Université de Strasbourg

Projet d'Infographie 3D

Département d'Informatique
CMI Image 1ère année

Ce projet est divisé en trois parties pouvant être traitées indépendamment les unes des autres. Dans une quatrième partie, on pourra réunir le résultat des trois premières parties. L'objectif final est de créer un paysage montagneux avec un lac et de peupler ce paysage d'objets ou de personnages. Une grande densité au niveau de l'eau et plus clairsemé au fur et à mesure qu'on monte. Le barême est indicatif.

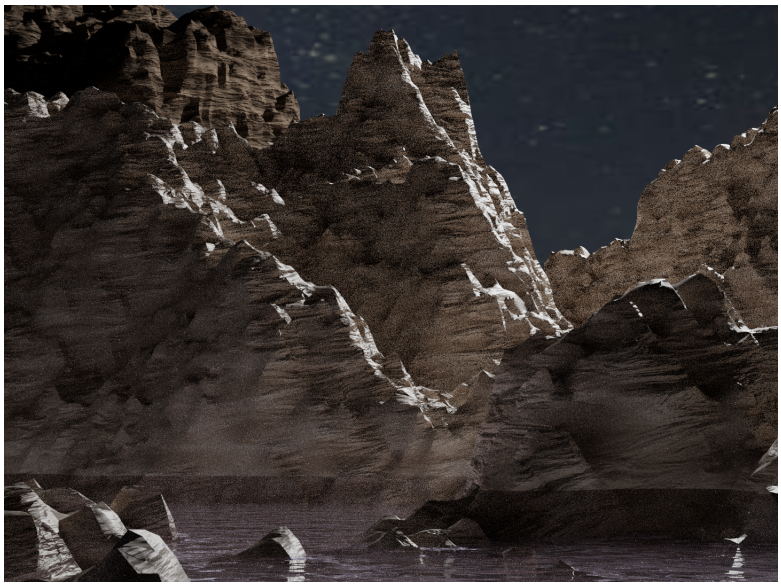

FIGURE 1 – Une montagne générée par l'algorithme décrit ci-dessous

1 Construction d'un terrain montagneux (7 points)

1.1 Modélisation

Vous écrirez les fonctions de cette section dans un fichier appelé `paysage.py`.

1. Écrire une fonction `hauteursAleatoires` qui admet deux arguments : un objet polygonal `objet` et un flottant `hauteur`. Cette fonction ne retourne aucune valeur. Elle parcourt tous les sommets de l'objet donné en paramètre et ajoute à la hauteur (composante z) de chaque point une valeur aléatoirement choisie entre `-hauteur` et `hauteur`.
2. Écrire une fonction `subdiviseAretes` qui admet un argument : un objet polygonal `objet`. Cette fonction ne retourne aucune valeur, mais elle subdivise chaque face de cet objet. Cette subdivision consiste à insérer un sommet au milieu de chaque arête et de chaque face. Ainsi chaque face est transformée en quatre faces coplanaires.
3. En utilisant les fonctions ci-dessus, écrire une fonction `creerMontagne` admettant 4 paramètres : une chaîne de caractères `nom`, représentant le nom de l'objet à créer, un nombre `largeur`, un nombre `hauteur` et un entier `Nsubdiv`. Étant donnés ces trois nombres, cette fonction devra créer une montagne de largeur `largeur` de hauteur approximative `hauteur` et dont la résolution est déterminée par `Nsubdiv`. La valeur de retour de cette fonction devra être l'objet créé. L'algorithme à implémenter est le suivant :

   L'objet de départ est un carré (plan) dont le nom est la valeur du paramètre `nom`. Ce carré devra être de côté `largeur` avec 4 sommets. À `Nsubdiv` reprises, il faudra faire les opérations suivantes :

   (a) Parcourir tous les sommets de l'objet et déplacer ces sommets de façon aléatoire le long de l'axe z. Le déplacement est un nombre choisi aléatoirement entre `-hauteur` et `hauteur`.

   (b) Subdiviser chaque face de l'objet.

   (c) Diminuer `hauteur`.

Comment allez-vous diminuer `hauteur` ? En soustrayant une valeur à chaque itération ? en divisant par une valeur ? Autrement ? L'algorithme est déroulé dans les figures qui suivent.

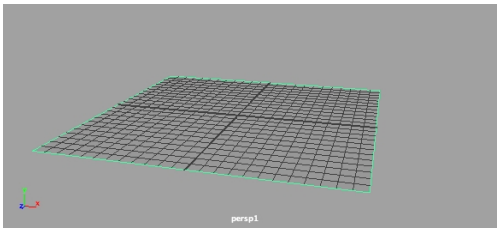

FIGURE 2 – Départ : un plan de 4 sommets

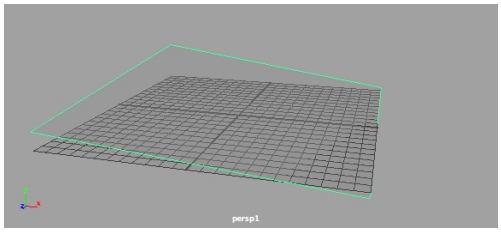

FIGURE 3 – Déplacement aléatoire d de chaque sommet (avec $d \in [-hauteur, hauteur]$)

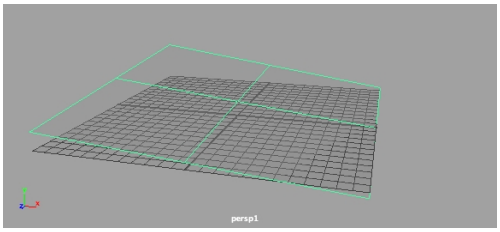

FIGURE 4 – Subdivision

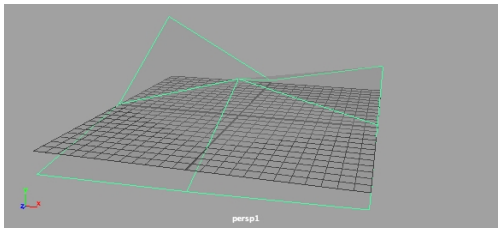

FIGURE 5 – Diminuer $hauteur$. Déplacement aléatoire d de chaque sommet (avec $d \in [-hauteur, hauteur]$).

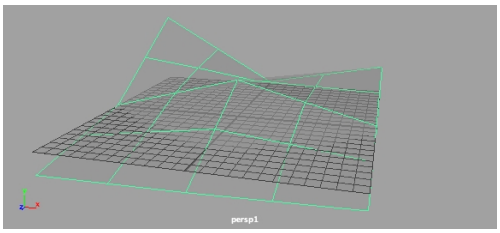

FIGURE 6 – Subdivision

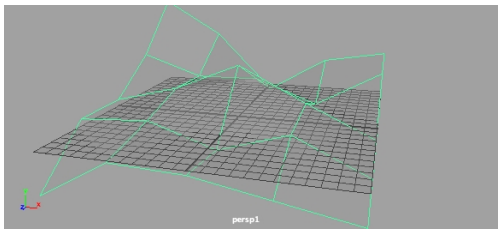

FIGURE 7 – Diminuer encore $hauteur$ et déplacement d de chaque sommet (avec $d \in [-hauteur, hauteur]$).

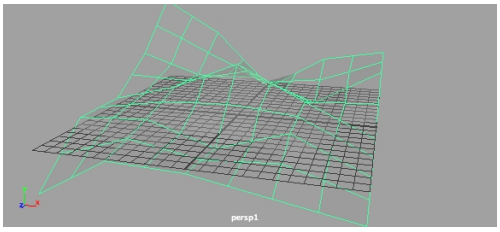

FIGURE 8 – Subdivision

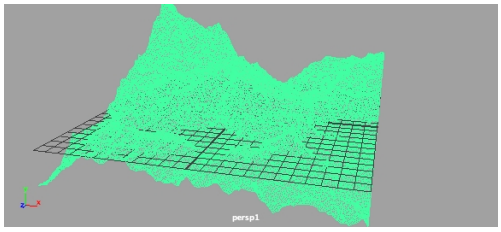

FIGURE 9 – Résultat après 8 itérations en partant de $h = 7$

4. Que se passe-t-il lorsque vous lancez cet algorithme plusieurs fois ? Est-ce que vous obtenez la même forme de montagne ? Faites de façon à ce que, lorsque vous aurez obtenu une forme de montagne satisfaisante, vous puissiez, à chaque fois, obtenir la même forme. Indication : à quoi sert la fonction `seed` du module `random` ?

5. Écrire une fonction `creerLac`, admettant un paramètre un flottant `largeur`. Cette fonction devra créer un plan ayant une face dont le nom sera `lac`. Ce plan représentera la surface d'un lac. La valeur de retour de cette fonction sera l'objet ainsi créé. Il ne sera pas nécessaire de subdiviser le plan du lac pour avoir des vagues.
6. De la même façon, écrire une fonction `creerCiel` prenant en argument un flottant `rayon`. Cette fonction devra créer, une sphère appelée `ciel` de rayon supérieur ou égal à 100 représentant le ciel. La valeur de retour sera l'objet effectivement créé.
7. À présent, on cherche à composer le paysage. Vous appelerez les fonctions `creerLac` et `creerCiel` une seule fois et vous laisserez ces objets en $(0, 0, 0)$. Par contre, je vous demande d'appeler `creerMontagne` plusieurs fois, avec des formes différentes, et de placer les différents objets obtenus à des positions, orientations et échelles qui, pour vous, produisent un paysage intéressant. Créer une caméra pour matérialiser le point de vue que vous avez choisi. Créer également une ou plusieurs sources de lumière qui mettent le mieux en valeur le relief de votre paysage. Il n'est absolument nécessaire que vous assembliez ces différents massifs pour qu'ils ne forment qu'un objet. Vous pouvez très bien les laisser bien séparés. Mais, vus depuis votre caméra, ils doivent sembler former un paysage cohérent. De même, le bord du carré représentant le lac ne doit pas être visible depuis la caméra.
8. Sauvegarder la scène dans un fichier appelé `montagne.blend` et noter la place qu'elle occupe sur le disque. Un des grands intérêts de la génération des modèles par script (appelée *modélisation procédurale*) est qu'elle évite de stocker de lourds modèles sur le disque. Il suffit de les régénérer à chaque fois à partir d'un script. Et le volume d'un script sur un disque est minime (quelques kilobytes). Je vous demande de noter tous les réglages et les opérations que vous avez faites pour que tout ce qui précède puisse être régénéré par script. Si vous réussissez cette question, il ne sera pas nécessaire de me rendre la scène `montagne.blend` mais seulement le script qui a permis de la générer.

1.2 Shading

Créer manuellement (et non par script) des matériaux et des textures pour les montagnes, le lac et le ciel. Je vous demande de donner à vos matériaux des noms porteurs de sens (par exemple `mat_ciel` ou `mat_montagne1`).

1. Pour le ciel, concevoir un matériau et une texture de façon à ce que l'aspect du ciel ne dépende pas de l'intensité ou de la position des sources de lumière ;
2. Pour l'eau, concevoir un matériau transparent avec de la réfraction et de la réflexion. Ne pas utiliser de texture de couleur, mais seulement une texture procédurale de type *Noise* pour créer des vaguelettes. Je vous demande de régler la fréquence et l'amplitude de ce bruit pour être en adéquation avec l'échelle de la scène et, en particulier, avec l'échelle des personnages ou objets que vous y aurez placés (section 4).
3. Pour les montagnes, concevoir une texture de couleur et une texture de normal (bas-relief). La texture de couleur peut être une texture explicite (une image) mais la texture de relief doit être une texture procédurale de noise permettant de reproduire des strates comme sur la figure 1.
4. Il n'est pas du tout impossible de générer tous les matériaux, textures et graphes de rendu par script, mais cette tâche peut s'avérer assez fastidieuse. C'est la raison pour laquelle je vous propose de les conserver dans la scène. Donc une fois que vous êtes satisfaits de vos matériaux, sauvegarder la scène sous `matMontagne.blend` et supprimer toutes les montagnes, le lac et le ciel. Ainsi, la scène `matMontagne.blend` ne contient que les matériaux que vous avez créés.
5. Vous avez déjà un script qui permet de créer la montagne, le lac, le ciel, la caméra et la ou les sources de lumière. Ajouter une fonction `raccrocheMats` à ce script pour que, s'il est lancé dans la scène `matMontagne.blend`, il puisse raccrocher ces matériaux sur les montagnes, le ciel et le lac. Ainsi, vous pouvez créer `montagne.blend` (scène imposante) à partir de `matMontagne.blend` (scène minuscule).

2 Création de personnages ou d'objets à partir de photos (2 points)

Les opérations dans cette section sont à réaliser à la main et non par script. Dans une scène `plaques.blend`, créer 10 plans à une face. Nous les appellerons des *plaques* par la suite. Attribuer à ces plaques un matériau transparent

et une texture de transparence ainsi qu'une texture de couleur de façon à représenter des arbres, des personnages ou tout autre objet reconnaissable. Vous pouvez choisir les textures du répertoire *detoures* dans `textures.tgz` que vous avez déjà utilisées en TP. Vous pouvez également en trouver d'autres si le cœur vous en dit. Inclure toutes les images qui vous servent de texture dans votre répertoire de projet.

3 Répartition d'objets simples sur un polygone (7 points)

Dans une scène `repart.blend`, soit un polygone P ayant beaucoup de sommets à des hauteurs différentes. Pour les figures de ce document, j'ai choisi un tore ayant 45000 sommets. On cherche à déterminer N positions aléatoirement sur ce polygone entre l'altitude z_{min} et z_{max}. Dans un premier temps, on pourra répartir ces positions de façon uniforme (cf figure 10). Dans un deuxième temps, on demande à ce qu'au niveau z_{min} la densité de points soit maximale. Au delà de z_{max}, la densité des points doit être nulle. Entre les deux altitudes, la densité devra décroître de façon exponentielle (cf figure 11). Je vous demande d'écrire ces fonctions dans un fichier appelé `repart.py`.

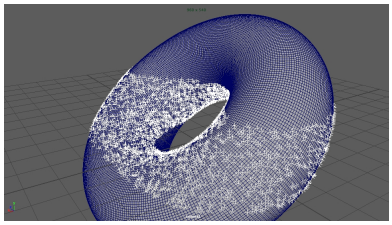

FIGURE 10 – 5000 positions répartis uniformément entre $z_{min} = 0$ et $z_{max} = 2$

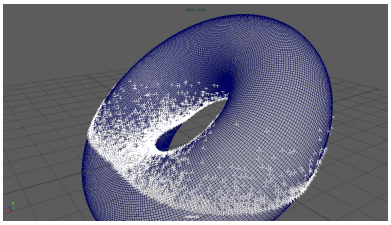

FIGURE 11 – 5000 positions avec une densité décroissant exponentiellement entre z_{min} et z_{max}

3.1 Répartition uniforme

1. Écrire une fonction `repartitionUniforme` qui admet quatre paramètres : `objet`, le nom d'un objet polygonal, un entier positif `N` et deux nombres `zmin` et `zmax` (avec `zmin < zmax`). Cette fonction retourne une liste de `N` triplets représentant des sommets choisis au hasard sur la surface de `objet`. La hauteur de ces positions devra être comprise entre `zmin` et `zmax`. Proposition d'algorithme : choisir des sommets au hasard sur le polygone jusqu'à ce que la position obtenue soit comprise entre `zmin` et `zmax`.
2. Modifier cette fonction pour que chaque sommet ne soit choisi qu'une seule fois (si ce n'est pas le cas déjà). Si tous les sommets compris entre `zmin` et `zmax` ont déjà été choisis, alors aucun triplet supplémentaire n'est généré. Il faudra alors afficher un message pour informer l'utilisateur que la tâche demandée n'est pas réalisable et lui indiquer le nombre de sommets choisis. Un algorithme de détection de doublons a une complexité qui croît en $o(n^2)$. Tâchez de trouver un algorithme dont la complexité croît en $o(n)$. (n étant le nombre de positions à trouver). Si vous deviez représenter toutes ces positions, vous devriez obtenir une figure semblable à la figure 10.

3.2 Répartition à densité décroissante

Dans cette section, nous souhaitons obtenir une plus forte densité en z_{min}, une densité nulle en z_{max} et une décroissance exponentielle entre les deux. Aurement dit, la densité d'individus doit pouvoir être exprimée sous la forme :

$$densite(z) = Ae^{-z/z_0} \tag{1}$$

où A et z_0 sont des nombres à déterminer. z_0 détermine la rapidité de la décroissance. Pour une faible valeur de z_0, la décroissance est rapide, alors que pour de grandes valeurs, elle est plus progressive. La constante A règle le nombre

total d'individus. Si vous avez des idées d'algorithmes aléatoires et qui produisent de tels résultats, sentez-vous libre de les utiliser.

Ce qui suit est une proposition d'algorithme. En principe, une densité est exprimée en nombre d'individus par unité de surface. Ici, en supposant que chaque face du polygone a, à peu près, la même surface, on mesurera la densité en nombre d'individus par sommet. On s'intéresse à des situations où il y a moins d'individus que de sommets, donc, la plupart du temps, la densité d en un point sera inférieure à 1. Une densité $d < 1$ en un sommet P signifie que ce sommet a une probabilité d de porter un individu. Ainsi, pour décider si ce sommet doit porter un individu, il suffira de tirer à pile ou face avec une probabilité d d'avoir *pile*. Si on obtient effectivement *pile*, alors le sommet est peuplé. Sinon, il ne l'est pas.

1. Écrire une fonction `pileOuFaceProbaP` qui admet un paramètre `p` compris entre 0 et 1. Elle retourne aléatoirement `True` ou `False` avec une probabilité `p` d'avoir `True`. Autrement dit :
   — pour `p=0`, la fonction retourne toujours `False` ;
   — pour `p=1`, la fonction retourne toujours `True` ;
   — pour `p=0.5`, la fonction retourne `True` ou `False` avec la même probabilité ;
   — pour `p=0.8`, la fonction retourne `True` avec une probabilité de 80% et `False` avec une probabilité de 20%.

   Ces valeurs sont des exemples. Il n'est pas du tout nécessaire de distinguer ces 4 cas séparément dans votre fonction. Pour tout vous dire, vous avez déjà quasiment écrit cette fonction dans le troisième TD d'*Algorithmique et Programmation 1*.
2. Il reste à définir la densité de population à chaque sommet. Nous l'avions mentionné plus haut : z_0 spécifie la rapidité de la progression. On décide que la densité en z_{min} doit être 100 fois supérieure à la densité en z_{max}. En déduire la valeur de z_0. Je rappelle que, pour tout x et y réels, $e^x/e^y = e^{x-y}$ et que, par ailleurs, $y = e^x$ si et seulement si $x = ln(y)$. Écrire une fonction `coefZ0` qui, à partir de deux paramètres `zmin` et `zmax` retourne la valeur de z_0.
3. On sait aussi que, si on fait la somme de toutes les densités à tous les sommets du polygone compris entre z_{min} et z_{max}, on devrait obtenir la population totale N.

$$\sum_i densite(y_i) = \sum_i Ae^{-z_i/z_0} = A\sum_i e^{-z_i/z_0} = N \tag{2}$$

   Il s'ensuit que

$$A = \frac{N}{\sum_i e^{-z_i/z_0}} \tag{3}$$

   Écrire une fonction `coefA` qui admet cinq paramètres : `objet`, représentant le nom de l'objet polygonal à peupler, un entier `N` représentant le nombre total d'individus qu'on souhaite avoir, et enfin, `z0`, `zmin` et `zmax` trois nombres (avec `zmax` > `zmin`). La valeur de retour de cette fonction devra être le coefficient A donné par la relation (3). Pour cela, cette fonction devra parcourir tous les sommets du polygone `objet`, calculer la hauteur `zi` de chaque sommet, en déduire la valeur de e^{-z_i/z_0} et faire la somme de toutes ces valeurs d'exponentielle pour tous les sommets. Ensuite, utiliser la relation (3) pour calculer le coefficient A et retourner sa valeur.
4. À présent, nous connaissons la densité de population pour chaque sommet du polygone. Cette densité est exprimée par la relation (1). Écrire une fonction `repartitionNonUniforme` qui admet 6 paramètres : `objet`, le polygone, `N` le nombre de positions à trouver sur ce polygone, `coef_A`, `coef_z0`, `zmin` et `zmax` qui permettent de calculer la densité d'individu pour chaque valeur de z. Cette fonction devra retourner une liste de `N` triplets représentant des positions de sommets de `objet`, dont la hauteur est comprise entre `zmin` et `zmax` et dont la densité décroît exponentiellement. Pour cela, cette fonction parcourt tous les points de l'objet. Pour chaque point, elle calcule la densité d (relation (1)). Elle utilise la fonction `pileOuFaceProbaP` avec une probabilité d. Si celle-ci retourne `True`, alors on ajoute la position de ce sommet à la liste. Si vos calculs sont justes, alors le nombre total d'individus obtenus devrait être approximativement égal à N.
5. Enfin écrire une fonction `peuplement` qui admet deux paramètres : `positions`, et `plaques`. Le premier est une liste de triplets représentant une liste de positions, comme celles obtenues dans les questions précédentes. Le second est une liste d'objets polygonaux dans la scène. Si vous avez fini la partie 2, cette liste pourra

être simplement la liste de vos plaques. Sinon, vous pourrez simplement créer une dizaine de plans à une face et les mettre dans une liste. Cette fonction devra parcourir la liste des positions et, pour chaque position p, choisir aléatoirement un objet dans `plaques`, le dupliquer et le placer la copie à la position p. Je vous demande de tester cette fonction pour un nombre de positions p bien supérieur au nombre de plaques.

4 Synthèse (4 points)

Si vous avez réalisé les trois volets de ce projet, vous pouvez les réunir en un seul script `projet.py` et une seule scène `projet.blend` contenant : les 10 plaques texturées représentant des objets ou des personnages ainsi que les matériaux des montagnes, du lac et du ciel. Le script `projet.py` crée la caméra et les sources de lumière, le ciel, le lac et les montagnes et y attache les matériaux, comme dans la section 1, puis, comme dans la section 3, elle calcule `N` positions réparties entre $z_{min} = 0$ (le niveau du lac) et $z_{max} \approx H_{max}/2$. Pour chaque position `p` de cette liste, le script choisira une plaque au hasard, la copiera et la placera au point `p` face à la camera. La figure 12 contient 5000 plaques. Je vous demande d'en placer au moins plusieurs dizaines.

À peu près la moitié des points de cette partie est dédiée aux réglages globales : la position que vous avez choisie pour la caméra (plutôt que la position par défaut), le nombre et la position des lumières qui doivent permettre de montrer le relief de la montagne et les personnages et enfin, le choix des échelles des textures. Nous avons choisi de modéliser le relief de l'eau par une texture de noise. Il s'agit donc d'une mer calme. Donc la taille d'une vaguelette ne peut pas être la même que celle d'un arbre ou celle d'une petite feuille.

Si vous avez fini cette partie, vous pouvez rendre seulement la scène `projet.blend` et `projet.py`.

5 Consignes

Avant le jeudi 21 décembre 2023 à 23h59, chacun de vous devra mettre dans le dépôt *Moodle* appelé *Rendu Projet* une archive (.zip ou .tgz) contenant :

- Toutes les textures que vous avez utilisées dans votre projet ;
- Une ou plusieurs images correspondant au rendu final de votre projet.
- (Si vous avez fini la partie 4) : la scène `projet.blend` et le script `projet.py` ;
- (Si ce n'est pas le cas) : les scènes (`matMontagne.blend`, `plaques.blend` et `repart.blend`) et les scripts (`paysage.py` et `repart.py`) correspondant aux parties que vous avez abordées.

Les rendus en retard ne sont pas interdits, mais chaque jour de retard diminuera la note de 3 points. Le vendredi 22 décembre 2023 aura lieu la dernière séance d'infographie 3D du semestre. Chacun de vous devra me présenter son travail de façon individuelle pendant une soutenance de 7 minutes. Pendant cette séance, je pourrai éventuellement vous demander de modifier certains aspects de votre projet. Je vous demande de réserver votre créneau de rendez-vous sur *Moodle (Prise de rendez-vous pour le projet d'infographie 3D)*.

FIGURE 12 – Une montagne peuplée d'arbres