IFT3355 : Infographie
Travail Pratique 3 : Modélisation et animation

Remise : Lundi 5 décembre, à 23:59

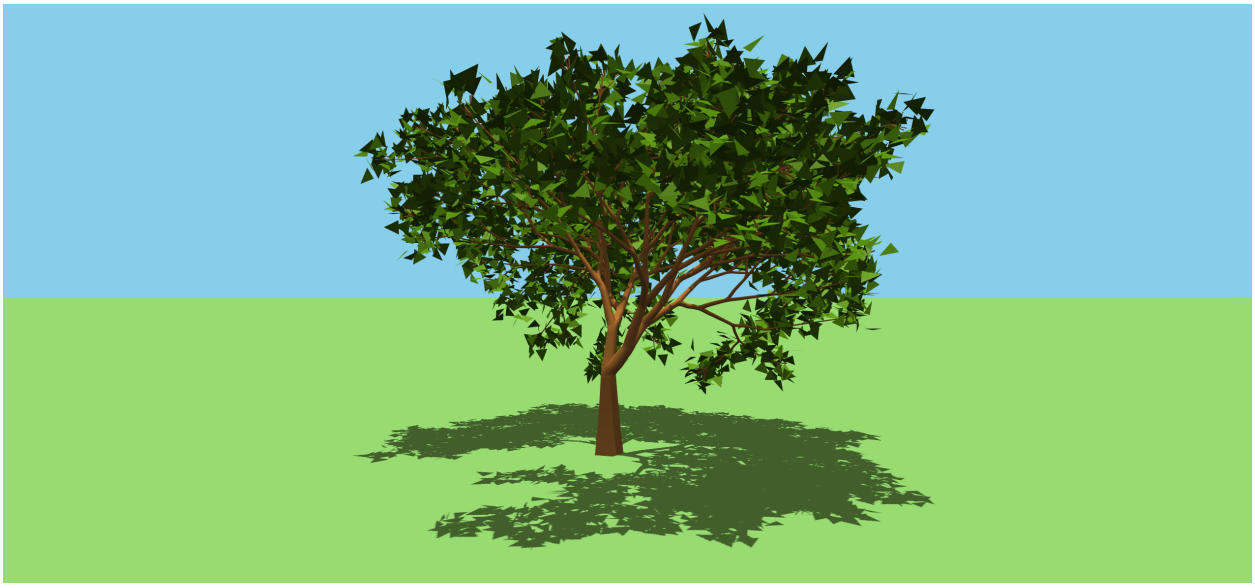


Figure 1: Arbre rendu grâce à un squelette de splines.

0 Introduction

L'objectif de ce TP est d'introduire des algorithmes et des méthodes utilisés en pratique pour la modélisation et l'animation en 3D. Dans ce TP, nous allons construire des modèles d'arbres à partir de squelettes pouvant être générés aléatoirement, puis animer ces arbres en temps réel avec de la gravité et du vent. La quantité de travail dans ce TP est adaptée pour des équipes de trois, mais reste faisable en équipe de deux.

1 Esquisse du code

Le TP est séparé en plusieurs parties pour faciliter sa complétion: géométrie, modélisation et animation. Du code support est fourni pour vous permettre de compléter les parties dans l'ordre que vous souhaitez. Par exemple, il est possible de compléter la partie animation partiellement en premier si vous êtes coincés dans la partie modélisation. Mais il est fortement recommandé de compléter le TP dans l'ordre donné, qui donne une progression plus naturelle. Pour des raisons historiques, nous nous servirons d'une base en JavaScript, ce qui réduit les performances du TP, mais en retour simplifiera grandement l'interface.

1.1 Moteur 3D

Nous allons utiliser `Three.js` pour le rendu en 3D. Toutes les méthodes présentes dans `Three.js` par défaut sont permises (aucune librairie externe n'est permise à part celles incluses dans le dossier `/js`), mais faites attention, aucune des méthodes par défaut (ex. `CylinderGeometry`, `TubeGeometry`) ne vous permet de créer un modèle continu d'un arbre avec les courbes de Hermite! Vous risquez de perdre des points si le modèle géométrique final contient des discontinuités avant et/ou durant l'animation. Construisez l'arbre manuellement avec des sommets, arêtes et faces. Une exception est permise dans la partie de modélisation approximative pour la visualisation, où les discontinuités sont permises.

1.2 Fichiers

Le dossier `/js` contient les librairies que vous pouvez utiliser pour ce TP. Notamment `three.js` pour le rendu en 3D et `parallel.js` pour ceux qui veulent ajouter du parallélisme.

`TP3_Data.js` contient les données des squelettes qui vous seront fournis. Cette partie du TP est déjà préremplie, vous n'avez pas besoin d'y toucher.

`TP3_Lindenmayer.js` contient des méthodes concernant la génération procédurale de modèles, et est obfusqué. Vous n'avez pas besoin d'ouvrir ce fichier.

`TP3_Geometry.js` contient toutes les méthodes qui servent à modifier les géométries.

`TP3_Physics.js` contient toutes les méthodes par rapport à l'animation.

`TP3_Render.js` contient toutes les méthodes par rapport au rendu.

`TP3_A1_Simplify.html` est une visualisation de la simplification des arbres.
Seule la fonction `TP3.Geometry.simplifySkeleton` doit être complétée pour cette partie.

`TP3_A2_RenderRough.html` est une visualisation du rendu approximatif. La touche de clavier espace (*space bar*) vous permet de générer un nouvel arbre (sauf pour `TP3.Data.TinyTree`).
Seule la fonction `TP3.Geometry.drawTreeRough` doit être complétée pour cette partie.

`TP3_B1_Segments.html` est une visualisation de la construction des segments de courbe Hermite.
Il faut avoir fait en prérequis le fichier `TP3_A1_Simplify.html`, et les fonctions `TP3.Geometry.generateSegmentsHermite`, et `TP3.Geometry.hermite`.

`TP3_B2_RenderHermite.html` est une visualisation du rendu final de l'arbre. La touche de clavier espace (*space bar*) vous permet de générer un nouvel arbre.
Il faut avoir fait en prérequis le fichier `TP3_B1_Segments.html`, et la fonction `TP3.Render.drawTreeHermite`.

`TP3_C1_SkeletonAnimate.html` est une visualisation de l'animation du squelette de l'arbre.
Il faut avoir fait en prérequis le fichier `TP3_B1_Simplify.html` et `TP3.Physics.applyForces`.

`TP3_C2_HermiteAnimate.html` est une visualisation de l'animation finale de l'arbre. La touche de clavier espace (*space bar*) vous permet de générer un nouvel arbre.
Il faut avoir fait tous les TODO présents dans le code pour voir le résultat attendu !

2 Description du travail (100 pts)

Ici se présente une courte description du travail requis pour chaque partie du TP. Pour les détails des algorithmes, visitez la section 3. Pour le barème, voir la section 4.

2.1 Modélisation 3D (65 pts)

Construire un mailliage 3D et l'ajouter dans la scène à partir d'un squelette à simplifier.

a) Simplification du squelette (5 pts)

Complétez la fonction `TP3.Geometry.simplifySkeleton(...)`. La fonction prend en paramètre le noeud principal `rootNode`, l'angle maximal de rotation `rotationThreshold`, et retire les noeuds (branches) dans l'arbre qui n'ont qu'un seul enfant et qui ont une rotation d'angle plus petite que `rotationThreshold` comparée à son parent. Attention de mettre à jour correctement les valeurs `parentNode`, `childNode`, `p0`, `p1`, `a0` et `a1` des noeuds précédents et suivants des noeuds retirés. Finalement, la fonction retourne le noeud principal de l'arbre.

b) Modélisation approximative (10 pts)

Construire un modèle approximatif de l'arbre avec des cylindres simples pour les branches et des carrés pour les feuilles. Complétez la fonction `TP3.Render.drawTreeRough(...)`. La fonction prend en paramètre le noeud principal `rootNode`, la scène du monde `scene`, la valeur alpha des branches `alpha` (Attention! les branches peuvent être plus longues que alpha après la simplification du squelette. N'utilisez pas cette valeur pour la longueur des cylindres. Cette valeur est seulement utilisée pour calculer la valeur de coupure des feuilles.), le nombre de subdivisions radiales des cylindres `radialDivisions`, le facteur de coupure des feuilles `leavesCutoff`, le nombre de feuilles par branche `leavesDensity` et la matrice de transformation `matrix` pour l'arbre en entier.

Vous pouvez utiliser `CylinderBufferGeometry` et `PlaneBufferGeometry` pour initialiser le maillage des segments de branches et des feuilles, puis utiliser `THREE.BufferGeometryUtils.mergeBufferGeometries` pour combiner les segments de branches en un seul maillage et combiner les feuilles en un autre maillage. Cette dernière fonction accélère le rendu puisque vous ajoutez seulement deux maillages dans la scène à la fin de votre fonction.

Utilisez les matériaux suivants:
`THREE.MeshPhongMaterial({color: 0x3A5F0B})` pour les feuilles,
`THREE.MeshPhongMaterial({color: 0x5F0B0B})` pour les pommes, et
`THREE.MeshLambertMaterial({color: 0x8B5A2B})` pour les branches.

c) Courbes de Hermite (15 pts)

Complétez la fonction `TP3.Geometry.hermite(h0, h1, v0, v1, t)`. La fonction doit retourner le vecteur du point interpolé et la tangente normalisée à ce point.

La fonction doit pouvoir être appelée comme ceci:

```
const [p, dp] = hermite(h0, h1, v0, v1, t);
```

En Javascript, vous pouvez retourner plusieurs variables comme ceci:

```
hermite: function (h0, h1, v0, v1, t) {
    const p = ...;
    const dp = ...;
    return [p, dp];
}
```

d) Modélisation avec maillage continu (35 pts)

Cette partie est séparée en deux fonctions pour faciliter le débuggage.

Complétez la fonction `TP3.Render.generateSegmentsHermite(...)`. La fonction prend en paramètre le noeud principal `rootNode`, le nombre de subdivisions le long des cylindres `lengthDivisions` et le nombre de subdivisions radiales des cylindres `radialDivisions`. La fonction doit générer, pour chaque noeud, une liste de sections du cylindre de taille `lengthDivisions` qui suit la courbe de Hermite et l'enregistrer dans la propriété `sections`. Chaque section est constituée d'une liste de points de taille `radialDivisions` qui forment un cercle. Ce cercle représente une coupe transversale du cylindre aux différentes positions sur la courbe de Hermite. Faites attention à l'orientation de chaque tranche, elle doit être cohérente à travers chaque section et chaque noeud pour pouvoir générer les faces de façon continue pour le maillage à la prochaine étape. Voir la section 3 pour plus de détails sur la génération des tranches. La fonction retourne le noeud principal. Vous pouvez aussi ajouter dans les noeuds toute information que vous jugez utile ou nécessaire pour les prochaines étapes.

Construire un maillage complet et continu de l'arbre avec des cylindres généralisés pour les branches et des faces triangulaires pour les feuilles.

Complétez la fonction `TP3.Render.drawTreeHermite(...)`. La fonction prend en paramètre le noeud principal `rootNode`, la scène du monde `scene`, la valeur alpha des branches `alpha`, le facteur de coupure des feuilles `leavesCutoff`, le nombre de feuilles par branche `leavesDensity`, la probabilité qu'une pomme soit sur un noeud `applesProbability`, et la matrice de transformation `matrix` pour l'arbre en entier.

Utilisez `BufferGeometry` pour initialiser le maillage des segments de branches et des feuilles. Utilisez `BufferGeometry.setAttribute("position", new THREE.BufferAttribute(f32Vertices, 3))` pour initialiser les sommets du maillage, où `f32vertices` est un tableau de `float32` de points x, y, z. Par exemple, si vous voulez initialiser un `BufferGeometry` avec les sommets $(0, 0, 0)$, $(2, 1.5, 1.7)$ et $(8.6, 3.1, 4.2)$, vous pouvez faire:

```
const vertices = [];
vertices.push(0.0, 0.0, 0.0);
vertices.push(2.0, 1.5, 1.7);
vertices.push(8.6, 3.1, 4.2);
const f32vertices = new Float32Array(vertices);

const geometry = new THREE.BufferGeometry();
geometry.setAttribute("position", new THREE.BufferAttribute(f32vertices, 3));
```

Vous devez aussi initialiser les faces du maillage. Utilisez `BufferGeometry.setIndex(indices)`, où `indices` est une liste d'indices des sommets a, b, c qui forment une face triangulaire dans l'ordre anti-horaire. La normale de la face calculée, avec l'ordre anti-horaire, est comme si vous êtes à l'intérieur du polygone, regardant vers l'extérieur. Un exemple de code qui ajoute une face pour les trois sommets précédents, et calcule les normales aux sommets:

```
const facesIdx = [];
facesIdx.push(0, 1, 2);

geometry.setIndex(facesIdx);
geometry.computeVertexNormals();
```

Finalement, la fonction doit ajouter les maillages dans la scène avec les matériaux mentionnés ci-haut, et retourner le maillage (`BufferGeometry`) complet des branches et le maillage complet des feuilles.

2.2 Animation 3D (35 pts)

Calculez le mouvement des branches et appliquez ce mouvement au maillage construit.

a) Animation du squelette (10 pts)

Complétez la fonction `TP3.Physics.applyForces(...)`. La fonction prend en paramètre un noeud `node`, l'intervalle de temps `dt`, le temps `time`, et elle doit modifier la position `p0` et `p1` du noeud en fonction de l'intervalle de temps et les forces calculées; elle s'appelle récursivement sur les noeuds enfants. Notez qu'il est nécessaire de propager ce mouvement aux noeuds enfants. La propagation peut être implémentée de façon inverse (ex. chaque noeud regarde son parent et applique les mêmes mouvements avant de calculer les forces). Vous pouvez aussi ajouter dans les noeuds toute information que vous jugez utile ou nécessaire pour les prochaines étapes. Par exemple, calculer et garder la matrice de transformation qui représente le mouvement de `p0` et `p1` sera utile pour la prochaine partie. Cette matrice de transformation pourrait aussi être utile pour propager le mouvement aux enfants.

b) Animation du maillage continu (25 pts)

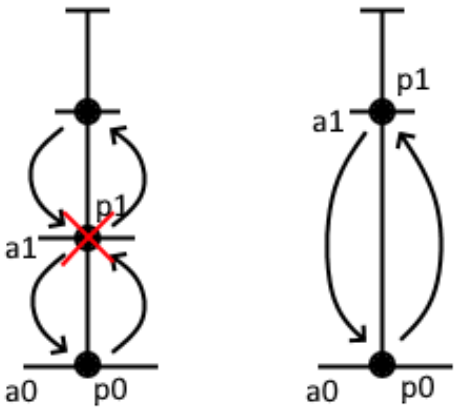
Complétez la fonction `TP3.Render.updateTreeHermite(...)`. La fonction prend en paramètre la liste de positions des sommets du maillage des branches `trunkGeometryBuffer`, des feuilles `leavesGeometryBuffer`, des pommes `applesGeometryBuffer`, le noeud principal `rootNode` et doit modifier la position des sommets du maillage `trunkGeometryBuffer`, `leavesGeometryBuffer`, et `applesGeometryBuffer`. Il sera utile de modifier la fonction `TP3.Render.generateSegmentsHermite(...)` pour garder les indices des sommets pour chaque noeud (branche). Utilisez ces indices pour retrouver les sommets associés à chaque noeud et appliquez les mêmes transformations que vous avez faites pour la partie précédente.

3 Description des algorithmes

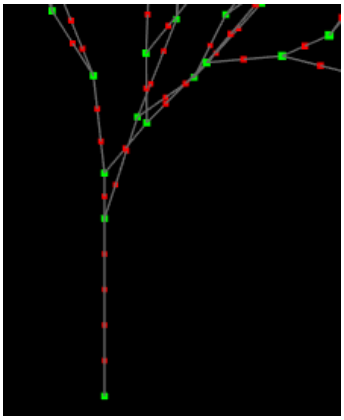
3.1 Modélisation 3D

a) Simplification du squelette

Retirez les noeuds (branches) dans l'arbre qui n'ont qu'un seul enfant et qui ont une rotation d'angle plus petite que `rotationThreshold` comparée à son parent. Attention de mettre à jour correctement les valeurs `parentNode`, `childNode`, `p0`, `p1`, `a0` et `a1` des noeuds précédents et suivants des noeuds retirés.



Simplification d'un noeud.



Simplification d'un arbre. Les noeuds rouges sont retirés.

Figure 2: Étape de simplification.

b) Modélisation approximative

La modélisation approximative nous permet de visualiser notre arbre de façon simple et rapide. Construisez l'arbre en utilisant un cylindre pour chaque branche et des carrés de taille `alpha`×`alpha` pour les feuilles. Placez `leavesDensity` feuilles aléatoirement (avec translation et rotation aléatoire) dans un rayon de taille

`alpha/2` autour des branches qui ont une largeur `a0` plus petite que `alpha * leavesCutoff`. Pour les branches terminales (qui n'ont pas d'enfant), laissez les feuilles dépasser la longueur de la branche de `alpha`. Construisez également des pommes aléatoirement sur certains noeuds: utilisez d'abord le même critère de largeur de branches que pour les feuilles, puis utilisez `applesProbability` pour savoir si une pomme doit se trouver sur chacun des noeuds. Pour cette modélisation approximative, construisez des pommes à partir de cubes de dimensions `alpha`×`alpha`×`alpha`.

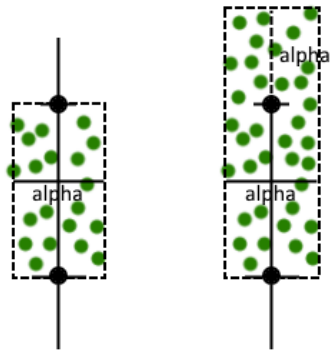


Figure 3: Placement des feuilles autour des branches non-terminales et terminales.

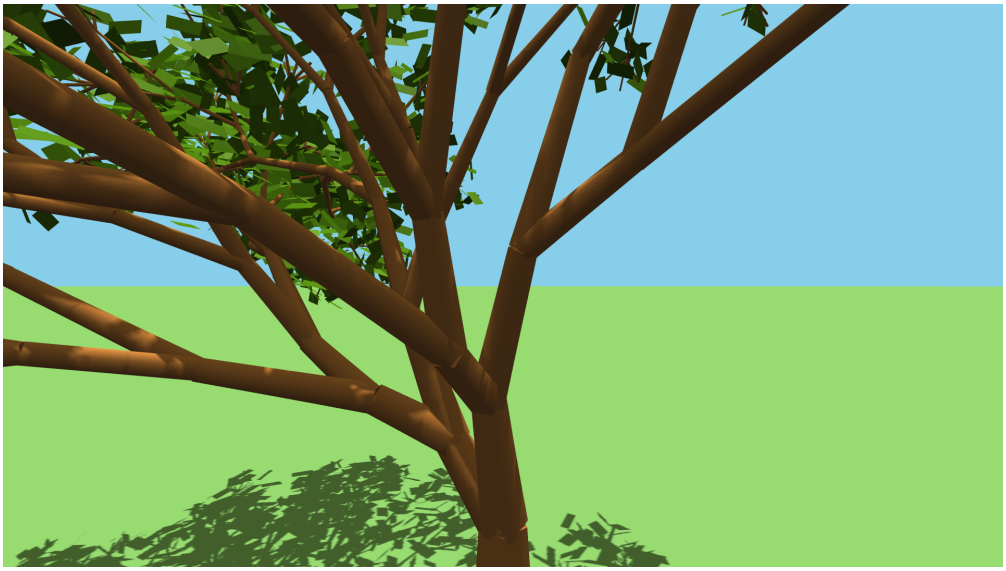


Figure 4: Problèmes de discontinuités avec la modélisation approximative.

c) Courbes de Hermite

Vous pouvez implémenter la courbe de Hermite paramétrique, ou plus simplement, convertir la courbe de Hermite en courbe de Bézier, et utiliser l'algorithme de De Casteljau pour la construire. Remarquez que trouver la tangente de la courbe avec l'algorithme de De Casteljau est trivial.

$$\begin{bmatrix} p_0 \\ p_1 \\ p_2 \\ p_3 \end{bmatrix} = \frac{1}{3} \begin{bmatrix} 3 & 0 & 0 & 0 \\ 3 & 0 & 1 & 0 \\ 0 & 3 & 0 & -1 \\ 0 & 3 & 0 & 0 \end{bmatrix} \begin{bmatrix} h_0 \\ h_1 \\ v_0 \\ v_1 \end{bmatrix}$$

Figure 5: Conversion d'une courbe de Hermite en courbe de Bézier.

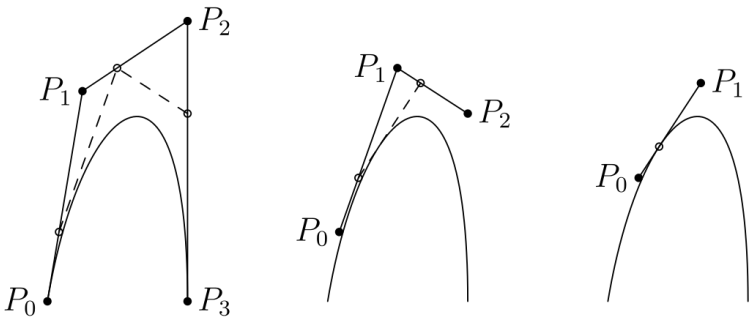


Figure 6: Construction d'une courbe de Bézier avec l'algorithme de De Casteljau.

d) Modélisation avec maillage continu

Pour chaque branche, générez la courbe de Hermite en prenant $h_0 = p_0$, $h_1 = p_1$, $v_0 = p_{1parent} - p_{0parent}$, $v_1 = p_1 - p_0$. Séparez cette courbe en un nombre de segments discrets selon `lengthDivisions`. Pour chaque segment, générez un nombre de points autour de ce segment avec un rayon interpolé entre `a0` et `a1` selon `radialDivisions`. Ces points seront les sommets de votre maillage.

Un problème majeur peut survenir lorsque vous générez vos points autour des segments. Ces points doivent être orientés par un "moving frame" qui suit la courbe de Hermite. Si ils ne sont pas orientés correctement, la génération des faces va être très difficile.

Puisque ce "moving frame" peut être difficile à calculer pour une courbe de Hermite, nous pouvons utiliser une approximation. Initialisez une matrice de transformation identité. Pour chaque segment de Hermite de votre arbre, calculez la matrice de rotation entre v_t et v_{t+dt}. Multipliez et affectez votre matrice de transformation pour garder une orientation cohérente.

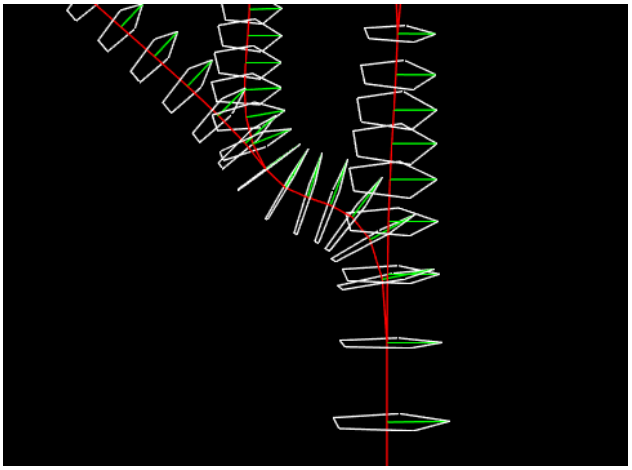


Figure 7: Construction correcte des segments de Hermite. Notez l'orientation cohérente du vecteur vert.

Pour compléter le maillage, il suffit de connecter les segments comme si c'étaient des cylindres simples. Les faces sont des triangles qui contiennent deux points dans un segment et un point dans un autre segment. N'oubliez pas de fermer le maillage sur les segments terminaux (*leaf nodes*) et principal (*root node*).

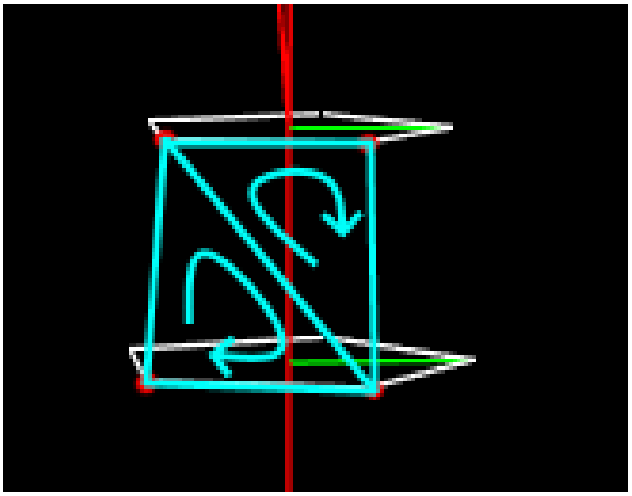


Figure 8: Construction des faces du maillage.

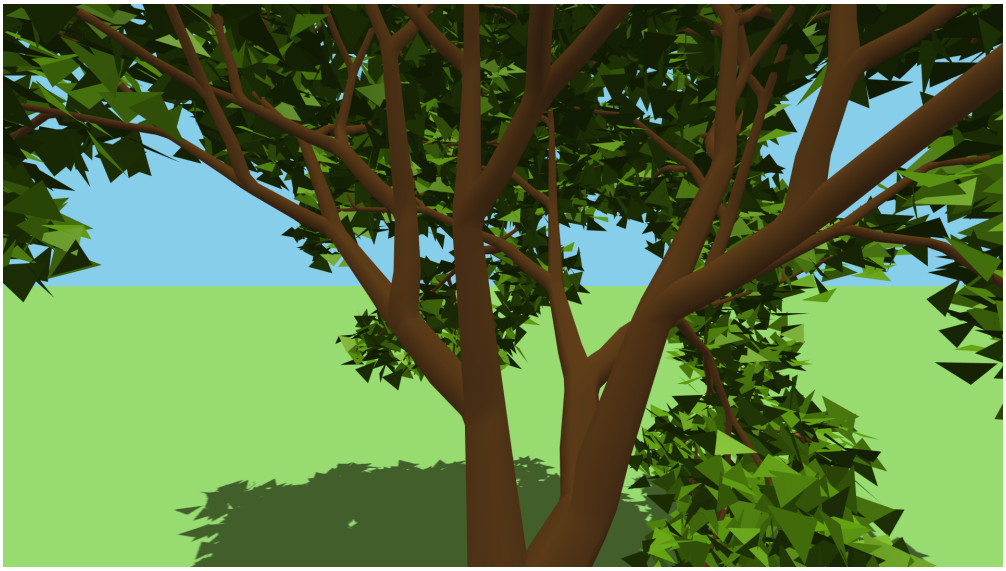


Figure 9: Arbre construit avec des cylindres généralisés.

Les feuilles doivent êtres générées de la même façon que la modélisation approximative, mais cette fois, vous pouvez utiliser une face triangulaire équilatérale pour les feuilles.

Affectez la propriété `side: THREE.DoubleSide` au matériau pour les feuilles, sinon elles seront invisibles par un côté.

Les pommes doivent être générées de la même manière probabilistes que pour la modélisation approximative, mais en utilisant cette fois une sphere de rayon `alpha`/2
(indice: vous pouvez utiliser `THREE.SphereBufferGeometry` pour générer les coordonnées ainsi que les indexes de ces sphères. Référez-vous à la documentation de THREE.js pour plus de détails).

3.2 Animation 3D

a) Animation du squelette

Pour l'animation des forces, on vous fournit déjà les vélocités de la gravité (le poid des pommes est déjà pris en compte pour vous) et le vent pour chaque noeud dans `Node.vel`. Votre travail consiste à calculer le mouvement sur tout l'arbre. Pour calculer ce mouvement, il faut projeter la vélocité pour conserver la longueur d'une branche. Calculez d'abord la nouvelle position $p_{1(t+dt)} = p_{1(t)} + v * dt$ de la branche, notez que cette position $p_{1(t+dt)}$ ne conserve pas la longueur. Trouvez la matrice de rotation entre le nouveau vecteur $p_{1(t+dt)} - p_0$ normalisé et l'ancien vecteur $p_1 - p_0$ normalisé. Cette matrice de rotation va conserver la longueur de votre branche. Appliquez cette matrice à $p_{1(t)}$, et calculez la vraie vélocité après la projection. Remplacez l'ancienne vélocité par cette vraie vélocité projetée.

Ensuite, calculez la force de restitution de la branche. Calculez l'angle entre la direction initiale de la branche et la direction présente de la branche. Ajoutez une vélocité dans le sens contraire de cet angle au carré (la vélocité de restitution au carré est proportionnel à l'angle). Multipliez cette vélocité par `a0 * 1000`. Ceci donnera une force de restitution plus grande pour les branches plus épaisses. Ajoutez cette vélocité de restitution à la vélocité totale `Node.vel`.

Pour réduire les problèmes des oscillations, appliquez un facteur d'amortissement (*damping*) sur la vélocité. Nous utiliserons un facteur de 0.7 (multipliez les éléments de la vélocité par 0.7).

Finalement, calculez et propagez la matrice de transformation aux noeuds enfants. Les noeuds enfants doivent appliquer cette matrice aussi. Appelez récursivement cette fonction sur les noeuds enfants.

b) Animation du maillage continu

Vous devriez avoir une matrice de transformation pour chaque branche par la partie précédente. Appliquez simplement les matrices aux sommets correspondants de votre maillage.

4 Barème

4.1 Code

a) Clarité du code (jusqu'à -10 pts)

Du code spaghetti, du code obscurci ou quasi-obscurci, trois mille variables d'un seul caractère, bref, tout code qui est illisible ou incompréhensible va entraîner une perte de 10 points au maximum sur la partie en question. Créez des nouvelles fonctions pour faciliter la réutilisation du code et éviter le code spaghetti. Commentez le code au besoin.

b) Performance du code (jusqu'à -10 pts)

Puisqu'on travaille avec du rendu en temps réel, une performance raisonnable est demandée pour ce TP, pensez à retirer du code qui ne sert à rien, sans nuire à la lisibilité. Tout code qui ne roule pas dans un temps raisonnable va entraîner une perte de 10 points au maximum sur la partie en question. Par exemple, une animation qui roule à 0.4FPS, ou un maillage qui prend 2 minutes à générer.

4.2 Modélisation 3D (65 pts)

Simplification correcte du squelette. (5 pts)

Construction correcte du maillage. Sommets, faces et normales correctes. Génération des feuilles correcte. (10 pts)

Construction de la courbe de Hermite correcte. (15 pts)

Construction correcte du maillage continu. Sommets, faces et normales corrects. Pas de sommets dupliqués. (i.e. deux faces qui sont supposées être connectées au même sommet ne doivent pas être connectées à des sommets différents mais qui sont à la même position!) Pas de faces dupliquées non plus. Aucune discontinuité entre les segments. Le maillage doit être fermé partout. Génération des feuilles correcte. (35 pts)

4.3 Animation 3D (35 pts)

Animation du squelette correcte. Stabilité du mouvement. Contrainte de longueur respectée. (10 pts)

Animation du maillage continu correcte. Stabilité du mouvement. Contrainte de longueur respectée. Aucune discontinuité visible durant l'animation. (25 pts)

5 Améliorations possibles - Même si elles ne seront pas évaluées

Il est recommandé de créer des nouvelles scènes et des nouvelles fonctions pour ces améliorations optionnelles. Ceci vous évitera de perdre des points dans les parties évaluées si un bug est découvert durant la correction.

a) Feuilles

Les feuilles statiques et triangulaires qui flottent autour des branches ne sont pas très belles. Vous pouvez modéliser des feuilles plus réalistes, qui ne flottent pas et sont attachées aux branches avec des rotations initiales plus ou moins aléatoires (à vous de décider comment). Ces feuilles pourraient aussi êtres animées correctement par la gravité et le vent.

b) Textures

L'arbre pourrait avoir des textures pour son tronc, branches et feuilles. Pensez à comment faire le mapping de texture sur les cylindres généralisés.

c) Collision et simplification du maillage

Certaines branches intersectent d'autres branches. Ce sont des discontinuités moins importantes, mais elles peuvent causer des problèmes. Traitez ces cas spéciaux sur le maillage. Traitez les collisions des branches séparées. Pour les branches qui proviennent d'une même branche parent, joindre les faces qui intersectent et retirer les faces qui sont cachées dans la branche intersectée.

Figure 10: La branche à droite s'enfonce directement dans la branche à gauche. Une petite discontinuité est visible.

6 Modalités de remise

Veuillez inclure tout votre code ainsi qu'un fichier README contenant simplement vos noms et vos matricules dans un fichier zip.
Nommez cette archive de la même manière qu'au TP2 (soit *matricule1_matricule2_matricule3.zip*).

7 Bon travail et bonne chance!

Commencez tôt pour pouvoir poser des questions en avance si des problèmes surviennent. Ne prenez pas de retard, car ce TP ne peut pas être fait en une ou deux journées avant la remise!