WebGL TP4

LP DAWIN

2015-2016

1 Ajouter de la couleur

Jusqu'à présent, la couleur de vos dessins était dictée par le fragment shader. Nous allons voir comment la changer.

Repartez du programme qui dessine un triangle.

- Ajoutez un attribut `color` dans le vertex shader et un buffer associé.
- Dans ce buffer, rajoutez une couleur par sommet du triangle (rappel : une couleur = 3 nombres).
- Ajoutez dans les shaders une nouvelle variable de type `varying`, que vous nommerez `vColor`. Les variables `varying` sont des sorties du vertex shader vers le fragment shader, elles doivent donc apparaitre de la même façon dans les deux. Dans le vertex shader, utilisez l'attribut color comme valeur du varying vColor. Dans le fragment shader, utilisez le varying vColor comme valeur de `gl_FragColor`.
- Dessinez tout ça. Que se passe-t-il ?

2 Transformations

2.1 Translation

Pour le moment, l'emplacement à l'écran de vos dessins est fixé dès leur création. Si vous vouliez les déplacer, il faut tout recalculer.

Dans cet exercice, nous allons remédier à ce problème en utilisant les variables `uniform`.

1. Commencez par créer un programme qui dessine un seul triangle coloré. Il ne sera ensuite plus nécéssaire de changer le contenu des buffers. Votre triangle doit être centré en (0,0).
2. Ajoutez dans le vertex shader une variable `uniform vec2 translation` et utilisez la pour déplacer les sommets.
3. Lors du dessin, passez un vecteur de translation à cette variable en utilisant `gl.uniform2f(location, x, y)`. Pensez à récupérer l'emplacement de l'uniform avec `gl.getUniformLocation`.
4. Grace à la translation, implémentez un déplacement du triangle au clavier.

2.2 Mise à l'échelle et rotation

En reprenant le programme précédent, nous allons ajouter les deux autres transformations élémentaires : la mise à l'échelle et la rotation.

1. Ajoutez deux variables `uniform` dans le vertex shader, une pour chaque transformation. La mise à l'échelle est un coefficient multiplicateur. Les formules pour la rotation d'angle `a` sont : `rotatedX = x*cos(a)-y*sin(a); rotatedY = y*cos(a)+x*sin(a);`.
2. De même que précédemment, implémentez un controle au clavier de ces transformations.

2.3 Ordre des transformations

Essayez de changer l'ordre dans lequel vous appliquez les transformations et observez la différence.

3 Animations

Dans l'exercice précédent, vous avez utilisé une mise à l'échelle et une rotation aléatoire par triangle. Nous allons maintenant voir comment les animer. Une animation est un processus qui varie dans le temps, on peut donc le décrire par une équation en fonction d'une variable `t`, par exemple, la rotation peut être `r(t) = t % 2*PI`.

1. Avant de penser à animer votre dessin, il faut qu'il se redessine tout seul. On utilise pour cela `requestAnimationFrame(callback)` qui permet de demander au navigateur de rappeler une fonction au prochain dessin de l'écran. Cet appel doit etre renouvelé à chaque dessin, le mieux est donc de le placer au début de votre fonction de dessin.
2. Pour controler l'avancement de l'animation, ajoutez un `uniform float time` dans le vertex shader et incrémentez-le à chaque dessin. La valeur de l'incrément contrôlera la vitesse de l'animation.
3. Dans le vertex shader, remplacez les utilisation des uniform de mise à l'echelle et de rotation par des equations fonction de `time`. Les fonctions trigonométriques (notament `sin`) sont très utiles, puisque periodiques.

4 Matrice de transformations

Vous avez dû constater que l'ordre dans lequel on applique les transformations est important. Si cet ordre est fixé dans le shader et qu'on veut le changer, il faut modifier le code du shader. De plus, il faut alors envoyer séparément toutes les transformations. Pour remédier à cela, on encode les transformations sous forme d'une matrice.

1. Nous utiliserons la bibliothèque `glmatrix`, que vous trouverez sur `glmatrix.net`. Récupérez la bibliothèque `.js` et incluez la dans votre page.
2. Pour encoder des transformations en 2D, nous utiliserons l'objet `mat3`, qui représente une matrice 3x3. Pensez à lire la documentation.
3. Réimplémentez les exercices de transformations avec une matrice. Pour passer la matrice au shader, vous utiliserez un `uniform mat3 matrix`. Pour appliquer les transformation, il suffit de multiplier la matrice par la position à transformer. Faites attention à la légalité des multiplication : on ne peut multiplier une `mat3` que par un `vec3`, ce qui donne un vec3.