POLYTECHNIQUE MONTRÉAL

LE GÉNIE EN PREMIÈRE CLASSE

INF2705 Infographie

Travail pratique 4

Tessellation et rétroaction

Département de génie informatique et génie logiciel
Polytechnique Montréal
Tristan Rioux, Automne 2023

Table des matières

1 Introduction

Ce document décrit les exigences du TP4 « *Tessellation et rétroaction* » (Automne 2023) du cours INF2705 Infographie.

1.1 But

Le but des travaux pratiques est de permettre à l'étudiant de directement appliquer les notions vues en classe.

1.2 Portée

Chaque travail pratique permet à l'étudiant d'aborder un sujet spécifique.

1.3 Remise

Faites la commande « `make remise` » ou exécutez/cliquez sur « `remise.bat` » afin de créer l'archive « **INF2705_remise_TPn.zip** » (ou .7z, .rar, .tar) que vous déposerez ensuite dans Moodle. (Moodle ajoute automatiquement vos matricules ou le numéro de votre groupe au nom du fichier remis.)

Ce fichier `zip` contient tout le code source du TP (`makefile`, `*.h`, `*.cpp`, `*.glsl`, `*.txt`).

2 Description globale

2.1 But

Le but de TP est de permettre à l'étudiant de mettre en pratique les concepts de tessellation et de calcul sur carte graphique. Il sera en mesure d'effectuer la tessellation pour définir plus de détails sur certains éléments dans la scène. Il sera aussi capable d'envoyer une partie des calculs sur la carte graphique pour mettre à jour une grande quantité de données. Il sera en mesure d'implémenter un système de particules pour un effet spécial de feu. Le travail fera l'utilisation des fonctions d'OpenGL `mix`, `smoothstep` et des shaders de tessellation, géométrie et rétroaction.

2.2 Travail demandé

Comme mentionner dans le tp3, l'illumination de la scène 3d a été implémenté. Il est possible de modifier le modèle d'illumination en appuyant sur R. Le matériel n'a pas été modifié, mais il aurait été possible d'augmenter davantage le réalisme de la scène avec un matériel adapté aux surfaces.

FIGURE 1 – Scène illuminée.

Un fichier de scènes pour le tp4 a été préparé pour vous, de même que les fichiers de shaders.

Pour le débuggage, il est encore recommandé d'utiliser l'application `RenderDoc`.

Vous avez le droit de modifier les constantes comme bon vous sembles, tant que le résultat est semblable.

Partie 1 : Tessellation

Une des applications du shader de tessellation est la génération de terrains à partir d'une image. Il est pratique d'utiliser le shader de tessellation au lieu de précalculer les données dans un vbo, puisqu'il est possible de changer dynamiquement le niveau de détail du terrain et de traiter une grande quantité d'informations sans avoir à passer par la mémoire.

Dans le cadre de ce tp, nous allons utiliser le shader de tessellation de contrôle et d'évaluation pour implémenter le niveau de détails variables. De plus, on utilisera un shader de géométrie pour faire l'affichage du maillage du terrain.

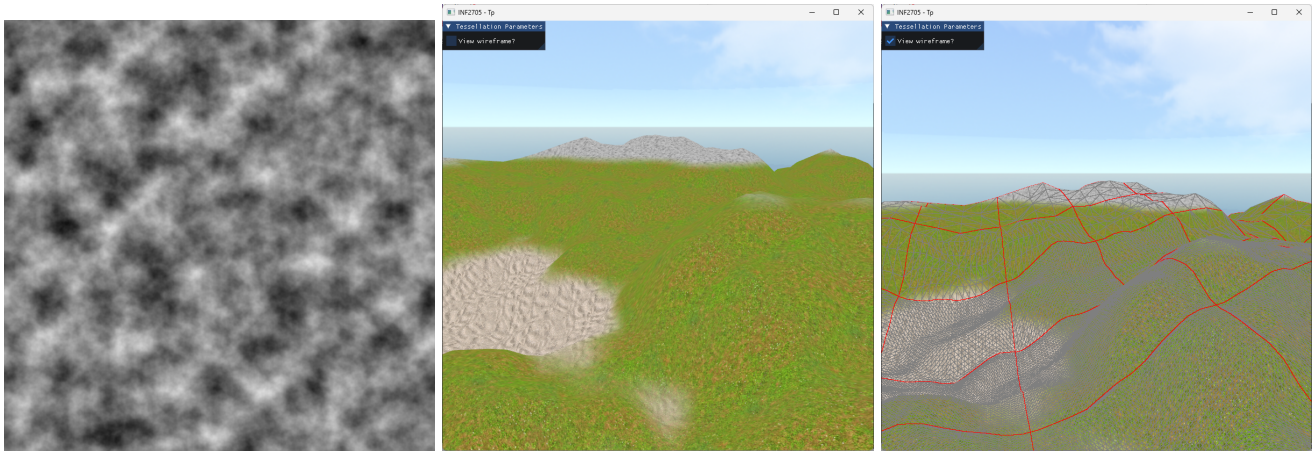

FIGURE 2 – Tessellation.

Partie 1a : Scène de tessellation

Une nouvelle scène a été préparée pour faire la tessellation. La caméra entre automatiquement dans le mode première personne, et il est possible de monter et descendre avec Q et E.

Le shader est déjà chargé pour vous, de même que les textures et les uniforms. On utilisera le shader tessellation, les 4 textures heightmap, ground, sand et snow. Le modèle `tesselationPlane` est déjà construit pour vous dans `resources.cpp`; vous pouvez le modifié comme bon vous semble pour l'agrandir ou avoir qu'une seule patch, etc (optionnel). Il est constitué de plusieurs patches sous le format de quadrilatères.

Vous devez finir l'initialisation de la tessellation du côté CPU et dessiner le modèle.

Vous pouvez vous aider de `glPolygonMode` pour debugger, mais il ne devrait pas être utilisé dans votre remise.

Partie 1b : Shader de vertex et de tessellation de contrôle

Pour le vertex shader, on ne fait qu'envoyer la position non modifiée.

Pour le shader de tessellation de contrôle, vous devez faire définir le niveau de tessellation par rapport à la distance de la caméra.

Pour ce faire, on calcule la longueur du vecteur de la position du milieu de chaque arête de la patch dans le référentiel de la vue.

On fera une interpolation linéaire entre le facteur MIN_TESS et MAX_TESS (utiliser mix). Le facteur de mix sera la distance normalisée dans l'intervalle $[MIN\_DIST, MAX\_DIST]$ et contraint dans l'intervalle $[0, 1]$.

Le facteur de tessellation de chaque arête est assigné au côté approprié. Pour les facteurs internes, on prendra le maximum des facteurs externes sur l'axehorizontal ou vertical.

Au final, on devrait avoir plus de détails de près et moins de détails de loin. Le calcul doit être effectué qu'une seule fois par patch.

Aidez vous de la figure 3 pour identifier les bonnes arêtes.

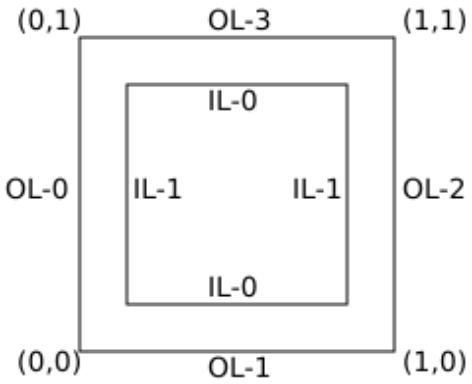

FIGURE 3 – Détail de la patch durant le shader de tessellation de contrôle.

Partie 1c : Shader de tessellation d'évaluation

Dans le shader de tessellation d'évaluation, on utilise une simple interpolation 2D pour trouver les positions des vertices à partir des 4 positions en entrées.

On pourra retrouver les coordonnées de textures en utilisant la position interpolée et en la convertissant dans un intervalle $[0, 1]$ avec la taille du plan au complet (prenez note que celui-ci est centré en $(0, 0)$). De plus, on divisera les coordonnées par 4 pour étirer la texture sur le plan et réduire la fréquence des montagnes.

Ces coordonnées nous permettront d'extraire un texel de l'image contenant les données de hauteur. Puisque c'est un bitmap de noir et blanc, seulement une composante est active (même chose qu'au tp3 avec la texture spéculaire).

Pour connaitre la hauteur du vertex, on convertira la donnée de hauteur $[0, 1]$ en $[-32, 32]$. L'attribut en sortie aura la valeur normalisée.

L'attribut de sortie patchDistance est un vecteur où chaque composante est la distance par rapport à un côté de la patch (coordonnées barycentriques). On peut facilement l'obtenir avec les coordonnées de $gl\_TessCoord$ et $1 - gl\_TessCoord$.

Pour l'attribut de coordonnées de texture, on utilise directement $gl\_TessCoord$ avec un multiplicateur de 2 pour augmenter la répétition dans la texture.

Ne pas oublier de faire les transformations matricielles dans celui-ci, et non dans le nuanceur de sommets.

Partie 1d : Shader de géométrie

Pour faire l'affichage en maillage, nous aurons besoin des coordonnées barycentriques. Celles-ci sont facilement obtenables dans le shader de géométrie. Ce dernier n'aura besoin que de générer les coordonnées barycentriques pour chaque triangle et de passer les attributs en entrée aux attributs de sortie.

Partie 1e : Shader de fragment

Dans le nuanceur de fragments, on utilisera la hauteur pour choisir la texture à utiliser. L'utilisation de mix et de smoothstep pour faire le mélange entre les deux couches est nécessaire.

- Entre 0.0 et 0.3, on utilisera le sable.
- Entre 0.3 et 0.35, on utilisera le sable et le gazon (mélange des textures).
- Entre 0.35 et 0.6, on utilisera le gazon.
- Entre 0.6 et 0.65, on utilisera le gazon et la neige (mélange des textures).
- Entre 0.65 et 1.0, on utilisera la neige.

Pour l'affichage du maillage, on fera un mix avec la couleur finale des textures et la couleur du maillage. Le facteur sera calculable par `edgeFactor`. Même chose pour le contour de la patch. L'affichage du maillage doit être activable avec `viewWireframe`.

Partie 2 : Particules

Les effets spéciaux de particules sont souvent utilisé en infographie. Cependant le nombre de particules est souvent une limitation pour ceux-ci. On peut remédier à ce problème en utilisant le nuanceur de rétroaction, qui permet de faire la mise à jour des particules sur le GPU, ce qui augmente grandement le parallélisme de l'application.

On demande de faire un système de particule simple pour faire un effet spécial de feu. Vous pouvez modifier les paramètres de mise à jour comme bon vous semble et customiser votre système de particules, tant que vous modifiez la majorité des paramètres avec des animations par rapport au temps (mix, smoothstep).

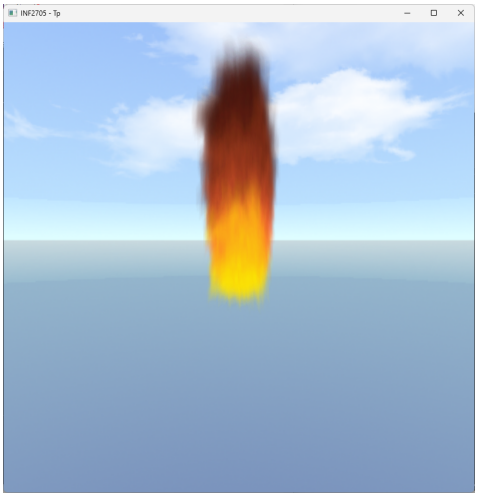

FIGURE 4 – Effet de particules de feu réalisé par rétroaction.

Partie 2a : Scène de particules

Une nouvelle scène a été préparée pour faire l'effet spécial.

Pour vous faciliter la tâche, on utilisera les vao, vbo et tfo directement dans la classe. Dans le constructeur, on fera l'initialisation des différents objets graphiques. N'oubliez pas d'activer les attributs du vao. Pour le `glTransformFeedbackVaryings`, utiliser la méthode `setTransformFeedbackVaryings` avant de faire le link (`resources.cpp`).

Vous devez implémenter les deux parties de la méthode render : la mise à jour des particules et le dessin des particules. Notez que le ciel sera dessineé en premier, principalement pour que le blending des particules se produise correctement.

Pour la mise à jour, vous devez faire la gestion du shader de rétroaction avec la technique de double buffer vue en classe. Pour le dessin des particules, on activera le blending classique ($alpha_{src}$ et $1 - alpha_{src}$) et désactivera l'écriture du tampon de profondeur.

Partie 2b : Shader de rétroaction

Le shader de rétroaction fait le calcul de la physique des particules et gère leur création. Lorsque leur temps de vie est en dessous de 0.0, on la réutilise et en réinitialise une nouvelle. Sinon on fait la mise à jour habituelle.

Pour la création, la particule aura ses propriétés :

- La position initiale est aléatoire dans un cercle de rayon 0.1 à $y = 0$.
- Les particules sont définis dans un cône. On trouve la direction de la vélocité avec un deuxième cercle de rayon 0.5 à $y = 5.0$.
- Le module de la vélocité est dans l'intervalle $[0.5, 0.6]$.
- Le temps de vie maximale est entre $[1.7, 2.0]$.
- La couleur initiale est la couleur jaune avec un alpha de 0.0.
- La taille initiale est écrasé d'un facteur 2 en x, et intact en y.

Pour la mise à jour, la particule aura ses propriétés :

- La position est modifié avec la méthode d'Euler ($position+ = vitesse * dt$).
- La vitesse est aussi modifié avec la méthode d'Euler, mais avec une accélération constante de 0.1 en y.
- Le temps de vie est réduit de la différence de temps dt.
- La couleur est jaune entre 0.0 et 0.25 du temps de vie normalisé.
- La couleur change du jaune au orange entre 0.25 et 0.3 du temps de vie normalisé.
- La couleur est orange entre 0.3 et 0.5 du temps de vie normalisé.
- La couleur change du orange au rouge foncé entre 0.5 et 1.0 du temps de vie normalisé.
- Le alpha change passe de 0.0 à 0.1, puis de 0.1 à 0.0. Utiliser `smoothstep() * 1-smoothstep()` pour produire cette courbe. Les intervalles sont $[0.0, 0.2]$, puis $[0.8, 1.0]$ respectivement.
- La taille passe d'un facteur variable de 1 à 1.5 fois la taille originale entre 0.5 et 1.0 du temps de vie normalisé.

Le temps de vie normalisé correspond au ratio de vie restante sur le temps de vie maximale. Utiliser le ratio inverse ($1 - tempsNormalise$) peut s'avérer pratique.

Partie 2c : Shader de dessin des particules

Il y a trois shaders à implémenter : vertex, géométrie et fragment.

Le shader de vertex doit calculer la position dans le référencielle de la vue.

Le shader de géométrie doit transformer les points en carré centré à la position. La taille des arêtes seront de longueur `size` (reçu dans les attributs). Les coordonnées de textures sont aussi générées par le shader à partir de la position des points du carré généré. Ne pas oublier la matrice de projection.

Le shader de fragments doit discard lorsque l'alpha est plus petit que 0.05. La couleur finale est un mélange de la couleur et du texel de la texture.

3 Exigences

3.1 Exigences fonctionnelles

Partie 1 :

- E1. La tessellation des patches fonctionne et le niveau de tessellation varie selon la distance de la caméra. [9 pts]
- E2. Les calculs dans le shader de tessellation de contrôle sont fait qu'une seule fois. [1 pt]
- E3. La fonction `interpole` effectue l'interpolation bilinéaire sur les vecteurs en entrée. [1 pt]
- E4. Les différentes textures sont mappées correctement (texCoords) et les facteurs d'atténuation sont appliqués. [3 pts]
- E5. La hauteur de chaque points est dans l'intervalle $[-32, 32]$ dans le référenciel du monde. [2 pts]
- E6. Les coordonnées barycentriques sont calculées correctement et permettent l'affichage en fil de fer des triangles. [3 pts]
- E7. Les coordonnées barycentriques de la patch sont calculées correctement et permettent l'affichage du contour des patches. [3 pts]
- E8. La texture du terrain dépend de la hauteur et les textures sont mixées aux jonctions entre elles. [3 pts]

Partie 2 :

- E9. Le mode d'utilisation des vbos correspond à un mode utile pour **toutes ses utilisations**. [1 pt]
- E10. La mémoire des objets OpenGL est nettoyée correctement à la destruction de l'objet. [1 pt]
- E11. La configuration de la rétroaction dans la méthode `render` permet de modifier les points. [2 pts]
- E12. La configuration du dessin des particules dans la méthode `render` permet de voir les particules. [1 pt]
- E13. Le blending est activé et les particules n'écrivent pas le tampon de profondeur. [1 pt]
- E14. Le shader de rétroaction fait l'initialisation des particules avec les bonnes valeurs par défaut. [2 pts]
- E15. La mise à jour de la position et de la vitesse est fait selon la méthode d'Euler. [1 pt]
- E16. La couleur débute en jaune et change en orange, puis en rouge foncé selon le temps de vie. Le changement est fait graduellement. [3 pts]
- E17. L'opacité de la particule fait un effet de fade in et fade out en début et fin de vie. [2 pts]
- E18. La taille augmente graduellement selon le temps de vie. [2 pts]
- E19. Le shader de géométrie convertit les points en quads texturés de tailles variables orientés vers l'observateur. [3 pts]
- E20. Les fragments transparents sont discard. [1 pt]

Bonus :

- E21. Modifier l'interface de `ImGui` pour la scène de tessellation. Modifier les constantes du shader pour être modifiable dans le menu. [2 pts]
- E22. Ajouter une interface de `ImGui` pour la scène de rétroaction. Modifier les constantes du shader pour être modifiable dans le menu. [2 pts]
- E23. Utiliser des uniform buffers pour envoyer les informations aux shaders. [3 pts]

3.2 Exigences non fonctionnelles

De façon générale, le code que vous ajouterez sera de bonne qualité. Évitez les énoncés superflus (qui montrent que vous ne comprenez pas bien ce que vous faites !), les commentaires erronés ou simplement absents, les mauvaises indentations, etc. [5 pts]

ANNEXES

A Liste des commandes

Touche	**Description**
`ESC`	Quitter l'application
`w`	Mouvement avant
`s`	Mouvement arrière
`a`	Mouvement gauche
`d`	Mouvement droit
`q`	Mouvement bas (scène de tessellation)
`e`	Mouvement haut (scène de tessellation)
`scroll`	Alterner entre fps/tps
`souris`	Changer l'orientation de la camera
`espace`	Activer/désactiver la souris
`t`	Changer de scène
`r`	Changer de modèle de lumière dans la scène 3d