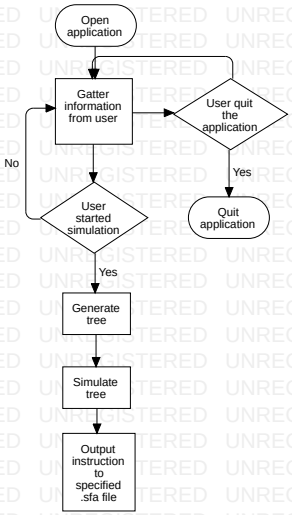
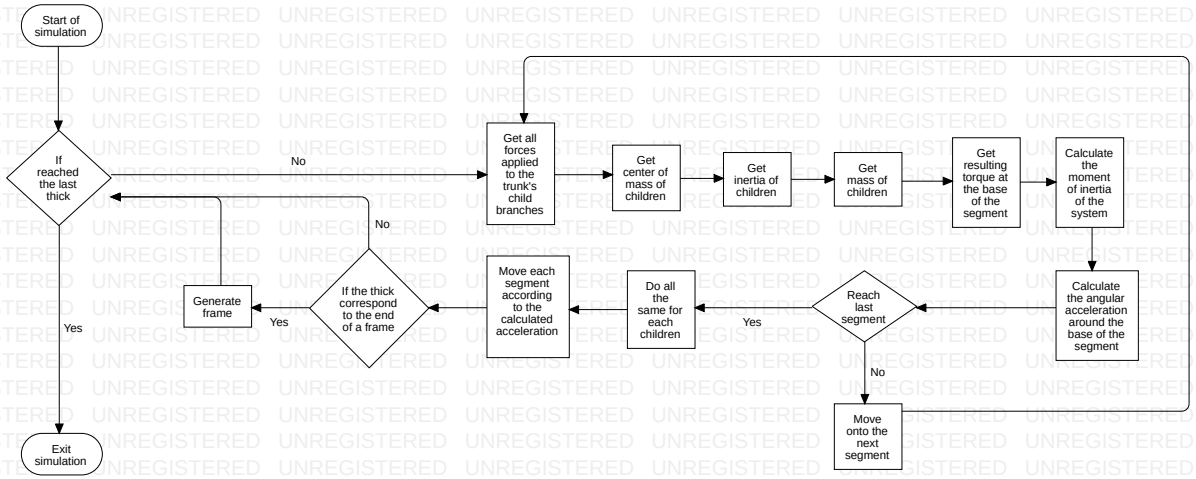
# Flowchart Flowchart Diagram



## Grille d'évaluation

En guise d'autoévaluation, imprimez et remplissez vous-même cette section. Mentionnez une référence (nom de fichier et numéro de ligne) lorsque demandé pour faciliter la vérification par le correcteur.

| | **Spécifications techniques** | **Oui** | **Non** | **Référence (fichier, ligne)** |
|---|---|---|---|---|
| **5** | Modélisation complète de l'effet du vent sur l'arbre, intégration avec le lab. 2 | ☑ | ☐ | Branch.h et Physics.h |
| **5** | Influence de la force du vent sur l'amplitude de la déformation | ☑ | ☐ | Branch.h et Physics.h |
| **5** | Influence du niveau de la branche sur la déformation due au vent | ☐ ✓ | ☐ | la résistance à la déformation dépend de la longueur et du Ø, indirectement dicté par le niveau Branch.h Tree.h Physics.h |
| **5** | Influence de l'orientation de la branche sur la déformation due au vent | ☑ | ☐ | Branch.h Torque.h Physics.h |
| **5** | Modélisation et affichage d'un vent horizontal variable | ☑ | ☐ | Wind.h supporte aussi l'axe vertical |
| **5** | Modélisation réaliste, correcte et fluide | ☑ | ☐ | Utilise les formules de mécanique des fluides, physique classique et résistance des matériaux pour gérer le mouvement |
| | **Spécifications de programmation** | **Oui** | **Non** | |
| **5** | Structures de données, TDA arbre | ☑ | ☐ | Tree.h et Branch.h |
| **5** | Gestion dynamique de la mémoire et libération de l'espace alloué | ☑ | ☐ | Tree.h, Branch.h et array.h |
| **(-10)** | Aucune variable globale | ☑ | ☐ | |

| | **Qualité logicielle** | **T.B.** | **Bon** | **Pauvre** | **Référence (fichier, ligne)** |
|---|---|---|---|---|---|
| **5** | Modularité : usage généralisé de fonctions et regroupement logique en modules | ☐ | ☑ | ☐ | array.h est un template<br>Physic.h contient des fonction de physique général purpose |
| **5** | Robustesse : Précaution pour éviter les problèmes de données erronées, etc. | ☐ | ☑ | ☐ | Branch.cpp ligne 317<br>Physics.cpp ligne 43 |
| **5** | Documentation : Commentaires pertinents qui accélèrent la compréhension du code | ☐ | ☑ | ☐ | |
| **5** | Clarté du code : Code concis et court utilisant les fonctions adaptées | ☐ | ☑ | ☐ | code écrit comme les formules mathématiques écrites en commentaires au dessus |
| **60** | **TOTAL** | | | | |

Section réservée au correcteur :

| | |
|---|---|
| / 10 | Organigramme |
| / 30 | Réponses aux questions |
| / 60 | Respect des spécifications |
| **/ 100** | **Total** |

Q : Quelle est la pertinence de la structure de données choisie dans ce laboratoire ? Est-ce que cette représentation des données est supérieure à un autre choix de TDA ? Quels sont les avantages et les inconvénients du TDA choisi par rapport à un autre TDA (de votre choix) ?
R : L'arbre n$^{aire}$ a pour principal avantage to pouvoir d'avoir plusieurs éléments rattachés à un même nœud. Ceci nous est particulièrement utile pour faire le lien entre une branche parent et ses nombreux enfants. En effet, nous ne savons pas combiens d'enfant une branche aura avant l'exécution du programme, il est donc important que le nombre d'enfant que peut accepter un nœud soit variable, comme c'est le cas pour la TDA Arbre. Dans mon implémentation de l'arbre pour ce laboratoire, les nœuds ont également un chemin d'accès vers leur parent afin de pouvoir obtenir de l'information sur leur position dans l'espace. Ceci évite de devoir conserver ses informations à même le nœud est devoir faire du « forward propagation » lorsque cette information a changé.

Q : Comment pourrait-on modifier la structure de données que vous avez définie afin de permettre l'addition de feuilles, de fruits ou d'autres éléments qui apparaissent aléatoirement aux dernières branches de l'arbre ?

R : Puisque les nœuds utilisent des pointeurs pour se référencer entre eu, nous pourrions ici directement utiliser la structure déjà en place et des éléments aléatoires des classes descendantes de Branch et écraser la fonction d'affichage pour afficher la bonne forme et la fonction de simulation si nous voulons que ses éléments se comportent différemment. Nous aurions ainsi les deux requis pour utiliser le polymorphisme, soit un lien d'héritage entre les classes et un appel au travers d'une référence, ici un pointeur. Donc non, la structure n'a pas besoins de changer elle-même, c'est la façon de créer les autres éléments qui doit être adaptée.

Q : Chez un arbre réel, il est possible que des branches seules ou des groupes de branches apparaissent après une longueur constante sur un tronc, ou simplement à des longueurs plus ou moins aléatoires. Nous voulons permettre que des branches sortent d'un nœud qui se trouverait au milieu (et non pas à l'extrémité) d'une branche. Expliquez comment cela est réalisable par une modification de votre structure de données.

R : La structure elle-même peut être conservée, il suffirait d'implémenter une fonction qui sectionnerait une branche à une longueur aléatoire, créant ainsi deux branches liées ensemble en un nœud. De nouvelles branches plus petites peuvent alors être ajoutée à ce nouveau nœud. Puisque la résistance en flexion d'une branche dépend en grande partie de son diamètre et que celui-ci serait constant des deux côtés du nouveau nœud, la dynamique du système ne devrait pas trop être affectée.