B Aspect de surface réaliste

- *Représentation des matériaux*, Chapitre du livre *Visualisation* (collection *Information Géographique et Aménagement du territoire* Hermès)

- *Pattern-Based Texturing Revisited* (Siggraph'99)

- *Animating Lava Flows* (GI'99)
  NB : ma contribution correspond à la partie "rendu" (section 4).

Visualisation

Représentation des matériaux

Fabrice Neyret

06 juillet 2001

Table des matières

Représentation des matériaux

Fabrice Neyret[1]

Mots-clés : matériaux, matière, texture, couleur, illumination, réflectance

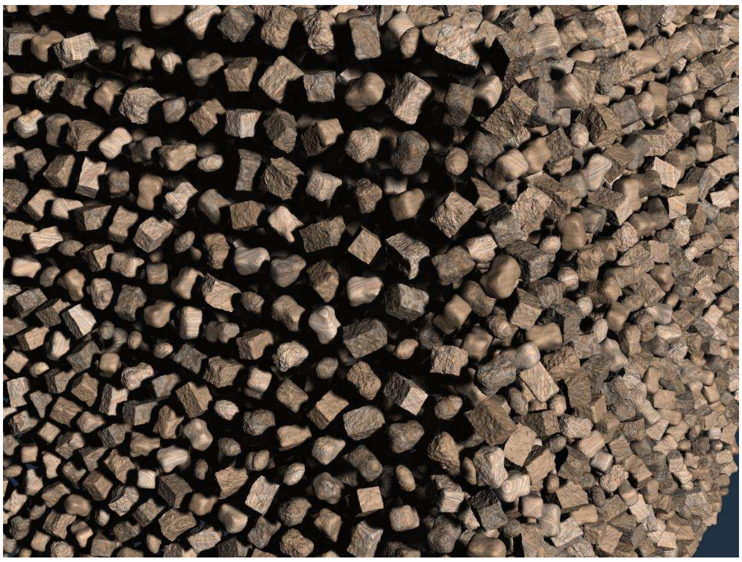

Figure 1 – Matériaux : une question d'échelle *(Image : Justin Legakis, MIT).*

[1]chargé de recherche au CNRS,
dans l'équipe iMAGIS du laboratoire GRAVIR/IMAG-INRIA,
INRIA Rhône-Alpes, ZIRST 655 avenue de l'Europe,
Montbonnot, 38334 Saint Ismier Cedex, FRANCE
`e-mail` : Fabrice.Neyret@imag.fr
www : http ://www-imagis.imag.fr/Membres/Fabrice.Neyret

1. Introduction et notions de base

Ce chapitre traite de la représentation des matériaux aux fins de visualisation. On s'intéresse donc ici à tout - mais seulement à - ce qui peut avoir une incidence directe ou indirecte sur l'apparence.

Qu'est ce qui caractérise *l'apparence* d'un objet, que celui-ci soit naturel ou manufacturé ? On peut distinguer sa *forme*, dans sa globalité et dans ses détails (i.e. sa géométrie), et le *matériau* dont il est fait ou recouvert.

La représentation et la construction des formes sont l'objet des techniques de modelage (ou *modeling*), dont on traite aux chapitres **NN**. La composante 'matériau', celle qui nous intéresse dans ce chapitre, comprend d'une part les *caractéristiques optiques* propre à la matière en présence, dans ses aspects intrinsèques (e.g. composition chimique initiale) et extrinsèques (état de surface, altérations mécaniques et chimique), et d'autre part les *variations locales de ces caractéristiques* (aspect de surface, granulosité, motifs, etc).

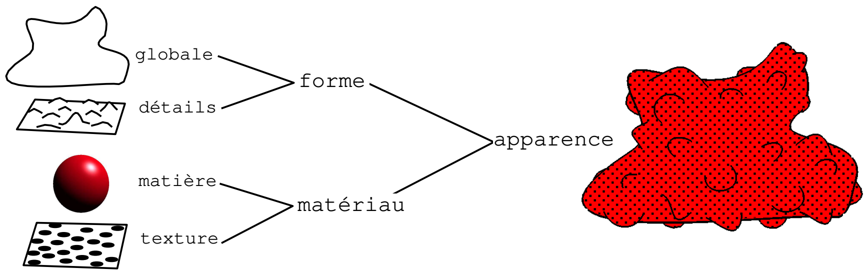

Figure 2 – Les diverses modalités de l' *apparence* d'un objet.

Ces différenciations sont en grande partie arbitraires : où s'arrêtent les détails géométriques, où commencent le grain et l'aspect de surface ? Quand faut-il considérer qu'une micro-altération (porosité, rugosité) est caractéristique d'une certaine matière, plutôt qu'une variation locale taillée dans une matière 'pure' et continue ? Concernant la première interrogation, c'est souvent une question de *connaissance* a priori, ou encore de *sens* que le designer veut distiller dans chaque aspect : les détails géométriques sont décrits un par un, tandis que le 'grain' est une notion plus statistique, décrite d'une façon plus globale, que l'on peut donc inclure parmi les caractéristiques d'un matériau. Concernant la seconde interrogation, i.e. pour ce qui ce qui relève du matériau, c'est essentiellement l'*échelle* qui fait la différence (cf figure 3) : on distingue les phénomènes qui interviennent *en dessous de la taille du pixel* de ceux qui interviennent *au dessus de la taille du pixel*, car les images numériques[2] ont une résolution finie, ce qui implique une taille minimale des détails observables. On range ainsi dans

[2]Tout comme la rétine humaine et les pellicules photographiques, d'ailleurs.

les ‘caractéristiques optiques’ ce qui procède de la première catégorie (niveau sub-pixel, couvrant les effets des échelles microscopiques jusqu’aux variations millimétriques comme la rugosité), et dans les ‘variations locales’ ce qui procède de la seconde catégorie (i dont les variations peuvent potentiellement se voir à l’écran).

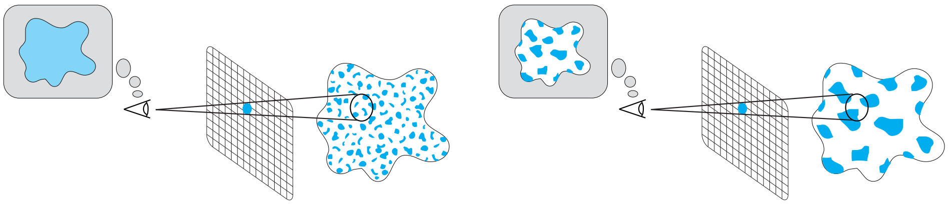

Figure 3 – Comme la résolution de l’image est finie, tous les éléments de la scènes qui figurent à l’intérieur du cône de vision passant par un pixel contribuent à former une valeur unique, la couleur du pixel. Si la taille apparente des motifs est inférieure à celle du pixel (*à gauche*), seul leur effet moyen est visible, aussi on les inclura plutôt dans le modèle de matière. Si elle est supérieure à la taille du pixel (*à droite*), ils induisent une fluctuation de la couleur d’un pixel à l’autre, et sont alors modélisés comme variation spatiale des paramètres du modèle de matière (ou *texture*).

Remarquons que ces considérations permettent d’adopter une acception très large de la notion de matériau (cf figure 4) : si l’échelle est celle d’une vue satellitaire, les reliefs terrestres peuvent être traités en matériau. Si le point de vue est situé à faible altitude, les reliefs relèvent de la géométrie, mais la forêt qui les recouvrent peut constituer un type de ‘matériau’. Si l’on est à proximité immédiate de la forêt, les arbres les plus proches ont une extension telle que leur forme globale doit être explicitée, par contre le feuillage peut encore être considéré en matériau. Si l’on est au pied de l’un des arbres, alors les feuilles ont potentiellement une taille apparente qui incite à les décrire géométriquement, et à les ‘habiller’ d’un matériau qui rende compte de la couleur, du grain et de la réflectivité de leur surface. Il sera donc économique[3], tant du point de vue du travail de spécification, du stockage des données, que du calcul du rendu, de faire des choix de représentation (i.e. d’outil) raisonnables vis à vis de l’échelle à laquelle les objets apparaissent. A noter que ces considérations sont à rapprocher de la notion de niveaux de détails, abordés au chapitre **NN**, qui visent à adapter dynamiquement la finesse de la représentation géométrique à la distance du point de vue.

[3] En outre le trop plein d’information se paie par un fléau récurrent dans les techniques numériques, l’*aliasing* : si plus d’un objet se trouve derrière un pixel donné, celui qui apparaîtra à l’écran risque fort sans précaution particulière d’être fixé au hasard. Pire encore, il a peu de chance d’être le même d’une image à l’autre lors d’une animation, ce qui engendre un grouillement peu agréable dans l’image. Les techniques d’anti-aliasing doivent sommer les contributions respectives des objets qui concourent à occuper ce pixel, ce qui a bien sûr un coût (pensez au million de feuilles d’un arbre dans le lointain). Choisir une représentation de plus grande échelle est alors équivalent à faire ce traitement dès la conception.

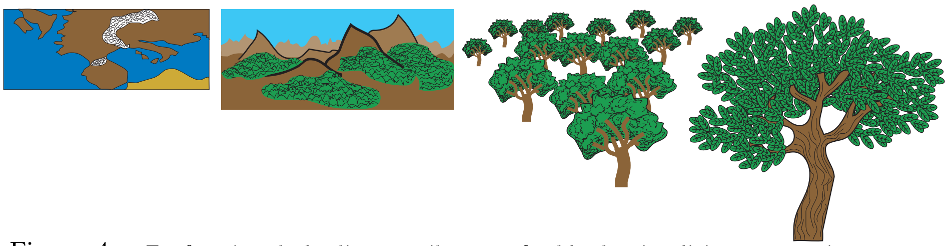

Figure 4 – En fonction de la distance, il est préférable de n'expliciter géométriquement que les objets de taille apparente non négligeable, et de représenter les échelles plus petites par un 'matériau' (en l'occurrence, de gauche à droite, la montagne, la forêt, le feuillage, la surface des feuilles).*(Images : Fabrice Neyret, iMAGIS / GRAVIR)*

Reprenons donc notre description de la représentation des matériaux, illustrée en pensant d'abord aux matériaux 'classiques' comme le bois, le crépis, la roche ou le métal, mais en gardant à l'esprit la généricité des concepts. Nous avons distingué les 'caractéristiques optiques' propres à la matière, ou du moins couvrant les phénomènes sub-pixel, et les 'variations locales' qui spécifient les modulations d'aspect qui interviennent à une échelle visible.

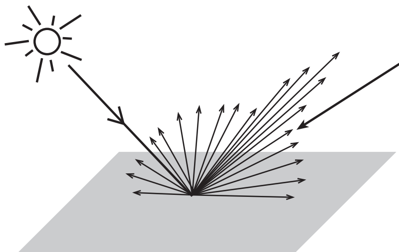

Figure 5 – La fonction de réflectance modélise la proportion de lumière réfléchie (dans la direction de l'observateur) en fonction de la position de la source de lumière.

Les caractéristiques optiques sont essentiellement constituées de la *fonction de réflectance*, qui donne la couleur et l'intensité lumineuse réfléchie vers l'observateur en un point de la surface (cf figure 5). On y adjoint parfois aussi la *transparence*. Divers modèles, empiriques ou issus de la physique, et aboutissant généralement à des formules analytiques, ont été proposés pour représenter la fonction de réflectance de matériaux génériques ; ils feront l'objet de la section 2. Ainsi, choisir une matière, c'est à dire spécifier les caractéristiques optiques, revient en pratique à opter pour l'un de ces modèles et à fixer ses paramètres.

Les 'variations locales d'aspect' permettent de passer d'une *matière* à un véritable *matériau*. Elles se spécifient sous forme de *textures*, pour lesquelles il faut préciser le paramètre optique que l'on module (e.g. la couleur diffuse), la façon dont on spécifie les variations (e.g. explicitement, à l'aide d'une image), et la manière d'appliquer ces variations sur la surface, ou *mapping* (i.e. méthode de tapissage de l'image sur la surface de l'objet à 'habiller'). Nous détaillerons ces trois points dans les sections 3.1, 3.2 et 4.

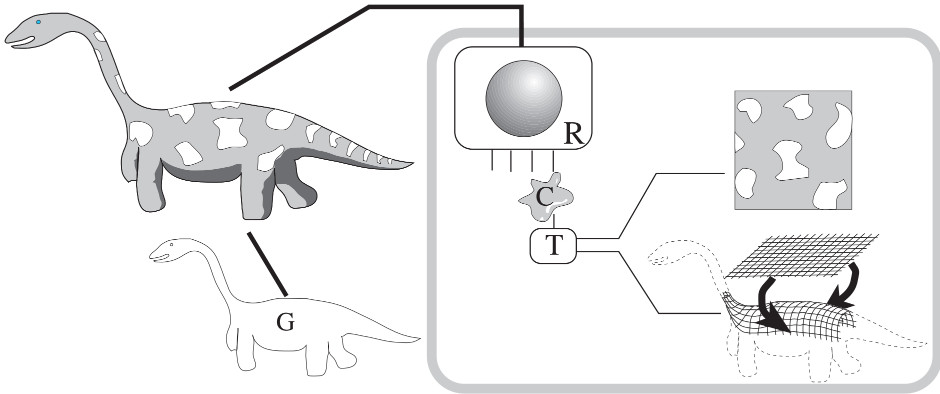

Figure 6 – Le matériau est constitué d'une *fonction de réflectance* (R), dont on fait varier certains *paramètres* (ici la couleur C) le long de la surface à l'aide d'une *texture*, laquelle est constituée d'une *description du contenu* (e.g. image) et d'une méthode d'application sur la surface (*mapping*).

Résumons-nous : on représente un matériau en spécifiant sa matière, c'est à dire sa **fonction de réflectance** et éventuellement sa **transparence**, et ses variations locales d'aspect à l'aide de **textures**, c'est à dire pour chaque attribut que l'on souhaite moduler, une **construction des variations** (procédure algorithmique, ou simple image) et un **mapping**.

Ces bases étant posées, nous verrons qu'elles servent aussi bien de support au cas trivial suggéré ci-dessus (e.g. image de rouille plaquée sur un objet), qu'aux techniques récentes qui définissent des matériaux particulièrement riches : textures en relief permettant de représenter la forêt sur les collines, corrosion et patine des surfaces rendant compte des effets du temps, etc. Nous décrirons quelques-unes d'entre elles dans la section 5.

Remarque :
les logiciels de rendu réaliste présents et à venir, utilisés pour l'illustration de projets ou les effets spéciaux, reposent entièrement sur leur programmation des différents modèles mentionnés plus haut. Les effets les plus aboutis leur sont donc potentiellement accessibles, à concurrence du temps et des ressources que l'on est prêt à investir. Inversement, les applications interactives, comme les jeux et les visualisateurs de scènes 3D, doivent s'appuyer sur des cartes graphiques, lesquelles ont des fonctionnalités autrement plus restreintes. A la fin de chaque section, nous mentionnerons les principes et les attributs qui restent accessibles dans le contexte de la visualisation accélérée par carte graphique (sachant bien cependant que ces dernières sont en constante évolution, et que les limites mentionnées sont appelées à être dépassées un jour ou l'autre).

2. Caractéristiques optiques d'une matière, ou *fonction de réflectance*

2.1. *Matériaux mats*

Le modèle de *Lambert*, issue de la physique, donne la fonction de réflectance la plus simple, laquelle décrit assez bien les surfaces mates de faible rugosité (e.g. mur en plâtre) : l'intensité réfléchie est égale au produit scalaire de la normale à la surface et de la direction de la source de lumière, ou zéro si le produit scalaire est négatif (quand le mur ne fait pas face à la lumière), multiplié par un coefficient de réflection diffuse (le mur est d'autant plus blanc que ce coefficient est proche de 1), et par l'intensité provenant de la source, ce qui s'exprime sous forme mathématique par [4] $I_r = k_d I_L(\overrightarrow{N}.\overrightarrow{L})\mathbb{1}_{\overrightarrow{N}.\overrightarrow{L}>0}$ (cf figure 7).

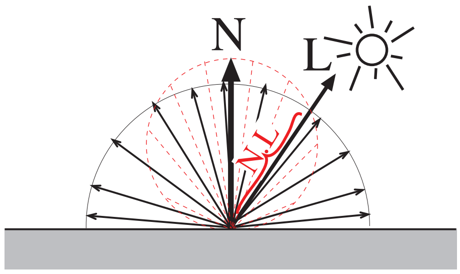

Figure 7 – Loi de Lambert, modélisant les matériaux mats.

Mais pour la synthèse d'images, on ne se contente pas d'intensités : on veut prendre en compte la couleur. On décompose généralement celle-ci selon trois composantes (rouge, verte, bleue), et l'on peut alors se figurer la couleur diffuse (extension de l'intensité diffuse k_d) comme un vecteur d'intensités $\overrightarrow{C_d} = (C_d^R, C_d^V, C_d^B)$, que l'on peut considérer comme un échantillonnage du spectre lumineux continu[5].

[4] **Notations :** dans la suite, les vecteurs $\overrightarrow{N}$, $\overrightarrow{L}$ et $\overrightarrow{V}$ désignent respectivement la normale à la surface, la direction de la lumière, et la direction de l'observateur. On utilisera aussi $\overrightarrow{R}$, indiquant la direction de la lumière réfléchie en miroir.
On note $\mathbb{1}_{cond}$ l'opérateur valant 1 si la condition est vraie et 0 si elle est fausse.

[5] Pour de nombreux matériaux, cette décomposition en trois composantes (justifiée par le seul fait que que le système visuel humain comporte trois types de filtres colorés) est insuffisante, ce qui peut se manifester de multiples façons. Tout d'abord plusieurs spectres différents peuvent donner la même couleur (une fois *perçue* par une rétine *humaine*, et sous *éclairage blanc*) : une peinture 'jaune' peut correspondre à un spectre de réflection concentré autour de la fréquence jaune (celle de l'arc-en-ciel), mais tout aussi bien à un spectre réfléchissant le rouge et le vert (rouge+vert=jaune en synthèse additive). Si l'éclairage est également 'jaune', il peut lui-même l'être de plusieurs façons différentes. On constate donc qu'il existe de multiples combinaisons de sources de lumière et de peintures qui, bien que paraissant chacune individuellement être toujours de la même couleur, engendrent une cou-

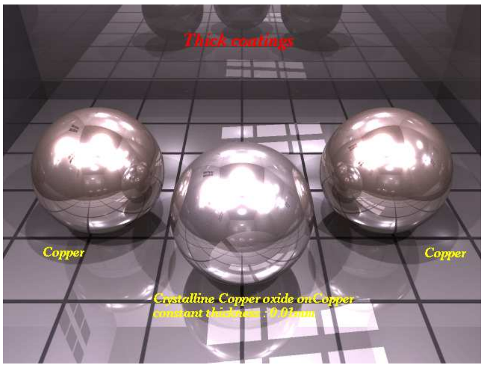

Figure 8 – Un exemple de modèle physique élaboré de matériaux. En particulier, le rendu de l'image simule finement le spectre de la lumière, permettant de rendre compte de la diversité de comportement de la source lumineuse et du matériau selon la longueur d'onde *(Image : Patrick Callet, ECP [CAL]).*

La formule devient alors $\overrightarrow{C_r} = \overrightarrow{C_d} I_L(\overrightarrow{N}.\overrightarrow{L})\mathbb{1}_{\overrightarrow{N}.\overrightarrow{L}>0}$, ce qui revient simplement à appliquer la première formule successivement pour les trois composantes R,V et B. Si la source lumineuse est elle-même colorée, il faut également remplacer I_L par un vecteur couleur, ce qui donne $C_r^\lambda = C_d^\lambda\ C_L^\lambda(\overrightarrow{N}.\overrightarrow{L})\mathbb{1}_{\overrightarrow{N}.\overrightarrow{L}>0}$ pour $\lambda =$ R,V,B , qui exprime le fait que la couleur perçue est le produit de la couleur propre et de la couleur de la source lumineuse : dans un tunnel dont l'éclairage ne comporte pas de composante bleue, l'apparence des objets illuminés (vêtements, images) est altérée par rapport à un éclairage naturel, car la composante bleue de leur teinte est 'éteinte'.

S'il y a plusieurs sources de lumière, il faut additionner les contributions de chacune. On ajoute de même généralement un terme constant qualifié d'*ambiant*, pour rendre compte de l'effet de la lumière diffuse environnante. La formule complète du modèle de Lambert est donc :
$C_r^\lambda = C_d^\lambda\ \ \sum_i C_{L_i}^\lambda(\overrightarrow{N}.\overrightarrow{L_i})\mathbb{1}_{\overrightarrow{N}.\overrightarrow{L_i}>0}\ +\ C_d^\lambda C_{L_a}^\lambda \quad$ pour $\lambda =$ R,V,B

Le seul paramètre du modèle de matière est donc la 'couleur propre' C_d de la surface, ou *couleur diffuse* (toutefois les divers logiciels et API laissent la possibilité de régler une *couleur ambiante* indépendamment de la couleur diffuse).

---

leur apparente chaque fois différente. D'autre part les éclairages réels (e.g. lampe à vapeur de sodium) peuvent fréquemment posséder un spectre constitué de quelques raies, toutes les autres fréquences étant éteintes. Ces raies dans l'illumination vont donc 'échantillonner' des positions très précises dans le spectre des matériaux, dont les valeurs ont très peu de chances de correspondre aux trois 'moyennes locales' que constituent les composantes R, V et B classiques. En conclusion, s'il ne s'agit que de produire des images, un graphiste saura doter les surfaces de couleurs produisant le résultat attendu. Par contre si l'on s'intéresse à des visualisations colorimétriques précises partant de données de matériaux réels, il faudra recourir à un logiciel de rendu *spectral*, capable de décomposer les spectres en un nombre adéquat de composantes (cf figure 8).

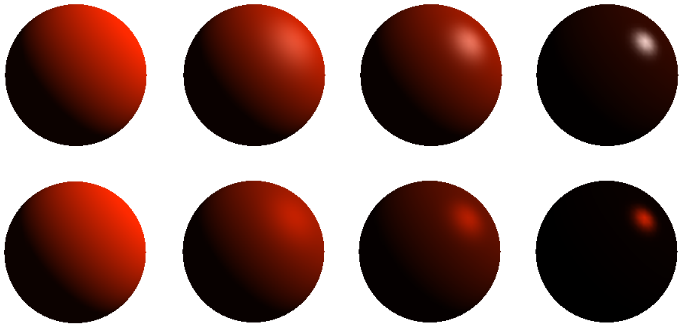

Figure 9 – De gauche à droite, modèle de Lambert, puis modèle de Phong avec un exposant spéculaire de plus en plus élevé. *En haut :* matériaux plastiques (spéculaire de la couleur de la source). *En Bas :* matériaux métalliques (spéculaire de la couleur du matériau).

2.2. *Matériaux brillants*

Cependant, toutes les surfaces ne sont pas mates. Pour rendre compte des surfaces brillantes, on utilise le modèle de *Phong* : celui-ci ajoute au modèle précédent un terme empirique visant à reproduire les reflets *spéculaires*, qui correspondent à la réflection en miroir - plus ou moins dépoli selon la rugosité de la surface - des sources lumineuses. Comme cette réflection en miroir est généralement due au revêtement superficiel de la surface (vernis, liant contenant les pigments, etc), sa couleur est indépendante de la couleur propre de celle-ci. Ce terme fait donc intervenir une *couleur spéculaire*, multipliée par le produit scalaire entre la direction de l'observateur et la direction des rayons issus de la source lumineuse et réfléchis en miroir (ou zéro si la surface ne fait pas face à la lumière). Le produit scalaire est élevé à une puissance inversement proportionnelle à la *rugosité* (cf figure 10) : si celle-ci est forte, le reflet est 'flou' et se transmet dans une large gamme de directions. Mais si elle est faible alors l'exposant est élevé, et le terme est significatif seulement si le produit scalaire est très proche de 1, c'est à dire si l'observateur est juste dans la direction des rayons réfléchis. Le terme supplémentaire s'exprime donc mathématiquement par $C_s^\lambda C_L^\lambda(\overrightarrow{V}.\overrightarrow{R})^\alpha \mathbb{1}_{\overrightarrow{V}.\overrightarrow{R}>0}$. On peut alors reproduire l'aspect d'une boule de billard rouge présentant un reflet blanc [6], cf figure 9. En pratique la taille de la tache spéculaire ne dépend pas que de la rugosité de la surface, mais aussi de la taille de l'objet lumineux vu en reflet (soleil, lampe). Les designers règlent donc essentiellement l'exposant spéculaire en fonction de l'effet qu'ils désirent obtenir. A noter que si les surfaces synthétiques ou vernies présentent effective-

[6]On remarquera que si l'on se déplace, la réflectance diffuse en un point donné reste constante, tandis que le reflet spéculaire change de position : la réflection spéculaire dépend fortement du point de vue.

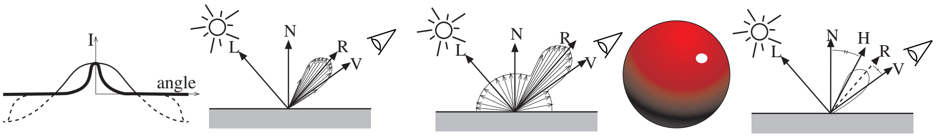

Figure 10 – Loi de Phong-Blinn, modélisant les matériaux brillants. *De gauche à droite* : effet de l'exposant sur le produit scalaire (en tracé fin, valeur du produit scalaire seul) ; lobe de réflection spéculaire dans la direction en 'miroir' de la lumière ; modèle de Phong complet intégrant diffus (Lambert) et spéculaire ; image obtenue avec ce modèle (couleur 'propre' -i.e. diffuse- en rouge, reflet spéculaire blanc dépendant du point de vue) ; variante de Blinn remplaçant $\vec{R}.\vec{V}$ par $\vec{N}.\vec{H}$, moins cher à calculer (on voit que les deux angles sont proportionnels).

ment une couleur spéculaire indépendante de la couleur diffuse, ce n'est pas le cas des surfaces métalliques pour lesquelles les deux couleurs sont identiques, à l'intensité près. On a alors $C_s = k_s C_d$, i.e. on ne contrôle plus qu'un *coefficient spéculaire*. Blinn a proposé un modèle permettant de choisir continuement entre une réflection 'plastique' ou 'métallique' au moyen d'un *coefficient de blancheur*, introduisant en outre une variante plus économique du terme séculaire : le lobe du reflet est obtenu par $(\overrightarrow{N}.\overrightarrow{H})^\alpha$, où $\overrightarrow{H}$ est le vecteur 'moitié' entre la direction de la lumière et la direction de l'observateur (que l'on peut considérer constant si la distance à la source lumineuse et à l'observateur sont grandes par rapport à la taille de l'objet). *Pour en savoir plus : [FOL 90, RUS].*

2.3. *Représentations avancées*

Les matériaux réels, même les plus courants, ont un comportement plus complexe que celui traduit par les modèles de Lambert et de Phong-Blinn, notamment aux incidences rasantes.

Des modèles plus fins ont donc été proposés pour améliorer la précision photométrique (e.g. couleur et intensité aux angles rasants), simuler les effets des micro-reliefs (dont l'orientation, les ombres portées et l'inter-réflexion peuvent avoir un effet très sensible sur l'intensité réfléchie, et même sur la couleur, notamment au bord des ombres et de la silhouette [7], cf figure 12), ou encore ceux qui résultent des interactions de la lumière avec la matière dans son voyage à l'intérieur de la couche superficielle de la surface (peinture, marbre, peau, cf figure 14). *Pour en savoir plus : [CAL 98, COO 82, GON 94, ORE 94, DOR 96b, HAN 93, WAR 92, ASH 00, HEI 00].*

[7] Voir à quel point la Lune ressemble peu à une sphère de plâtre, ou comme un revêtement en linoléum paraît mat à nos pieds et poli au bout du couloir.

Figure 11 – *En haut* : modèle physique (spectral) de matériaux à couches minces *(Images : Isabelle Icart, IGM / Université de Marne la Vallée). En bas* : extension du modèle de Lambert gérant mieux les incidences rasantes *(Image : Oren, Nayar, Columbia University [ORE 94])* ; modèle de pigments, avec effets de la rugosité, et de la finesse du pigment *(Images : Gondek, Meyer, Newman, University of Oregon [GON 94])*.

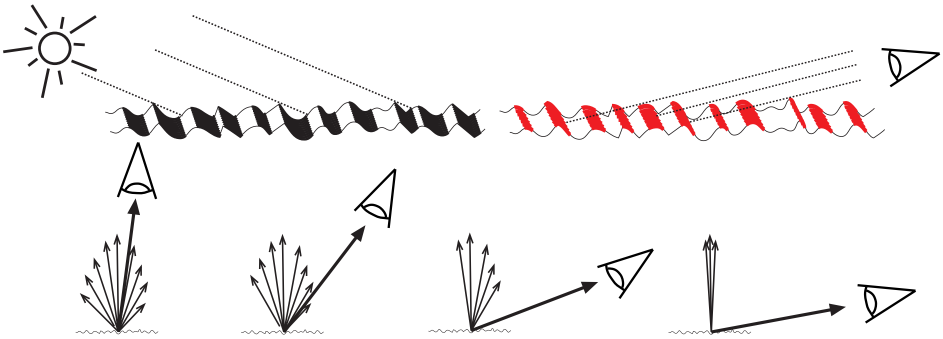

Figure 12 – Effet des micro-reliefs sur une surface apparemment plane. *En haut à gauche* : vue de dessus, la moitié de la surface se trouve à l'ombre de son propre relief. Même si celui-ci n'est pas directement visible, l'effet est important sur la réflectance moyenne de la surface. Par contre si l'observateur est dans la même direction que la source lumineuse, il ne verra au contraire que des parois très brillantes, produisant une réflectance moyenne bien plus élevée que ne l'aurait une surface réellement plane de même orientation. *En haut à droite et en bas* : selon le point de vue, seul le haut des anfractuosités est visible, ce qui entraîne un effet de censure des normales apparentes. Aux incidences rasantes seul le sommet des bosses est visible, et donc seules les normales égales à la normale moyenne de la surface interviennent dans l'illumination, laquelle est donc identique à celle d'une surface lisse. Cet effet de rugosité apparente dépendant de l'angle de vue est par exemple visible sur les longs couloirs revêtus de linoléum.

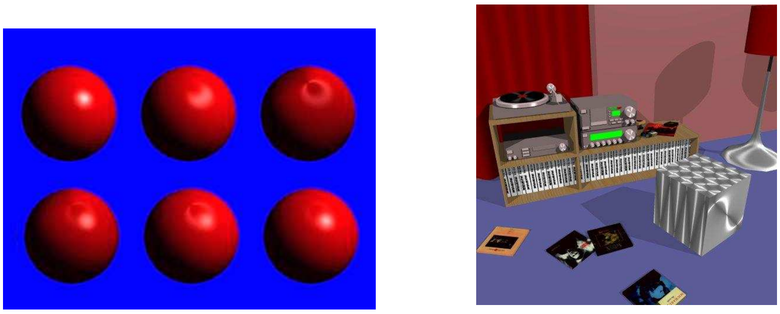

Figure 13 – Anisotropie de la réflectance, due à l'orientation du micro-relief : *à gauche*, boules de Noël façon satin ; *à droite*, aluminium brossé *(Images : Pierre Poulin, Alain Fournier, University of Toronto - UBC [POU 90]).*

En outre on peut avoir affaire à un matériau *anisotrope*, c'est à dire dont le comportement optique dépend du côté où on le regarde : ce comportement est parfois intrinsèque (certain cristaux), mais il est en général dû à l'état de surface, les cas les plus courant étant l'aluminium brossé (fonds de casseroles) et les surfaces composées de fibres (satin, chevelure, boules de Noël couvertes de fils...). Dans un certain nombre de cas simples dont ceux cités ici, on peut caractériser le caractère anisotrope du matériau par quelques paramètres (orientation, directivité des fibres), ce qui a permis à divers auteurs de proposer des expressions analytiques modélisant la fonction de réflectance de ces matériaux (cf figure 13). *Pour en savoir plus : [POU 90, KAJ 85].*

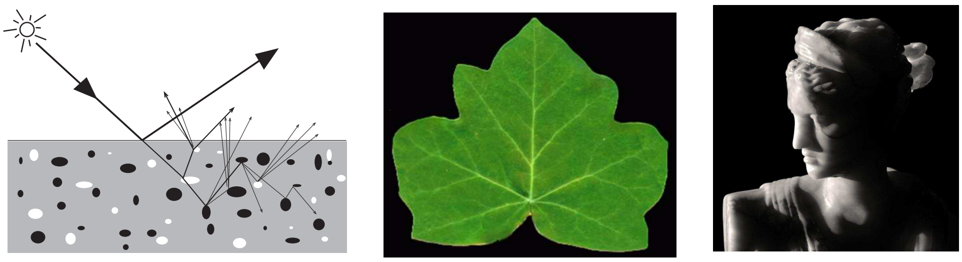

Figure 14 – *A gauche :* cheminement de la lumière à l'intérieur du matériau. La simulation de ce transport permet une modélisation plus fine de la réflectance du matériau. *Au milieu :* Rendu d'une feuille prenant en compte la matière interne *(Image : Hanrahan, Princeton, Krueger, GNRCCS [HAN 93]). A droite :* simulation complète du transport à l'intérieur du matériau, permettant la prise en compte d'effets non locaux (transmission de lumière) *(Image : Dorsey, Edelman, Jensen, Legakis, Pedersen, MIT [DOR 99]).*

Une autre approche consiste à recourir à des mesures réelles[8] du comportement optique d'un matériau, obtenues au moyen d'un gonioréflectomètre[9](cf fi-

[8]On trouve sur les sites web [Col , Cor] des tables de mesures de BRDF, et même (dans le premier) d'images de matériaux vus sous tous les angles.

[9]A noter que l'on peut également simuler un tel instrument, afin d'obtenir la fonction

gure 15). On obtient ainsi une large table à quatre dimensions, car il y a potentiellement une valeur de réflection différente pour chaque direction incidente de lumière, repérée par deux angles, et chaque direction d'observation, repérée par deux autres angles (la fonction de réflectance est d'ailleurs aussi nommée BRDF pour *bidirectional reflectance distribution function*).

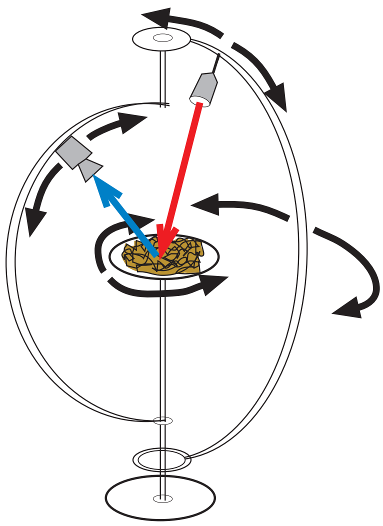

Figure 15 – Gonioréflectomètre permettant de mesurer la BRDF d'un matériau réel. Dans celui-ci la source de lumière a deux degrés de liberté angulaires ; la caméra n'en a qu'un, le second étant déporté au niveau du support à échantillons.

Pour la visualisation en temps-réel à l'aide d'une carte graphique : jusqu'à récemment, seuls étaient accessibles les modèles de Lambert et de Phong, évalués aux sommets du maillage des objets puis interpolés sur les faces. Autant le résultat était acceptable pour les surfaces mates, autant l'évaluation sur les seuls sommets du maillage permettait mal de capturer les reflets fins, sauf à subdiviser considérablement le maillage. Depuis peu, d'une part des techniques multi-passes[10] permettent de simuler des fonctions de réflectances plus variées, et d'autre part de nouvelles cartes graphiques sont à même d'évaluer la réflectance à chaque pixel.

de réflectance correspondant à une surface virtuelle dont on connaîtrait la géométrie des micro-détails [WES 92].

[10]Le *multi-passe* consiste à superposer plusieurs tracés successifs d'un même objet (en changeant par exemple la texture ou la couleur) en les combinant de manière à obtenir l'aspect souhaité.

3. Variations spatiales de l'aspect

3.1. *Ce qui peut être modulé*

Tout ce qui contribue à définir la couleur réfléchie, c'est à dire l'apparence de l'objet en un pixel donné, est susceptible d'être modulé, c'est à dire de varier d'un point à un autre de la surface, de façon à rendre compte de la variabilité ou du grain typique du matériau visé. Sont donc concernés tous les paramètres 'optiques' qui interviennent dans la fonction de réflectance (couleur diffuse, couleur ambiante, couleur spéculaire, exposant spéculaire, etc) voire la transparence, mais aussi tous les paramètres caractérisant la géométrie locale (profondeur, normale, densité volumique...). Pour les textures animées (eau, nébulosités, feu, sable...) ou évolutives (patine), les variations dépendent également du temps.

En particulier, perturber la valeur de la normale à la surface à l'aide d'une texture (ou *bump mapping*) est une technique économique et puissante pour simuler l'apparence d'un relief de faible amplitude comme les anfractuosités et la rugosité d'une roche, l'écorce d'un arbre, ou les rainures et les encoches d'une plaque métallique (cf figure 16). Comme on l'a vu lors de la section précédente, le seul élément de géométrie qui entre en compte dans le modèle d'illumination est la normale[11], c'est pourquoi faire varier ce seul paramètre suffit à donner l'illusion que l'orientation de la surface varie localement. Toutefois l'illusion a des limites : comme le relief généré est factice, il ne génère pas d'ombres, et ne paraît pas sur les silhouettes.

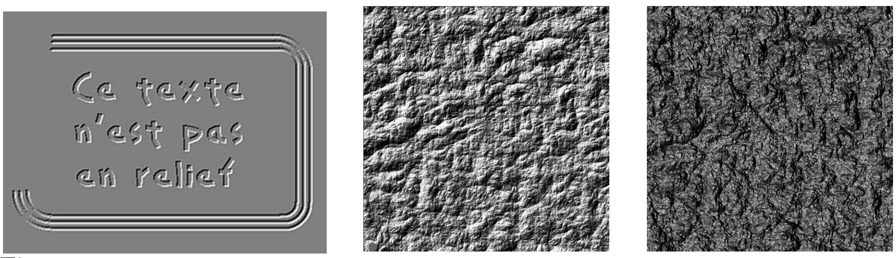

Figure 16 – *A gauche :* embossage et rainures factices. *Au milieu et à droite :* bump généré par texture de Perlin (cf section 3.3), utilisés avec deux modèles de réflectance différents *(Images : Fabrice Neyret, iMAGIS / GRAVIR).*

D'autre part, si les textures usuelles sont mono[12] ou bidimensionnelles, il est également possible de définir des textures véritablement en relief, c'est à dire approchant mieux l'apparence de détails géométriques que ne le fait une image plaquée sur une surface. Nous décrirons plusieurs de ces techniques à la

[11] Ou un peu plus si la réflectance est anisotrope : il faut alors un ou deux vecteurs supplémentaires pour indiquer la direction privilégiée. Mais le principe reste le même.

[12] Une texture monodimensionelle permet de visualiser un potentiel : altitude, couleur du ciel pour un azimut donné, charge à la surface d'une molécule... La texture consiste alors généralement en la donnée du potentiel mappé sur la surface, et d'une table de couleur (que l'on peut interpréter comme image 1D).

section 5.1. Ces divers dispositifs augmentent alors la quantité de paramètres susceptibles d'être modulés, ainsi que les variables 'structurantes' sur lesquelles fonder ces variations (profondeur dans l'épaisseur de la quasi-surface, orientation du regard, temps...).

Les textures peuvent aussi jouer un rôle d'intermédiaire technique indirectement visible : la modalité la plus courante consiste à utiliser une texture pour indiquer où utiliser un matériau (ou une texture) plutôt qu'un autre (e.g. échiquier utilisant deux essences de bois pour ses cases). Cette texture indicatrice se construit comme les autres, la différence tenant au rôle que le designer assigne aux divers ingrédients constituant le matériau. Mieux encore, la texture peut servir de structure de donnée ou d'échange avec d'autres éléments constitutifs de la scène, comme la géométrie ou l'animation : par exemple, on peut l'utiliser pour y indiquer la répartition de la densité ou du type de végétaux que le système doit construire sur le sol. Réciproquement, certains logiciels permettent de modifier la texture sur le lieu où un événement convenu se produit (e.g. pour figurer des traces de pas, ou des ronds dans l'eau). Nous en verrons des exemples en section 5.3.

Pour la visualisation en temps-réel à l'aide d'une carte graphique :
les modèles de réflectance disponibles étant restreints, on ne peut guère moduler que les couleurs (diffuse, ambiante, spéculaire) et la transparence. Depuis peu, les techniques multi-passes et certaines cartes graphiques permettent également de moduler la normale (technique du *bump-mapping*), de 'programmer' des formules simples d'illumination, ou de reconstruire certaines BRDF en passant par des tables.

3.2. *Construction des variations*

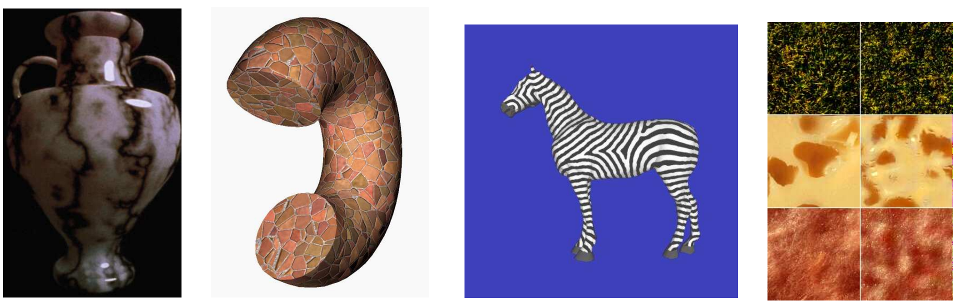

Figure 17 – *De gauche à droite* : textures de Perlin *(Image : Ken Perlin, New York University [PER 85])*, de Worley *(Image : Steven Worley, Worley labs [WOR 96])*, de réaction-diffusion *(Image : Greg Turk, University of North Carolina [TUR 91])*, de resynthèse *(Image : Thomas Maraghi, University of Toronto)*.

La façon la plus triviale de spécifier les variations d'un paramètre consiste à en fournir une image, ou plus généralement une *carte* (car le paramètre n'est pas forcément une couleur), à tapisser sur la surface. Cette image peut être globale, ou ne représenter qu'un motif (ou *pattern*, ou encore *tile*) à répéter.

L'autre solution consiste à calculer les valeurs, soit avant usage, soit au vol, à l'aide d'algorithmes dont il existe plusieurs familles :

– les *textures procédurales* [EBE 94] effectuent au vol des calculs répétitifs simples qui permettent d'imiter certains motifs naturels ou artificiels. La technique la plus utilisée est celle introduite par Perlin, qui permet d'imiter le bois, le marbre, et de nombreux matériaux à veines ou à grain. Une autre, introduite par Worley, imite bien les matériaux à cellules (tissus vivants, cristaux, écailles). Nous détaillerons ces techniques classiques plus loin.
– les techniques de *resynthèse* ou d'*analyse-synthèse* visent à reproduire les propriétés texturelles de matériaux réels, après analyse d'une photographie. Nous en traiterons en section 5.2.
– les techniques de *simulation* s'efforcent de reproduire le phénomène physique (e.g. chimique ou biologique) responsable des variations locales d'aspect dans le matériau réel à imiter. Des modèles ont été proposés pour les textures d'origine biologique (zébrures, taches de fourrure), et pour la corrosion des matériaux (patine, érosion, humidité, salissures). Nous décrirons ces techniques en section 5.3.
– les techniques *spectrales*, techniquement parentes des deux premières, utilisent la connaissance statistique (e.g. corrélations spatiale ou temporelle) dont on dispose sur un aspect ou un phénomène pour produire une texture ayant les mêmes propriétés.
– les grammaires se basent quant à elles sur une description logique de la construction de la texture (ce qui s'applique bien aux revêtements manufacturés, aux fractures, et aux végétaux).
– les textures 'techniques', que l'on a évoqué à la section précédente, permettent de rassembler une information géométrique synthétique, notamment l'*accessibilité*, qui se décline sous plusieurs formes selon l'usage prévu : ainsi, une *carte d'horizon*, indiquant la proportion libre du champ de vision depuis chaque point de la surface (mesurant ainsi si un lieu est enfoncé dans une vallée ou un repli, ou exposé sur une crête), permet d'estimer la quantité de lumière parvenant dans les anfractuosités, ou encore de contrôler la simulation de l'érosion (préférentiellement en surface pour le polissage anthropique, plutôt dans les replis pour les écoulements). Pour l'enneigement ou la prise de poussière, il faut considérer d'autres facteurs, comme la pente locale et la distribution de directions de chute. Cette catégorie de textures sera illustrée en section 5.3.

Pour la visualisation en temps-réel à l'aide d'une carte graphique : seul le plaquage d'images ou de motifs est actuellement disponible. Jusqu'à récemment on ne pouvait moduler qu'un paramètre à la fois, obligeant les programmeurs à recourir au multi-passe quand cela n'était pas suffisant ; mais les cartes graphiques récentes savent utiliser jusqu'à quatre textures simultanément. Des accélérateurs graphiques 'généralistes' capable d'évaluer n'importe quelle fonction en chaque pixel seraient promis à un grand avenir, mais cette évolution correspond à un saut quantitatif considérable dans la complexité des cartes, aussi les tentatives récentes ont échouées. On peut cependant augurer qu'un avenir vraisemblable se situe dans cette direction, puisque de plus en plus de calculs sont reportés au niveau du pixel. A moyen terme, on peut espérer l'apparition de textures procédurales de type Perlin sur les cartes graphiques. Par contre, certaines cartes savent manipuler des textures de dimension 3 ou 4, ce qui permet dores et déjà de simuler les textures volumiques et les textures pleines (introduites ci dessous).

3.3. *textures procédurales - textures pleines*

Textures de *Perlin* [PER 85]
Elles possèdent deux caractéristiques distinctes : elles sont *procédurales*, au sens défini plus haut, et elles sont *pleines* (ou *solides*, ou encore *3D* - à ne pas confondre avec les textures volumiques -) : elles sont définies dans tout l'espace, comme l'est un matériau massif (marbre, bois), et l'aspect de la surface correspond simplement à la valeur coïncidente de la couleur dans le volume, comme si la surface avait été taillée dans le matériau (cf figure 18). L'algorithme

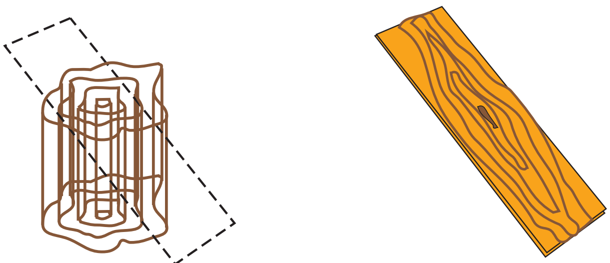

Figure 18 – Un matériau plein (ici du bois, modélisé avec une texture procédurale de Perlin) est défini dans tout l'espace (*à gauche*). Un objet doté de ce type de matériau est 'habillé' (*à droite*) en repérant la valeur du matériau coïncidant en chaque lieu de sa surface, comme si l'objet était taillé dans le matériau (*en pointillés, à gauche*).

procédural consiste à générer une *fonction de bruit*, c'est à dire une variation (un signal) à la fois continue, aléatoire et pseudo-périodique, que l'on cumule à plusieurs échelles pour la rendre fractale (on nomme le résultat *fonction de turbulence*, cf figure 19), puis à utiliser cette fonction soit directement, soit pour perturber un motif.

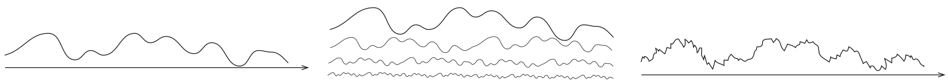

Figure 19 – Partant d'une fonction de bruit aléatoire continue pseudo-périodique (*à gauche*), on obtient une fonction fractale dite de 'turbulence'(*à droite*) par superposition de la fonction à diverses échelles (figurées *au centre*).

- L'utilisation directe passe par l'association d'une table de couleurs, qui a pour effet de transformer les extrema des variations en taches colorées, ou qui permet faire ressortir des 'veines' en isolant une valeur particulière au moyen d'une couleur contrastée. En appliquant la fonction à la transparence plutôt qu'à la couleur, on peut donner l'illusion de formes complexes, constituées des taches laissées opaques sur la surface ; c'est par exemple une méthode économique pour construire des arbres ou des nuages [GAR 85] (cf figure 20). On peut également utiliser la fonction pour modifier de manière plus ou moins appuyée la normale à la surface - c'est le *bump mapping* dont nous parlions en section 3.1 - , ce qui permet d'imiter le granulosité de la pierre, ou le froissement d'une feuille d'aluminium (cf figure 21).
- Perturber un motif permet d'imiter des matériaux complexes comme le marbre ou le bois. On choisit pour motif une 'idéalisation' de l'aspect du matériau, comme des bandes sombres parallèles pour le marbre, ou des anneaux circulaires concentriques pour le bois, dont la fonction de turbulence vient ensuite perturber l'application. A nouveau, on peut spécifier ainsi aussi bien la couleur que la transparence, la normale ou n'importe quel autre attribut.

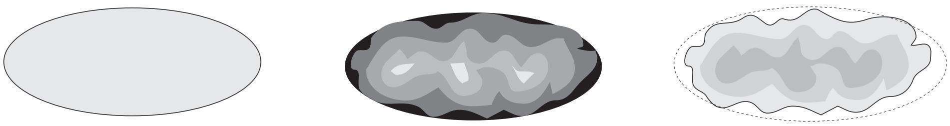

Figure 20 – Une technique pour générer économiquement des arbres et des nuages : on habille un ellipsoïde (*à gauche*) d'une texture turbulente dépendant du point de vue afin qu'elle tende vers zéro sur la silhouette (*au milieu*), et l'on affecte le résultat à la transparence (*à droite*). On confère ainsi à la surface l'illusion d'une grande complexité.

D'autre part, la fonction étant un signal 'banalisé' jusqu'à ce qu'on l'affecte à un attribut, on peut lui faire subir toute transformation qui s'applique à une fonction mathématique. Les transformations essentielles (ou *filtres* ; cf figure 22) portent sur l'échelle des abscisses ($f(x) \rightarrow f(x/\mathbf{k})$), qui permet à l'utilisateur de contrôler la dimension des perturbations (à noter qu'on les choisit souvent anisotropes, c'est à dire plus étirées dans une direction particulière), l'échelle des ordonnées ($f(x) \rightarrow \mathbf{k}f(x)$), qui contrôle l'intensité de la perturbation, le seuillage, qui permet de fabriquer des plateaux en bas ou en haut de la plage des variations, et le 'pliage' ($f(x) \rightarrow |\mathbf{2}\mathrm{f(x)} - \mathbf{1}|$), qui engendrent des ruptures produisant des motifs saillants (crêtes et vallées, lobes). Le designer dispose donc de beaucoup de moyens de contrôle : plage de fréquences spatiales

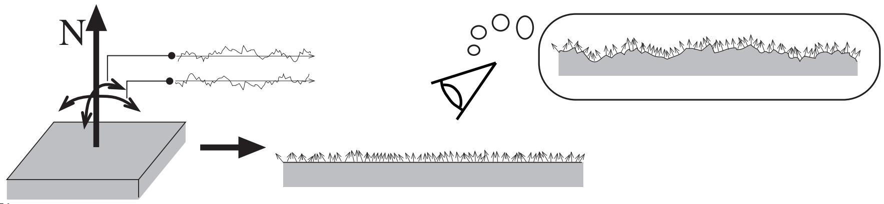

Figure 21 – Texture de perturbation de normale (ou *bump mapping*). Deux fonctions de turbulence viennent perturber la normale tangentiellement à la surface. Dans la mesure où la normale est le seul indice géométrique qui intervienne dans le calcul de l'illumination, l'apparence est similaire à celle produite par un relief authentique (par contre comme aucun relief n'est véritablement créé, il n'y a pas d'ombres portées, et la silhouette de l'objet reste lisse).

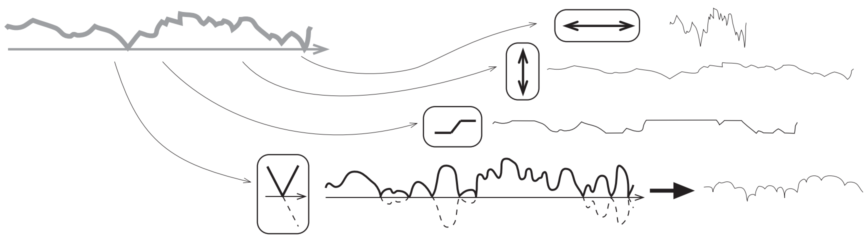

Figure 22 – Application de divers filtres au 'signal' issu de la fonction de turbulence. *De haut en bas* : scaling en abscisse (taille des motifs) et en ordonnée (intensité de la perturbation), seuillage (faisant apparaître des plateaux haut ou bas), 'repliage' permettant de créer des angles vifs (on applique plutôt ce filtre sur la fonction de bruit avant sommation, ce qui permet de créer des vallées ou des pics pointus à toutes les échelles).

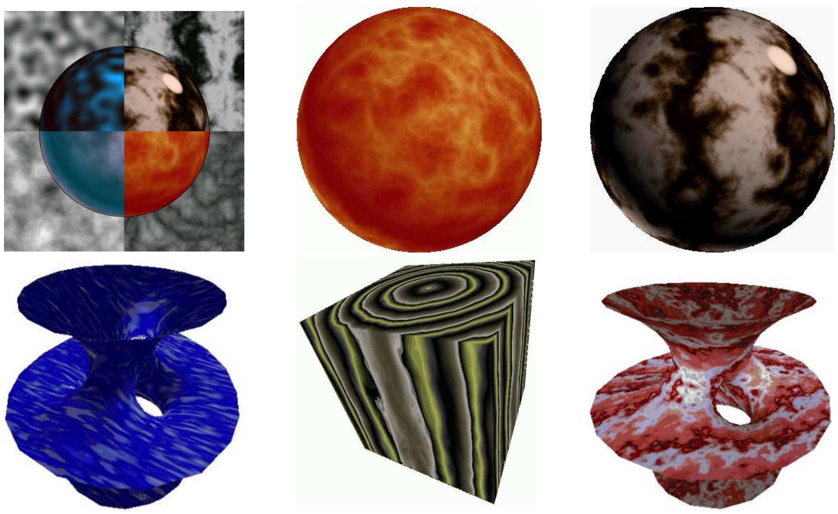

Figure 23 – *En haut* : textures procédurales 3D de Perlin *(Images : Ken Perlin, New York University [PER 85]). En bas* : textures de Perlin en temps-réel *(Images : Antoine Miné, Fabrice Neyret, iMAGIS / GRAVIR).*

caractérisant le grain du matériau ou la turbulence des veines, histogramme des couleurs, direction privilégiée, image idéalisée (avant perturbation), plus les divers choix de filtres.

Textures de Worley [WOR 96]
Elles reprennent le même schéma général, avec une autre fonction de bruit : celle-ci est construite à partir du calcul des distances à une distribution de points dans l'espace (ou *germes*), cf figure 24. On obtient des cellules en identifiant les zones de l'espace - ou en pratique, de la surface à texturer - qui sont à proximité du même point (i.e. lieux qui ont le même point comme plus proche voisin) [13], puis en les dotant d'une couleur particulière, éventuellement fonction de la distance à ce point. On obtient des cellules plus irrégulières en

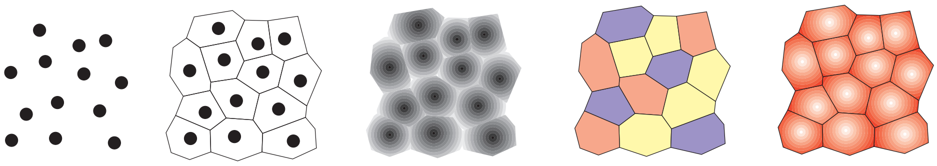

Figure 24 – Textures procédurales de Worley. *De gauche à droite* : distribution de points ; lieux les plus proches de chaque point ; distance associée en tout lieu (figurée en intensité de gris) ; affectation d'une couleur en fonction du numéro de point de plus proche ; affectation d'une couleur en fonction de la distance associée.

identifiant les zones de l'espace qui partagent le même point comme deuxième plus proche voisin ; et de même pour les troisième et quatrième plus proche voisin. En tout lieu de l'espace on dispose ainsi de quatre valeurs, les distances aux quatre points les plus proches, ainsi que des numéros de ces points. On peut alors choisir la couleur à partir d'une combinaison des quatre valeurs (e.g. $d_2 - d_1$), et éventuellement utiliser les numéros de point pour particulariser chaque cellule. Le designer peut choisir les coefficients de cette combinaison, et comme pour les textures de Perlin, il peut contrôler l'échelle et la direction privilégiée, le choix de couleurs, l'application de filtres.

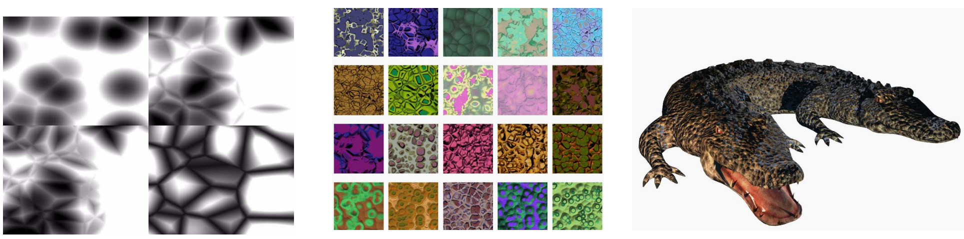

Figure 25 – Textures procédurales de Worley. *De gauche à droite* : images des distances d_1, d_2, d_3, d_4 aux germes ; textures obtenues par diverses combinaisons de ces distances ; objet texturé *(Images : Steven Worley, Worley labs [WOR 96])*.

[13] Ceci forme ce que l'on nomme *diagramme de Voronoï* associé la distribution de points.

4. Plaquage sur une surface, ou *mapping*

Comme on l'a vu, beaucoup de méthodes reposent sur la définition de l'aspect de la texture dans un 'espace intrinsèque', le plan où figure le motif, que celui-ci soit explicite ou calculé. Il faut alors ensuite 'tapisser', 'plaquer' cet espace sur la surface à texturer pour que l'aspect en tout point soit défini. Une solution simple consiste à *projeter* la texture, comme une diapositive, parallèlement à un axe, ou radialement à partir d'une forme simple (cube, cylindre, sphère), comme illustré figure 26. Mais peu de surfaces peuvent être convenablement 'habillées' d'une façon aussi frustre (par contre c'est suffisant pour des modèles de terrains, que l'on veuille plaquer une orthophotographie (verticalement) ou des courbes de niveau (horizontalement)).

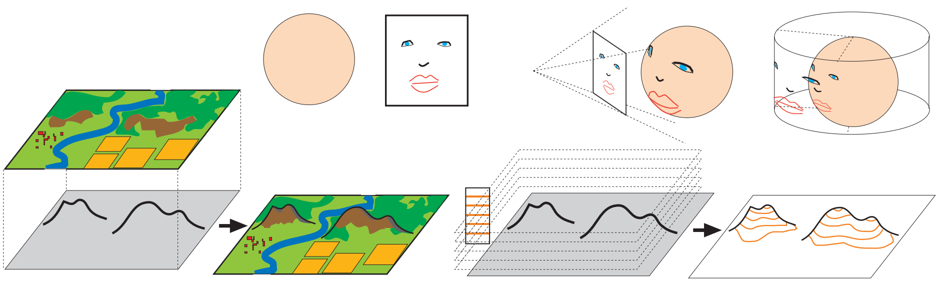

Figure 26 – Mapping par projection. *En haut* : on souhaite texturer la sphère avec l'image de visage. Projection 'en diapo' ; projection cylindrique (aucune ne donne un résultat très satisfaisant). *En bas* : Application à des terrains. Projection verticale d'une carte 2D (orthophotographie) ; projection horizontale d'une texture 1D mettant en valeur les courbes de niveau.

4.1. *Espace texturel et paramétrisation de surface*

Une méthode plus générale consiste à définir le 'tapissage' (ou *mapping*) en construisant une *paramétrisation* de la surface, c'est à dire un système de coordonnées plaqué sur l'objet (e.g. latitudes-longitudes) comme illustré en figure 27. La manière de procéder classique (et commode) est de définir des *coordonnées de textures* (souvent notées u, v) aux sommets du maillage, puis à interpoler leurs valeurs sur les faces. Ainsi, en tout pixel on peut connaître les coordonnées de texture, donc la référence sur le point de la carte qui doit apparaître à cet endroit.

Encore faut-il ensuite savoir affecter intelligemment des coordonnées de texture à chaque sommet. Certains objets de forme simple disposent d'une paramétrisation intrinsèque : solides de base, cylindres généralisés ('tuyaux'), carreaux de Bézier... Mais appliquée à la texture, elle ne donne pas toujours ce que l'on veut : la paramétrisation des sphères a des pôles, celle des carreaux a une distorsion en rapport avec la courbure ; bref, la paramétrisation utile pour définir

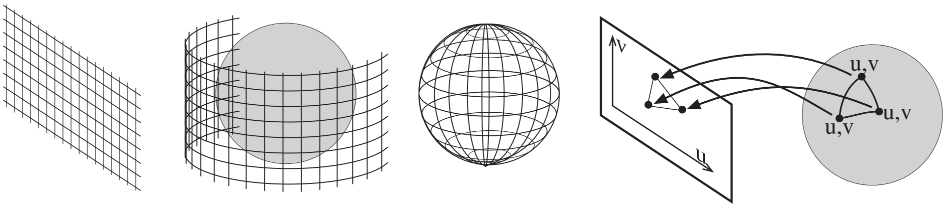

Figure 27 – Le mapping consiste à projeter l'espace texturel sur la surface, ou, ce qui revient au même, à définir une paramétrisation globale de la surface. *A droite* : en pratique, on spécifie le mapping en indiquant aux sommets du maillage les coordonnées correspondantes dans l'espace texturel (l'application en tout lieu de la surface est obtenue par interpolation).

une forme n'est pas toujours celle qui convient pour l'habiller d'une texture. D'une manière générale, quiconque a déjà essayer de poser de la tapisserie sur un mur bosselé, de couvrir à la fois le plafond et les murs, ou d'utiliser une tapisserie raccord (i.e. dont les motifs continuent d'un lé à l'autre), sait que le problème de créer un bon tapissage est loin d'être simple à résoudre : des plis et des craquelures se forment, et les raccords tombent mal (cf figure 28). On peut même démontrer mathématiquement que le problème est insoluble dans ses termes généraux, i.e. que c'est un problème 'mal posé'. Faute d'avoir le choix, dans la pratique les designers s'en accommodent de leur mieux, et vivent avec la fatalité d'avoir à subir des discontinuités ou des distorsions.

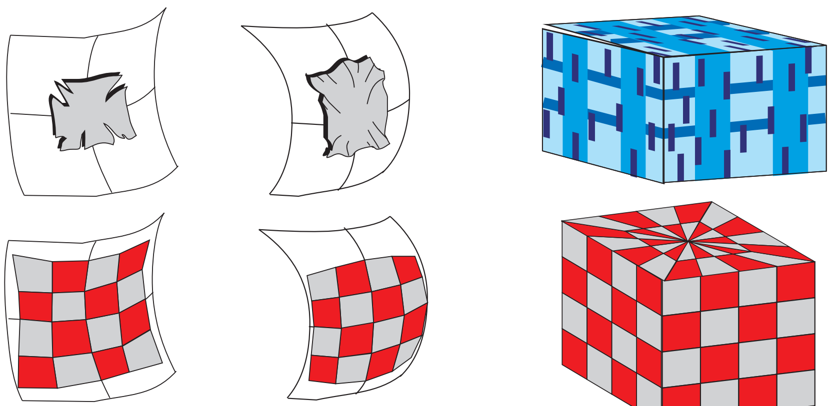

Figure 28 – *En haut, de gauche à droite* : la tapisserie se déchire sur les surfaces de courbure négative, et fait des plis quand la courbure est positive. En outre on fait fréquemment face à des problèmes de raccord, notamment sur les surfaces qui 'se referment'. *En bas* : à la différence de la tapisserie, la texture est élastique : elle se compresse ou se distend plutôt que de craquer ou plisser, mais cela engendre néanmoins une distorsion préjudiciable. Cette élasticité permet aussi de résoudre le problème de raccord (à droite), au détriment de la distorsion (qui est infinie au pôle).

Cependant la plupart des objets possèdent des régions où les discontinuités sont peu gênantes, soit que ces régions sont peu visibles ou peu regardées, soit que leur aspect est assez sombre ou du moins peu contrasté. D'autre part on peut contourner les distorsions du plaquage (cf figure 29) en les prenant en compte lors de la construction de la carte : une texture de visage peut ainsi être peinte avec des oreilles démesurées et un nez méconnaissable, prévus pour que tout revienne à sa juste proportion une fois l'image plaquée sur la tête.

4.2. *Texturer les surfaces sans les paramétrer*

D'autres solutions à base d'*atlas* [MAI 93] reviennent à constituer un 'patron' en plusieurs morceaux qui se recollent bien. Les logiciels de texturage modernes peuvent ainsi essayer d'optimiser le mapping, ou au moins assister le designer dans sa tâche. Pour les textures à base de motifs, pour lesquelles ces solutions ne s'appliquent pas (le même échantillon devant s'appliquer partout), une méthode à base de motifs triangulaires a récemment été proposée [NEY 99], laquelle permet en outre de masquer la périodicité et d'introduire des transitions entre zones d'aspect différent (limite de forêt, berges...).

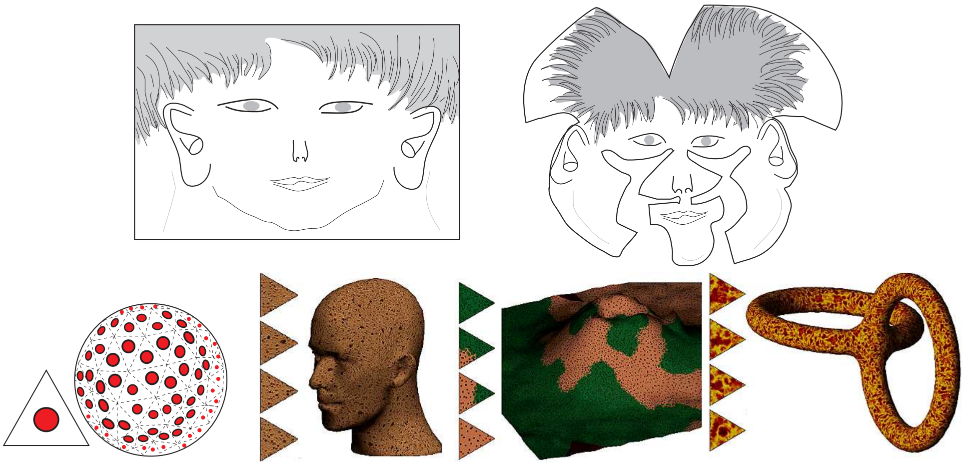

Figure 29 – *En haut à gauche* : texture de visage telle que doit la dessiner un designer pour compenser les distorsions qui interviendront au mapping. *En haut à droite* : texture par atlas, permettant de dessiner sans distorsion locale (similaire à la mise à plat de la peau d'une orange. Les atlas peuvent également être constitués d'amoncellement de morceaux non connectés). *En bas* : texturation sans distorsion à base de motifs, en utilisant des échantillons triangulaires *(Images : Neyret, Cani, iMAGIS / GRAVIR [NEY 99])*.

Comme on l'a vu plus haut, certaines méthodes ne se réfèrent pas à un plaquage : c'est le cas des textures pleines, qui sont définies dans tout l'espace, et des textures simulées, qui 'vivent' et se créent directement sur les surfaces, épousant ainsi sans heurts leur courbure et leur topologie. Ces méthodes évitent ainsi toutes les difficultés du mapping, et au delà, peuvent même offrir une continuité d'apparence entre deux objets (e.g. entre les veines du marbre d'une statue et celles du sol ou elle repose). Cependant il existe quand même un espace texturel de référence[14] : dans le premier cas il est tridimensionnel au lieu d'être surfacique, mais cet espace existe néanmoins et peut être manipulé (translation, rotation, scaling) ; dans le second, il est imposé par la nécessité de stocker l'information texturelle, et donc de pouvoir la référencer (comme elle est trop riche pour être stockée aux sommets, il faut bien passer par une table, i.e. une carte). Aussi, le formalisme général consiste à passer systématiquement par des coordonnées de texture (deux pour les images, trois pour les textures pleines), ce qui en outre facilite la manipulation des objets texturés : le designer peut par exemple choisir de tordre un objet *après* lui avoir appliqué une texture pleine, de telle sorte que la texture suive la surface dans sa déformation, ce qui n'est possible que si l'on a 'attaché' aux sommets la localisation dans l'espace texturel, comme illustré en figure 30.

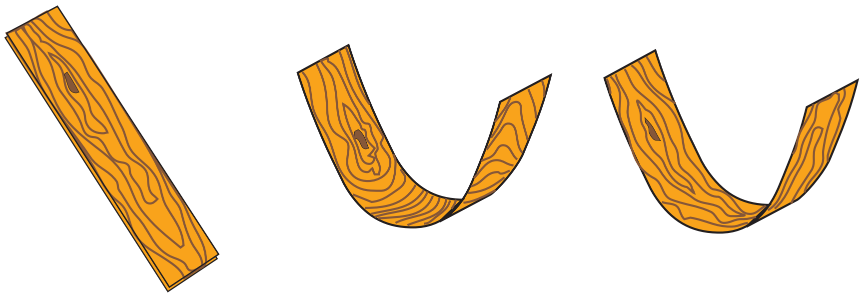

Figure 30 – On affecte une texture pleine à un objet (planche de bois, *à gauche*). Si l'on plie l'objet, il ne va plus 'tomber' aux mêmes endroits dans la texture pleine (*au milieu*), sauf si l'on attache à la surface *avant déformation* les références à la localisation dans l'espace texturel (*à droite*).

4.3. *Que penser des ennuis rencontrés à la paramétrisation ?*

La paramétrisation de surface n'est qu'un outil mathématique introduit pour faciliter l'application de la texture. Qu'il 'coince' ne veut pas dire que la surface n'est pas texturable, la preuve en est que la plupart des objets naturels présentent une surface texturée (pelage des animaux, écorce), parfois homogène sur toute leur surface (e.g. pierre en granit). Les méthodes à base d'atlas ou

[14] Ce qui ne menace pas pour autant la souplesse de ces méthodes. La contrainte faible qu'on ajoute (par exemple si l'on doit stocker des données dans la texture) est que le mapping doit être bijectif, c'est à dire qu'un point de la texture correspond à un seul point de la surface. Les logiciels de texturage pouvant construire ce genre de mapping (avec éventuellement un peu d'aide de l'utilisateur), beaucoup de designers en font maintenant usage afin de 'peindre' l'aspect directement à la surface des objets : le logiciel stocke les 'coups de pinceaux' dans la texture, laquelle peut s'avérer très distordue, ce qui est sans importance puisque l'utilisateur ne travaille plus dans l'espace texturel.

de motifs triangulaires mentionnées plus haut tirent parti de cette observation, puisque le résultat ne présente pas de discontinuité alors qu'elles n'utilisent pas de carte continue. D'autre part, si le problème du mapping sans défaut est insoluble dans ses termes généraux, il ne l'est pas forcément dans nombre de configurations particulières. Or les cas que l'on traite en pratique sont souvent des cas particuliers, qu'il faut savoir identifier comme tels faute de quoi on s'attaque à une tâche bien plus compliquée qu'il n'est nécessaire. C'est par exemple le cas pour habiller d'une texture de briques le modèle géométrique d'un immeuble : dans le monde réel on assemble des briques, et on aboutit finalement à un mur ; dans le monde virtuel on définit un mur, puis on le 'peint en briques'. La tâche se réalise en synthèse d'images dans l'ordre inverse du processus réel, ce qui risque de masquer une propriété essentielle : il y à une relation entre les dimensions de l'immeuble et celle des briques ! L'ignorer, c'est s'exposer à d'abominables problèmes de raccords le long des bords...

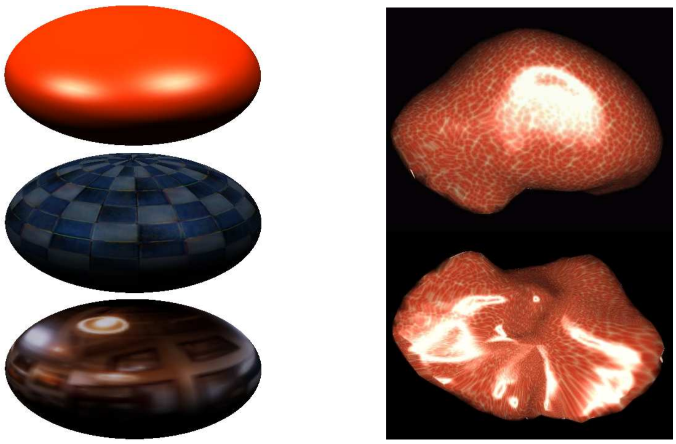

Figure 31 – *A gauche* : surface sans texture, texture plaquée, texture d'environnement. *A droite* : reflets en temps réel d'une source complexe par texture d'environnement *(Images : Raphaël Heiss, Fabrice Neyret, iMAGIS /GRAVIR).*

Pour la visualisation en temps-réel à l'aide d'une carte graphique :
Seul le mapping par spécification des coordonnées de texture aux sommets du maillage est disponible. Toutefois certaines API gèrent jusqu'à quatre coordonnées et offrent l'emploi d'une matrice de transformation spécifique, ce qui permet de simuler diverses projections (diapositive, altimétrique), ainsi que les textures volumiques et les textures pleines. Des astuces simples permettent également de créer des textures de reflets (encore appelées textures d'environnement), cf figure 31. Il n'y a donc pas vraiment de restriction par rapport aux techniques actuellement disponibles dans les logiciels de rendu évolués.

5. Matériaux complexes : quelques modèles adaptés

Nous avons suivi dans les sections précédentes le cheminement conduisant à la spécification complète de l'aspect d'un objet, hors sa forme elle-même. Bien que nous ayons surtout illustré cette démarche par des situations classiques, variantes et cas particuliers compris, nous avons insisté sur le fait que cette description était générique et s'appliquait aussi bien à des situations plus complexes.

Cependant à chaque fois que la nature du problème change, il faut recourir à un modèle capable de représenter celui-ci, tout en s'appuyant sur les bases précédemment décrites. Nous allons envisager ici trois familles de situations complexes, et présenter quelques modèles qui ont été proposés dans la littérature pour y répondre, sans entrer dans les détails. Nous allons traiter en 5.1 le cas des matériaux dont le relief ou le volume est visible (couvert végétal, métal très corrodé, treillis à grosse maille...), et qui ne peuvent donc s'assimiler à une simple 'tapisserie' posée sur une surface : ce que l'on voit dépend d'où l'on regarde (il y a un effet de parallaxe). Nous nous intéresserons ensuite à deux approches permettant de construire automatiquement des textures reproduisant l'aspect de surfaces réelles, l'une s'appuyant directement sur des images de référence (5.2), l'autre cherchant à simuler le processus de formation à l'origine de l'aspect dans le monde réel (5.3).

5.1. *Des matériaux en relief*

De nombreux matériaux se caractérisent par le fait qu'ils constituent une surface qui n'est pas lisse. L'idée qu'ils sont tapissés ou peints sur l'objet semble valable seulement 'en première approximation' ; l'illusion n'est satisfaisante que vue d'assez loin : ces matériaux forment des 'quasi-surfaces' (fourrure, cheveux, tricot, prairie...). Cependant nous avons dit au tout début du chapitre que ce que l'on qualifiait de 'matériau' ou de 'géométrie' était davantage une question d'échelle que de nature ; il est donc parfaitement envisageable que le matériau ait du relief, voire qu'il soit lui-même composé de formes 3D (e.g., matériau 'forêt' couvrant les collines). A bien y regarder, tapisser une couche de matériau épais (moquette, grillage, gazon en rouleau) n'est pas si différent de tapisser du papier peint. La difficulté n'est pas d'appliquer, mais de représenter en machine cette épaisseur.

Plusieurs techniques ont été proposées, chacune adaptée à un contexte ou un type de matériau différent. Quand il s'agit vraiment de 'tapisser de la géométrie sur de la géométrie', il est tentant de représenter le matériau par...de la géométrie !

Représentation géométrique du relief
L'avantage de la forme texturelle ne réside alors plus dans la compacité du stockage et l'efficacité du rendu, mais dans la généricité et le niveau de contrôle donné au designer. La représentation en *textures cellulaires* [FLE 95] permet ainsi de couvrir une surface d'écailles ou d'épines, le modèle se chargeant d'automatiser le placement, l'orientation, le développement et les influences réciproques entre les 'détails'.

Figure 32 – Matériaux définis géométriquement. *A gauche et au milieu :* Textures cellulaires *(Images : Fleischer, Laidlaw, Currin, Barr, CalTech [FLE 95]). A droite :* Représentation volumique *(Images : Fabrice Neyret, Syntim / INRIA).*

Les *textures de déplacement* entrent dans la même catégorie : le designer en construit le contenu avec ses outils usuels, puis la valeur en chaque pixel de texture est interprétée comme une élévation de la surface, produisant ainsi un vrai relief, sans les défauts du bump mapping (la silhouette, les ombres portées et reçues sont correctes). En outre, avec certaines contraintes supplémentaires, il est possible de représenter la carte de relief (appelée *champ de hauteurs*, ou *height field*) sous une forme hiérarchique, ce qui réduit le surcoût de rendu en limitant grandement le nombre de facettes dont il faut tester la contribution au pixel (i.e. que le rayon venant de l'œil risque d'intersecter).

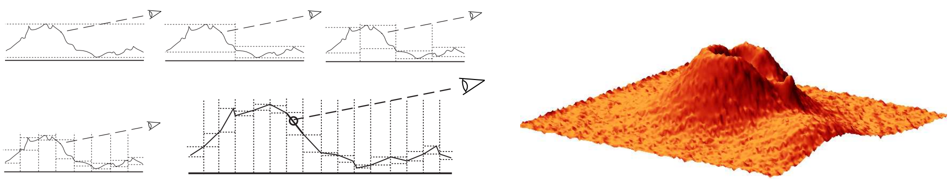

Figure 33 – Le champ de hauteurs est stocké dans une structure récursive, dont chaque cellule indique les altitudes minimale et maximale de sa zone. On ne teste ainsi les cellules plus fines que quand la zone est potentiellement visible dans le pixel en cours de calcul (i.e. le volume entre les deux altitudes est sur la trajectoire du rayon).

Cependant une représentation géométrique des détails finit par coûter beaucoup trop cher en mémoire et en temps de rendu dès que les détails ont un minimum de complexité, aussi la plupart des techniques s'intéressent avant tout à définir des représentations du relief alternatives à la représentation géométrique classique.

Représentation non géométrique du relief, ou *représentations alternatives*
L'une d'entre elles consiste à définir des *textures directionnelles* [DIS 98], dont l'apparence dépend du point de vue (caractérisé par deux angles) : on reproduit ainsi les effets de parallaxe, i.e. l'avant-plan ne masque pas les mêmes parties de l'arrière-plan quand on se déplace.

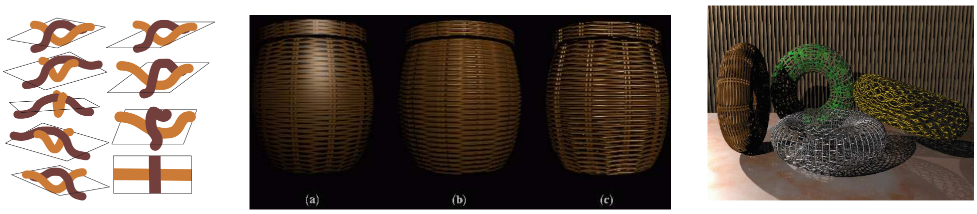

Figure 34 – Textures bidirectionnelles. *A gauche :* l'échantillon de rotin est stocké pour tous les angles de vue. *Au milieu :* texture image, texture de bump, texture bidirectionnelle. *(Images : Jean-Michel Dischler, MSI / Université de Limoge [DIS 98]).*

On peut également représenter dans la texture tout le volume se trouvant dans la 'couche superficielle', au voisinage de la surface (forêt sur une colline, fourrure sur un animal), ce qui fournit une représentation complète des détails. La couleur du pixel résulte alors de l'accumulation des couleurs trouvées sur la portion du rayon se dirigeant vers l'œil qui traverse la couche de texture. Ce volume peut être stocké explicitement (le motif - couleur, normale, transparence - étant 'peint' en 3D ou obtenu par conversion à partir d'une forme géométrique déjà construite), ce qui correspond à la représentation en *textures volumiques* [KAJ 89, NEY 98], ou défini implicitement et calculé 'au vol' par une texture procédurale en volume, ou *hypertexture* [PER 89]. Une représentation similaire aux textures volumiques est également employée lors de la simulation de matériaux dont nous parlerons en 5.3, afin de stocker la distribution des constituants internes uniquement au voisinage de la surface.

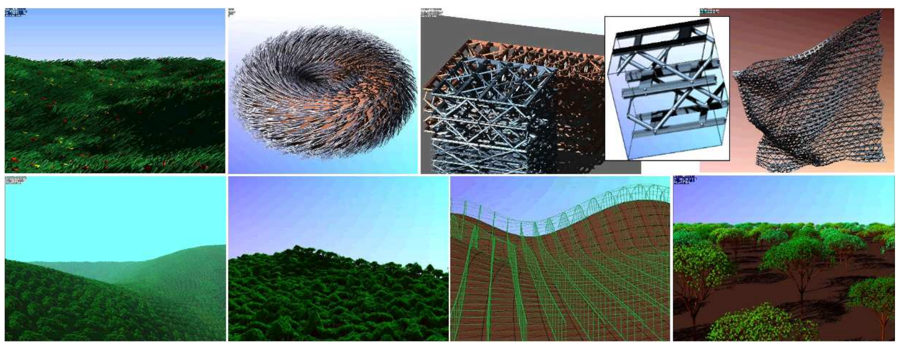

Figure 35 – Textures volumiques. Noter en médaillon le motif de texture (ou *texel*) utilisé pour les deux images voisines, et en dessous, le mapping utilisé pour la forêt, faisant apparaître l'épaisseur (dont l'orientation fait partie des degrés de liberté à la disposition du designer) *(Images : Fabrice Neyret, Syntim / INRIA).*

Figure 36 – *En haut :* hypertextures. Dès que le rayon pénètre en 'zone floue' au voisinage de la surface, on détermine la densité en chaque point à l'aide d'une texture procédurale *(Images : Ken Perlin, New York University, Eric Hoffert, ATT Pixel Machines [PER 89]). En bas :* Simulation du transport de lumière à l'intérieur des matériaux *(Images : Dorsey, Edelman, Jensen, Legakis, Pedersen, MIT [DOR 99]).*

D'autres techniques, plutôt que de spécifier explicitement toutes ces données, essayent de modéliser directement la valeur de cette somme (couleurs cumulées), ce qui est possible si l'on a une connaissance statistique de la distribution de matière (nuages, poussières, feuillage, peau) [MAX 86, STA 91, STA 01]. On est alors très proche de la méthodologie permettant d'obtenir le modèle d'illumination de certains matériaux complexes (comme l'aluminium brossé vu en section 2.3) : l'intégration analytique des effets de la distribution donne l'illumination en un point, et les bornes de cette intégrale, dépendant des variations à grande échelle de la distribution, constituent l'information texturelle[15]. La figure 37 présente un exemple hiérarchique de cette approche [MEY 00], où la répartition de la représentation entre illumination et texture dépend de la distance (i.e. de la taille visible).

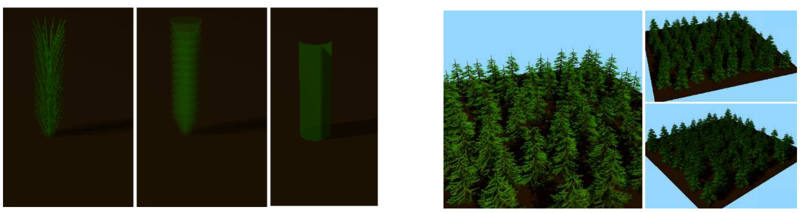

Figure 37 – Hiérarchie de *shaders* utilisés pour rendre les aiguilles de pin en fonction de la distance : une aiguille, un cône, une touffe *(Images : Alexandre Meyer, Fabrice Neyret, iMAGIS /GRAVIR [MEY 00]).*

[15] On comprend ainsi que dans divers logiciels de synthèse d'images, les *shaders*, 'programmés' par le designer pour définir la couleur en tout point, sont utilisés pour spécifier aussi bien l'illumination que la texture.

5.2. *Reproduire l'aspect du réel*

Une façon en apparence triviale de produire un matériau réaliste sur un objet de synthèse serait de prendre une photographie d'un échantillon réel, puis d'utiliser celui-ci comme texture. Les designers procèdent ainsi dans un certain nombre de cas, cependant cette méthode souffre de nombreux défauts : la texture est toujours exactement la même partout où elle est employée, elle paraît distordue si la surface n'est pas plate, et les raccords sont impossibles à assurer. De plus la texture acquise peut elle-même être distordue (par exemple par la perspective), et elle mélange la couleur propre et les effets de l'éclairage. D'un autre côté, les modèles de génération algorithmiques sont puissants (comme ceux de Perlin ou Worley, vus en section 3.2), mais il n'est pas toujours facile de régler leur paramètres de façon à obtenir un aspect réaliste, ou a fortiori l'aspect d'un motif de référence.

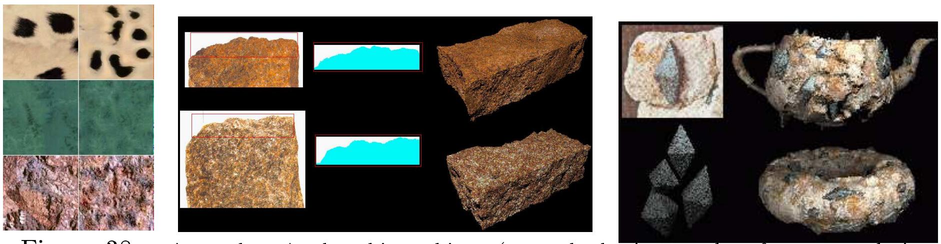

Figure 38 – *A gauche :* Analyse hiérarchique (à gauche les images de références, à droite leur resynthèse) *(Images : Heeger, Stanford University, Bergen,SRI). Au milieu et à droite :* Analyse à partir du profil et d'images 2D *(Images : Dischler, Ghazanfarpour, MSI / Université de Limoge [DIS 97]).*

Les approches par *analyse-synthèse* (ou *resynthèse*) allient ces deux techniques, en cherchant à utiliser au mieux une image de référence pour piloter un algorithme de synthèse de texture. Les techniques proposées se distinguent en deux familles : la première vise à produire une image ayant les mêmes propriétés statistiques que le modèle (spectre fréquentiel, autocorrélation...), par exemple en 'améliorant' progressivement une image aléatoire, de manière à se rapprocher de plus en plus des propriété perceptives de l'image de référence [GAG 85, DIS 97, HEE 95]. Ce germe aléatoire fait que deux images générées ne sont jamais rigoureusement semblables, et que l'on peut étendre à volonté la surface synthétisée sans jamais voir apparaître de répétition.

La seconde consiste à piloter un modèle procédural, par exemple celui de Perlin, de façon à trouver automatiquement un jeu de paramètres qui génère une texture aussi proche que possible de l'image de référence [GHA 96, LEF 00].

5.3. *Simuler le réel*

Certains motifs à l'allure très complexe résultent de processus relativement simples, par exemple d'interactions physico-chimiques locales. Il peut alors être

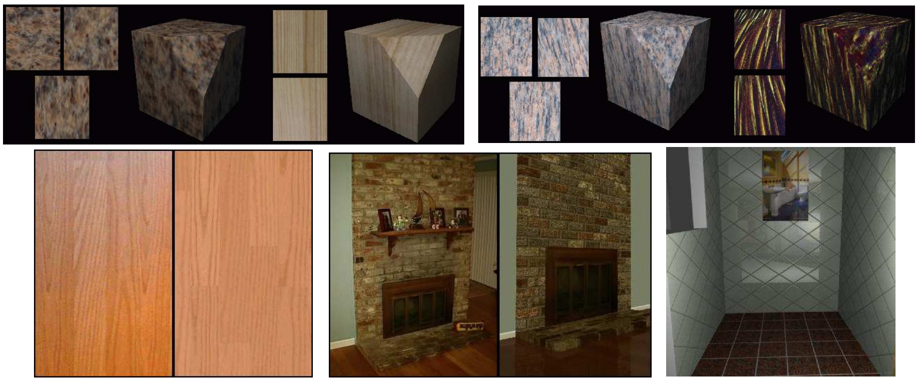

Figure 39 – *En haut :* Analyse à partir d'images dans les trois plans. *(Images : Dischler, Ghazanfarpour, MSI / Université de Limoge [GHA 96])). En bas :* Analyse de veines et de briques. Pour les planches et la cheminée, on présente à gauche les images de références, à droite leur resynthèse *(Images : Lefebvre, Poulin, Université de Montréal [LEF 00]).*

plus efficace et plus commode de reproduire directement le phénomène à l'origine de la texture que de chercher à caractériser l'aspect final lui-même.

Plusieurs modèles se sont fondés sur ce principe. Le règne biologique fourni une première vaste famille de textures d'origine chimique. Ainsi, les taches du pelage des zèbres, tigres et girafes, les décorations des coquillages, et parfois même les détails géométriques, sont gouvernées par des phénomènes de *réaction-diffusion* [WIL , FOW 92, WIT 91, TUR 91] : deux espèces chimiques ou plus, contrôlant un phénomène visible (pigment coloré, densité de fourrure, granulosité de carapace), s'affrontent tout en se propageant par diffusion, donnant des figures d'équilibre très variées[16] selon les proportions des composants, les formes d'action et de rétroaction, le temps de réponse comparé à la croissance de l'organisme, et le stade du développement de celui-ci lorsque ces réactions surviennent sur son enveloppe. Pour les coquillages, le front d'activité correspond à la cerne de coquille qui s'ajoute chaque année (il est donc unidimensionnel), et les fluctuations temporelles de la répartition des espèces chimiques se 'lit' en déroulant la coquille. Le pelage des mammifères résulte d'un affrontement entre espèces chimiques sur toute la surface de l'animal au stade embryonnaire, cependant les distorsions de la forme du corps sont telles lors de la croissance de l'embryon à l'adulte que l'allure finale des taches est totalement différente selon que celles-ci se sont fixées au début (pour le zèbre ou le tigre) ou à la fin (pour la panthère) de la croissance : les bandes du tigre sont vraisemblablement des taches étirées.

[16]Les figures en polygones apparaissent par exemple souvent dans la nature (écailles de tortue, pelage de girafe, graines de tournesol...), car elles constituent une solution d'équilibre pour beaucoup de phénomènes de répulsion ou de croissance : un polygone correspond à la zone où l'influence d'un 'noyau' l'emporte sur celle des autres noyaux.

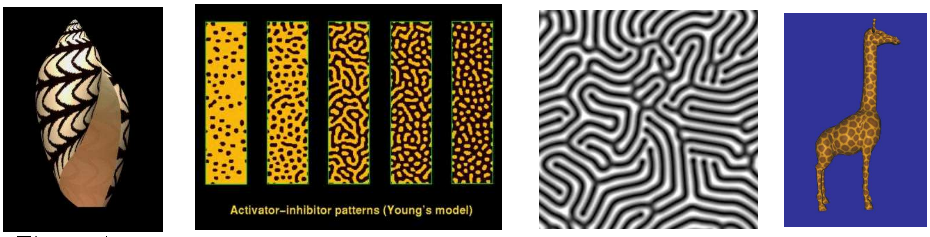

Figure 40 – Réaction-diffusion. *A gauche :* modèle 1D *(Images : Fowler, Meinhardt, Prusinkiewicz [FOW 92]). Au milieu :* différents motifs d'équilibre en 2D *(Images : idem ; Xmorphia [WIL]). A droite :* motifs de pelages ; voir aussi la figure 17. *(Images : Greg Turk, University of North Carolina [TUR 91]).*

Le monde minéral donne également lieu à nombre d'interactions et influences entre espèces chimiques, conduisant à des modifications physiques (érosion, changement de dureté, de rugosité ou de couleur). Des modèles ont été proposés notamment pour la corrosion des métaux, les salissures par écoulement, et l'érosion des minéraux (statues, constructions) [DOR 96b, DOR 96a, DOR 99]. Ce type de représentation requiert quatre éléments intervenant dans la construction de la texture du matériau : un codage des espèces chimiques en jeu et de leurs paramètres (e.g. concentration en chaque point), une correspondance entre ces données et l'aspect visuel qu'elles induisent, des lois ou des équations régissant les interactions locales entre espèces (absorption, dépôt, corrosion, entraînement), et une génération d'événements entraînant l'évolution du système (ruissellement, projection d'eau, frottement). Une texture indicatrice peut faire le pont avec le monde géométrique, par exemple pour tenir compte de l'exposition de chaque point aux intempéries. Dans le modèle proposé pour l'érosion de minéraux, la simulation est faite dans une couche volumique au voisinage de la surface, afin de reproduire les effets de migration dans l'épaisseur.

Ces textures simulées ont le grand avantage de pouvoir s'adapter facilement aux spécificités locales de la surface (courbure, exposition) et du voisinage. En particulier, elles ne souffrent ni de distorsion, ni de répétitivité. Le prix à payer se situe bien sûr dans le temps de calcul et le coût de stockage, puisqu'il faut simuler l'évolution de l'état de la texture en chaque point jusqu'à stabilisation...

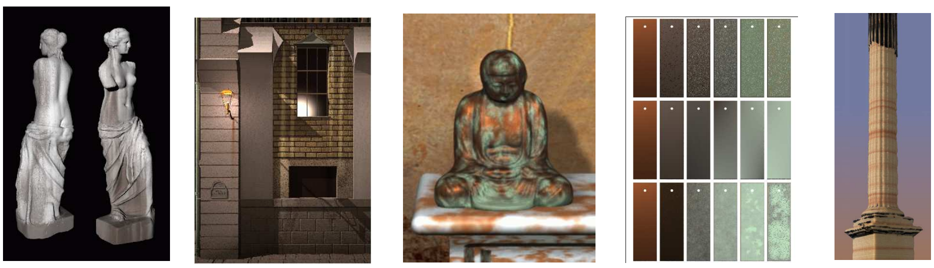

Figure 41 – Salissures dues au ruissellement *(Images : Dorsey, J., MIT H. Pedersen, and P. Hanrahan,Stanford [DOR 96b])* ; patine des métaux *(Images : Julie Dorsey and Pat Hanrahan [DOR 96a])* ; érosion des matériaux minéraux *(Images : Dorsey, Edelman, Jensen, Legakis, Pedersen, MIT [DOR 99]).*

Références

[ASH 00] Michael Ashikhmin, Simon Premoze et Peter Shirley. A microfacet-based BRDF generator. *Proceedings of SIGGRAPH 2000*, pages 65–74, July 2000. ISBN 1-58113-208-5.

[CAL] Patrick Callet. Méthode OCRE (optical constants for rendering evaluation). `http ://virtual.pl.ecp.fr :80/~callet/recherche/programme.html`.

[CAL 98] Patrick Callet. *Couleur-lumière, couleur-matière.* Diderot multimédia, January 1998. ISBN : 2-84352-087-8.

[Col] Columbia-Utrecht. Columbia-Utrecht reflectance and texture database. `http ://www.cs.columbia.edu/CAVE/curet/`.

[COO 82] R. L. Cook et K. E. Torrance. A reflectance model for computer graphics. *ACM Transactions on Graphics*, 1(1) :7–24, January 1982.

[Cor] Cornell University Program of Computer Graphics. Measurement data. `http ://www.graphics.cornell.edu/online/measurements/`.

[DIS 97] J.-M. Dischler et D. Ghazanfarpour. A procedural description of geometric textures by spectral and spatial analysis of profiles. *Computer Graphics Forum*, 16(3) :129–140, August 1997. Proceedings of Eurographics '97. ISSN 1067-7055.

[DIS 98] J.-M. Dischler. Efficiently rendering macrogeometric surface structures using bi-directional texture functions. In George Drettakis et Nelson Max, eds, *Eurographics Rendering Workshop 1998*, pages 169–180, New York City, NY, July 1998. Eurographics, Springer Wein. ISBN 3-211-83213-0.

[DOR 96a] Julie Dorsey et Pat Hanrahan. Modeling and rendering of metallic patinas. In Holly Rushmeier, ed, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 387–396. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[DOR 96b] Julie Dorsey, Hans Køhling Pedersen et Pat Hanrahan. Flow and changes in appearance. In Holly Rushmeier, ed, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 411–420. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

[DOR 99] Julie Dorsey, Alan Edelman, Justin Legakis, Henrik Wann Jensen et Hans Køhling Pedersen. Modeling and rendering of weathered stone. *Proceedings of SIGGRAPH 99*, pages 225–234, August 1999. ISBN 0-20148-560-5. Held in Los Angeles, California.

[EBE 94] David Ebert, Kent Musgrave, Darwyn Peachey, Ken Perlin et Worley. *Texturing and Modeling : A Procedural Approach.* Academic Press, October 1994. ISBN 0-12-228760-6.

[FLE 95] Kurt Fleischer, David Laidlaw, Bena Currin et Alan Barr. Cellular texture generation. In Robert Cook, ed, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 239–248. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[FOL 90] J. D. Foley, A. van Dam, S. K. Feiner et J. F. Hughes. *Computer Graphics : Principles and Practices (2nd Edition).* Addison Wesley, 1990.

[FOW 92] Deborah R. Fowler, Hans Meinhardt et Przemyslaw Prusinkiewicz. Modeling seashells. In Edwin E. Catmull, ed, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 379–388, July 1992.

Sur la morphogénèse en général, voir aussi :
`http ://www.cpsc.ucalgary.ca/Redirect/bmv/vmm/toc.html`.

[GAG 85] ANDRE GAGALOWICZ ET SONG DE MA. Model driven synthesis of natural textures for 3-D scenes. In *Eurographics '85*, pages 91–108. 1985.

[GAR 85] GEOFFREY Y. GARDNER. Visual simulation of clouds. In B. A. BARSKY, ed, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19, pages 297–303, July 1985.

[GHA 96] DJAMCHID GHAZANFARPOUR ET JEAN-MICHEL DISCHLER. Generation of 3D texture using multiple 2D models analysis. *Computer Graphics Forum*, 15(3) :311–324, August 1996. Proceedings of Eurographics '96. ISSN 1067-7055.

[GON 94] JAY S. GONDEK, GARY W. MEYER ET JONATHAN G. NEWMAN. Wavelength dependent reflectance functions. In ANDREW GLASSNER, ed, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 213–220. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[HAN 93] PAT HANRAHAN ET WOLFGANG KRUEGER. Reflection from layered surfaces due to subsurface scattering. In JAMES T. KAJIYA, ed, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 165–174, August 1993.

[HEE 95] DAVID J. HEEGER ET JAMES R. BERGEN. Pyramid-Based texture analysis/synthesis. In ROBERT COOK, ed, *SIGGRAPH 95 Conference Proceedings*, Annual Conference Series, pages 229–238. ACM SIGGRAPH, Addison Wesley, August 1995. held in Los Angeles, California, 06-11 August 1995.

[HEI 00] WOLFGANG HEIDRICH, KATJA DAUBERT, JAN KAUTZ ET HANS-PETER SEIDEL. Illuminating micro geometry based on precomputed visibility. *Proceedings of SIGGRAPH 2000*, pages 455–464, July 2000. ISBN 1-58113-208-5.

[KAJ 85] JAMES T. KAJIYA. Anisotropic reflection models. In B. A. BARSKY, ed, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19(3), pages 15–21, July 1985.

[KAJ 89] JAMES T. KAJIYA ET TIMOTHY L. KAY. Rendering fur with three dimensional textures. In JEFFREY LANE, ed, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23(3), pages 271–280, July 1989.

[LEF 00] LAURENT LEFEBVRE ET PIERRE POULIN. Analysis and synthesis of structural textures. In *Graphics Interface 2000*, pages 77–86, May 2000.
voir `http ://www.graphicsinterface.org/proceedings/2000/119/`.

[MAI 93] JÉRÔME MAILLOT, HUSSEIN YAHIA ET ANNE VERROUST. Interactive texture mapping. In JAMES T. KAJIYA, ed, *Computer Graphics (SIGGRAPH '93 Proceedings)*, volume 27, pages 27–34, August 1993.

[MAX 86] N. L. MAX. Light diffusion through clouds and haze. *Computer Vision, Graphics and Image Processing*, 33(3) :280–292, March 1986.

[MEY 00] ALEXANDRE MEYER ET FABRICE NEYRET. Multiscale shaders for the efficient realistic rendering of pine-trees. In *Graphics Interface*, pages 137–144. Canadian Information Processing Society, Canadian Human-Computer Communications Society, May 2000. ISBN 0-9695338-9-6
voir `http ://www.graphicsinterface.org/proceedings/2000/184/`
et `http ://www-imagis.imag.fr/Membres/Alexandre.Meyer/research/gi00/`.

[NEY 98] FABRICE NEYRET. Modeling animating and rendering complex scenes using volumetric textures. *IEEE Transactions on Visualization and Computer Graphics*, 4(1), January–March 1998. ISSN 1077-2626. Voir :
`http ://www-imagis.imag.fr/Membres/Fabrice.Neyret/texels/index-fra.html`.

[NEY 99] Fabrice Neyret et Marie-Paule Cani. Pattern-based texturing revisited. In *SIGGRAPH 99 Conference Proceedings*. ACM SIGGRAPH, Addison Wesley, August 1999.
Voir `http ://www-imagis.imag.fr/Membres/Fabrice.Neyret/publis/SIG99/`
et sur les textures : `http ://www-imagis.imag.fr/Membres/Fabrice.Neyret/textures/`.

[ORE 94] Michael Oren et Shree K. Nayar. Generalization of lambert's reflectance model. In Andrew Glassner, ed, *Proceedings of SIGGRAPH '94 (Orlando, Florida, July 24–29, 1994)*, Computer Graphics Proceedings, Annual Conference Series, pages 239–246. ACM SIGGRAPH, ACM Press, July 1994. ISBN 0-89791-667-0.

[PER 85] Ken Perlin. An image synthesizer. In B. A. Barsky, ed, *Computer Graphics (SIGGRAPH '85 Proceedings)*, volume 19(3), pages 287–296, July 1985.
voir aussi `http ://www.noisemachine.com/`.

[PER 89] Ken Perlin et Eric M. Hoffert. Hypertexture. In Jeffrey Lane, ed, *Computer Graphics (SIGGRAPH '89 Proceedings)*, volume 23(3), pages 253–262, July 1989.

[POU 90] Pierre Poulin et Alain Fournier. A model for anisotropic reflection. In Forest Baskett, ed, *Computer Graphics (SIGGRAPH '90 Proceedings)*, volume 24(4), pages 273–282, August 1990.

[RUS] Szymon Rusinkiewicz. bv - a BRDF browser.
`http ://graphics.stanford.edu/~smr/brdf/bv/`.

[STA 91] Jos Stam et Eugene Fiume. A multiple-scale stochastic modelling primitive. In *Proceedings of Graphics Interface '91*, pages 24–31, June 1991.

[STA 01] Jos Stam. An illumination model for a skin layer bounded by rough surfaces. In *Eurographics Workshop on Rendering*, pages 39–52, Jul 2001.

[TUR 91] Greg Turk. Generating textures for arbitrary surfaces using reaction-diffusion. In Thomas W. Sederberg, ed, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 289–298, July 1991.

[WAR 92] Gregory J. Ward. Measuring and modeling anisotropic reflection. In Edwin E. Catmull, ed, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 265–272, July 1992.

[WES 92] Stephen H. Westin, James R. Arvo et Kenneth E. Torrance. Predicting reflectance functions from complex surfaces. In Edwin E. Catmull, ed, *Computer Graphics (SIGGRAPH '92 Proceedings)*, volume 26, pages 255–264, July 1992.

[WIL] Roy Williams. Xmorphia, morphogenesis from a reaction-diffusion system.
`http ://www.cacr.caltech.edu/ismap/image.html`.

[WIT 91] Andrew Witkin et Michael Kass. Reaction-diffusion textures. In Thomas W. Sederberg, ed, *Computer Graphics (SIGGRAPH '91 Proceedings)*, volume 25, pages 299–308, July 1991.

[WOR 96] Steven P. Worley. A cellular texture basis function. In Holly Rushmeier, ed, *SIGGRAPH 96 Conference Proceedings*, Annual Conference Series, pages 291–294. ACM SIGGRAPH, Addison Wesley, August 1996. held in New Orleans, Louisiana, 04-09 August 1996.

Pattern-Based Texturing Revisited

Fabrice Neyret Marie-Paule Cani

iMAGIS*/ GRAVIR-IMAG

Abstract

We present a texturing method that correctly maps homogeneous non-periodic textures to arbitrary surfaces without any of the difficulties usually encountered using existing tools. Our technique requires little redundant designer work, has low time and memory costs during rendering and provides high texture resolution.

The idea is simple: a few triangular texture samples, which obey specific boundary conditions, are chosen from the desired pattern and mapped in a non-periodic fashion onto the surface. Our mapping algorithm enables us to freely tune the scale of the texture with respect to the object's geometry, while minimizing distortions. Moreover, it yields singularity-free texturing whatever the topology of the object. The sets of texture samples may be created interactively from pictures or drawings. We also provide two alternative methods for automatically generating them, defined as extensions of Perlin's and Worley's procedural texture synthesis techniques.

As our results show, the method produces textured objects that look reasonable from any viewpoint and can be used in real-time applications.

Keywords: Texture Mapping, Patterns, Texture Synthesis, Non-periodic Tiling

1 Introduction

Reproducing the visual complexity of the real world is a dream for many Computer Graphics practitioners. Since every detail cannot be modeled at the geometric level, textures are very useful for adding visual complexity to a synthetic scene. They can for instance be used for representing rocks or vegetation on a distant mountain, for simulating animals' fur or skin, human clothes, or the surface aspect of a material. Most of the textures we need for modeling natural objects, either mineral, vegetable, or animal, have a common feature: they may look homogeneous at a large scale (i.e., large scale statistical properties do not depend on the location), but no visible periodicity can be found anywhere.

Texturing arbitrary shapes with such textures is a challenge for artists, since no CG tool is really adequate to fit real-world constraints: generating the texture directly on the surface is memory and time consuming (either for the CPU or the artist), while using standard image mapping results in pattern distortions, discontinuities, and obvious periodicity.

We present a practical solution to this problem, which involves no increase in computational or memory cost at rendering time over standard image mapping techniques using repetitive patterns. Our method works for arbitrary surfaces. It yields little distortion of the texture, no singularities whatever the topology of the surface, and no periodicity.

1.1 Related work

Despite years of CG research and tool development, artists still have difficult (and time consuming) work to do in order to achieve the texturing of complex surfaces. This paper focuses on homogeneous non-periodic textures, such as those we need for natural objects (textures may define any surface attribute, such as color, transparency, normal perturbation or displacement). The two mains problems to solve are *texture generation* and *texture mapping*. Let us review the solutions offered by existing tools:

Standard 2D texture mapping: The first solution consists of mapping a single image of the desired texture onto the synthetic object. To do this, a global parameterization of the object surface is required. As a consequence, there will necessarily be discontinuities of the texture somewhere on the surface if the object is closed or has a higher topologic order. Moreover, the texture may be highly distorted if the object has an arbitrary geometry. Optimization techniques such as those in [1, 11] can be used to reduce distortions, either locally, or by allowing the introduction of 'cracks', i.e., discontinuities. Entirely suppressing distortions by editing the mapping is impossible, except if the object's surface can be unfolded onto a plane (such as a cloth). This is not the case for natural shapes. A solution for the user to eliminate apparent texture distortions is to draw a pre-distorted texture that will compensate for the distortions due to the mapping. However, this requires high designer skills[2], and the work needs to be re-done from scratch for every new object.

An alternative is to use pattern-based texture mapping, which consists of repetitively mapping a small rectangular texture patch representing a sample of the desired texture pattern onto the surface. The sample image has to obey specific boundary conditions in order to join correctly with itself. More precisely, it needs to have a toroidal topology: the texture on the left edge must fit the texture on the right, and respectively the top edge has to fit with the bottom. Such texture samples can be created by editing pictures or drawings using interactive 2D painting systems. An advantage with respect to the previous approach is that, being small, the texture sample will be stored at a higher resolution, and will demand less redundant work by the artist. Moreover, it can be re-used for texturing other objects. Discontinuity and distortion problems, however, will be exactly the same as for a single texture map as long as a global parameterization is used to map the texture pattern. See Figure 1.

It should be be noted that these two techniques are the only methods available in current graphics hardware. Thus, other representations or design techniques have to be converted into this representation for rendering if real time constraints apply.

*iMAGIS is a joint project of CNRS, INRIA, Institut National Polytechnique de Grenoble and Université Joseph Fourier.
Address: BP 53, F-38041 Grenoble cedex 09, France
E-mail: [Fabrice.Neyret|Marie-Paule.Cani]@imag.fr
WWW: http://www-imagis.imag.fr/TEXTURES/

[2]This is actually done in practice in industry!

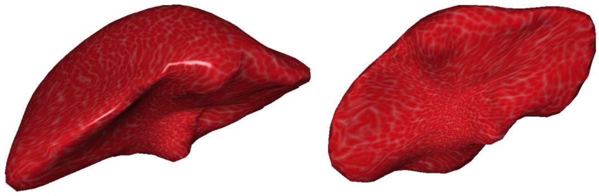

Figure 1: Standard pattern-based mapping used for applying a cellular pattern onto the geometric model of a liver. Distortions are clearly noticeable.

Interactive techniques: The problem of finding good local parameterizations for the surfaces is solved in patch-based interactive texturing systems by leaving the user to tile the surface [12, 14, 15]. In [14], the latter interactively subdivides an implicit surface into square patches. Surface geodesics are used for fitting the borders of these patches to the surface. Optimization is then used for deriving a minimally-distorted local parameterization inside each patch. This approach, which can be extended to parametric surfaces as well, can be combined with pattern-based texturing in order to cover an object with a given pattern. However, using a local instead of a global parameterization is not sufficient for avoiding texture discontinuities on closed surfaces (to be convinced, try to map a texture sample with a toroidal topology onto a cube): texture discontinuities will appear across some of the edges, since the neighboring borders of the sample image cannot be those expected everywhere.

Entirely avoiding both distortions and discontinuities can be achieved by using interactive texture painting software [8]. As in the first method, a single texture map corresponding to a global parameterization of the surface is used. However the texture content is directly designed on the object's surface before being stored as a map. The texture map may then appear distorted and discontinuous, but it will be correct when it is rendered. Depending on the user's skills, an homogeneous non-periodic texture may be designed using this method. However, this technique yields a high memory cost (as in the first approach) and consumes lots of user's time since texture details must be drawn all over the surface. Moreover, the user work is almost never re-usable on another shape.

Texture synthesis techniques: An alternative to painting the texture onto the surface is to automatically generate it, which has the advantage of saving user's time by replacing the redundant design work by a high level control of the texture features. A wide range of parametric texture synthesis techniques that are convenient for generating natural textures have been proposed [16, 21, 19, 22].

One such method is solid texturing, which involves defining a 3D material field (e.g. marble, wood) which is intersected with the object's surface to create the texture [16, 22]. No distortion nor discontinuity across the object's edges will be produced, since no surface mapping occurs. However the method is restricted to texture patterns that intrinsically come from a 3D phenomenon: it cannot capture surface properties such as the detailed appearance of the skin (e.g. regular scales). Another drawback is that synthesizing the texture during rendering will not allow real-time performance, since it is a per-pixel based computation. An alternative would be to store 3D texture tables at a high memory cost.

Other procedural techniques such as reaction diffusion [21, 19] can be used to generate a pattern-based texture directly on an object's surface. These methods are computationally costly or have a high memory cost, depending whether the texture is generated on the fly or precomputed and stored. We can note that Perlin's and Worley's techniques may also be used on surfaces (as opposed to solid material). However Perlin's noise requires a grid to be generated, so a global parameterization needs to be introduced.

Lastly, all the listed procedural techniques can easily be extended to the automatic generation of square 2D texture samples that have a toroidal topology. The latter can then be used in pattern-based mapping techniques, with the distortion and discontinuity problems discussed above. Moreover, this technique would produce patterns with obvious periodicity, which would probably spoil the natural appearance of the final object.

Towards non-periodic mapping: Artistic and mathematical work on tilings such as those of Escher and Penrose (see for instance [3, 5, 6]) can also be a source of inspiration. Escher's drawings include several tilings of the plane with complex shapes such as birds, fishes or reptiles. Penrose studies aperiodic tilings of the plane, and shows that some specific sets of tiles always form non-periodic patterns.

A first attempt to build a practical application of these ideas to texturing in Computer Graphics is Stam's work on "aperiodic textures" [17]. His aim is to render water surfaces and caustics. Stam tiles the plane with a standard grid of square patches, where he maps 16 different texture samples of an homogeneous texture. To do so in a non-periodic fashion, he uses an algorithm for aperiodically tiling the plane with convex polygons of different colors [7] (the colors of the tiles in the mathematical theory correspond to the boundaries of texture tiles). The boundary conditions between texture samples are met by using the same input noise for generating the texture in the rectangular regions that surrounds a shared edge. This method is restricted to applying textures onto a plane, otherwise the usual parameterization problems yielding distortions and discontinuities would appear. Moreover, the problem of synthesizing the texture samples is only addressed for a specific texture, and the algorithm is not explicitly described.

1.2 Overview

As shown above, none of the existing tools provides an acceptable solution to the problem of texturing arbitrary surfaces without distortions and discontinuities. With these constraints, previous methods demand too much designer intervention, are not compatible with real-time display, occupy a large memory space, or a combination of the above. This is a critical situation since most applications of Computer Graphics rely on photo-realistic texturing[3]. In this paper, we introduce a full solution to this problem, designed in the spirit of pattern-based texture mapping techniques. Our method avoids discontinuities for all surface topologies, minimizes texture distortions, and avoids the periodicity of the texture patterns.

Our solution is inspired from Escher's work since the surface will not be tiled with square patches as usual, but rather with triangles on which equilateral triangular texture samples will be mapped. It also has connections with Pedersen's geodesics [14], but they are used in our case in an automatic tiling framework, where the user just has to choose the size of the texture triangles with respect to the object's geometry. Our solution to non-periodicity is similar in spirit to the one used by Stam [17], however we have solved the more intricate problem of assigning sets of triangular texture samples onto a curved surface that may be closed, while meeting boundary conditions everywhere. Lastly, our work generalizes Perlin's and Worley's texture synthesis techniques [16, 22] by allowing the automatic generation of adequate sets of triangular texture samples. Solutions using real or hand-made images are also provided.

The remainder of this paper is developed as follows: Section 2 introduces the main features of our approach. Section 3 deals with the mapping of texture samples onto an arbitrary object geometry. Section 4 describes different methods for the generation of texture sample sets, and shows a variety of results. We conclude in Section 5.

[3]Moreover, tools have to cope with real-time constraints for video-games or flight simulators, and artist-time constraints for special effects in movies production, for which designing textures is a huge part.

2 Texturing with Triangular Patches

This paper focuses on applying textures that are homogeneous at a large scale. Moreover, as for natural textures, they should be continuous, and no periodicity should be observed. The first feature is obtained by using patterns that capture the short scale surface aspect variations, and by mapping them on the surface with low distortion (see subsection 2.1). The mapping deals with the boundary conditions at the junction between patterns, discussed in subsection 2.2. The final issue concerns the assignment method, explained in subsection 2.3.

The solution described here is designed for isotropic texture patterns, which can be found in many natural objects (for instance in most human and vegetable tissues, in rust, dust, and in numerous bumpy surfaces such as rock, ground, and roughcast wall). A possible extension enabling the introduction and control of some anisotropy is discussed in future work.

2.1 Local parameterization with triangles

As we have seen in Section 1.1, the main source of problems in usual mapping techniques is that they generally rely on global surface parameterizations. In most cases, finding a correct global mapping is simply impossible. However, Nature does not need to introduce global parameterizations to 'build' its textures; local parameterizations, together with continuity constraints, are sufficient. A tiling into continuous regions that can be locally parameterized can always be found for the continuous surfaces we wish to texture. Optimization methods will work better when applied to these local regions than to the whole surface.

Rather that defining a tiling with square patches, triangles can be used to tile a surface into regions where a local parameterization will be defined. However, to the authors knowledge, no previous work has used triangular texture patterns to design surface aspect in Computer Graphics. To address the different problem of mesh re-tiling [20], Turk produces a regular triangular surface tiling with controlled size. His algorithm fits precisely our requirements. We claim that if a polygonal tiling adequately captures an object's topology, it can be used for mapping textures. This is the case even if the tiling does not conveniently redefine the geometry, being either too coarse or too precise. Consequently, the first step of our algorithm consists of building a *triangular tiling* of the surface, computed at a user-defined scale. This tiling defines a set of local parameterizations of the surfaces, which will be used for texture mapping. More precisely, a given texture sample is going to be mapped onto each triangular patch of the tiling, whose scale thus controls the texture scale. In the remainder of the paper, we call this tiling the *texture mesh*, as opposed to the *geometric mesh* that is still used to define the shape during rendering.

2.2 Texture Samples and Boundary Conditions

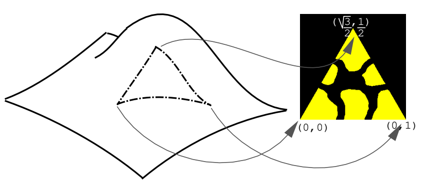

Figure 2: Each triangular patch of the *texture mesh* is mapped onto an equilateral region in a given texture sample.

The idea is to map a triangular image onto each patch of the texture mesh (see Figure 2). As these images are equilateral triangles, there will be no visible distortion if the patches of the texture mesh are approximately equilateral triangles. In order to generate a continuous texture over the entire surface, specific boundary conditions will have to be met between texture samples mapped onto adjacent patches. Basically, the two patterns in the neighborhood of the border separating two patches have to fit, which means that at least both texture values and derivatives are to be the same along a common edge. The methods presented in this paper solve for these local continuity constraints under the hypothesis that the textured patches are equilateral. If a texture sample happened to be mapped on a low quality patch (e.g. with sharp corners), the texture would be distorted and discontinuities of the texture gradient would appear on the patch edges. We have found that our solution is still sufficient in practice, since good quality meshes made of quasi-equilateral triangles can be computed for almost any surface.

Achieving texture continuity constraints using triangular tiles is more intricate than doing it with a square grid of tiles. As soon as a mesh node on a curved surface can be shared by an arbitrary (small) number of neighboring patches (either triangular or square), there is no longer anything equivalent to the 'toroidal topology' that exists when tiling the plane: no global orientation can be defined, so two patches may be connected by any edges. Moreover, this increases a priori the constraints on the texture content near the sample corners. More precisely, this enforces a zero texture gradient at those points, otherwise one would have to manage a continuity constraint between the edges of a triangle. The methods we provide for the generation of texture samples, described in Section 4, will have to cope with these boundary constraints. As our results will show, the gradient constraint at corners does not create any noticeable visual artifacts.

Since our method only relies on local parameterizations and on local continuity constraints between texture patches, it yields singularity free texturing whatever the topology of the object is (see Figure 3). However, the scale of the texture details drawn on the samples must not be too large. Otherwise, unexpected path shapes will appear at the object surface, such as those depicted in Figure 3. A practical solution to this signal processing problem consists of using patterns that are large enough to contain more than a single feature of the texture.

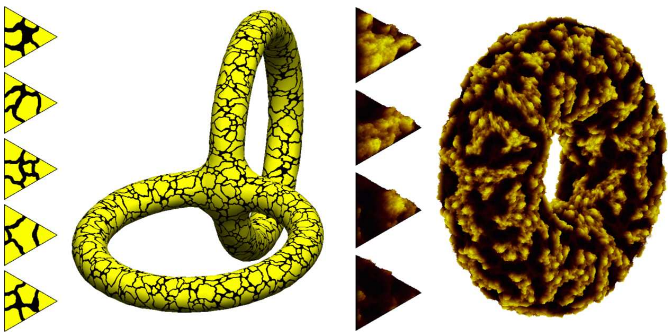

Figure 3: Two cases where unexpected path shapes appear since the lowest frequency of the texture (i.e. the grain size) is too close to the sample scale. *Left:* A naive set of texture samples designed to figure cells onto a surface. *Right:* A set of procedurally generated volumetric textures.

2.3 Assignment of Texture Samples

In order to create homogeneous textures that look like those of natural objects, several different texture samples have to be designed and non-periodically mapped onto the surface. The problem is to find how many samples are required in order to guarantee that the continuity constraints at boundaries will be respected, and to allow sufficient variations of the texture. We must also find an algorithm for assigning texture samples to the patches of the texture mesh.

The mathematical expression of this problem is not as simple as in Stam's case [17], where the mathematical theory provided a simple algorithm for achieving aperiodic mapping onto a plane, and linked the number of texture samples to the number of required boundary conditions (e.g. 16). We still have to solve a graph coloring problem (where the triangular patches represent the graph nodes and where the set of three boundaries conditions to assign them correspond to the different colors), but the graph is now a highly non-regular structure, with a varying number of neighbors per node.

A practical method for always providing a solution to continuity constraints is to use texture sample sets that include at least one texture triangle for each possible choice of three edge-constraints. More variation, or more user-control on the large-scale aspect, can be obtained by fixing a material value at each node of the texture mesh. Thus, at least one edge-constraint should be provided for each possible choice of pair of node values. A simple three step stochastic algorithm can then be used to consistently assign the triangular samples onto the surface, in a statistically non-periodic way:

1. randomly[4] choose which material value (among those used at corners of texture triangles in the sample set) is associated with each texture mesh node.

2. randomly choose which edge (among those used in the texture sample set that are compatible with the values at nodes) is associated with each geometrical edge of the texture mesh;

3. randomly assign a texture sample to each patch, among those that obey the three required boundary conditions.

The question now is: how many different texture samples do we need ? Since continuity conditions along an edge between two samples involve the gradient of the color map, the edges used in step 2 must be seen as *oriented edges*: they usually yield different boundary conditions on their two sides (to be convinced, note that a textured triangle does not smoothly weld with its mirror image, except in the special case where the gradient of the image is locally perpendicular to the common edge everywhere along it). Suppose now that we have mapped a single oriented edge e all over the texture mesh. Let us denote by E and $\bar{E}$ the different boundary conditions that the texture samples should fit on both sides of e (see Figure 4). Then, at least four texture triangles respectively obeying the conditions (E,E,E), $(E,E,\bar{E})$, $(E,\bar{E},\bar{E})$, or $(\bar{E},\bar{E},\bar{E})$ must be provided. The other possible values for the boundaries conditions (such as $(E,\bar{E},E)$ for instance) will be met by a rotated instance of one of these triangles.

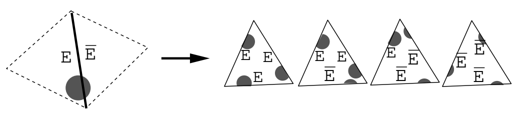

Figure 4: An oriented edge, and the set of four texture samples that need to be created to fit the different boundary conditions it produces.

In the general case of n different boundary conditions (i.e. two times the number of oriented edges), the number of texture triangles needed will be $n + n(n-1) + n(n-1)(n-2)/3$, in which the first term corresponds to the condition where the same edge is used three times, the second one to conditions with two different edge values, and the third one to solutions with three different edge values. For instance, 24 texture triangles will be needed if 2 oriented-edges are used instead of 1 (i.e. 4 boundary conditions instead of E and $\bar{E}$). Since the number of triangles required increases as a power of 3, our algorithm is not convenient for a larger number of edges. Note that if several values at nodes are used in order to constrain possible edge values, the combination of possible triangles is far smaller since edge choices have to be compatible.

In our examples, we use the minimum number of degrees of freedom, with correct results: A single kind of edge (thus two boundary conditions) is used for Figures 10,12,14. We even used the special case of symmetry mentioned above to avoid doubling the boundary condition per edge type in Figures 3(left) and 9. Noticeable repetitivity may happen with some kinds of pattern when using a small number of edge conditions. Our solution consists of providing several completely different texture samples that fit the same boundary conditions. For instance, in Figure 3 left, five texture samples fitting a single symmetric edge constraint are used. The generation of several samples fitting the same boundary conditions can be easily done using the automatic texture synthesis techniques that will be presented in Section 4. Figure 9 illustrates the use of 2 different edge conditions (using symmetry, in order to get only 2 boundary conditions, thus 4 possible triangles). An example where node values are used is presented in Figure 5, with 2 possible values (forest and ground), and symmetric boundary conditions (thus still only 4 triangles to define).

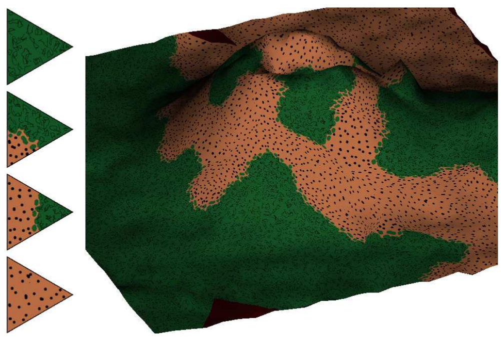

Figure 5: Mountain covered by forest. The location of forest and ground material is controlled by the values at the nodes of the texture mesh. These values are 'painted' by the user with their probability attribute (intensity of presence).

3 Mapping

In this section, we assume that we have a set of texture samples obeying adequate boundary conditions, and we describe our method for mapping them on a surface at a user controlled scale.

As suggested above, our solution for providing such control is to map texture samples onto a specifically defined *texture mesh* that tiles the surface, instead of mapping them onto the triangles that describe the object's geometry. This brings several advantages. Firstly, the texture scale becomes completely independent from the object's geometry, which is a very useful property in practice. Secondly, we can compute a high quality mesh in terms of the angular properties of the triangular patches. This will allow the mapping of equilateral texture samples without generating too large texture distortions, whatever the quality of the geometric mesh. Lastly, using a texture mesh that would be too coarse to adequately describe the object geometry is not a problem; the initial geometric mesh will still be rendered. The texture mesh just serves as a set of local parameterizations providing an image identifier and adequate texture coordinates for the geometric mesh vertices.

3.1 Overview of the texture mapping algorithm

The user first chooses the set of texture samples he wants to use, with the associated set of possible boundary conditions. He also indicates at which scale texture should be mapped by specifying

[4]This choice can be totally random to provide more variation, totally user defined, or defined with probabilities.

the desired density of texture control points (i.e., points that will be the vertices of the texture mesh) on the object's surface. Then, texture mapping is performed in the following four steps:

1. We randomly generate texture control points of the desired density on the object's surface, let them tune their relative position by simulating a repulsive force, and compute an associated high quality triangular mesh. The code we use for doing this is courtesy of Greg Turk, who uses the same process in his re-tiling algorithm [20].
2. We tile the surface with this mesh, i.e.:
   - we use surface geodesics to compute curved versions of the texture mesh edges;
   - we compute the set of geometric triangles covered by each of the resulting texture patches;
   - we compute the u, v coordinates of each geometric vertex with respect to the texture patch to which it belongs.
3. We use the algorithm described in Section 2.3 to consistently assign a specific texture sample to each patch of the texture mesh.
4. We render the object's geometry using the local u, v coordinates of the mesh vertices to map the texture samples.

A possible solution for implementing step 2 would be to adapt the set of methods introduced in [10]. In our current implementation, we rather compute geodesic curves using a standard length minimization process along a polygonal line, which is constrained to move onto the geometric mesh (the line is made of segments whose ends lie on the mesh edges). Then, we have developed a specific method, described below, for assigning u, v local coordinates to mesh vertices that lie on a texture patch without producing excessively large texture distortions. Alternative (and possibly better) solutions for implementing this part of the process can be found in [10, 4, 11].

3.2 Computing texture coordinates for mesh points

The texture mesh may have been designed at either a smaller or a larger resolution than the geometric mesh that describes the object. In the latter case, the local part of the surface that falls into a patch of the texture mesh (i.e., between three connected geodesics) may be highly curved. Computing u, v coordinates for mesh points included in this region must be done while trying to avoid texture distortions. Attention must also be paid to computing coordinates that exactly map the edges of the texture sample onto the geodesic edges of the patch, in order to avoid introducing discontinuities in the large scale texture at the junction between patches. Our solution is as follows:

To get rid of the border problem, we split the geometric triangles that are crossed by a geodesic, in order to be able to specify the exact texture coordinates along the texture patch edges[5].

The problem of computing a good u, v mapping inside each of the texture patches still remains. Since the problem is local, we have developed a simple solution that does not requires an optimization step. The basic idea is to use the three geodesic distances between a mesh vertex and the three edges of the texture patch to estimate the 'barycentric' coordinates of this vertex within the texture patch. Then, conversion into texture coordinates is immediate by analogy with the planar case. The algorithm develops as follows.

For each of the three edges of a texture patch:

1. Use a front propagation paradigm for computing the geodesic distances to the mesh vertices (see Figure 6):

   The front is implemented using a heap that stores the triangles whose three vertices are already provided with a distance value. The heap is initialized by the triangles that lie along the texture patch edge. Each front propagation step consists of taking the most reliable triangle within in the heap, i.e., the triangle which is closest[6] to this curve. The gradient of the distance within this triangle is computed. Then, for each neighboring triangle for which a vertex distance is still missing, we fold the distance gradient onto the plane of this triangle, and use the two already known values plus the gradient for evaluating the missing distance. This neighboring triangle is then inserted into the heap.

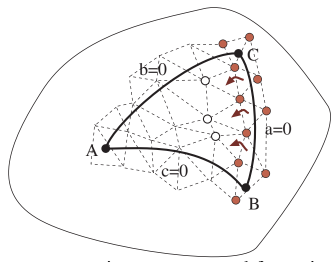

Figure 6: Front propagation process used for estimating the 'barycentric' coordinates of mesh vertices with respect to a texture patch. Distance values at pink points are already computed. The three next points for which the distance will be calculated are marked in white.

   In practice, there may be different ways of propagating distance to a given triangle, coming from several of its already computed neighbors. So we add a quality criterion to the distance value stored in the heap, and we modify a value each time we are sure quality will improve. The best quality is obtained when an vertex to be estimated falls between the two 'gradient lines' passing through the two known vertices. The estimation is less sure when it falls outside this band. The estimation is worst when the gradient is back-propagated, i.e., when the computed distance is smaller than the two known ones (then the result should only be used when no better evaluation is available).

2. Normalize all the distance values by dividing them by the value at the patch vertex that is opposite that edge.

Each vertex of the geometric mesh now stores three numbers $a, b, c \in [0, 1]$. To convert them into barycentric coordinates with respect to the three vertices of the texture patch, we divide them by their sum, so that $a + b + c = 1$. Lastly, we convert barycentric coordinates into texture coordinates with respect to the texture patch (the three corners of the image should map to $(0,0)$, $(\frac{1}{2}, \frac{\sqrt{3}}{2})$, $(1,0)$). The resulting mapping yields good results with only small texture distortions, as shown in Figure 7. However we have to keep in mind that avoiding excessive distortions is only possible 'locally': the surface's radius of curvature should not be too small relative to the patch size.

Figure 8 illustrates the control provided by texture mapping using a texture mesh: the scale of the texture can be increased while leaving the geometry of an object unchanged.

[5] The alternative solution that consists of keeping the triangles unsplit, computing a different texturing process (with half-transparent textures) for each texture patch that the triangle intersects does not work well: it does not provide enough control on the u, v coordinates near the edges of a texture patch, yielding texture discontinuities at this location.

[6] considering the maximum of the three distance values at vertices.

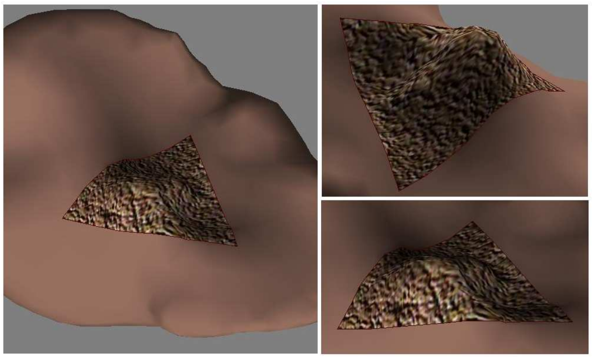

Figure 7: A texture patch mapped onto a curved region of a geometric model (views from three different viewpoints): texture distortions remain reasonable.

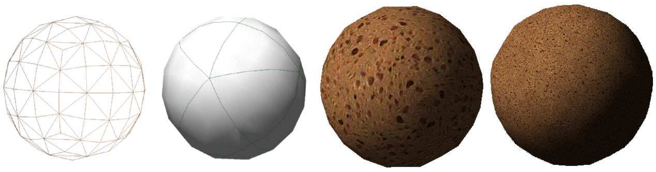

Figure 8: From left to right: the geometric mesh, which is rendered; the texture mesh, used for tuning the scale of the texture with respect to the object's geometry; the resulting textured sphere; the sphere with a finer scale texture.

4 Texture Samples Generation

4.1 Editing pictures or drawings

A first method for generating adequate sets of texture samples is direct editing, under a 2D paint system, of pictures or drawings. An example of hand-drawn texturing of a surface is depicted in Figure 9.

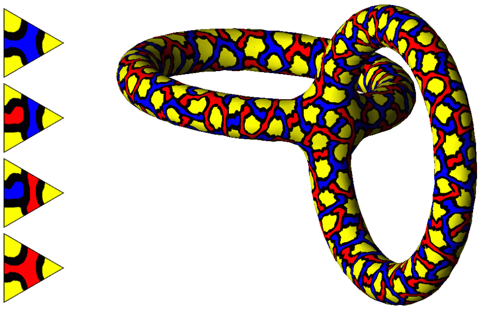

Figure 9: Texture samples drawn by hand, and the resulting image. Two different edge conditions (red, blue), both symmetric, are used.

Although it takes a certain amount of user-time, it is possible to edit real images in order to give them the required boundary conditions. The technique, that consists of copying and smartly pasting rectangular regions along edges and then eliminating texture discontinuities inside the sample, is almost the same as for square images. A single self-cyclical texture triangle, corresponding to a single symmetric edge, can be used. A more complex example of picture editing, where four different texture samples have been created for fitting the constraints associated with a single non-symmetric oriented edge, is depicted on Figure 10. The reference rectangular region figuring the oriented edge has been rotated by 180° or not when copied on the image borders, depending on which of the boundary conditions (E,E,E), $(E,E,\bar{E})$, $(E,\bar{E},\bar{E})$, or $(\bar{E},\bar{E},\bar{E})$ each of the four samples corresponds to.

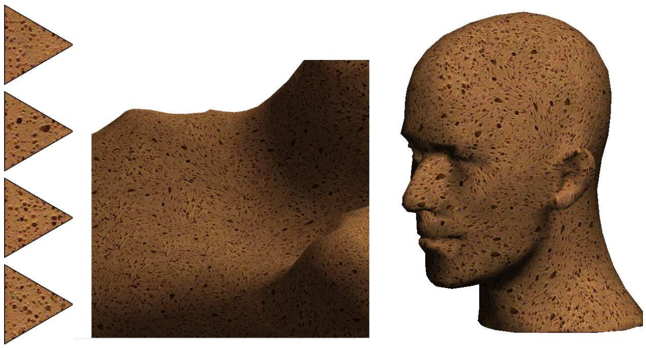

Figure 10: A set of texture samples designed by interactively editing an image of a sponge (left), and the resulting textured surface on a terrain and on a face.

We now describe two procedural synthesis techniques that can be used for automatically generating parameterized sets of texture samples thus saving user's time.

4.2 Extending Worley's algorithm

Worley's method [22] is an efficient approach for creating textures depicting small non-periodic cellular patterns such as rocks, scales, or living tissues. When applied in 2D, Worley's method basically consists of computing Voronoï diagrams of noise points randomly distributed on a plane. A square grid is used to accelerate the computation: a noise point is randomly chosen in each cell. Then, determining in which of the Voronoï region each pixel falls can be done efficiently, by only checking the noise points in the 9 closest cells. The portion of the plane covered by the noise points must be slightly larger than the square region to texture, in order to have nice Voronoï regions cross the edges. Worley's method combines the distances from a pixel point to the N nearest noise points to compute the texture value at the pixel (generally, $N \in [1..4]$). We only describe here how to deal with the nearest noise point; the other distance computations are adapted the same way.

Adapting Worley's technique for generating the texture on an equilateral triangle is easy: we just have to tile this triangle with a slightly larger triangular grid as depicted in Figure 11. A noise point is randomly chosen in each of the small triangles, and Voronoï diagrams are computed as in the standard case.

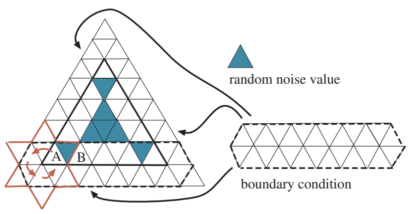

Figure 11: Triangular grid used for extending Worley's algorithm: Noise values that represent a given boundary condition along an edge are surrounded by a dashed line. Values which are the rotated copies of each other in order to maintain continuity constraints at a texture sample corner are indicated in pink.

For our pattern-based texturing application, we need to generate Worley triangles that obey specified boundary conditions along edges. More precisely, we want to be able to control the texture in the neighborhood of each triangle edge in order to ensure continuity between samples. Our solution is similar to the approach suggested in [17], and also to what we have described above for real image editing: we first generate the 'rectangular regions' representing each oriented edge. This rectangular region is implemented here by a two row grid storing the noise points that can influence the border neighborhood, on both parts of the boundary. Once again, we derive the complementary condition by rotating the rectangle that defines a boundary condition by 180°. Then, we duplicate the noise values into the adequate part of the grid, for each texture triangle that must obey this specific condition. Finally, the noise values of the inner unconstrained region of each triangle are chosen at random.

Particular attention must be paid to the achievement of texture continuity near texture sample corners. Two edges meet there, and the noise values they give to the texture triangle should thus be the same. As explained in Section 2.2, the solution is to define a texture with a given value and a zero gradient at these vertices. In terms of the algorithm above, this can be done by copying a rotated version of a given noise value into all the grid cells that surround a given vertex. In a naive implementation, this operation has to be done within two ranks of cells surrounding the corner (figured in pink), since the noise values in the second rank of cells may influence the texture gradient there. A trick for eliminating some possible visual artifacts due to symmetry is to restrict the range of possible noise point values in cells A and B (see Figure 11) so that the Voronoï region generated by the point in the blue cell between them will not intersect the edge of the texture sample. Then, this noise point value will have no influence on the texture at the vertex, enabling a (constrained) random choice for this cell. In Figure 11, all the noise points in the blue triangles can be chosen at random without spoiling boundary conditions.

Two examples of texture sample sets, and the resulting images they produce when mapped on a surface, are depicted in Figure 12.

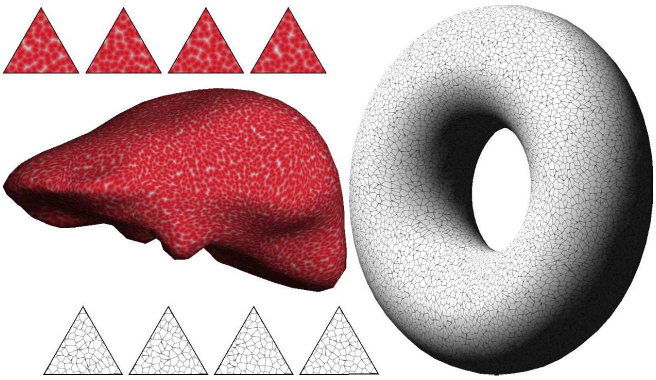

Figure 12: Sets of texture samples generated by our extension of Worley's synthesis technique, and the images produced by non-periodically mapping them onto a surface. *Left:* A human liver. *Right:* A china torus.

4.3 Extending Perlin's synthesis technique

Fractal noise based on Perlin's basis function [16] is a self-similar stochastic noise that has become a standard for generating objects that look like wood, marble, or the surface aspect of rock. One of its main features is to ensure continuity of both noise values and gradient at any point of an image (or of a volume, when the method is used in 3D, e.g. to figure smoke).

To adapt it to our texturing methodology, we first have to be able to generate the basis function on 2D equilateral triangles. Since Perlin's standard model is defined on a quadrangular grid, we modify the algorithm in the following way:

1. We first generate a pseudo-periodic noise function on a regular grid that tiles the equilateral triangle into sub-triangles (see Figure 13). This requires two steps:

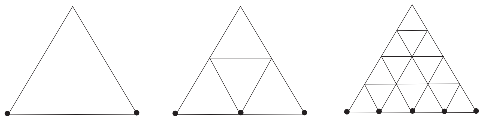

Figure 13: Triangular grids used for generating the noise function at different scales. The noise values that need to be fixed for ensuring boundary conditions are shown in bold.

   - as for textures based on Perlin's technique, we randomly associate a plane to each grid node, defined by its elevation above the node and by its normal vector;
   - we define the noise at any point inside the triangle as the barycentric interpolation of the distances to the three planes that are associated to the vertices of the small triangular cell where the point lies.

2. We define the final noise value at a pixel as the sum of instances of the pseudo-periodic noise function defined above, applied at different scales thanks to recursive subdivision of the triangular mesh, with a scaling factor that is the equal to the scale. This gives the fractal aspect to the noise.

3. The value obtained is used as usual as a seed or a perturbation to define the texture value at the pixel.

Modifying this algorithm to ensure a set of given boundary conditions around each triangle is easy: we just have to model a boundary condition as the set of noise values that control the texture values and derivatives along an edge. These values are those indicated in bold in Figure 13. We then duplicate these boundary values onto the adequate side of the grid, for all the texture samples that have to obey this specific boundary condition. Ensuring continuity at the three corners of texture sample is done as usual by giving the same mean value and zero gradient to the texture there, i.e., using specific noise values at each vertex. As in original Perlin's algorithm, all random values are precomputed in a (small) hash table, and no copy is done: instead we compute at any location which index in the random table should be accessed. To define the same control value at the vertices of two edges, one just has to ensure that the same index is produced, and rotate the built random normal vector on the fly (because adjacent triangles do not use the same frame).

Three examples of texture sample sets and the resulting images they produce are depicted in Figure 14. Note that computing procedural textures on triangular domains while ensuring continuity constraints can also be used on the fly for other kinds of applications. For instance, we have used an algorithm close from above for generating at rendering time a displacement texture modeling the crust of an evolving lava-flow without having to parameterize the flow surface [18].

5 Conclusions & Future Work

We have presented a general framework, based on triangular texture tiles, for texturing an arbitrary surface at low rendering time and memory cost. Our method has been designed for covering the surface with an homogeneous non-periodic texture such as those that can be found on many natural objects. The main features of our approach are the following: the texture can be applied at any scale with respect to the object geometry, and whatever the quality of the geometric mesh; no singularity is generated whatever the surface topology, and distortions are minimized. Moreover, using the method demands little user work, by avoiding redundancies. We describe how to use hand-drawing and real images, and we provide two automatic texture synthesis methods that adapt Perlin's and Worley's algorithms to a triangular domain. Adapting other

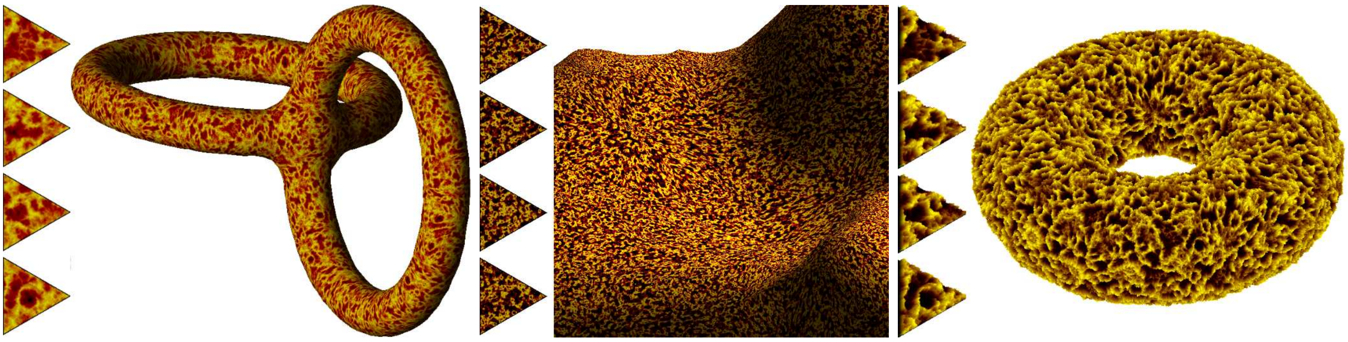

Figure 14: Three sets of texture samples generated by our extension of Perlin's synthesis technique, and the images produced by mapping them onto a surface. In the image on the right, Perlin's texture is used as displacement map, encoded with an OpenGL implementation of volumetric textures [13].

texture synthesis techniques in the same way, such as those based on a pyramidal analysis of real textures [9, 2] should be easy, as filtering kernels and pyramidal constructions can directly be 'translated' to triangular grids. Lastly, our framework is compatible with real-time applications, since the result of the texturing can be represented using classical geometric object formats.

Maintaining C^1 continuity of the texture across the surface has been achieved by building texture samples which obey specific boundary conditions, and whose border is mapped exactly onto the geodesic curves that tile the surface onto texture patches. In the current implementation, we ensure the second constraint by splitting the geometric triangles that intersect a geodesic, in order to have enough points for locally controlling the mapping. This is not a problem when large scale texture triangles are used since this splitting process will not greatly increase the number of triangles to render. However, in the case of almost flat surfaces built with a few large triangles, and that need being textured at a small scale, the tiling of the geometry may yield a high increase of rendering time. A solution would be to use a level of details approach for defining texture samples: large samples containing very small patterns would be recursively created by assembling smaller ones. Then, the former would be mapped onto a larger scale texture mesh defined on the surface, which would not split the geometry excessively. Conversely, this would provide a solution to correctly map geometric areas that are smaller than the initial sample size (such as handles or legs), without having to decrease the sample size elsewhere on the surface.

Our future work also includes the introduction of user-controlled pattern anisotropy, which is an important feature of many natural textures. Using anisotropic patterns directly in the scheme we have presented would not be a good idea, since there is no way to ensure continuity of characteristic directions between texture patches. Our idea is rather to rely on the texture mesh itself, to model the kind of anisotropy that corresponds to the stretching of an isotropic pattern: A user-defined tensor field would be used for locally specifying the amount and direction of desired anisotropy all over the surface. This field would be used to influence the generation of texture mesh control points and produce an anisotropic distribution (in the same spirit, Turk suggests in [20] to use the surface curvature). Finally, mapping the usual isotropic texture samples onto this texture mesh would result into adequately deformed patterns, with continuous directional features.

Acknowledgments

We wish to thank Greg Turk for providing his retiling code and Alexandre Meyer for adapting his volumetric textures to our framework. Thanks to David Bourguignon for creating the sponge texture samples, to Brian Wyvill and Georges Drettakis for rereading this paper, and to Jean-Dominique Gascuel for his help in video editing.

References

[1] Chakib Bennis, Jean-Marc Vézien, Gérard Iglésias, and André Gagalowicz. Piecewise surface flattening for non-distorted texture mapping. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH 91 Proceedings)*, volume 25, pages 237–246, July 1991.

[2] Jeremy S. De Bonet. Multiresolution sampling procedure for analysis and synthesis of texture images. In Turner Whitted, editor, *SIGGRAPH 97 Conference Proceedings*, pages 361–368. ACM SIGGRAPH, Addison Wesley, August 1997.

[3] H. S. M. Coxeter, M. Emmer, R. Penrose, and M. L. Teuber, editors. *M. C. Escher: Art and Science*. North Holland, Amsterdam, 1986.

[4] Matthias Eck, Tony DeRose, Tom Duchamp, Hugues Hoppe, Michael Lounsbery, and Werner Stuetzle. Multiresolution analysis of arbitrary meshes. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, pages 173–182. ACM SIGGRAPH, Addison Wesley, August 1995.

[5] M.c. Escher biography and annotated gallery. http://www.erols.com/ziring/escher_gal.htm.

[6] Andrews Glassner. Andrew Glassner's notebook: Penrose tiling. *IEEE Computer Graphics and Applications*, 18(4):78–86, July/August 1998.

[7] B. Grünbaum and G.C. Shephard, editors. *Tilings and Patterns*. Freeman, New York, 1986.

[8] Pat Hanrahan and Paul E. Haeberli. Direct WYSIWYG painting and texturing on 3D shapes. In Forest Baskett, editor, *Computer Graphics (SIGGRAPH 90 Proceedings)*, volume 24, pages 215–223, August 1990.

[9] David J. Heeger and James R. Bergen. Pyramid-based texture analysis/synthesis. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, pages 229–238. ACM SIGGRAPH, Addison Wesley, August 1995.

[10] Aaron Lee, Wim Sweldens, Peter Schröder, Lawrence Cowsar, and David Dobkin. MAPS: Multiresolution adaptive parametrization of surfaces. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, pages 95–104. ACM SIGGRAPH, Addison Wesley, July 1998.

[11] Bruno Lévy and Jean-Laurent Mallet. Non-distorted texture mapping for sheared triangulated meshes. In Michael Cohen, editor, *SIGGRAPH 98 Conference Proceedings*, pages 343–352. ACM SIGGRAPH, Addison Wesley, July 1998.

[12] Jérôme Maillot, Hussein Yahia, and Anne Verroust. Interactive texture mapping. In James T. Kajiya, editor, *Computer Graphics (SIGGRAPH 93 Proceedings)*, volume 27, pages 27–34, August 1993.

[13] Alexandre Meyer and Fabrice Neyret. Interactive volumetric textures. In G. Drettakis and N. Max, editors, *Rendering Techniques'98, Eurographics Rendering Workshop*, pages 157–168. Eurographics, Springer Wein, July 1998.

[14] Hans Køhling Pedersen. Decorating implicit surfaces. In Robert Cook, editor, *SIGGRAPH 95 Conference Proceedings*, pages 291–300. ACM SIGGRAPH, Addison Wesley, August 1995.

[15] Hans Køhling Pedersen. A framework for interactive texturing operations on curved surfaces. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, pages 295–302. ACM SIGGRAPH, Addison Wesley, August 1996.

[16] Ken Perlin. An image synthesizer. In B. A. Barsky, editor, *Computer Graphics (SIGGRAPH 85 Proceedings)*, volume 19, pages 287–296, July 1985.

[17] Jos Stam. Aperiodic texture mapping. Technical Report R046, European Research Consortium for Informatics and Mathematics (ERCIM), January 1997. http://www.ercim.org/publication/technical_reports/046-abstract.html.

[18] Dan Stora, Pierrer-Olivier Agliati, Marie-Paule Cani, Fabrice Neyret, and Jean-Dominique Gascuel. Animating lava flows. In *Graphics Interface 99*, June 1999.

[19] Greg Turk. Generating textures for arbitrary surfaces using reaction-diffusion. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH 91 Proceedings)*, volume 25, pages 289–298, July 1991.

[20] Greg Turk. Re-tiling polygonal surfaces. In Edwin E. Catmull, editor, *Computer Graphics (SIGGRAPH 92 Proceedings)*, volume 26, pages 55–64, July 1992.

[21] Andrew Witkin and Michael Kass. Reaction-diffusion textures. In Thomas W. Sederberg, editor, *Computer Graphics (SIGGRAPH 91 Proceedings)*, volume 25, pages 299–308, July 1991.

[22] Steven P. Worley. A cellular texturing basis function. In Holly Rushmeier, editor, *SIGGRAPH 96 Conference Proceedings*, pages 291–294. ACM SIGGRAPH, Addison Wesley, August 1996.

Animating Lava Flows

Dan Stora Pierre-Olivier Agliati Marie-Paule Cani Fabrice Neyret Jean-Dominique Gascuel
iMAGIS[†]-GRAVIR / IMAG
BP 53, F-38041 Grenoble cedex 09, France
Marie-Paule.Cani@imag.fr

Abstract

Animating lava flowing down the slopes of a volcano brings several challenges: modeling the mechanical features of lava and how they evolve over time depending on temperature; computing, in a reasonable time, the interactions inside the flow, and between the flow and a complex terrain data-base; and lastly, rendering the visual aspect of the flow. The methods described rely on smoothed particles governed by a state equation for animating the flow. We adapt this model to the animation of lava by linking viscosity to a temperature field and by simulating heat transfers. We propose particular data-structures that lead to linear computational time with respect to the number of particles. Lastly, we study a model based on a color-and-displacement procedural texture controlled by the flow for the realistic rendering of lava.

Key words: Natural phenomena, animation of fluids, particle systems, procedural textures, implicit surfaces.

1 Introduction

Modeling the richness and complexity of nature is a constant challenge for Computer Graphics. In addition to realistic shapes and rendering, Computer Animation techniques try to generate consistent motion and deformations. A wide range of natural phenomena have been animated over the past few years. They include gaseous phenomena such as fire or smoke [13, 18, 8], liquids – from waves to droplets [9, 12, 16, 7, 10], and the simulation of soft terrains, from footprints to landslides [4, 19, 11].

This paper addresses another complex problem: the animation of lava flows. Our aim is to provide visual realism in both motion and rendering. In order to generate complex trajectories due to the interactions between lava-flows and a complex terrain database, we need to model the main mechanical features of lava and the way they evolve over time. Another difficulty is to compute the simulation at acceptable rates. Lastly, we would like to provide a convincing visualization of lava "skin" appearance, even though we are not attempting to use a simulation approach for this part of the phenomenon.

Recently, several movies featuring volcanos have been launched, showing the potential impact of lava animation in the field of special effects. However, despite the advertising that announced the use of Computer Graphics techniques, 'Dantes Peak' and 'Volcano' mostly use regular special effects, including scale models for lava. As far as we know, Computer Graphics was used for the 2D compositing of these models with fire images, rather than for computing a graphical simulation of the flows.

Geologists have been interested for years in the simulation of volcanic explosions and lava flows. However their concern is not producing realistic images, but rather computing a realistic account of flow and ash coats (e.g. in a vertical section), and measuring the way some global physical values vary over time. Thus, their models (see `http://www-geo.lanl.gov/wohletz/Erupt.html`) do not seem directly usable for a Computer Graphics simulation.

In terms of physically-based simulation, lava can be seen as a liquid whose viscosity increases exponentially when the material cools down. Previous solutions to the animation of liquids range into two main categories: Eulerian approaches, that consist of discretizing space into a fixed 3D mesh and studying how physical fields evolve at the mesh nodes [12, 7]; and Lagrangian approaches that consist of following the motion of punctual mass elements, called particles, that sample the liquid. The second approach seems more convenient in the case of lava-flows since the use of particles enables us to attach a deformable lava skin to the flow without introducing extra marker elements. In addition, particle systems have successfully been used for modeling a wide range of behaviors that includes very viscous substances, and for animating transitions between solid and liquid states, using a temperature parameter [20, 21]. Among particle systems, the smoothed particle model introduced in [15, 6] plays an original part, since inter-particle forces are not tuned by hand, but are rather derived from a state equation that defines the macroscopic behavior of the material. This model has been applied to the simulation of mud

[†]iMAGIS is a joint project of CNRS, INRIA, Institut National Polytechnique de Grenoble and Université Joseph Fourier.

slides [11]. There has been no attempt yet, to the authors knowledge, to modify its mechanical features over time.

The approach described in this paper relies on an extension of the smoothed particle model. We control the evolution of the lava dynamics over time by providing each particle with a viscosity parameter, which depends on temperature. During a simulation, heat transfers at the surface of the lava and inside the flow causes the temperature to decrease, enabling the transition from a low-viscosity behavior to a highly viscous one. The visual aspect of the lava changes accordingly, thanks to a coating of the particles that controls local color and granularity of the lava skin as functions of temperature.

Section 2 details the model we have designed for the physical features of lava. Implementation issues for the efficient computation of a lava-flow, including the design of convenient data structures, are discussed in Section 3. Section 4 describes our method for generating a visually realistic rendering of the animation.

2 Physically-based Modeling of Lava

2.1 Modeling Lava

Natural lava-flows present a wide diversity of morphologies and structures. Among other factors, the topography of terrain, and the flow rate out of the crater affect the lava spread. These factors will be considered in Section 3 for the practical computation of lava-flows. Studies of the main mechanical features of lava [3] establish that:

- Viscosity depends on the chemical composition of the lava;
- For a given chemical composition, viscosity increases exponentially when the temperature decreases (see Figure 1, from [3]).
- The mass-density of lava remains quasi-constant until it commutes to the solid state. The value of mass-density for basaltic lavas is around 2500 $kg.m^{-3}$.

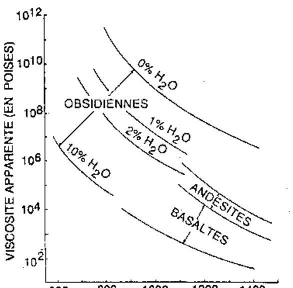

Figure 1: Viscosity as a function of temperature, for lavas of different chemical composition.

Our aim is to simulate the lava flowing process. We are looking for a general model of a viscous fluid, robust enough for enabling high viscosity changes inside the material. These changes will take place in space – as viscosity is not constant along a lava flow at a given time – and in time, since lava is cooling down during the simulation. During the simulation, the flow's mass density should be maintained around 2500 $kg.m^{-3}$.

The next section presents the deformable model that we have used as the background for our simulations. This model uses a particle system for modeling the flow, which is also convenient for rendering lava skin as will be shown in Section 4. Its main feature is that, contrary to standard particle systems, it sets the material to a specified mass-density at rest, thanks to a state-equation that describes the desired macroscopic behavior.

2.2 Smoothed particles maintaining a constant rest-density

Our model is an extension of the smoothed-particles simulation method developed by Mathieu Desbrun[1] [6]. This section reviews this previous work.

Smoothed particles simulate a substance whose physical characteristics are governed by a *state equation*. The following equation states that the pressure[2] field P inside the material is proportional to the difference between the current mass-density ρ and a given rest value ρ_0:

$$P = k(\rho - \rho_0)$$

where the parameter k represents a stiffness coefficient.

During a simulation, the material is sampled by "smoothed particles". Each of them represents a sample of mater of fixed mass m_i which is distributed in space around its current location x_i according to a smoothing kernel W_h (parameter h controls the extend of the kernel's support). The particles sample the values of continuous physical fields such as pressure P and mass-density ρ inside the material (so P_i and ρ_i respectively denote pressure and density at particle i). Interpolation yield the value and gradient of these physical fields anywhere in space:

$$\begin{aligned} f(x) &= \sum_j f_j \frac{m_j}{\rho_j} W_h(x - x_j) \\ \vec{\nabla} f(x) &= \sum_j f_j \frac{m_j}{\rho_j} \vec{\nabla} W_h(x - x_j) \end{aligned}$$

where f_i is the value of field f at particle i.

This can be used for deriving the expression of internal pressure forces exerted on particle i from the expression

[1]The formalism was originally inspired from works in astrophysics [15] on "Smoothed Particles Hydrodynamics" (SPH).

[2]As in [6], the term "pressure" we use is not totally adequate: P is rather a *difference of pressure* with a given reference value, thus it can take either positive or negative values.

of pressure given by the state equation [6]:

$$\vec{F}_{P_i} = \sum_{j \neq i} -m_i m_j \left(\frac{P_i}{\rho_i^2} + \frac{P_j}{\rho_j^2} \right) \vec{\nabla}_i W_h^{ij}$$

where $\vec{\nabla}_i W_h^{ij}$ is the gradient of $W_h(x_i - x_j)$. The density field is computed by integrating pressure changes over time from its initial value ρ_0 (corresponding to the initial particle distribution) using the continuity equation:

$$\rho \, div(V) + \frac{d\rho}{dt} = 0$$

The component of the interaction force $\vec{F}_{P_i}$ which is exerted between particle i and particle j can be either positive or negative according to the sign of the over-pressures P_i and P_j. It has been shown in [6] that these forces are similar to Lennard-Jones attraction-repulsion forces (ie. long-range attraction and short-range repulsion) that are commonly used in physically-based particle systems [14]. One of the advantages of SPH is that no local user-specified parameter is needed, since they are directly derived from the global state equation.

As with any particle system, the animation is computed by integrating, for each particle, applied forces over time. Non-conservative forces that model viscosity inside the material, as well as external forces such as gravity and collision forces with obstacles, are added to pressure forces. The smoothed particles representation provides an expression for the maximal time step at which each particle should be simulated, computed from Courant's condition [6]. The use of an adaptive time-step based on this minimal rate ensures stability of computations.

The main benefits of this model is that the user defines the macroscopic behavior of the material through the state equation. As a consequence, the same substance may be simulated with variable accuracy, by simply tuning the number of particles that sample it (this is not the case for particle systems based on Lennard-Jones interaction forces, such as those used in [20, 21], since the material behavior then depends on both forces parameters and the scale at which these specific forces are used). Another advantage of this model is that we are able to control the rest mass-density, whose value is well known for lava. In practice, we model lava by taking the same smoothing kernel that in [6] and by setting the parameters of the state equation to[3]:

$$\rho_0 = 2500 \; kg.m^{-3} \qquad k = 90000$$

The above model is not sufficient for modeling all the physical features of lava listed in Section 2.1. In the next two sections we extend the model to include variable viscosity and to simulate heat transfers over time.

2.3 Modeling viscosity inside a lava-flow

Viscosity plays a very important part in the dynamics of lava flows. The model for a viscosity force must be chosen with care. The expression of viscosity force used in [15, 6], originally designed for cosmologic fluids, was:

$$\vec{F}_{v_i} \; = \; -m_i \sum_{j \neq i} m_j \Pi_{ij} \vec{\nabla} W_h^{ij}$$

where Π_{ij} is a function of the relative speeds of particles i and j, and of the distance between them:

$$\Pi_{ij} = \begin{cases} \frac{hc(v_{ij} \cdot x_{ij})}{\rho_{ij}(x_{ij}^2 + h^2/100)} & \text{if } v_{ij} \cdot x_{ij} < 0 \\ 0 & \text{if } v_{ij} \cdot x_{ij} \geq 0 \end{cases}$$

where c is the speed of sound[4] inside the material and $v_{ij} = v_j - v_i$, $x_{ij} = x_j - x_i$, $\rho_{ij} = (\rho_i + \rho_j)/2$.

This expression is not convenient for us, even for modeling viscosity at a given temperature. Firstly, the force contribution from a given neighbor particle j becomes zero when the distance between the two particles increases. Secondly, even when the speed vectors are not oriented along the segment between the particles, this force contribution is always applied along this segment. Lastly, the parameters are not easy to calibrate due to the complex formulation of Π_{ij}.

We rather use a more intuitive and simpler expression for viscosity forces, aimed at modeling the variations of the velocity field at the neighborhood of a particle. The value of the force is proportional to the difference between the particle's speed and the mean speed around it:

$$\vec{F}_{v_i} \; = \; \alpha_v \frac{m_i}{\rho_i} \sum_{j \neq i} \frac{m_j}{\rho_j} v_{ij} W_h^{ij}$$

Since the viscosity of lava increases exponentially when temperature decreases, we set the parameter α_v to:

$$\alpha_v \; = \; b \exp(-aT_i), \; a > 0, \; b > 0$$

where T_i is a new parameter representing temperature, which is associated with each particle (we use $a = 220$ and $b = 0.5$ in our simulations). Next we have to design a simulation method for computing the local variations of the temperature field during a simulation.

2.4 Simulation of heat transfers

Two kinds of heat transfers should be taken into account during a simulation: those that are internal to the lava flow, and those that are external (i.e. transfers with the air and with the ground).

[3] the higher k is, the more incompressible the material will be, but the required time step is smaller.

[4] i.e. the speed of the fastest propagation wave.

As in [20, 21], we model heat transfers inside the material by integrating the general heat equation:

$$\frac{dT}{dt} = k\triangle T$$

However, the way we do this is different from previous works, since we have to express it is the smoothed particles formalism. Since SPH cannot be used for computing second derivatives due to the smoothing effect of the kernel W_h, directly computing $\triangle T$ (the Laplacian of temperature) is difficult. In practice, we compute $\triangle T$ for each particle using:

$$\vec{\nabla} T_i = \sum_{j \neq i} m_j \frac{T_j - T_i}{\rho_j} \vec{\nabla} W_h^{ij}$$

$$\triangle T_i = \sum_{j \neq i} m_j \frac{\vec{\nabla} T_j}{\rho_j} \vec{\nabla} W_h^{ij}$$

Thanks to temporal coherence and very slow speed of the heat transfers inside the material, we can use the value of $\vec{\nabla} T_i$ stored at a given time step for computing $\triangle T_i$ at the subsequent time step, in a single list traversal.

Now we need to model heat transfers that take place at the surface of the lava flow, including both lava-air and lava-ground interfaces. The term representing the time variation of temperature for a particle located at the surface of the flow should[5] be proportional to the difference between its temperature and the temperature of exterior medium, proportional to the size of the contact surface, and inversely proportional to the local mass density:

$$\left(\frac{dT}{dt}\right)_{ext_i} = k_{ext}(T_i - T_{ext})\frac{r_i^2}{\rho_i}$$

where r_i, the approximate radius of the small sample of lava the particle models, is computed from:

$$\frac{4}{3}\pi r_i^3 = \frac{m_i}{\rho_0}$$

In our simulations, we set the temperature of basaltic lava to 1200 $°C$ when it spreads out of the crater. It goes down to about 900 $°C$ during the simulation.

3 Computing animations of lava flows

3.1 Data structures

The bottleneck of any simulation of a particle system is the computation, at each time step, of the interactions between neighboring particles. In the smoothed particles formalism, interactions (through pressure and viscosity forces) take place as soon as the distance between two particles is smaller than $2h$, where h is the radius of their smoothing kernel. Since the model maintains the material around a given rest density, the number of neighbors always stays around a given small value (between 20 and 30 in our simulations). However, in a brute force implementation, the neighbor search would be quadratic in the number of particles, which would lead to unacceptably low performances. A well-known solution consists of using a 3D grid that stores the connected list of the particles that lie inside each voxel. In the specific case of lava spreading out of a volcano, this solution cannot be used directly, since the 3D grid would be too large.

Our solution exploits the fact that while lava spreads far away from the crater, the local vertical range of the flow always remains small compared to the size of the terrain (see Figure 2 (a)). Thus, we use a predefined grid of a large size in the horizontal plane, but with only a few elements per vertical column. The vertical scope of each column is adapted in order to cover the local vertical range of the flow over this precise point of the horizontal plane (see figure 2 (b)). During neighbor search, only particles lying in a grid voxel that is closer than $2h$ from the current particle are considered. Since lava approximately maintains its rest density everywhere, calibrating the grid in order to have an almost constant number of particles per occupated voxel is easy. This strategy provides a simulation time which is close to linear with respect to the number of particles.

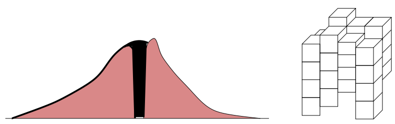

Figure 2: (a) Lava spreading out of a volcano (cross section). (b) Data structure for neighbors search.

The terrain is defined by a digital elevation model (DEM), i.e. a 2D grid where altitudes are stored. The use of a DEM makes easier the detection of collision between a particle and the terrain, since it only requires the computation of the terrain's altitude (through bilinear interpolation) at the particle's projection onto the horizontal plane. In practice, collision detection is performed only for particles that are at the surface of the flow. The computation of this subset of particles is also necessary to apply the heat transfer equations. The next section proposes a criteria for performing this computation.

3.2 Characterization of particles at the flow surface

An original feature of highly deformable materials compared with solid deformable bodies, is that the set of small matter elements that lie near the surface may change over time. This is particularly obvious when a

[5] This is just a simple model convenient for a Computer Graphics applications. We do not pretend to simulate the real physics of lava cooling such as degazing and radiative transfers.

flow separates into several disconnected components, or when some components merge. In our simulations, the set of particles lying near the surface has to be recomputed at each time step. We need a robust criteria for characterizing these particles.

The gradient of mass-density at a particle location indicates the direction where the highest neighbor mass is found. So the border should be searched for in the opposite direction. Our criteria states that a particle :

is on the surface if the quotient between the mass of its neighbors located in the half space oriented by $-\vec{\nabla}\rho_i$ *and the total mass of all neighbors is under a threshold.*

Since this criteria will not work when the lava has completely spread out (the gradient of mass-density is in the particle's plane when all the particles are in contact with a flat part of the terrain) we add a second criteria:

a particle is at the border if that particle and all its neighbors are approximately located in the same plane.

3.3 Simulation process for lava flows

At each integration time step:

- Generate some more particles inside the crater, according to the desired flow rate.
- For each particle:
  1. Compute the list of neighbors.
  2. Use it to compute ρ_i, $\vec{\nabla}T_i$, and $\triangle T_i$.
  3. Compute pressure and viscosity forces.
  4. Integrate the equation of motion according to the set of external and internal forces.
  5. If the particle is on the surface of the flow:
     - test for collisions with the ground,
     - compute the heat transfer with the exterior.
  6. Compute the heat transfers with neighboring particles.
- If necessary, modify the value of the integrate step in order to maintain Courant's condition.

We have seen that in order to maintain lava in a quasi-incompressible state, our simulations are computed with a high value for k. Courant condition then leads to about 64 integration steps between two frames, using integral integration schemes for increasing stability. Even in this case, computational time is reasonable: On a SGI O^2 workstation, an image is generated every 20 seconds for a simulation of 1000 particles, and every two minutes for a simulation of 6000 particles. If we decrease k to a less realistic value such as $k = 100$, less integration steps are needed between two images, so we obtain one image per second for 1000 particles.

Figure 4 shows some steps of a lava flow for our standard high value for k. Particles are displayed as spheres of radius r_i. Their color varies with temperature. Displaying particles as spheres is adequate for carefully following each particle's motion, but does not, however, yield the visually realistic rendering we need.

4 Visually realistic rendering

4.1 Trying to obtain the aspect of basaltic lava

Figure 7 depicts a real basaltic lava flow of type "aa", whose particularity is the formation of clinkers during the slow transition between fluid and solid states. These clinkers are made of the lava foam that has cooled down quicker that the rest of the flow, and merged into regularly spaced clusters. In this paper, we are trying to render this specific kind of lava, whose roughness, size of clinkers, and color highly depend on temperature.

At first sight, obtaining a similar visual aspect for our synthetic lava flows looks like a challenge. The most current solution for defining a surface around a set of particles consists of computing an iso-surface of a sum of scalar fields generated by the particles [1, 20, 21, 5]. This solution easily deals with topology changes (separations and fusions) but is not convenient for our application since implicit surfaces generate very smooth shapes [2].

Motion and deformations of our lava flows are simulated at a relatively large scale, then it seems obvious that more geometric complexity should be generated. Performing a micro-simulation of the lava skin mechanics would be difficult, computationally intensive, and is not really useful since our only aim is visual realism. Thus we must deal with this smaller scale by providing a geometric dressing of the larger one. Perturbating the surface with a stochastic continuous noise, such as Perlin's noise [17], seems a good approach. A first idea would be to define a 3D noise depending on temperature in a local frame linked to each particle, and to use it to deform the implicit primitive defined by the particle. The surface of the flow would then be computed as an iso-surface for the sum of perturbated fields. However, this approach would be extremely time consuming due to the resolution needed for polygonizing the surface. Moreover, when two neighboring particles move at different speeds, the addition of noise contributions would probably produce aliasing artifacts (such as interferences).

Our approach consists in generating a procedural displacement map, together with a color texture, on the smooth implicit-surface generated by a standard isotropic fields around the particles. Both color and roughness of the texture are function of the local temperature of the flow. One of the challenges is to maintain temporal coherence when an animation sequence is rendered: each surface detail may continuously deform and change its color, but must consistently follow the flow.

4.2 A lava skin that follows the flow

Texture is going to be generated on triangles, so the first point is to tessellate the implicit surface. This cannot be done by using a standard surface tiling algorithm [2] at each time step since, because of our need of temporal coherence, each triangle should *deform and move with the underlying flow*.

Our solution is the following:

1. We associate a sample point on the implicit surface to each particle that verifies the flow surface criteria of Section 3.2. This is done by searching for the iso-surface in the direction of the field's gradient on the particle.
2. We compute the Voronoï diagram of these sample points: for each of these points, we sort sample points that correspond to neighboring particles in counterclockwise order with respect to the surface's normal, and select those that are contributing to the Voronoï region.
3. We tessellate each Voronoï region into triangles that turn around the sample point defining the region.

This results into triangles that both tessellate the flow surface, define the neighborhood of a particle on the surface, and consistently follow the flow (see Figure 3).

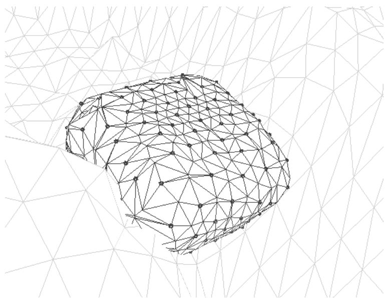

Figure 3: Tesselated Voronoï regions on a flow surface.

4.3 Generating a procedural texture

The idea is to generate a lava skin clinker into each Voronoï region. Since the sample point at the center of the region follows a given flow particle, the clinkers will be adequately carried by the flow.

Clinkers and lava skin between them are modeled by a texture that includes both local color and also a displacement map controlling surface altitude and roughness. We compute the displacement map as the sum of two terms (see Figure 6):

- a given 3D profile defining the shape of a clinker,
- a stochastic altitude perturbation modeling the local surface's roughness.

Continuity conditions at the border of triangles must be maintained carefully. This includes both color and displacement G^1 continuity (i.e. geometric continuity of derivatives).

Modeling specific profiles whose shape and altitude depend on temperature, and whose support is included into each Voronoï region is straightforward. Color consistency and continuity can easily be obtained by associating a given color to each altitude, with a color map depending on the local temperature of the flow. The next paragraph concentrates on the most difficult part of the process, ie, defining a G^1 stochastic noise which gives the desired visual effect.

4.4 Continuous Perlin's noise on a triangular mesh

Perlin's noise [17] is a stochastic auto-similar G^1 noise that has become a standard for generating objects that look like wood, marble, fog, or rock surface. One of its main features is to maintain continuity of both noise and noise derivatives. The idea is to use it for defining a 2D displacement texture that will serve as a perturbation of altitude over each lava skin triangle.

However, we need to generate the noise on a 2D triangular mesh, since it is easier to guarantee continuity through two adjacent texture patches if the noise control points fall exactly on their boundary. Perlin's standard model being defined on a quadrangular grid, we modify the algorithm in the following way:

1. we generate a pseudo-periodic noise function on a regular grid of equilateral triangles by:
   - randomly associating a plane to each grid node, defined by its elevation above the node and by its normal;
   - defining the noise at any point inside the mesh as the barycentric interpolation of the distances to the three planes associated to the vertices of the triangle where the point lies.
2. we define the final noise giving the altitude of a mesh point as the sum of instances of the pseudo-periodic noise defined above, applied at different scales thanks to recursive subdivisions of the triangular mesh. This gives a fractal aspect to the noise.

The color and displacement texture we obtain is depicted at the top right of Figure 6. Since the noise function has been generated on a regular grid made of equilateral triangles, applying it on the non-equilateral triangles resulting from the tessellation of Voronoï regions may locally alter the isotropy of the noise. In practice, we reduce this problem by suppressing the very thin triangles (by merging their edges) that may appear around a sample point.

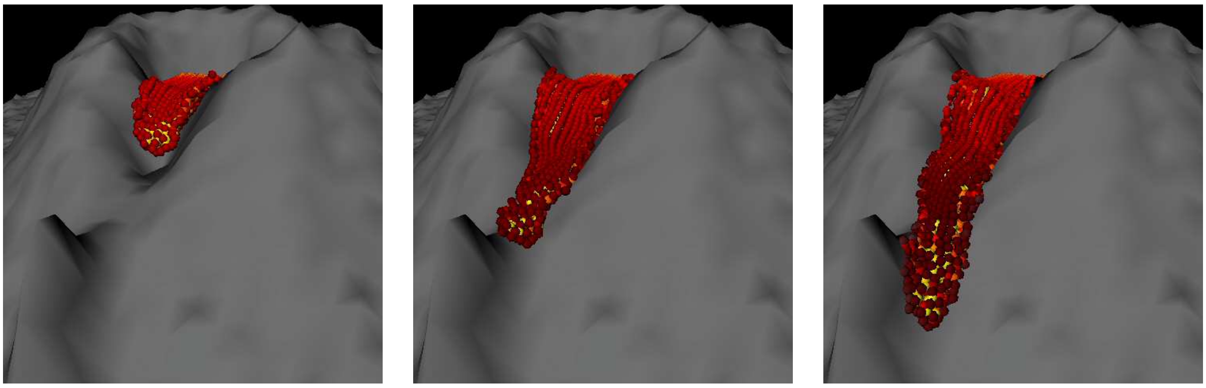

Figure 4: An example of lava-flow where particles are displayed as spheres. About 3000 particles have spread out of the volcano in the last image.

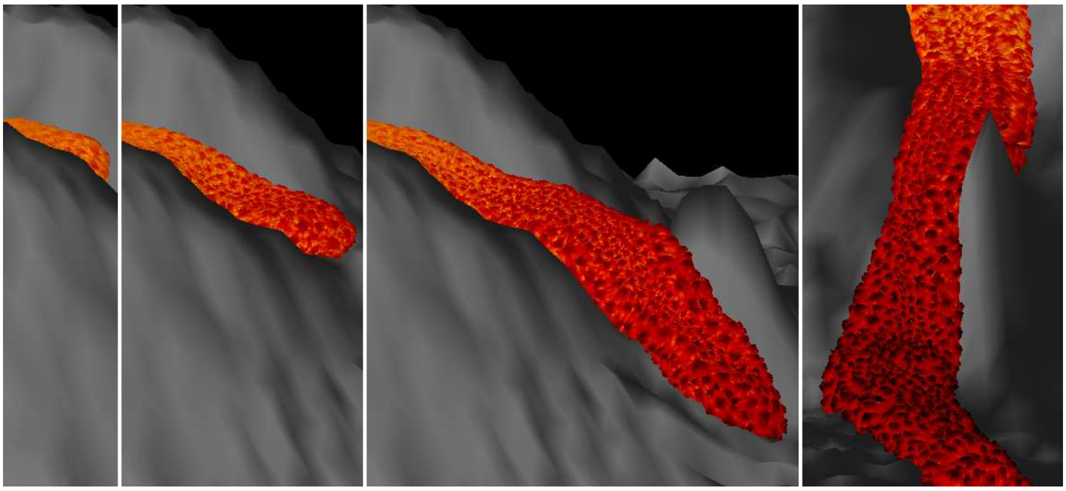

Figure 5: A textured lava flow.

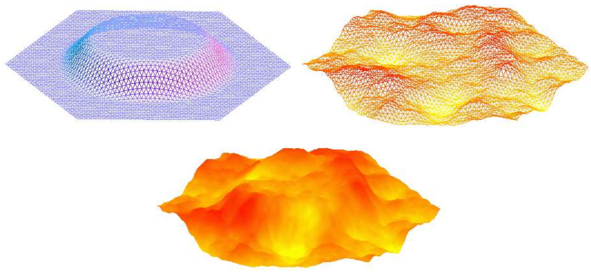

Figure 6: Top left: a clinker's profile; top right: stochastic perturbation modeling the surface's roughness; bottom: the resulting texture.

Figure 7: A real lava flow for basaltic lava of type "aa".

Computational time has been measured on a test scene containing one clinker made of 6 triangles (see figure 6, bottom) covering a video-sized screen and subdivided 32 times (so that 6144 very small triangles are drawn). Computing the texture takes about 0.15 seconds per frame on a SGI O^2 workstation. As small triangles that sample the texture should always be of the same size (about a few pixels), the subdivision depth decreases when the number of polygons increases, so the rendering time remains of the same order of magnitude for any scene (less than a second per frame).

5 Conclusion

This paper has focussed on the animation and rendering of lava flows. Figure 5 depicts some steps of an animation where lava flows down the slopes of a volcano. Animation sequences are available from `http//www-imagis.imag.fr/LAVA/`.

Lava flows represent a challenge for physically-based animation, since the mechanical features of lava change over time. This change is governed by a temperature field, that needs to be animated. The first contribution of this paper is to show that the smoothed particles formalism is convenient for performing this kind of simulation. In order to model lava, we extend the equations, and we propose an efficient way to simulate them. Internal forces inside the flow, and interactions between the flow and a complex terrain data-base are computed in a reasonable time, by the implementation of adequate data-structures. Performing an animation is made easier by the use of a macroscopic model for the lava flow, which only includes a few intuitive parameters (mass, density, stiffness, initial viscosity and temperature of lava, air and ground temperatures).

In the context of searching for visual realism, we also addressed the problem of rendering lava. Our solution, designed for a specific type of basaltic lava flow, consists of rendering a moving texture controlled by the flow and generated on the fly on an implicit surface that surrounds the particles. The texture combines color and displacement information. The displacement map is procedurally computed as the sum of a large scale shape and of a Perlin's noise component. Texture moves and deforms with the underlying flow according to the local variations of the flow speed and temperature. In particular, roughness of the texture increases when the lava cools down, while its color changes. Our methods for defining the displacement map allows us to control the visual aspect of the lava skin, and ensures spatial and temporal continuity during an animation.

Future work includes the modeling of other categories of lavas, such as block lavas for which an extra animation layer could be used for animating large lava-crust blocks carried by the flow, or obsidian flows whose skin behaves as a smooth deformable surface that folds and rolls during motion. The detailed modeling of a lava front, with for example liquid parts spreading out of the rigid crust, is also a challenging work to be done.

Acknowledgments:

We wish to thank Mathieu Desbrun, whose research work on the animation of highly deformable substances served as a basis for this work. Thanks to Pierre-Olivier Noirey for integrating Mathieu's work on our animation platform *FABULE*, and to Eugenia Montiel for re-reading this paper. Many thanks to Eric Ferley for whose practical help was determinant for making the whole thing work.

6 References

[1] J. Blinn. A generalization of algebraic surface drawing. *ACM Transactions on Graphics*, 1(3):235–256, July 1982.

[2] Jules Bloomenthal, editor. *Introduction to Implicit Surfaces*. Morgan Kaufmann, July 1997.

[3] Jean-Louis Bourdier. *Le Volcanisme*. Editions BRGM (manuels et méthodes).

[4] B. Chanclou, A. Luciani, and A. Habibi. Physical models of loose soils dynamically marked by a moving object. In *Computer Animation Conference*, pages 27–35, June 1996.

[5] M. Desbrun and M.P. Gascuel. Animating soft substances with implicit surfaces. In *SIGGRAPH'95 Conference Proceedings*, pages 287–290, August 1995. Los Angeles, CA.

[6] M. Desbrun and M.P. Gascuel. Smoothed particles: A new approach for animating highly deformable bodies. In *7th Eurographics Workshop on Animation and Simulation*, pages 61–76, Poitiers, France, September 1996.

[7] N. Foster and D. Metaxas. Realistic animation of liquids. *Graphical Models and Image Processing*, 58(5):471–483, 1996.

[8] N. Foster and D. Metaxas. Modeling the motion of a hot, turbulent gas. In *SIGGRAPH '97 Conference Proceedings*, pages 181–189, August 1997.

[9] A. Fournier and W.T. Reeves. A simple model of ocean waves. In *SIGGRAPH '86 Conference Proceedings*, pages 75–84, August 1986.

[10] Patrick Fournier, Arash Habibi, and Pierre Poulin. Simulating the flow of liquid droplets. In *Graphics Interface'98*, pages 133–142, May 1998.

[11] J.D. Gascuel, M.P. Cani, M. Desbrun, E. Leroy, and C. Mirgon. Simulating landslides for natural disaster prevention. In *9th Eurographics Workshop on Computer Animation and Simulation (EGCAS'98)*, September 1998.

[12] M. Kass and G. Miller. Rapid, stable fluid dynamics for computer graphics. In *SIGGRAPH '90 Conference Proceedings*, pages 49–57, August 1990.

[13] A. Luciani, A. Habibi, A. Vapillon, and Y. Duroc. A physical model of turbulent fluids. In *6th Eurographics Workshop on Animation and Simulation*, Maastricht, Netherlands, September 1995.

[14] Gavin Miller and Andrew Pearce. Globular dynamics: a connected particle system for animating viscous fluids. *Computers and Graphics*, 13(3):305–309, 89.

[15] J. J. Monaghan. Smoothed Particle Hydrodynamics. *Annu. Rev. Astron. Astrophys.*, 30:543, 1992.

[16] J.F. O'Brien and J.K. Hodging. Dynamic simulation of splashing fluids. In *Computer Animation'95*, pages 198–205, April 1995.

[17] Ken Perlin. An image synthesizer. In *SIGGRAPH '85 Conference Proceedings*, volume 19, pages 287–296, July 1985.

[18] J. Stam and E. Fiume. Depicting fire and other gaseous phenomena using diffusion processes. In *SIGGRAPH 95 Conference Proceedings*, pages 129–136, August 1995.

[19] R. Sumner, J. O'Brien, and J. Hodgins. Animating sand, mud, and snow. In *Graphics Interface*, pages 125–132, June 1998.

[20] D. Terzopoulos, J. Platt, and K. Fleisher. Heating and melting deformable models (from goop to glop). In *Graphics Interface'89*, pages 219–226, London, Ontario, June 1989.

[21] D. Tonnesen. Modeling liquids and solids using thermal particles. In *Graphics Interface'91*, pages 255–262, Calgary, AL, June 1991.