Space Carving multi-view video plus depth sequences for representation and transmission of 3DTV and FTV contents

Youssef Alj

Thèse

THESE INSA Rennes
sous le sceau de l'Université européenne de Bretagne
pour obtenir le titre de
DOCTEUR DE L'INSA DE RENNES
Spécialité : Informatique

présentée par
Youssef ALJ
ECOLE DOCTORALE : *MATISSE*
LABORATOIRE : *IETR*

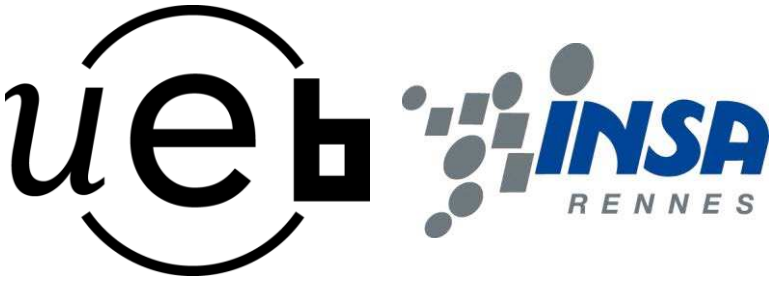
Space Carving Multi-view Video plus Depth Sequences for Representation and Transmission of 3DTV and FTV Contents

Thèse soutenue le 16.05.2013
devant le jury composé de :

Mohamed Daoudi
Professeur des universités - Telecom Lille 1 / Président
Marek Domanski
Prof. des universités - Poznan University of Technology / Rapporteur
Laurent Lucas
Prof.des universités - Université de Reims Champagne-Ardenne / Rapporteur
Edmond Boyer
Directeur de recherche - INRIA Rhône-Alpes / Examinateur
Guillaume Boisson
Ingénieur de recherche - Technicolor Rennes / Co-encadrant
Muriel Pressigout
Maître de conférence - INSA Rennes / Co-encadrant
Luce Morin
Professeur des universités - INSA Rennes / Directrice de thèse

Space Carving Multi-view Video plus Depth Sequences for Representation and Transmission of 3DTV and FTV Contents

Youssef ALJ



En partenariat avec

Contents

Remerciements

Je souhaite remercier en premier lieu tous les membres du jury pour avoir accepté de juger mes travaux.

Je souhaite ensuite remercier mes encadrants pour tout le temps qu'ils ont consacré pour me guider et me transmettre leur savoir-faire que ce soit du côté Technicolor ou IETR.

Du côté IETR, je souhaite exprimer ma profonde gratitude à ma directrice de thèse Luce Morin. Merci Luce de m'avoir soutenu dans de nombreuses situations. Merci de m'avoir supporté tout au long de ces trois années (+ ϵ). Merci pour la qualité de ton encadrement, pour ton professionnalisme et pour ta bonne humeur. Merci tout simplement pour ton humanité. Un grand merci également à Muriel Pressigout pour m'avoir co-encadré et conseillé durant cette thèse.

Du côté Technicolor, je tiens à exprimer ma reconnaissance et tous mes profonds remerciements à mon encadrant Guillaume Boisson. Merci Guillaume pour ton soutien, ton implication et ton professionnalisme exemplaires. Mes remerciements vont également à Philippe Bordes. Merci Philippe pour ta disponibilité et également pour toutes tes idées et remarques pendant les réunions et durant la lecture du manuscrit. Merci également à Paul Kerbiriou pour ton implication et les multiples et longues conversations qu'on a eues ensemble, qu'elles soient scientifiques ou pas. Finalement, je souhaite également remercier Philippe Guillotel pour m'avoir accueilli dans son labo.

Je n'oublierai surtout pas de remercier Pedro Franco De Carvalho pour ses travaux, pendant son stage, et ses contributions à la richesse des résultats de la thèse.

Je tiens également à remercier tous les stagiaires, thésards, post-docs et tous les autres collègues, qu'ils soient à Technicolor ou à l'IETR, qui ont contribué significativement à la bonne ambiance du travail au quotidien.

Je souhaite également remercier mes amis bordelais (on restera toujours bordelais malgré la distance!). Sans eux, la thèse aurait pu avoir un goût complètement différent. Un grand merci à Safouane, Hélène, David, Thomas, Georges, Emilie, Amine et Nawfal.

Enfin, rien de tout cela n'aurait été possible sans mes parents. Quoique je dise ou quoique je fasse, je ne saurai vous remercier. Je vous dédie cette thèse en reconnaissance de tout le soutien inconditionnel et intemporel que vous avez su m'apporter pendant toutes ces longues années. Merci!

Résumé en Français

Chapter 1

Résumé en Français

Contents

1.1 Introduction

L'Imagerie Multi-Vue (IMV) a suscité un vif intérêt durant ces dernières années. Grâce au développement récent des écrans 3D, l'IMV fournit une sensation réaliste de profondeur à l'utilisateur et une navigation virtuelle autour de la scène observée, ouvrant par conséquent un large éventail de sujets de recherche et d'applications telles que la télévision 3D (TV3D) ou la FTV (*Free-viewpoint TV*). Cependant, de nombreux défis techniques existent encore. Ces défis peuvent être liés à l'acquisition de la scène et à sa représentation d'une part ou à la transmission des données d'autre part. Dans le contexte de la représentation de scènes naturelles, de nombreux efforts ont été fournis afin de surmonter ces difficultés. Les méthodes proposées dans la littérature peuvent être classées en trois catégories : les représentations basées image, les représentations basées géométrie ou les représentations intermédiaires.

Les représentations basées image regroupent l'utilisation de MVV (Multi-view Video), 2D+Z [Fehn et al., 2002] ou MVD (*Multi-view Video plus Depth*) [Merkle et al., 2007], où chaque vue est constituée de l'image acquise et d'une carte de profondeur estimée. Les représentations basées géométrie s'appuient sur l'utilisation d'un modèle géométrique à base de surface maillée [Balter et al., 2006], de modèle volumétrique [Mueller et al., 2004a] ou de nuage de points [Waschbüsch et al., 2005]. Les représentations intermédiaires incluent l'utilisation des LDI (*Layered Depth Images*) [He et al., 1998], des représentations dites *billboards* [Waschbüsch et al., 2007b] ou de soupe de polygones [Colleu et al., 2010]. L'approche adoptée dans cette thèse consiste en une méthode hybride s'appuyant sur l'utilisation des séquences MVD, afin de conserver le photo-réalisme de la scène observée, combinée avec un modèle géométrique, à base de maillage triangulaire, renforçant ainsi la compacité de la représentation. Nous supposons que les cartes de profondeur fournies sont fiables et que les caméras utilisées durant l'acquisition sont calibrées, les paramètres caméras sont donc connus, mais les images correspondantes ne sont pas nécessairement rectifiées *i.e.* les lignes épipolaires ne sont pas forcément parallèles avec les lignes de l'image. Nous considérerons ainsi le cas général où les caméras peuvent être parallèles ou convergentes.

Les contributions de cette thèse sont les suivantes : D'abord, un schéma volumétrique dédié à la fusion des cartes de profondeur en une surface maillée est proposé, c'est l'objet de la section 1.2. Ensuite, un nouvel algorithme de plaquage texture multi-vues est présenté dans la section 1.3. Nous aborderons à l'issue ce ces deux étapes de modélisation, la transmission proprement dite et comparerons les performances de notre schéma de modélisation avec l'état de l'art dans la section 1.4.

1.2 Space carving pour la fusion de cartes de profondeur

Dans cette section, notre schéma volumétrique dédié à la fusion des cartes de profondeur est présenté. Le problème d'estimation de la disparité inter-vue n'est pas abordé dans cette thèse et les cartes de profondeurs fournies sont supposées être fiables. Le but est de fusionner ces cartes de profondeur en une seule surface cohérente avec les vues originales. Une vue d'ensemble de notre schéma de

fusion est présentée dans la figure 1.1.

Premièrement, un maillage triangulaire en haute résolution est extrait à partir de chaque carte de profondeur.

Ensuite, le volume englobant tous ces maillages est défini puis discrétisé en voxels.

Un voxel peut avoir deux états ou étiquettes : "opaque" ou transparent. Pour chaque caméra, les voxels se situant dans son cône de vue sont déterminés et étiquetés comme "opaque". A l'étape suivante, l'Enveloppe Volumétrique du Maillage (EVM) est déterminée pour chacune des vues. Étant donné un maillage correspondant à un point de vue, l'EVM définit l'ensemble des voxels situés sur la surface maillée. Notre formulation de l'algorithme de *Space Carving* repose sur l'EVM calculée. En effet, pour chaque vue, des rayons sont tracés à partir du centre de la caméra à chaque voxel constituant l'EVM correspondant à cette vue. Les voxels se situant sur chaque rayon sont déclarés transparents.

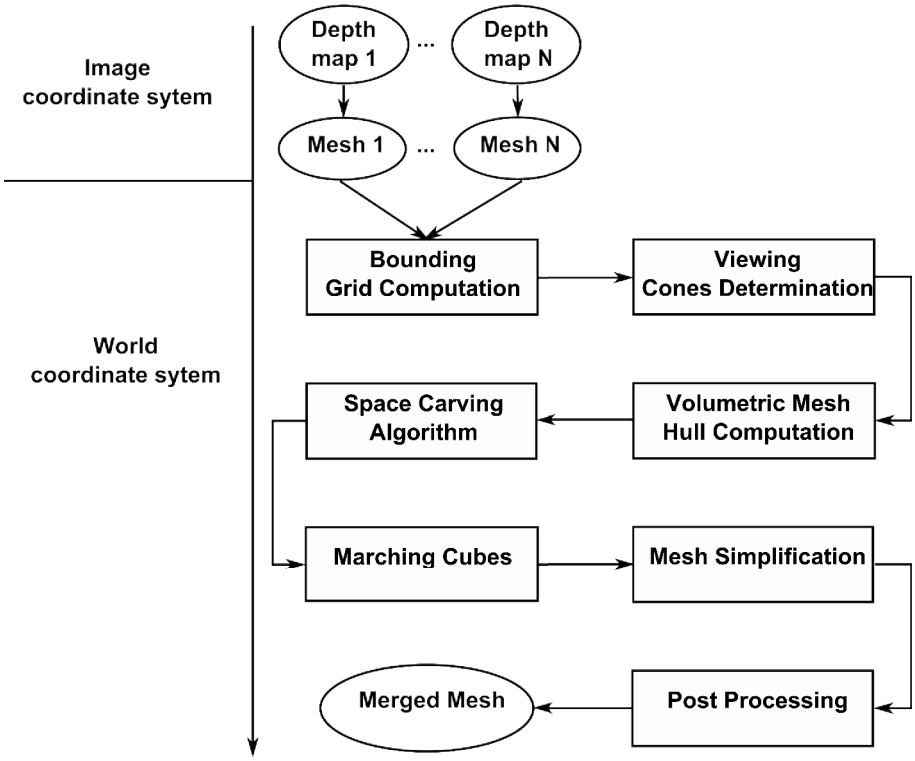


Figure 1.1: Le Schéma proposé de fusion de cartes de profondeur en un maillage triangulaire.

1.2.1 Construction des maillages à partir des cartes de profondeur

La première étape de l'algorithme est la génération d'un maillage pour chaque carte de profondeur donnée. Pour chaque vue, un maillage haute résolution est construit dans le repère image. Chaque sommet du maillage représente un pixel dans la carte de profondeur. Les maillages ainsi construits sont alignés en exprimant chaque sommet du maillage dans le repère monde en utilisant l'équation suivante :

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \equiv \begin{pmatrix} & R & & T \\ & & & \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} f_x & S_{uv} & c_u & 0 \\ 0 & f_y & c_v & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} u \\ v \\ 1 \\ 1/z_{cam} \end{pmatrix} \tag{1.1}$$

Où X, Y, et Z représentent les coordonnées d'un point 3D dans le repère monde. u et v sont les coordonnées de la projection du point 3D dans le repère image, et z_{cam} est la profondeur du pixel (u,v) dans le repère caméra. R et T définissent respectivement la matrice de rotation et de translation du repère caméra par rapport au repère monde. f_x, f_y, S_{uv}, c_u et c_v sont les paramètres intrinsèques de la caméra.

1.2.2 Calcul de la grille englobante

Les maillages ainsi construits sont les entrées de notre représentation volumétrique. La structure de données utilisée dans cette représentation est une grille englobant les maillages d'entrés. Cette grille est construite par subdivision régulière de la boîte englobante de l'ensemble des maillages en éléments parallélépipédiques appelés voxels. Un étiquetage binaire est utilisé pour attribuer à chaque voxel l'étiquette "opaque" ou transparent. L'étiquette de chaque voxel est initialisée à transparent. Les étiquettes des voxels sont modifiées tout au long de l'algorithme selon leur pertinence pour représenter la scène.

1.2.3 Détermination des cônes de vue

Dans cette étape le but est de trouver l'ensemble des voxels se situant dans le cône de vue de chaque caméra. Les voxels en dehors de l'union des cônes de vue sont étiquetés comme transparent. Le cône de vue de chaque caméra est modélisé par un frustum défini par quatre plans délimitant l'ensemble des sommets visibles selon chaque axe (voir Figure 9.5). Le but est donc de déterminer pour chaque caméra l'ensemble des voxels se situant dans le frustum correspondant. Les équations des plans du frustum sont donc calculés pour chaque caméra. Ces équations nous permettent de situer chaque voxel par rapport aux plans calculés et donc de définir les voxels dans chaque frustum.

1.2.4 Construction de l'Enveloppe Volumétrique du Maillage EVM

Dans cette section, le calcul de l'Enveloppe Volumétrique du Maillage (EVM) est présenté. Cette EVM sera utilisée durant le processus de *Space Carving*. Pour chaque maillage construit, l'ensemble des voxels intersectant la surface du maillage définit l'EVM associée à ce maillage. A la fin de la construction de chaque EVM, chaque voxel aura une information additionnelle définissant à quelle EVM ce voxel appartient. Pour chaque maillage, les triangles sont parcourus et analysés selon la grille volumétrique en utilisant un algorithme de *Scan Line 3D*, cette analyse permet de définir les voxels se situant sur chaque triangle. L'ensemble des voxels analysés seront marqués comme appartenant à l'EVM courante en mettant à jour l'information additionnelle associée au voxel. L'algorithme d'analyse des triangles d'un maillage est présenté dans l'algorithme 1.

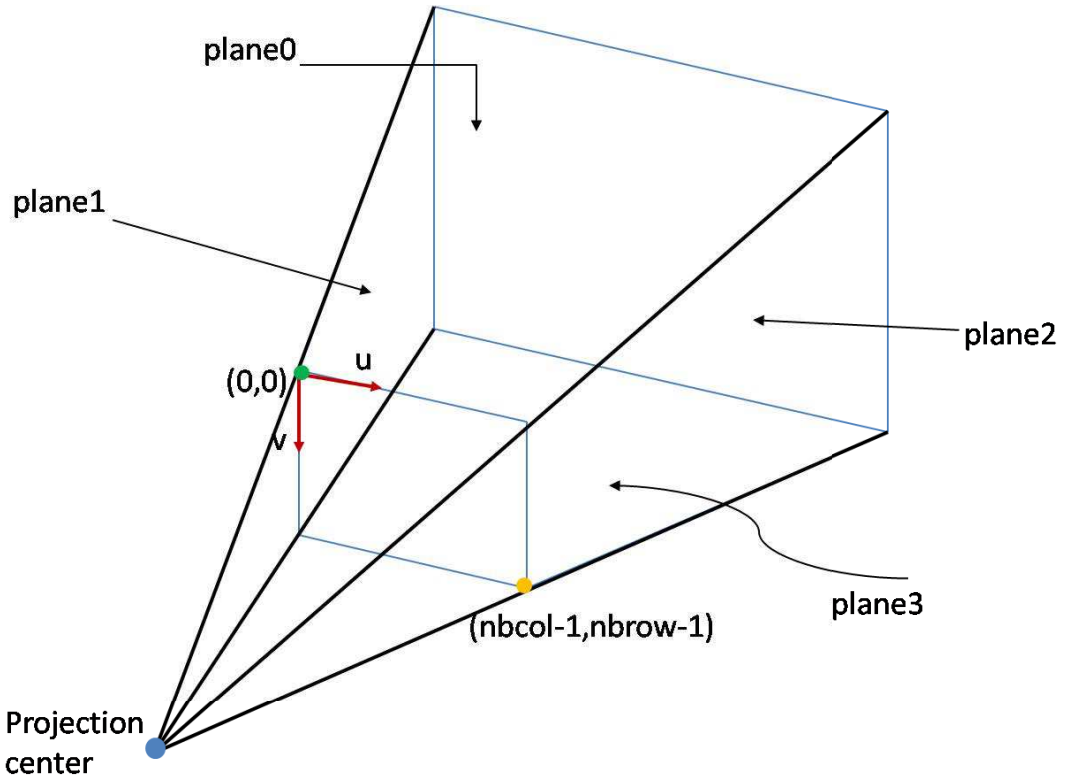


Figure 1.2: Calcul des plans du *Frustum* de projection.

Algorithm 1: Détermination de l'Enveloppe Volumétrique du Maillage (EVM).

```
  //  Parcours des triangles
1 pour chaque triangle T faire
2     pour chaque axe X, Y et Z faire
3         Calculer les limites entières b_min et b_max de T selon chaque axe.
4         pour m ← b_min to b_max faire
5             Calculer l'intersection du triangle T avec le plan à la coordonnée
              m.
6             Trouver les voxels au long de cette intersection en utilisant la
              ligne de Bresenham.
7             Ajouter ces voxels à l'EVM.
```

1.2.5 Algorithme de *Space Carving*

Jusqu'à cette étape de l'algorithme, un voxel "opaque" est un voxel qui se situe dans au moins un cône de vue et un voxel de l'EVM est un voxel qui se situe sur un des maillages. Dans cette étape, le *Space Carving* est effectué en mettant à jour les étiquettes des voxels de "opaque" à "transparent" pour tous les voxels situés devant l'EVM. Plus explicitement encore, les différents EVMs sont considérés successivement et un critère de cohérence géométrique est utilisé afin de mettre à jour les étiquettes des voxels. Ce critère est basé sur la position relative du voxel par rapport au maillage considéré et le point de vue correspondant à ce

maillage. Un voxel est dit géométriquement cohérent avec une surface du maillage s'il se situe derrière cette surface par rapport au point de vue correspondant. Les voxels détectés comme géométriquement incohérents sont étiquetés comme "transparent".

Le processus de *Space Carving* met à jour le volume sculpté pour chaque caméra. Ce volume est initialisé à l'union des voxels situés dans les cônes de vues calculés dans la section 1.2.3 (voir Figure 1.3). Le même volume est ensuite raffiné itérativement avec l'information géométrique provenant de chacune des vues, voir Figures 1.3 et 1.4), en sculptant les zones occultantes relatives à chaque vue. Pour chaque vue, les rayons sont tracés du centre de la caméra aux voxels de l'EVM correspondant à cette caméra. Pour chaque rayon, les voxels se situant sur le rayon courant sont scannés en utilisant l'algorithme de Bresenham 3D de traçage de lignes. Comme ils sont géométriquement incohérents, ces voxels seront creusés (i.e. étiquetés comme "transparent"). A la fin de cette étape de *Space Carving*, l'ensemble des voxels "opaque" définit le modèle volumétrique de nos données MVD.

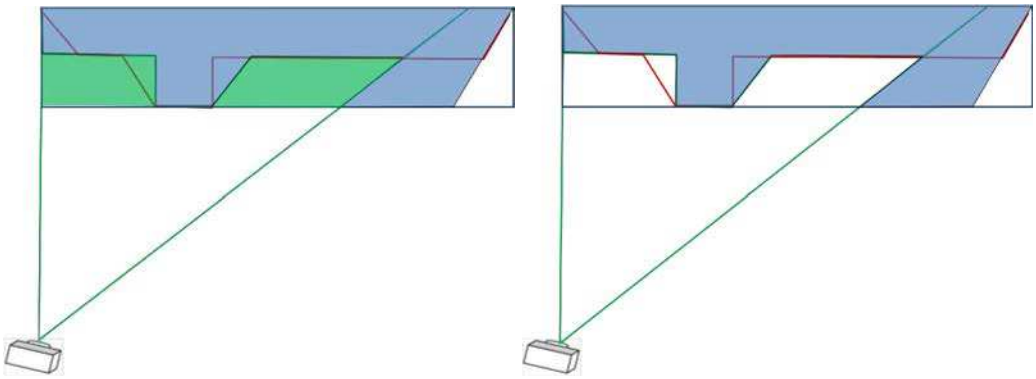


Figure 1.3: Space carving vis-à-vis de la caméra gauche. Les rayons sont tracés du centre de la caméra à l'EVM correspondante. Les zones vertes sont carvées (schéma à gauche). Le résultat du Space Carving vis-à-vis de la caméra gauche est montré dans le schéma à droite.

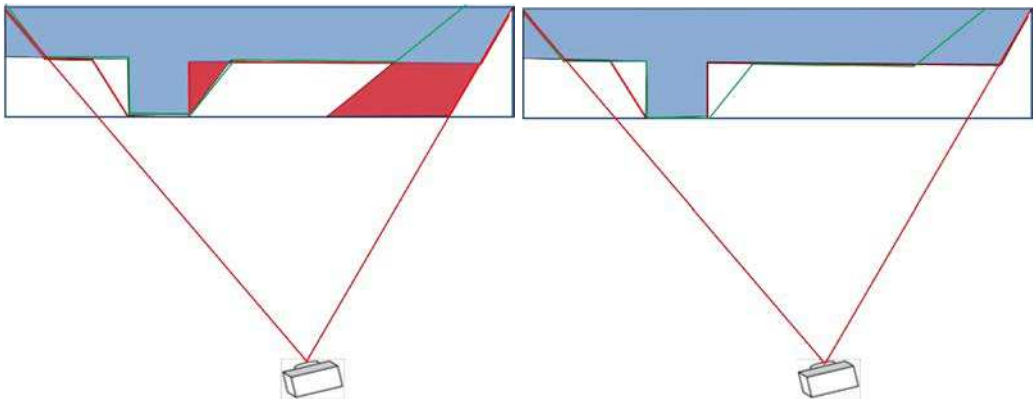


Figure 1.4: Space carving pour la caméra droite. Les zones rouges sont carvées. Le résultat est montré dans le schéma de droite.

Algorithm 2: L'algorithme de *Space Carving*.

```
// Parcours des vues
pour chaque caméra faire
    Extraire la position de la caméra.
    Extraire l'EVM correspondante.
    pour chaque voxel dans l'EVM faire
        Trouver la ligne de Bresenham 3D entre la position de la caméra et le
        centre du voxel.
        Mettre à jour les labels des voxels se situant sur cette ligne à
        "transparent".
```

1.2.6 Marching Cubes

Finalement, la surface fusionnée est extraite à partir du volume binairement étiqueté en utilisant l'algorithme des *Marching Cubes*. L'algorithme de *Marching Cubes* a été introduit par Lorenson et Cline [Lorensen and Cline, 1987]. Cet algorithme prend comme entrée une structure de données de type volume régulièrement subdivisé en voxels. A chaque voxel de ce volume est attribué une valeur. Durant le traitement, chaque sommet de voxel ayant une valeur supérieure ou inférieure à une valeur prédéfinie est marqué. Tous les autres sommets restent non-marqués. Par conséquent, le résultat de l'algorithme de *Marching Cubes* est le maillage triangulaire connectant tous les sommets qui ont été marqués.

1.2.7 Post-traitement

Le but dans cette étape est d'augmenter la cohérence du maillage fusionné avec les cartes de profondeur en entrée. Pour ce faire, chaque sommet du maillage fusionné v_{merged} est ramené au plus proche point v_{input} des données des cartes de profondeur indépendamment de la vue à laquelle ce plus proche point appartient. Ce transfert est effectué si :

$$\|v_{\text{input}} - v_{\text{merged}}\|_2 \leq R$$

Si v_x, v_y et v_z représentent les dimensions du voxel sur les axes oX, oY and oZ respectivement. R peut être défini comme suit :

$$R = \frac{\max(v_x, v_y, v_z)}{2} \tag{1.2}$$

La figure 1.5 illustre ce transfert.

1.3 Métrique de Photocohérence pour le Plaquage de Textures Multi-vues

Maintenant qu'on a déterminé le maillage fusionné. Dans cette section, nous proposons un nouvel algorithme de texturation du maillage fusionné. Le schéma global de notre méthode de texturation est présenté dans la figure 8.2. D'abord,

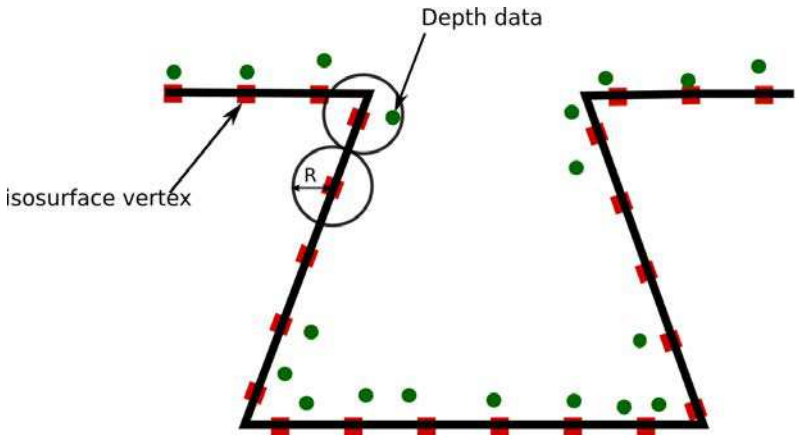


Figure 1.5: Post-traitement du maillage fusionné (rouge). Un sommet du maillage fusionné est ramené à la position du sommet le plus de l'ensemble des maillages d'entrée (vert) si la distance entre les deux points est inférieur à R.

un algorithme de visibilité basé sur le principe du Z-buffer est proposé dans la section 8.2. Ensuite, on parcourt les images de texture disponibles, et chaque texture est plaquée sur les triangles visibles du maillage fusionné. L'erreur de projection pour chaque triangle est calculée dans la section 1.3.2 vis-à-vis de chaque vue et est sauvegardée dans des images de distorsion. La meilleure texture est calculée dans la section 1.3.3 comme étant le minimum d'une énergie que l'on appelle métrique de photo-cohérence. Finalement, les vues originales sont synthétisées par rendu du maillage texture, chaque triangle étant texturé avec la meilleure texture selon cette métrique de photo-cohérence.

La figure 1.6 représente le plaquage de texture basé sur la métrique de photo-cohérence proposé. Le triangle t est visible dans les vues 1, 2 et 3. En parcourant les textures $I_i \in \mathcal{I}$, le triangle t est texturé avec I_i (flèche rouge pour l'opération de plaquage de texture) puis projeté (flèche verte pour l'opération de projection) sur les caméras 1, 2 et 3, *i.e.* l'ensemble des vues où le triangle t est visible. L'erreur de texturer le triangle t pour chaque texture est donc calculée. On appelle cette erreur la métrique de photo-cohérence du triangle t. Finalement, la meilleure texture est calculée comme celle fournissant le minimum de toutes les métriques de photo-cohérence calculées par rapport au triangle t.

1.3.1 Détermination de la visibilité

Etant donné le maillage initial $\mathcal{M}$, la visibilité partielle et totale sont d'abord déterminées vis-à-vis de chaque vue. A l'issue de cette étape, chaque triangle aura une des trois étiquettes suivantes : "totalement visible", "partiellement visible" ou "caché".

1.3.1.1 Visibilité totale

Dans cette étape, les étiquettes "totalement visible" et "caché" seront attribuées aux triangles du maillage. Les triangles sont initialisées à "totalement visible". La visibilité de chaque triangle est ensuite déterminée en calculant la visibilité

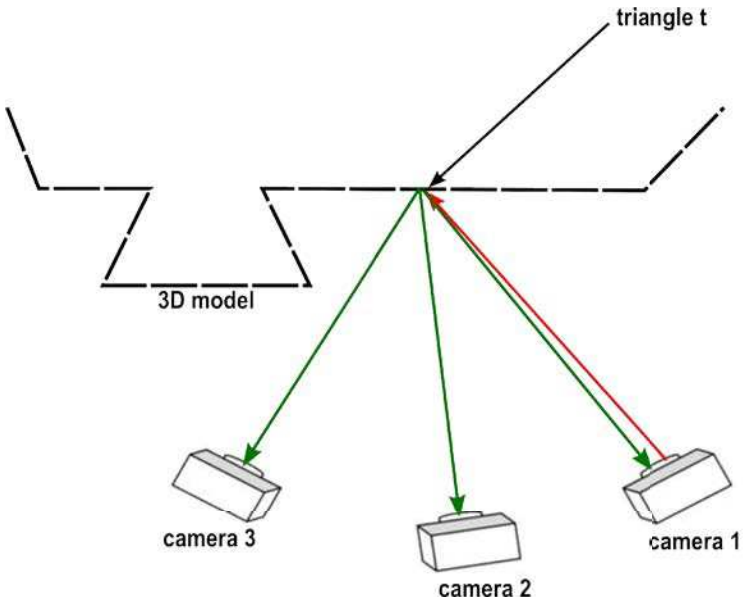


Figure 1.6: Détermination de la meilleure texture en utilisant la métrique de photo-cohérence. Flèche rouge désigne l'opération de plaquage de texture et la flèche verte désigne l'opération de projection.

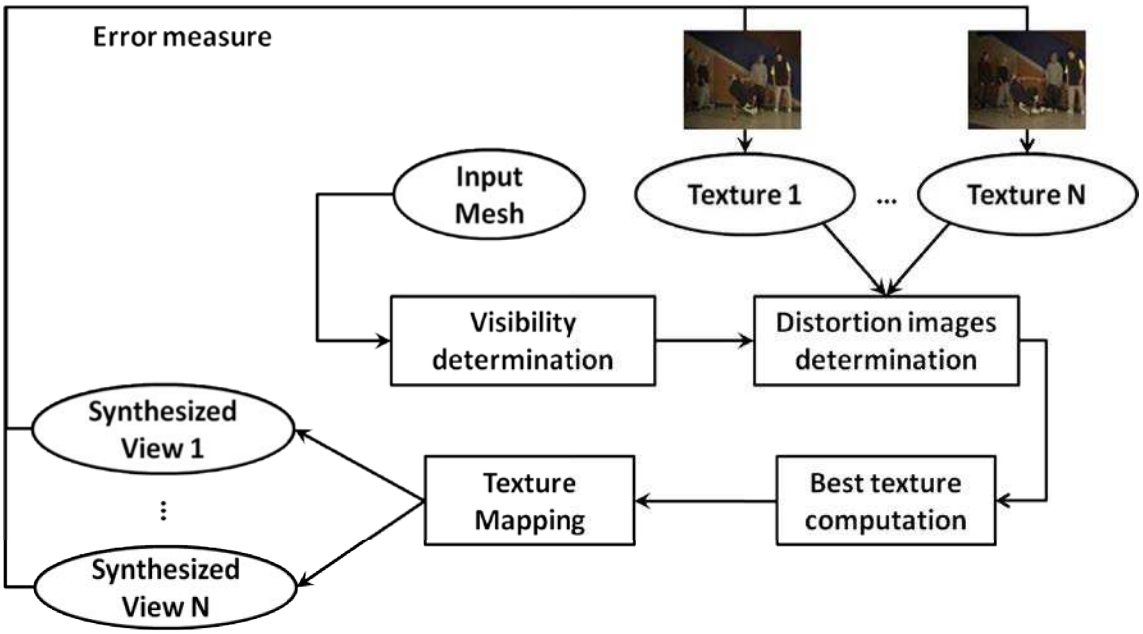


Figure 1.7: Le schéma global de la méthode de texturation proposée.

des sommets du triangle courant. Un triangle est dit caché si au moins un de ses sommets est caché. La visibilité pour chacune des vues est déterminée en utilisant le Z-buffer de OpenGL. On note Z^j_{buffer} le Z-buffer du maillage projeté sur la vue V_j.

Durant la première passe, le maillage est projeté sur la vue courante, et le z-buffer est calculé. La deuxième passe est dédiée au calcul de la visibilité par sommet en utilisant le z-buffer calculé.

Chaque sommet du maillage est alors projeté sur la vue courante, et la com-

posante de profondeur notée $z_{\text{projected}}$ de ce sommet projeté est comparée avec la valeur du pixel correspondant dans le z-buffer. Si le sommet projeté est derrière le pixel du z-buffer alors ce sommet est étiqueté comme caché, et par conséquent les triangles qui lui sont incident sont marqués comme "caché" également. Le pseudo-code permettant de déterminer la visiiblité totale est fourni dans algorithme 3.

Algorithm 3: Détermination de la visibilité globale.

```
// Parcours des vues
pour each view V_j faire
    Initialiser tous les triangles à visible.
    Projeter le maillage M sur la vue V_j.
    Sauvegarder le Z^j_buffer.
    // Parcours des sommets du maillage
    pour chaque sommet v faire
        Déterminer (q,l,z_projected) la projection du sommet vertex v sur la
        vue V_j.
        si Z^j_buffer[q,l] < z_projected alors
            v est un sommet caché.
            Marquer les triangles incident à v comme caché.
```

1.3.1.2 Visibilité partielle

Jusqu'à maintenant, chaque triangle a eu une des deux étiquettes : "totalement visible" ou "caché". A l'issue de cette étape, les triangles cachés seront classés en "partiellement visible" ou "caché". Pour cela, la profondeur de chaque pixel est comparé à la valeur du z-buffer stockée auparavant. L'algorithme permettant de calculer la visibilité partielle est présenté dans algorithme 4.

Algorithm 4: Détermination de la visibilité partielle.

```
// Parcours des vues
pour chaque vue V_j faire
    // Parcours des triangles
    pour chaque triangle t faire
        si triangle t est caché dans in V_j alors
            Déterminer P_j(t) la projection du triangle t sur la vue V_j.
            pour chaque pixel (q,l) in P_j(t) faire
                Accès à la profondeur z du pixel.
                si z ≠ Z^j_buffer[q,l] alors
                    Marquer t comme partiellement visible in V_j.
```

1.3.2 Calcul des images de distorsion

Afin de déterminer l'erreur de texturer un triangle avec une texture I_i, chaque texture disponible est plaquée sur le maillage d'entrée puis projeté sur chacune des vues. Soit $P_j^i(\mathcal{M})$ la projection du maillage $\mathcal{M}$ sur la vue V_j texturée avec l'image I_i. Pour chaque texture I_i l'image de distorsion suivante est calculé dans l'espace couleur RGB :

$$D_{i,j} = \|P_j^i(\mathcal{M}) - I_j\|_2$$

Algorithm 5: Calcul des images de distorsion.

```
   // Parcours des textures
1  pour chaque texture I_i faire
       // Parcours des vues
2      pour chaque vue V_j faire
3          Calculer P_j^i(M) : la projection du maillage texturé avec I_i et projeté
           sur la vue V_j.
4          D_{i,j} = ||P_j^i(M) - I_j||_2
```

1.3.3 Sélection de la meilleure texture

Dans cette section, la meilleure texture est sélectionnée pour chaque triangle en utilisant les images de distorsion calculées. La meilleure texture pour chaque triangle repose sur l'ensemble des pixels qui constituent la projection du triangle et qui sont visible dans la vue V_j, *i.e.* l'ensemble des pixels :

$$\text{Vis}_j(t) = \{(q,l) \in P_j(t),\ (q,l) \text{ visible in } V_j\}$$

La figure 1.8 montre comment $\text{Vis}_j(t)$ est calculé pour les triangles partiellement ou totalement visibles. La grille représente les pixels de l'image. Dans 1.8(a) le triangle bleu est totalement visible, les pixels définissant sa projection sur la vue V_j *i.e.* $\text{Vis}_j(t)$ sont montrés en rouge. D'autre part, le triangle bleu dans 1.8(b) est partiellement visible car il est occulté par le triangle vert. $\text{Vis}_j(t)$ dans ce cas est restreint aux pixels non-occultés. La métrique de photo-cohérence est ensuite calculée pour chaque triangle sur $\text{Vis}_j(t)$. Cette métrique mesure l'erreur de texturer le triangle t avec l'image de texture I_i. Dorénavant, par souci de simplicité, un triangle visible signifiera à la fois un triangle totalement visible et un triangle partiellement visible dans la mesure où ils seront traités de la même manière. $\forall i \in \{1,\dots,n\}$ on calcule :

$$\mathcal{E}_{t,I_i} = \sum_{\substack{j=1 \\ t \text{ visible in } V_j}}^{n} \left(\sum_{(q,l)\in \text{Vis}_j(t)} D_{i,j}[q,l] \right),$$

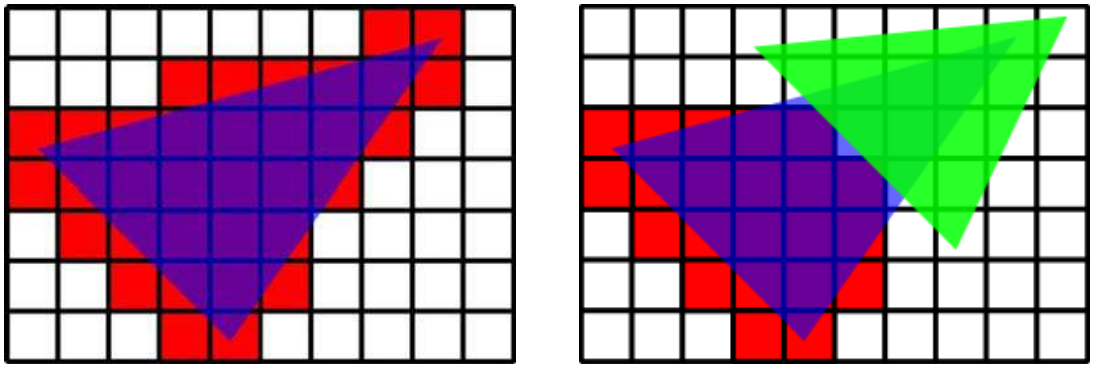
La meilleure texture pour un triangle visible t est donnée par la formule :

$$\hat{I}_t = \arg\min_{I_i \in \mathcal{I}} \mathcal{E}_{t,I_i} .$$

Les étapes du calcul de la meilleure texture sont résumées dans algorithme 6.

Algorithm 6: Calcul de la meilleure texture par triangle.

```
   // Parcours des triangles
1  for chaque triangle t do
       // Parcours des vues
2      for chaque vue V_j do
3          Déterminer Vis_j(t).
       // Parcours des textures
4      for chaque texture I_i do
5          Calculer E_{t,I_i}.
6      Déterminer Î_t.
```



(a) Visibilité totale (b) Visibilité partielle

Figure 1.8: Calcul de la meilleure texture. Gestion de la visibilité partielle/totale.

1.3.4 Plaquage de textures

Finalement, les vues originales sont rendues. Le modèle 3D est projeté en utilisant les paramètres caméras fournis par l'utilisateur. Chaque triangle étant texturé en utilisant les coordonnées de texture déterminées durant le calcul de la visibilité.

1.3.5 Discussion et résultats

Dans cette section, nous proposons de comparer la performance de notre algorithme de multi-texturation basé sur la métrique de photo-cohérence pour l'attribution de la meilleure texture avec des schémas de multi-texturation basés sur des critères géométriques pour l'attribution de la meilleure texture. Les critères géométriques utilisées sont :

- $C_1 := \langle \vec{d_j}, \vec{n_t} \rangle$ *i.e.* le produit scalaire entre la normale du triangle $\vec{n_t}$ et la direction de vue (*i.e.* l'axe optique qui est la normale au plan image) $\vec{d_j}$ de la vue V_j. Dans ce schéma, la vue où l'angle entre la direction de vue et la normale au triangle sera privilégiée. Soulignons au passage que dans le cas des caméras parallèles, ce critère n'est pas applicable.

- $C_2 := \langle \vec{e_j}, \vec{n_t} \rangle$, où $\vec{e_j}$ désigne la direction du rayon joignant le barycentre du triangle et la position de la caméra. La vue où le triangle est le plus parallèle au plan caméra sera donc privilégiée.

- $C_3 := \text{aire}_j(t)$ *i.e.* l'aire du triangle projeté sur la vue V_j. La vue où la résolution du triangle projeté est maximale sera privilégiée.

Les résultats sont présentés pour deux séquences breakdancers et balloons. Les modèles géométriques utilisées sont ceux correspondant à la fusion de cartes de profondeur présentés dans la section 1.2.

Les résultats des vues originales synthétisées sont présentées dans la figure 1.10 et dans la figure 1.9. On peut voir que l'utilisation de la métrique de photo-cohérence pour la multi-texturation du maillage fournit une meilleure qualité visuelle que les critères géométriques précédemment cités. Plus précisément encore, dans la figure 1.9 la frontière entre les deux murs marron est bien alignée en utilisant la métrique de photo-cohérence, alors que cette frontière est bruitée et mal-alignée en utilisant les critères géométriques. Le sol apparait également bruité en utilisant des critères géométriques et plus lisse en utilisant la métrique de photo-cohérence proposée. Ces résultats visuels sont renforcés par les résultats en termes de PSNR de la figure 1.11. En effet ces résultats montrent que la différence entre la méthode de photocohérence et toute autre méthode est remarquable (au moins une différence de 1dB entre les deux méthodes). Les résultats concernant la métrique de photocohérence montrent que le PSNR est plus grand pour les vues proches de la caméra centrale, ceci est dû au large nombre de triangle auxquels sont attribués la texture de la caméra centrale. En dépit des performances de l'algorithme basé sur le critère de photo-cohérence, l'algorithme proposé demande plus de temps de calcul que les critères géométriques cités (environ 10 fois plus). En conclusion, la multi-texturation basée sur la photo-cohérence est plus adaptée au rendu off-line. Par conséquent, les résultats suggèrent l'utilisation de ce critère dans un contexte de transmission de donnée où la minimisation de la distorsion est souvent un critère décisif à prendre en compte.

1.4 Transmission du modèle

Dans les sections précédentes, nous avons abordé le problème de modélisation de la géométrie. A l'issue de cette étape, on a pu générer un maillage triangulaire décrivant la géométrie de la scène. On a également décrit une nouvelle méthode de texturation du dit-maillage reposant sur la minimisation d'erreur lors du rendu. Dans cette section nous aborderons le problème de la transmission proprement dit. Un modèle de texture sera généré dans la section 1.4.1. Une méthode d'évaluation du modèle proposé sera ensuite présentée dans la section 1.4.2. Finalement, les résultats en termes de courbes débit-distorsion seront présentés dans la section 1.4.3.

1.4.1 Génération du modèle de texture

Dans cette section, nous allons extraire le modèle de texture à transmettre à partir de l'information de meilleure texture basée sur la métrique de photo-cohérence calculée dans la section 1.3. On peut alors extraire les fragments de texture utilisés par vue en utilisant l'algorithme 7 présenté ci-dessous :

Algorithm 7: Génération du modèle de texture à partir de l'information de meilleure texture.

```
   // Parcours des vues
1  pour chaque vue V_j faire
       // Parcours des triangles du maillage
2      pour chaque triangle t faire
3          Déterminer V_i la meilleure vue du triangle t
4          si i = j alors
5              texturer le triangle t avec texture correspondant à la vue V_i.
6          sinon
7              rendre le triangle t avec une couleur unie.
```

En sortie, on obtient une image par vue. Chaque image présente les pixels utilisés durant le rendu. Un example illustrant les textures générées à l'aide de cet algorithme est présenté dans la figure 1.12. Dans cette figure, on présente les fragments de texture utilisés pour les vues 0 et 6. Chaque image montre les pixels utilisés durant le texture mapping, les pixels non utilisés sont montrés en noir.

1.4.1.1 Textures alignées suivant les macroblocks

Pour des fins de transmission, attribuer une couleur unie pour les fragments de texture non-utilisés n'est pas suffisant. Les expériences que nous avons menées en utilisant le codec MPEG.4/MVC montrent que le débit des textures générées est supérieure au débit des textures originales. Notre première explication tient au fait que la structure de triangle n'est pas cohérente avec la structure de macroblock que requiert un codec vidéo. La prédiction d'un macroblock est plus optimale avec une transition lisse au sein de ce macroblock plutôt qu'une transition brutale comme celle présente dans le modèle de texture de la section 1.4.1. Afin de réduire le débit du modèle de texture à transmettre, nous proposons d'aligner les fragments de texture utilisés sur les macroblocks de l'image. Figure 1.14 illustre le résultat de l'alignement du modèle de texture précédent sur les macroblock de l'image.

1.4.1.2 Remplissage des Macroblocks

L'alignement proposé dans le paragraphe précédent nous a permis en effet de réduire le débit de l'information de texture que l'on souhaite transmettre. Cependant le débit est encore plus élevé que le débit des images originales. La raison est que la cohérence inter-vue a diminué. La prédiction du macroblock est donc plus couteuse que dans le cas des textures originales. Afin de diminuer le débit de la texture nous proposons de prédire chacun des macroblocks non utilisés

en utilisant un schéma de prédiction intra du standard MPEG.4/AVC. Dans nos expériences, nous avons choisi le mode de prédiction DC pour sa simplicité (*cf*. chapitre 3 pour plus de détails sur les différents modes intra utilisés). On s'attend à ce que les autres modes donnent les mêmes résultats également. Les résultats de cette méthode de prédiction sont présentés dans la figure 1.14.

1.4.2 Évaluation

Dans cette section, nous allons comparer la performance du modèle proposé composé du maillage (voir section 1.2) et des images de textures calculées dans la section 1.3.4 d'une part et du modèle de référence composé des images de textures originales et des cartes de profondeur associées. Cette évaluation est effectuée en considérant des images fixes (première image du flux vidéo). On propose d'évaluer la distorsion en évaluant la qualité des nouvelles vues synthétisées. Le débit est calculé pour la texture et la profondeur. Nous proposons cette évaluation en utilisant quatre séquences de test : Breakdancers, ballet balloons et kendo (*cf*. Annexe A). La méthode d'évaluation proposée consiste à transmettre un sous-ensemble des vues disponibles et synthétiser de nouvelles vues. L'évaluation de la distorsion est faite en calculant le PSNR entre les vues synthétisées et les vues originales.

1.4.2.1 Évaluation du modèle de référence

La méthode d'évaluation proposée pour les séquences breakdancers et ballet est présentée dans le schéma de la Figure 11.4. Afin d'évaluer le débit de l'information de profondeur, on considérera les cartes de profondeur associées aux vues 0, 2, 4 et 6. Ces cartes de profondeur sont encodées en utilisant MPEG.4/MVC pour les pas de quantification 22, 27, 32, 37, 42 et 47. Quant à la distorsion, elle est évaluée en synthétisant les vues 1, 3 et 5 en utilisant le logiciel MPEG/VSRS (*cf*. chapitre 9). La distorsion est calculée en utlisant le PSNR sur la composante de la luminance Y entre les textures synthétisées et les textures originales. De la même façon, cette évaluation a été effectuée pour les séquences balloons et kendo en considérant les cartes de profondeur associées aux vues 1, 3 et 5 et en synthétisant les vues 2 et 4.

1.4.2.2 Évaluation du modèle proposé

L'évaluation du modèle proposé est présenté dans la figure 1.16 présentant les résultats pour Microsoft breakdancers et ballet. D'abord un maillage triangulaire est extrait en fusionnant les cartes de profondeur associées aux vues 0, 2, 4 et 6 en utilisant le schéma de space carving présenté dans la section 1.2. Ce maillage fusionné est calculé pour trois résolutions volumétriques différentes : 50, 125 et 250 voxels. On rappelle que la résolution volumétrique correspond au nombre de voxels dans chaque direction. Une résolution volumétrique de 50 indique que le nombre de voxels de la grille englobante est 50 dans oX, oY et oZ. A cette étape, on peut évaluer le débit de la géométrie en utilisant MPEG-TFAN codec (*cf*. section 10.5). La prochaine étape est dédiée à la modélisation de la texture, d'abord la meilleure texture pour chaque triangle est calculée en utilisant la méthode décrite dans la section 1.3.4. Le choix est dans ce cas restreint aux vues 0, 2, 4 et 6. Un

modèle de texture comme présenté dans la section 1.3 correspondant au vues 0, 2, 4 et 6 est ainsi généré puis encodé en utilisant MPEG.4/MVC. Les textures décodés pour les différents pas de quantification sont générées puis plaquées sur notre modèle géométrique. On génère ainsi les vues 1, 3 et 5 et on calcule la moyenne des Y-PSNRs entre les images synthétisées et les images données en entrée. De la même façon, on évalue le modèle proposé pour les séquences balloons et kendo en considérant les vues 1, 3 et 5 comme entrées, en synthétisant les vues 2 et 4 et en calculant le Y-PSNR moyen sur ces dernières vues.

1.4.3 Résultats

Les résultats en terme de courbes débit/distorsion sont présentées dans les figures 1.17 et 1.18 pour les séquences breakdancers et ballet. Les figures 1.19 et 1.20 montrent les courbes débit/distorsion pour les séquences balloons et kendo respectivement. Ces résultats montrent un net avantage en faveur du standard MPEG.4/MVC malgré la qualité des vues synthétisées qui restent proches de la méthode VSRS pour le cas des séquences breakdancers et ballet. En effet, le problème majeur réside dans le débit qui reste supérieur par rapport au standard MPEG.4/MVC. Ce débit est particulièrement élevé pour coder le modèle géométrique et l'information de meilleur texture.

1.5 Conclusion

Dans cette thèse, nous avons abordé le problème de la représentation d'une scène à partir de séquences vidéos multi-vues plus profondeur. Considérant un contexte général où les caméras utilisées durant l'acquisition ne sont pas forcément parallèles. En outre, on suppose que les paramètres caméras sont connus et que les cartes de profondeur estimées sont fiables. Dans un premier temps nous avons abordé le problème de la fusion des cartes de profondeur. On a ainsi développé un nouvel algorithm basé sur une variante de l'algorithme de space carving. Selon ce schéma, un volume englobant la scène est défini puis subdivisé en éléments cubiques réguliers appelés voxels. La surface finale est extraite de cette représentation volumétrique grâce à l'algorithme de marching cubes. Le principal inconvénient de l'approche proposée c'est que la spécification de la résolution volumétrique doit être faite par l'utilisateur. Cependant, le choix judicieux de la résolution volumétrique s'avère d'une importance cruciale afin d'éviter tout phénomène de repliement de spectre. On peut, intuitivement, voir qu'il faut spécifier la résolution convenable de façon à ce que la surface finale prenne en compte toutes les variations géométriques de la scène. La détermination analytique du nombre de voxels nécessaires afin d'éviter ce repliement a été proposé par Brian Curless [Curless, 1997] pour le cas d'une seule vue dans le contexte d'une fonction de distance signée. L'extension de ce calcul pour l'ensemble des vues peut s'avérer fastidieux. Afin de remédier à cela, une autre alternative serait de considérer les structures hiérarchiques comme les octrees [Szeliski, 1993] à l'image de l'approche développée par [Fuhrmann and Goesele, 2011].

Le second axe de notre étude concerne la multi-texturation du maillage fusionné. Nous avons proposé une nouvelle approche de multi-texturation s'appuyant

sur la minimisation d'une métrique de photo-cohérence pour l'attribution de la meilleure texture pour chaque triangle. Par conséquent, l'approche proposée fournit de meilleurs résultats comparée aux méthodes géométriques existantes pour l'attribution de la meilleure texture. Cette approche nous a également permis de synthétiser de nouvelles vues grâce au maillage fusionné. La synthèse de vues intermédiaires grâce à notre schéma de modélisation géométrique et notre schéma de multi-texutration donne une moins bonne qualité que l'état de l'art basé image. L'inconvénient majeur de l'algorithme de multi-texturation réside dans sa complexité. En effet l'approche basée photo-cohérence est 10 fois plus lente que les approches basées sur les critères géométriques. Le temps de calcul pourrait être réduit si on opte pour une future implémentation de notre algorithme sur GPU.

Le troisième volet de cette thèse concerne la transmission proprement dite. Au lieu de transmettre le modèle de référence composé des cartes de profondeur et des images de texture. Nous avons proposé de transmettre le modèle géométrique et le modèle de texture. Ce modèle de texture est généré en calculant pour chaque vue les fragments de texture utilisés durant le rendu. Le modèle géométrique proposé est ensuite encodé avec le standard actuel de compression de maillage MPEG/TFAN. Quant au modèle de texture, il a été encodé avec le standard de compression vidéos multi-vues MPEG.4/MVC. Les résultats montrent que la performance du modèle de référence est meilleure que celle du modèle proposé en termes de compromis débit/distorsion. En effet, pour une valeur de distorsion donnée, le débit total pour transmettre le modèle proposé est en général supérieur au débit du modèle de référence. Dans le futur, on souhaite étudier comment réduire le débit en proposant une méthode plus efficace pour transmettre l'information de la meilleure texture et en étudiant la cohérence temporelle du modèle proposé.

(a) Image originale caméra 7.

(b) Zoom sur l'image originale de la caméra 7.

(c) Image synthétisée de la camera 7 en utilisant le critère du cone de vue C_2.

(d) Zoom sur l'image synthétisée de la caméra 7 en utilisant le critère du cône de vue C_2.

(e) Image synthétisée de la caméra 7 en utilisant le critère de photo-cohérence.

(f) Zoom sur l'image synthétisée de la caméra 7 en utilisant le critère de photo-cohérence.

Figure 1.9: Images synthétisées pour la séquence breakdancers en utilisant le critère de cone de vue vs. photo-cohérence

(a) Image original de la caméra 5.

(b) Zoom sur l'image originale de la caméra 5.

(c) Image synthétisée de caméra 5 en utilisant le critère résolution du triangle C_3.

(d) Zoom sur l'image synthétisée de la caméra 5 en utilisant le critère résolution du triangle C_3.

(e) Image synthétisée de la caméra 5 en utilisant le critère de photo-cohérence.

(f) Zoom sur l'image synthétisée de la caméra 5 en utilisant le critère de photo-cohérence.

Figure 1.10: Images synthétisées pour la séquence balloons en utilisant le critère résolution du triangle vs. photo-cohérence.

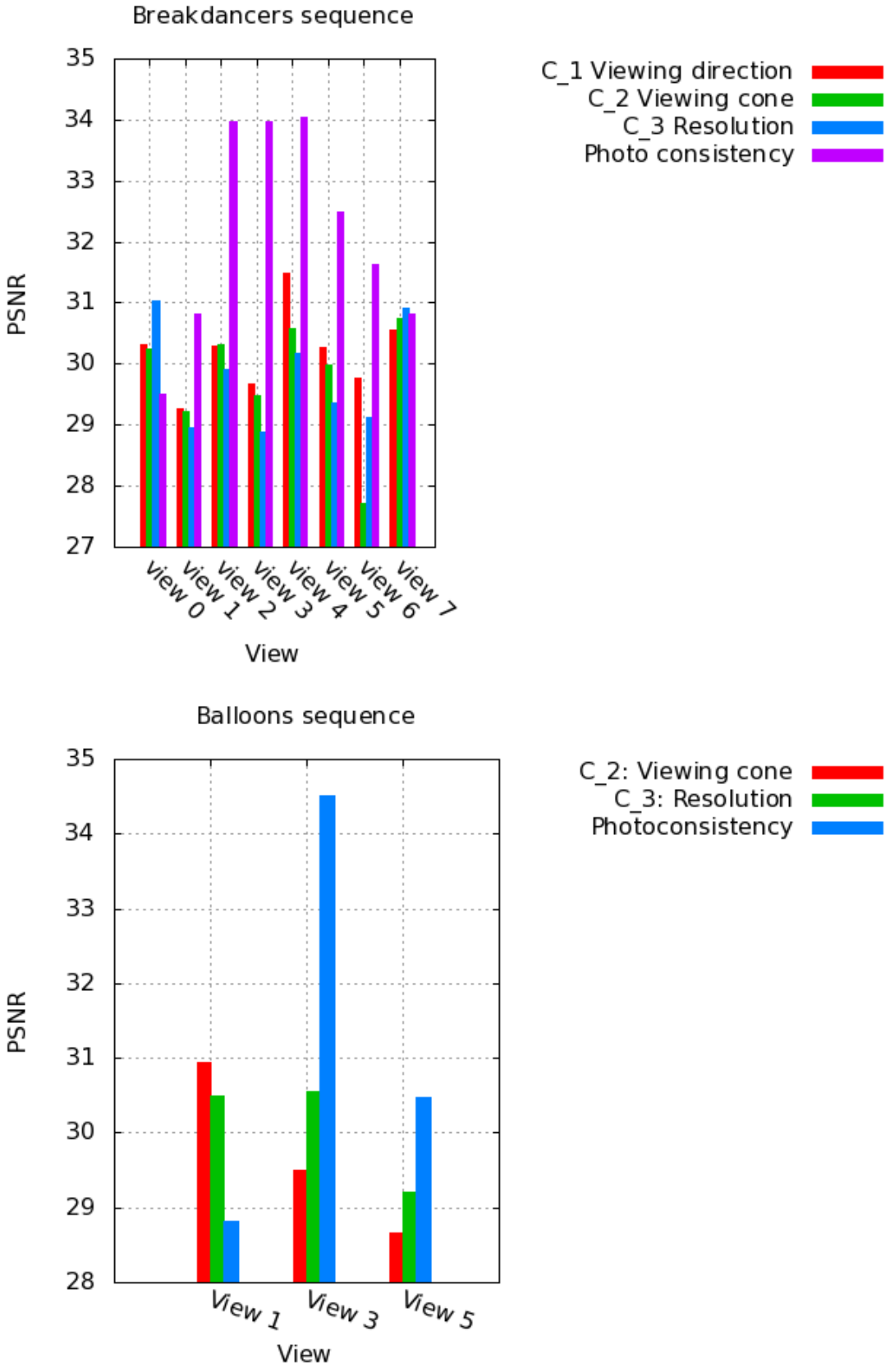


Figure 1.11: PSNR evolution through different views using different criteria for Breakdancers and Balloons sequences.

(a) Caméra 0 (b) Caméra 6

Figure 1.12: Modèle de texture pour les vues 0 et 6 pour la séquence Breakdancers. Les fragments de texture non utilisés sont montrée en noir.

(a) camera 0 (b) camera 6

Figure 1.13: Image de texture alignées suivant des Macroblock.

(a) camera 0 (b) camera 6

Figure 1.14: Macro block filling for efficient coding of multi-view texture maps.

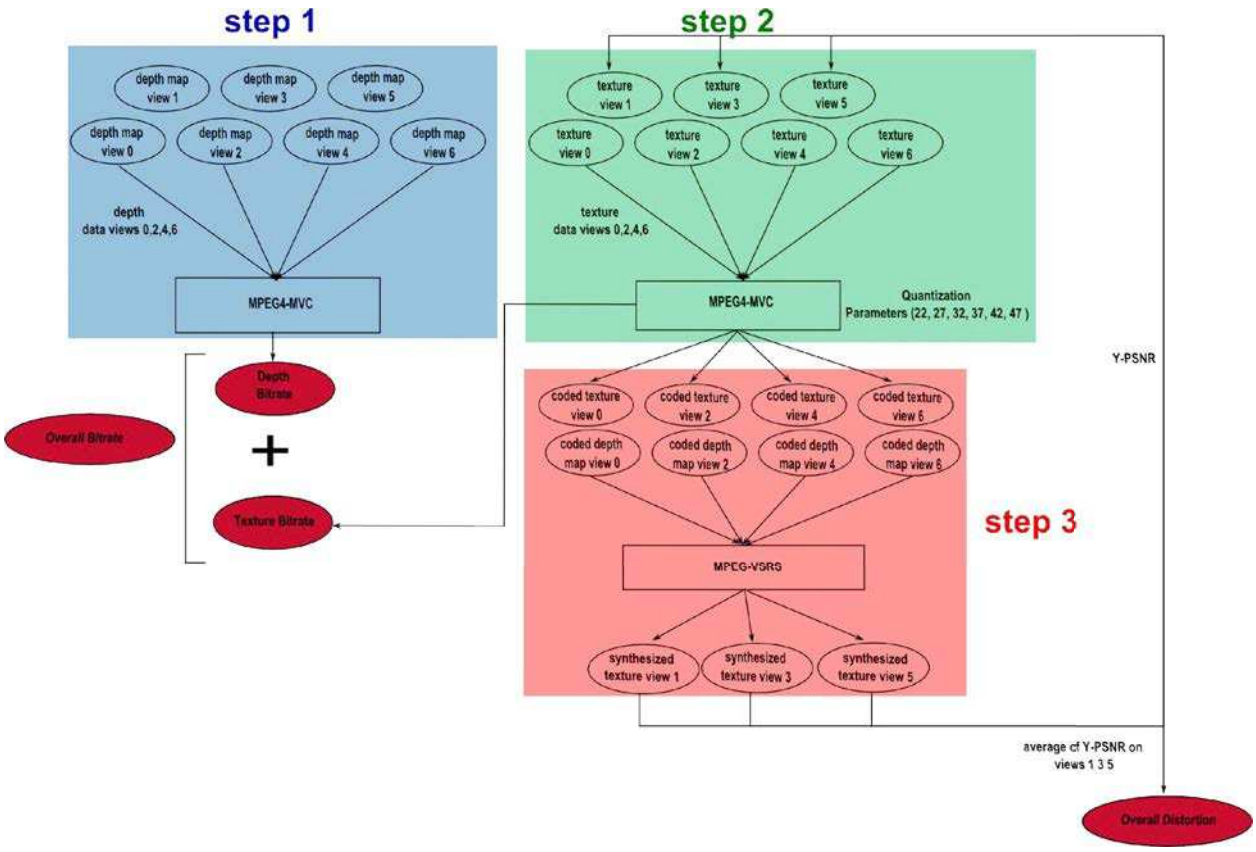


Figure 1.15: Évaluation du modèle de référence pour les séquences Microsoft ballet et breakdancers.

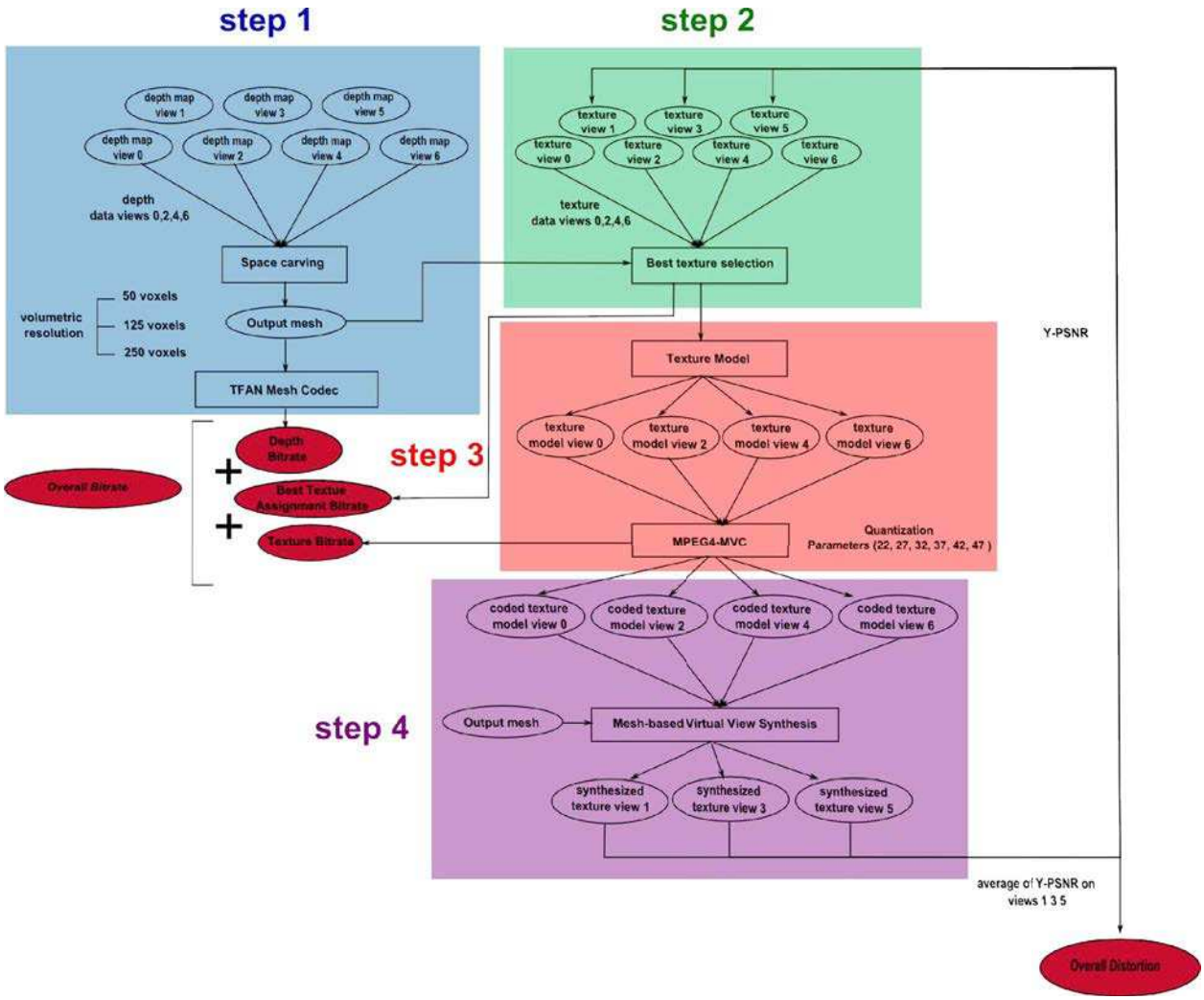


Figure 1.16: Évaluation du modèle proposé pour les séquences Microsoft ballet et breakdancers.

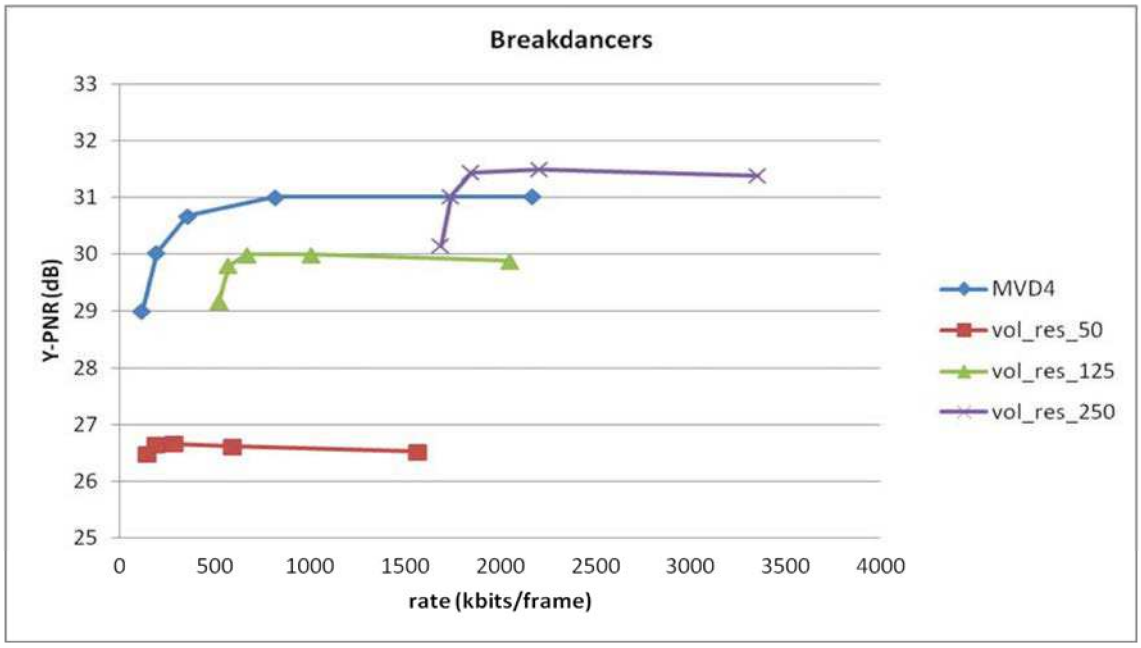


Figure 1.17: Courbes débit/distorsion pour différentes résolution volumétriques (50, 125 et 250) vs. le modèle de référence pour la séquence breakdancers.

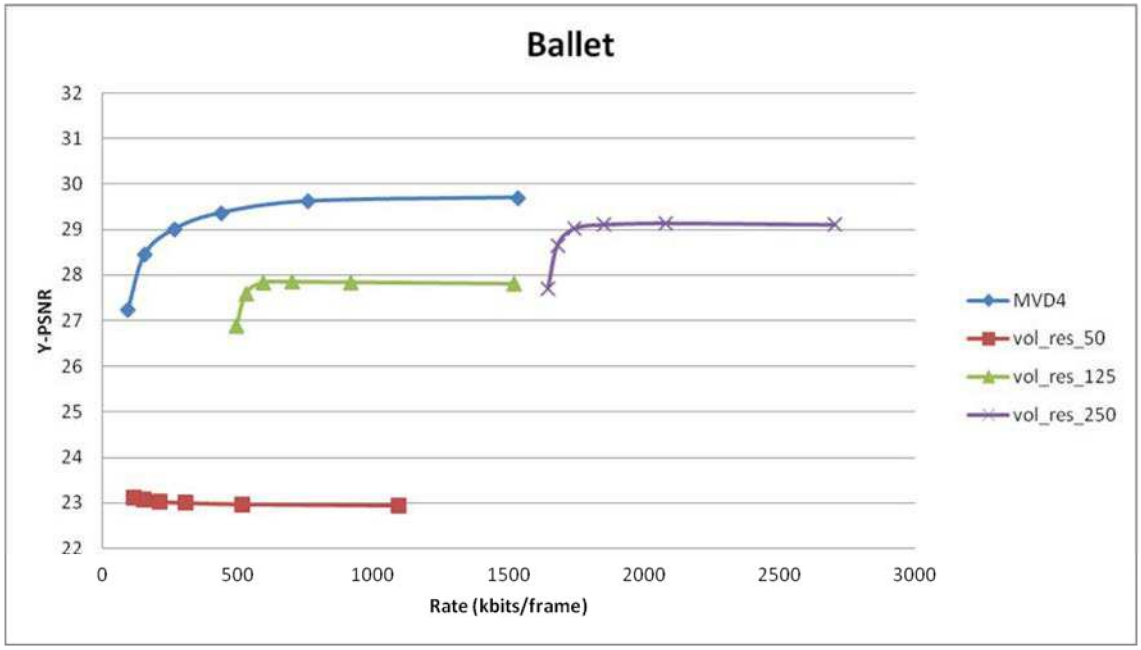


Figure 1.18: Courbes débit/distorsion pour différentes résolution volumétriques (50, 125 et 250) vs. le modèle de référence pour la séquence ballet.

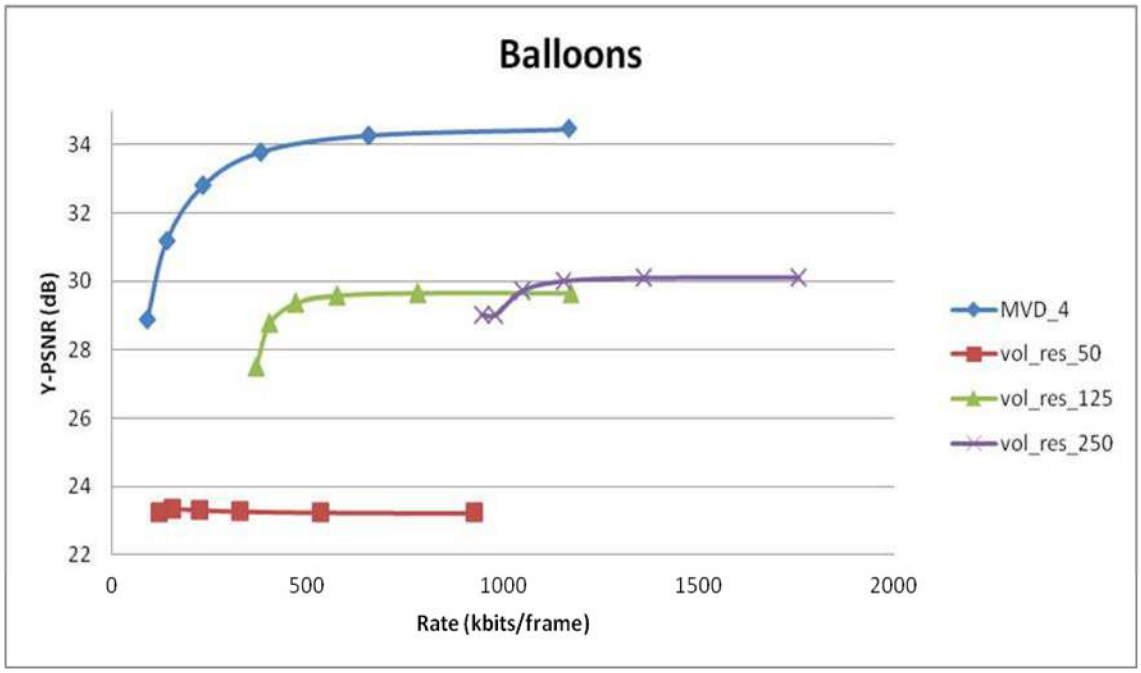


Figure 1.19: Courbes débit/distorsion pour différentes résolution volumétriques (50, 125 et 250) vs. le modèle de référence pour la séquence balloons.

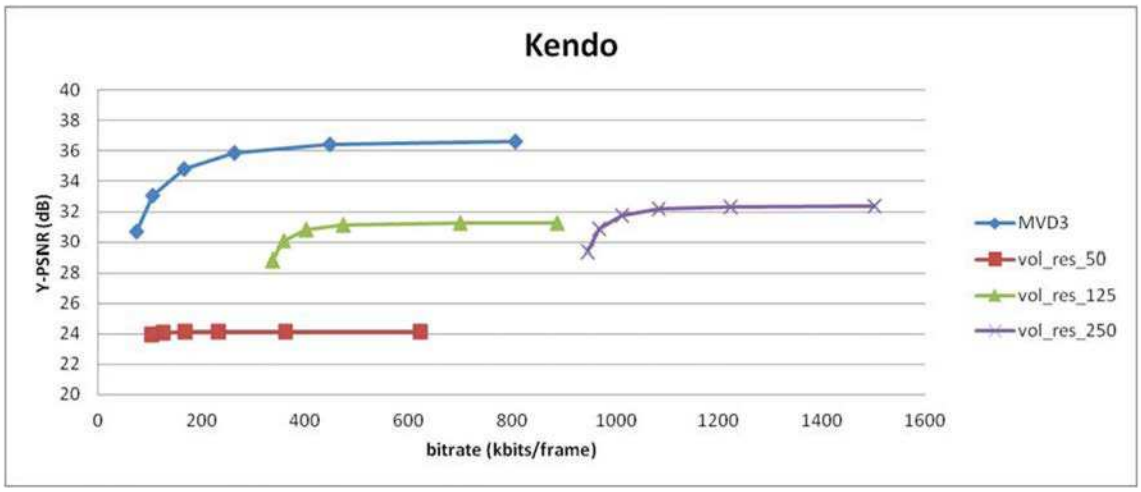


Figure 1.20: Courbes débit/distorsion pour différentes résolution volumétriques (50, 125 et 250) vs. le modèle de référence pour la séquence kendo.

Chapter 2

Introduction

Contents

Multi-View Imaging (MVI) has witnessed a growing interest in the last few years. The recent success of Hollywood blockbusters such as Avatar has undoubtedly led to the introduction of 3DTVs to the home, enriching therefore the 3D experience at the expense of existing HDTV displays. James Cameron even predicts that in three to five years there will be 100% adoption at the cinema level and rapid expansion in the home market [Child, 2011]. Despite these technical breakthroughs, several challenges related to scene capture, data representation or transmission are still remaining. Indeed, during scene acquisition, multiple cameras are used, hence several photometric errors may occur due to the changing illumination across the views, which increases the signal ambiguity during depth estimation. Scene representation is also a challenging task. One should strive to build a dense yet non-redundant scene model from the set of overlapping views. The compromise between model simplicity and easy rendering is usually hard to satisfy. The so-called scene representation is then subject to transmission through a communication channel. Determining the proper rate-distortion trade-off is another key point to take into account. At decoder side, high fidelity to original views is required. Besides, quality assessment using conventional objective metrics such as Peak of Signal to Noise Ratio (PSNR) or Structural Similarity Measure (SSIM) is not satisfying since reference images for such novel views are not available.

2.1 Context and contributions

In this thesis, we will focus on the problem of scene representation and transmission given Multi-view video plus Depth sequences (MVD). We make no assumption about cameras arrangement and consider therefore a general setting where capturing cameras could be either parallel or convergent. Besides, the capturing cameras are calibrated, hence cameras parameters are known. As far as depth maps are concerned, we do not address the complex issue of inter-view disparity estimation and consider the provided depth maps to be reliable. The purpose of this study is to build a compact representation of the scene intended to be transmitted with high fidelity at decoder side.

In this thesis, we propose an end-to-end multi-view video representation and compression chain. Starting from geometry and texture modeling via texture mapping and view synthesis and ending by compression of the proposed models. We also compare the performance of the proposed modeling system with the current multi-view video compression standard. More specifically, our contributions are the following:

- First, given the provided depth maps, we propose a new method for building a compact geometric model intended to represent the scene's geometry. Comparison with an existing point-based surface reconstruction method was performed and results in terms of visual and objective quality are also provided.

- Second, we propose a new multi-texturing method in order to texture the geometric model using the overlapping input texture images. The proposed multi-texturing method relies on a new image-based criterion for best tex-

ture selection. A comparison with geometric based criteria was also performed. The proposed multi-texturing method also allows virtual view synthesis of any view located between the input cameras.

- Finally, for transmission purposes, we derive from the proposed multi-texturing scheme a non-redundant texture model. An evaluation of the proposed geometry and texture models is proposed by comparing the compression efficiency with the current multi-view video compression standard MPEG-4/MVC.

2.2 Thesis outline

As stated above, we aim to remove redundancies present in multi-view video plus depth sequences. A straightforward approach is to exploit existing video compression standards which proved their efficiency for compressing such multi-view videos. Chapter 3 surveys these methods, first in a single view video case then shows how such techniques have been extended to handle multi-view video setup. However such standards rely on image-based means for scene representation. In chapter 4, we will review other existing techniques for scene representation and present the major advantages and drawbacks of each method. Chapter 5 discusses a particular category of techniques which is based on volumetric data structures. The result is a triangular mesh intended to represent the scene geometry. A new method that builds a triangular mesh from input depth maps is presented in chapter 6. We will also compare the performance of our algorithm with an existing algorithm in surface reconstruction from an oriented point cloud. In order to render original views or several novel views, an appropriate solution for texturing the mesh from the available texture images should be provided. Chapter 7 reviews existing methods for multi-texturing 3D models. We will focus our attention on different criteria that should be taken into account in order to render high quality images. Chapter 8 presents a new method for texturing 3D models that takes into account an image based criterion for texture mapping and which is optimal in the least squares sense. Based on this texturing scheme and the proposed depth maps fusion method, chapter 9 addresses the issue of virtual view synthesis from the constructed representation. We also compare the performance of the proposed method with an existing image-based view synthesis method. In order to address the compression issue, chapter 10 surveys existing mesh compression techniques and presents the current MPEG.4/TFAN standard for mesh compression. Finally, chapter 11 proposes a method for coding the texture signal and compares the compression performance of the proposed model with the current multi-view video compression standard.

Part I

Multi-view Video Representation and Coding: a State of the Art

Chapter 3

Multi-view Video Coding

Contents

3.1 Introduction

In this chapter, we will present existing techniques for coding Multi-view Video plus Depth (MVD) sequences. We need a deep understanding of such techniques since they will be considered as state-of-art reference and as a tool for coding the proposed model too. First, we will present in section 3.2 the basic ideas behind a single view video codec, considering as an example the MPEG-4/AVC (Advanced Video Coding) video codec standard currently used in several applications. Then we will show how these ideas are implemented in the current multi-view video standard MPEG-4/MVC (Multi-view Video Coding) in section 3.3.

3.2 Concepts of a Single View Video Codec

In this section, we will present some concepts related to video coding techniques. More details could be found in [Richardson, 2003]. Given an input video, a video codec (enCOder/DECoder) encodes a video sequence into a compressed form named video bistream. The video bitstream can decoded to reconstruct an approximation of the input video in case of lossy compression, or exactly the original video un case of losseless compression. In the remaining of this chapter, we will focus our attention on lossy compression methods only. Such methods expect a coding ratio greater than $\frac{1}{100}$. Typically, conventional hybrid video encoders, namely MPEG1-2, AVC-H264 and the incoming HEVC (High Efficiency Video Coding), partition the video images (a.k.a. frames) into squared blocks (also called Coding Units or Macroblocks) that are encoded sequentially. Video images are then encoded following the three following steps (*cf*. Figure 3.1): first spatial or temporal prediction, second spatial transformation and quantization and finally entropy coding. In the following subsections we propose to study in details each of these steps.

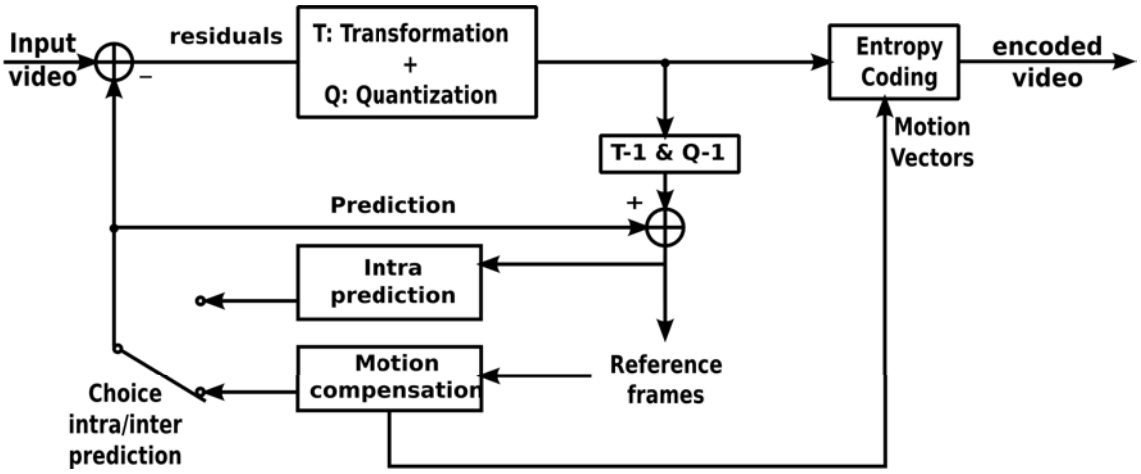


Figure 3.1: Video coding block diagram.

3.2.1 Spatial Prediction

Spatial prediction (also called intra prediction) consists in estimating the pixel's values (Y, U, V for YUV color space) of the current block using neighboring pixels. A block B is predicted according to previously coded and reconstructed blocks. AVC proposes 9 prediction modes which can be used for prediction. Figure 3.2 shows these modes with pixels' labeling in the top-left. The samples labeled A-M have been previously encoded and can be used as prediction reference. Samples a,b,c,..., p of the prediction block B are computed based on linear combination of these samples. For example, if mode 4 is selected, the top-right sample labeled 'd' is predicted by: round $\left(\frac{B}{4} + \frac{C}{2} + \frac{D}{4}\right)$.

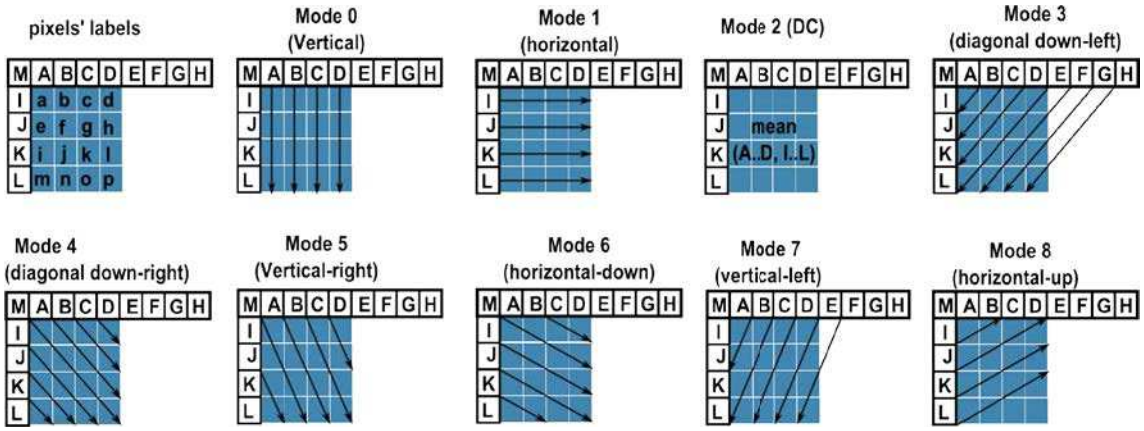


Figure 3.2: Spatial prediction modes.

3.2.2 Temporal Prediction

The purpose of temporal prediction is to exploit similarities between consecutive video images (or frames) in order to reduce temporal redundancy. The output of the temporal prediction step is a residual block B_{res} computed as the difference between the predicted block and the original block to encode. The current block B_{curr} is predicted from a reference frame I_{ref} (past or future frame, previously coded) using a motion estimation function. The main steps of temporal prediction are summarized in algorithm 8. First I_{curr} is partitioned into *macro-blocks*. A macro-block corresponds to a square region of a frame of 16×16 pixels. Motion estimation of a macro-block consists in finding a macro-block in a reference frame that matches the current macro-block. The corresponding macro-block in the reference frame I_{ref} is searched in order to find the macro-block that minimizes a matching criterion. The selected macro-block B_{ref} is the prediction signal that is subtracted to the current macro-block to produce a residual signal that is encoded and transmitted together with a motion vector describing the position of the best matching macro-block anda reference index indicating which reference frame has been used.

Algorithm 8: Block residual signal computation.

Data: The current block to be coded B_{curr}. A previously coded frame I_{ref}.
Result: Residual block B_{res}.

```
1 Decompose I_curr into macro-block.
  //  Macro-block traversal
2 for each macro-block MB_curr in I_curr do
3     Find the macro-block MB_ref in I_ref that best matches MB_curr.
4     Compute the residual macro-block MB_res = MB_ref − MB_curr.
5     Code MB_res (transformation, quantization, entropy coding).
```

3.2.3 Spatial Transformation and Quantization

Transformation aims to decorrelate the residual data and converts it into a form that can be easily compressed using an entropy coder. Such transformation consists in projecting the signal onto a new basis for a better compactness of this signal. Several criteria should be considered to choose the transformation to be applied: the capacity of the transformation to decorrelate the data, the transformation complexity, its reversibility.

Existing approaches include block based transforms such as Karhunen-Loeve Transform (KLT), Singular Value Decomposition (SVD) and Discrete Cosine Transform (DCT) or image based approaches like Discrete Wavelet Transform (DWT). Block based transforms have low memory requirements and are more suited to block based motion compensation. However at low bit rates, block-based schemes typically lead to discontinuities between reconstructed adjacent blocks, also called block artifacts. The block artifacts can be reduced using (in-loop) post-filtering. Image based approaches like DWT alleviates such artefacts but require tremendous amount of memory as they operate on the whole image. Due to its wide use in image and video compression standards, we will present the Discrete Cosine Transform (DCT).

3.2.3.1 Discrete Cosine Transform

The Discrete Cosine Transform operates on each residual block $\mathbf{X}$ and creates $\mathbf{Y}$, a block of $N \times N$ coefficients. The DCT (and its inverse, the IDCT) can be described in terms of a transform matrix $\mathbf{A}$. The relation between $\mathbf{X}$ and $\mathbf{Y}$ is given by the following:

$$\mathbf{Y} = \mathbf{A}\mathbf{X}\mathbf{A}^T$$

The IDCT is given by:

$$\mathbf{X} = \mathbf{A}^T\mathbf{Y}\mathbf{A}$$

Elements of the $N \times N$ matrix $\mathbf{A}$ are given by:

$$A_{ij} = C_i \cos\frac{(2j+1)i\pi}{2N} \text{ where } C_i = \begin{cases} \sqrt{\frac{1}{N}} & \text{if } i = 0, \\ \sqrt{\frac{2}{N}} & \text{if } i > 0. \end{cases}$$

3.2.3.2 Quantization

The role of quantization is to map a signal **X** with a wide range values to a quantized signal **Y** with a reduced range of values. In the context of video coding, quantization is used in order to reduce the precision of the transformed signal and hence its entropy. Another benefit is to remove near zero coefficients The quantized signal could be then coded with fewer bits than the original signal as the range of the quantized values is smaller. Quantization is performed by rounding **X** samples to the nearest integer using the following equation:

$$\mathbf{Y} = \text{round}\left(\frac{\mathbf{X}}{Q_{\text{step}}}\right)$$

where Q_{step} is the quantization step. If the quantization step is large the range of quantized values is small and can therefore be efficiently compressed (low entropy) but the quantized values are a coarse approximation of the original residuals. It is noteworthy that the quantization is a lossy process (*i.e.* not reversible) as it is not possible to determine the exact values of X samples from the rounded values of **Y** samples.

3.2.4 Entropy Coding

3.2.4.1 Basic Concepts of Information Theory

An information source is characterized by the set of symbols $X = \{x_i\}_{i\in[0,\,M-1]}$ and their probabilities of occurrence $P = \{p_i\}_{i\in[0,\,M-1]}$. The information associated with a symbol x_i $(0 \leq i \leq M-1)$, also called self information, is defined as:

$$I_{x_i} = \log_2\left(\frac{1}{p_i}\right) = -\log_2(p_i) \text{ (bits)}. \tag{3.1}$$

Equation 3.1 shows that if x_i is a certain event, *i.e.* $p_i = 1$, then $I_{x_i} = 0$. In other words, x_i can be predicted without any uncertainty. Besides, if x_i is a rare event, *i.e.* $p_i \to 0$, then $I_{x_i} \to \infty$. The information content of the source X can be measured using the source entropy $H(X)$, which is a measure of the average amount of information per symbol. The source entropy $H(X)$ is defined as the expected value of the self-information and is given by:

$$H(X) = \mathbb{E}\{I_{x_i}\} = -\sum_{i=0}^{M-1} p_i \log_2(p_i) \text{ (bits per symbol)}. \tag{3.2}$$

It has been shown [Cover et al., 1991] that the entropy $H(X)$ is a lower bound on the average number of bits to represent X. This was the idea behind Huffman coding [Huffman, 1952]. In this scheme, the length of each codeword depends on the inverse probability of occurrence of the corresponding symbol. More frequently occurring symbols are represented with short codewords, whereas symbols occurring less frequently are represented with longer codewords. The major drawback of Huffman coding is that it assigns an integer number of bits to each symbol which is suboptimal. The optimal number of bits for a symbol depends on the information content which is usually a fractional number. Based on this

idea, *arithmetic coding* outperforms Huffman coding. An arithmetic encoder converts a sequence of data symbols into a single fractional number and tends to approach the optimal fractional number of bits required to represent each symbol. In section 3.2.6, we will see how arithmetic coding is efficiently implemented within Context-based Adaptive Binary Arithmetic Coding (CABAC) in the current MPEG4/H.264 video compression standard.

3.2.5 Performance

The performance of a video codec is a trade-off between three variables: the objective or subjective quality or the reconstructed images, the encoded video's bit-rate and the computational complexity. In the context of video compression, bit-rate refers to the number of bits to be transmitted through a communication channel per time unit. The bitrate is measured in terms of bits per second (bps) or kilobits per second (Kbps). One seeks to minimize the bitrate for channel bandwidth's restrictions. In this section we will review how quality is measured in video coding schemes and how rate-distortion trade-off can be controlled.

3.2.5.1 Distortion Measures

The quality of reconstructed videos is measured by evaluating the distortion caused during the encoding process. This distortion is measured between the reconstructed and the original video. Most of video encoders consider the Mean Square Error (MSE) between the two videos. That is considering a reference image I and a reconstructed image $\hat{I}$:

$$MSE = \frac{1}{MN} \sum_{i=0}^{M-1} \sum_{j=0}^{N-1} [\hat{I}(i,j) - I(i,j)]^2 \tag{3.3}$$

Where M and N denote the width and the height of image I and $\hat{I}$. Computation of the MSE is very simple however its main drawback is that is depends on image intensity scaling. Peak of Signal to Noise ratio avoids this limitation by normalizing the MSE with the square of the highest possible signal value in the image that is $2^n - 1$ where n is the number of bits per image sample:

$$\begin{aligned} PSNR &= -10 \log_{10} \frac{MSE}{(2^n - 1)^2} \\ &= 20 \log_{10} \frac{(2^n - 1)}{MSE} \text{ (decibels).} \end{aligned}$$

The SSIM metric [Wang et al., 2004] is another method of measuring similarity between images. Basically, the SSIM separates the task of similarity measurement into three comparisons: luminance $l(x,y)$, contrast $c(x,y)$ and structure $s(x,y)$. The luminance estimation of each signal is estimated as the mean intensity, *i.e.*:

$$\mu_x = \frac{1}{N} \sum_{i=1}^{N} x_i \tag{3.4}$$

The luminance comparison function $l(x,y)$ is then a function of μ_x and μ_y. The authors use the standard deviation to measure the signal contrast,

$$\sigma_x = \left(\frac{1}{N-1} \sum_{i=1}^{N} (x_i - \mu_x)^2 \right)^{\frac{1}{2}}$$

The contrast comparison is then the comparison of σ_x and σ_y. The structure comparison $s(x,y)$ is conducted on the normalized signals: $\frac{x-\mu_x}{\sigma_x}$ and $\frac{y-\mu_y}{\sigma_y}$. The overall SSIM score between the signals x and y is written as the following:

$$SSIM(x,y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \tag{3.5}$$

Where $C_1 = (K_1 L)^2$, $C_2 = (K_2 L)^2$ and L is the dynamic range of pixel values (255 for 8-bit grayscale images) and $K_1 \ll 1$ and $K_2 \ll 1$ are small constants.

However, all these measures are only computational and suffer from severe limitations. For instance, the PSNR measure requires an existing reference video which may not be available in practice. Moreover, the PSNR is not a relevant measure for subjective quality assessment. Actually, a decoded video is intended to be viewed by humans. Human visual system properties must therefore be taken into account during distortion evaluation. Defining such measure a is still an active research topic. In this context, ITU-T Video Quality Experts Group (VQEG)[1] investigates solutions in order to develop standards for objective video quality assessment taking into account the human visual system.

3.2.5.2 Rate-distortion optimization

During encoding, several options may exist to encode the current frame (coding modes, motion vectors, etc). The challenge is to choose the best coding options corresponding to the best trade-off between encoding cost (number of bits) and reconstructed video quality. To this end, one could minimize the distortion D for a given rate R_c by appropriate selection of such coding options. In other words:

$$\begin{aligned} &\text{Find } \min(D) \\ &\text{subject to } R \leq R_c \end{aligned} \tag{3.6}$$

Solving such problem is not an easy task. Equation 3.6 is usually reformulated as:

$$\begin{aligned} &\text{Find } \min(\mathcal{J}) \\ &\text{where } \mathcal{J} = D + \lambda.R \end{aligned} \tag{3.7}$$

Considering a quantization step Q_{step} and the assumption of high-bitrate environment, it has been shown [Gish and Pierce, 1968,Sullivan and Wiegand, 1998] that λ could be expressed as the following:

$$\lambda_{\text{MODE}} \simeq 0,85\ Q_{\text{step}}^2 \tag{3.8}$$

[1] See www.vqeg.org for more details about this group's activities

This formulation of λ_{MODE} was adopted in MPEG-4 AVC software in order to determine the cost of a prediction mode. Besides, another λ_{MOTION} is used for computing motion vectors. It has been shown experimentally [Sullivan and Wiegand, 1998] that the value of λ_{MOTION} should be set to:

$$\lambda_{\text{MOTION}} = \sqrt{\lambda_{\text{MODE}}} \tag{3.9}$$

3.2.6 MPEG-4 AVC/H.264 Characteristics

In 2003, the Moving Picture Experts Group (ISO/MPEG) and Video Coding Experts Group (ITU-T/VCEG) released a new video compression standard entitled "Advanced Video Coding" (AVC). This standard outperforms former MPEG-4 and H.263 standards without altering the general structure of a predictive video codec presented in 3.2. In this section we will present the main features of MPEG-4 AVC that will be used to multi-view video setup. Such features can be summarized in Macroblock partitioning of each video frame, enhancing macroblock prediction by extending reference images to future frames in temporal order and finally by using a context-based adaptive binary arithmetic coding method (CABAC).

3.2.6.1 Macroblocks partitioning

Each 16×16 macroblock may be partitioned into blocks of different sizes (16×8, 8×16, 8×8) (*cf*. Figure 3.3). A 8×8 can further be split up into sub-macroblocks with sizes 8×4, 4×8 or 4×4 (*cf*. Figure 3.4). Choosing large partition size (*e.g*. 16×16) means that small number of bits are required to encode the corresponding motion vector. In contrast choosing small partition size (*e.g*. 4×4) gives a low energy residual but requires a larger number of bits to encode the associated motion vector.

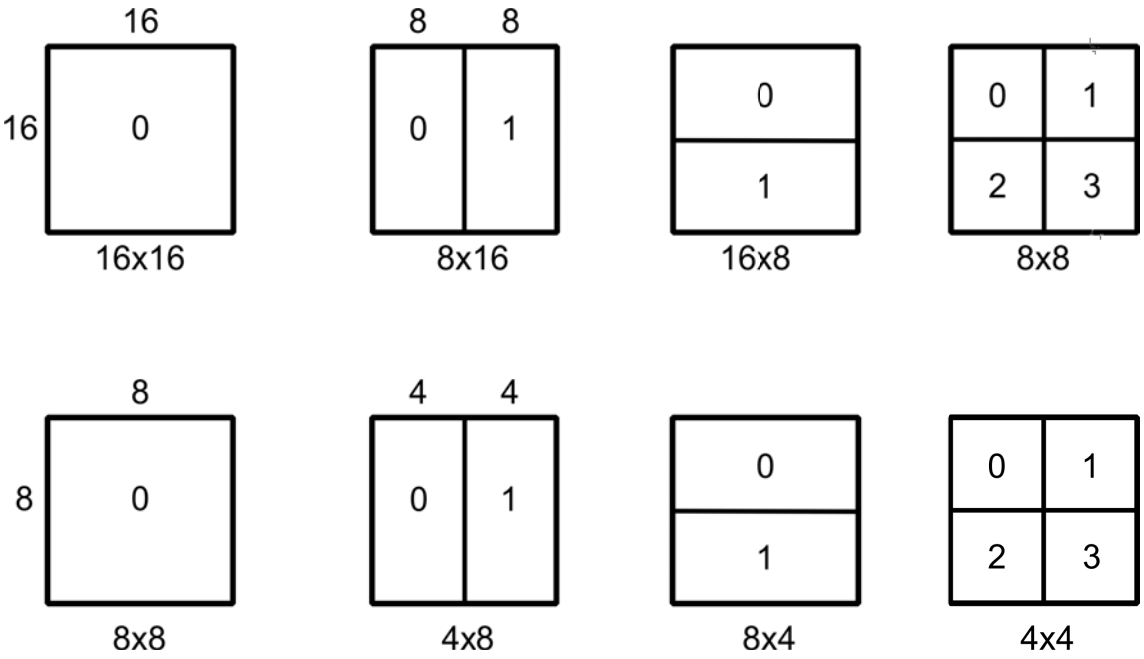


Figure 3.3: Macroblock partitions: 16×16, 8×16, 16×8, 8×8.

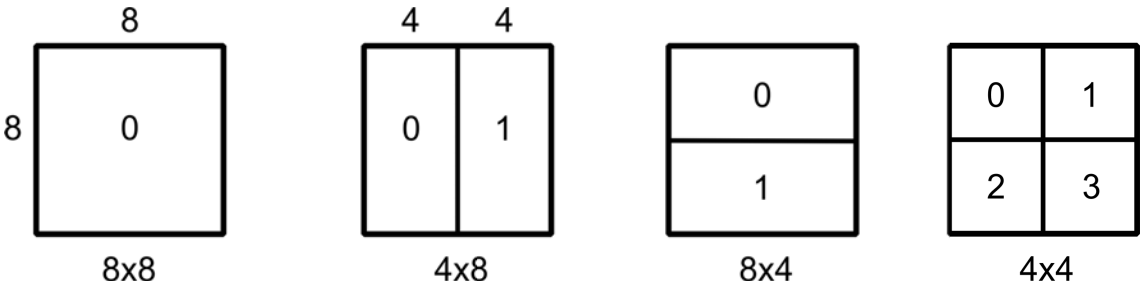


Figure 3.4: Sub-macroblock partitions: 8×8, 4×8, 8×4, 4×4.

3.2.6.2 Enhancing Inter Prediction

Existing MPEG standards decompose the input video sequence into Group Of Pictures (GOP). Each GOP may include three types of pictures as shown in Figure 3.5

- Intra picture (I-picture): such picture is encoded using the information contained in the picture itself. No motion estimation is included during intra picture coding, only spatial transformation is needed.
- Predicted picture (P-picture): the macro-blocks of this picture are encoded using a previously coded ones. Macroblocks of this picture could be predicted in inter or intra modes.
- Bi-directional predicted picture (B-picture): such picture may use both past and future picture as references to be predicted. The bi-prediction is an average of the 2 inter-predictions.

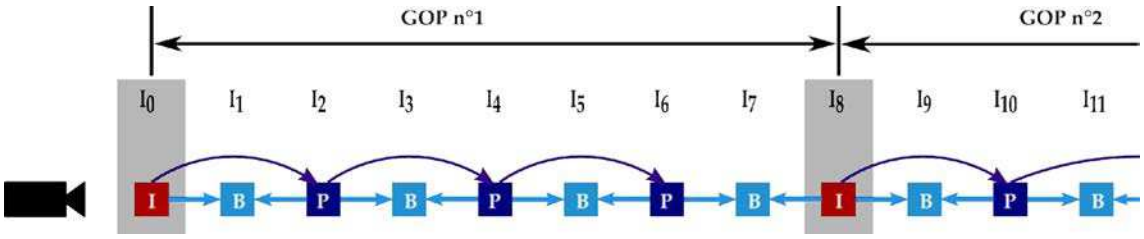


Figure 3.5: Prediction structures in earlier MPEG video standards.

The novelty of H.264/AVC is to allow to B-pictures to be used as reference to predict other B-pictures (*cf*. Figure 3.6 depicting the hierarchical prediction structure). More precisely, a macro-block can be predicted from one to two reference frames, before or after the current picture in temporal order. Such flexibility allows many options for choosing the prediction references for macro-block partition and hence leads to a more accurate macroblock partition prediction. Consequently fewer bits are required to transmit the residual signal than in earlier MPEG standards.

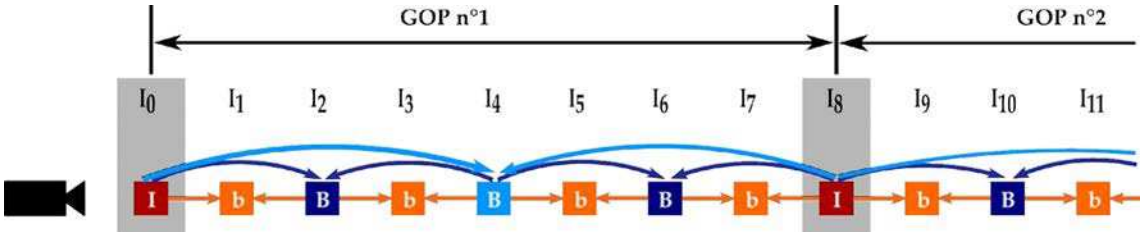


Figure 3.6: Prediction structures in H.264/AVC standard yielding a hierarchical GOP predicting structure.

3.2.6.3 Context-based Adaptive Binary Arithmetic Coding (CABAC)

Context-based Adaptive Binary Arithmetic Coding (CABAC) [Marpe et al., 2003] was implemented in H.264/AVC. Encoding a symbol involves four steps (*cf*. Figure 3.7)

1. Symbol binarisation: in this step, all non-binary symbols (*e.g*. transform coefficient) are mapped onto binary codewords. Each bit of this binary codeword is referred to as a *bin*.

2. Context modelling: a suitable context model is chosen for the current bin according to past observable symbols. A context model is a probability model that is selected from a set of predefined models depending on the probability of the past-coded symbols.

3. Arithmetic encoding: an arithmetic coder encodes each bin according to the symbols statistics by approaching the fractional number of bits necessary to represent the bin.

4. Probability update: the selected context model is updated with respect to the encoded bin (*e.g*. if the current bin is "1" the probability of occurrence of "1"s is increased).

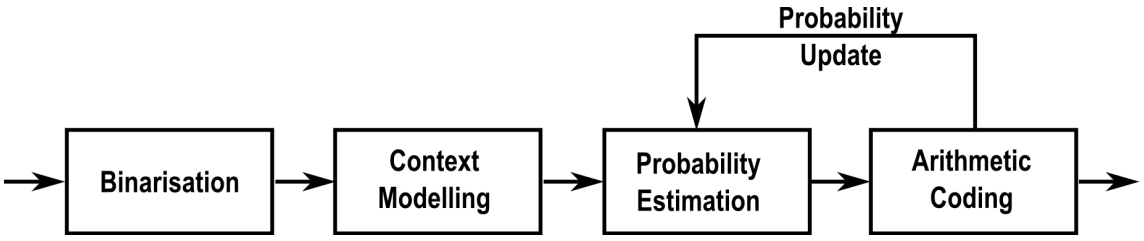


Figure 3.7: CABAC bloc diagram.

3.3 H.264-based Multi-view Video Coding

3.3.1 Simulcast Coding

A straightforward approach for coding multi-view video data consists in encoding multiple views using H.264/AVC for each view independently. This process

is referred to as *simulcast coding*. The advantage of simulcast coding is its conceptual easiness and backward compatibility with single view video coding technology. However, the major drawback of this approach is that coding efficiency is suboptimal as simulcast coding does not exploit the inter-view similarities. In the particular case of two views, asymmetrical coding (*i.e.* one view is coded with less quality than the other view) shows that significant savings in terms of bitrate could be achieved. In this scheme, one of the views is more coarsely quantized than the other yielding an imperceptible impact on the stereo quality.

3.3.2 Multi-view Video Coding: H.264 MVC

The Joint Video Team (JVT) has developed a multi-view extension based on H.264/AVC to handle the inter-view statistical dependencies. This extension is referred to as Multi-view Video Coding MVC. In this scheme, redundancy over time and over views are both exploited. Similarly to the temporal prediction using motion estimation and compensation (*cf*. 3.2), inter-view prediction in MVC is conducted by the so-called disparity estimation and compensation. Figure 3.8 shows that pictures are not only predicted from temporal neighbours but also from spatial neighbours in adjacent views leading to significant coding gains compared to H.264/AVC simulcast solution.

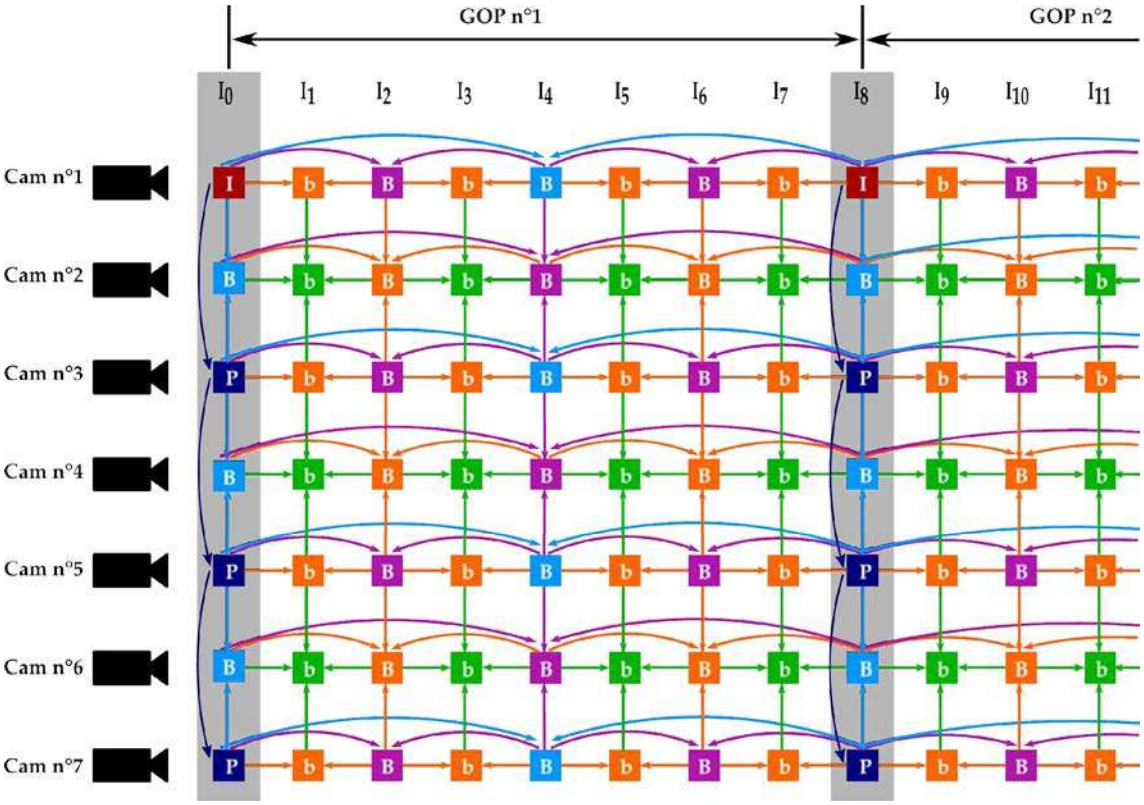


Figure 3.8: Prediction structures in MVC.

3.4 Conclusion

In this chapter we reviewed the structure of a single view video codec and showed how this structure can be extended to a multi-view video setup. Experimental results show that exploiting inter-view statistical dependencies allow substantial bit savings than when coding each view independently (simulcast). One common point of these approaches is the use of the set of the acquired multi-view videos as a representation of the 3D scene. Which is referred to as image based scene representation. The advantage of image based scene representation is to yield high quality videos at decoder side. However for 3DTV and FTV applications, the quality of the decoded views is not the only criterion to take into account. In the next chapter we will discuss several criteria for scene representation and present some solutions suitable for this study.

Chapter 4

3D Scene Representation

Contents

4.1 Introduction

In this chapter, we will review existing methods intended to represent a 3D scene. Representing a scene means describing it using a certain type of data (image, video, geometric model, *etc*.) which is appropriate to the desired application. The purpose of this chapter is to provide a better understanding of existing scene representation strategies and their properties in order to choose a suitable representation for 3DTV and FTV applications. Scene representation methods are classified into three categories: First image-based scene representations, which are studied in section 4.2. Second, representations relying on geometric models, these methods are presented in section 4.3. Finally methods that use both image and geometric models that will be presented in section 4.4. More detailed surveys can be found in [Smolic and Kauff, 2005, Alatan et al., 2007, Colleu, 2010]

4.2 Image-based representations

In this section, we will describe existing methods for scene representation based on the set of captured images. First we will study methods that represent the scene using only texture (or photometric) information captured by cameras. Second, we will present some methods that use depth information jointly with texture information as a scene representation.

4.2.1 Texture image based representation

Such representations do not require any depth information. Only the set of input textures are transmitted.

4.2.1.1 Plenoptic Modelling:

In this scheme, representing a scene is formulated as a sampling operation. Samples are taken from a continuous signal called the *plenoptic function*. The concept of the Plenoptic function was first introduced by Adelson and Bergen in [Adelson and Bergen, 1991]. Plenoptic derives from the Latin root *plenus* which means complete or full. They postulated that :

> *"The world is made of three-dimensional objects, but these objects do not communicate their properties directly to an observer. Rather, the objects fill the space around them with the pattern of light rays that constitutes the plenoptic function, and the observer takes samples from this function. The plenoptic function serves as the sole communication link between physical objects and their corresponding retinal images. It is the intermediary between the world and the eye."*

More formally, the plenoptic function $\mathcal{P}$ can be defined as the radiance perceived by any ray whose direction is defined by an azimuth θ and an elevation ϕ and a band of wavelengths λ at an arbitrary viewpoint (V_x, V_y, V_z) and at any time t (*cf*. Figure 4.1), *i.e*. the 7D function:

$$\mathcal{P} = P_7(\theta, \phi, V_x, V_y, V_z, \lambda, t) \tag{4.1}$$

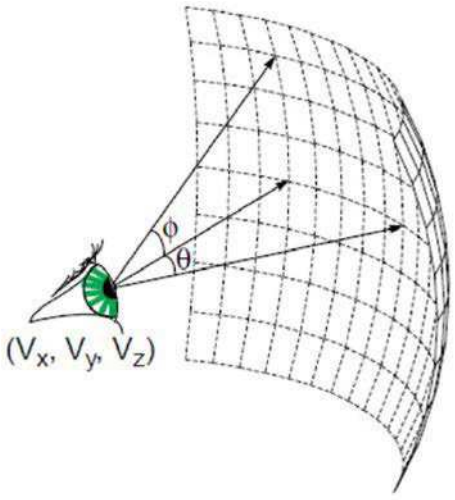


Figure 4.1: Plenoptic function (image from [McMillan and Bishop, 1995]).

The problem with this formulation of the plenoptic function is that it is so general (dynamic scenes, different lighting conditions, *etc*.) that sampling the whole 7D function into a unique representation is unfeasible. Research activities have been carried out in order to simplify the 7D plenoptic function by making reasonable assumptions. McMillan and Bishop [McMillan and Bishop, 1995], who first use the expression *plenoptic modeling*, assume static scenes and fix lighting conditions, hence they drop the time variable t and wavelength λ in the expression 4.1. In other words, their proposed approximation of $\mathcal{P}$ is as the following:

$$\mathcal{P} \simeq P_5(\theta, \phi, V_x, V_y, V_z) \tag{4.2}$$

In the proposed plenoptic modeling scheme, the authors propose a system for acquiring samples of the plenoptic function P_5 and a method in order to recover the plenoptic function from these samples. They call *Image Based Rendering* (IBR) the process of recovering the plenoptic function from a set of samples. In order to acquire the plenoptic samples, a static scene is recorded by positioning cameras in the 3D viewing space, each camera is positioned on a tripod capable of a continuous panning. At each camera position, they perform a cylindrical projection obtained from the acquired images during panning. This forms a 5D sampling of the plenoptic function, 3D for camera position and 2D for cylindrical image. In order to generate new samples of the plenoptic function, the cylindrical images are warped to the viewing position based on their epipolar relationship and some visibility tests.

Other assumptions have been made in order to simplify the plenoptic function. Relevant work includes *Light Field* [Levoy and Hanrahan, 1996] and *Lumigraph* [Gortler et al., 1996] approaches. Both of these methods ignored wavelength and time dimensions. Additionally, they assumed that the radiance does not change along a line in free space. The 5D plenoptic function can then be simplified into a 4D plenoptic function, *i.e*. the *lightfield*:

$$\mathcal{P} \simeq P_4 = P(u, v, s, t) \tag{4.3}$$

where (u,v) and (s,t) parametrize two parallel planes of the ray's bounding box. These planes are then discretized so that a finite number of light rays are recorded in a database (see Figure 4.2). In order to generate a new view, each ray passing from the object of interest to the position of the new view (*e.g.* camera center) intersects these two planes. Points of intersection are denoted as (s_0,t_0) and (u_0,v_0). If such a sample already exists in the database, the color value is applied, if not, the color value is interpolated using the nearest samples stored in the database. The major drawback of light field rendering approach is that a huge amount of images is required to represent the scene which is not efficient for transmission purposes. The Lumigraph [Gortler et al., 1996] approach overcomes this drawback and tends to reconstruct an approximate geometry of the scene using an octree algorithm in order to facilitate the rendering with smaller sets of images.

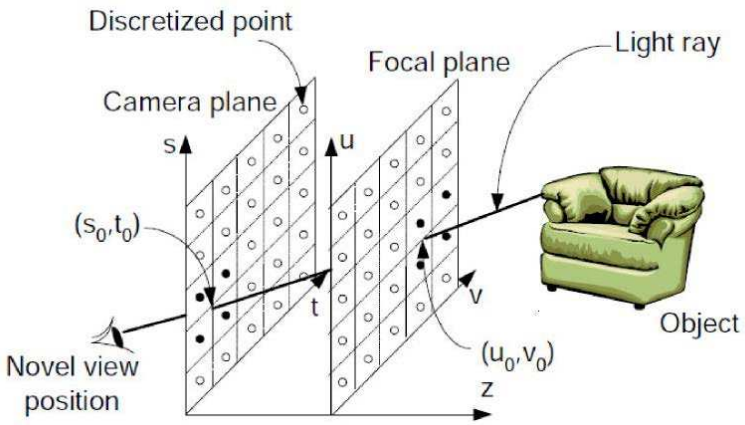


Figure 4.2: Representation of a light field (image from [Shum and Kang, 2000]).

An approach similar to the Lightfield/ Lumigraph techniques was proposed by [Fujii et al., 2007]. The authors propose a scene representation called ray-space. They propose an acquisition system that consists of a high speed camera, a double parabolic mirror and a galvanometer mirror reflecting the object image to the high speed camera. Ray space representation requires a tremendous amount of data, the authors propose to use the multi-view video coding MPEG-4/MVC standard for efficient data compression and storage.

4.2.2 Depth image based representation

In the last section we saw that representing a scene using only the captured views requires too much information to transmit. Depth based representations aim at reducing time complexity during novel view rendering while preserving image quality. In this section, depth aided methods for scene representation are presented.

4.2.2.1 Video plus Depth (2D+Z)

The simplest approach for representing a scene with minimal depth information consists in using one view and per time instant the captured image jointly with a depth map. A depth map is an array that contains the depth information related

to each pixel of the captured image (see Figure 4.3). Such depth maps can be either captured by a depth sensor or estimated using state of the art methods of disparity estimation[1] [Scharstein and Szeliski, 2002]. This approach is referred to as 2D+Z representation and was first used in the European ATTEST project [Fehn et al., 2002]. Besides its simplicity, the 2D+Z representation is also efficient in terms of coding cost. Actually, encoding a gray level depth map is less expansive than an RGB image. However 2D+Z representation has several limitations especially related to novel view synthesis. Using a unique depth map to render arbitrary views is not optimal, since the only depth map available can not capture the complete scene geometry. More precisely, some regions of the scene may be occluded in the view where the depth map is acquired. Only views not located far away from the reference view can be rendered with few errors. In other words, novel view synthesis is efficient for cameras with small baseline between the rendered and the original view (*e.g.* stereo videos).

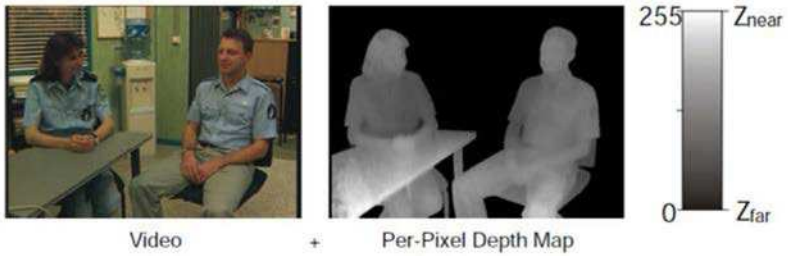


Figure 4.3: 2D+Z representation.

4.2.2.2 Multi-view Video plus Depth (MVD)

For 3DTV and FTV application, 2D+Z based representation is not sufficient to synthesize arbitrary views. Multi-view Video plus Depth [Merkle et al., 2007] representation has been proposed in order to handle this limitation. In this scheme, a larger number of viewpoints is captured and a depth image corresponding to each view is available 4.4. Uncovered regions of a scene by one view could be covered by neighboring views. Efficient virtual view synthesis can therefore be performed for any view between the cameras. The major drawback of such representation is that redundancy in terms of depth and texture information is increased. Efficient multi-view video compression algorithms are required. Compression of multi-view video plus depth representation has been widely investigated. In [Merkle et al., 2007], the authors show that coding depth videos is closely related to the quality of virtual view synthesis. For depth data coded at low bitrates, several artifacts around depth discontinuities (*e.g.* at object boundaries) are noticeable. To this end, Morvan *et al.* [Morvan, 2009] developed an approach based on platelets, which are piece-wise linear functions, for coding depth data. Daribo [Daribo, 2009] proposed a method that consists in using a common vector field for coding texture and depth videos and propose a wavelet decomposition for coding depth maps.

[1] See http://vision.middlebury.edu/stereo/ for existing algorithms and datasets used for disparity estimation

Figure 4.4: MVD representation, each view is composed of a texture image and a depth map.

4.2.2.3 Layered Depth Videos (LDV)

Layered Depth Videos originally introduced by He *et al*. [He et al., 1998]. This representation aims at reducing redundancies present in MVD representation. The basic idea behind LDV is to provide one full or central view as a main layer and additional enhancement layers that include residual texture and depth data intended to represent the neighboring views. Figure 4.5 illustrates this principle. The major drawback of LDV representation is that the construction of such representation is too complex to achieve a rendering quality similar to the MVD representation.

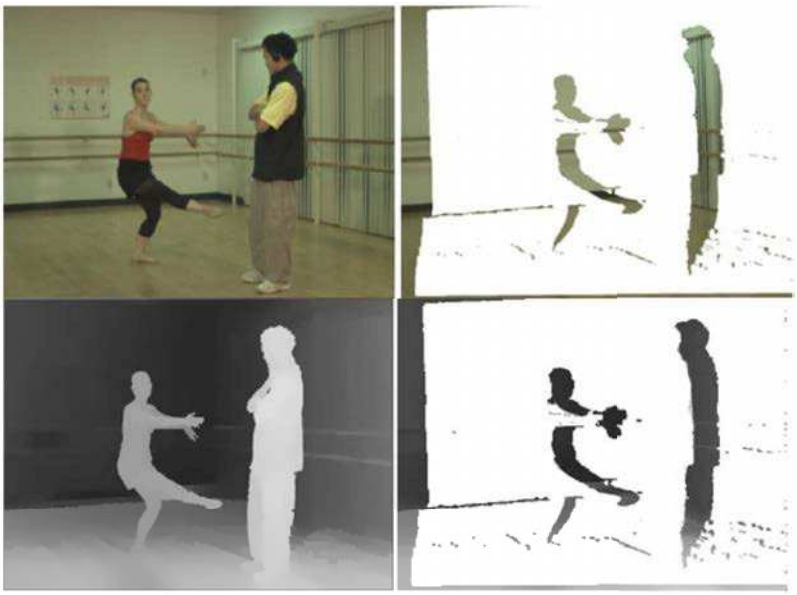


Figure 4.5: LDV representation (image from [Muller et al., 2010]).

4.3 Geometry-based representations

In the last section we presented methods that use a set of depth maps to represent the scene's geometry. Other methods use explicit 3D models to describe such geometry. A 3D model can be either a point cloud or a mesh.

4.3.1 Point based representation

Instead of using images for depth information storage, point-based representations use points clouds. The use of points as geometric primitives was coined by Levoy *et al*. [Levoy and Whitted, 1985], they showed that any continuous three dimensional surface can be converted into a point set, this surface can then be rendered from this point set using some perturbations. A perturbation is defined as any operation that changes the points' attributes (*i.e*. position or color). Following this pioneering work, QSplat [Rusinkiewicz and Levoy, 2000] was introduced as a point rendering system designed to interactively render large data sets acquired by scanning devices. Splatting consists in rendering the surface from a point set followed by texture filtering in order to reduce aliasing artifacts due to point samples. The most commonly used filter is Elliptical Weighted Average filter (EWA) [Pfister et al., 2000] and [Zwicker et al., 2002]. An application for 3D video was proposed by Waschbüsh *et al*. in [Waschbüsch et al., 2007a]. They proposed an acquisition system based on multiple 3D video bricks. A brick contains a projector, two gray-scale cameras and a high-resolution color camera (see Figure 4.6). Depth estimation is enhanced using structured light patterns and high-resolution depth maps are then obtained using the acquired pattern images. The surface samples corresponding to the depth values are merged into a view-independent, point-based 3D data structure. The proposed representation allows an acceptable rendering quality (see Figure 4.6). The advantage of point based representations is that they do not require any connectivity information to be stored. Moreover, the use of point samples make their insertion, deletion and repositioning very practical especially for dynamic scenes with frequent changes of scene's geometry. However compression of point clouds is a recent research area and has not been yet widely investigated. Besides, the splatting step leads to noticeable artifacts such as holes or isolated pixels (see Figure 4.6).

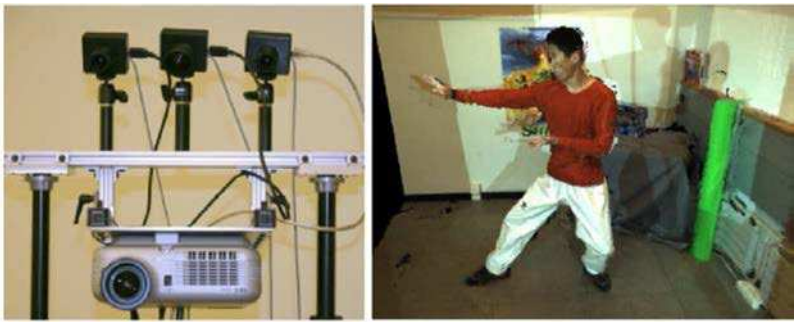


Figure 4.6: Left: the proposed 3D video brick with camera and projector. Right: rendering of the 3D video from new viewpoint [Waschbüsch et al., 2007a].

4.3.2 Mesh based representation

In order to alleviate the visual problems due to splatting, mesh based approaches add the connectivity information to the point cloud. More precisdely, a mesh $\mathcal{M}$ is a couple $(\mathcal{C},\mathcal{G})$, that represents a surface by the connectivity information $\mathcal{C}$, *i.e*. the set of triangles, edges and vertices and the geometry information $\mathcal{G}$, *i.e*. 3D vertices coordinates. Using meshes for scene representation has become

very popular in the last few decades thanks to the increasing hardware support. Moreover mesh based methods provide a more compact representation of the scene than depth maps. Mesh based scene representation methods can be classified into two categories: volumetric methods and non volumetric methods. In this section we will focus only on non-volumetric methods. Volumetric methods will be studied in detail in chapter 5. Among existing non volumetric techniques, some methods propose to exploit depth maps as input and produce a surface mesh. This can be done by fusing neighboring depth maps that minimizes visibility constraints' violation [Merrell et al., 2007]. This fusion could also be patch-based [Furukawa and Ponce, 2010] (see also Figure 4.8) or using purely geometric methods by triangulating each depth map, aligning the set of meshes using Iterative Closest Point (ICP) [Besl and McKay, 1992] algorithm and performing a mesh zippering algorithm [Turk and Levoy, 1994] (see Figure 4.7) in order to merge the overlapping areas. If the cameras poses are not known and there's no correspondence information between the acquired views, Structure from Motion algorithms may be used in order to recover the cameras's poses and build the underlying geometry [Snavely et al., 2010] and [Agarwal et al., 2009].

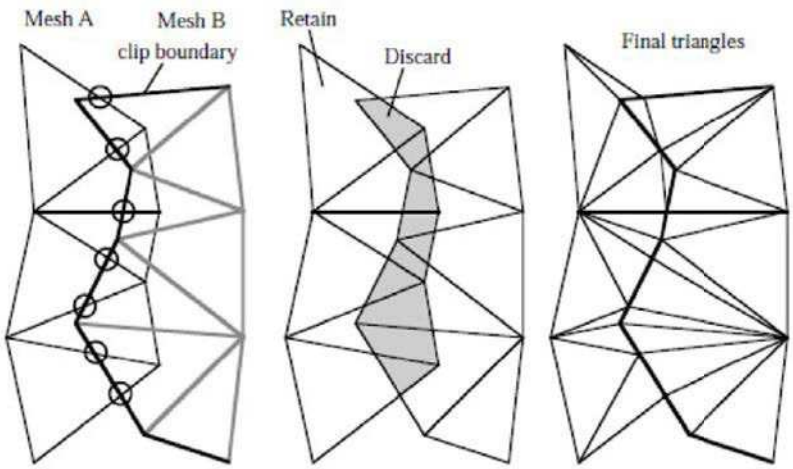


Figure 4.7: Mesh A is clipped against the boundary of mesh B. Circles (left) show intersection between edges of A and B's boundary. Portions of triangles from A are discarded (middle) and then both meshes incorporate the points of intersection (right) [Turk and Levoy, 1994].

Unlike, image based approaches, mesh based approaches provide a single model representing the scene geometry. However, acquiring such 3D models is not straightforward and could be time-consuming.

4.4 Hybrid representations

Having reviewed geometry based and image based representations, in this section we will present some methods that use both image and geometry to represent a scene. The idea is to take advantage of image based approaches to increase the visual quality during rendering and geometry based approaches in order to build compact representations.

Figure 4.8: From left to right: A sample input image, detected features, reconstructed patches after the initial matching, final patches after expansion and filtering, and the mesh model [Furukawa and Ponce, 2010].

4.4.1 Billboard Clouds

Billboards clouds were introduced by [Décoret et al., 2003]. In this scheme, 3D models intended to represent the scene are simplified onto a set of textured planes *i.e. billboard clouds* with independent size, orientation and texture resolution. 3D models are extremely simplified using an optimization approach to build the billboard cloud.

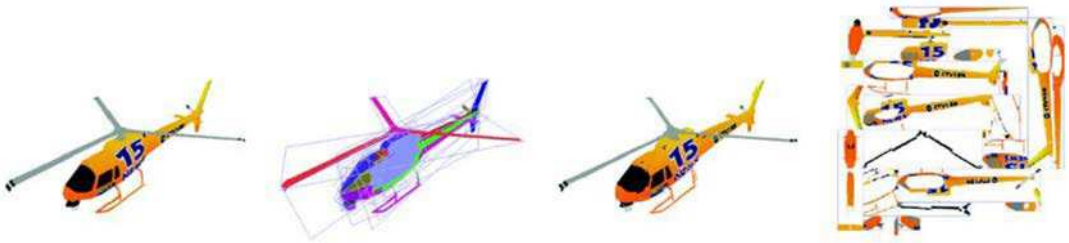


Figure 4.9: Example of a billboard cloud. From left to right: Original model (5,138 polygons), false-color rendering using one color per billboard to show the faces that were grouped, View of the (automatically generated) 32 textured billboards, the billboards side by side (image from [Décoret et al., 2003]).

4.4.2 3D Video Billboard Clouds

In this work [Waschbüsch et al., 2005], Waschbüsch *et al.* combine the generality of geometry-based representation with the regularization properties of image based representation. They extend the billboard clouds representation developed by [Décoret et al., 2003] and displacement-mapped billboard clouds [Mantler et al., 2007] and propose a new 3D video representation that consists in an arbitrarily placed and oriented texture-mapped plane approximating the geometry of the real object. They associate to this rectangle three textures (see Figure 4.10): a displacement map for geometric details, a color map modeling the surface appearance and an alpha map holding a smooth alpha matte representing the object's boundary and used for seamless blending with the background of the scene (See Figure 4.10).

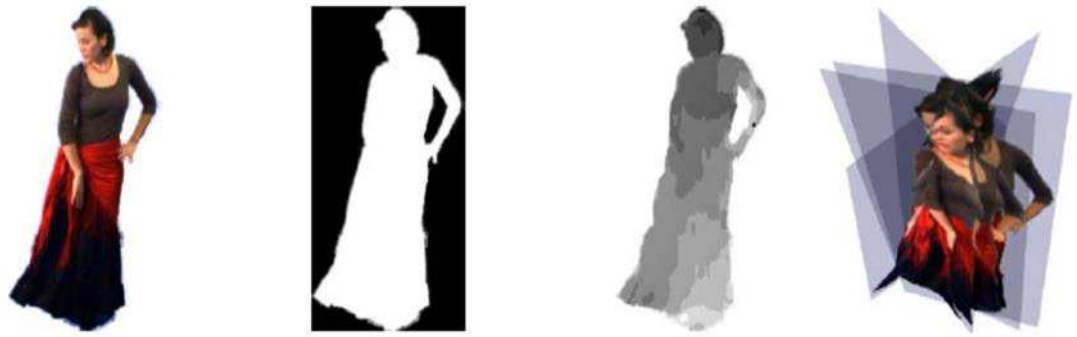


Figure 4.10: 3D Video Billboard Clouds. From left to right: texture image, alpha matte, displacement map, composition of planes from multiple input views to a billboard cloud [Waschbüsch et al., 2005].

4.4.3 Microfacet billboarding

While existing geometry modeling techniques assign a fixed facet to the geometric model. The idea behind microfacet billboarding is to model the geometry with facets that varies over the views. Based on this idea, Yamazaki *et al*. in [Yamazaki et al., 2002] proposed a new modeling method that uses view-dependent microfacets with view dependent textures. The facets approximate the geometry of the object and are perpendicular to the viewing direction. Depending on the viewpoint, the texture of each facet is selected from the most suitable texture images according to the viewpoint.

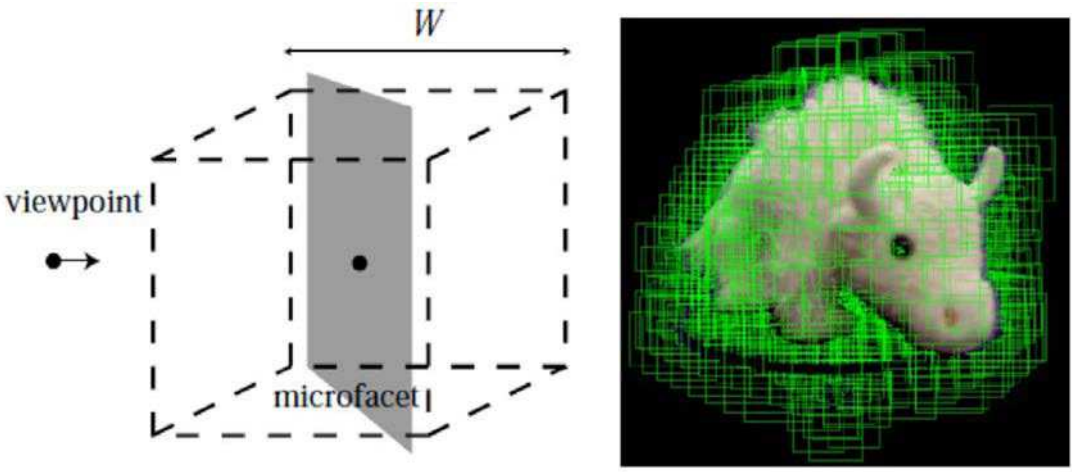


Figure 4.11: Microfacet billboarding. Left: a microfacet is a slice that intersects the center of the voxel and is vertical to the viewing direction. Right: microfacet billboarding result.

Several artifacts occur during rendering depending on the size, shape and orientation of the microfacets. An analysis of such artifacts has led to determining the proper sampling interval of microfacets and the optimal orientation of microfacets.

4.4.4 Polygon Soup representation

A polygon soup representation was proposed in [Colleu et al., 2010]. The authors proposed a new representation that considers MVD data as input and builds a set of disconnected 3D polygons defined with 2D+Z data (see Figure 4.12). Polygon soup generation is performed by a quad tree decomposition of the input depth maps. Polygons are then selectively eliminated in order to remove inter-view redundancies. The polygon soup representation allows a good trade off between rendering quality and data compactness. However, in polygon soup representation scheme, the number of quads depends on the geometry of the scene, hence more quads are required to represent very complex scenes.

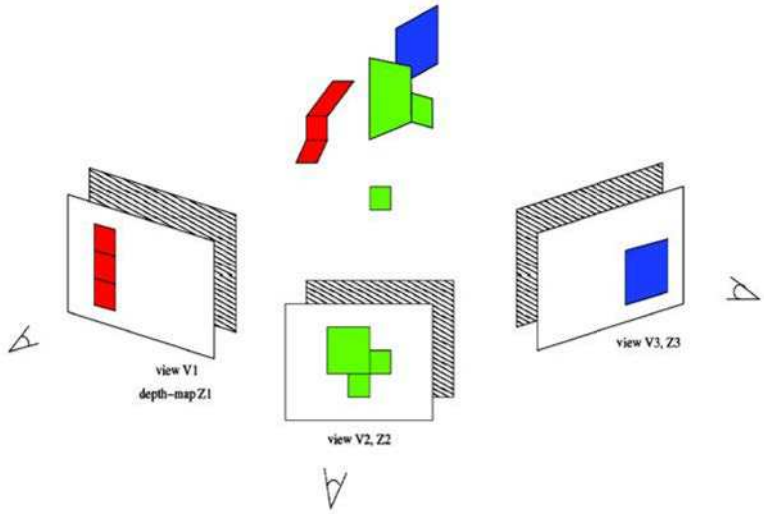


Figure 4.12: Polygon soup representation: a set of 3D polygons. Each 3D polygon is defined by a 2D polygon in one of the acquired views, and by the depth information for each corner of the polygon (image from [Colleu et al., 2010]).

Hybrid representations has the advantage of simplifying the scene geometry by few polygons. However, besides the complexity of polygons extraction, a complex scene would require a considerable amount of polygons which is not optimal for transmission purposes.

4.5 Conclusion

In this chapter, we presented existing techniques for scene representation. These methods can be either image based, geometry based or hybrid methods relying on both image and geometry. In image based schemes, photo-realism is preserved. However, efficient novel view synthesis can be performed with several depth maps (*e.g.* MVD representation) which increases data redundancy. Geometry based representations overcome this limitation and gather depth information into a unique and compact representation of the scene geometry. Geometry based representations provide a better alternative for representing occlusion areas. Nevertheless, photo-realism may be lost and several visual artifacts can be produced (*e.g.* point based representation). Hybrid representations seem to be a

trade-off between photo-realism and high quality rendering. However, the lack of explicit connectivity information make hybrid representation inadequate for transmission issues especially for complex scenes.

Part II

Volumetric Scene Reconstruction

Chapter 5

Volumetric Scene Reconstruction: a State of the Art

Contents

5.1 Introduction

In the last chapter we reviewed existing scene representation strategies. Mesh based representation are turned out to be more efficient for capturing the scene geometry and for efficient novel view synthesis. One famous strategy in order to generate meshes for scene representation is to use volumetric methods. Indeed, volumetric methods are known to be robust in presence of outliers. To this end, this chapter surveys existing methods for volumetric methods that generate a mesh intended to be used a scene representation. Other surveys can be found in [Slabaugh et al., 2001a, Dyer, 2001, Slabaugh et al., 2004]. This chapter starts with shape from silhouette techniques which are presented in 5.2, we then survey methods that use input texture images in section 5.3. Finally depth based reconstructions are presented in section 5.4.

5.2 Shape from Silhouette

In this section, we will describe existing methods for constructing a polygonal model representing an object of interest in the scene using a set of silhouette images. The purpose using these methods is to recover an object of interest in the scene. The methods described in this section assume that a foreground object can be segmented from the background. A silhouette image can be defined as:

Definition 5.2.1 (Silhouette image) *A silhouette image is a binary image whose value at a point indicates whether a ray from the camera center to this point intersects the object of interest or not.*

An example of a silhouette image is shown in Figure 5.1. Earlier approaches for 3D model reconstruction tend to use a set of silhouette images each one being generated from a given viewpoint. The reconstructed 3D model is an approximation of the so-called the *visual hull*s [Laurentini, 1994]. The visual hull can be defined as:

Definition 5.2.2 (Visual Hull) *The visual hull is the maximal volume that is consistent with a set of silhouette images.*

The first approach for visual hull construction using the set of silhouettes was proposed by laurentini [Laurentini, 1994]. In this scheme, the visual hull is recovered using the intersection of viewing cones, each viewing cone is described using silhouette images.

5.2.1 Voxels, Octrees and Tetrahedrons based Reconstruction:

Several algorithms have been proposed in the literature for volumetric construction of the visual hull using a set of silhouette images. Typically, in such algorithms, one first starts by defining the volume of interest that encloses the scene, the volume being subdivided into cubical elements called voxels. The task is then to assign to each voxel a binary label (Opaque/ Transparent) using the silhouette

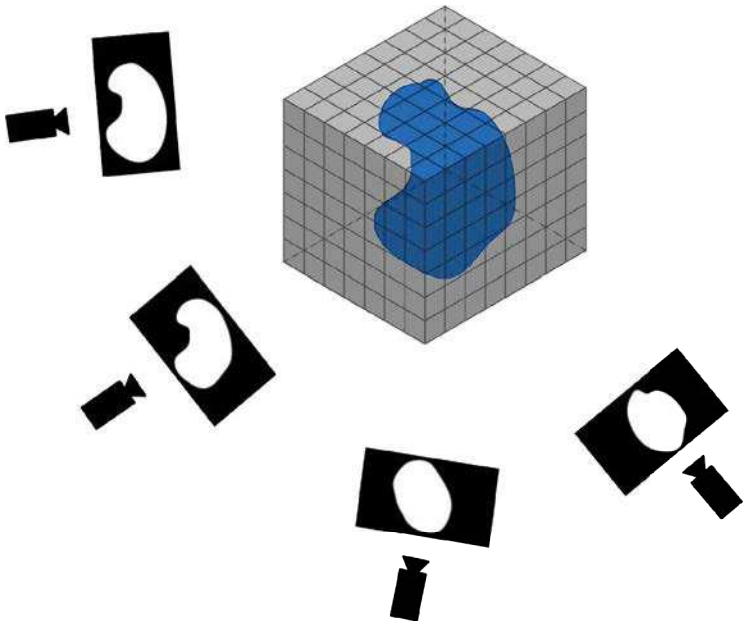


Figure 5.1: Volumetric reconstruction using shape from silhouette technique on a regular grid.

information. Some methods tend to determine the visual hull by back-projecting the silhouettes [Potmesil, 1987] (see also Figure 5.1), or by projecting the voxel and decide whether this projection falls into the set of images [Szeliski, 1993].

Instead of using a regular subdivision, an octree data structure may be used for a better space traversal as it was proposed in [Szeliski, 1993]. A formulation of such algorithm is described in Algorithm 9:

Other methods propose to use tetrahedrons instead of voxels. In the approach described in [Boyer et al., 2003], tetrahedrons are built using Delaunay triangulation of points sampled from the visual hull surface. Delaunay triangulation of this sample points outputs a set of tetrahedrons. From these tetrahedrons, those outside the visual hull are eliminated. In order to remove such tetrahedrons, the center of each tetrahedron is projected onto the set of input views and is localized whether it is inside or outside the silhouettes. The resulting set of tetrahedrons may contain some isolated tetrahedrons which may lead to a non-manifold surface. A reconstruction example on the "knots" data-set is shown in Figure 5.2.

Delaunay triangulation was also used in [Aganj et al., 2007] to build a spatio-temporal visual hull. First a static visual hull is computed from silhouettes images using an incremental Delaunay triangulation [Boissonnat and Oudot, 2006]. A first point set is initialized to a small set of points lying on the boundary of the visual hull. At each iteration, a new point is inserted and the Delaunay triangulation is updated. An update is valid if the circumcenters of the modified tetrahedron are projected in all images and checked whether all these projections lie inside the silhouettes. This same incremental Delaunay triangulation scheme

Algorithm 9: Visual hull construction from silhouette images using an octree.

Data: Silhouette images.
Result: Visual Hull VH.

```
1  Define a volume V that encloses the scene.
2  Subdivide V into voxels.
   // Voxels traversal
3  for each voxel v in V do
4      Compute P_v the projection of v onto the set of captured images.
5      if P_v does not intersect at least one silhouette then
6          mark v as transparent.
7      else if P_v intersects only silhouette pixels for each view then
8          mark v as opaque.
9      else
           // This voxel intersects with background pixels
10         subdivide v into 4 sub-voxels and reprocess each sub-voxel.
```

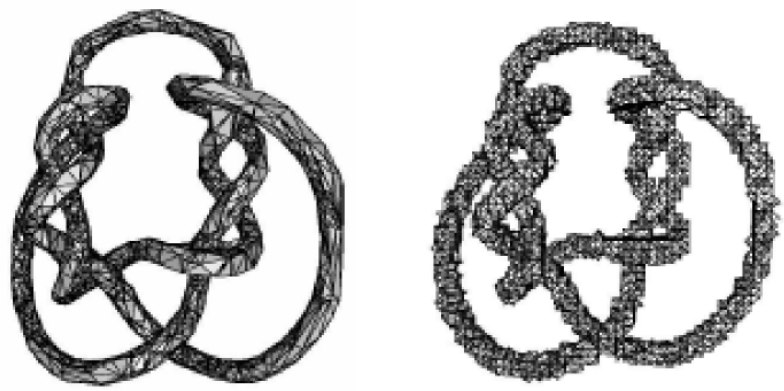


Figure 5.2: Reconstruction example on the knots dataset using tetrahedrons (left) versus regular grid (right) (image from [Boyer et al., 2003]).

was also considered by the authors in order to build a spatio-temporal visual hull. Shape from silhouette was also formulated using a bayesian framework [Franco and Boyer, 2005] or using an energy minimization problem [Hernández Esteban and Schmitt, 2004]. Finally, a recent work was proposed to build the visual hull using a new 3D primitive called Convex Bricks (CB) [Chari et al., 2012]. Their algorithm builds a polyhedral visual hull by combination of CBs. Each CB can have an arbitrary shape but depends on the given silhouette.

However all shape construction using silhouette information has several limitations. All these methods assume that foreground objects in the input images can be extracted from the background. Each foreground object should be reconstructed independently from the set of silhouette images. Therefore, the entire scene can be described by the each reconstructed object and the set of redundant background images. Another drawback of shape-from-silhouette approaches is that not all concavities present in the silhouette images can be represented. Actually, complex objects may contain some holes that can not be detected as holes in

the silhouettes. Other errors during reconstruction may be due to silhouette extraction which is not an easy task [Pardas et al.,], especially in presence of noise, since silhouette extraction is related to image segmentation problem. Some methods propose to incorporate other photo-metric cues to increase the accuracy of the visual hull [Sinha and Pollefeys, 2005].

5.3 Shape from Photo consistency

In the last section we saw that reconstruction based on only silhouette cues is not sufficient for accurate reconstruction. In order to improve the quality of the reconstructed mesh, additional information could be considered. Several researchers propose to build a 3D model by incorporating photometric information during reconstruction. In such schemes, we are not limited to recovering an object of interest as it is the case in Shape from silhouette methods 5.2. Besides, a higher quality 3D model is built by checking the consistency between the synthesized images and the original ones. In other words, the 3D model is built by recovering the set of points that are photo-consistent with the input images. More formally:

Definition 5.3.1 (photo-consistent point) *A point on a 3D scene is said to be photo-consistent with a set of images, if for each image where this point is visible, the radiance of that point is equal to the intensity at the corresponding image pixel [Dyer, 2001].*

Typically, as in volumetric reconstruction using silhouette images, the volume to be processed is initialized to a volume that surrounds the scene of interest, this volume is then subdivided into voxels. Each voxel is then projected on the set of images where it is visible and tested whether it is photo-consistent or not. A binary labelling strategy is then used to classify voxels. A voxel is labelled opaque if the voxel center is photo-consistent, otherwise it is labelled transparent. Figure 5.3 illustrates an example of photo-consistent and non-photo-consistent voxel. In 5.3(a) a photo-consistent voxel is shown. Both left anf right cameras see blue. In contrast in 5.3(b), The left camera sees red, whilst the right camera sees green.

(a) Photo-consistent voxel. (b) Non-photo-consistent voxel.

Figure 5.3: Photo-consistency principle.

It follows from definition 5.3.1 that two main steps are required in photo-consistency based reconstruction: visibility determination and consistency computation with respect to each voxel. The next two paragraphs survey different

methods for visibility and consistency computations under some constraints (*e.g.* related to cameras placement, lambertian surfaces, *etc.*). We will then present some extensions and improvements of these methods in the third paragraph.

5.3.1 Efficient visibility determination

Computing visibility of each voxel with respect to each view may be time consuming. Efficient algorithms are then required in order to simplify visibility computation. Seitz and Dyer in their voxel coloring scheme [Seitz and Dyer, 1999] introduced the *ordinal visibility constraint* that constrains the viewing cameras to be placed so that all voxels can be visited in a single scan in near to far order. This constraint can be satisfied by placing all the cameras on one side of the scene and scanning voxels in planes that are successively further from the cameras. Thus, the photo-consistency of all voxels that may occlude a given voxel might be determined before the photo-consistency of the current voxel is computed. In contrast, the space carving algorithm introduced in [Kutulakos and Seitz, 2000] allows arbitrary camera placement, but uses multiple scans using a plane sweeping along the positive and negative directions of each of the three axes. In each scan, only cameras behind the current sweeping plane are used to determine the consistency of the voxel. This means that only a set of the available images is used on each pass to determine the consistency. This is referred to as the monoticity constraint [Kutulakos and Seitz, 2000] which states that if a voxel is not consistent with a subset of views then it is also not consistent with the set of all available views. In order to improve the efficiency of voxels traversal, the *generalized voxel coloring* [Culbertson et al., 2000] scheme maintains two data structures: an Item Buffer (IB) and Layered Depth Images (LDI). An IB stores for each pixel in an image the surface voxels that are visible from this pixel. This allows efficient memory usage. Whereas an LDI records for every pixel a depth-sorted list of voxels that are projected to this pixel which eliminates unnecessary photo-consistency computation.

5.3.2 Photo-consistency function

Once visibility for each voxel with respect to each view is determined, another challenging task is determining the consistency of a voxel. Assuming that cameras projection matrices are known, photo-consistency allows to know if a 3D point is consistent with the input images. Definition 5.3.1 states that the irradiance of 3D point must be the same in all views. The definition of the "same" has a key role during voxel labelling process. A straightforward approach consists in thresholding the variance of the projected colors [Kutulakos and Seitz, 2000]. The threshold is supplied by the user. However, this is valid only for lighting models which are locally computable (*e.g.* lambertian assumption). Moreover, this is could be problematic when projecting a voxel to an area larger than a pixel. In this case, the information from all the pixels, not only the center pixel, should be considered. Broadhurst *et al.* [Broadhurst et al., 2001] proposed to use the F-statistic locally to choose an appropriate threshold for each voxel. This statistical approach turns out to be more efficient than the global threshold suggested in [Kutulakos and Seitz, 2000].

5.3.3 Improvements of Photo-consistency based methods

In the last section we presented some methods that may increase the complexity of the processing and therefore not suited for real time applications. Pan *et al*. [Pan et al., 2009] address this issue and develop ProFORMA[1] (Probabilistic Feature-based On-line Rapid Model Acquisition). In this scheme, the authors generate a 3D model on line as the input sequence is collected. A model is rapidly generated using a 3D Delaunay triangulation and updated with future frames, tetrahedrons are carved based on visibility of some keypoints as in Labatut *et al*. [Labatut et al., 2007] and a probabilistic voting scheme. But ProForma is only capable of reconstructing isolated objects. Lovi *et al*. [Lovi et al., 2010] extended the idea proposed in ProForma to handle more complex scenes. Other methods formulate photo-consistency as energy minimization [Labatut et al., 2007], a global minimum is obtained using volumetric graph cuts [Vogiatzis et al., 2005]. In order to address the problem of large scale modeling, Slabaugh *et al*. proposed volume warping [Slabaugh et al., 2001b] that aims to represent an infinite volume with small number of voxels. Finally other methods proposed to address the issue of non-lambertian surfaces [Yang et al., 2003] and [Treuille et al., 2004].

In this section, we presented shape from photo-consistency methods that use photometric constraints to carve voxels from the input volume surrounding the scene. The problem of this approach is that image data is almost always ambiguous especially in the case of non-lambertian scenes or texturless areas. Therefore, wrong decisions could be made during voxel carving that it is not possible to undo afterwards. Besides, explicit visibility determination may significantly increase the complexity of the algorithm which is not suited for real time applications.

5.4 Reconstruction Using Depth/Range Data

Instead of using input texture images, another alternative is to use depth data, acquired by either a range scanner or estimated directly using stereo matching techniques [Scharstein and Szeliski, 2002]. Acquiring a surface from such depth data can be performed by fitting a surface mesh on the sample points or by triangulation each depth/range image independently and merge the resulting surfaces into a unique surface the capture geometric details of the scene of interest.

5.4.1 Shape from Point Samples

The major advantage of shape reconstruction from point samples is that such algorithms do not make any assumption about the points' connectivity. In absence of depth/range images to provide connectivity information, these algorithms are best suited to recover a 3D model.

5.4.1.1 Alpha shapes

Edlesbrunner *et al*. [Edelsbrunner and Mücke, 1994] introduced alpha-shapes which is a hierarchy of shapes controlled by the radius of balls of radius α. A

[1] See also http://mi.eng.cam.ac.uk/~qp202/ for video demonstrations.

first approximation of the true shape is obtained by 3D Delaunay triangulation of the point set. This results in a set of connected tetrahedrons. As α changes new tetrahedrons may be added or deleted as intersections with the constructed Delaunay triangulation and the balls appear or disappear (See Figure 5.4).

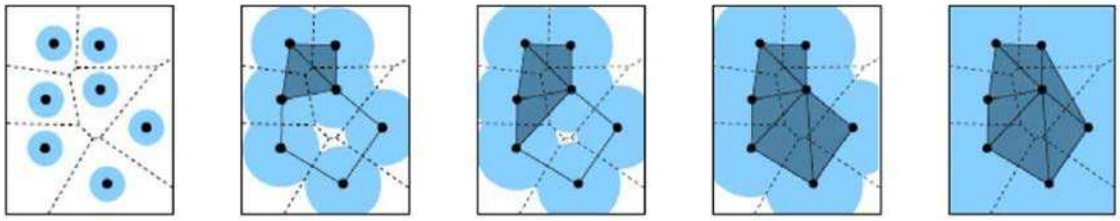


Figure 5.4: Construction of alpha shapes with increasing ball radius (image from [Dey et al., 2003]).

The problem of this approach is that the so-called radius α is determined by the user. Inappropriate values of α lead to coarse estimation of the true shape. In order to avoid this issue, Salman *et al.* [Salman et al., 2010] build a triangle soup from the initial depth maps. The triangle soup is then filtered using visibility and photometric constraints. A surface mesh is generated from the triangle soup using restricted Delaunay triangulation [Pons and Boissonnat, 2007].

5.4.1.2 Poisson Surface Reconstruction

The approach considered here is to build the desired surface S from a set of oriented points $\vec{V}$ by approximating an indicator function χ. This indicator function is defined as 1 inside S and 0 outside (see Figure 5.5). More precisely, the gradient $\nabla\chi$ of the indicator function is a vector field whose value is 0 almost everywhere, since χ is constant almost everywhere, except at points near the surface where it is equal to the inward surface normal. The oriented point set $\vec{V}$ can be regarded as samples of $\nabla\chi$.

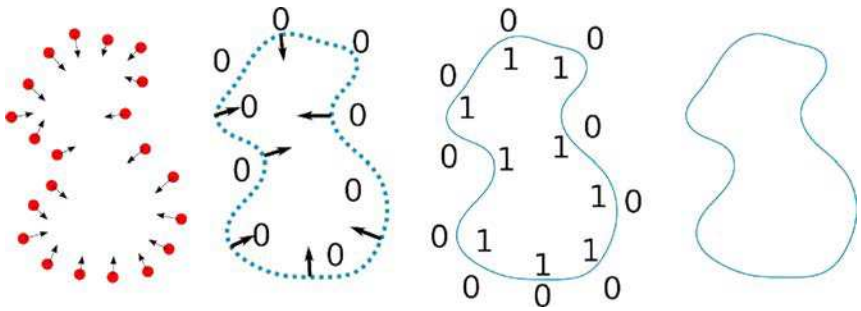


Figure 5.5: Poisson surface reconstruction. From left to right: an oriented point set $\vec{V}$, gradient of the indicator function $\nabla\chi$, indicator function χ, final surface S.

The problem is then reduced to inverting the gradient operator *i.e.* computing χ from the following equation:

$$\nabla\chi = \vec{V}$$

Applying the divergence operator will transform the last equation into a Poisson problem *i.e.* computing the scalar function χ whose Laplacian $\Delta\chi$ (divergence of the gradient) equals the divergence of the vector field $\vec{V}$:

$$\Delta\chi = \nabla.\nabla\chi = \nabla.\vec{V} \tag{5.1}$$

Poisson based solutions are known to be resilient in presence of imperfect data. Reconstructing a surface using Poisson solution creates a smooth surface that can approximate noisy data. Moreover, unlike radial basis function schemes, the solution of the Poisson based approach reduces to a well conditioned linear system. The problem of Poisson surface reconstruction is that it requires an oriented point set (*i.e.* point set with normals). However, normal estimation from point cloud is also a challenging task in surface reconstruction from point clouds literature.

5.4.2 Shape from depth maps/range images

In contrast to reconstruction using point samples that we have presented in section 5.4.1, reconstruction based on depth/range images provides us the connectivity information. We can obviously always fall into point based reconstruction by back projecting image pixels to 3D World coordinate system, producing therefore a set of point samples.

Reconstruction using distance function

The basic idea in reconstruction based distance function is to represent the input range/depth images as samples of a continuous function and strive to recover the continuous function on each voxel. The first method was proposed in [Hilton et al., 1996] and was designed to integrate a set of range images using implicit functions. This algorithm was the first one that performs integration of range data in 3D space. They developed a method that uses weighted signed distance functions for merging range images acquired using range scanners. However the proposed method does not address some issues such as sensor uncertainty and potential hole filling in the final surface. In contrast, Curless *et al.* [Curless and Levoy, 1996] addressed these issues. The algorithm proposes to build a continuous implicit function $D(x)$. From each input range image $I_i, i \in \{1,...,n\}$, they derive a signed distance function $d_1(x),...,d_n(x)$ from the sensor to a given voxle and weight functions $w_1(x),...,w_n(x)$ (see also Figure 5.6). The proposed combining rule provides for each point x in space a cumulative signed distance function $D(x)$ and a cumulative weight $W(x)$. These functions are then represented on a regular grid of voxels. They finally extract an isosurface that corresponds to $D(x) = 0$. Figure 5.6 shows an example of combining signed distances in the case of two range surfaces sampled from the same direction.

$$D(x) = \frac{\sum w_i(x) d_i(x)}{\sum w_i(x)} \tag{5.2}$$

$$W(x) = \sum w_i(x) \tag{5.3}$$

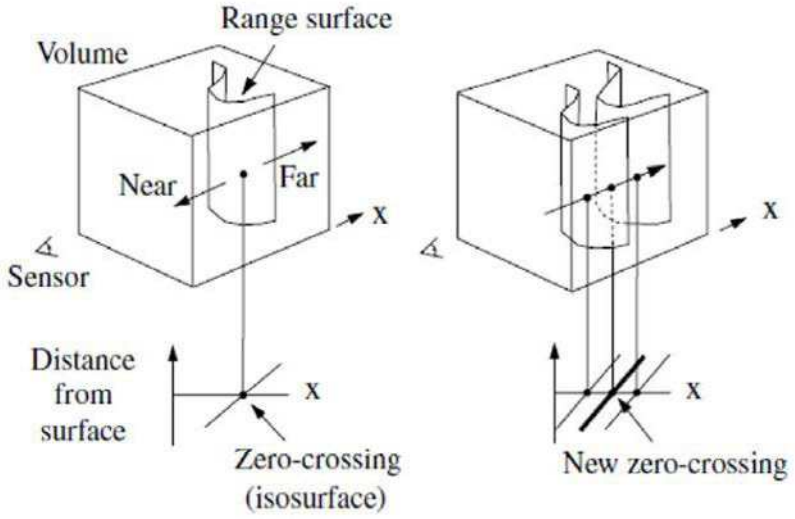


Figure 5.6: Unweighted signed distance functions in 3D. Left: A range sensor looking down the x-axis observes a range image, shown here as a range surface. Following one line of sight down the x-axis, a signed distance function as shown. The zero crossing of this function is a point on the range surface. Right: The range sensor repeats the measurement, but noise in the range sensing process results in a slightly different range surface. In general, the second surface would overlap the first, but we have shown it as an offset from the first surface for purposes of illustration. Following the same line of sight as before, we obtain another signed distance function. By summing these functions, a cumulative function can be obtained with a new zero crossing positioned midway between the original range measurements. See equations 5.2 and 5.3. Image from [Curless and Levoy, 1996]

Rather than using a regular grid of voxels, the authors in [Fuhrmann and Goesele, 2011] extend the work developped by [Curless and Levoy, 1996] and proposed a *hierarchical distance function* that is discretized using an octree. The considered depth maps may be at different scales *i.e.* the considered depth maps may represent the same regions but at different resolution. Thanks to the use of the octree, the final surface may consistently represent low resolutions as well as fine details contained in high resolution depth maps.

KinectFusion [Newcombe et al., 2011] and [Zach et al., 2007] use a modified version of the signed distance function called *truncated distance function*. The original distance function [Curless and Levoy, 1996] is fitted into the interval $[-1,1]$ which reduces memory consumption. In [Zach et al., 2007], they formulate the problem of distance function generation as an energy minimization problem. This energy is expressed as sum of two terms. The first term is a fidelity term expressed using L_1 norm since this norm is supposed to be robust than L_2 in presence of outliers, the second term employs a total variation regularization for optimal surface reconstruction. Although this method produces satisfying results, it is very expansive in terms of memory requirement and computation time.

5.5 Conclusion

In this chapter we have reviewed several methods to reconstruct a 3D surface using a volumetric framework. Depending on the input data considered, the quality of the constructed surface could be different. When using only silhouette information for example, besides its restriction to objects only, a lower quality surface can be obtained due to the corrupted silhouette information. The use of depth data may have several advantages: first, we can take advantage of a vast literature and methods in stereo correspondence techniques. Second, we can bypass delicate visibility determination and photo-consistency computation for each voxel which are the main tasks in photo-consistency based methods.

Chapter 6

Depth Maps based Space Carving for Volumetric Fusion

Contents

6.1 Introduction

Having surveyed existing volumetric methods for scene reconstruction. In this chapter, we will introduce a new volumetric framework for surface reconstruction from the acquired views. As in state-of the art methods studied in 5.4.2, we formulate the problem of surface reconstruction as a depth-map-fusion problem. The proposed algorithm outputs a surface mesh from a set of depth maps. Each depth map is regarded as a single mesh. Each mesh is then considered as input to our volumetric framework. An overview of the proposed algorithm is sketched in Figure 6.1. These steps can be summarized in the following:

- Depth map conversion into 3D meshes (section 6.2): each input depth map is converted into a 3D triangular mesh.
- Bounding grid computation (section 6.3): the bounding box of the set of 3D meshes is divided into a discrete grid of 3D voxels.
- Viewing cone determination (section 6.4): the set of voxels to be carved is defined as the intersections of the viewing cones. These voxels are set as opaque.
- VMH computation (section 6.5): the VMH is defined as the set of voxels intersecting the 3D mesh.
- Space carving (section 6.6): voxels intersecting any viewing line are set as transparent.

We show in Figure 6.2 an example of the expected reconstructed surface.

6.2 From depth maps to meshes

The first step of our framework is to exploit the underlying connectivity present in the depth maps and build a high resolution mesh from each available depth image. Each depth map is uniformly triangulated and each mesh vertex corresponds to a pixel if the depth map. This triangulation is performed in the image domain. Registration of the set of all meshes is performed by expressing each mesh vertex in world coordinate system using the following equation 6.1:

$$\begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix} \equiv \begin{pmatrix} & R & & T \\ & & & \\ 0 & 0 & 0 & 1 \end{pmatrix} \times \begin{pmatrix} f_x & S_{uv} & c_u & 0 \\ 0 & f_y & c_v & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}^{-1} \begin{pmatrix} u \\ v \\ 1 \\ 1/z_{cam} \end{pmatrix} \tag{6.1}$$

Where, X, Y and Z are the coordinates of a 3D point in the world reference Coordinate System. u and v are the coordinates of the projected 3D point in image Coordinate System, and z_{cam} is the depth of the pixel (u,v) with respect to the camera Coordinate System (with $z_{cam} \neq 0$).
R and T respectively denote the rotation matrix and the translation vector of the considered camera's Coordinate System in the world Coordinate System. f_x, f_y, S_{uv}, c_u and c_v are the camera's parameters.

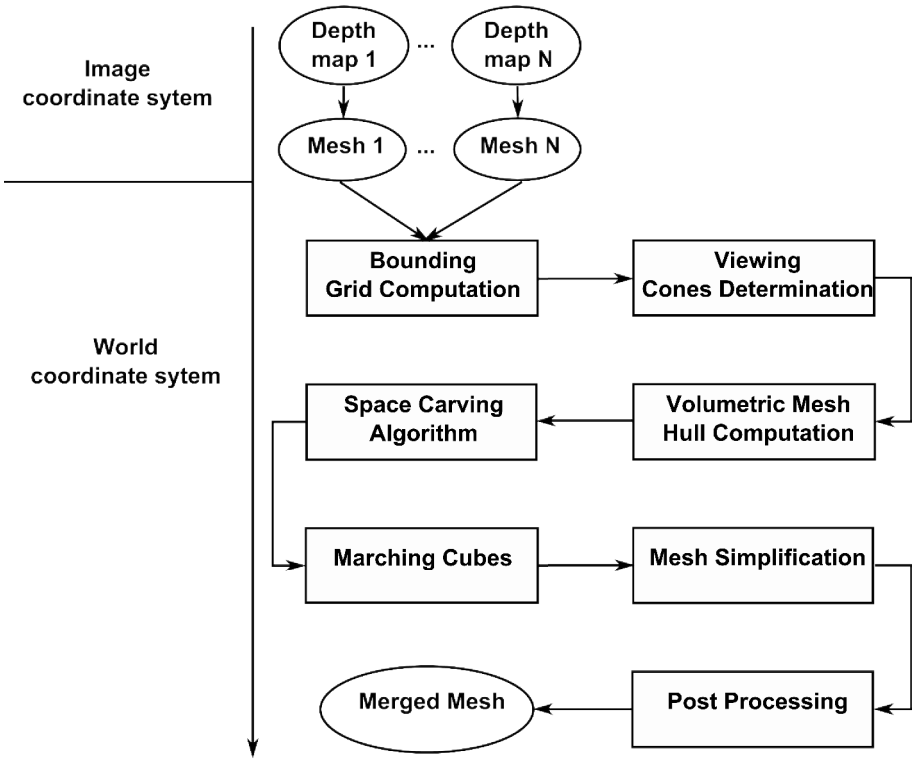


Figure 6.1: Our volumetric framework for depth maps fusion.

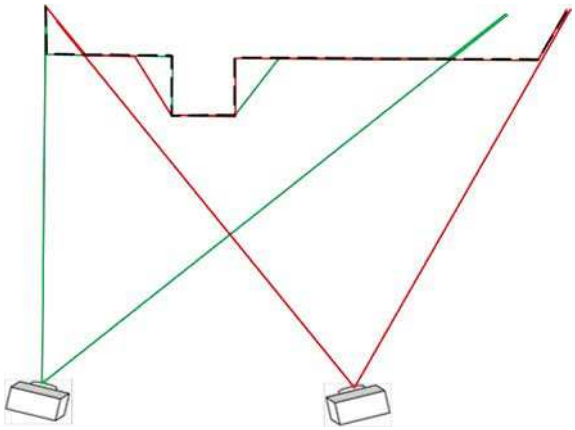


Figure 6.2: Example of two surfaces to be merged (one in green and the other in red). The expected surface as output of the algorithm is in black dotted line. Cameras viewing each mesh are also represented with their viewing cones.

6.3 Bounding grid computation

The set of input meshes are now considered for our volumetric representation. The basic data structure on which our volumetric framework operates is a three-dimensional grid. This grid is built by regular subdivision of the bounding box enclosing the input meshes into parallelepipedic voxels. A binary labeling strategy is used to label the voxels. Each voxel can be either opaque or transparent. Voxels labels are modified throughout the algorithm according to their relevance

to represent the scene.

6.4 Viewing cones determination

In order to be more efficient during voxels' traversal, we propose in this step to find the set of voxels lying in each camera's viewing cone. Voxels outside this viewing cone are labeled as transparent and such voxel don't need to be visited in the future. The viewing cone of each camera is modeled by a frustum defined by four planes bounding the range of visible points according to each axis (see Figure 9.5). The task is then to compute for each camera the set of voxels that lie in its corresponding frustum. To this end, frustum planes equations are determined for each camera, these equations allow to perform the inside/outside test for each voxel center against the considered frustum.

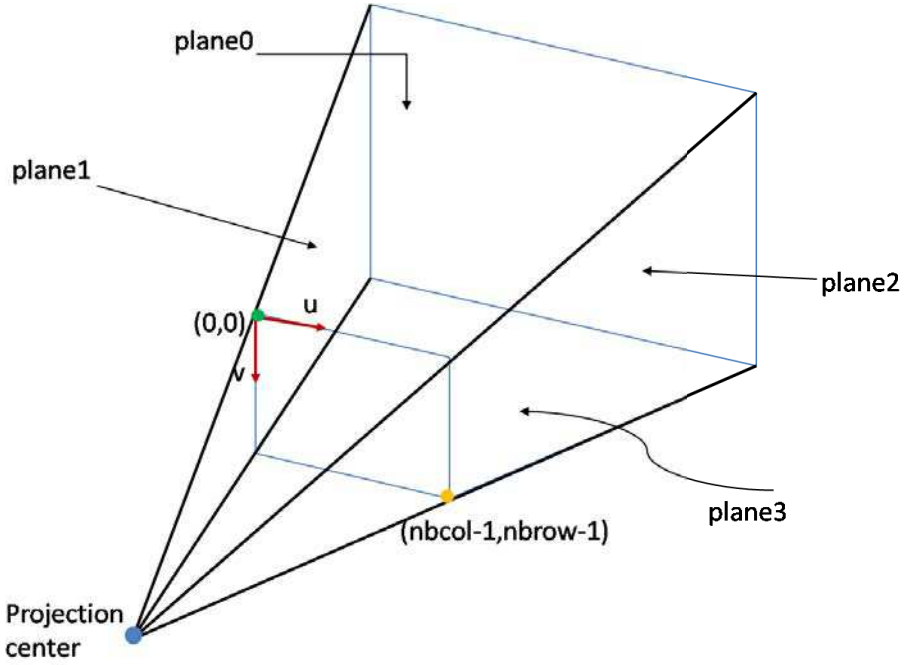


Figure 6.3: Frustum planes computation.

6.4.1 Frustum planes equations

As stated before, equation (6.1) represents the conversion from image to world coordinates system. This conversion is invertible, hence image coordinates can be expressed using the following equation:

$$\begin{pmatrix} su \\ sv \\ s \end{pmatrix} = \begin{pmatrix} f_x & S_{uv} & c_u & 0 \\ 0 & f_y & c_v & 0 \\ 0 & 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} & R^{-1} & & -R^{-1}.T \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix},$$

which can be reformulated by computing the product of the two central matrices:

$$\begin{pmatrix} su \\ sv \\ s \end{pmatrix} = \begin{pmatrix} p_{11} & p_{12} & p_{13} & p_{14} \\ p_{21} & p_{22} & p_{23} & p_{24} \\ p_{31} & p_{32} & p_{33} & p_{34} \end{pmatrix} \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix},$$

Where $s = z_{cam}$ denotes depth of the pixel (u,v) with respect to the camera Coordinate System. The last equation leads to the two following parametric plane equations:

$$X(p_{11} - up_{31}) + Y(p_{12} - up_{32}) + Z(p_{13} - up_{33}) + p_{14} - up_{34} = 0 \tag{6.2}$$

$$X(p_{21} - vp_{31}) + Y(p_{22} - vp_{32}) + Z(p_{23} - vp_{33}) + p_{24} - vp_{34} = 0 \tag{6.3}$$

Equation (6.2) with $u = 0$ (resp. $u = \text{nbcol} - 1$) is the equation $E_1(X,Y,Z)$ of the vertical left-hand plane 1 (resp. the equation $E_2(X,Y,Z)$ of the right-hand plane 2) see figure 9.5. Equation (6.3) with $v = 0$ (resp. $v = \text{nbrow} - 1$) determines the equation $E_3(X,Y,Z)$ of the upper plane 0 (resp. the equation $E_4(X,Y,Z)$ of the bottom plane 3). Once planes equations are computed, inside/outside points with respect to each camera can be determined by the following inside condition test.

6.4.2 Inside condition test

Given a point inside a given frustum $I(X_I, Y_I, Z_I)$, we can compute the signs that verifies the point I with respect to plane equations E_i. Each point inside the frustum should verify the same signs. $\forall i, 1 \leq i \leq 4$ compute $\text{sign}(E_i(X_I, Y_I, Z_I))$ where,

$$\text{sign}(x) = \begin{cases} +1 & \text{if } x \geq 0 \\ -1 & \text{otherwise} \end{cases}$$

Now, given a voxel defined by its center $V(X_V, Y_V, Z_V)$, if $\forall i, 1 \leq i \leq 4$, $\text{sign}(E_i(X_I, Y_I, Z_I)) = \text{sign}(E_i(X_V, Y_V, Z_V))$ then this voxel is inside the frustum, otherwise this voxel is outside.

At the end of this step, all voxels belonging to the union of cameras viewing cones are labeled as opaque. All the remaining voxels within the bounding grid are labeled transparent.

6.5 Volumetric Mesh Hull (VMH) computation

To be more efficient during voxel traversal, a Volumetric Mesh Hull (VMH) is built before the space carving process. For each input mesh, the set of voxels intersecting the input mesh defines the VMH associated with this mesh. At the end of this step, each voxel has an additional information defining which VMH(s) this voxel belongs to. For each mesh, each triangle is visited and scanned with respect to the volumetric grid, i.e. this scan defines the voxels lying on this triangle. All these voxels are then marked to belong to the current VMH by updating the additional information associated with the voxel. In order to scan the 3D triangle's surface, the classical 2D scan line algorithm is extended to the 3D setup. The 3D scan line algorithm is presented in algorithm 10 with an illustration in Figure 6.4.

Algorithm 10: Volumetric Mesh Hull (VMH) determination.

```
   //  Triangles traversal
1  for each triangle T do
2      for each axis X, Y and Z do
3          Compute the integer bounds b_min and b_max of T along the axis
           considered.
4          for m ← b_min to b_max do
5              Compute the intersection line of triangle T with the plane at
               coordinate m.
6              Find voxels along this intersection line using 3D Bresenham line
               algorithm.
7              Add these voxels to VMH.
```

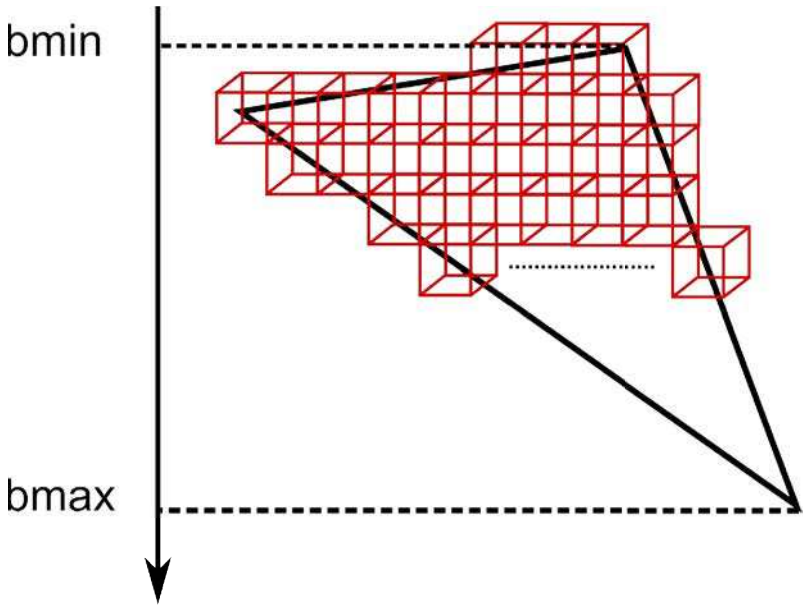


Figure 6.4: Construction of the VMH for a mesh triangle using scanline fill algorithm (See also algorithm 10).

6.6 Space carving

Until now, an opaque voxel is a voxel lying in at least one camera's viewing cone. A VMH voxel is a voxel lying on one of the surface mesh. In this step, space carving is performed by updating voxels labels from opaque to transparent for all voxels "in front of each VMH". More precisely, each VMH is considered and a geometric consistency criterion is employed in order to update voxels labels. This criterion is based on the relative position of the voxel according to the considered surface mesh and the corresponding viewpoint. A voxel is said to be geometrically consistent with a surface mesh if it lies behind it with respect to the viewing camera. Voxels detected as geometrically inconsistent are labeled as transparent. The carving process is performed by iteratively updating the carved

volume with respect to each camera. The volume to be carved is initialized to the union of voxels inside the set of viewing cones computed in section 6.4 (see figure 6.5). This volume is then refined iteratively with geometrical information lying in each view (see figures 6.5 and 6.6), carving occluding areas relative to each camera. For each of the available views, rays are cast from the viewpoint to each voxel of the camera's corresponding VMH. For each ray, voxels lying on the current ray are scanned using 3D Bresenham line retracing algorithm. Since they are geometrically inconsistent, these voxels are dedicated to be carved (i.e. their label is updated to transparent).

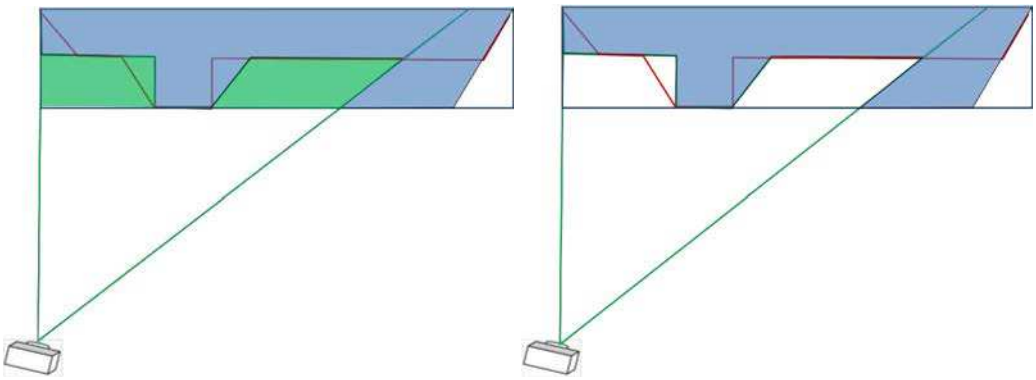


Figure 6.5: Space carving according to the left camera. Rays are cast from the camera center to the corresponding VMH (left). Green areas are carved. The result of this carving operation according to the green camera is shown on the right.

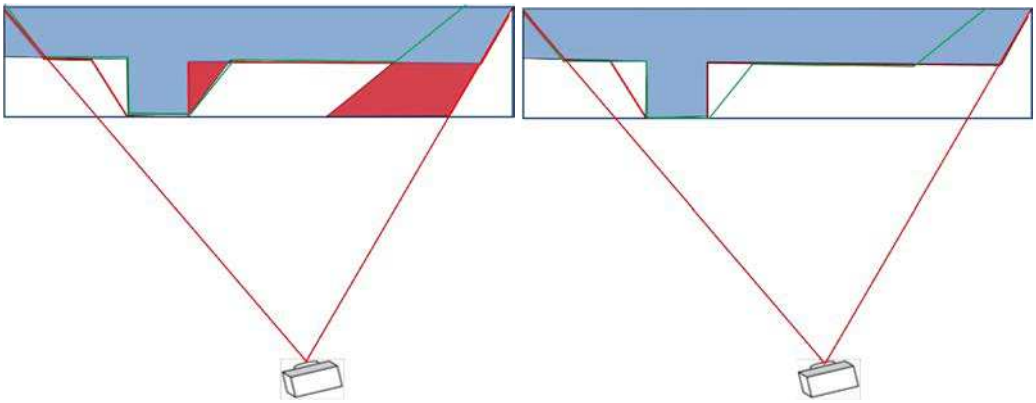


Figure 6.6: Space carving according to the right camera. For the carved volume to be updated, rays are cast from the right camera to its corresponding VMH (left). Red areas are carved. The result of this carving step is shown in the right image.

Algorithm 11: Space carving algorithm.

```
// Views traversal
for each camera do
    Get the camera position.
    Get the corresponding VMH.
    for each voxel in the VMH do
        Find the 3D Bresenham line between camera's position and voxel's
        center.
        Update each of these voxels' labels to transparent.
```

6.7 Marching cubes

The merged surface mesh is then extracted from our binary-labelled volumetric representation thanks to the Marching Cubes algorithm. Marching Cubes algorithm was originally introduced by [Lorensen and Cline, 1987]. This algorithm considers as input a regular volumetric data set. Such data set has a scalar value assigned to each voxel vertex. During processing, each voxel's vertex that has a value less or equal than a predefined isovalue is marked. All other vertices are left unmarked. Consequently the Marching Cubes algorithm outputs the triangular mesh connecting the marked vertices, this triangular mesh is referred to as iso-surface. As each of the eight vertices of a voxel can be either marked or unmarked, there are $256 = 2^8$ possible marking scenarios for a given voxel. Each scenario represents a voxel-isosurface intersection configuration. However, Marching Cubes considers also reflective and rotational symmetry between a pair of voxels. This results in just 15 marking scenarios represented in Figure 6.7.

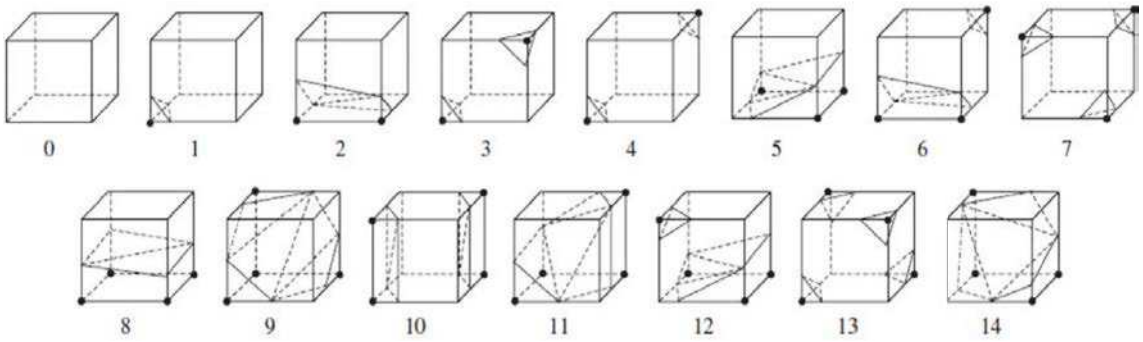


Figure 6.7: The 15 voxel-isosurface intersection scenarios in Marching cubes (image from [Lorensen and Cline, 1987]).

6.8 Mesh simplification

For transmission purposes, we need to simplify the merged mesh in order to remove geometric redundancies due to the regular subdivision of the grid. Since we aim to render texture images using this merged mesh at decoder side, we

should opt for a simplification algorithm that removes such redundancies without altering the quality of the rendered images. To this end, a study was carried out by [Franco De Carvalho, 2012] during his internship that I co-supervised. The work of De Carvalho [Franco De Carvalho, 2012] focused on the implementation of an image-based mesh simplification algorithm based on the idea of [Lindstrom and Turk, 2000]. In this section, we will present the main results obtained during this internship. The reader may refer to [Franco De Carvalho, 2012] for more details. The image driven simplification algorithm proposed by Lindstrom *et al.* in [Lindstrom and Turk, 2000] aims at simplifying the mesh using a set of edge collapse operations (see Figure 10.9) that are controlled by a cost function. The cost function is evaluated using an image distortion metric between the rendered mesh after each edge collapse and the input texture images. In [Franco De Carvalho, 2012], we used the L_2 error as image distortion metric since the PSNR is the reference metric used within the video compression community.

In order to re-evaluate the cost of each mesh edge after a given collapse, one should render the whole mesh after such edge collapse. This is time consuming. For this reason, [Lindstrom and Turk, 2000] proposed to render only affected triangles after each collapse. However, even with this approximation, [Franco De Carvalho, 2012] reports about 29 hours to simplify a mesh with 264566 triangles using a variant of the algorithm proposed by [Lindstrom and Turk, 2000].

The performance of this algorithm was tested and compared with a geometric based mesh simplification algorithm `vtkDecimatePro` [1].

Results in terms of PSNR are shown for two merged meshes balloons and breakdancers (*cf.* Appendix A) using image driven mesh simplification [Franco De Carvalho, 2012] in table 6.1 and the geometric simplification in table 6.2. The PSNR is first computed for each view between the rendered non-simplified mesh and the texture corresponding to this view. The PSNR is then computed for each view between the rendered simplified mesh and the texture image corresponding to this view.

The final PSNR shown in tables 6.1 and 6.2 corresponds to the average PSNR over all the views. For breakdancers sequence, the considered views are views from 0 to 7. For balloons sequence, the considered views are 1, 3 and 5.

Sequence	PSNR before simplification	PSNR after simplification
Breakdancers	31.58	32.22
Balloons	31.21	31.88

Table 6.1: PSNR before and after simplification using an image-driven mesh simplification algorithm.

These results show that image driven simplification algorithm provides better results in terms of PSNR comparing to the geometric simplification. However, the difference is quite small. Besides, the geometric simplification performs faster than image driven simplification (several seconds comparing to some hours). For this reason and because the image driven simplification does not improve the quality of simplified mesh during rendering (difference is at most 0.65 dB), we

[1] The algorithm used is based on (see `http://www.vtk.org/doc/release/4.0/html/classvtkDecimatePro.html` for more details.)

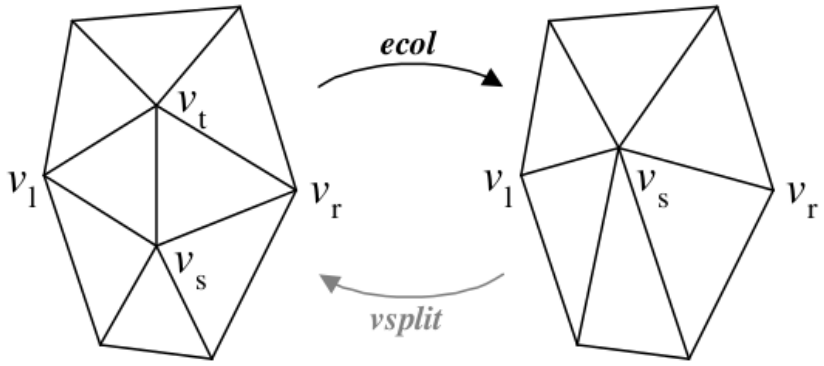


Figure 6.8: Edge collapse operation for mesh simplification (image from [Lindstrom and Turk, 2000]).

Sequence	PSNR before simplification	PSNR after simplification
Breakdancers	31.58	31.55
Balloons	31.21	31.20

Table 6.2: PSNR before and after simplification using an geometric mesh simplification algorithm.

choose the geometric mesh simplification algorithm as a solution to simplify the merged mesh.

6.9 Post-processing

The purpose in this step is to make the merged mesh more consistent with the input depth data. To this end, each vertex of the merged v_{merged} mesh is mapped to the closest vertex of the input meshes v_{input} regardless to which view the vertex v_{input} belongs to. This mapping is performed if:

$$\|v_{\text{input}} - v_{\text{merged}}\|_2 \leq R$$

If v_x, v_y and v_z represent respectively the voxel's size along X, Y and Z axis. R can be defined as:

$$R = \frac{\max(v_x, v_y, v_z)}{2} \tag{6.4}$$

6.10 Results

Figures of the proposed modeling scheme are provided for 4 test sequences: ballet (Figure 6.10) and breakdancers (Figure 6.11) and balloons (Figure 6.12) and Kendo (Figure 6.13) (*cf*. Appendix A for more information about these sequences). These results show that Poisson surface reconstruction method provides smoother models than ours. This is due to the Gaussian filter the authors

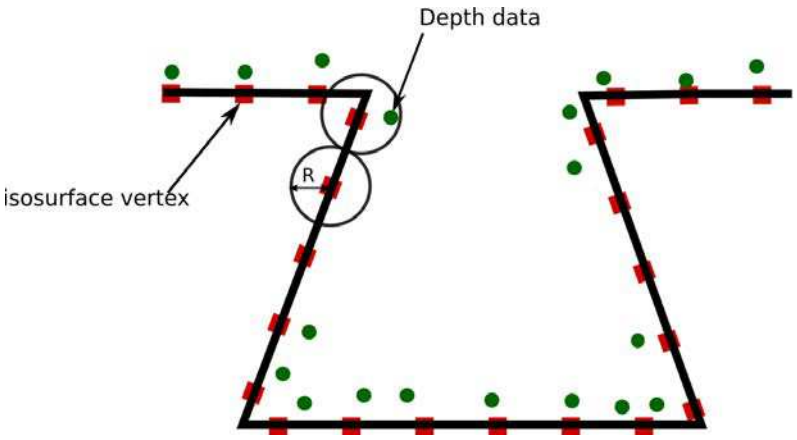
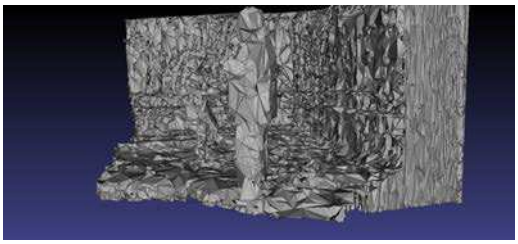
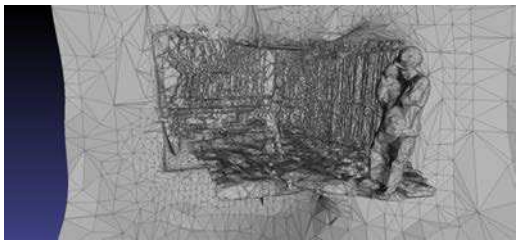


Figure 6.9: Post processing the merged mesh. A merged-mesh vertex represented with red square is mapped to the closest input-mesh-vertex represented with green circle if the distance between the vertices is below R.

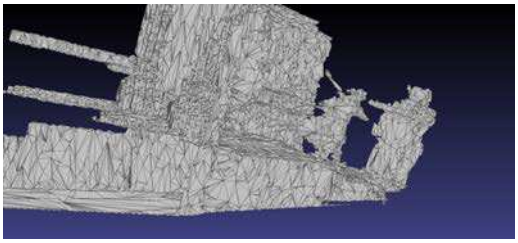
use [Kazhdan et al., 2006] in order to compute the gradient of the indicator function (which by definition piece-wise linear). Despite the smoothness of Poisson models, Poisson-based models are less efficient than ours in terms of PSNR (*cf*. table 6.3). This PSNR is computed after texturing these 3D models with the proposed multi-texturing method in chapter 8 and averaging over the available views.



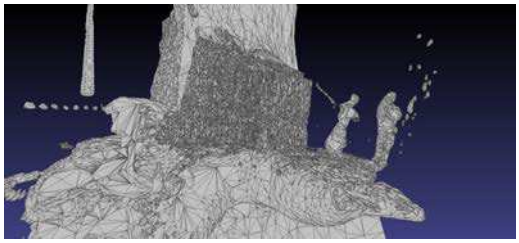
(a) Right view of our model



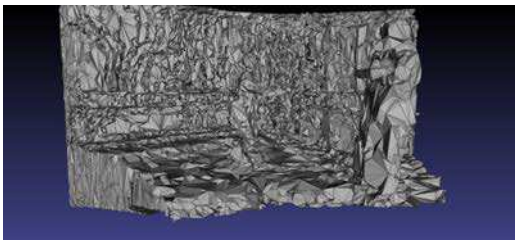
(b) Right view of Poisson model



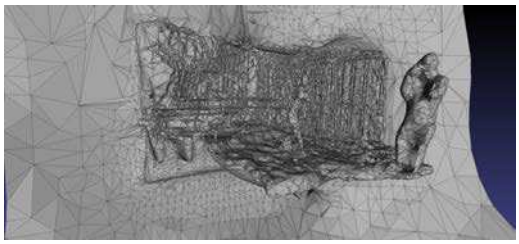
(c) Left view of our model



(d) Left view of Poisson model



(e) Frontal view of our model



(f) Frontal view of Poisson model

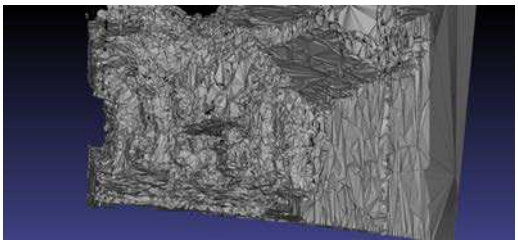
Figure 6.10: Visual results of the proposed model vs. Poisson Model for ballet sequence.

Sequence	PSNR (dB) our model (vol res 250)	PSNR (dB) Poisson (octree depth=10)
Breakdancers	32.14	31.46
Balloons	30.75	29.80
Kendo	33.92	32.14
Ballet	29.19	29.37

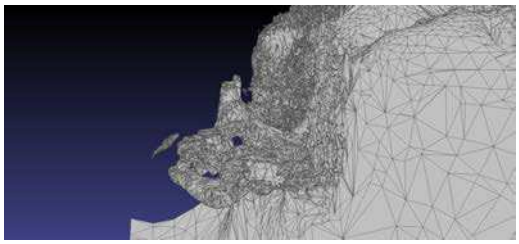
Table 6.3: Comparison between the proposed model and Poisson model in terms of PSNR between rendered views and input images.

6.11 Conclusion

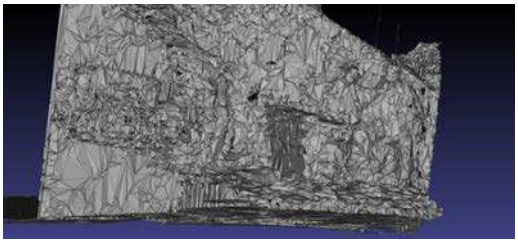
In this chapter, we introduced a new volumetric framework for depth maps fusion, the resulting merged mesh represents the depth information present in each depth map. We also showed that our modeling system outperforms the state of the art based on Poisson surface reconstruction. For a more photo-realistic rendering, this merged mesh is then intended to be textured using input textures.



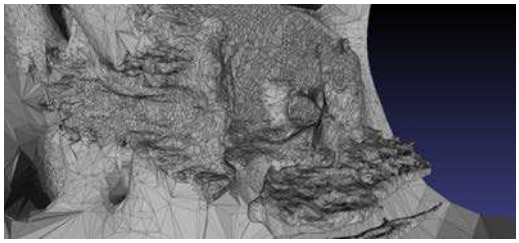
(a) Right view of our model



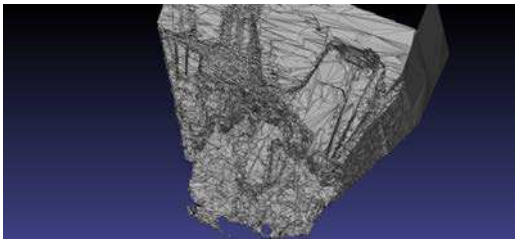
(b) Right view of Poisson model



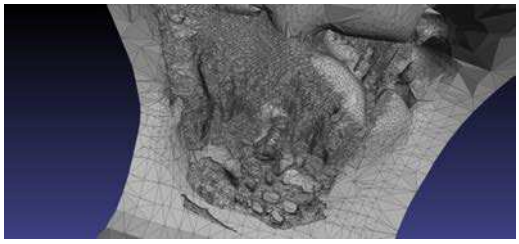
(c) Left view of our model



(d) Left view of Poisson model

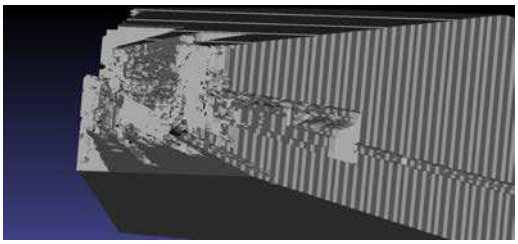


(e) Frontal view of our model

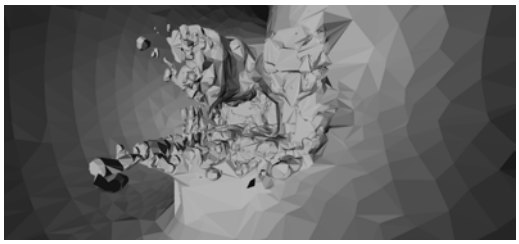


(f) Frontal view of Poisson model

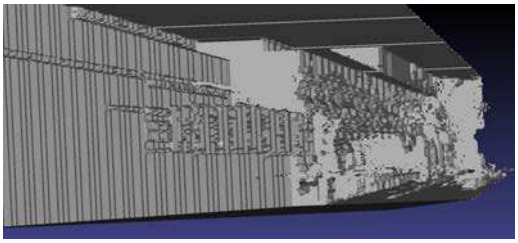
Figure 6.11: Visual results of the proposed model vs. Poisson Model for break-dancers sequence.



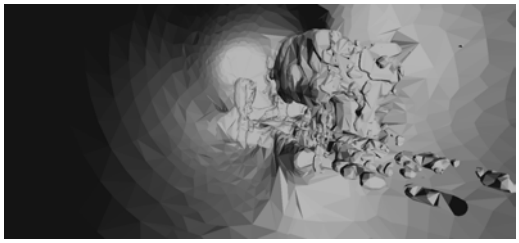
(a) Right view of our model



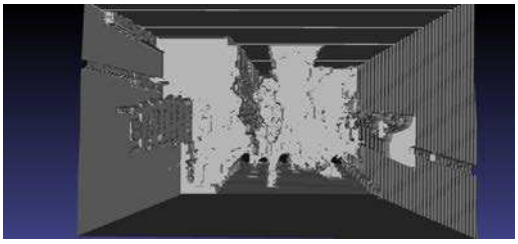
(b) Right view of Poisson model



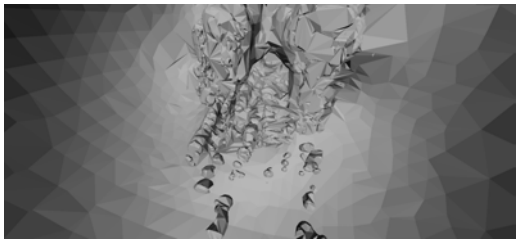
(c) Left view of our model



(d) Left view of Poisson model

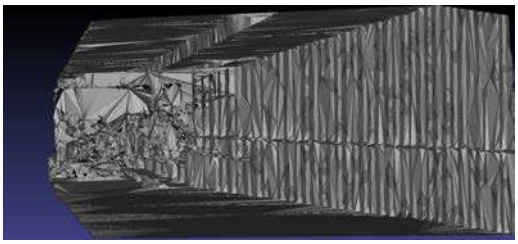


(e) Frontal view of our model

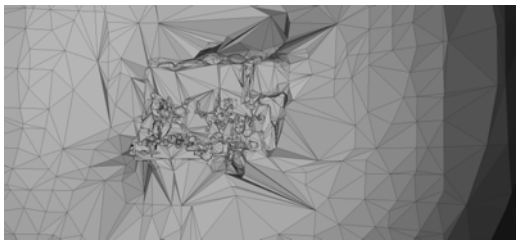


(f) Frontal view of Poisson model

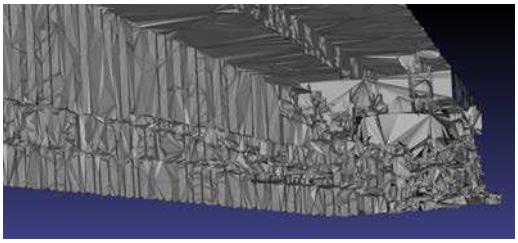
Figure 6.12: Visual results of the proposed model vs. Poisson Model for balloons sequence.



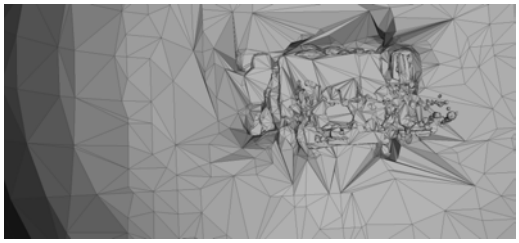
(a) Right view of our model



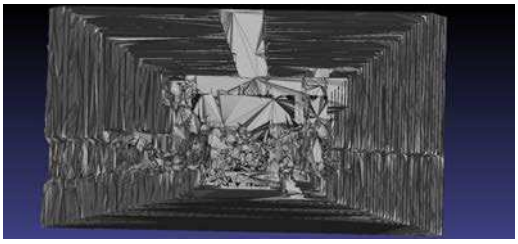
(b) Right view of Poisson model



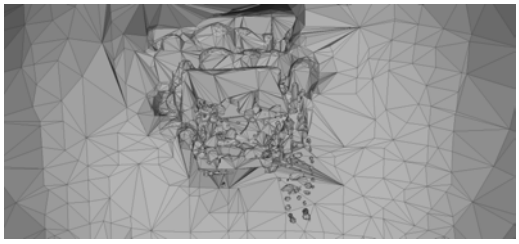
(c) Left view of our model



(d) Left view of Poisson model



(e) Frontal view of our model



(f) Frontal view of Poisson model

Figure 6.13: Visual results of the proposed model vs. Poisson Model for kendo sequence.

Part III

Multi-view Texture Mapping

Chapter 7

Multi-view Texture Mapping: a State of the Art

Contents

7.1 Introduction

Having determined the merged mesh as a representation of the scene geometry, we now need to texture this mesh given the set of available images. The purpose of texturing this mesh is to be able to generate any image as if it was captured from a real camera given this camera's parameters. Multi-texturing 3D models has witnessed an increasing interest in the last few decades. Applications range from architecture to entertainment industry. However, several issues are still challenging in this area. These challenges are related to the approximate input data or to the algorithm used for texturing. We propose to discuss such issues in section 7.2. Multi-texturing will then be surveyed in this chapter considering two methods: the first one consists in projecting textures onto the 3D model which is referred to as projective texture mapping. This will be the subject of section 7.3. The second one consists in flattening the 3D mesh onto a 2D domain which will be studied in 7.4.

7.2 Challenges of multi-texturing 3D models

Given a 3D model describing a scene and a set of overlapping photographs (texture images), multi-texturing aims at providing each mesh triangle one or several textures. In this section we will present the challenges related to multi-texturing 3D models.

7.2.1 Erroneous data

A multi-texturing system should be able to handle erroneous data. Such errors may be due to the approximate geometric model or to camera calibration.

Geometric inaccuracies: Previous studies [Gortler et al., 1996, Buehler et al., 2001, Debevec et al., 1996, Vedula et al., 2005] show that if a 3D model, also called geometric proxy, describing the scene is accurate enough then any view could be reconstructed without noticeable artefact. However, acquiring such reliable geometric proxies is not an easy task and usually relies on (dense) correspondence estimation between the views which is also an active research topic in computer vision community. Figure 7.1 shows how geometric inaccuracies lead to erroneous texture projection during rendering.

Camera calibration errors: If the 3D model is not sufficiently accurate and the cameras are not well calibrated this may cause significant texture misalignment during projection. Figure 7.2 illustrates this phenomenon.

In presence of such geometric inaccuracies and camera calibration errors, the final rendered views may present some visual artefacts such as ghosting and seams visibility.

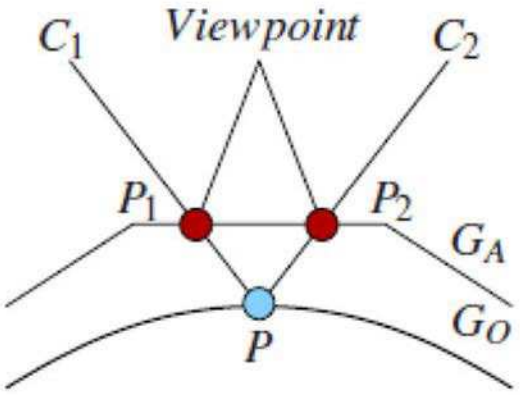


Figure 7.1: Geometric inaccuracies in multi-texturing a 3D model. A point P on the original surface G_O is erroneously projected to 3D-position P_1 from camera C_1 and to 3D-position P_2 from camera C_2 when the approximate geometry proxy G_A is available.(image from [Eisemann et al., 2008])

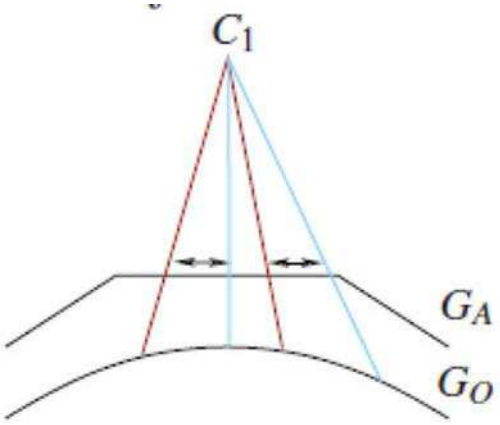


Figure 7.2: Camera calibration problem in the context of multi-texturing (image from [Eisemann et al., 2008]).

7.2.2 Rendering artifacts

If camera's parameters are approximated, Figure 7.2 shows that this leads to false pixel projection (red lines) comparing to real projections (blue lines) which causes a texture shift on the object surface and hence leads to ghosting artefacts during rendering. Furthermore, if the 3D model is not sufficiently accurate, this may cause significant texture misalignment during rendering.

Ghosting artifacts: Due to texture shifts on objects surface, several authors propose to perform some weighted averaging after texture image projections. This causes some visible ghosting artefacts during view synthesis.

Seams Visibility: Seams refer to texture discontinuities visible on the textured mesh. This problem is usually due to different lighting conditions between the views, and it occurs when assigning two different textures to adjacent triangles (see figure 7.3).

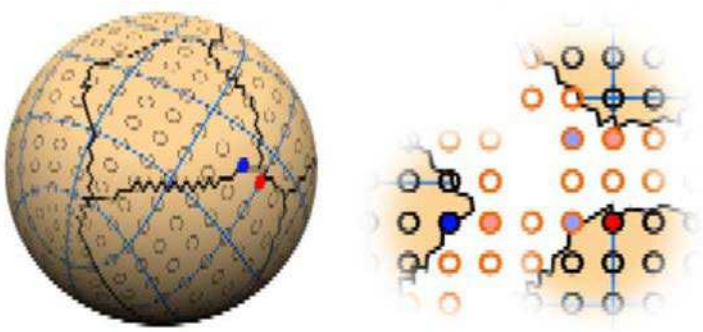


Figure 7.3: Neighbouring red and blue points during rendering (left) which are not neighbours after chart decomposition. (image from [Ray et al., 2010])

Texture Stretch: Texture stretch occurs where small texture distances are mapped on large surface distances.

7.2.3 Compression

A key challenge in texturing 3D models is the ability to generate texture maps using formats suitable for transmission or storage. This issue has not been widely investigated even with the introduction of texture atlases that will be discussed in detail in section 7.4.

7.2.4 Fidelity

Fidelity is related to the algorithm used for multi-texturing and refers to the ability of a multi-texturing system to allow high fidelity between original and rendered views using state of the art of image quality metrics.

In the following sections, we will present existing methods for multi-texturing. These methods can be divided into two categories: methods that use projective texture mapping, this will be the subject of section 7.3 and methods that parameterize locally the 3D model section 7.4.

7.3 Multi-texturing via projective texture mapping

The first family of methods in multi-texturing that will be described in this section is referred to as projective texture mapping (see the tutorial proposed by [Everitt, 2001] from Nvidia related to this topic[1]). Basically, projective texture mapping, first introduced by [Segal et al., 1992], consists in projecting a texture image onto a 3D scene exactly as if it would be done by a slide projector. See figure 7.4. In conventional texture mapping operation [Heckbert, 1986, Haeberli and Segal, 1993], a texture image is applied to a mesh triangle (or more generally a polygon) by assigning texture coordinates to the triangle's vertices. These texture coordinates

[1] For more details, the reader may also refer to projspot demo available online http://home.xyzw.us/ by the same author

define coordinates on the texture image to be applied. The so-called texture coordinates are interpolated all along the triangle to determine at each triangle's pixel, a texture image value. The result of this operation is that a portion of this texture image is mapped onto the triangle when this one is viewed on the screen. On the contrary, in projective texture mapping, a texture is projected and textures coordinates at a vertex are subsequently computed dynamically (*i.e.* as the projector moves) instead of being assigned as fixed values.

In order to project such texture onto the 3D model we must be able to determine which parts are viewed by the camera corresponding to the texture image. Considering the slide projector example, light will not reach invisible parts of the scene. To this end visibility should be determined first, this is the subject of section 7.3.1. In the second step, we should determine among the set of overlapping textures, the best texture for each mesh triangle, this will be presented in section 7.3.2. As a post processing step, blending is usually performed in order to guarantee smooth transitions between triangles with different texture 7.3.3.

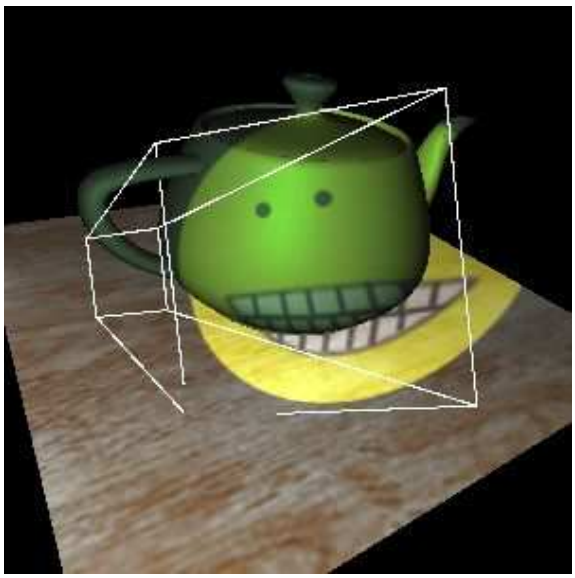


Figure 7.4: Projective texture mapping. A slide projector is projecting a smiley face onto the teapot. (image from http://home.xyzw.us/~cass/demo_images)

7.3.1 Visibility determination

In this section we will review some state of the art methods for visibility determination. The reader may refer to [Cohen-Or et al., 2003] for a more complete survey. Visibility determination is a well-studied problem in computer graphics. Originally, visibility determination algorithms are used to determine which pixels should be drawn on the screen given a collection of objects in the scene. Such algorithms can be classified into two categories [Foley, 1996]: image-space algorithms and object-space ones. Using an image-space algorithm, visibility aims to determine which object is visible at each pixel in the image plane. The algorithm can be summarized in the following:

Algorithm 12: Image space based visibility determination.

```
// Pixels traversal
for each pixel P do
    Determine the object closest to the viewer that is pierced by the ray from
    the camera center to the pixel P.
```

On the contrary, object space methods tend to compare objects of the scene with each other and to eliminate objects or parts of objects that are not visible according to the algorithm 13 hereunder:

Algorithm 13: Object space based visibility determination.

```
// Objects traversal
for each object in the scene do
    Determine parts of the object that are not obstructed by any other object
    or any other part of the same object.
```

However, naive formulations of image-based or object-based algorithms may require a huge amount of operations. In order to minimize such operations, several techniques have been developed that take into account coherence *i.e.* local similarities in object space or image space. Exploiting such coherence include triangle coherence, edge coherence, depth coherence, *etc*. We propose here to survey two methods exploiting such coherence. These methods are: ray-tracing, scan line and Z-buffer.

7.3.1.1 Ray casting

Ray casting (or ray tracing [2]) is an image space based algorithm that was developed by [Appel, 1968]. In his work, Appel proposed the following algorithm 14 (see also Figure 7.5).

Algorithm 14: Ray tracing algorithm

```
Select viewpoint and image plane where projections will be performed.
// Scan line traversal
for each scan line in image do
    for each pixel on scan line do
        Determine ray from viewpoint through the current pixel.
        for each object in the scene do
            if current object is intersected and is the closest then
                save intersection point and object ID.
        Set pixel color to the color of the closest intersection point.
```

[2] ray casting and ray tracing are often used synonymously. However, ray casting refers to visible determination algorithm, while ray tracing refers to the recursive algorithm used traditionally for shadow determination

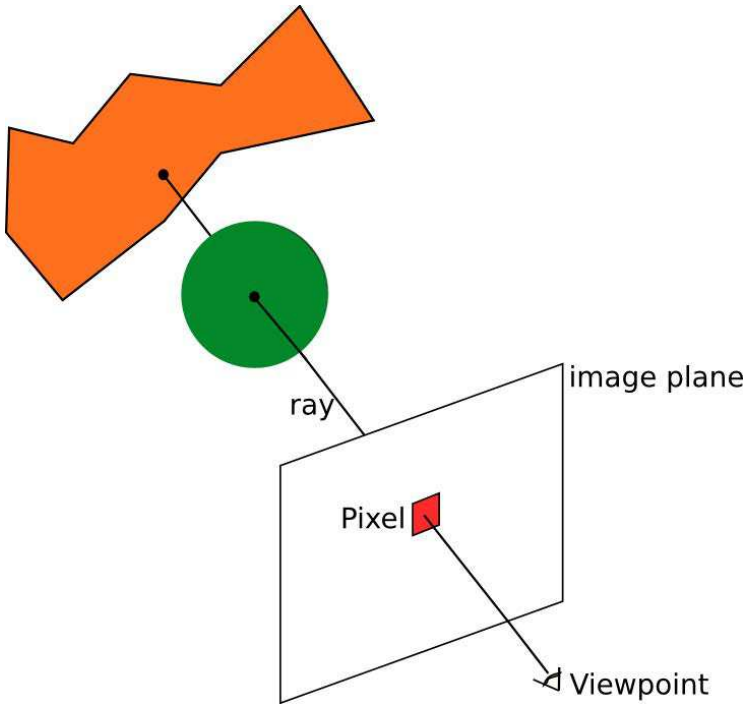


Figure 7.5: Ray casting algorithm for visibility determination. A ray is cast from the viewpoint to each pixel in order to determine the closest object intersected.

In this formulation of visibility based ray-tracing, the algorithm computes intersections from the camera's position with each object in the scene. In order to speed up computation, several approaches tend to speed up individual intersection computation or avoid unnecessary intersections by introducing hierarchies or spatial partitioning methods.

7.3.1.2 Z-buffer method

The Z-buffer is also an image-space method developed by [Catmull, 1974]. It is one of the simplest methods for visible surface determination. It requires two arrays for storing data, a frame buffer f where color values are stored and a **z-buffer** that stores depth values with the same number of points as in frame f (see Figure 7.6). The z-buffer algorithm has some advantages. Besides its simplicity the buffer can be saved using an image array. Visibility determination using z-buffer technique was proposed by [Debevec et al., 1998]. This method aims to split each polygon as much as necessary so that it becomes fully visible or fully invisible. The steps for visibility computation as presented by [Debevec et al., 1998] can be summarized in the following:

1. An ID is assigned to each mesh triangle. If this triangle is subdivided later the resulting triangles will keep the same ID.

2. If there are some intersecting triangles, subdivide them along the intersection line.

3. Clip the polygons against all images boundaries so that the resulting polygons lie exactly in the totally inside or totally outside the viewing frustum.

4. For each view render the mesh with Z-buffering activated (see Figure 7.6) and using triangles IDs as their colors.

5. For each front facing triangle, uniformly sample points and project them onto the image plane. Retrieve the polygon ID at each projected point from the color buffer. If the retrieved ID is different from the current polygon ID, the potentially occluding polygon is tested in object-space to determine whether it is an occluder or coplanar.

6. Clip each triangle with each of its occluders in object space.

7. Associate to each triangle a set of views where it is totally visible.

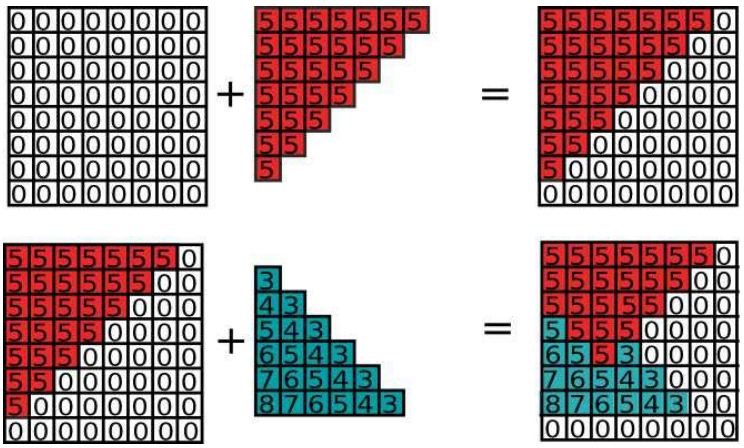


Figure 7.6: Z-buffer technique for scene drawing.

7.3.2 Best texture assignment

In a multi-texturing setup, if the set of input texture images overlap, one must determine the best texture for each triangle. Several criteria can be considered in order to achieve this assignment. This subsection surveys these criteria.

7.3.2.1 Geometric criteria for best texture selection

Debevec *et al*. [Debevec et al., 1996, Debevec et al., 1998] proposed a view dependent texture mapping approach. In this scheme, the user specifies a virtual view to render the mesh, and the best texture for each mesh triangle is chosen as the one the minimizes the angle between the triangle normal and the viewing directions of input cameras (see Figure 7.7). For instance, when generating a novel view from the left, it would be better to use the image corresponding to the left view as the texture map. The major drawback of View Dependent Texture Mapping approaches is that they require the transmission of the set of all input texture image which may be tremendous for transmission purposes. The Unstructured Lumigraph Rendering method [Buehler et al., 2001] uses a penalty function for

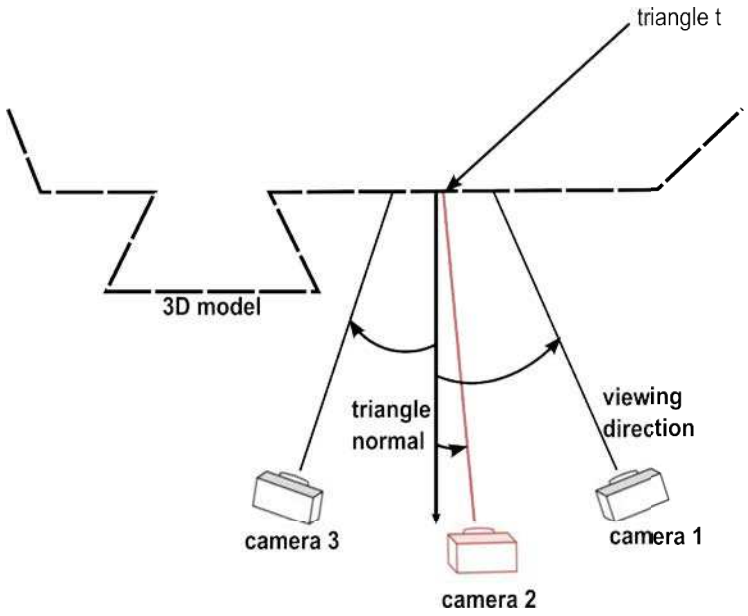


Figure 7.7: Best texture selection using triangle normal criterion.

each view in order to determine the best view for each triangle, this penalty is a linear combination of three terms: visibility constraints, the angle between the desired view to render and the set of input cameras and the resolution of the projected triangle.

7.3.2.2 Best texture as a labeling problem

In order to reduce the seams visible during rendering. Best texture assignment was also regarded as a labeling problem. In [Lempitsky and Ivanov, 2007], the authors define a cost of texturing the triangle T_i with the image from view V_j by a cost W_j^i. Cost computation is based on the angle between the viewing direction and the face normal. Each texture should is then assigned as a label to the triangle T_i. They expressed their labeling strategy as on energy minimization problem with two terms, the first one describes how better an input texture fits the triangle and the second one penalizes seams' visibility.

7.3.3 Blending

Based on the proposed energy minimization scheme [Lempitsky and Ivanov, 2007], Gal *et al*. [Gal et al., 2010] extend the best texture search space to include a set of warped images that compensate geometric errors and use Poisson blending [Pérez et al., 2003] to address lighting variation between adjacent textures. Other approaches form a single texture map as an image panorama. In this scheme, several authors formulate multi-texturing as an image stitching problem. These authors propose to blend the set of available images [Wang et al., 2001, Baumberg, 2002, Rocchini et al., 1999] using image processing tools in order to compute the correct weights for blending, or by formulating stitching as a super resolution problem [Iiyama et al., 2010]. Allen *et al*. [Allene et al., 2008] proposes a multi-band blending, they use Laplacian pyramids of the input images as the multi-band decomposition, and approximate them by differences of Gaussians.

The use of weighted blending between input lessens discontinuities but leads to some ghosting artefacts. It has been observed from [Dellepiane et al., 2012] that Flow-based optimization leads to sharp features and no discontinuities. They suggest the use of blending jointly with flow optimization in order to alleviate such artefacts.

In this section we reviewed existing methods for texturing 3D models using projective texture mapping. These methods try to alleviate visual artefacts on rendered images by modifying the input images or the input geometry or by blending the texture images. However, these methods consider geometric criteria for best texture assignment and do not provide a suitable format for transmission of input texture images.

7.4 Multi-texturing via texture atlases

Another family of methods propose to flatten the 3D model and establish a correspondence between input texture images and the flattened domain. Flattening consists in finding a parametrization, *i.e.* a one-to-one correspondence, of the 3D model onto a 2D domain. However applying such global transformation to the whole 3D model may drastically distort complex or a highly curved objects in the scene. For this reason, it is more convenient to partition the 3D model into a set of charts homeomorphic to discs. Each chart is then parametrized. The so-called charts are finally packed together in texture space. The result of the whole process is referred to as a texture atlas [Maillot et al., 1993]. In the next paragraphs, we propose to review the process of acquiring texture atlases that may be decomposed into the three following steps: first mesh decomposition into charts 7.4.1, second parametrization of each chart independently 7.4.2 and finally charts packing in texture space 7.4.3.

7.4.1 Mesh decomposition into charts

The first step for building a texture atlas is to decompose the input mesh into charts. Mesh segmentation techniques provide an elegant approach to such a partitioning. More specifically, since we try to flatten the 3D mesh locally, we should segment the input mesh into nearly planar regions. This was the idea behind [Maillot et al., 1993], who segments the mesh triangles into buckets according to their normals using the Gauss map. A connectivity graph is built from the set of buckets and the graph edges are sorted according to a predefined threshold. Edges with distance smaller than this threshold trigger successive buckets unions. A simpler method using triangles normals was proposed by [Ziegler et al., 2004] and Other approaches rely on incremental clustering of mesh triangles into charts. Typically, such methods start by selecting an initial seeds of mesh triangles and generate charts by growing these seeds. In [Lévy et al., 2002], the authors grow the seeds by detecting high mean-curvature regions. In [Sorkine et al., 2002] the authors grow chart while parametrizing them. And the chart stops growing when a distortion bound is reached or if an overlap is detected.

7.4.2 Parametrization of charts

Once the charts are defined, the next step is dedicated to charts flattening. Surface parametrization theory [Hormann et al., 2007, Floater and Hormann, 2005, Sheffer et al., 2006] provides an elegant framework to this end. Using such scheme, a chart is flattened by minimizing an objective functional that aim to reduce a particular distortion measure such as angle or area distortion. In [Lévy et al., 2002], they parametrize each chart using least-squares conformal maps with free boundaries. In [Sorkine et al., 2002], the authors use a stretch based metric based on the singular values of the Jacobian of the affine transformation between the original 3D triangle and its mapping in the plane. Sander *et al*. [Sander et al., 2001] propose also a texture stretch metric to parametrize the charts onto convex polygons, they first scale the polygons to have unit area and inside each chart, interior vertices are parameterized by minimizing the L_2 stretch metric.

7.4.3 Charts Packing in Texture Space

In order to address the compression issue, charts should be packed within a format suitable for transmission. Charts packing is the last step of multi-texturing based parametrization methods. The task is to arrange the unfolded patches into an image without overlap. In order to measure the packing efficiency, some researchers use the *packing efficiency* which measures the ratio between the total space occupied by patches and the image area. Consequently. The problem of charts packing is equivalent to the pants packing problem [Milenkovic, 1998] which is known to be an NP-complete problem. For this reason, several authors made some heuristics in order to speed up charts packing. [Sander et al., 2001] proposed an approach to pack minimal area bounding rectangles of the charts. In [Lévy et al., 2002], the authors propose a "Tetris packing" algorithm that for each chart to be added, they locate the horizontal position that minimizes the unused vertical space between the current and the new chart. Ziegler *et al*. use texture atlases in the context of multi-view video [Ziegler et al., 2004] Figure 7.8. A separate texture atlas is built for each body part and the built texture atlases are gathered together to form a unique texture.

In this section, we reviewed how texturing can be performed by building texture atlases from the input mesh and the texture images. Unlike projective texture mapping methods discussed in section 7.3. Multi-texturing based texture atlases provides a compact representation of the texture images suitable for transmission.

7.5 Conclusion

In this chapter we presented two methods for multi-texturing 3D models. The first one projects input texture images on visible parts of the mesh and choose the best texture for each triangle according to geometric criteria. Several ghosting artefacts due to approximate input data are noticeable and decrease dramatically

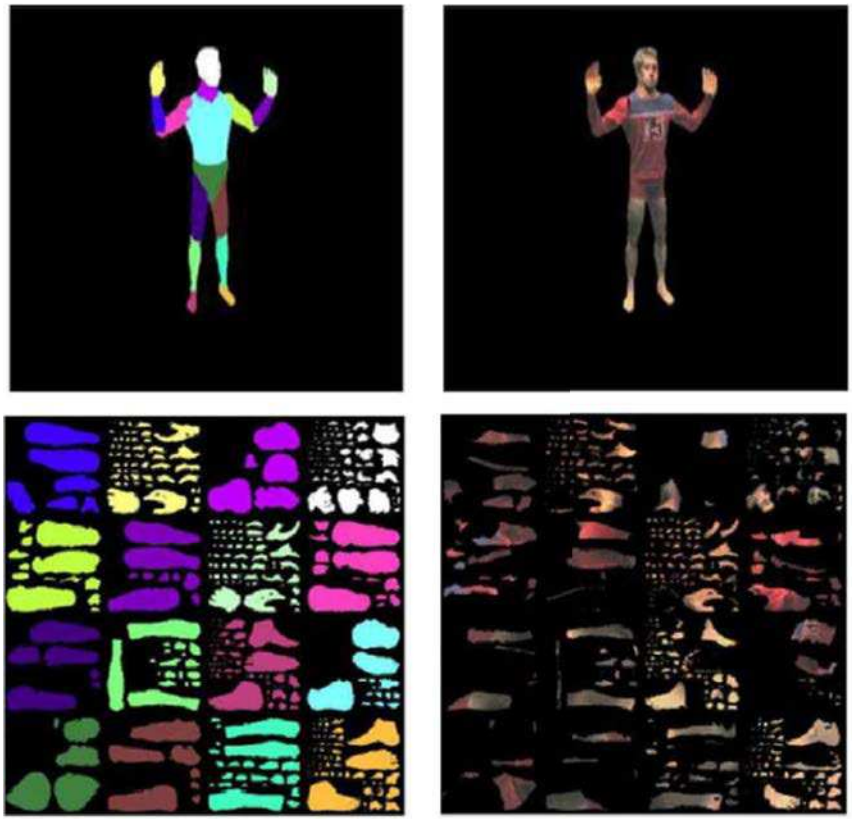


Figure 7.8: Use of texture atlas in the context of 3D Video. From top to bottom and from left to right: mesh partition into charts, textured model, charts are parametrized and then packed, texture atlas [Ziegler et al., 2004].

the quality of the rendered views. Besides, for transmission issues, the user has to encode the whole set of available textures. On the other hand, texture atlas generation provide a compact texture intended to be transmitted but introduces seams during charts parametrization. In the next chapter, we will present a new method intended to build high quality rendered views and that guarantees compactness of the transmitted data.

Chapter 8

Photo-consistency based Mesh Texturing

Contents

8.1 Introduction

In chapter 7, we presented some challenges related to multi-texturing 3D models. Quality of synthesized views as well as ability to produce compressible texture maps should be taken into account in a multi-texturing algorithm. However, several methods presented in chapter 7 do not fulfill these requirements. In this chapter, we propose a new multi-texturing framework in order to achieve high fidelity to input images and produce texture maps suitable for compression.

8.2 Photo-consistency based multi-texturing

In this chapter, we will present a new projective texture mapping algorithm designed for multi-texturing 3D models. The proposed framework is presented in Figure 8.2. First, a Z-buffer-based visibility determination algorithm with respect to each view is presented in section 8.3. Second, each available texture is mapped on visible triangles of the input mesh. Photometric projection error is then computed in section 8.4 for each triangle with respect to each view and stored in distortion images. Best texture assignment is then expressed in section 8.5 as the minimizer of an energy function that we refer to as photoconsistency metric. Finally, original views are synthesized by rendering the textured mesh, each triangle being textured with the best texture according to the photoconsistency metric. Figure 8.1 depicts the proposed photoconsistency based multi-texturing scheme. The triangle t is visible in cameras 1, 2, 3. For each available texture $I_i \in \mathcal{I}$, t is textured with I_i (red arrow for texture mapping operation) and projected (green arrow for projection operation) onto cameras 1, 2, and 3 - i.e. the set of cameras where t is visible. The error of texturing the triangle t with each available texture is computed. We refer to this error as the photoconsistency metric of the triangle t. Finally, the best texture is chosen as the minimizer of the photoconsistency metric.

8.3 Visibility Determination

Given the input mesh $\mathcal{M}$, total and partial visibility are first determined for each triangle with respect to each view. At the end of this step, each triangle will have one of the three labels: "totally visible" or "partially visible" or "hidden".

8.3.1 Total visibility

In this step only the labels "totally visible" and "hidden" are assigned to mesh triangles. The status of each mesh triangle is initialized to "totally visible". Visibility of each triangle is determined by computing the visibility of the triangle's vertices. A triangle is marked hidden if at least one of its vertices is hidden. Vertex visibility with respect to each view is determined using OpenGL z-buffer

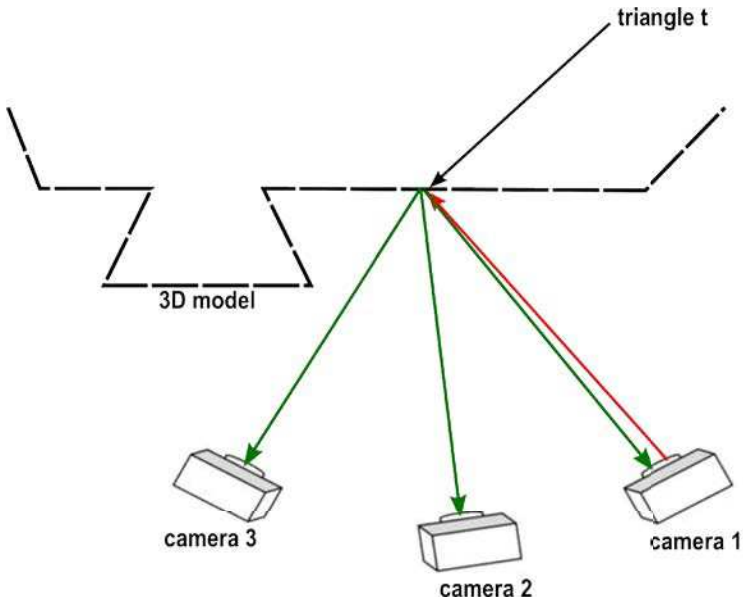


Figure 8.1: Best texture determination using photoconsistency. Red arrow for texture mapping operation and green arrow for projection operation.

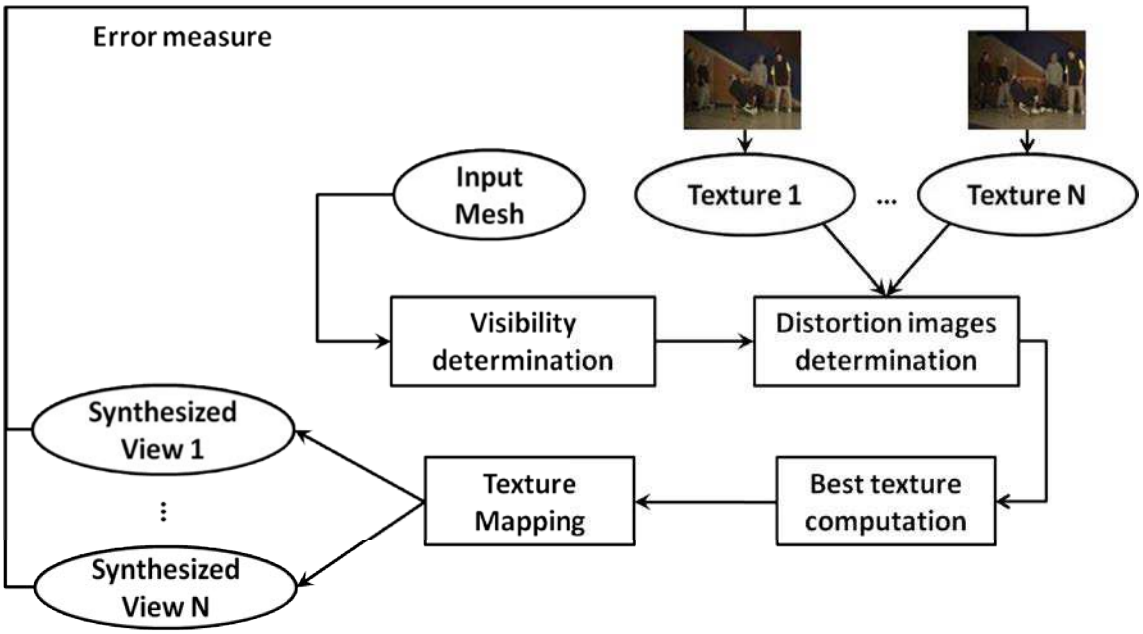


Figure 8.2: The proposed multi-texturing framework.

denoted as Z^j_{buffer} (*i.e.* z-buffer of the mesh projected onto the view V_j). During the first pass, the input mesh is projected onto the current view, and the z-buffer is extracted. The second pass is dedicated to vertex visibility determination using the computed z-buffer. Each mesh vertex is then projected onto the current view and the depth component, denoted as $z_{\text{projected}}$, is checked against the pixel depth $Z^j_{\text{buffer}}[q,l]$. If the projected vertex is behind the pixel in the z-buffer, then this vertex is hidden, and thus the set of mesh triangles sharing this vertex are marked hidden. The pseudo-code for total visibility determination is provided in algorithm 15.

Algorithm 15: Global visibility determination.

```
   //  Views traversal
1  for each view V_j do
2      Initialize all triangles to visible.
3      Project the M onto V_j.
4      Store the depth buffer Z^j_buffer.
       // Mesh vertices traversal
5      for each vertex v do
6          Determine (q, l, z_projected) the projection of the vertex v onto V_j.
7          if Z^j_buffer[q, l] < z_projected then
8              v is a hidden vertex.
9              Mark all the triangles with vertex v as hidden.
```

8.3.2 Partial visibility

Until now, each triangle has one of the two labels: "totally visible" or "hidden". At the end of this step, "hidden" triangles will be sorted out as "partially visible" or totally "hidden". To this end, the depth of each pixel is checked against the stored depth buffer in the projected triangle. Note that it is necessary to check the depth of the projected triangle. Partial visibility determination by counting the number of hidden vertices is not sufficient as some triangles could be partially visible and have a number of hidden vertices of three. The algorithm determining partial visibility is presented in algorithm 16.

Algorithm 16: Partial visibility determination.

```
   //  Views traversal
1  for each view V_j do
       // Mesh triangles traversal
2      for each triangle t do
3          if triangle t is hidden in V_j then
4              Determine P_j(t) the projection of triangle t onto V_j.
5              for each pixel (q, l) in P_j(t) do
6                  Get the pixel depth z from the triangle depth buffer.
7                  if z ≠ Z^j_buffer[q, l] then
8                      Mark t as partially visible in V_j.
```

8.4 Distortion images determination

Distortion image are intermediate data structures that will be used during selection of the best texture. In order to determine the error of texturing a triangle with an input texture I_i, each available texture is then mapped on the merged mesh and projected onto each camera. Let $P_j^i(\mathcal{M})$ be the projection of the mesh $\mathcal{M}$ onto the view V_j textured with the image I_i. For each available texture the following distortion image is computed in RGB color space:

$$D_{i,j} = \|P_j^i(\mathcal{M}) - I_j\|_2$$

Algorithm 17: Compute distortion images.

```
   // Textures traversal
1  for each texture I_i do
       // Views traversal
2      for each view V_j do
3          Compute P_j^i(M): projection of the mesh textured with texture I_i
           onto view V_j.
4          D_{i,j} = ||P_j^i(M) - I_j||_2
```

8.5 Best texture selection

In this section, the best texture is chosen for each mesh triangle using the distortion images. The best texture for each triangle relies on the computed set of pixels that belong to the projected triangle and visible in the current view V_j, i.e. the set of pixels:

$$\text{Vis}_j(t) = \{(q,l) \in P_j(t),\ (q,l) \text{ visible in } V_j\}$$

Figure 8.3 shows how $\text{Vis}_j(t)$ is determined for partially and totally visible triangles. The grid represents image pixels. In 8.3(a) the blue triangle is totally visible, the set of pixels defining $\text{Vis}_j(t)$ are shown in red. On the contrary, the blue triangle in 8.3(b) is partially visible as it is occluded by the green triangle. $\text{Vis}_j(t)$ in this case is restricted to the non-occluded pixels. The photoconsistency metric is then computed for each triangle with respect to $\text{Vis}_j(t)$. This metric measures the error of texturing the triangle t with the texture image I_i. From here on, for sake of simplicity, visible triangle will refer to totally or partially visible triangle as they will be treated the same way. $\forall i \in \{1,\dots,n\}$ we compute:

$$\mathcal{E}_{t,I_i} = \sum_{\substack{j=1 \\ t \text{ visible in } V_j}}^{n} \left(\sum_{(q,l)\in \text{Vis}_j(t)} D_{i,j}[q,l] \right),$$

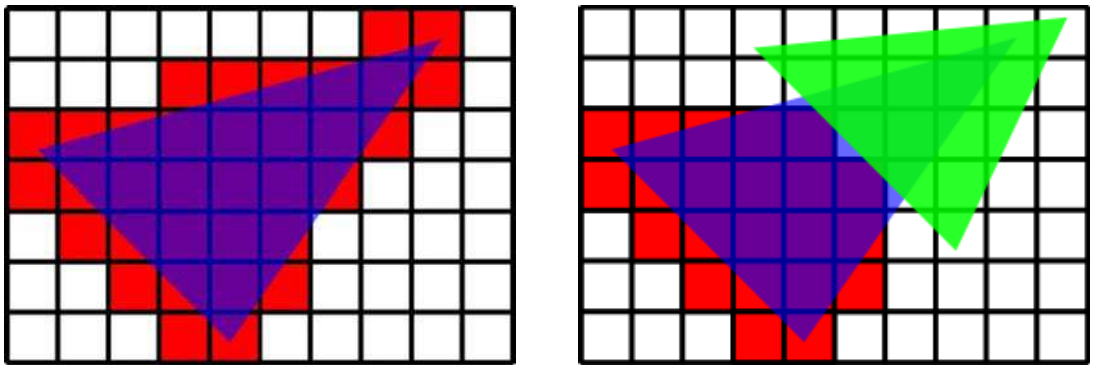
The best texture for a visible triangle t is then given by:

$$\hat{I}_t = \arg\min_{I_i \in \mathcal{I}} \mathcal{E}_{t,I_i}.$$

The steps of best texture computation are summarized in algorithm 18.

Algorithm 18: Compute the best texture for each triangle.

```
   // Triangles traversal
1  for each triangle t do
       // Views traversal
2      for each view V_j do
3          Determine Vis_j(t).
       // Textures traversal
4      for each texture I_i do
5          Compute E_{t,I_i}.
6      Determine Î_t.
```



(a) total visibility (b) partial visibility

Figure 8.3: Best texture computation. Handling partial/global visibility.

8.6 Texture mapping

Finally, original views are rendered. The 3D model is projected according to the camera's parameters supplied by the user. Then each triangle is textured using the texture coordinates determined during visibility determination step.

8.7 Discussion and results

In this section, we propose to compare the performance of our multi-texturing algorithm based on the photo-consistency metric for best texture assignment and a multi-texturing framework based on geometric criteria for best texture assignment. Geometric criteria used in this study are:

- $C_1 := \langle \vec{d}_j, \vec{n}_t \rangle$ *i.e.* dot product between triangle normal $\vec{n}_t$ and viewing direction (*i.e.* optical axis, that is image plane normal) $\vec{d}_j$ of the view V_j. Small angles between triangle normal and viewing directions of input cameras are then preferred. Note that in the case of parallel camera setup this criterion is not applicable.

- $C_2 := \langle \vec{e}_j, \vec{n}_t \rangle$, where $\vec{e}_j$ denotes the direction of the ray joining the triangle barycentre and the camera's position. Views where the triangle's viewing cone is larger are then preferred.

- $C_3 := \text{area}_j(t)$ the area of the projected triangle onto the view V_j. Views where the resolution of the projected triangle is larger are then preferred.

Results are presented for breakdancers and balloons (*cf*. Appendix A for more details about these sequences). The geometric model used in this study is based on the 3D reconstruction scheme presented in chapter 6.

Results of the rendered views are provided in figure 9.9 and in figure 9.7. Results in terms of best texture distribution are also provided in Figure 8.5 and Figure 8.6. It can be seen that photoconsistency based mesh texturing outperforms geometric based methods in terms of visual quality. More precisely, in figure 9.7, the frontier between the two brown walls is well-aligned using the photoconsistency criterion, while this frontier is misaligned using the geometric criterion. Besides, the floor appears to be noisy using geometric criterion, while it is much smoother using the photoconsistency criterion. Figure 8.8 shows that the difference in terms of PSNR between photoconsistency criterion and any other geometric criterion is noticeable (the average difference is at least 1dB between the two methods). Regarding the photoconsistency criterion, results also show that PSNR is higher for cameras near the frontal camera (see Figure 8.8), which is due to the large number of triangles assigned to the texture of the frontal camera. However, photoconsistency based multi-texturing algorithm is time-consuming (about 10x more) comparing to geometric methods. Therefore it turns out that photoconsistency based multi-texturing is best suited for off-line rendering. These results suggest the use of the photoconsistency based multi-texturing for transmission purposes where distortion minimization is often a key element to take into account.

8.8 Conclusion

In this chapter we presented a new multi-texturing algorithm for 3D meshes. Rather than using geometric criteria for best texture assignment, results show that the use of image based criteria, namely the photoconsistency metric, increases significantly the quality of the original views. In the next chapter we will study the impact of the proposed multi-texturing algorithm on virtual views.

(a) Original image camera 7

(b) Zoom on original image camera 7

(c) Synthesized image camera 7 using viewing cone criterion C_2

(d) Zoom on synthesized image camera 7 using viewing cone criterion C_2

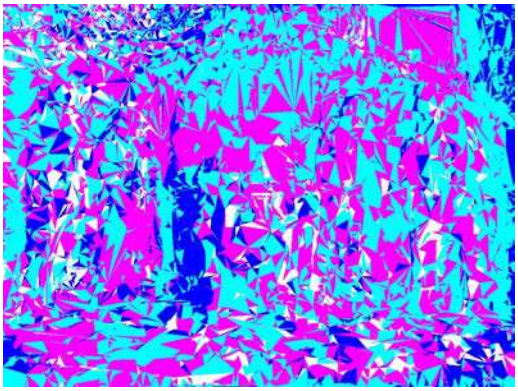
(e) Synthesized image camera 7 using photoconsistency

(f) Zoom on synthesized image camera 7 using using photoconsistency

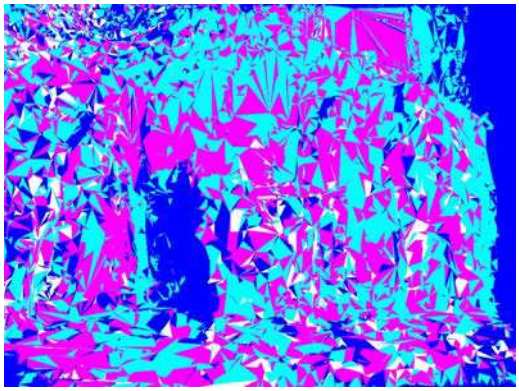
Figure 8.4: Rendered images for breakdancers sequence using viewing cone criterion vs. photoconsistency.

(a) Camera 0

(b) Camera 2

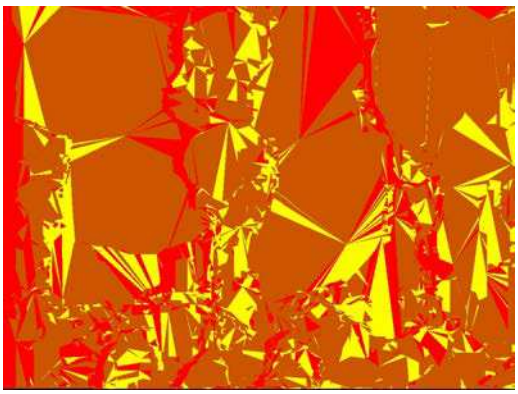


(c) Camera 4

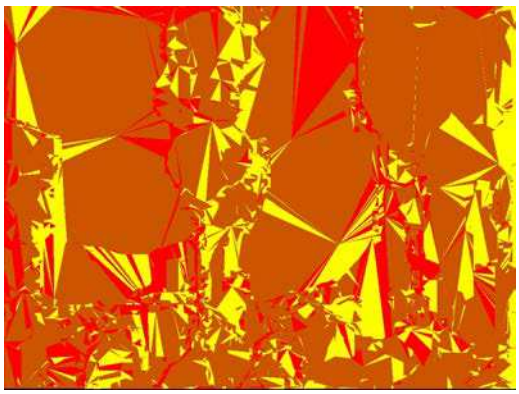


(d) Camera 6

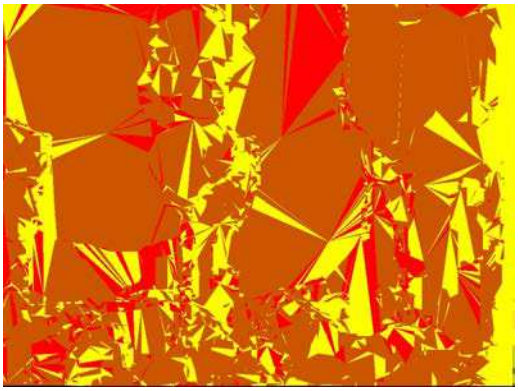
Figure 8.5: Best texture distribution for breakdancers sequence. White: camera 0, Magenta: camera 2, Cyan: camera 4, Blue: camera 6.



(a) Camera 1



(b) Camera 3



(c) Camera 5

Figure 8.6: Best texture distribution for balloons sequence. Red: camera 1, Orange: camera 3, Yellow: camera 5.

(a) Original image camera 5

(b) Zoom on original image camera 5

(c) Synthesized image camera 5 using triangle resolution criterion C_3

(d) Zoom on synthesized image camera 5 using triangle resolution criterion C_3

(e) Synthesized image camera 5 using photoconsistency

(f) Zoom on synthesized image camera 5 using photoconsistency

Figure 8.7: Rendered images for balloons sequence using resolution criteria versus photoconsistency.

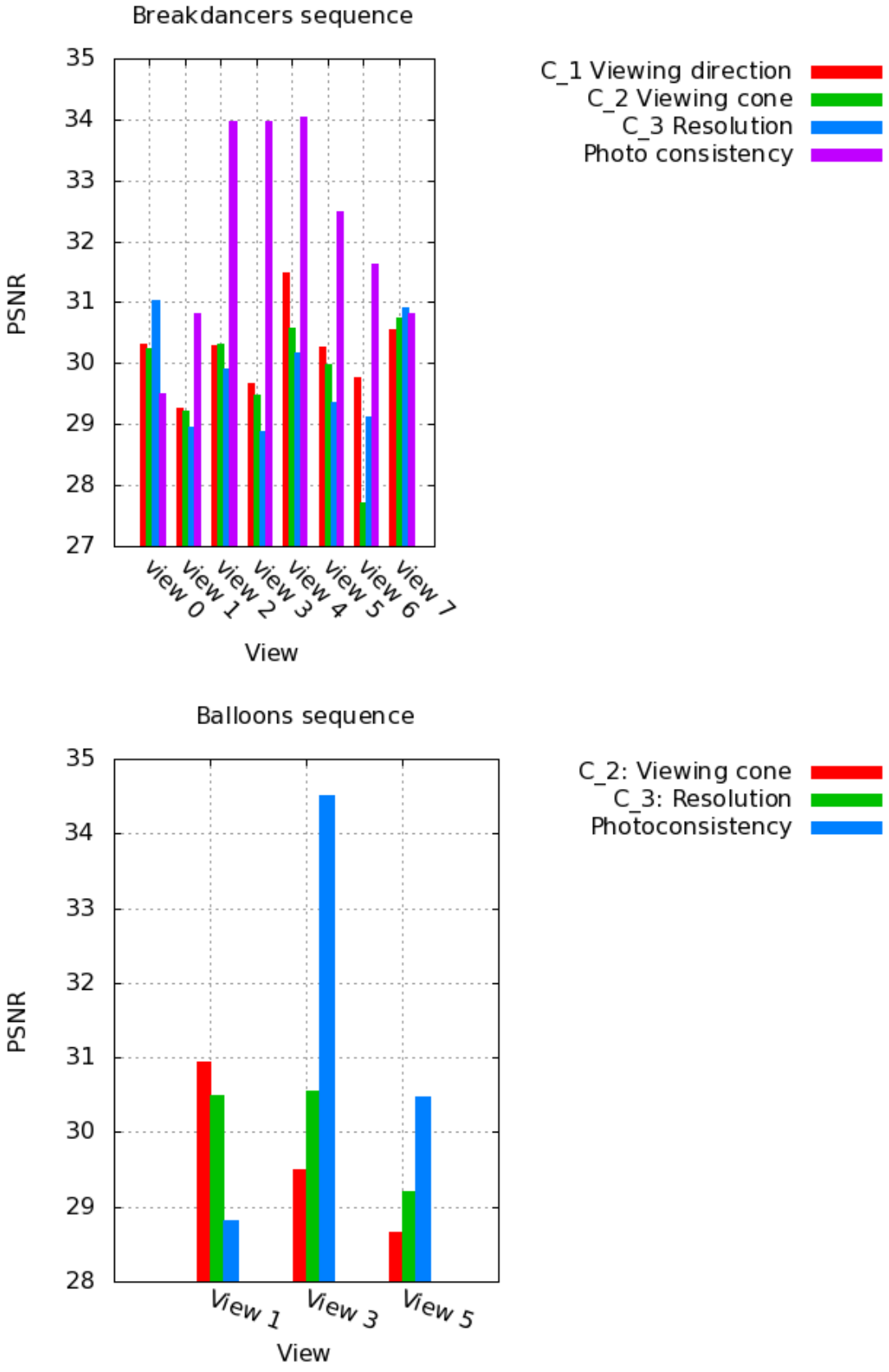


Figure 8.8: PSNR evolution through different views using different criteria for Breakdancers and Balloons sequences.

Chapter 9

View Synthesis

Contents

9.1 Introduction

In the last chapter, we presented a new multi-texturing approach for texturing 3D models. One of the most challenging problems in 3DTV and FTV applications is the ability to synthesize virtual views. In this chapter we will present how view synthesis is performed using the proposed mesh-based representation 9.2. In order to evaluate the proposed method, we will compare the performance of the proposed method with view synthesis from MVD data using the MPEG View Synthesis Reference Software (VSRS), which will be presented in section 9.3. Finally, a comparison of both approaches in terms of visual and objective quality is provided in section 9.4.

9.2 View synthesis from the proposed mesh-based representation

Novel view synthesis using 3D models is straightforward thanks to the graphics library OpenGL. OpenGL allows perspective correct texture mapping and scene rendering given camera's parameters. However, some issues related to OpenGL camera's specification and image sampling during rasterization are challenging. In this section, we will show how OpenGL cameras can be specified using pinhole camera model which is widely used in the computer vision community and how image is formed from synthetic data. First we will review the pinhole camera model presented in section 9.2.1. Second we will see how this camera model can be used in OpenGL for rendering 3D models 9.2.2.

9.2.1 Camera and projection models

The camera model that is widely used in computer vision is the pinhole camera model. This model denotes a transformation from 3D-world coordinate system $\mathcal{R}_w$ onto a 2D-image coordinate system $\mathcal{R}_I$. This transformation can be decomposed into the following steps (see Figure 9.1).

World to camera coordinate system: The relationship between on object point $M_W(X_W, Y_W, Z_w)_{\mathcal{R}_W}$ in world coordinate system and a 3D point $M_C(X_C, Y_C, Z_C)_{\mathcal{R}_C}$ in camera coordinate system can be expressed as the following:

$$\begin{pmatrix} X_C \\ Y_C \\ Z_C \\ 1 \end{pmatrix} = \begin{pmatrix} & R & & -R.T \\ 0 & 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{pmatrix} \tag{9.1}$$

Where R is the 3×3 rotation matrix from world to camera coordinate system (see Figure 9.3), and T is a 3×1 position vector of the camera C_i in world coordinate system $\mathcal{R}_W$. The parameters describing rotation and translation of the world coordinate system into camera coordinate system are referred to as extrinsic parameters and the matrix $M_{ext} = (R| - RT)$ is called an extrinsic matrix.

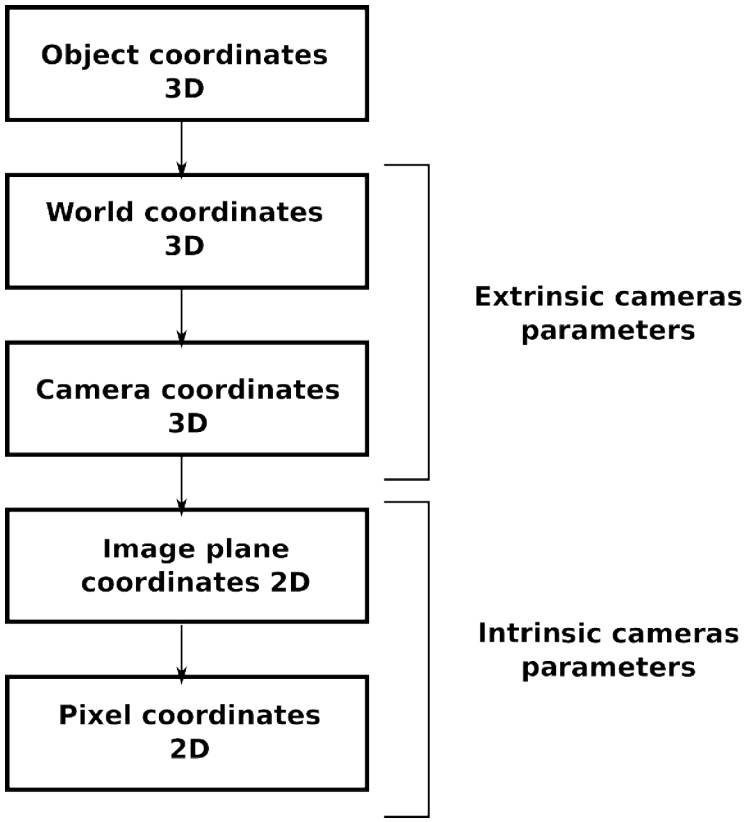


Figure 9.1: World to image coordinate system conversion.

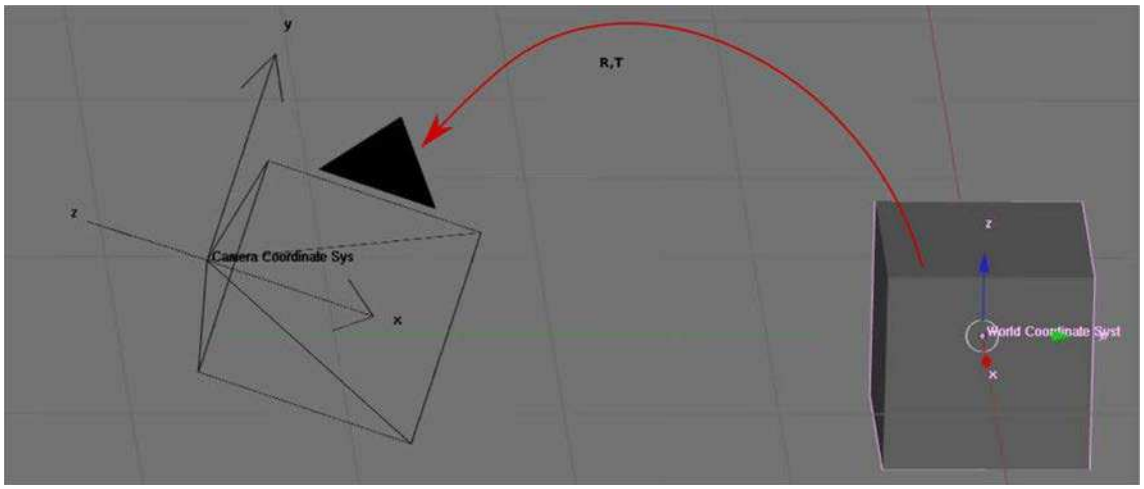


Figure 9.2: World to camera coordinate system.

Camera to image plane coordinate system: A perspective projection of a 3D-camera point $M_C(X_C, Y_C, Z_C)_{\mathcal{R}_C}$ to a 2D point $m_I(x_I, y_I)_{\mathcal{R}_I}$ on image plane can be described using only the camera focal length in milliliters:

$$x_I = f\frac{X_C}{Z_C} \qquad y_I = f\frac{Y_C}{Z_C} \tag{9.2}$$

Image plane coordinate system to pixels: Until now, we have the expression of a point on the image plane in physical coordinates (in millimeters). Our goal is

to express this point in terms of pixels. We can do this by scaling. We multiply (x_I, y_I) coordinates by the number of pixels per millimeter in the x and y direction. These scale factors are denoted s_x, s_y. The transformation gives positions in terms of pixel. However, it does not correspond to pixel indices used in an image, whose origin is the top-left corner. We need therefore a translation that gives coordinates of the principal point in pixels. Besides, pixels can be skewed, the image plane corresponds thus to a parallelogram rather than a rectangle. When the pixels are not manufactured to have a 90 degree angle, the skew parameter S_{uv} is non zero. We can integrate these transformations and the skew parameter and write the pixel (u, v) as:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \equiv \begin{pmatrix} f_x & S_{uv} & c_u \\ 0 & f_y & c_v \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_I \\ y_I \\ 1 \end{pmatrix} \tag{9.3}$$

where, $f_x = f \cdot s_x$ and $f_y = f \cdot s_y$. c_u and c_v denote the coordinates of the principal point in pixels.

Equations 9.2 and 9.3 encapsulate the inner camera's parameters also called intrinsic matrix, that is the matrix K expressed as the following:

$$K = \begin{pmatrix} f_x & S_{uv} & c_u \\ 0 & f_y & c_v \\ 0 & 0 & 1 \end{pmatrix} \tag{9.4}$$

$$= M_{int} \tag{9.5}$$

The overall transformation from 3D World coordinate system to image pixels can thus be written in a linear form when expressed in homogeneous coordinates:

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} \equiv M_{int} * M_{ext} \begin{pmatrix} X_W \\ Y_W \\ Z_W \\ 1 \end{pmatrix} \tag{9.6}$$

where $\equiv$ denotes a multiplication by a scalar $s = \frac{1}{Z_c}$ with $Z_c \neq 0$.

9.2.2 OpenGL transformations

OpenGL transformations consider synthetic data that describe the scene. Such data can be described using vertex positions, normal vectors, *etc*. In order to be displayed on the screen, these geometric data are transformed via Vertex Operation and and Primitive Assembly operations (see Figure 9.3). To specify viewing, modeling, and projection transformations, the user usually calls several OpenGL routines that are expressed as matrix operations internally. This matrix is then multiplied by the coordinates of each vertex v in the scene.

9.2.2.1 ModelView Matrix

"As far as OpenGL is concerned, there is no camera. More specifically, the camera is always located at the eye space coordinate (0., 0., 0.). To give the appearance

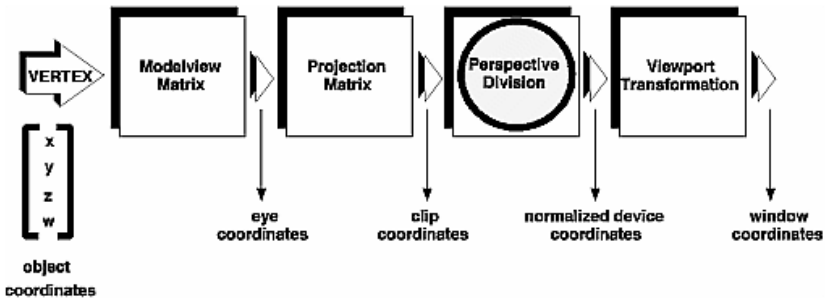


Figure 9.3: OpenGL stages of vertex transformations (image from [Shreiner et al., 2004])

of moving the camera, your OpenGL application must move the scene with the inverse of the camera transformation."[1] It turns out that if one wants to view the scene from a particular point of view, he should apply the inverse transformation to the scene using OpenGL matrix `GL_MODELVIEW`. The `gluLookAt()` utility routine can be used to fill this matrix. It takes three sets of arguments:

1. the location of the viewpoint.
2. the reference point toward which the camera is aimed;
3. which direction is up, also called view-up vector (see Figure 9.4 for an example of call to this routine).

Using the notation introduced in section 9.2.1 regarding extrinsic parameters, the modelview matrix can be updated using the following:

```
glMatrixMode(GL_MODELVIEW);
glLoadIdentity();
GLdouble R_11,R_12,R_13;
GLdouble R_21,R_22,R_23;
GLdouble R_31,R_32,R_33;
GLdouble T_x,T_y,T_z;
gluLookAt(T_x,T_y,T_z,
T_x+R_13,
T_y+R_23,
T_z+R_33,
R_12,R_22,R_32);
```

9.2.2.2 Projection Matrix

Projection in OpenGL is done by specifying a viewing volume for the projection. The viewing volume is called a frustum which can be defined as a pyramid whose top has been cut off by a plane parallel to its base (see Figure 9.5).

Specifying the frustum can be done with a call to `glFrustum()` using matrix K of equation 9.3 as the following:

[1] Quote from OpengGL FAQ http://www.opengl.org/archives/resources/faq/technical/viewing.htm

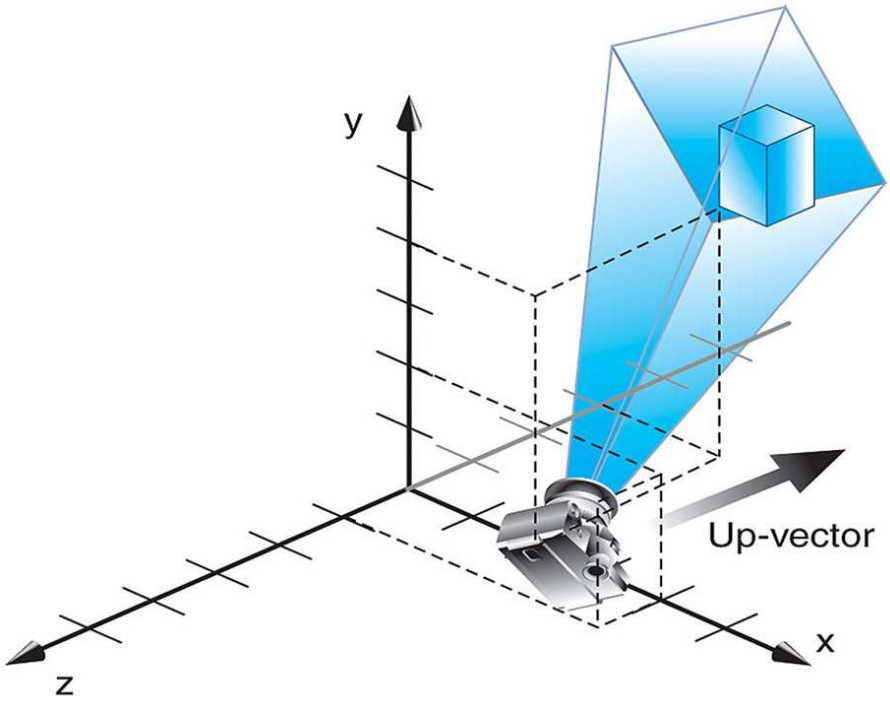


Figure 9.4: Call to glulookat$(4.0,2.0,1.0,2.0,4.0,-3.0,2.0,2.0,-1.0)$, the camera position is at $(4,2,1)$, the reference point is at $(2,4,-3)$ and the view up vector is at $(2,2,-1)$.

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
GLdouble f_x, f_y,c_u,c_v;
GLdouble left,right,bottom,top,near,far;
far=Z_far;
near= Z_near;
left =-c_u*near/f_x;
right= (width-c_u)*near/f_x;
top = c_v*near/f_y;
bottom= (c_v-height)*near/f_y;
glFrustum(left,right,bottom,top,near,far);
```

where Z_{near} and Z_{far} is the range of depth values specified in World coordinate system. `width` and `height` denote the width and the height of the captured image.

9.2.2.3 Perspective Division

After projection, the perspective division is applied by dividing coordinate values by w which is a value used to scale the x, y and z coordinates depending on the inverse of its distance from the camera. This produces normalized device coordinates.

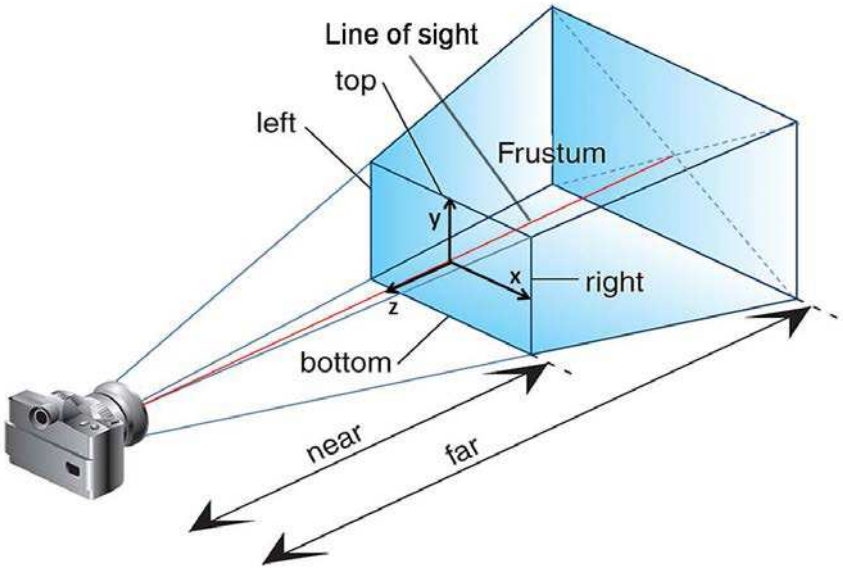


Figure 9.5: Perspective Viewing Volume Specified by glFrustum.

9.2.2.4 Viewport transform

Finally, the viewport transformation determines how the scene should be mapped on the screen. It indicates the size of the available screen area used during rendering. In our experiments, the viewport was set empirically as the following `glViewport(0, -1, nbcol+1, nbrow+1)`.

9.3 View synthesis from MVD: VSRS

Novel view synthesis consists in synthesizing a virtual view of scene given a scene representation. In this section, we will present how novel view synthesis using image based techniques. To this end, VSRS2.0 was introduced into MPEG as a reference tool for intermediate view synthesis given MVD data. Let's assume we have a left view LV and right view RV. View synthesis using VSRS2.0 consists in synthesizing an intermediate view (virtual view) denoted as IV using the steps depicted in the following block diagram. These steps can be summarized as the following:

Regarding texture images, each pixel of the reference views is mapped onto the intermediate view IV. If the cameras used during capturing and the camera of the intermediate view IV are parallel, the image of the intermediate view IV is generated using horizontal pixel shift. If the cameras are not parallel, 3D warping is performed [Tanimoto et al., 2007].

Depth Mapping: First, the depth maps of the view LV and RV are mapped to the intermediate view IV. The result is two depth maps that are shifted to the intermediate view IV. Small holes resulting from this shifting are filled using a median filter.

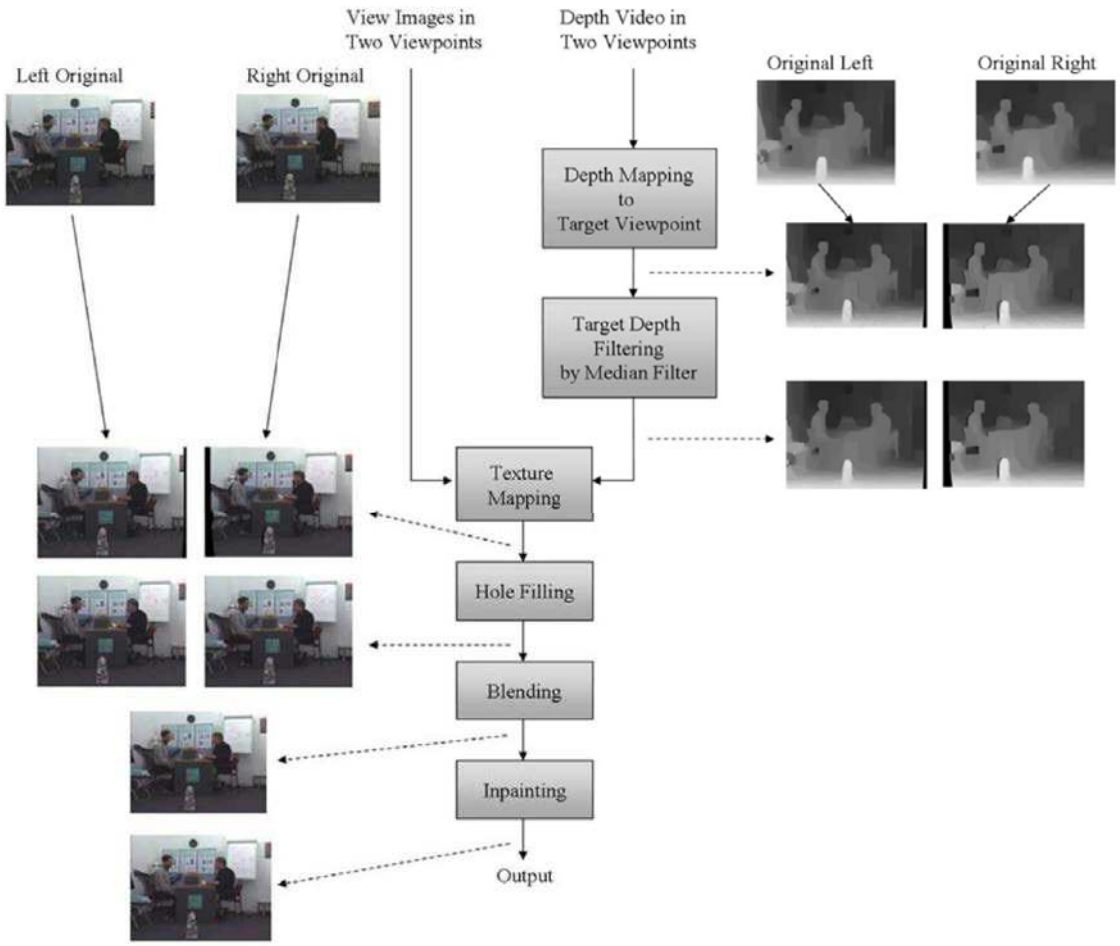


Figure 9.6: View synthesis in VSRS2.0 (image from [Tanimoto and Suzuki, 2009].)

Texture Mapping: The left image of the view LV and right image of the view RV are then mapped into the target viewpoint using the mapped depth images.

Hole filling: A consequence of this texture mapping is that some areas do not exist in the reference images. Hence, some areas are newly exposed. These areas also called holes, are detected using the mapped depth images. Holes in the image that were generated from the left view are filled using the image from the right view RV. Holes in the images generated from right view are filled using the image from the left view LV.

Image blending: The mapped texture images are then blended according to the ratio of left and right distances. Remaining hole areas are then filled by inpainting.

9.4 Results

In this section we will present some results of our mesh-based view synthesis method. We propose the following synthesis method for breakdancers and balloons sequences (See Appendix A). For breakdancers sequence, we propose to

synthesize the views 1,3,5 given a geometric model generated using depth maps of views 0,2,4 and 6 and texture images of views 0,2, 4 and 6. For balloons sequence, we propose to generate the novel views 2,4 given a geometric model that was generated from the views 1, 3 and 5 and texture images of views 1, 3 and 5. These results are presented using the proposed space carving algorithm for depth maps fusion presented in chapter 6 and the multi-texturing framework presented in chapter 8. We also propose to compare the performance of the proposed view synthesis method with MPEG-VSRS software version 3.5.

Results are provided in 9.7 and in 9.9. Besides, results in terms of PSNR are also provided in tables 9.1 and 9.2.

Regarding balloons sequence, some artifacts especially around depth discontinuities are noticeable for both image and mesh based approaches. However, results in terms of PSNR 9.2 show that image-based view synthesis method outperforms mesh based view synthesis (at least a difference of 3dBs).

Regarding breakdancers sequence, the same artifacts around depth discontinuities are noticeable using a geometric model. Such artifacts are lessened using an image based approach. A zoom on visual results particularly in image 9.8d shows that the dancer's nose is not perfectly reconstructed, whilst a smooth transition between adjacent textures is not necessarily guaranteed in image 9.8c. Results in terms of PSNR show that mesh based view synthesis provides, on average, slightly better results than image-based method.

View Synthesis Method	PSNR (dB)		
VSRS	Camera 1	Camera 3	Camera 5
	28.58	31.81	32.04
Proposed Model	Camera 1	Camera 3	Camera 5
	30.41	31.65	31.17

Table 9.1: Distortion in terms of PSNR between the synthesized and original view using VSRS and the proposed model *breakdancers* sequence.

View Synthesis Method	PSNR (dB)	
VSRS	Camera 2	Camera 4
	35.53	33.33
Proposed Model	Camera 2	Camera 4
	29.52	30.57

Table 9.2: Distortion in terms of PSNR between the synthesized and original view using VSRS and the proposed model *balloons* sequence.

9.5 Conclusion

In this chapter we presented two methods for novel view synthesis. The first method is a mesh based view synthesis method. The second one is image based

(VSRS) and relies on mapping of depth images onto the novel view. We also compare the results of the proposed model based method vs. VSRS method. Results show that the image based method outperforms the proposed mesh based method in terms of objective and visual quality. However, up until now, evaluation of the distortion is not constrained by the bitrate. In chapter 11 we will see how view synthesis can be incorporated in a compression framework where both bitrate and distortion are evaluated.

(a) Synthesized view 1 VSRS

(b) Synthesized view 1 proposed model

(c) synthesized view 3 VSRS

(d) synthesized view 3 proposed model

(e) synthesized view 5 VSRS

(f) synthesized view 5 proposed model

Figure 9.7: Virtual view synthesis for breakdancers sequence using the proposed geometric model and texture mapping algorithm vs. VSRS software.

(a) zoom synthesized view 1 VSRS

(b) zoom synthesized view 1 proposed model

(c) synthesized view 3 VSRS

(d) zoom synthesized view 3 proposed model

(e) synthesized view 5 VSRS

(f) zoom synthesized view 5 proposed model

Figure 9.8: Virtual view synthesis zoom for breakdancers sequence using the proposed geometric model and texture mapping algorithm vs. VSRS software.

(a) synthesized view 2 VSRS

(b) synthesized view 2 proposed model

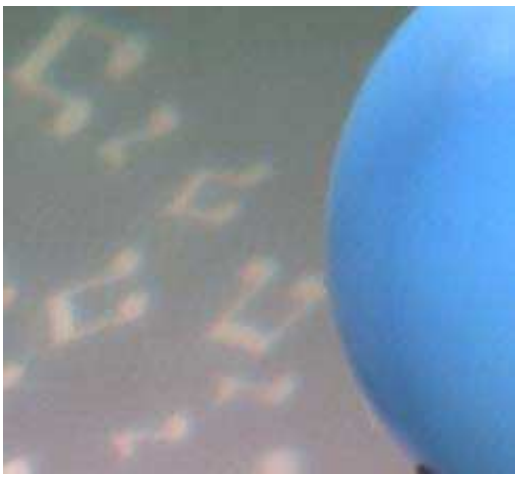
(c) synthesized view 4 VSRS

(d) synthesized view 4 proposed model

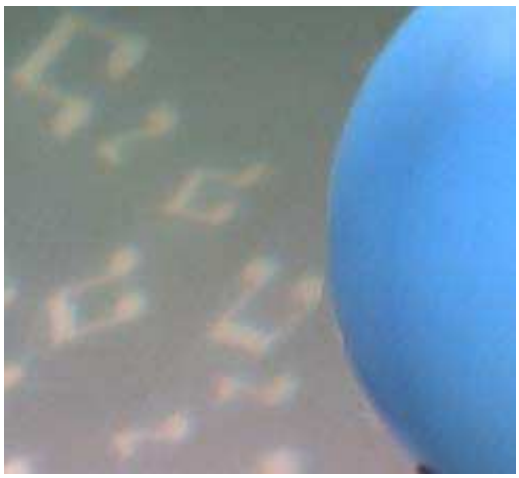
Figure 9.9: Virtual view synthesis for balloons sequence using the proposed geometric model and texture mapping algorithm vs. VSRS software.

(a) zoom synthesized view 2 VSRS

(b) zoom synthesized view 2 proposed model



(c) zoom synthesized view 4 VSRS



(d) zoom synthesized view 4 proposed model

Figure 9.10: Virtual view synthesis for balloons sequence using the proposed geometric model and texture mapping algorithm vs. VSRS software.

Part IV

Transmission

Chapter 10

Mesh Compression: a State of the Art

Contents

10.1 Introduction

In the last chapters, we defined a new representation of the scene geometry based on triangular mesh. We need therefore to transmit this representation. This chapter surveys existing methods for compression of static meshes. Section 10.2 reviews basics on meshes. Then section 10.3 presents a first group of methods designed to encode the mesh geometry into a single bit stream. Section 10.4 presents progressive methods. Finally we will present current MPEG mesh compression standard in section 10.5.

10.2 Basics on meshes

Definition 10.2.1 *Mesh*
A mesh $\mathcal{M}$ is a couple $\mathcal{M} = (\mathcal{C}, \mathcal{G})$, where $\mathcal{C}$ represents the connectivity information, i.e. *the set of faces, edges and vertices, and $\mathcal{G}$ is the geometry information,* i.e. *3D vertices coordinates. From here on, a face will refer to a triangle. Therefore we will restrict this study to triangular meshes only.*

A mesh is said to be a manifold mesh if every point of this mesh has a neighbourhood homeomorphic to a disc.

Definition 10.2.2 *Valence and Degree*

- *The valence of a vertex is the number of edges that are incident to that vertex.*
- *The degree of a face is the number of edges that are incident to that face.*

Definition 10.2.3 *Meshes that are homeomorphic to a sphere,* i.e. *meshes without holes and boundaries, are called simple meshes.*

Definition 10.2.4 *Considering mesh connectivity information with $v = |V|$ vertices, $e = |E|$ edges, and $f = |F|$, the Euler equation gives the following relationship:*

$$v - e + f = \chi \tag{10.1}$$

Moreover, for simple meshes,

$$\chi = 2$$

Thus,

$$v - e + f = 2 \tag{10.2}$$

Mesh compression techniques can be divided into two categories: single rate encoder methods and progressive encoder methods [Alliez and Gotsman, 2003, Peng et al., 2005, Ozaktas and Onural, 2007, Smolic et al., 2007]. In the following sections we propose to survey some exiting methods for both strategies.

10.3 Single rate encoders

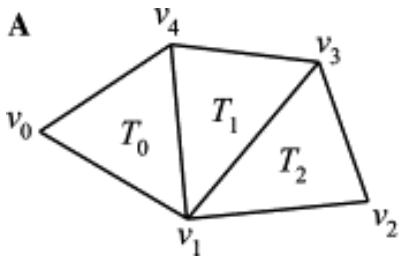
Single rate encoders encode the mesh into a single bit stream that contains both connectivity and geometry information of the input mesh. The decoder reads the bit stream and reconstructs all faces and vertex positions.

10.3.1 Connectivity coding

In this section we will present existing methods for connectivity coding. As the connectivity is an intrinsic property of the mesh, it is usually encoded losselessly.

10.3.1.1 Indexed face set

A triangular mesh in the Virtual Reality Modeling Langage (VRML) ASCII format is represented with an indexed face set that consists of two arrays. The first array contains the geometry information, *i.e.* vertices coordinates. The second array contains faces. Each face is defined by indices corresponding to the vertex array. Figure 10.1) describes a straightforward approach for mesh representation. However, no compression is involved in this scheme. Actually, each vertex is indexed several times by its adjacent triangles.



B

Index	Face
0	(0,1,4)
1	(1,3,4)
2	(1,2,3)

Figure 10.1: The indexed face set method: (A) an example of mesh and (B) its face array (image from [Peng et al., 2005]).

10.3.1.2 Generalized Triangle Mesh

Deering [Deering, 1995] introduced the concept of generalized triangle mesh. Triangles strips are used jointly with a vertex buffer to encode the mesh connectivity. A generalized triangle strip (see 10.2.C) is a serie of connected triangles that are arranged either in a strip fashion, where each vertex is combined with the previous two vertices to form a new triangle (*cf*. Figure 10.2.A), or in a fan fashion (*cf*. Figure 10.2.B), where each vertex forms a new triangle with previous and first vertices. The author [Deering, 1995] uses also a First-In-First-Out (FIFO) vertex buffer to store the indices of up to 16 visited vertices.

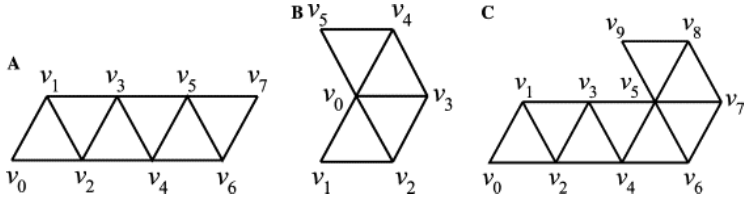
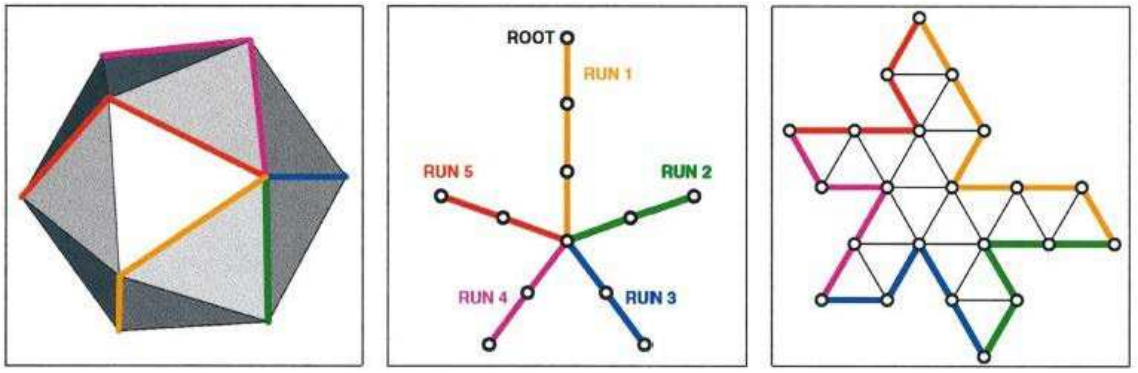


Figure 10.2: A triangle strip (A), a triangle fan (B), and the corresponding generalized triangle strip (C) (image from [Peng et al., 2005])

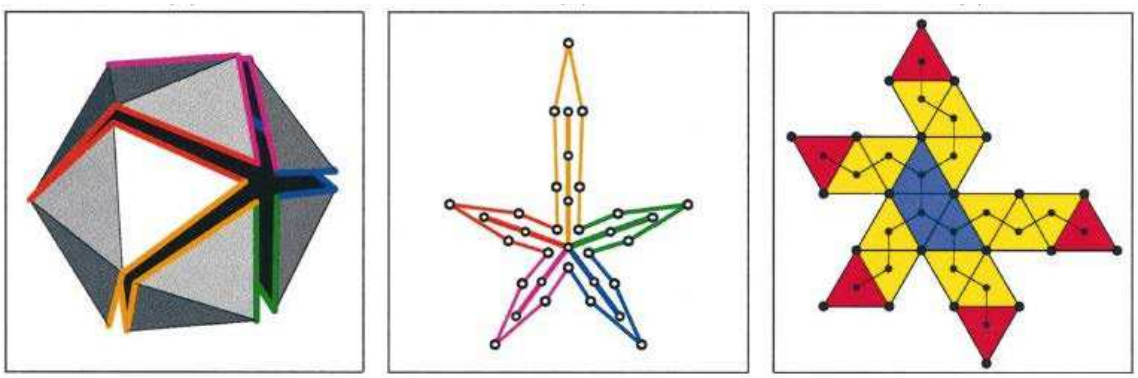
10.3.1.3 Spanning Trees

Turan [Turán, 1984] showed that the connectivity of a planar graph could be encoded with a constant number of bits/vertex using two spanning tree: a vertex spanning tree and a triangle spanning tree. Based on this idea, Taubin and Rossignac [Taubin and Rossignac, 1998] presented a topological surgery method that was later implemented within MPEG-3DMC (3D Mesh Coding) standard. In the proposed method, Taubin and Rossignac propose to cut a mesh along a set of cut edges in order to make a planar polygon. The mesh connectivity can be then described using the set of cut edges and the polygon. In the particular case of a simple mesh, any particular edge could be selected as a cut-edge.

The major drawback of the spanning trees methods is that it is not applicable directly for non manifold meshes. In this case, the input mesh should be decomposed into several manifold components which decreases drastically the efficiency of the encoding process.



(a) vertex spanning tree



(b) triangle spanning tree

Figure 10.3: Topological surgery principle

10.3.1.4 Layered decompostion

Layered decomposition [Bajaj et al., 1999] relies on mesh decomposition into several concentric layers (or levels) of vertices (*cf*. Figure 10.4). Triangle layers are constructed from pairs of adjacent vertex layers. It follows from this decomposition that the mesh connectivity can be represented by the total number of layers,

the layout of each vertex layer and the layout of triangles in each triangle layer. Ideally, a vertex layer does not intersect itself and triangle layers are generalized triangle strips, but these assumptions are not always met. The authors [Bajaj et al., 1999] then introduced additional operations for handling branching points, bubble triangles and triangle fans (*cf*. Figure 10.4).

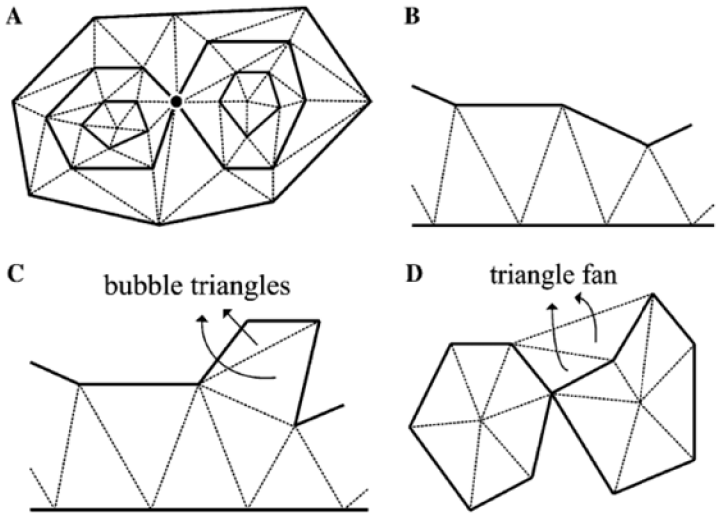


Figure 10.4: Layered decomposition method. (A): Layered vertex structure showing a different contours (thick lines) and branching point (black dot); (B): Triangle strip; (C): Bubble triangles: all vertices are belong to one contour; and (D): Triangle fan (image from [Peng et al., 2005]).

10.3.2 Geometry coding

Unlike connectivity, geometry data is coded in a lossy manner. Typically, coding a mesh geometry involves the three following steps: quantization of vertex positions, prediction of quantized positions, and entropy coding of the residuals.

10.3.2.1 Quantization

Early works in quantization in the context of geometry coding focus the attention on uniform quantization for each vertex coordinate from 8- to 16-bit resolutions [Deering, 1995, Taubin and Rossignac, 1998, Touma and Gotsman, 1998]. This quantization can be performed as follows [Isenburg and Alliez, 2002]. First the bounding box that encloses the input 3D coordinates is defined. The longest side of this bounding box is uniformly quantized with a user-defined number of bits k. Floating-point coordinates along this longest side are mapped onto an integer number between 0 and $(2k-2)$. Floating-point coordinates along the other two sides of the bounding box are mapped onto an integer range proportional to their length.

On the contrary, Chow *et al*. [Chow, 1997] proposed to partition the mesh into regions and apply an adaptive quantization resolution for each region according to the local curvature and the triangle size.

Other methods propose to quantize in the transformed domain [Karni and Gotsman, 2000].

10.3.2.2 Prediction

Different geometry predictors have been proposed in the literature: delta prediction [Chow, 1997, Deering, 1995] (*cf*. Figure 10.5), linear prediction [Taubin and Rossignac, 1998] (*cf*. Figure 10.6), or parallelogram prediction [Touma and Gotsman, 1998, Isenburg and Alliez, 2002] (*cf*. Figure 10.7).

Delta prediction The difference (or delta) between adjacent vertices tends to be small. Based on this idea, Deering [Deering, 1995] proposed to predict the current vertex P from an adjacent vertex as shown in Figure 10.5.

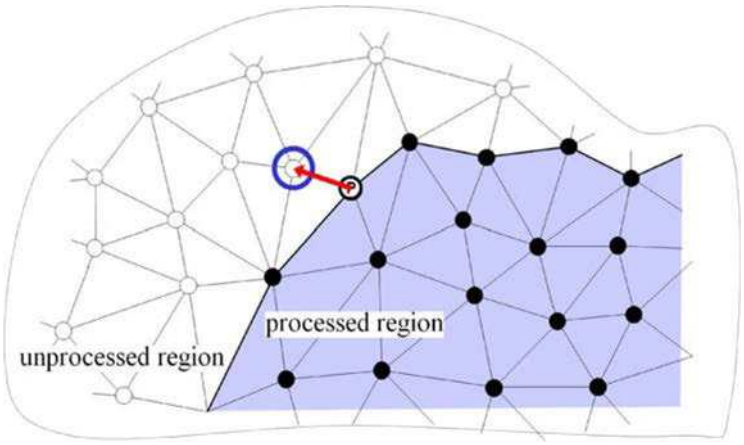


Figure 10.5: Delta prediction rule (image from [Isenburg and Alliez, 2002])

Linear prediction Taubin and Rossignac [Taubin and Rossignac, 1998] proposed to predict a vertex as a linear combination of previously visited vertices in the vertex spanning tree. The position of the vertex P to predict (*cf*. Figure 10.6) is expressed as the following:

$$P = \alpha.A + \beta.B + \gamma.C + \delta.D + \eta.E + \epsilon(P)$$

where α, β, γ, δ and η are chosen so that the error $||\epsilon(P)||^2$ is minimum.

Parallelogram prediction Touma and Gotsman [Touma and Gotsman, 1998] proposed a parallelogram rule for predicting a mesh vertex. The current vertex P (*cf*. Figure 10.7) can be predicted by forming a parallelogram between the previously coded vertices A, B and C. In other words,

$$P = A - B + C$$

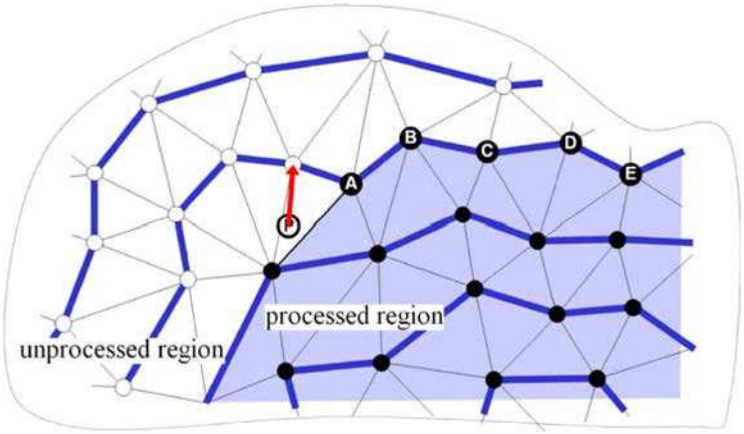


Figure 10.6: Linear prediction rule (image from [Isenburg and Alliez, 2002])

This approach was later generalized to handle polygon meshes [Isenburg and Alliez, 2002]. Exploiting polygons avoids bad predictions resulting from a crease angle between triangles.

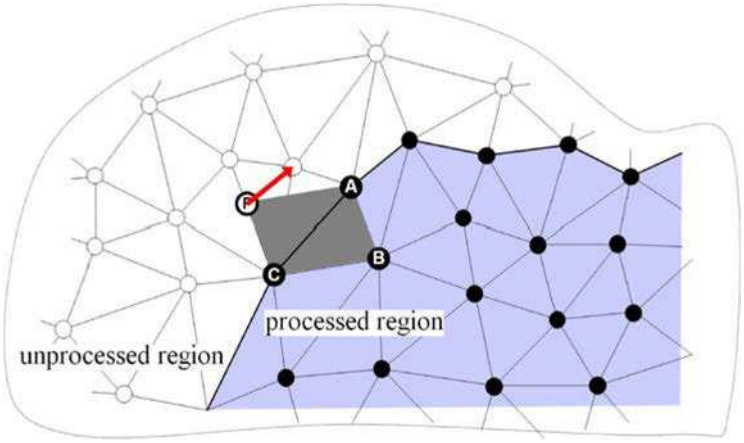


Figure 10.7: Parallelogram prediction rule (image from [Isenburg and Alliez, 2002])

In that section, we presented existing methods known as single rate techniques, *i.e.* that encode the mesh geometry and connectivity into a single bit stream. These algorithms operate by finding a new representation of the mesh (*e.g.* spanning tree) to be encoded.

10.4 Progressive encoders

Unlike single rate encoders, progressive encoders simplify the mesh using a sequence of simplification operations. Therefore the resulting mesh contains less triangles than the original mesh. The encoder then compresses the connectivity and the geometry information of this resulting mesh into a bit stream followed

by a sequence of operations that enable to recover the original mesh from simplification operations.

The decoder first reconstructs the resulting mesh, decodes the operations and then applies these operations to the resulting mesh until the original mesh is reconstructed (*cf*. Figure 10.8).

As in single rate compression scenarii, several progressive coding techniques have been proposed to simplify the connectivity of the input mesh first and treat the geometry afterwards. Such techniques are presented in 10.4.1.

On the contrary other attempts have been made in order to consider the geometry first and drive connectivity coding with geometry coding 10.4.2

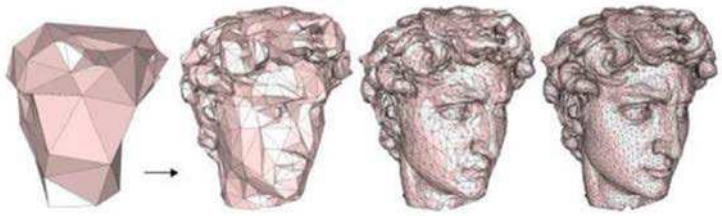


Figure 10.8: Decoding process using progressive encoders techniques (image from [Alliez and Gotsman, 2003]).

10.4.1 Connectivity-driven compression

10.4.1.1 Progressive Meshes

Hoppe [Hoppe, 1996] was the first to introduce the concept of Progressive Meshes (PM). The proposed algorithm simplifies an input mesh $\mathcal{M} = \mathcal{M}_k$ using a set of k edge collapse operations as shown in Figure 10.9. Each edge collapse $ecol_i$ transforms the mesh $\mathcal{M}_i$ to $\mathcal{M}_{i-1}$ with $i = k, k-1, ..., 1$. An edge collapse in an invertible operation. Actually, the same connectivity could be recovered if an edge collapse operation is performed followed by the corresponding vertex split operation (*cf*. Figure 10.9). It turns out that we can represent an arbitrary triangle mesh $\mathcal{M}$ with a base mesh $\mathcal{M}_0$ and a sequence of vertex split operations. Each vertex split operation $vsplit_i$ refines mesh $\mathcal{M}_{i-1}$ to $\mathcal{M}_i$, with $i = 1, 2, ..., k$.

A key point in the PM method is the determination of the edge to be collapsed without altering the quality of the decoded mesh. To this end, the edges are sorted in a priority queue according to an error metric. The collapse operation is applied to the edge with highest priority.

Despite the innovative character of the PM compression scheme, this method has several limitations. The proposed algorithm is only applicable for manifold meshes, and the mesh topology is required to remain identical during simplification and refinement operations. Besides, this method requires a huge amount of

data in order to specify vertex split operations [Peng et al., 2005]. Geometry data is encoded using delta prediction.

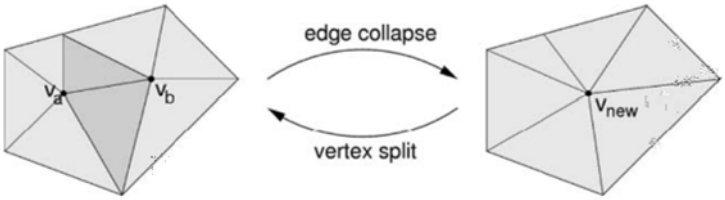


Figure 10.9: Edge collapse operation and its inverse: edge split (image from [Ozaktas and Onural, 2007])

10.4.1.2 Progressive Simplicial Complex

In order to avoid the PM's limitations, Popovic and Hoppe [Popović and Hoppe, 1997] propose Progressive Simplicial Complex (PSC) that generalizes the vertex split operation. It encodes the changes in geometry and connectivity. Representing a mesh using a PSC scheme consists in single vertex base model followed by a sequence of generalized vertex split operations (*cf*. Figure 10.10). The advantage of the PSC representation is that it can be employed to compress meshes of arbitrary topological type. However, as the PSC approach generalizes the PM methods, the PSC scheme requires more bits to encode the connectivity change around a split vertex. Similar to the PM approach, geometry data is encoded using delta prediction.

Simplex dimension	Before vertex split	After vertex split			
		Case 1	Case 2	Case 3	Case 4
0-dim	$\{a_i\}$	Undefined	Undefined	$\{i+1\}$ $\{a_i\}$	
1-dim					
2-dim					

Figure 10.10: Generalized vertex split operation (image from [Peng et al., 2005])

10.4.1.3 Progressive Forest Split

Taubin et al [Taubin et al., 1998] proposed the Progressive Forest Split compression (PFS) that was adopted in MPEG-3DMC standard [ISO/IEC, 2001]. Rather

than using vertex split operations, authors proposed to split the mesh a forest structure as shown in Figure 10.11.

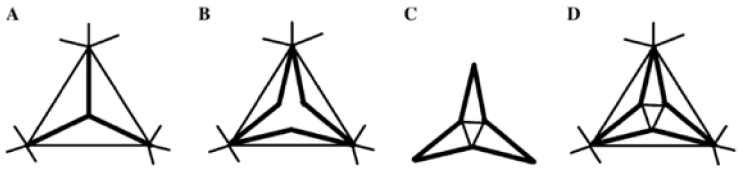


Figure 10.11: Progressive forest split operation. (A): Input mesh with forest marked with thick lines; (B): Mesh cut along forest edges; (C): Triangulation of the obtained structure; (D): Refined mesh (image from [Peng et al., 2005]).

The PFS method may double the number of triangles in a mesh. For that reason, the PFS approach achieves a higher compression ratio than vertex split approaches [Peng et al., 2005] at the expense of reduced granularity.

10.4.2 Geometry-driven encoding

10.4.2.1 Kd-tree decomposition

Gandoin and Develliers [Gandoin and Devillers, 2002] proposed a different approach in the context of progressive mesh coding. Rather than using edge operations that affect the connectivity of the input mesh, they propose to process the geometry data of the input mesh and the connectivity is guided by geometry encoding. According to this scheme, the mesh bounding box is subdivided recursively into cells until each cell contains at most a vertex mesh (*cf*. Figure 10.12 for a 2D example). Each time a subdivision occurs, a parent cell containing p vertices is decomposed into two child cells. The number of vertices in one of the child cells can be encoded using $\log_2(p+1)$ bits using an arithmetic coder [Witten et al., 1987].

Connectivity is encoded after each cell subdivision using vertex split or generalized vertex split operations. Comparing to PM approach, this approach has the advantage of implicit definition of vertex split operations. This scheme is even comparable to the single rate encoder proposed in [Touma and Gotsman, 1998]. Peng *et al*. [Peng and Kuo, 2005] propose an octree data structure instead of a kd-tree based approach. Rather than coding the vertex number in each cell, they encode the information whether each cell is empty or not and use an arithmetic encoder with context which leads to 30 to 50 % bit saving.

10.4.2.2 Spectral decomposition

Karni and Gotsman [Karni and Gotsman, 2000] proposed the extend the spectral theory, that proved its efficiency for coding audio and visual contents, to 3D

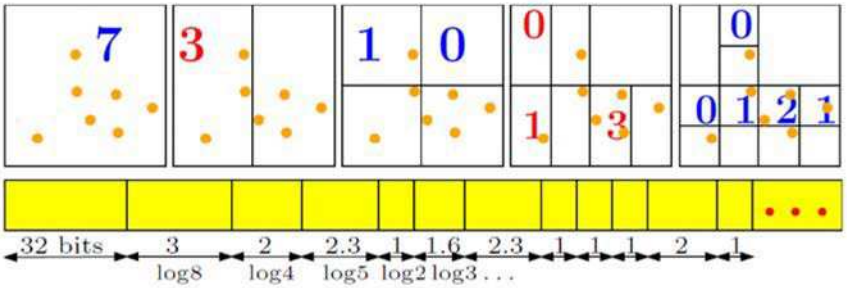


Figure 10.12: Kd-tree decomposition of mesh geometry in 2D (image from [Alliez and Gotsman, 2003]).

meshes [Taubin, 1995]. The idea is to use the mesh Laplacian defined by a $n \times n$ matrix, where n is the number of vertices, as the following:

$$L_{ij} = \begin{cases} 1 & \text{if } i = j \\ -\frac{1}{d_i} & \text{if vertices } i \text{ and } j \text{ are adjacent} \\ 0 & \text{otherwise} \end{cases} \tag{10.3}$$

Where d_i is the valence of vertex i.

Eigenvectors of L_{ij} form an orthogonal basis of $\mathbb{R}^n$. In order to encode the geometry, the encoder projects vertex coordinates onto the new basis composed of the eigenvectors of L_{ij}. The encoder quantizes this geometry in the transformed domain and truncates high frequency coefficients. The authors showed that this approach require only $\frac{1}{2}$ to $\frac{1}{3}$ of the bitrate achieved by Touma and Gotsman's algorithm [Touma and Gotsman, 1998]. However, finding the eigenvectors of $n \times n$ matrix is time consuming. Therefore, they propose to partition the mesh into segments and encode each segment individually. But, even when they use partitioning, decoding complexity is still a challenging task.

Even with their innovative solutions, progressive encoders are known to have the worst results in terms of compression efficiency comparing to single rate encoders [Mamou, 2008]. Besides, some methods introduce additional complexity to the decoder (*e.g.* [Karni and Gotsman, 2000]). However most single rate encoders assume some regularities on the input mesh (*e.g.* manifoldness). Research activities especially within MPEG-3DGC community has been conducted in this area in order to propose a mesh compression standard that satisfies both generality and low complexity at decoder side.

10.5 MPEG-SC3DMC Codec

Scalable Complexity 3D Mesh Coding (SC3DMC) is the current 3D mesh coding standard. SC3DMC compresses geometric data as well as vertex attributes (*e.g.* normals, texture coordinates). The SC3DMC chooses the best coding mode among the three following connectivity compression modes: Quantization-Based Compact Representation (QBCR), Shared Vertex Analysis (SVA) and Triangle FAN

(TFAN). The choice is based on the compression-efficiency and complexity trade-off. In this section we propose to study the structure of TFAN mesh codec.

10.5.1 Generalities about TFAN codec

TFAN codec was introduced by Mamou *et al*. [Mamou et al., 2009] with a progressive version referred to as PTFAN in [Mamou and Dehais, 2010]. The idea behind TFAN technique is to exploit a deterministic traversal of mesh vertices combined with a decompostion of mesh triangles into triangle fans.

10.5.1.1 Triangle fan

Definition 10.5.1 *[Mamou et al., 2009] A triangle fan of degree d is an ordered set of triangles $(t_j)_{j\in\{0,\ldots,d-1\}}$ defined by an ordered sequence of $d+2$ vertices $(v_0, v_1, \ldots v_{d+1})$ such that:*

$$\forall j \in \{0,\ldots,d-1\},\ t_j = \{v_0, v_{j+1}, v_{j+2}\} \tag{10.4}$$

A triangle fan has by definition the following properties (see Figure 10.13 depicting a triangle fan of degree 4):

- Two successive triangles in a triangle fan share a common edge.
- All the triangles have the same orientation.
- All the triangles share a common vertex v_0 called the center of the triangle fan.

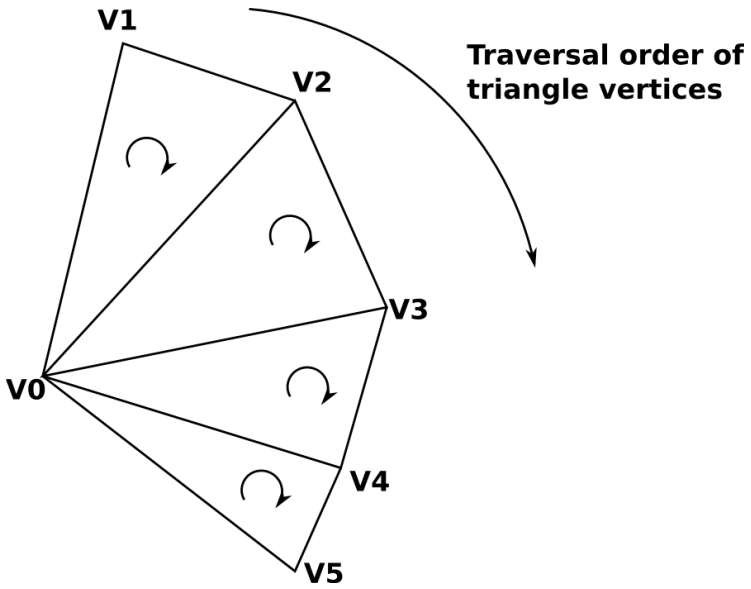


Figure 10.13: Triangle fan of degree 4.

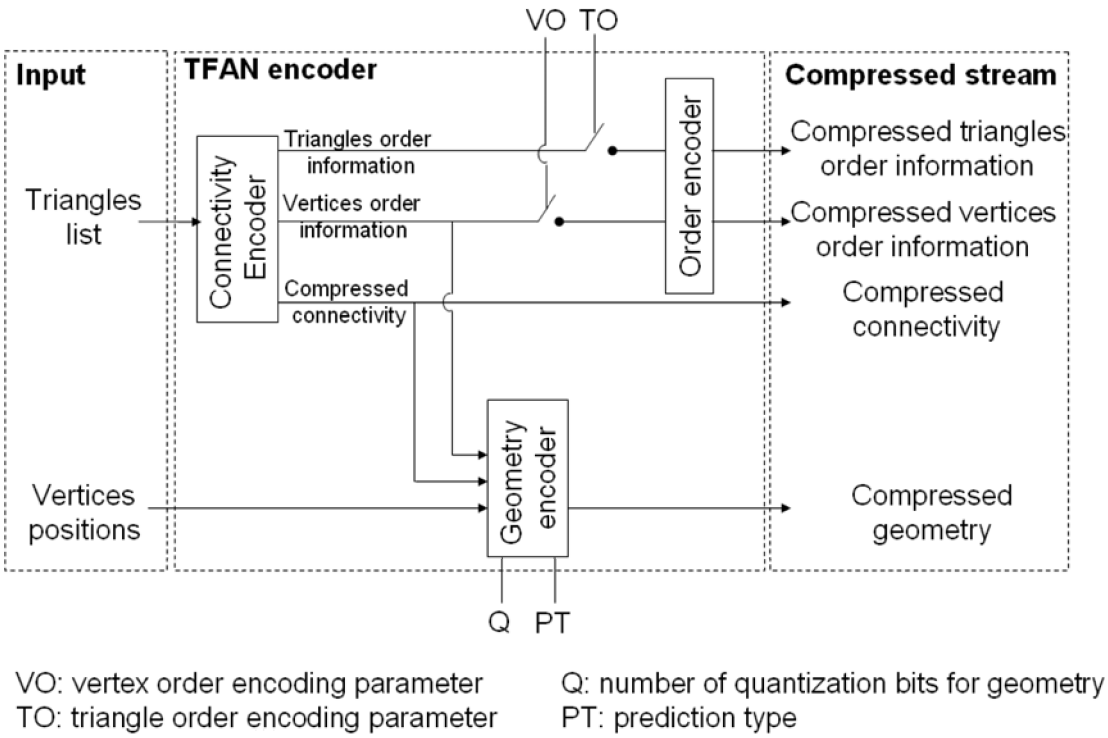


Figure 10.14: Structure of TFAN encoder (image [Mamou et al., 2008b]).

10.5.2 Coding connectivity

A triangle fan is completely described by the ordered sequence of its vertices $\{v_0, v_1, v_2, \ldots, v_{d+1}\}$. Therefore, encoding a triangle fan is equivalent to encoding $d+2$ integer indices representing its vertices. It is just needed to decompose an input mesh into a set of triangle fans (TFs).

Figure 10.15 illustrates this decomposition using an example [Mamou, 2008]. The input mesh in Figure 10.15 contains the following triangles: $t_1 = \{v_6, v_5, v_4\}$, $t_2 = \{v_1, v_9, v_2\}$, $t_3 = \{v_3, v_7, v_9\}$, $t_4 = \{v_1, v_8, v_9\}$, $t_5 = \{v_2, v_9, v_6\}$, $t_6 = \{v_9, v_7, v_6\}$ and $t_7 = \{v_6, v_7, v_5\}$.

This mesh can be decomposed into the three following TFs: $TF_1 = \{v_1, v_8, v_9, v_2\}$, $TF_2 = \{v_9, v_3, v_7, v_6, v_2\}$, $TF_3 = \{v_6, v_7, v_5, v_4\}$.

Converting these triangle fans into a list of mesh triangles is straightforward using equation 10.4. We obtain the following triangles: $t'_1 = \{v_1, v_8, v_9\}$, $t'_2 = \{v_1, v_9, v_2\}$, $t'_3 = \{v_9, v_3, v_7\}$, $t'_4 = \{v_9, v_7, v_6\}$, $t'_5 = \{v_9, v_6, v_2\}$, $t'_6 = \{v_6, v_7, v_5\}$, $t'_7 = \{v_6, v_5, v_4\}$. We obtained the same triangle list as the triangles of the initial mesh *modulo* a permutation of the mesh triangles. The traversal order of mesh triangles induced by TFAN representation is therefore different than the initial order of the mesh triangles. This re-ordering process due to TFAN representation actually allows an implicit coding of the mesh connectivity.

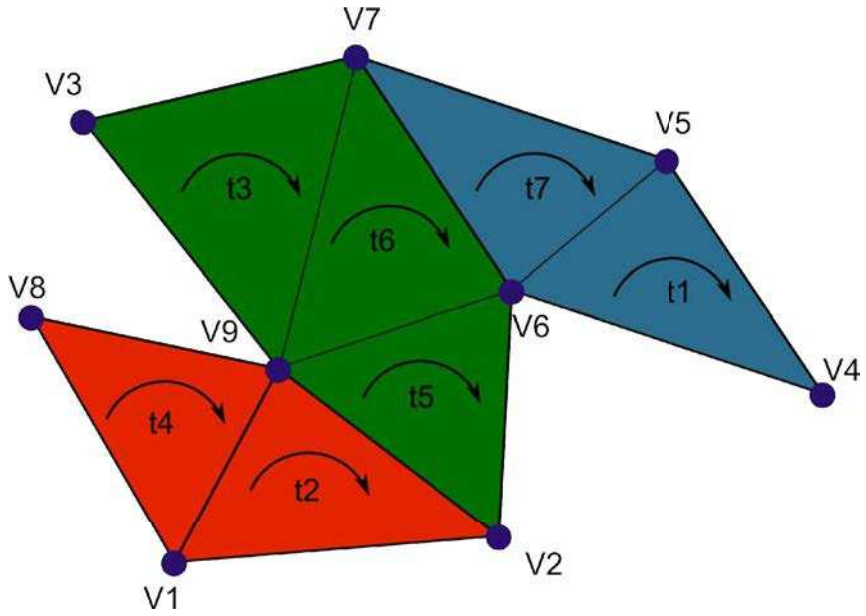


Figure 10.15: Decomposition of mesh connectivity into three triangle fans: $TF_1 = \{v_1, v_8, v_9, v_2\}$, $TF_2 = \{v_9, v_3, v_7, v_6, v_2\}$, $TF_3 = \{v_6, v_7, v_5, v_4\}$.

Using this same principle, mesh vertices can be reordered, which consequently leads to additional bit-savings. Actually, vertices of TF_1 can be renamed in the traversal order (*i.e.*, $v_1 \rightarrow v'_1$, $v_8 \rightarrow v'_2$, $v_9 \rightarrow v'_3$ and $v_2 \rightarrow v'_4$). It is now possible to encode the triangle fan TF_1 by only indicating its degree: $d_1 = 2$. The decoder in this case will reconstruct the following triangle fan composed of the two triangles: $TF_1 = \{v'_1, v'_2, v'_3, v'_4\}$.

Encoding the second triangle fan TF_2 is more complex. This triangle fan is partially composed of already traversed vertices of TF_1. In this case, having only the information about the degree $d_2 = 3$ of the triangle fan TF_2 is not sufficient to recover the already used vertices and that can not be renamed. This information will be then stored as a size information $\mathcal{S}_2$, with a bit indicating if the vertex has been already used in a previous triangle fan or not. Thus, in the case of TF_2, the new vertices are v_3, v_3 and v_6. The already encoded vertices are v_9 and v_2. The binary sequence $\mathcal{S}_2$ is then $\mathcal{S}_2 = \{1,0,0,0,1\}$, where 1 indicates that the vertex has already been encoded and 0 if not. Moreover, the indices corresponding to type 1 vertices have to be stored and transmitted to the decoder. In this example, the list of already encoded vertices are v'_3 and v'_4. Therefore the sequence $\mathcal{I}_2 = \{3,4\}$ is to be encoded.

Following the same principle, the triangle fan TF_3 can be represented as follows:

- $d_3 = 3$.
- $\mathcal{S}_3 = \{1,1,0,0\}$.
- $\mathcal{I}_3 = \{7,6\}$.

In order to preserve the initial ordering, TFAN encoder provides two options: TO (Triangle Ordering preservation), and VO (vertices ordering preservation).

10.5.3 Coding geometry

Having seen how connectivity is encoded using an example, we will now present how geometry is encoded within TFAN algorithm. First a uniform quantization is applied to the mesh vertices. The number of quantization bits is controlled by the parameter Q (*cf*. Figure 10.14). A linear prediction is then performed and the user selects one of five different prediction modes. Besides delta prediction and parallelogram prediction modes presented in section 10.3, TFAN allows to use barycenter prediction, adaptive prediction, and Fast parallelogram prediction.

In barycenter prediction scheme, a vertex is allowed to be predicted by the barycenter of its decoded neighbours. In the adaptive prediction, the vertex is predicted by either applying the barycenter or parallelogram prediction depending on the mode that minimizes the compressed stream's length. Fast parallelogram is a TFAN based prediction that exploits the TFAN decomposition generated during connectivity encoding and that is supposed to optimize the decoding complexity.

The authors [Mamou et al., 2008a] have shown the compression efficiency of TFAN approach comparing to existing mesh codecs [Touma and Gotsman, 1998]. Besides, TFAN decoding complexity outperforms previous MPEG-3DMC mesh compression standard.

10.6 Conclusion

In this chapter we reviewed existing methods for static mesh coding. The drawbacks of most of these techniques is their complexity at decoder side to handle arbitrary meshes. Current MPEG standard (TFAN) achieves satisfying results in terms of compression and decoding complexity. In next section we will incorporate this standard to encode the proposed model and to evaluate the performance of the proposed algorithms.

Chapter 11

Compression of the Proposed Model

Contents

11.1 Introduction

Having reviewed existing techniques for coding 3D models, we are now able to encode the scene geometry given the 3D model generated in 6. In order to transmit texture information, we have to represent texture signal needed to recover the input images. This is discussed in section 8. Evaluation of both the proposed model composed of this texture signal and the merged mesh is presented in section 11.3. Finally results in terms of R-D curves are presented in section 11.4.

11.2 Texture model generation

In this section we propose to generate the texture model to be transmitted. Using the proposed multi-texturing framework proposed in chapter 8. First, texture maps used during texture mapping of the input mesh are generated as a set of images 11.2.1. Second in section 11.2.2, these texture maps are aligned according to the macroblock structure used in video compression standards. Finally, for transmission purposes, unused macroblocks[1] of these texture maps are filled with DC-like prediction scheme in section 11.2.3.

11.2.1 Texture maps generation

In chapter 8, we presented a new algorithm that determines the best view for each triangle using image based criterion, *i.e.* by minimizing the L_2 error during rendering between the input images and the rendered views. We can therefore compute the set of texture maps that were used during rendering using the following simple algorithm 19:

Algorithm 19: Texture maps generation from best view assignment.

```
   //  Views traversal
1  for each view V_j do
       // Mesh triangles traversal
2      for each triangle t do
3          Determine V_i best view of triangle t
4          if i = j then
5              texture triangle t with texture corresponding to view V_i.
6          else
7              render the triangle t with a plain color.
```

An example of the generated texture maps is provided for Breakdancers sequence in Figure 11.1. In this figure, we present texture maps corresponding to views 0 and 6. Each image shows pixels used during texture mapping, unused pixels are shown in black.

[1] We recall that during texture mapping, among all the available textures, one texture is selected for each triangle. This texture is stored in as many texture images as views: each texture image T_i contains texture macroblocks coming from view V_i. Holes in T_i correspond to regions that use texture information from other views V_j. Such holes correspond to unused macroblocks in T_i

(a) camera 0 (b) camera 6

Figure 11.1: Input texture maps for views 0 and 6 for breakdancers sequence. Unused texture fragments are shown in black.

11.2.2 Macroblock-aligned Texture maps

For transmission purposes, filling unused pixels with black color is not optimal for high compression efficiency. Actually, during our experiments, we estimated the bitrate of transmitting the set of texture maps as described in 11.2.1 using MPEG.4/MVC (see table 11.1). In these experiments, we used MPEG.4/MVC on still images and only the first frame of each (video) view sequence is considered, which explains the very high bitrates obtained in table 11.1. Furthermore, in order to exploit inter-view prediction, the encoding order that we have chosen for views 0,2, 4 and 6 is $0 \rightarrow 4 \rightarrow 2 \rightarrow 6$. We chose this order in order to exploit bidirectional prediction of frame 2. this order implies that view 0 is first encoded in intra mode. View 4 is then predicted from view 0 and view 2 is predicted bidirectionally from both views 0 and 4, since both views 0 and 4 are now available. Finally, view 6 is predicted from view 4.

As table 11.1 shows, the computed bitrate is larger than when transmitting the input texture images. This can be s by the triangle structure that is not consistent with traditional video sequences, as it can be seen in Figure 11.1. At the encoding stage, the presence of signal discontinuities (black holes vs textured triangles) introduces high frequencies that are expensive to encode. Additionally, inter-texture similarities are smaller than with classical multi-view contents: the MPEG/MVC encoder can not exploit the inter-views redundancies.

In order to reduce texture bitrate, we propose to re-organize the triangle-based texture maps into macro-block based texture maps since a macroblock is the basic structure on which video compression standards operates for prediction schemes (see chapter 3 for more details). Figure 11.2 illustrates the macroblock aligned texture maps of figure 11.1. The proposed macroblock-aligned texture maps consist in simply finding the nearest macroblocks surrounding a given texture triangle.

11.2.3 Macroblock filling

The coding bitrate of the macroblock aligned texture maps presented in section 11.2.2 allows a bitrate saving up to 12%, comparing to transmitting input textures, thanks to the macroblock alignment. In another attempt to decrease the coding

(a) camera 0 (b) camera 6

Figure 11.2: Macro block aligned texture maps.

bitrate, we propose to predict each unused macroblock using MPEG.4/AVC like prediction schemes. We use DC prediction mode for its simplicity and easiness of implementation (see Figure 3.2 for more details about this prediction mode). By doing so, we constrain the current unused macroblock to be predicted using intra coding mode. It turns out that the overall bitrate of the input texture maps has slightly decreased (only 0.2% saved) comparing to macroblock-aligned texture maps. The results of this operation on texture model of Figure 11.3 is presented in Figure 11.3. In the remaining of this chapter, texture model will refer to the set of block aligned texture maps and where each unused macroblock is filled using DC-prediction scheme.

(a) camera 0 (b) camera 6

Figure 11.3: Macro block filling for efficient coding of multi-view texture maps.

11.3 Evaluation

In this section we propose to compare the performance of the proposed model composed of a 3D mesh (acquired from chapter 6) and the set of texture maps 11.2 with the reference model composed of input texture images and the associated input depth maps. This evaluation is performed on still images only. We propose

Texture model	rate tex 0 (kbits/s)	rate tex 2 (kbits/s)	rate tex 4 (kbits/s)	rate tex 6 (kbits/s)	overall rate (kbits/s)
Input textures	8186,28	6034,32	6612	6659,4	27492
original texture maps	5034,72	10046,64	9755,28	4549,08	29385,72
macro block aligned texture maps	6089,76	6066,72	6511,08	5595,36	24262,92
filled macroblocks	6142,56	6065,64	6524,52	5478,72	24211,44

Table 11.1: Evolution of texture bitrate (kbits/s) for different texture models for breakdancers sequence (volumetric resolution of geometric model 125x125x125).

to evaluate distortion by evaluating the quality of virtual view synthesis. Bit-rate is computed for both texture and depth information. We propose to evaluate the proposed model using four test sequences: breakdancers and ballet, balloons and kendo (*cf*. Appendix A for more details about these sequences). The evaluation process for the reference model and the proposed model one are detailed in the following sections 11.3.2 and 11.3.1 respectively:

11.3.1 Evaluation of reference model

The proposed evaluation method for Microsoft sequences is presented in Figure 11.4. In order to evaluate depth bitrate, we consider depth maps of views 0, 2, 4 and 6. These depth maps are then encoded using MPEG/MVC software for several quantization parameters. To measure the distortion of the reference model, appropriate image-based virtual view synthesis tools should be considered. To this end, we propose to use the MPEG/VSRS (View Synthesis Reference Software presented in chapter 9). First input depth maps and texture images corresponding to views 0, 2, 4 and 6 are encoded using MPEG.4/MVC with the quantization parameters QPs: 22, 27, 32, 37, 42 and 47. These quantization parameters are those used in MPEG common test condistions in order to evaluate MVC. Bitrate of depth and texture data is measured for each quantization parameter. For each of the encoded depth maps and texture images, virtual views corresponding to views 1, 3 and 5 are synthesized. Distortion is evaluated in terms of PSNR on Luminance component (since the human visual system is more sensitive to distortion on luminance component than chrominance components) between the synthesized views 1, 3 and 5 and original ones. Similar evaluation was performed for balloons and kendo sequences by encoding depth maps and textures corresponding to views 1, 3 and 5 and synthesizing virtual views 2 and 4.

11.3.2 Evaluation of the proposed model

The evaluation of the proposed model is presented in Figure 11.5 for Microsoft ballet and breakdancers sequences. First a merged mesh is computed from depth maps associated to views 0, 2, 4 and 6 using the proposed space carving algorithm

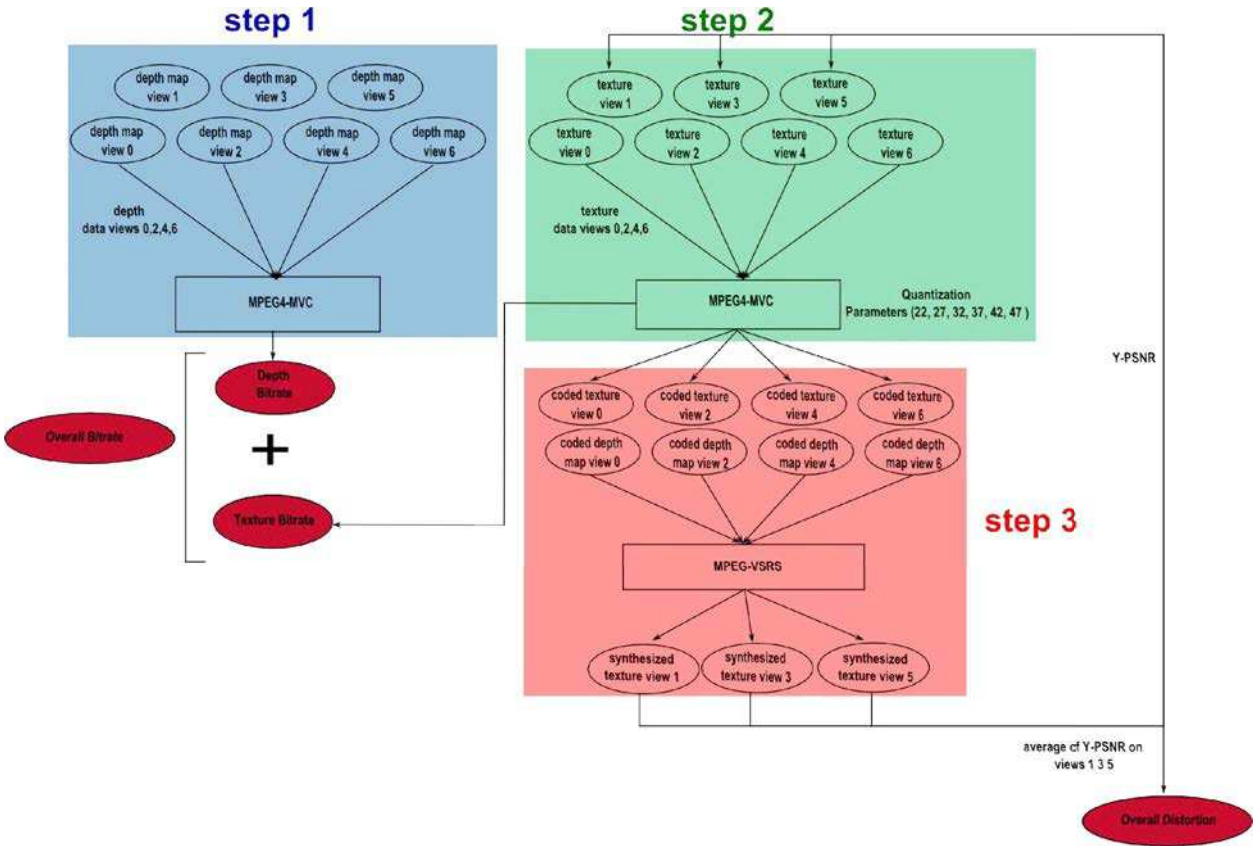


Figure 11.4: Evaluation of the reference model for Microsoft ballet and breakdancers sequences.

(*cf*. chapter 6). This merged mesh is computed for the following volumetric resolutions: 50, 125 and 250. We recall that the volumetric resolution parameter refers to the number of considered voxels in each direction. A volumetric resolution of 50 indicates that the number of voxels of the bounding grid is 50 in X, Y and Z directions. At this stage, we can evaluate the bitrate of the geometry using MPEG-TFAN codec presented in chapter 10.5. The next step is dedicated to texture modeling. Best texture assignment is determined for each triangle using the photoconsistency metric algorithm described in chapter 6. In this evaluation, best texture selection is restricted to the views 0, 2, 4 and 6. Additionally, we need to encode the information related to best view (or best texture) that is assigned for each triangle. The best texture information is provided by the photoconsistency-based multi-texturing algorithm presented in chapter 8.

The bitrate of best texture information is evaluated by computing the occurrence of each view and the overall entropy using equation 3.2. Consequently, texture model can be generated for views 0, 2, 4 and 6 as described in section 11.2. This texture model is then encoded using MPEG.4/MVC for varying quantization parameters QPs. The considered QPs are the same as those used during evaluation of the reference model. For each decoded texture model, virtual views 1, 3 and 5 are generated using the merged mesh as described in chapter 9. As input textures corresponding to views 1, 3 and 5 are available we can compute distortion between the rendered views and the original ones using the PSNR metric. Similarly, we evaluate balloons and kendo sequences by considering depth

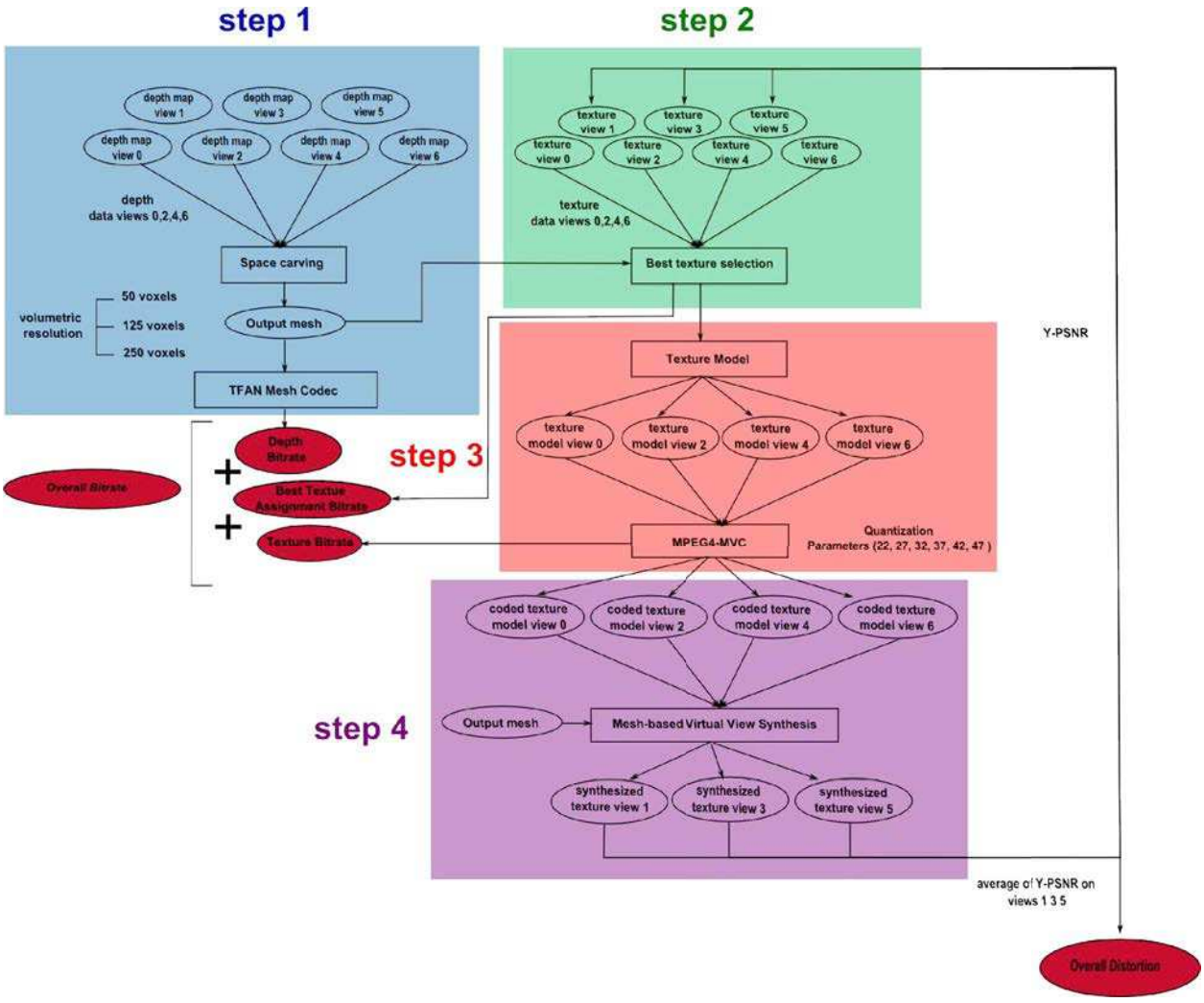


Figure 11.5: Evaluation of the proposed model for Microsoft ballet and breakdancers sequences.

maps associated to views 1, 3 and 5, and synthesizing views 2 and 4.

11.4 Results

Results in terms of Rate/Distortion curves are shown in Figures 11.6 and 11.7 for Microsoft breakdancers and ballet sequences respectively. And in figures 11.8 and 11.9 for balloons and kendo sequences. These results show that MPEG.4/MVC outperforms the proposed model in terms of bitrate/distortion trade-off. Except in case of breackdancers, where better reconstructed intermediate views quality can be achieved at high bit rate. Actually, a huge amount of geometry data is required in order to achieve the same quality especially for breakdancers and ballet sequences. In other words, coding the set of depth maps is less expansive than coding the geometry model. The quality of the estimated depth maps plays a key role in the obtained bitrate. For each introduced outlier in the estimated depth map, several triangles are needed to represent the outlier, which increases significantly the geometry bitrate.

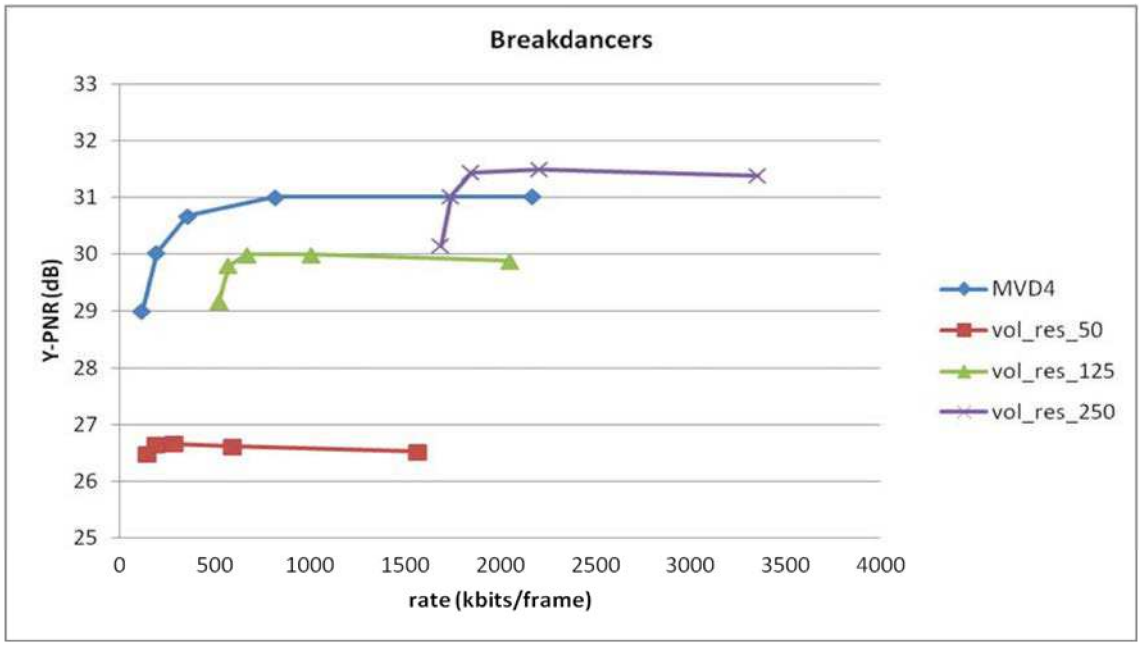


Figure 11.6: Rate/distortion curves for varying volumetric resolutions vs. reference model for breakdancers sequence.

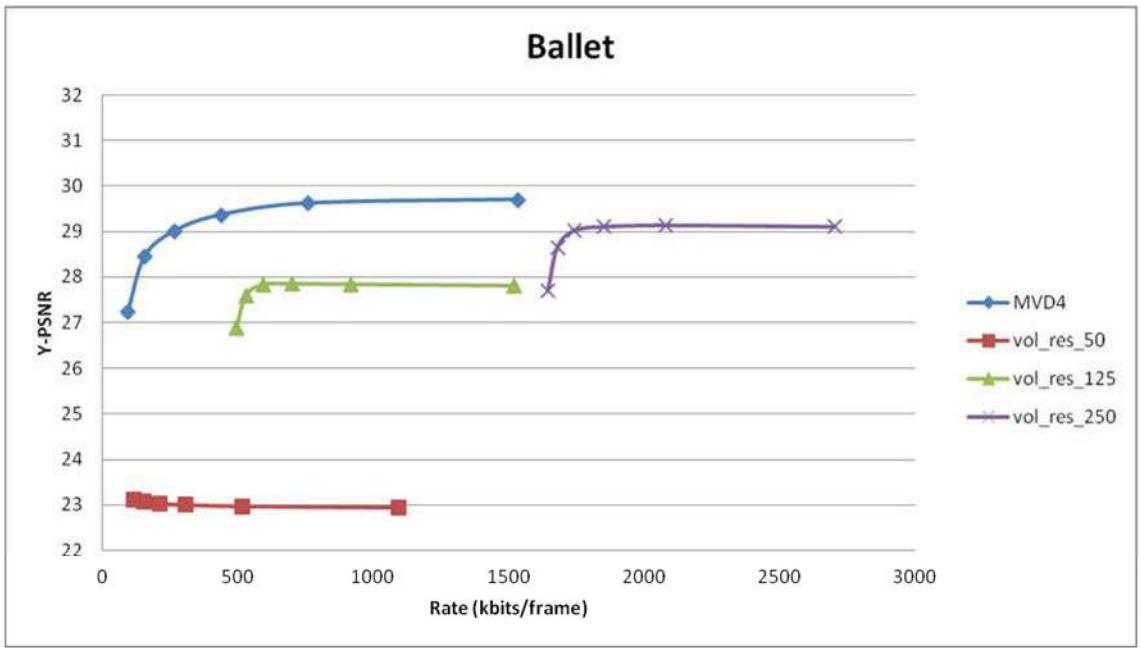


Figure 11.7: Rate/distortion curves for varying volumetric resolutions vs. reference model for ballet sequence.

11.5 Conclusion

In this chapter, we evaluated the compression of the proposed model composed of the geometry and the texture models versus the reference model composed of overlapping texture and depth images. Results in terms of Rate-Distortion performance is not yet optimal. These results show that a tremendous amount of data should be transmitted in order to achieve the visual quality of video com-

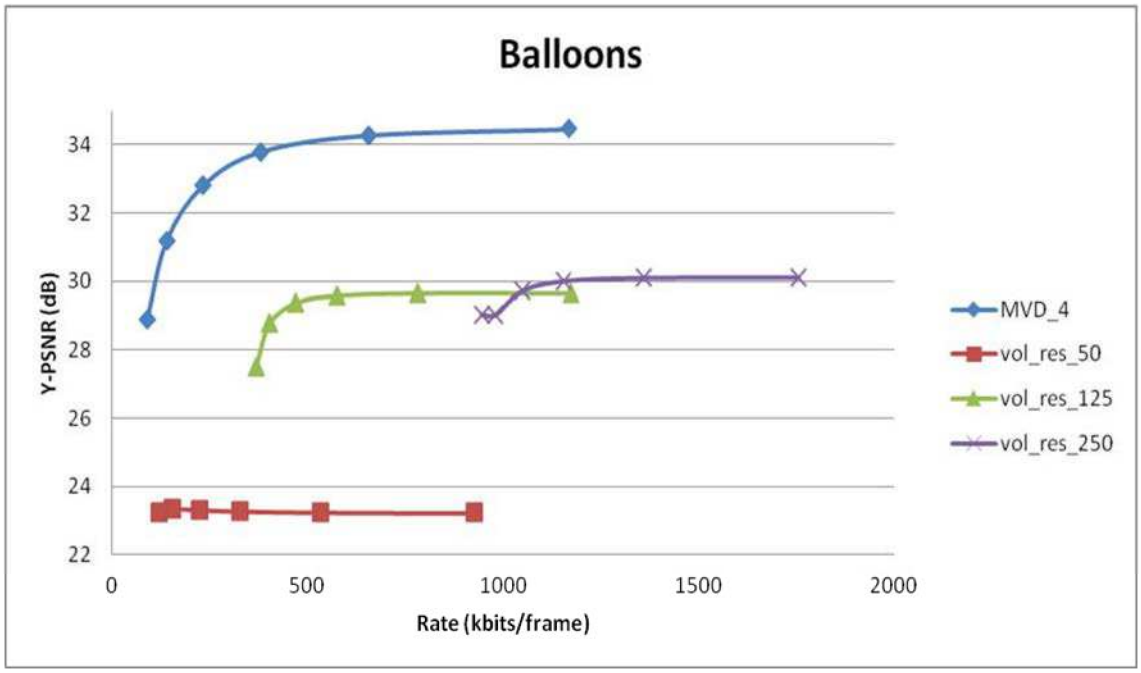


Figure 11.8: Rate/distortion curves for varying volumetric resolutions vs. reference model for balloons sequence.

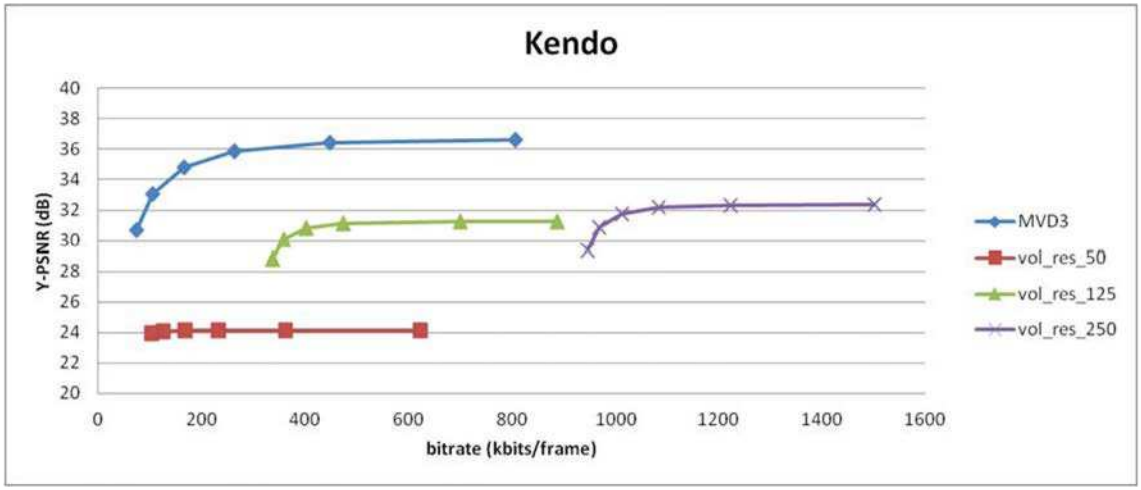


Figure 11.9: Rate/distortion curves for varying volumetric resolutions vs. reference model for kendo sequence.

pression standards if high quality depth maps are available. As perspectives, we plan to investigate how bitrate can be reduced without altering the overall quality by exploiting mesh-based temporal redundancies and exploring more image based mesh compression techniques.

Chapter 12

Conclusion

In this thesis we addressed the problem of scene representation from Multi-view video plus depth sequences. We considered a general framework where calibrated cameras are not necessarily parallel. We also assumed the depth maps to be reliable. Our contributions are as the following.

Space carving for depth maps fusion:

We first presented a new algorithm for merging the available depth maps into a single mesh (chapter 6). The proposed algorithm [Alj et al., 2012b] relies on a regular subdivision of space into cubical elements called voxels and processes each voxel until the volumetric model is consistent with the input depth maps. During processing, rays are cast iteratively from each camera to its corresponding depth map and carve the global volume. Successive carving of the bounding volume with respect to each view yields a volume that is geometrically consistent with each view. The final surface is obtained from this volumetric model thanks to the Marching Cubes algorithm.
The proposed method is performed on 4 test sequences and compared with the Poisson surface reconstruction method which relies on surface estimation from an oriented point cloud by solving Poisson equation. It turns out that the proposed method outperforms Poisson-based surface reconstruction in terms of image distortion during rendering. The so-called image distortion is measured in terms of PSNR since our purpose is multi-video compression and the PSNR is the most widely used quality metric within the video compression community.

The main drawback of the proposed algorithm is that volumetric resolution should be specified by the user. However, choosing an appropriate voxel resolution is crucial to avoid aliasing problems in the reconstructed surface. Intuitively, we can say that voxel resolution should be large enough to capture different changes of the mesh geometry. Analytic determination of the volumetric resolution was proposed by Brian Curless in the context of a signed distance function for a single view [Curless, 1997]. Yet, specifying the appropriate resolution for the set of all available range images is a tedious task. One alternative is to use hierarchical representations such as octrees [Szeliski, 1993], following the work developed by [Fuhrmann and Goesele, 2011]. As a perspective, we plan to extend the proposed method to handle video depth maps rather than still images.

Texture Mapping using Photo-consistency:

In chapter 8 we proposed a new multi-texturing scheme that assigns the best view for each triangle by minimizing an energy function referred to as photo-consistency metric [Alj et al., 2012a]. This photo-consistency metric is computed by projecting a triangle mesh onto the set of available views, and the best texture is assigned to the view that minimizes the so-called metric. Unlike conventional geometric based criteria for best texture assignment, the proposed method reduces several visual artifacts during rendering and improves both objective and visual quality. The proposed multi-texturing scheme allows therefore to synthesize virtual views.

In this context, a comparison with an image-based novel view synthesis method (MPEG-VSRS) was also performed in chapter 9. Results show that the image based view synthesis method outperforms the mesh-based method in terms of visual and objective quality. The advantage of the proposed texture mapping method is the generation of texture maps suitable for transmission. Nevertheless, this texture mapping is time consuming, it requires about 20 mins to process 3 views. This method is therefore best suited for offline rendering. Hence, efficient GPU-implementation of the proposed algorithm should be carried out in a future work.

Besides, for transmission purposes, we plan to investigate how homogeneity can be introduced so that a cluster of triangles share the same texture. Such homogeneity will reduce the entropy of the texture signal, yet it should be performed without altering too much the overall rendering quality.

Compression of the proposed model:

The proposed model is composed of the merged mesh (chapter 6) and the texture model (chapter 11). The texture model was generated so that the current multi-view video compression standard MPEG-MVC can transmit this model with the lowest bitrate and without altering the quality of the rendered views. The proposed geometric model is encoded using MPEG-TFAN mesh codec standard. Besides, an entropy encoding technique was employed in order to encode best-texture-assignment information. Coding efficiency was performed and compared to the current MPEG-4/MVC standard. These results show that, in terms of rate/distortion trade-off, MPEG-4/MVC outperforms the proposed model. This is due mainly to the high bit rate required to encode the proposed geometric model.

Consequently a more efficient method to encode the geometric model is needed. Besides, temporal dimension should be taken into account. This allows additional consistency with the input data and therefore efficient transmission of the proposed model. A straightforward approach in order to build a time-consistent mesh was proposed by Mueller *et al*. in [Mueller et al., 2004b]. A mesh is built for each time instant, then vertices of the resulting meshes are matched using neighborhood criteria (*e.g*. vertex position and normal features). Thus a time-consistent representation is obtained that presents the same connectivity and moving vertices. Even with the constant connectivity assumption, this method can be tested in order to evaluate the possible distortion and coding gains.

Appendices

Appendix A

Test sequences

A.1 Microsoft sequences

Microsoft has provided ballet and breakdancers sequences. These sequences have XGA resolution (*i.e.* 1024x768) and are provided by 8 convergent cameras (see Figure A.1).

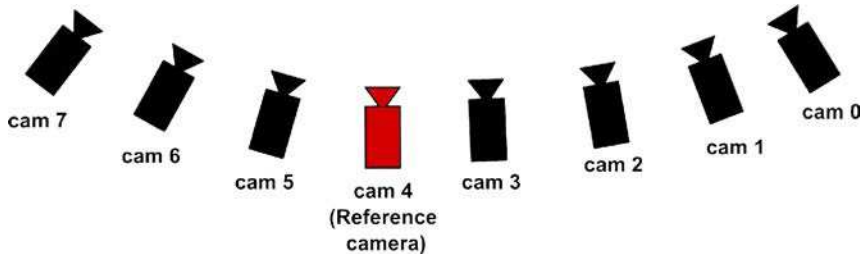
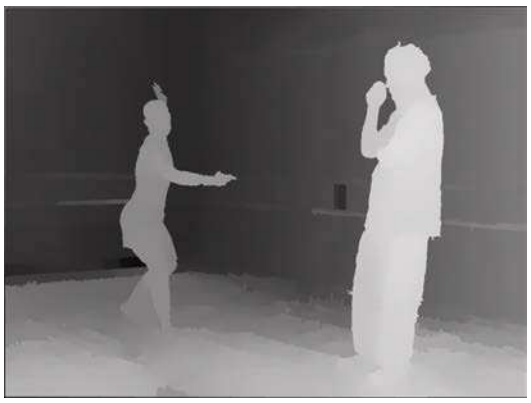


Figure A.1: Camera arrangement used for capturing Microsoft breakdancers and ballet sequences.

(a) Texture camera 0 (b) Depth camera 0

Figure A.2: Input texture and depth data corresponding to camera 0 for ballet sequence.

(a) Texture camera 6



(b) Depth camera 6

Figure A.3: Input texture and depth data corresponding to camera 6 for ballet sequence.

(a) Texture camera 0

(b) Depth camera 0

Figure A.4: Input texture and depth data corresponding to camera 0 for breakdancers sequence.

(a) Texture camera 6

(b) Depth camera 6

Figure A.5: Input texture and depth data corresponding to camera 6 for breakdancers sequence.

A.2 Nagoya University's sequences

Nagoya University has also provided balloons and kendo sequences. These sequences have also XGA resolution (*i.e.* 1024x768) and are captured by 7 parallel cameras (see Figure A.6). The quality of the provided depth maps is a bit lower than those provided by Microsoft. Only depth maps corresponding to views 1, 3 and 5 are available. These sequences are commonly used within the former MPEG/3DV.

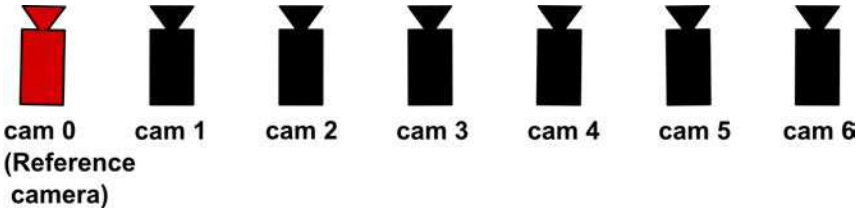
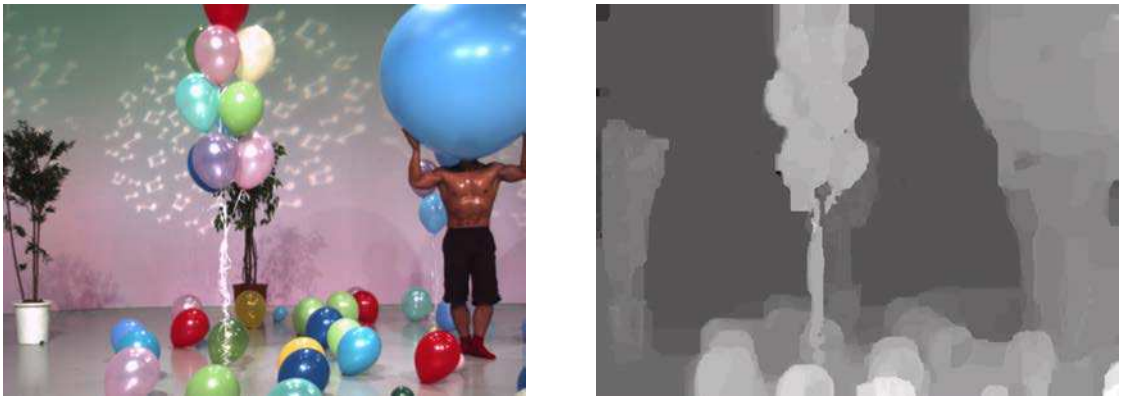


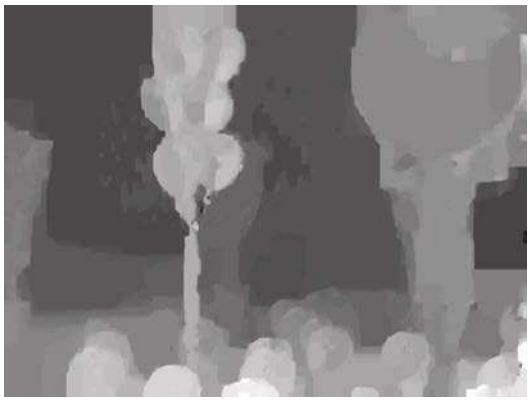
Figure A.6: Camera arrangement used for capturing MPEG balloons and kendo sequences.



(a) Texture camera 1 (b) Depth camera 1

Figure A.7: Input texture and depth data corresponding to camera 1 for balloons sequence.

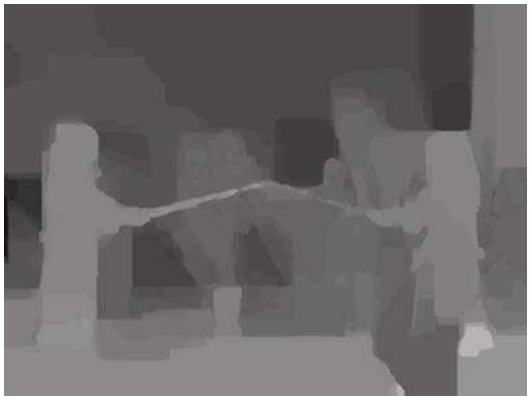
(a) Texture camera 5



(b) Depth camera 5

Figure A.8: Input texture and depth data corresponding to camera 5 for balloons sequence.

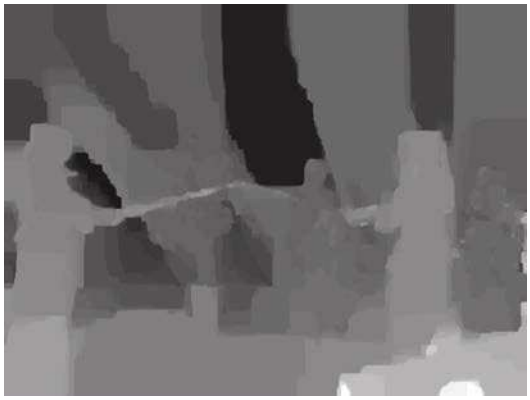
(a) Texture camera 1



(b) Depth camera 1

Figure A.9: Input texture and depth data corresponding to camera 1 for kendo sequence.

(a) Texture camera 5



(b) Depth camera 5

Figure A.10: Input texture and depth data corresponding to camera 5 for kendo sequence.

List of Figures

List of Tables

List of Algorithms

Bibliography

[Adelson and Bergen, 1991] Adelson, E. and Bergen, J. (1991). The plenoptic function and the elements of early vision. *Computational models of visual processing*, 1:3–20.

[Aganj et al., 2007] Aganj, E., Pons, J., Ségonne, F., Keriven, R., Team, W., and CERTIS, E. (2007). Spatio-temporal shape from silhouette using four-dimensional delaunay meshing. In *IEEE International Conference on Computer Vision*, pages 1–8. Citeseer.

[Agarwal et al., 2009] Agarwal, S., Snavely, N., Simon, I., Seitz, S., and Szeliski, R. (2009). Building rome in a day. In *Computer Vision, 2009 IEEE 12th International Conference on*, pages 72–79. Ieee.

[Alatan et al., 2007] Alatan, A., Yemez, Y., Gudukbay, U., Zabulis, X., Muller, K., Erdem, C., Weigel, C., Smolic, A., and METU, A. (2007). Scene representation technologies for 3dtv - a survey. *IEEE transactions on circuits and systems for video technology*, 17(11):1587–1605. http://www.ics.forth.gr/~zabulis/B1.pdf.

[Alj et al., 2012a] Alj, Y., Boisson, G., Bordes, P., Pressigout, M., and Morin, L. (2012a). Multi-texturing 3d models: How to choose the best texture? *IC3D proceedings*.

[Alj et al., 2012b] Alj, Y., Boisson, G., Bordes, P., Pressigout, M., and Morin, L. (2012b). Space carving mvd sequences for modeling natural 3d scenes. volume 8290, page 829005. SPIE.

[Allene et al., 2008] Allene, C., Pons, J., and Keriven, R. (2008). Seamless image-based texture atlases using multi-band blending. In *Pattern Recognition, 2008. ICPR 2008. 19th International Conference on*, pages 1–4. Ieee.

[Alliez and Gotsman, 2003] Alliez, P. and Gotsman, C. (2003). Recent advances in compression of 3d meshes. In *Proceedings of the Symposium on Multiresolution in Geometric Modeling*, volume 3. Springer. http://www.cs.technion.ac.il/~gotsman/AmendedPubl/Pierre/compression_survey.pdf.

[Appel, 1968] Appel, A. (1968). Some techniques for shading machine renderings of solids. In *Proceedings of the April 30–May 2, 1968, spring joint computer conference*, pages 37–45. ACM.

[Bajaj et al., 1999] Bajaj, C., Pascucci, V., and Zhuang, G. (1999). Single-resolution compression of arbitrary triangular meshes with properties. In *Data Compression Conference, 1999. Proceedings. DCC'99*, pages 247–256. IEEE.

[Balter et al., 2006] Balter, R., Gioia, P., and Morin, L. (2006). Scalable and efficient video coding using 3-d modeling. *Multimedia, IEEE Transactions on*, 8(6):1147–1155.

[Baumberg, 2002] Baumberg, A. (2002). Blending images for texturing 3d models. In *Proceedings of the British Machine Vision Conference*, pages 404–413. Citeseer.

[Besl and McKay, 1992] Besl, P. and McKay, N. (1992). A method for registration of 3-d shapes. *IEEE Transactions on pattern analysis and machine intelligence*, 14(2):239–256.

[Boissonnat and Oudot, 2006] Boissonnat, J. and Oudot, S. (2006). Provably good sampling and meshing of lipschitz surfaces. In *Proceedings of the twenty-second annual symposium on Computational geometry*, pages 337–346. ACM.

[Boyer et al., 2003] Boyer, E., Franco, J., et al. (2003). A hybrid approach for computing visual hulls of complex objects. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 695–701.

[Broadhurst et al., 2001] Broadhurst, A., Drummond, T., and Cipolla, R. (2001). A probabilistic framework for space carving. In *Computer Vision, 2001. ICCV 2001. Proceedings. Eighth IEEE International Conference on*, volume 1, pages 388–393. IEEE.

[Buehler et al., 2001] Buehler, C., Bosse, M., McMillan, L., Gortler, S., and Cohen, M. (2001). Unstructured lumigraph rendering. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 425–432. ACM.

[Catmull, 1974] Catmull, E. (1974). A subdivision algorithm for computer display of curved surfaces. Technical report, DTIC Document.

[Chari et al., 2012] Chari, V., Agrawal, A., Taguchi, Y., and Ramalingam, S. (2012). Convex bricks: A new primitive for visual hull modeling and reconstruction. In *Robotics and Automation (ICRA), 2012 IEEE International Conference on*, pages 770–777. IEEE.

[Child, 2011] Child, B. (2011). James cameron expects 100% 3d within the next five years. Guardian Tuesday 12 April, 2011.

[Chow, 1997] Chow, M. (1997). Optimized geometry compression for real-time rendering. In *Proc. IEEE Visualization*, volume 97, pages 347–354. http://home.earthlink.net/~mmchow/gcompiler/gcompiler.html.

[Cohen-Or et al., 2003] Cohen-Or, D., Chrysanthou, Y., Silva, C., and Durand, F. (2003). A survey of visibility for walkthrough applications. *Visualization and Computer Graphics, IEEE Transactions on*, 9(3):412–431.

[Colleu, 2010] Colleu, T. (2010). *A floating polygon soup representation for 3D video*. These, Université Rennes 1.

[Colleu et al., 2010] Colleu, T., Pateux, S., Morin, L., and Labit, C. (2010). A polygon soup representation for multiview coding. *Journal of Visual Communication and Image Representation*, 21(5-6):561–576.

[Cover et al., 1991] Cover, T., Thomas, J., Wiley, J., et al. (1991). *Elements of information theory*, volume 6. Wiley Online Library.

[Culbertson et al., 2000] Culbertson, W., Malzbender, T., and Slabaugh, G. (2000). Generalized voxel coloring. *Vision Algorithms: Theory and Practice*, pages 100–115.

[Curless and Levoy, 1996] Curless, B. and Levoy, M. (1996). A volumetric method for building complex models from range images. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 303–312. ACM.

[Curless, 1997] Curless, B. L. (1997). *New methods for surface reconstruction from range images*. PhD thesis, Stanford University.

[Daribo, 2009] Daribo, I. (2009). *Codage et rendu de séquence vidéo 3D et applications à la télévision tridimensionnelle (TV3D) et à la télévision bàse de rendu de vidéos*. PhD thesis, ENST Paris.

[Debevec et al., 1996] Debevec, P., Taylor, C., and Malik, J. (1996). Modeling and rendering architecture from photographs: A hybrid geometry-and image-based approach. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 11–20. ACM.

[Debevec et al., 1998] Debevec, P., Yu, Y., and Borshukov, G. (1998). Efficient view-dependent image-based rendering with projective texture-mapping. In *Eurographics Rendering Workshop*, volume 98, pages 105–116.

[Décoret et al., 2003] Décoret, X., Durand, F., Sillion, F., and Dorsey, J. (2003). Billboard clouds for extreme model simplification. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 689–696. ACM.

[Deering, 1995] Deering, M. (1995). Geometry compression. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, page 20. ACM. http://www.cse.ohio-state.edu/~hwshen/Su01_888/deering.pdf.

[Dellepiane et al., 2012] Dellepiane, M., Marroquim, R., Callieri, M., Cignoni, P., and Scopigno, R. (2012). Flow-based local optimization for image-to-geometry projection. *Visualization and Computer Graphics, IEEE Transactions on*, 18(3):463–474.

[Dey et al., 2003] Dey, T., Giesen, J., and John, M. (2003). Alpha-shapes and flow shapes are homotopy equivalent. In *Proceedings of the thirty-fifth annual ACM symposium on Theory of computing*, pages 493–502. ACM.

[Dyer, 2001] Dyer, C. (2001). Volumetric scene reconstruction from multiple views. *Foundations of Image Understanding*, pages 469–489.

[Edelsbrunner and Mücke, 1994] Edelsbrunner, H. and Mücke, E. (1994). Three-dimensional alpha shapes. *ACM Transactions on Graphics (TOG)*, 13(1):43–72.

[Eisemann et al., 2008] Eisemann, M., De Decker, B., Magnor, M., Bekaert, P., de Aguiar, E., Ahmed, N., Theobalt, C., and Sellent, A. (2008). Floating textures. *Computer Graphics Forum (Proc. of Eurographics)*, 27(2):409–418.

[Everitt, 2001] Everitt, C. (2001). Projective texture mapping. *White paper, NVidia Corporation*, 4.

[Fehn et al., 2002] Fehn, C., Kauff, P., De Beeck, M., Ernst, F., Ijsselsteijn, W., Pollefeys, M., Van Gool, L., Ofek, E., and Sexton, I. (2002). An evolutionary and optimised approach on 3d-tv. In *Proc. of IBC*, volume 2, pages 357–365.

[Floater and Hormann, 2005] Floater, M. and Hormann, K. (2005). Surface parameterization: a tutorial and survey. *Advances in multiresolution for geometric modelling*, pages 157–186.

[Foley, 1996] Foley, J. (1996). *Computer graphics: principles and practice*. Addison-Wesley Professional.

[Franco and Boyer, 2005] Franco, J. and Boyer, E. (2005). Fusion of multiview silhouette cues using a space occupancy grid. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 2, pages 1747–1753. IEEE.

[Franco De Carvalho, 2012] Franco De Carvalho, P. (2012). Implementation of image driven mesh simplification algorithm. Internship Report.

[Fuhrmann and Goesele, 2011] Fuhrmann, S. and Goesele, M. (2011). Fusion of depth maps with multiple scales. In *ACM Transactions on Graphics (TOG)*, volume 30, page 148. ACM.

[Fujii et al., 2007] Fujii, T., Yendo, T., and Tanimoto, M. (2007). Ray-space transmission system with real-time acquisition and display. In *Lasers and Electro-Optics Society, 2007. LEOS 2007. The 20th Annual Meeting of the IEEE*, pages 78–79. IEEE.

[Furukawa and Ponce, 2010] Furukawa, Y. and Ponce, J. (2010). Accurate, dense, and robust multiview stereopsis. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 32(8):1362–1376.

[Gal et al., 2010] Gal, R., Wexler, Y., Ofek, E., Hoppe, H., and Cohen-Or, D. (2010). Seamless montage for texturing models. In *Computer Graphics Forum*, volume 29, pages 479–486. Wiley Online Library.

[Gandoin and Devillers, 2002] Gandoin, P. and Devillers, O. (2002). Progressive lossless compression of arbitrary simplicial complexes. In *Proceedings of the 29th annual conference on Computer graphics and interactive techniques*, pages 372–379. ACM New York, NY, USA. http://hal.archives-ouvertes.fr/docs/00/16/72/16/PDF/hal.pdf.

[Gish and Pierce, 1968] Gish, H. and Pierce, J. (1968). Asymptotically efficient quantizing. *Information Theory, IEEE Transactions on*, 14(5):676–683.

[Gortler et al., 1996] Gortler, S., Grzeszczuk, R., Szeliski, R., and Cohen, M. (1996). The lumigraph. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 43–54. ACM.

[Haeberli and Segal, 1993] Haeberli, P. and Segal, M. (1993). Texture mapping as a fundamental drawing primitive. In *Fourth Eurographics Workshop on Rendering*, volume 259, page 266.

[He et al., 1998] He, L., Shade, J., Gortler, S., and Szeliski, R. (1998). Layered depth images. In *Proceedings of the 25th annual conference on computer graphics and interactive techniques (SIGGRAPH 1998), July*, pages 19–24.

[Heckbert, 1986] Heckbert, P. (1986). Survey of texture mapping. *Computer Graphics and Applications, IEEE*, 6(11):56–67.

[Hernández Esteban and Schmitt, 2004] Hernández Esteban, C. and Schmitt, F. (2004). Silhouette and stereo fusion for 3d object modeling. *Computer Vision and Image Understanding*, 96(3):367–392.

[Hilton et al., 1996] Hilton, A., Stoddart, A., Illingworth, J., and Windeatt, T. (1996). Reliable surface reconstruction from multiple range images. *Computer Vision–ECCV'96*, pages 117–126.

[Hoppe, 1996] Hoppe, H. (1996). Progressive mesh. In *Proceedings of SIGGRAPH*, volume 96, pages 99–108. http://research.microsoft.com/en-us/um/people/hoppe/pm.pdf.

[Hormann et al., 2007] Hormann, K., Lévy, B., Sheffer, A., et al. (2007). Mesh parameterization: Theory and practice. *SIGGRAPH Course Notes*.

[Huffman, 1952] Huffman, D. (1952). A method for the construction of minimum-redundancy codes. *Proceedings of the IRE*, 40(9):1098–1101.

[Iiyama et al., 2010] Iiyama, M., Kakusho, K., and Minoh, M. (2010). Super-resolution texture mapping from multiple view images. In *Pattern Recognition (ICPR), 2010 20th International Conference on*, pages 1820–1823. Ieee.

[Isenburg and Alliez, 2002] Isenburg, M. and Alliez, P. (2002). Compressing polygon mesh geometry with parallelogram prediction. In *Visualization, 2002. VIS 2002. IEEE*, pages 141–146. IEEE.

[ISO/IEC, 2001] ISO/IEC (July 2001). Text of iso/iec 14496-2 (mpeg-4 visual) 2001 edition.

[Karni and Gotsman, 2000] Karni, Z. and Gotsman, C. (2000). Spectral compression of mesh geometry. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 279–286. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.24.1185&rep=rep1&type=pdf.

[Kazhdan et al., 2006] Kazhdan, M., Bolitho, M., and Hoppe, H. (2006). Poisson surface reconstruction. In *Proceedings of the fourth Eurographics symposium on Geometry processing*, pages 61–70. Eurographics Association.

[Kutulakos and Seitz, 2000] Kutulakos, K. and Seitz, S. (2000). A theory of shape by space carving. *International Journal of Computer Vision*, 38(3):199–218.

[Labatut et al., 2007] Labatut, P., Pons, J., and Keriven, R. (2007). Efficient multi-view reconstruction of large-scale scenes using interest points, delaunay triangulation and graph cuts. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE.

[Laurentini, 1994] Laurentini, A. (1994). The visual hull concept for silhouette-based image understanding. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 16(2):150–162.

[Lempitsky and Ivanov, 2007] Lempitsky, V. and Ivanov, D. (2007). Seamless mosaicing of image-based texture maps. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–6. IEEE.

[Levoy and Hanrahan, 1996] Levoy, M. and Hanrahan, P. (1996). Light field rendering. In *Proceedings of the 23rd annual conference on Computer graphics and interactive techniques*, pages 31–42. ACM.

[Levoy and Whitted, 1985] Levoy, M. and Whitted, T. (1985). *The use of points as a display primitive*. University of North Carolina, Department of Computer Science.

[Lévy et al., 2002] Lévy, B., Petitjean, S., Ray, N., and Maillot, J. (2002). Least squares conformal maps for automatic texture atlas generation. In *ACM Transactions on Graphics (TOG)*, volume 21, pages 362–371. ACM.

[Lindstrom and Turk, 2000] Lindstrom, P. and Turk, G. (2000). Image-driven simplification. *ACM Transactions on Graphics (ToG)*, 19(3):204–241.

[Lorensen and Cline, 1987] Lorensen, W. and Cline, H. (1987). Marching cubes: A high resolution 3d surface construction algorithm. *ACM Siggraph Computer Graphics*, 21(4):163–169.

[Lovi et al., 2010] Lovi, D., Birkbeck, N., Cobzas, D., and Jagersand, M. (2010). Incremental free-space carving for real-time 3d reconstruction. In *Proc. of 3DPVT*.

[Maillot et al., 1993] Maillot, J., Yahia, H., and Verroust, A. (1993). Interactive texture mapping. In *Proceedings of the 20th annual conference on Computer graphics and interactive techniques*, pages 27–34. ACM.

[Mamou, 2008] Mamou, K. (September 2008). *Compression de maillages 3D statiques et dynamiques*. PhD thesis, Unversité Paris V-René Descartes, France.

[Mamou and Dehais, 2010] Mamou, K. and Dehais, C. (2010). Ptfan: a multi-resolution extension for the tfan codec. Technical report, FittingBox1 and University of Manouba (Manouba, Tunisia).

[Mamou et al., 2008a] Mamou, K., Stefanoski, N., Kirchhoffer, H., Müller, K., Zaharia, T., Preteux, F., Marpe, D., and Ostermann, J. (2008a). The new mpeg-4/famc standard for animated 3d mesh compression. In

3DTV Conference: The True Vision-Capture, Transmission and Display of 3D Video, 2008, pages 97–100. http://iphome.hhi.de/marpe/download/FAMC-3DTV-CON-2008.pdf.

[Mamou et al., 2008b] Mamou, K., Zaharia, T., Preda, M., and Francoise, P. (July 2008b). Tfan software description. *ISO/IEC JTC 1/SC 29/WG 11/ M15653*.

[Mamou et al., 2009] Mamou, K., Zaharia, T., and Prêteux, F. (2009). A triangle-fan-based approach for low complexity 3d mesh compression. In *Image Processing (ICIP), 2009 16th IEEE International Conference on*, pages 3513–3516. IEEE.

[Mantler et al., 2007] Mantler, S., Jeschke, S., and Wimmer, M. (2007). Displacement mapped billboard clouds. In *Proceedings of symposium on interactive 3D graphics and games*. Citeseer.

[Marpe et al., 2003] Marpe, D., Schwarz, H., and Wiegand, T. (2003). Context-based adaptive binary arithmetic coding in the h. 264/avc video compression standard. *Circuits and Systems for Video Technology, IEEE Transactions on*, 13(7):620–636.

[McMillan and Bishop, 1995] McMillan, L. and Bishop, G. (1995). Plenoptic modeling: An image-based rendering system. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 39–46. ACM.

[Merkle et al., 2007] Merkle, P., Smolic, A., Muller, K., and Wiegand, T. (2007). Multi-view video plus depth representation and coding. In *ICIP 2007*, volume 1, pages I–201. IEEE.

[Merrell et al., 2007] Merrell, P., Akbarzadeh, A., Wang, L., Mordohai, P., Frahm, J., Yang, R., Nistér, D., and Pollefeys, M. (2007). Real-time visibility-based fusion of depth maps. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE.

[Milenkovic, 1998] Milenkovic, V. (1998). Rotational polygon containment and minimum enclosure. In *Proceedings of the fourteenth annual symposium on Computational geometry*, pages 1–8. ACM.

[Morvan, 2009] Morvan, Y. (2009). *Acquisition, compression and rendering of depth and texture for multi-view video*. PhD thesis. http://vca.ele.tue.nl/publications/data/Morvan2009b.pdf.

[Mueller et al., 2004a] Mueller, K., Smolic, A., Merkle, P., Kaspar, B., Eisert, P., and Wiegand, T. (2004a). 3d reconstruction of natural scenes with view-adaptive multi-texturing. In *3DPVT 2004*, pages 116–123. IEEE.

[Mueller et al., 2004b] Mueller, K., Smolic, A., Merkle, P., Kautzner, M., and Wiegand, T. (2004b). Coding of 3d meshes and video textures for 3d video objects. In *Proc. Picture Coding Symposium*. Citeseer.

[Muller et al., 2010] Muller, K., Dix, K., Merkle, P., and Wiegand, T. (2010). Temporal residual data sub-sampling in ldv representation format. In *3DTV-Conference: The True Vision-Capture, Transmission and Display of 3D Video (3DTV-CON), 2010*, pages 1–4. IEEE.

[Newcombe et al., 2011] Newcombe, R., Davison, A., Izadi, S., Kohli, P., Hilliges, O., Shotton, J., Molyneaux, D., Hodges, S., Kim, D., and Fitzgibbon, A. (2011). Kinectfusion: Real-time dense surface mapping and tracking. In *Mixed and Augmented Reality (ISMAR), 2011 10th IEEE International Symposium on*, pages 127–136. IEEE.

[Ozaktas and Onural, 2007] Ozaktas, H. and Onural, L. (2007). *Three-Dimensional Television: Capture, Transmission, Display*. Springer.

[Pan et al., 2009] Pan, Q., Reitmayr, G., and Drummond, T. (2009). Proforma: Probabilistic feature-based on-line rapid model acquisition. In *Proc. 20th British Machine Vision Conference (BMVC)*.

[Pardas et al.,] Pardas, J., Casas, J., Landabaso, J., and Pardas, M. Shape from inconsistent silhouette.

[Peng et al., 2005] Peng, J., Kim, C., and Jay Kuo, C. (2005). Technologies for 3d mesh compression: A survey. *Journal of Visual Communication and Image Representation*, 16(6):688–733. http://www.jingliang.org/papers/PKK_Survey2005.pdf.

[Peng and Kuo, 2005] Peng, J. and Kuo, C. (2005). Geometry-guided progressive lossless 3d mesh coding with octree (ot) decomposition. *ACM Transactions on Graphics (TOG)*, 24(3):616. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.107.5938&rep=rep1&type=pdf.

[Pérez et al., 2003] Pérez, P., Gangnet, M., and Blake, A. (2003). Poisson image editing. In *ACM Transactions on Graphics (TOG)*, volume 22, pages 313–318. ACM.

[Pfister et al., 2000] Pfister, H., Zwicker, M., Van Baar, J., and Gross, M. (2000). Surfels: Surface elements as rendering primitives. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 335–342. ACM Press/Addison-Wesley Publishing Co.

[Pons and Boissonnat, 2007] Pons, J.-P. and Boissonnat, J.-D. (2007). Delaunay deformable models: Topology-adaptive meshes based on the restricted delaunay triangulation. In *Computer Vision and Pattern Recognition, 2007. CVPR'07. IEEE Conference on*, pages 1–8. IEEE.

[Popović and Hoppe, 1997] Popović, J. and Hoppe, H. (1997). Progressive simplicial complexes. In *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*, pages 217–224. ACM Press/Addison-Wesley Publishing Co. New York, NY, USA. http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.8.9102&rep=rep1&type=pdf.

[Potmesil, 1987] Potmesil, M. (1987). Generating octree models of 3d objects from their silhouettes in a sequence of images. *Computer Vision, Graphics, and Image Processing*, 40(1):1–29.

[Ray et al., 2010] Ray, N., Nivoliers, V., Lefebvre, S., and Lévy, B. (2010). Invisible seams. In *Computer Graphics Forum*, volume 29, pages 1489–1496. Wiley Online Library.

[Richardson, 2003] Richardson, I. E. (2003). *H.264 and MPEG-4 Video Compression: Video Coding for Next Generation Multimedia*. Wiley, 1 edition.

[Rocchini et al., 1999] Rocchini, C., Cignoni, P., Montani, C., and Scopigno, R. (1999). Multiple textures stitching and blending on 3d objects. In *Eurographics Rendering Workshop 1999*, pages 119–130.

[Rusinkiewicz and Levoy, 2000] Rusinkiewicz, S. and Levoy, M. (2000). Qsplat: A multiresolution point rendering system for large meshes. In *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*, pages 343–352. ACM Press/Addison-Wesley Publishing Co.

[Salman et al., 2010] Salman, N., Yvinec, M., et al. (2010). Surface reconstruction from multi-view stereo of large-scale outdoor scenes. *International Journal of Virtual Reality*, 9(1):19–26.

[Sander et al., 2001] Sander, P., Snyder, J., Gortler, S., and Hoppe, H. (2001). Texture mapping progressive meshes. In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pages 409–416. ACM.

[Scharstein and Szeliski, 2002] Scharstein, D. and Szeliski, R. (2002). A taxonomy and evaluation of dense two-frame stereo correspondence algorithms. *International journal of computer vision*, 47(1):7–42.

[Segal et al., 1992] Segal, M., Korobkin, C., Van Widenfelt, R., Foran, J., and Haeberli, P. (1992). Fast shadows and lighting effects using texture mapping. In *ACM SIGGRAPH Computer Graphics*, volume 26, pages 249–252. ACM.

[Seitz and Dyer, 1999] Seitz, S. and Dyer, C. (1999). Photorealistic scene reconstruction by voxel coloring. *International Journal of Computer Vision*, 35(2):151–173.

[Sheffer et al., 2006] Sheffer, A., Praun, E., and Rose, K. (2006). Mesh parameterization methods and their applications. *Foundations and Trends® in Computer Graphics and Vision*, 2(2):105–171.

[Shreiner et al., 2004] Shreiner, D., Woo, M., Neider, J., and Davis, T. (2004). *OpenGL programming guide*. Addison-Wesley.

[Shum and Kang, 2000] Shum, H. and Kang, S. (2000). A review of image-based rendering techniques. *IEEE/SPIE Visual Communications and Image Processing (VCIP)*, 213.

[Sinha and Pollefeys, 2005] Sinha, S. and Pollefeys, M. (2005). Multi-view reconstruction using photo-consistency and exact silhouette constraints: A maximum-flow formulation. In *Computer Vision, 2005. ICCV 2005. Tenth IEEE International Conference on*, volume 1, pages 349–356. IEEE.

[Slabaugh et al., 2001a] Slabaugh, G., Culbertson, B., Malzbender, T., and Schafer, R. (2001a). A survey of methods for volumetric scene reconstruction from photographs. In *International Workshop on Volume Graphics*, volume 2. Citeseer.

[Slabaugh et al., 2004] Slabaugh, G., Culbertson, W., Malzbender, T., Stevens, M., and Schafer, R. (2004). Methods for volumetric reconstruction of visual scenes. *International Journal of Computer Vision*, 57(3):179–199.

[Slabaugh et al., 2001b] Slabaugh, G., Malzbender, T., and Culbertson, W. (2001b). Volumetric warping for voxel coloring on an infinite domain. *3D Structure from Images–SMILE 2000*, pages 109–123.

[Smolic and Kauff, 2005] Smolic, A. and Kauff, P. (2005). Interactive 3-d video representation and coding technologies. *Proceedings of the IEEE*, 93(1):98–110. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=1369701&isnumber=29978?tag=1.

[Smolic et al., 2007] Smolic, A., Mueller, K., Stefanoski, N., Ostermann, J., Gotchev, A., Akar, G., Triantafyllidis, G., and Koz, A. (2007). Coding algorithms for 3dtv-a survey. *IEEE transactions on circuits and systems for video technology*, 17(11):1606–1620. http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=4373329&isnumber=4373313.

[Snavely et al., 2010] Snavely, N., Simon, I., Goesele, M., Szeliski, R., and Seitz, S. (2010). Scene reconstruction and visualization from community photo collections. *Proceedings of the IEEE*, 98(8):1370–1390.

[Sorkine et al., 2002] Sorkine, O., Cohen-Or, D., Goldenthal, R., and Lischinski, D. (2002). Bounded-distortion piecewise mesh parameterization. In *Proceedings of the conference on Visualization'02*, pages 355–362. IEEE Computer Society.

[Sullivan and Wiegand, 1998] Sullivan, G. and Wiegand, T. (1998). Rate-distortion optimization for video compression. *Signal Processing Magazine, IEEE*, 15(6):74–90.

[Szeliski, 1993] Szeliski, R. (1993). Rapid octree construction from image sequences. *CVGIP Image Understanding*, 58:23–23.

[Tanimoto et al., 2007] Tanimoto, M., Fujii, T., and Suzuki, K. (october 2007). Experiment of view synthesis using multi-view depth. In *ISO/IEC JTC1/SC29/WG11 M14889*.

[Tanimoto and Suzuki, 2009] Tanimoto, M. and Suzuki, K. (February 2009). View synthesis algorithm in view synthesis reference software 2.0 (vsrs2.0). *ISO/IEC JTC1/SC29/WG11*, M16090.

[Taubin, 1995] Taubin, G. (1995). A signal processing approach to fair surface design. In *Proceedings of the 22nd annual conference on Computer graphics and interactive techniques*, pages 351–358. ACM.

[Taubin et al., 1998] Taubin, G., Guéziec, A., Horn, W., and Lazarus, F. (1998). Progressive forest split compression. In *Proceedings of the 25th annual conference on Computer graphics and interactive techniques*, pages 123–132. ACM New York, NY, USA. http://mesh.brown.edu/taubin/pdfs/taubin-etal-sg98.pdf.

[Taubin and Rossignac, 1998] Taubin, G. and Rossignac, J. (1998). Geometric compression through topological surgery. *ACM transactions on Graphics*, 17(2):84–115. http://mesh.brown.edu/taubin/pdfs/TaubinRossignac-tog98.pdf.

[Touma and Gotsman, 1998] Touma, C. and Gotsman, C. (1998). Triangle mesh compression. In *Graphics Interface*, pages 26–34. http://www.cs.technion.ac.il/~gotsman/AmendedPubl/TriangleMesh/Convert-Triangle.pdf.

[Treuille et al., 2004] Treuille, A., Hertzmann, A., and Seitz, S. (2004). Example-based stereo with general brdfs. *Computer Vision-ECCV 2004*, pages 457–469.

[Turán, 1984] Turán, G. (1984). On the succinct representation of graphs. *Discrete Applied Mathematics*, 8(3):289–294.

[Turk and Levoy, 1994] Turk, G. and Levoy, M. (1994). Zippered polygon meshes from range images. In *Proceedings of the 21st annual conference on Computer graphics and interactive techniques*, pages 311–318. ACM.

[Vedula et al., 2005] Vedula, S., Baker, S., and Kanade, T. (2005). Image-based spatio-temporal modeling and view interpolation of dynamic events. *ACM Transactions on Graphics (TOG)*, 24(2):240–261.

[Vogiatzis et al., 2005] Vogiatzis, G., Torr, P., and Cipolla, R. (2005). Multi-view stereo via volumetric graph-cuts. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 2, pages 391–398. IEEE.

[Wang et al., 2001] Wang, L., Kang, S., Szeliski, R., and Shum, H. (2001). Optimal texture map reconstruction from multiple views. In *Computer Vision and Pattern Recognition, 2001. CVPR 2001. Proceedings of the 2001 IEEE Computer Society Conference on*, volume 1, pages I–347. IEEE.

[Wang et al., 2004] Wang, Z., Bovik, A. C., Sheikh, H. R., and Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. *Image Processing, IEEE Transactions on*, 13(4):600–612.

[Waschbüsch et al., 2007a] Waschbüsch, M., Würmlin, S., Cotting, D., and Gross, M. (2007a). Point-sampled 3d video of real-world scenes. *Signal Processing: Image Communication*, 22(2):203–216.

[Waschbüsch et al., 2005] Waschbüsch, M., Würmlin, S., Cotting, D., Sadlo, F., and Gross, M. (2005). Scalable 3d video of dynamic scenes. *The Visual Computer*, 21(8):629–638.

[Waschbüsch et al., 2007b] Waschbüsch, M., Würmlin, S., and Gross, M. (2007b). 3d video billboard clouds. In *Computer Graphics Forum*, volume 26, pages 561–569. Wiley Online Library.

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

[Yamazaki et al., 2002] Yamazaki, S., Sagawa, R., Kawasaki, H., Ikeuchi, K., and Sakauchi, M. (2002). Microfacet billboarding. In *Proceedings of the 13th Eurographics workshop on Rendering*, pages 169–180. Eurographics Association.

[Yang et al., 2003] Yang, R., Pollefeys, M., and Welch, G. (2003). Dealing with textureless regions and specular highlights-a progressive space carving scheme using a novel photo-consistency measure. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 576–584. IEEE.

[Zach et al., 2007] Zach, C., Pock, T., and Bischof, H. (2007). A globally optimal algorithm for robust tv-l1 range image integration. In *Computer Vision, 2007. ICCV 2007. IEEE 11th International Conference on*, pages 1–8. IEEE.

[Ziegler et al., 2004] Ziegler, G., Lensch, H., Ahmed, N., Magnor, M., and Seidel, H. (2004). Multivideo compression in texture space. In *Image Processing, 2004. ICIP'04. 2004 International Conference on*, volume 4, pages 2467–2470. IEEE.

[Zwicker et al., 2002] Zwicker, M., Pfister, H., Van Baar, J., and Gross, M. (2002). Ewa splatting. *Visualization and Computer Graphics, IEEE Transactions on*, 8(3):223–238.

INSA de RENNES
Service des Formations

AVIS DU JURY SUR LA REPRODUCTION DE LA THESE SOUTENUE

Titre de la thèse:
Space carving Multi-View Video plus Depth sequences for representation and transmission of 3DTV and FTV contents

Nom Prénom de l'auteur : ALJ YOUSSEF

Membres du jury :
- Madame MORIN Luce
- Madame PRESSIGOUT Muriel
- Monsieur Lucas Laurent
- Monsieur Domanski Marek
- Monsieur Daoudi Mohamed
- Monsieur Boyer Edmond
- Monsieur Boisson Guillaume

Président du jury : M. DAOUDI

Date de la soutenance : 16 Mai 2013

Reproduction de la these soutenue

☒ - Thèse pouvant être reproduite en l'état
☐ Thèse pouvant être reproduite après corrections suggérées

Fait à Rennes, le 16 Mai 2013

Signature du président de jury

M. DAOUDI

Le Directeur,

M'hamed DRISSI

Résumé

La vidéo 3D a suscité un intérêt croissant durant ces dernières années. Grâce au développement récent des écrans stéréoscopiques et auto-stéréoscopiques, la vidéo 3D fournit une sensation réaliste de profondeur à l'utilisateur et une navigation virtuelle autour de la scène observée. Cependant de nombreux défis techniques existent encore. Ces défis peuvent être liés à l'acquisition de la scène et à sa représentation d'une part ou à la transmission des données d'autre part.

Dans le contexte de la représentation de scènes naturelles, de nombreux efforts ont été fournis afin de surmonter ces difficultés. Les méthodes proposées dans la littérature peuvent être basées image, géométrie ou faire appel à des représentations combinant image et géométrie.

L'approche adoptée dans cette thèse consiste en une méthode hybride s'appuyant sur l'utilisation des séquences multi-vues plus profondeur MVD (Multi-view Video plus Depth) afin de conserver le photo-réalisme de la scène observée, combinée avec un modèle géométrique, à base de maillage triangulaire, renforçant ainsi la compacité de la représentation. Nous supposons que les cartes de profondeur des données MVD fournies sont fiables et que les caméras utilisées durant l'acquisition sont calibrées, les paramètres caméras sont donc connus, mais les images correspondantes ne sont pas nécessairement rectifiées. Nous considérerons ainsi le cas général où les caméras peuvent être parallèles ou convergentes.

Les contributions de cette thèse sont les suivantes. D'abord, un schéma volumétrique dédié à la fusion des cartes de profondeur en une surface maillée est proposé. Ensuite, un nouveau schéma de plaquage de texture multi-vues est proposé. Finalement, nous abordons à l'issue ce ces deux étapes de modélisation, la transmission proprement dite et comparons les performances de notre schéma de modélisation avec un schéma basé sur le standard MPEG-MVC, état de l'art dans la compression de vidéos multi-vues.

Abstract

3D videos have witnessed a growing interest in the last few years. Due to the recent development of stereoscopic and auto-stereoscopic displays, 3D videos provide a realistic depth perception to the user and allows a virtual navigation around the scene. Nevertheless, several technical challenges are still remaining. Such challenges are either related to scene acquisition and representation on the one hand or to data transmission on the other hand.

In the context of natural scene representation, research activities have been strengthened worldwide in order to handle these issues. The proposed methods for scene representation can be image-based, geometry based or methods combining both image and geometry.

In this thesis, we take advantage of image based representations, thanks to the use of Multi-view Video plus Depth representation, in order to preserve the photo-realism of the observed scene, and geometric based representations in order to enforce the compactness of the proposed scene representation.

We assume the provided depth maps to be reliable. Besides, the considered cameras are calibrated so that the cameras parameters are known but the corresponding images are not necessarily rectified. We consider, therefore, the general framework where cameras can be either convergent or parallel.

The contributions of this thesis are the following. First, a new volumetric framework is proposed in order to merge the input depth maps into a single and compact surface mesh. Second, a new algorithm for multi-texturing the surface mesh is proposed. Finally, we address the transmission issue and compare the performance of the proposed modeling scheme with the current standard MPEG-MVC, that is the state of the art of multi-view video compression.

N° d'ordre : **13ISAR 11 / D13-11**

Institut National des Sciences Appliquées de Rennes
20, Avenue des Buttes de Coësmes . CS 14315 . F-35043 Rennes Cedex
Tel : 02 23 23 82 00 – Fax : 02 23 23 83 96