République Algérienne Démocratique et Populaire.
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique.

Université Saad Dahlab, Blida
USDB.

Faculté des sciences.
Département de génie informatique.

Mémoire pour l'obtention
d'un diplôme d'ingénieur d'état en informatique.
Option : Intelligence Artificielle (IA)

Sujet :

Simulation et Vérification de l'Opération d'Ebauchage des Surfaces Gauches sur des Fraiseuses à Commande Numérique à 3 Axes

Présenté par : AGEUNINI Brahim
MESSAOUDI Hamza

Promoteur : BEY Mohamed
Encadreur : ACHELI Abdel Hakim

Organisme d'accueil : CDTA (Centre de Développement des Technologies Avancées).
Division productique et robotique

Soutenue le: 20/09/2006, devant le jury composé de :

BENSTITI Souad
SOUAMI Feryel,
MAZARI Rida,

- 2005/2006 -

REMERCIEMENT

Nous tenons à remercier avant tout le **Dieu** qui nous à aidé à réaliser ce modeste travail et un second remerciement à Mr le directeur du centre de développement des technologies avancé, qui nous a accepter au sein de son établissement.

Nos remerciant aussi notre promoteur **M^r^ BEY Mohamed** pour avoir bien voulu nous proposer ce sujet et pour son patience et son soutient continue durant notre stage, et nous remercions aussi **M^r^ ACHELI Hakim**.

إهـــــداء

بعون الله و رعايته أنجزت هذا العمل الّذي سيمنحني شهادة المهندس و الّذي بدوري أفتخر به كثيرا، لأنّه يعتبر باكورة عام كامل من البحث و النشاط و التنقّل بين الجامعة و مركز تطوير التكنولوجيا المتقدّمة، و الأكثر من ذالك لأنه ثمرة السنين التي قضيتها أتعلّم بداية بتلميذ ثم طالبا في مختلف الأطوار الدراسية الإبتدائيّة، المتوسّطة، الثانويّة و أخيرا و ليس أخيرا الجامعيّة و الّذي أتمنى أيضا أن يكون محصّلة علمية و مرجعا لكل باحث متعلّما حتى تعمّ الفائدة و تدوم، كما أرجو من الله عز و جلّ أن أكون قد وفّقت في عملي هذا و أحسنت تحصيله و تنظيمه ومن بعد ذالك تطبيقه فمن إجتهد و أصاب فله أجران فإن له يصب فله أجرواحد، فمزيدا من النجاح والتوفيق لما هو آت بعد الّذي كان و الله المستعان على أمري.

فإلى والدايا العزيزين اللّذين وصلت بفضلهما بعد فضل الله إلى ما أنا عليه اليوم برعايتهم لي و تعبيدهما الطريق أمامي طوال سنوات حياتي حتى لا يشغلني شيء عن دراستي.

إلى جميع أخواتي و إخوتي في العائلةو اللّذين طالما شجّعوني مستفيدا من دعمهم الدائم و خبرتهم في الحياة.

إلى الأطفال الصغار أبناء أختي: ياسر، يسرى والمولودة الجديدة لينة الّتي أتمنّى لها مديد العمر و حياة ملؤها الأزهار و الورود، و إلى إبن أخي: توفيق و إلى أمّه بصيّة، متمنيا لهم السّعادة و الهناء.

إلى صديقي العزيز و شريكي في هذا العمل حمزة الذي أسعدني كثيرا العمل معه و إيجابية الشديد بتواضعه إلى جانب عمله الدؤوب متمنيا له مستقبلا مهنيّا ناجحا كما نجح في دراسته.

إلى كل أصدقائي اللّذين عرفتهم و ياما أكثرهم بداية بكمال، عبد الرحمان، رمزي، صالح، ثم إسماعيل، علاء، هاني، العربي، عمر، عبد الرزاق، محمد والأخرون مع شكر خاصّ الى عبد الغني، معين، عبد الحق اللّذين طالما رافقوني في مسعاي.

الى أساتذتي الكرام دون تمييز و هم اللّذين تدرّجت بين أيديهم منذ السنة الأولى إبتدائي الى يومنا هذا حاملا المشعال من عندهم.

كما أتقدّم بشكر خاص الى من منحني هذا العمل و جعله بين يدي حتى رأى النّور، إلى الأخ و الصديق و الأستاذ باي محمد اللّذي، لم يتوانى ولو مرّة عن تقديم المساعدة و النصيحة، متمنيا من الله عز و جل، أن يمنحه أجره كاملا غير منقوص، على ما قدّمه لنا.

إبـــــراهيـــم

إهداء

إلى من حملتني ورعتني وقاسمتني متاعب الحياة و التي لم اهدي أحد أكثر منها قرة عيني وبلسم روحي:أمي الحنون.

إلى من عانى من اجل تعليمي ودعمني بكل ما يملك من عطف و حنان إلى أن وصلت إلى نهاية المطاف: أبي العزيز.

إلى اعز أختين إلى قلبي اللتين غمرتاني بكل الدعم و المحبة الكافية إلى نورة و ابنتيها أسماء و منال و إلى زوجها و إلى أختي الصغيرة كنزة اروع اخة في الدنيا.

إلى من أمدوني دائما بالثقة.إلى أفضل إخوة في الوجود: بوجمعة، عبد الحليم، عبد القادر، بشير، محمد.

إلى اعز الأصدقاء: علال، توفيق، لمين، حمزة، أمين، رشيد، بلال، محمد.

إلى كل زملاء وزميلات الدراسة: سمير، أسماء، إبراهيم، فاطمة الزهراء، عبد الغاني، أمينة، عبد الحق، رشيدة، مصطفى، أمال، عبد المعين، عمر، منير.

إلى كل أفراد عائلتي و كل من لم يسعني ذكرهم، إلى كل هؤلاء اهدي ثمرة جهدي المتواضع.

حمزة.

Résumé

Ce travail s'insère dans le cadre de développement d'outils de conception et de fabrication des surfaces gauches initié par l'équipe Conception et Fabrication Assistées par Ordinateur (CFAO) au niveau de la Division Robotique et Productique du Centre de Développement des Technologies Avancées (CDTA).

Dans ce projet nous nous intéressons à l'usinage des surfaces de formes libres sur des fraiseuses à commande numérique à 3 axes. Le but de ce travail est le développement d'une application logicielle graphique et interactive sous Windows qui permet : la simulation de l'opération d'ébauchage des surfaces gauche à temps réel, la vérification de tolérance exigé par l'utilisateur, l'adaptation des vitesses d'avance de l'outil en fonction de volume enlevé, paramètres d'outil et de la pièce à usinée, la correction de la trajectoire d'usinage en éliminant les positions d'interférences et en prenant en compte les vitesses adaptées, et finalement l'automatisation de la génération de programme d'usinage «G-Code» à partir des modèles CAO des surfaces à usiner et le trajectoire corrigée.

Summary

This work fits in the setting of development of tools of conception and left surface manufacture initiated by the team Conception and Fabrication Attended by Computer (CFAO) at the level of the Division Robotics and Production of the Centre of Development of Advanced Technology (CDTA).

In this project we are interested in the machining of the free shape surfaces on the numeric tools machine at 3 axes. The goal of this work is the development of a graphic and interactive software application under Windows which allows: the simulation of the left operation of roughing of the surfaces at real time, the checking of tolerance required by the user, the adaptation of the advanced speeds of the tool according to removed volume and tool parameters and the parameters of the machined part, correction of the trajectory of machining by eliminating the positions of interferences and by taking and finally the automation of the generation of the machining programs "G-Code" starting from models CAD of surfaces to be machined and the corrected trajectory.

ملخص

إنّ هذا العمل يندرج في إطار تطوير وسيلة, قادرة على تصميم و تصنيع الأسطح ذات الأشكال الحرّة و التي يقوم بتطويرها فرقة البحث الخاصّة بالتّصميم و التّصنيع عن طريق الكمبيوترعلى مستوى فرع التصنيع الآلي الخاص بمركز تطوير التكنولوجية المتقدّمة.

في هذا المشروع, نهتم بتصنيع المساحات الحرّة بإستعمال آلة التقطيع الرقمية, ذات الثلاث محاور. إنّ الهدف من هذا العمل هو تطوير برنامج آلي تصويري قابل للإستعمال على نظام التشغيل ويندوز و الذي يسمح بما يلي: محاكات عملية التقطيع الآلي, تحديد إن كان هناك مراعات للإرتياب المحدّد من طرف المستعمل, ملاءمة سرعة تحرك القاطعة وذلك حسب حجم المادّة المقتطعة و الخصائص الفيزيائيّة و الميكانيكيّة للقاطعة و للقطعة الأوليّة, تصحيح مسار التّصنيع وذلك بحذف المواضع التي قد تسبّب تداخلا من مسار التصنيع وأخيرا جعل عملية كتابة برنامج التّصنيع آلية إنطلاقا من السّطح المصمّم و المسار المصحّح.

SOMMAIRE

CHAPITRE 3 : USINAGE DES SURFACES GAUCHES

CHAPITRE 4 : METHODE DE SIMULATION DE L'USINAGE

CHAPITRE 5 : CONCEPTION ET IMPLEMENTATION

CHAPITRE 6 : PRESENTATION DE L'APPLICATION

CHAPITRE 7 : TEST ET VALIDATION

INTRODUCTION GENERALE

I. SITUATION DU PROBLEME

Aujourd'hui, l'utilisation de l'outil informatique devient de plus en plus très importante dans la résolution des problèmes de nature complexe car il offre l'efficacité, la fiabilité et la rapidité dans le traitement. Parmi les domaines qui exigent l'utilisation de cet outil, on trouve le domaine de l'industrie, tel que l'industrie automobile, l'industrie aéronautique, l'industrie mécanique, ...etc. Plusieurs industries ont été contraintes à augmenter la quantité et améliorer la qualité de leurs produits. Particulièrement, l'industrie d'outillage mécanique cherche à réduire le temps et le coût de fabrication tout en améliorant la quantité et la qualité de produit et les conditions de travail en offrant aux utilisateurs des ressources intégrées réduisant les tâches itératives et coûteuses.

Pour cela, l'industrie d'outillage mécanique s'appuie sur la puissance de la CFAO (Conception et Fabrication Assistées par Ordinateur), qui apporte la flexibilité et la souplesse dans la conception et la fabrication des pièces de complexités diverses comme l'usinage des pièces de formes gauches. Le processus de réalisation de ces formes est un processus industriel qui a subi de grandes évolutions ces dernières décennies par l'introduction de la CFAO et de la commande numérique.

Les pièces de formes libres (moules, matrices,...etc.) sont très rencontrées dans notre vie quotidienne. Ces pièces sont conçues dans le but d'assurer des fonctions inscrites dans le cahier des charges. Ces pièces ne peuvent être usinées que sur des fraiseuses à commande numérique à 03 ou à 05 axes en raison des géométries très complexes. L'usinage de ces surfaces passe généralement par trois étapes : ébauche qui permet d'enlever le maximum de matière, demi finition où on s'approche de la forme finale et finition où on obtient la forme voulue. Toutes ces étapes nécessitent la génération d'un ensemble d'instructions écrites dans un langage propre à la machine appelé programme « G-Code ».

Le travail que nous présentons dans ce mémoire s'inscrit dans le cadre de développement d'outils de conception et de fabrication des surfaces de formes libres (surfaces gauches) initié par l'équipe CFAO de la Division Productique et Robotique du Centre de Développement des Technologies Avancées (CDTA).

Notre projet est une continuité des travaux précédents traitant :

- La modélisation et la conception des courbes et des surfaces B-Spline et NURBS;
- La reconstruction des courbes et des surfaces B-Spline et NURBS par interpolation et par approximation à partir d'un nuage de points;
- Usinage des surfaces gauches par les courbes isoparamétriques en utilisant les six stratégies d'usinage (One-Way, Zig-Zag, Concentrique, Spiral-In, Spiral-Out et Radiale).
- Usinage des surfaces gauches par la méthode des plans parallèles.
- Ebauchage des surfaces gauches.

On peut synthétiser l'objectif global de notre travail de la façon suivante :

« Simulation et vérification de l'opération d'ébauchage des surfaces gauches sur des fraiseuses à commande numérique à 3 axes »

Les programmes d'usinage "G-Code" des surfaces gauches en ébauche contiennent un nombre très important de lignes d'instructions, et par conséquent, la vérification manuelle de ces programmes devient impossible. Les surfaces usinées contiennent toujours des erreurs qui peuvent être soit l'existence des zones non usinées soit d'interférence soit de collision. La simulation de l'opération d'usinage est un outil permettant de réduire les temps d'usinage effectif et de valider les trajets d'usinage en détectant les différentes erreurs et leurs positions afin de les corriger. Un autre problème qu'il faut résoudre c'est le choix des conditions de coupe et en particulier les vitesses d'avance qui affecte directement les temps d'usinage.

II. OBJECTIF DU TRAVAIL

Dans ce projet, nous nous s'intéresserons à la simulation et à la vérification virtuelle de l'opération d'ébauchage des surfaces de formes libres usinées sur des fraiseuses à commande numérique à 03 axes ainsi qu'à la correction du trajet d'usinage et l'adaptation des vitesses d'avance en fonction de la quantité de matière enlevée.

Le but de ce travail est le développement d'une application logicielle graphique et interactive sous Windows qui permet, à partir des différentes positions d'outil constituant la trajectoire d'usinage, de simuler et de vérifier virtuellement l'opération d'usinage en ébauche des surfaces gauches et de détecter automatiquement les zones en dehors de la tolérance d'usinage exigée et à apporter les corrections nécessaires au trajet d'usinage et par la suite faire une adaptation des vitesses d'avance.

III. DESCRIPTION DU TRAVAIL

Le présent mémoire est composé de sept chapitres :

Le premier chapitre est consacré à l'étude des différentes méthodes de conception des surfaces pour la représentation des formes a usinées.

Dans le deuxième chapitre, nous étudions les machines outil à commande numérique (MOCN) utilisées pour réaliser l'opération de fraisage.

Dans le troisième chapitre, nous allons présenter les opérations d'usinage et en particulier l'opération d'ébauchage.

Dans le quatrième chapitre, nous développons une étude sur les méthodes de simulation de l'usinage.

Dans le cinquième chapitre, nous allons présenter notre conception et implémentation.

La présentation de l'application logicielle développée est faite dans le chapitre six

Le chapitre 7 est réservé à l'étape de test et de validation du travail réalisé.

Enfin, nous terminons ce mémoire par une conclusion générale et quelques recommandations pour la continuité de ce travail.

Méthodes de conception des surfaces

I. INTRODUCTION :

Les surfaces de formes libres (surfaces gauches) sont très utilisées dans le vécu industriel pour la conception et la fabrication des moules, matrices, formes aérodynamiques et des formes esthétiques. Pour cela, nous avons besoins des outils et des méthodes permettant la conception, l'édition et la modification de ces surfaces d'une manière interactive et en temps réel afin d'obtenir la forme voulue.

Dans ce chapitre, nous allons étudier les différentes méthodes utilisées dans la conception des surfaces gauches et en particuliers les surfaces paramétriques.

II. METHODES DE REPRESENTATION DES SURFACE EN CAO :

On distingue deux méthodes de représentation des surfaces :

II.1 Surfaces non paramétriques [1] :

Ces surfaces peuvent être représentées sous deux différentes formes :

II.1.1 Forme explicite : dans ce cas, la surface est donnée par l'équation suivante :

$$z = S(x, y) \tag{1}$$

Où pour chaque couple (x, y) lui correspond une seule valeur de z. Puisque des valeurs multiples ne sont pas permises, cette surface ne peut pas être fermée.

II.1.2 Forme implicite : dans ce cas, la surface est donnée par l'équation suivante :

$$S(x, y, z) = 0 \tag{2}$$

La forme implicite n'a pas la limitation de la représentation explicite, mais la manipulation interactive des surfaces est difficile.

II.2. Surfaces paramétriques [1] :

Les surfaces paramétriques sont très utilisées dans les systèmes de modélisation des surfaces. Une surface paramétrique est définie par un ensemble de trois fonctions, et chacune d'elle dépend de deux paramètres u et v (voir figure 1). Cette surface est donnée par :

$$S(u, v) = (x(u, v), y(u, v), z(u, v)) \qquad \text{Avec } u, v \in [0, 1] \times [0,1] \tag{3}$$

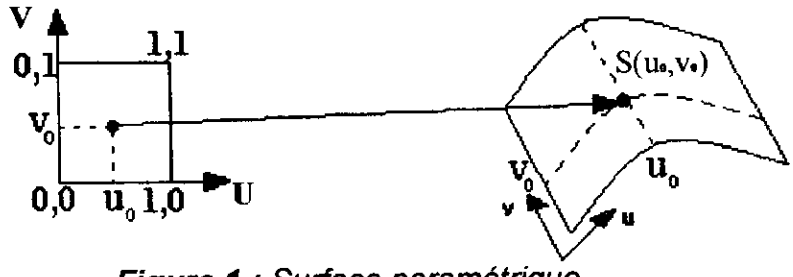


Figure 1 : Surface paramétrique.

II.3. Propriétés des surfaces paramétriques [1] :

Dans la phase d'usinage des surfaces gauches, nous avons besoins de calculer les propriétés géométriques intrinsèques en différents points de ces surfaces.

II.3.1. Vecteurs tangents et vecteur normal à la surface paramétrique [1] :

Les vecteurs tangents $\vec{T}_u$ et $\vec{T}_v$ à la surface paramétrique au point (u, v) sont donnés respectivement par :

$$\vec{T}_u = S'_u = \frac{\partial S}{\partial u} = \left(\frac{\partial x}{\partial u}, \frac{\partial y}{\partial u}, \frac{\partial z}{\partial u}\right) \quad (4)$$

$$\vec{T}_v = S'_v = \frac{\partial S}{\partial v} = \left(\frac{\partial x}{\partial v}, \frac{\partial y}{\partial v}, \frac{\partial z}{\partial v}\right) \quad (5)$$

Le premier vecteur est le vecteur tangent dans la direction u, tandis que le second vecteur est le vecteur tangent dans la direction v. Ces deux vecteurs définissent le plan tangent à la surface en un point (voir *figure 2*).

Le vecteur normal unitaire $\vec{n}$ à la surface en un point (voir *figure2*) est donné par :

$$\vec{n} = \frac{\frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v}}{\left|\frac{\partial S}{\partial u} \times \frac{\partial S}{\partial v}\right|} \quad (6)$$

II.3.2. Courbes isoparamétriques :

Une surface paramétrique peut être considérée comme l'union d'un nombre infini de courbes appelées courbes isoparamétriques. Quand on fixe le paramètre u (*resp. v*) et on fait varier le paramètre v (*resp. u*) on obtient une courbe isoparamétrique (voir *figure 2*).

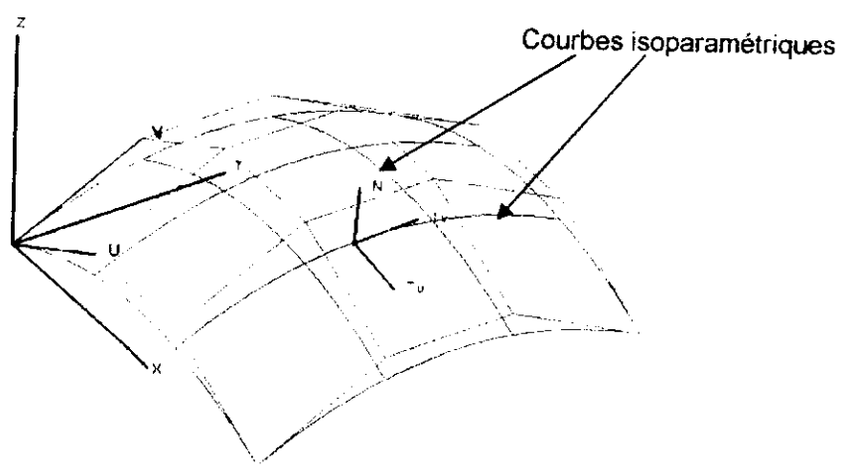


***Figure 2** : Vecteurs tangents et vecteur normal.*

II.3.3. Courbures d'une surface [1] :

II.3.3.1. Notion de courbure [1] :

L'inverse du rayon du cercle R qui approxime le mieux la courbe en un point précis est appelé la courbure K et elle est donnée par la formule suivante (voir figure 3) :

$$k = \frac{1}{R} \tag{7}$$

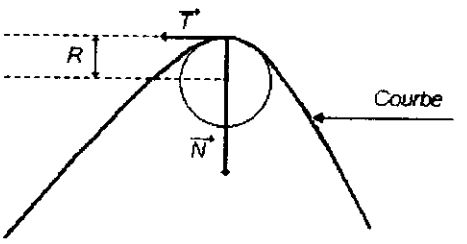


***Figure 3** : Courbure d'une courbe en un point.*

Pour chaque point de la surface paramétrique, il existe un nombre indéterminé de courbes qui appartiennent à cette surface et passent par ce point. Par conséquent, on aura un nombre infini de courbures associées à chaque courbe. La plus petite valeur est appelée courbure minimale K_1 et la plus grande valeur est appelée courbure maximale K_2. Ces courbures sont appelées courbures principales de la surface. Ces courbures permettant d'analyser la surface.

II.3.3.2. Courbures principales [1] :

Les courbures principales K_1 et K_2 en un point d'une surface sont calculées par la résolution de l'équation du second degré suivante :

$$(LM - M^2) * K^2 + (2MF - GL - EN) * K + EG - F^2 = 0 \tag{8}$$

Où *L, E, N, G, M* et *F* sont donnés par les relations suivantes :

$$E = \frac{\partial P}{\partial u} \times \frac{\partial P}{\partial u} \tag{9}$$

$$F = \frac{\partial P}{\partial u} \times \frac{\partial P}{\partial v} \tag{10}$$

$$G = \frac{\partial P}{\partial v} \times \frac{\partial P}{\partial v} \tag{11}$$

$$L = n \times \frac{\partial^2 P}{\partial u^2} \tag{12}$$

$$M = n \times \frac{\partial^2 P}{\partial u \partial v} \tag{13}$$

$$N = n \times \frac{\partial^2 P}{\partial v^2} \tag{14}$$

A partir des courbures principales, la courbure moyenne H et la courbure gaussienne K sont données par :

$$H = \frac{K_1 + K_2}{2} \quad (15)$$

$$K = K_1 \times K_2 \quad (16)$$

En un point *x* de la surface, et en fonction des valeurs des deux courbures (H, K) la surface aura les propriétés suivantes :

- Si $K>0$ alors *x* est un point elliptique (concave ou convexe);
- Si $K<0$ alors *x* est un point hyperbolique (selle de cheval) ;
- Si $K=0$ alors *x* est un point parabolique (développable);
- Si $K=0$ et $H=0$ alors *x* est un point plat.

III. METHODES DE CONCEPTION DES SURFACES [3, 5, 7, 8] :

En CAO (Conception Assistée par Ordinateur), deux classes de méthodes sont utilisées dans la conception des surfaces.

III .1. Méthodes basées sur les courbes :

Pour ces méthodes, la conception de la surface passe par la conception de quelques courbes clés. Cette méthode est utilisée dans la conception des surfaces suivantes :

- Surface balayée;
- Surface de Coons;
- Surface de Gordon;
- Surface de révolution;
- Surface extrudée;
- Surface lissée;
- Surface réglée.

III .2. Méthodes basées sur les points :

Pour ces méthodes, l'information élémentaire est le point. Cette méthode est utilisée dans la conception des surfaces suivantes :

- Interpolation d'un nuage de point (globale ou locale);
- Approximation d'un nuage de point (globale ou locale);
- Surfaces de Bézier;
- Surfaces de Bézier Rationnelle;
- Surfaces B-Spline;
- Surfaces NURBS (Non-Uniform Rational B-Spline).

Dans notre travail, nous allons considérés les surfaces les plus utilisées en CAO à savoir les NURBS et les B-Spline.

IV. SURFACE B-SPLINE [2, 3, 4] :

IV.1. Définition d'une surface B-Spline :

Une surface B-Spline est définie par les informations suivantes (voir figure 4) :

- Un réseau de $(m+1)\times(n+1)$ points de contrôle $p_{i,j}$, où $0 \le i \le m$ et $0 \le j \le n$,
- Un vecteur des nœuds de $h+1$ nœuds dans la direction u, $U = \{0 = u_0, u_1, \ldots, u_h = 1\}$,
- Un vecteur des nœuds de $k+1$ nœuds dans la direction v, $V = \{0 = v_0, v_1, \ldots, v_k = 1\}$,
- Le degré p de la surface dans la direction u,
- Le degré q de la surface dans la direction v.

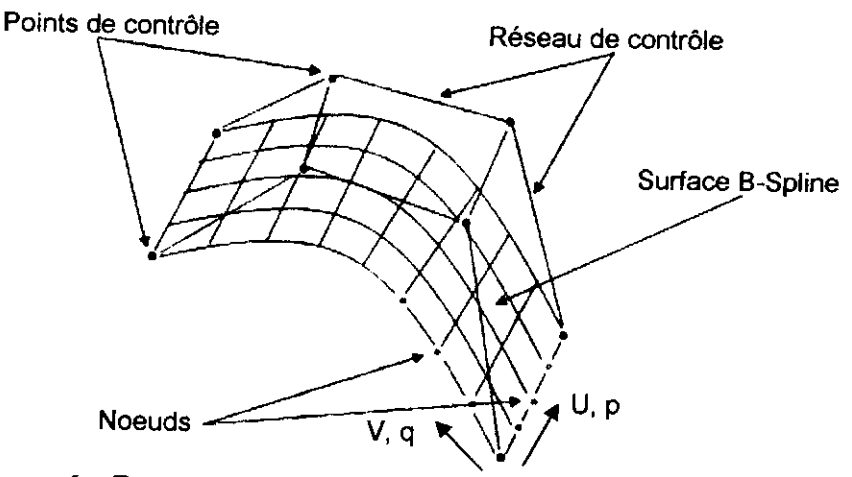


***Figure 4** : Paramètres de définition d'une surface B-Spline.*

La surface B-Spline est donnée par l'équation suivante [2, 4] :

$$S(u,v) = \sum_{i=0}^{m}\sum_{j=0}^{n} N_{i,p}(u) N_{j,q}(v) p_{i,j} \qquad (17)$$

Où $N_{i,p}(u)$ et $N_{j,q}(v)$ sont les fonctions base B-Spline de degré p et q respectivement données par :

$$N_{i,0}(u) = \begin{cases} 1 & \text{si} \quad u_i \le u < u_{i+1} \\ 0 & \text{sinon} \end{cases} \qquad (18)$$

$$N_{i,p}(u) = \frac{u - u_i}{u_{i+p} - u_i} N_{i,p-1}(u) + \frac{u_{i+p+1} - u}{u_{i+p+1} - u_{i+1}} N_{i+1,p-1}(u) \qquad (19)$$

$$N_{j,0}(v) = \begin{cases} 1 & si \quad v_j \le v < v_{j+1} \\ 0 & \sin on \end{cases} \qquad (20)$$

$$N_{j,q}(v) = \frac{v - v_j}{v_{j+p} - v_j} N_{j,q-1}(v) + \frac{v_{j+q+1} - v}{v_{j+q+1} - v_{j+1}} N_{j+1,q-1}(v) \qquad (21)$$

Puisque $N_{i,p}(u)$ et $N_{j,q}(v)$ sont des fonctions de degré p et degré q, alors on dit que la surface B-Spline est de degré (p,q).

Les fonctions base d'une surface B-Spline sont les coefficients des points de contrôle. La fonction base du point de contrôle $p_{i,j}$ est le produit de deux fonctions base B-Splines unidimensionnelles $N_{i,p}(u)$ dans la direction u et $N_{j,q}(v)$ dans la direction v. Tous ces produits sont des fonctions B-Splines bidimensionnelles.

IV.2. Forme des surfaces B-Spline :

Une surface B-Spline de degré (p, q) de $h+1$ nœuds dans la direction u et $k+1$ nœuds dans la direction v, peut se présenter sous trois formes possibles dans chaque direction (pincée, ouverte ou fermée).

- **Surfaces B-Spline pincées :** si la surface B-Spline est pincée dans les deux directions, alors cette surface passe par les points de contrôle $p_{0,0}$, $p_{m,0}$, $p_{0,n}$ et $p_{m,n}$ et elle est tangente aux huit segments du réseau de contrôle en ces points.

- **Surfaces B-Spline fermées :** si la surface B-Spline est fermée dans une direction, alors toutes les courbes isoparamétriques dans cette direction sont fermées.

- **Surfaces B-Spline ouvertes :** si la surface B-Spline est ouverte dans les deux directions, alors la surface ne passe pas par les points de contrôle $p_{0,0}$, $p_{m,0}$, $p_{0,n}$ et $p_{m,n}$.

IV.3. Importantes propriétés des surfaces B-Spline :

Les importantes propriétés des surfaces B-Spline sont les suivantes :

- La représentation des formes en surfaces B-Spline est définie par morceaux; ou chaque morceau est une portion de la surface de degré (p, q) ; l'union de toutes ces portions forme la surface B-Spline.
- $B_{i,p}(u)$ (respectivement $B_{j,q}(v)$) est strictement positive pour i,p (respectivement (j, q)), avec v et u appartenants à l'intervalle [0,1].
- La somme de toutes les fonctions base B-Spline est égale à '1'.
- Les surfaces de Bézier sont des cas particuliers des surfaces B-Spline sous certaines conditions.
- La surface B-Spline doit satisfaire les égalités fondamentales suivantes :

$$\begin{cases} h = m + p + 1 \\ k = n + q + 1 \end{cases} \qquad (22)$$

- La surface B-Spline vérifie la propriété de l'enveloppe convexe. Toute la surface est contenue dans son réseau de contrôle.
- Schéma de modification locale; si la position d'un point de contrôle change, cela n'affecte que localement la surface.
- La surface B-Spline n'est pas aussi complexe que son réseau de contrôle.

- Variance affine ; lorsqu'une surface B-Spline subie une transformation affine, on peut utiliser les images des points de contrôle pour construire la surface.
- Les courbes S(0,v), S(1,v), S(0,u) et S(1,u) sont les courbes limites.
- La surface est indépendante du système de coordonnées.

L'inconvénient des surfaces B-Spline, c'est l'impossibilité de représenter les surfaces coniques (sphère, ellipsoïde, ...etc.) puisque ces surfaces utilisent des fonctions polynomiales, d'où la nécessité de passer aux surfaces qui utilisent des fonctions rationnelles et qui sont les surfaces NURBS.

V. SURFACE NURBS :

Une surface NURBS est définie par les informations suivantes (voir figure 5) :

- Un réseau de $(m+1)\times(n+1)$ points de contrôle $p_{i,j}$, où $0 \le i \le m$ et $0 \le j \le n$,
- Pour chaque point de contrôle est associé un poids $w_{i,j} \ge 0$,
- Un vecteur des nœuds de $h+1$ nœuds dans la direction u, $U = \{0 = u_0, u_1, \ldots, u_h = 1\}$,
- Un vecteur des nœuds de $k+1$ nœuds dans la direction v, $V = \{0 = v_0, v_1, \ldots, v_k = 1\}$,
- Le degré p de la surface dans la direction u,
- Le degré q de la surface dans la direction v.

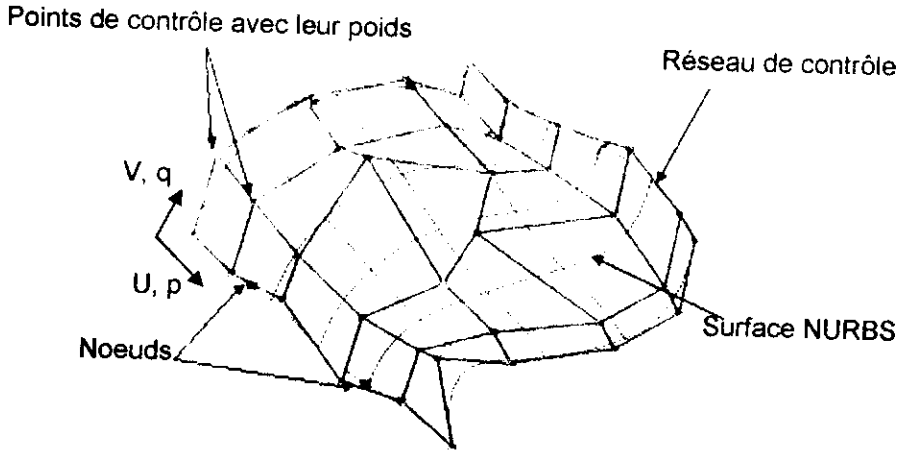


Figure 5 : *Paramètres de définition, d'une surface NURBS.*

La surface NURBS est donnée par [4, 6, 7] :

$$S(u,v) = \frac{\sum_{i=0}^{m}\sum_{j=0}^{n} N_{i,p}(u)N_{j,q}(v)w_{i,j}p_{i,j}}{\sum_{i=0}^{m}\sum_{j=0}^{n} N_{i,p}(u)N_{j,q}(v)w_{i,j}} \qquad (23)$$

Où $N_{i,p}(u)$ et $N_{j,q}(v)$ sont les fonctions base B-Splines de degré p et q respectivement. Les identités fondamentales (une pour chaque direction) doivent être satisfaites :

$$\begin{cases} h = m + p + 1 \\ k = n + q + 1 \end{cases} \quad (24)$$

Puisque $N_{i,p}(u)$ et $N_{j,q}(v)$ sont des fonctions de degré p et degré q, alors on dit que la surface NURBS est de degré (p,q).

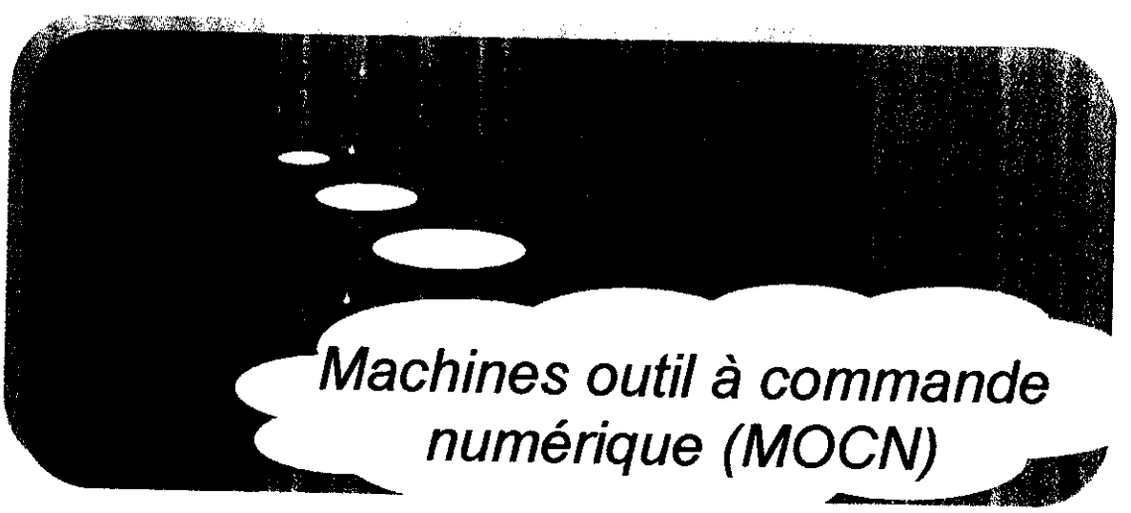
V.1. Importantes propriétés d'une surface NURBS :

En plus des propriétés des surfaces B-Spline, les surfaces NURBS possèdent les propriétés suivantes :

- Une courbe isoparamétrique sur la surface NURBS est une courbe NURBS.
- Si tout les poids sont égaux, la surface NURBS se transformera en une surface B-Spline.
- Une surface NURBS est une surface rationnelle, donc elle permet de représenter toutes les formes coniques (sphère, ellipsoïde, ...etc.).
- Si le poids d'un point de contrôle augmente, une portion de la surface est attirée vers ce point. Si le poids d'un point de contrôle diminue, une portion de la surface est repoussée de ce point.
- Invariance projective : si une transformation projective est appliquée à une surface NURBS, alors la surface projetée est construite à partir de la projection des points de contrôle.

VI. CONCLUSION :

Dans ce chapitre nous avons vu les différentes méthodes de conception des surfaces gauches. Comme nous avons étudié en particulier les surfaces les plus utiliser dans les logiciels de CFAO à savoir les B-Spline et les NURBS ainsi que leurs importantes propriétés. Dans la suite, nous allons étudier les machines outils à commande numérique permettant d'usiner ces surfaces.



Machines outil à commande numérique (MOCN)

I. INTRODUCTION :

Une fois les surfaces gauches sont conçues, il est nécessaire de choisir les machines à utiliser ainsi que l'outillage nécessaire. Généralement, ces surfaces sont usinées sur des fraiseuses. Mais en raison de la géométrie complexe de ces surfaces, ces fraiseuses doivent être commandée numériquement. D'où la nécessité de comprendre le mode de fonctionnement de ces machines.

Dans ce chapitre, nous allons présenter les différents composants d'une machine outil à commande numérique et en particulier la fraiseuse ainsi que la structure générale des programmes d'usinage.

II. MACHINE OUTIL A COMMANDE NUMERIQUE (MOCN) [9,10] :

Une machine outil d'usinage a pour but de réaliser physiquement les mouvements de coupe nécessaires à l'obtention d'une surface par enlèvement de matière (le cas des tours, des fraiseuses, des perceuses, ...etc.) par l'utilisation d'outils adéquats.

Une machine outil à commande numérique (MOCN) c'est une machine outil programmable équipée d'une commande numérique par calculateur (CNC). Une MOCN assure la réalisation automatisée des pièces et les mouvements nécessaires sont décrits dans un programme. Ces machines sont composées essentiellement de deux parties complémentaires :

- Partie opérative : qui est composée des organes agissant directement sur le produit final.
- Partie commande : qui contrôle la machine en envoyant des consignes et récupère des informations sur le déroulement des opérations.

Si la machine à commande numérique est équipée d'un magasin d'outils et d'un changeur automatique d'outils, alors est appelée un centre d'usinage [11]. On distingue les centres de tournage, les centres de fraisage et les centres de perçage. Souvent et par abus de langage, on appelle centre d'usinage, les centres de fraisage.

III. ETUDE ORGANIQUE D'UNE MOCN [10] :

La machine outil doit avoir la structure suivante :

- Les axes de la machine : qui assurent la mise en position de l'outil par rapport à la pièce et les mouvements d'avance.
- La broche : qui assure de mouvement de coupe donnée à l'outil ou à la pièce en fonction de la machine.
- Un système de contrôle – commande : qui permet le bon fonctionnement des axes et des fonctions auxiliaires.
- Le bâti : qui assure le lien entre ces systèmes.

IV. CLASSIFICATION DES MACHINE OUTILS [10] :

Le choix de la machine dépend de la forme de la pièce et de l'opération d'usinage. Traditionnellement, les machines sont classées en fonction des formes de surface à réaliser. Les machines employées dans les ateliers de construction de machines comprennent :

- Tours,
- Perceuses,
- Fraiseuses,
- Raboteuses.

Dans les ateliers de mécanique, les machines les plus utilisées dans la fabrication des pièces de formes libres (moules, matrices, ...etc.) sont les fraiseuses.

V. PRESENTATION DES FRAISEUSES [9,12] :

Les fraiseuses sont des machines utilisées pour la réalisation des formes prismatiques et peuvent réaliser des opérations de contournage. L'outil (fraise) est fixé à une broche qui le fait tourner (mouvement de coupe) et peut se déplacer en translation par rapport à la pièce suivant trois directions (mouvement d'avance).

V.1. Avantages des fraiseuses à commande numérique [9] :

Les principaux avantages des fraiseuses à commande numérique sont les suivants :

- Outil fiable et précis,
- Gestion automatique (chargement/déchargement d'outil/pièce à usiner d'une manière automatique),
- Travail continu,
- Réduction du temps d'usinage par rapport au coût de fabrication des pièces en raison de la réponse rapide,
- Souplesse et la polyvalence.

V.2. Constitution d'une MOCN (fraiseuse) [9, 11] :

Les différentes parties d'une fraiseuse sont les suivantes :

- **Bâti :** est la plateforme de la machine qui permet de supporter la chaleur et d'assurer une précision accrue lors de l'usinage.
- **Broche :** elle porte l'outil (fraise) et transmis ainsi le mouvement de rotation (mouvement de coupe) nécessaire à l'opération de fraisage.
- **Porte outil :** il assure la liaison entre l'outil et la broche.
- **Outil d'usinage :** l'outil utilisé dans le fraisage est appelé fraise.
- **Directeur de commande numérique (DCN) :** comme toutes les machines à commande numérique, les fraiseuses disposent d'un calculateur équipé d'une

mémoire intégrée et qui exécute les programmes chargés en mémoire d'une façon autonome.

- **Pupitre de commande :** son rôle primaire est de dialoguer avec le DCN par le biais de messages envoyés appelés commandes et il possède un écran et un clavier.

- **Armoire électronique :** interposée entre le DCN et la machine (coté mécanique), elle englobe tous les composants électroniques (transistor, condensateur, fusible, câblage, ...) et des cartes de gestion.

VI. DEFINITION DES ORIGINES DE LA MACHINE [11,14] :

Le processeur CN calcule tout les déplacements par rapport au point d'origine mesure de la machine. Les différentes origines à considérer sont les suivantes :

- **Origine mesure (OM) :** cette origine dépend de la machine et définit l'origine du système physique de mesure de la machine.

- **Origine machine (Om) :** la prise d'origine se fait sur une position physique précise c'est l'origine machine (Om) qui est déterminée à partir de l'origine mesure.

- **Origine programme (OP) :** pour écrire un programme pièce, le programmeur choisit une origine programme.

- **Origine pièce (Op):** est une origine qui lie la pièce à la machine. L'origine pièce peut être confondue avec l'origine programme.

VII. AXES DE DEPLACEMENT [10,12] :

Les axes de déplacement mettent en mouvement les parties mobiles des machines avec de fortes accélérations. Les axes sont constitués d'un guidage, d'un système d'entraînement, d'une motorisation et d'un système de mesure. Ces constitutions se sont reparties dans deux parties principales (voir figure 1):

- Une partie opérative : qui est essentiellement de nature matériel.
- Une partie commande : constituée elle même de deux parties :
  1. Une partie matérielle : dispositifs électroniques (numérique et analogique).
  2. Une partie logicielle.

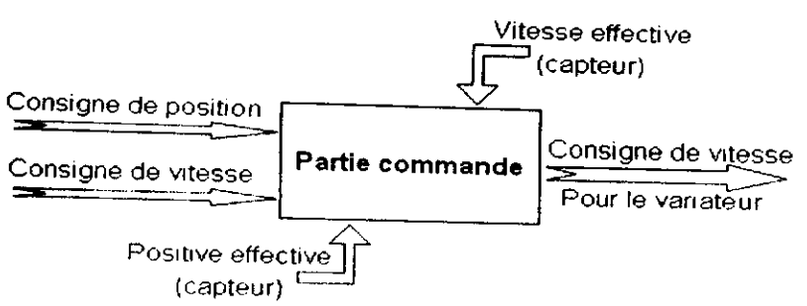


Figure 1: *Schéma fonctionnel (entrée et sortie) d'un axe numérique [13].*

Le guidage des éléments fonctionnels est assuré par des glissières. Le système de mesure transmet la position de l'élément fonctionnel (pièce ou outil) à la commande numérique.

- **Axes commandés [12]:** dans une structure mécanique, on appelle axe commandé une liaison équipée d'une motorisation qui permet à un instant donné de fournir une valeur à la coordonné articulaire.

- **Asservissement d'un axe numérique [12] :** la commande d'axe permet d'asservir en position et en vitesse le déplacement des mobiles. Le schéma général de l'asservissement est donné par la figure 2. il comporte deux boucles d'asservissement imbriqué :

  - Une boucle de régulation de vitesse.
  - Une boucle de régulation de position.

A chaque cycle de calcule, le directeur de commande numérique délivre une consigne de position. L'asservissement a pour fonction de faire suivre au mobile la succession de la position calculée.

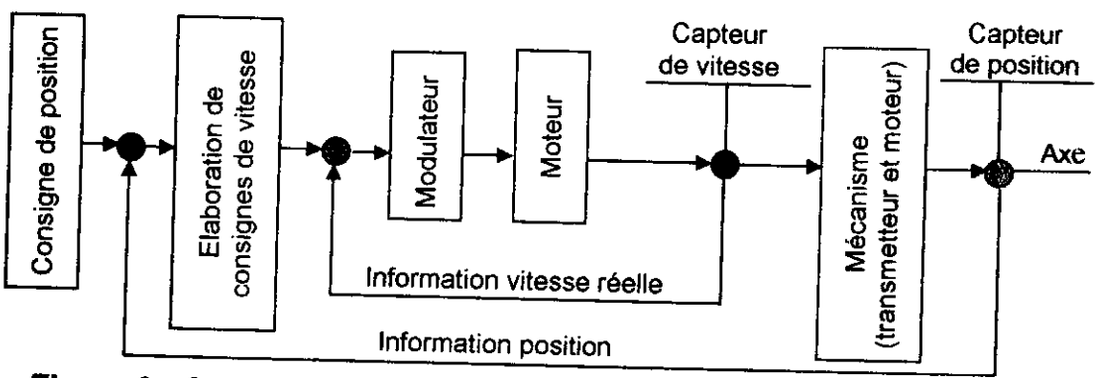


Figure 2 : *Schéma général de l'asservissement d'un axe numérique [13].*

VII.1. Définition normalisée des axes numériques [10,11] :

Un système de coordonnées permet de repérer les positions et les déplacements d'un objet par rapport à un point origine. La norme [NF ISO 841] définit un système de coordonnées et désigne les divers mouvements de manière à ce qu'un programmeur puisse décrire les opérations d'usinage indépendamment de la cinématique de la machine. On considère toujours les mouvements de l'outil par rapport à la pièce tenue pour fixe.

Le système normal de coordonnées est un système cartésien rectangulaire de sens direct, lié à une pièce placée sur la machine, et ayant des arêtes parallèles aux glissières principales de la machine. Il est désigné par les lettres X, Y, Z non munies du signe + ». «Le sens positif du mouvement d'un chariot de la machine est celui qui provoque un accroissement sur la pièce dans la coordonnée correspondante».

- **Axe de mouvement Z :** est l'axe du système normal parallèle à l'axe de la broche principale.

- **Axe de mouvement X :** l'axe X est perpendiculaire à l'axe Z. Pour les fraiseuses :
  - Si l'axe Z est horizontal, le sens X positif est dirigé vers la droite lorsqu'on regarde de la broche principale vers la pièce.
  - Si l'axe Z est vertical, le sens X positif est dirigé vers la droite, lorsqu'on regarde de la broche principale vers le montant de la machine.

- **Axe de mouvement Y :** il forme avec les axes **X** et **Z** un trièdre de sens direct.

La règle des trois doigts de la main droite permet de retrouver facilement l'orientation des axes X, Y et Z (voir figure 3).

- **Mouvements de rotations A, B, C :** les angles **A**, **B** et **C** définissent les mouvements de rotation effectués respectivement autour des axes parallèles à **X**, **Y** et **Z**. L'orientation positive d'un axe rotatif correspond à la rotation d'une vis de pas à droite avançant dans le sens positif de l'axe associé (sens du vissage).

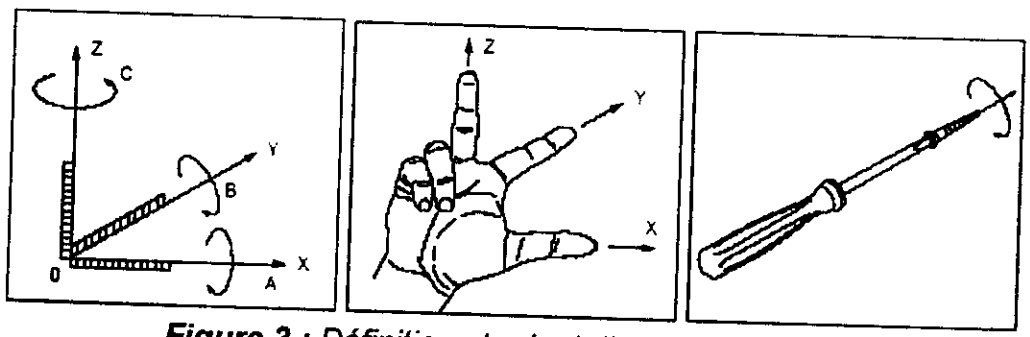


***Figure 3** : Définition et orientation des axes [12].*

Les déplacements suivant les axes X, Y et Z définissent 3 degrés de liberté de translation. Par contre, les rotations autour des axes X, Y et Z définissent 3 degrés de liberté de rotation. Par conséquent, le nombre d'axes d'une machine est le nombre de degrés de liberté autorisés.

Pour les axes de la machine, nous avons deux types d'axes :

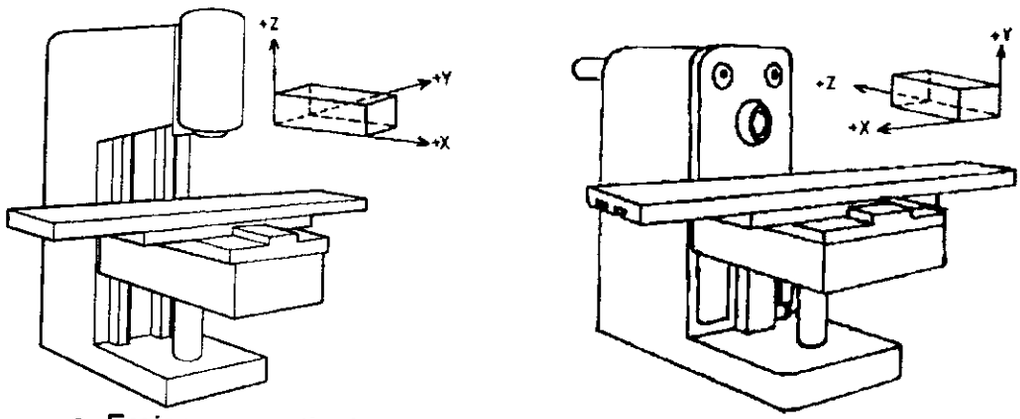
- **Demi axe numérique :** est un axe de déplacement pour lequel un ensemble fini de positions peut être atteint ou un axe de déplacement asservi en position ou en vitesse.
- **Axe numérique :** axe de déplacement pour lequel une infinité de positions peuvent être atteintes à la résolution de positionnement près ou un axe de déplacement asservi en position et en vitesse.

VII.3. Types de fraiseuses [9]:

On distingue deux types de fraiseuses selon la position des axes :

- **Fraiseuse verticale :** elle est appelée ainsi car la broche est verticale (voir figure 4.a).

- **Fraiseuse** horizontale : elle est appelée ainsi car la broche est horizontale (voir figure 4.b).



a. Fraiseuse verticale. **b.** Fraiseuse horizontale.

Figure 4: *Types de fraiseuses et axes associés.*

Pour notre travail, nous allons considérer les fraiseuses verticales à 3 axes.

VIII. PROGRAMMATION DES MOCN [12] :

Comme nous avons indiqués précédemment, les MOCN sont des machines commandées par un programme écrit dans un langage bien défini et stocker dans une mémoire et exécuter par le DCN.

VIII.1. Définition d'un programme [11]:

Un programme est une suite d'instructions écrites dans un langage codé propre à la commande numérique (le plus utilisé est le code ISO : International Standardization Organization) décrivant les opérations d'usinage. La commande numérique interprète le programme pour réaliser l'usinage sur la machine outil. Le programme d'usinage comporte deux types d'informations :

- des ordres de déplacements,
- des ordres auxiliaires.

On appelle « instruction » le plus petit élément d'un programme susceptible de donner lieu à une modification de [13] :

1. l'état physique de la machine commandée ;
2. son état logique.

VIII.2. Ordres programmables :

Dans un programme, on peut donner à la machine deux ordres :

- **Ordres de déplacements [11] :** pour donner un ordre de déplacement, il faut spécifier un mode d'interpolation, une position à atteindre et une vitesse de déplacement. Les interpolations utilisées sont les suivantes :
  - Interpolation linéaire,
  - Interpolation circulaire,

- Interpolation hélicoïdale,
- Interpolation polynomiale.

- **Ordres auxiliaires [11]:** sont des ordres séquentiels qui permettent soit de rendre réalisable, soit d'améliorer la réalisation de l'usinage.

VIII.3. Modes de préparation des programmes d'usinage [13,9]:

Quelque soit le langage de programmation utilisé pour l'écriture des programmes pièces, le seul langage compréhensible par la machine est le langage ISO. Le passage d'un langage de haut niveau au langage ISO est possible en utilisant un logiciel de traduction. La programmation peut être faite de deux manières :

- Manuelle;
- Assistée : pour ce cas, nous pouvons utiliser la méthode conversationnelle ou un logiciel de F.A.O.

VIII.4. Programmation en langage ISO [9]:

On appelle le programme interprété et exécuté par le DCN le « G-Code » employé pour définir les fonctions d'usinage (déplacements, changement d'outil, vitesse d'avance, vitesse de broche, sens de rotation, ...etc.).

Un programme « G-Code » est constitué d'un ensemble de lignes appelées « Bloc ». Ce dernier est un ensemble d'instructions relatives à une séquence d'usinage. Un Bloc est constitué d'un ensemble de « Mots » qui est aussi composé d'un ensemble de caractères et de chiffres constituant une information (voir figure 5).

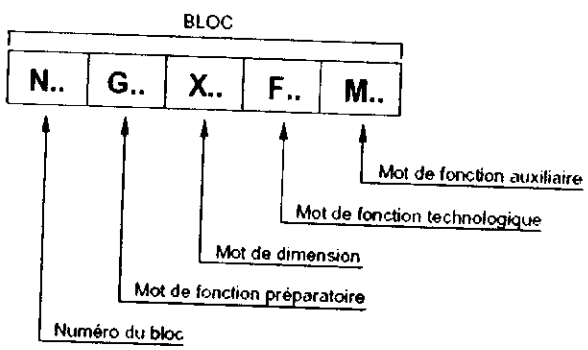


Figure 5: *Format général des blocs [12].*

VIII.4.1. Format de mot [10,9]:

Le mot définit une instruction ou donnée à transmettre au système de commande (voir figure 6). Il existe deux Types de mots :

- mots définissant des dimensions,
- mots définissant des fonctions : les plus usuelles sont les suivantes :

  G.. : Fonctions préparatoires.

  F.. : Fonctions vitesse d'avance (**F**eedrate = avance).

  S.. : Fonctions vitesse de broche (**S**peed = vitesse).

T.. : Fonctions outils (Tools = outils).

M.. : Fonctions auxiliaires (Miscellaneous = divers).

X, Y, Z.. : Translation suivant les axes X, Y et Z respectivement.

A, B, C.. : Angles de rotation autour des axes X, Y et Z respectivement.

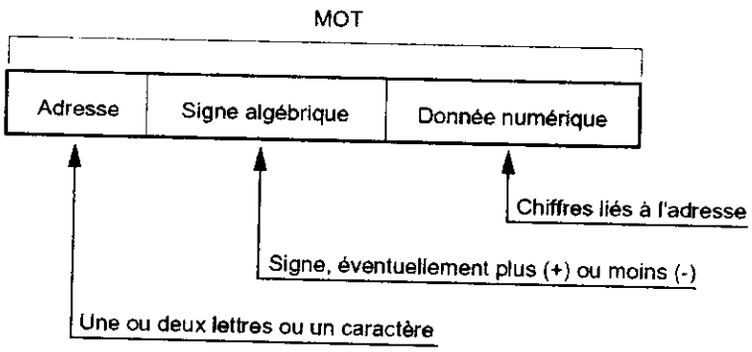


Figure 6: *Format général des mots [12].*

VIII.4.2. Structure générale d'un programme [11]:

Un programme d'usinage est caractérisé par (voir figure 7) :

- Un programme « G-Code » comporte des caractères obligatoires de début et de fin.
- Un programme est exécuté dans l'ordre d'écriture des blocs situés entre les caractères de début et de fin de programme.
- La numérotation des blocs n'intervient pas dans l'ordre de déroulement du programme.
- Les blocs sont numérotés dans l'ordre d'écriture.

Un programme ISO comporte les éléments essentiels suivants (voir figure 6) :

- Début de programme : caractère % suivi du numéro de programme et éventuellement d'un commentaire entre parenthèses,
- Ensemble des blocs à exécuter (programme effectif),
- Fin de programme : code M02,
- Fin de chargement de programme: caractère XOFF.

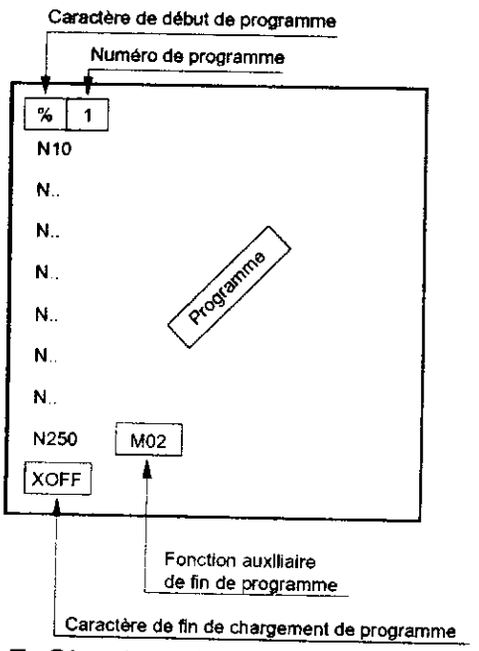


Figure 7: Structure d'un programme ISO [12].

VIII.4.3. Fonctions préparatoires G et fonctions auxiliaires M [11]:

VIII.4.3.1. Classification des fonctions préparatoires G :

Les fonctions préparatoires G sont de deux types :

- **Fonctions G modales :** fonctions appartenant à une famille de fonctions G se révoquant mutuellement. La validité de ces fonctions est maintenue jusqu'à ce qu'une fonction de même famille révoque leur validité.
- **Fonctions G non modales :** fonctions uniquement valide dans le bloc ou elles sont programmées (révoquée en fin de bloc).

VIII.4.3.2. Classification des fonctions auxiliaires M :

Les fonctions auxiliaires M déterminent les mouvements, la sélection de vitesse, l'arrosage,...etc. Les fonctions M sont comme les fonctions G et peuvent être soit des fonctions M modales ou des fonctions M non modales. De plus, elles peuvent être des fonctions «avant» ou «après».

- **Fonctions M «avant» :** fonctions exécutées avant le déplacement sur les axes programmés dans le bloc.
- **Fonctions M «après» :** fonctions exécutées après le déplacement sur les axes programmés dans le bloc.

IX. CONCLUSION :

Dans ce chapitre, nous avons présentés la construction des fraiseuses à commande numérique et leurs modes de fonctionnement ainsi que la syntaxe générale des programmes d'usinage décrivant la trajectoire des outils. Dans la suite de ce mémoire, nous allons considérer uniquement les fraiseuses à commande numérique à 3 axes pour l'usinage des surfaces gauches et en particulier l'opération d'ébauchage.


Usinage des surfaces gauches

I. INTRODUCTION :

Le processus d'usinage des surfaces gauches passe généralement par plusieurs étapes en fonction des contraintes imposées par le bureau des études. Parmi ces étapes, nous avons l'étape de choix des outils et des conditions de coupe ainsi que l'étape de choix des stratégies d'usinage pour les opérations d'ébauchage, de demi finition et de finition.

Dans ce chapitre, nous allons présenter les différents paramètres d'usinage ainsi que les stratégies d'ébauchage et de finition des surfaces gauches.

II. PRESENTATIN DE L'OPERATION DE FRAISAGE :

Le fraisage, comme son nom l'indique, regroupe les opérations d'usinage pouvant être effectuées sur une fraiseuse. Ces opérations aboutissent à l'obtention d'une géométrie quelconque (généralement une forme prismatique).

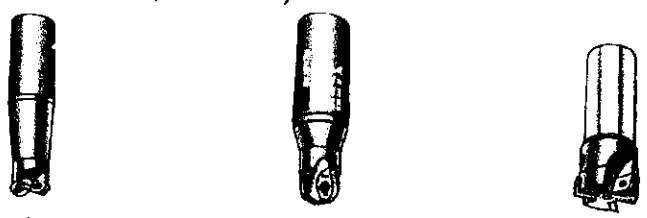
Avant d'entamer l'usinage (fraisage), il est nécessaire de tenir compte des paramètres suivants :

- Choix des outils (formes et dimensions) et des paramètres de coupe,
- Détermination des trajectoires des outils,
- Positionnement relatif outil/surface,
- Vérification de la compatibilité outil/surface.

II.1. Formes d'outils :

Les principaux types de fraises (outils) utilisées dans l'usinage des surfaces gauches sur des fraiseuses 3 axes sont les suivants (voir figure 1) :

- Fraise cylindrique (ébauche),
- Fraise hémisphérique (ébauche et finition),
- Fraise torique (ébauche).



a. Cylindrique. **b.** Hémisphérique **c.** Torique.

***Figure 1** : Différents types de fraises.*

Les principales dimensions de ces outils sont données par la figure 2 :

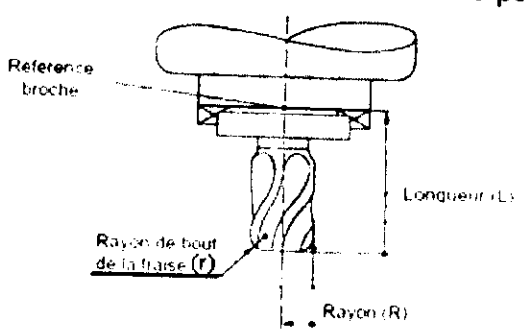


***Figure 2** : Paramètres d'outil [20].*

Pour les surfaces gauches, il n'existe pas de forme d'outils permettant l'obtention de ces surfaces en un seul mouvement élémentaire. Donc, le choix de l'outil est un compromis entre la rigidité de l'outil, la cinématique de la machine, l'opération d'usinage (ébauche, demi finition ou finition) et la forme de la pièce à usiner.

Pour générer une forme gauche par un enchaînement de déplacements, il faut une forme d'outil qui soit toujours tangente à la surface. La forme la plus simple est la sphère. L'outil hémisphérique est le plus utilisé dans la finition des surfaces gauches sur des fraisage 3 axes.

II.2. Paramètres de coupe [15] :

Lors d'un usinage par enlèvement de matière, on se retrouve dans la majorité des cas dans la configuration suivante

- Une lame d'outil (partie active) pénètre dans la matière pour enlever une partie de la pièce brute (copeau) (voir figure 3).
- L'outil suit une trajectoire par rapport à la pièce à usiner.

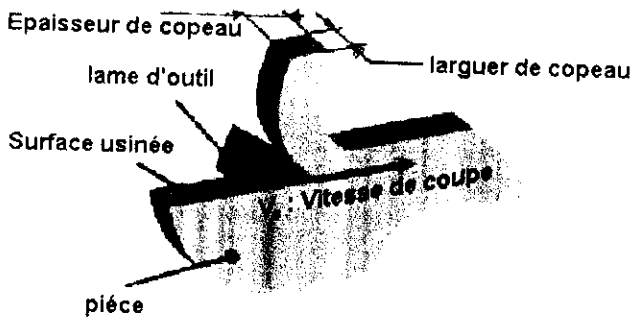


***Figure 3** : Vue générale sur l'usinage.*

Pour obtenir une pièce usinée dans des bonnes conditions (bon état de surface, rapidité de l'usinage, usure modérée de l'outil, ...etc.), il est nécessaire de régler certains paramètres spécifiques :

- Vitesse de coupe V_c,
- Vitesse d'avance F,
- Profondeur de passe a.

Ces paramètres dépendants de plusieurs paramètres : matière de la pièce, matière d'outil, rigidité d'outil, ...etc.

- **Vitesse de coupe :** c'est la vitesse linéaire du point le plus éloigné de la partie active de l'outil qui est en contact avec la pièce à usiner et elle est donnée par :

$$Vc = \frac{\pi.D.N}{1000} \qquad (2)$$

Avec :

D : correspond au point le plus éloigné de la partie active d'outil qui est en contact avec la pièce à usiner diamètre de la fraise en mm,
N : vitesse de rotation de la broche en tr/min,

V_c : vitesse de coupe en mm/min.

- **Vitesse d'avance :** c'est la vitesse de déplacement de l'outil sur la trajectoire d'usinage. Pour le fraisage, cette vitesse est donnée par :

$$F = Z.f_Z.N \qquad (3)$$

Avec :

Z : le nombre de dents de la fraise,
f_z : avance par dent par tour de l'outil en mm/ (tr.dent),
N : vitesse de rotation de la broche tr/min,
F : vitesse d'avance en mm/min.

- **Profondeur de passe :** c'est la pénétration axiale de l'outil dans la pièce pour enlever la matière. Cette profondeur dépend de l'outil (matière et dimensions) et de la matière de la pièce.

II.3. Types de trajectoires [16] :

Nous distinguons différents types de trajectoire :

- **Positionnement point à point :** le passage d'un point à un autre s'effectue en programmant la position finale et le trajet parcouru pour atteindre cette position n'est pas contrôlé (voir figure 4).

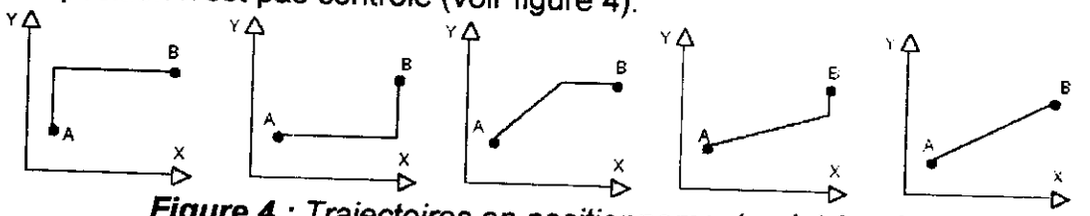


***Figure 4** : Trajectoires en positionnement point à point.*

- **Déplacement en paraxial :** les trajectoires sont parallèles aux axes de déplacement de la machine (voir figure 5) et la vitesse de déplacement (programmable) est contrôlée.

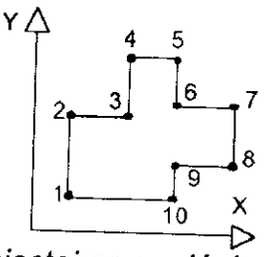


***Figure 5** : Trajectoires en déplacement paraxial.*

- **Déplacement en continu (contournage) :** pour ce cas, les différents axes sont contrôlés en vitesse et en position pour assurer une synchronisation permanente des mouvements soit dans le plan (contournage 2D) (voir figure 6) ou dans l'espace (contournage 3D).

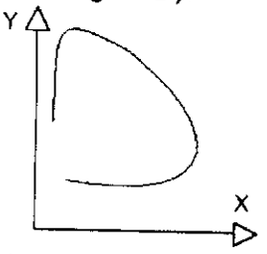


***Figure 6** : Trajectoires en continu (contournage en 2D).*

II.4. Modes d'usinage [17] :

On peut distinguer deux types d'opération d'usinage en fraisage :

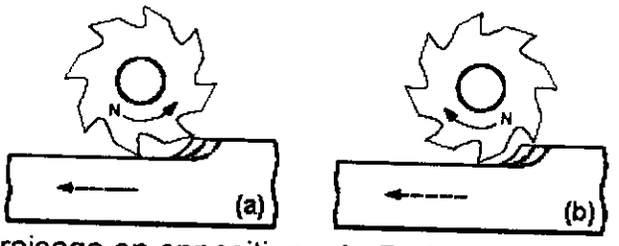
- les opérations axiales : l'outil se déplace uniquement le long de son axe (perçage, alésage, lamage, ...etc.).
- les opérations de fraisage : l'outil se déplace dans l'espace, essentiellement dans le plan normal à l'axe de la broche. Ces opérations peuvent être :
  - Opérations "en bout" : dans ce cas, l'outil est normal à la surface à usiner (voir figure 7.a).
  - Opérations "en roulant" : la surface à usiner est tangente à la génératrice de la fraise (voir figure 7.b).

a. Usinage en bout. **b.** Usinage en roulant.

Figure 7 : *Usinage en bout et en roulant.*

De même, on peut distinguer deux modes de fraisage [18] :

- **Fraisage en opposition :** les dents de la fraise en prise avec la matière avancent dans le sens de l'usinage (Figure 8.a). Elles attaquent donc la matière par une épaisseur de copeau nulle et terminent leur travail en quittant le copeau à son épaisseur maximale.
- **Fraisage en avalant :** les dents de la fraise en prise avec la matière avancent en sens inverse du sens d'usinage (Figure 8.b). Elles attaquent donc la matière par une épaisseur de copeau maximale et terminent leur travail en quittant le copeau à une épaisseur nulle.



a. Fraisage en opposition. **b.** Fraisage en avalant.

Figure 8 : *Types de Fraisage.*

III. STRATEGIE D'USINAGE :

Une stratégie d'usinage est une méthodologie utilisée pour générer une série d'opérations dans le but de réaliser une forme donnée. Elle permet d'associer un processus d'usinage à une entité d'usinage, c'est-à-dire un ensemble d'opérations comprenant la définition des outils, des conditions de coupe et des trajets d'usinage.

L'usinage des surfaces gauches passent généralement par trois étapes essentielles : ébauche, demi finition et finition.

III.1. Ebauchage [19] :

L'opération d'ébauchage des surfaces gauches est la première étape du processus de fabrication. Cette opération consiste à enlever le maximum de matière en un temps réduit sans tenir compte de la qualité de l'usinage tout en générant une forme plus moins proche de la forme finale avec une surépaisseur d'usinage pour les autres opérations. Cette opération peut être faite soit avec un outil cylindrique ou un outil hémisphérique.

Le trajet d'usinage le plus économique est celui qui nécessite un minimum de temps d'usinage. Ce temps peut être évalué par le taux d'enlèvement de matière; c'est-à-dire plus la quantité de matière enlevée est grande, plus le temps d'usinage est petit. Par conséquent, la meilleure politique d'usinage est l'usinage en ébauche en choisissant le plus grand outil possible sous des contraintes géométriques.

Le choix d'une stratégie d'usinage en ébauche doit permettre de réduire le temps total d'usinage. C'est un problème d'optimisation dont les variables sont :

- Paramètres géométriques des différents outils,
- Surépaisseurs d'usinage global ou local,
- Tolérance sur les surépaisseurs d'usinage,
- Différentes profondeurs de passes axiales,
- Trajets outils par niveau (mode de balayage et engagement radial),
- Paramètres des trajets outils hors matière.

Une stratégie d'usinage en ébauche doit donc définir l'ensemble de ces variables. Le choix devra se faire en considérant une optimisation en fonction du temps total d'usinage. Ce problème est complexe, sa résolution globale est difficile. Généralement, si la forme finale est obtenue à partir d'un bloc, l'ébauche est alors une succession d'usinages dans différents plans perpendiculaires à l'axe d'outil. Pour cette opération, l'outil le plus utilisé est un outil cylindrique (voir figure 9). Dans ce cas, l'ébauchage est une suite d'usinage de poches dans les différents plans de coupe (plan d'usinage) et dont la profondeur de passe (d) est fixé par l'opérateur. La forme cylindrique de l'outil laisse dans chaque plan des épaulements qu'il faut enlever en demi finition.

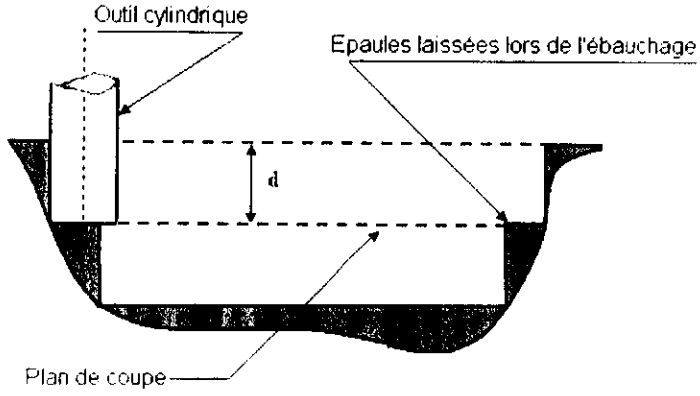


***Figure 9** : Ebauchage avec un outil cylindrique.*

III.1.1. Stratégies d'ébauchage :

Pour ébaucher des surfaces gauches, plusieurs stratégies peuvent être utilisées.

III.1.1.1. Stratégie des plans parallèle [21,22]:

Cette stratégie est composée de deux étapes. La première étape consiste à calculer les contours d'intersection entre des plans horizontaux et la surface à usiner. La deuxième étape quand à elle consiste à calculer les intersections entre les contours et des plans parallèles et verticaux ayant une certaine orientation par rapport au plan XZ (voir figure 10). Les plans d'intersection peuvent être dans n'importe quelle direction mais il y a toujours une direction optimale pour la quelle le temps d'usinage est le plus court. A la fin, l'usinage en ébauche se fait suivant des segments parallèles entre eux.

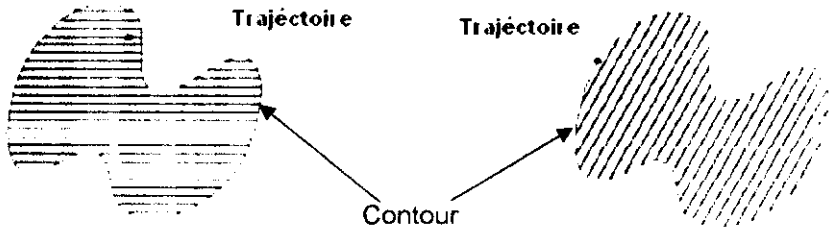


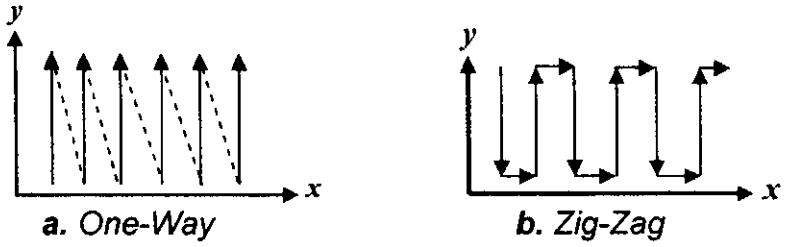
Figure 10: *Directions d'usinage.*

Aujourd'hui la stratégie des plans parallèles est la stratégie la plus utilisée en raison de ses avantages pratiques. Pour trouver la direction optimale des plans, il faut que :

- Chaque ligne de trajectoire doit être la plus long possible,
- Le nombre des rétractions de l'outil doit être le minimum possible.

Pour cette stratégie, deux modes de balayage de l'outil sont possibles.

- **One-Way (Aller simplement) :** pour ce mode d'usinage, le trajet d'outil en usinage est un ensemble de droites parallèles. Lors de l'usinage, l'outil suit une droite jusqu'à sa fin. Par la suite, l'outil quitte la surface et reprend l'usinage au début de la droite suivante. Ce processus est répété jusqu'à l'usinage de toute la surface (voir figure 11.a).
- **Zig-Zag (Aller et retour) :** pour ce mode d'usinage, le trajet d'outil en usinage est un ensemble de droites parallèles. Lors de l'usinage, l'outil suit une droite jusqu'à sa fin. Par la suite, l'outil se déplace à la fin de la droite suivante sans quitter la surface et reprend l'usinage en sens inverse. Ce processus est répété jusqu'à l'usinage de toute la surface (voir figure 11.b).



a. One-Way *b. Zig-Zag*

Figure 11 : *Modes de balayage de l'outil pour la stratégie des plans parallèles.*

Le choix de l'un des modes de balayage est un compromis entre la qualité et le temps d'usinage (productivité). L'ébauchage en Zig-Zag donne des flexions de l'outil dans des directions alternées. Tandis que l'ébauchage en One-Way, les flexions de l'outil sont dans la même direction. Pratiquement, Zig-Zag est utilisée pour l'usinage des matériaux tendres (aluminium), tandis que One-Way est utilisée pour l'usinage des matériaux durs (acier).

Pour que la stratégie d'ébauchage en plans parallèles soit optimale, il faut satisfaire les contraintes suivantes :

- Réduire au minimum le nombre de rétractions d'outil : la rétraction d'outil cause des mouvements d'outil dans l'espace d'usinage et peut causer des marques sur la surface usinée.
- Réduire au minimum le nombre de trajets élémentaires d'outil.
- Maximiser la longueur moyenne des trajets élémentaires d'outil puisque qu'elle permet d'améliorer la qualité de la surface.
- Permettre de choisir le mode de balayage en One-Way ou en Zig-Zag ainsi que l'usinage en opposition ou en avalant.
- Usinage des contours dans chaque plan pour éliminer la matière laissée entre les droites d'usinage.

III.1.1.2. Stratégie des contours décalés :

Cette stratégie est composée de deux étapes. La première étape consiste à calculer les contours d'intersection entre des plans horizontaux et la surface à usiner. La deuxième étape quand à elle consiste à calculer à chaque fois les contours décalés d'une certaine distance « ω » fixée par l'utilisateur et qui doit être inférieure au rayon de l'outil (voir figure 12). Les différents contours décalés constituent le trajet d'usinage final.

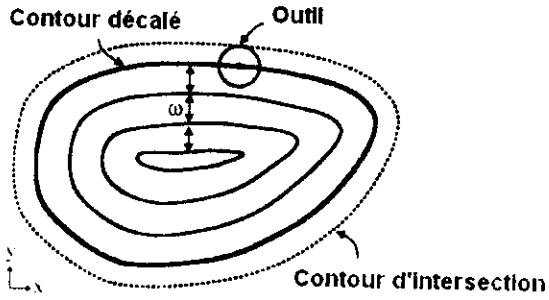


Figure 12: *Stratégie des contours décalés.*

L'inconvénient de cette stratégie est la possibilité d'existence de zones non usinées si la distance de décalage est plus grande que le rayon de l'outil ou si le contour est très courbé (voir figure 13).

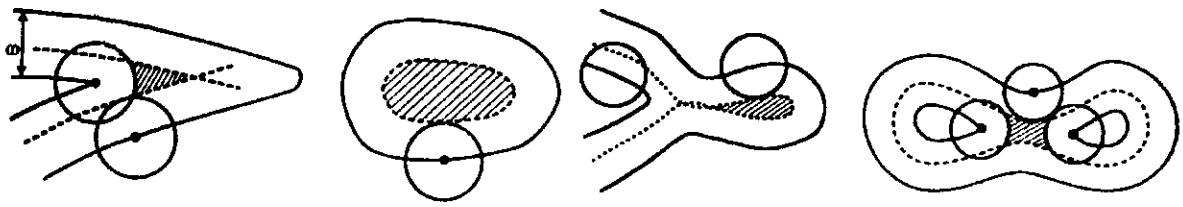


Figure 13: *Zones non usinées avec les contours décalés.*

III.2. Finition [19] :

Le but de l'opération de finition est de réaliser une surface respectant les contraintes du bureau d'études. Le choix d'une stratégie est un problème sous contraintes associées au défaut de forme et à l'état de surface.

Les trajets d'outils sont réalisés à partir d'une surface d'usinage S_M qui comporte toutes les informations nécessaires pour leur construction. Dans le cas de surfaces nominales S_D, S_M est définie par :

$$S_M(u,v) = S_D(u,v) + R.N_D \qquad (4)$$

Avec : N_D la normale à S_D au point $S_D(u,v)$ et R le rayon d'outil utilisé.

A partir de cette surface, la construction des trajets nécessite la connaissance d'une direction de balayage, d'un pas transversal (respect d'une hauteur de crête) et d'un pas longitudinal (précision de la génération de trajectoires) (voir figure 14).

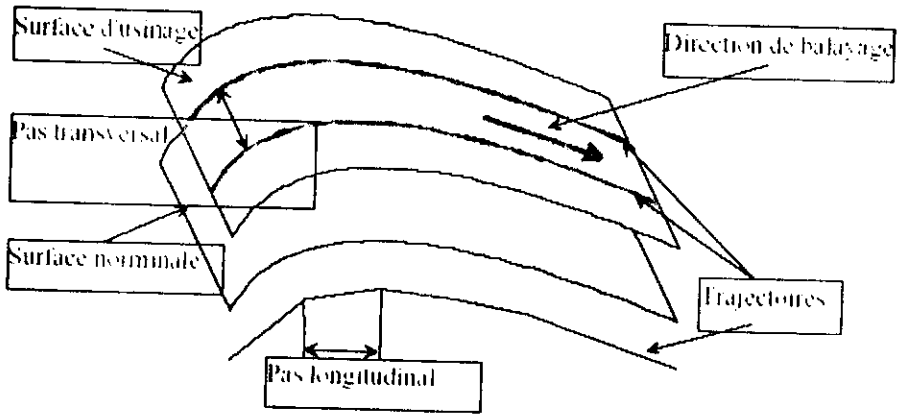


Figure14 : *Description d'une stratégie de finition.*

Pour l'usinage en finition des surfaces gauches sur des fraiseuses à 3 axes, l'outil le plus utilisé est l'outil hémisphérique qui doit être constamment tangent à la surface. Ses différentes positions sont données par (voir figure 15) :

$$\begin{cases} \overrightarrow{OC_E} = \overrightarrow{OC_C} + r\vec{n} \\ \overrightarrow{OC_L} = \overrightarrow{OC_C} + r\vec{n} - r\vec{u} \end{cases} \qquad (5)$$

- **Stratégie d'usinage avec Z-Constant :** elle consiste à utiliser des plans parallèles et horizontaux pour calculer l'intersection de ces plans avec la surface et par la suite générer le trajet d'usinage (voir figure 18).

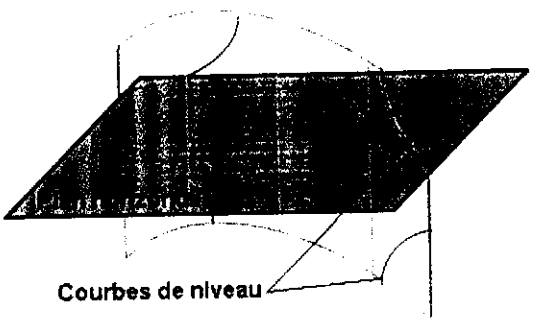
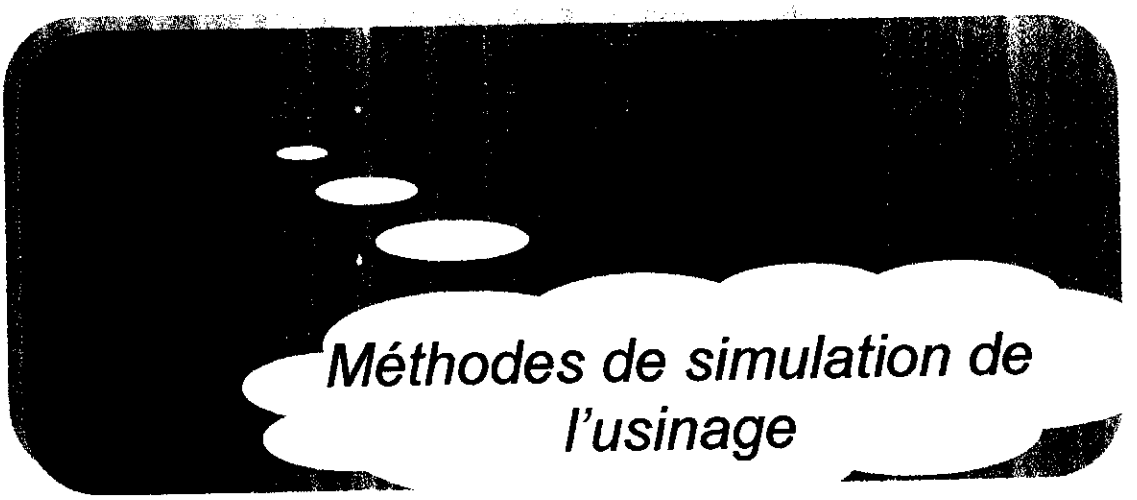


Figure 18 : *Usinage avec Z-Constant.*

IV. CONCLUSION :

Dans ce chapitre nous avons présentés l'opération de fraisage, les paramètres de coupe à considérer ainsi que les différentes stratégies d'usinage nécessaires pour réaliser l'ébauchage et la finition des surfaces gauches. Cette partie permet de passer dans le chapitre suivant au problème de simulation de l'opération d'enlèvement de matière pour l'opération d'ébauchage.


Méthodes de simulation de
l'usinage

I. INTRODUCTION :

La simulation d'usinage est un moyen très important et très utilisé pour examiner et vérifier les trajets d'usinage avant d'aller vers l'usinage réel. Sans simulation, les chemins d'outil sont examinés à plusieurs reprises sur des pièces modèles et les corrections sont effectuées au fur et à mesure. L'essai sur les machines réelles exige de travailler sur des machines chères où les risques d'erreurs sont très importants ainsi que les temps d'usinage.

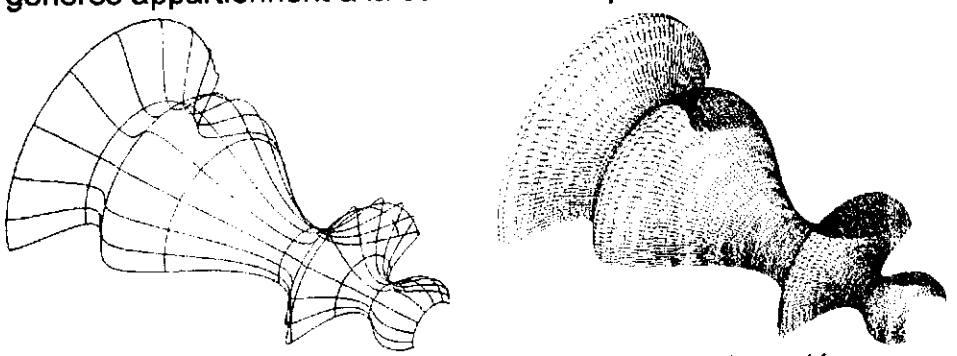
Dans ce chapitre, nous allons présenter les méthodes de représentation des surfaces et les méthodes les plus utilisées dans la simulation d'enlèvement de matière.

II. METHODES D'APPROXIMATION DES SURFACES :

Dans le but de réaliser l'étape de simulation d'enlèvement de matière (usinage), nous avons besoin de trouver une méthode de représentation des objets permettant de simplifier les calculs (approximation de surface). Les méthodes d'approximation d'une surface sont différentes et chaque méthode a ses avantages et ses inconvenants. Parmi ces méthodes on trouve :

II.1 Triangulation :

La méthode la plus simple c'est la triangulation. Cette méthode approxime la surface théorique par un ensemble de triangles avec une certaine précision (tolérance) spécifiée au départ. Plus la précision est importante, plus l'approximation de la surface est bonne (voir figure 1). Pour cette méthode, les sommets des triangles générés appartiennent à la surface théorique.



a. Surface théorique. **b.** Surface triangulée.

***Figure 1** : Approximation d'une surface par des triangles.*

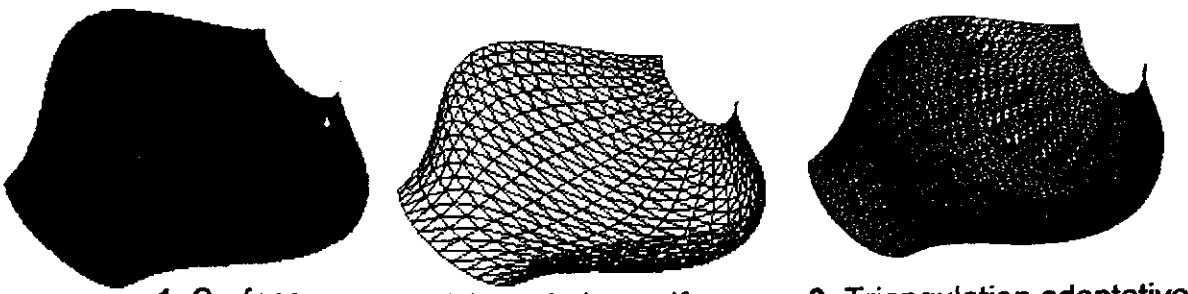
II.1.1 Types de triangulation :

Pour trianguler une surface, nous avons le choix entre deux méthodes.

- **Triangulation uniforme** : dans ce cas, le nombre de triangles est fixé suivant chaque direction dans le plan paramétrique (u, v) et par la suite, les triangles sont générés dans l'espace. Par conséquent, le nombre total de triangle est connu au départ (voir figure 2.2)

- **Triangulation adaptative** : dans ce cas, les triangles sont créés d'une manière dynamique et adaptative en fonction de la précision exigée par l'utilisateur. Donc, le nombre de triangle est inconnu est dépend de la précision et de la complexité géométrique de la surface théorique (voir figure 2.3).

La figure 2.1 représente la surface théorique à trianguler, tandis que la figure 2.2 représente la triangulation uniforme où nous avons utilisés 20 triangles suivant les direction *u* et *v*. La figure 2.3 montre la triangulation adaptative de la même surface.



1. Surface. **2.** Triangulation uniforme. **3.** Triangulation adaptative.

Figure2 *: Types de triangulation.*

Pour la triangulation adaptative, chaque triangle obtenu doit répondre à certains critères, sinon il sera triangulé à nouveau. Les différents critères à respecter sont les suivants :

- La taille de chaque coté ne doit pas dépasser une certaine longueur (d) imposée par le programmeur,
- la distance entre la surface à usiner et le milieu de chaque coté du triangle ainsi que son centre de gravité doivent aussi vérifier une certaine erreur imposée par l'utilisateur (d').

Donc, pour chaque triangle, nous avons :

1. (03) cotés dont la taille est inférieure à (d),
2. (03) cotés dont la distance entre leurs milieux et la surface théorique est inférieure à (d'),
3. Distance entre le centre de gravité et la surface théorique est inférieure à (d').

La triangulation adaptative génère des triangles qui s'adaptent à la forme locale d'où son nom. Par ailleurs, si le triangle obtenu ne répond pas aux critères cités ci-dessus, il est subdivisé récursivement jusqu à ce que le triangle généré vérifie tous les critères.

La figure 3 représente les différents cas pouvant être rencontrés après la génération d'un triangle :

(1): Un seul côté est hors tolérance : le triangle est divisé en deux.
(2): Deux côtés sont hors tolérance : le triangle est divisé en trois.

(3): Tous les côtés sont hors tolérance : le triangle est divisé en quatre.
(4): Le centre de gravité est hors tolérance : le triangle est divisé en trois.

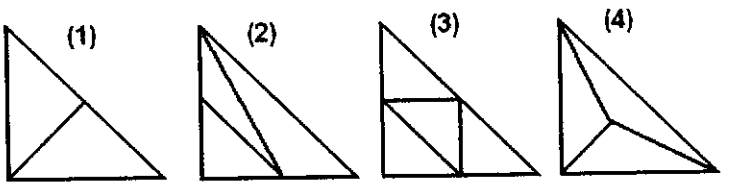


Figure 3 : Différents cas d'anomalies dans la triangulation adaptative.

II.2. Z-buffer [23,26] :

La méthode du Z-buffer (ou méthode Z-map) est aussi une méthode d'approximation des surfaces. Cette méthode est différente dans son approche et est particulièrement performante. La base de la méthode est de projeter une grille selon une direction donnée sur la surface, suivant le point de vue choisi.

La construction de la surface est obtenue par intersection entre un ensemble de droites parallèles à Z et la surface. Les surfaces les plus adaptées ont une fonction qui réalise une bijection entre le plan de base XY et les points de la surface. Pour chaque droite, on cherche toutes les intersections avec l'ensemble des surfaces, et on retient l'intersection la plus haute appartenant à la peau de la pièce (voir figure 4). Cette méthode est très simple à mettre en œuvre mais sa précision est en fonction du pas de la grille.

Comme la grille est indépendante de l'orientation de la pièce, cette méthode est peu précise dans le cas d'approximation de parois ayant un angle de dépouille faible, ou verticale, c'est-à-dire parallèle à la direction de projection.

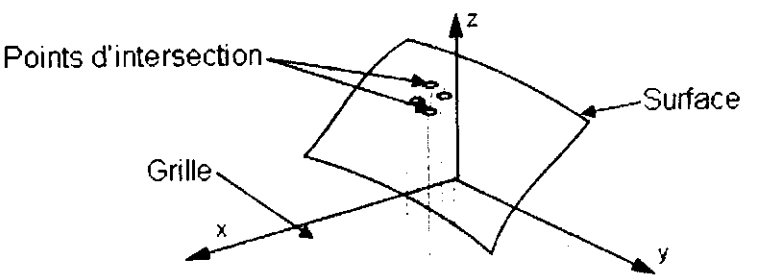


Figure 4 : Méthode du Z-buffer.

III. SIMULATION, VERIFICATION ET CORRECTION [23] :

Pour la détection et l'élimination des erreurs, trois aspects distincts sont utilisés : simulation, vérification et correction.

Dans la phase de simulation, le modèle géométrique de la pièce est modifié et le modelage du volume balayé de chaque mouvement d'outil doit être fait. La vérification de l'exactitude de l'opération d'usinage exige une comparaison du modèle géométrique de la pièce obtenue avec un modèle géométrique de la pièce désirée.

Les systèmes de simulation doivent contenir une information pour spécifier la tolérance désirée permettant la comparaison directe des dimensions des surfaces usinées et des surfaces théoriques avec la tolérance spécifiée. Finalement, les erreurs détectées doivent être corrigées. Si les erreurs peuvent être tracées pour les mouvements d'outil, alors une correction manuelle ou automatique de chemin d'outil sera possible.

IV. DEFINITION DE LA SIMULATION [23, 24, 27]:

La simulation de l'usinage est la construction de l'enveloppe de la surface générée par le mouvement élémentaire de l'outil. L'aspect le plus important dans la simulation d'usinage est l'évaluation des erreurs. Ces erreurs peuvent être soit interférence ou excès de matière. L'objectif de simulation est de comparer la surface théorique avec la surface usinée afin de localiser les erreurs. Deux approches principales de simulation des processus d'usinage ont été développées :

- Approche exacte analytique,
- Approche approximative.

L'approche exacte qui utilise des opérations booléennes entre l'enveloppe du mouvement de l'outil et la surface à usiner est informatiquement coûteuse en temps de calcul. Pour cette approche, le coût de simulation est proportionnel à $O(N^4)$, où **N** est le nombre de mouvements d'outil. Un programme d'usinage des surfaces gauches peut être composé de milliers de mouvements et par conséquent le temps de calcul devient très important.

Afin d'augmenter l'efficacité de simulation, un certain nombre de méthodes approximatives de simulation ont été proposées. Le coût informatique de ces méthodes est simplifié et il est proportionnel à O(N). Nous nous intéressons uniquement à l'approche approximative. Cette approche est adaptée pour rendre l'opération d'usinage réaliste (contrôle visuel et évaluation des erreurs).

Dans ce cas, l'objectif de la simulation est de détecter les endroits pour lesquels la surface usinée est en dehors de l'intervalle de tolérance imposé $[T_o, T]$ où T et T_o sont des valeurs définies par l'utilisateur. Les écarts en dehors de l'intervalle de tolérance sont appelés «erreurs d'usinage ».

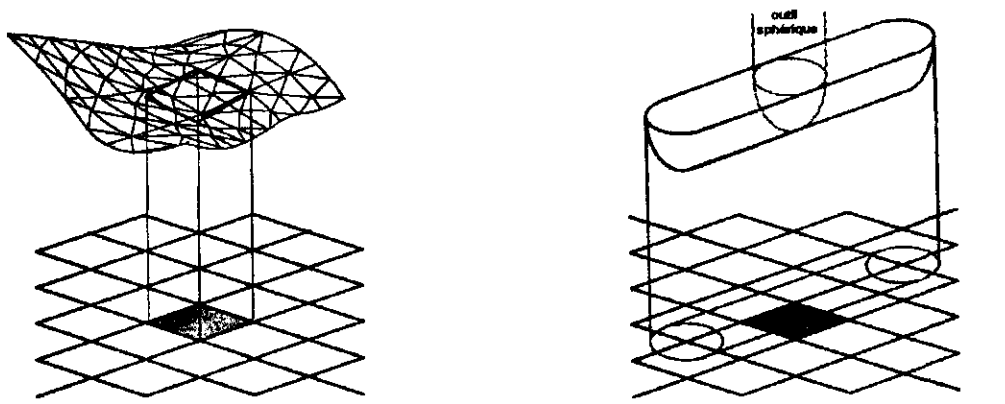
Pour réaliser la simulation avec l'approche approximative, il est nécessaire de passer par les trois étapes suivantes :

- **Discrétisation** : dans cette étape, un ensemble des points de la surface est calculé pour approcher la surface avec un espacement entre les points qui dépend de la taille, de la forme de l'outil, de la courbure locale et de la précision désirée de la simulation.
- **Intersection** : c'est l'étape de calcul des intersections entre les vecteurs normaux à la surface et l'enveloppe de l'outil pour chaque mouvement d'outil afin de générer le nouveau état de la surface.

- **Localisation :** pour accélérer les calculs d'intersection, il est nécessaire de localiser les calculs par l'élimination des vecteurs qui n'ont aucune possibilité d'intersection avec l'enveloppe de mouvement d'outil.

Généralement, les méthodes de simulation d'usinage à trois axes supposent que la surface a une seule valeur de z pour n'importe quel point dans le plan XY. Nous pouvons choisir la direction des vecteurs normaux pour être parallèles au plus grand axe (l'axe Z) de l'outil. Par conséquence, un vecteur peut être intersecté seulement s'il se trouve directement sous le chemin d'outil. Ceci signifie que les points peuvent être regroupé dans des "régions" comme montré par la figure 5-a.

La localisation est réalisée en trouvant l'ensemble des « régions » se trouvant sous l'ombre de l'outil et examiner seulement les points qui se trouvent dans ces régions (voir figue 5-b). Donc un petit pourcentage de tous les points est examiné pour chaque mouvement d'outil.



a. Création des régions dans le plan XY. **b**. Localisation des régions.

Figure 5 : Construction des régions.

V. METHODES DE SIMULATIONS USUELS :

Quatre méthodes principales de simulation usuelles existent :

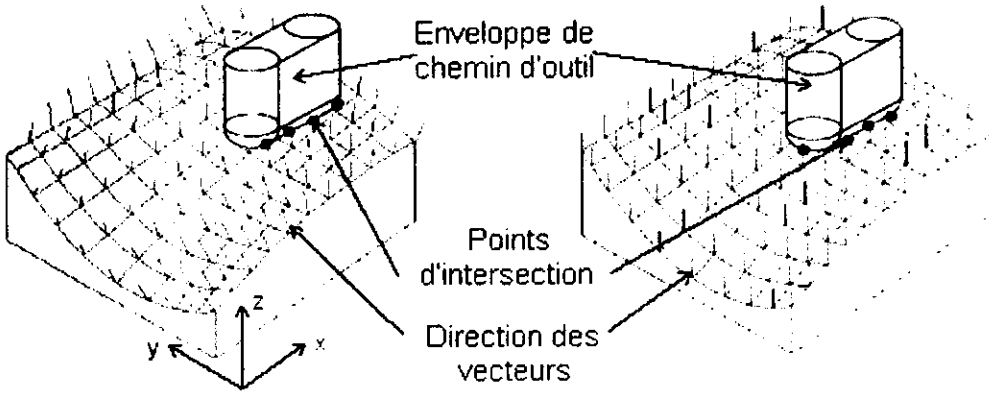
V.1. Technique point-vecteur [23, 24, 25, 26] :

La technique «point-vecteur» a pour but de calculer la distance entre la surface usinée et la surface théorique. A partir d'un point de la surface, on construit une droite dans une direction donnée, puis on cherche toutes les intersections de cette droite avec les trajets élémentaires de l'usinage. Ensuite, on calcule la distance entre chaque point d'intersection et la surface théorique. La plus petite distance est celle qui laisse le moins de matière avec une erreur minimale. Pour mettre en place cette technique il faut :

- Discrétiser la surface,
- Orienter les vecteurs,
- Modéliser le trajet d'usinage.

Un ensemble «point-vecteur» est construit sur la surface, tel que les points appartiennent à la surface théorique et l'extrémité des vecteurs appartient à la surface brute de la pièce. Le calcul se résume ainsi au calcul de l'intersection entre un segment et l'enveloppe d'outil. Le sens du vecteur indique la surépaisseur de matière laissée par rapport à la surface nominale après le passage de l'outil (voir figure 6).

A chaque point, on associe un vecteur. On peut donner à ce vecteur, une direction normale à la surface (voir figure 6.a), mais pour simplifier les calculs et ne considérer que la zone qui est sous l'outil, on préfère donner une direction parallèle à l'axe de l'outil (axe Z) (voir figure 6.b).



a. Orientation suivant les normales. ; **b.** Orientation suivant z.

Figure6 : Méthode «point-vecteur» [3].

V.2. Technique du Z-buffer [23, 24, 25, 26] :

Cette technique s'adapte très bien à l'usinage à trois axes pour une direction de projection l'axe de l'outil (l'axe Z). Cette méthode passe par la construction d'une grille dans un plan XY. A chaque point de la grille, on associe un segment vertical d'une altitude z initiale. Lors de l'usinage, on calcule l'intersection des segments avec le trajet d'usinage (voir figure 7). Pour chaque segment, on retient l'altitude d'intersection la plus basse. Cette méthode est très simple à mettre en œuvre mais la précision est fonction du pas de la grille. On obtient une représentation Z-buffer du trajet d'usinage. Les erreurs de simulation proviennent du pas de la grille et de la modélisation du trajet. Pour mener les calculs demandés lors de la simulation, il suffit de réaliser un Z-buffer de la surface finale, du brut, et de la surface usinée. On obtient alors par soustraction des différents Z-buffer entre eux, les erreurs d'usinage, les zones non usinées, les volumes de matière enlevée et les collisions.

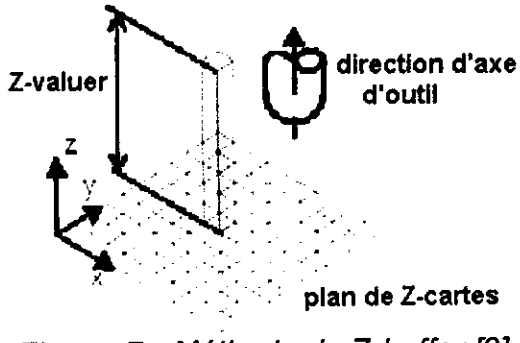


Figure 7 : Méthode du Z-buffer [9].

V.3 Technique solide -CSG- [26]:

Cette méthode s'appuie sur une modélisation solide de l'enveloppe du trajet de l'outil et qui n'est possible que pour un usinage à trois axes. La figure 8 montre les étapes de création de l'enveloppe d'un outil hémisphérique entre deux positions. Par les mouvements élémentaires d'un outil, on construit une primitive solide qu'il suffit de soustraire au solide modélisant la surface pour obtenir la contribution du mouvement élémentaire à la surface usinée. Cette solution est précise mais très coûteuse en temps de calcul surtout si le programme d'usinage est composé de milliers de déplacements.

a. Balayage rotatif. **b**. Balayage linéaire. **c**. Enveloppe de l'outil.

***Figure 8** : Etapes de création de l'enveloppe de l'outil.*

V.4. Ensemble des dexels [27]:

Un dexel est un solide prolongé rectangulaire à travers la direction de l'objet à représenter. Soit D un ensemble des dexels défini par un quadruplet (R; X; L; C) (voir figure 9) avec :

- $R = (R_x, R_y) \in N^2$: décrit la résolution du volume d'information dans le plan XY.
- X est un ensemble $R_x * R_y$ dexels.
- $L \in R^3$: est l'endroit de l'ensemble d'information dans l'espace 3D. Ce point est habituellement le point central du plan de grille contenant les dexels.
- C = (Cx ; Cy ; Cz), avec Cx ; Cy ; Cz $\in R^3$, est le système de coordonnées local décrivant l'orientation de l'ensemble des dexels dans l'espace 3D.

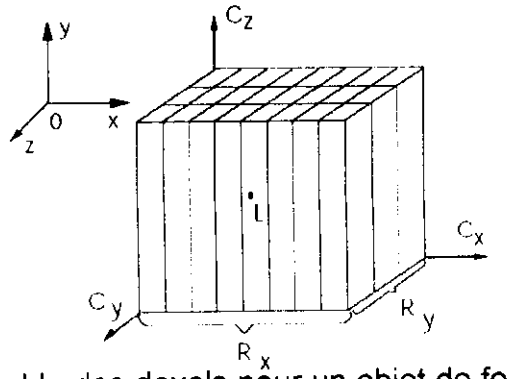


Figure 9 : Ensemble des dexels pour un objet de forme cubique [9].

Un dexel dex ∈ D.X est défini par un triplet (top, bottom, n) où :

- top: est la taille de la valeur supérieure d'un dexel.
- bottom : est la taille de la valeur inférieure d'un dexel.
- n ∈ R^3 : est la taille de vecteur gradient pour le dexel. Il est calculé en utilisant les valeurs du top de l'ensemble des dexels.

$$dex_{i,j}.\vec{n} = \begin{pmatrix} dex_{i+1,j}.top - dex_{i-1,j}.top \\ dex_{i,j+1}.top - dex_{i,j-1}.top \\ -1 \end{pmatrix} \quad (1)$$

Avec dex (i, j) ∈ D.X

Pour la simulation de l'usinage, la représentation de la pièce usinée et de l'outil se fait par deux ensembles des dexels pour chacun, DW (pièce) et DC (outil). L'opération d'intersection est évaluée pour la pièce et l'outil.

La figure 10 montre les différents cas d'intersection des dexels de la pièce et de l'outil.

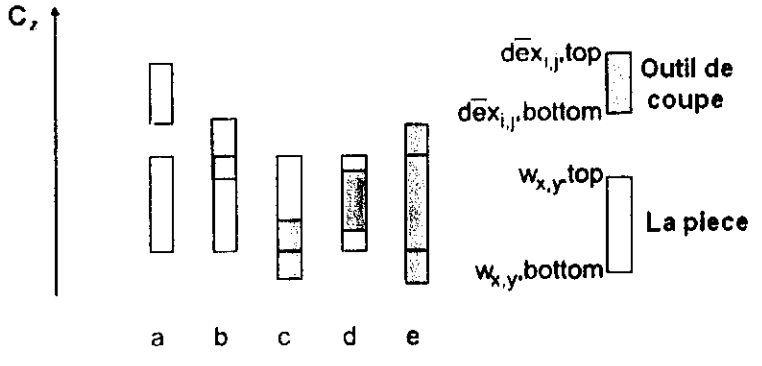


Figure 10 : Intersections possibles entre les dexels de l'outil et la pièce [9].

L'évaluation de l'erreur peut être facilement réalisée en comparant les dexels de la pièce intermédiaire DW.X avec les dexels de la pièce de conception désiré DD.X. L'évaluation locale d'erreur compare un seul dexel tandis que l'évaluation globale compare tous les dexels et qui est donnée par la formule suivante :

$$E = \sum_{i,j} (dex_{i,j}.top - w_{i,j}.top)^2 \quad (2)$$

Où dex(i,j) ∈ DW.X et w(i,j) ∈ DD.X.

La figure 11 montre les différentes zones qui peuvent exister à savoir : excès de matière, dans l'intervalle de tolérance et interférence.

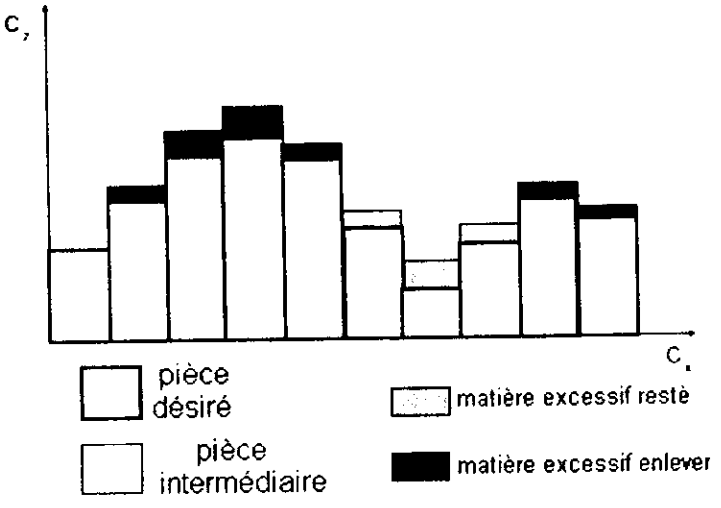


Figure 11 : Résultats de la simulation.

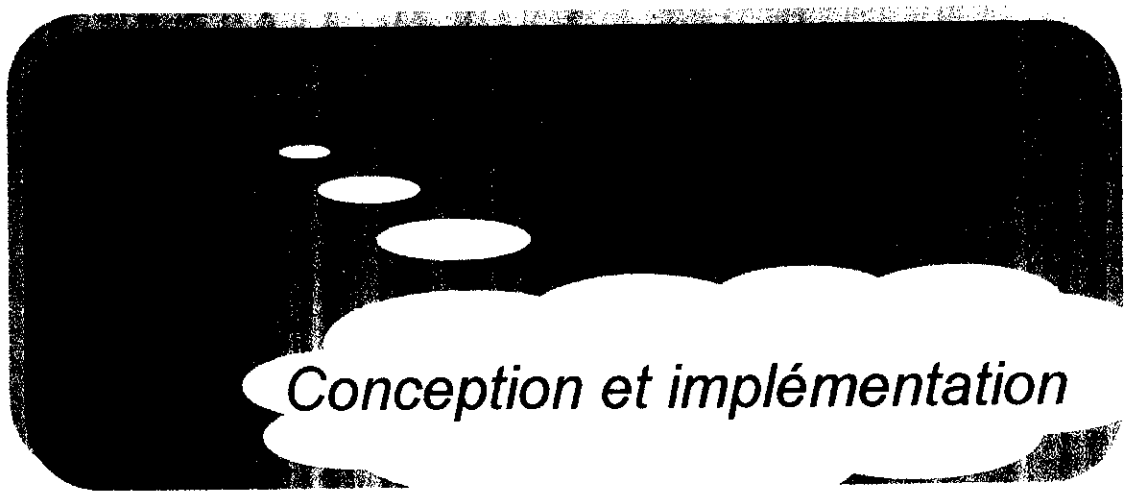
VI. RECHERCHE DES ZONES NON USINEES ET CONTROLE DE TOLERANCE :

Les zones non usinées sont des zones qui ont une erreur supérieure à la valeur maximale de la tolérance.

A partir de la surface théorique initiale, deux surfaces offset sont construites et positionnées de part et d'autre de la surface initiale en fonction de la répartition des tolérances d'usinage. Si la surface usinée est comprise entre les deux surfaces offset, alors la surface usinée est dans l'intervalle de tolérance. Si des zones sont au dessus de la surface offset extérieure, nous avons un excès de matière. Si des zones sont au dessous de la surface offset inférieure, nous avons des interférences.

VII. CONCLUSION :

Dans ce chapitre, nous avons présentés les méthodes de d'approximation des surfaces ainsi que les méthodes de simulation d'enlèvement de matière. Dans notre projet, nous allons utiliser la méthode point-vecteur pour réaliser la simulation d'enlèvement de matière en ébauchage. Dans le chapitre suivant, nous allons présenter notre conception et imlimentation.


Conception et implémentation

I. INTRODUCTION :

Dans ce chapitre nous allons présenter la conception de l'application avec le langage de modélisation UML en utilisant cinq diagramme de l'ensemble des diagrammes de cette langage en détaillant le diagramme de classe, diagramme de cas, diagramme de collaboration et en finissant par les diagrammes d'activité et les diagrammes de séquence, nous détaillons aussi quelques algorithmes utilisés dans la réalisation de l'application.

II. METHODE DE MODELISATION :

Comme nous avons utilisés un langage orientés objet dans la réalisation de notre application, nous allons faire la modélisation en utilisant le langage de modélisation UML qu'est un langage visuel et adapté à toutes les phases du développement et compatible avec toutes les techniques de réalisation de l'application et indépendant du langage de programmation utilisé et qui permet différentes vues pour représenter notre système.

III. REALISATION DE L'APPLICATION :

Pour réaliser notre application nous allons passés par plusieurs étapes.

III.1. Cahier de charge :

Dans le but de structurer les besoins de notre système, nous allons faire un cahier de charge.

III.1.1. Présentation du projet :

Notre projet s'insère dans le cadre de développement d'outils de conception et de fabrication des surfaces gauches initiés par l'équipe Conception et Fabrication Assistées par Ordinateur (CFAO) au niveau de la Division Productique et Robotique du centre de développement des technologies avancés (CDTA).

Dans ce projet on s'intéressera à la simulation et à la vérification virtuelle de l'opération d'ébauchage des surfaces de formes libres usinées sur des fraiseuses à commande numérique à 03 axes. Le but de ce travail est le développement d'une application logicielle graphique et interactive sous Windows qui permet, à partir des différentes positions d'outil constituant la trajectoire d'usinage, de simuler et de vérifier virtuellement l'opération d'usinage en ébauche des surfaces et de détecter automatiquement les zones en dehors de la tolérance d'usinage exigée et à apporter les corrections nécessaires au trajet d'usinage et par la suite faire une adaptation des vitesses d'avance en fonction de la quantité de matière enlevée, c'est pour cela le thème :

« Simulation et vérification de l'opération d'ébauchage des surfaces gauches sur des fraiseuses à commande numérique à 3 axes ».

III.1.2. Problématique :

Les pièces de formes libres (moules, matrices, formes esthétiques, formes aérodynamiques, ...etc.) sont très rencontrées dans notre vie quotidienne. Ces pièces sont conçues dans le but d'assurer des fonctions inscrites dans le cahier des charges. Ces pièces ne peuvent être usinées que sur des fraiseuses à commande numérique à 03 ou à 05 axes en raison des géométries très complexes. L'usinage de ces surfaces passe généralement par trois étapes : ébauche qui permet d'enlever le maximum de matière, demi finition où on s'approche de la forme finale et finition où on obtient la forme voulue. Toutes ces étapes nécessitent la génération d'un ensemble d'instructions écrites dans un langage propre à la machine appelé programme « G-Code ».

Les programmes d'usinage "G-Code" des surfaces gauches en ébauche contiennent un nombre très important de lignes d'instructions, et par conséquent, la vérification manuelle de ces programmes devient impossible. Les surfaces usinées contiennent toujours des erreurs et qui peuvent être soit l'existence des zones non usinées soit l'existence des zones d'interférence et de collision. La simulation de l'opération d'usinage est un outil permettant de réduire les temps d'usinage effectif et de valider les trajets d'usinage en détectant les différentes erreurs et leurs positions afin de les corriger.

III.1.3. Objectifs visés :

Dans notre application nous avons réalisé la simulation, vérification, correction et adaptation des vitesses d'avance, et les objectifs fixés sont les suivants :

❖ **Premier objectif :**

Notre objectif principal dans l'application est la simulation de l'opération d'ébauchage sur des fraiseuses numérique à 3axes ce même objectif peut être subdivisé en plusieurs sous objectifs :

- Détermination d'un certain nombre de points sur la surface nominale que nous souhaitons usiner.
- Approximation de la surface par un certain nombre des triangles à partir des points déterminés.
- Optimisation de nombre de triangles de surface en fonction des paramètres introduits par l'utilisateur (erreur maximum et longueur maximum).
- Détermination du brut minimum qui correspond à la surface à usiner.
- Approximation du brut par des triangles et des rectangles.
- Mise à jour des cordonnés des points en fonction de la position de l'outil.

❖ **Deuxième objectif :**

Variation de la vitesse de visualisation de la simulation pour être plus rapide.

❖ **Troisième objectif :**

La détermination des points vérifiant la tolérance exigée par l'utilisateur et les points non vérifiant. Nous avons 3 cas possibles qui sont :

1. Excès de matière.
2. Vérifie la tolérance.
3. Interférence.

❖ **Quatrième objectif :**

La correction de la trajectoire d'usinage en éliminant les zones d'interférences.

❖ **Cinquième objectif** :

L'adaptation de la vitesse d'avance de l'outil en fonction de son trajet d'usinage, la quantité de matière enlevée, les paramètres d'outil et matériaux de la surface à usiner entre chaque deux position successives.

❖ **Sixième objectif :**

La Génération du programme d'usinage « G-Code » de la pièce à usiner pour qu'il soit adapté aux nouvelles vitesses d'avance calculées et trajectoire corrigée.

III.1.4. Plateforme exigée :

1. Recommandations matérielles (configuration minimum nécessaire) :

   - Intel Pentium 4 de 2 GHz ou équivalent.
   - 512 Mo de mémoire RAM.
   - Carte accélératrice 3D de 64 Mo compatible OpenGL ou équivalente.
   - Lecteur de CD-ROM/DVD-ROM.
   - Souris PS/2.
   - Clavier.
   - Ecran de 19 pouces.

2. Systèmes d'exploitation nécessaires :

   - Windows 98, Windows ME, Windows XP et Windows 2000
   - Notez que Windows 95, Windows NT 4.0 ne sont pas compatibles.

3. Logiciels requis :

   - Builder C++ V.6.
   - Bibliothèque graphique OpenGL.

III.2. Solution de la problématique :

Pour atteindre les objectifs précédemment cités, nous sommes arrivés aux solutions suivantes :

1. **Premier objectif** : Pour chaque surface à usiner, nous passons par les étapes suivantes.

   1) Pour déterminer les points de la surface à usiner nous déterminons d'abord ses points dans le plan paramétrique (u,v) tel que le nombre des points est introduit par l'utilisateur puis nous faisons appel à une fonction qui permet d'associer à chaque point sur le plan paramétrique p(u,v) un et un seul point sur la surface dans l'espace p(x,y,z).

$$\mathbf{P(u, v)} \xrightarrow{\text{Fonction de transformation}} \mathbf{P(x, y, z)}$$

   2) Après la détermination des points qui représentent la surface, nous pouvons relier ces points par un certain nombre des triangles pour obtenir un réseau de triangles qui va approximer la surface à usiner. La figure 1 montre la démarche de génération des triangles.

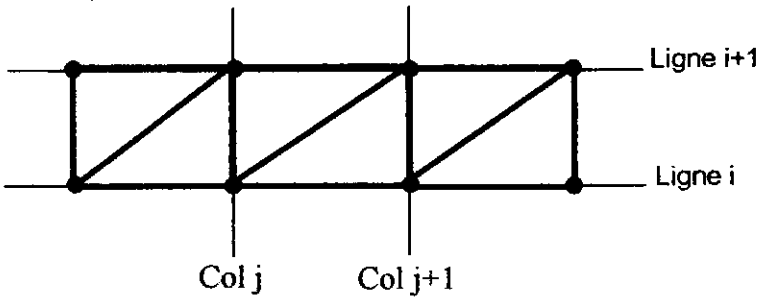


Figure 1: *Génération des triangles à partir des points.*

Donc, pour chaque itération nous générons deux triangles :

1. Le premier triangle est le triangle déterminé par les 3 points suivants :

   P(i, j) et P(i+1, j) et P(i+1, j+1)

2. Le deuxième triangle est le triangle déterminé par les 3 point, suivants :

   P(i, j) et P(i, j+1) et P(i+1, j+1)

Cette génération est répétée jusqu'à le balayage de tous les points de la surface qui correspondent au nombre de lignes n multiplié par le nombre de colonnes m.

   3) L'optimisation se faite pour que les contraintes introduites par l'utilisateur soient vérifiées en utilisant un nombre minimum possible de triangles. Le but de cette optimisation est la détermination de nombre minimum de lignes et de colonnes qui approximent la surface et vérifient les contraintes imposées :

Si l'une des contrainte (erreur maximum ou longueur maximum) n'est pas vérifiée pour un triangle donné et suivant les cas possibles (segments concernés par l'erreur ou le centre de gravité), nous multiplions soit le nombre de lignes par 2 soit le nombre de colonnes par 2 soit nous multiplions tout les deux par 2 et nous régénérons les points et les triangles. Chaque foi que nous avons multipliée par 2 nous faisons le même test, et ainsi de suite jusqu'à ce que les contraintes soient

vérifiée. Les valeurs déterminées ne sont pas nécessairement les valeurs optimales d'ou la nécessité de réduire ces valeurs.

Nous passons maintenant à l'étape de réduction du nombre de triangles pour obtenir le minimum nombre qui vérifié les contraints. Cette réduction est faite en 2 étapes : nombre de lignes et ensuite nombre de colonnes. Nous divisons le nombre de lignes (nombre de colonnes) par 2 pour avoir 2 intervalles (voir figure 2):

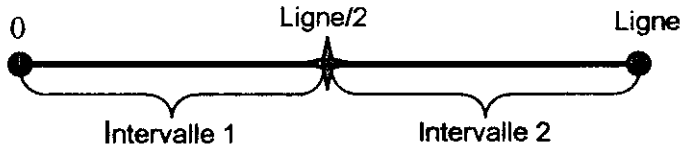


Figure 2: *Méthode utilisée dans l'optimisation des lignes.*

Ensuite, nous testons si le premier intervalle vérifié les contraintes, deux cas sont possibles :

Si oui alors nous divisons encore cet intervalle en deux et en prend

Ligne=Ligne/2 (colonne=colonne/2).

Si non, cela indique que la valeur optimale se trouve dans le deuxième intervalle. Donc nous testons cet intervalle et nous faisons le même traitement que le premier.

Ce processus est répété jusqu'à ce qu'aucune division n'est possible. À la fin la, nouvelle valeur de ligne, (respectivement colonne) est retenue et sera utilisée pour générer le nombre optimal des triangles.

4) La détermination de brut minimum de surface se fait comme suit : En premier lieu il faut déterminer les limites du brut en cherchant les points les plus éloignés sur la surface. Pour déterminer le brut min de la surface, nous avons besoin de calculer les valeurs suivantes : x min, x max, y min, y max, z min, z max, ces valeurs permettent de calculer les plans délimitant la surface. (voir figure 3):

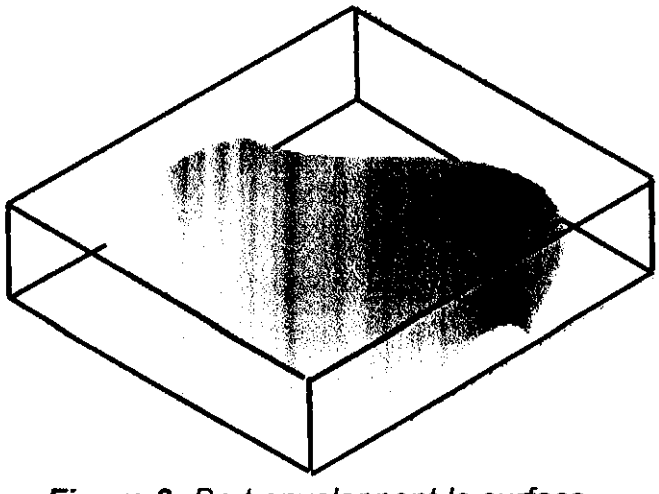


Figure 3: *Brut enveloppant la surface.*

5) Après que nous avons déterminé, les frontière de brut nous pouvons approximer ce brut par un ensemble des triangles. Ces triangles sont générés en déterminant l'ensemble des points qui couvre la face la plus haute du brut, Ces points sont déployés sur un certain nombre de lignes sur cette dernier face, tel que ces points se trouvent à l'extérieur de la surface en lui ajoutant les points d'intersections avec le contour de la surface à usiner (voir figure 4):

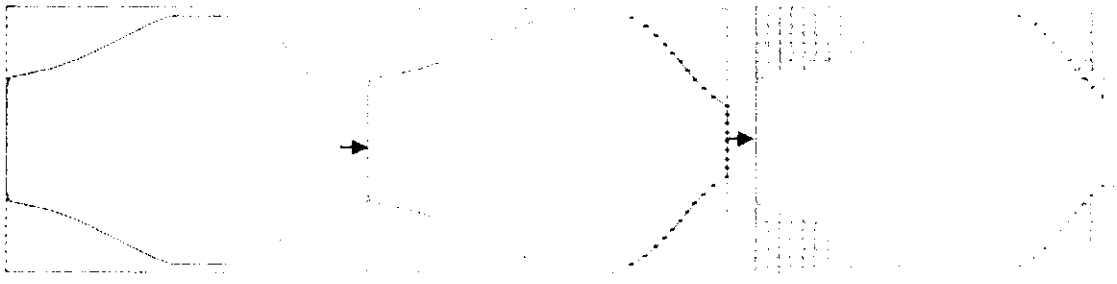


Figure 4: *Approximation du brut par des triangles.*

Une fois les triangles sont générées, nous ajoutons la projection des triangles de surface à la collections des triangles de brut pour obtenir une face de brut totalement triangulée comme montrer ci dessous (voir figure 5):

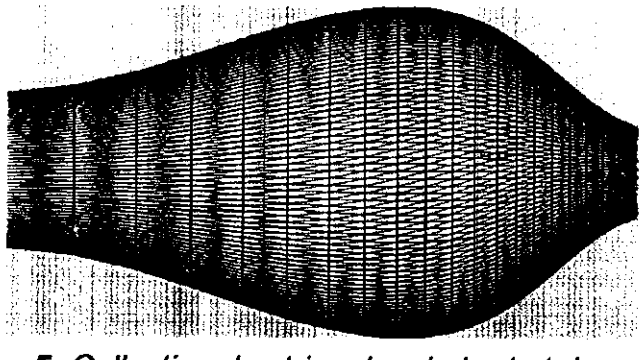


Figure 5: *Collection des triangles de brut et de surface.*

De même, nous avons besoin de représenter les cotés du brut par un ensemble des rectangles (voir figure 6).

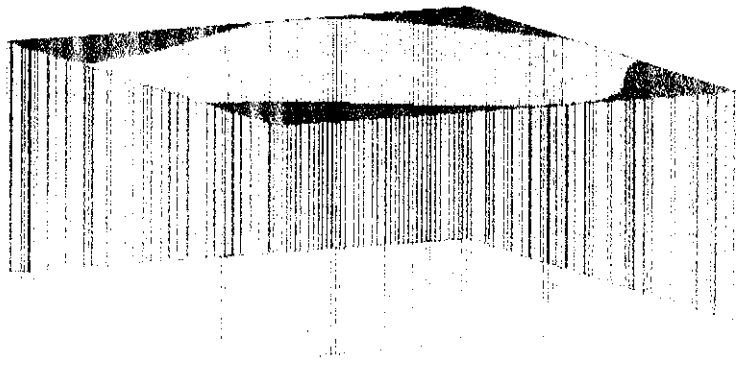
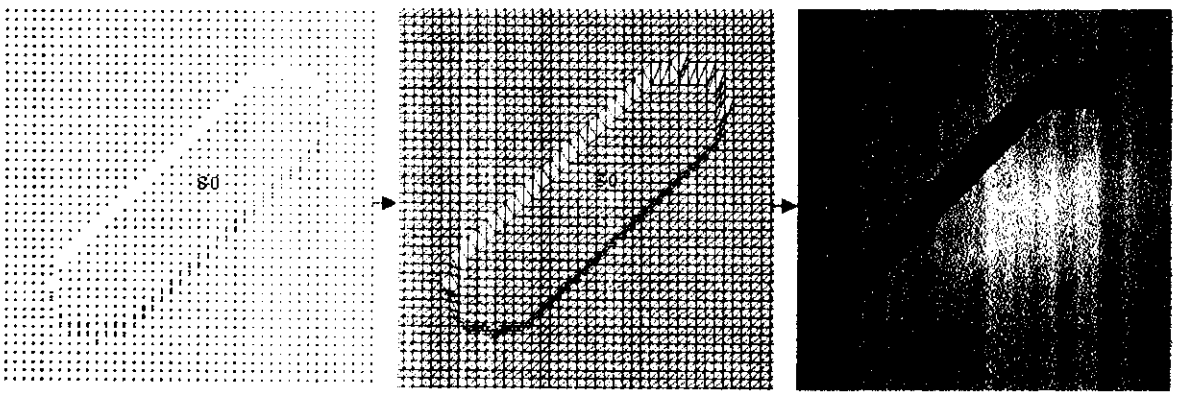


Figure 6: *Approximation des cotés de brut par des polygones.*

6) Finalement, et pour réaliser la simulation de l'opération d'ébauchage et après la récupération de la trajectoire d'outil et après que nous avons fait la projection des points de la surface et les points de brut sur la face supérieure

de brut. Pour chaque deux positions successives, nous pouvons faire la simulation en cherchons les points projetés qui se trouvent dans l'enveloppe d'outil entre ces deux positions. La valeur de cordonnés z est modifiées et sera égale à la valeur de z outil, ce qui nous donne un effet d'abaissements des points et par conséquence un effet d'abaissements de ses triangles, et qui correspond à la simulation de l'ébauchage comme montré ci-dessous (voir figure 7).



***a.** Points.* ***b.** Triangles.* ***c.** rendu*

***Figure 7:** Différentes étapes de réalisation de la simulation.*

2. Deuxième objectif :

Pour déterminer les points qui doivent être mis à jour pendant la simulation de l'opération d'ébauchage, la première idée qui vient est le parcours de tous les points et la modification de la valeur de z_usinage des points qui vérifient le test. Donc, pour chaque deux positions nous somme obligé de parcourir tout les points pour déterminer les points concerner par la mise à jour, ce qui prend un temps supplémentaire inutilement. Pour augmenter la vitesse de simulation il faut chercher une autre solution.

La solution de ce problème est l'utilisation des régions des points. Donc, au lieu de parcourir tout les points, nous parcouront seulement un certain nombre de points en déterminant pour chaque deux positions de trajet d'outil les régions qui doivent être parcourus, puis nous parcourons seulement les points de ces régions, ce qui entraîne à l'augmentation de la vitesse de simulation. Les régions sont crées en indiquant le nombre des régions suivant x et le nombre des régions suivant y par la suite il y a une localisation des points de chaque région (voir figure 8).

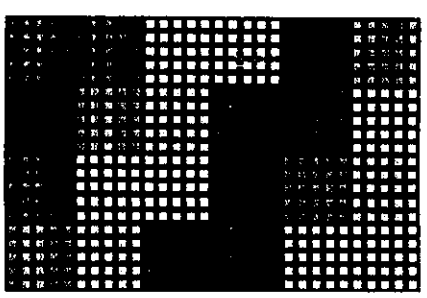


***Figure 8:** Régions d'une surface plane.*

3. Troisième objectif :

Pour la phase de vérification, nous donnons à l'utilisateur la possibilité d'introduite la valeur de tolérance qu'il veut (précision voulue). En fonction de cette tolérance, nous vérifions pour chaque point que nous voulons la mise à jour s'il se trouve en dessus de tolérance donc excès de matière ou en dessous de tolérance donc une interférence ou dans l'intervalle de tolérance pour un cas normal. Selon les cas, les points seront représentés par l'une des 3 couleurs bleues pour l'excès de matière, rouge pour l'interférence et vertes pour un cas normal. La vérification de tolérance est faite pour la surface et pour le brut.

4. Quatrième objectif :

Pour corriger la trajectoire, il faut chercher les positions d'outil qui posent des interférences et les éliminer.

5. Cinquième objectif :

Pour adapter la vitesse d'avance d'outil en fonction du volume enlever, il faut d'abord calculer le volume enlevé entre chaque deux positions .Ce volume est calculé après la détermination des points qui doivent être abaissés et par conséquence, les triangles à abaisser. Pour chaque triangle touché par l'enveloppe d'outil entre deux positions successives, 3 cas sont possibles soit un, deux ou trois points sont abaissés. Ces 3 cas sont présentés dans l'annexe A:

En faisant la somme des volumes de tous les triangles concernés, nous obtenons le volume enlevé entre chaque deux positions successives. En fonction de ce volume, nous allons adapter la vitesse d'avance de l'outil entre ces deux positions.

La vitesse optimale d'avance de l'outil entre deux positions est donnée par la formule suivant [31], [32], [33], [34]:

$$V_{optimale} = D/T_{optimale} \qquad (1)$$

Telle que :

D : est la distance entre deux positions.
$T_{optimale}$: est le temps optimum de déplacement de l'outil entre deux positions.

Le temps optimum de déplacement de l'outil entre deux positions est donné par la formule suivante :

$$T_{optimale} = Volume/MMR \qquad (2)$$

Telle que :

Volume : est le volume enlevé entre deux position.
MMR : est le taux d'enlèvement de la matière.

MMR est donné par la formule suivante :

$$MMR = P/K \quad (3)$$

Telle que :

P : est la puissance d'usinage.
K : est un constant qui dépend de la matière de la pièce à usiner.

P est donné par la formule suivante :

$$P = F_{\max}/V_{coupe} \quad (4)$$

Telle que :

F_{max} : est l'effort de coupe.
V_{coupe} : est la vitesse de coupe.

V_{coupe} est donné par la formule suivante :

$$V_{coupe} = (\Pi \times D \times N)/1000 \quad (5)$$

Telle que :

D : est le diamètre d'outil.
N : est la fréquence de rotation de l'outil.

F_{max} est donné par la formule suivante :

$$F_{\max} = (3 \times Fleche \times E \times I)/L^3 \quad (6)$$

Telle que :

Flèche : est une constant égal à 0.5 pour l'ébauchage.
E : est le module d'élasticité de la matière d'outil et il est égal à 2.1 x e5.
I : est le moment d'inertie.
L : est la longueur d'outil.

I est donné par la formule suivant :

$$I = (D_{effectif}^4 \times \Pi)/64 \quad (7)$$

Telle que :

$D_{effectif}$: est le diamètre effectif de l'outil.

$D_{effectif}$ est donné par la formule suivante :

$$D_{effectif} = D \times 0.8 \quad (8)$$

Telle que :
D : est le diamètre d'outil.

Les différentes étapes de l'adaptation sont montrées dans l'organigramme de la figure 9.

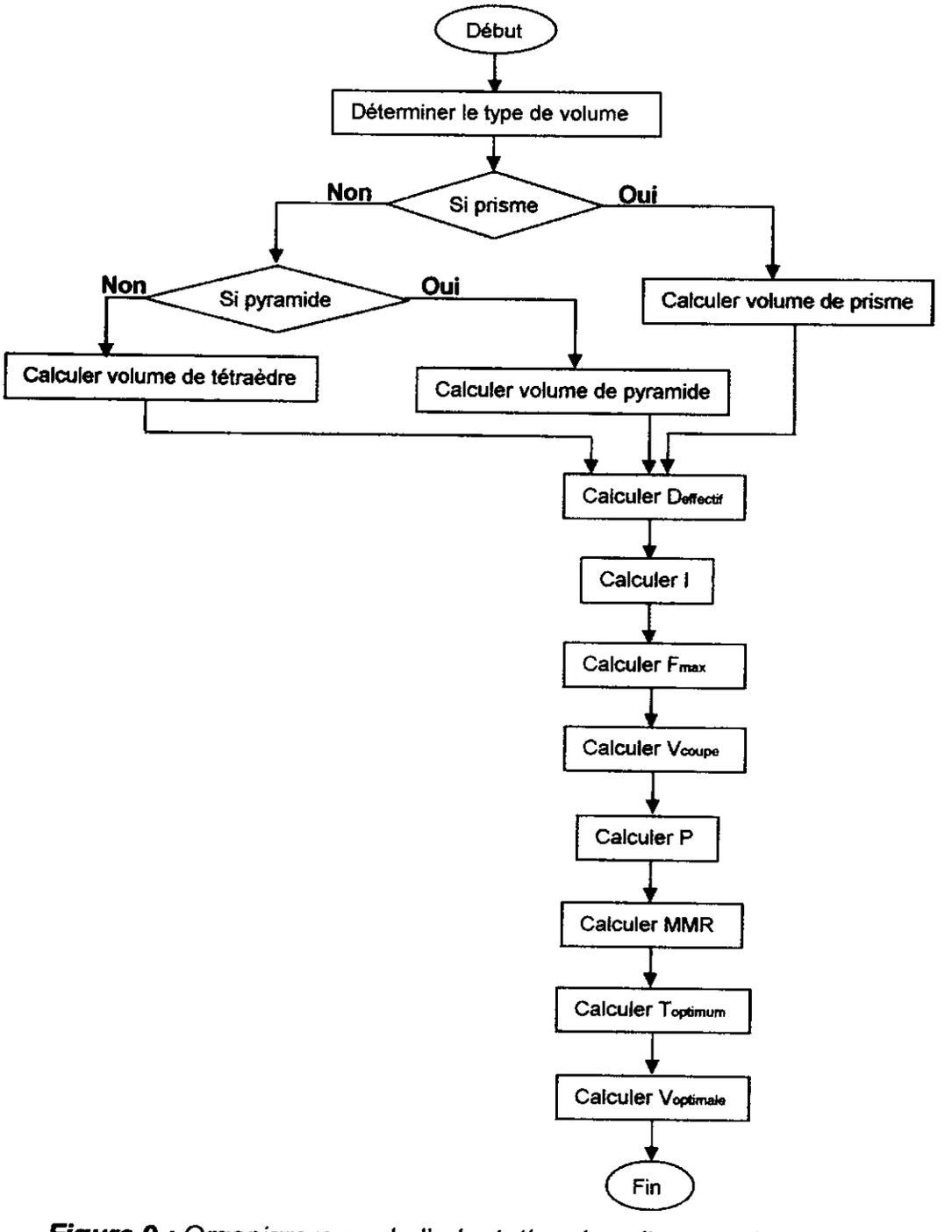


***Figure 9** : Organigramme de l'adaptation des vitesses d'avances.*

6. Sixième objectif :

Pour générer le programme d'usinage « G-code », nous prenons en compte :

- Les nouvelles vitesses d'avance adaptées en utilisant la commande F pour définir la valeur de la vitesse a donné pour chaque déplacement.
- La trajectoire corrigée en utilisant les commandes G00 et G01 pour donner les nouvelles positions à suivre.

III.3. Modélisation de l'application en UML :

Pour modéliser et concevoir notre système nous avons utilisés cinq diagrammes de l'ensemble de diagrammes du langage UML.

III.3.1. Diagramme de cas d'utilisation :

Pour représenter les besoins de notre système, nous avons utilisés le diagramme de cas d'utilisation, qu'est un moyen pour représenter les besoins en UML et qui permet de faire une organisation générale pour l'utilisation de notre système par ses acteurs et la description du comportement du système du point de vue de son utilisateur qui correspond à la manière spécifique d'utilisation de système.

Dans la figure 10, nous avons représentés les principes cas d'utilisation de notre système qui commence par « *l'ouverture d'une surface* » que nous voulons usiner en réaction à l'action d'un acteur qu'est l'utilisateur du système et qui doit aussi intervenir dans le cas d'utilisation « *génération de trajectoire* » s'il n'est pas été déjà généré auparavant avec une intervention d'un deuxième acteur dans cet cas d'utilisation qu'est un autre système c'est le « *le système de calcul* » qui réalise les différentes opérations de calcul qui permet de générer la trajectoire d'usinage.

Ensuite l'utilisateur peut « *générer les points de surface* » qui servent à la « *génération des triangles* ». À partir de ces triangles, l'utilisateur peut « *optimiser leur nombre* » en fonction des paramètres qu'il introduit au système et par conséquence l'optimisation du nombre des points qui approxime la surface.

En utilisant ces points, l'utilisateur peut « *générer les régions* » avec le nombre qu'il veut, nombre des régions par ligne et nombre des régions par colonne. Ces régions seront utilisées dans l'étape de la « *simulation de l'opération d'ébauchage* » déclenchée par l'action de l'utilisateur en utilisant la trajectoire générée.

Ensuite, il y a le cas d'utilisation de la « *vérification de tolérance* » qui se réalise en fonction de tolérance fixée par l'utilisateur et le cas d'utilisation de « *Correction de la trajectoire* » pour l'élimination des interférences puis, l'avant dernier cas « Adaptation des vitesses d'avance » en fonction de volume calculé et les paramètres choisit qui sont calculés par d'autres activités intégrées dans cette cas d'utilisation, le dernier cas d'utilisation est le cas de « *Régénération de programme d'usinage* » pour générer un programme d'usinage G-Code qui correspond à la trajectoire corrigée et adaptée.

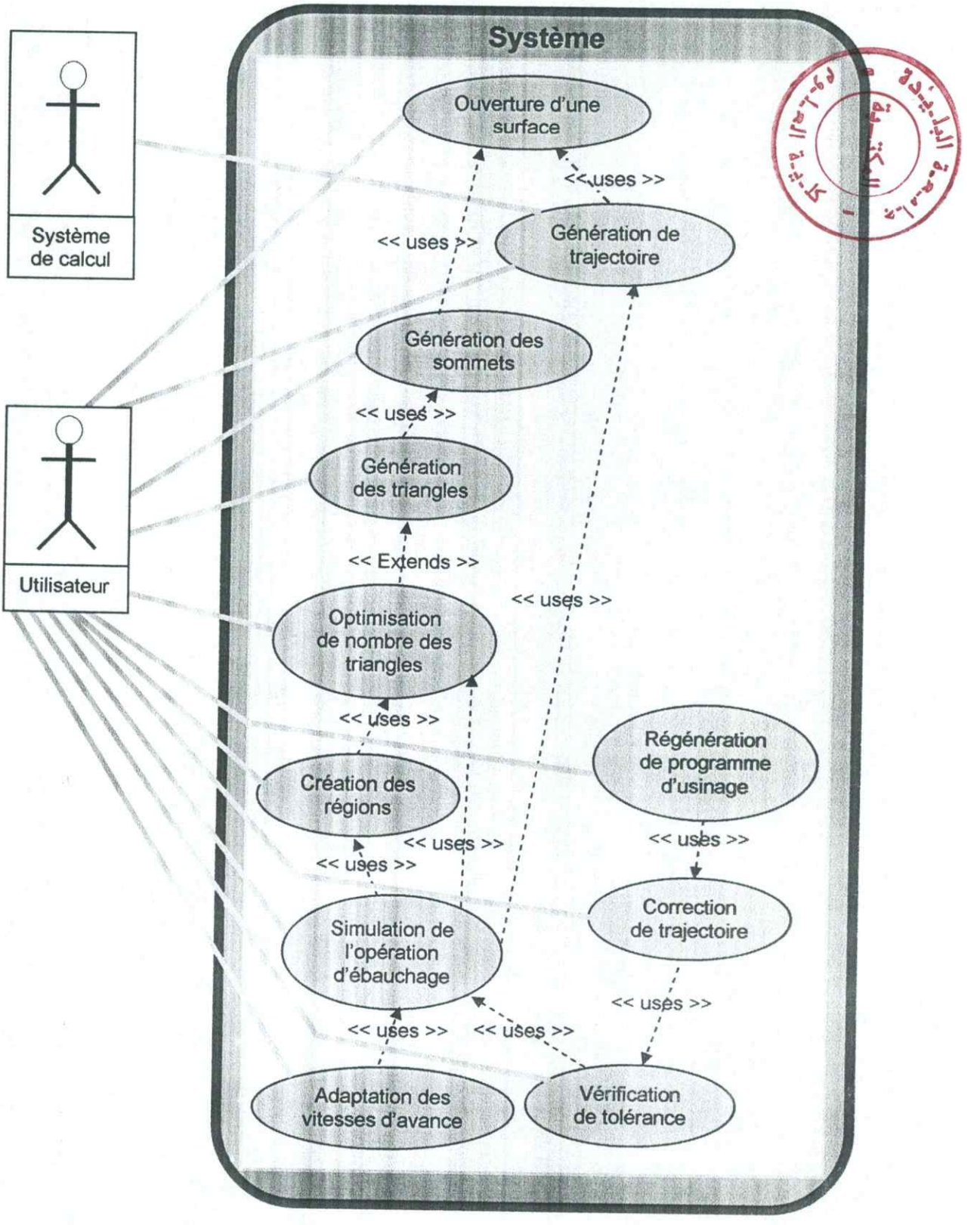


Figure 10: *Diagramme de cas d'utilisation.*

III.3.2. Diagramme de classes :

Pour représenter la structure de notre système, nous devons le modéliser par le diagramme de classe qui permet de représenté la structuration de notre système sous forme des classes, avec leurs attributs et opérations intégrées et des relations

entre ses classes avec ses multiplicités et ses types, telle que les classes sont une description abstraite des ensembles d'objets que nous avons utilisés dans notre système et les relations sont des associations entre les classes d'objets qui elle même peut être une classe dite classe d'association (voir figure 11).

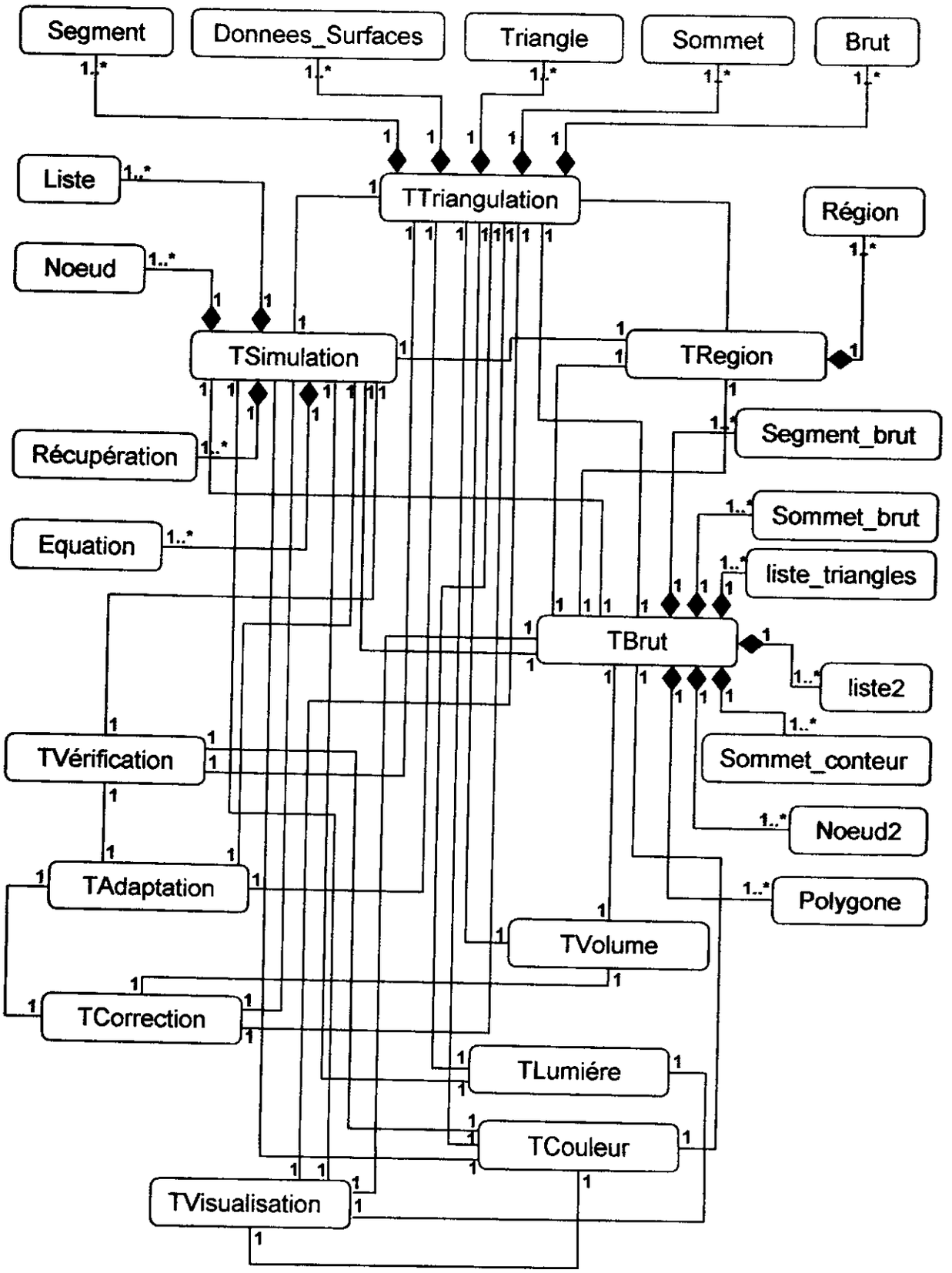


Figure 11: *Diagramme de classes.*

La figure 11 représente le diagramme de classe, de notre système qui comporte toutes les classes que nous avons implémentées avec ses relations. Nous avons représentés les classes sans ses attributs et ses fonctions pour réduire la taille de digramme.

- **La classe de triangulation (TTriangulation):**

Cette classe est spécialement destinée à la génération des sommets et des triangles. Elle comporte plusieurs attributs et plusieurs fonctions qui aident à cette génération (voir figure 12).

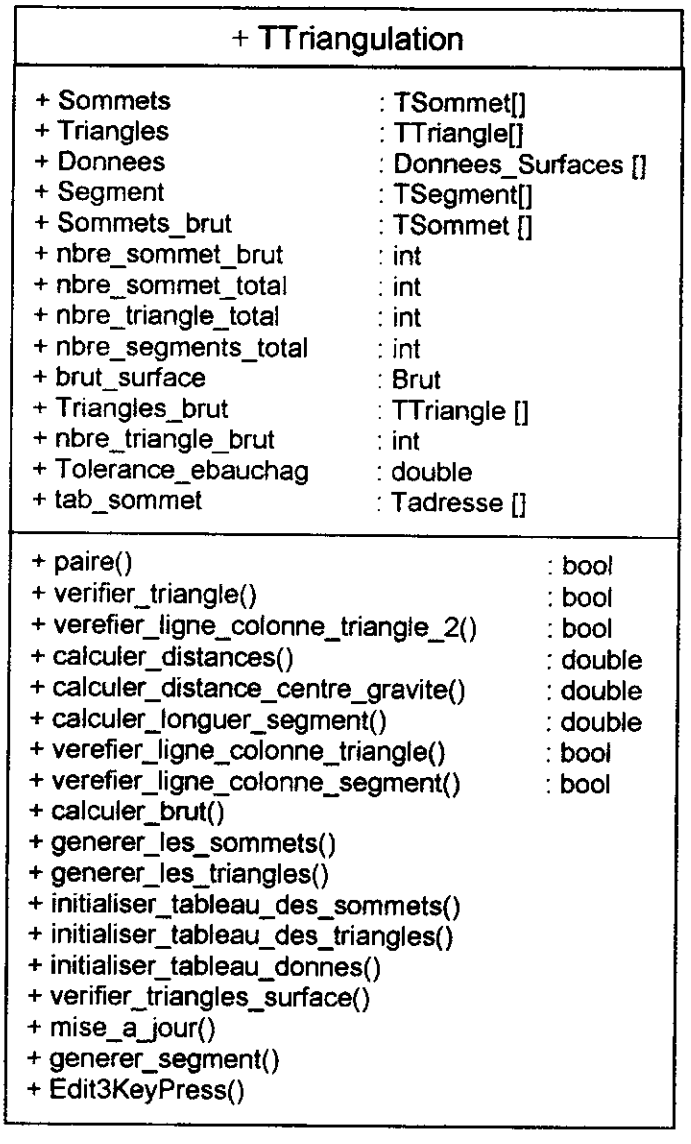


Figure 12 : *Classe TTriangulation.*

Parmi les attributs de cette classe, nous avons :

Sommets[] : est un tableau de type *TSommet* qui permet de stocker tout les sommets que nous avons générés pour la surface.

Sommets_brut[] : est un tableau de type *TSommet* qui permet de stocké tout les sommets que nous avons générés pour les parties du brut.

Triangles[] : est un tableau de type *TTriangle* qui permet de stocker tout les triangles que nous avons générés pour la surface.

Triangles_brut[] : est un tableau de type *TTriangle* qui permet de stocker tout les triangles que nous avons générés pour les parties de brut.

Donnees[] : est un tableau de type *Donnees_Surfaceset* qui est utile dans le cas ou nous ouvrons plusieurs surfaces simultanément pour stocker des informations spéciales pour chaque surface. Donc il permet de généraliser les opérations réalisées sur une seule surface pour plusieurs surfaces.

nbre_sommet_total : de type *int (entier)*, utilisé pour stocker le nombre de sommets générés pour la surface.

nbre_sommet_brut : de type *int (entier)*, utilisé pour stocker le nombre de sommets générés pour les parties de brut.

nbre_triangle_total : de type *int (entier)*, utilisé pour stocker le nombre de triangles générés pour la surface.

nbre_triangle_brut : de type *int (entier)*, utilisé pour stocker le nombre de triangles générés pour les parties de brut.

brut_surface : de type *Brut* qui permet de définir les extrémités de brut.

Maintenant, nous allons décrire quelques fonctions essentielles de cette classe avec les algorithmes associés.

Void initialiser_tableau_donnes() : permet d'initialiser le tableau des données des surfaces(*Donnees[ ]*) par le nombre des lignes et des colonnes, introduit par l'utilisateur.

Void initialiser_tableau_des_sommets() : permet d'initialiser la taille de tableau des sommets par la valeur (ligne+1) x (colonne+1) telle que ligne et colonne sont introduit, par l'utilisateur.

initialiser_tableau_des_triangles() : permet d'initialiser la taille de tableau des triangles par la valeur (ligne x colonne x 2) telle que ligne et colonne sont introduit par l'utilisateur.

generer_les_sommets() : c'est la fonction qui va générer tous les sommets de surface et les mette dans la table des sommets.

Void generer_les_triangles() : utilise les sommets généré, pour générer les triangles de surface telle que pour chaque itération elle génère deux triangles. L'algorithme de cette fonction est le suivant :

```
Début
    Pour chaque surface
      Pour i=0 à ligne  nbr ligne
        Pour j=0 à colonne / nbr colonne
          Faire
          Pour le premier triangle
```

```
                sommet1= i + j x (colonne+1) + nbr_sommets_avant ;
                sommet2= i + (colonne+1) + j x (colonne+1) + nbr_sommets_avant;
                sommet3= i + (colonne+2) + j x (colonne+1) + nbr_sommets_avant;

              Pour le deuxième triangle
                sommet1= i + j x (colonne+1) + nbr_sommets_avant ;
                sommet2= i  + 1 + j x (colonne+1) + nbr_sommets_avant ;
                sommet3= i + (colonne+2) + j x (colonne+1) + nbr_sommets_avant;
              Fait
        Fin
```

Void calculer_brut() : permet de déterminer les extrémités de brut suivant.

Bool verefier_triangles_surface (int num_surface, double longeur_maximal, *double hauteur*_maximal*)* : permet de vérifier pour chaque triangle s'il satisfait certaines contraintes introduites par l'utilisateur. Ces contraintes sont la longueur maximale du segment des triangles et la distance maximale entre la surface théorique et la surface approximée par des triangles.

Int adaptation_ligne_triangle(int num_surface,int ligne_min,int ligne_max, int colonne) : se charge de l'optimisation du nombre des lignes pour qu'il soit le plus petit possible, telle que en générant les triangles suivant ce nombre. Les contraintes précédemment citées seront vérifiées. Cette fonction fait appel à une autre fonction *verifier_ligne_colonne_segment(ligne, num_surface, colonne)* qui réalise la même vérification de la fonction précédente mais cette fois ci en fonction des contraintes et du nombre des lignes et des colonnes nouvellement calculées. L'algorithme de cette fonction est le suivant :

```
        Début
          ligne = (ligne_min + ligne_max)/2
          Si (verefier_ligne_colonne_segment(ligne, num_surface, colonne))
           Alors
           Si (! verefier_ligne_colonne_segment(ligne-1, num_surface, colonne))
              Alors
                 Résultat=ligne
              Si non
                 adaptation_ligne_segment(num_surface,ligne_min, ligne,colonne)
           Fin si

           Si non
              Si (verefier_ligne_colonne_segment(ligne+1, num_surface, colonne))
                 Alors
                     Résultat=ligne+1
                 Si non
                   adaptation_ligne_segment(num_surface,lig,ligne_max, colonne)
              fin si
        Fin
```

Int adaptation_colonne_triangle(int num_surface,int colonne_min,int colonne _max,int colonne) : réalise le même travail que la fonction précédente en optimisant le nombre des colonnes en fonction des critères introduit, par l'utilisateur.

La classe de triangulation comporte plusieurs sous classes, ces sous classes sont montrées dans la figure 13.

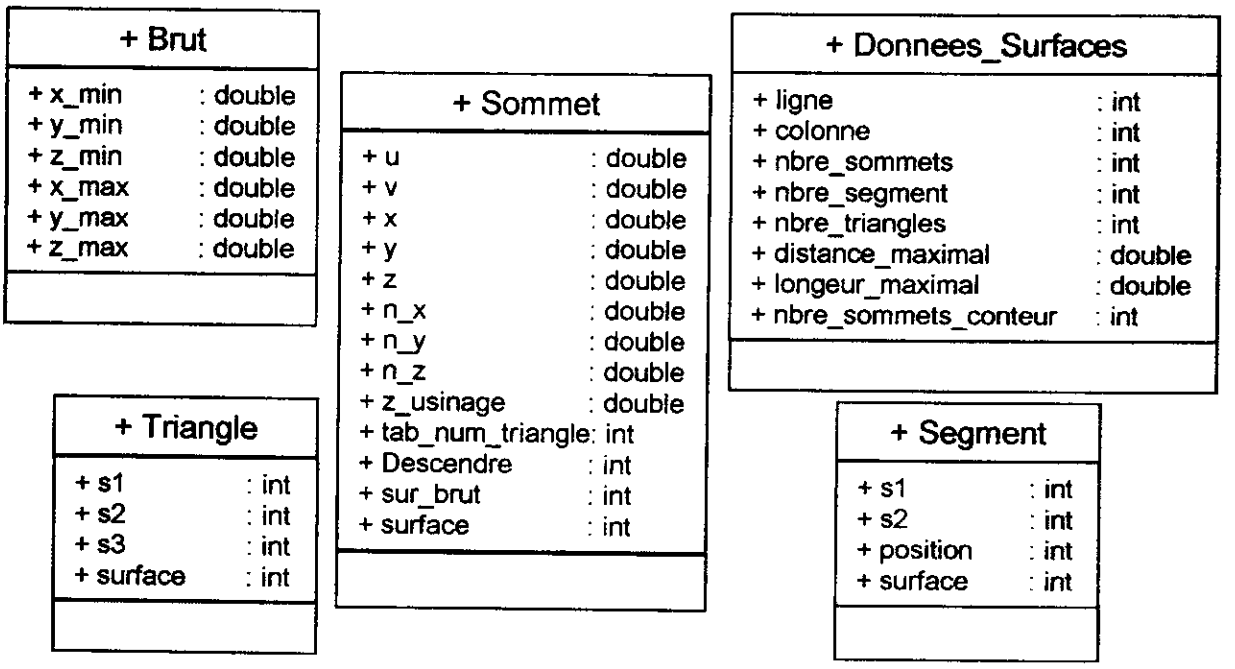


***Figure 13** : Sous classes de la classe de triangulation.*

- **La classe de création des régions (TRegion) :**

La classe TRegion permet de générer les régions qui seront utilisés par la suite. Elle permet de distribuer les sommets de surface et sommets de brut en des régions pour réduire le temps de recherche des sommets pendant la simulation (voir figure 14).

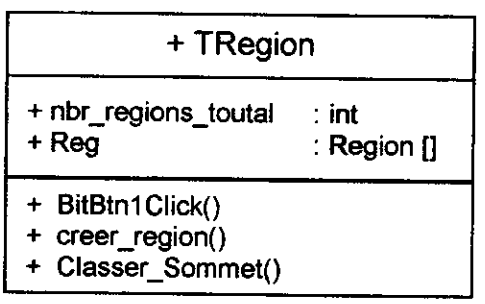


***Figure 14** : Classe TRegion.*

Les principaux attributs sont :

Reg[] : est un tableau de type *Region* permet de stocké les régions crées.

nbr_regions_toutal : pour stocker le nombre des régions.

Les principales fonctions sont :

La fonction creer_region() : se charge de la création de ces régions en calculant leurs limites suivant x et suivant y, son algorithme est le suivant :

```
Début
   dis_x = Xmax_brut – Xmax_brut ;
   dis_y = Ymax_brut – Ymin_brut ;
   pas_x =dis_x / Nbr_Reg_lig ;
   pas_y =dis_y / Nbr_Reg_col ;
   Ind_Reg=0 ;
   Pour i=0 à Nbr_Reg_col
    Faire
      Pour j=0 à Nbr_Reg_lig
         Faire
               Xmin _Reg[Ind_Reg] = j*pas_x + Xmin_brut ;
               Xmax_Reg[Ind_Reg] = j*pas_x + pas_x + Xmin _brut ;
               Ymin _Reg[Ind_Reg] = j*pas_y + Ymin_brut ;
               Ymax_Reg[Ind_Reg] = j*pas_y + pas_y + Ymin _brut ;
               Initialiser le liste des sommets de Reg[Ind_Reg] à nulle ;
         Fait ;
    Fait ;
Fin
```

La fonction *Classer_Sommet(TSommet *sommet, int Reg_lig, int Reg_col, double pas_x, double pas_y)* se charge de la remplissage des régions précédemment crées par les adresses des sommets de surface et les sommets de brut. L'algorithme de cette fonction est le suivant :

```
Début
   Pour tout les sommets de surface et de brut
      Faire
         i =( Xsommet - Xmin_brut ) / pas_x ;
         j =( Ysommet - Ymin_brut ) / pas_y ;
         Indice_Reg = Reg_lig*j + i
         Insérer la sommet à la fin de la liste de Reg[Indice_Reg];
      Fait ;
Fin
```

La classe de création des régions comporte une sous classe, ce sou classe est montré dans la figure 15.

+ Region	
+ x_min	: double
+ y_min	: double
+ x_max	: double
+ y_max	: double
+ deja_uti	: bool
+ nbre_sommet	: int
+ liste	: Tliste
+ prec	: &TNoeud
+ cour	: &TNoeud

Figure 15 : *Sous classe Région de la classe TRegion.*

- **La classe de simulation (TSimulation) :**

C'est la classe principale de simulation de l'opération d'ébauchage, elle comporte plusieurs attributs et fonctions qui se chargent de la réalisation de cette opération (voir figure 16).

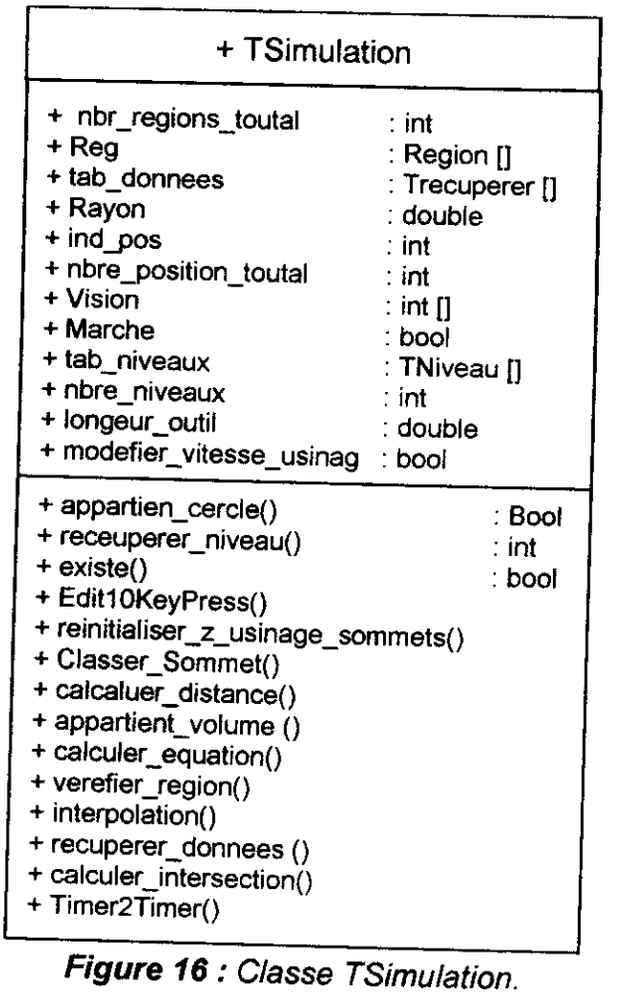


Figure 16 : *Classe TSimulation.*

Ses principaux attributs sont :

tab_donnees[] : est une table de type Trecuperer pour le stockage des informations concernant la trajectoire générée. Parmi ces informations, il y a les positions d'outil et les vitesses d'avance correspondent ainsi que le rayon d'outil.

ind_pos : de type int qui indique la position courante de l'outil dans la trajectoire en simulation.

nbre_position_toutal : de type int utilisé pour stocker le nombre des positions qui existe dans la trajectoire.

deux_position : de type TDeux_position, il permet d'enregistré pour chaque déplacement ses deux positions successives.

Vision[] : Ce tableau est de type int permet d'enregistrer les paramètres de la lumière appliquée sur les objets de la scène de simulation.

tab_niveaux[] : est un tableau de type TNiveau qui permet de stocker les différents niveaux de l'ébauchage en simulation ainsi que la couleur de chaque niveau.

nbre_niveaux : de type int utilisé pour stocker le nombre des niveaux trouvés.

Ces attributs sont utilisés par les différentes fonctions de cette classe pour réaliser la simulation de l'opération d'ébauchage. Parmi ces fonctions, nous pouvons décrire quelques fonctions avec ses algorithmes :

*Bool appartien_cercle(TSommet *s1,double x0,double y0,double rayon)* : est utiliser pour vérifier si un sommet donné appartient au cercle qui correspond à l'outil, ce cercle est défini par le centre *(x0,y0)* et le rayon *R* dans le plans (x,y).

*bool existe(double *tab,int taille,double x)* : Indique si une variable *x* existe dans le tableau *tab* de taille *taille*.

reinitialiser_z_usinage_sommets() : permet d'initialiser le paramètre z_usinage de chaque sommet de la surface et de brut par la valeur z maximal de brut.

double calcaluer_distance_3d(double x1,double y1,double z1,double x2,double y2,double z2) : permet de calculer la distance entre deux sommets.

*Void calculer_equation(double x1,double y1,double x2,double y2, TEquation *E)* : permet de déterminer l'équation *E (Y = Teng_Alpha * X + b)*,d'un segment défini par les deux points(p_1(x1,y1),p_2(x2,y2)).

int receuperer_niveau(int taille, tab_donnees[], tab[]) : est utiliser pour récupérer tout les niveaux d'ébauchage contenu dans la trajectoire d'outil qu'est un ensemble des positions stockés dans la table *tab_donnees[]* de taille *taille*. Son algorithme est le suivant :

```
Début
    Indice =0 ;
    Pour i=0 à taille
        Faire
            Si ( z_ tab_donnees[i]) <= zmax_brut )
               Alors
                    Si (Non existe(tab,indice, z_tab_donnees[i] )
                       Alors
                            tab[indice+1] = z_tab_donnees[i] ;
                            Indice = indice + 1 ;
                       Fin si ;
                Fin si ;
        Fait ;
        Résultat = indice ;
Fin
```

Void calculer_intersection(double x_usinage,double y_usinage,double z_usinage, double rayon) : permet de déterminer toutes les régions qui sont touchées par le cercle de l'extrémité active d'outil de centre (*x_usinage, y_usinage, z_usinage*) et de

rayon « *rayon* » dans la position définis par les même cordonnées de centre d'outil durant l'opération d'ébauchage, les régions sont comprises entre ligne1, ligne2 et colonne1,colonne2 qui doivent être déterminées puis les sommets de chaque région doivent être examinés pour déterminer si nous devons faire une mise à jour de son z-usinage (d'une autre manière si ce sommet doit être abaissé ou non). L'algorithme de cette fonction est le suivant :

```
Début
 dis_x = Xmax_brut - Xmin_brut ;
 dis_y = Ymax_brut - Ymin_brut ;
 pas_x = dis_x / Nbr_Reg_lig ;
 pas_y = dis_y / Nbr_Reg_col ;
 colonne1 = ( (x_usinage - rayon) - Xmin_brut ) / pas_x ;
 Si (colonne1<0)  Alors  colonne1=0
 Fin si ;
 colonne2 = ( (x_usinage + rayon) - Xmin_brut ) / pas_x ;
 Si (colonne2> Nbr_Reg_lig)  Alors  colonne2 =Nbr_Reg_lig-1 ;
 Fin si ;
 ligne1 = ( (y_usinage - rayon) - Ymin_brut ) / pas_y ;
 Si (ligne1 <0)  Alors  ligne1 =0
 Fin si ;
 ligne2 = ( (y_usinage + rayon) - Ymin_brut ) / pas_y ;
 Si (ligne2 > Nbr_Reg_col)  Alors  ligne2 =Nbr_Reg_col-1
 Fin si ;
 Pour i=ligne1 à ligne2
  Faire
    Pour j=colonne1 à colonne2
        Faire
          Num_région = Nbr_Reg_lig x ( i + j ) ;
          parcours = debut_region[Num_région] ;
          Tant que (parcours != Nulle)
                Faire
                  appartient = appartien_cercle(sommet_parcours,
                                x_usinage,y_usinage, rayon) ;
                  Si (appartient=vrai)
                    Alors
                      Si (Zusinage_ sommet_parcours > z-usinage)
                       Alors
                        Zusinage_ sommet_parcours= z_usinage ;
                        Parcours= region[Num_région]_sommet_suivant ;
                       Fin si ;
                      Si non ;
                        Parcours= region[Num_région]_sommet_suivant ;
                      Fin si ;
                Fait ;
        Fait ;
  Fait ;
Fin
```

Bool verefier_region(int Num_Rer,TEquation E) : permet de déterminer s'il existe une intersection entre le segment définit par l'équation *E* et la région numéro ***Num_Rer.*** L'algorithme de cette fonction est le suivant :

```
Début
    Y1 = Xmin_Reg[Num_Rer] x T_alpha_E2 + b_E2;
    Si (y1<Ymax _Reg[Num_Rer] && y1>Ymin _Reg[Num_Rer])
      Alors
          Résultat = vrai ;
          Sortie ;
    Fin si ;

    Y2 = Xmax_Reg[Num_Rer] x T_alpha_E2 + b_E2 ;
    Si (y1<Ymax _Reg[Num_Rer] && y1>Ymin _Reg[Num_Rer])
      Alors
          Résultat = vrai ;
          Sortie ;
      Fin si ;
    X1 = (Ymin _Reg[Num_Rer] – b_E2) / T_alpha _E2;
    Si(X1<Xmax _Reg[Num_Rer] && X1>Xmin _Reg[Num_Rer])
      Alors
          Résultat = vrai ;
          Sortie ;
    Fin si ;
    X2 = (Ymax_Reg[Num_Rer] – b_E2) / T_alpha _E2;
    Si(X2<Xmax _Reg[Num_Rer] && X2>Xmin _Reg[Num_Rer])
      Alors
          Résultat = vrai ;
          Sortie ;
      Fin si ;
Fin
```

Bool appartient_volume (double x,double y,TEquation E1,TEquation E2,TEquation E3,TEquation E4) : sert à déterminer si un sommet (x,y) appartient au rectangle définit par les équations *E1,E2,E3,E4* de ses quatre segment.

Void interpolation(double x1, double y1, double x2, double y2,double z_usinage) : permet de faire l'interpolation de la simulation de l'opération d'ébauchage entre deux positions (x1,y1) et (x2,y2) telle que x2>=x1 et y2>=y1. à partir de ces deux positions, elle détermine l'enveloppe des régions concernées en calculant ligne_region1, ligne_region2, colonne_region1, colonne_region2 et à partir de ses régions elle ne choisit que les régions qui ont des intersections avec les segments seg2,seg3,seg4,seg5 qui représentent le rectangle qui relie les deux positions successive de l'outil et qui sont relié par le segment seg1 puis elle parcours tout les sommets qui appartient à ces régions et chaque sommet qui se trouve dans le rectangle il sera mis à jour. L'algorithme de cette fonction est le suivant :

```
Début
  dis_x = Xmax_brut - Xmin_brut ;
  dis_y = Ymax_brut - Ymin_brut ;
```

```
pas_x = dis_x / Nbr_Reg_lig ;
pas_y = dis_y / Nbr_Reg_col ;
colonne_region1= ( (x1 - rayon) - Xmin_brut ) / pas_x ;
Si (colonne1<0)  Alors  colonne1=0
Fin si ;
colonne_region2= ( (x2 + rayon) - Xmin_brut ) / pas_x ;
Si (colonne2> Nbr_Reg_lig)  Alors  colonne2 =Nbr_Reg_lig-1
Fin si ;
ligne_region1= ( (y1 - rayon) - Ymin_brut ) / pas_y ;
Si (ligne1 <0)  Alors  ligne1 =0
Fin si ;
ligne_region2= ( (y2 + rayon) - Ymin_brut ) / pas_y ;
Si (ligne2 > Nbr_Reg_col)  Alors  ligne2 =Nbr_Reg_col-1 ;
Fin si ;
Si(x1=x2)  alors
                traiter cas spéciale1 ;
                sortie ;
            Fin si ;
distance=calcaluer_distance(x1,y1,x2,y2);
sin_alpha=(y2-y1)/distance;
cos_alpha=(x2-x1)/distance;
calculer_equation(x1, y1, x2, y2, seg1);
Si (cos_alpha=1)  alors
                    traiter cas spéciale2 ;
                    sortie ;
                Fin si
calculer_equation(x1+(sin_alpha*Rayon),y1-(cos_alpha* Rayon),x2+
                (sin_alpha*Rayon),y2-(cos_alpha*Rayon),
                 segment_A) ;
calculer_equation(x1-(sin_alpha*Rayon),y1+(cos_alpha*Rayon),x2-
                (sin_alpha*Rayon),y2+(cos_alpha*Rayon),
                 segment_B) ;
calculer_equation(x1, y1, x1+(sin_alpha*Rayon), y1-(cos_alpha*Rayon) ,
                segment_C);
calculer_equation(x2, y2, x2+(sin_alpha*Rayon), y2-(cos_alpha*Rayon),
                segment_D);
Si ( sin_alpha>0 et cos_alpha>0 )
  alors
      seg2 = segment_A;
      seg3 = segment_B;
      seg4 = segment_C;
      seg5 = segment_D ;
  Fin si ;
Si ( sin_alpha>0 et cos_alpha<0 )
  Alors
      seg2 = segment_B ;
      seg3 = segment_A ;
      seg4 = segment_C ;
      seg5 = segment_D ;
```

```
            Fin si
        Si ( sin_alpha<0 et cos_alpha<0 )
            alors
                seg2 = segment_B;
                seg3 = segment_A;
                seg4 = segment_D;
                seg5 = segment_C;
            Fin si ;
        Si (sin_alpha<0 et cos_alpha>0 )
            alors
                seg2 = segment_A;
                seg3 = segment_B;
                seg4 = segment_D;
                seg5 = segment_C ;
            Fin si ;
        Pour i=ligne_region1 à ligne_region2
            Faire
                Pour j= colonne_region1 à colonne_region2
                    Faire
                        Num_Rer=Nbr_Reg_lig*i+j ;
                        Appartient1=appartient_volume(Reg[Num_Rer].x_min,
                                    Reg[Num_Rer].y_min,seg2,seg3,seg4,seg5) ;
                        Appartient2=appartient_volume(Reg[Num_Rer].x_max,
                                    Reg[Num_Rer].y_min,seg2,seg3,seg4,seg5) ;
                        Appartient3=appartient_volume(Reg[Num_Rer].x_max,
                                    Reg[Num_Rer].y_max,seg2,seg3,seg4,seg5);
                        Appartient4=appartient_volume(Reg[Num_Rer].x_min,
                                    Reg[Num_Rer].y_max,seg2,seg3,seg4,seg5);
                        Si (Appartient1 ou Appartient2 ou Appartient3 ou Appartient4)
                            Alors
                                parc=debut_Reg[Num_Rer] ;
                                Tant que (parc != Nulle)
                                    Faire
                                        appartient = appartient_volume(x_sommet_parc,
                                                    y_sommet_parc seg2,seg3,seg4,seg5) ;
                                        Si (appartient)
                                            Alors
                                                Si (Zusinage_sommet_parc>z_usinage)
                                                    Alors
                                                        Zusinage_sommet_parc=z_usinage ;
                                                    Fin si ;
                                            Fin si ;
                                        parc=suivant_parc ;
                                    Fait ;
                            Fin si ;
                    Fait ;
            Fait ;
    Fin
```

La classe de simulation comporte plusieurs sous classes, ces sous classes sont montrées dans la figure 17.

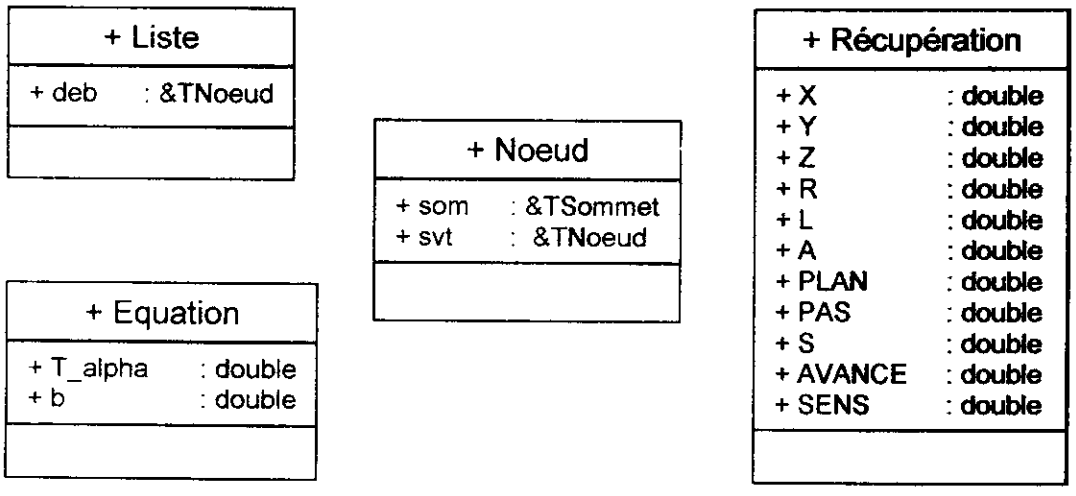


***Figure 17** : Sous classes de la classe de simulation.*

- **La classe de brut (TBrut) :**

C'est la classe qui concerne le brut. Elle permet de générer les sommets de brut, triangles de brut et les polygones pour les autres cotés de brut.

Ces triangles seront utilisés pour simuler l'enlèvement de la matière de brut en calculant d'abords les points d'intersection d'un certain nombres des lignes sur le plan supérieur de brut avec le contour de la projection de surface sur cet plan, ce contour sera la clôture qui sépare les triangles de brut de celui de la projection de surface (voir figure 18).

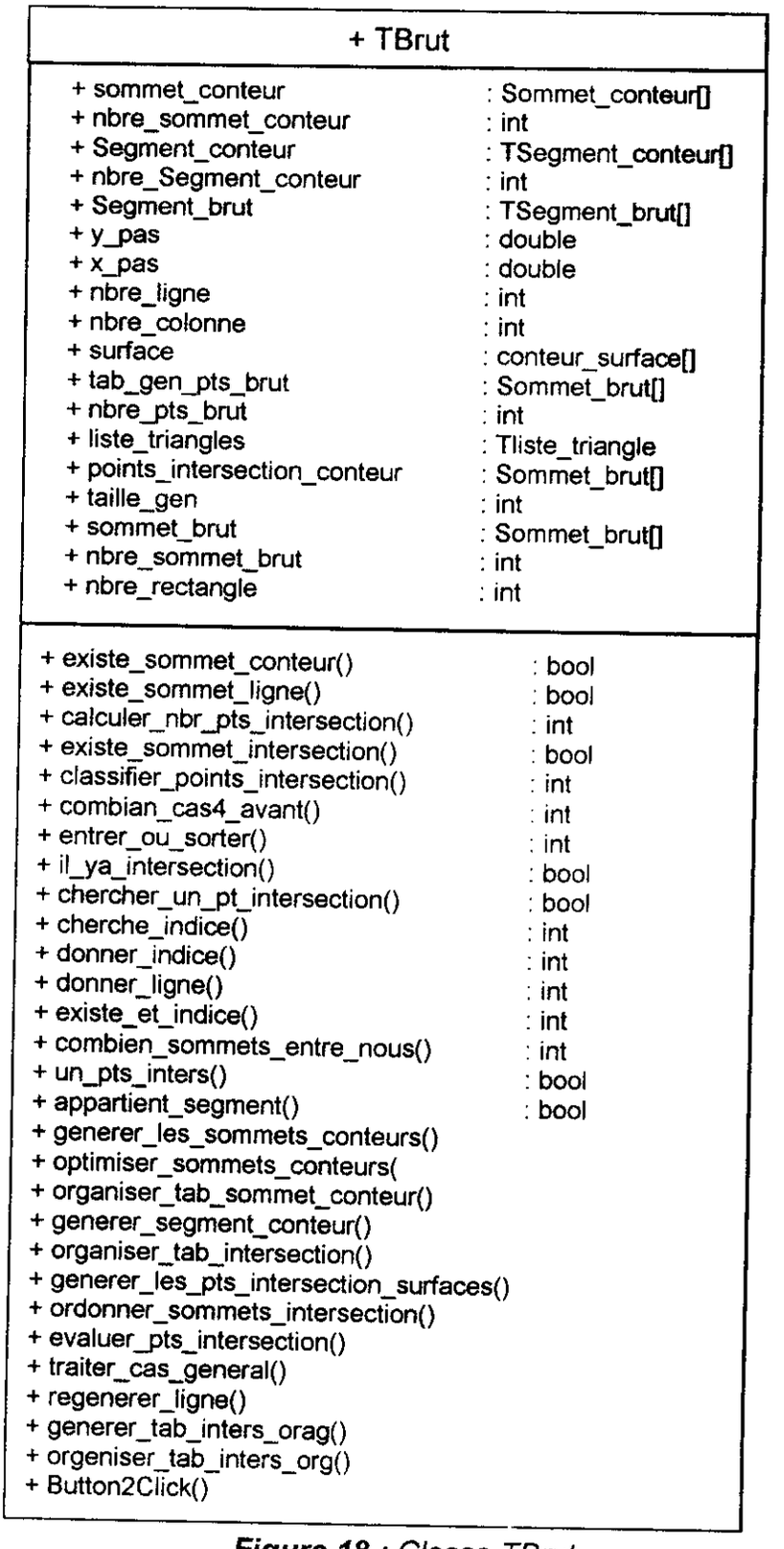


***Figure 18** : Classe TBrut.*

Les attributs essentiels de cette classe sont les suivants :

sommet_conteur[] : est un tableau de type *Sommet_conteur* qui contient les sommets de contour de la projection de surface sur le plan supérieur de brut.

nbre_sommet_conteur : de type *int* utilisé pour stocker le nombre des sommets de contour.

Segment_conteur[] : est un tableau de type *TSegment_conteur* utilisé pour stocker les segments qui relié les sommets de contour.

nbre_Segment_conteur : de type *int* utilisé pour stocker le nombre des segments de contour.

y_pas : de type réel contient la distance qui sépare deux sommets successives de brut suivant l'axe des y.

x_pas : de type réel contient la distance qui sépare deux sommets successives de brut suivant l'axe des x.

nbre_ligne, nbre_colonne : sont de type entier permettent de fixer le nombre de lignes et de colonnes de brut.

points_intersection_conteur[] : est un tableau de type *Sommet_brut* contient les points d'intersection des lignes avec les segments de contour.

sommet_brut[] : est un tableau de tout les sommets générés de brut.

liste_triangles : de type *Tliste_triangle*, est une liste chaînée qui contient tout les triangles générés de brut.

Nbre_triangles : de type entier stocke le nombre des triangles dans la liste.

Les fonctions principales de cette classe sont les suivant :

void generer_les_sommets_conteurs(int num_surface) : permet de générer les sommets de contours de la projection de surface et les mètres dans un tableau, tel que ces points sont d'abords générés dans le plan paramétrique puis seront transformés dans l'espace pour être utiliser dans la construction de conteurs. Son algorithme est le suivant :

```
Début
   nbre_sommet_conteur=(nbre_lig_brut + nbre_col_brut) x 2 ;
   Pour (i=0 à nbre_col_brut)
      Faire
         générer le point dan le plan paramétrique pour la première ligne ;
         transformer le point dans le plan cartésien ;
      Fait ;
   Pour (i=1 à nbre_lig_brut)
      Faire
         générer le point dan le plan paramétrique pour la première colonne ;
         transformer le point dans le plan cartésien ;
      Fait ;
   Pour (i=1 à nbre_col_brut)
      Faire
         générer le point dan le plan paramétrique pour la dernière colonne ;
         transformer le point dans le plan cartésien ;
      Fait ;
   Pour (i=1 à nbre_lig_brut)
```

```
            Faire
               générer le point dan le plan paramétrique pour la dernière ligne ;
               transformer le point dans le plan cartésien ;
            Fait ;
      Fin
```

void generer_segment_conteur(int num_surface) : permet de générer les segments de contour à partir des sommets précédemment générés.

Void generer_les_pts_intersection_surfaces() : générer les sommets d'intersection entre les lignes de brut et les segments de contour de la surface. L'algorithme de cette fonction est le suivant :

```
Début
   Pour chaque ligne i de brut
      Faire ;
         Indice=0 ;
         Pour chaque segment j de conteur
            Faire
               Si (existe intersection entre i et j )
                  Alors
                     ajouter point d'intersection à ligne_point_intersection[indice] ;
                     indice=indice+1 ;
                  Fin si ;
            Fait ;
      Fait ;
Fin
```

Int classifier_points_intersection (double x,double y,int num_surface) : permet de déterminer pour chaque point d'intersection (x,y) avec les segments de contour le cas dans lequel il se trouve. Le cas d'un point indique la situation de point par rapport au contour. Pour cela il, y a 4 cas possible (voir figure 19):

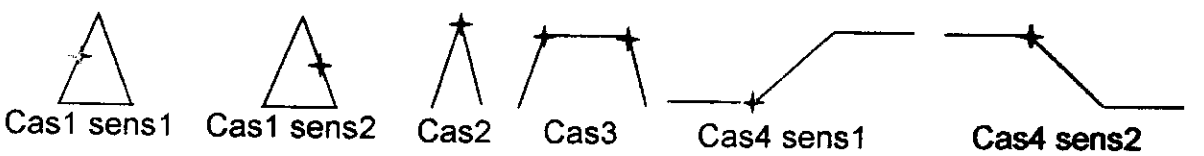


Figure 19: *Les différents cas d'intersection.*

Void evaluer_pts_intersection() : permet de déterminer si un sommet d'intersection se trouve à l'entrée ou à la sortie d'une surface en supposant que nous avons pris le sens gauche->droite par rapport au brut dans le plan(x, y). L'algorithme de cette fonction est le suivant :

```
Début
   Pour chaque ligne i de brut
      Faire
         Pour chaque sommet d'intersection j
            Faire
               Si (cas2 ou cas3)
```

```
                    Alors
                        Pas de changement ;
                    Fin si ;
                    Si ((cas1 sens1) ou (cas4 sens1))
                    Alors
                        entrer_conteur ;
                    Fin si ;
                    Si ((cas1 sens2) ou (cas4 sens2))
                    Alors
                        sortir_conteur ;
                    Fin si ;
                Fait ;
            Fait ;
        Fin
```

void traiter_cas_general(int l) : Après la génération de tous les sommets de brut, cette fonction permet de faire disparaître tout les sommets qui se trouve à l'intérieur de contour de surface ; en affectant une valeur très grand à x pour chaque sommet de brut (une valeur de 100 000000), pour ne laisser que les points qui se trouvent à l'extérieur de la surface projetée. L'algorithme de cette fonction est le suivant :

```
Début
    Pour i=0 à nbre_colonne_brut
        Faire
            Pour chaque deux sommets d'intersections successives k et k+1 ;
            Faire
                Si ((point_segment[i] est égale point_intersection[k]) ou
                    (point_segment[i] est égale point_intersection[k+1]))
                Alors
                    Sortir ;
                Fin si ;
                Si (point(segment[i] est entre point_intersection[k] et
                point_intersection[k+1] )
                Alors
                    Si (sortir_conteur)
                    Alors
                        pas de changement ;
                    Fin si ;
                    Si (entrer_conteur)
                    Alors
                        x_ point_segment[i] = 10 000000 ;
                    Fin si ;
                Fin si ;
        Fait ;
Fin
```

Void Générer_triangle_brut() : permet de générer les triangles de brut à partir des sommets des lignes de brut et les sommets d'intersection. L'algorithme de cette fonction est le suivant :

```
Début
   Pour i=0 à nbre_ligne_brut
      Faire
         Pour j=0 à nbre_colonne_brut
            Faire
               pts1= Segment_brut[i]_points_segment[j] ;
               pts2= Segment_brut[i]_points_segment[j+1] ;
               pts3= Segment_brut[i+1]_points_segment[j]
               pts4= Segment_brut[i+1]_points_segment[j+1] ;
               Si (tout les les x_pts sont déférentes de 10 000000)
                  Alors
                     Si (tout les pts ne sont pas des points d'intersections)
                        Alors
                           générer le triangle(pts1, pts3, pts4) ;
                           générer le triangle(pts1, pts2, pts4) ;
                     Fin si ;
                  Si non ;
                   Alors
                    Si (x_pts2 est égale à 10 000000)
                       Alors
                          pts2=Segment_brut[i]_point_intersection[j+1] ;
                          générer le triangle(pts1, pts2, pts4) ;
                    Fin si ;
                    Si (x_pts4 est égale à 10 000000)
                       Alors
                          pts4=Segment_brut[i+1]_point_intersection[j+1] ;
                          générer le triangle(pts1, pts2, pts4) ;
                    Fin si ;
                    Si (x_pts1 est égale à 10 000000)
                       Alors
                          pts1=Segment_brut[i]_point_intersection[j] ;
                          générer le triangle(pts1, pts2, pts4) ;
                    Fin si
                    Si (x_pts3 est égale à 10 000000)
                       Alors
                          pts3=Segment_brut[i+1]_point_intersection[j] ;
                          générer le triangle(pts1, pts2, pts4) ;
                    Fin si ;
                  Fin si non ;
      Fait ;
Fin
```

La classe de brut comporte plusieurs sous classes, ces sous classes sont montrées dans la figure 20.

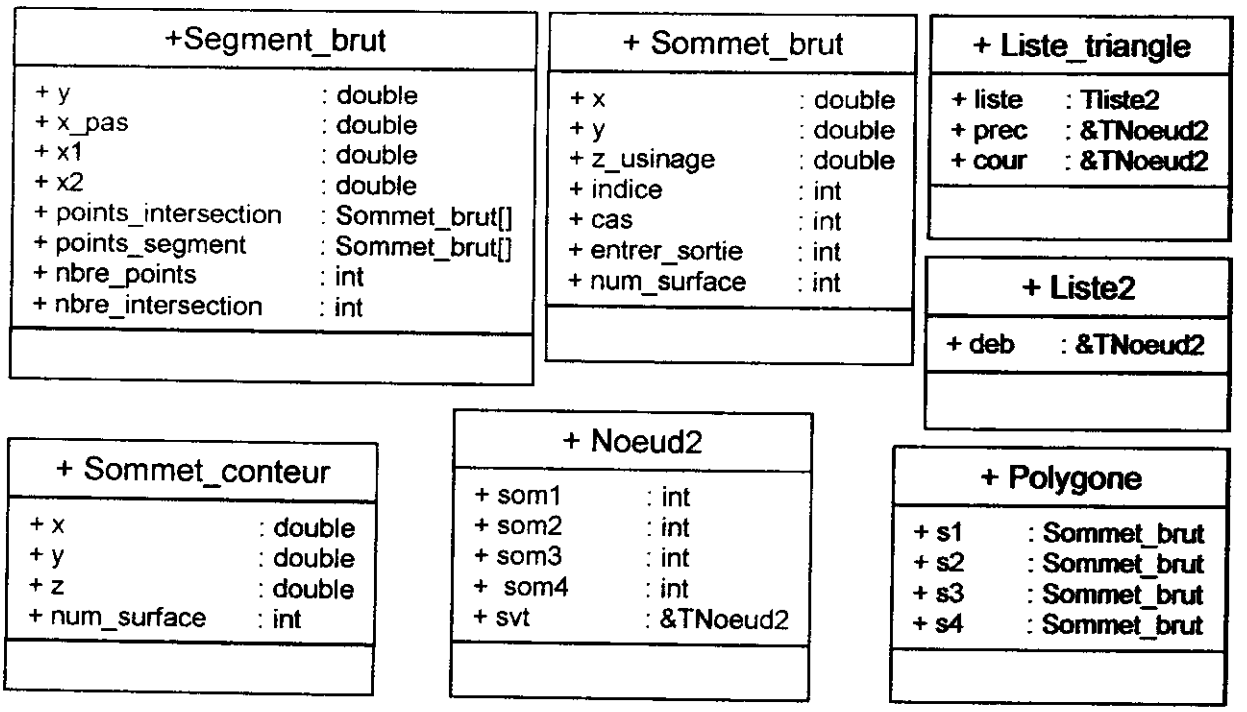


Figure 20 : *Sous classes de la classe de brut.*

- **La classe de vérification de tolérance (TVérification)** :

Elle se charge de la vérification de la tolérance introduite par l'utilisateur, son attribut principal est le suivant :

Tolerance_ebauchage : Il est de type réel, contient la valeur de tolérance introduite par l'utilisateur.

La fonction principale de cette classe s'appelle *Vérifier_tolérence(double tolérance)*, son algorithme est le suivant :

```
Début
   Pour chaque triangle de simulation
      Faire
         Si (z_ triangle_usinage > z_ triangle_téorique+Tolerance_ebauchage )
          Alors
               Excès de matière ;
          Fin Si ;
          Si non ;
           Alors
               Si (z_ triangle_usinage < z_ triangle_téorique)
                Alors
                    Interférence ;
                Fin Si ;
               Si non ;
                Alors
                    Dans l'intervalle de tolérance ;
                Fin Si
               Fin Si non
```

```
                Fin Si non ;
            Fait ;
Fin
```

- **La classe de volume (TVolume) :**

La classe TVolume permet de faire les calculs nécessaires pour la détermination de quantités de matières enlever pendant la simulation de l'opération d'ébauchage (voir figure 21).

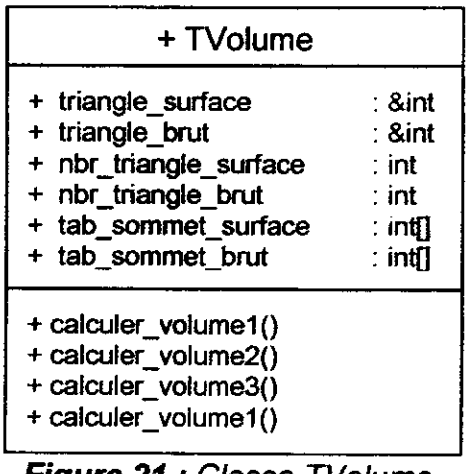


***Figure 21** : Classe TVolume.*

Ses attributs sont les suivants :

tab_sommet_surface[] : est un tableau de type entier qui contient des références sur les sommets de surface qui sont usinés par l'outil pendant la simulation de l'ébauchage sur un même plan.

tab_sommet_brut[] : est un tableau de type entier qui contient des références sur les sommets de brut qui sont usinés par l'outil pendant la simulation de l'ébauchage sur un même plan.

triangle_surface[] : est un tableau de type entier qui contient des références sur les triangles de surface qui sont usinés par l'outil pendant la simulation de l'ébauchage sur un même plan.

triangle_brut[] : est un tableau de type entier qui contient des références sur les triangles de brut qui sont usinés par l'outil pendant la simulation de l'ébauchage sur un même plan.

Pour le calcul du volume, cette classe comporte plusieurs fonctions, comme la fonction *calculer_volume1()* pour le calcul du volume d'un tétraèdre, *calculer_volume2()* pour le volume d'une pyramide et *calculer_volume3()* pour un prisme, et pour calculer le volume général enlever entre deux positions il y a la fonction *calculer_volume()* dont l'algorithme est le suivant :

```
Début
  Pour chaque plan de simulation
      Faire
          Pour chaque deux positions successives
              Faire
```

```
                    Pour chaque sommet dans l'enveloppe d'outil
                       Faire
                          Si (un sommet de surface)
                             Alors
                                Ajouter le sommet à tab_sommet_surfac[i] ;
                                i=i+1 ;
                          Fin si ;
                          Si (un sommet de brut)
                             Alors
                                Ajouter le sommet à tab_sommet_brut[j] ;
                                j=j+1 ;
                          Fin si ;
                       Fait ;
                    Pour chaque sommet de tab_sommet_surfac
                       Faire
                             Ajouter les triangles de cet sommet à
                             tab_triangle_surface[k] ;
                             k=k+1 ;
                       Fait ;
                    Pour chaque sommet de tab_sommet_brut
                       Faire
                          Ajouter les triangles de ce sommet à tab_triangle_brut[m] ;
                          m=m+1 ;
                       Fait ;
                    Pour chaque triangle dans tab_triangle_surface ou
                                                tab_triangle_brut
                       Faire
                          Si (un tétraèdre)
                             Alors
                                calculer_volume1() ;
                          Fin si ;
                          Si (un pyramide)
                             Alors
                                calculer_volume2() ;
                          Fin si ;
                          Si (un prisme)
                             Alors
                                calculer_volume3() ;
                          Fin si ;
                       Fait ;
                    calculer_volume() ;
                 Fait ;
           Fait ;
     Fin.
```

- **La classe d'adaptation des vitesses d'avance (TAdaptation) :**

Permet l'adaptation de la vitesse d'avance de l'outil en fonction de quantités de matières enlevés, en utilisant certains paramètres qui sont fixés, et un paramètre qu'est déjà calculé, c'est le volume. À partir de ces paramètres, nous calculons

d'autres paramètres qui permettent finalement le calcul de la vitesse d'avance adaptée pour un déplacement donné dans la matière. Le calcule se fait par la fonction *Optimiser_vitesse()*, l'algorithme de cette fonction est le suivant :

```
Début

    Pour chaque déplacement dep=0 à nbr_deplacement
        Faire
            Calculer le volume enlevé ;
            Calculer le temps optimal ;
            Calculer la distance ;
            Vitesse_optimale[dep] = Distance / Temps ;
        Fait ;
Fin
```

- **La classe de correction de trajectoire (TCorrection) :**

Réalise l'opération de correction de trajectoire, en basant sur la vérification déjà faite (voir figure 22).

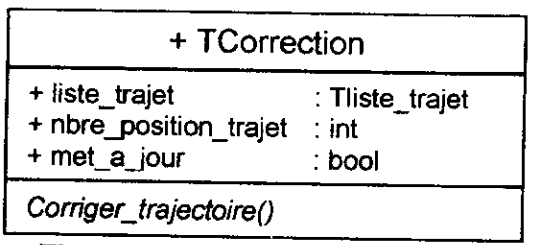


Figure 22 : *Classe TCorrection.*

Les important attributs de cette classe sont :

liste_trajet : Est une liste chaîné, contient les nouvelles positions d'outil corrigées.

nbre_position_trajet : De type entier, indique le nombre des positions d'outil obtenu après la correction.

La fonction principale de cette classe est *Corriger_trajectoire()*, son algorithme est le suivant :

```
Début
   Pour chaque deux positions successives de trajectoire pos1 et pos2
      Faire
          Si (interférence dans pos1)
           Alors
               Eliminer l'interférence de pos1 ;
          Fin si
          Si (interférence dans pos2)
           Alors
               Eliminer l'interférence de pos2 ;
          Fin si
          Si (Interférence entre pos1 et pos2)
           Alors
               Eliminer l'interférence entre pos1 et pos2 ;
          Fin si ;
```

```
        Faite ;
Fin
```

- **La classe de visualisation (TVisualisation) :**

C'est la classe principale d'affichage. Elle comporte plusieurs fonctions destiner à l'affichage des différentes caractéristique visuels de la surfaces, ainsi que les caractéristiques visuels de brut et de l'outil avec sa trajectoire, parmi ces fonctions, nous pouvons citer: la fonction *Dessiner_sommets()* pour l'affichage des sommets de surface et *Dessiner_triangle()* pour les triangles de surface, *Dessiner_Regions()* pour voir les régions de surface et de brut, *Dessiner_outil()* pour voir l'outil d'usinage, *Dessiner_niveaux()* pour afficher les différentes niveaux de l'ébauchage, *Dessiner_conteur()* pour le contour de surface, *Dessiner_rendu_surface()* pour visualiser le remplissage de surface et *Dessiner_remplissage_brut()* pour le remplissage de brut. Ces deux dernières fonctions, lorsqu'ils sont appelés simultanément permettent de voir la forme de la pièce en simulation. En plus de ces fonctions, il y a d'autre qui ne sont pas cités et qui sont en nombre total égales à 34 fonctions de visualisation (voir figure 23).

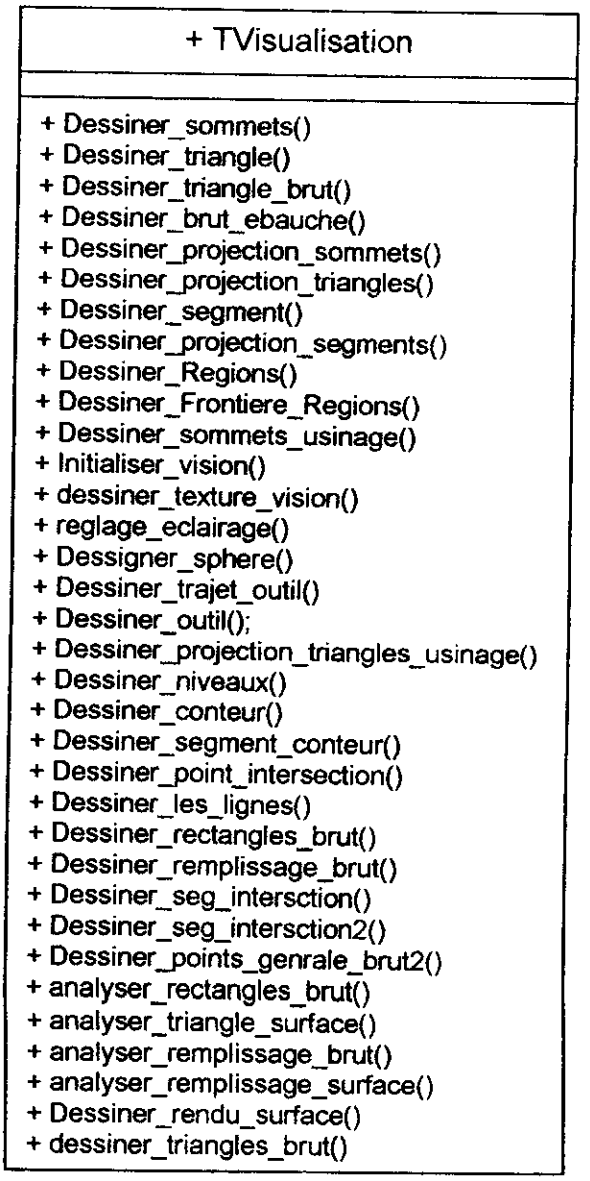


***Figure 23** : Classe TVisualisation.*

- **La classe de lumière (TLumière) :**

C'est la classe qui permet le réglage des paramètres de la lumière de la scène de simulation. Elle comporte plusieurs attributs qui correspondent aux différents paramètres de lumières permises par la bibliothèque graphique d'OpenGL ainsi qu'un certain nombre de fonctions permettant la modification de ces paramètres. (voir figure 24).

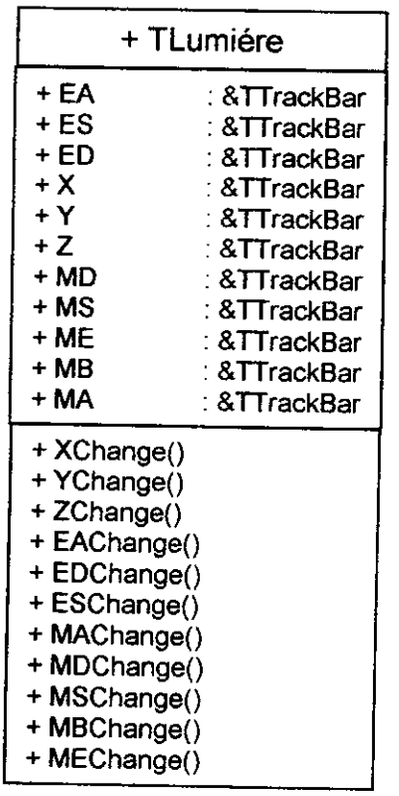


Figure 24 : *Classe TLumière.*

Ces attributs sont les suivants :

EA, ED, ES : sont des paramètres pour le réglage de l'éclairage.

X, Y, Z : pour le réglage de la position de la lumière.

MD, MS, ME, MB, MA : pour le réglage des paramètres de la matière.

- **La classe des couleurs (TCouleur) :**

Permet la modification des couleurs des différents objets affichés sur la scène de simulation comme le remplissage de surface et le remplissage de brut ainsi que la couleur de l'outil et la couleur de surface théorique...etc.(voir figure 25)

+ TCouleur	
+ sommets_visible	: bool
+ sommets_porjection_visible	: bool
+ triangles_visible	: bool
+ triangles_porjection_visible	: bool
+ segment_visible	: bool
+ segment_projection_visible	: bool
+ sommets_simulation	: bool
+ triangles_simulation	: bool
+ outil	: bool
+ trajet_outil	: bool
+ remplissage	: bool
+ brut	: bool
+ tab_coleur	: bool[]
+ form_num	: int
+ Triangles_projetés()	
+ Sommet_téorique()	
+ Triangle_téorique()	
+ Sommets_projetés()	
+ Polygones_brut()	
+ Sommets_brut()	
+ Sommets_surface()	
+ Triangles_surface()	
+ Outil()	
+ Trajet_outil()	
+ Rendu_surface()	
+ Rendu_brut()	

Figure 25 : *Classe TCouleur.*

III.3.3. Diagramme de collaboration :

Pour décrire les différentes opérations qui existent entre les objets, nous avons utilisés le diagramme de collaboration qu'est une extension de diagramme d'objets et qui décrit le comportement collectif de cet ensemble d'objets. En vu de réaliser les opérations de notre système en décrivant leurs interactions modélisées par des envois des messages.

La figure 26 représente le diagramme de collaboration de notre système avec ses objets, ainsi que les opérations entre ces objets et les différents échanges de données.

Tout commence par l'appel de l'utilisateur à l'objet *Triangulation*, ce dernier appelé l'objet *Région* par l'opération ***créer*** pour la création des régions, puis il y a un appel de l'objet *simulation* par l'opération ***Simuler***. Ensuit, l'objet de simulation envoi une demande de récupération de trajectoire (***Récupérer***) à l'objet *Données*, ce dernier envoi la trajectoire à l'objet de simulation (valeur de retour), puis l'objet simulation demande la création de brut (***générer***) à l'objet *Brut*, puis vient la demande de visualisation graphique de simulation, réalisée par l'opération ***Visualiser,*** avec une possibilité de modifier les couleurs ***(modifier)*** et régler la lumière (***Régler***), ces opérations sont envoyés à l'objet *Visualisation*. Ensuit il y a l'opération de vérification de tolérance (***Vérifier***) envoyé de l'objet de simulation à l'objet *Vérification*, puis l'adaptation des vitesses d'avance (***Adapter***) à l'objet

Adaptation, ce dernier appelé l'objet de calcule de volume *(volume)* par l'opération ***calculer***, ce dernier envoi le volume calculé comme valeur de retour à l'objet d'adaptation, l'objet d'adaptation demande de l'objet *Fiche_2* d'afficher les résultats d'adaptation (***Afficher***). Enfin l'objet de simulation demande la correction de la trajectoire (***Corriger***) à l'objet *Correction*.

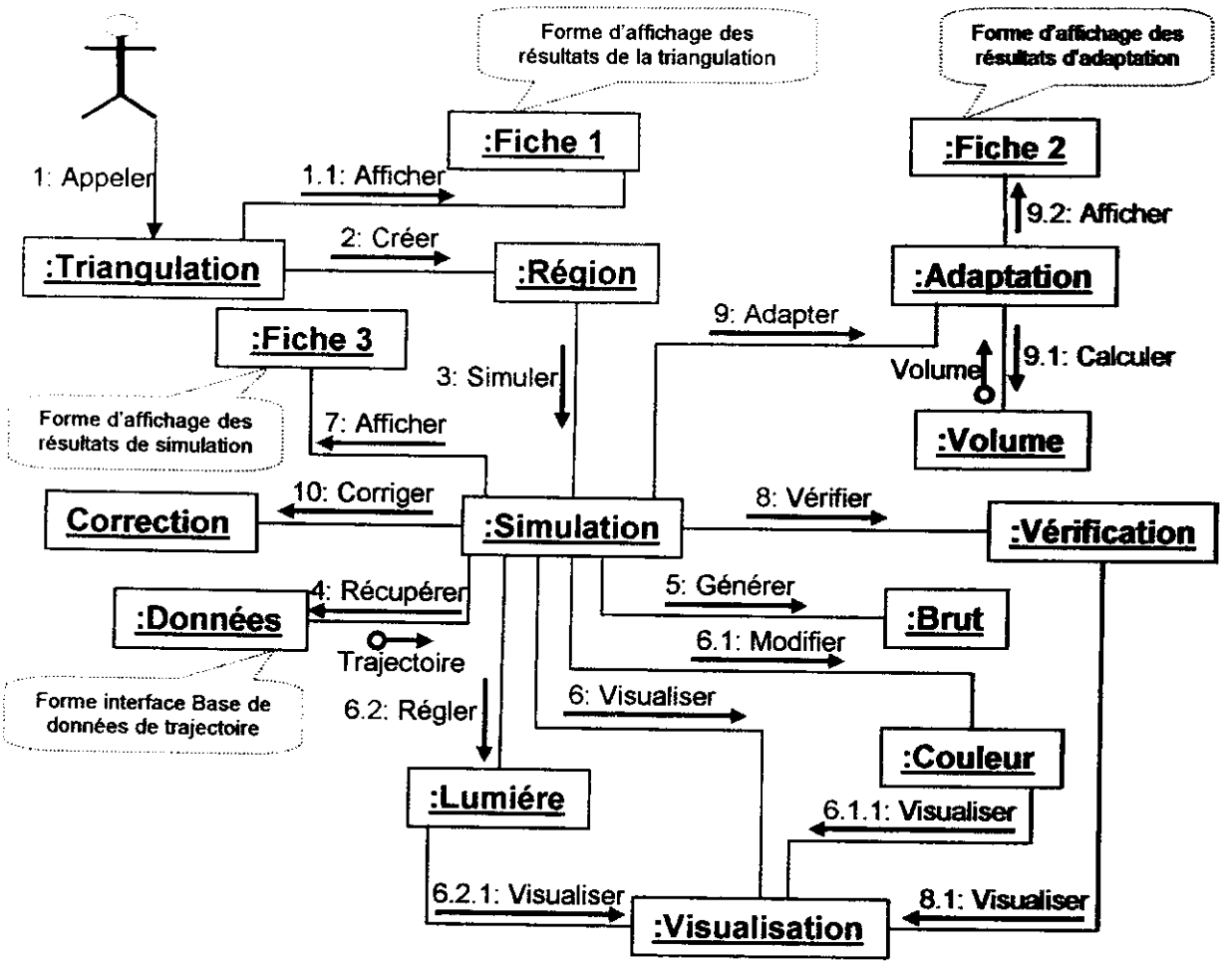


Figure 26: *Diagramme de collaboration.*

III.3.4. Diagramme d'activité :

Pour mettre l'accent sur les activités, leurs relations et leurs impacts sur les objets, nous avons utilisés le diagramme d'activité qui permet aussi de représenter les transitions entre les activités ainsi que leurs synchronisations.

La figure 27 représente le diagramme d'activité de la simulation d'ébauchage effectué par notre système. Ce diagramme est un enchaînement logique des différentes activités nécessaire pour la réalisation de la simulation, en commencent par l'ouverture d'une surface puis la triangulation de cet dernière jusqu'au lancement et la réalisation de simulation.

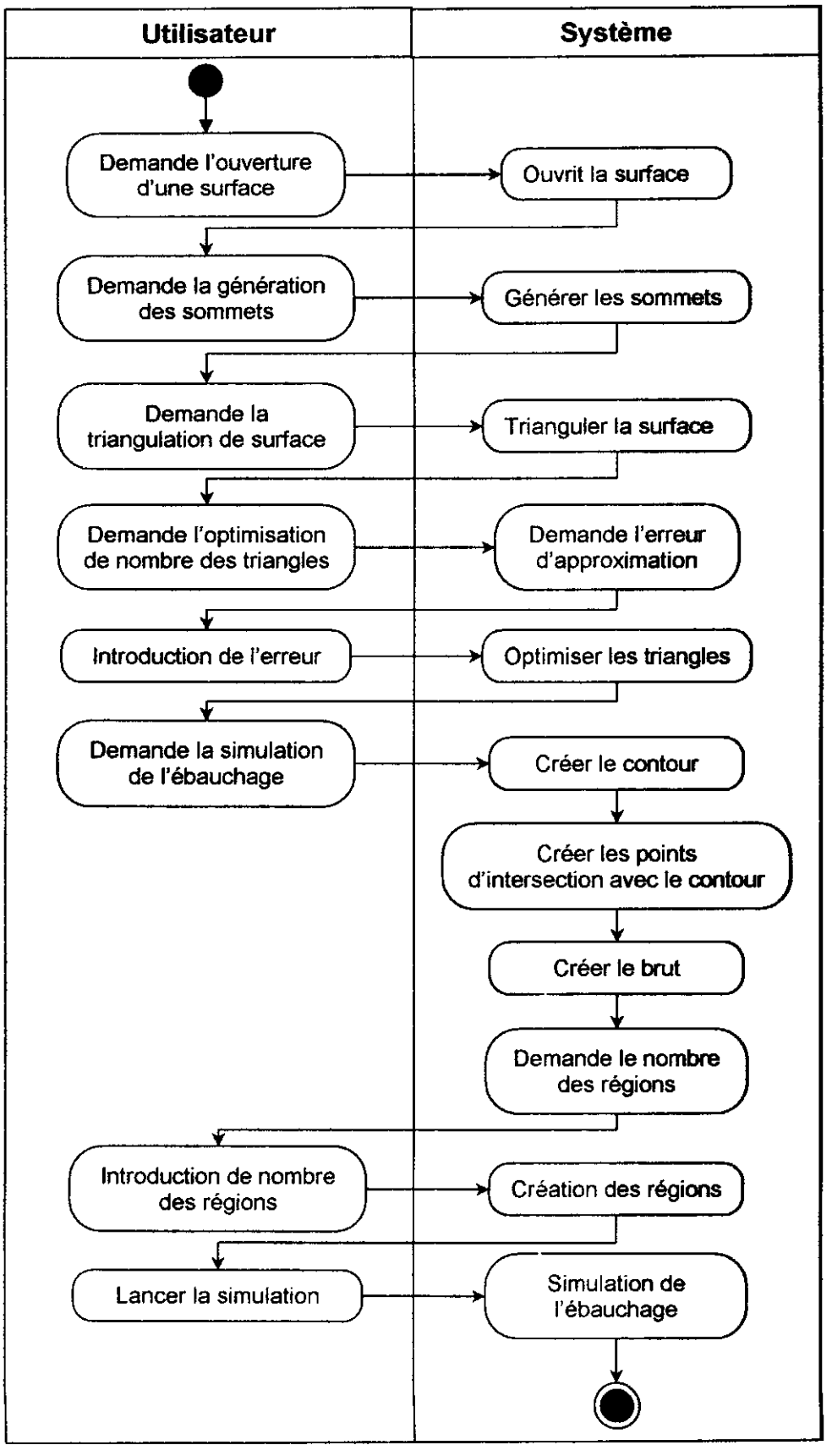


Figure 27: *Diagramme d'activité pour la simulation d'ébauchage.*

La figure 28 représente le diagramme d'activité de processus d'adaptation des vitesses d'avances, en fonction de volume enlevé calculé par notre système, avec la prise en compte de quelques paramètres introduits par l'utilisateur.

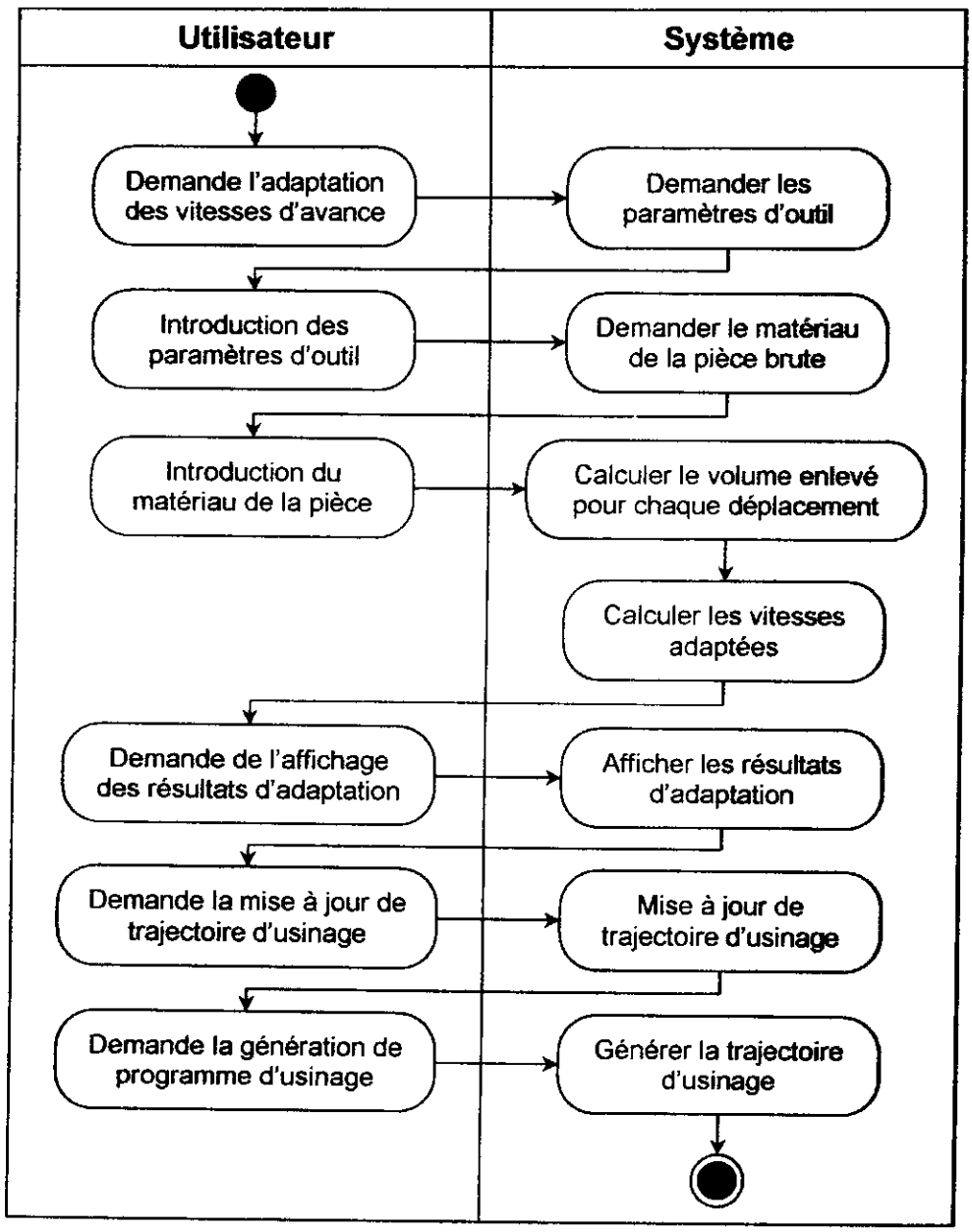


***Figure 28:** Diagramme d'activité pour l'adaptation des vitesses d'avance.*

La figure 29 représente le diagramme d'activité de processus de vérification de tolérance, en fonction de type de surface et la tolérance introduit par l'utilisateur, les résultats de ce processus est l'affichage des trois types des zones détecter (synchronisation des trois activités), sur la pièce brute en cour d'usinage (zones tolérées en vert, zones d'excès de métier en bleu et zones d'interférence en rouge).

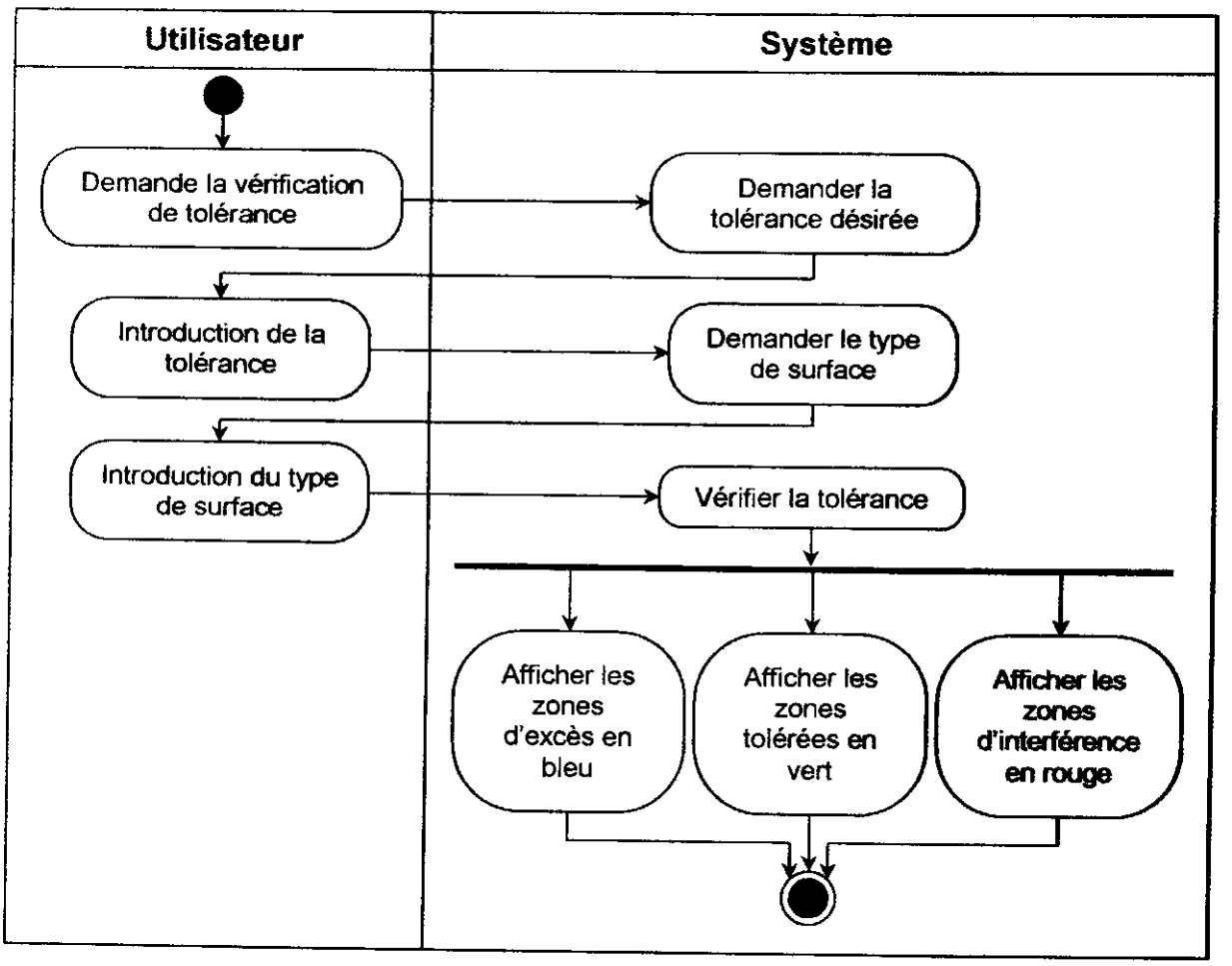


***Figure 29:** Diagramme d'activité pour la vérification de la tolérance.*

La figure 30 représente le diagramme d'activité de processus de correction de la trajectoire d'usinage, en fonction de type de la surface; la correction est effectuée en éliminant les positions qui causent l'interférence avec la pièce brute. Le processus se termine par l'enregistrement de la nouvelle trajectoire corrigés et la génération de programme d'usinage qui corresponde à cette trajectoire.

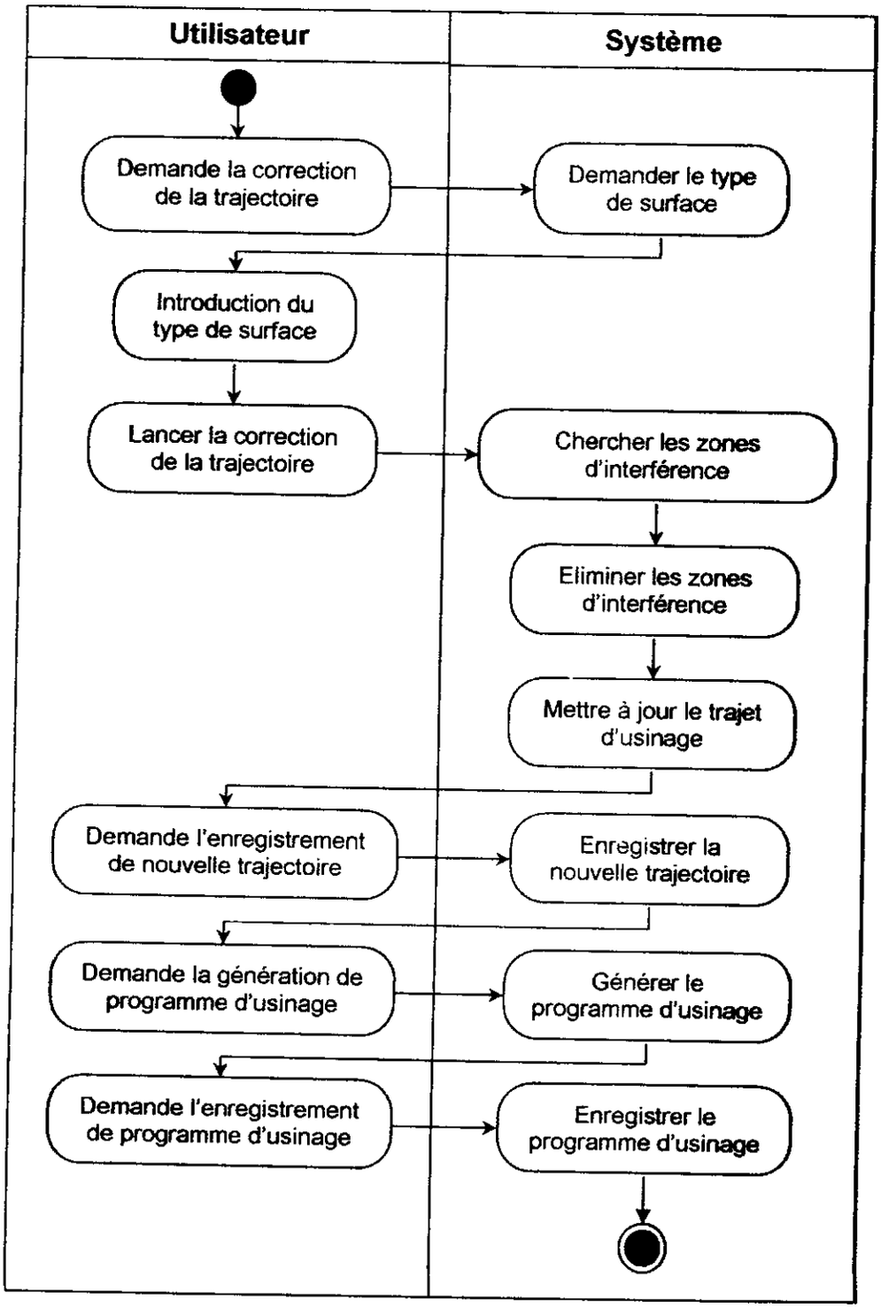


***Figure 30:** Diagramme d'activité pour la correction de la trajectoire d'usinage.*

III.3.5. Diagramme de séquence :

Pour décrire la manière d'utilisation de notre système, nous avons utilisés les diagrammes de séquence qui expriment la logique des scénarios des différents cas d'utilisation de notre système (voir figure 17 et 18).

Pour faire la simulation de l'opération d'ébauchage (voir figure 31) :

- L'utilisateur demande l'ouverture d'une surface.
- Le système ouvre la surface.
- L'utilisateur demande la triangulation de la surface.
- Le système triangulé la surface.
- L'utilisateur demande l'optimisation de nombre des triangles.
- Le système optimise le nombre des triangules.
- L'utilisateur demande la vérification des contraintes.
- Le système vérifié les contraintes.
- L'utilisateur demande la simulation de l'opération d'ébauchage.
- Le système demande le nombre des régions.
- L'utilisateur introduit le nombre des régions.
- L'utilisateur demande la création des régions.
- Le système créé les régions.
- L'utilisateur lance la simulation de l'ébauchage.
- Le système simulé l'opération d'ébauchage.

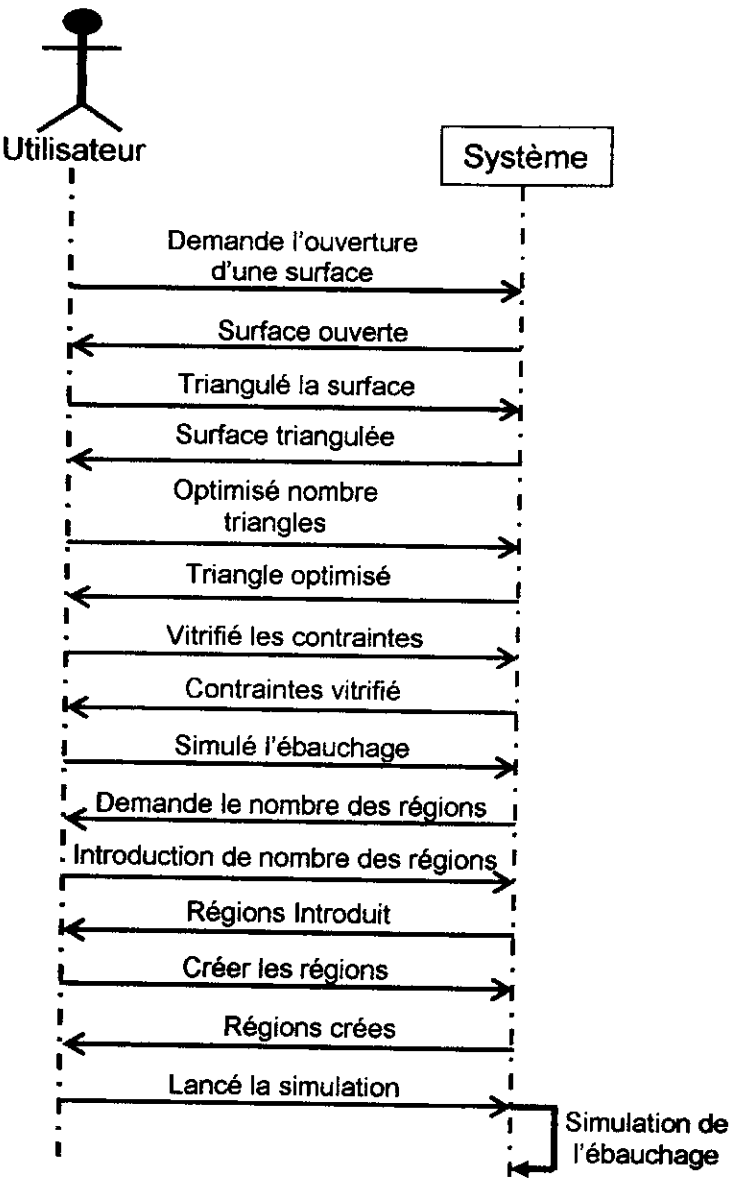


***Figure 31:** Diagramme de séquence pour la simulation d'ébauchage.*

Pour faire l'adaptation des vitesses d'avance en fonction de volume enlevé (voir figure 32) :

- L'utilisateur demande l'adaptation des vitesses d'avance.
- Le système demande les paramètres d'outil.
- L'utilisateur introduit les paramètres d'outil.
- Le système demande le matériau de la pièce.
- L'utilisateur Introduit le matériau de la pièce.
- L'utilisateur lance l'adaptation.
- Le système commence à faire l'adaptation des vitesses d'avance.

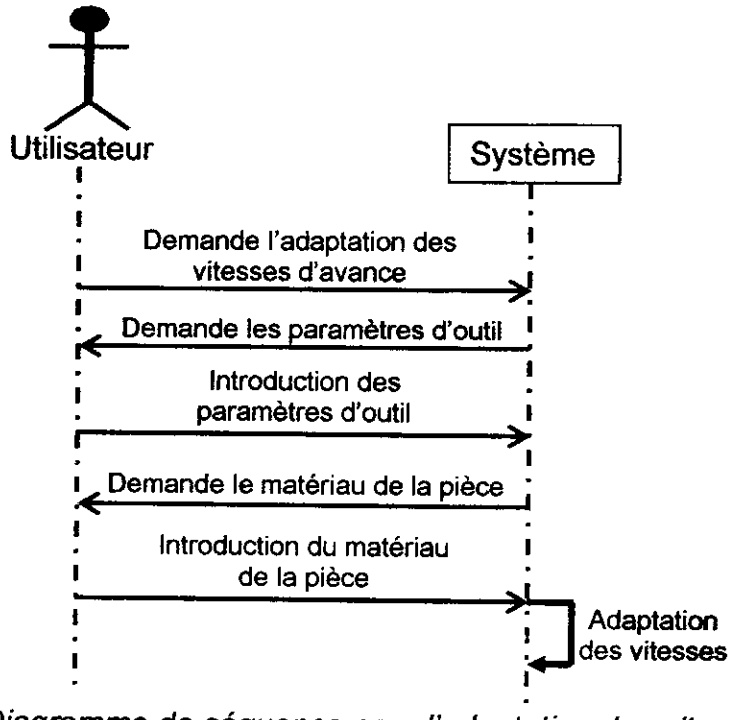


***Figure 32:** Diagramme de séquence pour l'adaptation des vitesses d'avance.*

Pour faire la vérification de la tolérance désirée par l'utilisateur **(voir figure 33)** :

- L'utilisateur demande la vérification de tolérance.
- Le système demande la tolérance désirée.
- L'utilisateur introduit la tolérance qu'il veut.
- Le système demande le type de surface.
- L'utilisateur Introduit le type de surface.
- Le système commence à faire la vérification de tolérance.

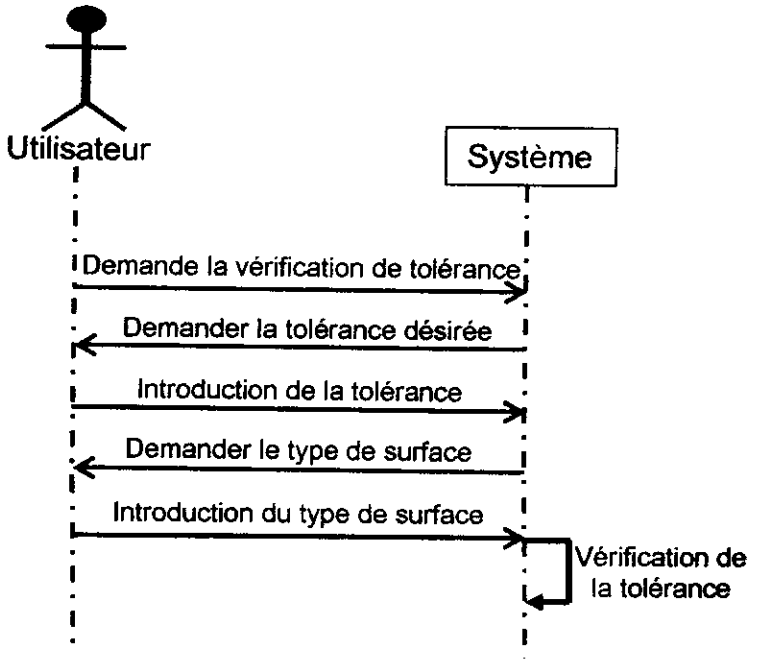


***Figure 33:** Diagramme de séquence pour la vérification de la tolérance.*

Pour faire la correction de trajectoire d'usinage (voir figure 34) :

- L'utilisateur demande la correction de la trajectoire
- Le système demande le type de surface.
- L'utilisateur introduit le type de surface.
- L'utilisateur lance la correction de la trajectoire.
- Le système commence à corrigé la trajectoire.

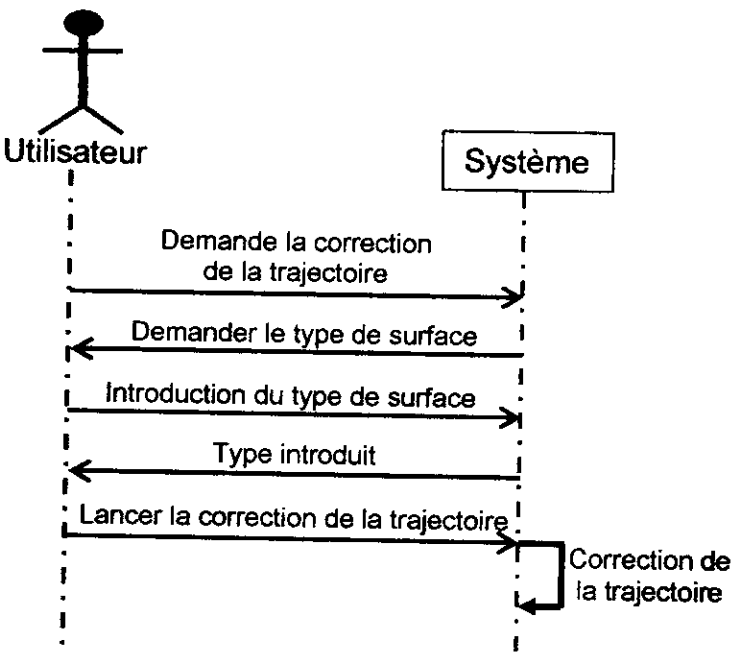
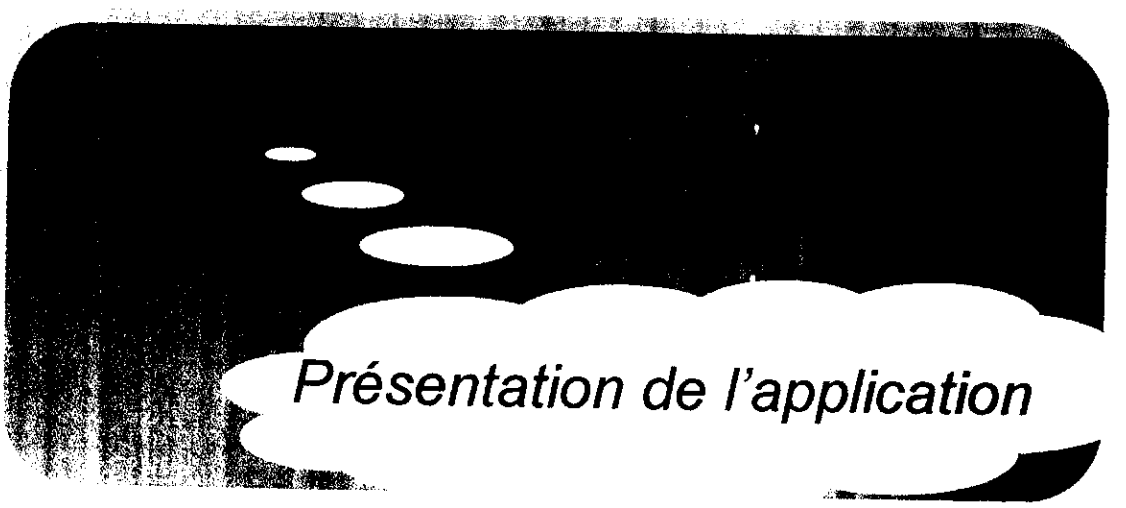


Figure 34: *Diagramme de séquence pour la correction de la trajectoire.*

IV. CONCLUSION :

Dans ce chapitre, nous avons commencés par la présentation de la problématique et les objectifs à atteindre. Par la suite, nous sommes passés à la solution proposée avec les différents diagrammes de conception et les algorithmes implémentés. Dans le chapitre suivant, nous allons présenter notre application et leur différente fenêtres utilisées.


Présentation de l'application

I. INTRODUCTION :

Après la description de la conception et des solutions proposées dans le chapitre précédent, l'étape suivante est de passer à la présentation de l'application logicielle et la réalisation de ces solutions avec la prise en compte des différents besoins des utilisateurs. Pour ce la, dans ce chapitre nous allons présenter le travail réalisé et l'environnement de travail.

II. PRESENTATION GENERALE DE L'APPLICATION :

Notre application est un logiciel de FAO complémentaire, qui comporte des fonctionnalités d'usinage et qui utilise des géométries conçues en 3D qui sont totalement associées à la CAO. Ce travail est un ensemble de fenêtres Windows permettant un affichage graphique de la simulation de l'opération d'ébauchage (visualisation de la pièce produite et des parcours d'outil (parcours d'usinage)) ainsi que la vérification graphique des tolérances exigées par l'utilisateur et la correction de la trajectoire d'usinage. Toutes ces fonctionnalités sont très utiles dans les ateliers de production.

III. ENVIRONNEMENT DE TRAVAIL :

III.1. Fenêtre principale :

La fenêtre principale se compose de deux sous fenêtres. Une des deux est composée d'un ensemble de boutons destinés à la conception CAO et qui comportent aussi une barre des menus principales et une barre d'information. Tel que cette fenêtre sert à la conception des courbes et des surfaces ainsi que la manipulation des paramètres des surfaces, polygone de contrôle et tout autre manipulation. L'autre sous fenêtre, est simplement une fenêtre destinée à la visualisation des différents aspects géométrique en 3D, en utilisant la bibliothèque graphique 3D « OpenGL ».La fenêtre principale est montrée par la figure 1.

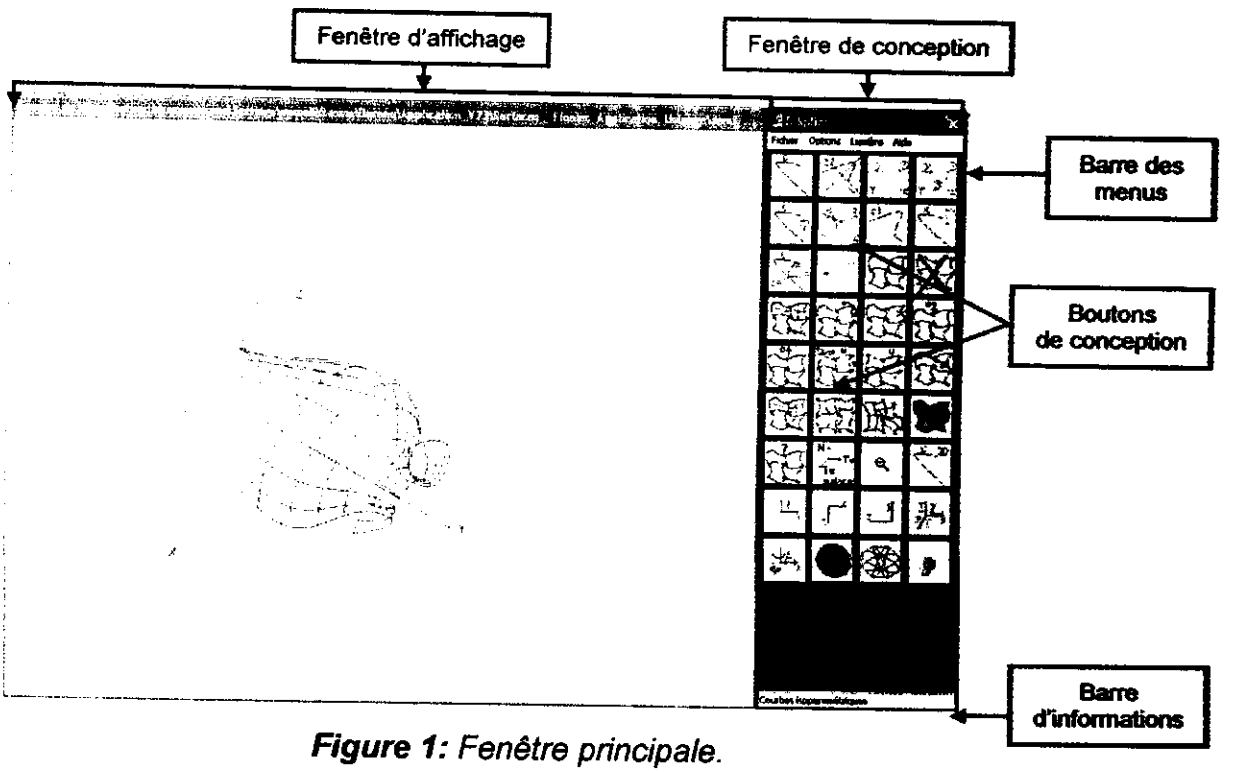


Figure 1: *Fenêtre principale.*

III.2. Barre du menu principal :

La barre du menu principal supérieur est composée de quatre rubriques et qui sont :

- Rubrique Fichier : qui comporte toutes les fonctionnalités de manipulation des fichiers comme l'ouverture d'un fichier, la création d'un nouveau fichier et la sauvegarde du fichier.

- Rubrique Option : cette rubrique permet la modification des différents paramètres des courbes ainsi que les paramètres des surfaces et les paramètres d'usinage comme elle permet aussi de faire la simulation d'usinage.

- Rubrique Lumière : permet la modification des paramètres de lumière appliquée sur la fenêtre de visualisation.

- Rubrique Aide : pour l'affichage de certaines informations concernant la réalisation et l'utilisation de l'application.

III.3 Rubrique de simulation :

La partie que nous avons réalisés effectivement et que nous avons ajoutés à cette application se trouve au niveau de la rubrique *Option* et qui porte le titre suivant *Simulation, Vérification et Correction*. Donc, pour entrer dans la partie que nous avons crées il faut d'abord lancer l'application et ouvrir une surface qui comporte le modèle COA et la trajectoire d'usinage (en ébauchage) s'il a été déjà générée auparavant et enregistrer avec le fichier de model CAO de surface, Si non, il faut le générer à nouveau. Une foi la surface est ouverte, et pour faire la simulation graphique il faut aller à la rubrique de menu Option puis aller vers le sous rubrique *Simulation, Vérification et Correction* (voir figure 2).

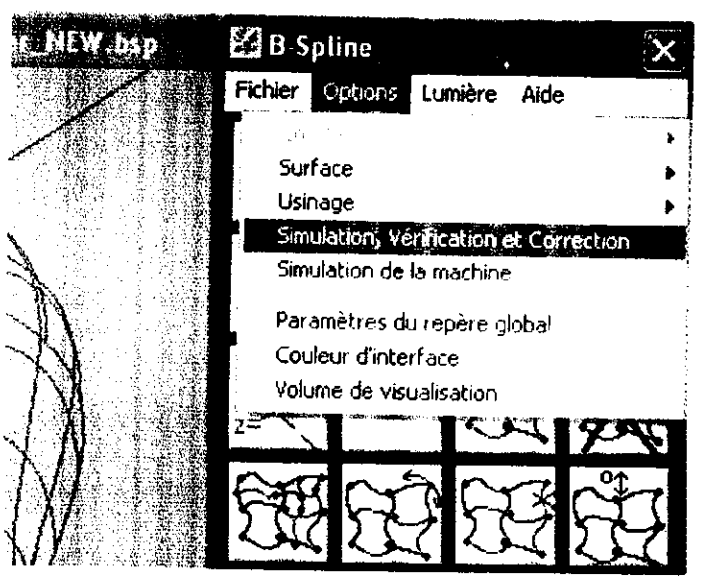


Figure 2: *Rubrique de simulation, vérification et correction.*

IV. PRESENTAION DU TRAVAIL REALISE :

Dans ce qui suit, nous allons présenter toutes les fenêtres créées dans notre travail pour l'extension de l'application.

IV.1. Fenêtre de triangulation de surface :

C'est la première fenêtre qui doit apparaître après que l'utilisateur clique sur le sou rubrique *Simulation, Vérification et Correction* de la rubrique *Option* de la barre des menus, La fenêtre de triangulation est montrée par la figure 3.

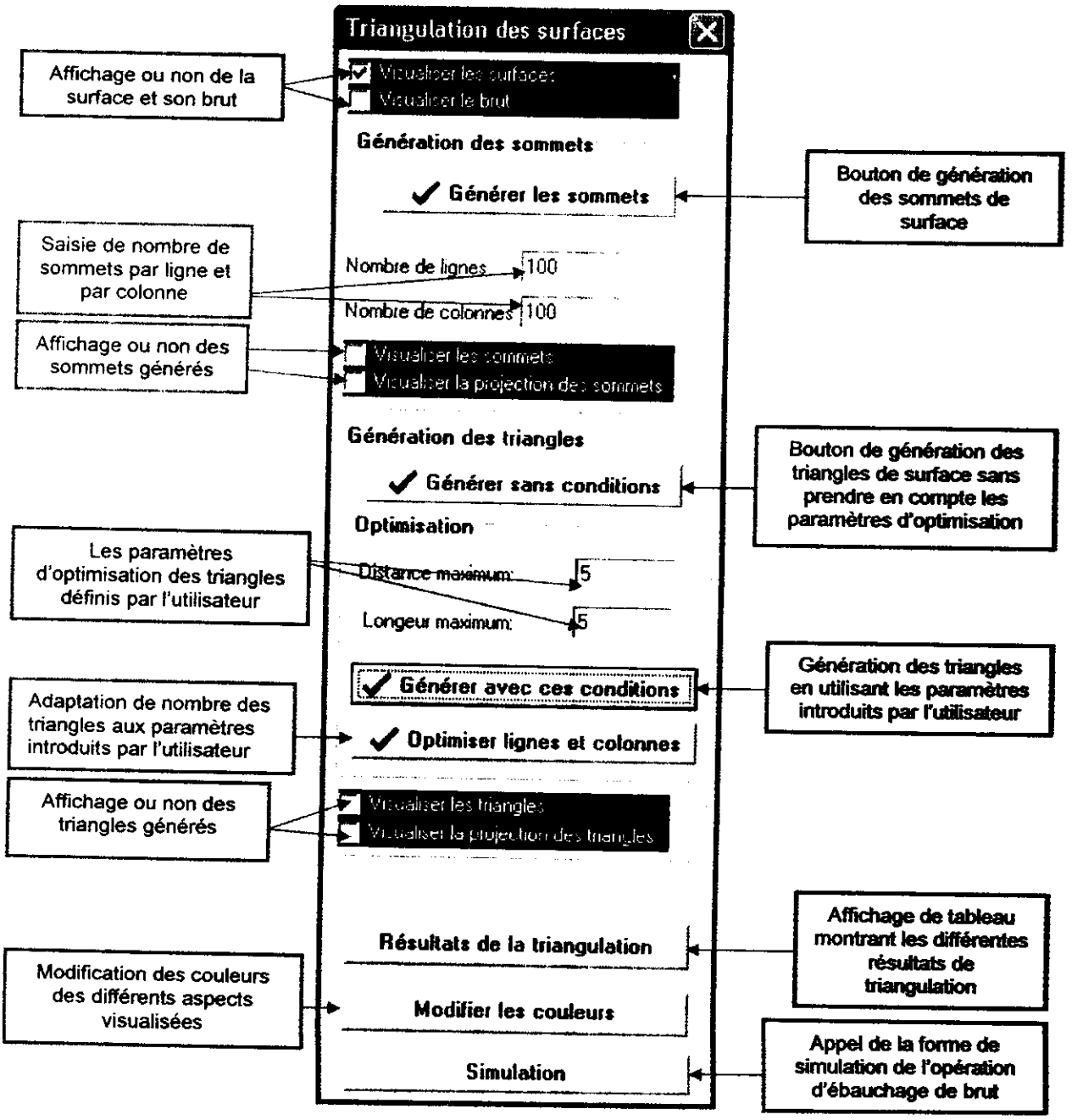


Figure 3: *Fenêtre de triangulation de surface.*

Dans cette fenêtre l'utilisateur, doit générer en premier lieu les sommets après l'introduction de nombre des lignes et de colonnes dans les zones de saisie (au premier appel de cette fenêtre le bouton *générer les sommets* est le seul bouton actif). Une fois l'utilisateur cliqué sur le bouton générer sommet, il y a une activation du bouton de génération des triangles *générer sans condition,* car il n y a pas des triangles sans les sommets, il y a aussi une activation de bouton ***Modifier les couleurs*** (lorsque les sommets sont générés, alors l'utilisateur peut modifier la couleur d'affichage des sommets).

Ensuite, l'utilisateur peut générer les triangles en un seul clique sur le bouton *générer sans condition* et activer les autre boutons sauf le bouton ***optimiser lignes et colonnes***. Comme il y a aussi une activation des zones de saisie pour permettre à l'utilisateur d'entrer le paramètre de longueur maximale des segments des triangles et le paramètre de la hauteur maximale entre la surface théorique et la surface approximées puis cliquer sur le bouton *générer avec ces condition* pour régénérer les triangles mais cette fois ci en utilisant les paramètres introduits. À ce clique, le bouton *optimiser lignes et colonnes* est activé pour donner la possibilité d'optimiser le nombre des lignes et des colonnes pour qu'il soit le minimum possible sachant que les paramètres introduits sont vérifiés.

Pour afficher les résultats, l'utilisateur peut les consulter en cliquant sur le bouton *Résultas de Triangulation* qui appelle la fenêtre des résultas de triangulation Dans cette fenêtre (voir figure 4), l'utilisateur à la possibilité d'afficher la surface, les sommets et triangles générer avec leur projection sur le plan supérieur de la surface, ainsi que les limites de brut qui enveloppe la surface.

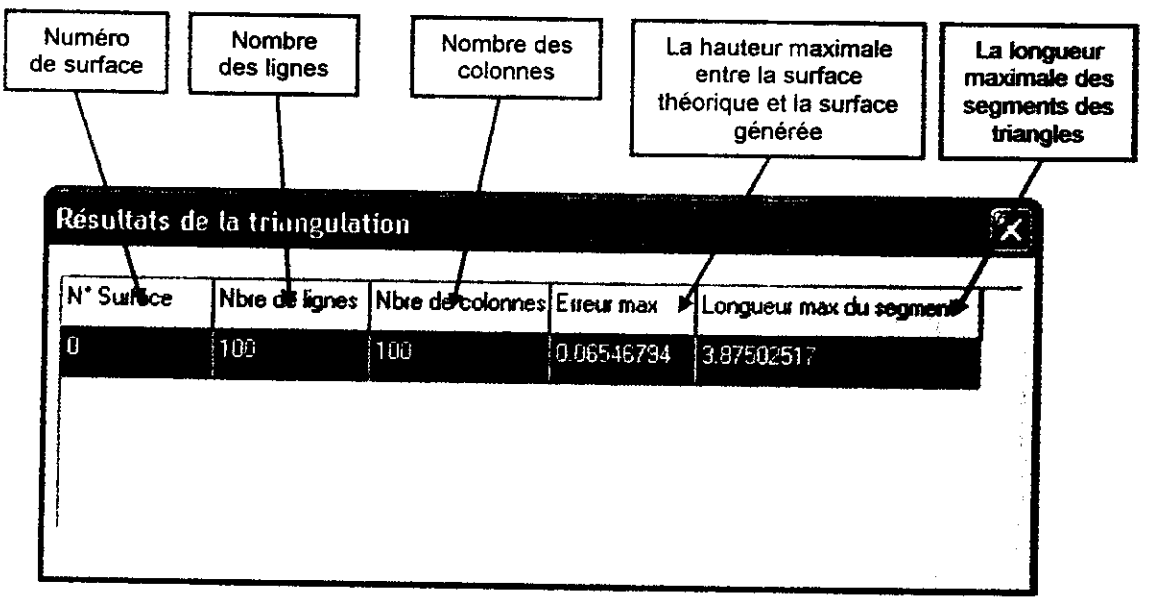


Figure 4: *Fenêtre d'affichage des résultats de triangulation.*

Dans la fenêtre de la figure 4, l'utilisateur peut voir pour chaque surface qui a le numéro de surface comme référence, il peut voir le nombre des lignes et le nombre des colonnes utilisés dans la génération des sommets tel qu'ils sont égaux aux valeurs introduites par l'utilisateur dans la première génération et ils seront diminués après l'optimisation en fonction des paramètres introduits par l'utilisateur. Enfin, le

bouton *Simulation* permet d'appeler la fenêtre de simulation de l'opération d'ébauchage.

IV.2. Fenêtre de création des régions :

Cette fenêtre permet de générer les régions de la face supérieure de brut en utilisant les projections des sommets précédemment générés, en utilisant les deux données nombre des régions par ligne et par colonne introduits par l'utilisateur. Cette fenêtre est montrée dans la figure 5.

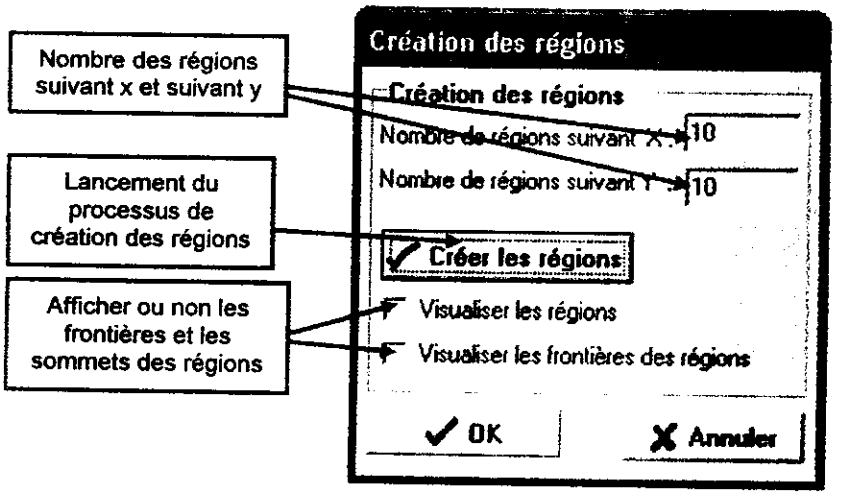


Figure 5: *Fenêtre de création des régions.*

Une foi que les données sont saisies l'utilisateur peut lancer l'opération de création des régions en cliquant sur le bouton *Créer les régions.* Ensuite il peut voir ces régions avec leurs contenus. Une fois terminée l'utilisateur peut cliquer sur le bouton *OK* pour appeler la fenêtre suivant ou *Annuler* pour revenir à la fenêtre précédente.

IV.3. Fenêtre de simulation de l'ébauchage :

C'est la fenêtre principale de notre application. Elle permet de faire la simulation, et de plus, elle permet d'afficher les différents aspects de visualisation soit de surface ou de brut lors d'usinage, comme elle permet d'appeler les fenêtres d'affichage des résultas d'adaptation des vitesses d'avance et la fenêtre de correction de la trajectoire ainsi que les fenêtres de modification des paramètres d'affichage et de modification de nombres des régions sans oublier la fenêtre de fixation de tolérance. Toutes ces modifications peuvent être faites même au cours de simulation. Cette fenêtre est montrée par la figure 6.

Il faut dire que cette fenêtre utiliser les différents calcules faite dans les fenêtres précédentes, tel que la génération des sommets (de surface et de brut), création des triangles (de surface et de brut) et en fin création des régions.

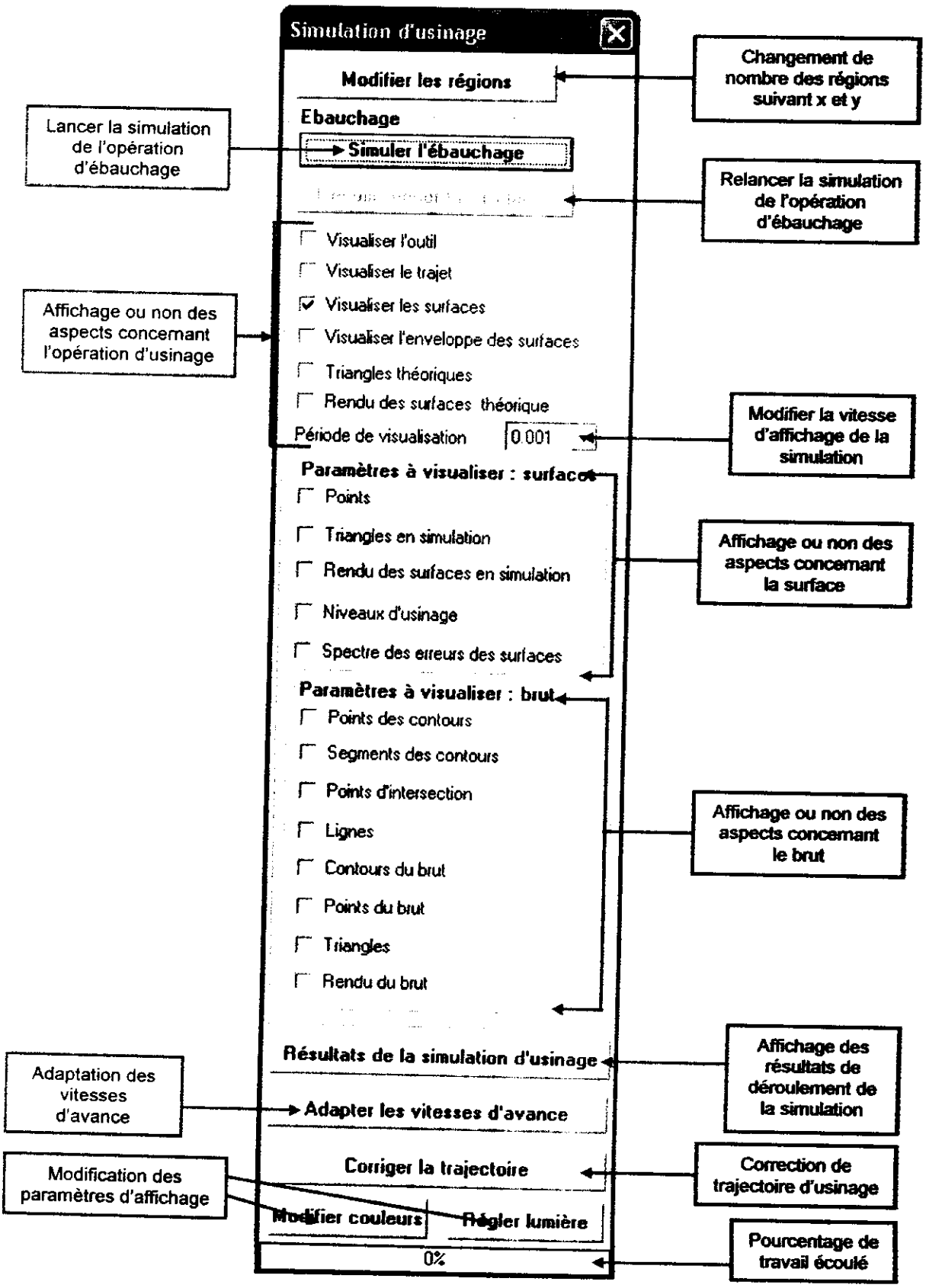


Figure 6: *Fenêtre de simulation de l'opération d'ébauchage.*

Une foi que la fenêtre de la figure 6 est affichée, l'utilisateur peut lancer l'opération de simulation en cliquant sur le bouton *Simuler l'ébauchage*. À ce moment

la barre de progression commence à s'incrémenter pour nous montrer le pourcentage de trajet parcourait de l'opération d'ébauchage de pièce. L'utilisateur peut aussi afficher les différents aspects de visualisation de la simulation de surface ou de brut. Durant la simulation, l'utilisateur peut cliquer sur le bouton *Résultats de la simulation d'usinage* pour montrer la fenêtre qui affiche la variation des vitesses d'avance d'outil en fonction du volume enlevé calculé entre chaque deux positions successives (voir figure 7). Comme il peut cliquer sur le bouton *Modifier couleur* pour changer les couleurs des différents aspects de visualisation (voir la figure 8) et sur le bouton *Régler lumière* pour activer la fenêtre de réglage de lumière (voir figure 9). Si l'utilisateur veut optimiser les vitesses d'avance de l'outil il n'a que cliquer sur le bouton *Optimiser les vitesses d'avance* pour passer à la fenêtre d'optimisation. Si l'utilisateur veut corriger la trajectoire, il doit cliquer sur le bouton *Corriger la trajectoire* pour afficher la fenêtre de correction. Lorsque l'utilisateur coche la case *spectre des erreurs des surfaces,* une petite fenêtre est apparaît pour permettre à l'utilisateur d'introduire la tolérance qu'il veut pour la vérification (Voir figure 13).

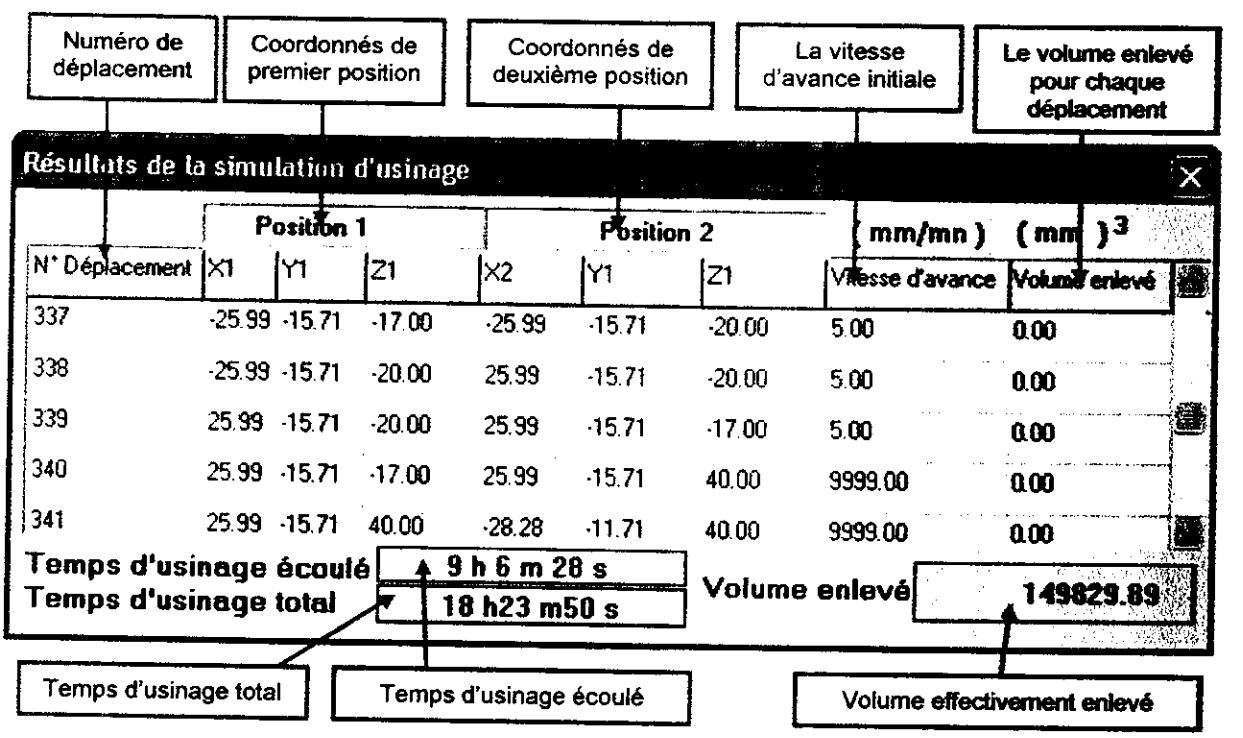


Figure 7: *Fenêtre d'affichage des résultats de simulation.*

La fenêtre de la figure 7 permet d'afficher certaines informations concernant le déroulement de la simulation d'ébauchage en temps réel, en affichant les coordonnées des deux positions successives de chaque déplacement et la vitesse initiale de ce déplacement ainsi que le volume enlevé. Il y a aussi une indication du temps d'usinage réellement écoulé et le temps d'usinage total nécessaire pour l'opération complète ainsi que le volume total enlevé après chaque déplacement d'outil au niveau de la pièce en cours d'ébauchage.

La clique sur le bouton *modifier couleur* permet d'afficher la forme montrer dans la figure 8. La fenêtre de changement de couleur permet de modifier les couleurs des sommets de surface et la couleur de ses projections, la couleur des triangles (surface, ou brut), couleur d'outil, de trajectoire, rendu de surface ou de brut...etc. De

même elle permet aussi de changer la taille de l'épaisseur pour le dessin des points et de segments.

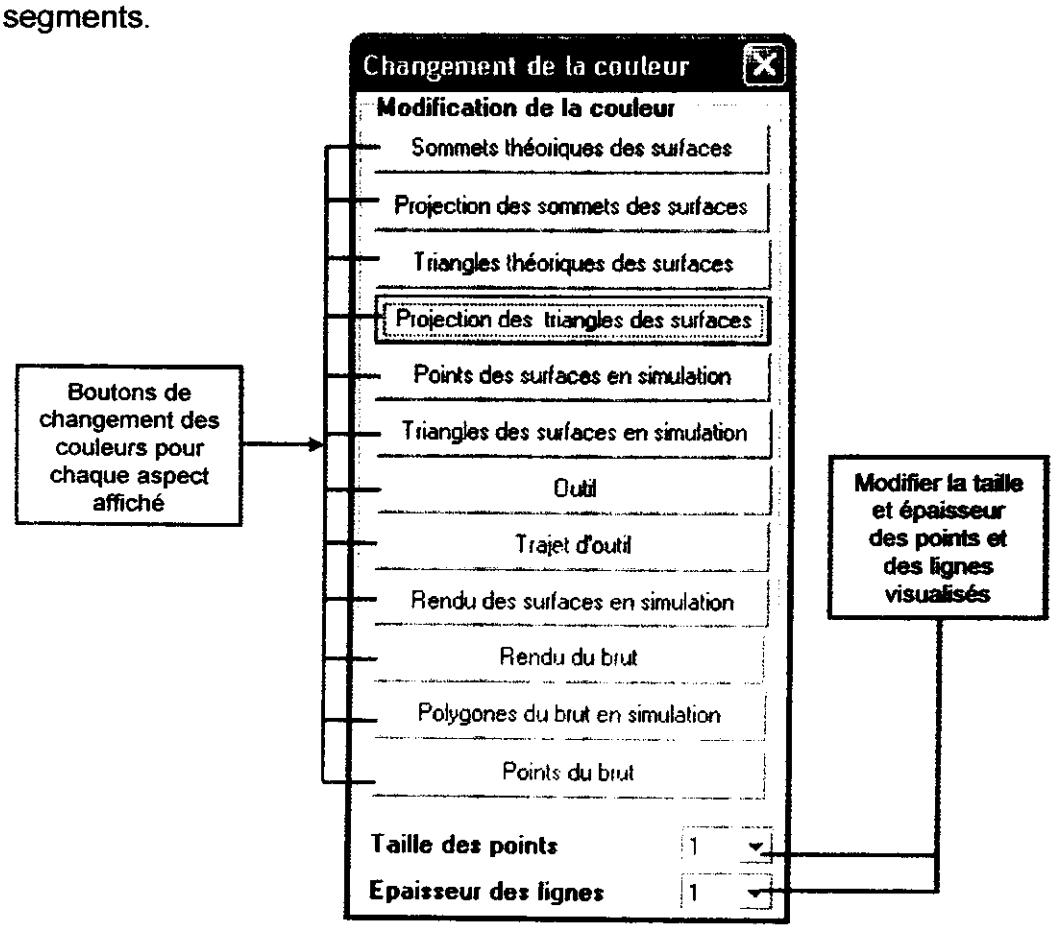


Figure 8: *Changement des couleurs des aspects affichés.*

Le bouton régler lumière nous affiche la fenêtre montrer dans la figure 9.

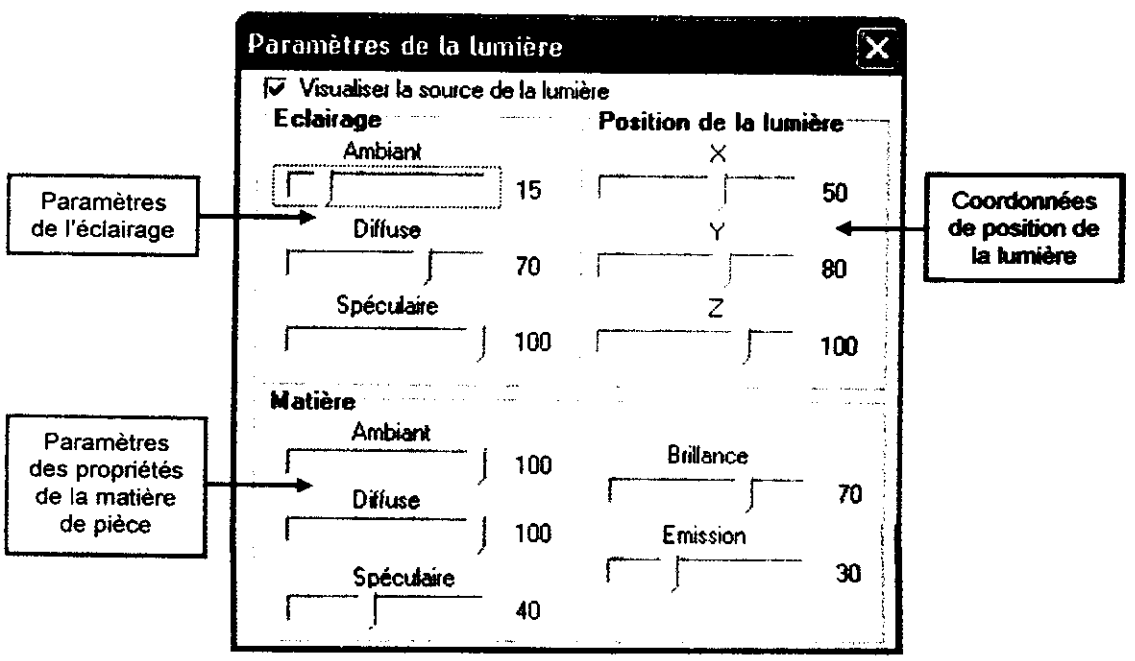


Figure 9: *Fenêtre de modification des paramètres de la lumière.*

La fenêtre de la figure 9 est divisée en trois parties principales. Une partie pour régler la position ou se trouve la source lumineuse en modifiant ses coordonnées cartésiens x, y, z. Une autre partie concernant les paramètres d'éclairage qui sont l'*ambiant* pour indiquer le milieu dans lequel on se trouve, la *diffuse* pour régler la clarté et la concision de l'environnement, et le *spéculaire* qu'est propre au miroir et à ses propriétés physique. La troisième partie est destinée aux propriétés de la matière et qui comporte les mêmes paramètres précédents avec deux autres paramètres qui s'ajoutent et qui sont la *brillance* pour régler la qualité de réflexion de la lumière pour la matière ainsi que son *émission* pour régler les productions de radiation. L'utilisateur a la possibilité de visualiser ou de cacher la source de lumière.

IV.4. Fenêtre d'adaptation des vitesses d'avance :

Les vitesses d'avance de l'outil sont initialement fixées une seule fois au début pour tout les déplacement sur la trajectoire d'outil. Les vitesses initiales ne sont pas optimales. Pour cela, cette fenêtre se charge de l'optimisation de ces vitesses pour qu'elles soient adaptées et par conséquence éviter les dégâts matériels, comme la casse de l'outil et l'endommagement de la pièce usinée ainsi que le gain de temps si cela soit possible. Les vitesses adaptées sont calculées en prenant en considération les dimensions de l'outil d'usinage (le rayon et la longueur) et la distance de chaque déplacement telle que ces paramètres sont utilisés pour calculer le volume enlever pour chaque déplacement. En prenant le volume en compte et les propriétés de matière de l'outil et de la pièce à usiner et certains autres paramètres, l'application calcule la vitesse d'avance nécessaire pour chaque déplacement sur la pièce en cours d'usinage. Cette fenêtre est montrée dans la figure 10.

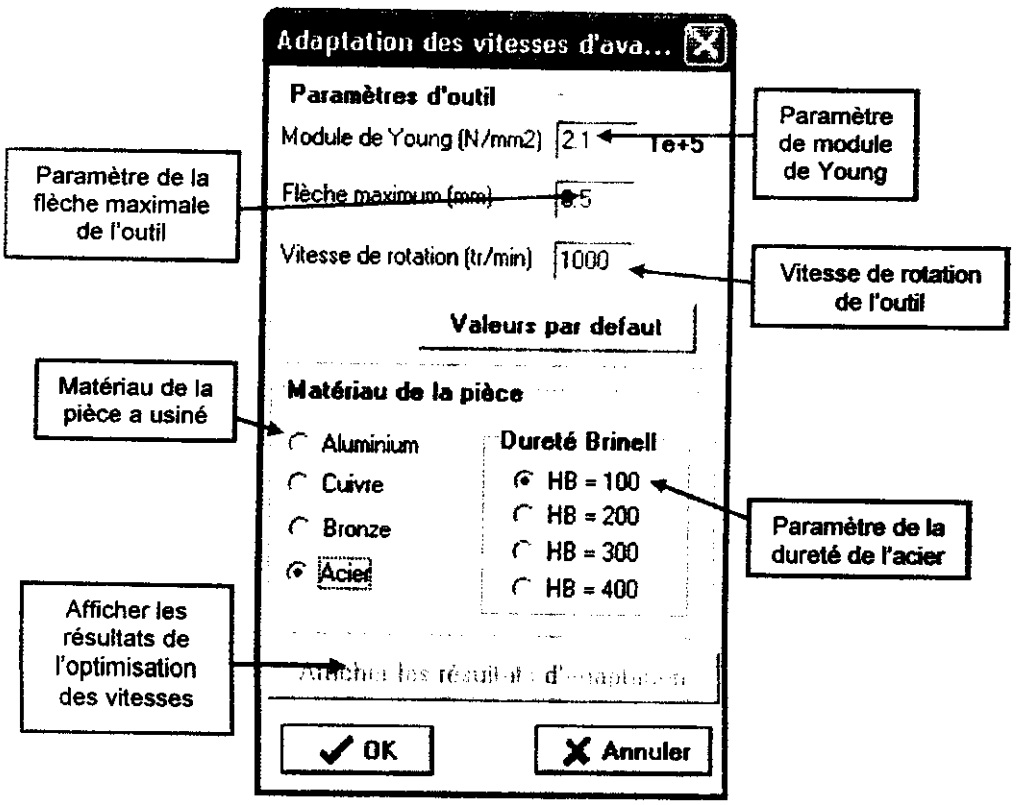


***Figure 10:** Fenêtre d'optimisation des vitesses d'avances d'outil.*

La fenêtre de la figure 10 comporte trois zones de saisie des trois paramètres suivants, module de Young, la flèche maximale de l'outil et le paramètre de la vitesse de rotation de l'outil qui sont par défaut initialisés à (210000 N/mm²) et (0.5 mm) et (1000 tours/min) respectivement ou en cliquant sur le *bouton Valeur par défaut*, mais l'utilisateur a la possibilité de les modifier. Cette fenêtre permet aussi de choisir le matériau de la pièce à usiner. Pour cela, il y a quatre types *Aluminium, Cuivre, Bronze* et *Acier* qui lui même est repartie en quatre catégories selon sa dureté, *100, 200, 300, 400* (ces valeur sont appelé la dureté de brinell). Donc, l'utilisateur n'à que cocher les cases de son choix, puis clique sur le bouton *OK* pour lancer l'opération d'adaptation des vitesses d'avance. Pour voir les résultats de l'optimisation l'utilisateur doit cliquer sur le bouton *Afficher les résultas de l'optimisation* (voir figure 11).

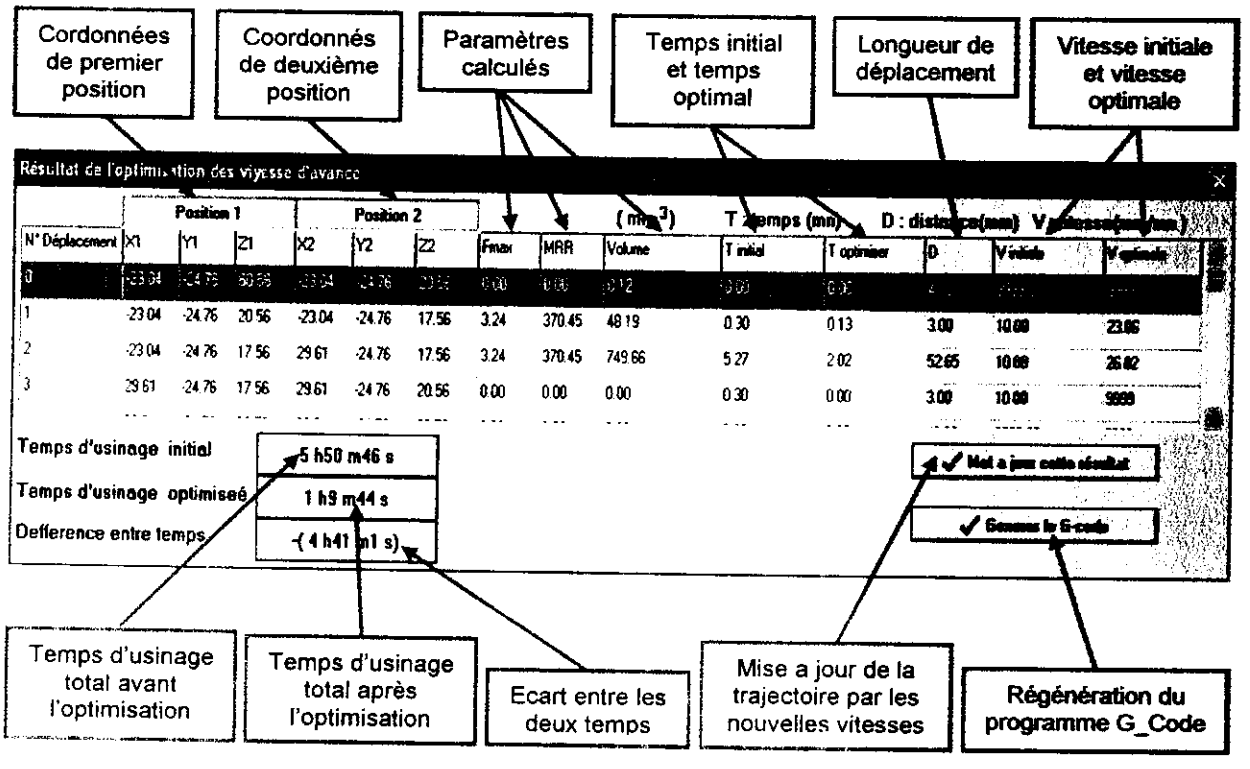


Figure 11: *Fenêtre d'optimisation des vitesses d'avance de l'outil.*

La fenêtre d'optimisation de la figure 11 permet de montrer plusieurs informations concernant l'optimisation telles que l'affichage pour chaque déplacement les différentes paramètres calculés comme la force de flexion maximale appliquée sur l'outil, le paramètre MMR, comme elle affiche aussi le volume enlevé avec indication du temps initial et optimal ainsi que la vitesse initiale et optimale pour chaque déplacement. Cette fenêtre nous affiche aussi le temps d'usinage initial et le temps d'usinage après l'optimisation. Il y a aussi une indication sur l'écart entre ces deux temps. Donc, il y a soit un gain de temps exprimer par (-) ou il y a un temps supplémentaire qui doit être consommé et qu'est exprimer par (+). L'utilisateur peut cliquer sur le bouton *Met a jour ce résultat* pour faire une mise à jour de la trajectoire d'usinage pour qu'elle soit adaptée aux nouveaux paramètres et il peut aussi cliquer sur le bouton *Générer le G-Code* pour appeler le processus de régénération du programme d'usinage dû à la nouvelle trajectoire et affiché sa fenêtre (voir figure 12).

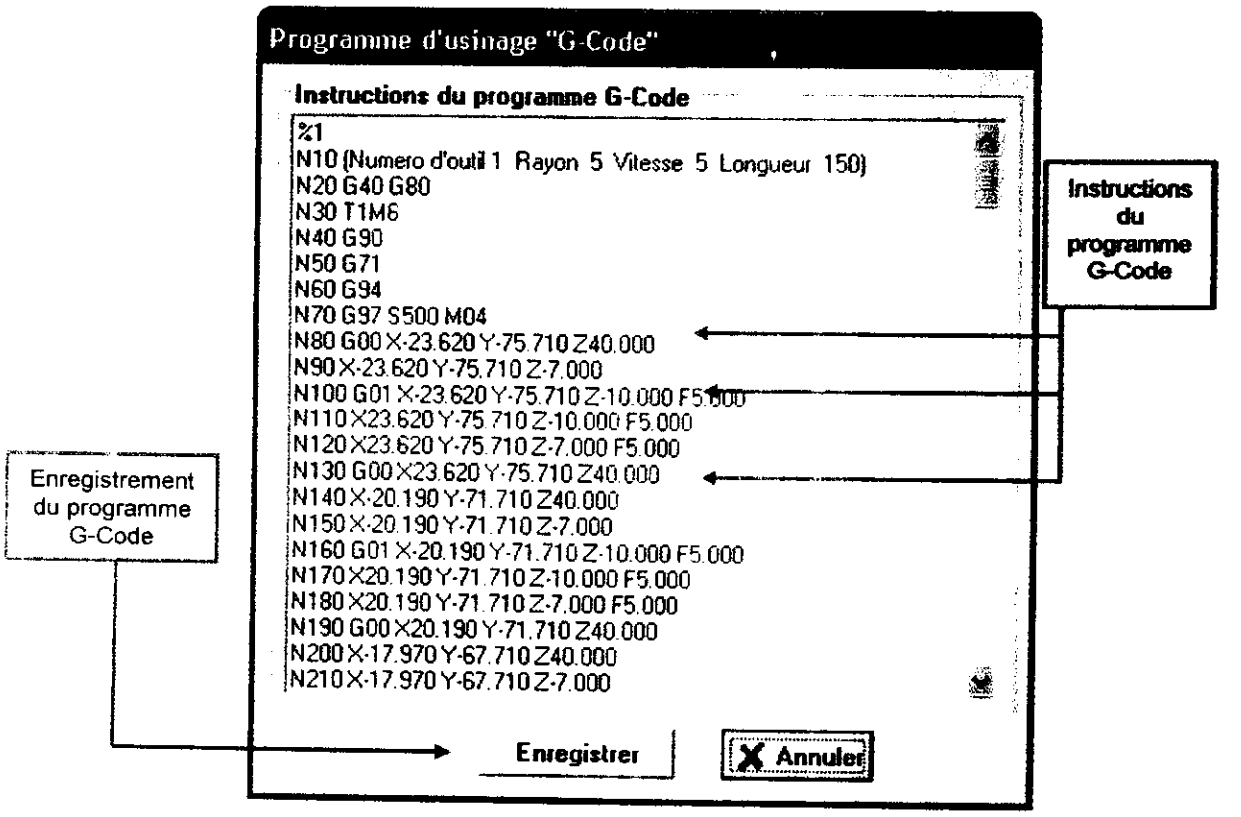


Figure 12: *Fenêtre d'affichage de programme G-Code.*

La fenêtre de la figure 12 permet d'afficher tout le programme G-Code nécessaire pour l'usinage de la pièce désirée et qui comporte des milliers d'instructions. Ce programme peut être enregistré en cliquant sur le bouton *Enregistrer*.

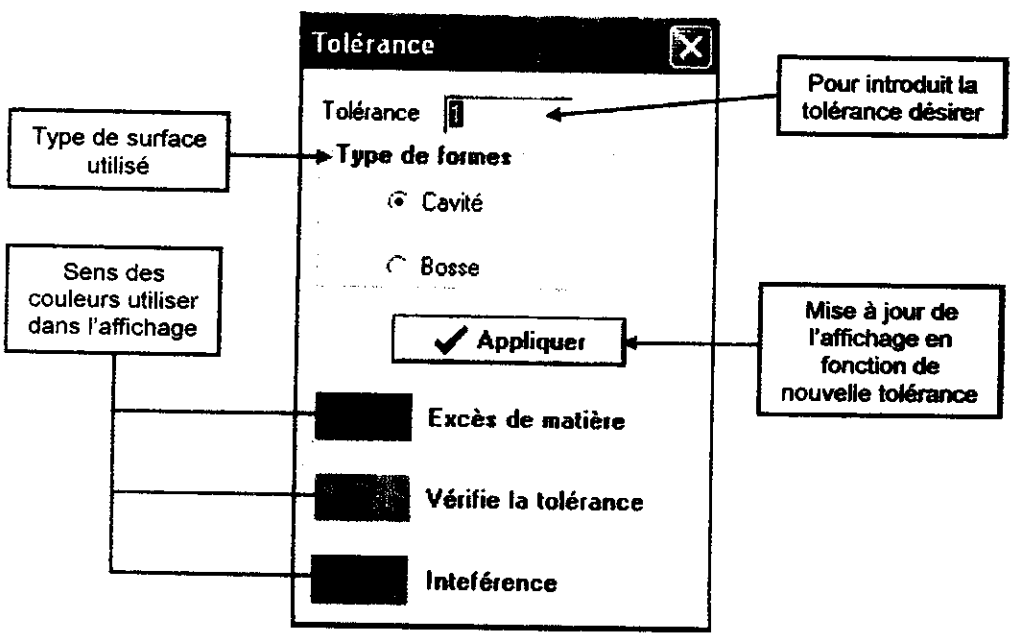


Figure 13: *Fenêtre d'introduction de tolérance.*

La fenêtre de la figure 13 permet à l'utilisateur d'introduire la valeur de la tolérance qu'il veut et le type de surface soit une cavité ou une bosse. Une fois

choisie, l'utilisateur doit cliquer sur le bouton *Appliquer* pour appeler le processus de vérification de tolérance et mettre à jour la simulation de la pièce pour afficher les couleurs des différentes zones de la pièce usinée en affichant les zones ou il y a un excès de matière en bleu et les zones ou il y a une interférence en rouge tandis que les zones qui sont dans l'intervalle de tolérance seront affichés en vert.

IV.5. Fenêtre de correction de la trajectoire :

C'est la fenêtre qui permet de faire la correction de la trajectoire d'usinage en éliminant les positions qui provoquer des interférences avec la surface ou avec le brut et qui sont dessinées en rouge. Cette fenêtre est montrée par la figure 14.

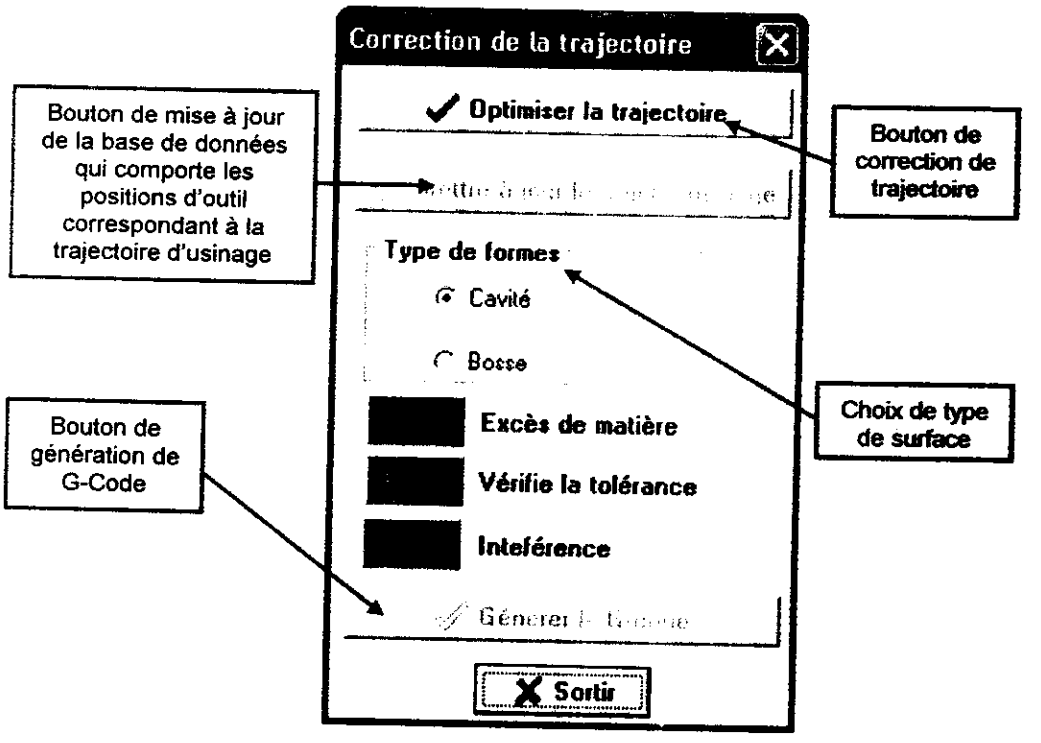
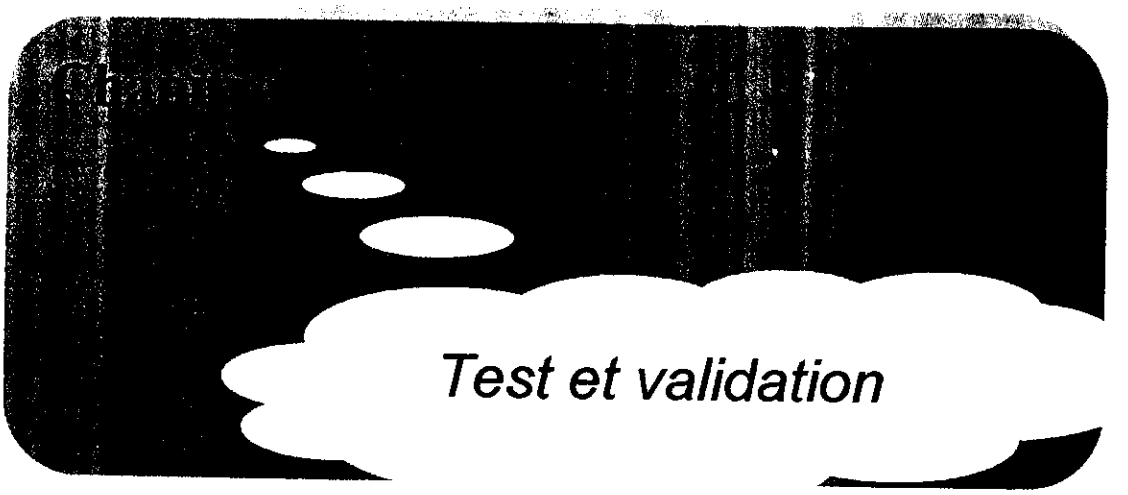


Figure 14: *Fenêtre de correction de la trajectoire d'usinage.*

La fenêtre de la figure 14 permet à l'utilisateur de corriger la trajectoire d'usinage en cliquant sur le bouton *Optimiser la trajectoire* après le choix du type de surface (Cavité ou Bosse). Une foi le trajectoire est corrigée l'utilisateur peut la sauvegarder en un seul clique sur le bouton *Mettre à jour le trajet d'usinage* puis cliquer sur le bouton *Sortir* pour fermer cette fenêtre.

V. CONCLUSION :

Dans ce chapitre nous avons présentés toutes les formes que nous avons crées permettant de simuler et vérifier l'usinage, corriger la trajectoire d'outils et par la suite adapter la vitesse d'avance de l'outil et en terminant par la génération du programme d'usinage optimisé « G-Code ». Dans le chapitre suivant, nous allons tester et valider notre application.


Test et validation

I. INTRODUCTION :

La dernière étape dans notre démarche de développement et d'écriture du code associé à notre système, c'est l'étape de tests et de validations des résultats des différentes étapes précédentes. Cette étape est très importante puisqu'elle permet de valider le logiciel par rapport au cahier de charges et de faire les corrections nécessaires en cas de problèmes qui peuvent apparaître pendant l'exécution.

II. TESTS ET VALIDATION :

Cette phase de validation consiste à simuler et tester les cas d'utilisation de l'usinage à l'aide de déroulement des scénarios représentés par le diagramme de séquence (voir figure 1).

✓ Cas d'utilisation de simulation de l'opération d'usinage des surfaces gauches :

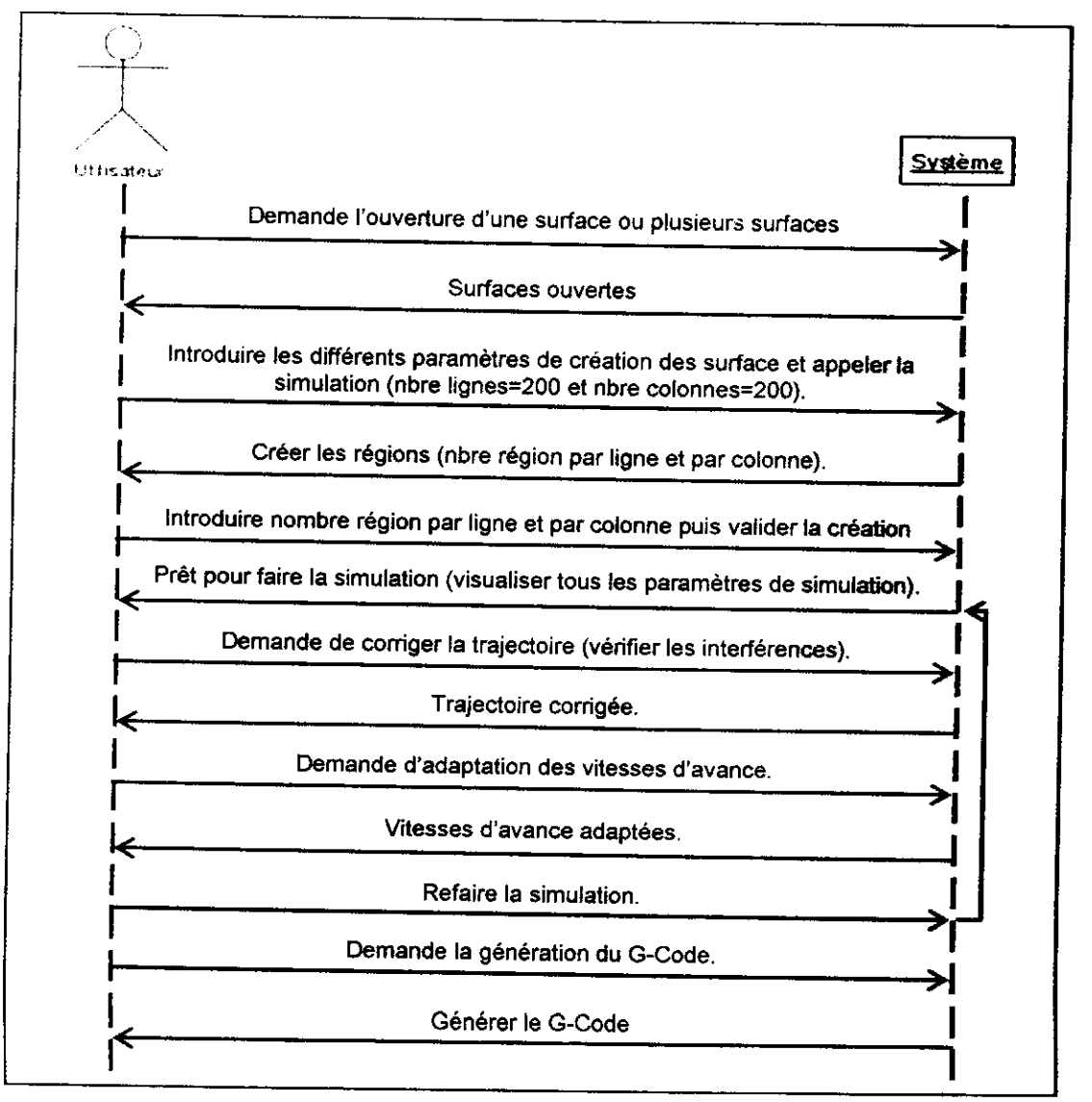


Figure 1: *Scénario de simulation, vérification et adaptation d'opération d'usinage.*

Dans ce qui suit, nous allons montrer un exemple d'application. Considérons la surface gauche (empreinte d'un moule de demi vase) montrée par la figure 2.

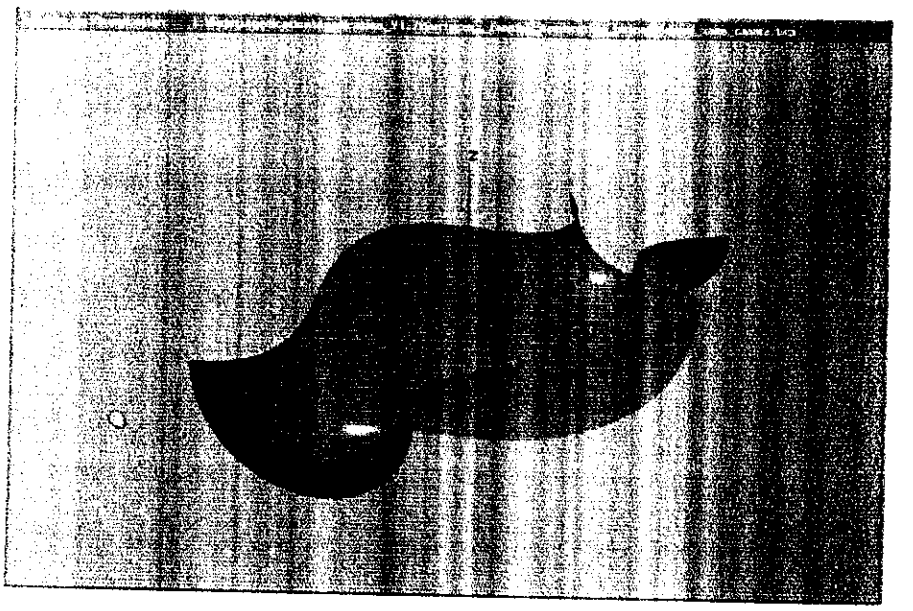


Figure 2: *Surface considérée (demi vase cavité).*

La trajectoire d'usinage de cette pièce est générée avec les paramètres d'usinage suivants :

- Le rayon de l'outil d'usinage est égal à 4 mm.
- La longueur d'outil est égale à 200 mm.
- La vitesse de rotation de l'outil est égale à 1000 tr/min.
- Le sens de rotation de l'outil est trigonométrique.
- La profondeur de passe (distance entre deux plans) est égale à 5 mm.
- La distance entre deux passes est égale à 4 mm.
- La vitesse d'avance initiale est égale à 10 mm/min.

Nous avons définis les différents paramètres et propriétés utilisées dans la simulation comme suit :

- Le nombre de lignes est égal à 200 lignes.
- Le nombre de colonnes est égal à 200 colonnes.

Avec ces données nous avons les résultats suivants :

- La hauteur maximale entre la surface théorique et la surface approximée est égale à 0.016 mm.
- La longueur maximale des segments des triangles est égale à 1.93 mm.

En utilisant 200 lignes et 200 colonnes pour les sommets, le nombre de sommets est égal à 201 x 201 = 40401 sommets, et par conséquent le nombre de triangles sera égal à 200 x 200 x 2 = 80000 triangles. Nous avons trouvés une erreur

maximum égal à 0.016 mm et une longueur maximum d'un segment égale à 1.93 mm.

Pour l'optimisation nous prenons l'erreur max=0.016 mm et longueur max=1.93 mm puis nous cliquons sur le bouton *Optimiser lignes et colonnes* de la fenêtre de triangulation et nous obtenons les résultats suivants :

Nombre de lignes =125
Nombre de colonnes =178

Pour afficher les résultats de l'optimisation, il suffit de cliquer sur le bouton résultat de triangulation (voir la figure 3).

Cette optimisation permet donc de réduire le nombre de lignes et de colonnes et par conséquent le nombre de triangles nécessaires pour approximer la surface (espace mémoire nécessaire au stockage des données). De même, cette optimisation permet de réduire le temps de simulation et d'adaptation des vitesses d'avance.

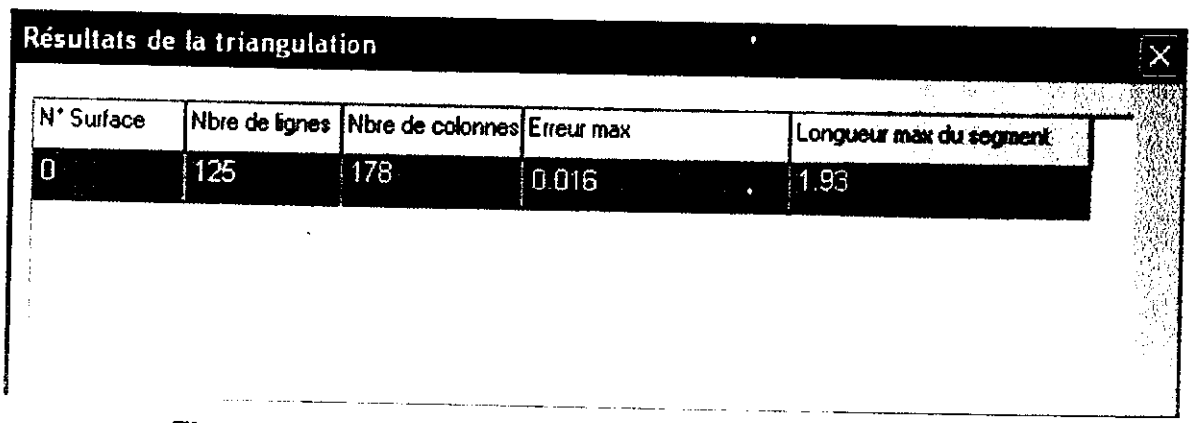


Figure 3: *Résultat de l'optimisation des lignes et des colonnes.*

Dans la suite, nous allons prendre les premières valeurs choisies c'est-à-dire 200 lignes et 200 colonnes.

En cliquant sur le bouton simulation, la fenêtre création des régions s'ouvre pour fixer le nombre des régions par ligne et par colonne à utiliser dans la simulation. Les valeurs introduites sont les suivantes :

- Le nombre de régions suivant X est égal à 10.
- Le nombre de régions suivant Y est égal à 10.

Ces régions sont montrées dans la figure 4. Pour différencier les régions, nous avons donnés à chaque région une couleur différente. Ces régions seront utilisées dans les étapes qui suivent.

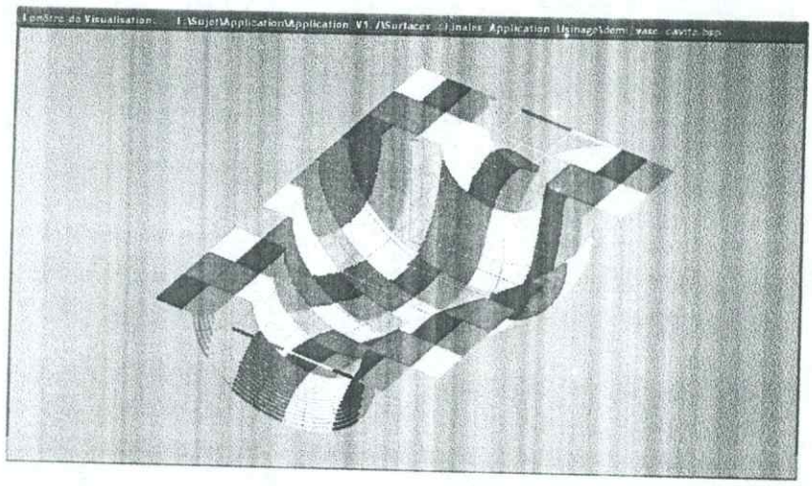


Figure 4: *Régions de la surface considérée (demi vase).*

Après la création des régions, le processus crée le brut de la pièce d'une manière automatique avant le lancement de la simulation de l'opération d'ébauchage (voir figure 5). Nous avons au début, une pièce parallélépipédique qui représente une pièce brute que nous voulons usiner. Les dimensions du ce brut sont :

Longueur = 160.75 mm,
Largeur = 102.65 mm,
Hauteur = 51.32 mm.

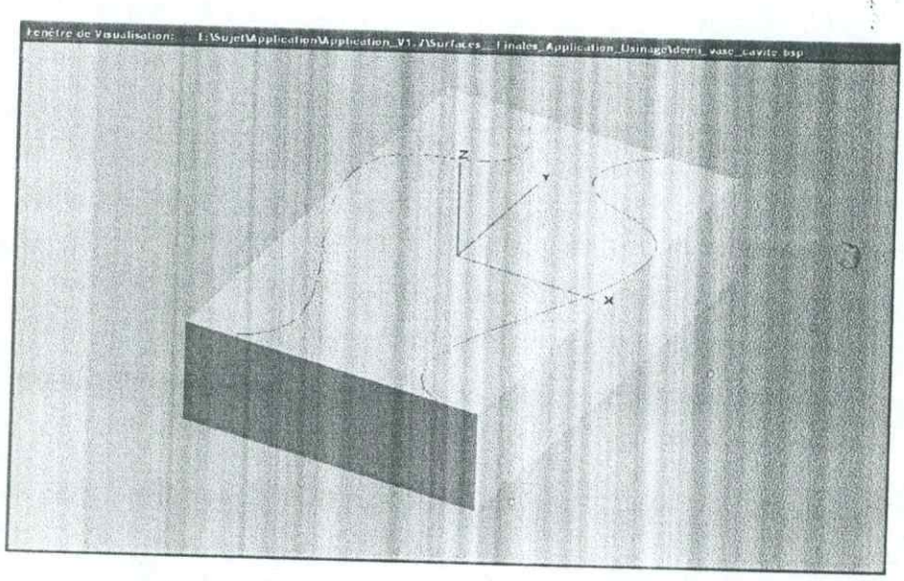


Figure 5: *Pièce brute à usiner en simulation.*

En cliquant sur le bouton *Simuler l'ébauchage* de la fenêtre de simulation, l'outil d'usinage commence à enlever la matière de la pièce brute en vue de réaliser une empreinte qui a la forme d'un demi vase (voir figure 1). Le début de la simulation est donné par la figure 6.

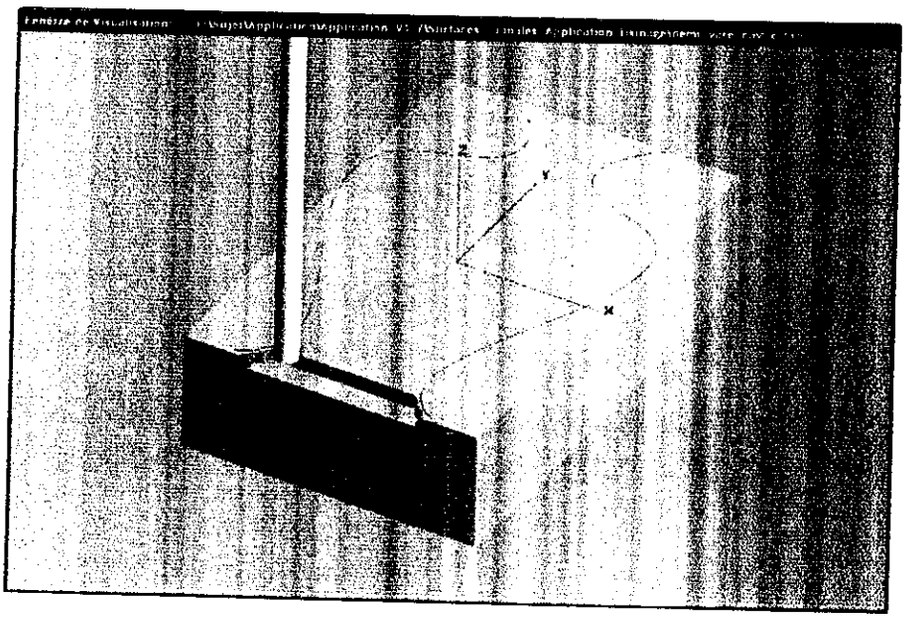


Figure 6: *Début de simulation.*

La figure 7 montre la forme de la pièce après 50% de travail d'usinage.

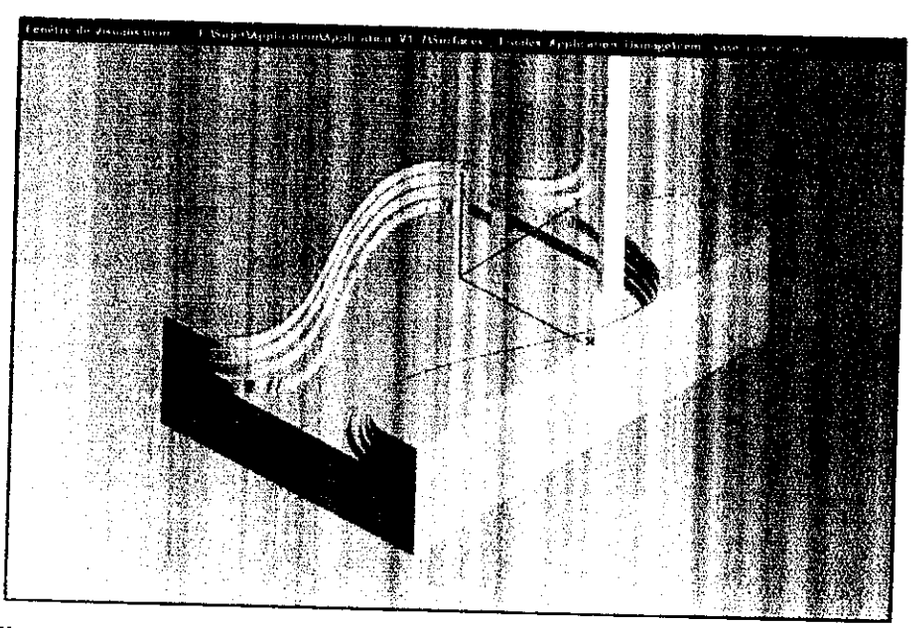
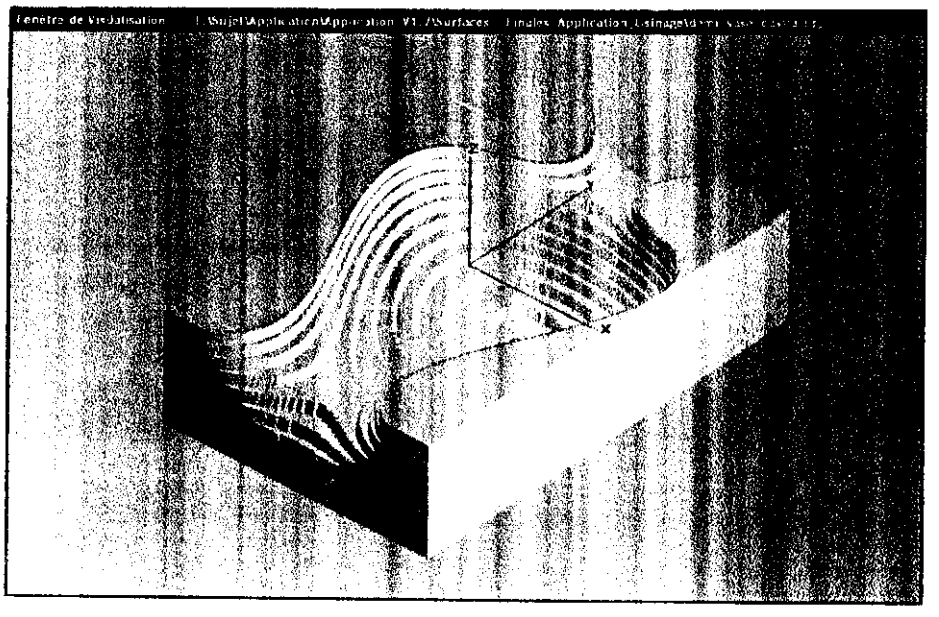


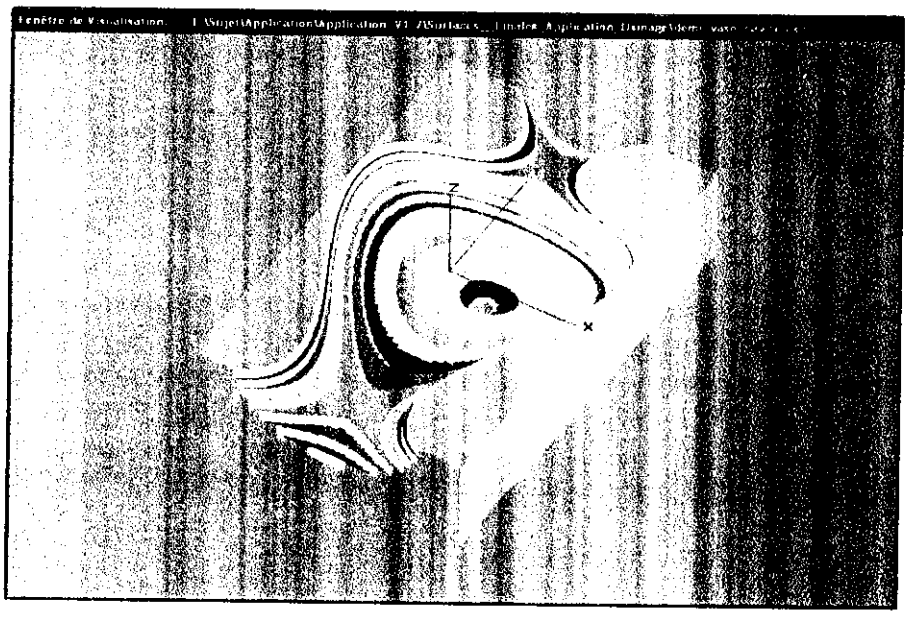
Figure 7: *Etat de simulation après 50% de temps total d'usinage.*

La figure 8 montre l'aspect final de la pièce après la fin de la simulation de l'ébauchage. L'utilisateur a la possibilité de refaire la simulation en tout moment pendant l'opération de simulation quelque soit l'état de la pièce.

Il est nécessaire de signaler que pendant la simulation, l'utilisateur a la possibilité d'afficher (visualiser) plusieurs autres aspects tels que le trajet d'outil, l'état des triangles, état de spectre (la tolérance), différents niveaux,...etc.



Dans cet état final de simulation, nous voyons bien la cavité réalisée dans la pièce brute. Nous pouvons constater les différents niveaux (escaliers) de l'ébauchage laissés pour la phase de demi finition et finition. Ces escaliers sont distribués sur 10 niveaux d'ébauchage (voir figure 9).



Notre simulation permet aussi de distinguer les différentes zones et de les visualiser avec des couleurs différentes : en rouge les zones des interférences, en bleu les zones d'excès de matière et en vert les zones qui sont dans l'intervalle de tolérance. En introduisant une tolérance égale à 1mm, les différentes zones sont représentées par la figure 10.

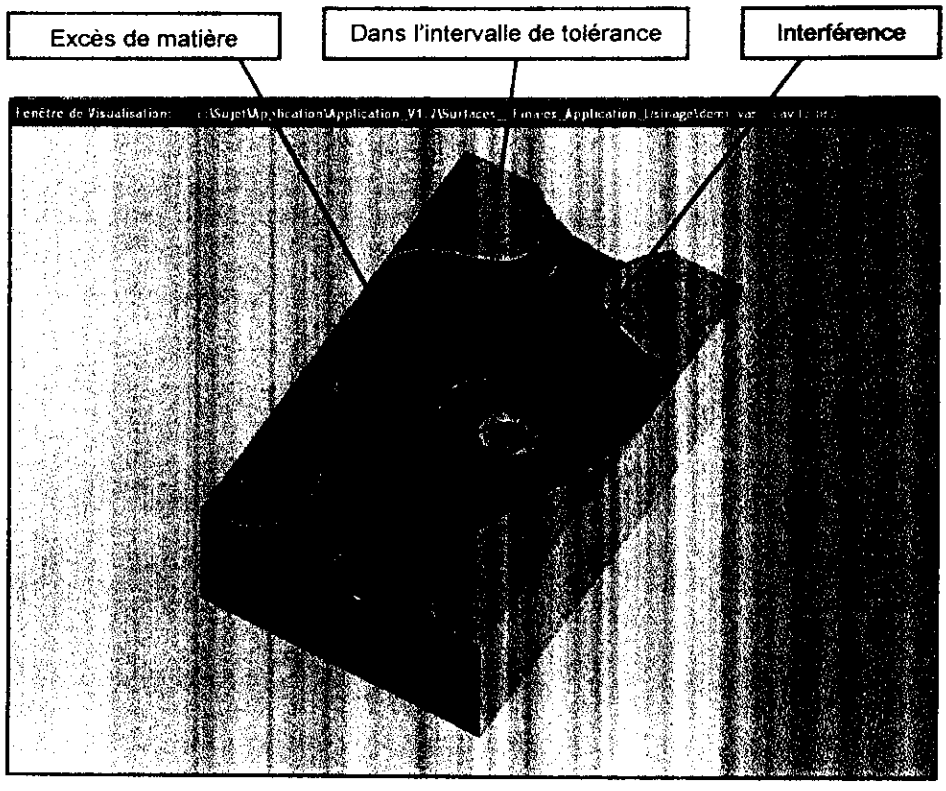


Figure 10: *Différentes zones détectées.*

Dans la fenêtre de la figure 10, nous voyons que la majorité de la surface est en bleu ce qui indique qu'il y a des excès de matière. La matière laissée sera enlevée dans les usinages de demi finition et finition. De même, il y a aussi des zones qui sont dans l'intervalle de tolérance exigée en vert et des zones d'interférence en rouge. Ces zones d'interférences seront éliminées par la suite.

L'élimination des zones d'interférences, peut être faite en cliquant sur le bouton *corriger la trajectoire*. Dans ce cas, une petite fenêtre apparaît pour inviter l'utilisateur à spécifier le type de surface (Cavité ou Bosse) et demander de fixer la valeur de l'intervalle de tolérance pour déterminer les différentes zones.

Dans notre exemple, nous avons choisi le type cavité et la valeur de 1 mm pour la tolérance. En cliquant sur le bouton *Optimiser la trajectoire,* les différents erreurs sont corrigés. Par la suite, nous pouvons sauvegarder la nouvelle trajectoire corrigée en cliquant sur le bouton *mettre à jour le trajet d'usinage.*

La figure 11 montre le résultat de la simulation avec la nouvelle trajectoire corrigée où nous voyons bien l'absence de zones d'interférences.

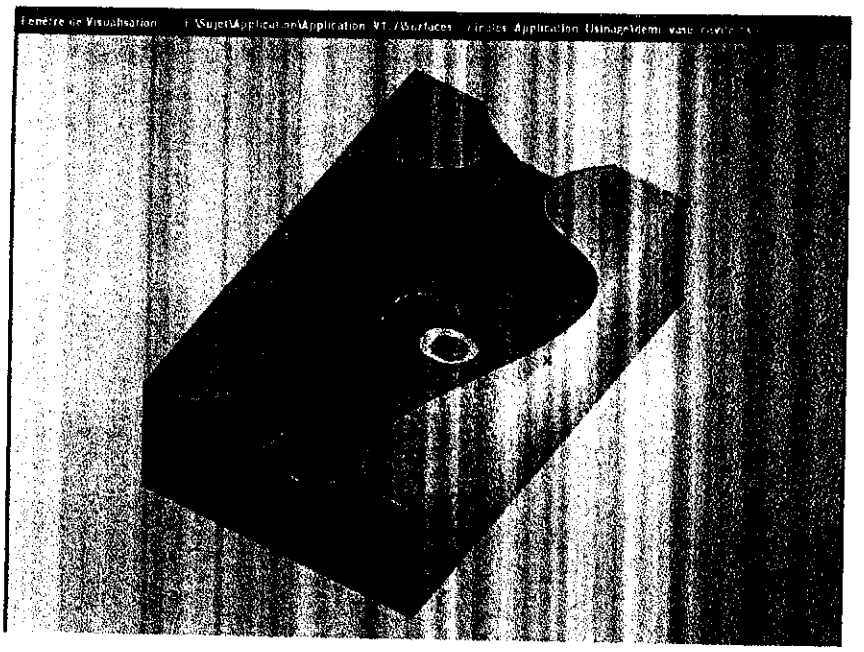


Figure 11: *Simulation d'une trajectoire corrigée.*

Pour afficher les résultats de la simulation en temps réel (quantité de matière enlevée, temps d'usinage écoulé, temps total nécessaire), il suffit de cliquer sur le bouton *Résultas de la simulation* (voir figure 12).

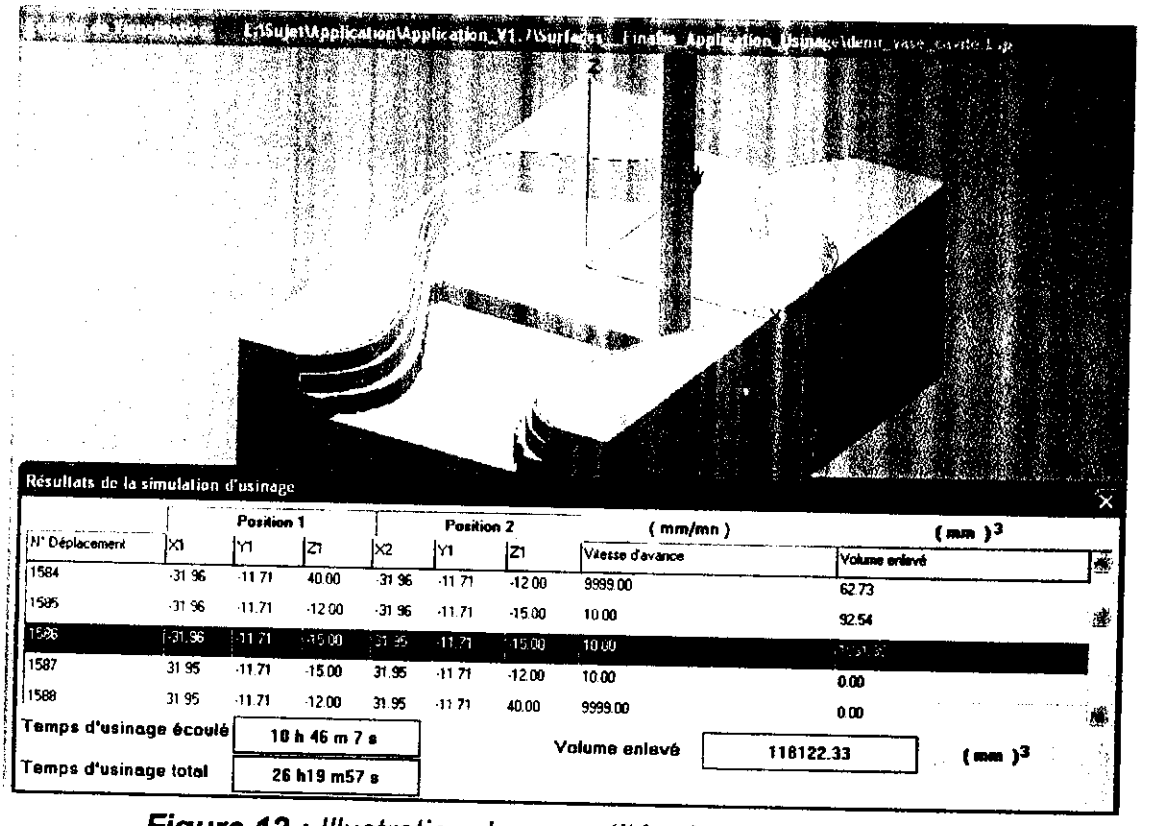


Figure 12 : *Illustration des quantités de matière enlevées.*

Pour adapter les vitesses d'avance de l'outil en fonction de la quantité de matière enlevée, il suffit de cliquer sur le bouton *optimiser les vitesses d'avances.* Cette adaptation se fait en fonction des paramètres introduits par l'utilisateur et qui sont :

Type de matière : Aluminium,
Longueur d'outil =200 mm,
Rayon d'outil = 4 mm.

Les résultats de l'optimisation sont montrés par la figure 13.

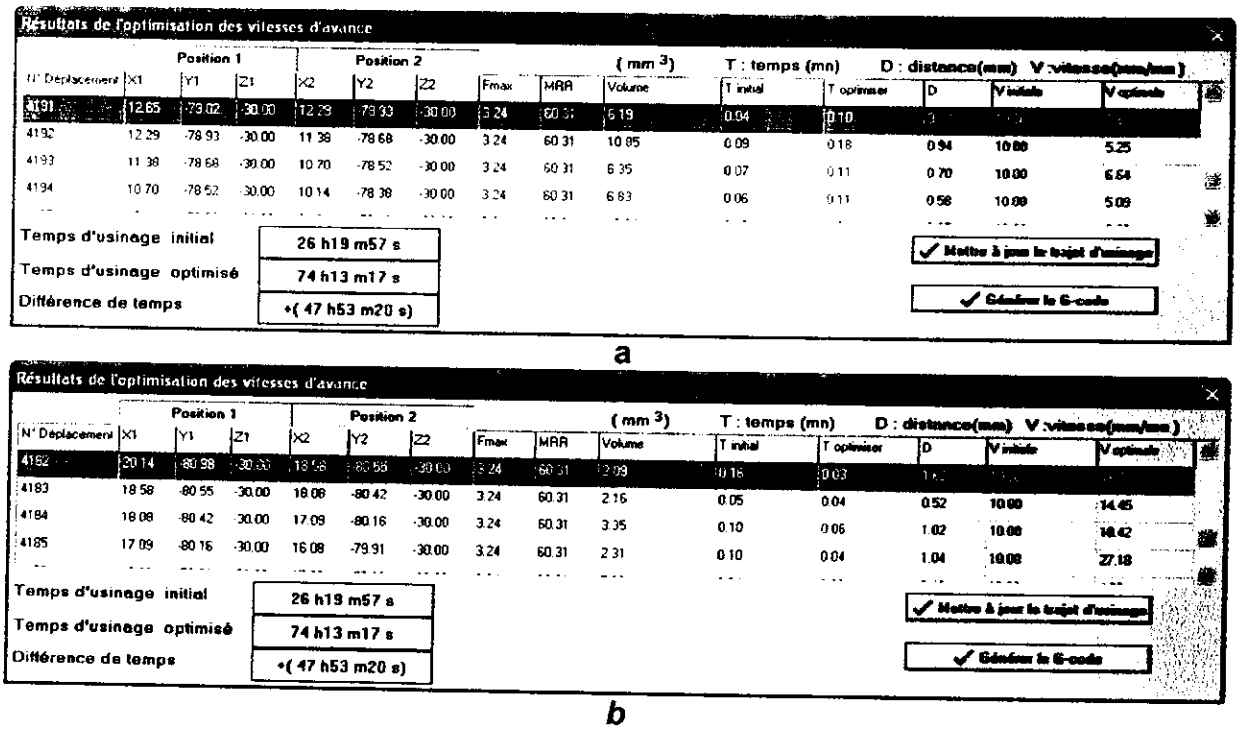


Figure 13 : *Adaptation des vitesses d'avance.*

Les figures 13.a et 13.b montrent les nouvelles valeurs des vitesses d'avance des différents déplacements qui sont soit augmentées soit réduites par rapport à la vitesse initiale. Dans cet exemple, nous voyons que le temps d'usinage total adapté est de 74h13m17s au lieu de 26h19m57s initial, ce qui donne un écart de +47h13m17s.

Maintenant, si nous modifions quelques paramètres, nous voyons qu'il y a un changement. Si nous posons que la vitesse de rotation de l'outil est de 5000 tour/min au lieu de 1000 tour/min et que les autres paramètres restent sans changement, alors les résultats sont exprimés par la figure 14.

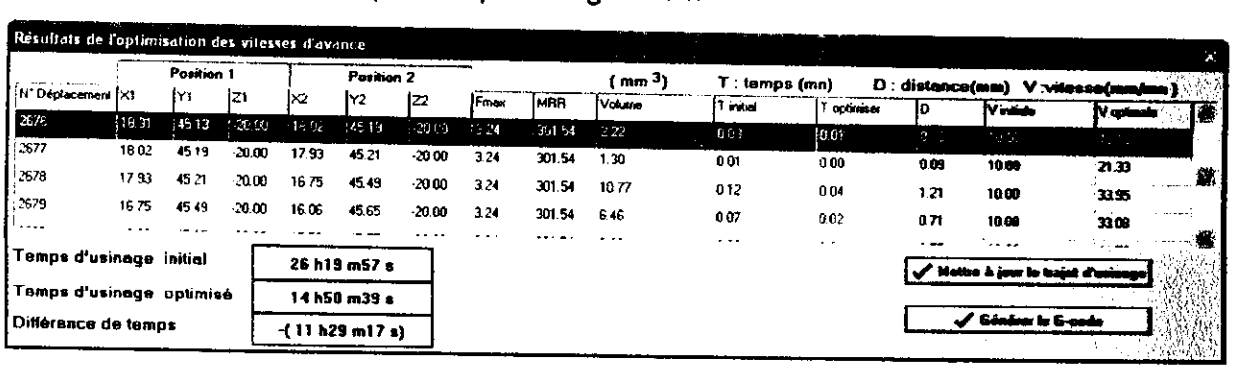


Figure 14 : *Réduction du temps total d'usinage*

Dans la fenêtre de la figure 14, nous remarquons que le temps total d'usinage devient 14h50m39s au lieu de 26h19m57s. Dans ce cas nous voyons qu'il y a une réduction du temps d'usinage de 11h29m17s.

Après toutes les vérifications, corrections et optimisations faites, il faut maintenant générer le programme d'usinage G-Code à transmettre à la machine d'usinage. Pour cela, il faut cliquer sur le bouton *générer le G-Code* de la fenêtre d'optimisation des vitesses. Le début de ce programme G-code est montré par la figure 15.

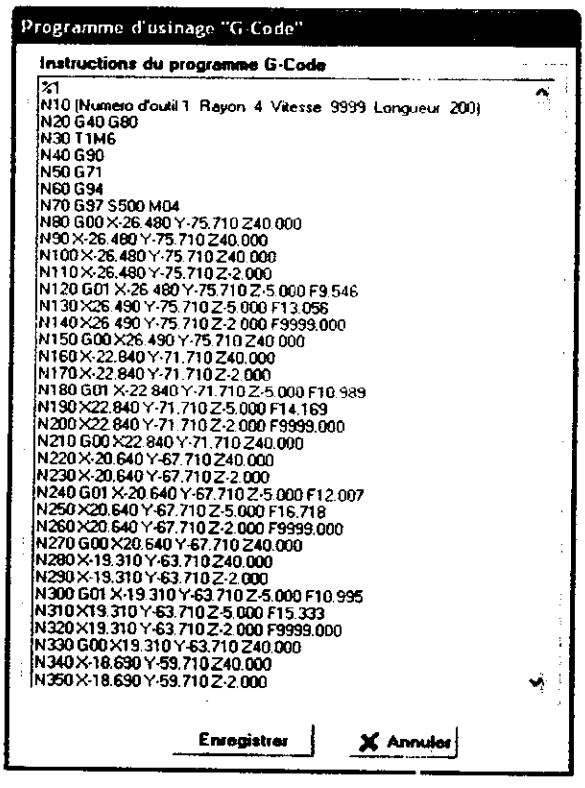


Figure 15 : *Génération de programme G-Code*

Pour enregistrer ce programme dans un fichier, il suffit de cliquer sur le bouton *enregistrer.*

III. CONCLUSION :

Dans ce chapitre, nous avons pris l'exemple d'une surface et nous avons testés et validés toutes les fonctions développées de la simulation d'enlèvement de matière jusqu'à l'adaptation des vitesses d'avances et la génération du programme d'usinage G-Code. Ces tests montrent l'importance et l'apport de l'outil informatique dans l'automatisation de l'usinage des surfaces gauches.

CONCLUSION GENERALE

Le travail que nous avons présenté est relatif à l'opération d'ébauchage des surfaces gauches sur des fraiseuses à commande numérique à 03 axes et qui consiste à automatiser la simulation d'enlèvement de matière, vérifier et corriger le trajet d'usinage ainsi qu'une adaptation de la vitesse d'avance de l'outil.

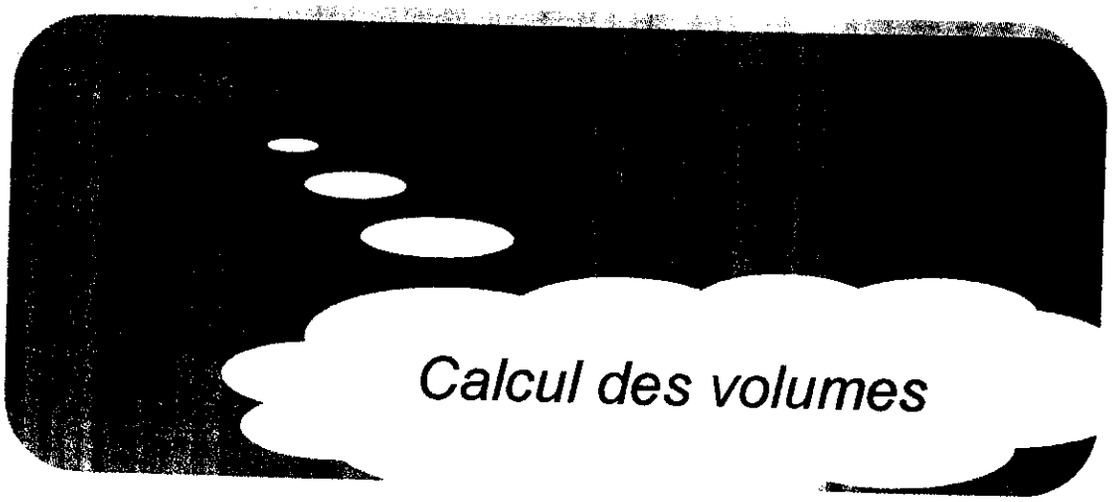
Dans ce mémoire, nous avons étudiés en premier lieu les méthodes de conception et de représentation des surfaces gauches en appuyant notre étude sur les surfaces B-Spline et NURBS. Ensuite, nous avons présentés les machines utilisées dans l'usinage de ces surfaces et la syntaxe des programmes G-Code. A la fin, nous avons montré les différentes fonctions offertes par l'application logicielle développée suivi d'un test de validation.

Le résultat de notre application est l'enrichissement de l'application logicielle graphique avec des fonctions qui permettent de :

- ✓ Approximer les surfaces gauches avec un nombre minimum de triangles.
- ✓ Simuler l'opération d'ébauchage avec visualisation des différents aspects (l'outil, trajectoire, pièce, ...etc.).
- ✓ Détecter les différentes zones en fonction de la tolérance imposée.
- ✓ Calculer le temps d'usinage et la quantité de matière enlevé en temps réel.
- ✓ Vérifier et corriger la trajectoire d'usinage.
- ✓ Adapter les vitesses d'avance en fonction des quantités de matière enlevées.
- ✓ Automatiser la génération des programmes d'usinage G-Code.

En perspective de notre travail, nous recommandons de traiter les points suivants :

- ✓ Ajouter la machine outil à l'aspect visuel de la simulation pour approcher l'opération réelle.
- ✓ Usinage des surfaces gauches à partir d'un nuage de points.
- ✓ Adapter les vitesses d'avance lors de l'usinage en finition avec des outils hémisphériques, cylindriques et toriques.
- ✓ Considérer l'usinage en 5 axes.



Calcul des volumes

Annexe A

CALCUL DES VOLUMES [30]

Les trois cas de volumes à calculer sont les suivants :

Premier cas : si les trois points sont dans le même plan et sont abaissés d'une hauteur h, nous devons calculer le volume d'un prisme droit à base rectangulaire (voir figure A.1).

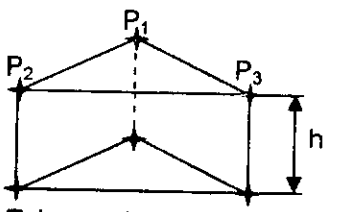


***Figure A.1** : Prisme droit à base rectangulaire.*

Etant donné les points $P_1(x_1, y_1)$, $P_2(x_2, y_2)$ et $P_3(x_3, y_3)$, le volume de ce prisme est donné par :

$$v = \left| \frac{1}{2} \left[x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2) \right] \right| \times h \qquad (1)$$

Deuxième cas : si deux points sont abaissés, alors le volume que nous devons calculer est le volume d'une pyramide qui est défini comme suit :

Le volume d'une pyramide est la somme des volumes de deux tétraèdres (voir figure A.2).

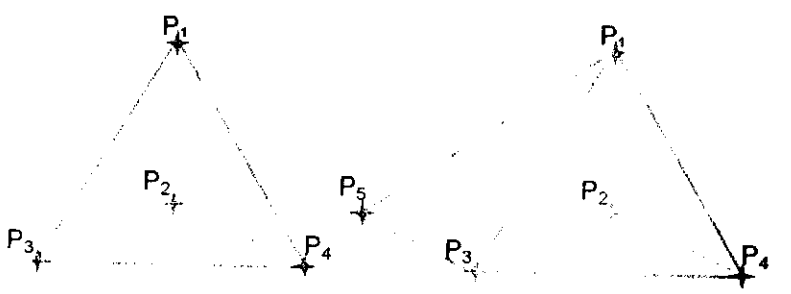


***Figure A.2** : Pyramide et tétraèdre.*

Si les points $P_1(x_1, y_1, z_1)$, $P_2(x_2, y_2, z_2)$, $P_3(x_3, y_3, z_3)$ et $P_4(x_4, y_4, z_4)$ sont les sommets d'un tétraèdre, son volume est donné par :

$$v = \frac{1}{6} \begin{vmatrix} x1 - x2 & y1 - y2 & z1 - z2 \\ x1 - x13 & y1 - y3 & z1 - z3 \\ x1 - x4 & y1 - y4 & z1 - z4 \end{vmatrix} \qquad (2)$$

Donc, si les points $P_1(x_1, y_1, z_1)$, $P_2(x_2, y_2, z_2)$, $P_3(x_3, y_3, z_3)$, $P_4(x_4, y_4, z_4)$ et $P_5(x_5, y_5, z_5)$ sont les sommets d'une pyramide, alors son volume est égal au volume du tétraèdre défini par les quatre points $P_1(x_1, y_1, z_1)$, $P_2(x_2, y_2, z_2)$, $P_3(x_3, y_3, z_3)$, $P_4(x_4, y_4, z_4)$ plus le volume du tétraèdre défini par les quatre points $P_1(x_1, y_1, z_1)$, $P_2(x_2, y_2, z_2)$, $P_3(x_3, y_3, z_3)$, $P_5(x_5, y_5, z_5)$.

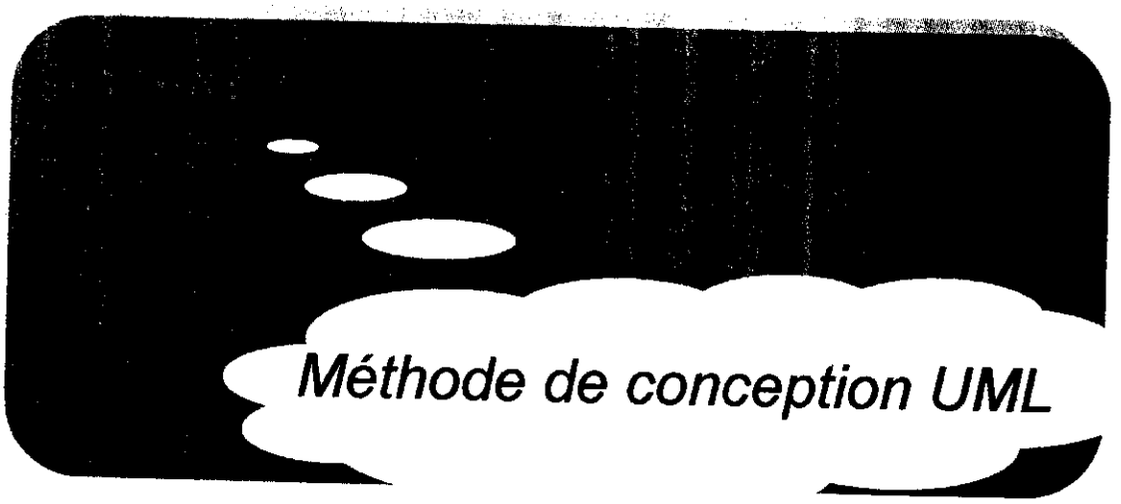
Nous pouvons aussi calculer le volume de la pyramide par la formule générale suivante :

$$v = \frac{1}{3} \times h \times s \qquad (3)$$

Où :

h : est la hauteur de la pyramide.
s : est la surface de la base de la pyramide.

Troisième cas : si un seul point est abaissé, alors le volume que nous devons calculer est le volume du tétraèdre qui est déjà défini dans le deuxième cas.


Méthode de conception UML

Annexe B

UML « Unified Modeling Language »

1. Historique [29]:

A partir de 1994, Rumbaugh et Booch (rejoints en 1995 par Jacobson) ont unis leurs efforts pour mettre au point la méthode unifiée (unified method 0.8), incorporant les avantages de chacune des méthodes précédentes.

La méthode unifiée à partir de la version 1.0 devient l'UML (Unified Modeling Language), une notation universelle pour la modélisation objet.

UML 1.0 est soumise à l'OMG (Object Management Group) en janvier 1997, mais elle ne sera acceptée qu'en novembre 1997 dans sa version 1.1, date à partir de laquelle UML devient un standard international.

Voici le récapitulatif des évolutions de ce langage de modélisation :

- En 1995: Méthode unifiée 0.8 (intégrant les méthodes Booch'93 et OMT)
- En 1995: UML 0.9 (intégrant la méthode OOSE)
- En 1996: UML 1.0 (proposée à l'OMG)
- En 1997: UML 1.1 (standardisée par l'OMG)
- En 1998: UML 1.2
- En 1999: UML 1.3
- En 2000: UML 1.4
- En 2003: UML 1.5

2. Introduction [27]:

L'Unified Modeling Language (UML) s'impose progressivement comme un standard de fait pour la modélisation à objets de systèmes, qu'ils soient logiciels, matériels ou organisationnels. La notation UML est suffisamment complète pour pouvoir remplacer ou compléter les notations à objets l'ayant précédée, et sa souplesse permet de l'utiliser pour modéliser toutes les facettes d'un système, depuis la phase d'identification des besoins jusqu'au test et au déploiement final du système opérationnel.

3. Définition [28]:

L'UML (Unified Modeling Language) est la notation standard qui s'est imposée pour la modélisation de systèmes informatiques. Elle permet de spécifier, de visualiser, de construire et de documenter l'ensemble des artefacts du système et s'applique aussi bien aux systèmes d'information qu'aux systèmes logiciels, techniques, business ou temps réel.

4. Avantages de l'UML [29]:

Les objectifs de l'UML sont les suivant :

- Langage de modélisation visuel.
- Adapté à toutes les phases du développement.
- Compatible avec toutes les techniques de réalisation.
- Mécanismes d'extension et de spécialisation en vue d'étendre les concepts de base.
- Indépendant des langages de programmation.
- Base formelle pour la compréhension du langage.
- Encourage l'utilisation d'outils OO.
- Supporte les concepts de développement de haut niveau.

5. Elément de l'UML [29]:

En UML il y a 9 diagramme principaux (en réalité il y a 12), ces diagramme sont distribues en 2 grand partie, vue statique et vue dynamique :

5.1. Vue statique : Il comporte 5 diagrammes qui permet de faire la structuration et l'analyse des besoins de système à réalisé.

- Diagramme de cas d'utilisation.
- Diagramme de classes.
- Diagramme d'objets.
- Diagramme de composants.
- Diagramme de déploiement.

5.2. Vue dynamique : Il comporte 4 diagrammes qui permet de représenté les comportements de système (actions et réactions).

- Diagramme de séquence.
- Diagramme d'activités.
- Diagramme d'états de transitions.
- Diagramme de collaboration.

1) Diagrammes de classes :

Un diagramme de classes représente la structure du système sous la forme de ***classes*** et des ***associations*** entre ces classes

- **Classe :** est une description abstraite d'un ensemble d'objets ayant des propriétés, des comportements et des relations similaires. Une classe comporte :

- Attributs : un attribut de classe définit une propriété commune aux objets d'une classe, chaque objet (instance d'une classe), a sa propre identité indépendante des valeurs de ses attributs.
- Opérations : une opération définit une fonction appliquée à des objets d'une classe.
- Accessibilité des attributs et opérations : il existe trois niveaux de protection :
  a) Public (+) : accès à partir de toute entité interne ou externe à la classe.
  b) Protégé (#) : accès à partir de la classe ou des sous-classes.
  c) Privé (-) : accès à partir des opérations de la classe.

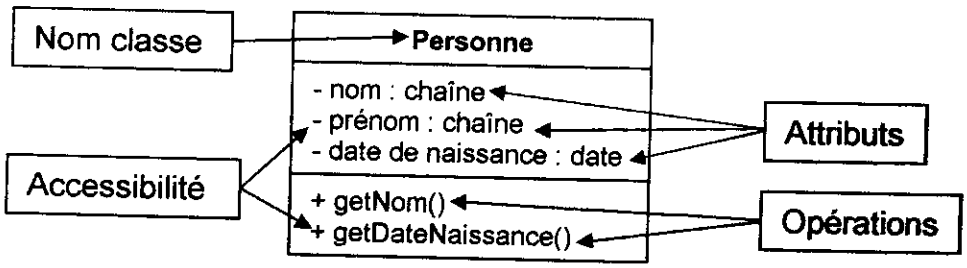


Figure B.1 : *Structure d'une classe (Personne).*

- **Association** : Une association représente une classe d'associations structurelles entre classes d'objets (2 classes ou plus), il peut être réifiée par une classe appelée classe associative ou classe association, il comporte :
  - Rôle : un rôle définit la manière dont une classe intervient dans une relation.
  - Multiplicité : la multiplicité est une information portée par le rôle, qui quantifie le nombre de fois où un objet participe à une instance de relation comme 1(un seul objet), 1..N (de 1 à N), 1..*(de 1 à plusieurs).
  - Contraint : une contrainte d'association porte sur une relation ou sur un groupe de relations pour indiqué une relation d'ordre (notée {contrainte}).

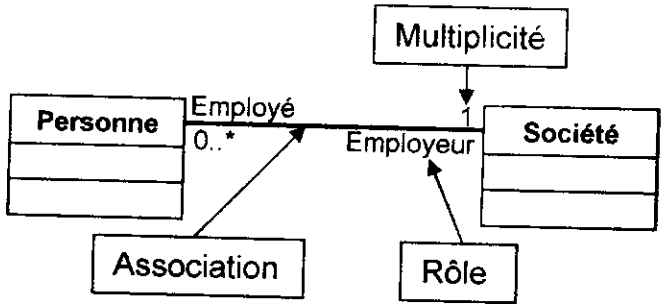


Figure B.2 : *Association.*

- Restriction : dite aussi (qualification en UML) consiste à sélectionner un sous-ensemble d'objets parmi l'ensemble des objets qui participent à une association.

- Agrégation : une agrégation est une association *non symétrique* l'une des extrémités joue un rôle prédominant par rapport à l'autre, il se justifie dans les cas suivants :
  a) Une classe B « fait partie » intégrante d'une classe A.
  b) Les valeurs d'attributs de la classe B se propagent dans les valeurs d'attributs de la classe A.
  c) Une action sur la classa A implique une action sur la classe B.
  d) Les objets de la classe B sont subordonnés aux objets de la classe A.
- Composition : la composition est une forme particulière d'agrégation, le composant est « physiquement » contenu dans l'agrégat.
- Héritage : une classe hérite d'une autre classe ses attributs et ses opérations.

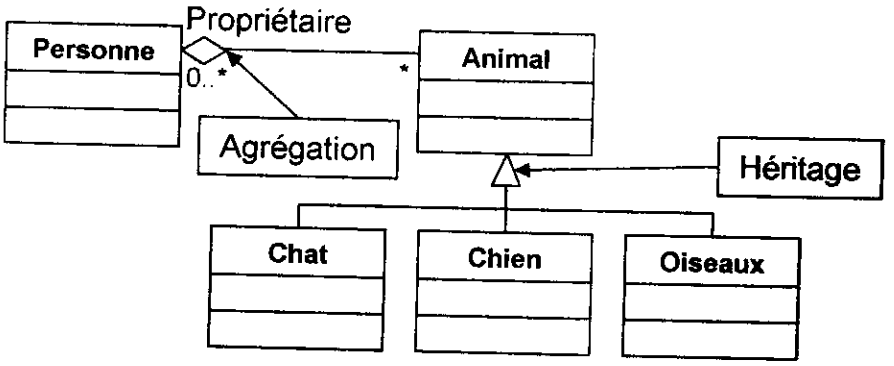


***Figure B.3** : Diagramme de classe.*

2) Diagramme de cas d'utilisation :

Est un moyens pour représenter les besoins en UML, il permet de décrit les fonctions du système selon le point de vue de ses futurs utilisateurs, il comporte :

- **Cas d'utilisation :** un cas d'utilisation correspond à une manière spécifique d'utiliser le système, c'est la représentation d'une fonctionnalité, déclenchée en réponse à une stimulation du système. Il peut être :
  - Une suite d'interactions entre un acteur et le système.
  - Il correspond à une fonction visible par l'utilisateur.
  - Il permet d'atteindre un objectif aux yeux de l'utilisateur.
- **Acteur :** Est un entité externe qui peut agit sur le système et prend des décisions contrairement à un élément logiciel. Il peut être :
  - Soit un utilisateur.
  - Soit un autre système.

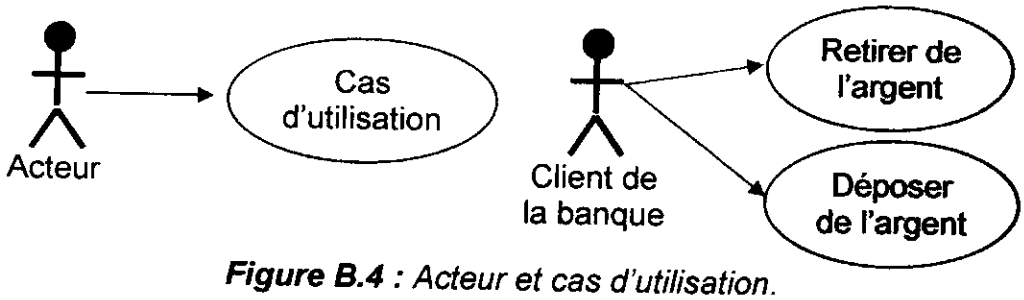


Figure B.4 : *Acteur et cas d'utilisation.*

- **Relation** : Est un relation d'utilisation entre les cas d'utilisation, il y a :
  - Relation uses : est une association entre un cas d'utilisation client et un cas d'utilisation fournisseur d'un comportement.
  - Relation extends : est une relation entre un cas d'utilisation qui étend un autre cas d'utilisation.

Figure B.5 : *Diagramme de cas d'utilisation.*

3) Diagramme de séquence :

Il décrit les échanges particuliers entre un ou plusieurs acteurs et le système, il permet de montré les informations échangés entre des instances d'acteurs ou d'objet du système sous forme des flots ordonné d'événements. Un scénario est un chemin particulier d'exécution de diagramme de séquence, il peut être :

- Scénario optimal : décrit l'interaction la plus fréquente.
- Scénarios dérivés : décrit certaines alternatives importantes non décrites dans le scénario optimal

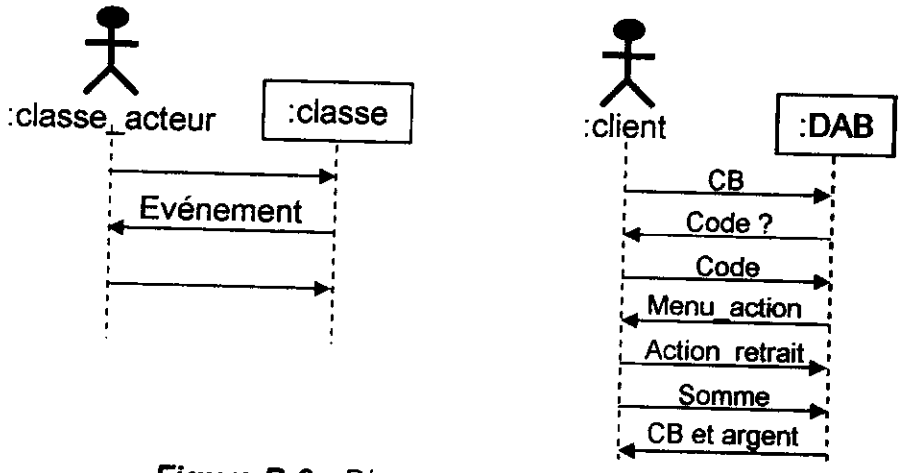


Figure B.6 : *Diagramme de séquence.*

Il existe deux types de messages :

- Synchrone : l'émetteur est bloqué jusqu'au traitement effectif du message.

- Asynchrone : l'émetteur n'est pas bloqué, il peut poursuivre son exécution.

4) Diagramme de collaboration :

Est un extension des diagrammes d'objets, il décrit le comportement collectif d'un ensemble d'objets en vue de réaliser une opération en décrivant leurs interactions modélisées par des envois (éventuellement numérotés) de messages.

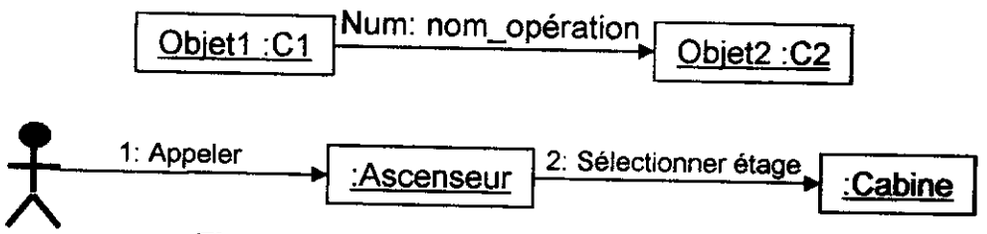


Figure B.7 : *Diagramme de collaboration.*

5) Diagramme d'activité :

Ce diagramme met l'accent sur les activités, leurs relations et leurs impacts sur les objets, il comporte :

- Activité : est un opération particulier déclanché par un objet donné.
- Transitions : ils sont les liens de séquence entre les activités, ils peuvent être gardées par des conditions.
- Gardes : sont les labels des transitions dont elles valident le déclenchement.
- Condition : une condition peut être matérialisée par un losange dont sortent plusieurs transitions.
- Synchronisation : les diagrammes d'activités représentent les synchronisations d'activités au moyen de barres de synchronisation.

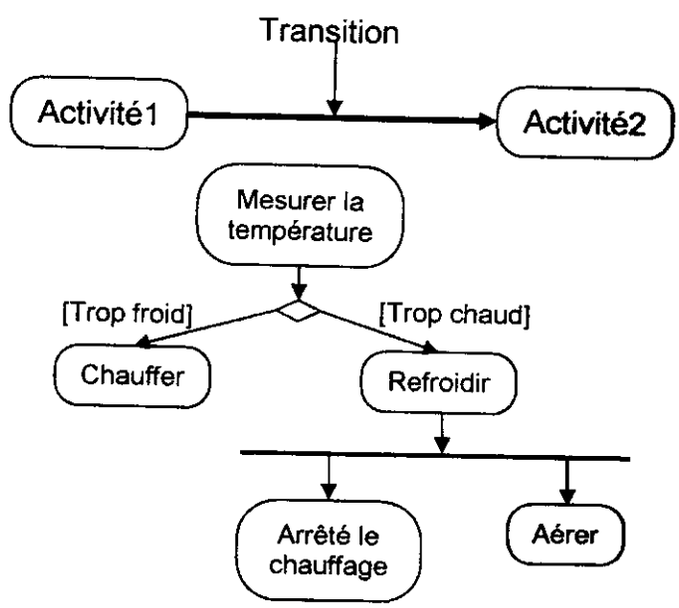
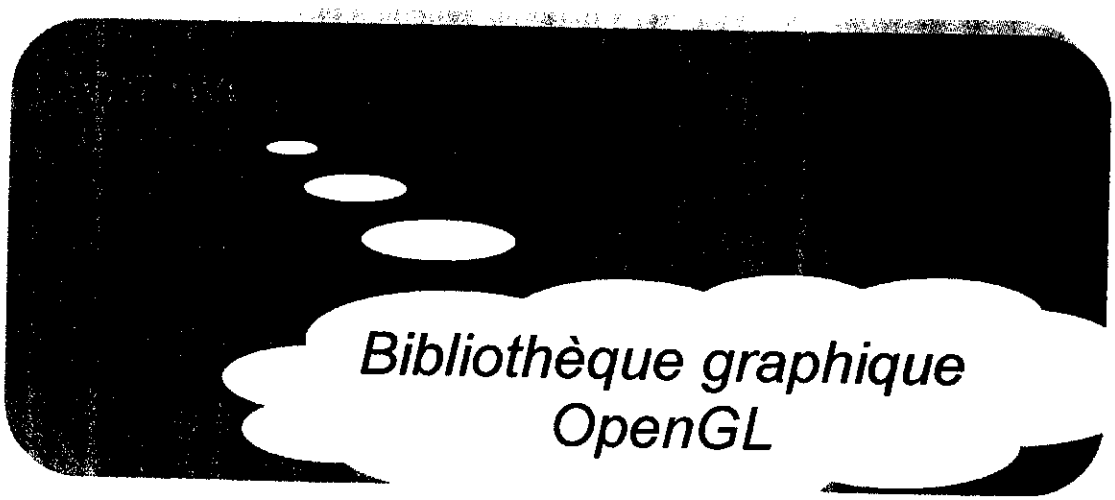


Figure B.8 : *Diagramme d'activité.*


Bibliothèque graphique
OpenGL

Annexe C

Bibliothèque Graphique OpenGL [35]

1. Définition :

L'OpenGL est un système graphique qui permet de visualiser une scène 3D (et aussi 2D). Les objets de cette scène peuvent être composés de points, de lignes, de polygones, de quadriques, de NURBS. Ils possèdent des attributs graphiques : paramètres de réflexion, couleur, texture. L'éclairage de la scène est réalisé par des lumières de différents types (spot, lumière à l'infini). La bibliothèque OpenGL a été crée par Silicon Graphics et bénéficie sur des machines à accélération matérielle.

2. Eléments de l'OpenGL :

2.1. Bibliothèques co-existant avec OpenGL :

GLU : (OpenGL Utility Library) contient les routines de bas niveau pour gérer les matrices de transformation et de projection, la facettisation des polygones et le rendu de surface. Les bibliothèques d'extension du système de fenêtres permettent l'utilisation du rendu OpenGL. Il s'agit de GLX pour X Windows (fonctions ayant pour préfixe **glX**) et de WGL pour Microsoft Windows (fonctions ayant pour préfixe **wgl**).

GLUT : OpenGL Utility Toolkit est une boite à outils indépendante du système de fenêtrage, écrite par Mark Kilgard pour simplifier la tâche d'utilision des systèmes différents (fonctions ayant pour préfixe **glut**).

GLUT possède des routines pour afficher les objets suivants : cone, cube, tétraèdre, octaèdre, icosaèdre, dodécaèdre, sphère, tore, théière.

2.2. Etats :

C'est le principe général d'OpenGL : On positionne un état interne, et sa valeur est ensuite utilisée comme valeur courante : les ordres de dessin suivants utiliseront cette valeur-ci. Prenons un exemple : pour dessiner un sommet rouge, on positionne d'abord l'état correspondant à la couleur courante à la valeur rouge, et ensuite on demande le dessin d'un sommet. Tous les sommets qui seront ensuite dessinés seront rouges, tant que l'on n'a pas modifié la couleur courante. Et ce principe que nous avons illustré à partir de l'état "couleur" s'applique à tous les états, comme l'épaisseur des traits, la transformation courante, l'éclairage, etc.

2.3. Animation :

Une animation peut se réaliser simplement en utilisant la technique du ***double buffer*** : montrer à l'écran une image correspondant à une première zone mémoire (buffer) et dessiner les objets dans une deuxième zone mémoire qui n'est pas encore à l'écran, et qui sera affiché lorsque la scène entière y sera calculée, et à une fréquence régulière. L'échange des buffers peut se faire par la commande ***glutSwapBuffers****(void)*.

2.4. Couleurs :

Les couleurs sont définies en OpenGL de deux manières :

- Couleurs indexées : 256 couleurs sont choisies, et on se réfère au numéro de la couleur (son index). C'est un mode qui était intéressant lorsque les écrans d'ordinateurs ne savaient afficher que 256 couleurs simultanées.

- Couleurs RVBA : une couleur est définie par son intensité sur 3 composantes Rouge, Vert, Bleu. La quatrième composante est appelée *canal Alpha*. La couleur d'un objet est spécifiée par l'appel à ***glColor(...)***.

2.5. Primitives géométriques :

La déclaration d'une primitive géométrique se fait par les deux commandes suivantes :

- **glBegin**(GLenum mode) ouvre la déclaration des sommets de la primitive.
- **glEnd**(void) termine cette déclaration.

Il y a dix types de primitives différents, qui correspondent à des points, des lignes, des triangles, des quadrilatères, et des polygones convexes.

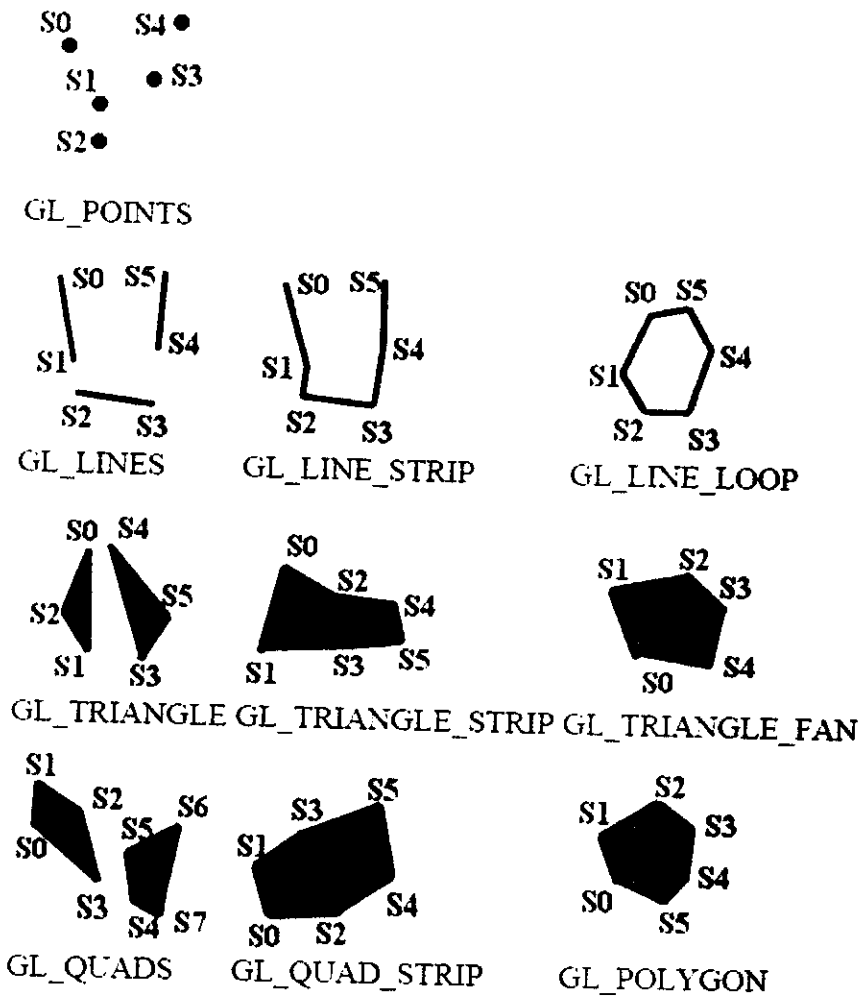


Figure C.1 : *Primitive géométrique.*

2.6. Vecteur normal :

Un vecteur normal (aussi appelé normale) à une surface en un point de cette surface est un vecteur dont la direction est perpendiculaire à la surface.

Pour une surface plane, les vecteurs normaux en tous points de la surface ont la même direction. Ce n'est pas le cas pour une surface quelconque.

C'est grâce au vecteur normal que l'on peut spécifier l'orientation de la surface dans l'espace, et en particulier l'orientation par rapport aux sources de lumière. L'appel à ***glNormal*()*** donne une valeur à la normale courante. Elle sera associée aux points spécifiés par les appels suivants à ***glVertex*()***.

2.7. Vision :

Le processus de transformation qui produit une image à partir d'un modèle de scène 3D est analogue à celui qui permet d'obtenir une photographie d'une scène réelle à l'aide d'un appareil photo. Il comprend quatre étapes (voir figure C.2).

	Avec un appareil photo	Avec un ordinateur	type de transformation
1	Positionner l'appareil photo	Placer la caméra virtuelle	transformation de vision
2	Arranger les éléments d'une scène à photographier	Composer une scène virtuelle à représenter	transformation de modélisation
3	Choisir la focale de l'appareil photo	choisir une projection	transformation de projection
4	Choisir la taille de la photographie au développement	choisir les caractéristiques de l'image	transformation de cadrage

Figure C.2 : *Processus de transformation.*

La transformation de vision se fait en utilisant les procédures OpenGL de translation et de rotations appliquées aux objets, ces procédures sont :

- **glTranslate***(TYPE x, TYPE y, TYPE z) translate le repère local de l'objet du vecteur (x, y, z).
- **glRotate***(TYPE angle, TYPE x, TYPE y, TYPE z) opère une rotation de l'objet autour du vecteur (x, y, z).
- **glScale***(TYPE a, TYPE b, TYPE c) opère un changement d'échelle sur 3 axes. Les coordonnées en x sont multipliées par a, en y par b et en z par c.

2.8. Pile de transformation :

Un modèle hiérarchique permet de décrire facilement des objets complexes composés d'objets simples. La scène est organisée dans un arbre tel que les objets ne sont plus définis par leur transformation absolue par rapport au repère de toute la scène, mais par leur transformation relative dans cet arbre.

- A chaque noeud est associé un repère. Le repère associé à la racine est le repère de la scène.
- A chaque arc est associé une transformation géométrique qui positionne l'objet fils dans le repère de son père.

L'OpenGL utilise les *coordonnées homogènes* pour manipuler ses objets. Il maintient trois matrices 4x4 distinctes pour contenir les différentes transformations.

Pour coder l'arbre de description de la scène, il faut utiliser la pile de transformation, en empilant la matrice de transformation courante (sauvegarde des caractéristiques du repère local associé) avant de descendre dans chaque noeud de l'arbre, et en dépilant la matrice en remontant (récupération du repère local associé).

Les primitives de manipulation des matrices de transformation sont :

- **glPushMatrix**() Empile la matrice courante pour sauvegarder la transformation courante.
- **glPopMatrix**() Dépile la matrice courante (La matrice du haut de la pile est supprimée de la pile, et elle devient la matrice courante).
- **glMatrixMode**(GLenum mode) spécifie quelle matrice sera affectée par les commandes suivantes de manipulation de transformations : la matrice de modélisation vision, de projection ou de texture.
- **glLoadIdentity**() donne à la *Matrice courante* la valeur identité.
- **glLoadMatrix***(const TYPE * m) donne à la *Matrice courante* la valeur de la matrice m.
- **glMultMatrix***(const TYPE * m) multiplie la *Matrice courante* par la matrice m.

2.9. Eclairage :

La perception de la couleur de la surface d'un objet du monde réel dépend de la distribution de l'énergie des photons qui partent de cette surface et qui arrivent aux cellules de la rétine de l'oeil. Chaque objet réagit à la lumière en fonction des propriétés matérielles de sa surface.

Le modèle d'éclairage d'OpenGL considère qu'un objet peut émettre une lumière propre, renvoyer dans toutes les directions la lumière qu'il reçoit, ou réfléchir une partie de la lumière dans une direction particulière, comme un miroir ou une surface brillante.

Les lampes, elles, vont envoyer une lumière dont les caractéristiques seront décrites par leurs trois composantes : ambiante, diffuse ou spéculaire.

OpenGL distingue quatre types de lumières :

- **Lumière émise :** Ne concerne que les objets. Les objets peuvent émettre une lumière propre, qui augmentera leur intensité, mais n'affectera pas les autres objets de la scène.
- **Lumière ambiante :** Concerne les objets et les lampes. C'est la lumière qui a tellement été dispersée et renvoyée par l'environnement qu'il est impossible de déterminer la direction d'où elle émane. Elle semble venir de toutes les directions. Quand une lumière ambiante rencontre une surface, elle est renvoyée dans toutes les directions.
- **Lumière diffuse :** Concerne les objets et les lampes. C'est la lumière qui vient d'une direction particulière, et qui va être plus brillante si elle arrive perpendiculairement à la surface que si elle est rasante. Par contre, après avoir rencontré la surface, elle est renvoyée uniformément dans toutes les directions.

- **Lumière spéculaire :** Concerne les objets et les lampes. La lumière spéculaire vient d'une direction particulière et est renvoyée par la surface dans une direction particulière. Par exemple un rayon laser réfléchi par un miroir.
- **Brillance :** Ne concerne que les objets. Cette valeur entre 0.0 et 128.0 détermine la taille et l'intensité de la tâche de réflexion spéculaire. Plus la valeur est grande, et plus la taille est petite et l'intensité importante.

2.10. Lampes :

OpenGL offre d'une part une lampe qui génère uniquement une lumière ambiante (lampe d'ambiance), et d'autre part au moins 8 lampes (GL_LIGHT0, ..., GL_LIGHT7) que l'on peut placer dans la scène et dont on peut spécifier toutes les composantes.

La lampe GL_LIGHT0 a par défaut une couleur blanche, les autres sont noires par défaut.

La lampe GL_LIGHT*i* est allumée par un ***glEnable****(GL_LIGHTi)*.

Il faut également placer l'instruction ***glEnable****(GL_LIGHTING)* pour indiquer à OpenGL qu'il devra prendre en compte l'éclairage.

2.11. Matérielles :

Les propriétés matérielles d'un objet sont celles qui ont été évoquées dans la partie *éclairage* : la lumière émise, la réflexion ambiante, diffuse et spéculaire du matériau dont est fait l'objet. Elles sont déterminées par l'instruction suivante :

- **glMaterial***(GLenum face, GLenum pname, TYPE param) Où pname vaut GL_AMBIENT, GL_DIFFUSE, GL_AMBIENT_AND_DIFFUSE, GL_SPECULAR, GL_SHININESS, ou GL_EMISSION.

2.12. Liste d'affichage :

Il s'agit d'un mécanisme pour stocker des commandes OpenGL pour une exécution ultérieure, qui est utile pour dessiner rapidement un même objet à différents endroits.
Les instructions pour stocker les éléments d'une liste d'affichage sont regroupées entre une instruction ***glNewList****(GLuint numList, GLenum mode)* et une instruction ***glEndList()***. Le paramètre *numList* est un numéro unique (*index*) qui est généré par ***glGenLists****(GLsizei range)* et qui identifie la liste.
Le paramètre *mode* vaut GL_COMPILE ou GL_COMPILE_AND_EXECUTE. Dans le deuxième cas, les commandes sont non seulement compilées dans la liste d'affichage, mais aussi exécutées immédiatement pour obtenir un affichage.

Les éléments stockés dans la liste d'affichage sont dessinés par l'instruction ***glCallList****(GLuint numList)*.

Une fois qu'une liste a été définie, il est impossible de la modifier à part en la détruisant et en la redéfinissant.

2.13. Quadriques :

La bibliothèque GLU permet de créer certains objets définis par une équation quadratique : des sphères, des cylindres, des disques et des disques partiels. Pour cela il y a cinq étapes :

1) Utiliser **gluNewQuadric**() pour créer une quadrique.

2) Spécifier les attributs de rendu :
   - **gluQuadricOrientation**() indique la direction des normales vers l'extérieur ou l'intérieur.
   - **gluQuadricDrawStyle**() indique le style de rendu : avec des points, des lignes ou des polygones pleins.
   - **gluQuadricNormals**() permet de générer des normales pour chaque face ou pour chaque sommet.
   - **gluQuadricTexture**() permet de générer les coordonnées de texture.
3) Indiquer la fonction qui traite les erreurs dans l'affichage de la quadrique par **gluQuadricCallback**()
4) Afficher la quadrique désirée avec **gluSphere**(), **gluCylinder**(), **gluDisk**(), ou **gluPartialDisk**(). Pour une meilleure efficacité, il est recommandé d'utiliser des listes d'affichage.
5) Enfin, pour détruire l'objet, appeler **gluDeleteQuadric**().

2.14. Textures :

Le placage de texture consiste à placer une image (la texture) sur un objet. Cette opération se comprend aisément lorsqu'il s'agit de plaquer une image rectangulaire sur une face rectangulaire.

De manière générale on procédant de la manière suivante :

1) Il s'agit d'abord de créer un Objet Texture et de spécifier la texture que l'on va utiliser.
2) Ensuite choisir le mode de placage de la texture avec **glTexEnv[f,i,fv,iv]()**
   - Le mode *decal* décalque la texture sur l'objet : la couleur de l'objet est remplacée par celle de la texture.
   - Le mode *modulate* utilise la valeur de la texture en modulant la couleur précédente de l'objet.
   - Le mode *blend* mélange la couleur de la texture et la couleur précédente de l'objet.
3) Puis autoriser le placage de textures avec **glEnable**(GL_TEXTURE_2D) si la texture a deux dimensions. (Ou GL_TEXTURE_1D si la texture est en 1D).
4) Il est nécessaire de spécifier *Les coordonnées de texture* en plus des coordonnées géométriques pour les objets à texturer.

Les coordonnées de texture vont de 0.0 à 1.0 pour chaque dimension (voir figure C.3).

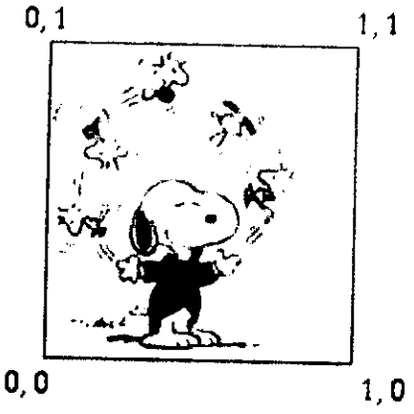


Figure C.3 : *Coordonnées de texture.*

Les principes primitifs de manipulation des *Objets Texture* sont :

- **glTexCoord*()** : Pour assigné à chaque sommet des objets à texturer un couple de valeurs qui indique les coordonnées de texture de ce sommet.
- **glGenTextures**(GLsizei *n*, GLuint **textureNames*) : Renvoie *n* noms (des numéros, en fait) de textures dans le tableau *textureNames[]*.
- **glbindTexture**(GL_TEXTURE_2D, GLuint *textureNames*) : à trois effets différents :
  - la première fois qu'il est appelé avec une valeur *textureNames* non nulle, un nouvel *Objet Texture* est crée, et le nom *textureNames* lui est attribué.
  - avec une valeur *textureNames* déjà liée à un *objet Texture*, cet objet devient actif.
  - avec la valeur zéro, OpenGL arrête d'utiliser des *objets textures* et retourne à la texture par défaut, qui elle n'a pas de nom.
- **glDeleteTextures**(GLsizei *n*,const GLuint **textureNames*) : Efface *n objets textures* nommés par les éléments du tableau *textureNames*.

2.15. Tableaux de Sommets :

La motivation pour l'utilisation de tableaux de sommets est double : il s'agit d'une part de réduire le nombre d'appels de fonctions, et d'autre part d'éviter la description redondante de sommets partagés par des polygones adjacents.

En général, on procédant de la manière suivant :

1) Il s'agit d'abord **d'activer** jusqu'à six tableaux, stockant chacun un des six types de données suivantes :
   - Coordonnées des sommets.
   - Couleurs RVBA.
   - Index de couleurs.
   - Normales de surfaces.
   - Coordonnées de texture.
   - Indicateurs de contour des polygones.

2) Ensuite spécifier les données du ou des tableaux.
3) Puis déréférencer le contenu des tableaux (accéder aux éléments) pour tracer une forme géométrique avec les données. Il y a trois méthodes différentes pour le faire :
   - Accéder aux éléments du tableau un par un : accès aléatoire.
   - Créer une liste d'éléments du tableau : accès méthodique.
   - Traiter les éléments du tableau de manière séquentielle : accès séquentiel ou systématique.

Six routines permettent de spécifier des tableaux en fonction de leur type :

- void **glVertexPointer**(GLint *taille*, GLenum *type*, GLsizei *stride*, const GLvoid **pointeur*).
- **void glColorPointer**(GLint *taille*, GLenum *type*, GLsizei *stride*, const GLvoid **pointeur*).
- void **glIndexPointer**(GLenum *type*, GLsizei *stride*, const GLvoid **pointeur*).
- **void glNormalPointer**(GLenum *type*, GLsizei *stride*, const GLvoid **pointeur*).
- void **glTexCoordPointer**(GLint *taille*, GLenum *type*, GLsizei *stride*, const GLvoid **pointeur*).
- void **glEdgeFlagPointer**(GLsizei *stride*, const GLvoid **pointeur*).

Le paramètre *pointeur* indique l'adresse mémoire de la première valeur pour le premier sommet du tableau.

Le paramètre *type* indique le type de données.

Le paramètre *taille* indique le nombre de valeurs par sommets, et peut prendre suivant les fonctions les valeurs 1, 2, 3 ou 4.

Le paramètre *stride* indique le décalage en octets entre deux sommets successifs.

2.16. Lissage :

Les lignes qui ne sont ni horizontales ni verticales sont crénelées. Ce phénomène est appelé *aliasing*.

La fonction **glHint**(GLenum *cible*, GLenum *hint*) permet d'indiquer les préférences que l'on a entre qualité de l'image et rapidité de l'affichage.

Le paramètre *cible* peut prendre les valeurs suivantes : GL_POINT_SMOOTH_HINT,GL_LINE_SMOOTH_HINT,GL_POLYGON_SMOOTH_HINT : pour une qualité d'échantillonnage souhaitable, GL_FOG_HINT : pour la qualité ou la rapidité, GL_PERSPECTIVE_CORRECTION_HINT : pour prend en compte la perspective ou non.

Le paramètre *hint* peut prendre la valeur GL_FASTEST, pour privilégier la vitesse, ou GL_NICEST pour privilégier la qualité, ou GL_DONT_CARE pour indiquer l'absence de préférence.

Liste des figures

CHAPITRE 1 : METHODES DE CONCEPTION DES SURFACES

CHAPITRE 2 : MACHINES OUTIL A COMMANDE NUMERIQUE (MOCN)

CHAPITRE 3 : USINAGE DES SURFACES GAUCHES

CHAPITRE 4 : METHODE DE SIMULATION DE L'USINAGE

CHAPITRE 5 : CONCEPTION ET IMPLEMENTATION

CHAPITRE 6 : PRESENTATION DE L'APPLICATION

CHAPITRE 7 : TEST ET VALIDATION

Annexe A :

Annexe B :

Annexe C :

REFERENCES BIBLIOGRAPHIQUES

[1]: A. Doneddu. « *Géométrie différentielle, intégrales multiples* ». Tome 6. Editions Vuibert. 1981.

[2]: Y. Zhou. « *Surface Interpolation and Approximation* ». Rapport de projet de Master. Université de Michigan, Février 1999.

[3]:Y. Zhao. « *Problems in Surface Intersection Representation and Construction* ». Thèse de Master. Université de Michigan, Août 1998.

[4]: W. Ma and P. He. « *B-spline surface local updating with unorganized points* ». Computer Aided Design, 1998, Vol 30, No. 11, pp. 853–862.

[5]: Y. Zhao, Y. Zhou, J.L. Lowther and C.K. Shene. « *Cross-Sectional Design: A Tool for Computer Graphics and Computer-Aided Design Courses* ». 29th ASEE/IEEE Frontiers in Education, 10-13 Novembre, San Juan, Puerto Rico, Vol. II (1999), pp. (12b3-1)-(12b3-6).

[6]: L.A Piegl and W. Tiller. « *Geometry-based triangulation of trimmed NURBS surfaces* ». Computer Aided Design, 1998, Vol 30, No. 1, pp. 11–18.

[7]: M. Peternell, H. Pottmann and B. Ravani. « *On the computational geometry of ruled surfaces* ». Computer-Aided Design, 1999, Vol 31, pp. 17–32.

[8]: H.S Heo, M.S. Kim and G. Elber. « *The intersection of two ruled surfaces* ». Computer-Aided Design, 1999, Vol 31, pp. 33–50.

[9]: Djamel Chabane et Lyes Mouterfi « Automatisation de l'opération d'ébauchage des surfaces sur des fraiseuses à commande numérique à 3 axes » PFE 2005 Université de Blida.

[10]: E.Duc E. Lefur « *Machine outils à commande numérique structure modélisation et réglage* » préparation à l'agrégation de génie mécanique 16 Septembre 1997.

[11]: « *Manuel de programmation volume 1* », 1020/1040/1060.

[12]: Bernard Méry. « *Machines à commande numérique* ».

[13]: STI. GMA. « *Programmation CN* ».

[14]: Roland Maranzana. « *Fabrication Assistée Par Ordinateur*». École de technologie supérieure . Génie de la production automatisée.

[15]: Cours de fabrication. « *Usinage par enlèvement de copeaux* ».

[16]: « *Commande numérique par calculateur* », 1er année G.M.P 2003/2004.

[17]: « *Introduction au fraisage* », rapport de CDTA.

[18]: « The right choice of highly productive cutting tools for roughing to finishing ».

[19]: Quinsta Yann, Laurent Sabourin et Grigore Gogu. « *Aide au choix des stratégies d'usinage, Etude de l'état de surface, formes complexes* ».

[20]: Ro. H. Kim, Royong K Choi, « *Machining efficiency comparison direction parallel tool path with contour parallel tool path* ».

[21]: Emmanuel DUC, « *Usinage de formes gauches contribution à l'amélioration de qualité de trajectoire d'usinage* », Thèse DOCTORAT de l'école normale supérieure de Cachan, 1999.

[22]: Robert B. Drysdale, Kenneth Hauck University of New Hampshire « *Methods for Detecting Errors in Numerically Controlled Machining of Sculptured Surfaces* ».

[23]: R.B. Jerard1, S.Z. Hussaini1, R.L. Drysdale, and B.Schaudt. « *Approximate methods for simulation and verification of numerically controlled machining programs* ».

[24]: Robert B. Jerard, Jennifer M. Angleton,Department of Mechanical Engineering University of New Hampshire. « *The Use of Surface Points Sets for Generation, Simulation, Verification and Automatic Correction of NC Machining Programs* ».

[25]: Emmanuel DUC, « *RÉALISATION DES FORMES GAUCHES* ». Thèse DOCTORAT de l'école normale supérieure de Cachan, 1999.

[26]: Andreas Holger Konig and Eduard Groller Institute of Computer Graphics, Vienna University of Technology. « *Real Time Simulation and Visualization of NC Milling Processes for Inhomogeneous Materials on Low-End Graphics Hardware* ».

[27]: Alain Le Guennec, IRISA / Université de Rennes 1 « *Méthodes formelles avec UML, Modélisation, validation et génération de tests1* ».

[28]: ORSYS, Stage pratique, La Grande Arche, Paroi Nord, Paris « *UML, analyse et conception* ».

[29]: Frédéric Julliard, Université de Bretagne Sud UFR SSI - IUP Vannes, « UML Unified Method Language -Langage unifié pour la modélisation objet- ».

[30]: Un site d'Etienne Mauvais, « L'intégral des maths ».

[31]: Elkeran, A. ET El-Baz, M.A, « *NURBS feedarate adaptation for 3 axes cnc machining* ».

[32]: Dongmok Sheen1, Chang Ho Lee2, Sang Do Noh2, and Kiwoo Lee2, School of transportation systems engineering, University of Ulsan, Ulsan, Kore « *A Process Planning System for Machining of Dies for Auto- Body Production - Operation Planning and NC Code Post-Processing* ».

[33]: S. K. Ong, L. Jiang and A. Y. C. Nee, Mechanical Engineering Department, National University of Singapore, 10 Kent Ridge Crescent, Singapore, *«An Internet-Based Virtual CNC Milling System »*.

[34]: Robert B. Jerard Barry K. Fussell Mustafa T. Ercan **Jeffrey G. Hemmett,** Mechanical Engineering Department University of New Hampshire **Durham, NH** 03824, U.S.A.*«Integration of geometric and mechanistic models of nc maching into an open architecture machine tool controller »*.

[35]: Eric BITTAR, Université de Reims *« Cours OpenGL »*.