|  |
| --- |
| Laboratoire 2: les dessins 2D – partie 2 |

Théorie

1. La transparence

En 2D, il est possible d’ajouter de la transparence. Pour ce faire, il faut utiliser la propriété ***.globalAlpha***. La valeur 1 signifie complètement opaque et la valeur 0 signifie complètement transparent. Par défaut, il n’y a pas de transparence.

Voici le dessin de trois cercles pleins à moitié transparents.

|  |  |
| --- | --- |
| **Transparence 1.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***// Cercles à 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();*** |  |

Voici le dessin de quatre cercles avec un contour. Les contours ne sont pas transparents mais l’intérieur de chaque cercle est transparent. Le taux de transparence n’est pas le même pour chaque cercle.

|  |  |
| --- | --- |
| **Transparence 2.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***objC2D.lineWidth = 2;***  ***// Cercle plein pas transparent (par défaut)***  ***objC2D.beginPath();***  ***objC2D.fillStyle = 'red';***  ***objC2D.strokeStyle = 'black';***  ***objC2D.beginPath();***  ***objC2D.arc(100,100,75,0,2\*Math.PI);***  ***objC2D.fill(); objC2D.stroke();***  ***// Cercle plein vert au 1/4 transparent***  ***objC2D.globalAlpha = 0.75;***  ***objC2D.fillStyle = 'green';***  ***objC2D.beginPath();***  ***objC2D.arc(200,100,75,0,2\*Math.PI);***  ***objC2D.fill();***  ***objC2D.globalAlpha = 1; objC2D.stroke();***  ***// Cercle plein bleu à moitié transparent***  ***objC2D.globalAlpha = 0.5;***  ***objC2D.fillStyle = 'blue';***  ***objC2D.beginPath();***  ***objC2D.arc(100,200,75,0,2\*Math.PI);***  ***objC2D.fill();***  ***objC2D.globalAlpha = 1; objC2D.stroke();***  ***// Cercle plein noir au 3/4 transparent***  ***objC2D.globalAlpha = 0.25;***  ***objC2D.fillStyle = 'black'***  ***objC2D.beginPath();***  ***objC2D.arc(200,200,75,0,2\*Math.PI);***  ***objC2D.fill();***  ***objC2D.globalAlpha = 1; objC2D.stroke();*** |  |

2. L’ombrage

En 2D, il est possible d’ajouter un ombrage à nos dessins. L’ombrage s’ajoute à l’aide de quatre (4) propriétés.

***.shadowColor :*** La couleur de l’ombre.

***.shadowBlur***: La quantité de flou dans l’ombre.

***.shadowOffsetX*** : Le décalage horizontal (en pixels) de l’ombre par rapport au dessin.

***.shadowOffsetY*** : Le décalage vertical (en pixels) de l’ombre par rapport au dessin.

Voici un exemple :

|  |  |
| --- | --- |
| **Ombrage.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***// Les paramètres de l'ombrage***  ***objC2D.shadowColor = 'black';***  ***objC2D.shadowBlur = 10;***  ***objC2D.shadowOffsetX = 15;***  ***objC2D.shadowOffsetY = 10;***  ***// Largeur de la ligne et couleurs***  ***objC2D.lineWidth = 2;***  ***objC2D.fillStyle = 'yellow';***  ***objC2D.strokeStyle = 'black';***  ***// Le dessin du texte***  ***var texte = 'Ombrage';***  ***objC2D.font = '50px Arial';***  ***objC2D.textAlign = 'left';***  ***objC2D.fillText(texte,0,50);***  ***objC2D.strokeText(texte,0,50);***  ***// Le dessin du cercle***  ***objC2D.beginPath();***  ***objC2D.arc(90,120,50,0,2\*Math.PI,false);***  ***objC2D.stroke();***  ***objC2D.fill();*** |  |

3. Les composites

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.
* On dessine un autre dessin. Ce dessin porte le nom de dessin source.

Dans les composites suivants, le carré bleu est dessiné puis la règle de composition est appliquée puis le cercle rouge est dessiné.

|  |  |
| --- | --- |
| ***source-over* :**  Le cercle rouge se dessine complètement par-dessus le carré bleu.  Il est important de prendre note que ce type est le type de composition par défaut. |  |
| ***source-atop* :**  Le cercle rouge se dessine partiellement par-dessus sur le carré bleu. En fait, concernant le cercle rouge, c’est son intersection avec le carré bleu qui est dessiné. |  |
| ***source-in* :**  Le cercle rouge se dessine partiellement mais on ne voit plus le carré. Concernant le cercle rouge, c’est son intersection avec le carré qui est dessiné. |  |
| ***source-out* :**  Le cercle rouge se dessine partiellement mais on ne voit plus le carré. Concernant le cercle rouge, c’est sa différence avec le carré (la partie qui n’est pas en intersection) qui est dessinée. |  |
| ***destination-over* :**  Le carré bleu se dessine complètement par-dessus le cercle rouge. En fait, c’est comme si on dessinait le cercle rouge puis le carré bleu. |  |
| ***destination-atop* :**  Le carré bleu se dessine partiellement par-dessus le cercle rouge. En fait, concernant le carré bleu, c’est son intersection avec le cercle rouge qui est dessiné. |  |
| ***destination-in* :**  Le carré bleu se dessine partiellement mais on ne voit plus le cercle. En fait, concernant le carré bleu, c’est son intersection avec le cercle qui est dessiné. |  |
| ***destination-out* :**  Le carré bleu se dessine partiellement mais on ne voit plus le cercle. En fait, concernant le carré bleu, c’est sa différence avec le cercle (la partie qui n’est pas en intersection) qui est dessinée. |  |

|  |  |
| --- | --- |
| ***lighter* :**  Le carré bleu et le cercle rouge se dessinent complètement mais la couleur de l’intersection entre les deux formes est l’union des deux couleurs. Les deux couleurs sont additionnées l’une à l’autre; c’est la raison pour laquelle la couleur de l’intersection est plus claire. |  |
| ***darker* :**  Le carré bleu et le cercle rouge se dessinent complètement mais la couleur de l’intersection entre les deux formes est l’intersection des deux couleurs. Les deux couleurs sont multipliées l’une à l’autre; c’est la raison pour laquelle la couleur de l’intersection est plus foncée. |  |
| ***xor* :**  Le carré bleu et le cercle rouge se dessinent complètement mais la couleur de l’intersection entre les deux formes est réalisée à l’aide d’un ou exclusif (**xor**). |  |
| ***copy* :**  Le cercle rouge se dessine complètement mais le carré bleu est complètement effacé. |  |

Voici le dessin de trois cercles pleins.

Le cercle rouge est dessiné à l’aide d’une règle de composition de type ***source-over*** (par défaut). Puis le cercle vert est dessiné à l’aide d’une règle de composition de type ***lighter.*** Finalement, le cercle bleu est dessiné à l’aide d’une règle de composition de type ***destination-atop***.

Prendre le temps de bien comprendre ce dessin.

|  |  |
| --- | --- |
| **Composition.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***// Composition de type destination-over (par défaut)***  ***// Cercle plein rouge***  ***objC2D.fillStyle = 'red';***  ***objC2D.beginPath();***  ***objC2D.arc(100,100,100,0,2\*Math.PI);***  ***objC2D.fill();***  ***// Composition de type lighter***  ***objC2D.globalCompositeOperation = 'lighter';***  ***// Cercle plein vert***  ***objC2D.fillStyle = 'green';***  ***objC2D.beginPath();***  ***objC2D.arc(200,100,100,0,2\*Math.PI);***  ***objC2D.fill();***  ***// Composition de type destination-atop***  ***objC2D.globalCompositeOperation = 'destination-atop';***  ***// Cercle plein bleu***  ***objC2D.fillStyle = 'blue';***  ***objC2D.beginPath();***  ***objC2D.arc(150,200,100,0,2\*Math.PI);***  ***objC2D.fill();*** |  |

4. La zone de découpage (le **clipping**)

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.

Mais il est possible de définir une zone de découpage particulière. Pour ce faire, il faut définir un tracé puis appliquer ce tracé à la méthode ***.clip().***

L’exemple suivant est le même exemple qu’à la page 1 sauf que la zone de découpage est un rectangle.

|  |  |
| --- | --- |
| **Découpage rectangulaire.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***// Zone de découpage rectangulaire***  ***objC2D.beginPath();***  ***objC2D.rect(100,100,100,100);***  ***objC2D.clip();***  ***// Le dessin des cercles***  ***objC2D.globalAlpha = 0.5;***  ***objC2D.fillStyle = 'red';***  ***objC2D.beginPath();***  ***objC2D.arc(100,100,100,0,2\*Math.PI);***  ***objC2D.fill();***  ***objC2D.fillStyle = 'green';***  ***objC2D.beginPath();***  ***objC2D.arc(200,100,100,0,2\*Math.PI);***  ***objC2D.fill();***  ***objC2D.fillStyle = 'blue';***  ***objC2D.beginPath();***  ***objC2D.arc(150,200,100,0,2\*Math.PI);***  ***objC2D.fill();*** |  |

L’exemple suivant est le même exemple qu’à la page 1 sauf que la zone de découpage est un cercle.

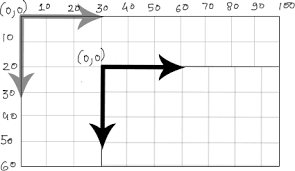
|  |  |
| --- | --- |
| **Découpage circulaire.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***// Zone de découpage circulaire***  ***objC2D.beginPath();***  ***objC2D.arc(150,150,100,0, 2\*Math.PI);***  ***objC2D.clip();***    ***// Le dessin des cercles***  ***objC2D.globalAlpha = 0.5;***  ***objC2D.fillStyle = 'red';***  ***objC2D.beginPath();***  ***objC2D.arc(100,100,100,0,2\*Math.PI);***  ***objC2D.fill();***  ***objC2D.fillStyle = 'green';***  ***objC2D.beginPath();***  ***objC2D.arc(200,100,100,0,2\*Math.PI);***  ***objC2D.fill();***  ***objC2D.fillStyle = 'blue';***  ***objC2D.beginPath();***  ***objC2D.arc(150,200,100,0,2\*Math.PI);***  ***objC2D.fill();*** |  |

Il est important de mentionner qu’il est possible d’utiliser n’importe quel tracé pour définir une zone de découpage, par exemple, des courbes de Bézier.

5. Les transformations du contexte

En 2D, il est possible de transformer le contexte du canevas. Lorsqu’on transforme le contexte du canevas, c’est comme si on transformait le canevas au complet.

Le principe est le suivant : on transforme le contexte puis on dessine comme si le contexte n’avait pas été transformé.

* **[](http://www.google.ca/url?sa=i&rct=j&q=&esrc=s&source=images&cd=&cad=rja&uact=8&ved=0CAcQjRw&url=http://codetheory.in/canvas-rotating-and-scaling-images-around-a-particular-point/&ei=dMzDVOvlHqfGsQSlz4KYDA&bvm=bv.84349003,d.cWc&psig=AFQjCNFggr9PEyTFtAhhUmlVJWAjXjIkyA&ust=1422204393273362)La translation du contexte**

Une des transformations bien connue est la translation. Translater le contexte signifie déplacer le contexte ailleurs à l’intérieur du canevas.

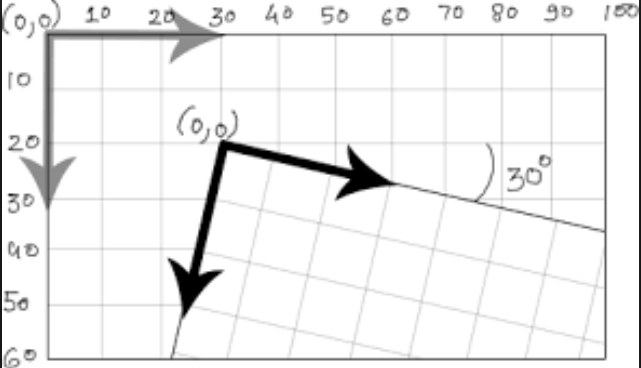
La méthode utilisée est ***.translate(intDeplx, intDeply)*** où ***intDeplx*** est le déplacement horizontal et où **intDeply** est le déplacement vertical. Voici un exemple.

|  |  |
| --- | --- |
| **Translation.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***// Cercle plein rouge***  ***objC2D.fillStyle = 'red';***  ***objC2D.beginPath(); objC2D.arc(50,50,50,0,2\*Math.PI); // Cercle rouge dessiné à la position (50,50)***  ***objC2D.fill();***  ***objC2D.translate(100,50); // Déplacer le contexte de (100,50)***  ***// Cercle plein vert***  ***objC2D.fillStyle = 'green';***  ***objC2D.beginPath(); objC2D.arc(50,50,50,0,2\*Math.PI); // Cercle vert dessiné à la position (50,50)***  ***objC2D.fill();***  ***objC2D.translate(-50,100); // Déplacer le contexte de (-50,100)***  ***// Cercle plein bleu***  ***objC2D.fillStyle = 'blue';***  ***objC2D.beginPath(); objC2D.arc(50,50,50,0,2\*Math.PI); // Cercle bleu dessiné à la position (50,50)***  ***objC2D.fill();*** |  |

Observez que les trois cercles sont toujours dessinés à la position (50, 50) à l’intérieur du contexte. Entre chaque dessin, le contexte s’est déplacé. C’est la raison pour laquelle on les voit à des positions différentes.

Il est important de prendre note que les transformations du contexte s’additionnent entre elles. Par exemple, si on déplace le contexte de (100, 50) puis, par la suite, on déplace le contexte de (-50,100), alors son déplacement réel est (50,150), c’est-à-dire (100,50) + (-50,100) = (50,150).

* **La rotation du contexte**

Une autre transformation bien connue est la rotation du contexte. La méthode utilisée est ***.rotate(fltAngle)*** où ***fltAngle*** est l’angle de rotation en radians (π radians = 180 degrés). Le point de rotation est toujours (0,0).

Voici un petit carré qui tourne autour du point (0,0) par coup de 20 degrés (ou π/9 radians).

|  |  |
| --- | --- |
| **Rotation.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***objC2D.fillStyle = 'purple';***  ***// Rotation du carré***  ***for (var i = 0; i < 18; i++) {***  ***objC2D.rotate(Math.PI/9); // Rotation du contexte de 20 degrés (Math.PI radians = 180 degrés)***  ***objC2D.fillRect(50,0, 10,10); // Le carré se dessine à la position (50,0)***  ***}*** |  |

Observez, encore une fois, que tous les carrés sont dessinés à la position (50, 0) à l’intérieur du contexte. Entre chaque dessin, le contexte a tourné. C’est la raison pour laquelle on les voit à des positions différentes.

Étant donné que les transformations du contexte s’additionnent entre elles et que chaque rotation est de 20 degrés, alors cela demande 18 rotations pour faire un tour complet.

Le problème ici, c’est que le point de rotation est toujours (0,0). Pour que le point de rotation soit un autre point, il faut déplacer le contexte avant d’effectuer la rotation.

|  |  |
| --- | --- |
| **Translation et rotation.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***// Déplacement du contexte au centre du canevas***  ***objC2D.translate(objCanvas.width/2, objCanvas.height/2);***  ***objC2D.fillStyle = 'purple';***  ***// Rotation du carré***  ***for (var i = 0; i < 18; i++) {***  ***objC2D.rotate(Math.PI/9); // Rotation du contexte de 20 degrés***  ***objC2D.fillRect(50,0, 10,10);***  ***}*** |  |

Ici, nous avons déplacé le contexte au centre du canevas avant d’effectuer les rotations. Cela fait en sorte, qu’au point de départ, le point (0,0) du contexte est situé au centre du canevas.

Une autre utilité de la rotation est d’écrire du texte dans un certain angle comme dans l’exemple suivant où nous tournons le contexte de 90 degrés (ou π/2 radians) avant d’écrire le texte.

|  |  |
| --- | --- |
| **Rotation du texte.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***// Tourner le contexte de 90 degrés***  ***objC2D.rotate(Math.PI/2);***  ***// Écrire le texte***  ***var strTexte = 'Écrit à 90 degrés';***  ***objC2D.fillStyle = 'magenta';***  ***objC2D.font = '30pt Verdana';***  ***objC2D.textBaseLine = 'top';***  ***objC2D.fillText(strTexte,0,0);*** |  |

* **La mise à l’échelle du contexte**

Une troisième transformation bien connue est la mise à l’échelle. Mettre à l’échelle le contexte signifie étirer ou contracter le contexte. La méthode utilisée est ***.scale(fltRapportX, fltRapportY)*** où ***fltRapportX*** est le rapport horizontal de mise à l’échelle et où ***fltRapportY*** est le rapport vertical de mise à l’échelle.

Il est à noter qu’un rapport inférieur à 1 va contracter le contexte et qu’un rapport supérieur à 1 va étirer le contexte. Par exemple, un rapport de 1/2 va contracter le contexte du double. Par contre, un rapport de 2/1 va étirer le contexte du double. Un rapport de 1 ne contracte ni n’étire le contexte.

Dans l’exemple suivant, on dessine un cercle mais, auparavant, on double la taille du contexte dans le sens horizontal. Cela donne l’impression qu’une ellipse a été dessinée.

|  |  |
| --- | --- |
| **Échelle 1.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***// 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();*** |  |

Dans l’exemple suivant, on dessine le même cercle mais on contracte le contexte dans le sens horizontal.

|  |  |
| --- | --- |
| **Échelle 2.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***// Contracter le contexte dans le sens horizontal***  ***objC2D.scale(0.5,1);***  ***// Cercle plein rouge***  ***objC2D.fillStyle = 'red';***  ***objC2D.beginPath();***  ***objC2D.arc(50,50,50,0,2\*Math.PI);***  ***objC2D.fill();*** |  |

Une valeur négative dans la mise à l’échelle donne un effet miroir ou un effet de retournement (un *flip*).

Dans l’exemple suivant, le texte « Effet miroir horizontal » est écrit. Observez qu’une translation horizontale a été effectuée avant d’effectuer la mise à l’échelle. La raison, c’est que le texte est écrit sur l’axe des X négatifs.

|  |  |
| --- | --- |
| **Miroir horizontal.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***// Effet miroir horizontal***  ***objC2D.translate(objCanvas.width,0);***  ***objC2D.scale(-1,1);***  ***// Écrire le texte***  ***var strTexte = 'Effet miroir horizontal';***  ***objC2D.fillStyle = 'Maroon';***  ***objC2D.font = '30px Verdana';***  ***objC2D.textBaseline='top';***  ***objC2D.textAlign='left';***  ***objC2D.fillText(strTexte,0,0);*** |  |

Dans l’exemple suivant, le texte « Effet miroir vertical » est écrit. Observez qu’une translation verticale a été effectuée avant d’effectuer la mise à l’échelle. La raison, c’est que le texte est écrit sur l’axe des Y négatifs.

|  |  |
| --- | --- |
| **Miroir vertical.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***// Effet miroir vertical***  ***objC2D.translate(0, 30);***  ***objC2D.scale(1,-1);***  ***// Écrire le texte***  ***var strTexte = 'Effet miroir vertical';***  ***objC2D.fillStyle = 'Maroon';***  ***objC2D.font = '30px Verdana';***  ***objC2D.textBaseline='top';***  ***objC2D.textAlign='left';***  ***objC2D.fillText(strTexte,0,0);*** |  |

* Les transformations personnalisées

Pour effectuer une transformation quelconque sur le contexte, une matrice de transformation qui a la forme suivante est toujours utilisée.

|  |
| --- |
| **X11  X12 X13**  **X21 X22 X23**  **0 0 1** |

Par exemple, pour effectuer une translation, c’est la matrice de transformation suivante qui est utilisée.

|  |
| --- |
| **1 0 *intDeplx***  **0 1 *intDeply***  **0 0 1** |

Par exemple, pour effectuer une mise à l’échelle, c’est la matrice de transformation suivante qui est utilisée.

|  |
| --- |
| ***fltRapportX* 0 0**  **0 *fltRapportY* 0**  **0 0 1** |

Il est possible de définir une matrice de transformation personnalisée en utilisant la méthode *.transform(****X11, X21, X12,X22, X13, X23****)*.

Par exemple, pour obtenir un effet de cisaillement (***shear***), nous devons utiliser la matrice de transformation suivante.

|  |
| --- |
| ***1 fltCisailleX* 0 *fltCisailleY* 1 0**  **0 0 1** |

où ***fltCisailleX*** est le cisaillement horizontal et où ***fltCisailleY*** est le cisaillement vertical.

Voici, par exemple, un effet de cisaillement horizontal de 0.5.

|  |  |
| --- | --- |
| **Cisaillement horizontal.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***// Effet de cisaillement horizontal***  ***var fltCisailleX = 0.5;***  ***var fltCisailleY = 0;***  ***objC2D.transform(1,fltCisailleY,fltCisailleX, 1, 0, 0);***  ***// Écrire le texte***  ***var strTexte = 'CISAILLEMENT';***  ***objC2D.fillStyle = 'Maroon';***  ***objC2D.font = '20px Verdana';***  ***objC2D.textBaseline='top';***  ***objC2D.textAlign='left';***  ***objC2D.fillText(strTexte,0,0);*** |  |

Voici, par exemple, un effet de cisaillement vertical de 0.5.

|  |  |
| --- | --- |
| **Cisaillement vertical.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***// Effet de cisaillement vertical***  ***var fltCisailleX = 0;***  ***var fltCisailleY = 0.5;***  ***objC2D.transform(1,fltCisailleY,fltCisailleX, 1, 0, 0);***  ***// Écrire le texte***  ***var strTexte = 'CISAILLEMENT';***  ***objC2D.fillStyle = 'Maroon';***  ***objC2D.font = '20px Verdana';***  ***objC2D.textBaseline='top';***  ***objC2D.textAlign='left';***  ***objC2D.fillText(strTexte,0,0);*** |  |

Grâce à la méthode ***.transform***, le nombre de transformations sur le contexte est pratiquement illimité. Il est possible de créer d’autres types de transformations pour obtenir des effets farfelus.

6. 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 écrivez ***objC2D.fillStyle='yellow'***, la couleur de remplissage sera jaune et ce, jusqu’à la fin du programme à moins qu’elle ne soit modifiée par la suite. Par exemple, si vous écrivez ***objC2D.clip()***, la zone de découpage sera celle que vous avez définie et ce, jusqu’à la fin du programme à moins qu’elle ne soit modifiée par la suite. Par exemple, si vous écrivez ***objC2D.rotate(Math.PI/2),*** le contexte subira une rotation de 90 degrés et ce, jusqu’à la fin du programme à moins qu’il ne soit modifié par la suite.

En 2D, il est possible de sauvegarder l’état du contexte et de le restaurer par la suite. Pour sauvegarder l’état du contexte, on utilise la méthode ***.save()*** et pour restaurer l’état du contexte, on utilise la méthode ***.restore().*** Cela est pratique lorsqu’on écrit des programmes qui sont composés de plusieurs fonctions.

Par exemple, supposons qu’on programme une fonction qui dessine une forme ovale (ici, le contexte est passé en paramètre).

|  |  |
| --- | --- |
| ***function dessinerOvale(objC2D) {***  ***objC2D.save(); // On sauvegarde le contexte actuel***  ***// Contracter le contexte dans le sens horizontal***  ***objC2D.scale(0.5,1); // Contexte modifié***  ***// Cercle plein rouge***  ***objC2D.fillStyle = 'red'; // Contexte modifié***  ***objC2D.beginPath(); objC2D.arc(50,50,50,0,2\*Math.PI);***  ***objC2D.fill();***  ***objC2D.restore(); // On restaure l’ancien contexte***  ***}*** |  |

Au début de la fonction, on sauvegarde le contexte. Puis, dans le dessin de la forme ovale, le contexte est modifié (***objC2D.scale(0.5,1); objC2D.fillStyle = 'red';***). A la fin de la fonction, on restaure le contexte qu’on a sauvegardé. De cette manière, avant l’appel de la fonction et après l’appel de la fonction, le contexte va être le même. Chacune des fonctions va utiliser son propre contexte.

7. L’accès direct aux pixels

En 2D, de manière interne, un canevas est un conteneur de pixels. Dans ce conteneur, chaque pixel est un quadruplet d’octets : **(Rouge, Vert, Bleu, Alpha)**. C’est la combinaison de ces quatre octets qui donnent la vraie couleur. Plus la valeur de la couleur est grande, plus la couleur est claire (et moins la valeur est grande et plus la couleur est sombre). La valeur minimum est 0 et la valeur maximum est 255.

Par exemple, le quadruplet **(255,0,0,255)** donne un rouge très clair opaque; le quadruplet **(0,0,64,128)** donne un bleu foncé à moitié opaque; le quadruplet **(64,64,0,64)** donne un jaune foncé au quart opaque; le quadruplet **(255,255,255,128)** donne un blanc à moitié opaque; le quadruplet **(128,128,128,255)** donne un gris moyen opaque; le quadruplet **(0,0,0,255)** donne un noir opaque.

Il est possible d’avoir un accès direct à ce conteneur. Cela nous donne la possibilité de faire du traitement pixel par pixel. Voici les trois (3) méthodes que l’on peut utiliser :

***.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 ***(intX,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).***

Le conteneur de pixels contient un tableau d’octets (***.data***) de taille **intLargeur \* intHauteur \* 4**. Par exemple, un conteneur de taille 20 X 10 pixels contient 800 octets (20\*10\*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.

etc…

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

Voici un programme qui affecte des couleurs au hasard dans le canevas.

|  |  |
| --- | --- |
| **Couleurs hasard.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***// Créer un conteneur de pixels de taille 200 X 100***  ***// Ce conteneur contient 20000 pixels (200\*100)***  ***// Le tableau contient 80000 octets (20000\*4)***  ***var objPixels = objC2D.createImageData(200,100);***  ***for (var i=0; i < objPixels.data.length; i +=4)***  ***{ // Parcourir le tableau d’octets***  ***// 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);*** |  |

Voici un programme qui transforme une image couleur en noir et blanc.

Le principe est le suivant : Pour chaque pixel, on additionne chacune des couleurs (sans toucher à l’alpha) puis on divise par 3. On affecte cette valeur à chaque couleur. Cela va donner un ton de gris spécifique. Par exemple: la couleur jaune claire (255,255,0,255) va donner le ton de gris suivant (170,170,170,255).

|  |  |
| --- | --- |
| **En noir et blanc.htm** |  |
| ***var objC2D = objCanvas.getContext('2d');***  ***var 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 l’image dessinée***  ***var objPixels = objC2D.getImageData( 10,10,objImage.width, objImage.height);***  ***// Parcourir le tableau d’octets***  ***for (var i=0; i < objPixels.data.length; i +=4) {***  ***var enGris = Math.floor((objPixels.data[i] +***  ***objPixels.data[i+1] + objPixels.data[i+2]) / 3);***  ***// Changer la couleur des pixels***  ***for (var j=0; j < 3; j++) objPixels.data[i+j] = enGris;***  ***}***  ***// Redessiner l'image transformée***  ***objC2D.putImageData(objPixels,10, 20 + objImage.height);***  ***}*** |  |

Ici, l’image du haut est l’image originale et l’image du bas est l’image transformée (en noir et blanc).

Note : Cela fonctionne bien avec **Internet Explorer** mais ne fonctionne pas avec **Chrome**.

Le problème avec **Chrome**, c’est qu’on ne peut accéder au conteneur de pixels d’une image qui provient d’un fichier local (enregistré sur le disque dur de l’utilisateur). **Google Chrome** invoque que ce comportement est une question de sécurité.

Une première solution à ce problème est de placer l’image originale sur un serveur Web qui permet l’accès au conteneur de pixels d’une image et de charger cette image située sur ce serveur Web (à l’aide de son adresse URL).

L’autre solution à ce problème est d’ouvrir **Chrome** en lui passant en paramètre **--allow-file-access-from-files**.

Pour ce faire, suivez les étapes suivantes :

1. Créez un raccourci sur **Chrome.exe**.
2. Dans les propriétés du raccourci, au niveau de la **cible**, ajoutez **‑‑allow-file-access-from-files.**
3. Ouvrez **Chrome** à l’aide de ce raccourci.
4. Sans fermer **Chrome**, double-cliquez sur le fichier **En noir et blanc.htm** pour l’ouvrir.

Il est important de prendre note que le conteneur de pixels contient des données brutes. Il est le reflet exact de ce qui est dessiné dans le canevas. Par conséquent, ils ne sont pas influencés par les changements d’état du contexte.

7. Références

<http://dev.w3.org/2006/canvas-api/canvas-2d-api.html>

<http://www.w3schools.com>

<http://www.alsacreations.com/tuto/lire/1484-introduction.html>

<http://cheatsheetworld.com/programming/html5-canvas-cheat-sheet/>

<http://www.html5canvastutorials.com/>

<https://developer.mozilla.org/samples/canvas-tutorial/6_1_canvas_composite.html>