Rapport de projet d'informatique : Tracé de surface minimale avec conditions aux limites determinées

Jean-Michel FRIEDT

26 mai 1995

1 Introduction

Le problème que je me suis proposé de résoudre est le suivant : étant donné des conditions aux limites fixées par l'utilisateur, quelle sera la surface minimale portée par ce cadre. Ce problème se pose dans de nombreux domaines : de l'astrophysique où la courbure de l'espace pourraît être déterminée par les cordes massives qui s'y trouvent à la forme des membranes en biologie. Il fut abordé il y a longtemps et formalisé avec le problème de Plateau. Mais il fallut attendre le développement de l'informatique et des puissants outils graphiques qui l'accompagnent pour voir émerger un univers peu connu basé sur un principe très simple : les surfaces minimales.

Ce problème se rapporte physiquement à tremper un cadre de fil de fer dans un liquide savonneux et à observer le film ainsi formé. Ma première approche de cette idée s'est faite au travers d'une conférence de David Hoffman, mathématicien à la University of Massachusetts, qui a découvert plusieurs surfaces minimales grâce à l'outil informatique [1].

Je vais présenter ici une approche graphique du problème dans laquelle l'utilisateur donne les conditions limites et où le programme calcule chaque point de la surface puis affiche graphiquement le résultat, laissant la liberté à l'utilisateur d'observer l'image obtenue sous tous ses angles.

Je montrerai successivement les différentes étapes de la conception de ce logiciel : problèmes physiques, mathématiques et infomatiques. Puis j'entrerai dans une description du programme lui même, des procédures et des structures de données qui le composent. Enfin j'avancerai les résultats obtenus.

Il peut être utile, pour une meilleure compréhension de l'étude, de savoir que je considère la surface comme un assemblage de facettes, et que les éléments sur lesquels j'agis sont les nœuds de ce maillage. Cette approche correspond a la façon dont sont présentées les intégrales de Riemann : le continu est approximé comme une succession serrée de segments. Une telle structure a de plus l'avantage d'être facilement implantable informatiquement au travers de tableaux à deux dimensions.

2 Différentes approches du probleme

2.1 Approche par la courbure

Une propriété fondamentale des surfaces minimales est leur courbure moyenne nulle en tout point. Si R_{min} est la courbure minimale et R_{Max} est la courbure maximale en un point, alors

$$\frac{1}{R_{min}} + \frac{1}{R_{Max}} = 0$$

Pour un film de liquide limité par deux membranes de tension supeficielle σ, nous avons la relation de Laplace :

$$\Delta P = 2\sigma(\frac{1}{R_{min}} + \frac{1}{R_{Max}})$$

Cette approche a l'avantage de permettre de tracer des surfaces pour lesquelles la pression intérieure n'est pas égale à la pression extérieure. Cependant, elle a l'inconvénient de nécessiter la définition d'une courbure sur une surface dont les points intéressants, les nœuds de la maillle, sont tous des points singuliers. J'ai tenté une première étude basée sur les plus petits angles entre les plans opposés mais trop de cas particuliers se présentaient pour pouvoir construire un algorithme efficace.

D'autre part, il faut toujours respecter une convention d'orientation de la surface qui n'est pas facile à définir dans le cas général pour une surface ouverte. Après plusieurs approches infructueuses quant à la définition d'une telle courbure, je me suis tourné vers l'approche présentée dans la section suivante.

2.2 Approche par la surface

Une surface minimale est avant tout caractérisée par une aire minimale. Cette approche, bien que restrictive au niveau des différences de pression entre les deux espaces délimités par le film, est beaucoup plus souple d'emploi et aisée à programmer. La lourdeur des calculs, qui n'était pas évitée avec la première approche, se traduit par une certaine lenteur du programme à partir d'une précision trop grande.

Des temps de calcul raisonnnables et un bon réalisme ont cependant pû être atteints avec cette description qui est celle que j'ai adopté.

3 Résolution du probleme

3.1 Problème physique

L'approche du problème par une minimalisation de l'aire de la surface n'a pas seulement été déterminée par un souci de simplicité de programmation mais avant tout par le désir de m'approcher du phénomène physique.

En-effet, un film va prendre la structure qui minimise son énergie. Or nous savons que la force de tension est proportionnelle à l'élément infinitésimal dS puisque la force de tension est liée à l'élément infinitésimal de longueur. En intégrant, nous obtenons une surface proportionnelle à l'élément infinitésimal de surface selon :

$$F = \sigma 2\pi R \Rightarrow dW = -FdR = -\sigma 2\pi R dR = -\sigma dS$$

soit une énergie de tension proportionnelle à l'aire du film. Dans le cas présent, je néglige toutes les forces extérieures, approximation valable expérimentalement pour des films de taille usuelle. Au delà de quelques centimètres, les effets du poids deviennent prédominants et les surfaces proposées par ce programme ne sont plus valables.

3.2 Problème mathématique

Le problème mathématique est le suivant : étant donnés quatre points M_i au milieu desquels se trouve le point P, comment choisir les coordonnées de P de façon à ce que l'aire de la surface ainsi définie soit minimale. Ceci revient à rendre extrémale localement la surface, et nous obtiendrons une surface minimale en réitérant le procédé à tous les points du film. La surface des quatre triangles M_iPM_j est extrémale si

$$S = \sum_{i,j=i+1}^{1..4} \frac{1}{2} ||\overrightarrow{M_iP} \wedge \overrightarrow{M_jP}||^2$$

est minimale. Je prends le carré des normes pour avoir une relation linéaire entre les coordonnées après dérivation.

Après avoir obtenu le développement de cette expression grâce à Maple V (code en fin de section), j'exprime le fait que la surface est extrémale par

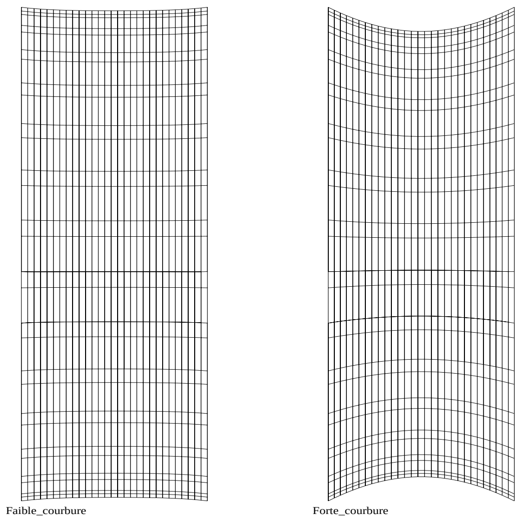
$$dS = 0 = \frac{\partial S}{\partial x_p} dx_p + \frac{\partial S}{\partial y_p} dy_p + \frac{\partial S}{\partial z_p} dz_p$$

et puisque dx_p, dy_p, dz_p sont indépendants nous obtenons le système d'équations linéaires :

$$\frac{\partial S}{\partial x_p} = 0, \ \frac{\partial S}{\partial y_p} = 0, \ \frac{\partial S}{\partial z_p} = 0$$

Ce système se résoud aisément et permet de rapidement trouver la position de P qui minimise localement la surface.

Un autre problème est lié à **l'unicité** de la solution : la résolution analytique de l'équation de la surface minimale reposant sur deux cerceaux donne deux solutions, une cathénoïde de faible courbure et une autre de courbure plus élevée. Le programme doit pouvoir esquisser ces deux possibilités. C'est ici le cas : les différentes solutions dépendent des conditions initiales choisies. La solution de faible courbure est obtenue en partant d'un cylindre, tandis que la solution de forte courbure est l'aboutissement du calcul pour une surface initialement plus courbée que la solution (le rayon suit une parabole). Ainsi, pour un même contour, il faudra effectuer plusieurs essais avec différentes conditions initiales pour être certain d'observer toutes les solutions.



Les différentes solutions pour un même contour.

Programme Maple V :

$$
\begin{array}{rcl}
with & & (linalg) : v_{01} := vector([x_1 - x_0, y_1 - y_0, z_1 - z_0]); \\
v_{02} & := & vector([x_2 - x_0, y_2 - y_0, z_2 - z_0]); \\
v_{03} & := & vector([x_3 - x_0, y_3 - y_0, z_3 - z_0]); \\
v_{04} & := & vector([x_4 - x_0, y_4 - y_0, z_4 - z_0]); \\
v_{12} & := & crossprod(v_{01}, v_{02}); \\
v_{23} & := & crossprod(v_{02}, v_{03}); \\
v_{34} & := & crossprod(v_{03}, v_{04}); \\
v_{41} & := & crossprod(v_{04}, v_{01}); \\
n_{12} & := & norm(v_{12}, 2)^2; \\
n_{23} & := & norm(v_{23}, 2)^2; \\
n_{34} & := & norm(v_{34}, 2)^2; \\
n_{41} & := & norm(v_{41}, 2)^2; \\
n & := & n_{12} + n_{23} + n_{34} + n_{41}; \\
dn_1 & := & diff(n, x_0); \\
dn_2 & := & diff(n, y_0); \\
dn_3 & := & diff(n, z_0); \\
n_1 & := & (z_1 * x_2 - z_1 * x_0 - z_0 * x_2 - x_1 * z_2 + x_1 * z_0 + x_0 * z_2) * (z_2 - z_1) + \\
& & (x_1 * y_2 - x_1 * y_0 - x_0 * y_2 - y_1 * x_2 + y_1 * x_0 + y_0 * x_2) * (y_1 - y_2) + \\
& & (z_2 * x_3 - x_0 * z_2 - z_0 * x_3 - x_2 * z_3 + z_0 * x_2 + x_0 * z_3) * (z_3 - z_2) + \\
& & (x_2 * y_3 - y_0 * x_2 - x_0 * y_3 - y_2 * x_3 + x_0 * y_2 + y_0 * x_3) * (y_2 - y_3) + \\
& & (z_3 * x_4 - x_0 * z_3 - z_0 * x_4 - x_3 * z_4 + z_0 * x_3 + x_0 * z_4) * (z_4 - z_3) + \\
& & (x_3 * y_4 - y_0 * x_3 - x_0 * y_4 - y_3 * x_4 + x_0 * y_3 + y_0 * x_4) * (y_3 - y_4) + \\
& & (z_4 * x_1 - x_0 * z_4 - x_1 * z_0 - x_4 * z_1 + z_0 * x_4 + z_1 * x_0) * (z_1 - z_4) + \\
& & (x_4 * y_1 - y_0 * x_4 - y_1 * x_0 - y_4 * x_1 + x_0 * y_4 + x_1 * y_0) * (y_4 - y_1); \\
n_2 & := & (y_1 * z_2 - y_1 * z_0 - y_0 * z_2 - z_1 * y_2 + z_1 * y_0 + z_0 * y_2) * (z_1 - z_2) + \\
& & (x_1 * y_2 - x_1 * y_0 - x_0 * y2 - y_1 * x_2 + y_1 * x_0 + y_0 * x_2) * (x_2 - x_1) + \\
& & (y2 * z_3 - z_0 * y_2 -_y 0 * z_3 - z_2 * y_3 + y_0 * z_2 + z_0 * y_3) * (z_2 - z_3) + \\
& & (x_2 * y_3 - y_0 * x_2 - x_0 * y_3 - y_2 * x_3 + x_0 * y_2 + y_0 * x_3) * (x_3 - x_2) + \\
& & (y_3 * z_4 - z_0 * y_3 - y_0 * z_4 - z_3 * y_4 + y_0 * z_3 + z_0 * y_4) * (z_3 - z_4) + \\
& & (x_3 * y_4 - y_0 * x_3 - x_0 * y_4 - y_3 * x_4 + x_0 * y_3 + y_0 * x_4) * (x_4 - x_3) + \\
& & (y_4 * z_1 - z_0 * y_4 - z_1 * y_0 - z_4 * y_1 + y_0 * z_4 + y_1 * z_0) * (z_4 - z_1) + \\
& & (x_4 * y_1 - y_0 * x_4 - y_1 * x_0 - y_4 * x_1 + x_0 * y4 + x_1 * y_0) * (x_1 - x_4); \\
n_3 & := & (y_1 * z_2 - y_1 * z_0 - y0 * z_2 - z_1 * y_2 + z_1 * y_0 + z_0 * y_2) * (y_2 - y_1) + \\
& & (z_1 * x_2 - z_1 * x_0 - z_0 * x_2 - x_1 * z_2 + x_1 * z_0 + x_0 * z_2) * (x_1 - x_2) + \\
& & (y_2 * z_3 - z_0 * y_2 - y_0 * z_3 - z_2 * y_3 + y_0 * z_2 + z_0 * y_3) * (y_3 - y_2) + \\
& & (z_2 * x_3 - x_0 * z_2 - z_0 * x_3 - x_2 * z_3 + z_0 * x_2 + x_0 * z_3) * (x_2 - x_3) + \\
& & (y_3 * z_4 - z_0 * y_3 - y_0 * z_4 - z_3 * y_4 + y_0 * z_3 + z_0 * y_4) * (y_4 - y_3) + \\
& & (z_3 * x_4 - x_0 * z_3 - z_0 * x_4 - x_3 * z_4 + z_0 * x_3 + x_0 * z_4) * (x_3 - x_4) + \\
& & (y_4 * z_1 - z_0 * y_4 - z_1 * y_0 - z_4 * y_1 + y_0 * z_4 + y_1 * z_0) * (y_1 - y_4) + \\
& & (z_4 * x_1 - x_0 * z_4 - x_1 * z_0 - x_4 * z_1 + z_0 * x_4 + z_1 * x_0) * (x_4 - x_1); \\
n_1 & := & expand(n_1); \\
n_2 & := & expand(n_2); \\
n_3 & := & expand(n_3);
\end{array}
$$

J'achève la résolution du système en x_0, y_0, z_0 à la main en écrivant le déterminant. Les trois lignes très longues ont dû être copiées et modifiées directement car j'ai été incapable de simplifier proprement les résultats avec Maple V.

3.3 Problème informatique

Le problème informatique est le suivant : modéliser une surface de forme variable qui puisse être étirée de façon à laisser à la figure la liberté de prendre sa configuration optimale, ie de surface minimale. Il a ensuite fallu obtenir la fonction permettant de déduire de proche en proche la forme recherchée. J'ai choisi de travailler en coordonnées cartésiennes car les expressions de la surface du triangle et de la norme y sont simples, l'équivalence avec un tableau à deux dimensions est directe et enfin les symetries sont suffisament rares pour ne pas chercher à les exploiter dans le cadre d'un programme qui se veut le plus général possible.

D'autre part, le souci de simplifier la tâche à l'utilisateur m'est apparu lors des tests successifs qui s'avéraient souvent laborieux. Ainsi, plusieurs options telles que l'éditeur et la sauvegarde m'ont semblées indispensables pour permettre l'utilisation par quelqu'un ne connaissant pas le programme. Malgré celà, le niveau de sécurité est assez bas et une fausse manœuvre peut souvent se solder par une perte de données. Les nombreux tests tentent de limiter ce risque, et c'est pourquoi l'éditeur en est resté à l'état d'embryon, les risques de mauvaise manipulation étant là beaucoup trop grands.

4 Le programme

4.1 Les structures de données

4.1.1 Les points

Les points sont représentés par un tableau de trois flottants contenant chacun une coordonnée du point. Ce type est très souvent utilisé dans le programme et s'appelle le type *point*.

4.1.2 La surface s

La surface étudiée est stockée dans un tableau à deux dimensions (nx$\times$ ny où nx ety ny sont des constantes fixées par l'utilisateur). Chaque point possède les informations suivantes :

* ses coordonnées cartésiennes stockées dans le type point.

* son type sous forme d'un charactère qui peut être **f** si c'est un point fixe (ie un point qui appartient aux limites déterminées par l'utilisateur et qui ne doit donc pas être bougé); **b** si c'est un point libre qui doit être déplacé de façon à rendre la surface minimale; ou **l** si c'est un point qui est lié à un autre point du plan qui n'est pas un de ses voisins directs (ie dont l'indice des voisins ne diffère pas de 1 de ses propres indices). Cette dernière option est notament utilisée dans la cathénoïde pour refermer le plan en un cylindre. J'ai prévu cette option plus généralement pour recoller deux morceaux du plan entre eux.

* dans le cas où le type du point est **l**, j'initialise les deux tableaux de 4 par 2 integers qui contiennent les voisins directs et diagonaux du point. Ces deux tableaux sont initialisés en pointeurs dans un souci d'économie de mémoire. En-effet, les voisins ne doivent être précisés explicitement que si le point est de type **l**. Or ces points sont rares. Il serait donc inutile de toujours allouer cette mémoire. Un simple calcul montre l'intérêt d'une telle structure : ces tableaux contiennent $2 \times 4 \times 2$ integers soit 16×16 bits. Pour la structure complète de s, on a donc $n^2(16 \times 16 + 8 + 32 \times 3)$ bits (on ajoute 3 floats et 1 char). Nous remplissons donc 64ko de mémoire pour $64 * 1024 = n^2 \frac{(3\times 32+8+16\times 16)}{8} \Rightarrow n = 24$ tandis qu'avec l'économie de mémoire, $n = 24$ occupe $\frac{3\times 32+8}{8} \times n^2 = 7,3$kb (en négligeant les points de type **l**, approximation qui est valable dans la plupart des cas).

4.1.3 Le tableau intermediaire sinter

sinter ne contient que les calculs intermédiaires et par conséquent a une structure beaucoup plus simple que s : sinter est simplement un tableau à deux dimensions contenant les coordonnées de chaque nœud. Ce tableau a été introduit suite à un problème lors du calcul de la cathénoïde : le déplacement des points provoquait un mouvement de vissage de l'ensemble de la surface. En fait, je faisais jusque là les calculs et le stockage des données sur un seul et même tableau. Ce problème ne s'était pas manifesté lors du calcul sur un plan car là les déformations étaient évacuées sur les limites du plan. La cathénoïde partant d'un cylindre, les erreurs de calcul s'accumulaient en sortant d'un côté du plan et en réentrant par le côté opposé auquel il était lié. Nous verrons plus loin que l'exemple de la cathénoïde était très instructif puisqu'il a aussi obligé à affiner les techniques de calcul.

4.2 Les variables globales

4.2.1 La surface s et le tableau intermediaire sinter

Ce sont les deux éléments au cœur du programme auxquels il est fait tout le temps appel. Les passer en paramètre serait lent, fastidieux et inutile puisque le programme n'a pas de sens sans ces structures. Il me semble donc difficile de passer ces deux tableaux à chaque procédure, d'où mon choix de les laisser en variables globales.

4.2.2 Le tableau angles

Le tableau des deux angles de rotation a volontairement été laissé en variable globale. J'ai tenté de passer les angles en paramètres entre toutes les procédures qui ont besoin de ce tableau mais ai dû abandonner le choix de la propreté au profit de la rapidité : le passage prenait tant de temps que toute animation était impossible (il faut rappeler que le passage se fait n^2 fois pour chaque image).

4.2.3 Les initialisations graphiques

Toutes les variables liées à l'initialisation graphique ont été placées globales pour les deux raisons suivantes :

* comme avant, passer tous les paramètres graphiques prendrait beaucoup de temps et ralentirait considérablement l'animation.

* les noms des variables graphiques sont plus ou moins standards si les conseils du manuel du 'Xlib programming manual' [2] ont été respectés.

4.3 Les procédures

4.3.1 La définition de la surface

Elle se fait par l'utilisateur au travers de la procédure def_surface.h ou par le chargement d'un fichier sauvé lors d'un cession précédente. Cette procédure est sauvée dans un fichier séparé pour être modifiée sans avoir à toucher au programme principal. Cette précaution a pour but déviter à l'utilisateur la recherche fastidieuse de la bonne procédure et le risque d'une fausse manœuvre sur les autres fonctions. La surface est définie par la position de chaque point, le cadre devant être choisi avec soin car il sera le support de tout le programme et ne sera jamais modifié. Les liaisons entre les parties du plan non voisines sont aussi déterminéees à ce stade. Il faut donc donner les informations suivantes pour chaque point :

le type de point : **l**ibre, cadre (**f**) ou à **b**ouger.

la position du point : plus ou moins au hasard si le point est de type **b** ou **l**, avec précision si c'est un point **f**.

si le type est **l**, il faut avoir initialisé les pointeurs des deux tableaux voisins_droi et voisins_diag, et y mettre les valeurs adéquates.

J'ai préféré définir la surface par une procédure plutôt que par un fichier de coordonnées car les boucles permettent d'exploiter, au moins à la définition, les symetries du cadre. De plus, une surface précise deviendrait fastidieuse à définir point par point. L'idéal aurait bien sûr été d'inclure un éditeur de surface, mais le manque de temps m'a arrêté dans ce projet.

4.3.2 Les opérations sur la surface

Si le nom de fichier proposé par l'utilisateur n'existe pas encore, il est créé et le résultat du calcul y sera stocké. Sinon, la figure contenue dans le fichier est chargée : la surface n'est plus définie par la procédure *init_surface* donnée dans le fichier séparé mais par les points qui avaient été sauvés. Ceci laisse notament l'option à l'utilisateur de créer sa surface de toutes pièces en ne donnant que des coordonnées. Une fois les données stockées dans le tableau s, le programme offre deux options : simplement visualiser la surface ou reprendre un calcul qui avait été interrompu. Cette option est intéressante dans le sens où elle limite les risques de perte des données lors d'un calcul long et fastidieux : l'utilisateur peut interrompre la cession de travail par l'appui d'une touche. Sa surface intermédiaire est alors sauvée puis rechargée pour continuer le calcul.

D'autre part il est demandé à l'utilisateur s'il désire afficher graphiquement les étapes intermédiaires du calcul. Ceci a pour but de permettre le calcul sur une machine qui n'est pas celle de l'utilisateur (calcul sur une machine distante plus rapide) et d'accélérer le programme en n'appelant pas la procédure d'affichage.

Ces opérations sont effectuées dans les procédures *commence_work* si le fichier doit être créé, ou *commence_load* si le fichier existe déjà.

4.3.3 Le calcul de la surface

Le principe du programme est de minimiser localement la surface. En ce sens, la surface est parcourue et chaque point est placé provisoirement au point qui rend minimale la surface environnante. Cette procédure doit être réitérée jusqu'à ce qu'aucun point ne soit bougé : une position stable a alors été atteinte. En-effet, la position de chaque point devrait être recalculée lorsqu'un voisin est déplacé. Nous obtenons ainsi de proche en proche la caractéristique désirée sur la globalité de la surface.

Cependant, une telle approche directe donne des résultats décevants, notament avec la cathénoïde. Un meilleur résultat et une plus grande rapidité de calcul sont obtenus en effectuant un premier parcours de la surface et en plaçant grossièrement chaque point. Ceci se fait en considérant chaque élément non par rapport à son voisin immédiat mais par rapport à ses voisins *diagonaux*.

4.3.4 L'éditeur

Cette procédure devait à l'origine assister l'utilisateur dans la création de la surface. En fait, cette option se réduit à donner des informations sur la surface formée : coordonnées et type des points, localisation d'un point donné...

L'ensemble des options qui devaient permettre de modifier la forme de la surface ont dû être retirées faute de temps pour obtenir un côde sûr sans risque de pertes des données.

Il est possible à partir de là d'exporter son image au format PostScript par l'appui de la touche r : le programme demande le nom du fichier et vérifie que celui n'est pas déjà présent. Le format PostScript permet de facilement inclure les résultats des calculs dans un document. Le fichier PostScript est créé de la façon suivante :

* l'entête standard de Encapsulated PostScript : il contient notament la taille de la fenêtre contenant l'image (du coin en bas à gauche au coin en haut à droite) et d'autres informations moins utiles telles que le titre et l'auteur.
* la série d'instructions x_0 y_0

`moveto`

x_1 y_1

`lineto`

qui trace une droite de x_0 y_0 à x_1 y_1.

* la conclusion qui imprime le titre et affiche la page.

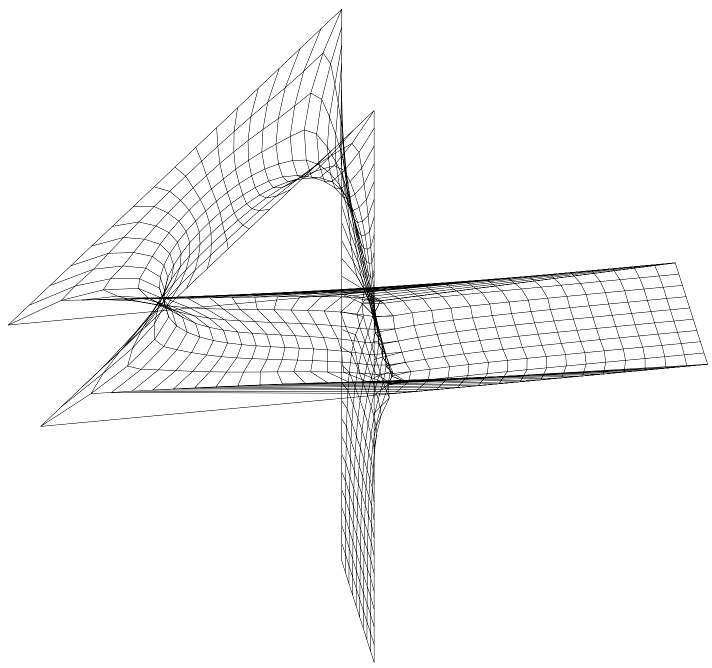
4.3.5 L'affichage

La surface est définie de façon vectorielle en trois dimensions. La projection est de type cavalière après avoir appliqué une matrice de rotation aux coordonnées de la surface. Les angles de la matrice de rotation varient en déplaçant la souris tout en laissant le bouton de gauche appuyé. Le bouton de droite enfoncé permet de modifier la taille de la surface. La sortie propre du programme de fait par le bouton du milieu. Si la surface a été chargée d'un fichier de sauvegarde créé antérieurement, l'appui du bouton 2 amène le choix de l'accès à l'éditeur puis à une confirmation par un nouvel appui sur le même bouton. Cette option a dû être retirée de la cession durant laquelle la surface est calculée. En effet, durant cette première étape, il est possible d'interrompre le calcul en appuyant sur une touche du clavier. Pour que ceci soit possible, il a fallu inclure une procédure qui retire l'attente du clavier lors de l'interrogation d'une touche. Ayant repris cette procédure des News, je suis incapable de remettre le clavier dans son état initial à partir du programme, et donc d'accepter toute entrée par cette périphérique dans les procédures suivantes. Le seul inconvénient de cette méthode est que je ne peux plus proposer à l'utilisateur d'entrer des commentaires à la fin de sa sauvegarde.

4.3.6 La sauvegarde

Etant donné le temps important de calcul nécessaire à déterminer avec précision les caractéristiques d'une surface, chaque résultat est sauvé après le calcul principal.

L'utilisateur doit entrer au lancement du programme un nom de fichier : s'il existe déjà, il est chargé à condition que la taille du tableau soit la bonne. Sinon, le fichier est créé et le résultat y sera stocké une fois le calcul achevé.



Essai sur une surface qui se recoupe

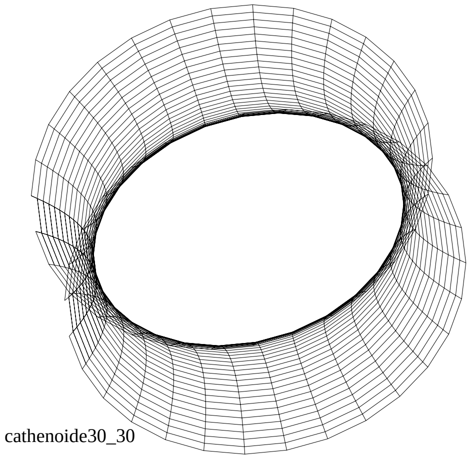
5 Résultats et limitations

Je suis satisfait des résultats obtenus compte tenu de la simplicité de l'algorithme. Bien que des améliorations notables puissent être apportées à l'affichage, la rapidité pour l'animation et le désir de conserver la portabilité du code m'ont poussé à m'en tenir à une image vectoriélisée fil de fer.

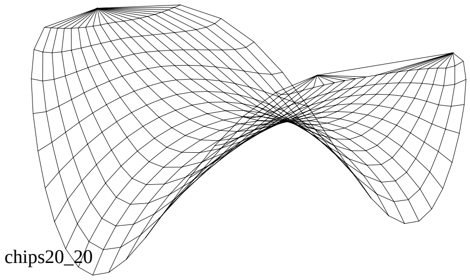
La limitation la plus notable est l'interdiction d'obtenir un film qui se recoupe. Je ne suis cependant pas certain que cette limitaion ne puisse être dépassée du fait de la grande souplesse des calculs et du type **l** des points (qui, rappelons le, permet de joindre deux points de la surface). L'absence de tels phénomènes physique m'ont conduit à ne pas pousser plus loin dans cette voie, bien que le cas de l'helicoïde soit remarquable : je ne m'attendais pas au succès obtenu dans ce calcul.

La seconde limitation tient au fait que la surface est définie par un plan et ne permet pas ainsi de créer toutes les formes de cadre possible. Il m'a par exemple été impossible de reproduire l'helicoïde percée découverte par David Hoffman puisque l'intérieur de ma surface ne peut être troué.

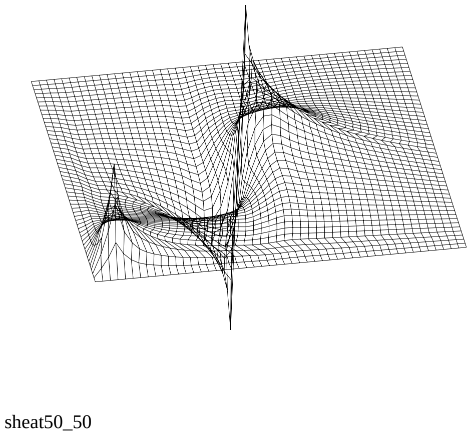
6 Conclusion



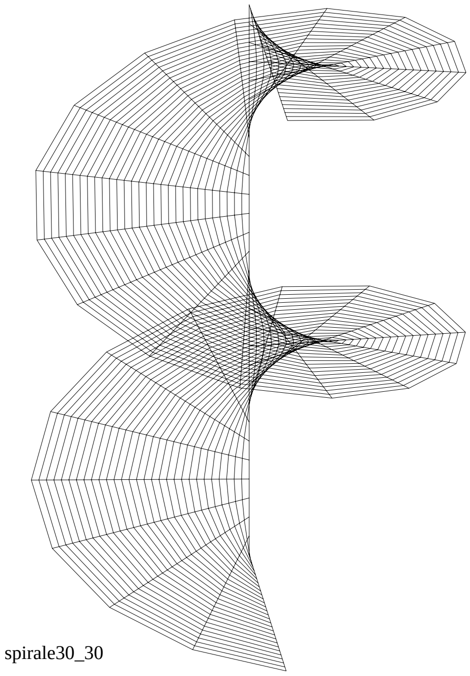
cathénoïde



cerceau tordu

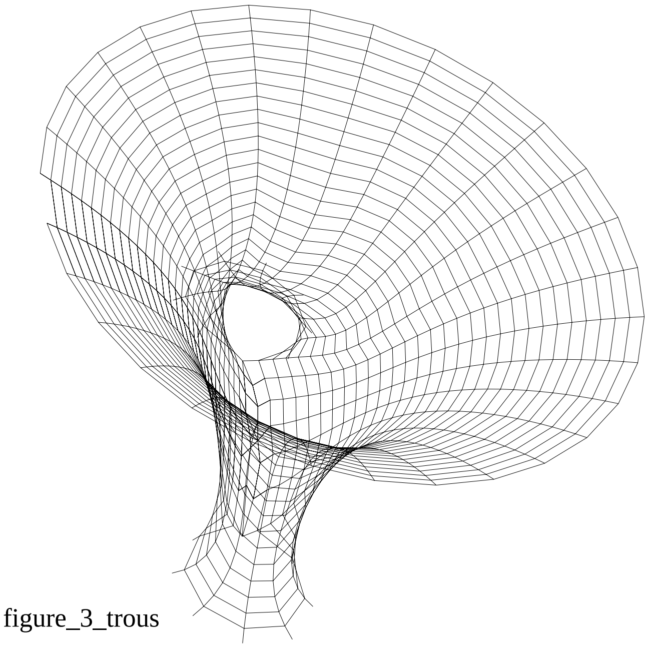


feuille de caoutchouc étirée

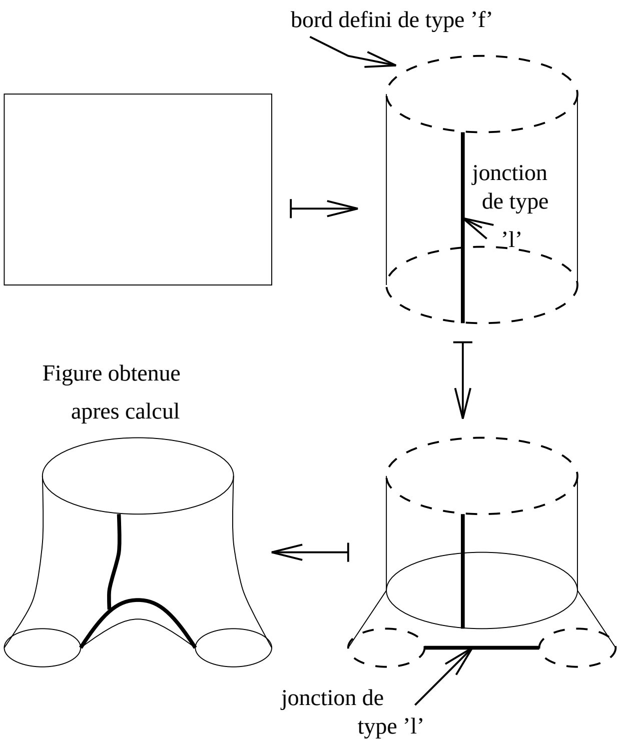


hélicoïde

Ceci sont des images obtenues au moyen de ce logiciel. Elles m'ont permis de vérifier la validité du calcul ainsi que de la théorie. Ce projet a été l'occasion de réaliser un programme auquel je réfléchissais depuis longtemps, alors que j'admirais les sorties d'écran des logiciels professionnels de calcul. Bien que n'atteignant pas la généralité de ces produits, je suis satisfait du réalisme et de la qualité des images obtenues. Ceci est un nouvel exemple de l'importance de l'outil informatique dans la physique, lorsque la simulation permet d'obtenir des résultats inaccessibles à l'analyse.



essai de surface a 3 trous



méthode de création de la surface à 3 trous.

7 Source C

7.1 La gestion graphique

```
/* ce fichier ouvre un environnement graphique (1024x768) X11 : il est appele par graphic */
/* pour le graphique, display et window pour la fenetre (noms utilises par Xlib program.  */
/* manual) : par exemple XDrawPoint(display,window,graphic,100,100);                      */
/* De plus la librairie mathematique est deja chargee. Apres avoir defini une nlle page   */
/* graphique, il faut envoyer  XShowSesame();  qui va afficher le graph a l'ecran         */
#include <X11/Xlib.h> /* compiler avec   cc *.c -lX11 -I/usr/openwin/share/include -L. -lm*/
#include <X11/Xutil.h>
#include <math.h>
   Display *display;
   Window window;
   int sc_num;
   Screen *sc_ptr;
   char *disp=NULL;
   unsigned int b_w=4;
   int scrnum;
   GC graphic;
   XGCValues values;
   static char d_list[]={12,24};
void XgetGC()                           /* initialisation de la fenetre en mode graphique */
{graphic=XCreateGC(display,window,0,&values);                      /* 0 : default values */
 XSetForeground(display,graphic,BlackPixel(display,sc_num));
 XSetLineAttributes(display,graphic,1,LineSolid,CapRound,JoinRound);       /* 1:linewidth */
 XSetDashes(display,graphic,0,d_list,2);
}
void XOpenSesame()                                                  /*ouverture d'une fenetre*/
{if ((display=XOpenDisplay(disp))==NULL) {printf("cannot connect to server%s\n",
      XDisplayName(disp));}
 sc_num=DefaultScreen(display);
 sc_ptr=DefaultScreenOfDisplay(display);
 window=XCreateSimpleWindow(display,RootWindow(display,sc_num),1,1,1024,768,b_w,
        BlackPixel(display,sc_num),WhitePixel(display,sc_num));
 XSelectInput(display,window,MotionNotify | ExposureMask | ButtonPressMask  |
  Button1MotionMask | Button3MotionMask);/*choix des Events : affichage, souris et boutons*/
}
void XShowSesame()            /* affichage de la page graphique a l'ecran en premier plan */
{ XMapRaised(display,window);
  XFlush();
}
void Xinitgraph()        /* initialisations graphiques : appel des procedures precedentes */
{XOpenSesame();
 XgetGC();
 XShowSesame();}
```

7.2 La définition des variables globales

```
/* definitype.h : cette procedure comprend la definition de toutes les variables globales */
typedef float point[3];   /* definition de la surface comme reseau de pts contenant leurs */
                                                        /* coordonnees cartesiennes       */
struct surf {   char genre;       /* definition de la structure principale de la surface  */
                point coord;/* coord en 3D et leur type : 'b' pour bouger, 'f' pour fixe  */
```

```
               int **voisins_diag;                                  /* 'l' pour lie  */
               int **voisins_droi;} s[nx][ny];   /* voisins diagonaux et directs si 'l'  */
point sinter[nx][ny];                            /* definition du tableau intermediaire   */
long htimer[4];                          /*tableau utilise lors de l'inhibition du clavier */
float angle[2]={1.4,1.0};          /* tableau contenant les angles de rotation : 2 angles */
```

7.3 Le programme principal

```
/*programme de trace de surface minimal connaissant les conditions aux bords : compilation*/
#include "graph.h" /*  /usr/ucb/cc surf_min.c -lX11 -I/usr/openwin/share/include -L. -lm  */
#include </home/eleves/jmfriedt/c/def_surface.h>
#include <stdio.h>
void edit();                                                  /* declaration de procedure */

void rotation(point p)                   /*application de la matrice de rotation au noeud p*/
{float inter1,inter2;                                        /*tourne des angles du tableau*/
 inter1=cos(angle[0])*p[0]-sin(angle[0])*p[1];
 inter2=sin(angle[0])*p[0]+cos(angle[0])*p[1];
 inter2=cos(angle[1])*inter2-sin(angle[1])*p[2];
 p[0]=inter1;p[1]=inter2;
}
void postscript(float r)     /* sauvegarde de l'image en format postscript */
{char fichier[20],title[30];
 int x,y,z;
 point p1,p2,p3;
 FILE *f1;
 float big_tableau[nx][ny][2];    /* on va mettre tous les points tournes dans ce tableau */
 float minix,maxix,miniy,maxiy;/*pour faire les operations adequates av. de sauver l'image*/
 printf("nom du fichier ");scanf("%s",fichier);
if ((f1=fopen(fichier,"r"))!=NULL)        /* test de l'existence du fichier de sauvegarde */
   {printf(" attention le fichier existe : recommencer \n");}
else {
  printf("titre de moins de 30 characteres sans espace ");scanf("%s",title);
  f1=fopen(fichier,"w");                                                /* titre du file */
  for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++)
     {point p1;
      for (z=0;z<=2;z++) p1[z]=s[x][y].coord[z];                /* on copie ts les points */
      rotation(p1);for (z=0;z<=1;z++) big_tableau[x][y][z]=p1[z];/*tournes dans ce tableau*/
     }
 maxix=-100000.;minix=100000.;maxiy=-1000000.;miniy=10000000.;
 for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++)       /* recherche des min et max du tableau */
     {if (big_tableau[x][y][0]>=maxix) maxix=big_tableau[x][y][0];
      if (big_tableau[x][y][0]<=minix) minix=big_tableau[x][y][0];
      if (big_tableau[x][y][1]>=maxiy) maxiy=big_tableau[x][y][1];
      if (big_tableau[x][y][1]<=miniy) miniy=big_tableau[x][y][1];
     }
 for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++) {       /* tranlation et addition d'un cadre */
      big_tableau[x][y][0]=big_tableau[x][y][0]-minix+(maxix-minix)/10.;
      big_tableau[x][y][1]=big_tableau[x][y][1]-miniy+(maxiy-miniy)/10.;
                                             }              /* minix et miniy sont a 0.0 :*/
 maxix=maxix-minix+(maxix-minix)/10.;                /* on a tout translate a l'orgine 0 0 */
 maxiy=maxiy-miniy+(maxiy-miniy)/10.;   /* puis renverser l'image : origine en bas gauche */
for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++) big_tableau[x][y][1]=maxiy-big_tableau[x][y][1];
 minix=400/maxix;                                      /* variable temporaire d'homotetie */
```

```
 for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++)                          /*largeur choisie a 400 */
     {big_tableau[x][y][0]=big_tableau[x][y][0]*minix;
      big_tableau[x][y][1]=big_tableau[x][y][1]*minix;}
 maxix=400;maxiy=minix*maxiy;                                      /* suit l'entete postscript */
 fprintf(f1,"%%!PS-Adobe-2.0 EPSF \n %%%%Title: %s \n %%%%Creator: jmfriedt \n
         %%%%CreationDate: Tue Nov 22 18:28:59 1994 \n ",title);
 fprintf(f1,"%%%%For: jmfriedt (Jean-Michel Friedt) \n %%%%BoundingBox: %f %f %f %f \n
         %%%%Pages: 0 \n %%%%EndComments \n",0.,0.,maxix,maxiy);
 fprintf(f1,"gsave \n newpath \n ");     /* on y met tous les points : droite de (moveto) */
 for (x=0;x<=nx-2;x++) for (y=0;y<=ny-2;y++)                               /* jusqu'a (lineto) */
     if (s[x][y].genre!='l')                /* je trace la meme chose que dans affiche(); */
         {fprintf(f1," %f %f moveto",big_tableau[x][y][0],big_tableau[x][y][1]);
          fprintf(f1," %f %f lineto \n",big_tableau[x+1][y][0],big_tableau[x+1][y][1]);
          fprintf(f1," %f %f moveto",big_tableau[x][y][0],big_tableau[x][y][1]);
          fprintf(f1," %f %f lineto \n",big_tableau[x][y+1][0],big_tableau[x][y+1][1]);
         }
 for (x=0;x<=nx-2;x++)
     if (s[x][ny-1].genre!='l') {
         fprintf(f1," %f %f moveto",big_tableau[x][ny-1][0],big_tableau[x][ny-1][1]);
         fprintf(f1," %f %f lineto \n",big_tableau[x+1][ny-1][0],big_tableau[x+1][ny-1][1]);
                               }
 for (x=0;x<=ny-2;x++)
      if (s[nx-1][y].genre!='l'){
          fprintf(f1," %f %f moveto",big_tableau[nx-1][x][0],big_tableau[nx-1][x][1]);
         fprintf(f1," %f %f lineto \n",big_tableau[nx-1][x+1][0],big_tableau[nx-1][x+1][1]);
                                }
 for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++)
      if (s[x][y].genre=='l') {
      fprintf(f1," %f %f moveto",big_tableau[x][y][0],big_tableau[x][y][1]);
      fprintf(f1," %f %f lineto \n",
              big_tableau[s[x][y].voisins_droi[0][0]][s[x][y].voisins_droi[0][1]][0],
              big_tableau[s[x][y].voisins_droi[0][0]][s[x][y].voisins_droi[0][1]][1]);
      fprintf(f1," %f %f moveto",big_tableau[x][y][0],big_tableau[x][y][1]);
      fprintf(f1," %f %f lineto \n",
              big_tableau[s[x][y].voisins_droi[1][0]][s[x][y].voisins_droi[1][1]][0],
              big_tableau[s[x][y].voisins_droi[1][0]][s[x][y].voisins_droi[1][1]][1]);
      fprintf(f1," %f %f moveto",big_tableau[x][y][0],big_tableau[x][y][1]);
      fprintf(f1," %f %f lineto \n",
              big_tableau[s[x][y].voisins_droi[2][0]][s[x][y].voisins_droi[2][1]][0],
              big_tableau[s[x][y].voisins_droi[2][0]][s[x][y].voisins_droi[2][1]][1]);
      fprintf(f1," %f %f moveto",big_tableau[x][y][0],big_tableau[x][y][1]);
      fprintf(f1," %f %f lineto \n",
              big_tableau[s[x][y].voisins_droi[3][0]][s[x][y].voisins_droi[3][1]][0],
              big_tableau[s[x][y].voisins_droi[3][0]][s[x][y].voisins_droi[3][1]][1]);
                              }
  fprintf(f1,"0.1 setlinewidth \n stroke \n /Times-Roman findfont 16 scalefont setfont
       newpath 10 10 ",title);          /* conclusion de PostScript et affichage du titre */
  fprintf(f1,"moveto ( %s ) show \ngrestore \n showpage \n");
  fclose(f1);}    /*fin du if du debut (on ne fait tout ca que si le fichier n'existe pas)*/
  }
void affiche(float r)          /*affichage de la surface, chaque carre etant de longueur r*/
{int x,y,z;
 point p1,p2,p3;
 unsigned int win_w,win_h;
```

```
XClearWindow(display,window);                          /*on efface la fenetre courante...*/
win_w=DisplayWidth(display,sc_num)/3;
win_h=DisplayHeight(display,sc_num)/3;
for (x=0;x<=nx-2;x++) for (y=0;y<=ny-2;y++)   /*... et on va afficher tous les points ...*/
     if (s[x][y].genre!='l') {
        for (z=0;z<=2;z++) p1[z]=s[x][y].coord[z];rotation(p1);
        for (z=0;z<=2;z++) p2[z]=s[x+1][y].coord[z];rotation(p2);
        for (z=0;z<=2;z++) p3[z]=s[x][y+1].coord[z];rotation(p3);
        XDrawLine(display,window,graphic,(int)(r*p1[0])+win_w,(int)(r*p1[1])+win_h,
                  (int)(r*p2[0])+win_w,(int)(r*p2[1])+win_h);
        XDrawLine(display,window,graphic,(int)(r*p1[0])+win_w,(int)(r*p1[1])+win_h,
                  (int)(r*p3[0])+win_w,(int)(r*p3[1])+win_h);
                               }
for (x=0;x<=nx-2;x++)
     if (s[x][ny-1].genre!='l') {
        for (z=0;z<=2;z++) p1[z]=s[x][ny-1].coord[z];rotation(p1);
        for (z=0;z<=2;z++) p2[z]=s[x+1][ny-1].coord[z];rotation(p2);
        XDrawLine(display,window,graphic,(int)(r*p1[0])+win_w,(int)(r*p1[1])+win_h,
                  (int)(r*p2[0])+win_w,(int)(r*p2[1])+win_h);
                                }
for (x=0;x<=ny-2;x++)
     if (s[nx-1][y].genre!='l'){
         for (z=0;z<=2;z++) p1[z]=s[nx-1][x].coord[z];rotation(p1);
         for (z=0;z<=2;z++) p2[z]=s[nx-1][x+1].coord[z];rotation(p2);
         XDrawLine(display,window,graphic,(int)(r*p1[0])+win_w,(int)(r*p1[1])+win_h,
                   (int)(r*p2[0])+win_w,(int)(r*p2[1])+win_h);
                               }
for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++)
     if (s[x][y].genre=='l') {
         point p1,p2;
         for (z=0;z<=2;z++) p1[z]=s[x][y].coord[z];rotation(p1);
         for (z=0;z<=2;z++)
     p2[z]=s[s[x][y].voisins_droi[0][0]][s[x][y].voisins_droi[0][1]].coord[z];rotation(p2);
         XDrawLine(display,window,graphic,(int)(r*p1[0])+win_w,(int)(r*p1[1])+win_h,
                   (int)(r*p2[0])+win_w,(int)(r*p2[1])+win_h);
         for (z=0;z<=2;z++)
     p2[z]=s[s[x][y].voisins_droi[1][0]][s[x][y].voisins_droi[1][1]].coord[z];rotation(p2);
         XDrawLine(display,window,graphic,(int)(r*p1[0])+win_w,(int)(r*p1[1])+win_h,
                   (int)(r*p2[0])+win_w,(int)(r*p2[1])+win_h);
         for (z=0;z<=2;z++)
     p2[z]=s[s[x][y].voisins_droi[2][0]][s[x][y].voisins_droi[2][1]].coord[z];rotation(p2);
         XDrawLine(display,window,graphic,(int)(r*p1[0])+win_w,(int)(r*p1[1])+win_h,
                   (int)(r*p2[0])+win_w,(int)(r*p2[1])+win_h);
         for (z=0;z<=2;z++)
     p2[z]=s[s[x][y].voisins_droi[3][0]][s[x][y].voisins_droi[3][1]].coord[z];rotation(p2);
         XDrawLine(display,window,graphic,(int)(r*p1[0])+win_w,(int)(r*p1[1])+win_h,
                   (int)(r*p2[0])+win_w,(int)(r*p2[1])+win_h);
                             }  /*... et on affiche enfin les points calcules a l'ecran. */
XDrawString(display,window,graphic,10,15,"Press 1 to move",15);
XDrawString(display,window,graphic,10,25,"Press 2 to stop",15);
XShowSesame();
}
void tourne_surface(int editer) /* animation de la surface : attente de Events et gestion */
{float r=10;                   /* des consequences les events possibles sont dans l'ordre*/
```

```
  int z;                       /* : redessiner, appui d'un bouton, deplacement de la souris*/
  int xmouse,ymouse,xnew,ynew;
  char c;
  XEvent report;                                      /* structure contenant les evenements */
  affiche(r);XShowSesame();                                      /* affichage de la surface */
  while (1) {XNextEvent(display,&report);      /*attente de l'evenement puis traitement : */
            switch(report.type) {           /* si c'est un affichage (fenetre decouverte) */
              case Expose : { affiche(r);XShowSesame();} break;
              case ButtonPress : {                /* un bouton de mouse appuye : lequel ? */
                 if (report.xbutton.button==Button2) /*bouton 2 -> on va sortie ou editer */
                   {if (editer==1) {printf("quit or edit\n");scanf("%c",&c);
                      if (c=='q') {XFreeGC(display,graphic);XCloseDisplay(display);exit(1);}
                      if (c=='e') edit(); }
                    else {printf("bye *:) \n \x00d");XCloseDisplay(display);exit(1);}
                   }             } break;        /* si c'est bouton 3 on change la taille */
              case MotionNotify : if (report.xbutton.button==Button3)
                                     {xnew=report.xmotion.x;ynew=report.xmotion.y;
                                      if (xnew>xmouse) r=r+5; if (xnew<xmouse) r=r-5;
                                      xmouse=xnew; affiche(r);}
                                  if (report.xbutton.button==Button1) {
                                      xnew=report.xmotion.x;ynew=report.xmotion.y;
                                      if (xnew>xmouse) {angle[0]=angle[0]+0.1;};
                                      if (xnew<xmouse) {angle[0]=angle[0]-0.1;};
                                      xmouse=xnew;
                                     if (ynew<ymouse) {angle[1]=angle[1]-0.1;};
                                     if (ynew>ymouse) {angle[1]=angle[1]+0.1;};
                                     ymouse=ynew;affiche(r);} break;
              default:{XFlush;} break;
                                 }
            }
}
void prod_vect(point v1,point v2,point result)            /*produit vectoriel result=v1^v2*/
{result[0]=v1[1]*v2[2]-v1[2]*v2[1];         /* en fait il reste d'une version antecedante */
 result[1]=v1[2]*v2[0]-v1[0]*v2[2];                               /* mais ne sert plus ici ! */
 result[2]=v1[0]*v2[1]-v1[1]*v2[0];
}
   /* calcul de la position du point centrale x0,y0,z0 qui minimise localement la surface */
void trouve_pt(point v1,point v2,point v3,point v4,point result)
                                          /*on entre et sort le point a bouger par result */
{float A1,A2,A3,A4,B1,B2,B3,B4,C1,C2,C3,C4,D1,D2,D3,D4,detA,x1,x2,x3,x4,y1,y2,y3,y4,z1,z2,
       z3,z4,x0,y0,z0;
x1=v1[0];x2=v2[0]-x1;x3=v3[0]-x1;x4=v4[0]-x1;y1=v1[1];y2=v2[1]-y1;y3=v3[1]-y1;y4=v4[1]-y1;
z1=v1[2];z2=v2[2]-z1;z3=v3[2]-z1;z4=v4[2]-z1;x0=result[0];y0=result[1];z0=result[2];
 A1=2*(-z3*z4-y3*y2-y4*y3+z2*z2-z2*z3+y2*y2+z3*z3+y3*y3+y4*y4+z4*z4);
 B1=x2*y3+x3*y2+x3*y4+x4*y3-2*(x4*y4+x3*y3+x2*y2);
 C1=x4*z3+z4*x3+x3*z2+x2*z3-2*(x4*z4+x2*z2+x3*z3);
 D1=x2*y3*y2+x3*y2*y3+x4*z4*z3+x3*z4*z3+x3*y3*y4+x4*y3*y4+x3*z2*z3+
x2*z2*z3-(x3*z2*z2+x2*z3*z3+x2*y3*y3+x4*y3*y3+x3*y4*y4+x3*z4*z4+x4*z3*z3+x3*y2*y2);
 A2=x4*y3+y4*x3+x3*y2+x2*y3-2*(x4*y4+x2*y2+x3*y3);
 B2=2*(-z3*z4-x3*x2-x4*x3+z2*z2-z2*z3+x2*x2+z3*z3+x3*x3+x4*x4+z4*z4);
 C2=z2*y3+z3*y2+z3*y4+z4*y3-2*(z4*y4+z3*y3+z2*y2);
 D2=y2*x3*x2+y3*x2*x3+y4*z4*z3+y3*z4*z3+y3*x3*x4+y4*x3*x4+y3*z2*z3+
y2*z2*z3-(+y3*z2*z2+y2*z3*z3+y2*x3*x3+y4*x3*x3+y3*x4*x4+y3*z4*z4+y4*z3*z3+y3*x2*x2);
 A3=z2*x3+z3*x2+z3*x4+z4*x3-2*(z4*x4+z3*x3+z2*x2);
```

```
 B3=z4*y3+y4*z3+z3*y2+z2*y3-2*(z4*y4+z2*y2+z3*y3);
 C3=2*(-x3*x4-y3*y2-y4*y3+x2*x2-x2*x3+y2*y2+x3*x3+y3*y3+y4*y4+x4*x4);
 D3=z2*y3*y2+z3*y2*y3+z4*x4*x3+z3*x4*x3+z3*y3*y4+z4*y3*y4+z3*x2*x3+
z2*x2*x3-(z3*x2*x2+z2*x3*x3+z2*y3*y3+z4*y3*y3+z3*y4*y4+z3*x4*x4+z4*x3*x3+z3*y2*y2);
 detA=A1*B2*C3+A2*B3*C1+A3*B1*C2-A1*C2*B3-A2*C3*B1-A3*C1*B2;
 result[0]=x1-(D1*B2*C3+D2*B3*C1+D3*B1*C2-D1*C2*B3-D2*C3*B1-D3*C1*B2)/detA;
 result[1]=y1-(A1*D2*C3+A2*D3*C1+A3*D1*C2-A1*C2*D3-A2*C3*D1-A3*C1*D2)/detA;
 result[2]=z1-(A1*B2*D3+A2*B3*D1+A3*B1*D2-A1*D2*B3-A2*D3*B1-A3*D1*B2)/detA;
}
void sauve_surface(FILE *f1)/*sauvegarde des donnees definissant la surface ds le fichier */
{int x,y,i;                                                           /* ouvert au depart */
 fprintf(f1,"Surface ");                               /* entete de reconnaissance du fichier */
 fprintf(f1,"%i ",nx); fprintf(f1,"%i ",ny);                             /* tailles du tableau */
 for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++)    /* sauvegarde de tous les points et leurs */
      {for (i=0;i<=2;i++) fprintf(f1,"%f ",s[x][y].coord[i]);         /* caracteristiques */
       fprintf(f1,"%c ",s[x][y].genre);                     /* : coord., type et voisins si l */
       if (s[x][y].genre=='l') {int a,b;
         for (a=0;a<=3;a++) for (b=0;b<=1;b++) fprintf(f1,"%i ",s[x][y].voisins_diag[a][b]);
         for (a=0;a<=3;a++) for (b=0;b<=1;b++) fprintf(f1,"%i ",s[x][y].voisins_droi[a][b]);
                                }
       } fclose(f1);
}
void load(FILE *f1) /*chargement des donnees definissant la surface dans le fichier ouvert*/
{char c[1];                                                                    /* au depart */
 char entete[100];       /* nom du fichier a charger */
 int x,y,i,taillex,tailley;
 fscanf(f1,"%s ",entete);
 if ((entete[0]!='S')||(entete[1]!='u'))           /* verification de l'entete de fichier */
     {printf("Ce fichier n'a pas ete cree pour ce programme");exit(1);}
 fscanf(f1,"%i ",&taillex);fscanf(f1,"%i ",&tailley);      /* verification taille tableau */
 if ((taillex!=nx)||(tailley!=ny))
     {printf("Wrong size : set n to %i or %i. Now exiting\n",taillex,tailley);exit(1);}
 printf("taille de l'objet : %i  %i",taillex,tailley);
 for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++)                   /* chargement des elements */
   {for (i=0;i<=2;i++) fscanf(f1,"%f ",&(s[x][y].coord[i]));
      fscanf(f1,"%c ",&(s[x][y].genre));printf("reading %i %i %c\n",x,y,s[x][y].genre);
        if (s[x][y].genre=='l') {int a,b,z;
        s[x][y].voisins_diag=(int**)malloc(4*sizeof(int*));
        for (z=0;z<=3;z++) s[x][y].voisins_diag[z]=(int*)malloc(2*sizeof(int));
        s[x][y].voisins_droi=(int**)malloc(4*sizeof(int*));
        for (z=0;z<=3;z++) s[x][y].voisins_droi[z]=(int*)malloc(2*sizeof(int));
        for (a=0;a<=3;a++) for (b=0;b<=1;b++) fscanf(f1,"%i ",&s[x][y].voisins_diag[a][b]);
        for (a=0;a<=3;a++) for (b=0;b<=1;b++) fscanf(f1,"%i ",&s[x][y].voisins_droi[a][b]);
                                 }
    }
 printf("done \n");fclose(f1);
}
void edit()     /*procedure donnant quelques informations succintes sur la surface obtenue*/
{char entree;
 int xp,yp,r=20;
 int sort=1;
 do {XDrawString(display,window,graphic,10,35,"EDIT mode",9);XShowSesame();
     printf("? >");
     scanf("%c %i %i",&entree,&xp,&yp);                               /*entree de la commande*/
```

```
      affiche(r);
      printf(" %c %i %i\n",entree,xp,yp);
      switch (entree) {                                      /* traitement de la commande */
         case 'h': printf("x, y, z, status s, pos p, quit q always            /* bref help */
                          <1-letters-string point point>\n");break;
         case 'x': printf("value x %f \n",s[xp][yp].coord[0]);break;   /*affiche la valeur */
         case 'y': printf("value y %f \n",s[xp][yp].coord[1]);break;   /* d'une coordonnee */
         case 'z': printf("value z %f \n",s[xp][yp].coord[2]);break;
         case 's': printf("point %i %i is %c\n",xp,yp,s[xp][yp].genre);break;     /* genre */
         case 'q': sort=0;break;                                       /* quitte l'editeur */
         case 'r': postscript(10);break;     /* sauvegarde de l'image au format PostScript */
         case 'u': {r=r+2;affiche(r);XDrawString(display,window,graphic,10,35,"EDIT mode",9);
                    XShowSesame();} break;                             /* agrandit l'image */
         case 'd': {r=r-2;affiche(r);XDrawString(display,window,graphic,10,35,"EDIT mode",9);
                    XShowSesame();} break;                     /* rend l'image plus petite */
         case 'p': {unsigned int win_w,win_h;point p;int z; /*affichage de la position d'un*/
                    win_w=DisplayWidth(display,sc_num)/3;                         /* point */
                    win_h=DisplayHeight(display,sc_num)/3;
                    for (z=0;z<=2;z++) p[z]=s[xp][yp].coord[z];rotation(p);
                        XDrawLine(display,window,graphic,(int)(r*p[0])+win_w-10,
                                  (int)(r*p[1])+win_h-10,(int)(r*p[0])+win_w+10,
                                  (int)(r*p[1])+win_h+10);
                        XDrawLine(display,window,graphic,
                                  (int)(r*p[0])+win_w-10,(int)(r*p[1])+win_h+10,
                                  (int)(r*p[0])+win_w+10,(int)(r*p[1])+win_h-10);
                    XShowSesame();
                    } break;
         default:break;
                      } printf("done\n");
      }
while (sort==1);
}

speedup() {setitimer(0,htimer,0);}   /*procedure prise de S. Blondeel, cf News du 29/9/94 */
init_kbd()
  { int v[]={(int)speedup,0,2};
    htimer[3]=2000;
    system("stty -isig raw -echo stop u");
    sigvec(14,v,0);
    speedup();
}


void remplit_sinter()                  /*remplissage initial du tableau intermediaire : */
{ int x,y,z;                                                    /*copie de s dans sinter*/
  for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++) for (z=0;z<=2;z++)
       sinter[x][y][z]=s[x][y].coord[z];
  printf("surface initialized \n");
}

void cree_surf_min(int show,FILE *f1)  /* calcul de la position des points rendant l'aire */
{int x,y,z;                                                                  /* minimale */
 char poubelle[200];
 int bouge,c,t;
```

```
 int sort_vite=0;                                                          /* flags */
 int nbdec=100;                                                   /*coef de precision*/
 init_kbd();                                                /* inhibe l'attente clavier */
 scanf("%sn",poubelle);     /* vide le buffer clavier pour l'attente d'une touche pressee */
 do {bouge=0;
 for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++) for (z=0;z<=2;z++)
      s[x][y].coord[z]=sinter[x][y][z];
      for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++) {       /* voisins diagonaux normaux (b)*/
      if (s[x][y].genre=='b') trouve_pt(s[x-1][y-1].coord,
                                    s[x+1][y-1].coord,s[x+1][y+1].coord,
                                    s[x-1][y+1].coord,sinter[x][y]);
      if (s[x][y].genre=='l') {point a1,a2,a3,a4;             /* voisins diagonaux pour l */
                               for (z=0;z<=2;z++)
                  a1[z]=s[s[x][y].voisins_diag[0][0]][s[x][y].voisins_diag[0][1]].coord[z];
                               for (z=0;z<=2;z++)
                  a2[z]=s[s[x][y].voisins_diag[1][0]][s[x][y].voisins_diag[1][1]].coord[z];
                               for (z=0;z<=2;z++)
                  a3[z]=s[s[x][y].voisins_diag[2][0]][s[x][y].voisins_diag[2][1]].coord[z];
                               for (z=0;z<=2;z++)
                  a4[z]=s[s[x][y].voisins_diag[3][0]][s[x][y].voisins_diag[3][1]].coord[z];
                               trouve_pt(a1,a2,a3,a4,sinter[x][y]);
                              }
                                                    }
      for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++) {         /* voisins directs normaux (b)*/
      if (s[x][y].genre=='b') {
    trouve_pt(s[x][y-1].coord,s[x+1][y].coord,s[x][y+1].coord,s[x-1][y].coord,sinter[x][y]);
    if ((abs((int)((s[x][y].coord[0]-sinter[x][y][0])*nbdec))!=0)|| /*si le point a bouge */
       (abs((int)((s[x][y].coord[1]-sinter[x][y][1])*nbdec))!=0)||  /*on set un flag pour */
       (abs((int)((s[x][y].coord[2]-sinter[x][y][2])*nbdec))!=0))  /* recommencer l'etude */
        {printf("%i %i\n \x00d",x,y);bouge=1;}            /* coordonnees du point deplace */
                              }
    if (s[x][y].genre=='l') {point a1,a2,a3,a4;                   /*voisins directs pour l */
                             for (z=0;z<=2;z++)
                  a1[z]=s[s[x][y].voisins_droi[0][0]][s[x][y].voisins_droi[0][1]].coord[z];
                             for (z=0;z<=2;z++)
                  a2[z]=s[s[x][y].voisins_droi[1][0]][s[x][y].voisins_droi[1][1]].coord[z];
                             for (z=0;z<=2;z++)
                  a3[z]=s[s[x][y].voisins_droi[2][0]][s[x][y].voisins_droi[2][1]].coord[z];
                             for (z=0;z<=2;z++)
                  a4[z]=s[s[x][y].voisins_droi[3][0]][s[x][y].voisins_droi[3][1]].coord[z];
                             trouve_pt(a1,a2,a3,a4,sinter[x][y]);
                              }
                                                     }
     if (show==1) affiche(7); c=getchar();           /* on sort si une touche est appuyee */
     if (c>=0) {bouge=0;printf("ATTENTION : l'etude n'est pas terminee \n \x00d");
     t=sigblock(8192);}
    } while (bouge==1);
     sauve_surface(f1);                   /* on sauve le calcul qui vient d'etre effectue */
     if (show==0) exit(1);              /* pas d'affichage : on peut tout de suite partir */
}
void commence_work(char fichier[20],int ouvert)      /*le fichier n'existe pas et il faut */
{FILE *f1;                                                 /* commencer tout le calcul */
 int show,editer;                                                          /* flags */
 printf("opening file for savings\n"); f1=fopen(fichier,"w");    /* fichier de sauvegarde */
```

```
 if (ouvert==0) init_surface();      /* ouvert=1 si deja charge : si ouvert=0 alors creer */
 remplit_sinter();                              /* avec init_surface et copie dans sinter*/
 printf("affichage graphique ? ");
 scanf("%s",fichier); if (fichier[0]=='n') show=0; else show=1;
 printf("edit ?");scanf("%s",fichier);if (fichier[0]!='n') {editer=1;show=1;} else editer=0;
 if (show==1) Xinitgraph();                                /* Initialisations graphiques */
 if (editer==1) edit();
 cree_surf_min(show,f1);       /* on fait tout le calcul de minimalisation de la surface */
 tourne_surface(0);
}
void commence_load(FILE *f1)             /*le fichier existe deja et il suffit de charger*/
{char c,fichier[20];
 load(f1); c=getchar();
 printf("faut il faire un Calcul ou seulement Afficher \n");
 do {scanf("%c",&c);
    if (c=='a') {Xinitgraph();tourne_surface(1);}
    else {printf("nom du fichier ou il faudra sauver le resultat \n");
         scanf("%s",fichier);  /*on appelle le calcul apres avoir charge un ancien fichier*/
         commence_work(fichier,1);}                                        /*et on continue*/
     } while((c!='a')&&(c!='c'));
}
main()    /* PROGRAMME PRINCIPAL : entree du fichier a utiliser et appel des differentes */
{char fichier[20];                                                     /*  procedures. */
 FILE *f1;
 printf("programme de trace de surface minimale :\n
         entrer nom de fichier \n");
 scanf("%s",fichier);if ((f1=fopen(fichier,"r"))!=NULL)
                         {commence_load(f1);} else {commence_work(fichier,0);}
}
```

7.4 Définition de quelques surfaces

7.4.1 La cathénoïde

```
#define nx  30
#define ny  30
#include </home/eleves/jmfriedt/c/definitype.h>
void init_surface()                                       /* definition de la cathenoide */
{float rayon=30.;                                          /* rayon des cercles extremes */
 float rcentre=5;           /* le rayon parabolique ne doit pas atteindre 0 : valeur min */
 float pas=(float)ny;
 int x,y,z;
 float cc=rayon;
 float cb=-4*(rayon-rcentre)/(float)(nx-1);
 float ca=4*(rcentre-rayon-cb/2.*(float)(nx-1))/(float)((nx-1)*(nx-1));
 for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++) {s[x][y].coord[0]=x;
                                     s[x][y].genre='b';} /* tous les pts bougent */
 for (x=0;x<=ny-1;x++) {s[0][x].genre='f';s[nx-1][x].genre='f';}          /*sauf les bords*/
 for (x=1;x<=nx-2;x++) {s[x][ny-1].genre='l';s[x][0].genre='l';} / *recollement de 2 bords*/
 for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++)
     {float r;
      r=ca*(float)(x*x)+cb*(float)(x)+cc;     /* prendre r=rayon pour un cylindre initial */
      s[x][y].coord[1]=r*cos((float)y*6.28/pas);            /*les cerceaux aux extremites*/
      s[x][y].coord[2]=r*sin((float)y*6.28/pas);}
```

```
 for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++) if (s[x][y].genre=='l')/*initialise pointeurs*/
      {int z;
       s[x][y].voisins_diag=(int**)malloc(4*sizeof(int*));
       for (z=0;z<=3;z++) s[x][y].voisins_diag[z]=(int*)malloc(2*sizeof(int));
       s[x][y].voisins_droi=(int**)malloc(4*sizeof(int*));
       for (z=0;z<=3;z++) s[x][y].voisins_droi[z]=(int*)malloc(2*sizeof(int));
      }
/* definition des voisins au recollement pour former le cylindre : 2 points opposes ont   */
/* memes voisins et semblent donc identiques au calcul. Le plan est referme en un cylindre*/
for (x=1;x<=nx-2;x++) {
     s[x][ny-1].voisins_diag[0][0]=x+1; s[x][ny-1].voisins_droi[0][0]=x+1;
     s[x][ny-1].voisins_diag[0][1]=ny-2;s[x][ny-1].voisins_droi[0][1]=ny-1;
     s[x][ny-1].voisins_diag[1][0]=x-1; s[x][ny-1].voisins_droi[1][0]=x;
     s[x][ny-1].voisins_diag[1][1]=ny-2;s[x][ny-1].voisins_droi[1][1]=0;
     s[x][ny-1].voisins_diag[2][0]=x-1; s[x][ny-1].voisins_droi[2][0]=x-1;
     s[x][ny-1].voisins_diag[2][1]=0;   s[x][ny-1].voisins_droi[2][1]=ny-1;
     s[x][ny-1].voisins_diag[3][0]=x+1; s[x][ny-1].voisins_droi[3][0]=x;
     s[x][ny-1].voisins_diag[3][1]=0;   s[x][ny-1].voisins_droi[3][1]=ny-2;

     s[x][0].voisins_diag[0][0]=x+1;   s[x][0].voisins_droi[0][0]=x+1;
     s[x][0].voisins_diag[0][1]=1;     s[x][0].voisins_droi[0][1]=0;
     s[x][0].voisins_diag[1][0]=x+1;   s[x][0].voisins_droi[1][0]=x;
     s[x][0].voisins_diag[1][1]=ny-1;  s[x][0].voisins_droi[1][1]=ny-1;
     s[x][0].voisins_diag[2][0]=x-1;   s[x][0].voisins_droi[2][0]=x-1;
     s[x][0].voisins_diag[2][1]=ny-1;  s[x][0].voisins_droi[2][1]=0;
     s[x][0].voisins_diag[3][0]=x-1;   s[x][0].voisins_droi[3][0]=x;
     s[x][0].voisins_diag[3][1]=1;     s[x][0].voisins_droi[3][1]=1;
                      }
}
```

7.4.2 L'helicoïde

```
#define nx  30
#define ny  30
#include </home/eleves/jmfriedt/c/definitype.h>
void init_surface()
{float rayon=5.;                                                      /* rayon de l'helice */
 float tours=2.;                                            /* nombre de tours de l'helice */
 float height=15.;                                            /*hauteur totale de l'helice */
 int x,y,z;
 for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++) {
     s[x][y].coord[2]=height*x/((float)(nx-1))-height/2;
     s[x][y].coord[1]=rayon*(float)y/((float)(ny-1))*sin((float)x*6.28/(float)(nx-1)*tours);
     s[x][y].coord[0]=rayon*(float)y/((float)(ny-1))*cos((float)x*6.28/(float)(nx-1)*tours);
     s[x][y].genre='b';}   /* definition parametrique de l'helice a partir du quadrillage */
 for (x=0;x<=ny-1;x++) {s[0][x].genre='f';s[nx-1][x].genre='f';}
 for (x=0;x<=nx-1;x++) {s[x][ny-1].genre='f';s[x][0].genre='f';}
 }
```

7.4.3 La feuille de caoutchouc tendue

```
#define nx  50                             /* dimensions du plan supportant la surface */
#define ny  50
#include </home/eleves/jmfriedt/c/definitype.h>
```

```
void init_surface()
{int x,y,z;
 for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++)
     {s[x][y].coord[2]=0;s[x][y].coord[0]=x;s[x][y].coord[1]=y;   /* quadrillqge regulier */
      s[x][y].genre='b';}                            /* tous les points bougent par defaut */
 for (x=0;x<=nx-1;x++) {s[x][0].genre='f';s[x][ny-1].genre='f';}  /* les bords sont fixes */
 for (x=0;x<=ny-1;x++) {s[0][x].genre='f';s[nx-1][x].genre='f';}
s[nx/2][ny/2].coord[2]=nx/2;s[nx/2][ny/2+2].coord[2]=-nx/2;s[nx-4][ny-4].coord[2]=nx/3;
s[nx/2][ny/2].genre='f';s[nx/2][ny/2+2].genre='f';s[nx-4][ny-4].genre='f';
}                                  /* on eleve et rend fixes quelques points de la surface */
```

7.4.4 Le cerceau courbé

```
#define nx  20
#define ny  20
#include </home/eleves/jmfriedt/c/definitype.h>
void init_surface()
{float rayon=5.;                                 /*rayon du cerceau support*/
 int x,y,z;
 for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++) {s[x][y].coord[0]=rayon*(float)x;
     if (y>=ny/2) s[x][y].coord[1]=rayon*sqrt((float)nx*(float)x-(float)(x*x)); /* cercle */
     if (y<ny/2) s[x][y].coord[1]=-rayon*sqrt((float)nx*(float)x-(float)(x*x));
     s[x][y].coord[2]=rayon*(float)nx/4.*cos(6.28*(float)x/(float)nx);/*courbure du cercle*/
     s[x][y].genre='b';}                             /*tous les points bougent par defaut*/
 for (x=0;x<=ny-1;x++) {s[0][x].genre='f';s[nx-1][x].genre='f';} /*les bords restent fixes*/
 for (x=0;x<=nx-1;x++) {s[x][ny-1].genre='f';s[x][0].genre='f';}
}
```

7.4.5 La surface à trois trous

```
#define nx  30                                            /* dimensions de la surface */
#define ny  31
#include </home/eleves/jmfriedt/c/definitype.h>
void init_surface()
{float rayon=30.;                                     /* rayon du grand cercle superieur */
 float rayon_petit=12.;                           /* rayon des petits cercles inferieurs */
 float droite=33.;           /* longueur du segment joignant les deux cercles inferieurs */
 int m1,m2,m3,m4,m5;
 float pas=(float)ny;
 int x,y,z;
m1=3;m2=13;m3=18;m4=27;m5=ny-1;/*intervalles : demi-segment;cercle;segment;cercle;demi-seg*/
 for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++) {s[x][y].coord[0]=x;s[x][y].genre='b';}
 for (x=0;x<=ny-1;x++) {s[0][x].genre='f';s[nx-1][x].genre='f';}          /* bords fixes */
 for (x=1;x<=nx-2;x++) {s[x][ny-1].genre='l';s[x][0].genre='l';}      /* jonction du bord */
 for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++)                          /* cercle superieur */
     {s[x][y].coord[1]=rayon*cos((float)y*6.28/pas);
     s[x][y].coord[2]=rayon*sin((float)y*6.28/pas);}
 for (x=0;x<=ny-1;x++)
     {if (x<=m1) {s[nx-1][x].coord[2]=droite/2.*(float)(x)/(float)(m1);/* 1er 1/2 segment */
 s[nx-1][x].coord[1]=0.;                                          /* pour x [0,m1] */
 s[nx-1][x].genre='l';}
      if ((x<=m2)&&(x>=m1+1))                                /* cercle pour x [m1+1,m2]*/
         {s[nx-1][x].coord[2]=droite/2.+rayon_petit-rayon_petit*cos((float)(x-m1-1)
                                                     *6.28/(float)(m2-1-m1));
```

```
          s[nx-1][x].coord[1]=rayon_petit*sin((float)(x-m1-1)*6.28/(float)(m2-1-m1));}
      if ((x<=m3)&&(x>=m2+1))                               /* segment pour x [m2+1,m3] */
         {s[nx-1][x].coord[2]=droite/2.-droite*(float)(x-m2-1)/(float)(m3-m2-1);
          s[nx-1][x].coord[1]=0.;
          s[nx-1][x].genre='l';}                /* le segment liant les cercles est souple */
      if ((x<=m4)&&(x>=m3+1))
         {s[nx-1][x].coord[2]=-droite/2.-rayon_petit+rayon_petit*cos((float)(x-m3-1)
                                                            *6.28/(float)(m4-m3-1));
          s[nx-1][x].coord[1]=-rayon_petit*sin((float)(x-m3-1)*6.28/(float)(m4-m3-1));}
      if (x>=m4+1)                              /* demi-segment de retour pour x [m3+1,m4] */
         {s[nx-1][x].coord[2]=-droite/2.+droite/2*(float)(x-m4-1)/(float)(m5-m4-1);
          s[nx-1][x].coord[1]=0.;
          s[nx-1][x].genre='l';}
     }                                          /* allocation memoire pour les points lies */
for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++) if (s[x][y].genre=='l')
    {int z;
     s[x][y].voisins_diag=(int**)malloc(4*sizeof(int*));
     for (z=0;z<=3;z++) s[x][y].voisins_diag[z]=(int*)malloc(2*sizeof(int));
     s[x][y].voisins_droi=(int**)malloc(4*sizeof(int*));
     for (z=0;z<=3;z++) s[x][y].voisins_droi[z]=(int*)malloc(2*sizeof(int));
    }                                                     /* liaison des points de la base */
for (x=1;x<=m1;x++) {                     /* liaison entre 0 et m1 (1/2 segment aller) */
    s[nx-1][x].voisins_diag[0][0]=nx-2;        s[nx-1][x].voisins_droi[0][0]=nx-1;
    s[nx-1][x].voisins_diag[0][1]=x-1;         s[nx-1][x].voisins_droi[0][1]=x-1;
    s[nx-1][x].voisins_diag[1][0]=nx-2;        s[nx-1][x].voisins_droi[1][0]=nx-1;
    s[nx-1][x].voisins_diag[1][1]=x+1;         s[nx-1][x].voisins_droi[1][1]=x+1;
    s[nx-1][x].voisins_diag[2][0]=nx-1;        s[nx-1][x].voisins_droi[2][0]=nx-2;
    s[nx-1][x].voisins_diag[2][1]=m2+m1-x+1+1; s[nx-1][x].voisins_droi[2][1]=x;
    s[nx-1][x].voisins_diag[3][0]=nx-1;        s[nx-1][x].voisins_droi[3][0]=nx-1;
    s[nx-1][x].voisins_diag[3][1]=m2+m1-x-1+1; s[nx-1][x].voisins_droi[3][1]=m2+m1-x+1;
}
for (x=m2+1;x<=m2+m1-1;x++)         /* liaison entre m2 et m2+m1 (1/2 segment de retour) */
    {s[nx-1][x].voisins_diag[0][0]=nx-2;        s[nx-1][x].voisins_droi[0][0]=nx-1;
     s[nx-1][x].voisins_diag[0][1]=x-1;         s[nx-1][x].voisins_droi[0][1]=x-1;
     s[nx-1][x].voisins_diag[1][0]=nx-2;        s[nx-1][x].voisins_droi[1][0]=nx-1;
     s[nx-1][x].voisins_diag[1][1]=x+1;         s[nx-1][x].voisins_droi[1][1]=x+1;
     s[nx-1][x].voisins_diag[2][0]=nx-1;        s[nx-1][x].voisins_droi[2][0]=nx-2;
     s[nx-1][x].voisins_diag[2][1]=m1+m2-x+1+1; s[nx-1][x].voisins_droi[2][1]=x;
     s[nx-1][x].voisins_diag[3][0]=nx-1;        s[nx-1][x].voisins_droi[3][0]=nx-1;
     s[nx-1][x].voisins_diag[3][1]=m1+m2-x-1+1; s[nx-1][x].voisins_droi[3][1]=m1+m2-x+1;
    }
for (x=m2+m1+1;x<=m3;x++)      /* liaison entre m2+m1 et m3 (2eme 1/2 segment de retour) */
    {s[nx-1][x].voisins_diag[0][0]=nx-2;           s[nx-1][x].voisins_droi[0][0]=nx-1;
     s[nx-1][x].voisins_diag[0][1]=x-1;            s[nx-1][x].voisins_droi[0][1]=x-1;
     s[nx-1][x].voisins_diag[1][0]=nx-2;           s[nx-1][x].voisins_droi[1][0]=nx-1;
     s[nx-1][x].voisins_diag[1][1]=x+1;            s[nx-1][x].voisins_droi[1][1]=x+1;
     s[nx-1][x].voisins_diag[2][0]=nx-1;           s[nx-1][x].voisins_droi[2][0]=nx-2;
     s[nx-1][x].voisins_diag[2][1]=m5-x+m2+m1+1+1; s[nx-1][x].voisins_droi[2][1]=x;
     s[nx-1][x].voisins_diag[3][0]=nx-1;           s[nx-1][x].voisins_droi[3][0]=nx-1;
    s[nx-1][x].voisins_diag[3][1]=m5-x+m2+m1+1-1;s[nx-1][x].voisins_droi[3][1]=m5-x+m2+m1+1;
    }
for (x=m4+1;x<=m5-1;x++)              /* liaison entre m4 et la fin  (1/2 segment retour) */
    {s[nx-1][x].voisins_diag[0][0]=nx-2;       s[nx-1][x].voisins_droi[0][0]=nx-1;
     s[nx-1][x].voisins_diag[0][1]=x-1;        s[nx-1][x].voisins_droi[0][1]=x-1;
```

```
     s[nx-1][x].voisins_diag[1][0]=nx-2;       s[nx-1][x].voisins_droi[1][0]=nx-1;
     s[nx-1][x].voisins_diag[1][1]=x+1;        s[nx-1][x].voisins_droi[1][1]=x+1;
     s[nx-1][x].voisins_diag[2][0]=nx-1;       s[nx-1][x].voisins_droi[2][0]=nx-2;
     s[nx-1][x].voisins_diag[2][1]=m3-x+m4+1+1;s[nx-1][x].voisins_droi[2][1]=x;
     s[nx-1][x].voisins_diag[3][0]=nx-1;       s[nx-1][x].voisins_droi[3][0]=nx-1;
     s[nx-1][x].voisins_diag[3][1]=m3-x+m4-1+1;s[nx-1][x].voisins_droi[3][1]=m3-x+m4+1;
    }                                                        /*apres je traite les cas...*/
/* ... particuliers des bords : 0, nx-1,m2+m1 (separation entre les segments en 0 et ny-1)*/
s[nx-1][0].voisins_diag[0][0]=nx-2;   s[nx-1][0].voisins_droi[0][0]=nx-2;
s[nx-1][0].voisins_diag[0][1]=m5-1;   s[nx-1][0].voisins_droi[0][1]=0;
s[nx-1][0].voisins_diag[1][0]=nx-2;   s[nx-1][0].voisins_droi[1][0]=nx-1;
s[nx-1][0].voisins_diag[1][1]=1;      s[nx-1][0].voisins_droi[1][1]=m5;
s[nx-1][0].voisins_diag[2][0]=nx-1;   s[nx-1][0].voisins_droi[2][0]=nx-1;
s[nx-1][0].voisins_diag[2][1]=m2+m1+1;s[nx-1][0].voisins_droi[2][1]=1;
s[nx-1][0].voisins_diag[3][0]=nx-1;   s[nx-1][0].voisins_droi[3][0]=nx-1;
s[nx-1][0].voisins_diag[3][1]=m2+m1-1;s[nx-1][0].voisins_droi[3][1]=m1+m2;

s[nx-1][m5].voisins_diag[0][0]=nx-2;   s[nx-1][m5].voisins_droi[0][0]=nx-1;
s[nx-1][m5].voisins_diag[0][1]=1;      s[nx-1][m5].voisins_droi[0][1]=m2+m1;
s[nx-1][m5].voisins_diag[1][0]=nx-2;   s[nx-1][m5].voisins_droi[1][0]=nx-1;
s[nx-1][m5].voisins_diag[1][1]=m5-1;   s[nx-1][m5].voisins_droi[1][1]=m5;
s[nx-1][m5].voisins_diag[2][0]=nx-1;   s[nx-1][m5].voisins_droi[2][0]=nx-2;
s[nx-1][m5].voisins_diag[2][1]=m2+m1-1;s[nx-1][m5].voisins_droi[2][1]=m5;
s[nx-1][m5].voisins_diag[3][0]=nx-1;   s[nx-1][m5].voisins_droi[3][0]=nx-1;
s[nx-1][m5].voisins_diag[3][1]=m2+m1+1;s[nx-1][m5].voisins_droi[3][1]=m2+m1;

s[nx-1][m2+m1].voisins_diag[0][0]=nx-2;   s[nx-1][m2+m1].voisins_droi[0][0]=nx-1;
s[nx-1][m2+m1].voisins_diag[0][1]=m2+m1-1;s[nx-1][m2+m1].voisins_droi[0][1]=m2+m1+1;
s[nx-1][m2+m1].voisins_diag[1][0]=nx-2;   s[nx-1][m2+m1].voisins_droi[1][0]=nx-1;
s[nx-1][m2+m1].voisins_diag[1][1]=m2+m1+1;s[nx-1][m2+m1].voisins_droi[1][1]=m2+m1-1;
s[nx-1][m2+m1].voisins_diag[2][0]=nx-1;   s[nx-1][m2+m1].voisins_droi[2][0]=nx-1;
s[nx-1][m2+m1].voisins_diag[2][1]=1;      s[nx-1][m2+m1].voisins_droi[2][1]=m5;
s[nx-1][m2+m1].voisins_diag[3][0]=nx-1;   s[nx-1][m2+m1].voisins_droi[3][0]=nx-2;
s[nx-1][m2+m1].voisins_diag[3][1]=m5-1;   s[nx-1][m2+m1].voisins_droi[3][1]=m2+m1;
/* liaison pour former un cylindre : repris de la definition de la cathenoide */
for (x=1;x<=nx-2;x++)
    {s[x][ny-1].voisins_diag[0][0]=x+1; s[x][ny-1].voisins_droi[0][0]=x+1;
     s[x][ny-1].voisins_diag[0][1]=ny-2;s[x][ny-1].voisins_droi[0][1]=ny-1;
     s[x][ny-1].voisins_diag[1][0]=x-1; s[x][ny-1].voisins_droi[1][0]=x;
     s[x][ny-1].voisins_diag[1][1]=ny-2;s[x][ny-1].voisins_droi[1][1]=0;
     s[x][ny-1].voisins_diag[2][0]=x-1; s[x][ny-1].voisins_droi[2][0]=x-1;
     s[x][ny-1].voisins_diag[2][1]=0;   s[x][ny-1].voisins_droi[2][1]=ny-1;
     s[x][ny-1].voisins_diag[3][0]=x+1; s[x][ny-1].voisins_droi[3][0]=x;
     s[x][ny-1].voisins_diag[3][1]=0;   s[x][ny-1].voisins_droi[3][1]=ny-2;

     s[x][0].voisins_diag[0][0]=x+1; s[x][0].voisins_droi[0][0]=x+1;
     s[x][0].voisins_diag[0][1]=1;   s[x][0].voisins_droi[0][1]=0;
     s[x][0].voisins_diag[1][0]=x+1; s[x][0].voisins_droi[1][0]=x;
     s[x][0].voisins_diag[1][1]=ny-1;s[x][0].voisins_droi[1][1]=ny-1;
     s[x][0].voisins_diag[2][0]=x-1; s[x][0].voisins_droi[2][0]=x-1;
     s[x][0].voisins_diag[2][1]=ny-1;s[x][0].voisins_droi[2][1]=0;
     s[x][0].voisins_diag[3][0]=x-1; s[x][0].voisins_droi[3][0]=x;
     s[x][0].voisins_diag[3][1]=1;   s[x][0].voisins_droi[3][1]=1;
    }
```

```
}
```

7.4.6 Essai sur une surface qui se coupe

```
#define nx  10 /*dimensions du plan qui sera replie sur lui meme          |\            */
#define ny  80                                                     /*     | \           */
#include </home/eleves/jmfriedt/c/definitype.h>                    /*     |  \          */
void init_surface()                                                /*   --+---          */
{int x,y,z;                                                        /*     |             */
 for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++) {s[x][y].coord[0]=x;s[x][y].genre='b';}
 for (x=0;x<=ny-1;x++) {s[0][x].genre='f';s[nx-1][x].genre='f';}          /* bords fixes */
 for (x=1;x<=nx-2;x++) {s[x][ny-1].genre='f';s[x][0].genre='f';}
 for (x=1;x<=nx-2;x++) {s[x][ny/5].genre='l';s[x][4*ny/5].genre='l';}     /* intersection */
 for (x=0;x<=nx-1;x++) for (y=0;y<=ny/5-1;y++) /*de la bulle avec elle meme au croisement */
        {s[x][y].coord[0]=x;s[x][y+ny/5].coord[0]=x;
         s[x][y].coord[1]=y;s[x][y+ny/5].coord[1]=y;/*premier plan horizontal et vertical */
         s[x][y].coord[2]=0;s[x][y+ny/5].coord[2]=0;
        s[x][2*ny/5+y].coord[0]=x;
        s[x][2*ny/5+y].coord[1]=2*ny/5-y;                                /* plan oblique */
        s[x][2*ny/5+y].coord[2]=y;
       s[x][3*ny/5+y].coord[0]=x;     s[x][4*ny/5+y].coord[0]=x;
       s[x][3*ny/5+y].coord[1]=ny/5;s[x][4*ny/5+y].coord[1]=ny/5;/*3eme plan horiz et vert*/
       s[x][3*ny/5+y].coord[2]=ny/5-y;s[x][4*ny/5+y].coord[2]=-y;
 }
 for (x=0;x<=nx-1;x++) for (y=0;y<=ny-1;y++) if (s[x][y].genre=='l') /*init pointeurs de l*/
{int z;
 s[x][y].voisins_diag=(int**)malloc(4*sizeof(int*));
 for (z=0;z<=3;z++) s[x][y].voisins_diag[z]=(int*)malloc(2*sizeof(int));
 s[x][y].voisins_droi=(int**)malloc(4*sizeof(int*));
 for (z=0;z<=3;z++) s[x][y].voisins_droi[z]=(int*)malloc(2*sizeof(int));
}         /*on joint correctement les points : voir commentaire sur la cathenoide */
 for (x=1;x<=nx-2;x++) {
      s[x][ny/5].voisins_diag[0][0]=x;       s[x][ny/5].voisins_droi[0][0]=x;
      s[x][ny/5].voisins_diag[0][1]=4*ny/5-1; s[x][ny/5].voisins_droi[0][1]=4*ny/5+1;
      s[x][ny/5].voisins_diag[1][0]=x+1;     s[x][ny/5].voisins_droi[1][0]=x+1;
      s[x][ny/5].voisins_diag[1][1]=4*ny/5;  s[x][ny/5].voisins_droi[1][1]=4*ny/5;
      s[x][ny/5].voisins_diag[2][0]=x;      s[x][ny/5].voisins_droi[2][0]=x;
      s[x][ny/5].voisins_diag[2][1]=4*ny/5+1; s[x][ny/5].voisins_droi[2][1]=4*ny/5-1;
      s[x][ny/5].voisins_diag[3][0]=x-1;     s[x][ny/5].voisins_droi[3][0]=x-1;
      s[x][ny/5].voisins_diag[3][1]=4*ny/5;  s[x][ny/5].voisins_droi[3][1]=4*ny/5;
      s[x][4*ny/5].voisins_diag[0][0]=x;     s[x][4*ny/5].voisins_droi[0][0]=x;
      s[x][4*ny/5].voisins_diag[0][1]=ny/5-1; s[x][4*ny/5].voisins_droi[0][1]=ny/5+1;
      s[x][4*ny/5].voisins_diag[1][0]=x+1;   s[x][4*ny/5].voisins_droi[1][0]=x+1;
      s[x][4*ny/5].voisins_diag[1][1]=ny/5;  s[x][4*ny/5].voisins_droi[1][1]=ny/5;
      s[x][4*ny/5].voisins_diag[2][0]=x;     s[x][4*ny/5].voisins_droi[2][0]=x;
      s[x][4*ny/5].voisins_diag[2][1]=ny/5+1; s[x][4*ny/5].voisins_droi[2][1]=ny/5-1;
      s[x][4*ny/5].voisins_diag[3][0]=x-1;   s[x][4*ny/5].voisins_droi[3][0]=x-1;
      s[x][4*ny/5].voisins_diag[3][1]=ny/5;  s[x][4*ny/5].voisins_droi[3][1]=ny/5;
                      }
}
```

8 Bibliographie

[1] "Surfaces minimales et solutions de problèmes variationnels"
Journée Annuelle du 5 Juin 1993
Harold Rosenberg et David Hoffman
Société Mathématique de France

[2] "Xlib programming manual for version 11, Volume 1"
Adrian Nye
O'Reilly & Associates Inc.

[3] "Mathématiques et surfaces optimales, l'explication des structures naturelles"
S. Hildebrandt et A. Tromba
Bibliothèque Pour la Science, Diffusion Belin

[4] "A survey of minimal surfaces"
Robert Osserman
Dover Books

[5] "Soap Bubbles. Their colors and forces which mold them."
C.V. Boys
Dover Books

Table des matières