Sculpture virtuelle

Eric Ferley, Marie-Paule Cani et Jean-Dominique Gascuel
iMAGIS[1]
{Eric.Ferley|Marie-Paule.Cani|Jean-Dominique.Gascuel}@imag.fr

Résumé : *Nous présentons ici une métaphore de sculpture destinée au prototypage rapide de formes $3D$. La surface sculptée correspond à l'iso-valeur d'un champ potentiel scalaire échantillonné spatiallement. L'utilisateur peut déposer de la matière où il le désire dans l'espace puis itérativement affiner sa forme grâce à un outil servant à déposer, retirer, peindre ou lisser de la matière, comme déjà proposé dans les approches précédentes. Nous étendons la forme des outils disponibles à des formes libres générées par l'utilisateur. Nous proposons aussi une méthode permettant d'utiliser l'outil pour déformer localement et/ou faire des empreintes sur une forme existante. Nous présentons enfin deux implémentations permettant de stocker ce champ potentiel discret. Leurs performances et limitations sont discutées dans l'article.*

Mots-clés : Surface implicites, déformations locales / empreintes, outils de forme libre, arbres binaires équilibrés, tables de hashage, . . .

1 Introduction

Le contexte général de notre travail est la génération de formes $3D$. Nous choisissons délibérément une *métaphore de sculpture* représentant la surface sculptée comme l'iso-surface d'un champ scalaire discret. En effet, l'élaboration d'une métaphore de sculpture directement à partir des surfaces implicites *classiques* n'est pas une tâche aisée. Les surfaces correspondent à une *tranche* (iso-valeur) d'un champ potentiel scalaire, défini comme le mélange de champs élémentaires générés par des primitives géométriques simples. Le nombre de ces primitives affecte grandement le coût d'évaluation du champ potentiel global. Comme le raffinement par petites touches est au cœur du processus de sculpture, une représentation par surfaces implicites *classiques* conduirait à une explosion de la complexité d'évaluation dûe au nombre croissant de primitives produites.

Dans ce cadre, une représentation discrète du champ potentiel sous forme d'une collection d'échantillons, par exemple sur une grille régulière, apparaît plus adaptée. Dès lors, quelque soit le nombre d'opérations effectuées sur le champ, ce dernier est toujours évalué par une interpolation trilinéaire en temps constant. Nous avons retenu cette représentation déjà employée dans le cadre de sculpture de formes par [GH91, AS96].
Dans cet article, nous nous attachons à stocker le potentiel uniquement là où il influe sur la définition de la surface, et pas sous forme d'une grille dense (un tableau). Nous étendons également la forme des outils utilisés à des formes libres pouvant être générées par l'utilisateur. Nous proposons de nouvelles actions permettant d'utiliser l'outil comme une matrice pour déformer localement l'objet *sculpté* ou faire des empreintes sur sa surface.
Nous nous intéressons aussi à la qualité du rendu, qui procure un plus grand confort d'utilisation et permet de mieux évaluer la forme modélisée, tâche essentielle tout au long du travail de *modelage*.

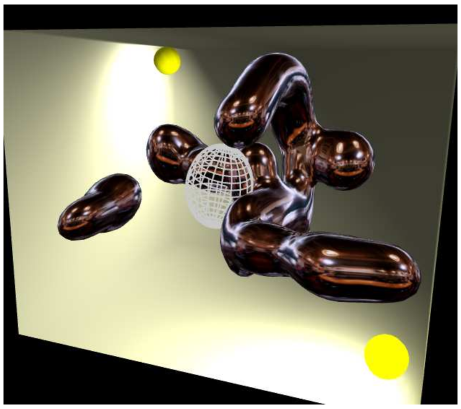


FIG. 1: Capture d'écran de notre application de sculpture. L'objet construit est rendu avec une texture d'environnement. L'outil est affiché en mode fil-de-fer. Les deux sphères jaunes représentent les lumières et la boîte matérialise l'espace de travail.

Nous décrivons tout d'abord notre implémentation (dont on peut voir un exemple de capture d'écran dans la Figure 1) : comment le champ potentiel scalaire est stocké, et mis à jour ; puis comment sont

[1]iMAGIS-GRAVIR/IMAG est un projet commun CNRS,INRIA,UJF,INPG. Adresse postale : INRIA Rhône-Alpes, 655 avenue de l'Europe, 38330 Montbonnot Saint-Martin

réalisés nos outils de formes libres et les déformations locales induites. Enfin, nous détaillons nos méthodes de rendu de la surface générée, donnons quelques performances de notre système sur quelques exemples, et évoquons les travaux futurs.

2 Stockage du champ potentiel discret

Nous considérons la taille induite par les frontières de la grille d'échantillonnage stockée directement sous forme d'un tableau[GH91, WK95, AS96] comme une limitation : non seulement parce que cela enferme le modèle et limite son extension spatiale, mais aussi parce que cela gaspille la mémoire en stockant des échantillons là où le champ potentiel n'est pas défini. Nous avons donc décidé d'utiliser une structure dynamique pour ne stocker que les points d'échantillonnage *intéressants*.
L'utilisation de tables de hashage nous a semblé délicate car nous n'avons aucune connaissance *a-priori* de la distribution spatiale des points à stocker. Dès lors, l'élaboration d'une clef de hashage efficace devient compliquée. L'utilisation d'une clef inadéquate tendrait vers une recherche dans une liste non ordonnée pour retrouver un point d'échantillonnage dans la collection.
Aussi nous nous sommes orientés tout d'abord vers l'utilisation d'arbres binaires de recherche équilibrés qui sont certes moins efficaces qu'une table de hashage dans les bonnes configurations, mais conservent de meilleurs performances dans les configurations moins adaptées pour les opérations courantes d'insertion, recherche, suppression.
Le remplacement par la suite des arbres binaires par des tables de hashage s'est révélé extrêmement simple car nous avons conservé les mêmes *structures* (classes/interfaces) dans les deux implémentations.

Nous décrivons dans la suite notre structure de données, en essayant de donner les motivations et rôles de chaque structure. Puis, dans le paragraphe 2.2 nous expliquons comment nous utilisons ces structures, i.e. comment elles interagissent lors d'une modification du champ potentiel.

2.1 Structure de données : description statique

Nous appelons les points d'échantillonnage disposés régulièrement (formant une grille virtuelle régulière) et échantillonnant le champ potentiel des `Corners`. Chacun stocke une valeur du champ potentiel, une couleur et des données redondantes comme le gradient du champ, les coordonnées du point formant une sorte de *cache* pour éviter de multiples re-calculs. Nous appelons la structure servant à stocker cette collection de sommets un `CornersTree`.
La clef que nous utilisons pour comparer deux `Corners` dans le cas de l'arbre binaire de recherche est simplement constituée des trois entiers, indices (i, j, k) dans la grille régulière virtuelle servant de support. La comparaison de deux clefs (i_1, j_1, k_1) et (i_2, j_2, k_2) se fait dans l'ordre lexicographique, c'est-à-dire qu'on compare d'abord i_1 et i_2. S'il sont égaux, on compare alors j_1 et j_2. Si finalement j_1 et j_2 sont égaux, on compare k_1 et k_2.
En ce qui concerne les tables de hashage, nous avons utilisé directement l'implémentation STL de SGI. Il nous a suffit pour celà de définir une fonction de hashage; nous avons repris celle proposée dans [Blo94] qui consiste à concaténer les 5 bits de poids faible de chacun des indices i, j, k pour construire une clefs sur 15 bits.
Chaque `Corner` ayant une valeur comprise entre les deux seuils arbitraires `minVal` et `maxVal` (respectivement 0 et 3 dans notre cas) est stocké dans le `CornersTree`. Un sommet dont la valeur devient inférieure à `minVal` est supprimé. Les valeurs au-dessus de `maxVal` sont tronquées à `maxVal`. Lorsqu'on demande à un `CornersTree` la valeur d'un sommet non stocké, la valeur renvoyée est `minVal`.

Cet échantillonnage régulier divise l'espace en éléments cubiques : on appelle ces éléments `Cubes`. De la même manière, chaque `Cube` ayant au moins un sommet (`Corner`) défini est stocké dans un `CubesTree`. La clef associée à chaque `Cube` est basée sur le triplet (i, j, k) correspondant au plus petit `Corner` parmis les huits le définissant. La comparaison ou le calcul de la clef de hashage sont les mêmes que pour ce `Corner`. Un `Cube` est constitué de :

- huit pointeurs vers ses `Corners` éventuels. L'un d'entre eux au moins pointe effectivement vers un `Corner` défini.

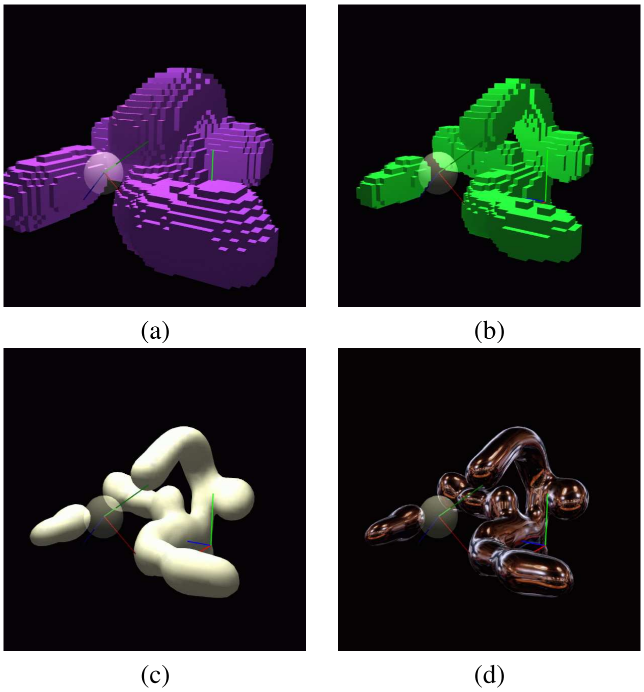


FIG. 2: Visualisation de structures de données : (a) montre tous les cubes ayant au moins un sommet défini. (b) montre les cubes intersectant la surface. (c) montre l'iso-surface extraite. (d) utilise une texture d'environnement pour aider à l'évaluation de la surface.

– un index calculé à partir des valeurs du champs aux huit `Corners` et représentant la configuration du `Cube` par rapport à l'iso-surface. C'est une décomposition classique, étape de l'algorithme de Marching-Cubes[Blo87, LC87, Blo88].
– douze pointeurs vers les arêtes éventuelles.

Les `Cubes` coupant l'iso-surface (dépend de la valeur de l'index calculé) sont également insérés dans un autre arbre/table de hashage que nous appelons `crossList`. La Figure 2 illustre ces structures sur un exemple.

Une arête `Edge` est créée uniquement pour calculer et stocker une intersection avec l'iso-surface. Les `Edges` sont stockées dans un autre arbre binaire équilibré (ou une table de hashage, dans notre seconde implémentation). La clef utilisée pour comparer deux `Edges` est la concaténation des clefs des deux `Corners` la constituant.

2.2 Appliquer un outil : mise à jour du champ potentiel

Lorsqu'un outil est appliqué, nous devons prendre en compte ses modification en mettant à jour le `CornersTree`. Pour ce faire, nous calculons sa boîte englobante dans son repère local `lBBox`, puis la boîte `aaBBox` englobant `lBBox` et alignée avec les axes du repère global. Ensuite, nous parcourons tous les sommets de la grille virtuelle couverts par `aaBBox`. Comme pour connaître la valeur en un point de la grille il faut projeter ce point dans le repère local de l'outil, nous choisissons plutôt de faire un double parcours en parallèle dans les deux repères global et local à la fois, comme suit :

1. on projette dans le repère local les deux points extrêmes P_{min} et P_{max} de `aaBBox` et les trois vecteurs de déplacement (qui passent d'un sommet à l'autre de la grille selon les trois directions).
2. en partant de P_{min}, on se déplace au point suivant simplement en ajoutant le vecteur de déplacement et son correspondant dans le repère local à l'outil : on obtient ainsi à la fois le point dans le repère global et le points correspondant dans le repère local à l'outil simplement par addition de vecteurs (voir Figure 3). On peut également prendre en compte d'éventuels changements d'échelle simplement en modifiant les vecteurs de déplacement.

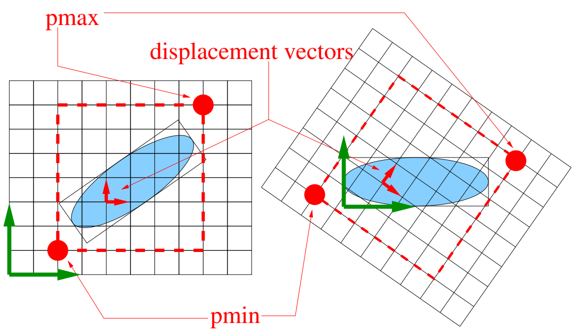
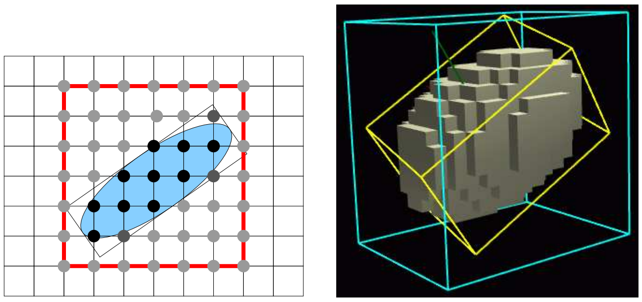


FIG. 3: Parcours en parallèle des boîtes englobantes : à la fois dans le repère global et local à l'outil. Seuls les deux points P_{min} et P_{max}, et les trois vecteurs de déplacement selon les axes du repère global sont ainsi transformés dans le système de coordonnées local à l'outil.

On distingue trois catégories pour chaque `Corner` examiné lors de cette traversée (voir Figure 4.a) :

1. le `Corner` est dans la boîte englobante alignée avec les axes `aaBBox`, mais en dehors de `lBBox`. Il est très rapidement rejeté, le test d'appartenance à une boîte englobante étant immédiat dans le repère local de l'outil. Ces `Corners` **visités** sont représentés en gris clair sur la Figure 4.
2. le `Corner` est dans la boîte englobante locale à l'outil `lBBox`, mais hors de la zone d'influence de l'outil (i.e. la contribution de l'outil est nulle en ce point). Toutefois, pour le savoir, il faut calculer le potentiel de l'outil en ce point. Ces `Corners` **calculés** sont représentés en gris dans la Figure 4.
3. la dernière catégorie concerne les `Corners` dont la valeur est effectivement modifiée. Ces `Corners` **modifiés** (représentés en gris foncé dans la Figure 4) sont insérés dans un arbre/table temporaire appelé `modified` pour être mis-à-jour ultérieurement.



(a) (b)

FIG. 4: Appliquer l'outil. (a) est une vue $2D$ de la grille virtuelle. Les points gris-clair représentent les `Corners` **visités**. Les points gris représentent les `Corners` **calculés**. Les points gris-foncé représentent les `Corners` **modifiés**. (b) montre les boîtes englobantes de l'outil : alignée selon les axes (`aaBBox` en bleu) et orientée (`lBBox` en jaune). Les cubes affichés sont les `Cubes` qui possèdent au moins un sommet où l'outil à une contribution non-nulle.

À chaque affichage, nous extrayons successivement (pop) tous les `Corners` de l'arbre/table `modified`. Pour chacun, nous devons mettre à jour les huit `Cubes` adjacents. Pour éviter les examens multiples de `Cubes`, nous utilisons soit un mécanisme d'étiquette temporelle (timestamp) ou une autre structure temporaire (arbre ou table, selon) pour stocker les `Cubes` à examiner.
L'examen d'un `Cube` se réduit à calculer son index (un masque à huit bits déduit des valeurs de ses huits `Corners` par rapport à iso). Si le `Cube` ne coupe pas l'iso-surface, sa mise-à-jour est terminée. Sinon, son index permet de déduire la configuration de la surface à partir d'une table précalculée (étape classique de l'algorithme de Marching Cubes[Blo87, LC87, Blo88]). Cette configuration indique quelles `Edges` intersectent l'iso-surface. Ces dernières sont à leur tour mises-à-jour.

Lorsqu'une `Edge` est mise-à-jour (ou créée), les gradients du champ potentiel à ses deux `Corners` extrémités sont (re-)calculés (avec un schéma de différences finies centré, dans notre cas). Ensuite, le point d'intersection avec l'iso-surface est obtenu en interpolant linéairement chaque attribut des `Corners` (comme la position, le gradient et la couleur) pondéré par la valeur de champ correspondante. Le gradient interpolé est utilisé comme normale à la surface (et conséquemment envoyé à la carte graphique pour le rendu).

2.3 Gestion des défaire/refaire

Une caractéristique essentielle encourageant la créativité est la facilité des essais multiples : l'utilisateur peut expérimenter ce qu'il désire sans aucune conséquence fâcheuse, car il peut à tout moment revenir à une configuration précédente.
Nous réalisons ce processus de défaire/refaire grâce à des fichier temporaires : à chaque fois qu'un outil est appliqué (modifie le champ potentiel), nous écrivons l'état avant et après modification de tous les `Corners` concerné dans un nouveau fichier temporaire.
Dans notre implémentation actuelle, cela correspond à :

1. écrire ses indices dans la grille virtuelle (i.e. le triplet (i, j, k) par rapport à une origine et un pas d'échantillonnage arbitraires, arbitraires signifiant ici qu'ils sont vides de sens, car sans signification physique, sans dimension).
2. écrire ses attributs avant modification. Les attributs se limitent ici à sa valeur et sa couleur car la position et le gradient étant simplement cachés, ils peuvent être recalculés au besoin.
3. écrire ses nouveaux attributs après modification.

Nous limitons arbitrairement le nombre de ces fichiers temporaires à 200. Lorsque plus de 200 modifications ont lieu, nous cyclons à travers les fichier temporaires déjà existants en les écrasant. Comme les systèmes de fichiers sont assez performants pour effectuer ces opérations en temps interactifs (aussi bien sous IRIX que WindowsNT), la limitation réelle au stockage des défaire/refaire est uniquement l'espace disque.

3 Outils de sculpture

Un outil est défini par :

– un **champ potentiel**, que nous appelons également la *contribution* de l'outil. La boîte englobante de l'outil est en fait la boîte englobante de ce champ. C'est ce champ potentiel qui définit indirectement (implicitement !) la forme de l'outil, qui correspond à l'iso-surface du champ.
– une **action**, qui définit la façon dont le potentiel de l'outil est combiné avec le potentiel objet (éventuellement) existant.

Dans le prochain paragraphe, nous présentons les formes/champ potentiels de nos outils, en insistant sur le premier type d'outils que nous avons utilisés. Dans le paragraphe suivant, nous évoquons les actions possibles des outils. Nous décrivons notre implémentation particulière des outils classiques (i.e. déjà proposés dans les approches précédentes[GH91, WK95, AS96]). Ensuite, nous présentons notre méthode de déformations locales permettant de faire des empreintes.

3.1 Formes des outils

3.1.1 Outil ellipsoïdaux

Au cours de notre implémentation, nous avons d'abord développé des outils ellipsoïdaux, basés simplement sur un potentiel de *Wyvill*[WMW86] classique ou un échelon pour générer un champ potentiel isotropique (sphérique) autours du centre de l'outil (voir Figure 5). On obtient un ellipsoïde général simplement en étirant (scale) le long des axes du repère local à l'outil. Dans ce cas particulier, la forme affichée pour matérialiser l'outil dans l'espace de travail est la limite de sa région d'influence.

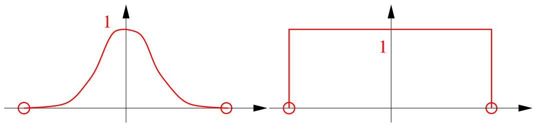


FIG. 5: Fonction potentiel continue utilisée pour un outil ellipsoïdal

3.1.2 Outils de forme libre

Nous avons également utilisé des outils de forme libre générés au sein de l'application. Pour ce faire, nous dupliquons les structures utilisées pour stocker le champ potentiel discret. La surface affichée correspond à l'iso-surface, qui est identique à celle visualisée lors de la construction de l'outil. Ces outils peuvent également être déformés (scaling) selon les axes du repère local à l'outil.

Comme "appliquer l'outil" requiert l'évaluation de son champ potentiel pas forcément aux points d'échantillonnage, nous *reconstruisons* un champ potentiel continu par interpolation tri-linéaire dans le `Cube` concerné : comme le `Cube` stocke les pointeurs vers ses sommets (`Corners`), nous économisons une recherche dans la structure `CornersTree` pour retrouver leur valeur.

3.2 Actions classiques

Les actions classiques sont, comme introduit dans [GH91] :

– dépôt de matière. Ce qui signifie qu'on **ajoute** la contribution de l'outil au potentiel (éventuellement) existant du point d'échantillonnage concerné.
– enlèvement de matière, progressif (on soustrait la contribution de l'outil à la valeur du `Corner` concerné) ou non progressif (on détruit tous les `Corners` où l'outil a une contribution non nulle).
– peinture. Ici aussi, progressivement ou non, selon que l'on prend en compte la couleur éventuelle en un `Corner` déjà défini ou pas.
– lissage, réalisé indirectement par le lissage du champ potentiel, ce qui correspond à un lissage par filtre passe-bas.

Notre implémentation des ces diverses actions est très proche de celles proposées précédemment[GH91, AS96].

3.3 Déformations locales : utilisation de l'outil comme une matrice

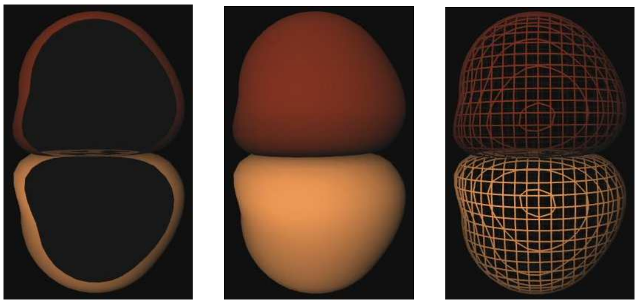
Notre but est de produire des déformations visuellement convaincantes en évitant les problèmes de coût de calcul et de stabilité liés à une simulation physique des déplacements de matière. Notre méthode s'inspire d'une approche développée dans le cadre des surfaces implicites *classiques* (i.e. à champ potentiel continu) [OCG97]. Elle consiste à appliquer un champ potentiel négatif pour comprimer l'objet dans les zones où il y a collision, tout en créant des excroissances imitant le déplacement de matière autours. Ce principe est illustré par les images de la Figure 6.

Nous conservons ici les mêmes pincipes :

1. utilisation d'un potentiel de déformation pour obtenir les *déformations géométriques* sans aucune simulation physique.
2. composition du champ potentiel de l'objet en collision (l'outil dans notre cas) par une fonction de déformation *ad-hoc* pour produire le champ potentiel de déformation.

nous utilisons une fonction de déformation légèrement différente, et notre stockage du champ potentiel échantillonné sur une grille nous permet d'obtenir des déformations plastiques, qui étaient impossible avec l'approche précédente.

Pour ce qui concerne le premier point, notre fonction de déformation provient d'intuitions simples : à l'intérieur de l'outil, nous devons enlever de la matière ; en dehors de l'outil, nous devons ajouter de la



(a) (b) (c)

FIG. 6: Illustration de la méthode de déformations locales introduite dans [OCG97]. Ces images ont été obtenues avec notre implémentation de cette méthode au sein du modeleur par surfaces implicite de l'équipe iMAGIS : *Fabule*[RCGG+97]

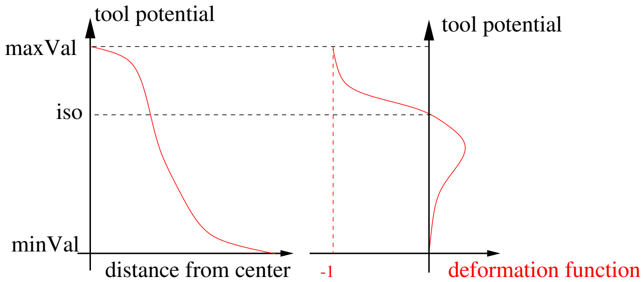
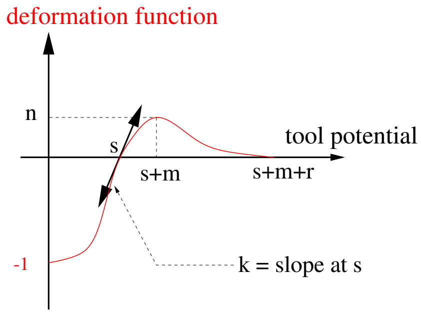


FIG. 7: Principe de la fonction de déformation. Sur la gauche, on voit le potentiel de l'outil généré par un outil sphérique : ce potentiel varie en fonction de la distance au centre de l'outil. Sur la droite, on voit comment la fonction de déformation est mise en correspondance avec le potentiel de l'outil. La composition des deux fonctions relie le terme de déformation à la distance au centre de l'outil.

matière pour imiter les bosses créées par le déplacement de matière lors d'une collision réelle entre un outil et un bloc de terre. Comme cette connaissance de l'intérieur/extérieur de l'outil est codée par son champ potentiel (valeur supérieure à iso signifiant intérieur ; plus le potentiel diminue, plus on s'éloigne de l'iso-surface), nous utilisons le potentiel de l'outil comme paramètre d'entrée pour notre fonction de déformation (voir Figure 7).
Les paramètres de notre fonction de déformation sont détaillés dans la Figure 8 :

$$def(v) = \begin{cases} \text{if } (v <= s) & -\frac{(s.k+2)}{s^3}.v^3 + \frac{(s.k+3)}{s^2}.v^2 - 1 \\ \text{if } (v <= s+m) & -\frac{(k.m+2.n)}{m^3}.v_0^3 + \frac{(2.k.m+3.n)}{m^2}.v_0^2 - k.v_0 \\ & \text{with: } v_0 \longleftarrow v-s \\ \text{if } (v <= s+m+r) & -\frac{2.n}{r^3}.v_0^3 - \frac{3.n}{m^2}.v_0^2 + n \quad \text{with: } v_0 \longleftarrow v-s-m \\ \text{if } (v > s+m+r) & 0 \end{cases}$$

(a)



(b)

FIG. 8: Fonction de déformation : (a) équation et (b) paramètres.

Nous fixons le paramètre s égal à notre iso-valeur, et contrôlons la somme $s + m + r$ pour qu'elle n'excède pas $maxVal - minVal$, sinon le domaine de définition de la fonction de déformation dépasse les variations du champ potentiel : la fonction de déformation serait tronquée.

Pour résumer, nous évaluons la contribution de l'outil déformant en un point P en calculant tout d'abord la valeur du champ potentiel de l'outil en P : $v_{tool}(P)$. Finalement, la contribution de l'outil est la composition de v_{tool} et def :

$$v_{deformTool} = def\,(v_{tool}(P))$$

(a) (b) (c)

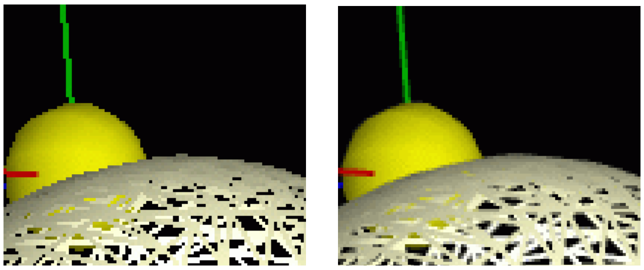
FIG. 9: (a) Déformation obtenue en déplaçant un outil ellipsoïdal le long d'une surface. Utilisation d'un outil de forme libre, construit au sein de l'application, pour réaliser des empreintes sur une forme existante. (b) l'outil est affiché en mode transparent. (c) la même vue sans l'outil.

On peut voir quelques résultats dans les images de la Figure 9.

4 Qualité visuelle

Sur cette plateforme d'essais, nous avons réalisé l'importance de la qualité visuelle aussi bien pour le confort d'utilisation que pour la compréhension de la scène, le placement de l'outil et l'estimation de la surface modelée.

Un des avantages de l'*Infinite Reality* est sa capacité de *filtrer* (antialias) les primitives OpenGL sans surcoût à l'affichage, grâce à une implémentation matérielle (extension SGI *multisample*, voir Figure 10).
Nous avons également essayé deux dispositifs de rendu stéréo : avec des lunettes obturatrices et un



(a) (b)

FIG. 10: Agrandissement d'une capture écran. (a) sans l'extension *multisample* et (b) avec.

visiocasque. Tous deux sont dans un état de développement assez précoce, en particulier nous ne traitons pas le problème classique de convergence/zéro-parallax et nous ne suivons pas la position de la tête. Toutefois, même dans cette configuration sommaire, la stéréo se révèle utile en particulier pour positionner l'outil dans l'espace.
Une autre technique qui améliore grandement la perception de la surface modelée est l'utilisation de textures d'environnement. Nous avons tout d'abord été guidés dans cette voie par la simulation de reflets de haute qualité basée sur l'utilisation de textures *sphère-mappées*. Cette technique est particulièrement utile lorsque la surface possède des triangles dégénérés, ce qui est un inconvénient classique de l'algorithme du Marching Cubes (voir Figure 11). Nous avons utilisé des textures de transparence réglable

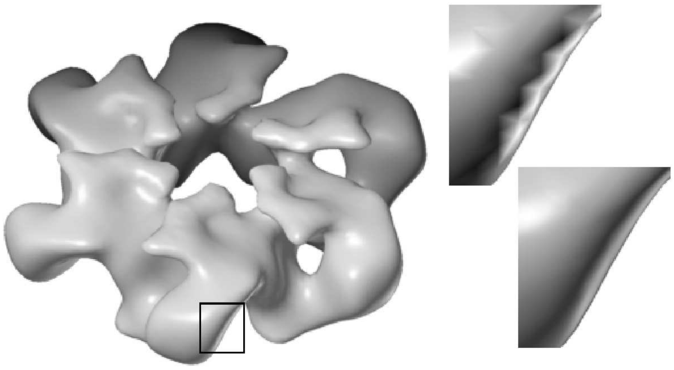


FIG. 11: Exemple d'artefacts dûs au maillage de l'objet, et illustration d'une solution proposée basée sur l'emploi de textures pour obtenir des reflets lumineux de grande qualité (dans le cas de sources lumineuses infiniment distantes, image extraite de [MBG+98], pp.273)

pour pouvoir voir les couleurs de l'objet sous-jacent. Nous avons également implémenté une version simplifiée de *ClearCoat*[2] permettant de simuler une couche de peinture en utilisant une texture d'environnement dont la transparence varie avec l'angle d'incidence entre l'observateur et la surface.

5 Résultats et performances

Nous obtenons des temps de réponse interactifs sans matériel spécifique/dédié de rendu volumique. Avec des performances moindres et une qualité visuelle un peu moins bonne (pas d'anti-aliassage, et moins d'images par seconde), notre programme marche aussi sur un PC standard sous Windows utilisant OpenGL.

Galyean[GH91] utilisait des grilles de 30^3 à 60^3. Avila[AS96] rapporte l'emploi de grilles allant jusqu'à 256^3. Pfister[PK96] utilise du matériel spécifique basé sur son ASIC *Cube-4* pour rendre par ray-casting des grilles de résolution jsuqu'à 256^3 à 30 images par seconde. Ici, l'utilisateur est libre de redimensionner la boîte matérialisant l'espace de travail à tout moment, et d'étendre spatiallement son modèle où il le souhaite, ce qui revient à une grille *apparemment* sans limite. Toutefois, comme le pas d'échantillonnage est fixe, deux types de limitation apparaîssent dans l'implémentation actuelle :

- petit outil : les points d'échantillonnage deviennent trop distants par rapport aux dimensions de l'outil. L'outil n'est pas correctement échantillonné, et des artefacts dûs au bruit apparaîssent.
- gros outil : l'outil couvre tant de points d'échantillonnage que leur mise à jour n'est plus possible interactivement.

À titre d'illustration, nous reportons dans la table suivante des ordres de grandeur de taille d'outil (nombre de `Corners` modifiés) permettant une mise-à-jour dans une gamme de fréquence donnée. L'outil utilisé est un *tube dentifrice* ajoutant de la matière à l'objet représenté à la Figure 2 Cet objet correspond à 15573 valeurs et des arbres `cornersTree`, `cubesTree` et `edgesTree` de profondeurs respectives 14, 16 et 13. L'iso-surface affichée contient 4200 sommets et 8392 triangles. Le programme est éxécuté sur une SGI *Onyx2/IR* ayant 1Go de RAM et deux processeurs R10k à 195MHz.
Pour chaque gamme de fréquence, nous indiquons une estimation, dans nos deux implémentations, du nombre de sommets et cubes couverts par l'outil dans le *meilleur* et *pire* cas, selon son orientation par rapport aux axes.

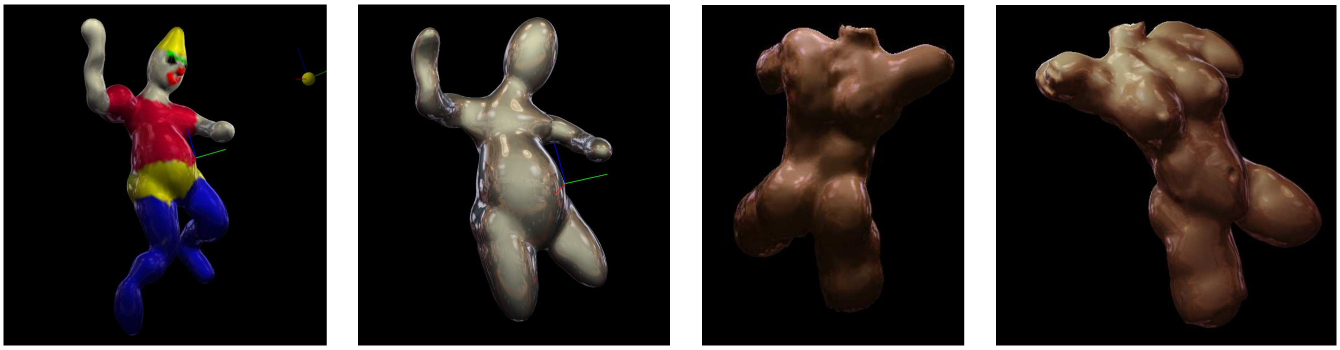
[2] de plus amples informations à propos du produit *ClearCoat* de SGI sont accessibles à la page `http ://www.sgi.com/newsroom/press_releases/1998/september/clearcoat.html`

	implémentation par arbres binaires équilibrés				implémentation par tables de hashage			
images/s	#corners visités	#corners calculés	#corners modifiés	#cubes traités	#corners visités	#corners calculés	#corners modifiés	#cubes traités
19-23	216	125	93	184	1200	891	470	722
	1331	203	**110**	209	4590	890	**463**	707
7-8	1452	887	501	751	3825	3136	1587	2130
	4352	975	**508**	771	11520	3012	**1571**	2104
3-4	2744	2197	1021	1424	6156	4608	2405	3107
	8316	1919	**1003**	1407	18144	4608	**2417**	3125

On voit que le nombre de sommets en dehors de la boîte englobante locale à l'outil influe peu (il sont rapidements rejetés) et qu'ainsi l'orientation de l'outil n'affecte pas les temps de mise-à-jour.
On voit également que l'implémentation par tables de hashage est plus efficace : entre 2.5 et 4 fois plus de `Corners` mis-à-jour dans des temps du même ordre.

(a) (b) (c)

FIG. 12: Exemples d'utilisation d'outil construits au sein de l'application : (a) Construction d'un nouvel outil. (b) Une forme modelée en moins de quinze minutes à l'aide de l'outil précédent. (c) Exemple de sculpture produite avec un outil en forme de *doigt* en moins de dix minutes.



(a) (b) (c) (d)

FIG. 13: (a) Un clown construit uniquement à l'aide d'outils ellipsoïdaux sur une station de travail SGI O2 en environ une heure et demie, ce temps incluant nombre d'essais et erreurs. (b) Le même clown à un moment précoce de conception. (c) et (d) un buste modelé dans les mêmes conditions en environ 4h : le modèle *photographié* ici correspond à 54066 valeurs ; l'iso-surface contient 9566 sommets et 19128 triangles, son affichage simple sur une SGI O2 se fait à une moyenne de 2.7 images par seconde (avec 2 lampes ponctuelles et sans display list, ni textures).

6 Travaux futurs

Il y a encore pas mal d'efforts pour améliorer la qualité visuelle, concernant la robustesse de notre dispositif d'affichage stéréo, ou bien l'ajout d'indices visuels comme les ombres ou la profondeur de champ. Une autre caractéristique clef qui améliorerait l'immersion de l'application dans la réalité de l'utilisateur serait l'exploitation de périphériques à retours d'effort : une première piste dans cette direction a été proposée par Avila[AS96, Avi98].
Une limitation importante dans notre implémentation actuelle est le pas d'échantillonnage fixe du champ potentiel. Nous envisageons dans l'immédiat de stocker le champ potentiel à l'aide d'octrees pour remédier à ce problème, mais une approche multigrilles pourrait se révéler plus efficace.

7 Remerciements

Ce travail est financé par Renault et le CNRS. Nous souhaitons remercier Andras Kemeny pour avoir permis la réalisation de ce projet. Nous remercions également Frédo Durand pour ses intéressantes discussions et intuitions concernant le rendu en une passe des textures dont la transparence varie avec l'angle entre la surface et l'observateur. Nous adressons également nos remerciements aux personnes qui ont contribué au développement de GLUT et Paul Rademacher plus particulièrement pour GLUI[3]. Tous deux sont une aide précieuse à la réalisation d'applications multi-plateforme basées sur OpenGL.

Références

[AS96] R.S. Avila and L.M. Sobierajski. A haptic interaction method for volume visualization. *Computer Graphics*, pages 197–204, October 1996. Proceedings of Visualization'96 (San Francisco).

[Avi98] R.S. Avila. Volume haptics. *Computer Graphics*, pages 103–123, July 1998. SIGGRAPH'98 Course Notes #01.

[Blo87] J. Bloomenthal. Polygonization of implicit surfaces. *Xerox Technical Report CSL-87-2*, pages 1–19, May 1987.

[Blo88] J. Bloomenthal. Polygonization of implicit surfaces. *Computer Aided Geometric Design*, 5 :341–355, 1988.

[Blo94] J. Bloomenthal. An implicit surface polygonizer. *Graphics Gems IV*, 1994.

[GH91] T.A. Galyean and J.F. Hughes. Sculpting : An interactive volumetric modeling technique. *Computer Graphics*, 25(4) :267–274, July 1991. Proceedings of SIGGRAPH'91 (Las Vegas, Nevada, July 1991).

[LC87] W.E. Lorensen and H.E. Cline. Marching cubes : a high resolution 3d surface construction algorithm. *Computer Graphics*, pages 163–169, July 1987. Proceedings of SIGGRAPH'87 (Anaheim).

[MBG$^+$98] T. McReynolds, D. Blythe, B. Grantham, T. McReynolds, and S. Nelson. Advanced graphics programming techniques using opengl. *Computer Graphics*, July 1998. SIGGRAPH'98 Course Notes #17.

[OCG97] A. Opalach and M.-P. Cani-Gascuel. Local deformation for animation of implicit surfaces. *Proceedings of SCCG'97 (Bratislava, Slovakia)*, June 1997. can be found at `http://www-imagis.imag.fr/`.

[PK96] H.P. Pfister and A. Kaufman. Cube-4 - a scalable architecture for real-time volume rendering. *Computer Graphics*, pages 47–55, October 1996. Proceedings of Visualization'96 (San Francisco).

[RCGG$^+$97] D. Rossin, M.-P. Cani-Gascuel, J.-D. Gascuel, A. Opalach, and M.Desbrun. Plateforme d'éxpérimentation pour la modélisation par surfaces implicites. September 1997. Proceedings of Modeleurs Géométriques'97 (Grenoble).

[WK95] S.W. Wang and A.E. Kaufman. Volume sculpting. *Computer Graphics*, pages 151–156, 1995. Proceedings, Symposium on Interactive 3D graphics.

[WMW86] B. Wyvill, C. McPheeters, and G. Wyvill. Animating soft objects. *Visual Computer*, 4(2) :235–242, August 1986.

[3]GLUI User Interface to GLUT est téléchargeable à `http ://www.cs.unc.edu/~rademach/glui`