Textures volumiques auto-zoomables pour une stylisation temporellement cohérente en temps réel

Pierre Bénard, Adrien Bousseau, Joëlle Thollot

AFIG 2008 / Toulouse
IRIT Presse

Textures volumiques auto-zoomables pour une stylisation temporellement cohérente en temps réel

Pierre Bénard, Adrien Bousseau, Joëlle Thollot

Grenoble Universités, INRIA

Figure 1: *Palette de styles (collage, aquarelle et rendu binaire) temporellement cohérents obtenus avec nos textures volumiques auto-zoomables (colonne de droite).*

Abstract

Les méthodes de stylisation dont l'objectif est de représenter une scène 3D dynamique avec des marques 2D comme des pigments ou des coups de pinceau, sont généralement confrontées au problème de la cohérence temporelle. *Dans cet article, nous présentons une méthode de stylisation en temps réel et temporellement cohérente basée sur des textures : les* ***textures auto-zoomables****. Une texture auto-zoomable est une texture plaquée sur les objets 3D et enrichie d'un nouveau mécanisme de* ***zoom infini****. Ce mécanisme maintient une taille quasi constante en espace image des éléments de texture. Lors de la stylisation, ce mécanisme renforce l'apparence 2D des marques de style tout en restant fidèle au mouvement 3D des objets représentés. Nous illustrons cette propriété par une variété de styles comme l'aquarelle ou le rendu par marques binaires. Bien que notre technique de zoom infini puisse être utilisée aussi bien pour des textures 2D que 3D, nous nous sommes attachés dans cet article au cas 3D (que nous appelons* ***textures volumiques auto-zoomables****), ce qui évite la définition d'une paramétrisation des surfaces 3D. En intégrant notre méthode à un moteur de rendu, nous validons la pertinence de ce compromis entre qualité et rapidité.*

Keywords: informatique graphique, rendu non-photoréaliste, rendu temps réel, textures volumiques

1. Introduction

De plus en plus de jeux vidéo tentent de se démarquer de la traditionnelle quête de l'ultra-réalisme pour des rendus plus stylisés. Plusieurs jeux récents se distinguant par leur originalité graphique (style BD pour *XIII*, Aquarelle/Sumi-e pour *Ōkami* ou prochainement « Illustrative Art Style » pour *Prince of Persia*) ont reçu un très bon accueil de la part des joueurs. Cette nouvelle tendance implique de nouvelles difficultés techniques induites par l'intégration de styles de rendu non-photoréalistes complexes dans les moteurs de jeux.

Dans cet article, nous présentons une méthode de stylisation simple et rapide avec une forte cohérence temporelle. En effet, la principale difficulté, partagée par tous les algorithmes de stylisation, consiste à maintenir la cohérence temporelle de la stylisation lors de l'animation d'une scène 3D. L'objectif est de représenter un mouvement 3D (celui de la caméra ou des objets de la scène) tout en préservant les caractéristiques 2D des marques de style (pigments, coups de pinceaux…) du médium simulé. Pour remplir parfaitement cet objectif sans produire d'artefacts visuels, la stylisation d'une animation doit satisfaire simultanément trois contraintes contradictoires. Premièrement, afin de préserver l'apparence 2D du médium, les marques de style devraient avoir une taille constante et une distribution uniforme dans l'image. Deuxièmement, pour éviter des effets de glissement du style sur la scène (effet *rideau de douche*), les marques de style devraient respecter exactement le mouvement 3D des objets qu'elles recouvrent. Enfin, pour éviter des effets de clignotement ou de variation aléatoire des marques, une continuité temporelle suffisante entre chaque image de l'ani-

mation serait nécessaire. De nombreuses solutions ont été proposées afin de trouver un compromis acceptable à ces contraintes [GG01, SS02]. Elles reposent cependant, pour la plupart, sur des algorithmes coûteux et des structures de données peu compatibles avec les pipelines de rendus existants.

Dans cet article nous présentons les *textures auto-zoomables* qui facilitent l'intégration de stylisation temporellement cohérente dans les pipelines de rendu temps réel. Notre méthode s'adresse plus particulièrement au problème de la stylisation des régions de couleur d'une image, le traitement des contours n'étant pas abordé. Notre approche se base sur des textures comme support de la stylisation, afin d'éviter la mise en place des structures de données complexes. Cela rend notre approche particulièrement adaptée aux médiums présentant une texture caractéristique comme l'aquarelle, le fusain... De plus, la gestion optimisée des textures par les cartes graphiques modernes rend notre méthode particulièrement adaptée au rendu temps réel.

Contrairement au texturage traditionnel, notre approche bénéficie d'un mécanisme de **zoom infini** produisant une distribution uniforme des éléments de texture à l'écran, quelle que soit la distance des objets à la caméra. Ce mécanisme préserve la majorité des caractéristiques 2D du médium traditionnel tout en assurant la cohérence temporelle lors de la navigation dans un univers 3D. Ce mécanisme de zoom infini peut être appliqué aux textures 2D comme 3D. Cependant, nous avons choisi de développer dans cet article le cas des textures 3D (les **textures volumiques auto-zoomables**), ce qui évite le calcul complexe ou la définition manuelle d'une paramétrisation à la surface des objets 3D. Ces textures volumiques peuvent facilement être produites soit procéduralement, soit par synthèse à partir d'exemples 2D.

Afin de démontrer l'efficacité de cette approche, nous proposons son intégration au moteur de rendu OGRE[†]. Les mesures de performance réalisées dans cet environnement indiquent un impact faible de la méthode sur la vitesse d'affichage pour des scènes complexes. Les différents styles présents dans cette implémentation illustrent l'utilité et la simplicité de la méthode pour le rendu temps réel non-photoréaliste.

2. Travaux précédents

Afin de répondre à la contradiction entre le respect du style 2D et celui du mouvement 3D, Meier [Mei96] propose dans son article précurseur de découpler l'apparence des marques de style et leur mouvement. Pour conserver une apparence 2D, les marques de style (dans ce cas des coups de pinceau) sont dessinés avec des imposteurs de taille constante. Pour préserver le mouvement 3D, chaque marque est associée à un point d'ancrage sur l'objet 3D qu'elle stylise. Bien que cette approche ait été améliorée et étendue à de nombreux styles (peinture [Dan99, VBTS07], *stippling* [PFS03], aquarelle [BKTS06a]), la structure de donnée nécessaire à la gestion des points d'ancrage et le surcoût introduit par le rendu individuel de chaque marque rendent son utilisation dans des moteurs de jeux vidéo peu praticable.

Ces limitations, dues à la gestion individuelle des marques de style, peuvent être évitées en traitant les marques globalement, sous forme de textures. La méthode de *Dynamic Canvas* [CTP*03] consiste à appliquer une texture de papier sur l'écran afin de styliser en temps réel un environnement 3D dans un contexte de navigation interactive. Le cœur de cette approche consiste à contrebalancer les rotations 3D de la caméra par des transformations 2D de la texture. Les translations en profondeur de la caméra sont elles compensées par un mécanisme de zoom infini qui maintient une taille quasi constante de la texture à l'écran. Bien qu'offrant un très bon compromis entre l'apparence 2D du papier et les mouvements 3D de la caméra, cette approche est limitée à la navigation dans des scènes statiques. Coconu et coll. [CDH06] et Breslav et coll. [BSM*07] adoptent une approche similaire en appliquant à une texture de stylisation la transformation 2D approximant au plus proche la transformation 3D de l'objet à représenter. Tout comme *Dynamic Canvas*, ces méthodes offrent une très bonne préservation de l'apparence 2D des marques de style, la texture n'étant déformée qu'en espace image. En contrepartie, l'approximation introduite par l'application d'une transformation 2D à la place d'une transformation 3D peut produire des effets de glissement pour des mouvements 3D extrêmes. Une autre approche du zoom infini a été proposée récemment par Han et coll. [HRRG08]. Ils développent un algorithme de synthèse de texture 2D multi-échelle qui génère des éléments de textures durant le zoom. Bien que l'illusion produite par cette dernière méthode soit plus précise pour certaines textures, elle semble trop lente pour le rendu temps réel de scènes complexes. Dans cet article, nous étendons le mécanisme de zoom infini de *Dynamic Canvas* aux textures en espace objet qui permettent la stylisation d'objets dynamiques en temps réel sans aucun glissement.

Ces différentes approches à base de textures sacrifient le respect du mouvement 3D au profit de l'apparence 2D, mais le compromis inverse est également possible. C'est le choix fait par les méthodes d'*art map* [KLK*00] et *tonal art map* [PHWF01] qui plaquent directement les textures de stylisation sur les objets 3D de la scène. Si elles sont plaquées de façon naïve, les marques de style restent certes parfaitement attachées aux objets mais sont fortement déformées par la projection perspective. La solution des *art maps* consiste à utiliser le mécanisme du *mip-mapping* pour adapter l'échelle de la texture en fonction de la distance à l'écran. Cette approche permet de maintenir une taille de marques presque constante dans l'image, tout en tirant profit de la gestion optimisée des *mipmaps* par les cartes graphiques. Freudenberg et coll. [FMS01] ont prouvé la performance de ces méthodes

[†] http://www.ogre3D.org

en intégrant un rendu non-photoréaliste de type *art map* dans le moteur de jeux *Fly3D*.

Ces approches souffrent cependant de quelques limitations. Tout d'abord, afin d'obtenir des transitions cohérentes entres les différents niveaux d'une *art map*, chaque niveau doit contenir les marques du niveau précédent [PHWF01]. La prise en compte de cette contrainte limite l'application des *art maps* aux styles binaires de type *hatching* ou *stippling*. Ensuite, l'utilisation de *mipmaps* n'est efficace que lorsque l'objet s'éloigne de l'écran (rétrécissement de la texture). Lorsque l'objet s'approche trop près de la caméra, la texture subit un simple agrandissement. Les *mipmaps* ne permettent pas non plus la correction des déformations perspectives dues à l'orientation de la surface par rapport à l'écran. Les *ripmaps* [KLK*00] permettent de corriger une partie de ces déformations mais ne sont pas aussi bien supportées par les cartes graphiques modernes. Enfin, le plaquage des textures nécessite la définition d'une paramétrisation à la surface des objets. Praun et coll. [PHWF01] proposent d'automatiser cette paramétrisation grâce aux *lapped textures*, mais ceci nécessite la mise en place d'une structure de données supplémentaire.

Tout comme les *art maps*, la méthode des textures autozoomables décrite dans cet article corrige uniquement les déformations dues à la distance de l'objet à l'écran. Cependant, contrairement au *mipmapping*, le mécanisme de zoom infini permet l'obtention de marques de taille constante pour n'importe quelle distance. De plus, en combinant plusieurs échelles de textures à la fois, le zoom infini assure des transitions douces quelle que soit la texture utilisée. Enfin, notre méthode étant basée sur des textures volumiques, elle ne nécessite aucune définition de paramétrisation supplémentaire des surfaces 3D, qui est souvent complexe pour limiter les contractions/dilatations ou, dans le cas d'atlas de textures, pour gérer les raccordements sans discontinuité.

3. Textures volumiques auto-zoomables

Dans cet article, nous désignons par *zoom infini* sur un signal la croissance infinie de sa fréquence perceptible. Dans le cas d'un son, cela correspond à une montée dans les aigus audibles à l'infini, comme l'a montré Shepard [She64]. Dans le cas d'une image 2D, Cunzi et coll. [CTP*03] et Han et coll. [HRRG08] considèrent que ce processus correspond à l'apparition infinie de nouveaux détails visibles. Ils ont cependant développé des solutions diamétralemnt opposées, comme nous l'avons décrit dans la section précédente.

Nous proposons d'étendre le mécanisme de zoom infini de Cunzi et coll. aux textures en espace objet (section 3.1). Notre implémentation (section 3.3) montre que cette approche, très peu coûteuse en temps de calcul et relativement peu en place mémoire, est parfaitement intégrable à un moteur de rendu de jeux vidéo comme OGRE. Les résultats obtenus (section 3.4) illustrent les avantages de cette méthode temps réel pour des scènes dynamiques profondes, sans dégradation importante de l'apparence 2D du médium.

3.1. Zoom infini en espace objet

L'objectif contradictoire du mécanisme de zoom infini est de maintenir une taille globalement constante de la texture volumique à l'écran, tout en préservant l'impression de grossissement / rétrécissement des éléments de texture lors du déplacement en profondeur de la caméra.

La méthode de *Dynamic Canvas* [CTP*03] utilise un mécanisme de zoom infini 2D basé sur la *fractalisation* d'une texture plaquée sur l'écran, de la même façon que Perlin [Per85] pour une fonction procédurale. Dans cette approche, n octaves (fréquence doublée) du motif original sont mélangées de façon linéaire afin de créer une image autosimilaire. Quand l'observateur se déplace en profondeur, la fréquence des octaves est décalée de façon continue pour produire une illusion de zoom. Ce mécanisme procure l'illusion d'une texture grossissant à l'infini tout en conservant une taille quasi constante à l'écran. Cependant, le calcul du facteur de zoom dépend d'une distance subjective entre un plan virtuel et la caméra. Cette distance étant fixée avant la navigation, elle ne tient pas compte de la profondeur réelle de la scène. Par conséquent, il peut se produire des glissements de la texture sur les objets qui ne sont pas exactement à cette distance. Nous proposons dans cet article l'extension de ce principe de fractalisation aux textures en espace objet, en nous affranchissant de l'approximation de la distance subjective.

Suivant cette approche, une **texture volumique autozoomable** est la somme pondérée de n octaves Ω_i du motif volumique d'origine. Notez que le mélange de plusieurs octaves assure la continuité temporelle du zoom au prix d'une perte de contraste, comme discuté en section 5. Nous avons défini empiriquement qu'il fallait au moins $n = 4$ octaves pour tromper la perception humaine, tandis qu'utiliser un nombre supérieur d'octaves n'a pas d'impact réel sur l'impression de continuité mais dégrade sensiblement la texture. Chaque objet 3D est ensuite plongé dans une texture volumique autozoomable. Pour préserver un sentiment convainquant de zoom, chaque octave est sujette aux règles de la projection perspective : une octave apparaitra deux fois plus grande quand elle sera deux fois plus proche de la caméra. Il faut néanmoins s'assurer de maintenir une taille quasi constante en espace image. Pour cela, nous introduisons la notion de **cycle de zoom** qui a lieu chaque fois que la taille de l'octave a doublé. Chaque octave est alors remplacée par l'octave suivante dans le cycle et une nouvelle octave de fréquence supérieure est créée, comme illustré sur la figure 2.

3.2. Algorithme proposé

En pratique, un objet est plongé dans une texture volumique auto-zoomable en dérivant ses coordonnées de texture

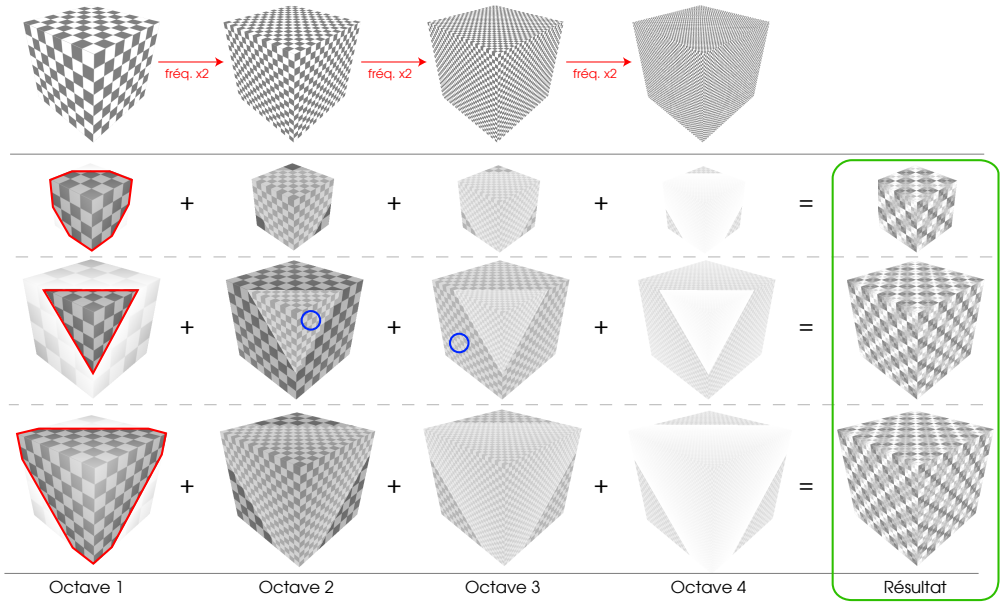


Figure 2: *Illustration du mécanisme du zoom infini tridimensionnel sur un échiquier 3D. Observez la fréquence globalement invariante du mélange pondéré (dernière colonne). La ligne rouge matérialise la limite entre deux cycles de zoom consécutifs. Notez les correspondances fréquentielles d'une octave à l'autre pour deux cycles de zoom consécutifs (cercles bleus).*

des coordonnées de ses sommets dans le repère de l'objet. En considérant chaque sommet indépendemment, nous résolvons la limitation de *Dynamic Canvas* qui approxime la scène entière par un plan. En pratique, nous utilisons un seul cube de texture que nous échantillonnons à différentes fréquences pour obtenir 4 octaves. Les coordonnées de texture (u,v,w) de chaque sommet $p(x,y,z)$, pour chaque octave Ω_i, sont données par :

$$(u,v,w)_i = 2^{i-1}(x,y,z)/2^{\lfloor \log_2(z_{cam}) \rfloor}$$

Ce calcul prend en compte à la fois le facteur d'échelle 2^{i-1} entre les octaves (chaque octave est deux fois plus grande que la suivante), et entre les cycles de zoom $2^{\lfloor \log_2(z_{cam}) \rfloor}$ (chaque octave est deux fois plus grande que l'octave correspondante au prochain cycle).

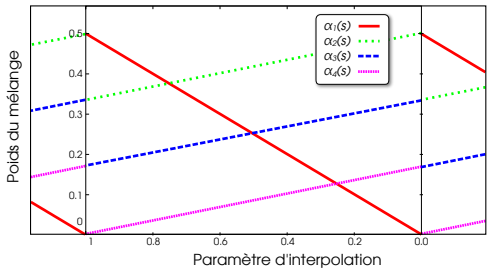


Figure 3: *Évolution des poids du mélange au cours d'un cycle de zoom. Notez comment le schéma d'interpolation permet de boucler, les poids en début et fin de cycle se raccordant, ce qui assure un nombre infini de cycles*

Durant un cycle de zoom, chaque octave est pondérée par un poids $\alpha_{i=1\ldots n}(s) \in [0,1]$ avec s le paramètre d'interpolation entre le début et la fin d'un cycle :

$$s = \log_2(z_{cam}) - \lfloor \log_2(z_{cam}) \rfloor \in [0,1]$$

Afin d'obtenir une transition douce entre les cycles de zoom, les poids α_i doivent respecter plusieurs contraintes (cf. figure 3). Premièrement, pour éviter l'apparition ou disparition soudaine d'éléments de texture, la première octave doit apparaître au début d'un cycle alors que la dernière octave doit disparaître à la fin :

$$\alpha_1(0) = 0 \text{ et } \alpha_n(1) = 0$$

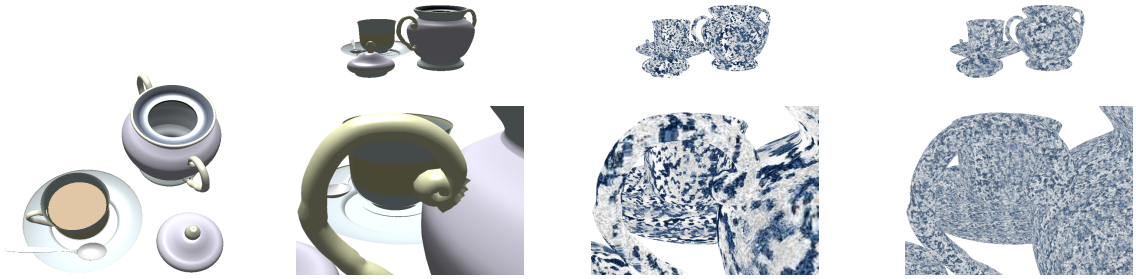
Deuxièmement, pour les octaves intermédiaires, le poids de chaque octave en fin de cycle doit être égal au poids de son octave suivante au début du cycle :

$$\alpha_i(1) = \alpha_{i+1}(0) \quad \forall i \in \{1,\ldots,n-1\}$$

Enfin, les poids doivent se sommer à 1 pour préserver une intensité constante. En pratique, nous utilisons un fondu linéaire – rapide à calculer et cohérent avec la linéarité du zoom – avec les poids suivants :

$$\begin{array}{lllllll} \alpha_1(s) & = & s/2 & \quad & \alpha_2(s) & = & 1/2 - s/6 \\ \alpha_3(s) & = & 1/3 - s/6 & & \alpha_4(s) & = & 1/6 - s/6 \end{array}$$

Le mécanisme complet de zoom est illustré sur la figure 2. Le passage d'un cycle de zoom à l'autre est délimité par la ligne rouge. Remarquez la correspondance fréquentielle entre les octaves i et $i+1$ (sur une même ligne) pour des



(a) vue plongeante (b) éclairage de Phong (c) texture 3D traditionnelle (d) texture volumique auto-zoomable

Figure 4: *Comparaison de nos textures volumiques auto-zoomables avec les techniques traditionnelles à échelle fixe de texturage 2D et 3D. Notez la taille globalement constante des éléments de texture – quelque soit le facteur de zoom – et la correction de la déformation perspective qu'elle produit.*

cycles de zoom successifs qui assurent, après mélange, la continuité du zoom.

3.3. Implémentation

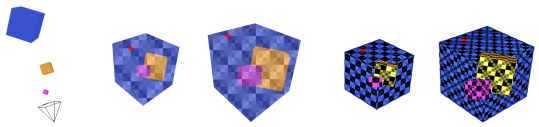
Afin de satisfaire la contrainte de rendu temps réel, nous avons implémenté cet algorithme de zoom infini en GLSL[‡] (voir le code source des shaders en annexe A). Le *vertex shader* est chargé de fixer les coordonnées 3D de texture, celles-ci pouvant directement correspondre à la position 3D du sommet dans le repère de l'objet ou, pour les objets déformables animés, ses coordonnées dans une position initiale de l'objet avant déformation. L'utilisateur peut également spécifier un facteur d'échelle supplémentaire définissant la taille globale de la texture par rapport à l'objet représenté. Nous calculons la distance entre un sommet et la caméra comme sa coordonnée en z dans le repère caméra. Le *fragment shader* réalise le fondu linéaire des 4 octaves de la texture volumique en fonction du facteur de zoom, selon les formules détaillées précédemment.

Les textures volumiques nécessaires à notre approche peuvent être obtenues selon deux procédés : procéduralement tout d'abord (bruit de Perlin [Per85, Ola05], par exemple), sous forme de shaders ; par synthèse à partir d'un exemple 2D ensuite (dans notre cas, la méthode *Solid texture synthesis from 2D exemplars* [KFCO*07] de Kopf et coll.), le cube de texture (typiquement, d'une résolution de $128 \times 128 \times 128$ pixels RGB, c'est-à-dire 6 MB sans compression au format *DirectDraw Surface*) devant alors être stocké dans la mémoire de la carte graphique.

3.4. Résultats

Nous comparons en figure 4 nos textures volumiques auto-zoomables avec les approches traditionnelles de texturage 2D et 3D. Dans le cas du texturage 2D et 3D, la taille des éléments de texture varie avec la profondeur à cause de la projection perspective. À l'inverse, avec notre mécanisme de zoom infini, les éléments de texture gardent un taille globalement constante en espace image, quel que soit le facteur de zoom.

En comparaison avec *Dynamic Canvas*, notre approche souffre de la déformation perspective quand la surface est presque tangente à la direction de vue, et de discontinuités aux bordures d'occlusion. Cependant ces artefacts sont restreints par le mécanisme du zoom infini qui limite la profondeur de la scène. D'un autre côté, comme illustré par le point rouge sur la figure 5, des glissements ont lieu avec Dynamic Canvas, alors que les éléments de texture suivent parfaitement le mouvement 3D des objets dans notre cas. Par ailleurs, lors de la navigation dans la scène 3D, on se rend pleinement compte de la bonne cohérence temporelle du zoom pendant le mouvement et du respect exact du mouvement 3D de la caméra. Notez que les approches de Coconu et coll. [CDH06] et de Breslav et coll. [BSM*07] souffriraient du même problème de glissement.



(a) Vue plongeante (b) Dynamic Canvas (c) Textures volumiques auto-zoomables

Figure 5: *Illustration de Dynamic Canvas : observez le glissement de la texture soulignée par le point rouge.*

Les principaux avantages de notre méthode sont la simplicité de son intégration dans les moteurs de jeux vidéo existants et ses performances temps réel. Notre implémentation dans OGRE entraîne un léger surcoût de moins de 10% en comparaison avec un éclairage lambertien par pixel (calculé dans un *shader*). Pour la scène complexe de la ville de la figure 1 (135k tris), le *framerate* passe de 70 à 65 images

[‡] OpenGL Shading Language : `http://www.opengl.org/documentation/glsl/`

par seconde, avec un *Core 2 Duo* 6600 à 2.4GHz et une *GeForce 8800 GT* pour une résolution de 1280×1024 pixels. Cela rend nos textures volumiques auto-zoomables tout à fait adaptées pour les jeux vidéo.

4. Application à la stylisation cohérente

Disposant désormais d'une brique de base solide pour la représentation d'un médium, nous proposons dans cette section différentes stylisations temps réel. Ces styles sont soit l'extension de pipelines existants – l'information de motif étant remplacée par une ou plusieurs textures volumiques auto-zoomables – soit totalement inédits, car difficile à produire sans notre approche. Notez qu'une étape de stylisation des contours des objets pourrait être ajoutée à celui du remplissage, mais cela dépasse le cadre cet article.

4.1. Aquarelle

Nous avons choisi d'étendre l'approche de Bousseau et coll. [BKTS06b], car leur processus de stylisation est basé sur une série de traitements d'image faisant intervenir des textures 2D de papier et de pigments. Nous remplaçons ces textures par nos textures volumiques auto-zoomables. Ainsi, chaque effet aquarelle lié aux textures (pigment, *wobbling*) reste complètement cohérent au cours de l'animation. Comme l'illustre la figure 1, les textures volumiques offrent une grande richesse de médium.

Nous avons comparé notre approche avec *l'advection de texture* de Bousseau et coll. [BNTS07] pour le rendu aquarelle de vidéos. Notez que cette méthode requiert la connaissance de l'intégralité de la séquence d'animation, ce qui la rend inutilisable pour une application temps réel. L'advection de texture produit des aquarelles animées avec quasiment aucune déformation ou glissement, ce qui leur donne une apparence bidimensionnelle forte pour des images fixes. Cependant, durant l'animation, le mouvement des éléments de texture – perçu et interprété comme tridimensionnel par l'observateur – a tendance à diminuer cette apparence 2D. Par conséquent nous pensons que les déformations perspectives produites par notre approche constituent une faible dégradation en comparaison avec la perception forte de la perspective induite par le mouvement 3D, en particulier en considérant le gain conséquent de performance.

4.2. Rendu binaire noir et blanc

Nous obtenons un rendu noir et blanc par marques binaires (points, hachures…) en utilisant une approche très similaire à celle de Durand et coll. [DOM*01], jusqu'alors limitée à la stylisation d'images 2D. Dans cette méthode, les marques binaires sont obtenues par seuillage d'un champ de hauteur – sous la forme d'une texture en niveaux de gris – en fonction du ton de destination souhaité. Nous proposons d'utiliser une texture volumique auto-zoomable en niveaux de gris comme champ de hauteur. La figure 1 illustre la diversité des styles binaires combinables dans une seule image, en attribuant simplement des textures différentes à chaque objet. Il est également à noter que la méthode de zoom infini s'adapte parfaitement à ce type de rendu, les nouvelles marques binaires apparaissant progressivement pour maintenir le ton défini par l'éclairage.

Notez que, contrairement aux méthodes existantes pour l'illustration [HZ00, PHWF01], notre approche à base de textures volumiques ne peut pas orienter les marques binaires suivant les directions de courbure principales de l'objet. Cette limitation est discutée en section 5.

4.3. Collage

Nous proposons un nouveau processus de stylisation que nous appelons *collage*. Ce nouveau style tire parti de la grande diversité de textures que nous pouvons synthétiser par l'exemple [KFCO*07]. Traditionnellement le collage consiste à créer une image en assemblant des bandes de papier découpées possédant des couleurs et textures diverses. Nous copions ce style en assignant différentes textures volumiques auto-zoomables à chaque ton de l'image (obtenus par un modèle d'éclairage discrétisé proche du *toon shading* [LMHB00]). Pour renforcer l'aspect papier du collage, une bordure blanche et un effet de *wobbling* sont ajoutés entre les bandes de toon (figure 1).

5. Discussion et travaux futurs

5.1. Mécanisme de zoom infini

La principale limitation de notre méthode, partagée par *Dynamic Canvas* [CTP*03], l'advection de texture [BNTS07] et, dans une certaine mesure, par les approches à base de *mipmaps* [KLK*00, PHWF01, FMS01] est le fondu linéraire entre plusieurs octaves qui entraîne l'apparition de nouvelles fréquences et une perte de contraste globale par rapport à la texture originale. Par conséquent, les textures les plus structurées sont visiblement altérées par le mécanisme de zoom infini (la texture d'échiquier étant un cas extrême). Nous prévoyons d'adresser cette limitation dans des travaux futurs. Une solution possible serait de remplacer le mélange linéaire qui préserve mieux les éléments de texture caractéristiques. Issue des travaux sur le *compositing* d'images [GVWD06] et en liaison avec l'extraction du motif d'une texture sous forme de *feature map* [WY04], des fonctions de mélange non-linéaires, réhaussant localement le contraste, peuvent être envisagées.

5.2. Plaquage de textures

Nous avons choisi de développer le mécanisme de zoom infini pour des textures volumiques car cela évite la définition d'une paramétrisation souvent complexe. Cependant, la contrepartie de ce choix est que les textures sont décorrélées

des surfaces 3D. Cela peut être vu comme une limitation pour certains styles qui bénéficient de l'orientation des éléments de texture selon la surface. Un exemple de ce type de style est l'illustration scinetifique, pour lequel il a été montré qu'orienter les marques binaires selon les directions principales de courbure de la surface souligne la forme des objets [HZ00, PHWF01]. Néanmoins, le mécanisme de zoom infini décrit dans cet article est indépendant de la dimension de la texture et peut très bien être appliqué à des textures 2D, si une paramétrisation adéquate est disponible. Il peut alors être vu comme une extension des *tonal art maps*, où le zoom infini est utilisé à la place des *mipmaps*.

Une limitation, partagée par toutes les méthodes basées sur des textures, concerne le texturage des objets très déformables, comme l'eau. Dans ce cas, définir les coordonnées de texture comme la position des sommets de l'objet non-déformé crée des dilatations/contractions additionnelles qui diminuent leur aspect bidimentionnel.

6. Conclusion

Nous avons présenté une approche permettant la création de textures volumiques auto-zoomables. Dès lors qu'un objet est plongé dans ces textures, un mécanisme de zoom infini maintient leur apparence 2D et assure une grande cohérence temporelle de la stylisation.

De nombreux autres styles pourraient tirer parti de ces textures. Ainsi, toutes les lignes caractéristiques qui restent fixes à la surface d'un objets 3D (*ridges* [IFP95,OBS04], *apparent ridges* [JDA07], *demarcating curves* [KST08]) pourraient être stylisées en utilisant notre approche.

References

[BKTS06a] BOUSSEAU A., KAPLAN M., THOLLOT J., SILLION F. : Interactive watercolor rendering with temporal coherence and abstraction. In *NPAR '06, Proceedings of the 4th international symposium on Non-photorealistic animation and rendering* (2006), ACM.

[BKTS06b] BOUSSEAU A., KAPLAN M., THOLLOT J., SILLION F. : Interactive watercolor rendering with temporal coherence and abstraction. In *NPAR '06, Proceddings of the 4th international Symposium on Non-Photorealistic Animation and Rendering* (2006), ACM.

[BNTS07] BOUSSEAU A., NEYRET F., THOLLOT J., SALESIN D. : Video watercolorization using bidirectional texture advection. In *SIGGRAPH '07, Computer Graphics Proceedings* (2007), ACM, p. 104.

[BSM*07] BRESLAV S., SZERSZEN K., MARKOSIAN L., BARLA P., THOLLOT J. : Dynamic 2d patterns for shading 3d scenes. *SIGGRAPH '07, Computer Graphics Proceedings 26*, 3 (2007), 20.

[CDH06] COCONU L., DEUSSEN O., HEGE H.-C. : Real-time pen-and-ink illustration of landscapes. In *NPAR '06, Proceedings of the 4th international symposium on Non-photorealistic animation and rendering* (2006), ACM, pp. 27–35.

[CTP*03] CUNZI M., THOLLOT J., PARIS S., DEBUNNE G., GASCUEL J.-D., DURAND F. : Dynamic canvas for immersive non-photorealistic walkthroughs. In *GI '03, Proceedings of Graphics Interface* (2003), A K Peters, LTD.

[Dan99] DANIELS E. : Deep canvas in disney's tarzan. In *SIGGRAPH '99 : ACM SIGGRAPH 99 Conference abstracts and applications* (1999), p. 200.

[DOM*01] DURAND F., OSTROMOUKHOV V., MILLER M., DURANLEAU F., DORSEY J. : Decoupling strokes and high-level attributes for interactive traditional drawing. In *EGSR '01, Proceedings of the 12th Eurographics Workshop on Rendering Techniques* (London, UK, 2001), Springer-Verlag, pp. 71–82.

[FMS01] FREUDENBERG B., MASUCH M., STROTHOTTE T. : Walk-Through Illustrations : Frame-Coherent Pen-and-Ink Style in a Game Engine. *Proceedings of Eurographics '01 20*, 3 (2001), 184–191.

[GG01] GOOCH B., GOOCH A. : *Non-Photorealistic Rendering*. AK Peters Ltd, july 2001.

[GVWD06] GRUNDLAND M., VOHRA R., WILLIAMS G. P., DODGSON N. A. : Cross dissolve without cross fade : preserving contrast, color and salience in image compositing. In *Computer Graphics Forum : Proceedings of Eurographics* (2006), vol. 25.

[HRRG08] HAN C., RISSER E., RAMAMOORTHI R., GRINSPUN E. : Multiscale texture synthesis. *SIGGRAPH '08, Computer Graphics Proceedings 27*, 3 (2008), 51.

[HZ00] HERTZMANN A., ZORIN D. : Illustrating smooth surfaces. In *SIGGRAPH '00 : Proceedings of the 27th annual conference on Computer graphics and interactive techniques* (New York, NY, USA, 2000), ACM Press/Addison-Wesley Publishing Co., pp. 517–526.

[IFP95] INTERRANTE V., FUCHS H., PIZER S. : Enhancing transparent skin surfaces with ridge and valley lines. In *VIS '95 : Proceedings of the 6th conference on Visualization '95* (Washington, DC, USA, 1995), IEEE Computer Society, p. 52.

[JDA07] JUDD T., DURAND F., ADELSON E. H. : Apparent ridges for line drawing. *ACM Trans. Graph. 26*, 3 (2007), 19.

[KFCO*07] KOPF J., FU C.-W., COHEN-OR D., DEUSSEN O., LISCHINSKI D., WONG T.-T. : Solid texture synthesis from 2d exemplars. *SIGGRAPH '07, Computer Graphics Proceedings 26*, 3 (2007), 2.

[KLK*00] KLEIN A. W., LI W. W., KAZHDAN M. M., CORREA W. T., FINKELSTEIN A., FUNKHOUSER T. A. : Non-photorealistic virtual environments. In *SIGGRAPH '00, Computer Graphics Proceedings* (2000), ACM, pp. 527–534.

[KST08] KOLOMENKIN M., SHIMSHONI I., , TAL A. : Demarcating curves for shape illustration. *ACM Transactions on Graphics (Proc. SIGGRAPH ASIA)"* (Dec. 2008).

[LMHB00] LAKE A., MARSHALL C., HARRIS M., BLACKSTEIN M. : Stylized rendering techniques for scalable real-time 3d animation. In *NPAR '00, Proceedings of the 1st international symposium on Non-photorealistic animation and rendering* (2000), ACM, pp. 13–20.

[Mei96] MEIER B. J. : Painterly rendering for animation. In *SIGGRAPH '96, Computer Graphics Proceedings* (1996), ACM, pp. 477–484.

[OBS04] OHTAKE Y., BELYAEV A., SEIDEL H.-P. : Ridge-valley lines on meshes via implicit surface fitting. In *SIGGRAPH '04 : ACM SIGGRAPH 2004 Papers* (New York, NY, USA, 2004), ACM, pp. 609–612.

[Ola05] OLANO M. : Modified noise for evaluation on graphics hardware. In *HWWS '05 : Proceedings of the ACM SIGGRAPH/EUROGRAPHICS conference on Graphics hardware* (New York, NY, USA, 2005), ACM, pp. 105–110.

[Per85] PERLIN K. : An image synthesizer. *SIGGRAPH '85, Computer Graphics Proceedings 19*, 3 (1985), 287–296.

[PFS03] PASTOR O. M., FREUDENBERG B., STROTHOTTE T. : Real-time animated stippling. *IEEE Computer Graphics and Applications 23*, 4 (2003), 62–68.

[PHWF01] PRAUN E., HOPPE H., WEBB M., FINKELSTEIN A. : Real-time hatching. In *SIGGRAPH '01, Computer Graphics Proceedings* (2001), ACM, pp. 579–584.

[She64] SHEPARD R. N. : Circularity in judgments of relative pitch. *The Journal of the Acoustical Society of America 36*, 12 (1964), 2346–2353.

[SS02] STROTHOTTE T., SCHLECHTWEG S. : *Non-Photorealistic Computer Graphics : Modeling, Rendering and Animation*. Morgan Kaufmann, 2002.

[VBTS07] VANDERHAEGHE D., BARLA P., THOLLOT J., SILLION F. : Dynamic point distribution for stroke-based rendering. In *EGSR '07, Proceedings of the Eurographics Symposium on Rendering* (2007), pp. 139–146.

[WY04] WU Q., YU Y. : Feature matching and deformation for texture synthesis. *SIGGRAPH '04, Computer Graphics Proceedings 23*, 3 (2004), 364–367.

Appendix A: Code source des *shaders* du zoom infini

Vertex Shader

```
uniform float obj_scale;
varying float dist;

void main(void)
{
        //distance to the camera
        vec4 posTransform =
             gl_ModelViewMatrix*gl_Vertex;
        dist = abs(posTransform.z);

        //volumetric texture coordinate
        gl_TexCoord[0] = gl_Vertex/obj_scale
             ;

        //projected position
        gl_Position = ftransform();
}
```

Fragment Shader

```
uniform sampler3D solidTex;
varying float dist;

vec4 main(void)
{
        //number of zoom cycles
        float z = log2(dist);
        float s = z-floor(z);

        //scale factor according to fragment distance to
             the camera
        float frag_scale = pow(2.0,floor(z))
             ;

        //octave weight according to the interpolation
             factor
        float alpha1 = s/2.0;
        float alpha2 = 1.0/2.0-s/6.0;
        float alpha3 = 1.0/3.0-s/6.0;
        float alpha4 = 1.0/6.0-s/6.0;

        //texture lookup
        vec4 oct1 = alpha1*texture3D(
             solidTex, gl_TexCoord[0].xyz/
             frag_scale);
        vec4 oct2 = alpha2*texture3D(
             solidTex, 2.0*gl_TexCoord[0].xyz
             /frag_scale);
        vec4 oct3 = alpha3*texture3D(
             solidTex, 4.0*gl_TexCoord[0].xyz
             /frag_scale);
        vec4 oct4 = alpha4*texture3D(
             solidTex, 8.0*gl_TexCoord[0].xyz
             /frag_scale);

        //blending
        vec4 n = oct1+oct2+oct3+oct4;

        gl_FragColor = n;
}
```