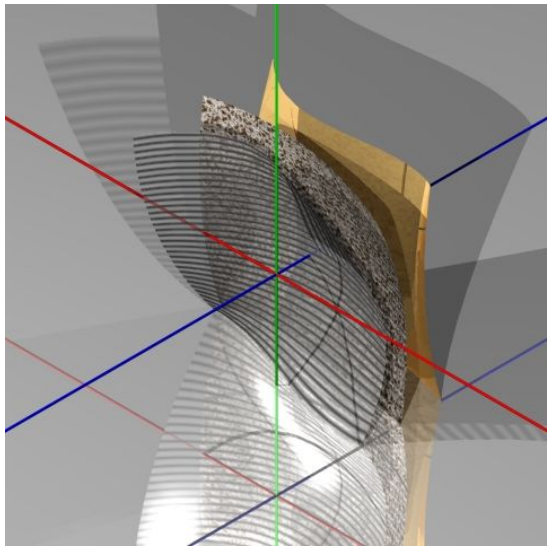
Cet ouvrage a été réalisé avec le concours du ministère de la Culture et de la Communication, direction de l'architecture et du patrimoine, bureau de la recherche architecturale et urbaine.

formes pascaliennes



un essai sur les formes gauches

alain marty

Editions
de l'Espérou

Table des matières

préface

L'ouvrage d'Alain Marty est le fruit du travail d'un auteur particulièrement exigeant, qui a longuement mûri le sujet qu'il nous propose sur les « Formes Pascaliennes ». La compréhension du dialogue entre « Formes et Forces », qui est au cœur de la collection des Editions de l'Espérou qui porte ce nom, nécessite une ouverture d'esprit qui allie dans un même mouvement des approches rationnelles et des approches sensibles.

La double formation d'Architecte et d'Ingénieur de l'auteur aurait pu suffire, mais il fallait une qualité supplémentaire en raison de l'algorithmique proposée pour la génération et la représentation des surfaces gauches qui constituent le propos de cet ouvrage. Cette qualité, c'est l'exigence de la simplicité qu'il offre dans une pratique de l'informatique ; cette exigence de simplicité, Alain Marty s'en fait quotidiennement le porte-parole dans sa fonction d'enseignant. C'est parce qu'il est à ce point de croisement et de pertinence, qu'Alain Marty peut nous guider « droit devant » sur le chemin des « espaces courbes ».

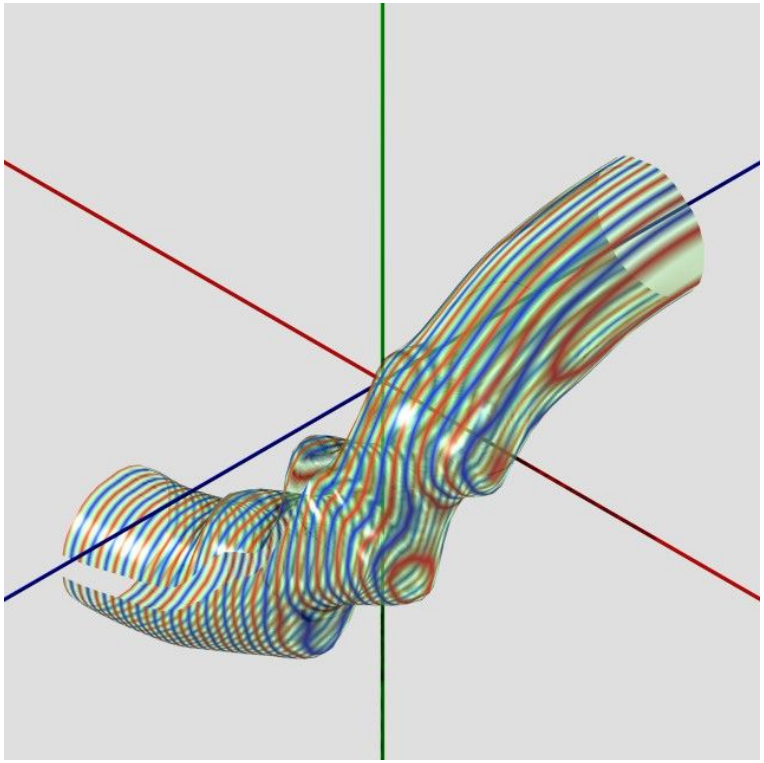
Robert Le Ricolais, dont les spécialistes des structures légères connaissent l'importance dans le domaine, insistait sur les « joies procurées par les mathématiques », ces joies qui ne pouvaient éclore, selon lui, sans « quelques larmes ». La lecture de cet essai sur les « surfaces gauches » ne peut, elle aussi, générer un état heureux que dans la mesure où le lecteur acceptera d'investir un peu de son temps.

Je suis convaincu, pour ma part, que ceux qui entreprendront cette démarche vérifieront ainsi que si trop souvent certains confondent l'idée avec l'outil, particulièrement dans le domaine de l'informatique, Alain Marty a su, lui, remettre au premier plan la géométrie et construire un outil à même de l'explorer dans sa complexité.

C'est un plaisir pour moi de l'en remercier dans ces quelques lignes qui veulent aussi témoigner d'une rencontre véritable au sein de l'équipe Structures Légères pour l'Architecture, de l'école d'Architecture Languedoc-Roussillon.

René Motro

professeur à l'Université de Montpellier II



un tubage modelé sur une cubique.

avant-propos

Cet essai sur les formes gauches est l'aboutissement (provisoire) d'un long chemin dans le domaine de la modélisation des formes, commencé en 1968 avec un travail de fin d'études sur les Coques Minces de forme libre, interrompu jusque dans les années 80 au profit d'un travail d'architecte de terrain construisant avec le té et l'équerre, repris alors sur les premiers micro-ordinateurs devenus abordables, puis accéléré en 1990 à l'occasion d'un contact fructueux à l'EALR avec René Motro, professeur à l'Université de Montpellier II et responsable du laboratoire GRSLA sur les Structures Spatiales Légères pour l'Architecture basé à l'Ecole d'Architecture Languedoc-Roussillon.

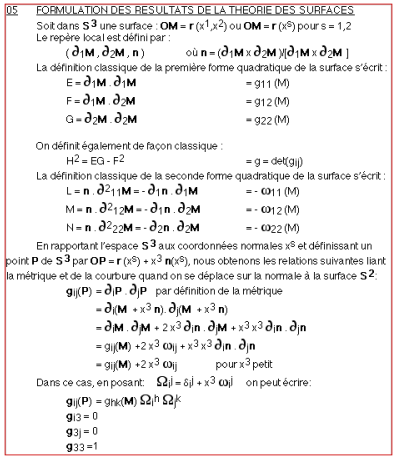
Ce travail est issu d'une réflexion « d'amateur », sans commande précise, dans une totale liberté, hors de tout cursus et de toute convention universitaire. Il est basé sur un ensemble de connaissances issues de lointaines études, sur une auto-formation construite sur des informations glanées ça et là dans les livres que l'on peut trouver dans les librairies en ville. C'est le produit d'une volonté constante d'aboutir à une approche claire et unitaire du monde merveilleux mais complexe des « formes gauches » et du besoin de partager cette réflexion avec d'autres « explorateurs », amateurs atypiques ou institutionnels, mathématiciens ou pas, théoriciens ou praticiens, experts de l'Art du Trait des maîtres-constructeurs du Moyen-Age ou virtuoses des nouvelles machines à modéliser de l'Infographie.

J'ai eu la chance de trouver chez René Motro cet intérêt qui m'a donné l'envie de reprendre le travail dans la voie des formes à points de contrôle (Bézier, splines), et je l'en remercie. J'ai eu également la chance de rencontrer dans les années 95 une équipe de chercheurs du laboratoire GRCAO de l'université de Montréal qui m'ont témoigné de la sympathie, je pense notamment à Giovanni de Paoli et à Claude Parisel. Et à l'école d'architecture de Montpellier, d'avoir eu l'écoute permanente et critique d'un Thierry Berthomier féru de Morphologie Structurale, et de temps en temps de quelques étudiants acceptant de s'extraire de leurs rendus d'architecture pour écouter (un temps) une présentation originale des formes courbes, je pense notamment à Vinicius Raducanu qui a depuis fait son chemin. Je souhaite trouver à travers cet essai d'autres passionnés des formes gauches qui pourraient aider à faire avancer encore cette exploration, qui pourraient peut-être l'utiliser dans leur propre travail, et/ou m'accompagner un temps dans le chemin. Je souhaite au moins au lecteur de prendre quelque plaisir à la lecture de cet essai.

Et je remercie Colette, qui a bien voulu faire semblant de me croire toutes les fois que je lui promettais n'en avoir plus que pour un quart d'heure avant de finir "vraiment" cette étude ...

alain marty

villeneuve de la raho, juillet 2004


05 FORMULATION DES RESULTATS DE LA THEORIE DES SURFACES

Soit dans S^3 une surface : $\mathbf{OM} = \mathbf{r}(x^1,x^2)$ ou $\mathbf{OM} = \mathbf{r}(x^s)$ pour s = 1,2
Le repère local est défini par :

$(\partial_1\mathbf{M}, \partial_2\mathbf{M}, \mathbf{n})$ où $\mathbf{n} = (\partial_1\mathbf{M} \times \partial_2\mathbf{M})/|\partial_1\mathbf{M} \times \partial_2\mathbf{M}|$

La définition classique de la première forme quadratique de la surface s'écrit :

$E = \partial_1\mathbf{M} \cdot \partial_1\mathbf{M} = g_{11}(M)$
$F = \partial_1\mathbf{M} \cdot \partial_2\mathbf{M} = g_{12}(M)$
$G = \partial_2\mathbf{M} \cdot \partial_2\mathbf{M} = g_{22}(M)$

On définit également de façon classique :

$H^2 = EG - F^2 = g = \det(g_{ij})$

La définition classique de la seconde forme quadratique de la surface s'écrit :

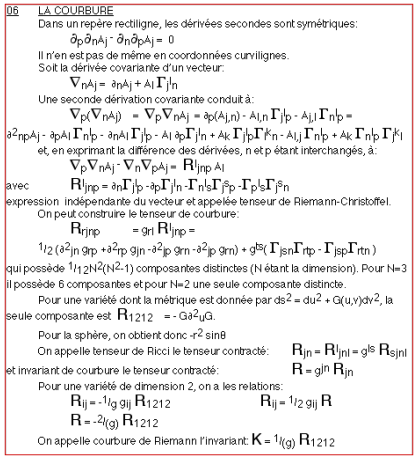
$L = \mathbf{n} \cdot \partial^2_{11}\mathbf{M} = -\partial_1\mathbf{n} \cdot \partial_1\mathbf{M} = -\omega_{11}(M)$
$M = \mathbf{n} \cdot \partial^2_{12}\mathbf{M} = -\partial_1\mathbf{n} \cdot \partial_2\mathbf{M} = -\omega_{12}(M)$
$N = \mathbf{n} \cdot \partial^2_{22}\mathbf{M} = -\partial_2\mathbf{n} \cdot \partial_2\mathbf{M} = -\omega_{22}(M)$

En rapportant l'espace S^3 aux coordonnées normales x^s et définissant un point **P** de S^3 par $\mathbf{OP} = \mathbf{r}(x^s) + x^3\,\mathbf{n}(x^s)$, nous obtenons les relations suivantes liant la métrique et de la courbure quand on se déplace sur la normale à la surface S^2:

$\mathbf{g}_{ij}(P) = \partial_i P \cdot \partial_j P$ par définition de la métrique
$= \partial_i(\mathbf{M} + x^3\,\mathbf{n}) \cdot \partial_j(\mathbf{M} + x^3\,\mathbf{n})$
$= \partial_i\mathbf{M} \cdot \partial_j\mathbf{M} + 2\,x^3\,\partial_i\mathbf{n} \cdot \partial_j\mathbf{M} + x^3\,x^3\,\partial_i\mathbf{n} \cdot \partial_j\mathbf{n}$
$= g_{ij}(\mathbf{M}) + 2\,x^3\,\omega_{ij} + x^3\,x^3\,\partial_i\mathbf{n} \cdot \partial_j\mathbf{n}$
$= g_{ij}(\mathbf{M}) + 2\,x^3\,\omega_{ij}$ pour x^3 petit

Dans ce cas, en posant: $\Omega_{ij} = \delta_{ij} + x^3\,\omega_{ij}$ on peut écrire:

$\mathbf{g}_{ij}(P) = g_{hk}(\mathbf{M})\,\Omega_i{}^h\,\Omega_j{}^k$
$\mathbf{g}_{i3} = 0$
$\mathbf{g}_{3j} = 0$
$\mathbf{g}_{33} = 1$


06 LA COURBURE

Dans un repère rectiligne, les dérivées secondes sont symétriques:

$\partial_p\partial_n A_j - \partial_n\partial_p A_j = 0$

Il n'en est pas de même en coordonnées curvilignes.
Soit la dérivée covariante d'un vecteur:

$\nabla_n A_j = \partial_n A_j + A_l\,\Gamma^l_{jn}$

Une seconde dérivation covariante conduit à:

$\nabla_p(\nabla_n A_j) = \nabla_p\nabla_n A_j = \partial_p(A_{j,n}) - A_{l,n}\,\Gamma^l_{jp} - A_{j,l}\,\Gamma^l_{np} =$
$\partial^2_{np}A_j - \partial_p A_l\,\Gamma^l_{np} - \partial_n A_l\,\Gamma^l_{jp} - A_l\,\partial_p\Gamma^l_{jn} + A_k\,\Gamma^l_{jp}\Gamma^k_{ln} - A_{l,j}\,\Gamma^l_{np} + A_k\,\Gamma^l_{np}\,\Gamma^k_{jl}$

et, en exprimant la différence des dérivées, n et p étant interchangés, à:

$\nabla_p\nabla_n A_j - \nabla_n\nabla_p A_j = R^l_{jnp}\,A_l$

avec $R^l_{jnp} = \partial_n\Gamma^l_{jp} - \partial_p\Gamma^l_{jn} - \Gamma^l_{ns}\Gamma^s_{jp} - \Gamma^l_{ps}\Gamma^s_{jn}$

expression indépendante du vecteur et appelée tenseur de Riemann-Christoffel.

On peut construire le tenseur de courbure:

$R_{rjnp} = g_{rl}\,R^l_{jnp} =$

$1/2\,(\partial^2_{jn}g_{rp} + \partial^2_{rp}g_{jn} - \partial^2_{jp}g_{rn} - \partial^2_{rn}g_{jp}) + g^{ts}(\Gamma_{jsn}\Gamma_{rtp} - \Gamma_{jsp}\Gamma_{rtn})$

qui possède $1/12\,N^2(N^2-1)$ composantes distinctes (N étant la dimension). Pour N=3 il possède 6 composantes et pour N=2 une seule composante distincte.

Pour une variété dont la métrique est donnée par $ds^2 = du^2 + G(u,v)dv^2$, la seule composante est $R_{1212} = -G\partial^2_{uu}G$.

Pour la sphère, on obtient donc $-r^2\sin\theta$

On appelle tenseur de Ricci le tenseur contracté: $R_{jn} = R^l_{jnl} = g^{ls}\,R_{sjnl}$

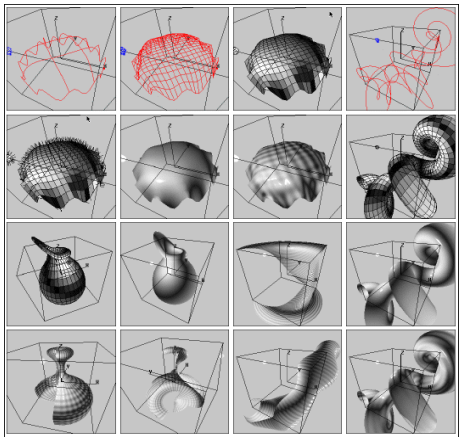
et invariant de courbure le tenseur contracté: $R = g^{jn}\,R_{jn}$

Pour une variété de dimension 2, on a les relations:

$R_{ij} = -1/g\;g_{ij}\,R_{1212}$ $\quad R_{ij} = 1/2\;g_{ij}\,R$
$R = -2/(g)\;R_{1212}$

On appelle courbure de Riemann l'invariant: $K = 1/(g)\;R_{1212}$

Des formules différentielles ...



... aux outils informatiques !

introduction

Construisez quatre triangles équilatéraux à l'aide de six allumettes. Telle est la question rappelée par Bernard Werber dans son premier livre sur les fourmis, assortie de l'énigmatique recommandation : "Pensez différemment !", clé de la solution. En pensant différemment, les Grecs inventèrent la Théorie des Coniques, rassemblant en un tout cohérent des courbes aussi diverses que le cercle, l'ellipse, la parabole et l'hyperbole, et aussi la droite et le point. Bien plus tard, les mathématiciens inventèrent le corps des nombres complexes "afin" qu'une équation de degré N possède "toujours" N solutions. Le chimiste Mendeleev imagina une belle classification pour les éléments primaires apparemment si différents constituant l'univers connu, et au-delà, un formidable outil pour la découverte de nouveaux éléments. En écrivant les formules de la mécanique et de l'électrodynamique dans un espace quadridimensionnel, Einstein réussit non seulement à en unifier les concepts fondamentaux, mais encore à créer les conditions de nouvelles découvertes, dont la célèbre formule $\mathbf{E} = m.c^2$ est la plus connue. Cas illustres ...

Une question complexe posée dans notre espace courant, à trois dimensions et bien réel, se reformule souvent plus simplement lorsqu'on on imagine "pour un temps" un espace un peu plus complexe dans lequel on la transpose. C'est une façon de diviser la difficulté pour mieux la vaincre. Par exemple, l'étude de la stabilité d'un voile courbe mince devient plus facile quand le problème est posé dans la géométrie courbe de ce voile et non dans l'espace euclidien orthogonal qui l'entoure.

Mais à elle seule la géométrie des courbes et des surfaces gauches fournit un exemple de problème complexe, difficile à modéliser, à représenter et à visualiser. Les mathématiciens des siècles derniers l'ont bien balisée de belles formules différentielles mais celles–ci restent le plus souvent hors du champ d'application d'une géométrie "descriptive" autorisant le dessin à l'aide d'outils simples comme la règle et le compas ou le simple dessin à main levée. Au-delà des formes élémentaires de la géométrie classique, les droites, les cercles, les plans, les sphères, etc…, il faut avouer qu'il n'y a rien de facilement manipulable sans le secours des outils informatiques.

Dans ses applications à l'infographie et à la CAO, l'informatique a mis à notre disposition une modélisation opérationnelle des formes géométriques classiques, et a également mis en lumière une nouvelle famille de formes gauches très pratiques à manipuler sur l'écran (Béziers, Splines, Nurbs, carreaux de Coons,..). Mais, du fait de l'orientation opérationnelle des outils informatiques développés, ces formes échappent à toute appréhension directe et a fortiori manuelle, les algorithmes développés dans une littérature imposante et parfois indigeste restant complexes ou tout simplement cachés au fond des boites noires logicielles. A ma connaissance, aucune approche unitaire accessible au commun des mortels ne s'est réellement dégagée qui donne un meilleur éclairage à ces formes et qui crée les conditions d'en découvrir de nouvelles.

C'est l'objectif de cet essai sur les formes gauches.

De Casteljau a proposé en 1959 un algorithme qui porte son nom, un algorithme récursif fondamental et étonnamment simple, une construction géométrique très intuitive menant à une puissante théorie, un type d'algorithme basé sur une approche «*gestuelle*» qui paraît décalée, vieillotte, et à coté duquel on peut passer de nombreuses fois sans jamais n'y voir qu'un simple croquis destiné à accompagner les formules algébriques, analytiques et matricielles qui peuplent la littérature sur le sujet. Un simple croquis quasi ésotérique relevant de l'art du trait des constructeurs du Moyen-Age ...

Le présent essai lui donne une place centrale et en fait une application systématique à l'aide d'une poignée d'opérateurs géométriques élémentaires ; il apporte quelques éléments à une géométrie descriptive de formes dites «*pascaliennes*», établissant un pont entre les formes classiques de la géométrie et les nouvelles formes mises en lumière par l'informatique, en définissant des formes dont les règles autorisent au final le dessin à main levée, avec un bout de corde comme unique guide… Sont ainsi reconstruites les formes connues sous le nom de courbes et surfaces de Bézier, par la simple utilisation d'opérateurs géométriques initialement appliqués à un couple de points de l'espace puis à un nombre quelconque de ces formes.

Cette approche unitaire, intuitive et basée sur une formulation analytique réduite au minimum, permet d'aborder sans grande difficulté le cas des formes immergées dans d'autres formes gauches, et d'en déduire aisément des formes plus complexes comme les Splines, les Nurbs (dont les coniques sont un cas particulier d'une grande importance), les tubages, les carreaux de Coons, etc..., le tout à l'aide d'opérateurs de projection, de concaténation et de simples combinaisons linéaires.

L'ensemble des *formes «pascaliennes»*, ou «*pFormes*», muni de ces opérateurs constitue ainsi une géométrie descriptive, un outil apportant une aide à la compréhension directe et «manuelle» de formes dont l'expression analytique peut parfois être très complexe, un outil pouvant accompagner l'exploration de formes inédites jusque dans des dimensions supérieures.

Le présent document est divisé en trois grandes parties :

1. **construction, définition :** cette partie présente une approche progressive des pFormes mettant en évidence le caractère unitaire de formes apparemment éloignées les unes des autres - comme le cube et la ... cubique - et conduisant à la définition générale des pFormes ;
2. **opérations, propriétés :** cette partie présente les opérations qu'il est possible de réaliser sur les pFormes -subdivisions, élévations du degré, ...-, et les propriétés fondamentales des pFormes - repères tangents, interpolations, immersions dans d'autres pFormes, ... - ;
3. **compositions, applications :** cette partie analyse quelques compositions particulières de pFormes, leurs rapports avec les coniques, notamment avec les surfaces de révolution, et pose les premiers jalons d'une géométrie dans les surfaces gauches.

implémentation informatique

C'est à la main qu'a commencé cette exploration des formes gauches, et les croquis à main levée ont été le premier support de la réflexion. Dessiner une cubique en utilisant un logiciel du commerce n'apporte en général que peu d'informations sur la mécanique interne des courbes produites ; on consomme, c'est tout. Mais l'esprit qui guide le crayon sur le papier ou la souris sur l'écran a souvent tendance à prendre ses désirs pour la réalité et il arrive que le dessin produit n'accompagne pas vraiment le raisonnement que l'on a en tête et amène à une conception erronée de la forme envisagée. Les raisonnements algébriques viennent alors assister le dessin, mais peut-on vraiment faire confiance au résultat obtenu au prix de cinq ou six pages de calculs serrés où une simple erreur de signe peut complètement faire échouer le raisonnement ?

Autant les logiciels manipulant les formes géométriques s'avèrent de peu d'intérêt dans l'étude de leurs fondements, autant les formidables outils que sont les langages de programmation peuvent venir soutenir la réflexion au niveau le plus fondamental. Reste le choix du bon langage et ce n'est pas une mince question : choisir les bons outils de développement, le bon environnement, la bonne plateforme, tout ceci suppose un lourd investissement à consacrer dans l'apprentissage de tous ces éléments. Le Basic, Pascal, Hypertalk, Logo, C, C++, Renderman, Open GL, Java, Java 3D, il en faut du temps pour en maîtriser le vocabulaire et la syntaxe, le tout en parallèle avec la réflexion purement géométrique à mener, et à ne pas oublier dans le maquis des outils informatiques. On a vite fait de passer plus de temps à tenter de réinventer l'algorithme de remplissage d'un polygone 3D à afficher dans une fenêtre que de réfléchir à la nature de la forme que l'on veut modéliser et étudier sur l'écran.

D'où l'intérêt d'un outil comme POVRAY, logiciel libre fonctionnant sur toutes les plateformes basé sur un moteur de rendu RayTracing et développé par une communauté d'universitaires et de chercheurs de tous pays, dont le site se trouve à l'adresse suivante : http://www.povray.org. Au-delà de son excellent moteur de rendu, de ses nombreuses capacités (opérations booléennes par exemple) et des nombreux objets prédéfinis, le logiciel POVRAY dispose d'un véritable langage de développement ; il est possible de définir des variables globales et locales, des chaînes, des tableaux, des structures de test et de contrôle permettant de créer des boucles, des macros qui se comportent comme de véritables procédures et fonctions, embarquant des paramètres, retournant des valeurs et pouvant de plus être appelées de façon récursive, ce qui est la propriété fondamentale exigée dans la présente étude. Ce logiciel constitue un véritable cadeau pour le chercheur en Infographie et quand on constate de plus que le langage embarqué est très classique et se rapproche beaucoup du langage C, le langage "universel", on "sait" qu'il sera toujours possible sans trop d'efforts de porter son travail sur n'importe quelle autre plateforme logicielle.

Les opérateurs étudiés ont fait l'objet d'une implémentation sur POVRAY et ont été incorporés dans une bibliothèque contenue dans un fichier 'pFlibs.inc' à inclure en tête des fichiers 'pFscene.pov' rendus par POVRAY. Cette implémentation a accompagné la mise au point des opérateurs de base et de nombreux autres, en permettant une visualisation confortable des formes produites, un contrôle «visuel» de leur validité, et en produisant au final des images de grande qualité. Par contrôle visuel il faut entendre que les pages de calculs serrés qui sont censés aboutir à la démonstration de tel ou tel résultat sont remplacées par des pages de code informatique tout aussi complexe. La différence est que ce code informatique constitue un ensemble d'algorithmes exécutés fidèlement et sans fatigue par l'ordinateur, autant de fois qu'on le veut et dans toutes les conditions possibles et imaginables. Et que cette machine a la facheuse tendance à exécuter exactement ce qu'on lui dit de faire et pas ce qu'on voudrait qu'elle fasse ! En général, le résultat n'est pas tout à fait ce que l'on attend et commence alors un long travail d'allers et retours jusqu'à ce qu'arrive enfin le résultat attendu, ... ou bien jusqu'à ce qu'on soit totalement convaincu de l'ineptie du concept envisagé.

POVRAY a vraiment été d'un grand secours dans l'aboutissement de cette étude. A tel point que la syntaxe initiale utilisée pour développer le raisonnement dans cet ouvrage, a été rapidement accompagnée et même parfois remplacée par celle de POVRAY. Par exemple, dans la syntaxe utilisée pour présenter les concepts géométriques, la définition d'une parabole définie par trois points s'écrit :

```
pL3 = MIR(p0,p1,p2), MIR opérateur récursif appliqué à trois
points
```

étant entendu que l'opérateur récursif est supposé appliqué à un niveau infini de récursion. Dans la syntaxe utilisée dans l'implémentation sur POVRAY, la définition et la représentation de la parabole s'écrivent :

```
#local pL3 = array[3] { p0, p1, p2 }
pFdraw( 1, pL3, finesse(3) + courbe(0.01) + ma_couleur(<1,0,0>) )
```

ce qui peut se lire ainsi : une variable locale (pL3, cf notation plus loin dans le texte) associée à la parabole est déclarée et définie comme un tableau de trois points (p0, p1,p2) ; la macro pFdraw() applique au tableau une récursion de niveau 3 (finesse(3)), produisant un tableau de 9 points et la dessine sous la forme d'une polyligne passant par ces neufs points, composée de cylindres de rayon 0.01 (courbe(0.01)) et de couleur rouge (ma_couleur(<1,0,0>)) .

De façon générale, le découplage sera total entre la déclaration et la représentation d'une forme ; une parabole sera définie par la donnée d'un tableau de trois points, sans souci du choix de sa représentation, l'épaisseur et la couleur du trait bien sûr, mais surtout le niveau de récursion, à tel point qu'on ne distinguera pas entre le polygone de contrôle de la parabole défini par les trois points initiaux et la parabole générée par une récursion infinie.

On trouvera en fin d'ouvrage le listing complet des opérateurs utilisables sous forme de macros, compilées dans un fichier, "pFlibs.inc" à inclure en tête des fichiers "pFscene.pov" pour analyser leur fonctionnement et produire des images ou des animations.

rappels de géométrie

Au sommaire de cette section :

Afin de mieux situer la « problématique » des formes gauches (des courbes, des surfaces, des volumes gauches), il paraît nécessaire de donner quelques points de repère sur la géométrie élémentaire et les formes primitives, sur la géométrie cartésienne et les équations et de proposer une première tentative de classification, sinon d'unification, des formes gauches.

01 la géométrie élémentaire et les formes primitives

Si bien peu de gens ont lu les "Eléments d'Euclide", une somme qui est restée la base de la géométrie jusqu'à nos jours, tout le monde a plus ou moins transpiré sur quelques figures tracées à la règle et au compas, sur les cas d'égalité des triangles, sur le théorème de Thalès et les parallèles, et il faut bien avouer que pour beaucoup, la géométrie s'arrête là, aux points, aux droites aux triangles et aux cercles. Quelques-uns s'aventurent au-delà dans le coin des polyèdres, et se perdent à jamais dans la jungle des icosaëdres, dodécaèdres rhombiques et autres formes utiles au cristallographe... Quant aux formes gauches comme la trajectoire d'un oiseau flanant dans le ciel, la courbe d'une vague déferlante, les hanches de la Danaïde de Rodin, elles semblent définitivement hors d'atteinte de toute méthode simple de construction, de toute tentative de description unitaire.

La forme courbe la plus simple est le cercle que l'on sait tracer à l'aide d'un piquet et d'une corde de longueur constante ; et la surface de la sphère est construite de la même façon. Mais vu de biais un cercle n'est plus un cercle et apparemment plus rien n'est constant, ni la distance de chaque point au centre, ni la courbure en chaque point, et le problème se pose de sa construction directe sur un plan sans référence au cercle dont elle est la vue biaise. Cette courbe que l'on appelle « ellipse », présente deux symétries par rapport à deux axes orthogonaux et l'idée est venue un jour de marquer deux points sur le grand axe, symétriques par rapport au petit et de faire coulisser une corde fermée tendue en triangle entre ces deux points et un point quelconque de l'ellipse ; par tatonnements successifs, il est apparu que pour une position particulière de ces deux points, la longueur totale de la corde restait constante et on appela ces deux points les foyers de l'ellipse. On venait ainsi de trouver la construction d'une courbe généralisant le cercle et son rayon constant ; le monde des courbes contrôlées de courbure variable était désormais ouvert.

Mais une autre courbe attirait également l'attention des géomètres, la courbe suivie (en première approximation) par un caillou lancé de biais en l'air ou par les jets d'eau puissants d'une fontaine. Cette courbe, qu'on appelle parabole, ne se retournait pas sur elle-même comme l'ellipse à laquelle elle ressemblait localement, et semblait même partir à l'infini ! La construction de cette courbe qu'on avait fini par découvrir était moins évidente que celle de l'ellipse et faisait intervenir un point sur l'axe de symétrie (appelé lui aussi foyer) et une droite orthogonale à cet axe (appelée directrice) ; rien à voir avec l'ellipse donc et encore moins avec le cercle. L'unification de ces courbes allait voir le jour avec la « Théorie des Coniques ».

La « Théorie des Coniques » que les Grecs nous ont révélée met en évidence de façon élégante et simple les relations fondamentales entre des courbes aussi différentes en apparence qu'un cercle, une ellipse et une parabole : toutes ces courbes sont des coniques, des sections d'un cône à base circulaire par un plan plus ou moins incliné par rapport à l'axe du cône. C'est tout, mais il fallait faire l'effort de sortir de l'espace à deux dimensions où se trouvaient ces trois courbes et raisonner dans un espace à trois dimensions où il serait possible de décrire un cône et ses intersections avec un plan. "Pensez différemment !", clé de la solution, comme le rappelait Bernard WERBER.

Et comme c'est souvent le cas, la fenêtre ouverte sur un nouvel espace faisant apparaître de nouvelles choses, en inclinant un peu plus fort le plan d'intersection, on découvrit les deux arcs d'une nouvelle courbe, l'hyperbole, le concept d'asymptote et au final une représentation unitaire de six éléments importants de la géométrie : le point, la droite, le cercle, l'ellipse, la parabole et l'hyperbole.

On connaît l'importance de Pythagore et de son théorème sur le triangle rectangle qui s'écrit en langage moderne : $a^2+b^2=c^2$; on sait le désespoir des Pythagoriciens quand ils appliquèrent cette formule à la mesure de la diagonale d'un carré de coté 1 et qu'ils s'aperçurent (et démontrèrent) qu'il était impossible d'extraire sous la forme d'un nombre entier ou fractionnaire la racine carrée du nombre 2 ; ce nombre était tout à fait "irrationnel" et il était déraisonnable d'en parler ; point final. Ils en déduisirent qu'il devait être absolument interdit (sous peine de mort) de relier le monde de la géométrie à celui de l'arithmétique et il fallut attendre deux mille ans avant que Descartes transgresse cet interdit, associe systématiquement aux objets géométriques des relations entre nombres et en recherche l'"équation".

02 la géométrie cartésienne et les équations

Dans cette approche de la géométrie, l'idée de base est d'associer à chaque point de l'espace un ensemble de nombres appelés "coordonnées", deux dans le plan (x,y), trois dans l'espace (x,y,z) ; chacun de ces nombres pourra prendre toute valeur dans l'intervalle]-infini, +infini[, dans un ensemble connu sous le nom de corps des Réels, R. On dira que R^2 est l'ensemble des points d'un espace à deux dimensions, que R^3 est l'ensemble des points d'un espace à trois dimensions. Et on définira les objets géométriques (droites, cercles, plans,...) par les relations existant entre les coordonnées - les équations - contraignant les points de ces objets à rester dans une partie limitée de R^2 ou de R^3. Nous abordons ici les équations implicites, les équations paramétriques et les équations différentielles.

021 équations implicites

L'équation implicite d'une courbe est une relation écrite sous la forme f(x,y) = 0, qui a l'avantage du traitement symétrique des coordonnées, ce qui n'est pas le cas de la forme dite explicite plus connue : y = f(x). Rapportée à un couple d'axes Ox et Oy, une droite du plan R^2 est le lieu des points P(x,y) tels que y varie proportionnellement à x, y/a = x/b, ce qui s'écrit habituellement sous la forme explicite suivante :

```
y = a/b.x + d, où d est l'ordonnée du point sur l'axe Oy,
```

et sous la forme implicite :

```
f(x,y) = ax + by - d = 0.
```

On retrouve de la même façon l'équation implicite d'une parabole :

$$f(x,y) = y - ax^2 - d = 0,$$

où d est l'ordonnée du point sur l'axe Oy. L'équation d'un cercle de rayon r centré sur l'origine des axes découle de l'application immédiate du théorème de Pythagore :

$$x^2 + y^2 = r^2,$$

et s'écrit sous forme implicite :

$$f(x,y) = x^2 + y^2 - r^2 = 0,$$

On trouve de même sans difficulté l'équation implicite d'une sphère dans l'espace :

$$f(x,y) = x^2 + y^2 + z^2 - r^2 = 0.$$

L'équation d'un plan est un peu plus complexe à trouver. On définit d'abord dans l'espace R^3 un plan passant par l'origine des axes à l'aide du vecteur unitaire qui lui est normal en ce point N (a,b,c), puis un plan parallèle à celui-ci et situé à une distance d ; soit H le point d'intersection de ce plan et de la droite portant N. Tout point P(x,y,z) de ce plan est tel que sa projection sur la droite portant N est le point H, autrement dit tel que le produit scalaire N•OP = OH, et l'équation s'écrit :

```
ax + by + cz - d = 0.
```

Que peut être l'équation implicite f(x,y,z) = 0 d'une droite dans l'espace R^3 ? Il n'existe pas en fait une telle équation et l'on est amené à considérer une droite de l'espace comme intersection de deux plans et à définir un point P(x,y,z) comme solution d'un système de deux équations linéaires :

```
ax  + by  + cz  - d  = 0
a'x + b'y + c'z - d' = 0
```

On voit donc que dans la représentation des objets les plus simples sous la forme d'équations implicites, il est difficile de trouver une unité de forme et des règles générales, chaque forme suppose une approche spécifique et cette approche dépend de l'espace dans lequel on se trouve, sa dimension, ses courbures...

022 équations paramétriques

Les équations paramétriques offrent plus de facilités pour définir de façon élégante bon nombre de formes géométriques. Il s'agit d'une approche dans laquelle les coordonnées d'un point d'une forme vont être exprimées en fonction d'un ou de plusieurs paramètres généralement confinés dans une intervalle unitaire [0,1].

Soit un espace a priori de dimension quelconque (disons 2 ou 3), O le point origine des axes et deux points quelconques P0 et P1 de cet espace ; un point du segment porté par PO et P1 peut s'exprimer sous la forme :

```
OP(t) = OP0 + P0P1.t    avec t € [0,1]
```

équation exprimant le vecteur OP comme somme du vecteur OP0 et d'un vecteur porté par P0P1 et de longueur réduite de celle de P0P1 dans la proportion t. Mais on privilégiera l'écriture équivalente suivante, plus élégante, sans référence au point origine et symétrique par rapport aux points P0 et P1 :

```
OP(t) = OP0 + (OP1-OP0).t
      = (1-t).OP0 + t.OP1
      = (1-t).P0 + t.P1
```

où l'on voit que P se déplace de façon linéaire de P0 à P1 quand t varie de 0 à 1. Noter que

cette équation vectorielle est équivalente dans l'espace R^3 à trois équations scalaires en x, y et z. En fait, cette écriture permet de ne pas avoir à se soucier du nombre de dimensions et l'on s'en servira abondamment par la suite. Cette écriture permet de définir simplement toute une famille de formes "linéaires" ; on mentionnera - sans autre explication pour l'instant - la portion de surface deux fois réglée connue sous le doux nom de *paraboloïde hyperbolique* (appelé aussi PH) et définie à partir de 4 points quelconques de l'espace (P00, P01, P10, P11) par l'expression bilinéaire suivante :

```
P(t) = (1-u).(1-v).P00 + u.(1-v).P01 + (1-u).v.P10 + u.v.P11,
       avec u,v €[0,1],
```

et l'arc de parabole obtenu à partir de ce PH en égalant u et v (u = v = t) et totalement défini (contrôlé) par 3 nouveaux points P0, P1, P2 :

```
P(t) = (1-t)^2.P0 + 2.u.(1-t).P1 + t^2.P2,     avec t €[0,1],
      et P0 = P00, P1 = (P01 + P10)/2, P2 = P11
```

constituant les premiers jalons d'une famille de formes abordées au chapître 1, formes contrôlées respectivement par 4 et 3 points.

Concernant le cercle - centré sur l'origine et de rayon unité - l'équation paramétrique est immédiatement basée sur les fonctions trigonométriques fondamentales :

```
P(t) = [ cos(A) , sin(A) ]       avec A € [0, 2.π],
```

ce qui n'arrange rien quand on sait que ces fonctions trigonométriques sont des fonctions transcendantes, inexprimables sous la forme d'un polynome en nombre fini de termes ; en définissant une nouvelle variable t = tangente(A/2), on peut se passer des fonctions trigonométriques et trouver une expression rationnelle - cad écrite sous forme de quotient, de ratios :

```
x = (1-t^2)/(1+t^2)
y = 2.t/(1+t^2)
```

Il est difficile de trouver un lien entre cette expression et celle du segment de droite, mais nous verrons dans le chapître 3 comment on peut rattacher cette expression à celle d'une parabole, grâce à la Théorie des Coniques...

Pour la sphère, l'expression paramétrique classique en u et v :

```
P(t) = [ r.cos(u).cos(v), r.cos(u).sin(v), r.sin(u) ]
```

se transforme difficilement sous forme rationnelle et nous attendrons également le chapitre 3 pour l'étudier plus avant.

023 équations différentielles

Certaines formes géométriques ne peuvent souvent être définies qu'à partir de propriétés dites "différentielles". C'est le cas de l'altitude d'une pomme qui tombe d'un pommier, d'une bille qui roule sur une surface gauche dans l'espace en dehors de toute gravitation, de la surface d'équilibre d'une lame de savon tendue dans un fil de fer.

0231 : la pomme qui tombe

La fonction z(t) qui régit l'altitude de la pomme qui tombe d'une branche du pommier satisfait à trois règles simples :

```
z(0)  = h,  hauteur initiale de l'arbre
z'(0) = 0,  vitesse initiale nulle (sauf écureuil taquin)
z"(t) = -g, accélération de la pesanteur (constante)
```

et la solution est une fonction parabolique z(t) obtenue par une double intégration :

```
z'(t) = -1.g.t
z(t)  = -1/2.g.t^2 + h
```

0232 : la bille qui roule

La trajectoire de la bille qui roule sans frottement et sans dérapage sur une surface gauche dans l'espace en dehors de toute gravitation s'appelle une géodésique ; cette courbe est telle que la poussée exercée par la bille sur la surface est toujours perpendiculaire à la surface au point de contact, égale et opposée à la réaction de la surface, toute force tangentielle annulée. On peut exprimer mathématiquement cette propriété en écrivant que l'accélération de la bille est un vecteur orthogonal à la surface, mais un peu (beaucoup) de géométrie différentielle est nécessaire pour celà.

Un point quelconque M d'une surface est défini dans l'espace par les expressions suivantes :

```
M(u,v) = [ x(u,v), y(u,v), z(u,v) ]      avec u,v € [0,1]
```

expressions appelées équations paramétriques de la surface. Le même point peut également être défini comme appartenant à une courbe immergée dans la surface sous la forme de l'équation paramétrique suivante :

```
M(t) = [ u(t), v(t) ]            avec t € [0,1]
```

La dérivée première (vitesse) de M(t) par rapport à t s'exprime en fonction des dérivées partielles $\partial_u M$ et $\partial_v M$ au point M(t) à la surface :

```
dM/dt = ∂M/∂u.du/dt + ∂M/∂v.dv/dt
```

ou pour simplifier l'écriture :

```
M'= ∂uM.u' + ∂vM.v'
```

On calcule ensuite la dérivée seconde (accélération) :

```
M" = dM'/dt
   = d(∂uM'.u' + ∂vM'.v')/dt
   = d(∂uM'.u')/dt + d(∂vM'.v')/dt
   = (∂²uuM.u'+ ∂²uvM.v').u' + ∂uM.u"
    +(∂²uvM.u'+ ∂²vvM.v').v' + ∂vM.v"
   = ∂uM.u" + ∂vM.v" + ∂²uuM.u'² + 2.∂²uvM.u'.v' + ∂²vvM.v'²
```

Sachant que les dérivées partielles ∂uM et ∂vM définissent des vecteurs tangents à la surface en M(t), il ne reste plus qu'à exprimer l'orthogonalité de ces vecteurs avec le vecteur accélération en annulant leurs produits scalaires :

```
M"•∂uM = 0,
M"•∂vM = 0
```

qui est le système différentiel que doit satisfaire M(t) pour que la courbe suivie soit une géodésique. On ne sait pas intégrer ce système dans le cas général, et même dans la plupart des cas ; seules des solutions approchées peuvent être obtenues et l'analyse des propriétés de ces courbes géodésiques est extrêmement complexe. C'est bien dommage quand on sait qu'une autre propriété de ces géodésiques est qu'elles sont le chemin le plus court entre deux points quelconques d'une surface (sous certaines conditions de proximité) et qu'elles sont aux surfaces courbes ce que sont les droites dans l'espace euclidien, l'être géométrique fondamental dont tout ou presque dérive, les polygones, les triangles, la définition des angles, du parallélisme, etc... Les surfaces gauches semblent former un monde quasiment inaccessible à l'exploration, si l'on exclut l'approche informatique qui donne à voir plus qu'à comprendre.

0233 : la lame de savon

La nature nous offre toutes sortes de surfaces d'équilibre, des courbes funiculaires, des surfaces minimales. La surface d'équilibre d'une lame de savon tendue dans un fil de fer est telle que chaque point se place en position moyenne par rapport à ses voisins ; pour commencer imaginons un maillage "discret" - non continu - se projetant sur un carré (x,y) et écrivons que l'altitude z(i,j) de chaque point est égale à la moyenne arithmétique des altitudes des points en croix :

```
z[i,j] = ( z[i-1,j] + z[i+1,j] + z[i,j-1] + z[i,j+1] ) / 4
```

ce qui s'écrit aussi :

```
z[i-1,j] + z[i+1,j] + z[i,j-1] + z[i,j+1] - 4.z[i,j] = 0
```

ou bien

```
( z[i-1,j] - 2.z[i,j]+ z[i+1,j] ) +
( z[i,j-1] - 2.z[i,j]+ z[i,j+1] ) = 0
```

expression dans laquelle on reconnaît la somme de deux différences finies du second ordre qui amène, quand on passe à un maillage continu, à une équation aux dérivées partielles connue sous le nom d'équation de Laplace :

```
Δ( z ) = ∂2z/∂x2 + ∂2z/∂y2 = 0
```

Cette équation étonnamment simple (ne pas oublier la forme discrète qui n'est rien d'autre qu'une moyenne arithmétique) est fondamentale en physique mathématique, elle est présente dans l'étude de nombreux phénomènes complexes, de l'électro-magnétisme à l'élasticité. On peut se demander comment une expression aussi simple peut décrire les formes complexes que peut prendre une lame de savon. Alan Turing, mathématicien et logicien anglais considéré comme une des pères de l'informatique et de l'intelligence artificielle avait, paraît-il, l'habitude de dire «la Science est une équation différentielle, La Religion est une condition aux limites» . C'est là peut-être la raison qui fait que cette équation, étudiée par des générations de mathématiciens n'a pas trouvé de solution dans le cas général, tout au moins exprimable de façon simple sous forme implicite ou paramétrique. On est donc bien loin de posséder les outils qui nous seraient bien utiles dans la recherche de relations entre les coniques et les géodésiques d'une surface minimale, pour prendre un exemple qui sera abordé un peu plus loin !

Remarque : ouvrez un tableur acceptant le calcul itératif et capable d'afficher des surfaces ; entrez la première formule dans un tableau de disons 20x20 cellules ; il y a donc 400 formules qui affichent la valeur zéro. Dans les cellules en périphérie, remplacez les formules par les valeurs zéro, puis quelque part vers le centre gauche entrez la valeur +1 et vers le centre droit la valeur -1. Affichez le graphe correspondant en choisissant le type surface et constatez : une belle surface minimale tendue sur un cadre rectangulaire horizontal, tirée vers le haut à gauche et vers le bas à droite. En complexifiant les « conditions aux limites » et avec un peu d'imagination, vous devriez pouvoir recréer la Sagrada Famillia ;-) Alan Turing avait bien raison !

03 première tentative de classification

Une première classification peut s'ébaucher à partir de ce que nous venons de survoler en classant les formes en trois familles en fonction de leur "dureté" : les formes solides, les formes flexibles et les formes élastiques.

031 les formes solides

Les formes solides sont telles que toute déformation détruit la forme ; une sphère dont on a modifié un point n'est plus une sphère, elle n'est plus le lieu de points à égale distance d'un centre et ne répond donc plus à sa définition première, ce n'est même plus une surface continue possédant un plan tangent en chaque point. Les coniques et la plupart des surfaces définies par des équations implicites ont le même comportement, et il est difficile en fait de les manipuler, de les modifier et de les combiner de manière continue à d'autres surfaces - le fuselage d'un avion construit à l'aide de ces formes n'est qu'une succession de raccords plus ou moins vifs avec tout un ensemble de conséquences désagréables sur les plans de l'aérodynamique (turbulences), de la résistance des matériaux (concentration d'efforts) et de l'aspect (discontinuité des reflets).

032 les formes flexibles

Les formes flexibles acceptent en revanche par le biais de points de contrôle une déformation partielle non destructrice ; nous n'avons abordé jusqu'ici que trois formes contrôlées par un ensemble réduit de points, le segment, le paraboloïde hyperbolique (PH) et la parabole. Quelles que soient les transformations que l'on fait subir aux trois points contrôlant une parabole, elle reste une parabole ; le PH passera de même du carré orthonormé bien plan à une forme élancée et gauche en selle de cheval jusqu'à devenir un triangle curviligne (le coté courbe est une parabole) bien connu des enfants jouant avec des ficelles sur deux files de pointes, et ceci sans perdre la moindre de ses propriétés. La Sagrada Familia de Gaudi à Barcelone est une composition complexe de paraboloïdes hyperboliques élancés et de paraboles s'approchant des formes organiques rêvées par l'architecte catalan. On verra de façon plus approfondie au chapitre 3 comment on peut construire un fuselage d'avion, depuis le nez conique jusqu'à l'empennage arrière en passant par la forme en poire au droit du cockpit, de façon continue et harmonieuse avec les avantages qu'on devine quant aux propriétés aérodynamiques, structurales et esthétiques.

033 les formes élastiques

Les formes élastiques se construisent à partir de contraintes aux limites en conservant un équilibre interne ; nous avons abordé les géodésiques, les surfaces minimales, l'équation de Laplace et pu apprécier la difficulté de traiter ce type de géométrie. Et pourtant, ce sont justement là les formes optimales de la nature, les formes qu'il faudrait être en mesure de comprendre au fond et de maîtriser vraiment, et c'est sur ces formes que nous disposons du minimum d'outils faciles à manier.

La présente étude privilégiera les formes flexibles, abordées de la façon la plus simple et unitaire possible afin d'éclairer les ponts qu'il est possible d'établir avec les surfaces solides d'un coté et les surfaces élastiques de l'autre. L'objectif est de tenter la description exacte ou approchée des formes solides et élastiques à partir des formes flexibles. Un arc de cercle redéfini à l'aide d'une courbe flexible pourra enfin se transformer en une belle poire ou en une came ou en une goutte d'eau, se gauchir dans l'espace, s'immerger dans une surface, une surface se tendant doucement vers l'équilibre d'une lame de savon.

Ces quelques rappels de géométrie étant faits, nous allons nous poser la question de savoir comment combiner entre eux des points de l'espace, et tout d'abord de savoir comment construire le milieu de deux points.

1 construction, définition

Au sommaire de cette section :

Ce chapitre présente une approche progressive de quelques formes gauches, met en évidence le caractère unitaire de formes apparemment éloignées les unes des autres - comme le cube et la ... cubique - et conduit à la définition générale des pFormes.

On se propose de construire à partir d'un ensemble de points une famille de formes (courbes, surfaces, volumes, hypervolumes, …) engendrées initialement par l'application récursive d'une opération simple :

« construire le milieu de deux points » .

Construisons le milieu de deux points en évitant la règle et le compas. Sur un sol bien plat par exemple, une méthode est de tendre une corde entre deux piquets, de tracer avec une craie le segment de droite passant par ces deux piquets, puis de ramener la seconde extrémité de la corde sur la première en suivant la trace laissée par la craie, repliant ainsi la corde en deux parts égales dont la nouvelle extrémité se trouve au milieu du segment.

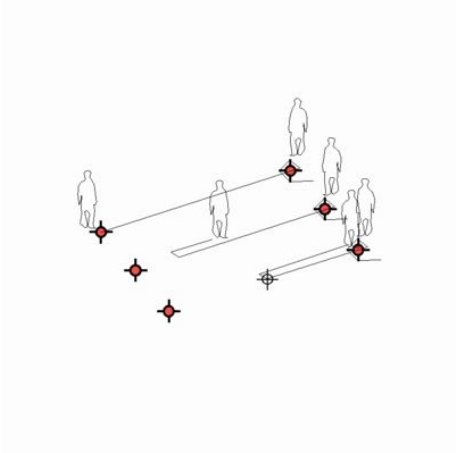


figure 1.1 : trouver le milieu de deux points, un geste élémentaire.

Sur une surface gauche le problème est un peu plus complexe, il existe une infinité de courbes reliant les deux piquets et l'on conviendra de tendre la corde en la disposant sur la surface suivant la courbe gauche unique, appelée *géodésique*, constituant le chemin le plus court entre ces deux points et le long duquel la normale à la corde (dans le plan osculateur) est confondue avec la normale à la surface. Dans les deux cas on fait l'hypothèse que la corde garde une longueur constante et est infiniment souple. En évitant précédemment la construction classique du milieu par la règle et le compas, ce principe de construction du point milieu reste valable quelle que soit la surface, et ceci sera le point de départ d'une généralisation des formes étudiées dans l'espace euclidien au cas de formes appartenant à des espaces courbes, à travers l'introduction du concept de segment immergé.

Mais comment écrire le milieu de deux points, et au-delà toute combinaison de points ?

L'espace de référence est l'espace "courant" peuplé de points que l'on souhaite pouvoir combiner suivant une expression linéaire du type :

$$\sum_i \; ki.pi, \text{ avec } i = [0,n-1]$$

Combiner linéairement des vecteurs ne pose pas de problème, ... dans un espace vectoriel ; mais l'espace des points (appelé espace affine) ne possède pas toutes les propriétés d'un espace vectoriel, même s'il lui ressemble et notamment, toutes les combinaisons linéaires de points ne sont pas valables. On peut par exemple définir la somme de deux vecteurs de façon indépendante du repère, mais on ne peut pas le faire pour deux points ; en revanche, la demi–somme de deux points produit un point invariant dans un changement de repère et constitue donc une combinaison linéaire pondérée valable.

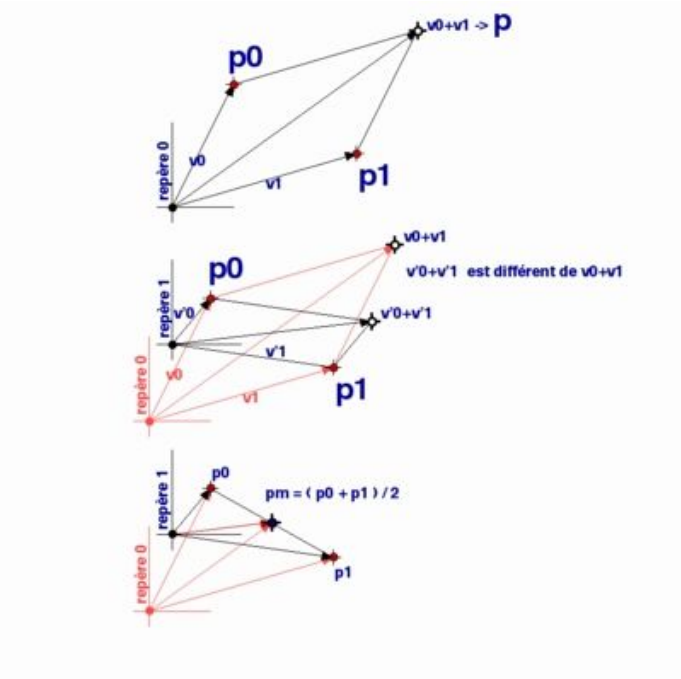


figure 1.2 : la somme de deux points ne produit pas un point invariant, la demi-somme oui.

De manière générale, on montre que toute combinaison linéaire de points est valable si elle satisfait à la condition :

$$\sum_i \; ki = 1, \text{ avec } i = [0,n-1]$$

Seront notamment correctes des expressions comme :

```
p = (p0 + p1)/2
p = 2.p0 – p1
p = (2.p0 + p1)/3
p = (p0 + 3.p1 + 3.p2 + p3)/8
```

qui représentent respectivement le milieu de deux points, le point symétrique de p1 par rapport à p0, le point au tiers du segment p0p1 et le milieu d'une cubique définie par les quatre points (p0,p1,p2,p3). On aura remarqué que tout ceci n'est qu'une autre expression des théorèmes bien connus sur les barycentres...

Ceci ayant été précisé, nous pouvons construire une première famille de 'formes multilinéaires récursives' construites avec un couple d'opérateurs MI() et MIR(), l'étendre par application d'un opérateur de « diagonalisation », DIAG(), et enfin proposer la définition fondamentale des formes pascaliennes.

11 formes multilinéaires récursives

Au sommaire de cette section :

L'application d'un couple d'opérateurs MI() et MIR() à un ensemble de points va nous permettre de générer naturellement les premières formes linéaires de la géométrie: des segments de droite, des facettes (gauches), des cubes (gauches), des hypercubes (gauches), et au-delà si nécessaire.

111 un point de R4

Les combinaisons linéaires envisagées plus haut sont exprimables dans un espace affine de dimension quelconque Rn. Pour des raisons qui seront données un peu plus loin, nous opèrerons sur des points toujours définis dans R4, en utilisant la forme dite des coordonnées homogènes ou forme projective : <x,y,z,t>, associant au point de R4 le point de R3 défini par <x/t,y/t,z/t>. Dans un premier temps, on prendra par défaut t=1 et on oubliera qu'on travaille dans l'espace R4.

Dans la syntaxe POVRAY/pFlibs, on définira donc un point courant (x,y,z) en déclarant une variable locale P et lui assignant 3 coordonnées x, y et z entre crochets '<' et '>':

```
#local P = <x,y,z,1>;
```

Les appels de macro:

```
draw( 0, <0.0,0.0,0.5,1.0>, STANDARD )
draw( 0, <0.0,0.0,0.0,1.0>, point(0.1)  + ma_couleur(<1,1,0,0.5>))
draw( 0, <0.0,0.0,-0.5,1.0>, point(0.1) + ma_texture(GOLD) )
draw( 0, <0.0,0.0,0.0,1.0>, point(0.1)  + ma_texture(MIROIR) )
draw( 0, <-0.5,0.0,0.0,1.0>, point(0.1) + ma_texture(GRANIT) )
draw( 0, <0.5,0.0,0.0,1.0>, point(0.1)  + ma_texture(MARBRE) )
```

dessineront 6 points respectivement sous la forme standard d'une sphère rouge de rayon 0.05, et sous les formes de sphères dorée, chromée, en granit, et plexiglas translucide strié de bandes noires de rayon 0.2.

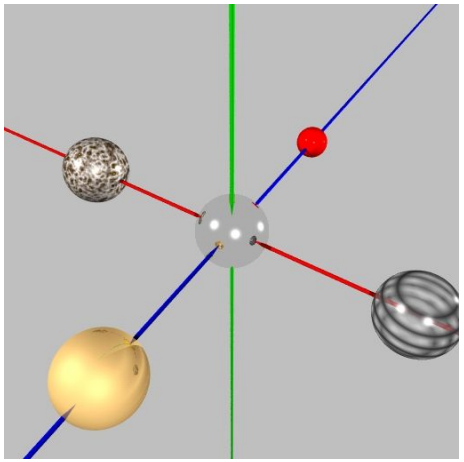


figure 111 : les 3 axes Ox, Oy, Oz (rouge, vert, bleu) et quelques points.

112 un segment de droite

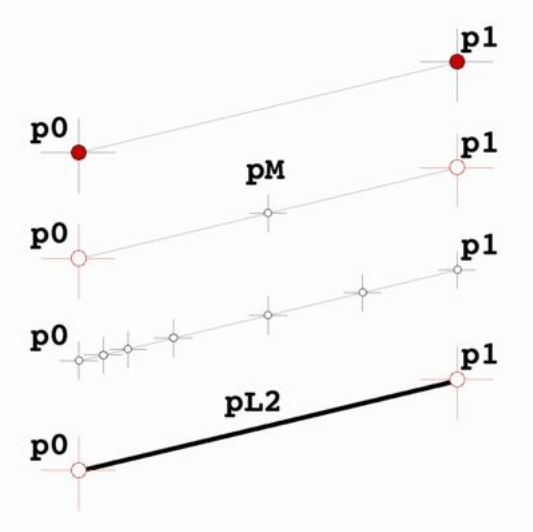


figure 112.1 : du couple de points au segment (pL2).

Appelons MI() l'opérateur retournant le point milieu de deux points p0 et p1 de l'espace :

```
pm = MI(p0,p1) = (p0+p1)/2
```

Noter que la somme des coefficients est: 1/2 + 1/2 = 1. Ce troisième point pm donne naissance à deux couples de points (p0,pm) et (pm,p1), et à l'irrésistible envie de recommencer en direction de p0 et de p1 :

```
p0m  = MI(p0,pm)     pm1  = MI(pm,p1)
p00m = MI(p0,p0m)    pmm1 = MI(pm,pm1)
p0mm = MI(p0m,pm)    pm11 = MI(pm1,p1)
etc…                 etc…
```

L'application récursive de l'opérateur MI() produit un ensemble infini et "dense" de points situés entre les deux points p0 et p1, un "segment de droite" que nous désignerons par pL2 (les raisons de cette notation seront données plus loin dans le texte) ; convenant d'appeler MIR() l'opérateur récursif ainsi défini, nous écrirons :

```
pL2 = MIR(p0,p1)
```

Cet ensemble de points (pL2) n'est pas à proprement parler un segment de droite tel qu'on le définit habituellement en géométrie. Si le processus récursif amène bien à un ensemble de points dont le nombre est aussi grand qu'on le souhaite, avec une distance entre chaque point tendant bien vers zéro, l'ensemble obtenu n'est pas continu et n'est pas "réellement" un segment de droite. On définit habituellement dans Rn un segment de droite reliant un couple de points (p0, p1) comme l'ensemble des points p satisfaisant à l'application linéaire bijective :

```
p(t) = (1-t)*p0 + t*p1,
avec t dans l'intervalle [0,1] de l'ensemble des nombres réels
(R).
```

Cet ensemble de points est infini et continu (comme R), et des points comme p(1/3) ou p(k) avec k = racine(2)/2, peuvent bien être atteints sur ce segment de droite ; ce qui n'est pas le cas de pL2 qui est en relation bijective avec l'ensemble des partitions de l'unité (1/2, 1/4, 1/8,...). On dit d'un tel ensemble qu'il est dense, et c'est bien cette propriété de densité qui nous intéresse, notamment parce qu'elle nous permettra de retrouver le concept indispensable de tangente. Nous y reviendrons plus longuement dans la section 2, et pour la suite nous conviendrons d'assimiler notre pL2 à un "vrai" segment de droite et les formes dérivées (surfaces, volumes,...) aux formes continues classiques de la géométrie.

Dans la syntaxe de POVRAY/pFlibs, nous définissons et dessinons un tableau de deux points ainsi :

```
#local pL2 = array[2] { p0, p1 } // tableau de deux points
 draw( 1, pL2, STANDARD )        // 1 est la dimension du segment
```

Noter que pL2 EST tout simplement CE tableau de deux points. L'appel de la macro :

```
#local pL2_subdivisee = pFsubdivision( 1, pL2, 4 )
```

implémente l'opérateur MIR(), en appliquant 4 fois l'opérateur MI() au tableau de points pL2 pour produire une série de 2^4+1 = 17 points qui seront dessinés par l'appel suivant :

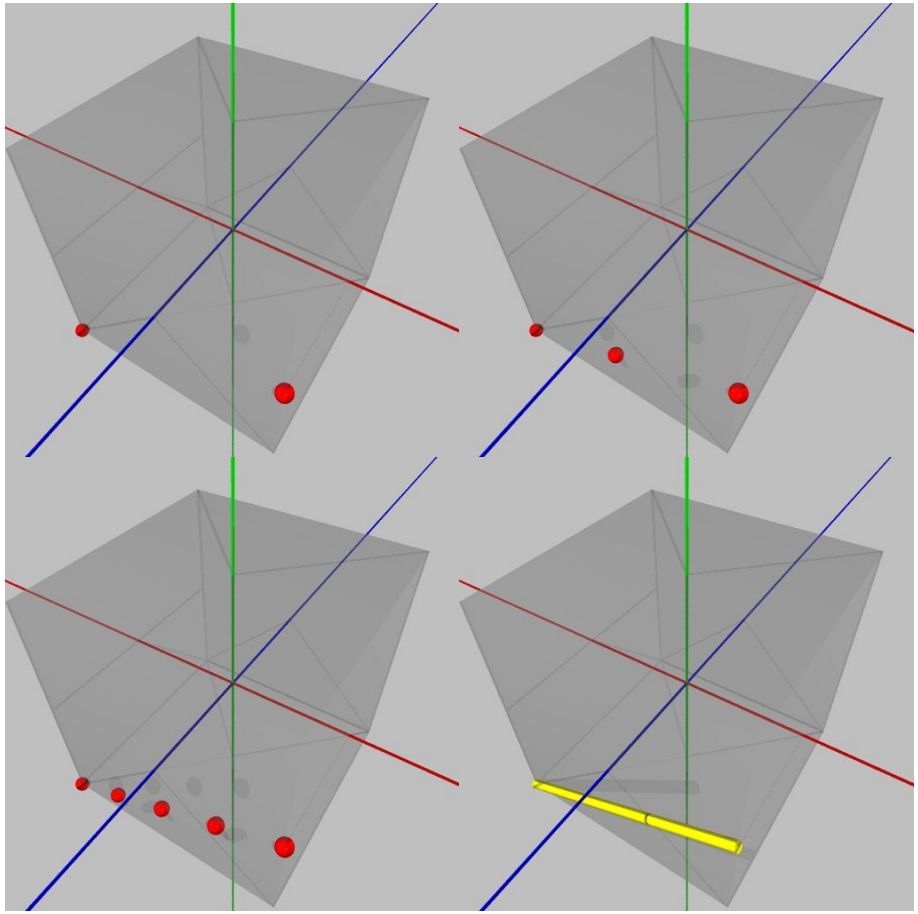
```
draw( 1, pL2_subdivisee, STANDARD )   // 17 petites sphères rouges
```

On peut plus directement incorporer la subdivision dans l'appel du dessin en insérant l'option finesse() :

```
draw( 1, pL2, finesse(4) ) // 17 petites sphères rouges
```

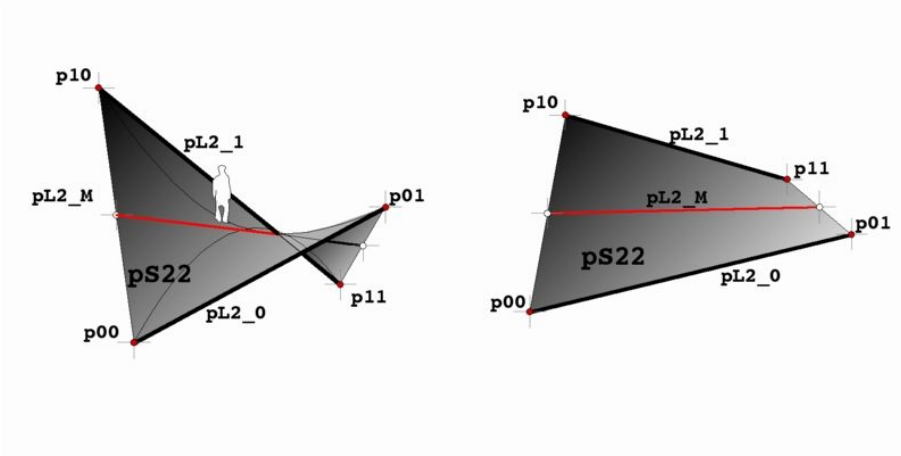
et même remplacer les sphères par des cylindres pour obtenir un segment jaune de rayon 0.02 en écrivant :

```
draw( 1, pL2, finesse(4) + courbe( 0.02 ) + couleur(<1,1,0>) )
```



figures 112.2 à 112.5 : un segment (pL2) construit sur deux points.

113 une facette gauche



figures 113.1 et 113.2 : deux représentations d'une facette gauche (pS22).

Soient deux segments pL2_0 et pL2_1 construits sur les points (p00,p01) et (p10,p11). Considérant le segment construit sur les milieux respectifs des extrémités des segments pL2_0 et pL2_1 :

```
pL2_M = MIR( MI(p00,p10), MI(p01,p11) )
```

et jouant sur la commutativité des opérateurs linéaires MI() et MIR(), nous sommes amenés à étendre la définition de l'opérateur MI() ainsi :

```
pL2_M  = MI( MIR(p00,p10), MIR(p01,p11) )
       = MI( pL2_0, pL2_1 )
```

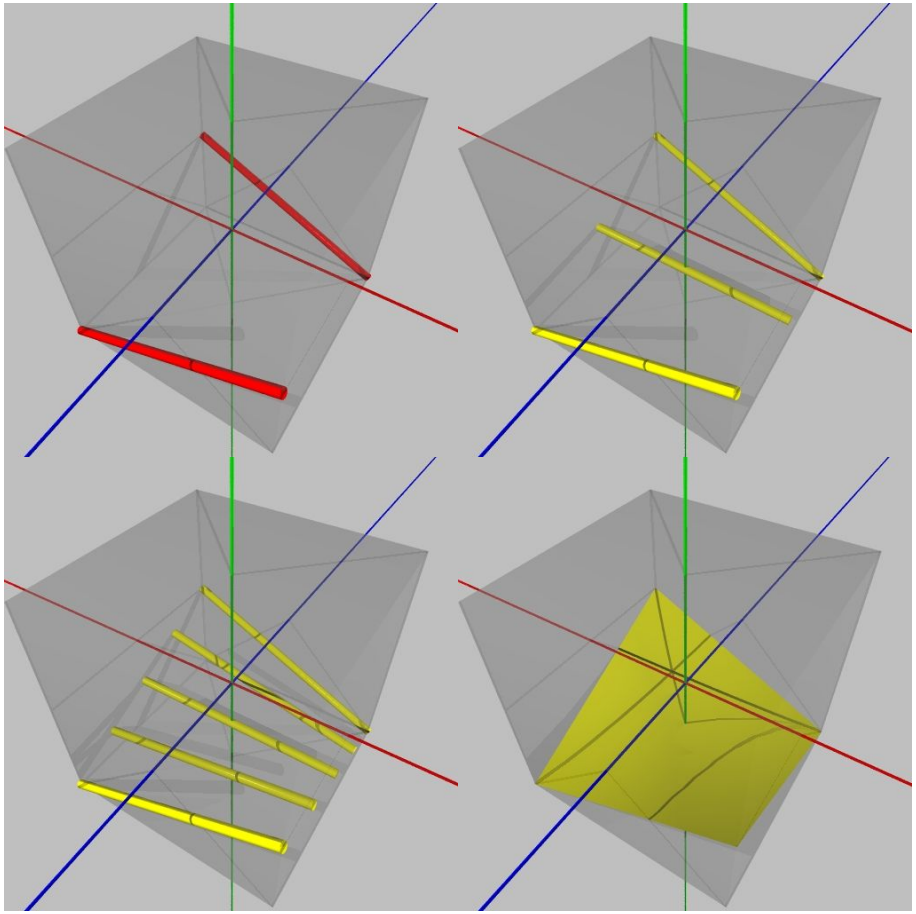
L'application récursive de l'opérateur MI() aux couples de segments à gauche (pL2_0,pL2_M) et à droite (pL2_M,pL2_1) produit un ensemble infini et dense de segments formant une portion de surface, une facette gauche que nous désignerons par pS22 ; étendant l'application de l'opérateur MIR() au cas de deux segments, nous écrirons :

```
pS22 = MIR( pL2_0, pL2_1 )
```

Dans la syntaxe de POVRAY/pFlibs, on écrit:

```
// création d'un tableau de deux segments :
#local pS22 = array[2] { pL2_0, pL2_1 }
```

```
// dessin de 2*2 = 4 petites sphères rouges :
 draw( 2, pS22, STANDARD )   // 2 est la dimension de la surface
// récursion de niveau 2 dans les deux dimensions :
#local spS22 = pFsubdivision( 2, pS22, <2,2> )
// dessin de (4+1)*(4+1) = 25 petites sphères rouges :
 draw( 2, spS22, STANDARD )
// variante : récursion 2 dans les deux dimensions
// et dessin d'une surface lisse jaune :
 draw( 2, pS22, finesse(<2,2>)+surface(LISSE)+couleur(<1,1,0>) )
```



figures 113.3 à 113.7 : facette gauche (pS22) construite sur deux segments (pL2).

114 un cube gauche

Poursuivant la progression et redéfinissant l'opérateur MI() pour produire la facette milieu de deux facettes gauches (pS22_0 et pS22_1) et l'opérateur MIR() pour son application récursive, nous pouvons construire un ensemble infini et dense de facettes, un mille–feuilles gauche, un volume que nous appelerons 'cube gauche' noté pV222.

```
pS22_M = MI( pS22_0, pS22_1 )
pV222  = MIR( pS22_0, pS22_1 )
```

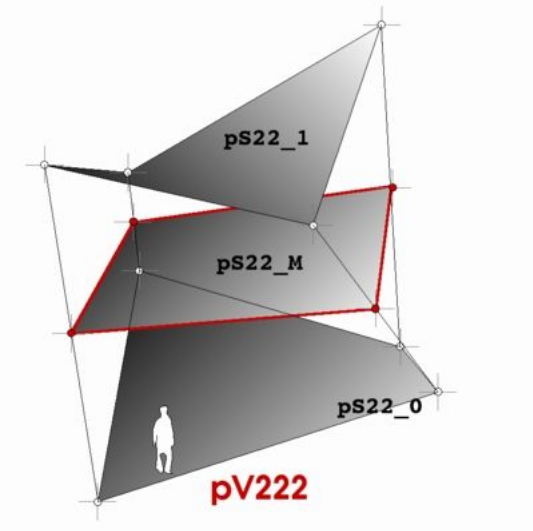
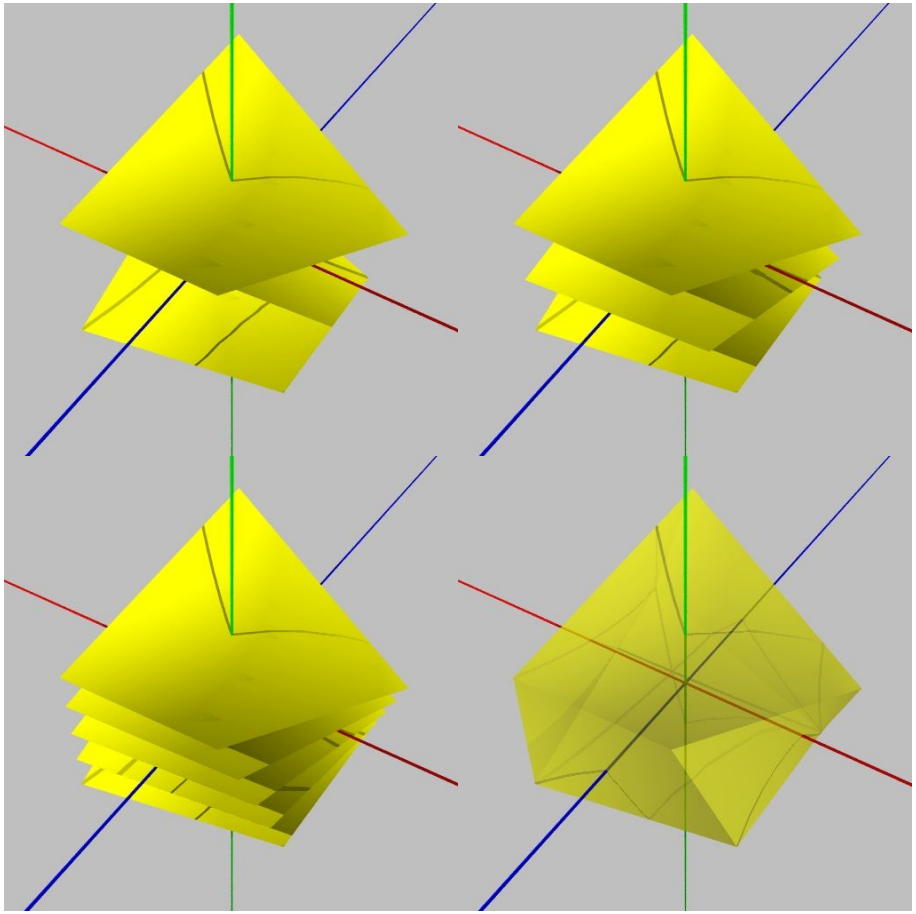


figure 114.1 : facette milieu de deux facettes gauches (pS22).

Dans la syntaxe de POVRAY/pFlibs, on écrit :

```
// création d'un tableau de deux surfaces :
#local pV222 = array[2] { pS22_0, pS22_1 }
// dessin de 2*2*2=8 petites sphères rouges :
 draw( 3, pV222, STANDARD )
// récursion 2 dans les trois dimensions
// et dessin des 6 facettes du cube gauche :
 draw( 3, pV222, finesse(<2,2,2>)+enveloppe(LISSE)+couleur
(<0,1,1>))
```

pour construire et dessiner les huit sommets d'un cube gauche et son enveloppe cyan.



figures 114.2 à 114.5 : cube gauche (pV222) construit sur deux facette gauches (pS22).

De même qu'une facette gauche peut être représentée sous plusieurs formes (matrice carrée de points, distribution de segments ou surface continue), de même un cube gauche peut être représenté sous plusieurs aspects : mille-feuilles de facettes gauches, matrice tridimensionnelle de points, faisceau de segments, ou le plus souvent sous la forme d'une enveloppe constituée par ses six facettes gauches. Dans tous les cas, un cube gauche est avant tout un tableau tridimensionnel de

points, un véritable objet volumique dans lequel il sera possible de tailler et d'extraire d'autres objets (surface, courbes, points).

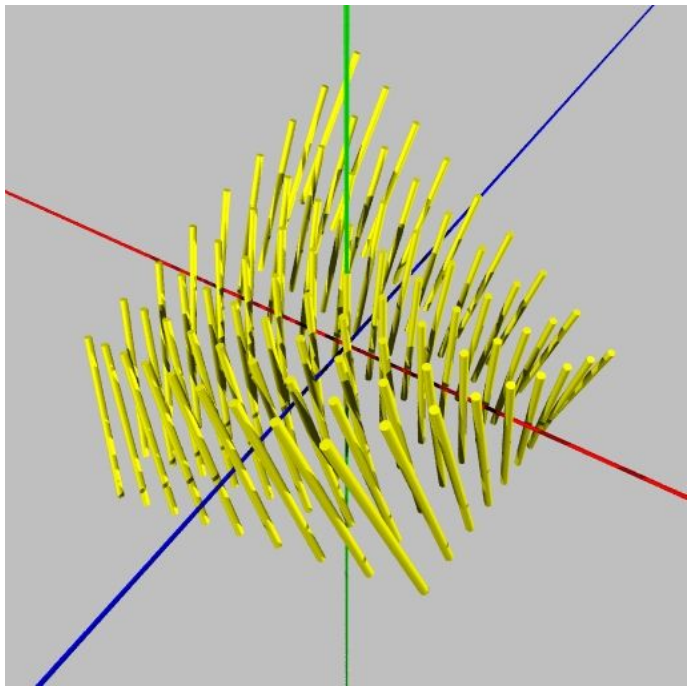


figure 114.6 : représentation fibrée d'un cube gauche.

115 un hypercube gauche

Enfin, en redéfinissant les opérateurs MI() pour produire le cube gauche milieu de deux cubes gauches (pV222_0 et pV222_1) et MIR() pour son application récursive, nous pouvons construire un hypercube gauche.

```
pV222_M = MI( pV222_0, pV222_1 )
pH2222  = MIR( pV222_0, pV222_1 )
```

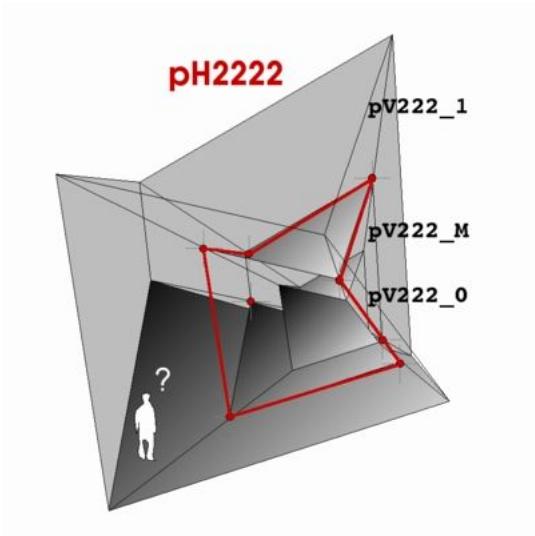


figure 115.1 : cube milieu de deux cubes gauches (pV222).

Dans la syntaxe de POVRAY/pFlibs, on écrit :

```
// création d'un tableau de deux volumes :
#local pH2222 = array[2] { pV222_0, pV222_1 }
// dessin de 2*2*2*2 = 16 petites sphères rouges:
 draw( 4, pH2222, STANDARD )
// dessin de (2*2+1)^4 = 625 petites sphères jaunes :
 draw( 4, pH2222,finesse(<2,2,2,2>)+point(0.02)+couleur(<1,1,0>) )
```

pour construire et dessiner les seize sommets rouges de l'hypercube et un remplissage par de petites sphères jaunes, en attendant une autre représentation plus parlante que l'image 115.2 ci-dessous ...

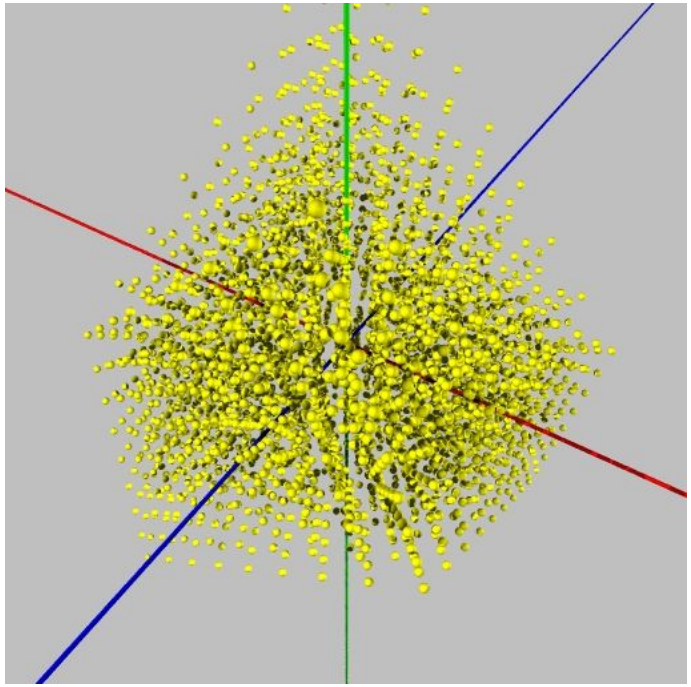


figure 115.2 : un hypercube.

116 première généralisation

Le couple d'opérateurs MI() et MIR() produit donc une famille de formes linéaires obtenues par un processus récursif. De façon générale, nous noterons que l'opérateur MI() produit une forme de même dimension et que l'opérateur MIR() produit une forme de dimension supérieure. Ces formes sont gauches dans le cas général (à l'exception du segment bien sûr), la facette gauche est une portion quadrangulaire du classique paraboloïde hyperbolique, une surface réglée à double courbure négative, et les six faces du cube gauche sont des facettes gauches.

Ces formes sont '*pleines*', chaque point d'une forme peut être adressé : un cube gauche est un volume rempli de points et non une simple enveloppe vide, et il en est de même pour l'hypercube. On a bien ainsi défini des formes cohérentes dans lesquelles il va être possible de travailler et dont on pourra par exemple extraire des '*sous–formes*'. Une opération inverse, la diagonalisation, produisant une forme de dimension inférieure mais plus complexe qu'un segment linéaire nous conduira à de nouvelles extensions des opérateurs MI() et MIR().

12 diagonalisation

Au sommaire de cette section :

Toutes les formes précédentes sont linéaires plusieurs fois, elles sont engendrées par des familles de droites, mais elles n'en contiennent pas moins de «vraies» courbes présentant un certain intérêt. La facette gauche, par exemple, est une surface en forme de selle de cheval présentant une certaine courbure dès que les deux segments générateurs ne sont plus coplanaires. En repliant une facette gauche sur elle–même jusqu'à confondre deux sommets opposés, on construit une sorte de triangle dont un côté est un arc de parabole. Ce *'pliage'* de la facette gauche nous permet de construire une courbe, une forme de dimension inférieure à celle de la facette gauche, mais de complexité supérieure au segment de droite.

121 la facette gauche et sa parabole

Soit une facette gauche construite sur deux segments (pL2_0,pL2_1) définis sur deux couples de points (p00,p01) et (p10,p11) :

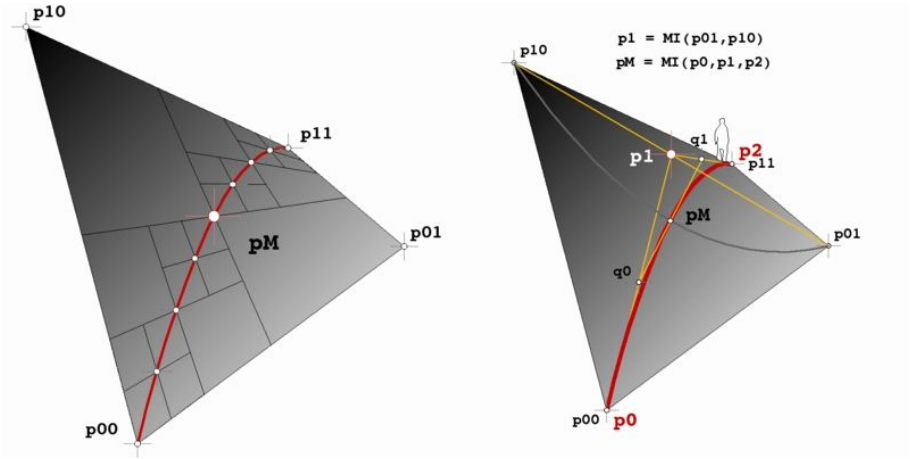
```
pS22 = MIR( pL2_0, pL2_1 )
     = MIR( MIR( p00, p01 ), MIR( p10, p11 ) ),
```

Le point milieu de la facette peut se définir par :

```
pm = MI( MI( p00, p01 ), MI( p10, p11 ) )
   = ( (pOO + p01)/2 + (p10 +p11)/2 )/2
   = ( pOO + p01 + p10 + p11 )/4
```

Ce point détermine quatre sous–facettes (p00,pm), (p01,pm), (p10,pm) et (p11,pm). En recommençant ainsi récursivement l'opération sur les deux sous–facettes en diagonale contruites sur les couples (p00,pm) et (pm,p11), on engendre un ensemble infini et dense de points, une courbe reliant en diagonale les points p00 et p11 de la facette gauche. Appelons DIAG() l'opérateur créant dans la facette cette ligne diagonale.

```
diagonale = DIAG( pS22 )
```



figures 121.1 et 121.2 : deux visions de la diagonale d'une facette gauche.

Cette diagonalisation produit une courbe de dimension inférieure à celle de la facette et l'on est conduit à rechercher un ensemble réduit de points de "contrôle" extraits des quatre points

générateurs de la facette.

En définissant les trois points :

```
p0 =   p00,
p1 = ( p01+p10 )/2,
p2 =       p11,
```

on peut réécrire le point milieu sous une forme qui nous amène à considérer un opérateur MI() applicable à trois points :

```
pm = ( p0 + 2*p1 + p2 )/4
   = MI( p0, p1, p2 )
```

Ce nouvel opérateur consiste en fait en un application récursive en deux temps de l'opérateur MI() initial conduisant au point pm à partir du triplet (p0,p1,p2), à l'aide de deux points milieux intermédiaires q0 et q1:

```
1) q0 = MI(p0,p1),
   q1 = MI(p1,p2),
2)  pm = MI(q0,q1)
```

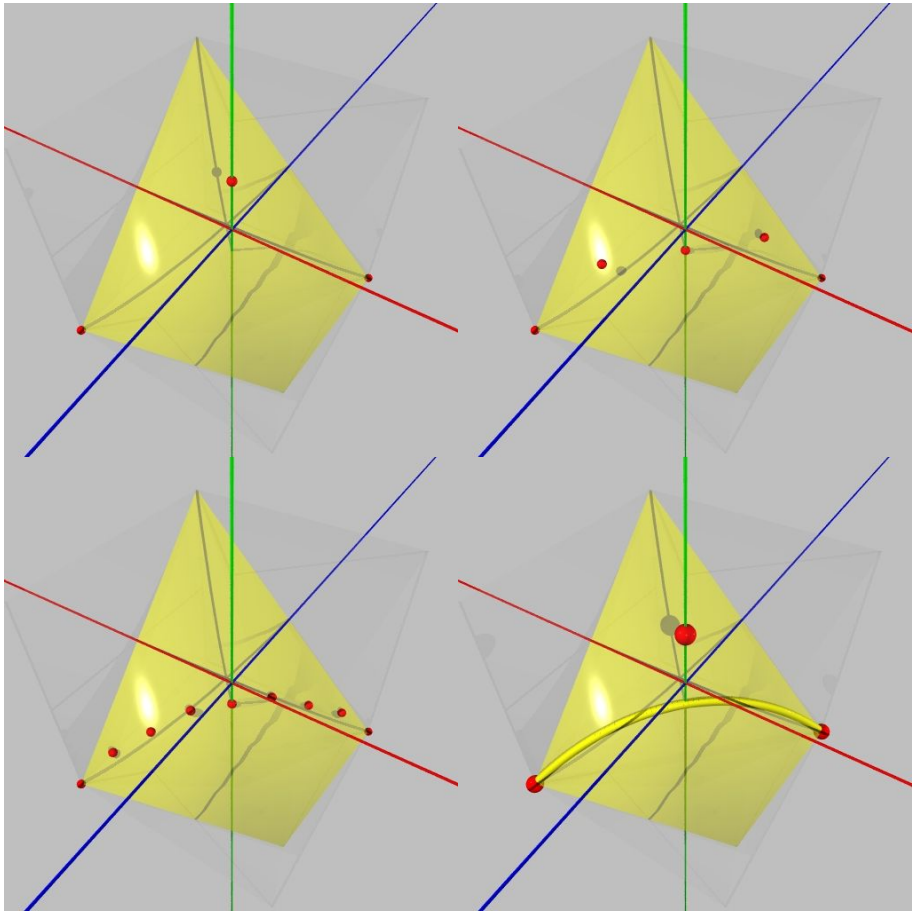
Nous pouvons donc le considérer comme une extension valable de l'opérateur MI().

De plus, l'application récursive de cet opérateur aux triplets de points (p0,q0,pm) et (pm,q1,p1) reproduit la courbe diagonale, que nous conviendrons maintenant de désigner par pL3, et étendant l'application de MIR() au cas de trois points, nous écrirons :

```
pL3 = DIAG( pS22 )
    = MIR(p0,p1,p2)
```

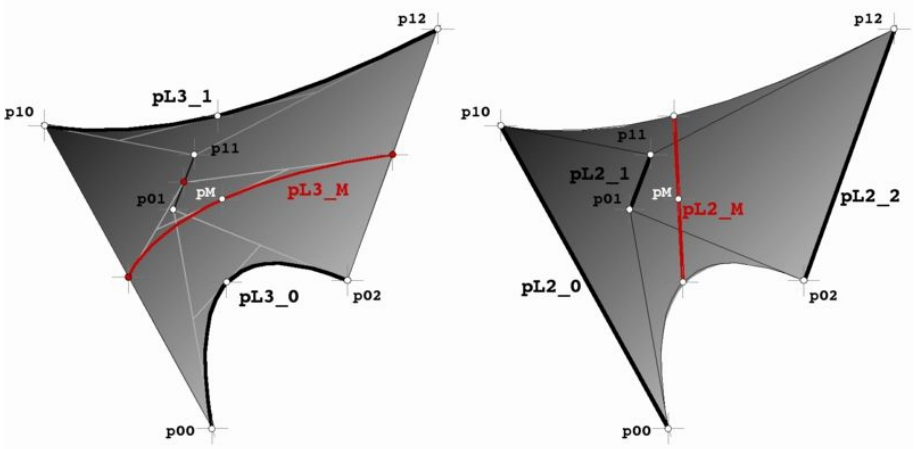
Nous retrouvons dans cette construction l'algorithme de base proposé par de Casteljau pour le cas d'une courbe (parabole) définie par trois points ; il se trouve logiquement relié au processus de génération de la famille des formes multilinéaires récursives, par une sorte de contraction/pliage/diagonalisation d'une facette gauche (une portion de paraboloïde hyperbolique). Cette reformulation par application de l'opérateur DIAG() ou des opérateurs étendus MI() et MIR() nous permet de quitter le monde purement rectiligne des formes multilinéaires récursives et d'aborder vraiment celui des formes gauches. Dans la syntaxe de POVRAY/pFlibs, on écrit :

```
// définition d'une facette gauche et affichage :
#local pS22 = array[2] { pL2_0, pL2_1 }
draw( 2, pS22, finesse()+surface(LISSE)+ma_couleur(<1,1,1>) )
// extraction des points de contrôle de la diagonale :
#local p0 =   pS22[0][0];
#local p1 = ( pS22[0][1]+pS22[1][0] ) / 2;
#local p2 =             pS22[1][1];
// définition de la parabole et affichage :
#local pL3 = array[3] { p0, p1, p2 }
draw( 1, pL3, STANDARD ) // points de contrôle
draw( 1, pL3, finesse(5)+courbe(0.02)+ ma_couleur(<1,1,0>) )
```



figures 121.3 à 121.7 : parabole (pL3) construite sur 3 points, diagonale d'une facette gauche (pS22).

122 le paraboloïde réglé et sa cubique



figures 122.1 et 122.2 : construction du paraboloïde réglé (pS32) à partir de deux paraboles (pL3), et à partir de trois segments (pL2).

Nous savons construire une parabole. Soient deux paraboles pL3_0 et pL3_1 construites sur les points (p00,p01,p02) et (p10,p11,p12). Considérant la parabole construite sur les milieux respectifs des points définissant les deux paraboles :

```
pL3_M = MIR( MI(p00,p10), MI(p01,p11), MI(p02,p12) )
```

nous sommes amenés à étendre la définition de l'opérateur MI() ainsi :

```
pL3_M = MI(pL3_0,pL3_1)
```

L'application récursive de l'opérateur MI() aux couples de paraboles (pL3_0,pL3_M) et (pL3_M,pL3_1) produit un ensemble infini et dense de paraboles, un paraboloïde réglé, que nous désignerons par pS32 ; étendant l'application de MIR() au cas de deux paraboles, nous écrirons :

```
pS32 = MIR(pL2_0,pL2_1)    // pS32 ou pS23
```

Notons que cette surface peut être aussi obtenue à partir des trois segments construits sur (p00,p10), (p01,p11), (p02,p12), ce qui conduit à l'extension des opérateurs MI() et MIR() au cas de trois segments :

```
pL2_M = MI(pL2_0,pL2_1,pL2_2),
pS32  = MIR(pL2_0,pL2_1,pL2_2)
```

De ce dernier point de vue et à notre connaissance, nous pouvons considérer qu'il s'agit d'une extension inédite de l'algorithme de de Casteljau appliqué à trois segments, et non plus à trois points comme il est habituellement présenté, ... du moins dans le cas de trois points.

Le raisonnement permettant de construire la diagonale d'une facette gauche s'applique de façon analogue au paraboloïde réglé pour produire un point milieu défini par quatre points :

```
p0 = p00,
p1 = (2*p01 + p10)/3,
p2 = (p02 + 2*p11)/3,
p3 = p12,

pm = (p0 + 3*p1 + 3*p2 + p3)/8
   = MI(p0,p1,p2,p3)
```

et une courbe diagonale (cubique) :

```
pL4 = DIAG( pS32 )
    = MIR(p0, p1, p2, p3)
```

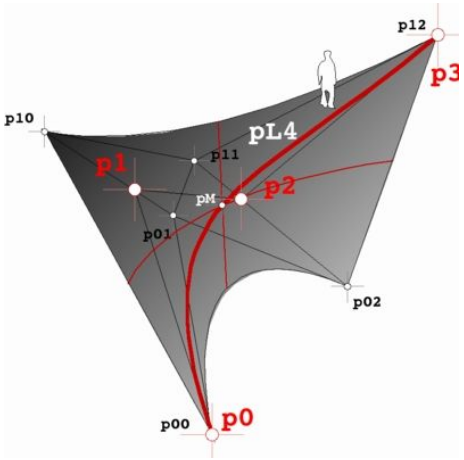


figure 122.3 : le paraboloïde réglé (pS32) et sa diagonale (pL4) construite sur 4 points.

Dans la syntaxe de POVRAY/pFlibs, on écrit :

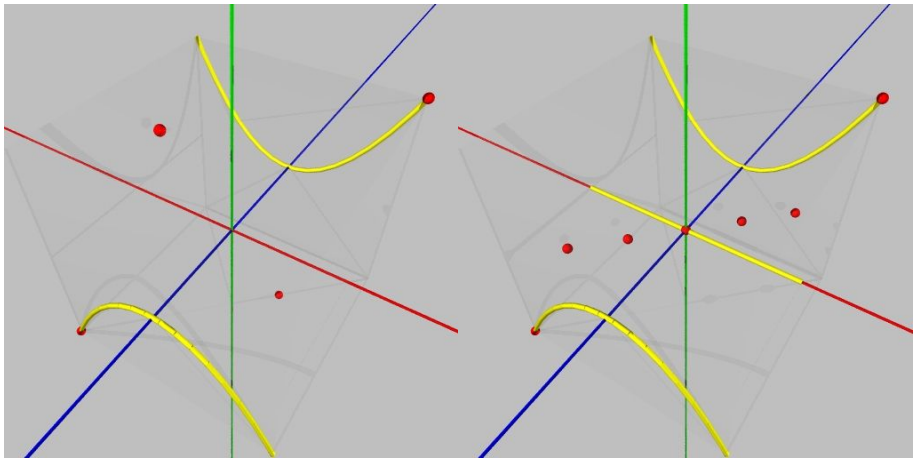
```
// définition d'un paraboloïde réglé à partir
// de 3 segments ou de 2 paraboles, et affichage :
#local pS32 = array[3] { pL2_0, pL2_1, pL2_2 }
```

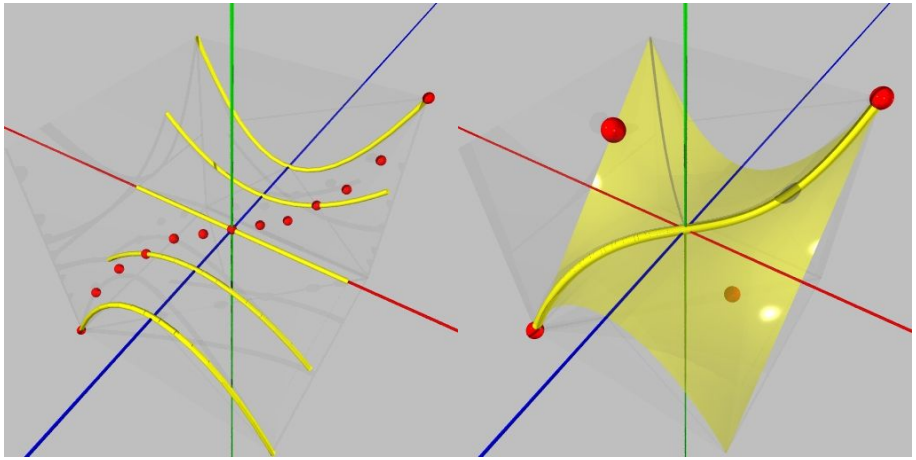
```
           = array[2] { pL3_0, pL3_1 }
draw( 2, pS32, finesse() + surface(LISSE) + ma_couleur() )
// extraction des points de contrôle de la diagonale :
#local p0   =    pS32[0][0];
#local p1   = (2*pS32[0][1]+  pS32[1][0]) / 3;
#local p2   = (  pS32[0][2]+2*pS32[1][1]) / 3;
#local p3   =                 pS32[1][2];
// définition de la cubique et affichage :
#local pL4  = array[4] { p0, p1, p2, p3 }
draw( 1, pL4, STANDARD ) // points de contrôle
draw( 1, pL4, finesse(5)+courbe(0.02)+ma_couleur(<1,1,0>) )
```

Remarque 1 : la cubique est souvent utilisée en infographie pour représenter de façon approchée un arc de cercle sur 90°, -cf figure 122.8- ; nous verrons dans la partie 31 comment on peut atteindre une représentation exacte du cercle.

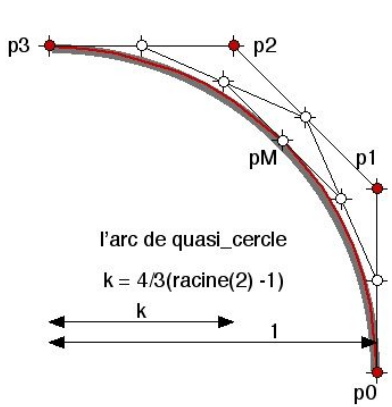
Remarque 2 : la cubique est la première courbe gauche rencontrée à ce point ; nous en verrons l'utilité dans la partie 34 sur les Concaténations. La figure 122.9 illustre l'occupation dans l'espace d'une cubique reliant deux points opposés d'un cube.

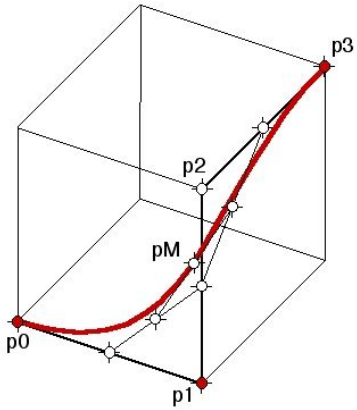
Remarque 3 : La figure 122.10 représente la cubique diagonale d'un paraboloïde construit sur une grille orthonormée de ses points de contrôle ; nous dirons qu'il s'agit d'une représentation orthonormée de la surface et de sa diagonale. On visualise bien ainsi les positions au 1/3 et au 2/3 des points p1 et p2.





figures 122.4 à 122.7 : progression vers le paraboloïde réglé (pS32) et sa diagonale (pL4).





figures 122.8 et 122.9 : cubique plane approximant un arc de cercle, et cubique dans l'espace.

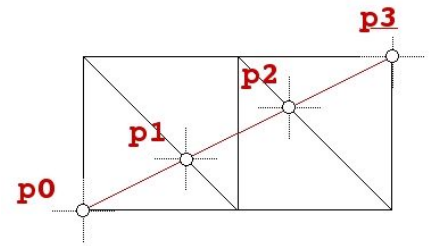


figure 122.10 : représentation orthonormée d'une facette gauche (pS32) et du positionnement des 4 points de contrôle de sa diagonale (pL4) .

123 le cube gauche et ses diagonales

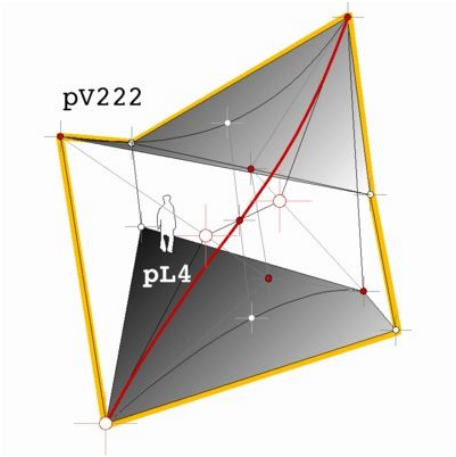


figure 123.1 : le cube gauche (pV222) et ses diagonales, le paraboloïde réglé (pS32) et la cubique (pL4).

Le paraboloïde réglé (pS32) construit sur les paraboles diagonales des facettes gauches génératrices d'un cube, peut être considéré comme surface diagonale du cube gauche ; nous avons vu qu'une cubique (pL4) pouvait être considérée comme diagonale d'un paraboloïde réglé (pS32), et donc comme diagonale "au carré" du cube :

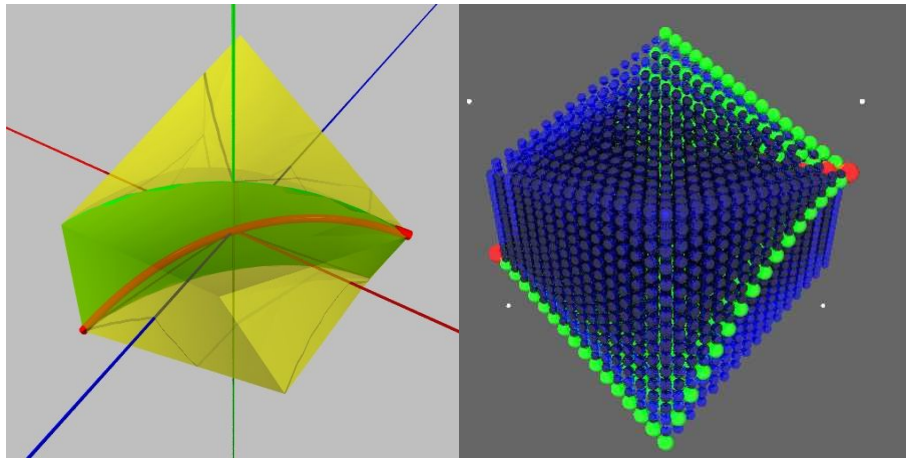
```
DIAG( pV222 )   -> pS32
pL4 = DIAG( pS32 )
    = DIAG( DIAG( pV222 ) )
    = DIAG2( pV222 )
```

De la même façon, la troisième diagonalisation d'un hypercube gauche pH2222 conduirait à une courbe contrôlée par 5 points, une pL5 :

```
DIAG( pH2222 )  -> pV322 // un cube contrôlé par 3 facettes
DIAG( pV322 )   -> pS33  // une surface contrôlée par 3 quads
pL5 = DIAG( pS33 )       // une courbe contrôlée par 5 points
    = DIAG( DIAG( pV322 ) )
    = DIAG( DIAG( DIAG( pH2222 ) ) )
    = DIAG3( pH2222 )
```

Dans la syntaxe de POVRAY/pFlibs, on écrit dans la cas du cube :

```
// définition d'un cube gauche et affichage :
#local pV222 = array[2] { pS22_0, pS22_1 }
draw( 3, pV222, finesse()+enveloppe(LISSE)+ma_couleur(<1,1,1>) )
// surface diagonale du pV222 et affichage :
#local pS32 = pFdiagonalisation( 3, pV222 )
draw( 2, pS32, finesse()+surface(LISSE)+ma_couleur(<1,1,0>) )
// courbe diagonale du pS32 et affichage :
#local pL4 = pFdiagonalisation( 2, pS32 )
draw( 1, pL4, STANDARD ) // points de contrôle
draw( 1, pL4, finesse(5)+courbe(0.02)+ ma_couleur(<1,0,0>) )
```



figures 123.2 : deux représentations du cube gauche et de ses diagonales.

Ces nouvelles formes, notamment la surface pS33 (qu'on peut convenir d'appeler biquadrique par analogie avec le carreau de Bézier bicubique contrôlé par 4 cubiques et appelé bicubique) et la courbe pL5 proviennent de diagonalisations d'un hypercube gauche pH2222 ; mais elles peuvent plus facilement être construites à l'aide des opérateurs MI() et MIR().

124 la biquadrique et sa diagonale

Deux paraboles nous ont permis de créer un paraboloïde réglé, trois paraboles conduisent à une biquadrique définie ainsi :

```
pS33 = MIR( pL3_0,pL3_1,pL3_2 )
     = MIR( MIR(p00,p01,p02),MIR(p10,p11,p12),MIR(p20,p21,p22) )
```

dont le point milieu peut s'écrire :

```
pm = MI( MI(p00,p01,p02), MI(p10,p11,p12), MI(p20,p21,p22) )
   = ( (p00+2p01+p02)/4+2(p10+2p11+p12)/4+(p20+2p21+p22)/4 )/4
   = ( p00+4(p01+p10)/2+6(p02+4p11+p20)/6+4(p12+p21)/2+p22 )/16
   = ( q0 + 4q1 + 6q2 + 4q3 + q4 )/16
   = MI( q0,q1,q2,q3,q4 )

avec : q0 = p00
       q1 = (p01+p10)/2
       q2 = (p02+4p11+p20)/6
       q3 = (p12+p21)/2
       q4 = p22
```

conduisant à la construction d'une courbe diagonale définie par 5 points :

```
pL5 = MIR( q0,q1,q2,q3,q4 )
```

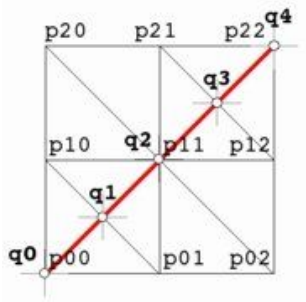
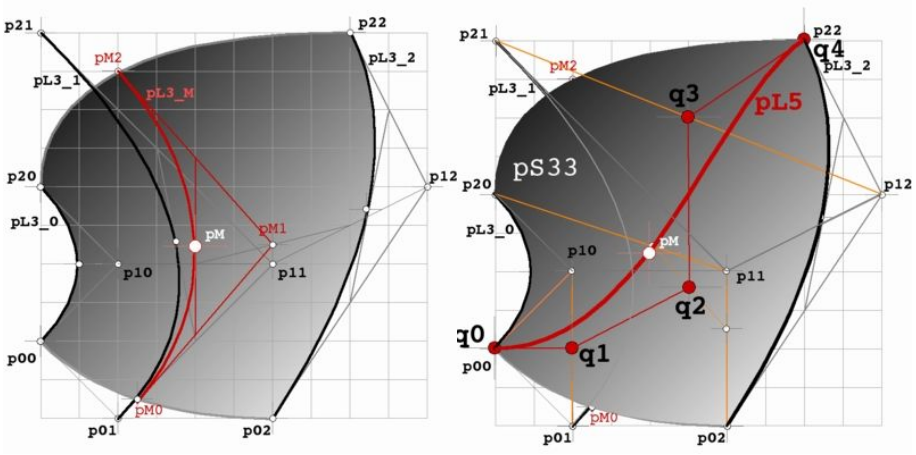


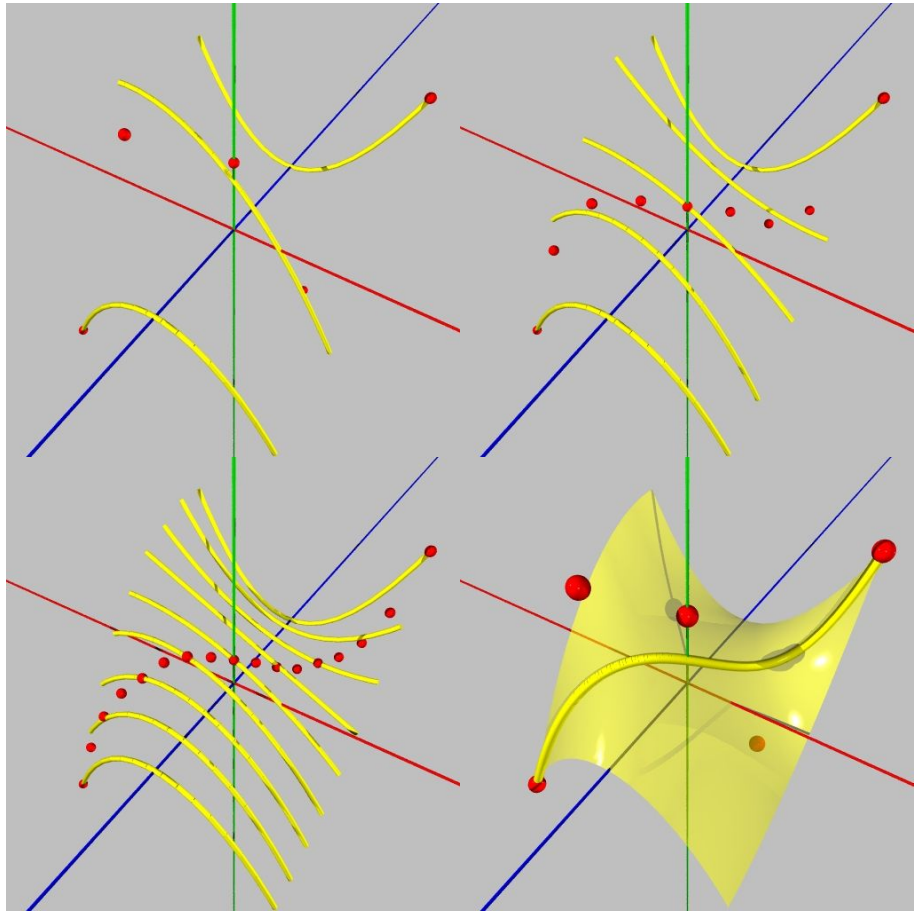
figure 124.1 : représentation orthonormée de la biquadrique (pS33) et des 5 points de contrôle de sa diagonale (pL5).



figures 124.2 et 124.3 : biquadrique (pS33) avec ses trois paraboles génératrices (de contrôle) (pL3), et diagonale (pL5) de la biquadrique avec ses 5 points générateurs (de contrôle),.

Dans la syntaxe de POVRAY/pFlibs, on écrit :

```
// définition d'une biquadrique à partir de 3 paraboles
// et affichage :
#local pS33 = array[3] { pL3_0, pL3_1, pL3_2 }
draw(2,pS33,finesse(<3,3>)+surface(LISSE)+ma_couleur(<1,1,1>))
// extraction des points de contrôle de la diagonale :
#local p0  =   pS33[0][0];
#local p1  = ( pS33[0][1]+  pS33[1][0]             ) / 2;
#local p2  = ( pS33[0][2]+4*pS33[1][1]+ pS33[2][0] ) / 6;
#local p3  = (              pS33[1][2]+ pS33[2][1] ) / 2;
#local p4  =                            pS33[2][2];
// définition de la pL5 et affichage :
#local pL5= array[5] { p0, p1, p2, p3, p4 }
draw( 1, pL5, STANDARD ) // points de contrôle
draw( 1, pL5, finesse(5)+courbe(0.02)+ma_couleur(<1,0,0>) )
```



figures 124.4 à 124.8 : biquadrique (pS33) définie par trois paraboles (pL3) et sa diagonale pL5.

On notera que la biquadrique (pS33) est la première, et donc et la plus simple, surface gauche non réglée de la famille des pFormes. La biquadrique, ainsi que sa diagonale (pL5), ont une importance considérable dans tout ce qui suit.

13 généralisation : les pFormes

Partant du point milieu d'une facette gauche nous avons appliqué un processus récursif en diagonale et créé une parabole ; partant du segment milieu d'un cube gauche nous avons appliqué le même processus récursif en diagonale et créé un paraboloïde réglé ; partant du point milieu du paraboloïde réglé nous avons de même créé une cubique. Nous avons noté la correspondance entre ces diagonales et les formes produites par les opérateurs MI() et MIR() redéfinis pour s'appliquer à un nombre quelconque de formes, écriture nouvelle et généralisée de l'algorithme proposé par de Casteljau en 1959. Convenant toujours de noter une courbe pL, une surface pS, un volume pV, etc…, suivis d'indices précisant le nombre de formes génératrices, on a construit pour l'instant les formes :

```
MIR( pP_0, pP_1 )                  -> pL2     segment
MIR( pL2_0, pL2_1 )                -> pS22    facette
MIR( pS22_0, pS22_1 )              -> pV222   cube
MIR( pV222_0, pV222_1 )            -> pH2222  hypercube
DIAG( pS22 )  = MIR(pP_0,.., pP_2) -> pL3   parabole
DIAG( pV222 ) = MIR(pL3_0, pL3_1)  -> pS32  paraboloïde réglé
DIAG( pS32 )  = MIR(pP_0,.., pP_3) -> pL4   cubique
```

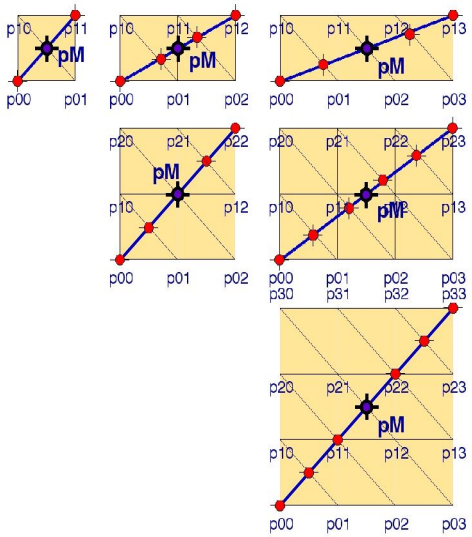


figure 13.1 : les premières pSurfaces et de leurs diagonales, de la pS22 à la pS44.

On va maintenant construire de façon plus systématique l'expression du point milieu des diagonales de différentes surfaces primitives , de la pS22 (facette gauche) à la pS44 (bicubique), cf figure 13.1 :

```
pS22:
pM = MI( MI(p00,p01),
         MI(p10,p11) )
   = ((p00+p01)/2 + (p10+p11)/2 )/2
   = ( p00 + 2(p01+p10)/2 + p11 )/4
   = ( q0 + 2q1 + q2 )/4
   = MI( q0,q1,q2 )

pS32:
pM = MI( MI(p00,p01,p02),
         MI(p10,p11,p12) )
   = ((p00+2p01+p02)/4 + (p10+2p11+p12)/4 )/2
   = ( p00 + 3(2p01+p10)/3 + 3(p02+3p11)/3 + p12)/8
   = ( q0 + 3q1 + 3q2 + q3 )/8
   = MI( q0,q1,q2,q3 )

pS33:
pM = MI( MI(p00,p01,p02),
         MI(p10,p11,p12),
         MI(p20,p21,p22) )
   = ( (p00+2p01+p02)/4+2(p10+2p11+p12)/4+(p20+2p21+p22)/4)/4
   = ( p00+4(p01+p10)/2+6(p02+4p11+p20)/6+4(p12+p21)/2+p22)/16
   = ( q0 + 4q1 + 6q2 + 4q3 + q4)/16
   = MI( q0,q1,q2,q3,q4 )

pS42:
pM = MI( MI(p00,p01,p02,p03),
         MI(p10,p11,p12,p13) )
   = ( (p00+3p01+3p02+p03)/8 + (p10+3p11+3p12+p13)/8 )/2
   = ( p00+4(3p01+p10)/4+6(3p02+3p11)/6+4(p03+3p12)/4+p13)/16
   = ( q0 + 4q1 + 6q2 + 4q3 + q4 )/16
   = MI( q0,q1,q2,q3,q4 )

pS43:
pM = MI( MI(p00,p01,p02,p03),
         MI(p10,p11,p12,p13),
         MI(p20,p21,p22,p23) )
   = (  (p00+3p01+3p02+p03)/8 + 2(p10+3p11+3p12+p13)/8
         + (p20+3p21+3p22+p23)/8 )/4
   = (  p00 + 5(3p01+2p10)/5 + 10(3p02+6p11+p20)/10
         + 10(p03+6p12+3p21)/10 + 5(2p13+3p21)/5 + p23)/32
```

```
   = ( q0 + 5q1 + 10q2 + 10q3 + 5q4 + q5 )/32
   = MI( q0,q1,q2,q3,q4,q5 )

pS44:
pM = MI( MI(p00,p01,p02,p03),
         MI(p10,p11,p12,p13),
         MI(p20,p21,p22,p23),
         MI(p30,p31,p32,p33) )
   = (  (p00+3p01+3p02+p03)/8 + 3(p10+3p11+3p12+p13)/8
            + 3(p20+3p21+3p22+p23)/8+(p30+3p31+3p32+p33)/8 )/8
   = (  p00 + 6(p01+p10)/2 + 15(p02+3p11+p20)/5
+20(p03+9p12+9p21+p30)/20+15(p13+3p22+p31)/5+6(p23+p32)/2+p33)/64
   = ( q0 + 6q1 + 15q2 + 20q3 + 15q4 + 6q5 + q6 )/64
   = MI( q0,q1,q2,q3,q4,q5,q6 )
```

Nous pourrions étudier de la même façon des pSurfaces diagonales dans des pVolumes, et ainsi de suite… De façon générale, nous constatons que :

- l'opérateur MI() produit une forme de même dimension,
- l'opérateur MIR() produit une forme de dimension supérieure,
- l'opérateur DIAG() produit une forme de dimension inférieure.

Une forme milieu peut toujours s'écrire sous une forme explicite du type :

```
n = 2 formes: Fm = (           1.F0 + 1.F1            ) /2
n = 3 formes: Fm = (        1.F0 + 2.F1 + 1.F2        ) /4
n = 4 formes: Fm = (    1.F0 + 3.F1 + 3.F2 + 1.F3     ) /8
n = 5 formes: Fm = ( 1.F0 + 4.F1 + 6.F2 + 4.F3 + 1.F4 ) /16
```

expressions où l'on voit apparaître les coefficients du triangle de Pascal, conduisant à l'expression générale suivante pour le cas de n formes :

$$MI(\ Fi\ ) = \sum_i C_{n-1}^{\ i}\ .\ Fi\ /\ 2^{(n-1)}$$

$$\text{avec}\quad C_n^{\ i} = n!/(i!.(n-i)!) \text{ et } i \text{ dans } [0,n-1]$$

Noter la propriété suivante, pour une combinaison linéaire valable :

$$\sum_i C_{n-1}^{\ i}\ /\ 2^{(n-1)} = 1$$

Ce type de coefficients et l'approche particulière utilisée dans cette étude nous amènent à proposer pour cette famille de formes le nom de "*formes pascaliennes*", ou "*pFormes*" (avec les notations dérivées : pCourbe, pSurface, pVolume, etc..), la lettre "p" rappelant le "P" de Pascal et le fait que ces formes sont toutes des ensembles denses (et non continus) de points, des formes pas tout à fait équivalentes aux formes de la géométrie ordinaire, même si elles peuvent en être aussi proche qu'il est souhaitable. Il y aurait beaucoup à dire sur le sujet de la continuité...

Remarque 1 : bien avant Blaise Pascal, philosophe et mathématicien français du XVIIème siècle, *Omar Khayyam*, poète et mathématicien iranien du XI/XIIème siècle et *Yanghui*, mathématicien chinois du XIIème siècle, avaient étudié les propriétés de ce triangle de nombres; on aurait donc pu appeler les pFormes des kFormes ou des yFormes, formes khayyamiennes ou yanghuiennes, mais c'est moins facile à prononcer pour un francophone que forme pascalienne, sans oublier que dans pascalien il y a 'alien' :). Alors pourquoi pas 'pascalien' ?

Remarque 2 : ces formes pascaliennes ne sont pas nouvelles. La littérature classique en la matière parle de courbes et de surfaces de Bézier qui ont été étudiées sous toutes leurs coutures, et les approches en sont diverses : Pierre Bézier, ingénieur chez Renault, en a fait une approche algébrique/analytique construite sur les coefficients de Bernstein ; Paul de Casteljau, ingénieur chez Citroën, en a davantage étudié le caractère géométrique et récursif. Les B_splines et les NURBS, en sont des prolongements importants. Mais la présentation qu'on en fait conduit souvent à une forêt de notations et de définitions parfois obscures qui peuvent faire perdre le contact avec les choses simples, avec les propriétés fondamentales et les gestes primitifs. En choisissant de limiter l'approche à une poignée d'opérateurs géométriques élémentaires appliqués à un ensemble de points, associée à une représentation matérielle qui peut être limitée à une simple corde pour tracer des segments et en trouver le milieu, on peut conserver le contact avec les gestes simples du dessin à main levée, et on est naturellement conduit à aborder facilement des formes appartenant à des espaces plus complexes, notamment les formes immergées.

Remarque 3 : les formes pascaliennes sont un sous–ensemble de l'ensemble des formes valables obtenues par combinaisons linéaires récursives de combinaisons linéaires de points, valables parce que les sommes des coefficients sont toujours unitaires. Elles conservent les propriétés des formes linéaires et peuvent constituer une bonne base d'approche de formes plus complexes, comme les surfaces définies par des équations implicites ou les équations contenant des fonctions transcendantes (trigonométriques par exemple) ou produites par des processus itératifs comme les courbes géodésiques et les surfaces minimales solutions de systèmes différentiels du type Laplace. On pourra les utiliser comme premiers termes d'un développement en série d'une forme en un point, (plan tangent à une surface, facette gauche ou biquadrique osculatrice,…), analyser le comportement des diagonales, et au delà envisager toute une géométrie locale.

La définition générale des formes pascaliennes étant donnée, nous pouvons en analyser les principales propriétés ainsi que les diverses déclinaisons et combinaisons.

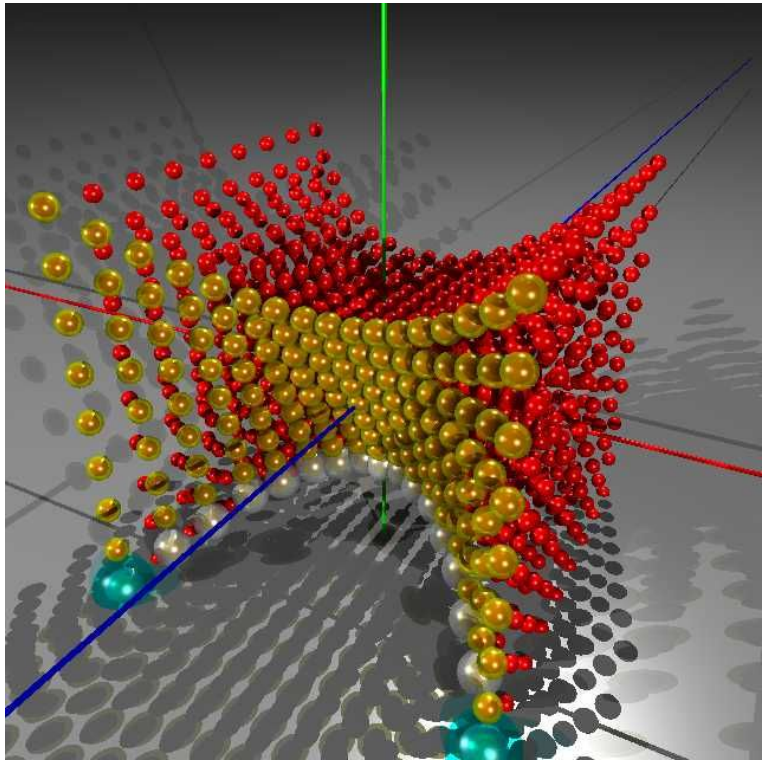


figure 13.2 : un cube gauche (pV322) construit sur deux paraboloïdes réglés (pS32) .

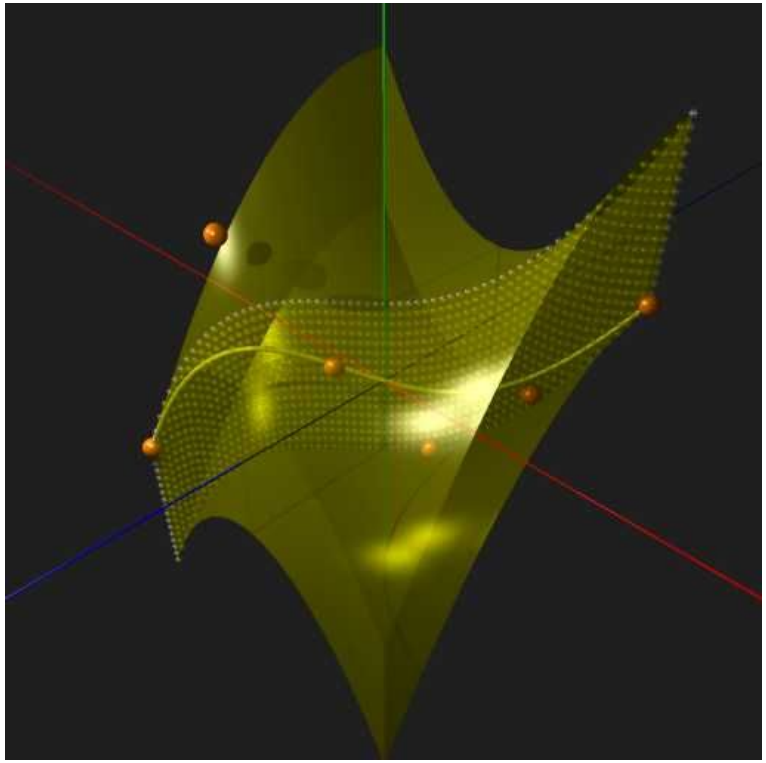


figure 13.3 : un pV332 jaune transparent construit sur deux pS33, sa surface diagonale billes jaunes (pS42) et sa courbe diagonale jaune (pL6).

2 opérations, propriétés

Au sommaire de cette section :

Les pFormes sont des formes de Bézier abordées d'une façon qui se veut unitaire et simple et héritent de toutes leurs propriétés. Nous citerons notamment dans le cas des pCourbes :

- convexité : la pCourbe est contenue dans son polygone de contrôle ;
- invariance affine : la transformée d'une pCourbe est la pCourbe construite sur les transformés des points de contrôle ;
- dérivation : la "dérivée" (on devrait dire hodographe) d'une pCourbe (pLn) est une pCourbe (pLn-1) définie sur les différences des points de contrôle initiaux (le nombre de points est donc décrémenté de 1) ; les points de contrôle aux extrémités donnent immédiatement les repères tangents à une pCourbe (trièdre de Serret-Frenet TNB) ;
- découpe du polygone de contrôle en deux concaténés: permettant de tracer effectivement la pCourbe par approche récursive, application immédiate de l'algorithme de de Casteljau, y compris dans sa généralisation aux pCourbes immergées.

Ces propriétés s'étendent aux pSurfaces et au-delà à toutes les pFormes. Pour plus d'informations, on pourra se rapporter à la bibliographie proposée en annexe.

Dans ce qui suit, nous avons choisi d'analyser les propriétés des pFormes en décrivant les opérateurs implémentés dans POVRAY/pFlibs, dont certains ont déjà été utilisés dans le chapitre précédent. Seront ainsi présentés :

1. les opérateurs fondamentaux assurant la subdivision des pFormes, le changement d'intervalle de définition, l'élévation du degré, les transformations dans l'espace (translations, rotations, homothéties) et l'affichage ;
2. une famille particulière d'opérateurs, les opérateurs get(), permettant de récupérer une sous-forme, un point d'une pForme et le repère local ;
3. enfin une famille d'opérateurs associés à une propriété importante des pFormes, les opérateurs d'immersion, qui étendent la définition des pFormes à des espaces courbes.

21 opérations fondamentales

Au sommaire de cette section :

Sont étudiés les opérateurs pFsubdivision(), pFelevation(), pFstretch() et les opérateurs d'extraction pFgetSubForm(), pFgetPoint() et pFgetPijk(). Le préfixe pF_____() indique l'applicabilité de l'opérateur à n'importe quel type de pForme, dans le cas contraire, l'opérateur n'est envisageable que pour un sous-ensemble de pFormes comme par exemple les pCourbes.

211 subdivision

Appliqué par exemple à trois points, l'opérateur MIR() produit un ensemble infini de points constituant un arc de parabole. Pratiquement, on est amené à définir un opérateur noté pFsubdivision() lançant une récursion sur un nombre d'étapes qui ne peut être que fini et produisant donc un nombre fini de points. De plus, pour ne pas avoir à définir autant d'opérateurs pFsubdivision() que de nombres de points de départ, on regroupera ces points en un tableau passé en paramètre à l'opérateur. Un tableau pouvant contenir soit des points, soit d'autres tableaux, il devient possible d'étendre l'application de l'opérateur aux pFormes de dimension supérieure ; POVRAY sachant traiter facilement les vecteurs à quatre dimensions, on s'arrêtera pratiquement à 4, c'est à dire aux pFormes de dimension 4. Une pForme (un tableau de sous–pFormes pouvant être réduites à des points) étant donnée, l'appel de l'opérateur pFsubdivision() s'écrira ainsi :

```
#local pForme = pFsubdivision( dim, pForme, recursion )
                avec  dim = un nombre entier compris entre 1 et 4
                et recursion = un vecteur  contenant 4 réels >=0
```

pour produire un nouvelle pForme, un tableau contenant une subdivision du tableau initial, plus "proche" de la pForme théoriquement atteinte au bout d'un nombre infini de récursions. On conviendra de confondre le tableau initial et la pForme finale, on appelera par exemple parabole un tableau de trois points, considérant que ces 3 points conduisent à une parabole par application de l'opérateur théorique MIR() et à une polyligne aussi proche qu'on le voudra de la parabole par application de l'opérateur "réel" pFsubdivision().

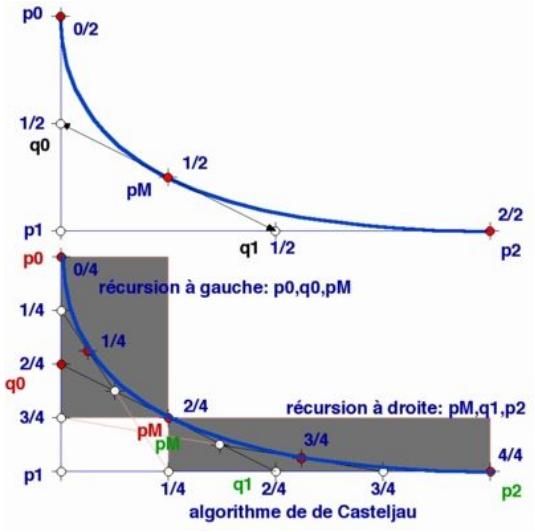


figure 211 : l'algorithme de de Casteljau, pour t=1/2, puis t=1/4 et t=3/4.

212 élévation du degré

Une condition d'application des opérateurs MI() et MIR() passée au second plan jusqu'ici était que les pFormes devaient être de même type, de même dimension, bien sûr, mais aussi de même nombre de sous–formes de définition (de même degré dans la terminologie classique). Rechercher la forme milieu d'une pL2 (segment) et d'une pL3 (parabole) semble a priori dénué de sens…, sauf si le segment est considéré comme une parabole "dégénérée" construite par exemple sur trois points alignés et équidistants. Noter que l'inverse est en général impossible, on ne peut pas réduire une parabole à un segment, sauf si elle est dégénérée, construite sur trois points alignés. Pour élever le degré d'une parabole, on définit 4 points en fonction des points de la parabole :

```
q0  =  p0
q1  = (p0 + 2*p1     )/3
q2  = (     2*p1 + p2)/3
q3  =              p2
le point milieu d'une parabole (p0,p1,p2) s'écrit:
pm  = ( p0 + 2*p1 + p2)/4
    = ( q0 + 3*q1 + 3*q2 + q3)/8
```

expression dans laquelle on reconnaît le point milieu d'une cubique (q0,q1,q2,q3).

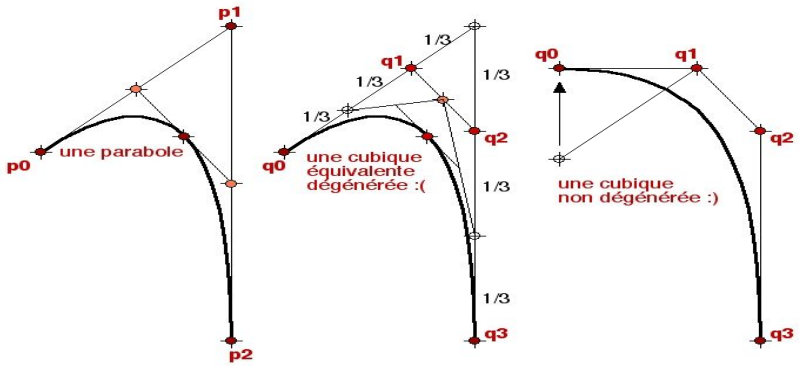


figure 212 : élévation du degré d'une pL3 (parabole), qui devient une pL4 (cubique dégénérée) puis une cubique à part entière.

De manière générale, on est amené à définir un opérateur noté pFelevation() qui appliqué à une pCourbe pLn produira la "même" pCourbe (pLN) contrôlée par N>n points, appliqué à une pSurface pSmn produira la même pSurface (pSMN) contrôlée par M>m et N>n points, et ainsi de suite pour toute pForme de dimension quelconque. Pour l'instant l'implémentation est opérationnelle pour les pCourbes et les pSurfaces, et en attente pour les pFormes de dimension quelconque, sans grande importance sur le plan pratique. L'appel de l'opérateur pFelevation() s'écrira ainsi :

```
#local pForm = pFelevation( dim, pForm, elevation )
               avec dim = un nombre entier compris entre 1 et 2
               et elevation  = un vecteur  contenant 2 réels >=0
```

213 reparamétrisation

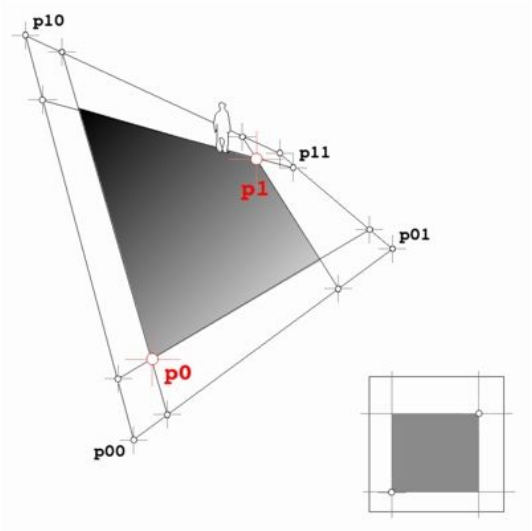


figure 213.1 : sous-facette gauche définie sur deux points p0 et p1.

On n'oublie pas que les pCourbes sont des courbes de Bézier et qu'on pourrait par exemple écrire une pL3 (arc de parabole) sous la forme algébrique classique suivante :

$$p = (1-u)^2.p0 + 2.(1-u).u.p1 + u^2.p2$$

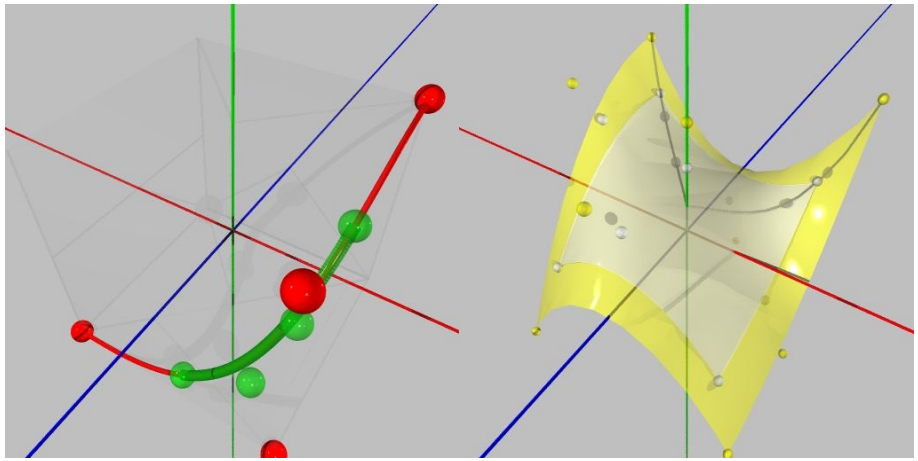
où p0, p1 et p2 sont trois points quelconques de l'espace, u un nombre réel dans l'intervalle [0,1], et p un point quelconque de la parabole situé entre p0 et p1. Mais l'expression algébrique produit encore un point de la parabole pour une valeur de t dans un intervalle quelconque [a, b], a et b pouvant être infinis.

Une pL3 étant au départ définie par 3 points (p0,p1,p2) entre les points extrêmes p0 et p1, il peut être utile d'étendre (ou de restreindre) l'intervalle de définition à des portions situées entre deux points quelconques de la parabole, pA et pB, et l'on est ainsi amené à définir un opérateur noté pFstretch(). Appliqué à une pCourbe, cet opérateur créera une sous–courbe entre les deux points spécifiés en abscisses curvilignes u0 et u1, appliqué à une pSurface, il créera une sous–facette entre les deux points spécifiés en 2 abscisses et ordonnées curvilignes (u0,v0) et (u1,v1), et ainsi de suite jusqu'à la dimension 4.

L'appel de l'opérateur pFstretch() s'écrira ainsi :

```
#local pForm = pFstretch( dim, pForm, pA, pB )
               avec  dim = un nombre entier compris entre 1 et 4
               et pA, pB = deux points écrits dans R4 (  )
```

pour produire une nouvelle pForme, "immergée" dans la première entre les points pA et pB. D'autres immersions seront envisagées un peu plus loin, qui nous conduiront à de nouvelles généralisations des pFormes.



figures 213.2 et 213.3 : reparamétrisation d'une pL4 (cubique), et d'une pS33 (biquadrique) aboutissant à une restriction de ces pFormes.

214 extractions, repères tangents

Trois opérateurs "get()", c'est à dire *prendre* ou *retourner* en anglais, sont définis pour (re) trouver une sous–forme génératrice de la pForme, avec au bout du compte un point quelconque de la pForme, et le repère local en ce point.

2141 pFgetSubForm()

Une pForme est le produit de l'application de l'opérateur MIR() à un tableau de sous–formes. L'opérateur pFgetSubForm() retourne la sous–forme génératrice en u de la pForme ; pour u compris dans l'intervalle [0,1] la sous–forme sera contenue dans la pForme. L'appel s'écrit ainsi :

```
#local pForm = pFgetSubForm( dim, pForm, u )
               avec dim dans [1,4] et u dans R
```

2142 pFgetPoint()

L'application récursive du précédent opérateur pFgetSubForm() poussée aux limites retournera dans R4 un point p ; c'est ce que produit l'opérateur pFgetPoint() dont la syntaxe d'appel est la suivante -cf figure 2143.1 - :

```
#local P = pFgetPoint( dim, pForm, p )
           avec dim dans [1,4] et p dans R4
```

Noter que le paramètre p est un point de R4 ; dans le cas d'une pSurface par exemple, on écrirait :

```
p = pFgetPoint( 2, pSurf, <u,v,0,1> ).
```

2143 pFgetPijk()

L'étude des propriétés locales en un point d'une pForme commence avec le calcul du repère local en ce point. On étudiera le cas des courbes, des surfaces et des volumes.

21431 cas d'une courbe

Le repère local en un point d'une courbe (appelé trièdre de Serret–Frenet) est l'ensemble de trois vecteurs unitaires construits sur la tangente au point courant, la normale contenue dans la plan osculateur à la courbe en ce point et la binormale perpendiculaire à ce plan -cf figure 2143.4 -. Dans le cas d'une pCourbe ces trois vecteurs sont aisément calculables au point de contrôle de départ ; soit p0 ce point de contrôle, p1 et p2 les deux points de contrôle suivants :

1. le vecteur porté par p0 et p1 est le vecteur tangent en p0 à la courbe ;
2. le plan défini par les points (p0,p1,p2) étant par construction osculateur à la courbe en p0, le produit vectoriel des vecteurs p0p1 et p0p2 donne la binormale ;
3. le produit vectoriel de la binormale et de la tangente donne la normale.

Le calcul du trièdre de Serret–Frenet en un autre point que le premier point de contrôle, par exemple au point d'abscisse curvilgne "s", se fait en calculant la courbe équivalente démarrant en ce point, c'est-à-dire en déplaçant l'intervalle de définition de [0,1] à [s,s+1] et en calculant le trièdre en ce nouveau point de départ. Il faut noter que le trièdre de Serret–Frenet n'est pas défini en tout point d'un segment, et pour les "vraies" courbes, au droit des points d'inflexion où la courbure est nulle et le plan osculateur indéterminé. De plus, la courbure s'inversant au droit d'un point d'inflexion, le trièdre bascule de 180° ce qui a pour effet de provoquer une brusque torsion des surfaces tubulaires construites sur le trièdre de Serret–Frenet, abordées au paragraphe 323.

21432 cas d'une surface

Le repère local en un point est l'ensemble de trois vecteurs unitaires construits sur deux tangentes au point courant (tu/|tu|,tv/|tv|), et la normale perpendiculaire à ce plan (produit vectoriel). De façon analogue à ce qui a été vu plus haut, ces trois vecteurs sont aisément calculables au point de contrôle de départ ; soit p00 ce point de contrôle, p01 et p10 les points de contrôle suivants dans les deux directions:

1. les vecteurs portés par les couples (p00,p01) et (p00,P10) sont tangents en p00 à la surface, mais non orthogonaux entre eux ;
2. le produit vectoriel de ces deux vecteurs est normal au plan tangent en ce point.

Le calcul du repère tangent en un autre point se fera de façon analogue à celle utilisée pour la courbe en déplaçant l'intervalle de définition -cf figures 2143.1 à 2143.3-.

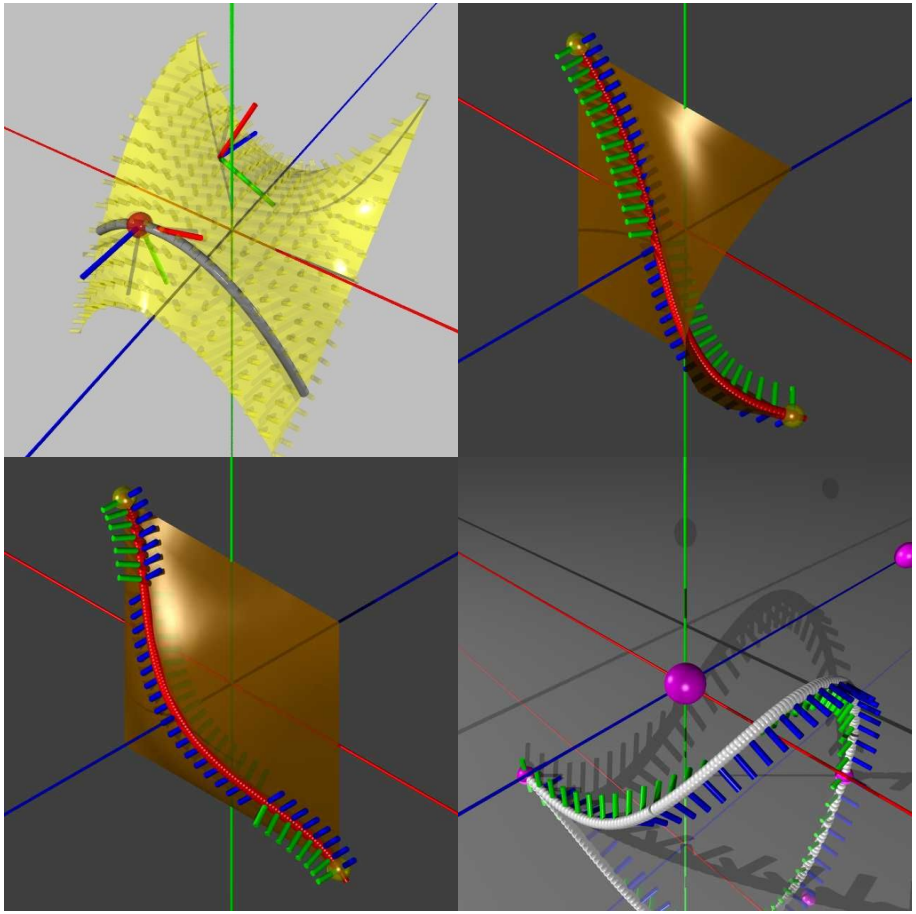
21433 cas d'un volume

Le repère local en un point est simplement constitué par le triplet de vecteurs unitaires (tu/|tu|, tv/|tv|,tw/|tw|) construits sur ce point et les 3 points suivants sur les axes u,v,w. On notera cependant que ce repère n'est pas orthogonal.

L'opérateur pFgetPijk() retourne le repère local dans le cas des pCourbes et des pSurfaces, syntaxe d'appel :

```
#local Pijk = pFgetPijk( dim, pForm, p )
              avec dim dans [1,2] et p dans R4
```

Remarque : Dans le cas d'une courbe, le repère (Serret-Frenet) en chaque point d'une courbe est orthonormé, dans le cas d'une surface les trois vecteurs sont unitaires mais les deux vecteurs tangents à la surface ne sont pas orthogonaux entre eux, dans le cas d'un volume, les trois vecteurs sont unitaires mais non orthogonaux entre eux. Faire appel aux concepts de norme et d'orthogonalité n'est pas vraiment en cohérence avec les principes d'une géométrie affine qui se veut indépendante de toute métrique. Nous verrons plus loin (323 Surfaces tubulaires) les conséquences négatives d'une telle introduction, à tel point qu'on peut se poser la question de l'intérêt d'introduire la norme et l'orthogonalité dans les repères tangents. La question reste levée...



figures 2143.1 à 2143.4 : l'opérateur pFgetPijk() retourne le repère tangent à une pSurface ou à une pCourbe.

22 immersions

Au sommaire de cette section :

Dans la première partie, nous avons vu l'importance que prenait l'opérateur DIAG() dans la définition même d'une pSurface, produisant une pL3 (parabole) dans une pS22 (facette gauche), une pL5 dans une pS33 (biquadrique), une pS32 (paraboloïde réglé) dans un pV222 (cube gauche), etc… Ces diagonales vont se révéler être des constituants fondamentaux de ces pSurfaces et au-delà de toutes les pFormes, à partir desquels toute une géométrie dans les surfaces gauches peut être envisagée. Au passage, il sera nécessaire d'étudier les conditions dans lesquelles une pForme peut interpoler un tableau de points, (ce qui n'est pas le cas des points de contrôle, hors les extrêmités). Et au final apparaîtront d'intéressantes propriétés d'immersion de pFormes dans d'autres pFormes, qui semblent justifier en elles–mêmes l'approche "pascalienne" des formes gauches.

221 interpolation

Les pCourbes n'interpolent pas les points de contrôle intermédiaires. On peut souhaiter travailler avec des points de contrôle se trouvant sur la pCourbe, générant en arrière plan ses points de contrôle "réels". Il n'y a pas unicité de solution, plusieurs courbes peuvent interpoler une suite de points, suivant ce qu'on pourrait appeler les poids respectifs associés à ces points, et/ou d'autres conditions sur les tangentes, les courbures, etc... Dans cette étude, on considèrera que les points interpolés , appelés noeuds, forment une suite répartie uniformément aux abscisses i/N avec i=[0,N], on laissera au lecteur le soin d'aller plus loin, au besoin ;)...

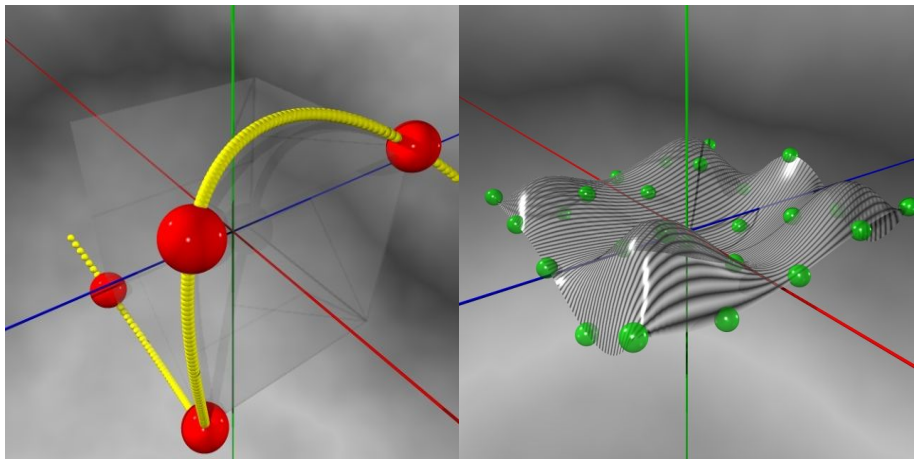
Le calcul se fait en exprimant ces points (noeuds) en fonction des points de contrôle de la pCourbe cherchée et en inversant le système linéaire ainsi formé. Ci–dessous sont détaillées les solutions correspondant aux cas courants de la pL3 (parabole), de la pL4 (cubique) et de la pL5, les points b0 à b4 étant les noeuds à interpoler :

```
- cas d'une parabole pL3:
#local L3 = array[3]
#local L3[0] = b0;
#local L3[1] = (-b0 +4*b1 -b2)/2;
#local L3[2] = b2;

- cas d'une cubique pL4: -cf figure 221.1-
#local L4 = array[4]
#local L4[0] = b0;
#local L4[1] = (-5*b0 +18*b1  -9*b2 +2*b3)/6;
#local L4[2] = ( 2*b0  -9*b1 +18*b2 -5*b3)/6;
#local L4[3] = b3;

- cas d'une pL5:
#local L5 = array[5]
#local L5[0] = b0;
#local L5[1] = (-13*b0 +48*b1  -36*b2 +16*b3  -3*b4)/12;
#local L5[2] = ( 13*b0 -64*b1 +120*b2 -64*b3 +13*b4)/18;
#local L5[3] = ( -3*b0 +16*b1  -36*b2 +48*b3 -13*b4)/12;
#local L5[4] = b4;
```

Au-delà il est préférable d'utiliser une méthode générale, et la bibliothèque POVRAY/pFlibs propose un opérateur d'inversion basé sur l'algorithme de Gauss–Jordan, qui s'avère très rapide et bien adapté au problème à traiter.

L'implémentation de cet opérateur est pour l'instant limitée aux courbes, pLinterpolante(), et aux surfaces pSinterpolante() avec une syntaxe d'appel suivante :

```
// curv tableau de points
#local courbe_interpolante = pLinterpolante( curv )

// surf tableau de tableaux de points
#local surface_interpolante = pSinterpolante( surf )
```



figures 221.1 et 221.2 : pl4 (cubique) interpolant 4 points, et pS55 (?) interpolant 25 points.

222 diagonalisation

En étudiant systématiquement l'expression du point milieu d'une pSurface depuis la pS22 jusqu'à la pS44, nous avons mis en évidence les points de contrôle des diagonales exprimés en fonction des points de contrôle de la pSurface. Une représentation "orthonormée" de la pSurface -la pSurface devient un carré unité- montre bien la distribution uniforme de ces points sur la diagonale et induit une expression générale qui a été implémentée dans un opérateur pFdiagonalisation() applicable à toute pForme de dimension quelconque. L'appel de cet opérateur se fait simplement ainsi :

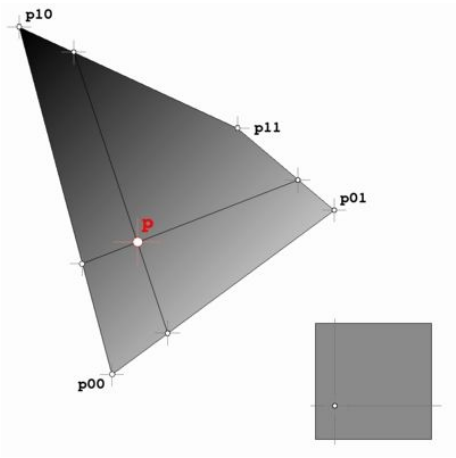
```
#local diag = pFdiagonalisation( dim, pForm ) // avec dim>=2
```

et produit une pForme de dimension inférieure, premier exemple de pForme immergée dans une autre pForme.

223 pForme immergée

Nous allons reconsidérer les diagonales sous un angle plus général qui nous conduira à la construction de pFormes immergées dans d'autres pFormes. Nous commencerons par étudier les courbes qu'il est possible de tracer dans des pSurfaces, après une réflexion rapide sur le point dans une surface.

2231 un point dans une pSurface



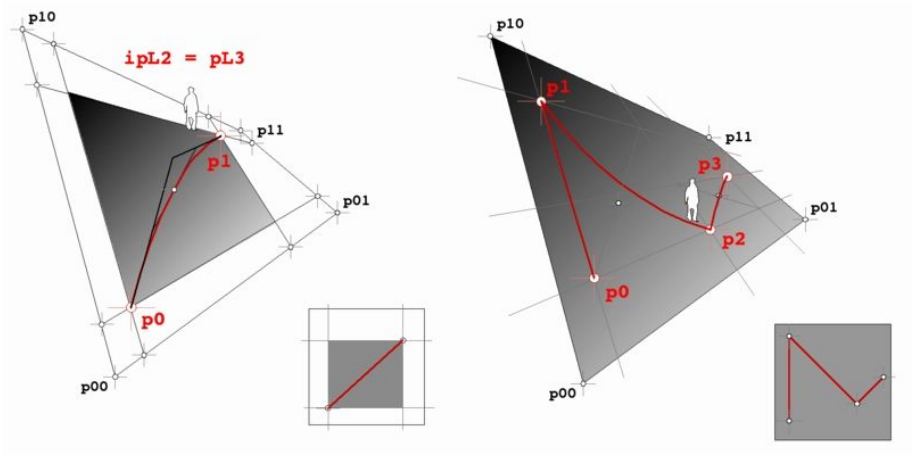
figures 223 : un point dans une pS22 (facette gauche), et représentation orthonormée.

Un point quelconque d'une pSurface est parfaitement défini par la donnée de deux nombres réels (coordonnées curvilignes) généralement compris dans l'intervalle [0,1] ; en fait rien n'interdit de sortir de cet intervalle, la surface est parfaitement définie dans l'intervalle [-infini, +infini]. Dans la syntaxe POVRAY/pFlibs, l'opérateur pFgetPoint() dont la syntaxe d'appel pour une pSurface est la suivante :

```
#local P = pFgetPoint( 2, pSurface, p )
avec l'entrée p = <u,v,0,1> et le résultat P = <x,y,z,t>
```

attend un point p défini dans le repère local (curviligne) sous la forme <u,v,0,1> et retourne un point P dans l'espace global sous la forme <x,y,z,t>, cf 2142. Par le biais de cet opérateur, il est possible d'immerger toute courbe, et même toute surface, dans une pSurface ; un cercle, par exemple, défini par son équation paramétrique canonique P(t) = [cos(t), sin(t)] dans le repère curviligne attaché à la surface, pourra ainsi être transformé point par point en une courbe gauche plaquée sur la surface. La nature de cette courbe gauche définie ponctuellement est à ce stade inconnue, et le but de l'étude qui va suivre est de la découvrir ; on commencera par les courbes les plus simples traçables sur une pSurface, les courbes déterminées par deux points.

2232 deux points dans une pS22



figures 2232.1 et 2232.2 : par deux points d'une pS22 passe un ipL2 (iSegment), et par plusieurs passe une iPolyligne.

Considérons dans un premier temps la plus simple des surfaces gauches, la pS22 (facette gauche). Deux points quelconques p0 et p1 appartenant à la pS22 définissent de manière unitaire une sous–pS22 immergée, notée ipS22, et donc une courbe diagonale unique lui appartenant, une parabole (pL3) reliant ces deux points p0 et p1.

Pour un "habitant" de l'espace extérieur à la surface, cette courbe est une parabole définie par 3 points, mais pour un "habitant" de la surface ignorant l'espace dans lequel cette surface est immergée, cette courbe reliant deux points p0 et p1 se comporte comme un segment «classique ». La courbe est parfaitement définie par la donnée des deux points (le troisième point n'appartient pas à la surface et l'habitant ne le connait pas) ; elle est « isocline » en ce sens qu'elle coupe les génératrices suivant une « pente » constante par construction (l'incrémentation récursive en divisions par deux est égale dans les deux directions) ; et en définissant la métrique sur la base du nombre de points la définissant, on devrait même pouvoir parler de courbe géodésique. Pour ces raisons, on conviendra d'appeller cette courbe « segment immergé » et on la notera « iSegment ».

Construire entre deux points P0 et P1 un segment immergé dans une pS22 revient à construire la diagonale de la sous–facette ipS22 construite sur ces deux points, c'est-à-dire une parabole définie par trois points (B0, B1, B2) ; les points extrêmes étant connus (B0 = P0 et B2 = P1), il reste à déterminer le point intermédiaire B1. On connaît le point P1/2 milieu de la facette ipS22 construite sur (p0,p1):

```
#local P1/2 = pFgetPoint( 2,pS22, <1/2,1/2,0,1>);
```

Inversant l'expression donnant ce point comme le milieu de la parabole externe :

```
P1/2 = MI( B0, B1, B2 )
     = (B0 + 2*B1 + B2)/4
     = (P0 + 2*B1 + P1)/4
```

on trouve :

```
B1   = ( -P0 + 4*P1/2 -P1 )/2
```

et la parabole cherchée :

```
pL3  = MIR( P0, (-P0 + 4*P1/2 -P1 )/2, P1 )
```

De façon "interne", oubliant la parabole, on est naturellement amené à étendre la définition des opérateurs MI() et MIR(), et traitant cette courbe comme une courbe pascalienne immergée notée ipL2, on peut écrire :

```
P1/2  = MI( P0, P1 )
ipL2  = MIR( P0, P1 )
      = pL3
```

Une porte est maintenant ouverte vers la définition de formes pascaliennes dans des espaces courbes, et tout au moins dans des pFormes. Il s'agit dans un premier temps de dépasser le seul cas de la facette gauche.

2233 deux points dans une pS32

Dans le cas d'une pS32 (paraboloïde réglé), construire entre deux points P0 et P1 le segment immergé revient à construire une cubique « externe » définie par quatre points (B0, B1, B2, B3) ; les points extrêmes étant connus (B0 = P0 et B3 = P1), il reste à déterminer les points intermédiaires B1 et B2. On exprime les points P1/3 et P2/3 au 1/3 et au 2/3 de la diagonale de la ipS32 construite sur (P0,P1) comme points au 1/3 et au 2/3 de la pL4,

```
P1/3 = (8.B0 + 12.B1 + 6.B2 + B3)/27
P2/3 = (B0 + 6.B1 + 12.B2 + 8.B3)/27
```

et on inverse pour aboutir aux deux points intermédiaires :

```
B1 = (-5.P0 + 18.P1/3 -9.P2/3 + 2.P1)/6
B2 = (2.P0 - 9.P1/3 + 18.P2/3 - 5.P1)/6
```

et à la cubique cherchée.

Le point milieu de la cubique étant connu, on pourra donc ici aussi écrire :

```
P1/2  = MI( P0, P1 )
ipL2  = MIR( P0, P1 )
      = pL4
```

2234 deux points dans une pSurface

On remarquera qu'une ipL2 (segment immergé) est une pL3 dans une pS22, une pL4 dans une pS32. On aboutirait de même à une pL5 dans une pS42 ou une pS33. De façon générale une ipL2 (segment immergé) dans une pSmn est une pCourbe contrôlée par N = (m+n–1) points. Ces points sont calculés à partir d'une distribution uniforme sur le segment immergé (ipL2) de N points Pi/(N–1) avec i = [0, N–1]. La pCourbe interpolant ces points est la solution du problème, et nous pourons écrire :

```
P1/2  = MI( P0, P1 )
ipL2  = MIR( P0, P1 )
      = pLN  avec N = m+n-1
```

On sait donc à présent tracer des "segments" de droite dans toute pSurface, et l'on peut faire quelques remarques sur cette importante propriété :

Remarque 1 : sur cette base généralisée de segment, on pourrait étendre les concepts de parallélisme, d'orthogonalité, d'angle, étudier l'intersection de deux iSegments, ses divisions, ses prolongements, etc..., et définir ainsi une géométrie dans les pSurfaces.

Remarque 2 : un iSegment n'est pas en général une géodésique, ce n'est pas le chemin le plus court entre deux points, la normale en chaque point de la courbe n'est pas confondue avec la normale à la surface en ce point. Mais il pourrait être intéressant de prendre les iSegments comme base pour l'étude et la construction des géodésiques d'une pSurface.

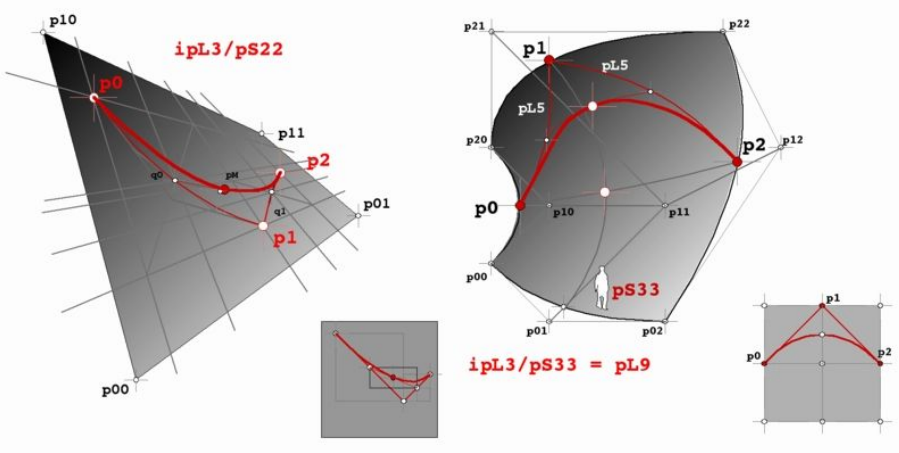
Remarque 3 : l'idée de considérer une parabole comme un simple segment immergé dans une facette gauche peut simplifier un problème difficile à traiter dans l'espace classique euclidien en le ramenant à un problème plus simple dans l'espace courbe défini par la facette. Noter qu'il existe une infinité de facettes admettant une parabole comme diagonale: P0,P1,P2 étant donnés, tous les couples de points q0 et q1 tels que P1 en soit le milieu conviennent.

Remarque 4 : dans le cas où la pSurface est ramenée à sa forme orthonormée (carré unité), ces segments sont bien rectilignes, et on peut considérer toute pSurface et ses courbes immergées comme une "anamorphose" de la forme orthonormée.

Remarque 5 : la construction des pFormes dans un espace indépendant de toute métrique a rendu possible la définition de segments immergés dans des "espaces courbes", tout au moins dans des pFormes gauches. On se souviendra qu'en introduction, afin de trouver le milieu de deux points sur une surface, on envisageait de "*tendre une corde sur la surface en la disposant suivant la courbe gauche unique, appelée géodésique, constituant le chemin le plus court entre ces deux points et le long duquel la normale à la corde dans le plan osculateur est confondue avec la normale à la surface*". Suivre la géodésique entre deux points était la condition imaginée pour assurer l'unicité du chemin. Dans la cas d'une pForme, il devient possible de remplacer la géodésique par le segment immergé, nettement plus pratique à construire à l'aide d'une simple corde: le géomètre chargé de tracer une ligne "droite" sur un terrain de forme localement assimilable à une facette gauche, donc une parabole dans notre espace, positionnera facilement le point intermédiaire au milieu d'une corde tendue suivant une ligne droite entre les deux sommets opposés aux sommets à relier, et à partir des trois points de contrôle, construira le premier point milieu du segment immergé, et ainsi de suite.

Au-delà de deux points donnés dans une pSurface, on peut envisager la construction de polylignes constituées de segments immergés reliant la suite de points, et tracer entr'autre des triangles, des polygones réguliers tendant vers des cercles immergés, des courbes sinusoïdales, etc... Une voie peut–être plus riche consiste à y construire des pCourbes, à commencer par des paraboles immergées, des ipL3.

2235 trois points dans une pSurface



figures 2235.1 et 2235.2 : ipL3 (parabole immergée) dans une pS22 et dans une pS33.

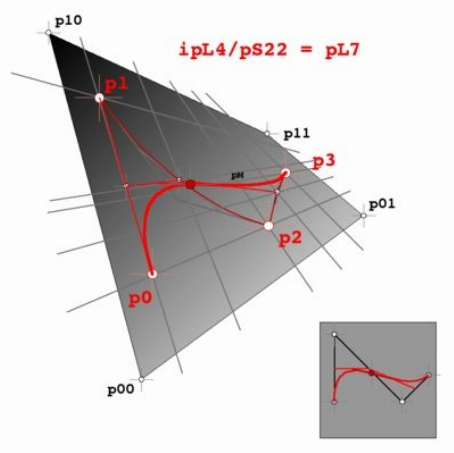
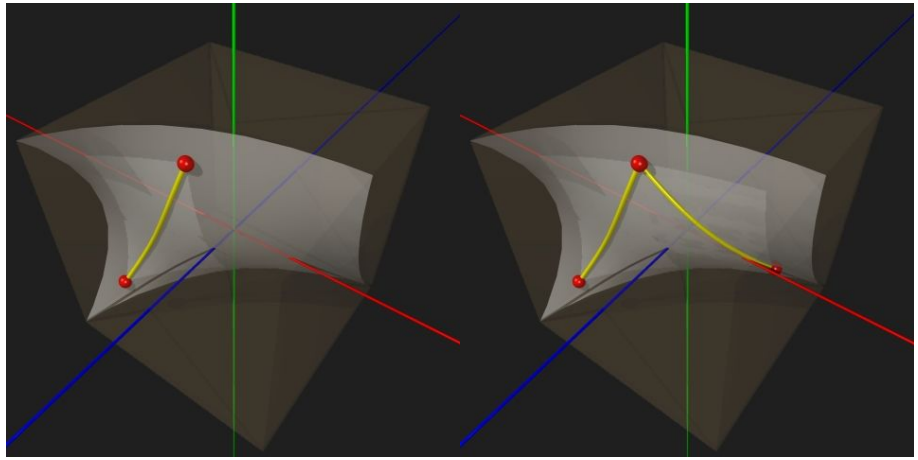


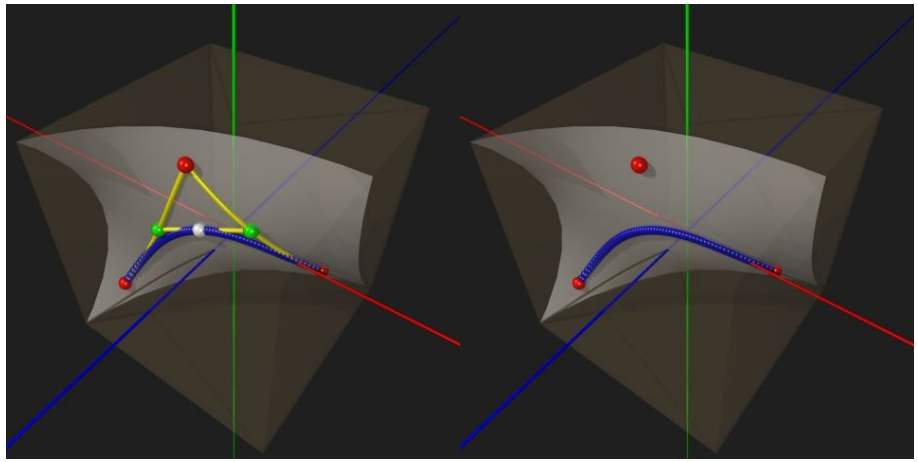
figure 2235.3 : ipL4 (cubique immergée) dans une pS22.

Par analogie avec ce qui précède, on envisage une généralisation des opérateurs MI() et MIR() telle qu'à partir de trois points appartenant à une pSurface ils puissent génèrer une parabole immergée, une ipL3. Et de même qu'à la ipL2 immergée dans une pSurface est associée une pCourbe de l'espace, on va rechercher à quelle pCourbe sera associée la ipL3. Compte tenu de la façon dont sont construites les formes pascaliennes et de leurs propriétés "linéaires", on "sait" que cette pCourbe existe, "on reste en famille". Le processus de construction d'un iSegment faisait appel à une distribution uniforme de points "alignés" sur la surface, le processus de construction d'une ipL3 travaillera sur une distribution uniforme de points sur un chemin parabolique dans la surface (penser à sa représentation orthonormée). Un raisonnement purement inductif conduit aux résultats suivants : dans le cas d'une pS22, la courbe cherchée est une pL5 dont les points de contrôle sont calculés à partir d'une distribution uniforme sur la ipL3 de 5 points Pi/4 avec i = [0, 4]. Dans le cas d'une pS33, on peut montrer qu'il s'agit d'une pL9 .

Et dans le cas général, une parabole immergée (ipL3) dans une pSmn est une pCourbe contrôlée par N = (m+n–2)*2+1 points.

```
P1/2  = MI( P0, P1, P2 )
ipL3  = MIR( P0, P1, P2 )
      = pLN   avec N = (m+n-2)*2+1
```





figures 2235.4 à 2235.8 : à partir de 3 points, construction d'une ipL3 bleue (parabole immergée) et de son polygone de contrôle jaune dans une pS33 (biquadrique).

2236 généralisation : les pFormes immergées

On trouverait de même qu'une cubique immergée (ipL4) définie par quatre points appartenant à une pS22 est une pCourbe de l'espace construite sur 7 points.

Dans le cas d'une pCourbe quelconque immergée dans une pSurface quelconque on peut écrire :

```
N = ( m + n – 2 ) * (q–1) + 1
      où q est le nombre de points de contrôle
      de la pCourbe définis dans la pSurface,
      et (m,n) sont les nombres des points de contrôle
      de la pSurface dans les deux directions.
```

Enfin dans le cas d'une pCourbe quelconque immergée dans une pForme de dimension quelconque, en se basant sur un raisonnement purement inductif (cf remarque), on peut se risquer à écrire :

```
M = ( m1+m2+..+mi – d ) * (q–1) + 1
      où q est le nombre de points de contrôle
      de la courbe définis dans la pForme,
      d est la dimension de la pForme,
      et (m1, m2, ..., mi) sont les nombres des points de contrôle
      de la pForme dans les différentes directions (dimensions).
```

Cette formule s'applique également aux pSurfaces immergées dans une pSurface ou un pVolume, et de façon générale aux pFormes immergées dans des pFormes; il suffit de raisonner séparément sur chaque dimension. Voici un tableau récapitulatif, dans lequel on peut lire par exemple qu'une ipL3 dans une pS33 est une pL9 :

	pS22	pS32	pS33	pS43	pS44	pV222
ipL2	pL3	pL4	pL5	pL6	pL7	pL4
ipL3	pL5	pL7	pL9	pL11	pL13	pL7
ipL4	pL7	pL10	pL13	pL16	pL19	pL10
ipS22	pS33	pS44	pS55	pS55	pS77	pS44
ipV222	/	/	/	/	/	pV444

Une fois connu le nombre N de points de contrôle définissant la pCourbe, on calcule une distribution uniforme de N points sur la pCourbe immergée, et la pCourbe interpolant ces points est la solution du problème.

Ce résultat est remarquable, en ce sens qu'il établit une dualité entre deux représentations d'une même forme, avec ou sans interpolation de points de contrôle, immergée ou pas dans un espace plus complexe. Mais le résultat fondamental est qu'une pForme immergée dans une autre pForme est équivalente à une pForme de l'espace englobant de degré plus élevé; c'est ce qui fait l'intérêt du concept de forme immergée, et l'on peut convenir de parler dans tous les cas de pFormes. Toutes ces pFormes relèvent de l'utilisation de deux opérateurs MI() et MIR() appliqués au départ sur un ensemble quelconque de points de l'espace, puis sur les formes générées, puis sur des formes appartenant à ces formes, etc…. Toutes ces pFormes sont basées sur des combinaisons linéaires à coefficients de Pascal. Nous étudierons dans le prochain chapitre de nouvelles combinaisons de ces pFormes afin d'étendre leur champ d'application à des cas plus complexes et tout d'abord les formes basées sur les cercles.

Remarque : affirmer les résultats précédents, qu'il s'agisse des formules de correspondance des points de contrôle ou de l'affirmation «*... la pCourbe interpolant ces points est la solution du problème*. » suppose une démonstration mathématique rigoureuse qui reste à donner. Les pFormes ne sont en fait que des combinaisons multilinéaires de points, une parabole n'est qu'une forme bilinéaire dégénérée, une cubique est une forme trilinéaire dégénérée, et ainsi de suite. Il semble raisonnable de penser que les résultats constatés sur les premières formes sont transmises aux pFormes plus complexes qu'elles engendrent, et une formalisation mathématique précise de cette réflexion pourrait en constituer une démonstration rigoureuse. Les algorithmes développés dans l'implémentation informatique constituent une « machine expérimentale » à produire des pFormes et à tester leurs propriétés, et une fois rodés, ils ont validé jusqu'à présent au moins visuellement tous les cas de figure envisagés. Ces algorithmes ont peut-être par eux-mêmes valeur de démonstration ! Et s'il s'avérait que les courbes immergées ne l'étaient pas en dehors des points des pSurfaces qu'elles interpolent, cela n'enlèverait probablement pas grand chose à leur utilité pratique, et de plus, sur un plan purement intellectuel, le mauvais moment passé, c'est tout un nouveau champ d'expérimentation qui s'ouvrirait, à la recherche des raisons mathématiques plus profondes invalidant ces affirmations.

Coté implémentation dans POVRAY/pFlibs de l'opérateur pFimmersion(), outre l'opérateur pFdiagonalisation() qui ne peut produire que des iSegments dans toute pForme, deux versions ont été réalisées, la première applicable à toute pForme mais produisant la ipForme sans ses points de contrôle, et la seconde applicable aux pSurfaces produisant une ipCourbe avec ses points de contrôle.

```
#local iForm = pFimmersion( dim, pForm, idim, ipForm )
           avec dim, idim dans [2,4], f et imf deux pFormes

#local ipCourbe = courbe_in_surface( courbe, surf )
```

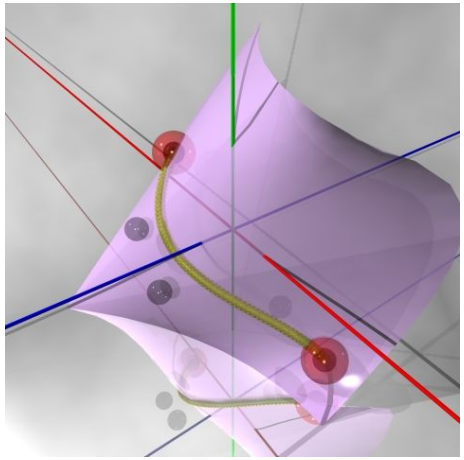


figure 2236.1 : un ipL2 dans une pS33 est une pL5 de l'espace .

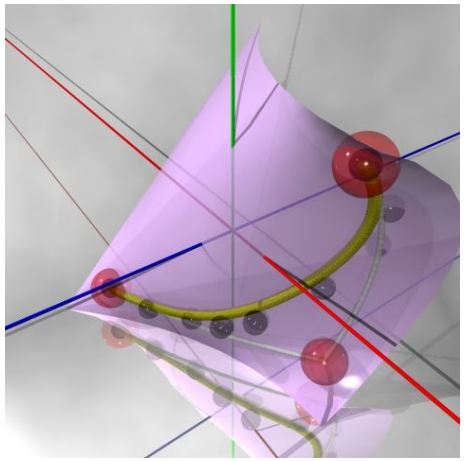


figure 2236.2 : une ipL3 dans une pS33 est une pL9 de l'espace .

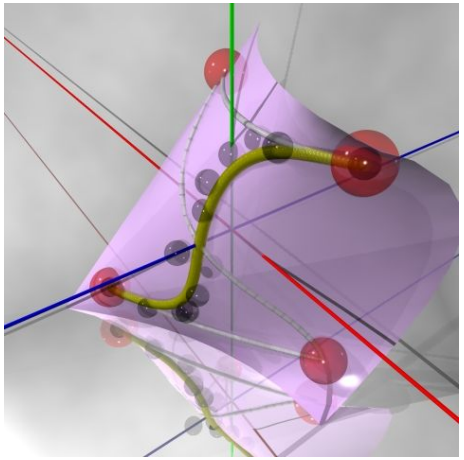
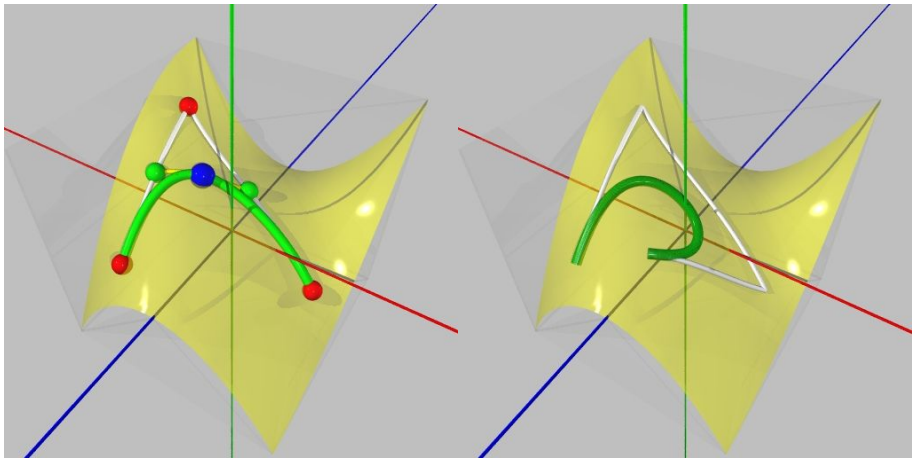


figure 2236.3 : une ipL4 dans une pS33 est une pL13 de l'espace.



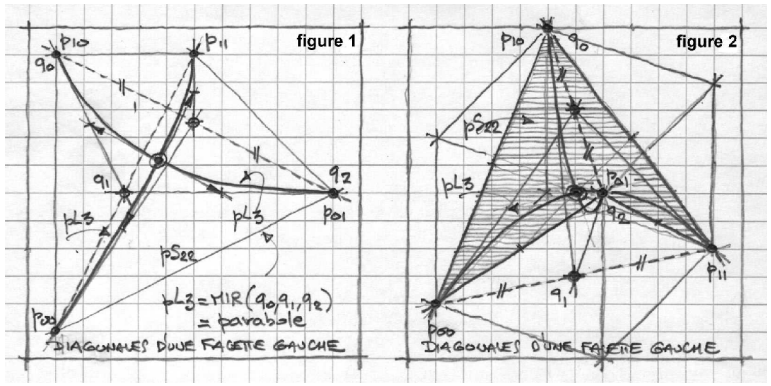
figures 2236.4 et 2236.5 : une parabole et une cubique dans une biquadrique.

23 interface

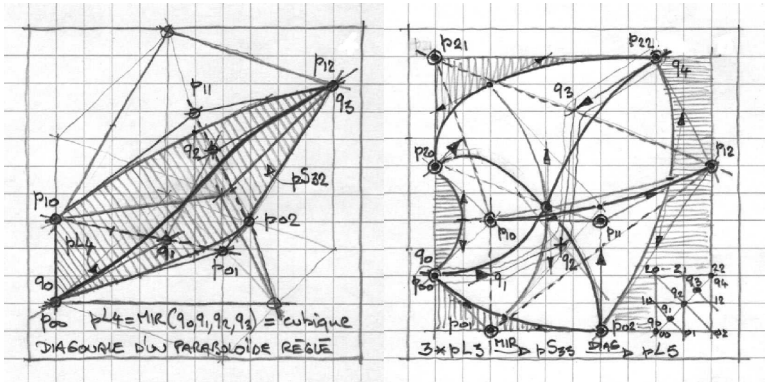
Au sommaire de cette section :

Une fois définies les pFormes, vient le temps de les manipuler et de les visualiser. On peut les dessiner à la main - après tout c'était le but initial, pouvoir dessiner à main levée à l'aide d'une corde et de quelques piquets - et quelques exemples de constructions à la main sur papier quadrillé en sont données dans les figures 23.1 à 23.4.

On peut aussi utiliser la bibliothèque pFlibs.inc écrite dans l'environnement POVRAY, conçue pour celà. On a déjà utilisé quelques opérateurs (des macros) comme pFsubdivision(), pFstretch(), etc... et l'opérateur indispensable draw() qui affiche tout ce qu'on a pu imaginer, y compris ce que l'on n'attendait pas... Les transformations (translation, rotation, homothétie) et les particularités de l'opérateur d'affichage sont analysées ci-après.



figures 23.1 et 23.2: les pFormes sont AUSSI constructibles à la main : diagonales et paraboles dans une facette gauche.



figures 23.3 et 23.4 : les pFormes sont AUSSI constructibles à la main : paraboloïde réglé et biquadrique assortis de leurs diagonales...

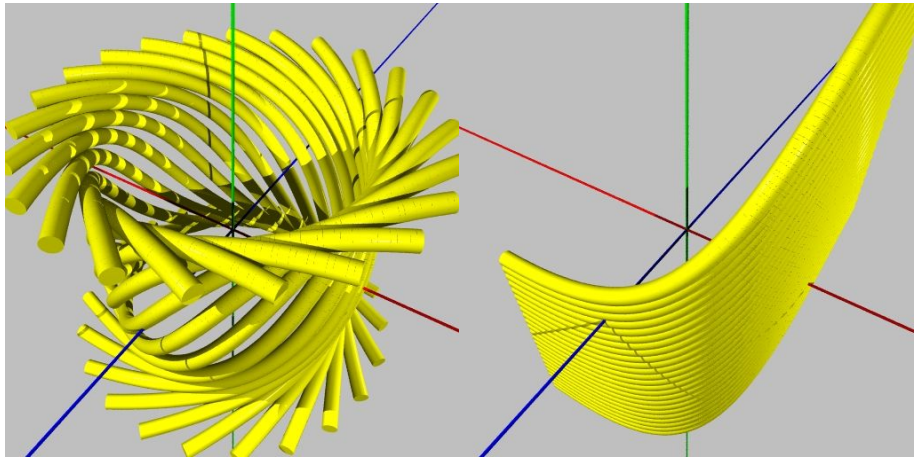
231 transformations

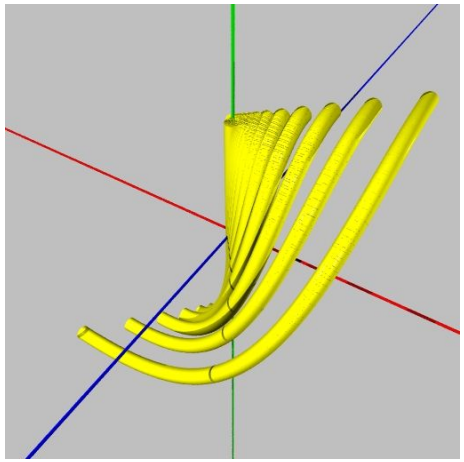
Une pForme étant définie, il est souvent nécessaire de lui appliquer une translation, une rotation, un changement d'échelle (scale, homothétie), une déformation par glissement (shear), ou une transformation affine plus générale comme celle utilisée dans le cas des tubages étudiés plus loin dans le texte. Concernant la transformation la plus générale, on définit un opérateur pFtransform() appliquant une matrice 4x4 à un point de R4 ; cet opérateur est à usage purement interne, pour construire par exemple les opérateurs produisant des formes de révolution, des tubages, etc…. La syntaxe d'appel est la suivante :

```
#local p1 = pFtransform( mat, p0 )
// l'opérateur retourne p1 transformé de p0
```

Les opérateurs classiquement utilisés sont pFtranslate(), pFrotate() et pFscale() qui font ce qu'on en attend, c'est–à dire transforment des pFormes dans notre espace R3 par translations, rotations et homothéties. Les appels de ces opérateurs s'écrivent ainsi :

```
// attention : les trois opérateurs modifient pForm :
// translation d'une pForme dans R3:
pFtranslate( dim, pForm, < tx, ty, tz > )
// rotation par rapport aux trois axes Ox,Oy,Oz:
pFrotate( dim, pForm, < rx, ry, rz > )
// homothétie par rapport aux trois axes Ox,Oy,Oz:
pFscale( dim, pForm, < sx, sy, sz > )
```





figures 231.1 à 231.3 : rotations, translations, homothéties appliquées à une pL4 (cubique).

232 représentation

En général après avoir défini une pForme, puis l'avoir étirée, transformée et enfin subdivisée, on ressent l'irrésistible envie de la visualiser… Mais on ne représente pas de la même manière des points, des courbes, des surfaces, des volumes et a fortiori des hypervolumes! De plus les représentations ne sont pas forcément uniques pour une forme donnée: un volume peut être représenté par l'ensemble des points qui le composent, par les surfaces limites, par un empilement de surfaces (mille–feuilles), par un faisceau de fibres (normales aux feuilles du mille–feuilles), par une distribution de repères tangents (tu,tv,n), par des courbes caractéristiques comme les courbes coordonnées curvilignes, des diagonales, etc…

On distinguera deux formes de représentations, une forme générale valable pour toute pForme, et des formes spécifiques adaptées aux pCourbes, aux pSurfaces et aux pVolumes.

2321 forme fondamentale

Une pForme n'est au fond qu'un nuage de points, un tableau de tableaux de points, et la représentation la plus élémentaire peut naturellement prendre la forme d'un nuage de petites sphères. On définit pour celà un opérateur pFdraw() dont la syntaxe d'appel est la suivante :

```
pFdraw( dim, pForm, R )  // avec dimension dim, rayon R qq
```

POVRAY affichera un nuage de sphères noires, ce qui est un peu triste ; pratiquement, on appliquera à cet ensemble de sphères une couleur ou une matière, et l'appel suivant produira ainsi les 4 sommets d'une facette gauche pS22 (ou plus si cette facette a été subdivisée) sous la forme de sphère rouges de rayon 0.02.

```
union { pFdraw( 2, pS22, 0.02) une_couleur( < 1,0,0,0 > ) }
```

Noter que pFdraw() s'applique à toute pForme quelle que soit sa dimension, y compris donc aux hypervolumes !

2322 formes spécifiques

Pour les courbes, les surfaces, les volumes et même les hypervolumes, une forme plus adaptée de l'opérateur pFdraw() est proposée: un nouvel opérateur draw() (sans le préfixe pF pour en souligner le caractère spécifique) simplifie et concentre l'écriture des divers paramètres et fait appel à un certain nombre d'options décrites dans le tableau suivant :

```
draw( dim, pForme, options )
      avec:
      dim : un nombre réel dans l'intervalle [0,4]
      pForme : une pForme quelconque
      options : somme partielle d'un ensemble des choix suivants :


STANDARD : valeurs de tracé par défaut, sphère rouge de rayon 0.02

finesse( recursion )  choisit le niveau de subdivision de la forme
où recursion = r | < rx,ry > | < rx,ry,rz > | < rx,ry,rz,rt >

point( R )          trace des sphères de rayon R

courbe( R )         trace des cylindres de rayon R

surface( FACETTES | LISSE | SUPER )
      trace un maillage de triangles
      aux arêtes marquées ou lissées
      avec une interpolation des couleurs +ou- fine

enveloppe( FACETTES | LISSE | SUPER )
      trace les 6 faces d'un volume

feuilles( FACETTES | LISSE | SUPER )
      trace un volume en mille-feuilles

fibres( rayon )
      trace les fibres d'un volume

normales( rayon, longueur )
      trace les normales à une surface

repere_tangent( rayon, longueur )
      trace les repères tangents à une courbe ou à une surface

couleur( r,g,b,t  )       choisit une couleur et une opacité

matiere( choix )          choisit une matière :
      choix = GOLD | MIROIR | GRANIT | MARBRE
```

Des exemples d'utilisation de cet opérateur et des choix sont donnés un peu partout dans ce document.

Le plus souvent, on tracera une courbe, une surface et un volume en utilisant les appels suivants :

```
draw( 1, pCourbe,   finesse(5) +        // niveau récursion 5
                    courbe( 0.02 ) +          // ligne rayon 0.02
                    ma_couleur(<1,0,0,0>) )   // rouge

draw( 2, pSurface,  finesse(<4,4>) +    // 4 sur U et V
                    surface(LISSE) +    // lissage des triangles
                    ma_couleur(<0,1,0,0>) )   // jaune

draw( 3, pVolume,   finesse(<2,2,2>) +  // 2 sur U, V, W
                    enveloppe(LISSE) + // les 6 faces du cube
                    ma_couleur(<1,1,0,0.5>) ) // jaune transparent
```

D'autres options sont envisageables en fonction des besoins: la représentation des hypervolumes pourrait évoluer vers des combinaisons plus parlantes de feuilles et de fibres, ou intégrer une gestion du temps sous la forme d'animations que POVRAY sait gérer sans problème.

3 compositions, applications

Au sommaire de cette section :

Au terme d'une première approche, le chapitre 1 nous a permis de définir de façon générale les formes pascaliennes, et le chapitre 2 nous en a présenté les propriétés fondamentales, sans présumer de toute position particulière des points de contrôle. Ce chapitre nous fera découvrir des agencements particuliers des points de contrôle reproduisant les formes classiques de la géométrie - les cercles, les surfaces de révolution,... -, des opérateurs combinant plusieurs pFormes pour produire des tubages, des surfaces de Coons, des surfaces parallèles, des concaténations utiles engendrant des courbes et des surfaces splines, et nous permettra d'aborder discrètement sur le continent merveilleux de la géométrie des espaces courbes... :)

31 formes rationnelles

Au sommaire de cette section :

- 311 coniques
- 312 cônes, cylindres, tores et sphères
- 313 applications
    - 3131 la fenêtre de Viviani
    - 3132 des cercles immergés
    - 3133 des noeuds
    - 3134 des géodésiques

Il a déjà été dit au paragraphe 111 que les points étaient définis dans R^4 sous la forme dite des coordonnées homogènes ou forme projective: < x,y,z,t >, associant au point de R^4 le point de R^3 défini par < x/t,y/t,z/t >. Jusqu'à présent, nous avons toujours pris par défaut la valeur 1 pour la quatrième coordonnée; c'est le moment d'étudier les cas où t est différent de 1. Dans le cas où t varie (en principe t peut varier de -infini à +infini), à un point donné de R^4 va correspondre un ensemble de points de R^3 définis par < x/t,y/t,z/t >, et à une courbe définie dans R^4 va correspondre un ensemble de courbes de R^3, (noter que l'on peut ainsi construire une surface dans R^3 à partir de la donnée d'une seule courbe dans R^4 !).

Une pCourbe s'exprime toujours algébriquement sous forme polynomiale, le point courant d'une pL3 (arc de parabole) s'exprimant ainsi en fonction des 3 points de contrôle :

```
p = (1-u)^2.p0 + 2.(1-u).u.p1 + u^2.p2    // u dans [0,1]
```

ou de façon explicite, en détaillant les 4 coordonnées des points en (x, y, z ,t) :

```
x = (1-u)^2.x0 + 2.(1-u).u.x1 + u^2.x2
y = (1-u)^2.y0 + 2.(1-u).u.y1 + u^2.y2
z = (1-u)^2.z0 + 2.(1-u).u.z1 + u^2.z2
t = (1-u)^2.t0 + 2.(1-u).u.t1 + u^2.t2
```

Les coordonnées du point de R^3 définies par < x/t,y/t,z/t > seront des expressions "rationnelles", c'est à dire des quotients (ratios) de polynomes ; on parle alors de courbes rationnelles, et au delà de formes rationnelles. Les formes pascaliennes, toujours définies dans R^4 produisent dans R^3 des formes rationnelles dans le cas général, se réduisant à des formes polynomiales dans le cas où les points de contrôle possèdent tous la valeur 1 pour la quatrième coordonnée.

Mais à quoi peuvent donc servir les formes rationnelles ?

311 coniques

Si l'on s'en tient à l'espace R^3, une pCourbe ne peut en aucun cas repésenter un arc de cercle, ce qui est bien dommage quand on connaît l'importance de ce type de courbe et des surfaces construites sur des cercles comme les cônes, les cylindres, les tores, les sphères et au-delà toutes les surfaces de révolution !

Mais la "Théorie des Coniques" que les Grecs nous ont révélée met en évidence les relations fondamentales entre des courbes aussi différentes en apparence qu'un cercle, une hyperbole, une ellipse et une parabole: toutes ces courbes sont des coniques, des sections d'un cône à base circulaire par un plan plus ou moins incliné par rapport à l'axe du cône. Vues du sommet du cône, elles sont toutes identiques et, notamment, le cercle de base peut toujours être considéré comme projection "conique" de l'intersection parabolique du cône par un plan parallèle à une génératrice.

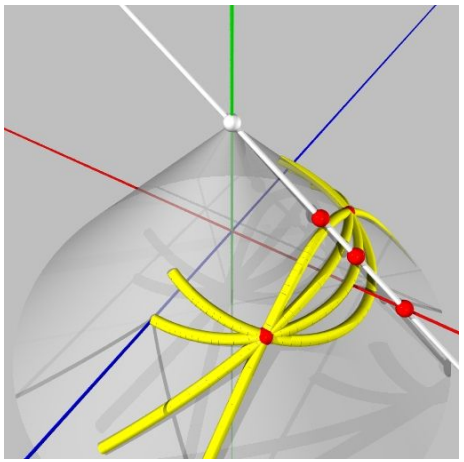


figure 3111.1 : les 4 coniques comme projections d'une pL3.

Pour comprendre ce résultat fondamental, rappelons l'expression analytique dans R^3 d'un cercle de rayon 1 centré sur l'origine dans le plan Oxy :

```
P(theta) = [ cos( theta ), sin( theta ), 0 ], theta = [-pi,pi]
```

Le changement de variable u = tangente(theta/2) transforme cette expression en une forme rationnelle :

```
P(u)     = [ (1-u^2)/(1+u^2), 2u/(1+u^2), 0 ], u = ]-infini,+infini[
```

et le passage dans R^4 la transformera en une expression polynomiale :

```
P(u)      = [ 1-u^2, 2u, 0, 1+u^2 ]
```

les quatre polynomes étant du second ordre dont deux dégénérés, chacun pouvant être exprimés sous la forme polynomiale définissant une parabole dans R^4 :

```
P(u)      = (1-u)^2.p0 + 2.(1-u).u.p1 + u^2.p2
          avec : p0 = < 1, 0, 0, 1>
                 p1 = < 1, 1, 0, 1> * sqrt(2)/2 // p1.t ≠ 1
                 p2 = < 0, 1, 0, 1>
```

Ainsi, s'il reste vrai qu'une pCourbe ne peut pas représenter un arc de cercle, elle peut naturellement en générer un par projection conique, ce qu'est en fait le passage de R^4 à R^3, une simple *perspective*. On peut ainsi raisonner dans R^4 sur des paraboles en sachant que le retour dans R^3 nous donnera bien des cercles. Nous constaterons que ce raisonnement est encore valable pour les pFormes immergées.

Dans la syntaxe POVRAY/pFlibs, nous définirons donc un arc de parabole ainsi :

```
#local r = une valeur quelconque;
#local k = sqrt(2)/2;
#local quart_cercle = array[3] { < r, 0, 0, 1 >,
                                 < r, r, 0, 1 >*k,
                                 < 0, r, 0, 1 > }
```

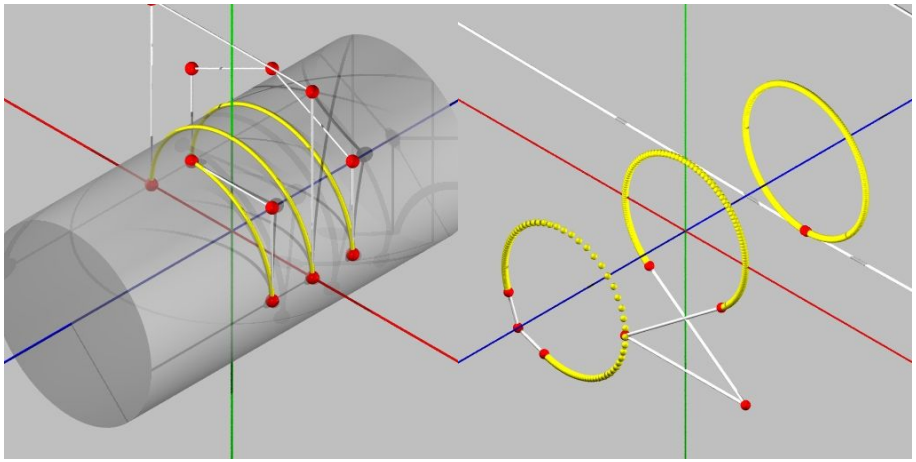
La projection dans R^3 de cet arc de parabole de R^4 produit un arc de cercle parfait, centré à l'origine, de rayon r et d'angle 90°. D'autres valeurs de k produisent des ellipses et des hyperboles. Noter que l'implémentation faite dans la syntaxe de POVRAY/pFlibs applique toujours par défaut la projection R^4 -> R^3, et il n'est pas nécessaire de s'en occuper. L'appel suivant affichera bien l'arc de cercle rouge attendu:

```
draw(1,quart_cercle,finesse(3)+courbe(0.02)+couleur(< 1,0,0,0 >))
```

L'arc de parabole se projette sur un arc de cercle, mais on peut imaginer d'autres courbes du cône dont la projection sera également un arc de cercle, en fait n'importe quelle courbe d'un cone convient. En voici deux exemples utiles, deux pCourbes définies par 4 et 5 points de contrôle :

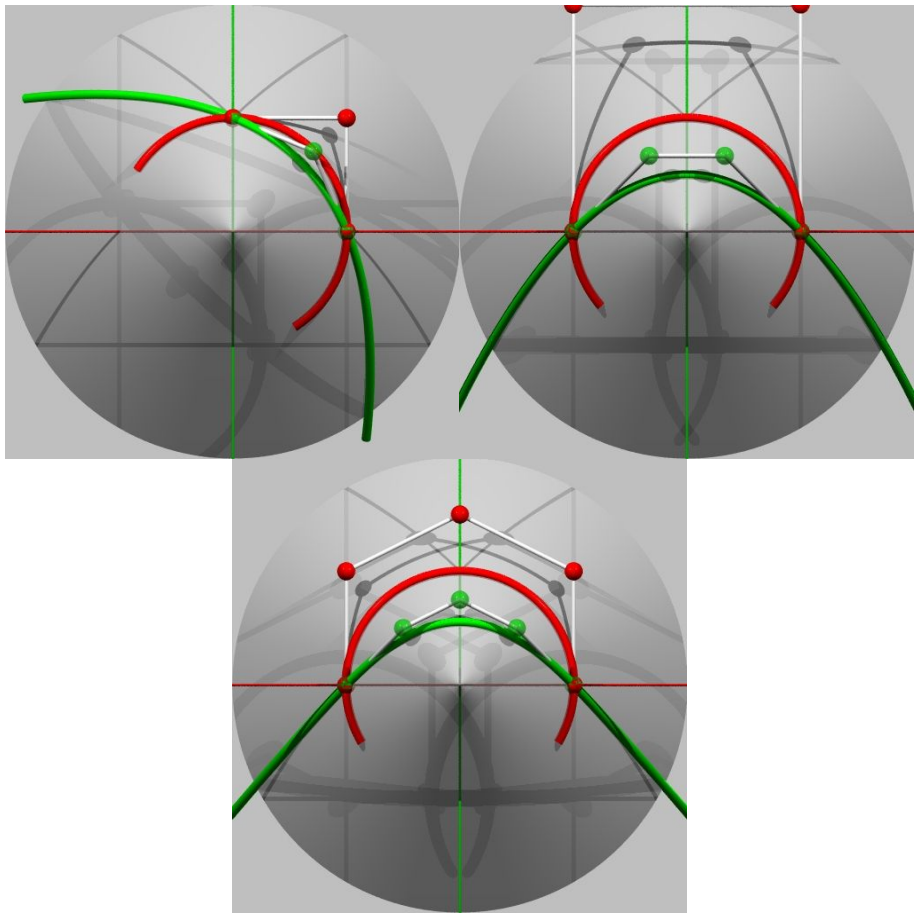
```
#local demi_cercle = array[4] {
                                < r,   0, 0, 1 >,
                                < r, 2*r, 0, 1 >/3,
                                < -r, 2*r, 0, 1 >/3,
                                < -r,   0, 0, 1 >    }

#local k = sqrt(2)/2;
```

```
#local demi_cercle = array[5] {
                                  <  r,     0, 0, 1 >,
                                  <  r,     r, 0, 1 >*k,
                                  <  0, 3/2*r, 0, 1 >*2/3,
                                  < -r,     r, 0, 1 >*k,
                                  < -r,     0, 0, 1 >
                                }
```



figures 3111.2 et 3111.3 : 3 arcs de cercle à partir d'une pL3, d'une pL4 et d'une pL5 ; 3 approches du cercle complet avec une pL3, une pL4 et une pL5.

Ces deux représentations peuvent produire un arc de cercle complet en étendant l'intervalle de définition à l'aide de l'opérateur pFstretch(). Dans le premier cas (une pL3), il est nécessaire de travailler sur l'intervalle [-infini,+infini], ce qui ne produit rien de bon, dans le dernier cas (une pL5), un intervalle [-k,1+k] avec k=sqrt(2)/2, est suffisant pour produire le cercle complet avec une bonne répartition des points ; il se trouve de plus que cette dernière représentation correspond à la diagonale d'une biquadrique, surface construite à l'aide de trois paraboles, ... vous suivez ?



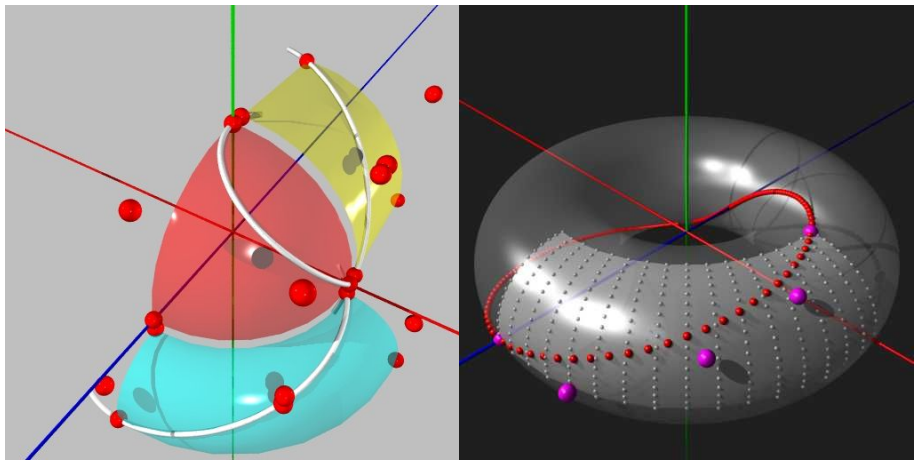
figures 3111.4 à 3111.6 : pL3, pL4 et pL5 dans R^4 en vert, et les arcs de cercle correspondants dans R^3 en rouge.

312 cônes, cylindres, tores et sphères

Une fois incorporé l'arc de cercle dans la famille des pCourbes, le monde des pFormes de révolution nous est ouvert ! Nous allons les construire en deux étapes: dans un premier temps, nous composons "à la main" les expressions de ces surfaces, et dans un second temps, nous rechercherons des opérateurs de composition plus généraux qui nous conduiront aux formes tubées et aux formes affines. Pour l'instant donc, en choisissant convenablement les points de définition d'une pSurface, il est facile de créer les surfaces de révolution fondamentales comme une portion de cylindre, de tore ou de sphère. En voici les expressions dans la syntaxe POVRAY/pFlibs :

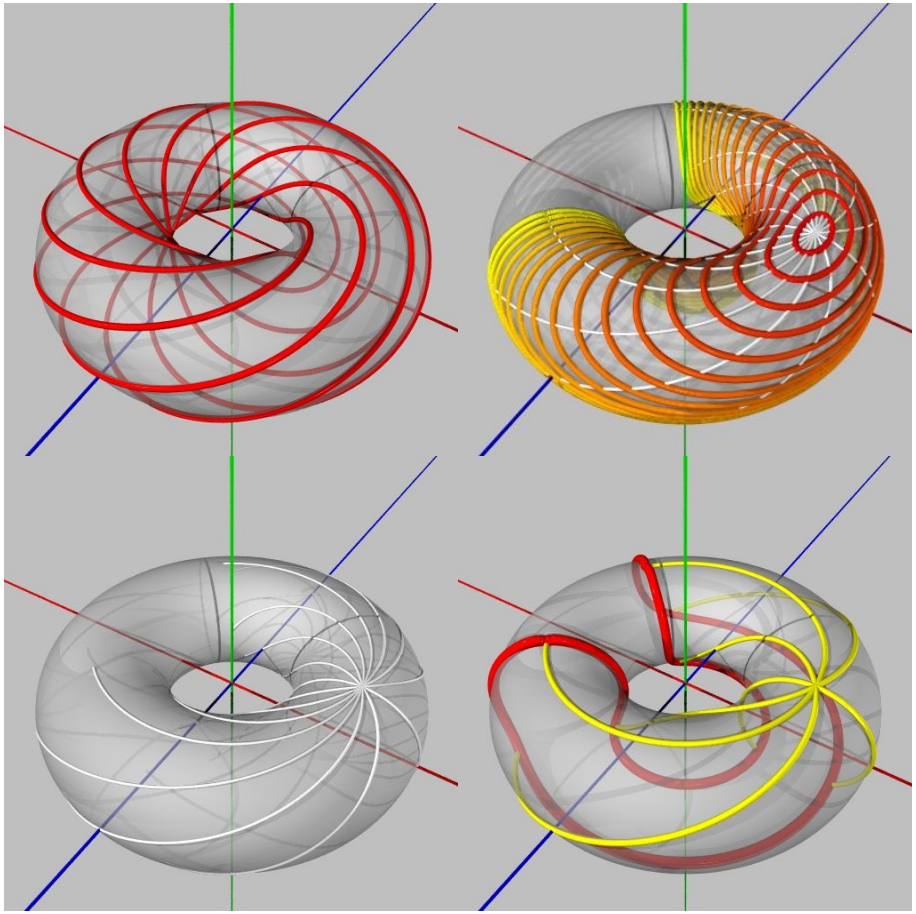
```
// un quart de cylindre de rayon R et de hauteur H:
#local k = sqrt(2)/2;
#local pCylindre = array[2]
{       array[3] {
                < R, 0, 0, 1 >,
                < R, R, 0, 1 >*k,
                < 0, R, 0, 1 >
                },
        array[3] {
                < R, 0, H, 1 >,
                < R, R, H, 1 >*k,
                < 0, R, H, 1 >
                }
}
// un seizième de tore de rayons R1 et R2:
#local R12 = R1+R2;
#local R2 = abs(R2);
#local pTore = array[3]
{       array[3] {
                < 0,   0, -R12, 1 >,
                < 0,  R2, -R12, 1 >*k,
                < 0,  R2,  -R1, 1 >
                },
        array[3] {
                < R12,  0, -R12, 1 >*k,
                < R12, R2, -R12, 1 >*k*k,
                <  R1, R2, -R1,  1 >*k
                },
        array[3] {
                < R12,  0,  0, 1 >,
                < R12, R2,  0, 1 >*k,
                <  R1, R2,  0, 1 >
                }
}
```

```
// une huitième de sphère de rayon R:
#local pSphere = array[3]
{     array[3] {
            < R, 0, 0, 1 >,
            < R, R, 0, 1 >*k,
            < 0, R, 0, 1 >
},
      array[3] {
            < R, 0, -R, 1 >*k,
            < R, R, -R, 1 >*k*k,
            < 0, R,  0, 1 >*k
            },
      array[3] {
            < 0, 0, -R, 1 >,
            < 0, R, -R, 1 >*k,
            < 0, R,  0, 1 >
            }
}
```



figures 312.1 et 312.2 : 3 pS33 cylindre, tore et sphère et leurs diagonales ; diagonale (pL5) sur 1/16ème de tore (pS33) tracée dans l'intervalle [-k,1+k], où k=sqrt(2)/2.

La encore il sera possible avec l'opérateur pFstretch() de créer des sphères, des cylindres et des tores complets (sur 360°), comme vu pour l'arc de cercle, en utilisant les bonnes pCourbes, la pL5 étant a priori la plus adaptée dans l'intervalle [-k,1+k].

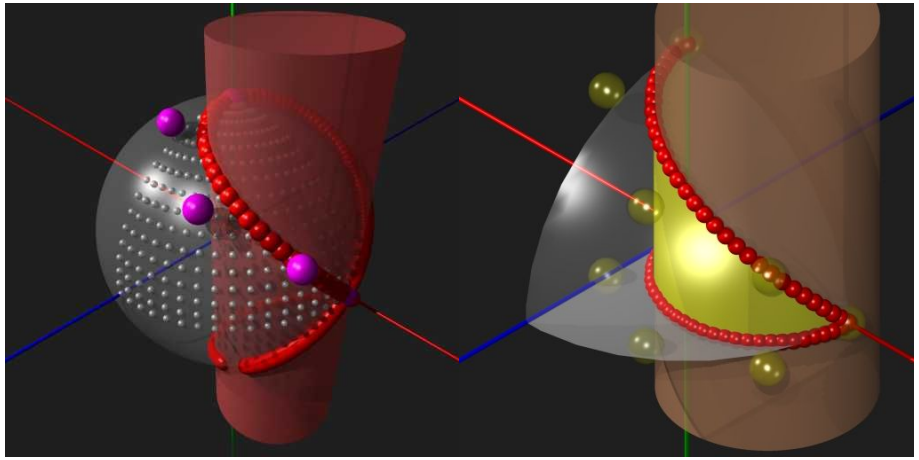


figures 312.3 à 312.7 : dans un tore, droites parallèles rouges, cercles concentriques du rouge au jaune, segments rayonnants blancs, et cercle rouge et ses rayons jaunes.

313 applications

La pS33 et sa diagonale pL5 jouent un rôle important dans de nombreuses applications. Avec la diagonale d'une portion de sphère construite sur une pS33, on peut ainsi retrouver la courbe appelée fenêtre de Viviani. En immergeant dans une sphère ou dans un tore une pL5 produisant un arc de cercle, on découvre un exemple de confinement de points partant à l'infini dans notre espace « droit ». Et la courbe connue sous le nome de Threefoil Knot (noeud à triple boucle) apparaît comme une simple ligne droite dans un « univers » torique.

3131 la fenêtre de Viviani



figures 3131.1 et 3131.2 : la fenêtre de Viviani est un segment immergé (pL5) dans 1/8ème de sphère (pS33) tracé dans l'intervalle [-k,1+k], où k=sqrt(2)/2.

La portion de sphère (1/8ème) est une pS33 dont on sait que la diagonale est une pL5. Il se trouve que la projection verticale sur le plan équatorial des cinq points de contrôle produit les points de contrôle d'un demi-cercle ; on retrouve ainsi la courbe gauche appelée fenêtre de Viviani, intersection d'une sphère et d'un cylindre. L'expression classique de la courbe est la suivante :

```
x = R*(1+cos(t)),
y = R*sin(t),
z = 2R*sin(t/2)
```

ce qui n'est pas compliqué, mais la définition du trièdre tangent en chaque point fait intervenir des expressions bien plus complexes. Considérer cette courbe comme une pCourbe donne accès à 5 points de contrôle qui aident à *dessiner* la courbe, ainsi qu'à tous les opérateurs des pFormes et notamment à celui qui produit le trièdre tangent en chaque point. De plus, savoir que cette courbe est un simple segment immergé dans une portion de sphère est intellectuellement agréable !

Voici dans la syntaxe de POVRAY/pFlibs trois méthodes utilisées pour construire la fenêtre de Viviani :

```
Définition du huitième de sphère:

#local R = 1/2;
#local k = sqrt(2)/2;
#local pSphere = array[3]  { array[3] {
                                        < R, 0, 0, 1 >,
                                        < R, R, 0, 1 >*k,
                                        < 0, R, 0, 1 >
                                      },
                             array[3] {
                                        < R, 0, -R, 1 >*k,
                                        < R, R, -R, 1 >*k*k,
                                        < 0, R,  0, 1 >*k
                                      },
                             array[3] {
                                        < 0, 0, -R, 1 >,
                                        < 0, R, -R, 1 >*k,
                                        < 0, R,  0, 1 >
                                      }
                           }
 draw( 2, pSphere, finesse < 3,3 > + surface( 0.01 ) + couleur( <
1,1,1,0.5 >) )

1) construction par la méthode d'immersion de la diagonale définie
et subdivisée dans la surface entre les points < 0,0 > et < 1,1 >,
puis exprimée dans R3 :

#local diag = array[2] { < 0,0,0,1 >, < 1,1,0,1 > }
#local pDiag = pFsubdivision( 1, diag, <4,0,0> )
pFimmersion( 2, pSphere, 1, pDiag )
draw( 1, pDiag, point(0.02) + couleur(< 1,1,1,0.8 >) )

2) construction par appel de l'opérateur de diagonalisation :

#local diag = pFdiagonalisation( 2, pSphere )
draw( 1, diag, finesse(3)+point(0.02)+couleur(< 1,0,0,0.9 >) )
draw( 1, diag, point(0.04) + couleur(< 1,1,0,0.8 >) )
```

```
3) construction par définition de la diagonale dans la surface
entre les points < 0,0 > et < 1,1 > , et appel de l'opérateur
général courbe_in_surface():

#local diag = array[2] { < 0,0,0,1 >, < 1,1,0,1 > }
#local diag = courbe_in_surface( diag, pSphere )
draw( 1, diag, finesse(3)+point(0.02)+couleur(< 1,0,0,0 >) )
draw( 1, diag, point(0.04+couleur(< 1,1,0,0.8 >) )
```

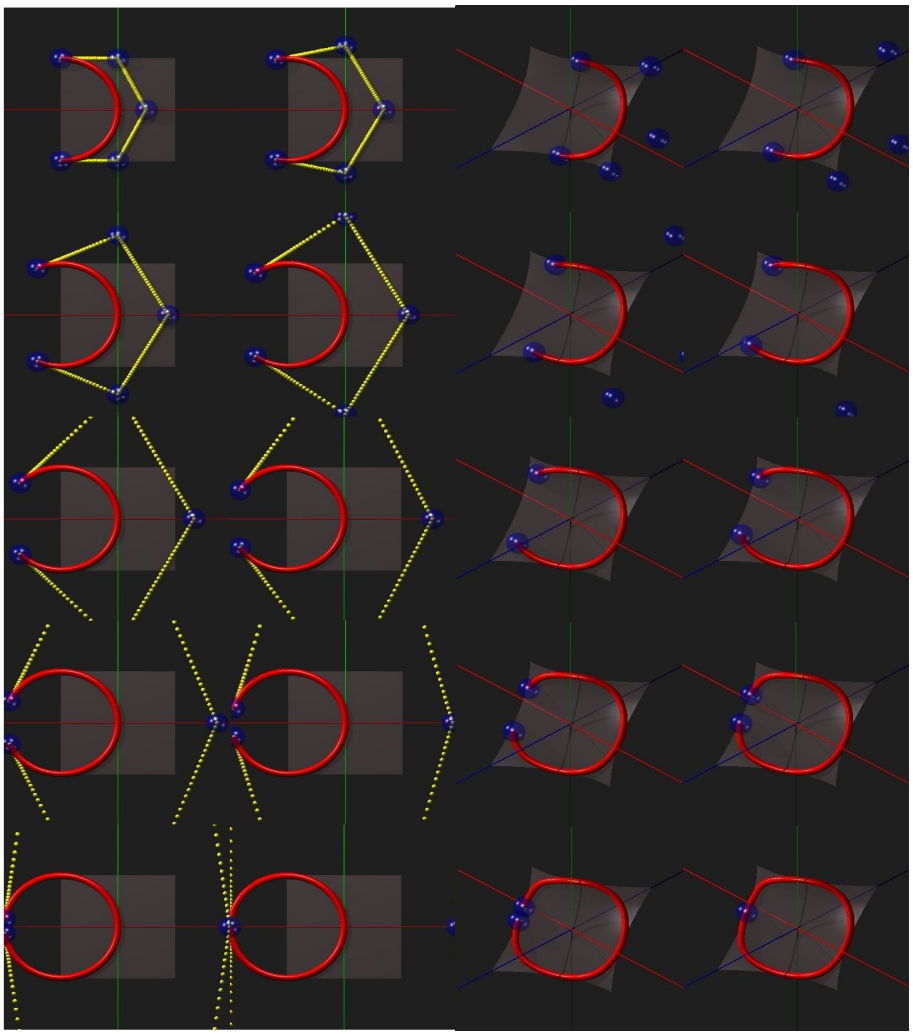
La première méthode de calcul par immersion est générale, elle est valable pour toute ipForme immergée dans toute pForme, mais elle ne permet pas de connaître les points de contrôle dans R3. La seconde méthode ne peut traiter que les diagonales, donc les iSegments, mais a l'avantage de produire les points de contrôle. La dernière méthode ne convient qu'aux pCourbes immergées dans des pSurfaces, mais produit les points de contrôle pour toute ipCourbe. Bien sûr, les trois méthodes fournissent la même courbe dans le cas présent. Noter qu'en applatissant la sphère, on obtient le disque projection dont la diagonale est bien un demi-cercle.

3132 des cercles immergés

On a vu qu'il est facile de placer les 5 points d'une pL5 pour construire un demi-cercle paramétré dans l'intervalle standard [0,1] . En utilisant l'opérateur pFstretch() avec l'intervalle de définition [-k,1+k], où k = sqrt(2)/2 = 0.707, on produit un cercle complet avec une distribution acceptable des points. Le problème est que deux des 5 points de contrôle partent à l'infini, et qu'il n'est jamais agréable de travailler avec des points à l'infini. La figure 3132.1 représente le cas où l'arc de cercle est immergé dans une facette plane, en fait dans l'espace euclidien englobant ; dans la figure 3132.2 la facette est une pSurface quelconque légèrement gauchie, ce qui ne change pas grand chose.

En immergeant un cercle dans une surface finie, on confine ces points et on peut ainsi visualiser leur comportement plus efficacement. L'immersion de la pL5 générant un demi-cercle dans une pS33 générant une sphère ou un tore permet de représenter proprement les deux points partant initialement à l'infini dans le passage de l'intervalle [0,1] à l'intervalle [-k,1+k]. Les figures 3132.3 et 3132.4 correspondent à une immersion dans des pS55, dont les 25 points de contrôle sont choisis pour produire une sphère et un tore, des surfaces finies et complètes dans lesquelles les points de contrôle de l'arc de cercle immergé se développant vers le cercle complet sont confinés à distance finie.

On peut entrevoir une similitude avec la sphère de Riemann comme représentation d'un espace projectif, on pense aux droites de Poincaré dans le plan hyperbolique, et plus généralement aux tentatives de représentation de l'univers supposé de topologie elliptique, parabolique, hyperbolique, et ces temps-ci, toroïdale. Et à l'utilisation des pFormes immergées dans la représentation des théories cosmologiques. Une piste à explorer !



figures 3132.1 et 3132.2 : dans le plan ou dans une pSurface gauche quelconque, deux des points de contrôle d'un arc de cercle défini par une pL5 partent à l'infini.

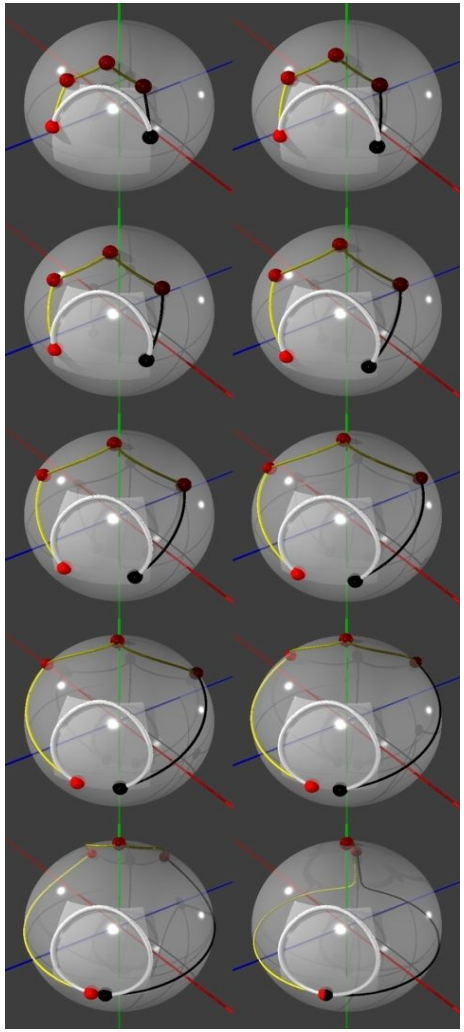


figure 3132.3 : un demi cercle blanc (pL5) et son polygone de contrôle (segments du noir au jaune) immergés dans une sphère, l'intervalle de définition est progressivement étendu de [0,1] à [k, 1+k] où k=sqrt(2/2, le demi-cercle devient un cercle complet, les points de contrôle restent confinés dans la surface de la sphère.

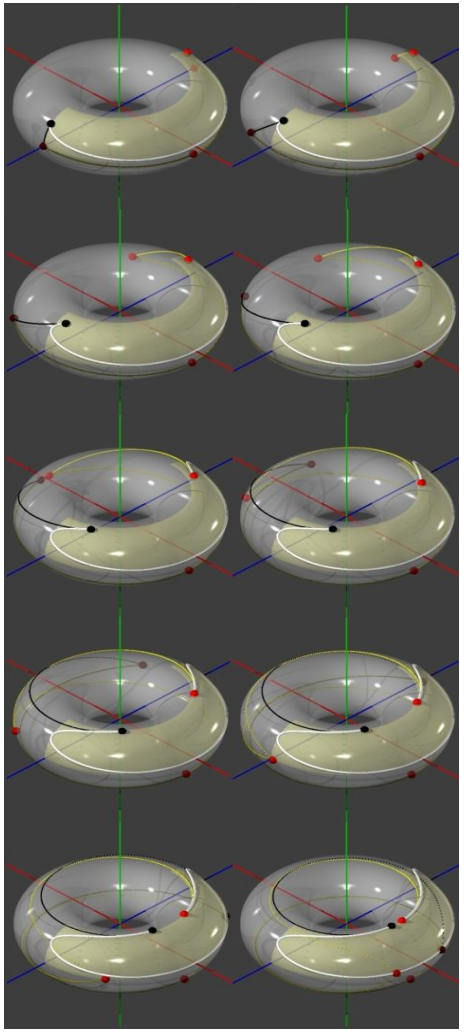
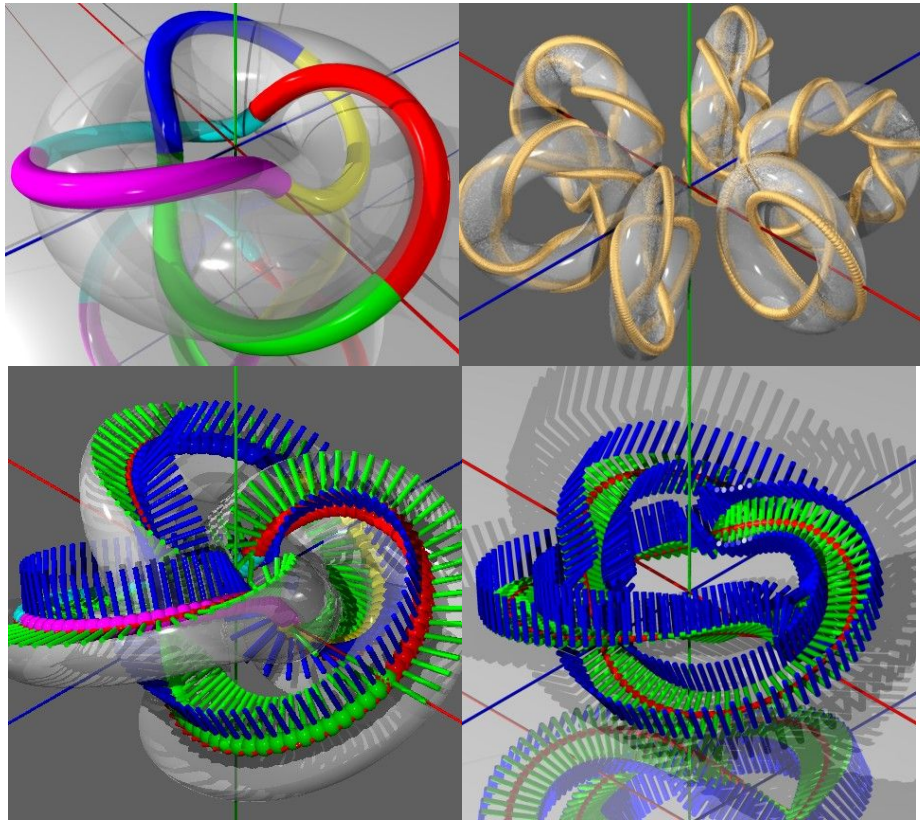


figure 3132.4 : même exemple avec une immersion dans un tore.

3133 des droites qui se font des noeuds

Les escaliers de Montréal sont connus pour être assez étonnants, placés en façade de petits collectifs à deux ou trois appartements empilés, se tordant dans tous les sens pour desservir chaque niveau en évitant de trop géner les baies donnant sur la rue, faisant penser aux escaliers impossibles de Marc Escher. La sculpture représentée figure 3133.1 est le résultat d'un concours gagné par un architecte Montréalais, Guillaume Labelle (site: http://www.grcao.umontreal.ca/goo/), sur l'idée de réaliser un escalier suivant une courbe connue sous le nom Threefoil Knot (noeud à triple boucle). Si l'expression analytique de cette courbe reste assez simple, la détermination des repères tangents (Serret-Frenet) nécessaires à la détermination des marches et des garde-corps fait appel à des expressions qui deviennent assez lourdes et peu parlantes. Les segments immergés dans un tore sont une solution agréable pour l'analyse, la manipulation et la représentation de noeuds divers tracés sur le tore. Les illustrations suivantes en montrent quelques étapes qui ont semble-t'il été utiles pour la détermination des marches et des garde-corps de l'escalier réalisé.

figure 3133.1 : une sculpture à Montréal, une ligne droite dans un espace torique !!



figures 3133.2 à 3133.5 : la courbe complète est la concaténation de 6 diagonales ; 6 exemples de noeuds à partir de segments immergés sous des angles différents (produisant des courbes continues ou non) ; application à l'escalier de Montréal en représentant une distribution de trièdres de Serret-Frenet, puis les binormales (esquissant les marches) et les normales (esquissant les garde-corps). Noter que les garde-corps effectivement réalisés sont inclinés de 45° pour être orthogonaux aux marches, parallèles aux contre-marches.

32 formes composées

Au sommaire de cette section :

Jusqu'à présent les pFormes ont été construites "à la main", les points de contrôle étant positionnés a priori ; c'est notamment le cas des portions de cylindre, de tore et de sphère vues plus haut. On va chercher à en automatiser le processus de création, ce qui donnera accès à des formes plus complexes.

321 maillages

Dans le chapitre sur l'approche des pFormes, nous avons eu l'occasion de construire "à la main" les premières pFormes: en appliquant l'opérateur MIR() à trois pL3 (paraboles), nous avons créé une pS33, puis des pL5, et aussi des pV222 (cubes gauches), etc... Il est utile de définir des opérateurs produisant automatiquement des pCourbes, des pSurfaces et des pVolumes contrôlées par un plus grand nombre de points: on trouvera un opérateur creer_ligne() produisant une pCourbe au départ rectiligne et contrôlée par un nombre arbitraire de points, un opérateur creer_facette() produisant une pSurface au départ plane et enfin un opérateur creer_cube() produisant un pVolume au départ parfaitement cubique, les trois pFormes produites étant de taille donnée et centrés à l'origine parallèles aux axes. En voici un exemple implémenté dans POVRAY/pFlibs dans le cas du cube :

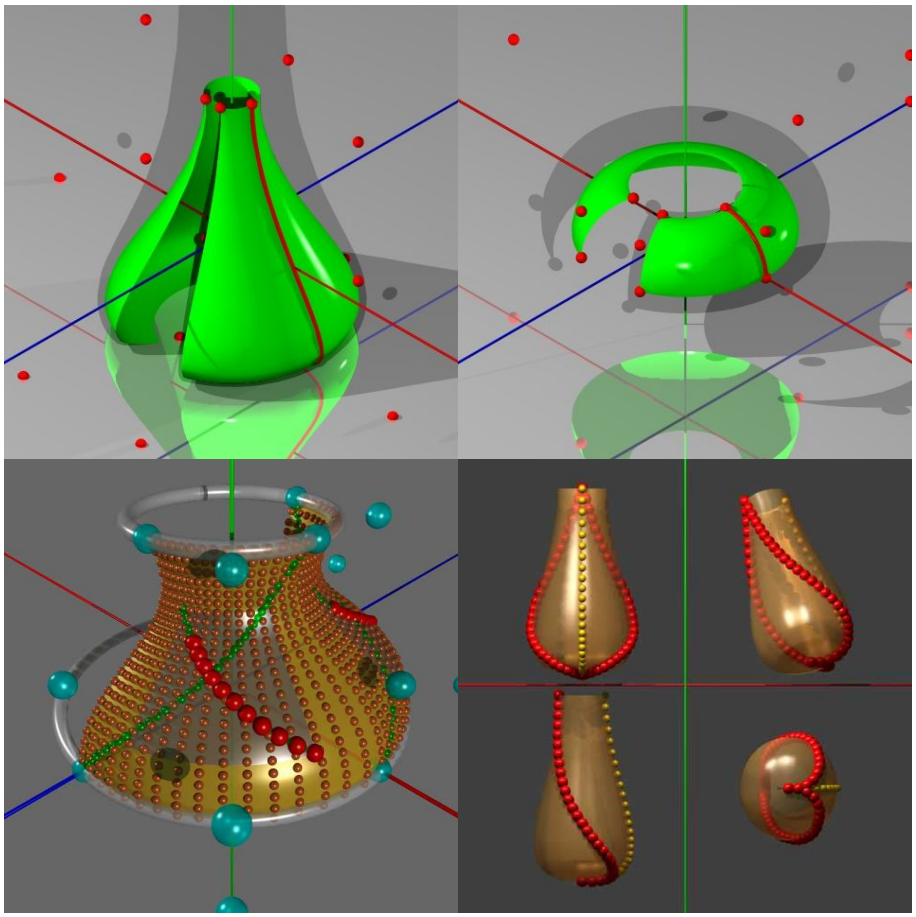
```
#macro creer_cube( n1, n2, n3, ttt )  // ttt est la taille
  #local f = array[n1]
  #local ff = array[n2]
  #local fff = array[n3]
  #local i=0; #while (i< n1)
    #local j=0; #while (j< n2)
      #local k=0; #while (k< n3)
        #local p = < -0.5+i/(n1-1),
                     -0.5+j/(n2-1),
                     -0.5+k/(n3-1),1 >;
        #local fff[k] = p * ;
      #local k=k+1; #end
      #local ff[j] = fff;
    #local j=j+1; #end
    #local f[i] = ff
  #local i=i+1; #end
  f
#end
```

Une application parmi d'autres d'un cube gauche au delà du très basique pV222 sera exposée au paragraphe sur les déformations.

322 surfaces produits, surfaces de révolution

On peut définir un opérateur produisant des pSurfaces à partir d'une courbe "section" définie dans un plan Oxz, déplacée suivant l'axe Oy selon une courbe "profil". Dans le cas où la courbe section est un arc de cercle, on produit une surface de révolution avec commes cas particuliers les cylindres, tores et sphères vus plus haut. En voici une implémentation dans la syntaxe POVRAY/pFlibs :

```
/*
  SURFACE PRODUIT:
  input: deux pLn planes dans Oxy, profil et section
  output: une pSmn
  nota: cas particuliers: prismes et surfaces de révolution
        profil gauche à etudier
  appel: #local surf = cross( c1, c2 )
*/
#macro cross( c1, c2 )
    #local nb1 = taille( c1 );    // courbes profil
    #local nb2 = taille( c2 );    // courbe section
    #local surf = array[nb1]      // surface produite
    #local i = 0; #while (i< nb1) // pour chaque point du profil
        #local profil = c1[i];    // un point du profil
        #local pp = array[nb2]
        #local j=0; #while (j< nb2)// pour chaque pt  section
            #local section = c2[j];// un pt de la section
        #local pp[j] = <
              section.x*profil.x,    // base sur x
              section.t*profil.y,  // le long de l'axe y
              section.y*profil.x,  // base sur z
              section.t*profil.t  >  ;
        #local j=j+1; #end
        #local surf[i] = pp
    #local i=i+1; #end
    surf
#end
```



figures 322.1 à 322.4 : quelques surfaces de révolution, et divers êtres géométriques tracés dessus.

Noter que dans le cas d'une section et d'un profil non rationnels (dont tous les points ont une valeur de t égale à 1), l'expression du point courant devient :

```
#local pp[j] = < section.x*profil.x,   // base sur x
                 profil.y,             // le long de Oy
                section.y*profil.x,    // base sur z
                1 >;                   // toujours
```

expression dans laquelle on reconnait pour les coordonnées en (x,z) du point résultant l'application d'une homothétie de rapport profil.x au point courant (x,y) de la section, et pour la coordonnée en y du point courant l'application de la valeur profil.y du point courant de la section ; la courbe section se déplace bien sur l'axe Oy en suivant la courbe profil.

Bien sûr, l'un des grands avantages des pSurfaces est qu'elles SONT des pSurfaces, c'est à dire qu'il est possible de les déformer par l'intermédiaire de leurs pCourbes et points générateurs. C'est la porte ouverte à la création de tout un champ de formes plus libres basées sur des sections initialement coniques, on pense aux différentes sections de la carlingue d'un avion (un Airbus par exemple), depuis le nez jusqu'à à la queue, cercles parfaits, en passant par le cockpit en forme de poire et les élargissements au droit des ailes et de l'empennage arrière, le tout sans aucune discontinuité, avec une même courbe dont seuls les points de contôle varient suivant le contexte. L'illustration ci-dessous montre un exemple plus modeste, une poterie faite au tour dont la tête a été déformée à la main par le potier pour constituer un bec verseur.

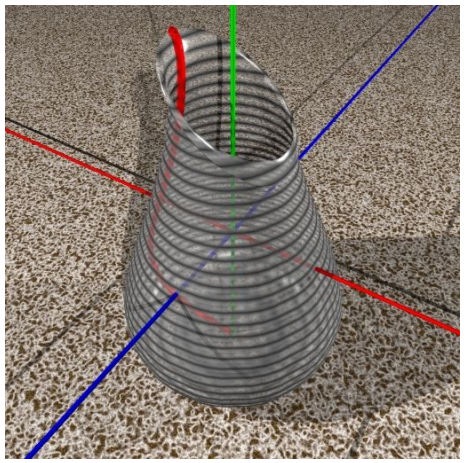


figure 322.5 : une surface de révolution déformée en tête.

323 surfaces tubulaires

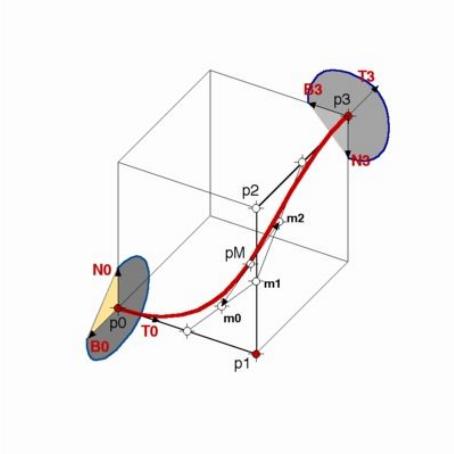


figure 323.1 : une surface tubulaire est la surface enveloppe d'une courbe section calée sur le trièdre de Serret-Frenet de la courbe chemin qu'elle parcourt.

Etant données une courbe appelée "chemin" et une courbe appelée "section", il est assez facile de créer une surface tubulaire en utilisant le trièdre de Serret-Frenet de la courbe chemin comme opérateur de transformation (translation + rotation) des points de la courbe section. On peut écrire symboliquement le tube produit sous la forme -cf schéma 323.1- :

```
tube = chemin + TNB(chemin) * section
```

où TNB est une matrice 3x3 construite sur les vecteurs tangente, normale, binormale.

Il faut cependant noter un problème. Jusqu'à présent, les combinaisons étudiées produisaient des pSurfaces ; un tore construit en utilisant les formules données dans la section "surfaces de révolution" est un ensemble de 9 points disposés dans l'espace de telle sorte que l'opérateur MIR() produise une surface parfaitement torique. Tenter de construire le tore en distribuant les 3 points de contrôle d'un arc de cercle section sur les 3 points de contrôle d'un arc de cercle chemin en fonction des 3 repères locaux (trièdres de Serret-Frenet) en chacun de ces points ne produit pas le résultat cherché, les 3 points de contrôle intermédiaires ne sont pas correctement placés, et l'opérateur MIR() ne produira pas un tore.

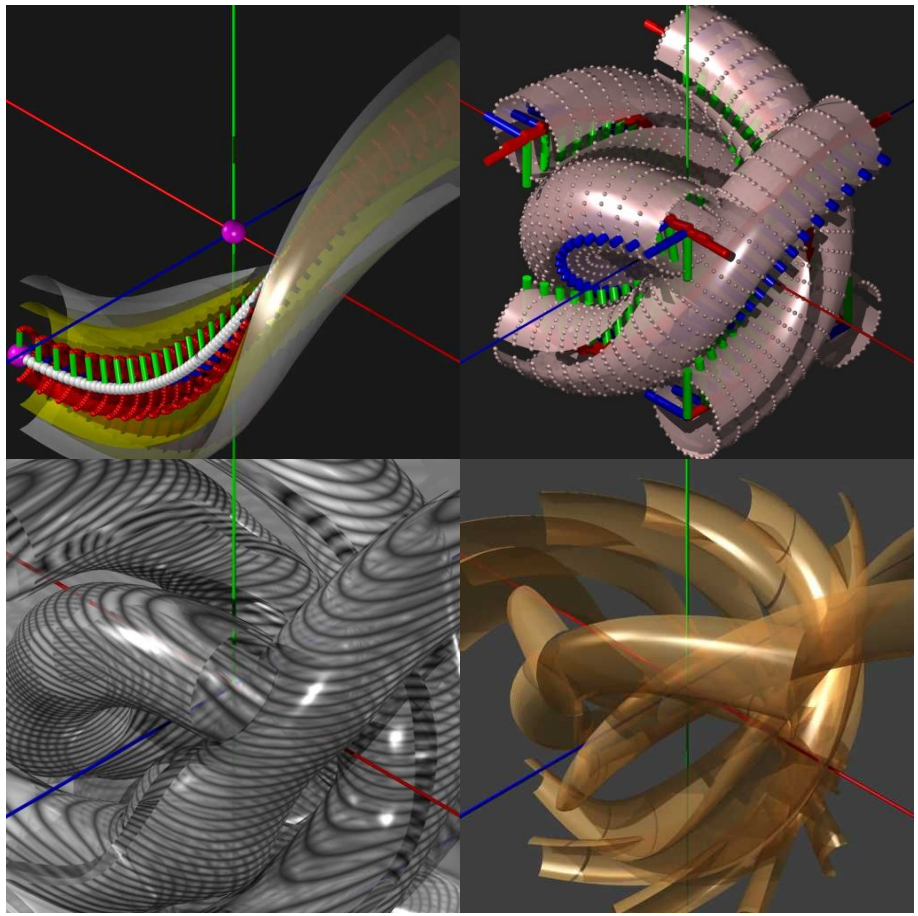
La seule solution est d'appliquer l'opérateur MIR() à la courbe chemin avant de calculer les points de contrôle de la surface. Le résultat est atteint, mais on ne peut pas considérer que l'opérateur tubage() produise une pSurface, au sens où on l'a entendu jusqu'ici.

Voici une implémentation dans la syntaxe POVRAY/pFlibs, sous la forme d'un opérateur pipe() :

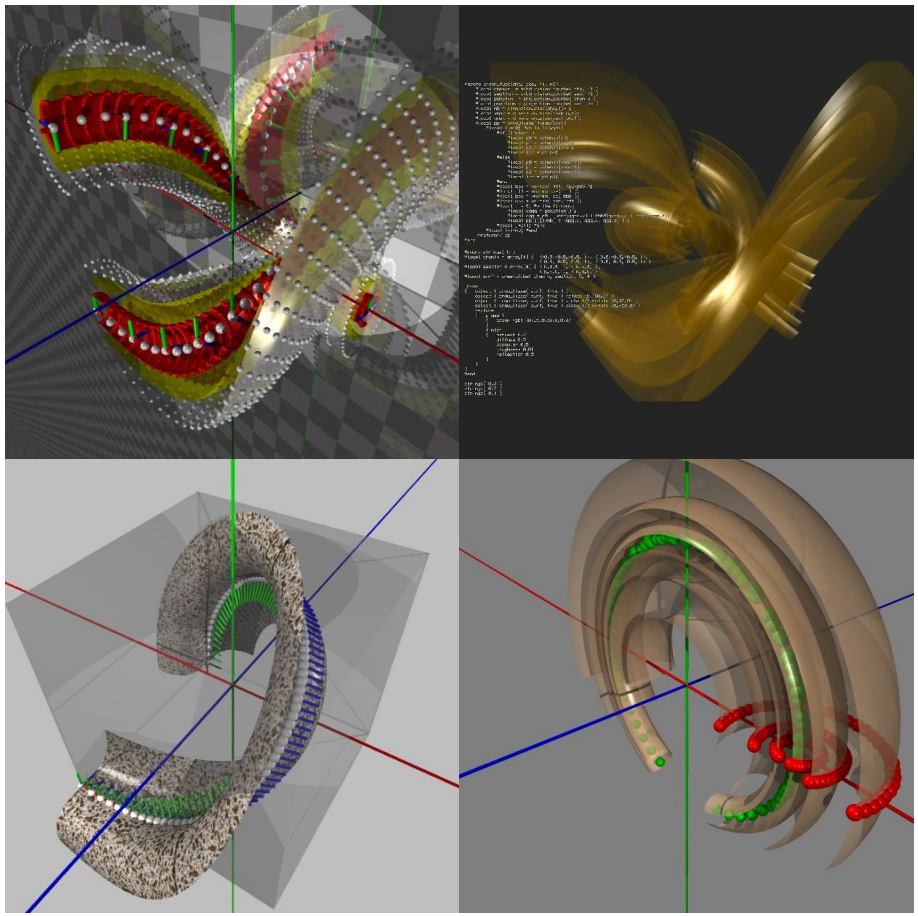
```
/*
   SURFACE TUBULAIRE
  input: une pLm chemin et une pLn section
  les valeurs de recusrion sur u et v (pre-subdivision)
  output: une pSmn (qui pourra etre subdivisee)
  nota: fonctionne avec les courbes rationnelles
  appel:  #local surf = pipe( chemin, section, 4, 3 )
*/
#macro pipe(che, sec, r1, r2)
  #local chemin  = pFsubdivision( 1, che, < r1,0,0,0 > )
  #local section = pFsubdivision( 1, sec, < r2,0,0,0 > )
  #local vmax = taille(chemin);
  #local umax = taille(section);
  #local tube = array[vmax]
  #local pp = array[umax]
  #local i = 0; #while (i< vmax)
    #local mat = pFgetPijk( 1, che, i/(vmax-1) )
    #local j = 0; #while (j< umax)
      #local pp[j] = pFtransform( mat, section[j] );
    #local j=j+1; #end
    #local tube[i] = pp
  #local i=i+1; #end
  tube
#end
```

Noter que contrairement à l'opérateur cross() vu précédemment, cet opérateur pipe() incorpore les paramètres de récursion utilisés par les opérateurs pFsubdivision() ; on a vu qu'il était nécessaire en effet d'appliquer la subdivision à la courbe chemin avant la composition, et pour des raisons de symétrie dans l'appel, on subdivise également la courbe section avant la composition.

Remarque : la non commutativité apparue entre l'opérateur MIR() et l'opérateur tubage() qui a pour effet de produire deux surfaces différentes, amène à distinguer deux classes d'opérateurs de composition: ceux qui commutent avec MIR() et les autres. Seul l'ensemble des formes pascaliennes muni d'opérateurs commutatifs produisant des éléments de cet ensemble, peut être considéré comme base sûre d'une géométrie unitaire des formes gauches. Les autres sont à manipuler avec un maximum de précautions, quelle que puisse être leur utilité pratique, en attendant une généralisation des pFormes qui permette d'englober naturellement les tubages.



figures 323.2 à 323.5 : quelques exemples de tubage dupliqués en rotation.



figures 323.6 à 323.9 : quelques aliens !

324 surfaces affines

Les surfaces issues du produit (profil x section) et les surfaces tubées peuvent être considérées comme cas particuliers de surfaces s'exprimant sous la forme plus générale suivante :

```
surface = AFFINE * courbe
```

dans laquelle AFFINE est une matrice 4x4 qui peut se lire ainsi :

```
           |  Nx*Kx Bx    Tx    dx  |
AFFINE =   |  Ny    By*Ky Ty    dy  |
           |  Nz    Bz    Tz*Kz dz  |
           |  Sx    Sy    Sz    dt  |

avec   N:     vecteur normal
       B:     vecteur binormal
       T:     vecteur tangente
       d:     vecteur translation
       K:     vecteur scalaire
       S:     vecteur déformation
```

les quatre premiers vecteurs étant liés à une courbe chemin, le vecteur k apportant une modulation et le vecteur S une déformation tangentielle (shear), et toute autre interprétation des 16 coefficients de la matrice 4x4 pouvant a priori être envisagée.

Voici dans la syntaxe POVRAY/pFlibs, une implémentation d'une extension de l'opérateur pipe (), waving_pipe() appliquant à la surface tubulaire produite une ondulation sinusoïdale avec une amplitude et une fréquences données :

```
/*
SURFACE TUBULAIRE ONDULEE
#local surf = waving_pipe( chemin, section,< 4, 3 >,< 0.5, 5 > )
*/
#macro waving_pipe(che, sec, r, ondula )
  #local chemin  = pFsubdivision( 1, che, < r.x,0,0,0> )
  #local section = pFsubdivision( 1, sec, < r.y,0,0,0> )
  #local vmax = taille(chemin);
  #local umax = taille(section);
  #local tube = array[vmax]
  #local pp = array[umax]
  #local i = 0; #while (i< vmax)
    #local uu = i/(vmax-1);
    #local mat = pLgetFrenet( che, uu )
    #local coeff = 1 + ondula.x*sin(2*pi*uu*ondula.y );
    #local temp = mat
```

```
    #local temp[0][0] = mat[0][0]* coeff;
    #local temp[1][1] = mat[1][1]* coeff;
    #local temp[2][2] = mat[2][2]* coeff;
    #local j = 0; #while (j< umax)
      #local pp[j] = pFtransform( temp, section[j] );
    #local j=j+1; #end
    #local tube[i] = pp
  #local i=i+1; #end
  tube
#end
```

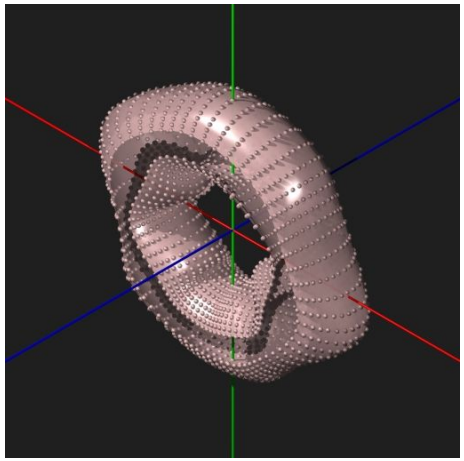
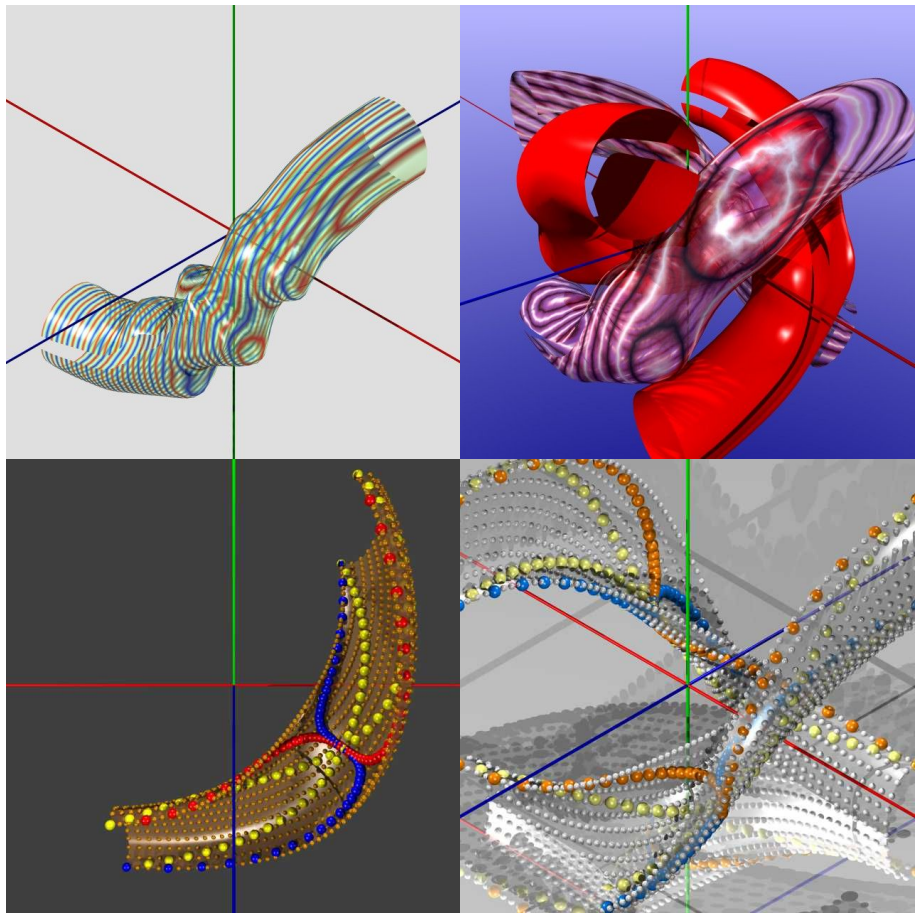


figure 324.1 : une portion de tore à rayon variable sinusoïdalement.

Remarque 1 : ce qui a été dit dans la section 323 sur les surfaces tubulaires s'applique également à cette section : l'opérateur waving_pipe() n'est pas un opérateur produisant une pSurface et doit être utilisé avec précaution.

Remarque 2 : cela n'enlève rien au caractère général de l'opérateur AFFINE, un opérateur linéaire n'impliquant aucunement l'utilisation d'une métrique et d'une opération de normalisation et d'orthogonalisation. Il faudrait explorer cette piste ...



Figures 324.2 à 322.5 : d'autres tubages variables et quelques lignes tracées dessus.

325 formes parallèles

Le surfaces tubulaires à section circulaire sont un exemple de forme obtenue en recherchant un ensemble de points situés à distance constante d'une courbe donnée ; on peut considérer la surface comme produite par le déplacement du centre d'une sphère à rayon constant le long de la courbe chemin. Un problème analogue consiste à rechercher les surfaces situées à une distance constante d'une surface donnée, des surfaces parallèles.

L'implémentation dans la syntaxe POVRAY/pFlibs est donnée ci-après :

```
/*
  SURFACES PARALLELES
  crée une surface située à la distance dd d'une surface
  autres cas possibles avec modulation de dd
*/
#macro surface_parallele( surf, dd )
  #local m = taille( surf );
  #local n = taille( surf[0] );
  #local S = surf
  #local i = 0; #while (i< n)
    #local j=0; #while (j< m)
      #local mat = pFgetPijk( 2,surf,< j/(n-1),i/(m-1),0,1 > )
      #local nn = < mat[0][0], mat[1][0], mat[2][0] >;
      #local pp = S[j][i];
      #local pt = pp.t;
      #local qq = < pp.x/pp.t, pp.y/pp.t, pp.z/pp.t >;
      #local qq = qq + nn*dd;
      #local S[j][i] = < qq.x*pt, qq.y*pt, qq.z*pt, pt >;
    #local j=j+1; #end
  #local i=i+1; #end
  S
#end
```

Remarque : comme vu avec le tubage, la construction des surfaces parallèles fait intervenir le repère local obtenu par l'appel de l'opérateur pFgetPijk(). Une surface parallèle n'est donc pas à proprement parler une pSurface.

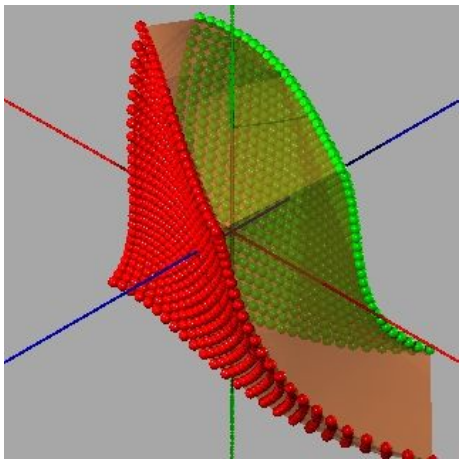
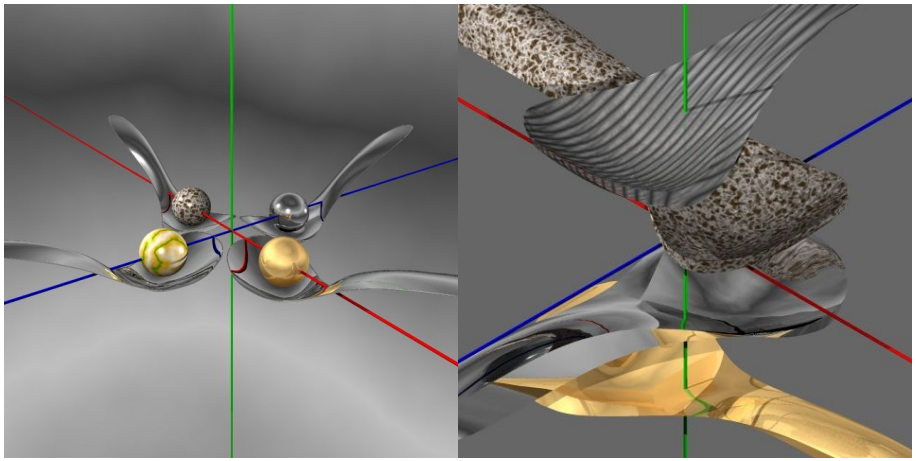


figure 325.1 : deux pSurfaces parallèles contenant un pVolume.



figures 325.2 à 325.3 : des volumes construits sur des surfaces parallèles, pour produire des cuillières épaisses en granit, bois, chrome et plexiglas strié !

326 surfaces développées

Du fait des propriétés des dérivées des pCourbes (qui sont des pCourbes de degré-1), l'extrémité du vecteur tangent (non normalisé) en un point parcourant une pLn se déplace sur une pLn; la surface engendrée par ce vecteur tangent est alors une pS2n, une surface réglée dont on sait qu'elle est de plus développable, c'est-à-dire qu'elle peut être déroulée sur un plan sans pli ou déchirement (un cône est une surface développable, une sphère non). L'implémentation dans POVRAY/pFlibs est laissée au lecteur, à titre d'exercice ;) -cf schéma 326-.

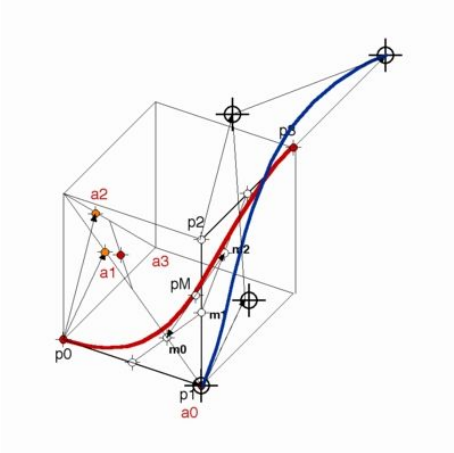


figure 326 : une pS32 (cubicoïde réglé) construit sur deux pL4 (cubiques) (rouge et bleue).

Remarque : l'étude des surfaces tubulaires et des surfaces parallèles a montré la conséquence de l'introduction d'une normalisation dans l'opérateur pFgetPijk(), le résultat sort de l'ensemble cohérent de la géométrie non métrique des pSurfaces. L'étude des surfaces développées montre l'intérêt de l'utilisation d'un vecteur tangente non normalisé, et on pourrait imaginer qu'un opérateur pFgetPijk() ne normalisant pas les trois vecteurs tangente, normale et binormale, produirait les résultats souhaités, entrant dans la famille des pSurfaces. Ceci reste à étudier !

33 combinaisons linéaires spéciales

Au sommaire de cette section :

Si toute combinaison linéaire de points dont la somme des coefficients est égale à 1 produit un point "valable" (invariant dans un changement de repère), il en est de même pour les combinaisons linéaires de pFormes en général, issues en dernière analyse de combinaisons valables de points :

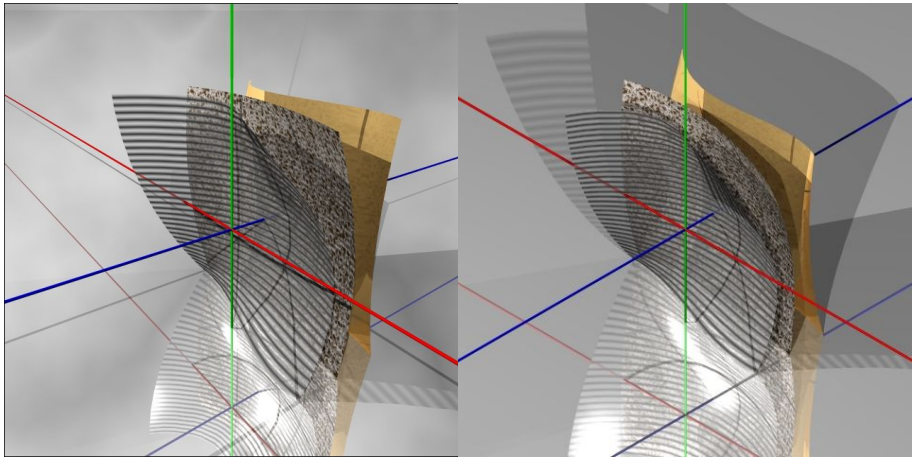
$$pF = \sum_i ki.pFi \qquad \text{avec } \sum_i ki = 1.$$

On connait déjà la forme milieu de deux ou N formes, les formes produites à l'aide de l'opérateur MIR() et celles obtenues via les opérateurs commutatifs de composition, toutes formes à combinaison linéaire pascalienne. Nous allons aborder d'autres combinaisons linéaires, non pascaliennes, et étudier dans quelle mesure elles peuvent entrer dans leur champ.

331 formes symétriques

Deux points P1 et P2 étant donnés, l'expression P = 2*P1 -P2 produit le point P symétrique de dP2 par rapport à P1. On peut remplacer les points par toute pForme, et par exemple jouer avec la combinaison S = 2*S1 -S2 qui produit la forme symétrique de S2 par rapport à la forme S1 ; dans le cas particulier où S1 est un plan, on obtient une symétrie plane. En voici une implémentation dans la syntaxe POVRAY/pFibs pour le cas de surfaces :

```
/*
SURFACES SYMETRIQUES
crée une surface symétrique par rapport à une autre surface
  cas particulier: symétrie plan
*/
#macro surface_symetrique( s1, s2 ) // surf = 2*s1 - s2
  #local M = taille( s1 );
  #local N = taille( s2[0] );
  #local surf = array[M]
  #local i=0; #while (i< M)
    #local pp = array[N]
    #local j=0; #while (j< N)
      #local pp[j] = 2*s1[i][j] - s2[i][j];
    #local j=j+1; #end
    #local surf[i] = pp
  #local i=i+1; #end
  surf
#end
```



figures 331.1 et 331.2 : la feuille dorée et la feuille transparente striée sont symétriques par rapport à la feuille en granit.

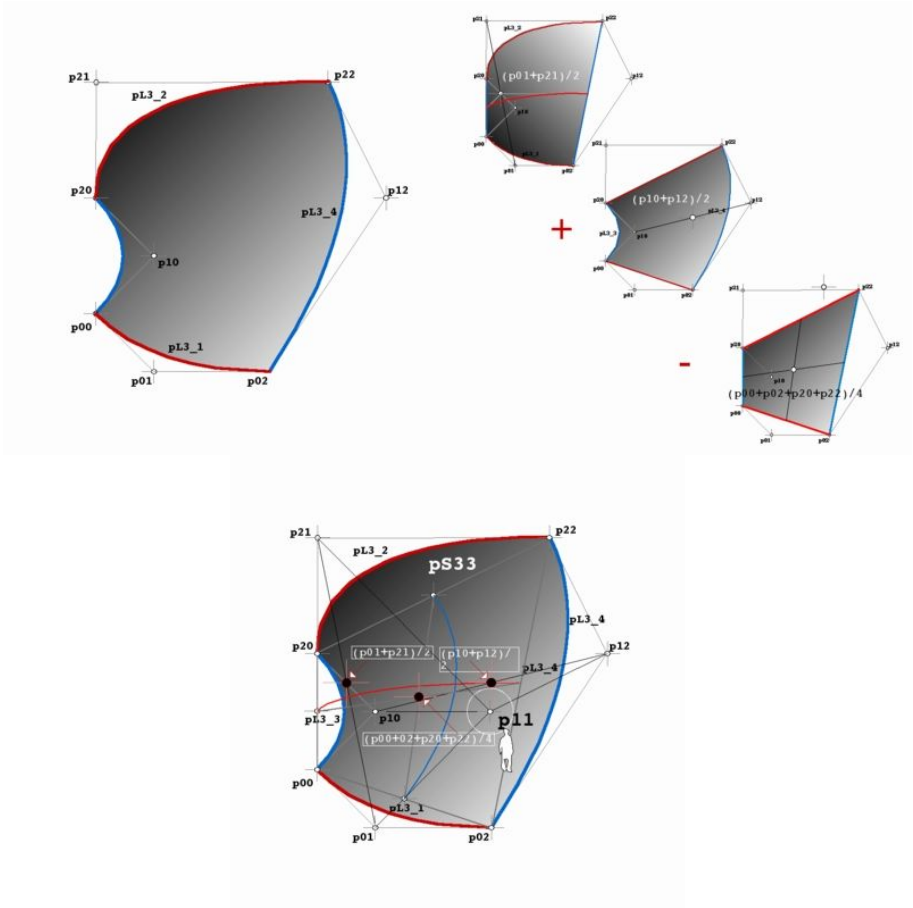
Il est possible de traiter de façon analogue toute autre combinaison de surface, et nous allons en voir une application simple et puissante dans les surfacs de Coons.

332 surfaces de coons

Un type de combinaison de surfaces particulièrement utile est celui découvert par Coons. La combinaison suivante de trois surfaces :

```
S = S0 + S1 - S2                ( noter: 1+1-1 = 1, donc c'est OK )
```

est utilisée dans la définition de carreaux ayant la propriété d'interpoler 4 courbes quelconques concourantes deux à deux et formant une limite continue (homéomorphe à un cercle complet).



figures 332.1 à 332.3 : une surface de Coons sur 4 pL3, combinaison linéaire de 3 pSurfaces, pS1+pS2-pS3, est une pSurface.

Nous limitant au cas des pCourbes, considérons en 4 (pLn1_1, pLn2_2, pLn3_3, pLn4_4) concourantes deux à deux aux points P00, P01, p10, P11. En égalisant par insertion le nombre de points de contrôle des 4 pCourbes à une valeur commune n (n1 = n2 = n3 = n4 = n), on peut réécrire la combinaison sous la forme :

```
S =  MIR( pLn_1, pLn_3 )         // S0
  +  MIR( pLn_2, pLn_4 )         // S1
  -  MIR( P00, P01, P10, P11 )   // S2
```

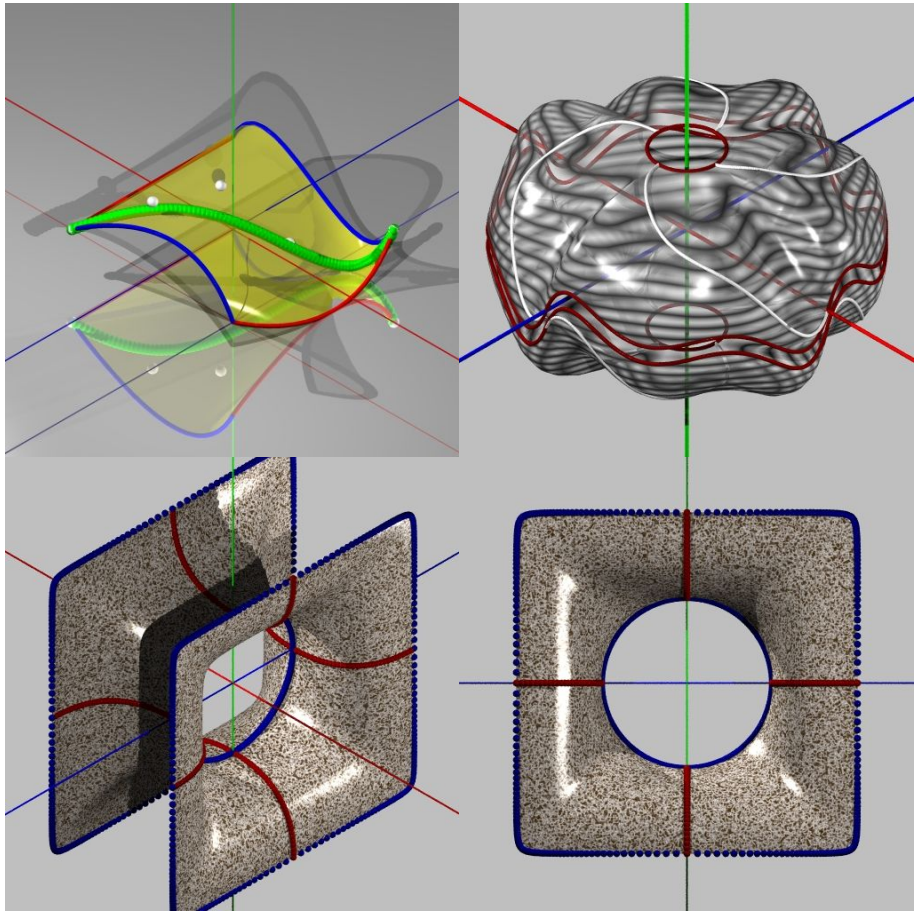
S0 est une surface réglée construite sur pLn1_1, pLn3_3, S1 une surface réglée construite sur pLn2_2, pLn4_4, et S2 une facette gauche construite sur les quatre points de rencontre. Le nombre des points de contrôle de S0, S1 et S2 étant identiques, on peut appliquer cette expression au point milieu :

```
Pm = MI( pLn_1, pLn_3 )
   + MI( pLn_2, pLn_4 )
   - MI( P00, P01, P10, P11 )
```

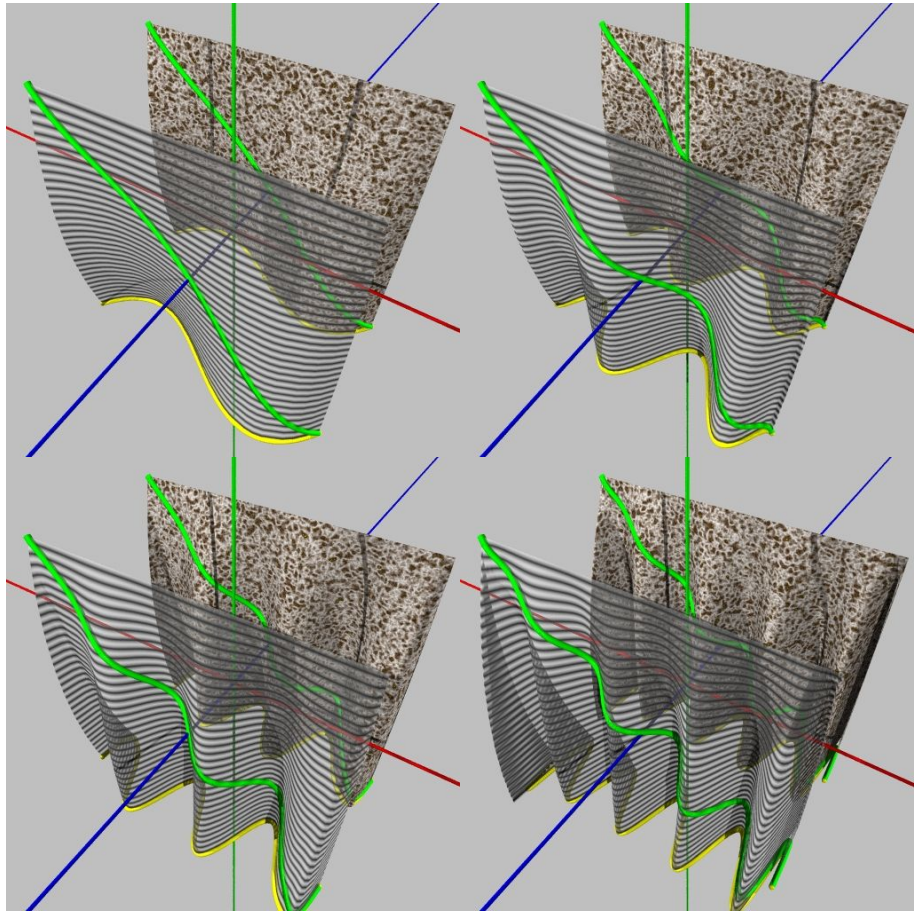
point de départ pour lancer une récursion et obtenir une pSurface. Le résultat est qu'une surface de Coons construite sur des pCourbes est une pSurface (mais on le savait déjà). D'autres combinaisons linéaires plus complexes pourraient être abordées, notamment les surfaces de Gordon, interpolant une série de courbes (penser par exemple à la génération d'un Joystick). L'implémentation dans la syntaxe POVRAY/pFlibs est donnée en annexe (le code est similaire au code des surfaces symétriques présenté plus haut), voici un exemple d'appel produisant un carreau de Coons et sa diagonale à partir de 4 pCourbes de degrés différents (pL3, pL4, pL2, pL3) :

```
#local L1 = array[3] {< -1,0,-1,1 >,< 1/2,1,-1,1 >,< 1,0,-1,1 >}
#local L2 = array[4] {< -1,0,1,1 >,< -1/2,1,1,1 >,< 1/2,-1,1,1 >,
<1,0, 1,1 >}
#local L3 = array[2] {< -1,0,-1,1 >,< -1,0,1,1 >}
#local L4 = array[3] {< 1,0,-1,1 >,< 1,-1,1/2,1 >,< 1,0, 1,1 >}
#local coons = creer_coons( L1, L2, L3, L4 )
#local diag = pFdiagonalisation( 2, coons )

draw( 2, coons, finesse(< 3,3 >)+surface(LISSE)+ma_couleur( <
1,1,0,0.5 > ) )
draw( 1, L1, finesse(4)+courbe(0.02)+ma_couleur( < 0,0,1 > ) )
draw( 1, L2, finesse(4)+courbe(0.02)+ma_couleur( < 0,0,1 > ) )
draw( 1, L3, finesse(4)+courbe(0.02)+ma_couleur( < 1,0,0 > ) )
draw( 1, L4, finesse(4)+courbe(0.02)+ma_couleur( < 1,0,0 > ) )
draw( 1, diag, point( 0.05 )+ma_couleur( < 1,1,1 > )
draw( 1, diag, finesse(4)+point(0.05)+ma_couleur(< 0,1,0 > ) )
```

Les figures suivantes montrent à quel point une simple combinaison linéaire (S = S1 + S2 – S3) peut se révéler riche de possibilités, et la route est ouverte à d'autres combinaisons entre pFormes, en commençant par les pVolumes.



figures 332.4 à 332.7 : pSurface de Coons et sa diagonale, une coquilles comprenant deux surfaces symétriques construites chacune sur un petit cercle rouge et un grand cercle ondulé, et une pseudo-surface minimale construite sur 8 Coons, AXO et vue de face.



figures 332.8 à 332.11 : quatre pSurfaces de Coons construites sur un segment et des pCourbes ondulées sur des fréquences croissantes, des rideaux sous le vent !

34 concaténations, splines

Au sommaire de cette section :

- 341 splines non interpolantes
- 342 splines interpolantes
- 343 les NURBS

Lorsque l'on est conduit à utiliser plus de trois ou quatre points de contrôle les constructions peuvent devenir problématiques. On imagine alors de concaténer plusieurs pCourbes contrôlées par un même nombre réduit de points en définissant des règles de raccordement assurant différentes conditions de continuité sur les tangentes, la courbure et la torsion. Plusieurs approches sont possibles, suivant qu'on souhaite ou non l'interpolation des points de contrôle.

341 splines non interpolantes

L'approche la plus connue est l'approche "B_spline" que nous allons brièvement exposer.

Sur N points donnés Qi on fait glisser une "fenêtre" de largeur 2, 3 ou 4 points, le glissement se faisant par incrément d'un point dans chaque cas.

1) Avec une fenêtre de 2 points (Qi,Qi+1) on définit deux points comme suit :

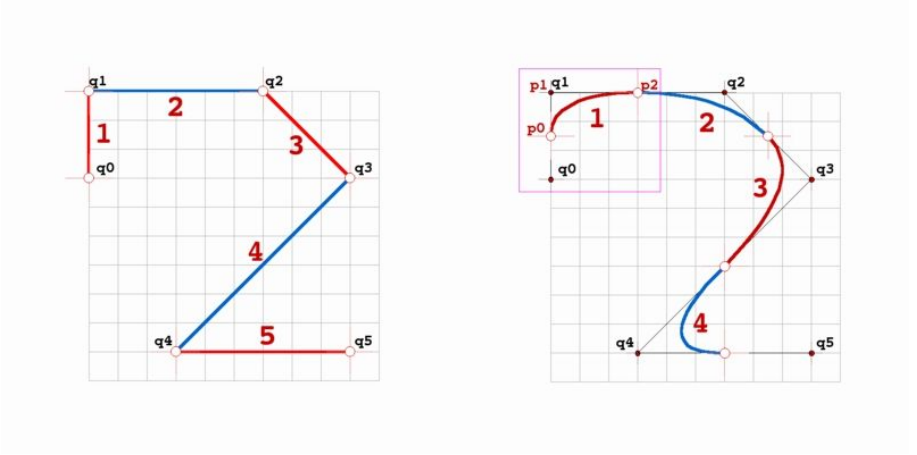
```
p0 = Qi
p1 = Qi+1
```

utilisés comme points de contrôle d'une pL2 (segment) par fenêtre et l'on obtient simplement la suite des segments formant le polygone de base, une B-spline de degré 1.

2) Avec une fenêtre de 3 points (Qi,Qi+1,Qi+2) on définit trois points (en i/2 pour i=[0,2]) comme suit :

```
p0 = 1/2(Qi + Qi+1 )
p1 = 1/2(   2.Qi+1 )
p2 = 1/2(     Qi+1 + Qi+2)
```

utilisés comme points de contrôle d'une pL3 (parabole) par fenêtre. Par construction, les courbes successives se raccordent, elles possèdent des tangentes colinéaires et de même module, mais la planéité des paraboles implique des discontinuités de la torsion qui est nulle à l'intérieur des paraboles mais qui peut être infinie aux points nodaux ; cette construction est donc réservée au dessin plan et les logiciels de DAO/CAO en font un usage intensif sous le nom de splines quadriques.

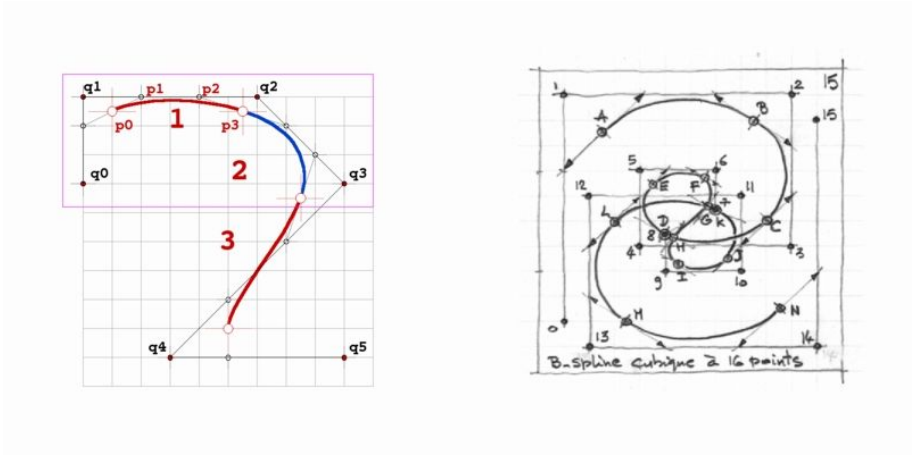


figures 341.1 et 341.2 : splines linéaire et quadratique.

3) Avec une fenêtre de 4 points (Qi,Qi+1,Qi+2,Qi+3) on définit quatre points (en i/3 pour i=[03]) comme suit :

```
p0 = 1/6(Qi + 4.Qi+1 +   Qi+2       )
p1 = 1/6(     4.Qi+1 + 2.Qi+2       )
p2 = 1/6(     2.Qi+1 + 4.Qi+2       )
p3 = 1/6(       Qi+1 + 4.Qi+2 + Qi+3)
```

utilisés comme points de contrôle d'une pL4 (cubique) par fenêtre. Par construction, les cubiques successives se raccordent, elles possèdent des tangentes colinéaires et de même module, et les plans osculateurs confondus assurent la continuité de la torsion, en la distribuant en fait sur la longueur de chaque courbe. Cette dernière propriété en fait l'outil idéal pour piloter des courbes complexes gauches dans l'espace, la trajectoire d'une caméra, d'un bras de robot, etc... et les logiciels de DAO/CAO en ont fait l'outil à tout faire.

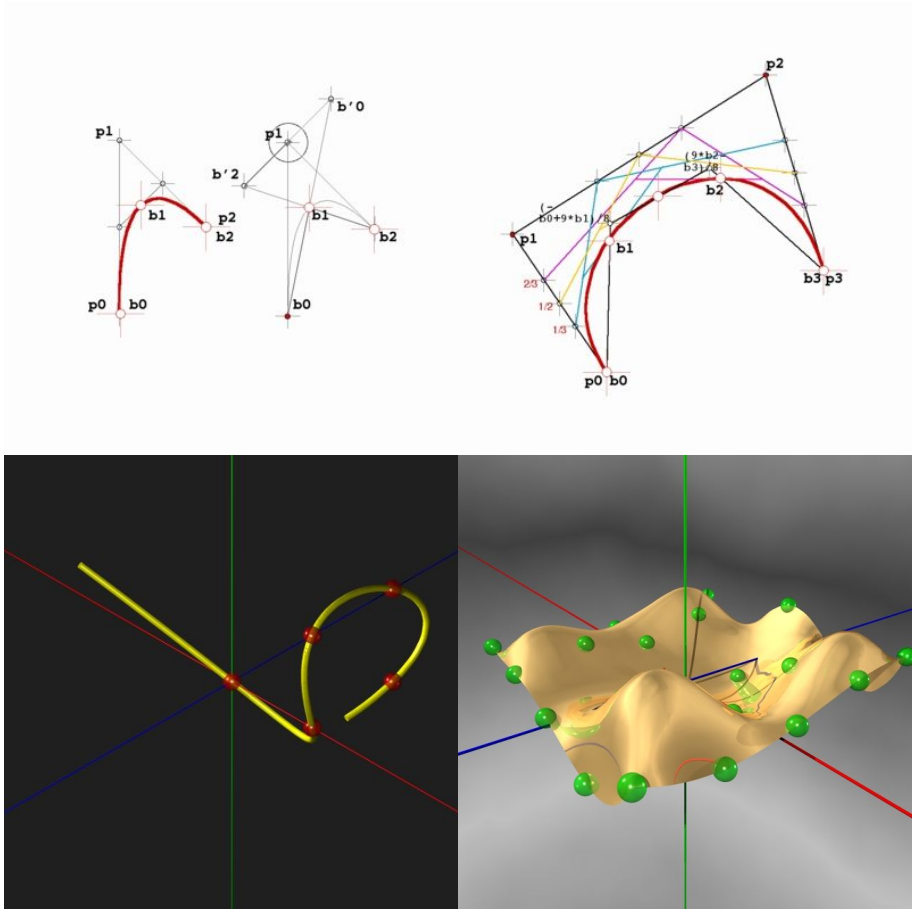


figures 341.3 et 341.4 : splines cubiques, sur 6 et 16 points.

Remarque 1 : en fait, la construction simplifiée des B8splines présentée ici correspond à ce que l'on appelle les splines uniformes ; dans le cas quadrique par exemple, on a choisi de positionner les points nodaux au milieu des points de contrôle, mais il serait tout à fait possible et intéressant d'étudier des positionnements dissymétriques et différents, générant ce que l'on appelle des splines non uniformes, des beta-splines, etc, traitées abondamment dans la littérature sur le sujet.

Remarque 2 : il faut noter que ces courbes n'interpolent pas les points donnés, à l'exception du cas linéaire (où l'on obtient simplement le polygone de contrôle) ; on peut s'ingénier à créer des points fantomes en début et en fin pour "attirer" la courbe sur les premier et dernier points, on peut "dupliquer" certains points intermédiaires pour leur donner plus de poids et y accrocher la courbe, mais la continuité peut s'en ressentir jusqu'à provoquer l'apparition de points anguleux (ce qui peut par ailleurs être une intéressante propriété) ; on peut aussi reconsidérer le problème et construire de véritables splines interpolantes.

342 splines interpolantes



figures 342.1 à 342.4 : construction de pL3 et pL4 interpolantes ; une pCourbe jaune interpolant 5 points, et une pSurface interpolant 25 points.

La construction d'une concaténation de courbes de degré 1, 2 ou 3 interpolant les points donnés ne présente pas une grande difficulté de principe, mais on en limitera ici la présentation au cas quadrique. Soit N points bi avec i=[0,N-1] à interpoler à l'aide de paraboles ; en se donnant un

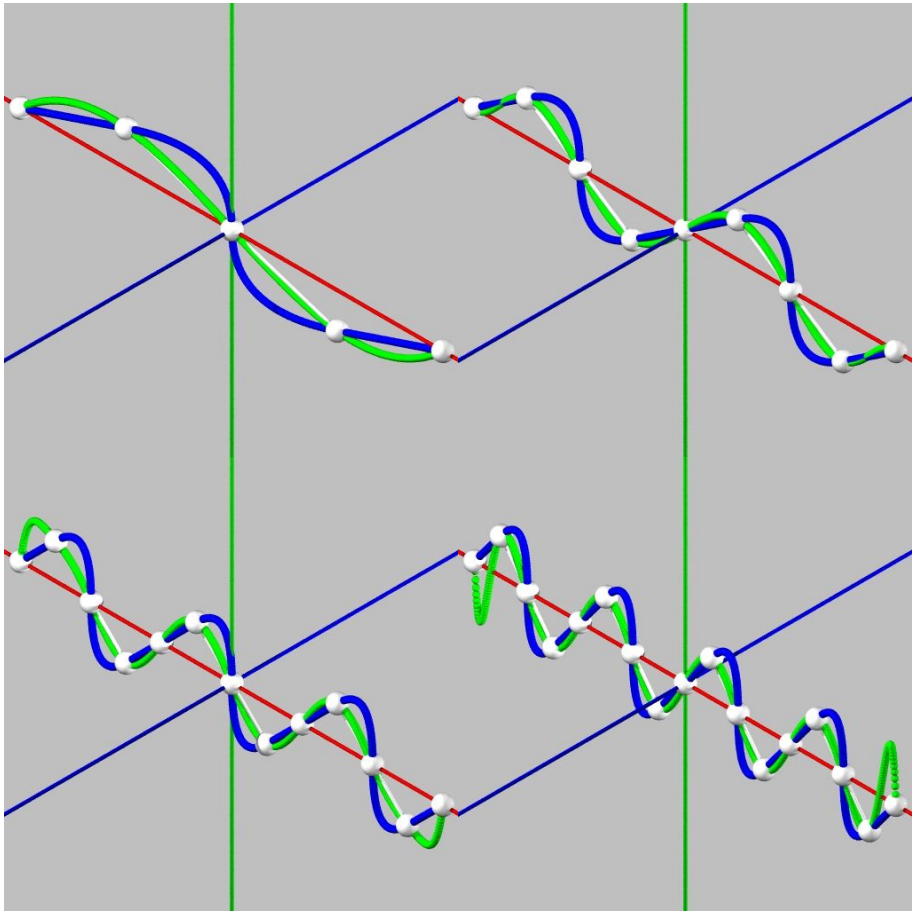
point b_1 définissant avec b0 un vecteur tangent au départ de la courbe, on peut démarrer la construction de la première parabole en remarquant que son second point de contrôle est le point symétrique de b_1 par rapport à b0 :

```
p0 = b0
p1 = 2*b1 - b_1
p2 = b2
```

et ainsi de suite pour les autres paraboles, b_1 étant le point p1 de la parabole précédente.

Remarquons la simplicité de l'opération, aucune inversion de système linéaire n'est requise pour trouver la solution, comme c'était le cas en 221. Le lecteur traitera le cas cubique à titre d'exercice :). Le programme suivant produit un cercle parfait suivant la méthode quadrique (une parabole génère facilement un arc de cercle sur 90°). Il est intéressant de remarquer qu'en déplaçant le point b_1 parallèlement à l'axe OZ, la courbe reste sur le cylindre d'axe OZ et de rayon 1/2.

```
#macro spline_quadrique( b0, b )
        #local k = sqrt(2)/2;   // -> cercle
        #local n = _size( b );
        #local spline = array[n-1]
        #local q = array[3]
        #local q[1] = b0;
        #local i=0; #while (i< n-1)
                #local q[0] = bb[i];
                #local q[1] = 2*bb[i]-q[1];
                #local q[2] = bb[i+1];
                #local spline[i] = q
                #local spline[i][1] = spline[i][1]*k;
        #local i=i+1; #end
        spline
#end
#local b_1 = < 1/2,-1/2,0,1 >;  // point init
draw_point( b_1, point( 0.05, < 1,0,0,0.7 > ) )
#local bb = array[5] { < 1/2,0,0,1 >, < 0,1/2,0,1 >, < -1/2,0,0,1
>,
                      < 0,-1/2,0,1 >, < 1/2,0,0,1 > }
draw_courbe( bb, 0, point( 0.04, < 1,0,0 > ) )
#local circ = spline_quadrique( b_1, bb )
#local i=0; #while (i< _size(circ))
        draw_courbe(circ[i],3,point(0.02,< 1,1,0 > ))
#local i=i+1; #end
```



figures 342.5 à 342.8 : splines interpolantes quadriques bleues et pCourbes interpolantes vertes.

343 les NURBS

Nous n'avons parlé jusqu'ici que de concaténations basées sur des distributions uniformes de points ; mais on peut également envisager d'imposer des comportements différents (asymétriques) au droit des points nodaux, ou de multiplier des points en un même endroit et construire ainsi des splines non uniformes, et le raisonnement reste analogue. On peut donc construire des concaténations de splines interpolantes ou non, et uniformes ou non. En se souvenant que le travail se fait dans l'espace R^4, on a ainsi retrouvé les NURBS, "Non Uniform Rational B-splines", d'une façon qui reste simple, ce qui n'est pas rien... Cette famille de formes présentée comme un "must" en CAO, une sorte de monstre mathématique inaccessible à la compréhension directe est assez malmené par les logiciels de CAO ; en dehors de logiciels spécifiques comme Rhino qui les prend pour base, les NURBS ne sont la plupart du temps réellement utilisés que dans la construction de cercles et de surfaces de révolution. On retiendra simplement que les NURBS peuvent être vues comme des concaténations de pFormes interpolantes ou non, uniformes ou non dont les points de contrôle sont obtenues à partir des points de base par de simples transformations linéaires "glissantes". La question qui se pose fréquemment de savoir si les courbes de Bézier sont des cas particuliers des NURBS n'a au fond pas plus d'intérêt que de savoir si un segment est un cas particulier du cercle ou de savoir qui prime de l'atome ou de la molécule. Dans l'approche pascalienne, il s'agit de membres d'une même famille !

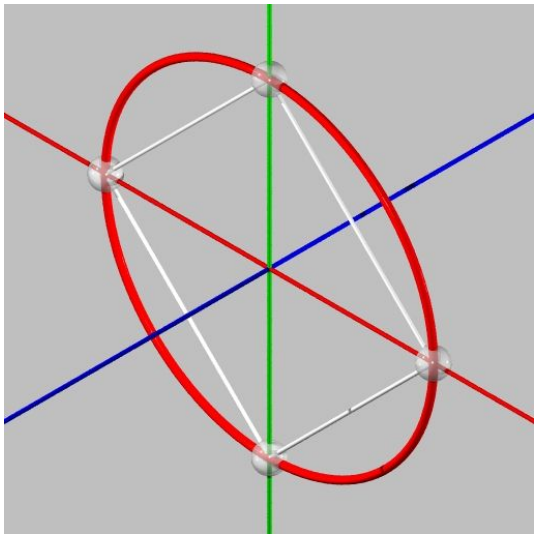


figure 343 : un cercle NURBS, spline quadrique interpolant 4 points.

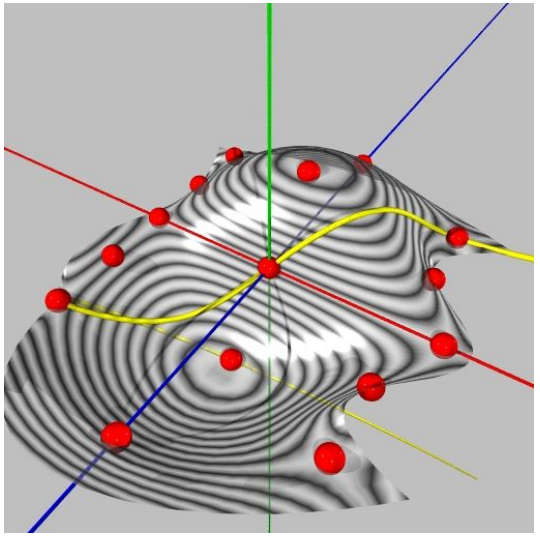


figure 343.2 : une surface NURBS interpolant 3 splines contrôlées par 5 points, dans un intervalle de définition légèrement agrandi, et sa diagonale jaune ; la texture marbre permet de visualiser les lignes de niveau.

35 opérateurs de déformation

En théorie, toute pForme peut être manipulée à volonté dans le mondre détail au moyen des sous-pFormes la générant, dont les plus profondes sont les points de contrôle eux-mêmes. En pratique, il n'est pas toujours facile de faire ainsi : comment s'y prendre pour appliquer de façon cohérente à un objet une déformation du type étirement/compression, une flexion, une torsion, ou une ondulation ? Imaginez la difficulté de positionner à la main les points de contrôle de la vingtaine de bicubiques reconstituant la forme d'une théière - pour prendre l'exemple favori des infographistes- dans laquelle on verse du thé bouillant afin de lui faire subir un gonflement, une torsion de douleur et enfin une fusion plastique l'applatissant complètement sur la table... Il faut donc envisager un opérateur capable d'appliquer globalement de telles déformations. Deux approches, au moins, sont envisageables : la première consiste à appliquer une fonction mathématique (par exemple une sinusoïde pour l'ondulation) à tous les points de la pForme, mais l'inconvénient est que la forme résultat n'est plus une pForme, et on ne peut plus lui appliquer les opérateurs vus plus haut. La seconde consiste à "immerger" la pForme dans une autre pForme, et l'on sait que le résultat est une pForme. On peut ainsi immerger une pCourbe dans une pSurface (par exemple une pS32) et en déplacer les points de contrôle intermédiaires pour simuler la déformation.

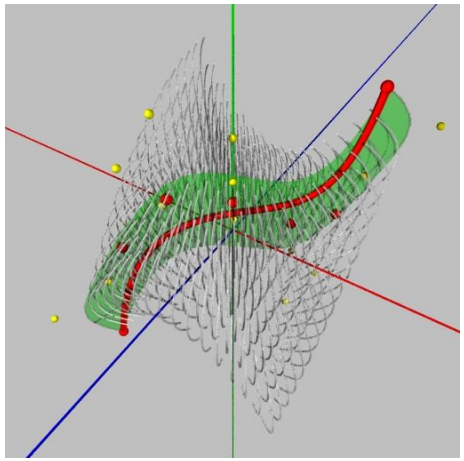


figure 35.1 : une facette plane et sa diagonale immergées dans un cube (pV332) déformé .

On peut également immerger un pVolume complexe dans un autre pVolume plus simple et facile à déformer. On dispose ainsi d'une méthode pour créer des anamorphoses.

La figure 35.1 montre la déformation d'une facette plane et de sa diagonale immergées dans un cube (pV332) construit sur deux biquadriques (pS33) ; on dispose de 3x3x2 = 18 points de contrôle pour déformer cette facette et sa diagonale, et toute autre pForme (courbe, surface, volume) immergée.

La figure 35.2 montre un autre exemple de déformation : une portion de tore en granit (pS33) et sa diagonale blanche (pL5) sont immergées dans un cube (pV223) contrôlé par 3 facettes (pS22) et représenté par une grille de 12 points rouges ; la facette médiane est déplacée sinusoïdalement à l'horizontale, et le cube entraine dans sa déformation la portion de tore et sa diagonale.

Remarque 1 : un point important à garder en mémoire est le fait qu'avant d'être gauche, une pForme peut parfaitement être « droite » ! Une pS22 peut parfaitement représenter un carré plan bien classique et un pV222 peut représenter un simple cube orthonormé. La particularité des pFormes est que leur définition ne suppose a priori aucune métrique et c'est grâce à cela qu'il est possible de les embarquer sans problème dans des espaces courbes.

Remarque 2 : un autre point à ne pas oublier est qu'une forme complexe dans notre espace peut être considérée comme une forme un peu plus simple dans un espace un plus complexe ; la diagonale blanche de la portion de tore embarqué dans le cube déformé de la figure 35.2 est un simple segment immergé dans la portion de tore, la portion de tore est un tore bien classique à rayon constant embarqué dans le cube. Question : quel est, dans notre espace, le nombre de points de contrôle de la pCourbe équivalente à la diagonale du tore déformé ?

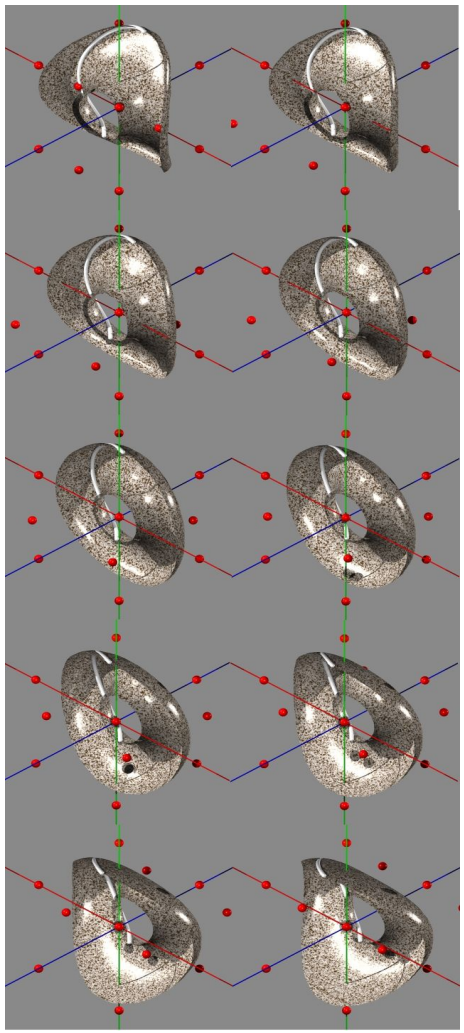
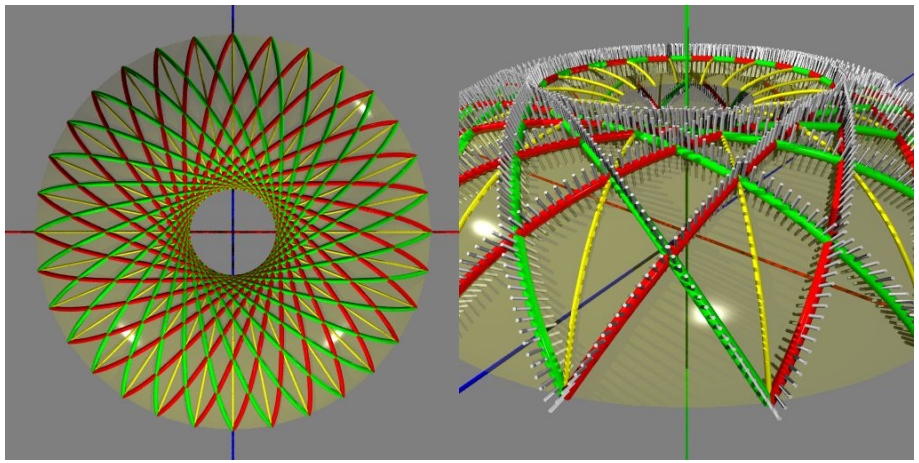


figure 35.2 : un tore (pS33) et sa diagonale (pL5) immergés dans un cube (pV322) qui se déforme par translation sinusoïdale horizontale de son plan de contrôle médian.

36 géométrie dans les pFormes

On a déjà vu que sur le concept de segment immergé dans une pSurface, on pourrait étendre les concepts de parallélisme, d'orthogonalité, d'angle, étudier l'intersection de deux iSegments, ses divisions, ses prolongements, etc..., et définir ainsi une géométrie dans les pSurfaces. On n'est pas limité aux segments immergés dans des pSurfaces et le raisonnement reste valable dans toute pForme, à commencer par les pVolumes. Le présent essai en constitue une première approche, et au delà il reste à construire avec toute la précision et la cohérence nécessaires une pGéométrie non métrique des formes gauches, et ceci reste à faire ...

Si les segments immergés dans des pSurfaces ne sont en général pas des géodésiques, il est parfois nécessaire de calculer des géodésiques sur des pSurfaces. Ce qui a été fait à l'occasion d'une participation à la réalisation d'une couverture de piscine à St Quentin en Yvelines pour le bureau d'étude Michael Flach spécialisé dans les structures bois. Le principe constructif est particulièrement élégant : des planches alignées suivant des lignes géodésiques sont empilées alternativement pour constituer des arcs de forte inertie perpendiculairement à la surface, et leur croisement forme naturellement les noeuds, l'ensemble constitue une grille devant recevoir les planches de la toiture.



figures 36.1 et 36.2 : étude de géodésiques croisées sur une portion de tore .

Le problème de cette technique « est » d'appliquer une planche mince sur une surface courbe, et ce n'est pas toujours évident : dans le cas d'un hyperboloïde de révolution, on peut suivre les deux familles de génératrices rectilignes, dans le cas d'une sphère on suivra les arcs de grand cercle (ce qui est le cas des méridiens mais pas des parallèles), dans le cas d'un tore et a fortiori d'une forme plus complexe , il n'y a plus qu'une solution : suivre les lignes géodésiques. C'est donc le système différentiel posé dans la section 0232 qui doit être traité, dans une approche itérative classique aboutissant à un tableau de points définis en [x,y,z]. Dans l'implémentation réalisée dans POVRAY/pFlibs, on écrit :

```
#local geo = geodesique( surf, P, A, N, dt )
// où P= point depart, A= angle de tir, N= nb de points, dt= pas
```

figures 36.3 et 36.4 : réalisation, toiture toroïdale composée d'arcs géodésiques en planches contre-clouées permettant le croisement au droit des noeuds.

37 autres opérations sur les pFormes

D'autres formes seraient à étudier comme produits d'opérateurs sur les pFormes, les opérations booléennes sur les pFormes (intersection, différence, union), et pourquoi pas, les surfaces quasi-minimales satisfaisant à l'équation de Laplace, dont l'expression aux différences finies du point courant :

```
Pi,j = ( Pi-1,j + Pi+1,j + Pi,j-1 + Pi,j+1 ) / 4
```

rappelle la définition du point milieu d'une facette gauche ; une pSurface judicieusement choisie pourrait peut-être, avec ses iSegments et autres iCourbes, être un bon guide pour l'étude des surfaces quasi-minimales. La figure 37 a été calculée suivant une méthode classique (résolution itérative de l'équation de Laplace) ; à comparer avec la figure 332.5 utilisant la méthode de Coons pour produire une pSurface.

Remarque 1 : les opérateurs MI() et MIR() sont centraux dans la génération des formes pascaliennes, et on peut en imaginer des variantes fructueuses. L'ensemble de Cantor (1883) est un semis fractal de points défini comme l'attracteur de la famille de deux homothéties de rapport 1/3 et de centres 0 et 1. Le rapport 1/3 n'est pas choisi au hasard: si l'on prenait 1/2 on obtiendrait une bonne pCourbe (pL2) qui n'est pas du tout fractale... On peut penser qu'un opérateur TIERS() retournant deux points au 1/3 et au 2/3 et l'opérateur TIERSR() associé pourraient engendrer des pFormes fractales bien intéressantes. On peut également penser à une définition engendrant des pFormes de Peano généralisées, par exemple des pFormes de dimension topologique 1 et de dimension fractale 2, diagonales d'on ne sait quel pMonstre. Et pourquoi pas un opérateur ALEA() retournant un point aléatoire entre deux points, bien utile à vrai dire pour engendrer une distribution équilibrée (gaussienne) sur toute la forme)?

Remarque 2: les traités de géométrie différentielle se doivent d'étudier les surfaces dans leur plus grande généralité, simplement exprimables par des fonctions continues et différentiables N fois. On y trouve analysées en chaque point successivement les propriétés linéaires, ce qui se passe dans le plan tangent, et les propriétés du second ordre, la courbure, la torsion, ce qui se passe dans les quadriques osculatrices (elliptiques, paraboliques, hyperboliques), et en général ça s'arrête là, à une sorte de développement en série de Taylor des surfaces limité au second ordre. On doit au passage introduire le concept de métrique, on peut généraliser de façon très abstraite et parler de variétés riemanniènnes, mais il semble qu'on laisse au delà un grand vide sans balisage, du moins pour le commun des mortels ! En limitant l'étude des surfaces à la famille des pSurfaces, les choses se simplifient considérablement : le développement en série est a priori fini (les pSurfaces sont des polynomes de degré fini) et au delà des surfaces de référence d'ordre 2 du type ellipsoïde, paraboloïde, hyperboloïde, on trouve des êtres intéressants et faciles à manipuler (hors de toute métrique) à commencer par la biquadrique pS33 et sa diagonale pL5. La pS33 est la première vraie surface à double courbure assez malléable avec ses 9 points de contrôle pour servir de modèle à de nombreuses surfaces comme la sphère ou le tore. Quant à la pL5, on a vu dans cet essai son importance, tant dans la représentation du cercle complet, que dans le fait qu'elle se trouve être le segment immergé de la pS33, c'est à dire le composant essentiel de toute sa géométrie.

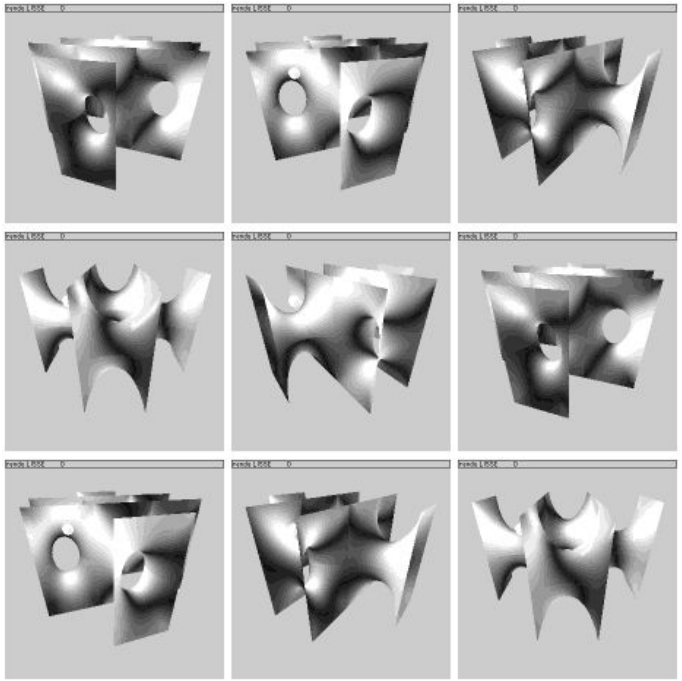
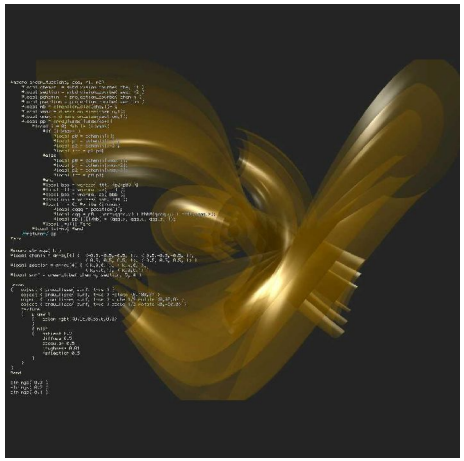


figure 37 : surface minimale périodique.

Mais tous ces chemins ont été explorés bien avant par des générations de mathématiciens talentueux qui les ont balisés dans de puissantes théories, et il n'y a donc rien de bien nouveau ici ; j'espère simplement que cette relecture différente sinon originale, basée sur des gestes élémentaires permettra au non mathématicien d'y voir plus clair, d'acquérir une meilleure maîtrise sur les formes gauches et d'ouvrir une porte vers d'autres géométries plus exotiques.

conclusion



Sous ces tubages flous sont des phrases simples, sous les formes complexes de la Sagrada Familia se trouve l'art du trait formalisé, maîtrisé et transmissible, l'art créateur partagé !

Il faut se souvenir que sans besoin de recourir à une formulation analytique complexe, les deux opérateurs fondamentaux MI() et MIR() nous ont permis d'engendrer une famille de formes, les pFormes ; nous avons étudié les relations entre les pFormes immergées dans d'autres pFormes (les ipFormes) et les pFormes de l'espace, nous les avons imaginées dans l'espace à quatre dimensions et projetées coniquement dans le notre pour produire des formes rationnelles, concaténées pour produire des splines, et combinées en sommes et produits affines pour produire des formes plus complexes, et retrouver la plupart des formes connues de la géométrie.

Les formes pascaliennes et les opérateurs associés constituent ainsi un outil de conception, de représentation et de manipulation d'une famille importante de formes géométriques. Nous noterons pour finir les points suivants :

1. **Simplicité de l'approche**
   Tous les chemins revisités dans cette approche ont été explorés bien avant par des générations de mathématiciens talentueux qui les ont balisés dans de puissantes théories, et il n'y a donc rien de bien nouveau ici ; on peut espérer que cette relecture basée sur une approche unitaire et des gestes élémentaires permettra autant à celui qui a à dessiner des formes gauches à main levée, qu'à l'utilisateur des puissants outils infographiques d'aujourd'hui, d'y voir un peu plus clair, de mieux maîtriser sa main et les outils, et même d'explorer et de découvrir de nouvelles formes. C'est justement l'un des grands intérêts de la simplicité de l'approche, cette ouverture vers de nouvelles combinaisons et immersions aboutissant à des géométries dont l'étude classique relève souvent du défi mathématique, ou du moins suppose un niveau inaccessible au commun des mortels ; je pense à la géométrie projective, à la sphère de Riemann , aux cercles de Poincaré, aux tentatives de représentation des théories cosmologiques post relativistes. La géométrie (d'après Hilbert) est l'étude des figures/propriétés invariantes dans une transformation donnée, et on est ainsi amené à envisager différentes géométries emboitées, les géométries euclidienne, affine, projective, topologique,... en enlevant successivement des «contraintes », la métrique, les angles, les points à l'infini, chaque contrainte enlevée libérant la géométrie vers une plus grande abstraction et une plus grande richesse. Dans quel «coin » de cette progression pourrait bien se trouver une géométrie construite sur les pFormes ?

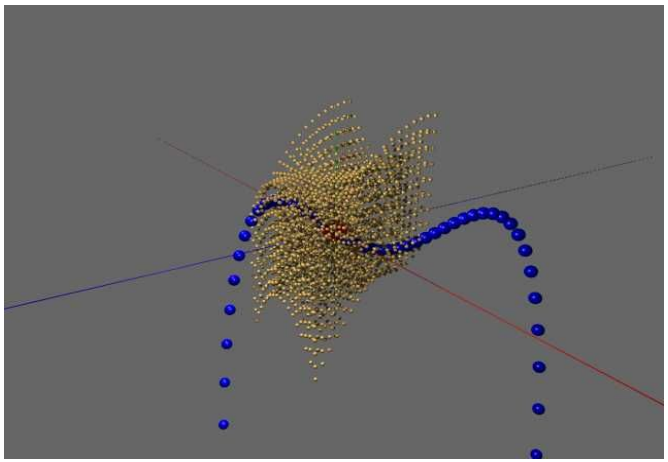
2. **Un seul outil, la corde**
   Rappelons que le geste initial défini dans l'opérateur MI() est la construction d'un point milieu, et qu'une telle construction peut se faire "à la main" dans l'espace euclidien -bien droit- avec une simple corde d'abord tendue entre deux piquets, puis repliée sur elle-même pour marquer le point milieu. Sous certaines conditions vues dans cet essai, ce geste peut encore être utilisé dans les pSurfaces : on pourra en effet les décomposer de façon plus ou moins précise en un "patchwork" de pSurfaces, et tracer «à la main » avec (en principe) une simple corde les pCourbes immergées, les pCourbes d'assemblage dans une carrosserie d'une voiture, les pLignes de couture d'un vêtement, les pCourbes d'assemblage des surfaces tendues avant ou après optimisation, etc...

3. **Anamorphose**
   Cette approche sera également utilisable dans le cas des anamorphoses ; la déformation d'une surface carrée laissant rectilignes et coplanaires les quatre cotés engendre une facette pS22 ; un segment immergé deviendra une pL3 (parabole), une parabole deviendra une pL5, un arc de cercle deviendra la projection conique d'une parabole, etc... On peut aussi envisager la déformation cohérente d'un ensemble de pFormes par immersion dans un cube gauche V222, V333, ou plus, dans des opérations de torsion, de pliage, d'étirement, etc... On dispose en tout cas des bases d'un outil d'anamorphose vectorielle à explorer plus avant et à exploiter.

4. **Règles de composition**
   Cette approche permet d'entrevoir un système de règles applicables à la «composition harmonieuse » de pFormes, étendant les règles bien connues qui s'appliquent aux formes géométriques rectilignes ; la disposition « harmonieuse » des points de contrôle d'une pForme assurerait ainsi le caractère harmonieux de la forme, et trois points situés aux sommets d'un rectangle d'or (1,618) seraient par exemple supposés engendrer une parabole « harmonieuse ». Ceci pourrait fournir un bon outil d'analyse pour les architectures courbes : il serait intéressant de savoir sur quel réseau se trouvent les points de contrôle des courbes de la Sagrada Familia, du musée Guggenheim de Bilbao, des hanches de Laetitia Casta, ou de la KA de Ford ?

5. **Tableur 3D**
   Le concept de tableur est idéalement simple : un tableau de cellules pour entrer des données et visualiser les résultats et la possibilité d'établir toute sorte de relations entre les cellules. En entrant le nombre « 2 » dans la cellule A1, le nombre « 3 » dans la cellule A2 et la formule « = A1 + A2 » dans la cellule A3, on obtient « 5 » dans cette cellule. Imaginons maintenant qu'on entre trois « pL3 » respectivement dans les cellules A1, A2 et A3, qu'on écrive dans la cellule A4 la formule «= MIR(A1, A2, A3) » et dans la cellule A5 la formule « =DIAG(A4) », et qu'on appelle la représentation graphique de ces deux cellules, en ayant bien choisi les 9 points de contrôle des trois pL3 initiales on pourrait voir par exemple une portion de sphère et sa diagonale portion de fenêtre de Viviani. Les formes pascaliennes munies de leurs opérateurs internes pourraient ainsi nourrir une sorte de tableur 3D, ... qui reste à écrire.

6. **Implémentation informatique**
   Enfin il se trouve que l'approche gestuelle (corde, piquets, milieu, ...) développée peut facilement être traduite en termes immédiatement compréhensibles par l'ordinateur ; un ordinateur est plus mal à l'aise dans le traitement des expressions algébriques de degré élevé que dans l'application récursive de divisions par deux, et c'est à peu près tout ce qui lui est demandé dans cette approche. Une implémentation dans le langage POVRAY (téléchargeable sur le site: http://www.povray.org) a abouti à un outil logiciel complet (POVRAY+pFlibs.inc) permettant de construire l'ensemble des pFormes (présentées dans ce travail et d'autres encore) et de contrôler "visuellement" leur validité. Plus d'informations sur cette implémentation peut être trouvée sur le site suivant: http://marty.alain.free.fr.



Un cube gauche (pV332) déformé représenté dans l'intervalle u,v,w = [0,1] et sa diagonale étendue dans l'intervalle [-infini,+infini] ; on note que l'influence de la déformation du cube gauche ne s'arrête pas aux limites de sa représentation, c'est tout l'espace qui est concerné.

En feuilletant récemment deux nouveaux ouvrage sur la géométrie projective et sur les NURBS, le premier volant très haut dans les concepts mathématiques à la Bourbaki, le second rempli de lourdes expressions indicées plus complexes les unes que les autres, donc pratiquement inaccessibles pour moi, j'ai repensé aux longues heures passées à l'université à étudier les méthodes de résolution de l'équation de Laplace dans tous les systèmes de corrdonnées possibles, et au peu que j'en avais retenu. Et puis j'ai repensé à cette manipulation de la formule de Laplace sur un simple tableur (cf 0233 lame de savon) qui m'avait été présentée à l'occasion d'un colloque Apple « Think different !», et à la découverte que cela avait été pour moi : je venais de comprendre l'équation de Laplace, l'évidente simplicité de son comportement local (une simple moyenne arithmétique, une accélération zéro) et l'importance fondamentale des conditions aux limites qui à elles seules modèlent l'essentiel de la forme résultante. Alors j'ai pensé que si, malgré ses défauts, l'approche proposée dans cet essai sur les *formes pascaliennes* pouvait avoir le même effet sur le lecteur, alors j'en serais bien heureux.

Alain Marty, Juillet 2004

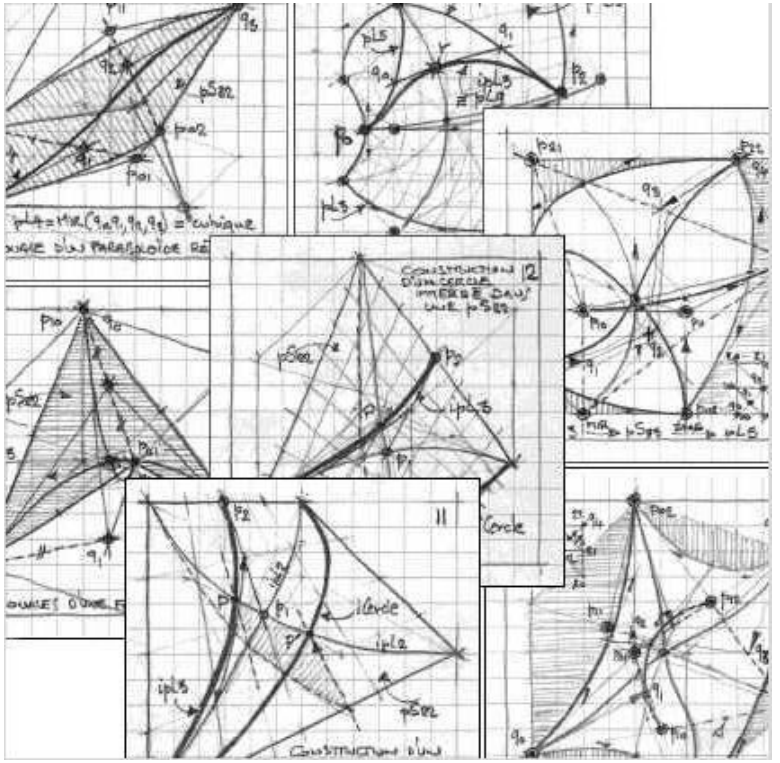
Villeneuve de la Raho

marty.alain@free.fr

références

La littérature sur le sujet est abondante. La liste réduite ci-dessous ne cite que les ouvrages qui ont vraiment accompagné cette étude et servi de guide véritable et de modèle.

- [1] GAMOV G (1946), Monsieur Tompkins au pays des Merveilles, réédité chez Dunod en 1965 : une belle histoire d'amour et de physique relativiste et quantique.
- [2] DE CASTELJAU, Paul (1959), Courbes et surfaces à pôles, Citroën, Paris : non publié à l'époque, secret industriel Citroën oblige ; Pierre Bézier a eu plus de chance chez Renault, et c'est son nom qui est resté pour les courbes et surfaces à pôles.
- [3] ROGERS DF ADAMS JA édition Mc Graw Hill (1990) Mathematical Elements for Computer Graphics : une somme en Infographie.
- [4 UPSTILL S édition Addison-Wesley (1990) The Renderman Companion : un outil logiciel de grande qualité utilisé par la société PIXAR pour ses films d'animation.
- [5] SNYDER JM édition Academic Press (1992) Generative Modeling for Computer Graphics & CAD : une approche unitaire bien construite, un vrai modèle pour une approche unitaire des formes géométriques.
- [6] FARIN G édition Masson (1992) Courbes et surfaces pour la CGAO : la base pour comprendre vraiment les courbes de Bézier et ce qui suit.
- [7] BERNARD WERBER édition de poche (1994) Les Fourmis : penser autrement !
- [8] MICHIO KAKU édition Oxford University Press (1994) Hyperspace, Through The 10th Dimension : de plus en plus de dimensions pour un univers de plus en plus simple.
- [9] LES PIEGL WAYNE TILLER édition Springer (1996) The NURBS Book : une somme sur les NURBS.
- [10] FARIN G édition AK Peters, Ltd (1999) NURBS, from Projective Geometry to Practical Use : certainement la voie à suivre, mais si difficile à comprendre ; et si les pFormes en étaient une expression plus simple ?
- [11] POVRAY, (1999) POV-Help, POV-Team, accompagnant le logiciel Open Source Libre, disponible à l'adresse : http://www.povray.org ;
-
- ... et quelques autres ouvrages utiles dont on trouvera les références dans les ouvrages bien documentés listés ci-dessus, des ouvrages sur les mathématiques pures et moins pures, sur la géométrie classique et la géométrie différentielle, sur le calcul tensoriel, les variétés riemaniennes, la relativité, la mécanique des milieux continus, l'élasticité, l'infographie, la CAO, sur les langages de programmation, et tous les outils qu'on est amené à utiliser au passage dans ce genre d'exploration...



Construction de pFormes à la main sur papier quadrillé.

implémentation

Toutes les images de formes pascaliennes ont été produites dans l'environnement open source POVRAY (http://www.povray.org). Les macros fondamentales ainsi que celles qui sont à la base des rendus des images incorporées dans cet ouvrage ont été regroupées dans deux fichiers, pFlibs.inc et pFbook.inc.

Un exemple type est présenté ci-dessous et le listing complet des deux fichiers en suivant.

Ce listing peut être téléchargé sur le site http://marty.alain.free.fr (ce qui évitera une longue entrée au clavier et les erreurs inévitables de frappe) ; en attendant un manuel/tutoriel (qui reste à écrire), la lecture des déclarations des macros interfaces (draw(), pFdiagonalisation(), etc...) pourra apporter les indications nécessaires à leur utilisation.

exemple type

Exemple type de programme utilisant les librairies POVRAY/pFlibs et produisant la figure ci-dessous :

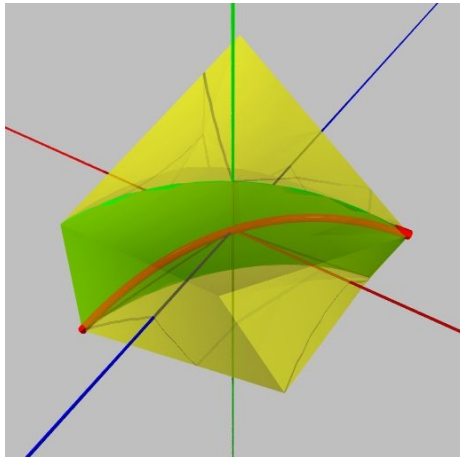


figure : cube gauche jaune transparent (pV222), sa surface diagonale verte (pS23) et la diagonale rouge de cette surface (pL4)

```
// POVRAY seul ne connait pas les pFormes,
// il faut inclure le fichier pFlibs.inc :
#include "pFlibs.inc"

//////////////////////////////////////////////////////////////

// 1) REGLAGES CONSTANTS :
// couleur du fond de l'image :
background { color rgb 3/4 }     // gris 75%
// cube unité transparent centré sur l'origine :
// box { -1/2, 1/2 pigment { color rgbt 0.9 } scale 1.0001}
// trois sources lumineuses :
light_source { <  2,2,-2 >   color rgb 1 }
light_source { < -2,2,-2 >   color rgb 1 }
light_source { <  2,0, 2 >   color rgb 1 }
// camera, soit axonometrique, soit perspective :
// vue_axonometrique( AXO, 7/8 )
vue_perspective( < 0.7,1.3,-1.1 >*6/6 )

//////////////////////////////////////////////////////////////

// 2) SCENE :
// a) construction des pFormes :
#local pL2_0 = array[2] {<-0.5,-0.5,-0.5,1.0>,<0.5,0.0,-0.5,1.0>}
#local pL2_1 = array[2] {<-0.5, 0.0, 0.5,1.0>,<0.5,-0.5,0.5,1.0>}
#local pS22  = array[2] { pL2_0, pL2_1 }
#local pS22_0 = pS22
#local pS22_1 = pS22
pFtranslate( 2, pS22_1, <0, 0.5,0> )
pFrotate( 2, pS22_1, <0,-15,0> )
#local pV222 = array[2] { pS22_0, pS22_1 }
#local pS23 = pFdiagonalisation( 3, pV222 )
#local pL4 = pFdiagonalisation( 2, pS23 )
// b) dessin des pFormes :
draw( 3, pV222, finesse(<1,3,3,0>)
      + enveloppe(LISSE) + ma_couleur(<1,1,0,0.5>) )
draw( 2, pS23, finesse(<4,4>)
      + surface(LISSE) + ma_couleur(<0,1,0,0>) )
draw( 1, pL4, finesse(5)
      + courbe( 0.02 ) + ma_couleur(<1,0,0,0>) )
```

le fichier pFlibs.inc

Les macros fondamentales sont listées dans les pages qui suivent.

```
// fichier pFlibs.inc
// sous licence GNU open source
// http://marty.alain.free.fr
// version du 01/06/2004

#macro taille( _f ) // retourne la taille d'un tableau
    dimension_size( _f,1 )
#end

/*
       AFFICHAGE DES COMPOSANTES D'UN VECTEUR
*/
#macro affiche_vect( vvv )
    #render "\n vecteur: < "
    #local n = taille( vvv );
    #local i=0; #while (i< n)
        #render concat( str( vvv[i], 4,4 ), "," )
    #local i=i+1; #end
    #render " >  \n"
#end

/*
    AFFICHAGE DES TERMES D'UNE MATRICE
*/
#macro affiche_mat( mat )    //
    #render "\n matrice:\n "
    #local n1= dimension_size( mat, 1 );
    #local n2= dimension_size( mat, 2 );
    #local i=0; #while (i< n1)        // pour chaque ligne i
        #render concat( "ligne ", str(i,0,0), ": " )
        #local j=0; #while (j< n2)    // pour chaque colonne j
            #render concat( str( mat[i][j],4,4 ), ", " )
        #local j=j+1; #end
        #render "\n"
    #local i=i+1; #end
    #render "\n"
#end


/*
    INVERSION D'UNE MATRICE CARREE
    PAR LA METHODE DE GAUSS_JORDAN
    test:
    #local mat = array[3][3] { {3,2,1}, {1,3,2}, {2,4,3} }
    affiche_mat( mat )
    #local tam = inverse_mat( mat )
    affiche_mat( tam )
    stop
*/
#macro inverse_mat( ma )
    #local N = taille(ma);
    // creation d'une matrice [N][2*N]
    #local mat = array[N][2*N]
    #local I=0; #while (I< N)
        #local J=0; #while (J< N)
            #local mat[I][J] = ma[I][J];
        #local J=J+1; #end
    #local I=I+1; #end
    #local K=0; #while (K< N)
        #local I=0; #while (I< N)
            #if (K=I)
                #local mat[I][N+K] = 1;
            #else
                #local mat[I][N+K] = 0;
            #end
        #local I=I+1; #end
    #local K=K+1; #end
    // start:
    #local EPS  = 0.0000001;
    #local singularite = false;
    #local DetA = 1;
    #local K = 0;    #while ( (K< N) & (!singularite) )
    // recherche de l'indice L du pivot maximum
        #local L=K;
        #local I=K; #while (I< N)
            #if ( abs(mat[I][K])  >   abs(mat[L][K]) )
                #local L=I;
            #end
        #local I=I+1; #end
        #if ( abs(mat[L][K]) <= EPS )
            #local singularite = true;
        #else
            #if (L!=K)
                #local I=K; #while (I< 2*N)
                    #local mmm = mat[K][I];
                    #local mat[K][I] = mat[L][I];
                    #local mat[L][I] = mmm;
                #local I=I+1; #end
                #local DetA = -DetA;
```

```
            #end
        #end
        #local DetA = DetA*mat[K][K];
    // elimination de xk
        #if (!singularite)
            #local J=K+1; #while (J< 2*N)
            #local mat[K][J] = mat[K][J]/mat[K][K];
            #local J=J+1; #end
            #local mat[K][K] = 1;
            #local I=0; #while (I< N)
                #if (I!=K)
                    #local J=K+1; #while (J< 2*N)
                    #local mat[I][J] = mat[I][J] -
                         (mat[I][K]*mat[K][J]);
                    #local J=J+1; #end
                    #local mat[I][K] = 0;
                #end
            #local I=I+1; #end
        #end
    #local K=K+1; #end
    #local tam = array[N][N]
    #local I=0; #while (I< N)
        #local J=0; #while (J< N)
            #local tam[I][J] = mat[I][J+N];
        #local J=J+1; #end
    #local I=I+1; #end
    tam
#end
#macro produit_mat( mat, vect )
    #local n= dimension_size( mat, 1 );
    #local tcev = array[n]
    #local i=0; #while (i< n)
        #local dd = 0;
        #local j=0; #while (j< n)
            #local dd = dd + mat[i][j]*vect[j];
        #local j=j+1; #end
        #local tcev[i] = dd;
    #local i=i+1; #end
    tcev
#end

/* ----------- EXPORT ET IMPORT DE FICHIERS FORMES --------- */

/*
    conserve une forme calculée sous forme texte
    utile dans le cas d'animations ou éventuellement
    pour transmettre des informations à d'autres programmes
*/

/*
    ECRITURE D,UNE COURBE DANS UN FICHIER:
    input:   la courbe et le nom du fichier
    output:  le fichier
    nota:    attention a ne pas ecraser un fichier
             de meme nom, il n'y a pas de controle...
    appel:   #local courb = creer_une_courbe(x,x,...)
             ecrire_courbe( courb, "fichier.txt" )
*/
#macro pLwrite( f, fichier )
    #fopen myfile fichier write
    #local umax = taille(f);
    #write( myfile, "\"Une pCourbe\",\n" )
    #write( myfile, umax, ",\n" )
    #local i=0; #while (i< umax)
        #write( myfile, f[i], ",\n" )
    #local i=i+1; #end
    #fclose myfile
#end

/*
    LECTURE D,UNE COURBE DEPUIS UN FICHIER:
    input:         le nom du fichier
    output:        la courbe
    nota:
    appel:         #local courb = lire_courbe( "fichier.txt" )
*/
#macro pLread( fichier )
    #fopen myfile fichier read
    #read( myfile, titre )
    #read( myfile, umax )
    #local ff = array[umax]
    #local i=0; #while (i< umax)
        #ifdef (ddd) #undef ddd #end    // peut etre a revoir
        #read( myfile, ddd )
        #local ff[i] = ddd;
    #local i=i+1; #end
    #fclose myfile
    /*return*/ ff
#end

/*
    ECRITURE D,UNE SURFACE DANS UN FICHIER:
    input:   la surface et le nom du fichier
    output:  le fichier
```

```
    nota:    attention a ne pas ecraser un fichier de meme nom
    appel:   #local surf = creer_une_surface(x,x,...)
             ecrire_surface( surf, "fichier.txt" )
*/
#macro pSwrite( f, fichier )
    #fopen myfile fichier write
    #local umax = taille(f);
    #local vmax = taille(f[0]);
    #write( myfile, "\"Une pSurface\",\n" )
    #write( myfile, umax, ",\n" )
    #write( myfile, vmax, ",\n" )
    #local i=0; #while (i< umax)
        #local j=0; #while (j< vmax)
            #write( myfile, f[i][j], ",\n" )
        #local j=j+1; #end
    #local i=i+1; #end
    #fclose myfile
#end

/*
     LECTURE D,UNE SURFACE DEPUIS UN FICHIER:
    input:   le nom du fichier
    output:  la surface
    nota:
    appel:   #local surf = lire_surface( "fichier.txt" )
*/
#macro pSread( fichier )
    #fopen myfile fichier read
    #read( myfile, titre )
    #read( myfile, umax )
    #read( myfile, vmax )
    #local ff = array[umax]
    #local pp = array[vmax]
    #local i=0; #while (i< umax)
        #local j=0; #while (j< vmax)
            #ifdef (ddd) #undef ddd #end
            #read( myfile, ddd )
            #local pp[j] = ddd;
        #local j=j+1; #end
        #local ff[i] = pp
    #local i=i+1; #end
    #fclose myfile
    /*return*/ ff
#end

/*----------- MACROS GENERALES DANS R4 ------------*/
```

```
/*
    TRANSFORMATION GENERALE AFFINE:
        pp.x      |  00 01 02 03 |    |  p.x
        pp.y   =  |  10 11 12 13 |  * |  p.y
        pp.z      |  20 21 22 23 |    |  p.z
        pp.t      |  30 31 32 33 |    |  p.t
*/
#macro pFtransform( mat, p )
< mat[0][0]*p.x + mat[0][1]*p.y + mat[0][2]*p.z + mat[0][3]*p.t,
  mat[1][0]*p.x + mat[1][1]*p.y + mat[1][2]*p.z + mat[1][3]*p.t,
  mat[2][0]*p.x + mat[2][1]*p.y + mat[2][2]*p.z + mat[2][3]*p.t,
  mat[3][0]*p.x + mat[3][1]*p.y + mat[3][2]*p.z + mat[3][3]*p.t >
#end

#macro pFtranslate( dim, f, dt )    // translation d'une pForme
    #if (dim=0)
        #local tt = f.t;  //  à partir d'une translation
                          // dans R3 < dt.x,dt.y,dt.z >
        #local pp = < f.x/tt, f.y/tt, f.z/tt >   + dt;
        #local f = < pp.x*tt, pp.y*tt, pp.z*tt, tt >  ;
    #else
        #local n = taille( f );
        #local i=0; #while (i< n)
            pFtranslate( dim-1, f[i], dt )
        #local i=i+1; #end
    #end
#end

#macro pFrotate( dim, f, dr )    // rotation d'une pForme
    #if (dim=0)
        #local tt = f.t;  //  à partir d'une rotation
                          // dans R3 < dr.x,dt.y,dt.z >
        #local pp = < f.x/tt, f.y/tt, f.z/tt >  ;
        #local rad = pi/180;
        #if (dr.x != 0)
            #local pp = <
            pp.x,
            cos(rad*dr.x)*pp.y - sin(rad*dr.x)*pp.z,
            sin(rad*dr.x)*pp.y + cos(rad*dr.x)*pp.z >  ;
        #end
        #if (dr.y != 0)
            #local pp = <
            cos(rad*dr.y)*pp.x -sin(rad*dr.y)*pp.z,
            pp.y,
            sin(rad*dr.y)*pp.x + cos(rad*dr.y)*pp.z >  ;
        #end
        #if (dr.z != 0)
```

```
            #local pp = <
            cos(rad*dr.z)*pp.x - sin(rad*dr.z)*pp.y,
            sin(rad*dr.z)*pp.x + cos(rad*dr.z)*pp.y,
            pp.z >  ;
        #end
        #local f = < pp.x*tt, pp.y*tt, pp.z*tt, tt >  ;
    #else
        #local n = taille( f );
        #local i=0; #while (i< n)
            pFrotate( dim-1, f[i], dr )
        #local i=i+1; #end
    #end
#end

#macro pFscale( dim, f, ds )    // homothetie d'une pForme
    #if (dim=0)
        #local tt = f.t;  //  à partir d'une homothetie
                          // dans R3 < ds.x,ds.y,ds.z >
        #local pp = < f.x/tt*ds.x, f.y/tt*ds.y, f.z/tt*ds.z >  ;
        #local f = < pp.x*tt, pp.y*tt, pp.z*tt, tt >  ;
    #else
        #local n = taille( f );
        #local i=0; #while (i< n)
            pFscale( dim-1, f[i], ds )
        #local i=i+1; #end
    #end
#end


/*
    MACROS INTERNES
*/

#macro _interpol( dim, f0, f1, uu )    // interpole deux formes
    #if (dim=0)
        (1-uu)*f0 + uu*f1;
    #else
        #local nb = taille( f0 );
        #local fm = array[nb]
        #local i=0; #while (i< nb)
        #local fm[i] = _interpol( dim-1, f0[i], f1[i], uu )
        #local i=i+1; #end
        fm
    #end
#end

#macro _shift( dim, n, sens, uu )
    // calcule la sous_forme droite ou gauche
    #if (n >  0)
        #local i=0; #while (i< n)
            #if (sens)            // droite
    #declare _d[i] = _interpol( dim-1, _d[i], _d[i+1], uu )
            #else                 // gauche
                #local nb = taille( _g );
                #local k = nb-1-i;
    #declare _g[k] = _interpol( dim-1, _g[k], _g[k-1], uu )
            #end
        #local i=i+1; #end
        _shift( dim, n-1, sens, uu )
    #end
#end

#macro _stretch( dim, f, p0, p1 )  // cale f entre p0.x et p1.x
    #local _g = f
    _shift( dim, taille(f)-1, false, 1-p1.x ) // travail sur g
    #local _d = _g
    _shift( dim, taille(f)-1, true, p0.x/p1.x )// travail sur d
    _d                                 // mêmes dims que f
#end

#macro _sous_subdi( dim, p, r ) // charge out et incrémente index
    #if (dim=0)
        #declare out[index] = p;
    #else
        #local pp = pFsubdivision( dim,p,< r.y, r.z, r.t, 0 > )
        #declare out[index] = pp
    #end
    #declare index=index+1;
#end

#macro _subdi( dim, f, r, uu ) // corps récursif de subdivision()
    #if (r.x=0)
        #local i=0; #while (i< nb_in-1)
            _sous_subdi( dim-1, f[i], r )
        #local i=i+1; #end
    #else
        #local _g = f
        #local _d = f
        _shift(dim, nb_in-1,true,uu)
        _shift(dim, nb_in-1,false,uu)
        _subdi( dim, _g, < r.x-1, r.y, r.z, r.t >  , uu )
        _subdi( dim, _d, < r.x-1, r.y, r.z, r.t >  , uu )
    #end
#end
```

```
/*
    3 MACROS FONDAMENTALES
*/

#macro pFsubdivision( dim, in, r )
    // retourne la forme subdivisee
    #local nb_in = taille(in);        // récursivement au 1/2
    #local nb_out = (nb_in-1)*pow(2, r.x)+1;    //
    #local out = array[nb_out]
    #local index = 0;
    _subdi(dim, in, r, 1/2)  //_subdi(dim, in, r, 1/3) sierpinsky
    _sous_subdi(dim-1, in[nb_in-1], r)
    out
#end

#macro pFstretch( dim, f, p0, p1 )
    #if (dim=1)
        _stretch( 1, f, p0, p1 )
    #else
        #local ff = _stretch( dim, f, p0, p1 )
        // ff prend f recalé entre p0.x et p1.x
        #local n = taille(ff);    // nb de sous_formes
        #local i=0; #while (i< n) // pour chaque sous_forme:
            #local ff[i] = pFstretch( dim-1, ff[i],
                    < p0.y,p0.z,0,1 >  , < p1.y,p1.z,0,1 >   )
        #local i=i+1; #end
        ff    // retourne ff recalée
    #end
#end

#macro pFgetSubForm( dim, f, uu )     // retourne la sous_forme
    #local _d = f    // generatrice en uu (uu est un REEL !!)
    _shift(dim, taille(f)-1, true, uu)
    _d[0]    // tableau unidim
#end

/*
    IMMERSION
1) get_point() retourne les coordonnées globales d'un point défini
   en coordonnées locales (p) dans la pForme f
2) immersion() transforme une ipForme définie en coordonnées
locales dans une pForme,
   en un tableau de points définis en coordonnées globales.
   Ce tableau ne définit pas une pForme (une post-subdivision
n'est pas possible)
3) voir plus loin les macros interpolation plus générales et
retournant
   les points de contrôle dans le cas de pCourbes dans une
pSurface
*/

#macro pFgetPoint( dim, f, p )
    #local qt = p.t;
    #local q = < p.x/qt, p.y/qt, p.z/qt,1 >  ;
    #switch (dim)
        #case (1)
            #local pp = pFgetSubForm( 1, f, q.x );
        #break
        #case (2)
            #local ff = pFgetSubForm( 2, f, q.y )
            #local pp = pFgetSubForm( 1, ff, q.x );
        #break
        #case (3)
            #local gg = pFgetSubForm( 3, f, q.z )
            #local ff = pFgetSubForm( 2, gg, q.y )
            #local pp = pFgetSubForm( 1, ff, q.x );
        #break
        #case (4)
            #local hh = pFgetSubForm( 4, f, q.t )
            #local gg = pFgetSubForm( 3, hh, q.z )
            #local ff = pFgetSubForm( 2, gg, q.y )
            #local pp = pFgetSubForm( 1, ff, q.x );
        #break
        #else
            #render "...nothing out there !!! "
    #end
    < pp.x*qt, pp.y*qt, pp.z*qt, pp.t*qt >  ;
#end

#macro pFimmersion( dim, f, idim, imf )
    // ATTENTION: imf est modifiée
    #if (idim=0)        // et n'est plus une pForme
        #local imf = pFgetPoint( dim, f, imf )
        // NO post-subdivision
    #else
        #local n = taille( imf );
        #local i=0; #while (i< n)
            pFimmersion( dim, f, idim-1, imf[i] )
        #local i=i+1; #end
    #end
#end

/*
```

```
    DIAGONALISATION D'UNE FORME
*/

#macro _diag_N2( M, s0, s1 )
    // diagonale d'une surface de type [N,2]
    #local pp = array[M+1]
    #local pp[0] = s0[0];
    #local pp[M] = s1[M-1];
    #local i=1;     #while (i< M)
        #local uu = i/M;
        #local pp[i] = (1-uu)*s0[i] + uu*s1[i-1];
    #local i=i+1;   #end
    pp
#end

#macro _diag_surf( surf )
    // diagonale d'une surface de type [M,N]
    #local M = taille( surf[0] );
    #local N = taille( surf );
    #if (N >  2)
        #local ss = array[N-1]
        #local j=0;         #while (j< N-1)
        #local ss[j] = _diag_N2( M, surf[j], surf[j+1] )
        #local j=j+1;    #end
        _diag_surf( ss )
    #else
        _diag_N2( M, surf[0], surf[1] )
    #end
#end

#macro pFdiagonalisation( dim, f )
    // diagonale d'une pForme quelconque
    #if (dim=2)
        _diag_surf( f )
    #else
        #local M = taille( f );    // nb de sous-formes
        #local ddd = array[M]
        #local i=0;    #while (i< M)
            #local ddd[i] = pFdiagonalisation(dim-1,f[i])
        #local i=i+1;    #end
        ddd
    #end
#end

/*
    ELEVATION DU DEGRE D'UNE FORME
*/

#macro pLup( f, dd ) // en attendant une version up_forme qq...
    #local ff = f
    #local i=0; #while (i< dd)
        #local N = taille(ff);
        #local fff = array[N+1]
        #local fff[0] = ff[0];
        #local fff[N] = ff[N-1];
        #local j=1; #while (j< N)
            #local fff[j] = j/N*ff[j-1] + (1-j/N)*ff[j];
        #local j=j+1;    #end
        #local ff = fff
    #local i=i+1;    #end
    ff
#end

#macro pSup( f, dd ) // en attendant une version up_forme qq...
    #local M = taille( f );      // nb de courbes
    #local i=0; #while (i< M)    // pour chaque courbe:
        #local f[i] = pLup( f[i], dd.u )    // élever degré
    #local i=i+1; #end
    #local N = taille( f[0] );   // nb de points d'une courbe
    #local ff = array[N]         // création d'une surface ortho
    #local i=0; #while (i< N)    // pour chaque point:
        #local M = taille( f );  // nb de courbes
        #local fff = array[M]    // création d'une courbe ortho
        #local j=0; #while (j< M)
            #local fff[j] = f[j][i];
        #local j=j+1; #end
        #local ff[i] = pLup( fff, dd.v )    // élever degré
    #local i=i+1; #end
    ff
#end

/*
    AFFICHAGE D'UNE FORME DANS LE CAS GENERAL
        cas des points, courbes, surfaces et volumes
        traité plus loin sous différentes formes
*/

#macro pFdraw( dim, f, rayon )
    // dessine les points de la pForme dans R3
    #if (dim=0)
        sphere { < f.x/f.t, f.y/f.t, f.z/f.t  >  , rayon }
    #else
        #local n = taille( f );
        #local i=0; #while (i< n)
```

```
            pFdraw( dim-1, f[i], rayon )
        #local i=i+1; #end
    #end
#end

/*------------- COURBES INTERPOLANTES ET
            COURBES IMMERGEES DANS UNE SURFACE ------------*/

//    MATRICE DE POINTS UNIFORMEMENT DISTRIBUES SUR UNE pCOURBE

#macro fact( n )
    #if (n=0)
        1;
    #else
        n*fact(n-1)
    #end
#end

#macro calcul_mat( n )                  // matrice de n points
       //#local n = n-1;                // uniformement distribues
       #local mat = array[n+1][n+1]     // sur une pLn
       #local fn = fact( n )
       #local CC = array[n+1]
       #local j=0; #while (j<n+1)       // precalul coeffs
binomiaux
             #local fnj = fact( n-j )
             #local fj  = fact( j )
             #local CC[j] = fn/(fnj*fj);
       #local j=j+1; #end
       #local EPS = 0.0000001;    // intervalle ouvert
                                  // pour pow() dans MEGAPOW !!
       #local i=0; #while (i<=n)        // pour chaque ligne
             #local j=0; #while (j<=n)  // pour chaque colonne
//     #local mat[i][j] = CC[j]*pow(1-i/n,n-j) * pow(i/n,j);
                   #local temp = CC[j];
                   #local temp = temp*pow(EPS+1-i/n,n-j);
                   #local temp = temp*pow(EPS+i/n,j);
                   #local mat[i][j] = temp;
             #local j=j+1; #end
       #local i=i+1; #end
       mat
#end

#macro ctrliCurvInSurf( surf, curv )
    // nb de points de controle
    #local nb1 = taille( surf );        // de la courbe de R4
    #local nb2 = taille( surf[0] );     // correspondant à la
courbe
    #local nb = taille( curv ); // immergée dans une surface
    ((nb1+nb2-2)*(nb-1)+1)
#end

/*
    pCOURBE et pSURFACE INTERPOLANTES
    en attente d'une formule générale pour les pFormes
*/

#macro pLinterpolante( curv )
    #local n = taille( curv );
    #local vx = array[n]
    #local vy = array[n]
    #local vz = array[n]
    #local vt = array[n]
    #local i=0; #while (i< n)
        #local vx[i] = curv[i].x;
        #local vy[i] = curv[i].y;
        #local vz[i] = curv[i].z;
        #local vt[i] = curv[i].t;
    #local i=i+1; #end
    #local mat = calcul_mat( n-1 )
    #local tam = inverse_mat( mat )
    #local _vx = produit_mat( tam, vx )
    #local _vy = produit_mat( tam, vy )
    #local _vz = produit_mat( tam, vz )
    #local _vt = produit_mat( tam, vt )
    #local icurv = array[n]
    #local i=0; #while (i< n)
        #local icurv[i] = < _vx[i], _vy[i], _vz[i], _vt[i]  >  ;
    #local i=i+1; #end
    icurv
#end

#macro pSinterpolante( surf )
    #local m = taille(surf);
    #local n = taille(surf[0]);
    #local ipLm = array[m]
    #local i=0; #while (i< m)
        #local ipLm[i] = pLinterpolante( surf[i] )
    #local i=i+1; #end
    #local temp = array[m]
    #local isurf = array[n]
    #local i = 0; #while (i< n)
        #local j=0; #while (j< m)
            #local temp[j] = ipLm[j][i];
```

```
        #local j=j+1; #end
        #local isurf[i] = pLinterpolante( temp )
    #local i=i+1; #end
    isurf
#end

//    pCOURBE IMMERGEE DANS UNE pSURFACE

#macro courbe_in_surface( courbe_immergee, surf )
    #local nb = ctrliCurvInSurf( surf, courbe_immergee );
    #local b = array[nb]
    #local i=0; #while (i< nb)
    #local pp = pFgetSubForm( 1,courbe_immergee,i/(nb-1) );
        #local b[i] = pFgetPoint( 2, surf, pp )
    #local i=i+1; #end
    #local ss = pLinterpolante( b )
    ss
#end

/* ----------- SPLINES INTERPOLANTES -----------------*/
// retourne des TABLEAUX de quadriques ou de cubiques

//    SPLINE INTERPOLANTE QUADRIQUE:

#macro spline_interpolante_quadrique( b_1, b, cercle )
    #local n = taille( b );
    #local spline = array[n-1]
    #local q = array[3]
    #local q[1] = b_1;
    #local i=0; #while (i< n-1)
        #local q[0] = b[i];
        #local q[1] = 2*b[i] - b_1;
        #local q[2] = b[i+1];
        #local spline[i] = q
        #local b_1 = q[1];
        #if (cercle)
            #local spline[i][1] = spline[i][1]*sqrt(2)/2;
        #end
    #local i=i+1; #end
    spline
#end

//    SPLINE INTERPOLANTE CUBIQUE:
#macro spline_interpolante_cubique( b_2, b_1, b )
    #local n = taille( b );
    #local spline = array[n-1]
    #local q = array[4]
    #local i=0; #while (i< n-1)
        #local q[0] = b[i];
        #local q[1] = 2*b[i] -b_1;
        #local q[2] =  4*b[i] -4*b_1 +b_2;
        #local q[3] = b[i+1];
        #local spline[i] = q
        #local b_2 = q[1];
        #local b_1 = q[2];
    #local i=i+1; #end
    spline
#end

/*----------- CREATION DE COURBES
        ET SURFACES PRIMITIVES (dans R4) ------------*/

/*
     MAILLAGES BIDIM et TRIDM GAUCHES:
input:     le nombre de points de controle sur u,v,w et la ttt
output:    une pSmn ou un pVmnp orthogonaux a points de controle
equirepartis
nota:
appel:
*/

#macro creer_ligne( n1, ttt )
    #local f = array[n1]
    #local i=0; #while (i< n1)
        #local p = < -0.5+ i/(n1-1), 0.0, 0.0, 1 > ;
        #local f[i] = p * < ttt,ttt,ttt,1 > ;
    #local i=i+1; #end
    f
#end

#macro creer_facette( n1, n2, ttt )
    #local f = array[n1]
    #local ff = array[n2]
    #local i=0; #while (i< n1)
        #local j=0; #while (j< n2)
        #local p = < -0.5+ i/(n1-1), -0.5 + j/(n2-1), 0.0, 1 > ;
        #local ff[j] = p * < ttt,ttt,ttt,1 > ;
        #local j=j+1; #end
        #local f[i] = ff
    #local i=i+1; #end
    f
#end

#macro creer_cube( n1, n2, n3, ttt )
```

```
    #local f = array[n1]
    #local ff = array[n2]
    #local fff = array[n3]
    #local i=0; #while (i< n1)
        #local j=0; #while (j< n2)
            #local k=0; #while (k< n3)
                    #local p = < -0.5+i/(n1-1),
                                 -0.5+j/(n2-1),
                                 -0.5+k/(n3-1), 1 >  ;
            #local fff[k] = p * < ttt,ttt,ttt,1 >  ;
            #local k=k+1; #end
            #local ff[j] = fff
        #local j=j+1; #end
        #local f[i] = ff
    #local i=i+1; #end
    f
#end

#macro pFmaillage( n, nn, ttt )
    #switch (n)
        #case (1)creer_ligne( nn, ttt )    #break;
        #case (2)creer_facette( nn.x, nn.y, ttt )    #break;
        #case (3)creer_cube(nn.x, nn.y, nn.z, ttt)   #break;
        #case (4)    #render "... not yet !"
    #end
#end

#macro surface_aleatoire( surf, see, delta )
    #local ran = seed( see );
    #local m = taille(surf);
    #local n = taille(surf[0]);
    #local i = 0; #while (i< n)
        #local j=0; #while (j< m)
            #local rrr = delta*rand( ran );
            pFtranslate( 0, surf[j][i], < 0,0,rrr >   )
        #local j=j+1; #end
    #local i=i+1; #end
    surf
#end

/*
     COURBES ARC DE CERCLE:
input:     le rayon du cercle
output:      une pL3 quart de cercle
             une pL4 demi-cercle
une pL5 demi_cercle, extensible au cercle complet par stretch
nota:
*/

#macro quart_cercle_3( r, coeff )
    // coeff: attirance du point médian
    #local k = sqrt(2)/2;    // -> ellipse, cercle, hyperbole
    #local p0 = < r, 0, 0, 1 >  ;
    #local p1 = < r, r, 0, 1 >  *k*coeff;
    #local p2 = < 0, r, 0, 1 >  ;
    array[3]     { p0, p1, p2 }
#end

#macro demi_cercle_4( r )
    #local p0 = <  r, 0,  0, 1 >  ;
    #local p1 = <  r, 2*r,0, 1 >  /3;
    #local p2 = < -r, 2*r,0, 1 >  /3;
    #local p3 = < -r, 0,  0, 1 >  ;
    array[4]     { p0, p1, p2, p3 }
#end

#macro demi_cercle_5( r )
    #local k = sqrt(2)/2;
    #local p0 = <  r, 0,      0, 1 >  ;
    #local p1 = <  r, r,      0, 1 >  *k;
    #local p2 = <  0, 3/2*r,0, 1 >  *2/3;
    #local p3 = < -r, r,      0, 1 >  *k;
    #local p4 = < -r, 0,      0, 1 >  ;
    array[5]     { p0, p1, p2, p3, p4 }
#end

#macro arc_cercle( n, r )
    #switch(n)
        #case (3)    quart_cercle_3( r, 1 ) #break
        #case (4)    demi_cercle_4( r ) #break
        #case (5)    demi_cercle_5( r ) #break
    #end
#end

#macro creer_cylindre( R, H )
    #local k = sqrt(2)/2;
    #local temp = array[2]
    #local temp[0] = array[3]
        {   < R, 0, 0, 1 >,
            < R, R, 0, 1 >*k,
            < 0, R, 0, 1 >
        },
    #local temp[1] = array[3]
        {   < R, 0, H, 1 >,
```

```
            < R, R, H, 1 >*k,
            < 0, R, H, 1 >
        }
    temp
#end

#macro creer_tore( R1, R2 )
    #local k = sqrt(2)/2;
    #local R12 = R1+R2;
    #local R2 = abs(R2);
    #local temp = array[3]
    #local temp[0] =   array[3]
        {   < 0,   0, -R12, 1 >,
            < 0,  R2, -R12, 1 >*k,
            < 0,  R2,  -R1, 1 >
        }
    #local temp[1] =   array[3]
        {   < R12,  0, -R12, 1 >*k,
            < R12, R2, -R12, 1 >*k*k,
            <  R1, R2, -R1,  1 >*k
        }
    #local temp[2] =   array[3]
        {   < R12,  0,  0, 1 >,
            < R12, R2,  0, 1 >*k,
            <  R1, R2,  0, 1 >
        }
    temp
#end

/*
     SURFACE PRODUIT:
input:     deux pLn planes dans Oxy, profil et section
output:    une pSmn
nota:    cas particuliers: prismes et surfaces de révolution
        profil gauche à etudier
*/

#macro cross( c1, c2 )
    #local nb1 = taille( c1 );    // courbes profil
    #local nb2 = taille( c2 );    // courbe section
    #local surf = array[nb1]      // surface produite
    #local i = 0; #while (i< nb1) // pour chaque point du profil
        #local profil = c1[i];    // un point du profil
        #local pp = array[nb2]
        #local j=0; #while (j< nb2)// pour chaque pt  section
            #local section = c2[j];// un pt de la section
        #local pp[j] = <
            section.x*profil.x,    // base sur x
              section.t*profil.y,  // le long de l'axe y
              section.y*profil.x,  // base sur z
              section.t*profil.t  >  ;
        #local j=j+1; #end
        #local surf[i] = pp
    #local i=i+1; #end
    surf
#end

/*
     SURFACE TUBULAIRE
input: une pLm chemin et une pLn section
    les valeurs de recusrion sur u et v (pre-subdivision)
output:        une pSmn (qui pourra etre subdivisee)
nota:         fonctionne avec les courbes rationnelles
*/

#macro pipe(che, sec, r1, r2)
    #local chemin  = pFsubdivision( 1, che, < r1,0,0,0 >   )
    #local section = pFsubdivision( 1, sec, < r2,0,0,0 >   )
    #local vmax = taille(chemin);
    #local umax = taille(section);
    #local tube = array[vmax]
    #local pp = array[umax]
        #local i = 0; #while (i< vmax)
            #local mat = pLgetFrenet( che, i/(vmax-1) )
            #local j = 0; #while (j< umax)
            #local pp[j] = pFtransform( mat, section[j] );
            #local j=j+1; #end
            #local tube[i] = pp
        #local i=i+1; #end
    tube
#end

/*
     SURFACE TUBULAIRE ONDULEE
    #local surf = waving_pipe( chemin,section,<4,3>,<0.5,5> )
*/

#macro waving_pipe(che, sec, r, ondulation )
    #local chemin  = pFsubdivision( 1, che, < r.x,0,0,0 >   )
    #local section = pFsubdivision( 1, sec, < r.y,0,0,0 >   )
    #local vmax = taille(chemin);
    #local umax = taille(section);
    #local tube = array[vmax]
    #local pp = array[umax]
```

```
        #local i = 0; #while (i< vmax)
            #local uu = i/(vmax-1);
            #local mat = pLgetFrenet( che, uu )
#local coeff = 1 + ondulation.x *sin( 2*pi*uu * ondulation.y );
            #local temp = mat
            #local temp[0][0] = mat[0][0]* coeff;
            #local temp[1][1] = mat[1][1]* coeff;
            #local temp[2][2] = mat[2][2]* coeff;
            #local j = 0; #while (j< umax)
        #local pp[j] = pFtransform( temp, section[j] );
            #local j=j+1; #end
            #local tube[i] = pp
        #local i=i+1; #end
    tube
#end

/*
    SURFACES PARALLELES
    crée une surface située à la distance dd d'une surface
    autres cas possibles avec modulation de dd
*/

#macro surface_parallele( surf, dd )
    #local m = taille( surf );
    #local n = taille( surf[0] );
    #local S = surf
    #local i = 0; #while (i< n)
        #local j=0; #while (j< m)
        #local mat = pSgetPijk( surf, < j/(n-1),i/(m-1),0,1 >   )
        #local nn = < mat[0][0], mat[1][0], mat[2][0]  >  ;
            #local pp = S[j][i];
            #local pt = pp.t;
            #local qq = < pp.x/pp.t, pp.y/pp.t, pp.z/pp.t >  ;
            #local qq = qq + nn*dd;
        #local S[j][i] = < qq.x*pt,qq.y*pt,qq.z*pt, pt >  ;
        #local j=j+1; #end
    #local i=i+1; #end
    S
#end

/*
    SURFACES COMBINAISONS LINEAIRES
    exemples de surfaces résultant d'une combinaison linéaire
d'autres surfaces ;
    la somme des coefficients doit être égale à 1.
    le cas le plus connu est celui des surfaces de Coons
*/

/*
    SURFACES SYMETRIQUES
    crée une surface symétrique par rapport à une autre surface
    cas particulier: symétrie plan
*/

#macro surface_symetrique( s1, s2 )    // surf = 2*s1 - s2
    #local M = taille( s1 );
    #local N = taille( s2[0] );
    #local surf = array[M]
    #local i=0; #while (i< M)
        #local pp = array[N]
        #local j=0; #while (j< N)
            #local pp[j] = 2*s1[i][j] - s2[i][j];
        #local j=j+1; #end
        #local surf[i] = pp
    #local i=i+1; #end
    surf
#end

/*
     SURFACE DE COONS:
input:     4 pLi deux a deux concourantes en quatre points
output:    une pSmn interpolant ces courbes
nota:     les 4 courbes peuvent être de degrés différents
*/

#macro creer_coons( L1, L2, L3, L4 )
    // ajuster les degrés de L1 et L2, et de L3 et L4
    #local M = taille(L1);
    #local m = taille(L2);
    #if (M >  m)    #local L2 = pLup( L2, M-m )    #end
    #if (M< m)      #local L1 = pLup( L1, m-M )    #end
    #local N = taille(L3);
    #local n = taille(L4);
    #if (N >  n)    #local L4 = pLup( L4, N-n )    #end
    #if (N< n)      #local L3 = pLup( L3, n-N )    #end
    #local M = taille(L1);
    #local N = taille(L3);
    // construire s1 (pSM2) sur L1/L2 et élever à pSMN:
    #local s1 = array[2] { L1, L2 }
    #local s1 = pSup( s1, < 0,N-2 >   )
    // construire s2 (pS2N) sur L3/L4 et élever à pSMN:
    #local s2 = array[N]
    #local i=0; #while (i< N)
        #local s2[i] = array[2] { L3[i], L4[i] }
```

```
    #local i=i+1; #end
    #local s2 = pSup( s2, < M-2,0 >   )
    // construire s3 (pS22) sur les 4 points d'angle
    // et élever à pSMN:
    #local L21 = array[2] { L1[0], L1[M-1] }
    #local L22 = array[2] { L2[0], L2[M-1] }
    #local s3 = array[2] { L21, L22 }
    #local s3 = pSup( s3, < M-2,N-2 >   )
    // construire coons = s1 + s2 - s3:
    #local coons = array[M]
    #local i=0; #while (i< M)
        #local pp = array[N]
        #local j=0; #while (j< N)
            #local pp[j] = s1[i][j] + s2[i][j] - s3[i][j];
        #local j=j+1; #end
        #local coons[i] = pp
    #local i=i+1; #end
    coons
#end

// GEODESIQUES
// macro calculant les points d'une geodesique
// input: surface, point depart, angle de tir, nombre de points
// output: tableau des points

#macro geodesique( surf, P, A, N, dt )
      #local EPS   = 1.0e-4;
      // premier point en coords locales :
      #local U    = P.x;
      #local V    = P.y;
      #local du    = EPS;
      #local dv    = EPS;
      // calcul du premier point en coords globales :
      #local f     = pFgetPoint( 2, surf, <U,V,0,1> )
      #local fdu   = pFgetPoint( 2, surf, <U+du,V,0,1> )
      #local fdv   = pFgetPoint( 2, surf, <U,V+dv,0,1> )
      #local F           = <f.x/f.t, f.y/f.t, f.z/f.t> ;
      #local FDU   = <fdu.x/fdu.t, fdu.y/fdu.t, fdu.z/fdu.t> ;
      #local FDV   = <fdv.x/fdv.t, fdv.y/fdv.t, fdv.z/fdv.t> ;
      // derivees premieres sur U et V
      #local dfdu = (FDU - F) / du;
      #local dfdv = (FDV - F) / dv;
      #local coeff = vlength( dfdu ) / vlength( dfdv );
      #local tu = cos( A*pi/180 );              // pente sur U
      #local tv = sin( A*pi/180 ) * coeff;      // pente sur V
      #local tableau = array[N]
      #local i=0; #while (i<N)
      // calcul du point en coords globales :
      #local f     = pFgetPoint( 2, surf, <U,V,0,1> )
      #local fdu   = pFgetPoint( 2, surf, <U+du,V,0,1> )
      #local fdv   = pFgetPoint( 2, surf, <U,V+dv,0,1> )
      #local f2du  = pFgetPoint( 2, surf, <U+2*du,V,0,1> )
      #local f2dv  = pFgetPoint( 2, surf, <U,V+2*dv,0,1> )
      #local fdudv = pFgetPoint( 2, surf, <U+du,V+dv,0,1> )
      #local F = <f.x/f.t, f.y/f.t, f.z/f.t> ;
      #local FDU = <fdu.x/fdu.t, fdu.y/fdu.t, fdu.z/fdu.t> ;
      #local FDV = <fdv.x/fdv.t, fdv.y/fdv.t, fdv.z/fdv.t> ;
      #local F2DU = <f2du.x/f2du.t,f2du.y/f2du.t,f2du.z/f2du.t> ;
      #local F2DV = <f2dv.x/f2dv.t,f2dv.y/f2dv.t,f2dv.z/f2dv.t> ;
      #local FDUDV = <fdudv.x/fdudv.t, fdudv.y/fdudv.t,
                                 fdudv.z/fdudv.t> ;
      #local dfdu   = (FDU - F) / du;  // derivee premiere sur U
      #local dfdv   = (FDV - F) / dv;  // derivee premiere sur V
      // derivees secondes sur UU, VV, UV :
      #local d2fduu = ((F2DU  - FDU)/du - (FDU-F)/du)/du;
      #local d2fduv = ((FDUDV - FDU)/du - (FDV-F)/du)/dv;
      #local d2fdvv = ((F2DV  - FDV)/dv - (FDV-F)/dv)/dv;
      #local U = U + tu*dt;   // point suivant en coords locales
      #local V = V + tv*dt;   // ...
      #local dudu = vdot(dfdu, dfdu);
      #local dudv = vdot(dfdu, dfdv);
      #local dvdv = vdot(dfdv, dfdv);
      #local cu = -(  tu*tu*vdot(dfdu, d2fduu)
      + 2*tu*tv*vdot(dfdu, d2fduv) + tv*tv*vdot(dfdu, d2fdvv) );
      #local cv = -(  tu*tu*vdot(dfdv, d2fduu)
      + 2*tu*tv*vdot(dfdv, d2fduv) + tv*tv*vdot(dfdv, d2fdvv) );
      #local kk = dudu*dvdv - dudv*dudv;
      #if (abs(kk)<EPS)
            #local kk = (kk>=0) ? EPS : -EPS ;
      #end
      #local gu = (cu*dvdv - cv*dudv)/kk;     // acceleration
      #local gv = (cv*dudu - cu*dudv)/kk;     // ...
      #local tu = tu + gu*dt;                 // vitesse
      #local tv = tv + gv*dt;                 // ...
      #local tableau[i] = F; // chargement du point dans tableau
      #local i=i+1; #end
      tableau      // retourne le tableau.
#end

#macro tracer_geodesique( surf, P, A, N, dt, coul, avec_normales )
      var tableau = geodesique( surf, P, A, N, dt )
            union
            {     #local i=0; #while (i<taille(tableau))
                        sphere { tableau[i], 0.01  }
```

```
                #local i=i+1; #end
                une_couleur(coul)
            }
            #if (avec_normales)
            #local i=0; #while (i<taille(tableau)-2)
                        #local p0 = tableau[i];
                        #local p1 = tableau[i+1];
                        #local p2 = tableau[i+2];
                        #local tt = p1-p0;
                        #local bb = vcross( p1-p0, p2-p0 );
                        #local nn = vcross( bb, tt );
                        #local nn = vnormalize( nn );
                        cylinder { p0,p0 - nn*0.1,
                        0.005 pigment { color rgb <1,0,0> } }
            #local i=i+1; #end
            #end
#end

/*----------- MACROS COULEUR ET TEXTURE ------------*/

//#include "golds.inc"
#declare GOLD       = 1;
#declare MIROIR     = 2;
#declare GRANIT     = 3;
#declare MARBRE     = 4;

#macro une_couleur( c )
       texture
       {      pigment { color rgbt c }
              finish {      ambient 0.2
                            diffuse 0.6
                            specular 0.9
                            roughness 0.001
                     //     reflection 0.4 }
       }
#end

#macro une_texture( choix )
    #switch (choix)
    #case (GOLD)
        texture { T_Gold_5A }
    #break
    #case (MIROIR)
        texture
        {    pigment { color rgbt < 0,0,0 >   }
              finish {     ambient 0.2
                           diffuse 0.7
                           specular 0.9
                           roughness 0.005
                           reflection 0.9 }
        }
    #break
    #case (GRANIT)
        texture
        {    pigment
            {     granite
                color_map
                {
                [ 0.0    color rgbt < 1,1,1,0.0 >   ]
                [ 0.8    color rgbt < 0.2,0.1,0,0.0 >   ]
                [ 1.0    color rgbt < 1,1,0,0.0 >   ]
                }
                scale 1/16
            }
              finish {    ambient 0.2
                          diffuse 0.6
                          specular 0.9
                          roughness 0.005
                          reflection 0.0 }
        }
    #break
    #case (MARBRE)
        texture
        {    pigment
            {     marble
                color_map
                {
                [ 0.0    color rgbt < 0.9,0.9,0.9,0.7 >   ]
                [ 1.0    color rgbt < 0.2,0.2,0.2,0.0 >   ]
                }
                scale 1/32
                //turbulence 0.5
                rotate < 0,0,90 >
            }
              finish {     ambient 0.2
                           diffuse 0.7
                           specular 0.9
                           roughness 0.005
                           reflection 0.0 }
        }
    #break
    #end
#end
```

```
/*----------- MACROS POUR LE DESSIN DES COURBES,
            DES SURFACES ET DES VOLUMES ------------*/

//    PROJECTIONS R4 - >   R3:
#macro pLprojection( f )    // cree un tableau de points dans R3

    #local n1 = taille( f );
    #local ff = array[n1]
    #local i=0; #while (i< n1)
        #local p = f[i];
        #local temp = < p.x/p.t, p.y/p.t, p.z/p.t  >  ;
        #local ff[i] = temp;
    #local i=i+1; #end
    ff
#end

#macro pSprojection( f )    // cree un tableau bidim dans R3
    #local n1 = taille( f );
    #local n2 = taille( f[0] );
    #local ff = array[n1]
    #local pp = array[n2]
    #local i=0; #while (i< n1)
        #local j=0; #while (j< n2)
            #local p = f[i][j];
            #local temp = < p.x/p.t, p.y/p.t, p.z/p.t  >  ;
            #local pp[j] = temp;
        #local j=j+1; #end
        #local ff[i] = pp
    #local i=i+1; #end
    ff
#end

#macro pLgetFrenet( curv, uu )
    #local ff = pFstretch( 1, curv, uu, uu+1 )
#local p0 = < ff[0].x/ff[0].t,ff[0].y/ff[0].t,ff[0].z/ff[0].t
>  ;
#local p1 = < ff[1].x/ff[1].t,ff[1].y/ff[1].t,ff[1].z/ff[1].t
>  ;
#local p2 = < ff[2].x/ff[2].t,ff[2].y/ff[2].t,ff[2].z/ff[2].t
>  ;
    #local tt = p1-p0;
    #local bb = vcross( tt, (p2-p0) );
    #local tt = vnormalize( tt );
    #local bb = vnormalize( bb );
    #local nn = vcross( bb, tt );
    array[4][4] {   { nn.x, bb.x, tt.x, p0.x },
                    { nn.y, bb.y, tt.y, p0.y },
                    { nn.z, bb.z, tt.z, p0.z },
                    { 0,0,0,1 } }
#end

#macro pSgetPijk( surf, p )
    #local ff = pFstretch( 2, surf,
            < p.x,p.y,p.z,p.t >  , < p.x+1,p.y+1,p.z,p.t >   )
    #local p0 = < ff[0][0].x/ff[0][0].t, ff[0][0].y/ff[0][0].t,
                    ff[0][0].z/ff[0][0].t  >  ;
    #local px = < ff[0][1].x/ff[0][1].t, ff[0][1].y/ff[0][1].t,
                    ff[0][1].z/ff[0][1].t  >  ;
    #local py = < ff[1][0].x/ff[1][0].t, ff[1][0].y/ff[1][0].t,
                    ff[1][0].z/ff[1][0].t  >  ;
    #local tx = px-p0;    #local tx = vnormalize( tx );
    #local ty = py-p0;    #local ty = vnormalize( ty );
    #local nn = vcross( tx, ty );
    array[4][4] {   { nn.x, tx.x, ty.x, p0.x },
                    { nn.y, tx.y, ty.y, p0.y },
                    { nn.z, tx.z, ty.z, p0.z },
                    { 0,0,0,1 } }
#end

#macro pSgetNormale( surf, p )
    #local ff = pFstretch( 2, surf,
            < p.x,p.y,p.z,p.t >  , < p.x+1,p.y+1,p.z,p.t >   )
    #local p0 = < ff[0][0].x/ff[0][0].t, ff[0][0].y/ff[0][0].t,
                    ff[0][0].z/ff[0][0].t  >  ;
    #local px = < ff[0][1].x/ff[0][1].t, ff[0][1].y/ff[0][1].t,
                    ff[0][1].z/ff[0][1].t  >  ;
    #local py = < ff[1][0].x/ff[1][0].t, ff[1][0].y/ff[1][0].t,
                    ff[1][0].z/ff[1][0].t  >  ;
    #local nn = vcross( px-p0, py-p0 );
    #local nn = vnormalize( nn );
    nn
#end

#macro _pScalculer_normales( surf, M, N )
    // methode exacte mais longue
    #local nn = array[M][N]// surf est la surface NON subdivisée
    #local i=0; #while (i< M)// M et N : dims surface subdivisée
        #local j=0; #while (j< N)
    #local nn[i][j] = pSgetNormale(surf,< i/(M-1),j/(N-1),0,1 > );
        #local j=j+1; #end
    #local i=i+1; #end
    nn
#end
```

```
#macro pScalculer_normales( aa )
    // methode rapide mais approchée
    #local umax = taille(aa); // tableau des normales unitaires
    #local vmax = taille(aa[0]);        //
    #local nn = array[umax][vmax]
    // calcul des nn sauf aux bords maxis
    #local i = 0; #while (i< umax-1)
        #local j = 0; #while (j< vmax-1)
            #local p0 = aa[i][j];
            #local p1 = aa[i+1][j];
            #local p2 = aa[i][j+1];
            #local nn[i][j] = vnormalize(vcross((p1-p0),(p2-p0)));
        #local j=j+1; #end
    #local i=i+1; #end
    // calcul aux bords
    #local i = 0; #while (i< umax-1)
        #local nn[i][vmax-1] = nn[i][vmax-2]*vdot(nn[i][vmax-3],
                               nn[i][vmax-2])*2-nn[i][vmax-3];
    #local i=i+1; #end
    #local i = 0; #while (i< vmax-1)
        #local nn[umax-1][i] = nn[umax-2][i]*vdot(nn[umax-2][i],
                               nn[umax-3][i])*2-nn[umax-3][i];
    #local i=i+1; #end
    #local nn[umax-1][vmax-1] =
                   nn[umax-2][vmax-2]*vdot(nn[umax-2][vmax-2],
                   nn[umax-3][vmax-3])*2 - nn[umax-3][vmax-3];
    // moyenne des normales sur deux triangles adjacents sur la
diagonale
    #local i = 1; #while (i< umax-1)
        #local j = 1; #while (j< vmax-1)
            #local nn[i][j] = (nn[i-1][j-1]
                               + 2*nn[i][j] + nn[i+1][j+1])/4;
        #local j=j+1; #end
    #local i=i+1; #end
    nn
#end

//    MACROS DE DESSIN DE COURBES, DE SURFACES ET DE VOLUMES:

#macro _pLdraw( f, rayon ) // dessine une suite de cylindres
    #local ff = pLprojection( f )
    #local n = taille( f );
    #local i = 0; #while (i< n-1)
        cylinder { ff[i], ff[i+1], rayon }
    #local i=i+1; #end
#end
```

```
#macro _pSdraw( ff, nn, qualite )
    // dessine un pFmaillage de triangles
    #local umax = taille(ff);
    #local vmax = taille(ff[0]);
    mesh
    {    #local i = 0; #while (i< umax-1)
        #local j = 0; #while (j< vmax-1)
            #local p00 = ff[i][j];
            #local p10 = ff[i+1][j];
            #local p01 = ff[i][j+1];
            #local p11 = ff[i+1][j+1];
            #if (qualite)
                #local n00 = nn[i][j];
                #local n10 = nn[i+1][j];
                #local n01 = nn[i][j+1];
                #local n11 = nn[i+1][j+1];
            #end
            #if (qualite)
        smooth_triangle { p00, n00, p10, n10, p11, n11 }
        smooth_triangle { p00, n00, p11, n11, p01, n01 }
            #else
                triangle { p00, p10, p11 }
                triangle { p00, p11, p01 }
            #end
        #local j=j+1; #end
        #local i=i+1; #end
    }
#end

#macro _pVfaces( vvv )// retourne les 6 surfaces limites d'un
volume
    #local nx = taille( vvv );
    #local ny = taille( vvv[0] );
    #local nz = taille( vvv[0][0] );
    #local surf = array[6]
    #local ss = array[nx]
    #local i=0;    #while (i< nx)
        #local pp = array[ny]
        #local j=0;    #while (j< ny)
            #local pp[j] = vvv[nx-1-i][ny-1-j][nz-1];
        #local j=j+1; #end
        #local ss[i] = pp
    #local i=i+1; #end
    #local surf[0] = ss
    #local ss = array[nx]
    #local i=0;    #while (i< nx)
        #local pp = array[nz]
```

```
        #local j=0;    #while (j< nz)
            #local pp[j] = vvv[nx-1-i][ny-1][j];
        #local j=j+1; #end
        #local ss[i] = pp
    #local i=i+1; #end
    #local surf[1] = ss
    #local ss = array[ny]
    #local i=0;    #while (i< ny)
        #local pp = array[nz]
        #local j=0;    #while (j< nz)
            #local pp[j] = vvv[nx-1][i][j];
        #local j=j+1; #end
        #local ss[i] = pp
    #local i=i+1; #end
    #local surf[2] = ss
    #local ss = array[nx]
    #local i=0;    #while (i< nx)
        #local pp = array[ny]
        #local j=0;    #while (j< ny)
            #local pp[j] = vvv[nx-1-i][j][0];
        #local j=j+1; #end
        #local ss[i] = pp
    #local i=i+1; #end
    #local surf[3] = ss
    #local ss = array[nx]
    #local i=0;    #while (i< nx)
        #local pp = array[nz]
        #local j=0;    #while (j< nz)
            #local pp[j] = vvv[nx-1-i][0][nz-1-j];
        #local j=j+1; #end
        #local ss[i] = pp
    #local i=i+1; #end
    #local surf[4] = ss
    #local ss = array[ny]
    #local i=0;    #while (i< ny)
        #local pp = array[nz]
        #local j=0;    #while (j< nz)
            #local pp[j] = vvv[0][i][nz-1-j];
        #local j=j+1; #end
        #local ss[i] = pp
    #local i=i+1; #end
    #local surf[5] = ss
    surf
#end

#macro pVdrawPijk( vol, p, longueur, rayon )
// A REVOIR, REPERE NON ORTHONORME
    // dessine le repère tangent en p au volume
    #local p0 = pFgetPoint( 3, vol, <p.x, p.y, p.z, p.t> )
    #local pu = pFgetPoint( 3, vol, <p.x+1, p.y, p.z, p.t> )
    #local pv = pFgetPoint( 3, vol, <p.x, p.y+1, p.z, p.t> )
    #local pw = pFgetPoint( 3, vol, <p.x, p.y, p.z+1, p.t> )
    #local q0 = <p0.x/p0.t, p0.y/p0.t, p0.z/p0.t> ;
       #local qu = <pu.x/pu.t, pu.y/pu.t, pu.z/pu.t> ;
       #local qv = <pv.x/pv.t, pv.y/pv.t, pv.z/pv.t> ;
       #local qw = <pw.x/pw.t, pw.y/pw.t, pw.z/pw.t> ;
    #local ti = vnormalize( qu-q0 )
    #local tj = vnormalize( qv-q0 )
    #local tk = vnormalize( qw-q0 )
    union
{ cylinder {q0,q0+ti*longueur, rayon une_couleur( < 1,0,0 >   ) }
  cylinder {q0,q0+tj*longueur, rayon une_couleur( < 0,1,0 >   ) }
  cylinder {q0,q0+tk*longueur, rayon une_couleur( < 0,0,1 >   ) }
}
#end

#macro pSdrawPijk( surf, p, longueur, rayon )
    // dessine le repère tangent en p à la surface
    #local mat = pSgetPijk( surf, p )
    #local nn = < mat[0][0], mat[1][0], mat[2][0]  >  ;
    #local tx = < mat[0][1], mat[1][1], mat[2][1]  >  ;
    #local ty = < mat[0][2], mat[1][2], mat[2][2]  >  ;
    #local p0 = < mat[0][3], mat[1][3], mat[2][3]  >  ;
    union
{ cylinder {p0,p0-nn*longueur, rayon une_couleur( < 1,0,0 >   ) }
  cylinder {p0,p0+tx*longueur, rayon une_couleur( < 0,1,0 >   ) }
  cylinder {p0,p0+ty*longueur, rayon une_couleur( < 0,0,1 >   ) }
}
#end

#macro pSdrawNormale( surf, p, longueur, rayon )
    // dessine la normale en p à la surface
    #local ff = pFstretch( 2, surf,
              < p.x,p.y,p.z,p.t >, < p.x+1,p.y+1,p.z,p.t >   )
    #local p0 = < ff[0][0].x/ff[0][0].t,
              ff[0][0].y/ff[0][0].t, ff[0][0].z/ff[0][0].t  >  ;
    #local px = < ff[0][1].x/ff[0][1].t,
              ff[0][1].y/ff[0][1].t, ff[0][1].z/ff[0][1].t  >  ;
    #local py = < ff[1][0].x/ff[1][0].t,
              ff[1][0].y/ff[1][0].t, ff[1][0].z/ff[1][0].t  >  ;
    #local nn = vcross( px-p0, py-p0 );
    #local nn = vnormalize( nn );
    cylinder { p0, p0-nn*longueur, rayon }
#end
```

```
#macro pLdrawFrenet( curv, uu, longueur, rayon )
    // dessine le repère de frenet en uu à la courbe
    #local mat = pLgetFrenet( curv, uu )
    #local nn = < mat[0][0], mat[1][0], mat[2][0]  >  ;
    #local bb = < mat[0][1], mat[1][1], mat[2][1]  >  ;
    #local tt = < mat[0][2], mat[1][2], mat[2][2]  >  ;
    #local p0 = < mat[0][3], mat[1][3], mat[2][3]  >  ;
    union
    { cylinder{ p0,p0+nn*longueur, rayon une_couleur(<0,1,0>) }
      cylinder{ p0,p0+bb*longueur, rayon une_couleur(<0,0,1>) }
      cylinder{ p0,p0+tt*longueur, rayon une_couleur(<1,0,0>) }
    }
#end

//    INTERFACE POUR LE DESSIN DES FORMES:

//    1)    GESTION DES PARAMETRES D'AFFICHAGE

#declare STANDARD       = -1;
#declare POINT          = 0;
#declare COURBE         = 1;
#declare SURFACE        = 2;
#declare ENVELOPPE      = 3;
#declare HYPER          = 4;
#declare FEUILLES       = 5;
#declare FIBRES         = 6;
#declare NORMALES       = 7;
#declare REPERE         = 8;
#declare COULEUR        = 0;
#declare TEXTURE        = 1;
#declare FACETTES       = -1;
#declare LISSE          = 0;
#declare SUPER          = 1;
#declare FORME          = POINT;
#declare ASPECT         = COULEUR;
#declare QUALITE        = FACETTES;
#declare RAYON          = 0.05;
#declare LONGUEUR       = 0.1;
#declare CHOIX_COULEUR  = < 1,0,0 >  ;
#declare CHOIX_TEXTURE  = 1;

#macro standards()
    #declare FORME          = POINT;
    #declare ASPECT         = COULEUR;
    #declare QUALITE        = FACETTES;
    #declare RAYON          = 0.05;
    #declare LONGUEUR       = 0.1;
    #declare CHOIX_TEXTURE  = 1;
    #ifdef (CHOIX_COULEUR) #undef CHOIX_COULEUR #end
    #declare CHOIX_COULEUR  = < 1,0,0 >  ;
    #ifdef (RECURSION) #undef RECURSION #end
#end

#macro finesse( r )
    #ifdef (RECURSION) #undef RECURSION #end
    #declare RECURSION = r; 0
#end

#macro point( r )
    #declare FORME = POINT; #declare RAYON = r; 0
#end

#macro courbe( r )
    #declare FORME = COURBE; #declare RAYON = r; 0
#end

#macro surface( q )
    #declare FORME = SURFACE; #declare QUALITE = q; 0
#end

#macro enveloppe( q )
    #declare FORME = ENVELOPPE; #declare QUALITE = q; 0
#end

#macro feuilles( q )
    #declare FORME = FEUILLES; #declare QUALITE = q; 0
#end

#macro fibres( r )
    #declare FORME = FIBRES; #declare RAYON = r; 0
#end

#macro normales( r, l )
    #declare FORME = NORMALES; #declare RAYON = r;
    #declare LONGUEUR = l; 0
#end

#macro repere( r, l )
    #declare FORME = REPERE; #declare RAYON = r;
    #declare LONGUEUR = l; 0
#end

#macro ma_couleur( c )
```

```
    #declare ASPECT = COULEUR;
    #ifdef (CHOIX_COULEUR) #undef CHOIX_COULEUR #end
    #declare CHOIX_COULEUR = c; 0
#end

#macro ma_texture( c )
    #declare ASPECT = TEXTURE; #declare CHOIX_TEXTURE = c; 0
#end

////////////////////////////////////////////////////////////////////
/

//    2)    APPEL DES MACROS

#macro __pSdraw( f, ff, etat )
    #switch (etat)
    #case (FACETTES)
        #local nn = ff    // pour leurer la macro _draw_surface
        _pSdraw( ff, nn, false)
    #break
    #case (LISSE)
        #local nn = pScalculer_normales( ff )
        _pSdraw( ff, nn, ((QUALITE >=LISSE) ? true : false))
    #break
    #case (SUPER)
        #local nn =
    _pScalculer_normales( f, taille(ff), taille(ff[0]) )
        _pSdraw( ff, nn, ((QUALITE >=LISSE) ? true : false))
    #break
    #end
#end

#macro pPdraw( f )
    pFdraw( 0, f, RAYON )
#end

#macro pLdraw( f )
    #ifndef (RECURSION) #declare RECURSION = 0; #end
    #local pf = pFsubdivision( 1, f, < RECURSION,0,0,0 >   )
    #local ppf = pLprojection( pf )
    #switch (FORME)
    #case (POINT)
        union { pFdraw( 1, pf, RAYON )     }
    #break
    #case (COURBE)
        union { _pLdraw( pf, RAYON ) }
    #break
    #case (REPERE)
        #local n = taille( pf );
        union
        {    #local i = 0; #while (i< n)
            pLdrawFrenet( f, i/(n-1), LONGUEUR, RAYON )
            #local i=i+1; #end
        }
    #break
    #end
#end

#macro pSdraw( f )
    #ifndef (RECURSION) #declare RECURSION = < 0,0 >  ; #end
    #local pf =
    pFsubdivision( 2, f, < RECURSION.x,RECURSION.y,0,0 >   )
    #local ppf = pSprojection( pf )
    #switch (FORME)
    #case (POINT)
        union { pFdraw( 2, pf, RAYON ) }
    #break
    #case (COURBE)
        union
        {   #local i=0; #while(i< taille(pf))
                _pLdraw( pf[i], RAYON )
            #local i=i+1; #end
        }
    #break
    #case (SURFACE)
        __pSdraw( f, ppf, QUALITE )
    #break
    #case (NORMALES)
    // il faut calculer le juste point base de la normale...
        #local m = taille(ppf);
        #local n = taille(ppf[0]);
        #local nn = _pScalculer_normales( f, m, n  )
        union
        {    #local i=0; #while(i< m)
            #local j=0; #while(j< n)
    cylinder { ppf[i][j], ppf[i][j] + nn[i][j]*LONGUEUR, RAYON }
            #local j=j+1; #end
            #local i=i+1; #end
        }
    #break
    #end
#end

#macro pVdraw( f )
```

```
#ifndef (RECURSION) #declare RECURSION = < 0,0,0 >  ; #end
#switch (FORME)
#case (POINT)
    #local pf = pFsubdivision( 3, f, <
                RECURSION.x,RECURSION.y,RECURSION.z,0 >   )
    union { pFdraw( 3, pf, RAYON ) }
#break
#case (ENVELOPPE)
    #local surf = _pVfaces( f )
    union
    {   #local i=0; #while (i< 6)
        #local psurfi = pFsubdivision( 2, surf[i], <
                 RECURSION.x,RECURSION.y,RECURSION.z,0 > )
        #local ffi = pSprojection( psurfi )
        __pSdraw( surf[i], ffi, QUALITE )
        #local i=i+1;    #end
    }
#break
#case (FEUILLES)
     #local pf = pFsubdivision(3,f,RECURSION )
    union
    {   #local n = taille(pf);
        #local i=0; #while (i< n)
        #local psurfi = pf[i]
        #local ffi = pSprojection( psurfi )
        __pSdraw( pf[i], ffi, QUALITE )
        #local i=i+1; #end
    }
#break
#case (FIBRES)
    #local pf = pFsubdivision( 3, f, <
                 RECURSION.x,RECURSION.y,RECURSION.z,0 >  )
    union
    {   #local nb = taille(pf);
        #local m = taille(pf[0]);
        #local n = taille(pf[0][0]);
        #local i=0; #while (i< m)
        #local j=0; #while (j< n)
            #local poil = array[nb]
            #local k=0; #while (k< nb)
                #local poil[k] = pf[k][i][j];
            #local k=k+1; #end
            _pLdraw( poil, RAYON )
        #local j=j+1; #end
        #local i=i+1; #end
    }
#break
    #end
#end

#macro pHdraw( f )
    #ifndef (RECURSION) #declare RECURSION = < 0,0,0,0 >  ; #end
    #local pf = pFsubdivision( 4, f, RECURSION )
    union { pFdraw( 4, pf, RAYON ) }
#end

//    2)    MACRO FONDAMENTALE

#macro draw( dim, f, options )
    object
    {    #switch (dim)
            #case (0)    pPdraw( f )    #break
            #case (1)    pLdraw( f )    #break
            #case (2)    pSdraw( f )    #break
            #case (3)    pVdraw( f )    #break
            #case (4)    pHdraw( f )    #break
            #else    #render "...nothing out there! Yet..."
        #end
        #if (ASPECT = COULEUR)
            une_couleur( CHOIX_COULEUR ) #end
        #if (ASPECT = TEXTURE)
            une_texture( CHOIX_TEXTURE ) #end
    }
    standards() // réinitialise les valeurs des paramètres
#end

/*----------- MACROS UTILES ------------*/

#macro axes( Ox, Oy, Oz )
    #if (Ox) cylinder { < -10,0,0 >  ,< 10,0,0 >  , 0.005
             pigment{ color rgb  < 1,0,0 >   } }     #end
    #if (Oy) cylinder { < 0,-10,0 >  , < 0,10,0 >  , 0.005
             pigment { color rgb  < 0,1,0 >   } }    #end
    #if (Oz) cylinder { < 0,0,-10 >  , < 0,0,10 >  , 0.005
             pigment { color rgb  < 0,0,1 >   } }    #end
#end

#declare OX  = 1;
#declare OY  = 2;
#declare OZ  = 3;
#declare AXO = 4;

#macro vue_axonometrique( axe, zoom )
    camera
```

```
    {    orthographic
        #switch (axe)
            #case (OX)    location < 10,0,0 >        #break
            #case (OY)    location < 0,10,0 >        #break
            #case (OZ)    location < 0,0,-10 >       #break
            #case (AXO)   location < 10,10,-10 >     #break
        #end
        right < 1,0,0 >
        look_at < 0,0,0 >   scale 1/zoom
    }
    #switch (axe)
        #case (OX)    axes( false, true, true ) #break
        #case (OY)    axes( true, false, true ) #break
        #case (OZ)    axes( true, true, false ) #break
        #case (AXO)   axes( true, true, true )  #break
    #end
#end

#macro vue_perspective( observateur )
    camera {    location observateur
                right < 1,0,0 > look_at < 0,0,0 >   }
    axes( true, true, true )
#end

#macro sol( hauteur )
    plane
    {     < 0,1,0 >  , hauteur
        texture
        {    pigment {  color rgb 0.5 }
        // checker color rgb 0.3 color rgb 0.7 scale 1/4
            finish {     ambient 0.2
                    diffuse 0.5
                    specular 0.7
                    roughness 0.01
                    reflection 0.8
                }
        }
    }
#end

#macro ciel()
    sky_sphere
    {     pigment
        {     marble    // gradient y
            color_map {
                    [0.0 color rgb 1/4]
                    [1.0 color rgb 3/4]
                }
            //scale 1/2
            turbulence 0.5
            rotate < 0,0,90 >
        }
    }
#end

//    THE END, the show can start now...
```

le fichier pFbook.inc

Les macros utilisées pour rendre les images incorporées dans cet ouvrage sont listées en reprenant la structure des différentes sections . Elles illustrent l'unicité de l'approche et peuvent servir de base à de nouvelles explorations.

```
// fichier pFbook.inc
// sous licence GNU open source
// http://marty.alain.free.fr
// version du 01/06/2004

// 11 : FORMES MULTILINEAIRES RECURSIVES

#macro pF_111() // 4 points
    draw(0,<-0.5,-0.5,-0.5,1.0>,point(0.1)+ma_couleur(<1,0,0,0>))
    draw(0,< 0.5, 0.5, 0.5,1.0>, point(0.1) + ma_texture(GRANIT))
    draw(0,< 0.5, 0.5,-0.5,1.0>, point(0.1) + ma_texture(MARBRE))
    draw(0,< 0.0, 0.0, 0.0, 1.0>, point(0.3) + ma_texture(GOLD))
#end

#macro pF_112( aspect ) // pL2
    #local p0 = <-0.5,-0.5,-0.5, 1.0> ;
    #local p1 = < 0.5,-0.0,-0.5, 1.0> ;
    #local pL2 = array[2] { p0, p1 }
    #switch (aspect)
    #case (0) draw( 1, pL2,
       finesse(0) + point(0.02) + ma_couleur(<1,1,0,0>) ) #break
    #case (1) draw( 1, pL2,
       finesse(1) + point(0.02) + ma_couleur(<1,1,0,0>) ) #break
    #case (2) draw( 1, pL2,
       finesse(2) + point(0.02) + ma_couleur(<1,1,0,0>) ) #break
    #case (3) draw( 1, pL2,
       finesse(3) + point(0.02) + ma_couleur(<1,1,0,0>) ) #break
    #case (4) draw( 1, pL2,
       finesse(0) + courbe(0.02) + ma_couleur(<1,1,0,0>) ) #break
    #end
#end

#macro pF_113( aspect ) // pS22
    #local pL2_0 = array[2] { <-0.5,-0.5,-0.5, 1.0>,
                                        < 0.5, 0.0,-0.5, 1.0> }
    #local pL2_1 = array[2] { <-0.5, 0.0, 0.5, 1.0>,
                                    < 0.5,-0.5, 0.5, 1.0> }
    #local pS22  = array[2] { pL2_0, pL2_1 }
    #switch (aspect)
    #case (0) draw( 2, pS22,
       finesse(<0,0>) + courbe(0.01) + ma_couleur(<1,1,0,0>) )
#break
    #case (1) draw( 2, pS22,
       finesse(<1,0>) + courbe(0.01) + ma_couleur(<1,1,0,0>) )
#break
    #case (2) draw( 2, pS22,
       finesse(<2,0>) + courbe(0.01) + ma_couleur(<1,1,0,0>) )
#break
    #case (3) draw( 2, pS22,
       finesse(<3,0>) + courbe(0.01) + ma_couleur(<1,1,0,0>) )
#break
    #case (4) draw( 2, pS22,
       finesse(<3,3>)+surface(LISSE)+ma_couleur(<1,1,0,0.5>))
#break
    #case (5) draw( 2, pS22,
       finesse(<3,3>)+point(0.01) + ma_couleur(<1,1,0,0.5>) )
#break
    #case (6) draw( 2, pS22,
       finesse(<3,3>)+normales(0.01,-0.1)+ma_couleur(<1,1,0,0.5>))
#break
    #end
#end

#macro pF_114( aspect ) // pV222
    #local pL2_0 = array[2] { <-0.5,-0.5,-0.5, 1.0>,
                                          < 0.5, 0.0,-0.5, 1.0> }
    #local pL2_1 = array[2] { <-0.5, 0.0, 0.5, 1.0>,
                                          < 0.5,-0.5, 0.5, 1.0> }
    #local pS22  = array[2] { pL2_0, pL2_1 }
    #local pS22_0 = pS22
    #local pS22_1 = pS22    pFtranslate( 2, pS22_1, <0, 0.5,0> )
pFrotate( 2, pS22_1, <0,-15,0> )
    #local pV222 = array[2] { pS22_0, pS22_1 }
    #switch (aspect)
    #case (0) draw( 3, pV222, finesse(<0,3,3,0>) + feuilles(LISSE)
+ ma_couleur(<1,1,0,0>) ) #break
    #case (1) draw( 3, pV222, finesse(<1,3,3,0>) + feuilles(LISSE)
+ ma_couleur(<1,1,0,0>) ) #break
    #case (2) draw( 3, pV222, finesse(<2,3,3,0>) + feuilles(LISSE)
+ ma_couleur(<1,1,0,0>) ) #break
    #case (3) draw( 3, pV222, finesse(<3,3,3,0>) + feuilles(LISSE)
+ ma_couleur(<1,1,0,0>) ) #break
    #case (4) draw( 3, pV222, finesse(<2,3,3,0>)+enveloppe(LISSE)
+ ma_couleur(<1,1,0,0.5>) ) #break
    #case (5) draw( 3, pV222, finesse(<0,3,3,0>) + fibres(0.01) +
ma_couleur(<1,1,0,0>) ) #break
```

```
    #case (6) draw( 3, pV222, finesse(<3,3,3,0>) + point(0.01) +
ma_couleur(<1,0,0,0>) ) #break
    #end
#end

#macro pF_115() // pH2222
    #local pL2_0 = array[2] { <-0.5,-0.5,-0.5, 1.0>,
                                        < 0.5, 0.0,-0.5, 1.0> }
    #local pL2_1 = array[2] { <-0.5, 0.0, 0.5, 1.0>,
                                        < 0.5,-0.5, 0.5, 1.0> }
    #local pS22  = array[2] { pL2_0, pL2_1 }
    #local pS22_0 = pS22
    #local pS22_1 = pS22
      pFtranslate( 2, pS22_1, <0, 0.5,0> )
      pFrotate( 2, pS22_1, <0,-15,0> )
    #local pV222 = array[2] { pS22_0, pS22_1 }
    #local pV222_0 = pV222    pFscale( 3, pV222_0, <1.0,1.8,1.0> )
    #local pV222_1 = pV222    pFscale( 3, pV222_1, <0.8,0.8,0.8> )
    #local pH2222 = array[2] { pV222_0, pV222_1 }
    draw( 4, pH2222,
      finesse(<2,3,3,3>) + point(0.01) + ma_couleur(<1,1,0,0>) )
#end

// 12 : DIAGONALISATION

#macro pF_121( aspect ) // pS22 et pL3
    #local pL2_0 = array[2] { <-0.5,-0.5,-0.5, 1.0>, < 0.5, 0.0,-
0.5, 1.0> }
    #local pL2_1 = array[2] { <-0.5, 0.5, 0.5, 1.0>, < 0.5,-0.5,
0.5, 1.0> }
    #local pS22  = array[2] { pL2_0, pL2_1 }
    #local pL3 = pFdiagonalisation( 2, pS22 )
    draw( 2, pS22,
      finesse(<3,3>) + surface(LISSE) + ma_couleur(<1,1,0,0.5>) )
    #switch (aspect)
    #case (0) draw( 1, pL3,
      finesse(0) + point(0.02) + ma_couleur(<1,0,0,0>) ) #break
    #case (1) draw( 1, pL3,
      finesse(1) + point(0.02) + ma_couleur(<1,0,0,0>) ) #break
    #case (2) draw( 1, pL3,
      finesse(2) + point(0.02) + ma_couleur(<1,0,0,0>) ) #break
    #case (3) draw( 1, pL3,
      finesse(3) + point(0.02) + ma_couleur(<1,0,0,0>) ) #break
    #case (4) draw( 1, pL3,
             finesse(6) + courbe(0.02) + ma_couleur(<1,1,0,0>) )
      draw( 1, pL3,
             finesse(0) + point(0.04) + ma_couleur(<1,0,0,0>) )
    #break
    #end
#end

#macro pF_122( aspect ) // pS23 et pL4
    #local pL3_0 = array[3] { <-0.5,-0.5,-0.5, 1.0>,
              < 0.0, 0.5,-0.5, 1.0>, < 0.5,-0.5,-0.5, 1.0> }
    #local pL3_1 = array[3] { <-0.5, 0.5, 0.5, 1.0>,
             < 0.0,-0.5, 0.5, 1.0>, < 0.5, 0.5, 0.5, 1.0> }
    #local pS23  = array[2] { pL3_0, pL3_1 }
    #local pL4 = pFdiagonalisation( 2, pS23 )
    #switch (aspect)
    #case (0) draw( 2, pS23, finesse(<0,4>) + courbe(0.01) +
ma_couleur(<1,1,0,0>) ) #break
    #case (1) draw( 2, pS23, finesse(<1,4>) + courbe(0.01) +
ma_couleur(<1,1,0,0>) ) #break
    #case (2) draw( 2, pS23, finesse(<2,4>) + courbe(0.01) +
ma_couleur(<1,1,0,0>) ) #break
    #case (3) draw( 2, pS23, finesse(<3,4>) + courbe(0.01) +
ma_couleur(<1,1,0,0>) ) #break
    #case (4) draw( 2, pS23, finesse(<4,4>) + surface(LISSE) +
ma_couleur(<1,1,0,0.5>) ) #break
    #end
    #switch (aspect)
    #case (0) draw( 1, pL4, finesse(0) + point(0.02)
                    + ma_couleur(<1,0,0,0>) ) #break
    #case (1) draw( 1, pL4, finesse(1) + point(0.02)
                    + ma_couleur(<1,0,0,0>) ) #break
    #case (2) draw( 1, pL4, finesse(2) + point(0.02)
                    + ma_couleur(<1,0,0,0>) ) #break
    #case (3) draw( 1, pL4, finesse(3) + point(0.02)
                    + ma_couleur(<1,0,0,0>) ) #break
    #case (4)
      draw( 1, pL4, finesse(5) + courbe(0.02)
                    + ma_couleur(<1,1,0,0>) )
      draw( 1, pL4, finesse(0) + point(0.04)
                    + ma_couleur(<1,0,0,0>) )
    #break
    #end
#end

#macro pF_123() // pV222, pS23 et pL4
    #local pL2_0 = array[2] { <-0.5,-0.5,-0.5, 1.0>,
                                        < 0.5, 0.0,-0.5, 1.0> }
    #local pL2_1 = array[2] { <-0.5, 0.0, 0.5, 1.0>,
                                        < 0.5,-0.5, 0.5, 1.0> }
    #local pS22  = array[2] { pL2_0, pL2_1 }
```

```
    #local pS22_0 = pS22
    #local pS22_1 = pS22  pFtranslate( 2, pS22_1, <0, 0.5,0> )
                          pFrotate( 2, pS22_1, <0,-15,0> )
    #local pV222 = array[2] { pS22_0, pS22_1 }
    #local pS23 = pFdiagonalisation( 3, pV222 )
    #local pL4 = pFdiagonalisation( 2, pS23 )
    draw( 3, pV222, finesse(<1,3,3,0>) + enveloppe(LISSE)
                    + ma_couleur(<1,1,0,0.5>) )
    draw( 2, pS23, finesse(<4,4>) + surface(LISSE)
                    + ma_couleur(<0,1,0,0>) )
    draw( 1, pL4, finesse(5) + courbe( 0.02 )
                    + ma_couleur(<1,0,0,0>) )
#end

#macro pF_124( aspect ) // pS33 et pL5
    #local pL3_0 = array[3] { <-0.5,-0.5,-0.5, 1.0>,
              < 0.0, 0.5,-0.5, 1.0>, < 0.5,-0.5,-0.5, 1.0> }
    #local pL3_1 = array[3] { <-0.5, 0.5,-0.0, 1.0>,
              < 0.0, 0.5,-0.0, 1.0>, < 0.5,-0.5,-0.0, 1.0> }
    #local pL3_2 = array[3] { <-0.5, 0.5, 0.5, 1.0>,
              < 0.0,-0.5, 0.5, 1.0>, < 0.5, 0.5, 0.5, 1.0> }
    #local pS33  = array[3] { pL3_0, pL3_1, pL3_2 }
    #local pL5 = pFdiagonalisation( 2, pS33 )
    #switch (aspect)
    #case (0) draw( 2, pS33, finesse(<0,4>) + courbe(0.01) +
ma_couleur(<1,1,0,0>) ) #break
    #case (1) draw( 2, pS33, finesse(<1,4>) + courbe(0.01) +
ma_couleur(<1,1,0,0>) ) #break
    #case (2) draw( 2, pS33, finesse(<2,4>) + courbe(0.01) +
ma_couleur(<1,1,0,0>) ) #break
    #case (3) draw( 2, pS33, finesse(<3,4>) + courbe(0.01) +
ma_couleur(<1,1,0,0>) ) #break
    #case (4) draw( 2, pS33, finesse(<4,4>) + surface(LISSE) +
ma_couleur(<1,1,0,0.5>) ) #break
    #end
    #switch (aspect)
    #case (0) draw( 1, pL5, finesse(0) + point(0.02)
                        + ma_couleur(<1,0,0,0>) ) #break
    #case (1) draw( 1, pL5, finesse(1) + point(0.02)
                        + ma_couleur(<1,0,0,0>) ) #break
    #case (2) draw( 1, pL5, finesse(2) + point(0.02)
                        + ma_couleur(<1,0,0,0>) ) #break
    #case (3) draw( 1, pL5, finesse(3) + point(0.02)
                        + ma_couleur(<1,0,0,0>) ) #break
    #case (4)
      draw( 1, pL5, finesse(5) + courbe(0.02)
                        + ma_couleur(<1,1,0,0>) )
      draw( 1, pL5, finesse(0) + point(0.04)
                        + ma_couleur(<1,0,0,0>) )
    #break
    #end
#end

// 13 : GENERALISATION, LES pFORMES

#macro pF_130() // pV333, pS35 et pL7
    #local pL3_0 = array[3] { <-0.5,-0.5,-0.5, 1.0>,
              < 0.0, 0.5,-0.5, 1.0>, < 0.5,-0.5,-0.5, 1.0> }
    #local pL3_1 = array[3] { <-0.5, 0.5,-0.0, 1.0>,
              < 0.0, 0.5,-0.0, 1.0>, < 0.5,-0.5,-0.0, 1.0> }
    #local pL3_2 = array[3] { <-0.5, 0.5, 0.5, 1.0>,
              < 0.0,-0.5, 0.5, 1.0>, < 0.5, 0.5, 0.5, 1.0> }
    #local pS33  = array[3] { pL3_0, pL3_1, pL3_2 }
    #local pS33_0 = pS33  pFtranslate( 2, pS33_0, <0,-0.25,0> )
    #local pS33_1 = pS33  pFscale( 2, pS33_1, <1.5,1,1.5> )
    #local pS33_2 = pS33  pFtranslate( 2, pS33_2, <0, 0.25,0> )
      #local pV333 = array[3] { pS33_0, pS33_1, pS33_2 }
      #local pS35 = pFdiagonalisation( 3, pV333 )
                        // 3 et (3+3-2)*(2-1)+1 = 5 -> 35
      //#local pS35 = pFstretch( 2, pS35, <-0.1,-0.1,0,1>,
                                          <1.1,1.1,0,1> )
      #local pL7 = pFdiagonalisation( 2, pS35 )
                        // (3+5-2)*(2-1)+1 = 7
//    #local pL7 = pFstretch( 1, pL7, <-0.1,0,0,1>, <1.1,0,0,1> )

    draw( 3, pV333, finesse(<3,3,3,0>) + fibres(0.005)
                        + ma_couleur(<1,1,1,0.5>) )
//    draw( 3, pV333, finesse(<3,3,3,0>)
            + enveloppe(LISSE) + ma_couleur(<1,1,1,0.8>) )
      //draw( 3, pV333, finesse(<2,3,3,0>)
            + feuilles(LISSE) + ma_couleur(<1,1,1,0.8>) )
      //draw( 3, pV333, finesse(<3,3,3,0>)
      + fibres(0.005) + ma_couleur(<1,1,1,0>) )
    draw( 2, pS35, finesse(<2,4>) + surface(LISSE)
                                  + ma_couleur(<0,1,0,0.5>) )
    //draw( 2, pS35, finesse(<2,4>) + courbe(0.01)
                                  + ma_couleur(<1,1,0,0.5>) )
    draw( 1, pL7, finesse(5) + courbe(0.02)
                                  + ma_couleur(<1,0,0,0>) )

    draw( 2, pS35, finesse(<0,0>) + point(0.02)
                                  + ma_couleur(<1,1,0,0>) )
    //draw( 1, pL7, finesse(0) + courbe(0.005)
                                  + ma_couleur(<1,1,1,0>) )
```

```
    draw( 1, pL7, finesse(0) + point(0.03)
                              + ma_couleur(<1,0,0,0>) )
#end

// 21 : OPERATIONS FONDAMENTALES

#macro pF_212()     // ondulations sur un arc de cercle
       cylinder { <0,-0.5,0>, <0,0.5,0>, 0.5
                          pigment { color rgb <1,1,1,0.5> } }
       #local arc = quart_cercle_3( 0.5, 1 )
       pFrotate( 1, arc, <-90,0,0> )
       #local ond = pLup( arc, 2 )
                              //
       #local n = taille(ond);
       #local i=0; #while (i<n)
             pFtranslate( 0, ond[i], <0,0.5*sin(2*pi*i/(n-1)),0> )
       #local i=i+1; #end
       draw( 1, ond, finesse(0) + point(0.03)
                              + ma_couleur(<1,0,0>) )
       draw( 1, ond, finesse(5) + courbe(0.01)
                              + ma_couleur(<1,1,0>) )
#end

#macro pF_2131() // pL4 reparamétrisée
       #local pL4 = array[4] {   <-1/2,-1/2,-1/2,1>,
       < 1/2,-1/2,-1/2,1>, < 1/2, 1/2,-1/2,1>, < 1/2, 1/2, 1/2,1> }
       draw( 1, pL4, finesse(0) +point(0.05)+ma_couleur(<1,0,0>) )
       draw( 1, pL4, finesse(5) +courbe(0.01)+ma_couleur(<1,0,0>) )
       #local pL4 = pFstretch( 1, pL4, < 1/4,0,0,1>, <3/4,0,0,1> )
       draw( 1, pL4, finesse(0) + point(0.05)
                              + ma_couleur(<0,1,0,0.5>) )
       draw( 1, pL4, finesse(5) + courbe(0.02)
                              + ma_couleur(<0,1,0,0.5>) )
#end

#macro pF_2132() // pS35 reparamétrisée
    #local pL3_0 = array[3] { <-0.5,-0.5,-0.5, 1.0>,
              < 0.0, 0.5,-0.5, 1.0>, < 0.5,-0.5,-0.5, 1.0> }
    #local pL3_1 = array[3] { <-0.5, 0.5,-0.0, 1.0>,
              < 0.0, 0.5,-0.0, 1.0>, < 0.5,-0.5,-0.0, 1.0> }
    #local pL3_2 = array[3] { <-0.5, 0.5, 0.5, 1.0>,
              < 0.0,-0.5, 0.5, 1.0>, < 0.5, 0.5, 0.5, 1.0> }
    #local pS33  = array[3] { pL3_0, pL3_1, pL3_2 }
      draw( 2, pS33, finesse(<0,0>) + point(0.02)
                           + ma_couleur(<1,1,0,0.5>) )
      draw( 2, pS33, finesse(<3,3>) + surface(LISSE)
                           + ma_couleur(<1,1,0,0.5>) )
      #local pS33 = pFstretch( 2, pS33,
                           <1/8,1/8,0,1>, <7/8,7/8,0,1> )
      pFtranslate( 2, pS33, <0,0.01,0> )
      draw( 2, pS33, finesse(<0,0>) + point(0.02)
                           + ma_couleur(<1,1,1,0.5>) )
      draw( 2, pS33, finesse(<3,3>) + surface(LISSE)
                           + ma_couleur(<1,1,1,0.5>) )
#end

#macro pF_214( ) // pFgetSubForm, pFgetPoint, pFgetPijk
    #local pL3_0 = array[3] { <-0.5,-0.5,-0.5, 1.0>,
              < 0.0, 0.5,-0.5, 1.0>, < 0.5,-0.5,-0.5, 1.0> }
    #local pL3_1 = array[3] { <-0.5, 0.5,-0.0, 1.0>,
              < 0.0, 0.5,-0.0, 1.0>, < 0.5,-0.5,-0.0, 1.0> }
    #local pL3_2 = array[3] { <-0.5, 0.5, 0.5, 1.0>,
              < 0.0,-0.5, 0.5, 1.0>, < 0.5, 0.5, 0.5, 1.0> }
    #local pS33  = array[3] { pL3_0, pL3_1, pL3_2 }
    draw( 2, pS33, finesse(<4,4>) + surface(LISSE)
                           + ma_couleur(<1,1,0,0.5>) )
    draw( 2, pS33, finesse(<3,3>) + normales(0.01,-0.05)
                           + ma_couleur(<1,1,0,0.8>) )

    #local pL3 = pFgetSubForm( 2, pS33, 1/4 )
    draw( 1, pL3, finesse(5) + courbe(0.02)
                           + ma_couleur(<1/2,1/2,1/2,0>) )

    #local p = pFgetPoint( 2, pS33, <1/4,1/4,0,1> )
    draw( 0, p, point(0.05) + ma_couleur(<1,0,0,0.5>) )

    pSdrawPijk( pS33, <4/6,1/3,0,1>, 0.3, 0.01 )
    pLdrawFrenet( pL3, 1/4, 0.3, 0.01 )
#end

// 22 : IMMERSIONS

// 221 : INTERPOLATION

#macro pF_2211() // pL5 interpolante
      #local pL5 = array[5] {
      <-1/2,-1/2,-1/2,1>, < 1/2,-1/2,-1/2,1>,
      < 1/2, 1/2,-1/2,1>, < 1/2, 1/2, 1/2,1>, < 1/2,-1/2, 1/2,1> }
      draw( 1, pL5, finesse(0) + point(0.05)+ma_couleur(<1,0,0>) )
      #local pL5 = pLinterpolante( pL5 )
      #local pL5 = pFstretch( 1, pL5, <-0.03,0,0,1>,
<1.03,0,0,1> )
      draw( 1, pL5, finesse(5) + courbe(0.01)+ma_couleur
(<1,1,0>) )
```

```
#end

#macro pF_2212() // pS35 interpolante
      #local pL3 = array[3] { <-1/2, 0,-1/2,1>,
                          < 0, 0,-1/2,1>, < 1/2, 0,-1/2,1> }
      #local pL3_0 = pL3 pFtranslate( 1, pL3_0, <0,0,0/4> )
      pFtranslate( 0, pL3_0[1], <0,0,-1/4> )
      #local pL3_1 = pL3 pFtranslate( 1, pL3_1, <0,0,1/4> )
      pFtranslate( 0, pL3_1[1], <0,-1/4,0> )
      #local pL3_2 = pL3 pFtranslate( 1, pL3_2, <0,0,2/4> )
      pFtranslate( 0, pL3_2[2], < 1/8,0,0> )
      #local pL3_3 = pL3 pFtranslate( 1, pL3_3, <0,0,3/4> )
      pFtranslate( 0, pL3_3[1], <0, 1/4,0> )
      #local pL3_4 = pL3 pFtranslate( 1, pL3_4, <0,0,4/4> )
      pFtranslate( 0, pL3_4[1], <0,0, 1/4> )
      #local pS35 = array[5] { pL3_0, pL3_1, pL3_2, pL3_3, pL3_4 }

      draw( 2, pS35, finesse(<0,0>) + point(0.04)
                              + ma_couleur(<1,0,0>) )
      #local pS35 = pSinterpolante( pS35 )
      #local pS35 = pFstretch( 2, pS35,
                          <-0.06,-0.06,0,1>, <1.06,1.06,0,1> )
      draw( 2, pS35, finesse(<4,4>) + surface(LISSE)
                              + ma_texture(MARBRE) )
      #local pL7 = pFdiagonalisation( 2, pS35 )
      #local pL7 = pFstretch( 1, pL7, <-0.3,0,0,1>, <1.01,0,0,1> )
      draw( 1, pL7, finesse(6)+courbe(0.01)+ma_couleur(<1,1,0>) )
#end

#macro pF_2235() // pL3 immergée dans une PS33
    #local pL3_0 = array[3] { <-0.5,-0.5,-0.5, 1.0>,
              < 0.0, 0.5,-0.5, 1.0>, < 0.5,-0.5,-0.5, 1.0> }
    #local pL3_1 = array[3] { <-0.5, 0.5,-0.0, 1.0>,
              < 0.0, 0.5,-0.0, 1.0>, < 0.5,-0.5,-0.0, 1.0> }
    #local pL3_2 = array[3] { <-0.5, 0.5, 0.5, 1.0>,
              < 0.0,-0.5, 0.5, 1.0>, < 0.5, 0.5, 0.5, 1.0> }
    #local pS33  = array[3] { pL3_0, pL3_1, pL3_2 }

       #local p0 = <1/4,1/8,0,1> ;
       #local p1 = <1/8,3/4,0,1> ;
       #local p2 = <7/8,2/4,0,1> ;
       #local q0 = (p0+p1)/2 ;
       #local q1 = (p1+p2)/2 ;
       #local r0 = (q0+q1)/2 ;

       #local pL2_0 = array[2] { p0, p1 }
       #local pL2_1 = array[2] { p1, p2 }
       #local pL2_2 = array[2] { (p0+p1)/2, (p1+p2)/2 }
       #local pL3 = array[3] { p0, p1, p2 }

       #local ipL2_0 = courbe_in_surface( pL2_0, pS33 )
       #local ipL2_1 = courbe_in_surface( pL2_1, pS33 )
       #local ipL2_2 = courbe_in_surface( pL2_2, pS33 )
       #local ipL3 = courbe_in_surface( pL3, pS33 )

       #local ip = pFgetPoint( 2, pS33, p0 )
       draw( 0, ip, point(0.04) + ma_couleur(<1,0,0,0>) )
       #local ip = pFgetPoint( 2, pS33, p1 )
       draw( 0, ip, point(0.04) + ma_couleur(<1,0,0,0>) )
       #local ip = pFgetPoint( 2, pS33, p2 )
       draw( 0, ip, point(0.04) + ma_couleur(<1,0,0,0>) )
       #local ip = pFgetPoint( 2, pS33, q0 )
       draw( 0, ip, point(0.04) + ma_couleur(<0,1,0,0>) )
       #local ip = pFgetPoint( 2, pS33, q1 )
       draw( 0, ip, point(0.04) + ma_couleur(<0,1,0,0>) )
       #local ip = pFgetPoint( 2, pS33, r0 )
       draw( 0, ip, point(0.05) + ma_couleur(<0,0,1,0>) )

    draw( 2, pS33, finesse(<4,4>) + surface(LISSE)
                            + ma_couleur(<1,1,0,0.5>) )
       draw( 1, ipL2_0, finesse(5) + courbe(0.01)
                                   + ma_couleur(<1,1,1,0>) )
       draw( 1, ipL2_1, finesse(5) + courbe(0.01)
                                   + ma_couleur(<1,1,1,0>) )
       draw( 1, ipL2_2, finesse(5) + courbe(0.01)
                                   + ma_couleur(<1,1,0,0>) )
       draw( 1, ipL3, finesse(5) + courbe(0.02)
                                   + ma_couleur(<0,1,0,0>) )

#end

#macro pF_2236() // pL4 immergée dans une PS33
    #local pL3_0 = array[3] { <-0.5,-0.5,-0.5, 1.0>,
              < 0.0, 0.5,-0.5, 1.0>, < 0.5,-0.5,-0.5,1.0> }
    #local pL3_1 = array[3] { <-0.5, 0.5,-0.0, 1.0>,
              < 0.0, 0.5,-0.0, 1.0>, < 0.5,-0.5,-0.0, 1.0> }
    #local pL3_2 = array[3] { <-0.5, 0.5, 0.5, 1.0>,
              < 0.0,-0.5, 0.5, 1.0>, < 0.5, 0.5, 0.5, 1.0> }
    #local pS33  = array[3] { pL3_0, pL3_1, pL3_2 }
       #local p0 = <1/4,1/8,0,1> ;
       #local p1 = <1/8,3/4,0,1> ;
       #local p2 = <7/8,2/4,0,1> ;
       #local p3 = <4/8,1/4,0,1> ;
```

```
    draw( 2, pS33,
       finesse(<4,4>) + surface(LISSE) + ma_couleur(<1,1,0,0.5>) )

    #local ipL2 = courbe_in_surface( array[2] { p0, p1 }, pS33 )
       draw( 1, ipL2,
       finesse(5) + courbe(0.01) + ma_couleur(<1,1,1,0>) )
    #local ipL2 = courbe_in_surface( array[2] { p1, p2 }, pS33 )
       draw( 1, ipL2,
       finesse(5) + courbe(0.01) + ma_couleur(<1,1,1,0>) )
    #local ipL2 = courbe_in_surface( array[2] { p2, p3 }, pS33 )
       draw( 1, ipL2,
       finesse(5) + courbe(0.01) + ma_couleur(<1,1,1,0>) )

    #local ipL4 = courbe_in_surface(array[4] {p0,p1,p2,p3}, pS33 )
       draw( 1, ipL4,
       finesse(5) + courbe(0.02) + ma_couleur(<0,1,0,0.5>) )
#end

// 23 : INTERFACE

// 231 : TRANSFORMATIONS

#macro pF_2311()    // pFrotate sur une pL4
       #local pL4 = array[4] {
       <-1/2,-1/2,-1/2,1>, < 1/2,-1/2,-1/2,1>,
       < 1/2, 1/2,-1/2,1>, < 1/2, 1/2, 1/2,1> }
       #local N = 24;
       #local i=0; #while (i<N)
              draw( 1, pL4,
              finesse(6) + courbe(0.03) + ma_couleur(<1,1,0,0>) )
              pFrotate( 1, pL4, <0,360/(N-1),0> )
       #local i=i+1; #end
#end

#macro pF_2312()    // pFtranslate sur une pL4
       #local pL4 = array[4] {
       <-1/2,-1/2,-1/2,1>, < 1/2,-1/2,-1/2,1>,
       < 1/2, 1/2,-1/2,1>, < 1/2, 1/2, 1/2,1> }
       pFtranslate( 1, pL4, <0,-1/2,0> )
       #local N = 24;
       #local i=0; #while (i<N)
              draw( 1, pL4,
              finesse(6) + courbe(0.03) + ma_couleur(<1,1,0,0>) )
              pFtranslate( 1, pL4, <0,1/(N-1),0> )
       #local i=i+1; #end
#end
```

```
#macro pF_2313()    // pFscale sur une pL4
       #local pL4 = array[4] {
       <-1/2,-1/2,-1/2,1>, < 1/2,-1/2,-1/2,1>,
       < 1/2, 1/2,-1/2,1>, < 1/2, 1/2, 1/2,1> }
       #local N = 24;
       #local i=0; #while (i<N)
              draw( 1, pL4,
              finesse(6) + courbe(0.03) + ma_couleur(<1,1,0,0>) )
              pFscale( 1, pL4, <0.7,1,0.7> )
       #local i=i+1; #end
#end


// 3 : COMPOSITIONS

// 31 : FORMES RATIONNELLES

#macro pF_3111()    // les 4 coniques comme projections de pL3
       #local oo = <0,0.5,0,1> ;
       #local p0 = <0.5,0,0,1> ;
       #local p1 = <0.5,0,0.5,1> ;
       #local p2 = <0,0,0.5,1> ;
       #local k = 1/sqrt(2);
       #local eps = 0.01 ;

#macro conic( uu )
       union {
#local pL3 = array[3] { p0, k/uu*((1-uu)*oo+uu*p1), p2 }
draw( 1, pL3, finesse(0) + point(0.03) + ma_couleur(<1,0,0>) )
#local pL3 = pFstretch( 1, pL3, <-k+eps,0,0,1>, <1+k-eps,0,0,1> )
draw( 1, pL3, finesse(6) + courbe(0.02) + ma_couleur(<1,1,0>) )
              }
#end

union
{      cone { <0,0.5,0>, 0, <0,-0.5,0>, 1
                     pigment { color rgbt <1,1,1,0.5> } }
       draw( 0, oo, point(0.03) + ma_couleur(<1,1,1>) )
       #local pL2 = array[2] { oo, p1 }
       #local pL2 = pFstretch( 1, pL2, <-k,0,0,1>, <1+k,0,0,1> )
       draw( 1, pL2, finesse(0)+courbe(0.01)+ma_couleur(<1,1,1>) )

              conic( 0.500 )      // hyperbole verticale
              conic( 0.707 )      // cercle
              conic( 1.000 )      // parabole
              conic( 1.500 )      // ellipse
              rotate <0,45,0>
```

```
      }
#end

#macro pF_31121()   // pL3, pL4 et pL5 -> arcs de cercle
      cylinder { <0,0,-1>, <0,0,1>, 0.5
                   pigment { color rgbt <1,1,1,0.5> } }
      #local pL3 = quart_cercle_3( 0.5, 1 )
      pFtranslate( 1, pL3, <0,0,-0.25> )
      #local pL4 = demi_cercle_4( 0.5 )
      #local pL5 = demi_cercle_5( 0.5 )
      pFtranslate( 1, pL5, <0, 0, 0.25> )
      draw( 1, pL3,
      finesse(0) + point(0.03) + ma_couleur(<1,0,0>) )
      draw( 1, pL3,
      finesse(0) + courbe(0.005) + ma_couleur(<1,1,1>) )
      draw( 1, pL3,
      finesse(6) + courbe(0.01) + ma_couleur(<1,1,0>) )
      draw( 1, pL4,
      finesse(0) + point(0.03) + ma_couleur(<1,0,0>) )
      draw( 1, pL4,
      finesse(0) + courbe(0.005) + ma_couleur(<1,1,1>) )
      draw( 1, pL4,
      finesse(6) + courbe(0.01) + ma_couleur(<1,1,0>) )
      draw( 1, pL5,
      finesse(0) + point(0.03) + ma_couleur(<1,0,0>) )
      draw( 1, pL5,
      finesse(0) + courbe(0.005) + ma_couleur(<1,1,1>) )
      draw( 1, pL5,
      finesse(6) + courbe(0.01) + ma_couleur(<1,1,0>) )
#end

#macro pF_31122( choix )
// pL3, pL4 et pL5, projections R4->R3 et arcs de cercle
      cone { <0,0,-0.5>, 0, <0,0,0.5>, 1.0
                   pigment { color rgbt <1,1,1,0.5> } }
      #local k = 1/sqrt(2);
      #local r = 0.5;
      #local oo = < 0,0,-0.5,1 >;
      #switch (choix)
      #case (1)
          #local p0 = < r, 0, 0, 1 > ;
          #local p1 = < r, r, 0, 1 > ;
          #local p2 = < 0, r, 0, 1 > ;
          #local uu = k;
          #local pL = array[3] { p0, k/uu*((1-uu)*oo+uu*p1), p2 }
          #local arc = array[3] { p0, p1*k, p2 }
      #break
      #case (2)
          #local p0 = <  r, 0,  0, 1 > ;
             // k/uu*((1-uu)*oo+uu*p1) = (2*oo+p1)/3
          #local p1 = <  r, 2*r,0, 1 > ;
             // k/uu*oo -k*oo + k*p1 = 2/3*oo + p1/3
          #local p2 = < -r, 2*r,0, 1 > ;            //
          #local p3 = < -r, 0,  0, 1 > ;
          //#local uu = ??; k/uu*((1-uu)*oo+uu*p1) doesn't work
         #local pL = array[4] { p0, (2*oo+p1)/3, (2*oo+p2)/3, p3 }
          #local arc = array[4] { p0, p1/3, p2/3, p3 }
      #break
      #case (3)
          #local p0 = <  r, 0,      0, 1 > ;
          #local p1 = <  r, r,      0, 1 > ;
          #local p2 = <  0, 3/2*r,  0, 1 > ;
          #local p3 = < -r, r,      0, 1 > ;
          #local p4 = < -r, 0,      0, 1 > ;
          #local uu = 1/2;
          #local pL = array[5] { p0,   k/uu*((1-uu)*oo+uu*p1),
                                       k/uu*((1-uu)*oo+uu*p2),
                                       k/uu*((1-uu)*oo+uu*p3),p4 }
          #local arc = array[5] { p0, p1*k, p2*2/3, p3*k, p4 }
      #break
      #end

      union
      {
      #local h = 0.0;
      #switch (choix)
            #case (1)     #local h = 1.0;     #break
            #case (2)     #local h = 0.6;     #break
            #case (3)     #local h = 0.2;     #break
      #end
      draw( 1, pL,
      finesse(0) + point(0.04) + ma_couleur(<0,1,0,0.5>) )
      draw( 1, pL, finesse(0) + courbe(0.01)+ma_couleur(<1,1,1>) )
      #local pL = pFstretch( 1, pL, <-h,0,0,1>, <1+h,0,0,1> )
      draw( 1, pL, finesse(6) + courbe(0.02) + ma_couleur(<0,1,0>)
)
      draw( 1, arc,
      finesse(0) + point(0.04) + ma_couleur(<1,0,0,0.05>) )
      draw( 1, arc, finesse(0) + courbe(0.01)+ma_couleur(<1,1,1>))

      #local arc = pFstretch( 1, arc, <-h,0,0,1>, <1+h,0,0,1> )
      draw( 1, arc, finesse(6) + courbe(0.02)+ma_couleur(<1,0,0>))
      rotate <0,0,0>
      }
```

```
#end

#macro pF_3113()    // 3  approches du cercle complet
      #local k = 1/sqrt(2) ;
      #local eps = 0.01 ;
//    cylinder { <0,0,-1>, <0,0,1>, 0.5
                        pigment { color rgbt <1,1,1,0.5> } }
      #local pL3 = quart_cercle_3( 0.5, 1 )
      pFtranslate( 1, pL3, <0,0,-1> )
      #local pL4 = demi_cercle_4( 0.5 )
      #local pL5 = demi_cercle_5( 0.5 )
      pFtranslate( 1, pL5, <0,0, 1> )

      #local pL3 = pFstretch( 1, pL3, <-5,0,0,1>, <6,0,0,1> )
      draw( 1, pL3, finesse(0) +point(0.03)+ma_couleur(<1,0,0>) )
      draw( 1, pL3, finesse(0) +courbe(0.01)+ma_couleur(<1,1,1>))
      draw( 1, pL3, finesse(6) +point(0.02)+ma_couleur(<1,1,0>) )

      #local pL4 = pFstretch( 1, pL4, <-1,0,0,1>, <2,0,0,1> )
      draw( 1, pL4, finesse(0) +point(0.03)+ma_couleur(<1,0,0>) )
      draw( 1, pL4, finesse(0) +courbe(0.01)+ma_couleur(<1,1,1>))
      draw( 1, pL4, finesse(6) +point(0.02)+ma_couleur(<1,1,0>) )

      #local pL5 = pFstretch( 1, pL5,
                             <-k+eps,0,0,1>, <1+k-eps,0,0,1> )
      draw( 1, pL5, finesse(0) + point(0.03)+ma_couleur(<1,0,0>) )
      draw( 1, pL5, finesse(0) + courbe(0.01)+ma_couleu(<1,1,1>))
      draw( 1, pL5, finesse(6) + point(0.02)+ma_couleur(<1,1,0>) )
#end

#macro pF_3114( uu )       // cercle immergé dans une pS33
    #local pL3_0 = array[3] { <-0.5,-0.5,-0.5, 1.0>,
             < 0.0, 0.5,-0.5, 1.0>, < 0.5,-0.5,-0.5, 1.0> }
    #local pL3_1 = array[3] { <-0.5, 0.5,-0.0, 1.0>,
             < 0.0, 0.5,-0.0, 1.0>, < 0.5,-0.5,-0.0, 1.0> }
    #local pL3_2 = array[3] { <-0.5, 0.5, 0.5, 1.0>,
             < 0.0,-0.5, 0.5, 1.0>, < 0.5, 0.5, 0.5, 1.0> }
    #local pS33  = array[3] { pL3_0, pL3_1, pL3_2 }
    draw( 2, pS33, finesse(<3,3>) + surface(LISSE)
                             + ma_couleur(<1,1,0,0.5>) )
      #local pL5 = demi_cercle_5( 0.25 )
    pFtranslate( 1, pL5, <0.5,0.5,0> )
    #local pL5 = pFstretch( 1, pL5, <0-uu,0,0,1>, <1+uu,0,0,1> )
/*
      #local i=0; #while (i<5)
            #local ip = pFgetPoint( 2, pS33, pL5[i] )
            draw( 0, ip, point(0.03) + ma_couleur(<1,0,0>) )
      #local i=i+1; #end

      #local i=0; #while (i<4)
            #local pL2 = array[2]{ pL5[i], pL5[i+1] }
            #local ipL2 = courbe_in_surface( pL2, pS33 )
            draw( 1, ipL2, finesse(6) + courbe(0.01)
                             + ma_couleur(<1,1,1>) )
      #local i=i+1; #end
*/
      #local ipL5 = courbe_in_surface( pL5, pS33 )
      draw( 1, ipL5, finesse(6) + courbe(0.02)
                             + ma_couleur(<0,1,0>) )
#end

#macro pF_312()     // le cylindre, le tore et la sphère
      #local R = 0.5;
      #local H = 0.5;
      #local R1 = 0.5;
      #local R2 = 0.25;
// un quart de cylindre de rayon R et de hauteur H:
      #local k = sqrt(2)/2;
      #local pCylindre = array[2]
      {     array[3] { < R, 0, 0, 1 >,
                       < R, R, 0, 1 >*k, < 0, R, 0, 1 > },
       array[3] { < R, 0, H, 1 >,
                       < R, R, H, 1 >*k, < 0, R, H, 1 > }
      }
// un seizième de tore de rayons R1 et R2:
      #local R12 = R1+R2;
      #local R2 = abs(R2);
      #local pTore = array[3]
      {     array[3] { < 0,   0, -R12, 1 >,
            < 0,  R2, -R12, 1 >*k, < 0,  R2,  -R1, 1 > },
       array[3] { < R12,  0, -R12, 1 >*k,
            < R12, R2, -R12, 1 >*k*k, <  R1, R2, -R1,  1 >*k },
       array[3] { < R12,  0,  0, 1 >,
            < R12, R2,  0, 1 >*k, <  R1, R2,  0, 1 > }
      }
// une huitième de sphère de rayon R:
      #local pSphere = array[3]
      {     array[3] { < R, 0, 0, 1 >,
                  < R, R, 0, 1 >*k, < 0, R, 0, 1 > },
       array[3] { < R, 0, -R, 1 >*k,
                  < R, R, -R, 1 >*k*k, < 0, R,  0, 1 >*k },
       array[3] { < 0, 0, -R, 1 >, < 0, R, -R, 1 >*k,
                  < 0, R,  0, 1 > }
      }
```

```
      pFtranslate( 2, pCylindre, <0,0,0.05> )
      pFtranslate( 2, pTore, <0,-0.3,0> )
      pFtranslate( 2, pSphere, <0,0,0> )
      draw( 2, pCylindre, finesse(<0,0>) + point(0.03)
                          + ma_couleur(<1,0,0,0>) )
      draw( 2, pCylindre, finesse(<3,3>) + surface(LISSE)
                          + ma_couleur(<1,1,0,0.5>) )
      draw( 2, pTore, finesse(<0,0>) + point(0.03)
                          + ma_couleur(<1,0,0,0>) )
      draw( 2, pTore, finesse(<3,3>) + surface(LISSE)
                          + ma_couleur(<0,1,1,0.5>) )
      draw( 2, pSphere, finesse(<0,0>) + point(0.03)
                          + ma_couleur(<1,0,0,0>) )
      draw( 2, pSphere, finesse(<3,3>) + surface(LISSE)
                          + ma_couleur(<1,0,0,0.5>) )

      #local diag = pFdiagonalisation( 2, pCylindre )
      #local diag = pFstretch( 1, diag, <-0.2,0,0,1>,<1.2,0,0,1> )
      draw( 1, diag, finesse(6) + courbe(0.01)
                          + ma_couleur(<1,1,1>) )

      #local diag = pFdiagonalisation( 2, pTore )
      #local diag = pFstretch( 1, diag, <-0.2,0,0,1>,<1.2,0,0,1> )
      draw( 1, diag, finesse(6) + courbe(0.01)
                          + ma_couleur(<1,1,1>) )

      #local diag = pFdiagonalisation( 2, pSphere )
      #local diag = pFstretch( 1, diag, <-0.2,0,0,1>,<1.2,0,0,1> )
      draw( 1, diag, finesse(6) + courbe(0.01)
                          + ma_couleur(<1,1,1>) )
#end

#macro pF_3131(N)   // diagonales // immergées dans un tore
      torus { 0.5, 0.25 pigment { color rgbt <1,1,1,0.5> } }
      #local c2 = demi_cercle_5( 1 )   // chemin dans xOz
      // demi-diagonale 1
      #local c1 = demi_cercle_5( 0.25 ) // section dans xOy
      pFtranslate( 1, c1, <0.5,0,0> )
      #local pS33 = cross( c1, c2 )
      #local ipL2_1 = pFdiagonalisation( 2, pS33 )
      // demi-diagonale 2
      #local c1 = demi_cercle_5( 0.25 ) // section dans xOy
      pFscale( 1, c1, <-1,-1,1> )
      pFtranslate( 1, c1, <0.5,0,0> )
      pFrotate( 1, c1, <0,180,0> )
      #local pS33 = cross( c1, c2 )
      #local ipL2_2 = pFdiagonalisation( 2, pS33 )
      #local i=0; #while(i<N)
            union
            {     draw( 1, ipL2_1, finesse(5) + courbe(0.01)
                                    + ma_couleur(<1,0,0>) )
                  draw( 1, ipL2_2, finesse(5) + courbe(0.01)
                                    + ma_couleur(<1,0,0>) )
                  rotate <0,360*i/(N-1),0>
            }
      #local i=i+1; #end
#end

#macro pF_3132(N)   // faisceau de droites immergées dans un tore
      torus { 0.5, 0.25 pigment { color rgbt <1,1,1,0.5> } }
      #local c1 = demi_cercle_4( 0.25 )
      pFtranslate( 1, c1, <0.5,0,0> )         // section dans xOy
      #local c2 = demi_cercle_4( 1 )          // chemin dans xOz
      #local pS33 = cross( c1, c2 )
      pFrotate( 2, pS33, <0,-90,0> )
      #local i=0; #while(i<N)
            #local aa = 2*pi*i/N ;
            #local pL2 = array[2]{<0,0,0,1>,<cos(aa),sin(aa),
0,1>}
            pFtranslate( 1, pL2, <0.5,0.5,0> )
            #local ipL2 = courbe_in_surface( pL2, pS33 )
            draw( 1, ipL2, finesse(5) + courbe(0.005)
                                    + ma_couleur(<1,1,1>) )
      #local i=i+1; #end
#end

#macro pF_3133( N ) // cercles concentriques immergés dans un tore
//    torus { 0.5, 0.249 pigment { color rgbt <1,1,1,0.5> } }
      #local c1 = demi_cercle_4( 0.25 )
      pFtranslate( 1, c1, <0.5,0,0> )         // section dans xOy
      #local c2 = demi_cercle_4( 1 )          // chemin dans xOz
      #local pS33 = cross( c1, c2 )
      pFrotate( 2, pS33, <0,-90,0> )
//    draw( 2, pS33, finesse(<3,3>) + point(0.005)
                                    + ma_couleur(<1,1,1,0>) )
      #local i=0; #while (i<N)
            #local uu = i/(N-1);
            #local pL5 = demi_cercle_4( (1-uu)*0.05 + uu*1.0 )
            #local pL5_up = pL5
            #local pL5_down = pL5
      pFrotate( 1, pL5_up, <0,0,180> )
      pFtranslate( 1, pL5_up, <0.5,0.5,0> )
      pFtranslate( 1, pL5_down, <0.5,0.5,0> )
```

```
            #local ipL5_up = courbe_in_surface( pL5_up, pS33 )
            #local ipL5_down = courbe_in_surface( pL5_down,pS33 )
            draw( 1, ipL5_up, finesse(4) + courbe(0.01)
                                  + ma_couleur(<1,uu,0>) )
            draw( 1, ipL5_down, finesse(4) + courbe(0.01)
                                  + ma_couleur(<1,uu,0>) )
      #local i=i+1; #end
#end

#macro pF_3134( N, R )    // cercle et ses rayons (R<2) dans un
tore
      torus { 0.5, 0.25 pigment { color rgbt <1,1,1,0.5> } }

      #local c1 = demi_cercle_4( 0.25 )
      pFtranslate( 1, c1, <0.5,0,0> )        // section dans xOy
      #local c2 = demi_cercle_4( 1 )         // chemin dans xOz
      #local pS33 = cross( c1, c2 )
      pFrotate( 2, pS33, <0,-90,0> )
   // draw( 2, pS33, finesse(<3,3>) + point(0.005)
                                  + ma_couleur(<1,1,1,0>) )

      #local pL5 = demi_cercle_4( R )
      #local pL5_up = pL5
      #local pL5_down = pL5
      pFrotate( 1, pL5_up, <0,0,180> )
      pFtranslate( 1, pL5_up, <0.5,0.5,0> )
      pFtranslate( 1, pL5_down, <0.5,0.5,0> )
      #local ipL5_up = courbe_in_surface( pL5_up, pS33 )
      #local ipL5_down = courbe_in_surface( pL5_down, pS33 )
      draw( 1, ipL5_up, finesse(6) + courbe(0.02)
                                  + ma_couleur(<1,0,0>) )
      draw( 1, ipL5_down, finesse(6) + courbe(0.02)
                                  + ma_couleur(<1,0,0>) )

      #local i=0; #while(i<N)
            #local aa = 2*pi*i/N ;
            #local pL2 = array[2] {<0,0,0,1>,
                              <R*cos(aa),R*sin(aa),0,1> }
            pFtranslate( 1, pL2, <0.5,0.5,0> )
            #local ipL2 = courbe_in_surface( pL2, pS33 )
            draw( 1, ipL2, finesse(5) + courbe(0.01)
                                  + ma_couleur(<1,1,0>) )
      #local i=i+1; #end

#end

#macro pF_3221()    // un vase          ( no box + pers 2/3 )
      #local k = 1/sqrt(2);
      #local c1 = array[7] {     <0.01,-0.50,0,1>,
                                 <0.10,-0.50,0,1>,
                                 <0.50,-0.50,0,1>,
                                 <0.50,-0.25,0,1>,
                                 <0.10, 0.00,0,1>,
                                 <0.10, 0.50,0,1>,
                                 <0.15, 0.50,0,1>
                           }     // section dans xOy
      #local c2 = demi_cercle_5( 1 )          // chemin dans xOz
      #local c2 = pFstretch( 1, c2, <-k,0,0,1>, <1+k,0,0,1> )
      #local pS33 = cross( c1, c2 )
      pFtranslate( 0, pS33[6][2], < 0,-0.5,0> )
                                 // poignée et bec verseur
      pFrotate( 1, c1, <0,180,0> )
      draw( 1, c1, finesse(6) + courbe(0.01)
                                 + ma_couleur(<1,0,0,0>) )
      pFrotate( 2, pS33, <0,-90,0> )
      draw( 2, pS33, finesse(<4,4>) + surface(LISSE)
                                 + ma_texture(MARBRE) )
      object { plane { <0,1,0>, -1/2 } une_texture( GRANIT ) }
 #end

#macro pF_3321()    // surface minimale periodique COONS
      #local p0 = <-1/2,-1/2,-1/2,1> + <0,1/2,-1/2> ;
      #local p1 = <-1/2, 1/2,-1/2,1> + <0,1/2,-1/2> ;
      #local p2 = <-1/2, 1/2, 1/2,1> + <0,1/2,-1/2> ;
      #local p3 = <-1/2,-1/2,-0/2,1> + <0,1/2,-1/2> ;
      #local p4 = <-0/2,-0/2, 1/2,1> + <0,1/2,-1/2> ;
      #local p5 = <-1/2,-0/2, 1/2,1> + <0,1/2,-1/2> ;
      #local p6 = <-0/2,-1/2,-0/2,1> + <0,1/2,-1/2> ;
      #local p7 = <-0/2,-0/2,-0/2,1> + <0,1/2,-1/2> ;
      #local p8 = <-0/2,-0/2, 1/2,1> + <0,1/2,-1/2> ;
      #local L1 = array[6] { p0, p1, p1, p1, p1, p2 }
                                 // approche d'un coin carré
      #local L2 = array[3] { p6, p7/sqrt(2), p8 }
                                 // arc de cercle
      #local L3 = array[3] { p0, p3, p6 }
                                 // arc de parabole
      #local L4 = array[3] { p2, p5, p8 }
                                       // arc de parabole
      #local coons = creer_coons( L3, L4, L1, L2 )
            #local i=0; #while (i<2)
            #local j=0; #while (j<4)
            union
            {     draw( 2, coons, finesse(<3,2>) + surfac(LISSE)
                                 + ma_texture(GRANIT) )
```

```
                    draw( 1, L1, finesse(4) + point(0.02)
                                 + ma_couleur( < 0,0,1/2 > ) )
                    draw( 1, L2, finesse(4) + point(0.02)
                                 + ma_couleur( < 0,0,1/2 > ) )
                    draw( 1, L3, finesse(4) + point(0.02)
                                 + ma_couleur( < 1/2,0,0 > ) )
                    draw( 1, L4, finesse(4) + point(0.02)
                                 + ma_couleur( < 1/2,0,0 > ) )
                    rotate <90*j,0,0>
                    #if (i=1) scale <-1,1,1> #end
             }
             #local j=j+1; #end
             #local i=i+1; #end
#end

#macro pF_3322()    // coquillage COONS
      #local L1 = array[3] { <0.0,0.0,-0.5,1>,
                   <0.0,0.25,-0.5,1>/sqrt(2), <0.0,0.25,-0.1,1> }
      #local L2 = array[3] { <0.5,0.0, 0.0,1>,
                   <0.5,0.25, 0.0,1>/sqrt(2), <0.1,0.25, 0.0,1> }
      #local L3 = array[3] { <0.0,0.25,-0.1,1>,
                   <0.1,0.25,-0.1,1>/sqrt(2), <0.1,0.25,0.0,1> }
      #local L4 = array[3] { <0.0,0.0,-0.5,1>,
                   <0.5,0.0,-0.5,1>/sqrt(2), <0.5,0.0,0.0,1> }

      #local L4 = pLup( L4, 6 ) //4,6
      #local n = taille(L4);
      #local i=0; #while (i<n)
            pFtranslate( 0, L4[i], <0,0.25*sin(4*pi*i/(n-1)),0> )
      #local i=i+1; #end

      #local coons = creer_coons( L1, L2, L3, L4 )
      #local j=0; #while (j<2)
      #local i=0; #while (i<4)
      union
      {     draw( 2, coons, finesse(<3,3>) + surface(LISSE)
                              + ma_texture(MARBRE) )
            #local diag = pFdiagonalisation( 2, coons )
            draw( 1, diag, finesse(4) + courbe(0.005)
                              + ma_couleur( < 1,1,1 > ) )
            //draw( 1, L1, finesse(4) + point(0.005)
                              + ma_couleur( < 0,0,1/2 > ) )
            //draw( 1, L2, finesse(4) + point(0.005)
                              + ma_couleur( < 0,0,1/2 > ) )
            draw( 1, L3, finesse(4) + courbe(0.005)
                              + ma_couleur( < 1/2,0,0 > ) )
            draw( 1, L4, finesse(4) + courbe(0.005)
                              + ma_couleur( < 1/2,0,0 > ) )
            rotate <0,90*i,0>
            #if (j=1)
                  scale <-1,-1,1>
                  translate <0,-0.05,0>
            #end
      }
      #local i=i+1; #end
      #local j=j+1; #end
#end

#macro pF_3323( k ) // rideau COONS (no box+PERS 5/6,
k=1,2,3,4,..)
      #local L1 = creer_ligne( 4*k+1, 1 )
      #local L2 = L1
      #local n = taille(L2);
      #local i=0; #while (i<n)
            pFtranslate( 0, L2[i],
                         <0,0,0.125*sin(2*k*pi*i/(n-1))> )
      #local i=i+1; #end
      pFtranslate( 1, L1, <0,0.5,0> )
      pFtranslate( 1, L2, <0,-0.5,0> )
      #local L3 = array[3] {<-0.5,0.5,0,1>,
                   <-0.5,0,-0.125,1>, <-0.5,-0.5,0,1> }
      #local L4 = array[3] {< 0.5,0.5,0,1>,
                   < 0.5,0, 0.125,1>, <0.5,-0.5,0,1> }
      #local L2 = pLinterpolante( L2 )
      #local coons = creer_coons( L1, L2, L3, L4 )
      #local diag = pFdiagonalisation( 2, coons )
      union {
      draw( 1, L2, finesse(6) + point(0.01)
                              + ma_couleur(<1,1,0,0>) )
      draw( 2, coons, finesse(<3,3>) + surface(LISSE)
                              + ma_texture(MARBRE) )
      draw( 1, diag, finesse(6) + courbe(0.01)
                              + ma_couleur( <0,1,0> ) )
      translate <0,0,-0.25>
      }
      union {
      draw( 1, L2, finesse(6) + point(0.01)
                              + ma_couleur(<1,1,0,0>) )
      draw( 2, coons, finesse(<3,3>) + surface(LISSE)
                              + ma_texture(GRANIT) )
      draw( 1, diag, finesse(6) + courbe(0.01)
                              + ma_couleur( <0,1,0> ) )
      translate <0,0,0.25>}
#end
```

```
#macro pF_3421( k ) // spline interpolante quadrique
                              (no box + AXO 7/8, k=1,2,3,4,...)
      #local curv = creer_ligne( 4*k+1, 1.5 )
      #local n = taille(curv);
      #local i=0; #while (i<n)
            pFtranslate( 0, curv[i],
                        <0,0.125*sin(2*k*pi*i/(n-1)),0> )
      #local i=i+1; #end
      // tracer les points nodaux:
      draw( 1, curv, finesse(0) + point(0.03)
                              + ma_couleur(<1,1,1>) )
      draw( 1, curv, finesse(0) + courbe(0.005)
                              + ma_couleur(<1,1,1>) )
      // tracer une pL interpolante
      #local spi = pLinterpolante( curv )
      draw( 1, spi, finesse(6) + point(0.01)
                              + ma_couleur(<0,1,0,0>) )
      // tracer une spline interpolante ;
      // ATTENTION: spi est un tableau de paraboles !!
      #local b_1 = curv[0] ; // 2*curv[0]-curv[1];
      #local spi = spline_interpolante_quadrique(b_1,curv, false)
      #local i=0; #while (i<taille(spi))
            draw( 1, spi[i], finesse(5) + point(0.01)
                              + ma_couleur(<0,0,1,0>) )
      #local i=i+1; #end
#end

#macro pF_3431()    // le cercle NURBS  (no box + AXO 7/8)
      #local curv = array[5]
      { <0.5,0.0,0.0,1>, <0.0,0.5,0.0,1>,
      <-0.5,0.0,0.0,1>, <0.0,-0.5,0.0,1>, <0.5,0.0,0.0,1> }
      #local b_1 = <0.5,-0.5,0.0,1>;
      // tracer les points nodaux:
      draw( 1, curv, finesse(0) + point(0.04)
                              + ma_couleur(<1,1,1,0.5>) )
      draw( 1, curv, finesse(0) + courbe(0.005)
                              + ma_couleur(<1,1,1>) )
      draw( 0, b_1, point(0.04) + ma_couleur(<1,1,0,0>) )
      // tracer la spline quadrique interpolante
      // (ATTENTION: spi est un tableau de paraboles)
      #local spi = spline_interpolante_quadrique(b_1,curv,true )
      #local i=0; #while (i<taille(spi))
            draw( 1, spi[i], finesse(5) + courbe(0.01)
                              + ma_couleur(<1,0,0,0>) )
      #local i=i+1; #end
#end
```

```
... et quelques autres à venir !
```

Note finale :

1) les algorithmes ont été programmés et les images ont été produites sur POVRAY, formidable environnement de développement et de rendu RayTracing open source libre (licence GPL) ; les sources pFlibs.inc et pFbook.inc sont libres (disons sous licence GPL) ;

2) Les images ont fait un petit détour par The Gimp, superbe logiciel de retouche d'images open source libre (licence GPL) et enregistrées au format standard ouvert jpg ; le poids total des images est de 15 Mo environ ;

3) ce document a été finalisé sur OpenOffice.org 1.1., magnifique suite bureautique open source libre (licence LGPL), sous format ouvert (XML compressé ZIP). Le fichier texte du document complet composé dans le module WRITER pèse 108 ko (les images sont liées et non intégrées au fichier) ; ce fichier a été exporté directement depuis OpenOffice au format pdf dans une qualité « optimisée pour presse à imprimer », son poids est de 6,48Mo . La couverture et la quatrième de couverture ont été réalisées dans le module DRAW, le fichier pèse 19,4 ko (les images sont liées et non intégrées au fichier) et a été également exporté au format pdf pour produire un fichier pesant 237 ko.

4) l'auteur, pas mal non plus, en transit sur Finestrouze entre Macintosh et Linux, est lui-même open source libre, et attend avec intérêt toutes les remarques qui pourront faire avancer cet essai sur les formes gauches.