WebGL TP2

LP DAWIN

2015-2016

1 Les shaders

Avant de pouvoir dessiner quoi que ce soit avec WebGL, vous devez charger les *shaders*. Les shaders sont de petits programmes qui s'éxécutent sur la carte graphique, codés en *GLSL*.

Il sont de deux types en WebGL :

- Le *vertex shader* est exécuté pour pour chaque vertex que vous définissez.
- Le *fragment shader* (ou *pixel shader*) est exécuté pour chaque fragment (pour simplifier, un pixel).

Le code de chacun de ces shaders doit être *compilé*, puis *linké*, comme un programme C ou C++ classique. Cependant, cette étape a lieu à l'éxécution du javascript (dans la fonction `main()`) et non en amont (avant l'éxécution). L'ensemble d'un vertex et d'un fragment shader compilés et linkés forme un *programme*.

Nous définirons les shaders dans des fichiers séparés. Vous devez donc avoir quatre fichier : un .html, un .js et les deux shaders.

Par convention, le vertex shader porte l'extension *.vert* et le fragment shader *.frag*. Commencez par créer deux fichiers shaders avec le code minimal suivant:

```
void main() {

}
```

1.1 Initialisation des shaders

Les shaders sont créés après avoir initialisé le contexte WebGL, dans le `main()` de votre fichier javascript. La procédure pour initialiser les shaders est fastidieuse, mais indispensable, suivez bien les étapes :

1. Récupérez le code source des shaders à l'aide du code js suivant (url sera le chemin de votre fichier shader) :

```
function loadText(url) {
    var xhr = new XMLHttpRequest();
    xhr.open('GET', url, false);
    xhr.send(null);
    if(xhr.status === 200)
        return xhr.responseText;
```

```
    else {
        return null;
    }
}
```

2. Créez un vertex shader et un fragment shader à l'aide de `createShader(type)`. L'argument `type` est `VERTEX_SHADER` ou `FRAGMENT_SHADER`.
3. Donnez les sources de chaque shader avec `shaderSource(shader, source)`.
4. Compilez chaque shader à l'aide `compileShader(shader)`.
5. Créez un nouveau programme avec `createProgram()`.
6. Attachez les deux shaders au programme avec `attachShader(program, shader)`.
7. Linkez le programm avec `linkProgram(program)`.
8. Demandez à WebGL d'utiliser ce programe avec `useProgram(program)`.

1.2 Gestion des erreurs de compilation

Sauf si vous connaissez sur le bout des doigts la syntaxe GLSL, vous aurez des erreurs de compilation de vos shaders. Voici comment vous assurer que la compilation fonctionne et, au besoin, récupérer les erreurs associés.

1. Pour les erreurs de compilation, après l'étape 4 de l'exercice précédent, vérifiez le statut de la compilation avec `gl.getShaderParameter(shader, gl.COMPILE_STATUS)`. En cas d'échec, il est possible de récupérer les erreurs avec `gl.getShaderInfoLog(shader)`.
2. De même, pour les erreurs de linkage (après l'étape 7), utilisez `gl.getProgramParameter(program, gl.LINK_STATUS)` et `gl.getProgramInfoLog (program)`.

2 Dessin d'un point

Maintenant que vous avez des shaders actifs, vous allez pouvoir (enfin !) dessiner un point. La fonction `drawArrays(mode, first, count)` déclenche le dessin. Son paramètre `mode` spécifie le type de dessin à effectuer, pour le moment vous utiliserez `POINTS` pour dessiner des points (oh surprise !). Les paramètres `first` et `count` controlent ce qui est déssiné. Pour l'instant nous ne dessinons qu'un seul point, donc first=0 et count=1.

1. Utilisez `drawArrays` pour demander le dessin d'un point.
2. Normalement, le point n'est pas apparu lors de l'appel à `drawArrays`. Pour cela, vous allez devoir modifier les shaders. Dans le vertex shader, définissez les variables `gl_Position` (de type vec4) et gl_PointSize (de type float) et dans le fragment shader, définissez `gl_FragColor` (de type vec4). Pour gl_Position, on ne s'occupe pas du `z` et le quatrième paramètre reste à `1.0`. Relancez le dessin, le point devrait apparaitre.
3. Essayez différentes valeurs pour les trois variables précédentes. Déduisez-en leur rôle.
4. Determinez les valeurs limites de gl_Position pour lesquelles le point est visible à l'écran.

3 Suivre la souris

Nous allons maintenant modifier le programme pour que le point se dessine à la position de la souris. Pour cela, nous utiliserons un attribut pour modifier la position dans le vertex shader. Pensez à isoler votre code de dessin dans une fonction `draw` pour pouvoir le réutiliser.

1. Ajoutez un `attribute vec2 position_souris` dans le vertex shader. Sa déclaration doit se trouver avant `main`. Pensez à modifier votre shader pour utiliser cette valeur dans gl_Position.
2. Récupérez l'adresse de l'attribut dans le programme avec `getAttibLocation(program, attribName)`.
3. Passez une valeur à l'attribut avant de dessinez, à l'aide de `vertexAttrib2f(attrib, x, y)`.
4. Utilisez l'event `canvas.onmousemove` pour récupérer la position de la souris et redessiner. Pensez à faire un changement de repère, les coordonnées de la souris et ceux de WebGL sont différents.

4 Peinturlurage

1. Ajoutez l'option `preserveDrawingBuffer` à `true` lors de la création du contexte.
2. Commentez l'appel à `clear`
3. ???
4. PROFIT !!!

5 Points multiples

5.1 Grille de points

1. Ecrivez un programme qui dessine une grille de 10*10 points. Indice : utilisez (au moins une) une boucle for.

5.2 Variations de couleur

1. Utilisez les coordonnées de chaque fragment pour modifier sa couleur, grace à la variable `glFragCoord`. Faites en sorte que les points les plus à gauche soient plus rouges, et les points les plus bas soient plus bleus.
2. De même, faites en sorte que la couleur verte dépende de la distance au centre.

5.3 Variations de taille

Reprenez l'exercice précédent, mais faites varier la taille au lieu de la couleur.

5.4 Variations autour de la souris

Réimplémentez les variations des exercices précédents, mais faites en sorte qu'elles se fassent en fonction de la position de la souris. Pour cela, vous aurez besoin de transmettre la position de la souris au shader par une variable `uniform`. La procédure est la même que pour un attribut : modifier le shader, récupérer la `location` puis mettre une valeur avec `gl.uniform2f`.

5.5 Variations sous la souris

Maintenant, faites en sorte que le point sous la souris change de taille et de couleur. Vous aurez besoin d'une variable `varying` pour passer des informations du vertex shader au fragment shader.

5.6 Dessin dans le point

1. Utilisez la variable `gl_PointCoord` pour dessiner un cercle de couleur différente à l'interieur de chaque point.
2. Faites varier la taille de ce cercle en fonction de la distance au centre de l'écran.

5.7 Points au clic

Modifiez le programme qui suit la souris pour dessiner plusieurs points placés avec un clic.

1. Utiliser l'evenement canvas.onclick pour récuperer et stocker les positions des clics de souris.
2. Modifier le programme pour dessiner autant de points qu'il y a eu de clic.
3. Ajoutez une couleur aléatoire à chaque point, tirée à la création.
4. Faites de même pour la taille.