Programmation Graphique Interactive et Animations pour le Web

Mountaz Hascoët, Université de Montpellier

Plan

- SVG et le canvas
  - deux approches différentes de la création de graphiques
- La gestion des évènements utilisateurs
  - typologie d'évènements
  - modèle de gestion des évènements
  - des évènements aux interactions
- Les interactions en pratique
  - Interaction avec le canvas
    - un exemple de gestion des évènements utilisateurs
  - Animation pour le canvas
- Animations

SVG – scalable vector graphics

- SVG est aux objets graphiques ce que HTML est au texte
- SVG est multi-formes
  - fragment SVG intégré en HTML5
    - élément/fragment intégré dans un dom
    - élément/fragment intégré dans un Path2D
  - document SVG autonome
    - export possible de la plupart des logiciels de dessin vectoriel
  - SVG intégré dans un document XML
- Remarque
  - ce cours ne parle que de l'utilisation de SVG comme fragment intégré en HTML

SVG – éléments

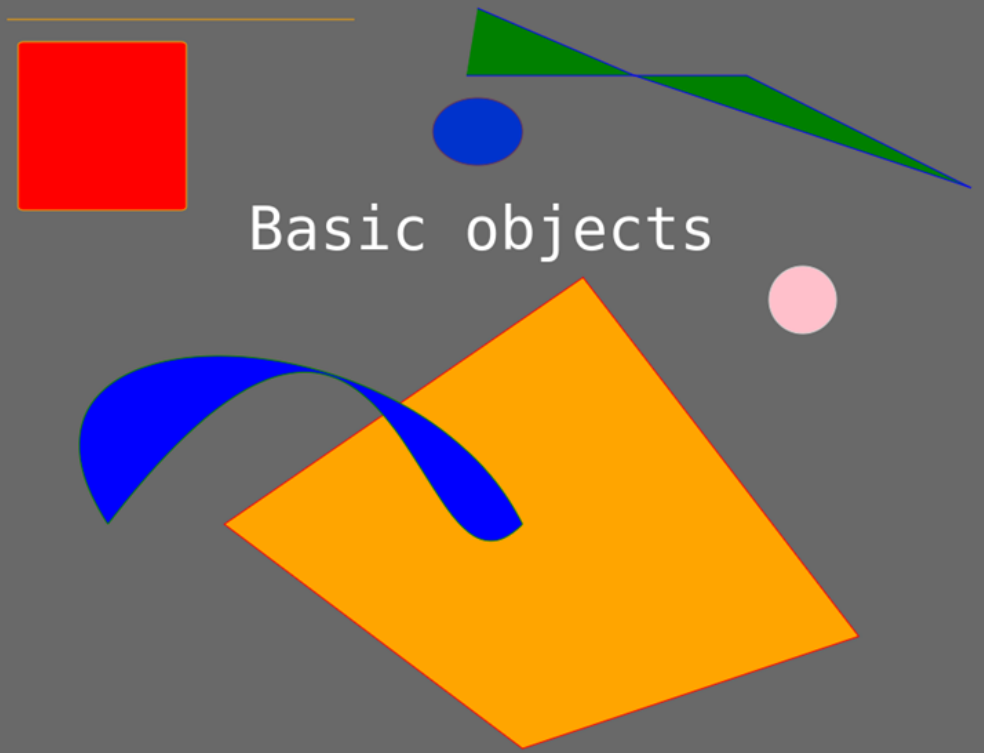
- Une image est composée d'éléments SVG
  - objets graphiques « classiques »
    - `rect, circle, ellipse, polyline, polygon,`
  - texte
    - `text, tspan et textPath`
  - contenus « embarqués »
    - `image`
  - d'objets graphiques particuliers
    - `path`
      - cet élément permet de définir des objets graphiques plus complexes que les objets classiques
  - des groupes d'éléments SVG
    - `g`

SVG – attributs des éléments

- Les éléments SVG ont des attributs
  - positions
  - dimensions
  - remplissage
  - contour

```
<ellipse
    cx="60" cy="80"
    rx="40" ry="30"
    fill="orange"
    stroke="gray"
/>
```

SVG en pratique



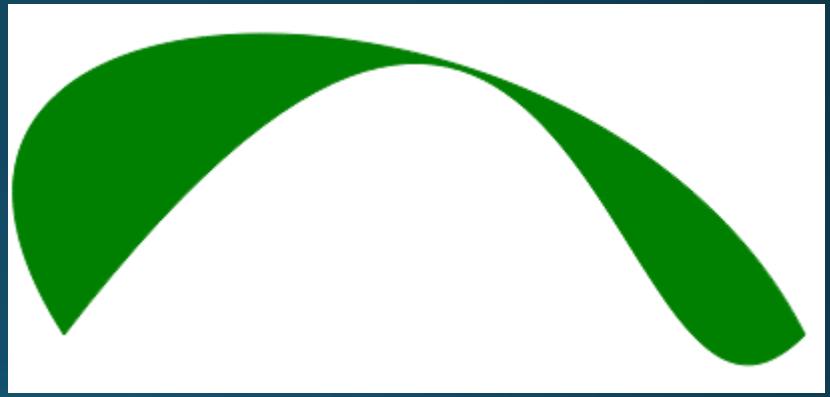
SVG – l'élément **path** (1/2)

- Permet de définir des objets graphiques personnalisés
  - path ~ le tracé (« chemin du stylo »)
  - décrit par une séquence de commandes de dessin avec leurs paramètres
  - description compacte

```
<path  d="M 130 500 C 1 300 400 300 500 500 C 400 600 400 150 130 500"
stroke="lightGray" fill="lightGray"/>
```

SVG – l'élément **path** (2/2)

- Commandes de base
  - M x y
  - L x y
  - Q x1 y1 xe ye
  - C x1 y1 x2 y2 xe ye



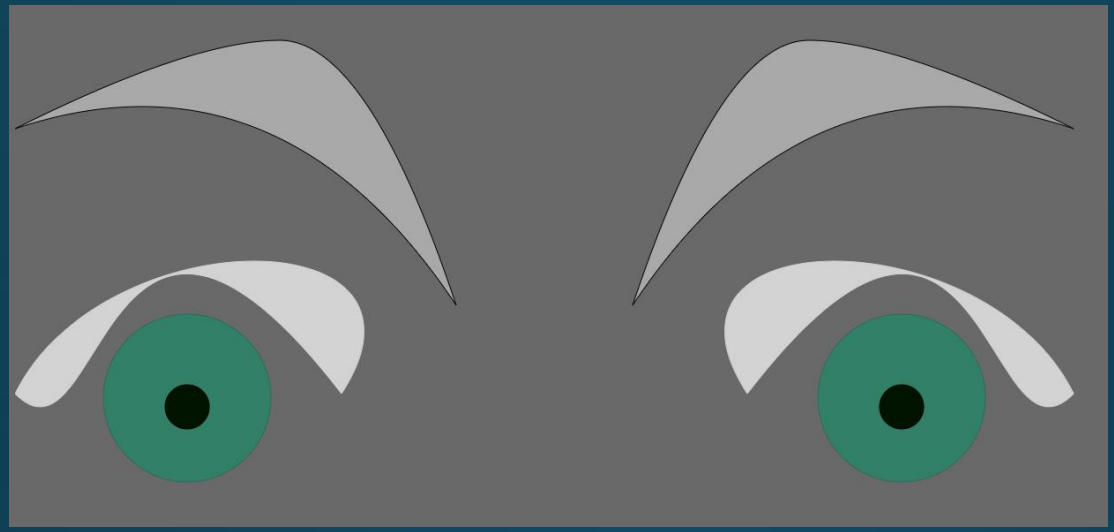
```
<path  d="M 130 500 C 1 300 400 300 500 500 C 400 600 400 150 130 500"
stroke="lightGray" fill="lightGray"/>
```

Les systèmes de coordonnées

- <svg> attributs width et height
  - définit les dimensions « externes », c'est-à-dire pour le navigateur
  - en l'absence de viewbox
  - les coordonnées « internes » sont comprises entre 0,width et 0,height
- noms de commandes en majuscules ou en minuscules
  - majuscule
    - repère global
  - minuscule
    - repère courant
- Transformations géométriques
  - translate, rotate, scale, et autres transformations linéaires
  - agissent sur les coordonnées
  - les coordonnées définies dans svg ne sont donc pas toujours les coordonnées des objets affichés

Remarque: ce cours fait l'hypothèse d'une css "neutre" et ne parle pas des modifications possibles avec css

Exemple



```
<svg width="1280" height="1280">
  <g transform="translate(500,0) scale(-1,1)">
    <path d="M0 400 Q 100 100 200 100 Q 300 100 500 200 Q 200 100 0 400" stroke="black" fill="darkGrey"/>
    <path  d="M130 500 C 1 300 400 300 500 500 C 400 600 400 150 130 500" stroke="lightGray" fill="lightGray"/>
    <circle class="eyes" cx="305" cy="505" r="95" stroke="rgba(51, 104, 87, 1)" fill="rgba(51, 128, 104, 1)" />
    <circle cx="305" cy="515" r="25" stroke="black" fill="rgba(0, 20, 0, 1)" />
  </g>
  <g transform="translate(700,0)">
    <path d="M0 400 Q 100 100 200 100 Q 300 100 500 200 Q 200 100 0 400" stroke="black" fill="darkGrey"/>
    <path d="M130 500 C 1 300 400 300 500 500 C 400 600 400 150 130 500" stroke="lightGray" fill="lightGray"/>
    <circle class="eyes" cx="305" cy="505" r="95" stroke="rgba(51, 104, 87, 1)" fill="rgba(51, 128, 104, 1)" />
    <circle cx="305" cy="515" r="25" stroke="black" fill="rgba(0, 20, 0, 1)" />
  </g>
```

Conclusion sur SVG

- Points forts
  - description vectorielle abrégée mais lisible
  - intégration à HTML 5
  - format de description extensible
  - compatibilité
    - format d’export de la plupart des logiciels de dessin vectoriels
    - utilisé par de nombreuses bibliothèques graphiques

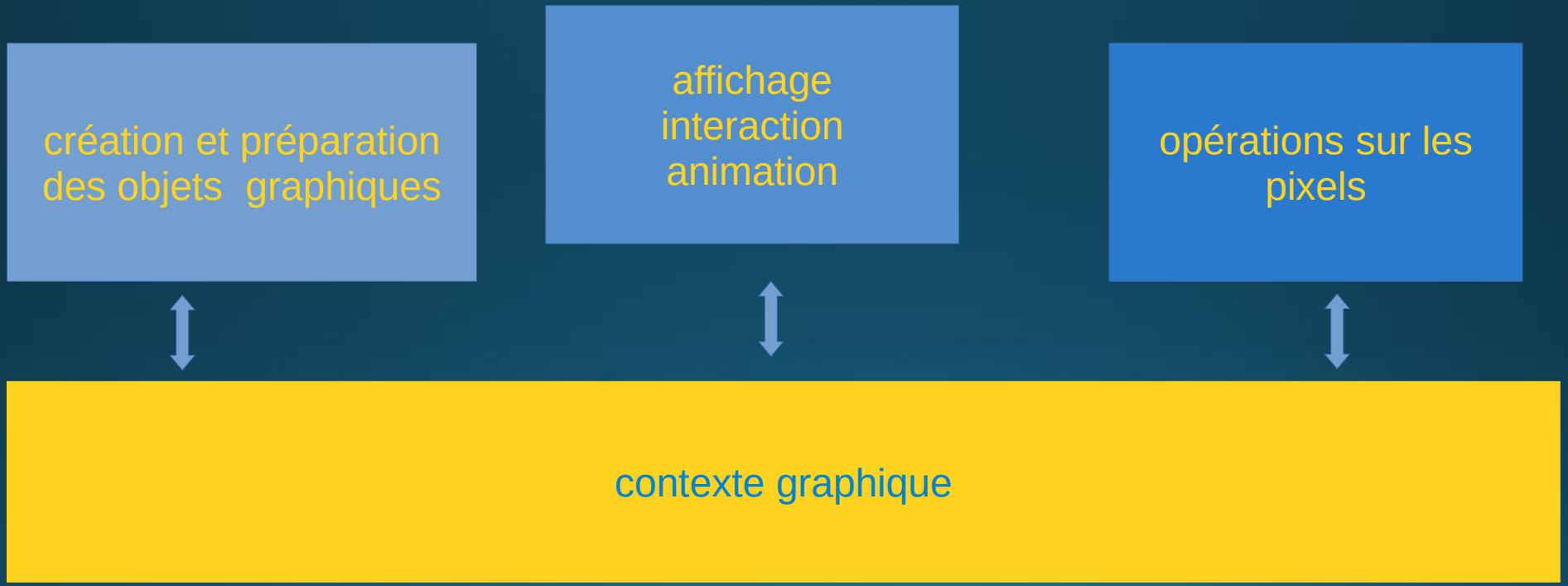
et maintenant, dans un style un peu différent,

Le Canvas

Canvas

- le canvas est un élément HTML
  - permet d'afficher des objets graphiques
    - dont des éléments svg
  - mais ...
- les objets graphiques du canvas
  - ne sont pas des éléments du DOM
- la création / affichage / interaction / animation des graphiques est gérée en javascript
  - ce qui n'empêche pas de mettre des éléments HTML qui pourraient s'ajouter aux éléments graphiques

Canvas – principes de base



Le contexte graphique est un objet javascript

- structure de données centrale qui permet de
  - créer les objets des graphiques
    - **`beginPath`, `fillRect`, `fillText`, `ellipse`,** …
  - modifier les attributs graphiques du dessin courant
    - **`strokeStyle`, `fillStyle`,** …
  - afficher les objets graphiques
    - **`fill`, `stroke`**

https://developer.mozilla.org/en-US/docs/Web/API/CanvasRenderingContext2D

Le contexte graphique en pratique

```
let canvas1 = document.getElementById('s1');
let cg = canvas1.getContext('2d');
cg.beginPath();
cg.moveTo(a.x, a.y);
cg.lineTo(b.x, b.y);
cg.strokeStyle = 'rgb(0,200,200)';
cg.stroke();
```

Remarque sur le contexte graphique

- différents types de contextes graphiques existent dont
  - 2D
    - dimension 2, modes vectoriel et bitmap
  - webgl – webgl2
    - dimension 3, vectoriel, bitmal, avec textures - implémentation des versions d'OpenGL ES
  - webgpu
    - un modèle d'affichage alternatif aux approches GL
- ce cours et les td liés n'utilisent que le contexte graphique 2D

- source : https://html.spec.whatwg.org/multipage/canvas.html#the-canvas-element

1. Le contexte graphique du Canvas
2. La création des objets graphiques
    a. le tracé courant (par le contexte graphique)
    b. les instances de Path2D (indépendantes du contexte graphique)
    c. les instances de Path2D initialisées par commandes SVG

Création des objets graphiques
le tracé (path) courant

- Commence avec l'appel de méthode beginPath et se termine avec l'appel suivant

```
let cg = canvas1.getContext('2d');

cg.beginPath();
cg.moveTo(a.x, a.y);
cg.lineTo(b.x, b.y);
cg.stroke();
```

```
cg.beginPath();
cg.moveTo(c.x, c.y);
cg.arcTo(...);
```

Création des objets graphiques Path2D

- Création d'instances de Path2D et modification du tracé avec les méthodes de Path2D (très proches des méthodes du contexte graphique)

```
shape = new Path2D()
shape.moveTo(1.3 * u, 5 * u);
shape.bezierCurveTo( 1.5, 3*u, 4*u, 3*u, 5*u, 5*u);
shape.bezierCurveTo(4*u, 6*u, 4*u, 1.5*u, 1.3*u, 5*u);
```

Remarque : dans cet exemple, u représente une unité (~ en pixels)

```
class Shape {
   constructor(){
     this.sourcil = new Path2D();
     this.sourcil.moveTo(0, 4 * u);
     this.sourcil.quadraticCurveTo(u, u, 2 * u, u);
     this.sourcil.quadraticCurveTo(3 * u, u, 5 * u, 2 * u);
     this.sourcil.quadraticCurveTo(2 * u, u, 0, 4 * u);

     this.oeil = new Path2D()
     this.oeil.moveTo(1.3 * u, 5 * u);
     this.oeil.bezierCurveTo(1.5, 3*u, 4*u, 3*u, 5*u, 5*u);
     this.oeil.bezierCurveTo(4*u, 6*u, 4*u, 1.5*u, 1.3*u, 5*u);

     this.iris = new Path2D();
     this.iris.arc(3.05*u, 5.05*u, .95*u, 0, 2 * Math.PI);

     this.pupille = new Path2D();
     this.pupille.arc(3.05*u, 5.15*u, .25*u, 0, 2 * Math.PI);
   }
 }
```

Remarque : dans cet exemple, u représente une unité (~ en pixels)

Création des objets graphiques Path2D

- Création d'instances de Path2D initialisées avec la syntaxe SVG

```
shape = new Path2D('M 130 500
C 1 300 400 300 500 500
C 400 600 400 150 130 500');
```

- 1. Le contexte graphique du Canvas
- 2. La création des objets graphiques
- 3. L'affichage d'objets graphiques

L'affichage d'un tracé quelconque (path)

- Méthodes du contexte graphique
  - afficher le tracé (path) courant

```
cg.fill();      // remplissage
cg.stroke(); // contour
```

  - afficher un objet Path2D

```
shape = new Path2D()
...
cg.fill(shape);
...
cg.stroke(shape);
```

Objets graphiques prédéfinis

- Pour afficher des objets graphiques les plus fréquents
  - image, texte, rectangle, ellipse, etc
- On utilise d'autres méthodes du contexte graphique

```
void ctx.drawImage(image, dx, dy);
void ctx.fillText(texte, x, y );
void ctx.fillRect(x, y, largeur, hauteur);
void ctx.ellipse(x, y, rx, ry, rotation, angleDébut,
angleFin, antihoraire);
```

src: https://developer.mozilla.org/fr/docs/Web/API/CanvasRenderingContext2D/

Plan

- SVG et le canvas
  - deux approches différentes de la création de graphiques
- La gestion des évènements utilisateurs
  - typologie d'évènements
  - modèle de gestion des évènements
  - des évènements aux interactions
- Les interactions en pratique
  - Interaction avec le canvas
    - un exemple de gestion des évènements utilisateurs
  - Animation pour le canvas
- Animations

Evènements (définition)

- un évènement représente des informations utiles qui ne sont pas connues avant le déroulement d'un programme.
- les évènements sont typés pour représenter des informations de natures différentes
- les typologies d'évènements évoluent pour prendre en compte les nouveaux contextes
  - https://www.w3.org/TR/uievents/#event-types

- les actions des utilisateurs sont représentées par des évènements

Types d’évènements en javascript

Mouse Event Types

```
click
dblclick
mousedown
mouseenter
mouseleave
mousemove
mouseover
mouseup
```

Wheel Event Types

```
wheel
```

Keyboard Event Types

```
keydown
keyup
```

User Interface Event Types

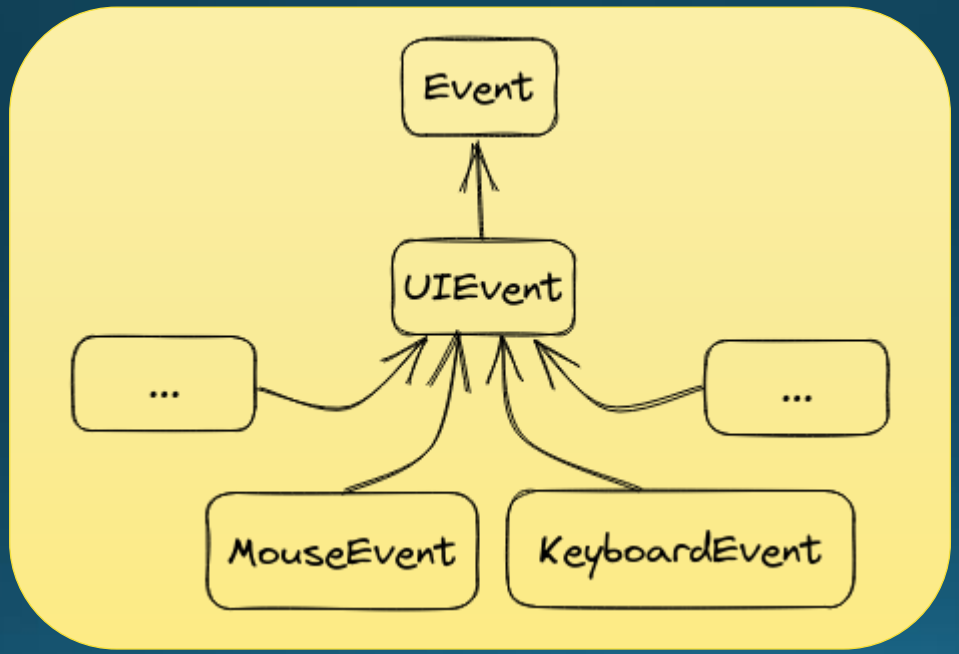
```
load
unload
abort
error
select
resize
scroll
```

Drag-and-Drop Events

```
dragstart, drag, dragend
dragenter, dragleave, dragover
```

…

Les classes d’évènements



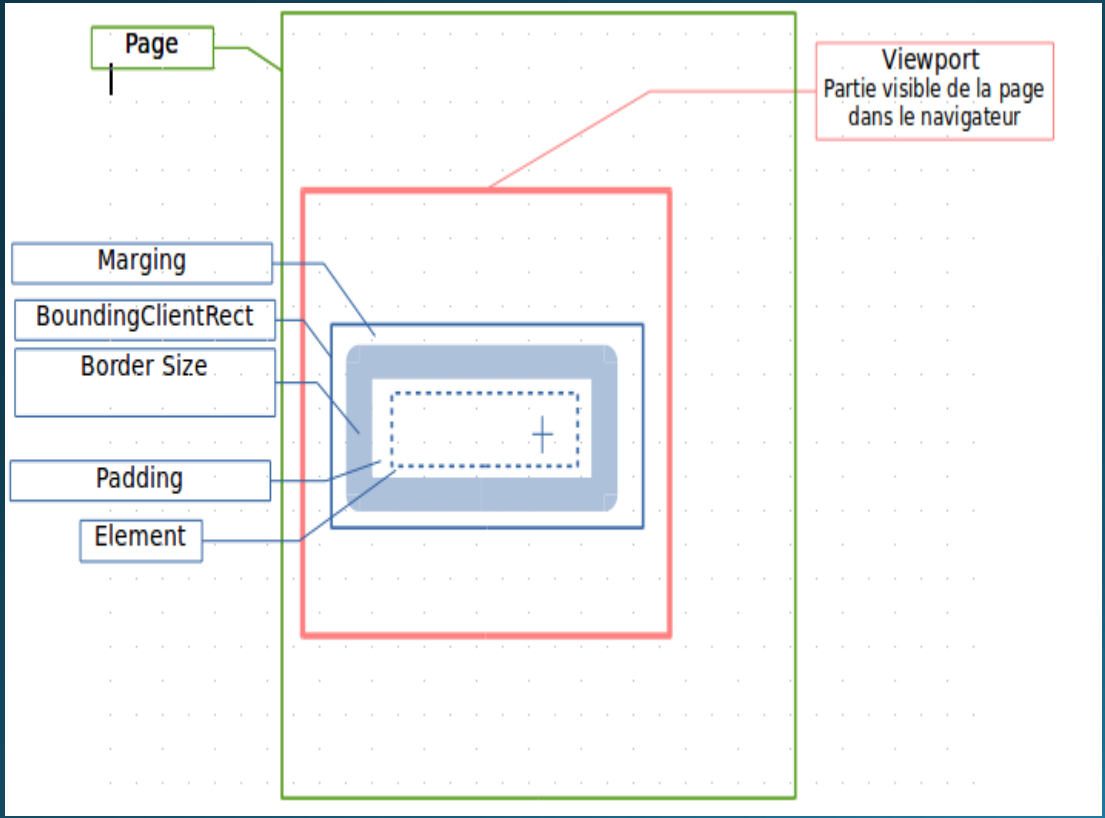
source https://www.w3.org/TR/uievents/

La class Event

- event.target
  - target de l'évènement ie élément le plus profond des éléments du chemin de propagation de l'évènement
- event.timeStamp
- event.currentTarget
  - élément du chemin de propagation auquel l'évènement se trouve à gérer
- event.type
  - retourne le type de l'évènement (click, hasChanged, ...)
- event.eventPhase
  - renvoie un entier identifiant : 1. → capture, 2 → at Target, 3 → bubbling
- ...

source : https://dom.spec.whatwg.org/#event [last accessed 2022-03-16]

La class MouseEvent



- screenX, screenY → coordonnées par rapport à l'origine de l'écran
- pageX, pageY → coordonnées par rapport à l'origine de la page
- clientX, clientY → coordonnées par rapport à l'origine du viewport
- **offsetX, offsetY -> coordonnées dans l'élément**
- ctrlKey, shiftKey, altKey, metaKey → booléen qui indique quelles sont les touches de contrôle activées
- button → indique les boutons de la souris activés (valeurs possibles 0,1,2,3,4)

source

https://www.w3.org/TR/uievents/

Gestion des évènements #1 - html

- écriture dans l'attribut HTML de l'élément
  -
- avantages
  - augmentation automatique de la chaîne de portée de la callback avec le formulaire englobant lorsque la callback est associée à un élément de formulaire.
    - accès direct à tous les attributs de l'élément du formulaire ainsi qu'aux autres champs du formulaire recevant l'évènement.
- inconvénients
  - comportement différent en fonction des navigateurs
  - code difficile à maintenir et/ou à faire évoluer
    - peu modulaire

Gestion des évènements #2 – DOM 0

- définition par les méthodes de l'élément
  - getElementById("bouton").click = redraw;
- avantages et inconvénients
  - la callback est considérée comme une méthode de l'élément qui reçoit l'évènement.
    - portée de l'élément
    - this référence, en théorie, l'élément qui a reçu l'évènement.

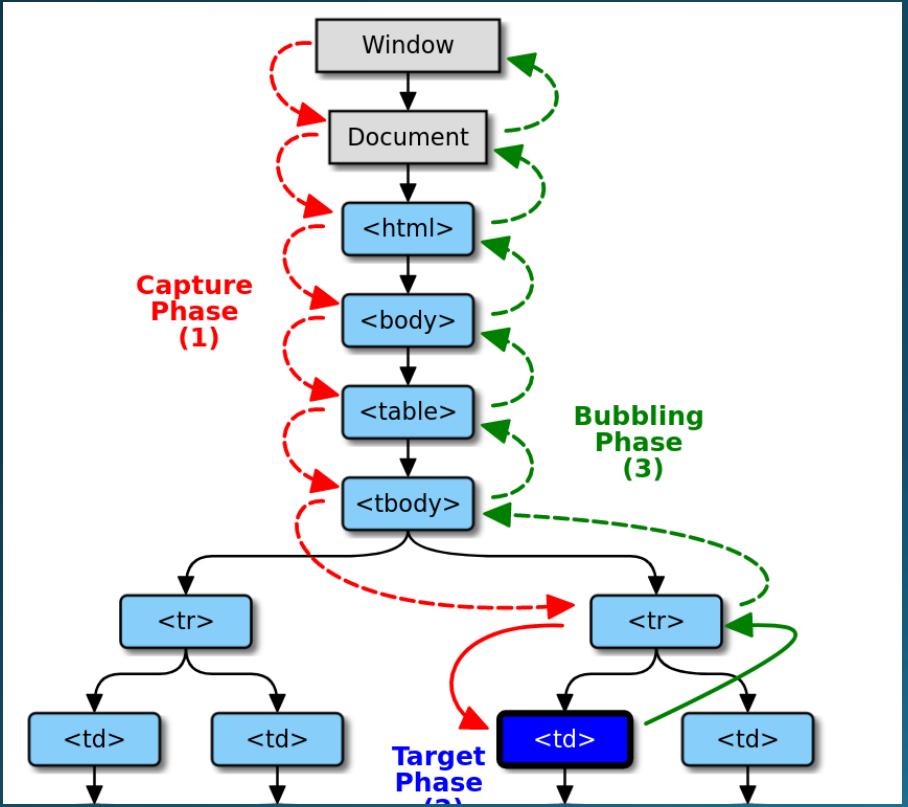
Technique #3 – DOM 2

- Gestion par délégation

```
canvas.addEventListener("click",redraw, false);
```

- permet de lier: élément du dom + type d'évènement + callback
- choix de la phase de la propagation (capture ou bubble)
- avantages
  - modularité, réutilisabilité
    - permet de développer stratégies de gestion de l'interaction plus élaborées
  - portée de l'élément (ou autre en fonction du contexte)

Propagation des évènements



- Lorsqu'un évènement est émis sur un élément du dom, il est propagé à tous les éléments qui le contiennent
- Des gestionnaires d'évènements peuvent être associés à n'importe quel élément
- Chaque gestionnaire d'évènement associé à un élément du dom se trouvant sur le chemin de propagation reçoit l'évènement ...
- ... sauf lorsque la propagation est stoppée avant
- Tout gestionnaire d'évènement peut stopper la propagation

source - https://www.w3.org/TR/uievents/#event-flow

Plan

- SVG et le canvas
  - deux approches différentes de la création de graphiques
- La gestion des évènements utilisateurs
  - typologie d'évènements
  - modèle de gestion des évènements
  - des évènements aux interactions
- Les interactions en pratique
  - Interaction avec le canvas
    - un exemple de gestion des évènements utilisateurs
  - Animation pour le canvas
- Animations

L'interaction en pratique #1

Objectif : suivre les mouvements de la souris et afficher les coordonnées de la souris dans le canvas

1. Créer une variable js qui référence le canvas html
2. Lier le `canvas` avec le gestionnaire d'évènement pour la gestion d'évènements de type `move`
3. Ecrire le code source du gestionnaire d'évènement

```
let canvas = document.getElementById("canvas_id");
canvas.addEventListener("mousemove", e => handleMove(e));

function handleMove(e) {
    let x = e.offsetX, y = e.offsetY;
    drawTextCentered(x + "," + y);
}
```

L'interaction en pratique #2

Objectif : un clic affiche les points de contrôle des courbes, un nouveau clic les cache

1. Créer une variable js qui référence le canvas html
2. Lier le `canvas` avec le gestionnaire d'évènement pour la gestion d'évènements de type …

   `click` ou `mousedown` ?
3. Ecrire le code source du gestionnaire d'évènement

   quand afficher ? quand cacher?

L'interaction en pratique #2

Sur quel type d'évènements : `click` ou `mousedown` ?

> `click est déclenché après que les évènements mousedown et mouseup aient été déclenchés.`
> `Source: https://developer.mozilla.org/.../click_event`

=> pour un feedback immédiat et pour gérer la suite des interactions possible avant le `mouseup`, `mousedown`, est la meilleure option

L'interaction en pratique #2

- Le gestionnaire d'évènement doit-il afficher ? où cacher?
  - premier clic -> affiche, clic d'après -> cache
  - premier clic? Ou clic d'après?
    - cette information n'est pas dans l'évènement
    - c'est une information typique du contexte d'interaction
- Problème fréquent
  - les évènements émis sont discrets, indépendants
  - les callbacks qui gèrent les évènements, ont les informations apportées par les évènements.
    - mais, a priori, elles n'ont pas la mémoire du contexte d'interaction
  - Les interactions ne peuvent être mises en œuvre sans la mémoire du contexte d'interaction

L'interaction en pratique #2

- Comment passer de la gestion d'évènements à la gestion d'interaction?

Vaste débat …

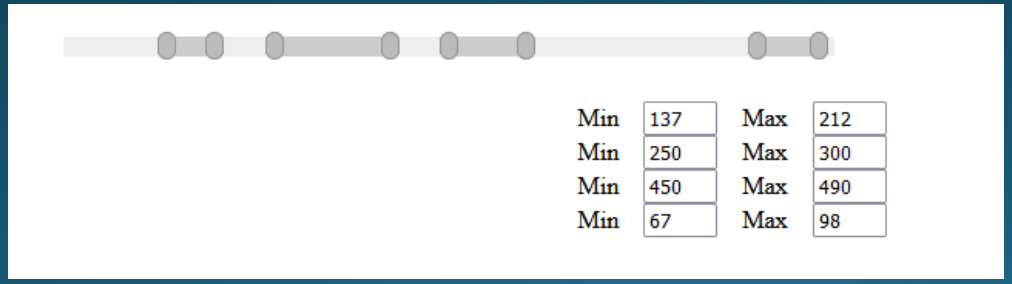
… mais, pour les exercices réalisés dans ce module

- création d'une classe qui gère les interactions
  - Assure la gestion des évènements
  - Et la représentation et l'encapsulation du contexte d'interaction.

```
if (this.state === "idle") {
    this.#controlsOn = !this.#controlsOn;
    this.repaint();
}
```

L'interaction en pratique #3

- Objectif : création d'un slider multiple



L'interaction en pratique #3

- Etapes de l'implémentation
  1. Création du DOM en html et javascript
  2. Liaison des gestionnaires d'évènements avec les éléments du DOM
  3. Modification des attributs des éléments du DOM ou modification du style css des éléments par les gestionnaires d'évènements.
  4. Encapsulation du tout (DOM + gestion d'évènements) dans un widget réutilisable

L'interaction en pratique #3

Création du DOM

```
<div id="slider">
   <div class="sliderBg" style="width: 500px;">
      <span class="range" style="margin-left: 137px; width: 75px;">
           <span class="start-handle"></span>
           <span class="end-handle"></span>
      </span>
…
      <span class="range" style="margin-left: 67px; width: 75px;">
           <span class="start-handle"></span>
           <span class="end-handle"></span>
      </span>
  </div>
</div>
```

L'interaction en pratique #3

Modification des attributs des éléments du DOM ou modification du style css des éléments par les gestionnaires d'évènements.

```
.range {
    position: absolute;
    top:25px;
    display: inline-block;
    height: 13px;
    background-color: #CCC;
}
```

```
function updateStyle() {
    slider.style.marginLeft = min +"px";
    let w = (max - min);
    slider.style.width =  w +"px"; }
```