TP : OpengGL 4 - *Tessellation shaders*
Practical Work: OpengGL 4 - *Tessellation shaders*

Jonathan Fabrizio
http://jo.fabrizio.free.fr/

Objectif — Goal

L'objectif de ce T.P. est de programmer les *tessellation shaders*. Pour cela, on simulera des vagues sur une surface (comme un drapeau). On tachera de lui donner un aspect qui rappel celui de la surface de l'eau.

The goal of the practical work is to implement the *tessellation shaders*. To do so, we will simulate waves over a surface (like a flag). We will render the surface as if it were water.

1 Travail préliminaire — Preliminary work

Récupérez le squelette du tp précédent, décompressez l'archive et essayez de comprendre le fonctionnement du programme. Remplacez le lapin par un simple plan :

Get the skeleton from the previous practical work, look at the files in the tarball and try to understand the content. Substitute the rabbit with a simple plan.

```
static const std::vector<GLfloat> vertex_buffer_data
{
  -0.5, 0.0, +0.5,
  +0.5, 0.0, +0.5,
  +0.5, 0.0, -0.5,
  -0.5, 0.0, -0.5
};
```

Vous n'avez pas besoin des normales (la normale est évidente : $(0.0, 1.0, 0.0)$), ni des coordonnées textures (vous pouvez simplifier le programme).

Il faudra modifier la matrice projection pour mieux voir l'objet (le lapin n'aillant pas la même dimension).

You do not need neither the normals (the normal is obvious: $(0.0, 1.0, 0.0)$), nor the texture coordinates (you can simplify the program).
You also have to modify the projection matrix in order to see the object (the rabbit hasn't the same size).

$$\begin{pmatrix} 250.00000 & 0.00000 & 0.00000 & 0.00000 \\ 0.00000 & 250.00000 & 0.00000 & 0.00000 \\ 0.00000 & 0.00000 & -1.02020 & -10.10101 \\ 0.00000 & 0.00000 & -1.00000 & 0.00000 \end{pmatrix}$$

1.1 Des *Tessellation shaders* minimals Minimals *Tessellation shaders*

Modifiez le code pour ajouter un *Tessellation control shader* et un *Tessellation evaluation shader* dans le pipeline. Pour l'instant vos *Tessellation shaders* ne fera que renvoyer ce qu'il reçoit. Ils ne travaillent que sur les coordonnées des points.
Pensez à changer votre `glDrawArrays` par un `glDrawArrays(GL_PATCHES, 0, 4);` et à spécifier la taille du *patch* avec `glPatchParameteri(GL_PATCH_VERTICES, 4)`.

Modify your code to add a *Tessellation control shader* and a *Tessellation evaluation shader*. Until now your *Tessellation shaders* only send what they receive (no transformation). They only manage vertices coordinates.
Do not forget to change `glDrawArrays` by `glDrawArrays(GL_PATCHES, 0, 4);` and to specify the size of the patch with `glPatchParameteri(GL_PATCH_VERTICES, 4)`.

2 Affichage des vagues Display waves

Améliorez votre *Tessellation evaluation shader* pour modifier la position des points dans l'espace afin de faire une vague (avec un sin). Au besoin, augmentez le niveau de tessellation dans le *Tessellation control shader*.
Une fois fait, cumulez plusieurs vagues (disons 3 au moins) dans différentes directions et à différentes fréquences.

Improve your *Tessellation evaluation shader* to modify the position of your vertices in ordre to simulate a wave (with *sin* function). You are allowed to increase the tessellation level in the *Tessellation control shader*.

Pour l'animation, vous pouvez ajouter un timer dans votre code (vos vagues seront paramétrées par `anim_time` dans le *Tessellation evaluation shader*) :

For the animation, you can add a timer in your code (the waves will be parameterized by `anim_time` in the *Tessellation evaluation shader*) :

```
void anim() {
  GLint anim_time_location;
  //glUseProgram(program_id);
  anim_time_location =
    glGetUniformLocation(program_id, "anim_time");
  glUniform1f(anim_time_location, anim_time);
  anim_time += 0.1;
// testez une borne max.
  glutPostRedisplay();
}


void timer(int value) {
  anim();
  glutTimerFunc(33,timer,0);
}

void init_anim() {
  glutTimerFunc(33,timer,0);
}
```

3 Une petite pause — Have a break!

Avant d'aller plus loin :

- Il est parfois difficile de bien apprécier le résultat avec un point de vu fixe. Vous pouvez changer le point de vu en changeant la matrice `model_view_matrix`. Pour comprendre comment cette matrice est formée, vous pouvez vous appuyez sur la documentation de `glu_look_at`. Le mieux serait d'avoir un peu d'interactivité (c.f. `glutMouseWheelFunc`, `glutMouseFunc` et `glutMotionFunc`).
- Il est bien de lire les descriptions des fonctions glsl suivantes : `cross`, `dot`, `clamp`, `min`, `max` et `reflect`.

Before going further:

- It is difficult to judge the result with a fixed point of view. You can change to point of view by modifying the `model_view_matrix` matrix. To understand how this matrix is generated, you can read the documentation of `glu_look_at`. It would be best to have some interactivity (see: `glutMouseWheelFunc`, `glutMouseFunc` and `glutMotionFunc`).
- You should also read the documentation of the following glsl functions: `cross`, `dot`, `clamp`, `min`, `max` et `reflect`.

4 Calcul des normales — Normals computation

Calculez dans le *Tessellation evaluation shader* la normale à la surface en chaque sommet.

Pour rappel, vous avez une surface

Compute the normal of the surface for every vertex in the *Tessellation evaluation shader*

$$s(x,z) = \begin{pmatrix} x \\ f(x,z) \\ z \end{pmatrix}$$

La normale peut être déduite par :

The normal is deduced from:

$$\left(\frac{\partial s(x,z)}{\partial x}\right) \wedge \left(\frac{\partial s(x,z)}{\partial z}\right)$$

5 Calcul de la couleur

Dans un premier temps, à l'aide de la normale, calculez l'apport du diffus à votre surface. Dans un deuxième temps, toujours à l'aide de la normale, calculez l'apport du spéculaire à votre surface.

Note : vous placerez une lumière blanche (omnidirectionnelle ou à l'infini) et vous donnerez une couleur diffuse bleu à votre surface.

In a first step, compute the diffusion part according to the normals. In a second step, compute the specular part.
Note : Use a white light (omnidir. or at infinity) and the surface will be blue.