ICT in SES

Interactivity

Lesson №15

Working with the mouse

Events

Events and interactivity

- Events are DOM-related objects
- Originally covered in Lesson №6
- Used to implement interactivity

Events are used to process

- Mouse movements
- Mouse button clicks
- Using the keyboard

Mouse events

Mouse movement

- mousemove – movement
- mouseenter – entering HTML element
- mouseleave – exiting HTML element
- mouseover – movement over HTML element or its subelements
- mouseout – exiting HTML element and its subelements

Mouse buttons

- mousedown – button is pressed
- mouseup – button is released
- click – click
- dblclick – double click
- contextmenu – click with the right (secondary) button

Other events

- Not directly related to graphics:
  - Events for drag and drop of elements and files
  - Events for controlling multimedia

Properties

Event object

- Every event is represented by a JS object
- The object’s properties inform about the event

Properties

- target – DOM element where the event occurred
- clientX, clientY – coordinates in the window
- screenX, screenY – coordinates in the screen
- buttons – what buttons are pressed
- altKey, ctrlKey, shiftKey – press status of Alt, Ctrl, Shift

Coordinates

Local coordinates

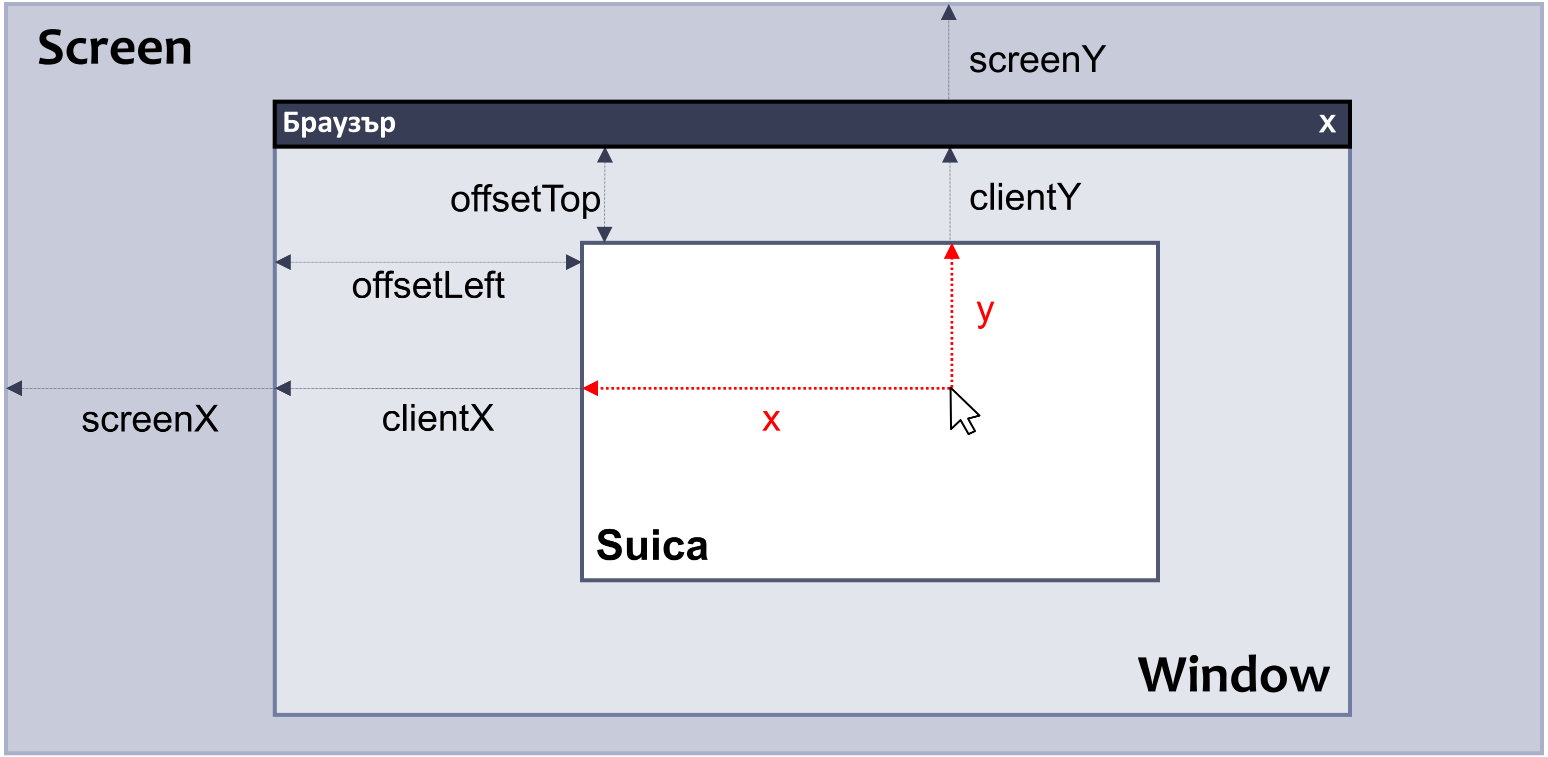
- Graphic element canvas
- Calculating coordinates (x, y) of the mouse cursor relative to the upper left corner of the Suica object

x = clientX – offsetLeft

y = clienyY – offsetTop

x = clientX – offsetLeft

y = clienyY – offsetTop



Example

Blank graphic box

- Showing mouse coordinates when it is moved
- Coordinates are relative to the graphical canvas
- No coordinates when moving outside the canvas

Idea

- Using two events
  - mousemove - movement inside the canvas
  - mouseout - exiting the canvas

Adding event listeners

- Coordinates are shown in info
- Every Suica object contains WebGL object gl, that keeps reference to the DOM element canvas
- Creating two event listeners for p.gl.canvas
- Mouse movement is processed by mouseMove
- Exiting the canvas is processed by mouseOut

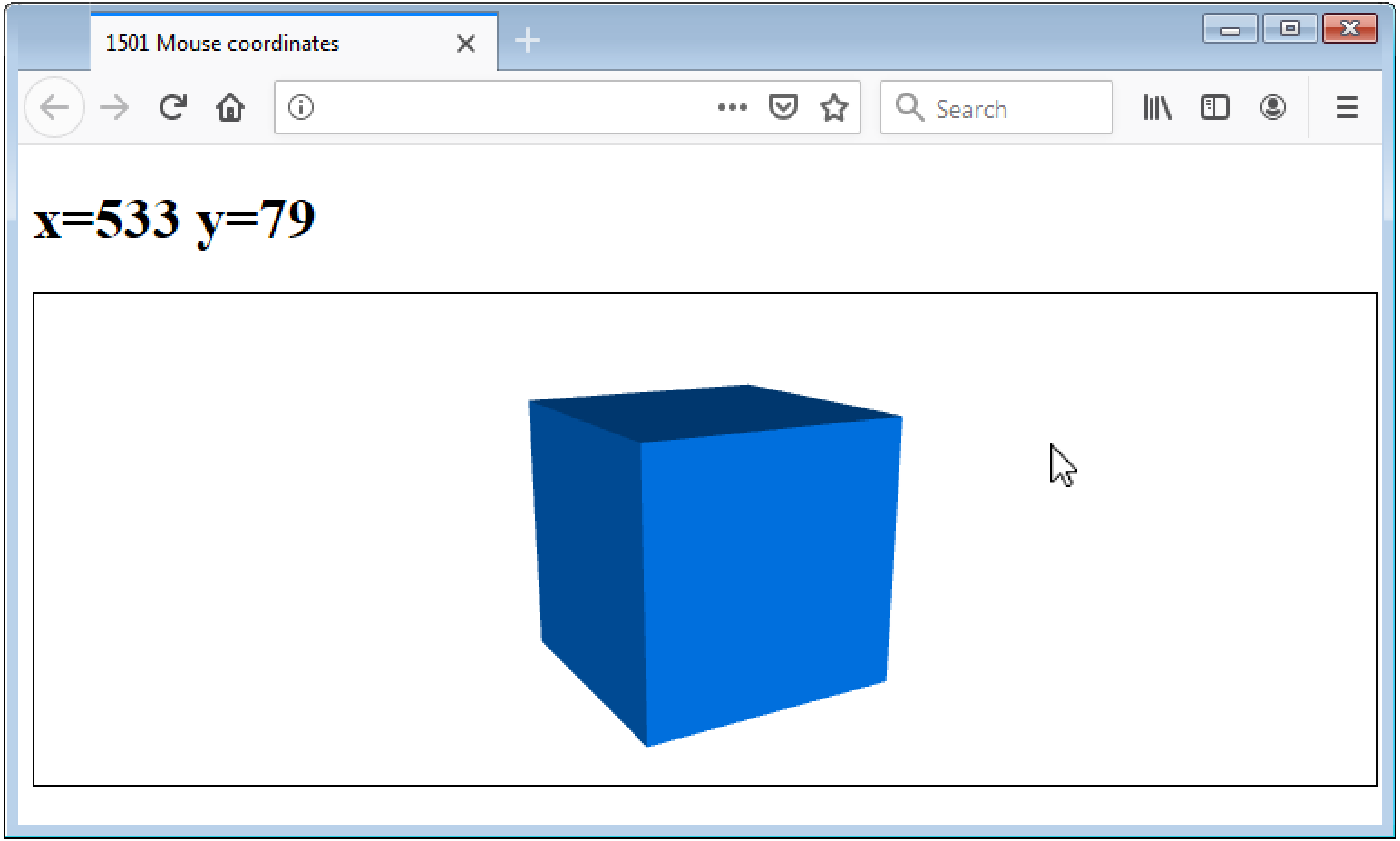
```
info = document.getElementById('info');

p = new Suica();
p.gl.canvas.addEventListener('mouseout',mouseOut,false);
p.gl.canvas.addEventListener('mousemove',mouseMove,false);
```

Processing events

- Parameter event provides mouse coordinates and the offset in 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:... ';
}
```



x=533 y=79

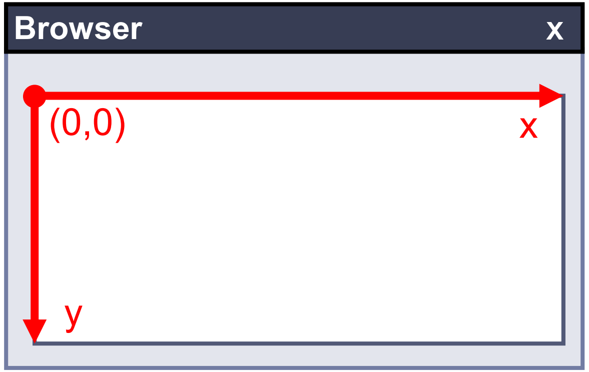
TRY IT

Drawing with the mouse

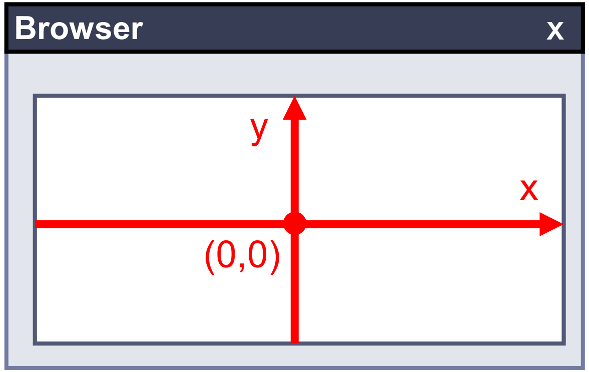
Graphical coordinates

Graphical coordinates

- Coordinates of objects in the canvas
- Differ from mouse coordinates
- Point (0,0) is in the center, X is towards right, Y is upwards



Mouse coordinates

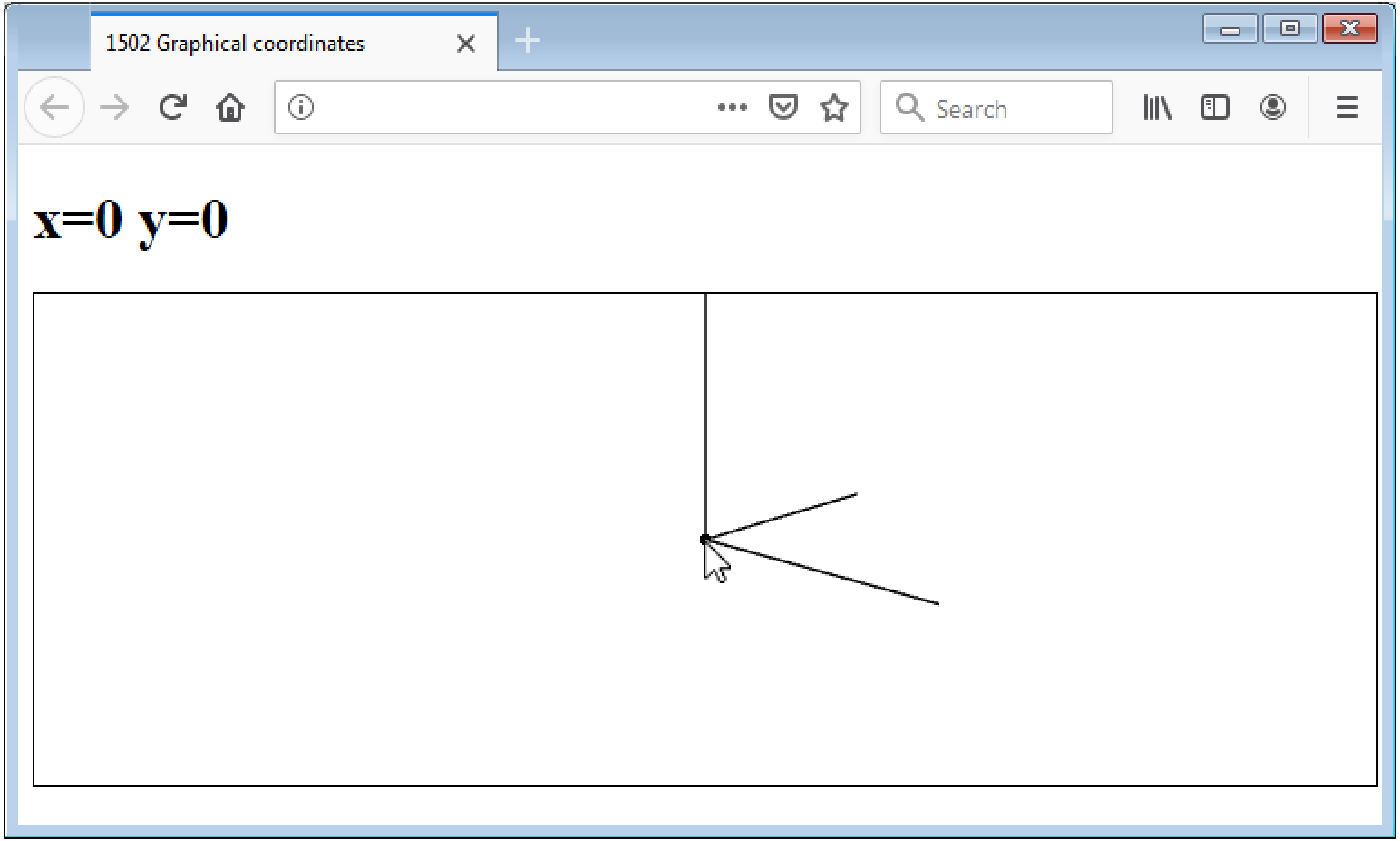


Graphical coordinates

Calculating graphical coordinates

- Center offset and halves of offsetWidth and offsetHeight
- Y coordinates must have opposite sign

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



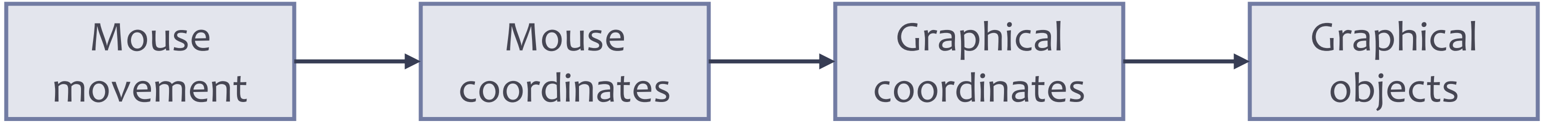
x=0 y=0

TRY IT

Drawing

Drawing of points

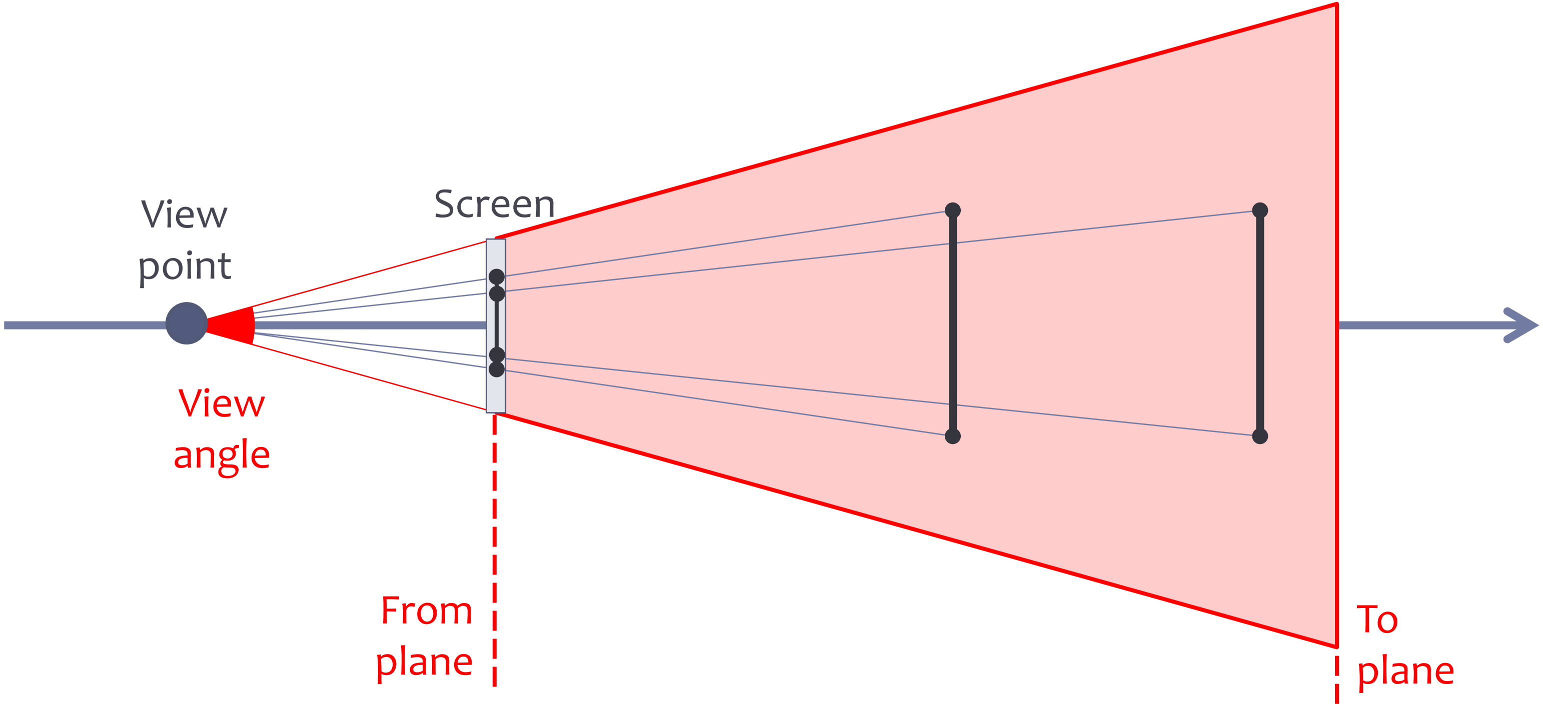
- Mouse movement leaves trace



- Looking at a 2D scene “from top”
- Mapping 1:1 between mouse and graphical coordinates

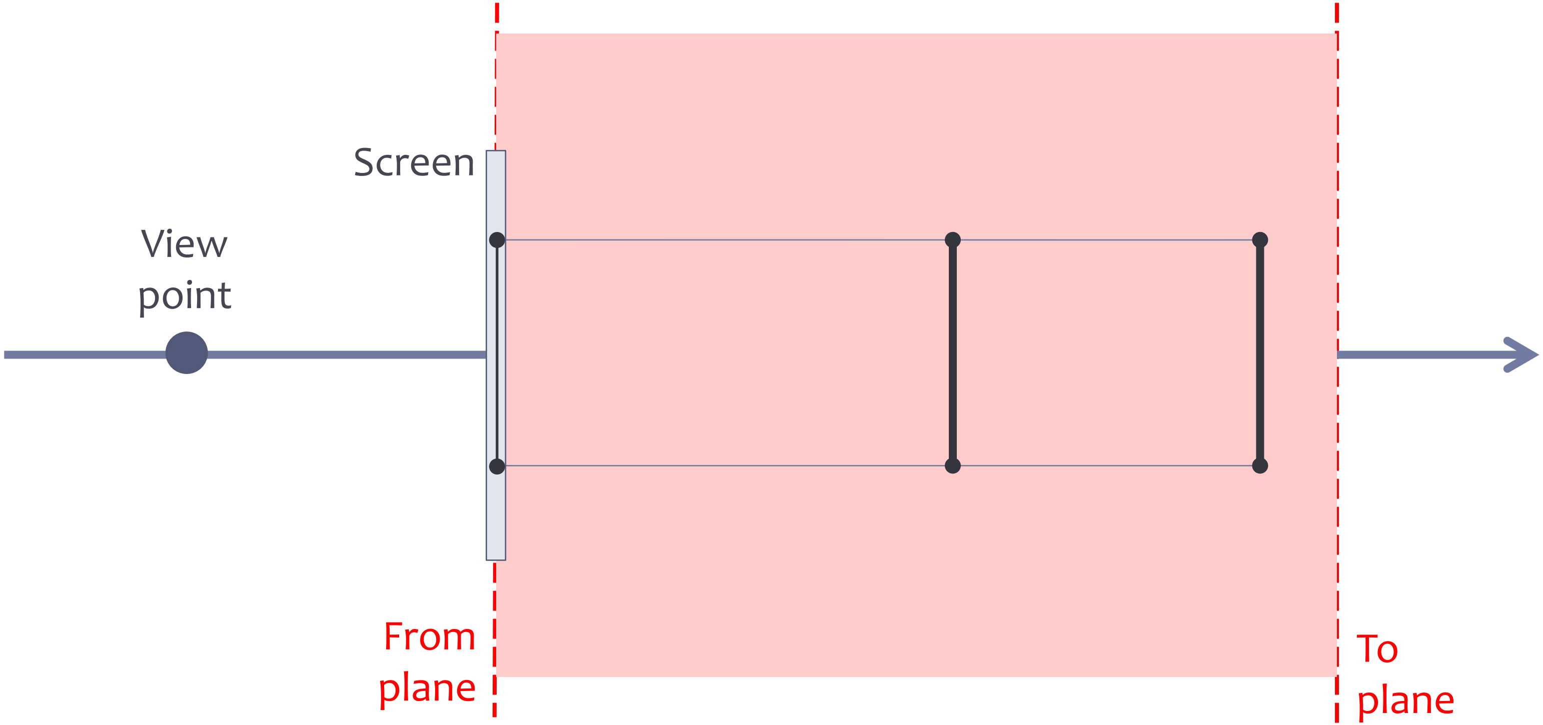
Perspective projection

- Objects farther away appear smaller
- Function perspective (angle, from, to)



Orthographic projection

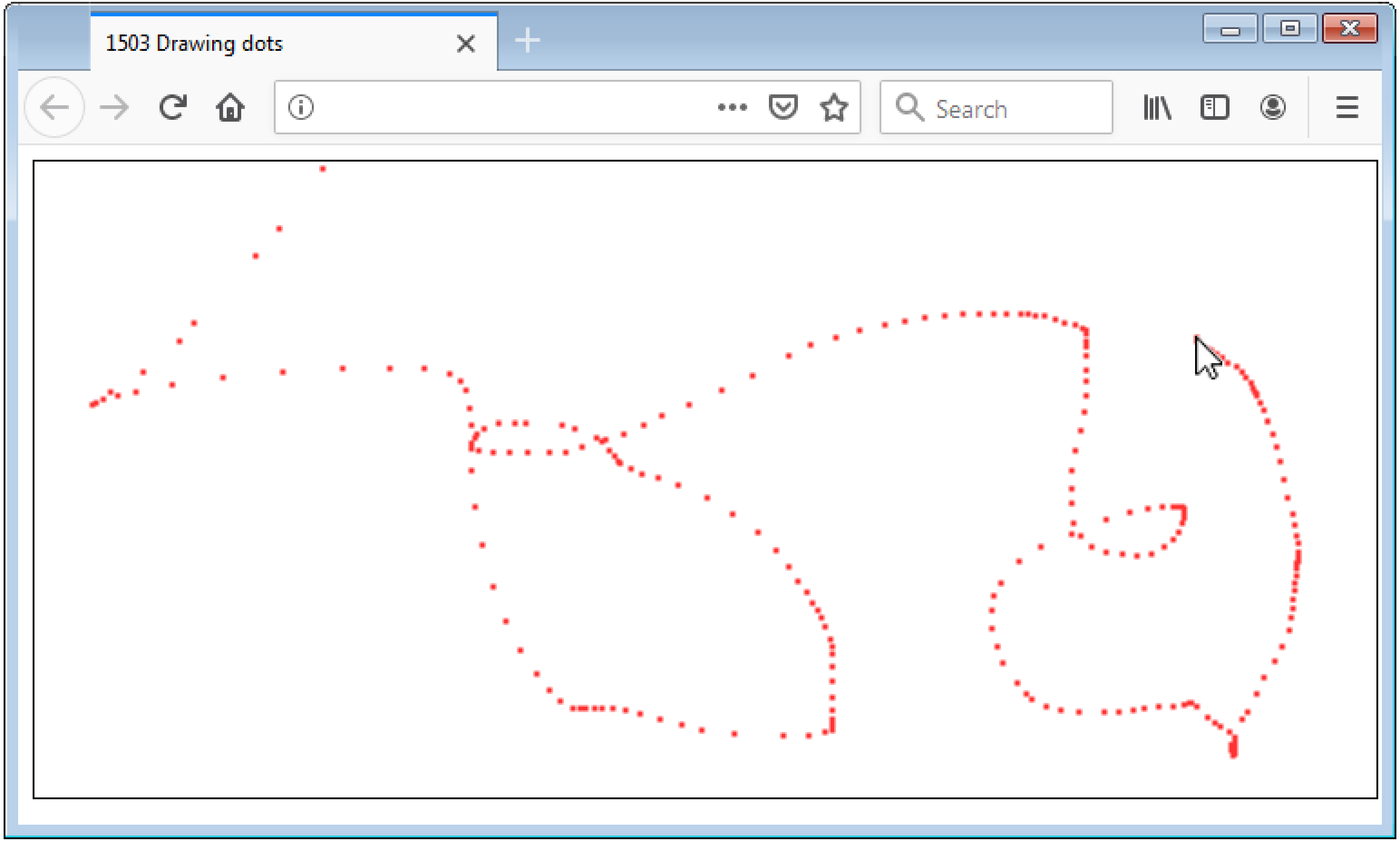
- Object size does not depend on distance
- Function orthographic (from, to)



Implementaiton of drawing

- Using orthographic projection for 1:1 mapping
- Looking "from top" with lookAt: looking from [0,0,1] towards [0,0,0] and [0,1,0] points up
- Finding graphical coordinates and creating a point

```
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]);
}
```



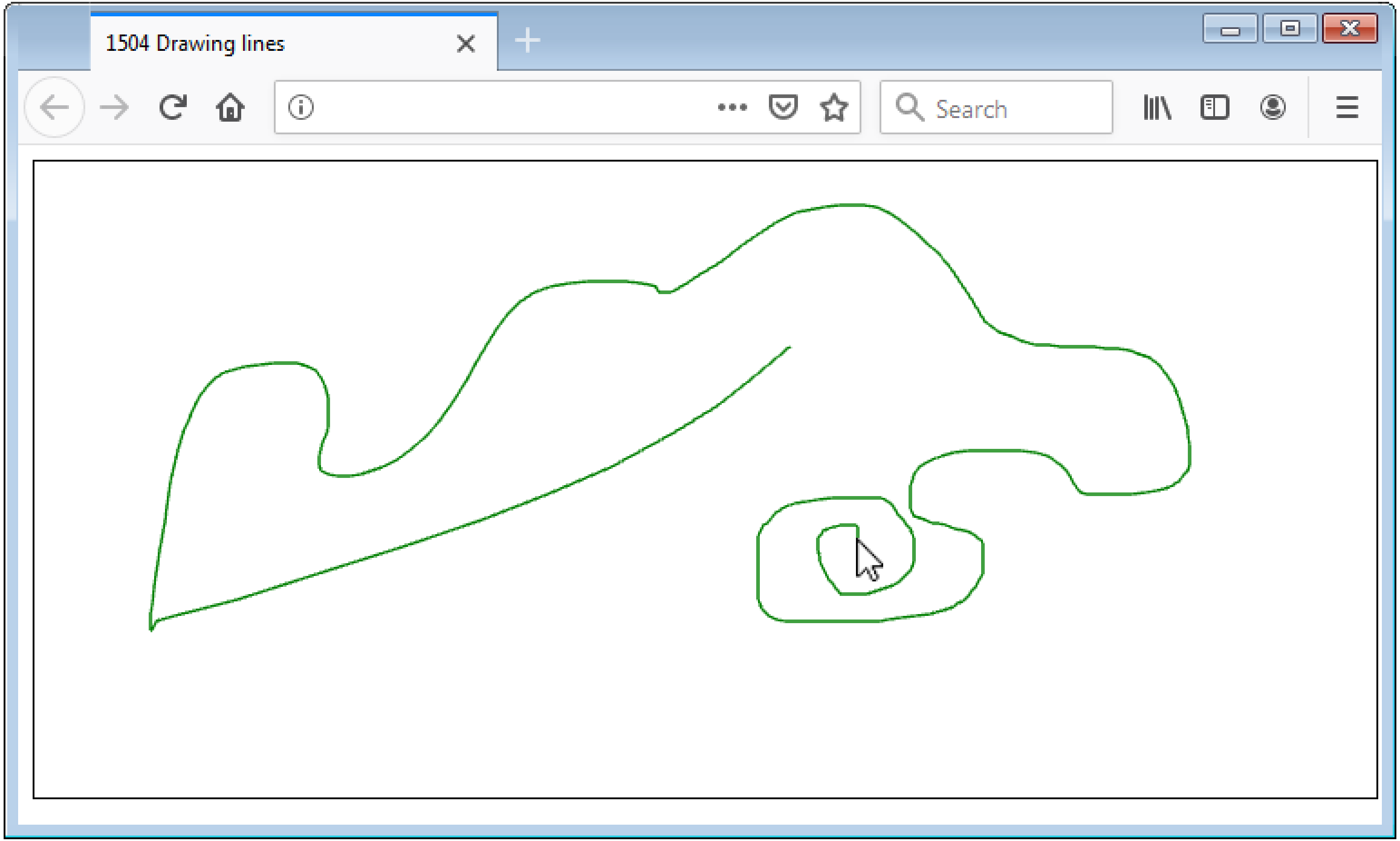
TRY IT

Drawing lines

- Remembering the last point in last
- Drawing a segment if last has a value
- Otherwise draw nothing

```
var last;

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



TRY IT

New functionality

- Drawing starts with pressing the left mouse button
- Drawing ends with releasing the button

Idea

- When pressing – enter a drawing mode
- When releasing – exit the drawing mode
- When moving – draw if in the drawing mode

Implementation

- Listening to three events

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

- In mouseUp exit drawing mode and forget the last remembered position

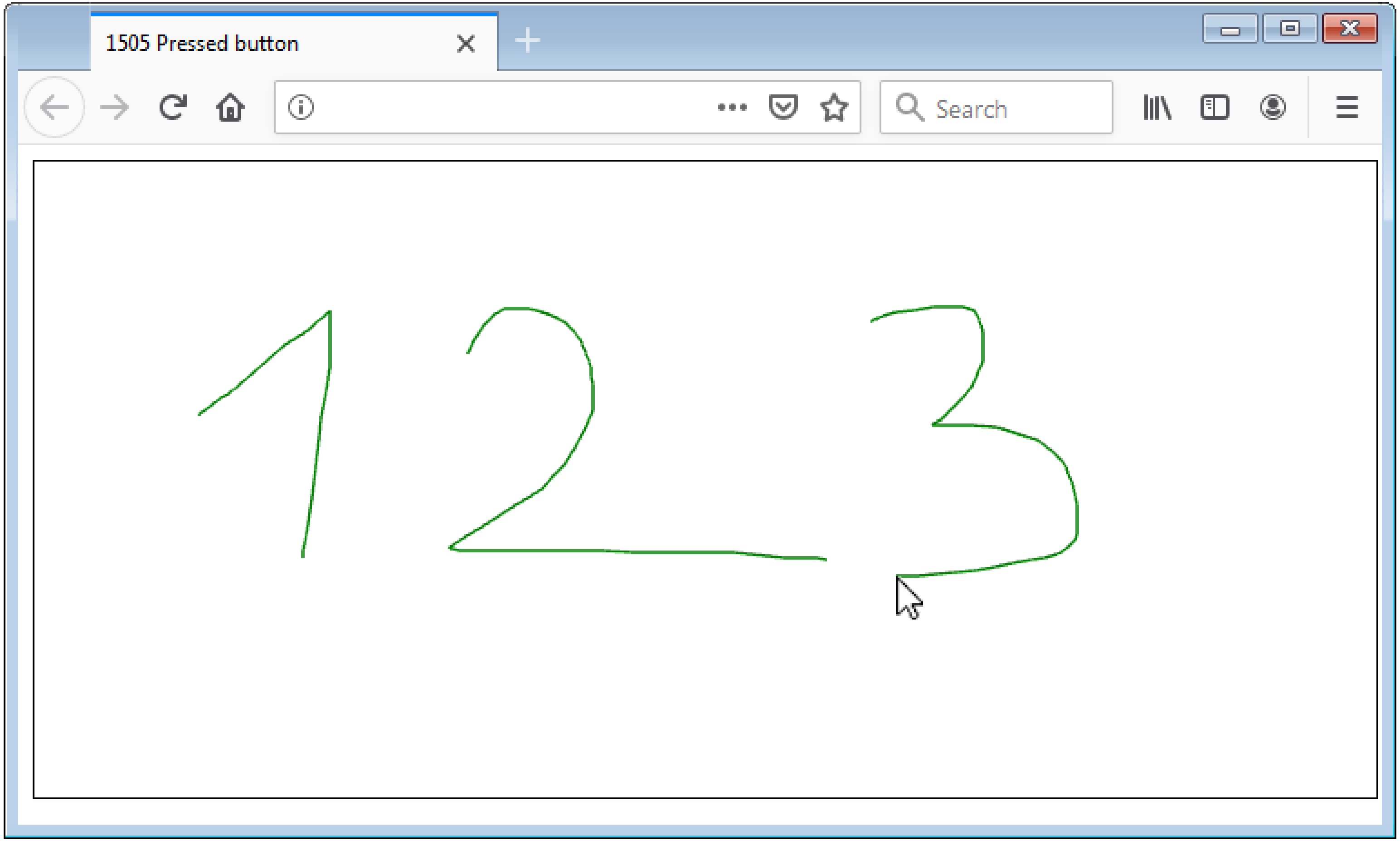
```
draw = false;
last = undefined;
```

- In mouseDown start a drawing mode if the left button is pressed and remember the current position

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

- In mouseMove check for the drawing mode

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

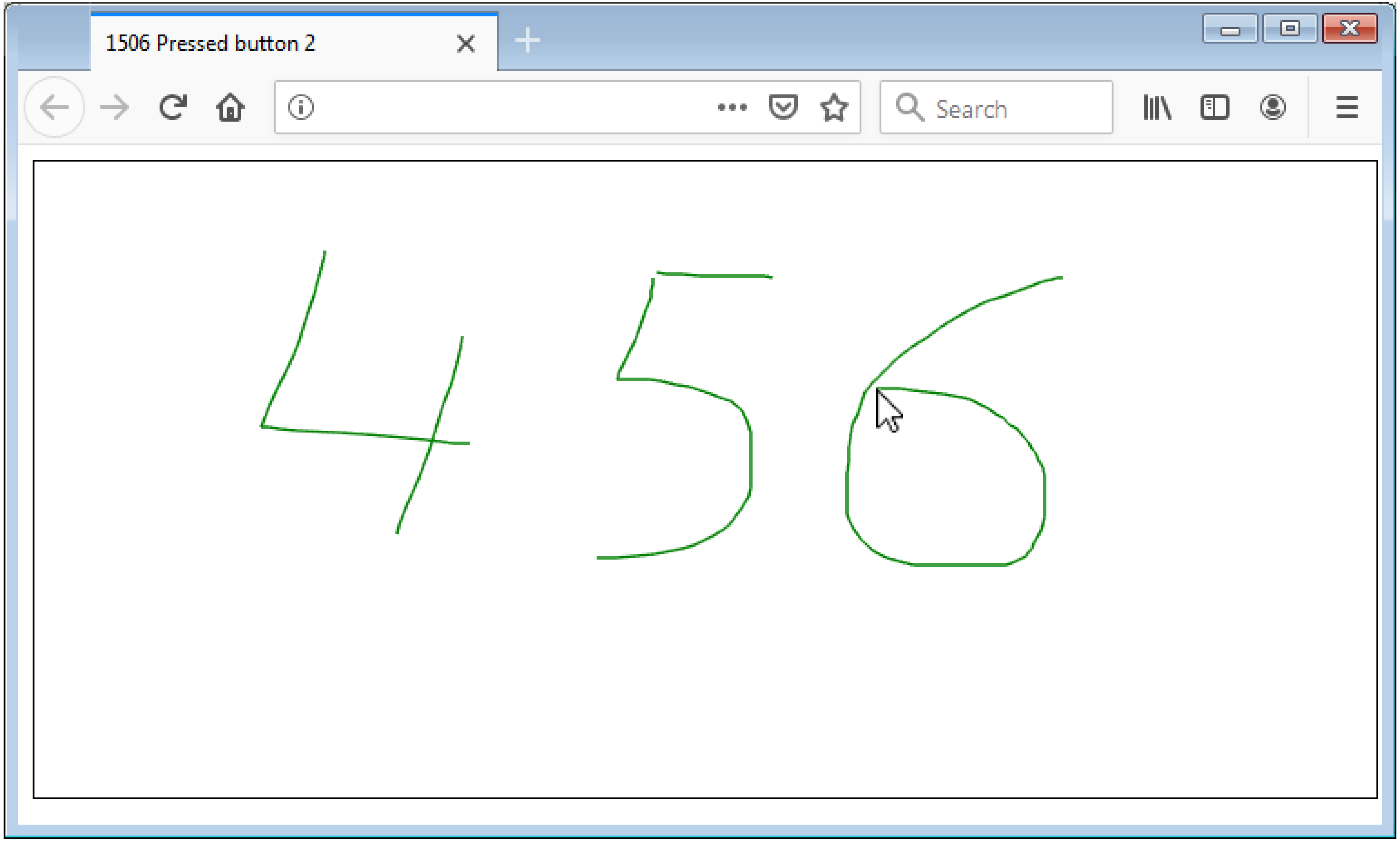


TRY IT

Shorter code? (won’t work in some browsers, like FireFox)

- The last remembered position last is a drawing flag
- If it has a value, then the drawing mode in on
- Ignoring events mousedown and mouseup, considering only mousemove

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



TRY IT

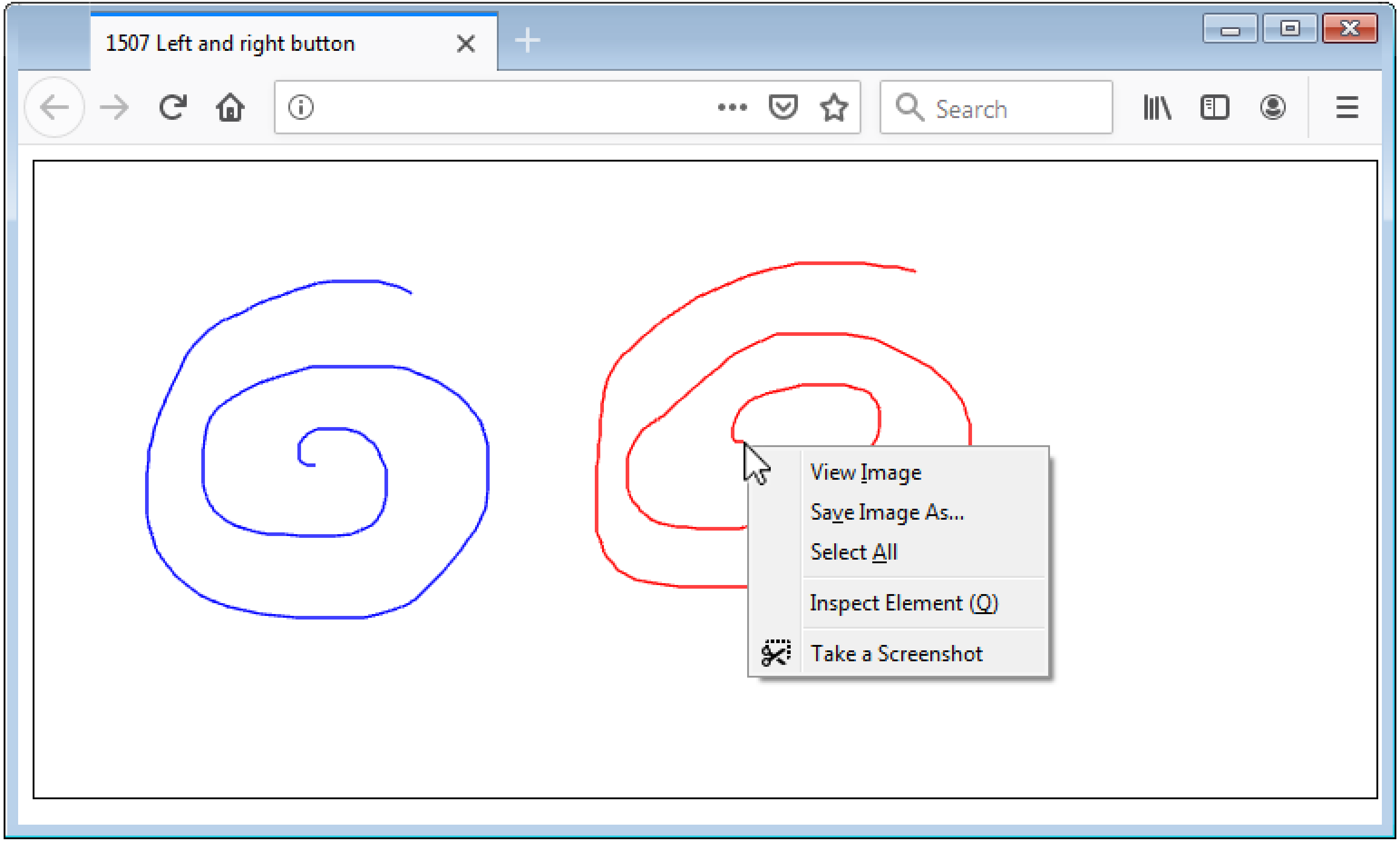
Left and right buttons

- Left button draw blue line, right button – red line

Implementation

- Drawing when a button is pressed
- Storing the colour in style

```
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];
}
```


1507 Left and right button
Search
View Image
Save Image As...
Select All
Inspect Element (Q)
Take a Screenshot

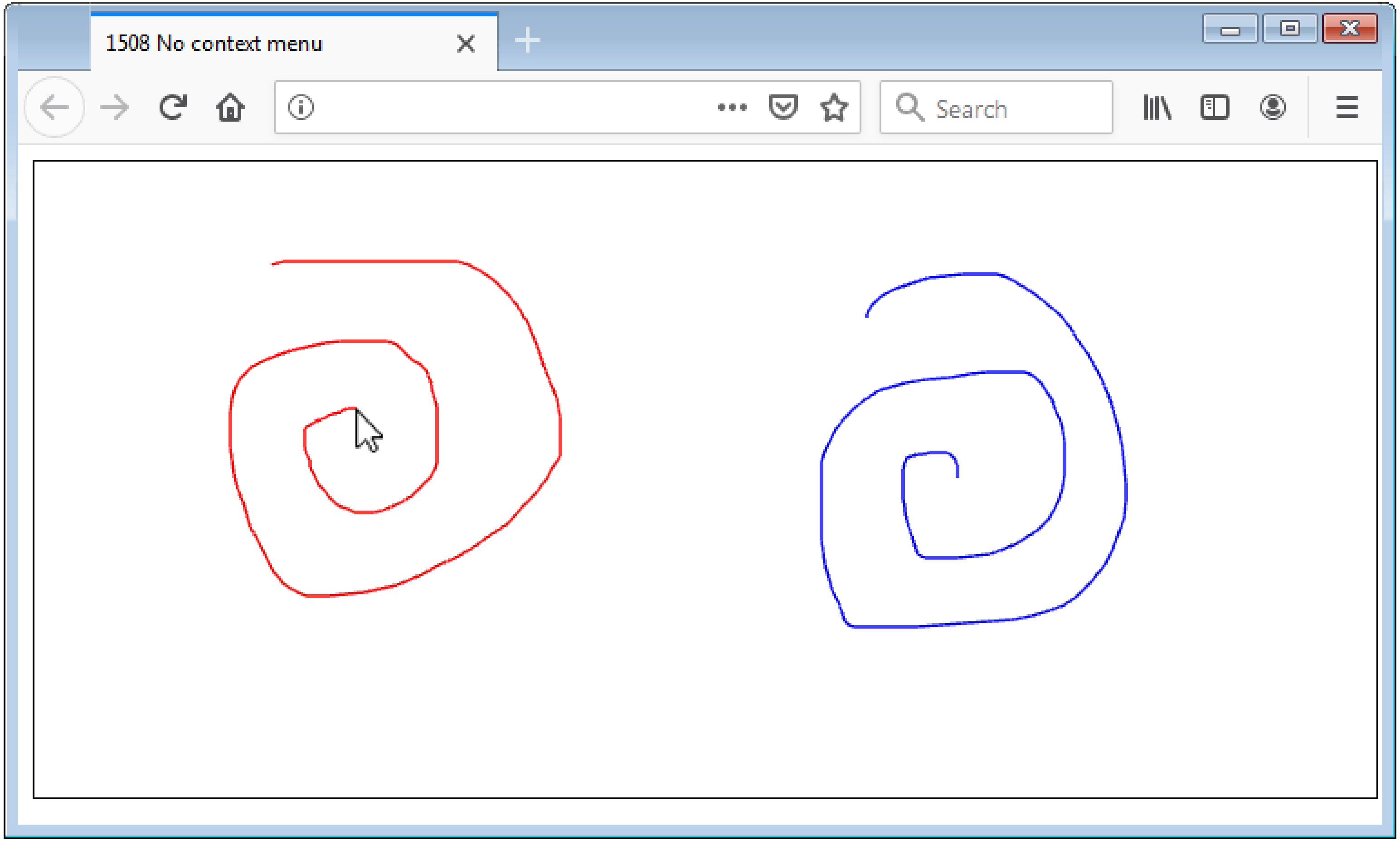
TRY IT

Context menu

- Clicking the right button pop ups the context menu
- Removing the context menu by listening to event contextmenu
- Using the method preventDefault to prevent the default action, which is to show the context menu

```
...addEventListener('contextmenu',contextMenu,false);

function contextMenu(event)
{
   event.preventDefault();
}
```



TRY IT

Sketching with the mouse

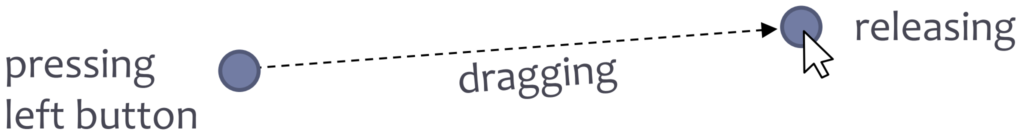
Sketching

Goal

- Making sketches with the mouse
- Drawing points, segments and circle

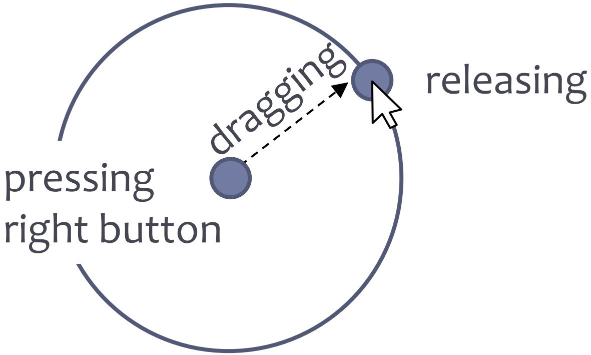
Drawing segments

- Pressing the left mouse button for beginning
- Dragging for construction and releasing for finalization



Drawing circles

- Press right button for the center
- Drag and release when the radius is correct



Drawing points

- Clicking with the left button

Help functions

- Function mouseXY calculates graphical coordinates, corresponding to the mouse coordinates in an event

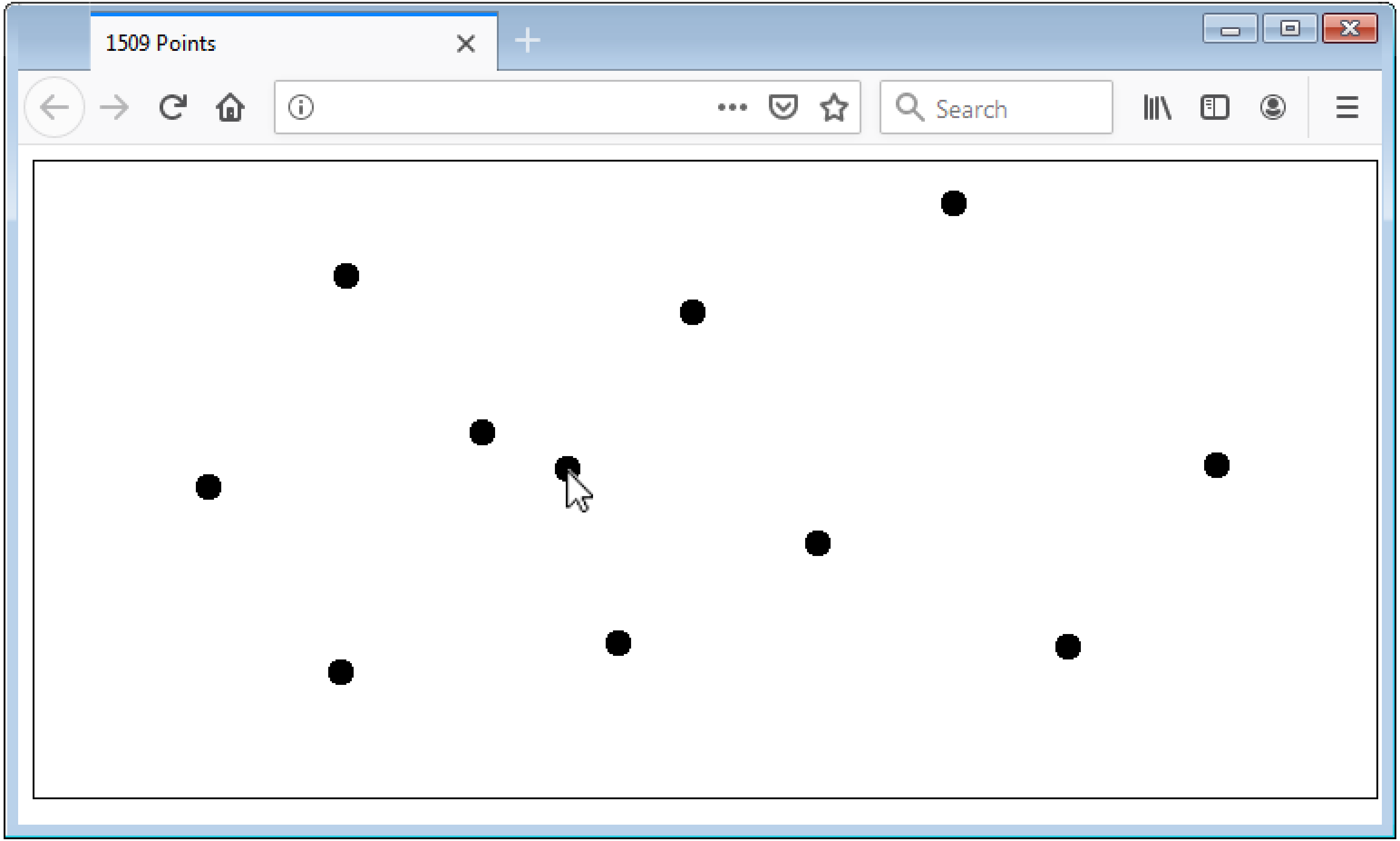
```
function mouseXY(event)
{
  var x = event.clientX
          - event.target.offsetLeft
          - event.target.offsetWidth/2;
  var y = -(event.clientY
            - event.target.offsetTop
            - event.target.offsetHeight/2);
  return [x,y,0];
}
```

Drawing points

- Capturing the pressing of a mouse button (left or right)
- Generating a point at these coordinates
- The style of all points is stored in pointStyle

```
pointStyle = {color:[0,0,0], pointSize:14.5};

function mouseDown(event)
{
   point(mouseXY(event)).custom(pointStyle);
}
```



TRY IT

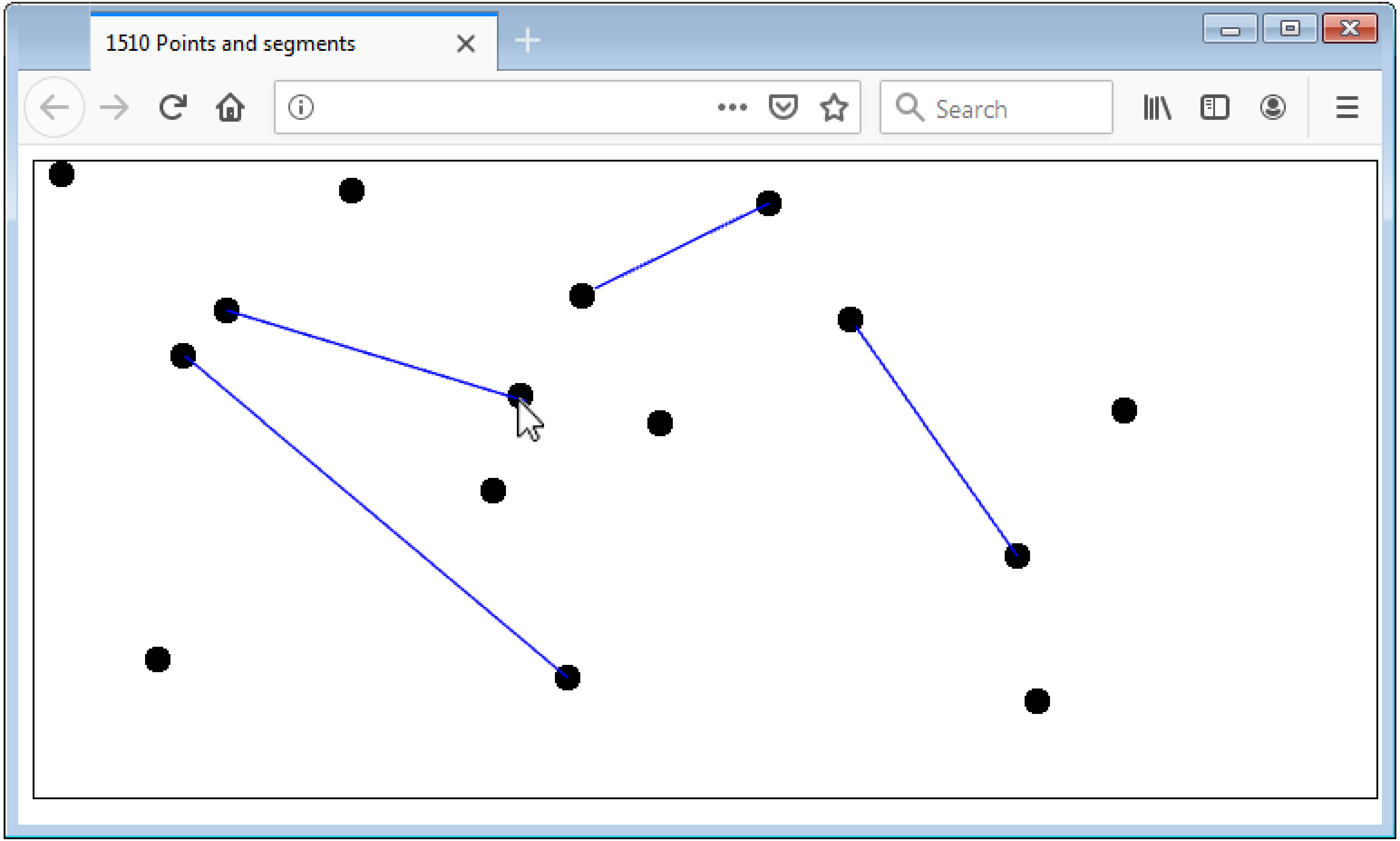
Drawing segments

- How to distinguish drawing points from drawing segments?
- Solution: drawing a point when a button is pressed
- This is either individual point or the beginning of a segment
- In pnt and obj are the other end of the segment and the segment itself – they still do not exist at the time of pressing the buttonсъществуват

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

- Segment obj and its end point pnt are generated at the first mouse movement with pressed left button
- Following movements update them

```
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;
   }
}
```

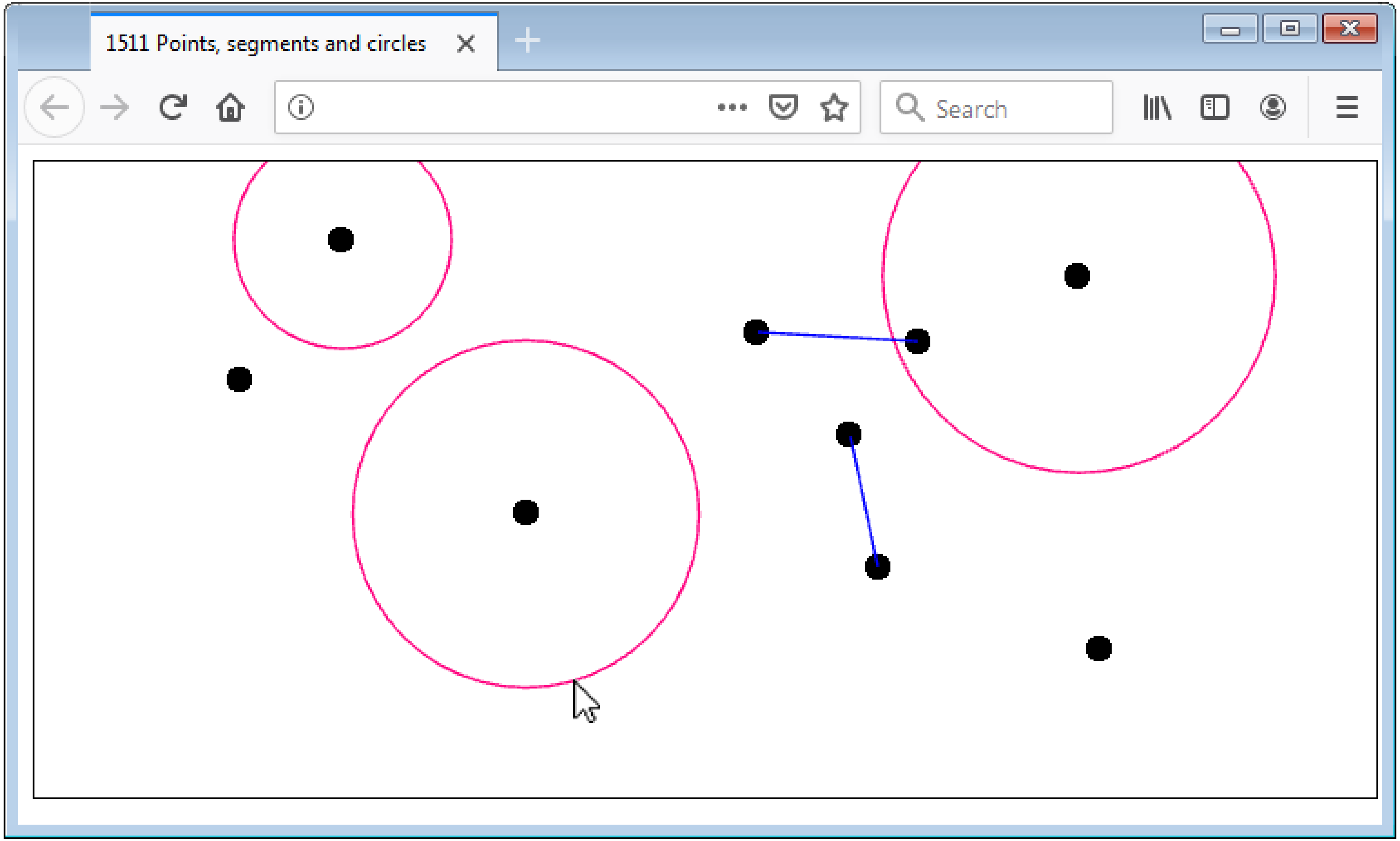


TRY IT

Drawing circles

- Drawing a point when pressing the right mouse button
- This is individual point or the center of a circle
- A point on the circle is in obj and the distance to the center defines the radius (distance is a help function)

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



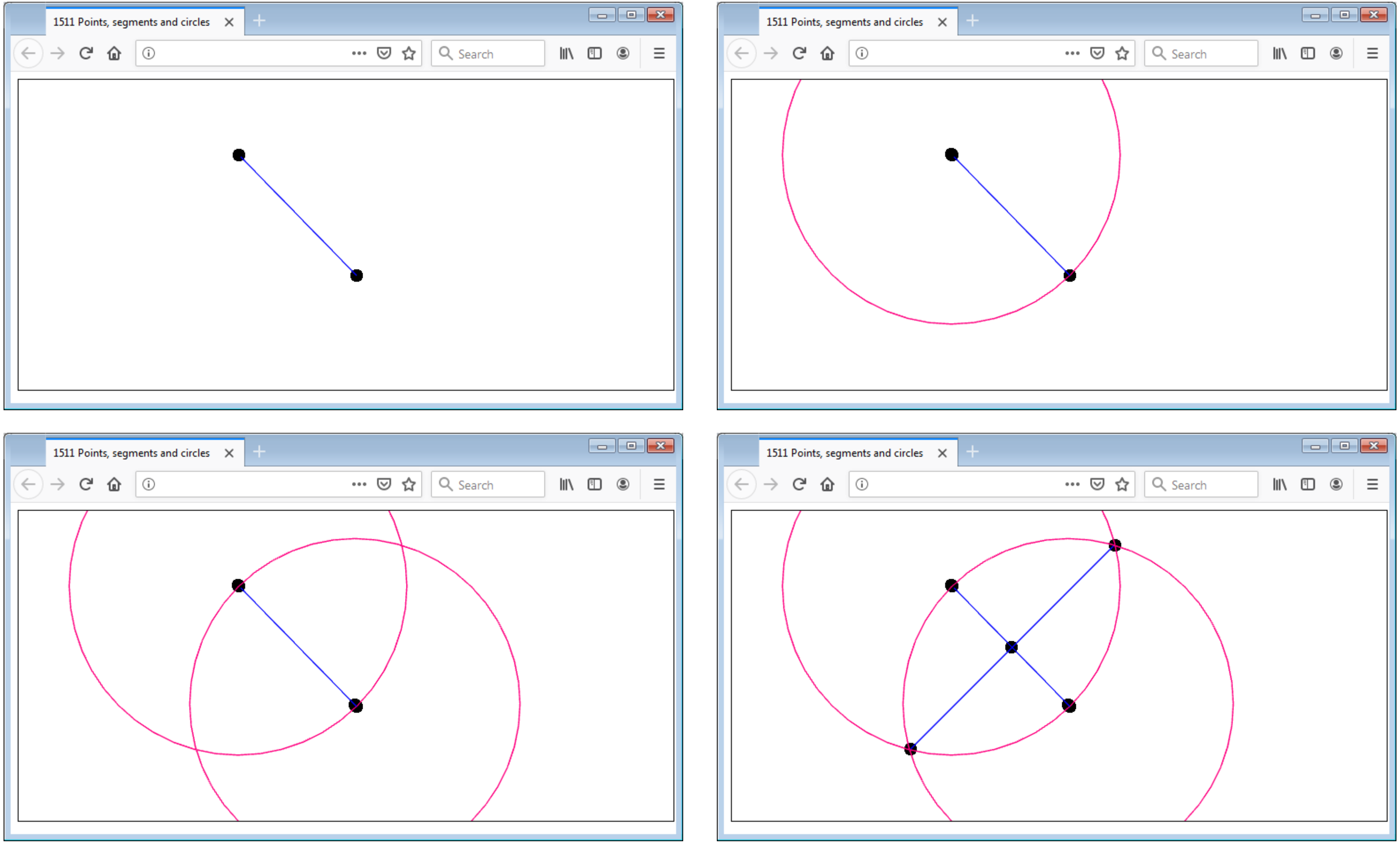
TRY IT

Functional testing

Finding the midpoint of a segment

Interactive procedure

- Construct two random points
- Connect them with a segment
- Draw two circle with centers these points and radii as the segment length
- Connect the intersecting point of the circles with a segment
- This segment intersect the first segment in the mid point

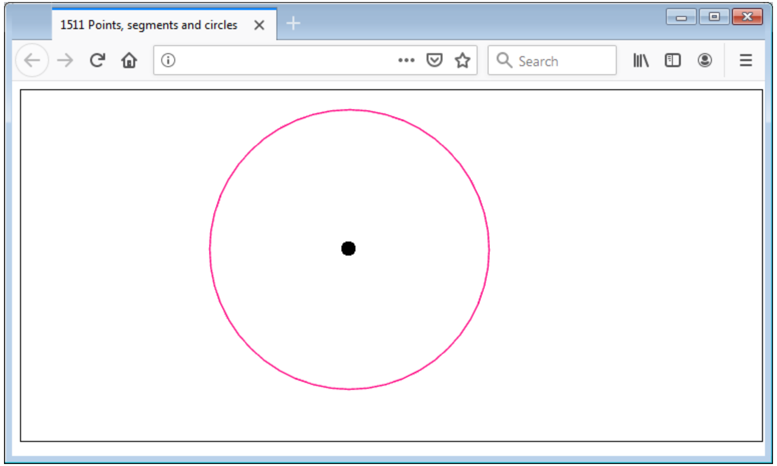

1511 Points, segments and circles
Search
1511 Points, segments and circles
Search
1511 Points, segments and circles
Search
1511 Points, segments and circles
Search

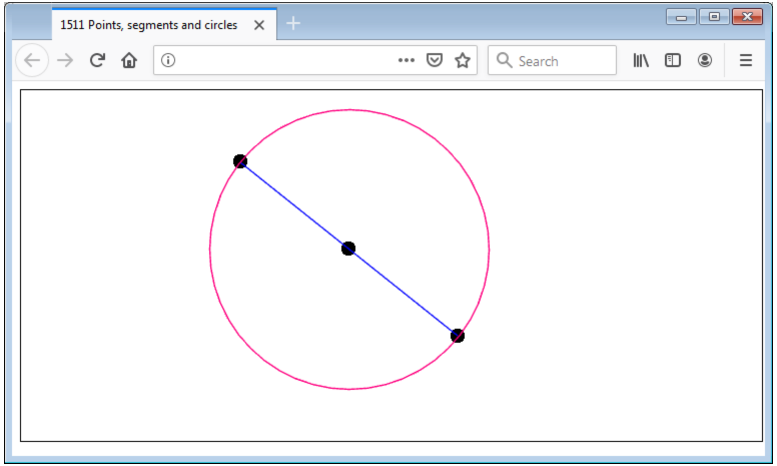
TRY IT

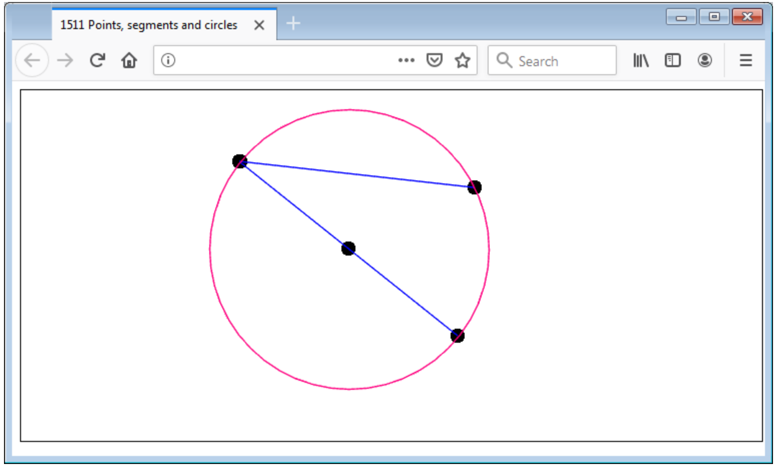
Drawing a right triangle

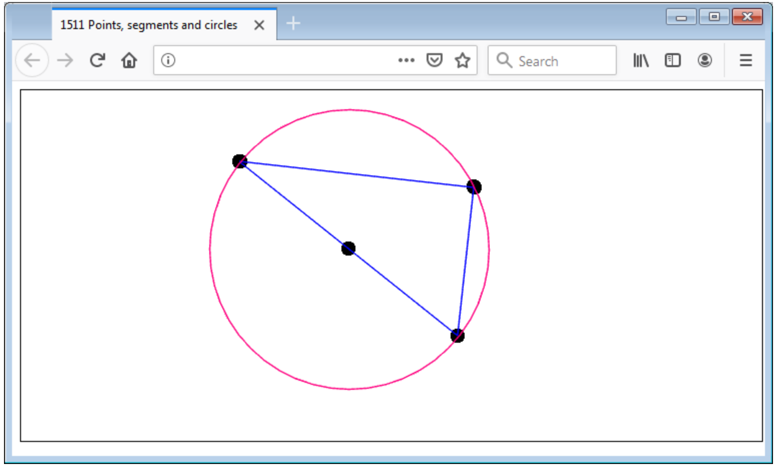
Interactive procedure

- Draw a circle
- Draw a diameter – a segment passing through the center
- Pick a point on the circle
- Connect it with the segment


1511 Points, segments and circles
Search


1511 Points, segments and circles
Search


1511 Points, segments and circles
Search


1511 Points, segments and circles
Search

TRY IT

Summary

Working with the mouse

Useful events

- Mouse movement: mousemove, mouseenter, mouseleave, mouseover and mouseout
- Mouse buttons: mousedown, mouseup, click and dblclick
- Context menu: contextmenu

Properties

- DOM element of the event: target
- Coordinates: clientX, clientY, screenX and screenY
- Buttons and keys: buttons, altKey, ctrlKey and shiftKey

Drawing and sketching

Drawing with the mouse

- Transformation of coordinates
- Selected of proper projection (usually orthographic)
- Listening to button/key pressing
- Disabling context menu with preventDefault in contextmenu

Sketching with the mouse

- Object are created once, then are only modified

ICT in SES

The end

Comments, question