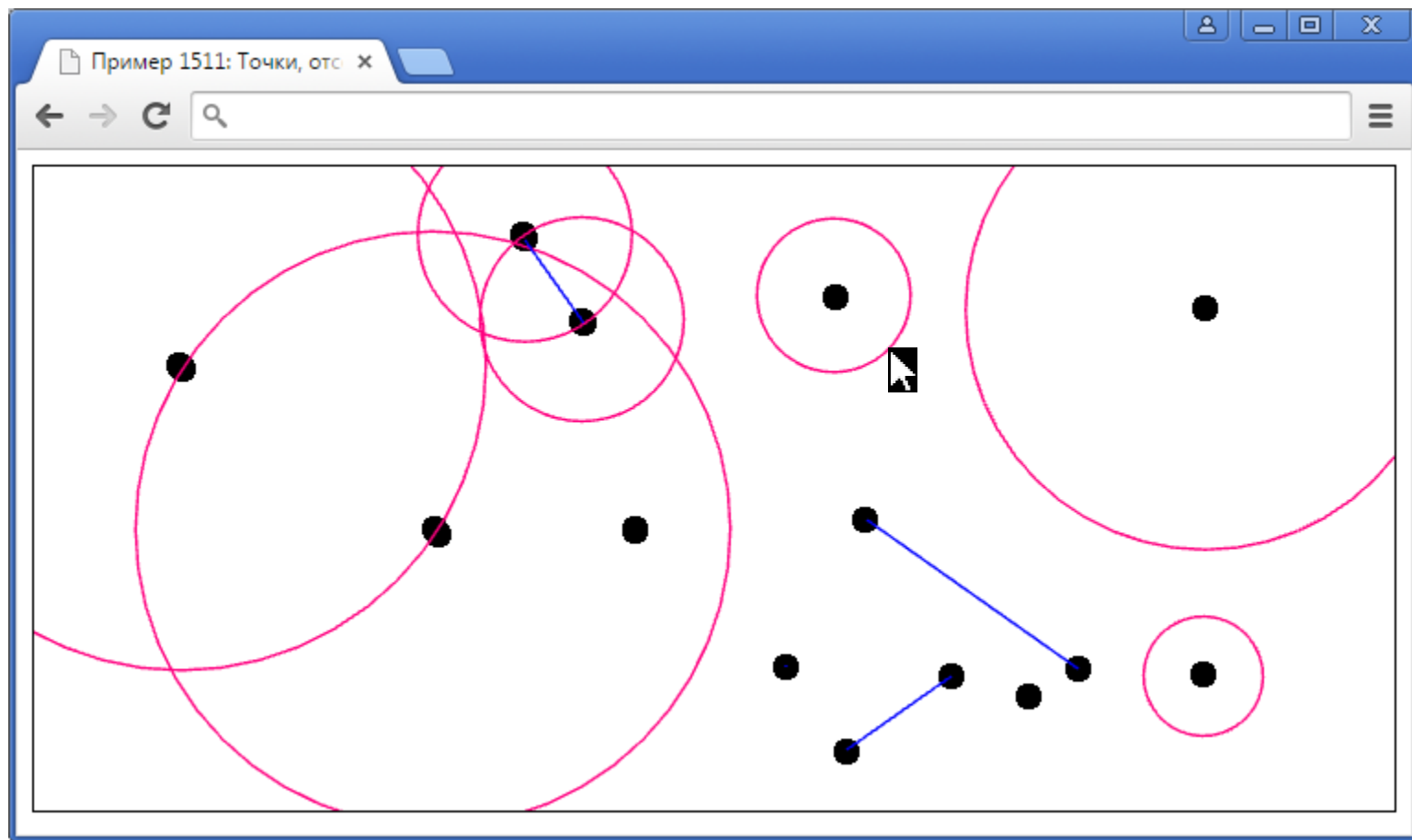


ПРОБА

Чертане и на окръжности

- При натискане на десен бутон рисуваме точка
- Това е или самостоятелна точка, или центъра на окръжност
- В **obj** е точка от окръжността и разстоянието ѝ до центъра определя радиуса (**distance** е наша помощна функция)

```
function mouseMove(event)
{
  ...
  if (event.buttons==2)
  {
    if (!obj) obj = circle(pos,0).custom(...);
    obj.radius = distance(obj.center,pos);
  }
}
```



ПРОБА

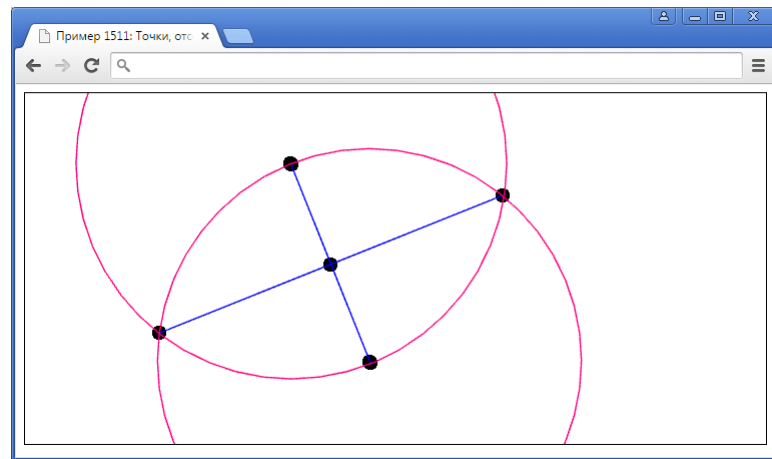
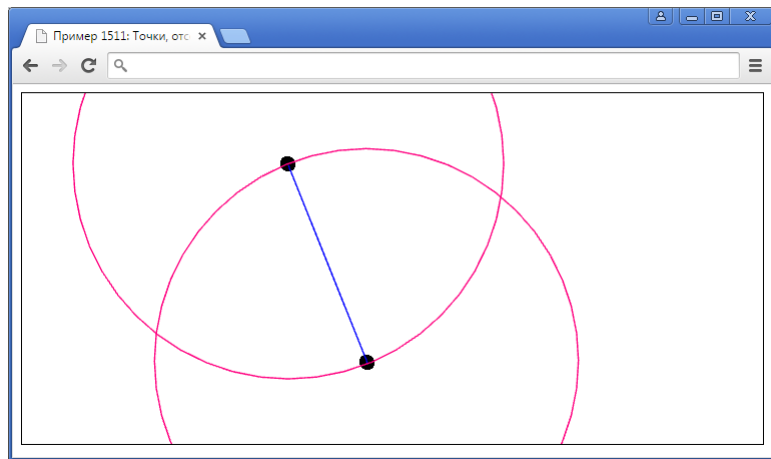
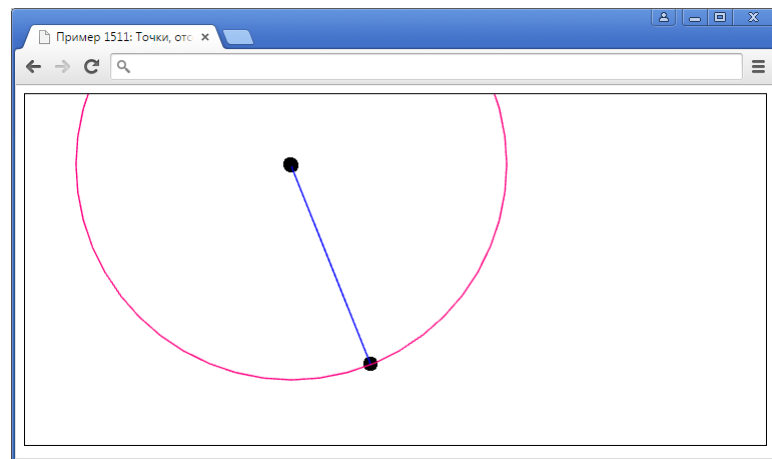
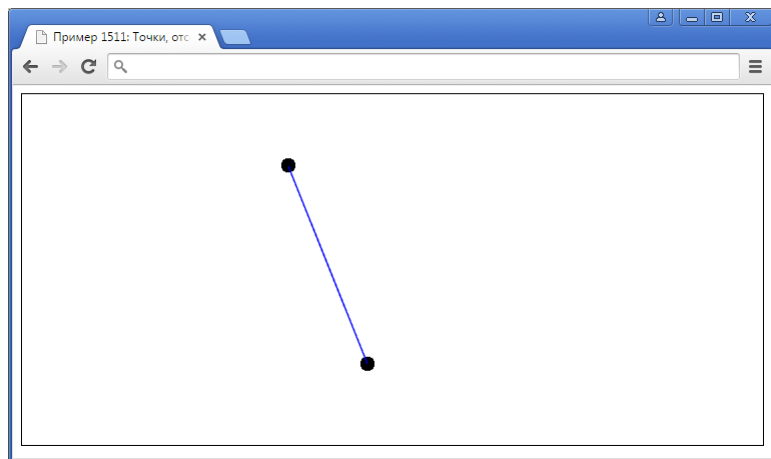
Функционално тестване

Намиране на среда на отсечка



Интерактивна процедура

- Конструираме две случайни точки
- Свързваме ги с отсечка
- Рисуваме окръжности с центрове двете точки и радиуси колкото дължината на отсечката
- Свързваме с отсечка двете пресечни точки на окръжностите
- Сечението на двете отсечки е търсената среда



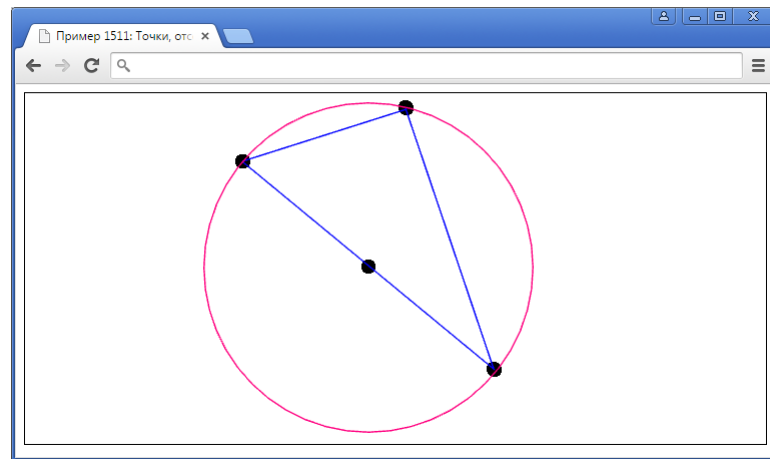
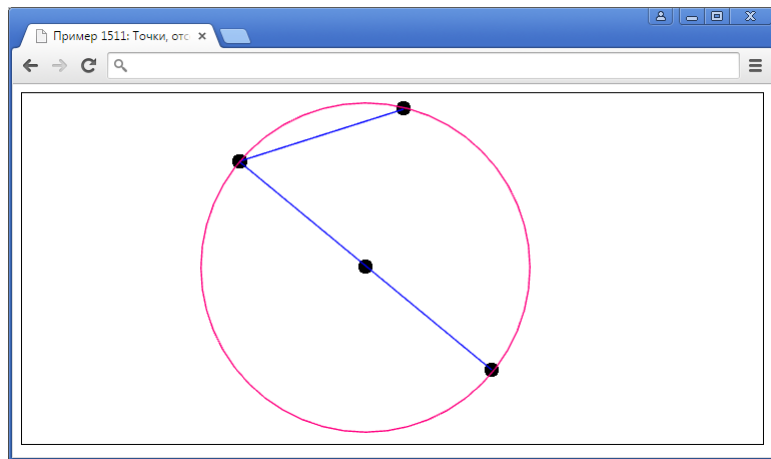
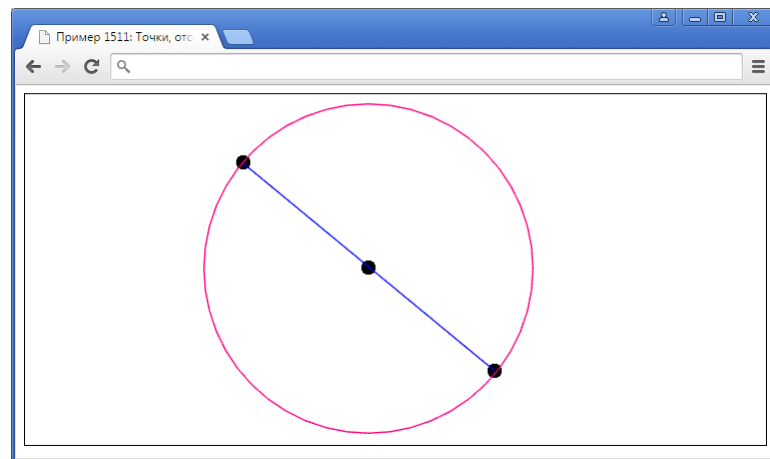
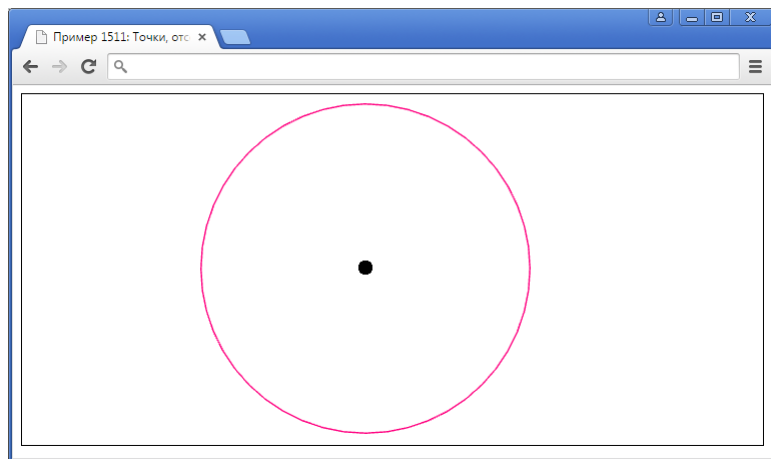
ПРОБА

Рисуване на правоъгълен триъгълник



Интерактивна процедура

- Рисуваме окръжност
- Рисуваме отсечка с краища противоположни точки от окръжността (т.е. минава през центъра)
- Избираме трета точка от окръжността
- Свързваме я с двете точки



ПРОБА

Обобщение

Работа с мишка



Основни събития

- Движение на мишката: `mousemove`, `mouseenter`, `mouseleave`, `mouseover` и `mouseout`
- Работа с бутоните: `mousedown`, `mouseup`, `click` и `dblclick`
- Контекстно меню: `contextmenu`

Основни свойства

- DOM елемент на събитието: `target`
- Координати: `clientX`, `clientY`, `screenX` и `screenY`
- Бутони и клавиши: `buttons`, `altKey`, `ctrlKey` и `shiftKey`

Рисуване и чертане



Рисуване с мишката

- Преобразуване на координатите
- Избор на подходяща проекция, най-често ортографска
- Следене на натиснат бутон и/или клавиш
- Контекстното меню може да се забрани с `preventDefault` в рамките на събитието `contextmenu`

Чертане с мишката

- Обектите, които подлежат на интерактивна промяна, трябва да се създават еднократно, а после само да се модифицират



ИКТ В НОС

Край

Коментари, въпроси