Abstraction and processing of large amounts of animated 3D data

Bert Buchholz

ParisTech
INSTITUT DES SCIENCES ET TECHNOLOGIES
PARIS INSTITUTE OF TECHNOLOGY

TELECOM ParisTech

2012-ENST-0080

EDITE - ED 130

Doctorat ParisTech

T H È S E

pour obtenir le grade de docteur délivré par

TELECOM ParisTech

Spécialité « SIGNAL et IMAGES »

présentée et soutenue publiquement par

Bert BUCHHOLZ

le 20 Décembre 2012

Abstraction et Traitement de Masses de Données 3D Animées

Directeur de thèse : **Tamy BOUBEKEUR**

Jury

M. Jean-Michel DISCHLER, Professeur, Université de Strasbourg	Rapporteur
Mme. Joëlle THOLLOT, Professeur, INRIA Rhône-Alpes	Rapporteur
M. Tamy BOUBEKEUR, Maître de Conférences (HDR), Télécom-ParisTech	Directeur de thèse
M. Pascal GAUTRON, Chercheur, Technicolor	Examinateur
M. Yann GOUSSEAU, Professeur, Télécom-ParisTech	Examinateur

TELECOM ParisTech
École de l'Institut Télécom – membre de ParisTech

THÈSE

SUMMARY

Rendering is the process of converting virtual 3D scenes into 2D images or animated sequences. Many techniques exist for this process to create a wide variety of different types of rendering, depending on the context and the aim. Rendering is used in computer games, film production, product visualization, medical imaging, computer-aided design and others. Two principal domains exist in rendering, the photorealistic (PR) and the non-photorealistic rendering (NPR). They are used depending on the area and purpose of application, often not solely but in some combination. The first one tries to achieve the creation of images that resemble reality as much as possible, while the second aims for the creation of stylized images that can emphasize certain features of the scene or imitate traditional styles of drawing and painting.

In this thesis, we explore intermediary structures and their relationship to the employed algorithms in the context of NPR and PR. In rendering, the input data is processed into data structures that simplify, abstract or reorder the data in such a way that specialized algorithms can be applied. In this thesis, new structures are explored for the application in rendering as well as new uses for existing structures.

We present three original contributions in the NPR and PR domain: First, we present a method to generate stylized black and white images with large regions, inspired by comic artists, using the appearance and geometry of the input data, called binary (i. e., two-color) shading. The core is an energy formulation that allows trading off multiple forces competing over the color classification. The energies are cast into a 2D image space graph representation and mimized using the Graph Cut method. We allow the user to control these energies to generate images of different styles and representations. The second contribution, like the first, resides in the NPR domain, working on animated lines instead of regions. Here, we present the temporally coherent parameterization of line animations for texturing purposes. We introduce a spatio-temporal structure over the input data and an energy formulation for the generation of a globally optimal parameterization. Similar to the work on binary shading, the energy formulation provides a continuous scale mainly between two different types of coherence giving the user an important and simple control over the output. Finally, we present an extension to Point-based Global Illumination (PBGI), a method used extensively in movie and film production during the last years. Our work allows compressing the data generated by the original algorithm using quantification without ever needing the complete data in memory with only a small timing overhead during the preprocessing. It is memory-efficient and enables the rendering of larger scenes without resorting to out-of-core methods. The user can easily control the strength and quality of the compression.

We also propose a number of possible extensions and improvements to the methods presented in the thesis.

RÉSUMÉ

Dans cette thèse, nous explorons des structures intermédiaires et le rapport entre eux et des algorithmes utilisés dans le contexte du rendu photoréaliste (RP) et non photoréaliste (RNP). Lors du processus de rendu, les données d'entrée, par exemple une scène virtuèlle 3D, sont transformées en structures spécialisées. Elles simplifient, abstraient ou réarrangent les données afin de faciliter l'accès par les algorithmes de rendu. Ici, nous explorons des nouvelles structures pour la mise en pratique dans le rendu ainsi que l'utilisation alternative des structures existantes.

Nous présentons trois contributions originales dans les domaines RP et RNP : Dans un premier temps, nous montrons une méthode pour la génération des images stylisées. Notre approche est inspirée par la démarche de dessinateurs de bandes dessinées, utilisant l'apparence et la géometrie des données d'entrée resultant en images caracterisées par des larges régions en noir et blanc, appelé "ombrage binaire". La partie principale est la formulation du problème comme énérgie balançant plusieurs forces influençant la décision sur la distribution des couleurs. Nous représentons l'énérgie dans une graphe en l'espace 2D et minimisons-la par la méthode Graph Cut. En contrôlant ces énérgies, l'utilisateur peut générer des images de differents styles et représentations. La deuxième contribution, comme la première, se trouve dans le domaine RNP, dans le contexte des lignes animées. Dans ce travail, nous proposons une nouvelle méthode pour la paramétrisation temporellement cohérente des lignées animées ayant pour but leur texturisation. Nous introduisons une structure spatiotemporelle sur les données d'entrée et une formulation d'énérgie permettant une paramétrisation globalement optimale. Ressamblant les travaux sur l'ombrage binaire, notre formulation du problème avec une énérgie donne à l'utilisateur le choix entre deux type de paramétrisation fournissant un contrôle important et simple sur le résultat. Finalement, nous présentons une extension sur une méthode de l'illumination globale basée sur la représentation par points (PBGI), ayant été utilisée largement dans la production de films au cours des dernières années. Notre extension effectue une compression par quantification de données générées par l'algorithme PBGI. Notre algorithme fonctionne sans avoir besoin les données entières en mémoire et ne prolonge pas le temps du calcul significativement. En même temps, le coût de memoire n'excède pas considérablement celui de la méthode d'origin et permet ainsi le rendu des scènes plus grande sans recours à l'utilisation de la mémoire de masse lente. Notre méthode permet à l'utilisateur un contrôle facile du facteur et de la qualité de compression.

Nous proposons également un nombre d'extensions ainsi que des augmentations potentielles pour les méthodes présentées dans cette thèse.

CONTENTS

RÉSUMÉ LONG

Introduction

Le rendu est la génération des images à partir des scènes 3D animées, se composant des objects (de la géométrie), matériaux, lumières et l'environnement. Pour les techniques différentes de rendu, il existe des structures de données très variées bien adaptées au problème posé. Dans cette thèse, nous présentons des nouvelles structures et l'application alternative de structures connues, en utilisant la géometrie at l'apparence de scènes 3d animées

Pour faire le rendu d'une scène, il faut un point de vue et, dans le cas d'une scène dynamique, c'est-à-dire une scène comportante des objets, lumières et/ou vues animés, un instant de temps. Une caméra virtuelle est placée au point de vue et une image est enrégistrée. Une image consiste en pixels (élements d'image) et le but est de trouver une couleur pour chaque pixel. Plusieurs méthodes existent à cette fin, mais en général, on commence par trouver la partie de cette surface qui est projetée sur un pixel. Les propriétés geometriques (direction de la surface, courbure etc.) et de matériaux (couleur, fonction de réflectance, FDRB) de la partie trouvée sont utilisées pour calculer la part de la lumière qui est reflété vers la camera. Cette partie est enrégistrée comme couleur du pixel.

Il existe plusieurs différentes approches pour un rendu, celles-ci ont des buts différents. On distingue essentiellement les techniques de rendu photoréaliste et non photoréaliste (voir Fig. 1 pour des exemples) mais souvent les deux sont mélangées pour créer des résultats désirés. Une technique n'utilise pas forcement l'entier des propriétés des données. Surtout les méthodes non photoréaliste n'utilisent qu'un sous-ensemble.

Etant donnée des masses de données 3D, possiblement animées, chaque technique doit avoir des moyens pour traiter ces données. Un défi majeur est de choisir la structure de données adaptée au problème, c'est-à-dire, une structure adaptée aux algorithmes utilisés pour résoudre le problème donné et économe de mémoire et de temps de calcul. Sans ces structures, on arrive à des temps de calcul prohibitevement longs. Dans cette thèse, on va présenter des approches de rendu de scènes 3D dans les domaines photoréaliste and non photoréaliste. Le focus est sur le développement des nouvelles structures pour la représentation de données, surtout de la géométrie, et l'application de structures connues dans des nouveaux contextes.

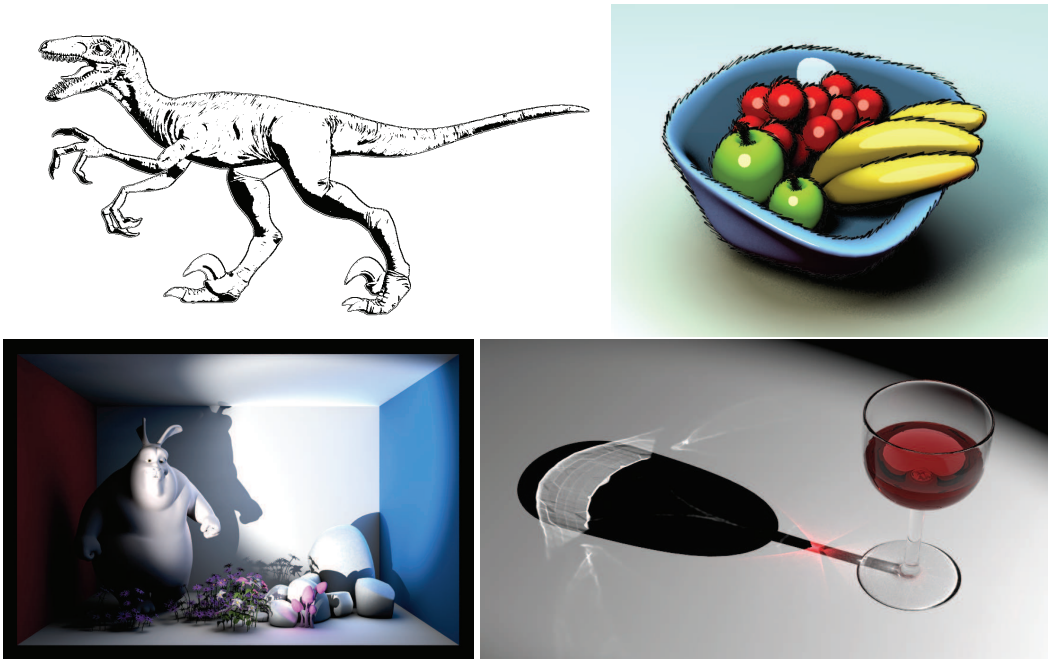


Figure 1 : Deux images en haut : Exemples de rendus non photoréalistes, de manière expressive, style noir et blanc (à gauche) et traits texturés (à droite). **Deux images en bas** : Rendus photoréalistes, l'illumination globale avec des effets basses fréquences (à gauche) et des caustiques (à droite).

Le rendu photoréaliste en non photoréaliste

Le rendu photoréaliste est la création des images photoréalistes (c'est-à-dire comme prise avec un appareil photo), ou de les approcher à une impression réaliste. Il y a plusieurs moyens pour y parvenir, on discerne deux méthodes principales : D'une part, les méthodes temps réel pouvant créer des dizaines d'images par seconde, utilisées dans les jeux vidéos et les applications en temps réel. De l'autre part celles prenant des secondes jusqu'à des heures pour produire une seule image, normalement utilisé dans la production de films et des images de haute qualité. La grande différence entre ces approches est la technique utilisée pour obtenir une image. Les méthodes plus rapides temps réel ou non, utilisent souvent de la géométrie simple et approximent les calculs tandis que les techniques lentes sont souvent basées sur un calcul physiquement correct. Le but est souvent de trouver une solution pour accélérer un type de rendu tout en conservant la qualité de l'apparence de la solution lente. Cela est habituellement fait en trouvant un moyen d'omettre des parties de la solution physiquement correcte ne faisant pas partie du résultat visible ou ayant une influence négligeable mais étant coûteuses en temps de calcul.

Les méthodes temps réel profitent souvent de la pouvoir des cartes graphiques modernes, permettant de calculer la géometrie de scènes même très larges. Cependant, le calcul de certaines effets non local comme l'ombrage ou la lumière indirecte reste très difficile à réaliser sur le processeur graphique.

Les méthodes physiquement correctes sont basé sur le raytracing [Whi80], dont l'idée principale est de lancer des rayons à partir de la caméra et suivre leurs interactions avec les

surfaces de la scène, resultant en le calcul de l'illumination directe. Si plusieurs interactions (des rebonds de la lumière sur les surfaces) sont prise en compte, on parle de l'illumination globale permettant de générer un fort réalisme visuel [Kaj86 ; Jen96 ; Vea97]. Ces méthodes sont d'habitude lentes, mais il existe des efforts de les accelerer par exemple en utilisant le processeur graphique [Par+10].

Entre les méthodes temps réel et physiquement correcte, il existe une classe de méthodes balançant le coût de calcul et qualité, surtout dans le domaine de l'illumination globale [War+88 ; Kri+05 ; Chr08]. Basé sur ces méthodes, récemment plusieurs approches ont été faites pour achever un temps de calcul au moins interactive, même temps réel [Rit+09 ; Hol+11 ; Sch+12].

Au début, le but principale du rendu était la création des images réaliste, utilisant des simulations physiquement correctes de l'interaction de la lumière avec des surfaces. Néanmoins, l'objectif d'une image est la présentation et transmission des informations aux spectateurs. Par exemple, si la propriété importante d'un objet est sa forme, le rendu de toutes les caractéristiques physiques peut encombre la perception des parties importantes.

Le rendu non photoréaliste (RNP, voir fig. 1, en haut) essaie surtout de mettre en avant des caractéristiques de l'objet jugées importantes pour les spectateurs humains, par le biais d'une stylisation artistique ou le rendu de figures techniques. Dans tous ces cas, l'auteur veux diriger l'attention du spectateur sur certains détails en les accentuant et en supprimant les informations qu'il estime ne pas être importantes. Les techniques RNP s'orientent beaucoup aux connaissances d'artistes. Soit que les techniques imitent des styles artistiques de dessins ou de la peinture, soit qu'ils utilisent la connaisance sur la perception humaine. Le dernier est un sujet bien étudié par des artistes et la psychologie et il existe également plusieures études spécifique au rendu : La question quelle façon de rendu transmet le mieux la forme d'un objet était etudiée par Winnemöller et al. [Win+07]. Pour les dessins au trait, Cole et al. ont etudiés les questions quelles lignes sont dessinées par un artiste pour une vue donné d'un objet et quelles lignes transmettent le mieux la forme d'un objet [Col+08 ; Col+09].

Il existe un nombre important de travaux s'occupant de la génération des images artistiques stylisées, y compris la simulation de techniques manuelles traditonelles comme la peinture [Goo+02 ; Van+07 ; Lu+10] ou celles plus modernes par exemple ressemblant au style de bandes dessinées [Bar+06 ; MG08 ; Eis+08].

L'autre côté est la génération des images pour un environnement professionel dans le contexte de la visualisation soit technique, par exemple la création automatique des illustration techniques[Goo+99], soit médicale[Rie+11]. Finalement, une autre application est la préparation de données dans un contexte dirigé par l'utilisateur, par exemple pour la présentation des endroits ou bâtiments importants sur un plan d'orientation interactif [GD09 ; Pin+12]

Abstraction et traitement des scènes pour le rendu

Pour organiser et préparer les données d'entrée pour l'algorithme, les méthodes de rendu utilisent des structures spécialisées. Ces structures ont pour but l'abstraction et simplification des données et leur représentation adaptée permettant l'execution efficace de l'algorithme de rendu. Il existe des types très variés de structures utilisés dans des rapports

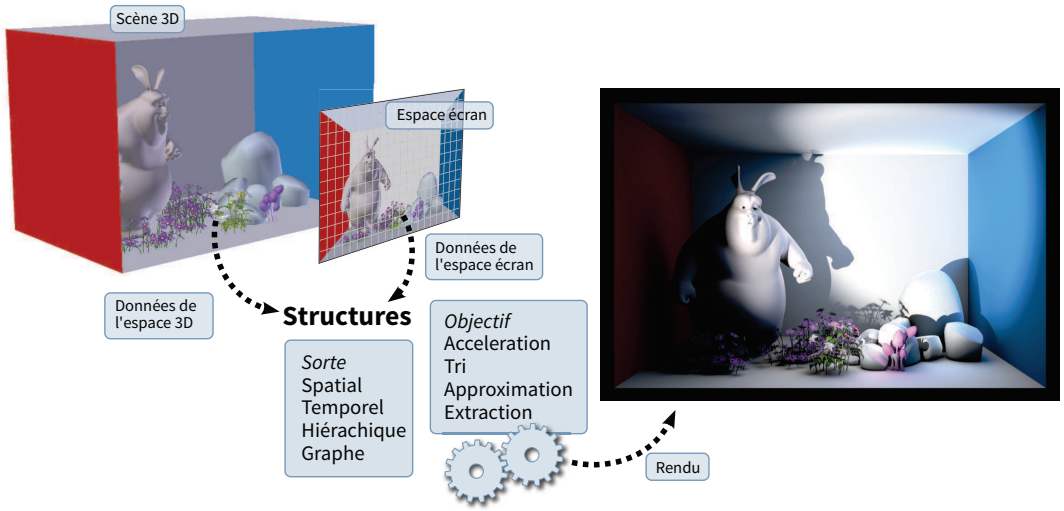


Figure 2 : Les données d'entrée (la scène 3D) sont transformées en structures adaptées à l'algorithme de rendu. Selon cette algorithme, les structures sont de differents types et objectives. L'algorithme se sert des données préparées dans les structures et crée l'image.

differents, par exemple les structures temporelles, spatiales, travaillantes sur des dimensions differentes ainsi que des combinaisons entre eux.

Dans le rendu, on trouve normalement des structures spatiales 3D et temporelles surtout pour l'organisation des masses de données 3D dont l'accès sans structure serait trop lent. Les structures spatiales 3D faisant une partition binaire de l'espace comme des arbres offrent non seulement de l'accélération mais également un accès hiérarchique aux données permettant le choix du niveau de détail. Ce dernier est souvent utilisé par des méthodes pour qui une approximation de données est suffisante, par exemple le rendu rapide de l'illumination globale ou le rendu des objets aux distances differentes avec un niveau de détail variable.

Dans le RNP, on trouve souvent des structures très proches et bien spécialisées à l'algorithme. Des graphes sont souvent utilisés pour établir le voisinage des regions de la scène, ou, dans le cas d'une méthode travaillant dans l'espace d'écran, celui des pixels.

Un problème important se posant pour les données animées est la cohérence temporelle du rendu. Pour la plupart de méthodes RP physiquement correctes, la cohérence temporelle est établi automatiquement par la nature de l'algorithme. Au cas de l'utilisation des approximations ou dans le domaine de RNP, il n'existe pas cette cohérence inhérente à l'algorithme. Pour les méthodes utilisantes un échantillonage aléatoire de la scène, on observe souvent un changement léger de couleurs pendant l'animation. Pour éviter cet artefact, il faut des structures temporelles connectant les positions des échantillons d'une façon qui minimise le changement des résultats de l'intégration pendant que la scène change. Pour le RNP, il faut un traitement dédié car ces artefacts sont souvent très specifique à l'algorithme.

Contributions

Dans cette thèse, nous présentons des structures soit nouvelles soit utilisées dans un nouveau contexte dans le domaine du rendu photoréaliste et non photoréaliste et nous examinons leur rapport avec les algorithmes utilisés. Lors du processus de rendu, les données d'entrée, la scène 3D, sont extraites de manière générale ou d'un point de vue speciale et transformées en structures spécialisées. L'algorithme de rendu travaille sur ces structures et calcule l'image.

Nous proposons trois nouvelles méthodes de rendu en RP et RNP ainsi que des extensions potentielles. Dans un premier temps, nous montrons une méthode pour la génération des images stylisées. Notre approche est inspirée par la démarche de dessinateurs de bandes dessinées, utilisant l'apparence et la géometrie des données d'entrée resultant en images caracterisées par des larges régions en noir et blanc, appelé "ombrage binaire". Nous fournissons une interface permettant à l'utilisateur un outil pour la création des images de differents styles et représentations. La deuxième contribution, comme la première, se trouve dans le domaine RNP, dans le contexte des lignes animées. Dans ce travail, nous proposons une nouvelle méthode pour la paramétrisation temporellement cohérente des lignées animées ayant pour but leur texturisation. Nous introduisons une structure spatiotemporelle sur les données d'entrée et une formulation d'énérgie permettant une paramétrisation globalement optimale. Notre formulation permet également un contrôle très simple du type de cohérence. Finalement, nous présentons une extension sur une méthode de l'illumination globale basée sur la représentation par points (PBGI), ayant été utilisée largement dans la production de films au cours des dernières années. Notre extension effectue une compression par quantification de données générées par l'algorithme PBGI. Pour cela, le coût ni de memoire ni de temps excède considérablement celui de la méthode d'origin et permet ainsi le rendu des scènes plus grande sans recours à l'utilisation de la mémoire de masse lente. Notre méthode permet à l'utilisateur un contrôle facile du facteur et de la qualité de compression. Nous proposons également un nombre d'extensions ainsi que des augmentations potentielles pour les méthodes présentées dans cette thèse.

Dans le suivant, nous présenterons des nouvelles structures ainsi que les algorithmes de rendu ayant pour but la création des images. Nous finirons ce résumé long par une conclusion globale et quelques perspectives et pensées liées aux travaux potentielles du futur.

Ombrage binaire

Soit pour des raisons artistiques, soit pour la simplification, il y a plusieurs techniques de l'abstraction d'images. Notre but est la réduction en deux couleurs mais différemment qu'un dessin noir/blanc avec des lignes. Dans ce dernier cas, les lignes se retrouvent normalement sur les endroits géométriquement importants. Notre but est de faire ressortir les surfaces différentes d'un objet ou une scène en colorant des grandes parties de la surface avec une couleur et en même temps gardant une bonne représentation de la géométrie. Ce style est utilisé surtout dans des bandes dessinées, mais peut être également utile dans l'affichage sur des écrans qui ne permettent que deux couleurs (par exemple papier électronique).

Nous proposons une technique qui fait la réduction à deux couleurs d'une scène 3d (voir Fig. 3). A partir d'un point de vue et un plan d'image, on rassemble des informations de la scène. L'apparence de la scène depuis un certain point de vue ainsi que plusieurs propriétés géométriques et de l'apparence de la scène sont regroupées dans un graphe connecté qui correspond au plan d'image. L'utilisateur peut pondérer ces propriétés de l'apparence et de la géométrie. Ensuite, la segmentation noir/blanc est trouvé en coupant le graphe en deux ensemble de la façon Graphcut [BJ01] (min-cut/max-flow).

Il y a quelques techniques qui génèrent des images binaires en dessinant des lignes [Jud+07 ; DeC+03]. Ces techniques partent normalement d'une scène 3d et utilisent la géométrie de l'objet et la position de la caméra pour déterminer quelles lignes à dessiner. Les autres techniques partent souvent d'une image 2d (par exemples des photographies) et les couleurs sont réduites en utilisant des algorithmes différents. Xu et al. [Xu+07] génèrent des images binaires dans le but de pouvoir les couper (silhouette), donc ils ont besoin de grandes régions colorées de façon cohérente qui sont toutes connectées. Pour la stylisation artistique, Xu et Kaplan [XK08] créent des images binaires en utilisant du seuillage sophistiqué. Mould et Grant [MG08] proposent une technique un peu plus coûteuse qui utilise entre autres le Graphcut, également utilisé dans notre méthode pour générer l'image binaire.

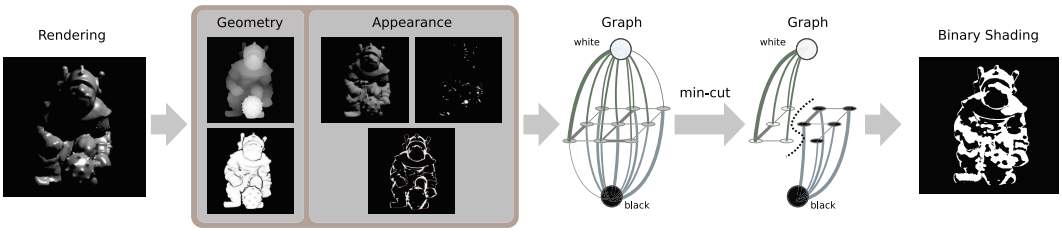


Figure 3 : Principe. Plusieurs propriétés géométriques et de l'apparance sont rassemblées d'un rendu de la scène pour un point de vue donnée. La structure de l'image est convertie en un graphe correspondante de la façon utilisée en Graphcut [BJ01]. Les propriétés sont utilisées pour définir les capacités d'arrêtes du graphe, la partie géométrique pour donner une cohérence aux regions géométriquement similaires et la partie de l'apparence pour un biais vers une de deux couleur. La segmentation, l'attribution d'une couleur à chaque sommet/pixel et fait par le « min-cut », segmentant le graphe en deux ensembles.

Principe

Partant d'une scène 3d, le but de notre approche est de créer des images composées de deux couleurs (habituellement blanc et noir) constituées de larges régions colorées de façon cohérente. L'utilisateur influence le résultat en décidant quelles sont les propriétés qu'il trouve important et en les pondérant.

La génération des images avec forte réduction du nombre de couleurs est motivée par des usages artistiques, pour faire de la stylisation, pour respecter les contraintes d'écran ou pour une meilleure représentation visuelle. Dans ce cas-ci c'est une réduction à deux couleurs qui donne donc des images binaires.

L'algorithme se décompose en deux étapes principales : La première, consiste à obtenir des informations sur la scène. La deuxième, est d'utiliser ces informations pour générer un graphe qui est segmenté en deux ensembles représentants les deux couleurs de l'image.

Implémentation/Résultats

Dans la première partie, les informations sont générées par raytracing de la scène à partir d'un point de vue et une résolution données par l'utilisateur. Le résultat est une image 2d qui contient un vecteur de deux genres principaux d'informations sur la scène. Pour chaque pixel contenant une intersection avec la géométrie de la scène, on conserve des informations sur l'apparence et la géométrie de la surface en ce point. Un pixel correspond bien sûr à une position dans l'espace de l'image 2d mais il correspond également à une position 3d sur la surface de l'objet.

Il y a plein de différentes propriétés de l'apparence et de la géométrie que l'on peut utiliser. Dans ce travail, on utilise pour l'apparence la lumière diffuse et spéculaire (qui dépendent d'au moins une source de lumière dans la scène) et la silhouette de l'objet (dérivée de l'angle entre le rayon de vue et la normale de la surface à chaque pixel).

L'information géométrique concerne la géométrie de la surface de l'objet. On utilise la courbure locale de la surface et la position de la surface à chaque pixel. La courbure locale est mesurée par la différence entre les normales dans une petite région autour d'un point de la surface. Cela permet une estimation des discontinuités sur la surface et donc de trouver les limitations naturelles des régions sur la surface, donc des caractéristiques (features) comme bumps, ridges et valleys. L'information sur la position est utilisée pour trouver les discontinuités de la distances entre deux pixels qui peuvent être voisins dans l'espace de l'image mais qui sont très éloignés dans l'espace 3d.

Ces informations sont rassemblées dans une grille d'une image 2d dans laquelle chaque pixel contient l'information sur l'apparence et la géométrie du point de la surface qui correspond à ce pixel. Cette image 2d est convertie en graphe dont chaque sommet correspond à un pixel. Même si la connectivité entre les sommets n'est pas fixe, le 8-voisinage est habituellement utilisé, donc chaque sommet est connecté à ses voisins directs horizontaux, verticaux et diagonaux. Chaque sommet est aussi connecté à deux sommets spéciaux (appelés terminaux, terminals en anglais), source et sink (en anglais). Le but est de couper le graphe en deux ensembles disparates où chaque sommet n'est connecté qu'à un seul de ces deux sommets spéciaux (soit directement, soit indirectement).

La coupe même est trouvée en résolvant le problème min-cut/max-flow pour lequel on cherche à trouver la coupe qui minimise la somme des coûts des arêtes coupées. Le coût

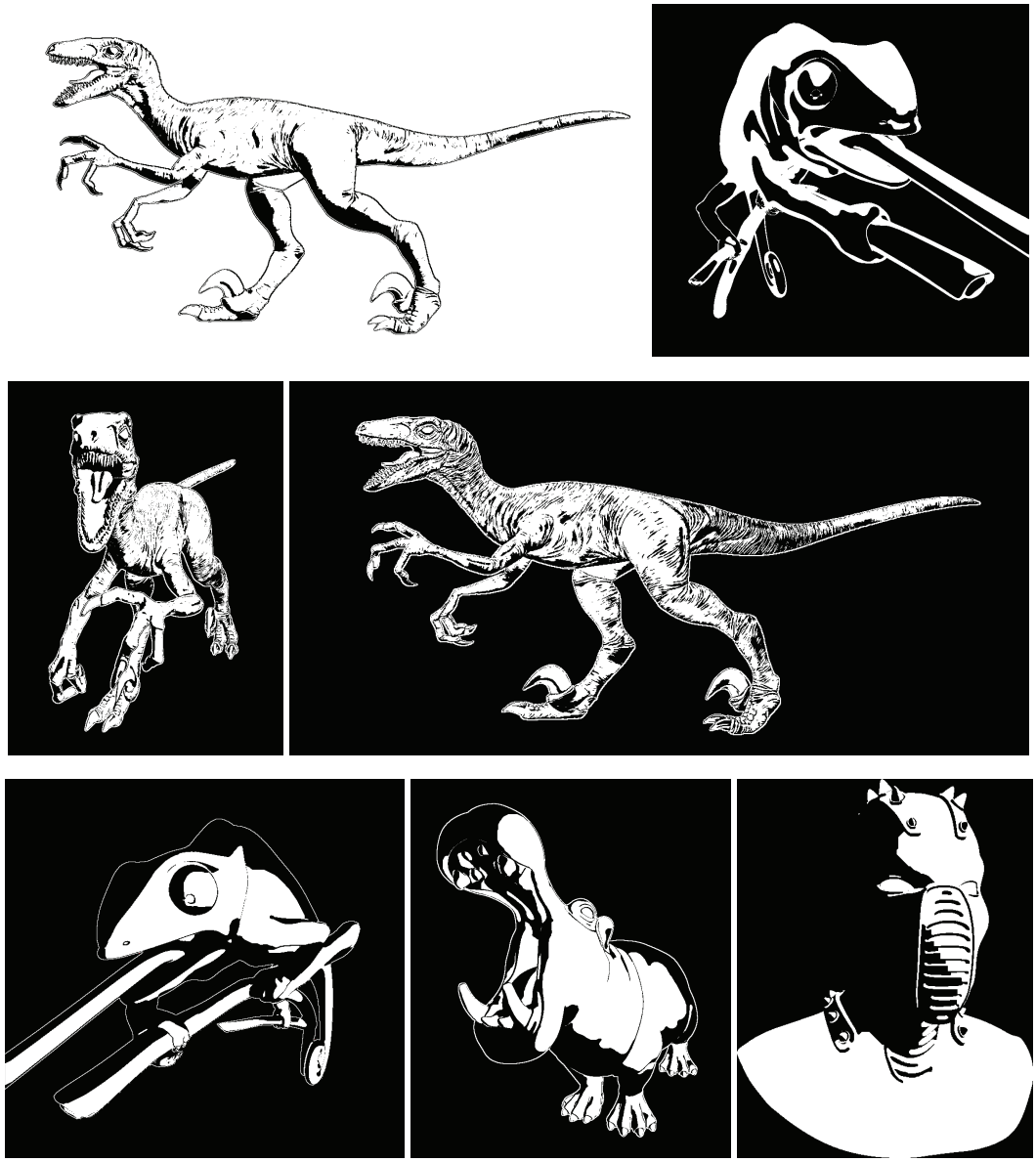


Figure 4 : Quelques résultats de « Ombrage Binaire » avec des styles différents.

de coupe d'une arête est sa capacité. Donc, en général, plus une capacité est élevée, plus la probabilité que l'arête ne soit pas coupée est élevée. Mais comme la solution est globale, la coupe finale dépend de toutes les arêtes. On utilise la méthode Graphcut [BJ01 ; BK04] qui est très répendue depuis quelques années dans le domaine de la segmentation d'image.

Les capacités entre les sommets normaux (capacité de voisins) sont dérivées à partir de l'information géométrique. Comme le but, mentionné ci-dessus, est de trouver des régions larges de blanc et noir, on essaie de remplir les régions de l'objet qui sont géométriquement cohérentes avec la même couleur. Pour y arriver, on utilise le fait que des différences basses de la courbure et des petites discontinuités de la profondeur sont présentes aux grandes

capacités des arêtes. Donc, des régions par exemple plates (et donc cohérentes) ont une forte probabilité de rester connectées.

Les capacités entre chaque sommet et les deux terminaux (capacités terminales) sont déterminées à partir de l'information d'apparence. Une valeur élevée de l'apparence va résulter dans une capacité élevée vers la source et basse vers le sink et inversement.

Les capacités terminales peuvent être considérées comme une tendance à une de les deux terminales (et donc à une couleur) pour chaque pixel individuel. Les capacités de voisins servent à la préservation de la cohérence des régions.

Après l'exécution d'une coupe respectant ces contraintes (trouver une coupe séparant les deux sommets terminaux), les sommets du graphe sont reconvertis en une image 2d où à chaque sommet connecté au sommet terminal sink est attribué une couleur (normalement blanc) et à chaque sommet connecté au sommet terminal source est attribué l'autre couleur (normalement noir), donnant ainsi l'image finale de deux couleurs.

L'utilisateur peut influencer les résultats finaux en changeant des poids associés à chaque type d'information de l'apparence et géométrie. En agissant sur ces poids, l'utilisateur peut influencer directement et indirectement les capacités dans le graphe qui déterminent le résultat final.

Analyse

Le plus grand problème avec cette approche est la contrôlabilité du résultat. Sans expérience avec le système, l'utilisateur a du mal à diriger le résultat vers la direction voulue. Et dû au Graphcut, un petit changement de poids peut effectuer un grande changement de la segmentation.

Utilisant raytracing pour obtenir des informations sur la scène provoque des longs temps d'attente (quelques seconds ou plus) pour voir le nouveau résultat, même aggravant le problème. Cette problème peut être résolu en utilisant des méthodes temps réel mais sous les contraintes du processeur graphique.

Dans le cas d'une animation, le système peut générer les résultats pour des trames seuls d'animation, mais on observe souvent des sautes de la segmentation entre deux trames. Pour obtenir des résultats temporellement cohérents, une solution possible serait de connecter les graphes de chaque trame, donnant un graphe volumique. Pendant le développement, on a essayé plusieurs arrangements de connexion entre les sommets des trames. Entre autres la connexion prenant en compte le mouvement des sommets en 3d et 2d mais nous n'arrivions pas à une solution satisfaisante.

De l'ombrage binaire à l'ombrage multi-couleurs

Il existe plusieurs pistes de travaux futurs. L'utilisation des propriétés supplementaires (par exemple reflection) aiderait également à l'augmentation du nombre de style. L'interaction directe peut être améliorée en ajoutant des differents modèles d'apprentissage basé sur des collections de travaux des artistes, possiblement en utilisant ces propriétés supplémentaires. Finalement, pour augmenter le nombre de style possible, l'extension de nombre de couleurs est une piste très prometteuse (voir Fig. 5).

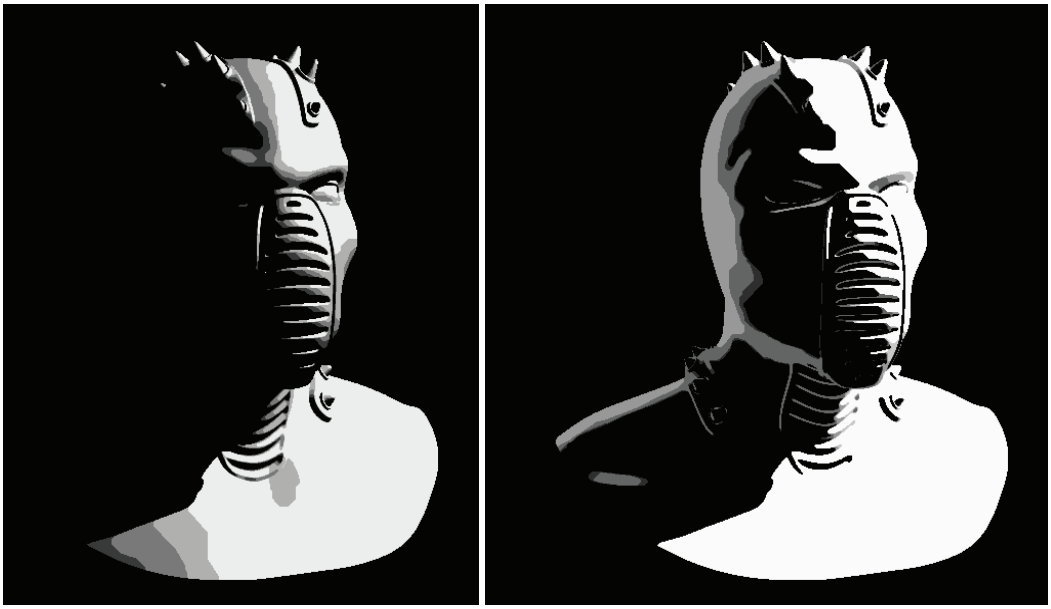


Figure 5 : k-couleurs. L'extension à plusieus couleurs pourrait ameliorer la perception de la surface ainsi que augmenter le nombre de styles. Cet exemple montre l'idée en superposant des rendus avec des paramètres differents. **À gauche** : Poids constants diffus et des poids differents spéculairs de la source. **À droite** : Poids constants diffus et des poids differents de la silhouette.

Dans ce travail présenté on a utilisé les propriétés d'une scène 3d que l'on convertit dans une structure d'un graphe 2d déterminé par l'utilisateur en leur importance pour générer des images binaires stylisées. Ce travail est paru dans [Buc+10]. Dans la section prochaine, nous regardons un problème dans le domaine de la stylisation de lignes animées. Comme la représentation en régions en noir et blanc, celle en lignes est une représentation très réduite. Dans ce cas, la définition des régions n'est pas implicite mais explicite par les lignes. L'une peut être considéré comme le dual de l'autre, permettant la générations des styles très differents.

L'analyse spatiale et temporelle pour la paramétrisation des lignes animées

La travail précédent montrait la définition d'un objet avec des régions blanches et noires. Dans ce travail, nous regardons également le dual ce type de représentation, le dessin au trait. Dans ce cas, les traits forment les bords entre le régions et marquent les caractéristiques de l'objet. Le dessin au trait est la manière la plus facile de transmettre la forme d'un objet, souvent utilisé par des artistes pour la stylisation par exemple de films animés ou le rendu de scènes 3D de façon dessinée. Mais même si la forme est bien représentée, des lignes ne permettent pas une bonne compréhension d'autres propriétés, par exemple celles de la surface d'un objet. Pour cela, gardant la simplicité des lignes, une possibilité est donnée par la texturisation des lignes, c'est-à-dire mettre des textures non uniformes sur les lignes, ressemblant par exemple des coups de pinceau ou permettant le changement de l'épaisseur de ligne.

Cela demande une paramétrisation de chaque ligne extraite. Une méthode triviale serait de prendre chaque trame de l'animation et paramétrer toutes les lignes indépendamment, ce qui donnerait une paramétrisation temporellement très incohérente. Ce travail-ci décrit un moyen de trouver une paramétrisation temporellement cohérent et en même temps de permettre à l'utilisateur d'influencer la paramétrisation entre les extrêmes de la cohérence temporelle d'une part et de la cohérence spatiale de l'autre part. Le premier pro-

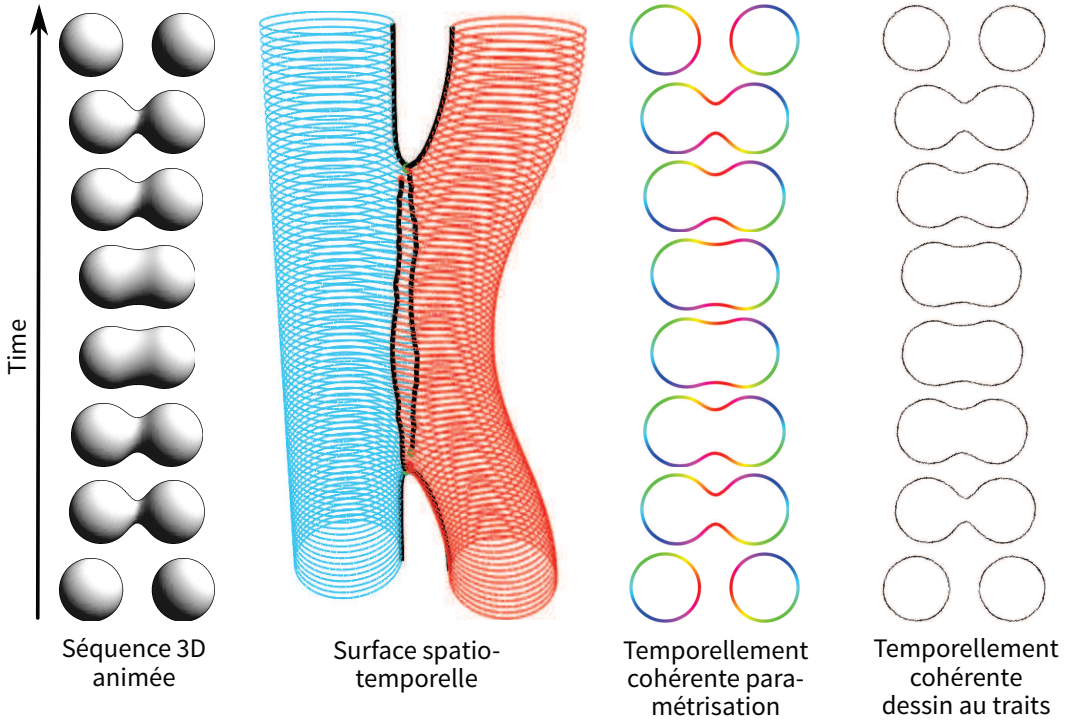


Figure 6 : Les étapes différentes de la paramétrisation cohérente des lignes.

blème à résoudre, c'est l'extraction des lignes de tous les trames de l'animation et trouver les correspondances de lignes entre tous les trames. Une fois que les correspondances sont établies, on paramètre indépendamment les groupes de lignes correspondantes. Avec la paramétrisation des toutes les lignes de toutes les trames on peut mettre des textures sur les lignes de façon cohérente.

De nombreux travaux existent sur le sujet de l'extraction de lignes à partir d'un objet 3D [Jud+07 ; DeC+03], mais normalement le but n'est pas la paramétrisation. Quelques travaux s'intéressent également à la cohérence temporelle des lignes mais pas à leur paramétrisation [DeC+04]. La méthode, présentée ici, suit les travaux de Kalnins et al. [Kal+03] qui a le même objectif mais propose une technique temps réel. Celle-ci ne peut donc pas prendre en compte la partie de l'animation future et ainsi ne peut pas trouver une solution globalement optimale mais juste une solution d'une trame à l'autre. Il existe en outre des travaux s'occupants principalement de la stylisation temporellement cohérente des lignes sans fournir une paramétrisation cohérente globale [Bén+10 ; Bén+12], ce qui est le but principal de la méthode présentée.

Principe

La paramétrisation de lignes de manière temporellement cohérente est intéressante pour des nombreux domaines mais particulièrement pour les dessins animés. Dans les dessins animés générés par ordinateur, les lignes sont normalement les contours des objets de la scène. Souvent, ces lignes ne sont pas seulement dessinées par un trait de la même couleur et la même épaisseur, mais par des traits variants (en utilisant des textures) pour ressembler à des dessins d'un artiste humain. Pour cela, une paramétrisation des lignes est nécessaire. Dans ce cas, il faut trouver un moyen de forcer une certaine cohérence temporelle, car si chaque ligne est paramétrée indépendamment pour chaque image d'une animation, les textures sur les lignes semblent glisser sur la surface de l'objet. Avec une paramétrisation temporellement cohérente, la texture semble être attachée à la surface.

On commence depuis une séquence de points de vue et un objet/une scène animé, donné par l'utilisateur. Après l'extraction des lignes et le groupement, on trouve une valeur de paramétrisation pour chaque sommet en prenant les contraintes comme une énergie à minimiser. Chaque contrainte peut être pondérée par l'utilisateur mais le choix principale est la balance entre la cohérence temporelle et spatiale. Cela donne à l'utilisateur la possibilité d'influencer le résultat surtout entre ces deux opposés. Après, on minimise l'énergie au sens des moindres carrés et obtient une valeur de paramétrisation pour chaque sommet.

Implémentation/Résultats

Notre algorithme est composé de deux parties. Étant donné est un ensemble des lignes que l'on a obtenu avant. Par exemple, dans le cas d'une animation 3D, les lignes peuvent être les contours de l'objet 3D. Les lignes qui sont paramétrées à la fin sont les projections des ces contours 3D. Chaque trame de l'animation contient quelques lignes qui ont des correspondances dans les trames précédentes et suivantes. On appelle l'ensemble des lignes la surface spatiotemporelle.

Dans une première phase de l'algorithme, il faut trouver ces correspondances temporelles entre les lignes. Un problème à résoudre sont des événements de la visibi-

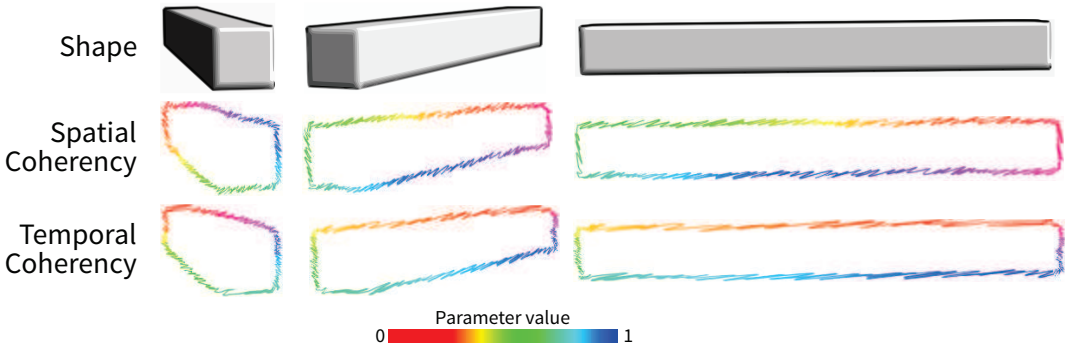


Figure 7 : De la cohérence spatiale à la cohérence temporelle.

lité/occlusion, quand deux lignes se joignent ou se séparent. On utilise deux méthodes pour trouver ces situations. L'une est une façon de vote dans lequel chaque ligne est comparée avec les lignes dans la trame précédente et suivante avec la distance Hausdorff. Le vote est utilisé pour faire des relations 1-à-x ou x-à-1 entre les lignes, c'est à dire, on essaie de trouver quelles lignes se sont rejointes/séparées avec/de quelles autres. Le résultat de cette opération est un ensemble de points rejoints/séparés où chaque événement produit quatre de ces points, deux sur la ligne rejointe et respectivement un sur les deux lignes séparées.

La deuxième méthode trouve ces événements directement. Un événement ne émerge que dans le cas où le point de vue se retrouve dans le plan d'un sommet (le plan avec la normale du sommet et sa position). On détecte cela on regardant la position relative de la caméra en relation au plan de chaque sommet, c'est à dire, si la caméra est devant ou derrière le plan.

Avec ces points rejoints/séparés sur la surface spatiotemporelle, on cherche à trouver les chemins les plus courtes entre eux avec l'algorithme de Dijkstra. Le long de ces connexions on coupe les lignes et groupe les correspondantes.

Une fois ces correspondances établies, les groupes des lignes correspondantes sont paramétrés indépendamment des autres lignes.

Pour établir la cohérence temporelle de la paramétrisation, on utilise plusieurs mesures d'erreur comme contraintes et on utilise la méthode des moindres carrés pour minimiser l'erreur. Pour chaque sommet d'une ligne, on essaie de trouver une valeur de paramétrisation qui satisfait le mieux les contraintes suivantes :

- Conserver une valeur de zéro au début de la ligne et une valeur de un à la fin de la ligne
- Garder une distance uniforme entre les valeurs de paramétrisation des sommets qui correspond à leur distance après la projection
- La valeur de paramétrisation d'une partie de la ligne doit suivre le mouvement de la ligne. Il s'ensuit la cohérence temporelle.

On peut noter que les deux dernières contraintes se contredisent. N'avoir que la contrainte uniforme va aboutir au problème du glissement mentionné ci-dessus. L'autre extrême, n'avoir que la contrainte temporelle, va aboutir à l'étirement des valeurs et la texture peut être transformée d'une manière non voulue. En conséquence il faut trouver

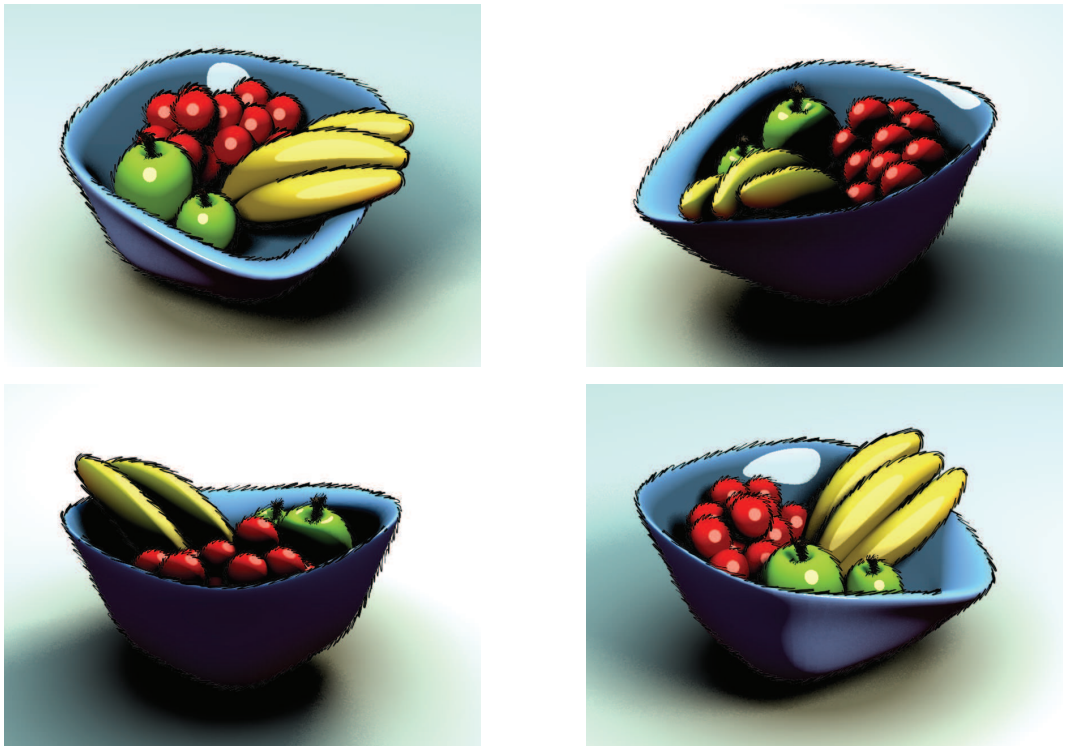


Figure 8 : Quatre trames extraites d'une animations montrants les contours stylisés de manière temporellement cohérente.

un équilibre entre les deux, mais comme cet équilibre est plutôt de nature esthétique, le mélange final est cédé à l'utilisateur (voir Fig. 7).

Analyse

On utilise deux méthodes pour trouver les événements rejoints/séparés parce que la méthode de vote ne trouve pas toujours les bonnes correspondances, surtout dans les cas où il y a plusieurs événements entre deux trames avec des lignes très courtes.

Les cas où l'extraction de lignes donne trop de lignes courtes posent un problème à l'algorithme. Cela arrive souvent avec les modèles peu lisses où plein de détails créant plein des silhouettes petites. Il n'est pas possible de simplement supprimer toutes les lignes qui sont plus courtes qu'un seuil donné car un tel seuil empêche de suivre la formation de lignes de leurs début ou peut supprimer une ligne pendant quelques trames pendant l'animation.

Vers une paramétrisation en temps réel

Une extension interéssante est la paramétrisation en temps réel comme par exemple dans le travail de Kalnins et al. [Kal+03]. Le problème principale, traité dans notre méthode avec l'analyse de la séquence entière, reste le fait qu'il n'est pas possible de prévoir les événements de visibilité. Donc, une certaine segmentation ne peut pas être evité, mais sa

réduction importante est possible. Nous avons exploré une approche potentielle un peu plus dans le détail en utilisant deux étapes. En premier, les valeurs de paramétrisation sont diffusées pondérées par une mésure de similarité d'une trame à l'autre pendant l'animation. Après, les valeurs sont reconstruites par une minimisation d'énérgie ressemblant à notre approche précédente. L'exploration de cette idée plus extensive reste comme travail de futur.

Dans cette section, nous avons proposé une nouvelle méthode pour la paramétrisation temporellement cohérente de manière globalement optimale. Nous utilisons une structure spatiotemporelle pour l'analyse de lignes et leur segmentation. Notre formulation d'énérgie se sert de résultats d'analyse ainsi que les choix de l'utilisateur pour déterminer la paramétrisation. Ensuite, cette paramétrisation permet, par exemple, la stylisation de lignes en mettant des textures de trait sur eux.

PBGI Quantifiée

L'illumination globale par la représentation par points

L'illumination globale est l'un des effets les plus importants en rendu photoréaliste. Il décrit le phénomène physique du transport de la lumière reflétée sur des surfaces de la scène. Chaque fois que la lumière touche une surface, elle est entièrement ou partiellement reflétée voire absorbée, suivant les propriétés de la surface. Une surface d'un matériau diffus va refléter la lumière uniformement vers toutes les directions de l'hémisphère qui se trouve dans la direction de la normale de surface. Cependant, un materiau spéculaire reflète la lumière en majeure partie vers une certaine direction avec peu de variance. Cet effet crée de la lumière indirecte, c'est-à-dire, de la lumière ne venant pas directement d'une source lumineuse mais reflétée une ou plusieures fois par des surfaces.

La méthode "Point-based Global Illumination" [Chr08] (PBGI) (l'illumination globale basée sur des points) est une méthode qui permet de calculer la lumière indirecte pour des surfaces diffuses. Dans une première étape, une hiérachie spatiale de points (un arbre) est créée. En chaque noeud une fonction représentante la reflexion de la lumière directe est stockée. Pour cela, la géométrie de la scène est echantillonée par des points et en chaque point, la lumière directe reflétée par la surface en ce point, est stockée comme une fonction sphérique. Seules les surfaces diffuses sont prises en compte, ainsi cette fonction ne contient que des basses fréquences cela permet de l'approximer par des Harmoniques Sphériques (HS) avec peu de coefficients. Les points forment les feuilles de l'arbre et sont accumulés dans les noeuds internes.

La deuxième étape est le rendu. Pour éclairer un point visible avec de la lumière indirect, la hiérarchie est traversée de la façon suivante. L'angle solide de chaque noeud parcouru est calculé par sa taille sphérique projétée sur le point, puis comparé à un seuil. Si celui-ci est inférieur au seuil, sa contribution lumineuse vers la direction du point à éclairer est stockée dans une image temporaire, appellé "framebuffer" (prenant en compte la visibilité des points en utilisant une image de profondeur) et la traversée de ce sous-arbre est arrêtée. Sinon, la traversée est continuée jusqu'à trouver un noeud dont la valeur d'angle solide est inférieure au seuil ou on arrive à une feuille.

Problème de mémoire Le problème principal dans l'application de la méthode PBGI est le stockage en mémoire des fonctions sphériques. Même en utilisant des HS de degré 3, la mémoire utilisée par un noeud compte déjà 27 réels (9 coefficients fois 3 canaux de couleurs, RVB). Pour des scènes typiques contenant des centaines de millions de points, la mémoire d'un ordinateur ne suffit pas à stocker ces données. Dans le suivant, on propose une méthode de réduction de consommation mémoire.

Principe

Redondance de données

Si on analyse les HS stockées dans les noeuds internes de l'arbre, on trouve une redondance importante dans ces fonctions. Cette redondance ne se trouve pas uniquement dans le voisinage spatial de l'arbre, mais également dans des noeuds bien éloignés spatialement

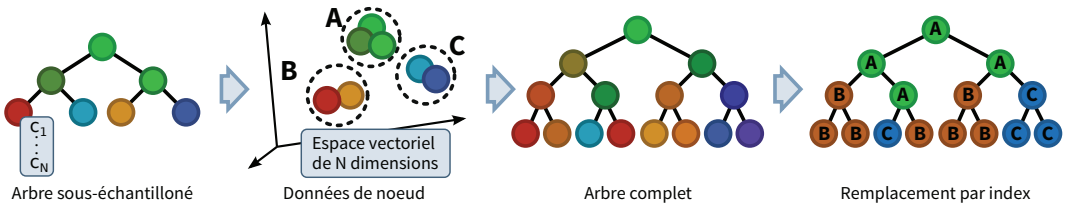


Figure 9 : Vue d'ensemble. A partir d'un sous-échantillonage de la scène, un arbre PBGI réduit est construit. Les fonctions sphériques des noeuds sont groupées en fonction de leur distance dans l'espace vectoriel de la représentation des fonctions sphériques. Le centre de chaque groupe est le représentant de son groupe. Les représentants forment une table de correspondances avec un index. Pour l'arbre complet, les fonctions sphériques des noeuds sont remplacées par les indices de la table de correspondances, réduisant la consommation de mémoire de l'arbre par un facteur entre 3.7 et 5.

et dans différents niveaux de l'arbre. Beaucoup de mémoire est donc utilisée pour le stockage de données qui sont essentiellement identiques. Nous proposons d'exploiter de ces redondances dans le contexte de la compression de fonctions spheriques pour réduire la consommation mémoire.

Vue d'ensemble

L'algorithme s'organise en deux étapes principales. Une première trouvant les redondances dans des HS d'un arbre généré à partir d'un sous-ensemble de points. Ces HS sont utilisées pour créer des représentants en groupant les HS dans leur espace vectoriel. A partir de ces représentants, une table de correspondances lie chaque représentant à un index. Dans la deuxième étape, l'arbre complet de rendu est créé et chaque HS dans cet arbre est remplacée par l'index du représentant le plus proche trouvé dans la table de correspondances (voir Fig. 9). Pour éviter d'avoir l'arbre complet dans la mémoire à un instant donné, le remplacement se fait à la volée.

Algorithme/Implémentation

Création de la table de correspondances

Pour trouver les représentants des HS, un arbre est construit sur la base d'un sous-échantillonage de la scène par rapport à l'échantillonage du rendu final. Comme pour l'abre finale, pour chaque noeud, les HS sont accumulées. Les 9 coefficients de HS forment des vecteurs de 27 dimensions (9 fois 3 canaux de couleurs), appellés vecteurs de données de noeud (VDN). L'ensemble de VDNs se trouve dans un espace vectoriel de même dimension. La proximité des vecteurs correspond à leur similarité et un amas de vecteurs correspond à la redondance des fonctions spheriques. On trouve ces redondances en employant l'algorithme des k-moyennes (relaxation de Lloyd [Llo82]), qui sépare l'ensemble de VDNs en k sous-ensembles correspondants à une partition de Voronoï. Le centre de chaque sous-ensemble agit comme son représentant, resultant en k représentants et tous les centres forment la table de correspondances où chaque entrée est identifiable par un index.

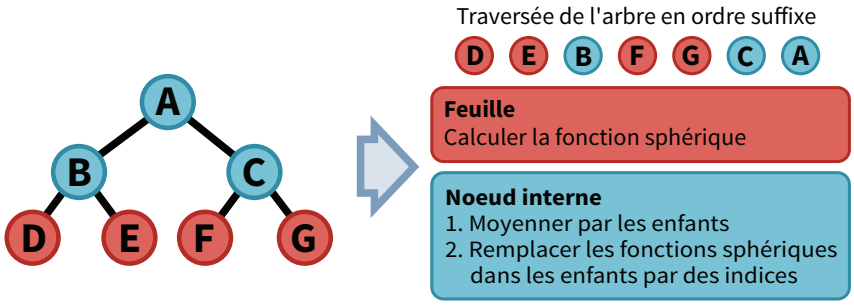


Figure 10 : Remplacement des HS par des indices à la volée. Pour éviter d'avoir à garder l'arbre complet dans la mémoire à un instant donné, les fonctions sphériques sont remplacées ``à la volée''.

Compression : Remplacement des HS par des indices

La compression se fait par le remplacement des fonctions spheriques dans l'arbre finale par des indices de la table de correspondances. En premier, l'algorithme commence comme la méthode PBGI normale, c'est-à-dire, la scène est echantillonée avec l'ensemble de points et la hiérarchie est construite à partir de ces points, toujours sans calculer les HS. Ensuite, les HS sont créées et remplacées par des indices à la volée (en "streaming"), c'est-à-dire, un après l'autre, pour éviter d'avoir les HS de tous les noeuds en mémoire à un instant donné. Pour cela, les noeuds de l'arbre sont parcourus en ordre suffixe. Si il s'agit d'une feuille, la représentation HS est générée et stockée dans la feuille. Si il s'agit d'un noeud interne, les HS des enfants sont accumulées dans le noeud. Puis, les HS dans les enfants sont remplacées par les indices des entrées de la table de correspondances qui sont les plus proches au sens de la distance euclidienne des HS des enfants (voir Fig. 10). Le parcours en ordre suffixe assure que pour chaque noeud interne, les HS de ses enfants ont été calculées avant qu'il soit traversé.

La compression finalement est obtenue en remplaçant les HS de 27 réels (108 octets) par un index de la table de correspondances. La taille de l'index depend du nombre d'entrées de la table. Pour $k = 1000$ entrées (une valeur moyenne), on a besoin de 10 bits pour encoder l'index. Par conséquent, on obtient une compression d'une facteur 86 pour les données des noeuds internes. La compression de l'arbre entier est moins importante car une grande quantité de données est stockée dans les feuilles n'utilisant pas des HS mais une représentation directe. Le facteur de compression de l'arbre complet se trouve entre 3.7 et 5, dépendant du type de l'arbre (arbre binaire ou octree) et du type de stockage de données dans les feuilles. Cette compression permet donc de faire des rendus de scènes constituée un nombre de points 3.7 à 5 fois supérieur.

Résultats

Nous avons comparé la génération des représentants par le partitionnement de VDNs en utilisant l'algorithme de k-moyennes à un choix aléatoire. Nous avons effectué plusieurs rendus avec un nombre différent de représentants aléatoires et générées par l'algorithme de k-moyennes comparant l'erreur des images résultantes avec la référence (un rendu sans compression). Nous avons quantifié les résultats par le nombre de pixels perceptuellement différents (PPD) [Yee04] et le PSNR (Peak Signal-to-Noise Ratio). En augmentant le nombre

Figure 11 : Résultats et Comparaison. Quelques résultats et la comparaison avec un choix aléatoire de représentants. Nous montrons le nombre de pixels perceptuellement différents (PPD) [Yee04] et le peak signal-to-noise ratio (PSNR) par rapport à la référence (image dénotée ` ` Lumière indirecte'') en format <PSNR/PPD> sous chaque résultat. On observe une baisse rapide de PPD et une hausse de PSNR qui sont supérieures à celles du choix aléatoire. En général, les résultats convergent rapidement à un niveau d'erreur peu perceptible.

de représentants par un facteur de dix, on observe une baisse de PPD plus rapide pour les k-moyennes par rapport au choix aléatoire et de même pour le PSNR, qui augmente plus rapidement pour les k-moyennes (voir Fig. 11).

Conclusion

Nous avons proposé une méthode pour la compression de données dans le contexte de la méthode PBGI par l'exploitation de redondances présentes dans les fonctions sphériques d'un arbre PBGI. En utilisant un procédé avec une gestion efficace de la mémoire nous arrivons à compresser de façon significative des arbres PBGI dépassant l'espace mémoire disponible. La comparaison de nos résultats avec des références montre des erreurs très faibles qui disparaissent rapidement en augmentant le nombre de représentants.

Conclusion et Perspectives

Conclusion

Dans cette thèse, nous avons exploré des structures de données dans le contexte de rendu. D'un part, des nouvelles structures, de l'autre part l'utilisation de structures existantes dans des nouveaux contextes. Le but de l'utilisation de ces structures est l'abstraction et simplification des données d'entrée pour permettre ou faciliter l'accès aux algorithmes. Il y a une forte correspondence entre l'algorithme et la structure. Dans la plupart de méthodes présentées dans cette thèse, la géometrie animée des scènes guide la forme des structures. Les valeurs specifiques gardées dans les structure sont dérivées de la géometrie mais également de l'apparence de données. Nous avons utilisé cet approche dans le domaine du rendu photoréaliste et non-photoréaliste, plus spécifiquement au contexte de la création des image binaires, de la paramétrisation temporellement cohérente des lignes animées et de la compression de données pour l'illumination globale.

Les problèmes examinés ont montré qu'il est de grande importance de trouver des structures bien adaptées à l'algorithme utilisé. Le problème peut être formulé plus facilement si une structure existe, qui bien représente ce problème. La bonne choix de la structure permet une expression claire de l'algorithme et aide également à l'implementation. L'algorithme dicte pour la plupart le choix de la structure. L'algorithme demande certaines données et suivant cette demande, des structures correspondantes sont développées. Leur fonction est principalement la bonne extraction et préparation des données d'entrée pour faciliter par exemple l'accès plus rapide ou l'application des algorithmes specialisés.

Les algorithmes et structures possèdent souvent un nombre important de paramètres permettants la modification du fonctionnement de l'algorithme ou du traitement des données. Dans la majorité des cas, les utilisateurs ne peuvent pas bien exploiter les paramètres venantes directement de l'algorithme. Au lieu de cela, il faut une exposition des paramètres qui représente bien les possibilités offertes par un algorithme. Cette exposition devrait être sur un niveau suffisamment haut que l'utilisateurs est capable de diriger les résultats vers une direction souhaitée sans avoir à comprendre l'algorithme en détail. Dans la technique présentée avant, l'ombrage binaire, nous avons exposé pour la plupart les paramètres directes de l'algorithme, rendant l'interface difficile à contrôler pour une utilisateur avec peu d'expérience. Dans ce cas, un remplacement des paramètres individuels par un paramètre mieux représentant les effets du changement de paramètre aurait facilité la manipulation dirigée des résultats.

Perspectives

Lors de la préparation de la thèse, nous avons trouvé plusieurs pistes de travaux potentiels, que nous regardons comme intéressant pour une recherche supplémentaire.

Méthodes de multiples vues Comme l'illumination globale par la représentation par points, il existe un nombre d'autres méthodes utilisantes plusieures vues de basse résolution. Ces vues contiennent beaucoup d'information dont seule une partie est utilisé (dans la plupart de cas pour le calcul de la lumière indirecte). Il devrait être possible de tirer beaucoup plus d'information de ces vues que cela est le cas dans les techniques existantes.

Pour la technique du rendu "pathtracing", plusieures méthodes ont été proposées utilisant un échantillonnage clairsemé de la lumière indirecte duquel de l'information supplémentaire a été tiré sans nécessité d'un échantillonnage additionnel (par exemple Dammertz et al. [Dam+10], Lehtinen et al. [Leh+12]). Suivant à cette idée, l'information contenue aux vues pourrait être également utilisé pour un calcul rapide des nouvelles vues sans avoir à acceder les données d'entrée. Une autre application est l'amélioration des solutions du transport de lumière en général pour des differentes méthodes, par exemple la simulation de la transluminescence. Ici, les vues peuvent être distribuées dans le volume d'un objet avec des densités et propriétés differentes permettant la simulation du transport de lumière à travers l'objet. Ces vues pourraient également aider à établir une cohérence temporelle dans le cas d'une scène animée. Les vues, distribuées non seulement dans la dimension spatiale mais également dans la dimension temporelle peuvent indiquer des changements et de la géometrie et des autres propriétés de la scène comme la lumière ou les materiaux des objets. Dans ce contexte, le développement d'une structure serait intéressant permettant cette interpolation dans les deux dimensions mentionnées.

Rendu stylisé Un sujet très important dans le domaine du rendu stylisé est la cohérence temporelle. La plupart des fonctions de calcul d'illumination issu d'une simulation physique sont temporellement cohérentes par défaut. Par contre, dans le domaine du rendu stylisé, cela n'est pas forcement le cas, par exemple dans le travail présenté ici de l'ombrage binaire trouvant une coupe minimale d'un graphe. On pouvait dans certains cas observer un changement important de cette coupe (et ainsi de résultat) avec très peu de changement de la scène. Conséquemment, un grand intérêt existe d'établir d'abord une classification et mésure de la cohérence temporelle Bénard et al. [Bén+11]. Correspondant à cette mésure est également la possibilité de trouver des facteurs communs entre des différentes méthodes du rendu stylisé, permettant une prédiction de la cohérence temporelle. Cela pourrait également permettre de trouver des points communs entre les méthodes et à partir de cela même la création de la cohérence temporelle avec très peu de connaissance de la fonctionnement des algorithes utilisés, par exemple par l'analyse de leur résultat en l'espace de l'écran. Cela pourrait même étendu à un système comme rtsc [Rus] combinant des algorithmes différents mais offrant une méthode pour analyse et établir la cohérence temporelle.

Finalement, avoir travaillé sur les deux domaines principales du rendu, photoréaliste et non-photoréaliste, je pense qu'il est très important dans le domaine du rendu stylisé d'éviter de s'occuper uniquement de la simulation numérique de techniques traditionelles comme la peinture ou du dessin. Les possibilités offert par la numérisation sont très nombreux et bien au délà et differentes de ce qui est possible avec les méthodes traditionelles. En conséquence, je propose une concentration plus importante sur des nouvelles méthodes de création des images sans oublier les connaissances des artistes traditionelles mais qui bénéficient plus de larges possibilités numériques

CHAPTER 1

INTRODUCTION

1.1 From 3D Scenes to Images: Rendering

Rendering is the process of generating a visual representation of virtual objects or scenes taking into account scene properties like geometry, lights, materials, environment and media (or a subset of these). It is extensively used in movie and TV production as well as product visualization, for medical applications, for design and architecture and illustrations.

In order to render a scene or, in other words, to create a single image of the scene, a viewpoint from which the scene is perceived, needs to be chosen as well as, in animated scenes, a point in time. An animated (or dynamic) scene can mean movement of the point of view as well as arbitrary changes of any of the properties listed above (e. g., deforming and moving objects, changing lights). The view point can be understood as a virtual camera, recording the scene much like a photo camera, using a 2D image plane. Rendering creates an image of the scene in that image plane as seen from the point of view. The image is made up from pixels and the goal of rendering is to assign a color to each pixel of the image, i. e., "shade" each pixel.

Rendering can be divided into different steps. First, visibility of the geometry as seen by the camera is established. Which parts are directly visible, which are hidden by other parts and which part of the surface is projected onto which pixel. Depending on the representation of the geometry, there are different ways to achieve this. The geometry (i. e., the surface of possibly complex objects) is often represented as sets of triangles (as the most simple geometric primitive in 3D) or polygons, but other representations exist like point clouds, spline surfaces (e. g., NURBS) or level-set surfaces. In the case of triangles, raytracing and rasterization are the most common choices to find the 2D projection of the 3D scene. While rasterization iterates over each triangle and projects them directly onto the image plane, raytracing follows the opposite way by casting a ray through each pixel which intersects the triangles. The closest intersected triangle gives the position visible through that pixel.

The technical goal of rendering is to assign a color to each pixel, i. e., to shade the pixel. This is normally a function of light, surface and material properties. Light hits a surface in a point and is reflected off of that surface in directions depending on the material at the hit location and the reflected color depending on the material's and light's color. Diffuse

Figure 1.1: Rendering in Movies. Top: Non-photorealistic rendering from the movie « Renaissance » (Onyx Films, 2006), which was recorded with real actors and virtual backgrounds and was converted into (pure) black-and-white as a post-process; photorealistic rendering from « Sky Captain and the World of Tomorrow » (Paramount, 2004), where everything but the characters are computer-generated. **Bottom:** Photorealistic rendering of a non-photorealistic environment in « Up » (Pixar, 2009), often found in animation movies; Slightly painterly rendering style trying to mimic hand-drawn styles in « Tangled » (Disney, 2010).

materials for example may reflect the light evenly in all directions independent of the direction of the incoming light. Glossy (i. e., somewhat specular) materials on the other hand reflect light in a more directed way depending on the relative position of the light to the surface point and the orientation of that surface.

In computer rendering though, arbitrary reflection functions and more general shading functions are possible. These functions may even be completely unrelated to physical light transport in how they relate the input data to pixel color.

In summary, the color for a pixel is found by first finding the part of the surface the pixel "sees" and extracting its geometric and material properties. Subsequently, this information is used in combination with a shading function (which in turn possibly uses other scene properties like lights, media like fog etc.) to calculate the final color. This process is done for each pixel, generating the rendered image.

The two main approaches to rendering are photorealistic and non-photorealistic, both can exploit the same input data to generate their results but can also ignore parts of the available input when it is not needed to achieve the desired result. Both are also combined to enhance results obtained from either technique.

Each rendering technique that takes large animated scenes as input needs ways of dealing with the large amount of data that complex 3D scenes contain. Due to its large amounts, the data is never used as-is but is always restructured or abstracted in some way, be it for faster, more selective access to the data directly (spatial partitioning) or to

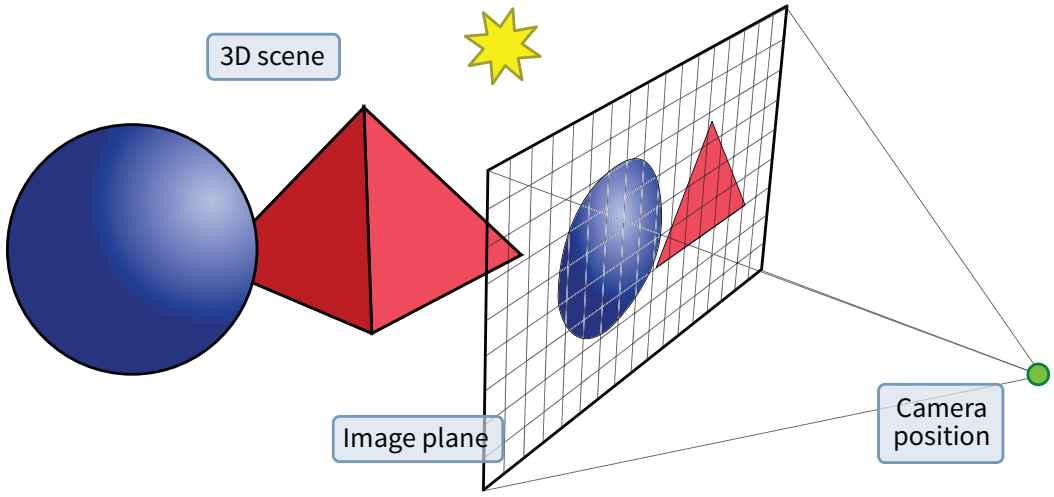


Figure 1.2: Rendering overview: The basic setup for a rendering consists of a 3D scene, containing objects, lights, materials and a point of view (« camera ») with an image plane onto which the scene is projected. Each pixel of the image is assigned a color according to the part of the surface projected onto it, its geometric and material properties and the lights.

create intermediate structures that allow different formulations of problems which are more adapted towards solving a given problem.

Via these intermediary structures, the information necessary for the rendering is extracted from the input data using corresponding algorithms. This pipeline leads from the input data to the rendered image.

In the following, we will first introduce the two main approaches to rendering, photorealistic and non-photorealistic, and then show the structures underlying these approaches derived from the (possibly animated) 3D scenes and how these are helpful in creating the type of renderings for which they are employed. We will then give an overview of the contributions in this thesis that are concerned with these structures in the context of rendering. In the context of image rendering, we propose new structures and the alternative use of existing structures by extracting information from geometry and other scene information.

1.2 Realistic Results: Photorealistic Rendering

Photorealistic rendering strives to achieve visually realistic (and therefore objective, i. e., indistinguishable from the real world) results.

In photorealistic rendering, two principal approaches exist: First, methods that are derived from the physical reality of light interaction with surfaces/materials. These give very accurate results but are often slow. Second, methods giving physically plausible results but whose internal models do not necessarily represent an accurate physical model. These are usually used in contexts where speed is important, possibly even real-time speeds (i. e., more than 30 frames per second).

The techniques derived from physical models usually give physically correct results and can easily be extended to capture a wide range of physical effects. In that sense, they

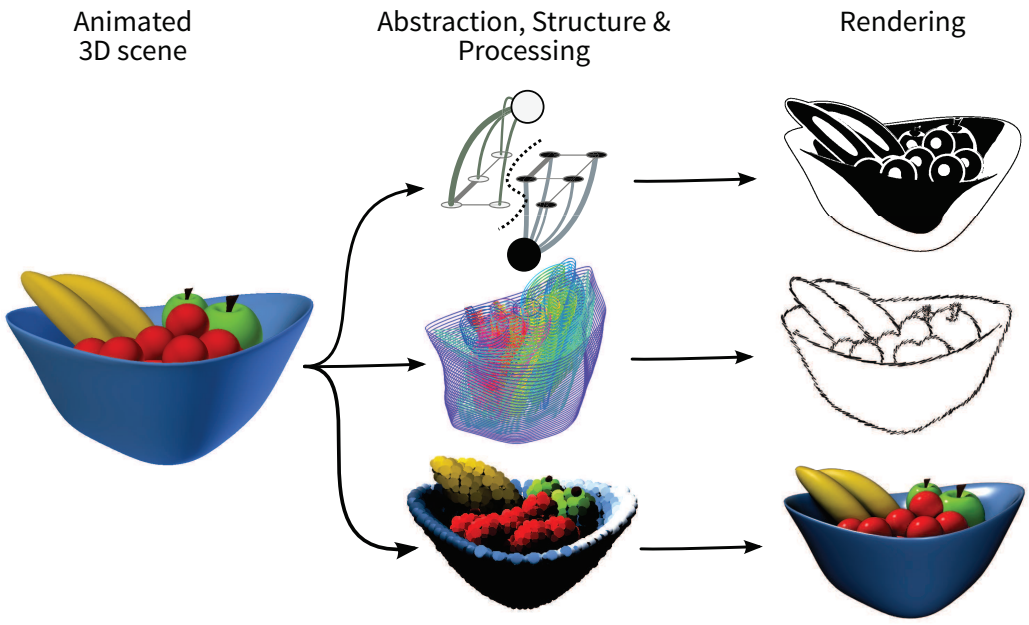


Figure 1.3: Scene Abstraction and Processing for Rendering: Starting from a single 3D input scene, a multitude of different results can be generated using different techniques. Of special interest are the intermediary structures which make techniques feasible and on which the algorithms work. This thesis proposes the development of new structures in the context of rendering and the application of existing structures in new ways for stylized rendering, line parameterization and texturing and global illumination. From top to bottom, the intermediary structures are a screenspace graph as used in Graphcut, a spatio-temporal surface obtained from animated lines and a hierarchical, point-based scene approximation used in the context of Point-based Global Illumination.

are often very general and converge towards correct results. This generality on the other hand is the reason that most of these methods are not very efficient. In production rendering they are not frequently used "standalone", unless physical accuracy is an absolute necessity, due to their comparably low speed. They are rather combined with other methods for the complete rendering and only used to render certain effects that are hard to accomplish with other methods. They are often used in the film industry and design visualization for example of objects that only exist as design studies. Most major films produced today use computer generated imagery (CGI), often for special effects but also for the creation of landscapes or whole cities in which real actors are embedded.

The more efficient methods usually constrain themselves to solve only a subset or even a single physical effect in a way that is not necessarily physically correct but allows the use of approximation structures which in turn allow the more efficient calculation of said effects. These are often used in games and other interactive applications, where the dynamic input scene needs to be quickly converted to a visible result. "Dynamic" here refers to change in the view or the scene itself (changes in the geometry, the materials etc.). Another field is the use in applications where exact results are not needed because the visual difference is too small to be perceived or not considered important enough. Often a combination of both categories is used.

Figure 1.4: Applications of PR. Left: Photorealistic movie rendering is an offline process, taking minutes to hours for a single frame, supporting highly tessellated geometry and physically-based rendering methods. Large portions of scenes (including characters) in the movie « Avatar » (Twentieth Century Fox) are completely digital. **Right**: Photorealism in computer games has steadily improved during the last three decades. In this example of the « Unreal 4 » engine (Epic Games), a large variety of effects are computed in real-time, including depth-of-field, soft-shadows and global illumination. Due to time constraints, the computation usually uses approximations to the physically correct simulation of light.

The physically-based methods are commonly based on raytracing, i. e., shooting rays from the camera into the scene and let the rays interact with the scene surfaces [Whi80]. This technique has been extended by indirect light calculations (also called Global Illumination), i. e., the light bounces off of surfaces in the scene, thus transporting illumination to not directly illuminated regions [Kaj86; Jen96; Vea97]. These methods create images with an error that can be reduced by increasing their computation time, therefore allow a trade-off between speed and accuracy. They will eventually converge to the correct result, usually in the order of minutes to hours (offline rendering).

Even though there are efforts to make raytracing methods viable for real-time applications by using the parallel processing capabilities of the graphics processor (GPU) [Par+10], the preferred way for real-time rendering is rasterization. Instead of tracing rays through a scene, the scene's geometry (usually stored as triangles) is projected onto the image plane and rasterized with an implicit ordering using a z-buffer or the painter's algorithm to ensure correct visibility.

A lot of work exists that combines rasterization with advanced rendering effects like shadows, reflections, indirect lighting etc. in a specialized way, i. e., each of these effects is basically independent from the others in the rendering pipeline.

Between real-time rendering and the aforementioned offline rendering methods, a trade-off exists between speed and quality. Especially in the computation of indirect lighting a mix of raytracing and accelerated methods can be found. Usually, this is made possible by the additive nature of light, i. e., the direct and indirect light contributions can be calculated independently (possibly with different methods) and the respective results can be correctly combined by simple addition to the final result. Examples are (Ir)Radiance Caching [War+88; Kri+05] which uses points distributed over the scene, where the incoming indirect light, computed via raytracing, is stored. The indirect light at any point in the scene can be calculated by interpolating between the stored points. Another example is Point-Based Global Illumination [Chr08] which uses a point-based approximation of the

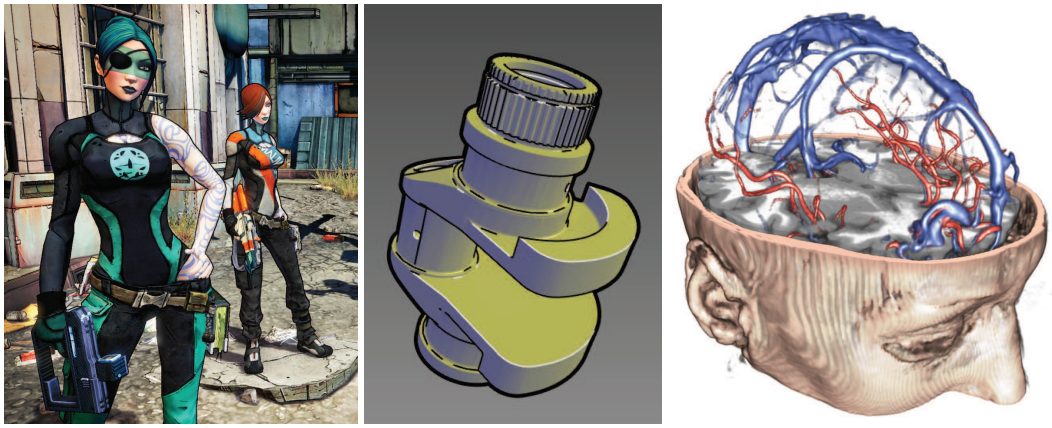


Figure 1.5: Applications of NPR. From left to right: NPR in computer games, here « Borderlands 2 » (Gearbox Software), using a cartoon style. Application in CAD software and technical illustrations allowing better estimation of shape (image taken from [Goo+99]). In the medical domain, for example rendering of volume data generated from MR images (image taken from [Rie+11]).

scene and the transported light to efficiently compute the indirect light at any point in the scene.

These methods, developed for efficient offline rendering of global illumination from diffuse surfaces have recently been turned into simplified real-time versions [Rit+09; Hol+11; Sch+12].

In most photorealistic rendering techniques, the user only controls quality constraints to affect the rendering speed but the final outcome of a scene rendering cannot be influenced directly. The massive amount of information contained in the 3d data though can be also presented in other ways than photorealistic, nameley in more directed, non-photorealistic rendering.

1.3 Expressive Results: Non-photorealistic Rendering (NPR)

For a long time, the goal in computer-based rendering was to achieve photorealistic reproductions and a constant progress towards that goal can be observed. On the other hand, realistic depiction is not a sensible goal for every application. The principal interest in rendering is conveying information and often, this information is not well represented in a photorealistic rendering using physically correct BRDFs and light sources or optical lens effects like depth of field etc. For example renderings in the medical domain, expressing the form and relation of organic body parts, do not profit from a physically correct rendering. On the contrary, the amount of information contained in such a depiction may well have the opposite effect, notably showing too much irrelevant information, hiding the information that should actually have been conveyed. A. Gooch [AG10] gives an overview on possible mappings of input data to easier perceivable and understandable alternative representations.

A good representation is largely also a matter of human perception. In order to know in what ways to abstract from physically-based depiction, knowledge about perception is

important, i. e., how a human observer perceives, which features carry most information and which may actually be detrimental for conveying specific information. To understand these relations, several studies exist, looking into the perceptive side of rendering, for example what type of rendition allows the best perception of a given shape [Win+07]. In the domain of line drawing, studies try to find where artists would draw lines for a given view of an object and which lines are the most important ones to convey the shape of the object [Col+08; Col+09]. These studies help guiding the development of NPR algorithms by allowing to understand which data can most easily be ommitted and which needs be kept to convey a specific property.

The motivation for NPR is (at least) twofold: First, the aforementioned simplification and reduction of the data with the aim of exposing only specific information, displaying the information in limited environments or using the output for crafting purposes. Second, the generation of interesting, possibly emotion evoking imagery, oriented often on artistic techniques like paint brushing, pencilling and other drawing and painting styles as well as the simulation of the media like paper, canvas etc.

One of the means in traditional arts has always been the abstraction and reduction of information leaving only those parts which are actually meant to be conveyed. Many artists though also have very recognizable styles and throughout history, many distinct art directions evolved, each featuring a unique, recognizable style. In NPR, a number of techniques exist that orient themselves on traditional artistic styles maybe even simulating the process an artist might use. Such systems [Goo+02; Van+07; Lu+10] often offer ways of creating temporally coherent renderings, making them suitable for animated scenes. Techniques also exist geared towards more recent artistic styles, like cartoon styles or expressive black and white depiction [Bar+06; MG08; Eis+08]. All these methods provide a certain amount of control to the user to create his own creative styles by defining the parameters like brushes, canvas type etc. Another possibility is the recreation of a given style, i. e., to render a given input in a different style, possibly stemming from an artist. In these style transfer techniques, an input image or 3d scene is analyzed for its defining visual properties which are then applied to a second “unstylized” input [Her+01; Lee+10; Ngu+12].

Recalling the medical example given at the start of this section, NPR is concerned with conveying information and the creation of methods and tools that allow a user to extract and display a certain subset of the available input data. The central part in this remains the user who has the information he wants to convey, possibly covered among other information, and the target audience that is supposed to receive what the user wanted to convey.

In medical imaging and computer-aided design (CAD), the visual output is often in a style similar to that of technical illustrations [Goo+99; Rie+11]. In both domains, NPR representations are usually preferred since the clarity of the exposition and the possibility to remove unnecessary details can be very important. For example, in a schematic drawing of a technical device, shadows, reflections and similar effects from the physical domain can distract easily and hide the important features.

Interactive maps are getting more and more popular, especially due to their use on mobile devices. Traditionally, these maps show everything in scale, not regarding the importance a building or an area might have for the user even though landmarks like large

buildings, towers etc. can be helpful for the orientation. Similarly for touristic purposes, emphasizing interesting parts of a city while de-emphasizing others can give hints which parts might be worth a look. This is well known and used in traditional map drawing, where landmarks and important buildings are drawn out-of-scale or details are removed from uninteresting areas. When this idea is translated to the digital domain, it can be applied in an interactive way to 3d maps, creating similar depiction as mentioned above. Some possibilities include focusing the view on certain areas by displaying them in greater detail or exaggerating important buildings [Tra+08; GD09]. Similar for 2d maps, even abstract ones like metro maps: zooming into a region until necessary details (like station names) become identifiable may cause a loss of context to the rest of the map. Instead, it can be easier and faster to navigate if only a small region, indicated by the user, is magnified in an adaptive, image-aware way [Pin+12].

Other goals in NPR are the creation of results for uses beyond the digital domain. This is often of interest for manufacturing purposes, when an image or a 3D object should in some way be transformed into physical reality (Digital Fabrication). One example is the the creation of stencils, i. e., images consisting of back- and foreground with all foreground parts being connected [Bro+08]. Here, two requirements need to be balanced: how to best convey the input data with two colors and how to connect the foreground parts. One can also consider special output devices with limited amounts of color (for example e-book readers with two colors) where specialized renditions are useful.

1.4 Scene Abstraction and Processing for Rendering

Most rendering methods use special structures and organization of the data on which their algorithms work. These are often spatial, temporal or spatio-temporal structures, but also others, more specialized ones. These structures are abstractions or simplifications of the data and represent it in such a way that calculations can be executed more efficiently or at all. They often are the core of an approach for solving a problem.

In the rendering of 3D scenes, spatial 3D structures are used while in image-based methods (mostly in NPR) 2D structures are employed. The traditional structures in 3D rendering are spatial search acceleration structures to quickly find the geometry in a certain region of space. Examples are binary space partitioning using arbitrary (BSP) or axis aligned (kD-tree) splitting planes, regular 3D-grid trees (octrees) and bounding volume hierarchies (using primitives like spheres or boxes).

Instead of using structures that partition the whole space, often a cheaper alternative can be hash maps, especially for sparsely filled space. While these can encode data in arbitrary dimensions, they are mostly used in spatial hashing. In principle, a function exists that takes an element of the data that is to be hashed and returns an integral number. Usually, a certain interval of (possibly multi-dimensional) input space gets mapped to the same number, the hash, and each input value falling in a given interval is stored under the same hash value. This can be used for example when searching for points in a certain neighborhood, since the hash (and the hashes of neighboring cells) can easily be computed and used to obtain neighboring points.

In recent years, ray tracing and path tracing techniques have been improved with multi-dimensional spatio-temporal sampling structures. Instead of independently sampling the

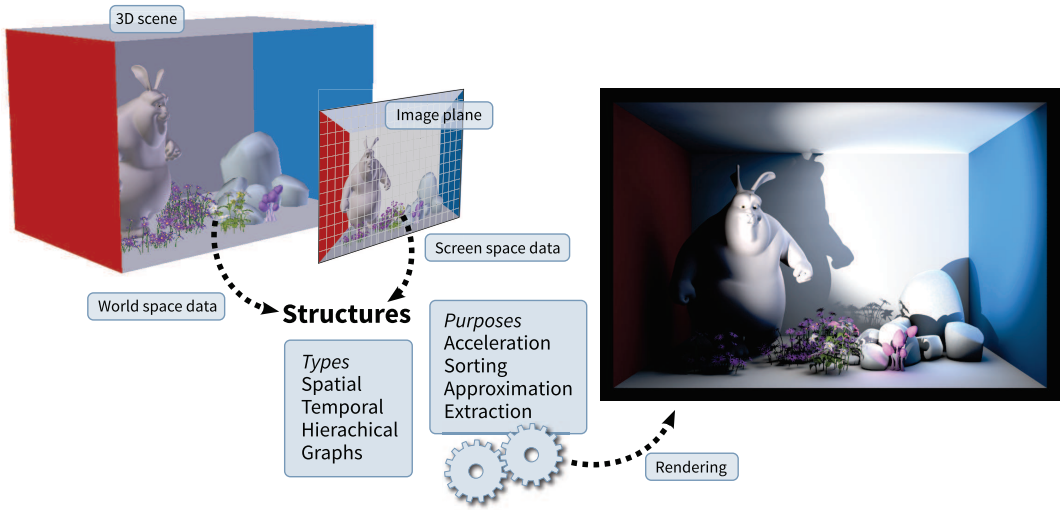


Figure 1.6: The possibly animated 3d geometry data in object and image space (i. e., projected data) is processed and abstracted into structures used for rendering the scene. Depending on the algorithm, these structures are of different types and purposes, for example a spatial structure to accelerate the access of the geometry data for a given ray through the scene.

image, lens, time, BRDFs, light sources etc., a BSP tree is used over many sampling dimensions allowing for adaptive sampling not only in image space (as is the case with adaptive anti-aliasing) but in all the dimensions integrated into the BSP [Hac+08a]. Sen and Darabi [SD12] use a feature vector structure combining several dimensions related to sampling and the scene. The information from that vector is used to bilaterally filter the sampled color resulting in strong noise reduction.

Often, data is not needed at its highest degree of detail but rather the information in an area or region as a whole is of interest. In texture mapping, when shading a pixel, a color value is requested from the texture map. An approximation to the correct color is the convolution of colors of the pixel's (approximately isotropic) footprint in the texture map. In order to avoid summing up pixels under the footprint, an image pyramid is created, a hierarchical structure where each level is the filtered image of its previous level. Obtaining the color value can now be achieved by linear interpolating values taken from this hierarchy, depending on the size of the footprint.

In a similar spirit, point-based scene representations abstract from the details in the scene by replacing the actual scene's geometry by a point cloud structure. To allow even further abstraction, the point cloud can be converted into a hierarchy, combining several points into "larger" points. This is done in Point-Based Global Illumination, where each point approximates a part of the scene's surface. The hierarchy groups points in such a way that higher levels approximate even larger parts of geometry with increasing error. Now, when trying to find a property of a surface part, the hierarchy can be queried and depending on the permitted error, the query will be vastly accelerated when compared to directly querying the points themselves or even the scene's actual geometry representation.

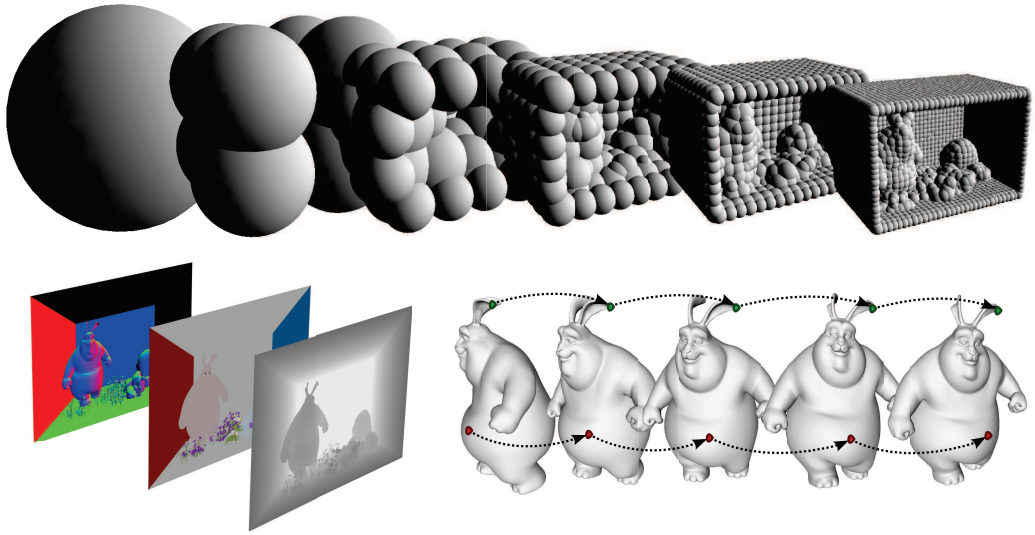


Figure 1.7: Examples of data structures used in rendering. Top: Spatial hierarchy using spherical bounding volumes, allows accelerated traversal of the scene and geometric approximation. **Bottom-left**: Geometric and appearance information stored in screen space (G-Buffer), used in Deferred Shading, allows filtering of scene data in screen space. **Bottom-right**: Temporal tracking of scene geometry, allows establishing temporal coherence for example in sampling.

Such hierarchies can also be realized using graph structures that relate a set of vertices (or nodes) via edges to each other. Usually, graphs are not hierarchical, but this can be achieved by grouping nodes into "supernodes". A large advantage of casting a problem into a graph problem is the huge amount of research on Graph Theory and the number of existing algorithms. In Computer Graphics, segmentation of data (mostly in image and volumetric applications) poses an important problem and has seen strong improvements through the recent introduction of the Graphcut algorithm (see Sec. 3.2.2) which necessitates the formulation of the problem as a special graph.

Temporal structures are of interest for enforcing temporal coherence, one of the major topics in real-time non-photorealistic rendering and other fields. Particularly in NPR in stylization approaches, changing the input slightly can result in a large change in the output, possibly leading to visually unpleasing artifacts. To improve this, objects or features are tracked through time and related to each other in graph structures. Keeping track of changes over time allows then to combine or separate features and objects or to adapt the output to avoid visual artifacts.

1.5 Contributions

In this thesis, we present new structures and the use of known structures in a new context in the domains of NPR and realistic rendering. Starting from a possibly animated 3d scene, we extract those parts of the data needed for the algorithm, either directly from the 3d information or the information as seen from a certain view. This data is introduced in

an intermediary structure from which additional information is derived which is used to define the pixel colors of the final rendering.

Using Appearance and Geometry for Binary Shading Artists often use reduced palettes (down to the minimum of two colors) for depicting scenes with large regions in a strongly stylized and expressive manner. We propose a new method to generate two-color (binary) renderings from 3D scenes. Conveying the shape and the material of an object with only two colors is a considerable challenge and simple approaches like thresholding a rendered image do not give satisfying results. We therefore combine geometry and appearance information but unlike other methods, we do not combine them in the rendering step. Instead, we extract different features into separate channels and combine them afterwards in a separate screen-space process. At this step, the user has the control over the combination of the channels, enabling the generation of a large range of different styles of two-color images.

Temporally Coherent Parameterization of Animated Lines Another abstraction technique is line drawing of 3d scenes, usually depicting the lines as uniform black strokes. However, uniform lines, while conveying shape, cannot convey surface properties. To evoke the impression of specific surface properties, textures on the lines can be used. When such a scene is animated, the lines will smoothly move along with the animation, i. e., they change in a temporally coherent way. To allow the same behaviour for the textures, the lines need to be parameterized in an equally coherent manner. Current methods achieving this work in real-time. While the lines change over time, topological events occur where lines split and merge. However, real-time methods can not take future events into account, leading to an ongoing segmentation of the lines. We propose the representation of the line data as a space-time surface which allows to analyse the evolution of the lines from the complete animation. We introduce two different analysis schemes and use them to find and handle these topological events in such a way that the subsequent parameterization is independent of them. To achieve temporal coherence, we propose a user-controllable trade-off between screen-space and object-space coherence. Our method can render textured lines from arbitrary animations and will avoid oversegmentation even for longer animations due to the temporal analysis.

Data Compression in the Context of Point-Based Global Illumination In Point-Based Global Illumination, the scene is represented in a hierarchical point structure. The inner nodes in that tree contain spherical functions for the visible area and the diffusely reflected light, stored in low-frequency approximations like Spherical Harmonics (SH, see Sec. 6.3.1). Large scenes can generate huge amounts of data using this representation which are currently handled by out-of-core approaches. We propose a scene-aware method to reduce the memory consumption: in the spherical functions, one can observe a strong redundancy all over the hierarchy, due to the slow changing nature of diffuse reflection and repeating materials. We propose the exploitation of this redundancy in order to compress the data in the hierarchy. We find representatives for the redundant data using an implicit analysis of the function space and replace the actual functions by indices to the representatives in a memory-efficient way, effectively quantizing the function space. Other techniques com-

pressing trees with similar information usually only compress locally while our approach is able to find similarities anywhere in the tree. Rendering with the quantized nodes shows no significant differences in practical scenes.

1.6 Thesis Organization

In the following chapters, we present several new structures used as intermediary representations of 3d scenes and alternative uses of existing structures. We also introduce algorithms corresponding to these structures with the purpose of creating 2d images from 3d scenes. The presentation of these contributions is largely divided into non-photorealistic (Part I) and photorealistic rendering (Part II). Each part is started by introducing methods and structures used in the specific domain, followed by the presentation of our contributions. We close with a conclusion and some thoughts on perspectives of how future work could relate to and profit from this work (Part III).

Part I

Non-photorealistic Rendering

CHAPTER 2

LINES AND REGIONS

3D data can be represented in different ways. In certain contexts, the focus is not to create a photorealistic depiction using the complete set of input data (consisting of geometry, materials, lights etc.). On the contrary, it is often of interest to show only particular details or features of the data in a rendering. The methods not aiming for a photorealistic rendition are usually subsumed under the term "non-photorealistic rendering".

A large body of work is concerned with the (semi-)automatic creation of stylized images as artists might produce them manually, for example the simulation of drawing and painting techniques in contexts like artistic painting, technical illustration or stylized drawing. In the following, current non-photorealistic techniques and data structures for the creation of stylized renderings and images are presented, focusing on those that produce images with reduced palettes (to the extreme of two colors) and line drawings, since those form the basis of the contributions presented in the following chapters.

The human visual system derives a lot of its information about the properties of an object and its materials by visual cues coming from the interaction between the light and the object's surface. Many of these cues can be subtle but are important nonetheless. For example, the roughness or glossiness of an object's material may be easily derived from the gradients of the highlights. When these gradients are removed, the assessment of the material properties can become more difficult. Similarly for an object's shape, a psychophysical study by Winnemöller et al. [Win+07] designed for dynamic systems shows that Lambertian shading performs best in terms of providing cues for the shape while mere texturing without shading information performs worst. Contour lines rank between these two (see Fig. 2.1 for a comparison of different shading types).

2.1 Lines

The general notion is that it is difficult to represent material properties without some kind of shading. Therefore, most feature line drawing techniques constrain themselves to conveying geometric features (not taking into account shading techniques like hatching that are technically also line drawing techniques).

A large number of line drawing algorithms exist, and most of them take the view position into account. Silhouettes (contours) are the simplest form. They are are located where

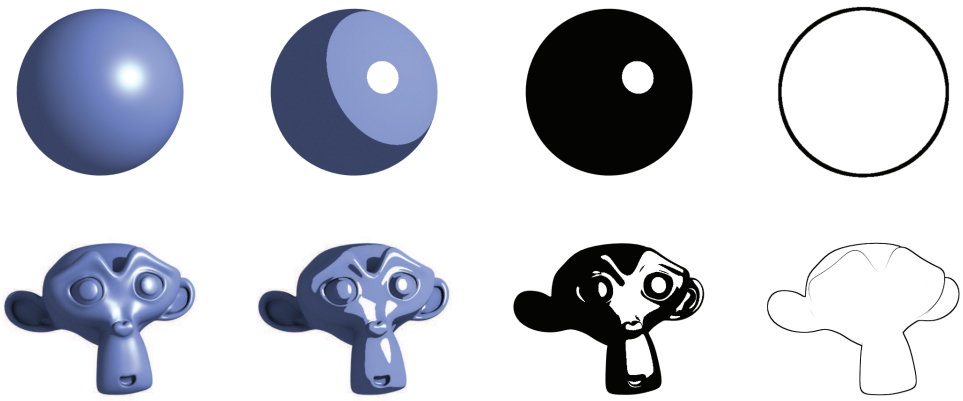
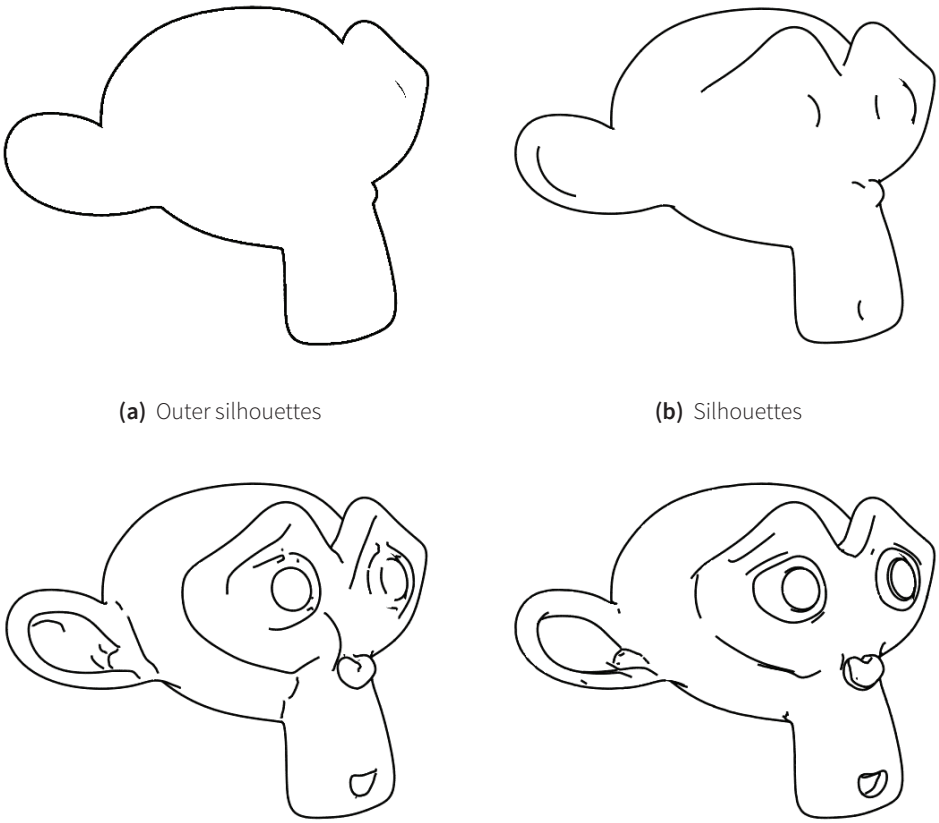


Figure 2.1: Conveying shape by shading. Lambertian + Phong shading (a) allows for the best assessment of the depicted shape. While shading with reduced palettes like cel-shading (b) and (naive) two-color shading (c) make it already harder to recognize the shape, using mere silhouette lines (d) gives very few cues about the depicted shape.

the surface normal direction is perpendicular to the viewing direction. Outer silhouettes are a subset of the contours where an interface between the object's surface and the background exists (see Fig. 2.2(a) and 2.2(b)). Depending on the depicted object, these contours may or may not give some impression of the shape, usually though their depiction of shape is not successful. The reason is that in order to well depict shape, line drawings need also to take into account curvature that is not necessarily that which corresponds to the surface currently being perpendicular to the view, but possibly neighboring views or the object's surface curvature itself.

In order to understand line drawings better, two studies were conducted, raising two important questions: The first, how well can lines convey shape [Col+09] and the second, where do humans draw lines in order to achieve this [Col+08]? For the first question, Cole et al. performed a study showing that a line drawing can convey shapes to a large degree just as well as a shaded image of the same object would. Errors in perception only occur locally depending on the properties of the lines used. The other question of where *humans* draw lines is interesting in so far as artists have studied line drawing for centuries and have a good understanding of where to best place lines to maximize their effect. This also becomes obvious in the second study's results [Col+08], stating that a large number of lines for a given object is drawn by all artists. These lines are identified by all artists as important for conveying the shape while others are seemingly less important and therefore not necessarily shared by all artists.

While proposed before these studies, line drawing works were presented which already took into account some of these results. DeCarlo et al. [DeC+03] propose suggestive contours, which are contours not visible in the current view but from neighboring viewpoints. To avoid suggestive contours which would lie directly in front of actual contours, a contour from a neighboring viewpoint is a suggestive contour only when no (radially)



(a) Outer silhouettes **(b)** Silhouettes

(c) Suggestive Contours **(d)** Apparent Ridges

Figure 2.2: Line Types. The most simple line descriptions are silhouettes, and while the lines described by these works are part of the subset humans would draw, they are not sufficient to give a good impression of the shape. Suggestive Contours (c) and Apparent Ridges (d) are extending the simple description of silhouettes. The first by taking into account « nearby » contours, i. e., those points that become contours when seen from neighboring views. Apparent Ridges describe contours as those points that are the maximum of the view-dependent curvature. (c) and (d) have been rendered using rtsc [Rus].

closer viewpoint exists in which this contour exists as well. This definition creates to kind of additional contours which complement the normal contours. The first are anticipated contours, i. e., those which will become normal contours when changing the viewpoint slightly in their direction. The second are extended contours, as they will attach at the (visible) end of a contour and continue it for a bit. This formulation of suggestive contours combined with normal contours conveys the shape of an object significantly better than the contours alone (see Fig. 2.2(c)).

Remarking that human perception is sensitive towards edge-like features, Judd et al. [Jud+07] propose a line definition that uses view-dependent (i. e., apparent) ridges. While the non-view-dependent ridges and valleys use the first and second derivative of the surface normal, here these are derived in a view-dependent manner. View-dependent curvature means the amount of change of the normal on the surface projected onto the image

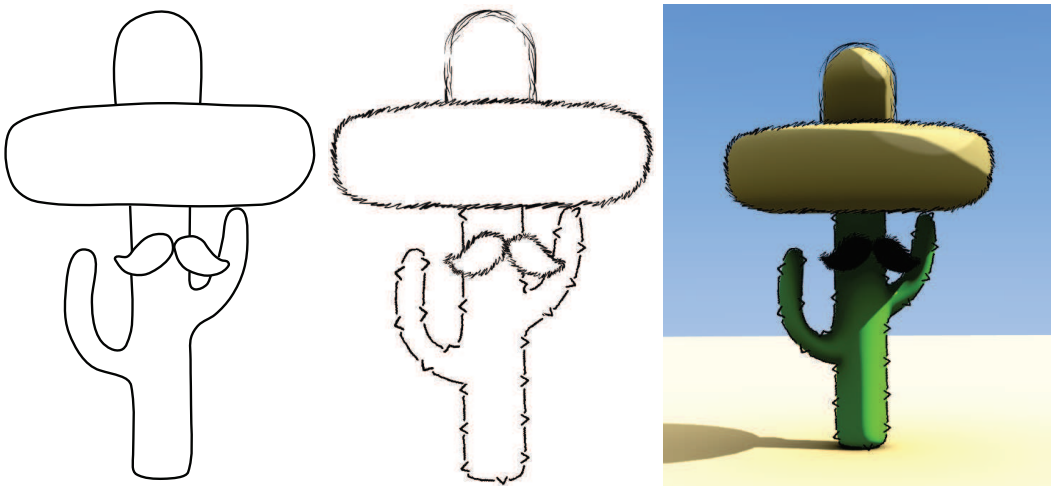


Figure 2.3: Expressive Lines. **Left**: Pure line drawings (i. e., no hatching or other cues about the surface) with uniform strokes can not well convey surface properties. **Center**: Giving the lines a brush texture however, allows hinting at the composition of the surface to a certain degree. In this example, the mustache of the cactus has a hairy texture, while the cactus itself gives the impression of being thorny. The hat is a bit lighter on the top and somewhat rougher around the brim. **Right**: The lines combined with a rendering, giving the finished image.

plane. This definition can be considerably different from the curvature on the surface itself. The apparent ridges are those points where the view-dependent curvature is maximal (see Fig. 2.2(d)). This definition can also contain valleys, depending on the viewpoint and the geometry. It also draws lines at edges which are very important for the understanding of the underlying shape which Suggestive Contours would not draw since they don't correspond to contours for any nearby view.

These techniques propose where to draw lines. Often, especially in an artistic context, it is desirable to apply strokes on these lines that may resemble drawing tools like brushes or pens. Previously was stated that lines can not depict material properties. Using strokes and textures *on* the lines themselves however, they can be made expressive. For example, rendering a cactus with an outline that features little thorns (that do not necessarily correspond to any geometric feature of the underlying object) will convey the impression of a "thorny surface", while a wiggly line on the other hand could convey softness (see Fig. 2.3). This means that it is possible to convey certain properties using a very reduced depiction like line drawings with fairly simple means.

2.2 Regions

While the line drawings focus mostly on geometric features, using colored regions for the depiction allows for a stylized rendition of geometric and material cues. Especially the material cues depend a lot on the possibility of using a wide range of colors and shades (e. g., shaded objects usually feature smooth transitions between bright and dark). Therefore, a particular challenge is the depiction with reduced palettes, be it for artistic purposes or constraints of the display (the latter is rarely they case nowadays).

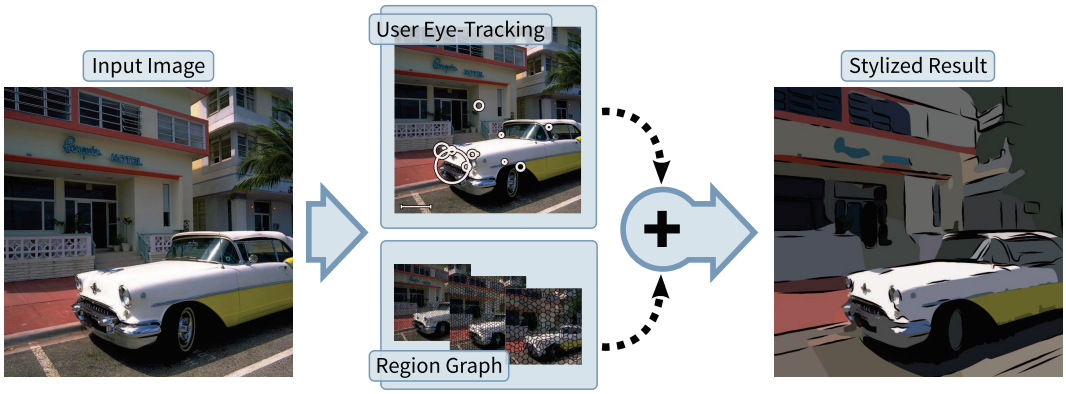


Figure 2.4: Stylized Abstraction of Photographs. An input image is analyzed for its important regions by eye-tracking an observer. This importance map is then used to guide the abstraction using a region structure segmenting the image hierarchically. Images taken from [DS02] except « Region Graph » illustration.

An obvious example is the case of cel-shading (also called toon-shading, Fig. 2.1(b)). Here, a glossy highlight is usually either at full strength or absent with no visual transition between the two states, in essence like a threshold. Diffuse surfaces, usually featuring a smooth transition of color caused by the slow change between the surface normal and the incoming light direction, are usually depicted by a single or a few banded shades of the same color (hue).

In most cases, the results are images where regions are depicted in a constant color. The definition of a region can be derived geometrically, from the appearance of the underlying material or the interaction of both with the scene's lighting. The abstraction achieved in this way should still be conveying the features of the underlying data or even enhance its perception.

At the same time, many techniques combine line drawing and region generation. Often, the lines form the dual of the regions, in the sense that the regions are separated naturally along feature lines and the lines act as the complement of the regions emphasizing their separation.

Color palette reduction can be achieved explicitly by first defining the regions and then coloring them or implicitly by changing color values in such a way that regions emerge automatically.

DeCarlo and Santella [DS02] show abstraction and stylization of photographs in such a way that certain regions are emphasized while others are deemphasized. Regions of interest (RIO) are found by analyzing the eye movement of a spectator of a the given image. Over multiple scales, the image is segmented into regions and boundaries which are then related to each other, resulting in an hierarchical image region structure. The final segmentation is then generated from the hierarchy, taking into account the previously recorded RIOs by pruning the hierarchy tree earlier in regions of low interest and going further towards the leaves (i. e., smaller regions) as interest increases. Finally, the regions are colored in with constant colors derived from the original image and possibly separated using lines, giving the image a manually colored and drawn expression (see Fig. 2.4).

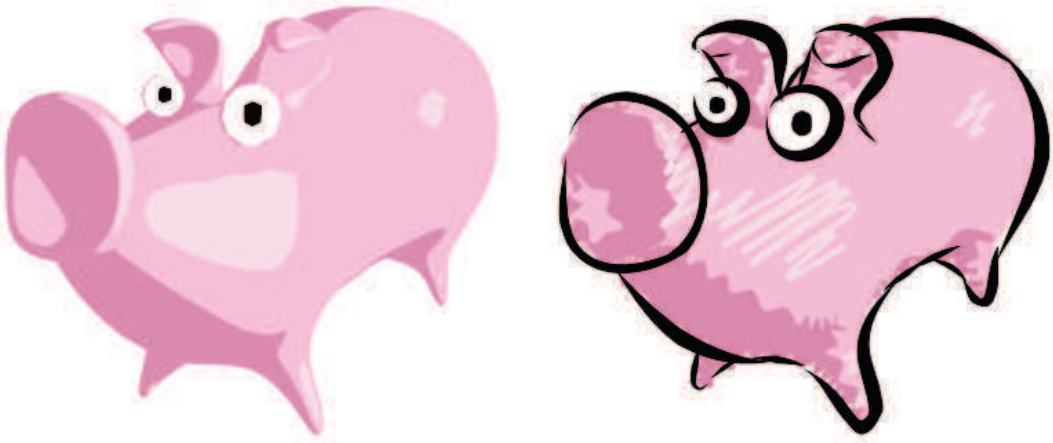


Figure 2.5: Stylized Vector Art. Given a lighted 3d model, its contours, highlights and shadows are extracted, forming coherent regions described by vector curves. The regions and their surrounding borders are stylized, resulting in a clip-art look. Images taken from [Eis+08].

Wen et al. [Wen+06] propose the generation of images similar to watercolor painting with large uniform regions and smoothly drawn lines as separators. Their process is divided into two steps. First, the user interactively segments the input photograph into regions and a fore- and background. The region boundaries are softened and depicted with black strokes in the final rendering. In the second step, the regions are colored in. Simply using the average color for each region does not lead to convincing results. In order to achieve a better and more artistically steered coloring, graph structures for fore- and background are used in which the regions are vertices and adjacent regions are connected by an edge. A color database is then used to assign each region its final color in a globally optimal way, taking into account artistic distribution of colors. These final colors usually deviate in tone and value from the average.

Vectorization or (closed) edge representation is an obvious choice for defining regions explicitly. In the 2d domain, i. e., the explicit conversion of images/photographs to regions, the process usually consists of smoothing the input data and a subsequent quantization step. For example, Olsen and Gooch [OG11] employ these methods by using an edge integral convolution mainly to smooth and simplify the edges. Then the resulting smoothed image in quantized using a given mapping. Finally, the edges in the quantized image are clear enough that a 2d polygon tracer (e. g., [Sel03]) can be employed to generate the final vector representation.

In the 3d domain, given a model, vectorization can be accomplished from a certain viewpoint using inner and outer silhouettes as a starting point. Eisemann et al. [Eis+08] generate a set of smoothed closed curves (regions) from the contours. In order to depict shadows, another set of curves is generated from the contours as seen from the light source. Highlight regions are generated by contracting the previously generated regions. As a final step, all hidden contour lines are removed. For stylization purposes, these regions are then optionally textured and the contours drawn with stylized lines (see Fig. 2.5).

With the goal of controlling the simplification of objects depending on their importance in the scene, Bezerra et al. [Bez+08] propose another way to find temporally and

Figure 2.6: 3d Dynamic Grouping for Guided Stylization. Starting from a 3d scene (left), a number of representative samples is clustered in feature space and a « level of abstraction » map (middle) is generated from the clustering and given features. This map is then used to guide the abstraction and stylization of the scene (right). Images taken from [Bez+08].

spatially coherent groups in 3d scenes. To allow for realtime computation, the groups are not directly derived from the geometry but representative points are uniformly sampled over the scene's surfaces. These points are clustered temporally coherently using a modified mean-shift clustering. The clustering can take not only position into account but arbitrary features from the scene giving the user control over the objects and their level of abstraction. The final grouping of the scene geometry is then found by remapping the representative points to the geometry (see Fig. 2.6).

With the implicit methods, the range of stylization is larger as it remains possible to keep small features or gradients which would usually be removed by the vectorization. Note that the 2d methods presented here mostly work on the color values of the input images, i. e., they do not take into account any geometric or material data of the underlying scene. Cel-shading on the other hand is also an implicit method but (not necessarily) constrained to color space transformations but can rather work in the shader's parameter space.

Kang et al. [Kan+09] propose a method based on the edge tangent flow of the input image (i. e., the directions perpendicular to the image gradient). The flow is used in the extraction of lines steering a difference-of-Gaussian filter and in a preliminary image smoothing combined with a bilateral filter. The image smoothing ensures the removal of noise and its combination with the flow field creates natural regions in the direction of image boundaries. The smoothed image is then quantized in the luminance channel and combined with the extracted lines.

Comic styles are among the most common styles recreated with regions. Winnemöller et al. [Win+06] propose a comic style rendering of videos where temporal coherence is of importance. After the smoothing step similar to the previous work, in this case a bilateral filter, a soft quantization (as opposed to standard "hard" quantization) is effected on the luminance channel. During the animation, small changes in luminance occur, which would normally be quantized into different bins, possibly leading to artifacts like flickering (i. e., jumping colors). The soft quantization avoids this by having small, smooth transitions from one quantization level to the next. To enhance the comic look, the quantized image is combined with a line drawing generated from the smoothed image using a "difference of Gaussian" filter (see Fig. 2.7).

This can be brought to the extreme by using only two colors for the depiction, to convey shape and material cues as "economically" as possible. Line drawing and region two-color

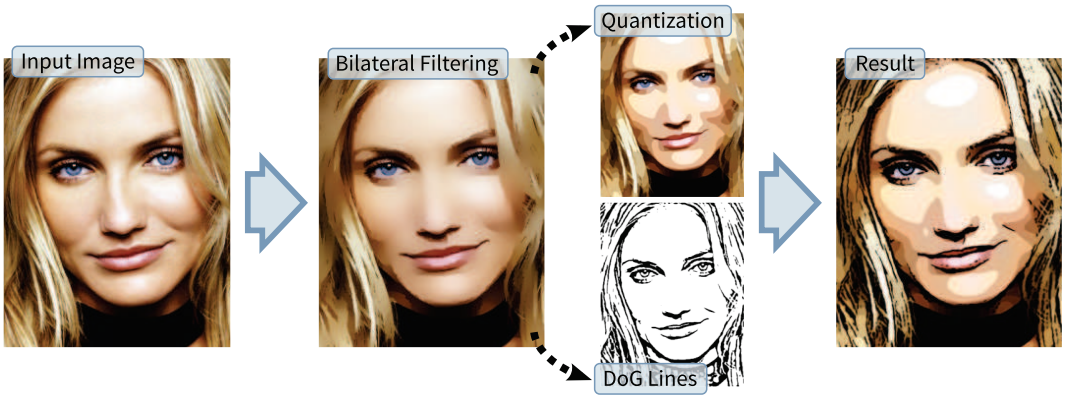


Figure 2.7: Real-time video abstraction. This image space method takes a photograph as input which is bilaterally smoothed. This smoothed image is quantized to generate colored smooth regions and converted into a line drawing using a « difference of Gaussian » (DoG) filter. The combination of lines and quantization results in the comic-like look. Images taken from [Win+06].

shading techniques can be understood as two extreme opposite ways of depicting a scene. A closer look at pure two-color shading techniques is taken in Sec. 3.1.

The image-based methods presented here can only take into account the color to guide their approach, while a lot of useful information to guide the stylization may not lie in the appearance but also in geometry and surface properties. In the following, we first present a new approach to create two-color images from 3d scenes using geometry and appearance information featuring large regions (Chap. 3) and secondly a new technique to generate a temporally coherent parameterization of lines extracted from animated 3d data using a space-time structure to analyze the line development. (Chap. 4).

CHAPTER 3

BINARY SHADING

In this chapter, we propose a new method for the rendering of stylized binary images. We extract different geometry and appearance attributes from a scene and combine them in a deferred step in an image-space process which uses a graph as an intermediate structure relating the gathered scene information via a Graphcut to the desired binary segmentation.

3.1 Stylized Black and White Images

Depiction of an object requires at least one foreground and one background color. Oftentimes it is desirable or necessary to restrict the depiction to just two colors; examples are artistic black-and-white illustrations and images generated from stencils. In technical terms, such images are commonly referred to as bilevel or binary images. We consider the problem of generating binary images by means of rendering a 3D shape. There are several ways of conveying tone using just two colors, notably half-toning [FS76; Uli87], which can also be used for the creation of artistic images [OH95; OH99]. We are rather concerned with depictions that mostly assign black color to dark regions and white color to bright regions of the image, inspired by the popular black and white style of several graphic novels, and recent graffiti art.

The black and white illustration by comic book artist Alex Ross displayed in Fig. 3.1 is a quintessential example of this style. In analyzing drawings such as these, we find that the composition combines aspects of tone reproduction with rendering of geometric features. As expected, large black or white regions in the drawing correspond to low and high levels of illumination, respectively. Furthermore, high local contrast in illumination can override the absolute illumination levels. For instance, a coherent area which is darker than surrounding regions might be depicted in black, even when the illumination levels are all fairly high. Finer details are often depicted this way, such as the musculature on the arms and torso in the image on the right. At the same time, the particular boundaries between black and white are selected to effectively convey the shape. These boundaries might run along surface features like ridges and valleys, such as along the ridge of the nose or the boundary of the brow. Or they might run along occluding contours (discontinuities in depth), such as on the cape. They also appear to be as short as possible, given these constraints. Thin lines, which are often haloed, may also be used to further delineate parts

Figure 3.1: Hand drawn illustration of the *Batman* comic (DC) by Alex Ross. Note that assignment of white and black color combines aspects of tone reproduction with rendering of geometric features (see main text).

of the shape. In many situations, including such lines is crucial – the bottom of the cape is drawn in white so it is easily separated from the black background, but the lines along the occlusions on the bottom of the cape are what clearly convey its wing-like nature. In other work, the effects of highlights or shadows can be largely discounted when artists make these decisions.

In terms of rendering, a pixel may be black or white because of the illumination of the shape, or because the contrast between neighboring pixels of different color helps to better convey the geometric structure of the shape, regardless of the shading. These goals are often contradictory, meaning that taking into account only *appearance* or only *geometry* is insufficient. Here, appearance refers to various terms that are commonly used in realistic shading (Sec. 3.5.1), while geometry refers to geometric properties of the shape relative to screen coordinates (Sec. 3.5.2). Note that a binary image could be generated from the appearance of the shape alone by thresholding [KZ93; LL98; MG08; XK08] or from geometry using only toon-shaders [Dec96; Bar+06].

An artist would need to make a range of decisions to arrive at a successful depiction using only black and white, balancing the two contradictory goals. Thus, we model the process of assigning black or white to the pixels as an *optimization* problem that takes into account both appearance and geometry.

The optimization problem to be solved is essentially a segmentation of the image into black and white pixels. As in other recent approaches, we model the image as an undirected graph and determine the solution using graph cut energy minimization [Boy+01]

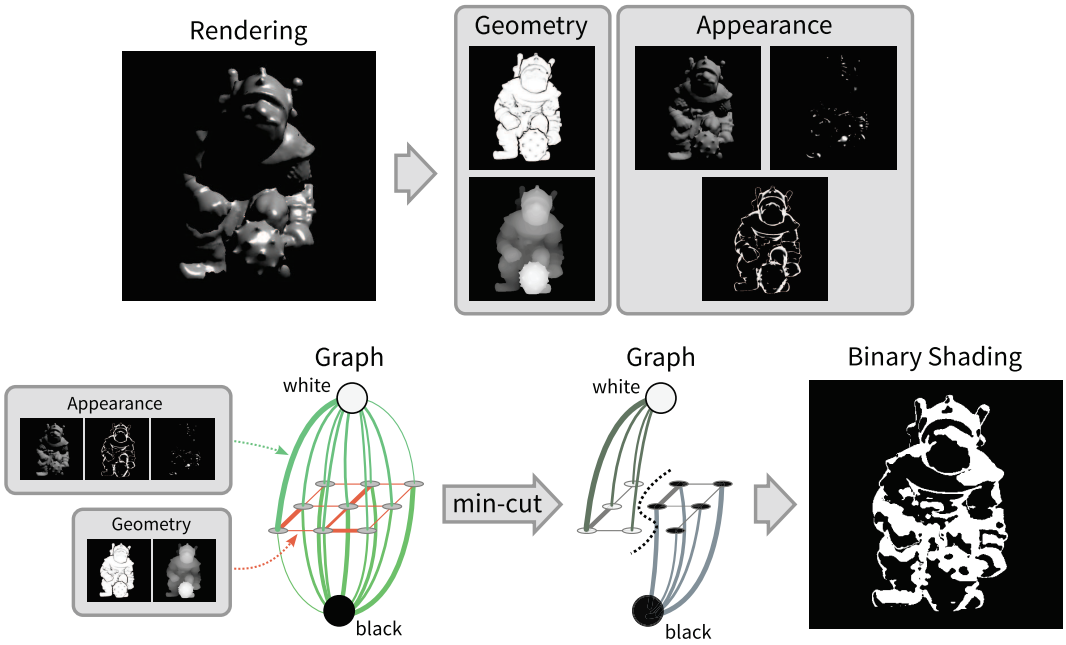


Figure 3.2: Overview. Several sources of information about the appearance and geometry of the scene influence the edge weights in a graph representing the image (terminal edges in green are influenced by appearance, while neighborhood edges in orange are influenced by the geometry). The black and white assignment is computed by finding the minimal cut in the graph. Thus, the output respects geometric features, and not only appearance.

(see Sec. 3.2 for an overview). This graph acts as an intermediate structure between the 3D scene and the final binary image.

Although the form of the function being optimized is restricted, these restrictions allow for efficient algorithms for its solution. (Note that because of its connection to the maximum-flow problem, the typical terminology used for parts of the graph are related to flows. Here, instead, we use terms related to colors and pixels, and note the flow terms parenthetically.) In this framework, there is a node in the graph for each of the pixels in the image. There are also two additional nodes in the graph (terminal nodes) that represent the color white (the source node) and the color black (the sink node). The edges in the graph are either connections between neighboring pixels, connections from a pixel to the white node, or connections from a pixel to the black node. Each edge has a non-negative weight (capacity) that signals the strength of the connection. Initially, edges connect pixels to each of their neighbors, and to both the white and black nodes. Any cut in the graph that separates the white and black nodes (the source from the sink) is a possible depiction in black and white. An appropriate depiction in a binary style is determined by computing the minimal cut – the globally minimal set of edges (by summing their weights) that when removed separates the white from the black pixels. An overview of this approach is diagrammed in Fig. 3.2.

The main contributions of our work are the use of geometric information in this style of depiction, and further, modeling the separate influence of appearance and geometry elegantly as weights on the two different types of edges in the graph:

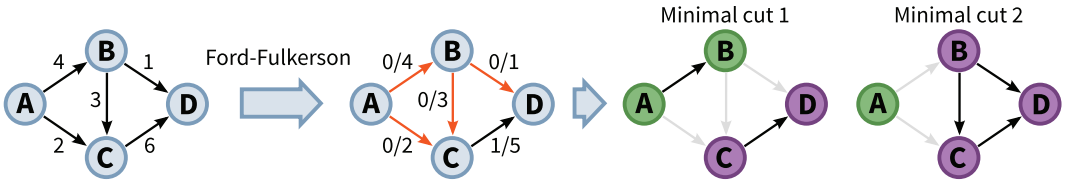


Figure 3.3: Max-flow/min-cut. Example graph with source A and sink D and the capacities per directed edge. The Ford-Fulkerson algorithm is applied, which finds unsaturated paths from source to sink and increases the flow over the found path by the minimum remaining capacity on any edge of the path. It terminates when no unsaturated path from the source to the sink can be found. The edges with no remaining capacity form the set of minimal edges and any cut through this set that separates sink and source is considered a minimal cut. In this example, the maximum flow is 6 (amount outgoing from the source). The capacity of each minimal cut equals the maximum flow.

- appearance defines the weights of edges connecting pixels to the white and black nodes, and specifies the tendency for pixels to be black or white,
- geometry defines the weights of edges connecting neighboring nodes in the image, and specifies the tendency for pixels to be different colors because of geometric structures.

The primary challenge here is to assign weights to edges so that the resulting minimal cut represents a coherent binary image. After an overview in Sec. 3.4, we describe how weights are assigned using appearance (Sec. 3.5.1) and geometry (Sec. 3.5.2).

We show a range of results in Sec. 3.7. In particular, we argue that with our approach it is possible to generate results that cannot be produced with simpler approaches such as thresholding or using local decisions. Following this, we discuss the effects of using our method in animation in Sec. 3.8.

3.2 Background

We view the generation of binary images as a segmentation problem, i. e., assigning regions in the image to foreground or background. Several instances of this problem have lately been modeled as a max-flow/min-cut problem – the so called graph cut approach [Boy+01; BK04; Rot+04].

3.2.1 Max-flow/Min-cut

Let $\mathcal{G} = (\mathcal{V}, \mathcal{E})$ be a graph with a set of vertices $\mathcal{V}$ and edges $\mathcal{E}$ connecting the vertices with an edge e defined as $e = (u, v)$ with $u, v \in \mathcal{V}, u \neq v$. Each edge has a capacity $c(u, v)$ which defines the maximum flow $f(u, v)$ that can go through this edge. Let there be two special vertices, the source s and the sink t. The former has only outgoing flow while the latter only incoming, all other vertices in the graph have a neutral flow balance, i. e., the sum of incoming and outgoing flow is zero.

The amount of flow $|f|$ in a graph is the flow leaving the source (and reaching the sink due to the zero-flow sum in all other vertices). The maximum-flow problem seeks to maximize $|f|$.

The maximum-flow/minimal-cut theorem for a graph states that the maximum flow from s to t corresponds to the sum of edge capacities of the minimal s-t cut. A s-t cut through a graph is a set of edges that separates the graph in two disjoint sets of vertices $\mathcal{S}$ and $\mathcal{T}$ where $s \in \mathcal{S}$ and $t \in \mathcal{T}$. The minimal cut is the s-t cut of which the sum of its capacities is minimal. Therefore, the maximum flow through a graph is the sum of capacities of the minimal cut.

An algorithm to solve this problem was first formulated by Ford and Fulkerson [FF56]. It works by iteratively finding unsaturated paths from source to sink, i. e., paths that contain edges with remaining capacity. When such a path is found, flow is added along this path and its amount is that of the lowest capacity left on any edge along this path. The remaining capacities of the edges are then reduced by the amount of added flow. The algorithm stops when no path with remaining capacity can be found (see Fig. 3.3).

Historically, the applications lie in network problems (e. g., physical, transportation, electrical etc. networks) where certain quantities need to be routed from one point to another over several possible ways. This problem has been adapted in Computer Graphics/Vision problems, known as Graphcut.

3.2.2 Graphcut

The goal of Graphcut [BJ01] is to allow interactive segmentation of images (or video frames) into background and foreground regions. This problem is formulated as the max-flow/min-cut problem. The graph is directly derived from the image pixels, i. e., each pixel corresponds to a vertex, and the edges are 8- or 26-neighborhoods around each pixel. Naturally, an image consists of pixels with no source or sink. Therefore, these two terminals are introduced as special vertices in the graph and each vertex is connected to both. In the following, the user sparsely marks pixels as either background or foreground. From this initial segmentation, constraints are derived: The pixels marked directly have hard constraints with a maximal capacity towards the terminal it was marked for (usually source corresponds to foreground and sink to background). The terminal capacities (i. e., on the edges leading to a terminal) of unmarked pixels can be derived in different ways, depending on the problem. One possibility is to use a similarity measure between an unmarked pixel and those marked as foreground or background, for example from intensity histograms generated using the marked pixels. The neighbor capacity between two vertices can equally be formulated as, for example, the similarity in intensity of their corresponding pixels.

The quality of any segmentation is formulated as an energy taking into account these two measures as the sum of the regional terms (i. e., their matching with the assigned terminal) and the boundary terms (i. e., the similarity of two neighboring vertices assigned to different terminals). The optimal segmentation is the one whose energy is minimal. It is shown that the minimization corresponds to finding the minimal cut through the graph for which any max-flow/min-cut algorithm can be employed. In this work Graphcut is also used to find a segmentation of an image albeit not with the goal of finding an optimal foreground/background segmentation but in the context of stylized imaging.

3.3 Previous Work

Our goal is to create large regions of black and white pixels that convey the essential features of the shape. Perhaps the simplest such operation on images that works towards this goal is thresholding. For most inputs, a global threshold (see Lee et al. [Lee+90] for an overview) will not preserve important features, which is why thresholds are typically adapted locally [KI86; YB89].

Recently, algorithms for the binarization of photographs have been considered for the purpose of creating artistic or abstract images. In particular, Mould and Grant [MG08] as well as Xu and Kaplan [XK08] are considering the same problem we do, albeit starting from an image and not a 3D scene. In both approaches, desirable properties of a binarization are described using an energy function that is a weighted sum of terms. This energy function is optimized to yield the image. While the approaches differ in how the energies are defined, their general strategy is similar: there are terms for encouraging similarity to the input (in terms of absolute and local contrasts), preservation of image features (i. e., as region boundaries), and smoothness of regions in the result. Mould and Grant optimize the energy by casting it as a min-cut problem while Xu and Kaplan use simulated annealing [Kir+83].

A closely related problem to binary shading is the creation of a papercutting [Xu+07] or stencil [Bro+08]. In this case, regions of black and white are replaced with where material is cut out, and where it is retained. However, there is an additional constraint, which has been the main focus of the research: ensuring the result is a single connected piece of material. The binarization step is performed using simple thresholding, and the pieces are connected with lines that minimize particular cost measures.

Performing binarization in a 3D rendering pipeline has also been studied in the context of real-time stylized shading. Spindler et al. [Spi+06] combine a simple toon-shader with haloed lines. Stylized forms of shading that are exaggerated or expressive [Rus+06; Ver+08; Ver+09] combine enhancements for increasing contrast locally with adjustments of illumination based on the surface geometry. However, these decisions are made locally – each point is shaded independently, and thresholded. As a consequence, boundaries between regions will not necessarily run along surface features, and thus lack an element of coherence that our approach provides (see Sec. 3.7 for comparisons).

Artistic black and white rendering is quite well studied for other styles than the one we are considering here. In particular, drawing feature lines in black or white [DeC+03; Lee+07; Jud+07] results in informative renderings, which might be combined with toon-shaders to produce a different black and white rendering style [DR07].

3.4 Overview

We model the problem of assigning black or white to each pixel in the image as a graph cut: each pixel is a node in a graph; all pixels are connected to two nodes that separately represent white and black (the terminal nodes – source and sink). In addition, each node is connected to a set of nodes representing neighbors in the image. We define the weights (capacities) of the edges in this graph using the full data available from the 3D scene, namely appearance and geometric surface properties. Finding a cut that disconnects the white from the black node effectively assigns the pixel nodes to either black or white. Our goal

is to find the cut with minimum total edge weight, and we do this using existing optimization methods [Boy+01].

Assume we wish to compute a binary image of $m \times n$ pixels in size. Our shading technique proceeds in three steps (summarized in Fig. 3.2):

1. **Rendering**: Generate arrays $\mathbf{A}_i \in [0,1]^{m \times n}$ capturing different channels of the appearance of the object and $\mathbf{G}_i \in [0,1]^{m \times n}$ capturing geometric properties of the shape relative to screen space. The arrays can be generated using any rendering technique. We choose ray-tracing for the flexibility it offers during experiments.
2. **Graph construction**: Edge weights from pixel nodes are defined based on the values in the arrays $\mathbf{A}_i$ and $\mathbf{G}_i$.
   - **Appearance – edges to white and black**: For pixel (x, y) in the image, let $W[x, y]$ and $B[x, y]$ be the weights of edges that connect its node to the white and black nodes, respectively. These are defined as the linear combination of $W_i[x, y]$ and $B_i[x, y]$ (using w_i and b_i), where i is the index of the appearance component $\mathbf{A}_i$:

$$\begin{aligned} W[x,y] &= \sum_i w_i \, W_i[x,y] \\ B[x,y] &= \sum_i b_i \, B_i[x,y] \end{aligned} \tag{3.1}$$

     The derivation of W_i and B_i from $\mathbf{A}_i$ is explained in Sec. 3.5.1. The values w_i and b_i manage the relative influence of these different appearance components (see Fig. 3.4).
   - **Geometry – edges to neighbors**: Let $N[x_0, y_0, x_1, y_1]$ be the weight of edges connecting pixel (x_0, y_0) to (x_1, y_1). These weights are the product of a positive constant $\tilde{N}$ and factors N_j that are derived from the geometric components $\mathbf{G}_j$:

$$N[x_0, y_0, x_1, y_1] = \tilde{N} \prod_j N_j[x_0, y_0, x_1, y_1]. \tag{3.2}$$

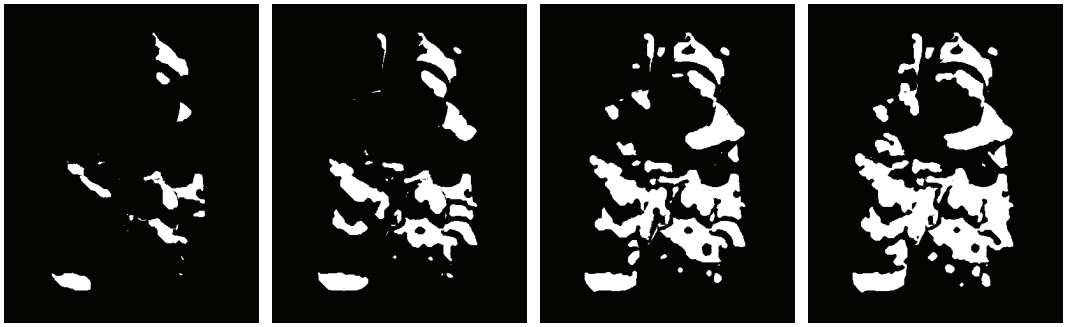
     The details of deriving N_j from $\mathbf{G}_j$ are discussed in Sec. 3.5.2. $\tilde{N}$ controls the relative weight of these edges compared to edges connected to white and black.
3. **Minimal cut**: Based on assignment of weights, a minimal cut is approximated following Boykov et al. [Boy+01]. This minimum cut defines an assignment of pixels to white or black, based on the two connected components: one contains the white node, and one contains the black node.

The approximation of the minimum cut is fast enough to enable the user to iteratively improve the result by adjusting parameters used to combine the various appearance and geometry terms.

3.5 Contributions to the Graph

3.5.1 Appearance Contribution

The appearance of an object depends on many material and illumination parameters and it is beyond the scope of this work to attempt a systematic exploration. We demonstrate our approach to binary shading using simple appearance components:



$w = 0.35$ $w = 0.5$ $w = 0.75$ $w = 1.0$

Figure 3.4: The effect of changing weights w on edges connecting pixels to white. Throughout these changes, $b = 0.5$.

Diffuse: The diffuse component is computed using a Lambertian reflection model: its value is the cosine of the angle between the surface normal and lighting direction. It is independent of the viewpoint.

Specular: The specular component takes into account the viewing position, the light source and the surface normal. The Beckmann distribution is used to calculate the specularity of a point.

Head-lamp: By placing a light at the camera, and using a Lambertian model, we shade based on surface orientation with respect to the camera. This term helps separate the object from the background by being darkest along the silhouette (on smooth parts of the surface).

While our approach is not limited to this list of appearance attributes, we have found that it generates enough variability for different binary rendering styles. We generate all values by ray-tracing and remap them onto the range $[0, 1]$ into arrays $\mathbf{A}_i[x, y]$. We show the influence of different appearance components (upper row) on the weighted binary shading (lower row) in Fig. 3.5.

The assignment of weights based on the $\mathbf{A}_i[x, y]$ alone will only consider global features. We know from previous work that we must consider local contrasts as well. Consequently, we model both global and local features explicitly. For the global contribution of each component we simply use $\mathbf{A}_i[x, y]$ and $1 - \mathbf{A}_i[x, y]$ as contributions to the weights $W_i[x, y]$ and $B_i[x, y]$, respectively.

For modeling the local contribution to the weights we relate the values in $\mathbf{A}_i$ to spatial averages of $\mathbf{A}_i$. Let the average value $\overline{\mathbf{A}}_i[x, y]$ be based on a neighborhood in image space, i.e.

$$\overline{\mathbf{A}}_i[x, y] = \sum_{u,v} \mathbf{A}_i[u, v] e^{-\frac{(x-u)^2+(y-v)^2}{h_i^2}} \tag{3.3}$$

where h_i is a parameter to control the size of the neighborhood. For practical purposes we cut off the Gaussian weight function by only considering pixels within a square window.

We consider the difference of appearance attributes and their local averages:

$$\mathbf{A}'_i[x, y] = \mathbf{A}_i[x, y] - \overline{\mathbf{A}}_i[x, y]. \tag{3.4}$$

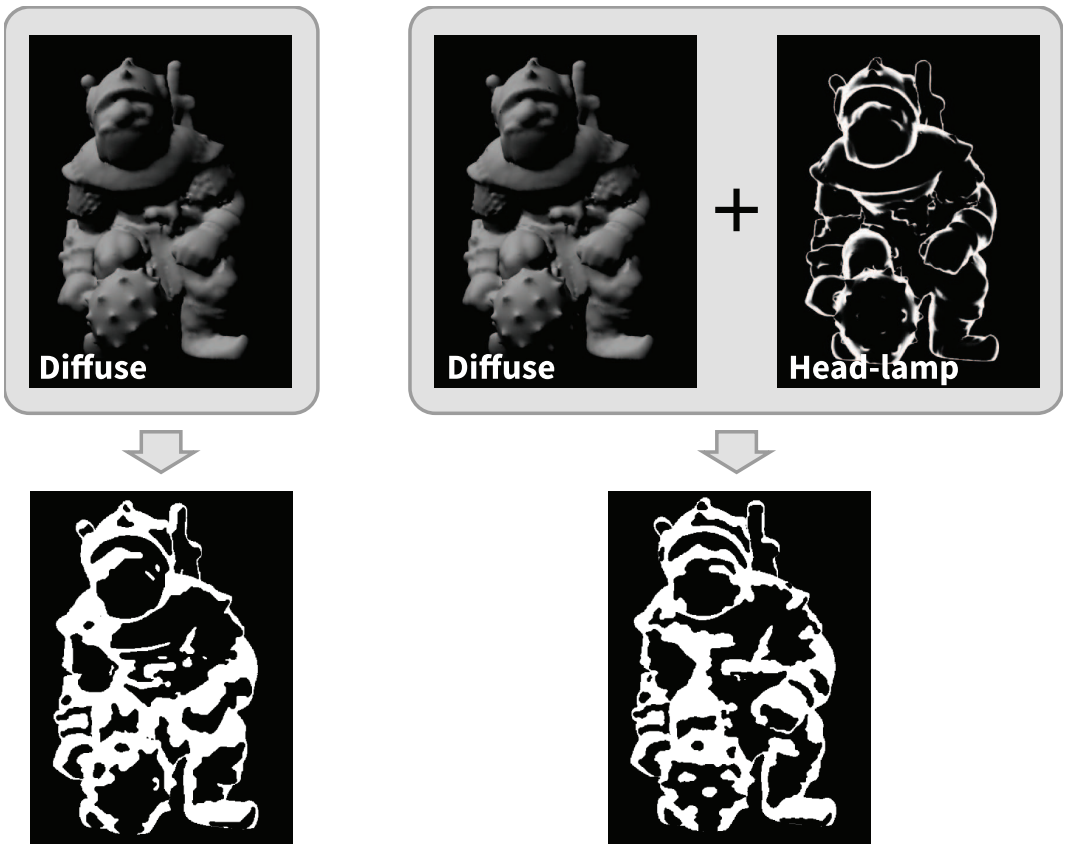


Figure 3.5: Top: Appearance arrays rendered from the 3D model. Bottom: influence of these arrays on binary shading; diffuse (left), diffuse and head-lamp (right).

The sign of $\mathbf{A}'_i[x, y]$ determines whether pixel (x, y) is locally brighter or darker, and thus whether it should tend towards white or black. Thus we weight white as $\max(0, \mathbf{A}'_i[x, y])$ and black as $\max(0, -\mathbf{A}'_i[x, y])$, thereby only increasing one of the weights (the other is zero). These local terms are combined with the global measure so that:

$$\begin{aligned} W_i[x, y] &= g_i \mathbf{A}_i[x, y] \quad + (1 - g_i) \max(0, \quad \mathbf{A}'_i[x, y]) \\ B_i[x, y] &= g_i(1 - \mathbf{A}_i[x, y]) + (1 - g_i) \max(0, -\mathbf{A}'_i[x, y]) \end{aligned} \tag{3.5}$$

where the parameter $g_i \in [0, 1]$ controls the balance between global and local features. Fig. 3.6 shows the difference between thresholding globally and locally. Beyond this, we ensure that the weights W and B are strictly positive by adding a small fixed value, which prevents pixels from being permanently bound to either white or black.

3.5.2 Geometry Contribution

Nodes in the graph are connected to their 8-neighborhood. These connections result in a tendency of neighboring pixels to be assigned to the same terminal, despite potentially varying bias towards white or black based on the terminal weights. This yields the desired large black and white regions as well as smoothness of region boundaries.

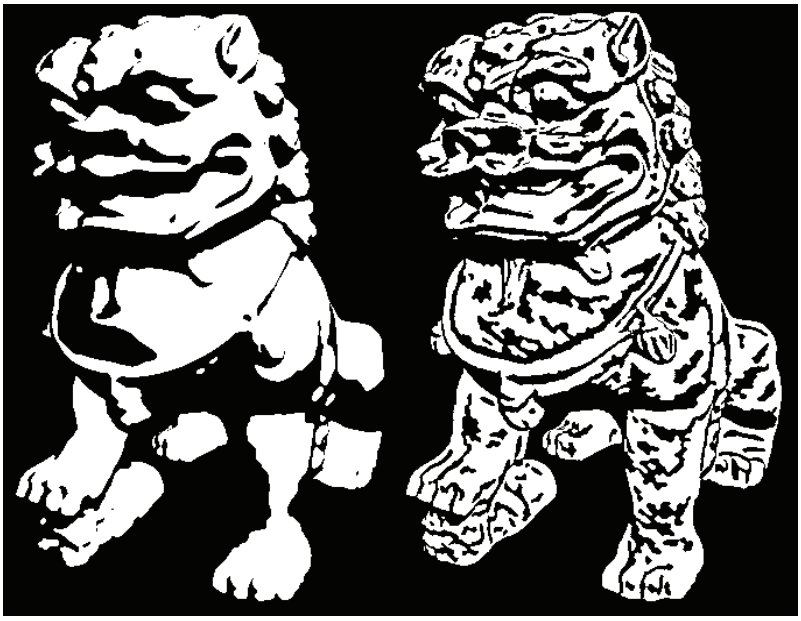
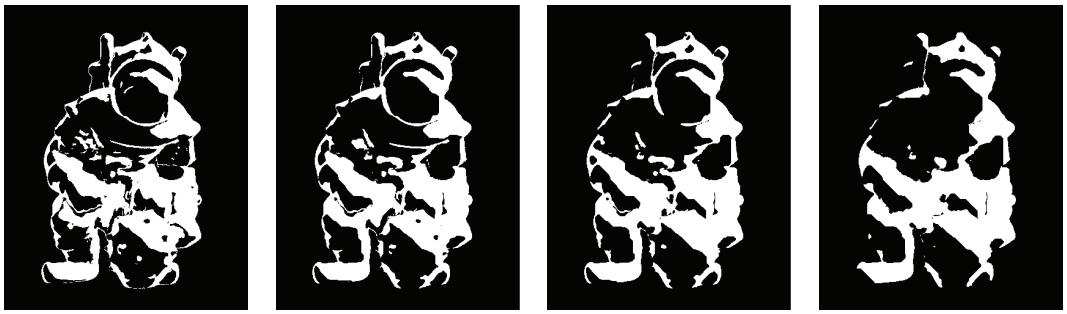


Figure 3.6: Global (left) versus local (right) thresholding.



$\tilde{N} = 0.0$ $\tilde{N} = 0.5$ $\tilde{N} = 1.0$ $\tilde{N} = 2.5$

Figure 3.7: Influence of edge weights. In these examples, the maximum neighbor weight $\tilde{N}$ is modulated. Note that small disconnected regions are successively connected when increasing weight. The left image has zero neighbor weight, and is equivalent to thresholding.

We set the maximum neighbor edge weight to $\tilde{N}$. Larger values for $\tilde{N}$ lead to smoother and fewer regions in the final result – see Fig. 3.7. This value is then modulated (i.e. multiplied with factors between zero and one) to account for naturally connected or disconnected regions in the image based on the geometry. Here we consider as geometric features the depth and normal variation (i. e., curvature) relative to screen space.

Occluding contours are important cues for shape understanding. However, the local geometry of the shape could lead to similar appearance across contours. In such situations, the coherence enforced by the minimum cut would enforce the same color for pixels across contours. Consequently, we modulate the neighbor edge weights with the factor:

$$N_0[x_0, y_0, x_1, y_1] = e^{-\frac{(z[x_0,y_0]-z[x_1,y_1])^2}{d^2}} \tag{3.6}$$

where $z[x, y]$ is the distance to the camera for pixel (x, y), and the parameter d encourages cuts to be along depth discontinuities as it approaches zero.

It is also important to convey important features on the shape. Following the works of Lee et al. [Lee+05] and Gal and Cohen-Or [GCO06], we understand features as high local variation of curvature. Consequently, we assume that coherent regions on the shape have the property that the curvature at a point is similar to an average over the curvatures in a neighborhood of the point. Such regions meet in points whose curvatures greatly differ from the average curvature.

Rather than estimating curvatures on the shape, we follow the idea of Judd et al. [Jud+07] and consider the derivatives of normals relative to screen space directions. Let $\mathbf{n}[x, y]$ be the unit vector normal gathered from the 3D scene at pixel (x, y). Then we assign a directional curvature measure to pairs of neighboring pixels $[x_0, y_0], [x_1, y_1]$ as

$$\kappa[x_0, y_0, x_1, y_1] = d_{\mathbb{S}^2}(\mathbf{n}[x_0, y_0], \mathbf{n}[x_1, y_1]) \tag{3.7}$$

where $d_{\mathbb{S}^2}$ is the geodesic distance on the Gauss sphere.

Based on this measure, we modulate the weight of edges by relating this curvature measure to the average curvature measure in the neighborhood $[\overline{x}, \overline{y}] = \left[\frac{x_0+x_1}{2}, \frac{y_0+y_1}{2}\right]$

$$\overline{\kappa}[x_0, y_0, x_1, y_1] = \sum_{[u,v]} \kappa\left[\overline{x}, \overline{y}, u, v\right] e^{-\frac{(\overline{x}-u)^2+(\overline{y}-v)^2}{c^2}} \tag{3.8}$$

where c controls the neighborhood size. This yields the edge weight modulation

$$N_1[x_0, y_0, x_1, y_1] = e^{-\frac{(\kappa[x_0,y_0,x_1,y_1]-\overline{\kappa}[x_0,y_0,x_1,y_1])^2}{k^2}} \tag{3.9}$$

where k is a user parameter to attenuate the effect of feature lines. The effect of considering curvature for attenuating edge weights is demonstrated in Fig. 3.8.

All parameters introduced in the preceding sections can be controlled in a graphical user interface with sliders. In addition, the appearance arrays are shown and updated interactively while the user is changing values. This helps in predicting the result and achieving specific effects.

3.6 User interaction

Still, controlling the color in particular regions is cumbersome and requires some understanding of the underlying system. In the spirit of interactive segmentation [BJ01; Rot+04] we provide an easier way to achieve the desired result using stroke-based user interaction: the user paints on regions and determines them to be black or white.

This input changes the weights in the graph in two ways: first, nodes corresponding to pixels that have been assigned a color by the user are disconnected from the sink if the user wants them to be white or, respectively, from the source. Second, a probability distribution is estimated from the labeled pixels and the terminal weights are modified to reflect the probability for a pixel to be white or black. Figure 3.9 shows the effect. The user can opt for determining edge weights entirely through the painting interface. This is shown on the left side in Figure 3.9. Or, painting can be used to improve an intermediate

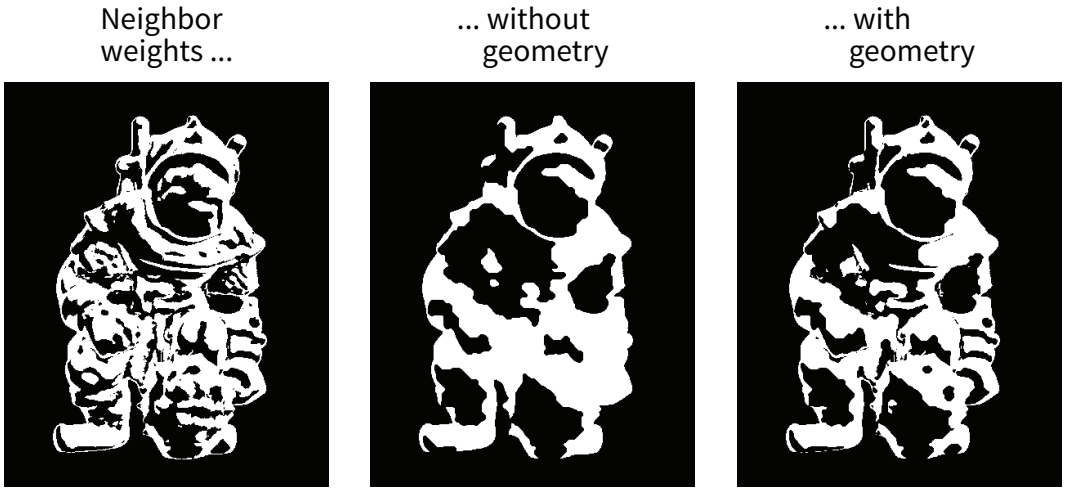


Figure 3.8: The left-most image shows a segmentation with neighbor weights equal to zero which results in mere thresholding. In the middle image the neighbor weights are set to a constant value of $\tilde{N} = 0.5$, whereas the right image has the same base $\tilde{N}$ but uses the geometry terms to modulate the neighbor weight.

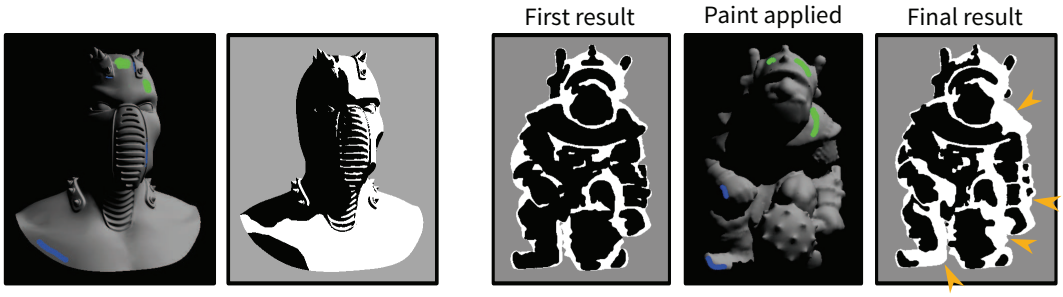


Figure 3.9: The user can define or modify edge weights by painting on the model. Marked pixel will have the user defined color (in the illustration green corresponds to white results and blue to black results). A probability distribution is learned from the user-specified pixels so that all other nodes in the graph can be assigned weights based on the user interaction. The left two images show a result generated entirely from input through the painting interface. The sequence on the right shows an intermediate result generated through adjusting the global parameters and then modifying the result through the paint interface. Note that the user input has influence on more than just the areas where the strokes are applied.

solution, by blending the old edge weights with what has been gathered from the user input (illustrated on the right in Figure 3.9).

Let W be the set of pixels to be white (the case for black pixels is treated similarly). Conceptually, we estimate probability distributions for each of the appearance and geometry attributes by filtering the samples $\{\mathcal{A}_i[x, y]\}$ and $\{\mathcal{G}_i[x, y]\}$ each with a hat kernel and then normalize appropriately. For illustration we consider $\mathcal{A}_i$. The sample functions f_i consists of Dirac delta functions moved to sample values:

$$f_i(s) = \sum_{[u,v]\in W} \Delta\left(\mathcal{A}_i[u, v] - s\right) \tag{3.10}$$

	Torus model (4096 triangles)			Caesar model (774164 triangles)		
Nodes/Pixels	2290	17493	48272	5404	37030	118173
Raytracing	0.74	1.76	2.50	1.81	2.40	3.67
Graph (fill)	0.076	0.33	0.74	0.18	0.60	1.70
Graph (cut)	0.0022	0.007	0.011	0.0067	0.019	0.056
Overall	0.82	2.10	3.25	2.00	3.02	5.43

Table 3.1: Timings (in seconds) for different numbers of triangles and filled pixels.

This function is convolved with a hat function $h(p)$, yielding a smooth function

$$\hat{f}_i(s) = h(s) \otimes f(s)_i = \sum_{[u,v] \in W} h\left(\mathcal{A}_i[u,v] - s\right), \tag{3.11}$$

which is normalized to have unit area in the interval $[0,1]$, i.e. divided by

$$A_i = \int_0^1 \hat{f}_i(t)\, dt = \int_0^1 \sum_{[u,v] \in W} h\left(\mathcal{A}_i[u,v] - t\right)\, dt. \tag{3.12}$$

yielding the probability distribution as

$$p_i(s) = A_i^{-1} \hat{f}_i(s). \tag{3.13}$$

We approximate A_i by a discrete sum and store it. Then the probability for a given value p can be quickly computed by collecting the samples that are closer to p than the support of the hat function and compute $\hat{f}_i(p)$ over this subset. While in some applications the main challenge is to estimate the width of the hat function or to adjust it locally, we simply use 0.1.

This technique yields a probability distribution function for the appearance and geometry attributes. We take negative log-likelihoods for each attribute to convert the functions into edge weight functions. Then the old edge weights are linearly combined with the edge weights derived from the user interaction. With these updates on the edge weight functions the graph cut is performed again. We like to stress, however, that all examples except the one illustrating this section are based on adjusting the parameters with sliders and make no use of painting interaction.

3.7 Performance and Comparisons

We have implemented our approach on the CPU, using the graph cut code of Boykov and Kolmogorov [BK04]. Tab. 3.1 shows rendering times for a laptop with a 1.83 GHz Core2Duo over different input mesh sizes and different viewpoints. Note that the times are good enough for interactive optimization of the results even on this platform.

The time required to fill the graph and generate the minimum cut, as well as the raytracing time, depend mostly on the number of pixels covered by the object, while being

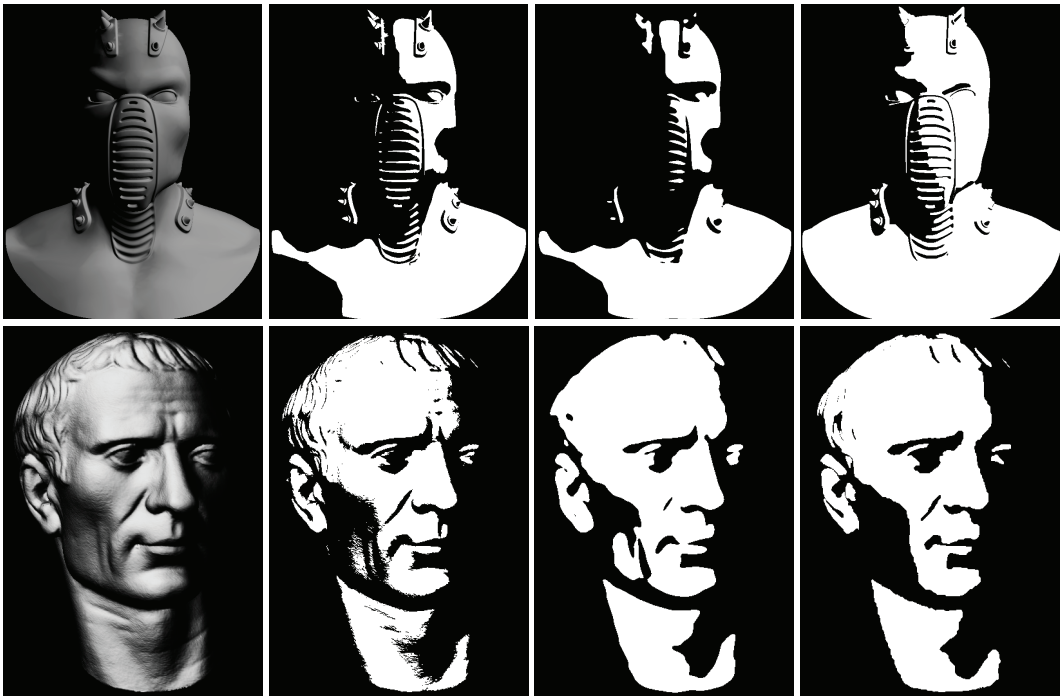


Figure 3.10: Comparison to thresholding (a) diffuse input, (b) direct uniform thresholding, (c) uniform thresholding over a Gaussian blurred input (10x10 kernel) and (d) our approach using the unblurred diffuse input for setting the terminal weights, but adding neighbor weights from geometry. This effects homogeneous regions where geometry is smooth, while still conveying features -- without searching for a compromise among Gaussian kernels.

largely independent from the geometric complexity (i. e., by using a kD-Tree). Note that we use ray-tracing as a simple way to evaluate and explore various appearance properties. When rendering performance is of higher concern, GPU rasterization may be used instead.

In the remainder of this section, we compare our technique to different families of binary image creation methods.

Uniform Image Thresholding and Local Binary Shading. Our approach to binary shading achieves better results than direct uniform image thresholding because it exploits more information about the scene. In particular, the contributions and influence of appearance components and geometric structure are taken into consideration explicitly. To visualize the difference this makes we compare thresholding a shaded image as well as a smoothed version of the shaded image. For the Ninja model, smoothing removes important geometric features and the result without smoothing looks better, while for the Caesar model the geometric details make smoothing necessary. For a fair comparison, we use the same input for setting the terminal weights in our approach (i.e. using only global information from the diffuse component). While preserving features, we can still achieve homogeneity in smooth regions, and the result automatically adapts to the necessary amount of detail. Fig. 3.10 shows the results.

(a)

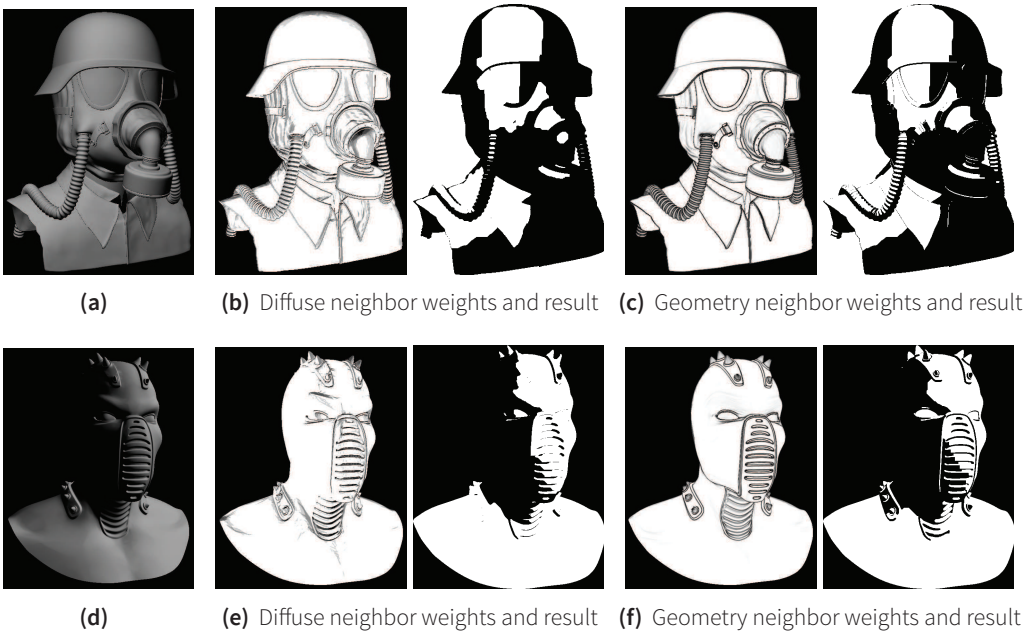
(b)

Figure 3.11: (a) Our approach. **Left and center**: Global terms only ($g_i = 1$). **Left**: Head-lamp and specular terms, with the head-lamp term giving the basic sink and source weights and the specular term only adding to the source. $\tilde{N} = 0.8, c = 0.07$. **Center**: Diffuse term combined with haloed occluding contours. $\tilde{N} = 1.1$, $c = 0.1$. **Right**: Mix between local and global diffuse term, $g_i = 0.1, \tilde{N} = 0.15, c = 0.07$. When the local term is used, the terminal weights are smaller as compared to the global term, thus $\tilde{N}$ is set lower. (b) Results obtained by Vergne et al. [Ver+08].

Real-time binary shaders are often similarly based on thresholding a mix of appearance properties (or other quantities such as normals), implemented on the GPU for performance reasons [Ver+08]. While the context is different from image thresholding, the results are necessarily the same, as the assignment of pixel color is based on thresholding local information. Our use of global optimization here means our approach can make more coherent decisions across significant parts of the image and shape, albeit at the price of not running at interactive rates. Fig. 3.11(a) shows results with clearly visible coherent decisions, such as the shading boundaries that run roughly along the ridges on the head of the chameleon – these particular results can be compared directly to the black and white results from [Ver+08] (see Fig. 3.11(b)), which lack this coherence.

Of course, these advantages come at the price of computation time, and even if our implementation can be significantly improved using rasterization (see Tab. 3.1), local binary shading methods are still methods of choice for applications such as games.

Variational Image Binarization. While the advantage over simple image thresholding might be rather obvious, it is important to point out that even global variational techniques



(a) **(b)** Diffuse neighbor weights and result **(c)** Geometry neighbor weights and result

(d) **(e)** Diffuse neighbor weights and result **(f)** Geometry neighbor weights and result

Figure 3.12: Comparison to an image based binarization approach. All results are generated with the same input for terminal weights, namely only global information from a diffuse shading, seen in (a) and (d). For the image based approach ((b) and (e)), neighbor weights are derived from the diffuse shading only (therefore being similar to what can be achieved with image-space only approaches like Mould and Grant [MG08]), while we use geometric information in our approach ((c) and (f)). The neighbor weights are visualized on the left side of each pair as the normalized sum of all neighbor weights of each node (black meaning no edge weight at the corresponding pixel) while the segmentation result is displayed on the right side of each pair.

for thresholding are less convincing if they are based on a single shaded depiction of the object. The reason is, again, that geometric structures might be lost in the shaded image, and no thresholding technique could ever revive them.

We have used our approach to implement an artistic image thresholding approach similar to the one of Mould and Grant [MG08] (which is also based on graph cut). In particular, we derive neighbor weights from the input images by comparing neighboring pixels (for details see [MG08]). We apply this thresholding implementation to a diffuse rendering and compare to the result of our approach, which has been similarly limited to use only the diffuse channel as an appearance attribute. Thus, both approaches start from the same appearance information, but only our technique uses geometric information for the modulation of neighbor weights. Fig. 3.12 shows the effect of the additional information, emphasizing surface structures that are hard to derive from the diffuse shading.

On the other hand, using "invisible information" suggests our binary images differ more from realistic shading. One might argue that the restriction to only two levels makes it harder to convey both appearance and structure, and in our approach we specifically opt for conveying structure rather than appearance.

Line Drawing Finally, it is interesting to compare the results of binary shading to line drawings (see Fig. 3.14). Line drawings implicitly define the regions that we try to assign

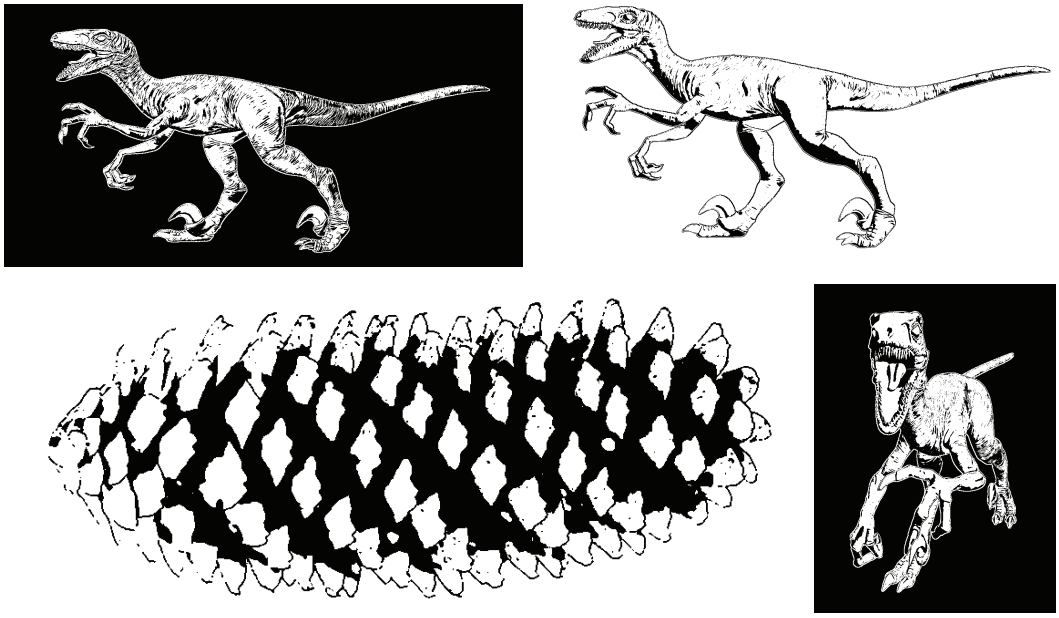


Figure 3.13: Top and bottom-right: Results using only the local diffuse term ($g_i = 0$ for diffuse with neighborhood size radius of 4 pixels) combined with haloed occluding contours. The base neighbor weight is $\tilde{N} = 0.2$ and modulated using our curvature geometry term. The respective neighborhood size control is $c = 0.2$ for the images on the top and $c = 0.1$ for the image on the bottom-left. **Left-bottom**: Our method applied to RGBN data [TF+06]; in this case, contours are drawn on top using the provided discontinuity map, after median filtering and thresholding.

Figure 3.14: Depictions of a golf ball: line art rendering using suggestive contours [DeC+03] as well as a toon shader combined with suggestive contours and suggestive highlights [DR07], and our black and white results.

constant colors; conversely, binary shading implicitly defines feature lines as boundaries between regions. In this sense, the two approaches are dual.

However, concavities can possibly be communicated more clearly using larger black regions (as can be seen in the golf ball example in Fig. 3.14). A combination of toon shading and line art rendering [DR07] that includes suggestive contours and suggestive highlights produces results that bear some similarity to our style.

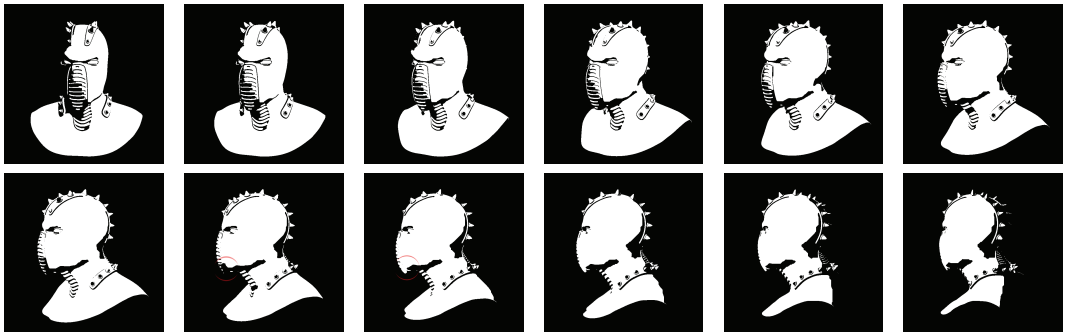


Figure 3.15: Binary shading applied to animation, showing every other frame of a sequence. Top row: Regions appear and disappear progressively, and coherently. Bottom row: failure case with tangential regions (circled in red), where the incoherence is easily perceptible.

As is obvious from the dual relationship, the fundamental idea of optimizing the result in image space could also be applied to line art rendering, potentially helping to connect otherwise disconnected feature lines or help to fill empty spaces – something artists report doing [Col+08]. Similarly, global optimization could help combining different types of lines [DR07].

3.8 Temporal Coherence

We have also conducted experiments on the degree of temporal coherence achieved by our technique, when applied to each frame separately. The notion of temporal coherence for styles such as binary shading is hard to define, since jumps must happen on occasion – fading in and out is not possible with just black and white. In typical situations, we found that, apart from visibility changes, large regions do not drastically change from one frame to the other. Thus, the algorithm often gives acceptable results when performed on a per-frame basis – see Fig. 3.15(top). This comes from the fact that the energy implicitly defined on the graph is strongly attached to object space quantities (from geometry and appearance), while the particular screen-space local connectivity has a minor impact on the graph flow and perceived region boundaries. However, this general observation fails on regions which, during the animation, quickly become viewed edge-on. In such cases, the screen-space projection effects significant differences from frame-to-frame. This appears as a sudden large change between black and white – see Fig. 3.15(bottom). Inspired by work such as [Wan+04], we seek to improve temporal coherence through the analysis of a spatio-temporal volume of images, where each image in the stack represents a particular moment in time. In our case, the graph can be constructed to correspond to this volume. We experimented with different approaches for the weights and the connectivity of the nodes between the frames, but none delivered more coherent but still visually satisfactory results. Temporal coherence of this style presents a significant challenge and is left for future work.

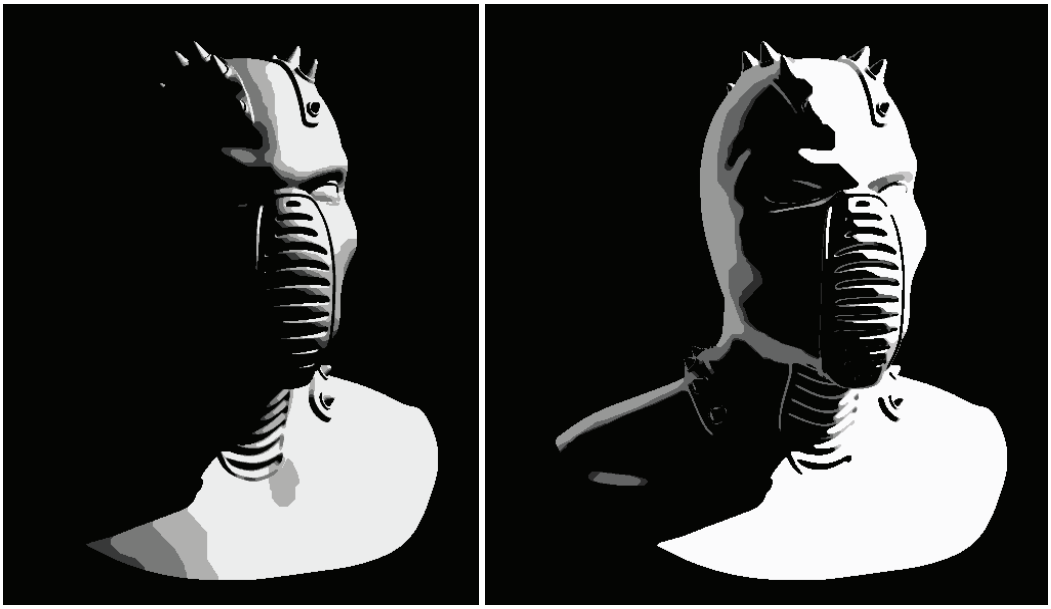


Figure 3.16: k-colors. Binary shading extended to multiple colors could provide better surface property cues and more possibilities for expressive renderings and stylization. To give an idea of a possible style, a very simple approach of overlaying several renderings with changing parameters. **Left**: Constant diffuse weight with different specular source weights. **Right**: Constant diffuse weight with different head-lamp source weights.

3.9 Extended Features and Multi-Coloring

We think that the idea of using different appearance attributes could be exploited further: mirror or glossy reflections would aid the depiction of plastic or metallic surfaces, a style often used in Manga drawings. More appearance and geometry channels might also enable or improve the success of learning attribute statistics from user data.

We have focused on generating binary images. Clearly, extending the approach towards more colors (see Fig. 3.16), a set of dithering or hatching styles, or other sets of attributes is possible.

Our approach to binary shading offers a flexible framework to control the effects of appearance and geometry on the shape and placement of black and white regions in a binary rendering. The geometry term controls how regions partition the surface, with boundaries that often run along surface features. Changing the underlying rendering procedures allows for different styles.

User interaction might help in reaching visually pleasing results with less effort. We have implemented a user interface based on learning the component statistics of regions that the user suggests to be white or black (similar in spirit and design to Rother et al. [Rot+04]). However, we find that the control of style is comparable to adjusting the parameters directly using sliders – at least for us. As we lack conclusive information on the usability of this interface for other users, we leave this for future work. A similar strategy based on deriving probability distributions could be used for learning a model from a large collection of artist data. However, no such data set is available at present.

The main idea is to render appearance attributes into channels and to compute the screen space properties from the different attributes, rather than combining the attributes before rendering. This basic idea clearly has applications in a variety of rendering techniques.

In this technique, the aim was the creation of stylized binary images with larges regions. While the results are expressive, the reduced palette also limits the amount of surface features that can be conveyed. But large regions of black and white are only one possbility for reducing the palette, line drawings are another. As we saw before (Sec. 2.1), line drawings are also capable of conveying shape but equally lack surface property definition. However, when texturing the lines expressively, i. e., implying surface properties like roughness or even larger features not existing in the geometry itself, line drawings can be made expressive in that direction. In the next chapter, we present our work of parameterizing animated lines in a temporally coherent way to allow this kind of expressive texturing.

CHAPTER 4

SPATIO-TEMPORAL ANALYSIS FOR PARAMETERIZING ANIMATED LINES

Similar to the previously presented « Binary Shading », line drawing is a strongly reduced and stylized form of depiction. In this chapter, we present a new way of parameterizing animated lines for texturing in a temporally coherent way by relating the input lines and the final rendering with an intermediary spatio-temporal analysis of the line development.

4.1 Temporal Coherence of Animated Lines

Line drawing is a popular rendering style commonly used for mechanical illustrations, cartoons, and sketches and, in many cases, these are derived from three-dimensional models. For instance, many 3D CAD software applications offer line drawing as one of the stylized rendering options. In line drawing, shapes are represented by a few carefully selected lines such as silhouettes [HZ00], suggestive contours [DeC+03], or apparent ridges and valleys [Jud+07]. The simplest approach is to render lines as continuous and textureless, as if one used a ball-point pen with even pressure. But often, the lines are textured to offer greater expressiveness – for instance simulating brush strokes, dashed lines, or calligraphic curves. In this work, we are interested in parameterizing the lines such that such textures can be applied to them.

A typical work session starts by creating a 3D animation. While many research studies still work on improving this step, we use existing tools such as Maya and Blender and assume that a 3D animation is available. The next step is to select the lines to be drawn. In this work, we focus on the case where these lines are silhouettes. We describe a procedure to track silhouettes across frames in Sec. 4.5. Without this grouping, each line would be parameterized independently of the others, which produces "sliding" artifacts, e. g., brush strokes look "disconnected from the underlying 3D model". Once the lines are grouped through time, we compute a parameterization over these lines. This is the main focus of our work, it is described in Sec. 4.6. Our goal is to define how the texture is mapped onto a line. This parameterization is directly responsible for the stretch and compression of the texture and for its motion. As such, it plays a critical role in the look of the rendered animation. Once we have parameterized the lines, the last step is to render the animation, for which we use standard techniques.

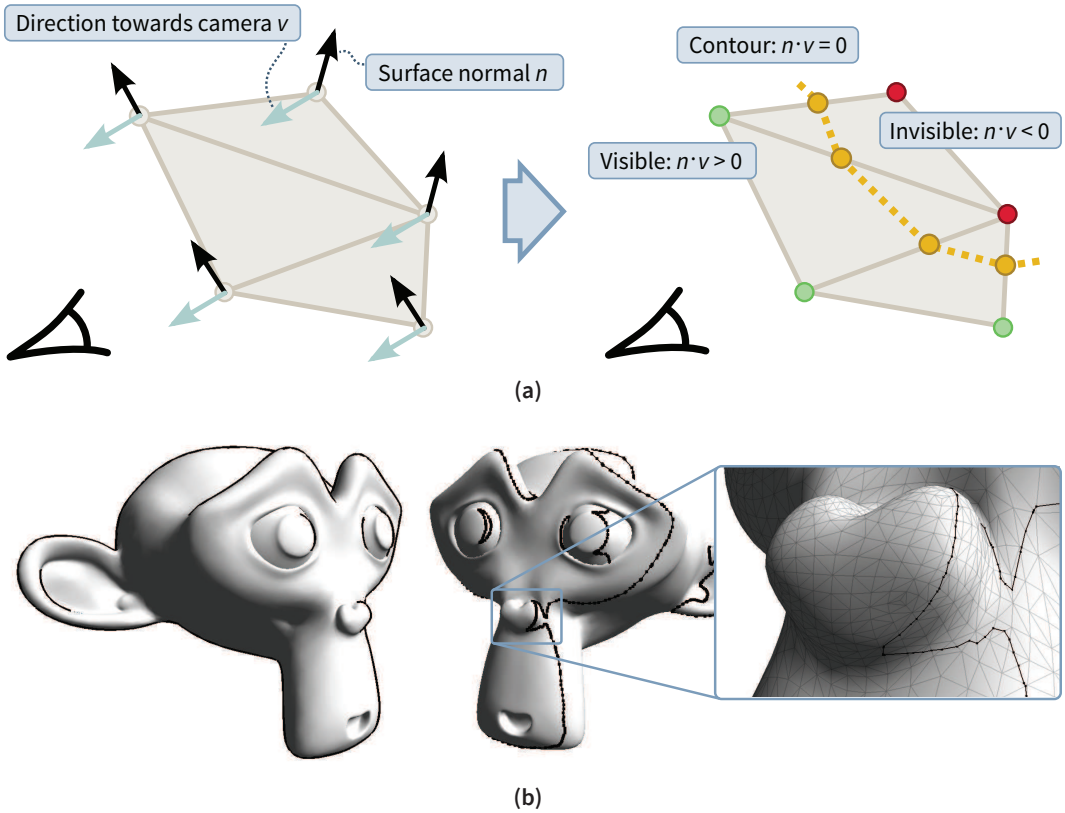


Figure 4.1: Contours. **(a)** The lines used in our examples are standard contours, i. e., the points on the surface at which the view direction and the surface normal are perpendicular. On a triangle mesh, these are found on segments where the orientation changes, i. e., where one vertex has a positive and the other a negative dot product between surface normal and view direction. The point on the segment is found by linearly interpolating the difference of the orientation. **(b)** Example of a contour of a triangle mesh (hidden lines removed) and the same contour seen from a different viewpoint (with hidden lines shown). On closed meshes, contours are always loops on the surface.

The difficulty of parameterizing animated lines comes from the fact that they are extracted from a three-dimensional model whereas the final drawing lives in the 2D image plane. We seek a temporally consistent parameterization such that the drawing looks like it follows the actual motion of the scene. However, there is no general solution to this problem. For instance, if one constrains the 2D lines to exactly follow the 3D model, effects like foreshortening may lead to unsightly texture distortion and brush stroke may become overly compressed or stretched. On the other hand, avoiding distortions can lead to significant motion inconsistencies. Establishing temporal coherence of the lines (or of the points that represent the lines) is the prerequisite for our parameterization (see Sec. 4.2). In our approach, we formulate temporal coherence as a least-squares optimization problem where each constraint is weighted by parameters that let users control the look of the final output.

Another major difficulty comes from lines that merge and split, which introduces topological discontinuities in the drawing. Ignoring these events yields unpleasing “popping”

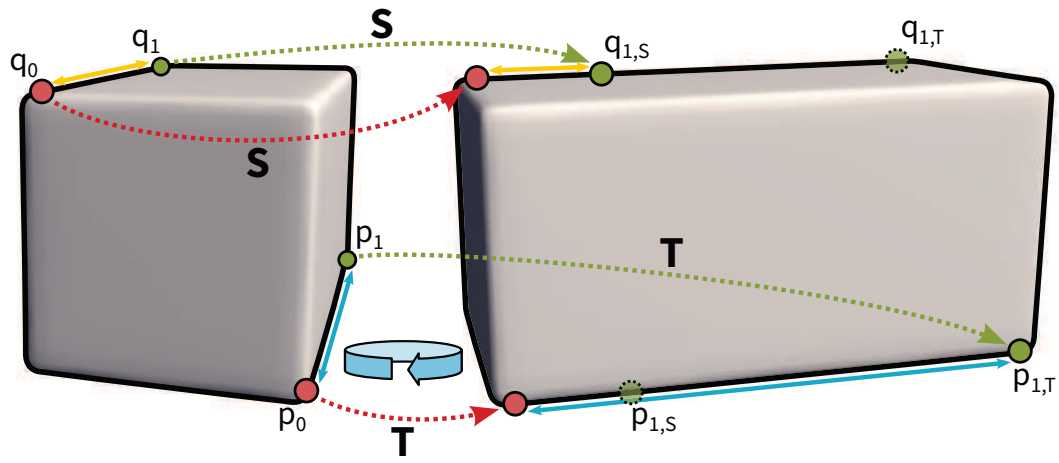


Figure 4.2: Temporal Coherence. When dealing with animated scenes and objects, knowing the correspondence between frames is necessary to establish temporal coherence. While change in object space can come from object movement and deformation, in screen space it can also be caused by camera movement. In this example of a rotating elongated cube, the lines p and q on the left have different corresponding lines in screen space (**S**) and world space (**T**). While p is short in screen space on the left due to foreshortening, it becomes longer on the right. Similar q, which is long in world space on the left becomes shorter in world space on the right. Both spaces can be used to establish coherence of the lines, in this work, a user-controlled trade-off between both is used.

artifacts, e. g., a brush strokes suddenly becomes two strokes. Kalnins et al. [Kal+03] have shown that such discontinuities can be handled by extending them in time, e. g., by always drawing two strokes even if, in some frames, it is a single connected line. Because Kalnins' approach works in real time, this strategy is only partially effective. It prevents popping due to disocclusion but not artifacts due to occlusions. Further, it also tends to over-segment lines, creating a large number of short strokes. We will discuss this difficulty in more detail (Sec. 4.7). Unlike Kalnins and colleagues, we assume that the whole animation is known from the start. Although this choice makes our method inapplicable in a game context, it makes it more suitable for feature animations where quality is paramount. In this respect, our approach is complementary to previous work.

Our approach works in the space-time domain that is the product of the (x, y) image space with the time axis t. For each line to be drawn, we consider the space-time set of 1D lines spanned by the line throughout the animation, which we name $\ell(t)$. It can be seen as a 2D surface in the (x, y, t) domain and we cast the problem of parameterizing $\ell(t)$ as the parameterization of this 2D space-time surface. This is the central idea of our contribution. This construction allows us to account for the 3D motion of the input model and the distortion of the 2D line texture, and we can ensure the complete coverage of the line. Further, we handle splitting and merging events with a graph embedded in the line space-time surface. Our approach enables the reuse of the line discontinuities for several events, thereby drastically reducing the number of actual strokes needed in a given animation.

4.2 Related Work

4.2.1 Line Drawing

A wealth of articles deal with rendering line drawings from 3D models. Several methods have been proposed to decide where to draw lines on a given 3D model [HZ00; DeC+03; Jud+07; Col+08; Gra+10] (see Sec. 2.2 for details). Whereas most of these studies focus on static scenes, DeCarlo et al. [DeC+04] describe a technique specific to animation. In this work, we focus on line parameterization and illustrate our approach on silhouette lines. We will also discuss how it can be extended to other lines.

A few methods specifically target the rendering of animations as line drawings. However, most of them produce uniform lines with no texture [Lee+07; Win+06].

4.2.2 Temporally Coherent Stylization

The most related work to ours is by Kalnins et al. [Kal+02; Kal+03] who also focus on temporally consistent textured lines. We share their goal of adding texture onto lines which greatly improves the expressiveness of the drawing. For instance, texture can be used to represent dotted lines, brush strokes, or even surface details such as cactus needles. However, Kalnins et al. demonstrate that a naive parameterization yields unacceptable results with texture sliding on the object surface. They address this problem in the context of real-time interactive simulation, where only the next frame is known at a point. The coherence is established using by using samples on the lines that propagate their parameterization value and their line ID from one frame to the next. For this, a sample on a line in frame f_i searches for the closest line in the next frame f_{i+1} which is usually very close as the silhouettes move coherently over the surface. If that is not the case, then it usually means that a camera movement is so strong that it breaks this coherence. They reason that in this case, any kind of coherence in the parameterization couldn't be perceived anyway. This process usually lets most samples end up on the line that corresponds to the same line from the previous frame.

To handle cases where samples find lines that do not correspond to those on which they were before, a grouped voting scheme is applied. If more than one group remains on a line, the situation can be resolved using one of several policies. Keeping the line in one piece, the line could become a new one mixing all voters, or the line becomes that of the majority of voters. Both policies exhibit problems with temporal coherence. The other possibility is to split up the lines where groups meet, which will keep temporal coherence but can, over time, lead to a segmentation of the lines.

In a final step, the parameterization is then created using the votes on the lines. In order to find a balance between screen-space uniform and world-space coherence, these two goals are formulated as an energy which is minimized in an optimization step providing the final parameter values.

Although they demonstrate convincing results in a number of cases, this partial knowledge of the animation limits their ability to adapt to topological events. For instance, they cannot deal with appearing discontinuities because these events are unknown until they occur [Kal04, § 4.5]. This kind of "unexpected event" is inherent to any real-time scenario. In this work, we explore a different, complementary case in which the entire animation

is known in advance. In this context, our approach can "anticipate" appearing discontinuities and adapt to them. In particular, our parameterization does not produce popping artifacts when an occlusion occurs and requires fewer cuts along lines, thereby rendering longer and more visually pleasing strokes.

Depending on the parameterization, the line texture may be compressed or stretched producing unpleasant renderings. Bénard et al. [Bén+10] address this issue by generating a multi-scale texture. The proposed parameterization focuses on the same real-time scenario as Kalnins et al. [Kal+03] and shares the same limitations inherent in this setup.

In [Bén+12], Bénard et al. use Active Contours to generate an underlying parameterization of line pieces which is in turn used to achieve temporally coherent texturing. Unlike the method presented here and for the same reasons mentioned before, Bénard et al. do not provide a global temporally coherent parameterization

Depending on the view distance, lines may become too close or even start overlapping, altering shape perception. Shesh and Chen [SC08] solve this problem by defining a line hierarchy and replacing multiple overlapping strokes by a single "average" line.

Temporal consistency has also been studied for the texture that represents the interior of the models and the canvas on which the illustration is drawn (e. g., [Cun+03; Kim+08; Bén+09]). However, the 1D nature of lines raises specific challenges, as the topology of 1D curves significantly differs from the topology of a 2D canvas.

4.3 Contributions

In this work, we introduce the following contributions:

Space-time formulation We formulate the parameterization of a animated line as the parameterization of the space-time surface that it swept during the animation (see Fig. 4.3).

Least-squares optimization of geometric constraints We express our objectives as a series of geometric constraints. We represent each of them by a least-squares energy term and the trade-off between the constraints is controlled by a small set of meaningful parameters.

Discontinuity reuse We show that line discontinuities can be reused to limit the number of cuts made to the lines. We control the trade-off between cut reuse and temporal coherence with a simple parameter for the maximum allowed sliding.

4.4 Overview

Input Our model takes as input an animated 3D model $\mathcal{M}(t)$ where t is the time variable defined over an interval $\mathbb{T}$, and a camera that is represented by the function π that projects 3D points onto the image plane. We assume that $\mathcal{M}$ has a temporally consistent parameterization, that is, for any point $\mathbf{P}$ of the model, we can compute its trajectory $\mathbf{P}(t)$ during the animation. In practice, these conditions are always satisfied when the data come from a 3D modeler in which a base mesh is deformed while its topology is preserved. The mesh vertices and faces are naturally linked through time, which implicitly ensures a temporally consistent parameterization of the 3D model. We also propose a heuristic

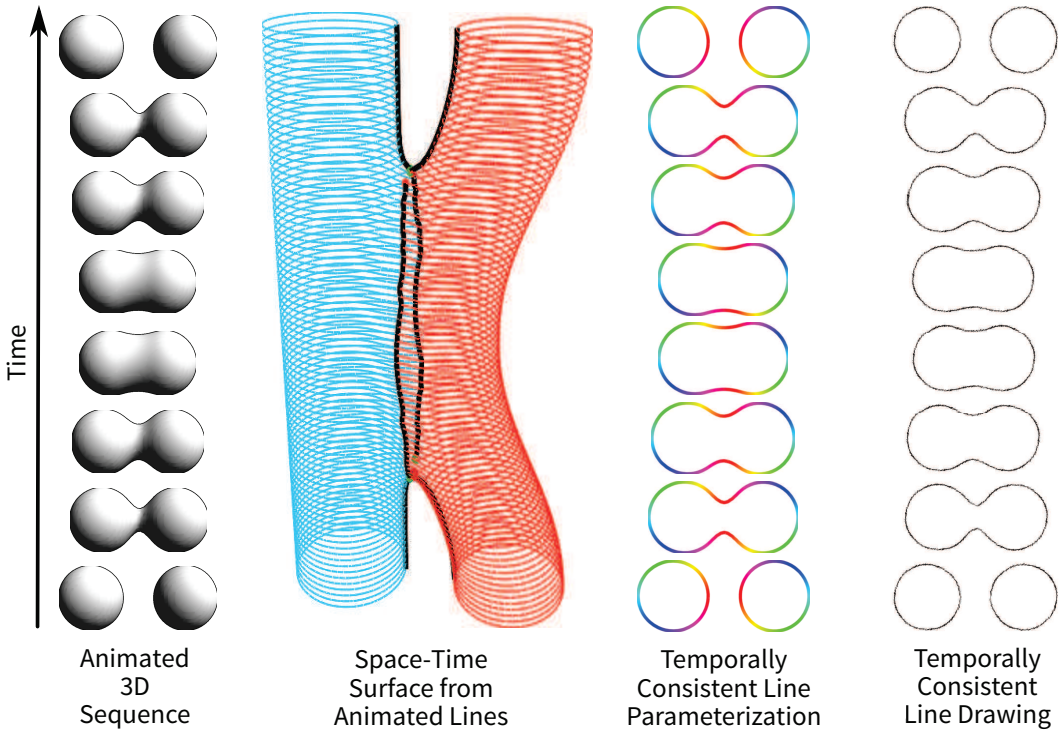


Figure 4.3: Principle: a temporally consistent parameterization of a line drawing is computed by parameterizing the space time surface it defines over time and used for texturing.

to cope with simple models that do not have such parameterization, e. g., meta-balls. We also assume that the camera is given as input, that is, we have a projection function π that maps 3D points onto the image space.

Objective We seek to parameterize the 2D lines that correspond to the silhouettes of $\mathcal{M}(t)$. We process the lines one by one. We name $\mathcal{L}(t)$ a 3D line at the surface of $\mathcal{M}$ at time t. As t changes, $\mathcal{L}$ may move on $\mathcal{M}$. We name ℓ the 2D projection of $\mathcal{L}$, that is, $\ell = \pi(\mathcal{L})$. In this work, we seek a parameterization of ℓ that is temporally consistent. Formally, we aim to define a function $f(u,t)$ such that at any time $t \in \mathbb{T}$, f is a continuous one-to-one mapping between the parameter space, e. g., $u \in [0;1]$, and $\ell(t)$.

Strategy We formulate the parameterization of a line ℓ as the parameterization of the space-time surface $\mathcal{S}$ swept by ℓ during the animation, that is, $\mathcal{S} = \bigcup_t \ell(t)$. First, we construct $\mathcal{S}$ from the lines extracted at each frame. Then, we expose how we deal with temporal coherence when the topology of the lines does not change over time. We formulate desired properties such as temporal coherence, lack of distortion, and line coverage in geometric terms expressed on $\mathcal{S}$. Then, we translate this problem into a least-squares optimization that can be solved with a sparse linear system. In a second part, we cope with splitting and merging events of these lines. To avoid popping, we cut lines and propagate the resulting discontinuities. We show how to use the same cut for several discontinu-

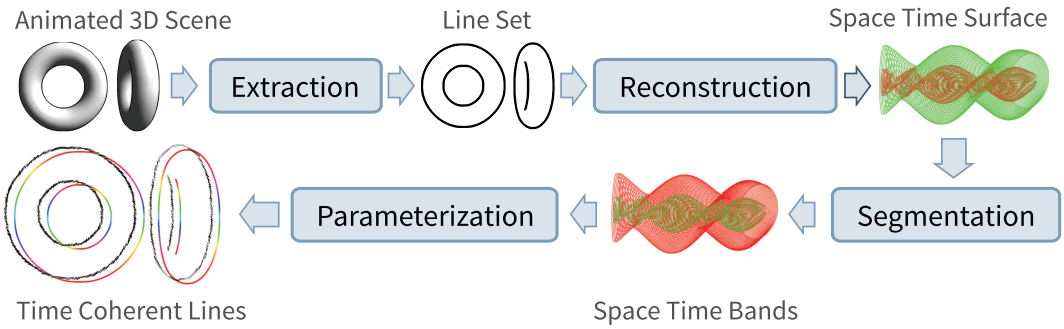


Figure 4.4: Overview: lines are extracted independently for each frame of the animation, before being grouped as plausible corresponding lines from frame to frame. Lines may split or merge during the animation and create a graph which is subsequently decomposed in bands corresponding to single lines over time. These bands, or space-time lines, are finally parameterized independently under user control to provide time-coherent parametric lines.

ities to avoid over-segmenting the lines. With our space-time formulation, the cuts are geodesic lines on $\mathcal{S}$ and our handling of discontinuities corresponds to decomposing $\mathcal{S}$ into charts with a disc topology. Finally, we present and discuss representative results of our approach.

4.5 Building the Space-Time Surface

In this section, we expose how to build the space-time surface $\mathcal{S}$ generated by a line ℓ during the animation. First we describe a technique that we use on simple examples. Then we expose a robust method that copes with complex topological changes and more realistic models in the case where the lines can be defined as the level set of scalar functions defined on the mesh, as is the case for silhouettes.

4.5.1 Simple Construction

For educational examples such as Fig. 4.3, we follow a simple approach to build the space-time surface $\mathcal{S}$. We consider the lines extracted in two consecutive frames and use a voting scheme to decide which lines should be paired, i. e., which lines are actually adjacent on $\mathcal{S}$. Each vertex of each line casts a vote for the nearest vertex in the other frame according to the image space distance. For a given line, we observe the votes cast by its vertices and link it to the line in the other frame that received the most votes. This process creates a graph where lines extracted at each frame are the nodes and where the arcs indicate how to build the surface. Topology changes, i. e., when lines split or merge, are detected when a line is linked to more than one line in an adjacent frame.

The advantage of this approach is that it only assumes that we can extract lines at each frame. For instance, it can deal with an animated 3D model inconsistently meshed from frame to frame, which allows for processing meta-balls as shown in Fig. 4.3. Moreover it can handle any kind of line drawing. However, it also relies on the fact that the image distance represents well the evolution of the lines on the 3D model which may not always

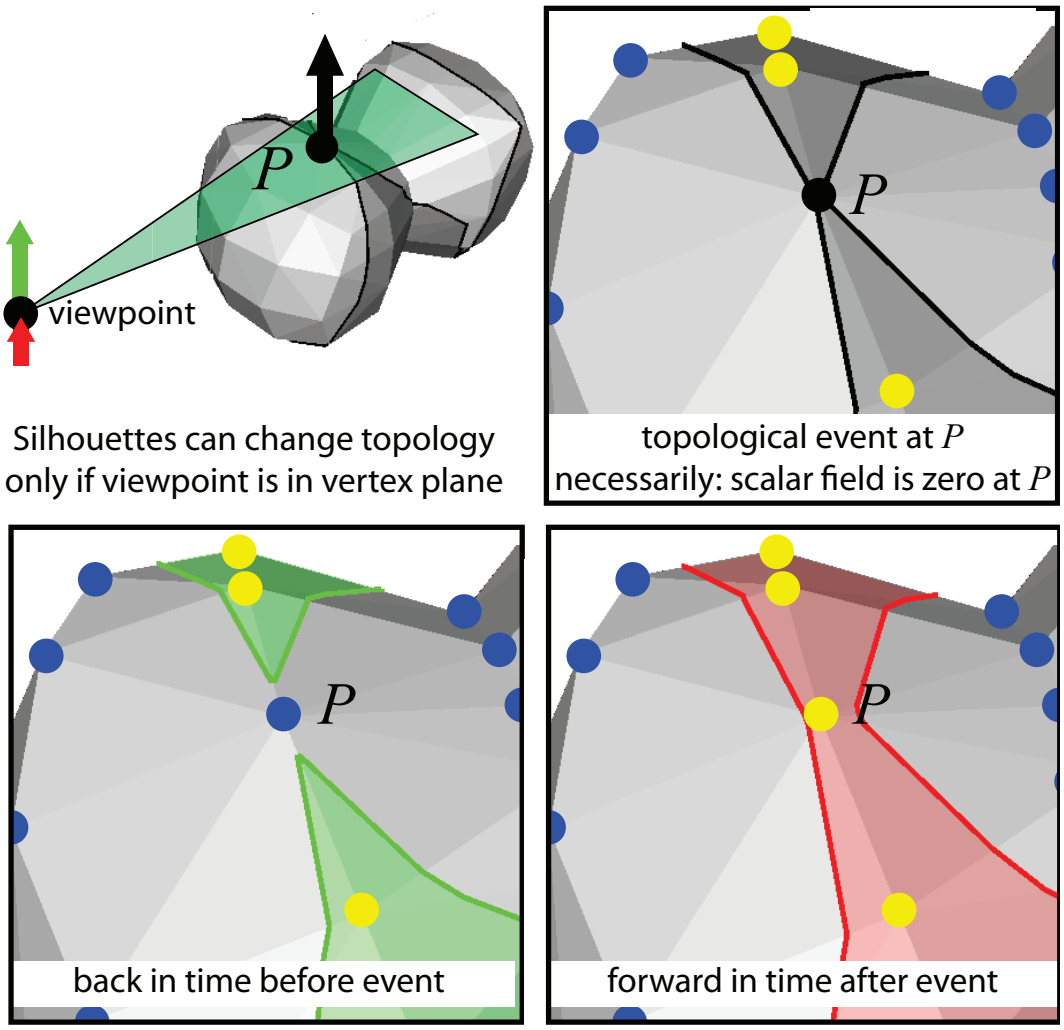


Figure 4.5: Robust topology change detection for lines which can be modeled as the zero level sets of a scalar function (e. g., silhouettes).

be true for more complex models. We describe a robust approach that handles these cases in the following section.

4.5.2 Robust Construction

We assume that the input model $\mathcal{M}$ is a triangular mesh. If it is not, we convert it before processing it. The silhouette of $\mathcal{M}$ is made of one or several closed loops. First, we characterize when topological changes occur, i. e., when loops split or merge. Silhouettes are characterized by a zero dot product between the mesh normal $\mathbf{n}$ and the view direction $\mathbf{V}$. We estimate a normal $\mathbf{n}$ at each vertex and linearly interpolate it over the faces. With this scheme, at most one silhouette line can cross a triangular face: if $\mathbf{n} \cdot \mathbf{V}$ has the same sign for all three vertices, there is no silhouette, and if the sign changes, one silhouette line crosses the two edges with different signs. Since a topological event corresponds to two or more

lines being in contact, this cannot happen inside a face and must occur at a vertex. Thus, we only need to examine vertices and time instants where $\mathbf{n} \cdot \mathbf{V} = 0$. Assuming that the camera and mesh move linearly between frames, this amounts to finding the zero-crossing of a linear function. For each possible candidate, we check whether there is more than one line going through the vertex. If that is the case, we mark the vertex and time instant as a topological event (Fig. 4.5).

Once we have listed all the events, we build the space-time surface $\mathcal{S}$ by considering what happens between two frames at t and $t + \Delta t$. There are two cases. If there is no topological event, the silhouette loops have moved over the mesh without splitting or merging. In this case, we label the mesh with the sign of $\mathbf{n} \cdot \mathbf{V}$ at t and its sign at $t + \Delta t$. Mesh regions swept by the silhouette between the two frames have opposite signs, and since there is no topological events and the camera moves along a segment, these regions are disconnected. In particular, the linear camera movement ensures that a vertex can change its sign at most once. Using these properties, we build $\mathcal{S}$ by pairing lines at t and $t + \Delta t$ that are linked by a mesh region with opposite $\mathbf{n} \cdot \mathbf{V}$ signs. In the other case when there are one or more events between the two frames, we split the time interval so that a single event happens at time t_0 in each interval. We extract the loops at $t_0 - \epsilon$ just before the event, and at $t_0 + \epsilon$ just after it (Fig. 4.5). The $t_0 - \epsilon$ lines can be linked to the lines in the earlier frames using the no-event case. The same applies to the $t_0 + \epsilon$ lines and the later frames. We also link the $t_0 - \epsilon$ lines to the $t_0 + \epsilon$ lines to reflect the change of topology. If we split the interval between two frames to isolate events, we concatenate the information of all sub-intervals and only represent the links between the lines at t and the lines at $t + \Delta t$. Fig. 4.9, 4.10, 4.11 and 4.12 show examples of our space-time surface reconstructed by our approach.

Discussion Our robust reconstruction relies on the fact that lines are a zero level set of a scalar function defined on the mesh $\mathcal{M}$. In the case of silhouettes, the function is $\mathbf{n} \cdot \mathbf{V}$. While not all lines can be expressed as a zero level set, several others lines fall in this category [Str+08]. This work focuses on silhouette parameterization, and we believe that studying other types of lines is a natural extension for our work in the future. In particular, the above ideas can be applied directly to albedo and specular curves. Currently, our implementation supports rigid motions only but we do not expect any major difficulty to extend it to non-rigid transformations.

4.6 Parameterizing a Line Over Time

In this section, we explain how to find a temporally consistent parameterization of a line ℓ. For now, we assume a single open 1-manifold line evolving over time with no topological events. The case of multiple lines and topological events is discussed in the next section. We first discuss the parameterization in geometric terms before translating it into a discrete optimization problem.

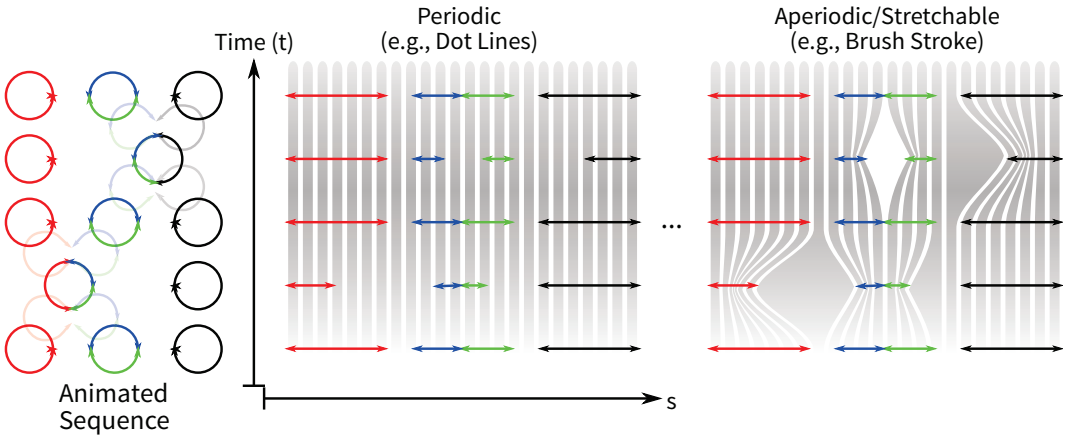


Figure 4.6: Space-time parameterization of a line set: our variational formulation offers intuitive control w.r.t. to the texture function type (stretchable or periodic).

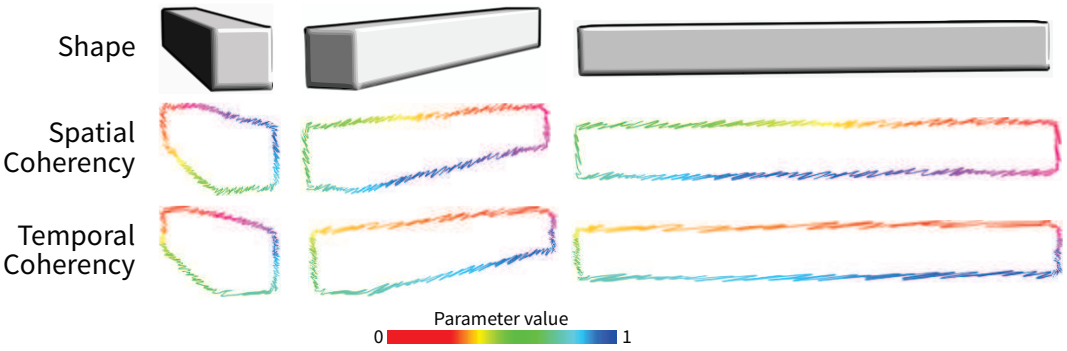


Figure 4.7: From spatial to temporal coherence control.

4.6.1 Geometric Formulation

Lines as Time Slices During the animation, a single open line ℓ sweeps a space-time surface $\mathcal{S}$ that has a disc topology. We seek a uv parameterization of $\mathcal{S}$, that is, a function $f(u, v)$ such that $f(\mathbb{U}, \mathbb{V}) = \mathcal{S}$ with $\mathbb{U}$ and $\mathbb{V}$ the spaces on which u and v are defined. Because our goal is to parameterize the lines $\ell(t)$ which are "slices" of $\mathcal{S}$ along planes orthogonal to the t axis, we impose that the v parameter is the time variable t. That is, we seek a function $f(u, t)$ such that for a given $t_0 \in \mathbb{T}$, $f(\mathbb{U}, t_0) = \ell(t_0)$.

Temporal Coherence To ensure the coherence between the 3D motion and the 2D drawing, the trajectory of a point on the line should match the trajectory of its corresponding 3D point. This implies that the speed of the 3D model projected in the image plane should be equal to the speed of the line. Formally, at a given time t_0, we seek:

$$\frac{\mathrm{d}}{\mathrm{d}t}\pi\big(\mathbf{P}(t)\big) = \frac{\mathrm{d}}{\mathrm{d}t} f\,(u_0, t) \tag{4.1}$$

where u_0 is the parameter of the projection of $\mathbf{P}(t_0)$ on $\mathcal{S}$ at t_0, i. e., $f(u_0, t_0) = \pi(\mathbf{P}(t_0))$.

Coverage and Distortion We consider two practical cases, *aperiodic* textures such as brush strokes, that stretch to cover the line, and *periodic* patterns such as dotted lines that repeat to ensure coverage (see Fig. 4.6).

▷ For aperiodic textures, we seek a parameterization function f that maps the $[0;1]$ interval onto ℓ. That is, at every time t, we want $f([0;1],t) = \ell(t)$. Since f is continuous, it is sufficient to consider $\mathbf{p}_1$ and $\mathbf{p}_2$, the end points of ℓ.

$$\{\mathbf{p}_1(t), \mathbf{p}_2(t)\} = \{f(0,t), f(1,t)\} \tag{4.2}$$

To ensure this coverage, the texture has to be stretched or compressed. Large variations in this stretching/compression yields unsightly results and we would like this distortion to be uniform along the line, which means:

$$\left\| \frac{\mathrm{d}}{\mathrm{d}u} f(u,t) \right\| = \text{length}\big(\ell(t)\big) \tag{4.3}$$

▷ With periodic patterns such as dotted lines, coverage is not an issue since we can repeat the texture as much as needed. Because of this property, we do not need to stretch or compress the texture as in the aperiodic case and we seek to preserve the original aspect of the texture. We name a the length of the pattern, and to prevent distortion, we seek:

$$\left\| \frac{\mathrm{d}}{\mathrm{d}u} f(u,t) \right\| = a \tag{4.4}$$

4.6.2 A Least-Squares Approach

In general, the constraints described above cannot be satisfied simultaneously. We use a least-squares approach to find a trade-off. For aperiodic textures, this corresponds to:

$$\begin{aligned} \arg\min_f \quad & w_{\text{dis}} \left[\left\| \frac{\mathrm{d}f}{\mathrm{d}u} \right\| - \text{length}(\ell) \right]^2 \\ & + w_{\text{end}} \Big[\big(\mathbf{p}_1(t) - f(0,t)\big)^2 + \big(\mathbf{p}_2(t) - f(1,t)\big)^2 \Big] \\ & + w_{\text{proj}} \left[\frac{\mathrm{d}}{\mathrm{d}t} \pi\big(\mathbf{P}(t)\big) - \frac{\mathrm{d}}{\mathrm{d}t} f(u_0,t) \right]^2 \end{aligned} \tag{4.5}$$

where w_{proj}, w_{dis}, w_{end} control the relative importance of each constraint. With periodic patterns, this becomes:

$$\begin{aligned} \arg\min_f \quad & w_{\text{dis}} \left[\left\| \frac{\mathrm{d}f}{\mathrm{d}u} \right\| - a \right]^2 \\ & + w_{\text{proj}} \left[\frac{\mathrm{d}}{\mathrm{d}t} \pi\big(\mathbf{P}(t)\big) - \frac{\mathrm{d}}{\mathrm{d}t} f(u_0,t) \right]^2 \end{aligned} \tag{4.6}$$

Discretization In this section, we discretize Equations 4.5 and 4.6 so that they can be minimized with standard linear optimization toolkits. First, we regularly sample the time t every Δt and use i to denote the i^{th} frame of the animation, that is, the frame at $t = i\Delta t$.

For simplicity, we use an i subscript for entities related to frame i, i. e., $\ell_i = \ell(i\Delta t)$. We also sample each ℓ_i with n_i points $\{\mathbf{r}_{i,1}, \ldots, \mathbf{r}_{i,n_i}\}$. We use length$(\mathbf{r}_{i,j}, \mathbf{r}_{i,j+1})$ to denote the length of the segment of ℓ_i between $\mathbf{r}_{i,j}$ and $\mathbf{r}_{i,j+1}$. The unknowns of our optimization problem are the scalar values $u_{i,j}$ such that $f(u_{i,j}, i\Delta t) = \mathbf{r}_{i,j}$.

Temporal Coherence Given a point $\mathbf{r}_{i,j}$, we seek to find its corresponding points in frame $i+1$. We consider its 3D counterpart $\mathbf{R}_{i,j}$, that is, $\pi(\mathbf{R}_{i,j}) = \mathbf{r}_{i,j}$, and the 3D line $\mathcal{L}_i$ on which it is located. We search for the corresponding point $\mathbf{R}' \in \mathcal{L}_{i+1}$ at the next frame. If there is no topological event between i and $i+1$, we know that the two lines $\mathcal{L}_i$ and $\mathcal{L}_{i+1}$ are separated by a region $\mathcal{R}$ of the model $\mathcal{M}$ where the line passed, i. e., where $\mathbf{n} \cdot \mathbf{V}$ changed signs in the case of silhouettes (Sec. 4.5.1). We find $\mathbf{R}'$ as the closest point $\mathbf{R}_{i,j}$ when considering only paths within $\mathcal{R}$. We implemented this as a shortest path problem on the mesh edges within $\mathcal{R}$. It is also useful to use the image metric instead of the 3D distance since we seek to preserve the coherence in the image plane. Although the camera may be moving, we found it sufficient to use the projection $\pi_{i+\frac{1}{2}}$ at the middle time instant to estimate distances. If more accuracy is needed, one can subdivide the time interval. If there are topological events, we subdivide the frame interval such that each sub-interval has no event and apply the same pairing algorithm within each of them. In practice, we reuse the same subdivision as when we built the space time surface (Sec. 4.5.1). Once we find the closest 3D point $\mathbf{R}'$, we project it onto the image plane to get $\mathbf{r}' = \pi_{i+1}(\mathbf{R}')$. We name j' the index of $\mathbf{r}'$ and define:

$$E_{\text{proj}} = \sum_i \sum_j \left(u_{i,j} - u_{i+1,j'}\right)^2 \tag{4.7}$$

Coverage and Distortion (Aperiodic Case) The coverage constraint (Eq. 4.2) is $u_{i,1} = 0$ and $u_{i,n_i} = 1$ for all i. The corresponding energy term is:

$$E^{\text{a}}_{\text{end}} = \sum_i \left[u_{i,1}^2 + (u_{i,n_i} - 1)^2\right] \tag{4.8}$$

The discrete version of the distortion constraint (Eq. 4.3) is:

$$u_{i,j+1} = u_{i,j} + \frac{\text{length}(\mathbf{r}_{i,j}, \mathbf{r}_{i,j+1})}{\text{length}(\ell_i)} \tag{4.9}$$

for all i and all $1 \leq j < n$. The corresponding energy term is:

$$E^{\text{a}}_{\text{dis}} = \sum_i \sum_{j=1}^{n_i-1} \left(u_{i,j+1} - u_{i,j} - \frac{\text{length}(\mathbf{r}_{i,j}, \mathbf{r}_{i,j+1})}{\text{length}(\ell_i)}\right)^2 \tag{4.10}$$

Distortion (Periodic Case) The equations are similar except that the length of the texture is a and we do not impose the $[0, 1]$ coverage.

$$E^{\text{p}}_{\text{dis}} = \sum_i \sum_{j=1}^{n_i-1} \left(u_{i,j+1} - u_{i,j} - \frac{\text{length}(\mathbf{r}_{i,j}, \mathbf{r}_{i,j+1})}{a}\right)^2 \tag{4.11}$$

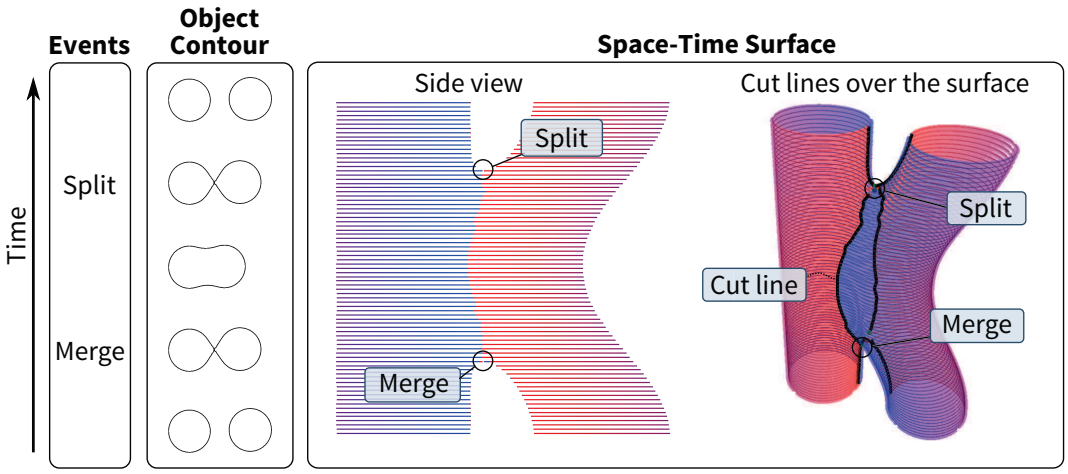


Figure 4.8: Line parameterization for a scene with dynamic geometry and topology. Our discontinuity-reuse algorithm pairs the split and the merger, which limits the number and extent of the cuts.

Putting it Together We assemble all the terms to obtain the final energy in the aperiodic case:

$$E^{\mathrm{a}}_{\mathrm{total}} = w_{\mathrm{proj}} E_{\mathrm{proj}} + w_{\mathrm{dis}} E^{\mathrm{a}}_{\mathrm{dis}} + w_{\mathrm{end}} E^{\mathrm{a}}_{\mathrm{end}} \tag{4.12}$$

and in the periodic case:

$$E^{\mathrm{p}}_{\mathrm{total}} = w_{\mathrm{proj}} E_{\mathrm{proj}} + w_{\mathrm{dis}} E^{\mathrm{p}}_{\mathrm{dis}} \tag{4.13}$$

This is a classical least-squares energy where the $u_{i,j}$ variables are unknown. This linear system is sparse and can be solved in the least-squares sense. In our implementation, we use a sparse Cholesky factorization.

4.7 Handling Discontinuities with Cuts

The parameterization algorithm presented in the previous section handles a single open line, that we assume that the space-time surface $\mathcal{S}$ has a disc topology. However, in general $\mathcal{S}$ has a more complex topology and lines split or merge when a topological event occurs. Our strategy is to decompose $\mathcal{S}$ into disc-like charts so that splits and mergers always happen at chart boundaries. Intuitively, when two lines meet, instead of merging them, we keep them separated by cutting $\mathcal{S}$. While the produced lines are shorter, this ensures that there is no discontinuity within a line. Further, we pair mergers and splits so that they share cuts, thereby minimizing the length of these cuts. Fig. 4.8 shows an example of the cuts.

We detect the split and merger points during the construction of the space-time surface (Sec. 4.5). We extend these points in time to produce lines free of discontinuities. A possible option is to propagate each cut in time individually. Although this would generate discontinuity-free lines, this would also overly segment the lines and generate visually unpleasing results. We propose a better approach where forward cuts emerging from mergers are combined with backward cuts coming from splits. Considering all possible

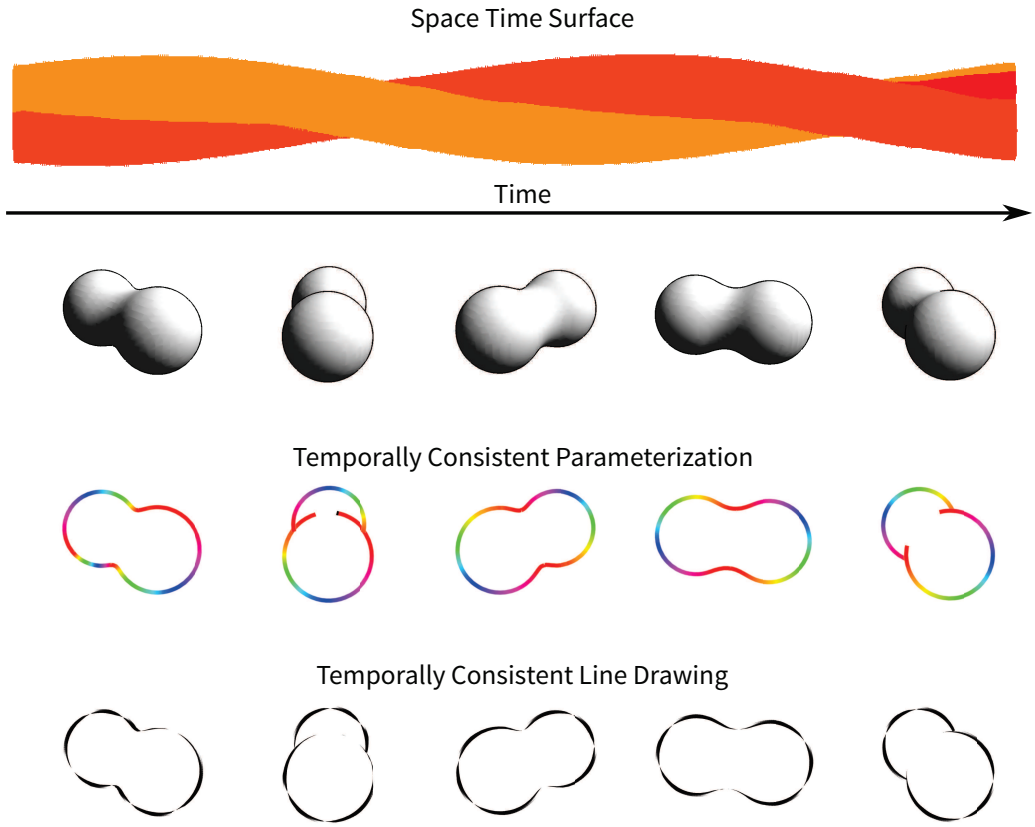


Figure 4.9: « Peanut » shape. Even simple shapes can generate complex lines: The silhouette of a peanut folds in non-trivial ways. Continued on page 57 and page 58.

combinations of forward and backward cuts would generate a combinatorial explosion. We propose a simple heuristic that yields satisfying results in practice.

Once we have detected all the mergers and splits, we analyze the sequence from the first frame to the last, and process the events in order. We describe the case of a merger $\mathbf{k}_i \in \ell_i$, splits are handled symmetrically. We assume that users have specified a parameter σ that represents the maximum sliding that can be introduced for cut reuse. Similar to Sec. 4.6.2, we find the point $\mathbf{k}'_{i+1} \in \ell_{i+1}$ that corresponds to $\mathbf{k}_i$. We collect the points $\mathbf{r}_{i+1,k} \in \ell_{i+1}$ such that $\text{length}(\mathbf{k}'_{i+1}, \mathbf{r}_{i+1,k}) \leq \sigma$ and such that there is no cut between them and the propagated $\mathbf{k}'_{i+1}$ point. If one of them is a split, we pair it with $\mathbf{k}_i$ and nothing needs to be done. If several of them are splits, we pick the one with shortest length. If none of them is a split, we propagate the $\mathbf{r}_{i+1,k}$ to the next frame and iterate the process until we find a split at frame $i + n$. During this process, we keep track of the $\mathbf{k}'$ points that directly corresponds to the initial cut point $\mathbf{k}_i$. If we find several splits at the same time, we pick the point $\mathbf{k}^\star_{i+n}$ for which the distance $\text{length}(\mathbf{k}^\star_{i+n}, \mathbf{k}'_{i+n})$ is the smallest. Finally, we build the shortest path between $\mathbf{k}_i$ and $\mathbf{k}^\star_{i+n}$ on the space-time surface $\mathcal{S}$ and perform a cut along it.

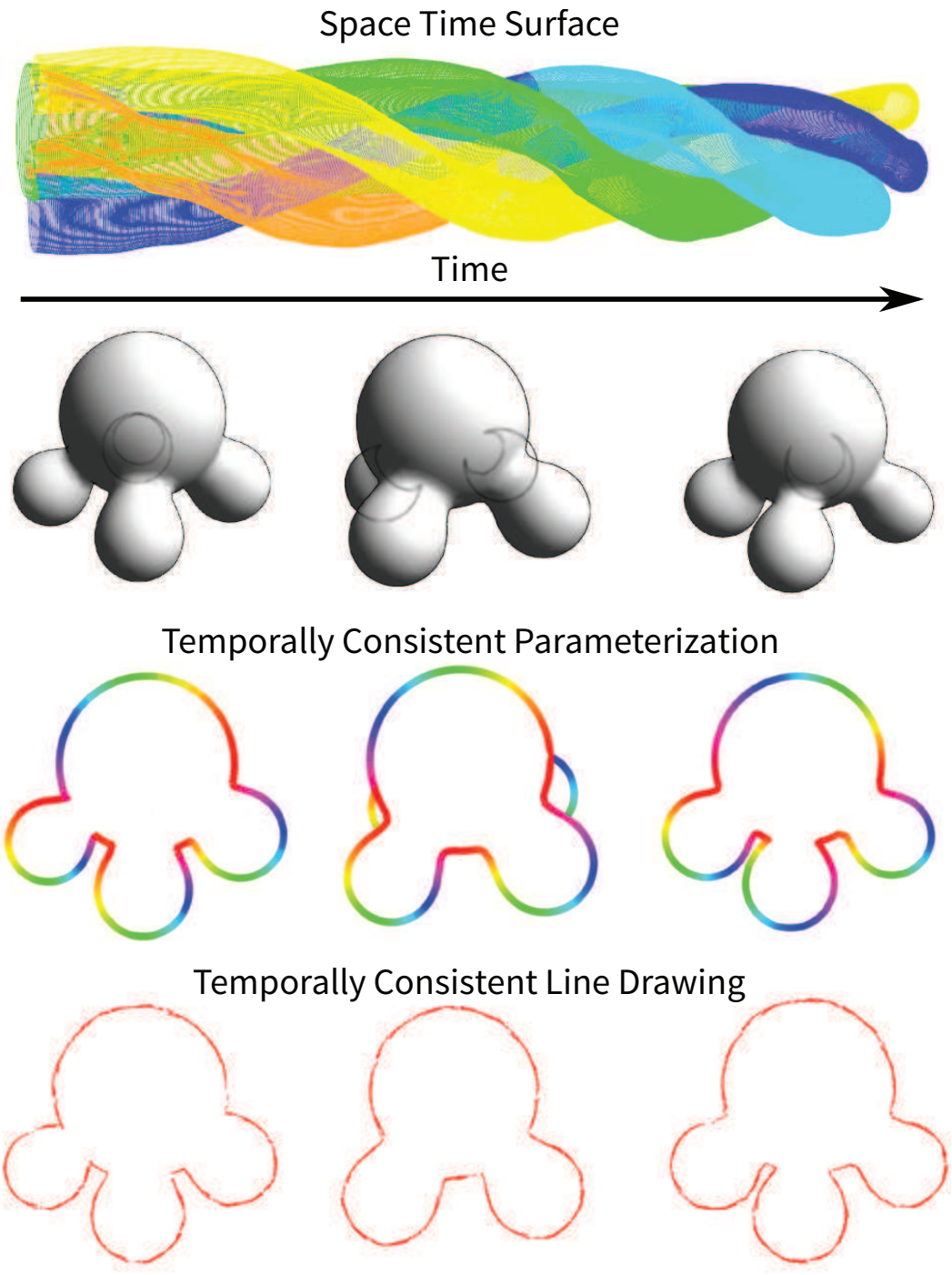


Figure 4.9: « Multipod » shape. The multipod generates many visibility events. Figure continued from page 56.

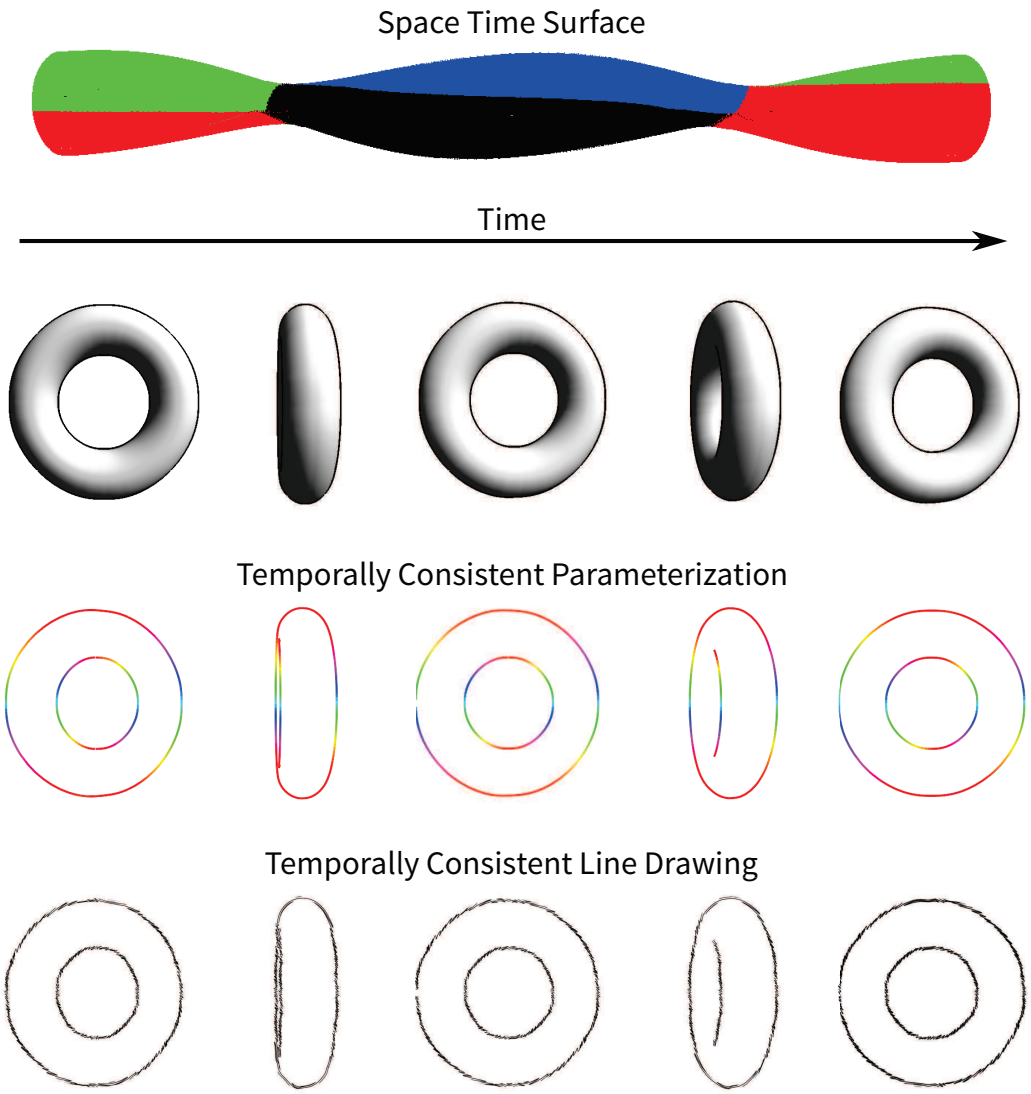


Figure 4.9: Torus. Inner and outer silhouettes of a torus repeatedly merge and split. Figure continued from page 57.

4.8 Results

We have applied our approach to several scenes with various degrees of dynamism and complexity, ranging from simple static scenes with moving viewpoint to deforming geometry with dynamic topology. Some minor jittering can be observed because we handle visibility at the vertex level – this could be addressed by computing visibility at each pixel at the expense of slower computation times. We use the CHOLMOD solver [Che+09] to handle the linear system associated with each space-time line. Solving the least-squares systems takes about 5 seconds for a mesh made of 20k vertices over a sequence of 100 frames. The algorithm's speed depends heavily on the number of split and merge events, and may require up to several minutes for few seconds of animation. Figures 4.9, 4.10, 4.11

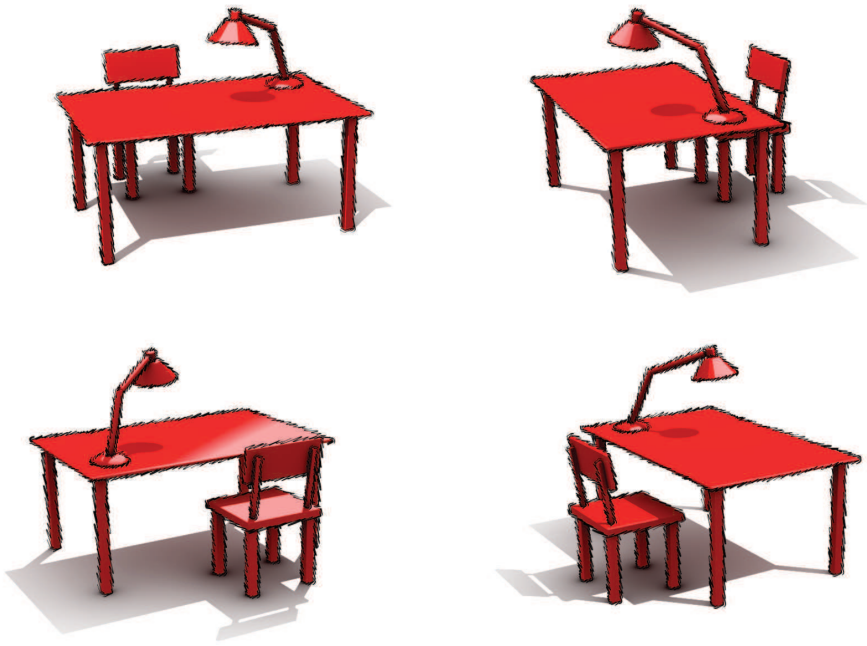


Figure 4.10: Four frames of a desk model.

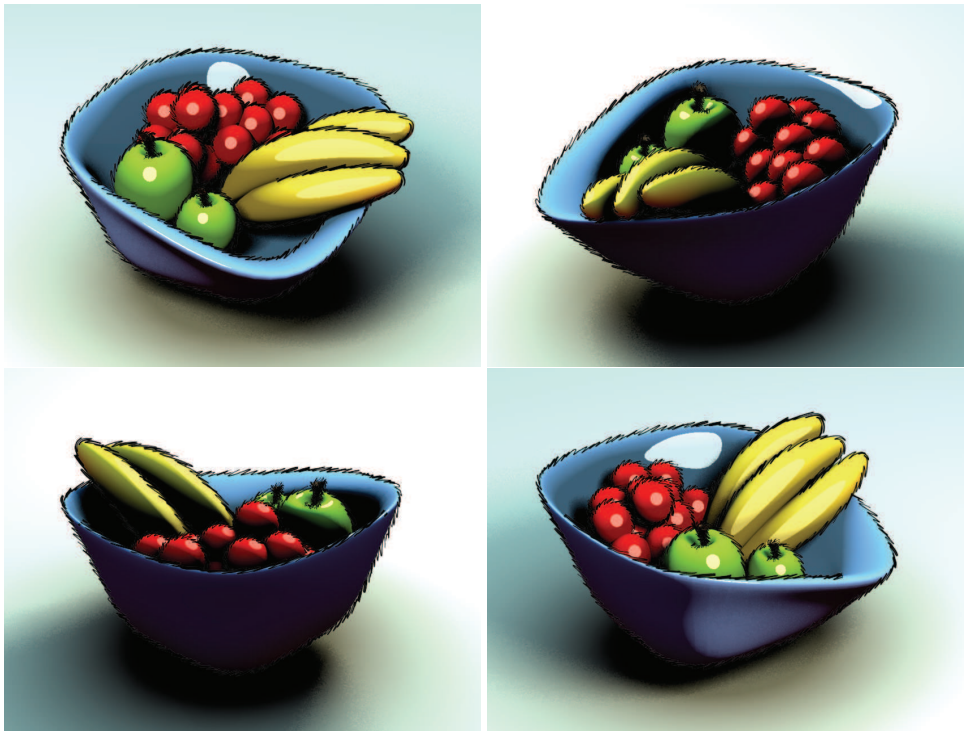


Figure 4.11: Four frames of a bowl containing several objects.

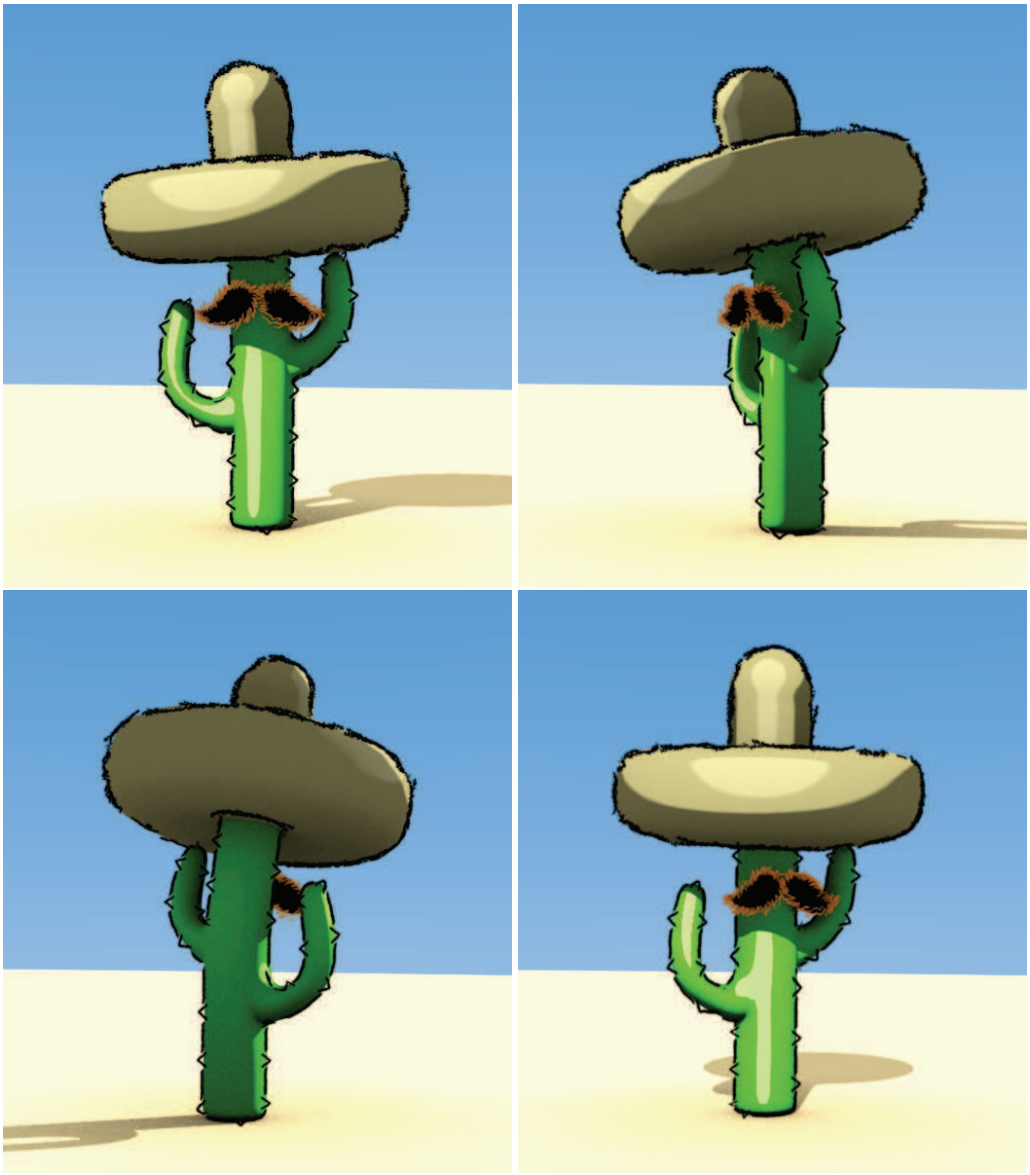


Figure 4.12: Four frames of a more complex example: the cactus scene.

and 4.12 show sample results that we obtain with various shapes. Even with seemingly simple models, the space-time surface can often be complex and nontrivial to analyze, including numerous splits and mergers. Nevertheless, our approach finds a temporally consistent parameterization each time. Our current implementation of the line tracking algorithm handles rigidly moving objects and we are planning to extend it to non-rigid transformations, which requires only technical adaptations since the algorithm itself does not assume this condition. Beside this, our simple proximity-based tracking can handle simple non-rigid animations and demonstrates that our approach also works in this case.

For rendering purpose, we exploit the visibility of the lines stemming from the 3D model to either hide the occluded ones or apply a different style to them.

4.9 Towards Interactive Time-coherent Line Parameterization

Our approach for generating a temporally coherent parameterization on completed animation works fine and we have clearly stated the problems of line segmentation in real-time works that can not use information about upcoming topological events on the space-time surface. Nonetheless, often a real-time method may be interesting for example for previews or interactive use when no completed animation exists, similar to Kalnins' work [Kal+03]. We think it is possible to improve on the problems we analyzed in this work, notably the need to handle topological events.

Parameter Value Diffusion

The basic idea is to approximate our previously presented approach which was globally optimal in such a way that the segmentation occuring in non-global approaches is alleviated. While it is not possible to completetely remove some sort of segementation while keeping temporal coherence, it is possible to remove the segmentation in certain cases. A possible approach which we explored in more detail is the weighted per-frame diffusion of parameter values with consecutive reconstruction via energy minimization similar to our previous approach.

Overview

Notation

V_i	Vertices of all lines at time t_i
$v_{i,j}$	j-th vertex at time t_i, $v_{i,j} \in V_i$
$u_{i,j}$	Parameter value at vertex $v_{i,j}$

The objective is to propagate the parameter values from time t_i to timestep t_{i+1} in a spatially and temporally coherent manner. In order to achieve this, we employ a two-step algorithm, similar to Kalnins et al. [Kal+03] in that we first propagate values and then reconstruct them. However, we propose to propagate parameter values only and independent of their origin, i. e., we no longer track the curves' development which, in turn, removes the need to detect or handle topological events in the diffusion. First, the parameter values in V_i at time t_i are diffused to the vertices V_{i+1} at time t_{i+1}. The parameter values for V_{i+1} are then reconstructed using the diffused values and a set of constraints to minimize the objective function.

Initialization and Diffusion

In order to initialize the values, we assume a uniform parameterization, so that $u_{0,j} = \text{uniform}(v_{0,j})$. Here, $\text{uniform}(v)$ corresponds to the uniform arc-length at the position of vertex v on the curve it belongs to.

To diffuse the values from one timestep to the next, we use a Gaussian-weighted distance function d. In addition to the actual distance between two vertices (in screen- and

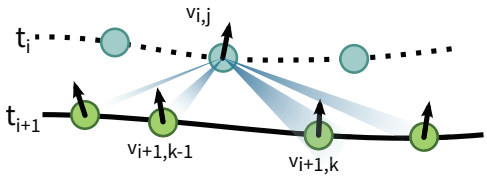


Figure 4.13: Diffusion of $u_{i,j}$ at vertex $v_{i,j}$ to the vertices in the next time step t_{i+1} with the anisotropic weighting function d, taking into account screen- and world-space distance as well as the normal difference.

world-space), we also take other properties into account, in our tests the normal deviation in world-space Fig. 4.13. The weighted sum of these properties is the distance d:

$$d(v_{\cdot,j}, v_{\cdot,k}) = \sum_i w_i p_i$$

where p_i is the property value (distances, normal deviation) and w_i the corresponding weight. The amount of diffusion w_d is then calculated using a Gaussian weighting of the distance:

$$w_d\left(v_{\cdot,j}, v_{\cdot,k}\right) = e^{-d\left(v_{\cdot,j}, v_{\cdot,k}\right) c_f}$$

with a falloff factor c_f. c_f is usually chosen so that in average $w_d > \epsilon$ is only true for a small subset of V.

Finally, the diffused value $y_{i+1,j}$ in vertex $v_{i+1,j}$ takes all vertices V_i into account:

$$y_{i+1,j} = \sum_k w_d\left(v_{i,k}, v_{i+1,j}\right) u_{i,k}$$

Note that $y_{i+1,j}$ is only an intermediate value value that is used in the reconstruction step in order to find the actual $u_{i+1,j}$. It should also be possible to take the motion of the vertices into account. For example, vertices moving towards each other should have a smaller distance assuming that they might eventually and inversely for vertices moving away from each other.

Reconstruction

If it was only for the diffusion to reconstruct the values, it is obvious that after a certain amount of time, the parameter values u would all diffuse to a single value. To counter this, we connect the diffusion with several constraints in an objective function. The minimization in the least-squares sense gives us the final reconstructed values u.

The objective function contains the diffused values as well as constraints in order to keep the slope of u close to the slope of a uniform parameterization:

$$E = w_{diff} E_{diff} + w_w E_w + w_s E_s$$

where w and s correspond to world- and screen-space respectively, in the following denoted by $\Omega \in \{w, s\}$.

$$E_{diff} = \sum_j \left\| y_{i,j} - u_{i,j} \right\|^2$$

$$E_\Omega = \sum_j w_{\sigma,j} \left\| u_{i,j+1} - u_{i,j} - \mathrm{dist}_\Omega(v_{i,j}, v_{i,j+1}) \right\|^2$$

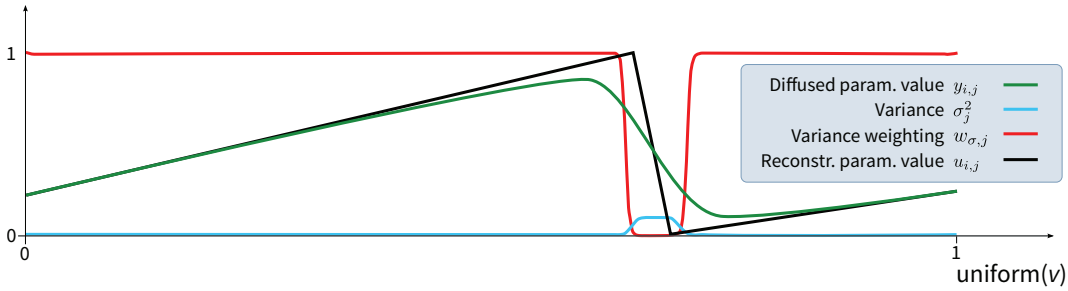


Figure 4.14: Special weighting of the spatial coherence terms E_Ω. The variance σ^2 is shown in blue and the corresponding variance weight w_σ in red, the diffused and reconstructed parameter values in green and black, respectively. In regions of high variance, the uniformity constraint is lowered in order to allow jumps in the parameterization.

where $\text{dist}_\Omega(v_1, v_2) = \|p_\Omega(v_1) - p_\Omega(v_2)\|$ denotes the distance between two vertices in Ω while $p_\Omega(v)$ is the position of v in that space.

In regions where strong differences in the diffusion values $u_{i-1,\cdot}$ arise, e. g., when two curves are close to each other or where the initial uniform parameterization has its start/end point, it is not possible to keep the uniform slope from E_Ω (see Fig. 4.14). Therefore, we reduce these constraints in regions that show a lot of variance in $u_{i-1,\cdot}$. We use the weight $w_{\sigma,j} = e^{-\sigma_j^2}$ with σ_j^2 being the weighted variance

$$\sigma_j^2 = \frac{\sum_k w_d(v_{i,j}, v_{i-1,k}) \left(u_{i-1,k} - y_{i,j}\right)^2}{\sum_k w_d(v_{i,j}, v_{i-1,k})}.$$

Note that $y_{i,j}$ corresponds to the weighted mean.

Implementation We implemented this idea to test its feasibility and we found that we could achieve real-time framerates on moderately sized meshes (100-500k triangles). The bottleneck in our test implementation was the lookup of nearby vertices in the diffusion step which could be resolved by the addition of a spatial acceleration structure. The line extraction itself uses the same method as in the previous work, and poses no problem in terms of timing even on larger meshes.

Discussion

While this approach removes the need for curve tracking or voting schemes, the issues with segmentation from other real-time methods surface again. After a topological event, the parameterization of a curve is no longer monotone (see Fig. 4.15). If the differences (i. e., the variance) in the parameterization are too large, the weight for the E_Ω term in the energy will be reduced and the parameterization will not return to being monotone. One solution would be to allow higher weights for hidden parts of a line. Thus, the uniform parameterization could be restored step by step whenever a part of a line is hidden.

While the general lack of line tracking is on one hand an advantage, on the other hand it makes it hard to allow different styles of lines on a single object. For this case, line IDs could be diffused as in Kalnins' work, but this would remove some of the simplicity of the proposed system.

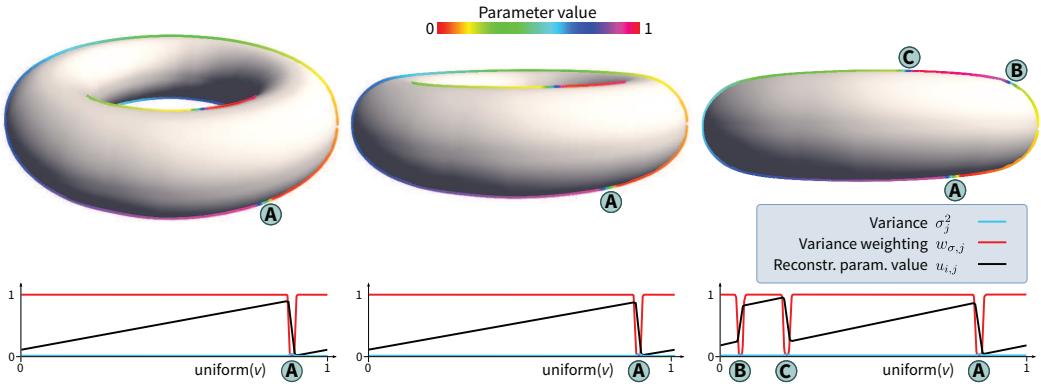


Figure 4.15: The graphs show the parameterization of the outer contour. From the second to the third frame, a topological event occurs. The third graph shows the partial incorporation of the smaller curve into the longer curve between (B) and (C). Depending on the variance weighting w_σ and the energy minimization, this « segmented » parameterization will stay (possibly until it can be resolved when the corresponding curve parts are hidden) or it might smooth out quickly afterwards. If the difference is too large, a quick smoothing may be visually unpleasant, i. e., parameter values could shift rapidly resulting in fast changing textures.

As a different approach, instead of diffusing parameter values for texturing, stroke parameters could be diffused directly. A stroke parameter vector could consist of several recordable properties like direction, pressure, angle etc. These could be recorded from an artist drawing on a set of lines directly. To avoid flickering when very different stroke parameters get merged, the diffusion could be done in a multi-scale fashion, where parameters of low frequency are diffused further while high-frequency parameters, having a stronger effect on the lines, could be diffused more locally. The strokes themselves would then be generated along the lines using the diffused parameters as if an artist had explicitly used a pen to record the strokes. This approach would be less general, since it creates strokes of varying parameters but without parameterization (therefore not allowing texturing) but would have a strong temporal coherence using varying strokes.

Overall, generating a coherent parameterization over time is still up for improvement. For completed animations however, we have described a method to produce temporally-consistent line parameterization that can be used to render animated line drawings. The cornerstone of our approach is the introduction of the space-time surface that represents an animated line. Our approach is grounded on a geometric analysis of temporal consistency based on this surface. We translate this spatio-temporal analysis into a discrete least-squares problem. We have shown that the user can control the trade-off between temporal coherence and distortion using a few meaningful parameters. Furthermore, we avoid over-segmentation by pairing complementary topological discontinuities. Together, these elements enable temporally consistent parameterizations that we demonstrate by rendering several complex animations as line drawings. Using the insight gained from this work, we then proposed a possible approach for interactive line parameterization.

Part II

Photorealistic Rendering

CHAPTER 5

GLOBAL ILLUMINATION

While we investigated intermediary structures and their use in non-photorealistic rendering in the previous part, we will now look at their use in photorealistic rendering, notably in global illumination solutions. While non-photorealistic rendering aims for expressive renderings that give special weight on certain properties of the input data, photorealistic rendering tries to create images from 3d scenes in such a way that the result resembles a photography that was taken from the scene if it existed in reality. With the purpose of creating a photorealistic depiction of the scene, photorealistic rendering methods use only properties relevant for the light transport and only with the aim of correctly resolving the light transport problem. Inbetween these two extremes of photorealistic and non-photorealistic rendering, techniques exist which are based on photorealism in the sense that they show physically based lighting effects but they enhance these results in a way that moves them more towards expressivity.

In this chapter, we will briefly introduce the methods and intermediary structures used in photorealistic rendering which allow the translation of the scene data into 2d images. We focus on global illumination in particular since the contributions in the domain of photorealistic rendering in this thesis are in the context of global illumination.

5.1 The Rendering Equation

In reality, a photo or video camera usually uses a sensor to register the electromagnetic radiation in the visible spectrum, called light or photons. Therefore, everything a camera sees is photons stemming from light sources emitting their light into a scene. After having been emitted and before reaching the sensor, photons interact with the surfaces of the scene or the media in the space between (see 5.2). This interaction is (slightly differently from the the original) expressed in the rendering equation [Kaj86]:

$$\begin{aligned} L\left(\mathbf{x} \rightarrow \omega\right) =& L_e\left(\mathbf{x} \rightarrow \omega\right) \\ &+ \int_{\Omega} f_r\left(\mathbf{x} \rightarrow \omega, \mathbf{x} \leftarrow \omega'\right) L\left(\mathbf{x} \leftarrow \omega'\right) \cos\left(\angle\left(\mathbf{n}_{\mathbf{x}}, \omega'\right)\right) \, \mathrm{d}\omega' \end{aligned} \tag{5.1}$$

where $L(\mathbf{x} \rightarrow \omega)$ is the light leaving point $\mathbf{x}$ into direction ω and inversely $L(\mathbf{x} \leftarrow \omega)$ the light coming to point $\mathbf{x}$ from direction ω. L_e is the emitted light, f_r the bidirectional

(a) Direct only **(b)** Indirect only **(c)** Combined

Figure 5.1: Visual Effect of Indirect Lighting: The addition of indirect lighting, i. e., light bouncing off of surfaces multiple times before hitting the camera, has a strong impact on the perceived visual realism.

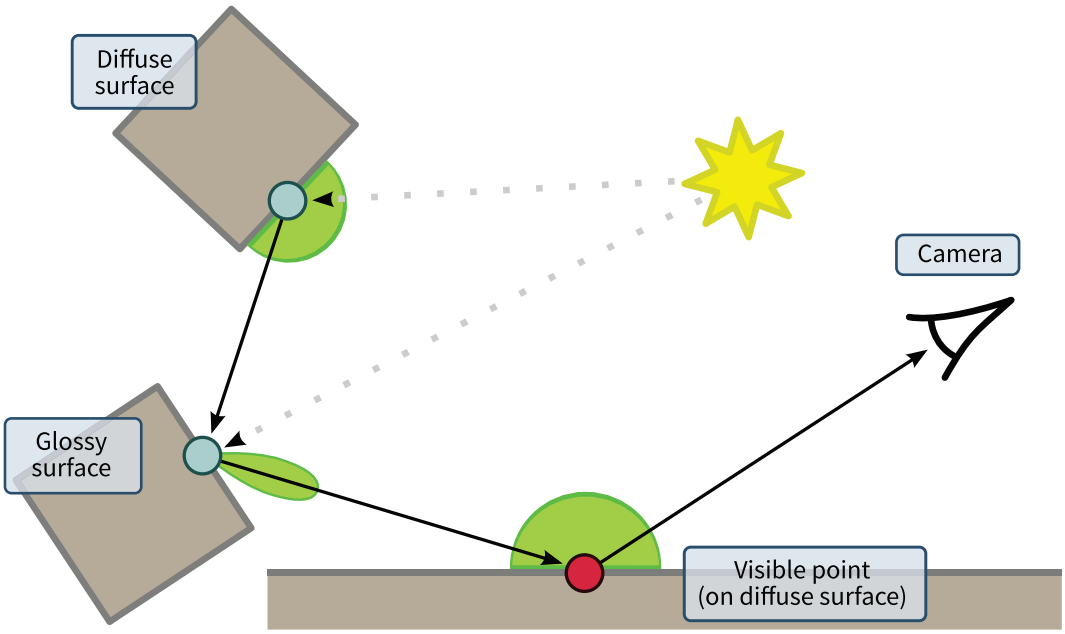


Figure 5.2: Global Illumination/Indirect Lighting: A light source emits light into the scene. This light hits surfaces, where some of the energy is absorbed by the surface while the rest is bounced off. The amount of light being reflected, the changes it undergoes (mainly in wavelength, i. e., color) and its distribution depend on the properties of the surface material. The distribution is indicated by the green areas. Diffuse materials reflect incoming light uniformly in all directions. The glossier a material is, the more directed the reflected light becomes and its main direction is the reflection of the incoming ray over the points normal. When light reaches the camera, its color and incoming direction are recorded (its position on the image plane).

reflection distribution function (BRDF), which models the light transport at a point coming from a certain direction and leaving into another direction. This is the operator that takes into account surface properties like diffuseness and glossiness, color etc. Ω is the hemisphere over the point $\mathbf{x}$ and $\mathbf{n_x}$ the surface normal at $\mathbf{x}$.

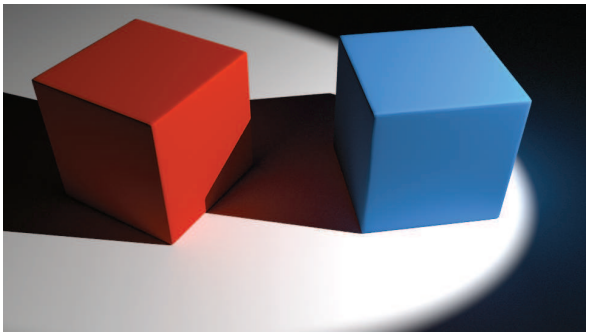
Missing in this formulation are volumetric effects from participating media like fog or smoke or from translucent objects like skin or wax where the light scatters inside of solid object (subsurface scattering). These are not part of the contributions of this thesis but they play an important role in photorealistic rendering.

Indirect lighting and color bleeding are usually the most sought-after effects in Global Illumination (see Fig. 5.1 and Fig. 5.3(a)) as they strongly increase the visual realism and the credibility of an image. Caustics are another effect, where light is reflected by highly specular surfaces or refracted by transparent objects onto diffuse surfaces in such a way that, depending on the geometry, focused patterns can be created (see Fig. 5.3(b)). When the light is treated as a function of wavelength, dispersion effects can be simulated (see Fig. 5.3(c)). These occur when light passes through media of different density. At the interface of the media, each wavelength contained in the light is refracted slightly differently, "splitting up" the light into its components, commonly known from prisms. One of its well known occurrences are in chromatic aberration, an artifact in photography, where edges become blurry from dispersed colors.

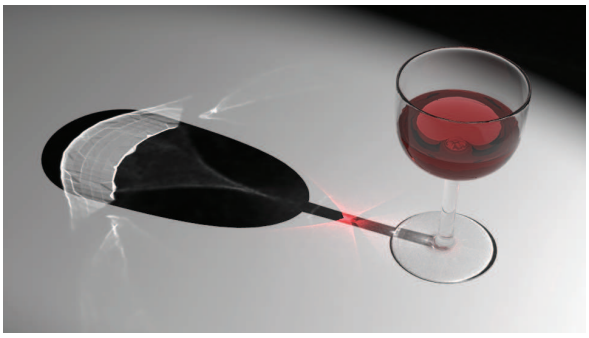
In the following, works in the domain of photorealistic rendering with a focus on global illumination are presented, largely divided into exact, high-quality but not necessarily efficient methods and methods that concentrate on fast (even interactive) computation of these effects.

5.2 Early Works

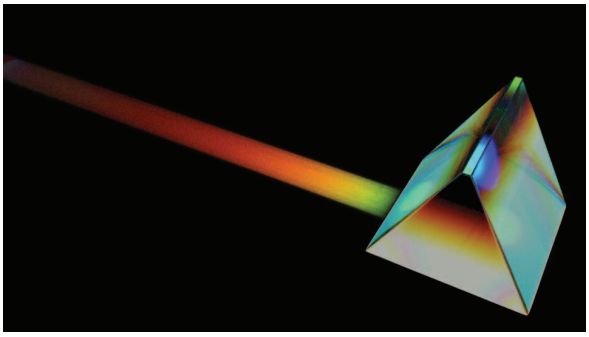
With the introduction of raytracing for photorealistic rendering [Whi80] a slightly simpler equation was used, basically allowing incoming light only from direct light sources or perfectly specular (mirror-like) reflections or refractions, thus omitting indirect lighting from light bouncing in the scene. Indirect lighting though contributes a lot to visual realism especially in scenes where large parts of the geometry are not reached by direct light. Another visually important effect is color bleeding, i. e., strong light hitting brightly colored surfaces which act in turn as colored area lights towards close surfaces. For perfectly diffuse surfaces (i. e., Lambertian reflectors where the BRDF is a constant factor) this can be modeled by radiosity [Gor+84]. To calculate the transfer of energy between the surfaces of a scene, the scene surface is structured into small patches that are related to each other depending on their geometry and visibility. This is a preprocess and all scene surfaces need to be taken into account, even those which will not contribute to a given view. On the other hand, this preprocess generates an energy transfer for the whole scene which remains valid as long as its geometry and lighting remain static allowing changes of the viewpoint without recomputation. Depending on the size of these patches, the results can have fairly strong discretization artifacts. The more general work by Kajiya [Kaj86], called Pathtracing (see Fig. 5.4(a)), solves Eq. 5.1 without being constrained to diffuse surfaces and the need for surface approximations. In this method, rays are sent out from the eye and then connected to the direct light sources as in standard raytracing (possibly following a perfect reflection or a refraction). But in contrast the rays are continued also for not perfectly specular surfaces by taking into account their BRDF. This forms (sub)paths through the scene, which all contribute to the light arriving at the the eye. A closed solution is not possible due to the unknown integrand, which depends on the scene's geometry, materials and lights. Therefore, the integral in the equation must be numerically integrated by sampling the domain. This function F is called an estimator of the integral. If its expected value $E[F]$ equals the correct value of the integral, this estimator is unbiased. The common approach is using Monte Carlo integration, based on sampling with random val-



(a) Color Bleeding



(b) Caustics



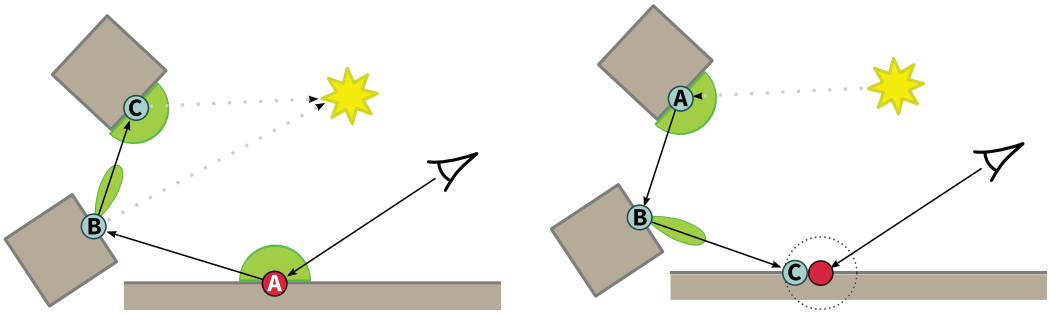
(c) Dispersion

Figure 5.3: Different indirect lighting effects achieved with GI methods. Color Bleeding: The white light hitting a colored surface gets absorbed in those wavelengths not corresponding to the surface color, the other wavelengths get bounced off. Here, the white light is bounced off of the red box, « bleeding » onto the ground in front of the box. **Caustics**: Light going through transparent objects is refracted depending on the material's index of refraction (IOR). The geometry can create focusing and diffusing effects. **Dispersion**: The light is refracted differently depending on the specific wavelength. White light, containing all wavelengths in the visible part of the electromagnetic spectrum, is dispersed (« spread out ») in space by a prism, showing the wavelengths individually in the same way as a rainbow.

ues and using probability distribution functions (PDF) to weight the contribution of each sample. When these PDF follow the function that is to be integrated, Monte Carlo sampling achieves a better variance reduction with increasing sample count than, for example, uniforming or stratified sampling.

5.3 Physically-Based Methods

Pathtracing provides several advantages over the previous method: it allows arbitrary BRDFs and geometry and eliminates artifacts from geometry approximation, replacing them by noise. The noise in Monte Carlo integration (which is an expression of the vari-



(a) Path Tracing **(b)** Photon Mapping

Figure 5.4: Path Tracing: Path tracing starts from the eye and constructs paths through the scene. From the primary hit point *A*, a secondary ray is generated by sampling the BRDF at this point. This ray hits *B*, lying on a glossy surface. From there, another element for the path is generated hitting *C* and so on. At each hit point, connections are also made to all light sources. **Photon Mapping**: Photons are emitted from the light sources. Here, a photon hits a diffuse surface in point *A* and is recorded in the photon map and is then, by sampling the BRDF in *A*, bounced in the direction of *B*. There it is recorded again and bounced off again, finally recorded in *C*. To calculate the indirect light at the visible points (as an example the red point), the photon map is queried for all recorded points in a certain radius around the point. The final indirect light value is computed as the weighted sum of the returned points.

ance of the estimator) will converge to the correct result (an overview and details about MC integration and its use in rendering can be found in Veach's thesis [Vea97]). Several extensions to pathtracing have been developed like Bidirectional Pathtracing or Metropolis Light Transport, addressing several problems mostly linked to the light transport in difficult scenes (i. e., where indirect light needs to pass through small openings) [Vea97].

The drawback of all Monte Carlo-based methods is that even though noise-free (i. e., error-free) results can be obtained theoretically, the computation time needed may be impractical for a given purpose. Other methods exist therefore that solve the rendering equation using an estimator whose expected value does not correspond to the correct value of the integral but always keeps a bias. These methods usually don't exhibit noise artifacts as Monte Carlo methods but possibly other artifacts. It depends on the method which is more preferable for the viewer.

One such method, modeling radiosity with surface patches [Gor+84], was already presented above. The scene approximation via surface patches and the approximated visibility between them for the energy transfer leads to an error that can be reduced by decreasing the patch size. Another problem is the quadratic complexity in the number of patches, which makes it prohibitively expensive to render large scenes or scenes with too much detail.

Instead of relating all surfaces (surface patches) and their energy transfer to each other, it is more efficient to start from the actual light sources and follow their contribution through the scene. Photon Mapping (PM [Jen96], see Fig. 5.4(b)) is an offline method that simulates photons being emitted from light sources and traced through the scene. At their point of impact, the photons are then randomly either absorbed or reflected/refracted depending on the surface properties. For diffuse surfaces, they are reflected in a uniform manner over the impact point's hemisphere, in case of glossy surfaces they are reflected

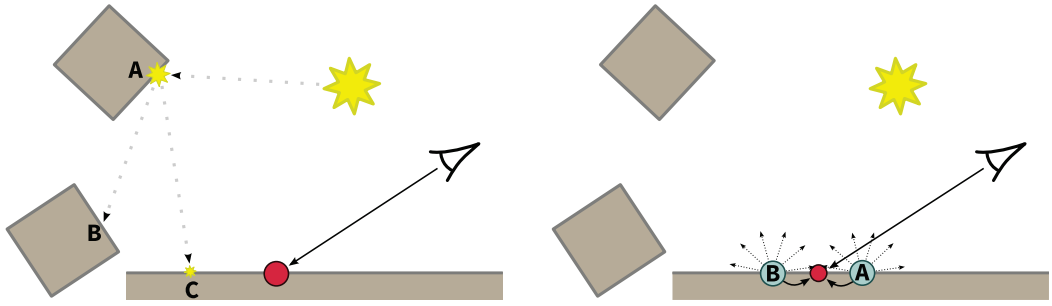
proportionally to the BRDF (i. e., scattered towards the main reflection direction) and for transparent and translucent objects, they are refracted towards the inside. At each intersection of a photon with the surface, its position, energy and incoming direction are recorded in a spatial point structure (photon map).

From this information, an image of the indirect light can be generated by gathering the indirect illumination stored in the photon map. For a point corresponding to a pixel in the image plane, the photons in a given radius around that point are gathered. The BRDF response of each photon is weighted while accounting for surface changes and summed up giving the indirect light. Since photons are shot from light sources, problems can be caused by large area or environment light sources as they need many photons to cover their area. Another problem, albeit more general (i. e., also applicable to other methods), are paths containing one or more glossy reflections. In that case, a large number of photons are needed to avoid undersampling the scattering. Recently, a number of extensions to Photon Mapping have been presented, notably a version where an arbitrary number of photon passes allows to increase the accuracy of the solution progressively (PPM [Hac+08b]) and, as an extension to that, stochastic PPM [HJ09], which combines PPM with an efficient way to compute distributed rendering effects where several samples need to be averaged like depth-of-field, etc.

5.4 Time-efficient Methods

While Photon Mapping is faster than Pathtracing in many cases, it may not be efficient enough for many production rendering environments. Keller introduced Instant Radiosity [Kel97] (see Fig. 5.5(a)) that gave (with the hardware at that time) rendering times of a few seconds per image. The method resembles Photon Mapping slightly in the sense that starting from the direct light sources rays are traced into the scene with a certain sample count. At each impact an additional point light source (so-called virtual point light, VPL) is created to approximate the diffuse reflection of the incoming light at that point. Since this can only take into account the diffuse reflection (i. e., the BRDF cannot contain directional components), it is a radiosity solution. Like PM, the rays are traced further in a direction chosen proportionally to the BRDF (diffuse scattering). At each bounce, the samples count is reduced until it reaches zero. For each created point light source, the scene is rendered with shadows. These images are then combined with uniform weighting resulting in the final image containing indirect lighting.

One major issue of Instant Radiosity is the necessary compromise between speed and quality. The detail Instant Radiosity can resolve depends on the number of VPLs used. At the same time, its time complexity is depending linearly on that number and therefore making it possibly unfeasible when a large number of VPLs are required in complex scenes. Instead of always iterating over all VPLs, a solution is to use clusters that approximate a number of similar and spatially close VPLs. Lightcuts [Wal+05] is such an approach. First, as a preprocess, a given set of VPLs (possibly created using the method described in Instant Radiosity) is clustered hierarchically using a binary spatial tree structure. The clustering is executed such that the error it introduces is minimized. Then, for each point that is to be shaded, the set of clusters is recovered whose estimated error (as compared to evaluating all the lights individually in that cluster) falls below a given threshold. This search is



(a) Instant Radiosity **(b)** Irradiance Caching

Figure 5.5: Instant Radiosity: A number of samples are launched from the direct light sources. When they hit a diffuse surface (e. g., *A*), a virtual point light source (VPL) is created taking into account the point's material. For multiple bounces, samples are launched from the hit point (with adjusted energy) and new point light sources are created (*C*). On non-diffuse surfaces (*B*), no lights are created. After all samples are used, the scene is rendered using the direct lights and the VPLs, thus creating the indirect lighting. **Irradiance Caching**: Samples are distributed over the surface of the (visible part of the) scene. For each sample point, the scene is rendered over the hemisphere at that point (*A, B*). Assuming diffuse surfaces, these renderings are integrated with a cosine weighting, giving the irradiance at the sample points. The indirect light for any point is then calculated by interpolating between the existing samples in the neighborhood. If no sample exists in the neighborhood, a new one is generated and appended to the list of samples.

done by traversing the tree from the root in a breadth-first manner. Whenever a node (representing a cluster) is found whose error for shading the given point falls below the threshold, this cluster is used as an (approximated) VPL and the traversal of that subtree is stopped. The set of nodes found for shading a point forms a cut through the tree, hence the name “Lightcuts”. The point is then shaded by the representatives of the clusters instead of the full number of VPLs, strongly reducing the amount of computations needed. Therefore, as opposed to Instant Radiosity with a linear complexity in the number of VPLs, Lightcuts is shown to have a strongly sublinear complexity, allowing the use of several hundreds of thousands of VPLs while still achieving reasonable render times of a few minutes.

The indirect light from diffuse surfaces is in most cases fairly soft, e. g., on a flat surface with no occluders close by, the indirect light does not contain high frequencies, i. e., the integrand (the light incoming over the hemisphere of a point) does not change rapidly. Since the indirect lighting is for the most part soft, instead of calculating its contribution at every pixel, it can be more efficient to only calculate it at certain positions and interpolate the values in between. This idea was presented as Irradiance Caching [War+88] (see Fig. 5.5(b)). Note that irradiance refers to the total amount of incoming light, hence it is a scalar value with no directional information (i. e., how much light from which direction). In its basic form (interpolation of irradiance) it could be applied to any global illumination method as it is independent of the way the irradiance is computed. In Irradiance Caching, the indirect illumination is evaluated via raytracing samples over the hemisphere of the receiving point and is stored in an irradiance cache. To calculate the indirect light for the image, a “lazy” approach is applied: For a point that needs to be shaded, a sampling metric is used to calculate weights for each of the already existing caches in its neighborhood. If there are no or too few points with weights above a given threshold, a new cache is created at the given point and added to the list of caches. Crucial here is how the weights are

calculated and that the structure that stores the caches allows fast access to the neighbors. In practice, the sampling metric is a measure of the rotational and translational geometric difference of the position of two caches (illuminance gradients). This approach results in a denser sampling at strong curvature (e. g., edges, corners) where the indirect lighting is expected to change faster. Only few samples need to be used where the irradiance changes slowly (e. g., flat surfaces). Finally, to quickly find the nearest neighbors of a cache, an octree is used as a spatial acceleration structure to avoid linear searches through the list of caches. Still, the approach of illuminance gradients only partially captures possible reasons for changes in the irradiance (occlusion by unconnected objects for example). For a better estimation of the change in irradiance, Irradiance Gradients [WH92] exploit the information gathered in the hemispherical sampling for the caches. The depth and the light value in each direction can be used to improve the prediction of change and thus the sampling metric.

Even with the improved gradients, Irradiance Caching remains a method to capture the diffuse indirect light only, the reason being that when the scalar irradiance value is calculated, the viewpoint is not given. Therefore, directional dependency as in glossy or specular BRDFs cannot be taken into account. The obvious solution is Radiance Caching [Kri+05] where in each cache the incoming light is stored together with its directional component (called radiance). The radiance is represented in spherical harmonics (SH, see Sec. 6.3.1) coefficients as a form of a more compact and useful storage than storing the contribution from each discretized direction in itself. As in Irradiance Caching, gradients can be computed for the change of radiance per translation and rotation. In contrast to Irradiance Caching, the amount of indirect light at a given point reflected towards an outgoing direction can be evaluated as the convolution of the incoming radiance with the (possibly slightly glossy) BRDF of the given point over its hemisphere. When both, the radiance and the BRDF, are represented in the same (spherical) base, then the convolution is the inner product of the their coefficients. The SH representation of the scene's BRDFs can be precomputed for a number of outgoing directions, allowing a fast evaluation. The evaluation's complexity is linearly proportional to the number of SH coefficients used. The more coefficients are used, the higher the frequencies are that can be contained in the BRDF, allowing to capture glossier BRDFs.

Going one step further, the radiance, i. e., the *incoming* light (as opposed to the *outgoing*), can be convolved with the BRDF due to its bidirectionality. In [Sch+12] the incoming light is pre-convolved with a number of BRDFs with different magnitudes of glossiness (from glossy to diffuse). This is done by recording the radiance in a high-resolution map (corresponding to the highest glossiness possible) and then building an image pyramid hierarchy on top of it, resulting in a type of mip-map for different factors of glossiness. Now, in order to evaluate the outgoing light at a point towards a certain direction, the level corresponding to the point's BRDF is chosen – the reflected light is then a simple pixel lookup. Consequently, for interpolated caches, no new convolution needs to be done, resulting in a constant complexity time evaluation with only minor quality impairment when compared to Radiance Caching.

In the next chapter, we present a contribution to Point-Based Global Illumination [Chr08], which is a method that basically combines Lightcuts and Radiance Caching in the sense that it gathers contributions from a hierarchy of lights (actually a sphere-based scene ap-

proximation) and creates radiance caches that are then convolved with the points' BRDFs. A more detailed explanation can be found in the following chapter.

CHAPTER 6

QUANTIZED POINT-BASED GLOBAL ILLUMINATION

This chapter we present an extension to the spatial hierarchy that is used in PBGI showing that this structure in the current implementation contains strong redundancies. We propose an algorithm to find these redundancies and exploit them for compression via quantification.

6.1 Bottleneck Memory Usage

As pointed out in the last chapter, Point-based Global Illumination (PBGI) uses several elements of previous techniques for the computation of global illumination. It is a popular technique, intensively used in film production and recently adapted to real-time scenarios. On the contrary to unbiased, physically-based methods such as path tracing, PBGI cannot easily reproduce all indirect lighting effects but is rather used to approximate a subset of the most critical ones in a fast, noise-free fashion. This includes ambient and directional occlusion effects, as well as color bleeding, all of which stem from one-bounce diffuse light transport.

In the following, we will give a short summary of Spherical Harmonics which are typically used in PBGI to store reflected light over a sphere. This summary is followed by an explanation of PBGI itself with respect to the problem approached in this work.

6.1.1 Data Storage in PBGI

In a preprocessing step, PBGI starts by distributing a dense point set on the scene's surfaces before shading them according to the scene's primary light emitters, taking into account direct visibility only. This point sampling can be performed for example using Poisson Disk distributions or surface tessellation. Starting from this colored point set, a spatial tree is constructed bottom-up by computing at each node a spherical function capturing the diffuse directional reflection of its related subtree. Octrees and Bounding Sphere Hierarchies (BSH) have been successfully used as such PBGI structures. The spherical color functions in the tree's nodes are often stored as Spherical Harmonics (SH) representations

Figure 6.1: Visual comparison. In the ground truth image (left), the nodes' reflectance is approximated using 3 bands of spherical harmonics resulting in 108 bytes per node. In our quantized result (right), nodes reflectance data is indexed over a LUT with 10 000 entries optimized using k-means in the parameter space, resulting in a 14 bits per node data coding scheme, giving an effective compression ratio for the reflectance data of approximately 60. The quantized result captures most of the nodes' spherical function space and only leads to slight discretization errors. The overall result is perceptually and numerically very close to the ground truth (PSNR 66dB).

with a low number of coefficients. A complete and more detailed explanation of the full PBGI algorithm can be found in Sec. 6.3.2 and Fig. 6.2.

6.1.2 Memory Issues

One of the main problems PBGI applications are currently facing is the large amount of data that needs to be stored in the tree nodes. Even when using spherical function approximations with a small memory footprint (e. g., SH), the sheer amount of nodes needed to correctly approximate a large scene makes it often impossible to keep all nodes in memory. For instance, it is quite usual to use SH with 3 bands only – a very coarse approximation able to capture diffuse BRDFs only. However, this already results in a coefficients vector of 27 floats per node (9 floating-point coefficients per RGB channel).

6.2 Overview

Studying the node values in practical scenes, we observe a significant coherence between nodes, independent of their position in space (e. g., two similar distant buildings lit by the sun). Many nodes share indeed very similar spherical functions and a large portion of the memory is wasted in replicating again and again similar data chunks. Therefore, we propose to exploit this scattered redundancy by quantizing the nodes' spherical function data over a small look-up table (LUT), optimized in a fast pre-process. We use a k-means clustering in the SH parameter space over a small subset of the scene's nodes to define the LUT entries and progressively quantize all nodes' data entries at first bounce shading time by substituting the nodes' spherical functions with a simple index over the LUT. This substitution is performed in a streaming algorithm to avoid having the full, non-quantized tree in memory at any time. As a result, our PBGI tree memory footprint is significantly smaller and allows to process larger scenes without resorting to out-of-core methods and

with almost no visual differences in the final rendering. Note however that our approach can be combined with out-of-core methods to reach even larger scene sizes.

6.3 Background

6.3.1 Spherical Harmonics

Spherical Harmonics (SH) are largely a way to approximate spherical functions, i. e., functions defined over the surface of a sphere, for example in spherical coordinates (ϑ, φ) with $0 \leq \vartheta \leq \pi, 0 \leq \varphi \leq 2\pi$. SH are comparable to the Fourier Series in the sense that SH are represented by a set of orthogonal basis functions representing different frequencies. In Fourier Series, the basis functions are expressed in terms of sine and cosine. For SH the 1D associated Legendre polynomials $P_l^m(x)$ are used, defined over x in the range $[-1, 1]$. Here, l refers to the degree of the polynomial while m is the order, where $l \geq 0$ and $-l \leq m \leq l$. Roughly, l can be understood as the frequency with $l = 0$ being the constant component.

Since Spherical Harmonics are a 2D representation and the associated Legendre polynomials form a 1D basis, the SH basis functions $Y_l^m(\varphi, \vartheta)$ are expressed in terms of the associated Legendre polynomials P_l^m. Although SH are defined as complex numbers, we are only interested in the real part:

$$Y_l^m(\varphi, \vartheta) = \begin{cases} \sqrt{2} N_{(l,m)} P_l^m \cos\vartheta \sin(m\varphi) & m > 0 \\ Y_l^0 & m = 0 \\ \sqrt{2} N_{(l,-m)} P_l^{-m} \cos\vartheta \sin(-m\varphi) & m < 0 \end{cases}$$

where $N_{(l,m)}$ is a normalization factor so that the Y_l^m form an orthonormal basis.

With these basis functions, it is now possible to express an arbitrary spherical function $f(\varphi, \vartheta)$ by projecting it into the SH space. Depending on the highest degree L, the function is band-limited, i. e., the higher frequencies are cut off. For each degree and its corresponding orders, a coefficient c_l^m can be found by evaluating the inner product of the respective basis function Y_l^m with the given function f over the sphere:

$$c_l^m = \int_0^{2\pi} \int_0^{\pi} f(\varphi, \vartheta) Y_l^m(\varphi, \vartheta) \sin\vartheta \, \mathrm{d}\vartheta \, \mathrm{d}\varphi$$

The function f can then be evaluated as $\hat{f}$, the sum of the products of available coefficients and their respective basis functions:

$$\hat{f}(\varphi, \vartheta) = \sum_{l=0}^{L} \sum_{m=-l}^{l} c_l^m Y_l^m(\varphi, \vartheta) \tag{6.1}$$

In practice, the integral is evaluated with a cosine weighted random distribution of directions over the sphere, Ω:

$$\omega = (\varphi, \vartheta) = \left(2 \arccos\left(\sqrt{1 - \xi_0}\right), 2\pi\xi_1\right)$$

where ξ_i are uniformly distributed, independent random variables. This slightly simplifies the integral to:

$$c_l^m = \int_\Omega f(\omega) Y_l^m(\omega) \, \mathrm{d}\omega \tag{6.2}$$

A good overview for the theoretical background and practical implementation can be found in [Gre03].

6.3.2 Point-Based Global Illumination

Preprocessing

Sampling The point sampling of the scene can be done in any manner that uniformly distributes points over the scene's surfaces. A fast approach is building a cumulative distribution function (CDF) over the triangle areas and sample uniformly from that function. For a large number of samples, this ensures a distribution of points proportional to the triangles' areas. A triangle's area itself is then sampled again in a uniform way. This gives in average a good sampling but does not ensure a minimum distance between the points. If this is desired, then Poisson Disk sampling may be used. Usually though, this approach is much slower than the CDF sampling and the gains are not significant, when the CDF sampling is done using low discrepancy sequences like Halton [Hal64]. The points approximate the scene's surface and can therefore be understood as discs with a certain radius. It is important that the radius is chosen large enough so that there are no holes and at the same time not too large to avoid large approximation errors.

Storing Reflected Light For each sample, the diffusely reflected light (possibly from multiple light sources) is gathered and stored using a spherical function representation. In PBGI, this representation is usually in the form of Spherical Harmonics using a low number of bands L (degrees), for example $L = 3$, resulting in 9 coefficients c_l^m. These 3 bands allow the representation of functions with a low upper frequency such as diffuse reflections. The error of the SH approximation rises quickly when a BRDF has a glossy component. To capture those, more bands would be needed, rapidly increasing the memory requirements with only little gain. Therefore, 3 bands are usually chosen as a trade-off between quality of approximation and memory requirements with the constraint of diffuse-only BRDFs.

Additionally, the visible area from all directions for a point is stored. As mentioned above, the points are assumed to have a certain extent. The angle-dependent visible area of a disc is $A_v(\omega) = \max\left(0, \pi r^2 \langle \mathbf{n}, \omega \rangle\right)$, with no visibility when seen from its back (with respect to the disc's normal $\mathbf{n}$). The SH coefficients for the area are thus (see Eq. 6.2):

$$a_l^m = \int_\Omega A_v(\omega) Y_l^m(\omega) \, \mathrm{d}\omega$$

The coefficients for the reflected light also take into account the visible area, which is necessary for the evaluation step of averaged SH representations. Note though that there is no angular dependency for the reflected light in diffuse reflection:

$$p_l^m = \int_\Omega \mathbf{B} A_v(\omega) Y_l^m(\omega) \, \mathrm{d}\omega$$

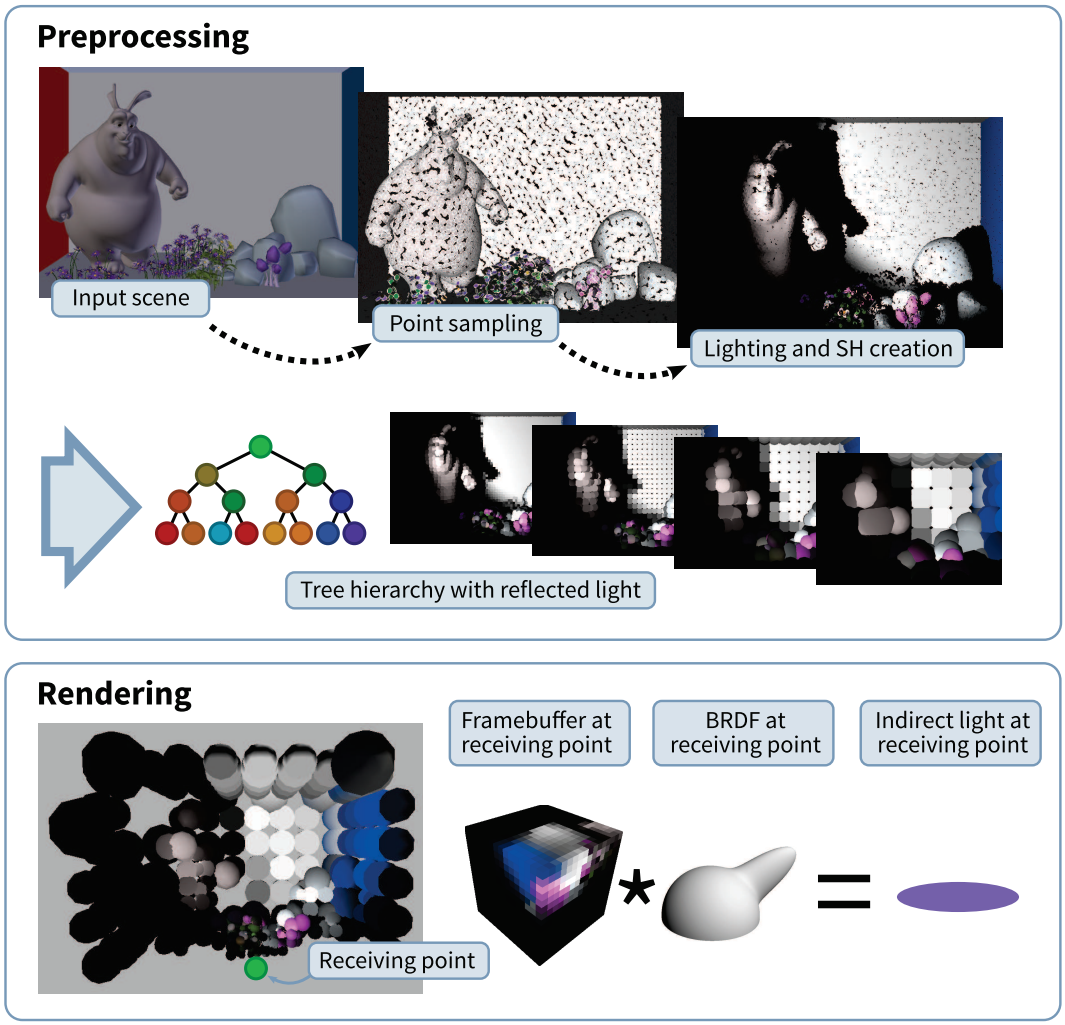


Figure 6.2: PBGI overview. PBGI consists of two steps, preprocessing and rendering, shown here. For an explanation of each step, see Sec. 6.3.2.

where **B** is the diffusely reflected direct light (usually a RGB triple). To evaluate the outgoing (i. e., reflected) light in a direction ω, Eq. 6.1 is used and the result divided by the visible area in that same direction, correctly taking into account averaged SH with different visible areas.

Point Hierarchy As shown in the next section, when evaluating the contribution from a certain solid angle, it is possible, instead of evaluating all points that lie in that solid angle, to evaluate a single point that averages the points in its neighborhood. In order to allow this for any position and solid angle, a spatial hierarchy is built over the points, averaging their position and spherical function representations. These hierarchies can be spatial partitioning structures like an octree or kD-tree or bounding volume hierarchies (BVH) like a bounding sphere hierarchy (BSH). When the inner nodes of the tree are created (i. e., having two or more children), the position of an inner node is simply the average of the

position of its children. The color and area SH coefficients need to be averaged as well, even though technically it is not the "average" but the sum (color and area are becoming larger when adding two children). Therefore, the SH coefficients in an inner node are simply the sums of the coefficients in its children.

Rendering

The positions that are to be shaded with indirect lighting are called *receiving points* and are usually the scene intersections with the camera (primary) rays (i. e., those that are projected onto pixels in the rendered image). At each receiving point, a cube buffer (i. e., 6 2D image buffers forming a cube) is placed onto which the points' contributions are splatted. As pointed out before, it is not necessary to splat all the points themselves but rather approximations from the hierarchy. For this, a threshold solid angle σ is defined. With this threshold in place, the hierarchy is traversed in a breadth-first fashion starting at the root. For each encountered node, the visible area towards the receiving point is evaluated and used to calculate the solid angle that subtends the receiving point. If this solid angle falls below σ, the node is splat onto the buffer and the traversal of the tree at that node is stopped. The splat's color is that of the node evaluated into the direction of the receiving point and the size is its solid angle (possibly approximated by simpler shapes like a square of the same area for easier splatting). Once a leaf is reached, depending on its distance it may no longer be splat but the points(s) (discs) inside are raytraced as the solid angle method becomes too inaccurate.

When the tree has been traversed completely, the actual indirect light color can be computed as the convolution of the colors over the hemisphere of the receiving point with its BRDF (i. e., the integral in the rendering equation, see Eq. 5.1). Note that even slightly glossy BRDFs can be taken into account here as this means only a stronger weighting in a certain direction. Also note that PBGI allows arbitrary area lights (for example using texture maps) with $L_e > 0$.

Multiple indirect bounces are possible by recursively applying the above procedure for each sampling point as the receiving points during the preprocess after the first bounce has been calculated.

6.3.3 K-means Clustering

K-means is a method of partitioning a given point distribution into k clusters. The goal is to find a partitioning, where the sum of distances between the arithmetic center of a cluster C_i and its assigned points $p_j \in C_i$ is minimal. More formally:

$$\arg\min_{C} \sum_{i=1}^{k} \sum_{p \in C_i} \|p_j - \mu_i\|$$

where $\mu_i = \frac{1}{|C_i|} \sum_{p_j \in C_i} p_j$ is the centroid position of cluster C_i. This minimization is often done using the Lloyd relaxation algorithm [Llo82]. First, k initial cluster centers are chosen, either randomly among existing points or using a heuristic to find a good initial distribution. Now, each point is assigned to the cluster which it is closest to. The center of each cluster is then updated as the average of the points assigned to it. These two steps,

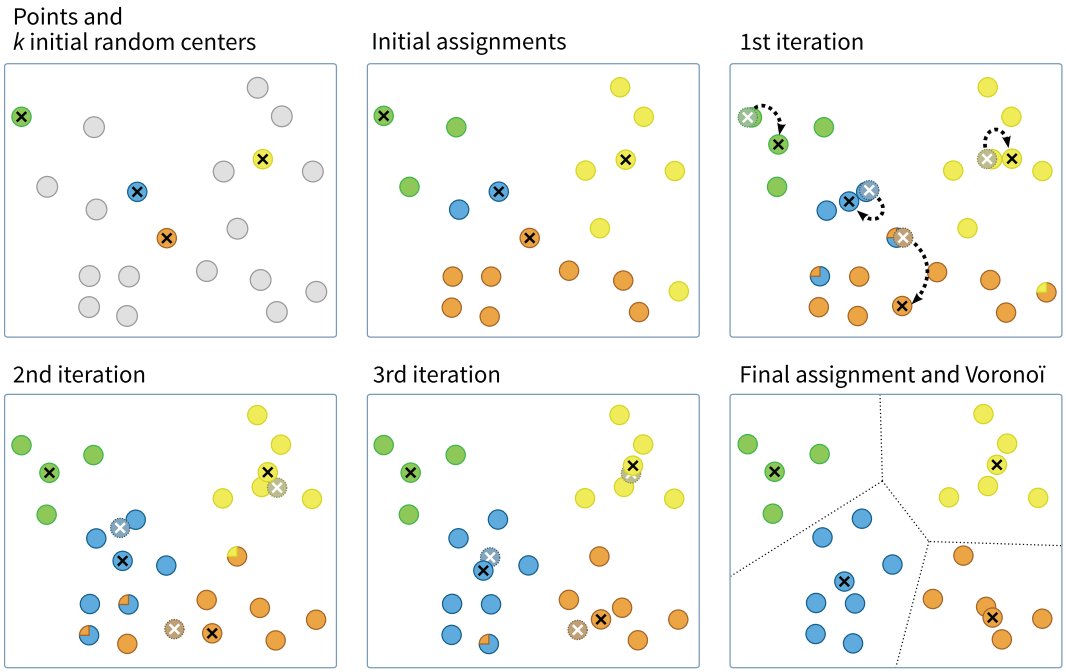


Figure 6.3: K-means example. Shown is the initial point set in a 2d example in the top-left and the initial $k = 4$ cluster centers at the positions of points randomly chosen from the set, marked with a black cross. Each point is first assigned to the cluster center closest to it, then the cluster centers are updated to the mean of their assigned points (top-right) with the position of the previous center marked with a white cross. Points that changed their assignment show their previous assignment in the upper-left quarter. The final partitioning equals the Voronoï diagram (bottom-right).

assignment and center update, are repeated iteratively. Since at each iteration the position of the cluster centers can change, the assignment in the following iteration may change as well. The algorithm terminates when no assignment changes anymore (or after a fixed number of iterations, which is often the case in applications that do not need the exact solution but for which an approximation is sufficient), see Fig. 6.3 for an example. The resulting partitioning equals the Voronoï tesselation with the cluster centers being the Voronoï cells' centers.

This algorithm can handle arbitrary spaces for which a distance metric can be defined. The search for the closest cluster center can be accelerated significantly by using a spatial partitioning structure, that allows finding the centers in $O(\log\ k)$ instead of $O(k)$.

6.4 Previous Work

PBGI has been originally introduced by Christensen [Chr08] to compute ambient occlusion and color bleeding, before quickly becoming a reliable solution for fast global illumination. Its principle builds upon surfel-based ambient occlusion for real time applications [Bun05] as well as direct point-based rendering techniques [GP07], which first substituted point hierarchies to polygons in a rasterization process.

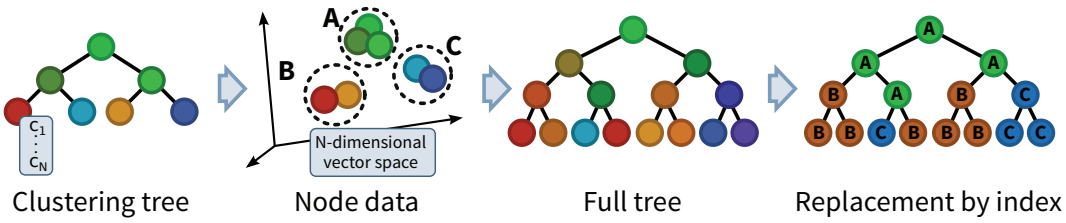


Figure 6.4: *Left*: A quantization tree is constructed from a subsampling of the scene's points. Each node in this tree contains spherical function approximations, usually in the form of coefficients vectors (NDV) which typically exhibit high redundancy. The resulting set is clustered using k-means in the NDV space (*middle-left*), leading to a NDV LUT. At rendering time, the full tree is constructed, quantizing progressively each NDV to the closest LUT entry (*right*). Note that leaf nodes are omitted as, usually, they do not contain NDVs during the rendering step.

PBGI can be implemented on the GPU for final gathering [Rit+09] and even reach real-time performances [Hol+11] with high resolution dynamic scenes using adaptive, per-receiver level-of-detail extraction in the scene. These efficient implementations usually replace the original octree with a more flexible bounding volume hierarchy [RL00] while simplifying the internal node structure (e. g., single scalar value). Memory issues have so far been mostly tackled using out-of-core frameworks maximizing cache usage and uniform quantization to code nodes' data with half-float precision [Kon+11].

The approach we propose in this work is orthogonal to such methods. Our focus on factorizing PBGI data within a scene is inspired by recent trends in geometric modeling [Pau+08] and image representation [Wan+08], which develop object-/category-specific compression spaces, while our particular choice of a LUT-based approach relates to popular fast image and shape retrieval methods [SZ03].

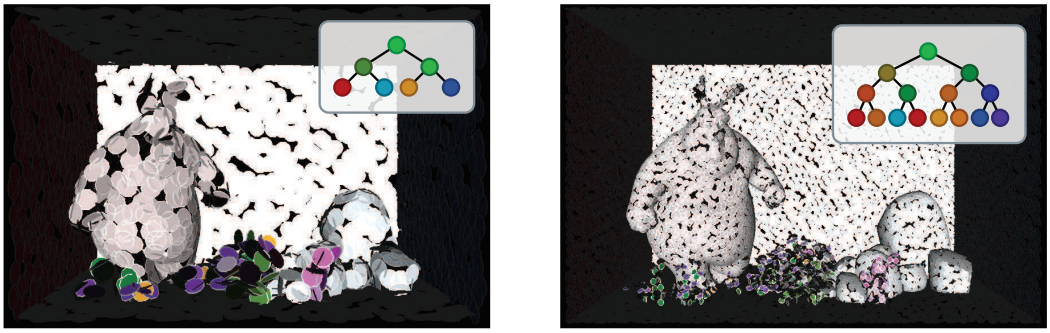
The general problem of GI data compression has been extensively studied over the last decade and the popular SH basis [Slo+02] is already a form of local compression, easy to combine with subspace analysis (e. g., PCA-based methods [Slo+03]). Our scene-aware quantization scheme is orthogonal to such methods and, from an implementation point of view, much simpler.

6.5 Tree Data Compression

6.5.1 LUT Construction

Our quantization scheme starts by defining a spherical function LUT, learned from the scene at initialization (see Fig. 6.4 for an overview). Our approach is generic in the sense that we can apply it on arbitrary spherical functions, as long as it takes the form of a N-dimensional node data vector (NDV). In practice, we use SH with 3 bands per color channel, resulting in 27-dimensional NDVs. Let $w \in R^N$ be such a NDV, with individual entries $w_{i,k}$ (e. g., SH coefficients). We express similarity between two NDVs as a real-valued distance function $d(w_i, w_j)$ In practice, we use the L2-norm of their difference:

$$d_{L2}(w_i, w_j) = \sqrt{\sum_{k=1}^{N} (w_{i,k} - w_{j,k})^2},$$



(a) Sparse sampling for LUT constr. **(b)** Final dense sampling

Figure 6.5: Sparse and dense sampling. For the construction of the lookup table, a significantly smaller amount of samples (a) is used than for the final rendering (b). Shown here are the surfels (i. e., the point samples and their spatial extent).

We use RGB as the color value space of our functions, but more perceptually motivated spaces (e. g., Lab space) can also be used.

We control the quantization process by specifying the size l of the LUT for which we need to determine the most representative NDV in the scene. We adopt a variational approach in the form of a k-means clustering in R^N. Given S_f, the initial point-sampling of the scene, we start by randomly selecting a small subset S_d, typically one to two orders of magnitude smaller than S_f (see Fig. 6.5). Then, in order to capture NDVs at various scales and properly cover the parameter space for our LUT optimization, we shade S_d before constructing a temporary PBGI tree over it. We use **all** its NDVs as a N-dimensional point set over which we compute a k-means clustering. To do so, we use the Lloyd algorithm [Llo82], starting with l random centers, and relocating them to minimize d_{L2}, see Sec. 6.3.3 for details. Ten to twenty iterations are usually enough to converge. Moreover, a dimensionality of $N = 27$ is low enough to allow the efficient use of acceleration structures like kD-trees for the nearest-neighbor (i. e., nearest cluster center) search. Finally, we extract the l stabilized centers and use them as LUT entries. Later on, they are indexed using $\log_2(l)$ bits by the full PBGI tree nodes.

6.5.2 On-the-fly Quantization

As soon as the LUT is computed, the full PBGI tree can be constructed for indirect lighting evaluations. Once the basic tree structure is initialized (e. g., bounding sphere hierarchy), without NDV, the leaves are shaded (i. e., the spherical functions for the reflected light are constructed) from the scene's light sources using spherical sampling and the NDVs are propagated bottom-up to the root, averaging children NDVs at each internal node.

As our initial problem was that a whole PBGI tree may possibly not be held in memory, it is often not possible to build the whole tree first and then quantize NDVs to their respective LUT indices. Instead, we quantize the NDVs on the fly. More precisely, once the NDV of a node is computed from its children, it is classified against the LUT centers in R^N and replaced by the LUT index of the closest center according to d_{L2}. We speed this step up by using a kD-tree for the closest-cluster search.

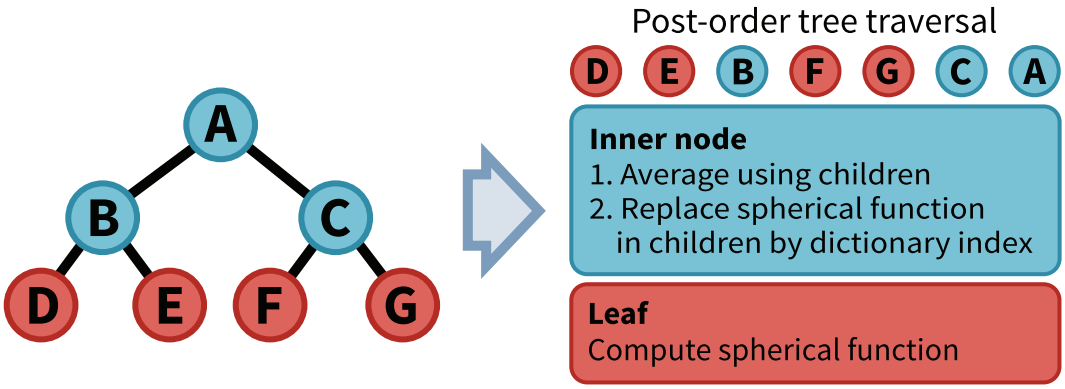


Figure 6.6: On-the-fly quantization of the full rendering tree. To avoid having the full, unquantized tree in memory, we substitute the spherical functions in each node on the fly. This is done by traversing the full tree in a post-order fashion where the spherical functions are only temporarily computed (i. e., when they need to be averaged in the parent node) and substituted by the index as soon as they are no longer needed.

As summing up quantized NDVs would lead to error amplification in the bottom-up propagation process, each NDV must be computed from actual **non-quantized** children NDVs. We solve this issue by traversing the tree in post-order (see Fig. 6.6), so that the children of a node are always traversed first and can be safely quantized by the node itself, which is traversed immediately after (see Alg. 1). This avoids maintaining more than $q+1$ non-quantized NDV in memory, with q being the tree node's arity (e. g., 2 for a BSH, 8 for an octree).

Algorithm 1: Post-order node quantization

```
Data: tree, LUT
Result: quantized tree
begin
    nodes ← post_order(tree)
    foreach n ∈ nodes do
        if n is leaf then
            n.NDV ← compute_spherical_function(n)
        else
            n.NDV ← average_children(n.children)
            foreach c ∈ n.children do
                c.NDV ← LUT_Quantize(c.NDV)
```

6.5.3 Compression

The number of centers dictates the compression ratio. For up to 65k different LUT entries (or cluster centers), each index can be encoded in 2 bytes. Therefore, considering our initial scenario, each 27-floats NDV (i. e., 108 bytes, assuming 4 bytes per float) can be quantized

Clusters	Index size (in bits)	Memory usage (in MB) / **Compression rate** Quantized SH only	Quantized Full Tree Leaf method A	Quantized Full Tree Leaf method B
100	7	8.75 / **123.4x**	388.75 / **3.76x**	288.75 / **5.05x**
1000	10	12.50 / **86.4x**	392.50 / **3.72x**	292.50 / **4.99x**
10 000	14	17.50 / **61.7x**	397.50 / **3.67x**	392.50 / **4.91x**

Table 6.1: Compression rates for different amounts of cluster centers. The index size (i. e., the number of bits required to encode a cluster index) depends directly on the number of clusters. The memory usage is given for a scene with 10M surfels using a binary tree with one leaf per surfel, resulting in 10M leaves and 10M inner nodes. The compression corresponds to two different leaf formats, as shown in Tab. 6.2. The size of the unquantized SH NDVs is 1080 MB (corresponding compression factor in column « Quantized SH only ») and the size of the full unquantized tree is 1460 MB (corresponding compression factor in column « Quantized Full Tree »).

		Position	Color	Normal	Area
Memory (in bytes)	A	12	6	6	2
	B	2	6	6	2

Table 6.2: Leaf formats. « A » refers to storing the data using half-floats [Kon+11] except for the position which keeps full precision. « B » refers to storing the position in a parent-relative fashion as done in QSplat [RL00].

to 2 bytes, resulting in a compression factor of more than 50 for the spherical functions, the LUT size being negligible for large enough scenes.

Of course, the tree structure still needs to be encoded (e. g., positions), but efficient solutions exist [RL00]. Besides our primary focus on spherical function compression, the full tree memory footprint depends on the leaf format and the arity of the tree. Usually, no spherical functions are stored in the leaves. In some cases this is beneficial though, for instance when the surfels' BRDF is not purely diffuse but slightly directional. In such cases, storing a spherical function in the leaves improves the results and therefore benefits from our quantization scheme.

In the case of a binary tree with one surfel per leaf and no spherical functions in the leaves, we achieve compression rates of 61x (10k LUT entries) to 123x (100 LUT entries) for the nodes' spherical functions and 3x to 5x for the whole tree (see Tab. 6.1), mostly depending on the way the data is stored in the leaves (see Tab. 6.2). In the case of an octree, the full tree compression is lower due to the higher leaves-to-inner-nodes ratio.

6.6 Results

We implemented our quantization technique in the Yafaray raytracing engine using Poisson Disk sampling to generate the initial point set. In all quality comparisons, the ground truth stands for the original (unquantized) PBGI algorithm [Chr08].

In Fig. 6.8(a) we compare the k-means clustering against a scene-oblivious random sampling of cluster centers. We can observe that the k-means clustering provides a significantly smaller error. This is displayed in Fig. 6.7. The cluster error is a measure of the difference of each node's quantized spherical function from its original, non-quantized value and is given as an absolute value. The overall error is computed as the mean over

	No LUT	Number of clusters		
		100	1000	10 000
LUT constr.	—	12.25	17.03	22.60
Tree constr.	49.72	101.89	145.37	193.89

Table 6.3: Clustering and tree construction timings (in sec) for the « Bunny and Bird » scene using 5M surfels (resulting in 5M inner nodes) and a representative node fraction of 10% on a single CPU thread. A kD-tree is used to speed up the closest-cluster search in the LUT and tree construction, resulting in sublinear computation time growth.

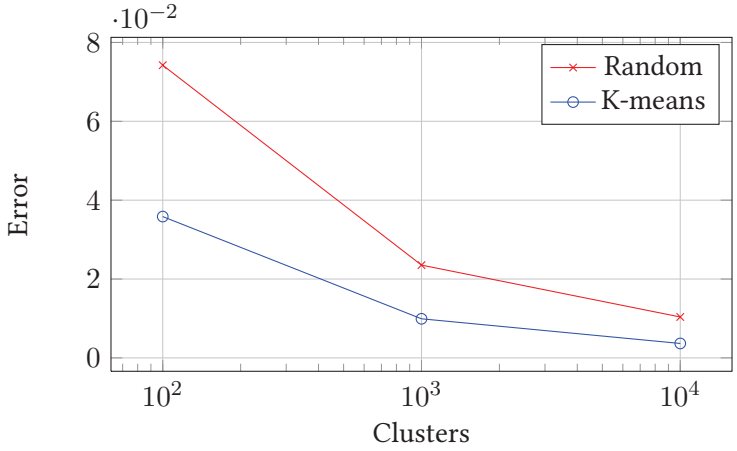


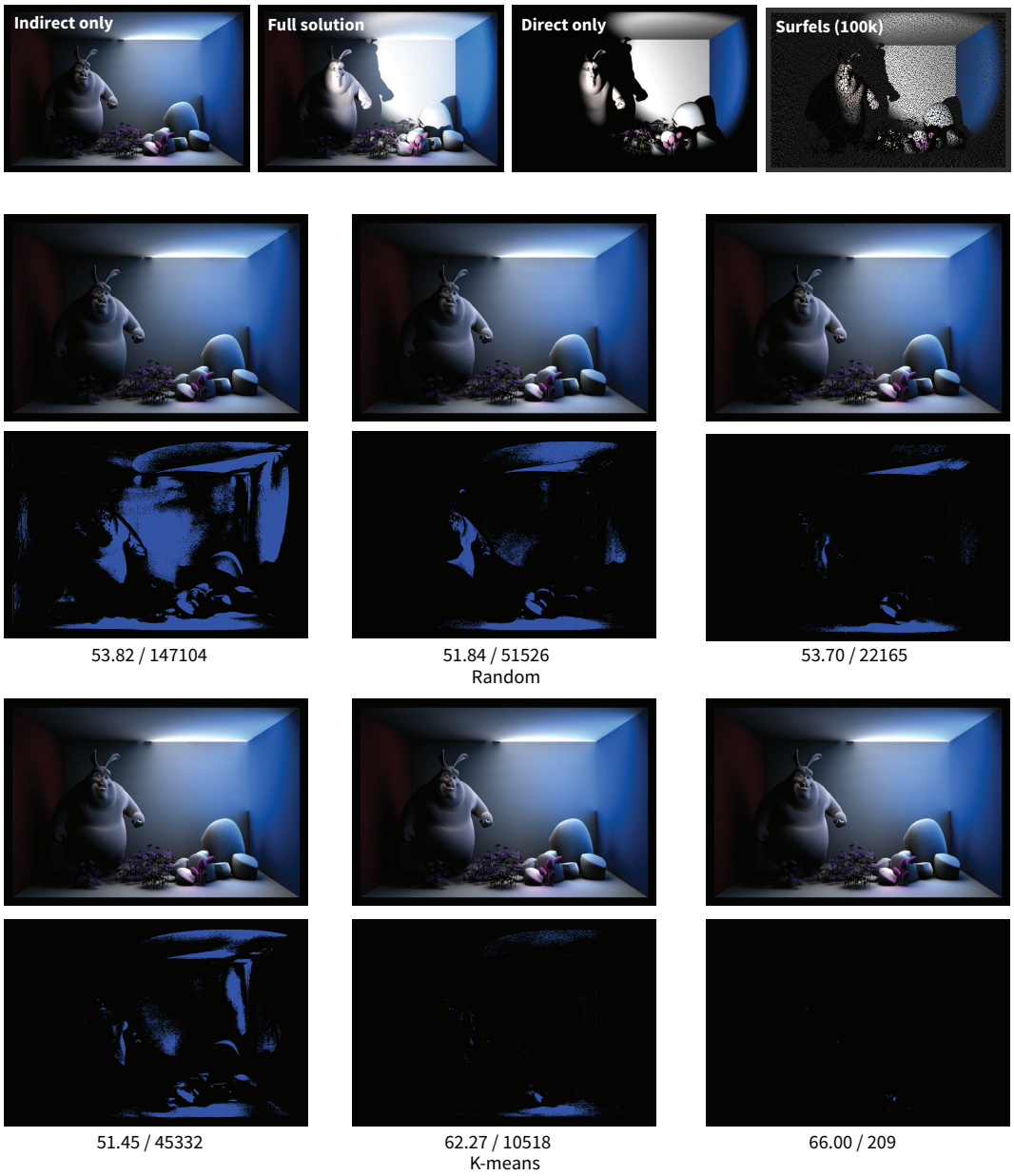
Figure 6.7: Approximation quality for random and k-means cluster LUT in the « Big Buck Bunny » scene. The *error* is the average of the absolute mean error between each node's actual data and its approximation.

the absolute errors in each node. Using k-means clustering decreases the probability of missing frequently repeating spherical functions, leading to a smaller mean error.

The influence of the LUT size can be observed in Fig. 6.8(c): using 100 clusters only, the red curtains color bleeding is not captured correctly. This introduces a significant error when compared to the ground truth, which is fixed by increasing the number of cluster/LUT entries. Similarly, a too sparse subset for the initial LUT construction can produce artifacts. In particular, when some very small surfaces (i. e., undersampled by the point generation process) have unique or rare spherical functions that are not otherwise sampled in the scene, the closest cluster can be far away in NDV space and the node cluster error is high. In the case where such a surface is strongly lit and other surfaces are close-by, the expected light bounce may not occur with the correct color or intensity (see Fig. 6.9).

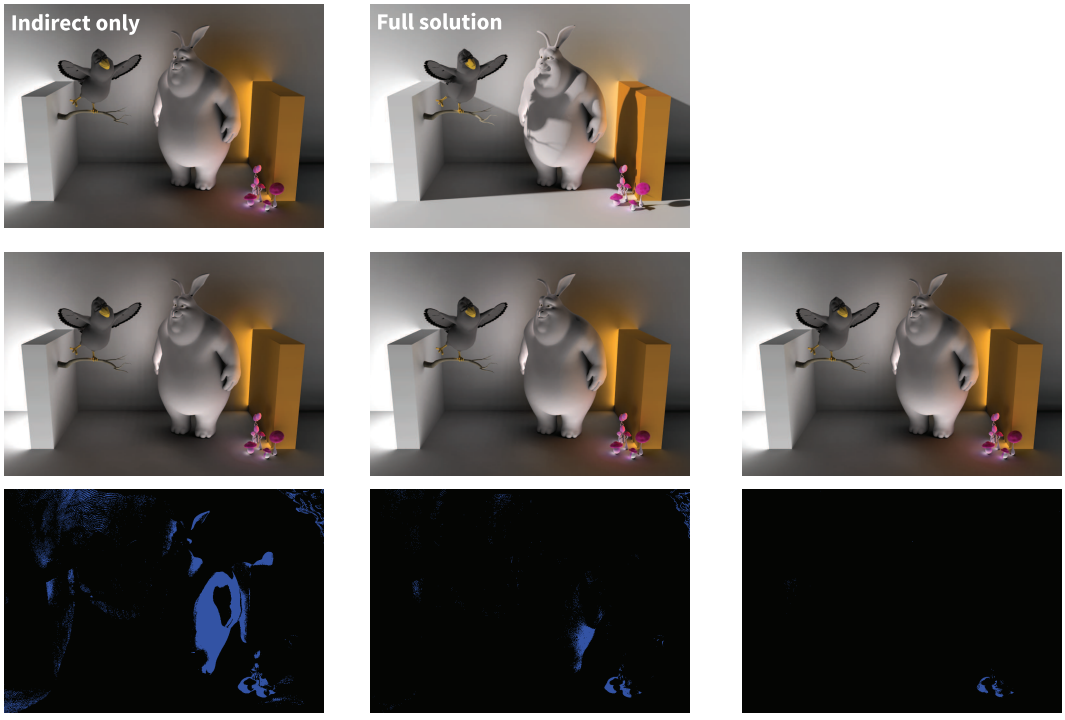
The different perceptual error measures [Yee04] presented in this work show that, overall, a negligible quantization error is introduced, even in complex scenes with highly varying color distributions and under strong quantization rate.

About temporal coherence, slight flickering artifacts occurred in our experiments when using too few LUT entries (e. g., 100 to 1000 depending on the scene), which were all fixed by increasing this number.



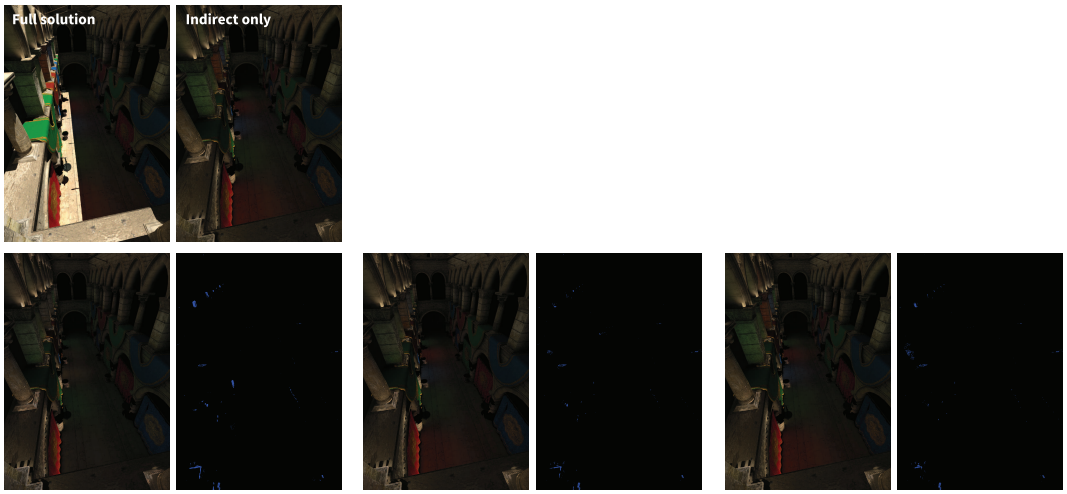
(a) « Big Buck Bunny » scene, random and k-means optimized LUT

Figure 6.8: Visual comparison (indirect contribution only) between ground truth (top left) and our quantization, using 100 (top middle left), 1000 (top middle right) and 10 000 clusters (top right). The error images (bottom) are the perceptual differences in the Lab color space [Yee04], depicted in black (no visible difference) and blue (visible difference). For numerical comparison, we give the PSNR for RGB images and the number of Perceptually Different Pixels [Yee04] (PDP) below each error image (<PSNR>/<PDP>). The full rendering is displayed on the bottom left. (Figure continued on the next pages.)



40.66 / 52 574 | 49.22 / 9510 K-means | 56.92 / 1519

(b) « Bird and Bunny »



43.24 / 2232 | 46.17 / 1287 K-means | 47.61 / 1323

(c) « Sponza Yard »

Figure 6.8: « Sponza Yard » and « Bunny and Bird » scenes. (Figure continued from previous page.)

(d) « Sponza Side »

Figure 6.8: « Sponza Side » scene. (Figure continued from previous page.)

Figure 6.9: We vary the number of initial nodes for the LUT construction between 0.1% (left), 1% and 10% (right) of the final node count (5M here). While only a little visual difference appears when dropping from 10% to 1%, using 0.1% starts to introduce visible artifacts (close-up), missing the pink spherical functions of the mushroom tops and damaging color bleed on the ground.

(a) Direct illumination only

(b) With PBGI

(c) With Quantized PBGI

(d) Perceptual error

Figure 6.10: A more complex example (LUT with 10 000 clusters).

Finally, we report the LUT and full tree construction time in Tab. 6.3 (Intel Core2Quad, 2.83MHz, 8GB), using $32 \times 32 \times 6$ precomputed SH coefficients for uniformly distributed normal directions. Note, however, that speed was not the prime focus of this study.

6.7 Towards Alternative Compression Techniques

We have introduced a new scene-aware quantization scheme for PBGI data which exploits its redundancy. By learning a small set of representative spherical functions in the parameter space, we are able to substitute full node data with accurate quantized values in a memory-efficient streaming process, leading to significant compression ratios. Our approach is simple, easy to implement in any existing PBGI system and intuitive to control. We experimented with various scenes, showing that it introduces almost no visual difference. So far, our work is mostly focused on one-bounce indirect diffuse lighting and the low frequency nature of band-limited SH certainly helps the quantization process. Other research directions include factorized PBGI LUT among scenes, sampling strategies accounting for surface proximity, combination with out-of-core schemes, symmetry analysis, and application to real time PBGI systems. Our quantization scheme also may be helpful in other SH applications.

In the current state of this work, the user specifies the number of clusters k. Instead of this fixed number, it might be possible to generate an adaptive number of clusters, depending on an accuracy measure and the actual clustering that is already in the data. One could use mean-shift instead of k-means which would find dense areas of NDVs and automatically mark them as clusters, thus adapting better to different scenes without user interaction.

Lobe Compression with Directed Spherical Functions

Generalization to glossy and specular PBGI remains an open question. In order to allow reflection function of higher frequency than diffuse Lambertian reflection, using SH with more coefficients is computationally and memory-wise prohibitive. Using more coefficients would increase the dimensionality of the NDV. Since we use kd-trees for the clustering and querying of the LUT, the "curse of dimensionality" (here the computational unfeasibility of maintaining a search structure of high dimensionality vs. brute-force search) would quickly pose a serious problem. In order to make high-frequent BRDFs (as occurring in glossy materials) viable, we have looked into possible extensions of PBGI using spherical function representations that are better adapted for the higher frequencies of glossy light transport.

Compression using Gaussian-like Distributions

Glossy reflections contain much higher frequencies than diffuse reflections, as noted before. Therefore, storing them using SH with many coefficients or even directly in a cube map with high resolution is unfeasible due to memory constraints. We therefore explored the possibility to store the reflected light in strongly directional spherical Gaussian-like distributions. We approximate a given distribution of reflected light L using a set of spherical functions $\hat{L}$. We used the "von Mises-Fisher" (vMF) distribution, but similar distributions like the spherical Gaussian are alternatives.

The vMF distribution has a mean direction μ and a concentration κ:

$$s(\omega) = \frac{\kappa}{2\pi} e^{\kappa(\mu \cdot \omega)}$$

where ω is the direction. This formulation is an approximation to the spherical 3d vMF distribution with sufficient accuracy for $\kappa > 2$ [Han+07].

A set of k lobes is the sum of their respective vMF distributions:

$$\hat{L}(\omega) = \sum_{i=1}^{k} s(\omega).$$

To find the distribution of light reflected from a surfel, it would now be possible to estimate κ directly from the BRDF of the underlying rendering system which usually already has a coefficient for the concentration of a glossy BRDF (for example the Phong exponent) and set μ to the direction of perfect reflection. However, this would mean that this technique needed intrinsic knowledge of the underlying rendering system as well as a constraint to possible BRDFs. Therefore, in a first step, we record the distribution L at each

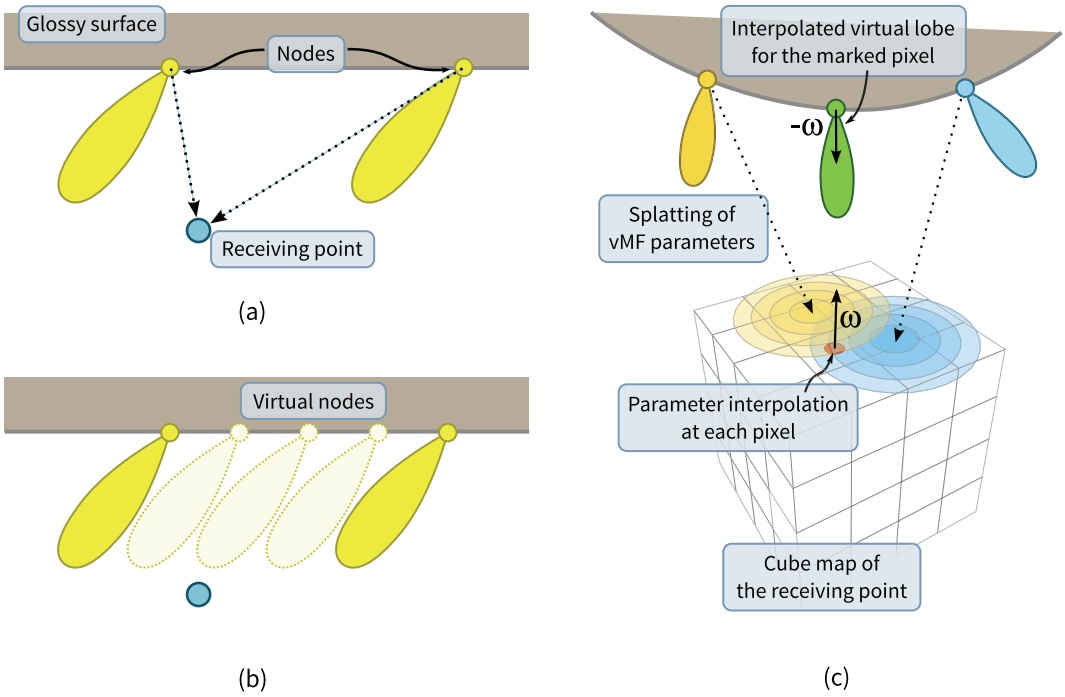


Figure 6.11: Left: **(a)** Evaluating the contributed energy of the two nodes in the direction of the receiving point directly would give nearly no contribution at all since the directions are outside of the respective lobes. **(b)** Instead of splatting the color contribution, the parameters of the lobe are splatted and interpolated over the pixels. The result resembles a denser (towards continuous) sampling of the points. **Right**: The parameter splatting and interpolation on the cube map.

surfel in a cube map of a resolution high enough to capture the frequencies contained in the BRDF responses. The set of k vMF lobes to approximate the distribution is then fitted to the data. Here, we employ a technique similar to Laurijssen et al. [Lau+11] who merge multiple vMF lobes into a combined multimodal distribution using Expectation Maximization (EM). Instead of several vMF lobes, we have a spherical distribution L onto which we fit, in a similar manner, k vMF lobes, resulting in the set of lobes $\hat{L}$. The parameters for the R, G and B channel are stored separately. The cube map has typical resolution of $6 \times 32 \times 32$, but the fitting is completely done in the directional domain. We found, that the simple, almost-Gaussian structure of radiance functions is fitted usually in only a few EM iteration steps.

To avoid approximation errors during the construction of the PBGI tree, we always sum up the actual distributions (i. e., the cube maps) and replace them with the set of vMF lobes afterwards. As mentioned before, keeping high-frequency distributions in full resolution in memory (i. e., the cube maps) for the whole tree is not possible. Therefore, we employ the same technique as in QPBGI by summing up on the fly (see Sec. 6.5.2).

Rendering with vMF lobes

In PBGI, the indirect illumination component is obtained by using a cube map per receiving point onto which the color of all contributing nodes are splatted. These receiving buffers

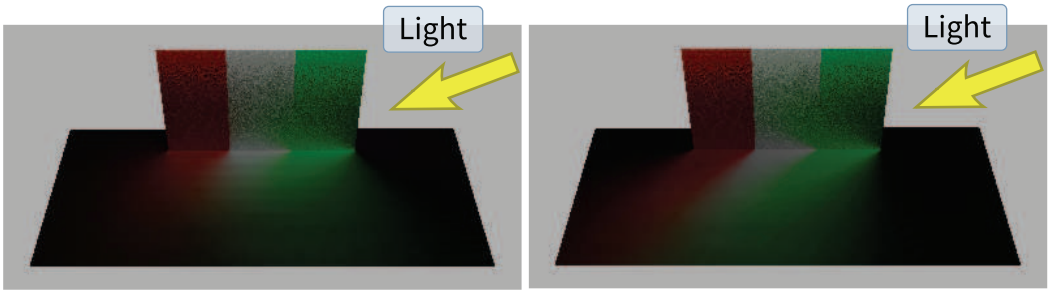


Figure 6.12: Comparison « SH » and « Set of vMF distributions ». A panel with a colored glossy surface is lighted by a light source to the right of the view point, the plane capturing the reflection has a diffuse BRDF. The result using Spherical Harmonics with 3 bands (left) show much less directionality than our method using the vMF distributions (right).

usually contain one channel per color (RGB) and a depth channel for visibility. The color of a splat is obtained by evaluating the color function stored in the node and is then directly splatted (and possibly interpolated) on the buffer. In the case of specularly reflected light, this method can lead to large errors due to the high frequency of the reflection function (i. e., the sharpness of the lobe) (see Fig. 6.11 (a)).

Following the idea of Phong Shading [Pho75] and Phong Splatting [Bot+04], we do not evaluate and splat the color of a given node. Instead, we splat the vMF parameters contained in each node with a radial Gaussian weighted kernel for each splat. For each pixel center of a receiving point's framebuffer (which corresponds to a direction ω on the sphere) the vMF parameters are reconstructed by interpolating the weighted parameters of all splats. This results in a *virtual* vMF lobe which would lie in the direction ω (assuming that the surface between the interpolated lobes is smooth). This lobe is then evaluated in the reverse direction $-\omega$ which points from the *virtual* lobe towards the receiving point (see Fig. 6.11 (c)).

Depending on the size of the splatted nodes and the variance parameter of the Gaussian kernel, the support for each splat is fairly small, so that effectively only a few (spatially close) splats are used for each interpolation.

Comparison to SH

We compared the difference in directional reflection of the proposed method to that of SH with 3 bands (see Fig. 6.12). Our method obviously captures the glossy BRDF of the colored panels better than the SH representation.

Discussion

While this method works well for plane surfaces, curved surfaces can still pose a problem in spite of the proposed splatting, resulting in blotchy artifacts. Another issue is the EM fitting time, since it scales linearly with the number of surfels and superlinear with the framebuffer resolution. One possibility to resolve this issue might be to exploit the similarity of neighboring surfels and nodes.

Currently, rendering systems using PBGI for illumination use it only for diffuse light transport (i. e., radiosity, color bleeding) and need to resort to other methods for rendering light transport effects over glossy surfaces. We think that combining glossy and diffuse light transport in one framework is an interesting direction of research and that our proposed method and preliminary results show one possible way towards that goal.

Part III

Conclusion

CHAPTER 7

CONCLUSION

In this thesis, we explored new data structures and the alternative use of existing structures in the context of rendering large, possibly animated 3d scenes. These structures are used to abstract and reorder the input data in such a way that it can be easier accessed and algorithms can run faster or at all on the abstracted data.

In most cases, the algorithm is closely linked to the data structures, in our case mostly derived from the geometry of the input scenes. The geometry and appearance of the input data is then used to specify the concrete values in the structures. We used this approach for photorealistic and non-photorealistic rendering, specifically in the creation of black and white images, the parameterization of animated lines and in the context of point-based global illumination.

From the work presented here we could see that, given a problem, finding a structure that is well adapted to the problem is of great importance. First of all, the formulation of a problem becomes easier if the structure is a good representation of the data contained in the problem. But this usually already needs to take into account the algorithm that is used for solving the problem. This is the point where it becomes obvious if the chosen structures and the algorithms are fitting well to the given problem. It is mostly the algorithm solving the problem which also guides the development or choice of the structures. The algorithm needs certain data from the input while the structures need to transform and extract that data in such a way that the resulting representation is well adapted for the algorithm. In some situations adapting the structures to the algorithm may be more appropriate while in other cases the reverse may be true. For example in QPBGI, the rendering algorithm was largely unchanged (i. e., one indirection was added for the index lookup), but the structure was heavily extended with a lookup table.

One problem that can arise, especially in NPR, is exposing the parameters of the structures and algorithms too directly to the user. For example in Binary Shading, even without having conducted user studies for the interface, letting the user directly control the parameters steering the Graphcut is not the ideal interface, also being the reason why the usage of our interface needs experience. More appropriately may have been the translation of these direct parameters into higher-level controls enabling a more directed control even for unexperienced users. While the amount of exposed control must always be assessed for the given method, we achieved this in the works on line parameterization and the quantization of spherical harmonics, where in both methods a single parameter is exposed to

the user, the trade-off between spatial and temporal coherence and the number of representatives, respectively.

Binary Shading Here, we explored the use of geometry and appearance data derived from the scene to create expressive black and white images with large regions. In rendering, the color of a pixel is usually derived directly from the properties of the underlying surface and the lighting, leading to local solutions. In our approach, we use an intermediary image space structure, that allows the combination of different attributes in such a way that the controllability of the outcome is enhanced, for example when compared to binary methods like thresholding. Especially the use of geometry features allows the generation of large, well defined regions. Using a Graphcut-based approach allows the creation of these results without the need of explicitely defining regions. The Graphcut also enables the creation of different types of results by adjusting the weighting of the edge capacities of the underlying graph. We obtain heavily stylized two-color results whose style can be controlled by the user by weighting the different terms and directly marking pixels by painting over the image. While the painting is very intuitive, our user-interface for the manually setting the weights may be coupled too tightly with the underlying structure. This results in the user needing to gather experience with the system before being able to create results in a targeted manner. We also showed the use of RGBN images in our system and possibly any input that provides separated geometry and appearance. While the results certainly are a matter of taste, we think that we created a system that can be used to create interesting results by an experienced user. We proposed a possible extensions of our system, among others to use more information from the input data and to achieve k-color renderings, a style often found in artistic renditions also using a heavily reduced palette just not as far as two colors.

Spatio-Temporal Analysis for Time-coherent Line Parameterization Similar to binary shading in its extreme reduction, we proposed a method to coherently texture animated contour lines. Previous approaches for parameterizing lines are usually real-time. This constrains their ability to look ahead when it comes to the development of the lines, notably topological events which, when unhandled, lead to discontinuities in the parameterization or, when handled in a backward-only manner, can lead to an undesired strong segmentation of the lines after a certain time of animation. We propose a method that solves the segmentation problem, though under the constraint of working only on completed animations. With this constraint in mind, we are able to do a complete analysis of the line development through time. For this, we use a space-time surface, which subsequently allows finding the topological events. While it is not possible to prevent the segmentation of the lines, in our work, the lines are segmented in a globally optimal way by re-using points of topological events. This allows keeping the number of segments low, avoiding the oversegmentation happening without this reuse. The use of the space-time surface, i. e., having a well established connectivity between the lines through time, equally allows the use of simple mesh and search algorithms on the surface. This is especially notable in the implementation when dealing with neighborhoods for example when finding the cut paths from one topological event to another or when building the linear system. While our system still has problems in some situations leading to unwanted parameterization behav-

ior, we have overcome the previous limitations of possibly strong line segmentation after lengthy animations at the cost of it being an offline method. Having gained some insight from our work, we then present a possible real-time approach that uses parameter value diffusion and subsequent reconstruction in a similar way as our non-realtime approach.

Quantized PBGI In the domain of photorealistic rendering, we proposed an extension to Point-Based Global Illumination in order to compress the large amounts of data generated during the preprocess. PBGI uses a hierarchical approximated scene representation, allowing fast access to geometry and the (indirect) light coming from that geometry. However, the storage of that indirect light poses a problem for large scenes, since it usually consists of spherical harmonics coefficients which have a large memory footprint. To allow the handling of larger scenes nonetheless, one solution is an out-of-core approach, streaming the data into memory as needed. This approach, however, uses the slow hard drive as an extension to the fast memory access, slowing down the access and making random accesses to the tree unfeasible. While the latter is not an issue with PBGI, since the tree traversal is very predictable, it may be problematic in other contexts. Our approach works directly on the data contained in the tree, creating representatives and replacing the node data by indices. While there is some amount of preprocessing, the actual changes to the PBGI rendering algorithm are a single additional indirection from the index, allowing for easy integration in existing frameworks. Beyond the context of PBGI, we think that the quantization using clustering of spherical harmonics coefficients may be of interest as well as the on-the-fly replacement, which allows the construction of compressed trees with arbitrary sized node data in a memory-efficient way.

Overall, we introduced several new techniques and extensions of methods for photorealistic and non-photorealistic rendering. Together with the possible extensions we proposed, we extended the range of possible depictions in NPR and the possible size of scenes for in-core rendering in the context of PBGI.

CHAPTER 8

PERSPECTIVES

This chapter is aimed to present a few ideas and avenues of research that could be of interest in rendering and the exploration and use of intermediary structures for that purpose.

Multi-view methods Currently, a number of methods exist that create small views of the scene distributed on its surfaces, like PBGI, (Ir-)Radiance Caching and others. While there is a lot of information in these views, only a small part of it is actually used in those methods and it seems that it should be possible to make more use of this information than for the calculation of indirect lighting alone.

Recently, a number of techniques have been proposed that use sparse data from an unbiased pathtracer to reconstruct information not directly contained in the data given by the pathtracer. In a fairly simple approach, Dammertz et al. [Dam+10] use bilateral smoothing steered by geometry properties to generate a smooth result from the noisy 1-sample-per-pixel pathtracer data. Lehtinen et al. [Leh+12] propose a method that uses only a few samples per pixel from a pathtracer as initial data to reconstruct secondary light rays not directly contained in the input data. These additional rays are then used to generate images of similar quality as if vastly more samples per pixel had been used but avoiding the costly casting of rays.

The information contained in the views of multi-view methods is much larger and consistent for a given view than the information gathered from a few pathtracer samples. An interesting avenue of investigation would be to analyse the data contained in these views more in-depth to improve on current algorithms when it comes to the number of needed views to reach a certain level of error but also to find new ways to use these views. There are several light transport problems which could benefit from the use of distributed views or, more generally, « spherical local attributes buffers ». In the previously mentioned methods, these views contain geometry and shading information, but could be extended to any sort of attribute needed to solve a given light transport problem. One example might be subsurface scattering (SSS) or even the general light transport through participating media, where these attribute buffers could store local density and phase function information. These buffers could be distributed, similar to radiance caching, in a lazy approach throughout the volume, and help to gather the light that moves through the medium. Depending on the properties of the attributes, mostly their rate of change throughout space,

the buffers could vary in their resolution and hierarchical structures could be used for fast approximations.

Another avenue in this analysis could be the temporal coherence of these views. In non-interactive rendering (e. g., production rendering in film), each frame is usually rendered by itself with little reuse in the temporal domain even though the complete animation is known beforehand. In view-based rendering of global illumination, the views are often interpolated only in the spatial domain. Disposing of the whole animation though might allow the use of not only spatial but also temporal interpolation and reuse of already calculated views. This could be used to improve the current spatial interpolation of views or to create views for which none exist at a given time, i. e., to interpolate new views not only from the spatial but also the temporal neighborhood of a point. This would also call for a structure that supports the generation of samples not only in the spatial domain as for example in (Ir-)radiance Caching but also distributes points taking into account the change of the geometry over time. Since the changes of the views stem not only from change of geometry but possibly any other change of scene properties like materials or light, it might be necessary to take those into account as well when also sampling in the spatial domain. One idea would be to employ heuristics similar to those used by Sen and Darabi [SD12] that allow taking into account changes in multiple dimensions. Enforcing temporal coherence of the samples, i. e., knowing the relation of samples in the temporal domain, might allow to further improve their interpolation.

Stylized Rendering One topic that is very prominent especially in stylized rendering is temporal coherence. The reason is that many methods process the data in such a way that the functions relating the input with the output (i. e., the image) are not necessarily smooth even though the input data might be. As an example, this can be observed in the work presented in Chap. 3, where the Graphcut can vary strongly from small changes in the input. Bénard et al. [Bén+11] published a state-of-the-art report on temporal coherence focusing on stylized animations and conclude the need for more, especially perceptive, evaluation of NPR results. They mention the need of unified assessment criteria for temporal coherence focusing on human perception where currently only very few attempts exist, most of them based on per-pixel differences or singular user studies. While the temporal coherence is largely dependent on the underlying algorithm, very little research has been conducted in finding common factors to predict temporal coherence or assess it. This topic especially might be of interest to see if it is possible to find ways of generating temporal coherence independently or with very little knowledge about the underlying algorithm, possibly by analyzing screen-space properties in the results.

In the same category of temporal coherence, the development and implementation of a general interface for applying and propagating rendering styles over time might be one step into that direction of abstracting from the underlying algorithms. For line drawings, this could be an interface that allows an artist to set textures or directly paint onto lines and a subsequent automatic propagation of this input through time. To help with the analysis of the temporal coherence, this interface would need to allow easy inspection of all facets of temporal coherence, in screen-space and world-space. Possibly, such an interface could unify several different algorithms without important changes to its workflow, similar to rtsc [Rus] which incorporates several different line drawing techniques.

Finally and in hindsight, I think it is important to also investigate methods of stylized rendering that go beyond the simulation of manual techniques. While it may seem appealing at first to be able to render entire computer games or videos in a painting or drawing style of well known artists, the possibilities offered by digital methods are much larger than this. This is admittedly not a mere scientific perspective, but NPR already incorporates to a large part also artistic decisions made by the authors of such methods that the exploration of a wider space of possibilities of NPR is well within the bounds of computer graphics. The step from the current state of NPR towards the one described above might be similar of that from photorealistic rendering to NPR indicated in Sec. 1.3, but here going from the simulation of manual methods towards methods unique to the digital domain.

Annex

OTHER WORK

Layered Volumetric Meshing

For modeling as for parameterization, well-structured meshes are very important. When a mesh has an easily and intuitively understandable surface structure, an artist can easily modify and extend it. It also simplifies parameterization which in turn allows for easier texturing. In the domain of surface meshes, good structures (often hand-made from artists) are made from "well-behaved" quads, i. e., quadrangular, roughly plane and rectangular polygons.

In volumetric meshing though, usually tetrahedral meshes are used, since they can readily be created from any given triangular mesh [Tou+09]. Tetrahedrons are the most simple volumetric primitive and also form a natural extension of a surface consisting of triangles. Unfortunately, tetrahedral meshing methods fill the volume with tetrahedrons in such a way that the resulting structure is no longer easily understandable by a human user, and thus, easy modifiability is not given.

Starting from the assumption, that meshes modeled by artists use quads wherever possible for easy processing, we propose a method to mesh and parameterize the interior (and to a certain extent the exterior) of such meshes. In our approach, we contract the mesh to a skeletal representation, all the while recording the trajectory of the vertices of the mesh from the surface towards the skeleton. The mesh contraction uses the gradient field derived from the normals of the surface. Each vertex on the surface independently follows in small steps the gradient in the opposite direction (i. e., towards the inner). The variance of the gradient field increases towards the inner and has its maximum at the desired skeleton. We use the variance as a stopping criterion for the iteration once we hit the maximum.

Given the trajectories of the vertices, layers are introduced at certain (possibly user-defined) intervals, resulting in a number of inset surfaces between the actual mesh surface and the skeleton Fig. 1. Since the gradient field is also defined on the outside of the mesh, the trajectories can, up to a certain extent, also be constructed towards the outside of the mesh, resulting in outset surfaces.

The resulting trajectories usually expose a few problems, mostly intersecting each other, deforming the quads on the layers. In order to achieve the desired quad properties on each inner layer, the trajectories are optimized taking into account the position of the points forming the trajectories as well as their position on the layers Fig. 2.

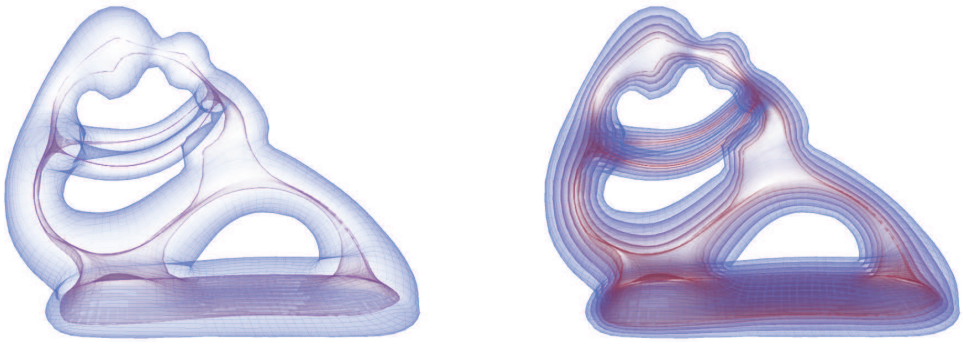


Figure 1: Left: Starting from the base mesh (blue), the normal gradient field is used to contract the mesh towards the skeleton (red), resembling the medial axis. **Right**: By recording the trajectory of each vertex during the contraction, layers or inset surfaces can be constructed.

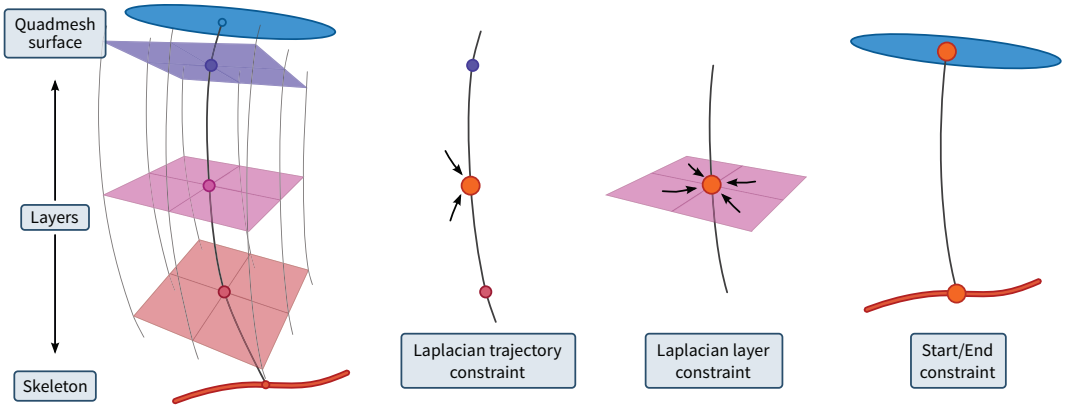


Figure 2: Layer and trajectory optimization. The initial trajectories show several flaws which necessitate an optimization step. An energy is formulated with soft constraints for each trajectory vertex (in orange) to move it in the center of its predecessor and successor on the trajectory, i. e., vertical neighborhood (« Laplacian trajectory constraint ») as well as into the center of the neighborhood on its layer, i. e., horizontal neighborhood (« Laplacian layer constraint »). The start and end of each trajectory are fixed on the original vertex position on the base mesh surface and the skeleton respectively. The resulting optimized trajectories give well-shaped volume elements.

The result is a volumetric, hexahedral representation of the mesh that can easily be understood due to the use of boxes stacked on layers. Fig. 3 When the base mesh is parameterized, this parameterization can also be used for the interior. Together with a parameterization of the trajectories, the interior volume of the mesh is completely parameterized.

While the algorithm works equally well for triangular meshes, extruded triangles are not as intuitively understandable as boxes. While the trajectories get closer to the skeleton (i. e., their end at the inside), the volume elements (be it triangular prisms or hexahedrons) start to degenerate. One solution could be to stop the gradient walk way before the maximum variance and replace the innermost volume elements with more isotropic elements. On the other hand, such an approach would also slightly hinder the easy understanding of that part of the volume.

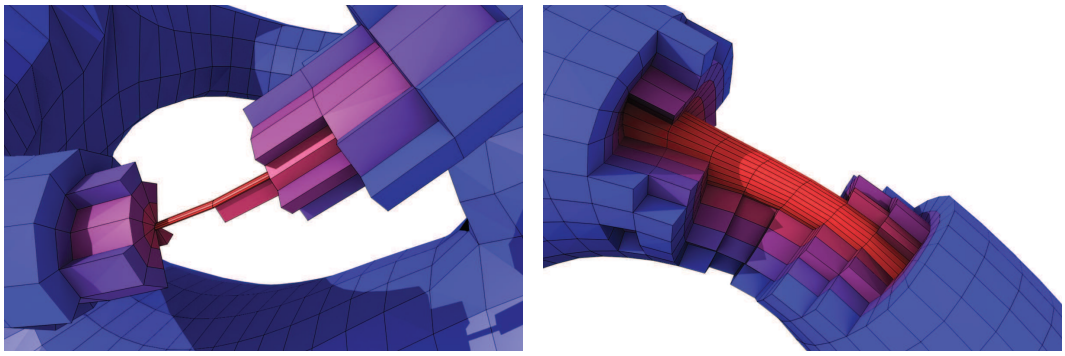


Figure 3: Inner structure. The volumetric representation consists of mostly well-formed hexahedrons (depending on the base mesh and the actual geometry) with a well-defined neighborhood.

In combination with the exterior layers (the outset surfaces), this can be used for example for shell mapping [Por+05]. Another application is volumetric texturing. Using the layers, an artist can texture the volume on each layer separately. The intuitive structure allows to easily find the correspondences between each layers. A continuous volumetric texturing function could then use interpolation of the textures layers to render slices. When using density instead of color values, the density of participating media could be designed. From density values, geometry synthesis would be another possibility: by defining a threshold and using level-set methods, a surface inside the volume could be reconstructed.

Analytic Curve Skeletons for 3D Surface Modeling and Processing

This work proposes a method for the analytical parameterization of a surface mesh using a new curve skeleton model designed for surface modeling and processing. This skeleton is defined as the geometrical integration of a piecewise harmonic parameterization defined over a disk-cylinder surface decomposition. This decomposition is computed using a progressive Region Graph reduction based on both geometric and topological criteria which can be iteratively optimized to improve region boundaries. The skeleton has an analytical form with regularity inherited from the surface one. Such a form offers well-defined surface-skeleton and skeleton-surface projections. The resulting skeleton satisfies quality criteria which are relevant for skeleton-based modeling and processing. Applications that benefit from our skeleton model include local thickness editing, inset surface creation for shell mapping, as well as a new mid-scale feature preserving smoothing.

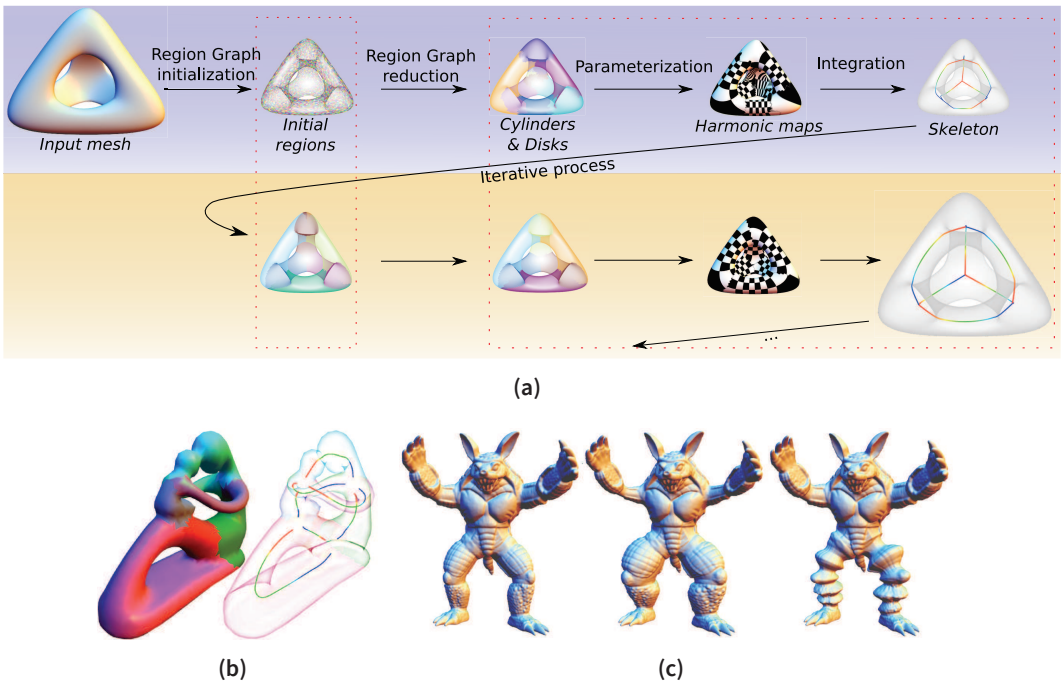


Figure 4: **(a)** Overview « Analytic Curve Skeletons » **(b)** The resulting skeleton with the parameterization displayed on the bones and the corresponding segmentation. **(c)** A possible application, using the parameterization to change the width of the base mesh (left) along the cylinders, preserving local features.

PUBLICATIONS

Publications

- J.-M. Thiery, B. Buchholz, J. Tierny, and T. Boubekeur. « Analytic Curve Skeletons for 3D Surface Modeling and Processing ». *In: Computer Graphics Forum (Proc. Pacific Graphics 2012) (2012)*
- B. Buchholz and T. Boubekeur. « Quantized Point-Based Global Illumination ». In: *Computer Graphics Forum*. Vol. 31. 4. Wiley Online Library. 2012, pp. 1399–1405.
- B. Buchholz, N. Faraj, S. Paris, E. Eisemann, and T. Boubekeur. « Spatio- temporal Analysis for Parameterizing Animated Lines ». In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*. ACM. 2011, pp. 85–92.
- B. Buchholz, T. Boubekeur, D. DeCarlo, and M. Alexa. « Binary Shading Using Appearance and Geometry ». In: *Computer Graphics Forum*. Vol. 29. 6. Wiley Online Library. 2010, pp. 1981–1992.

Talks & Posters

- Bert Buchholz, Tamy Boubekeur, Sylvain Paris, Noura Faraj and Elmar Eisemann. "Parameterizing Animated Lines for Stylized Rendering". *ACM SIGGRAPH 2011 – Talk Program*
- Jean-Marc Thiery, Tamy Boubekeur, Bert Buchholz. "Analytical Skeleton from a Cylinders and Disks Decomposition". *Symposium on Geometry Processing (SGP) 2010 – Poster*

BIBLIOGRAPHY

[AG10] A. A. Gooch. « Towards mapping the field of non-photorealistic rendering ». In: *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*. NPAR '10. New York, NY, USA: ACM, 2010, pp. 159–164.

[Bar+06] P. Barla, J. Thollot, and L. Markosian. « X-Toon: An extended toon shader ». In: *International Symposium on Non-Photorealistic Animation and Rendering (NPAR)*. ACM, 2006.

[Bez+08] H. Bezerra, E. Eisemann, X. Décoret, and J. Thollot. « 3d dynamic grouping for guided stylization ». In: *Proceedings of the 6th international symposium on Non-photorealistic animation and rendering*. ACM. 2008, pp. 89–95.

[BJ01] Y. Y. Boykov and M. P. Jolly. « Interactive Graph Cuts for Optimal Boundary and Region Segmentation of Objects in N-D Images ». In: *ICCV*. 2001, I: 105–112.

[BK04] Y. Boykov and V. Kolmogorov. « An Experimental Comparison of Min-Cut/Max-Flow Algorithms for Energy Minimization in Vision ». In: *IEEE trans. on PAMI* 26.9 (Sept. 2004), pp. 1124–1137.

[Bot+04] M. Botsch, M. Spernat, and L. Kobbelt. « Phong splatting ». In: *SPBG'04 Symposium on Point-Based Graphics 2004*. The Eurographics Association. 2004, pp. 25–32.

[Boy+01] Y. Boykov, O. Veksler, and R. Zabih. « Fast Approximate Energy Minimization via Graph Cuts ». In: *IEEE Trans. Pattern Anal. Mach. Intell* 23.11 (2001), pp. 1222–1239.

[Bro+08] J. Bronson, P. Rheingans, and M. Olano. « Semi-Automatic Stencil Creation Through Error Minimization ». In: *Proc. NPAR*. 2008, pp. 31–37.

[Buc+10] B. Buchholz, T. Boubekeur, D. DeCarlo, and M. Alexa. « Binary Shading Using Appearance and Geometry ». In: *Computer Graphics Forum*. Vol. 29. 6. Wiley Online Library. 2010, pp. 1981–1992.

[Bun05] M. Bunnell. « GPU Gems 2 ». In: 2005. Chap. Dynamic ambient occlusion and indirect lighting, pp. 223–233.

[Bén+09] P. Bénard, A. Bousseau, and J. Thollot. « Dynamic Solid Textures for Real-Time Coherent Stylization ». In: *I3D*. 2009, pp. 121–127.

[Bén+10] P. Bénard, F. Cole, A. Golovinskiy, and A. Finkelstein. « Self-Similar Texture for Coherent Line Stylization ». In: *NPAR*. 2010.

[Bén+11] P. Bénard, A. Bousseau, and J. Thollot. « State-of-the-Art Report on Temporal Coherence for Stylized Animations ». Anglais. In: *Computer Graphics Forum* 30.8 (Dec. 2011), pp. 2367–2386.

[Bén+12] P. Bénard, L. Jingwan, F. Cole, A. Finkelstein, and J. Thollot. « Active Strokes: Coherent Line Stylization for Animated 3D Models ». In: *NPAR 2012 - 10th International Symposium on Non-photorealistic Animation and Rendering*. ACM, June 2012, pp. 37–46. URL: http://hal.inria.fr/hal-00693453.

[Che+09] Y. Chen, T. A. Davis, W. W. Hager, and S. Rajamanickam. « Algorithm 887: CHOLMOD, supernodal sparse Cholesky factorization and update/downdate ». In: *ACM Trans. Math. Software* 35.3 (2009).

[Chr08] P. Christensen. *Point-based approximate color bleeding*. Tech. rep. 08-01. Pixar, 2008.

[Col+08] F. Cole, A. Golovinskiy, A. Limpaecher, H. Barros, A. Finkelstein, T. Funkhouser, and S. Rusinkiewicz. « Where do people draw lines? » In: *ACM Transactions on Graphics (TOG)*. Vol. 27. 3. ACM. 2008, p. 88.

[Col+09] F. Cole, K. Sanik, D. DeCarlo, A. Finkelstein, T. Funkhouser, S. Rusinkiewicz, and M. Singh. « How well do line drawings depict shape? » In: *ACM Transactions on Graphics-TOG* 28.3 (2009), p. 28.

[Cun+03] M. Cunzi, J. Thollot, S. Paris, G. Debunne, J.-D. Gascuel, and F. Durand. « Dynamic Canvas for Non-Photorealistic Walkthroughs ». In: *Graphics Interface*. June 2003, pp. 121–130.

[Dam+10] H. Dammertz, D. Sewtz, J. Hanika, and H. Lensch. « Edge-avoiding a-trous wavelet transform for fast global illumination filtering ». In: *Proceedings of the Conference on High Performance Graphics*. Eurographics Association. 2010, pp. 67–75.

[DeC+03] D. DeCarlo, A. Finkelstein, S. Rusinkiewicz, and A. Santella. « Suggestive contours for conveying shape ». In: *ACM Trans. Graph.* 22.3 (2003), pp. 848–855.

[DeC+04] D. DeCarlo, A. Finkelstein, and S. Rusinkiewicz. « Interactive rendering of suggestive contours with temporal coherence ». In: *NPAR 2004*. June 2004, pp. 15–24.

[Dec96] P. Decaudin. *Cartoon-Looking Rendering of 3D-Scenes*. Tech. rep. 2919. INRIA, 1996.

[DR07] D. DeCarlo and S. Rusinkiewicz. « Highlight Lines for Conveying Shape ». In: *NPAR*. 2007, pp. 63–70.

[DS02] D. DeCarlo and A. Santella. « Stylization and abstraction of photographs ». In: *ACM Transactions on Graphics (TOG)*. Vol. 21. 3. ACM. 2002, pp. 769–776.

[Eis+08] E. Eisemann, H. Winnemöller, J. C. Hart, and D. Salesin. « Stylized Vector Art from 3D Models with Region Support ». In: *Computer Graphics Forum* 27.4 (2008), pp. 1199–1207.

[FF56] L. Ford and D. Fulkerson. « Maximal flow through a network ». In: *Canadian Journal of Mathematics* 8.3 (1956), pp. 399–404.

[FS76] R. W. Floyd and L. Steinberg. « An Adaptive Algorithm for Spatial Greyscale ». In: *Proc. of the Society for Information Display* 17.2 (1976), pp. 75–77.

[GCO06] R. Gal and D. Cohen-Or. « Salient geometric features for partial shape matching and similarity ». In: *ACM Transactions on Graphics* 25.1 (Jan. 2006), pp. 130–150.

[GD09] T. Glander and J. Döllner. « Abstract representations for interactive visualization of virtual 3D city models ». In: *Computers, Environment and Urban Systems* 33.5 (2009), pp. 375–387.

[Goo+02] B. Gooch, G. Coombe, and P. Shirley. « Artistic vision: painterly rendering using computer vision techniques ». In: *Proceedings of the 2nd international symposium on Non-photorealistic animation and rendering*. ACM. 2002, 83–ff.

[Goo+99] B. Gooch, P.-P. J. Sloan, A. Gooch, P. S. Shirley, and R. Riesenfeld. « Interactive Technical Illustration ». In: *1999 ACM Symposium on Interactive 3D Graphics*. Apr. 1999, pp. 31–38.

[Gor+84] C. Goral, K. Torrance, D. Greenberg, and B. Battaile. « Modeling the interaction of light between diffuse surfaces ». In: *ACM SIGGRAPH Computer Graphics*. Vol. 18. 3. ACM. 1984, pp. 213–222.

[GP07] M. Gross and H. Pfister, eds. *Point-Based Graphics*. Morgan Kaufmann Series on Computer Graphics, 2007.

[Gra+10] S. Grabli, E. Turquin, F. Durand, and F. X. Sillion. « Programmable Rendering of Line Drawing from 3D Scenes ». In: *ACM Transactions on Graphics* 29.2 (Mar. 2010), 18:1–18:20.

[Gre03] R. Green. « Spherical harmonic lighting: The gritty details ». In: *Archives of the Game Developers Conference*. Vol. 5. 2003.

[Hac+08a] T. Hachisuka, W. Jarosz, R. Weistroffer, K. Dale, G. Humphreys, M. Zwicker, and H. Jensen. « Multidimensional adaptive sampling and reconstruction for ray tracing ». In: *ACM Transactions on Graphics (TOG)*. Vol. 27. 3. ACM. 2008, p. 33.

[Hac+08b] T. Hachisuka, S. Ogaki, and H. W. Jensen. « Progressive photon mapping ». In: *ACM SIGGRAPH Asia 2008 papers*. SIGGRAPH Asia '08. Singapore: ACM, 2008, 130:1–130:8.

[Hal64] J. Halton. « Algorithm 247: Radical-inverse quasi-random point sequence ». In: *Communications of the ACM* 7.12 (1964), pp. 701–702.

[Han+07] C. Han, B. Sun, R. Ramamoorthi, and E. Grinspun. « Frequency domain normal map filtering ». In: *ACM Trans. Graph.* 26 (3 2007).

[Her+01] A. Hertzmann, C. Jacobs, N. Oliver, B. Curless, and D. Salesin. « Image analogies ». In: *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM. 2001, pp. 327–340.

[HJ09] T. Hachisuka and H. Jensen. « Stochastic progressive photon mapping ». In: *ACM Transactions on Graphics (TOG)* 28.5 (2009), p. 141.

[Hol+11] M. Holländer, T. Ritschel, E. Eisemann, and T. Boubekeur. « ManyLoDs: Parallel Many-View Level-of-Detail Selection for Real-Time Global Illumination ». In: *Comp. Graph. Forum (Proc. EGSR)* 30.4 (2011), pp. 1233–1240.

[HZ00] A. Hertzmann and D. Zorin. « Illustrating smooth surfaces ». In: *Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 2000, pp. 517–526.

[Jen96] H. Jensen. « Global illumination using photon maps ». In: *Rendering Techniques* 96 (1996), pp. 21–30.

[Jud+07] T. Judd, F. Durand, and E. Adelson. « Apparent Ridges for Line Drawing ». In: *ACM Transactions on Graphics* 26.3 (July 2007), 19:1–19:7.

[Kaj86] J. Kajiya. « The rendering equation ». In: *ACM SIGGRAPH Computer Graphics* 20.4 (1986), pp. 143–150.

[Kal+02] R. D. Kalnins, L. Markosian, B. J. Meier, M. A. Kowalski, J. C. Lee, P. L. Davidson, M. Webb, J. F. Hughes, and A. Finkelstein. « WYSIWYG NPR: Drawing Strokes Directly on 3D Models ». In: *ACM Transactions on Graphics* 21.3 (July 2002), pp. 755–762.

[Kal+03] R. D. Kalnins, P. L. Davidson, L. Markosian, and A. Finkelstein. « Coherent Stylized Silhouettes ». In: *ACM Transactions on Graphics* 22.3 (July 2003), pp. 856–861.

[Kal04] R. D. Kalnins. « Interactive stylization for stroke-based rendering of 3D animation ». PhD thesis. Princeton University, 2004.

[Kan+09] H. Kang, S. Lee, and C. Chui. « Flow-based image abstraction ». In: *Visualization and Computer Graphics, IEEE Transactions on* 15.1 (2009), pp. 62–76.

[Kel97] A. Keller. « Instant radiosity ». In: *Proceedings of the 24th annual conference on Computer graphics and interactive techniques*. ACM Press/Addison-Wesley Publishing Co. 1997, pp. 49–56.

[KI86] J. Kittler and J. Illingworth. « Minimum error thresholding ». In: *Pattern Recognition* 19.1 (1986), pp. 41 –47.

[Kim+08] Y. Kim, J. Yu, X. Yu, and S. Lee. « Line-art Illustration of Dynamic and Specular Surfaces ». In: *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)* 27.5 (2008).

[Kir+83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. « Optimization by simulated annealing ». In: *Science* 220 (1983), pp. 671–680.

[Kon+11] J. Kontkanen, E. Tabellion, and R. S. Overbeck. « Coherent Out-of-Core Point-Based Global Illumination ». In: *Comp. Graph. Forum (Proc. EGSR)* 30.4 (2011), pp. 1353–1360.

[Kri+05] J. Krivanek, P. Gautron, S. Pattanaik, and K. Bouatouch. « Radiance caching for efficient global illumination computation ». In: *Visualization and Computer Graphics, IEEE Transactions on* 11.5 (2005), pp. 550–561.

[KZ93] M. Kamel and A. Zhao. « Extraction of Binary Character/Graphics Images from Grayscale Document Images ». In: *Graphical Models and Image Processing* 55.3 (May 1993), pp. 203–217.

[Lau+11] J. Laurijssen, R. Wang, P. Dutré, and B. J. Brown. « Fast Estimation and Rendering of Indirect Highlights ». In: *Comp. Graph. Forum (Proc. EGSR)* (2011).

[Lee+05] C. H. Lee, A. Varshney, and D. W. Jacobs. « Mesh saliency ». In: *ACM Trans. on Graph.* 24.3 (Aug. 2005), pp. 659–666.

[Lee+07] Y. Lee, L. Markosian, S. Lee, and J. F. Hughes. « Line drawings via abstracted shading ». In: *ACM Transactions on Graphics* 26.3 (July 2007), 18:1–18:5.

[Lee+10] H. Lee, S. Seo, S. Ryoo, and K. Yoon. « Directional texture transfer ». In: *Proceedings of the 8th International Symposium on Non-Photorealistic Animation and Rendering*. ACM. 2010, pp. 43–48.

[Lee+90] S. U. Lee, S. Y. Chung, and R. H. Park. « A Comparitive Performance Study of Several Global Thresholding Techniques for Segmentation ». In: *Comp. Vis., Gfx, & Im. Proc.* 52 (1990), pp. 171–190.

[Leh+12] J. Lehtinen, T. Aila, S. Laine, and F. Durand. « Reconstructing the indirect light field for global illumination ». In: *ACM Transactions on Graphics (TOG)* 31.4 (2012), p. 51.

[LL98] C. K. Leung and F. K. Lam. « Maximum Segmented Image Information Thresholding ». In: *GMIP* 60.1 (Jan. 1998), pp. 57–76.

[Llo82] S. P. Lloyd. « Least squares quantization in PCM ». In: *IEEE Transactions on Information Theory* 28.2 (1982), pp. 129–137.

[Lu+10] J. Lu, P. Sander, and A. Finkelstein. « Interactive painterly stylization of images, videos and 3D animations ». In: *Proceedings of the 2010 ACM SIGGRAPH symposium on Interactive 3D Graphics and Games*. ACM. 2010, pp. 127–134.

[MG08] D. Mould and K. Grant. « Stylized Black and White Images from Photographs ». In: *NPAR*. 2008, pp. 49–58.

[Ngu+12] C. H. Nguyen, T. Ritschel, K. Myszkowski, E. Eisemann, and H.-P. Seidel. « 3D Material Style Transfer ». In: *Computer Graphics Forum (Proc. EUROGRAPHICS 2012)* 2.31 (2012).

[OG11] S. Olsen and B. Gooch. « Image simplification and vectorization ». In: *Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Non-Photorealistic Animation and Rendering*. ACM. 2011, pp. 65–74.

[OH95] V. Ostromoukhov and R. D. Hersch. « Artistic Screening ». In: *Proc. of SIGGRAPH*. 1995, pp. 219–228.

[OH99] V. Ostromoukhov and R. D. Hersch. « Multi-Color and Artistic Dithering ». In: *Proc. of SIGGRAPH*. 1999, pp. 425–432.

[Par+10] S. Parker, J. Bigler, A. Dietrich, H. Friedrich, J. Hoberock, D. Luebke, D. McAllister, M. McGuire, K. Morley, A. Robison, et al. « Optix: A general purpose ray tracing engine ». In: *ACM Transactions on Graphics (TOG)* 29.4 (2010), p. 66.

[Pau+08] M. Pauly, N. J. Mitra, J. Wallner, H. Pottmann, and L. Guibas. « Discovering Structural Regularity in 3D Geometry ». In: *ACM Trans. Graph.* 27.3 (2008), 43:1–43:11.

[Pho75] B. Phong. « Illumination for computer generated pictures ». In: *Communications of the ACM* 18.6 (1975), pp. 311–317.

[Pin+12] C. Pindat, E. Pietriga, O. Chapuis, C. Puech, et al. « JellyLens: Content-Aware Adaptive Lenses ». In: *UIST-25th Symposium on User Interface Software and Technology-2012*. 2012.

[Por+05] S. Porumbescu, B. Budge, L. Feng, and K. Joy. « Shell maps ». In: *ACM Transactions on Graphics (TOG)*. Vol. 24. 3. ACM. 2005, pp. 626–633.

[Rie+11] C. Rieder, S. Palmer, F. Link, and H. Hahn. « A Shader Framework for Rapid Prototyping of GPU-Based Volume Rendering ». In: *Computer Graphics Forum*. Vol. 30. 3. Wiley Online Library. 2011, pp. 1031–1040.

[Rit+09] T. Ritschel, T. Engelhardt, T. Grosch, H. Seidel, J. Kautz, and C. Dachsbacher. « Micro-rendering for scalable, parallel final gathering ». In: *ACM Transactions on Graphics (TOG)*. Vol. 28. 5. ACM. 2009, p. 132.

[RL00] S. Rusinkiewicz and M. Levoy. « QSplat: A multiresolution point rendering system for large meshes ». In: *Proc. SIGGRAPH*. 2000, pp. 343–352.

[Rot+04] C. Rother, V. Kolmogorov, and A. Blake. « "GrabCut": interactive foreground extraction using iterated graph cuts ». In: *ACM Trans. Graph.* 23.3 (2004), pp. 309–314.

[Rus] S. Rusinkiewicz. *rtsc*. URL: http://gfx.cs.princeton.edu/proj/sugcon/index.html#software.

[Rus+06] S. Rusinkiewicz, M. Burns, and D. DeCarlo. « Exaggerated shading for depicting shape and detail ». In: *ACM Transactions on Graphics* 25.3 (July 2006), pp. 1199–1205.

[SC08] A. Shesh and B. Chen. « Efficient and Dynamic Simplification of Line Drawings ». In: *Comp. Graph. Forum* 27.2 (2008), pp. 537–545.

[Sch+12] D. Scherzer, C. Nguyen, T. Ritschel, and H. Seidel. « Pre-convolved Radiance Caching ». In: *Computer Graphics Forum*. Vol. 31. 4. Wiley Online Library. 2012, pp. 1391–1397.

[SD12] P. Sen and S. Darabi. « On filtering the noise from the random parameters in Monte Carlo rendering ». In: *ACM Trans. Graph.* 31.3 (June 2012), 18:1–18:15.

[Sel03] P. Selinger. « Potrace: a polygon-based tracing algorithm ». In: *PDF on Website* (2003).

[Slo+02] P.-P. Sloan, J. Kautz, and J. Snyder. « Precomputed Radiance Transfer for Real-Time Rendering in Dynamic, Low-Frequency Lighting Environments ». In: *Trans. Graph.* 21.3 (2002), pp. 527–536.

[Slo+03] P.-P. Sloan, J. Hall, J. Hart, and J. Snyder. « Clustered Principal Components for Precomputed Radiance Transfer ». In: *ACM Trans. Graph.* 22.3 (2003), pp. 382–391.

[Spi+06] M. Spindler, N. Röber, R. Döring, and M. Masuch. « Enhanced Cartoon and Comic Rendering ». In: *Proc. of Eurographics Short Papers*. 2006, pp. 141–144.

[Str+08] M. Stroila, E. Eisemann, and J. Hart. « Clip Art Rendering of Smooth Isosurfaces ». In: *IEEE TVCG* 14.1 (2008), pp. 135–145.

[SZ03] J. Sivic and A. Zisserman. « Video Google: a text retrieval approach to object matching in videos ». In: *Proc. Int'l Conf. Computer Vision*. 2003, pp. 1470–1477.

[TF+06] C. Toler-Franklin, A. Finkelstein, and S. Rusinkiewicz. « Illustration of Complex Real-World Objects using Images with Normals ». In: *NPAR*. 2006, pp. 111–119.

[Tou+09] J. Tournois, C. Wormser, P. Alliez, and M. Desbrun. « Interleaving Delaunay refinement and optimization for practical isotropic tetrahedron mesh generation ». In: *ACM Transactions on Graphics (TOG)*. Vol. 28. 3. ACM. 2009, p. 75.

[Tra+08] M. Trapp, T. Glander, H. Buchholz, and J. Dolner. « 3D generalization lenses for interactive focus + context visualization of virtual city models ». In: *Information Visualisation, 2008. IV'08. 12th International Conference*. IEEE. 2008, pp. 356–361.

[Uli87] R. Ulichney. *Digital Halftoning*. MIT Press, 1987.

[Van+07] D. Vanderhaeghe, P. Barla, J. Thollot, F. Sillion, et al. « Dynamic point distribution for stroke-based rendering ». In: *Eurographics Symposium on Rendering, Rendering Techniques 2007*. 2007, pp. 139–146.

[Vea97] E. Veach. « Robust Monte Carlo methods for light transport simulation ». PhD thesis. Stanford University, 1997.

[Ver+08] R. Vergne, P. Barla, X. Granier, and C. Schlick. « Apparent relief: a shape descriptor for stylized shading ». In: *Proc. of NPAR*. ACM, 2008, pp. 23–29.

[Ver+09] R. Vergne, R. Pacanowski, P. Barla, X. Granier, and C. Schlick. « Light warping for enhanced surface depiction ». In: *ACM Trans. Graph.* 28.3 (2009), pp. 1–8.

[Wal+05] B. Walter, S. Fernandez, A. Arbree, K. Bala, M. Donikian, and D. Greenberg. « Lightcuts: a scalable approach to illumination ». In: *ACM Transactions on Graphics (TOG)*. Vol. 24. 3. ACM. 2005, pp. 1098–1107.

[Wan+04] J. Wang, Y. Xu, H.-Y. Shum, and M. F. Cohen. « Video tooning ». In: *ACM Trans. on Graph.* 23.3 (2004), pp. 574–583.

[Wan+08] H. Wang, Y. Wexler, E. Ofek, and H. Hoppe. « Factoring repeated content within and among images ». In: *ACM Trans. Graph.* 27.3 (2008), 14:1–14:10.

[War+88] G. Ward, F. Rubinstein, and R. Clear. « A ray tracing solution for diffuse interreflection ». In: *ACM SIGGRAPH Computer Graphics*. Vol. 22. 4. ACM. 1988, pp. 85–92.

[Wen+06] F. Wen, Q. Luan, L. Liang, Y. Xu, and H. Shum. « Color sketch generation ». In: *Proceedings of the 4th international symposium on Non-photorealistic animation and rendering*. ACM. 2006, pp. 47–54.

[WH92] G. Ward and P. Heckbert. « Irradiance Gradients ». In: *Third Eurographics Workshop on Rendering*. Vol. 8598. Alan Chalmers and Derek Paddon. 1992.

[Whi80] T. Whitted. « An improved illumination model for shaded display ». In: *Communications of the ACM* 23.6 (1980), pp. 343–349.

[Win+06] H. Winnemöller, S. Olsen, and B. Gooch. « Real-time video abstraction ». In: *ACM Transactions On Graphics (TOG)*. Vol. 25. 3. ACM. 2006, pp. 1221–1226.

[Win+07] H. Winnemöller, D. Feng, B. Gooch, and S. Suzuki. « Using NPR to evaluate perceptual shape cues in dynamic environments ». In: *Proceedings of the 5th international symposium on Non-photorealistic animation and rendering*. ACM. 2007, pp. 85–92.

[XK08] J. Xu and C. S. Kaplan. « Artistic Thresholding ». In: *Proc. of NPAR)*. 2008, pp. 39–47.

[Xu+07] J. Xu, C. S. Kaplan, and X. Mi. « Computer-Generated Papercutting ». In: *Pacific Graphics*. 2007, pp. 343–350.

[YB89] S. D. Yanowitz and A. M. Bruckstein. « A new method for image segmentation ». In: *Comp.Vis., Gfx & Im. Proc.* 46.1 (1989), pp. 82–95.

[Yee04] H. Yee. « A perceptual metric for production testing ». In: *Journal of Graphics Tools* 9.4 (2004), pp. 33–40.