IUT - Département Informatique

LP Dawin - WebGL - 2016-2017

Semaine 3 - Travaux Pratiques

WebGL TP3

Le but de ce TP est de passer de le 2D à la 3D. Pour cela, vous aurez besoin de la bibliothèque `glmatrix`, que vous trouverez à l'url `https://raw.githubusercontent.com/toji/gl-matrix/master/dist/gl-matrix-min.js`. Pensez à lire la documentation, le seul type que nous utiliserons est `mat4`. Pour commencer, vous devez avoir un dessin de triangle en 2D fonctionnel.

1 Objet 2D dans un monde 3D

Pour dessiner un objet 3D, il faut le transformer en image 2D affichable à l'écran. L'affichage dépend de trois paramètres :
— le point de vue, c'est-à-dire la position de la caméra
— la lumière
— l'objet (matériau, texture)
Dans ce TP, on ne s'occupe que d'un rendu 3D simple, qui prend en compte uniquement la position de la caméra. Le but est d'effectuer une **projection** du monde 3D en une image 2D.

Questions

1. La **matrice de projection** définit le champ de vision : elle définit notamment la distance à partir de laquelle les objets ne sont plus apparents. Ajoutez dans votre programme une **matrice de projection** et remplissez la simplement en utilisant la fonction `mat4.perspective`. Quels sont les arguments de cette fonction ? A quoi chacun correspond-il ?
2. Ajoutez un `uniform` de type `mat4` dans le vertex shader et utilisez `gl.uniformMatrix4fv` pour y envoyer la matrice créée en question 1.
3. Appliquer la projection. Le triangle est-il visible ? Dans le vertex shader, essayer plusieurs valeurs de `z` pour `gl_Position` et observez le résultat. Pour quelles valeurs le triangle apparaît-il ? Pourquoi ?
4. Modifiez votre programme pour que le buffer de position prenne en compte une troisième coordonnée (z). Effectuez toutes les modifications nécessaires associées à l'utilisation du buffer.

2 Transformations 3D

Le but est d'appliquer les transformations (rotation, homothétie et translation) vues en TP2 dans le monde 3D. Avant d'appliquer les transformations, il est nécessaire de déplacer l'objet afin de le voir à partir de la position par défaut de la caméra (0,0,0). Ce déplacement s'effectue par l'intermédiaire d'une matrice supplémentaire, appelée matrice **modèle**, qui permet de positionner l'objet dans la scène.

2.1 Déplacer la caméra

1. Remettez les z de l'objet à 0.
2. Créer la matrice **modèle** dans le vertex shader. Faites les modifications nécessaires associées pour l'utiliser (cf. partie précédente, dans le fichier javascript et le shader).
3. Quelle transformation utiliser pour visualiser l'objet ? Appliquer cette transformation dans la matrice **modèle**. L'objet devrait apparaître.

2.2 Rotation 3D

Implémentez les **animations** suivantes :

1. Effectuer une rotation autour de l'axe Z.
2. Faites tourner le triangle sur lui-même autour de la verticale.

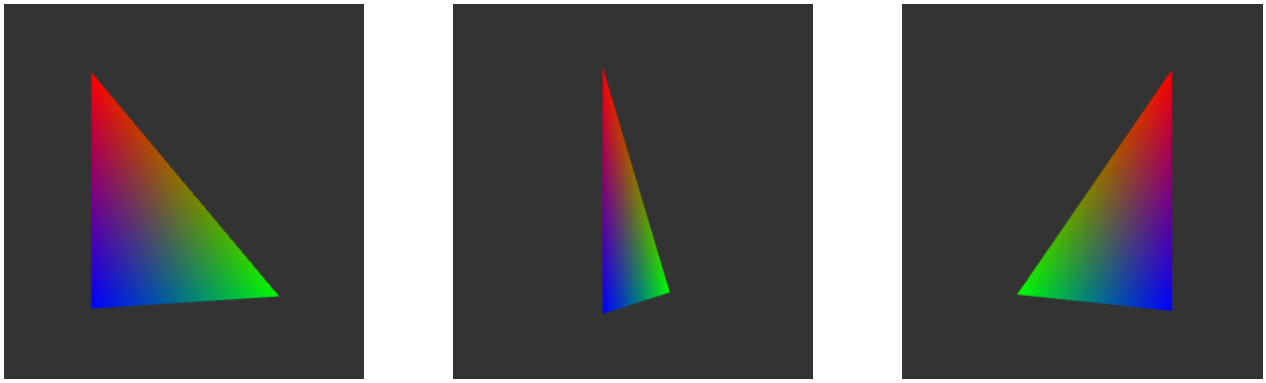

FIGURE 1 – Rotation du triangle 2D autour de l'axe Y

3 Rendu d'un cube

Dans cette partie, on va définir un objet 3D (x, y, **et** z), le cube, dont on s'attachera à faire le rendu.

Questions

1. Modifier les buffers pour dessiner un cube (6 faces, 12 triangles, 36 sommets)
2. Ajouter de la couleur aux sommets du cube en fonction de leur position. Pour cela, pas besoin d'ajouter un buffer, vous pouvez directement passer les positions des sommets à l'attribute `color` défini dans le vertex shader.
3. Ajouter une animation de rotation du cube. Que constatez-vous ?
4. Y remedier en ajoutant `gl.enable(gl.DEPTH_TEST)` dans la fonction de dessin. Il faut aussi changer le clear en `gl.clear(gl.COLOR_BUFFER_BIT | gl.DEPTH_BUFFER_BIT)`.
5. Rechercher sur Internet des informations sur le Depth Buffer. De quoi s'agit-il ? Quelle est sa fonction ?