Textures projectives à calques dépendantes du point de vue

Alex Reche, George Drettakis

Textures projectives à calques dépendantes du point de vue

Alex Reche-Martinez[1,2] **and George Drettakis**[1]

[1]REVES/INRIA Sophia-Antipolis,
http://www-sop.inria.fr/reves
[2]Centre Scientifique et Technique du Bâtiment,
http://www.cstb.fr
{Alex.Reche,George.Drettakis}@sophia.inria.fr

Résumé : *La capture et le rendu des scènes réelles dans le contexte des environnements virtuels immersifs sont encore des tâches très difficiles. Nous présentons une nouvelle approche, en utilisant des textures projectives dépendantes du point de vue, de haute qualité et avec un faible coût en mémoire texture. Dans un premier temps, des photographies de la scène réelle sont utilisées pour créer un modèle 3D avec des outils standards. Notre méthode ordonne automatiquement la géométrie en groupes à niveaux de visibilité différents pour chaque point de vue. Ces groupes sont ensuite utilisés pour découper les images en calques, permettant aux artistes le remplissage des textures manquantes, dues à leur non-visibilité, en utilisant des outils comme le tampon de duplication (clone brush). Le résultat de ce processus est utilisé par notre nouvel algorithme de rendu en utilisant des textures projectives à calques, ayant un faible coût d'utilisation de mémoire texture, une haute qualité de rendu et dans le cas de textures projectives dépendantes du point de vue, évite la subdivision de la géométrie engendrée par la visibilité en utilisant des méthodes antérieures.*

1 Introduction

Les développements récents en modélisation à partir d'images et d'autres techniques de capture comme le scanner laser ou les méthodes stéréo, ont eu comme conséquence un intérêt croissant pour la création et l'affichage des modèles 3D dans le domaine du rendu réaliste et les environnements virtuels. La qualité visuelle de tels environnements peut être très bonne, grâce à la richesse des textures extraites à partir de la photographie numérique de haute résolution et la qualité de la géométrie 3D reconstruite.

Les applications de ces méthodes sont nombreuses. Les études de planification urbaine et les études d'impact sur l'environnement, l'archéologie et l'éducation, la formation, la conception, mais aussi la production de films ou les jeux vidéo sont juste à peine quelques cas pour lesquels ces environnements virtuels hautement réalistes peuvent être employés.

Les approches de modélisation à base d'images ont été en grande partie basées sur les travaux pionniers dans la vision par ordinateur [TK95, F$^+$97]. Des méthodes de modélisation et de rendu à base d'Images (MRBI) ont été également développées par [LH96, G$^+$96]. Ces approches représentent une alternative pour capturer et afficher de scènes réelles. L'approche de Facade de Debevec et ses collaborateurs [D$^+$96], et sa version interactive [D$^+$98], emploient un algorithme de rendu dépendant du point de vue. Les textures utilisées pour l'affichage sont le résultat du mélange de multiples vues. Dans le cas interactif, ils emploient des textures projectives, nécessitant un prétraitement de visibilité géométrique, c'est à dire la subdivision des polygones d'entrée.

Dans la pratique, plusieurs produits commerciaux de modélisation à base d'images ont été développés et sont employés au sein de véritables projets[1]. Cependant tous ces produits ont adopté la méthode standard de texturation pour l'affichage, plutôt que les textures projectives : chaque polygone a une texture associée, qui est extraite à partir des photographies d'entrée en appliquant la projection inverse. Le résultat peut être ainsi directement visualisé avec un système de rendu traditionnel. Ce choix permet également à des artistes d'intervenir à divers moments du processus, à l'aide d'outils standard d'édition d'image (tels qu'Adobe Photoshop^{TM}(http://www.adobe.com), ou GIMP (http://www.gimp.org), afin de remplir les textures manquantes ou pour enlever les objets indésirables, etc., ou à l'aide d'outils de modélisation standard pour éditer la géométrie.

Dans cet article, nous adoptons un modèle d'affichage dépendant du point de vue. Nous développons une nouvelle approche de rendu à base de textures projectives, qui ne nécessite exige pas la subdivision des polygones. Nous

1. E.g., www.realviz.com, www.canoma.com, www.photomodeler.com.

générons automatiquement des calques d'image, qui réduisent la quantité de mémoire texture utilisée, comparée aux approches d'extraction de texture. Les calques d'image peuvent être employés dans un programme d'édition d'image standard (comme PhotoShop), permettant aux artistes de contrôler la qualité finale du rendu.

Nous avons implémenté notre système, et nous montrons des résultats d'un modèle 3D utilisé dans un projet réel, qui est la construction du tramway de Nice. Un rapport technique décrivant les détails et plus de résultats de ce travail se trouvent sur http://www.inria.fr/reves/publications/data/2003/RD03/.

2 Travaux Précédents

Dans leur papier original, Debevec et ses collaborateurs [D+96] ont présenté le système Facade, qui est le premier logiciel de modélisation à base d'images. Dans cet article ils présentent les textures projectives dépendantes du point de vue (TPDV). Ce travail a été transposé dans un contexte de rendu interactif [D+98] au sein duquel les polygones partiellement visibles dans une des images d'entrée sont subdivisés. Cette méthode a l'avantage d'être complètement automatique ; par contre, il y manque le contrôle de qualité nécessaire pour la création de contenu. En particulier, pour les surfaces qui ne reçoivent pas une texture projective, l'interpolation simple de couleur est employée pour remplir les trous, produisant des erreurs visibles. La subdivision de la géométrie d'entrée est également problématique selon le contexte, à cause des imprécisions numériques. La méthode que nous présentons ici peut être vue comme une extension de cette approche, en abordant ces deux questions, le remplissage des trous et la subdivision de la géométrie.

D'autres méthodes de modélisation et rendu à base d'images ont été développées depuis les années 90. Les algorithmes de Lightfield [LH96] et de Lumigraph [G+96] sont basés sur la capture relativement dense d'images, de sorte que l'interpolation d'images pure (Lightfield) ou aidée par la géométrie (Lumigraph) suffit pour rendre une image. La taille des données nécessaire pour ces approches (gigaoctets de données d'image) les rend très difficiles à employer dans des projets réels.

Le rendu dépendant du point de vue [P+97] ou les "surface light fields" [W+00] améliorent les méthodes de base. L'"Unstructured Lumigraph" [B+01] peut être vu comme un compromis entre le Lumigraph et le TPDV. Le critère utilisé pour le mélange de textures est plus sophistiqué que dans TPDV, et le modèle géométrique utilisé peut être très approximatif. Une autre approche de l'MRBI utilise les calques d'images avec profondeur (Layered Depth Images, LDI) [S+98], qui ajoutent la profondeur à une image multicalque. Toutes ces méthodes nécessitent des algorithmes de rendus spécifiques. Dans certains cas ces méthodes sont difficiles à intégrer dans un système traditionnel. Pour les LDI, les résultats dépendent de la qualité et de la disponibilité de l'information de profondeur.

Notre algorithme pour la construction des calques est basé sur un algorithme de tri de la géométrie par visibilité développé par Krishnan et collaborateurs [K+01]. Une solution plus complète a été présentée par Snyder et Lengyel [SL98]. Les cycles de visibilité sont résolus dans l'algorithme développé par Newell et autres [N+72]. Notre travail est également lié à "Tour Into de Picture" [H+97], où une seule vue est employée en établissant des calques manuellement. Le travail de Oh et collaborateurs [O+01], est également lié à notre recherche en ce qui concerne l'utilisation de la 3D pour l'édition d'images.

3 Vue d'ensemble

Nous supposons que nous avons construit un modèle 3D à partir d'images avec des appareils photo calibrés (nous utilisons REALVIZ ImageModeler^{TM} dans nos exemples pour la calibration et la modélisation à partir d'images).

Notre but est d'utiliser des textures projectives, tout en évitant de subdiviser la géométrie d'entrée. Dans la fig. 1 la scène comporte trois objets, *A*, *B* et *C*. Nous considérons 3 vues différentes, c_1,c_2 et c_3. En utilisant la méthode de textures projectives dépendantes du point de vue [D+98], même dans une scène si simple, les objets *A* et *B* doivent être subdivisés. Pour la vue c_1, *B* sera coupé en B_1, B_2 (voir la fig. 1 deuxième ligne). En se déplaçant de c_1 vers c_2, l'objet B_2 aura l'image de c_1 comme texture projective, et puis mélangera les images de c_1 et c_2. Par contre, l'objet B_1 sera toujours montré avec la texture projective correspondante à l'image de c_2, puisqu'il est invisible depuis c_1.

Au lieu de faire ceci, nous subdivisons l'image en calques, issus du tri de la géométrie en *calques de visibilité*. Pour un point de vue donné, un calque de visibilité est un ensemble de surfaces pour lesquelles aucune paire n'est

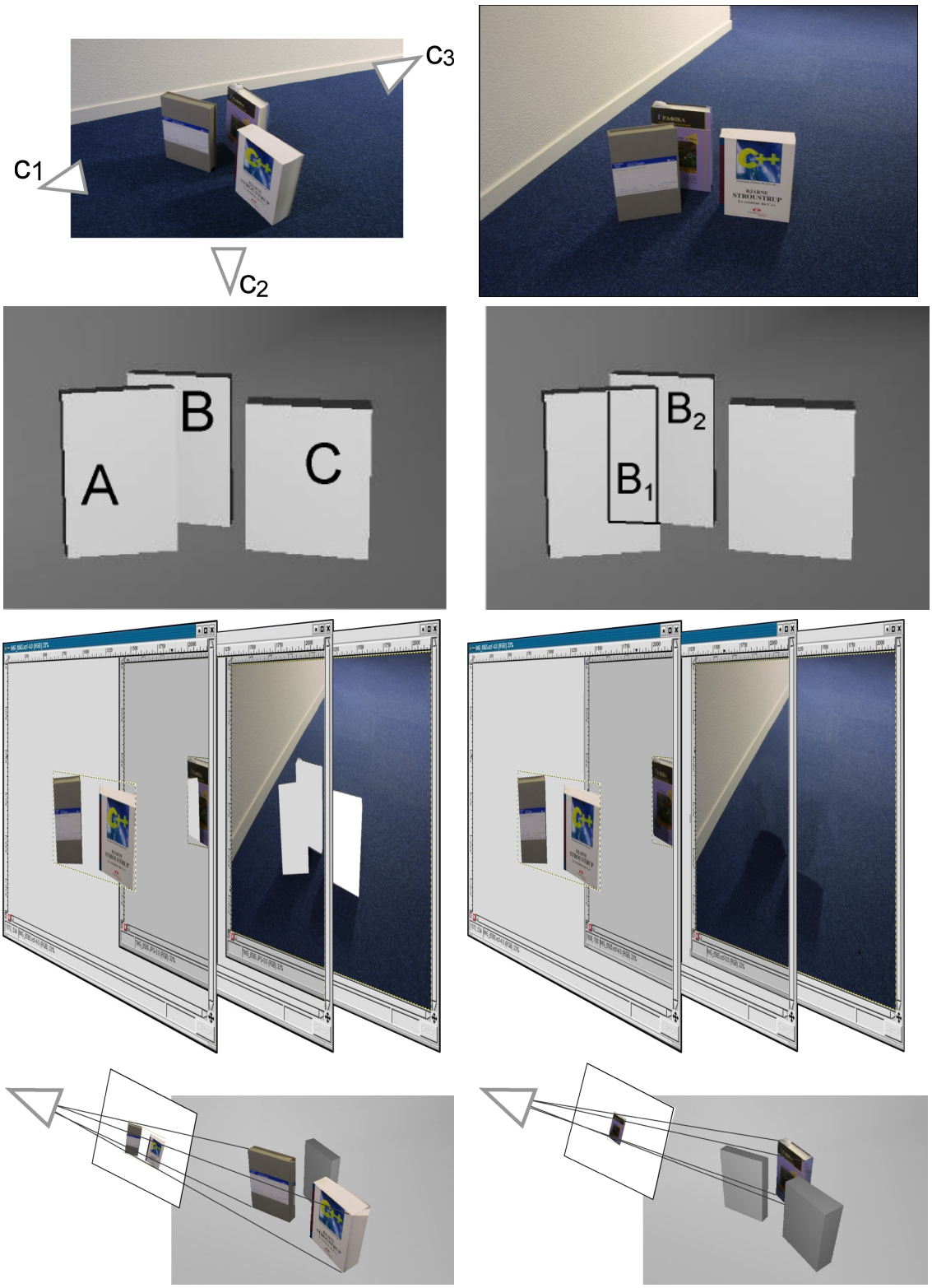

FIG. 1 – *Ligne 1 gauche : Configuration géométrique de la scène et des points de vue de d'entrée. Ligne 1 droite : L'image d'entrée correspondant à c_1. Rangée 2. : la géométrie 3D correspondante. Ligne 3 gauche : Calques d'image résultat après le tri de la géométrie. Rangée 3 droite : Calques d'image après l'édition (tampon de duplication, etc.). Rangée 4 : Visualisation des calques.*

ni partiellement, ni complètement, mutuellement cachée. Dans l'exemple de la fig. 1, pour le point de vue c_1, il y a trois calques. Le première contenant *A* et *C*, le deuxième contenant *B*, qui est partiellement caché par *A* et la

troisième contenant le fond. Nous créons alors un *calque d'image* pour chaque calque de visibilité (fig. 1 troisième rangée). Le premier calque correspond à la partie de l'image de c_1 couverte par les pixels des objets *A* et *C*, et ainsi de suite.

Nous construisons des calques géométriques de visibilité en adaptant l'algorithme de [K$^+$01], puis nous créons des calques d'image standard (Adobe Photoshop^{TM} ou GIMP), qui seront employés en tant que textures projectives.

L'artiste édite alors le calque d'image, afin de, par exemple, remplir les morceaux de texture manquants (par exemple fig. 1, 3ème ligne, droite). Les calques des images sont employés en tant que textures projectives pour les polygones du modèle 3D correspondants, évitant ainsi le besoin de subdivision de la géométrie (ceci est montré dans la dernière ligne de la fig. 1) puisque les calques et l'édition d'images ont résolu correctement la visibilité.

4 Création des calques de visibilité et d'image

La première étape est d'estimer l'ensemble de paramètres intrinsèques et extrinsèques des appareils photo correspondants aux images, et créer un modèle 3D approximatif de la scène. Nous utilisons REALVIZ ImageModeler^{TM} pour cette étape. Notre but est de créer un ensemble de calques pour chaque image d'entrée de l'ensemble, qui sera plus tard employé en tant que textures projectives pour l'affichage. Ces calques ont la propriété suivante : pour n'importe quel objet dans le premier calque, aucun autre objet de la scène ne le cache ; pour les objets du deuxième calque, une fois les objets du premier calque enlevés, la même propriété s'applique, et ainsi de suite pour les calque suivants. Une fois que tous les calques ont été créés, nous pouvons utiliser des textures projectives sans avoir besoin de subdiviser la géométrie, puisque nous avons une texture projective séparée pour chaque calque.

Comme nous l'avons mentionné précédemment, nous intégrons la création de ces calques dans un programme d'édition d'image standard, de sorte qu'un artiste puisse compléter les parties absentes d'un calque donné, en utilisant des techniques standard telles que le tampon de duplication (*clone brush*), et autres outils d'édition.

Pour créer ces calques d'image, nous trions d'abord la géométrie par ordre de visibilité, pour créer ce que nous appelons les *calques de visibilité*, qui sont des groupes d'objets ou polygones. Nous créons ensuite ce que nous appelons les *calques d'image*, qui sont des calques utilisables dans un programme d'édition d'image tel qu'Adobe Photoshop^{TM} ou GIMP.

Pour la première étape, nous adaptons un algorithme de tri par visibilité existant afin de créer les calques de visibilité. La deuxième étape est réalisée en projetant les calques de visibilité dans l'espace image pour créer des calques 2D éditables, optimisés pour réduire la quantité de texture utilisée.

4.1 Calques de visibilité

Pour calculer les calques de visibilité nous adaptons l'algorithme de tri par visibilité de [K$^+$01]. Au début, nous rendons la scène entière dans un *Item Buffer*, c'est-à-dire, nous donnons un identificateur unique pour chaque objet de la scène. Nous récupérons ensuite cette information pour créer un premier ensemble de polygones potentiellement complètement visibles, contenant ainsi tous les objets avec au moins un pixel dans l'*Item Buffer* contenant son identificateur.

L'algorithme construit itérativement un ensemble de calques de visibilité. Pour chaque itération, l'ensemble d'objets non-classés, qui au début de chaque itération est aussi l'ensemble de candidats pour le calque, est rendu dans l'*Item Buffer* et le *Stencil Buffer* avec le test de profondeur inversé (`GL_LESS`). Pour chaque pixel ayant une valeur supérieure à 1 dans *Stencil Buffer*, l'objet correspondant au même pixel dans l'*Item Buffer* est partiellement caché, puisque le test de profondeur est inversé et ce qu'on voit dans les buffers est l'arrière de la scène. L'objet est ensuite enlevé de l'ensemble de candidats pour le calque considéré. L'ensemble de candidats pour le calque est alors rendu itérativement, jusqu'à ce que le *Stencil Buffer* ne contienne que des valeurs inférieures ou égales à 1. Ces objets sont les objets complètement visibles pour le calque courant. Un dernier test est fait sur ces objets pour résoudre un défaut de l'algorithme de Krishnan [K$^+$01]. Dans le cas 2D montré dans la Fig. 2 l'algorithme va classer comme objets du premier calque les objets *A* et *C*, par contre nous voyons clairement que seul l'objet *A* est complètement visible. Le test ajouté pour résoudre ce problème consiste simplement en une comparaison entre le premier et dernier rendu de l'itération courante. Tout objet qui apparaît partiellement visible au premier rendu ne peut être considéré comme complètement visible, il est donc éliminé de l'ensemble de candidats pour le calque.

Les objets restants sont insérés dans le calque de visibilité courant. L'ensemble d'objets n'ayant pas encore été classés est utilisé comme ensemble de candidats pour le prochain calque.

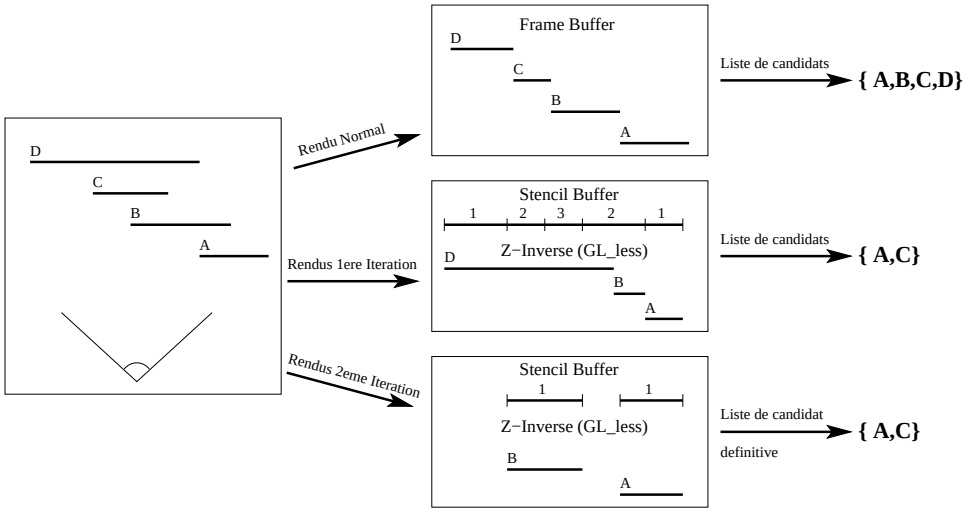

FIG. 2 – *Illustration de l'algorithme de tri de visibilité.*

Le résultat de cet algorithme est un ensemble de calques de visibilité. Dans chaque calque de visibilité, nous avons un ensemble d'objets ou polygones, qui sont ensuite utilisés pour construire les calques d'images.

Dans notre implémentation nous avons optimisé hiérarchiquement le processus grâce à la structure de scène fournie par le programme de modélisation à base d'images utilisé. En particulier, la scène fournie par ImageModeler^{TM} est organisée en objets qui contiennent un ensemble de polygones. Au début, les identificateurs sont donnés aux objets plutôt qu'aux polygones, et l'algorithme est exécuté sur des objets, éliminant rapidement de grands objets. Lorsque nous ne pouvons plus continuer le tri des objets, nous remplaçons les objets par leurs polygones constitutifs. Dans nos exemples, cette démarche est suffisante pour résoudre la visibilité dans la scène. Néanmoins, des cycles pourraient se produire, même au niveau de polygones/triangles (par exemple si deux triangles se coupent), et dans ce cas nous devrions identifier les cycles et découper la géométrie en utilisant un algorithme comme celui de Newell et collaborateurs [N^{+}72].

4.2 Création des calques d'image

Pour créer les calques d'image, nous utilisons l'ensemble de calques de visibilité. Pour chaque calque de visibilité, nous projetons chaque polygone de l'ensemble dans l'espace image, ce qui résulte en un découpage 2D pour chacun. Par contre, étant donné que les objets de chaque calque peuvent être n'importe où dans l'image et que souvent ils sont très séparés, nous sommes obligés d'optimiser le résultat en utilisant un algorithme de clustering. L'algorithme que nous utilisons est très simple. Nous prenons avantage du fait que les textures d'`OpenGL` soient forcement à résolution puissance de 2 (sauf avec des nouvelles extensions qui ne sont pas disponibles sut toutes les cartes). Pour une paire c_1 et c_2 d'objets projetés nous fusionnons c_1 avec c_2 dans le même calque, si : $A_{12} < A_1 + A_2$, où A_i est l'aire de la boîte englobante de la projection avec côtés $x'_i = \min_n(2^n) > x_i$ et $y'_i = \min_n(2^n) > y_i$ où x_i et y_i sont les côtés de la vraie boîte englobante de la projection. Ainsi, chaque calque est potentiellement séparé en plusieurs calques. Ceci a comme conséquence une augmentation du nombre de calques, mais ceci est nécessaire pour réduire l'utilisation de mémoire texture. L'ensemble final de calques est alors prêt pour l'édition.

Les calques sont envoyés vers le programme d'édition d'image (dans notre cas GIMP) qui va convertir l'ensemble de calques en une image composée qui est visuellement identique à l'image originale. L'artiste peut alors éditer en utilisant tous les outils d'édition disponibles pour remplir toutes les textures manquantes, effacer les objets non désirés, etc.

5 Affichage

Après avoir créé les calques de visibilité et avoir manuellement édité les calques d'image, nous employons des textures projectives pour l'affichage. Pour tous les points de vue qui correspondent aux points de vue d'entrée tout ce que nous devons faire est rendre la géométrie et pour chaque objet assigner comme texture le calque image correspondant. Pour permettre l'affichage d'autres points de vue, nous mélangeons les textures avec les facteurs de *blending* appropriés [D$^+$98].

5.1 Structure de données

Nous avons conçu l'algorithme afin de nous adapter à un système de rendu par scène-graphe. Nous utilisons OpenGL Performer^{TM} [htt] dans notre implementation.

Pour utiliser les textures projectives par calques, nous avons modifié le graphe pour contenir un noeud de texture avec les informations du calque. En plus de l'information d'un noeud de texture standard, ce noeud contient les coordonnées de la sous-image correspondant au calque (x_i,y_i,w_i,h_i) qui sera employé pour calculer la matrice de projection de texture.

Pour rendre les calques, nous devons appliquer une transformation perspective asymétrique. Ceci peut être directement mis en application en utilisant la commande `glFrustum` d'OpenGL avec les paramètres suivants :

$$x'_{min} = x_{min} + \frac{x_i(x_{max}-x_{min})}{w} \quad x'_{max} = x_{min} + \frac{(x_i+w_i)(x_{max}-x_{min})}{w}$$

$$y'_{min} = y_{min} + \frac{y_i(y_{max}-y_{min})}{h} \quad y'_{max} = y_{min} + \frac{(y_i+h_i)(y_{max}-y_{min})}{h}$$

où x_{min}, x_{max}, y_{min} et y_{max} sont les valeurs du frustum pour la projection de l'image originale, w et h sont la largeur et la taille de l'image originale et x_i, y_i, w_i et h_i correspondent à la position et à la taille du calque dans l'image originale.

5.2 Rendu avec multiples textures

Pour rendre d'autres points de vue que ceux qui correspondent aux point de vue d'origine, nous utilisons un mélange de plusieurs textures, avec une approche similaire à celle de Debevec [D$^+$98]. Pour atteindre ce but nous devons modifier notre structure de graphe. Nous créons un noeud dépendant du point de vue qui contient une liste de noeuds de texture projectifs et chacun d'entre eux lié au noeud géométrie unique.

Quand la procédure d'affichage parcourt ce noeud, des facteurs de *blending* sont choisis pour chaque texture correspondante [D$^+$98]. Ensuite, chaque noeud de texture projectif met à jour la matrice de projection pour sa texture et le facteur de *blending* que le noeud père a calculé, puis le noeud géométrie associé s'affiche autant de fois que nécessaire, tant que les facteurs de *blending* sont supérieurs à 0.

6 Résultats

Nous montrons comme exemple un modèle reconstruit de la place Massena à Nice. Ce modèle a été créé dans le contexte d'une simulation de réalité virtuelle/augmentée du projet de tramway à Nice.

Nous utilisons deux photos d'entrée à d'une résolution de 2160x1440. Dans la fig. 3, nous montrons des vues synthétiques intermédiaires en utilisant notre méthode. L'algorithme crée un total d'entre 98 et 160 calques (optimisées) pour chaque image, dont beaucoup sont très petites et n'ont pas besoin d'édition. La mémoire texture nécessaire est de 10-12 méga-octets selon l'image. L'algorithme pour créer les calques prend moins de 3 minutes sur un Pentium IV cadencé à 1,2 Gigahertz. Le processus entier d'édition des calques a pris environ deux jours pour notre artiste. Le même processus en utilisant l'approche d'extraction de textures par objet, avec la même scène et les mêmes vues a pris trois semaines.

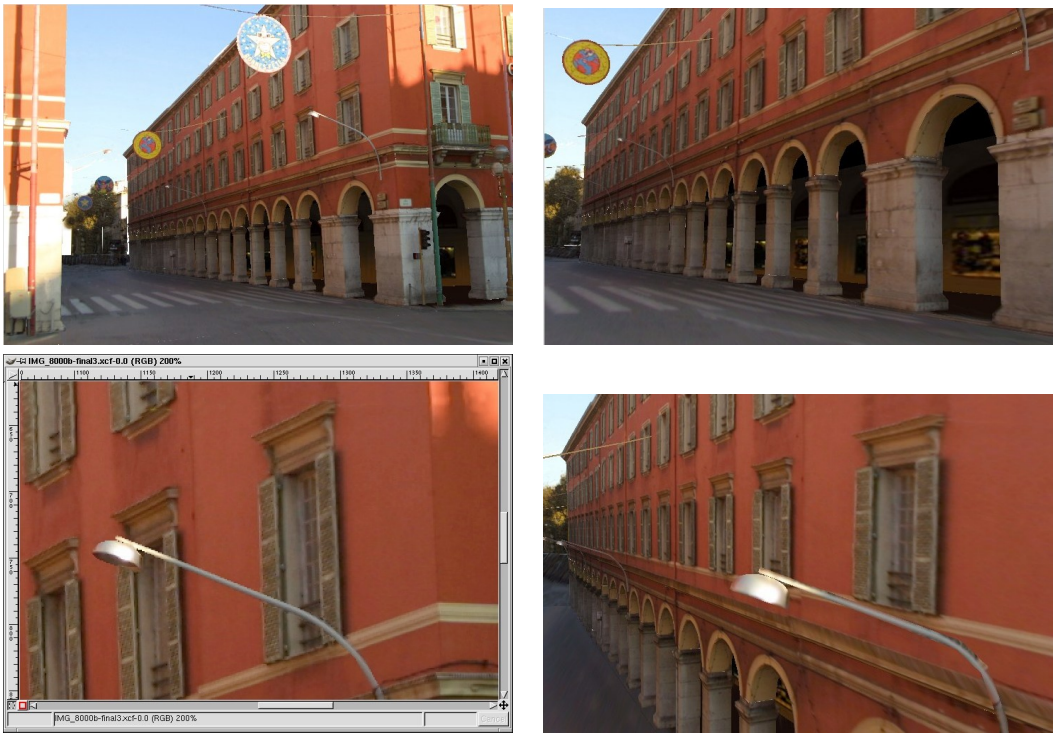

FIG. 3 – *Première ligne : deux nouvelles vues synthétisées, en utilisant notre nouvel affichage avec des textures projectives à calques. Deuxième ligne (droite) : Vue rapprochée d'une image d'entrée ; (gauche) Vue rapprochée dans notre système d'un point de vue différent. La qualité de la texture dans la vue synthétique est comparable à l'image d'entrée.*

7 Conclusions

Comparé à la méthode interactive des textures projectives dépendantes du point de vue [D+98], nous évitons la subdivision de la géométrie pour la visibilité, et nous permettons l'interaction de l'artiste pour l'édition des textures manquantes, plutôt qu'utiliser l'interpolation. Les avantages de notre approche sont que nous n'augmentons pas le nombre de polygones de la scène, et que l'approche s'adapte bien au habitudes de travail des artistes, où le contrôle de la qualité d'image est primordial. L'inconvénient est que le remplissage des textures manquantes n'est plus automatique.

Comparé aux méthodes d'extraction de textures classique, notre approche réduit de manière significative l'utilisation de mémoire texture avec des résultats d'une meilleure qualité visuelle, car la projection inverse génère souvent de grandes textures et a pour conséquence une perte de qualité par le re-échantillonnage. L'utilisation de notre système indique que les utilisateurs sont moins sensibles aux erreurs de parallaxe grâce à la qualité élevée des textures projectives.

Pour nos travaux futurs, nous automatiserons beaucoup d'aspects de la capture et de l'édition des textures et nous intégrerons cette méthode dans notre configuration de réalité virtuelle.

8 Remerciements

Cette recherche a été partiellement effectuée dans le cadre du projet de l'UE IST CREATE, IST-2001-34231, http://www.cs.ucl.ac.uk/create. ImageModeler a été fourni par REALVIZ dans le contexte de CREATE. Merci à A. Olivier-Mangon pour ses idées et pour son aide avec ImageModeler, à F. Durand pour ses commentaires et à M-C. Frasson pour sa relecture et correction de la version française.

Références

[B+01] Chris Buehler et al. Unstructured lumigraph rendering. In *SIGGRAPH 2001, Computer Graphics Proc.*, Annual Conference Series, pages 425–432, 2001.

[D+96] P.E. Debevec et al. Modeling and rendering architecture from photographs: A hybrid geometry- and image-based approach. In *Proc. SIGGRAPH 96*, pages 11–20, August 1996.

[D+98] Paul Debevec et al. Efficient view-dependent image-based rendering with projective texture-mapping. In *Rendering Techniques '98*, 9th EG workshop on Rendering, Vienna, Austria, June 1998. Springer Verlag.

[F+97] Olivier Faugeras et al. 3-d reconstruction of urban scenes from image sequences. *CVGIP: Image Understanding*, 1997.

[G+96] Steven J. Gortler et al. The lumigraph. In *SIGGRAPH 96 Conference Proc.*, Annual Conference Series, pages 43–54, August 1996.

[H+97] Youichi Horry et al. Tour into the picture. In *Computer Graphics Proceedings, SIGGRAPH'97*, Annual Conference Series, pages 225–232, Los Angeles, CA, August 1997. ACM.

[htt] Performer SGI Library http://www.sgi.com/software/performer/.

[K+01] S. Krishnan et al. A Hardware-Assisted Visibility-Ordering algorithm with applications to volume rendering. In *Data Visualization 2001*, pages 233–242, 2001.

[LH96] Marc Levoy and Pat Hanrahan. Light field rendering. In *SIGGRAPH 96 Conference Proc.*, Annual Conference Series, pages 31–42, August 1996.

[N+72] M. E. Newell et al. A solution to the hidden surface problem. In *Proc. of the ACM Nat. Conf.*, pages 443–450, 1972.

[O+01] Byong Mok Oh et al. Image-based modeling and photo editing. In *SIGGRAPH 2001, Computer Graphics Proc.*, Annual Conference Series, 2001.

[P+97] Kari Pulli et al. View-based rendering: Visualizing real objects from scanned range and color data. In *Rendering Techniques '97 (Proc. of the 8th EG Workshop on Rendering) held in St. Etienne, France*, pages 23–34, 1997.

[S+98] Jonathan W. Shade et al. Layered depth images. In *SIGGRAPH 98 Conference Proc.*, volume 32 of *Annual Conference Series*, pages 231–242, 1998.

[SL98] John Snyder and Jed Lengyel. Visibility sorting and compositing without splitting for image layer decompositions. In *SIGGRAPH 98 Conference Proc.*, 1998.

[TK95] Camillo J. Taylor and David J. Kriegman. Structure and motion from line segments in multiple images. *IEEE Trans. on Pat. Analysis and Mach. Intelligence*, 17(11):1021–1032, 1995.

[W+00] Daniel N. Wood et al. Surface light fields for 3D photography. In *SIGGRAPH 2000, Annual Conference Proc.*, pages 287–296, 2000.