



Rendu de scènes 3D imitant le style “dessin animé”

Philippe Decaudin

► To cite this version:

Philippe Decaudin. Rendu de scènes 3D imitant le style “dessin animé”. [Rapport de recherche] RR-2919, INRIA. 1996. inria-00073778

HAL Id: inria-00073778

<https://hal.inria.fr/inria-00073778>

Submitted on 24 May 2006

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L’archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d’enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.



INSTITUT NATIONAL DE RECHERCHE EN INFORMATIQUE ET EN AUTOMATIQUE

***Rendu de scènes 3D imitant le style
«dessin animé»***

Philippe Decaudin

N° 2919

juin 1996

THÈME 3

Interaction homme-machine,
images,
bases de données, connaissances

 ***rapport
de recherche***

1996

Rendu de scènes 3D imitant le style «dessin animé»

Philippe Decaudin¹

Thème 3 : Interaction homme-machine, images, bases de données, connaissances

Projet Syntim

Rapport de recherche N° 2919

juin 1996

Résumé : Nous décrivons un algorithme de rendu qui génère des images imitant le style «dessin animé» traditionnel (ou «bande dessinée») à partir de la description tridimensionnelle d'une scène fixe ou animée. Pour ce faire, l'algorithme fait appel à des techniques qui permettent:

- de détourner les objets (profils et arêtes sont dessinés en traits noirs),
- de colorer uniformément les surfaces intérieures à ces contours,
- de faire apparaître sur les objets les ombres propres et les ombres portées dues aux sources de lumière éclairant la scène.

Mots-clé : synthèse d'image, rendu non-photoréaliste, dessin animé

1. Philippe.Decaudin@inria.fr
<http://www-rocq.inria.fr/syntim/recherche/decaudin>

Cartoon-Looking Rendering of 3D Scenes

Abstract: We present a rendering algorithm which produces images having the appearance of a traditional cartoon from a 3D description of the scene (a static or an animated scene). The 3D scene is rendered with techniques allowing to:

- outline the profiles and edges of objects in back,
- color uniformly the patches,
- render shadows (self-shadows and projected-shadows) due to light sources.

Key-words: computer graphics, non-photorealistic rendering, cartoon.

1 Introduction

La technique présentée dans cet article a pour but de produire des images de synthèse ayant un style «dessin animé» à partir de scènes tridimensionnelles.

Contrairement aux images de synthèse traditionnelles où l'objectif est d'obtenir un rendu aussi proche que possible de la réalité (rendu photoréaliste), les images de dessins animés utilisent un rendu plus épuré et plus suggestif [1]. Ainsi, les personnages et les objets présents sur l'image sont caractérisés par leur contour, l'intérieur du contour étant généralement rempli par une couleur unie (voir figure 1).

Aspects caractéristiques

Il existe différents niveaux de qualité de dessin. L'épaisseur des contours peut varier, les aplats de couleur peuvent inclure des dégradés, les effets de lumière et d'ombre peuvent créer une ambiance... Nous ne cherchons pas ici à reproduire tous les effets possibles. Nous souhaitons obtenir des images ayant clairement un style «dessin animé» ou «bande dessinée», pour cela, nous nous attacherons à produire des contours d'épaisseur constante et des aplats de couleur unis. Nous ajoutons à cela trois effets supplémentaires: la possibilité de remplacer la couleur unie des aplats par une texture, l'ajout de tâches spéculaires sur certains objets, et les ombres.

- La texture n'est pas indispensable, mais elle permet d'enrichir facilement un dessin (pour obtenir, par exemple, un motif répétitif sur une surface).
- Les tâches spéculaires permettent de caractériser la matière d'un objet (brillant, métallique,...). Elles correspondent aux reflets des sources de lumière sur l'objet et sont représentées par des tâches très localisées de couleur claire (voisine du blanc).
- Quant aux ombres, elles ajoutent considérablement à la compréhension de la scène et à la perception de la 3D. Il y a deux types d'ombres: les ombres propres (ou auto-ombres) correspondant aux côtés non éclairés des objets et les ombres portées correspondant aux zones occultées par un objet intercalé entre la zone et la source de lumière. Nous allons traiter les deux. Dans les dessins animés classiques «bons marchés» les ombres ne sont généralement pas représentées. Dans les dessins animés plus élaborés, les ombres propres sont figurées; une zone d'un objet à moitié dans la lumière et à moitié dans l'ombre aura alors deux couleurs : la couleur de l'objet pour la partie éclairée et cette même couleur assombrie pour la partie non éclairée. La frontière entre les deux est franche, contrairement aux images de synthèse traditionnelles où la transition est douce (en dégradé). Dans le cas d'une source de lumière unique et immobile, ces ombres propres sont relativement simples à dessiner à la main même si l'objet est en mouvement car elles restent relativement localisées au court d'une animation, surtout si la lumière est directionnelle (source à l'infini). Par contre, les ombres portées d'objets en mouvement peuvent évoluer beaucoup lors d'une animation, cela explique qu'on ne représente généralement que les ombres portées d'objets fixes, comme le décor.

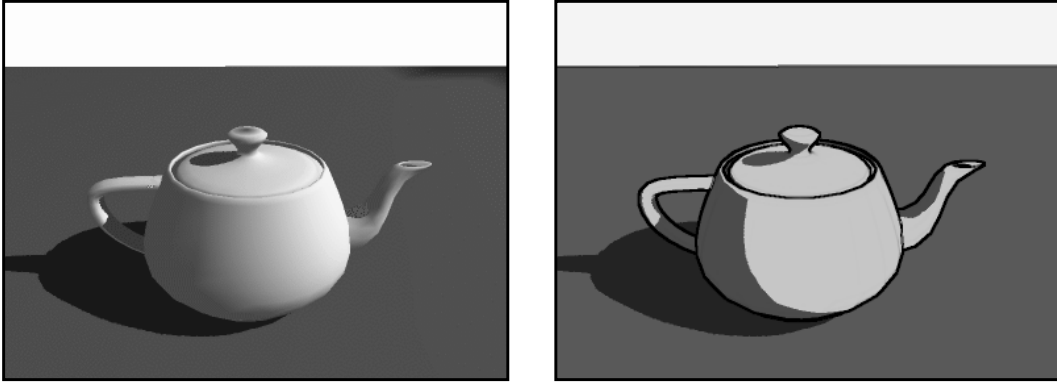


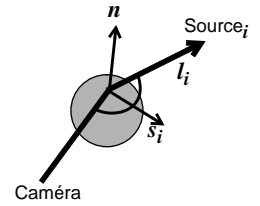
Figure 1 : théière rendue par un algorithme de synthèse classique (à gauche) et par notre algorithme «dessin animé» (à droite).

Principe

Notre algorithme produit une image à partir d'une description géométrique de la scène 3D. Tous les types de représentation peuvent convenir; de ce point de vue, l'algorithme est relativement général. Dans notre implémentation, nous utilisons, en entrée de l'algorithme, des scènes représentées sous forme d'arbre OpenInventor [7]. Cette description nous permet d'utiliser des techniques de rendu projectif classique comme base pour notre algorithme. Ces rendus sont effectués via les bibliothèques OpenInventor et OpenGL [3], ce qui permet de profiter des performances de la machine et de sa carte graphique pour accélérer le calcul. L'extension à d'autres techniques de rendu ne doit pas poser de problème.

Le rendu OpenGL utilise un modèle d'illumination dérivé du modèle de Phong. La couleur d'un point est calculée ainsi :

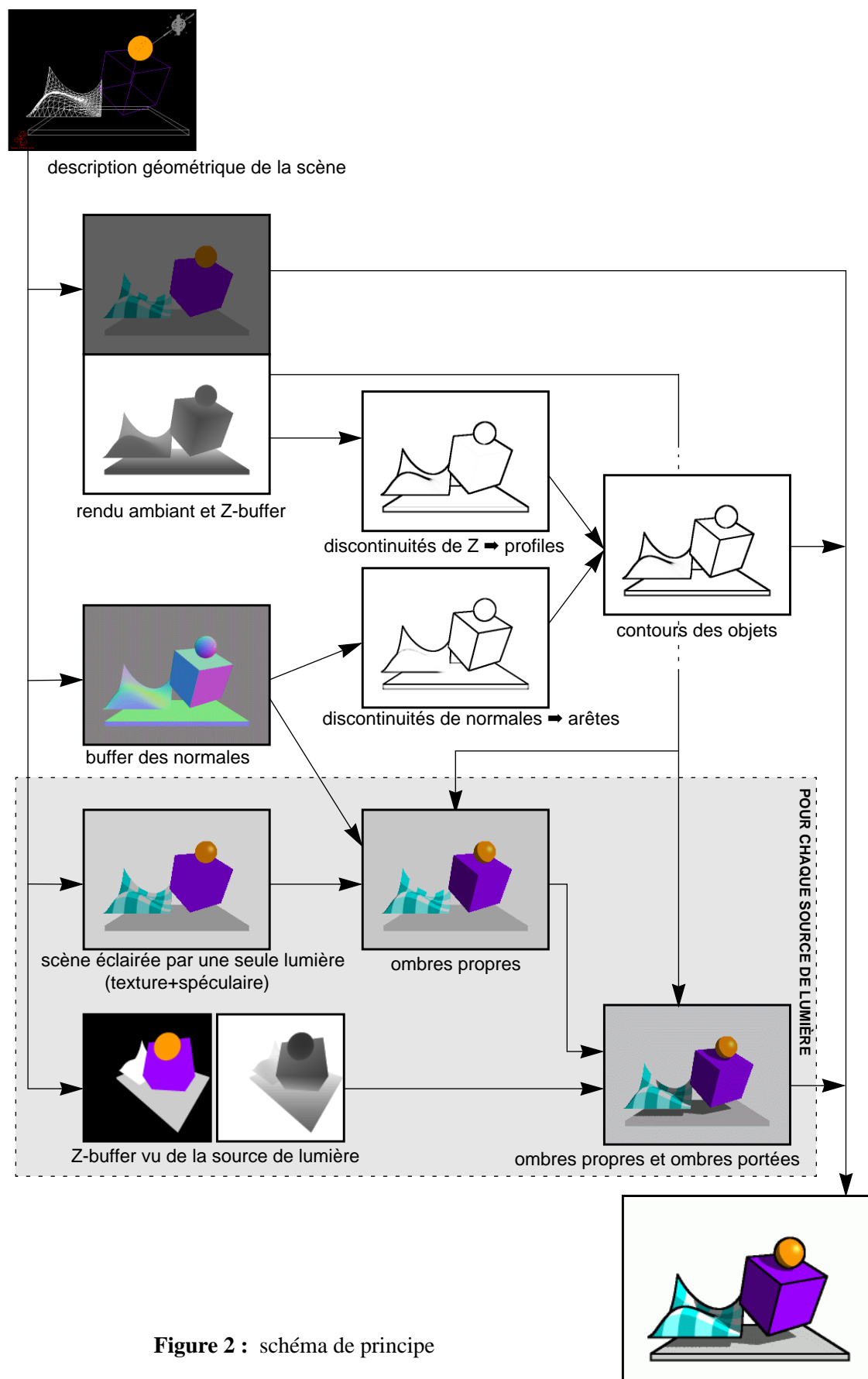
$$\begin{aligned}
 \text{couleur} = & \text{ambient}_{\text{global}} \times \text{ambient}_{\text{matiere}} \\
 & + \sum_{i=0}^{\text{nb sources}} \text{ambient}_{\text{source}_i} \times \text{ambient}_{\text{matiere}} \\
 & + \max\{l_i \cdot n, 0\} \times \text{diffus}_{\text{source}_i} \times \text{diffus}_{\text{matiere}} \\
 & + \max\{s_i \cdot n, 0\} \times \text{speculaire}_{\text{source}_i} \times \text{speculaire}_{\text{matiere}}
 \end{aligned}$$



Dans la suite, nous allons appliquer ce modèle de façon à obtenir les différents effets désirés.

Winkenbach et Salesin [9] ainsi que Salisbury et al. [6] se sont intéressés aux illustrations à l'encre et à la plume. L'effet obtenu est assez proche du style bande dessinée, mais la méthode utilisée est plus limitative. Dans [9], seules les objets composés de surfaces planes peuvent être traités, et dans [6] le processus est interactif, donc difficilement utilisable pour générer une animation.

Notre algorithme utilise un ensemble de techniques plus ou moins classiques en image de synthèse. Ainsi, le calcul des ombres utilise la technique des «shadow maps» [8] et le tracé des contours s'inspire de techniques de traitement d'images décrites par Saito et Takahashi [5]. L'originalité de notre travail vient principalement de la façon dont ces techniques sont adaptées et mixées ensemble.



2 L'algorithme

2.1 Vue d'ensemble

La figure 2 présente les différentes étapes de notre algorithme de rendu. Chaque étape produit une image, la combinaison de ces images donne l'image finale. Ces étapes que nous détaillerons par la suite sont:

- rendu de la scène en lumière ambiante;
- calcul des contours des objets de la scène;
- pour chaque source de lumière:
 - rendu de la scène éclairée par cette seule lumière;
 - calcul des ombres propres et des ombres portées dues à cette lumière;
- combinaison des images.

2.2 Les différentes images calculées

Les images et buffers nécessaires pour chaque étape de calcul sont les suivant:

- Rendu de la scène en lumière ambiante. Cette image est produite en calculant un rendu de la scène vue de la caméra en ayant préalablement éteint les sources de lumière. Chaque objet a alors une couleur ou une texture sans effet d'illumination (les couleurs sont unies), atténuée par l'intensité ambiante de la scène. Par exemple, si l'intensité ambiante vaut 0.2, un objet rouge ($R=1, V=0, B=0$) apparaîtra uniformément rouge foncé ($R=0.2, V=0, B=0$).
- Z-buffer de la scène vue de la caméra. On profite du calcul de l'image précédente (rendu ambiant) pour récupérer son Z-buffer (i.e. carte de profondeur, en chaque point est stocké la distance du point 3D à la caméra).
- Buffer des normales associées à la scène. En chaque point de ce buffer sont stockées trois valeurs réelles: les coordonnées de la normale associée au point 3D correspondant. En fait, il est possible d'obtenir ce buffer en calculant deux rendus de la scène vue de la caméra (figure 6). Pour ces deux rendus, les matières et textures des objets sont remplacés par une matière parfaitement diffuse et uniformément blanche (ambiant=noir, diffus=blanc ($R=1, V=1, B=1$), pas de spéculaire). Les sources de lumière sont éteintes. Le premier rendu (appelons le I_1) est obtenu en ajoutant trois sources de lumière directionnelles à la scène respectivement de couleur rouge et de direction $+x$, de couleur verte et de direction $+y$ et de couleur bleue et de direction $+z$. Le calcul d'illumination en un point où la normale vaut $\mathbf{n}=(n_x, n_y, n_z)$ donne:

$$\text{couleur} = \text{rouge} \times \max\{\mathbf{n} \cdot \mathbf{x}, 0\} + \text{vert} \times \max\{\mathbf{n} \cdot \mathbf{y}, 0\} + \text{bleu} \times \max\{\mathbf{n} \cdot \mathbf{z}, 0\}.$$

La couleur du point vaut donc $\max\{n_x, 0\}$ dans la composante rouge, $\max\{n_y, 0\}$ dans la composante verte et $\max\{n_z, 0\}$ dans la composante bleue. Le deuxième rendu (I_2) est obtenu en inversant la direction des trois sources directionnelles: la rouge suivant $-x$, la verte suivant $-y$, la bleue suivant $-z$. La couleur du point vaut alors

$$\text{couleur} = \text{rouge} \times \max\{-n_x, 0\} + \text{vert} \times \max\{-n_y, 0\} + \text{bleu} \times \max\{-n_z, 0\}.$$

Finalement, il suffit de calculer $I_1 - I_2$ pour obtenir un buffer dans lequel les composantes rouge, vert, bleu correspondent respectivement à n_x, n_y, n_z .

Pour chaque source de lumière :

- Rendu de la scène éclairée uniquement par cette lumière. Pour ce rendu, on ne tient pas compte de l'ambient global de la scène. On souhaite calculer une image sans véritable calcul d'illumination afin d'obtenir les aplats de couleur sans dégradé qui caractérisent le coloriage des dessins animés ou BD classiques. Pour cela, il faut éliminer le terme en $\mathbf{n} \cdot \mathbf{l}$ (\mathbf{n} est la normale à l'objet, \mathbf{l} est le vecteur dirigé vers la source) dans l'équation d'illumination de Phong. Si l'objet est spéculaire, on conservera tout de même la tâche spéculaire car elle renseigne sur la nature de la matière de l'objet (mat, brillant, métallique,...). On effectue donc un rendu classique en ayant au préalable modifié la matière des objets de la façon suivante: on affecte à la couleur ambiante de l'objet la valeur de sa couleur diffuse, on met la couleur diffuse à noir, la couleur spéculaire ne change pas. Ce rendu permet d'obtenir une image où chaque objet est uniformément coloré par sa couleur diffuse (modulo la texture et la tâche spéculaire).
- Z-buffer vu de la source de lumière. Ce buffer est obtenu en remplaçant la source de lumière par une caméra. Un rendu de la scène vue de cette caméra est calculé et son Z-buffer est, alors, récupéré. Ce buffer sera utilisé pour le calcul des ombres (détaillé plus loin) dues à cette source de lumière; la technique utilisée est celle dite des «shadow maps».

2.3 Les étapes de l'algorithme

L'image finale est obtenu par traitements et combinaisons des buffers et images détaillées dans le paragraphe précédent.

2.3.1. Les contours des objets

Deux types de contours nous intéressent: les profils (ou contours apparents) et les arêtes vives des objets.

Les profils correspondent aux lieux où le regard (demi-droite qui part du centre de la caméra vers la scène en passant par un pixel donné) est tangent à la surface des objets. Ce lieu peut être obtenu par des méthodes numériques, mais celles-ci sont généralement complexes à programmer car elles doivent gérer de nombreux cas particuliers et se révèlent souvent instables aux points où les profils s'interrompent à cause d'un changement de courbure de la surface (voir, par exemple, le patch bleu et blanc de la figure 2). Nous avons préféré utiliser une méthode plus simple et stable, mais un peu moins souple (inspirée de [5]). Les profils correspondent aussi aux discontinuités (d'ordre 0) présents dans le Z-buffer vu de la caméra. Il suffit donc d'appliquer un filtre «détecteur de contours» sur ce Z-buffer. Pour ce faire, nous utilisons l'opérateur différentiel d'ordre 1 de taille 3x3 suivant:

$$g = \frac{1}{8}(|A - x| + 2|B - x| + |C - x| + 2|D - x| + 2|E - x| + |F - x| + 2|G - x| + |H - x|)$$

où A,B,...,H sont les pixels voisins de x:

A	B	C
D	x	E
F	G	H

Cet opérateur permet d'obtenir le gradient de l'image en Z, il faut maintenant le seuiller pour obtenir les contours qui correspondent aux lieux de fort gradient. Le filtre non-linéaire 3x3 suivant est utilisé:

$$p = \min \left\{ \left(\frac{g_{max} - g_{min}}{k_p} \right)^2, 1 \right\}$$

où g_{max} et g_{min} sont les valeurs maximum et minimum du gradient dans le voisinage 3x3 considéré, et k_p est le seuil de détection compris entre 0 et 1. Plus k_p est petit, plus il y aura de contours détectés. Le choix de cette valeur est délicat; il s'agit d'un paramètre d'entrée de l'algorithme, il dépend principalement de la taille (largeur x hauteur) du Z-buffer car plus les variations de z sont précises, plus il est facile de dissocier une discontinuité d'une décroissance rapide mais continue de z . Dans notre implémentation, la valeur $k_p=0.0001$ donne des résultats satisfaisant pour une image calculée à la résolution vidéo (768 x 576).

Les arêtes vives visibles des objets peuvent être détectées de façon similaire. Théoriquement, un opérateur différentiel d'ordre 2 appliqué sur le Z-buffer devrait convenir (les arêtes sont des discontinuités de z d'ordre 1), mais, pratiquement, cet opérateur se révèle complètement instable car beaucoup trop approximatif. Un filtre de plus grande taille devrait donné de meilleurs résultats, mais les calculs sont alors plus coûteux. Nous avons opté pour une autre solution: ces arêtes correspondent également à des discontinuités d'ordre 0 du buffer des normales, il suffit donc d'appliquer à ce buffer un détecteur de contours, identique à celui proposé pour les profils, à ce buffer, à la différence près qu'il faut considérer les trois composantes n_x, n_y, n_z de ce buffer dans le calcul du gradient. Le choix du seuil k_a (équivalent à k_p mais pour les arêtes) est moins sensible que celui de k_p , une valeur $k_a=0.2$ donne de bons résultats.

Ces différents filtres nous permettent d'obtenir les contours caractérisant les objets de la scène de façon stable et en un temps très raisonnable. Il reste tout de même deux problèmes à résoudre: l'épaisseur des traits obtenus et leur aliasing. L'épaisseur des contours est imposée par les 2 passes de filtres 3x3, qui donnent au final un trait d'approximativement 5 pixels de large. Ce trait ne convient pas car il est alié, c'est la conséquence du seuillage assez abrupte que nous avons effectué: il évite d'avoir des traits flous, mais il génère de l'aliasing. Pour remédier à cet inconvénient, nous allons effectuer l'extraction des contours sur une image de résolution plus grande que la résolution finale. Le résultat sera rééchantillonné pour obtenir une image à la bonne taille. Ainsi, en effectuant le filtrage sur une image de taille double, les contours auront au final une épaisseur d'environ 2.5 pixels et seront de plus anti-aliasés puisqu'ils sont obtenus par moyennage des contours de 5 pixels de large. Bien entendu, plus on veut des contours fin et précis (anti-aliasé), plus les calculs sont lourds. Le calcul en taille double donne des résultats acceptables pour des images calculées à la résolution vidéo; les exemples présents dans cet article (figures 1 et 4) sont calculés ainsi.

2.3.2. Les ombres

Pour chaque source de lumière, un rendu de la scène éclairée par cette seule source est calculé. Nous allons faire apparaître les ombres dues à cette source sur cette image.

Les ombres portées sont obtenues par la technique dite des «shadow maps». Cette technique consiste à vérifier pour chaque pixel si le point 3D correspondant est visible ou

non de la source de lumière (figure 3). S'il ne l'est pas, ce point correspond à une zone d'ombre, le pixel est donc noirci. Les étapes sont donc les suivantes:

- pour chaque pixel (x_e, y_e) de l'image:
 - calculer les coordonnées (x_w, y_w, z_w) du point 3D correspondant à ce pixel (obtenu à partir de x_e, y_e , la valeur lue dans le Z-buffer z_e et la matrice 4x4 de projection de la caméra) dans le repère de la scène,
 - calculer les coordonnées (x_l, y_l, z_l) du point 3D dans le repère de la caméra associée à la source de lumière,
 - comparer z_l et la valeur z_s lue dans le Z-buffer associé à la source en (x_l, y_l) :
 - si $z_l > z_s$ alors le point n'est pas «vu» par la source, il est dans l'ombre
 - sinon, il est éclairé.

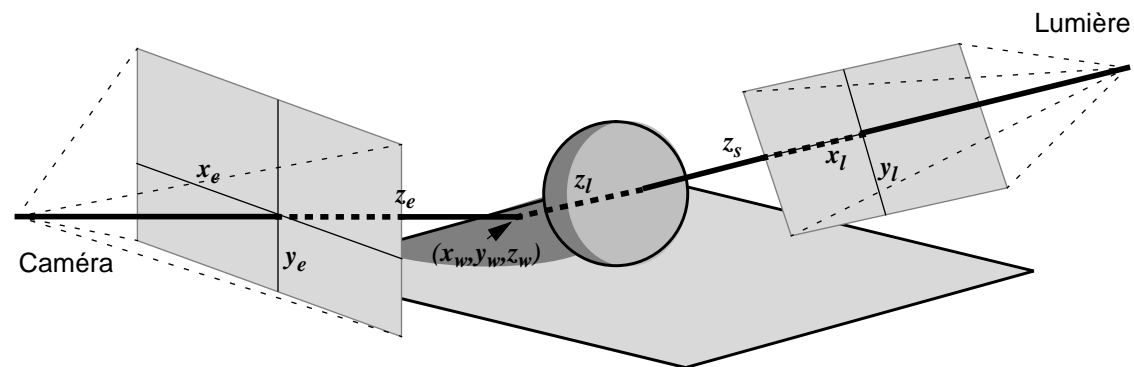


Figure 3 : notations utilisées pour le calcul des ombres portées

En pratique, cet algorithme a un défaut majeur : il génère des ombres aliasées, car le Z-buffer associé à la source possède une résolution arbitraire qui peut s'avérer insuffisante par endroit. Un objet occupant une grande place sur l'image vue de la caméra peut être petit sur l'image vue de la source, l'ombre portée par l'objet est alors imprécise (aliasée). Une astuce proposée par Reeves, Salesin et Cook [4] permet de contourner ce défaut, voire même de le transformer en qualité puisqu'il permet d'obtenir des ombres douces ("soft shadows"). Il s'agit d'effectuer la comparaison en plusieurs pixels proches de (x_l, y_l) et de moyenniser le résultats (se reporter à l'article pour plus de détails).

Cet algorithme devrait permettre de détecter aussi les ombres propres des objets, mais le résultat n'est pas satisfaisant car, pour notre rendu «style dessin animé», la limite des ombres propres doit être franche, de plus, le z_s lu dans le Z-buffer de la source n'est généralement pas assez précis au niveau de cette limite d'ombre. Heureusement, il existe une autre façon d'obtenir les ombres propres des objets. L'ombre propre correspond à la partie d'un objet opposé à la lumière, c'est-à-dire la partie où $\mathbf{n} \cdot \mathbf{l}$ est négatif (\mathbf{n} est la normale au point considéré, \mathbf{l} est le vecteur dirigé de ce point vers la source de lumière). Donc, le buffer des normales peut-être utilisé conjointement au Z-buffer de la caméra¹ pour effectuer ce test.

1. le Z-buffer permet de retrouver z_e , (x_w, y_w, z_w) est déduit de (x_e, y_e, z_e) et \mathbf{l} est obtenu en normalisant le vecteur (position de la source - (x_w, y_w, z_w))

On obtient ainsi une image où apparaissent les ombres propres et portées dues à la lumière considérée. Le cumul des images pour chaque source permettra de déduire l'image finale; cette étape est décrite dans la section suivante.

2.3.3. Calcul de l'image finale

L'image finale produite par notre algorithme est obtenue en additionnant le rendu ambiant et toutes les images ombrées éclairées indépendamment par chacune des sources, puis en multipliant le résultat par $\{1 - \text{buffer de contours}\}$, c'est-à-dire un buffer valant 1 partout sauf sur les contours où il avoisine¹ 0. L'ambiant global de la scène, les sources de lumière et leurs ombres sont donc pris en compte, et les contours apparaissent en noir dans l'image.

3 Implémentation et résultats

Nous avons implémenté cet algorithme en C++ sur station Silicon Graphics. L'algorithme prend en entrée une scène décrite au format OpenInventor et utilise les bibliothèques graphiques OpenInventor et OpenGL pour effectuer les différentes étapes de l'algorithme. Le principe consiste à utiliser ces bibliothèques pour modifier la scène (matières et caméra) et à lancer des rendus *en mémoire uniquement* (offscreen rendering) pour obtenir les différents buffers.

Nous avons utilisé cet algorithme pour créer un petit dessin animé (1mn12) portant le doux nom de «Rendez-vous». Il est généré directement à partir de descriptions 3D des scènes qui le composent et de leurs animations. Il n'y a aucun dessin à la main. Ces scènes ont été modélisées en utilisant quelques outils standards sur station Silicon Graphics tels que *SceneViewer* (figure 5) et *Noodle*, et récupérées sous forme de scripts OpenInventor. L'animation est obtenue en éditant ces scripts et en ajoutant quelques noeuds animés dans l'arbre Inventor de description de la scène. La figure 4 montre quelques images extraites de ce film. Le film est composé en tout plus de 1500 images différentes calculées en résolution vidéo (768 x 576). Le calcul d'une image a pris en moyenne 6 mn sur une station INDY (MIPS R4400 200MHz).

4 Conclusion et perspectives

Par rapport aux techniques du dessin animé traditionnelles, l'avantage de notre méthode est d'automatiser la production des images. En fait, on reporte le problème de l'animation en amont : il faut désormais créer une scène avec un modelleur 3D et l'animer. Ce système est certes moins souple que le dessin image par image, car il est difficile de produire des animations aussi délirantes que celles vues dans les cartoons avec les outils actuels d'animation en synthèse d'image. Mais, en contrepartie, outre le calcul automatique des images, il devient beaucoup plus facile de conserver la cohérence spatiale des objets dans la scène, et on peut, par exemple, générer facilement des mouvements de rotation d'objets ou de caméra ainsi que des déplacements de sources lumineuses. De plus, les ombres aussi restent cohérentes.

Il reste cependant quelques lacunes à la méthode. On aimerait, par exemple, pouvoir traiter des objets transparents, ou encore contrôler l'épaisseur des traits de contour indépendamment de l'anti-aliasing, et faire varier cette épaisseur en fonction de la courbure de la surface à cet endroit (technique fréquemment utilisée en bande dessinée pour appuyer la courbure).

1. les contours sont anti-aliasés

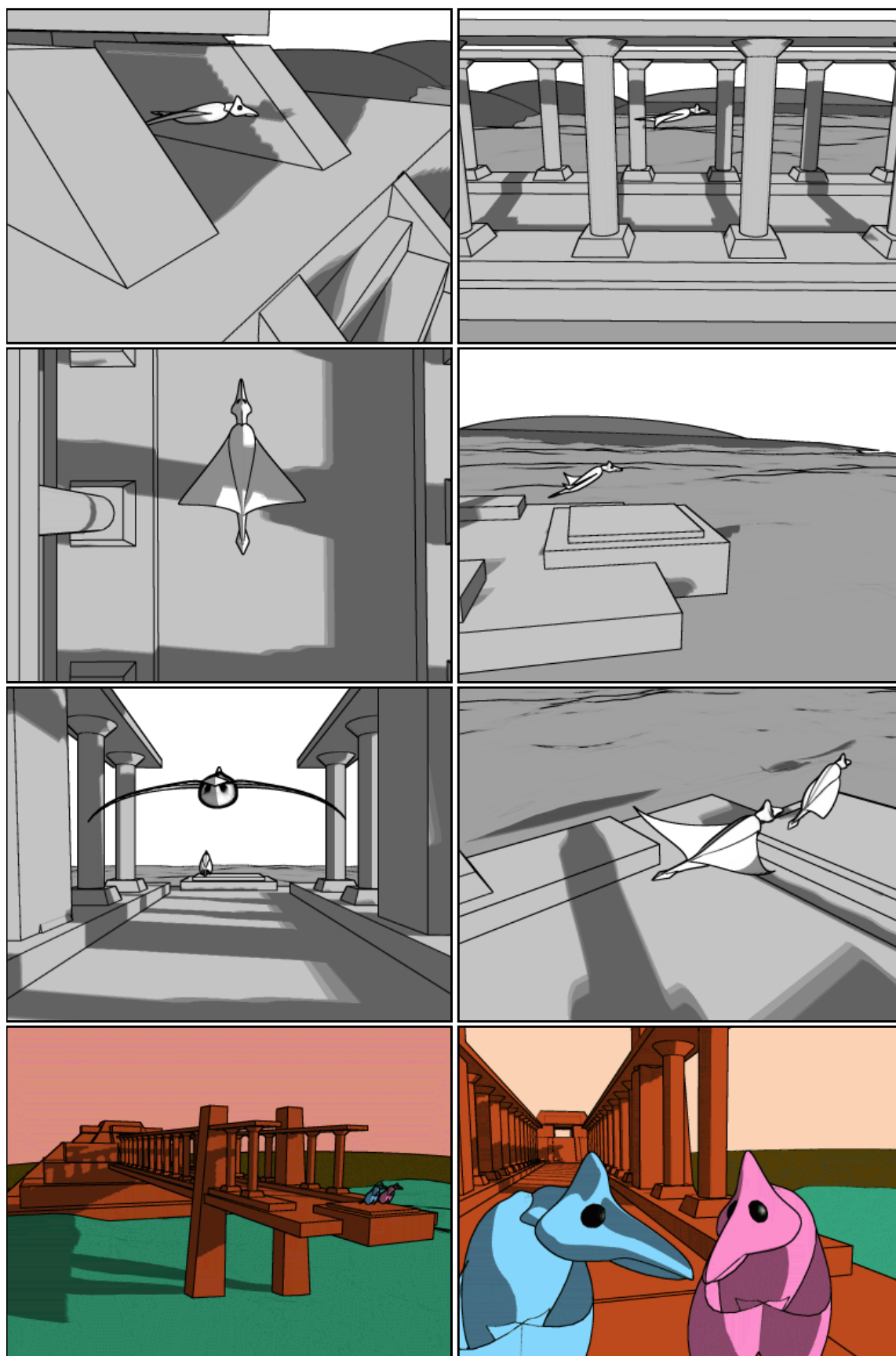


Figure 4 : quelques images extraites du dessin animé «Rendez-vous»

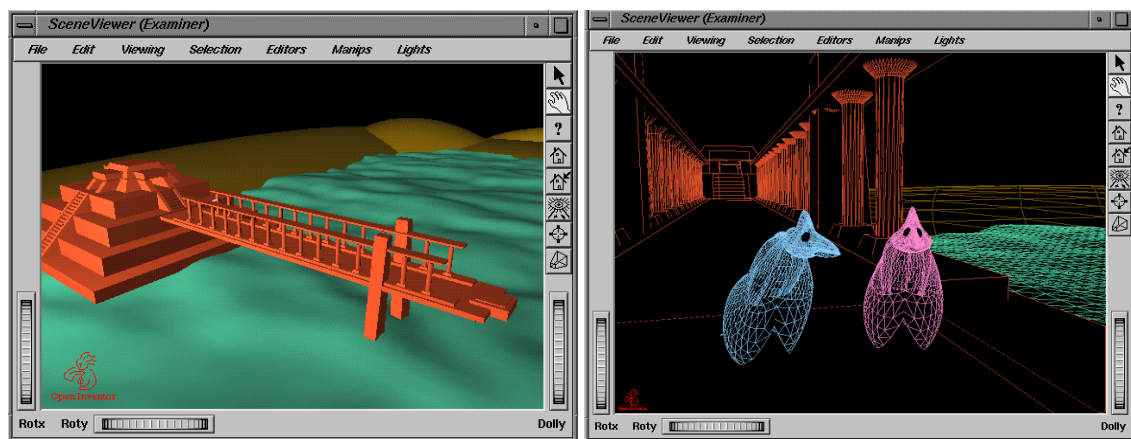


Figure 5 : scènes Inventor utilisées pour «Rendez-vous»

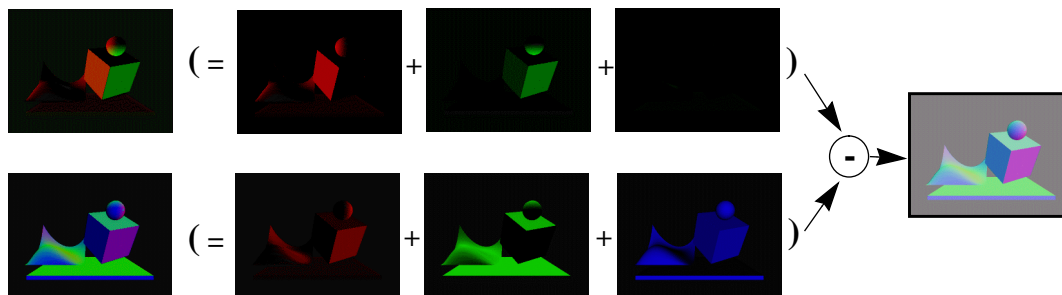


Figure 6 : construction du buffer des normales. L'image résultat (à gauche) comporte des valeurs R,V,B comprises dans $[-1, +1]^3$ correspondant à n_x, n_y, n_z (codage utilisé: -1,-1,-1=noir; 0,0,0=gris; 1,1,1=blanc).

Bibliographie

- [1] B. Duc, L'Art de la BD, Tome 2: La Technique du Dessin, *Editions Glénat*, 1983.
- [2] J. D. Foley, A. van Dam, S. K. Feiner and J. F. Hughes, *Computer Graphics: Principles and Practices (2nd edition)*, Addison Wesley, 1990.
- [3] J. Neider, T. Davis and M. Woo, OpenGL Programming Guide (Release 1), *Addison Wesley*, 1995.
- [4] W. T. Reeves, D. H. Salesin and R. L. Cook, Rendering Antialiased Shadows with Depth Maps, *Computer Graphics 21,4 (SIGGRAPH'87 proceedings)*, July 1987, pp. 283-291.
- [5] T. Saito and T. Takahashi, Comprehensible Rendering of 3-D Shapes, *Computer Graphics 24,4 (SIGGRAPH'90 proceedings)*, August 1990, pp. 197-206.
- [6] M. P. Salisbury, S. E. Anderson, R. Barzel and D. H. Salesin, Interactive Pen-and-Ink Illustration, *Computer Graphics, Annual Conference Series, (SIGGRAPH'94 proceedings)*, July 1994, pp. 101-108.
- [7] J. Wernecke, The Inventor Mentor - Programming Object-Oriented 3D Graphics with Open Inventor (Release 2), *Addison Wesley*, 1995.
- [8] L. Williams, Casting Curved Shadows on Curved Surface, *Computer Graphics 12,3*, August 1978, pp. 270-274.
- [9] G. Winkenbach and D. H. Salesin, Computer-Generated Pen-and-Ink Illustration, *Computer Graphics, Annual Conference Series, (SIGGRAPH'94 proceedings)*, July 1994, pp. 91-100.



Unité de recherche INRIA Lorraine, technopôle de Nancy-Brabois, 615 rue du jardin botanique, BP 101, 54600 VILLERS-LÈS-NANCY
Unité de recherche INRIA Rennes, IRISA, Campus universitaire de Beaulieu, 35042 RENNES Cedex
Unité de recherche INRIA Rhône-Alpes, 655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN
Unité de recherche INRIA Rocquencourt, domaine de Voluceau, Rocquencourt, BP 105, LE CHESNAY Cedex
Unité de recherche INRIA Sophia-Antipolis, 2004 route des Lucioles, BP655, avenue de l'Europe, 38330 MONTBONNOT ST MARTIN 93,
06902 SOPHIA-ANTIPOLIS Cedex

Éditeur

INRIA, Domaine de Voluceau, Rocquencourt, BP 105 LE CHESNAY Cedex (France)

ISSN 0249-6399