2- LES DESSINS 2D – PARTIE 2

Présenté par Ronald Jean-Julien

420-P46: Programmation 3D

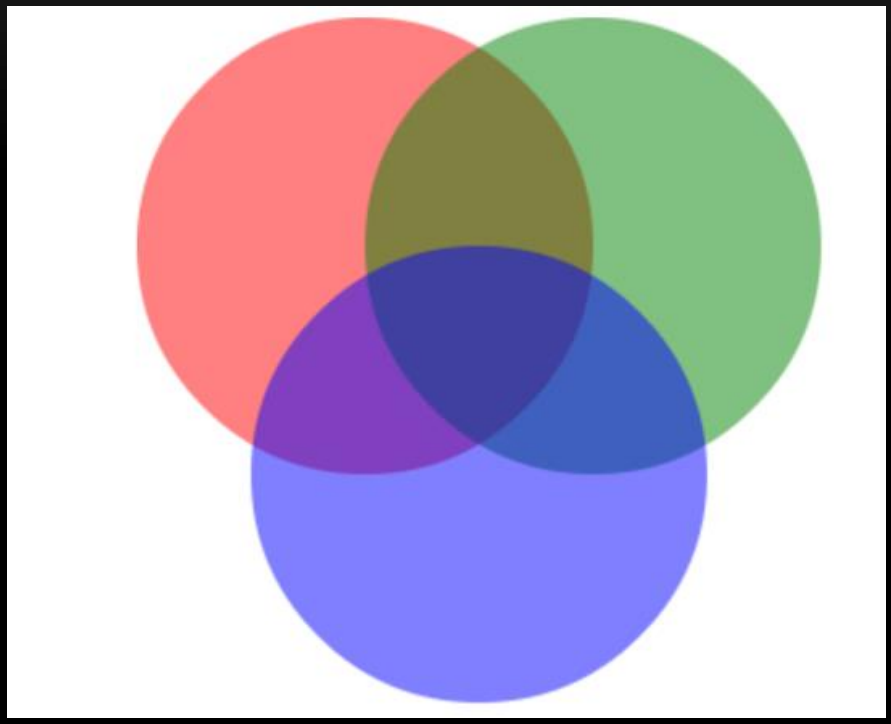
Hiver 2020

LA TRANSPARENCE

- Pour définir la transparence, on doit définir l'opacité (nommé composante ***alpha***). Cette valeur doit être en %. Le nombre 1 signifie que c'est complètement opaque. Le nombre 0 signifie que c'est complètement transparent. Le nombre 0.25 signifie que c'est 25% opaque (donc 75% transparent).
- .globalAlpha = L'opacité en pourcentage (par défaut 1)
- Lorsqu'on définit l'opacité tout ce qu'on va dessiner par la suite va revêtir cette opacité mais on peut modifier l'opacité au fur et à mesure que l'on dessine.

EXEMPLE DE LA TRANSPARENCE

```
// A moitié transparent
objC2D.globalAlpha = 0.5;

// Cercle plein rouge
objC2D.fillStyle = 'red';
objC2D.beginPath();
objC2D.arc(100,100,100,0,2*Math.PI);
objC2D.fill();

// Cercle plein vert
objC2D.fillStyle = 'green';
objC2D.beginPath();
objC2D.arc(200,100,100,0,2*Math.PI);
objC2D.fill();

// Cercle plein bleu
objC2D.fillStyle = 'blue';
objC2D.beginPath();
objC2D.arc(150,200,100,0,2*Math.PI);
objC2D.fill();
```

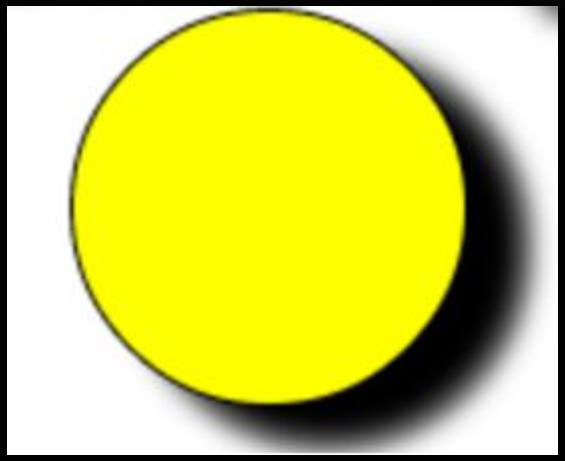


L'OMBRAGE

- Il y a 4 propriétés qui concernent l'ombrage.
  - .shadowColor = La couleur de l'ombre (par défaut noir)
  - .shadowBlur = La quantité de flou dans l'ombre (par défaut 0 = no blur)
  - .shadowOffsetX = Le décalage horizontal de l'ombre par rapport au dessin (par défaut 0)
  - .shadowOffsetY = Le décalage vertical de l'ombre par rapport au dessin (par défaut 0)
- Lorsqu'on définit l'ombrage tout ce qu'on va dessiner par la suite va revêtir cet ombrage mais on peut modifier l'ombrage au fur et à mesure que l'on dessine.

EXEMPLE DE L'OMBRAGE

```
// Les paramètres de l'ombrage
objC2D.shadowColor = 'black';
objC2D.shadowBlur = 10;
objC2D.shadowOffsetX = 15; // Vers la droite
objC2D.shadowOffsetY = 10; // Vers le bas

// Largeur du contour et couleurs
objC2D.lineWidth = 2;
objC2D.fillStyle = 'yellow';
objC2D.strokeStyle = 'black';

// Le dessin du cercle
objC2D.beginPath();
objC2D.arc(90,120,50,0,2*Math.PI,false);
objC2D.stroke();
objC2D.fill();
```



LES COMPOSITES (1)

- En 2D, un composite est la fusion de 2 dessins à l'aide d'une règle de composition.
- Par exemple, le remplissage d'un cercle à l'aide d'un motif est un composite. La transparence et l'ombrage sont également des composites.
- Pour dessiner un composite, on procède de la manière suivante :
  - On dessine un premier dessin. Ce dessin porte le nom de dessin cible ou dessin de destination.
  - On applique la règle de composition :
    .globalCompositeOperation = 'La règle de composition'
  - On dessine un autre dessin. Ce dessin porte le nom de dessin source.

LES COMPOSITES (2)

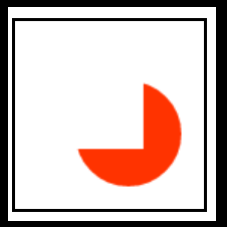
- En 2D, il existe 12 règles de composition différentes. En voici quelques unes.
- Dans chacun des dessins suivants, le dessin cible est le carré plein bleu et le dessin source est le cercle plein rouge. Le carré bleu a été dessiné AVANT le cercle rouge.
- La règle de composition par défaut est source-over.
  c'est-à-dire que le dessin source se dessine par-dessus le dessin cible.
  Ici, on dessine un carré bleu puis on dessine un cercle rouge.
  Le cercle rouge écrase le carré bleu.

- Ici, la règle de composition est source-atop.
  Le cercle rouge se dessine par dessus le carré bleu
  mais seulement la partie du cercle située à l'intérieur du carré bleu.

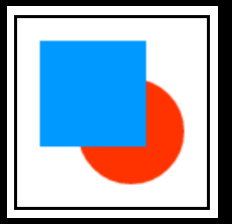
LES COMPOSITES (3)

- Ici, la règle de composition est source-in.
  Le cercle rouge se dessine à l'intérieur du carré bleu
  mais on ne voit pas la couleur du carré bleu (on voit seulement sa forme).

- Ici, la règle de composition est source-out.
  Le cercle rouge se dessine à l'extérieur du carré bleu
  mais on ne voit pas la couleur du carré bleu (on voit seulement sa forme).

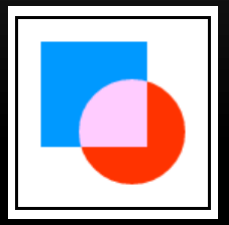


- Ici, la règle de composition est destination-over.
  Le cercle rouge se dessine en dessous du carré bleu.
  C'est comme si on avait dessiné le cercle rouge puis le carré bleu.



LES COMPOSITES (4)

- Ici, la règle de composition est lighter.
  On voit les deux dessins mais, à l'intersection,
  les deux couleurs sont additionnées entre elles.



- Ici, la règle de composition est darker.
  On voit les deux dessins mais, à l'intersection,
  les deux couleurs sont multipliées entre elles.

- Ici, la règle de composition est xor.
  On voit les deux dessins mais, à l'intersection,
  les deux couleurs ont subi une opération de type **xor**.

LES COMPOSITES (5)

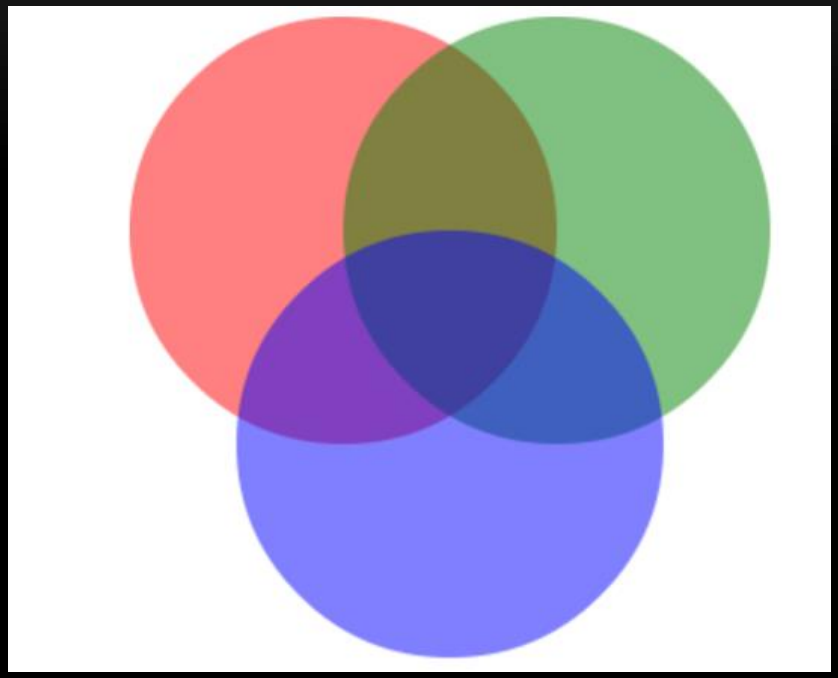
- Il faut faire très attention lorsqu'on travaille avec les composites. Ce sont des outils puissants mais parfois cela donne des effets étranges et imprévus.
- Par exemple :
  Dessin D1
  Règle de composition R1
  Dessin D2
  Dessin D3
- Ici la règle de composition R1 va s'appliquer aux dessins D2 et D1. Cela donne le composite suivant: R1(D2,D1). Puis la règle de composition R1 va s'appliquer au dessin D3 et au composite R1(D2,D1). Le composite final est R1(D3, R1(D2,D1)).
- Parfois, pour obtenir l'effet que l'on désire, on doit mettre une règle de composition entre chaque dessin.
- Dessin D1
  Règle de composition R1 – Elle s'applique au dessin D2 sur le dessin D1
  Dessin D2
  Règle de composition R2 – Elle s'applique au dessin D3 sur le composite R1(D2, D1)
  Dessin D3

LA ZONE DE DÉCOUPAGE

- En 2D, la zone de découpage est la partie visible du dessin. Par défaut, la zone de découpage, c'est le canevas au complet; c'est-à-dire que tout ce qui est dessiné à l'intérieur du canevas est visible et tout ce qui est dessiné à l'extérieur du canevas est masqué.
- Pour définir une zone de découpage, on définit un tracé puis on applique la méthode .clip() à ce tracé. Par la suite, tout ce qui va être situé à l'intérieur du tracé va être visible et tout ce qui va être dessiné à l'extérieur du tracé va être masqué.
- **Attention**: On ne doit pas dessiner le tracé. On doit seulement le définir.

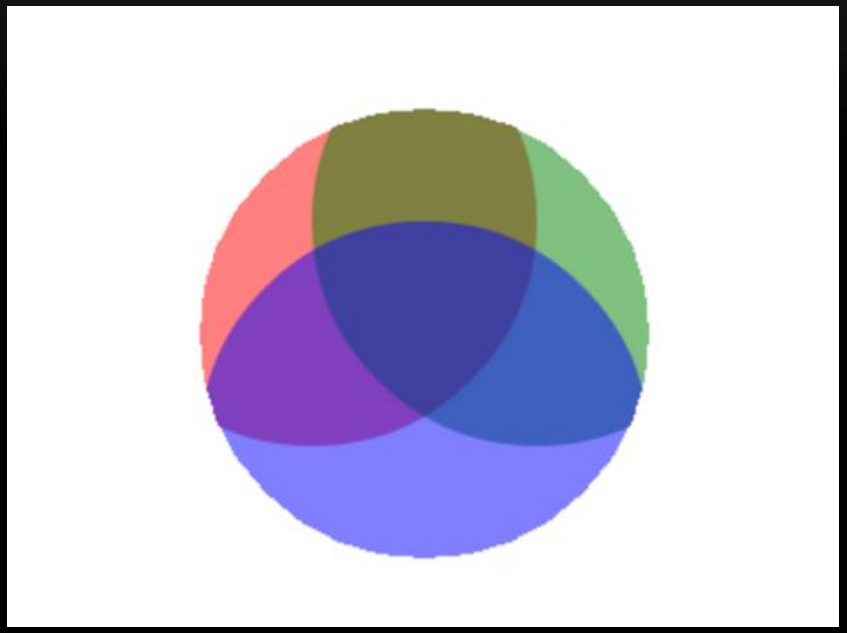
EXEMPLE SANS ZONE DE DÉCOUPAGE

```
// A moitié transparent
objC2D.globalAlpha = 0.5;

// Cercle plein rouge
objC2D.fillStyle = 'red';
objC2D.beginPath();
objC2D.arc(100,100,100,0,2*Math.PI);
objC2D.fill();

// Cercle plein vert
objC2D.fillStyle = 'green';
objC2D.beginPath();
objC2D.arc(200,100,100,0,2*Math.PI);
objC2D.fill();

// Cercle plein bleu
objC2D.fillStyle = 'blue';
objC2D.beginPath();
objC2D.arc(150,200,100,0,2*Math.PI);
objC2D.fill();
```



EXEMPLE AVEC ZONE DE DÉCOUPAGE

```
// Zone de découpage circulaire
objC2D.beginPath();
objC2D.arc(150,150,100,0, 2*Math.PI);
objC2D.clip(); // Pour découper

// A moitié transparent
objC2D.globalAlpha = 0.5;

// Cercle plein rouge
objC2D.fillStyle = 'red';
objC2D.beginPath();
objC2D.arc(100,100,100,0,2*Math.PI);
objC2D.fill();

// Cercle plein vert
objC2D.fillStyle = 'green';
objC2D.beginPath();
objC2D.arc(200,100,100,0,2*Math.PI);
objC2D.fill();

// Cercle plein bleu
objC2D.fillStyle = 'blue';
objC2D.beginPath();
objC2D.arc(150,200,100,0,2*Math.PI);
objC2D.fill();
```

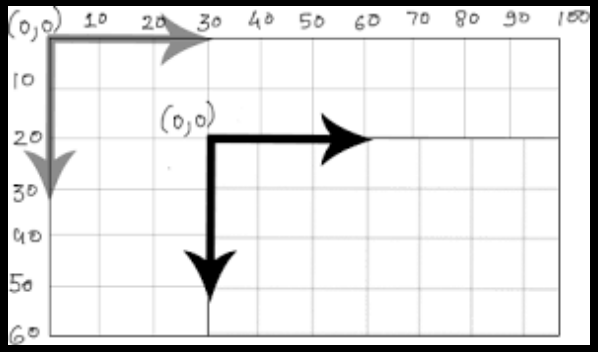


LES TRANSFORMATIONS DU CONTEXTE

- Les transformations du contexte sont des outils très puissants lorsqu'on veut ajouter des effets particuliers sur notre dessin.
- Il existe plusieurs types de transformation du contexte.
- Les trois (3) principales transformations du contexte sont:
  - La translation du contexte
  - La rotation du contexte
  - La mise à l'échelle du contexte
- En 2D, il est possible de définir et de personnaliser d'autres types de transformations du contexte tel que le cisaillement du contexte (voir théorie).
- **ATTENTION**: La transformation du contexte doit s'appliquer AVANT de dessiner. Jamais après.

LA TRANSLATION DU CONTEXTE

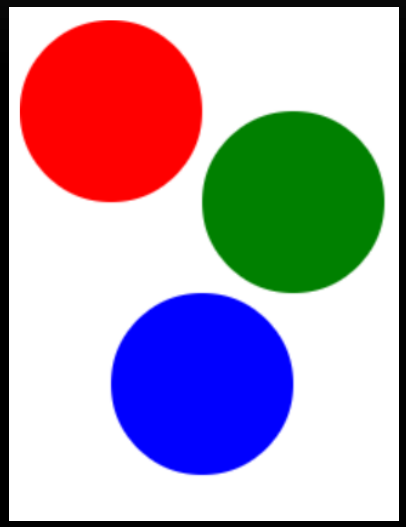
- Par défaut, le point (0,0) du contexte est situé dans le coin supérieur gauche du canevas.
- .translate(intDeplx, intDeply)
  Translater le contexte signifie déplacer le contexte ailleurs à l'intérieur du canevas. Lorsqu'on translate le contexte, le plan cartésien change de place à l'intérieur du canevas.



- Le principe est le suivant. On déplace le contexte et on dessine en ayant en tête que le contexte a changé de place à l'intérieur du canevas.

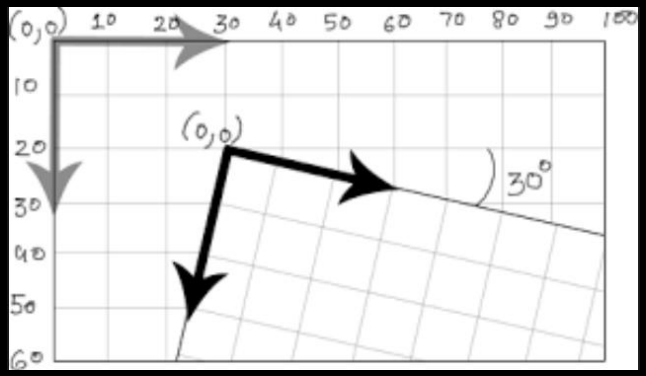
EXEMPLE DE TRANSLATION DU CONTEXTE

```
// Cercle plein rouge
objC2D.fillStyle = 'red';
// Cercle rouge dessiné à la position (50,50)
objC2D.beginPath();
objC2D.arc(50,50,50,0,2*Math.PI);
objC2D.fill();

objC2D.translate(100,50); // Déplacer le contexte de (100,50)
// Cercle plein vert
objC2D.fillStyle = 'green';
// Cercle vert dessiné à la position (50,50)
objC2D.beginPath();
objC2D.arc(50,50,50,0,2*Math.PI);
objC2D.fill();

objC2D.translate(-50,100); // Déplacer le contexte de (-50,100)
// Cercle plein bleu
objC2D.fillStyle = 'blue';
// Cercle bleu dessiné à la position (50,50)
objC2D.beginPath();
objC2D.arc(50,50,50,0,2*Math.PI);
objC2D.fill();
```



LA ROTATION DU CONTEXTE

- .rotate(fltAngleRadian)
  Il est possible de faire tourner le contexte à l'intérieur du canevas. Le contexte tourne toujours autour du point (0,0) et dans le sens horaire. C'est la raison pour laquelle, souvent, on translate le contexte puis, par la suite, on le fait tourner.



- Le principe est le suivant. On fait tourner le contexte et on dessine en ayant en tête que le contexte a tourné à l'intérieur du canevas.

EXEMPLES DE ROTATION DU CONTEXTE

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

objC2D.fillStyle = 'purple';
// 18 rotations du carré
for (let i = 0; i < 18; i++) {
      objC2D.rotate(Math.PI/9); // Rotation du contexte de 20 degrés
      objC2D.fillRect(50,0, 10,10);
}
```

```
// Tourner le contexte de 90 degrés
objC2D.rotate(Math.PI/2);

// Écrire le texte
let strTexte = 'Écrit à 90 degrés';
objC2D.fillStyle = 'magenta';
objC2D.font = '30pt Verdana';
objC2D.textBaseline = 'top';
objC2D.fillText(strTexte,0,0);
```

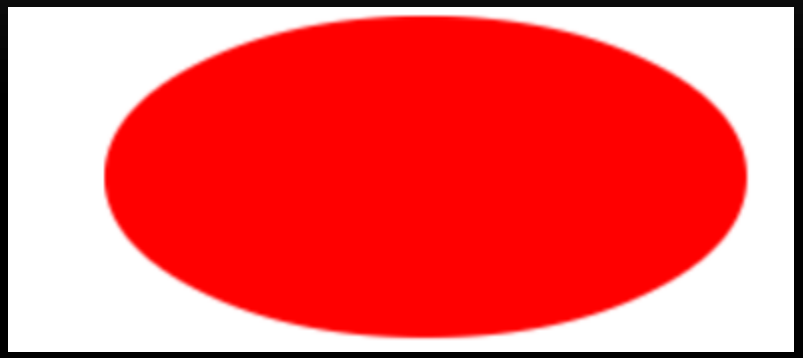
Écrit à 90 degrés

LA MISE À L'ÉCHELLE DU CONTEXTE

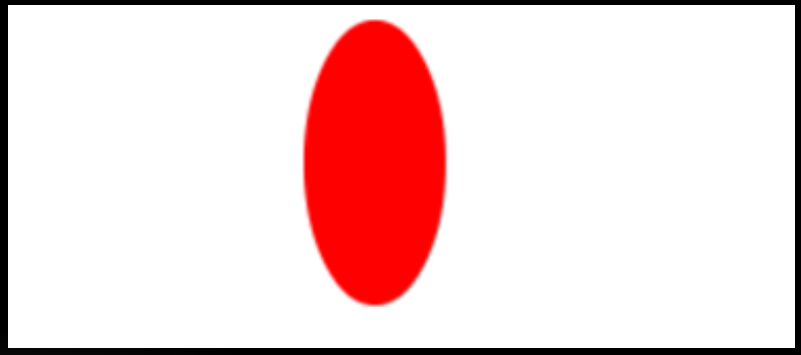
- .scale(fltRapportX, fltRapportY)
  Il est possible de mettre le contexte à l'échelle avec un certain rapport. Cela signifie étirer ou contracter le contexte (à la manière d'un élastique).
- Le contexte se met toujours à l'échelle par rapport au point (0,0).
- Un rapport inférieur à 1 (sans atteindre le 0) va contracter le contexte et un rapport supérieur à 1 va étirer le contexte. Par exemple, un rapport de 1/2 va contracter le contexte du double par rapport à sa taille originale et un rapport de 2/1 va étirer le contexte du double par rapport à sa taille originale.
- Un rapport égal à 1 ne met pas la taille du contexte à l'échelle. Le contexte conserve sa taille originale.
- Un rapport égal à 0 enlève complètement la dimension. En fait, le contexte est tellement contracté qu'il n'a plus aucune dimension.
- Un rapport négatif donne un effet miroir ou un effet de retournement (un *flip*).

EXEMPLES DE MISE À L'ÉCHELLE DU CONTEXTE

```
// Étirer le contexte dans le sens horizontal
objC2D.scale(2,1);
// Cercle plein rouge
objC2D.fillStyle = 'red';
objC2D.beginPath();
objC2D.arc(50,50,50,0,2*Math.PI);
objC2D.fill();
```



```
// Contracter le contexte dans le sens horizontal
objC2D.scale(1/2,1);
// Cercle plein rouge
objC2D.fillStyle = 'red';
objC2D.beginPath();
objC2D.arc(50,50,50,0,2*Math.PI);
objC2D.fill();
```



EXEMPLES D'EFFET MIROIR (OU DE RETOURNEMENT OU *FLIP*)

```
// Effet miroir horizontal
objC2D.translate(objCanvas.width,0);
objC2D.scale(-1,1);

// Écrire le texte
const strTexte = 'Effet miroir horizontal';
objC2D.fillStyle = 'Maroon';
objC2D.font = '30px Verdana';
objC2D.textBaseline='top';
objC2D.textAlign='left';
objC2D.fillText(strTexte,0,0);
```


```
// Effet miroir vertical
 objC2D.translate(0, 30);
objC2D.scale(1,-1);

// Écrire le texte
const strTexte = 'Effet miroir vertical';
objC2D.fillStyle = 'Maroon';
objC2D.font = '30px Verdana';
objC2D.textBaseline='top';
objC2D.textAlign='left';
objC2D.fillText(strTexte,0,0);
```

L'ÉTAT DU CONTEXTE

- Le contexte 2D est une machine à états. Dès que l'état du contexte est modifié, il l'est jusqu'à la fin du programme.
- Par exemple, si vous modifiez le style de remplissage, ce style sera modifié jusqu'à la fin du programme. Si vous faites tourner le contexte, celui-ci sera retourné jusqu'à la fin du programme.
- En 2D, il est possible de sauvegarder l'état du contexte à un moment précis à l'aide de .save() puis de le restaurer par la suite à l'aide de .restore().
- Habituellement, pour programmer proprement, au début de chaque fonction, on sauvegarde le contexte puis on restaure l'ancien contexte à la fin de la fonction. De cette manière, chaque fonction travaille dans son propre contexte.

```
function dessinerOvale(objC2D) {
        objC2D.save() // Sauvegarder le contexte actuel au début
        // On dessine la forme ovale
        // Cette fonction modifie l'état du contexte: le contexte est mis à l'échelle
        objC2D.restore() // Restaurer l'ancien contexte à la fin
}
```

L'ACCÈS DIRECT AUX PIXELS (1)

- De manière interne, un pixel est un quadruplet d'octets (Rouge, Vert, Bleu, Alpha). C'est la combinaison de ces quatre octets qui donnent la vraie couleur. Chaque nombre varie de 0 à 255.
- A l'exception de la composante **Alpha**, plus le nombre est élevé, plus la couleur est claire et plus le nombre est bas, plus la couleur est sombre.
- Exemples:
  (0,0,0,255) : Noir opaque
  (255,255,255,255) : Blanc opaque
  (128,128,128,128) : Gris moyen à moitié opaque
  (255,0,0,255) : Rouge très clair opaque
  (0,0,64,128) : Bleu foncé à moitié opaque
  (128,128,0,255) : Jaune moyen (mélange de rouge et de vert) opaque

L'ACCÈS DIRECT AUX PIXELS (2)

- Un canevas est un conteneur de pixels (**pixels map**). En 2D, il est possible d'avoir un accès direct à ce conteneur. Cela est très pratique si on veut travailler directement sur le dessin.
- *.createImageData(intLargeur,intHauteur) :*

  Pour créer un nouveau conteneur de pixels vide de taille (*intLargeur, intHauteur).*
- *.getImageData(intX,intY, intLargeur,intHauteur)*

  Pour extraire du canevas le conteneur de pixels de taille (*intLargeur, intHauteur)* et ce, à partir de la position *(int,intY).*
- *.putImageData(conteneurPixels, intX,intY)*

  Pour écrire **directement** le conteneur de pixels à l'intérieur du canevas et ce, à partir de la position *(intX,intY).*

L'ACCÈS DIRECT AUX PIXELS (3)

- Un conteneur de pixels est un tableau d'octets (*.data*) de taille **intLargeur * intHauteur * 4**

  L'octet #0 représente la composante rouge du pixel situé sur la ligne 0, colonne 0.

  L'octet #1 représente la composante verte du pixel situé sur la ligne 0, colonne 0.

  L'octet #2 représente la composante bleue du pixel situé sur la ligne 0, colonne 0.

  L'octet #3 représente la composante alpha du pixel situé sur la ligne 0, colonne 0.

  L'octet #4 représente la composante rouge du pixel situé sur la ligne 0, colonne 1.

  L'octet #5 représente la composante verte du pixel situé sur la ligne 0, colonne 1.

  …

- Par conséquent, pour parcourir l'ensemble des pixels, il faut parcourir le tableau d'octets de 4 en 4.

UN EXEMPLE D'ACCÈS DIRECT AUX PIXELS (1)

```
// Créer un conteneur de pixels de taille 200 X 100
// Ce conteneur contient 20000 pixels (200*100)
// Le tableau contient 80000 octets (20000*4)
const objPixels = objC2D.createImageData(200,100);

for (let i=0; i < objPixels.data.length; i +=4)
{ // Parcourir le tableau d'octets de 4 en 4
  // Composante rouge au hasard entre 0 et 255
  objPixels.data[i] = Math.floor(Math.random() * 256);
  // Composante verte au hasard entre 0 et 255
  objPixels.data[i+1] = Math.floor(Math.random() * 256);
  // Composante bleue au hasard entre 0 et 255
  objPixels.data[i+2] = Math.floor(Math.random() * 256);
 // Composante alpha est égale à 255 (opaque)
  objPixels.data[i+3] = 255;
}

// Affecte ces pixels au canevas à la position (10, 10)
objC2D.putImageData(objPixels, 10,10);
```

UN EXEMPLE D'ACCÈS DIRECT AUX PIXELS (2)

UN AUTRE EXEMPLE D'ACCÈS DIRECT AUX PIXELS (1)

```
const objImage = new Image();   // Créer l'image
objImage.src = 'Superman.jpg';  // Le fichier
objImage.onload = function() {
    // Dessiner l'image originale
    objC2D.drawImage(objImage, 10, 10);

    // Aller chercher les pixels de l'image dessinée sur le canevas
    const objPixels = objC2D.getImageData(10,10,objImage.width, objImage.height);

    // Parcourir le tableau d'octets et faire une moyenne des couleurs
    for (let i=0; i < objPixels.data.length; i +=4) {
      const enGris = Math.floor((objPixels.data[i] +
            objPixels.data[i+1] + objPixels.data[i+2]) / 3);
       // Changer la couleur des pixels pour une teinte de gris
       for (let j=0; j < 3; j++)  objPixels.data[i+j] = enGris;
    }

   // Redessiner dans le canevas l'image transformée
   objC2D.putImageData(objPixels,10, 20 + objImage.height);

    objC2D.putImageData(objPixels,10, 40 + 2 * objImage.height);
}
```

UN AUTRE EXEMPLE D'ACCÈS DIRECT AUX PIXELS (2)