D.E.A. Intelligence Artificielle et Optimisation Combinatoire,
Université de Paris 8 / Paris 13

Environnement 3D pour Pilotes Automatiques

Farès Belhadj
Sous la direction de M. Pierre Audibert
Laboratoire d'Intelligence Artificielle, Université Paris 8
amsi@ai.univ-paris8.fr

Septembre 2002

Résumé

Nous présentons un environnement 3D temps réel pour l'apprentissage de la planification de trajectoires en vol de pilotes automatiques. Le simulateur d'environnement que nous avons développé gère le comportement de véhicules, précisément des hélicoptères, dans un univers virtuel. Pour suivre un parcours jusqu'au point d'arrivée qui lui a été imposé, chaque véhicule équipé d'un pilote automatique doit — à l'aide d'outils de perception — éviter les obstacles rencontrés — statiques ou dynamiques — tout en gardant une altitude minimale et une vitesse soutenue.

Nous réalisons et comparons deux implémentations de pilotes automatiques, un *automate volant* et un *Perceptron Multi-Couches*. Le premier utilise l'algorithme du *plus sûr chemin* fourni par l'environnement. Pour atteindre le point d'arrivée fixé par l'utilisateur, ce pilote combine la connaissance globale du trajet et la réaction locale aux obstacles, en faisant le meilleur compromis entre la minimalisation des distances et des altitudes. Le deuxième pilote, par le biais du *Perceptron Multi-Couches*, réalise les mêmes objectifs en les améliorant. Dans une première phase d'apprentissage, il est couplé au jeu de *l'automate volant* et reproduit les mêmes comportements. Dans un deuxième temps, l'utilisateur humain intervient aux commandes du véhicule pour affiner les réactions du réseau de neurones. Le nouveau pilote obtenu réalise alors des trajectoires de vol en rase-mottes et garde une vitesse sensiblement constante.

Nous décrivons, pour le processus de création du simulateur, l'implémentation de l'interface de gestion des interactions des objets avec leur environnement. Cette interface est déclinée sous la forme d'une librairie. Celle-ci gère la création des paysages — terre, mer, arbres —, place les objets — obstacles, hélicoptères — et coordonne les mouvements et les interactions de l'ensemble. Trois différents types de terrains sont proposés : paysages fractals (*plasma* ou *Brown-Gauss*), paysage existant (cartes des altitudes) ou labyrinthe.

Nous pouvons placer, dans ces paysages, autant de véhicules que souhaité. Chaque véhicule dispose d'instruments de bord et de capteurs de mesure paramétrables en emplacement comme en direction. Ces capteurs servent à la perception de l'environnement immédiat.
À ce stade, aidés de notre simulateur d'environnement 3D, nous avons intégré et testé nos deux types de pilotes automatiques. Chaque pilote est assigné à un ou plusieurs véhicules créés ; nous observons directement leur comportement à l'écran.

Nous envisageons par la suite d'enrichir la base des véhicules / capteurs afin d'ouvrir la voie à de nouvelles dynamiques de mouvements, comme par exemple

des véhicules terrestres à roues, des robots sur pattes et des avions. Dans cette optique, nous souhaitons aboutir à un environnement complet de construction et de pré-test des systèmes embarqués en robotique mobile.

Remerciements

Je remercie tout particulièrement mon directeur de recherche Monsieur Pierre Audibert pour son aide inestimable, ses conseils précieux et sa contribution à l'élaboration de l'ensemble de mon travail.

Je remercie également tous les membres du laboratoire d'Intelligence Artificielle de Université de Paris 8 pour leur esprit de groupe et d'entre-aide.

Enfin, je voudrais également remercier toute ma famille et en particulier ma femme, ma mère, mon père et ma sœur pour tout le soutien qu'ils m'ont apporté.

Table des matières

Table des figures

Liste des tableaux

Chapitre 1

Introduction

1.1 L'environnement 3D

Nous présentons notre interface pour la simulation d'univers virtuel. L'environnement numérique est créé à partir des données topographiques de terrain, nous y plaçons des arbres et des véhicules volants, en l'occurence des hélicoptères[1].
Nous réalisons un rendu, temps réel, en synthèse d'image de l'ensemble des éléments constituant l'environnement virtuel, cf. chapitre 3. Les paysages du simulateur sont générés aléatoirement ; nous n'utilisons aucune donnée extérieure pour la construction des éléments du paysage[2], cf. chapitre 2.
Notre étude s'oriente vers la réalisation d'un outil d'apprentissage et de pré-test pour la robotique mobile. L'aspect temps réel de l'application donne la possibilité d'effectuer un apprentissage interactif entre module de navigation et utilisateur. Nous implémentons deux pilotes automatiques, un automate déterministe, cf. §4.3, et un *Perceptron Multi-Couches* pour lequel nous effectuons une série d'apprentissages et de tests, cf. §4.4.

[1] Notre interface gère actuellement le comportement et les interactions avec l'environnement d'un ou de plusieurs hélicoptères. Nous avons choisi l'hélicoptère dans une première approximation, mais nous souhaitons élargir la gamme des véhicules gérés.

[2] Nous pouvons, le cas échéant, importer des données topographiques réelles pour effectuer une simulation sur un terrain existant.

1.1.1 Des fractales pour la génération de paysages

Les objets fractals[3] sont au centre de notre processus de génération de paysages. Elles permettent la création des éléments à partir desquels nous construisons notre environnement.

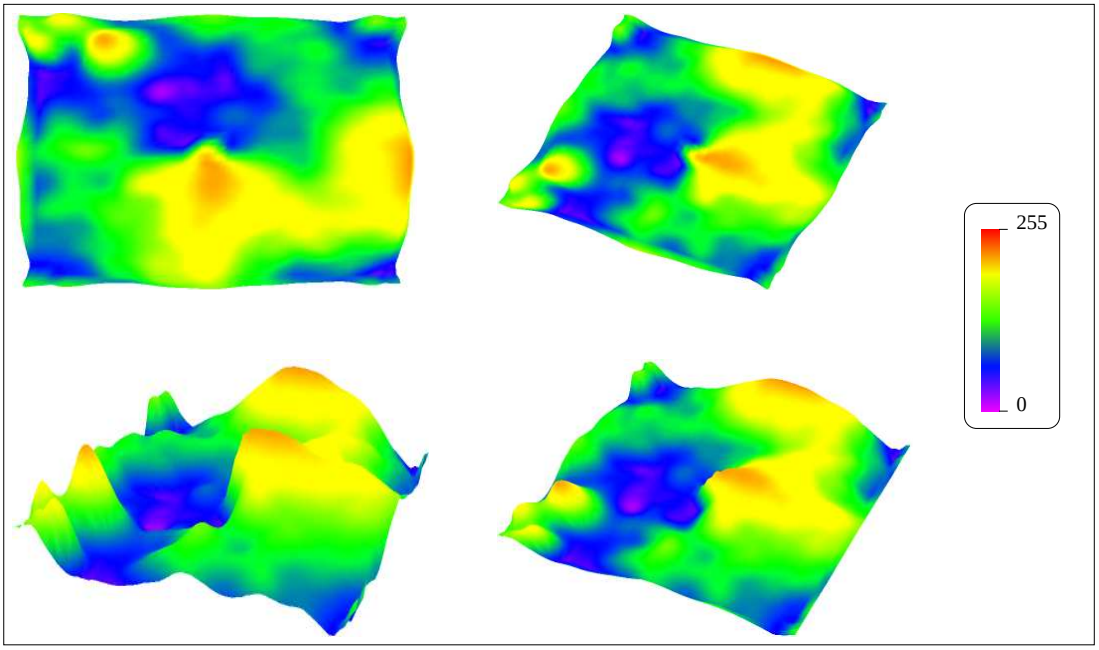


FIG. 1.1 – Différentes vues du plasma élevé en trois dimensions. Les altitudes sont données dans un intervalle $[0, 255]$.

Nous réalisons, pour le terrain, deux algorithmes fractals. Ces algrithmes génèrent des images en 256 niveaux de couleurs[4] ; ces couleurs représentent les altitudes de chaque point 2D interprété en 3D, cf. §3.1.1. Les algorithmes utilisés sont : L'algorithme du *plasma*, cf. §2.1, et le *Mouvement Brownien avec la loi de Gauss*, cf. §2.2. Nous pouvons voir sur la figure 1.1 le résultat obtenu avec l'algorithme

[3]Benoît Mandelbrot défini les fracrales et écrit [24] : «... des objets naturels très divers, dont beaucoup sont fort familiers, tels la Terre, le Ciel et l'Océan, sont étudiés à l'aide d'une large famille d'objets géométriques, jusqu'à présent jugés ésotériques et inutilisables, mais dont j'espère montrer tout au contraire qu'ils méritent, de par la simplicité, la diversité et l'étendue extraordinaire de leurs nouvelles applications, d'être bientôt intégrés à la géométrie élémentaire ... Pour les étudier, j'ai conçu, mis au point et largement utilisé une nouvelle géométrie de la nature. La notion qui lui sert de fil conducteur sera désignée par l'un des deux néologismes synonymes, *objet fractal* et *fractale*, termes que je viens de former, pour les besoins de ce livre, à partir de l'adjectif latin fractus, qui signifie *irrégulier* ou *brisé* ...»

[4]Généralement, nous représentons ses couleurs par un dégradé de gris ; les valeurs varient du noir au blanc et elles sont respectivement interprétées comme des points de basse et de haute altitude.

du plasma porté en trois dimensions.

Pour varier les types de terrains créés par l'environnement, nous créons aussi un générateur de labyrinthes, le détail de l'algorithme est donné dans la séction 2.3.

Pour les arbres, notre choix s'est porté sur l'écriture d'un interprète de *L-Systèmes*. Notre interface incorpore un module dédié à la création d'arbres à partir d'un fichier descriptif. Ce fichier contient les informations relatives à chaque type d'arbre créé dans le paysage. L'interprète gère les *D0L-Systèmes*, les *0L-Systèmes* déterministes. Les arbres sont produits par notre implémentation de la *Tortue*; elle incorpore une composante stochastique qui modifie sensiblement le résultat pour chaque génération d'arbres. Le détail de l'implémentation est donné dans la section 2.4. Nous pouvons voir sur la figure 1.2 le processus décrit pas à pas de la création d'un de nos arbres L-Systèmes.

FIG. 1.2 – Génération d'un arbre en *L-Systèmes*. Les différentes étapes du processus.

1.1.2 Le simulateur 3D temps réel

Notre base d'algorithmes fractals produit les données nécessaires à la construction d'un univers tridimensionnel. Nous réalisons à partir de ces données la mise en œuvre de notre simulateur 3D. Il gère simultanément plusieurs aéronefs de type hélicoptère. Nous en donnons une modélisation physique, cf. §3.2. Les commandes de chacun des hélicoptères sont accessibles pour l'utilisateur par l'intermédiare de l'interface clavier. Une librairie de fonctions est écrite afin de permettre l'accès à ces commandes par le biais d'un module de pilotage automatique.

La représentation 3D de l'hélicoptère est créée à partir des plusieurs fichiers descriptifs de type *A.S.E.*, *ASCII Scene Export* ; notre module d'importation d'objets tridimensionnels modélise le véhicule, calcule son enveloppe convexe et l'incorpore à l'environnement, voir figure 1.3.
Les librairies *OpenGL* sont choisies pour l'écriture du système de rendu. Le choix de cette librairie n'est pas anodin. Celle-ci permet un accès direct aux fonctions disponibles dans la carte graphique ; elle est disponible sur la majorité des systèmes d'exploitation et autorise donc la portabilité de notre application sur plusieurs plate formes[5].

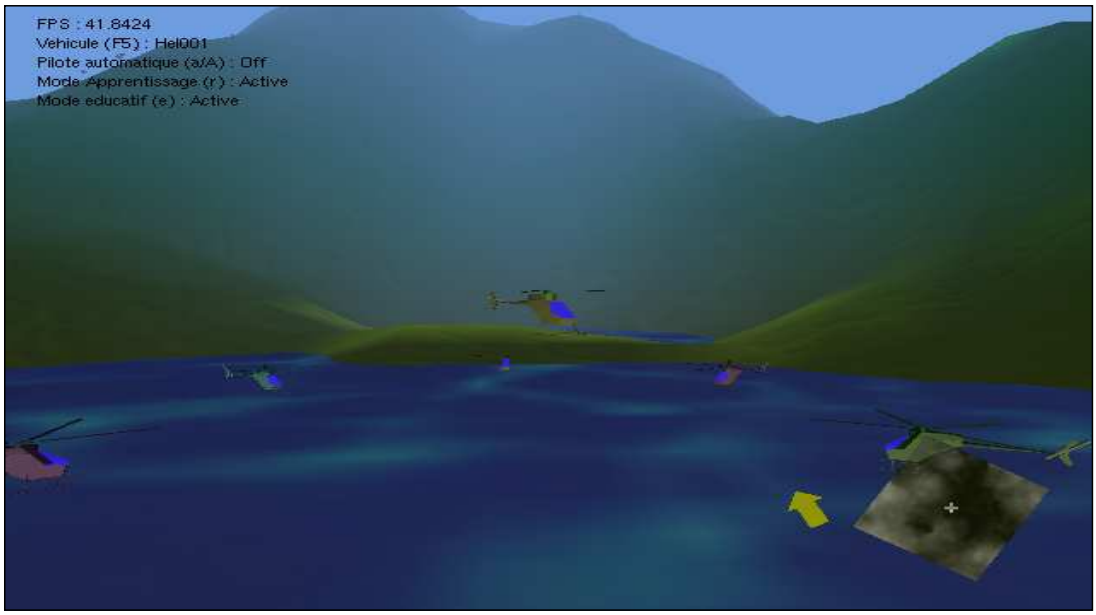


FIG. 1.3 – L'interface 3D temps réel de notre environnement.

[5]Nous avons testé l'interface sur plusieurs systèmes d'exploitations : *Linux-Redhat, Linux-Mandrake, FreeBSD, Sgi-Irix* et *Microsoft-Windows*.

1.2 Le vol du Perceptron

Nous réalisons un module de pilotage capable d'apprendre à voler en copiant les réactions d'un pilote humain ou d'un automate. L'apprentissage doit être réalisé en temps réel ; la fluidité du rendu tridimensionnel est primordiale pour la synchronisation des réactions du «*formateur*[6]» aux perceptions de l'apprenant. Nous choisissons d'implémenter un réseau de neurones de type *Perceptron Multi-Couches*, noté *P.M.C.*

1.2.1 Outils pour l'Intelligence Artificielle

Afin d'effectuer l'apprentissage du *P.M.C.*, nous ajoutons à notre librairie un ensemble de fonctions permettant d'émuler la perception de l'aéronef; cette perception est donnée sous la forme d'instruments de bord. Les instruments de chaque véhicule sont paramètrables.

Nous demandons au pilote de l'appareil de rejoindre à partir de sa position initiale le point d'arrivée défini par l'utilisateur. Un algorithme calcule le chemin minimisant les altitudes empruntées. Nous appelons le chemin résultant : le *plus sûr chemin*, cf. §4.2. Il est assimilé à une trajectoire en rase-mottes, «*radar-safe*[7]». Chaque hélicoptère est muni de capteurs de distance lui permettant de localiser les obstacles et pratiquer une trajectoire sans risque de collision.
Nous construisons à partir de ces outils un *automate volant* capable de suivre la trajectoire définie par le *plus sûr chemin*. Nous l'utilisons pour faciliter l'apprentissage du *Perceptron Multi-Couches*.

1.2.2 L'apprentissage

L'apprentissage réalisé pour le *P.M.C.*[8] est effectué en deux étapes. Nous pratiquons la correction des pondérations du réseau de neurones par rétropropagation du gradient, cf. §4.4.

Dans une première phase d'apprentissage, le *P.M.C.* observe les réactions produites par l'automate volant pour un ensemble de perceptions données[9]. À chaque instant, l'ensemble *Perception / Réaction* constitue l'échantillon pour lequel le réseau de neurones doit corriger ses poids. Cette phase se termine par la convergence

[6]Nous utilisons le terme *formateur* pour faire référence à l'entité qui dirige l'apprentissage, le professeur.

[7]La position de l'hélicoptère est indétectable par les radars.

[8]Notre *P.M.C.* est constitué d'une seule couche cachée ; ce choix s'est avéré nécessaire pour des contraintes de temps réel et suffisant au vu des résultats obtenus.

[9]Les deux pilotes (le *Perceptron Multi-Couches* et l'automate volant) lisent les données de perception des hélicoptères qui leur sont assignés. Dans cette phase, seul l'automate volant agit à la perception et donne les commandes correspondantes à la réaction voulue.

des réactions du *P.M.C.* vers un comportement similaire à celui de l'automate volant. Nous donnons et comparons les résultats de chacun des pilotes, cf. §4.4.2. La figure 1.4 montre les temps de convergence pour différentes configurations de *P.M.C.* à trois couches.

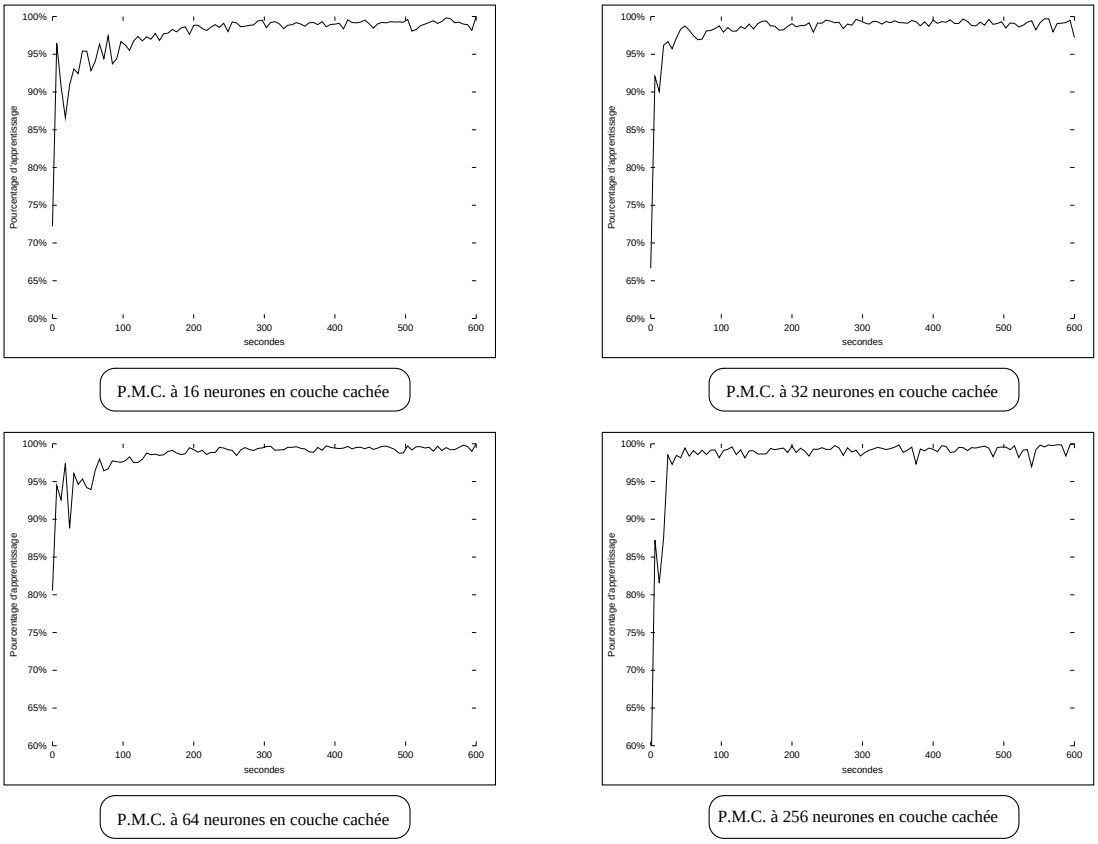


FIG. 1.4 – Courbe des pourcentages d'apprentissage du *Perceptron Multi-Couches*. Cette courbe est donnée pour un réseau à 16, 32, 64 et 256 neurones en couche cachée.

Pour finir, nous réalisons une seconde phase d'apprentissage à partir du réseau de neurones obtenu. Le *formateur* assigné à cette phase est un utilisateur relié à l'environnement par l'intermédiaire de l'interface clavier. L'utilisateur doit prendre plus de risques dans cette phase de vol. Il doit donner à l'hélicoptère une trajectoire de vol en rase-mottes afin d'affiner l'apprentissage du *P.M.C.*. Le comportement du *Perceptron Multi-Couches* est alors amélioré, sa distance au sol est réduite et sa vitesse est visiblement constante et soutanue. Nous donnons les résultats obtenus dans la section 4.4.2.

Chapitre 2

Éléments de paysages

Nous présentons dans ce chapitre les différentes méthodes utilisées pour la création des éléments nécessaires à la génération de paysages ; ces paysages constituent la base de notre environnement tridimensionnel.

Nous étudions les algorithmes employés dans la création de terrains. Les structures de données des terrains sont représentées sous la forme de matrices à deux dimensions contenant des informations sur les altitudes de chaque sommet ; ces matrices dressent les données topographiques des terrains.
Les algorithmes utilisés génèrent des images en 256 niveaux de gris. L'interprétation de ces images permet de construire une représentation 3D des terrains, cf. chapitre 3. La composante aléatoire de la génération des terrains est primordiale dans notre application, elle permet l'obtention d'une grande variété dans les paysages générés. Ainsi le comportement des pilotes automatiques est testé dans de multiples conditions.

Nous réalisons, pour la création des terrains, des générateurs de nuages fractals et de labyrinthes aléatoires. Les nuages fractals sont obtenus avec les algorithmes du *plasma* et de *Brown-Gauss*. Nous donnons pour les labyrinthes générés un algorithme de recherche du plus court chemin.

Nous poursuivons notre démarche par l'étude d'une méthode de génération d'arbres fractals. Un interprète de *L-Systèmes* est implémenté, celui-ci génère plusieurs types d'arbres, ces arbres sont aléatoirement placés dans le paysage et constituent des obstacles potentiels à la navigation des pilotes automatiques, cf. chapitre 3.

2.1 Le plasma

La méthode du plasma «*randomize plasma*» est un algorithme fractal [2, 32, 40] permettant d'obtenir une image représentant un mélange de couleurs vives,

une interprétation artistique d'une matière en fusion[1].
Les images obtenues avec cet algorithme, après leur transformation en niveaux de gris, sont utilisées comme cartes des altitudes pour le rendu de paysages. La composante aléatoire de cet algorithme nous garantit la diversité des terrains générés. Nous commençons l'étude de l'algorithme par une approche à une dimension[2]. Dans la description de l'algorithme, nous montrons pourquoi il peut servir à la génération de paysages montagneux, puis comment l'utiliser dans le cadre d'une approche à deux dimensions[3] afin d'obtenir des *nuages fractals*[4].

2.1.1 L'approche à une dimension

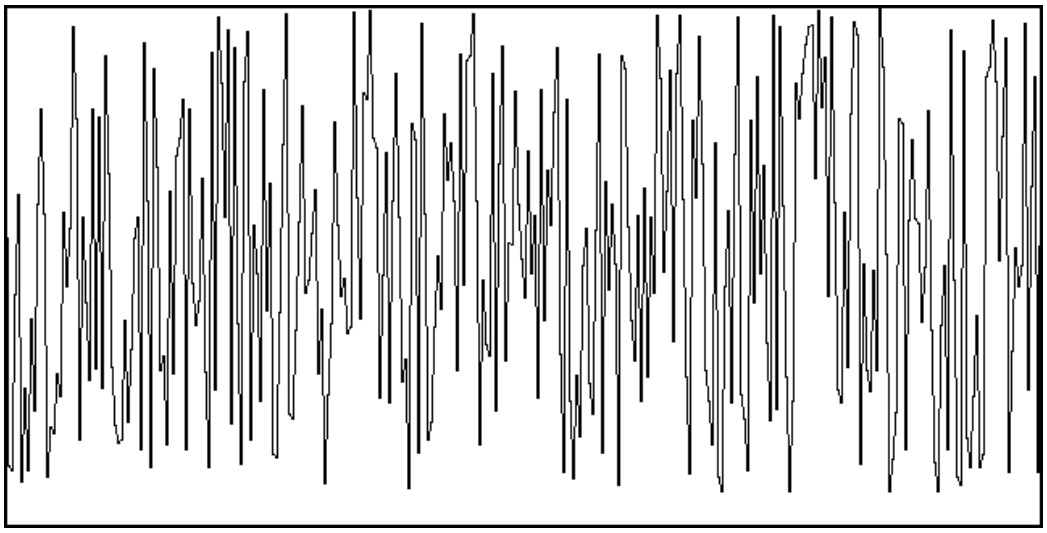


FIG. 2.1 – Courbe aléatoire générée par ordinateur.

Notre approche à consiste obtenir une suite de valeurs d'altitudes. Ces valeurs, représentées sur l'axe du temps, forment une courbe au relief montagneux ou encore une courbe de fluctuations boursières. L'aspect aléatoire des valeurs calculées par l'algorithme nous amène à étudier et à transformer les sorties des générateurs pseudo-aléatoires[5] afin d'obtenir le résultat attendu.

À partir d'une distribution équiprobable de nombres, nous construisons, dans

[1]Le plasma, en physique, est défini comme le quatrième état de la matière, état dans lequel toute matière est formée d'électrons libres, d'ions et de particules neutres, il est obtenu à partir d'une température supérieure à 7 000 °K ; le milieu reste macroscopiquement neutre du point de vu électrique.

[2]Cette méthode est basée sur l'interpolation linéaire.

[3]Cette méthode est basée sur l'interpolation bilinéaire.

[4]Nous utilisons ce mot pour désigner les cartes d'altitudes générées par un algorithme fractal.

[5]Terme utilisé pour les générateurs de nombres aléatoires sur ordinateur.

le temps, une courbe reliant[6] les valeurs aléatoires obtenues. Cette courbe constituée d'une suite de points indépendants est difficilement exploitable pour générer des paysages aléatoires. Le terrain résultant serait trop accidenté pour être réaliste, voir figure 2.1.
Nous ajoutons donc une contrainte sur les altitudes données par la suite de valeurs aléatoires. Un seuil de variation entre ces valeurs est fixé. Ce seuil dépend de l'intervalle de temps δt qui sépare deux valeurs consécutives[7].
Par conséquent, l'altitude au moment t dépend de la moyenne[8] de l'altitude à $(t-1)$ et de l'altitude à $(t+1)$.

Description de l'algorithme

Posons $Y(t)$, l'altitude d'un point donné sur la courbe au moment t.
Pour $t = t_0 = 0$ et $t = t_{max}$, où t_{max} est la limite fixée du temps, donnons les conditions initiales d'altitude, $Y(t)$. Ces valeurs sont prises aléatoirement.
À partir des valeurs $Y(t_0)$ et $Y(t_{max})$, calculons l'altitude du point intermédiaire :

$$Y\left(\frac{t_{max}}{2}\right) = \frac{Y(t_0) + Y(t_{max})}{2} + o(t_{max} - t_0)$$

où $o(t_{max} - t_0)$ est un nombre réel aléatoire proportionnel à $(t_{max} - t_0)$.

À l'étape suivante, nous calculons les altitudes aux instants $\frac{1}{4}t_{max}$, $\frac{3}{4}t_{max}$:

$$\begin{cases} Y(\frac{t_{max}}{4}) & = & \frac{Y(t_0)+Y(\frac{t_{max}}{2})}{2} & + & o(\frac{t_{max}}{2} - t_0) \\ Y(\frac{3t_{max}}{4}) & = & \frac{Y(\frac{t_{max}}{2})+Y(t_{max})}{2} & + & o(t_{max} - \frac{t_{max}}{2}) \end{cases}$$

Nous réitérons ce processus jusqu'à obtenir le niveau de détail souhaité. Cette méthode est appelée *interpolation linéaire par déplacement des milieux*, voir figure 2.2.

En généralisant la méthode, récursivement, pour chaque paire de points, nous obtenons toutes les altitudes à des intervalles réguliers dans le temps δt, voir figure 2.3.

[6] La liaison des différentes valeurs aléatoires est réalisée deux à deux par des segments.
[7] Ce type de variations est habituellement appelé «*Mouvement Brownien*» .
[8] À un facteur $o(\delta t)$ près.

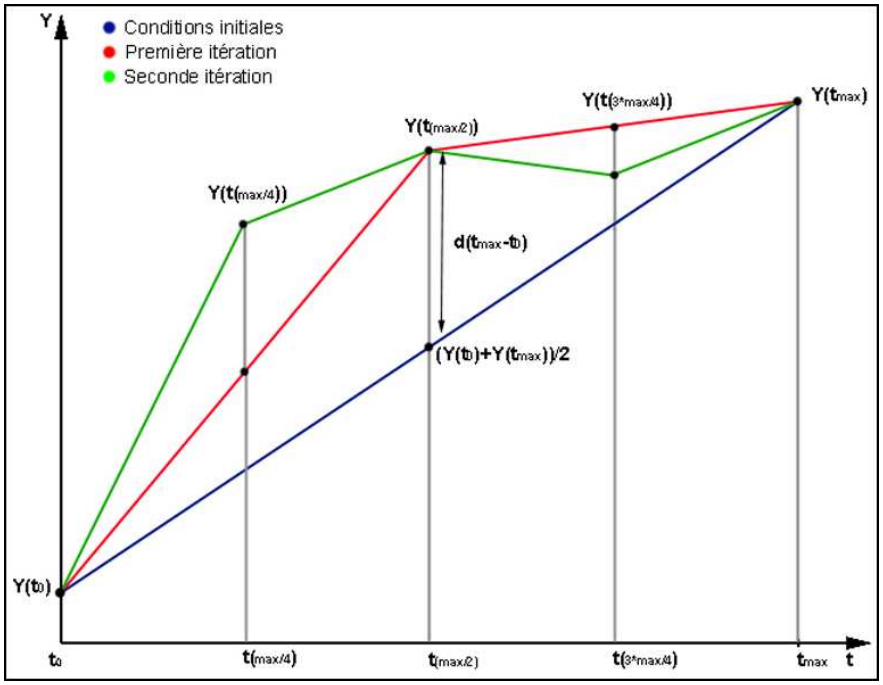


FIG. 2.2 – Le déplacement des milieux dans l'algorithme du plasma.

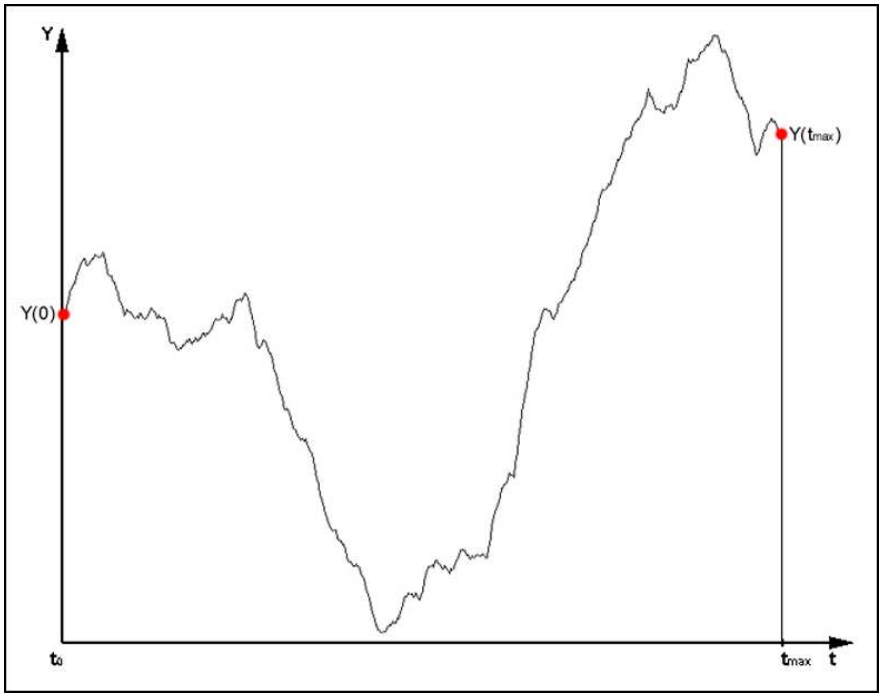


FIG. 2.3 – Le plasma à une dimension ; représentation bidimensionnelle sur l'axe du temps.

2.1.2 Le plasma à deux dimensions

Pour réaliser l'algorithme du plasma à deux dimensions, nous changeons la méthode d'interpolation pour le calcul des altitudes. Nous appliquons une *interpolation bilinéaire* dans un quadrillage de côté de longueur L.

Aux conditions initiales, nous avons quatre points situés aux coins de ce quadrillage[9]. À la première itération, nous divisons le quadrillage afin d'obtenir quatre sous régions identiques et cinq nouveaux points[10]. Pour chaque nouveau point dans le quadrillage, l'altitude est fonction, à un $o(L)$ près, des altitudes aux points dont il est le milieu.

Par exemple, pour un quadrillage dans un carré (A, B, C, D) de largeur L où les valeurs aux coins sont aléatoires :

- E $(L/2, 0)$: milieu des points A et B
  $\rightarrow \quad Y(E) = \frac{1}{2}(Y(A) + Y(B)) + o(L)\,;$
- F $(L, L/2)$: milieu des points B et C
  $\rightarrow \quad Y(F) = \frac{1}{2}(Y(B) + Y(C)) + o(L)\,;$
- G $(L/2, L)$: milieu des points C et D
  $\rightarrow \quad Y(G) = \frac{1}{2}(Y(C) + Y(D)) + o(L)\,;$
- H $(0, L/2)$: milieu des points D et A
  $\rightarrow \quad Y(H) = \frac{1}{2}(Y(D) + Y(A)) + o(L)\,;$
- I $(L/2, L/2)$: centre du carré (A, B, C, D)
  $\rightarrow \quad Y(I) = \frac{1}{4}(Y(A) + Y(B) + Y(C) + Y(D)) + o(L).$

où $o(L)$ est un nombre réel aléatoire proportionnel à L.

À l'étape suivante, le processus est réitéré récursivement sur chaque sous région du quadrillage pour que la surface, discrète, du quadrillage principal soit recouverte[11], voir figure 2.4.

Remarque : Si une altitude a été attribuée à un point, alors cette altitude ne sera plus modifiée, voir figure 2.4.

[9] Les valeurs initiales aux quatre points sont prises au hasard.

[10] Nous obtenons un point central et quatre points sur les milieux des bords

[11] À chaque nouvelle étape, notée N, les sous régions ont une largeur égale à $(L/2^N)$. L'algorithme prend fin avant que cette largeur ne soit inférieure ou égale à 1.

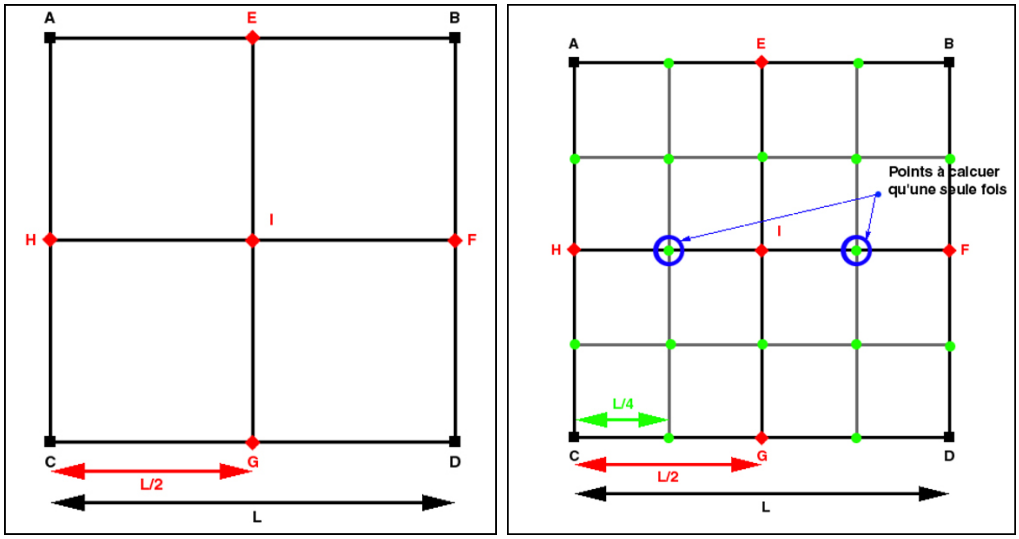


FIG. 2.4 – Interpolation bilinéaire dans l'algorithme du plasma.

2.1.3 Le nuage de plasma

Nous pouvons visualiser notre plasma sous différentes formes :

En niveaux de gris

Les niveaux de gris sont représentés par des valeurs entières comprises dans l'intervalle $[0, 255]$. Les altitudes aux différents points sont ramenées dans cet intervalle soit par modulo, soit par *min* et *max*, voir figure 2.5.

En couleur

- Pour obtenir un plasma par modèle de couleur, par exemple, RGB, CMYK, HLS, HSV [9]. Nous effectuons la même opération sur chacune des composantes du modèle de couleurs utilisé, voir figure 2.5 pour un plasma en RGB.
- Dans le cas du plasma en couleurs indexées, une palette de transition de couleurs est créée[12] (exemple : du bleu au vert, du vert au jaune et du jaune au rouge), et la même opération que dans le cas du plasma en niveaux de gris est réalisée. L'intervalle des valeurs possibles devient $[0, X-1]$ où X est le nombre de couleurs de la palette [2, 40].

[12]Usuellement nommée : *Color LUT*.

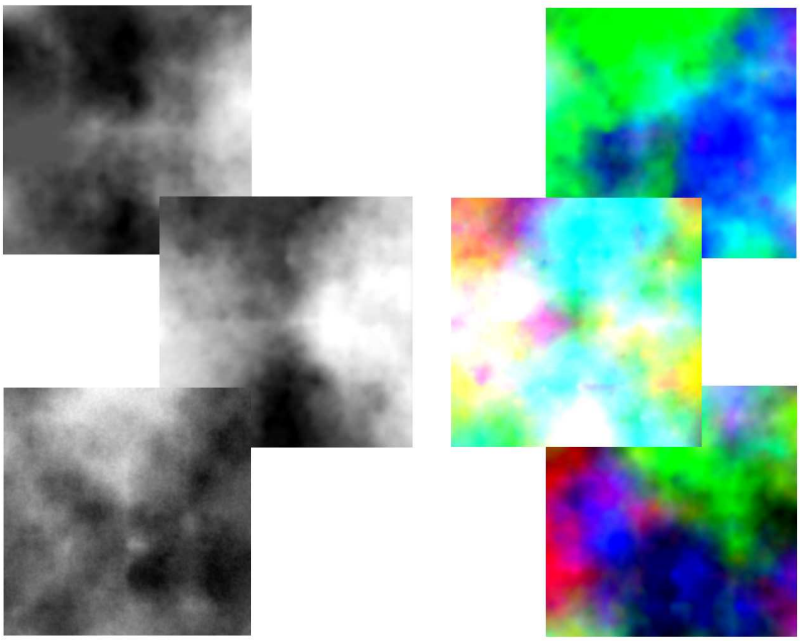


FIG. 2.5 – Rendu du plasma à deux dimensions.

Ainsi cet algorithme nous permet d'obtenir des nuages fractals de différents types[13]. Ces nuages sont utilisés comme cartes des altitudes pour le rendu de paysages montagneux en trois dimensions. En faisant varier l'échelle des valeurs aléatoires, nous obtenons des terrains allant du plus plat[14] au plus escarpé[15]. Ces terrains comportent plusieurs zones homogènes tout en restant globalement diversifiés, voir figure 2.5.
Afin d'obtenir d'autres types de terrains, nous introduisons une nouvelle méthode de génération de nuages fractals. Cette méthode est basée sur le *mouvement Brownien* et la *courbe de Gauss*.

[13]Nous obtenons un nuage différent pour chaque initiateur de chaîne pseudo-aléatoire (exemple : «*srand*» ou «*randomize*»).

[14]Pour des petites variations, nous obtenons un terrain homogène.

[15]Pour des grandes variations, nous obtenons un terrain rugueux, qui a la forme d'un paysage chinois.

2.2 Mouvement Brownien et loi de Gauss

Nous présentons une nouvelle méthode permettant de générer des paysages aléatoires à partir du *mouvement Brownien* [24] dont l'aspect aléatoire est régi par la *loi de Gauss*. L'introduction de la courbe de Gauss permet d'obtenir des courbes dans lesquelles le phénomène d'auto-similarité[16] est plus significatif.
Nous commençons par définir le mouvement Brownien, puis la courbe de Gauss. Il s'agira ensuite de reproduire une *distribution Gaussienne* sur ordinateur et enfin d'utiliser ces résultats pour générer un nuage *Brown-Gauss*[17].

2.2.1 Le mouvement Brownien

Le mouvement Brownien [18], ou sous forme simplifiée «*la marche de l'homme ivre*» , est le mouvement aléatoire d'une particule dans le temps. Cette particule, représentée dans un espace à une dimension (mouvement sur une droite), subit à chaque intervalle de temps δt une collision qui la déplace soit vers la gauche (-1) soit vers la droite ($+1$) par rapport à son ancienne position, ceci de façon équiprobable. Donc, sur un temps fini t_{max} , cette particule sera soumise à n collisions avec $t_{max} = n \times \delta t$ et elle aura effectué un déplacement total de $n \times |\delta X|$ où $\delta X = \pm 1$. Nous notons que la variation entre la position initiale de la particule ($t = 0$) et sa position finale ($t = t_{max}$) est en moyenne nulle.
Dans le cas général, pour $\delta X = \pm d$, avec $0 < d < 1$, D la distance quadratique moyenne parcourue est proportionnelle à $n \times d^2$, avec $n \times d = v \times t_{max}$, on a $n \times d^2 = v \times d \times t_{max}$, v étant la vitesse de déplacement de la particule, voir figure 2.6.

Cette distance est appelée la *variance* ou *l'écart quadratique moyen*. Elle est proportionnelle à t_{max}, d'où un *écart type* proportionnel à $\sqrt{t_{max}}$.
On dit alors que ce mouvement obéit à une *loi normale*, dite *loi de Gauss*, *d'espérance* 0 et d'écart type $\sqrt{t_{max}}$.

[16] L'auto-similarité [31] est la propriété principale du résultat obtenu par un algorithme fractal. On dit que l'aspect fractal est préservé.

[17] Ce nom est utilisé dans l'ensemble de l'étude pour désigner l'algorithme appliquant le mouvement Brownien et la courbe de Gauss à la génération d'images fractales.

[18] Le mouvement Brownien, ou le processus de Wiener, décrit un phénomène observé par le botaniste écossais Robert Brown en 1827. Des grains de pollen en suspension dans un liquide suivaient un mouvement rapide et désordonné, ce mouvement est entretenu par la disponibilité d'énergie fournie par des chocs moléculaires du liquide.
En 1905, Albert Einstein développa cette théorie par une approche de type mécanique statistique ; le mouvement est causé par des chocs atomiques ou moléculaires.
A partir de 1920, Norbert Wiener proposa une définition mathématique du phénomène :
- $B(0) = 0$.
- pour tout $0 \leq s \leq t$, $B(t) - B(s)$ est une variable gaussienne du type $N(0, t - s)$ indépendante de l'ensemble engendré par : $B(u)$ tel que $u \leq s$

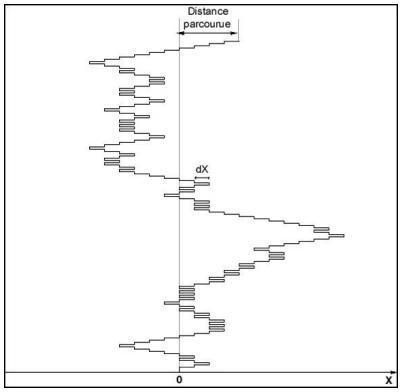


FIG. 2.6 – Mouvement Brownien à une dimension.

2.2.2 La loi normale ou la courbe de Laplace - Gauss

La loi normale[19] intervient dans l'étude de phénomènes aléatoires dont la répartition des valeurs s'étale autour de leur moyenne. Ces phénomènes aléatoires sont soumis à des événements qui agissent additivement et indépendamment.

Une distribution de valeurs aléatoires est dite *normale*, quand elle a l'allure d'une courbe exponentielle :

$$f(t) = \frac{1}{\sqrt{2\pi}} e^{-\frac{t^2}{2}}$$

La loi normale est une loi des aléas numériques continus, dont la densité de probabilité admet une représentation graphique dite «*courbe en cloche*» ou «*gaussienne*» , symétrique par rapport à la valeur moyenne, *l'espérance*, et dont la fonction de répartition est donnée par des tables[20]. Pourvu que la taille de l'échantillon observé soit assez grande, tout aléa défini sur un ensemble d'éléments peut être assimilé à un aléa gaussien.

Plus généralement, la loi normale, notée $N(m, \sigma)$, est caractérisée par son espérance mathématique m et son écart-type σ (ou sa variance σ^2 notée $\mathcal{V}$).
Elle est telle que :

- 68% des valeurs sont comprises dans l'intervalle $m \pm \sigma$;
- 95% des valeurs sont comprises dans l'intervalle $m \pm 2\sigma$ (plus exactement, $m \pm 1.96\sigma$) ;

[19]La recherche de la paternité de cette loi n'est pas simple. De Moivre l'avait énoncée en termes de probabilités approchées d'une distribution binomiale lorsque le nombre d'expériences est grand. Laplace et Gauss la construisent dans le cadre de l'évaluation des erreurs commises sur des mesures d'observations.

[20]Pour une espérance nulle et un écart type égal à 1.

– 99,7% des valeurs sont comprises dans l'intervalle $m \pm 3\sigma$.

La loi $N(0,1)$ est appelée *loi normale centrée réduite*, voir figure 2.7.

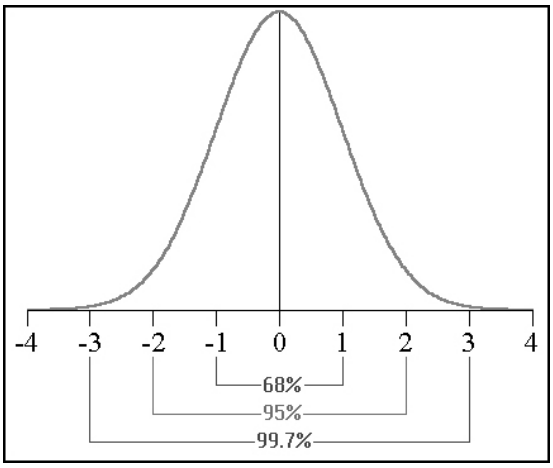


FIG. 2.7 – La loi normale centrée réduite

Pour une variable aléatoire X telle que[21] : $E(X) = m$ et $\sigma(X) = \sigma$. Nous obtenons une loi normale centrée réduite pour X' en posant : $X' = (X - m)/\sigma$.

2.2.3 Génération d'un bruit Gaussien sur ordinateur

Les ordinateurs génèrent une suite de nombres pseudo-aléatoires avec une distribution de probabilité qui est la même pour chaque nombre. Les entiers générés appartiennent à l'intervalle $[0, RAND\_MAX - 1]$ et chacun a une probabilité de $\frac{1}{RAND\_MAX}$.

À partir de cette distribution uniforme de nombres aléatoires, nous souhaitons obtenir une distribution qui suit la loi normale.

Le théorème de la limite centrée

Soit (X_n) une suite de n variables aléatoires indépendantes et obtenues de façon équiprobable (de même loi). On note pour $X_k (1 \leq k \leq n)$:

– $\sigma_k = \sigma$

– $E(X_k) = m$

En posant pour n la variable aléatoire Y_n telle que :

$$Y_n = \frac{\sum_{i=0}^{n} X_i - n \times m}{\sigma \times \sqrt{n}}$$

[21] L'espérance mathématique de X est notée $E(X)$.

D'après le théorème de la limite centrée, pour tout n, la suite (Y_n) converge en loi vers une variable aléatoire normale centrée réduite [2].
Par conséquent, à partir de la suite de n nombres aléatoires appartenant à l'intervalle $[0, A]$ avec $A = (RAND\_MAX - 1)$, pour chaque variable aléatoire X_k on obtient :

$$E(X_k) = m = \frac{A}{2}$$

par définition :

$$\mathcal{V}(X_k) = \sum_{i=0}^{A} (X_i - E(X_k))^2 \times Proba(X_k = X_i)$$

qui assimilée en intégrale donne :

$$\begin{aligned} \int_0^A (u - \tfrac{A}{2})^2 \times \tfrac{du}{A} = & \quad \tfrac{1}{A} \int_0^A (u^2 - A \times u + \tfrac{A^2}{4}) \times du \\ = & \quad \tfrac{1}{3A}[u^3]_0^A - \tfrac{1}{2}[u^2]_0^A + \tfrac{A}{4}[u]_0^A \\ = & \quad \tfrac{A^3}{3A} - \tfrac{A^2}{2} + \tfrac{A^2}{4} \\ = & \quad \tfrac{A^2}{12} \end{aligned}$$

on a donc :

$$\mathcal{V}(X_k) = \tfrac{A^2}{12}$$

$$\Rightarrow \sigma(X_i) = \sqrt{\mathcal{V}(X_k)} = \tfrac{A}{2\sqrt{3}}$$

La variable aléatoire Y_n devient, voir figure 2.8 :

$$Y_n = \frac{\Sigma_{i=0}^{A} X_i - \frac{n \times A}{2}}{\sqrt{\frac{n \times A}{2\sqrt{3}}}}$$

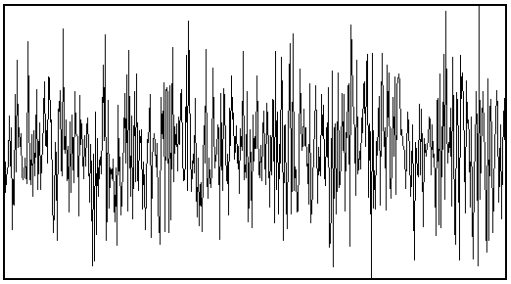


FIG. 2.8 – Bruit Gaussien (*White Noise*)

2.2.4 Utilisation de la loi normale dans la génération du paysage aléatoire

La partie traitée ici est similaire à celle abordée lors de la définition de la méthode de déplacement des milieux de l'algorithme du plasma. Ici, la variation, lors des déplacements des milieux, obéit à une distribution gaussienne, voir figure 2.9.
Pour accentuer l'effet d'auto-similarité, nous introduisons Δ, facteur multiplicatif de la valeur aléatoire ; Δ doit décroître dans le temps.

À chaque nouvelle étape n, la résolution est divisée de moitié, ainsi que la variance. Soit r $= \frac{1}{2}$:

$$\mathcal{V}_n = \mathcal{V}_0 \times (r^n)^{2H}$$
avec $0 < H < 1$ (dans notre cas $H = \frac{1}{2}$).

Ce qui donne l'écart type à l'étape n :

$$\sigma_n = \sigma_0 \times r^{nH}$$

Pour chaque étape supplémentaire, nous obtenons un écart-type multiplié par r^H par rapport à l'étape précédente. Soit avec $r = \frac{1}{2}$ et $H = \frac{1}{2}$, un facteur multiplicatif de $\frac{1}{\sqrt{2}}$.

$$\Rightarrow \Delta_n = \frac{\Delta_{n-1}}{\sqrt{2}}$$

Détail de l'algorithme

1. $Y(t_{debut} = t_0), Y(t_{fin} = t_{max})$ donnés ;
2. $\Delta = \sigma$;
3. $Gauss()$ est la fonction retournant un réel aléatoire dans l'intervalle $[-4, +4]$ suivant une distribution *Normale* centrée réduite ;
4. $Y(\frac{t_{debut}+t_{fin}}{2}) = \frac{Y(t_{debut})+Y(t_{fin})}{2} + \Delta \times Gauss()$;
5. $\Delta = \frac{\Delta}{\sqrt{2}}$;
6. Tant que $t_{debut} < t_{fin}$ faire :
   - $t_{debut} = t_{debut}$ et $t_{fin} = \frac{t_{debut}+t_{fin}}{2}$, aller à "4" ;
   - $t_{debut} = \frac{t_{debut}+t_{fin}}{2}$ et $t_{fin} = t_{fin}$, aller à "4".
7. Fin.

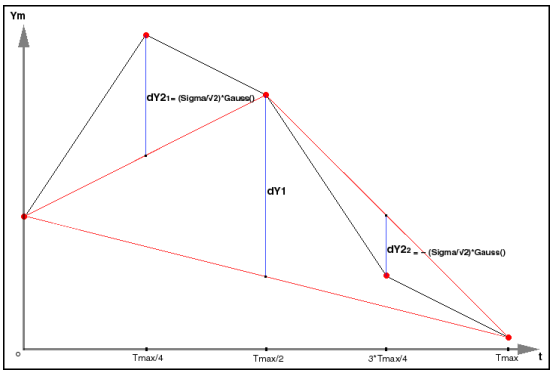
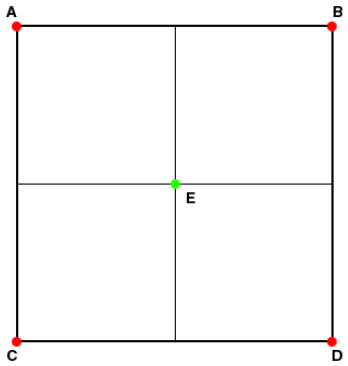


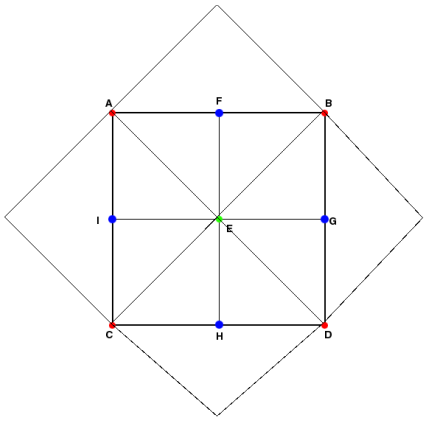
FIG. 2.9 – Brown-Gauss à une dimension ; représentation bidimensionnelle sur l'axe du temps.

Interpolation bilinéaire dans l'algorithme Brown-Gauss

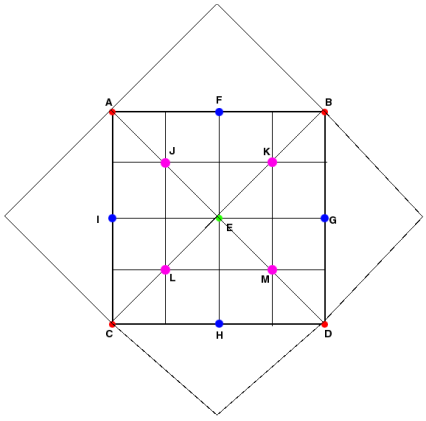
Pour deux dimensions, l'interpolation est détaillée sur le schéma suivant :



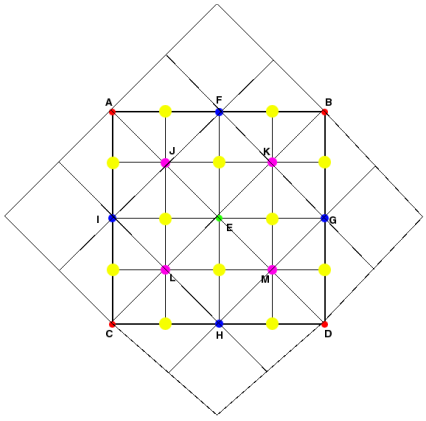
Étape 1 : $E \leftarrow (A+B+C+D)/4$



Étape 1bis :
$F \leftarrow (A+B+E)/3$, $G \leftarrow (B+D+E)/3$,
$H \leftarrow (C+D+E)/3$ et $I \leftarrow (A+C+E)/3$



Étape 2



Étape 2bis

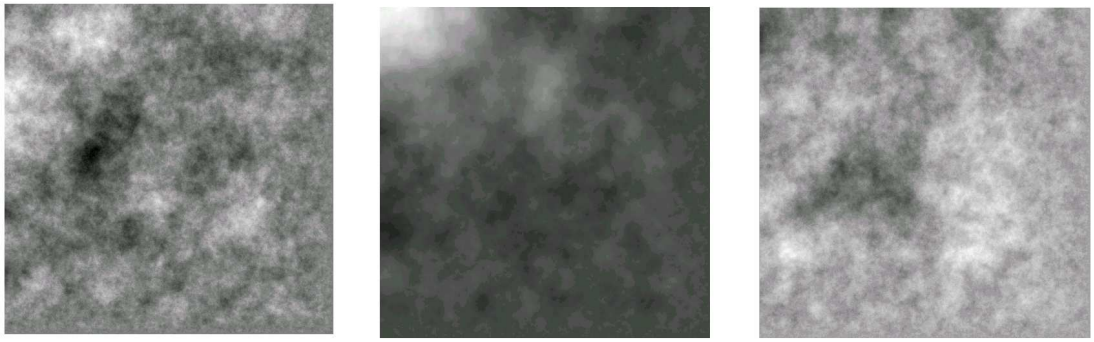


FIG. 2.10 – Nuages fractals générés par la méthode Brown-Gauss ; mouvement Brownien et courbe de Gauss.

Ce second algorithme génère des nuages fractals pour lesquels l'aspect d'autosimilarité est plus significatif, voir figure 2.10. À l'instar de l'algorithme du plasma, nous pouvons faire varier la dureté du terrain généré. Cette variation est donnée par le paramètre H de l'algorithme.

Nous poursuivons notre démarche de création de terrains par l'étude et l'implémentation de notre algorithme stochastique de création de labyrinthes. Nous donnons pour ces labyrithes une méthode de calcul du plus court chemin.

2.3 Les labyrinthes

Dans le but de pouvoir tester d'autres types de déplacements que celui d'un aéronef au dessus d'un paysage naturel (montagnes) ; nous avons décidé d'ajouter la possibilité de générer des labyrinthes. Nous imposons que les déplacements dans le labyrinthe se fassent dans un plan, à l'intérieur d'un réseau carré.
Les labyrinthes incorporés au simulateur sont au minimum 1-connexes[22].
À partir de ce type de labyrinthe[23], nous pouvons à tout moment augmenter la connexité en cassant des murs, et de ce fait créer d'autres chemins possibles[24].

2.3.1 Création d'un labyrinthe

L'algorithme employé pour la génération de labyrinthes utilise des propriétés qui s'apparentent au *Quick Union Find* [38] ; algorithme de recherche et de construction de graphes de connexité dans un ensemble d'éléments donné.

L'embryon du labyrinthe est tel que toutes les positions de départ (les nœuds) sont entourées de 8 murs. Il se présente sous la forme d'une matrice d'entiers de dimensions $(2N+1, 2M+1)$ où $N \times M$ est le nombre de cases non-murées. Dans chacune de ces cases, un identifiant unique est placé (de 0 à $N \times M - 1$). Nous posons la valeur -1 pour les cases murées.
À l'initialisation, nous obtenons la matrice L pour $N = M = 3$:

$$L = \begin{pmatrix} -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & \mathbf{0} & -1 & \mathbf{1} & -1 & \mathbf{2} & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & \mathbf{3} & -1 & \mathbf{4} & -1 & \mathbf{5} & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 \\ -1 & \mathbf{6} & -1 & \mathbf{7} & -1 & \mathbf{8} & -1 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{pmatrix}$$

Chacune des cases non-murées se trouve dans une des composantes d'un même graphe ; chaque composante est réduite à un seul nœud. Pour $N = M = 3$, un graphe à 9 composantes connexes est créé.
Afin d'obtenir un graphe à une composante connexe, nous sélectionnons aléatoirement un mur séparant deux composantes disjointes ; le mur est détruit et l'identifiant d'une des deux composantes est propagé sur les nœuds de l'autre. Nous choisissons de propager la valeur du plus petit identifiant.

[22] Pour tout point (emplacement non muré) dans ce labyrinthe, il existe un et un seul chemin le reliant à n'importe quel autre point.
[23] Assimilable à un graphe acyclique.
[24] Assimilable à un graphe cyclique.

Quand les $N \times M$ nœuds de départ prennent[25] la valeur[26] 0, alors toutes les composantes du graphe sont jointes ; l'unique graphe résultant est 1-connexe[27].

Détail de l'algorithme

1. Initialisation :
   - Une matrice $L[N][M]$ donnée ;
   - $k \leftarrow 0$;
   - Pour i de 0 à N faire :
     - Pour j de 0 à M faire :
       - Si (i impair et j impair) alors :
         - $L[i][j] \leftarrow k$;
         - $k = k + 1$;
       - Sinon : $L[i][j] \leftarrow -1$;
       - FIN "Si" ;
     - FIN "Pour" ;
   - FIN "Pour" ;
2. Pour $NbCasesAZero$, le nombre de cases à 0 : $NbCasesAZero = 1$;
3. Tant que $NbCasesAZero < N \times M$ faire :
   - Prendre au hasard (x, y) tel que :
     $L[x][y] = -1$ et (x impair ou y impair[28]) ;
   - Si x impair alors :
     - $d \leftarrow L[x][y-1] - L[x][y+1]$;
     - Si $d > 0$ alors :
       - $L[x][y] \leftarrow L[x][y+1]$;
       - Propager la valeur $L[x][y+1]$ à partir du point $(x, y-1)$ (en 4-connexité) pour les cases contenant la valeur $L[x][y-1]$[29] ;
     - Sinon, si $d < 0$ alors :
       - $L[x][y] \leftarrow L[x][y-1]$;
       - Propager la valeur $L[x][y-1]$ à partir du point $(x, y+1)$ (en 4-connexité) pour les cases contenant la valeur $L[x][y+1]$[29] ;
   - Sinon (y impair) :
     - $d \leftarrow L[x-1][y] - L[x+1][y]$;
     - Si $d > 0$ alors :
       - $L[x][y] \leftarrow L[x+1][y]$;
       - Propager la valeur $L[x+1][y]$ à partir du point $(x-1, y)$ (en 4-connexité) pour les cases contenant la valeur $L[x-1][y]$[29] ;
     - Sinon, si $d < 0$ alors :
       - $L[x][y] \leftarrow L[x-1][y]$;
       - Propager la valeur $L[x-1][y]$ à partir du point $(x+1, y)$ (en 4-connexité) pour les cases contenant la valeur $L[x+1][y]$[29] ;
4. FIN "Tant que".

[25] Ou dont la valeur était déjà à 0.

[26] Dans le cadre d'une représentation matricielle, pour un nœud du graphe, la valeur contenue dans une case à l'initialisation doit être différente de -1.

[27] Graphe à une composante 1-connexe.

[28] Si x et y sont tous les deux impairs, alors $L[x][y] \neq -1$

[29] Pendant la propagation, incrémenter $NbCasesAZero$ s'il y a lieu (comme décrit plus haut).

Exemple de propagation et résultat

+	+	+	+	+	+	+			+	+	+	+	+	+	+			
+	0	+	**1**	**1**	**1**	+			+	0	+	**0**	0	**0**	+			
+	0	+	+	+	**1**	+			+	0	+	+	+	0	+			
+	0	0	0	+	**1**	+	→		+	0	0	0	0	**0**	+			
+	+	0	+	+	+	+			+	+	0	+	+	+	+			
+	0	0	0	0	0	+			+	0	0	0	0	0	+			
+	+	+	+	+	+	+			+	+	+	+	+	+	+			

$$NbCasesAZero \leftarrow NbCasesAZero + 3$$

Pour $N = M = 200$ (une matrice de 401×401) nous obtenons[30] le labyrinthe figure 2.11 :

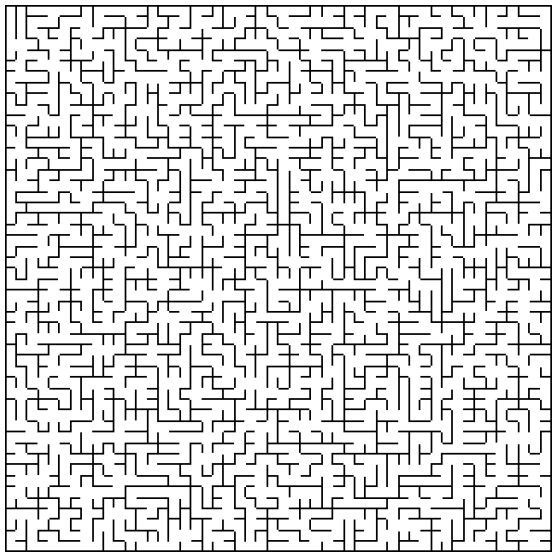


FIG. 2.11 – Labyrinthe ($N = M = 200$) généré aléatoirement.

2.3.2 Le plus court chemin dans un labyrinthe

Nous allons décrire l'algorithme utilisé pour calculer les plus courts chemins dans un labyrinthe ; c'est à partir de cet algorithme que nous avons eu l'idée de construire une méthode rapide et précise de calcul du *Plus sûr chemin*[31] dans le cas d'un paysage montagneux.

La méthode est basée sur l'algorithme du même nom introduit par E.W. Dijkstra en 1959 [6, 21]. Il s'agit de trouver le plus court chemin dans un graphe orienté

[30] Temps de calcul sur *AMD Athlon^{tm} XP 1700+, 1 Go de RAM* : 0.22 seconde.
[31] Un chemin minimisant les altitudes.

$G = (S, A)$. Dans ce graphe, les sommets sont reliés entre eux par des arêtes, chacune est notée a_i et possède un poids p_i. Le coût de parcours du chemin $C = \{a_1, a_2, \cdots, a_n\}$ équivaut à la somme des poids des arêtes qui le constituent.

Dans le cas du labyrinthe, et à l'inverse du *Plus sûr chemin*, tous les p_i valent 1[32]. Nous utilisons la matrice du labyrinthe pour stocker les informations liées au coût de déplacement, voir figure 2.12.

Détail de l'algorithme

$Pcc(x_1, y_1, x_2, y_2, v)$
Début :
    $v \leftarrow v + 1$;
    $L[x_1][y_1] \leftarrow v$;
    Si $x_1 = x_2$ et $y_1 = y_2$ alors :
        retour ;
    Pour chaque $d(x_{dir}, y_{dir})$ pris parmi les quatre directions possibles :
        Si $v < L[x_1 + x_{dir}][y_1 + y_{dir}]$ ou $L[x_1 + x_{dir}][y_1 + y_{dir}] = 0$ alors :
            $Pcc(x_1 + x_{dir}, y_1 + y_{dir}, x_2, y_2, v)$;
Fin.

1. Initialiser le labyrinthe[33] $M \times N$, cf. §2.3.1 ;
2. Pour un point de départ (x_d, y_d) et un point d'arrivée (x_f, y_f) faire : $Pcc(x_d, y_d, x_f, y_f, 0)$;
3. Partir du point d'arrivée et reconstruire le chemin (pour un point à valeur v, prendre le voisin qui a pour valeur $v - 1$) jusqu'au point de départ ;

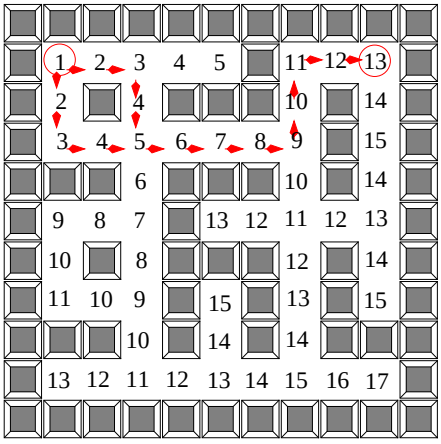


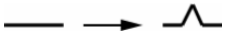
FIG. 2.12 – Schéma de résolution du plus court chemin dans un labyrinthe.

[32] Graphe non pondéré
[33] Nous obtenons une matrice L / $\forall a_{i,j} \in L$; $a_{i,j} \in \{-1, 0\}$.

2.4 Les L-Systèmes

Pour enrichir la liste des éléments composant notre générateur de paysages, nous avons décidé d'y incorporer des arbres. Afin d'obtenir une large variété d'arbres et pour rester dans le domaine des fractales et de la génération aléatoire, notre choix s'est porté sur un interpréteur de *L-Systèmes*.

L-Systèmes pour Lindenmayer-Systèmes est une théorie mathématique pour la modélisation des plantes[34] développée par le biologiste hongrois Aristid Lindenmayer en 1968 [34, 35, 19, 23, 43]. Le concept central à la notion de L-Systèmes est la création d'objets complexes par remplacements successifs de parties d'un objet initial (simple) en utilisant des règles de réécriture ou de production.
Un exemple significatif est la courbe "Flocon de neige" de Von Koch 1905 [3, 14, 40], redéfinie par B. Mandelbrot [24, 14] comme :

- Un objet initial (Initiateur) :  ;
- Un générateur ou règle de production :  .

Ce qui donne la figure 2.13.

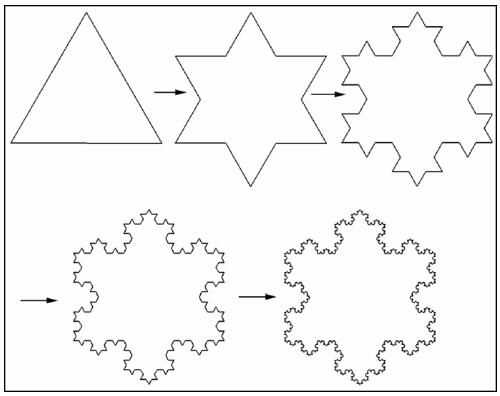


FIG. 2.13 – Processus de génération du flocon de neige de Von Koch.

La grammaire générative

Les avancées en matière de théorie des langages formels (la grammaire formelle de Chomsky) ont fait émerger l'idée d'utiliser les chaînes de caractères dans la description d'images. En 1968, Aristid Lindenmayer, introduit un nouveau type de mécanisme de réécriture de chaînes basé sur la substitution *parallèle* de caractères, plus tard appelé L-Systèmes.

[34] Elle sert essentiellement dans le domaine du graphisme généré par ordinateur ; génération de fractales, reproduction réaliste de plantes et, plus exotique, algorithmes graphiques pour compositions musicales.

2.4.1 L'interpréteur L-Systèmes

Afin de pouvoir générer des arbres en L-Systèmes et les intégrer au paysage, nous définissons les spécificités d'un interprète du langage. Cet interprète créé, à partir d'un fichier source décrivant les caractéristiques des arbres, les images fractales correspondantes. Avant de spécifier l'interprète, nous donnons les définitions des types de L-Systèmes utilisés.

D0L-Systèmes

C'est la plus simple classe de L-Systèmes ; définie comme 0L-Système, elle est le triplet ordonné $\{V, \omega, P\}$ tel que [34, 35, 30, 23] :

- V alphabet tel que :
  - V^* tous les mots générés à partir de V ;
  - V^+ tous les mots non vides générés à partir de V.
- $\omega \in V^+$ appelé axiome ;
- $P \subset V \times V^*$ ensemble fini de fonctions de règles de production tel que :
  - Si (a, X) est une production, on note : $a \to X$ où a est le prédécesseur et X est le successeur de la production ;
  - $\forall a \in V$, $\exists X \in V^* / a \to X$;
  - Si aucune règle de production n'est définie pour une lettre, on appliquera la production identité : $a \to a$.

Un 0L-Systèmes est *Déterministe*, noté D0L-Systèmes, si et seulement si : $\forall a \in V$, $\exists! X \in V^* / a \to X$.
Et pour chaque :

- $M = a_1 \cdots a_m \in V$
- $N = X_1 \cdots X_m \in V^*$

N est dérivé (ou généré) par M et on écrit $M \Rightarrow V$ si et seulement si : $\forall i \in \{1, \cdots, m\}, a_i \to X_i$.
N est généré par M en n dérivations s'il existe une suite de séquences de mots $S_0, \cdots, S_n$ telle que $S_0 = M$, $S_n = N$ et :

$$S_0 \underbrace{\Rightarrow \cdots \Rightarrow}_{n} S_n$$

Interprète D0L-Systèmes

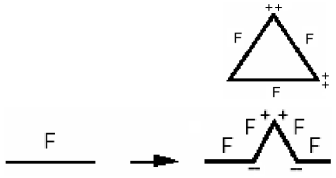
L'un des interprètes D0L-Systèmes les plus connus est la *tortue*, outil graphique principale du langage LOGO créé en 1968 [34, 35, 21, 14, 4, 30, 23, 36, 41, 40]. Les spécificités de l'outil nous servent de base dans notre interprète.

On le défini par :

- Un triplet (x, y, α) où (x, y) est la position de la tortue dans le plan, et α son angle d'orientation ;
- En donnant un pas de déplacement d et une incrémentation d'angle σ, la tortue répond aux commandes représentées par les symboles suivants :
  - F : (*Forward*) Avancer d'un pas d ; l'état de la tortue passe à (x', y', α) où $x' = x + d \times \cos(\alpha)$ et $y' = y + d \times \sin(\alpha)$. Alors un segment de droite est tracé entre les points (x, y) et (x', y') ;
  - f : Même chose que F sans tracer la droite.
  - $+$: tourner à gauche d'un angle σ. La tortue passe à l'état $(x, y, \alpha + \sigma)$;
  - $-$: tourner à droite d'un angle σ. La tortue passe à l'état $(x, y, \alpha - \sigma)$.

On obtient donc pour la courbe du flocon de neige les paramètres suivants :

- (x_0, y_0) au centre de l'écran, $\alpha_0 = 0$;
- $\sigma = \frac{\pi}{3}$ et $d > 1$;
- $\omega = F + +F + +F$;
- $F \to F - F + +F - F$.



Sur la figure 2.13 les 5 premières applications (itérations) de la règle de production sont représentées.

«Bracketed» L-Systèmes

Pour dessiner facilement des structures arborescentes, nous introduisons une amélioration des D0L-Systèmes, "Les *Bracketed*[35] L-Systèmes" permettant de faire du *backtracking* sur les branches [14]. Deux nouveaux symboles de langage sont ajoutés :

- [: *Push* : Empiler l'état actuel de la tortue.
-] : *Pop* : Dépiler l'état de la tortue afin de revenir à l'état précédent.

La figure 2.14 montre l'intérêt, l'auto-similarité [31], de cette classe de L-Systèmes sur l'exemple pris dans [30] :

- $(x_0, y_0, \alpha_0) = (centre_x, 0, \frac{\pi}{2})$;
- $\sigma = \frac{\pi}{3}$ et $d > 1$;
- $\omega = F$;
- $F \to F[+F]F[-F]F$.

[35] **Bracket :** Mettre entre parenthèses ; (*in square brackets*) mettre entre crochets.

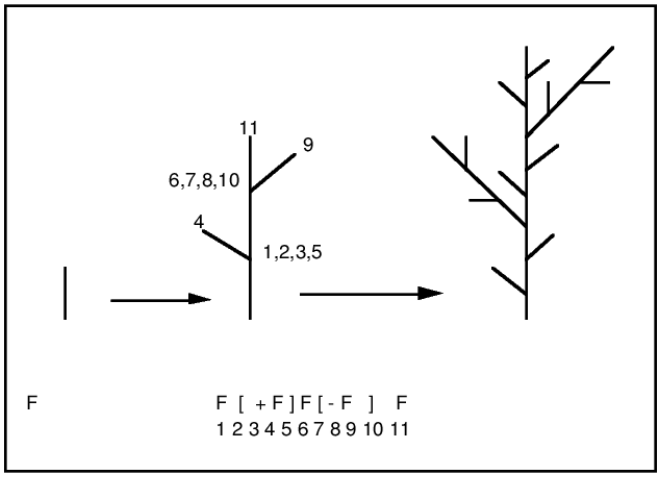


FIG. 2.14 – Bracketed L-Systèmes.

Améliorations de l'interprète et résultats

Pour enrichir la base d'arbres pouvant être générés par notre interprète, nous avons introduit une liste de nouveaux symboles ; ces symboles affectent les paramètres de la tortue au niveau de la pile dans laquelle elle se trouve. Chaque dépilement réinitialise l'état de la tortue aux paramètres connus avant création de la pile correspondante.

Les symboles sont :

- @X : multiplier d, le pas de déplacement de la tortue, par le réel X ;
- @iX : multiplier d, le pas de déplacement de la tortue, par le réel $\frac{1}{X}$;
- @qX : multiplier d, le pas de déplacement de la tortue, par le réel $\sqrt{X}$;
- ! : inverser les commandes de rotation ;
- | : effectuer une rotation de 180 degrés ;
- random : effectuer des variations aléatoires sur l'angle de la tortue ; ces variations sont proportionnelles à σ.

Exemple du langage[36] :

```
1. Arbre001{                                  // Nom de l'objet
axiom ++++++++++++++X                         // L'axiome ω
angle 7.2                                     // La variation d'angle σ
angle0 85.0                                   // L'angle initial α0
step 50.0                                     // Le pas de déplacement d en pixels
gen 9                                         // Le nombre de générations
X=F[@.5++++++++++X]-F[@.4————!X]@.6X          // Une règle de production
}
```

[36] L'exemple suivant représente l'arbre centre/bas de la figure 2.15.

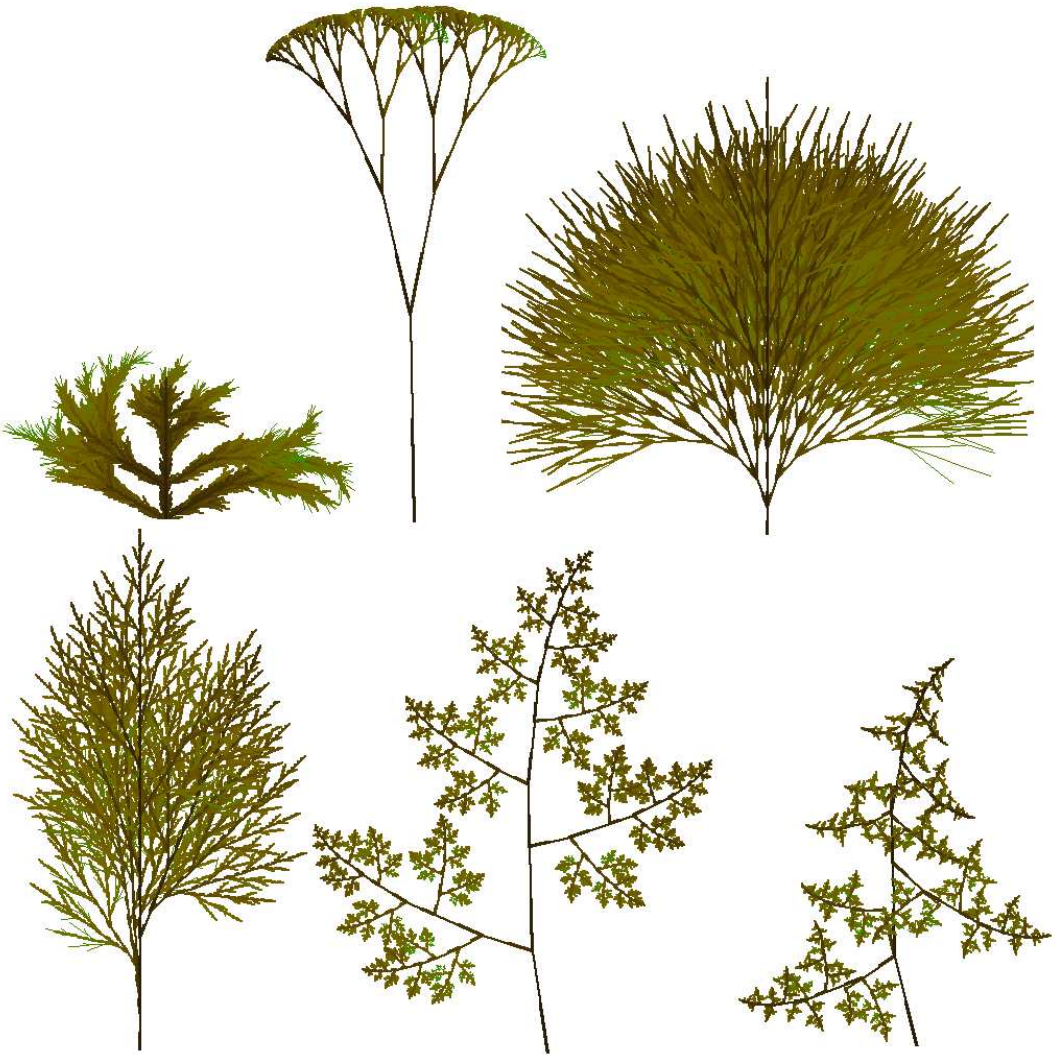


FIG. 2.15 – Arbres générés par l'interprète L-Systèmes.

Chapitre 3

L'univers tridimensionnel

Nous décrivons dans ce chapitre les opérations effectuées pour l'obtention d'une représentation tridimensionnelle des paysages. L'ensemble de notre univers virtuel est composé d'un terrain, d'arbres et de véhicules qui interagissent avec leur environnement directe.
Nous présentons les techniques utilisées dans le traitement des données topographiques. Les nuages fractals sont lissés puis portés en trois dimensions sous la forme d'un maillage régulier. Nous décrivons les méthodes employées pour l'habillage et l'affichage des terrains et des arbres. Ces méthodes permettent un rendu en temps réel des paysages générés.
Enfin, nous montrons la procédure utilisée à l'importation des objets extérieurs aux paysages. Nous importons les fichiers descriptifs du véhicule, l'hélicoptère, et nous donnons les caractéristiques de l'interaction du véhicule avec son environnement.

3.1 Les paysages 3D

Nous construisons ici l'interface graphique de notre générateur de paysages tridimensionnels. Nous voulons obtenir un rendu temps réel en image de synthèse des terrains et des arbres créés, cf. chapitre 2. Notre gestion des éléments tridimensionnels en listes d'affichage produit un rendu continu et rapide des parties visibles du paysage. Nous utilisons les librairies *OpenGL* afin d'accroître la rapidité[1] de notre application.

[1] L'accès directe aux fonctions de la carte graphique améliore considérablement les temps de calcul pour le rendu 3D.

3.1.1 Les données topographiques

Nous réalisons les opérations nécessaires à l'affichage des cartes d'altitudes[2] dans notre univers 3D. Nous obtenons une représentation tridimensionnelle du terrain. Le terrain est construit par un maillage régulier, nous l'affichons avec un rendu en mode *fil de fer* [25].

Interprétation des cartes d'altitudes

Pour passer d'une représentation matricielle[3] du terrain à une représentation 3D nous effectuons un maillage régulier sur les données et leurs emplacement dans la matrice. Les éléments de la matrice sont des entiers compris dans l'intervalle $[0, 255]$. Pour chaque élément (i, j) de la matrice $\mathcal{A}_{M,N}$, son abscisse "i", son ordonnée "j" et sa valeur "$a_{i,j}$" représentent respectivement l'emplacement horizontal "x", la profondeur "z" et l'altitude "y" de l'élément dans un espace 3D. L'élément (i, j) une fois décrit dans l'espace tridimensionnel, est appelé sommet et noté $S_{i,j}$. Nous ordonnons ces sommets par voisinage[4], de façon à obtenir une liste de *triangles*[5].

Exemple :

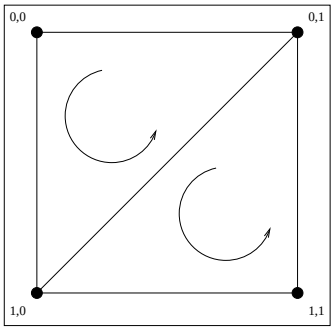
$$(S_{0,0}S_{1,0}S_{0,1}), (S_{0,1}S_{1,0}S_{1,1}), (S_{1,0}S_{2,0}S_{1,1}), \cdots, (S_{0,1}S_{1,1}S_{0,2}), (S_{0,2}S_{1,1}S_{1,2}), \cdots$$

Nous réalisons, à partir de cette liste de sommets, un maillage régulier suivant la méthode «*TRIANGLE-STRIP*» décrite dans «*OpenGL, version 1.2*» [25].

[2]Les cartes d'altitudes sont des cartes fractales générées par la méthode du plasma ou Brown-Gauss ; labyrinthes ; fichiers représentant des données topographiques quelconques.

[3]Matrices deux dimensions $M \times N$ à valeurs entières.

[4]Dans un triangle, l'orientation est donnée par l'ordre des sommets qui le constituent. Cet ordre doit être préservé afin d'obtenir des triangles dont la face avant est orientée vers le haut. Le schéma suivant montre les sens de l'orientation pour les deux premiers triangles de la matrice :



[5]Dans notre espace tridimensionnel, un triangle ou *polygone* à trois sommets, est l'enveloppe convexe constituée de segments de droites reliant trois sommets deux à deux.

Nous pouvons voir sur la figure 3.1 le résultat obtenu à partir d'un nuage de plasma[6] (20×20).

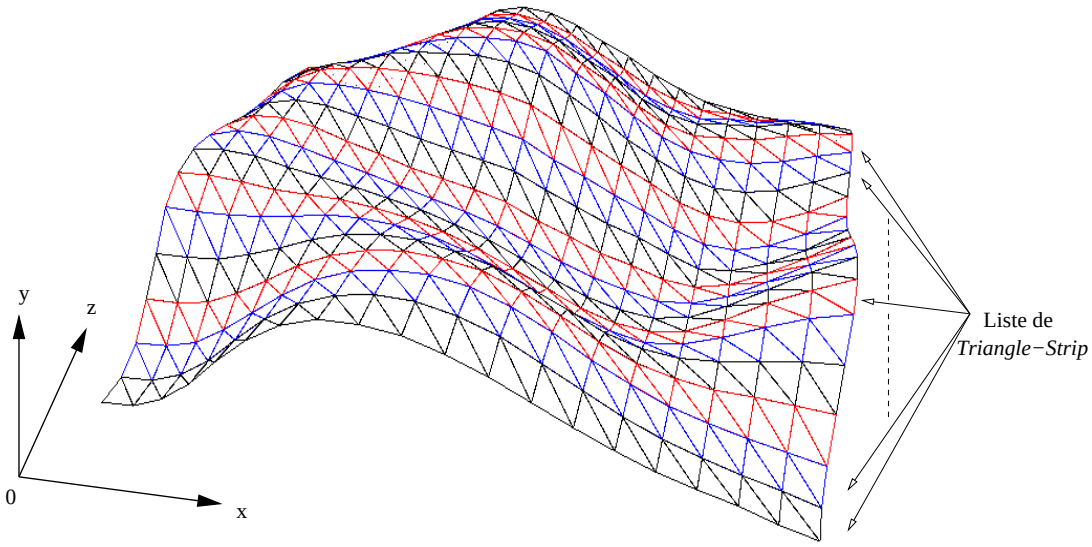


FIG. 3.1 – Maillage régulier de type *Triangle Strip*.

Le lissage

Le rendu tridimensionnel d'un paysage, créé à partir d'un nuage fractal, présente quelques défauts. Ces défauts sont dûs à des variations brutales d'altitude et donnent une impression de discontinuité[7]. Nous réalisons alors un lissage sur les données topographiques avant d'effectuer leur rendu.
Notre lissage est basé sur *l'équation de la diffusion de la chaleur*[8]. Cette équa-

[6] Le nuage de plasma est une image fractale obtenue avec l' algorithme du plasma (cf. §2.1.3).

[7] Ces défauts sont plus prononcés sur les nuages fractals générés par l'algorithme du plasma. En effet, dans certaines conditions, nous obtenons des points de discontinuité aux bords appartenant à plusieurs sous-régions (la subdivision récursive du quadrillage, cf. §2.1.2 et figure 2.4). Le résultat de l'algorithme Brown-Gauss est, par définition, lissé. Le filtre gaussien [12] «*Gaussian Blur*» y est sous-jacent ; il suffit de modifier le paramètre H de l'algorithme (cf. §2.2.4) pour varier l'intensité du lissage.

[8] Cette équation est donnée par : $\frac{\delta T}{\delta t} = k \times \left(\frac{\delta^2 T}{\delta x^2} + \frac{\delta^2 T}{\delta y^2} \right)$ pour $T_{(x,y)}$ la température au point (x, y), t le temps et k une constante liée à la matière conductrice.

tion est discrétisée[9] et appliquée à un voisinage 4-connexe. Pour la matrice $\mathcal{A}_{M,N}$ contenant les altitudes à lisser, nous réalisons la transformation suivante :

$$a_{i,j} \leftarrow \frac{a_{i+1,j} + a_{i-1,j} + a_{i,j+1} + a_{i,j-1} - 4 \times a_{i,j}}{4}$$

Le résultat de ce lissage, réalisé sur un nuage de plasma (75×75), est visible sur la figure 3.2.

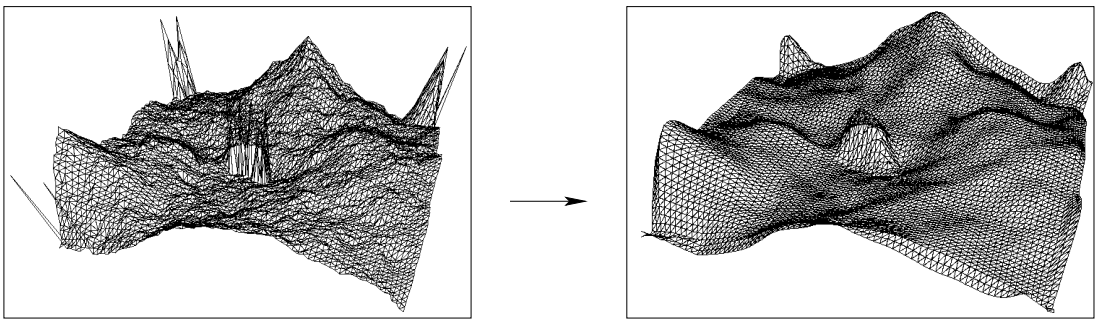


FIG. 3.2 – Rendu tridimensionnel d'un nuage de plasma avant et après lissage.

Gestion des listes d'affichage

Nous décrivons ici la méthode utilisée lors du rendu 3D. Nous souhaitons obtenir un affichage *temps réel* de la partie du terrain[10] dans laquelle se trouve la *caméra*[11] [25]. Pour préserver la charge du processeur (CPU), le terrain ne doit pas être affiché dans sa totalité. Nous sélectionnons la zone comprise dans le *champs de vision* de la caméra[12]. Cette zone est découpée en sous-parties et chaque sous-partie est stockée dans une *liste d'affichage*[13]. De cette façon nous

[9] La discrétisation est réalisée par l'approximation des dérivées partielles en x, y :

$$\begin{array}{lll|lll}
\frac{\delta T_{(x,y)}}{\delta x} & \approx \frac{\Delta T_{(x,y)}}{\Delta x} & = \frac{T_{(x+1,y)} - T_{(x,y)}}{\Delta x} & \frac{\delta T_{(x,y)}}{\delta y} & \approx \frac{\Delta T_{(x,y)}}{\Delta y} & = \frac{T_{(x,y+1)} - T_{(x,y)}}{\Delta y} \\
 & & = \frac{T_{(x,y)} - T_{(x-1,y)}}{\Delta x} & & & = \frac{T_{(x,y)} - T_{(x,y-1)}}{\Delta y} \\
 & & = \frac{T_{(x+1,y)} - T_{(x-1,y)}}{2 \times \Delta x} & & & = \frac{T_{(x,y+1)} - T_{(x,y-1)}}{2 \times \Delta y} \\
\frac{\delta^2 T_{(x,y)}}{\delta x^2} & \approx \frac{1}{\Delta x}\left(\frac{T_{(x+1,y)} - T_{(x,y)}}{\Delta x} - \frac{T_{(x,y)} - T_{(x-1,y)}}{\Delta x}\right) & & \frac{\delta^2 T_{(x,y)}}{\delta y^2} & \approx \frac{1}{\Delta y}\left(\frac{T_{(x,y+1)} - T_{(x,y)}}{\Delta y} - \frac{T_{(x,y)} - T_{(x,y-1)}}{\Delta y}\right) & \\
 & \approx \frac{T_{(x+1,y)} + T_{(x-1,y)} - 2 \times T_{(x,y)}}{\Delta x^2} & & & \approx \frac{T_{(x,y+1)} + T_{(x,y-1)} - 2 \times T_{(x,y)}}{\Delta y^2} &
\end{array}$$

[10] Dans notre environnement, un terrain a en moyenne, une taille de 1024×1024. Ce qui fait, en terme de polygones, 2×1024^2 triangles.

[11] La caméra est le plan de projection de l'univers virtuel sur écran. Elle est généralement positionnée derrière le *véhicule* sélectionné par l'utilisateur (cf. §3.2.1).

[12] Zone carrée de largeur l donnée. Cette zone se déplace pendant le mouvement du véhicule pour que la caméra soit toujours placée au centre.

[13] La liste d'affichage est l'objet *OpenGL* constitué d'une liste de sommets décrivant un ensemble de polygones.

pouvons charger les listes dont les sommets sont inclus dans le champ de vision, et inversement supprimer les autres. Nous obtenons un affichage partiel dynamique du terrain, voir figure 3.3.

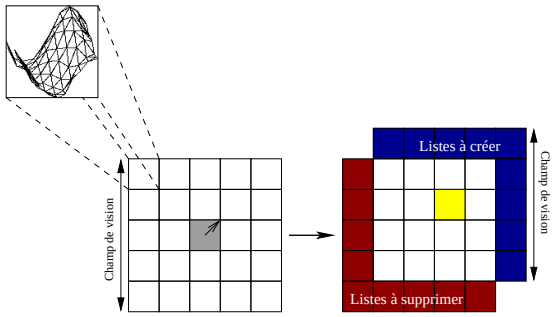


FIG. 3.3 – Gestion des listes d'affichage.

3.1.2 Ombrage et texture

À partir des résultats obtenus précédemment, nous voulons aboutir au rendu le plus réaliste du terrain généré. L'aspect temps réel de notre application nous contraint à préserver la charge CPU mais aussi à optimiser la taille mémoire requise à la représentation tridimensionnelle. Dans cette optique, nous introduisons une amélioration du calcul de l'ombrage effectué sous *OpenGL*.
L'ombrage d'un polygone est calculé à partir de la différence d'angle entre les *vecteurs normaux* aux sommets de ce polygone et la *source de lumière*[14] [9, 25]. Afin d'afficher, dynamiquement, les parties du terrain dans lesquelles se trouve l'utilisateur, cf. §3.1.1, le pré-calcul du vecteur normal à chaque sommet est indispensable. Ce pré-calcul correspond, en espace mémoire, pour un terrain de taille 1024×1024 à $(1024 \times 1024 \times 3 \times 4)$ octets[15], soit 12 Mo. Nous arrivons à diviser cette valeur par trois en utilisant quatre octets par vecteur normal ; soit 2^{10} variations possibles pour chaque composante tridimensionnelle[16].
Le rendu est finalisé par une application de texture de couleur unie, et une variation des couleurs[17] suivant l'altitude du sommet, voir figure 3.4[18].

[14] Vecteur décrivant la direction, l'intensité et la couleur de la lumière.

[15] Sous *OpenGL* un vecteur normal est composé de trois valeurs à virgule flottante. Soit $3 \times (4$ octets$) = 12$ octets par vecteur.

[16] La précision obtenue est de l'ordre de la troisième décimale. Nous obtenons globalement, un milliard d'ombrages possibles par sommet.

[17] Cette variation est comprise dans l'intervalle $[0, 255]$ et la couleur appliquée est prise dans une palette LUT ; vert $\rightarrow$ marron $\rightarrow$ blanc, cf. §2.1.3.

[18] Cette figure est composée d'une séquence d'images obtenues à partir d'un nuage de plasma 75×75 après un maillage régulier, un ombrage et une application de texture.

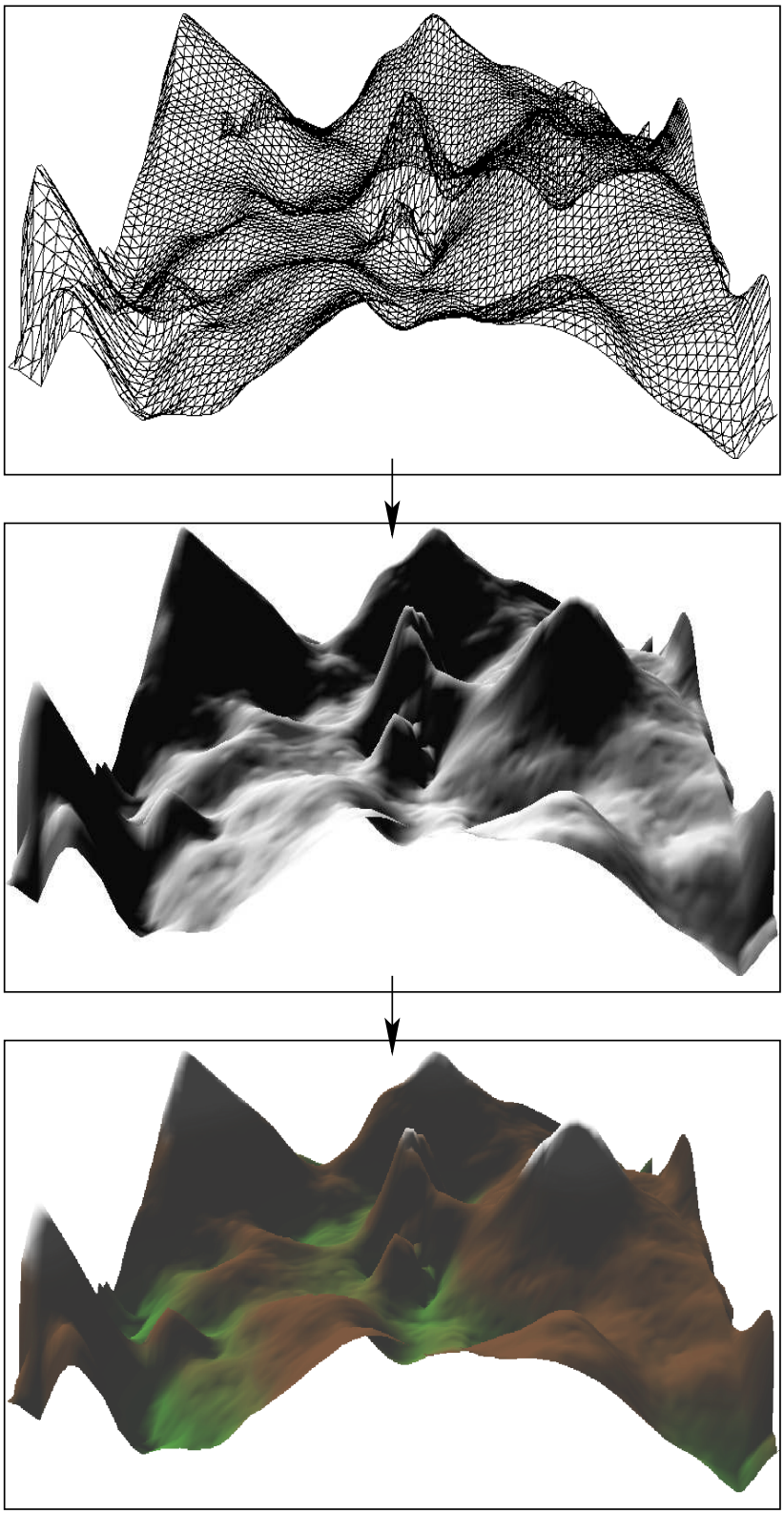


FIG. 3.4 – Calcul de l'ombrage et application de texture pour le rendu de terrain.

3.1.3 Les arbres

Afin de conclure notre démarche de génération de paysages, nous introduisons à notre environnement les arbres générés par l'interpréteur de L-Systèmes, cf. §2.4. Hormis l'aspect visuel que procure l'ajout d'arbres aux paysages, nous les considérons comme des obstacles statiques supplémentaires, voir figure 3.5, pouvant modifier le cheminement d'un véhicule, cf. §3.2.2 et §4.1. Les arbres sont placés sur le terrain de façon semi-aléatoire.
Nous définissons une classe d'arbres par intervalle d'altitudes. Chaque classe est associée à une fonction L-Systèmes[19], cette fonction génère l'image correspondante à la classe. Alors, pour un point (x, y, z), avec (x, z) aléatoirement sélectionné, nous plaçons une *famille d'arbres*[20] composée d'individus de la classe dont l'intervalle d'altitudes inclut la valeur "y".
Afin d'optimiser l'utilisation de la CPU, les arbres placés dans notre environnement sont gérés en listes d'affichage ; la gestion de l'affichage est comparable à celle utilisée pour le terrain, cf. §3.1.1. Pour les mêmes raisons, Nous réduisons la représentation tridimensionnelle d'un arbre à deux plans perpendiculaires texturés. La texture utilisée correspond à la classe de l'arbre créé, voir figure 3.5.

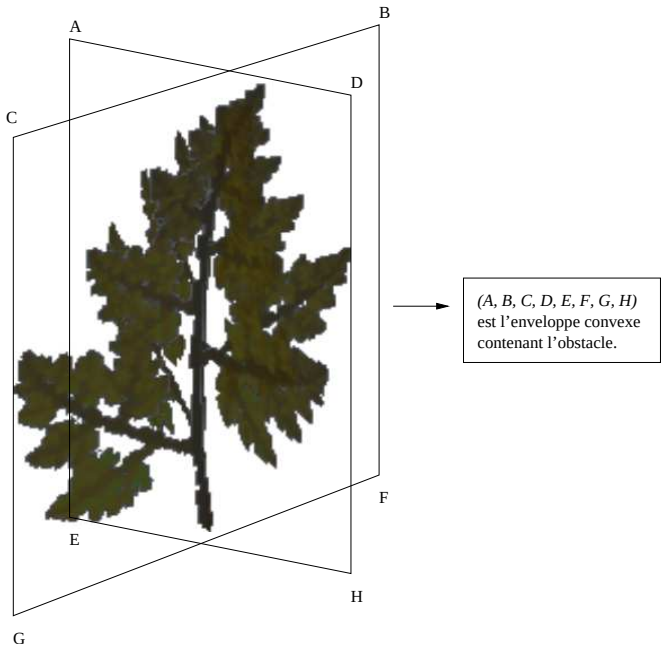


FIG. 3.5 – Représentation tridimensionnelle d'un arbre.

[19]Une fonction L-Systèmes détaille les règles utilisées pour la construction d'une classe d'arbres ; exemple : axiome, règles de productions, nombre de générations, intervalle d'altitudes.

[20]Une famille d'arbres est composée d'individus de la même classe. Ces individus sont disposés dans un périmètre donné. Chaque arbre possède ses propres dimensions.

Nous obtenons finalement un paysage tridimensionnel *complet* et affiché en temps réel[21]. La figure 3.6 est une capture d'écran d'un paysage généré par notre environnement.

FIG. 3.6 – Rendu temps réel d'un paysage tridimensionnel.

[21] Nous obtenons pour une résolution de 640 × 480 pixels avec un champ de vision de rayon 24 *ut «unité-terrain»* :

- une fréquence d'affichage supérieure à 60 images par seconde sur *AMD Athlon*^{tm} XP 1700+, 1 Go de RAM avec une carte graphique *ATI Rage 128 Pro* 32 Mo de mémoire graphique ;
- une fréquence d'affichage supérieure à 30 images par seconde sur *Intel Celeron*^{tm} 433 Mhz, 128 Mo de RAM avec une carte graphique *ATI Rage 128* 16 Mo de mémoire graphique.

3.2 Le véhicule

Notre environnement 3D pour pilotes automatiques génère différents types de terrains et effectue leur rendu tridimensionnel en temps réel. Nous complétons le rendu par l'importation des objets 3D représentant les véhicules gérés par *l'interface*[22]. La représentation 3D des véhicules et leur comportement simulent un modèle simplifié d'hélicoptère, cf. §3.2.2.

3.2.1 Importation du véhicule

Nous réalisons un module d'importation d'objets 3D quelconques. Les objets sont construits à partir d'un modeleur[23] et sauvegardés en fichiers *A.S.E.*[24]. Ils peuvent être simples ou complexes. Chaque objet est décrit par une liste de *matériaux*[25] et de formes géométriques simples. Ces formes sont données par une liste de sommets *«vertices»* et de *facettes*[26] ; les facettes sont créées à partir de la liste des sommets.
Ce module nous permet donc d'importer une large variété d'objets et de les incorporer à l'environnement 3D. Pour l'hélicoptère, nous utilisons quatre fichiers *A.S.E.*, chacun d'eux décrit respectivement son cockpit (cellule), sa queue, son rotor, son rotor de queue ou rotor anti-couple. Cette subdivision aide à la construction de l'enveloppe convexe, voir figure 3.7, contenant chaque sous-partie. Les enveloppes convexes sont utilisées dans la détection de collisions.

[22] L'interface est la partie du programme qui garantit la liaison entre un pilote automatique, ou l'utilisateur, et le couple véhicule - environnement.

[23] Un modeleur est un logiciel qui permet de créer des objets tridimensionnels complexes par assemblage d'objets plus simples. Il est souvent couplé à un *raytraceur* qui effectue le rendu d'une scène décrite par le modeleur. Exemple : *3D Studio Max, Blender*.

[24] *ASCII Scene Export* est un format de fichier décrivant une scène tridimensionnelle. Nous donnons un exemple obtenu pour une simple facette :

```
*GEOMOBJECT { ...
  *MESH { ...
    *MESH_VERTEX_LIST {
      *MESH_VERTEX 0 -0.0991 -4.0337 -4.4505
      *MESH_VERTEX 1 -0.0991 -4.5272 -3.9483
      *MESH_VERTEX 2 -0.0991 4.5556 3.9903
    }
    *MESH_FACE_LIST {
      *MESH_FACE 0 : A : 0 B : 1 C : 2 AB : 1 BC : 0 CA : 1
    }
  }
}
```

[25] Un matériel détaille l'aspect visuel de l'objet tridimensionnel. Il contient les informations sur la lumière affectée à l'objet - ambiante, diffuse, spéculaire - sa couleur, sa transparence, son ombrage, sa brillance *«Shininess»* et les éventuelles textures qui lui sont appliquées. Dans le cas d'une application de texture, les informations relatives au placage de texture sont ajoutées, voir *« Texture Map Indices »* dans [12, 25].

[26] Une facette est l'intérieur du polygone obtenu avec des sommets appartenant à un même plan. C'est l'élément de base, indispensable à notre importation d'objets dans un espace tridimensionnel.

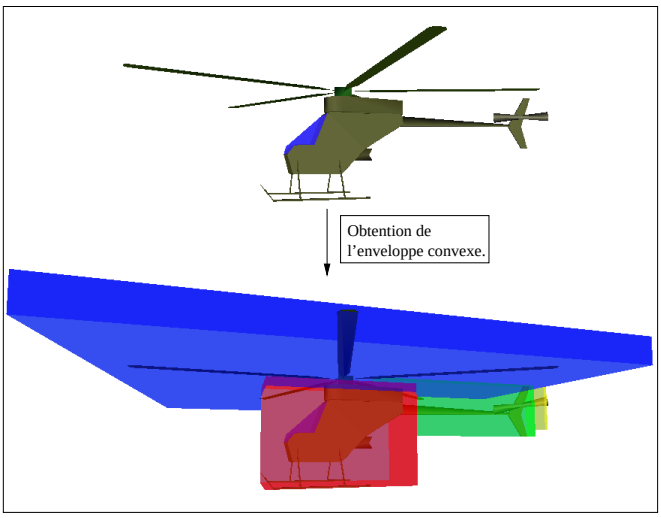


FIG. 3.7 – Importation du véhicule.

3.2.2 L'interaction du véhicule avec son environnement

Nous réalisons à partir des données importées un modèle physique simplifié d'un hélicoptère. L'hélicoptère possède une liberté de mouvement spécifique et interagit avec son environnement. Nous écrivons un *simulateur de vol* [42] pour hélicoptère dans lequel nous gérons la dynamique de mouvement du véhicule et les éventuelles collisions avec les éléments de l'environnement.

Dynamique de vol

Dans un hélicoptère, le *disque rotor*[27] est entraîné mécaniquement par le moteur et assure la *sustentation*[28] et la propulsion de l'appareil. Le disque rotor produit deux types de mouvements : le *tangage «pitch»* et le *roulis «roll»*, voir figure 3.8. Le rotor anti-couple, placé verticalement à l'extrémité de la queue, produit une force contraire à la direction de rotation du disque rotor ; il annule ainsi le *moment de la force* généré par la rotation des pales[29]. La variation de cette force entraîne donc une rotation à plat par rapport à l'axe vertical de l'appareil ; le mouvement résultant est appelé *lacet «yaw»*, voir figure 3.8. Dans notre modèle, nous ne considérons que la variation.

[27] Le disque rotor est une voilure tournante constituée de *pales* ; elle est montée horizontalement au dessus de l'appareil. Le disque rotor est aussi appelé rotor principal.

[28] La sustentation est générée par la portance aérodynamique. La portance est une force perpendiculaire au plan décrit par le disque rotor. Elle est obtenue par la dépression d'air créée par les pales. Les inclinaisons possibles des pales à différents endroits du disque rotor permettent de produire un déplacement dans pratiquement toutes les directions.

[29] Le rotor anti-couple, monté sur la plupart des modèles d'hélicoptères, permet d'éviter que l'aéronef tourne sur place.

Nous représentons le mouvement de l'hélicoptère par l'équation suivante[30] :

$$\begin{aligned} \sum \overrightarrow{\mathcal{F}} &= m \times \overrightarrow{a} \\ \overrightarrow{F} + \overrightarrow{P} &= m \times \overrightarrow{a} \\ \overrightarrow{F} + m \times \overrightarrow{g} &= m \times \overrightarrow{a} \end{aligned}$$

où la portance $\overrightarrow{F}$ est la force produite par le disque rotor, $\overrightarrow{P}$ est le poids de l'appareil, m la masse, $\overrightarrow{g}$ la gravité et $\overrightarrow{a}$ le vecteur accélération de l'hélicoptère (voir figure 3.8).

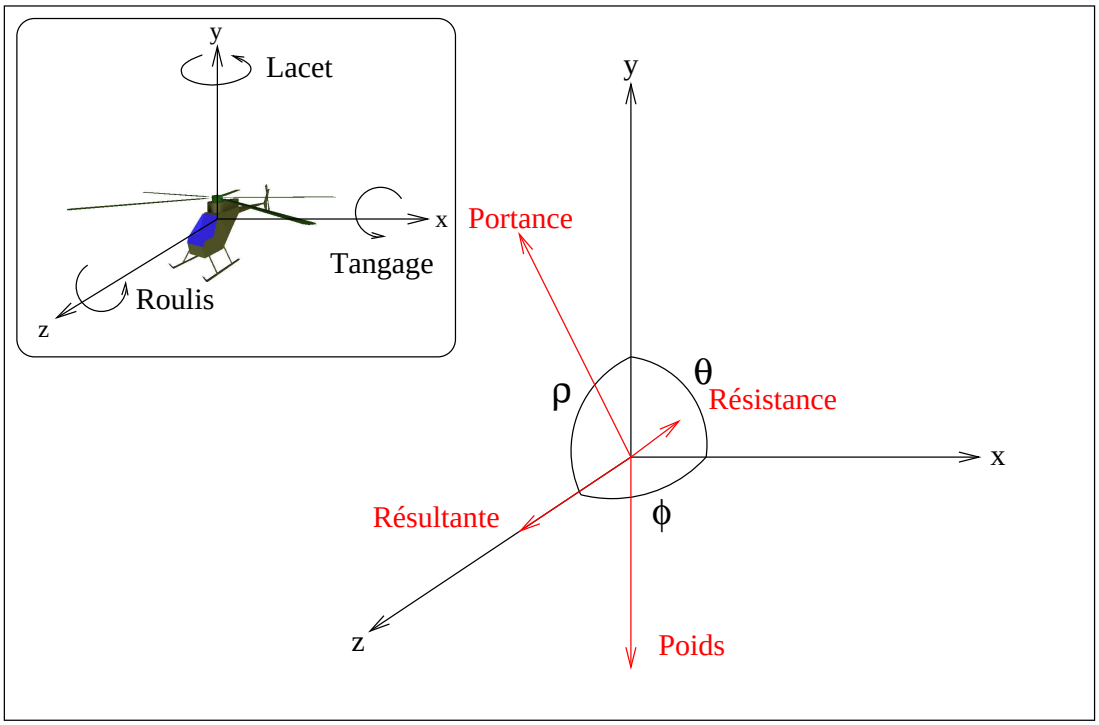


FIG. 3.8 – Dynamique de vol de l'hélicoptère.

En projetant les forces sur chacun des axes, nous obtenons pour :

$$\overrightarrow{F} = \begin{pmatrix} F_x \\ F_y \\ F_z \end{pmatrix}, \quad \overrightarrow{a} = \begin{pmatrix} a_x \\ a_y \\ a_z \end{pmatrix} : \begin{cases} a_x &= F_x \\ a_y &= F_y - g \\ a_z &= F_z \end{cases} \Rightarrow \begin{cases} a_x &= \frac{\cos\theta \times \| \overrightarrow{F} \|}{m} & (x) \\ a_y &= \frac{\sin\rho \times \| \overrightarrow{F} \|}{m} - g & (y) \\ a_z &= \frac{\cos\rho \times \| \overrightarrow{F} \|}{m} & (z) \end{cases}$$

[30] En première approximation, nous négligeons la force de frottement d'air, ou la traînée, qui sera introduite comme coefficient correcteur de la vitesse.

Le vecteur de vitesse résultant[31] est corrigé en ajoutant une force de résistance à l'air. Nous posons alors pour chaque composante de vitesse :

$$v_i = v_i - (k \times v_i^2)$$

où k est le *coefficient aérodynamique*[32].

Détection de collision

Afin de finaliser l'intégration de l'aéronef à notre environnement, nous déterminons les cas où celui-ci pourrait entrer en collision avec un élément du paysage ou avec un autre véhicule[33] ; pour chaque cas nous calculons la réaction produite par la collision. Dès que l'enveloppe convexe de l'hélicoptère, voir figure 3.7, contient une partie d'un élément extérieur - arbre[34] ou véhicule - ou que l'un de ses points se trouve en dessous du plan défini par le terrain - terre[35] ou *mer*[36] - le *gestionnaire d'événements*[37] de l'interface déclenche la détection de collision.

Les réactions générées par les collisions affectent le vecteur vitesse du véhicule. Nous donnons les modifications du vecteur pour chaque cas. Nous obtenons le tableau 3.1 pour R_v, R_t, R_a et R_m des réels tels que : $0 < R_m \ll R_a < R_t < R_v < 1$.

Collision	Réaction
Véhicule - Véhicule	$\overrightarrow{v_1} = -R_v \times \overrightarrow{v_2}$, $\overrightarrow{v_2} = -R_v \times \overrightarrow{v_1}$, avec $\overrightarrow{v_1}$ et $\overrightarrow{v_2}$ les vitesses réspectives des deux véhicules
Véhicule - Terre	$\overrightarrow{v} = -R_t \times \overrightarrow{v}$
Véhicule - Arbre	$\overrightarrow{v} = -R_a \times \overrightarrow{v}$
Véhicule - Mer	$\overrightarrow{v} = -R_m \times \overrightarrow{v}$

TAB. 3.1 – Modification du vecteur vitesse du véhicule dans les différents cas de collision.

[31] La correction est effectuée pour chacune des composantes de la vitesse : v_x, v_y, v_z.

[32] Le coefficient aérodynamique k est calculé pour une vitesse maximale donnée, $\approx 90\ m.s^{-1}$ pour ce type d'appareils, à partir de l'équation : $k = \frac{m \times a}{v_{max}^2}$.

[33] Dans le cas d'une collision inter-véhicules, nous considérons les collisions deux à deux.

[34] Dans notre environnement, un arbre est défini par son enveloppe convexe (voir figure 3.5)

[35] Nous vérifions la collision pour chaque plan défini par les facettes du maillage du terrain, cf. §3.1.1. Nous ne prenons que les facettes présentes dans le voisinage du véhicule.

[36] La mer est représentée par un seuil d'altitude minimale qui, une fois atteint par un véhicule, déclenche l'événement de collision correspondant. Nous représentons la mer dans le rendu 3D par une suite de polygones horizontaux liés en carré et texturés avec un degré de transparence.

[37] Le module de gestion des événements notifie les opérations à effectuer dans notre environnement tridimensionnel.

Chapitre 4

Navigation et Pilotes Automatiques

Nous présentons ici les moyens mis à la disposition des pilotes automatiques pour naviguer dans notre environnement 3D. Une interface de liaison entre l'univers tridimensionnel et le pilote du véhicule est créée. Nous réalisons un ensemble d'outils de perception, de commande et d'aide à la navigation. Deux pilotes automatiques sont écrits ; nous effectuons des tests par l'intermédiaire de notre interface.

Afin d'attribuer un maximum d'autonomie aux différents pilotes automatiques, nous mettons en place plusieurs instruments de bord. L'interface permet à chaque pilote d'accéder aux données de perception et aux commandes des hélicoptères qui lui sont assignés. Nous décrivons, en particulier, la modélisation des capteurs de distances. Ces capteurs sont les principeaux outils de perception des deux implémentations de pilotes automatiques.

Nous effectuons des tests afin de comparer nos deux pilotes automatiques. À partir d'un même point de départ, chacun des pilotes doit rallier le point d'arrivée défini par l'utilisateur. Nous posons une contrainte sur les altitudes obtenues pendant le vol. Pour minimiser ces altitudes, nous introduisons notre algorithme du *plus sûr chemin*. Cet algorithme calcule le chemin faisant le meilleur compromis entre distance et altitude. Nous donnons au pilote la possibilité de s'orienter par rapport à ce chemin ; il reçoit les coordonnées, angles et distances, des points successifs à atteindre.

Nous voulons obtenir une trajectoire de vol en rase-mottes. Le pilote doit suivre au plus près le chemin défini par l'algorithme du plus *sûr* chemin. Pour le premier pilote, nous écrivons un automate volant ; il réagit aux événements signalés par les capteurs de distances et corrige sa trajectoire par rapport au plus *sûr* chemin. Nos donnons les résultats obtenus pour ce pilote.
Nous développons, pour le second pilote, un réseau de neurones à apprentissage supervisé de type *Perceptron Multi-Couches*. Ce pilote automatique doit, dans

un premier temps, copier le comportement de l'automate volant ; des résultats intermédiaires sont obtenus. Nous effectuons ensuite une correction de la dynamique de vol du réseau de neurones. Pour le faire adhérer au plus près du terrain et maintenir une vitesse sensiblement constante, nous réalisons un apprentissage à partir d'un pilote humain ; ce dernier prend les commandes de l'appareil par l'intermédiaire de l'interface clavier. Nous testons le comportement du *Perceptron Multi-Couches* obtenu et nous donnons ses résultats.

4.1 Les Entrées / Sorties

Nous décrivons les outils d'aide au pilotage et à la navigation de l'hélicoptère dans notre environnement 3D. La gestion de l'hélicoptère est effectuée dans sa totalité par l'intermédiaire d'une *librairie*[1] de fonctions paramétrables. Nous classons ces fonctions en deux catégories[2] ; la perception et la réaction.
Cette librairie permet d'interfacer un *module externe de pilotage*[3], ou pilote automatique, pour tout véhicule créé. Chaque véhicule bénéficie donc d'un ensemble de commandes de réaction et d'outils de perception [20, 7, 22, 5, 33] de son environnement[4].

4.1.1 La perception

Les outils de perception se présentent sous la forme de capteurs et d'instruments de bord [42]. Nous implémentons — des *capteurs de distance* [33] — des instruments de mesure de *l'angle de direction*[5], *l'angle de tangage*, *l'angle de roulis* et la mesure de la *vitesse*[6] en *unité-terrain*[7] que nous noterons *ut* — un système radar. Ce dernier permet de positionner les différents véhicules présents dans le champ de vision de l'appareil.

[1] Toute l'interface est écrite en *ANSI-C* [17] / *OpenGL-Glut* et compilée sous la forme d'une librairie. Notre librairie est portable sur tous les systèmes disposant d'un compilateur *C* standard et des librairies *OpenGL* et *Glut - OpenGL Utility Toolkit*.

[2] Nous inversons délibérément le sens du couple *entrées / sorties* pour le faire correspondre au point de vu du pilote automatique. Nous ne parlons pas d'entrées / sorties de l'interface mais de *perception / réaction* du pilote.

[3] Le module de pilotage est un programme extrinsèque à notre librairie. Il communique avec l'interface par un ensemble de fonctions. Chaque module peut piloter plusieurs véhicules simultanément.

[4] Comme le décrivent Patrick Rives et Michel Devy dans *«La robotique mobile»* [20], nous utilisons des outils de perception pour la localisation et la navigation - les entrées - et des outils de perception pour la commande - les sorties ou la réaction.

[5] L'angle de direction est donné par une émulation de boussole numérique. Cette boussole renvoie l'angle longitudinal de l'appareil dans l'univers virtuel.

[6] La vitesse est le pas de déplacement discret multiplié par la fréquence d'affichage du simulateur. Elle est calculée en $ut.s^{-1}$.

[7] Une unité-terrain est la distance horizontale séparant deux sommets voisins, en 4-connexités, appartenant au maillage du terrain, cf. §3.1.1.

Dans cet ensemble de capteurs et d'instruments de mesure qui représentent une partie des données envoyées à l'entrée du module de pilotage[8], nous étudions l'implémentation des capteurs de distance. Ceux-ci servent à la perception des modules de pilotage implémentés dans les sections *«Automate volant»* et *«Perceptron Multi-Couches»*, cf. §4.3 et §4.4.

Les capteurs de distance

Les capteurs de distance renvoient la distance les séparant d'un obstacle. Ces capteurs détéctent les types d'obstacles suivant : la terre, la mer, les arbres et les véhicules. Tous les véhicules peuvent être munis de ce type de capteurs. Nous donnons, via l'interface, la possibilité de paramétrer leur usage.

Le nombre de capteurs de distance n'est pas fixé pour chaque véhicule ; nous le déterminons avant de lancer l'application. L'emplacement et l'orientation (ϕ, ρ) — longitude et latitude —, voir figure 3.8, de chacun des capteurs sont paramétrables.

Nous décrivons à partir de ces paramètres le fonctionnement des capteurs de distance. Pour chaque capteur orienté vers (ϕ_c, ρ_c) et positionné en (x_c, y_c, z_c) par rapport au repère décrit par le véhicule[9], un rayon est lancé en direction de (ϕ_c, ρ_c). Si le rayon rencontre un obstacle[10], le capteur renvoie la distance parcourue jusqu'à cet obstacle. En l'absence d'obstacles, le capteur renvoie la distance de sa portée maximale.
Dans notre environnement, nous utilisons des capteurs de distance dont la portée maximale vaut 10 *ut* avec un degré de précision de 0.01 *ut*. Pour un capteur de distance ne rencontrant pas d'obstacles, la complexité du calcul atteint 1000 itérations et chacune contient au minimum 3 tests[11]. Nous réalisons une méthode

[8] D'autres types de données ou d'informations sont récupérables ; par exemple : la topologie du terrain, le positionnement discret du véhicule sur le terrain, les variables d'état liées à l'appareil, les coordonnées du point cible à atteindre et le chemin à suivre défini par l'utilisateur, cf. §4.2.

[9] Ce repère a pour origine le centre de gravité de l'appareil. Ses axes x, y et z sont respectivement orientés vers la droite, le haut et l'avant du véhicule.

[10] Nous détectons un obstacle par la même méthode utilisée dans la détection de collision vue dans §3.2.2.

[11] Les tests sont au minimum faits sur un arbre, une parcelle du terrain et la mer. La complexité augmente si plusieurs véhicules ou arbres se trouvent dans le voisinage de l'appareil.

de *recherche dichotomique d'obstacles*[12] qui permet d'obtenir 28 itériations[13] au lieu de 1000 pour les cas extrêmes et sa complexité moyenne est logarithmique.

Nous avons sur la figure 4.1 une représentation tridimensionnelle de trois hélicoptères munis des capteurs de distance implémentés par défaut[14] dans l'interface.

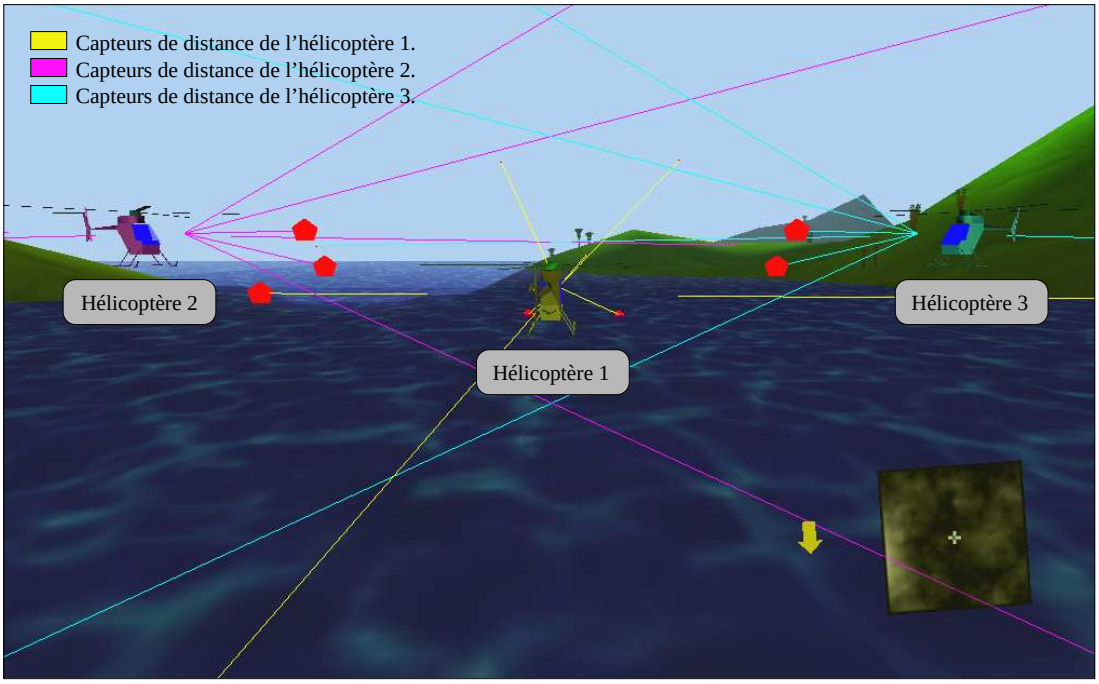


FIG. 4.1 – Capteurs de distance placés par défaut sur le véhicule.

[12]Nous donnons les étapes de l'algorithme de recherche dichotomique d'obstacles :

1. $D = 0$ *ut*, la distance.
2. Pour un pas de déplacement du rayon égal à 1 *ut* :
   - avancer ($D \leftarrow D + 1$) jusqu'à la collision avec un obstacle O_1 (aller à 3).
   - sinon sortir et renvoyer 10 *ut*.
3. Passer au pas de déplacement 0.1 *ut* :
   - reculer ($D \leftarrow D - 0.1$) jusqu'à sortir de l'obstacle O_1 (aller à 4).
4. Passer au pas de déplacement 0.01 *ut* :
   - avancer ($D \leftarrow D + 0.01$) jusqu'à retrouver l'obstacle O_1.
   - renvoyer D.

[13]Pour le cas extrême de la recherche dichotomique d'obstacles, l'obstacle est à une distance égale à 9.19 *ut*. Nous avons 10 itérations de pas 1 *ut*, 9 itérations de pas -0.1 *ut* et 9 itérations de pas 0.01 *ut*.

[14]Si aucune information concernant le nombre de capteurs de distance n'est spécifiée.

4.1.2 Les commandes

Nous donnons au module de pilotage la possibilité d'accéder à l'ensemble des commandes des véhicules qui lui sont assignés. Sur un hélicoptère, ces commandes représentent le *manche de pas cyclique*[15], le *palonnier*[16], *le levier de pas collectif*[17] et *la manette des gaz*[18].

Nous émulons les commandes réelles d'un hélicoptère par l'intermédiaire d'un automate à états ; chaque état est lié à un mouvement particulier[19]. En passant d'un état à un autre, l'automate produit une modification des paramètres de l'hélicoptère. Cette modification génère la dynamique de vol correspondante. De cette manière, toute action produite par le pilote devient accessible en *lecture / écriture* ; nous pouvons, à tout moment, interroger l'interface sur l'état de chaque commande.
Nous obtenons donc une représentation numérique des actions effectuées par le pilote. Cette représentation nous permet d'introduire la notion *d'apprentissage* par observation ; elle est utilisée pour l'apprentissage du *Perceptron Multi-Couches*, cf. §4.4. L'observateur apprend à piloter en *«copiant »* les réaction du pilote[20] de l'appareil, voir Fig 4.2.

[15]Le manche de pas cyclique incline le disque rotor afin de modifier la direction de la portance. Les mouvements générés sont le tangage ou le roulis, cf. §3.2.2.

[16]Les pédales du palonnier contrôlent le rotor anti-couple. Le palonnier génère un mouvement en lacet, cf. §3.2.2.

[17]Le levier de pas collectif contrôle le pas, ou l'inclinaison, des pales. Il fait varier l'intensité de la dépression aérodynamique générée par le disque rotor.

[18]La manette des gaz est placé sur le levier de pas collectif. Elle contrôle la vitesse de rotation du disque rotor.

[19]Chacun de ces mouvements est une combinaison construite à partir d'un ensemble de huit mouvements primaires.

[20]Le pilotage est assuré soit par un module ; un programme de pilotage automatique ; soit par un utilisateur via le clavier.

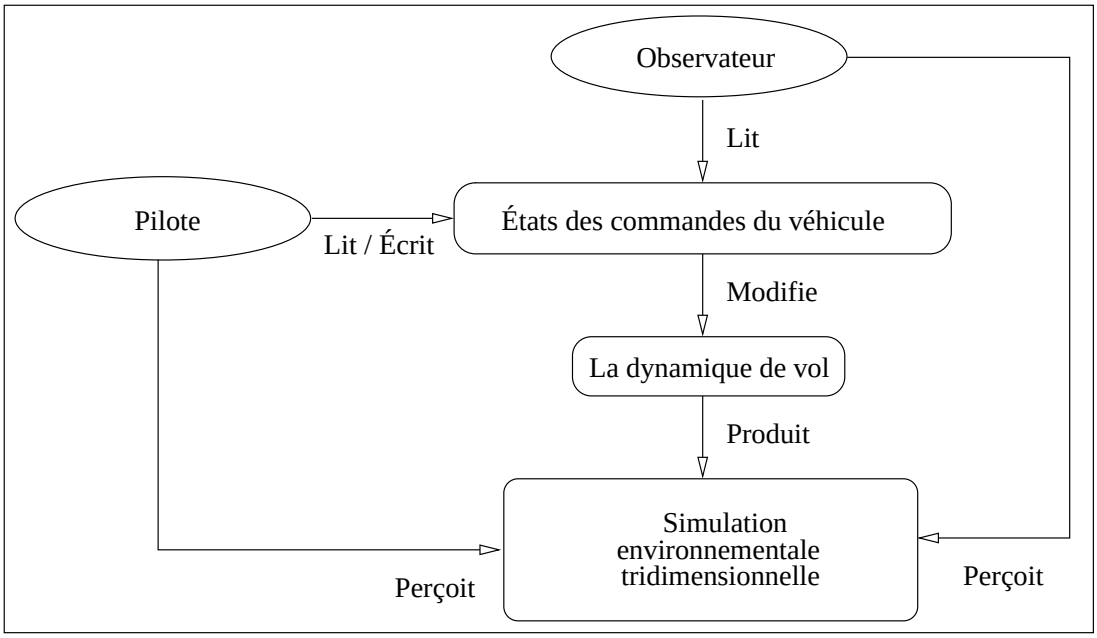


FIG. 4.2 – Processus d'apprentissage par observation.

4.2 Le *plus sûr chemin*

Afin de faciliter l'orientation des pilotes automatiques dans notre univers virtuel, nous développons un module de recherche du *plus sûr chemin*[21]. Notre algorithme du plus *sûr* chemin relie la position en cours du véhicule au point d'arrivée sélectionné par l'utilisateur. Cet algorithme produit un chemin minimisant le coût du couple distance[22], altitude[23].

L'algorithme est construit sur la méthode de recherche du plus court chemin dans un labyrinthe, cf. §2.3.2. Nous le modifions pour calculer des chemins dans des petites[24] cartes d'altitudes. Puis nous l'optimisons afin d'obtenir un temps de réponse satisfaisant sur des cartes plus grandes[25].

[21] L'utilisation du module du plus *sur* chemin est optionnelle. Chaque pilote automatique peut incorporer son propre algorithme de *planification de trajectoire* [5, 10, 11, 18]. En effet, les données relatives à la topologie du terrain sont accessibles pour chaque module de pilotage via les entrées de l'interface.

[22] La distance parcourue est calculée dans le plan bi-dimensionnel (x, z). La variation d'altitudes, la distance en y, n'est pas prise en compte dans cette distance ; l'optimisation de l'altitude est obtenue indépendamment.

[23] Le coeficient du coût de l'altitude est variable. En augmentant ce coût, nous préconisons, dans la mesure du possible, l'utilisation des chemins minimisant les altitudes.

[24] Carte d'altitudes dont la largeur n'excède pas quelques dizaines d'unités.

[25] La largeur de ces cartes dépasse le milier d'unités.

4.2.1 Du labyrinthe aux cartes d'altitudes

Nous généralisons l'algorithme utilisé pour les labyrinthes afin de calculer un plus court chemin sur un terrain quelconque. Les terrains générés par l'interface sont représentés sous la forme d'une matrice de valeurs entières comprises dans l'intervalle $[0, 255]$. Chaque valeur est interprétée comme une position (x, z) et une altitude y dans notre univers tridimensionnel. À l'inverse d'un labyrinthe, toutes les positions peuvent être envisageables[26] et le nombre de directions possibles à partir d'une position est fixé à huit[27].
Chaque déplacement sur le plan (changement de position) produit un déplacement vertical positif, négatif ou nul[28] noté $V(x, z)$. Par conséquent, nous posons que le coût du déplacement planaire en 8-connexité est pondéré, à un facteur multiplicatif près, par l'altitude en position d'arrivée.

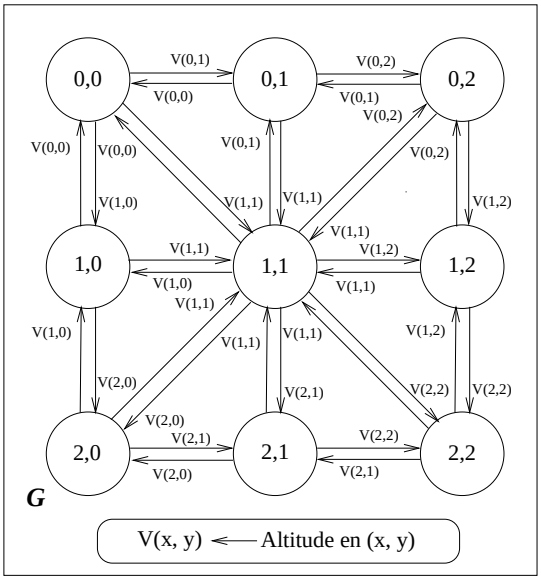


FIG. 4.3 – Parcours d'un graphe planaire pondéré pour le calcul du plus *sûr* chemin.

Nous représentons alors les déplacements en 8-connexités[29] par un graphe planaire G où chaque sommet exprime une position. Les sommets sont reliés entre eux par des arêtes entrantes et sortantes ; les arêtes entrantes d'un sommet sont

[26] Dans un labyrinthe, le déplacement vers une position murée est écarté.

[27] La connexité du cheminement dans un labyrinthe est fixé à quatre, cf. §2.3.1 ; nous l'augmentons dans le cas des terrains afin de diminuer la discrétisation des courbes de trajectoires du véhicule.

[28] Le déplacement vertical est calculé par la différence entre les valeurs contenues dans la nouvelle et l'ancienne position.

[29] Pour les positions placés sur les bords et coins de la carte d'altitude, les connexités sont respectivement égales à six et quatre.

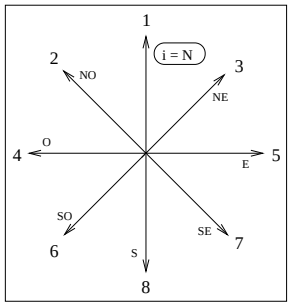
pondérées par l'altitude exprimée au sommet, voir Fig 4.3.
À partir de cette structure, nous construisons l'algorithme récursif de calcul du plus *sûr* chemin. Le parcours du graphe s'effectue en additionnant les pondérations des arêtes. Le sommet pour lequel les pondérations sont minimales est favorisé. Nous stockons les valeurs de distance (somme des pondérations) dans une matrice de dimensions égales aux dimensions du terrain ; cette matrice est utilisée afin de générer des coupes et n'explorer qu'une partie du graphe. Nous favorisons le minimum local et les sommets se trouvant dans la direction du but ; un ordre de préférence est établi[30].
Comme pour les labyrinthes, la construction du chemin est donnée en parcourant les distances pour rallier le point d'arrivée à partir du point de départ, cf. §2.3.2.

Détail de l'algorithme

Nous donnons les conditions initiales de l'appel de la fonction $Psc()$ puis son descriptif.

1. Pour une matrice de terrain $T[M][N]$ donnée ;
2. Pour un coefficient du coût de l'altitude $CoefCoutAltitude$ donnée ;
3. Initialiser une matrice d'entiers à 0 ; $Dist[M][N] = \{0\}$;
4. Représenter les déplacements possibles dans la matrice $d[8]$; d est un tableau de la structure composée de deux entiers x et z ;
5. Pour $d[8]$ matrice des déplacements possibles, définir l'ordre de passage préférentiel $depOrder[8][8]$ pour chaque direction de déplacement donnée[30] ;
6. Pour un point de départ (x_d, z_d) et un point d'arrivée (x_f, z_f) faire : $Psc(x_d, z_d, x_f, z_f, 0, -1)$;
7. Partir du point d'arrivée et reconstruire le chemin jusqu'au point de départ[31] ; pour un point à valeur v, prendre en 8-connexités le voisin qui a pour valeur $v - 1$, voir figure 2.12.

[30] Cette ordre est calculé pour chaque direction prise dans $d[8]$, table des directions possibles, nous donnons un exemple sur la figure suivante pour la direction $d[i]$ avec i = Nord :



[31] Pour reconstruire le chemin, nous utilisons la matrice des distance $Dist[M][N]$.

$Psc(x_1,\ z_1,\ x_2,\ z_2,\ Cout,\ lastPos)$
Début :
  $Cout \leftarrow Cout + T[x_1][z_1]$;
  Si $Dist[x_1][z_1] \geq 0$ et $Dist[x_1][z_1] \leq Cout$ alors :
    retour ;
  $T[x_1][z_1] \leftarrow CoefCoutAltitude \times Cout$;
  Si $x_1 = x_2$ et $z_1 = z_2$ alors :
    retour ;
  $dirB \leftarrow$ indice de la direction vers le but[32](x_2, z_2) ;
  $dirP \leftarrow$ indice de direction vers le voisin contenant la plus petite valeur ;
  Si $dirB = dirP$ alors :
    Si $dirB \neq lastPos$ alors :
      $Psc(x_1 + d[dirB].x,\ z_1 + d[dirB].z,\ x_2,\ z_2,\ Cout + 1,\ (dirB + 4)\%8)$;
  Sinon :
    Si $dirB \neq lastPos$ alors :
      $Psc(x_1 + d[dirB].x,\ z_1 + d[dirB].z,\ x_2,\ z_2,\ Cout + 1,\ (dirB + 4)\%8)$;
    Si $dirP \neq lastPos$ alors :
      $Psc(x_1 + d[dirP].x,\ z_1 + d[dirP].z,\ x_2,\ z_2,\ Cout + 1,\ (dirP + 4)\%8)$;
  Pour i dans $\{1, \cdots, 7\}$ faire :
    $b \leftarrow depOrder[dirB][i]$;
    si ($b = lastPos$ ou $b = dirP$) alors :
      continuer d'incrémenter i ;
    $n_x \leftarrow x_1 + d[b].x$;
    $n_z \leftarrow z_1 + d[b].z$;
    si ($n_x \geq 0$ et $n_x < M$ et $n_z \geq 0$ et $n_z < N$) alors :
      $Psc(n_x,\ n_z,\ x_2,\ z_2,\ Cout + 1,\ (b + 4)\%8)$;
Fin.

4.2.2 Optimisation du temps de calcul

Pour réduire le temps de calcul du plus *sûr* chemin, nous modifions les entrées de notre algorithme et changeons les structures de données.

Nous découpons la carte d'altitudes, matrice de dimensions $M \times N$, en sous-régions rectangulaires de dimensions $M' \times N'$. Une moyenne des altitudes est calculée pour chacune des sous-régions. Nous obtenons alors une matrice d'altitudes altérée, notée *terrain-altéré*, de dimensions $\frac{M}{M'} \times \frac{N}{N'}$; M' et N' sont choisis tels que : $\left(\frac{M}{M'}, \frac{N}{N'}\right) \in I\!N^{\,2}$. Le plus sûr chemin est calculé de la même manière que précédemment sur le *terrain-altéré* et les points le constituant sont stockés dans

[32] Cette direction est obtenue à partir des points (x_1, z_1) et (x_2, z_2). Nous découpons le plan en huit octants ; les indices sont respectivement pris dans Est, Nord-Est, Nord, Nord-Ouest, Ouest, Sud-Ouest, Sud et Sud-Est.

une liste chainée.
Nous modifions les coordonnées de chacun de ces points pour les remettre à l'échelle du terrain originel et nous calculons le mini-chemin qui les relie deux à deux. Nous obtenons de cette façon le chemin total reliant le point de départ au point d'arrivée, voir figure 4.4.

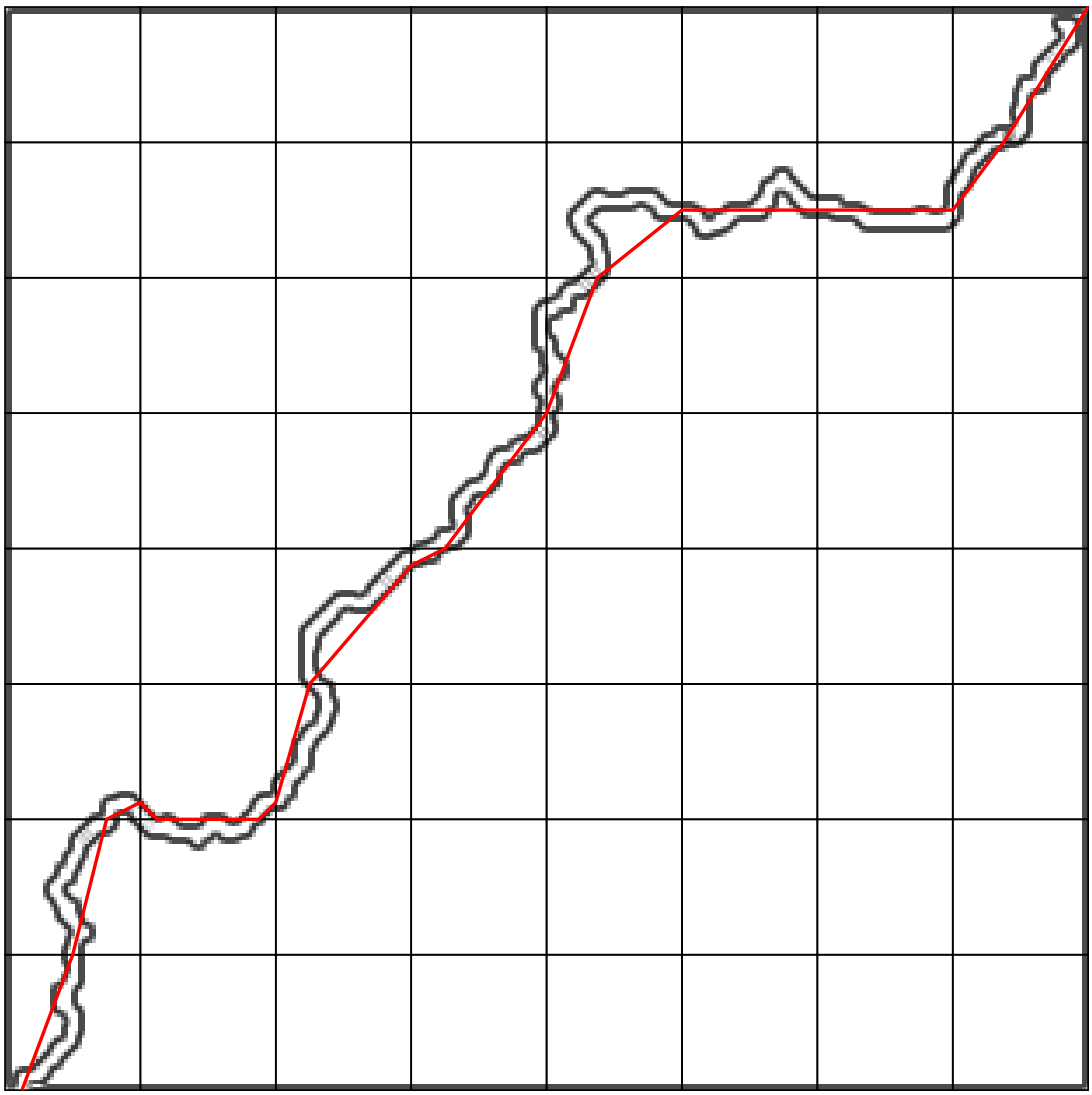


FIG. 4.4 – Dans le plus *sûr* chemin optimisé. Les segments de droites représentent le chemin *grossier* créé à partir du *terrain-altéré*. Le chemin détaillé est constitué des *mini-plus-sûrs-chemins* reliant les points du chemin *grossier*.

Résultats

Nous donnons dans la table 4.1 quelques résultats des temps de calcul[33] obtenus pour chacune des versions de notre algorithme. Les tests sont effectués, pour le même couple de points de départ et d'arrivée, sur différents nuages Browniens générés aléatoirement.

Première version	Version optimisée
249,479 s	0,665 s
141,838 s	0,573 s
164,485 s	0,722 s

TAB. 4.1 – Tableau comparatif des temps de calcul obtenus avec les différentes versions du plus *sûr* chemin.

[33] Ce temps de calcul est obtenu sur un *AMD Athlon^{tm} XP 1700+, 1 Go de RAM.* Système d'exploitation utilisé : *Linux Mandrake 8.2.*

4.3 Un automate volant

Nous avons développé un environnement 3D complet. Il génère différents types de paysages tridimensionnels et gère la création, l'affichage et le comportement de plusieurs hélicoptères par l'intermédiaire d'une librairie de fonctions. À partir de notre environnement, nous implémentons un *automate volant*[34].

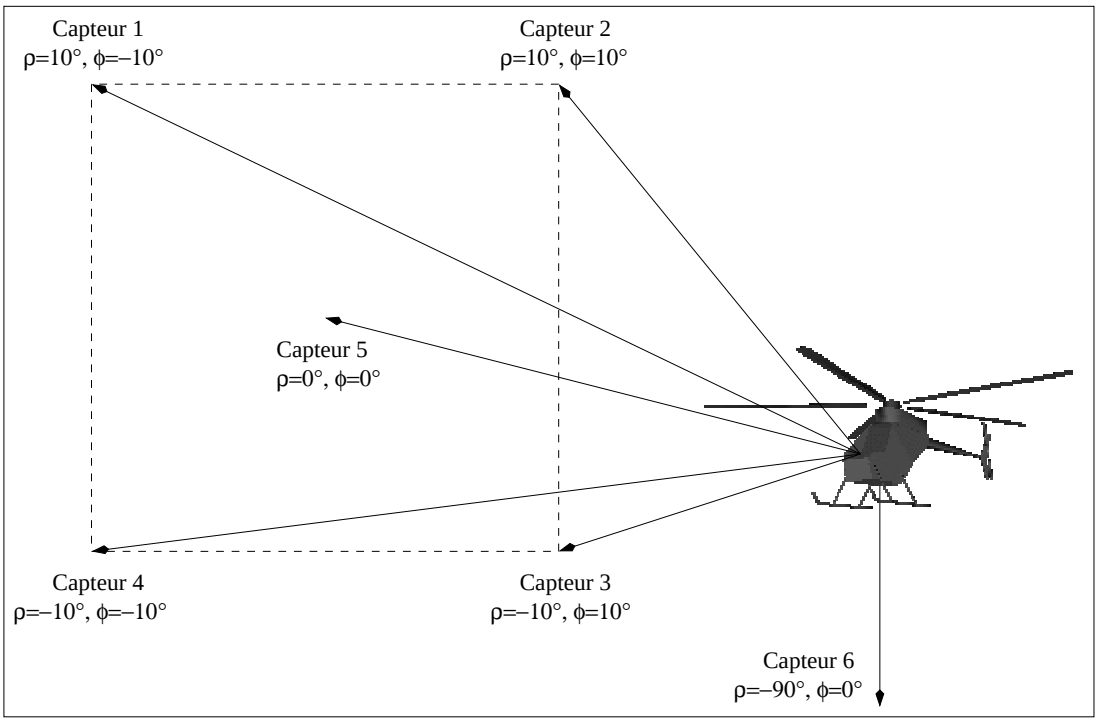


FIG. 4.5 – Le positionnement et l'orientation des six capteurs de distance placés sur l'automate volant.

4.3.1 Création de l'automate volant

Pour créer et tester l'automate, nous demandons à l'environnement de générer un paysage 3D quelconque : cartes plasma, Brown-Gauss ou labyrinthe telles que décrites dans le chapitre 2. Nous plaçons un hélicoptère sur le terrain ; cet hélicoptère est muni de six capteurs de distance, cf. §4.1.1, dont l'emplacement est montré sur la figure 4.5 ; il dispose aussi d'un capteur de vitesse horizontale vH. Muni de ces capteurs, l'automate volant s'oriente dans l'univers tridimensionnel. D'une part, l'emploi des informations données par le plus *sûr* chemin lui permet de naviguer dans un plan horizontal ; une variation d'angle longitudinale

[34]L'automate volant est un pilote automatique. Il produit un ensemble de réactions, des commandes de l'hélicoptère, suivant les données de perception. Son comportement est invariant.

$\delta\phi$ est calculée, elle donne l'orientation de l'hélicoptère par rapport à la prochaine position (coordonées (x, z)) du plus *sûr* chemin. D'autre part l'automate utilise les données des capteurs de distance pour réagir aux obstacles, à la fois verticalement et horizontalement, de façon à voler le plus bas possible. À partir de cette configuration d'hélicoptère nous construisons l'automate de pilotage[35] décrit sur la figure 4.6. La table 4.2 donne les conditions de changement d'état de l'automate.

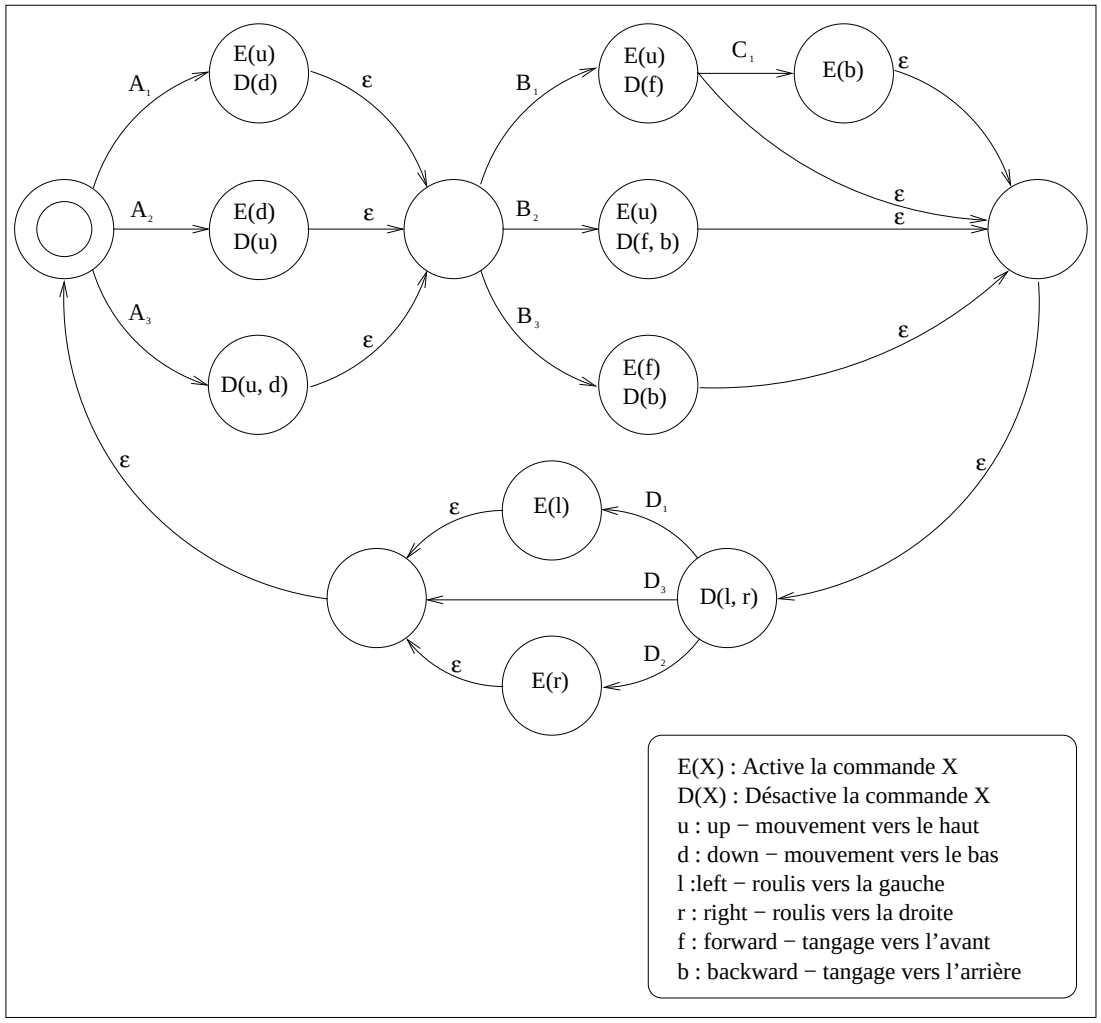


FIG. 4.6 – La réaction aux événements - perception via les capteurs de distance, de vitesse et d'orientation - de l'automate volant produit un changement dans le comportement de l'hélicoptère.

[35] Cet automate permet de piloter séquenciellement plusieurs unités d'hélicoptères. Tous ces hélicoptères contiennent au minimum les capteurs spécifiés pour l'automate ; celui-ci gère indépendamment la perception de chacun des aéronefs.

A_1	$\Longleftrightarrow$	$(Distance(Capteur\ 6) < 1\ ut)$ $\vee\ (Distance(Capteur\ 3) < 5\ ut)$ $\vee\ (Distance(Capteur\ 4) < 5\ ut)$
A_2	$\Longleftrightarrow$	$\neg A_1$ $\wedge\ (Distance(Capteur\ 6) > 2\ ut)$ $\wedge\ (Distance(Capteur\ 3) > 10\ ut)$ $\wedge\ (Distance(Capteur\ 4) > 10\ ut)$
A_3	$\Longleftrightarrow$	$(\neg A_1 \wedge \neg A_2)$
B_1	$\Longleftrightarrow$	$(Distance(Capteur\ 5) < 5\ ut)$ $\vee\ (Distance(Capteur\ 1) < 5\ ut)$ $\vee\ (Distance(Capteur\ 2) < 5\ ut)$
B_2	$\Longleftrightarrow$	$\neg B_1 \wedge \begin{pmatrix} (Distance(Capteur\ 5) < 10\ ut) \\ \vee\ (Distance(Capteur\ 1) < 10\ ut) \\ \vee\ (Distance(Capteur\ 2) < 10\ ut) \end{pmatrix}$
B_3	$\Longleftrightarrow$	$(\neg B_1 \wedge \neg B_2)$
C_1	$\Longleftrightarrow$	$(vH > 0)$
D_1	$\Longleftrightarrow$	$(\delta\phi > 0.5\ rad)$
D_2	$\Longleftrightarrow$	$(\neg D_1 \wedge (\delta\phi < -0.5\ rad))$
D_3	$\Longleftrightarrow$	$(\neg D_1 \wedge \neg D_2)$
ε	$\Longleftrightarrow$	vrai

TAB. 4.2 – Définition des conditions de changement d'état de l'automate volant.

4.3.2 Résultats

Nous réalisons une série de deux tests sur le comportement de l'hélicoptère piloté par l'automate volant.

Le premier test renseigne sur le chemin emprunté par l'automate[36]. Nous le comparons aux résultats du plus *sûr* chemin. La figure 4.7 montre le tracé des deux chemins.

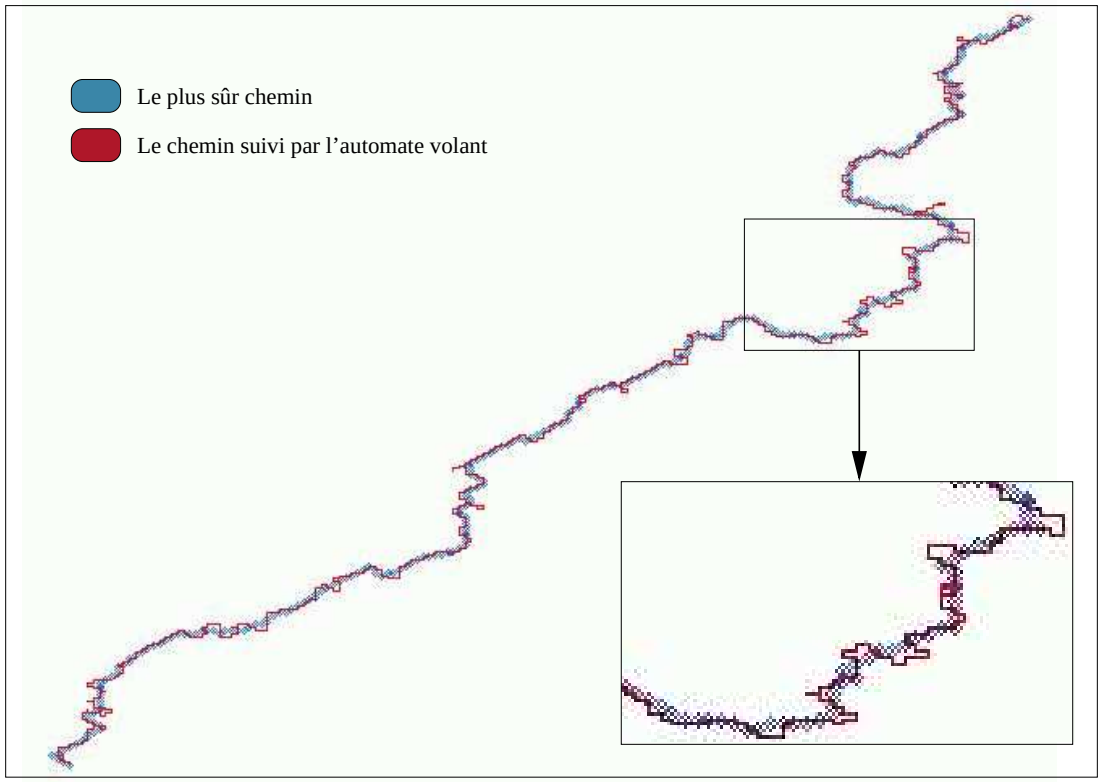


FIG. 4.7 – Comparaison entre le plus *sûr* chemin et le chemin parcouru par l'automate volant.

[36]Il s'agit de la trajectoire de l'hélicoptère ; elle est représentée en vue de haut sur une carte de positions discrètes.

Le second test réalisé, voir figure 4.8, génère la courbe des altitudes de l'automate volant. Cette courbe est obtenue le long du cheminement présent sur la figure 4.7. En bas de la figure 4.8 nous représentons les variations de vitesse horizontale de l'automate[37]. L'ensemble de ces résultats et ceux obtenus pour le *Perceptron Multi-Couches* sont comparés dans la section 4.4.

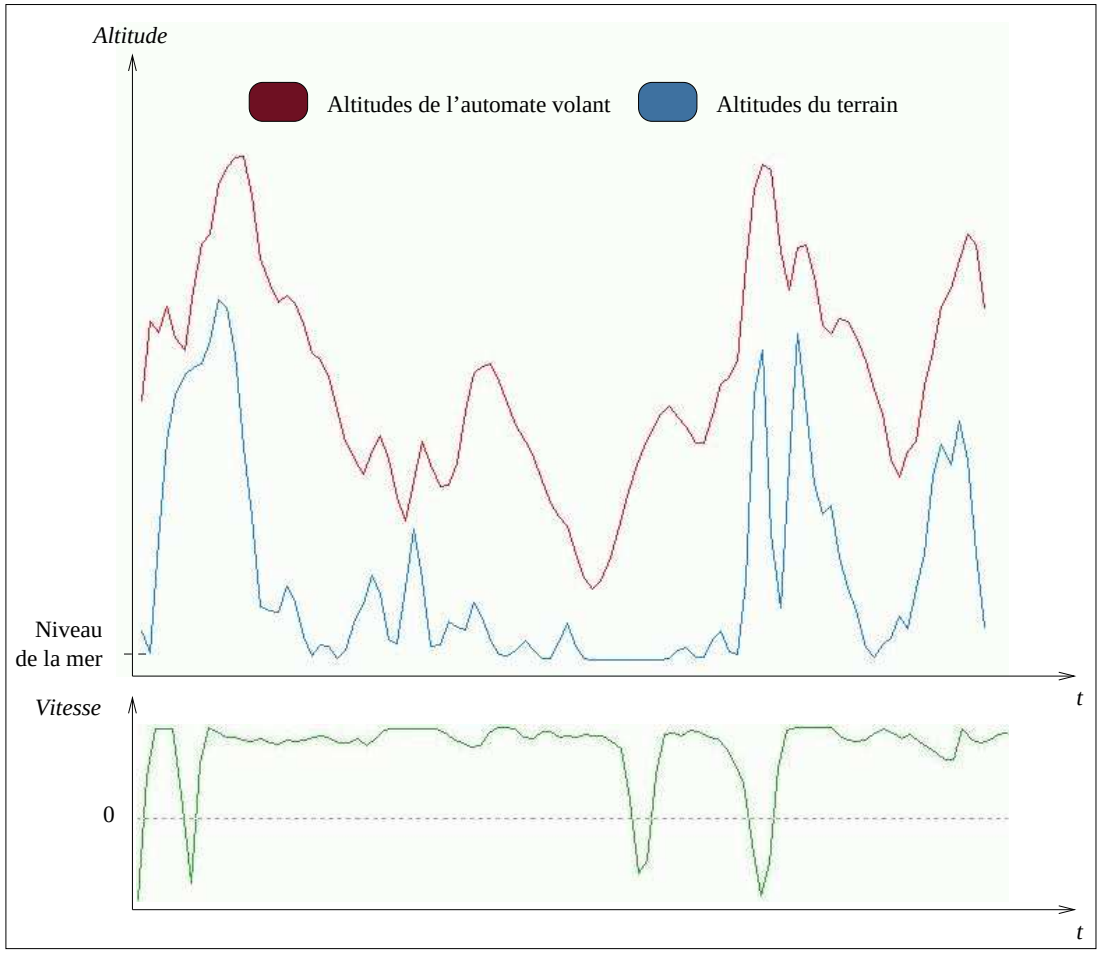


FIG. 4.8 – Courbe des altitudes et de vitesse horizontale de l'automate volant.

[37]La représentation de vitesses négatives signifie que la projection sur l'axe z de la portance de l'hélicoptère, la résultante, est négative ; voir figure 3.8.

4.4 Le Perceptron Multi-Couches

Nous réalisons au sein de notre environnement une seconde implémentation de pilote automatique. Notre approche consiste à obtenir un module de pilotage capable d'apprendre à *voler*; il doit observer[38] les réactions aux événements[39] du pilote[40] de l'hélicoptère. Pour l'apprentissage, nous choisissons d'écrire un réseau de neurones [26, 27, 13, 37, 39, 28, 16, 8] de type *Perceptron Multi-Couches* noté *P.M.C.* [37, 28, 16, 29, 8].

4.4.1 Modélisation du réseau

Notre *P.M.C.* est constitué de trois couches : les entrées, la couche cachée et les sorties. Chaque couche contient un nombre prédéfini de neurones.
Nous représentons sur la figure 4.9 la modélisation d'un des neurones de notre réseau. Chaque neurone est noté $N_{k,l}$, où k est l'identifiant de la couche et l la position du neurone dans cette couche. Il possède n entrées ou connexions entrantes[41] ; n étant le nombre de neurones de la couche précédente[42] $i = k - 1$; chaque entrée du neurone $N_{k,l}$ notée $X_{i,j}$ correspond à la sortie du neurone $N_{i,j}$, elle est pondérée par un poids $\omega_{k,l,j}$.
Nous obtenons la sortie $O_{k,l}$ du neurone $N_{k,l}$ par l'équation suivante :

$$O_{k,l} = f(Y_{k,l})$$

pour

$$Y_{k,l} = \sum_{j=0}^{n} X_{i,j} \times \omega_{k,l,j}$$

et f fonction[43] telle que :

$$f(Y) = \frac{1}{1 + e^{-Y}}$$

Nous obtenons alors pour chaque sortie[44] une valeur comprise dans l'intervalle $]0, 1[$. Pour les neurones de la dernière couche, la sortie est approximée par l'entier le plus proche, c'est à dire 0 ou 1. En posant $FALSE = 0$ et $TRUE = 1$ nous avons une interprétation du résultat du réseau de neurone.

[38] Le processus d'observation est décrit dans la section §4.1.2.

[39] Les événements sont représentés par l'ensemble des données de perception et les réactions s'effectuent via les commandes du véhicule.

[40] Le pilote de l'hélicoptère peut être un utilisateur humain ou un programme, tel que l'automate volant décrit dans la section §4.3.

[41] Les neurones d'une même couche ne sont pas connectés entre eux.

[42] Pour les neurones de la première couche, le nombre d'entrées correspond aux données de perception ; nous fixons ce nombre à un et nous posons qu'il existe autant de neurones d'entrée que de données de perception.

[43] La fonction f est communément appelée : *seuil du neurone*. Dans notre cas, la fonction de seuil est une sigmoïde.

[44] Le calcul de l'ensemble des sorties de chaque neurone est appelé : *propagation*.

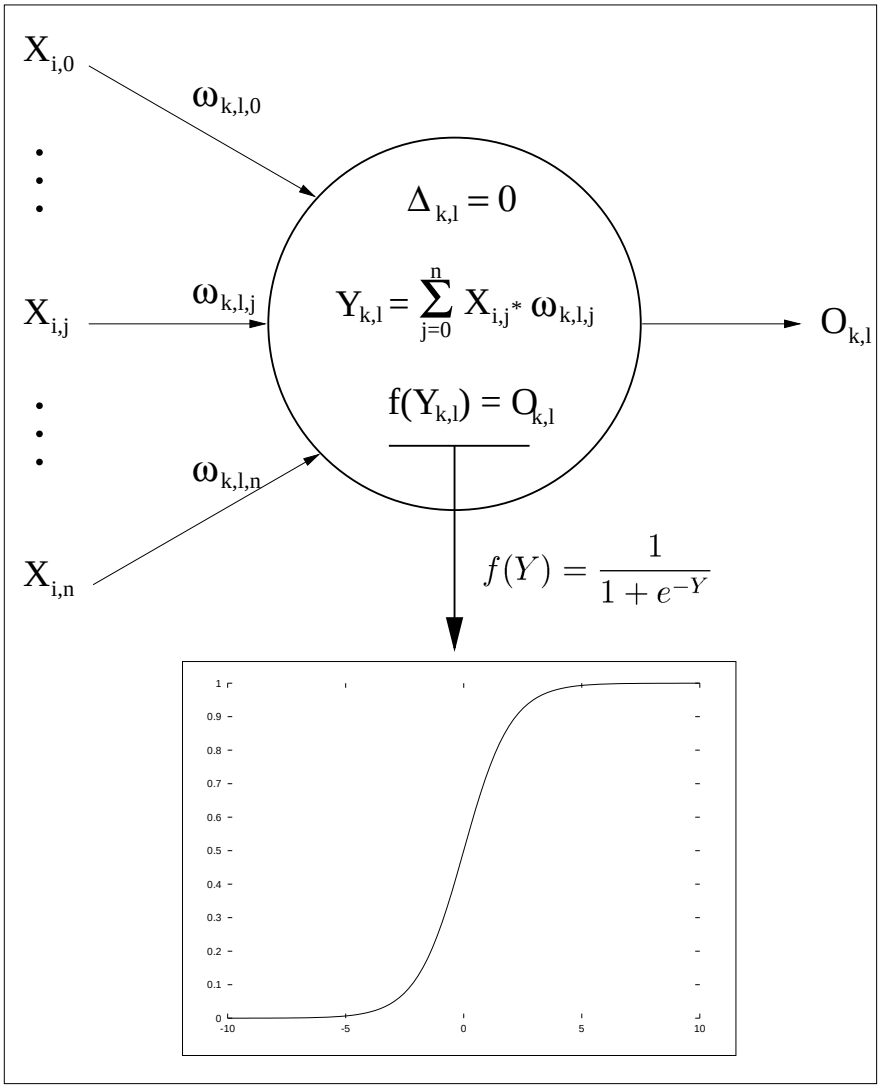


FIG. 4.9 – Descriptif du modèle de neurone utilisé pour le pilote automatique. Pour un neurone noté $N_{k,l}$, $\Delta_{k,l}$ est l'erreur de sortie du neurone ; elle sert, dans l'algorithme de rétropropagation du gradient, à la correction des poids $\omega_{k,l,j}$. Elle est initialement nulle.

Nous avons donc un réseau de neurones de type *P.M.C.* construit sur trois couches. Il reçoit en entrée les données de perception du véhicule[45] et émet en sortie les états des commandes de ce véhicule.

La figure 4.10 montre la construction du *Perceptron Multi-Couches*. Pour les

[45] Ces données sont des valeurs réelles émises par les différents types de capteurs placés sur le véhicule.

besoins du test, nous fixons pour le *P.M.C.* les mêmes entrées / sorties utilisées dans le cas de l'automate volant, cf. §4.3.

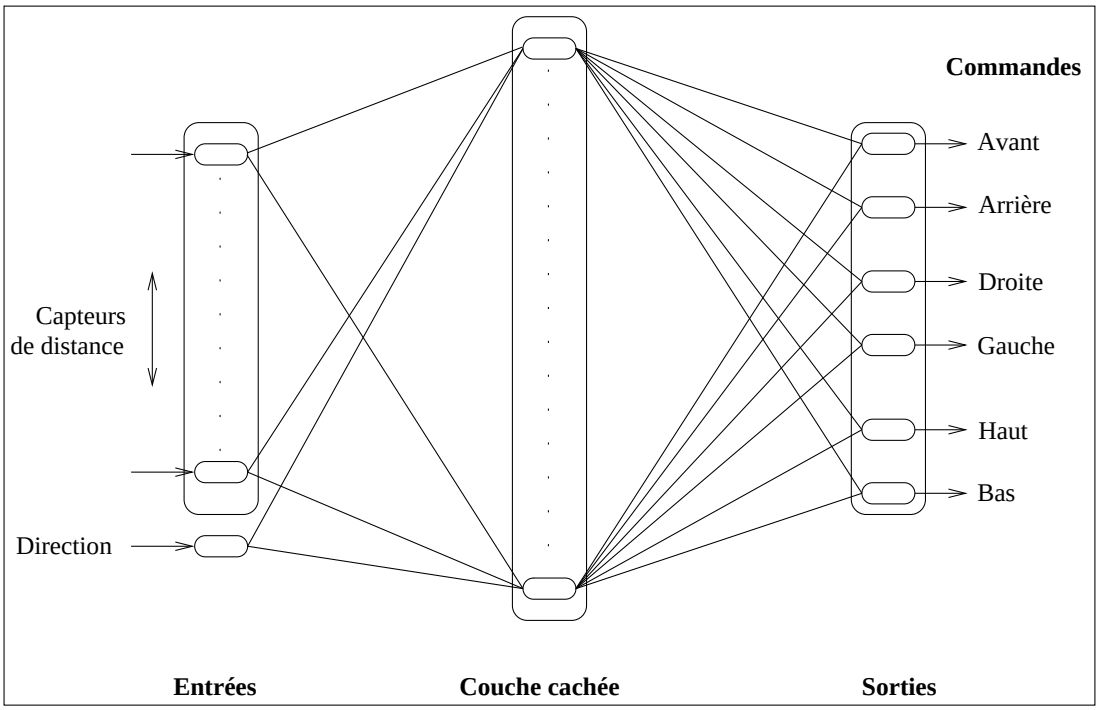


FIG. 4.10 – Structure du Perceptron Multi-Couche choisi pour l'apprentissage et le pilotage automatique.

4.4.2 La procédure d'apprentissage du *P.M.C.*

Nous décrivons la procédure d'apprentissage réalisée pour le test du *Perceptron Multi-Couches*. Nous commençons par donner le détail de l'algorithme de *rétropropagation du gradient*, extension de l'algorithme de Widrow-Hoff [44]. Il est employé dans les différentes phases de la procédure.
Nous montrons pour chaque étape les résultats intermédiaires obtenus.

La rétropropagation du gradient

Afin de réaliser l'apprentissage du *P.M.C.* nous effectuons une correction sur les poids $\omega_{k,l,j}$ du réseau. Cette correction est produite par l'algorithme de rétropropagation du gradient [29, 15].

Pour le vecteur d'entrées, ou la perception $\overrightarrow{X_0}$, le vecteur de sorties $\overrightarrow{O_{m-1}}$, avec

$m-1$ l'index de la dernière couche, et le vecteur de sorties attendues[46] $\overrightarrow{C}$, nous corrigeons les poids du *P.M.C.* afin de réduire l'erreur $\|\overrightarrow{C} - \overrightarrow{O_{m-1}}\|$.
Nous donnons les étapes d'initialisation du *P.M.C.*, la propagation, cf. 4.4.1, et le détail de l'algorithme de rétropropagation de gradient :

1. construire le réseau, voir figures 4.9 et 4.10 ;
2. pour tout neurone $N_{k,l}$, initialiser aléatoirement les poids $\omega_{k,l,j}$ du réseau[47], i = k - 1 ;
3. prendre un échantillon $(\overrightarrow{X_0}, \overrightarrow{C}) \in IR^e \times IR^s$;
4. Pour $\overrightarrow{X_0}$ le vecteur d'entrées de la première couche du réseau, effectuer la propagation jusqu'à la couche de sortie : $O_{k,l} = f(Y_{k,l})$, cf. 4.4.1 ; nous obtenons le vecteur de sorties du réseau $\overrightarrow{O_{m-1}}$ pour $k = m-1$;
5. Pour $\overrightarrow{C}$ le vecteur de sorties attendues posons[48] $\varepsilon = \frac{\|\vec{C} - \vec{O_{m-1}}\|}{S}$, faire :

   $pmcRetropropagation(\overrightarrow{C}, \varepsilon)$;

$pmcRetroPropagation(\overrightarrow{C}, \varepsilon)$
Début :
  $i = m-1$, index de la dernière couche ;
  pour tout neurone $N_{i,j}$ de la couche i, faire :
    $\Delta_{i,j} = O_{i,j} \times (1 - O_{i,j}) \times (C_j - O_{i,j})$;
  décrémenter i ;
  tant que $i \geq 0$, faire :
    pour tout neurone $N_{i,j}$ de la couche i, faire :
      $\Delta_{i,j} = 0$;
      pour tout neurone $N_{k,l}$ de la couche $k = i+1$, faire :
        $\Delta_{i,j} = \Delta_{i,j} + (\Delta_{k,l} \times \omega_{k,l,j})$;
      $\Delta_{i,j} = \Delta_{i,j} \times O_{i,j} \times (1 - O_{i,j})$;
    décrémenter i ;
    $nbEntreesParNeurone = 1$;
    pour toute couche i du réseau, faire :
      pour tout neurone $N_{i,j}$ de la couche i, faire :
        pour tout k de 0 à $nbEntreesParNeurone$, faire :
          $\omega_{i,j,k} = \omega_{i,j,k} + \varepsilon \times \Delta_{i,j} \times X_{i,k}$
      $nbEntreesParNeurone$ = nombre de neurones de la couche i ;
Fin.

[46] Le couple $(\overrightarrow{X_0}, \overrightarrow{C})$ est appelé échantillon ; il est tel que : $(\overrightarrow{X_0}, \overrightarrow{C}) \in IR^e \times IR^s$ avec e le nombre d'entrées et s le nombre de sorties du *P.M.C.*. Dans notre cas, e et s sont respectivement donnés par le nombre de neurones de la première et dernière couche du réseau.
[47] Les poids $\omega_{k,l,j}$ sont pris dans l'intervalle $]-1, +1[$.
[48] Dans la pratique, nous modifions la valeur ε pour 100 itérations, c'est à dire après 100 échantillons. À l'initialisation ε vaut 1.

Première phase d'apprentissage, copie de l'automate volant

Dans cette phase, nous partons d'un *P.M.C.* dont les poids $\omega_{k,l,j}$ sont initialisés aléatoirement. Pour pouvoir comparer les deux pilotes automatiques, nous générons le même environnement que celui utilisé dans le cas de l'automate volant, cf. 4.3. Nous plaçons d'abord l'automate aux commandes de l'hélicoptère, le P.M.C. *observe* le comportement de l'automate. Pour l'ensemble des données de perception $\overrightarrow{X_0}$ pris à un moment donné, l'automate renvoie les commandes de l'hélicoptère, $\overrightarrow{C}$: la sortie attendue. Le couple $(\overrightarrow{X_0}, \overrightarrow{C})$ est utilisé pour l'apprentissage par rétropropagation du gradient du réseau de neurones. Nous arrêtons l'apprentissage après la convergence du réseau, voir figure 4.11.

Remarque : Nous pouvons accélerer cette phase d'apprentissage en allouant plusieurs hélicoptères au même pilote. Nous obtenons de cette façon une variété de données de perception et plusieurs commandes de réactions en simultané.

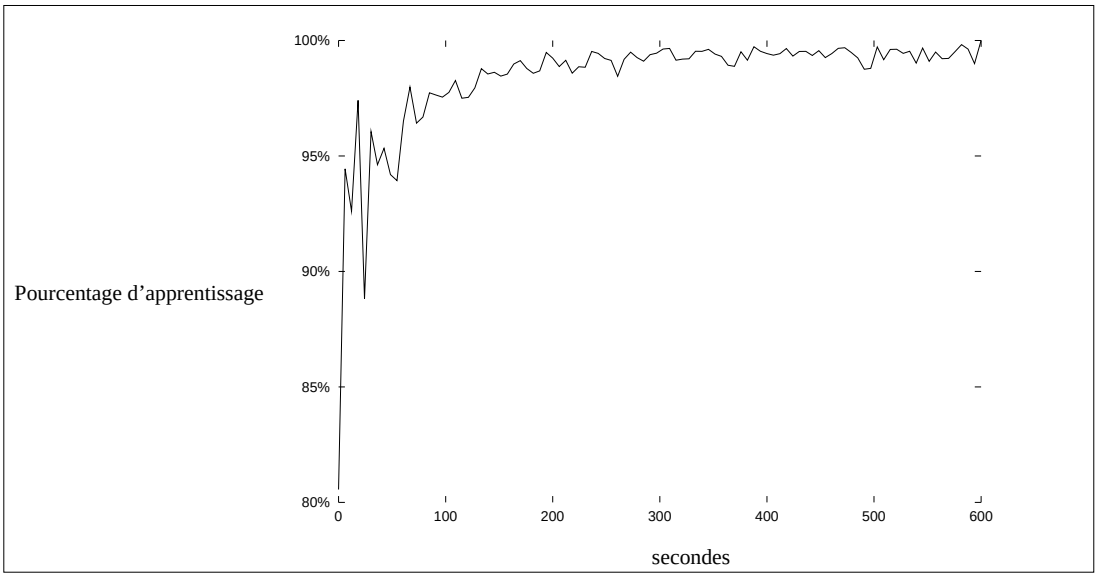


FIG. 4.11 – Courbe d'apprentissage du *Perceptron Multi-Couches*, 64 neurones en couche cachée. Cette courbe est réalisée à partir d'échantillons générés par l'automate volant en temps rééel, 40 images par seconde. L'automate est simultanément aux commandes de sept hélicoptères.

Une fois l'apprentissage du *P.M.C.* terminé, nous désactivons l'automate volant et nous relions directement la sortie du réseau de neurones aux commandes de(ou des) hélicoptère(s).

Nous effectuons les tests, dans des conditions équivalentes au test de l'automate volant, sur le chemin suivi par le *P.M.C.* et les variations d'altitudes obtenues.

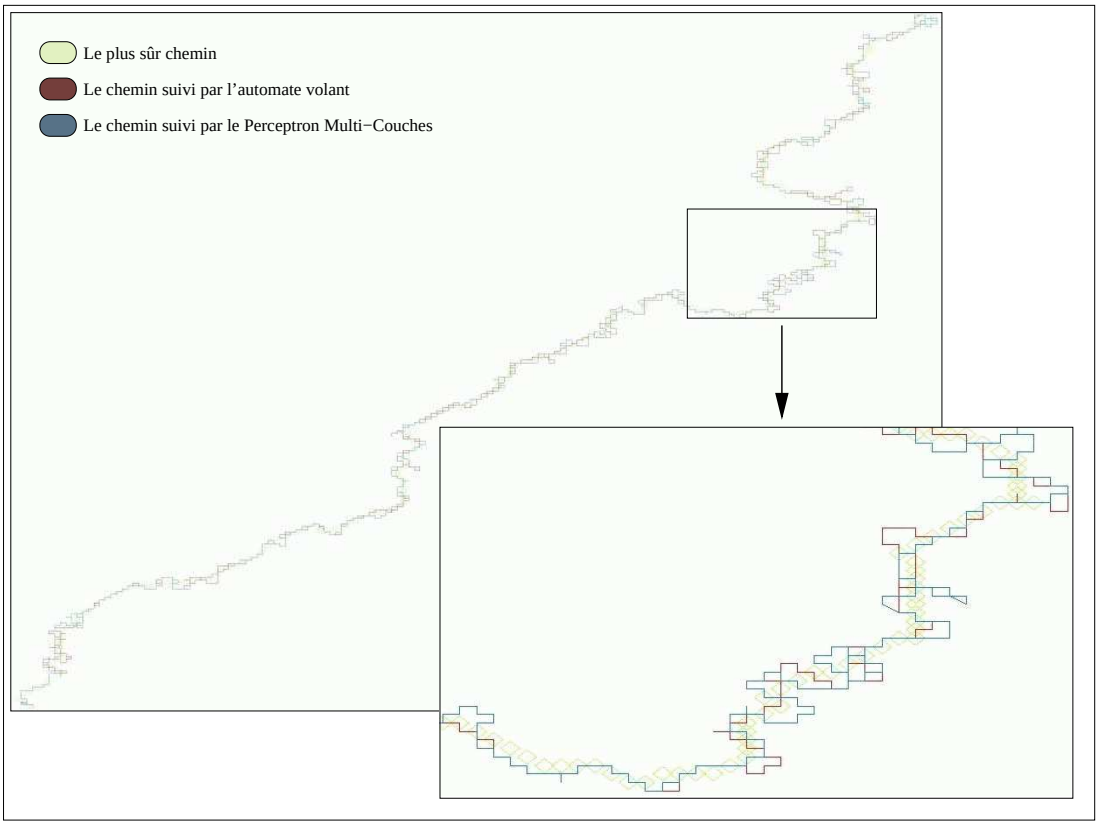


FIG. 4.12 – Comparaison entre le plus *sûr* chemin, le parcours de l'automate volant et du *Perceptron Multi-Couches*.

La figure 4.12 montre les variations dans le chemin parcouru par l'automate volant et le *Perceptron Multi-Couches*. Le comportement des deux pilotes automatiques est sensiblement équivalent. Ce rapprochement est dû à la convergence, $\approx 99\%$, du réseau de neurones obtenue lors de la phase d'apprentissage.

Le second test concernant la variation des altitudes, la distance au sol et la vitesse horizontale est aussi effectué dans des conditions équivalentes à celles du test de l'automate volant.

La figure 4.13 montre les variations d'altitude et de vitesse horizontale obtenues pour le *P.M.C.* La table 4.3 donne un comparatif entre nos deux pilotes automatiques pour les moyennes de vitesse et de distance par rapport au sol.

	Vitesse horizontale	Distance par rapport au sol
Automate volant	$1,344\ ut/s$	$44,784\ ut$
P.M.C.	$1,501\ ut/s$	$49,3375\ ut$

TAB. 4.3 – Comparatif-1 entre l'automate volant et le $P.M.C$; la vitesse horizontale moyenne est obtenue en *unité-terrain* par seconde, pour une fréquence de 40 itérations par seconde ; la distance par rapport au sol est donnée en *unité-terrain*.

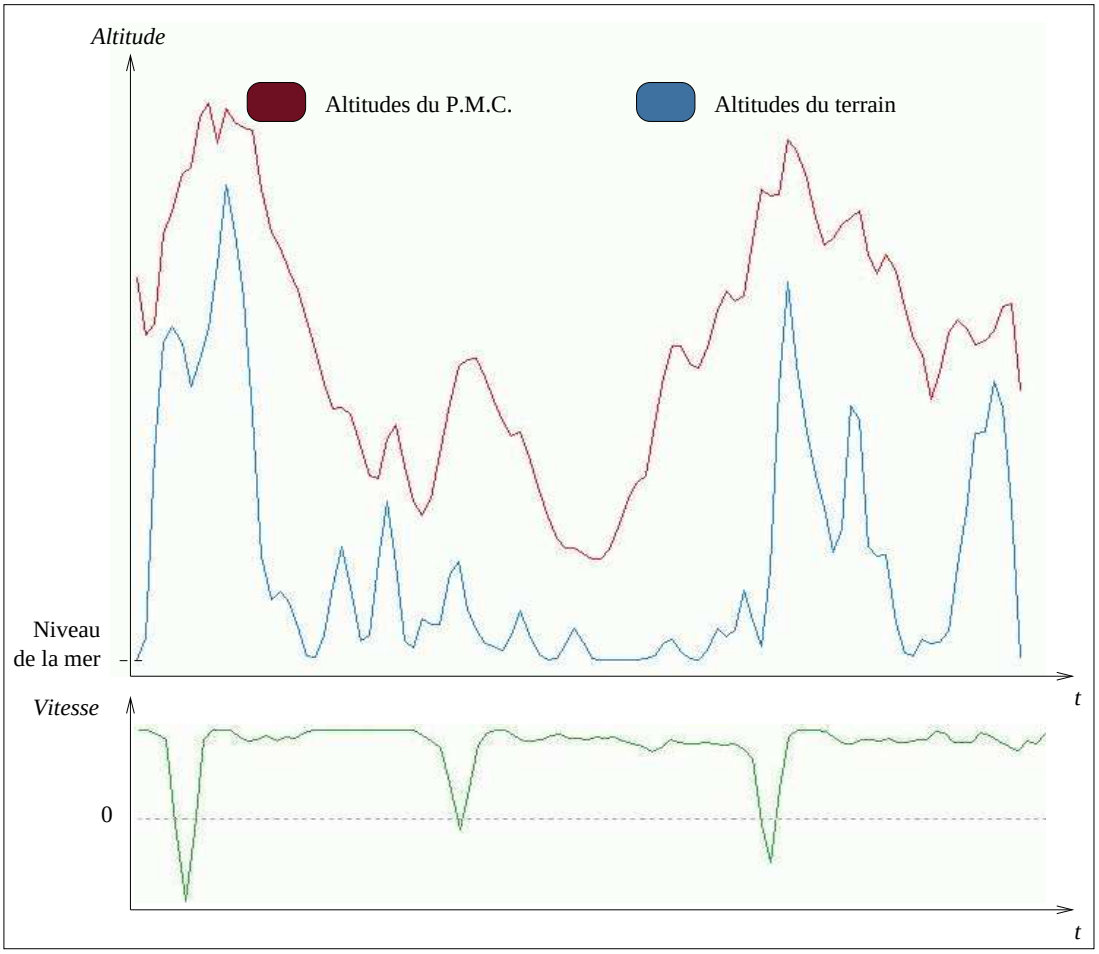


FIG. 4.13 – Courbe des altitudes et de vitesse horizontale du *Perceptron Multi-Couches*. Cette courbe est obtenue après la première phase d'apprentissage.

Deuxième et dernière phase d'apprentissage, vol en rase-mottes

Pour cette deuxième et dernière phase de l'apprentissage de notre *P.M.C.* nous réalisons une réduction de la distance par rapport au sol. Le réseau de neurones doit aboutir à une dynamique de vol en rase-mottes.
Nous partons de la configuration du *P.M.C* obtenue dans la première phase de l'apprentissage[49]. Le réseau de neurones est placé comme observateur ; nous laissons les commandes de roulis, droite / gauche, à l'automate volant et nous, l'utilisateur humain, prenons les commandes restantes de l'hélicoptère. Nous contrôlons ces commandes par l'intermédiaire de l'interface clavier. Nous adoptons un comportement *dangereux*, c'est à dire voler au plus près du sol et à grande vitesse. Cette phase d'apprentissage se déroule sur plusieurs minutes, $\approx 10\ mn$. Nous pouvons tester, à tout moment, le comportement du réseau de neurones en lui redonnant les commandes du véhicule ; nous arrêtons cet apprentissage dès que le réseau de neurones aura acquis le comportement souhaité.

Dans les mêmes conditions[50] que précédemment, nous réalisons un test sur les altitudes et vitesses horizontales du nouveau pilote automatique. Nous obtenons sur la figure 4.14 les nouvelles courbes des altitudes et de vitesse. Le *P.M.C.* a appris à voler en rase-mottes et à vitesse sensiblement constante. La table 4.4 donne les nouvelles moyennes de distance par rapport au sol et de vitesse horizontale.

	Vitesse horizontale	Distance par rapport au sol
Automate volant	$1,344\ ut/s$	$44,784\ ut$
P.M.C.	$1,501\ ut/s$	$49,3375\ ut$
P.M.C. **amélioré**	$1.2522\ ut/s$	$19,3815\ ut$

TAB. 4.4 – Comparatif-2 entre l'automate volant, le *P.M.C* et le *P.M.C* amélioré ; la vitesse horizontale moyenne est obtenue en *unité-terrain* par seconde, pour une fréquence de 40 itérations par seconde ; la distance par rapport au sol est donnée est *unité-terrain*.

[49]Les poids du réseau sont sauvegadés et réutilisés comme poids initiaux pour la deuxième phase d'apprentissage.
[50]Le test est réalisé pour le même terrain et dans les mêmes conditions de départ et d'arrivée.

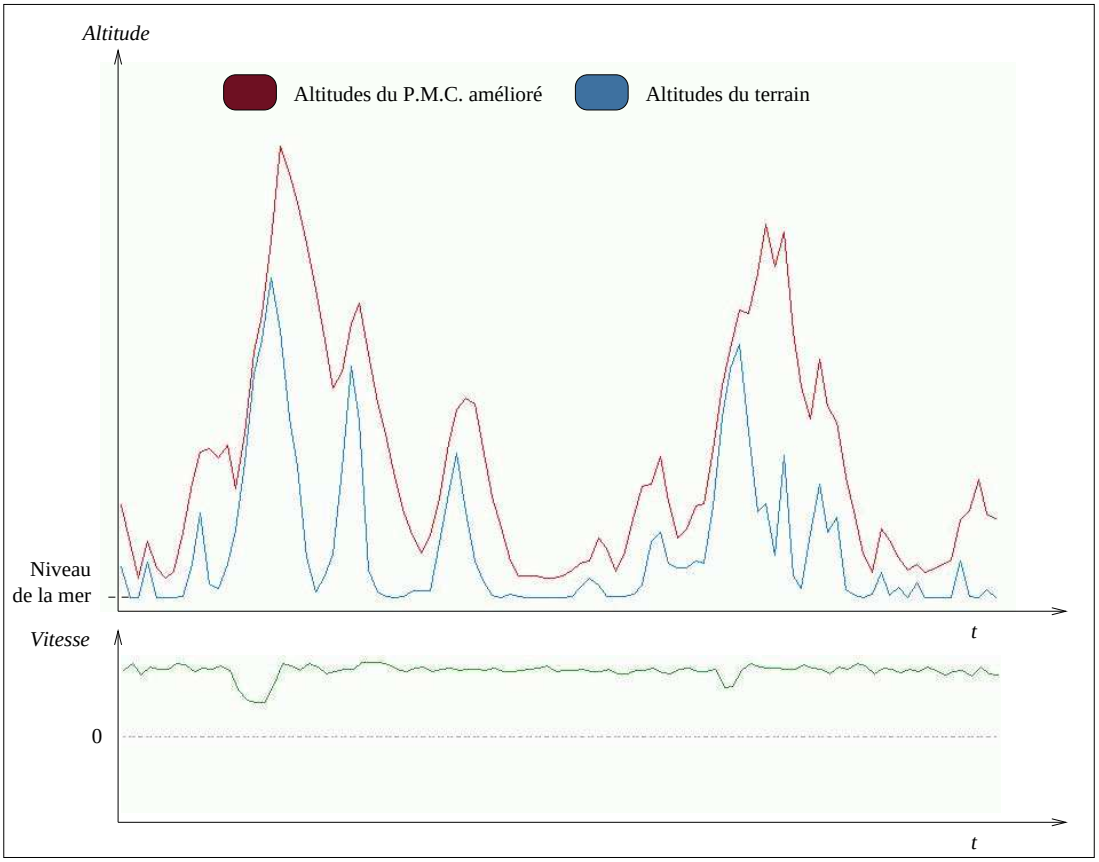


FIG. 4.14 – Courbe des altitudes et de vitesse horizontale du *Percepton Multi-Couches* après la deuxième phase d'apprentissage. La distance par rapport au sol est considérablement réduite, l'hélicoptère effectue un vol en rase-mottes. La vitesse horizontale est globalement constante, aucun retour arrière n'est effectué.

Chapitre 5

Conclusion et perspectives

Nous avons développé un simulateur d'environnement 3D temps réel pour module de pilotage automatique. Notre environnement gère le comportement d'un modèle mathématique simplifié d'hélicoptères. Nos aéronefs sont munis d'un ensemble d'outils de commande pour la navigation et d'instruments de bord et capteurs paramétrables pour la perception de leur environnement. Les commandes ainsi que les perceptions de chaque hélicoptère sont respectivement accèssibles en lecture-écriture et lecture seule. D'une part, le couple «Interface clavier / Écran[1]» en permet un accès pour l'utilisateur humain, puis d'autre part, une librairie de fonctions fait de même pour le cas d'un module de pilotage, ou pilote automatique.

Nos hélicoptères évoluent dans un univers virtuel infini[2] et généré aléatoirement. Cet univers est construit à partir de données topographiques de terrain ; il englobe le relief du sol, la mer et les arbres. Les terrains sont générés aléatoirement par des algorithmes fractals ; les algorithmes utilisés sont : le labyrinthe, le *plasma* et *Brown-Gauss* ; ces deux derniers produisent des paysages montagneux. Nous avons mis en œuvre un interprète de *L-Systèmes* pour la génération de plusieurs types d'arbres ; il sont placés aléatoirement et par regroupement dans le paysage.

Pour les besoins de planification de la trajectoire des véhicules, nous avons développé et intégré à l'interface notre algorithme du *plus sûr chemin*. Cet algorithme

[1] La visualisation sur écran est effectuée par un rendu temps réel en image de synthèse. Nous obtenons sur un calculateur de type — *AMD Athlon*^{tm} XP 1700+, 1 Go de RAM avec une carte graphique *ATI Rage 128 Pro* 32 Mo de mémoire graphique — un rendu supérieur à 60 images par seconde pour approximativement 10 hélicoptères gérés. Nous atteignons les limites d'un rendu temps réel à partir de 50 hélicoptères ; ces limitations sont uniquement dépendantes du calculateur utilisé.

[2] Les cartes d'altitudes sont lus cycliquement ; leur représentation tridimensionnelle est linéaire. Pour un même point de coordonnées (x, z) sur la carte des altitudes de dimensions (L, H), il existe une infinité de coordonnées (x', z'), projections dans l'espace tridimensionnel de (x, z), telles que : $x \equiv x'[L]$ et $z \equiv z'[L]$.

calcule instantanément[3] le chemin qui relie la position du véhicule au point d'arrivée défini par l'utilisateur ; il minimise les altitudes empruntées.

L'implémentation

L'ensemble de notre environnement est réalisée en *ANSI-C* et *OpenGL/Glut* et compilé sous la forme d'une librairie. Il fonctionne donc sur tout système d'exploitation disposant d'un compilateur *C* et des librairies *OpenGL* et *Glut*.
L'utilisation de l'environnement devient aisé ; nous pouvons exclusivement diriger les efforts de programmation sur l'implémentation des pilotes automatiques. Nous donnons un exemple de code source utilisant notre librairie pour généré un paysage de type labyrinthe, y placer des hélicoptères munis de capteurs de distance par défaut et gérer les événements et les interactions. La figure 5.1 montre le résultat obtenu pour le fichier *C* contenant :

```
#include "AmsiEnv3D.h"
int main(int argc, char * argv[]){
   int i, nbHelico = 10 ;
   aEnvSeti(AENV_TERRAIN_TYPE, AENV_LABYRINTH) ; /* Le type de terrain est un labyrinthe */
   aEnvSeti(AENV_TERRAIN_LARGEUR, 500) ; /* La largeur du terrain est égale à 500 ut */
   aEnvSeti(AENV_VISION_LARGEUR, 14) ; /* La largeur du champ de vision est égale à 500 ut */
   aEnvSeti(AENV_NOMBRE_HELICO, nbHelico) ; /* Le nombre d'hélicoptères est fixé à nbHelico */
   for(i = 0 ; i < nbHelico ; i++){
      aEnvSetPiloteFunc(i, 0) ; /* Pour l'hélicoptère i, aucun pilote automatique n'est donné */
      aEnvAddDefaultCapteurs(i) ;/* Ajouter les capteurs par défaut à l'hélicoptère i */
   }
   aEnvStart() ;/* Boucle principale - Gestion de l'affichage, événements et interactions */
   return 0 ;
}
```

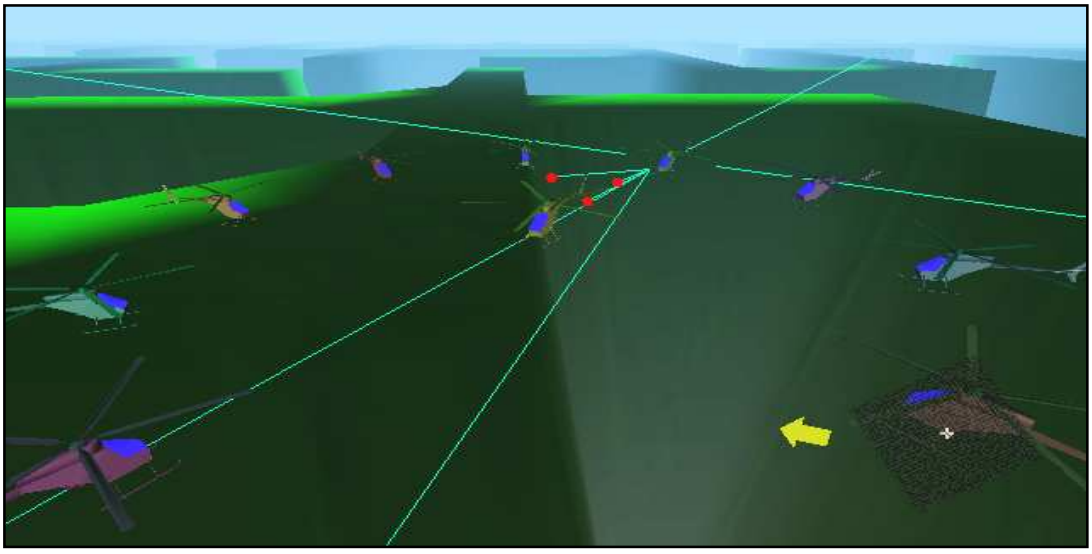


FIG. 5.1 – Rendu d'un environnement de type labyrinthe.

[3]Voir dans la section 4.2.2, les temps de calcul moyens d'un plus *sûr* chemin obtenu par la version améliorée de l'algorithme.

Tests et résultats obtenus

Nous avons réalisé deux implémentations de pilotes automatiques. La première, *l'automate volant*, est un automate déterministe qui possède une dynamique de vol sans risques ; il suit le plus *sûr* chemin en gardant une distance moyenne par rapport aux obstacles (arbres, véhicules, sol). Nous donnons les graphes représentant ses trojectoires horizontales (voir figure 4.7), ses altitudes et vitesse (voir figure 4.8).

Afin de tester l'interface et vérifier sa capacité à gérer un mode d'apprentissage, nous avons choisi d'implémenter notre deuxième pilote automatique sous la forme d'un *Perceptron Multi-Couches*. L'apprentissage du *Perceptron Multi-Couches* a été réalisé sous la supervision de *l'automate volant* et d'un utilisateur via le clavier. Les deux phases d'apprentissage du réseau de neurones sont réspectivement effectuées par ces deux superviseurs, cf. §4.4.2.
Le *Perceptron Multi-Couches* abouti, après sa seconde phase de d'apprentissage, à un comportement plus dynamique (une vitesse soutenue) ; il réalise une meilleur adhérence par rapport au sol.
Nous donnons les tableaux comparatifs (voir tables 4.3 et 4.4) des deux pilotes après chaque phase de l'apprentissage réalisé.

Évolutions souhaitées

Nous projetons à l'avenir de faire évoluer notre environnement vers une solution plus globale. Nous donnons les améliorations prévues à court, moyen et long terme.

Dans un premier temps, nous souhaitons réaliser, pour un type particulier d'hélicoptère, une modélisation mathématique poussée[4]. Puis nous implémenterons une couche réseau pour le passage en mode *client-serveur* ; ceci implique, pour une synchronisation de la génération des cartes d'altitudes, l'écriture d'un générateur de nombres aléatoires. Nous éviterons ainsi les disparités entre les différentes architectures connéctées au *serveur*.

Dans un deuxième temps, nous projetons de définir et d'écrire un protocole de communication *inter-véhicules* pour faire émerger les notions de concurrence et d'entre-aide, la création d'une communité virtuelle de robots. Ces notions impliquent la présence de danger potentiel. Pour cela, nous souhaitons élargir la liste des catégories d'environnements et de véhicules possibles. Nous aurons alors le choix entre des environnements plus ou moins hostiles.

[4]Nous pensons notamment à utiliser des modélisations d'hélicoptères miniatures ; l'accès à leurs données spécifiques est moins restreint [1].

Dans la même optique, relier deux positions de la carte représente une action «*restreinte*» ; nous souhaitons donc élargir la liste des tâches à effectuer ; à l'image du *Concours International Universitaire de Drones Miniature* organisé par la «*Délégation Générale pour l'Armement*»et «*l'Office National d'Études et de Recherches Aérospatiales*».

Enfin, nous présentons quelques perspectives plus générales et dont le détail reste à définir :

- Création d'un modeleur pour véhicule virtuel. Le modeleur doit gérer plusieurs types de dynamiques de mouvement. Nous y intégrerons un module de création et de gestion de capteurs et instruments complexes.
- Écriture d'un langage de description des propriété des véhicules et capteurs créés par le modeleur. Ces propriétés pourraient être améliorées dans le simulateur ; nous souhaitons donner une possiblité d'évolution «*génétique*».
- Donner les moyens, via le réseau *Internet*, aux personnes intéressées par le projet, de développer des pilotes automatiques autonomes et capables d'apprentissage non supervisé. À partir d'un grand nombre d'échantillons, effectuer une «*sélection naturelle*» et faire émerger les meilleurs éléments.

Bibliographie

[1] Christian Munzinger. *Development of a Real-Time Flight Simulator for an Experimental Model Helicopter*. PhD thesis, Georgia Institute of Technology - School of Aerospace Engineering, December 1998.

[2] Pierre Audibert. *Algorithmes et Programmation*. Université de Paris 8, 1995-1997.

[3] Pierre Audibert. *Ordre et Chaos, Fractales, Pavage*. Université de Paris 8, 2000.

[4] Gill Bernard et Jym Feat. Langage naturel et intelligence artificielle. *Stratégies Théoriques, Collection Histoire et Théories Linguistiques 10/1, Presses Universitaires de Vincennes*, 1988.

[5] Michel Buffa, Olivier Faugeras, and Zhengyou Zhang. A stereovision-based navigation system for a mobile robot. Technical report, Unité de recherche INRIA - Sophia Antipolis, avril 1993.

[6] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1 :269–271, 1959.

[7] Stéphane Doncieux. Évolution d'architectures de contrôle pour robots volants. Dans *Intelligence artificielle située : cerveau, corps et environnement*. Hermès, 1999.

[8] G. Dreyfus, J.-M. Martinez, M. Samuelides, M.B. Gordon, F. Badran, S.Thiria et L. Hérault Sous la direction de Gérard Dreyfus. *Réseaux de neurones : Méthodologie et applications*. Eyrolles, 2002.

[9] James D. Foley, Andries van Dom, Steven K. Feiner, and John F. Hughes. *Computer Graphics : principles and practice*. Addison Wesley, second edition, 1996.

[10] Th. Fraichard. Trajectory planning in dynamic workspaces : a 'state-time space' approach. *Submitted to Advanced Robotics*, 1997.

[11] R. Glasius and A. Komoda S. Gielen. Neural network dynamics for path planning and obstacle avoidance. *Neural Networks*, March 1994.

[12] Andrew S. Glassner, editor. *Graphics Gems*. Academic Press, 1990.

[13] Donald O. Hebb. The organization of behavior. *New York : Wiley, Introduction and Chapter 4, «The first stage of perception : growth of the assembly,» pp.xi-xix, 60-78*, 1949.

[14] Jean-Claude Heudin. *La Vie Artificielle*. Hermès, 1994.

[15] K. M. HO and C. J. WANG. Experiments on estimating random mapping. I.C.C.N. International Conference on Computational Nanoscience and Nanotechnology, June 1996.

[16] J.J. Hopfield. Neural networks and physical systems with emergent collective computational abilities. *Proceedings of the National Academy of Sciences 79 : 2554-2558*, 1982.

[17] B.W. Kernighan et D.M. Ritchie. *Le langage C*. Masson, deuxième édition, 1992.

[18] Ron Kimmel, Nahum Kiryati, and Alfred M. Bruckstein. Multi-valued distance maps for motion planning on surfaces with moving obstacles. *IEEE Trans. Robot. & Autom.*, June 1998.

[19] Christopher G. Langton. Aritificial life. In Christopher G. Langton, editor, *Artificial Life volume VI : Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems*. Addison Wesley, 1989.

[20] Jean-Paul Laumond, Florent Lamiraux, Sepanta Sekhavat, Pascal Morin, Claude Samson, Patrick Rives, Michel Devy, Malik Ghallab, Bernard Espiau et Frank Génot sous la direction de Jean-Paul Laumond. *La robotique mobile*. Hermès, 2001.

[21] Cathy A. Lazere and Dennis Elliott Shasha. *Out of Their Minds*. Copernicus Books, 1995.

[22] Charles Lenay, Olivier Gapenne, Sylvain Hanneton et John Stewart. Perception et couplage sensori-moteur : expériences et discussion épistémologique. Dans *Intelligence artificielle située : cerveau, corps et environnement*. Hermès, 1999.

[23] Aristid Lindenmayer and Przemyslaw Prusinkiewicz. Developmental models of multicellular organisms : A computer graphics perspective. In Christopher G. Langton, editor, *Artificial Life volume VI : Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems*. Addison Wesley, 1989.

[24] Benoît Mandelbrot. *Les Objets Fractals*. Flammarion, quatrième édition, 1975-1995.

[25] Woo Mason, Neider Jackie, Davis Tom et Shreiner Dave. *Le guide officiel à l'apprentissage d'OpenGL, version 1.2*. CompusPress France, 1999.

[26] Warren S. McCulloch and Walter Pitts. A logical calculus of the ideas immanent in nervous activity. *Bulletin of Mathematical Biophysics 5 : 115-133*, 1943.

[27] Warren S. McCulloch and Walter Pitts. How we know universals : the perception of auditory and visual forms. *Bulletin of Mathematical Biophysics 9 : 127-147*, 1947.

[28] Marvin L. Minsky and Seymour A. Papert. Perceptrons. 1990.

[29] Michael Negnevitsky. *Artificial Intelligence : A Guide to Intelligent Systems.* Addison Wesley, 2002.

[30] Hansrudi Noser, Pascal Fua et Daniel Thalmann. L-systèmes. Rapport technique, École Polytechnique Fédérale de Lausanne, juin 1997.

[31] Peter Oppenheimer. The artificial menagerie. In Christopher G. Langton, editor, *Artificial Life volume VI : Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems.* Addison Wesley, 1989.

[32] Heinz-Otto Peitgen and Dietmar Saupe. *The Science of Fractal Images.* Springer Verlag, 1988.

[33] Laurence Pelletier. Cartes de visibilité pour la planification de stratégies de déplacement. Rapport technique, INRIA - Sophia Antipolis, 1999.

[34] Przemyslaw Prusinkiewicz and James Hanan. *Lindenmayer Systems, Fractals, and Plants.* Springer Verlag, 1992.

[35] Przemyslaw Prusinkiewicz and Aristid Lindenmayer. *Algorithmic Beauty of Plants.* Springer Verlag, 1990.

[36] Mitchel Resnick. Lego, logo, and life. In Christopher G. Langton, editor, *Artificial Life volume VI : Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems.* Addison Wesley, 1989.

[37] F. Rosenblatt. The perceptron : a probabilistic model for information storage and organization in the brain. *Psychological Review 65 : 386-408*, 1958.

[38] Robert Sedgewick. *Algorithms in C.* Addison-Wesley, third edition, 1998.

[39] O.G. Selfridge. Pandemonium : a paradigm for learning. *Mechanisation of thought Processes : Proceedings of a Symposium Held at the national Physical Laboratory, London : HMSO, pp. 513-526*, November 1958.

[40] Roger T. Stevens. *Advanced Fractal Programming in C.* M&T, 1990.

[41] Michael Travers. Animal construction kits. In Christopher G. Langton, editor, *Artificial Life volume VI : Proceedings of an Interdisciplinary Workshop on the Synthesis and Simulation of Living Systems.* Addison Wesley, 1989.

[42] Christopher D. Watkins and Stephen R. Marenka. *Taking Flight.* M&T Books, 1994.

[43] Harald Wertz. *Introduction aux Objets Fractals.* Rapport Interne, Université de Paris 8, Septembre 1990.

[44] B. Widrow and M. Hoff. Adaptive switching circuits. pages 96–104. IRE WESCON Convention record, 1960.