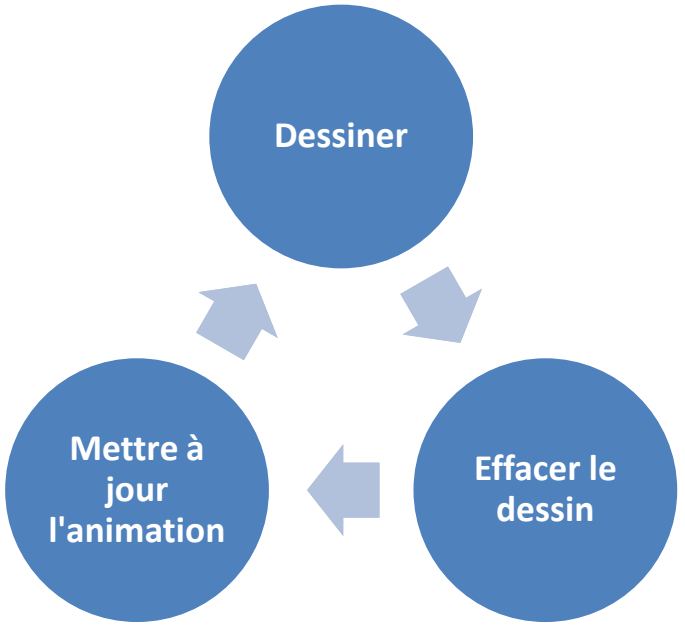
LABORATOIRE 4A: L'ANIMATION

Théorie

1. Le cycle d'animation

Le cycle d'animation est illustré dans l'image ci-contre. Toute animation, peu importe sa nature, respecte ce cycle.



Plus l'intervalle de temps entre chaque cycle est petit, plus la cadence d'animation est rapide; plus l'intervalle de temps entre chaque cycle est grand, plus la cadence d'animation est lente.

Si la cadence d'animation est trop rapide, cela peut produire un effet de superposition. Cela survient lorsque l'intervalle de temps entre chaque cycle est tellement petit que le dessin précédent n'a pas le temps de se terminer lorsque le dessin suivant débute. En d'autres mots, la durée pour dessiner est plus longue que la durée du cycle.

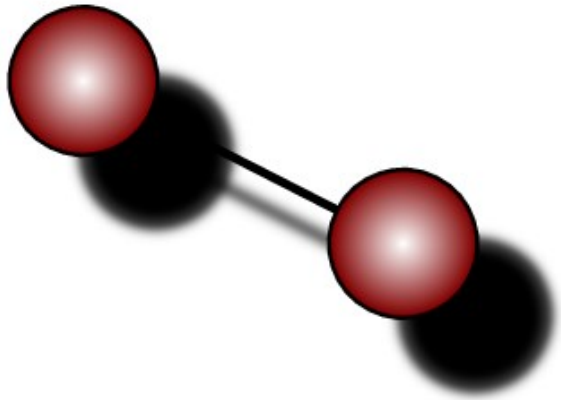
Si la cadence d'animation est trop lente, cela peut produire un effet de saccade; l'animation n'est pas fluide. Cela se produit lorsque l'intervalle de temps entre chaque cycle est tellement grand qu'on n'a pas l'impression qu'il s'agit d'une animation.

Sur un écran d'ordinateur, on estime qu'une bonne cadence d'animation est de 60 cycles par seconde. A 60 cycles par seconde, l'animation est habituellement fluide. Par conséquent, l'intervalle de temps entre chaque cycle doit être à peu près de 17 millièmes de secondes.

2. Une animation à l'aide d'une minuterie

Ouvrez la page Web **Théorie 4A-1.htm** et observez l'animation.

Observez la manière que la fonction **animer** a été programmée. Cette fonction est appelée au chargement de la page Web (dans l'événement **onload**).



```
function animer() {
    objCanvas = document.getElementById('monCanvas');
    objC2D = objCanvas.getContext('2d');
    dessiner(); // Dessiner une première fois
    // Un cycle d'animation
    setTimeout(cycleAnimer, Math.floor(1000 / 60));
}
```

Tout d'abord, nous dessinons l'objet une première fois puis nous appelons la fonction **cycleAnimer** à l'aide de la fonction ***setTimeout***.

La fonction ***setTimeout*** est une minuterie. Elle demande 2 paramètres : le premier paramètre est le nom de la fonction appelée (ici **cycleAnimer**) et le deuxième paramètre est le délai d'appel en millièmes de secondes (ici, 1/60 de seconde).

Cela signifie que, dès que le 1/60 de seconde se sera écoulé, la fonction **cycleAnimer** va être appelée.

Cette fonction est le cœur du programme. Elle représente un cycle d'animation :

- on efface le dessin précédent,
- on met à jour l'animation puis
- on redessine.

A la fin de cette fonction, on appelle de nouveau la fonction **cycleAnimer** pour un autre cycle d'animation qui va durer 1/60 seconde.

```
// Un cycle d'animation
function cycleAnimer() {
    effacerDessin();
    mettreAjourAnimation();
    dessiner();
    setTimeout(cycleAnimer, Math.floor(1000 / 60));
}
```

Le code précédent signifie que la cadence d'animation est de 60 cycles par seconde. Si vous diminuez cette cadence, l'animation va être plus lente et si vous augmentez la cadence, l'animation va être plus rapide. Vous pouvez tester si vous le voulez mais vous pouvez perdre de la fluidité au niveau de l'animation.

Ici, nous utilisons la méthode ***.clearRect*** pour effacer tout ce qui a été dessiné à l'intérieur du canevas.

```
// Pour effacer le dessin
function effacerDessin() {
    objC2D.clearRect(0,0, objCanvas.width, objCanvas.height);
}
```

Pour faire tourner notre objet, nous avons déclaré la variable globale **fltAngleRotation**. Au point de départ, elle vaut 0.

```
var fltAngleRotation = 0;
```

A chaque cycle d'animation, nous l'augmentons de 2.25 degrés (ou $\pi/80$ radians). Cela signifie, qu'à chaque cycle d'animation, l'objet tourne de 2.25 degrés.

```
// Pour mettre à jour l'angle de rotation
function mettreAjourAnimation(){
    // Rotation par coup de 2.25 degrés
    fltAngleRotation += Math.PI/80;
}
```

Il ne nous reste qu'à dessiner. Tout d'abord, nous sauvegardons le contexte (car la fonction ***dessiner*** modifie le contexte) et nous le restaurons à la fin.

```
// Pour dessiner
function dessiner() {
    // Sauvegarder le contexte
    objC2D.save();

    // Restaurer le contexte
    objC2D.restore();
}
```

Avant de dessiner, nous transformons le contexte. Nous déplaçons le contexte au centre du canevas (pour que l'objet qu'on veut dessiner soit centré) puis nous tournons le contexte en utilisant l'angle de rotation de l'objet qu'on va dessiner.

```
// Déplacer le contexte au centre du canevas
objC2D.translate(objCanvas.width/2, objCanvas.height/2);

// Faire tourner le contexte
objC2D.rotate(fltAngleRotation);
```

Finalement, nous dessinons l'objet (deux cercles reliés par une ligne). Tentez de comprendre le dessin.

```
// Dessiner la forme qui tourne
objC2D.strokeStyle = 'black';
objC2D.lineWidth = 5;
objC2D.beginPath();
objC2D.moveTo(-intXCercle,0);
objC2D.lineTo(intXCercle,0);
objC2D.closePath();
objC2D.stroke();

var objDegrade=objC2D.createRadialGradient(intXCercle,0,0,intXCercle,0,intRayonCercle);
objDegrade.addColorStop(0,'rgb(255,255,255)');
objDegrade.addColorStop(1,'rgb(128,0,0)');
objC2D.fillStyle=objDegrade;
objC2D.beginPath();
objC2D.arc(intXCercle, 0, intRayonCercle, 0, 2 * Math.PI, false);
objC2D.stroke();
objC2D.fill();

var objDegrade2 = objC2D.createRadialGradient(-intXCercle, 0, 0, -intXCercle, 0, intRayonCercle);
objDegrade2.addColorStop(0,'rgb(255,255,255)');
objDegrade2.addColorStop(1,'rgb(128,0,0)');
objC2D.fillStyle=objDegrade2;
objC2D.beginPath();
objC2D.arc(-intXCercle, 0, intRayonCercle, 0, 2 * Math.PI, false);
objC2D.stroke();
objC2D.fill();
```

Notez qu'il y a un effet d'ombrage dans le dessin.

```
// L'ombrage
objC2D.shadowColor = 'black';
objC2D.shadowBlur = 10;
objC2D.shadowOffsetX = intDecalOmbre;
objC2D.shadowOffsetY = intDecalOmbre;
```

2. Une animation à l'aide d'une minuterie répétitive

Ouvrez la page Web **Théorie 4A-2.htm** et observez l'animation.

Observez la manière que les fonctions **animer** et **cycleAnimer** ont été programmés.

Ici, pour programmer, le cycle d'animation, nous utilisons la fonction ***setInterval*** au lieu de la fonction ***setTimeOut***.

La fonction ***setInterval*** est une minuterie répétitive. Elle demande 2 paramètres : le premier paramètre est le nom de la fonction appelée de manière répétitive (ici **cycleAnimer**) et le deuxième paramètre est le délai d'appel en millièmes de secondes (ici, 1/60 de seconde).

Cela signifie que la fonction **cycleAnimer** va être appelée à répétition (et à l'infini) à chaque 1/60 de seconde (donc 60 fois par seconde).

Le reste du programme n'a pas été modifié.

```
function animer() {
    objCanvas = document.getElementById('monCanvas');
    objC2D = objCanvas.getContext('2d');
    dessiner(); // Dessiner une première fois
    // Un cycle d'animation à répétition
    setInterval(cycleAnimer, Math.floor(1000 / 60));
}

// Un cycle d'animation
function cycleAnimer() {
    effacerDessin();
    mettreAjourAnimation();
    dessiner();
}
```

3. Une animation à l'aide d'une requête

L'utilisation d'une minuterie répétitive (la fonction ***setInterval***) pour réaliser une animation est correcte dans la majorité des cas mais parfois elle peut s'avérer inadéquate et non performante. Voici les principaux inconvénients reliés à cette minuterie.

- Premièrement, avec la minuterie, c'est le programmeur qui doit déterminer la durée du cycle d'animation. Habituellement, pour obtenir une animation fluide, cette durée est de 1/60 seconde. Mais, en réalité, cela dépend de l'écran. Le taux de rafraichissement de l'écran peut varier d'un écran à un autre. Si le taux de rafraîchissement de l'écran n'est pas synchronisé avec la durée du cycle d'animation, cela peut produire une animation non fluide.

- Ensuite, cette minuterie fonctionne en permanence même lorsque la page Web n'est pas visible sur l'écran (par exemple, lorsque la fenêtre du navigateur est réduite). L'animation va continuer de jouer à l'intérieur d'un page Web que l'utilisateur ne voit pas. La conséquence, c'est que cela demande du temps CPU inutilement. Si l'animation s'exécute sur un mobile, la batterie va s'épuiser plus rapidement.

- Finalement, la minuterie répétitive est programmée de manière à ce que chaque cycle d'animation soit placé dans une file d'attente. Le cycle s'exécute lorsque son temps est arrivé. Cela ne cause habituellement aucun problème. Le problème se produit lorsque la durée du dessin est plus longue que la durée allouée au cycle d'animation (par exemple, la durée du dessin est de 1/30 de seconde et la durée du cycle d'animation est 1/60 seconde). Cela fait en sorte que plusieurs cycles d'animation sont bloqués en attente d'exécution. S'il y a trop de cycles d'animation non exécutés dans la file d'attente, le navigateur peut caler (comme un moteur qui cale lorsqu'il y a trop d'essence) et cela peut faire « laguer » et même « bloquer » le navigateur.

Dans le but de régler ces problèmes, en HTML5, la fonction ***requestAnimationFrame*** a été définie. Cette fonction est dédiée exclusivement aux animations. Voici les avantages.

- Premièrement, la durée du cycle d'animation est déterminée automatiquement par cette fonction. Par défaut, elle est de 1/60 seconde mais peut varier légèrement d'un écran à l'autre pour se synchroniser sur le taux de rafraîchissement de l'écran.

- Ensuite, le cycle d'animation s'exécute seulement lorsque la page Web est visible. Non seulement cela a l'avantage d'économiser du temps CPU, mais cela a également l'avantage que l'animation se met sur pause lorsque la page Web est masqué et reprend au même endroit lorsque la page Web redevient visible.

- Finalement, il n'y a aucune file pour mettre les cycles d'animation en attente. Les cycles d'animation ne s'exécutent pas de manière stupide. Chaque cycle d'animation s'exécute seulement si le navigateur est prêt à l'exécuter. Par exemple, si la page Web est masquée, le navigateur ne va pas exécuter le cycle d'animation. Étant donné qu'il n'y a pas de file d'attente, cela fait en sorte qu'il est impossible que le navigateur cale.

Ouvrez la page Web **Théorie 4A-3.htm** et observez l'animation.

Observez la manière que les fonctions **animer** et **cycleAnimer** ont été programmés.

Tout d'abord, nous faisons appel à la fonction **cycleAnimer**.

Au point de départ, la fonction ***requestAnimationFrame*** est appelée en lui passant en paramètre le nom de la fonction dans laquelle le cycle d'animation a été implanté (ici, **cycleAnimer**).

En fait, cette fonction est une requête pour le prochain cycle. On demande au navigateur d'exécuter la fonction **cycleAnimer** si possible.

```
function animer() {
    objCanvas = document.getElementById('monCanvas');
    objC2D = objCanvas.getContext('2d');
    dessiner(); // Dessiner une première fois
    // Un cycle d'animation à répétition
    cycleAnimer();
}

// Un cycle d'animation
function cycleAnimer() {
    requestAnimationFrame(cycleAnimer);
    effacerDessin();
    mettreAjourAnimation();
    dessiner();
}
```

Il est possible que la fonction **cycleAnimer** ne soit pas exécutée. Par exemple, si la page Web est masquée, la fonction **cycleAnimer** ne sera pas exécutée. Elle sera exécutée seulement lorsque la page Web deviendra visible.