# Rapport de stage

## Programmation d’un escalier à l’aide du logiciel SGDLsoft.

par

Mathieu PARADIS,
étudiant 2e cycle à l'Université Laval
94 071 597

présenté à

M. Temy TIDAFI,
professeur à l'École d'architecture de l'Université de Montréal

M. Pierre CÔTÉ
professeur à l'École d'architecture de l'Université Laval

Été 1999
GRCAO
École d’architecture
Université de Montréal.

# Table des matières

# Introduction

Ce texte a pour but de résumer les différentes réalisations, découvertes et explorations réalisées dans le cadre d'un stage que nous avons effectué à l'été 1999 au sein du Groupe de recherche en conception assistée par ordinateur (GRCAO) de l'Université de Montréal. L'ensemble du travail a été réalisé sur une station *Sun* équipée du système d'exploitation *Solaris*, à l'aide du logiciel *SGDLSoft*, via l'éditeur de texte *xemacs*.

La tâche à accomplir lors du stage n'a pas été définie précisément à l'avance, ceci pour permettre au professeur responsable, Temy TIDAFI, d'adapter le contenu pratique du stage en fonction de nos souhaits et capacités, ainsi que des ressources matérielles et temporelles disponibles. Les objectifs que nous nous sommes fixés ont ainsi plusieurs fois évolué pour m'inciter à exploiter un maximum des ressources qui m'étaient offertes au GRCAO durant mon séjour. Le rapport qui va suivre prendra donc un peu la forme d'un carnet de voyage, à l'intérieur duquel nous essaierons de bien faire comprendre quelles ont été nos démarches, à quels résultats aboutirent-elles, et dans quelle mesure elles ont démontré l'utilité d'un langage de programmation comme SGDLsoft dans l'exercice de la modélisation architecturale.

Il faut ici évoquer les conditions dans lesquelles la rédaction de ce document s'est effectuée, car elles auront eu une certaine influence sur la forme et le contenu dudit document. En effet, pour mener à bien l'éxécution de ce rapport, nous n'avons eu comme témoignage de notre travail qu'un ensemble de quatorze fichiers *Scheme*, correspondant chronologiquement aux divers stades de notre démarche de travail. À partir de ces archives, nous avons reconstitué des documents SGDL abrégés qui représentent bien les diverses étapes de notre travail et l'évolution des structures globales du programme. Nous allons donc, pour construire le corps du texte, parcourir ces différents documents et préciser, aux moments opportuns, quels furent les objectifs particuliers poursuivis lors de la réalisation de chacune des étapes et les défis ayant été inhérents aux tâches accomplies. L'absence du logiciel comme tel nous empêche pour le moment

d'ajouter des images au texte, mais ces limitations devraient être éliminées lors d'une version ultérieure du présent document.

Notons que les documents originaux et, conséquemment, fonctionnels, sont disponibles en annexe 2, mais que leur lecture sera beaucoup moins évidente que celle des documents abrégés annotés en annexe 1. Nous avons volontairement éliminé, en première annexe, divers éléments redondants ainsi que la majeure partie des fonctions d'opérations mathématiques, ceci pour pouvoir permettre au lecteur de concentrer son attention sur la structure générale du code informatique développé.

Bien que les objectifs aient été fluctuants, comme nous l'avons précédemment mentionné, le thème gouvernant les différentes phases de programmation est resté le même, soit la génération d'un escalier. Comme nous allons donc le voir, l'escalier a été envisagé de plusieurs façons, qui ont toutes (ou presque) été traduites, au mieux de mes connaissances, en langage SGDLsoft.

# Rapport de stage

Nous aimerions commencer ce rapport en adressant nos remerciements les plus sincères à Frédéric ARMENJON et David GUENNIFEY, stagiaires français et membres du GRCAO, pour l'aide inestimable qu'il nous ont apportée aux premiers moments du stage. Ce sont eux qui, par leurs conseils pertinents et généreux relatifs aux système d'exploitation *Solaris* et à l'éditeur de texte *xemacs*, nous ont permis de commencer très rapidement à explorer le logiciel SGDLsoft. Nous leur devons donc beaucoup, car, bien que nous n'ayons pas eu quantité de temps disponible pour effectuer ce stage, nous en avons retiré beaucoup et en sommes parvenus à exploiter productivement quelques-unes des nombreuses possibilités qu'offre SGDLsoft.

**Marche.scm**

Nous avions, en commençant le stage, comme expérience pertinente, une certaine connaissance des langages C et C++, quelques lectures concernant les langages *Scheme* et *Lisp*, ainsi que quelques descriptions sommaires des caractéristiques particulières de SGDLsoft . Après avoir compris la dualité qui existait entre le langage interfacique (Scheme) et le corps constituant le logiciel proprement dit, nous avons entrepris une première approche du langage via quelques expériences avec la primitive géométrique universelle de SGDLsoft. Lorsque ces divers essais furent terminés, Temy TIDAFI détermina le thème global qui allait gouverner le stage, soit la programmation de la génération d'un escalier.

Nous avons ensuite réalisé notre premier programme (marche.scm), qui génère une marche puis produit une vue de cette marche. Cet essai fort primitif nous a toutefois permis de réaliser quelques fonctions améliorant considérablement notre méthode de travail , soit une fonction de rechargement automatique ainsi qu'une fonction globale gérant toutes les fonctions de l'affichage à partir de la fonction créant le modèle lui-même. Ces deux fonctions restèrent, à l'exception de quelques améliorations, inchangées dans les versions ultérieures de

nos exercices, et ceci nous a permis de mieux nous concentrer sur les aspects structurels de la génération du modèle. (v. annexe 1 ).

**Escalier_1.scm**

Ayant entrevu la puissance d'un langage fonctionnel comme Scheme, nous avons décidé de ne pas recourir aux solutions traditionnelles des langages dits impératifs et de nous forcer à exploiter de nouvelles ressources. Nous avons donc essayé d'étudier le problème sous un nouvel angle, et nous avons rapidement effectué un rapprochement entre la suite des marches d'un escalier et la facilité avec laquelle il était possible d'utiliser le mécanisme de la récursivité. Nous avons donc terminé un deuxième programme (escalier_1.scm) qui permettait de construire une volée de marches en fonction de trois paramètres numériques passés à la fonction de base, soit le nombre de marches, leur largeur, ainsi que leur hauteur. (v. annexe 1). La construction de la volée proprement dite s'accomplissait au moyen d'une fonction récursive unissant les marches au fur et à mesure qu'elles étaient créées. De plus, cette structure était triplée pour permettre de générer des volées droites, circulaires et en biseau.

**Escalier_2.scm**

Après examen du programme, Temy nous a alors conseillé d'essayer de fusionner les codes générant des marches droites et circulaires, en utilisant les possibilités géométriques de la primitive solide de SGDLsoft. En effet, comme nous l'avons découvert, cette primitive permet de modéliser n'importe quel volume représenté par une équation quadrique dans l'espace. Toutefois, cette primitive n'est pas définie par son équation explicite, mais bien par un certain nombre de points dans l'espace qui, par leur position, fournissent assez d'information pour définir exactement ses paramètres mathématiques. De plus, étant donné que SGDLsoft intègre les propriétés de la géométrie dite affine, ces points peuvent être placés à l'infini de manière exacte, via ce qu'on appelle des coordonnées homogènes, lesquelles ne sont en fait que des coordonnées entières

accompagnées d'une coordonnée supplémentaire représentant leur dénominateur commun. Ces coordonnées permettent ainsi à la fois de représenter l'ensemble des coordonnées dans $R^3$ uniquement au moyen de nombres entiers, et de positionner des points à l'infini en fixant le dénominateur à zéro. Ces caractéristiques permettent donc de gérer les paramètres de la primitive avec beaucoup de flexibilité. Nous ne nous étendrons pas ici sur les caractéristiques particulières de la géométrie de base de SGDL, pour la simple et bonne raison que nous n'en avons utilisé les possibilités que d'une manière macroscopique et sans véritable compréhension totale de ses mécanismes internes, le temps nécessaire à la maîtrise d'un tel sujet nous faisant défaut.

Nous avons ensuite compris l'équivalence qui pouvait exister entre, d'une part, la forme générale d'escaliers droits ou circulaires, et d'autre part, le positionnement de certains points de contrôle de la primitive à l'infini ou non. En effet, on peut considérer, pour un escalier droit, qu'il est en fait un escalier circulaire mais dont le point pivot, c'est à dire le centre de sa circonférence, serait situé à l'infini (figure 1). Une fois ce constat réalisé, nous avons réalisé un troisième programme (escalier_2.scm), qui tentait d'exploiter ce phénomène au profit d'une simplification du code. (v. annexe 1).

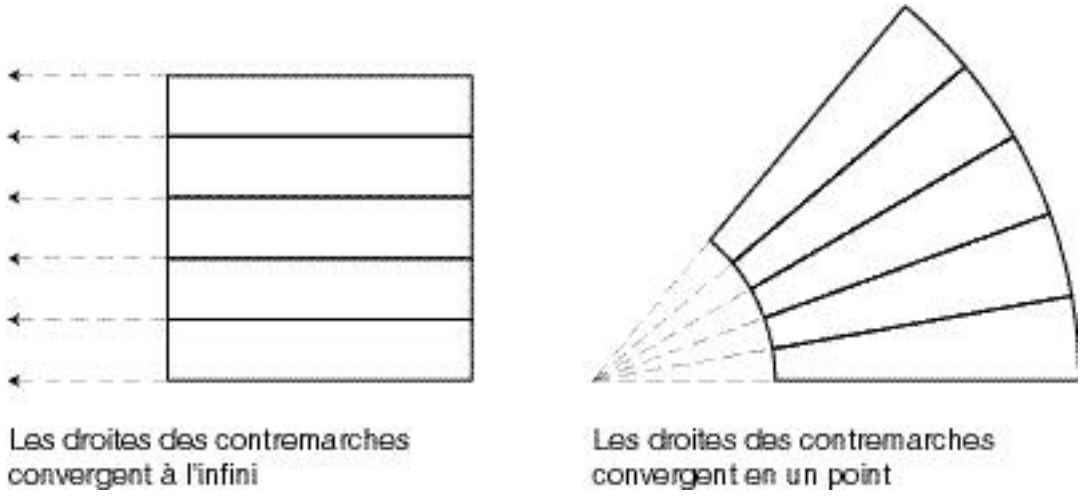



Figure 1.

Nous avons ainsi conçu le programme de manière à ce que l'escalier soit défini à l'aide d'opérateurs volumiques appliqués à des primitives planes infinies horizontaux correspondant aux marches, appliqués à des primitives planes infinies

horizontaux correspondant aux contremarches et les volumes, et à des primitives représentant les côtés de l'escalier, qu'ils soient cylindriques ou planaires. De plus, nous avons cette fois–ci considéré la récursivité pour ajouter des volumes à une liste d'éléments, au lieu de s'en servir pour éxécuter plusieurs opérations volumiques successives, comme nous l'avions fait précédemment.

Pour réaliser la tâche de construire la volée de marches, la fonction requérait cinq variables, soit le nombre de marches, leur hauteur, leur largeur, ainsi que deux variables représentant le numérateur et le dénominateur du rayon de courbure. De cette manière, il était possible de positionner le centre de rotation à l'infini, en fixant le dénominateur à zéro. Par exemple :

**( m 8 14 0.9 3 1 )**

Cette fonction génère une volée de 8 marches, de 0.9 mètres de largeur, d'une hauteur de 14 centimètres et d'un rayon de courbure de 3 mètres, soit 3 / 1 mètres ( la lettre "m" est totalement arbitraire ).

Nous avons aussi profité de cet exercice pour intégrer à notre fonction d'interface l'union du volume de l'escalier avec trois cylindres représentant les axes X, Y et Z, pour améliorer le repérage dans l'espace tridimensionnel.

**Escalier_3.scm**

Poursuivant notre exploration des différentes structures permettant d'accomplir la tâche, nous avons, à la suggestion de Temy, qui nous avait conseillé d'exploiter plus efficacement les possibilités offertes par les opérations de création et de manipulation de listes, essayé de rationaliser notre approche en créant trois fonctions recursives qui généreraient chacune une liste de volumes, soit les volumes des plans horizontaux, les volumes des plans des contremarches et les volumes des côtés. Nous avons adopté cette méthode parce qu'elle représentait mieux l'équivalence conceptuelle qui devait exister entre, d'une part, les éléments constituant chaque marche, soit la marche, la contremarche et les côtés, et d'autre part, les fonctions définissant chacun de ces éléments. Le processus de génération de la volée de marches se poursuivait avec une fonction qui créait une liste de marches à partir des trois listes précédemment évoquées. L'union entre

ces diverses marches représentait alors le volume final. Ces modifications ont donc été implantées dans un nouveau programme (escalier_3.scm), dont les principales caractéristiques sont notées en annexe 1.

**Escalier_4.scm - escalier_5.scm - escalier_6.scm**

Après ces divers résultats, Temy et nous–même nous sommes fixé comme objectif la réalisation d'un programme utilisant la même structure, mais modifié de façon à pouvoir générer un escalier elliptique. Ceci devait finalement déboucher sur une difficulté majeure de ce stage, et, bien que nous ne présenteront pas de façon annotée le code des programmes ayant constitué ces essais, nous allons ici brièvement énoncer les multiples raisons de cette difficulté.

Premièrement, il faut mentionner que l'ellipse est une figure qui, bien que facile à tracer et dont l'équation algébrique soit bien connue, recèle quelques particularités mathématiques loin d'être évidentes pour le profane. Ainsi, il n'existe pas d'équation simple représentant la circonférence d'une ellipse, comme il en existe pour le cercle ou le triangle. Seules des approximations, et loin d'être concises, pouvaient donc nous permettre de déterminer quelle était la circonférence d'une ellipse, ce qui rendait le calcul des largeurs de marche extrêmement ardu. De plus, pour être réaliste par rapport aux escaliers ellipsoïdes qu'il m'avait été donné de voir, nous avons essayé d'intégrer à nos calculs le fait que l'orientation des contremarches ne converge pas vers un centre unique, car ces contremarches doivent, pour que l'escalier soit praticable, être perpendiculaires à la pente (selon les axes X et Y) de l'ellipse définissant le parcours dudit escalier. Ensuite, nous avons essayé de tenir compte du fait que les contremarches n'ont pas toutes le même écart angulaire, car elles convergent toutes vers des centres différents, tout en devant borner des arcs d'ellipse de longueur égale (figure 2). La relation entre la convergence, l'écart angulaire et la dimension des arcs d'ellipse ayant résisté à nos tentatives pour l'approximer, nous avons, en raison de facteurs temporels et de la relative inutilité d'une telle fonction, fini par abandonner cet objectif, sans être parvenu à aucun résultat jugé satisfaisant. Ces différentes tentatives sont décrites par des programmes

(escalier_4.scm, escalier_5.scm et escalier_6.scm ), que nous n'avons pas jugé utile d'annoter en annexe.

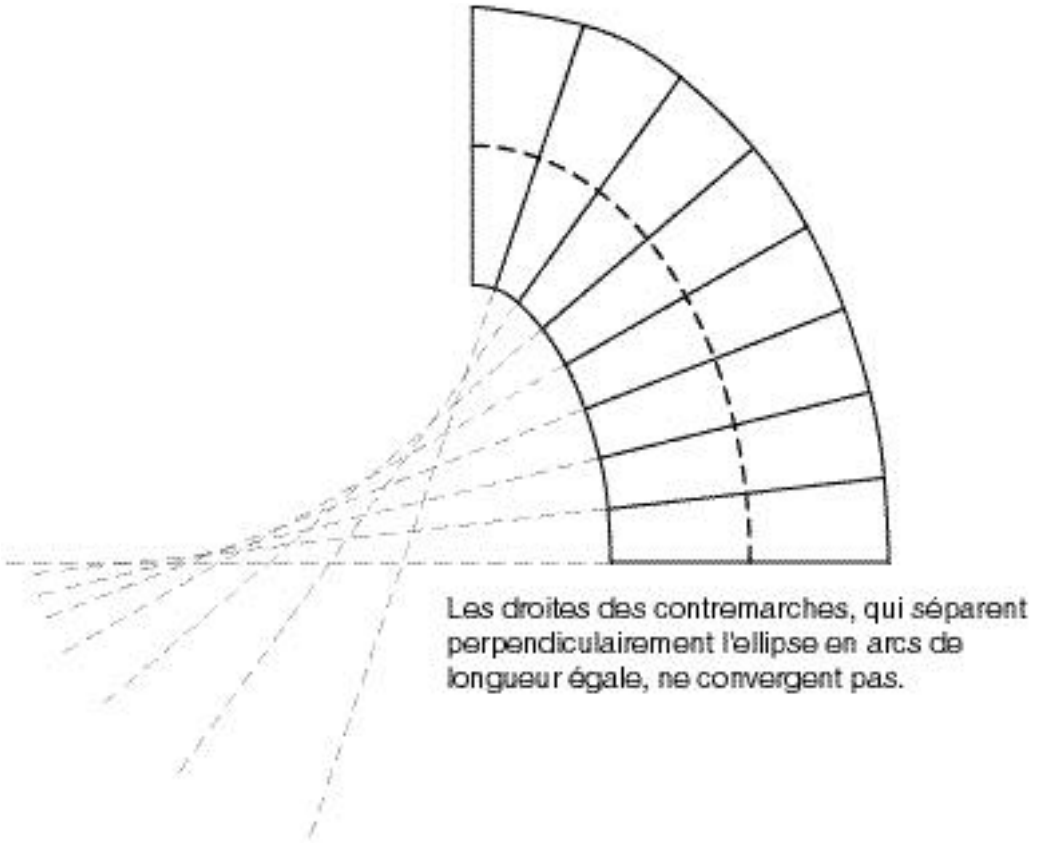



Figure 2.

**Escalier_7.scm**

Après ces quelques expériences infructueuses, Temy et nous–même avons décidé d'orienter nos efforts pour tenter d'améliorer la stucture globale de notre projet, en séparant les fonctions de création des volumes finaux des fonctions de calcul des données géométriques de l'escalier. En effet, jusqu'alors, les marches des escaliers étaient de simples boîtes, sans aucun détail superflu, parce que les calculs des différents éléments de chaque marche étaient immédiatement traduits en primitives SGDLsoft. Une fois ces volumes créés, il n'était pratiquement plus possible de les modifier localement, et les plans des marches et contremarches, ainsi que les surfaces des côtés constituaient toute la définition de l'escalier. Or, pour essayer de modéliser plus adéquatement l'escalier, nous avons compris qu'en séparant les deux ensembles de fonctions précédemment évoqués, nous

étions capables de séparer l'information générale et les particularités. Pour ce faire, nous avons modifié nos fonctions de construction de listes pour qu'elles construisent, au lieu d'une liste de primitives géométriques, une liste de vecteurs représentant chacun les huit points des coins de chaque marche. Ce faisant, il nous était ensuite possible de considérer une fonction indépendante (un style) qui exploite cette liste pour générer les volumes de la volée de marches. Ces différentes évolutions constituent le corps du programme "escalier_7.scm", dont les fonctions annotées sont disponibles en annexe 1. Pour vérifier l'efficacité d'une telle méthode, nous avons choisi, arbitrairement, comme fonction de construction de marches, l'union entre douze cylindres représentant les arêtes de chaque marche.

Évidemment, toute méthode est orientée vers l'exploitation de certains phénomènes au détriment d'autres. À cette étape, nous avons dû éliminer tous les algorithmes qui tenaient compte du positionnement des points de contrôle des primitives à l'infini, pour les remplacer par des algorithmes conditionnels, pour générer les sommets des marches. Cette conclusion s'explique par le fait que l'information géométrique de chaque marche est définie, dans notre structure, par les coordonnées de ses huit sommets, sans tenir compte de l'orientation de ses éléments. Cette information étant locale, le positionnement d'un point à l'infini n'a plus été nécessaire, et les différences entre les volées droites et circulaires n'a plus été visible qu'au niveau de la localisation relative des sommets des marches. Nous avons toutefois gardé le même interface au niveau de la fonction primaire (nombre de marches, hauteur des marches, largeur des marches, nominateur du rayon de courbure et dénominateur du rayon de courbure ), mais les données que la fonction a eues à traiter ont été alors interprétées différemment.

**Escalier_8.scm**

Maintenant que nous n'exploitions plus les caractéristiques spécifiques de la primitive géométrique SGDLsoft, il était logique d'explorer l'autre aspect du logiciel plus à fond, c'est à dire d'essayer d'exploiter les capacités naturelles du langage *Scheme*. Nous avons alors, après des discussions avec Temy, tenté de traduire

l'équivalence conceptuelle qui existait entre une série d'éléments constituant un escalier (volées de marches et paliers) et une liste de listes, étant donné que les langages de la famille *Lisp*, dont *Scheme* fait partie, sont extrêmement efficaces en ce qui a trait à la manipulation de telles structures de données.

L'étape suivante a donc été de, premièrement, modifier l'interface de la fonction primaire pour lui faire accepter un nombre arbitraire de paramètres représentant les différents éléments de l'escalier à modéliser. Nous avons conséquemment dû ensuite créer une syntaxe et des fonctions pour la traduire en listes valides. Troisièmement, il nous a fallu modifier considérablement les fonctions de positionnement des marches, car ce positionnement, fonction d'un certain nombre de paramètres, avait toujours comme point de départ l'origine du système, alors qu'il fallait maintenant pouvoir positionner une volée de marches ou un palier à partir d'une localisation arbitraire et fonction de la position de l'élément précédent dans la liste. Nous avons donc résolu ce problème en construisant des fonctions de rotation et de translation qui, pour chaque élément de la liste (palier ou volée de marches), étaient capables de le positionner en fonction des points terminaux de l'élément précédent. Nous avons aussi, en plusieurs endroits, conçu des fonctions récursives doubles, dont l'effet combiné permettait d'atteindre un résultat satisfaisant. Reprenant toutefois la méthode développée précédemment (escalier_7.scm), l'aspect final de l'escalier est géré par une fonction indépendante appliquée à la liste des sommets des éléments unitaires de l'escalier.

Étant donné que la structure du programme (escalier_8.scm) s'est considérablement complexifiée, la version abrégée, située en annexe 1, contient des annotations plus détaillées qui dérivent les mécanismes globaux qui nous ont permis d'arriver à une solution satisfaisante et d'une manière qui nous paraissait élégante.

Comme exemple, voyons l'effet de la fonction :

**( e 1.7 15 ( c ( m 5 2 1 ) ( p 3 3.14 ) ( m 6 1 0 )))**

Cette fonction va créer un escalier large de 1.7 mètres et dont les marches et paliers auront 15 centimètres de hauteur. La profondeur des marches sera calculée avec la fonction : profondeur = 64 – 2 x hauteur, qui propose un rapport

habituellement utilisé.  Cet escalier sera composé de trois parties, soit une première volée de 5 marches circulaires ayant un rayon de courbure de 2 mètres (2 / 1), un palier long de 3 mètres qui effectuera une rotation de 3.14 radians, soit approximativement 180 degrés, puis une autre volée de 6 marches, droites en raison de leur rayon de courbure infini (1 / 0). (La lettre "c" est totalement arbitraire, au contraire de "e" pour "escalier", "m" pour "marches" et "p" pour "palier", dans ce cas–ci.)

Cette syntaxe permet ainsi de facilement modéliser une variété infinie d'escaliers en fonction d'un nombre variable de paramètres passés au moyen d'une syntaxe qui nous apparaissait claire et efficace.

**Escalier_9.scm**

Pour ce dernier programme, le temps qui nous restait n'étant pas suffisant pour développer de nouvelles structures dans les fonctions de création de listes de coordonnées, nous avons décidé d'essayer d'enrichir la fonction de gestion du style de l'escalier, qui était restée jusqu'alors assez sommaire.  Nous avons donc utilisé plusieurs des fonctions macroscopiques de SGDLsoft, lesquelles sont évidemment basées sur la primitive unique, pour créer des formes permettant de donner à l'escalier un aspect plus traditionnel, et ainsi démontrer qu'il était facile d'implanter un style plus complexe de façon concise et rapide.

Le nouveau style était composé, pour chaque marche ou morceau de palier, d'un prisme remplissant l'espace défini par les sommets de l'élément, d'un rebord en quart–de–rond et de deux poteaux soutenant une portion de rampe (figure 4). L'implantation de ce style s'est déroulée sans véritable problème, à l'exception du fait que certaines fonctions de création de formes géométriques génèrent des volumes infinis lorsqu'on leur soumet des points confondus et qui ne devraient pas généralement l'être.  Ce genre de situation était généré sans plus de traitement par la combinaison de certaines valeurs des paramètres d'entrée, car nous n'avions pas conçu notre programme pour filtrer ces points, les difficultés subséquentes nous étant alors inconnues.  Nous avons résolu ce problème en soumettant les listes de points à des fonctions de filtration qui devaient rendre inopérants ces

points confondus, ce qui a réglé les problèmes et nous a permis de parachever ce dernier programme (escalier_9.scm) de manière fort satisfaisante.

Les détails d'implantation de ce style figurent évidemment en annexe 1, ce qui permettra au lecteur de mieux comprendre le rôle de chaque fonction géométrique dans la construction du volume final de l'escalier.

# Conclusion

Ces diverses évolutions constituent donc l'essentiel de notre travail pendant ce mois de stage, et les différentes étapes que nous avons franchies montrent bien, d'une part, que la modélisation par programmation est une méthode viable et efficace, et, d'autre part, que SGDLsoft est un logiciel extrêmement polyvalent, car il nous a permis d'arriver à nos fins de plusieurs manières conceptuellement très éloignées.

Cette polyvalence n'a d'ailleurs pas été étudiée à fond par nos essais, car nous n'avons utilisé que les fonctions et aspects utiles aux méthodes successives que nous avons employées. Il est donc évident que si nous avions considéré l'escalier sous un autre aspect, d'autres fonctionnalités auraient pu nous aider à atteindre nos buts.

En particulier, la capacité qu'a SGDLsoft à gérer la densité au moyen d'un nombre d'opérateurs arithmétiques et autres aurait pu s'avérer pertinente si nous avions concentré nos efforts dans cette direction. Cet aspect de SGDLsoft est d'ailleurs, à notre sens, un des éléments les plus prometteurs, et définitivement celui que nous regrettons le plus de ne pas avoir exploré en profondeur.

D'autre part, nous devions évidemment essayer de ne pas faire œuvre de dilettante et d'en arriver à une connaissance trop superficielle de l'ensemble des ressources SGDL. En ce sens et au regard du temps et des ressources qui nous étaient imparties, nous croyons en être arrivé à un compromis honnête entre une surspécialisation bornée et hermétique et une exploration intégrale mais stérile.

C'est donc avec un grand plaisir que nous avons étudié ce logiciel, ceci en raison de l'intérêt du logiciel lui–même, mais aussi en raison de la générosité et de la sympathie des diverses personnes avec lesquelles nous avons eu à travailler. C'est pourquoi, si l'occasion devait se présenter, nous serions extrêmement heureux de pouvoir travailler à nouveau en collaboration avec les membres du GRCAO, particulièrement avec Temy TIDAFI, dans un environnement aussi intéressant et stimulant.

# Annexe 1.

# Marche.scm (extraits)

## Fonction de rechargement

```
( define
  ( lambda ()
    ( begin
      ( load "marche.scm" ))))
```

## Fonction d'interface

```
(define m
  (lambda (Rx_2)
    (begin
      (PIgen ( marche Rx_2 ) "marche_vol")

      (PIreset)
      (PIsobs (vector (- 18) 9  18   1 ))
      (PIstarget ( vector 0 0 0 1 ))
      (PIswin ( vector 480 400 ))

      (PIalight
       (vector
                (vector  5  20  10  1)
                20
                (vector-ref (PIglight 0) 2)
                (vector-ref (PIglight 0) 3)))

      (PIview "marche_vol" "marche_vue_1" )
      (PIshow "marche_vue_1"))))
```

## Fonction principale

```
(define marche
  ( lambda (Rx)
    (DLint
     (Gdhxahxa … )
     (DLformz2 … )))
```

# Escalier_1.scm (extraits).

## Fonction de génération de la volée droite

```
(define volee
 ( lambda ( x y z n )
   (DLuni
    ( if (< n 2 )
          ( marche x ( * 3 n ) ( * 3 n )  )
          (DLuni
           ( marche x ( * 3 n ) ( * 3 n )  )
           (volee x y z ( - n 1 ) )
           )))))
```

## Fonction de génération de marche

```
(define marche
  ( lambda ( x y z )

    (GDhxahxa
     (vector … )
     ( vector
       (vector … ) … (vector … ))))))
```

## Autres fonctions

```
(define volee_biseau … )
(define marche_biseau …)
(define volee_spirale … )
(define marche_spirale … )
```

# Escalier_2.scm (extraits).

## Fonction d'interface

```
(define m
  (lambda ( x y z w k )
    (begin
     (PIgen
      (DLuni
        (escalier  x  y z w k )
        (axes) ) "sweep_vol") … ))))
```

## Fonction d'axes

```
(define axes
  (lambda ()
    (DLint
     ( GDhxahxa                    ;——cube des limites des axes
```

```
  ( vector 100 100 100 1 )
  ( vector
     ( vector … ) … ) )
 (DLuni                          ;———-axes
  (DLformz2 … )
  (DLformz2 … )
  (DLformz2 … )))))
```

## Fonction de génération de l'escalier

```
( define escalier
  ( lambda ( n_marches h_marches l_marches r_nom r_den )

    ( DLint                      ;———enveloppe de l'escalier (côtés)
     ( Dldif
        ( DLformz2               ;---------côté exterieur
          ( vector
            ( vector  …  ) … ))
        ( DLformz2               ;---------côté intérieur
          ( vector
            ( vector  …  ) … )))
     ( apply                     ;—union de la liste de marches  ( infinies en largeur )
        DLuni
        (ajoute_marche n_marches h_marches  '() r_nom r_den l_marches)))))
```

## Fonction de génération de la liste de marches

```
( define ajoute_marche
  ( lambda ( n_marches h_marches liste_marches r_nom r_den l_marches)

    ( if (  = n_marches 0 )      ; ——test de récursion
         liste_marches
         ( ajoute_marche         ;———auto–appel de la fonction
           ( - n_marches 1 )
           h_marches
           ( cons                ;———nouvelle liste de marches
             (DLint              ;———-nouvelle marche
              ( DLformz2           ;--------- marche (dalles horizontales  infinies)
                 ( vector … )))

              (DLatt             ;----------transformation (rotation ou translation)
               (SDmatrep                 appliquée aux plans des contremarches
                  ( if
                    (  not ( = 0 r_den) )
                    (SGmatrot
                     ( vector … ))
                    ( SGmattrl
                      ( vector …  )))))

              ( DLdif            ;---------—plans des contremarches
                 ( DLformz2
                   ( vector … ))
                 ( DLformz2
```

```
                ( vector … )))))))

            liste_marches )          ;———ancienne liste de marches
            r_nom
            r_den
            l_marches )))))
```

# Escalier_3.scm

## Fonction principale

```
( define escalier
  ( lambda ( n_marches h_marches l_marches r_nom r_den )
     ( apply
        DLuni
        ( construction_marches
          '() n_marches
          ( ajoute_contremarche n_marches
                ( / (- 64 ( * 2 h_marches )) 100 )  '() r_nom r_den )
          ( ajoute_plan n_marches h_marches  '() )
          ( ajoute_cotes '() l_marches r_nom r_den )))))
```

## Fonction de construction de la liste de marches

```
( define construction_marches
  ( lambda
     ( liste_marches n_marches
        liste_contremarches liste_plans liste_cotes )
    ( if ( = n_marches 0 )
         liste_marches
         ( construction_marches
           ( cons
             ( DLint
               ( list-ref liste_contremarches ( - n_marches 1 ) )
               ( list-ref liste_plans ( - n_marches 1 ) )
               ( list-ref liste_cotes 0 )
               ) liste_marches
            ) ( - n_marches 1 )
              liste_contremarches liste_plans liste_cotes))))
```

## Fonction de génération de la liste de marches

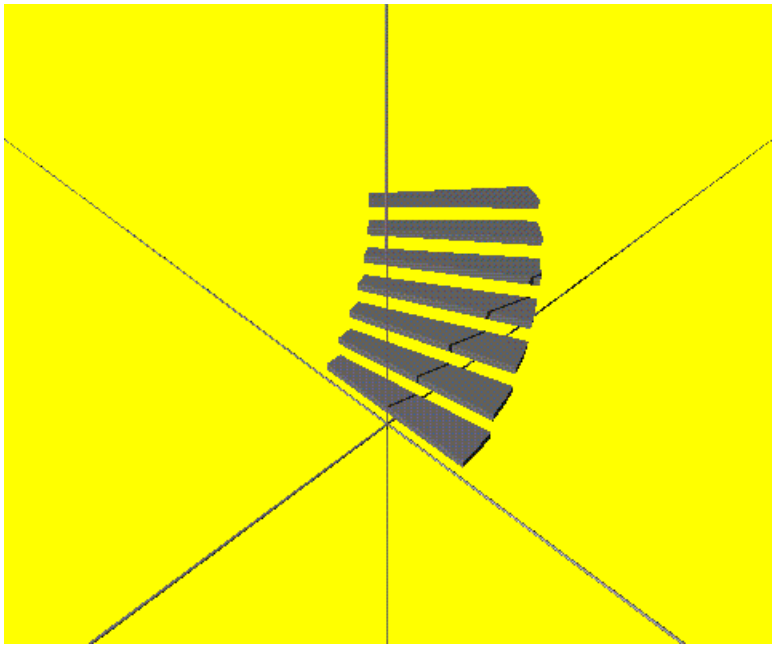
```
( define ajoute_plan
  ( lambda ( n_plans h_plans liste_plans )
    ( if (  = n_plans 0 )
         liste_plans
         ( ajoute_plan
           ( - n_plans 1 ) h_plans
           ( cons
              ( DLformz2
                  ( vector … ))
             liste_plans )))))
```

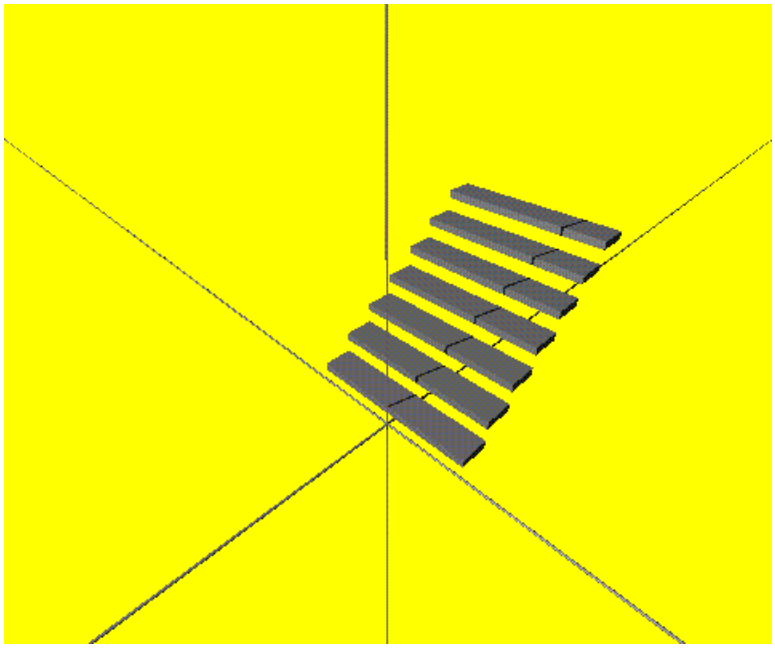
## Fonction de génération de la liste de contremarches

```
( define ajoute_contremarche
  ( lambda ( n_contremarches g_marches liste_contremarches r_nom r_den)
    ( if ( = n_contremarches 0 )
         liste_contremarches
         ( ajoute_contremarche
           ( - n_contremarches 1 ) g_marches
           ( cons
             (DLatt              ;————————transformations
              (SDmatrep
               ( if
                   (  not ( = 0 r_den) )
                   (SGmatrot
                    ( vector …  ))
                   ( SGmattrl
                     ( vector … )))))

              ( DLdif
                 ( DLformz2
                   ( vector… )
                   )
                 ( DLformz2
                   ( vector…    )) )
             ) liste_contremarches
            ) r_nom r_den
          ))))
```

## Fonction de génération de la liste des côtés

```
( define ajoute_cotes
  ( lambda ( liste_cotes l_marches r_nom r_den )
    ( cons
    ( Dldif
     ( DLformz2      ;-----------------------------------cote exterieur
        ( vector …))

     ( DLformz2      ;-----------------------------------cote interieur
        ( vector …))
     ) liste_cotes  )))
```





# Escalier_7.scm (extraits)

## Fonction principale

```
(define escalier
  (lambda ( n_marches h_marches l_marches r_nom r_den )

    (apply
      DLuni
      (map
        fonction_marche          ;————fonction de génération des volumes

        (boite '()               ;————-fonction de génération de la liste de
               n_marches                 vecteurs représentant les marches.
               l_marches
               h_marches
               r_nom
               r_den ))))
```

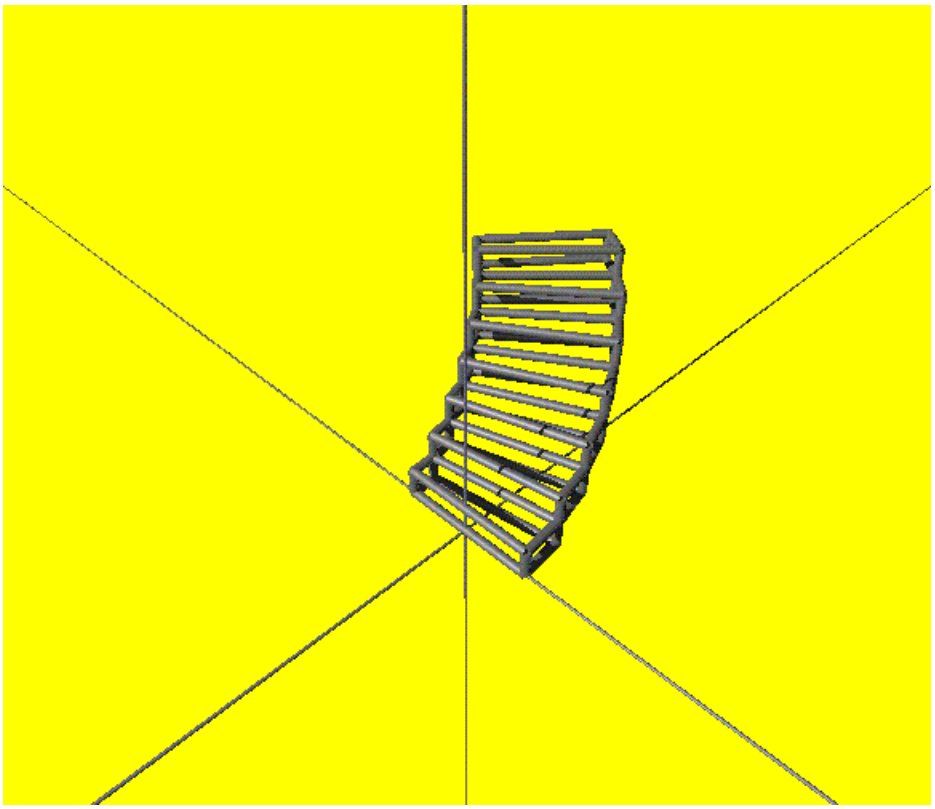
## Fonction de génération de la liste de vecteurs

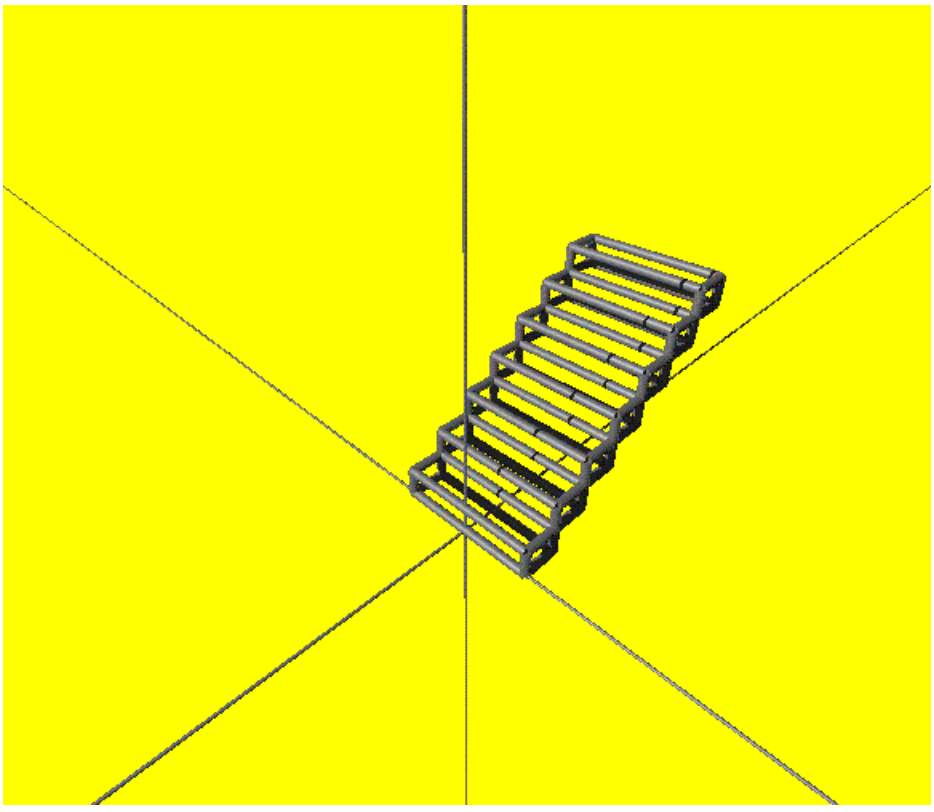
```
(define boite
  (lambda ( liste_marches_2
            n_marches
            l_marches
            h_marches
            r_nom
            r_den )
    (let                                          ;————calculs préliminaires
        ((g_marches ( / ( - 64 ( * 2 h_marches )) 100 )))
      (let (
            ( cos_angle_1
              ( cos ( / ( * n_marches g_marches r_den ) r_nom)))
            ( cos_angle_2 … ))

        (if ( = n_marches 0 )       ;————test de récursivité
            liste_marches_2
            (boite
             (cons
              ( vector              ;————-vecteur de marche
                ( vector            ;————vecteur de point
                  (- r_nom (* cos_angle_1 rayon_1))
                  hauteur_1
                  ( + longueur_1 ( *  sin_angle_1  rayon_1)) 1 )
                ( vector … )
                ( vector … )
                ( vector … )
                ( vector … )
                ( vector … )
                ( vector … )
                ( vector … ))

              liste_marches_2
              )
             ( - n_marches 1 )
```

```
l_marches
h_marches
r_nom
r_den   ))))))
```

## Fonction de construction de marche

```
(define fonction_marche
  (lambda ( tetraedre_list )
    ( DLuni
      (GDcylseg                ;————————————cylindre d'arête
       ( vector 1 30 )
       ( vector-ref tetraedre_list 0 )
       ( vector-ref tetraedre_list 1 ))
      (Gdcylseg ... )
      (Gdcylseg ... )
      (Gdcylseg ... )
      (Gdcylseg ... )
      (Gdcylseg ... )
      (Gdcylseg ... )
      (Gdcylseg ... )
      (Gdcylseg ... )
      (Gdcylseg ... )
      (Gdcylseg ... )
      (Gdcylseg ... ))))
```





# Escalier_8.scm (extraits)

## Fonction principale

```
(define escalier
  (lambda ( l_marches h_marches liste_elements )

     (fonction_marche    ;——Fonction gérant l'aspect (le style ) de l'escalier

      (construction          ;——Fonction de construction de la liste de coordonnées
       l_marches
       h_marches
       liste_elements)))     ;——La liste d'éléments passée à la fonction est
                                    traduite par les fonctions de syntaxe en liste valide
```

## Fonctions de syntaxe

```
(define p
  (lambda ( a b )
    ( list a b )))

(define m
  (lambda ( a b c )
    ( list a b c )))

(define c
  (lambda args args ))      ;——-Ces fonctions créent des listes valides à partir
                                  des entrées de l'usager.
```

## Fonctions de construction de la liste de sommets

```
(define construction
  (lambda ( l_marches h_marches liste )
    (if ( null? liste )                          ;———test de récursivité
        '()
        ( construction_2
           ( construction l_marches h_marches ( cdr liste ))
           (if ( = ( length ( car liste )) 2 )
              ( palier                           ;———fonction de création de la liste des
                 l_marches                             sommets d'un palier.
                 h_marches
                 ( list-ref ( car liste ) 0 )
                 ( list-ref ( car liste ) 1 )
                 )
              ( marche '()                       ;———fonction de création de la liste des
                 ( list-ref ( car liste ) 0 )          sommets d'une volée de marches.
                 l_marches
                 h_marches
                 ( list-ref ( car liste ) 1 )
                 ( list-ref ( car liste ) 2 )))))))


(define construction_2
```

```
(lambda ( liste_1 liste_2 )
   (if (null? liste_2 )
       liste_1
  (if (null? liste_1 )
       liste_2

       (let …)     ;————————————————————calculs préliminaires

            (append                          ;———————Cette fonction opère une fusion entre
             (translation                                deux listes, la première ayant été
                x_offset y_offset z_offset                déplacée en fonction de la position
                (rotation                                 des derniers éléments de la deuxième,
                 angle                                    au moyen des fonctions de
                 liste_1 ))))))))))                        transformations géométriques.
```

## Fonction de création de la liste des sommets d'une volée de marches

```
(define marche
  (lambda ( liste_marches_2
            n_marches
            l_marches
            h_marches
            r_nom
            r_den )
    (let
        ((g_… ))              ;————————————calculs préliminaires
     (let ( … )

        (if ( = n_marches 0 )     ;———————test de récursivité
            liste_marches_2
            (marche
             (cons
             ( vector          ;————————————vecteur des vecteurs des sommets
                 ( vector
                   (- r_nom (* cos_angle_1 rayon_1))
                   hauteur_1
                   ( + longueur_1 ( *  sin_angle_1  rayon_1)) 1 )
                 ( vector … )
                 ( vector … )
                 ( vector … )
                 ( vector … )
                 ( vector … )
                 ( vector … )
                ( vector … ))

             liste_marches_2
             )
            ( - n_marches 1 )
            l_marches
            h_marches
            r_nom
            r_den ))))))
```

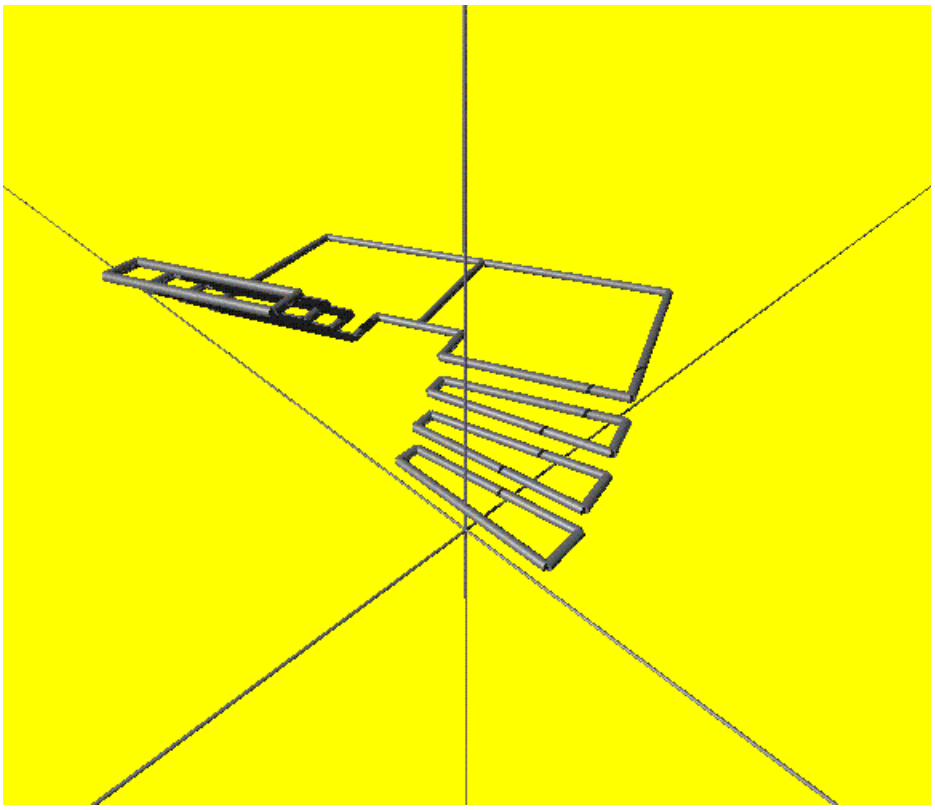
## Fonctions de création de la liste des sommets d'un palier

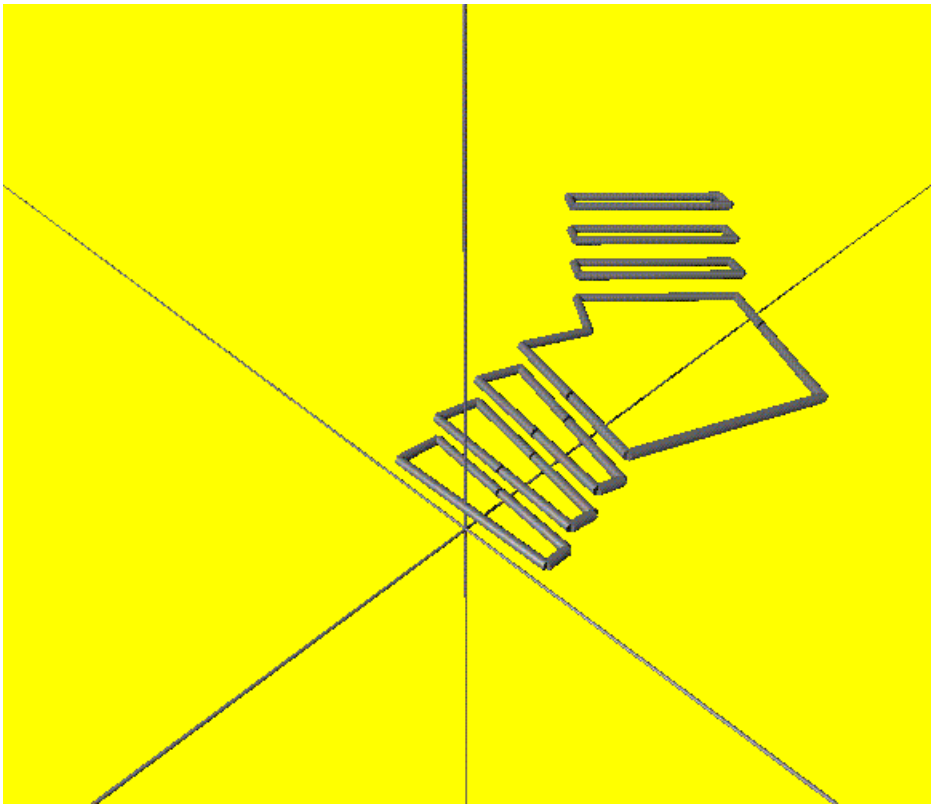
```
(define mise_a_niveau
  (lambda (   h_palier liste  )
    (if ( = 1 ( length liste ))
        liste
        (cons
         ( list-ref liste 0 )
         ( translation
           0 ( - h_palier ) 0
           (list
            ( list-ref liste 1 )))))))

(define palier
  (lambda ( l_palier h_palier g_palier angle_1   )
    (let (( angle ... ))
      (let  ( ... )          ;————————————calculs préliminaires

        (mise_a_niveau h       ;———————fonction d'ajustement de la hauteur du
                               deuxième morceau.

        (construction_2        ;———————Cette fonction est invoquée pour joindre
         (list                 les deux morceaux du palier ( s'il y en a deux ).

          ( vector         ;————————————vecteur de vecteurs des sommets du
                           premier morceau du palier
           ( vector ( + ( - l_2 ) ( * sin_1  ( + g  z_offset )))
                    h ( +  g  z_offset ( * cos_1 ( + g  z_offset ))) 1 )
           ( vector ... )
          ( vector ... )
           ( vector ... )
           ( vector ... )
           ( vector ... )
           ( vector ... )
           ( vector ... )
          ( vector ... )
          ( vector ... )
           ( vector ... )
          ( vector ... ))))

          (if ( > angle_1 ( / ( SGcst_pi ) 2 ))      ;——test de nécessité d'un deuxième
             ( list                                      morceau

              ( vector     ;————————————vecteur de vecteurs des sommets du
                           deuxième morceau du palier
                ( vector ( + ( - l_2 ) (  + g  z_offset_2 ))
                         h ( +  g  z_offset_2 ) 1 )
                ( vector ... )
                ( vector ... )
                ( vector ...  )
                ( vector ... )
                ( vector ... )
                ( vector ... )
                ( vector ... )
                ( vector ... )
                ( vector ... )
                ( vector ... )
```

```
        ( vector … )
        ( vector … )
    '()
    )))))))
```

## Fonctions de transformations géométriques

```
(define translation                ;———fonction principale de translation
  (lambda ( x y z liste )
    (if ( null? liste )
        '()
        ( cons ( vec_tra x y z ( car liste ))
               ( translation x y z  (cdr liste ))))))

(define vec_tra                    ;———-fonction adjointe de translation
  (lambda ( x y z vecteur )
    ( list->vector
      (map
       (lambda ( vecteur_2 ) ( vec_tra_2 x y z vecteur_2 ))
      ( vector->list vecteur )))))

(define vec_tra_2                  ;———-fonction adjointe de translation
  (lambda ( x y z vecteur )
    ( vector
      ( + ( vector-ref vecteur 0 ) x )
      ( + ( vector-ref vecteur 1 ) y )
      ( + ( vector-ref vecteur 2 ) z )
      1 )))

(define rotation                   ;———fonction principale de rotation
  (lambda ( angle liste )
    (if ( null? liste )
        '()
        ( cons ( vec_rot angle ( car liste ))
               ( rotation angle ( cdr liste ))))))

(define vec_rot                    ;———fonction adjointe de rotation
  (lambda ( angle vecteur )
    (let (( cos_1 ( cos angle ))
          ( sin_1 ( sin angle )))
    ( list->vector
      (map
       (lambda (  vecteur_2 ) ( vec_rot_2 cos_1 sin_1 vecteur_2 ))
      ( vector->list vecteur )
      )))))

(define vec_rot_2                  ;——— fonction adjointe de rotation
  (lambda ( cos_1 sin_1 vecteur )
   ( vector
     ( - ( * ( vector-ref vecteur 0 ) cos_1 )
          ( * ( vector-ref vecteur 2 ) sin_1 ))
     ( vector-ref vecteur  1 )
     ( + ( * ( vector-ref vecteur 0 ) sin_1 )
          ( * ( vector-ref vecteur 2 ) cos_1 ))
     1 )))
```





# Escalier_9.scm (extraits)

## Fonction principale d'application du style

```
(define fonction_marche
  (lambda ( liste )
    (apply
     DLuni
     (map
      contour
      liste
      ))))
```

## Fonction de gestion des éléments du style

```
(define contour
  (lambda ( poly_vector )
    (let (( p_0 ( vector-ref poly_vector 0 ))      ;———calculs préliminaires
          ( p_1 ( vector-ref poly_vector 1 ))
          ( p_2 ( vector-ref poly_vector 2 ))
          ( p_3 ( vector-ref poly_vector 3 ))
          ( p_4 ( vector-ref poly_vector 4 ))
          ( p_5 ( vector-ref poly_vector 5 ))
          ( p_6 ( vector-ref poly_vector 6 ))
          ( p_7 ( vector-ref poly_vector 7 )))

      (if ( = 8 ( vector-length poly_vector ))    ;———-––test déterminant si un vecteur
          (DLuni                                        représente une marche ou non.
           (rampe p_1 p_6 )
           (rampe p_7 p_0 )
           ( Dlint                                  ;———fonction d'élimination des
             (GDparseg p_0 p_4 )                          portions inutiles des coins et rebords
             (apply
              DLuni
              (filtre (list
                       (marche_prisme p_0 p_1 p_2 )  ;--fonctions de création de prismes
                       (marche_prisme p_3 p_2 p_0 )      triangulaires
                       (rebord  p_1 p_2 )
                       (rebord  p_2 p_3 )  ;———————————fonctions de création de rebords
                       (rebord  p_3 p_0 )                  au moyen de quarts de cylindres
                       (coin p_2 )
                       (coin p_3 ))))))  ;———————————————fonctions de création de coins
                                                          au moyen de huitièmes de sphères

          (DLuni                                  ;———————fonctions pour un morceau de palier
           (rampe_2 p_1 p_2 p_3)
           (rampe_2 p_4 p_5 p_0)
           (DLint
            (GDparseg p_0 p_6 )                    ;———-fonction d'élimination des
            (apply                                       portions inutiles des coins et rebords
             DLuni
             (filtre (list
                      (marche_prisme p_0 p_1 p_2 )  ;—fonctions de création de prismes
                      (marche_prisme p_0 p_2 p_5 )      triangulaires
                      (marche_prisme p_5 p_3 p_2 )
```

```
(marche_prisme p_5 p_4 p_3 )
(rebord p_1 p_2 ) ;————————————fonctions de création de rebords
(rebord p_2 p_3 )                    au moyen de quarts de cylindres
(rebord p_3 p_4 )
(rebord p_4 p_5 )
(rebord p_5 p_0 )
)))))))))
```

## Fonctions de filtration

```
(define eq_point          ;————————fonction de déplacement infinitésimal
  (lambda ( p1 p2 )
    (let (( dis (SGdisseg p1 p2 )))
     ( < ( / ( vector-ref dis 0 ) ( vector-ref dis 1))  0.01 ))))

(define filtre            ;————————fonction de filtration
  (lambda (liste)
    (if (null? liste )
    '()
    (if (equal? '() (car liste ))
        (filtre (cdr liste ))
        (cons (car liste) ( filtre (cdr liste )))
        ))))
```

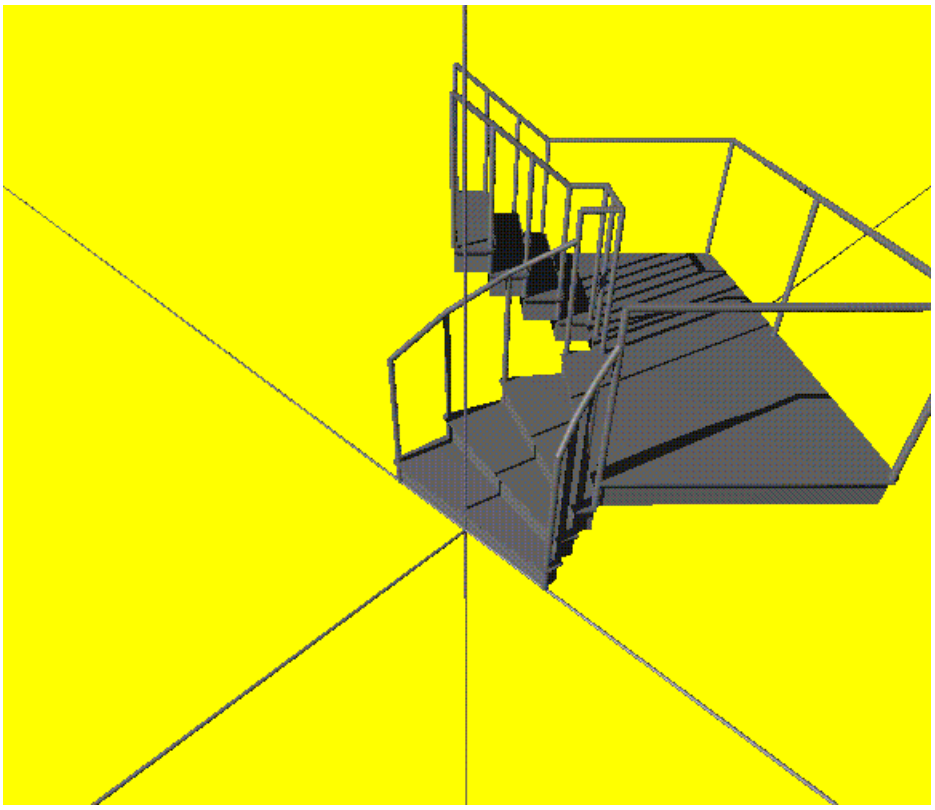
## Fonctions de création de la rampe d'escalier

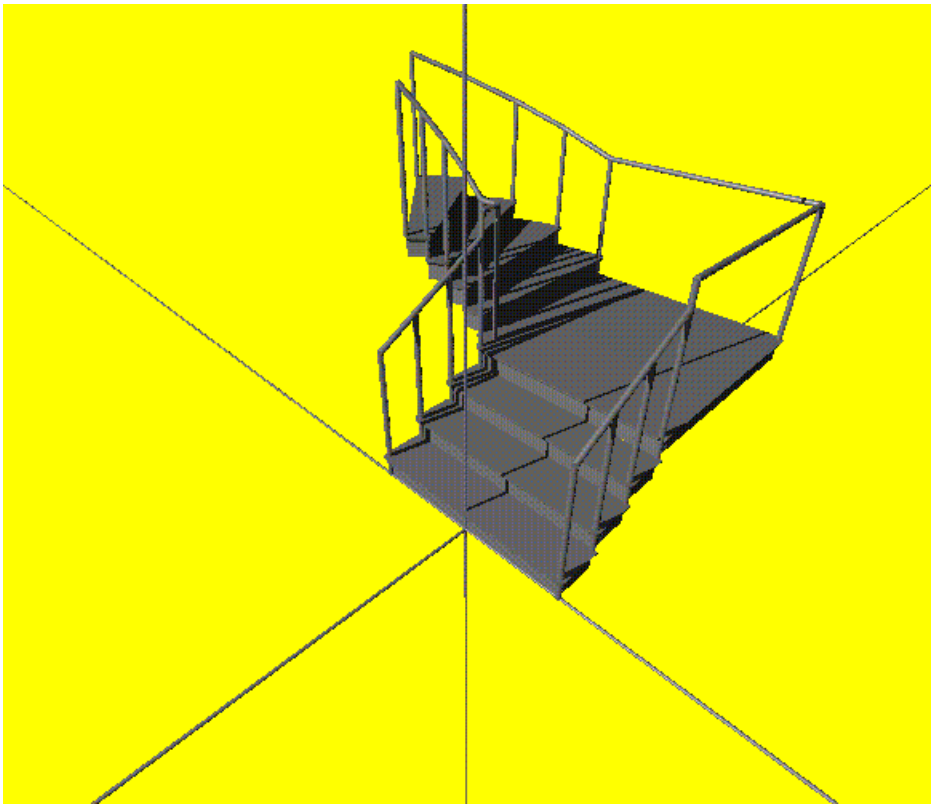
```
(define rampe                ;————————fonction pour une marche
  (lambda ( pb_1 pb_2 )
    (let (( ph_1 (h_point  pb_1 ))
          ( ph_2 (h_point  pb_2 ))
          ( rayon (vector 1 50 )))
      (if (not (equal? pb_1 pb_2) )
          (DLuni
           (GDcylseg rayon ph_1 ph_2 )
           (GDcylseg rayon pb_1 ph_1 )
           (GDcylseg rayon pb_2 ph_2 ))
          (DLuni
           (GDcylseg rayon pb_1 ph_1 )
           (GDcylseg rayon pb_2 ph_2 ))
          ))))

(define rampe_2              ;————————-fonction pour un morceau de palier
  (lambda ( pb_1 pb_2 pb_3)
    (let (( ph_1 (h_point pb_1 ))
          ( ph_2 (h_point pb_2 ))
          ( ph_3 (h_point pb_3 ))
          ( rayon (vector 1 50 )))
      (if (not (or
                (equal? pb_1 pb_2) (equal? pb_2 pb_3) (equal? pb_3 pb_1)))
      (DLuni
       (GDcylseg rayon ph_1 ph_2 )
       (GDcylseg rayon ph_2 ph_3 )
       (GDcylseg rayon pb_1 ph_1 )
       (GDcylseg rayon pb_2 ph_2 )
```

```
      (GDcylseg rayon pb_3 ph_3 ))
     (DLuni
      (GDcylseg rayon pb_1 ph_1 )
      (GDcylseg rayon pb_2 ph_2 )
      (GDcylseg rayon pb_3 ph_3 ))
     ))))

(define h_point                ;————————fonction de création des points de la
  (lambda ( p )                                partie supérieure de la rampe
    (vector
     (vector-ref p 0 )
     (+ (vector-ref p 1 ) (* (vector-ref p 3 ) 1 ))
     (vector-ref p 2 )
     (vector-ref p 3 ))))
```

## Fonctions de création des prismes, rebords et coins

```
(define marche_prisme          ;————————fonction de création des prismes
                                            triangulaires
  (lambda (  p_1 p_2 p_3 )
    (if (not (or  (eq_point p_1 p_2) (eq_point p_2 p_3) (eq_point p_3 p_1)))
        (GDpr3per
         (vector 0 1 0 0 )
         (vector
          p_1
          p_2
          p_3
          (vector  0 1  0 0 )
          ))
        '())))

(define rebord                 ;————————fonction de création des rebords
  (lambda (  p_1 p_2 )
    (if (not (eq_point p_1 p_2) )
    (GDcylseg
     (vector 1 25 )
     p_1 p_2 )
    '()
    )))

(define coin                   ;————————fonction de création  des coins
  (lambda (  p_1 )
    (GDsphdis
     (vector 1 25 )
     p_1 )))
```





# Annexe 2.

## Marche.scm

```
(define marche
  ( lambda (Rx)
    (DLint
     (GDhxahxa
      (vector 7 7 7  1 )
      ( vector
         (vector 0 0 0 1)
         (vector 1 0 0 0)
         (vector 0 0 1 0)
         (vector 0 1 0 0)))

     (DLformz2
      (vector
       ( vector(- 30) 0  0 1)
       ( vector Rx 0 0 1 )
       ( vector 0 1 0 0 )
       ( vector 0 0  Rx 1 )
       ( vector (+ 1 Rx )  Rx  0 1 )
       ( vector 0  0 Rx  2 ))))))

(define m
  (lambda (Rx_2)
    (begin
      (PIgen ( marche Rx_2 ) "marche_vol")

      (PIreset)
      (PIsobs (vector (- 18) 9  18   1 ))
      (PIstarget ( vector 0 0 0 1 ))
      (PIswin ( vector 480 400 ))

      (PIalight
       (vector
                  (vector  5  20  10  1)
                  20
                  (vector-ref (PIglight 0) 2)
                  (vector-ref (PIglight 0) 3)
                  )
       )
      (PIview "marche_vol" "marche_vue_1" )
      (PIshow "marche_vue_1")
      )
    )
  )

( define r
  ( lambda ()
    ( begin
      ( load "marche.scm" ))))
```

## Escalier_1.scm

```
;-------------------------------------------------MARCHE_SPIRALE

(define marche_spirale
  ( lambda ( x y r  n )
   ; (DLint

     (DLdif
      (GDcylhxa
       (vector x y ( + ( * 2 x ) r ) 1 )
       ( vector
          (vector ( * ( - ( + x r )) 2 )  ( - ( * 2 y ) 1 ) 0 2)
          (vector 1 0 0 0)
          (vector 0 0 1 0)
          (vector 0 1 0 0)))

      (GDcylhxa
       (vector ( - x ) y r  1 )
       ( vector
          (vector ( * ( - ( + x r )) 2 )  ( - ( * 2 y ) 1 ) 0 2)
          (vector 1 0 0 0)
          (vector 0 0 1 0)
          (vector 0 1 0 0)))
      )
    ; (DLatt
    ;  (SDmatrep
   ;    (SGmatrot
;         ( vector ( SGcst_pi ) ( / 12 n )  )
;         ( vector ( - ( + x r )) ( - 1 ) 0  1 )
;         ( vector ( - ( + x r )) 1  0  1 )

;         )
   ;    )

  ;    (DLint
  ;     (GDParseg
;         (vector 0 0 0 1  )
;         (vector  0 0 500  1 )
;         )
 ;      (DLatt
;         (SDmatrep
;          (SGmatrot
;           ( vector ( SGcst_pi ) ( - 12)  )
;           ( vector ( - ( + x r )) ( - 1 ) 0  1 )
;           ( vector ( - ( + x r )) 1  0  1 )
;           )
;          )
;         (GDParseg
;          (vector 0 0 0 1  )
;          (vector  0 0 ( - 500 )  1 )
;          ))
    ;   )
   ;   )
  ;   )
    )
```

```
  )


;-----------------------------------------------VOLEE_SPIRALE

(define volee_spirale
  ( lambda ( x r n )


    (DLuni


     ( if (< n 2 )
          ( marche_spirale x ( * 4 n ) r n  )
          (DLuni
           ( marche_spirale x ( * 4 n )  r n  )
           (volee_spirale  x r ( - n 1 ) )
           )
          )
     )


    )
  )
;-----------------------------------------------MARCHE_BISEAU

(define marche_biseau
  ( lambda ( x y z_1 z_2 )
    (DLint
     (GDhxahxa
      (vector x y z_1 1 )
      ( vector
         (vector 0 ( - ( * 2 y ) 1 )  z_1  2 )
         (vector 1 0 0 0)
         (vector 0 0 1 0)
         (vector 0 1 0 0)))

     (GDParseg
      (vector x y z_1 1  )
      (vector ( + ( - z_1 z_2 ) x ) y ( - z_1 ( * 2  x) ) 1 )
      )
     (GDParseg
      (vector ( - x ) y ( - z_2 2 ) 1  )
      (vector ( - ( - x )( - z_1 ( + z_2 2 ))) y ( + z_2 ( * 2 x )) 1)

      )
     )
    )
  )

;-----------------------------------------------VOLEE_BISEAU

(define volee_biseau
  ( lambda ( x y z n )
```

```
    (DLuni


     ( if (< n 2 )
           ( marche_biseau x ( * 4 n ) ( * 4 n ) ( * 2 n) )
           (DLuni
            ( marche_biseau x ( * 4 n ) ( * 4 n ) ( * 2 n ) )
            (volee_biseau  x y z ( - n 1 ) )
            )
           )
     )


    )
  )
;-----------------------------------------------MARCHE

(define marche
  ( lambda ( x y z )

    (GDhxahxa
     (vector x y z 1 )
     ( vector
       (vector 0 ( - ( * 2 y ) 1 ) ( + ( * 2 z ) 3 ) 2 )
       (vector 1 0 0 0)
       (vector 0 0 1 0)
       (vector 0 1 0 0)))

    )
  )

;-----------------------------------------------VOLEE

(define volee
  ( lambda ( x y z n )

    (DLuni


     ( if (< n 2 )
           ( marche x ( * 3 n ) ( * 3 n )  )
           (DLuni
            ( marche x ( * 3 n ) ( * 3 n )  )
            (volee x y z ( - n 1 ) )
            )
           )
     )

    )
  )

;-----------------------------------------------M
```

```
(define m
  (lambda ( x y z n )
    (begin
     (PIgen
      (DLuni

        (volee_spirale  x  y z )
        (axes) ) "sweep_vol")

     (PIreset)
    ; (PIsobs ( vector 0 0 28  0 ))
     (PIsobs (vector ( - 35) 55 ( - 35) 1 ))
     (PIstarget ( vector 0 25 0 1 ))
     (PIsbox ( vector 60 60 ))
     (PIswin ( vector 480 400 ))
     (PIswincol ( vector 1 1 0 ))
     (PIalight
      (vector
        (vector  5  20 ( - 10)  1)
        50
        (vector-ref (PIglight 0) 2)
        (vector-ref (PIglight 0) 3)
        )
      )
     (PIview "sweep_vol" "sweep_view" )
     (PIshow "sweep_view")
     )
    )
  )

;-----------------------------------------------RELOAD


( define r
  ( lambda ()
    ( begin
     ( load "sweep.scm" )
     )
    )
  )


;-----------------------------------------------AXES

(define axes
  (lambda ()
    (DLint
     ( GDhxahxa
       ( vector 100 100 100 1 )
      ( vector
         ( vector 0 0 0 1 )
         ( vector 1 0 0 0 )
         ( vector 0 1 0 0 )
         ( vector 0 0 1 0 )
         )
      )
```

```
(DLuni

 (DLformz2
  (vector
    ( vector(- 1) 0 0 20)
    ( vector 1 0 0 20)
    ( vector 0  1 0 0 )
    ( vector 0 0 1 0)
    ( vector 0 1 0 0 )
    ( vector 0 0 1 20) ) )

 (DLformz2
  (vector
    ( vector 0 (- 1) 0 20)
    ( vector 0 1 0 20 )
    ( vector        1 0 0 0 )
    ( vector 0 0 1 0 )
    ( vector 1 0 0 0 )
    ( vector 0 0 1 20 ) )
  )

 (DLformz2
  (vector
    ( vector (- 1) 0 0 20)
    ( vector 1 0 0 20 )
    ( vector 0 0 1 0 )
    ( vector 0 1 0 0)
    ( vector 0 0 1 0 )
    ( vector 0 1 0 20 )
    )
  )
 )
)
)
```

## Escalier_2.scm

```
;-------------------------------------------------MARCHE
( define ajoute_marche
  ( lambda ( n_marches h_marches liste_marches r_nom r_den l_marches)
    ( if (  = n_marches 0 )
         liste_marches
         ( ajoute_marche
           ( - n_marches 1 ) h_marches
           ( cons
             (DLint
              ( DLformz2                ;--------- marches
                 ( vector
                   ( vector 0 ( * n_marches  h_marches ) 0 100 )
                   ( vector 0 ( - ( * n_marches h_marches ) 7 )  0  100)
                   ( vector 1 0 0 0 )
                   ( vector 0 0 1 0 )
                   ( vector 1 0 0 0 )
                   ( vector 0 0 1 0 )
                   )
                )

              (DLatt                    ;--------------transformations
               (SDmatrep

                 ( if
                   (  not ( = 0 r_den) )
                   (SGmatrot
                    ( vector
                      ( *  ( -  n_marches 1)
                           ( atan
                             ( / ( * ( - 64 ( * 2 h_marches )) r_den )
                                 ( * 100 ( + r_nom ( * r_den l_marches )))
                                 )))  1 )

                    ( vector ( + ( * r_den l_marches ) r_nom )
                             ( - 1 ) 0  r_den )
                    ( vector  ( + ( * r_den l_marches ) r_nom )
                              1  0  r_den )
                    )

                   ( SGmattrl
                     ( vector 0 0 ( * ( - n_marches 1 )
                                      ( - 64 ( * 2 h_marches ))) 100 )
                     )
                   )
                 )
               ( DLdif




                 ( DLformz2             ;---------faces marches
                   ( vector
                     ( vector 0 0 0 1 )
```

```
                ( vector 0 0 (  -  64  ( * 2 h_marches  )) 100 )
                ( vector  ( + ( * r_den l_marches) r_nom ) 0 0 r_den )
                ( vector 0 1 0 0 )
                ( vector  ( + ( * r_den l_marches) r_nom ) 0 0 r_den )
                ( vector 0 1 0 0 )

                )
              )
            ( DLformz2
              ( vector
                ( vector ( + ( * r_den l_marches) r_nom ) 0 0 r_den )
                ( vector 1  0 0 0 )
                ( vector 0 1 0 0 )
                ( vector 0 0 1 0 )
                ( vector 0 1 0 0 )
                ( vector 0 0 1 0 )
                )
              )


            )

          )
          )
          liste_marches
          ) r_nom r_den l_marches
        )
      )
  )
 )


;-----------------------------------------------ESCALIER

( define escalier
  ( lambda ( n_marches h_marches l_marches r_nom r_den )
    ( DLint

     ( Dldif

        ( DLformz2 ;-----------------------------------cote exterieur
          ( vector
            ( vector  ( - l_marches ) 0 0 1 )
            ( vector  ( + ( * 3 l_marches r_den )
                          ( * 2 r_nom ) ) 0 0 r_den )

            ( vector 0 0 1 0 )
            ( vector 0 1 0 0 )
            ( vector  ( + ( * r_den l_marches ) r_nom ) 0
                      ( + ( * 2 l_marches r_den ) r_nom ) r_den )
            ( vector 0 1 0 0 )
            ))

        ( DLformz2 ;-----------------------------------cote interieur
          ( vector
            ( vector   l_marches  0 0 1 )
```

```
            ( vector   ( + ( * r_den l_marches ) ( * 2 r_nom )) 0 0 r_den )
            ( vector 0 0 1 0 )
            ( vector 0 1 0 0 )
            ( vector  ( + ( * r_den l_marches ) r_nom ) 0 r_nom r_den )
            ( vector 0 1 0 0 )
            ))
          )


  ;    ( DLformz2
;         ( vector
;           ( vector ( - 1 ) 0 0 1 )
;           ( vector 1 0 0 1 )
;           ( vector 0 1 0 0 )
;           ( vector 0 0 1 0 )
;           ( vector 0 1 0 0 )
;           (


      ( apply
         DLuni
         (ajoute_marche n_marches h_marches  '() r_nom r_den l_marches)))))


;-----------------------------------------------M



(define m
  (lambda ( x y z w k )
    (begin
      (PIgen
       (DLuni

         (escalier  x  y z w k )
         (axes) ) "sweep_vol")

      (PIreset)
    ;  (PIsobs ( vector  0 10  0  0 ))
      (PIsobs (vector ( - 2) 3 ( - 2) 1 ))
      (PIstarget ( vector 0 1  0 1 ))
      (PIsbox ( vector 4 4 ))
      (PIswin ( vector 480 400 ))
      (PIswincol ( vector 1 1 0 ))
      (PIalight
       (vector
         (vector  5  15  20  1)
         50
         (vector-ref (PIglight 0) 2)
         (vector-ref (PIglight 0) 3)
         )
       )
      (PIview "sweep_vol" "sweep_view" )
      (PIshow "sweep_view")
      )
    )
```

```
  )

;-----------------------------------------------RELOAD


( define r
  ( lambda ()
    ( begin
      ( load "escalier.scm" ))))


;-----------------------------------------------AXES

(define axes
  (lambda ()
    (DLint
     ( GDhxahxa
       ( vector 100 100 100 1 )
       ( vector
          ( vector 0 0 0 1 )
          ( vector 1 0 0 0 )
          ( vector 0 1 0 0 )
          ( vector 0 0 1 0 )
          )
       )
     (DLuni

      (DLformz2
       (vector
         ( vector(- 1) 0 0 100)
         ( vector 1 0 0 100)
         ( vector 0  1 0 0 )
         ( vector 0 0 1 0)
         ( vector 0 1 0 0 )
         ( vector 0 0 1  100) ) )

      (DLformz2
       (vector
         ( vector 0 (- 1) 0 100)
         ( vector 0 1 0 100 )
         ( vector         1 0 0 0 )
         ( vector 0 0 1 0 )
         ( vector 1 0 0 0 )
         ( vector 0 0 1 100 ) )
       )

      (DLformz2
       (vector
         ( vector (- 1) 0 0 100)
         ( vector 1 0 0 100 )
         ( vector 0 0 1 0 )
         ( vector 0 1 0 0)
         ( vector 0 0 1 0 )
         ( vector 0 1 0 100 )))))))
```

## Escalier_3.scm

```
;-------------------------------------------------CONTREMARCHES

( define ajoute_contremarche
  ( lambda ( n_contremarches g_marches liste_contremarches r_nom r_den)
    ( if ( = n_contremarches 0 )
         liste_contremarches
         ( ajoute_contremarche
           ( - n_contremarches 1 ) g_marches
           ( cons
             (DLatt
              (SDmatrep
               ( if
                   (  not ( = 0 r_den) )
                   (SGmatrot
                    ( vector
                      ( *  ( -  n_contremarches 1)
                           ( atan ( / ( * g_marches  r_den )  r_nom ) )) 1 )

                    ( vector   r_nom ( - 1 ) 0  r_den )
                    ( vector  r_nom   1  0  r_den )
                    )

                   ( SGmattrl
                     ( vector 0 0 ( * ( - n_contremarches 1 ) g_marches ) 1 )
                     )
                   )
               )

              ( DLdif
                 ( DLformz2
                   ( vector
                     ( vector 0 0 0 1 )
                     ( vector 0 0 g_marches  1)
                     ( vector  r_nom 0 0 r_den )
                     ( vector 0 1 0 0 )
                     ( vector  r_nom 0 0 r_den )
                     ( vector 0 1 0 0 )
                     )
                   )
                 ( DLformz2
                   ( vector
                     ( vector r_nom 0 0 r_den )
                     ( vector r_nom 0 0 0 )
                     ( vector 0 1 0 0 )
                     ( vector 0 0 1 0 )
                     ( vector 0 1 0 0 )
                     ( vector 0 0 1 0 )
                     )) )
              ) liste_contremarches
             ) r_nom r_den
           ))))


;-------------------------------------------------MARCHES
```

```
( define ajoute_plan
  ( lambda ( n_plans h_plans liste_plans )
    ( if (  = n_plans 0 )
          liste_plans
          ( ajoute_plan
            ( - n_plans 1 ) h_plans
            ( cons
              ( DLformz2
                 ( vector
                   ( vector 0 ( * n_plans  h_plans ) 0 100 )
                   ( vector 0 ( - ( * n_plans h_plans ) 7 )  0 100)
                   ( vector 1 0 0 0 )
                   ( vector 0 0 1 0 )
                   ( vector 1 0 0 0 )
                   ( vector 0 0 1 0 )
                   )
                 )
              liste_plans
              )
            )
          )
    )
  )
;-----------------------------------------------COTES
( define ajoute_cotes
  ( lambda ( liste_cotes l_marches r_nom r_den )
    ( cons
    ( Dldif
     ( DLformz2       ;-----------------------------------cote exterieur
        ( vector
          ( vector ( - ( * ( / ( abs r_nom) r_nom ) l_marches ) ) 0 0 1 )

          ( vector  ( * ( / ( abs r_nom ) r_nom )
                        ( + ( * 2 ( abs r_nom )) ( * r_den l_marches )))
                    0 0 r_den  )

          ( vector 0 0 1 0 )
          ( vector 0 1 0 0 )

          ( vector r_nom 0  ( + (  * 1 ( abs r_nom) )
                               ( * r_den l_marches ) ) r_den  )

          ( vector 0 1 0 0 )
          ))

     ( DLformz2       ;-----------------------------------cote interieur
        ( vector
          ( vector ( * ( / ( abs r_nom) r_nom ) l_marches  ) 0 0 1 )

          ( vector  ( * ( / ( abs r_nom ) r_nom )
                        ( - ( * 2 ( abs r_nom )) ( * r_den l_marches )))
                    0 0 r_den  )


          ( vector 0 0 1 0 )
```

```
          ( vector 0 1 0 0 )

          ( vector r_nom 0 ( - ( * 1 (abs r_nom )  )
                             ( * r_den l_marches ) ) r_den  )

          ( vector 0 1 0 0 )
          ))
     ) liste_cotes
   )
  )
)
;-----------------------------------------------CONSTRUCTION_MARCHES

( define construction_marches
  ( lambda
     ( liste_marches n_marches
       liste_contremarches liste_plans liste_cotes )
    ( if ( = n_marches 0 )
         liste_marches
         ( construction_marches
           ( cons
             ( DLint
               ( list-ref liste_contremarches ( - n_marches 1 ) )
               ( list-ref liste_plans ( - n_marches 1 ) )
               ( list-ref liste_cotes 0 )
               ) liste_marches
             ) ( - n_marches 1 )
               liste_contremarches liste_plans liste_cotes)
         )
    )
  )

;-----------------------------------------------ESCALIER

( define escalier
  ( lambda ( n_marches h_marches l_marches r_nom r_den )
    ( DLint
      ( apply
        DLuni
        ( construction_marches
          '() n_marches
          ( ajoute_contremarche n_marches
                ( / (- 64 ( * 2 h_marches )) 100 )  '() r_nom r_den )
          ( ajoute_plan n_marches h_marches  '() )
          ( ajoute_cotes '() l_marches r_nom r_den )
          )
        )
      )
    )
  )

;-----------------------------------------------M



(define m
```

```
  (lambda ( x y z w k )
    (begin
     (PIgen
      (DLuni

        (escalier  x  y z w k )
        (axes) ) "sweep_vol")

     (PIreset)
    ;  (PIsobs ( vector  0 10  0  0 ))
     (PIsobs (vector ( - 2) 3 ( - 2) 1 ))
     (PIstarget ( vector 0 1  0 1 ))
     (PIsbox ( vector 4 4 ))
     (PIswin ( vector 480 400 ))
     (PIswincol ( vector 1 1 0 ))
     (PIalight
      (vector
        (vector  5  15  ( - 20)  1)
        50
        (vector-ref (PIglight 0) 2)
        (vector-ref (PIglight 0) 3)
        )
      )
     (PIview "sweep_vol" "sweep_view" )
     (PIshow "sweep_view")
     )
    )
  )

;-------------------------------------------------RELOAD


( define r
  ( lambda ()
    ( begin
     ( load "esca_3.scm" )
     )
    )
  )


;-------------------------------------------------AXES

(define axes
  (lambda ()
    (DLint
     ( GDhxahxa
       ( vector 100 100 100 1 )
      ( vector
         ( vector 0 0 0 1 )
         ( vector 1 0 0 0 )
         ( vector 0 1 0 0 )
         ( vector 0 0 1 0 )
         )
      )
     (DLuni
```

```
(DLformz2
 (vector
   ( vector(- 1) 0 0 100)
   ( vector 1 0 0 100)
   ( vector 0  1 0 0 )
   ( vector 0 0 1 0)
   ( vector 0 1 0 0 )
   ( vector 0 0 1  100) ) )

(DLformz2
 (vector
   ( vector 0 (- 1) 0 100)
   ( vector 0 1 0 100 )
   ( vector        1 0 0 0 )
   ( vector 0 0 1 0 )
   ( vector 1 0 0 0 )
   ( vector 0 0 1 100 ) )
 )

(DLformz2
 (vector
   ( vector (- 1) 0 0 100)
   ( vector 1 0 0 100 )
   ( vector 0 0 1 0 )
   ( vector 0 1 0 0)
   ( vector 0 0 1 0 )
   ( vector 0 1 0 100 )
   )
 )
)
)
)
)
```

## Escalier_4.scm

```
;-------------------------------------------------CONTREMARCHES

( define ajoute_contremarche
  ( lambda ( n_contremarches g_marches liste_contremarches r_nom tour)
    ( if ( = n_contremarches 0 )
         liste_contremarches
         ( ajoute_contremarche
           ( - n_contremarches 1 ) g_marches
           ( cons
             ( DLdif

               (DLatt
                  (SDmatrep
                   (SGmatrot
                   ( vector ( -
                              ( * ( / ( SGcst_pi ) 2 )
                                   ( - 1( cos
                                     ( *
                                       ( * ( SGcst_pi )  2 )
                                       ( /  g_marches  tour )
                                         n_contremarches
                                       )
                                     ))
                                     )
                              )
                                   1 )


                    ( vector   (  +  (  *
                                    ( cos
                                     ( * n_contremarches
                                        ( * ( SGcst_pi ) 2 )
                                        ( /  g_marches  tour )
                                        )
                                        )
                                   ( / 33  4 )) 4 )
                              1 0 1 )


                   ( vector   (  +   (  *
                                ( cos
                                    ( * n_contremarches
                                       ( * ( SGcst_pi ) 2 )
                                       ( /  g_marches  tour )
                                       )
                                       )
                                  ( / 33 4 ))  4 )

                              ( - 1 ) 0 1 )

;                  ( vector 2 1 0 1 )
;                  ( vector 2 ( - 1 ) 0 1 )

                   )
```

```
 )
 ( DLformz2
  ( vector
   ( vector  0 0 0 1 )
   ( vector  0 0 ( - 1 )  0 )
   ( vector 0 1 0 0 )
   ( vector 1 0 0 0 )
   ( vector 0 1 0 0 )
   ( vector 1 0 0 0 )
   )
  )
 )


(DLatt
  (SDmatrep
   (SGmatrot
    ( vector ( -
              ( * ( / ( SGcst_pi ) 2 )
               ( - 1
                ( cos
                     ( *
                      ( * ( SGcst_pi )  2 )
                      ( /  g_marches  tour )
                      ( -  n_contremarches 1 )
                      )
                     )
                )
               )
              )
             1 )

    ( vector   ( + (  *
                  ( cos
                   ( * ( - n_contremarches 1 )
                      ( * (SGcst_pi ) 2 )
                      ( /  g_marches  tour )
                      )
                      )
                  ( / 33 4 )) 4 )
               1 0 1 )

    ( vector   ( +  ( *
                  ( cos
                   ( * (  - n_contremarches 1 )
                      ( * ( SGcst_pi ) 2 )
                      ( /  g_marches  tour )
                      )
                      )
                  ( / 33 4 )) 4 )

              ( - 1 ) 0 1 )

;              ( vector 2 1 0 1 )
;              ( vector 2 ( - 1 ) 0 1 )
```

```
                )
               )
               ( DLformz2
                 ( vector
                   ( vector  0 0 0 1 )
                   ( vector  0 0 ( - 1 )  0 )
                   ( vector 0 1 0 0 )
                   ( vector 1 0 0 0 )
                   ( vector 0 1 0 0 )
                   ( vector 1 0 0 0 )
                  )
                 )
               )
             )

            liste_contremarches
            ) r_nom tour
          ))))


;-----------------------------------------------MARCHES

( define ajoute_plan
  ( lambda ( n_plans h_plans liste_plans )
    ( if (  = n_plans 0 )
          liste_plans
          ( ajoute_plan
            ( - n_plans 1 ) h_plans
            ( cons
               ( DLformz2
                  ( vector
                    ( vector 0 ( * n_plans  h_plans ) 0 100 )
                    ( vector 0 ( - ( * n_plans h_plans ) 7 )  0 100)
                    ( vector 1 0 0 0 )
                    ( vector 0 0 1 0 )
                    ( vector 1 0 0 0 )
                    ( vector 0 0 1 0 )
                    )
                  )
            liste_plans
            )
           )
          )
    )
  )
;-----------------------------------------------COTES
( define ajoute_cotes
  ( lambda ( liste_cotes l_marches r_nom r_den )
    ( cons
    ( Dldif
     ( DLformz2       ;-----------------------------------cote exterieur
        ( vector
         ( vector ( - ( * ( / ( abs r_nom) r_nom ) l_marches ) ) 0 0 1 )

         ( vector  ( * ( / ( abs r_nom ) r_nom )
```

```
                    ( + ( * 2 ( abs r_nom )) ( * r_den l_marches )))
                0 0 r_den  )

        ( vector 0 0 1 0 )
        ( vector 0 1 0 0 )

        ( vector r_nom 0  ( + (  * ( / 7 4 ) ( abs r_nom) )
                           ( * r_den l_marches ) ) r_den  )

        ( vector 0 1 0 0 )
        ))

    ( DLformz2      ;---------------------------------cote interieur
      ( vector
        ( vector ( * ( / ( abs r_nom) r_nom ) l_marches  ) 0 0 1 )

        ( vector  ( * ( / ( abs r_nom ) r_nom )
                    ( - ( * 2 ( abs r_nom )) ( * r_den l_marches )))
                0 0 r_den  )


        ( vector 0 0 1 0 )
        ( vector 0 1 0 0 )

        ( vector r_nom 0 ( - ( * ( / 7 4 ) (abs r_nom )  )
                          ( * r_den l_marches ) ) r_den  )

        ( vector 0 1 0 0 )
        ))
    ) liste_cotes
  )
 )
)
;-----------------------------------------------CONSTRUCTION_MARCHES

( define construction_marches
 ( lambda
    ( liste_marches n_marches
      liste_contremarches liste_plans liste_cotes )
   ( if ( = n_marches 0 )
        liste_marches
        ( construction_marches
          ( cons
            ( DLint
              ( list-ref liste_contremarches ( - n_marches 1 ) )
              ( list-ref liste_plans ( - n_marches 1 ) )
              ( list-ref liste_cotes 0 )
              ) liste_marches
            ) ( - n_marches 1 )
              liste_contremarches liste_plans liste_cotes)
        )
   )
 )

;-----------------------------------------------ESCALIER
```

```
( define escalier
  ( lambda ( n_marches h_marches l_marches r_nom r_den )
    ( DLint
      ( apply
         DLuni
         ( construction_marches
           '() n_marches
           ( ajoute_contremarche n_marches
         ;         ( / (- 64 ( * 2 h_marches )) 100 )
                                  1
                   '() r_nom


                                        ( *
                                          ( SGcst_pi )
                                          ( -
                                             ( +
                                               ( * 3 4  )
                                               ( * 3 7  )
                                               )
                                             ( sqrt
                                               ( *
                                                 ( + 4
                                                      ( * 3 7 )
                                                      )
                                                 ( + 7
                                                      ( * 3  4 )
                                                      )
                                                 )
                                               )
                                             )
                                          )
                                        )
           ( ajoute_plan n_marches h_marches  '() )
           ( ajoute_cotes '() l_marches r_nom r_den )
           )
         )
      )
    )
  )

;---------------------------------------------M



(define m
  (lambda ( x y z w k )
    (begin
      (PIgen
       (DLuni

         (escalier  x  y z w k )
         (axes) ) "sweep_vol")

      (PIreset)
    ;  (PIsobs ( vector  0 10  0   0 ))
```

```
      (PIsobs (vector  ( - ( / 1 3 ) )  5  ( - 3 )  1  ))
      (PIstarget ( vector 0 0  0 1 ))
      (PIsbox ( vector 10  10  ))
      (PIswin ( vector 480 400 ))
      (PIswincol ( vector 1 1 0 ))
      (PIalight
       (vector
         (vector  5  15  10  1)
         50
         (vector-ref (PIglight 0) 2)
         (vector-ref (PIglight 0) 3)
         )
       )
      (PIview "sweep_vol" "sweep_view" )
      (PIshow "sweep_view")
      )
    )
  )

;-----------------------------------------------RELOAD


( define r
  ( lambda ()
    ( begin
      ( load "esca_4.scm" )
      )
    )
  )


;-----------------------------------------------AXES

(define axes
  (lambda ()
    (DLint
     ( GDhxahxa
       ( vector 100 100 100 1 )
       ( vector
          ( vector 0 0 0 1 )
          ( vector 1 0 0 0 )
          ( vector 0 1 0 0 )
          ( vector 0 0 1 0 )
          )
       )
     (DLuni

      (DLformz2
       (vector
         ( vector(- 1) 0 0 100)
         ( vector 1 0 0 100)
         ( vector 0  1 0 0 )
         ( vector 0 0 1 0)
         ( vector 0 1 0 0 )
         ( vector 0 0 1  100) ) )
```

```
    (DLformz2
     (vector
       ( vector 0 (- 1) 0 100)
       ( vector 0 1 0 100 )
       ( vector         1 0 0 0 )
       ( vector 0 0 1 0 )
       ( vector 1 0 0 0 )
       ( vector 0 0 1 100 ) )
     )

    (DLformz2
     (vector
       ( vector (- 1) 0 0 100)
       ( vector 1 0 0 100 )
       ( vector 0 0 1 0 )
       ( vector 0 1 0 0)
       ( vector 0 0 1 0 )
       ( vector 0 1 0 100 )
       )
     )
    )
   )
  )
)
```

## Escalier_5.scm

```
;-------------------------------------------------CONTREMARCHES

( define ajoute_contremarche
  ( lambda ( n_contremarches g_marches liste_contremarches r_nom r_den)
    ( if ( = n_contremarches 0 )
         liste_contremarches
         ( ajoute_contremarche
           ( - n_contremarches 1 ) g_marches
           ( cons
             (DLatt
              (SDmatrep
               ( if
                   (  not ( = 0 r_den) )
                   (SGmatrot
                    ( vector
                      ( *  ( -  n_contremarches 1)
                           ( atan ( / ( * g_marches  r_den )  r_nom ) )) 1 )

                    ( vector   r_nom ( - 1 ) 0  r_den )
                    ( vector  r_nom   1  0  r_den )
                    )

                   ( SGmattrl
                     ( vector 0 0 ( * ( - n_contremarches 1 ) g_marches ) 1 )
                     )
                   )
               )

              ( DLdif
                 ( DLformz2
                   ( vector
                     ( vector 0 0 0 1 )
                     ( vector 0 0 g_marches  1)
                     ( vector  r_nom 0 0 r_den )
                     ( vector 0 1 0 0 )
                     ( vector  r_nom 0 0 r_den )
                     ( vector 0 1 0 0 )
                     )
                   )
                 ( DLformz2
                   ( vector
                     ( vector r_nom 0 0 r_den )
                     ( vector r_nom 0 0 0 )
                     ( vector 0 1 0 0 )
                     ( vector 0 0 1 0 )
                     ( vector 0 1 0 0 )
                     ( vector 0 0 1 0 )
                     )) )
              ) liste_contremarches
             ) r_nom r_den
           ))))


;-------------------------------------------------MARCHES
```

```
( define ajoute_plan
  ( lambda ( n_plans h_plans liste_plans )
    ( if (  = n_plans 0 )
          liste_plans
          ( ajoute_plan
            ( - n_plans 1 ) h_plans
            ( cons
              ( DLformz2
                 ( vector
                   ( vector 0 ( * n_plans  h_plans ) 0 100 )
                   ( vector 0 ( - ( * n_plans h_plans ) 7 )  0 100)
                   ( vector 1 0 0 0 )
                   ( vector 0 0 1 0 )
                   ( vector 1 0 0 0 )
                   ( vector 0 0 1 0 )
                   )
                 )
              liste_plans
              )
            )
          )
    )
  )
;-----------------------------------------------COTES
( define ajoute_cotes
  ( lambda ( liste_cotes l_marches r_nom r_den )
    ( cons
    ( Dldif
     ( DLformz2       ;-----------------------------------cote exterieur
        ( vector
          ( vector ( - ( * ( / ( abs r_nom) r_nom ) l_marches ) ) 0 0 1 )

          ( vector  ( * ( / ( abs r_nom ) r_nom )
                        ( + ( * 2 ( abs r_nom )) ( * r_den l_marches )))
                    0 0 r_den  )

          ( vector 0 0 1 0 )
          ( vector 0 1 0 0 )

          ( vector r_nom 0  ( + (  * 1 ( abs r_nom) )
                               ( * r_den l_marches ) ) r_den  )

          ( vector 0 1 0 0 )
          ))

      ( DLformz2       ;---------------------------------cote interieur
        ( vector
          ( vector ( * ( / ( abs r_nom) r_nom ) l_marches  ) 0 0 1 )

          ( vector  ( * ( / ( abs r_nom ) r_nom )
                        ( - ( * 2 ( abs r_nom )) ( * r_den l_marches )))
                    0 0 r_den  )


          ( vector 0 0 1 0 )
```

```
          ( vector 0 1 0 0 )

          ( vector r_nom 0 ( - ( * 1 (abs r_nom )  )
                             ( * r_den l_marches ) ) r_den  )

          ( vector 0 1 0 0 )
          ))
     ) liste_cotes
   )
  )
)
;-----------------------------------------------CONSTRUCTION_MARCHES

( define construction_marches
  ( lambda
     ( liste_marches n_marches
       liste_contremarches liste_plans liste_cotes )
    ( if ( = n_marches 0 )
         liste_marches
         ( construction_marches
           ( cons
             ( DLint


               ( list-ref liste_contremarches ( - n_marches 1 ) )


               ( list-ref liste_plans ( - n_marches 1 ) )


               ( list-ref liste_cotes 0 )



               ) liste_marches
             ) ( - n_marches 1 )
               liste_contremarches liste_plans liste_cotes)
         )
    )
  )

;-----------------------------------------------ESCALIER

( define escalier
  ( lambda ( n_marches h_marches l_marches r_nom r_den )
    ( DLint
     ( apply
        DLuni
        ( construction_marches
          '() n_marches
          ( ajoute_contremarche n_marches
                ( / (- 64 ( * 2 h_marches )) 100 )  '() r_nom r_den )
          ( ajoute_plan n_marches h_marches  '() )
          ( ajoute_cotes '() l_marches r_nom r_den )
          )
        )
```

```
    )
   )
  )
;-------------------------------------------------FONCTION_MARCHE

( define fonction_marche
  ( lambda ( tetraedre_list )

    ( GDhxahxa
      ( list-ref tetraedre_list 0 )
      ( list-ref tetraedre_list 1 )
      )

    )
  )

;-------------------------------------------------ESCALIER_2

( define escalier_2
  ( lambda ( n_marches h_marches l_marches r_nom r_den )
    ( DLint
      ( apply
         DLuni
         ( map
           fonction_marche
           ( boite '()
                    n_marches
                    l_marches
                    h_marches
                    r_nom
                    r_den )

           )
         )
      )
    )
  )
;-------------------------------------------------BOITE

( define boite
  ( lambda ( liste_marches_2
             n_marches
             l_marches
             h_marches
             r_nom
             r_den )
    ( if ( = n_marches 0 )
         liste_marches_2
         ( boite
           ( cons

             ( list
               (vector  3  n_marches   4 -3 )


               (vector
```

```
                (vector  1 (  - n_marches 2 )  0 3 )
                (vector 4 0 0 1 )
                (vector 0 0 4 1 )
                (vector  0 4 0 1 )
                ) )


            liste_marches_2
            )
          ( - n_marches 1 )
          l_marches
          h_marches
          r_nom
          r_den
          )
        )
   )
  )
;-----------------------------------------------M



(define m
  (lambda ( x y z w k )
    (begin
     (PIgen
      (DLuni

        (escalier_2  x  y z w k )
        (axes) ) "sweep_vol")

     (PIreset)
    ;  (PIsobs ( vector  0 10  0  0 ))
     (PIsobs (vector ( - 2) 3 ( - 2) 1 ))
     (PIstarget ( vector 0 1  0 1 ))
     (PIsbox ( vector 4 4 ))
     (PIswin ( vector 480 400 ))
     (PIswincol ( vector 1 1 0 ))
     (PIalight
      (vector
        (vector  5  15  ( - 20)  1)
        50
        (vector-ref (PIglight 0) 2)
        (vector-ref (PIglight 0) 3)
        )
      )
     (PIview "sweep_vol" "sweep_view" )
     (PIshow "sweep_view")
     )
   )
  )

;-----------------------------------------------RELOAD


( define r
```

```
  ( lambda ()
    ( begin
      ( load "esca_5.scm" )
      )
    )
  )


;-------------------------------------------------AXES

(define axes
  (lambda ()
    (DLint
     ( GDhxahxa
       ( vector 100 100 100 1 )
       ( vector
          ( vector 0 0 0 1 )
          ( vector 1 0 0 0 )
          ( vector 0 1 0 0 )
          ( vector 0 0 1 0 )
          )
       )
     (DLuni

      (DLformz2
       (vector
         ( vector(- 1) 0 0 100)
         ( vector 1 0 0 100)
         ( vector 0  1 0 0 )
         ( vector 0 0 1 0)
         ( vector 0 1 0 0 )
         ( vector 0 0 1  100) ) )

      (DLformz2
       (vector
         ( vector 0 (- 1) 0 100)
         ( vector 0 1 0 100 )
         ( vector         1 0 0 0 )
         ( vector 0 0 1 0 )
         ( vector 1 0 0 0 )
         ( vector 0 0 1 100 ) )
       )

      (DLformz2
       (vector
         ( vector (- 1) 0 0 100)
         ( vector 1 0 0 100 )
         ( vector 0 0 1 0 )
         ( vector 0 1 0 0)
         ( vector 0 0 1 0 )
         ( vector 0 1 0 100 )
         ))))))
```

## Escalier_6.scm

```
;-----------------------------------------------FONCTION_MARCHE

(define fonction_marche
  (lambda ( tetraedre_list )

    (GDhxahxa
      ( list-ref tetraedre_list 0 )
      ( list-ref tetraedre_list 1 )
      )

    )
  )
;-----------------------------------------------FONCTION_COS
(define fonction_cos
  (lambda ( g_marches n_marches r_nom r_den )

    ( if
      ( = r_den 0 )
      r_nom
      ( * ( cos ( /
                  ( * g_marches n_marches )
                  r_nom ))
          r_nom )

      )
    )
  )

;-----------------------------------------------FONCTION_SIN
(define fonction_sin
  (lambda ( g_marches n_marches r_nom r_den )

    ( if
      ( = r_den 0 )
      (  * n_marches g_marches  )
      ( * ( sin ( /
                  ( * g_marches n_marches )
                  r_nom ))
          r_nom )
      )
    )

  )
;-----------------------------------------------BOITE

(define boite
  (lambda ( liste_marches_2
            n_marches
            l_marches
            h_marches
            r_nom
            r_den )
    (let
        ((g_marches ( / ( - 64 ( * 2 h_marches )) 100 )) )
```

```
(if ( = n_marches 0 )
     liste_marches_2
     (boite
      (cons

       (list
        (vector  ( - r_nom ( fonction_cos
                    g_marches
                    n_marches
                    ( + r_nom ( / l_marches 2 ))
                    r_den
                    ))

                 ( * h_marches n_marches ( / 1 100 ) )

                 ( fonction_sin
                    g_marches
                    n_marches
                    ( + r_nom ( / l_marches 2 ))
                    r_den
                    )
                 1  )

        (vector
         (vector  ( - r_nom ( fonction_cos
                     g_marches
                     ( - n_marches ( / 1 2 ))
                     r_nom
                     r_den
                     ))

                    ( * h_marches ( - n_marches ( / 1 2 )) ( / 1 100 ))


                 ( fonction_sin
                     g_marches
                     ( - n_marches ( / 1 2 ))
                     r_nom
                     r_den
                     )
                  1 )

         (vector  r_nom
                  ( * h_marches
                       ( - n_marches ( / 1 2 )) ( / 1 100 )
                       r_den )
                  0
                  r_den  )

         (vector ( - ( sin ( /
                           ( * g_marches n_marches )
                           r_nom )) )
                 0
                 ( - ( cos ( /
                           ( * g_marches n_marches )
                            r_nom )) )
```

```
                    0 )

               (vector  0
                        1
                        0
                        1 )
               ) )


               liste_marches_2
              )
             ( - n_marches 1 )
             l_marches
             h_marches
             r_nom
             r_den
             )
            )
      ))
  )

;---------------------------------------------ESCALIER_2

(define escalier_2
  (lambda ( n_marches h_marches l_marches r_nom r_den )
    (DLint
      (apply
         DLuni
         (map
           fonction_marche
           (boite '()
                    n_marches
                    l_marches
                    h_marches
                    r_nom
                    r_den )

           )
         )
      )
    )
  )
;---------------------------------------------M



(define m
  (lambda ( x y z w k )
    (begin
      (PIgen
       (DLuni

         (escalier_2  x  y z w k )
         (axes) ) "sweep_vol")

      (PIreset)
```

```
   ;  (PIsobs (  vector  0 10  0  0 ))
     (PIsobs (vector ( - 2) 3 ( - 2)  1  ))
     (PIstarget ( vector 0 1  0 1 ))
     (PIsbox ( vector 4 4 ))
     (PIswin ( vector 480 400 ))
     (PIswincol ( vector 1 1 0 ))
     (PIalight
      (vector
        (vector  5  15  ( - 20)  1)
        50
        (vector-ref (PIglight 0) 2)
        (vector-ref (PIglight 0) 3)
        )
      )
     (PIview "sweep_vol" "sweep_view" )
     (PIshow "sweep_view")
     )
   )
 )

;-----------------------------------------------RELOAD


( define r
  ( lambda ()
    ( begin
     ( load "esca_6.scm" )
     )
   )
 )


;-----------------------------------------------AXES

(define axes
  (lambda ()
    (DLint
    ( GDhxahxa
       ( vector 100 100 100 1 )
      ( vector
          ( vector 0 0 0 1 )
          ( vector 1 0 0 0 )
          ( vector 0 1 0 0 )
          ( vector 0 0 1 0 )
          )
      )
    (DLuni

     (DLformz2
      (vector
        ( vector(- 1) 0 0 100)
        ( vector 1 0 0 100)
        ( vector 0  1 0 0 )
        ( vector 0 0 1 0)
        ( vector 0 1 0 0 )
        ( vector 0 0 1  100) ) )
```

```
(DLformz2
 (vector
   ( vector 0 (- 1) 0 100)
   ( vector 0 1 0 100 )
   ( vector        1 0 0 0 )
   ( vector 0 0 1 0 )
   ( vector 1 0 0 0 )
   ( vector 0 0 1 100 ) )
 )

(DLformz2
 (vector
   ( vector (- 1) 0 0 100)
   ( vector 1 0 0 100 )
   ( vector 0 0 1 0 )
   ( vector 0 1 0 0)
   ( vector 0 0 1 0 )
   ( vector 0 1 0 100 )
   )
 )
)
)
)
)
```

## Escalier_7.scm

```
;-------------------------------------------------FONCTION_MARCHE

(define fonction_marche
  (lambda ( tetraedre_list )
    ( DLuni
      (GDcylseg
       ( vector 1 30 )
       ( vector-ref tetraedre_list 0 )
       ( vector-ref tetraedre_list 1 )
       )
      (GDcylseg
       ( vector 1 30 )
       ( vector-ref tetraedre_list 1 )
       ( vector-ref tetraedre_list 2 )
       )
      (GDcylseg
       ( vector 1 30 )
       ( vector-ref tetraedre_list 2 )
       ( vector-ref tetraedre_list 3 )
       )
      (GDcylseg
       ( vector 1 30 )
       ( vector-ref tetraedre_list 3 )
       ( vector-ref tetraedre_list 0 )
       )
      (GDcylseg
       ( vector 1 30 )
       ( vector-ref tetraedre_list 0 )
       ( vector-ref tetraedre_list 4 )
       )
      (GDcylseg
       ( vector 1 30 )
       ( vector-ref tetraedre_list 1 )
       ( vector-ref tetraedre_list 5 )
       )
      (GDcylseg
       ( vector 1 30 )
       ( vector-ref tetraedre_list 2 )
       ( vector-ref tetraedre_list 6 )
       )
      (GDcylseg
       ( vector 1 30 )
       ( vector-ref tetraedre_list 3 )
       ( vector-ref tetraedre_list 7 )
       )
      (GDcylseg
       ( vector 1 30 )
       ( vector-ref tetraedre_list 4 )
       ( vector-ref tetraedre_list 5 )
       )
      (GDcylseg
       ( vector 1 30 )
       ( vector-ref tetraedre_list 5 )
       ( vector-ref tetraedre_list 6 )
```

```
      )
     (GDcylseg
      ( vector 1 30 )
      ( vector-ref tetraedre_list 6 )
      ( vector-ref tetraedre_list 7 )
      )
     (GDcylseg
      ( vector 1 30 )
      ( vector-ref tetraedre_list 7 )
      ( vector-ref tetraedre_list 4 )
      )

     )
   )
 )



;-----------------------------------------------BOITE

(define boite
  (lambda ( liste_marches_2
            n_marches
            l_marches
            h_marches
            r_nom
            r_den )
    (let
        ((g_marches ( / ( - 64 ( * 2 h_marches )) 100 )))
     (let (
            ( cos_angle_1
              ( cos ( / ( * n_marches g_marches r_den ) r_nom)))
            ( cos_angle_2
              ( cos ( / ( * ( - n_marches 1 ) g_marches r_den ) r_nom )))
            ( sin_angle_1
              ( sin( / ( * n_marches g_marches r_den ) r_nom)))
            ( sin_angle_2
              (sin (/ (* ( - n_marches 1) g_marches r_den ) r_nom )))
            ( rayon_1
              ( + r_nom ( / l_marches 2 )))
            ( rayon_2
              ( - r_nom ( / l_marches 2 )))
            ( longueur_1
              (if ( = r_den 0 )
                   ( * g_marches n_marches )
                   0 ))
            ( longueur_2
              (if ( = r_den 0 )
                   ( * g_marches ( - n_marches 1 ))
                   0 ))
            ( hauteur_1 ( * n_marches h_marches ( / 1 100 )))
            ( hauteur_2 ( * ( - n_marches 1 ) h_marches ( / 1 100 )))
            )

        (if ( = n_marches 0 )
            liste_marches_2
```

```
(boite
 (cons

  ( vector
     ( vector
       (- r_nom (* cos_angle_1 rayon_1))
       hauteur_1
       ( + longueur_1 ( *  sin_angle_1  rayon_1)) 1 )

     ( vector
       (- r_nom ( * cos_angle_1 rayon_2))
       hauteur_1
       ( + longueur_1 ( *  sin_angle_1  rayon_2)) 1 )

     (vector
      (- r_nom (* cos_angle_2 rayon_2))
      hauteur_1
      ( + longueur_2 ( * sin_angle_2  rayon_2)) 1 )

     (vector
      (- r_nom (* cos_angle_2 rayon_1))
      hauteur_1
      ( + longueur_2 ( *  sin_angle_2  rayon_1)) 1 )

     (vector
      (- r_nom (* cos_angle_1 rayon_1))
      hauteur_2
      ( + longueur_1 ( *  sin_angle_1  rayon_1)) 1 )

     (vector
      (- r_nom (* cos_angle_1 rayon_2))
      hauteur_2
      ( + longueur_1( *  sin_angle_1  rayon_2)) 1 )

     (vector
      (- r_nom (* cos_angle_2 rayon_2))
      hauteur_2
      ( + longueur_2 ( *  sin_angle_2  rayon_2)) 1 )

     (vector
      (- r_nom (* cos_angle_2 rayon_1))
      hauteur_2
      ( + longueur_2 ( *  sin_angle_2  rayon_1)) 1 )

     )


  liste_marches_2
 )
 ( - n_marches 1 )
 l_marches
 h_marches
 r_nom
 r_den
 )
)
```

```
        )
      )
    )
  )

;-----------------------------------------------TEST

(define test
  (lambda ()
    ( list
      ( vector
         ( vector  4 2 6 1 )
         ( vector  6 2 4 1 )
         ( vector  4 2 2 1 )
         ( vector  1 2 2 1 )
         ( vector  4 0 6 1 )
         ( vector  6 0 4 1 )
         ( vector  4 0 2 1 )
         ( vector  1 0 2 1 )
         )
      ( vector
         ( vector 2 4 6 1 )
         ( vector 3 4 5 1 )
         ( vector 3 4 3 1 )
         ( vector 1 4 3 1 )
         ( vector 2 3 6 1 )
         ( vector 3 3 5 1 )
         ( vector 3 3 3 1 )
         ( vector 1 3 3 1 )
         )
      )
    )
  )


;-----------------------------------------------ESCALIER_2

(define escalier_2
  (lambda ( n_marches h_marches l_marches r_nom r_den )

     (apply
        DLuni
        (map
          fonction_marche
;         ( test);
          (boite '()
                  n_marches
                  l_marches
                  h_marches
                  r_nom
                  r_den )

          )
        )

    )
```

```
  )
;-----------------------------------------------M



(define m
  (lambda ( x y z w k )
    (begin
      (PIgen
       (DLuni

         (escalier_2  x  y z w k )
         (axes) ) "sweep_vol")

      (PIreset)
    ;  (PIsobs ( vector  0 10  0  0 ))
      (PIsobs (vector ( - 2) 3 ( - 2)  1  ))
      (PIstarget ( vector 0 1  0 1 ))
      (PIsbox ( vector 4 4 ))
      (PIswin ( vector 480 400 ))
      (PIswincol ( vector 1 1 0 ))
      (PIalight
       (vector
         (vector  5  15  ( - 20)  1)
         50
         (vector-ref (PIglight 0) 2)
         (vector-ref (PIglight 0) 3)
         )
       )
      (PIview "sweep_vol" "sweep_view" )
      (PIshow "sweep_vi ew")
      )
    )
  )

;-----------------------------------------------RELOAD


( define r
  ( lambda ()
    ( begin
      ( load "esca_7.scm" )
      )
    )
  )


;-----------------------------------------------AXES

(define axes
  (lambda ()
    (DLint
     ( GDhxahxa
       ( vector 100 100 100 1 )
       ( vector
          ( vector 0 0 0 1 )
```

```
        ( vector 1 0 0 0 )
        ( vector 0 1 0 0 )
        ( vector 0 0 1 0 )
        )
     )
    (DLuni

    (DLformz2
     (vector
       ( vector(- 1) 0 0 100)
       ( vector 1 0 0 100)
       ( vector 0  1 0 0 )
       ( vector 0 0 1 0)
       ( vector 0 1 0 0 )
       ( vector 0 0 1  100) ) )

    (DLformz2
     (vector
       ( vector 0 (- 1) 0 100)
       ( vector 0 1 0 100 )
       ( vector         1 0 0 0 )
       ( vector 0 0 1 0 )
       ( vector 1 0 0 0 )
       ( vector 0 0 1 100 ) )
     )

    (DLformz2
     (vector
       ( vector (- 1) 0 0 100)
       ( vector 1 0 0 100 )
       ( vector 0 0 1 0 )
       ( vector 0 1 0 0)
       ( vector 0 0 1 0 )
       ( vector 0 1 0 100 )
       )
     )
    )
   )
  )
)
```

## Escalier_8.scm

```
;-----------------------------------------------FONCTION_MARCHE
(define y_offset
  (lambda (vecteur)
    ( vector
      ( vector-ref vecteur 0 )
      ( + ( vector-ref vecteur 1 ) 1 )
      ( vector-ref vecteur 2 )
      ( vector-ref vecteur 3 )
      )
    )
  )

(define contour
  (lambda ( poly_vector )
    (if ( = 8 ( vector-length poly_vector ))
    ( DLint
      (GDparseg
       (vector-ref poly_vector 0 )
       (vector-ref poly_vector 4 ))
      (GDparseg
       (SGx__med
         (vector-ref poly_vector 0 )
         (vector-ref poly_vector 3 ))))
  ;     (SGx__med
;         (vector-ref poly_vector 1 )
;         (vector-ref poly_vector 2 )))




;      (GDcylseg
;       ( vector 1 30 )
;       ( vector-ref poly_vector 0 )
;       ( vector-ref poly_vector 1 )
;       )
;      (GDcylseg
;       ( vector 1 30 )
;       ( vector-ref poly_vector 1 )
;       ( vector-ref poly_vector 2 )
;       )
;      (GDcylseg
;       ( vector 1 30 )
;       ( vector-ref poly_vector 2 )
;       ( vector-ref poly_vector 3 )
;       )
;      (GDcylseg
;       ( vector 1 30 )
;       ( vector-ref poly_vector 3 )
;       ( vector-ref poly_vector 0 )
;       )   )
    (DLuni
     (GDcylseg
```

```
      ( vector 1 30 )
      ( vector-ref poly_vector 0 )
      ( vector-ref poly_vector 1 )
      )
     (GDcylseg
      ( vector 1 30 )
      ( vector-ref poly_vector 1 )
      ( vector-ref poly_vector 2 )
      )
     (GDcylseg
      ( vector 1 30 )
      ( vector-ref poly_vector 2 )
      ( vector-ref poly_vector 3 )
      )
     (GDcylseg
      ( vector 1 30 )
      ( vector-ref poly_vector 3 )
      ( vector-ref poly_vector 4 )
      )
     (GDcylseg
      ( vector 1 30 )
      ( vector-ref poly_vector 4 )
      ( vector-ref poly_vector 5 )
      )
     (GDcylseg
      ( vector 1 30 )
      ( vector-ref poly_vector 5 )
      ( vector-ref poly_vector 0 )
      )   )

    ;  (GDcylseg
    ;   ( vector 1 50 )
    ;   ( y_offset ( vector-ref poly_vector 0 ))
    ;   ( y_offset ( vector-ref poly_vector 7 ))
    ;   )
    ;  (GDcylseg
    ;   ( vector 1 50 )
    ;   ( y_offset ( vector-ref poly_vector 1 ))
    ;   ( y_offset ( vector-ref poly_vector 6 ))
    ;   )
     )
    )
  )

(define fonction_marche
  (lambda ( liste )

     (apply
      DLuni
      (map
       contour
       liste
       )
      )
```

```
    )
   )


;-------------------------------------------------MARCHE

(define marche
  (lambda ( liste_marches_2
            n_marches
            l_marches
            h_marches
            r_nom
            r_den )
    (let
        ((g_marches ( / ( - 64 ( * 2 h_marches )) 100 )))
      (let (
            ( cos_angle_1
              ( cos ( / ( * n_marches g_marches r_den ) r_nom)))
            ( cos_angle_2
              ( cos ( / ( * ( - n_marches 1 ) g_marches r_den ) r_nom )))
            ( sin_angle_1
              ( sin( / ( * n_marches g_marches r_den ) r_nom)))
            ( sin_angle_2
              (sin (/ (* ( - n_marches 1) g_marches r_den ) r_nom )))
            ( rayon_1
              ( + r_nom ( / l_marches 2 )))
            ( rayon_2
              ( - r_nom ( / l_marches 2 )))
            ( longueur_1
              (if ( = r_den 0 )
                  ( * g_marches n_marches )
                  0 ))
            ( longueur_2
              (if ( = r_den 0 )
                  ( * g_marches ( - n_marches 1 ))
                  0 ))
            ( hauteur_1 ( * n_marches h_marches ( / 1 100 )))
            ( hauteur_2 ( * ( - n_marches 1 ) h_marches ( / 1 100 )))
            )

        (if ( = n_marches 0 )
            liste_marches_2
            (marche
             (cons

              ( vector
                 ( vector
                   (- r_nom (* cos_angle_1 rayon_1))
                   hauteur_1
                   ( + longueur_1 ( *  sin_angle_1  rayon_1)) 1 )

                 ( vector
                   (- r_nom ( * cos_angle_1 rayon_2))
                   hauteur_1
```

```
                ( + longueur_1 ( *  sin_angle_1  rayon_2)) 1 )

               (vector
                (- r_nom (* cos_angle_2 rayon_2))
                hauteur_1
                ( + longueur_2 ( * sin_angle_2  rayon_2)) 1 )

               (vector
                (- r_nom (* cos_angle_2 rayon_1))
                hauteur_1
                ( + longueur_2 ( *  sin_angle_2  rayon_1)) 1 )

               (vector
                (- r_nom (* cos_angle_1 rayon_1))
                hauteur_2
                ( + longueur_1 ( *  sin_angle_1  rayon_1)) 1 )

               (vector
                (- r_nom (* cos_angle_1 rayon_2))
                hauteur_2
                ( + longueur_1( *  sin_angle_1  rayon_2)) 1 )

               (vector
                (- r_nom (* cos_angle_2 rayon_2))
                hauteur_2
                ( + longueur_2 ( *  sin_angle_2  rayon_2)) 1 )

               (vector
                (- r_nom (* cos_angle_2 rayon_1))
                hauteur_2
                ( + longueur_2 ( *  sin_angle_2  rayon_1)) 1 )

               )


             liste_marches_2
             )
            ( - n_marches 1 )
            l_marches
            h_marches
            r_nom
            r_den
            )
           )
        )
     )
   )
  )

;-----------------------------------------------PALIER

(define mise_a_niveau
  (lambda (   h_palier liste  )
    (if ( = 1 ( length liste ))
        liste
        (cons
```

```
        ( list-ref liste 0 )
        ( translation
          0 ( - h_palier ) 0
          (list
           ( list-ref liste 1 )
           )
          )
        )
       )
   )
 )

(define palier
  (lambda ( l_palier h_palier g_palier angle_1   )
    (let (( angle
            (if ( > angle_1 ( / ( SGcst_pi ) 2 ))
                ( - angle_1 ( / ( SGcst_pi ) 2 ))
                angle_1 )))
      (let (
            ( z_offset ( * ( tan ( / angle 2 ))  ( / l_palier 2 ) ))
            ( z_offset_2  ( / l_palier 2 ) )
            ( g ( / g_palier 2 ))
            ( sin_1 ( sin  angle ))
            ( cos_1 ( cos  angle ))
            ( l_2 ( / l_palier 2 ))
            ( h ( / h_palier 100 )))
        (mise_a_niveau h
        (construction_2
         (list
          ( vector
            ( vector ( + ( - l_2 ) ( * sin_1  ( + g  z_offset )))
                     h ( +  g  z_offset ( * cos_1 ( + g  z_offset ))) 1 )
            ( vector ( + l_2 ( * sin_1 ( - g  z_offset ))) h
                     (+ (- g  z_offset) (* cos_1 (-  g  z_offset))) 1)
            ( vector l_2  h  ( - g  z_offset ) 1 )
            ( vector l_2 h  0 1  )
            ( vector ( - l_2 )  h  0 1 )
            ( vector ( - l_2 )  h  ( + g   z_offset ) 1 )

            ( vector ( + ( - l_2 ) ( * sin_1  ( + g  z_offset )))
                     0 ( +  g  z_offset ( * cos_1 ( + g  z_offset ))) 1 )
            ( vector ( + l_2 ( * sin_1 ( - g  z_offset ))) 0
                     (+ (- g  z_offset) (* cos_1 (-  g  z_offset))) 1)
            ( vector l_2  0  ( - g  z_offset ) 1 )
            ( vector l_2 0  0 1  )
            ( vector ( - l_2 )  0  0 1 )
            ( vector ( - l_2 )  0  ( + g   z_offset ) 1 ))
          )

          (if ( > angle_1 ( / ( SGcst_pi ) 2 ))
              ( list

                ( vector
                  ( vector ( + ( - l_2 ) (  + g  z_offset_2 ))
                           h ( +  g  z_offset_2 ) 1 )
                  ( vector ( + l_2 (  - g  z_offset_2 )) h
```

```
                    ( - g  z_offset_2 ) 1)
                ( vector l_2  h  ( - g  z_offset_2 ) 1 )
                ( vector l_2 h  0 1  )
                ( vector ( - l_2 )  h  0 1 )
                ( vector ( - l_2 )  h  ( + g   z_offset_2 ) 1 )

                ( vector ( + ( - l_2 ) (  + g  z_offset_2 ))
                    0 ( +  g  z_offset_2 ) 1 )
                ( vector ( + l_2 (  - g  z_offset_2 )) 0
                    (- g  z_offset_2) 1)
                ( vector l_2  0  ( - g  z_offset_2 ) 1 )
                ( vector l_2 0  0 1  )
                ( vector ( - l_2 )  0  0 1 )
                ( vector ( - l_2 )  0  ( + g   z_offset_2 ) 1 )))
            '()
             )
          )
        )
         )
        )
     )
   )


;-------------------------------------------------VEC_ROT_2

(define vec_rot_2
  (lambda ( cos_1 sin_1 vecteur )
   ( vector
    ( - ( * ( vector-ref vecteur 0 ) cos_1 )
        ( * ( vector-ref vecteur 2 ) sin_1 ))
    ( vector-ref vecteur 1 )
    ( + ( * ( vector-ref vecteur 0 ) sin_1 )
        ( * ( vector-ref vecteur 2 ) cos_1 ))
    1 )
  )
 )


;-------------------------------------------------VEC_ROT

(define vec_rot
  (lambda ( angle vecteur )
    (let (( cos_1 ( cos angle ))
          ( sin_1 ( sin angle )))
    ( list->vector
      (map
       (lambda (  vecteur_2 ) ( vec_rot_2 cos_1 sin_1 vecteur_2 ))
     ( vector->list vecteur )
     ))
   )
  )
)
;-------------------------------------------------ROTATION

(define rotation
```

```
  (lambda ( angle liste )
    (if ( null? liste )
          '()
          ( cons ( vec_rot angle ( car liste ))
                 ( rotation angle ( cdr liste ))))))

;-------------------------------------------------VEC_TRA_2

(define vec_tra_2
  (lambda ( x y z vecteur )
    ( vector
      ( + ( vector-ref vecteur 0 ) x )
      ( + ( vector-ref vecteur 1 ) y )
      ( + ( vector-ref vecteur 2 ) z )
      1 )
    )
  )
;-------------------------------------------------VEC_TRA

(define vec_tra
  (lambda ( x y z vecteur )
    ( list->vector
      (map
       (lambda ( vecteur_2 ) ( vec_tra_2 x y z vecteur_2 ))
      ( vector->list vecteur ))
      )
    )
  )

;-------------------------------------------------TRANSLATION

(define translation
  (lambda ( x y z liste )
    (if ( null? liste )
          '()
          ( cons ( vec_tra x y z ( car liste ))
                 ( translation x y z  (cdr liste ))))))

;-------------------------------------------------CONSTRUCTION_2

(define construction_2
  (lambda ( liste_1 liste_2 )
     (if (null? liste_2 )
          liste_1
    (if (null? liste_1 )
          liste_2

          (let (( vecteur ( list-ref liste_2 ( - ( length liste_2 ) 1 ))))
            (let (
                  ( delta_z ( - ( vector-ref ( vector-ref vecteur  1 ) 2 )
                                ( vector-ref ( vector-ref vecteur  0 ) 2 )))
                  ( delta_x ( - ( vector-ref ( vector-ref vecteur  1 ) 0 )
                                ( vector-ref ( vector-ref vecteur  0 ) 0 ))))
              (let (( angle
                       ( if ( = 0 delta_x )
                            ( if ( > delta_z 0 )
```

```
                        ( / ( SGcst_pi ) 2 )
                        ( * (SGcst_pi ) ( / 3 2 )))
                    ( if ( < delta_x 0 )
                        ( + ( SGcst_pi )
                            ( atan ( / delta_z delta_x )))
                        ( atan ( / delta_z delta_x )))))

                ( x_offset
                  ( / ( + (  vector-ref ( vector-ref vecteur  0 ) 0 )
                           ( vector-ref ( vector-ref vecteur  1 ) 0 )) 2 ))
                ( y_offset
                  (  vector-ref ( vector-ref vecteur  0 ) 1 ))
                ( z_offset
                  ( / ( + (  vector-ref ( vector-ref vecteur  0 ) 2 )
                           ( vector-ref ( vector-ref vecteur  1 ) 2 )) 2 )))

            (append
             liste_2
             (translation
                x_offset y_offset z_offset
                (rotation
                 angle
                 liste_1 ))
             )
            )
           )
          )
         )
    )
  )
)
;-----------------------------------------------CONSTRUCTION

(define construction
  (lambda ( l_marches h_marches liste )
    (if ( null? liste )
        '()
        ( construction_2
           ( construction l_marches h_marches ( cdr liste ))
          (if ( = ( length ( car liste )) 2 )
              ( palier
                 l_marches
                 h_marches
                 ( list-ref ( car liste ) 0 )
                 ( list-ref ( car liste ) 1 )
                 )
              ( marche '()
                 ( list-ref ( car liste ) 0 )
                 l_marches
                 h_marches
                 ( list-ref ( car liste ) 1 )
                 ( list-ref ( car liste ) 2 )
                 )
              )
         ))))
```

```
;-----------------------------------------ESCALIER

(define escalier
  (lambda ( l_marches h_marches liste_elements )


     (fonction_marche

      (construction
       l_marches
       h_marches
       liste_elements
       )
      )


    )
   )
;-----------------------------------------SYNTAXE
(define p
  (lambda ( a b )
    ( list a b )))

(define m
  (lambda ( a b c )
    ( list a b c )))

(define c
  (lambda args args ))

;-----------------------------------------e

(define e
  (lambda ( x y z  )
    (begin
     (PIgen
      (DLuni
        (test )
;       ( escalier  x y z   )
        (axes) ) "sweep_vol")

     (PIreset)
        ;  (PIsobs ( vector  0 10  0  0 ))
     (PIsobs (vector ( - 2) 3 ( - 2)  1  ))
      (PIstarget ( vector 0 1  0 1 ))
     (PIsbox (  vector 4 4 ))
     (PIswin ( vector 480 400 ))
     (PIswincol ( vector 1 1 0 ))
     (PIalight
      (vector
        (vector  5  15  ( - 20)  1)
        50
        (vector-ref (PIglight 0) 2)
        (vector-ref (PIglight 0) 3)
        )
      )
```

```
      (PIview "sweep_vol" "sweep_view" )
      (PIshow "sweep_view")
      )
    )
  )

;-------------------------------------------------RELOAD


(define r
  (lambda ()
    (begin
      (load "esca_9.scm" )
      )
    )
  )


;-------------------------------------------------AXES

(define axes
  (lambda ()
    (DLint
     ( GDhxahxa
       ( vector 100 100 100 1 )
       ( vector
          ( vector 0 0 0 1 )
          ( vector 1 0 0 0 )
          ( vector 0 1 0 0 )
          ( vector 0 0 1 0 )
          )
       )
     (DLuni

      (DLformz2
       (vector
         ( vector(- 1) 0 0 100)
         ( vector 1 0 0 100)
         ( vector 0  1 0 0 )
         ( vector 0 0 1 0)
         ( vector 0 1 0 0 )
         ( vector 0 0 1  100) ) )

      (DLformz2
       (vector
         ( vector 0 (- 1) 0 100)
         ( vector 0 1 0 100 )
         ( vector 1 0 0 0 )
         ( vector 0 0 1 0 )
         ( vector 1 0 0 0 )
         ( vector 0 0 1 100 ) )
       )

      (DLformz2
       (vector
         ( vector (- 1) 0 0 100)
```

```
        ( vector 1 0 0 100 )
        ( vector 0 0 1 0 )
        ( vector 0 1 0 0)
        ( vector 0 0 1 0 )
        ( vector 0 1 0 100 )
        )
      )
     )
    )
   )
  )

;-----------------------------------------------------TEST
(define test
  (lambda ()
    (DLuni
     (GDpr3per
      (vector 0 1 0 0 )
      (vector
       (vector 0 (sqrt 3) 1 1 )
       (vector (sqrt 3) ( * 2 (sqrt 3 )) -1 2)
       (vector (-(sqrt 3)) (* 2 (sqrt 3)) -1 2 )
       (vector 0 1 0 0 )
       ))
     )
    )
  )
```

## Escalier_9.scm

```
;-------------------------------------------------FONCTION_MARCHE
(define eq_point
  (lambda ( p1 p2 )
    (let (( dis (SGdisseg p1 p2 )))
     ( < ( / ( vector-ref dis 0 ) ( vector-ref dis 1))  0.01 ))))

(define marche_prisme
  (lambda (  p_1 p_2 p_3 )
    (if (not (or  (eq_point p_1 p_2) (eq_point p_2 p_3) (eq_point p_3 p_1)))
         (GDpr3per
          (vector 0 1 0 0 )
          (vector
           p_1
           p_2
           p_3
           (vector  0 1  0 0 )
           ))
         '())))

(define rebord
  (lambda (  p_1 p_2 )
    (if (not (eq_point p_1 p_2) )
    (GDcylseg
     (vector 1 25 )
     p_1 p_2 )
    '()
    )))

(define coin
  (lambda (  p_1 )
    (GDsphdis
     (vector 1 25 )
     p_1 )))

(define h_point
  (lambda ( p )
    (vector
     (vector-ref p 0 )
     (+ (vector-ref p 1 ) (* (vector-ref p 3 ) 1 ))
     (vector-ref p 2 )
     (vector-ref p 3 ))))

(define rampe
  (lambda ( pb_1 pb_2 )
    (let (( ph_1 (h_point  pb_1 ))
          ( ph_2 (h_point  pb_2 ))
          ( rayon (vector 1 50 )))
      (if (not (equal? pb_1 pb_2) )
          (DLuni
           (GDcylseg rayon ph_1 ph_2 )
           (GDcylseg rayon pb_1 ph_1 )
           (GDcylseg rayon pb_2 ph_2 ))
          (DLuni
           (GDcylseg rayon pb_1 ph_1 )
```

```
          (GDcylseg rayon pb_2 ph_2 ))
         ))))

(define rampe_2
  (lambda ( pb_1 pb_2 pb_3)
    (let (( ph_1 (h_point pb_1 ))
          ( ph_2 (h_point pb_2 ))
          ( ph_3 (h_point pb_3 ))
          ( rayon (vector 1 50 )))
      (if (not (or
                (equal? pb_1 pb_2) (equal? pb_2 pb_3) (equal? pb_3 pb_1)))
      (DLuni
       (GDcylseg rayon ph_1 ph_2 )
       (GDcylseg rayon ph_2 ph_3 )
       (GDcylseg rayon pb_1 ph_1 )
       (GDcylseg rayon pb_2 ph_2 )
       (GDcylseg rayon pb_3 ph_3 ))
      (DLuni
       (GDcylseg rayon pb_1 ph_1 )
       (GDcylseg rayon pb_2 ph_2 )
       (GDcylseg rayon pb_3 ph_3 ))
      ))))

(define filtre
  (lambda (liste)
    (if (null? liste )
    '()
    (if (equal? '() (car liste ))
        (filtre (cdr liste ))
        (cons (car liste) ( filtre (cdr liste )))
        ))))



(define contour
  (lambda ( poly_vector )
    (let (( p_0 ( vector-ref poly_vector 0 ))
          ( p_1 ( vector-ref poly_vector 1 ))
          ( p_2 ( vector-ref poly_vector 2 ))
          ( p_3 ( vector-ref poly_vector 3 ))
          ( p_4 ( vector-ref poly_vector 4 ))
          ( p_5 ( vector-ref poly_vector 5 ))
          ( p_6 ( vector-ref poly_vector 6 ))
          ( p_7 ( vector-ref poly_vector 7 )))

      (if ( = 8 ( vector-length poly_vector ))
          (DLuni
           (rampe p_1 p_6 )
           (rampe p_7 p_0 )
           ( DLint
             (GDparseg p_0 p_4 )
             (apply
              DLuni
              (filtre (list
                       (marche_prisme p_0 p_1 p_2 )
```

```
                    (marche_prisme p_3 p_2 p_0 )
                    (rebord  p_1 p_2 )
                    (rebord  p_2 p_3 )
                    (rebord  p_3 p_0 )
                    (coin p_2 )
                    (coin p_3 ))))))

          (DLuni
           (rampe_2 p_1 p_2 p_3)
           (rampe_2 p_4 p_5 p_0)
           (DLint
            (GDparseg p_0 p_6 )
            (apply
             DLuni
             (filtre (list
                    (marche_prisme p_0 p_1 p_2 )
                    (marche_prisme p_0 p_2 p_5 )
                    (marche_prisme p_5 p_3 p_2 )
                    (marche_prisme p_5 p_4 p_3 )
                    (rebord p_1 p_2 )
                    (rebord p_2 p_3 )
                    (rebord p_3 p_4 )
                    (rebord p_4 p_5 )
                    (rebord p_5 p_0 )
                    )))))))))

(define fonction_marche
  (lambda ( liste )
    (apply
     DLuni
     (map
      contour
      liste
      ))))

;-----------------------------------------------MARCHE

(define marche
  (lambda ( liste_marches_2
            n_marches
            l_marches
            h_marches
            r_nom
            r_den )
    (let
        ((g_marches ( / ( - 64 ( * 2 h_marches )) 100 )))
     (let (
            ( cos_angle_1
              ( cos ( / ( * n_marches g_marches r_den ) r_nom)))
            ( cos_angle_2
              ( cos ( / ( * ( - n_marches 1 ) g_marches r_den ) r_nom )))
            ( sin_angle_1
              ( sin( / ( * n_marches g_marches r_den ) r_nom)))
            ( sin_angle_2
              (sin (/ (* ( - n_marches 1) g_marches r_den ) r_nom )))
            ( rayon_1
```

```
      ( + r_nom ( / l_marches 2 )))
    ( rayon_2
      ( - r_nom ( / l_marches 2 )))
    ( longueur_1
      (if ( = r_den 0 )
          ( * g_marches n_marches )
          0 ))
    ( longueur_2
      (if ( = r_den 0 )
          ( * g_marches ( - n_marches 1 ))
          0 ))
    ( hauteur_1 ( * n_marches h_marches ( / 1 100 )))
    ( hauteur_2 ( * ( - n_marches 1 ) h_marches ( / 1 100 )))
    )

(if ( = n_marches 0 )
    liste_marches_2
    (marche
     (cons

      ( vector
         ( vector
           (- r_nom (* cos_angle_1 rayon_1))
           hauteur_1
           ( + longueur_1 ( *  sin_angle_1  rayon_1)) 1 )

         ( vector
           (- r_nom ( * cos_angle_1 rayon_2))
           hauteur_1
           ( + longueur_1 ( *  sin_angle_1  rayon_2)) 1 )

         (vector
          (- r_nom (* cos_angle_2 rayon_2))
          hauteur_1
          ( + longueur_2 ( * sin_angle_2  rayon_2)) 1 )

         (vector
          (- r_nom (* cos_angle_2 rayon_1))
          hauteur_1
          ( + longueur_2 ( *  sin_angle_2  rayon_1)) 1 )

         (vector
          (- r_nom (* cos_angle_1 rayon_1))
          hauteur_2
          ( + longueur_1 ( *  sin_angle_1  rayon_1)) 1 )

         (vector
          (- r_nom (* cos_angle_1 rayon_2))
          hauteur_2
          ( + longueur_1( *  sin_angle_1  rayon_2)) 1 )

         (vector
          (- r_nom (* cos_angle_2 rayon_2))
          hauteur_2
          ( + longueur_2 ( *  sin_angle_2  rayon_2)) 1 )
```

```
                (vector
                 (- r_nom (* cos_angle_2 rayon_1))
                 hauteur_2
                 ( + longueur_2 ( *  sin_angle_2  rayon_1)) 1 )

                )


              liste_marches_2
              )
             ( - n_marches 1 )
             l_marches
             h_marches
             r_nom
             r_den
             )
            )
          )
       )
     )
   )

;-----------------------------------------------PALIER

(define mise_a_niveau
  (lambda (   h_palier liste  )
    (if ( = 1 ( length liste ))
        liste
        (cons
         ( list-ref liste 0 )
         ( translation
           0 ( - h_palier ) 0
           (list
            ( list-ref liste 1 )
            )
           )
         )
        )
    )
  )

(define palier
  (lambda ( l_palier h_palier g_palier angle_1   )
    (let (( angle
            (if ( > angle_1  ( / ( SGcst_pi ) 2 ))
                ( - angle_1 ( / ( SGcst_pi ) 2 ))
                angle_1 )))
      (let (
            ( z_offset ( * ( tan ( / angle 2 ))  ( / l_palier 2 ) ))
            ( z_offset_2  ( / l_palier 2 ) )
            ( g ( / g_palier 2 ))
            ( sin_1 ( sin  angle ))
            ( cos_1 ( cos  angle ))
            ( l_2 ( / l_palier 2 ))
            ( h ( / h_palier 100 )))
        (mise_a_niveau h
```

```
(construction_2
 (list
  ( vector
    ( vector ( + ( - l_2 ) ( * sin_1  ( + g  z_offset )))
             h ( +  g  z_offset ( * cos_1 ( + g  z_offset ))) 1 )
    ( vector ( + l_2 ( * sin_1 ( - g  z_offset ))) h
             (+ (- g  z_offset) (* cos_1 (-  g  z_offset))) 1)
    ( vector l_2  h  ( - g  z_offset ) 1 )
    ( vector l_2 h  0 1  )
    ( vector ( - l_2 )  h  0 1 )
    ( vector ( - l_2 )  h  ( + g   z_offset ) 1 )

    ( vector ( + ( - l_2 ) ( * sin_1  ( + g  z_offset )))
             0 ( +  g  z_offset ( * cos_1 ( + g  z_offset ))) 1 )
    ( vector ( + l_2 ( * sin_1 ( - g  z_offset ))) 0
             (+ (- g  z_offset) (* cos_1 (-  g  z_offset))) 1)
    ( vector l_2  0  ( - g  z_offset ) 1 )
    ( vector l_2 0  0 1  )
    ( vector ( - l_2 )  0  0 1 )
    ( vector ( - l_2 )  0  ( + g   z_offset ) 1 ))
  )

  (if ( > angle_1 ( / ( SGcst_pi ) 2 ))
      ( list

       ( vector
          ( vector ( + ( - l_2 ) (  + g  z_offset_2 ))
                   h ( +  g  z_offset_2 ) 1 )
          ( vector ( + l_2 (  - g  z_offset_2 )) h
                   ( - g  z_offset_2 ) 1)
          ( vector l_2  h  ( - g  z_offset_2 ) 1 )
          ( vector l_2 h  0 1  )
          ( vector ( - l_2 )  h  0 1 )
          ( vector ( - l_2 )  h  ( + g   z_offset_2 ) 1 )

          ( vector ( + ( - l_2 ) (  + g  z_offset_2 ))
                   0 ( +  g  z_offset_2 ) 1 )
          ( vector ( + l_2 (  - g  z_offset_2 )) 0
                   (- g  z_offset_2) 1)
          ( vector l_2  0  ( - g  z_offset_2 ) 1 )
          ( vector l_2 0  0 1  )
          ( vector ( - l_2 )  0  0 1 )
          ( vector ( - l_2 )  0  ( + g   z_offset_2 ) 1 )))
        '()
        )
    )
  )
 )
)


;-----------------------------------------------VEC_ROT_2

(define vec_rot_2
```

```
  (lambda ( cos_1 sin_1 vecteur )
   ( vector
    ( - ( * ( vector-ref vecteur 0 ) cos_1 )
         ( * ( vector-ref vecteur 2 ) sin_1 ))
    ( vector-ref vecteur 1 )
    ( + ( * ( vector-ref vecteur 0 ) sin_1 )
         ( * ( vector-ref vecteur 2 ) cos_1 ))
    1 )
   )
  )


;-----------------------------------------------VEC_ROT

(define vec_rot
  (lambda ( angle vecteur )
    (let (( cos_1 ( cos angle ))
          ( sin_1 ( sin angle )))
    ( list->vector
      (map
       (lambda (  vecteur_2 ) ( vec_rot_2 cos_1 sin_1 vecteur_2 ))
      ( vector->list vecteur )
      ))
    )
  )
)
;-----------------------------------------------ROTATION

(define rotation
  (lambda ( angle liste )
    (if ( null? liste )
         '()
         ( cons ( vec_rot angle ( car liste ))
                ( rotation angle ( cdr liste ))))))

;-----------------------------------------------VEC_TRA_2

(define vec_tra_2
  (lambda ( x y z vecteur )
    ( vector
      ( + ( vector-ref vecteur 0 ) x )
      ( + ( vector-ref vecteur 1 ) y )
      ( + ( vector-ref vecteur 2 ) z )
      1 )
    )
  )
;-----------------------------------------------VEC_TRA

(define vec_tra
  (lambda ( x y z vecteur )
    ( list->vector
      (map
       (lambda ( vecteur_2 ) ( vec_tra_2 x y z vecteur_2 ))
      ( vector->list vecteur ))
      )
    )
```

```
  )

;-----------------------------------------------TRANSLATION

(define translation
  (lambda ( x y z liste )
    (if ( null? liste )
        '()
        ( cons ( vec_tra x y z ( car liste ))
              ( translation x y z  (cdr liste ))))))

;-----------------------------------------------CONSTRUCTION_2

(define construction_2
  (lambda ( liste_1 liste_2 )
     (if (null? liste_2 )
         liste_1
    (if (null? liste_1 )
         liste_2

         (let (( vecteur ( list-ref liste_2 ( - ( length liste_2 ) 1 ))))
           (let (
                 ( delta_z ( - ( vector-ref ( vector-ref vecteur  1 ) 2 )
                               ( vector-ref ( vector-ref vecteur  0 ) 2 )))
                 ( delta_x ( - ( vector-ref ( vector-ref vecteur  1 ) 0 )
                               ( vector-ref ( vector-ref vecteur  0 ) 0 ))))
             (let (( angle
                      ( if ( = 0 delta_x )
                           ( if ( > delta_z 0 )
                                ( / ( SGcst_pi ) 2 )
                                ( * (SGcst_pi ) ( / 3 2 )))
                           ( if ( < delta_x 0 )
                                ( + ( SGcst_pi )
                                     ( atan ( / delta_z delta_x )))
                                ( atan ( / delta_z delta_x )))))

                    ( x_offset
                      ( / ( + (  vector-ref ( vector-ref vecteur  0 ) 0 )
                               ( vector-ref ( vector-ref vecteur  1 ) 0 )) 2 ))
                    ( y_offset
                      (  vector-ref ( vector-ref vecteur  0 ) 1 ))
                    ( z_offset
                      ( / ( + (  vector-ref ( vector-ref vecteur  0 ) 2 )
                               ( vector-ref ( vector-ref vecteur  1 ) 2 )) 2 )))

               (append
                liste_2
                (translation
                   x_offset y_offset z_offset
                   (rotation
                    angle
                    liste_1 ))
                )
               )
             )
           )
```

```
        )
    )
  )
)
;-----------------------------------------------CONSTRUCTION

(define construction
  (lambda ( l_marches h_marches liste )
    (if ( null? liste )
        '()
        ( construction_2
           ( construction l_marches h_marches ( cdr liste ))
          (if ( = ( length ( car liste )) 2 )
              ( palier
                 l_marches
                 h_marches
                 ( list-ref ( car liste ) 0 )
                 ( list-ref ( car liste ) 1 )
                 )
              ( marche '()
                 ( list-ref ( car liste ) 0 )
                 l_marches
                 h_marches
                 ( list-ref ( car liste ) 1 )
                 ( list-ref ( car liste ) 2 )
                 )
              )
         ))))

;-----------------------------------------------ESCALIER

(define escalier
  (lambda ( l_marches h_marches liste_elements )


     (fonction_marche

      (construction
       l_marches
       h_marches
       liste_elements
       )
      )


    )
   )
;-----------------------------------------------SYNTAXE
(define p
  (lambda ( a b )
    ( list a b )))

(define m
  (lambda ( a b c )
    ( list a b c )))
```

```
(define c
  (lambda args args ))

;-----------------------------------------------e

(define e
  (lambda ( x y z  )
    (begin
      (PIgen
       (DLuni
;         (test )
          ( escalier  x y z   )
          (axes) ) "sweep_vol")

      (PIreset)
         ;  (PIsobs ( vector  0 10  0  0 ))
      (PIsobs (vector ( - 2) 3 ( - 2)  1  ))
      (PIstarget ( vector 0 1  0 1 ))
      (PIsbox ( vector 4 4 ))
      (PIswin ( vector 480 400 ))
      (PIswincol ( vector 1 1 0 ))
      (PIalight
       (vector
         (vector  5  15  ( - 20)  1)
         50
         (vector-ref (PIglight 0) 2)
         (vector-ref (PIglight 0) 3)
         )
       )
      (PIview "sweep_vol" "sweep_view" )
      (PIshow "sweep_view")
      )
    )
  )

;-----------------------------------------------RELOAD


(define r
  (lambda ()
    (begin
      (load "esca_10.scm" )
      )
    )
  )


;-----------------------------------------------AXES

(define axes
  (lambda ()
    (DLint
     ( GDhxahxa
       ( vector 100 100 100 1 )
       ( vector
          ( vector 0 0 0 1 )
```

```
          ( vector 1 0 0 0 )
          ( vector 0 1 0 0 )
          ( vector 0 0 1 0 )
          )
        )
       (DLuni

       (DLformz2
        (vector
          ( vector(- 1) 0 0 100)
          ( vector 1 0 0 100)
          ( vector 0  1 0 0 )
          ( vector 0 0 1 0)
          ( vector 0 1 0 0 )
          ( vector 0 0 1  100) ) )

       (DLformz2
        (vector
          ( vector 0 (- 1) 0 100)
          ( vector 0 1 0 100 )
          ( vector 1 0 0 0 )
          ( vector 0 0 1 0 )
          ( vector 1 0 0 0 )
          ( vector 0 0 1 100 ) )
        )

       (DLformz2
        (vector
          ( vector (- 1) 0 0 100)
          ( vector 1 0 0 100 )
          ( vector 0 0 1 0 )
          ( vector 0 1 0 0)
          ( vector 0 0 1 0 )
          ( vector 0 1 0 100 )
          )
        )
       )
      )
     )

;-------------------------------------------------------TEST
(define test
  (lambda ()
    (DLuni
     (GDpr3per
      (vector 0 1 0 0 )
      (vector
       (vector 0 (sqrt 3) 1 1 )
       (vector (sqrt 3) ( * 2 (sqrt 3 )) -1 2)
       (vector (-(sqrt 3)) (* 2 (sqrt 3)) -1 2 )
       (vector -1 1  0 0 )
       ))
     (GDsphdis
      (vector 1 20 )
      (vector 0 (sqrt 3) 1 1 )
```

```
    )
   (GDsphdis
    (vector 1 20 )
    (vector (sqrt 3) ( * 2 (sqrt 3 )) -1 2)
    )
   (GDsphdis
    (vector 1 20 )
    (vector ( - (sqrt 3)) ( * 2 (sqrt 3 )) -1 2)
    )

   )
  )
)
```