

Réalisation d'un système de tracé de fractals basé sur le système L

Florent 'Bruce' Revelut

Laurent Casse

18 mars 2002

1 Introduction

Le but de ce projet est de réaliser un programme de création de fractals, et en particulier d'arbres. La sortie devra être un fichier qui contient un objet tridimensionnel, défini selon la syntaxe du ray-traceur disponible en freeware POV-Ray.

1.1 Pov-Ray

POV-Ray est un logiciel open-source disponible sur de nombreuses plateformes, et qui permet de tracer des images de synthèse à partir d'une scène définie par un fichier texte. L'intérêt pour nous de générer un fichier d'include pour ce logiciel est de pouvoir effectuer un tracé de bonne qualité, incluant les gestions éventuelles des ombres, des réflexions et autres effets usuels, sans avoir à se soucier de développer un moteur de rendu indépendant, et même, sans avoir à gérer de fenêtre graphique, ce qui simplifie très nettement le problème de la conception du programme et améliore de manière sensible la portabilité du code, les bibliothèques graphiques étant gérées de manière différente sous Windows et sous Linux, par exemple.

1.2 Modèle utilisé

Nous avons utilisé pour créer nos fractals le système L (du nom de son inventeur, le professeur Lindenmayer, biologiste), qui est basé sur un principe d'auto-réplication. Sans rentrer dans les détails de l'implémentation qui seront discutés ultérieurement, le principe consiste à associer à des caractères des actions, puis créer une chaîne initiale de caractères (ce sera l'axiome), et enfin, utiliser des règles de remplacement des axiomes en répétant un certain nombre de fois. On obtient alors une chaîne de caractères représentant l'objet à dessiner.

Ce modèle a été inventé à l'origine non par un mathématicien, mais par un biologiste qui cherchait un modèle valable pour représenter les arbres réels.

2 Le programme que nous avons créé

Le programme que nous avons créé a été codé en C sous Linux à l'origine. Actuellement, il a été compilé sous plusieurs environnements, dont djgpp (compilateur gratuit) sous Windows 2000 et gcc sous Linux, mais a priori, rien ne s'oppose à ce qu'il se compile sur d'autres plateformes, attendu que le code n'utilise que des bibliothèques standards. D'autre part, le programme est abondamment commenté et suit les règles du C discipliné, c'est à dire que le code suit une indentation cohérente tout du long, n'a pas de transtypage implicite et suit des règles strictes de conventions de nommage des types et des variables.

2.1 Description de l'algorithme utilisé

2.1.1 Description théorique à 2 dimensions

Le système L est basé sur la réplication de motifs. L'algorithme en lui-même est en fait très simple : on choisit une chaîne de départ, qui va contenir des symboles, et ensuite, on choisit tout un ensemble de règles de remplacement, qui vont nous dire à chaque itération par quoi est remplacé chaque symbole, et ensuite, on fait effectuer toutes les substitutions. On répète cette dernière étape le nombre de fois souhaité, et on obtient au final une chaîne de caractères souvent très longue. La deuxième étape va consister à traiter cette chaîne de caractères pour en créer une fractale, qui vu le mode de création, sera forcément auto-similaire. La construction graphique en elle-même est maintenant très simple : on va lire la chaîne de caractères lettre par lettre, et en fonction de chacune d'entre elle, on fera exécuter à une tortue des actions prévues. Les règles de mouvement de la tortue sont les suivantes :

- + : effectue une rotation par rapport à l'axe normal de alpha
- : effectue une rotation par rapport à l'axe normal de -alpha

F : avance et trace une ligne
 G : avance, mais ne trace pas de ligne
 R : recule et trace une ligne
 T : recule, mais ne trace pas de ligne
 & : allonge la branche courante de facteur
 ~ : raccourcit de facteur
 \$: augmente le diamètre
 ^ : réduit le diamètre
 [: enregistre l'état courant dans une pile
] : restaure le dernier état sauvegardé dans la pile
 * : rotation propre de α^2
 / : rotation propre de $-\alpha^2$
 (: rotation orthonormale de α^3
) : rotation orthonormale de $-\alpha^3$

tous les autres caractères ont la même fonction que F

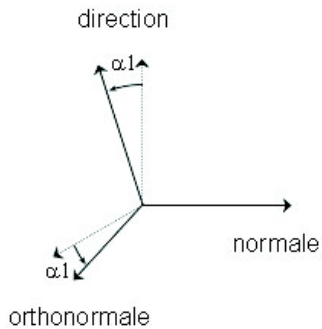
L'utilisation des caractères [et] est très utile, car ils permettent (entre autre) d'explorer complètement une branche avant de revenir au tronc et de continuer comme si on n'avait pas dessiné la branche.

2.1.2 Description théorique à 3 dimensions

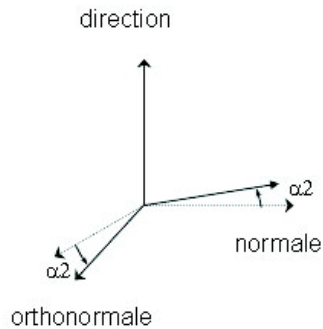
Pour le modèle à trois dimensions, seules les rotations changent. En effet, une branche dessinée dans l'espace a maintenant trois axes principaux de rotation (un triplet orthonormé direct de vecteurs) à partir desquels on peut reconstituer n'importe quelle rotation de l'espace. Le principe théorique reste donc le même que pour le cas de 2 dimensions, avec trois angles au lieu d'un.

Nous allons expliciter un peu plus ces angles de rotations. Une branche est repérée par 4 valeurs : son point de départ (vecteur position à 3 dimensions), sa direction (vecteur direction à 3 dimensions), un vecteur normal (vecteur direction à 3 dimensions tel que le produit scalaire avec la direction est nul) et enfin le diamètre de la branche (représenté par un nombre en virgule flottante). On peut introduire le produit vectoriel entre le vecteur direction et le vecteur normal. On obtient un second vecteur normal au vecteur direction et les trois vecteurs direction associés au point forment un trièdre direct. On définit alors les angles de rotation de la branche autour de ces trois vecteurs de direction. On remarquera par ailleurs que les dessins 3D sont compatibles avec les dessins 2D (il suffit de faire intervenir seulement l'angle de rotation autour de la normale dans l'axiome définissant le dessin), avec comme petite différence que les branches sous 2D sont dessinées avec des cylindres au lieu de simples traits, ce qui leurs donne un aspect 3D quand même.

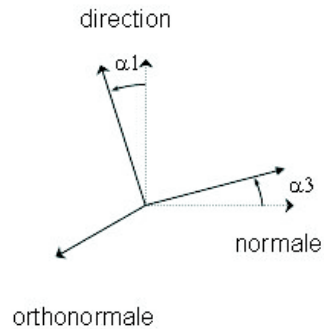
Rotation de α_1 autour de la normale



Rotation de α_2 autour de la direction



Rotation de α_3 autour de l'orthonormale



2.2 Codage

2.3 La bibliothèque de gestion vectorielle

Dans ces fichiers sont définis tous les programmes relatifs à la gestion des vecteurs, comme par exemple le produit vectoriel, la rotation d'un vecteur direction autour d'un autre vecteur, etc... Le listing est donné dans les sous-sections suivantes.

2.3.1 Fichier d'en-tête : vecteur.h

```
/*
Include de calcul vectoriel

Florent "Bruce" Revelut
Supélec 03/2001
*/

#ifndef GESTION_VECTEURS_BRUCE
#define GESTION_VECTEURS_BRUCE

typedef double scalaire;
typedef double Coord;
typedef struct _Position Position;
typedef struct _Direction Direction;

typedef double Angle;
typedef double Norme;

struct _Direction {
    Coord x;
    Coord y;
    Coord z;
};

struct _Position {
    Coord x;
    Coord y;
};
```

```

    Coord z;
};

/* produit vectoriel de 2 vecteurs */
extern Direction prodVect (const Direction d1, const Direction d2);

/* affiche les coordonnées d'un vecteur de type direction */
extern void affiche_Dir (FILE * sort, const Direction d1);

/* idem mais avec position */
extern void affiche_Pos (FILE * sort, const Position d1);

/* produit scalaire de 2 directions */
extern Scalaire prodScal (const Direction d1, const Direction d2);

/* multiplication d'un vecteur par une constante */
extern Direction scale (const Direction d, float s);

/* translation d'une position par ne direction */
extern Position addPosDir (Position p, Direction d);

/* ajout de 2 vecteurs */
extern Direction addDirDir (Direction p, Direction d);

/* rotation de la direction orig autour de l'axe axe d'un angle alpha (en degrés) */
extern Direction rotate (Direction orig, Direction axe, Angle alpha);

/* normalise un vecteur */
/* La fonction teste si le vecteur n'est pas nul */
extern Direction normalize (const Direction vect);

/* Projette le vecteur 1 sur le vecteur 2 */
extern Direction projette (const Direction vect, const Direction vect2);

/* Compare 2 vecteurs */
/* La comparaison est faite à EPSILON près */
extern bool compVect (const Direction v1, const Direction v2);

/* Calcule la norme du vecteur d */
extern Norme norme (const Direction d);

/* initialisation de vecteurs */
extern Position initPos (const Coord x, const Coord y, const Coord z);
extern Direction initDir (const Coord x, const Coord y, const Coord z);

/* prend resp le maximum et le minimum de chaque composante */
extern Position vectMax (Position p1, Position p2);
extern Position vectMin (Position p1, Position p2);
#endif

```

2.3.2 Code : vecteur.c

```

/*
Include de calcul vectoriel

Florent "Bruce" Revelut
Supélec 03/2001
*/

#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include "bool.h"
#include "vecteur.h"

#define EPSILON 1E-10
#define PI M_PI
#define RADIANS(x) PI*x/180

#define MAX(a,b) ((a>b)?a:b)
#define MIN(a,b) ((a>b)?b:a)

static const Direction vectNul={0,0,0};

/* Déclaration des fonctions vectorielles */

Direction prodVect (const Direction d1, const Direction d2) {
    Direction d3;

    d3.x = d1.y * d2.z - d1.z * d2.y;
    d3.y = d1.z * d2.x - d1.x * d2.z;
    d3.z = d1.x * d2.y - d1.y * d2.x;

    return d3;
}

void affiche_Dir (FILE * sort, const Direction d1){
    fprintf (sort, "< %f, %f, %f >", d1.x, d1.y, d1.z);
}

```

```

void affiche_Pos (FILE *sort, const Position d1){
    fprintf ( sort, "<math>\langle f, f, f \rangle</math>", d1.x, d1.y, d1.z);
}

Scalaire prodScal (const Direction d1, const Direction d2) {
    Scalaire res;

    res = (Scalaire) (d1.x*d2.x + d1.y*d2.y + d1.z*d2.z);

    return res;
}

Direction scale (const Direction d, float s) {
    Direction res;

    res.x = d.x*s;
    res.y = d.y*s;
    res.z = d.z*s;

    return res;
}

Position addPosDir (Position p, Direction d) {
    Position res;

    res.x = p.x + d.x;
    res.y = p.y + d.y;
    res.z = p.z + d.z;

    return res;
}

Direction addDirDir (Direction p, Direction d) {
    Direction res;

    res.x = p.x + d.x;
    res.y = p.y + d.y;
    res.z = p.z + d.z;

    return res;
}

bool compVect (const Direction v1, const Direction v2) {
    return ((fabs(v1.x - v2.x)<EPSILON)&&(fabs(v1.y - v2.y)<EPSILON)&&(fabs(v1.z - v2.z)<EPSILON));
}

Direction normalize (const Direction vect){
    if (compVect (vect, vectNul)) {
        fprintf (stderr, "Impossible de normaliser le vecteur nul!\n");
        exit (EXIT_FAILURE);
    }
    return scale (vect, 1/sqrt(prodScal(vect, vect)));
}

/* projette vect 1 sur vect 2 */
Direction projette (const Direction vect, const Direction vect2) {
    Direction s;

    s=normalize (vect2);
    return scale (s, prodScal (s, vect));
}

/* L'angle de rotation DOIT etre exprime en degres */
Direction rotate (Direction orig, Direction axe, Angle alpha) {
    Direction s1, s2, s_axe;
    Direction p_axe, p_norm;
    double sin_a, cos_a;
    Scalaire norm;

    s_axe=normalize (axe);
    s1=prodVect (s_axe, orig);
    if (compVect (s1, vectNul)) {return orig;}
    s1 = normalize (s1);
    s2 = normalize (prodVect (s1, s_axe));

    /* printf ("angle %f\n", alpha); */
    sin_a=sin(PI*alpha/180);
    cos_a=cos(PI*alpha/180);
    p_axe=projette (orig, s_axe);
    p_norm=projette (orig, s2);
    norm=sqrt (prodScal (p_norm, p_norm));

    /* return addDirDir (addDirDir (scale (s1, norm*cos_a), scale (s2, norm*sin_a)), p_axe); */
    return addDirDir (addDirDir (scale (s1, norm*sin_a), scale (s2, norm*cos_a)), p_axe);
}

Norme norme (const Direction d) {

```

```

        return sqrt (prodScal (d,d));
    }

Position initPos (const Coord x,const Coord y,const Coord z) {
    Position p;

    p.x = x;
    p.y = y;
    p.z = z;
    return p;
}

Direction initDir (const Coord x,const Coord y,const Coord z) {
    Direction p;

    p.x = x;
    p.y = y;
    p.z = z;
    return p;
}

Position vectMax (Position p1, Position p2) {
    Position res;

    res.x = MAX(p1.x,p2.x);
    res.y = MAX(p1.y,p2.y);
    res.z = MAX(p1.z,p2.z);

    return res;
}

Position vectMin (Position p1, Position p2) {
    Position res;

    res.x = MIN(p1.x,p2.x);
    res.y = MIN(p1.y,p2.y);
    res.z = MIN(p1.z,p2.z);

    return res;
}
/* fin des fonctions vectorielles */

```

2.4 Le programme principal

On trouve ici la programmation proprement dite de l'algorithme du système L, ainsi que les gestions de l'ouverture du fichier d'entrée et de l'écriture du fichier include de POV. D'autre part, une quantité d'informations non négligeable est directement sortie à l'écran.

```

/*
Programme de tracé d'arbres en utilisant le système L
à l'attention du cours "systèmes fractals"

F. 'Bruce' Revelut - L. 'Superman' Casse
Supélec 2001
*/

/* mode debugage ou final?*/
/* #define DEBUG */

/* includes standards */
#include <stdio.h>
#include <stdlib.h>
/* utilisation de string.h et ctype.h, qui vont servir pour analyser le fichier de depart */
#include <string.h>
#include <ctype.h>
/* utilisé pour le calcul d'une valeur absolue */
#include <math.h>

/* Include persos */
/* gestions des booleens */
#include "bool.h"
/* gestion des vecteurs */
#include "vecteur.h"

/* permet d'utiliser le fichier de debugage, qui contient quelques fonctions qui
ne sont pas utiles dans la version finale du programme */
#ifdef DEBUG
#include "debug.h"
#endif

/* Declaration des constantes */
/* Longueur d'une chaîne de caractères*/
#define LONG_CHAINE 80
/* Nombre de lettres de l'alphabet */
#define NB_LETTRES_AL 26

/* declaration des constantes de CLEF */
#define CLEF_NOT_FOUND 255
#define NIVEAU 1

```

```

#define NIVEAU_S "NIVEAU"
#define VECTEUR1 2
#define VECTEUR1_S "ORIGINE"
#define VECTEUR2 3
#define VECTEUR2_S "DIRECTION"
#define VECTEUR3 4
#define VECTEUR3_S "NORMALE"
#define QUALITE 5
#define QUALITE_S "QUALITE"
#define AXIOME 6
#define AXIOME_S "AXIOME"
#define ITERATIONS 7
#define ITERATIONS_S "ITERATIONS"
#define ANGLE1 8
#define ANGLE1_S "ANGLE1"
#define ANGLE2 9
#define ANGLE2_S "ANGLE2"
#define ANGLE3 10
#define ANGLE3_S "ANGLE3"
#define FACT_ALL 11
#define FACT_ALL_S "FACTEUR_ALLONGEMENT"
#define FACT_DIL 12
#define FACT_DIL_S "FACTEUR_DILATATION"
#define DIAMETRE 13
#define DIAMETRE_S "DIAMETRE"

/* declaration des règles standards */
/* tous les caractères de cette chaîne de caractères sont les caractères spéciaux, qui ont
un rôle dans la génération du système L. leur loi d'évolution est simple, ils sont remplacés
par eux-mêmes. On ne peut donc pas les redéfinir */
#define REGLES_STD "+-[]()*&~^$"

/* Définition des types utilisés */
typedef char Regle[LONG_CHAINE];

/* Diamètre d'un objet */
typedef double Diametre;

/* Nature de l'instruction qui sera passée */
/* C'est le type des constantes définies plus haut */
typedef short Nature_Instr;

/* Qualité de la sortie */
/* peut prendre les valeurs 1 - 2 ou 3 */
typedef unsigned int Qualite;

/* type servant uniquement à compter les itérations ou à fixer la limite */
typedef unsigned int Nb_Iterations;

/* structure de la variable init, qui contiendra toutes les informations
de base pour lancer la moulinette */
typedef struct _Init Init;

struct _Init {
    Position depart;
    Direction direction;
    Direction normale;
    Qualite qualite;
    Diametre diametre;
    Angle angle1;
    Angle angle2;
    Angle angle3;
    double facteur_allongement;
    double facteur_dilatation;
    Nb_Iterations iterations;
    char axiome[LONG_CHAINE];
};

/* type de l'enregistrement servant à stocker les noms des fichiers d'entrée et de sortie */
typedef struct {
    char ouverture[LONG_CHAINE];
    char sortie[LONG_CHAINE];
} Fichiers ;

/* constantes servant à l'initialisation de la moulinette */
/* avant de lire les directives stockées dans le fichier d'entrée */
#define DEF_POS {0,0,0}
#define DEF_DIR {0,1,0}
#define DEF_NOR {0,0,1}
#define DEF_QUAL 0
#define DEF_DIAM .2
#define DEF_IT 2
#define DEF_AX "A"
#define DEF_ANGLE 60
#define DEF_SCAL 1.5

/* Pile de règles */

```

```

/* Va servir à stocker les différentes règle d'évolution, sous forme d'un pile */
typedef struct _Regle *ListRegle;

struct _Regle {
    char indicateur;
    Regle courante;
    ListRegle suivante;
};

/* Pile d'états */
/* Va servir à stocker puis restaurer les états (position, direction, normale, diametre)
lors des appels '[' et ']' dans les lois d'évolution */
typedef struct _Pile *Pile;

struct _Pile {
    /* les donnees de la pile */
    Position pos;
    Direction dir;
    Direction norm;
    Diametre diam;
    /* et le pointeur vers la position suivante */
    Pile suivant;
};

/* fin de la définition des types */

/* Debut des fonctions de pile */

/* initialisation d'une pile d'etats */
/* fonction actuellement non utilisée! */
/* Idéalement, ce serait mieux de la modifier pour qu'elle rende une pile vide */
static Pile initialize_Pile (const Position pos, const Direction dl, const Direction norm, const Diametre diam) {
    Pile res;

    res = malloc (sizeof(*res));
    if ( res == NULL ) {
        fprintf (stderr, "Erreur_dans_initialize_pile_\n");
        fprintf (stderr, "impossible_d'allouer_la_memoire!\n");
        exit (EXIT_FAILURE);
    } else
    {
        res->pos = pos;
        res->dir = dl;
        res->norm = norm;
        res->diam = diam;
        res->suivant = NULL;
    }
    return res;
}

/* ajout d'une position courante */
/* dans la pile précisée */
/* La syntaxe est de la forme pile = push (position, direction, normale, diametre, pile); */
static Pile push (const Position pos, const Direction dl, const Direction norm, const Diametre diam, Pile p) {
    Pile res;

    res = malloc (sizeof(*res));
    if ( res == NULL ) {
        fprintf (stderr, "Erreur_dans_push_\n");
        fprintf (stderr, "impossible_d'allouer_la_memoire!\n");
        exit (EXIT_FAILURE);
    } else
    {
        res->pos = pos;
        res->dir = dl;
        res->norm = norm;
        res->diam = diam;
        res->suivant = p;
    }
    return res;
}

/* on supprime la tete de pile */
/* dans la pile précisée */
/* La syntaxe est de la forme pile = pop (pile); */
static Pile pop (Pile p) {
    Pile res;

    if (p == NULL) {
        fprintf (stderr, "La_pile_est_vide::impossible_de_faire_un_pop!\n");
        exit (EXIT_FAILURE);
    } else {
        res = p->suivant;
        free (p);
    }
    return res;
}

/* Fin des fonctions de pile */

```

```

/* Debut des fonctions du système L */

/* test de l'existence de la regle */
/* vérifie que la règle d'évolution du caractère a existe, et rend un booléen */
/* syntaxe if (existe ('A',liste_regle) {} */
static bool existe (const char a,ListRegle list) {
    if (list==NULL) {
        return FALSE;
    } else {
        return ( (a==(list->indicateur)) || (existe (a, list->suivante)) );
    }
}

/* permet d'extraire de la pile de règles la loi d'évolution du monôme a
si s est une chaîne de caractères, la syntaxe est la suivante
extrait (liste_regles,'A',s); */
static void extrait (const ListRegle list,const char a,char b[]) {
    if (list==NULL) {
        fprintf (stderr,"impossible_d'extraire_la_regle_%c\n",a);
        fprintf (stderr,"Arret_du_programme!\n");
        exit (EXIT_FAILURE);
    } else {
        if (a==(list->indicateur)) {
            strcpy (b,list->courante);
            return;
        } else {
            extrait (list->suivante,a,b);
        }
    }
}

/* Permet d'ajouter une règle d'évolution à une liste de regles */
/* syntaxe liste = ajoute ('A',"ABA",liste); */
static ListRegle ajoute (const char a,const char b[],ListRegle list) {
    ListRegle l;

    l=(ListRegle) malloc (sizeof(*l));
    if (l==NULL) {
        fprintf (stderr,"default_d'allocation_de_memoire_\n");
        fprintf (stderr,"erreur_produite_dans_ajoute_une_regle\n");
        exit (EXIT_FAILURE);
    }
    l->indicateur = a;
    l->suivante = list;
    strcpy (l->courante,b);
    return l;
}

/* Permet de détruire intégralement une liste de règles, afin de libérer
la mémoire qu'elle occupait.
Syntaxe : detruit (&liste_regle);
*/
static void detruit (ListRegle *l) {
    ListRegle l3;

    if (*l == NULL) {
        return;
    }
    l3=*l;
    *l=l3->suivante;
    free (l3);
    detruit (l);
}

/* fonction servant à la mise en forme des chaînes brutes qui sont lues */
/* Le résultat va être un formatage qui consiste à :
- supprimer les espaces en trop
- convertir en majuscules
- supprimer les commentaires

Syntaxe, si s est une chaîne de caractères disponible
nettoie ("La chaîne à nettoyer",s);
*/
static void nettoie (const char a[],char b[]) {
    unsigned long l;
    unsigned long i;
    unsigned long j=0;
    bool espace=FALSE;

    l=strlen(a);

    for (i=0;i<l;i++) {
        char c;

        c=a[i];
        switch (c) {
            /* si on arrive sur un caractère LF (10) ou CR (13), ou sur un commentaire
on arrête le traitement de cette ligne là */
            case '\r' : /*~ NoBreak */

```

```

        case '\n' : /*~ NoBreak */
        case '#' : /*~ NoBreak */
        case '%' : b[j]='\0';j++;break;
/* on supprime tous les espaces de début de ligne
dans la suite on remplace les espaces multiples par un seul espace
le caractère <TABULATION> est considéré comme un espace */
        case '\t' : /*~NoBreak */
        case ' ' :
                if (!espace) {
                        if (j != 0) {
                                espace = TRUE;
                                b[j]=' ';
                                j++;
                        }
                };
                break;
/* Pour toutes les autres lettres, on les passe en majuscules */
        default : b[j] = toupper(c);j++;espace=FALSE;
    }
    b[j]='\0';
}

/* fonction qui sert à extraire de la ligne formatée a le premier mot
(ie le premier groupe de caractères jusqu'à un espace*/
/* Syntaxe : avec a une chaine formatée et s une chaine libre
lis_mot (a,b);
*/
static void lis_mot (const char a[],char b[]) {
    unsigned int i;

    for (i=0;(a[i] != ' ')||(i==strlen(a));i++) {
        b[i]=a[i];
    }
    b[i]=a[i];
    b[i]='\0';
}

/* Analyse la nature du mot passé en paramètre, et rend une constante numérique correspondant
à ce qui a été détecté, afin de faciliter un ultérieur filtrage */
static Nature_Instr analyse_clef (const char a[]) {
    if (strcmp (a,NIVEAU_S)==0) {return NIVEAU;}
    if (strcmp (a,VECTEUR1_S)==0) {return VECTEUR1;}
    if (strcmp (a,VECTEUR2_S)==0) {return VECTEUR2;}
    if (strcmp (a,VECTEUR3_S)==0) {return VECTEUR3;}
    if (strcmp (a,VECTEUR3_S)==0) {return VECTEUR3;}
    if (strcmp (a,QUALITE_S)==0) {return QUALITE;}
    if (strcmp (a,AXIOME_S)==0) {return AXIOME;}
    if (strcmp (a,ANGLE1_S)==0) {return ANGLE1;}
    if (strcmp (a,ANGLE2_S)==0) {return ANGLE2;}
    if (strcmp (a,ANGLE3_S)==0) {return ANGLE3;}
    if (strcmp (a,FACT_ALL_S)==0) {return FACT_ALL;}
    if (strcmp (a,FACT_DIL_S)==0) {return FACT_DIL;}
    if (strcmp (a,DIAMETRE_S)==0) {return DIAMETRE;}
    if (strcmp (a,ITERATIONS_S)==0) {return ITERATIONS;}
    return CLEF_NOT_FOUND;
}

/* Extrait d'une chaine de caractères formatée le 2e mot
ie le deuxième groupe de caractères, les espaces étant les seuls séparateurs */
static void lis_param_chaine (const char a[],char param[]) {
    unsigned int i;
    unsigned int j=0;

    for (i=0;(a[i]!=' ')&&(i!=strlen (a));i++) {}
    if ( (i==strlen (a)) || ((i+1)==strlen(a)) ) {
        fprintf (stderr,"Pas_de_paramètres_de_donnés_sur_un_axiome_à_paramètres_type_1\n");
        fprintf (stderr,"zone_fautive_:%s\n",a);
        exit (EXIT_FAILURE);
    }
    i++;
    for (;(a[i]!=' ')&&(i!=strlen (a));i++,j++) {
        param [j]=a[i];
    }
    param [j] = '\0';

    if ((i == strlen (a))||((i+1) == strlen (a))) {return;}
    fprintf (stderr,"Trop_de_paramètres_de_donnés_sur_un_axiome_à_paramètres_type_2\n");
    fprintf (stderr,"zone_fautive_:%s\n",a);
    exit (EXIT_FAILURE);
}

/* permet de ressortir les 3 coordonnées d'un vecteur passé sous forme de chaine
de caractères.
Syntaxe ; analyse_vect (&x,&y,&z,"<12,2.1,-3>");
un vecteur est défini par des <>, les séparateurs de composantes étant les ',' ou les ';' */
static void analyse_vect (Coord *x,Coord *y,Coord *z,const char a[]) {
    unsigned int i=0;
    unsigned int j;
    char b[LONG_CHAINE];

```

```

/* test du debut du vect */
if (a[i]!='<') {
    fprintf (stderr, "Vecteur_attendu_de_vrait_commencer_par_<n");
    fprintf (stderr, "chaîne_presente:_%s\n", a);
    exit (EXIT_FAILURE);
}
for (i++,j=0;(isdigit (a[i])||(a[i]=='.')||(a[i]=='-'));i++,j++) {
    b[j]=a[i];
}
b[j]='\0';
*x=atof(b);

/* test de la 1re separation */
if ((a[i]!='.')&&(a[i]!='-')) {
    fprintf (stderr, "Vecteur_attendu_séparateur_ou_(1)\n");
    fprintf (stderr, "chaîne_presente:_%s\n", a);
    exit (EXIT_FAILURE);
}
for (i++,j=0;(isdigit (a[i])||(a[i]=='.')||(a[i]=='-'));i++,j++) {
    b[j]=a[i];
}
b[j]='\0';
*y=atof(b);

/* test de la 2de intersection */
if ((a[i]!='.')&&(a[i]!='-')) {
    fprintf (stderr, "Vecteur_attendu_séparateur_ou_(2)\n");
    fprintf (stderr, "chaîne_presente:_%s\n", a);
    exit (EXIT_FAILURE);
}
for (i++,j=0;(isdigit (a[i])||(a[i]=='.')||(a[i]=='-'));i++,j++) {
    b[j]=a[i];
}
b[j]='\0';
*z=atof(b);
if (a[i]!='>') {
    fprintf (stderr, "Vecteur_attendu_de_vrait_finir_par_>n");
    fprintf (stderr, "chaîne_presente:_%s\n", a);
    exit (EXIT_FAILURE);
}
return;
}

/* extrait d'une chaîne de caractères un vecteur position */
/* Utilise analyse_vect et lis_param_chaine */
static void lis_param_pos (const char a[], Position *p) {
    char b[LONG_CHAINE];
    Coord x, y, z;

    lis_param_chaine (a, b);
    analyse_vect (&x, &y, &z, b);
    p->x=x;
    p->y=y;
    p->z=z;
}

/* extrait d'une chaîne de caractères un vecteur direction */
/* Utilise analyse_vect et lis_param_chaine */
static void lis_param_dir (const char a[], Direction *p) {
    char b[LONG_CHAINE];
    Coord x, y, z;

    lis_param_chaine (a, b);
    analyse_vect (&x, &y, &z, b);
    p->x=x;
    p->y=y;
    p->z=z;
}

/* extrait d'une chaîne de caractères une Qualite */
/* Utilise lis_param_chaine */
static void lis_param_qual (const char a[], Qualite *pos) {
    Qualite p;
    char b[LONG_CHAINE];
    Coord x, y, z;

    lis_param_chaine (a, b);
    p=(Qualite) atoi(b);
    *pos = p;
}

/* extrait d'une chaîne de caractères un nombre d'itérations */
/* Utilise lis_param_chaine */
static void lis_param_nb (const char a[], Nb_Iterations *pos) {
    Nb_Iterations p;
    char b[LONG_CHAINE];

```

```

        lis_param_chaine (a,b);
        p=(Nb_Itérations) atoi(b);
        *pos = p;
    }

/* extrait d'une chaîne de caractères un nombre en virgule flottante */
/* Utilise lis_param_chaine */
static void lis_param_float (const char a [], double *pos) {
    float p;
    char b[LONG_CHAINE];

    lis_param_chaine (a,b);
    p=atof(b);
    *pos = p;
}

/* s'applique à une ligne contenant une directive déjà identifiée*/
/* permet de lire le paramètre de la directive, et de modifier la variable de config */
/*
    vieux : ligne d'origine, telle qu'elle a été lue dans le fichier
    a : chaîne nettoyée
    mot : motif qui a été reconnue comme correspondant à une directive
        (mais qui n'en est pas forcément une)
    n : nature de l'instruction, définie en utilisant la fonction analyse_clef
    *config : pointeur sur la variable à modifier
*/
static void modif_config (const char vieux [], const char mot [], const char a [],
    Nature_Instr n, Init *config) {
    char param[LONG_CHAINE];
    Direction dir;
    Position pos;
    Qualite qual;
    Nb_Itérations nb;
    Angle angle;
    double fac;

    switch (n) {
        case CLEF_NOT_FOUND :
            fprintf (stderr, "Ligne:_%s", vieux);
            fprintf (stderr, "Motif:_%s_non_reconnu\n", mot);
            exit (EXIT_FAILURE);
        case AXIOME :
            lis_param_chaine (a, param);
            fprintf (stdout, "inscription_de_l'axiome_%s\n", param);
            strcpy (config->axiome, param);
            break;
        case VECTEUR1 :
            lis_param_pos (a, &pos);
            fprintf (stdout, "Nouvelle_origine:_%s", pos);
            affiche_Pos (stdout, pos);
            fprintf (stdout, "\n");
            config->depart=pos;
            break;
        case VECTEUR2 :
            lis_param_dir (a, &dir);
            config->direction=dir;
            fprintf (stdout, "Nouvelle_direction:_%s", dir);
            affiche_Dir (stdout, dir);
            fprintf (stdout, "\n");
            break;
        case VECTEUR3 :
            lis_param_dir (a, &dir);
            fprintf (stdout, "Nouvelle_normale:_%s", dir);
            affiche_Dir (stdout, dir);
            fprintf (stdout, "\n");
            config->normale=dir;
            break;
        case QUALITE :
            lis_param_qual (a, &qual);
            fprintf (stdout, "Niveau_de_qualité_defini_à_%u\n", qual);
            config->qualite = qual;
            break;
        case NIVEAU : /*~NoBreak */
        case ITERATIONS :
            lis_param_nb (a, &nb);
            fprintf (stdout, "Niveau_de_reccurrence_defini_à_%u\n", nb);
            config->iterations = nb;
            break;
        case ANGLE1 :
            lis_param_float (a, &angle);
            fprintf (stdout, "Angle_de_rotation_defini_à_%f\n", angle);
            config->angle1 = angle;
            break;
        case ANGLE2 :
            lis_param_float (a, &angle);
            fprintf (stdout, "Angle_de_rotation_defini_à_%f\n", angle);
            config->angle2 = angle;
            break;
        case ANGLE3 :
            lis_param_float (a, &angle);

```

```

        fprintf (stdout, "Angle_de_rotation_défini_à%f\n", angle);
        config->angle3 = angle;
        break;
    case FACT_ALL :
        lis_param_float (a,&fac);
        fprintf (stdout, "Facteur_d'allongement_défini_à%f\n", fac);
        config->facteur_allongement = fac;
        break;
    case FACT_DIL :
        lis_param_float (a,&fac);
        fprintf (stdout, "Facteur_de_dilatation_défini_à%f\n", fac);
        config->facteur_dilatation = fac;
        break;
    case DIAMETRE :
        lis_param_float (a,&fac);
        fprintf (stdout, "diametre_défini_à%f\n", fac);
        config->diametre = (Diametre) fac;
        break;
/* Normalement, tous les cas possibles ont été testés
si on arrive dans ce cas là, il s'agit d'une défaillance du code! */
    default :
        fprintf (stderr, "Cas_normalement_inaccessible_-_défaillance_du_code\n");
        fprintf (stderr, "Code_de_mot:_%i\n", n);
        fprintf (stderr, "Ligne_||%s", vieux);
        fprintf (stderr, "motif:_%s\n", mot);
        exit (EXIT_FAILURE);
}

/* même fonction que la ligne précédente, mais s'appliquant à une ligne qui
correspond à une règle d'évolution */
static void modif_regle (const char vieux [], const char mot [], const char a [], ListRegle *regle) {
    char c[LONG_CHAINE];

    if (strlen (a) != 1) {
        fprintf (stderr, "Le_motif_doit_être_de_longueur_1\n");
        fprintf (stderr, "Motif_trop_long_(%s)_dans_%s\n", a, mot);
        fprintf (stderr, "Erreur_sur_la_ligne_||%s\n", vieux);
        exit (EXIT_FAILURE);
    }
    if (!isalpha(a[0])) {
        fprintf (stderr, "Le_motif_doit_être_une_lettre\n");
        fprintf (stderr, "Motif_errone_(%s)_dans_%s\n", a, mot);
        fprintf (stderr, "Erreur_sur_la_ligne_||%s\n", vieux);
        exit (EXIT_FAILURE);
    }
    if (existe (a[0], *regle)) {
        fprintf (stderr, "Double_définition_de_la_regle_%c\n", a[0]);
        fprintf (stderr, "Erreur_fatale!\n");
        exit (EXIT_FAILURE);
    }

    lis_param_chaine (mot, c);
    *regle = ajoute (a[0], c, *regle);
    fprintf (stdout, "Motif_%c=>%s", a[0], c);
}

/* Fonction qui permet d'ajouter les règles d'évolution standards,
donc celles qui correspondent aux motifs standards du système L, tels
que définis dans la documentation et dans la chaîne REGLES_STD */
static void add_regles_standards (ListRegle *l) {
    char s[LONG_CHAINE];
    unsigned int i;

    strcpy (s, REGLES_STD);
    for (i=0; i<strlen (s); i++) {
        char ss[3];

        ss[0]=s[i];
        ss[1]='\0';
        *l = ajoute (s[i], ss, *l);
    }
}

/* Analyse une chaîne de caractères nettoyée pour savoir s'il s'agit d'une règle ou d'une
directive */
/* il faut passer la chaîne non nettoyée, pour pouvoir l'afficher en cas de problème */
static void traite (const char vieux [], const char a [], Init *config, ListRegle *regle) {
    if (strcmp (a, "") == 0) {return;}
    if (a[1] == ' ') {
        char mot [LONG_CHAINE];

        /* traitement d'une règle d'évolution */
        printf ("traitement d'une règle\n");
        lis_mot (a, mot);

        printf ("\n");

        modif_regle (vieux, a, mot, regle);
    }
}

```

```

    } else {
        /* traitement d'une directive */
        char mot [LONG_CHAINE];
        Nature_Instr ind_mot;

        lis_mot (a,mot);
        ind_mot = analyse_clef (mot);
        modif_config (vieux,mot,a,ind_mot,config);
    }
}

/* Lecture du fichier de définition */
/* lis chaque ligne, la nettoie, et l'envoie à l'analyse */
static void charge (const char fichier [], Init *header, ListRegle *liste) {
    typedef FILE *Flot;
    /* stream de sortie */
    Flot flot;
    /* ligne courante brute de lecture */
    char info [LONG_CHAINE];
    /* ligne courante nettoyée */
    char info2 [LONG_CHAINE];
    /* ligne précédente nettoyée*/
    char info3 [LONG_CHAINE];

    /* ouverture du fichier de sauvegarde */
    if ((flot = fopen (fichier, "r")) == NULL) {
        fprintf (stderr, "Impossible_d'ouvrir_le_fichier!\n");
        exit (EXIT_FAILURE);
    }

    strcpy (info3, "");

    /* on lit toutes les lignes du fichier une par une */
    while (!feof (flot)) {
        if ( fgets (info, LONG_CHAINE, flot) == NULL) {
            if (!feof (flot)) {
                fprintf (stderr, "Erreur_de_lecture_pendant_le_chargement!\n");
                exit (EXIT_FAILURE);
            }
            nettoie (info, info2);
        }
        /* j'ai été obligé de rajouter ce test pmour ne pas planter en fin de fichiers, quand
le fichier d'entrée se termine sur la règle, sans aucun caractère derrière */
        /* visiblement, le EOF n'était pas détecté */
        if (strcmp (info2, info3) != 0) {
            traite (info, info2, header, liste);
        }

        /* on sauvegarde l'ancienne chaine et on recommence */
        strcpy (info3, info2);
    }

    /* Enfin, fermeture du fichier*/
    if (fclose (flot) != 0) {
        fprintf (stderr, "BEEEP_:_ERREUR_DE_CLOTURE\n");
        exit (EXIT_FAILURE);
    }
}

/* permet d'afficher sur le flot de sortie sélectionné un résumé de toutes les
directives de traitement qui seront utilisées */
static void resume_conf (FILE* sortie, Init const * conf) {
    fprintf (sortie, "\n\nResume_de_la_config\n\n");
    fprintf (sortie, "Position_de_depart_");
    affiche_Pos (sortie, conf->depart);
    fprintf (sortie, "\nDirection_principale_");
    affiche_Dir (sortie, conf->direction);
    fprintf (sortie, "\nDirection_normale_");
    affiche_Dir (sortie, conf->normale);
    fprintf (sortie, "\nQualite_%u\n", conf->qualite);
    fprintf (sortie, "Nombre_d'iterations_:%u\n", conf->iterations);
    fprintf (sortie, "Axiome_de_depart_:%s\n", conf->axiome);
    fprintf (sortie, "Angle_de_rotation_alpha1_:%f\n", conf->angle1);
    fprintf (sortie, "Angle_de_rotation_alpha2_:%f\n", conf->angle2);
    fprintf (sortie, "Angle_de_rotation_alpha3_:%f\n", conf->angle3);
    fprintf (sortie, "Coefficient_d'allongement_:%f\n", conf->facteur_allongement);
    fprintf (sortie, "Coefficient_de_dilatation_:%f\n", conf->facteur_dilatation);
    fprintf (sortie, "Diametre_d'origine_:%f\n", conf->diametre);
    fprintf (sortie, "\n");
}

/* permet de mettre à jour un tableau de booléens */
/* le fichier comporte 26 booléens, correspondant aux 26 lettres de l'alphabet */
/* si une lettre est présente dans les motifs d'évolution, on met à TRUE le booléen
à la bonne place (exemple, si on a une loi A->ABD, on va mettre à vrai les cases 0,1 et 3 */
static void ajoute_lettres_regle (ListRegle l, bool let []) {
    unsigned int i;
    char s [LONG_CHAINE];

```

```

    if (l==NULL) {return;}
    ajoute_lettres_regle (l->suivante, let);
    strcpy (s, l->courante);
    for (i=0; i<strlen (s); i++) {
        let [((int) (s[i])) - ((int) 'A')] = TRUE;
    }
}

/* Teste si les règles d'évolution sont cohérentes, donc si toutes les lettres utilisées
dans les motifs ont elles aussi des règles d'évolution */
static bool incoherence (FILE* sortie, ListRegle l) {
    static bool lettres [NB_LETTRES_AL];
    int i;
    bool res=FALSE;

/* initialisation du tableau de booléens */
    for (i=0; i<NB_LETTRES_AL; i++) {
        lettres [i]=FALSE;
    }

/* met à vrai les booléens correspondant à des lettres apparaissant */
/* utilise la fonction ajoute_lettres_regle */
    ajoute_lettres_regle (l, lettres);

/* et enfin, on remet à Faux les booléens des lettres qui ont des règles d'évolution
toutes les cases à TRUE du tableau correspondront donc à des règles manquantes */
    for (i=0; i<NB_LETTRES_AL; i++) {
        if (lettres [i]) {
            bool temp;

            temp = existe ( (char) (i + ((int) 'A')), l);
            lettres [i] = !temp;
            /* Essai */
        }
    }

/* on relit le tableau, si une case est à VRAI, on affiche un message d'avertissement
sur le flot choisi, et on met à VRAI la sortie */
    for (i=0; i<NB_LETTRES_AL; i++) {
        bool temp;

        temp=lettres [i];
        if (temp) {
            fprintf (sortie, "La_regle_d'evolution_de_%c_n'existe_pas\n", (char) (i + ((int) 'A')));
        }
        res = (res || temp);
    }

    return res;
}

/* va permettre de tracer l'objet dans sur le flot de sortie, en utilisant la syntaxe du pov */
/* 3 qualités possibles :
- 1 : on trace juste un cylindre
- 2 : on trace un cylindre avec une sphère au bout dans le sens du déplacement
- 3 : on trace le cylindre avec une sphère de chaque bout */
static void trace_tige (FILE * sort, Qualite qual, const Position or, const Position fi, const Diametre diam) {
    switch (qual) {
    case 3 :
        fprintf (sort, "\tsphere_");
        affiche_Pos (sort, or);
        fprintf (sort, "_,%f_\n", diam);
        /*~ NoBreak */
    case 2 :
        fprintf (sort, "\tsphere_");
        affiche_Pos (sort, fi);
        fprintf (sort, "_,%f_\n", diam);
        /*~ NoBreak */
    case 1 : /*~ NoBreak */
    default :
        fprintf (sort, "\tcylinder_");
        affiche_Pos (sort, or);
        fprintf (sort, "_");
        affiche_Pos (sort, fi);
        fprintf (sort, "_,%f_\n", diam);
    }
}

/* Modifie les paramètres d'état (qui sont passés sous forme de pointeurs) en fonction du
caractere d'évolution */
static void dessine_caract (const Init *init, const char a, FILE* sortie, Position * p,
    Direction *d, Direction *n, Diametre *diam, Position *maxi, Position *mini, Pile *pile) {

    Direction n_d, n_norm, axe;
    Position n_pos;

```

```

fprintf (stdout, "%c", a);
switch (a) {
case '+' : /* rotation normale + */
    n_d=rotate (*d,*n,init->angle1);
    *d = n_d;
    break;
case '-' : /* rotation normale - */
    n_d=rotate (*d,*n,-init->angle1);
    *d = n_d;
    break;
case '*' : /* rotation propre + */
    n_norm=rotate (*n,*d,init->angle2);
    *n = n_norm;
    break;
case '/' : /* rotation propre - */
    n_norm=rotate (*n,*d,-init->angle2);
    *n = n_norm;
    break;
case '(' : /* rotation orthonormale + */
    axe=prodVect (*n,*d);
    n_norm=rotate (*n,axe,init->angle3);
    n_d=rotate (*d,axe,init->angle3);
    *n = n_norm;
    *d = n_d;
    break;
case ')' : /* rotation orthonormale - */
    axe=prodVect (*n,*d);
    n_norm=rotate (*n,axe,-init->angle3);
    n_d=rotate (*d,axe,-init->angle3);
    *n = n_norm;
    *d = n_d;
    break;
case 'G' : /* avance sans tracer */
    n_pos=addPosDir (*p,*d);
    *p = n_pos;
    break;
case 'R' : /* recule en tracant */
    n_d = scale (*d,-1);
    n_pos=addPosDir (*p,n_d);
    trace_tige (sortie,init->qualite,*p,n_pos,*diam);
    *p = n_pos;
    break;
case 'T' : /* recule sans tracer */
    n_d = scale (*d,-1);
    n_pos=addPosDir (*p,n_d);
    *p = n_pos;
    break;
case '&' : /* alonge branche */
    n_d=scale (*d,init->facteur_allongement);
    *d = n_d;
    break;
case '~' : /* raccourcit branche */
    n_d=scale (*d,1/(init->facteur_allongement));
    *d = n_d;
    break;
case '$' : /* augmente le diam */
    *diam = *diam * (init->facteur_dilatation);
    break;
case '^' : /* diminue le diam */
    *diam = *diam / (init->facteur_dilatation);
    break;
case '[' : /* push dans l'état courant */
    *pile=push (*p,*d,*n,*diam,*pile);
    break;
case ']' : /* pop dans l'état courant */
    if (*pile==NULL) {
        fprintf(stderr, "erreur_de_pile\n");
        exit (EXIT_FAILURE);
    } else {
        *p = (*pile)->pos;
        *d = (*pile)->dir;
        *n = (*pile)->norm;
        *diam = (*pile)->diam;
        *pile=pop(*pile);
    }
    break;
}
/* F et toutes les lettres non utilisées tracent en avançant */
case 'F' : /* ~NoBreak */
default :
    n_pos=addPosDir (*p,*d);
    trace_tige (sortie,init->qualite,*p,n_pos,*diam);
    *p = n_pos;
    break;
}
/* on stocke les positions extremes (voir vecteur.c) afin de permettre le redimensionnement final */
*maxi = vectMax (*maxi,*p);
*mini = vectMin (*mini,*p);
}

/* fonction de coeur, eminentement recursive */

```

```

static void moulinette (const Init *init, Nb_Iterations courant, FILE* sortie, const ListRegle l,
                      const char cha [], Position *p/*~PseudoOut*/, Direction * d1/*~PseudoOut*/,
                      Direction * norm/*~PseudoOut*/, Diametre *diam/*~PseudoOut*/, Position *maxi,
                      Position *mini, Pile *pile) {

/* a-t-on atteint la limite du niveau de récurrence? */
if (courant == (init->iterations)) { /* oui : on fait évoluer l'état courant pour tout le motif */
    unsigned int i;

    for (i=0; i<strlen (cha); i++) {
        dessine_caract (init, cha [i], sortie, p, d1, norm, diam, maxi, mini, pile);
    }
} else { /* non, on va appeler tous les motifs suivants */
    unsigned int i;
    char loc_mot[LONG_CHAINE];

    for (i=0; i<strlen (cha); i++) {
        if (existe (cha[i], l)) {
            extrait (l, cha[i], loc_mot);
            moulinette (init, courant+1, sortie, l, loc_mot, p, d1, norm, diam, maxi, mini, pile);
        } else {
            fprintf (stderr, "La regle pour %c n'existe pas\n", cha[i]);
            exit (EXIT_FAILURE);
        }
    }
}
}

/* créée sur le flot précisé un bel header pour le pov */
static void header_pov (FILE* sort) {
    fprintf (sort, "/* *****\n");
    fprintf (sort, "/* Fichier include cree par L-Syst *****\n");
    fprintf (sort, "/* disponible en open-source sur *****\n");
    fprintf (sort, "/* http://florent.revelut.free.fr/L_syst/index.html *****\n");
    fprintf (sort, "/* Programmation Florent 'Bruce' REVELUT *****\n");
    fprintf (sort, "/* Contact : Florent.Revelut@netcourrier.com *****\n");
    fprintf (sort, "/* *****\n");
    fprintf (sort, "#declare Arbre=union{\n");
}

/* texte de fermeture du fichier de pov */
static void fin_pov (FILE* sort) {
    fprintf (sort, "*/\n");
    fprintf (sort, "/* fin de la definition de l'arbre */");
}

/* en fonction des dimensions extremes, redimensionne l'objet afin qu'il
* conserve ses proportions
* rentre dans un cube de côté 1 */
static void resize (FILE* sort, Position maxi, Position mini) {
    float x, y, z, sc;

    x=((maxi.x-mini.x));
    y=((maxi.y-mini.y));
    z=((maxi.z-mini.z));
    sc=(x>y)?x:y;
    sc=(sc>z)?sc:z;
    fprintf (sort, "\t scale 1/%f\n", sc);
}

/* permet d'appeler la fonction recursive de moulinette */
static void appelle_moulinette (const Init *init, const char fichier_sortie [], const ListRegle l,
                                Pile *pile/*~PseudoOut*/) {
    Position pos;
    Direction d, n;
    Position maxi, mini;
    Diametre diam;
    FILE* sortie;

    pos= initPos (0,0,0);
    d = init->direction;
    n = init->normale;
    diam = init->diametre;
    maxi=pos;
    mini=pos;

/* on ouvre le fichier d'include du POV en écriture */
if ((sortie = fopen (fichier_sortie, "w")) == NULL) {
    fprintf (stderr, "Impossible d'ouvrir le fichier %s\n", fichier_sortie);
    exit (EXIT_FAILURE);
}

/* on ajoute la définition de l'objet et l'header */
header_pov (sortie);

/* on fait mouliner l'algorithme et écrire le fichier */
fprintf (stdout, "on lance la moulinette\n");
moulinette (init, 0, sortie, l, init->axiome, &pos, &d, &n, &diam, &maxi, &mini, pile);
}

```

```

        fprintf (stdout, "\nfin_de_la_moulinette\n");

/* on redimensionne pour qu'il tienne dans une cube unité */
resize ( sortie, maxi, mini);

/* on le déplace à la position d'origine */
fprintf ( sortie, "\ttranslate");
affiche_Pos ( sortie, init->depart);
fprintf ( sortie, "\n");

/* et on ferme l'objet et on met en place l'header de fin (!) */
fin_pov ( sortie);

/* et on ferme proprement le fichier */
if (fclose ( sortie) != 0) {
    fprintf ( stderr, "BEEP:_ERREUR_DE_CLOTURE\n");
    exit (EXIT_FAILURE);
}
}

/* Fin des fonctions du système L */

/* analyse des parametres de passes */

/* fonction affichant la version du programme */
static void display_version (void) {
    fprintf (stdout, "*****\n");
    fprintf (stdout, "*_L_syst_~_version_1.0_*\n");
    fprintf (stdout, "*****\n\n");
    fprintf (stdout, "Auteurs:_Florent_~Bruce_~Revelut\n");
    fprintf (stdout, "~~~~~Laurent_~'Superman'_~Casse\n\n");
    fprintf (stdout, "La_version_officielle_~se_trouve_sur\n");
    fprintf (stdout, "~~~~~_~http://florent.revelut.free.fr/L_syst/index.html_~-~\n");
    fprintf (stdout, "Pour_toute_remarque,_report_de_bug,_conseil_d'amélioration,_contacter:_\n");
    fprintf (stdout, "\tFlorent.Revelut@netcourrier.com\n");
    fprintf (stdout, "\n");
}

/* affiche aide */
static void display_help (void) {
    fprintf (stdout, "*****\n");
    fprintf (stdout, "*_L_syst_~_version_1.0_*\n");
    fprintf (stdout, "*****\n\n");
    fprintf (stdout, "syntaxe_L_syst_[options]\n");
    fprintf (stdout, "s'il_ny_a_pas_d'options_de_passées,_les_fichiers_seront\n");
    fprintf (stdout, "\t_en_entrée:_arbre.l\n");
    fprintf (stdout, "\t_en_sortie:_arbre.inc\n\n");
    fprintf (stdout, "Les_options_~que_l'on_peut_passer\n");
    fprintf (stdout, "-I: fichier_=>spécifie_le_fichier_d'entrée\n");
    fprintf (stdout, "-O: fichier_=>spécifie_le_fichier_de_sortie\n");
    fprintf (stdout, "-H_~=>affiche_cette_page_d'aide\n");
    fprintf (stdout, "-V_~=>affiche_la_version_courante_et_quelques_infos\n");
    fprintf (stdout, "\n");
}

/* analyse les paramètres passés en ligne de commande, et agit en conséquence */
static void analyse_ligne_commande (const int argc, const char * const argv [], Fichiers *f) {
    int i;

    for (i=1; i<argc; i++) {
        if (argv[i][0] != '-' ) {
            fprintf (stderr, "Les_directives_passées_sont_de_la_forme_~truc\n");
            fprintf (stderr, "La_directive_~%s_~est_donc_incorrecte\n", argv [i]);
            exit (EXIT_FAILURE);
        } else { /* directive correcte */
            unsigned int j,k;
            char s[LONG_CHAINE];

            switch (toupper (argv[i][1])) {
                case 'V' :
                    display_version ();
                    exit (EXIT_SUCCESS);
                case 'H' :
                    display_help ();
                    exit (EXIT_SUCCESS);
                case 'I' :
                    if (argv[i][2] != ':') {
                        fprintf (stderr,
                            "La_syntaxe_pour_définir_le_fichier_d'entrée_est\n");
                        fprintf (stderr, "\tL_syst_-I: fichier.l\n");
                        fprintf (stderr, "Vous_avez_écrit_~%s\n", argv [i]);
                        fprintf (stderr, "Arret_du_programme\n");
                        exit (EXIT_FAILURE);
                    }
                    for (j=3, k=0; j<strlen (argv [i]); j++, k++) {
                        s[k]=argv [i][j];
                    }
                    s[k]='\0';
                    if (strlen (s) == 0) {

```

```

        fprintf (stdout, "paramètre_ignoré_%s\n", argv [i]);
        fprintf (stdout,
                "La_syntaxe_pour_définir_le_fichier_d'entree_est\n");
        fprintf (stdout, "\tL_syst_O: fichier.inc\n");
    } else {
        strcpy (f->ouverture, s);
    }
    fprintf (stdout, "Fichier_d'entrée_: '%s'\n", f->ouverture);
    break;
case 'O' :
    if (argv [i][2]!=':') {
        fprintf (stderr,
                "La_syntaxe_pour_définir_le_fichier_de_sortie_est\n");
        fprintf (stderr, "\tL_syst_I: fichier.inc\n");
        fprintf (stderr, "Vous_avez_écrit_%s\n", argv [i]);
        fprintf (stderr, "Arret_du_programme\n");
        exit (EXIT_FAILURE);
    }
    for (j=3,k=0;j<strlen(argv [i]);j++,k++){
        s[k]=argv [i][j];
    }
    s[k]='\0';
    if (strlen(s) == 0) {
        fprintf (stdout, "paramètre_ignoré_%s\n", argv [i]);
        fprintf (stdout,
                "La_syntaxe_pour_définir_le_fichier_de_sortie_est\n");
        fprintf (stdout, "\tL_syst_I: fichier.inc\n");
    } else {
        strcpy (f->sortie, s);
    }
    fprintf (stdout, "Fichier_de_sortie_: '%s'\n", f->sortie);
    break;
default :
    fprintf (stderr, "La_directive_%s'_est_inconnue\n", argv [i]);
    fprintf (stderr, "Faire_L_syst_h_pour_connaître_les_directives\n");
    exit (EXIT_FAILURE);
    }
}
}

/* Fonction principale */
/* appelée au lancement du programme */
int main (const int argc, const char * const argv []) {
    /* tout un tas de variables utiles */
    static Init init={DEF_POS,DEF_DIR,DEF_NOR,DEF_QUAL,DEF_DIAM,DEF_ANGLE,DEF_ANGLE,DEF_ANGLE,
        DEF_SCAL,DEF_SCAL,DEF_IT,DEF_AX};
    static ListRegle list=NULL;
    static Fichiers fic={"arbre.l", "arbre.inc"};
    int a;
    Pile pile=NULL;

    /* on analyse les paramètres passés en ligne de commande */
    analyse_ligne_commande (argc, argv, &fic);

    /* on ajoute les règles d'évolution standard */
    add_regles_standards (&list);

    /* on analyse le fichier d'entrée */
    charge (fic.ouverture, &init, &list);
    /* et on affiche un résumé sur la sortie standard les paramètres de simulation */
    resume_conf (stdout, &init);

    /* on teste la coherence des règles d'évolutions */
    if (incoherence (stderr, list)) {
        /* si non, on arrete le programme tout de suite */
        fprintf (stderr, "Regles_d'evolution_non_coherentes\n");
        fprintf (stderr, "Arret_force_du_programme\n");
        exit (EXIT_FAILURE);
    }

    /* tout est ok, on lance la moulinette */
    appelle_moulinette (&init, fic.sortie, list, &pile);

    /* et on detruit la pile de règles */
    detruit (&list);

    /* fin du programme, tout s'est bien passé */
    return EXIT_SUCCESS;
}

```

3 Utilisation du programme

3.1 Le fichier de paramètres

Les caractéristiques de l'arbre sont passées par un fichier de paramètres. Par défaut, il s'appelle "arbre.l". Les données du fichiers sont case-insensitive, mais il vaut mieux éviter l'usage des accents. Ce fichier se compose de 2 parts importantes : les directives et les motifs.

3.1.1 Les directives

Elles vont servir à donner des informations au programme.

Les mots clefs reconnus sont :

AXIOME chaine : où chaine est une suite de motifs (explicité dans la section suivante) qui fixe l'axiome de départ de l'arbre. Cette directive est essentielle !

NIVEAU n ou ITERATIONS n : avec n un entier qui sert à choisir le niveau de récurrence, donc la complexité du resultat (et par la même occasion le temps de calcul).

ORIGINE v : v est un vecteur qui fixe une origine au dessin. Par défaut, il part de <0,0,0>

DIRECTION v : v est un vecteur qui fixe la direction d'origine pour les récurrences. Par défaut <0,1,0>

NORMALE v : v est un vecteur qui fixe la normale d'origine pour les récurrences. Cette variable ne sert que pour les rotations. Par défaut, v vaut <0,0,1>

QUALITE n : avec n un entier qui fixe la qualité du tracé (et donc également le temps de calcul)

- 1 : cylindres

- 2 : cylindres + spheres

- 3 : double spheres + cylindres

ANGLE1 alpha1 : avec alpha1 variable en virgule flottante qui définit l'angle de rotation normale en degrés. Par défaut, il vaut 60 degrés.

ANGLE2 alpha2 : avec alpha1 variable en virgule flottante qui définit l'angle de rotation propre en degrés. Par défaut, il vaut 60 degrés.

ANGLE3 alpha3 : avec alpha1 variable en virgule flottante qui définit l'angle de rotation orthpnormale en degrés. Par défaut, il vaut 60 degrés.

DIAMETRE diam : avec diam diametre de l'objet, ie des cylindres. Par défaut, il vaut 1.

FACTEUR_ALLONGEMENT facteur : avec facteur nombre en virgule flottante qui définit le facteur d'allongement (>1). Par défaut, il vaut 1.

FACTEUR_DILATATION facteur : avec facteur nombre en virgule flottante qui définit le facteur de dilatation (ie d'agrandissement du diamètre des branches) (>1). Par défaut, il vaut 1.

3.1.2 Les motifs

Ce sont les règles d'évolution. Elles se présentent sous la forme caract chaine : où caract est un caractère, et chaine une suite de caractères.

Exemple :

A	AB
B	A+B-A

On lit n (NIVEAU) fois la chaîne en remplaçant à chaque fois chaque caractère par sa chaîne correspondante. Les chaînes peuvent contenir tous les caractères cités dans le cadre de la 2D (et repris dans la 3D). Des exemples concrets avec leur illustration sont fournis dans les sections suivantes.

3.2 Tutoriel pour créer sa première image

Pour créer une image en utilisant ce logiciel, c'est assez simple, au moins tant que l'on se limite à des scènes basiques.

Le premier temps va consister à créer le fichier .l, comme ce dernier a été décrit précédemment. Pour cela, n'importe quel éditeur convient, et le document final peut aussi bien être enregistré au format unix (caractère 0h0A en fin de lignes) ou DOS (caractères 0h0D0A en fin de ligne). Appelons ce fichier par exemple "mon_arbre.l".

Ensuite, il va falloir interpréter ce fichier, et en faire un fichier include pour le POV. Supposons que l'on veuille que l'include pour le POV s'appelle essai.inc, il suffira de lancer la commande suivante :

```
./L_syst -i :mon_arbre.l -o :essai.inc sur un linux
```

```
L_syst -i :mon_arbre.l -o :essai.inc sur un windows ou dos
```

Ensuite, il suffit de charger le POV, de prendre une scène simple (par exemple arbre.pov qui est fournie avec le logiciel), et de la faire rendre avec ses options de prédilection. Il est à noter que le nom de l'objet de

l'inclure va s'appeler Arbre. Pour plus d'informations sur la partie de rendu, ne pas hésiter à se reporter à la documentation de Persistence of Vision, cette dernière est réellement très bien faite.

3.3 Pour créer des fractals usuels

Nous vous proposons ici trois des courbes fractales les plus classiques : le flocon de Von Koch, la courbe de Peano et le flocon de neige.

3.3.1 Le flocon de Von Koch

Une implémentation du flocon de Von Koch est donnée par le fichier vonkoch.l

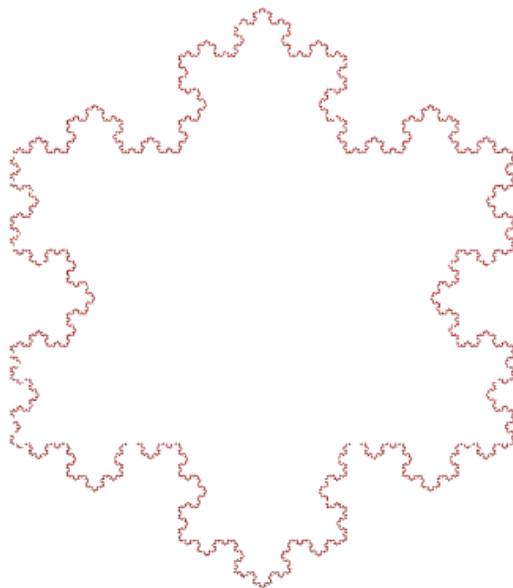
```
# flocon de Von Koch
iterations      5
origine <-.5,.2886,0>
direction <1,0,0>
QuAlItE 2
diametre .2
angle1 60
```

```
axiome f--f--f--
```

```
% Von Koch
```

```
f f+f--f+f
```

On obtient l'image suivante :



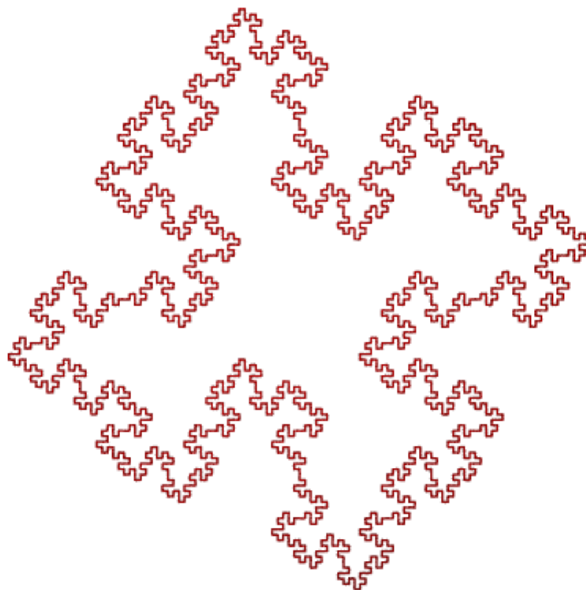
3.3.2 L'île quadratique de von Koch

```
# ile quadratique de von koch
iterations      3
origine <-.25,-.25,0>
direction <1,0,0>
QuAlItE 2
%diametre 2
diametre .2
angle1 90
```

```
axiome f+f+F+f
```

```
% Von Koch
f F+F-F-FF+F+F-F
```

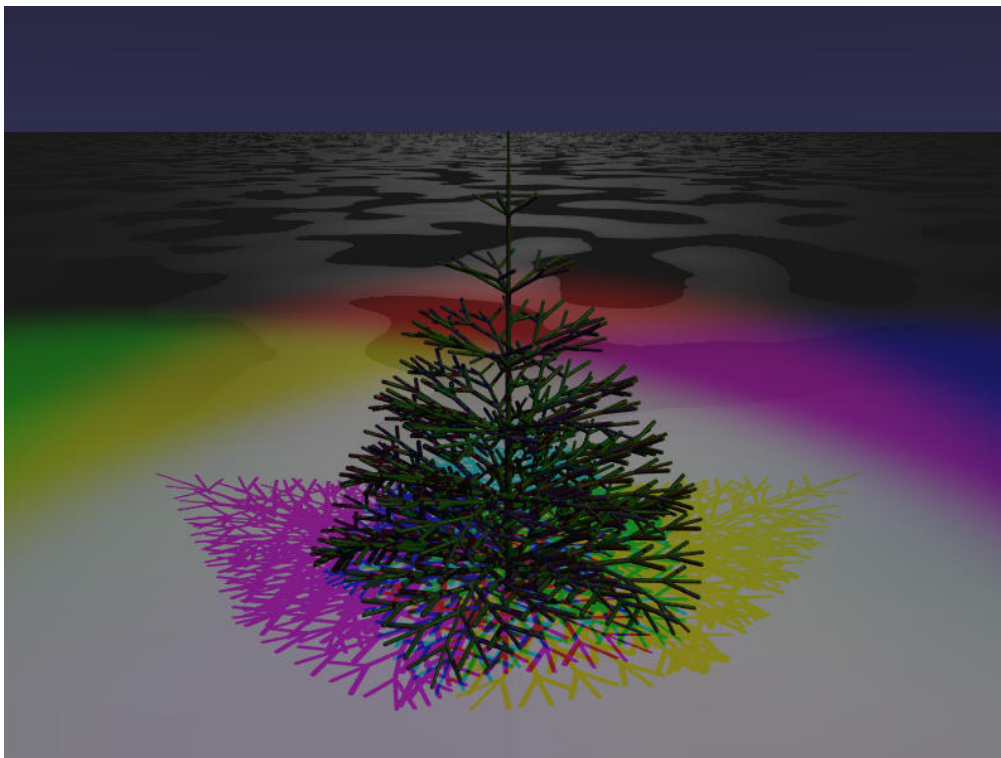
On obtient l'image suivante :



3.4 Pour créer des arbres en 3D

La création d'arbres en 3D est plus complexe. Il faut déjà avoir bien compris le principe des rotations dans l'espace, ainsi que la récursion de l'algorithme du système L. Ensuite, il faut arriver à prévoir ce que va donner le modèle quand on va réitérer, ce qui n'est pas toujours évident. Cependant, avec un petit peu de pratiques et quelques essais, il est possible d'arriver à des résultats très corrects, comme le prouvent les deux exemples qui suivent.

3.4.1 Le plus classique de tous : le sapin !



Et le fichier en .l nécessaire pour le construire :

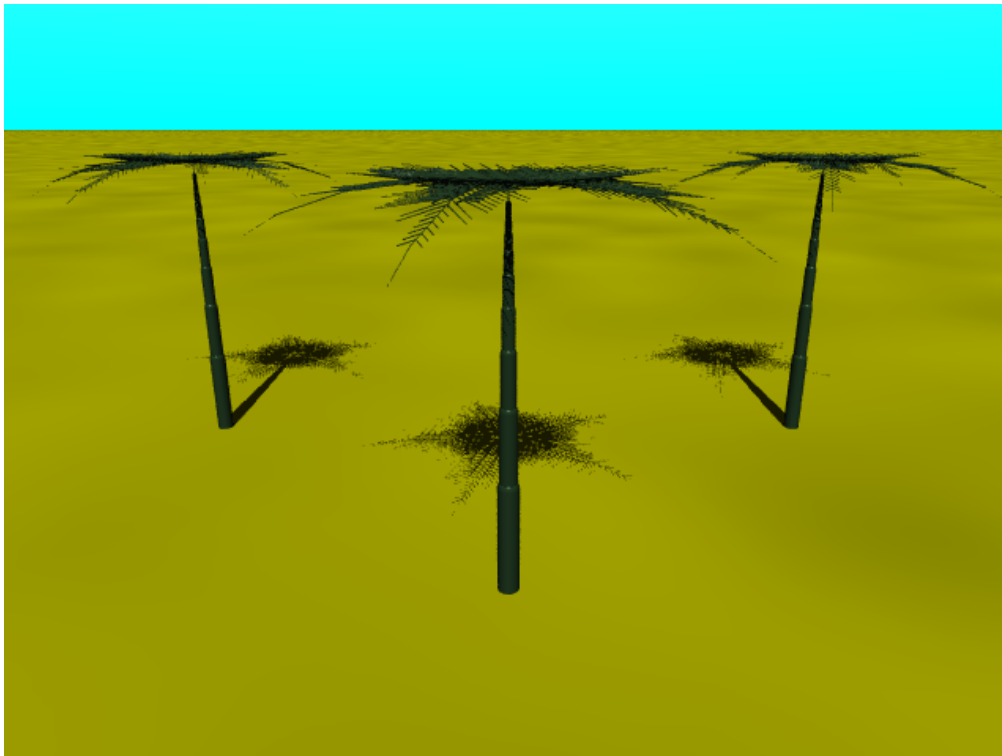
```
# sapin en 3 dimensions
iterations      8
origine <0,0,0>
direction       <0,1,0>
QualItE 2
angle1 12 %25
angle2 72
angle3 40

diametre .1
facteur_dilatation 1.2 % 1.1
facteur_allongement 1.2

axiome fa

a [+++++d][//+++++d][////+++++d][**+++++d][*****+++++d] $h^^a
d f[(e)[ ]e]^d
e f[(f)[ ]f]^e
c f^c
h ff
f f
```

3.4.2 Plus inattendu : le palmier



Et le fichier en .l nécessaire pour le construire :

```
# Palmier
iterations      8
origine <0,0,0>
direction       <0,1,0>
qualite 2
angle1 12 %25
angle2 72
angle3 40

diametre 3
facteur_dilatation 1.2 % 1.1
facteur_allongement 1.2

axiome hhai~b

a e&&^a
b [*c][**c][[/c][//c][c]
c +++++++d
d mmmmm^+k
e ee
h $l
i ^i
k mm^d
p f^m^p %f^f^p
l h
m $$[(p)[]p]^~~~f&&
f f
```

4 Sources utilisées

Notre source principale de documentation a été Internet, qui est un média très fourni pour tous les problèmes liés à la création de fractals. En particulier, nous avons été inspirés par les sites suivants :

- <http://www.cogs.susx.ac.uk/users/gabro/lsys/lsys.html> : une très bonne page, qui décrit de manière simple le fonctionnement du système L dans le cas de la génération en 2 dimensions.
- <http://spanky.triumf.ca/www/fractint/lsys/plants.html>
- <http://www.povray.org> : le site sur ray tracer que nous avons utilisé, qui permet de le télécharger, ainsi que de trouver des liens vers de nombreux autres sites concernant la synthèse d'images 3D.

D'autre part, lorsque Internet ne nous a pas fourni directement des URL utilisables, nous avons souvent pu télécharger des résumés de thèses ou articles intéressants, comme :

Dynamic 3-Dimensional Lindenmayer Systems, de Dee Jay Randall (université de Régina, canada)

Table des matières

1	Introduction	1
1.1	Pov-Ray	1
1.2	Modèle utilisé	1
2	Le programme que nous avons créé	1
2.1	Description de l'algorithme utilisé	1
2.1.1	Description théorique à 2 dimensions	1
2.1.2	Description théorique à 3 dimensions	2
2.2	Codage	3
2.3	La bibliothèque de gestion vectorielle	3
2.3.1	Fichier d'en-tête : vecteur.h	3
2.3.2	Code : vecteur.c	4
2.4	Le programme principal	6
3	Utilisation du programme	20
3.1	Le fichier de paramètres	20
3.1.1	Les directives	20
3.1.2	Les motifs	20
3.2	Tutoriel pour créer sa première image	20
3.3	Pour créer des fractals usuels	21
3.3.1	Le flocon de Von Koch	21
3.3.2	L'île quadratique de von Koch	21
3.4	Pour créer des arbres en 3D	22
3.4.1	Le plus classique de tous : le sapin !	23
3.4.2	Plus inattendu : le palmier	24
4	Sources utilisées	25