


# HTML canvas

07.04.2022

Marek Nagy

# HTML canvas

- kresliaca plocha
  - rastrová, pixle
- písanie textu
- vykresľovanie obrázkov
- animovanie
- interakcia



Zjednodušene si to možno predstaviť ako <img> obrázok, ale s tým, že možno doň kresliť.

```
canvas = document.createElement('canvas');  
document.body.appendChild(canvas);
```

# canvas

- atribúty **width**, **height** nastaví rozmer
  - default je 300, 150
- pozor pri nastavení CSS **style width, height** !
  - canvas sa škáluje (podobne ako keby to bol img)
- vykresľovanie podľa nastavení v kontexte
- bod **(0,0)** ľavý horný roh

# Context

- atribúty kreslenia
  - farba, hrúbka čiary, ...
- nie je potrebné zakaždým nastavovať
- využije sa pri kresliacich príkazoch

```
canvas = document.createElement('canvas');  
document.body.appendChild(canvas);  
  
ctx = canvas.getContext('2d');
```

Je možné nastaviť aj „webgl“, „webgl2“. Využíva štandard OpenGL ES, ktorý však musí prehliadač a hardvér podporovať. My ďalej uvažujeme iba jednoduché „2d“ operácie.

Kontext bude vždy ten istý pre daný canvas a typ.



# Atribúty čiary a výplne

## fillStyle

- farba, vzor alebo gradient **výplne**

## strokeStyle

- farba, vzor alebo gradient **čiar**

## lineWidth

- šírka čiar

## lineJoin, lineCap, miterLimit

- spojenie čiar

```
ctx.strokeStyle = "#FF0000";  
ctx.fillStyle = "red";
```

# Gradient

- lineárny

- dané súradnice x,y dvoch bodov
- farby sa pridávajú na úsečku v pozícii  $0 \leftrightarrow 1$



```
let grd = ctx.createLinearGradient (0,0, 170,0);  
grd.addColorStop (0, "black");  
grd.addColorStop (1, "white");
```

- radiálny (kruhový)

- daný stred x,y a polomer dvoch kružníc
- farby sa pridávajú na „spojnicu“ kružníc ( $0 \leftrightarrow 1$ )



```
let grd = ctx.createRadialGradient (75,50,5, 90,60,100);  
grd.addColorStop (0, "red");  
grd.addColorStop (1, "white");
```

# Pattern (vzor)

- tapetovanie plochy obrázkom
  - opakovane:  
**repeat, repeat-x, repeat-y**
  - iba raz:  
**no-repeat**



```
let img = document.createElement("img");  
img.src = "ziarovka.png";  
  
let pattern = ctx.createPattern (img, "repeat");
```





# Kreslenie obdĺžnikov

**fillRect** (x,y, width,height)

- vykreslí vyplnený obdĺžnik
- x,y je horný ľavý roh

**strokeRect** (x,y, width,height)

- vykreslí obrys obdĺžnika

**clearRect** (x,y, width,height)

- vymaže canvas v danej obdĺžnikovej oblasti

```
ctx.fillRect (20,20, 150,100);  
ctx.strokeRect (20,20, 150,100);
```



# Ťah

- postupnosť bodov – súradníc (x,y)
- v kontexte existuje iba jeden → treba ho rušiť
- ťah treba potom do canvasu aj vykresliť
- do ťahu sa vždy pridávajú ďalšie a ďalšie body, t.j. nezruší sa po vykreslení !

```
ctx = canvas.getContext('2d');  
// Vytvorenie ťahu  
ctx.beginPath (); // Začne prázdny ťah  
ctx.moveTo (0,0);  
ctx.lineTo (100,40);  
  
// Vukreslenie do canvasu  
ctx.stroke ();
```

# Ťah - vytvorenie a vykreslenie

**beginPath ()**

- začiatok vytvárania ťahu (pôvodný sa zruší)

**closePath ()**

- spojí posledný a prvý bod ťahu

**stroke ()**

- vykreslí aktuálny ťah

**fill ()**

- vykreslí iba výplň ťahu (ak nie je uzavretý, uzavrie ho)

# Ťah - čiara

**moveTo** (x, y)

- presun do bodu

**lineTo** (x, y)

- čiara do bodu

```
ctx = canvas.getContext('2d');
```

```
ctx.beginPath ();
```

```
ctx.moveTo (0,0);
```

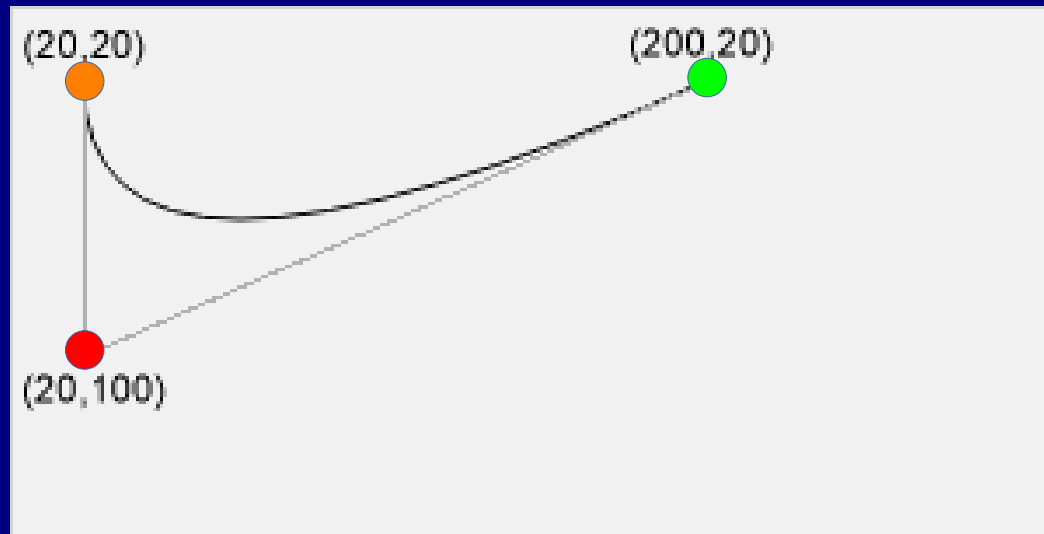
```
ctx.lineTo (100, 40);
```

```
ctx.stroke ();
```

# Ťah - kvadratická krivka

`quadraticCurveTo (cpX,cpY, x,y)`

– jeden kontrolný bod

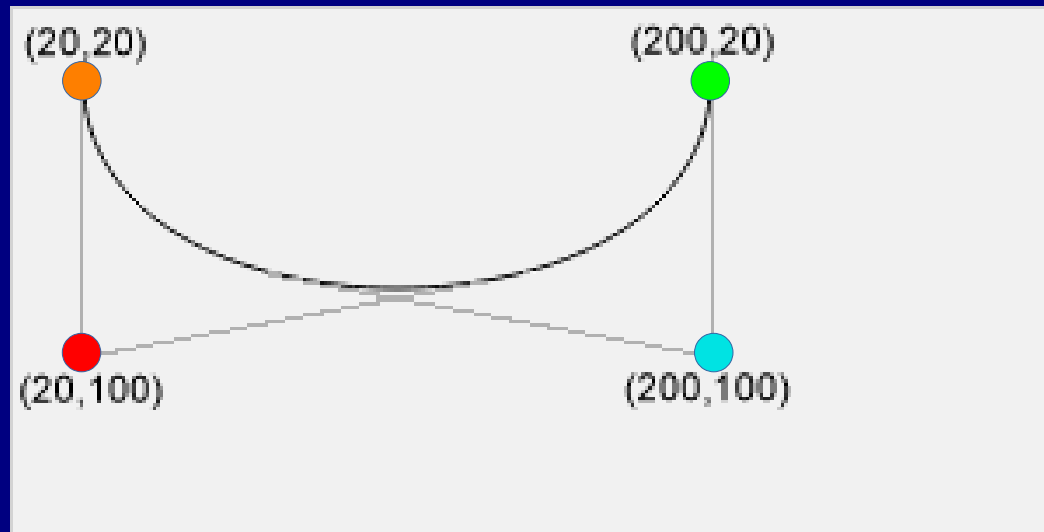


```
ctx.beginPath ();  
ctx.moveTo (20,20); // Začiatkový bod  
  
ctx.quadraticCurveTo (20,100, 200,20); // Riadiaci, koncový bod
```

# Ťah - b zierova krivka

```
bezierCurveTo (cp1X,cp1Y, cp2X,cp2Y, x,y)
```

- dva kontrolné body

[illegible]



# Ťah - obdĺžnik

**rect** (**x,y**, **width,height**)

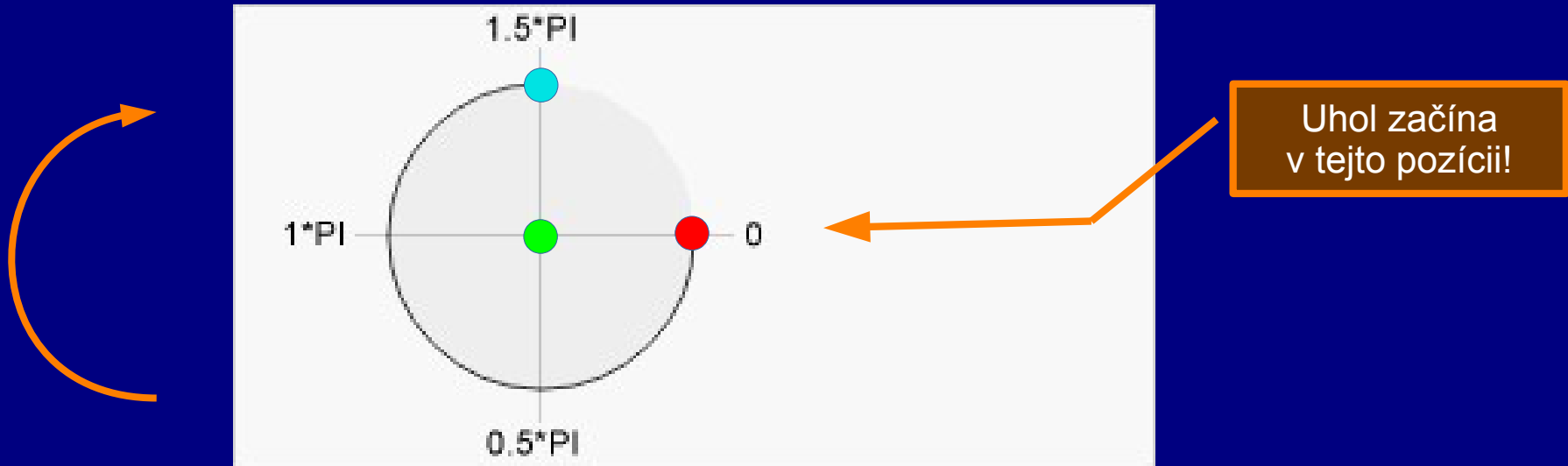
- doplní obdĺžnik do aktuálneho ťahu
- x,y je ľavý horný roh (kam sa najprv presunie)

```
ctx.beginPath ();  
ctx.rect(20,20, 150,100);  
ctx.stroke();
```

# Ťah - kruh a kružnica

`arc (x,y, radius, startAngle, endAngle  
[, anticlockwise])`

- výsek kružnice podľa uhlov v **radiánoch**
- v smere hodinových ručičiek (default)

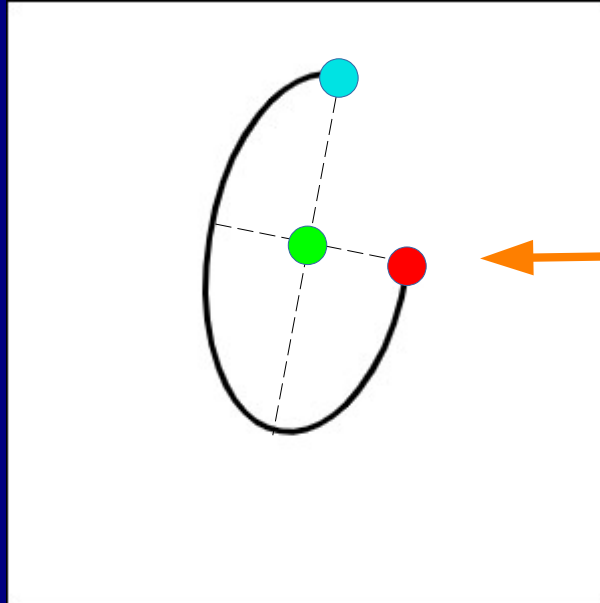


```
ctx.arc (100,75, 50, 0*Math.PI,1.5*Math.PI, false)
```

# Ťah - elipsa

`ellipse (x,y, radiusX,radiusY, rotation, startAngle, endAngle [, anticlockwise])`

- výsek elipsy podľa uhlov v **radiánoch**
- v smere hodinových ručičiek (default)



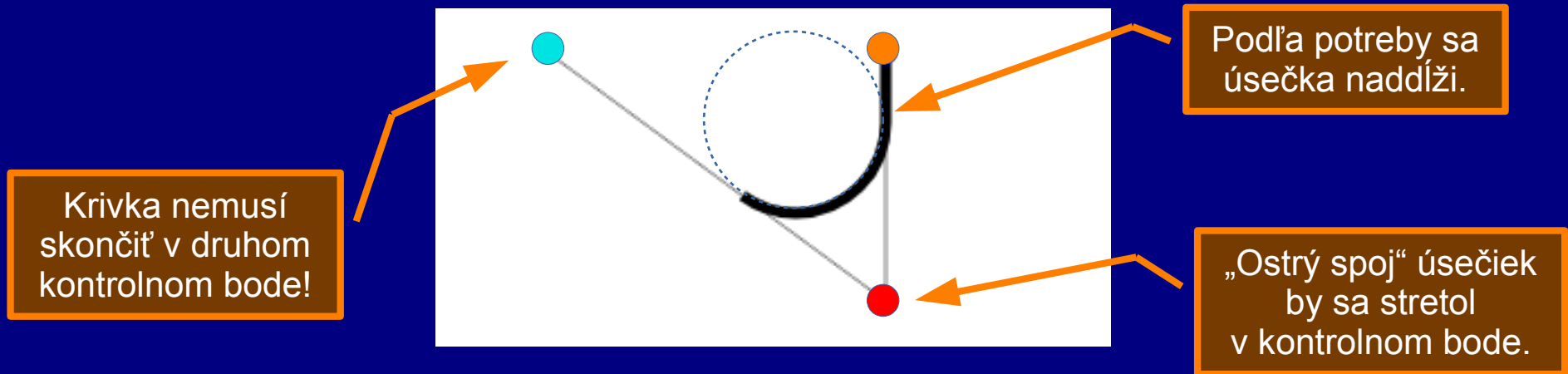
Uhol začína  
v tejto pozícii!

```
ctx.ellipse (100,75, 50,90,0.05*Math.PI, 0*Math.PI,1.5*Math.PI)
```

# Ťah - oblúčková krivka

**arcTo** (**x1,y1**, **x2,y2**, **radius**)

- oblúčkový spoj medzi dvoma úsečkami
- polomer kružnice oblúku



```
ctx.beginPath();  
ctx.moveTo(200,20);           // začiatočný bod  
  
ctx.arcTo(200,130, 50,20, 40);  
  
ctx.lineTo(50,20);           // potiahnutie do koncového bodu
```

# Viac ťahov

- trieda **Path2D**
  - jednoduchšie vykresľovanie

```
p = new Path2D ();

p.moveTo(200,20);
p.lineTo(50,20);
p.rect(50,20, 100, 100);

ctx.stroke (p);

p2 = new Path2D (p);
ctx.fill (p2);

p3 = new Path2D ();
p3.addPath (p);
```



# Atribúty tieňa

## shadowColor

- farba tieňa

## shadowBlur

- rozmazanie tieňa. Výpočtovo náročné!!!

## shadowOffsetX, shadowOffsetY

- posun tieňa
  - > 0 vpravo / dole
  - < 0 vľavo / hore

```
ctx.shadowColor = "black";  
ctx.shadowBlur = 20;  
ctx.shadowOffsetX = 20;  
  
ctx.fillStyle = "red";  
ctx.fillRect (20, 20, 100, 80);
```





# Transformácie súradníc

- aplikujú sa na body (x,y) pri volaní metód
  - treba nastaviť **pred** vytváraním ťahu
- vzájomne sa skladajú
  - prinásobujú sa sprava k pôvodnej matici transformácie  $M$
  - t.j. aplikujú sa v poradí od posledne zadanej transformácie ku prvej

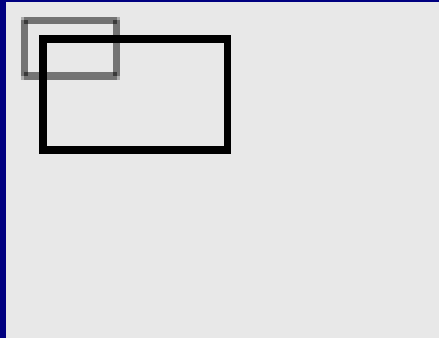
$$M M_1 M_2 \cdots M_N \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \hat{x} \\ \hat{y} \\ 1 \end{pmatrix}$$

# Škálovanie

**scale** (a, d)

- násobenie bodov (ťahu) danými koeficientami

$$M \begin{pmatrix} a & 0 & 0 \\ 0 & d & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \hat{x} \\ \hat{y} \\ 1 \end{pmatrix}$$



```
ctx.strokeRect(5, 5, 25, 15);  
ctx.scale(2, 2);  
ctx.strokeRect(5, 5, 25, 15);
```

# Otočenie

## rotate ( $\alpha$ )

- otočenie bodov (ťahu) o uhol v radiánoch
- stred otáčania je (0,0)

$$M \begin{pmatrix} \cos \alpha & -\sin \alpha & 0 \\ \sin \alpha & \cos \alpha & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \hat{x} \\ \hat{y} \\ 1 \end{pmatrix}$$



```
ctx.rotate (20 * Math.PI / 180); // prevod stupňov na radiány  
ctx.fillRect (50, 20, 100, 50);
```

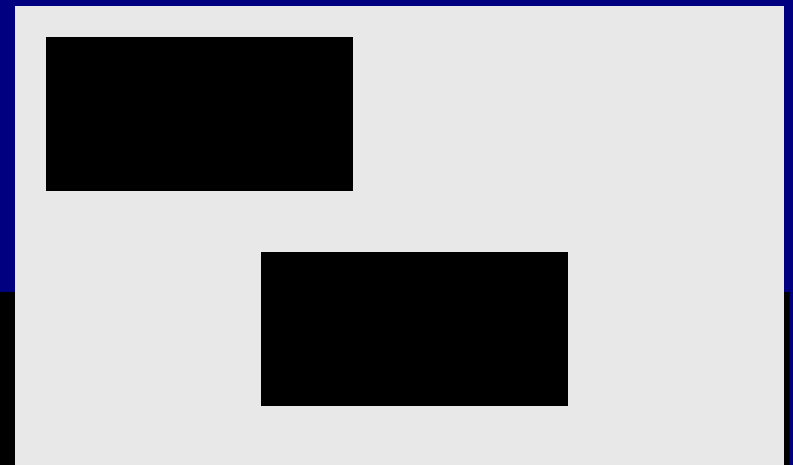
# Posun

**translate** (**e**, **f**)

– posun bodov (ťahu)

$$M \begin{pmatrix} 1 & 0 & e \\ 0 & 1 & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \hat{x} \\ \hat{y} \\ 1 \end{pmatrix}$$

```
ctx.fillRect(10, 10, 100, 50);  
ctx.translate(70, 70);  
ctx.fillRect(10, 10, 100, 50);
```



# Všeobecná transformácia

**transform** (**a**, **b**, **c**, **d**, **e**, **f**)

- prinásobí maticu transformácie (škálovanie, otočenie, posun) k doterajšej matici

$$M \begin{pmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} = \begin{pmatrix} \hat{x} \\ \hat{y} \\ 1 \end{pmatrix}$$

# Nastav maticu

**setTransform** (**a**, **b**, **c**, **d**, **e**, **f**)

– nastavenie matice transformácie

$$M = \begin{pmatrix} a & c & e \\ b & d & f \\ 0 & 0 & 1 \end{pmatrix}$$



# Text

## font

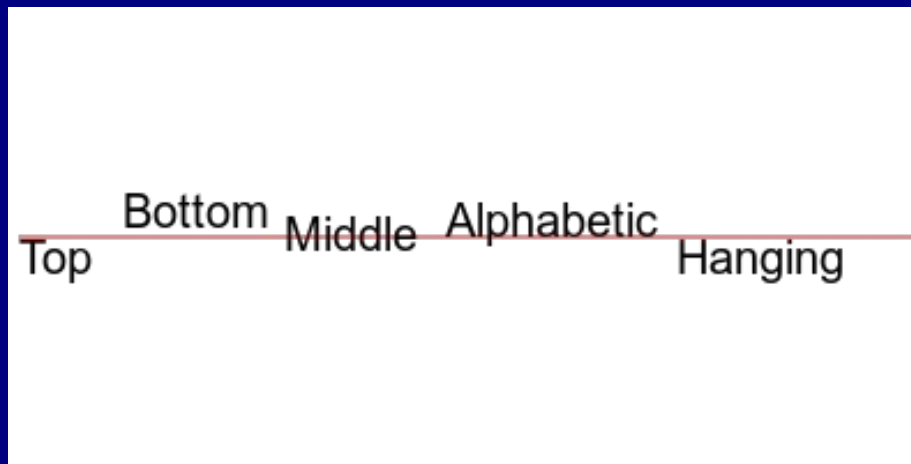
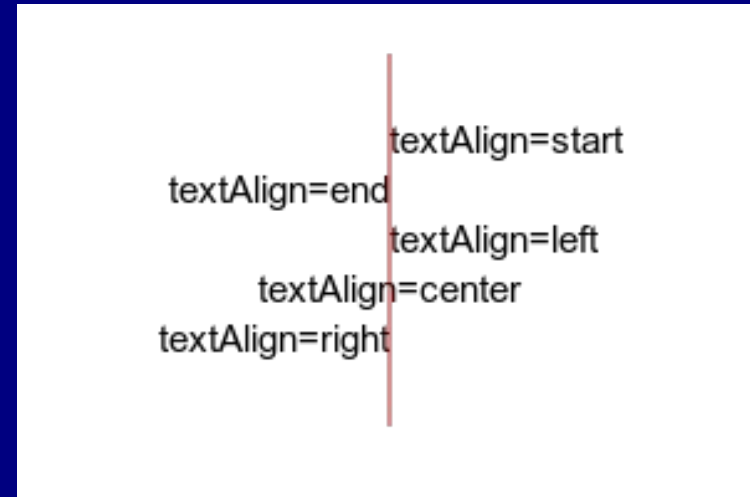
- určenie fontu a jeho modifikácií

## textAlign

- horizontálne zarovnanie vzhľadom na bod výpisu

## textBaseline

- vertikálne zarovnanie vzhľadom na bod výpisu



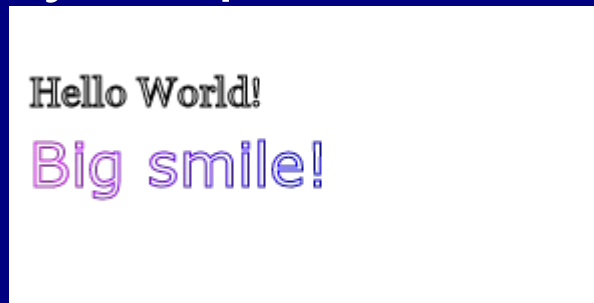


# Text

Snaží sa  
to vtesnať  
do danej  
šírky.

**strokeText** ('Text', x,y [,maxWidth])

- vykreslenie obrysov písmen textu

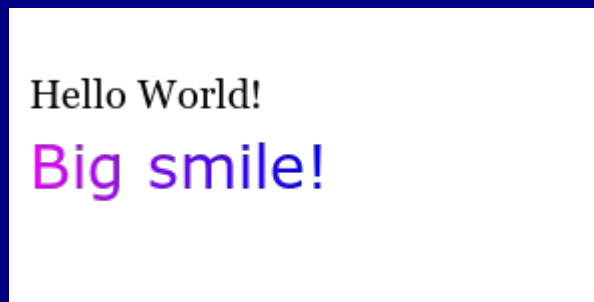


Hello World!  
Big smile!

A white rectangular canvas displaying the text 'Hello World!' in a black serif font and 'Big smile!' in a purple serif font. The text 'Big smile!' is rendered with only its outlines (strokes), making it appear hollow.

**fillText** ('Text', x,y [,maxWidth])

- vykreslenie vyplnených písmen textu



Hello World!  
Big smile!

A white rectangular canvas displaying the text 'Hello World!' in a black serif font and 'Big smile!' in a purple serif font. The text 'Big smile!' is rendered as solid, filled-in letters.



# Obrázok (img element)

**drawImage** (img, x,y)

- vykresli celý obrázok na danú pozíciu x,y

**drawImage** (img, x,y, width,height)

- obrázok sa najprv transformuje do požadovaných rozmerov width, height

**drawImage** (img, sx,sy, swidth,sheight, x,y, w,h)

- z obrázku sa najprv vyreže daná oblasť (sx,sy, swidth,sheight) a vykreslí sa na pozíciu x,y so šírkou w a výškou h.



# Práca s pixelmi canvasu

**getImageData** (*x,y, width,height*)

- vykopíruje danú oblasť canvasu do špeciálneho objektu **ImageData**, ktorý vráti

**putImageData** (**ImageData**, *x,y*)

- vloží obrázok z objektu triedy **ImageData**

Na súradnice  
sa neaplikuje  
transformačná  
matica!

**createImageData** (*width,height*)

- vytvorí nový objekt-obrázok **ImageData**


```
imgData = ctx.getImageData (10, 10, 50, 50);  
ctx.putImageData (imgData, 10, 70);
```

# Objekt triedy **ImageData**

**.width** – šírka

**.height** – výška

**.data** – údaje o pixeloch. Jednorozmerné pole čísiel (bajtov), kde jednému pixelu zodpovedajú 4 za sebou nasledujúce hodnoty [R,G,B,A] z intervalu 0-255.



Možno využiť  
metódy poľa.

```
// Hodnota RGBA v bode x,y
```

```
Red    = data[y*4*width + x*4 +0]
```

```
Green  = data[y*4*width + x*4 +1]
```

```
Blue   = data[y*4*width + x*4 +2]
```

```
Alpha  = data[y*4*width + x*4 +3]
```



jeden pixel na pozícii (x,y)



# Context

## save ()

- uloží stav kontextu aj ťahu do zásobníka

## restore ()

- obnoví uložené z vrchu zásobníka

- vhodné pre funkcie

```
ctx.save ();  
ctx.fillStyle = '#00ff00';  
//...  
ctx.restore ();
```



# Kompozícia

- kombinovanie novokresleného útvaru (source) s pôvodným obsahom canvasu (destination)

## globalCompositeOperation

- rôzne módy: source-over, destination-over, ...

## globalAlpha

- nepriesvitnosť (opacity)  $0 \leftrightarrow 1$  ( $1 \rightarrow$  nepriesvitné)

## clip ()

- podľa aktuálneho ťahu vytvorí oblasť a iba do nej sa bude v ďalšom vykresľovať

```
ctx.rect(50, 20, 200, 120);  
ctx.clip();
```

Ďalšie volania metódy `clip()` pridávajú oblasti. Je ideálne si kontext najprv uložiť a potom obnoviť.



# Animovanie

- využitie **setInterval()**
  - nemusí stíhať dané časovanie → nakopenie zmien
- opätovné volanie **setTimeout()**
  - nekopia sa zmeny, nie je to synchronizované s vykresľovaním stránky
- **handle = requestAnimationFrame (fnc)**
  - prehliadač **jedenkrát** zavolá funkciu **fnc** pri vykresľovaní nasledujúcej fázy vzhľadu stránky
  - opätovne treba spúšťať **requestAnimationFrame()**
  - možnosť dosiahnuť maximálne možné fps (cca 60fps)
  - ak je otvorená stránka na pozadí, nevolá sa
  - zrušenie čakateľa **cancelAnimationFrame (handle)**
  - **fnc** dostane ako parameter aktuálny čas v [ms] od otvorenia stránky

```
const elem = document.getElementById('ElementYouWantToAnimate');  
elem.style.position = 'absolute';
```

```
const start = null;
```

```
function step (timestamp) { //argument je čas v milisekundách
```

```
    if (!start) start = timestamp;
```

```
    let progress = timestamp - start;
```

```
    elem.style.left = Math.min (progress / 10, 200) + 'px';
```

```
    if (progress < 2000) {  
        // opäť treba zavesiť callback  
        window.requestAnimationFrame(step);  
    }
```

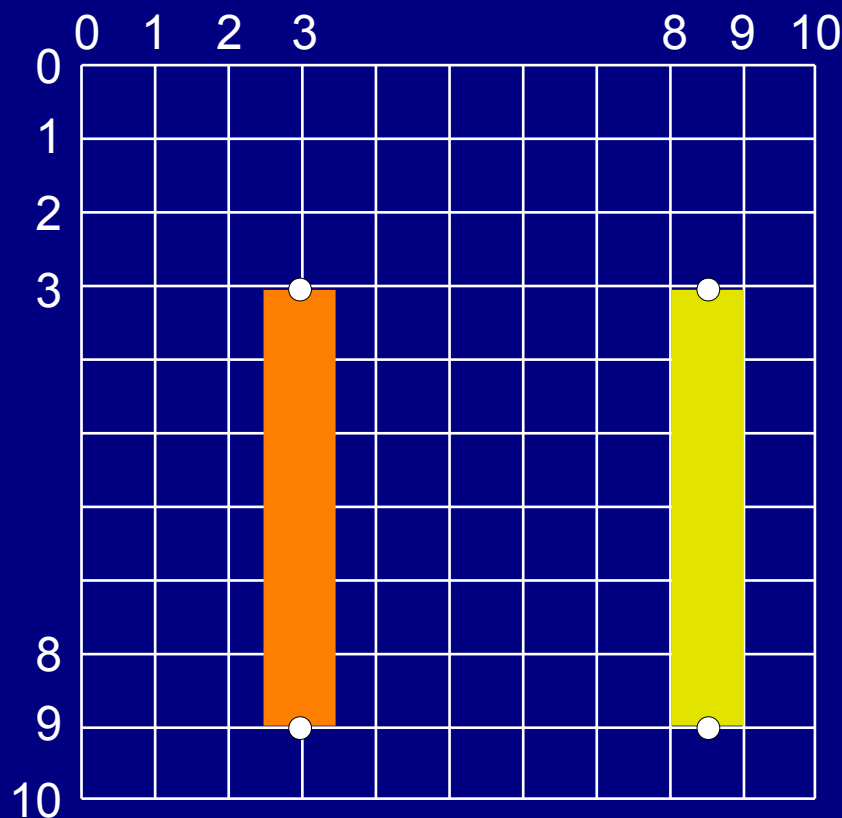
```
}
```

```
window.requestAnimationFrame(step);
```

# Odporúčania

- používať **celočíselné** súradnice pri drawImage()
  - lebo anti-aliasing
- používať **offscreen** canvas
  - napr. príprava fázy postavičky
- používať **prekrývajúce** sa canvasy
  - napr. viacero postavičiek
- veľké pozadia dať ako div element **za** canvas
- texty používať cez HTML elementy **pred** canvas

# Súradnice sú medzi pixelmi



```
ctx.lineWidth = 1  
ctx.lineCap = 'butt'
```

```
ctx.beginPath ()  
ctx.moveTo (3, 3)  
ctx.lineTo (3, 9)  
ctx.stroke ()
```

Šírka čiary sa rozdelí na obe strany. T.j. bude rozmazaná.  
Ostrá čiara bude pre súradnice (8.5, 3) (8.5, 9).



Ďakujem za pozornosť