Rendu de clone vidéo-réaliste pour TempoValse
Exposé des algorithmes ICP - Version du 12/10/2000

elisei@icp.inpg.fr

A. Buts du document

Ce document présente les données et algorithmes joints, dérivés de l'application *Mother*. Ensembles, il doivent permettre l'étude d'une implémentation matérielle, et notamment une éventuelle approximation par des opérations en 2D. Une mise en œuvre minimale en C et utilisant *Open GL* est aussi fournie, avec un jeu de données d'entrée et les résultats associés.

a/ Le clone : animation et rendu réalistes

Une fois présent en mémoire, le clone peut être animé à partir d'un faible nombre de paramètres : son modèle articulatoire représente de façon compacte un ensemble de postures atteignables. Pour le rendu réaliste, on fait appel à un *Morphing 3D*, entre plusieurs postures et textures de référence. Les deux sections qui suivent présentent les données nécessaires et leur utilisation concrète. L'avant-dernière section apporte quelques précisions sur les programmes et données fournis.

Le problème algorithmique de l'analyse (déjà traité dans le rapport d'Éric Petit) n'a pas été détaillé à nouveau ; les données nécessaires sont en revanche rappelées, et les incompatibilités qu'imposeraient certaines optimisations excessives sont mises en avant.

B Sémantique des données de base

Pour effectuer le rendu d'un clone (par exemple celui de Pierre Badin), le logiciel *Mother* (ou un éventuel décodeur) utilisera :

	Exemple pour Pierre Badin :
– **un modèle des points mobiles**. Piloté par un petit nombre de paramètres, le modèle articulatoire est un modèle linéaire, donc facile à évaluer. Il gère le déplacement, 3D dans *Mother*, d'un certain nombre des points mobiles (visage ou lèvres) du clone .	≈ 200 points mobiles, commandés par 6 paramètres
– **un ensemble de points statiques**. Avec des coordonnées 3D totalement figées, ils complètent la collection précédente pour décrire le reste du visage (front, cheveux...).	≈ 200 points
– **un modèle géométrique**. On spécifie la topologie, sous forme d'un réseau de triangles, qui permet de connecter tous les points 3D du visage. Selon qu'ils sont ou ne sont pas liés à des points mobiles, certains triangles se déformeront à la projection. Dans *Mother*, la différenciation n'est pas faite : la caméra et l'objet sont mobiles et traités le plus génériquement.	de 300 à 700 triangles, dont une cinquantaine pour l'analyse du pourtour des lèvres
– **un modèle de caméra**. Il spécifie comment l'on transforme tous les points 3D en coordonnées écran. Dans le cas d'un rendu de face, on peut imaginer le réduire à sa plus simple expression, et utiliser directement les abscisses et ordonnées. Dans le cas général, utile pour l'analyse, ce modèle génère deux divisions par point à transformer.	au maximum 16 réels
– **un modèle de rendu texturé**. On part d'un nombre restreint de configurations articulatoires réelles, prises comme exemples, avec pour chacune : la texture ainsi que les positions 3D et les positions 2D de tous les points. Pour pouvoir mélanger ces postures clefs et créer un rendu de toute configuration, chacune est fournie avec un coefficient, précalculé, qui exprime sa «zone d'influence». On verra (équation 2) ce que ces calculs (flottants avec exponentielle) ont de coûteux.	3 textures (512x512, 24bits) + 3 jeux de points (2D et 3D) + 3 réels pour la pondération

Pour pouvoir effectuer aussi le travail d'analyse des images micro-caméra, il faudra en plus disposer :

– de vrais modèles de caméra et de position de l'objet. Il semble cette fois complètement indispensable de disposer d'une vraie transformation paramétrable, pour modéliser la direction de prise de vue de la caméra ou le positionnement relatif du sujet (variables d'une séance à l'autre ?), mais aussi pour minimiser les déformations (non linéaires) de la micro-caméra.

– de deux modèles couleur de la teinte chair et de celle des lèvres. L'un s'applique aux textures et est donc connu en même temps qu'elles, l'autre est lié aux images micro-caméra et est donc susceptible de varier avec l'éclairage et le sujet. Parce qu'il s'applique pixel à pixel, sans notion de voisinage, ce traitement — un changement d'espace couleur paramétré — doit pouvoir s'effectuer efficacement en matériel.

– d'un sous-ensemble de l'objet. Les lèvres et le proche pourtour, plutôt que le modèle complet de visage, peuvent être utilisés pour une analyse plus rapide.

– d'une fonctionnalité de différence (pixel à pixel) d'image, pour quantifier si la synthèse s'approche de l'image (rehaussée par le modèle couleur) de la micro-caméra. Dans l'implémentation logicielle actuelle, la comparaison est très coûteuse et se fait après la synthèse. On imagine qu'une accélération matérielle peut envisager d'autres pistes plus efficaces : composition directe des triangles synthétisés en différence sur l'image de fond, accumulation à la volée du résultat...

C Détails des données

a/ Unités et coordonnées des points 3D

L'origine du repère 3D du monde (celui où sont exprimés tous les points avant projection à l'écran) se situe en bas des incisives supérieures du modèle. Toutes les coordonnées de points 3D sont exprimées en centimètres par rapport à cette origine, avec l'axe des X qui se confond avec l'axe gauche/droite du visage, un axe des Y qui est plutôt orienté haut/bas, et les Z qui codent plutôt la profondeur. En utilisant pour la projection une simple remise à l'échelle des couples (X, Y) des points 3D du modèle on aura donc bien une vue de face (aux éventuels problèmes de visibilité avant/arrière près).

b/ Modèle de visage : points et triangles

À chaque point mobile sont associés une position au repos et autant de vecteurs (3D) de déplacements qu'il y a de paramètres articulatoires. Ainsi la position du point P_i pour des valeurs articulatoires $\delta_{1..m}$ se calculera par la combinaison linéaire suivante :

$$P_i(\delta_{1..m}) = P_{0,i} + \sum_{j \in [1..m]} \delta_j . V_{j,i} \qquad (1)$$

Avec n_{fixe} points statiques et n_{mobile} points animés, il y a donc $n_{fixe} + (m + 1)\, n_{mobiles}$ vecteurs à mémoriser.

Les 30 premiers points mobiles sont toujours des points de lèvres, et leur maillage (pas leur position !) restera toujours le même, quelque soit le locuteur.

Optimisation : Dans l'architecture *Mother*, les triangles des lèvres sont subdivisés pour le rendu en fragments plus petits (par des patches polynomiaux). Ce raffinement — supposé superflu pour le rendu de face que vise le terminal Tempo Valse — ne sera pas présenté ici.

Le nombre de points, mobiles ou statiques, comme le nombre de triangles est par contre susceptible de changer selon le locuteur ou le modèle construit : le modèle de Pierre n'est qu'un exemple et en attendant d'en avoir créé d'autres, il serait risqué de borner leur nombre par les valeurs données en exemple dans la table initiale.

Optimisation : Les points statiques sont assez nombreux dans le cas 3D ; on pourrait envisager de réduire leur nombre pour les seules vues de face, et ce d'autant plus qu'on s'éloigne des zones animées. Mais on aurait alors à texturer des triangles plus grands. Quel serait le bon compromis pour le matériel ?

c/ Modèle de caméra

Le modèle de projection perspectif s'exprime en coordonnées homogènes sous la forme :

$$\begin{pmatrix} w.x \\ w.y \\ depth \\ w \end{pmatrix} = \begin{pmatrix} a_{11} & a_{12} & a_{13} & a_{14} \\ a_{21} & a_{22} & a_{23} & a_{24} \\ a_{31} & a_{32} & a_{33} & a_{34} \\ a_{41} & a_{42} & a_{43} & a_{44} \end{pmatrix} . P \begin{pmatrix} X \\ Y \\ Z \\ 1 \end{pmatrix}$$

C'est à dire que, par exemple, l'abscisse de la projection à l'écran de P vaut :

$$x = \frac{X.a_{11} + Y.a_{12} + Z.a_{13} + a_{14}}{X.a_{41} + Y.a_{42} + Z.a_{43} + a_{44}}$$

En l'absence de Z-buffer, les troisièmes lignes de la matrice et du résultat peuvent être ignorées. Pour une simple projection euclidienne, on n'utiliserait bien sûr plus que les deux premières lignes.

Dans le cas d'OpenGL, les coordonnées écran sont normalisées : quelle que soit sa taille, le centre de l'écran est représenté par (0,0) et les coins sont de la forme (±1, ±1). Les matrices fournies pour les jeux d'exemples respectent cette règle.

Optimisation : Un modèle trop simple de caméra interdira l'utilisation du terminal pour l'analyse ! Il faut pouvoir déplacer la tête (au niveau du modèle de caméra ou par une autre matrice de rotation/ translation qui s'applique aux objets), à cause du positionnement ou du réglage du casque, variables d'une séance sur l'autre. Il faut aussi pouvoir compenser les déformations (la composante perspective au moins) dues à la focale courte de la micro-caméra.

d/ Modèle de texture

Chaque texture est fournie avec la position 2D associée des points du modèle (dans le même ordre : les 30 points de lèvres, puis les autres points mobiles puis les points statiques). S'il n'y avait qu'une seule texture, cela suffirait à spécifier le rendu : chaque triangle à afficher aurait son antécédent dans l'image de référence. Mais on a déjà montré qu'il fallait plusieurs textures (on en utilise typiquement 3) pour avoir un rendu plus réaliste (apparition du pli naso-génien, meilleure définition des lèvres en protrusion...).

Avec plusieurs textures, il faut spécifier comment chacun des (trois) antécédents texturés influence le rendu de chaque triangle d'une posture cible. On ne peut pas se permettre de choisir une seule texture (la plus proche) à un moment donné : cela produirait des discontinuités (temporelles) lors du rendu. Au contraire, on applique à tout moment un mélange des différentes textures, dans des proportions qui varient continûment dans le temps. C'est le rôle du jeu de positions 3D et du coefficient de pondération associés à chaque posture de référence que de déterminer en dynamique le dosage du mélange des pixels de texture :

$$\alpha_j = \frac{\beta_j}{\sum_k \beta_k} \text{ avec } \beta_j = exp\left(-d_j \sum_i d\left(P_i(\delta_{1..m}), P_{j,i}\right)\right) \quad (2)$$

où l'ensemble de points $P_i(\delta_{1..m})$ représente la posture à synthétiser (produite par l'équation (1) à un moment donné d'après les paramètres articulatoires) et où chaque $P_{j,i}$ est le jeu (fixe pour un modèle donné) des points de la $j^{ème}$ position de référence, associée au coefficient de pondération d_j.

Optimisation : On peut envisager de calculer la distance entre deux configurations autrement que par la somme des distances entre tous les points. Calculer la distance entre les paramètres articulatoires ne semble a priori pas très pertinent (les articulateurs n'ont pas une influence homogène : en protrusion, la peau est plus déformée que par les simples mouvements de mâchoire). Un sous-ensemble de points serait peut-être suffisant si l'implémentation matérielle de l'équation 2 pose problème ?

Optimisation : Alors que les coefficients α_j s'appliquent à tous les pixels de la texture, il n'est pas possible de mélanger efficacement (pixel à pixel, en balayage *raster*) les images pour ne faire qu'un seul rendu texturé. Cela serait possible si les textures n'étaient pas des images réelles, mais soient pré-déformées (lors de la réception dans le décodeur, ou à l'envoi) afin que les coordonnées 2D soient identiques pour

tous les visèmes. Cette piste est en cours d'étude, en même temps que les textures cylindriques. Une telle solution présente aussi l'intérêt de maximiser la zone utile d'une texture (moins de fond) ou de la spécialiser à une architecture de rendu (grille de paramètres réguliers, en puissance de 2...). Quelles sont les opportunités au niveau du matériel ?

D Ressources fournies

Deux programmes — compilables et exécutables — sont proposés. Les exemples de données d'entrée et de sortie fournis doivent permettre de reproduire nos résultats ou de les comparer à d'éventuelles versions 2D de ces algorithmes.

Le premier programme, *minim_wire.c,* réalise un rendu sans texture ni *Z-buffer* (triangles en fil de fer, tous visibles). Il permet de tester les modèles de caméra, de géométrie et le modèle articulatoire.

Le second, *minim_tex.c,* est plus complexe et inclut le rendu multi-textures. Il hérite des nombreuses contraintes imposées par *Open GL* : les textures sont tête-bêches et ont pour tailles des puissances de 2, le rendu doit être fait en plusieurs passes en modulant la transparence de chacune... Une implémentation matérielle pourrait se permettre des pistes plus simples et plus efficaces, notamment si l'on borne à 3 le nombre maximal de textures.

a/ Librairies utilisées

Très simplifiés par rapport à la plate-forme de démonstration *Mother*, évitant les optimisations, les programmes fournis n'utilisent en plus du C standard que quelques unes des fonctionnalités d'*Open GL.* Si l'on fait abstraction des (nombreuses) commandes de configuration, il s'agit principalement de déclarer la caméra et de transférer les coordonnées (3D et texture) des points des triangles. On trouvera par exemple une documentation en ligne des fonctions *GL* sur :

http://www.dgp.toronto.edu/people/van/courses/opengl/opengl_index_alpha.html

Pour la recompilation, une librairie compatible *Open GL* sera nécessaire. On trouve sur ces systèmes :

– *Linux* : *Mesa GL* (solution générique et gratuite multi plate-formes) est normalement inclus dans les distributions récentes (sans forcément utiliser l'accélération matérielle des cartes 3D...),

– *Windows* : les versions récentes de *Direct X* incluent, selon la carte vidéo de la machine, une émulation logicielle ou une passerelle vers du *GL* accéléré,

– *Solaris* : une version *SUN* de *GL*, optimisée pour leurs stations et cartes 3D, est aussi disponible en *packages* prêts à l'installation.

Afin de pouvoir afficher directement leurs résultats, les programmes fournis utilisent quelques appels à la librairie *GLUT*. Cette petite librairie n'est généralement pas installée en standard mais est par exemple disponible (sources ou versions pré-compilées pour différents OS + fichiers .h) sur le site de *Silicon Graphics :* http://reality.sgi.com/mjk/glut3/glut3.html

b/ Utilisation des fichiers de données

Hormis les images (au format *PNM*), tous les fichiers sont en *ASCII* (avec des commentaires le plus souvent). Les lettres initiales d'un fichier différencient son type et rappellent la syntaxe de l'option pour le charger : *minim_tex -art art_isi.txt -cam cam_mc.txt -tex tex_a_afa_upu.txt* demande le rendu d'un visème *[isi]* en mode micro-caméra à partir de trois textures clefs (a, afa et upu). Par défaut, un certain nombre de fichiers paramètres (affichés lors du lancement, avec des valeurs quantitatives) sont chargés et *minim_tex* suffit à produire un résultat visible. Enfin, il est possible de sauver les résultats calculés par *GL,* par exemple l'image au format *PNM* (*-pnm out.pnm*) ou d'afficher l'aide par *minim_tex -help*.

E. Et l'analyse ?

Les algorithmes d'analyse dernièrement développés à l'*ICP* sont manifestement trop lourdes pour le matériel envisagé. La version initiale (modèle couleur et zone limitée autour de la bouche), telle que décrite et déjà analysée par Éric Petit reste celle que nous proposons pour le terminal. La discussion vers d'autres simplifications reste bien sûr ouverte, selon les propositions des équipes du matériel.