Simulation de deltaplanes et de cerf-volants en temps réel

Éric Beaudry
Université de Sherbrooke
Eric.Beaudry@USherbrooke.ca

Richard Egli
Université de Sherbrooke
Richard.Egli@USherbrooke.ca

Éric Patenaude
Ubisoft
eric_patenaude2001@hotmail.com

31 janvier 2005

Résumé

Ce papier décrit la conception géométrique d'un deltaplane et d'un cerf-volant virtuels pour effectuer une animation simulée en temps réel sur laquelle on peut apercevoir les battements de toile au contact de l'air. La méthode utilisée pour la simulation des toiles et des structures rigides des objets est basée sur l'intégration Verlet jumelée avec une méthode simple de contraintes de distance.

1 Introduction

La génération d'animation automatique est de plus en plus utilisée dans le domaine du multimédia. L'utilisation d'une méthode automatique représente un gain de temps significatif dans la production d'animations 3D. Au lieu de recourir aux services d'un animateur pour animer un personnage ou un objet, on fait une modélisation physique de ce dernier. Ainsi, en utilisant des lois de la physique mécanique, on peut simuler le comportement des objets virtuels avec un bon degré de réalisme. De plus, dans certains cas, comme dans les jeux vidéo, les objets animés dépendent des actions du joueur. Puisqu'il est pratiquement impossible de prévoir toutes les actions possibles du joueur, il est essentiel de pouvoir simuler les effets de ses actions sur les objets ou les personnages contrôlés.

Dans cet article, nous présentons la conception d'objets volants basés sur des toiles de tissu, soit un deltaplane, un cerf-volant, et un parachute. La simulation de ces objets est basée sur des méthodes relativement simples donnant de bons résultats. Afin de simuler de tels objets, nous avons évalué deux approches différentes. La première, qui est d'ailleurs largement connue, est l'utilisation d'un réseau de masses et ressorts[Pro95]. La deuxième méthode, proposée par [Jak01], est la satisfaction de contraintes de distances avec l'intégration Verlet [Ver67].

Le reste de cet article sera organisé de la façon suivante : nous ferons un résumé des méthodes de simulation étudiées ; suivi d'une justification du choix de la méthode utilise ; nous élaborons la conception d'un deltaplane et du cerf-volant ; nous présenterons quelques limitations avec l'exemple sur un parachute ; enfin, la conclusion suivra.

2 Simulation physique d'objets

Avant de réaliser la simulation d'un deltaplace et d'un cerf-volant, il faut d'abord s'intéresser à la simulation de ses composantes. Or ces deux objets sont composés d'une structure rigide sur laquelle une toile est fixée. Dans cette section, nous faisons un résumé des techniques de base pour la conception d'objets en vue de simulation.

La simulation physique d'objets est à la base très simple. Avec des lois de physique mécanique, il est possible de simuler, avec une allure très réaliste, des objets dans un environnement virtuel.

2.1 Particules

La simulation du mouvement d'une particule est triviale. La trajectoire d'une particule peut être calculée avec l'équation 1 dans laquelle on introduit les paramètres suivants : la masse *m*, la somme des forces exercées *f* sur la particule, la position initiale *p(0)* et la vélocité initiale *v(0)*.

$$p(t) = p(0) + \frac{(v(0) + (f/m) * t)}{2} * t \qquad (1)$$

2.2 Objets rigides

Après les particules, les objets rigides sont sans doute les objets les plus faciles à simuler. Pour y arriver, il faut d'abord calculer les propriétés suivantes : la masse *m*, le centre de gravité *c*, et le torque *t* représentant la résistance à la rotation. De plus, quelques variables dynamiques sont nécessaires, telles que la position *p*, la vélocité *v* ainsi que le moment d'inertie *e*. Tout comme les objets particules, ces paramètres dynamiques peuvent être calculés à l'aide de formules mécaniques.

2.3 Tissus et toiles

Enfin, les tissus sont des objets plus difficiles à simuler en raison de leur déformation complexe. Une technique populaire pour la simulation de tissus est l'utilisation d'un réseau de masses et de ressorts [Pro95, OZH00, Bou01]. Cette méthode consiste à décomposer le tissu simulé en un maillage de petites particules reliées par des ressorts très rigides qui excercent des forces maintenant la forme de l'objet (voir 1).

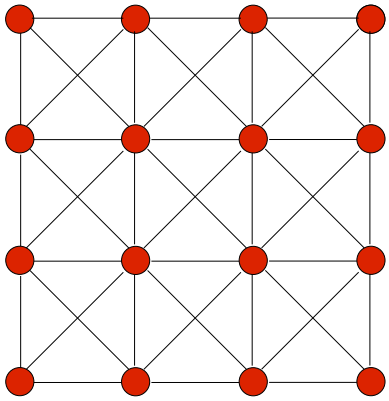

FIG. 1 – Tissu représenté par un réseau de masses et de ressorts

2.4 Objets mixtes

Un cas plus complexe est la simulation d'objets mixtes, soit ceux composés d'une structure rigide et d'une partie flexible, tel un tissu. Le deltaplane et le cerf-volant font partie de cette catégorie. Une façon de procéder est l'unification des deux techniques précédement décrites. Chacune des parties sont simulées indépendamment, et à chaque itération de simulation, des forces de fixations peuvent être calculées afin de garder l'objet unifié.

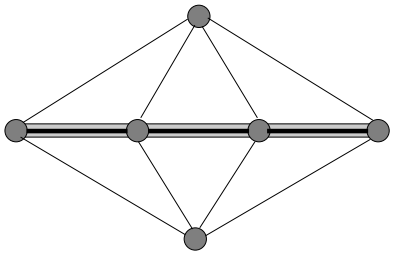

FIG. 2 – Contraintes de rigiditées

Par contre, il existe une autre technique plus simple pour simuler les objets mixtes. Au lieu de simuler la partie rigide de l'objet, on la décompose elle aussi en plusieurs particules. Ensuite, en ajoutant des particules et des ressorts supplémentaires à des endroits stratégiques, il est possible d'imposer des contraintes de rigidité. Un exemple est montré à la figure 2. L'objet modélisé est une tige formée de quatre particules reliées en ligne droite par trois ressorts. Afin de conserver l'allongement et la droiture de l'objet, deux particules supplémentaires ont été ajoutées en haut et en bas de l'objet. Grâce à des ressorts supplémentaires reliant ces deux particules à celles de l'objet, une contrainte de rigidité est créée.

3 Méthodes d'intégration

Au début de la section précédente, nous avons présenté une équation permettant de calculer la trajectoire d'un projectile. Cette équation est obtenue par la résolution d'une intégrale. Or, en pratique, il est très rare que l'utilisation d'une intégration symbolique soit possible en raison de l'aspet dynamique de l'environnement simulé. Par exemple, il faut tenir compte des collisions, des changement d'accélération, des commandes de l'usager, etc. Pour ces raisons, nous avons recours à l'intégration numérique, ce qui consiste à calculer itérativement des équations plusieurs fois par seconde.

Ordinairement, la méthode d'intégration est appelée dans une boucle de simulation telle que montrée à la figure 3. En premier lieu, on calcule les forces appliquées sur les objets, comme la gravité, la friction avec l'air (vent), les tensions exercées par les ressorts, etc. Ensuite, à l'aide des forces calculées, on fait une intégration. Après, si nécessaire, on peut appliquer des contraintes, comme la détection de collision. En dernier lieu, on peut procéder à l'affichage et synchroniser le temps de simulation avec le temps réel.

```
void simulation(){
   while(true){
      calculerForces(); // Physique
      faireIntegration();
      appliquerContraintes();
      afficher();
      synchroniserTemps();
   }
}
```

FIG. 3 – Précédure générale de simulation

3.1 Intégration Euler

L'intégration par Euler est l'une des techniques les plus utilisées puisque c'est une méthode très simple. La méthode consiste à itérativement (voir Fig. 4) évaluer la position et la vélocité de l'objet simulé pour de petite tranche de temps, soit

un $\triangle t$ variant entre 1/100 à 1/1000 sec. Voir [Bou01] pour plus d'explications.

$$pos = pos + vel * \triangle t$$
$$acc = force/mass$$
$$vel = vel + acc * \triangle t$$

FIG. 4 – Algorithme itératif Euler

3.2 Intégration Verlet

Contrairement à l'intégration Euler, Verlet ne conserve pas la vélocité de l'objet, mais plutôt sa dernière position en plus de sa position actuelle. En regardant son pseudo-code (voir figure 5), on peut constater que l'intégration Verlet est une méthode mathématiquement équivalente à celle de Euler. Cette équivalence provient du fait que la vélocité peut être connue à partir de la différence entre la position actuelle et l'ancienne position ($vel = (pos - pos_{old}) / \triangle t$).

$$acc = force/mass$$
$$pos_{new} = 2 * (pos - pos_{old}) + acc * \triangle t^2$$
$$pos_{old} = pos$$
$$pos = pos_{new}$$

FIG. 5 – Algorithme itératif Verlet

Puisqu'elles sont équivalentes, on peut se demander quel est l'intérêt d'utiliser l'intégration Verlet plutôt que celle d'Euler. Les avantages d'une méthode par rapport à une autre se manifestent lorsq'on les combine avec les satisfactions de contraintes. Par exemple, examinons le cas d'une détection de collision. Si un objet touche le sol, il faut corriger sa position afin d'assurer une valeur seuil à l'objet. Avec l'intégration Euler, en plus de corriger la position, il faut s'assurer de corriger la vélocité afin de freiner l'objet au sol. Or, avec l'intégration Verlet, cette situation est plus facile à traiter puisqu'il suffit de corriger la position. Implicitement cela a pour effet immédiat de corriger la vitesse puisque celle-ci est déduite avec la différence entre la dernière position et la position actuelle. De cette manière, la vitesse et la position sont toujours bien synchronisées tout au long de la simulation[Jak01].

4 Simulation de tissu par satisfaction de contraintes

Maintenant que les principles de base sont acquis, intéressons nous plus précisément à la simulation de tissus avec une technique proposée par [Jak01]. Au lieu de recourir à des ressorts pour préserver la forme des tissus, on utilise des contraintes de distance. Ces dernières peuvent être vues comme des tiges reliant deux particules. À chaque itération de la simulation, on va tenter de préserver toutes les contraintes de distance en les corrigeant de façon itérative. À la figure 6, on apperçoit les règles appliquées. Le code nécessaire à cette étape est présenté à la figure 7. Dans le cas où les particules n'ont pas toutes la même masse, on peut adapter le code afin d'en tenir compte. Au lieu de multiplier par 0.5, il suffit de multiplier par un ratio inversement proportionnel à la masse de la particule à corriger.

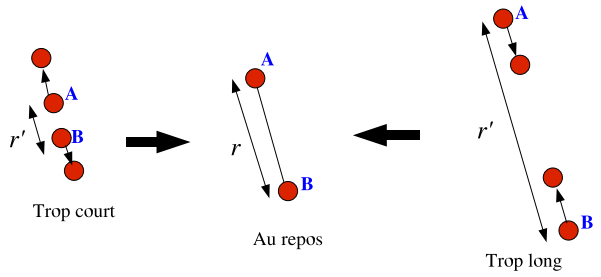

FIG. 6 – Contrainte de distance

```
void satisfaireContrainte(Contrainte c){
  vector v = c.b.pos - c.a.pos;
  double d = v.norm(); // distance
  d -= c.distance_cible;
  c.a.pos += v * d * 0.5;
  c.b.pos -= v * d * 0.5;
}
```

FIG. 7 – Code de satisfaction de contrainte

Dans un cas réel ayant plusieurs contraintes de distance, il faut porter une attention particulière à la façon d'effectuer les calculs afin d'éviter que l'ordre du parcours des contraintes ait un impact sur le résultat de la simulation. Pour ce faire, à chaque contrainte, on calcule des corrections qui doivent être appliquées aux positions des particules. Après avoir passé à travers toutes les contraintes, pour chacune des particules, on applique la moyenne de toutes les corrections. Enfin, pour donner de bons résultats, il faut appliquer cette procédure plusieurs fois. En pratique, 40 à 55 itérations de satisfaction de contrainte par itération de simulation donnent de bons résultats. Le code associé à cette partie est à la figure 8.

L'avantage de cette méthode est qu'elle est très simple et très rapide. Contraintement aux réseaux de masse-ressorts, la satisfaction de contrainte de distance ne requiert pas d'ajustement de paramètres comme des coefficients de tension et des facteurs d'amortissement qui ne sont pas toujours facile à paramétrer correctement. Ici, pour modifier la rigidité, on n'a qu'à jouer avec le nombre d'itérations. Plus il est grand, plus les tiges seront rigides. Par contre, plus on désire de la

```
int n; // nb de particules
int m; // nb de contraintes
Particule  parts[n];
Contrainte contraintes[m];
void satisfaireContraintes(){
  for(int i=0;i<55;i++){
    for(int j=0;j<n;j++){
      parts[j].corr = vector(0,0,0);
      parts[j].nb = 0;
    }
    for(int j=0;j<n;j++)
      satisContrainte(contraintes[j]);
    for(int j=0;j<n;j++){
      parts[j].pos +=
        parts[j].cor / parts[j].nb;
    }
  }
}
void satisContrainte(Contrainte c){
  vector v = c.b.pos - c.a.pos;
  double d = v.norm(); // distance
  d -= c.distance_cible;
  c.a.correction += v * d * 0.5;
  c.a.nb++;
  c.b.correction -= v * d * 0.5;
  c.b.nb++;
}
```

FIG. 8 – Code de satisfaction de contraintes

rigidité, plus le temps de calcul requis pour la simulation sera important. Il est à noter que la technique discutée ici ainsi que le code à la figure 8 ne sont pas optimisés. Certaines optimisations possibles sont présentées dans [Jak01].

5 Lois de physique utilisées

Regardons de plus prêt les éléments à considérer pour implanter les lois de physique mécanique. Ces lois seront implantées dans la fonction calculerForces().

5.1 Gravité

La gravité est la force la plus facile à calculer. Dans nos expérimentations, nous avons utilisé une force gravitationnelle fixe $g = 9.80655N/kg$, soit celle mesurée au niveau du sol sur terre.

5.2 Friction avec l'air

La force résultant de la friction avec l'air est sans doute la force la plus difficile à évaluer avec précision. Le mouvement d'un objet solide à travers un fluide ou un gaz comme l'air est très complexe. En aérodynamique, on illustre souvent ce phénomène à l'aide de courbes de pression [Sme97] comme montré à la figure 9.

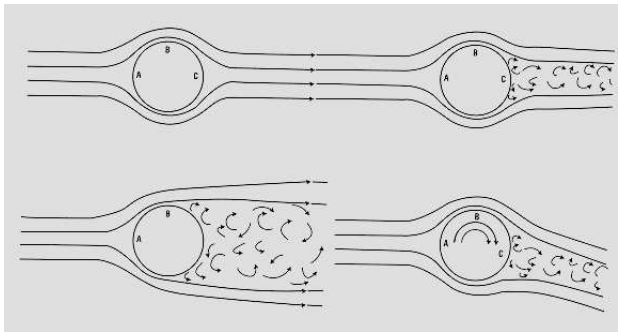

FIG. 9 – Courbes de pression d'air

Afin de simuler des objets se déplacant dans l'air, on utilise souvent des coefficients de traînée et de portance pouvant être obtenues par l'étude approfondie des courbes de pression. Le coefficient de traînée représente la résistance d'avancement dans la direction de l'objet tandis que le coefficient de portance représente la force de soulèvement de l'objet dans la direction perpendiculaire à sa trajectoire de déplacement. Cet technique est en autre très utilisée dans les simulations de sports de balle, comme le golf (voir figure 10) [Jor99, WH91] ainsi que des les simulateurs de jeux d'avion [Bou01].

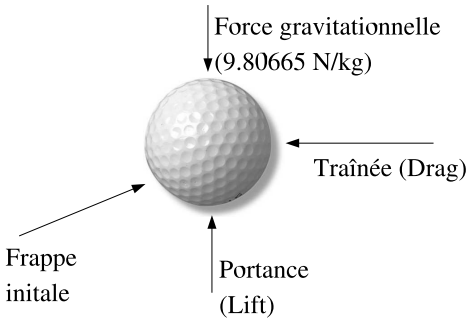

FIG. 10 – Traîné et portance

Or l'utilisation de ces coefficients n'est pas appropriée au cas présent. Généralement, ces coefficients sont calculés pour un objet ou une partie d'objet en entier, comme une aile d'avion ou une balle. Dans notre cas présent, on veut simuler le mouvement de la toile. Ainsi, pour des objets flexibles, comme la toile d'un deltaplane ou d'un cerf-volant, il n'est pas possible de précalculer des coefficients pour chaque morceau de la toile, puisque la forme de l'objet est en constante déformation. De plus, les coefficient de traînée et de portances dépendent de nombreux facteurs, comme la forme[NAS], l'angle d'attaque, la vitesse, la pression atmosphérique, etc.

Puisque nous avons comme objetif de faire une simulation

réaliste en temps réel, il n'est pas nécessaire d'avoir une simulation exact. La force de friction avec l'air peut être calculée par petit morceau de surface, de la même façon que les animations de drapeaux et de tissus sont conçues. La toile est décomposée en plusieurs triangles. Pour chacun de ces triangles, on calcule une force approximative de son impact avec l'air. Cette force est directement proportionnelle à la vitesse avec laquelle l'air frappe le morceau de toile, représenté par la variable $wrel$, soit la vitesse relative entre la vitesse du vent et celle du morceau de toile en question. De plus, la force recherchée est aussi proportionnelle à l'aire de la surface projectée sur le plan perpendiculaire à la vélocité relative du vent. Enfin, cette force est appliquée dans la même sens que vecteur normal, comme montré à la figure 11 et elle est calculée selon les formules suivantes.

$N = (AB)$ x (BC)
$WreldotABxCD = N * wrel$
$Force = airDensity * WreldotABxCD^2/|N|$

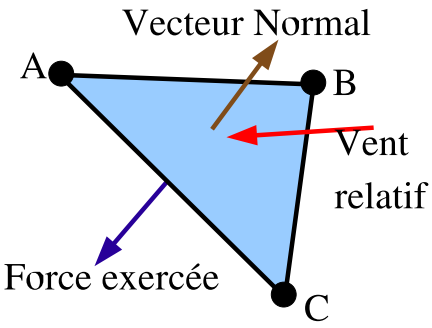

FIG. 11 – Force de friction avec l'air

6 Conception d'un deltaplane

6.1 Structure

Ayant à présent tous les éléments pour réaliser une simulation physique, attaquons nous au deltaplane. Afin de simplifier notre modèle, nous allons considérer le deltaplane comme une structure rigide sur laquelle une toile triangulaire est fixée. Cette dernière est décomposée en un maillage de particules et de petits triangles. La figure 12 montre un deltaplane avec une résolution de quatre particules par côté. Il est à noter que les particules de cette structure ont toutes une masse identique dont la somme de l'ensemble donne la masse d'un deltaplane typique, soit entre 20 et 30 Kg.

Afin d'imposer une structure rigide au deltaplane, deux particules virtuelles ont été placées en haut et en bas du triangle. Ces particules sont chacune reliée aux particules situées sur les côtés et au centre (voir figure 13). Ainsi,

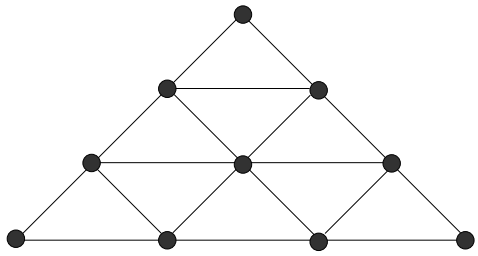

FIG. 12 – Base triangulaire du deltaplane

les barres latérales et la barre centrale (appelé quille) sont modélisées. Les particules restantes, formant le tissu de la toile, restent flexible avec une certaine tension excercée par la fixation des côtés.

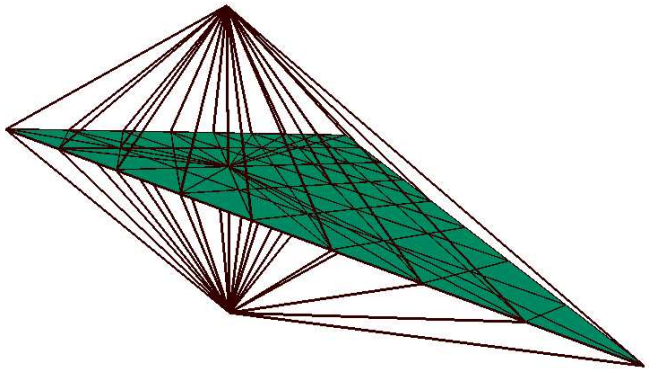

FIG. 13 – Deltaplane avec contraintes de rigidité

6.2 Contrôle

Le pilote d'un deltaplane arrive à le diriger en se servant de la barre à laquelle il est fixé afin de déplacer la position relative de son poids par rapport au centre de la structure du deltaplane. Par exemple, en tirant sur la barre transversale, le pilote peut faire piquer du nez le deltaplane pour accélérer sa descente. À l'inverse, en poussant sur la barre, il déplace sa masse à l'arrière du deltaplane, ce qui a pour effet de freine sa descente. Par contre, s'il perd trop de vitesse, il peut descendre trop rapidement à la verticale.

Pour notre simulateur, nous avons représenté un pilote virtuel qui peut contrôler le deltaplane de la même façon qu'un deltaplane réel, soit en déplacant sa masse. Pour y arriver, nous avons introduit une particule, juste au-dessous du deltaplane, ayant une masse très importante, soit d'environ celle d'une personne normale (55 à 70 Kg). Ce pilote virtuel est relié aux côtés à l'aide de tiges (contraintes de distance). La figure 14 illustre un exemple sur un deltaplane simplifié, où

au centre, le grand cercle représente le pilote virtuel et les tiges de contrôles sont en gros trais.

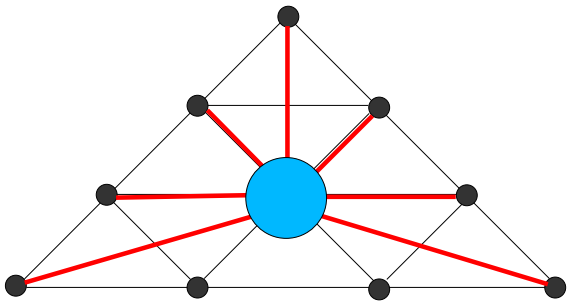

FIG. 14 – Tiges de contrôle

Contrairement aux tiges de la structure du deltaplane, les distances cibles (au repos) des tiges de contrôle peuvent être modifiées après leur création. Initialement, les tiges tendent à maintenir le pilote au centre du deltaplane. Quand l'usager pilote le deltaplane, on modifie les longueurs au repos de ces tiges de façon à excercer une tension qui déplacera la position relative du pilote sous le deltaplane. Ainsi, pour amorcer une descente rapide, l'usager peut appuyer sur la flèche vers le haut (↑), ce qui aura pour effet de modifier de réduire la longeur des trois contraintes au haut et d'allonger les quatres contraintes d'en bas. Au cours des prochaines itérations d'intégration, le pilote sera progressivement déplacé à l'avant du deltaplane, comme montré à la figure 15. De plus, puisque la masse du pilote est largement supérieure à celle du deltaplane, le déplacement final sera plus important sur la structure du deltaplane que sur le pilote lui-même. De cette façon, on reproduit avec réalisme la possibilité de modifier facilement l'angle de piqué du deltaplane.

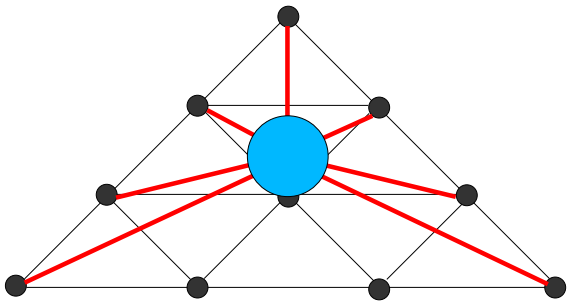

FIG. 15 – Contrôle vers l'avant du deltaplane

6.3 Implémentation et expérimentations

Afin de tester notre conception, nous avons implémenté un noyau physique supportant des contraintes de distance comme proposé par [Jak01]. L'implémentation a été faite en C++ et les rendus de simulation sont réalisés à l'aide d'OpenGL. Afin de modéliser le sol, nous avons utilisé un terrain de hauteur (heightfield en anglais) d'une résolution de 256 par 256. Une simple gestion de collision a été utilisée afin de maintenir l'altitude des particules supérieur ou égale à celle du terrain. Une capture de la simulation d'un deltaplane est illustrée à la figure 16. Ce dernier a une résolution de huit (8) particules par côté. Au total, ce deltaplane est construit à l'aide de 49 particules, 198 contraintes de distances et 64 surfaces triangulaires. Tous ces éléments peuvent être simulé à 100 itérations par seconde sur notre machine de test. Cette dernière était un PC équipé d'un processeur Pentium 4 condensé à 2GHz.

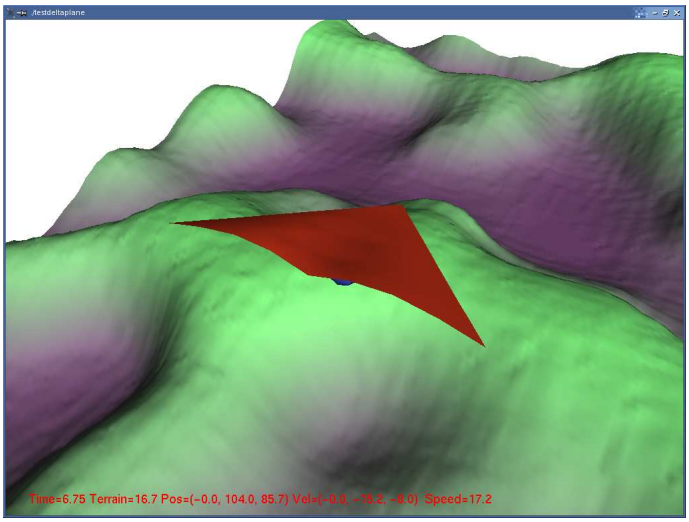

FIG. 16 – Deltaplane simulé

7 Conception d'un cerf-volant

Le cerf-volant que nous présenterons ici est de type delta. Comme son nom l'indique, la conception de ce dernier est à la base similaire au deltaplane présenté à la section précédente.

7.1 Structure

Le cerf-volant delta est composé de trois parties principales : l'aile principale formée d'un grand triangle, la poutre principale (keel) formée d'un petit triangle situé perpendiculairement sous l'aile principale et une corde.

Tout d'abord, pour le triangle principal du cerf-volant, nous avons repris la même base que le deltaplane illustré à la figure 12. Tout comme le deltaplane, nous avons introduit deux

particules supplémentaires ayant comme unique fonction le maintien d'une structure rigide sur les côtés et au centre.

La première différence du cerf-volant de type delta par rapport au deltaplane est l'introduction d'un petit triangle (keel) sous l'aile principale. À ce triangle, une corde est aussi attachée. Ce triangle est conçu de façon similaire au triangle principal, mais à une résolution deux fois moindre. De plus, il ne commence pas immédiatement au devant du cerf-volant, mais à un endroit légèrement reculé par rapport à la pointe du cerf-volant (figure 17). Tout comme nous l'avons vu pour le deltaplane, la somme des particules du cerf-volant donnent une masse similaire à une cerf-volant réel.

FIG. 17 – Cerf-volant en delta

7.2 La corde

La corde du cerf-volant est elle aussi composée de particules. Contraitement aux deux autres triangulaires du cerf-volant, la corde n'est pas un maillage en deux dimensions, mais une simple série de particules reliées l'une à la suite des autres à l'aide de contraintes de distance. L'extrémité au sol de la corde a une caractéristique spéciale. Puisque sa position doit être fixe, elle est repositionnée au même endroit à toutes les itérations de la simulation. L'autre extrimité, du côté du cerf-volant, est rattachée au keel. De plus, afin d'éviter un étirement trop grand de la corde du keel, des tiges ont été doublées sur la corde afin de la rendre plus rigide. Il est aussi à noter que les particules formant la corde ont une masse nettement inférieur à celles composant le cerf-volant.

7.3 Expérimentations

Afin de réaliser nos expérimentations, nous avons repris la même base que pour le deltaplane. De plus, le simulateur permet un contrôle de la corde à l'aide des flèches du clavier. La figure 18 montre le résultat d'une capture lors d'une simulation. Tout comme le cerf-volant, le noyau physique implanté permet une simulation en temps réel. Le cerf-volant simulé a une résolution de 14 particules par côté sur le triangle principal, 6 particules par côté du triangle formant le keel, ainsi que 14 particules pour la corde. Au total, nous avons 152 particules, 458 contraintes de distance et 221 surfaces triangulaires.

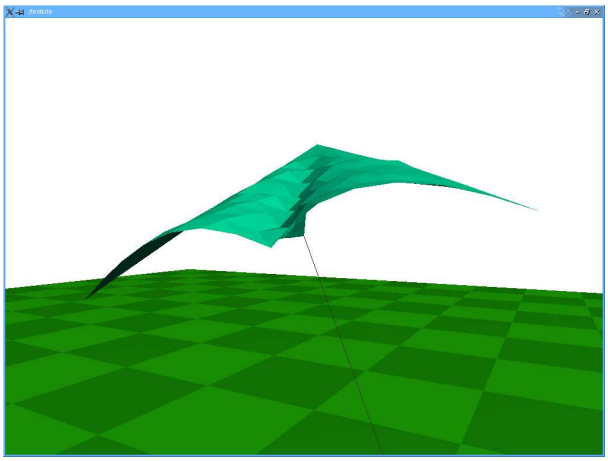

FIG. 18 – Cerf-volant simulé

8 Limitations - cas du parachute

La technique de simulation présentée dans ce papier ne tient aucunement compte de la viscosité de l'air. Ainsi, la pression résultant du contact de l'air avec la surface d'un objet ne peut pas être modélisée. Tenir compte de ce phénomène augmenterait de façon significative le temps de calcul requis pour faire nos simulations et briserait notre objectif de temps réel. Par ces raisons, certains objets ne peuvent pas être modélisés par une simple conception inspirée de la réalié. Le cas du parachute est un très bon exemple d'un objet volant exploitant bien le phénomène d'accumulation de la pression d'air sous sa toile. Or, en ajoutant certaines contraintes supplémentaires, il est possible de simuler de tels objets.

8.1 Structure

Le type de parchute sur lequel nous avons fait des expérimentations est circulaire. C'est un très ancien modèle de parachute, mais il est un exemple approprié. L'objet est représenté par un maillage formant un cercle divisé en plusieurs anneaux décomposés en triangles. La figure 19 montre un cercle divisé en six étages. Tout comme le deltaplane, la personne virtuelle est modélisée à l'aide d'une grosse particule attachée au moyen de cordes sous le parachute. Ces cordes sont construites de la même façon que la corde du cerf-volant à la section précédente, mais à une résolution plus faible.

8.2 Problèmes

Dès les premières secondes de simulation, on peut constater qu'il y a quelque chose d'incorrect dans la modélisation. Un rapide coup d'oeil au parachute sur le côté gauche de la figure 20 permet de constater un problème. Contrairetement à celui de droite, il n'arrive pas à se gonfler.

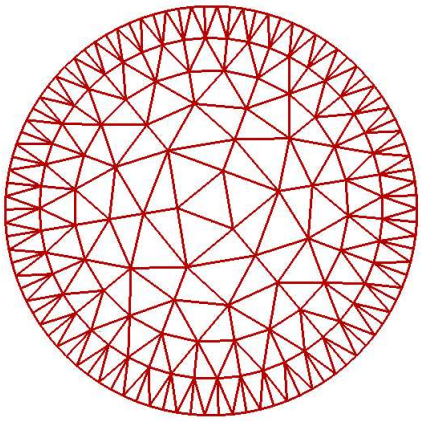

FIG. 19 – Maillage du parachute

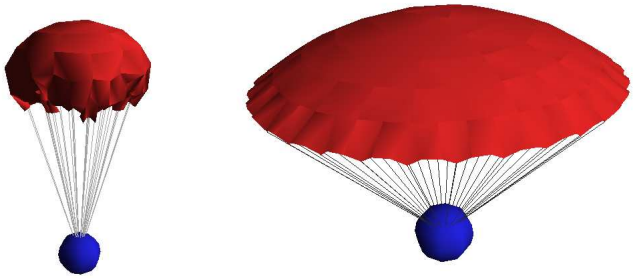

FIG. 20 – Parachutes

Comme indiqué précédement, ce problème vient du fait qu'on ne modélise pas les lignes de pression sous les objets simulés. Afin de mieux comprendre le problème, faisons un parallèle avec un sac de plastique exposé au vent. La pression d'air s'accumulant dans le sac permet aux parrois de se gonfler et s'ouvrir, même s'elles sont dans le même sens que le vent. Avec un parachute réel, la même chose se produit. La pression d'air accumulée sous le parachute lui permet de rester ouvert. Or notre modèle de physique ne peut modéliser cela. Puisque les cordes sont fixées sur la circonférence du parachute, le tour de ce dernier est rapidement tiré vers l'intérieur en raison de la grande force excercée par la personne attachée sous le parachute.

8.3 Solution

Sans avoir à modifier notre modèle physique en simulant mieux les phénomènes aérodynamiques, on peut recourir à une astuce nous permettant de contourner le problème. La solution proposée consiste à ajouter des contraintes de distance dans le dernier anneau de particules du parachute. Ces contraintes ont la même fonction que les contraintes ajoutées au deltaplane et au cerf-volant les permettant d'avoir une structure rigide. Enfin, dans le cas du parachute, ces contraintes permettent de créer un cerceau de rigidité tout le tour du parachute. Ainsi, même si la personne virtuelle tire le tour vers le bas et le centre, le parachute conserve sa forme ouverte. Bien que cette astuce ne soit pas fidèle à la réalité, elle donne un résultat avec un bon niveau de réalisme.

9 Conclusion

La simulation d'objets volant basés sur des toiles est un problème très complexe lorsqu'on désire le faire avec haute précision. Par contre, en ayant un objectif de faire simulation approximative, nous avons démontré à l'aide de trois exemples (deltaplane, cerf-volant et parachute) qu'il était possible de le faire. L'utilisation de l'intération Verlet combinée avec une méthode de satisfaction de contrainte de distance permet d'obtenir de bons résultats sans être gourmant en temps de calcul. Cette méthode peut être appliquée dans plusieurs domaines, comme les jeux vidéo, dans lesquels la précision n'est pas prioritère et que le temps alloué à la simulation physique est souvent très limité. Les codes sources et binaires de nos expérimentations sont disponible sur notre site Web (http ://planiart.usherb.ca/˜eric/simdelta/).

Références

[Bou01] David Bourg. *Physics for Game Developers*. O'Reilly, 2001.

[Jak01] Thomas Jakobsen. Advanced character physics. In *Game Developer's Conference*, San Jose, USA, 2001.

[Jor99] Theodore P Jorgensen. *The physics of golf*. Springer : AIP Press, 2nd edition, 1999.

[NAS] NASA. Shape effets on drag. Nasa Web Site. http ://www.grc.nasa.gov/WWW/K-12/airplane/shaped.html.

[OZH00] James F. O'Brien, Victor B. Zordan, and Jessica K. Hodgins. Combining active and passive simulations for secondary motion. *IEEE Computer Graphics and Applications*, 20(4) :86–96, July/August 2000.

[Pro95] Xavier Provot. Deformation constraints in a mass-spring model to describe rigid cloth behavior. In *Graphics Interface*, pages 147–154, Quebec, Quebec, Canada, May 1995.

[Sme97] Frederick O. Smetana. *Introduction Aerodynamics and Hydrodynamics of Wings and Bodies : A Software-Based Approach*. AIAA, 1997.

[Ver67] Loup Verlet. Computer experiments on classical fluids. *Phys. Rev.*, 1(5) :98–103, July 1967.

[WH91] J. Wejchert and D. Haumann. Animation aerodynamics. In *Siggraph*, pages 19–22, July 1991.