IUT - Département Informatique

LP Dawin - WebGL - 2016-2017

Semaine 2 - Travaux Pratiques

WebGL TP2

NB : Si vous avez un doute sur l'utilisation d'une fonction WebGL, référez-vous au mémo.

1 Les buffers

Les buffers sont des espace-mémoire sur la carte graphique permettant de limiter le temps de transfert des données et d'accélérer significativement le rendu 3D. Ils sont aussi indispensables lorsque l'on cherche à afficher plus d'un point à la fois (cf. avertissement dans le navigateur si l'on n'utilise pas de buffer). Les étapes pour utiliser un buffer sont les suivantes :

1. On réserve l'espace mémoire par (on crée un objet `buffer`) :

```
var buffer = gl.createBuffer();
```

2. On choisit le buffer sur lequel on veut travailler avec :

```
gl.bindBuffer(type, buffer);
```

   Le type de buffer peut être `gl.ARRAY_BUFFER`, ou `gl.ELEMENT_ARRAY_BUFFER`. Pour l'instant, on utilisera `gl.ARRAY_BUFFER`.

3. On remplit l'espace mémoire avec les points que l'on veut dessiner :

```
gl.bufferData(type, points, dessin)
```

   Concernant la variable `dessin`, on choisira `gl.STATIC_DRAW`.

Les données sont sur la mémoire de la carte graphique. Il reste à les utiliser lors du dessin :

1. Autoriser la transmission des données contenues dans le buffer à la variable attribute par : `gl.enableVertexAttribArray(location)`.
2. Passer les données contenues dans le buffer "bindé" à la variable `attribute position` avec la fonction `gl.vertexAttribPointer(location, size, type, normalized, stride, offset)`. Les 2 derniers paramètres servent à faire des décalages dans la mémoire. Ici, on prend la première case mémoire, les paramètres sont donc fixés à 0.
3. Enfin, dessiner avec `drawArrays` (n'oubliez pas de spécifier le paramètre `count`).

Points multiples v2 Récupérer le code du TP2 puis :

1. Dans le code chargé d'afficher des points à chaque clic de souris, complétez la fonction `initBuffer()` qui se charge d'initialiser le buffer, ainsi qu'une méthode `refreshBuffer()` chargée de mettre à jour le buffer avec les nouvelles coordonnées de points après un clic.

   Intégrez ce buffer à votre code pour effectuer le dessin.
2. Essayer de dessiner avec d'autres **primitives de dessin** dans `drawArrays` : utiliser par exemple `LINES`, `LINE_STRIP`, `LINE_LOOP`, `TRIANGLES`, `TRIANGLE_STRIP` et `TRIANGLE_FAN`. Quelles sont les caractéristiques de chacune des primitives ?

2 Dessin de polygones

Vous pouvez maintenant dessiner des triangles en utilisant `gl.TRIANGLES` pour le mode de dessin. Réalisez trois programmes (les buffers ne doivent pas être modifiés après l'initialisation) :

1. Un qui dessine un simple triangle. Quelle primitive de dessin utiliser ?
2. Un qui dessine des triangles répartis sur une grille.
3. Un qui dessine un carré. Quelle primitive de dessin utiliser ? L'ordre des points dans le tableau a-t-il de l'importance ? Pourquoi ?

3 Couleur des sommets

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

1. Ajoutez un attribut `color` dans le **vertex shader** et un buffer associé.
2. Dans ce buffer, rajoutez une couleur par sommet du triangle (rappel : une couleur = 4 canaux).
3. Ajoutez dans les shaders une nouvelle variable de descripteur `varying`, que vous nommerez `vColor`. Les variables `varying` sont des sorties du vertex shader vers le fragment shader, elles doivent donc apparaître de la même façon dans les deux shaders. 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 la scène. De quelle couleur est le triangle ?

4 Transformations

Pour effectuer les transformations qui suivent (translation, rotation, homothétie) assurez vous que le triangle soit centré en (0,0). Il sera ensuite inutile de modifier le contenu des buffers.

4.1 Translation

Pour le moment, l'emplacement des sommets est fixé dès leur création. Si vous vouliez les déplacer, il faudrait redéfinir de nouveaux sommets. Dans cet exercice, nous allons remédier à ce problème en utilisant les variables de descripteur `uniform`.

1. Ajoutez dans le vertex shader une variable `uniform vec2 translation` et utilisez la pour déplacer les sommets.
2. Récupérez la variable `translation` dans le Javascript
3. Lors du dessin, passez un vecteur de translation à cette variable en utilisant `gl.uniform2f`
4. Grâce à la translation, implémentez un déplacement du triangle au clavier

4.2 Homothétie

L'homothétie consiste à agrandir ou rétrécir une forme suivant un facteur d'échelle.

Question Suivant la même méthodologie que pour la translation, reprenez les étapes pour effectuer un agrandissement ou un rétrécissement de l'objet au clavier.

4.3 Rotation

Les nouvelles coordonnées $(x'y')$ d'un point (x, y) par une rotation d'angle α sont :
$x' = x * cos(\alpha) - y * sin(\alpha)$
$y' = y * cos(\alpha) + x * cos(\alpha)$

Question Suivant la même méthodologie que pour la translation, reprenez les étapes pour effectuer une rotation du triangle au clavier.

4.4 Ordre des transformations

Changer l'ordre des transformations (essayer toutes les combinaisons). L'ordre des transformations est-il important ? Pourquoi ?

5 Animations

Dans l'exercice précédent, vous avez utilisé une mise à l'échelle et une rotation en fonction d'événements du clavier. Nous allons maintenant voir comment générer automatiquement une transformation en fonction du temps. C'est ce que l'on appelle une animation. Une animation est un processus qui varie dans le temps.

1. Il est nécessaire que la scène soit redessinée automatiquement. On utilise pour cela la fonction `requestAnimationFrame(callback)`. A quoi correspond ici la fonction `callback` ?

2. Pour contrôler l'avancement de l'animation, ajoutez une variable `time` dans le fichier javascript et incrémentez-le à chaque dessin. La valeur de l'incrément contrôlera la vitesse de l'animation.
3. Modifiez la valeur d'angle de rotation en fonction du temps. On rappelle que les valeurs d'angle sont comprises entre 0 et 2π.
4. Effectuer une translation horizontale automatique de sorte à ce que le triangle "rebondisse" sur les bords droits et gauche du canvas.

6 Matrices de transformation

Effectuer les transformations une à une est contraignant et source d'erreurs, notamment car l'ordre des transformations est important (cf question 4.4). Pour remédier à cela, on encode les transformations sous la forme d'**une seule** matrice (contenant toutes les transformations) et qu'il suffira de multiplier par les positions des sommets pour obtenir les nouvelles coordonnées transformées.

1. Javascript ne dispose pas nativement d'une bibliothèque de gestion de matrices. On importe donc la bibliothèque `glmatrix`, récupérable à l'adresse : `https://raw.githubusercontent.com/toji/gl-matrix/master/dist/gl-matrix-min.js`. Incluez la bibliothèque `gl-matrix-min.js` dans la page HTML.
2. Pour encoder des transformations en 2D, nous utiliserons l'objet `mat4`, qui représente une matrice 4x4. Pensez à lire la documentation à l'adresse `http://glmatrix.net/docs/mat4.html`
3. Encodez les transformations (rotation, translation) sous la forme d'une seule matrice `mat4`.
   (a) Définissez les transformations à l'aide des méthodes `rotate()` et `translate()` de `glmatrix` dans le fichier javascript.
   (b) Dans le vertex shader, il est nécessaire de déclarer un `uniform` de type `mat4`. Récupérez cette variable dans le fichier javascript avec `getUniformLocation()`. La fonction `gl.uniformMatrix4fv()` permet de passer la matrice définie dans le fichier javascript au vertex shader.
   (c) Pour appliquer les transformations dans le shader, il suffit de multiplier la matrice par la position à transformer. Faites attention à l'ordre des multiplications : on ne peut multiplier une mat4 que par un vec4 (et non l'inverse) : le résultat est un vec4.