EPM/RT-85-27

ESPACES TRIDIMENSIONNELS DE TEXTURE

Alain Brossard
Ricardo Camarero

Département de mathématiques appliquées

Daniel Talhman

Département d'informatique et recherche opérationnelle
Université de Montréal

Ecole Polytechnique de Montréal

Septembre 1985

REMERCIEMENTS

Ce travail a été réalisé grâce au support financier du CRSNG et l'appui matériel du LRCAO et des centres de calcul de l'Ecole Polytechnique et de l'Université de Montréal.

L'appui de messieurs Benoit Ozell et Yves Martel est vivement apprécié.

AVERTISSEMENT

Ce rapport contient 36 photographies en couleurs dont la reproduction a été réalisée en noir et blanc. On peut se procurer des copies couleurs auprès de monsieur Ricardo Camarero.

SOMMAIRE

Les images produites par ordinateur ont atteint un niveau de réalisme très élevé, mais elles ont généralement une apparence artificielle parce que les objets sont représentés par des surfaces lisses. Ceci peut être souhaitable pour dessiner un verre, mais pas pour une orange par exemple; c'est pourquoi on applique de la texture aux objets pour simuler la rugosité de leur surface.

Les techniques existantes d'ajout de texture donnent de très bons résultats dans leur domaine d'application respectif, mais elles demeurent limitées à des objets composés de surfaces paramétriques simples. Le but de ce mémoire est de présenter un algorithme de texture qui s'applique à n'importe quel objet, quelle que soit sa représentation. En particulier la méthode permet d'appliquer de la texture à des objets composés de polygones ce qui était impossible précédemment.

La méthode proposée consiste à définir un espace de texture à 3 dimensions. La figure à afficher est placée à l'intérieur de cet espace, ainsi une valeur de texture se trouve associée à chaque point de la figure. Ceci libère la texture de toutes les restrictions liées à la

forme et à la composition de la figure. De plus, ceci garantit la continuité de la texture d'une surface à l'autre de la figure. Les problèmes complexes d'intersections entre deux surfaces paramétriques pour garantir la continuité de la texture sont appelés à disparaître.

La valeur de texture peut servir à faire varier la hauteur, la couleur et la transparence de la surface au niveau du pixel, et ainsi simuler les irrégularités d'une surface réelle.

La variation de hauteur permet, en perturbant la normale à la surface, de simuler la rugosité de celle-ci; la variation de couleur permet de simuler des terrains naturels où la couleur varie beaucoup, ou d'ajouter des motifs de couleur à l'objet; tandis que la variation de transparence permet de simuler des objets de densités variables, comme un arbre ou un nuage.

En utilisant une combinaison de ces trois méthodes, les possibilités d'ajout de texture deviennent illimitées, et il est alors possible de simuler, de manière réaliste, une très grande classe d'objets.

ABSTRACT

Computer generated shaded images have reached a high level of realism with the current state of the art but the surfaces still generally tend to look artificial due to their extreme smoothness. Although this may be adequate to draw a glass, it is not so for an orange; that is why we need to apply some texture to simulate the irregularities of a real surface.

Existing techniques that apply a texture to an object are restricted to parametrically defined surfaces. This thesis presents a texture algorithm that can be applied to any 3-D surface and in particular to the figures of the graphic language MIRA-Shading which are built of polygons.

The technique consists in defining the texture as a 3-D texture space within which the object is placed. Thus every point on the surface of the object has a texture value assigned to it corresponding to its position in the 3-D texture space. It is important to mention that this technique frees the texture from any limitation arising from the shape of the object. Another advantage is the fact that the texture will always be continuous across the whole object.

TABLE DES MATIERES

page

SOMMAIRE iv

ABSTRACT vi

REMERCIEMENTS vii

LISTE DES FIGURES x

LISTE DES SYMBOLES xi

INTRODUCTION 1

CHAPITRE I - DESCRIPTION DE L'ESPACE DE TEXTURE 4

Introduction 4

I-1 Définition de l'espace de texture 5

I-2 Implication de choix de la méthode 7

I-3 Particularités de l'espace de texture 8

I-4 Types de textures disponibles 11

CHAPITRE II - PERTURBATION DE LA NORMALE 12

Introduction 12

II-1 Calcul de la normale 14

II-1.1 Calcul du système d'axes 15

II-1.2 Calcul des dérivées partielles 16

II-1.3 Calcul du déplacement entre les pixels 17

II-2 Résumé 20

II-3 Résultats 21

CHAPITRE III - TEXTURE DE COULEUR ET DE TRANSPARENCE 23

CHAPITRE IV - BROUILLARD 26

CHAPITRE V - FONCTIONS DE TEXTURE UTILISEES 28

V-1 Perturbation de la normale 28

V-2 Modification de la couleur et de la transparence . . . 31

V-3 Ajout de brouillard 33

CONCLUSION 34

BIBLIOGRAPHIE 36

APPENDICE "A" - MANUEL DE L'USAGER 37

A-1 Initialisation des espaces de texture 37

A-2 Espace de perturbation de la normale 38

A-2.1 Initialisation 38

A-2.2 Fonction de perturbation de la normale 38

A-2.3 Perturbation de la normale par la méthode des tableaux 40

A-3 Espace de couleur/transparence 45

A-3.1 Initialisation 45

A-3.2 Définition de la fonction 46

A-4 Espace de brouillard 51

APPENDICE "B" - EXEMPLE DE PROGRAMMES 52

APPENDICE "C" - PHOTOS 62

LISTE DES FIGURES

		page
2.1	Exemple de perturbation de la normale	14
2.2	Systèmes d'axes U et V	15
2.3	Déplacement de la normale	15
2.4	Effet d'un zoom sur l'aliasing	17
2.5	Correction de l'aliasing	18
2.6	Calcul du nombre de pixels qu'occupera la figure	19
2.7	Axes U et V	20
2.8	Calcul de DeltaU et DeltaV	20
2.9	Calcul de la nouvelle normale	21

LISTE DE SYMBOLES

D	: Vecteur déplaçant N vers N'
Dp	: Vecteur tangent à la surface et servant à générer U et V
F	: Distance (scalaire) entre P et P' le long de N
F(X,Y,Z)	: Fonction de 3 paramètres
F_u,F_v	: Gradient de la surface, sert au calcul de D
Has	: Vecteur contenant 32 nombres aléatoires
N	: Normale à la surface
N'	: Nouvelle normale après la perturbation
T	: Symbole indiquant une valeur de texture quelconque
U et V	: Système d'axes perpendiculaires à la normale

INTRODUCTION

L'ajout de texture aux images générées par ordinateur pour en améliorer le réalisme n'est pas une idée récente; dès 1975, Ed Catmull[1] proposait d'utiliser les paramètres d'une surface paramétrique comme entrées d'une fonction de texture qui modifiait l'intensité de la lumière réfléchie. Cette méthode réussit très bien à appliquer un dessin à 2 dimensions, une photo ou un patron quelconque à une surface. Cependant elle simule incorrectement les imperfections d'une surface réelle comme les aspérités d'un mur de béton ou la surface d'une orange car ces surfaces ont un micro-relief dont on doit tenir compte lors du calcul de la luminosité.

Pour résoudre ce problème, en 1978 Blinn[2] proposait d'utiliser les paramètres d'une surface paramétrée comme indices d'un tableau définissant une perturbation de la normale à la surface. Cette perturbation de la normale donne de bons résultats pour simuler le relief d'une surface paramétrique et s'avère particulièrement utile lorsque le tableau de texture est rempli à l'aide d'un programme de peinture («paint system»).

Ces techniques d'ajout de texture donnent de très bons résultats dans leur domaine d'application, mais elles demeuraient limitées à des objets composés de surfaces paramétriques. Ce travail se propose de

lever cette restriction de façon à pouvoir appliquer ces méthodes à n'importe quel objet.

Pour résoudre ce problème, nous proposons de définir la texture non pas comme une fonction de 2 paramètres (Catmull[1], Blinn[2]), mais plutôt comme un espace à 3 dimensions. Ces trois dimensions correspondant aux trois dimensions de l'espace dans lequel la figure à afficher est définie. Cette figure est située à l'intérieur de l'espace de texture, donc chacun des points de l'objet correspond à une position dans l'espace de texture et par conséquent à une valeur de texture. Cette valeur de texture peut alors indiquer soit une perturbation de la normale, soit une perturbation de la couleur de l'objet et/ou une modification de la transparence de l'objet.

Cette proposition d'espace de texture à 3 dimensions permet d'utiliser les textures suggérées par Catmull et Blinn sans pour autant se limiter à des figures paramétrées. En effet, puisque l'espace de texture a trois dimensions, la forme de l'objet n'a pas d'importance car chacun des points de sa surface a automatiquement une valeur de texture qui lui est assignée selon sa position. Ceci est particulièrement important pour des objets ayant des surfaces complexes. La surface de ces objets ne pouvant être décrite facilement au moyen de deux paramètres, alors les techniques classiques à deux dimensions ne peuvent être appliquées. En général ceci est le cas de tous les objets qui n'ont pas été construits de surfaces de Bézier, B-Spline et autres. Les objets composés de surfaces paramétriques à intersections complexes

bénificient aussi énormément de cette nouvelle méthode car le calcul de la continuité de la texture à l'intersection des surfaces n'est plus à faire. Cette continuité est garantie sur tout l'espace et non seulement à l'intérieur d'une seule surface paramétrique.

L'impact du module de texture sur le temps de calcul et l'espace mémoire a été soigneusement contrôlé lors de l'implantation afin d'en limiter les effets lorsque le module n'est pas utilisé et aussi pour éviter un temps de calcul prohibitif s'il est utilisé, de façon à en favoriser l'utilisation.

L'organisation de ce mémoire comprend trois parties principales. Le premier chapitre décrit de l'espace de texture et les méthodes disponibles pour son implantation. Les trois chapitres suivant expliquent les différents types de texture disponibles tandis que le Chapitre V présente les différentes fonctions de texture utilisées à l'aide des photos de l'Annexe C.

Le manuel de l'usager se trouve à l'Annexe A, tandis que l'Annexe B contient un exemple d'utilisation du module de texture.

Après la rédaction de ce mémoire, deux articles ont été publiés qui reprennent certaines idées présentées ici. Un de ces articles a été écrit par Ken Perlin[7] et l'autre par un étudiant de l'Université de Saskatchewan, D.R. Peachey[8].

Chapitre I

DESCRIPTION DE L'ESPACE DE TEXTURE

Introduction

La méthode de simulation de texture proposée repose sur un principe de base: utiliser un espace de texture à trois dimensions à l'intérieur duquel on place la figure à afficher. Chaque point de la surface de la figure correspond à un point de l'espace de texture donnant les valeurs de texture à appliquer sur ce point. Cette méthode permet d'appliquer facilement de la texture à n'importe quel objet, quelle que soit sa forme, mais surtout, et c'est là l'avantage, quelle que soit sa représentation. De plus, cette méthode garantit la continuité de la texture à traves tout l'objet puisque la définition de la texture ne dépend pas de la forme de l'objet.

Il est bien entendu que si on déplace l'objet dans l'espace, son espace de texture doit se déplacer avec lui, de manière à ce qu'on puisse le redessiner de façon consistante. Pour réaliser ceci, au moment de l'affichage on applique aux coordonnées de l'objet les transformations inverses fournissants les coordonnées initiales. Ces coordonnées initiales ont été fixées au moment où l'usager a défini la texture.

1- Définition de l'espace de texture

Deux techniques existent pour trouver la valeur de la texture assignée à une position de l'espace, c'est-à-dire deux méthodes différentes de définir l'espace de texture.

Dans la première technique on défini une fonction $F:R^3 \dashrightarrow R$ qui à partir de la position (x,y,z) dans l'espace retourne la valeur T=F(x,y,z) de la texture. Cette fonction est évaluée au besoin au moment de l'affichage. Par exemple si un pixel de l'écran doit représenter une partie d'une figure qui doit utiliser de la texture, alors le module de texture transmet la position dans l'espace de texture de cette partie de la figure à la fonction de texture qui retourne la valeur de la texture.

La deuxième technique telle que proposée par Blinn[2] (pour une texture en 2 dimensions) consiste à précalculer un tableau de texture de 64 par 64 éléments avant l'affichage de la figure. Au moment de l'affichage, le module de texture n'a qu'une simple interpolation bilinéaire à effectuer pour trouver la valeur de la texture. Ce calcul se fait ainsi :

1- On extraie du tableau de texture déjà calculé les valeurs de texture des quatre points qui encadrent le point dont on recherche la texture
2- On interpole à partir de ces valeurs la valeur recherchée

Le principal avantage de cette dernière méthode est la flexibilité avec laquelle on peut remplir le tableau. Celui-ci peut être rempli par une fonction, à la main ou par l'intermédiaire d'un "paint program" où l'usager "peinture" le tableau interactivement par l'intermédiaire d'un écran graphique. Cet avantage combiné au fait que cette méthode nécessite peu de calcul a justifié le choix de Blinn. Par contre il faut prévoir un tableau par objet à texturer même si par la suite cet objet se révèle en dehors de la fenêtre d'affichage ou caché par un autre objet et donc invisible à l'écran.

La première méthode n'a pas cet inconvénient puisque les calculs s'effectuent seulement pour la partie des objets visibles à l'écran, mais par contre le calcul de la fonction de texture s'effectue pour tous les pixels où cela est nécessaire. Or sur un écran de 512 x 512 pixels, il y a une possibilité de 262 144 pixels donc 262 144 calculs de la fonction : on se retrouve loin des 4096 (64 x 64) calculs nécessaires au remplissage du tableau. En général cependant, le nombre de pixels affectés par la texture est beaucoup moindre.

Mais en pratique, l'espace de texture proposé a 3 dimensions et donc un tableau qui doit le représenter doit lui aussi en avoir 3 soit 64 x 64 x 64 : 262 144 positions. Le nombre de calculs à effectuer pour le remplir augmente d'un facteur de 64 ce qui rend cette méthode plus lente que la première. De plus, un tel tableau occupe à lui seul 1 megabyte de mémoire, ce qui représente un coût prohibitif.

L'espace de texture proposé sera donc basé sur la première méthode: l'usager fournit une fonction de texture (F(x,y,z) --> T) qui est appelée à l'affichage par le module de texture.

2- Implication du choix de la méthode

L'avantage principal de procéder par fonctions fournies par l'usager est de libérer l'espace de travail de 1 megabyte par figure contenant de la texture. Un autre avantage qui sera expliqué en détail plus loin est que la fonction de variations de la normale peut tenir compte de l'échelle à laquelle est affichée la texture. Par exemple si la figure est comprimée selon une dimension, la variation de normale sera calculée en conséquence de façon à limiter les effets d'aliasing.

Le principal inconvénient survient lorsque la fonction fournie repose sur des valeurs prises au hasard. Un problème d'inconsistance apparaît puisque la fonction de texture ne fournit plus la même valeur deux fois de suite pour le même point. Alors on utilise une fonction de Hashing qui retourne une valeur aléatoire prise dans un vecteur de 32 positions selon la valeur de (x,y,z). Donc pour une position la valeur retournée sera toujours la même. Le vecteur de nombres aléatoires n'a qu'une dimension pour limiter l'espace mémoire, on se souvient qu'un tableau de 64 x 64 x 64 occupe 1 megabyte de mémoire. Ce vecteur (Has) contient 32 valeurs aléatoires qui sont définies lors de la création de la fonction de Hashing.

Cependant il faut s'assurer que si les valeurs fournies sont prises au hasard, il doit y avoir aussi une certaine continuité pour éviter l'aliasing. On préserve donc la continuité en passant d'une coordonnée dans l'espace à 3 dimensions à une seule dimension.

Pour cela, le vecteur de nombres aléatoires est utilisé une première fois pour la coordonnée X; on le réutilise pour la coordonnée Y, mais en sautant une valeur sur deux : pour Y=0, on utilise Has(0), Y=1 -> Has(2), Y=16 -> Has(1), Y=17 -> Has(3), etc.; pour la coordonnée Z, on le réutilise en prenant une valeur sur trois. Ces trois valeurs sont ensuite combinées en utilisant des interpolations linéaires à partir des dx, dy et dz.

dx est la partie fractionnaire de X. Par exemple si X=1,49 alors dx=0,49 et la position du nombre aléatoire dans le vecteur sera X0=1. De même pour Y, par exemple si Y=1,49 alors dy=0,49 et Y0 = 1 -> Has(2). Pour Z: si Z=1,49 alors dz=0,49 et Z0=1 -> Has(3).

dx est utilisé pour les interpolations linéaires entre Has(X0) et Has(X1), entre Has(Y0) et Has(Y1) et entre Has(Z0) et Has(Z1). Ces trois interpolations fournissent trois nouvelles valeurs (Ax, Ay, Az) qui sont continues pour des variations de X. Les valeurs Has(X1), Has(Y1) et Has(Z1) correspondent aux valeurs suivantes de Has(X0), Has(y0) et Has(Z0) respectivement dans le vecteur de nombres aléatoires. Par exemple si pour Y, Has(Y0) correspond à Has(3) alors Has(Y1)

correspondra à Has(5).

dy est utilisé pour une interpolation linéaire entre Ax et Ay, ce qui donne Ay0 qui varie d'une façon continue pour un changement de X et de Y. dz est utilisé pour la dernière interpolation linéaire entre Az et Ay0 qui fournit la valeur aléatoire recherchée. Cette valeur est continue pour X, Y et Z.

3- Particularités de l'espace de texture

Pour simplifier l'utilisation de l'espace de texture, ses coordonnées sont normalisées entre (0,0,0) et (1,1,1). Le lien entre ces coordonnées normalisées et les coordonnées de la figure n'est pas fixé d'avance, donc l'espace de texture ne contient pas nécessairement toute la figure. Pour fixer l'espace de texture, on spécifie ses dimensions en coordonnées de la figure et si ces dimensions sont plus petites que celles de la figure alors la figure verra sa texture répétée autant de fois que sa grandeur le demande.

Par exemple, si les dimensions de l'objet sont comprises entre (0,0,0) et (100,50,400), l'usager peut définir l'espace de texture entre (0,0,0) et (50,25,100). Alors la coordonnée (0,0,0) de l'espace d'origine correspondra à la coordonnée (0,0,0) de l'espace de texture. La coordonnée (50,25,100) correspondra à (1,1,1). Par contre le parallélipède formé par les coins (50,0,0) et (100,25,100) correspondra lui-aussi à l'espace de texture défini entre (0,0,0) et (1,1,1). En

fait, l'espace de texture sera répété 2 fois, 2 fois et 4 fois respectivement en x, y et z.

Il est bien entendu que dans ce cas, l'espace de texture devra être continu à ses extrémités pour qu'on ne puisse voir la démarcation entre les répétitions. La Photo 30 est un bon exemple de répétitions, sa texture a été répété 20 fois en X et en Y.

A l'inverse, il est possible de définir un espace de texture plus grand que la figure et selon la correspondance figure <-> espace de texture, seule une partie de la texture sera utilisée.

Cette possibilité de varier le nombre d'occurences de l'espace de texture à l'intérieur de la figure permet de facilement modifier l'échelle de cette texture. Par exemple, si on augmente beaucoup le nombre de répétitions, alors la texture sera dense et petite. Par contre si on ne répète la texture qu'une seule fois alors le patron de la texture sera affichée sur la figure à une plus grande échelle. La différence est très visible sur les Photos 3 et 4.

Par ailleurs, dû à une particularité de l'implantation, l'espace de texture possède 3 plans de symétrie : x=0, y=0 et z=0. Ceci permet de réaliser facilement des effets qui seraient très difficiles à obtenir autrement. Par contre, si cet effet n'est pas voulu, il suffit de définir la position initiale de façon à ce que l'objet ne croise aucun des plans de symétrie. Pour cela on déplace l'objet dans le quadrant

$x \geq 0$, $y \geq 0$ et $z \geq 0$ avant d'initialiser la texture. Ensuite en initialisant la texture, cette position sert de position initiale à partir de laquelle la texture est calculée. Ceci étant fait, il suffit de ramener l'objet à sa position initiale. La Photo 5 illustre bien un effet possible par symétrie.

4- Types de textures disponibles

L'application d'un espace de texture sur une figure au moment de l'affichage permet une énorme flexibilité aussi bien dans le choix du type de texture que dans le choix de la fonction de texture. Quatre possibilités d'espace de texture ont été retenues:

1- Perturbation de la normale (Blinn[2])

2- a) Modification de la couleur (Catmull[1])

   b) Modification de la transparence (Gardner[3])

3- Ajout de brouillard

Ces techniques seront expliquées dans les chapitres qui suivent, mais il est important de réaliser que ces méthodes sont complémentaires et gagnent souvent à être utilisées en combinaison. Par exemple, si on se réfère à la Photo 35, les murs ont été modifiés par une combinaison des méthodes 1 et 2a, la table par les méthodes 2a et 3, le vase par perturbation de la normale, tandis que le plancher n'a subi que l'ajout de brouillard.

Chapitre II

PERTURBATION DE LA NORMALE

Introduction

Le calcul de l'intensité lumineuse dûe aux différentes sources de lumière repose sur l'équation suivante: (Phong[4])

$$I = I_a + I_d + I_s$$

Où I_a est due à la lumière ambiante (constante)

I_d est la lumière réfléchie de façon diffuse

I_s est la lumière réfléchie spéculaire

On a

$I_d = K_d \Sigma_j \vec{N} \bullet \vec{L}_j$ avec $\vec{N}$: normale à la surface

$\vec{L}_j$: direction de la $j^{ème}$ source lumineuse

$I_s = K_s \Sigma_j (\vec{N} \bullet \vec{M}_j)^n$ n: exposant représentant la capacité à réfléchir la lumière

$\vec{M}$: vecteur pointant selon la moyenne vectorielle des vecteurs $\vec{N}$ et $\vec{L}$

Comme on peut voir, l'intensité lumineuse dépend directement de la normale à la surface. Donc pour simuler une surface rugueuse, il n'est

pas nécessaire de construire une telle surface au moyen de polygones, il suffit de perturber la normale comme les véritables variations de hauteur de cette surface le feraient. C'est ce que fait la méthode de perturbation de la normale.

Même si on perturbe la normale, ce sont les variations de hauteur de la surface que l'on cherche à simuler. Il est beaucoup plus simple pour l'usager de spécifier un déplacement de la surface (un nombre variant de -1 à 1 généralement) que de spécifier directement la variation de la normale. Cette variation est représentée par un vecteur perpendiculaire à la normale. Il est très difficile à l'usager de spécifier ce vecteur lorsque le plan qui le contient n'est pas connu à priori. De plus les lois régissant la perturbation de la normale pour simuler un trou par exemple ne sont pas simples. Donc ces calculs sont effectués par le module de texture et l'espace de texture défini par l'usager fournit tout simplement la variation de hauteur de la surface. Le détail de ces calculs se situent dans les pages suivantes.

Par exemple, voici ce qu'on pourrait obtenir sur une ligne:

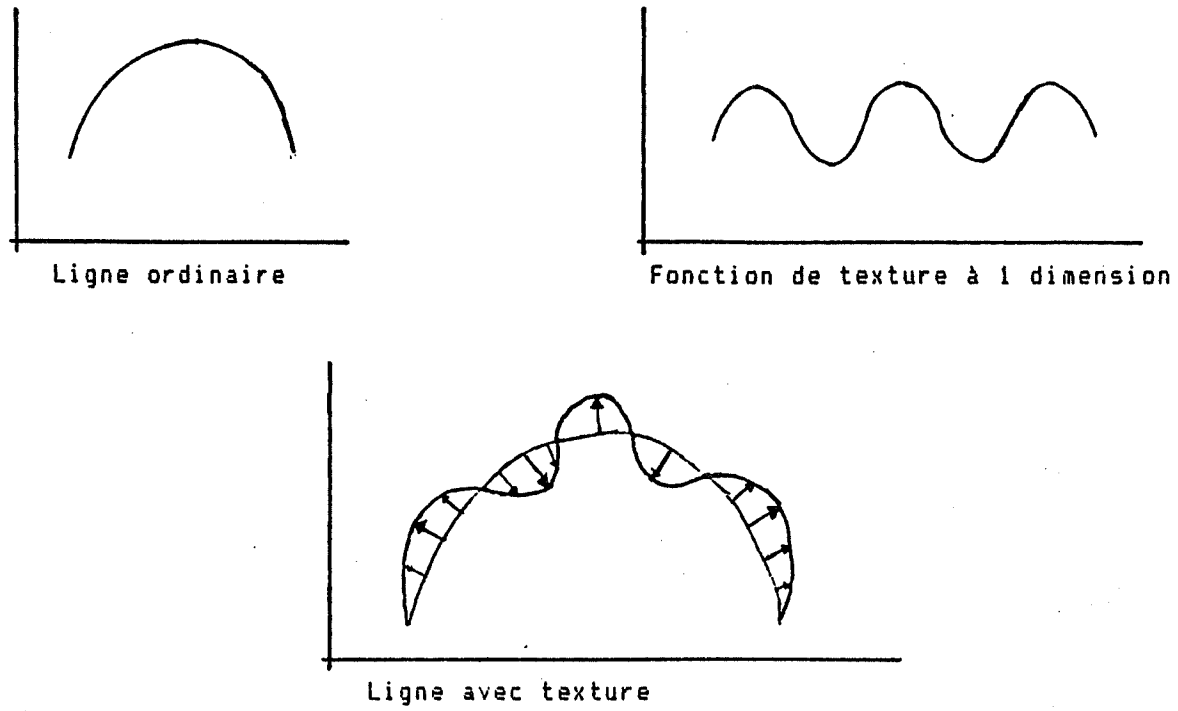


Figure 2-1 Exemple de perturbations de la normale

1. Calcul de la normale

Les données suivantes sont disponibles au début du calcul de la normale : la coordonnée du point dans l'espace d'origine, la normale et la fonction de texture. Pour trouver le déplacement de la normale, on calcule la variation de hauteur le long d'un système d'axes perpendiculaires à la normale. Cette variation de hauteur est fournit par la fonction de texture. Le gradient de la variation de hauteur fournit alors directement le déplacement de la normale.

1.1 Calcul du système d'axes

Soit U, V les axes perpendiculaires à la normale (unitaire) :

$$\vec{U} = \frac{\vec{Dp} \times \vec{N}}{|\vec{Dp} \times \vec{N}|}$$

$$\vec{V} = \vec{N} \times \vec{U}$$

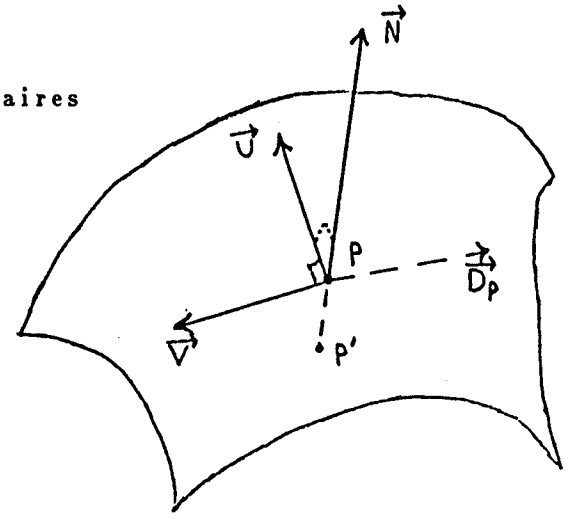
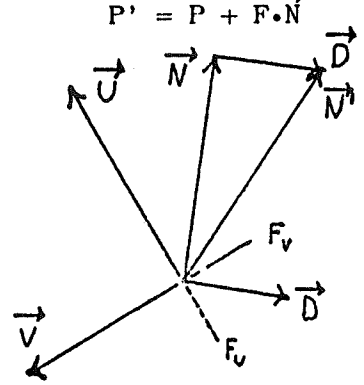


Fig. 2.2 Système d'axes U et V

Dp est un vecteur tangent à la surface du polygone. Il est généré par l'algorithme d'affichage et sert à calculer le système d'axe U et V car il a l'avantage de ne jamais être parallèle à la normale. P est le point à la surface qui doit être déplacé le long de la normale d'une distance F (fournie par l'espace de texture) pour donner P'. Ensuite on redresse le système d'axe U et V (U' et V') pour le faire correspondre à la position et l'orientation de la figure originale.

$$P' = P + F \bullet \vec{N}$$



$$N' = N + D$$

$$D = -F_u \bullet U - F_v \bullet V$$

Fig. 2.3 Déplacement de la normale

La nouvelle normale est calculée par l'addition d'un vecteur déplacement (D) qui correspond au gradient de la hauteur fournie par la fonction de texture.

1.2 Calcul des dérivées partielles

Ce calcul est essentiellement une généralisation dans l'espace à 3 dimensions de l'équation suivante :

$$\frac{dF}{dx} = \frac{F(x+1) - F(x-1)}{2}$$

En pratique, la valeur du déplacement le long de l'axe dépend de chaque figure et doit donc être calculée à priori; en particulier il faut tenir compte des transformations qu'a subies la figure et de la tranformation de perspective. Ce calcul du déplacement est détaillé plus loin.

La valeur du déplacement (Dx, Dy), c'est-à-dire la distance entre deux pixels en coordonnées de l'objet est calculée une fois par figure et est donc connue à ce moment-ci du calcul de la normale. On a donc pour les dérivées :

$$dF/du = Fu = (F(P + \Delta U) - F(P - \Delta U)) / (2 \cdot \Delta U)$$
$$dF/dv = Fv = (F(P + \Delta V) - F(P - \Delta V)) / (2 \cdot \Delta V)$$

1.3 Calcul du déplacement entre les pixels

1.3.1 Justification du calcul

Il est très important de connaître le rapport entre le nombre de pixels de l'écran qu'occupera l'objet et sa grandeur car par exemple, si l'objet occupe une très faible partie de l'écran et qu'on calcule les dérivées partielles à une distance fixe du point, on obtient un effet d'aliasing important :

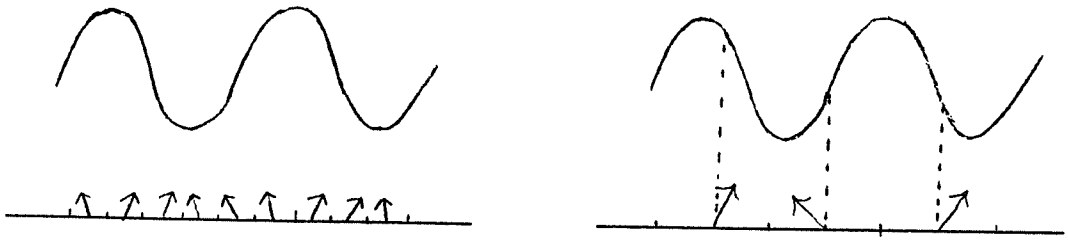


Fig. 2.4 Effet d'un zoom sur l'aliasing

Comme on peut voir, les variations de la normale sont continues sur l'image qui occupe un grand nombre de pixels, mais sont brusques et inconsistantes si le nombre de pixels est petit. Il faut donc tenir compte du déplacement entre deux pixels :

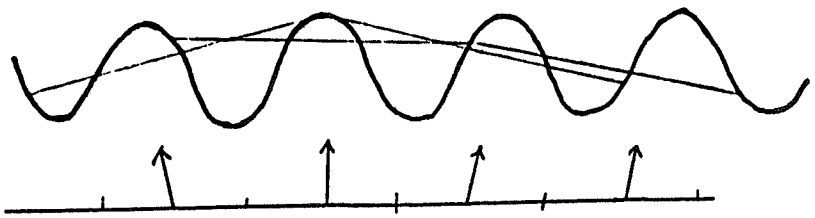


Fig. 2.5 Correction de l'aliasing

Les changements brusques se trouvent filtrés et on pourrait alors apercevoir une variation à plus grande échelle et par opposition, si on fait un zoom sur l'image, on peut apercevoir des variations plus fines. En pratique, on utilise deux fois la distance entre les pixels, de façon à réduire davantage l'aliasing.

On calcule les déplacements à partir de la position qu'occupent les points extrêmes de la figure sur l'écran et on en déduit le vecteur déplacement entre deux pixels en X ($\vec{D}x$) et celui entre deux pixels en Y ($\vec{D}y$).

1.3.2 Etapes du calcul

1- Trouver les extrêmes de la figure d'origine, on retient les six points suivants :

$(Xmin, \overline{Y}, \overline{Z})$, $(Xmax, \overline{Y}, \overline{Z})$ où $(\overline{X}, \overline{Y}, \overline{Z})$ est le point

$(\overline{X}, Ymin, \overline{Z})$, $(\overline{X}, Ymax, \overline{Z})$ central de la figure

$(\overline{X}, \overline{Y}, Zmin)$, $(\overline{X}, \overline{Y}, Zmax)$

2- On fait passer les points par les transformations (rotations, etc.) qu'a subies la figure

3- On applique la projection parallèle ou de perspective

4- Transformation finale sur le plan de vue

5- On calcule le nombre de pixels résultants en X, Y, Z

6- On calcule le déplacement :

Pour chacune des dimensions de départ, on se retrouve avec un certain nombre de pixels en X et Y, il s'agit d'en déduire que pour un déplacement d'un pixel en X de l'écran, on se déplace de $\vec{D}x$ dans l'espace d'origine.

Soit A, B le nombre de pixels en X, Y respectivement impliqués par un déplacement de dx en coordonnées d'origine :

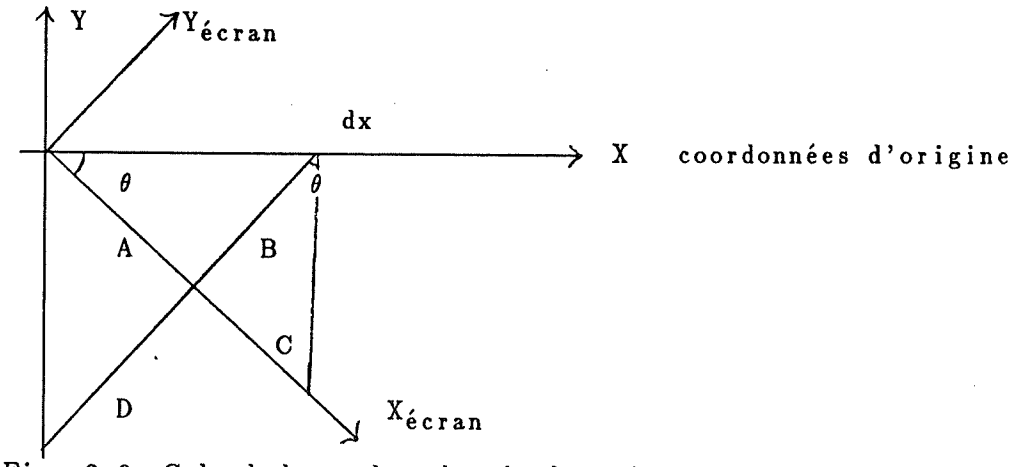


Fig. 2.6 Calcul du nombre de pixels qu'occupera la figure

On cherche les valeurs : $DX.X = dx/(A + C)$
$DY.X = dx/(B + D)$

$\text{Tg } \theta = B/A = C/B \qquad C = B^2/A \qquad DX.X = dx \cdot A/\ (A^2 + B^2)$
$\text{Tg } \theta = B/A = A/D \qquad D = A^2/B \qquad DY.X = dx \cdot B/\ (A^2 + B^2)$

On fait de même en Y et Z et on obtient les vecteurs $\vec{DX}$ et $\vec{DY}$.

2. Résumé

1- On calcule le déplacement entre les pixels : DX et DY

2- A partir de la normale à la surface, on calcule un système d'axe: U et V

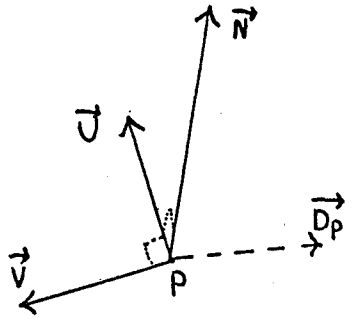


Fig. 2.7 Axes U et V

3- On redresse le système d'axes U et V pour le faire correspondre à celui de la figure d'origine(avant les transformations): U' et V'

4- On calcule des dérivées partielles

A- On calcule le déplacement selon les axes

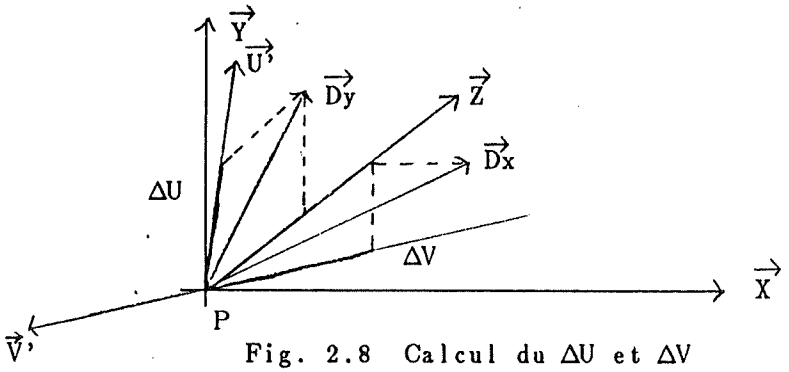
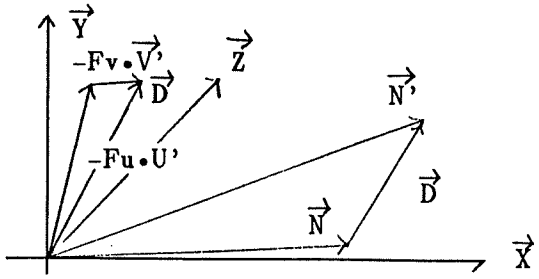


Fig. 2.8 Calcul du ΔU et ΔV

B- On calcule les dérivées

$$Fu = (F(P + \Delta U) - F(P - \Delta U))/ (2 \cdot \Delta U)$$

5- On calcule la nouvelle normale



$$\vec{N'} = \vec{N} - Fu \cdot \vec{U'} - Fv \cdot \vec{V'}$$

Fig. 2.9 Calcul de la nouvelle normale

3. Résultats

L'espace de texture à trois dimensions pour perturber la normale s'est révélé très flexible et facile d'utilisation. Les différentes photos de l'Annexe C permettent de très bien visualiser les résultats obtenus. Par exemple, les Photos 1 et 2 représentent exactement la même texture sauf que l'amplitude de la variation de hauteur est deux fois plus grande en 1 qu'en 2. Autre exemple des possibilités offertes, les Photos 3 et 4 représentent la même texture mais avec un facteur de répétition de 1 et 5 respectivement. L'effet de symétrie est très visible sur la Photo 5 et montre bien les effets que l'on peut obtenir. Certains de ces résultats peuvent même être inattendus : si on examine attentivement le centre de la Photo 8, on peut voir apparaître de petits diables et la tête d'un renard!

Par contre, de tels résultats ne peuvent être obtenus sans pénalités. Le temps de calcul nécessité par une figure utilisant de la texture est généralement le double de celui requis par cette même figure utilisant l'éclairage de Phong[4] sans texture. Le temps d'exécution de

la fonction de texture est évidemment un facteur important, et cela vaut la peine d'essayer d'optimiser cette fonction de texture en précalculant certaines valeurs. Par exemple dans un cas la texture utilisée faisait une utilisation intensive de la fontion EXP(). Celle-ci a été remplacée par un tableau de 400 valeurs et ainsi le temps de calcul a été diminué de 25%.

L' espace mémoire supplémentaire requis par une figure nécessitant de la texture est suffisamment petit pour être négligeable.

Suivant la fonction utilisée, des problèmes d'aliasing peuvent apparaître dû à des variations trop brusques ou trop fréquentes de la fonction. L'usager devra alors modifier sa fonction pour en diminuer les variations ou utiliser le calcul d'antialiasing de MIRA. La méthode utilisée ne filtrant que d'une manière grossière les changements brusques. En parcourant les photos de l'Annexe C, on s'aperçoit cependant qu'il y a très peu de problèmes d'aliasing et que la méthode reste valable dans la majorité des cas. Un autre problème peut apparaître lorsque le relief demandé est trop abrupte. Dans ce cas, la variation de normale devient trop prononcé pour le module d'éclairage qui juge que ce point devrait être invisible. Le résultat de cette limitation est une zone sombre à l'écran. Pour éviter ce problème, il suffit de limiter l'ampleur du relief ou d'ajouter plusieurs sources lumineuses.

Le Chapitre V couvre les différentes fonctions de texture qui ont été testées.

Chapitre III

TEXTURE DE COULEUR ET DE TRANSPARENCE

Contrairement à la perturbation de la normale qui demande beaucoup de calculs de la part du module de texture, la modification de la couleur ou de la transparence repose entièrement sur un choix adéquat de l'espace de texture de la part de l'usager.

Pour fournir un maximum de flexibilité, la procédure de l'usager définissant l'espace de texture n'est pas nécessairement liée exclusivement aux coordonnées d'origine de l'objet, mais peut être liée à la position actuelle de l'objet, à son orientation (accès à la normale) et/ou à la valeur de l'espace de texture de perturbation de la normale.

Par exemple, sur la Photo 9, la variation de couleur est reliée aux coordonnées d'origine; sur la Photo 10 la même variation de couleur est reliée aux coordonnées actuelles; sur la Photo 11, la variation est reliée à l'espace de perturbation de la normale.

Si l'espace de texture est relié aux coordonnées d'origine de la figure, alors sa définition se fera sur le même principe que pour la perturbation de la normale: les coordonnées de l'espace de texture seront normalisées. Par conséquent cet espace pourra être répété et

pourra subir les effets des plans de symétrie. Par contre si l'espace de texture n'est pas défini en fonction des coordonnées d'origine, alors les coordonnées de l'espace de texture ne seront pas normalisées et l'usager devra lui-même gérer une répétition éventuelle de l'espace de texture.

Cet espace de texture permet de très bien complémenter d'autres techniques de graphisme par ordinateur. Par exemple des montagnes construites par fractales gagnent beaucoup en réalisme lorsqu'on leur applique des nuances de couleurs au lieu de toutes les colorer de la même couleur. Par ailleurs, la même idée peut très bien servir à gagner sur le temps de calcul en utilisant une texture pour simuler les petits détails de ladite montagne et ainsi éviter de les calculer en fractale.

Il est important de souligner la flexibilité de cet espace de texture. Cette flexibilité illustrée par les photos 9 à 16 a été à peine touchée par ce mémoire. Par exemple, cela vaut la peine de mentionner que la transparence utilisée dans le contexte d'un espace de texture pourrait servir à dessiner des arbres ou des nuages tel que suggéré par Gardner[3].

Les résultats obtenus illustrent bien les promesses offertes par cette texture. Sur la Photo 12, l'espace de couleur et de tranparence est relié à l'espace de perturbation de la normale de façon à obtenir un dé où les faces sont transparentes mais pas les points qui eux sont colorés en blanc et creusés pour les mettre en évidence. La Photo 13

présente le même dé avec un espace de couleur différent.

Une utilisation judicieuse de la transparence permet de faire des coupes sur un objet et ainsi d'en voir l'intérieur (Photo 14), ou d'obtenir des effets impossibles à réaliser autrement (Photo 21 : "Pelures d'oranges").

Une courte description des fonctions utilisées se trouve au Chapitre V.

Chapitre IV

BROUILLARD

L'ajout du brouillard est essentiellement la possibilité de modifier la couleur de l'objet après le calcul de l'illumination de façon à simuler un brouillard.

L'espace de texture est relié à la position actuelle de l'objet et à la position de l'oeil. Par exemple pour simuler un léger brouillard sur le sol, la fonction de brouillard tient compte de la hauteur d'un point d'une figure. Si un point de l'objet est moins élevé qu'une certaine hauteur, alors il est affecté par le brouillard et selon la distance à l'oeil, il sera plus ou moins blanchi par celui-ci.

Pour simuler un brouillard non limité au sol, il suffit d'utiliser la distance entre l'objet et l'oeil de l'observateur et de présumer que plus l'objet sera loin moins il sera visible et plus il sera blanc.

Cette technique pourrait aussi être utilisée pour projeter une image sur le fond d'une scène, il suffirait de poser un plan à Z=loin et de poser comme espace de texture du brouillard que si $Z \geq$ loin alors on colore selon un patron fourni par l'image à apposer.

Cette modification de la couleur est bien sûr complètement

indépendante des autres espaces de texture et peut être utilisée seule ou en combinaison. Par exemple, la photo 34 montre le brouillard utilisé sur un vase qui est lui-même sous l'effet d'une perturbation de sa normale.

Cet espace de texture n'est pas vraiment de la texture, mais le fait que le brouillard est représenté comme un espace à 3 dimensions comme les autres espaces de texture justifie son inclusion dans ce mémoire. De plus, le mécanisme d'application du brouillard est similaire au mécanisme d'application des autres textures: le pixel est modifié au moment de son affichage. En effet, conceptuellement il n'y a pas de différences entre modifier la couleur d'un objet avant le calcul d'éclairage (texture) ou modifier sa couleur après l'éclairage (brouillard).

Les photos 34 et 35 donnent une bonne idée de l'effet obtenu en utilisant une interpolation linéaire entre la couleur de l'objet et la couleur du brouillard (blanc) selon la hauteur de l'objet.

Chapitre V

FONCTIONS DE TEXTURE UTILISEES

1- PERTURBATION DE LA NORMALE

Cette technique peut entre autre représenter graphiquement n'importe quelle fonction de $R^2 \dashrightarrow R$, par exemple la photo 17 représente la fonction

$$R = \sqrt{X^2 + Y^2}$$

$$H = \frac{SIN\ 8R}{R}$$

Mais bien sûr, ceci n'est pas le principal objectif de la méthode, mais cela illustre bien la versatilité de celle-ci et montre bien que le nombre de fonctions possibles est illimité. En pratique, les essais se sont concentrés sur les deux fonctions suivantes :

$$H = Fact * Sin[2\pi\ (Ax + B/10\ [C * Hasard\ (x,y,z) + 1 + Cos(2\ \pi\ Dy)])]$$

$$H = -\ Fact * \exp(-N^2R)/D$$

$$\text{où } R = \sqrt{(x - x_0)^2 + (y - y_0)^2) + (z - z_0)^2}$$

$$\text{et } N = 4/D$$

La fonction Hasard (x,y,z) retourne une valeur comprise entre 0 et

1 selon la coordonnée du point, ceci permet de retrouver la même valeur d'une image à l'autre. Les différents paramètres (A, B, C, D...) varient selon la figure, les valeurs exactes sont fournies avec certaines photos.

La première fonction permet de facilement obtenir un motif régulier (notez l'effet de la symétrie sur les photos) et l'addition d'un déphasage aléatoire permet de simuler des irrégularités sur un patron régulier (comparez les photos 18 et 19) ou si le déphasage devient très important on obtient un peu d'ordre parmi les textures qui semblent complètement aléatoires (photo de l'orange, 20). Les photos 3 à 7 sont des exemples de l'utilisation de cette première fonction.

La deuxième fonction a servi à créer les points du dé (photos 12 et 13), évidemment leur emplacement n'a pas été décidé aléatoirement, mais dans le cas des photos 22 et 23 au contraire, quarante trous ont été répartis aléatoirement dans l'espace de texture qui a lui-même été répété trois fois, tandis que pour la photo 24 un plus petit nombre de trous a été utilisé.

La première fonction permet aussi de simuler du papier sablé plus ou moins granuleux selon le facteur de variation de hauteur, si on utilise un coefficient de hasard suffisamment élevé ($C > 0,3$), voir les photos 26 et 27. La même fonction peut aussi simuler du plâtre si on augmente l'amplitude de la perturbation (photo 29) ou des montagnes/vallées (photo 28).

Il faut souligner l'importance du paramètre Fact qui contrôle l'amplitude de la variation de hauteur de la surface, le module de texture a été arrangé de façon à ce que si la fonction H varie entre -1 et 1 le résultat soit satisfaisant à l'écran. Le paramètre Fact doit donc d'abord être ajusté pour que H donne un résultat dans cet intervalle et ensuite être modifié pour obtenir le résultat recherché.

Le paramètre qui indique le nombre de répétitions (cf. appendice A, manuel de l'usager) est aussi très important puisqu'il contrôle la "largeur/longueur" des variations par opposition à Fact qui contrôle la "hauteur". En jouant sur ce paramètre, on peut obtenir des résultats très variés comme il a été montré sur les photos 3 et 4.

Bien sûr, même si ce sont surtout les deux fonctions précédentes qui ont beaucoup servi, d'autres ont été essayées. Les fonctions suivantes ont servi à composer les photos indiquées :

$$R = \sqrt{(x - x_0)^2 + (y - y_0)^2 + (z - z_0)^2}$$

$$H = Fact \cdot (D_1 \cdot \exp(-N_1 R) - D_2 \cdot \exp(-N_2 R))$$

Photo 25 : $D_1 = 0,5$ $\qquad 10 \cdot 0,45 \leq D_0 \leq 0,47 \cdot 30$

$N_1 = 4/0,5$ $\qquad 1/0,45 \geq N_2/4 \geq 1/0,47$

$$H = Fact \cdot Cos\ [\sqrt{(x - 0,5)^2 + (y - 0,5)^2}]$$

Photo 30 et 31

2- MODIFICATION DE LA COULEUR ET DE LA TRANSPARENCE

Peu de fonctions ont été développées pour cet espace de texture, car il a en général été défini en énumérant les couleurs à utiliser et les conditions dans lesquelles les utiliser (photos 9 à 11 et 13).

Soit par exemple en reliant la couleur à la coordonnée Y du vase (photo 9) ou à la profondeur du trou (photo 32); cette technique bien que grossière peut cependant fournir des résultats très beaux (photo des cercles, 30) ou très bizarres (photo 33); ces deux dernières photos ont été réalisées en reliant l'espace de couleur à l'espace de perturbation de la normale qui a été défini comme suit :

Photo 30 $H = Fact \cdot Cos\ [\pi\sqrt{2}\sqrt{(x - 0.5)^2 + (y - 0,5)^2}]$

Photo 33 $H = Fact \cdot Cos\ [\pi\sqrt{2}\sqrt{(x - 0,5)^2 + (y - 0,5)^2 + (z - 0.5)^2}]$

Cependant une classe de fonction a été examinée et a servi à créer le dessus de la table des photos 15, 16 et 35, il s'agit de :

$$TT = 1/2 + 0,5 \cdot [\sum_i C_i\ (Sin(w_i x + P_z) + 1) \cdot \sum_i C_i (Sin\ (w_i z + P_x) + 1)]$$

avec $C_{i+1} = C_i/\sqrt{2}$,

$w_{i+1} = 2w_i$

et $P_z = K_1 \text{ Sin } (2\pi z)$ $P_x = K_2 \text{ Sin } (2\pi x)$

Avec K_1 et K_2 suffisamment élevé (≥ 4), P_x et P_z font perdre toute apparence régulière à la fonction TT. On a intérêt à limiter le nombre de sommations pour réduire le temps de calcul et en pratique trois sommations suffisent. La table de la photo 35 a été réalisée avec les paramètres suivants : $K_1 = K_2 = 8$, $w_1 = 8\pi$ et $C_1 = 1$; la couleur a été modifiée comme suit :

Rouge = TT • Rouge_Prévu

Vert = TT • Vert_Prévu

Bleu = TT • Bleu_Prévu

Cette classe de fonctions peut très bien servir à représenter du terrain (Gardner[3]) et même des arbres si on joue aussi avec la transparence (Gardner[3]).

Les essais avec la transparence ont été très limités et les seules photos disponibles sont celle du dé (photo 12), celle des pelures d'orange (photo 21) et enfin celle de la coupe sur le vase (photo 14).

3- FONCTION DE BROUILLARD

La seule fonction utilisée consiste à remplacer la couleur de l'objet par une combinaison linéaire de celle-ci et la couleur du brouillard selon la hauteur de l'objet.

Par exemple la figure 34 a été réalisée comme suit :

```
IF Y < 40 THEN BEGIN
   TT     := Y/40.0  ;
   Rouge  := 1 - TT + TT • Rouge;
   Vert   := 1 - TT + TT • Vert;
   Bleu   := 1 - TT + TT • Bleu;
END;
```

Tandis que le brouillard sur le plancher de la pièce de la photo 35 a été créé par la fonction suivante :

```
IF Y < 10 THEN BEGIN
   T      := Y/20.0 + 0.5;
   Rouge  := 1 - T + T • Rouge;
   Vert   := 1 - T + T • Vert;
   Bleu   := 1 - T + T • Bleu;
END;
```

CONCLUSION

Les résultats obtenus permettent de conclure que l'idée d'utiliser un espace de texture en trois dimensions se révèle très efficace et très flexible. Cet espace de texture permet d'appliquer de la texture à des objets formés de polygones, ce qui était impossible avec les techniques déjà existantes. Maintenant celles-ci (bumpmapping, jeu sur la couleur, etc.) peuvent être utilisées avec avantage, que ce soit indépendemment les unes des autres ou en combinaison.

Cependant, comme déjà mentionné, ceci ne se fait pas sans une certaine pénalité au point de vue du temps de calcul qui correspond approximativement au double de celui nécessité par un éclairage de "Phong". Cette valeur dépend évidemment beaucoup de la fonction de texture choisie et n'est donc qu'approximative.

Cette technique possède tout de même certains inconvénients dont il faut être conscient; premièrement, si la fonction de texture utilisée présente trop de variations brusques, un phénomène d'aliasing va apparaître et il faudra le réduire en utilisant l'antialiasing du langage MIRA-Shading. Deuxièment, si un espace de texture de "bumpmapping" mise sur un élément de hasard, la fonction fournie (Hasard(x,y,z)) peut se révéler inadéquate par ses changements trop abrupts et dans ce cas, les résultats fournis par la méthode des

tableaux se révèlent meilleurs; c'est pourquoi cette possibilité a été conservée et c'est celle-ci qui a permis de créer la photo de l'orange (Photo 20).

Dans le futur, il vaudra la peine d'examiner en plus grande profondeur l'espace de texture de transparence dont les grandes possibilités ont à peine été traitées dans ce travail.

De même, les possibilités qu'offre l'espace de couleur dans la représentation de terrains naturels devraient être explorées, en particulier les processus stochastiques qui présentent un grand nombre de débouchés.

BIBLIOGRAPHIE

1. CATMULL, E.E., "Computer Display of Curved Surfaces", Proc. IEEE Conf. on Computer Graphics, Pattern Recognition and Data Structures, Los Angeles (May 1975).
2. BLINN, J.F., "Simulation of Wrinkled Surfaces", Siggraph 78 Proceedings, Computer Graphics, pp. 286-292 (August 1978).
3. GARDNER, G.Y., "Simulation of Natural Scenes Using Textured Quadric Surfaces", Siggraph 84 Proceedings, Computer Graphics, pp. 11-20 (July 1984).
4. BUI-TUONG, PHONG, "Illumination for Computer-Generated Pictures", Comm. of the ACM, pp. 311-317 (June 1975).
5. BATSON, R.M., EDWARDS, E. and ELIASON, E.M., "Computer Generated Relief Images", Journal, Research U.S. Geology Survey, pp. 401-408 (July-August 1975).
6. BLINN, J.F., "Texture and Reflection in Computer Generated Images", Comm. of the ACM, pp. 542-547 (October 1976).
7. PERLIN, Ken, "An Image Synthesizer", Proc. of Siggraph, pp. 287-296 (July 1985).
8. PEACHERY, D.R., "Solid Texturing of Complex Surfaces", Proc. of Siggraph, pp. 279-286 (July 1985).
9. MAGNENAT-THALMANN, N. et al., "MIRA-SHADING A Structured Language for the Synthesis and the Animation of Realistic Images", Frontiers in Computer Graphics, Springer-Verlag, Tokyo 1985.

APPENDICE "A"

MANUEL DE L'USAGER

1- INITIALISATION DES ESPACES DE TEXTURE

L'usager doit d'abord signaler son intention d'utiliser un espace de texture en activant celui-ci au moyen d'une des procédures "Set" suivantes :

SetTexture	(b : boolean);	pour l'espace de perturbation de la normale
SetModCoul	(b : boolean);	pour l'espace de modification de couleur et de transparence
SetTransparency	(b : boolean);	pour activer la transparence
SetFog	(b : boolean);	pour l'espace de brouillard

La variable "b" est une variable booléenne qui doit être vraie pour activer l'espace de texture et fausse pour le désactiver. La procédure SetModCoul active l'espace de couleur et de transparence, mais la transparence ne sera visible que si SetTransparency a été spécifiée.

2- ESPACE DE PERTURBATION DE LA NORMALE

2.1- Initialisation

Chaque figure nécessitant cette texture doit être dessinée en "shading" de Phong et être initialisée avant son affichage, mais seulement une fois que les paramètres de l'affichage (oeil, plan de vue, projection, etc.) ont été spécifiés. L'initialisation se fait par l'appel à la procédure Init_Text en lui passant en paramètre la figure et la position dans l'espace 3D qu'occupera l'espace de texture (et par conséquent le nombre de fois qu'il sera répété). Cette position est exprimée en coordonnées de l'objet en spécifiant : x, y, z minimum et maximum :

```
Procédure    Init_Text (F : Fig; Xmin, Xmax, Ymin, Ymax,
                                 Zmin, Zmax : Real);
```

Par exemple, si les dimensions de l'objet s'étendent de (-5,0,0) à (5,100,200) et qu'on spécifie les limites de l'espace de texture comme étant (0,0,0) à (5,50,50), alors l'espace de texture sera répété 2 fois en y et 4 fois en z tandis qu'il y aura une symétrie autour du plan x = 0.

2.2- Fonction de perturbation de la normale

L'usager doit fournir la fonction de perturbation de la normale

sous la forme d'une fonction qui retourne la hauteur de la variation de la surface au point (X, Y, Z) demandé :

```
Function   BUMP(x,y,z : Real; VAR  Nom : ALFA) : REAL;
```

La coordonnée (X, Y, Z) correspond à un point de l'espace de texture et est automatiquement comprise entre (0,0,0) et (1,1,1); autrement dit, la coordonnée d'origine a été normalisée selon les rapports fournis par l'usager lors de l'utilisation de la procédure Init_Text. La variable Nom contient le nom de la figure et sert à différencier différents espaces de texture si plus d'une figure se trouvent texturées. Bump devrait retourner une valeur comprise entre -1 et 1, mais selon la fonction utilisée, un résultat plus extrême peut être souhaité; la valeur exacte de l'amplitude dépendra du résultat graphique souhaité.

Par exemple, la fonction pourrait être définie comme suit :

```
Function    Bump ( x,y,z : Real; Var Nom : Alfa) : Real;
  CONST
     PI = 3.1415927
  Begin
     Bump :=  1 • Cos ( 2πx  +  4πy);
End;
```

La fonction Bump doit se trouver dans un fichier à part du

programme principal, autrement dit à l'intérieur d'un module de façon à ce que le pré-compilateur ISMIRA lui donne une visibilité "globale" et bien sûr ce fichier doit être mentionné au niveau du "Link".

Un exemple plus complexe et complet de la définition de fonctions d'espaces de texture est fourni à l'appendice B.

2.3- Perturbation de la normale par la méthode des tableaux

Comme déjà mentionné, la méthode des tableaux fournit parfois un meilleur résultat lorsqu'on utilise des valeurs aléatoires dans la définition de la fonction de texture. Pour tirer parti de ce fait, cette méthode a aussi été implémentée, mais il faut réaliser la très grosse pénalité qu'on paye en l'utilisant : 1 megabyte de mémoire se trouve alloué à la figure à texturer. Autrement dit, n'essayez pas de mettre deux figures avec texture sur le même dessin ou même de mettre trop de figures car il n'y aura pas assez d'espace mémoire disponible.

2.3.1- Initialisation

L'initialisation se fait selon le même principe sauf que lors de l'initialisation, un tableau de texture est rempli à l'intérieur de la figure selon la fonction de texture fournie par l'usager :

```
Procedure   Init_Text2 ( F : Fig; Xmin, Xmax, Ymin, Ymax,
                                  Zmin, Zmax  : Real;
                          Function Bump (x,y,z : Real) : Real);
```

Les dimensions de l'espace de texture suivent les mêmes lois que précédemment.

L'usager doit passer la "fonction" lors de l'initialisation et cette fonction peut donc être locale au programme de l'usager et n'a plus besoin d'être définie dans d'un module séparé.

Les variables x, y, z sont comprises entre (0,0,0) et (1,1,1) et représentent une position dans l'espace de texture. Il n'y a pas de variable "Nom" puisque l'usager fournit automatiquement le bon espace de texture.

La texture sera ensuite appliquée lors de l'affichage sans autre intervention de l'usager, mais entre le DRAW et l'opération IMAGE, la figure ne devra pas avoir été modifiée car pour économiser de l'espace mémoire, le tableau de texture n'est pas copié.

La page suivante contient un exemple de programme qui a servi à réaliser l'orange de la Photo 20.

```
PROGRAM      ORANGE( INPUT, OUTPUT , fichier);

TYPE
   FILINTEGER =      FILE OF INTEGER;
   IDENTFILE  =      PACKED ARRAY [1..40] OF CHAR;

PROCEDURE      AFFType ( Dithering : INTEGER );                    EXTERN;
PROCEDURE      ANTIALIAS ( N : INTEGER );                          EXTERN;
PROCEDURE      INIT_TEXT2(      F : FIG;
                     XMi, XMax, YMi, YMax, ZMi, ZMax :  REAL;
                     FUNCTION  Bump( X, Y, Z : REAL): REAL ); EXTERN;
PROCEDURE      ClearMonitor;                                       EXTERN;
PROCEDURE      BckColor( H, L, S : REAL);                          EXTERN;
PROCEDURE      Diffuse( I:Vector );                                EXTERN;
FUNCTION      INQMAXVIEWPORT      :      VECTOR;                  EXTERN;
PROCEDURE      MONITOR( NBLigne : INTEGER );                       EXTERN;
PROCEDURE      PERSCAM( O, I, U: VECTOR; A : REAL);                EXTERN;
PROCEDURE Point( N:Integer; I:Vector; Dir:Vector );                EXTERN;
PROCEDURE      Extrema( F: FIG; VAR XMIN, XMAX, YMIN, YMAX,
                              ZMIN, ZMAX : VECTOR) ;          EXTERN;
PROCEDURE      CLOSEFILE(VAR FL : FILINTEGER; VAR ERROR : BOOLEAN );
                                                                   EXTERN;
PROCEDURE      READBINFIG(VAR FL : FILINTEGER; VAR FG : FIG);      EXTERN;
PROCEDURE      OPENFILE( VAR F: FILINTEGER; NAM : IDENTFILE;
                     KIND: INTEGER ; VAR ERROR : BOOLEAN) ;      EXTERN;
PROCEDURE      SetTexture( V :  BOOLEAN );                        EXTERN;
PROCEDURE      SetModCoul( V :  BOOLEAN );                        EXTERN;

VAR
   Ran                    :      INTEGER;
   Oeil                   :      Vector;
   Diviseur, Anti         :      INTEGER;
   Fichier                :      FILINTEGER;
   Nom                    :      IdentFile;
   F1                     :      Fig;
   XMin, Xmax,
   YMin, Ymax,
   ZMin, Zmax             :      Vector;
   Erreur                 :      BOOLEAN;


CONST
   PI  =  3.1415926;
   PI2 =  6.2831852;
   Angle    =      28.0;
```

```
FUNCTION     Bump( X, Y, Z : REAL ): REAL;
{
   X et Y sont compris dans l'intervalle [0,1]
   Bump doit retourner une valeur centree sur 0

   Bump sert a creer la surface ondulee de l'orange
}
 Function  random( Var seed:integer ):Real;  Extern;
   BEGIN
      BUMP := 0.5 * SIN( PI2 * ( 2 * X + 4 * ( 0.6 * RANDOM(RAN) + 1 +
                        COS( PI2 * Y )) ) );
   END;  {       Bump        }

BEGIN

   Monitor( 2 );
   CLEARMONITOR;
   WRITELN;

   Diviseur := 4;
   AffType( Diviseur );          { Choisi le type d'affichage }

   CLEARMONITOR;
   WRITELN( '+Antialiasing (1..4) ?');
   READLN( Anti );
   ANTIALIAS( Anti );
{
   On lit la forme de l'orange (sphere ecrasee)
}
   Nom := 'ORANGE                                          ' ;
      OPENFILE( Fichier, Nom, 0, Erreur);

      RESET ( Fichier );
      READBINFIG( Fichier, F1 );
      closefile( fichier, Erreur);
{
   On enregistre les extremes de la figure
}
   Extrema( F1, Xmin, Xmax, Ymin, Ymax, Zmin, Zmax );
   IF PROJX(XMax) - PROJX(Xmin) = 0 THEN XMax := <<PROJX(XMin) + 1,
                                          PROJX(XMax), ProjX(XMax)>>;
   IF PROJY(YMax) - PROJY(Ymin) = 0 THEN YMax := <<PROJY(YMin),
                                        PROJY(YMax) + 1, ProjY(YMax)>>;
   IF PROJZ(ZMax) - PROJZ(Zmin) = 0 THEN ZMax := <<PROJZ(ZMin),
                                          PROJZ(ZMax), ProjZ(ZMax)+1>>;
```

```
{
   Initialiser la couleur du fond et le module de texture
}
   BCKColor( 0.0, 0.75, 1.0 );
   Surface( TRUE );
   SetTexture( TRUE );
   SetModCoul( FALSE );
{
   Initialiser la texture
}
   INIT_TEXT2(  F1, PROJX( XMin )/4, PROJX( XMax )/4,
                    PROJY( YMin )/4, PROJY( YMax )/4,
                    PROJZ( ZMin )/4, PROJZ( ZMax )/4,
                    Bump );
{
   Initialiser l'environnement visuel
}
   Window( << -45, -45>>, << 45, 45>>);
   Viewport( <<0,0>>, INQMAXVIEWPORT );

   Oeil       :=   << 0, 180,-300>>;
   Perscam( Oeil, << 0, 0, 0>>, <<0,1,0>>, Angle);
   Point( 1, << 0.6, 0.6, 0.6>>, << 160, -320, 90>>); { RGB, direction }
   Diffuse( << 0.5, 0.6, 0.6 >> );       { RGB }
{
   Dessiner l'orange
}
   DRAW F1;
   image;

      ClearMonitor;
      WRITELN( ' Fini');
end.
```

3- ESPACE DE COULEUR/TRANSPARENCE

3.1 Initialisation

La figure à texturer doit être définie avec le "shading" de Phong, son initialisation se fait exactement de la même façon que pour l'espace de perturbation de la normale et par la même procédure. De fait, une fois qu'une figure a été initialisée pour un espace de texture, il est redondant de la réinitialiser pour un autre puisque les informations requises ont déjà été enregistrées.

Cela a le désavantage de faire que la grandeur des espaces de texture et de couleur/transparence (relié aux coordonnées d'origine) soit la même, on ne pourrait donc répéter un des espaces 2 fois sans répéter l'autre 2 fois aussi; cependant, le gain d'espace mémoire, la simplification de la logique requise et le gain en temps de calcul font plus que compenser pour ce léger désavantage qui peut d'ailleurs facilement être contourné en redéfinissant un des espaces comme étant défini sur (0,0,0) à (0.5, 0.5, 0.5) et en le répétant sur (0.5, 0.5, 0.5) à (1,1,1).

3.2 Définition de la fonction

```
Procédure   ModifCoul(  OldCoord, Coord, Normale : Vector;
                        Function Hauteur(x,y,z : Real) : Real;
                        Var Nom : ALFA; Var Couleur; RECCOUL;
                        Var T min, T max, M : Real;
                        Var Trans : Boolean);
avec        Type
               RecCoul = RECORD
                         Coul : Record
                                R, G, B : Real;
                            End;
                         W, N : Real;
                      End;
```

L'usager doit fournir cette procédure dans un module de façon à ce qu'elle soit accessible au module de texture.

3.2.1 Variables d'entrée

L'usager recevra les variables suivantes :

OldCoord : Coordonnées d'origine dans l'espace de texture normalisées de (0,0,0) à (1,1,1).

Coord : Coordonnées actuelles de l'objet (au moment du "DRAW") non normalisées.

Normale : Normale à la surface (non unitaire) avant la perturbation à la normale

Nom : Nom de la figure

Hauteur : Fonction de perturbation de la normale qui s'attend à recevoir OldCoord comme input et fournit la hauteur de la

perturbation à la surface

Ces variables servent à définir l'espace de texture.

3.2.2 Variables de sortie

Couleur.Coul.R,
Couleur.Coul.G,
Couleur.Coul.B : Couleur de la figure qui sera modifiée par l'usager

Couleur.W,
Couleur.N : Paramètres du "highlight" qui pourront être modifiés

Tmin,
Tmax, M : Paramètres de la transparence qui peuvent être modifiés dans le cadre de l'espace de texture de transparence

Transp : Active la transparence pour cette figure de ce pixel si cela n'a pas été fait lors de la définition de la figure.

Les programmes qui suivent montrent comment on peut définir cette fonction de texture. Le premier programme montre comment relier l'espace de couleur/transparence à l'espace de perturbation de la normale et a servi à créer la photo 12. Le deuxième programme crée un espace de texture basé sur les coordonnées actuelles de l'objet et a servi à créer la photo 10.

```
MODULE          MCUBE( INPUT, OUTPUT );

TYPE
   alfa     =   PACKED ARRAY [ 1..10 ] OF CHAR;
   SysCouleur  = ( RGB, H);
   Color = RECORD
           CASE SysCouleur OF
             HLS : (  H,L,S: Real );
             RGB : (  R,G,B: Real );
          END;
   RecCoul = RECORD
             Coul : Color;
             W,N  : Real;
            END;

PROCEDURE        ModifCoul( OldCoord, Coord, Normale  :  VECTOR;
                            FUNCTION  Hauteur( X, Y, Z : REAL ):REAL;
                            VAR  NOM : ALFA;   VAR  Couleur : RECCOUL ;
                          VAR  TMin, Tmax, M  : REAL ; VAR Trans : BOOLEAN);
VAR
      B   :   REAL;
BEGIN
   B := Hauteur( PROJX( OldCoord ), PROJY( OldCoord ), PROJZ( OldCoord) );


   if  B < -0.01  then  With Couleur.Coul  DO   BEGIN
            G :=  0.9;
            R :=  0.9;
            B :=  0.9;
   END
   ELSE  BEGIN
     Tmin := 0.3 ;
     TMax := 0.7 ;
     M    := 0.6 ;
     WITH Couleur.Coul DO BEGIN
         R := 1;
         G := 0;
         B := 0;
      END;
     TRANS := TRUE;
   END;

END;        {        ModifCoul        }
END.
```

```
MODULE          MZACT( INPUT, OUTPUT );

TYPE
   alfa     =   PACKED ARRAY [ 1..10 ] OF CHAR;
   SysCouleur  = ( RGB, HLS );
   Color = RECORD
           CASE SysCouleur OF
             HLS : (  H,L,S: Real );
             RGB : (  R,G,B: Real );
          END;
   RecCoul = RECORD
             Coul : Color;
             W,N  : Real;
          END;

PROCEDURE        ModifCoul( OldCoord, Coord, Normale  :  VECTOR;
                            FUNCTION  Hauteur( X, Y, Z : REAL ):REAL;
                            VAR  NOM : ALFA;   VAR  Couleur : RECCOUL ;
                          VAR  TMin, Tmax, M  : REAL ; VAR Trans : BOOLEAN);
VAR
      B, Z    :    REAL;
BEGIN
   Z := PROJZ( Coord );
     WITH Couleur.Coul DO
   if  Z < -20 THEN BEGIN
         R := 1;
         G := 0;
         B := 0;
   END  ELSE  IF  Z < -18 THEN BEGIN
         R := 1;
         G := 0.5;
         B := 0;
   END  ELSE  IF  Z < -15 THEN BEGIN
         R := 1;
         G := 1;
         B := 0;
   END  ELSE  IF  Z < -12 THEN BEGIN
         R := 0.5;
         G := 1;
         B := 0;
   END  ELSE  IF  Z < -10 THEN BEGIN
         R := 0;
         G := 1;
         B := 0;
   END  ELSE  IF  Z < -8 THEN BEGIN
         R := 0;
         G := 1;
         B := 0.5;
```

```
    END  ELSE  IF  Z < -6 THEN BEGIN
           R := 0;
           G := 1;
           B := 1;
    END  ELSE  IF  Z < -4 THEN BEGIN
           R := 0;
           G := 0.5;
           B := 1;
    END  ELSE  IF  Z < -1 THEN BEGIN
           R := 0;
           G := 0;
           B := 1;
    END  ELSE  BEGIN
           R := 1;
           G := 1;
           B := 0;
    END;

END;        {        ModifCoul        }
END.
```

4- ESPACE DE BROUILLARD

Pour qu'il y ait du brouillard à un endroit de l'écran, il doit y avoir une figure à cet endroit et celle-ci doit être dessinée en shading de Phong ou en shading constant. Aucune initialisation spéciale doit être faite, mais l'usager doit fournir une fonction décrivant l'espace de brouillard :

```
Procédure   FOG ( Point, Oeil : Vector; var CC : Color);
avec
  Type
    Color = Record
              R, G, B : Real;
            End;
```

"Point" représente la coordonnée de l'objet dans l'espace 3D et "oeil", la position de l'oeil. L'usager modifiera CC selon la forme/couleur du brouillard souhaité.

APPENDICE "B"

EXEMPLE DE PROGRAMMES

```
PROGRAM      SCENE( INPUT, OUTPUT , fichier);

TYPE
   FILINTEGER =      FILE OF INTEGER;
   IDENTFILE  =      PACKED ARRAY [1..40] OF CHAR;

PROCEDURE      AFFType ( Dithering : INTEGER );                    EXTERN;
PROCEDURE      ANTIALIAS ( N : INTEGER );                          EXTERN;
PROCEDURE      Debut_PIE;                                          EXTERN;
PROCEDURE      INIT_TEXT2(      F : FIG;
                        XMi, XMax, YMi, YMax, ZMi, ZMax :  REAL;
                        FUNCTION  Bump( X, Y, Z : REAL): REAL );  EXTERN;
PROCEDURE      INIT_TEXT(      F : FIG;
                      XMi, XMax, YMi, YMax, ZMi, ZMax :  REAL); EXTERN;
PROCEDURE      ClearMonitor;                                       EXTERN;
PROCEDURE      BckColor( H, L, S : REAL);                          EXTERN;
PROCEDURE      Diffuse( I:Vector );                                EXTERN;
FUNCTION      INQMAXVIEWPORT      :      VECTOR;                   EXTERN;
PROCEDURE      MONITOR( NBLigne : INTEGER );                       EXTERN;
TRANSFORM      PutColor      ( H, L, S      : REAL );              EXTERN;
PROCEDURE      PutNameFig( F : FIG; Nom : ALFA );                  EXTERN;
PROCEDURE      PERSCAM( O, I, U: VECTOR; A : REAL);                EXTERN;
PROCEDURE Point( N:Integer; I:Vector; Dir:Vector );                EXTERN;
PROCEDURE      SetRefl( F : FIG; NoFace : INTEGER;
                        WW, NN : REAL);                           EXTERN;
PROCEDURE      Extrema( F: FIG; VAR XMIN, XMAX, YMIN, YMAX, ZMIN, ZMAX :
                        VECTOR) ;                                 EXTERN;
PROCEDURE      CLOSEFILE(VAR FL : FILINTEGER; VAR ERROR : BOOLEAN );
                                                                  EXTERN;
PROCEDURE      READBINFIG(VAR FL : FILINTEGER; VAR FG : FIG);  EXTERN;
PROCEDURE      OPENFILE( VAR F: FILINTEGER; NAM : IDENTFILE;
                         KIND: INTEGER ; VAR ERROR : BOOLEAN) ;  EXTERN;
PROCEDURE      SetTexture( V :  BOOLEAN );                        EXTERN;
PROCEDURE      SetTranspa( V :  BOOLEAN );                        EXTERN;
PROCEDURE      SetModCoul( V :  BOOLEAN );                        EXTERN;
PROCEDURE      SetFog( V :  BOOLEAN );                            EXTERN;

VAR
   Oeil                  :      Vector;
   Repetition            :      REAL;
   Diviseur, Anti        :      INTEGER;
   Fichier               :      FILINTEGER;
   Erreur                :      BOOLEAN;
```

```
    Nom                     :       IdentFile;
    F1, F2, F3, F4          :       Fig;
    XMin, Xmax,
    YMin, Ymax,
    ZMin, Zmax              :       Vector;
    W                       :       REAL;
    NN, I                   :       INTEGER;

CONST
    Angle     =       28.0;

BEGIN
{
      1.0 Activation des espaces de texture
}
    BCKColor( 0.0, 0.75, 1.0 );
    Surface( TRUE );
    SetTexture( TRUE );
    SetModCoul( TRUE );
    SetFog( TRUE );

    Monitor( 2 );
    CLEARMONITOR;
{
      2.0 Initialisation des paramètres de texture
}
    Debut_PIE ;
{
      3.0 Lecture des paramètres de l'image
}
    CLEARMONITOR;
    WRITELN( '+Diviseur (0/(1..32)?');
    READLN  (Diviseur);
    WRITELN  (Diviseur);
    AffType( Diviseur );          { Choisi le type d'affichage }
    WRITELN;
    CLEARMONITOR;
    WRITELN( '+Antialiasing (1..4) ?');
    READLN( Anti );
    WRITELN( Anti );
    ANTIALIAS( Anti );
    WRITELN;
    CLEARMONITOR;
{
      4.0 Lecture du fichier et des figures
}
    WRITELN( '+Nom du fichier? ');
    READLN( Nom  );
    WRITELN( Nom  );
    WRITELN;
    CLEARMONITOR;
```

```
   OPENFILE( Fichier, NOM , 0, Erreur);
   RESET ( Fichier );
   READBINFIG( Fichier, F1 );
   READBINFIG( Fichier, F2 );
   READBINFIG( Fichier, F3 );
   READBINFIG( Fichier, F4 );
   closefile( fichier, Erreur);
{
     5.0 Lecture des informations manquantes pour le vase
}
   WRITELN( '+Nombre de répétition (vase)? ');
   READLN  (Repetition);
   WRITELN  (Repetition);
   WRITELN;
   CLEARMONITOR;

   WRITELN( '+Coefficient de reflexion (vase)(0..1)?');
   READLN  (w);
   WRITELN  (w);
   WRITELN;
   CLEARMONITOR;
   WRITELN( '+Facteur de highlight (vase)(1..200)?');
   READLN  (NN);
   WRITELN  (NN);
   WRITELN;
   CLEARMONITOR;
   FOR   I   :=  1 TO  NbPolygones(F1)   DO
      SetRefl( F1, I, W, nn);
{
     5.1 Calcul des dimensions de l'espace de texture
}
   Extrema( F1, Xmin, Xmax, Ymin, Ymax, Zmin, Zmax );
   IF PROJX(XMax) - PROJX(Xmin) = 0 THEN XMax := <<PROJX(XMin) + 1,
                                          PROJX(XMax), ProjX(XMax)>>;
   IF PROJY(YMax) - PROJY(Ymin) = 0 THEN YMax := <<PROJY(YMin),PROJY(YMax)
                                            + 1, ProjY(YMax)>>;
   IF PROJZ(ZMax) - PROJZ(Zmin) = 0 THEN ZMax :=<<PROJZ(ZMin),PROJZ(ZMax),
                                            ProjZ(ZMax)+1>>;
{
     5.2 Initialisation de l'espace de texture du vase
}
   PutNameFig( F1, 'VASE      ');
   INIT_TEXT(  F1, PROJX( XMin )/Repetition, PROJX( XMax )/Repetition,
                   PROJY( YMin )/Repetition, PROJY( YMax )/Repetition,
                   PROJZ( ZMin )/Repetition, PROJZ( ZMax )/Repetition );
```

```
{
    6.0 Maintenant, la table
}
  PutNameFig( F2, 'TABLE      ');
  WRITELN( '+Nombre de repetition pour la table? ');
  READLN  (Repetition);
  WRITELN  (Repetition);
  WRITELN;
  CLEARMONITOR;
  Extrema( F2, Xmin, Xmax, Ymin, Ymax, Zmin, Zmax );
  IF PROJX(XMax) - PROJX(Xmin) = 0 THEN XMax := <<PROJX(XMin) + 1,
                                        PROJX(XMax), ProjX(XMax)>>;
  IF PROJY(YMax) - PROJY(Ymin) = 0 THEN YMax := <<PROJY(YMin),
                                     PROJY(YMax) + 1, ProjY(YMax)>>;
  IF PROJZ(ZMax) - PROJZ(Zmin) = 0 THEN ZMax := <<PROJZ(ZMin),
                                      PROJZ(ZMax), ProjZ(ZMax)+1>>;

  INIT_TEXT(  F2, PROJX( XMin )/Repetition, PROJX( XMax )/Repetition,
                  PROJY( YMin )/Repetition, PROJY( YMax )/Repetition,
                  PROJZ( ZMin )/Repetition, PROJZ( ZMax )/Repetition );
{
    7.0 Les murs
}
  PutNameFig( F3, 'MURS       ');
  WRITELN( '+Nombre de repetition pour le mur? ');
  READLN  (Repetition);
  WRITELN  (Repetition);
  WRITELN;
  CLEARMONITOR;
  Extrema( F3, Xmin, Xmax, Ymin, Ymax, Zmin, Zmax );
  IF PROJX(XMax) - PROJX(Xmin) = 0 THEN XMax := <<PROJX(XMin) + 1,
                                        PROJX(XMax), ProjX(XMax)>>;
  IF PROJY(YMax) - PROJY(Ymin) = 0 THEN YMax := <<PROJY(YMin),
                                     PROJY(YMax) + 1, ProjY(YMax)>>;
  IF PROJZ(ZMax) - PROJZ(Zmin) = 0 THEN ZMax := <<PROJZ(ZMin),
                                      PROJZ(ZMax), ProjZ(ZMax)+1>>;

  INIT_TEXT(  F3, PROJX( XMin )/Repetition, PROJX( XMax )/Repetition,
                  PROJY( YMin )/Repetition, PROJY( YMax )/Repetition,
                  PROJZ( ZMin )/Repetition, PROJZ( ZMax )/Repetition );


  PutNameFig( F4, 'PLANCHER   ');

{
    8.0 Paramètre de l'image
}
  Window( << -70, -70>>, << 70, 70>>);
  Viewport( <<0,0>>, INQMAXVIEWPORT );

  Oeil       :=   << 0, 60,-250>>;
```

```
   Perscam( Oeil, << 0, 0, 0>>, <<0,1,0>>, Angle);
   Point( 1, << 0.6, 0.6, 0.6>>, << 160, -280, 90>>);
   Point( 2, << 0.3, 0.3, 0.3>>, << -90, -90, 10>>);
   Diffuse( << 0.3, 0.3, 0.3 >> );       { RGB }
{
     9.0 Affichage
}
   DRAW F1, F2, F3, F4;
   image;

      ClearMonitor;
      WRITELN( ' Fini');
end.
```

```
MODULE          DEBUT_TEX( INPUT, OUTPUT );

TYPE
   SysCouleur  = ( RGB, HLS );
   Color = RECORD
           CASE SysCouleur OF
             HLS : (  H,L,S: Real );
             RGB : (  R,G,B: Real );
          END;
   RecCoul = RECORD
             Coul : Color;
             W,N  : Real;
            END;

PROCEDURE       ClearMonitor;                                          EXTERN;
Function  Hasard( X, Y, Z : REAL; Repetition : INTEGER ):Real;  Extern;

VAR
   Fact     :     REAL;
   Per_X, Ampl_R  :  real;
   PERIODE :     INTEGER;
   Ampl     :     REAL;
   FactM, FactT     :     REAL;
   Per_XM, Ampl_RM :  real;
   PERIODEM :     INTEGER;
   AmplM     :     REAL;
   N         :          INTEGER;
   Omega     :  real;
   PX, PY    :          REAL;


FUNCTION     Bump( X, Y, Z : REAL ; Nom : ALFA): REAL;
CONST        PI  =  3.1415926;
{
   X et Y sont compris dans l'intervalle [0,1]
   Bump doit retourner une valeur centree sur 0,
}
VAR
   R : REAL;
BEGIN
   IF  Nom  = 'VASE        '  THEN
       BUMP := Fact * SIN( 2 * PI * ( Per_X * X + 0.1 * Ampl *
                        ( Ampl_R * Hasard(X,Y,Z,0) + 1 +
                         COS( 2 * PI * Y * Periode )) ) )
    ELSE IF  Nom  = 'MURS        '  THEN BEGIN
       R := ( X + Z ) / 2;
       BUMP := FactM * SIN( 2 * PI * ( Per_XM * r + 0.1 * AmplM *
                        ( Ampl_RM * Hasard(X,Y,Z,0) + 1 +
                         COS( 2 * PI * Y * PeriodeM )) ) )
    END
   ELSE IF  Nom  = 'TABLE       '  THEN;
```

```
END;  {       Bump        }

PROCEDURE  Debut_PIE;
BEGIN
   CLEARMONITOR;
   WRITELN;
   CLEARMONITOR;
{
     1.0 Lecture des paramètres du vase
}
   WRITELN( '+Periode en X (Vase) ?');
   READLN  ( Per_X );
   WRITELN  ( Per_X );
   WRITELN;
   CLEARMONITOR;

   WRITELN( '+Amplitude du decalage (vase)?');
   READLN  ( Ampl );
   WRITELN  ( Ampl );
   WRITELN;
   CLEARMONITOR;

   WRITELN( '+Amplitude du random (vase)?');
   READLN  ( Ampl_r );
   WRITELN  ( Ampl_r );
   WRITELN;
   CLEARMONITOR;

   WRITELN( '+Periode en Y (vase)?');
   READLN  ( Periode );
   WRITELN  ( Periode );
   WRITELN;
   CLEARMONITOR;

   WRITELN( '+Facteur de multiplication (vase)?');
   READLN  (fact);
   WRITELN  (fact);
   WRITELN;
   CLEARMONITOR;
{
     2.0 Lecture des paramètres de texture des murs
}
   WRITELN( '+Periode en X (Murs)?');
   READLN  ( Per_XM );
   WRITELN  ( Per_XM );
   WRITELN;
   CLEARMONITOR;

   WRITELN( '+Amplitude du decalage (Murs)?');
   READLN  ( AmplM );
```

```
    WRITELN  ( AmplM );
    WRITELN;
    CLEARMONITOR;

    WRITELN( '+Amplitude du random  (Murs)?');
    READLN  ( Ampl_rM );
    WRITELN  ( Ampl_rM );
    WRITELN;
    CLEARMONITOR;

    WRITELN( '+Periode en Y (Murs)?');
    READLN  ( PeriodeM );
    WRITELN  ( PeriodeM );
    WRITELN;
    CLEARMONITOR;

    WRITELN( '+Facteur de multiplication  (Murs)?');
    READLN  (factM);
    WRITELN  (factM);
    WRITELN;
    CLEARMONITOR;
{
      3.0 Lecture des paramètres de texture de la table
}
    WRITELN( '+ Table ');
    WRITELN;
    CLEARMONITOR;
    WRITELN( '+Nombre de sommation (3..7)?');
    READLN  ( N );
    WRITELN  ( N );
    WRITELN;
    CLEARMONITOR;

    WRITELN( '+Periode des sinus?');
    READLN  ( Omega );
    WRITELN  ( Omega );
    WRITELN;
    CLEARMONITOR;

    WRITELN( '+Amplitude du 1er sinus?');
    READLN  ( PX );
    WRITELN  ( PX );
    WRITELN;
    CLEARMONITOR;

    WRITELN( '+Amplitude du 2eme sinus?');
    READLN  ( PY );
    WRITELN  ( PY );
    WRITELN;
    CLEARMONITOR;
```

```
   WRITELN( '+Facteur de multiplication (Table)?');
   READLN  (factT);
   WRITELN  (factT);
   WRITELN;
   CLEARMONITOR;
   FactT  :=  FactT / 4;
end;



PROCEDURE        ModifCoul( OldCoord, Coord, Normale  :  VECTOR;
                            FUNCTION  Hauteur( X, Y, Z : REAL ):REAL;
                            VAR  NOM : ALFA;   VAR  Couleur : RECCOUL ;
                       VAR  TMin, Tmax, M  : REAL ; VAR Trans: BOOLEAN);
VAR
      I                 :      INTEGER;
      C, W, Temp, TempY  :  REAL;
      TT                :      REAL;

      X, Y, Z           :      REAL;
BEGIN
   IF  Nom  = 'TABLE      '  THEN BEGIN
      X := PROJX( OldCoord );
      Y := PROJY( OldCoord );
      Z := PROJZ( OldCoord );
      IF Y > 0.99 THEN BEGIN
      C  :=  1;
      Temp := 0;
      TempY := 0;
      W    := 2 * PI * Omega;
      FOR  I  :=  1 TO N DO  BEGIN
         Temp := C * (SIN( W * X + PX * sin(2*pi*Z)) + 1) + Temp;
         TEMPY := C * (SIN( W * Z + PY * SIN(2*PI*X)) + 1) + TempY;
         W  := 2 * W;
         C  := 0.707 * C;
      END;
      TT  :=  0.5 + FactT * ( Temp * TempY );
      IF  TT >  1  THEN TT := 1;
      WITH  Couleur.Coul  DO  BEGIN
         R  :=  TT * R;
         G  :=  TT * G;
         B  :=  TT * B;
      END;
   END   END
   END;        {        ModifCoul       }

PROCEDURE FOG( Point, Oeil : VECTOR; VAR CC : COLOR );
VAR
   X, Y, Z : REAL;
   TT      : REAL;
```

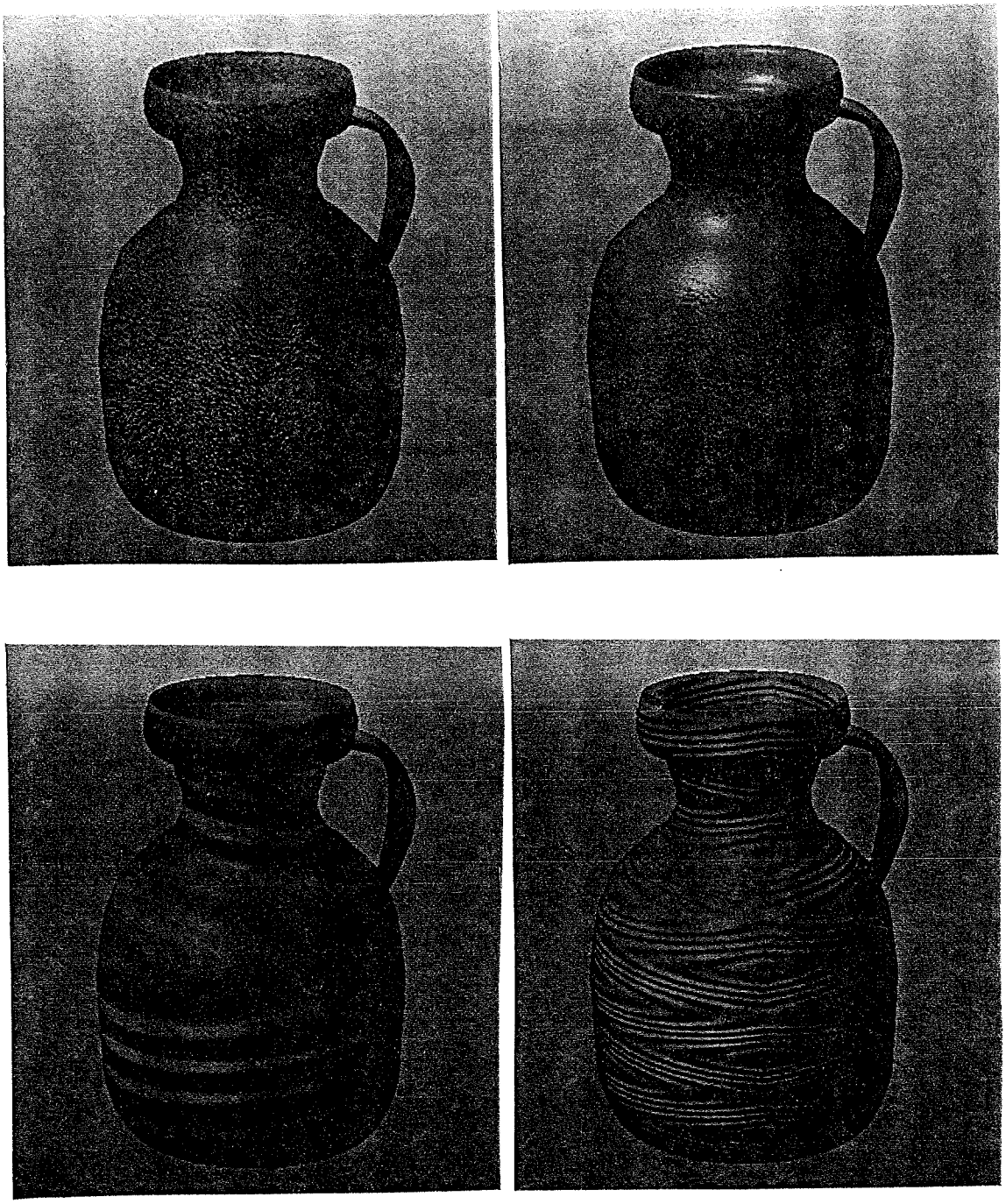
```
BEGIN
   X := PROJX( Point );
   Y := PROJY( Point );
   Z := PROJZ( Point );

   IF  Y  <  10.0  THEN  BEGIN
      TT  :=  Y / 15.0 + 0.33;
      CC.R  :=  1-TT + TT * CC.R;
      CC.G  :=  1-TT + TT * CC.G;
      CC.B  :=  1-TT + TT * CC.B;
   END;
END;
END.
```

APPENDICE "C"

PHOTOS

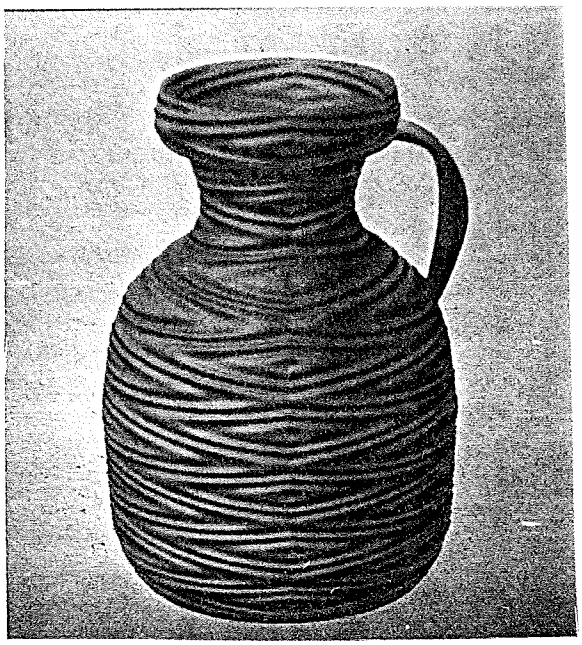
Les photos qui suivent ont toutes été prises sur un écran AED767 de résolution 767 x 575 avec une table de couleurs de 256 positions sauf les photos 9 et 34 qui ont été affichées sur un écran Raster Technology de résolution 1280 x 1024 avec 24 bits de couleur par pixel.

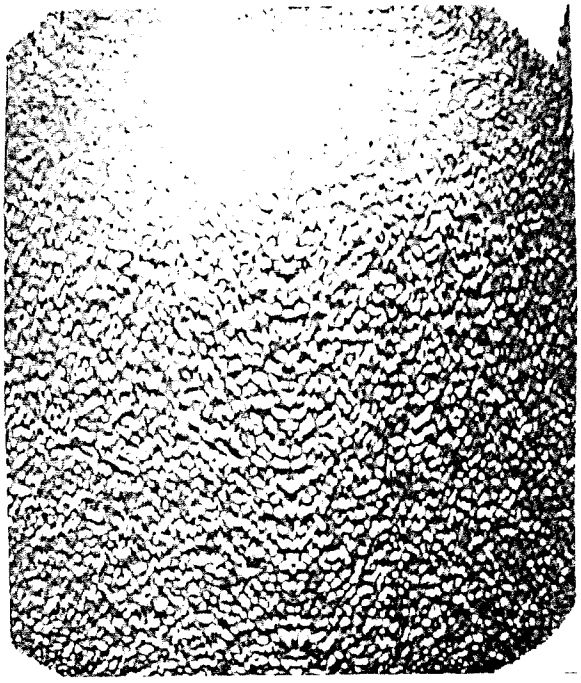


Photos 1, 2, 3 et 4

Photo 3 : A = 1, B = 20, C = 0, D = 1, Fact = 1, Répétition = 1

Photo 4 : A = 1, B = 20, C = 0, D = 1, Fact = 1, Répétition = 5



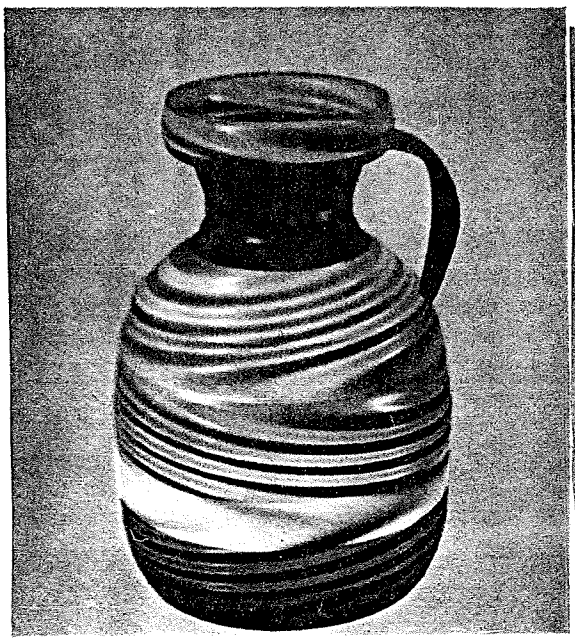


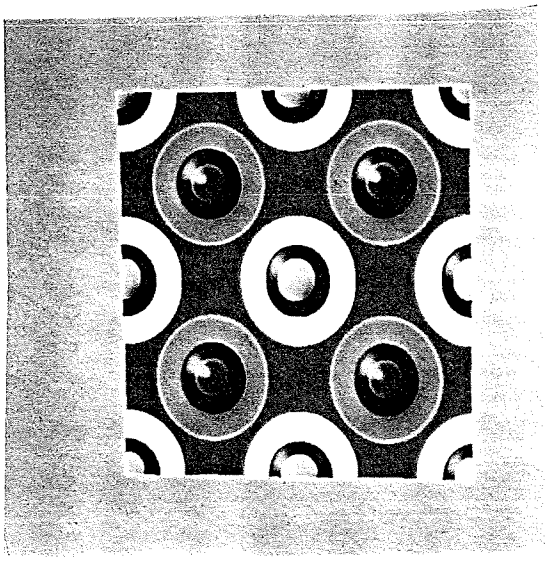
Photos 5, 6, 7 et 8

Photo 5 : A = 1, B = 10, C = 0, D = 2, Fact = 1.5, Répétition = 4

Photo 6 : A = 1, B = 50, C = 0, D = 1, Fact = 0.5, Répétition = 2

Photo 7 : A = 1, B = 10, C = 0, D = 10, Fact = 0.5, Répétition = 1



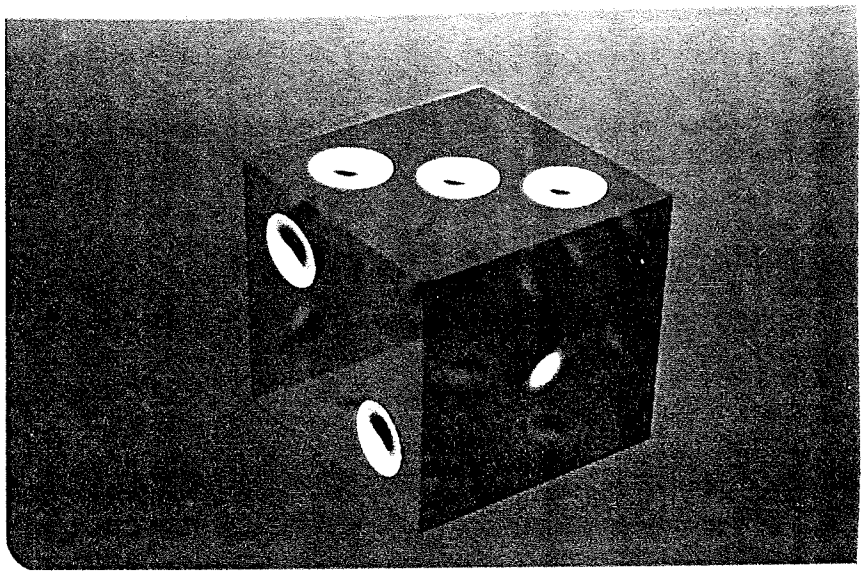


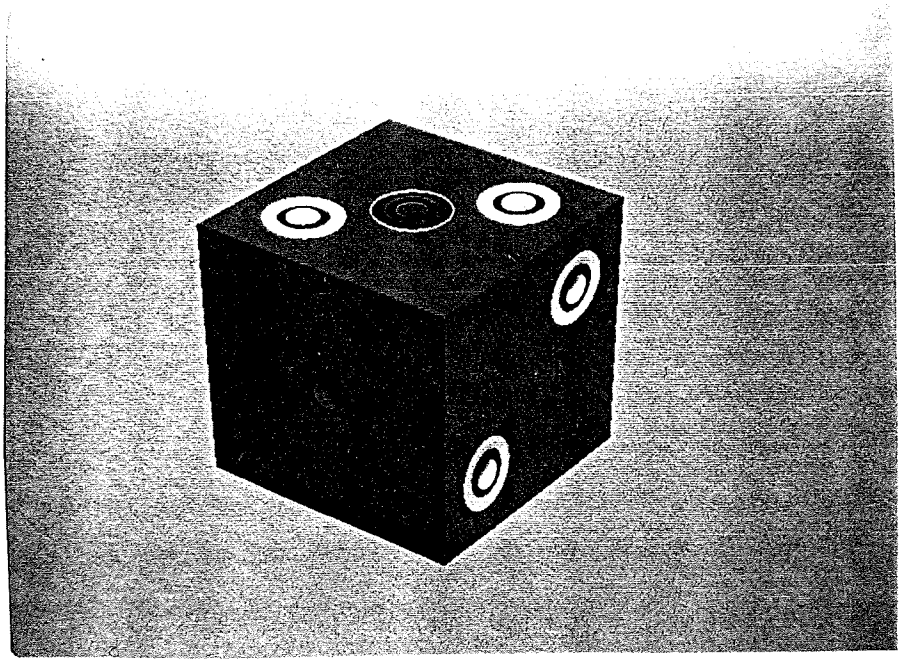
Photos 9, 10 et 11

Photo 9 : Couleur variant selon Y d'origine

Photo 10 : Couleur variant selon Z actuel

Photo 11 : Couleur variant selon l'espace de perturbation de la normale

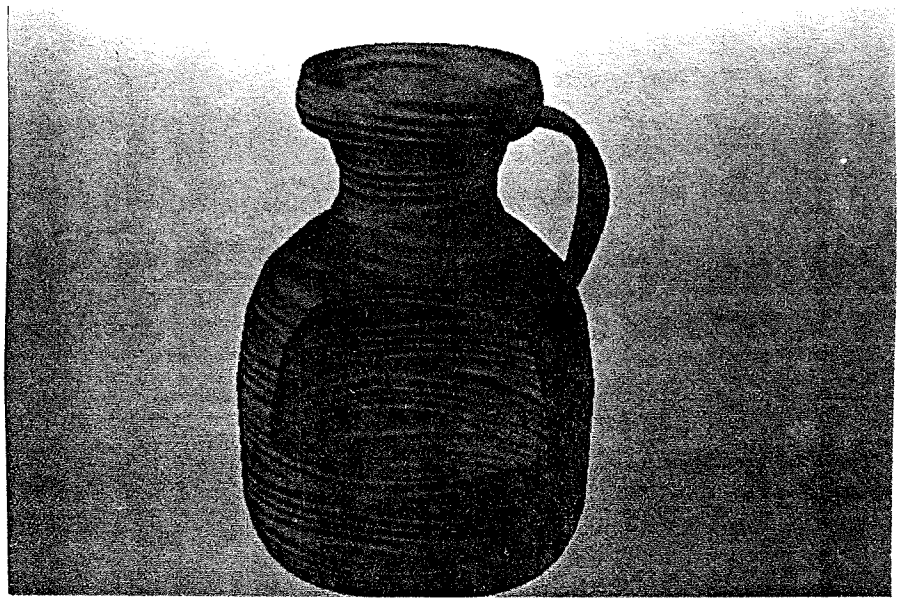


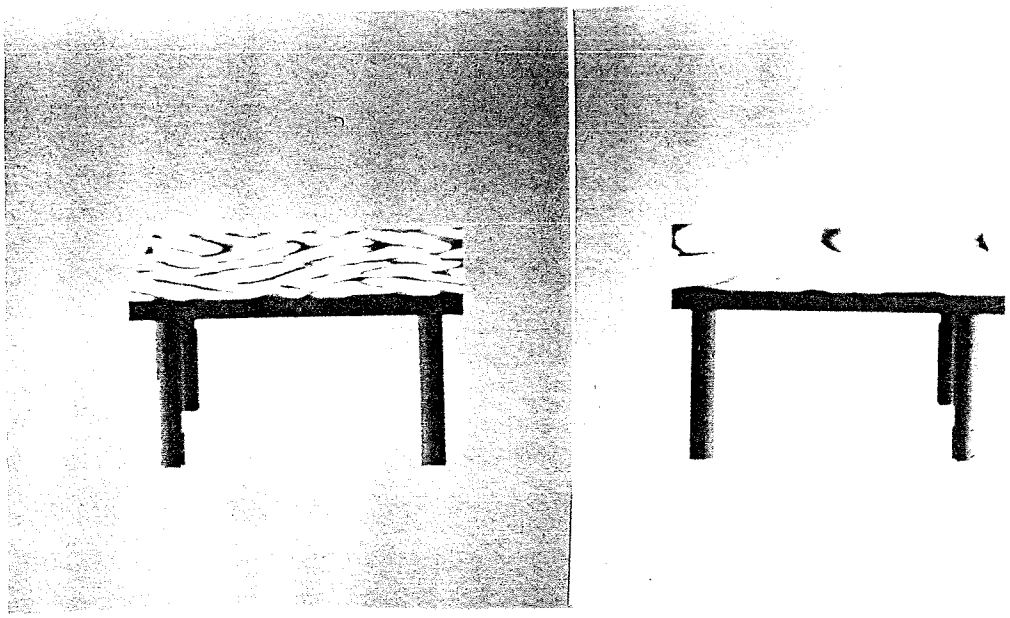


Photos 12 et 13

Photo 12 : Dé semi-transparent

Photo 13 : Dé avec couleur variant selon l'espace de perturbation de la normale

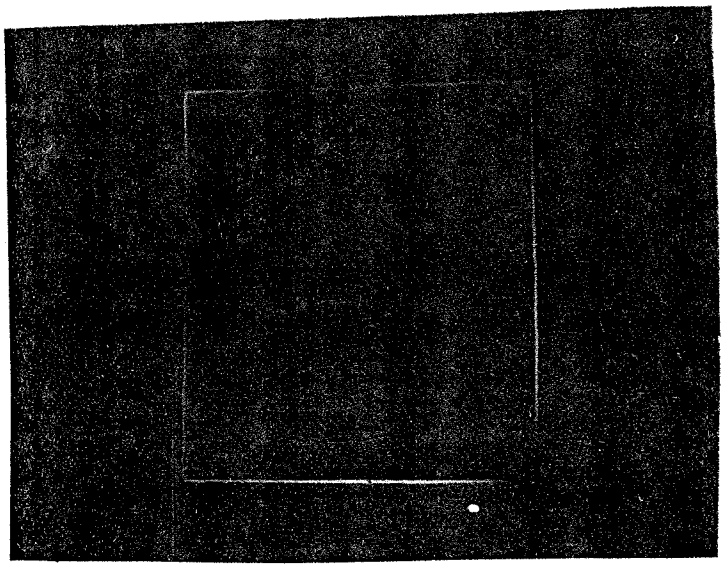


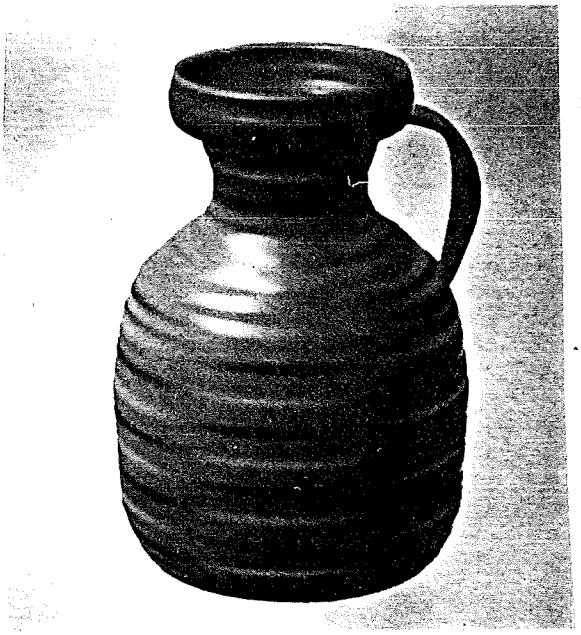


Photos 14, 15 et 16

Photo 14 : Vase coupé par l'espace de transparence

Photos 15 et 16 : Table modifiée par l'espace de couleur

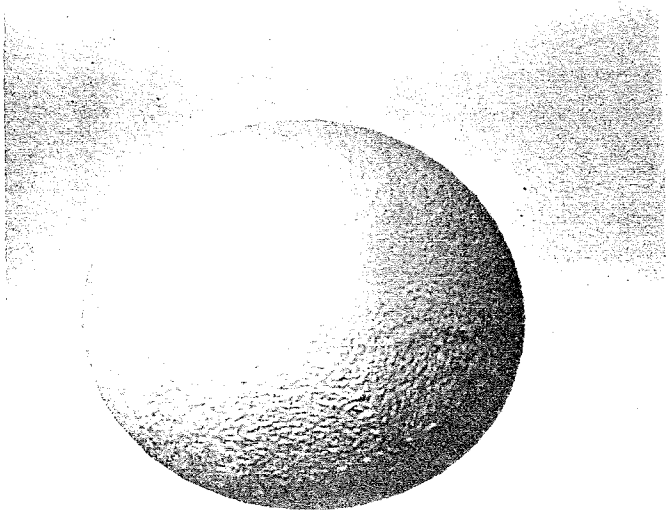


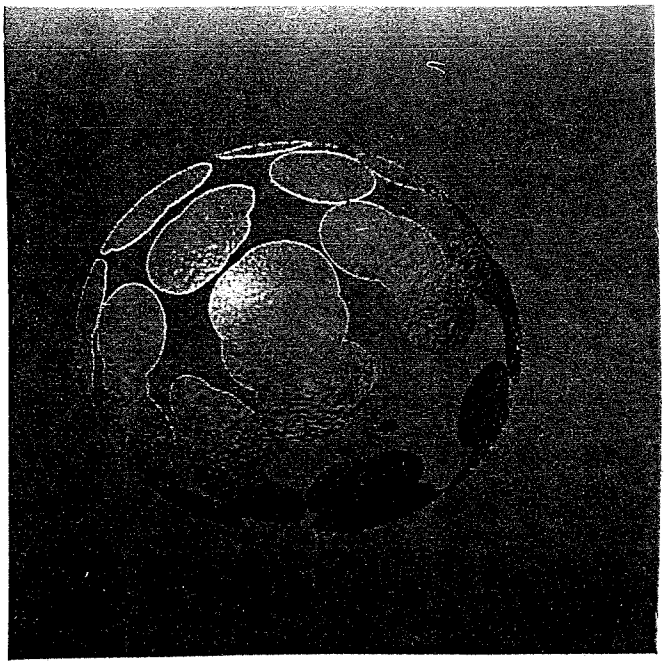


Photos 17, 18 et 19

Photo 18 : A = 1, B = 40, C = 0, D = 40, Fact = 1, Répétition = 1

Photo 19 : A = 1, B = 40, C = 0.1, D=40, Fact = 1, Répétition = 2

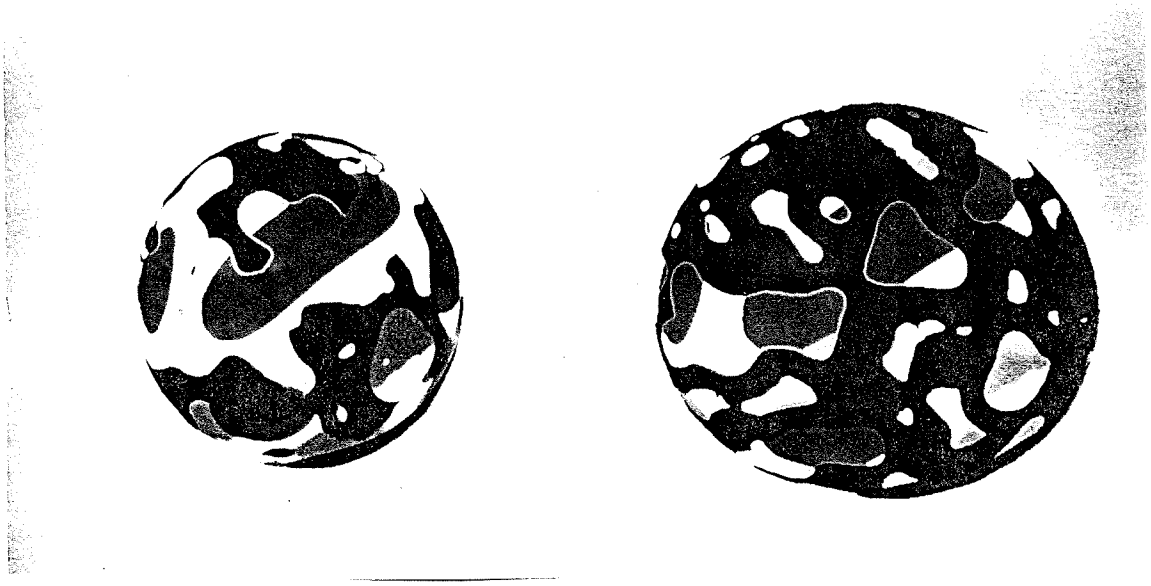


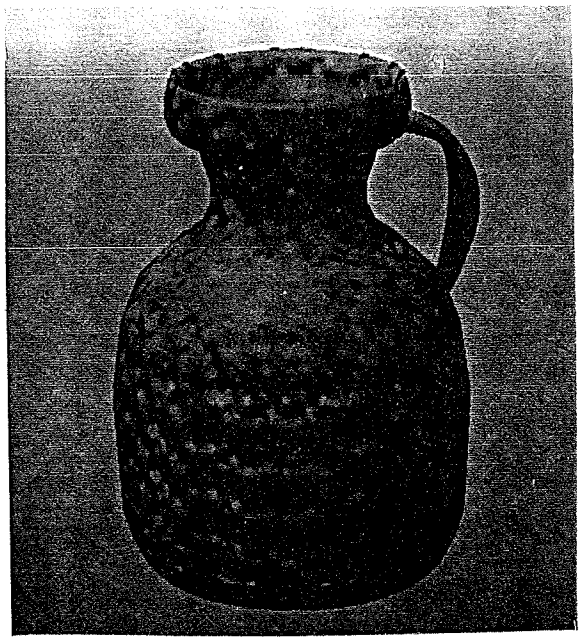


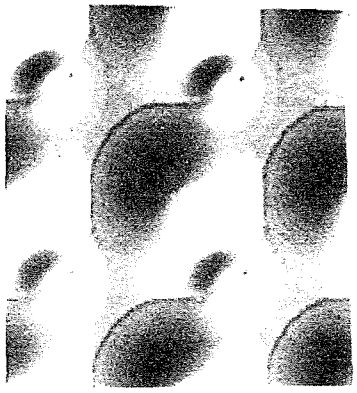
Photos 20 et 21

Photo 20 : "Orange" : A= 2, B= 40, C= 0.6, D= 1, Fact = 0.8, Répétition=4

Photo 21 : "Pelures d'orange"





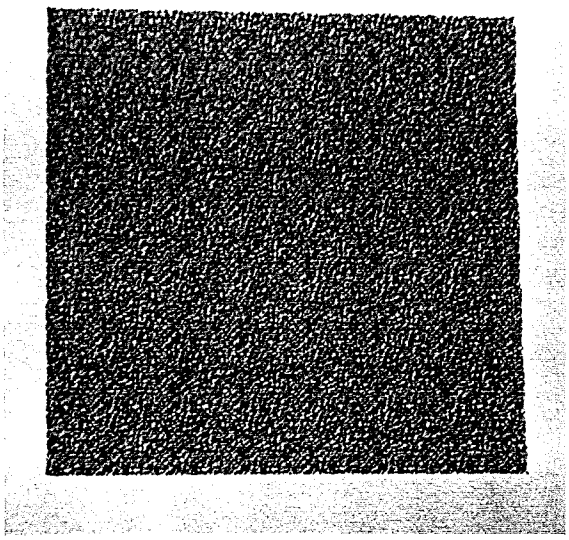


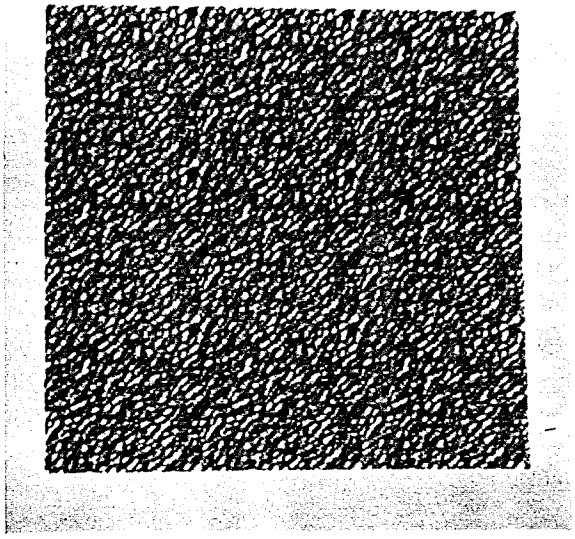
Photos 22, 23, 24 et 25

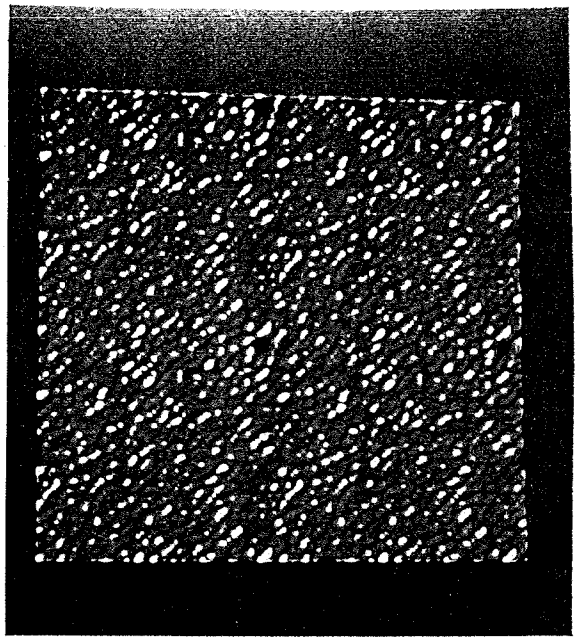
Photo 22 : # de trous = 40, Fact = 4, Répétition = 4

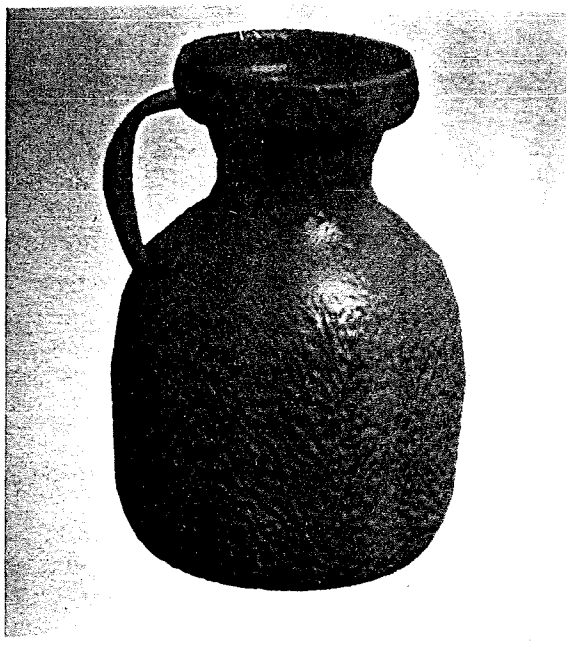
Photo 23 : # de trous = 40, Fact = 10, Répétition = 4

Photo 24 : # de trous = 40, Fact = 1, Répétition = 3

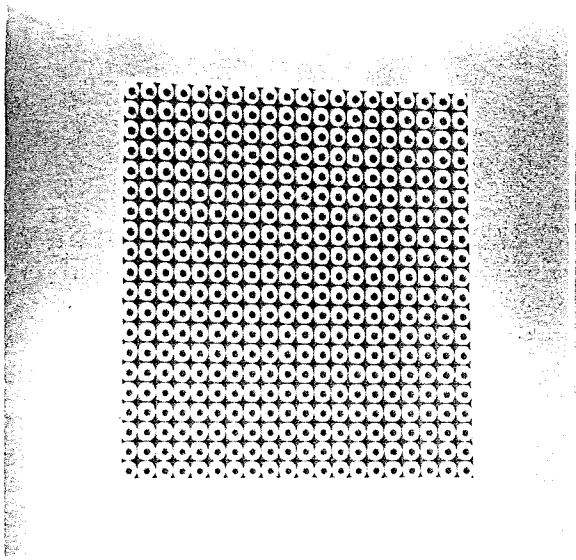


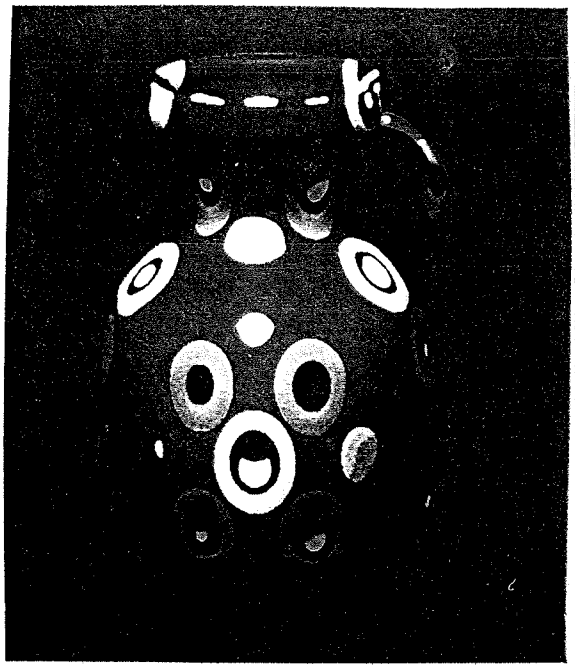


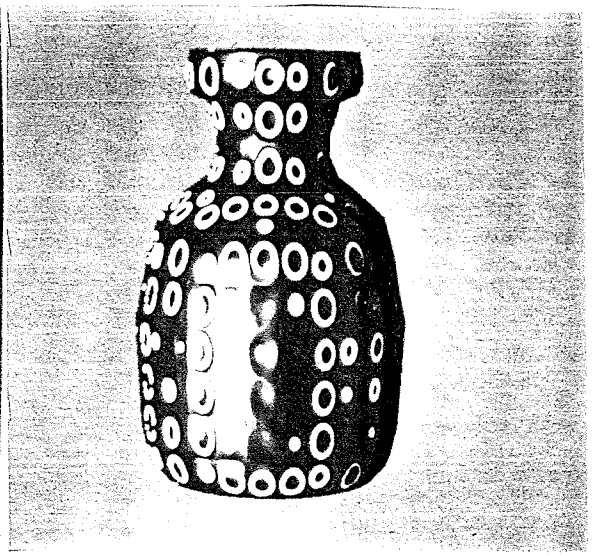




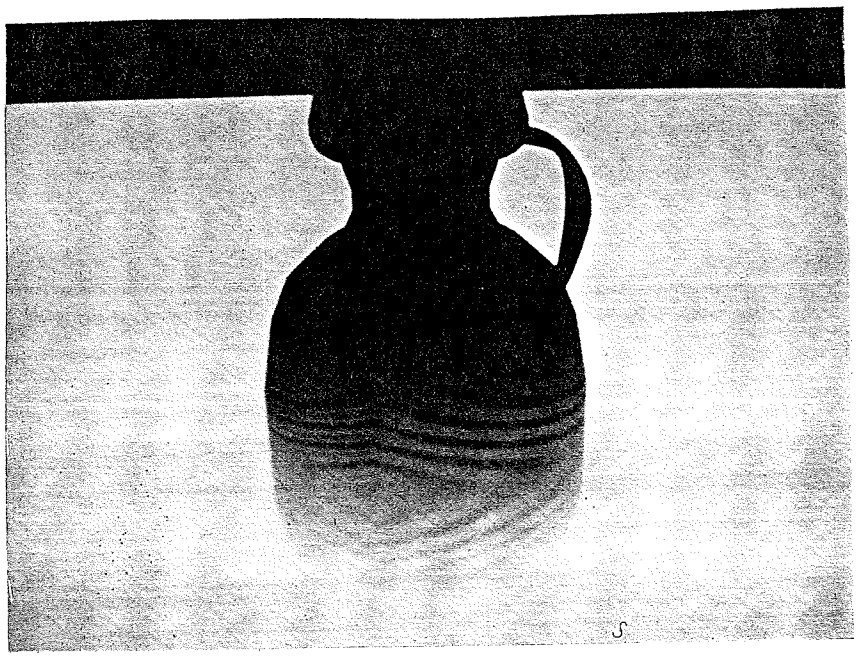
Photos 26, 27, 28 et 29

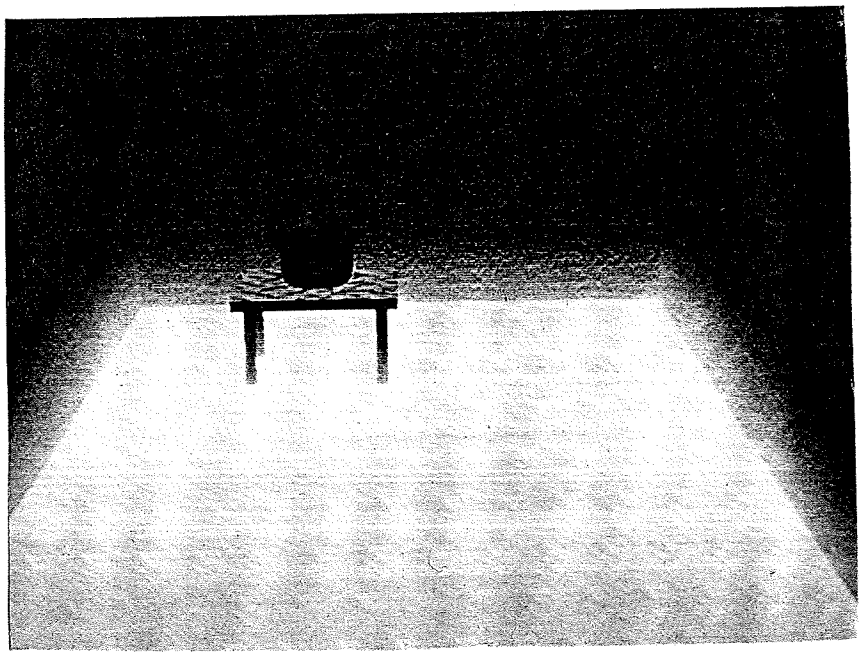






Photos 30, 31, 32 et 33





Photos 34 et 35

Photo 34 : Brouillard avec vase

Photo 35 : Brouillard dans une pièce