

Programmation Multimédia

R5.A.06, R5.C.04 - Prog multimédia - CM n°1

Sensibilisation à la programmation multimédia

Manipulation d'images 2D, 3D ; colorimétrie

Press Space for next page →

Introduction - Programme national

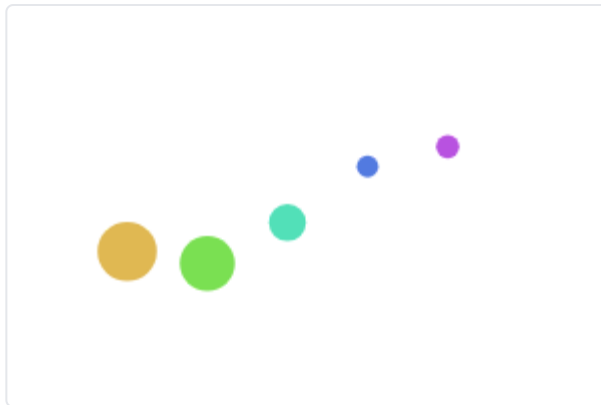
Le PPN est très vague et ne cible pas des compétences typiques du diplôme, en particulier la programmation.

2024 : Initiation a OpenGL, création d'un "vrai" jeux style spaceshooter -> niveau insuffisant.

Donc, le volume horaire, les prérequis mathématiques et l'utilité professionnelle ne permettent pas d'envisager des cours de traitement et de synthèse d'images.

Data Visualization - Programmation de graphiques dynamiques :

- Dessinés en JavaScript via SVG / HTML5
- Animés et interactifs



Visualisation de données

La **data visualization** consiste à traduire des données en graphiques :

- Courbes, histogrammes
- Schémas, cartes
- Graphiques interactifs

De manière à les rendre **facilement compréhensibles**.

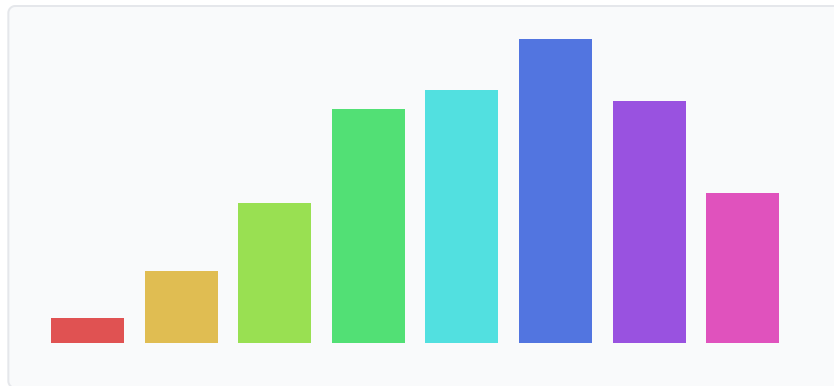
Outils existants

- Chart.js, canvasJS, amCharts
- Mais on est prisonnier de leurs fonctionnalités

Créer des **graphiques non conventionnels**, spécifiques aux besoins professionnels.

Les données changent → création **dynamique** d'éléments :

- Lignes, rectangles, courbes...
- Après le chargement de la page



Concepts du cours

Technologies utilisées

- Images vectorielles SVG : balises XML de HTML5
- JavaScript : modifier dynamiquement
- Manipulation du DOM : modification en temps réel
- Bibliothèques d'animation : GSAP, D3.js
- Les éléments restent manipulable via CSS / SCSS

Avantages

- Consolidation des connaissances existantes
- Applications pratiques et visuel direct
- Potentiel professionnel élevé
- Technologies modernes du web



Plan du cours

- 1 ****Découverte des images SVG**** d'un point de vue programmation
- 2 ****Modification du DOM**** en JavaScript
- 3 ****Utilisation de GSAP**** pour animer ces images
- 4 ****Utilisation de d3.js**** pour dessiner des diagrammes complexes (a discuter)

Ce cours = points 1 à 3

Cours n°2 = point 4

2 TD et 2 TP suivent ce plan :

- TD = découverte guidée des concepts
- TP = mini-projets notés à rendre sur Git

Concept d'image



Images matricielles (raster/pixmap)

- Définies par un tableau rectangulaire de pixels
- Chaque pixel a une couleur
- Résolution fixe
- Taille de fichier importante

Images vectorielles

- Définies par des formes géométriques
- Cercles, rectangles, courbes, textes...
- Résolution indépendante
- Modifiable et animable par script

Définition formelle des images vectorielles

Principe fondamental

Une image vectorielle est définie par un ensemble de **primitives géométriques** :

- Points, lignes, courbes
- Rectangles, cercles, polygones
- Textes avec propriétés typographiques

Avantages pour la programmation

- **Modularité** : chaque élément est indépendant
- **Paramétrage** : couleurs, tailles, positions modifiables
- **Interactivité** : réaction aux événements utilisateur
- **Animation** : transformation des propriétés dans le temps

Utilité des images vectorielles

Applications :

- pas du tout adapté à la photographie !
- graphiques géométriques : plans, schémas, logos, polices, icônes (ex: Font Awesome)
- modélisations de données : cartographie, CAO. . .
- c'est la structure interne de documents : pdf, postscript, tkiz ...

Parce que :

- qualité indépendante de la taille d'affichage (aucun crénelage)
- coordonnées des éléments = nombres réels, indépendants de toute notion de pixel
- taille de fichier réduite comparée à une image raster équivalente
- relativement simples à produire par logiciel

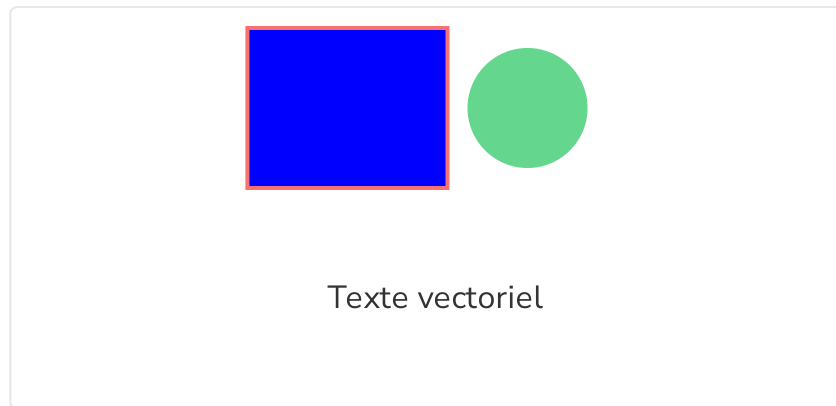
Format SVG (Scalable Vector Graphics)

Caractéristiques techniques

- Standard W3C depuis 1999
- Basé sur XML, donc lisible et éditable
- Intégration native dans HTML5
- Support complet dans les navigateurs modernes
- Accessible aux technologies d'assistance

```
<svg width="200" height="200">
  <!-- Éléments graphiques -->
  <rect x="10" y="10" width="100" height="80"
    fill="blue" stroke="red" stroke-width="2" />
  <circle cx="150" cy="50" r="30"
    fill="green" opacity="0.7" />
  <text x="50" y="150" font-size="16">
    Texte vectoriel
  </text>
</svg>
```

Rendu en temps réel



Largeur rectangle :

Couleur cercle :

Structure générale d'images SVG

Le format SVG est ouvert et assez simple à comprendre. Il est correctement affiché par de nombreux logiciels. Une image SVG est définie par un document XML qui peut soit être dans un fichier à part, soit inclus dans un HTML.

```
<svg version="1.0" xmlns="http://www.w3.org/2000/svg">  
... balises définissant l'image ...  
</svg>
```

Le point important est que toutes les balises filles de `<svg>` doivent être rattachées au namespace SVG. L'attribut `xmlns` fait cela implicitement pour toutes les balises déjà écrites, mais pas pour celles rajoutées par programme. On verra plus loin comment s'en assurer en JavaScript.

Pour l'affichage sur un écran, on ajoute deux attributs `width` et `height` à la balise `<svg>`. Ils définissent la taille d'affichage de l'image en nombre de pixels. Ces attributs définissent les limites de la zone visible, c'est-à-dire les coordonnées utilisables. Ce sont des réels qui vont de 0 à `width` pour x , et de 0 à `height` pour y .

Définition de la zone visible

On peut rajouter l'attribut `viewBox="X Y W H"` pour définir une transformation du système de coordonnées : un décalage d'origine et un changement de dimensions.

```
<svg ... width="640" height="400" viewBox="-10 -5 160 100">
```

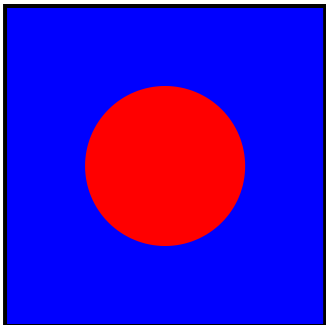
Les valeurs X et Y définissent l'origine des coordonnées de la zone visible, ici en $(-10, -5)$. Les deux nombres suivants, W et H définissent l'étendue de la zone visible à partir de (X, Y) . Dans cet exemple, un point sera visible si son abscisse x est comprise entre -10 et 150 , et son ordonnée y entre -5 et 95 . Donc W et H sont liés à `width` et `height`.

Attention, l'aspect de l'image peut changer si le rapport `width/height` est différent de W/H .

Exemple de viewBox

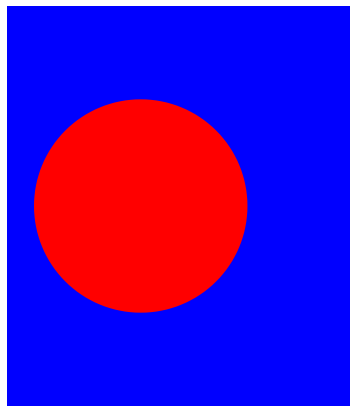
L'attribut `viewBox="X Y W H"` définit un système de coordonnées dans lequel (x, y) est visible si $x \in X, X + W$ et $y \in Y, Y + H$

```
<svg width="200" height="200" viewBox="0 0 100 100">  
  <rect x="10" y="10" width="80" height="80"  
    fill="blue" stroke="black"/>  
  <circle cx="50" cy="50" r="20" fill="red"/>  
</svg>
```



Zone complète visible

```
<svg width="200" height="200" viewBox="25 25 50 50">  
  <rect x="10" y="10" width="80" height="80"  
    fill="blue" stroke="black"/>  
  <circle cx="50" cy="50" r="20" fill="red"/>  
</svg>
```



Zoom sur le centre

Éléments SVG principaux

Rectangle : `<rect>`

```
<rect x="10" y="20" width="100" height="50"
      fill="blue" rx="5" />
```

Cercle : `<circle>`

```
<circle cx="50" cy="50" r="30"
        fill="red" stroke="black" />
```

Ellipse : `<ellipse>`

```
<ellipse cx="50" cy="50" rx="40" ry="20"
         fill="green" />
```

Ligne : `<line>`

```
<line x1="10" y1="10" x2="100" y2="50"
      stroke="blue" stroke-width="3" />
```

Polygone : `<polygon>`

```
<polygon points="10,10 50,30 30,70" fill="orange" />
```

Chemin : `<path>`

```
<path d="M10,20 Q50,10 90,20 T150,30"
      fill="none" stroke="purple" />
```

Texte : `<text>`

```
<text x="20" y="40" font-family="Arial"
      font-size="16" fill="black"> Texte SVG </text>
```

Groupe : `<g>`

```
<g transform="rotate(45)" opacity="0.8">
  <!-- Éléments groupés -->
</g>
```

Démonstration interactive des éléments

Rectangle

Cercle

Ellipse

Ligne

Polygone

Chemin

Texte

Code XML :

```
<rect x="20" y="30" width="120" height="80"  
fill="blue" stroke="darkblue" stroke-width="2"  
rx="10" />
```

Rendu :



Propriétés et attributs SVG

Propriétés de remplissage

`fill` : Couleur de remplissage

- `fill="red"`
- `fill="#FF0000"`
- `fill="rgb(255,0,0)"`
- `fill="none"` (transparent)

`fill-opacity` : Transparence du remplissage de 0 (transparent) à 1 (opaque)

Propriétés de contour

`stroke` : Couleur du contour

`stroke-width` : Épaisseur du trait

`stroke-opacity` : Transparence du contour

`stroke-dasharray` : Trait en pointillés

Transformations

`transform` : Transformations géométriques

- `translate(x,y)` : Translation
- `rotate(angle)` : Rotation
- `scale(sx,sy)` : Redimensionnement
- `skewX(angle)`, `skewY(angle)` : Inclinaison

Autres propriétés importantes

`opacity` : Transparence globale

`class`, `id` : Pour CSS et JavaScript

`style` : Styles CSS inline

Style CSS internes

Il est possible de définir des styles CSS à appliquer aux éléments. Les propriétés sont nommées comme les attributs.

```
<svg version="1.0" xmlns="http://www.w3.org/2000/svg" ... >
<style><![CDATA[
  #arriere {
    fill: #008080;
  }
  .bulle {
    stroke: black;
    stroke-width: 0.2;
    fill: #FF4500;
  }
]]>
</style>
<rect id="arriere" x="0" y="0" width="8" height="6"/>
<circle class="bulle" cx="4" cy="3" r="2"/>
```

Style CSS externes

Les styles CSS peuvent être hors de l'arbre SVG dans un HTML.

```
<!DOCTYPE html>
<html>
<head>
<style>
#arriere { ... }
.bulle { ... }
</style>
</head>
<body>
<svg version="1.0" xmlns="http://www.w3.org/2000/svg" ... >
<rect id="arriere" x="0" y="0" width="8" height="6" />
<circle class="bulle" cx="4" cy="3" r="2" />
</svg>
```

SVG dans HTML5 - Intégration

Méthodes d'intégration

1. Inline SVG (recommandé)

```
<html>
<body>
  <svg width="100" height="100">
    <circle cx="50" cy="50" r="30" fill="red" />
  </svg>
</body>
</html>
```

2. Fichier externe

```

<!-- ou -->
<object data="image.svg" type="image/svg+xml"></object>
```

Avantages du SVG inline

- Manipulation JavaScript directe via selecteur
- Styles CSS applicables
- Événements attachables
- Animations CSS/JS
- Accessibilité complète



SVG intégré avec interactivité JavaScript

Chemins SVG

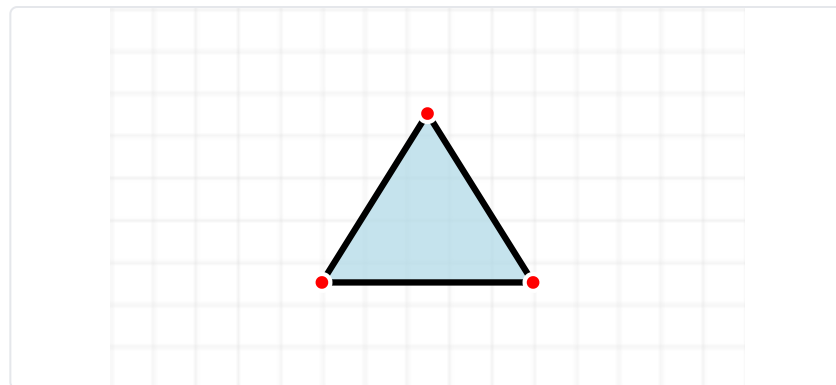
L'élément `<path d="chemin">` permet de dessiner des figures complexes avec lignes droites et courbes

Le chemin SVG utilise l'attribut `d` pour définir une série de commandes de tracé :

```
<path d="M 5 2 L 3 4 L 2 1 Z"  
      stroke="#000" fill="Tan"/>
```

Signification :

- `M 5 2` : Démarrer en (5, 2)
- `L 3 4` : Ligne vers (3, 4)
- `L 2 1` : Ligne vers (2, 1)
- `Z` : Fermer le tracé



Triangle

Carré

Étoile

 Animer

Commandes de chemin SVG

L'attribut `**`d`**` utilise des commandes spécifiques pour tracer lignes et courbes

`M x y` - Move to (déplacer vers) `L x y` - Line to (ligne vers)
`H x` - Horizontal line (ligne horizontale)
`V y` - Vertical line (ligne verticale) `Z` - Closepath (fermer le tracé)

Syntaxe

```
<!-- Rectangle avec commandes basiques -->  
<path d="M 50 30 L 150 30 L 150 80 L 50 80 Z"  
      stroke="black" fill="lightblue"/>  
  
<!-- Équivalent plus concis -->  
<path d="M 50 30 H 150 V 80 H 50 Z"  
      stroke="black" fill="lightblue"/>
```



M L H V Z (Close)

M 50 50

Reset

Chemins absolus et relatifs

Coordonnées absolues

```
<path d="M 5 2 L 3 4 L 2 1 Z"  
      stroke="#000" fill="Gold"/>
```

Points exacts : (5,2) → (3,4) → (2,1) → (5,2)

Coordonnées relatives

```
<path d="M 5 2 l -2 2 l -1 -3 Z"  
      stroke="#000" fill="Gold"/>
```

Déplacements : (5,2) → (-2,+2) → (-1,-3) → retour

Groupes d'objets SVG

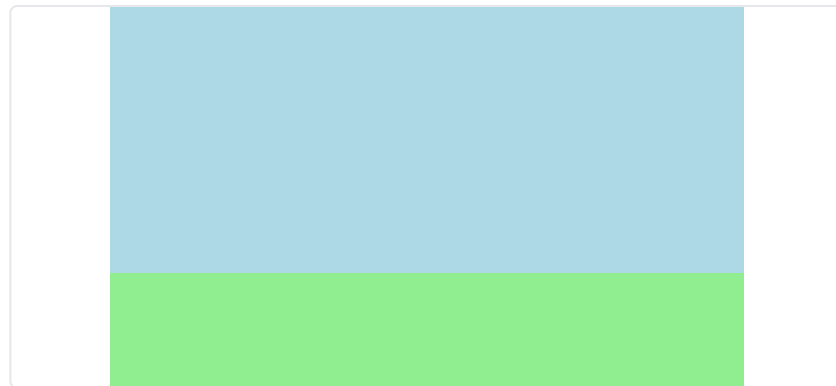
L'élément `<g>` permet de grouper plusieurs éléments pour leur appliquer les mêmes transformations

Structure et avantages

```
<g id="maison" transform="translate(100,50)">
  <rect x="0" y="20" width="60" height="40" fill="brown"/>
  <path d="M 0 20 L 30 0 L 60 20 Z" fill="red"/>
  <rect x="20" y="40" width="10" height="20" fill="darkbrow"
  <rect x="40" y="30" width="8" height="8" fill="lightblue"
</g>
```

Bénéfices :

- Transformations appliquées au groupe entier
- Organisation hiérarchique du code
- Facilite les animations complexes
- Réutilisation possible avec `<use>`



Transformation globale:



Transformations SVG

L'attribut `transform` permet de modifier géométriquement les éléments

Translation

`translate(dx, dy)`



Translate

Reset

Rotation

`rotate(angle)`



Rotate

Reset

Échelle

`scale(k)`



Scale

Reset

Points importants

- `dy` positif fait descendre (axe Y vers le bas)
- Rotations sont dans le sens trigonométrique (anti-horaire)
- Pas de virgule entre `dx` et `dy` : `translate(50 30)` ✓ pas `translate(50, 30)` ✗

Composition des transformations

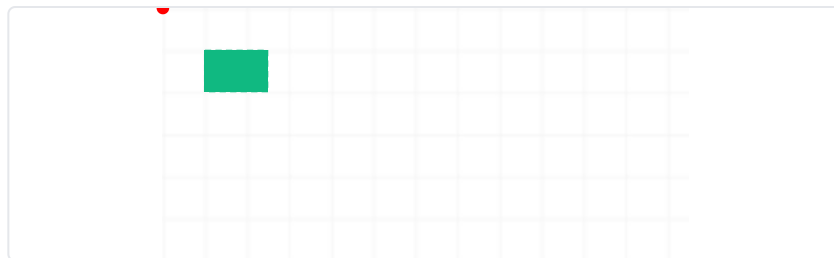
Ordre crucial : Les transformations s'appliquent de droite à gauche (comme en mathématiques)

- 1 D'abord : `rotate(45)`
- 2 Ensuite : `translate(100,50)`



▶ Étape par étape

- 1 D'abord : `translate(100,50)`
- 2 Ensuite : `rotate(45)`



▶ Étape par étape

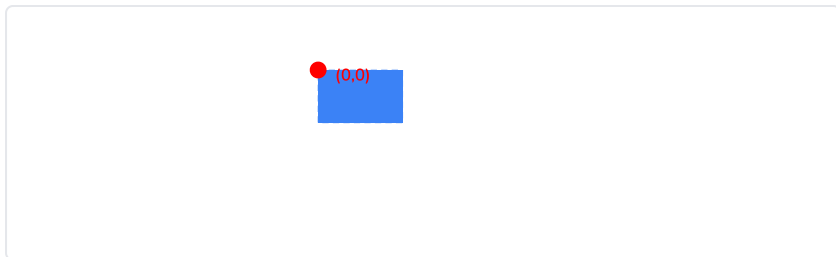
Centre des transformations

- Par défaut : rotation et scale autour de (0,0)
- Modifiable avec `transform-origin`

Rotation par défaut

```
<rect transform="rotate(45)" ... />
```

Centre : (0,0) - coin supérieur gauche du SVG



↺ Rotate 45°

Reset

Centre personnalisé

```
<rect transform-origin="60 45" transform="rotate(45)" />
```

Centre : Point spécifié par transform-origin



↺ Rotate 45°

Reset

Avantages des groupes imbriqués

- Groupes héritent des transformations des parents
- Chaque groupe peut définir son propre centre
- Permet des animations complexes avec hiérarchies
- Réutilisation facilitée avec `<use href="#mon-groupe" />`

Manipulation du DOM avec JavaScript

Le **SVG inline** permet une manipulation directe par JavaScript via le DOM

Créer un élément SVG

```
const svg = document.createElementNS(
  'http://www.w3.org/2000/svg', 'svg'
);
const circle = document.createElementNS(
  'http://www.w3.org/2000/svg', 'circle'
);
```

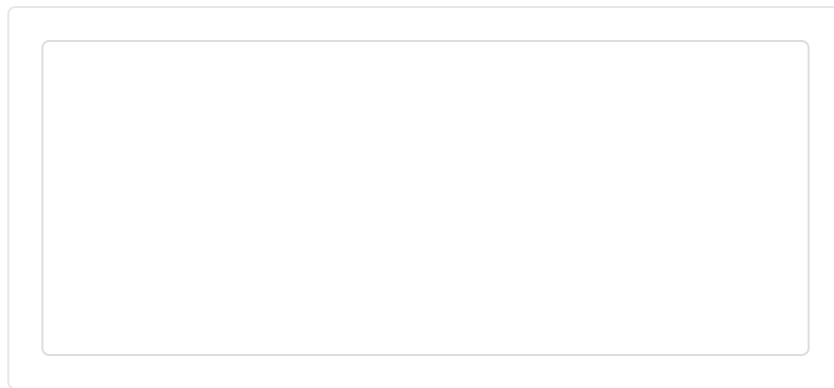
Définir les attributs

```
circle.setAttribute('cx', '50');
circle.setAttribute('cy', '50');
circle.setAttribute('r', '30');
circle.setAttribute('fill', 'red');
```

Ajouter au DOM

```
svg.appendChild(circle);
document.body.appendChild(svg);
```

Exemple interactif



Ajouter cercle

Ajouter rectangle

Effacer tout

Événements et interactivité

SVG supporte tous les événements DOM standard : `click`, `mouseover`, `mousemove`, etc.

```
// Sélectionner un élément SVG existant
const circle = document.querySelector('#mon-cercle');

// Événements de souris
circle.addEventListener('click', function(e) {
  this.setAttribute('fill', 'red');
});

circle.addEventListener('mouseover', function(e) {
  this.setAttribute('r', '35');
});

circle.addEventListener('mouseout', function(e) {
  this.setAttribute('r', '30');
});

// Événement de glisser-déposer
circle.addEventListener('mousedown', startDrag);
circle.addEventListener('mousemove', drag);
circle.addEventListener('mouseup', stopDrag);
```



- Clic : Change la couleur
- Survol : Agrandit l'élément
- Glisser-déposer : Déplace l'élément

Introduction à GSAP

GreenSock Animation Platform - Bibliothèque d'animation JavaScript la plus performante

Pourquoi GSAP ?

- Performance : 60fps même sur mobiles
- Compatibilité : Tous navigateurs (IE9+)
- Simplicité : API intuitive et puissante
- Flexibilité : Anime tout (DOM, SVG, Canvas, etc.)
- Timeline : Séquencement d'animations complexes

```
<!-- CDN -->  
<script src="https://cdnjs.cloudflare.com/ajax/  
libs/gsap/3.12.2/gsap.min.js"></script>
```

```
<!-- ou npm -->  
<script>  
npm install gsap  
</script>
```

```
// Animation simple  
gsap.to("#element", {  
  duration: 2,  
  x: 100,  
  y: 50,  
  rotation: 360,  
  scale: 1.5  
});  
  
// Animation depuis  
gsap.from("#element", {  
  duration: 1,  
  opacity: 0,  
  y: -50  
});  
  
// Animation depuis/vers  
gsap.fromTo("#element",  
  { opacity: 0, scale: 0 },  
  { opacity: 1, scale: 1, duration: 1 }  
);
```

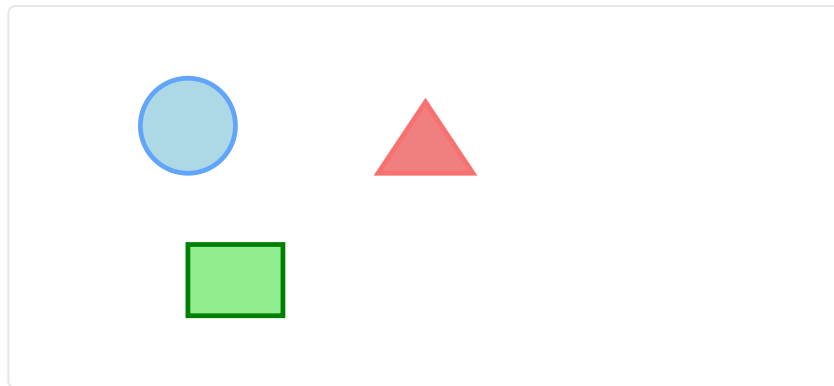
Première animation GSAP

```
// Créer un élément SVG
const svg = document.querySelector('#mon-svg');

// Animation basique
gsap.to("#cercle", {
  duration: 2,
  cx: 200,
  fill: "red",
  ease: "bounce.out"
});

// Avec callbacks
gsap.to("#rectangle", {
  duration: 1.5,
  x: 100,
  rotation: 360,
  onComplete: function() {
    console.log("Animation terminée !");
  }
});

// Timeline pour séquencer
const tl = gsap.timeline();
```



Animation basique

Séquence

Boucle

Reset

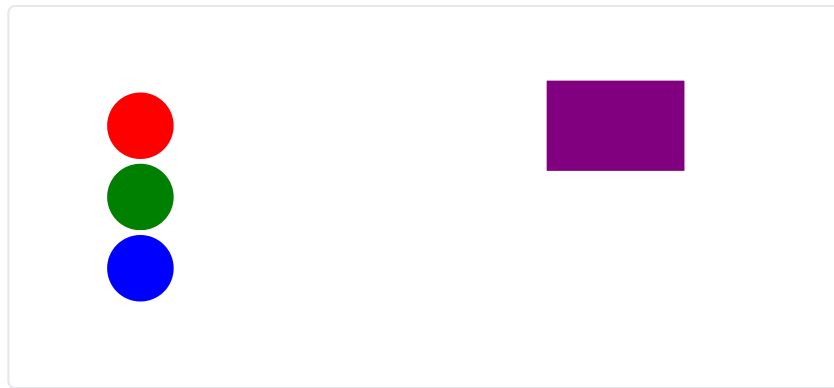
GSAP Timeline - Séquencement avancé

Une Timeline permet de :

- Séquencer plusieurs animations
- Contrôler le timing précisément
- Rejouer, mettre en pause, inverser
- Synchroniser des animations complexes

```
// Créer une timeline
const tl = gsap.timeline();
// Ajouter des animations en séquence
tl.to("#elem1", { duration: 1, x: 100 })
  .to("#elem2", { duration: 0.5, y: 50 })
  .to("#elem3", { duration: 1, rotation: 360 });
// Avec décalages temporels
tl.to("#elem1", { duration: 1, x: 100 })
  .to("#elem2", { duration: 1, y: 50 }, "-=0.5")
  // 0.5s avant la fin
  .to("#elem3", { duration: 1, scale: 2 }, "+=0.2");
  // 0.2s après
```

Démonstration Timeline



Progress:



Propriétés animables avec GSAP

```
gsap.to(element, {  
  // Position  
  x: 100, y: 50, // translate X et/ou Y  
  rotation: 360, // en degrés  
  scale: 1.5,    // proportionnelle  
  scaleX: 2,     // horizontale  
  scaleY: 0.5,   // verticales  
  transformOrigin: "center top"  
});
```

```
gsap.to("circle", {  
  attr: {  
    cx: 200, cy: 100,  
    r: 50,  
    fill: "red"  
  }  
});  
  
gsap.to("path", {  
  attr: {  
    d: "M10,10 L100,100 L200,10"  
  }  
});
```

```
gsap.to(element, {  
  opacity: 0.5,  
  css: {  
    backgroundColor: "red",  
    borderRadius: "10px"  
  }  
});
```

Options d'animation

```
gsap.to(element, {  
  duration: 2,           // durée  
  delay: 0.5,           // délai  
  ease: "bounce.out",   // courbe d'accélération  
  repeat: 3,            // répétitions  
  yoyo: true,           // aller-retour  
  
  // Callbacks  
  onStart: () => console.log("Début"),  
  onComplete: () => console.log("Fin"),  
  onUpdate: () => console.log("Mise à jour")  
});
```

Courbes d'accélération (Easing)

Les ****courbes d'accélération**** contrôlent comment l'animation progresse dans le temps

Types principaux

Power : `power1` , `power2` , `power3` , `power4`

- `.in` : démarre lentement
- `.out` : finit lentement
- `.inOut` : les deux

Bounce : `bounce.out` , `bounce.in` ,
`bounce.inOut`

Elastic : `elastic.out` , `elastic.in` ,
`elastic.inOut`

Back : `back.out(1.7)` , `back.in` , `back.inOut`

Sine : `sine.out` , `sine.in` , `sine.inOut`

