Primitives et shader variables

Familiarisation avec le code

Examinez le code dans les fichiers cube.js et cube.html et visualisez le résultat.

Cube.js contient une classe permettant d'initialiser et de dessiner un cube (un peu comme la boite utilisée dans le TP précédent pour le robot).

Comparez l'initialisation des buffers du cube avec ceux que l'on a utilisé pour la boite du robot. Qu'est-ce qui a changé ? Que représente le tableau « vertices ». Que représente le tableau « tri ». Comment ces tableaux sont-ils liés ?

Regardez la fonction **draw** (toujours dans cube.js). Pourquoi a-t-on utilisé gl.drawElements plutôt que gl.drawArrays comme on l'avait fait avec le robot ? En quoi cette méthode peut-elle être plus avantageuse ?

Cube.html contient le reste du script. Familiarisez vous avec les fonctions et lisez les commentaires. Quelle fonction est appelée en premier ? Repérez la fonction de dessin. Quel type de projection est utilisée ici ? Ou a-t-on placé la caméra ?

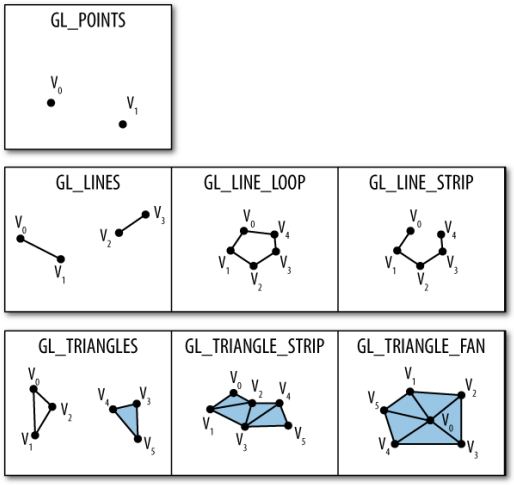
Repérez les vertex et fragment shaders. Expliquez ce que fait la ligne suivante :

gl_Position = uPMatrix * uMVMatrix * vec4(aVertexPosition, 1.0);

Dans le fragment shader, repérez ou on a affecté la couleur de chacun des pixels. Modifiez cette couleur et visualisez le résultat. Quelle est la différence entre un attribut et une variable uniforme ?

Primitives et variables uniformes

Dans le fichier cube.js, repérez ou se trouve l'instruction qui permet de dessiner les lignes du cube. Les primitives pouvant être affichées par OpenGL ainsi que leur mode de rendu sont listées ci-dessous :



D'après l'initialisation des buffers qui a été mise en place, quelle est le mode adapté pour dessiner des triangles plutôt que des lignes ? Pourquoi ? Le tester.

Placez vous ensuite dans le mode point et tester le rendu. Repérez la ligne dans le vertex shader qui contrôle la taille des points affichés. Essayez de modifier cette taille.

On souhaite contrôler cette taille en appuyant sur des touches du clavier. Pour cela, il faut procéder de la même manière que pour les matrices :

1. Créer une variable (globale pour simplifier) représentant la taille des points dans le script.
2. Incrémentez/décrémentez la dans la fonction « keypressed »
3. Dans le vertex shader, définissez une nouvelle variable (uniform) du nom de votre choix
4. Affectez la au bon endroit dans le shader
5. Dans la fonction « initShaders », ajoutez un attribut à shaderProgram pour récupérer la localisation de votre variable définie dans le shader (le nom doit être le même).
6. Lors du rendu, envoyez votre variable définie coté CPU à votre shader (GPU) avec la fonction gl.uniform1f(votreLocalisation,votreVariableCPU)

Attributs et varyings

Les variables de type uniform permettent d'effectuer des modifications globales du maillage qui est dessiné. Mais il est souvent nécessaire d'avoir des attributs qui sont définis par sommets. Par exemple, on aimerait définir une couleur différente sur chacun des sommets du maillage. Pour cela, on va procéder exactement de la même manière que pour l'envoie des attributs positions :

1. dans le fichier cube.js, modifiez le constructeur pour définir et ajouter un tableau de couleurs (il en faut autant que les sommets → 8). Procédez exactement de la même manière que pour this.vertexBuffer (vous pouvez l'appeler this.colorBuffer).
    a. créer le buffer et l'activer createBuffer, bindBuffer
    b. créer un tableau de couleurs (différente pour chacun des sommets)
    c. envoyez le sur le GPU avec bufferData
    d. rappelez vous du nombre / taille des données (numItems / itemSize)
2. dans la fonction « draw », activez le buffer et faites le lien avec le shader
    a. bindBuffer et vertexAttribPointer
    b. doit être fait avant le dessin des buffers actifs
3. dans le vertex shader (fichier cube.html), ajouter un attribut (comme aVertexPosition) du nom de votre choix pour la couleur du sommet courant.
4. Récupérez la localisation de cet attribut dans la fonction « initShaders » -- comme pour la position. Attention : les noms choisis doivent correspondre dans le shader **ET** dans cube.draw().

On dispose désormais d'une couleur par sommet dans le vertex shader. Il faut maintenant faire passer cette couleur au fragment et la rastérisation s'occupera du reste pour interpoler entre les sommets :

1. Dans le vertex shader, définissez une nouvelle variable de type varying à laquelle vous attribuerez la couleur reçue en ce sommet.
2. Dans le fragment shader définissez la même variable (avec le même nom). Il s'agit de la couleur qui vient d'être interpolée entre les sommets. Attribuez la à la couleur du fragment.

Observez le résultat sur des points, des lignes et des triangles. Définissez les différences entre un attribut, une variable uniforme et une variable varying.

Vous enverrez une archive du dossier (à votre noms) contenant les sources (html+js) à romain.vergne@inria.fr. Les sources doivent être commentées et contenir les réponses aux questions. Elles doivent aussi contenir vos noms et des notes sur ce que vous avez compris/pas compris trouvés facile/difficile, etc.