Transmission progressive de modèles triangulés sur le réseau

TRENTINI Mario DEVILLERS Olivier GANDOIN Pierre-Marie

INRIA, BP93, F-06902 Sophia-Antipolis.

Résumé

Nous présentons une implantation d'une méthode de compression de maillage 3D basé sur des kd-arbres [1, 2] , dans un contexte réseau.

1 Introduction

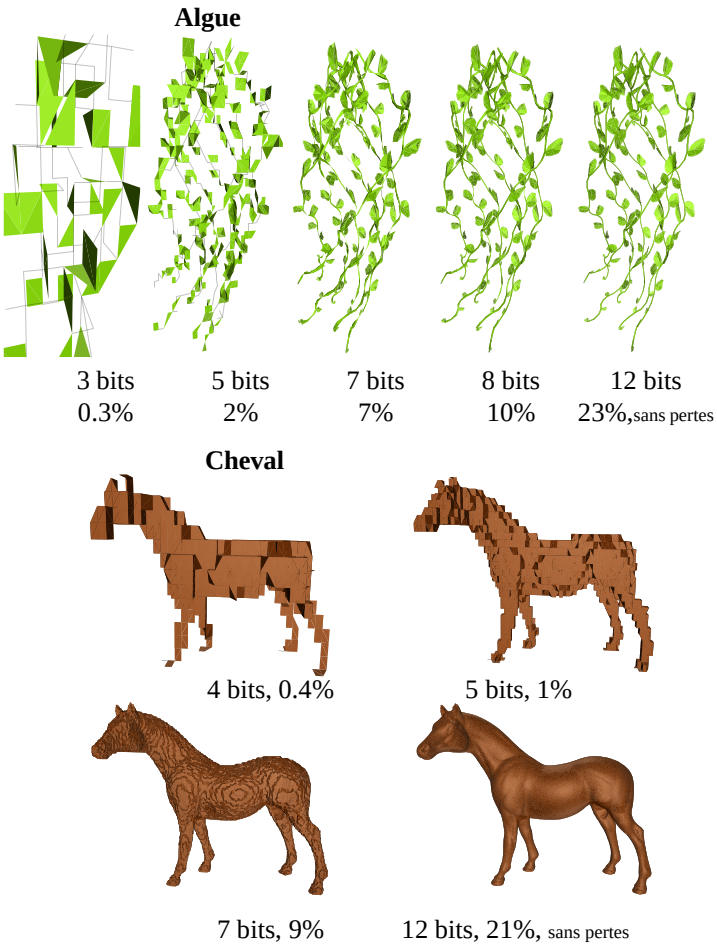

FIG. 1 – *Exemples de décompressions progressives.*

En quelques années, les maillages ont conquis une position prédominante parmi les différents modes de représentation informatique d'objets géométriques. Plus particulièrement, les maillages à base de simplexes — les triangles pour la représentation de surfaces plongées en 3D, les tétraèdres pour la représentation de volumes — semblent être actuellement les plus répandus. Le développement rapide des applications manipulant ces structures géométriques dans des domaines aussi divers que le calcul par éléments finis, la simulation chirurgicale, ou les jeux vidéo a très vite soulevé le problème d'un codage efficace et adapté à la visualisation. L'expansion du World Wide Web, qui nécessite une représentation compacte et progressive des données pour garantir la convivialité de l'interface homme-machine, a fini de conférer à ce problème une place centrale dans la recherche en informatique.

Ainsi, depuis 1995, de nombreux algorithmes ont été proposés pour la compression de maillages triangulaires, en utilisant le plus souvent l'approche suivante : les sommets du maillage sont codés dans un ordre établi pour contenir partiellement la topologie (ou connectivité) du maillage [3, 4, 5, 6]. Parallèlement, quelques règles simples permettent de prédire la position du sommet courant à partir des positions de ses voisins qui ont déjà été codés. Nous avons développé une méthode donnant plutôt la priorité à la compression des positions des sommets [1, 2] et traitant la connectivité dans un second temps. Nous avons proposé une méthode de codage progressif d'objets triangulés qui ne réclame pas que les objets soit *manifold* et donne de bons résultats tant sur des objets bien structurés que sur des soupes de triangles. Les taux de compression obtenus se positionnent avantageusement par rapport aux méthodes progressives actuelles les plus efficaces : par exemple, pour le cas particulier des maillages triangulaires surfaciques, des taux moyens autour de 3,6 bits par sommet sont atteints sur des modèles usuels pour le codage de la connectivité.

Dans cet article, après une brève présentation de la méthode de compression utilisée [1, 2], nous décrirons une application de type client/serveur utilisant cette méthode de compression pour la transmission de données sur le réseau.

2 Algorithme de compression

2.1 Compression des coordonnées

L'algorithme utilisé pour la compression est basé sur l'utilisation d'un kd-arbre comme décrit dans la figure 2.

Les points sont d'abord arrondi sur des coordonnées entières et placés dans une boite englobante, ces renseignements ainsi que le nombre de points total à coder sont donnés en entête du fichier. Ensuite la boite englobante est récursivement subdivisé en deux, et à chaque subdivision on

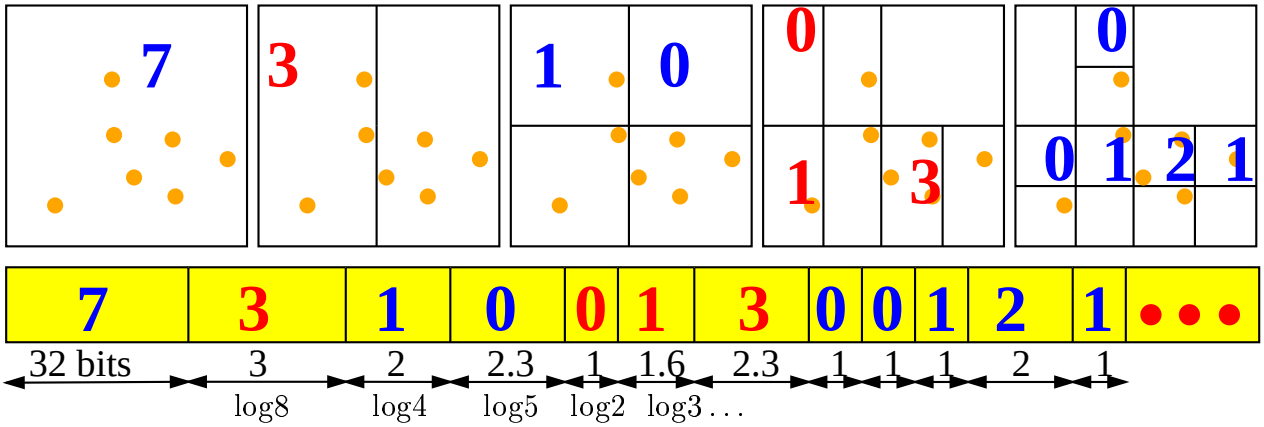

FIG. 2 – *Le codeur géométrique sur un exemple 2D*

indique dans le code le nombre de points allant dans la première sous-boite. Si le nombre de points dans une boite est k alors le nombre de points dans cette sous-boite est compris entre 0 et k et peut donc être codé sur $\log(k+1)$ bits. En d'autre termes, on ne paye que $\log(k+1)$ bits pour envoyer un bit de coordonné de k points d'où un gain par rapport aux coordonnées brutes des points. De plus les coordonnées des points sont transmises progressivement et s'affinent au fur et à mesure de la transmission, aux étapes intermédiaires les points dans une même boite sont confondus [1].

2.2 Compression de la connectivité

Cette première partie concerne les coordonnées des sommets, on peut considérer que chaque boite non vide est représentée par un seul point en son centre. Lors d'une subdivision, un point est soit éclaté en deux soit seulement déplacé. La décompression peut alors être vue comme un ensemble de points, qui au fur et à mesure des éclatement et des déplacements converge vers l'ensemble final.

Pour coder maintenant la connectivité des sommets de cet ensemble final, on va commencer par définir une connectivité sur tous les ensembles de points apparaissant au fur et à mesure de la décompression. Pour cela, on commence par utiliser la connectivité donnée pour le modèle sur l'ensemble de points final, puis on joue la séquence de décompression à rebours. À chaque déplacement de sommet, la connectivité peut suivre sans problème, et lorsque deux sommets sont réunis en un seul, la nouvelle connectivité est obtenue par les opérations suivantes : la contraction d'arêtes [7] (*edge collapse*) dans les cas faciles où le genre de la surface représentée n'est pas modifiée et d'unification de sommets [8] (*vertex merge*) dans les cas plus complexes. Une fois la séquence complètement jouée à rebours on dispose d'une connectivité pour chacun des ensembles de points rencontrés au cours du processus. On va maintenant rejouer la séquence dans l'ordre, mais à la suite du code indiquant le nombre de points allant dans la première sous-boite, on va donner un code permettant de déterminer l'évolution de la connectivité. Ce code décrit soit une d'expansion d'arête soit une séparation de sommet, les opérations réciproques le la contraction d'arête et de l'unification de sommets ; on y explique comment les différentes arêtes incidentes au sommet représentant la boite se divisent pour s'attacher à l'une, à l'autre ou aux deux sous-boites après la division [2].

2.3 Prédiction

Lors de la compression des coordonnées, lorsqu'une boite contenant k points est subdivisée, il est possible d'attacher des probabilités aux $k+1$ possibilités, en utilisant du codage entropique [9] on bénéficie alors de ces probabilités pour obtenir un code plus court : les cas les plus probables sont codés sur moins de de bits et les cas les moins probables sur plus.

La prédiction donne en particulier d'excellents résultats pour le codage de la connectivité où le code moyen pour une expansion d'arête n'est que de 0.2 bits en utilisant des critères de prédiction simples basés sur la position des points.

2.4 Résultats

Tout en permettant la progressivité, on obtiens de très bons résultats de compression, par exemple sur les modèles de la figure 1 le cheval (19851 sommets) est codé avec 16.4 bits par sommets pour les coordonnées et 3.9 bits par sommets pour la connectivité. Pour des modèles avec une connectivité plus pauvre comme l'algue (16784 sommets) le code est de 16.4 bits par sommets pour les coordonnées et 8.5 bits par sommets pour la connectivité. Pour les modèles à faible connectivité, c'est à notre connaissance la seule technique permettant d'obtenir un tel taux de compression.

3 Réseau

Nous avons une méthode de compression de structures tridimensionnelles dont le décodage est progressif. Nous allons, dans cette seconde partie, présenter une application

architecturée *client / serveur* qui utilise cet algorithme de codage pour la transmission des données sur le réseau.

L'application se compose de deux programmes : un serveur de scène et une visionneuse de scène. Le serveur possède des objets tridimensionnels codés en utilisant l'algorithme présenté et des scènes composées de ces objets.

La visionneuse demande au serveur une scène, ce dernier envoie la définition de la scène et les données compressées correspondant aux objets qui composent la scène.

3.1 Présentation de la visionneuse

L'algorithme de compression de structures tridimensionnelles trouve naturellement sa place au sein d'une telle application. La transmission sous forme compressée des objets vers la visionneuse par le réseau permet une économie de bande passante ainsi qu'un gain de temps de téléchargement. De plus, la progressivité du décodage, atout appréciable de l'algorithme, est exploitée à tous les étages par la visionneuse :

– lors du téléchargement, puisque tous les objets sont téléchargés et décodés progressivement pour permettre à l'utilisateur une vision immédiate de la scène. Cette dernière se raffine au fur et à mesure du téléchargement.
– lors de l'affichage car la représentation de la scène exploite la multirésolution de l'algorithme de décompression : le client adapte le niveau de détails de chaque objet à l'écran en fonction de sa position par rapport à l'observateur afin de limiter le travail de rendu et donc de conserver un nombre d'images par seconde acceptable.

De plus la priorité du téléchargement des objets dépend du point de vue de l'utilisateur et du niveau de détail déjà obtenu. Les objets dont les faces à l'écran sont les plus grandes seront téléchargés en priorité.

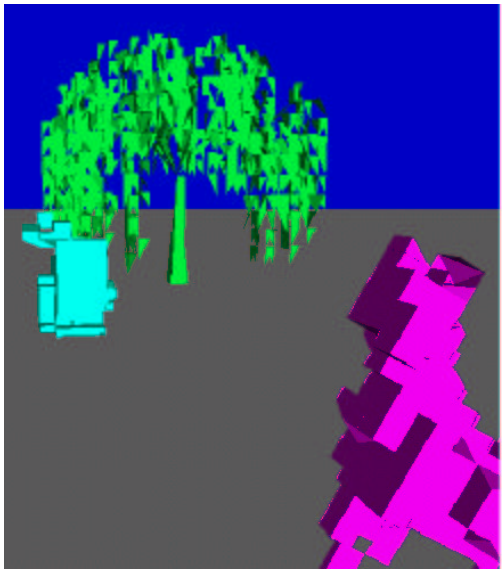

FIG. 3 – *Vue aux détails grossiers*

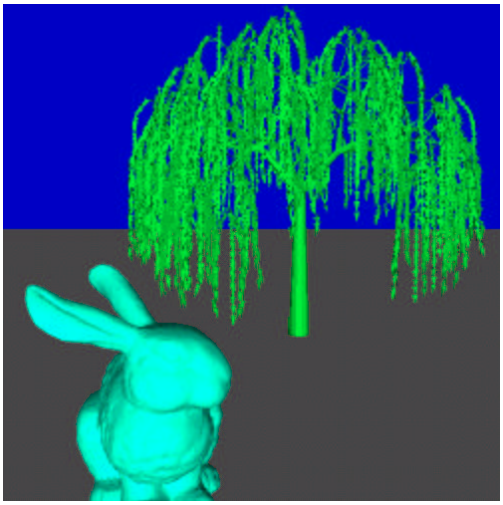

FIG. 4 – *Vue aux détails fins*

La visionneuse offre à l'utilisateur une vue subjective de la scène en utilisant un rendu *OpenGl*. Tous les déplacements classiques sont possibles, l'utilisateur peut aussi contrôler le téléchargement de la scène (téléchargement en continue ou en mode pas à pas) et le niveau de détails maximal à utiliser pour l'affichage des objets. La visionneuse reçoit rapidement, de la part du serveur, les données nécessaires pour l'affichage d'une scène aux détails minimums (figure 3). L'utilisateur peut se déplacer dans cette scène pendant son téléchargement, les objets dans le champ de vision sont raffinés en priorité (figure 4).

3.2 Transmission entre le client et le serveur

Le serveur de scène et le client (la visionneuse) communiquent entre eux par l'intermédiaire d'un réseau `IP`. Le choix entre les deux modes de transports `UDP` et `TCP` s'offre pour leur communication.

Le protocole `UDP`[1] implante un service de paquets datagrames non fiable, non connecté. Les paquets peuvent être perdus ou arrivés désordonnés.

Le protocole `TCP`[2] établie une connexion *full-duplex* fiable orientée flux. `TCP` crée un canal de communication entre deux processus qui garantit l'arrivée dans l'ordre des données envoyées.

Le protocole `UDP`, du fait de son caractère sans connexion, a pour principal avantage d'être simple du point de vue communication et donc d'avoir un faible *overhead*. Sur un réseau local par exemple, il est plus efficace que le protocole `TCP` car on part du principe que les liaisons physiques sont suffisamment fiables et les temps de transmissions suffisamment court pour qu'il n'y ait pas de perte ou d'inversion de paquets. Le protocole `UDP` est également adapté pour les services ne nécessitant pas de contrôle de flux et où le temps des paquets est prédominant. Au contraire, le pro-

[1] User Datagram Protocol (RFC 768)
[2] Transmission Control Protocol (RFC 793)

tocole TCP convient aux applications effectuant des transferts de données sur un réseau peu fiable.
Dans notre cas, le décodeur a besoin de tous les octets dans l'ordre pour décoder correctement les données. En effet, il n'y pas de redondance dans le codage et il n'y a aucun mécanisme pour supporter la perte de paquet. Le protocole UDP n'assurant pas cette fiabilité, n'est pas à privilégier, de plus le client et le serveur sont amenés à être reliés par un réseau lent et peu fiable, une communication utilisant TCP s'impose donc naturellement.

3.3 Multiplexage des objets

Le serveur de scène doit envoyer en parallèle au client plusieurs fichiers en utilisant TCP. Pour cela différentes méthodes sont envisageables. Il est par exemple possible d'ouvrir une connexion TCP pour chaque objet à transmettre et de contrôler le flux à travers chacune de ces connexions. Une telle solution serait très coûteuse en terme réseau car elle utilise un grand nombre de connexions TCP (de l'ordre du nombre d'objets dans la scène). Or, l'établissement d'une connexion TCP coûte cher et doit être évité autant que possible.
La solution retenue ici est d'utiliser une seule connexion TCP pour l'envoi de tous les objets. Il faut donc multiplexer plusieurs flux de données en un seul. Pour cela on construira une sorte de *trame* qui contiendra un morceau de chaque objet. Le serveur va envoyer par *trame* une partie de chaque objet, chaque *trame* sera reçue par le client qui sélectionnera la partie associée à chaque objet. Une grande économie, surtout pour les scènes possédant de nombreux objets, est ainsi réalisée.

3.4 Gestion de la priorité des objets

La mise en oeuvre du serveur de scène s'est attachée à conserver une généralité et une simplicité dans le serveur de scène. Le serveur de scène doit rester un serveur de fichier, certes un peu évolué car il envoie plusieurs fichiers en même temps et adapte la priorité de chacun, mais ne doit en aucun cas interpréter les objets qu'il envoie pour décider de leur priorité. C'est d'ailleurs pour l'instant impossible car les ressources du décodage d'un objet dépassent de beaucoup la charge acceptable d'un serveur.
L'intelligence en terme de priorité doit se trouver au maximum chez le client. Le serveur utilise un schéma simple pour allouer la taille des blocs dans la trame.
Lorsque un client demande une scène, la première trame du serveur contient des blocs de tailles identiques pour chaque objet de la scène. Cette première trame permet au client d'afficher et de placer tous les objets dans la scène. Dans les trames suivantes les tailles allouées pour chaque objet sont pondérées par la taille restante à envoyer (priorité par défaut).
Le client peut demander un changement de la proportion des objets dans la trame. Ce changement peut se faire soit en absolu (le client demande au serveur le nombre d'octet à envoyer pour chaque objet) soit proportionnellement (le client demande le pourcentage à utiliser dans la trame pour chaque objet).
Le serveur, peut néanmoins modifier la composition de la trame. Par exemple lorsque la taille totale comptabilisée des données à envoyer est en deçà d'un minimum, alors le serveur passe en priorité par défaut, à savoir une taille pour chaque objet proportionnelle à la taille restante à envoyer. De plus lorsqu'un objet a été complètement envoyé, le serveur réajuste la taille des blocs correspondant à chaque objet en conservant la proportion et la taille totale de la trame. Ces deux mesures visent à éviter une situation où la trame posséderait un faible nombre d'octets utiles.

Remmerciements

Merci à Pierre Alliez pour son aide lors de la rédaction de ce document et à Wallid Dabbous et aux autres membres du projet Planete à l'INRIA pour leur concours sur les aspects réseau.

Références

[1] O. Devillers et P.-M. Gandoin. Geometric compression for interactive transmission. Dans *IEEE Visualization 2000 Conference Proc.*, 2000.

[2] P.-M. Gandoin et O. Devillers. Progressive lossless compression of arbitrary simplicial complexes. *ACM Transactions on Graphics*, 21 :372–379, 2002. SIGGRAPH '2002 Conference Proceedings.

[3] C. Touma et C. Gotsman. Triangle mesh compression. Dans *Graphics Interface 98 Conference Proc.*, pages 26–34, 1998.

[4] J. Rossignac. Edgebreaker : Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, pages 47–61, 1999.

[5] D. Cohen-Or, D. Levin, et O. Remez. Progressive compression of arbitrary triangular meshes. Dans *IEEE Visualization 99 Conference Proc.*, pages 67–72, 1999.

[6] P. Alliez et M. Desbrun. Progressive compression for lossless transmission of triangle meshes. Dans *SIGGRAPH 2001 Conference Proc.*, 2001.

[7] H. Hoppe, T. DeRose, T. Duchamp, J. McDonald, et W. Stuetzle. Mesh optimization. Dans *SIGGRAPH 93 Conference Proc.*, 1993.

[8] J. Popović et H. Hoppe. Progressive simplicial complexes. Dans *SIGGRAPH 97 Conference Proc.*, 1997.

[9] I.H. Witten, R. Neal, et J.G. Cleary. Arithmetic coding for data compression. *Communications of the ACM*, 30(6) :520–540, 1987.