Chapitre 5

Dégénérescences et imprécisions des calculs numériques

Dans ce chapitre nous montrons comment traiter les cas dégénérés dans les divers calculs de visibilité étudiés dans les chapitres précédents, grâce à l'idée de *simulation de simplicité guidée par la visibilité*. Nous étudions ensuite les problèmes posés par les imprécisions de calculs numériques et proposons plusieurs solutions pour y remédier.

— C'est une machine qui joue au bilboquet sans jamais rater son coup. [...] La force et l'ampleur de son mouvement sont calculées en fonction du poids exact de la boule, dont la trajectoire est instantanément décidée avec une précision...Hou lââ ! Au micropoil ! [...]

— Fou ! Il est fou !

— [...]

— Clic ! Djiiiiiii ! **KRAK !** *Tchink !*

— Lorsque la brave machine, dans un éclair de lucidité, a compris combien elle était moche et mal fichue, elle décida, courageusement, de se supprimer ...

Le gang des gaffeurs.

5.1 Traitement des dégénérescences

Nous avons supposé dans les chapitres précédents que les points de la scène (sommets des polygones ou points de vue) n'étaient jamais alignés. L'hypothèse ≪ nous supposons que les objets sont en *position générale* ≫ est couramment rencontrée dans l'étude théorique d'un problème géométrique. Cette hypothèse permet d'éliminer les cas dégénérés qui pourraient se présenter (points alignés, points cocycliques, droites concourantes ...). En effet, des arguments de perturbation montrent qu'en général la complexité théorique des structures de données et des algorithmes relatifs au problème étudié est maximale pour les *configurations générales* (c'est-à-dire les scènes où les objets sont en position générale). De plus, si la complexité théorique n'est pas modifiée par la présence de scènes dégénérées, il n'en est pas de même pour la complexité pratique : il faut alors considérer un nombre parfois important de situations particulières propres aux situations dégénérées.

Cette abondance de cas particuliers n'apporte en général rien à l'étude théorique du problème géométrique abordé et détailler tous les cas particuliers aurait plutôt tendance à rallonger et obscurcir l'exposé. Le traitement de ces cas est donc laissé de côté. Cependant les cas dégénérés sont souvent la règle en pratique, et les programmes qui ne les prennent pas en compte terminent souvent de façon brutale et prématurée leur exécution lorsqu'ils traitent des exemples concrets (figure 5.1).

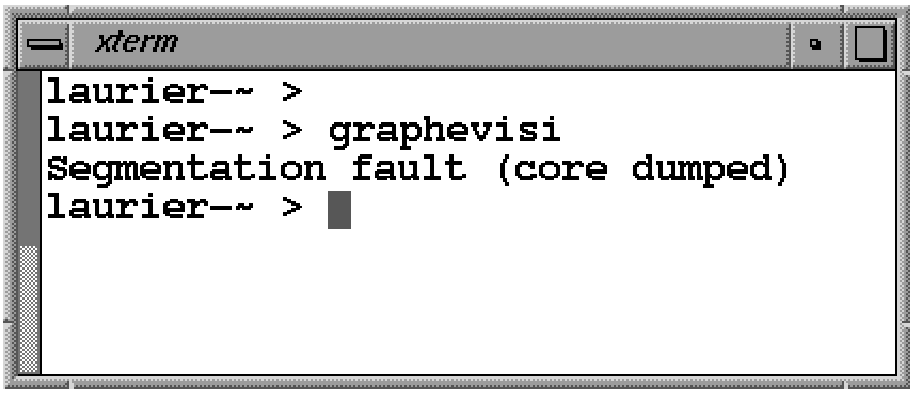


FIG. 5.1 – *Résultat d'un programme traitant une scène dégénérée.*

Dans cette section nous étudions la nature des problèmes posés par les dégénérescences. Nous étudions ensuite la possibilité d'appliquer un schéma de perturbation symbolique pour traiter ces problèmes. Nous montrons enfin comment améliorer ces schémas pour traiter les problèmes liés à la visibilité dans le plan.

5.1.1 Problèmes posés par les dégénérescences

Une configuration dégénérée est une configuration limite entre deux configurations non dégénérées. Lorsqu'un programme écrit pour des scènes générales rencontre un cas dégénéré, il l'assimile à l'une des deux configurations générales voisines. Or les traitements à appliquer aux deux configurations générales sont différents et souvent incompatibles. De plus, le choix de la configuration générale assimilée au cas dégénéré est souvent différent selon l'endroit du programme où le cas dégénéré est rencontré. Un même cas dégénéré peut donc recevoir des traitements différents et incompatibles au cours du déroulement du programme ; la cohérence est alors perdue et le programme devant faire face à des informations contradictoires s'interrompt alors brutalement.

Prenons comme exemple le balayage du graphe de visibilité d'une scène polygonale. La situation où trois points p_1, p_2 et p_3 sont alignés est une situation limite entre la configuration *complète* où les trois points sont mutuellement visibles et la configuration *simple* où les points extrêmes p_1 et p_3 sont cachés par le polygone en p_2 (figure 5.2).

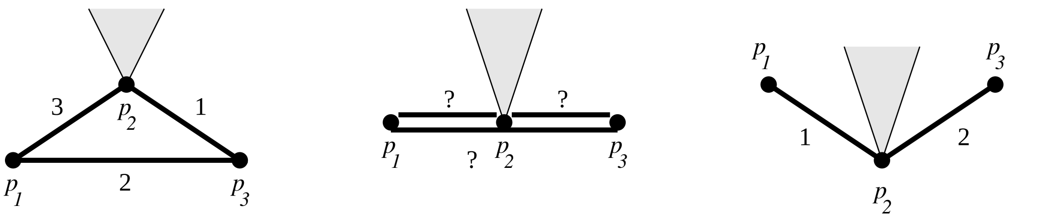


FIG. 5.2 – *Deux situations générales et leur cas dégénéré limite : ordre de passage des arêtes.*

Nous avons indiqué dans la figure 5.2 près des arêtes du graphe de visibilité l'ordre relatif dans lequel celles-ci doivent être balayées : dans la configuration complète, les arêtes doivent être passées dans l'ordre (p_2, p_3), (p_1, p_3) et (p_1, p_2) ; dans la configuration simple, elles doivent être passées dans l'ordre (p_1, p_2), (p_2, p_3). L'ordre de passage est différent dans les deux situations générales pour les arêtes (p_1, p_2) et (p_2, p_3). Si le programme assimile à un moment la situation dégénérée à la situation simple puis à un autre moment l'assimile à la situation complète, l'ordre de passage des arêtes devient incohérent Ce scénario catastrophe a de grandes chances de se produire par exemple pour un algorithme de balayage *droit* qui passe les arêtes dans l'ordre de leurs pentes : le choix d'une arête parmi plusieurs de même pente dans la queue de priorité risque de ne pas être cohérent.

La méthode la plus directe pour résoudre ces problèmes de dégénérescence est de traiter directement les cas dégénérés, c'est-à-dire de tenir compte de chaque cas dégénéré pouvant se produire. Cela suppose le recensement de toutes les configurations dégénérées, l'écriture du code spécifique pour chaque configuration et l'écriture de tests particuliers dans le programme pour repérer la présence de ces dégénérescences. Cette méthode, bien que directe, n'est pas la plus simple : il faut parfois étudier un grand nombre de configurations dégénérées. De plus, les solutions apportées sont spécifiques au problème traité et ne sont pas applicables à d'autres problèmes pour lesquels tout est à refaire.

Plusieurs méthodes générales ont été étudiées pour proposer des techniques peuvant s'appliquer à une grande variété de problèmes et dont l'emploi permet de ne pas bouleverser le code déjà écrit.

5.1.2 Schémas de perturbation symbolique

L'idéal pour le programmeur est d'avoir un moyen automatique de transformer (de façon purement syntaxique) un programme qui résout un problème pour toute entrée non dégénérée en un programme qui résout le problème pour toutes les entrées. Les *schémas de perturbation symbolique* permettent de transformer le programme en un programme qui « fonctionne » toujours, même sur des données dégénérées, mais qui cependant ne résout pas le problème initial mais un problème perturbé « proche » du problème initial. Nous rappelons ici quelques grandes lignes de la technique de perturbation (pour plus de détails, lire par exemple l'exposé de Seidel [Sei94] dont nous nous sommes inspirés pour cette présentation).

Un programme est formellement modélisé par un arbre de décision : le programme prend en entrée un vecteur de données $\mathbf{p} = (x_1, \dots, x_n) \in \mathbb{R}^n$ et, à chaque branche de son exécution, choisit la branche suivante à exécuter en fonction du signe de $f(\mathbf{p})$, où f est une fonction de décision de $\mathbb{R}^n$ dans $\mathbb{R}$ associée à la branche en cours d'exécution.

Un schéma de perturbation est défini par une fonction π, qui à tout vecteur de données $\mathbf{p}$ de $\mathbb{R}^n$ associe une fonction continue π_p de $\mathbb{R}^+$ dans $\mathbb{R}^n$ telle que $\pi_p(0) = \mathbf{p}$. Dans le programme, le calcul de $f(\mathbf{p})$ est alors remplacé par le calcul de $\overline{f}(\mathbf{p}) = \lim_{\varepsilon \to 0^+} f(\pi_p(\varepsilon))$. Si f est continue (en tout point non dégénéré), alors $\overline{f}(\mathbf{p}) = f(\mathbf{p})$ et le programme perturbé se comporte comme le programme initial pour des données non dégénérées. L'emploi de perturbations a pour but d'éviter d'exécuter les branches associées à des données dégénérées ($f(\mathbf{p}) = 0$) ; le schéma de perturbation π est alors dit valide si $\lim_{\varepsilon \to 0^+} \mathrm{sgn} f(\pi_p(\varepsilon))$ existe et est différent de 0 pour toutes les données $\mathbf{p}$ et toutes les fonctions de décision f. Grâce à l'emploi de perturbations, les cas dégénérés n'existent plus à l'intérieur du programme.

Le schéma de perturbation est symbolique, c'est-à-dire que le signe limite est en fait calculé sans calculs numériques impliquant ε (ce point est développé plus loin sur un exemple).

Enfin un schéma de perturbation doit être consistant pour être utilisable : si une fonction f n'est en fait qu'une fonction à m variables, alors l'ensemble des résultats obtenus pour toutes les permutations de taille m des coordonnées de $\mathbf{p}$ doivent être cohérents.

Plusieurs auteurs ont élaboré différents schémas particuliers de perturbation. Yap [Yap90] a conçu un schéma basé sur l'évaluation des dérivées partielles successives des fonctions de décision et a démontré un théorème de consistance. Ce schéma est cependant assez difficile à mettre en œuvre.

Seidel [Sei94] a montré qu'on pouvait toujours utiliser un schéma de perturbation linéaire, c'est-à-dire une fonction de perturbation $\pi_p(\varepsilon) = \mathbf{p} + \varepsilon\mathbf{a}$ où $\mathbf{a}$ est un vecteur de $\mathbb{R}^n$ non dégénéré ($f(\mathbf{a}) \neq 0$ pour toutes les fonctions de décision). Ce schéma est assez simple, cependant le calcul du vecteur $\mathbf{a}$ utilisé n'est pas toujours facile. De plus, un désavantage de cette méthode est que le résultat obtenu dépend du vecteur $\mathbf{a}$ choisi : le programme de calcul de graphe de visibilité calculerait, pour une même scène, différents graphes de visibilité suivant le vecteur $\mathbf{a}$ utilisé.

Edelsbrunner et Mücke [EM90] ont élaboré un schéma de perturbation appelé *simu-*

lation de la simplicité, surtout utilisé pour le calcul de déterminants. Nous le détaillons ici, car en plus d'être simple à mettre en œuvre, il nous permet d'illustrer concrètement l'utilisation d'un schéma de perturbation sur une opération – le calcul d'un déterminant – qui est une opération de base de nos algorithmes de calculs de visibilité. En effet, le signe du déterminant de trois points indique l'orientation relative des trois points, et permet par exemple de faire le test $u \prec \mathcal{E}^s(u)$ et de balayer les chaînes d'arêtes dans le calcul du graphe de visibilité (chapitre 2).

La simulation de la simplicité effectue la perturbation symbolique suivante : les coordonnées (x_i, y_i) d'un point p_i sont transformées en coordonnées perturbées $(x'_i, y'_i) = (x_i + \varepsilon^{2i}, y_i + \varepsilon^{2i+1})$, et le déterminant de trois points devient le déterminant perturbé

$$\begin{aligned}\det{}'(p_i,p_j,p_k) &= \begin{vmatrix} x'_i & x'_j & x'_k \\ y'_i & y'_j & y'_k \\ 1 & 1 & 1 \end{vmatrix} \\ &= \begin{vmatrix} x_i & x_j & x_k \\ y_i & y_j & y_k \\ 1 & 1 & 1 \end{vmatrix} + \varepsilon^{2i} \begin{vmatrix} y_j & y_k \\ 1 & 1 \end{vmatrix} - \varepsilon^{2i+1} \begin{vmatrix} x_j & x_k \\ 1 & 1 \end{vmatrix} - \varepsilon^{2j} \begin{vmatrix} y_i & y_k \\ 1 & 1 \end{vmatrix} \\ &\qquad + \varepsilon^{2j+1} \begin{vmatrix} x_i & x_k \\ 1 & 1 \end{vmatrix} + \varepsilon^{2k} \begin{vmatrix} y_i & y_j \\ 1 & 1 \end{vmatrix} - \varepsilon^{2k+1} \begin{vmatrix} x_i & x_j \\ 1 & 1 \end{vmatrix}.\end{aligned}$$

Le déterminant perturbé est un polynôme en ε, dont le signe est celui du premier coefficient non nul (en supposant ici $i < j < k$). Si les trois points sont différents alors, même si le déterminant initial est nul, au moins un des sous-déterminants extraits est non nul et le déterminant perturbé est toujours non nul. La perturbation reste symbolique : la procédure qui calcule le signe du déterminant perturbé n'utilise pas ε, mais elle calcule le déterminant et les sous-déterminants non perturbés jusqu'à trouver le premier non nul et renvoie alors son signe.

L'inconvénient de cette méthode est qu'elle augmente parfois inutilement la complexité des situations dégénérées. Ainsi, dans notre calcul de graphe de visibilité, si quatre points sont alignés toutes les arêtes entre paires de points risquent d'être calculées (figure 5.3a) alors que nous voulons seulement considérer les arêtes entre points successifs (figure 5.3b).

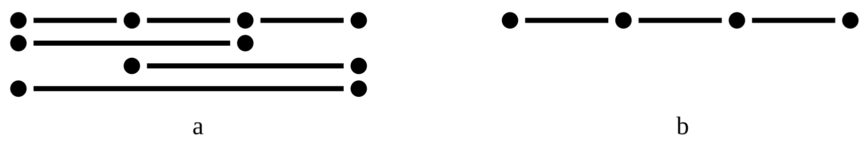


FIG. 5.3 – *Situation dégénérée.* **a.** *Situation complète.* **b.** *Situation simple.*

De plus, quand le déterminant initial est nul, le signe du déterminant perturbé dépend fortement de l'ordre relatif des indices i, j et k de numérotation des trois points considérés. L'existence et l'ordre de passage des arêtes entre points alignés sont alors dictés par la numérotation de ces points, et nous avons une situation surprenante : le programme de calcul de graphe de visibilité utilisant ce déterminant perturbé calcule

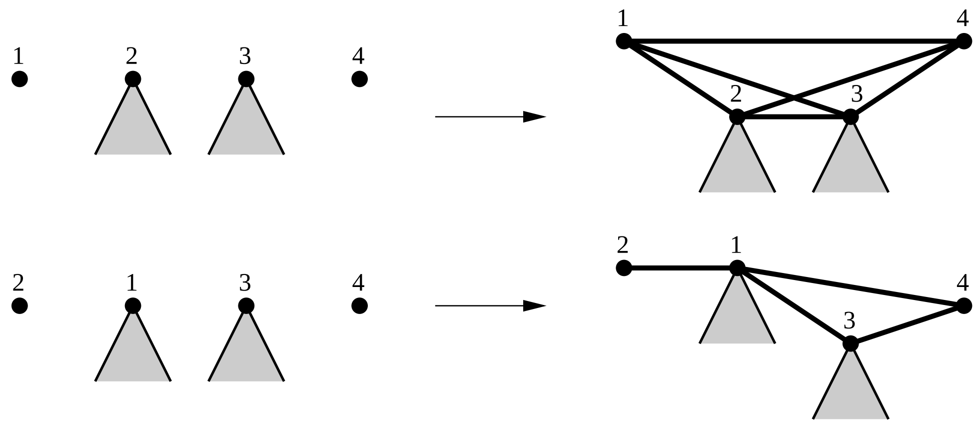


FIG. 5.4 – *Graphes de visibilité d'une même scène dégénérée calculés avec des déterminants perturbés : différences dues à la numérotation des points.*

pour une même scène des graphes différents suivant l'ordre dans lequel les points qui lui sont donnés sont numérotés (figure 5.4) !

En fait, le problème de cette méthode de perturbation et des autres méthodes vues précédemment est qu'elles utilisent des données extérieures au problème pour enlever les dégénérescences (le point **a** pour les perturbation linéaires, l'ordre de numérotation des points pour la simulation de simplicité). Pour éviter cette dépendance artificielle, nous nous servons des relations de visibilité pour guider la simulation de la simplicité.

5.1.3 Simulation de simplicité guidée par la visibilité

Nous retenons l'idée de faire comme si les cas dégénérés n'existaient pas, mais nous voulons en plus garder la « simplicité » géométrique de la situation. Considérons les situations dégénérées où trois points p_1, p_2 et p_3 sont alignés. Nous assimilons alors la configuration dégénérée à la configuration générale où les deux points extrêmes p_1 et p_3 sont cachés l'un de l'autre par le polygone en p_2 (figure 5.5).

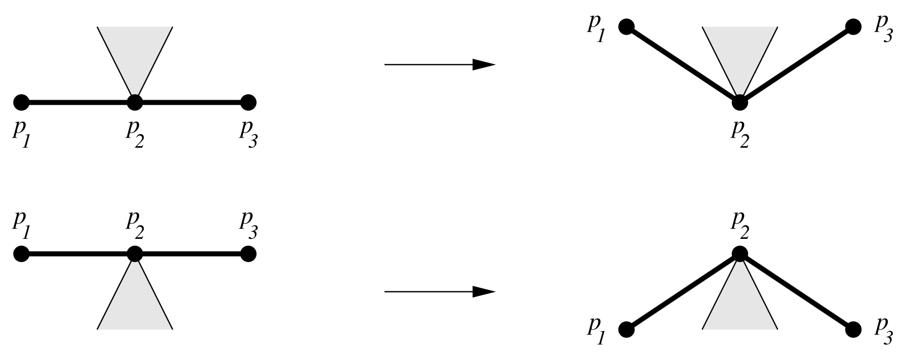


FIG. 5.5 – *Configurations dégénérées et configurations générales simulées.*

Cette méthode s'applique directement dans les tests du programme où la géométrie locale autour des trois points testés est connue. Le test

si $\det(p_1, p_2, p_3) > 0$ **alors** `traitement1`
sinon `traitement2`

est laissé tel quel si le cas dégénéré ($\det(p_1, p_2, p_3) = 0$) doit être assimilé à la situation où p_3 est au-dessus de (p_1, p_2). Le test doit être modifié en

si $\det(p_1, p_2, p_3) < 0$ **alors** `traitement2`
sinon `traitement1`

si le cas dégénéré doit être assimilé à la situation où p_3 est en dessous de (p_1, p_2).

Nous montrons maintenant comment appliquer cette règle simple aux calculs de visibilité étudiés dans les chapitres précédents.

5.1.4 Application aux calculs de visibilité

Reprenons tout d'abord le calcul de vue exposé au chapitre 3, et plus particulièrement le test d'une arête de la chaîne supérieure pour trouver l'arête de sortie de la face. La figure 5.6 montre les situations dégénérées – où le point de vue est aligné avec deux sommets de la scène – et les situations générales simulées correspondantes.

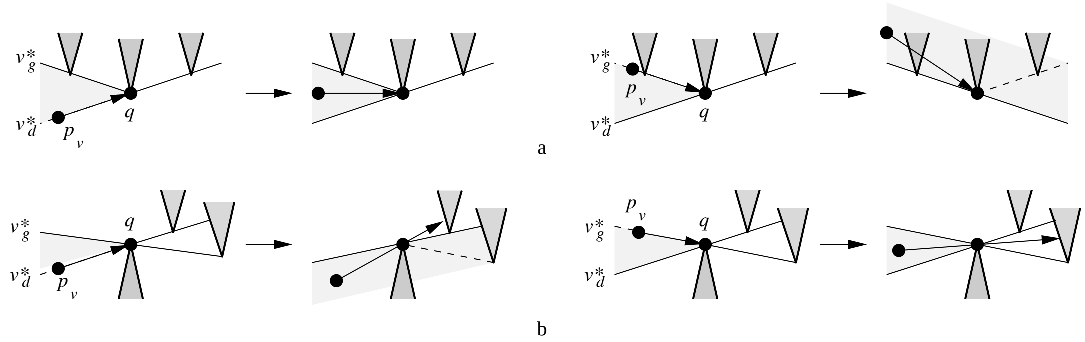


FIG. 5.6 – *Configurations dégénérées pour le test de sortie d'une face dans le calcul de vue et configurations générales associées.* **a.** *arête normale.* **b.** *arête grasse.*

Les tests de comparaison entre le point de vue p_v et l'arête $e = (v_g, v_r)$ qui s'écrivaient

sortie après l'arête $e \iff p_v$ est en dessous de v_d^*
sortie avant l'arête $e \iff p_v$ est au-dessus de v_g^*

deviennent maintenant pour une arête normale

sortie après l'arête $e \iff p_v$ est **strictement** en dessous de v_d^*
sortie avant l'arête $e \iff p_v$ est au-dessus de **ou sur** v_g^*

et pour une arête grasse

sortie après l'arête $e \iff p_v$ est en dessous de **ou sur** v_d^*
sortie avant l'arête $e \iff p_v$ est **strictement** au dessus de v_g^*

Les tests de comparaison pour une sortie par la chaîne basse sont modifiés de façon similaire.

Reprenons maintenant le test $u \prec \mathcal{E}^s(u)$ utilisé dans notre algorithme de balayage de graphe de visibilité (exposé au chapitre 2). En notant $u = (p, q)$ et $\mathcal{E}^s(u) = (q, r)$, le test s'écrit pour des scènes de polygones en position générale

$$u \prec \mathcal{E}^s(u) \iff r \text{ est au-dessus de } (p, q).$$

La figure 5.7 montre les configurations dégénérées possibles de u et $\mathcal{E}^s(u)$ quand p, q et r sont alignés et les configurations générales simulées correspondantes.

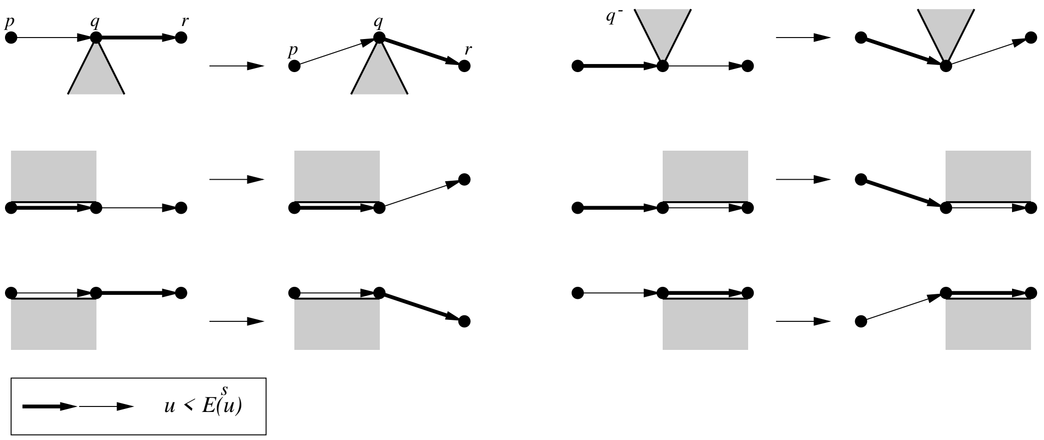


FIG. 5.7 – *Configurations dégénérées pour le test $u \prec \mathcal{E}^s(u)$ et configurations générales associées.*

Pour le test des arêtes de sortie dans le calcul de vue, la situation de test impose la géométrie de la configuration et la modification à apporter au test est immédiate. Ici, la situation (c'est-à-dire la donnée de u et $\mathcal{E}^s(u)$) ne nous donne pas toutes les informations géométriques nécessaires pour modifier immédiatement le test. Nous devons faire quelques tests supplémentaires pour identifier la configuration, et le test $u \prec \mathcal{E}^s(u)$ s'écrit alors

$$u \prec \mathcal{E}^s(u) \iff \begin{array}{l} r \text{ est strictement au-dessus de } (p, q) \\ \textbf{ou} \\ p = q^- \\ \textbf{ou} \\ q^- \text{ est strictement au-dessus de } (p, q) \end{array}$$

En revanche, pour le balayage des chaînes d'arêtes dans le même algorithme, la géométrie locale est connue et il suffit de faire attention au signe du déterminant (positivité ou positivité stricte) de façon similaire aux modifications faites pour le calcul de vue.

L'avantage de notre méthode est qu'elle ne nécessite pas de remaniements profonds des programmes. Lors des tests des signes des déterminants, nous devons juste savoir à quel cas général le cas dégénéré doit être assimilé, et parfois préciser le test ; mais la structure même du programme reste inchangée. Notre méthode est aussi consistante, puisqu'elle s'appuie directement sur la géométrie de la configuration.

Cette méthode est très liée aux problèmes de visibilité et quand la visibilité ne peut être utilisée pour guider le choix (par exemple pour choisir la chaîne d'arêtes de sortie dans le calcul d'une vue), un choix arbitraire doit être fait et de façon consistante. Cependant, la philosophie derrière cette méthode – assimiler un cas dégénéré au cas général le plus « simple » – pourrait trouver des applications à d'autres problèmes.

5.2 Traitement des imprécisions numériques

Les algorithmes étudiés sont programmés avec l'hypothèse implicite que les calculs sont effectués de façon exacte. Cependant les nombres ne sont représentés qu'avec une précision limitée dans les processeurs. Cette limitation peut ne pas avoir beaucoup d'importance pour les calculs effectués avec des nombres entiers : le risque principal encouru est un débordement des calculs qui peut être évité si les entiers considérés ne sont pas trop grands. Les calculs effectués avec des réels en virgule flottante posent en revanche des problèmes plus importants. En effet, les réels représentés en machine ne sont qu'une approximation des nombres « réels » et des problèmes d'arrondis se posent à la fois dans la représentation des nombres et dans les résultats des calculs.

Ces problèmes d'arrondis vont parfois très loin : deux expressions représentant la même expression mathématique mais qui diffèrent par l'ordre dans lequel les opérations sont effectuées, le degré de factorisation ..., peuvent donner deux résultats numériques différents.

Les imprécisions numériques peuvent se propager et s'amplifier dans le programme jusqu'à donner des résultats aberrants. Ainsi l'instabilité numérique des systèmes chaotiques est bien connue : par exemple, les deux suites définies par récurrence $p_{n+1} = p_n + rp_n(1 - p_n)$ et $q_{n+1} = (1 + r)q_n - rq_n^2$ (où r est un paramètre réel) sont mathématiquement identiques ; cependant au bout d'une dizaine d'itérations les valeurs de p_n et q_n diffèrent de plus en plus jusqu'à devenir décorrélées.

Mais des erreurs plus subtiles peuvent se produire, comme pour le calcul de la suite définie par

$$u_0 = 2,\ u_1 = -4 \text{ et } u_{n+1} = 111 - \frac{1130}{u_n} + \frac{3000}{u_n u_{n-1}},$$

dont la limite est $\lim u_n = 6$, et qui cependant converge numériquement vers 100 sur presque toutes les machines.

Nous exposons d'abord les problèmes spécifiques posés par les imprécisions des calculs numériques à nos applications. Nous exposons ensuite plusieurs méthodes pour palier à ces problèmes sans avoir à recourir à des techniques de calculs exacts.

5.2.1 Calculs d'un déterminant

Après avoir modifié notre programme de calcul de graphe de visibilité pour qu'il traite correctement les dégénérescences, celui-ci calcule correctement le graphe de visibilité de la scène dégénérée de la figure 5.8(a). Cependant, si nous appliquons préalablement par calcul numérique une rotation à la scène (figure 5.8b) le programme se plante.

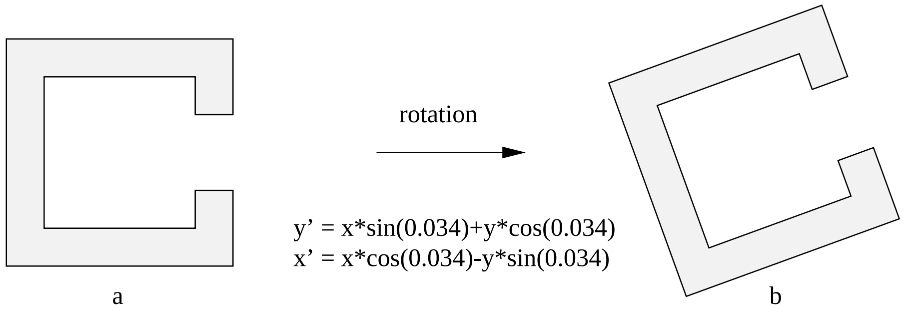
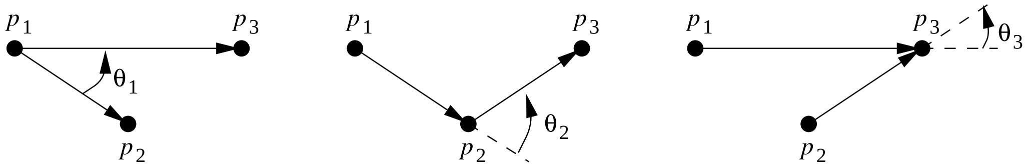


FIG. 5.8 – *Rotation numérique d'une scène test.*

Nous avons vu que le calcul du déterminant de trois points du plan était une opération de base de cet algorithme et du programme qui en résulte. Or le déterminant de trois points p_1, p_2 et p_3 peut s'écrire de trois façons différentes selon les deux côtés du triangle (p_1, p_2, p_2) qui sont choisis. Plus précisément, nous pouvons écrire ce déterminant selon les trois expressions suivantes (figure 5.9) :



$$
\begin{aligned}
D_1(p_1, p_2, p_3) = \left|\overrightarrow{p_1p_2}, \overrightarrow{p_1p_3}\right| &= (x_2 - x_1)(y_3 - y_1) - (y_2 - y_1)(x_3 - x_1) \\
&= \|\overrightarrow{p_1p_2}\|\,\|\overrightarrow{p_1p_3}\| \sin\theta_1 \\
D_2(p_1, p_2, p_3) = \left|\overrightarrow{p_1p_2}, \overrightarrow{p_2p_3}\right| &= (x_2 - x_1)(y_3 - y_2) - (y_2 - y_1)(x_3 - x_2) \\
&= \|\overrightarrow{p_1p_2}\|\,\|\overrightarrow{p_2p_3}\| \sin\theta_2 \\
D_3(p_1, p_2, p_3) = \left|\overrightarrow{p_1p_3}, \overrightarrow{p_2p_3}\right| &= (x_3 - x_1)(y_3 - y_2) - (y_3 - y_1)(x_3 - x_2) \\
&= \|\overrightarrow{p_1p_3}\|\,\|\overrightarrow{p_2p_3}\| \sin\theta_3
\end{aligned}
$$

FIG. 5.9 – *Signification géométrique des déterminants.*

Bien que mathématiquement les trois quantités exprimées par ces déterminants soient égales, si les calculs sont effectués en arithmétique flottante alors, comme les opérations sont effectuées sur des variables différentes et dans des ordres différents, les résultats numériques sont différents pour chacun des déterminants. Ces différences deviennent critiques quand les points sont presque alignés, comme le montre l'expérience suivante où nous avons effectué les calculs avec des réels en virgule flottante double précision.

Nous avons programmé les trois déterminants, soit par exemple

```
double D1(double x1,double y1,double x2,double y2,double x3,double y3)
{
  return (x2-x1)*(y3-y1)-(y2-y1)*(x3-x1);
}
```

pour le déterminant D_1. Nous avons ensuite pris trois points alignés ($p_1 = (-3.2, 2)$, $p_2 = (1.3, 2)$, $p_3 = (5.9, 2)$) et nous leurs avons appliqué numériquement une rotation de i degrés :

```
double co = cos(i*0.01745329252), si = sin(i*0.01745329252);
xr1 = x1*co-y1*si; yr1 = x1*si+y1*co;
xr2 = x2*co-y2*si; yr2 = x2*si+y2*co;
xr3 = x3*co-y3*si; yr3 = x3*si+y3*co;
```

Enfin, nous avons calculé et affiché les trois déterminants :

```
cout<<D1(xr1,yr1,xr2,yr2,xr3,yr3)<<endl;
cout<<D2(xr1,yr1,xr2,yr2,xr3,yr3)<<endl;
cout<<D3(xr1,yr1,xr2,yr2,xr3,yr3)<<endl;
```

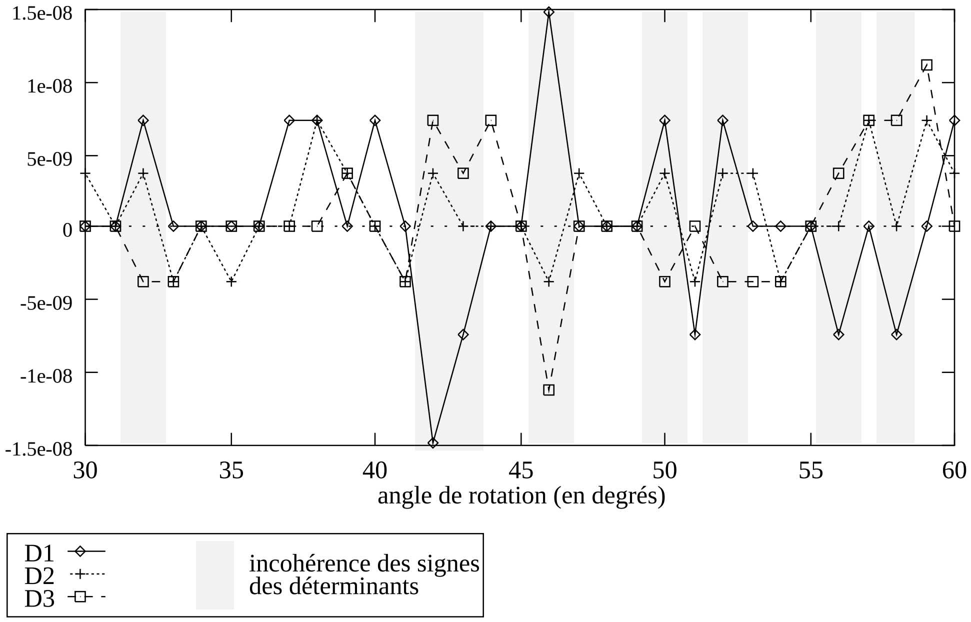


FIG. 5.10 – *Valeurs calculées par les trois déterminants : inversion du signe des résultats.*

Le diagramme de la figure 5.10 affiche les valeurs des déterminants obtenues (avec une arithmétique flottante double précision à la norme IEEE 754) en fonction de i, pour i variant de 30° à 60° (par pas de 1°).

Nous voyons que les valeurs calculées sont différentes pour les diverses rotations appliquées, mais surtout que les valeurs sont différentes entre les trois déterminants pour une même rotation et, ce qui est plus grave, que les signes des trois déterminants sont différents dans certains cas (parties grisées de la figure 5.10). Cela signifie que pour certains points p_1, p_2 et p_3, le signe du déterminant est différent selon que nous utilisons la formule de D_1, de D_2 ou de D_3. Ce problème subsiste toujours lorsque nous employons les expressions développées des déterminants à la place des expressions factorisées utilisées ici.

La cohérence géométrique de la situation est perdue : pour le programme, à certains endroits p_3 est au-dessus de (p_1, p_2), et à d'autres endroits p_3 est en dessous de (p_1, p_2). Ces incohérences expliquent l'échec de notre programme sur la scène à laquelle nous avions appliqué une rotation.

5.2.2 Calculs exacts

La solution la plus directe pour résoudre les problèmes d'imprécision des calculs numériques est d'effectuer des calculs exacts. Il est possible de définir exactement les nombres algébriques et de faire des calculs algébriques exacts. En effet un nombre algébrique α peut être représenté par une paire $\alpha = (P, I)$ telle que α soit la racine du polynôme P dans l'intervalle I. Il est alors possible de comparer ces nombres et d'effectuer dessus les quatre opérations arithmétiques ([Yapre]).

Cependant ces calculs font appel à des techniques lourdes et sont très coûteux (la théorie montre même que ces calculs sont doublement exponentiels en temps). Ils sont donc inutilisables en pratique et plutôt réservés à des programmes de calcul formel.

Cette notion de calcul exact étant trop forte, Yap et Dubé [YD95] ont défini un *paradigme de calcul exact*. Ce paradigme garantit que

- toutes les valeurs numériques sont représentées exactement ;
- toutes les décisions de branchement se font sans erreur.

Ce paradigme nécessite l'emploi de calculs multi-précision, c'est-à-dire de calculs mettant en jeu des nombres réels dont la taille de la mantisse (resp de l'exposant) peut être arbitrairement grande. Cependant ces calculs doivent être menés avec précaution pour éviter d'avoir des nombres représentés avec une précision trop grande, c'est-à-dire avec des chiffres non significatifs. Les nombres sont alors assortis d'un intervalle d'erreur.

Les manipulations de nombres s'accompagnent de manipulations d'expressions, qui sont représentées par des arbres dont les nœuds contiennent soit directement une valeur, soit une opération s'appliquant aux valeurs représentées par les sous-arbres du nœud. Les intervalles d'erreurs associés aux nombres sont alors redéfinis dynamiquement en fonction des besoins de précision.

Dans les calculs « paresseux » de l'équipe de St Etienne [BJMM93], les intervalles d'erreur sont ajustés depuis les feuilles vers la racine de l'arbre, jusqu'à ce que l'erreur sur la valeur finale (racine de l'arbre) satisfasse la précision demandée. Yap et Dubé [YD95] défendent plutôt des calculs « dirigés par la précision », où la précision, c'est-à-dire la taille de l'intervalle d'erreur, est fixée à la racine et est propagée vers les feuilles de manière à ce qu'elle soit garantie.

Yap et Dubé [YD95] défendent vivement le paradigme de calcul exact, car ce paradigme a une portée générale et permet de résoudre directement tous les problèmes de robustesse dans les programmes, alors qu'il serait selon eux impensable de recréer pour chaque programme une version robuste *ad hoc* de ce programme. Ils reconnaissent cependant que les calculs multi-précision ont un coût, qui peut être trop important pour certaines applications.

5.2.3 ε-géométrie

D'autres chercheurs, remarquant que dans certaines applications les données elles-mêmes sont imprécises, ont élaboré une géométrie « approximative », qu'ils ont appelée ε-géométrie. Dans cette géométrie, Guibas, Salesin et Stolfi [GSS89] modélisent les erreurs en supposant que les coordonnées des points ne sont connues qu'à une précision ε près. Pour un prédicat P défini sur un ensemble $\mathcal{O}$ de points, ils définissent le prédicat ε-P tel que

- $\varepsilon > 0$: ε-$P(p)$ est vrai $\iff \exists p' \in \mathcal{O}\ \|pp'\| < \varepsilon,\ P(p')$ est vrai ;
- $\varepsilon < 0$: ε-$P(p)$ est vrai $\iff \forall p' \in \mathcal{O}\ \|pp'\| < |\varepsilon|, P(p')$ est vrai.

Alors chaque test géométrique $P(p)$ ne renvoie plus `vrai` ou `faux`, mais renvoie deux limites f et v telles que le test ε-$P(p)$ est faux pour $\varepsilon < f$, vrai pour $\varepsilon \geq v$ et a un résultat non défini sinon. En particulier, si f est positive, alors $P(p)$ est « vraiment » fausse, et si v est négative, alors $P(p)$ est « vraiment » vraie.

Les limites renvoyées dépendent de la géométrie de la situation. Par exemple, un point r est strictement en dessous de (p, q) si le cercle centré en r et de rayon ε n'intersecte pas la zone au-dessus de (p, q) limitée par les bitangentes aux deux cercles centrés respectivement en p et q et de rayon ε (figure 5.11).

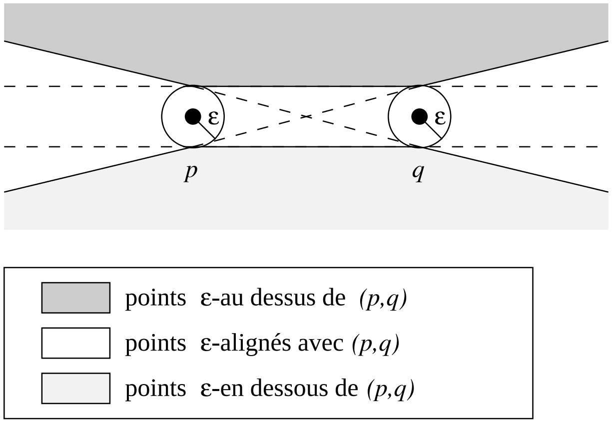


FIG. 5.11 – *ε-position relative par rapport à deux points.*

Les auteurs font aussi intervenir dans ces limites des encadrements d'erreurs liés aux problèmes d'arrondis des calculs effectués.

Nous cherchons ici à mettre en œuvre les méthodes les plus simples possibles pour traiter les problèmes d'imprécisions. Ces méthodes, bien qu'étudiées dans le cadre de notre algorithme de calcul de graphe de visibilité, ont une portée générale qui dépasse le seul cadre de notre programme et pourraient être réutilisées. Il s'agit en fait d'étudier le calcul des déterminants et leur utilisation.

5.2.4 Cohérence des calculs

Nous savons que des erreurs de calculs vont se glisser lors de l'exécution de notre programme de calcul de graphe de visibilité. Nous ne cherchons pas alors à obtenir un résultat exact, mais plutôt un résultat cohérent : nous voulons que le graphe de visibilité calculé soit un graphe de visibilité valide, même s'il correspond à une scène légèrement différente de la scène initiale.

Les problèmes d'inconsistance proviennent des résultats contradictoires obtenus par les différentes formules du déterminant. Pour être sûrs de ne plus avoir ces inconsistances, nous devons vérifier que pour trois points donnés la même formule de déterminant sera toujours utilisée. Nous devons faire alors attention à l'ordre dans lequel les points sont donnés dans le calcul d'un déterminant. En effet, nous avons les égalités suivantes entre *formules* des déterminants :

$$\begin{array}{lll}
D_1(p,r,q)=-D_1(p,q,r) & D_2(p,r,q)=-D_3(p,q,r) & D_3(p,r,q)=-D_2(p,q,r) \\
D_1(q,p,r)=-D_2(p,q,r) & D_2(q,p,r)=-D_1(p,q,r) & D_3(q,p,r)=-D_3(p,q,r) \\
D_1(q,r,p)=\ \ D_2(p,q,r) & D_2(q,r,p)=\ \ D_3(p,q,r) & D_3(q,r,p)=\ \ D_1(p,q,r) \\
D_1(r,p,q)=\ \ D_3(p,q,r) & D_2(r,p,q)=\ \ D_1(p,q,r) & D_3(r,p,q)=\ \ D_2(p,q,r) \\
D_1(r,q,p)=-D_3(p,q,r) & D_2(r,q,p)=-D_2(p,q,r) & D_3(r,q,p)=-D_1(p,q,r)
\end{array}$$

Dans notre programme de calcul de graphe de visibilité, nous avons inversé le sens des arêtes de l'arbre d'horizon inférieur : de cette façon, l'arbre inférieur est vraiment égal à l'arbre supérieur de la scène tournée de 180°. Nous avons alors utilisé le même code pour les deux arbres.

FIG. 5.12 – *Tests de position relative de trois points dans les arbres d'horizon.*

Considérons maintenant trois points p, q et r reliés successivement par une arête du graphe de visibilité. Alors à un moment le programme effectue le test $u \prec \mathcal{E}^s(u)$ avec $u = (p,q)$ et $\mathcal{E}^s(u) = (q,r)$; ce test est effectué par la comparaison $D_1(p,q,r) > 0$. Le programme effectue aussi à un autre moment le test $v \prec \mathcal{E}^i(v)$ avec $v = (r,q)$ et $\mathcal{E}^i(v) = (q,p)$; comme nous utilisons le même code pour les deux arbres d'horizon, ce test est effectué par la comparaison $D_1(r,q,p) > 0$ (figure 5.12).

Or nous avons vu que la formule $D_1(r,q,p)$ est la même que la formule $D_3(p,q,r)$. Pour ce triplet de points, différentes formules sont utilisées (au moins D_1 et D_3) pour les calculs de déterminant, ce qui explique les incohérences observées.

Nous avons alors écrit un code spécifique pour les arbres d'horizon inférieurs, de façon à ces que les formules utilisées pour les calculs de déterminant soient effectivement les mêmes que celles utilisées pour les arbres supérieurs. Par exemple, le test $v \prec \mathcal{E}^i(v)$ a été réécrit $D_3(r,q,p) > 0$, pour calculer effectivement $D_1(p,q,r)$ comme dans l'arbre supérieur.

Cependant cette cohérence ne peut être obtenue directement que si une certaine position relative des points est connue. Par exemple, dans le test $u \prec \mathcal{E}^s(u)$ nous savons que q est « entre » p et r. Cette information n'est malheureusement pas connue quand nous balayons les chaînes d'arêtes pour trouver la nouvelle arête de l'arbre d'horizon. Pour tester si l'arête partant de p est tangente à la chaîne en q, nous comparons la position relative des points p, q et $r = e^s(q)$. Or la position de p n'est pas connue : p peut être entre q et r, ou peut être derrière q et r. Suivant la position de p, le déterminant calculé ne sera pas le même que celui utilisé dans les tests $u \prec \mathcal{E}^s(u)$ et une nouvelle incohérence géométrique apparaîtra (figure 5.13).

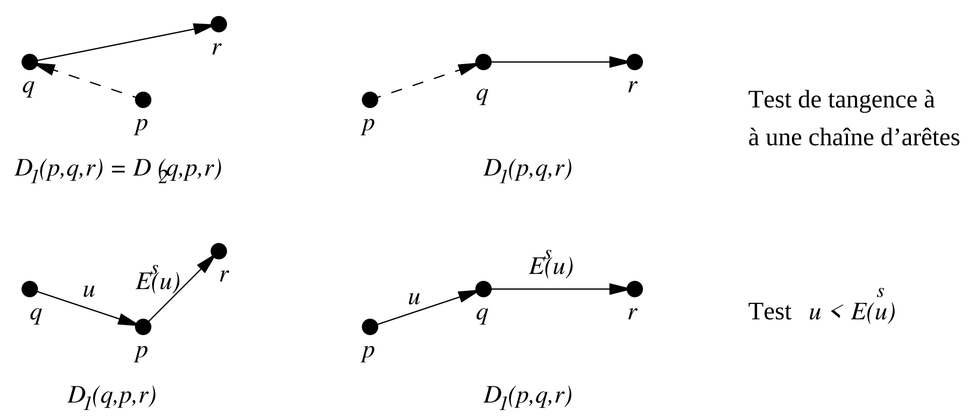


FIG. 5.13 – *Différences de déterminants dues à la différence de position du point p avec l'arête balayée de la chaîne.*

Nous pourrions, pour être sûrs de toujours utiliser la même formule de déterminant, utiliser un déterminant *lexicographique*, c'est-à-dire calculer pour trois points p_i, p_j et p_k le déterminant $D_1(p_i, p_j, p_k)$ où les points sont ordonnés selon l'ordre de leur numérotation (dans cet exemple $i < j < k$). Nous retrouvons alors les problèmes rencontrés dans l'utilisation de la simulation de la simplicité pour ôter les dégénérescences : le graphe de visibilité obtenu est différent selon les numérotations des points d'une même scène.

Nous voulons garder une certaine robustesse vis à vis de l'ordre dans lequel les données sont rentrées. Nous ne voulons pas faire de tests de position supplémentaires pour choisir le bon déterminant à calculer : ces tests seraient sujets eux-mêmes aux imprécisions de calcul et pourraient provoquer des incohérences. Nous cherchons alors à calculer un déterminant « robuste », qui reste cohérent dans tous ses emplois dans le programme.

5.2.5 Opérations unanimes

Nous voulons qu'à chaque requête concernant la position relative de trois points

p, q et r effectuée pendant le déroulement du programme, la réponse donnée soit toujours la même. Comme ces calculs de position relative peuvent faire intervenir trois formulations différentes de déterminant (D_1, D_2 et D_3), pour obtenir l'unanimité entre ces trois formulations nous disons que r est (strictement) au-dessus de (p, q) si et seulement si les signes des trois déterminants sont tous positifs, sinon r est supposé être en dessous de ou aligné avec (p, q). Le test `au_dessus(p,q,r)`, qui indique si r est au-dessus de (p, q), s'écrit maintenant

```
au_dessus(p,q,r) :
   si  D1(p,q,r)<=0.0 → faux
   sinon
      si  D2(p,q,r)<=0.0 → faux
      sinon
         → D3(p,q,r)>0.0
```

et la même modification est faite au test `en_dessous(p,q,r)` (qui indique si r est en dessous de (p, q)).

Les problèmes sont posés par les triplets de points (p, q, r) pour lesquels les signes des déterminants sont différents. Pour ces triplets, les tests `au_dessus(p,q,r)` et `en_dessous(p,q,r)` sont maintenant tous les deux faux. Cela signifie implicitement que les trois points sont alignés. Or comme notre programme traite correctement les cas dégénérés, il traite maintenant correctement les triplets qui causaient des incohérences locales.

Plus précisément, nous n'avons pas besoin de modifier notre programme car nous avons pris soin de ne pas particulariser les cas dégénérés mais de les assimiler directement à des cas non dégénéres. En effet, rappelons que dans notre programme, les tests de la forme

```
si au_dessus(p,q,r) alors traitement1
sinon                      traitement2
```

signifient que `traitement1` concerne le cas non dégénéré où r est **strictement** au-dessus de (p, q), et que `traitement2` concerne les autres cas – cas dégénéré inclus – où r est en dessous de **ou aligné avec** (p, q). Si le triplet (p, q, r) provoque des incohérences, alors `au_dessus(p,q,r)` est faux et le triplet, implicitement considéré comme dégénéré, reçoit bien le traitement adéquat.

Grâce à ces déterminants « unanimes », nous obtenons une cohérence *locale*. Malheureusement, les imprécisions numériques peuvent provoquer des incohérences *globales*. Par exemple, après avoir effectué une rotation de 11° aux points $p_1 = (-7.3, 2.2)$, $p_2 = (-2.1, 2.2)$, $p_3 = (1.4, 2.2)$ et $p_4 = (3.2, 2.2)$, les calculs de déterminants unanimes donnent pour résultat :

```
au_dessus(2,3,4) : vrai ; en_dessous(2,3,4) : faux
au_dessus(1,3,4) : faux ; en_dessous(1,3,4) : faux
au_dessus(1,2,4) : faux ; en_dessous(1,2,4) : vrai
au_dessus(1,2,3) : faux ; en_dessous(1,2,3) : faux
```

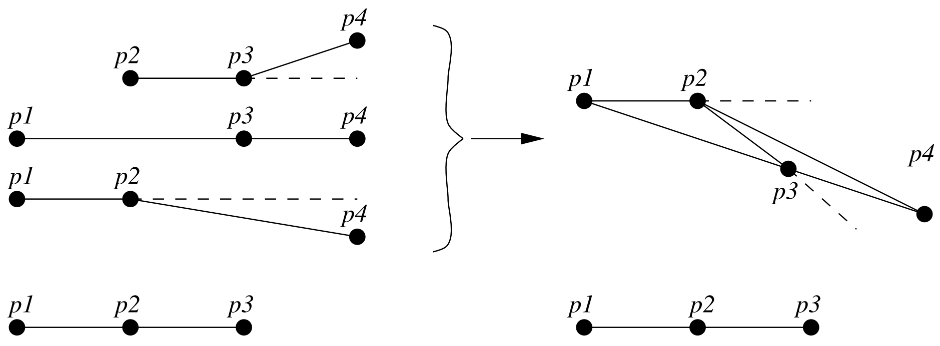


Fig. 5.14 – *Incohérences des résultats locaux de déterminant sur une situation globale.*

La figure 5.14 montre pour chacun des triplets de points les situations géométriques correspondant aux résultats des calculs. Nous voyons que ces résultats ne sont pas cohérents : il n'existe pas de configuration de quatre points compatible avec ces résultats.

5.2.6 Calculs d'erreurs

Nous devons, pour assurer une cohérence globale, mieux étudier les imprécisions numériques qui se produisent dans le calcul d'un déterminant. La norme IEEE 754 sur les calculs en virgule flottante définit un standard sur la représentation des nombres réels en virgule flottante double précision et sur les arrondis effectués sur les résultats d'opérations sur de tels nombres. Cette norme spécifie notamment que le résultat d'une opération algébrique ($+,-,*,/,\sqrt{}$) est arrondi à une demie *ulp* (Unit in Last Position) près, et que deux nombres x et x' consécutifs, c'est-à-dire ne différant que d'une ulp, vérifient

$$1.1\ 10^{-16} < 0.5^{53} < \frac{(x'-x)}{x} < 0.5^{52} < 2.3\ 10^{-16}\ .$$

Cela signifie que lorsque le processeur effectue une opération algébrique sur deux nombres, il renvoie une valeur arondie x^* qui vérifie par rapport au résultat exact x de l'opération : $x^* = (1+\varepsilon)x$, où l'erreur est bornée par $|\varepsilon| < u = 2.3\ 10^{-16}$. La valeur exacte x se situe alors dans l'intervalle $[x^* - \varepsilon|x^*|, x^* + \varepsilon|x^*|]$.

Si nous appliquons ces bornes (en ne gardant à la fin que les termes du premier ordre en u) à toutes les opérations effectuées lors du calcul du déterminant

$$D_1(p1,p2,p3) = (x2-x1)(y3-y1) - (y2-y1)(x3-x1)$$

alors nous obtenons l'encadrement

$$D_1 \in D_1^* \pm (u|D_1^*| + 3uM_1^*)$$

où $M_1 = |x2-x1||y3-y1| + |y2-y1||x3-x1|$.

De façon générale, nous notons $D_i^{min} = D_i - (u|D_i| + 3uM_i)$ et $D_i^{max} = D_i + (u|D_i| + 3uM_i)$ les bornes d'erreurs du déterminant D_i. Nous disons alors que le déterminant D_i est strictement positif (resp. négatif) si les deux bornes de l'intervalle sont strictement positives (resp. négatives), c'est-à-dire si $D_i^{min} > 0$ (resp. $D_i^{max} < 0$). Comme les encadrements d'erreur du determinant D_i certifient que la « vraie » valeur du déterminant D est comprise dans l'intervalle $[D_i^{min}, D_i^{max}]$, alors si D_i^{min} est strictement positif, nous pouvons en déduire que le déterminant D est strictement positif. En revanche, si l'intervalle d'erreur contient 0, cela ne nous permet pas d'affirmer que D n'est pas positif.

En effet, nous avons repris l'expérience de calculs de déterminants de la partie 5.2.1. Nous avons calculé les déterminants D_1, D_2 et D_3 sur les points $p_1 = (-3345671, 1)$, $p_2 = (1, 1.000000001)$ et $p_3 = (6654312, 1)$ après leur avoir appliqué des rotations (figure 5.15a). Nous avons aussi calculé les bornes d'erreur de ces déterminants (figure 5.15b).

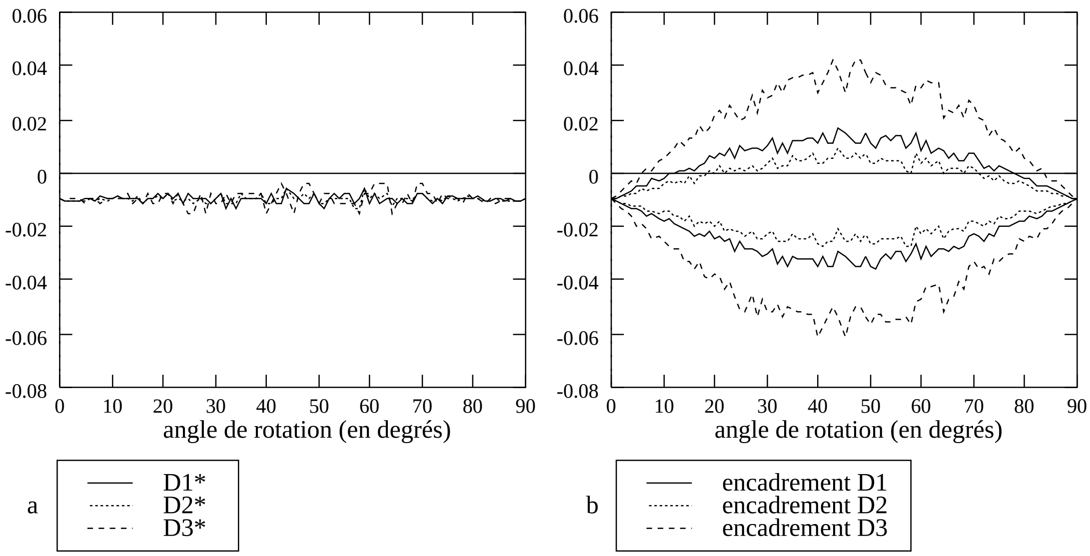


FIG. 5.15 – *Précision des calculs numériques.* **a.** *Déterminants calculés.* **b.** *Encadrement calculé des déterminants.*

Nous voyons que les valeurs des encadrement sont différentes selon les déterminants. En effet, si les points sont relativement alignés, alors $M_1 \approx \|\overrightarrow{p_1, p_2}\| \|\overrightarrow{p_1, p_3}\| \sin 2\theta$, $M_2 \approx \|\overrightarrow{p_1, p_2}\| \|\overrightarrow{p_2, p_3}\| \sin 2\theta$ et $M_1 \approx \|\overrightarrow{p_1, p_3}\| \|\overrightarrow{p_2, p_3}\| \sin 2\theta$, où θ est l'angle de la droite passant par les trois points. Nous voyons que pour certains angles, certains encadrements contiennent 0, et les autres non.

En fait nous devons avoir ici une unanimité des déterminants, non pas pour affirmer que le déterminant est positif (resp. négatif), mais pour affirmer qu'il ne l'est pas et qu'il doit être considéré comme nul. Nous n'avons donc plus

$$D \text{ positif} \iff D_1 \text{ positif } \textbf{et } D_2 \text{ positif } \textbf{et } D_3 \text{ positif}$$

comme pour les déterminants unanimes, mais plutôt

$$D \text{ positif} \iff D_1 \text{ positif } \textbf{ou } D_2 \text{ positif } \textbf{ou } D_3 \text{ positif}$$

et les fonctions de position relative s'écrivent alors :

au_dessus(p,q,r) :
  si D_1^{min}(p,q,r)>0.0 $\rightarrow$ vrai
  sinon
    si D_2^{min}(p,q,r)>0.0 $\rightarrow$ vrai
    sinon
      $\rightarrow$ D_3^{min}(p,q,r)>0.0

en_dessous(p,q,r) :
  si D_1^{max}(p,q,r)<0.0 $\rightarrow$ vrai
  sinon
    si D_2^{max}(p,q,r)<0.0 $\rightarrow$ vrai
    sinon
      $\rightarrow$ D_3^{max}(p,q,r)<0.0

En reprenant la formule de calcul d'erreur, nous remarquons que l'erreur absolue commise sur le déterminant peut être assez grande : uM est de l'ordre de grandeur de ud^2, où d est la longueur du plus grand côté du triangle (p_1, p_2, p_3). Il ne faudrait pas que l'encadrement calculé soit tellement grand qu'il contienne alors 0 et que le déterminant soit considéré comme nul, alors que le déterminant est *a priori* légitimement positif (resp. négatif).

Nous avons choisi les points utilisés pour les calculs de la figure 5.15 obtenir une distance d importante. La figure 5.15(a) montre que les valeurs D^* calculées des déterminants se répartissent autour de -0.01. Le déterminant peut être alors raisonnablement considéré comme négatif. Cependant, quand nous examinons les valeurs encadrant les déterminants (figure 5.15b), nous voyons que dans certains cas les trois encadrements contiennent 0, et que d'après nos décisions, ils doivent être considérés comme nuls.

Notre encadrement des erreurs de calculs est-il alors voué à l'échec ? Remettons certaines choses en perspective. Si nous reprenons l'interprétation géométrique du déterminant, nous voyons que la valeur du déterminant est de l'ordre de grandeur de $\sin\theta\, d^2$, où θ est l'un des angles du triangle (p_1, p_2, p_3). Or nous calculons une erreur possible de ud^2 sur cette valeur de $\sin\theta\, d^2$, avec $u = 2.3\ 10^{-16}$. Cette erreur uM ne devient importante par rapport à la valeur du déterminant que quand θ est très petit – de l'ordre de 10^{-15} radians –, c'est-à-dire quand les trois points doivent être considérés comme alignés. Si nous reprenons notre exemple « fautif », nous nous apercevons en fait que les angles du triangle formé par les trois points sont de l'ordre de 10^{-15} radians, et donc que les points peuvent être légitimements considérés comme alignés, malgré des déterminants autour de -0.01.

Si nous reprenons le calcul du graphe de visibilité de la scène de la figure 5.8, qui nous posait problème à cause des incohérences globales entre quatre points (figure 5.14), grâce à la maitrise des erreurs de calculs, ces point sont tous considérés comme alignés et le programme s'exécute correctement.

5.2.7 Coût du traitement des imprécisions numériques

Nous avons mis en place deux méthodes pour lutter contre les erreurs de calculs numériques des déterminants simples : les déterminants unanimes et les déterminants avec bornes d'erreur. Nous avons mis en place ces méthodes pour essayer de lutter contre les imprécisions de calcul avec un surcoût ausis faible que possible.

Nous avons effectué des mesures sur nos programmes pour évaluer ces surcoûts. Nous avons pris 10 scènes aléatoires de 600 triangles, de densité de visibilité croissante (voir chapitre ?), et nous avons exécuté notre programme de calcul de graphe de visibilité, en utilisant respectivement une seule formule de déterminant (déterminant simple), les déterminants unanimes et les déterminants tenant compte des bornes d'erreur de calcul.

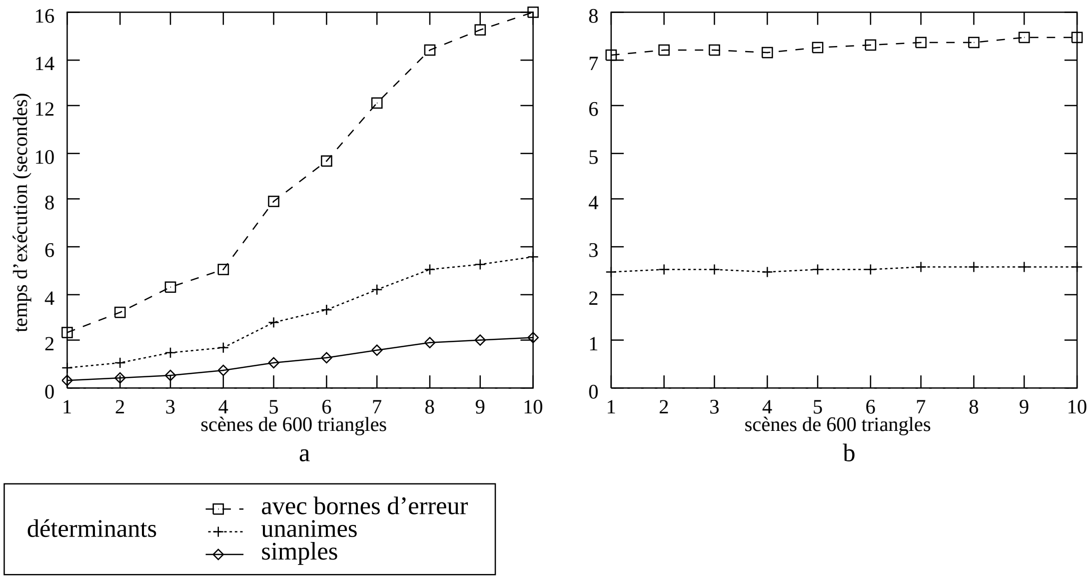


FIG. 5.16 – *Coûts de calcul des déterminants.* **a.** *Temps passé par le programme de calcul de graphe de visibilité à calculer des déterminants.* **b.** *Rapport de ces temps avec celui des déterminants simples.*

Nous avons tout d'abord isolé le temps passé par le programme dans le calcul des déterminants. La figure 5.16(a) affiche ce temps (en secondes) pour chacune des scènes tests, et la figure 5.16(b) affiche le rapport du temps utilisé par les déteminants avec borne d'erreur (resp. unanimes) avec le temps utilisé par les déterminants simples. Nous voyons que les déterminants unanimes coûtent environ 2.5 fois plus de temps que les déterminants simples, et que les calculs de bornes d'erreur coûte environ 7 fois plus de temps.

Ce surcoût est assez important. Cependant nous devons aussi examiner ce surcoût de manière plus relative, en regardant le surcoût occasionné par rapport à l'ensemble du programme. Le calcul des déterminants prend environ 18 % du temps total d'exécution du programme. Le temps d'exécution du programme devrait être donc multiplié par $1+(r-1)*0.18$, où r est le rapport entre les déterminants (figure 5.16b), soit un surcoût d'environ 1.3 (resp 2.1) pour l'utilisation de déterminants unanimes (resp. avec bornes d'erreur).

La figure 5.17(a) montre le temps d'exécution total des programmes, et la figure 5.17(b) les rapports de ces temps par rapport à celui du programme utilisant les déterminants simples. Nous voyons qu'effectivement les surcoûts mesurés sont concordants avec ceux que nous venons de calculer, et que le calcul de bornes d'erreur double le temps d'exécution du calcul de vues.

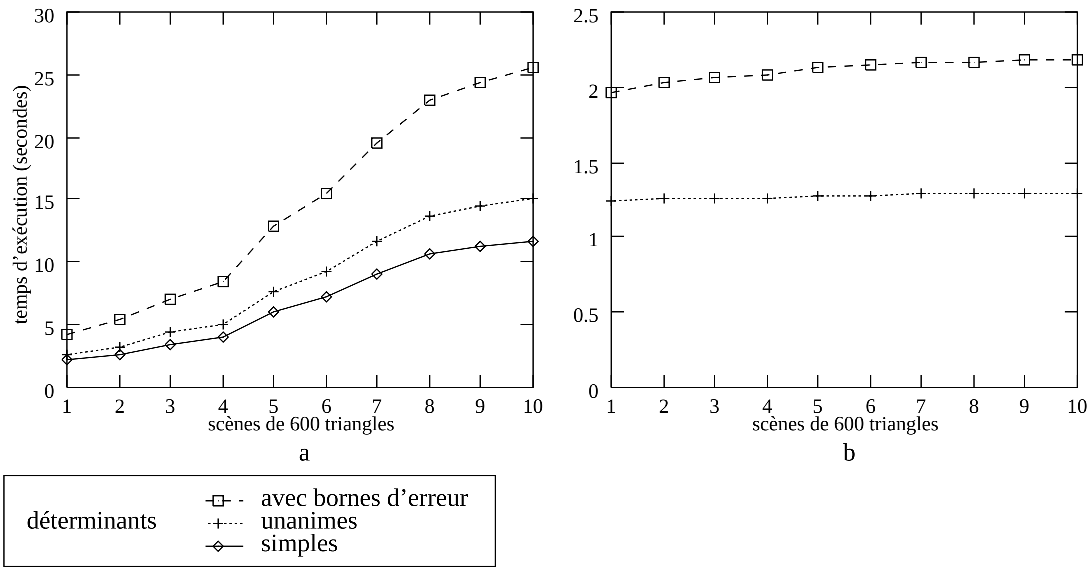


FIG. 5.17 – *Calculs de graphe de visibilité avec plusieurs déterminants.* **a.** *Temps d'exécution des programmes.* **b.** *Rapport de ces temps avec le programme utilisant les déterminants simples.*

5.2.8 Conclusion

Le traitement des problèmes posés par les imprécision des calculs en virgule flottante ne peut pas être séparé des problèmes de dégénérescences.

En effet, les configurations pour lesquelles se produisent des incohérences numériques sont des configurations qui doivent être considérées comme des configurations dégénérées, puisque les calculs ne permettent pas de faire la différence entre les situations générales auxquelles elles appartiennent. Ceci est encore plus marqué dans la ε-géométrie, où les ensembles de points ε-dégénérés sont continus et donc ont une certaine importance. Cependant, le traitement des dégénérescences ne peut se faire en utilisant les schémas de perturbation symbolique, car les erreurs de calculs entravent la gestion des perturbations. Par exemple, pour le calcul de déterminant utilisant la simulation de simplicité, non seulement le déterminant principal est souvent non nul, mais les imprécisions touchent aussi le calcul des sous-déterminants.

En revanche, notre simulation de simplicité guidée par la visibilité, qui n'utilise que des décisions basées sur la géométrie, s'adapte bien aux traitements des imprécisions numériques.

L'idée directrice majeure que nous avons utilisée pour lutter contre les problèmes d'imprécision de la façon la moins pénalisante possible est la *cohérence.*

Nous avons d'abord voulu être cohérents en essayant d'utiliser, parmi les diverses expressions correspondant à un même calcul, une seule de ces expressions. Malheureusement, nous avons vu que dans notre programme de calcul de graphe de visibilité, nous manquions d'informations pour décider directement quelle expression utiliser.

Nous avons ensuite assuré une cohérence locale en requérant l'unanimité entre les expressions pour donner une réponse positive. Cependant cette cohérence locale est insuffisante dans certains cas pour assurer une cohérence globale.

Nous avons alors utilisé la norme IEEE 754 pour déterminer les marges d'erreurs dans le calcul des déterminants, et tenir compte dans les décisions non pas seulement de la valeur calculée des déterminants, mais aussi des encadrements de cette valeur.

Certain auteurs poussent encore plus loin la modélisation des imprécisions pour garantir la robustesse des programmes ($\varepsilon-$géométrie par exemple). Enfin il reste la possibilité de recourir à des calculs exacts pour éliminer plus directement ces problèmes.

Nous pouvons cependant modérer un peu cette chasse aux imprécisions numériques. Les problèmes que nous avons rencontrés étaient construits de façon artificielle, en effectuant une rotation numérique à des points alignés. Ainsi les coordonnées de ces points avaient vraiment 16 chiffres après la virgule significatifs (en fait 33 bits), c'est-à-dire que le dernier chiffre de la mantisse est important et peut faire basculer le résultat d'un calcul.

Si la précision peut être importante à prendre en compte dans le cadre de simulations numériques par exemple, elle peut être excessive pour d'autres problèmes, comme nos programmes de visibilité par exemple. Les cordonnées des points de la scène ne nécessitent pas une précision excessive et quelques chiffres significatifs après la virgule suffisent. Avec une précision utile plus faible, les problèmes d'imprécision sont alors beaucoup moins importants. Un bon traitement des problèmes d'imprécision des calculs numériques passe peut-être d'abord par un bon conditionnement (si le cadre le permet) des données utilisées.