Structuring 3D Geometry based on Symmetry and Instancing Information

Aurélien Martinet

INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

N° attribué par la bibliothèque
|__|__|__|__|__|__|__|__|__|__|

T H E S E

pour obtenir le grade de

DOCTEUR DE L'INP Grenoble

Spécialité : ***Mathématiques et Informatique***

préparée au laboratoire GRAVIR/IMAG-INRIA, UMR CNRS C5527,

dans le cadre de **l'Ecole Doctorale « Mathématiques,**

Sciences et Technologies de l'Information , Informatique »

présentée et soutenue publiquement

par

Aurélien MARTINET

le 2 Mars 2007

Structuring 3D Geometry based on Symmetry and Instancing Information

Directeur de thèse : François X. SILLION
Co-directeur : Cyril SOLER

JURY

M. Roger MOHR	Professeur – INP Grenoble	, Président
M. Bernard PEROCHE	Professeur – UCB Lyon	, Rapporteur
M. Thomas FUNKHOUSER	Professeur – Université de Princeton	, Rapporteur
M. François X. SILLION	Directeur de Recherche – INRIA	, Directeur de thèse
M. Cyril SOLER	Chargé de Recherche – INRIA	, Co-directeur de thèse
M. Bruno LEVY	Chargé de Recherche – INRIA	, Examinateur

Abstract

The growing complexity of the 3D data generated by various modeling techniques requires efficient algorithms to process and visualize these data. The efficiency of such algorithms is related to the underlying structure of the data: without any structuring, an algorithm has no hope to achieve sub-linear complexity which is a huge problem when processing large amount of 3D data with millions of primitives.

The central aspect of this thesis is *structural information in computer graphics*, that is the information concerning the structure of objects or scenes. A large number of information may be qualified *structural*, ranging from simple geometric features to semantic information such as: the symmetry group of an object, the parameters of a swept surface, a scene graph or even a complete semantic representation of a scene. In this thesis, we define structural information as a two-scale notion, namely the *object* and the *scene* levels: structural information at the object level captures the way an object has been modeled *i.e.* this object is a revolution surface, a Nurbs surface, ... Conversely, structural information at the scene level contains information about how objects are organized in the scene *i.e.* the hierarchy of objects in the scene or their semantic representation. We specially focus on two specific types of structural information and propose a way of structuring the scene at both the object and scene levels using the information of symmetry and instancing.

In the first part of this thesis, we propose an original method to structure the geometry at *the object level* based on the information of symmetry. The symmetry information is object-based structural information that is of great importance for many geometric tasks such as compression, mesh editing, shape matching, ... The work developed in this thesis focuses on computing the whole symmetry group of a 3D mesh *i.e.* either discrete (such as planar or rotational symmetries) or continuous (such as cylindrical or spherical symmetries). Finding all symmetries of a shape is much more difficult than simply checking whether a given transform actually is a symmetry. In particular the naive approach that would consist of checking as many potential candidates as possible to find a symmetry is far too costly. We thus need a deterministic method to find good candidates. Inspired by the work on principal component analysis, we introduce for this purpose the *generalized moment* functions of a 3D shape.

In the second part, we propose a method to structure the geometry at *the scene level* based on instancing information *i.e.* the structure of what is repeated in the scene at multiple levels of scale. The knowledge of such information can be used in many tasks such as rendering efficiency for ray-tracing or radiosity, or for mesh-editing. From the symmetry information that we have computed on the objects of the scene, we show that computing such a hierarchy is a non-trivial process and divide it in two steps: a first step which aims at discovering frequent geometric patterns *i.e.* objects or set of objects that occur multiple times in a scene and a second step that organizes these patterns into a hierarchy.

Acknowledgements

There are lots of people I would like to thank for a huge variety of reasons.

First of all, I want to express my sincere gratitude to all the members of my jury: Roger Mohr, Bernard Peroche, Thomas Funkhouser and Bruno Levy.

I would also like to thank my supervisor, Francois X. Sillion. I could not have imagined having a better advisor and mentor for my PhD, and without her common-sense, knowledge and perceptiveness i would never have finished.

Another big thank-you to Cyril for helping me in these three years. I cannot enumerate or even remember all the things I've learned with you during this PhD. Thank you again and *Ubuntu for ever !!*

More generally, i want to thank all the people of the SHOW[1] research project for my PhD funding as well as for past, current and maybe future collaborations.

I really loved working with you Nicolas *aka* H^3. You provided me great help as well as encouragements during my work and thank for you review job on this dissertation. I look forward to biking with you to the top of Alpe d'Huez !

More generally, I'd like to thanks all the members of the Artis team for their support and friendship, specially all the people i've shared an office with during these three years: Sylvain and Manuel, JC, Yannick, Stephane, Laurent, David, Alexandrina and Kartic. Special thanks to Kartic and Alexandrina who largely contribute to improve the quality of this dissertation. There are many other people that I will not list, with whom I have had useful and inspiring conversations about my work and related topics.

Another special thanks to the baby-foot team: Pascal, Laurent (Arggg), JC, Lionel and many others . . . for all the very good moments we spent during our (short) breaks.

On a different note, I would like to thank the "Bruleurs de Loups" of Grenoble for all the terrific Saturday nights we spent in the arena, supporting this great team. This PhD will always be associated to this good time.

Last but no least, i have to say a particular "thank-you" to all my friends and family, wherever they are, particularly my mum and dad; and, most importantly of all, to Sandrine, for everything.

[1] http://artis.imag.fr/Projets/Show/

4

Contents

Abstract 1

Acknowledgements 3

Extended Abstract (in French) 7

I Introduction and Preliminaries 19

1 Introduction 21
- 1.1 Structural Information . . . 21
- 1.2 Accessibility of Structural Information . . . 22
  - 1.2.1 Asset Exchange . . . 23
  - 1.2.2 Non-interactive modeling . . . 24
- 1.3 Contributions . . . 30

2 Preliminaries: Pre-processing geometry 33
- 2.1 Object Creation . . . 33
- 2.2 Results . . . 34

II Detecting Symmetries in 3D Shapes 37

3 Background and Related work 39
- 3.1 Background . . . 39
  - 3.1.1 Affine and Vector Isometry . . . 39
  - 3.1.2 Orientation-preserving ability of isometry . . . 40
  - 3.1.3 Vector Isometry in $\mathbb{R}^3$ . . . 41
  - 3.1.4 Symmetry of 3D Shapes . . . 42
  - 3.1.5 Symmetry Group . . . 42
  - 3.1.6 Geometric Congruence . . . 44
- 3.2 Related Work in Symmetry Detection . . . 44
  - 3.2.1 Symmetry as a binary feature . . . 45
  - 3.2.2 Symmetry as a continuous feature . . . 53

4 Generalized moments And Symmetries **57**
4.1 Definition . . . 57
4.2 Shape symmetries and moments . . . 57
4.3 Efficient computation . . . 58
4.4 Finding symmetries of a single shape . . . 60
4.4.1 Determination of the axis . . . 60
4.4.2 Determination of rotation parameters . . . 61
4.4.3 Filtering results . . . 62
4.4.4 Results . . . 62
4.4.5 Discussion . . . 64
4.5 Finding symmetries of groups of objects . . . 67
4.5.1 Computing the symmetries of each tile . . . 67
4.5.2 Detecting tiles congruency . . . 67
4.5.3 Algorithm for assembled objects . . . 70
4.5.4 Step-by-step example . . . 72
4.5.5 Application scenarios . . . 73
4.5.6 Computation cost . . . 76

5 Results And Applications **77**
5.1 Geometry compression and instantiation . . . 77
5.2 Mesh Editing . . . 79
5.3 Isotropic Models . . . 79

6 Summary and Discussion **83**
6.1 Summary . . . 83
6.2 Discussion . . . 84

III Hierarchical Instancing of Geometry **87**

7 Instancing and Computer Graphics **89**
7.1 Importance of Instancing Information . . . 89
7.1.1 Compression . . . 89
7.1.2 Rendering . . . 91
7.1.3 Scene Editing . . . 96
7.2 Previous Work . . . 96
7.2.1 Instancing and Linear Fractal Modeling . . . 96
7.2.2 Approximate Instancing . . . 96
7.2.3 Automatic Instancing . . . 98
7.3 Overview of our approach . . . 99

8 Frequent Geometric Patterns Discovery **101**
8.1 Generalities And Notation . . . 101
8.1.1 Problem Representation And Complexity . . . 103
8.1.2 Compressing Frequent Patterns. . . . 104
8.2 Agglomerative Approach . . . 106

8.2.1 Early pruning of nodes . . . 107
8.2.2 Lattice traversal as a tree . . . 109
8.2.3 Algorithm Overview . . . 110
8.2.4 Filtering Frequent Patterns . . . 110
8.2.5 Results . . . 111
8.3 Symmetry-Based approach . . . 113
8.3.1 Basic Statement . . . 113
8.3.2 The Transformation Space . . . 113
8.3.3 Filling the Transformation space . . . 115
8.3.4 Forming patterns . . . 119
8.3.5 Reducing the mappings set . . . 123
8.3.6 Results . . . 126

9 Building a Hierarchy of Instances **131**
9.1 Representing Hierarchies of Instances . . . 131
9.2 From Frequent Patterns to HAG . . . 132
9.2.1 HAG Construction . . . 132
9.2.2 Observations . . . 134
9.3 Deriving a usable hierarchy of instances . . . 136
9.3.1 The Maximum Weighted Independent Set . . . 136
9.3.2 Hierarchy of instances optimized for Ray-Tracing . . . 138

10 Conclusions **147**
Summary of Contributions . . . 147
Future Work and Thoughts . . . 148

A Symmetry: Proof of theorems **151**

B Proof of the Classification Theorem **157**

C Mean Shift Clustering **163**
C.1 Estimating density gradient . . . 164
C.2 Algorithm . . . 165

List of Figures **165**

List of Algorithms **169**

Bibliography **171**

Extended Abstract (in French)

Contexte et Motivation

La complexité croissante des données 3D générées par diverses techniques de modélisation nécessite des algorithmes efficaces afin de traiter ces données ou de les visualiser. L'efficacité de ces algorithmes est fortement liée *à la structure* de ces données. En effet, sans aucune structure, un algorithme ne peut espérer une complexité sous-linéaire, ce qui représente un problème conséquent lorsque les données à traiter sont constituées de millions de primitives 3D.

Dans cette thèse, on considère l'information de structure comme un outil, pouvant être utilisé, par exemple, par un artiste ou de manière plus abstraite par un algorithme pour répondre à la question: "Comment les données sont-elle structurées ?".

Une grande quantité d'informations peut être qualifiées d'information de structure, allant d'une simple caractéristique géométrique aux informations complexes de sémantiques. Parmi celles-ci, on peut citer (voir Figure 1):

- le groupe de symétrie d'un objet,
- le $L-$systeme permettant de générer un modèle naturel telle qu'une plante,
- les paramètres d'une surface de révolution,
- la paramétrisation d'une surface NURBS,
- un graphe de scène,
- une représentation sémantique complète d'une scène, capturant les fonction, les caractéristiques et les relations entre chaque objet dans la scène,
- ...

L'efficacité d'un grand nombre d'algorithmes est basée sur la connaissance d'une telle information: le groupe de symétrie d'un objet peut être utilisé pour compresser un objet 3D, ou l'éditer de manière consistante avec ses symétries; la paramétrisation d'une surface est une donnée indispensable pour texturer des objets; un graphe de scène peut être utilisé pour améliorer la vitesse de rendu d'une géométrie à l'aide de techniques de type view-frustum culling ou par l'utilisation des informations d'instantiation qu'il contient. Enfin, associer des informations sémantiques à un contenu 3D est une nécessité afin de pouvoir ré-utiliser ce contenu après l'avoir extrait d'une scène 3D.

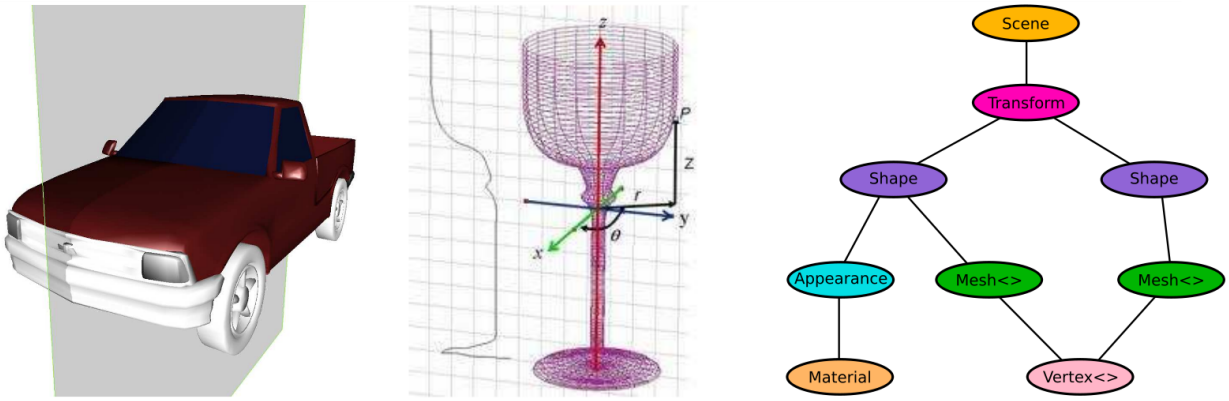


Figure 1: Quelques exemples d'information de structure. De gauche à droite: Le groupe de symétrie d'une forme permet de l'éditer ou de la compresser. La connaissance des paramètres d'une surface de révolution permet de représenter cet objet de manière concise. Le graphe de scène permet d'organiser les objets d'une scène pour, par exemple, accélérer le rendu de la scène.

La notion de structure est souvent associée à la notion d'échelle. C'est par exemple le cas dans la modélisation de scènes naturelles qui sont le plus souvent modélisées à l'aide de techniques procédurales telles que les fractales, qui sont des représentations multi-échelles par définition.

Dans ce document, nous nous inspirons de cette notion d'échelle et définissons l'information de structure comme une information à deux niveaux:

- Le niveau objet : ce type d'information capture la nature de l'objet (la façon dont un objet à été modélisé) *i.e.* cet objet est une surface de révolution, une surface NURBS, . . . ,
- le niveau scène : ces informations nous renseignent sur l'organisation des objets à l'intérieur d'une scène *i.e.* la hiérarchie des objets ou une représentation sémantique.

La connaissance de ces informations de structure étant importante pour l'efficacité de nombreux algorithmes, il semble important de se questionner sur leur accessibilité. Ce questionnement est à l'origine des travaux développés dans cette thèse et se résume par le postulat suivant:

"Dans la majorité des cas, l'information de structure n'est pas accessible"

Dans le cadre de cette thèse, nous justifions ce postulat par les deux points suivants:

1. L'information de structure était présente mais à été *perdue durant les échanges de données*.
2. Les données sont *non-structurées par nature.*

Nous donnons quelques éléments de notre réflexion ci-dessous.

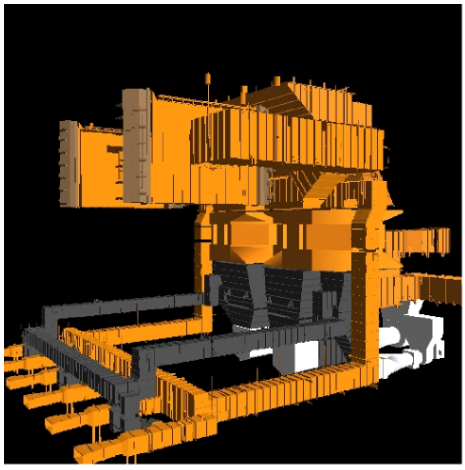
Échange de données

Les échanges de données ont été identifiées comme un problème majeur car il se situe au carrefour de plusieurs difficultés, telle que l'utilisation de multiple plat-formes, logiciels et format de fichiers.

D'un point de vue pratique, les données géométriques modélisées de manière interactive ne sont pas produites de manière séquentielle *i.e.* primitive par primitive. Ces données sont le plus souvent générées en utilisant des outils semi-automatiques telles que l'instantiation, la symétrie et les surfaces paramétriques qui forment l'information de structure associée à cette géométrie.

En théorie, ces information sont donc accessibles car faisant partie du processus de création de cette géométrie. En pratique, il arrive très souvent que cette information soit perdue lors d'une conversion d'un format de fichier à un autre. Dans la plupart des cas, cette conversion ne peut être supprimée, par exemple dans le cas ou un format de fichier particulier n'est pas lisible par l'utilisateur du modèle.

Il peut également arriver qu'une géométrie, modélisée de manière interactive par des outils standards, soit distribuée sous une forme non adaptée comme illustrée Figure 2. Dans ce cas précis, l'information de structure utilisée pour générer la géométrie est perdue car ces formats ne supportent pas le stockage d'information de structure telle que l'instantiation et la modélisation hiérarchique.



Part of the PowerPlant model

```
ply
.....
.....
0 0 0                { start of vertex list }
0 0 1
0 1 1
1 1 1
1 1 0
......
......
4 0 1 2 3            { start of face list }
4 7 6 5 4
4 0 4 5 1
4 1 5 6 2
4 2 6 7 3
4 3 7 4 0
........
<eof>
```

3D description of the model

Figure 2: La Powerplant est un modèle complexe formé de 13 millions de polygones. Une partie de ce modèle est présentée sur la gauche de la figure. Ce modèle est distribué sous la forme de fichier PLY qui n'est pas conçu pour être un format général de description de scène complexe. Ainsi, ce format ne supporte pas, par exemple, les propriétés d'instantiation, qui sont pourtant largement utilisées dans ce genre de scènes.

Enfin, dans certains cas extrêmes, les problèmes liés aux formats de fichiers sont "résolus" en transmettant les données 3D de manière incohérente sous une forme appelée *soupe de polygones* (voir Figure 3).

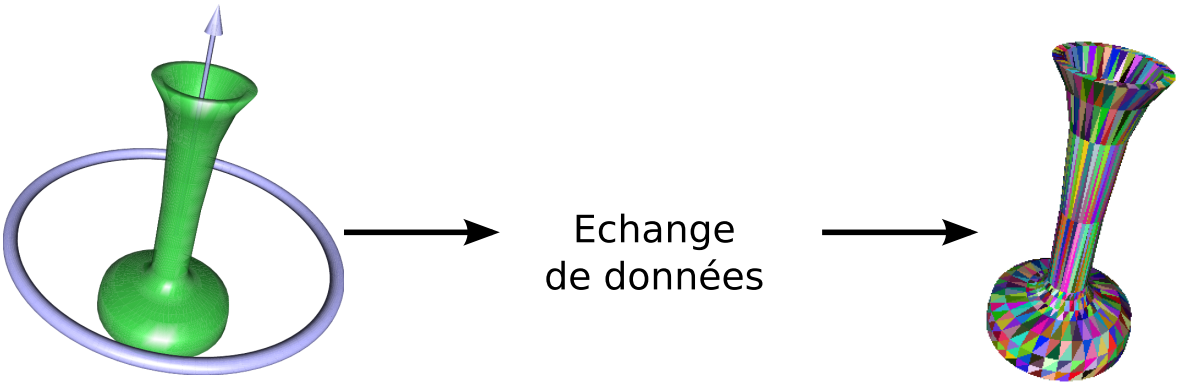


Figure 3: Les informations de structure d'un objet peuvent être perdues lors d'échanges de données. La nature d'un objet, dans ce cas une surface de révolution est perdue et l'objet est simplement décrit comme un ensemble de primitives 3D.

A partir de ces constats, deux axes de travail peuvent être adoptés:

- Un axe permettant de retrouver l'information de structure initialement présente dans la géométrie.
- Un second axe dont l'objectif serait de rendre cette information accessible tout au long de la chaîne de traitement de la géométrie *i.e.* un format d'échange fait pour le travail collaboratif.

Le travail développé dans cette thèse se concentre sur le premier point *i.e.* re-générer l'information de structure et l'utiliser pour structurer la géométrie. Le second axe de travail constitue un domaine de recherche actif en particulier avec le développement de *Collada*(**COLLA**borative **D**esign **A**ctivity).

Modélisation non-interactive

Parmi l'ensemble des méthodes de modélisation, l'avantage principal des techniques dites "non-interactive" telles que la modélisation procédurale, les techniques de scanner 3D ou la modélisation basée image est leur nature automatique, permettant la génération de modèles extrêmement détaillé qu'il serait très difficile voir impossible d'obtenir à l'aide d'outils classiques de modélisation. Cette caractéristique est essentielle car elle permet d'atteindre, par exemple, le degré de complexité présent dans les scènes naturelles (voir Figure 4).

Cependant, cette façon de générer des données constitue également le principal défaut de ces méthodes: Dans une majorité des cas, les données sont produites de manière séquentielle, *sans aucune structure* et ne contiennent donc jamais d'information sur leur nature. Dans certains cas, cette information peut être présente mais ne pas correspondre à la notion de structure voulue par l'utilisateur de la géométrie. Par exemple, lors de la modélisation procédurale d'un ville, le modeleur va "grouper" tous les objets partageant un matériel commun (par exemple la brique) et ainsi perdre l'information de redondance pouvant être présent dans la scène.

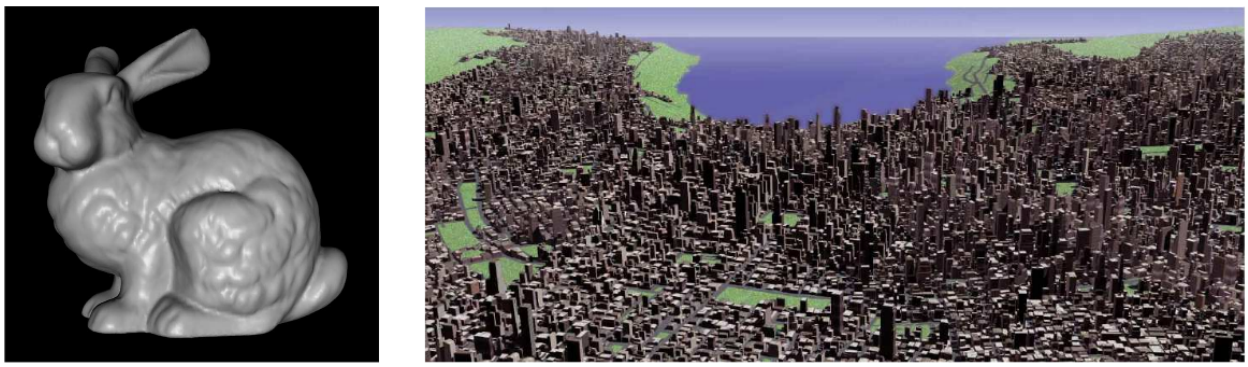


Figure 4: La modélisation non-interactive permet d'obtenir des modèles extrêmement complexes et détaillés. Généralement, ces géométries sont peu ou pas structurées ce qui peut représenter une limite à leur utilisation, par exemple (droite) pour le rendu interactif de scènes urbaines complexes.

Contributions

Le travail présenté dans cette thèse se concentre sur deux types d'information de structure et propose des méthodes permettant de structurer une scène 3D aux niveaux objets et scènes. Les deux types d'information sont les suivants:

- Symétrie des objets 3D *i.e.* information de structure au niveau objet.
- Représentation hiérarchique des instances présents dans la scène *i.e.* information de structure au niveau scène.

On qualifie les scènes que l'on cherche à traiter de *non structurées* et le but des travaux développés dans cette thèse est de structurer ces données d'entrées comme représenté sur la Figure 1.9.

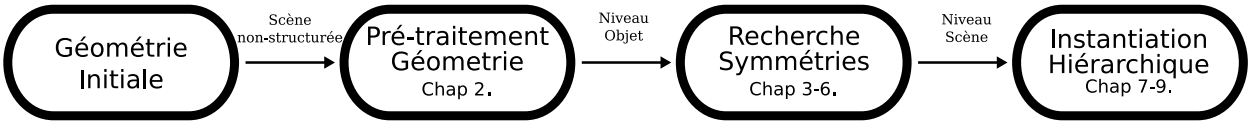
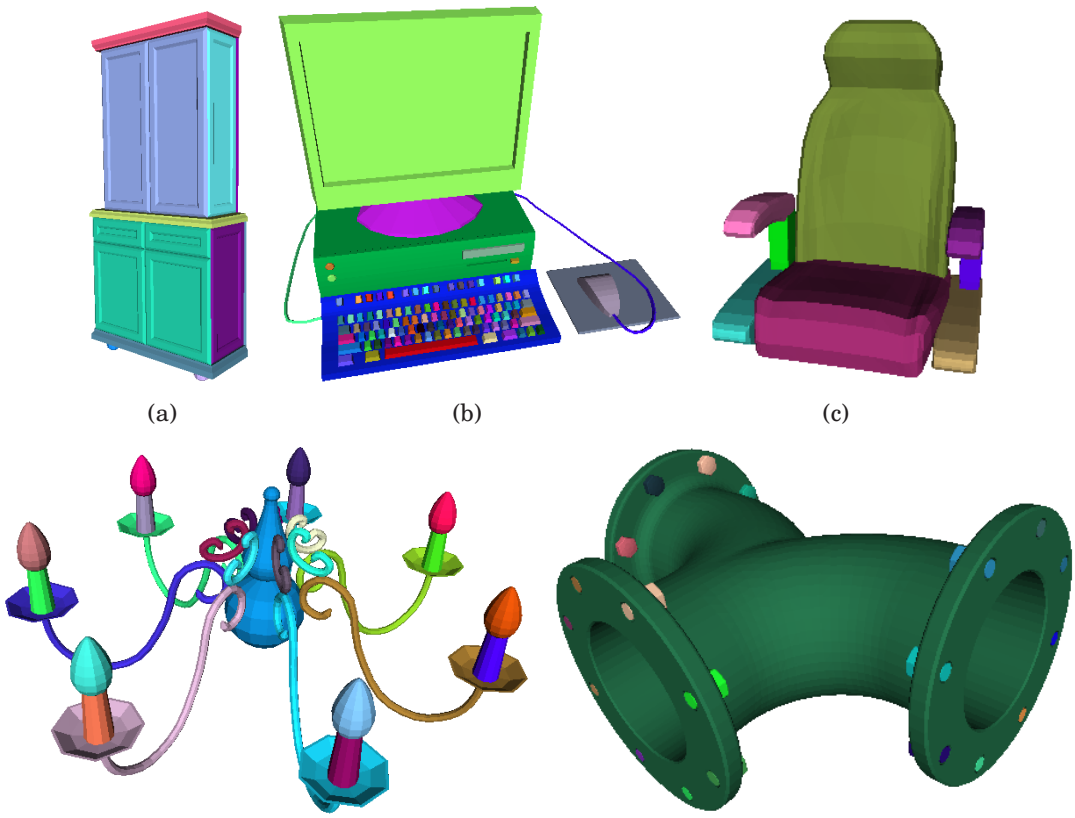


Figure 5: *Vue d'ensemble des travaux de cette thèse.* Partant d'une géométrie non structurée, nous définissons et créons des objets dans cette géométrie, calculons les symétries de chacun de ces objets, et utilisons cette information pour représenter la géométrie initiale sous la forme d'une hiérarchie d'instances.

L'aspect non structuré des scènes que l'on cherche à traiter pose le problème de définir la notion d'objet. Nous proposons un moyen simple et efficace de pré-traiter la géométrie afin de la décomposer en objets, appelés tuiles dans la suite de cette thèse. Les tuiles sont simplement définies comme les ensembles maximaux de polygones connexes par arêtes.

Des exemples de décomposition en tuiles sont présentés Figure 6.

Cette décomposition de la géométrie en tuiles, en se basant uniquement sur les informations géométriques est présentée en détail dans le chapitre 2.



(a) (b) (c)

(d) (e)

Figure 6: Quelques exemples de décomposition en tuiles d'objets simples. Pour chaque modèle, chacune de ces tuiles est représentée par une couleur distincte.

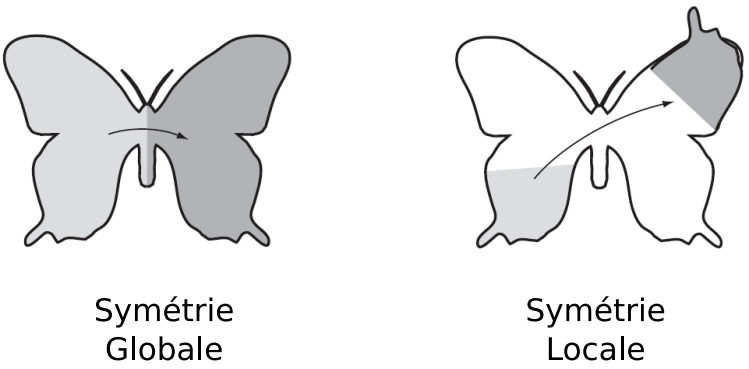
Basées sur cette définition des objets, les deux principales contributions de cette thèse sont présentées ci-dessous.

Détection de symétries des objets 3D

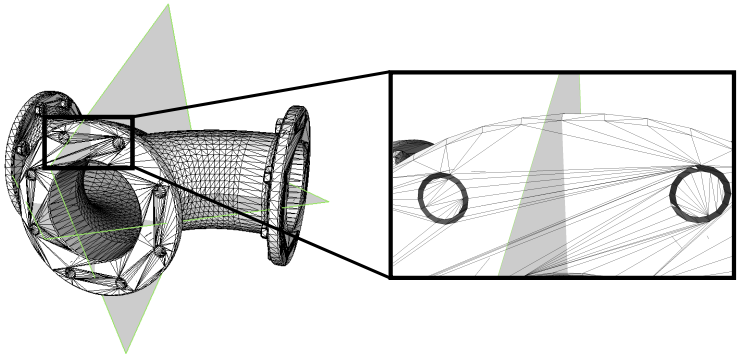
L'information de symétrie est une information importante pour de nombreuses taches géométriques telles que la compression, la mise en correspondance de maillage, ... Le travail développé dans cette thèse a pour but de calculer le groupe de symétrie complet d'un maillage 3D *i.e.* symétries discrètes telle que les symétries planaires et/ou symétries continues telles que les symétries de révolution.

En particulier, le travail développé se concentre sur les symétries globales de la forme 3D (par opposition aux symétries locales) et est indépendante de la tesselation de l'objet original. Ces deux aspects sont illustrés sur la Figure 7.

Trouver toutes les symétries d'une forme est bien plus difficile que de simplement tester si une transformation est une symétrie. En particulier, l'approche naïve consistant à tester autant de candidats que possible pour trouver des symétries est trop coûteuse et imprécise. Une méthode efficace de recherche de symétries est donc indispensable et est divisée en deux



(a)



(b)

Figure 7: Deux caractéristiques importantes de notre recherche de symétries. *a*) Nous détectons les symétries globales *i.e.* qui s'applique au centre de gravité de la forme, par opposition aux symétries locales. *b*) La recherche de symétries doit être indépendante du maillage de la géométrie.

parties dans cette thèse :

Symétries des formes simples Dans une première phase, nous introduisons des fonctions intermédiaires, dont l'ensemble des symétries est un sur-ensemble de l'ensemble des symétries de la forme elle-même mais dont le calcul des symétries est aisé. En utilisant les bonnes propriétés de ces fonctions, nous en déduisons un algorithme déterministe pour isoler un nombre restreint d'isométries candidates à être symétrie de la forme. Ces candidats sont ensuite filtrés *i.e.* testés sur la forme de base afin de décider si elle constitue une symétrie de cette forme.

Inspiré par les travaux sur l'analyse en composantes principales, nous introduisons pour cela, les fonctions moments généralisées d'ordre p d'une forme dont l'expression est donnée par :

$$\mathcal{M}^p(\boldsymbol{\omega}) = \int_{\boldsymbol{s}\in\mathcal{S}} \|\boldsymbol{s}\times\boldsymbol{\omega}\|^p \,\mathrm{d}\boldsymbol{s} \tag{1}$$

Le chapitre 4 est consacré à l'étude de ces moments et à leur importance dans la recherche des symétries d'une forme 3D.

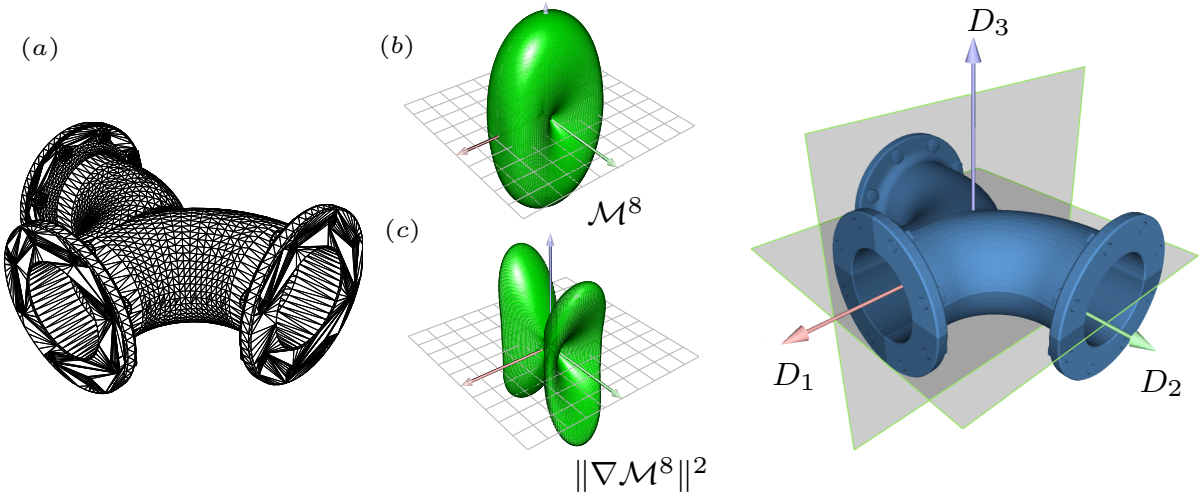


Figure 8: Extraction des symétries pour une forme simple. Partant de la forme originale (a), ces moments généralisés (b) ainsi que les gradients de ces moments (c) sont calculés. L'ensemble de leurs directions d'extrema contient les axes de symétries de la forme, représenté à droite. Ici, deux symétries planaires ont été trouvées ainsi qu'une symétrie de rotation de π. Il est important de noter que la forme n'est ni convexe ni étoilée et que le maillage ne respecte pas les symétries de la forme.

Nous montrons que ces fonctions possèdent de bonnes propriétés dans la recherche des symétries d'une forme. En particulier, ces fonctions ont les mêmes symétries que les formes, plus un petit nombre d'autres candidats qui nécessite d'être filtré. De plus, nous présentons un ensemble de traitement élégant basé sur les harmoniques sphériques pour calculer de manière propre et efficace les symétries d'une forme 3D quelconque. La méthode développée est robuste et permet, par exemple, de calculer les symétries des modèles scannées ou les symétries sont généralement approximatives. Un exemple complet de calcul de symétrie d'une forme 3D est présenté sur la figure 8.

Symétries des formes composées La seconde contribution de cette thèse dans la recherche des symétries est d'étendre l'algorithme précédent en un algorithme constructif, qui calcule de manière indépendante les symétries de chacune des composantes d'un objet - en utilisant l'algorithme présenté précédemment - et combine ensuite cette information pour calculer les symétries de l'objet complet. Lorsque la décomposition d'un objet en accord avec ces symétries est possible, cet algorithme constructif se révèle être plus précis et efficace que la stratégie direct *i.e.* calcul des symétries de l'objet complet (voir Figure 9).

Ces travaux sont présentés en détails à partir de la page 67 de ce manuscrit. L'ensemble des travaux présenté dans cette thèse pour le calcul des symétries d'un objet 3D ont fait l'objet d'une publication dans le journal ACM Transactions on graphics [Martinet et al., 2006].

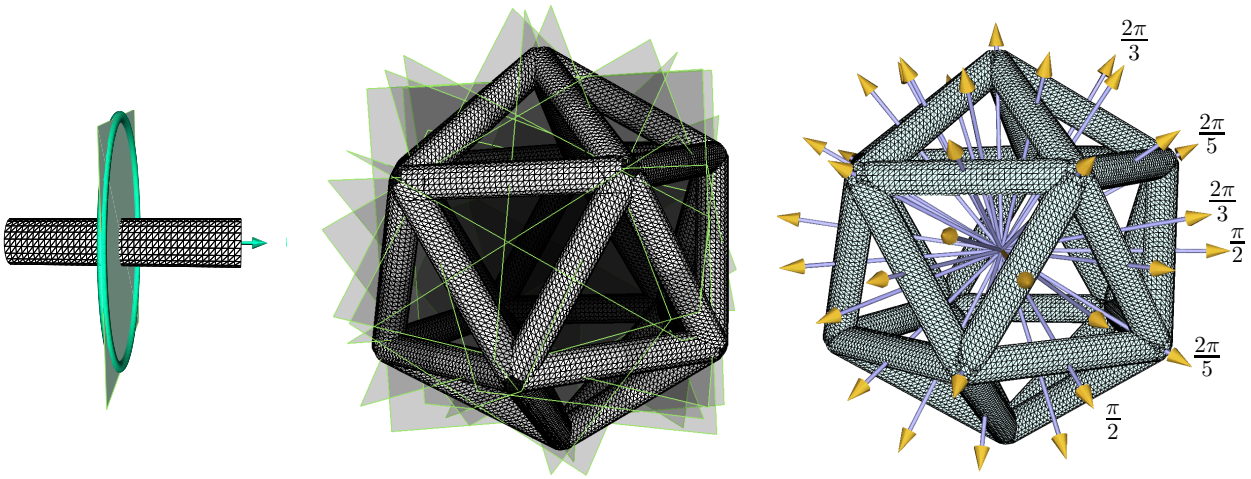


Figure 9: Un modèle complexe possédant le même groupe de symétries qu'un icosaèdre. L'algorithme constructif calcule les 15 plans de symétries (*centre*) et les 31 axes distincts de symétries de rotation (*droite*) en utilisant les symétries de chacune des tuiles de ce modèle (*gauche*).

Instantiation Hiérarchique de Géométrie

La principale motivation de la seconde partie de ce travail est d'ajouter de l'information de structure dans les mondes virtuels faits d'un grand nombre de primitives 3D. Dans cette thèse, nous concentrons notre travail sur la notion d'instantiation à plusieurs niveaux d'échelle aussi appelée *Instantiation Hiérarchique*. La connaissance de l'information d'instantiation dans une géométrie 3D revient à savoir identifier les objets se répétant dans cette géométrie, potentiellement à plusieurs niveaux de détails.

La connaissance d'une telle information peut être utilisée dans de nombreux domaines de l'informatique graphique telles que le lancer de rayon, la radiosité ou l'édition de maillage. En utilisant l'information de symétries calculée par les techniques présentées précédemment, nous présentons une méthode divisée en deux parties afin de représenter une géométrie sous la forme d'une hiérarchie d'instances (voir Figure 10).

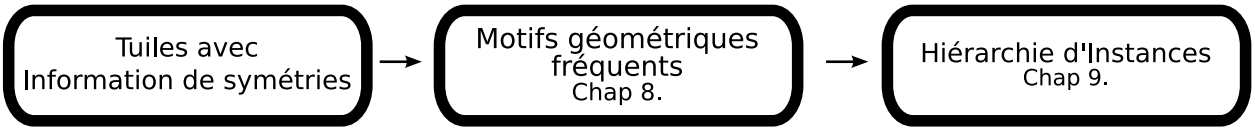


Figure 10: Vue d'ensemble. Partant d'une structure au niveau objet de la géométrie, *i.e.* la décomposition de la géométrie en tuiles, associée à leur information de symétries, nous utilisons une approche en deux étapes pour représenter la géométrie sous la forme d'une hiérarchie d'instances.

Génération des motifs géométriques fréquents La première étape a pour but la recherche des motifs géométriques fréquents dans la géométrie; nos motifs géométriques étant définis

comme des ensembles de tuiles.

Nous prouvons qu'une méthode simple basée sur l'agglomération des tuiles est non utilisable due à sa complexité exponentielle et présentons une méthode originale basée sur la recherche des symétries locales dans la scène. Tous ces éléments sont développés en détails dans le chapitre 8 de cette thèse.

Création d'une hiérarchie d'instances Dans la deuxième étape, nous cherchons à organiser les motifs fréquents de la géométrie sous la forme d'une hiérarchie d'instances. Nous avons identifié plusieurs critères que doit respecter une hiérarchie d'instances afin d'être utilisable (par exemple pour l'édition de maillages et/ou la compression) et montrons que générer une hiérarchie qui respecte ces critères est un problème qui se révèle de complexité exponentielle. Afin de pouvoir appliquer l'algorithme d'instantiation sur des scènes complexes des algorithmes d'approximation doivent être utilisés. Nous présentons dans cette thèse, un algorithme d'approximation permettant la représentation optimale d'une scène pour le lancer de rayon (voir Figure 11).

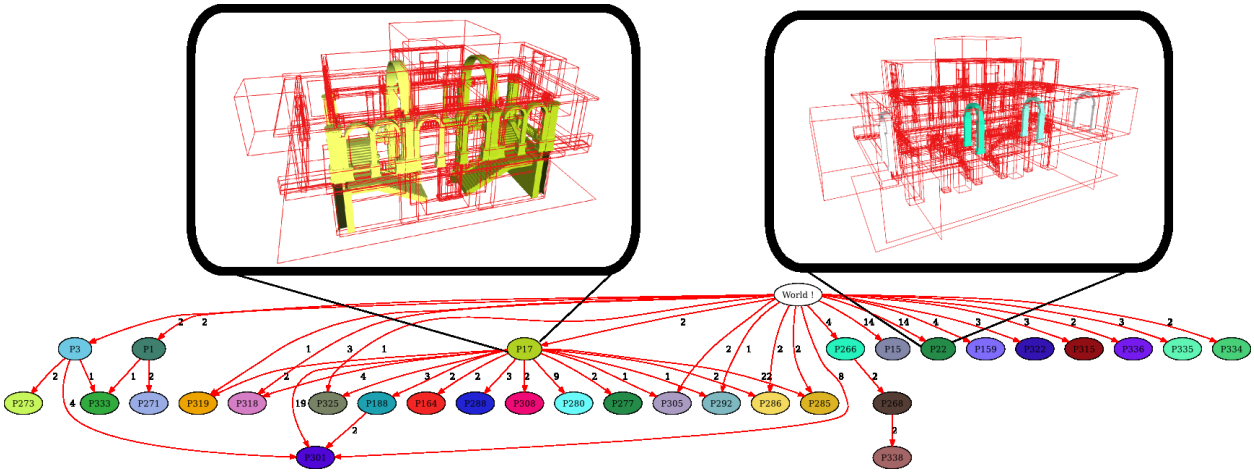


Figure 11: Hiérarchie d'instances optimisée pour le lancer de rayons. Pour cette géométrie, le taux de compression obtenu est d'environ 1:8.

Tous ces éléments sont développés en détails dans le chapitre 9 de cette thèse.

Part I

Introduction and Preliminaries

1 Introduction

The central aspect of this thesis is *structural information* in computer graphics, that is the information concerning the structure of objects or scenes. More precisely, we focus on the question of its accessibility, motivated by the fact that, most of the time, structural information is not present mainly for two reasons: it is either lost, for example during the translation from one format standard to another, or it is not present by nature due to the way the data have been modeled or acquired.

In the first part of this chapter, we focus on the structural information from a general point of view and present some examples of its utilization in the field of computer graphics.

In the second part, we motivate and justify the main assumption of this thesis, that is the lack of structural information in 3D virtual worlds. We first review the problem of exchange of 3D information which may be grouped under the term of *asset exchange*. We then focus on the relation between some particular ways of modeling or acquiring data such as procedural modeling or 3D scanners and the set of structural information that may be accessible in the data they produced.

Finally, in a third part, we present the main contributions of this thesis.

1.1 Structural Information

The growing complexity of the 3D data generated by various modeling techniques requires efficient algorithms to process and visualize these data. The efficiency of such algorithms is related to the underlying structure of the data: without any structuring, an algorithm has no hope to achieve sub-linear complexity which is a huge problem when processing large amount of 3D data with millions of primitives.

In this thesis, we consider structural information to be a tool that might be used by users or designers of the model and more abstractly by any processing and rendering technique to answer the question: “How are things constructed ?”. A large amount of information may be qualified *structural*, ranging from simple geometric features to semantic information such as:

- the symmetry group of an object,
- the $L-$system that generates a natural model such as a plant,
- the parameters of a swept surface,
- the parametrization of a NURBS surface,
- a scene-graph,
- a complete semantic representation of a scene, that captures the functions, the characteristics and relationships between each object in the scene,
- ...

The efficiency of a large number of algorithms is based on such information: the symmetry group of an object may be used for mesh compression, shape matching or mesh editing; The parametrization of a surface has applications in geometric modeling and computer graphics; the scene graph may improve rendering speed through view-frustum culling or by using instancing information it may contain. Finally, associating some semantics with 3D contents may be a major issue for reusing such contents or pieces of content after having extracted them from existing 3D scenes.

The notion of structure is often associated to the notion of scale. This is for example the case of natural models which are modeled using some procedural algorithms such as fractals, which contains repetition of forms over some ranges of scale.

The few examples given above inspired us for a definition of structural information as a two-scale notion, namely the *object* and the *scene* levels: structural information at the object level captures the way the object has been modeled *i.e.* this object is a revolution surface, a Nurbs surface, ... Conversely, structural information at the scene level contains information about how objects are organized in the scene *i.e.* the hierarchy of objects in the scene or their semantic representation.

1.2 Accessibility of Structural Information

The structural information introduced above is of central importance and is broadly used to improve rendering or any other processing of the geometry of the scene. This raises the problem of its accessibility which is at the core of this thesis. More precisely, the basic assumption that motivates our work is:

"Most of the time, structural information is not accessible"

We identify two reasons that justify this assumption:

1. The structural information was present but has been *lost during asset exchange*,
2. The structural information is *not present by nature*.

In the following sections, we review each of these two points.

1.2.1 Asset Exchange

Asset exchange groups every action related to the transmission of 3D information. Exchanging assets has been identified as a major problem as it runs into the difficulty of using multiple platforms, software and files formats.

From a practical point of view, the geometric data modeled by interactive tools[1] are not produced in a sequential way *i.e.* primitive per primitive. The data are usually generated using some semi-automatic tools such as instancing, symmetry or parametric surfaces which *form the set of structural information associated to the geometry*.

In theory, these information must be accessible as part of the modeling process. Experimentally, it is however very frequent that such information is lost during translation from one file format to another. Most of the time, this translation part cannot be avoided, for example if the output of the modeling tools such as Alias Maya, Discreet 3D-Studio is not readable by the user of the model. These problems are very frequent in more complex processes as the industry of Digital Content Creation (DCC) is a highly collaborative process.

In most creation pipelines, the file formats used for asset exchange are defined as *Interchange formats*. An interchange format is used to exchange data from one DCC tool to another and the goal for such a format is to be able to transfer files with the features that the DCC tools provide. A simple example would be the ability to exchange NURBS primitives from Maya to 3ds max, which implies that the programs need to have export and import support for a specific interchange format. The set of interchange format contain the major set of file formats usually used: ply, obj, vrml, x3d, ...

In order for an interchange format to be efficient, an import/export plug-in must exist for every DCC tool that output every *information* that would be exported with the format of the source software. This raises some major difficulties such as:

- the problem of *proprietary formats*. For example, the FBX file format is broadly used in DCC tools but is not an open format. It means that it has not been designed by a working group, but by a single company.
- the need for constant updates of importer/exporter, based on the new features of a modeling tool.

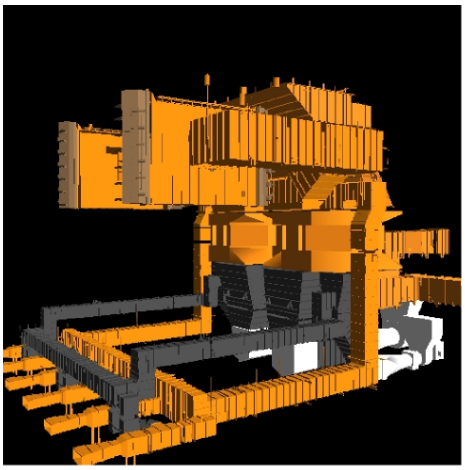
It may also happen that some scenes, modeled by any DCC tool, are distributed under the form of a file format not intended to be a general scene description (see Figure 1.1 on the following page). In this case, structural information that may have been used to originally model the scene is lost as *by nature* these file formats do not support structural information such as object instantiation, modeling hierarchies or objects sub-parts.

Finally, an extreme case that might be used to avoid potential problems with any file format is to transmit 3D data in a incoherent way sometimes called *polygon soup*. This way of encoding the data can be seen as the lowest common denominator of file formats.

Starting from these statements, the two potential working avenues that can be adopted are:

- A way of recovering the lost information or even computing usable information if not initially part of the data, which is the approach retained in this thesis,

[1] The set of interactive software are often refer to as Digital Content Creation tools (DCC tools)



Part of the PowerPlant model

```
ply
.....
.....
0 0 0                { start of vertex list }
0 0 1
0 1 1
1 1 1
1 1 0
......
......
4 0 1 2 3            { start of face list }
4 7 6 5 4
4 0 4 5 1
4 1 5 6 2
4 2 6 7 3
4 3 7 4 0
........
<eof>
```

3D description of the model

Figure 1.1: The powerplant model is a complex model made of 13 millions polygons. A part of it is shown on the left. This model is distributed under the form of PLY file format which is not intended to be a general scene description language. This means for example that it includes no instancing properties which however seems to occur a lot in this kind of scenes.

- a way of keeping the whole set of information accessible during the whole pipeline, *i.e.* an exchange open format made for collaborative work.

The work developed in this thesis focuses on the first point *i.e.* recovering structural information. The second point is about to be solved with the introduction of a tool dedicated to collaborative work named *Collada*(**COLLA**borative **D**esign **A**ctivity):

Collada: The expectations of the Collada file format are very important mainly due to the participants involved in its creation. First introduced by Sony, Collada is now directly supported by major DCC vendors. This way, it will support all the features needed by the major modeling software, and thus facilitate exchange between them.

Right now, Collada is primarily used to extract the content from a DCC tool, or to interchange data between DCC tools. At the final production stage, developers generally transform the Collada content to their own format *i.e.* PlayStation, X3D, Xbox 360, etc. . . , and use their own "conditioning pipeline" or "asset pipeline" to optimize the data, test the data for problems, merge data with other content, and "compile" the data in platform specific format for use with a specific game engine / real-time application. We refer to [Arnaud and Barnes, 2003] for more details about Collada.

1.2.2 Non-interactive modeling

Among the set of modeling techniques, the main advantages of the non-interactive techniques such as procedural modeling, scanning or image-based modeling is their automatic nature, that allows the generation of extremely detailed models that would be very difficult, or even

impossible, to obtain through interactive modeling techniques. This feature is essential as it allows to reach the natural degree of complexity that arise in natural scene.

However, this way of generating the data is also the main drawback of those methods: most of the time, data are produced in a sequential way, *without any structuring* and hence do not contain, in general, any information about the way they have been generated. It may however happen that such an information is present but does not match the wanted notion of structure. For example, when procedurally modeling cities, the modeler will group together all objects that share the common materials and hence loose information of which objects are repeated and where. Another example concerns the use of 3D scanners to acquire large environments. In this case, the output of such modeling method is an unstructured set of primitives. We rapidly review the main non-interactive techniques used for modeling:

Scanning tools

Scanning tools are used to acquire geometry of objects through active sensors.

CT or MRI scanner

Computed Tomography (CT) and MRI (Magnetic Resonance Imaging) are two ways of acquiring 2D data sets. Usually those data are acquired under the form of a regular pattern (*i.e.* one slice every millimeter) in order to form a complete 3D data set. This final data set is usually represented as a voxel grid (see Figure 1.2). The major application area of such techniques is medical imaging.

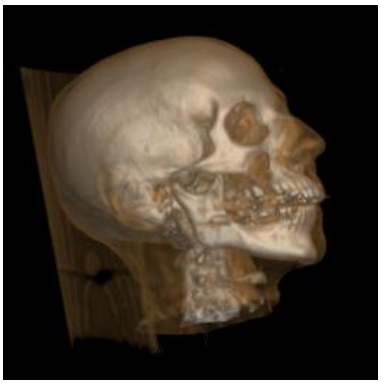


Figure 1.2: Volume rendering of 3D acquired from Magnetic Resonance Imaging.

3D scanners

The purpose of 3D scanners (see Figure 1.3 on the following page) is to create a point cloud of geometric samples on the surface of the object. Starting from these points, a reconstruction process can be used to recover a 3D polygonal model from the scan. If color information is collected at each point, then the colors on the surface of the object can also be determined. 3D scanners are very analogous to cameras. While a camera collects color information about surfaces within its field of view, 3D scanners collect distance informations about surfaces

within its field of view. The "picture" produced by a 3D scanner is a *depth image* that describes the distance to a surface at each point in the picture.

For most situations, a single scan will not produce a complete model of the subject. Multiple scans, even hundreds, from many different directions are usually required to obtain information about all sides of the subject. These scans have to be brought in a common reference system, a process called *registration*, before a *reconstruction* process could be used to form a complete 3D polygonal model. 3D scanners are very popular and have been employed to acquire large models such as the well-known digital Michelangelo project [Levoy et al., 2000].

The 3D scanner techniques are however not limited to a single object and methods have been developed to acquire entire environments from multiple views [Wang and Oliveira, 2002].

Figure 1.3: Laser Scanners. Model Shop Color 3D Scanner and high-resolution Desktop 3D Model 15

The geometry acquired by 3D scanner technology is highly unstructured and the output of such modeling techniques is a large point cloud or even large meshes. The processing of such large unstructured 3D data sets is now an important subject of research where more relevant work deal with multi-resolution techniques on arbitrary meshes.

Image-based Modeling

Although laser scanning are able to produce high quality 3D reconstructions, they still lack flexibility with respect to material and lighting conditions, are relatively expensive, and are often applicable only to restricted types of objects. This has motivated techniques to reconstruct 3D objects from simple photos or video.

In computer vision, the techniques to recover 3D shapes from 2D images are called shape-from-X techniques, where X can be shading, stereo, motion, texture, ...

An important domain of research is the shape-from-stereo techniques and more specifically the *multi-view stereo*, who reconstructs a complete 3D model from a collection of images taken from known camera viewpoints (see Figure 1.4 on the next page).

[Wilczkowiak et al., 2003; Debevec, 1996] proposed an interesting approach of semi-automatic modeling based on geometric constraints. The main objective is to reconstruct 3D

Figure 1.4: Reconstruction *(right)* with the method of [Goesele et al., 2006] of a temple and an input image *(left)* for comparison.

models from one or few images by introducing some geometric constraints of incidence, parallelism or symmetries. We refer the interested readers to [Seitz et al., 2006; Wilczkowiak, 2004].

Procedural Modeling

The term "Procedural Modeling" refers to the automated (or partially automated) generation of 3D geometry. It is often used to create complex natural objects such as plants or mountains but can also be applied to man-made geometry as well. More generally, it is a modeling technique well adapted for models resulting from repeating, self-similar or random process (see Figure 1.5).

The two main advantages of this modeling technique are its ability to create complex worlds, tedious to build by hand, and its scalability.

There exist various types of procedural modeling techniques which we detailed below.

swept surfaces

Those surfaces are obtained by sweeping a 2D curve along a 3D trajectory. The simplest swept surfaces are *revolution surface* in which the 3D trajectory is simply a line. By using a more complex profile, complex models can be obtained such as seashells [Fowler et al., 1992] as presented in Figure 1.6.

fractals

The precise mathematical definition of fractals defines a fractal object as an object that present self-similarity at infinite resolution. In the computer graphics field, [Ebert et al.,

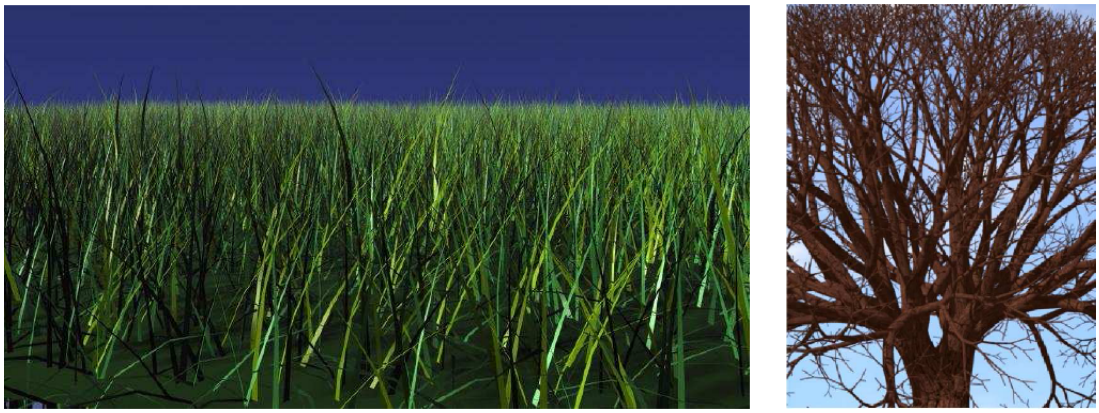


Figure 1.5: Procedural modeling is well adapted for models resulting from repeating, self-similar or random process.

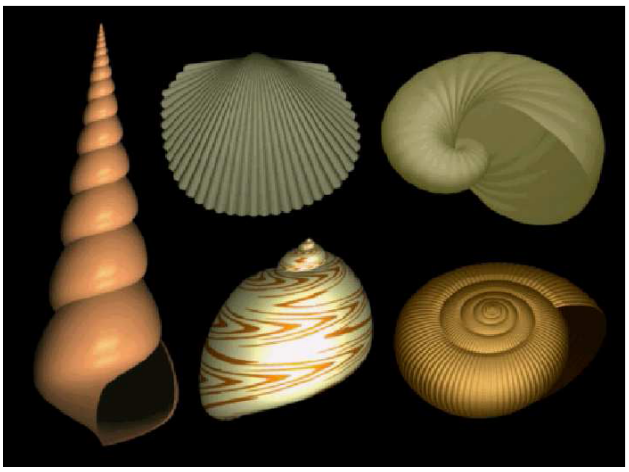


Figure 1.6: An example of seashells obtained by extrusion with the methods of [Fowler et al., 1992]. Various shapes are obtained by varying the parameters of the seashells generation.

2002] defined a fractal as "a geometrically complex objects , the complexity of which arises through the repetition of form over some range of scale". Many natural objects exhibit this characteristic such as terrains, plants, clouds, water, fur, *etc.*...

In general, fractals can be classified as deterministic and non-deterministic (also called random fractal), depending on whether they contain randomness.

Random fractals have been used extensively in computer graphics to model natural objects, most notably terrains. Most fractal terrain generation algorithms work through recursive subdivision and pseudo-random perturbation. An original surface is defined and divided equally into subparts. New vertices are added and pseudo-randomly displaced from the original surface, with a displacement magnitude that decreases at each iteration. Therefore, the first iteration gives the large peaks on the surface, and later subdivisions add smaller-scale detail. The major advantage of such methods is that only the parameters for controlling the random number generator, the level of subdivision, and the "roughness" of the surface need

to be tuned to define an extremely complex terrain.

Grammar-based Models

Grammar-based models, primarily $L-$systems, allow natural complexity to be specified with a few parameters. Grammar-based models have been broadly used in computer graphics [Fowler et al., 1992; Deussen et al., 1998] to produce remarkably realistic models and images of trees, plants and seashells. These models use formal languages, parallel graph grammars called $L-$systems, to describe natural structures algorithmically and are closely related to deterministic fractals in their self-similarity.

An $L-$system is a formal language where all the rules are applied in parallel to provide a final "sentence" describing the object. In the $L-$system, each terminal symbol represents a part of the object or a directional command to be interpreted by a three-dimensional drawing mechanism. A "sentence" for a tree would contain words describing each branch, its length, size and branching angle, when its develops, and its connection in the tree. A simple $L-$system, as well as one complete grammar-based model are presented in Figure 1.7

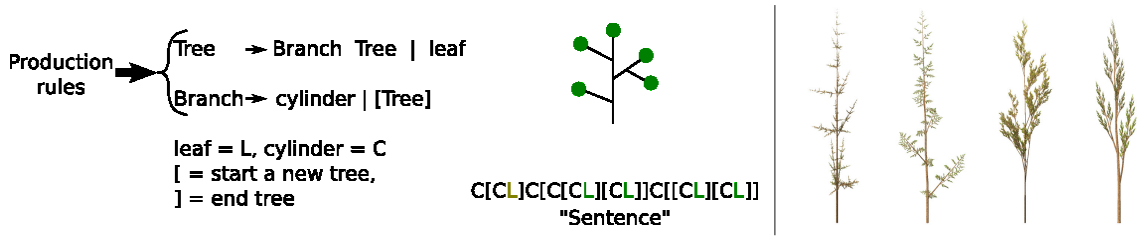


Figure 1.7: Left) A simple $L-$system formed by two production rules and the resulting sentence and trees. The only terminal symbol of this grammar is the "leaf". Right) Several examples of Grammar-based models of trees.

Combined methods

The "Combined methods" do not represent new procedural ways of modeling data but rather a way of combining existing methods (such as $L-$systems) with domain specific knowledges. Such methods have been mainly developed to improve the simulation of plant growing processes [Deussen et al., 1998; Prusinkiewicz et al., 2001].

Such methods are capable of simulating the growth interaction with the features of the field, such as ecological characteristics of plant species and the initial distribution of plants. At the very end, a procedural model is used to generate each individual plant.

Once again, the geometry modeled by procedural methods is most of the time unstructured. For example, as pointed out by [Hart, 1992], the geometry generated by $L-$system models is not organized efficiently for rendering. Due to their automatic nature, these modeling techniques have most of the time *no knowledge about the global structure of the geometry* they output, such as terrains modeled by fractals. Yet generated by introducing some randomness in the generating process, some global structure may be present, such as "packs" of geometry repeated at different scales.

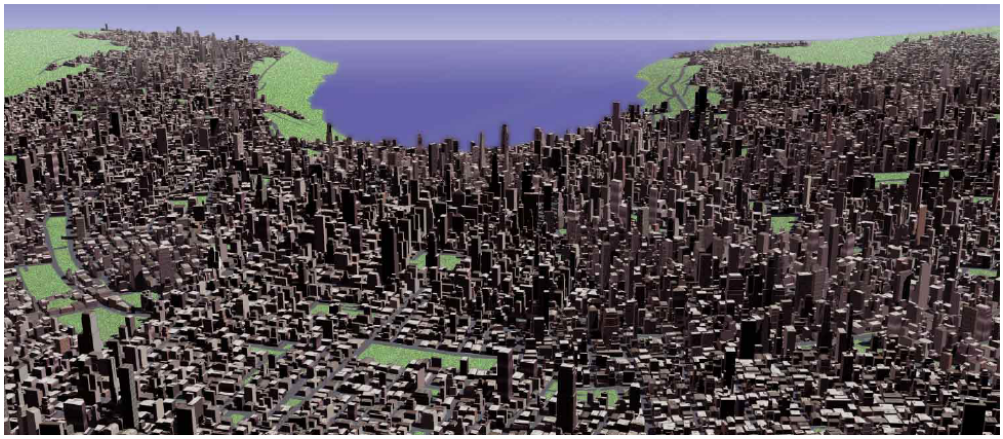


Figure 1.8: A city procedurally created by the method of [Parish and Muller, 2001]. Approximately $26,000$ buildings were created.

1.3 Contributions

In this thesis, we focus on two specific types of structural information and propose a way of structuring the scene at both the object and scene levels. More precisely, we structure the objects and scenes using:

- Symmetry of objects *i.e.* structuring at the object level.
- Hierarchical representation of instances in the scene *i.e.* structuring at the scene level.

We qualify the input scenes that we want to process as *unstructured* and the work developed in this thesis aimed at structuring these scenes as represented in Figure 1.9.

The unstructured aspect of the scenes raises the problem of *defining what an object is*. We propose, in Chapter 2, a way of pre-processing this unstructured geometry and a new way of defining what an object is, based purely on geometric information.

Based on this definition of objects, the two main contributions of this thesis are detailed below (see Figure 1.9):

Structuring at the object level: *The symmetry information* is object-based structural information that is of great importance for many geometric tasks such as compression, mesh editing, shape matching, ... The work developed in this thesis focuses on computing the whole symmetry group of a 3D mesh *i.e.* either discrete (such as planar or rotational symmetries) or continuous (such as cylindrical or spherical symmetries). Finding all symmetries of a shape is much more difficult than simply checking whether a given transform actually is a symmetry. In particular the naive approach that would consist of checking as many potential candidates as possible to find a symmetry is far too costly. We thus need a deterministic method to find good candidates. The work done in this thesis for symmetry computation is divided in two phases:

- In a first phase, we use intermediate functions, whose set of symmetries is a superset of the set of symmetries of the shape itself, but for which computing the symmetries is

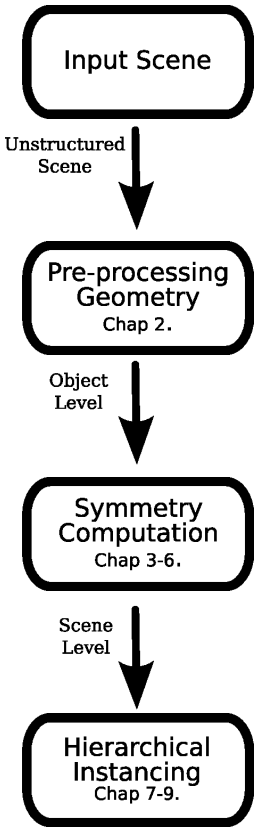


Figure 1.9: *Thesis Outline.* Starting from an unstructured scene, we first form objects in this scene, compute the symmetries of each of these objects and then form a hierarchy of instances of the scene.

much easier. By examining these functions, we will derive a deterministic algorithm which finds a finite number of possible candidates. These candidates are then checked back on the original shape. Choosing a family of functions which fulfills these requirements is relatively easy. More difficult is the task of finding such functions for which computing the symmetries can be done both accurately and efficiently.

Inspired by the work on principal component analysis, we introduce for this purpose the *generalized moment* functions of the shape (see Chapter 4). These functions indeed, have the same symmetries as the shape itself, plus a small number of extra candidates. Furthermore, we propose an elegant framework based on spherical harmonics to accurately and efficiently find symmetries of the generalized moment functions.

- The second contribution, presented in Section 4.5, is to extend the proposed algorithm into a constructive algorithm, which separately computes the symmetries of sub-components of an object – using the first method –, and then associates this information to compute symmetries of the whole composite shape. When it is possible to decompose an object according to its symmetries, this constructive algorithm proves to be more accurate and efficient than directly computing the symmetries of the whole object.

The work developed in this part has been published in [Martinet et al., 2006].

Structuring at the scene level: The main motivation of the second part of this work is to add some structural information to virtual worlds made of massive 3D polygonal data, obtained from modeling through interactive or non-interactive tools. We focus on *instancing information i.e.* the structure of what is repeated in the scene at multiple levels of scale. The knowledge of such information can be used in many tasks such as rendering efficiency for ray-tracing or radiosity, or for mesh-editing.

From the symmetry information that we have computed on the objects of the scene, we show that computing such a hierarchy is a non-trivial process and divide it in two steps:

- The first step, presented in Chapter 8, aims at discovering frequent geometric patterns *i.e.* objects or set of objects that occur multiple times in a scene. We translate this problem into finding local symmetries in the scene and propose an efficient scheme to compute such information.
- The second step, presented in Chapter 9, organizes these patterns into a hierarchy. We identify some criteria that must be respected by the hierarchy in order for it to be efficient, and prove that computing a hierarchy that respects this constraint is an exponential problem *i.e.* a problem where no polynomial algorithm exists to solve this problem. We thus propose an approximation algorithm based on the criteria the hierarchy of instances must respect.

2

Preliminaries: Pre-processing geometry

As presented in the Introduction, the scenes we want to process are unstructured. Some non-interactive modeling techniques such as 3D scanners output the geometry of the model as an incoherent list of polygons *i.e.* polygons soup. Conversely, it is very rare that the geometry modeled by interactive software came in a totally unstructured way.[1] Most of the time, the model came as an Indexed Face Set which describe a complete mesh in a more convenient way. However, this way of structuring the data is most of the time not usable as it may for example regroup objects by common material and hence lose the information that an object is replicated elsewhere in the scene with another material.

This motivate us for considering each input scene as an incoherent list of polygons and hence "forget" about any connexity or material information that may be present initially. Starting from this incoherent list of polygons, we have to form objects in the scene, which is the purpose of this chapter.

2.1 Object Creation

Defining "objects" in an incoherent list of polygons must respect two constraints:

1. The way of forming "objects" must be independent of the tessellation of this object.
2. The complexity of the object creation step must be low as we aimed to compute it on large 3D environments with up to 15 millions of polygons.

We propose to define objects as *tiles*:

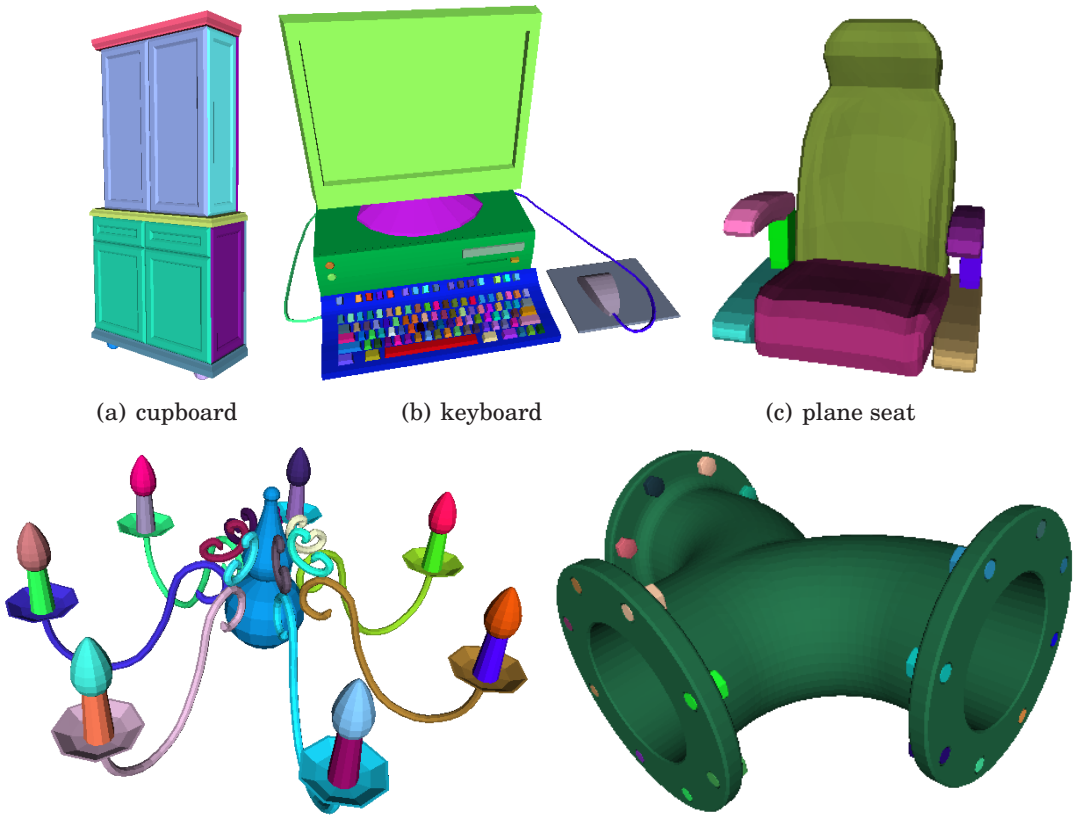
Definition 1 *A tile is a maximal set of edge-connected polygons*

[1] This may however happen as a list of polygons constitutes a practical common denominator to all possible formats.

To obtain them, we insert all vertices of the model into a KDTree and use this KDTree to efficiently recover which polygons share vertices up to a given precision and share an edge. By propagating connectivity information between neighboring polygons we then build classes of edge-connected polygons which we define to be our tiles.

It is quite easy to verify that the two conditions introduced as the beginning of this section are respected. First of all, as the definition of the tiles is based on connectivity between polygons, it is completely independent of the degree of tessellation of an object. Moreover, computing the tiles of a scene is a very fast process as it only requires to build a KDTree an iterate trough every vertices of the scene (see the computation times in Table 2.2).

2.2 Results



(a) cupboard (b) keyboard (c) plane seat

(d) chandelier (e) petrolpipe

Figure 2.1: Examples of tiles decomposition. For every models, each of its tiles is representated by a distinct color. Numerical values associated to these models are presented in Table 2.2 on the facing page.

scene name	# polygons	# tiles	computation time(secs)
cupboard	1,204	13	0.1
computer	2,437	115	0.1
plane seat	3,346	8	0.1
chandelier	9,632	29	0.4
petrolpipe	22,304	25	1.6
LBXStudio	514,890	8,389	12
Powerplant	12,748,510	155,348	2mns

Part II

Detecting Symmetries in 3D Shapes

3
Background and Related work

Symmetry is one of the most important features of objects. Identifying symmetries of an object is an interesting problem as this knowledge can lead to various applications such as compression, reconstruction, classification, analysis, alignement and model matching. Furthemore, symmetry is a fundamental concept which human visual perception utilizes and many objects constructed by humans are symmetrical, because this makes them easier to interpret and manufacture.

In this chapter, we first introduce the fundamental concepts of symmetry from a mathematical point of view and then present the most relevant work done in symmetry detection.

3.1 Background

In the following, $\mathscr{E}$ represents an **affine euclidean space** of dimension two or three with a direct orthonormal frame $(O, \boldsymbol{e}_1, \boldsymbol{e}_2)$ or $(O, \boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3)$ depending on the dimension. The vector space associated to $\mathscr{E}$ is noted E, which is of dimension two or three with direct orthonormal basis $(\boldsymbol{e}_1, \boldsymbol{e}_2)$ or $(\boldsymbol{e}_1, \boldsymbol{e}_2, \boldsymbol{e}_3)$ depending on the dimension. Except when otherwise specified, we note $M^{'}$ the image of a point M by an affine application.

3.1.1 Affine and Vector Isometry

Definition 2 *$f : \mathscr{E} \to \mathscr{E}$ is an affine isometry if:*

$$\forall A, B \in \mathscr{E} \quad d(f(A), f(B)) = d(A, B)$$

By definition, an affine isometry has the property of conserving distance.

An affine isometry may be seen as a particular type of approximate isometry, also refer to as $\varepsilon-$*isometry*:

Definition 3 *$f : X \to X$ is an $\varepsilon-$affine isometry if:*

1. *for every* $(x,x') \in X^2$, *one has* $|d(f(x),f(x')) - d(x,x')| \leq \varepsilon$ *and*
2. *for any point y in X, there exists a point x in X such that* $d(y,f(x)) \leq \varepsilon$.

That is, an $\varepsilon-$isometry preserves distances to within ε and leaves no element of the codomain further than ε away from the image of an element of the domain.

Property 1 *The distance-preserving property of an affine isometry induces angle and area preserving property.*

Proof Let consider three non-aligned points A, B and C of $\mathscr{E}$, for which:

$$BC^2 = BA^2 + AC^2 + 2\overrightarrow{BA}.\overrightarrow{AC}$$

and considering their images by an isometry A', B' and C':

$$B'C'^2 = B'A'^2 + A'C'^2 + 2\overrightarrow{B'A'}.\overrightarrow{A'C'}$$

By applying distance-conservation property, we have $BA = B'A'$, $AC = A'C$ and $BC = B'C'$ which make the two scalar products equal, *i.e.* $\overrightarrow{BA}.\overrightarrow{AC} = \overrightarrow{B'A'}.\overrightarrow{A'C'}$ and hence prove the angle-preserving property.

Theorem 1 *Let* f *be an affine isometry and M,N two points of* $\mathscr{E}$.
$\varphi : E \to E$, *defined by* $\varphi(\overrightarrow{MN}) = \overrightarrow{f(M)f(N)} = \overrightarrow{M'N'}$ *is the vector isometry associated to* f.

Theorem 2 *If* f *is an affine isometry, its associated vector isometry* φ *is an orthogonal endomorphism.*

Proof By simply using the distance-preserving property of an affine isometry, we have

$$\forall A,B \in \mathscr{E} \quad d(f(A),f(B)) = d(A',B') = d(A,B) \Leftrightarrow \forall \overrightarrow{u} \in E \quad ||\varphi(\overrightarrow{u})|| = ||\overrightarrow{u}||$$

and equivalence can be established with $\overrightarrow{u} = \overrightarrow{AB}$.

The vector isometry associated to f only indicates a change of direction induced by the isometry on a vector and not a change of origin of this vector.

Example 1 *The vector isometry associated to a translation on* $\mathscr{E}$ *is the identity.*

3.1.2 Orientation-preserving ability of isometry

The set of vector and affine isometries can be partitioned based on their ability to preserve orientation.

Definition 4 *An affine isometry* f *is orientation preserving if, for all non collinear points A, B, C, the proper angle measures of the angles ABC and A'B'C' have the same sign. It is orientation reversing if the measured angles have opposite signs.*

A vector isometry associated to an orientation preserving affine isometry is called a *direct isometry*. Convertly, a vector isometry associated to an orientation reversing affine isometry is called an *indirect isometry*.

Theorem 3 *The Classification Theorem: Every isometry is one of the following: the identity, a translation, a rotation, a reflection, or a glide reflection.*

The proof of this theorem is presented in Appendix B.

3.1.3 Vector Isometry in $\mathbb{R}^3$

If (e_1, e_2, e_3) represents a basis of $\mathbb{R}^3$, a vector isometry φ is completely defined by a system of three vectors:

$$a_1 = \varphi(e_1) \quad a_2 = \varphi(e_2) \quad a_3 = \varphi(e_3)$$

which correspond to the images of the vectors of the basis through the isometry φ.

In this basis, this isometry is represented by a matrix formed by filling its rows with the vectors a_i:

$$\begin{pmatrix} a_1^1 & a_1^2 & a_1^3 \\ b_1^1 & b_1^2 & b_1^3 \\ c_1^1 & c_1^2 & c_1^3 \end{pmatrix}$$

Since a vector isometry is a norm-preserving application, we have the following property:

Property 2 *In an orthonormal basis, the matrix of an isometry is orthogonal, which implies that its determinant is* ± 1.

This property naturally induces the following corollary:

Corollary 4 *Any orthogonal matrix is the matrix of an isometry in an orthonormal basis.*

A useful task that will be used in the following is to compute the parameters of an isometry, that is its nature (direct or indirect), its axis and angle. We explain below how to compute such parameters from the matrix representation of the vector isometry.

Direct Isometry. Let φ be a direct vector isometry, given by its matrix M_φ in any basis. As direct, the determinant of its matrix representation is equal to 1.

There exists a basis (e_1, e_2, e_3) of $\mathbb{R}^3$ in which the matrix of φ is:

$$M_\varphi = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix}$$

The vector isometry φ is either the identity or the rotation of axis e_1 and angle θ. The axis e_1 is the only non-null vector invariant with respect to φ.

This vector can be easily obtained by solving:

$$\varphi(e_1) = e_1$$

To compute the angle of rotation θ, we:

- compute $\cos(\theta)$ with the trace of the matrix M_φ, which is equal to $1 + 2\cos(\theta)$, independently of the coordinate system,
- search the sign of $\sin(\theta)$ which is the same as $\det(e_1, u, \varphi(u))$, where u is any vector non-colinear to e_1.

If $\theta = 0$, φ is the identity vector isometry, otherwise φ represent the rotation of axis e_1 and angle θ.

Notation: When the angle of rotation of a vector isometry φ can be expressed as a fraction of 2π, *i.e.* $\theta = 2\pi/k$, φ is called a $k-$fold rotational isometry.

Indirect Isometry. Let φ be an indirect isometry, given by its matrix M_φ. There exists a basis (e_1, e_2, e_3) of $\mathbb{R}^3$ in which the matrix of φ is:

$$M_\varphi = \begin{pmatrix} -1 & 0 & 0 \\ 0 & \cos(\theta) & -\sin(\theta) \\ 0 & \sin(\theta) & \cos(\theta) \end{pmatrix}$$

The indirect vector isometry φ is the commutative composition of a vector rotation and a reflexion with the respect to the plane orthogonal to the rotation axis.

The axis of rotation e_1 is changed to its opposite by φ. It can be obtained by solving:

$$\varphi(e_1) = -e_1$$

To compute the angle of rotation θ, we:

- compute $\cos(\theta)$ with the trace of the matrix M_φ, which is equal to $-1+2\cos(\theta)$,
- search the sign of $\sin(\theta)$ which is the same as $\det(e_1, u, \varphi(u))$, where u is any vector non-colinear to e_1.

3.1.4 Symmetry of 3D Shapes

As presented above, for a given isometry I, there always exists an orthonormal basis $(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{Z})$ into which the matrix of I takes the following form:

$$I(\lambda, \alpha) = \begin{pmatrix} \lambda & 0 & 0 \\ 0 & \cos\alpha & -\sin\alpha \\ 0 & \sin\alpha & \cos\alpha \end{pmatrix} \text{ with } \begin{cases} \alpha \in [0, 2\pi[\\ \lambda = \pm 1 \end{cases} \tag{3.1}$$

As illustrated by the example of the cube (see Figure 3.1 on the next page), this corresponds to three different classes of isometries: rotations, mirror-symmetries and their composition, depending whether λ is positive and/or $\alpha = 0 (\mathrm{mod}\ \pi)$.

Finding a symmetry of a shape thus resolves into finding the parameters of the isometries, that is a vector $\boldsymbol{X}$ — which we call the *axis* of the isometry — and an angle α — which we call the *angle* of the isometry — such that $I(\lambda, \alpha)$ maps this shape onto itself.

3.1.5 Symmetry Group

The vector isometries presented above form a group structure refered to as a symmetry group. More precisely, the Euclidean group $E(n)$ is the symmetry group of the euclidean space $\mathbb{R}^n$, *i.e.*, the elements of this group are the vector isometries of this space. The group of direct isometries $E^+(n)$ is a subgroup of the euclidean group $E(n)$. The group of indirect isometries is called $E^-(n)$.

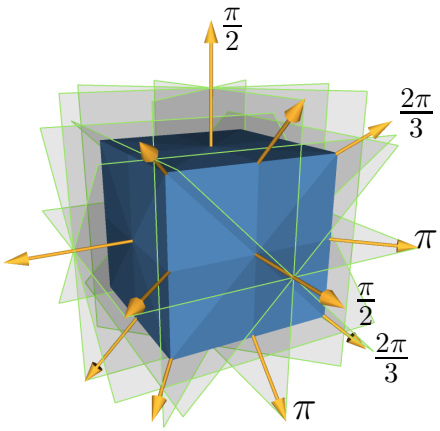


Figure 3.1: Mirror-symmetries and rotational-symmetries found by our algorithm for a cube (for clarity, not all elements are represented).

Definition 5 *The symmetry group of an object is the group of all isometries under which it is invariant with composition as the operation. It is a subgroup of the isometry group of the space concerned, i.e.* $E(n)$.

Example 2 $C_4 = \{I, R, R^2, R^3\}$ *is a symmetry group, where* R *is a rotation of angle* $\frac{\pi}{2}$ *in* $\mathbb{R}^3$. R^2 *is the half-turn around the rotation axis, which is the composition of two rotations* R. *It is the only element, distinct from neutral element* I, *which is its own inverse. The group* C_4 *is often referred to as the cyclic group of order 4*[1].

The symmetry groups are often divided in two groups: the *discrete* and the *continuous* symmetry groups.

Discrete symmetry groups. These symmetry groups come in three types:

- *Finite point groups*, *i.e.* group of symmetries leaving a point fixed. These groups included rotations, reflections and central symmetry.
- *Infinite lattice groups*, which include only translations.
- *Infinite space groups* which combines elements of both previous types

Continuous symmetry groups. Groups with an infinite number of elements that contain rotations of arbitrary small angles and/or translations of arbitrary small distances. Once again, these symmetry groups come in two types:

- Continuous symmetry groups with a fixed point, including *cylindrical* and *spherical* symmetry.

[1] A cyclic group is a group that can be generated by a single element (in this case the rotation R)

- Continuous symmetry groups without a fixed point, including those with a screw axis, such as an infinite helix.

If the symmetry group of an object O is only formed of the identity isometry, then O will be called *asymmetric*, otherwise *symmetric*.

3.1.6 Geometric Congruence

Definition 6 *Two geometric figures that can be related by an isometry are called congruent.*

A necessary, but not sufficient condition, for two figures to be congruent is that these two figures must be of the same symmetry type:

Definition 7 *Two geometric figures are considered to be of the same symmetry type if their symmetry groups are conjugate subgroups of the Euclidean Group $E(n)$. Two subgroups H_1 and H_2 of a group G are said to be conjugate, if there exists $g \in G$, such that $H_1 = g^{-1}H_2g$.*

As an example, two geometric figures both with mirror symmetry, but with respect to a different mirror plane are considered to be of the same symmetry type (see Figure 3.2.)

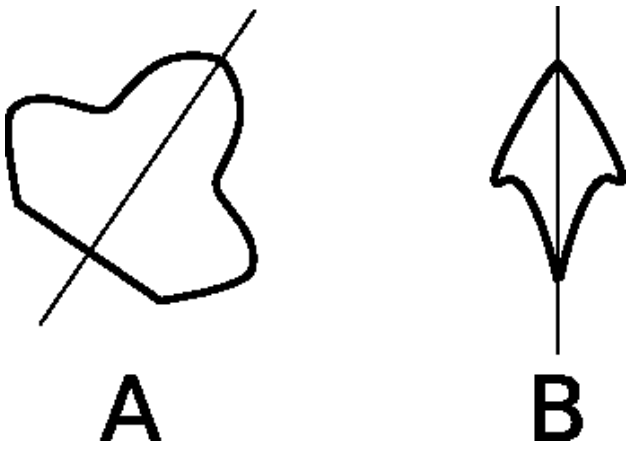


Figure 3.2: A necessary, but not sufficient condition, for two figures to be congruent is that those two figures must be of the same symmetry type. The 2D geometric figures A and B are not congruent but have the same symmetry type, since they both have a mirror symmetry.

3.2 Related Work in Symmetry Detection

In this section, we present related work in Symmetry Detection. These papers can be broadly classified in two categories based on the symmetry information that they output:

- *Symmetry as a binary feature:* "real" symmetries of the model. Searching for those kind of symmetries is essentially a binary question since an object is either symmetric or non-symmetric with respect to a target isometry. These algorithms may be categorized into detection of either global or local symmetries and as either involving exact or approximate symmetries.

- *Symmetry as a continuous feature:* Deriving a measure of symmetry for an object. In this case, the problem is not binary but continuous since these methods try to answer the question: "Which amount of the model is symmetric with respect to this isometry ?".

3.2.1 Symmetry as a binary feature

We present in this section, the most relevant methods that have been released concerning the detection of symmetry of a geometric entity in 2D and 3D. As stated before, these methods may be categorized into detection of either global or local symmetries and as either involving exact or approximate symmetries.

Symmetry as a pattern-matching problem:

Many methods [Attalah, 1985; Wolter et al., 1985; Highnam, 1985] propose to turn the symmetry detection problem into a substring matching algorithm.

The substring matching problem can be formulated as follows:

"Given strings S and T, find all occurrences of T as a substring of S"

To solve this problem, [Knuth et al., 1977] proposed a method that solves the substring matching problem with complexity proportional to the sum of the length of the strings. One interesting application of this technique is symmetry detection which has been initially introduced by [Attalah, 1985]. The starting point of these methods consists in considering a string as a discrete signal on a circle as depicted on Figure 3.3 on the next page.

From this, symmetry detection can be done on the signal represented by S. More precisely, we can test if S has rotational symmetry by testing if S is a non-trivial substring of SS, that is the concatenation of S with itself (see Figure 3.4 on page 47).

In the same way, we can test if the signal S has a reflective symmetry by testing if S is a substring of $(SS)^t$ (see Figure 3.5 on page 47).

The idea of considering the string as a discrete signal on a circle leads to efficient algorithms for computing discrete symmetries of set of points in 2D and 3D:

Testing symmetries of 2D point set. Computing symmetries of a 2D point set is a three steps process:

1. Replace each point of the point set by a point on the unit circle by encoding its distance by a symbol from an alphabet,

2. encode, for each point, its angular distance to its neighbor around the circle (also using some symbol in an alphabet).

3. The signal on the circle is transformed to a string whose main property is that symmetries of the initial point set are also symmetries of this string.

The process described above can so be used to detect reflective and rotational symmetries of the set of points (see Figure 3.6 on page 48).

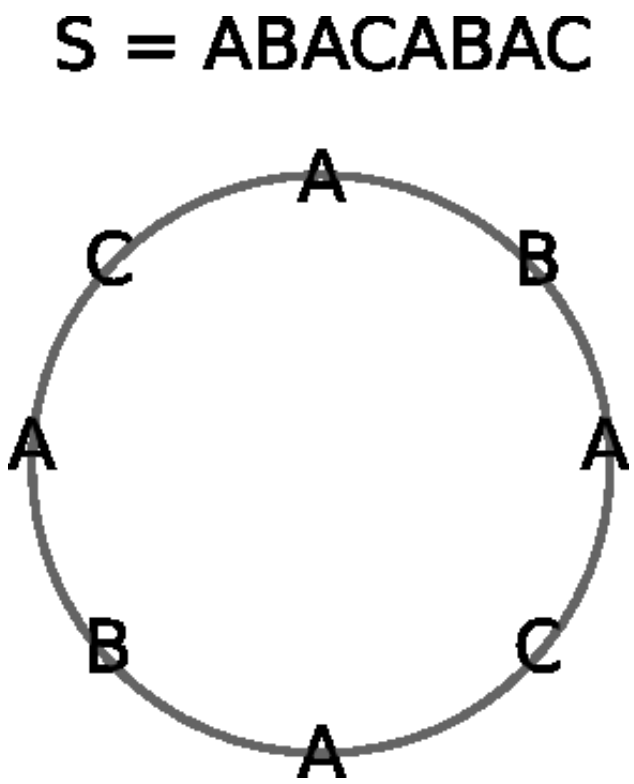


Figure 3.3: The starting point when mapping the symmetry detection problem to the substring matching one is to consider the string as a discrete signal on the circle.

Testing symmetries of 3D point set. For the extension to the 3D case, [Alt et al., 1988] proceed in a similar way, by first projecting all points on the unit sphere. As the set is now formed of labeled points on the unit sphere, their convex hull is some polyhedron, which may be described using edges graph augmented by edges label carrying the edge length and the angular distance to the next vertex around this edge. The problem is now to detect the symmetries of this polyhedron using one of the techniques described in the next section.

Although these methods are capable of recovering reflective and rotational symmetries of point set, their main limitations are:

- limited to global symmetry,
- hard to extend for approximate symmetries. A first attempt to reach this goal has been introduced by [Alt et al., 1988] but the complexity of its approach makes it impractical for large data sets.

As a natural extension of symmetry of point set, we now focus on methods that aimed at discovering symmetries of polyhedra.

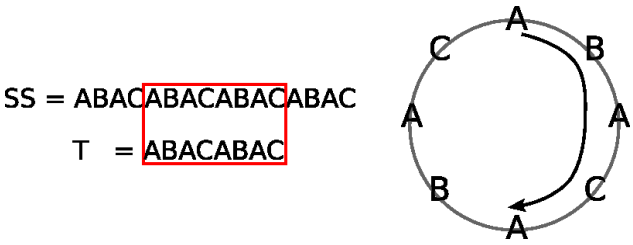


Figure 3.4: To test if the signal S has a rotational symmetry, it suffices to test if S is a non-trivial substring of SS.

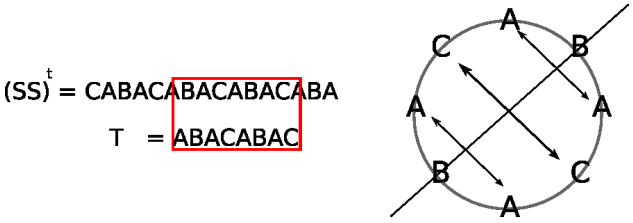


Figure 3.5: To test if the signal S has a reflective symmetry, it suffices to test if S is a substring of $(SS)^t$.

Symmetries of Polyhedra:

A lot of work in computational geometry has dealt with the task of detecting symmetries of polyhedra. In the following, we present some classes of methods developed to detect symmetries of polyhedra based on graph techniques and "generate-and-test" methods. For a complete overview of these techniques, readers can refer to [Jiang and Bunke, 1991].

A polyhedron consisting of n vertices, m edges and h faces can be defined as a graph $G = (V,E,F)$ embedded on the surface of a 3D solid object, where V represent the set of vertices, E the set of undirected edges and F the set of oriented faces.

Graph-based methods: Maybe the most natural classes of algorithms that deal with symmetries of polyhedra are the one that are based on graph theory. Any symmetry of a polyhedron consists of an automorphism of the graph G (isomorphism of G onto itself) and a three-dimensional rotation. In its general form, the graph isomorphism problem is known to be NP-hard [Eppstein, 1995]. It means that no efficient (polynomial) algorithm is known to solve it. Some polynomial algorithms have however been proposed for planar graphs and especially triply connected graphs. All the algorithms that rely on graph-based methods make use of this properties to efficiently detect symmetries of the polyhedron. This however introduces some restrictions about the class of polyhedron that can be treated by these methods, due to the fact that for some types of polyhedra, the associated graph is not planar as presented on Figure 3.7 on the next page.

In theory, after computing the automorphism of the graph G of the polyhedra, one has to check geometric conditions, *i.e.* check that there exists a rotation R that map the corresponding vertices of the automorphism. [Jiang and Bunke, 1991] proposed to incorporate geometric conditions during the graph automorphism detection thus reducing the overall runtime of the

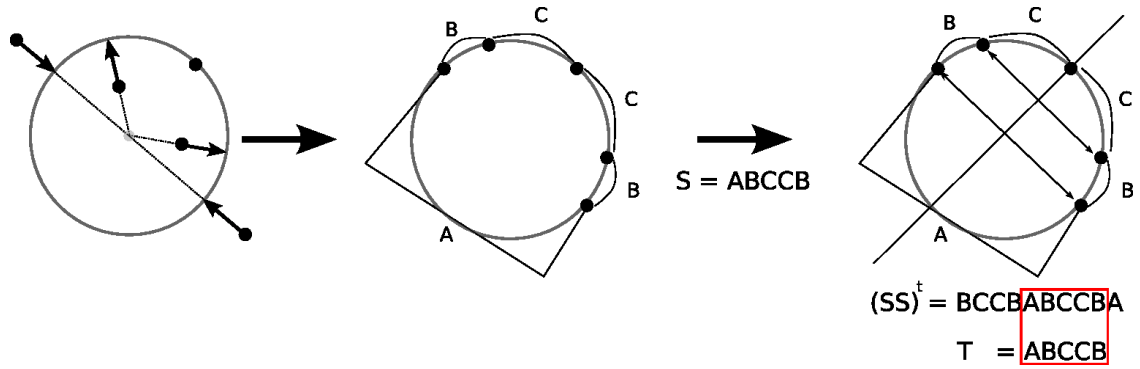


Figure 3.6: The three steps of the methods to find the symmetry of a 2D point set. Left) Each point of the set is projected to the unit circle. Middle) Each point is assigned a symbol, computed from the angular distance to its nearest neighbor. Right) The symmetry of the point set is computed with substring matching algorithm (a planar symmetry in this example).

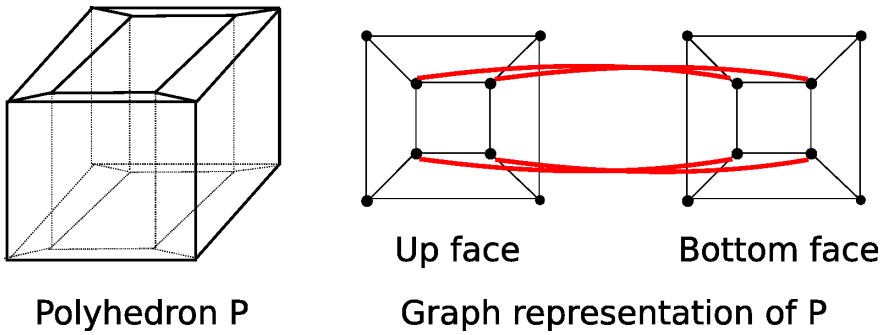


Figure 3.7: If the polyhedron has a hole, its graph representation is non-planar. The red edges show that it is not possible to connect the up and bottom faces of the polyhedron without crossing an existing edge.

algorithm.

Generate-and-test methods: Such methods proceed in two steps: generation of potential symmetry axes and check of this candidate symmetry of the polyhedron.

In [Waltzman, 1989], the authors proposed a method based on the observation that any symmetry axis of a polyhedron $G = (V, E, F)$ *without holes* passes at least through two elements of the set $V \cup E \cup F$.

This observation restrict the set of potential axis of symmetry as the axis may pass through:

1. two faces,
2. a face and a vertex,
3. a face and a edges,
4. two vertices,

5. a vertex and a edge, or

6. two edges.

The approach followed by [Waltzman, 1989] separately considers three different cases: symmetry axis which pass through at least one face (1-3), symmetry axes which pass through a vertex(4-5) and symmetry axes which pass through two edges(6).

The authors included constraint for each of potential axis of symmetry: If a symmetry axis l passes through the face f of P, then f, considered as a polygon, must be symmetric. In addition, l must be orthogonal to f and intersect f at its centroid. Another constraint for the existence of a symmetry axis passing through two edges is that the number of edges of P is divisible by two. Each potential symmetry axis that has survived to these constraints are verified on the polyhedron to see if they are real symmetry axis.

As a conclusion, these methods seem effective to compute symmetries of a certain class of polyhedra but are tedious to implement and are, once again, highly noise-sensitive. Moreover, these methods seem to be too restrictive to be applied on general 3D shapes as the topology of the polyhedra must respect its symmetries. For general 3D meshes, this means that a model must be tessellated according to its symmetries, which is rare in common Computer Graphics models.

Symmetries of 3D Objects:

For detecting the symmetries of general 3D objects, many methods have used the following fundamental property of symmetry to determine reflective and rotational symmetries:

Analytic Conditions for Symmetry: For an object in $N-$dimensional space, the term "principal axes" refer to the set of N vectors that satisfy the following conditions: a) they point in the direction of the maximum variance of the data *i.e.* variance of shape for solids and b) they are mutually orthogonal. Principal axes are eigenvectors of the object's inertia matrix (and of its covariance matrix too). Each eigenvalue is related to the variance of the shape along the corresponding eigenvector. The Principal axes are uniquely defined (up to their directions) only if the eigenvalues are distinct.

As stated in [Minovic et al., 1993], in the 3D-space, the principal axes of the covariance matrix are influenced by the equality of the corresponding eigenvalues in such a way that if two (three) eigenvalues are equal, two (three) eigenvectors are not uniquely determined. However, principal axes have the following important relations with body's properties:

Theorem 5 *The eigenspaces of the body inertia matrix are invariant under symmetries of the body. Furthemore:*

- *Any plane of symmetry of a body is perpendicular to a principal axis.*
- *Any axis of symmetry of a body is a principal axis.*

The following corollaries are the result of this theorem:

Corollary 6 *If all the eigenvalues of the inertia matrix of the body are distinct, then it has a finite number of simple i.e. reflective, rotational symmetries.*

Corollary 7 *If exactly two eigenvalues of the inertia matrix are equal, then it has a rotational or simple symmetry. The potential axis of symmetry is the principal axis that corresponds to the distinct eigenvalue.*

In the first method that relies on these properties, [Minovic et al., 1993] determined symmetries of an object using an octree representation. Their method first computes the principal axis of the object, aligns it into a canonical frame and build its associated octree. The authors also present a way of detecting symmetries when two or three eigenvalues associated to the principal axes are equal. The symmetry of the objects is obtained by computing the *symmetry degree* by recursively examining symmetric cells of the octree.

[Sun and Sherrah, 1997] proposed a method to detect symmetry based on the *Extended Gaussian Image* (EGI). This method aimed at converting the symmetry detection problem to the correlation of the Gaussian Image based on the observation that if an object is symmetric, then so is its EGI.

However, the methods relying on the principal axes of the shape to compute symmetries of the object are, in general, not reliable and cannot be used to accurately detect all symmetries. More precisely, if all eigenvalues are distinct then the symmetry will be accurately reported by these methods. It is however not rare that two or three eigenvalues are numerically equal which makes these methods failed (see Figure 3.8).

In addition, in the case where the three eigenvalues are equal, for example in the case of a cube, these methods will not be able to detect symmetries as any direction will be considered as a potential axis of symmetry. The methods using principal axes “see” such shapes as spherically symmetric objects.

When the shape has cylindrical symmetry, two of its eigenvalues are equal. In this case, the axis of rotation can be efficiently found by considering the eigenvector associated to the unique eigenvalue. On non-cylindrical symmetric shapes, this case may however happen as presented on the pillar in Figure 3.8 on the preceding page. In this case, there is no robust way of computing the real axis of reflective symmetry as the shape is considered as a cylinder. This fact is a real problem as those kind of objects often populate architectural scenes.

At this point, it is worth mentioning that all the following methods of detecting symmetries are posterior to our methods of symmetry detection.

Recently, [Mitra et al., 2006] proposed a method to compute partial, approximate or both symmetries. The authors consider the euclidean group generated by translation, reflection, rotation and uniform scaling and consider a model, or a part of a model symmetric if any element of this group left the model unchanged (see Figure 3.9). This formulation enables them to detect local symmetry which is the major contribution of their method.

We briefly detailed the various steps of their methods in the following:

- At a first step, input model needs to be sampled and a signature must be computed for each sample.

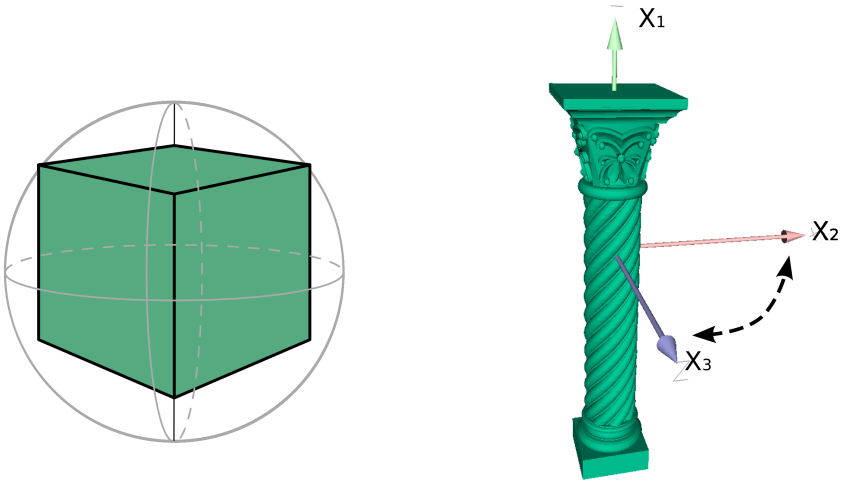


Figure 3.8: Methods relying on the principal axes of the shape may fail if all eigenvalues are not distinct. Left). As the variance of the cube is independent of the direction, all eigenvalues are equal which makes the cube equivalent to a sphere. Right) A common model were only one principal axis can be determined. In theory, this case may only happen for cylindrical symmetric object.

- This signature is computed so has to be invariant with respect to the elements of the euclidean group. The authors choose to use curvature as a signature.
- After computing these signatures, the transformation between each pair of points is computed and a new point is generated in the transformation space. The smart idea here is to see each new point in the transformation space as *a vote for a specific symmetry* (see Figure 3.10 on the next page). It must be noted that they propose a method to only compute a smaller number of couples rather than the theoretical $O(n^2)$ which need to be processed, where n is the number of samples.
- After filling the transformation space, detecting symmetry is done by clustering points

Figure 3.9: In [Mitra et al., 2006], the authors presented a technique to compute partial and approximate symmetries, based on a clustering in a well-chosen vector space.

in space Γ that have voted for the same symmetry.

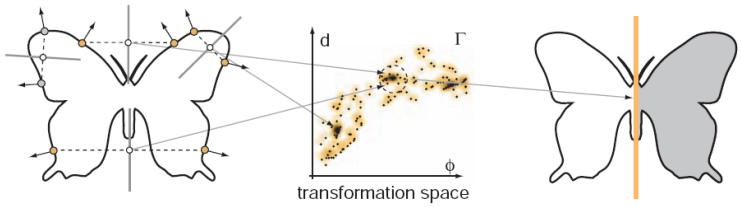


Figure 3.10: Illustration of symmetry detection in 2D by the method of [Mitra et al., 2006]. Left) Every pair of points defines a symmetry line that can be described by a distance d and an angle ϕ. This correspond to one point in the transformation space (Middle) and thus clusters of points in this space provide evidence of a symmetry (Right).

This method is efficient and can find global and local symmetry of a model. However, an important drawback of their method is that it is only able to detect discrete symmetry as shapes with continuous symmetry groups translate as a continuous curves or surfaces in the transformation space. This is the main limitation of the method as they cannot find, for example, symmetry axis of a cylindrical symmetric object.

Also recently, [Simari et al., 2006] introduced a method which can be seen as a way of compressing a model based on a particular type of symmetry. A new structure called *the folding tree* is proposed, which encode the non redundant regions of the mesh by recursively detecting planar symmetry of the mesh, or part of the mesh (see Figure 3.11).

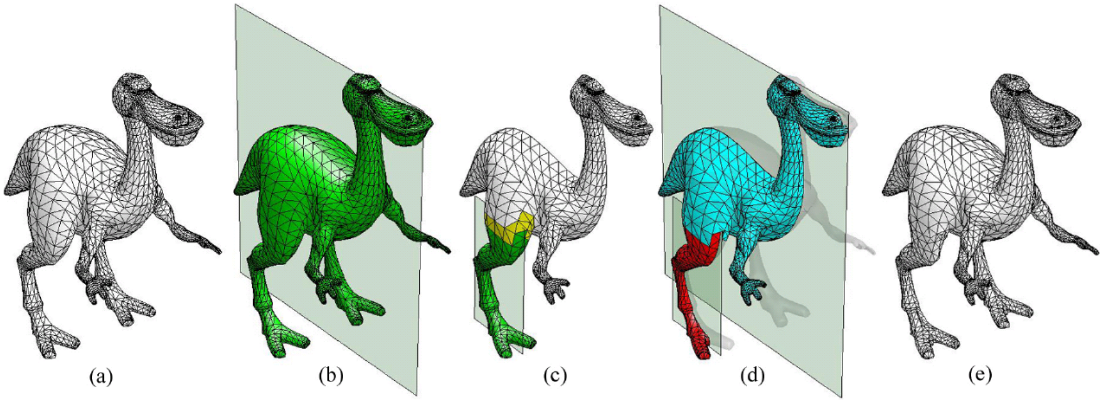


Figure 3.11: (a) Original model. (b) The detected global symmetry. (c) and (d) The method is recursively applied to the detected symmetric region. (e) The reconstructed model from the folding tree.

When searching for local symmetry, a great challenge is the robust removal of asymmetric outliers of the shape. To address this, [Simari et al., 2006] use maximum likelihood estimator (M-estimator) and process in a top-down way by first removing larger symmetric region and recursively apply their method to the newly created subpart of the object.

To remove asymmetric outliers in the mesh, their method used an $M-$estimator that map error values to an associated cost, which is controlled by a single parameter γ. This estimator is used to assign a weight to each vertex with respect to a planar symmetry. More precisely, for a plane p and a point r^i, the distance d^i is the distance from the reflected point of r^i with respect to p to the mesh. This distance is then used to assign weights to vertices where vertices with larger distance are assigned a smaller weight. These weights are then used to compute a new weighted covariance matrix C relative to the weight by:

$$C = \frac{1}{S}\sum_{i=1} nw_i(v_i - m)(v_i - m)^t$$

The axis of the planar symmetry is chosen as the eigenvector associated to the maximal eigenvalue computed from the matrix C.

By recursively applying this process, their method is able to build symmetric regions by giving more importance to in-progress building planar symmetry.

Their method is efficient for finding local planar symmetry, but the initialization of the algorithm using the covariance matrix of the mesh can lead to miss symmetry axes. Moreover, it is not clear if this method is able to robustly detect local planar symmetries in the presence of dominant assymmetric outliers

As a conclusion, to our knowledge, no method has been released that discovers perfect discrete and continuous symmetries (up to a threshold) of a 3D model. As a lot of methods have focused on finding planar symmetries [Simari et al., 2006] or rotational symmetry [Jiang and Bunke, 1991; Brass and Knauer, 2004] or both [Mitra et al., 2006] none of them is able to detect all global symmetries of a shape in a unified framework. We will present, in the next chapter, a fast and robust method that is able to detect the complete symmetry group of a model (discrete and continuous) as well as a method to find symmetry of a set of assembled objects computed from the symmetries of each element of the set.

The second part of this chapter focuses on symmetry seen as a continuous feature.

3.2.2 Symmetry as a continuous feature

At the time were most methods that search for symmetry could not allow noise in input data, [Zabrodsky et al., 1995] introduce a continuous measure of symmetry which transform the binary question: “Does a model have a given symmetry ?” to the continuous question: “How much of a given symmetry does a model have ?”.

In this paper, the distance between two shapes P and Q each formed of points P_i and Q_i is defined as:

$$d(P,Q) = \frac{1}{n}\sum_{i=0}^{n} ||P_i - Q_i||^2$$

The *Symmetry Transform ST* of a shape P is the symmetric shape closest to P in terms of the metric d. The *Symmetry Distance SD* is defined as the minimum amount of work needed to transform a model into a symmetric one, in other words:

$$SD_P = d(P, ST(P))$$

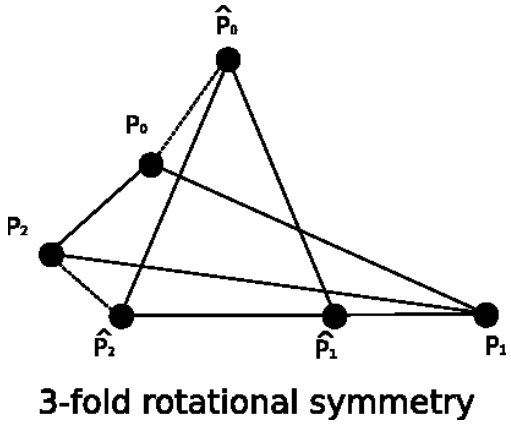


Figure 3.12: The initial shape, formed by points P_i and its "nearest" shape, formed of points $\hat{P}_i$ that has 3−fold rotational symmetry.

The authors present a methods to compute the Symmetry Transform of a shape. An example of such Symmetry Distance is presented in Figure 3.12 on the next page.

This method is however limited as it can only consider symmetry with respect to only one point or plane at a time.

One of the first approach develop to evaluate the symmetries of a 2D model, at every symmetries, were developed using shape descriptor. These methods [Sun, 1995; Marola, 1989] rely on a generalization of discrete substring matching to continuous correlation with the Fast Fourier Transform. These methods compute the symmetries of a model by using correlation to compare the shape descriptor of a 2D model with all of its rotations and reflections. This approach is a general one that could be applied to any shape descriptor that represents a model with a function defined on a circle, or in 2D.

In the 3D case, where shape descriptors are represented as spherical function or as 3D function, [Kazhdan et al., 2003a, 2004b] use the analogous of FFT on the sphere, namely the Fast Harmonic Transform to compute the measure of all symmetries. In their work, the authors present a mathematical foundation for symmetry measure, where the key points are detailed below:

Definition 8 *Given a vector space V and a group G that acts on V, we say that $v \in V$ is symmetric with respect to G if $\gamma(v) = v$ for all $\gamma \in G$. In this case, we say that v is $G-$invariant.*

Definition 9 *The symmetry distance of a vector v with respect to group G is the L_2 distance to the nearest vector $G-$invariant:*

$$SD_G(v) = \min_{w \in G-inv} ||v - w||$$

As the vector that are $G-$invariant define a subspace of G, it follows that the nearest $G-$invariant vector that minimize SD_G is the projection of v onto the subspace of invariant vector, that is:

$$SD_G = ||v - \gamma(v)|| = ||v - \Pi_G(v)|| \tag{3.2}$$

where Π_G represent the projection onto the subspace invariant under the action of G. It must be noticed that the symmetry distance defined in Equation (3.2) is coherent with the symmetry distance defined in [Zabrodsky et al., 1995].

As the elements of G are orthogonal transformations, a theorem from representation theory can be applied that states that a projection of a vector onto the subspace invariant under the action of an orthogonal group is the average of the vector over the different elements in this group. Thus, we have:

$$SD_G^2 = ||v||^2 - \frac{1}{|G|}\sum_{\gamma \in G} < v, \gamma(v) >$$

which turns the problem of symmetry measure into a problem of function correlation. To compute the symmetry measure of a shape descriptor represented as a spherical function, their method first express this function on the spherical harmonic basis and use rotation property of spherical harmonic to derive an algorithm in $O(b^4)$, where b represents the bandwidth of the spherical harmonic decomposition, to compute symmetry descriptor for a given symmetry.

Examples of symmetry descriptors are presented in Figure 3.13. The descriptors are represented by scaling points on the unit sphere in proportion to the measure of symmetry, so that points corresponding to axes of near symmetry are pushed out from the origin and corresponding to axes of near anti-symmetry are pulled in the origin.

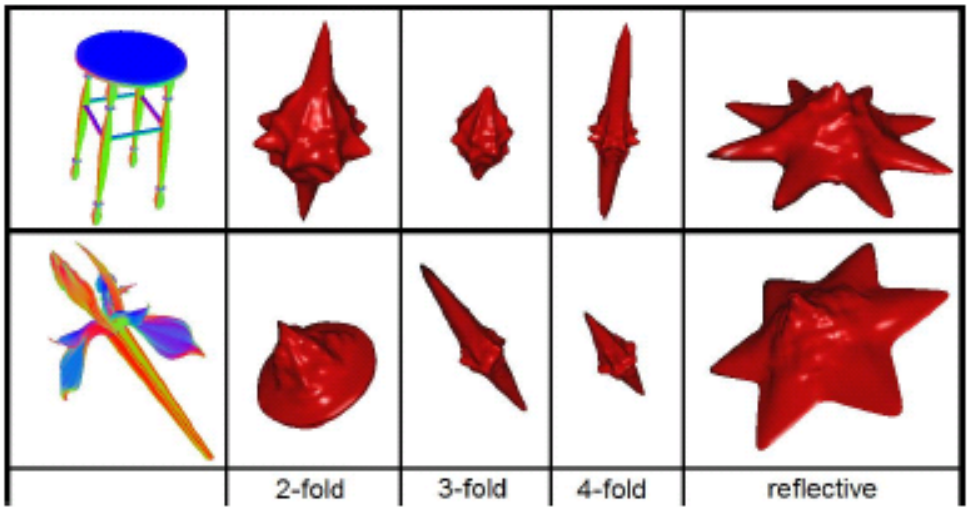


Figure 3.13: Examples of symmetry descriptor computed for a planar symmetry with the method of [Kazhdan et al., 2004b]. The descriptors are represented by scaling points on the unit sphere in proportion to the measure of symmetry, so that points corresponding to axes of near symmetry are pushed out from the origin and corresponding to axes of near anti-symmetry are pulled in the origin.

The main contribution of this method is the ability to quickly obtain a continuous measure of symmetry for all possible symmetries. However, as will be described further, having a continuous measure of symmetry is not the guarantee to have a perfect symmetry, *i.e.* it is not

easy to get back to the binary question: “Does a model have a given symmetry ?”. It requires to accurately find local maxima of a tabulated function which is non-trivial.

[Podolak et al., 2006] build upon this work to introduce a Planar-Reflective Symmetry Transform (PRST). This work can been seen as a extension of the work of [Kazhdan et al., 2004b] as it aim to compute a measure of the planar symmetry of an object *with respect to all planes inside the object bounding volume*. For any given plane, the PRST indicates the degree of symmetry the object exhibits with respect to it.

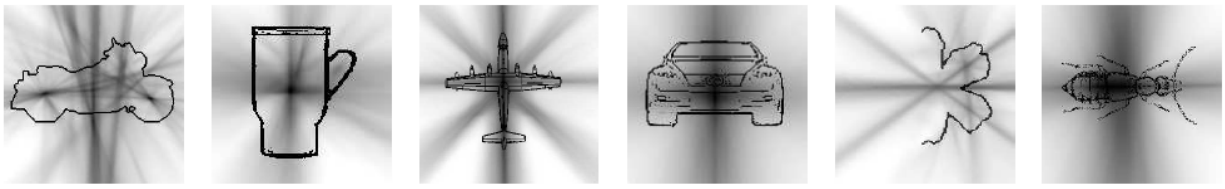


Figure 3.14: [Podolak et al., 2006] presents a method to compute a measure of symmetry of the planar transform of an object with respect to all planes inside the object bounding volum. Each point of space is colored by the symmetry measure of the plane with the largest symmetry passing through it, with darker lines representing greater symmetries.

A discretized version of the PRST is computed on a uniform grid of 64^3 resolution and for efficiency of computation, the authors propose an importance sampling scheme, in which pairs of randomly selected points vote for the plane between them. These votes are accumulated in discrete bins over polar parameters in a manner reminiscent of the Hough transform.

The second contribution of this paper is to provide a method to refine local maxima of the tabulated PRST. These local maxima correspond to planes for which the object exhibits a degree of local or global symmetry. The authors also provide multiple applications based on this knowledge such as mesh segmentation or viewpoint selection (see Figure 3.15).

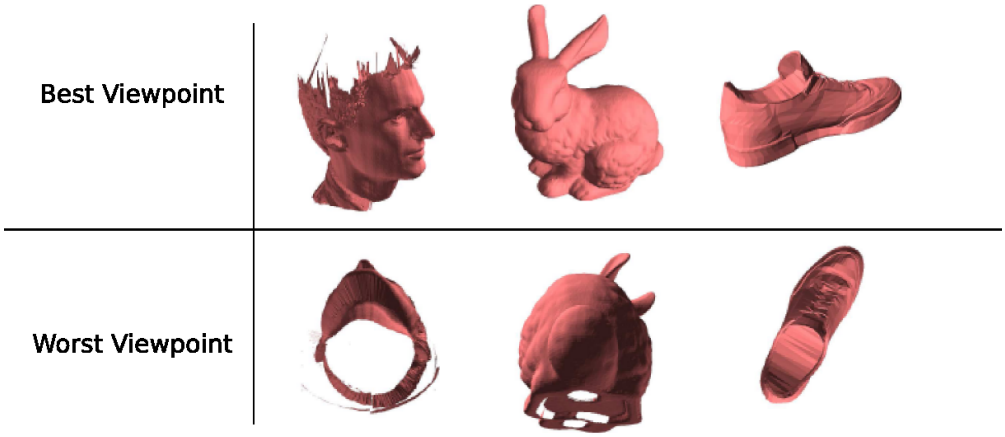


Figure 3.15: In [Podolak et al., 2006], the selection of the best viewpoint of an object is based on the assumption that symmetry present of the object present redundant information and is therefore to be avoided

4

Generalized moments And Symmetries

In this section we introduce a new class of functions: the *generalized moments* of a shape. We then show that these functions have at least the same symmetries than the shape itself, and that their own symmetries can be computed in a very efficient way.

4.1 Definition

For a surface $\mathcal{S}$ in a 3-dimensional domain, we define its *generalized moment* of order $2p$ in direction $\boldsymbol{\omega}$ by

$$\mathcal{M}^{2p}(\boldsymbol{\omega}) = \int_{\boldsymbol{s}\in\mathcal{S}} \|\boldsymbol{s}\times\boldsymbol{\omega}\|^{2p}\,\mathrm{d}\boldsymbol{s} \tag{4.1}$$

In this definition, $\boldsymbol{s}$ is a vector which links the center of gravity of the shape (placed at the origin) to a point on the surface and $\mathrm{d}\boldsymbol{s}$ is thus a infinitesimal surface element. $\mathcal{M}^{2p}$ itself is a directional function.

It should be noted that, considering $\mathcal{S}$ to have some thickness dt, the expression $\mathcal{M}^2(\boldsymbol{\omega})dt$ (i.e the generalized moment of order 2) corresponds to the moment of inertia of the thin shell $\mathcal{S}$ along $\boldsymbol{\omega}$, hence the name of these functions. Furthermore, the choice of an even exponent and a cross-product will lead to very interesting properties.

4.2 Shape symmetries and moments

Symmetry properties of a shape translate into symmetry properties of its moment functions. We now introduce a theorem that we will be rely on (see proof in Appendix):

Theorem 8 *: Any symmetry I of a shape $\mathcal{S}$ also is a symmetry of all its $\mathcal{M}^{2p}$ moment functions:*

$$I(\mathcal{S}) = \mathcal{S} \quad \Rightarrow \quad \forall\boldsymbol{\omega}\ \ \mathcal{M}^{2p}(I(\boldsymbol{\omega})) = \mathcal{M}^{2p}(\boldsymbol{\omega})$$

Furthemore, if $\mathcal{M}^{2p}$ has a symmetry I with axis ω, then the gradient of $\mathcal{M}^{2p}$ is null at ω:

$$\forall \omega \quad \mathcal{M}^{2p}(I(\omega)) = \mathcal{M}^{2p}(\omega) \quad \Rightarrow \quad (\nabla \mathcal{M}^{2p})(\omega) = 0$$

This theorem implies that the axes of the symmetries of a shape are to be found in the intersection of the sets of directions which zero the gradients of each of its moment functions. The properties are not reciprocal however: once the directions of the zeros of the gradients of the moment functions have been found, they must be checked on the shape itself to eliminate false positives.

4.3 Efficient computation

At first sight, looking for the zeros of the gradient of the moment functions requires precise and dense sampling of these functions, which would be very costly using their integral form of Equation 4.1. We thus present an efficient method to compute the generalized even moment functions of a shape, using spherical harmonics. In particular, we can accurately compute the spherical harmonic coefficients of the moment functions without sampling these functions. The search for zeros in the gradient will then be performed efficiently on the spherical harmonic decomposition itself.

Spherical harmonics We use real-valued spherical harmonics [Hobson, 1931] to represent directional functions. Real spherical harmonics are defined, for integers $l \geq 0$ and $-l \leq m \leq l$, by:

$$Y_l^m(\theta,\varphi) = \begin{cases} \sqrt{2}\ N_l^m P_l^m(cos\theta)\cos(m\varphi) & \text{for} \quad 0 < m \leq l \\ N_l^m P_l^0\ (cos\theta) & \text{for} \quad m = 0 \\ \sqrt{2}\ N_l^m P_l^{-m}(cos\theta)\sin(m\varphi) & \text{for} \quad -l \leq m < 0 \end{cases}$$

where P_l^m are the associated Legendre polynomials; the normalization constants N_l^m are such that the spherical harmonics form an orthonormal set of functions for the scalar product:

$$< f, g >= \int_{\|\omega\|=1} f(\omega) g(\omega)\, \mathrm{d}\omega$$

This corresponds to choosing:

$$N_l^m = \sqrt{\frac{2l+1}{4\pi}\frac{(l-|m|)!}{(l+|m|)!}}$$

We will use the following very powerful property of spherical harmonics: any spherical harmonic of degree l can be expressed in a rotated coordinate system using harmonics of same degree and coefficients depending on the rotation R:

$$Y_l^m \circ R = \sum_{-l \leq m' \leq l} D_l^{m,m'}(R) Y_l^{m'} \tag{4.2}$$

Any combination of spherical harmonics of degree less than l can therefore be expressed in a rotated coordinate system using spherical harmonics of degree less than l without loss of information. Coefficients $D_l^{m,m'}(R)$ can efficiently be obtained using recurrence formulae [Ivanic and Ruedenberg, 1996] or directly computed [Ramamoorthi and Hanrahan, 2004].

Computation of moment functions As defined by Equation 4.1, the $2p-$moment function of a shape $\mathcal{S}$ is expressed as:

$$\begin{aligned}\mathcal{M}^{2p}(\boldsymbol{\omega}) &= \int_{\boldsymbol{s}\in\mathcal{S}} \|\boldsymbol{s}\times\boldsymbol{\omega}\|^{2p}\,\mathrm{d}\boldsymbol{s} \\ &= \int_{\boldsymbol{s}\in\mathcal{S}} \|\boldsymbol{s}\|^{2p}\sin^{2p}\beta\,\mathrm{d}\boldsymbol{s}\end{aligned}$$

In this expression, β is the angle between s and ω.

Function $\beta\mapsto\sin^k\beta$ has angular dependence on β only and therefore decomposes into zonal harmonics (i.e. harmonics Y_l^m for which $m=0$). Performing the calculation shows that when k is even, the decomposition is finite. Setting $k=2p$, we obtain :

$$\sin^{2p}\beta = \sum_{l=0}^{p} S_p^l Y_{2l}^0(\beta,.)$$

with:

$$S_p^l = \frac{\sqrt{(4l+1)\pi}}{2^{2l}}\sum_{k=l}^{2l}(-1)^k\frac{2^{2p+1}p!(2k)!(p+k-l)!}{(2(p+k-l)+1)!(k-l)!k!(2l-k)!} \tag{4.3}$$

We provide the corresponding derivation and the proof of the finite decomposition in the appendix section of this paper, for the sake of completeness.

Let $R_{\boldsymbol{s}}$ be a rotation which maps z, unit vector along $z-$axis, to s. Using Equation 4.2 for rotating the Y_{2l}^0 zonal harmonics, we have :

$$\sin^{2p}\beta = \sum_{l=0}^{p} S_p^l \sum_{m=-2l}^{2l} D_{2l}^{0,m}(R_{\boldsymbol{s}})Y_{2l}^m(\boldsymbol{\omega})$$

And finally:

$$\mathcal{M}^{2p}(\boldsymbol{\omega}) = \sum_{l=0}^{p}\sum_{m=-2l}^{2l} C_{2l,m}^{2p}Y_{2l}^m(\boldsymbol{\omega}) \tag{4.4}$$

using

$$C_{2l,m}^{2p} = S_p^l\int_{\boldsymbol{s}\in\mathcal{S}}\|\boldsymbol{s}\|^{2p}D_{2l}^{0,m}(R_{\boldsymbol{s}})\,\mathrm{d}\boldsymbol{s} \tag{4.5}$$

Equation 4.4 says that $\mathcal{M}^{2p}$ decomposes into a *finite* number of spherical harmonics, and Equation 4.5 allows us to directly compute the coefficients. The cost of computing $\mathcal{M}^{2p}$ is therefore $(p+1)(2p+1)$ surface integrals (one integral per even order of harmonic, up to order $2p$). This is much cheaper than the alternative method of computing the scalar product of $\mathcal{M}^{2p}$ as defined by Equation 4.1, with each spherical harmonic basis function: this would indeed require many evaluations of $\mathcal{M}^{2p}$, which itself is defined as a surface integral.

Furthermore, numerical accuracy is only concerned when computing the $C_{2k,p}^m$ coefficients, and we can now compute both $\mathcal{M}^{2p}$ and its gradient analytically from Equation 4.4.

4.4 Finding symmetries of a single shape

In this section, we present our algorithm for identifying symmetries of a shape seen as a single entity, as opposed to the algorithm presented in the next section where the shape is considered as an aggregation of multiple sub-parts. For a given shape, we want to determine the axis $\boldsymbol{X}$ and the (λ, α) parameters of the potential isometries, using the generalized moment functions, and check the isometries found against the actual shape.

Central symmetries ($\lambda = -1$ and $\alpha = \pi$) form a specific case, since by construction, $\mathcal{M}^{2p}$ always has a central symmetry. Because central symmetries also do not require an axis, we treat this case directly while checking the other candidate symmetries on the shape itself in Section 4.4.3.

4.4.1 Determination of the axis

As we saw in Section 4.2 the axis of isometries which let a shape globally unchanged also zero the gradient of the generalized even moments of this shape. We thus obtain a superset of them by solving for:

$$\boldsymbol{\nabla}(\mathcal{M}^{2p})(\boldsymbol{\omega}) = \mathbf{0}$$

In a first step, we estimate a number of vectors which are close to the actual solutions, by refining the sphere of directions starting from an icosahedron. In each face, the value of $\|\boldsymbol{\nabla}(\mathcal{M}^{2p})(\boldsymbol{\omega})\|^2$ is examined in several directions and faces are sorted by order of the minimal value found. Only faces with small minimum values are refined recursively. The number of points to look at in each face as well as the number of faces to keep at each depth level are constant parameters of the algorithm.

In a second step we perform a steepest descent minimization on $\|\boldsymbol{\nabla}(\mathcal{M}^{2p})(\boldsymbol{\omega})\|^2$, starting from each of the candidates found during the first step. For this we need to evaluate the derivatives of $\|\boldsymbol{\nabla}(\mathcal{M}^{2p})\|$, which we do using analytically computed second order derivatives of the spherical harmonics along with Equation 4.4. The minimization converges in a few steps because starting positions are by nature very close to actual minima. This method has the double advantage that (1) the derivatives are very efficiently computed and (2) no approximation is contained into the calculation of the direction of the axis beyond the precision of the calculation of the $C^{2p}_{2l,m}$ coefficients.

During this process, multiple instances of the same direction can be found. We filter them out by estimating their relative distance. While nothing in theory prevents the first step from missing the area of attraction of a minimum, it works very well in the present context. Indeed, moment functions are very smooth, and shapes having two isometries with very close – yet different – axis are not common.

Finally, because all moment functions whatever their order, must have an extremum in the direction of the axis of the symmetries of the shape, we compute such sets of directions for multiple moment functions (e.g. $\mathcal{M}^4$, $\mathcal{M}^6$ and $\mathcal{M}^8$), but keep only those which simultaneously zero the gradient of all these functions – which in practice leaves none or very few false positives to check for.

4.4.2 Determination of rotation parameters

After finding the zero-directions for the gradient of the moment functions, we still need to find the parameters of the corresponding isometric transforms. This is done deterministically by studying the spherical harmonic coefficients of the moment functions themselves. We use the following properties:

Property 1 *: A function has a mirror-symmetry S_z around the $z = 0$ plane if and only if all its spherical harmonic coefficients for which $l + m$ is even are zero (i.e. it decomposes onto $z-$symmetric harmonics only). In the specific case of the moment functions:*

$$\forall \omega \;\; \mathcal{M}^{2p}(\omega) = \mathcal{M}^{2p}(S_z\omega) \;\; \Leftrightarrow \;\; m \equiv 0 (mod\ 2) \Rightarrow C^{2p}_{2l,m} = 0$$

Property 2 *: A function has a revolution-symmetry around the z axis if and only if it decomposes onto zonal harmonics only, i.e.*

$$\forall l \;\; \forall m \quad m \neq 0 \;\; \Rightarrow \;\; C^m_l = 0$$

Property 3 *A function is self similar through a rotation R_α of angle α around z if and only if all its spherical harmonic coefficients C^m_l verify:*

$$\forall l \;\; \forall m \quad C^m_l = \cos(m\alpha)C^m_l - \sin(m\alpha)C^{-m}_l \tag{4.6}$$

Property 3 can be adapted to check if the function is self-similar through the composition of a rotation and a symmetry with the same axis (*i.e.* the case $\lambda = -1$). In this case the equation to be checked for is:

$$\forall l \;\; \forall m \quad (-1)^{l+m}C^m_l = \cos(m\alpha)C^m_l - \sin(m\alpha)C^{-m}_l \tag{4.7}$$

These properties are easily derived from the very expression of the spherical harmonic functions [Hobson, 1931].

Before using these properties, the moment function must be expressed in a coordinate system where the z axis coincides with the previously found candidate axis. This is performed using the rotation formula in Equation 4.2. Then checking for properties 1 and 2 is trivial provided that some tolerance is accepted on the equalities. Using property 3 is more subtle: coefficients of the function are first examined by order of decreasing m. For $\lambda = 1$ for instance, when the first non zero value of C^m_l is found, Equation 4.6 is solved by:

$$\tan\frac{m\alpha}{2} = \frac{C^{-m}_l}{C^m_l} \quad \text{i.e.} \quad \alpha = \frac{2}{m}\arctan\left(\frac{C^{-m}_l}{C^m_l}\right) + \frac{k\pi}{m}$$

then all the remaining coefficients are checked with the obtained values of α. If the test passes, then α is the angle of an existing rotation-symmetry for the moment function. A very similar process is used to search for α when $\lambda = -1$.

The error tolerance used when checking for properties 1, 2 and 3 can be considered as a way of detecting approximate symmetries on objects. We will show in the results section that symmetries can indeed be detected on noisy data, such as scanned models.

4.4.3 Filtering results

The condition extracted from Theroem 8 is a necessary condition only. To avoid false positives, the directions and rotation angles obtained from the moment functions must therefore be verified on the shape itself. We do this using a *symmetry measure* inspired by the work of Zabrodsky et al. [Zabrodsky et al., 1995]. Let $\mathcal{S}$ and $\mathcal{R}$ be two tessellated shapes. Let $V_\mathcal{S}$ and $V_\mathcal{R}$ be the mesh vertices of $\mathcal{S}$ and $\mathcal{R}$. We define the *measure* d_M between $\mathcal{S}$ and $\mathcal{R}$ by:

$$d_M(\mathcal{S},\mathcal{R}) = \max_{p\in V_\mathcal{S}}(\min_{q\in\mathcal{R}}\|p-q\|) \tag{4.8}$$

The *symmetric measure* $d_A(\mathcal{S})$ of a shape $\mathcal{S}$ with respect to a symmetry A is then defined by:

$$d_A(\mathcal{S}) = \max(d_M(\mathcal{S},A\mathcal{S}), d_M(A\mathcal{S},\mathcal{S}))$$

It should be noted that this definition is different from that of the *Hausdorff distance* since, in Equation 4.8, not all points of $\mathcal{S}$ are considered but only the mesh vertices, whereas all points of $\mathcal{R}$ are used. However, because $\mathcal{S}$ is polyhedral, $d_A(\mathcal{S}) = 0$ still implies that $A\mathcal{S} = \mathcal{S}$.

From an implementation point of view, we choose a treshold ε, a fraction of the boudning box diagonal and keep A as a symmetry if $d_A(\mathcal{S}) \leq \varepsilon$.

Computing d_A is costly, but fortunately we only compute it for a few choices of A which are the candidates we found at the previous step of the algorithm. This computation is much cheaper than computing a full symmetry descriptor [Kazhdan et al., 2004b], for a sufficient number of directions to reach the precision of our symmetry detection algorithm.

4.4.4 Results

Complete example

The whole process is illustrated in Figure 4.1. Starting from the original object (a), the moment functions of orders $4, 6$ and 8 are computed (See *e.g.* $\mathcal{M}^8$ in (b)). The gradients of these moments are then computed analytically (c), and used for finding the directions of the minima. The unfiltered set of directions contains 7 directions, among which only 3 are common extrema of $\mathcal{M}^4$, $\mathcal{M}^6$ and $\mathcal{M}^8$. This set of 3 directions ($\boldsymbol{D}_1$,$\boldsymbol{D}_2$ and $\boldsymbol{D}_3$) must contain the axes of the symmetries of the shape. After checking the symmetry axis and parameters on the actual shape, $\boldsymbol{D}_1$ reveals to be the axis of a 2-fold symmetry, which is the composition of the two remaining mirror-symmetries of axes $\boldsymbol{D}_2$ and $\boldsymbol{D}_3$.

The example of the cube, shown in Figure 4.2 illustrates the extraction of rotations and mirror-symmetries. Experiments have shown that our method finds all 48 symmetries whatever the coordinate system the cube is originaly expressed in.

Robustness tests

We now study the sensitivity of our method to small perturbations of the 3D model in two different ways :

1. **Noise :** We randomly perturb each vertex of each polygon independently in the original model by a fraction of the longest length of the model's bounding box.

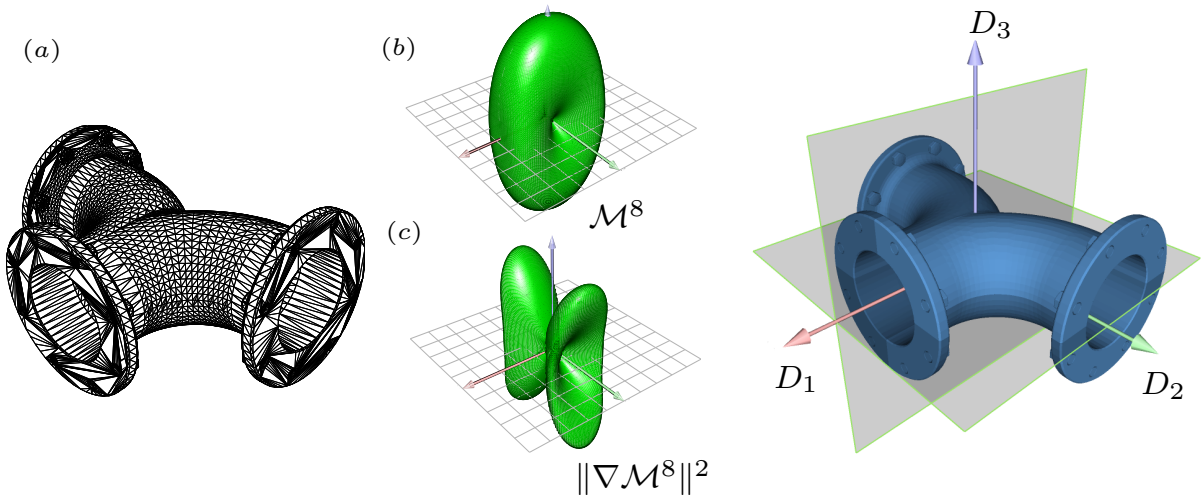


Figure 4.1: Extraction of symmetries for a single shape. Starting from the original shape (a), generalized moments (b) and their gradients (c) are computed. The set of their common extrema directions contains the axes of the symmetries of the shape, depicted at right. Here, both mirror-symmetries have been found as well as the 2-fold rotational symmetry. Note that the original shape is neither convex nor star-shaped, and that the mesh is not consistent with the symmetries of the geometry.

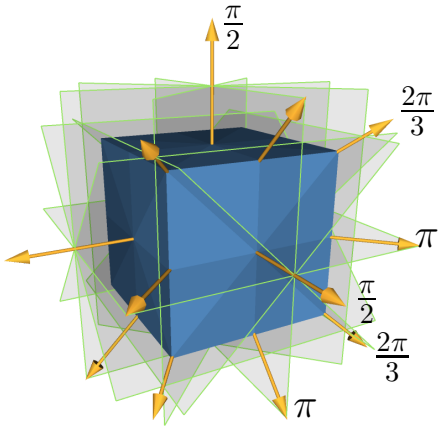


Figure 4.2: Mirror-symmetries and rotational-symmetries found by our algorithm for a cube (for clarity, not all elements are represented).

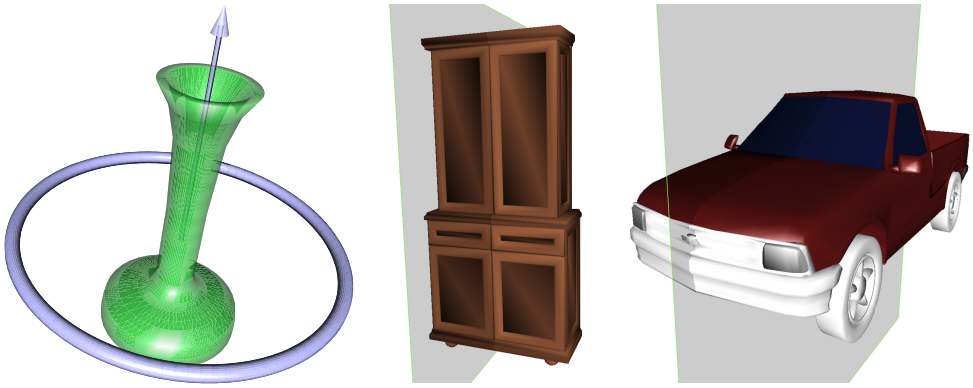


Figure 4.3: View of the three 3D models used in the robustness tests presented in Figure 4.4 shown with their symmetries. For the sake of clarity, we chose models with only one symmetry each.

2. **Delete :** We randomly delete a small number of polygons in the model.

We use a set of three models to test the robustness of our method. These models are shown in Figure 4.3 as well as their symmetry. For sake of clarity, we use objects with only one symmetry.

In order to test the robustness of the method, we progressively increase the magnitude of the noise and let the algorithm automatically detect the symmetry. In our robustness tests, we consider shapes as single entities and use the first algorithm presented in Section 4.4 to detect these symmetries. To evaluate the reliability of the results, we compute the angular deviation between the found axis of symmety and the real one i.e. computed with no noise. In our experiments, noise magnitude varies from 0 to 1% of the longest length of the model's bounding box and the number of deleted polygons ranges from 0 to 5% of the total number of polygons in the model (see Figure 4.4).

The results of these experiments show that for small variations, our method has approximatively linear dependency regarding noise and delivers high-quality results even for non-perfect symmetries. These statistical results can also be used to derive an upper bound on the mean angular error obtained as a function of the noise in the model.

Application to scanned models

We present in Figure 4.5 examples of applying the single-shape algorithm to scanned models, retreived from a web database and used as is (See at `http://shapes.aim-at-shape.net`). Our algorithm perfectly detects all the parameters of candidate symmetries for all these shapes. When testing these symmetries, one should allow a large enough symmetry distance error (as defined in Section 4.4.3) because these models are by nature not perfectly symmetric.

4.4.5 Discussion

The $\mathcal{M}^{2p}$ functions being trigonometric polynomials on the sphere, they have a maximum number of strict extrema depending on p: the larger p, the more $\mathcal{M}^{2p}$ is able to capture the

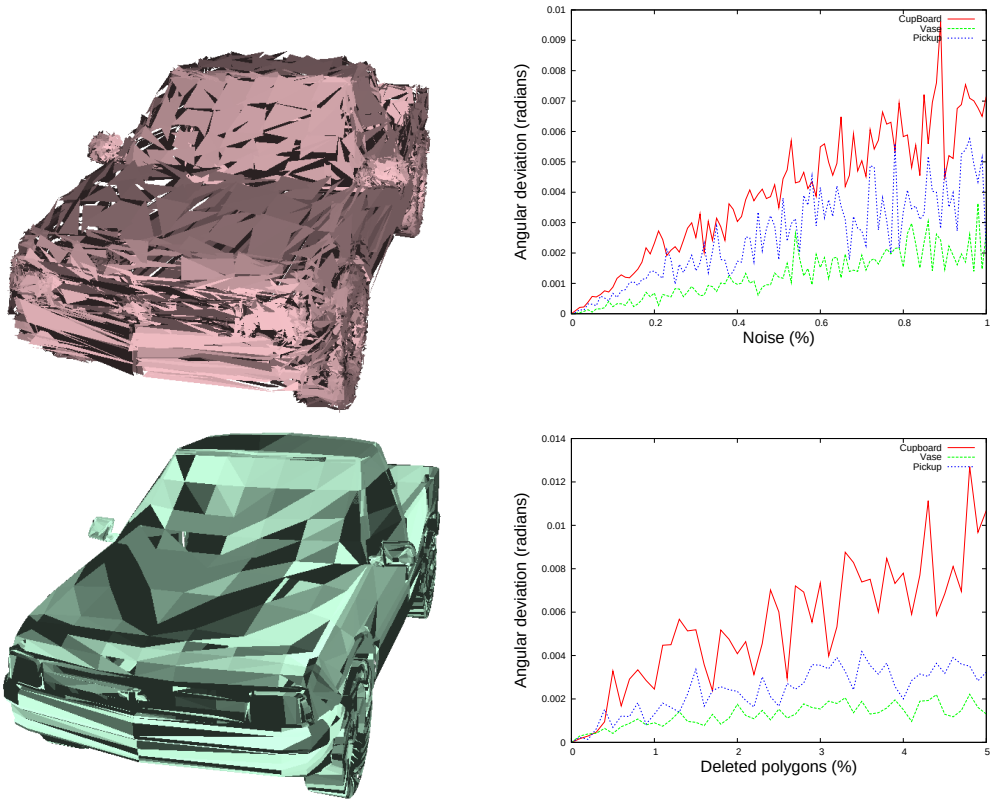


Figure 4.4: We test the sensitivity of the method to noise by progressively increasing noise magnitude and letting the algorithm detect the symmetry for each of our three test models. We evaluate the accuracy of the results by computing the angular deviation between the axis found and the axis of the symmetry of the original model. *Top row:* We perturb each vertex of each polygon independently by a fraction of the longest length of the bounding box on each of the three test models. Left figure shows a noisy "pickup" model with a noise magnitude of 1% and right figure shows angular deviation evolution for the three models, for a magnitude ranging from 0% to 1%.*Bottom row:* We randomly delete polygons of the models. Left figure shows a noisy pickup obtained by deleting 5% of the polygons and right figure shows angular deviation evolution by deleting 0% to 5% of the polygons of the three models. As can be seen from the curve, for small variation of the models, our method has approximatively linear dependency regarding noise and delivers high-quality results even for non-perfect symmetries.

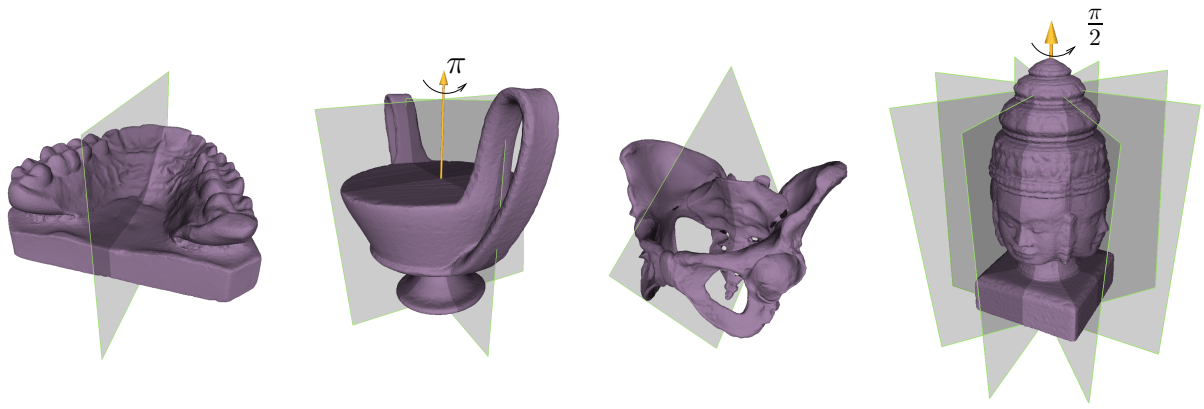


Figure 4.5: Our algorithm perfectly detects approximate symmetries of scanned models. Detecting those symmetries requires relaxing the constraints when checking candidate symmetries on the model. Please note that these scanned models are by nature neither *axis-aligned* nor tesselated according to their symmetries. This illustrates the fact that our algorithm does not depend on the coordinate system nor on the mesh of the objects.

Model	Teeth	Vase	Pelvis	Angkor statue
# polygons	233,204	76,334	50,000	163,054
Computing moments*	33.7	11.8	7.26	23.26
Finding parameters	0.4	0.6	0.4	0.7
Checking candidates	9.4	11.1	5	12.2
Total	43.5	23.5	12.66	36.16

Figure 4.6: Computation times in seconds for the four scanned models presented in Figure 4.5.**Global computation times for moments of order 2 to 8

information of a symmetry, i.e. to have an extremum in the direction of its axis. But because all moment functions *must* have a null gradient in this direction (according to Theorem 8), these extrema are "bound" to become non-strict extrema for small values of p, and $\mathcal{M}^{2p}$ is forced to be constant on a sub-domain of non-null dimension. Using the cube as an example, in which case $\mathcal{M}^2$ is a constant function: a trigonometric polynomial of order 2 can simply not have enough strict extrema to represent all 12 distinct directions of the symmetries of the cube.

In all tests we conducted however, using moments up to order 10 has never skipped any symmetry on any model. But it would still be interesting to know the exact maximum number of directions permitted by moments of a given order.

4.5 Finding symmetries of groups of objects

In Section 4.4 we have presented an algorithm for finding the symmetries of single shapes. In this section, we present a *constructive* algorithm which recovers the symmetries of a group of objects – which we call *tiles* to indicate that together they form a larger object –, from the symmetries and positions of each separate tile.

The constructive algorithm first computes (if necessary) the symmetries of all separate tiles using the single shape algorithm. Then it detects which tiles are similar up to an isometric transform and finds the transformations between similar tiles. Then it explores all one-to-one mappings between tiles, discarding mappings which do not correspond to a symmetry of the group of tiles as a whole.

Section 4.5.2 explains how we detect similar tiles and Section 4.5.3 details the algorithm which both explores tile-to-tile mappings and finds the associated symmetry for the whole set of tiles.

Because it is always possible to apply the algorithm presented in Section 4.4 to the group of tiles, considering it as a single complex shape, questioning the usefulness of the constructive method is legitimate. For this reason we will explain in Section 4.5.5 in which situations the constructive method is more preferable to the algorithm for single shapes; but let us first explain the method itself.

4.5.1 Computing the symmetries of each tile

If not available, the symmetries of each tile are computed using the algorithm presented in Section 4.4. When assembling known objects together, the economy of this computation can of course be performed by simply computing the symmetries of one instance for each class of different tiles.

4.5.2 Detecting tiles congruency

In this subsection we introduce a shape descriptor suitable for detecting whether two shapes are identical up to an — unknown — isometry. We will use this tool for classifying tiles before trying to find a mapping of a composite object onto itself.

Let $\mathcal{S}$ be a shape and $C^{2p}_{2l,m}$ the spherical harmonic coefficients of its generalized even moment functions $\mathcal{M}^{2p}$ up to an order p. Our shape descriptor is defined as the $p(p+1)/2$-vector

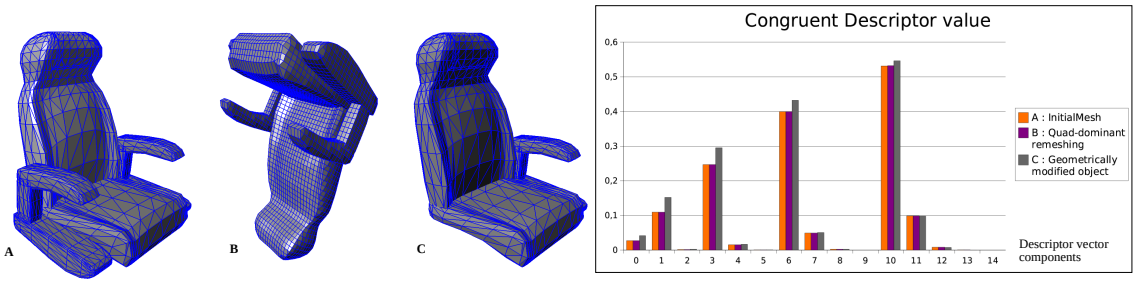


Figure 4.7: This figure illustrates the reliability of our congruency descriptor (as defined by Equation 4.9): Two identical objects meshed differently and expressed in two different coordinate systems (A and B) have extremely close descriptor vectors, but a slightly different object (C) has a different descriptor. The graphics on the *right* shows each component of the three descriptors.

obtained by packing together the frequency energy of the spherical harmonic decomposition of all moments of $\mathcal{S}$ up to order p:

$$D_{2p} = \left[d_0^0, d_0^2, d_2^2, \ldots, d_0^{2p}, d_2^{2p} \ldots d_{2p}^{2p} \right] \tag{4.9}$$

with

$$d_{2l}^{2k} = \sum_{-2l \leq m \leq 2l} \left(C_{2l,m}^{2k} \right)^2 \tag{4.10}$$

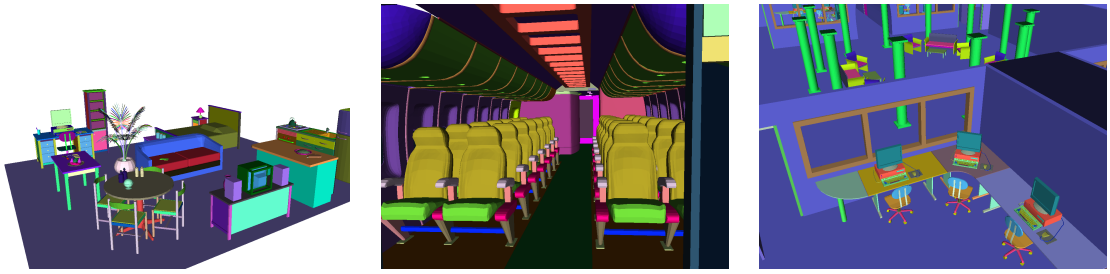
It has been shown by Kazhdan et al. [Kazhdan et al., 2003b] that d_l^k, as defined in Equation 4.10, does not depend on the coordinate system the spherical harmonic decomposition is expressed in. This means that each d_{2l}^{2p}, and therefore D_{2p} itself, is not modified by isometric transforms of the shape. Mirror-symmetries do not affect d_{2l}^{2p} either, since they only change the sign of the coefficient for some harmonics in a coordinate system aligned with the axis.

Two tiles A and B are considered to be similar up to an isometric transform, at a precision ε, when:

$$\| D_{2p}(A) - D_{2p}(B) \| < \varepsilon$$

Theoretically, this shape descriptor can produce false positives, i.e. tiles that are not congruent but have the same descriptor, but it can not produce false negatives because of its deterministic nature. Our experiments have shown that using moments up to order 6 produces a sufficiently discriminant shape descriptor on all test scenes. This is illustrated in Table 4.9 where we present the average "precision" value, that is the percentage of matched tiles that are actually identical up to an isometric transform, for a set of architectural scenes (Figure 4.8).

By definition, congruent tiles should have the same set of symmetries, possibly expressed in different coordinate systems. Since we know the symmetries of each of the tiles, we introduce this constraint, thereby increasing the discriminating power of our shape descriptor as shown in Table 4.10.



Room
39,557 Polygons

Plane
182,224 Polygons

Studio
515,977 Polygons

Figure 4.8: Scenes used for testing the object congruency descriptor. In each scene the descriptor has been used to detect objects with similar geometry (but possibly different meshes) up to a rigid transform. Objects found to be congruent are displayed with the same color.

Max order	39,557 Polygons 851 Tiles	182,224 Polygons 480 Tiles	515,977 Polygons 5,700 Tiles
2	92.1 %	43.9 %	92.3 %
4	100 %	78.0 %	100 %
6	100 %	92.2 %	100 %
8	100 %	100 %	100 %

Figure 4.9: Percentage of tiles matched by our shape descriptor that are effectively identical for our test scenes.

Max order	39,557 Polygons 851 Tiles	182,224 Polygons 480 Tiles	515,977 Polygons 5,700 Tiles
2	95.6 %	73.4 %	97 %
4	100 %	96.0 %	100 %
6	100 %	100 %	100 %
8	100 %	100 %	100 %

Figure 4.10: Percentage of tiles matched by our shape descriptor that are effectively identical using the added constraint that identical tiles must have the same set of symmetries up to a rigid transform.

4.5.3 Algorithm for assembled objects

Overview

Once we have determined all classes of congruent tiles, the algorithm examines all the one-to-one mappings of the set of all tiles onto itself, which map each tile onto a similar tile. For each one-to-one mapping found, it determines the isometric transforms which are simultaneously compatible with each tile and its symmetries.

The algorithm works recursively: at the beginning of each recursion step, we have extracted two subsets of tiles $\mathcal{H}_1$ and $\mathcal{H}_2$ of the composite shape $\mathcal{S}$, and we have computed the set of all possible isometric transforms that globally transform $\mathcal{H}_1$ into $\mathcal{H}_2$. Then, taking two new similar tiles $\mathcal{S}_1 \in \mathcal{S} \setminus \mathcal{H}_1$ and $\mathcal{S}_2 \in \mathcal{S} \setminus \mathcal{H}_2$, we restrict the set of isometric transforms to the isometric transforms that also map $\mathcal{S}_1$ onto $\mathcal{S}_2$ (but not necessarily $\mathcal{S}_2$ onto $\mathcal{S}_1$). Because these tiles have symmetries, this usually leaves multiple possibilities.

Note that the global symmetries found must always be applied with respect to the center of mass $\boldsymbol{g}$ of $\mathcal{S}$, according to the definition of a symmetry of $\mathcal{S}$.

At the end of the recursion step, we have the set of isometric transforms that map $\mathcal{H}_1 \cup \{\mathcal{S}_1\}$ onto $\mathcal{H}_2 \cup \{\mathcal{S}_2\}$.

Each recursion step narrows the choice of symmetries for $\mathcal{S}$. The recursion stops when either this set is reduced to identity transform, or when we have used all the component tiles in the model. In the latter case, the isometric transforms found are the symmetries of the composite shape. The recursion is initiated by taking for $\mathcal{H}_1$ and $\mathcal{H}_2$ two similar tiles, that is two tiles of the same class.

In the following paragraphs, we review the individual steps of the algorithm: finding all the isometric transforms which map tile $\mathcal{S}_1$ onto similar tile $\mathcal{S}_2$, and reducing the set of compatible symmetries of $\mathcal{S}$. We then illustrate the algorithm on a step-by-step example.

Finding all the isometries which transform a tile onto a similar tile

At each step of our algorithm, we examine pairs of similar tiles $\mathcal{S}_1$ and $\mathcal{S}_2$, and we have to find all the isometries which map $\mathcal{S}_1$ onto $\mathcal{S}_2$.

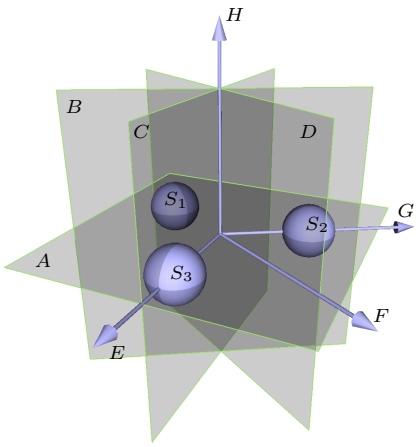
If $\boldsymbol{g}_i$ is the center of mass of tile $\mathcal{S}_i$ and $\boldsymbol{g}$ is the center of mass of the composite shape $\mathcal{S}$, this condition implies that the isometries we are looking for transform vector $\boldsymbol{g}_1 - \boldsymbol{g}$ into $\boldsymbol{g}_2 - \boldsymbol{g}$. In order to generate the set of all isometric transforms that map $\mathcal{S}_1$ onto $\mathcal{S}_2$, we use the following property:

Property 4 *: If J is an isometry that maps $\mathcal{S}_1$ onto a similar tile $\mathcal{S}_2$, then all the isometries K which map $\mathcal{S}_1$ onto $\mathcal{S}_2$ are of the following form:*

$$K = JT^{-1}AT \quad \textit{with} \quad A \in G_{\mathcal{S}_1} \quad \textit{such that} \quad A(\boldsymbol{g}_1 - \boldsymbol{g}) = \boldsymbol{g}_1 - \boldsymbol{g} \tag{4.11}$$

where $G_{\mathcal{S}_1}$ is the group of symmetries of $\mathcal{S}_1$ and T is the translation of vector $\boldsymbol{g} - \boldsymbol{g}_1$ (please refer to the Appendix for proof of this property).

This property states that once we know a single *seed* isometric transform which maps $\mathcal{S}_1$ onto $\mathcal{S}_2$, we can generate all such transforms by using the elements of $G_{\mathcal{S}_1}$ in Equation 4.11.



Mapping	Symmetry
$S_1 \to S_1$ $S_2 \to S_2$ $S_3 \to S_3$	mirror-symmetry A
$S_1 \to S_3$ $S_3 \to S_1$ $S_2 \to S_2$	mirror-symmetry B 2-fold symmetry G
$S_1 \to S_2$ $S_2 \to S_1$ $S_3 \to S_3$	mirror-symmetry C 2-fold symmetry E
$S_2 \to S_3$ $S_3 \to S_2$ $S_1 \to S_1$	mirror-symmetry D 2-fold symmetry F
$S_1 \to S_2$ $S_2 \to S_3$ $S_3 \to S_1$	3-fold symmetry H

Figure 4.11: Three spheres uniformly distributed on a circle in the z-plane. Etablishing all one-to-one mappings of the set of all tiles onto itself, which map each tile onto a similar tile, are used to detect all the symmetries of the shape. Note that the 3−fold symmetry H is detected and is associated to a circular permutation mapping.

Finding a seed transform

We need to find a seed transform J that maps $\mathcal{S}_1$ onto $\mathcal{S}_2$. For each tile, we extract a minimum set of independent vectors that correspond to extremas of their generalized even moment functions. The number of vectors needed depends on the symmetries of the tile. J is then defined as any isometric transform that maps the first set of vectors onto the second, as well as vector $g_1 - g$ onto $g_2 - g$. Most of the time, a single isometric transform is possible at most. When multiple choices exist the candidate transforms are checked onto the shapes using the distance presented in Section 4.4.3. This ensures that we find at least one seed transform.

Ensuring compatibility with previous isometries

During the recursion, we need to store the current set of compatible isometries we have found. We do this by storing a minimal set of linearly independent vectors along with their expected images by these isometries. For example, if we have to store a symmetry of revolution, we store only one vector, the axis of the symmetry, and its image (itself). For mirror symmetries, rotations and central symmetries, we store three independent vectors, along with their images by this isometric transform. For instance, in the case of a rotation of angle π around axis $\boldsymbol{X}$, we have:

$$\boldsymbol{X} \mapsto \boldsymbol{X} \quad \boldsymbol{Y} \mapsto -\boldsymbol{Y} \quad \boldsymbol{Z} \mapsto -\boldsymbol{Z} \tag{4.12}$$

By examining all the one-to-one mappings of the set of all tiles onto itself, which map each tile onto a similar tile, we are able to detect all symmetries of the set of tiles (See figure 4.11). Note on this example that the 3−fold symmetry H is detected and is associated to a circular permutation mapping.

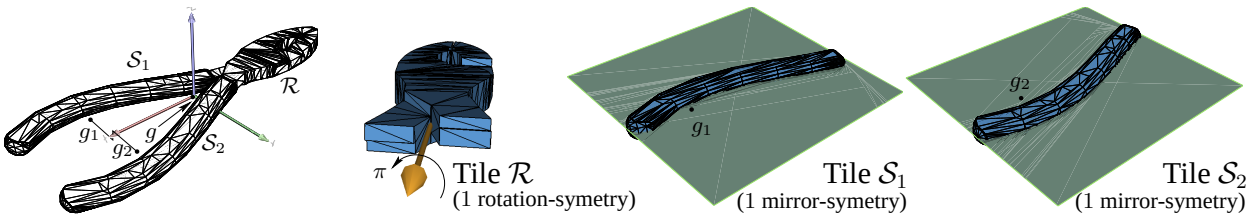


Figure 4.12: Illustration of the constructive algorithm on a very simple example: from the symmetries of each of the 3 parts of the object, the symmetries of the whole object are recovered. Please note that no symmetry was ommitted in this figure, in particular tile $\mathcal{R}$ only has a rotational symmetry but no mirror symmetry. See text of Section 4.5.4 for a detailed explanation.

4.5.4 Step-by-step example

Figure 4.12 presents a very simple example of a shape (a pair of pliers) composed of 3 tiles $\mathcal{S}_1, \mathcal{S}_2$ (the handles) and $\mathcal{R}$ (the head). Two of the tiles are similar up to an isometric transform: $\mathcal{S}_1$ and $\mathcal{S}_2$. Figure 4.12 also displays the centers of mass $\boldsymbol{g}_1$ and $\boldsymbol{g}_2$ of tiles $\mathcal{S}_1$ and $\mathcal{S}_2$ (which are not in the plane $z = 0$) and the center of mass $\boldsymbol{g}$ of the whole shape. In the coordinate systems centered on their respective centers of mass, $\mathcal{S}_1$ and $\mathcal{S}_2$ have a mirror-symmetry of axis $\boldsymbol{Z}$ and $\mathcal{R}$ has a rotation-symmetry around axis $\boldsymbol{X}$ of angle π.

Our constructive algorithm starts by selecting tile $\mathcal{R}$ and a similar tile (here, the only possible choice is $\mathcal{R}$).
Step 1: the algorithm explores the possibilities to transform $\mathcal{R}$ into itself. Two possibilities exist: (a) the identity transform, and (b) the rotation around $\boldsymbol{X}$ of angle π, deduced from (a) by property 4.
At this point, the algorithm branches, and either tries to map $\mathcal{S}_1$ to itself (branch 1) or to $\mathcal{S}_2$ (branch 2).
Branch 1, step 1: the algorithm tries to match $\mathcal{S}_1$ to itself. The only compatible transform is the identity transform.
Branch 1, step 2: The algorithm then tries to map $\mathcal{S}_2$ to itself. Once again, the only possible transform is the identity transform, and the recursion stops because all the tiles in the model have been used.
Branch 2, step 1: the algorithm tries to match $\mathcal{S}_1$ to $\mathcal{S}_2$. The only compatible transform is the rotation around $\boldsymbol{X}$ of angle π.
Branch 2, step 2: the algorithm then tries to match $\mathcal{S}_2$ to $\mathcal{S}_1$. Once again, the only compatible transform is the rotation around $\boldsymbol{X}$ of angle π, and the recursion stops because all the tiles in the model have been used.

Two symmetries have been found that map the shape onto itself: the identity transform and the rotation around $\boldsymbol{X}$ of angle π. Note that although our algorithm can potentially create lots of branching, we prune branches that result in empty sets of transforms and in practice, we only explore a small number of branches.

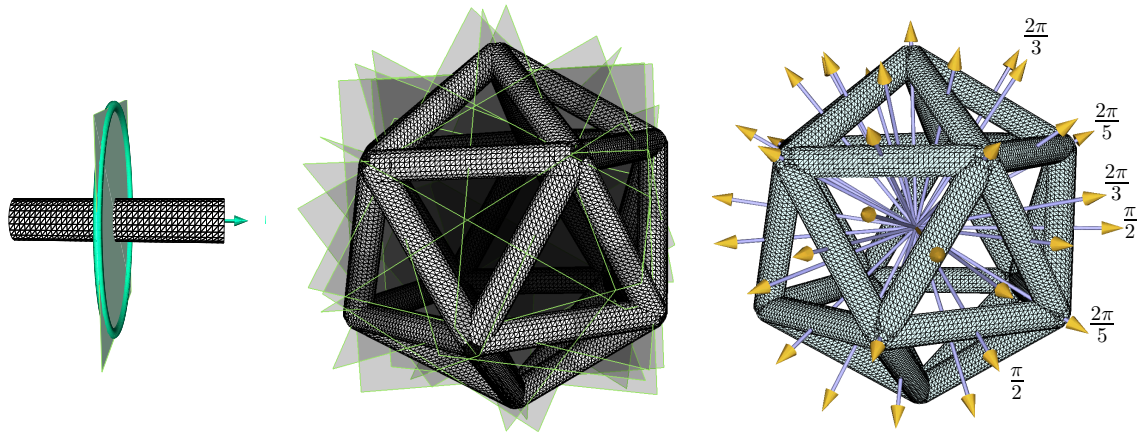


Figure 4.13: A complex model which has the same group of symmetries than the icosahedron. The constructive algorithm successfully retrieves all 15 planes of mirror symmetries (*center*) and all 31 distinct axes of rotational-symmetries (*right*) using the rotational and mirror symmetry of each tile (at *left*). The presence of 3−fold and 5−fold symmetries proves that our algorithm also detects symmetries which map a set of similar tiles onto itself through a complex permutation.

4.5.5 Application scenarios

In order to illustrate the efficiency of the constructive algorithm, we show in this section various situations where this method is a valuable alternative to the single-shape algorithm.

Application to an agregation of many objects

Figure 4.13 presents a complex model which has the same group of symmetries as an icosahedron. The constructive algorithm retrieves all the 31 distinct axis of rotational-symmetries (Figure 4.13, *right*) as well as the 15 axis of plannar symmetries (Figure 4.13, *middle*) of the shape, using the symmetries of each tile (Figure 4.13, *left*), which are 1 revolution-symmetry and 1 mirror-symmetry.

Conversely, directly applying the first algorithm on such a shape shows that $\mathcal{M}^2$ to $\mathcal{M}^8$ are extremely close to constant functions, making the extraction of directions an inaccurate process. The single-shape algorithm still correctly finds all the axis if using moments up to order 10, but this has some impact on computation times. Furthermore, the single-shape algorithm needs checking all symmetries found on the model – which is a significant part of its computation time. This is not the case for the constructive algorithm because it relies on its knowledge of the symmetries of the tiles only. Because many symmetries exist for this model, the total computation time of the single-shape algorithm is therefore much higher. This is summarized in Table 4.14, where we compare the computation times for both methods at equivalent precision (i.e. 10^{-4} radians).

Method	Single shape (order 10)	Constructive (order 4)
Moments calculation	500 sec	30×0.5 sec
Symmetry verification	46×55 sec	$30 \times 2 \times 1.5$ sec
Tile congruency	N/A	2 sec
Tile mappings	N/A	10 sec
Total	**50mn 30 sec**	**1mn 57 sec**

Figure 4.14: Comparison of the costs of the single-shape algorithm presented in Section 4.4 to the cost of the constructive algorithm to find all 46 symmetries of the icosahedron shape displayed on Figure 4.13 at equivalent precision. Because the object is close to a sphere and because it has many symmetries, the constructive algorithm performs much better.

Finding symmetries inside non coherent geometry

There exist common situations where 3D scenes do not come as a set of closed separate objects, but as an incoherent list of polygons. This happens for instance when retrieving geometric data from a web site, mostly because a list of polygons constitutes a practical common denominator to all possible formats.

In such a case, applying the single-shape algorithm would certainly give the symmetries of the whole scene, but if we are able to partition the set of polygons into adequate groups – i.e. tiles – to which we apply the constructive algorithm we may be able to extract symmetric objects from the scene, as well as the set of symmetries for the whole scene more rapidely, as illustrated in Figure 4.13.

The gain in using the constructive algorithm to recover symmetries in the scene resides in the fact that, once tile symmetries have been computed, grouping them together and testing for symmetries in composed objects only adds a negligible cost, which is not the case when we try to apply the single shape algorithm to many possible groups of polygons or even to the entire scene itself.

Figure 4.15 gives examples of such tiles for objects collected from the web as a raw list of polygons.

Our simple heuristic approach of making tiles produced very good results on all scenes we tested and suffices for a proof of concept of the constructive algorithm. This is illustrated in Figure 4.15, where a lamp object and a chess game are shown along with their global symmetries. These symmetries were computed from the symmetries of each of the sub-parts. These in turn were separately computed using the algorithm presented in Section 4.4.

Obviously this application needs that constructed tiles be consistent with symmetries, i.e. that it is possible to partition the scene into tiles which will map onto each other through the symmetries of the scene. This may not be easy with scanned models for instance nor in perturbated data. In such a case, our simple heuristic should be modified so as to base polygon neighborood relationships on proximity distances between polygons rather than vertex positions only. Doing so, cutting one tile into two parts and remeshing them independently would have a high probability of producing the same original tile after reconstruction. If not, then the existance of a symmetry inside the model may become questionnable: suppose for instance that the wrench in the step by step example (Section 4.5.4) gets split into tiles that are not exact symmetrical copies of one another, and that these two tiles are too far away to

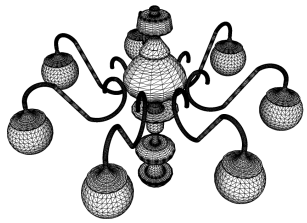
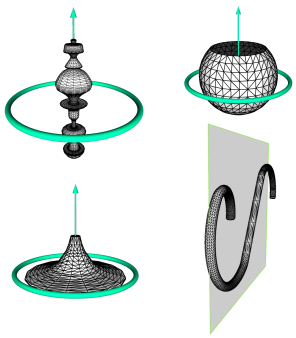
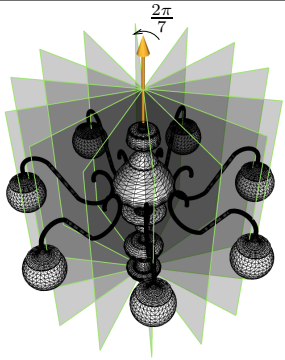
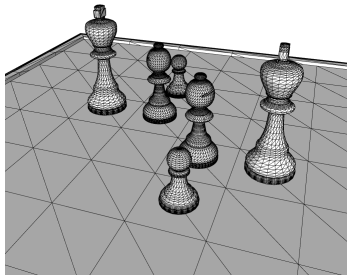
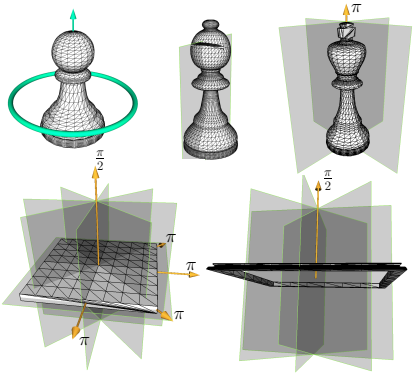
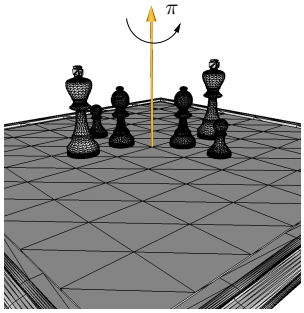
Original model	Tiles extracted using heuristic of Section 4.5.5 and their symmetries, computed with the single-shape algorithm	Global symmetries of the model
		
		

Figure 4.15: Two models taken from the web. From the raw list of polygons (*left*) our heuristic for scene partitionning extracts tiles before the single-shape algorithm computes the symmetries for each of them (*center*). Using this information, the constructive algorithm computes the symmetries of the whole model (*right*). *Top row*: A lamp object which has seven mirror-symmetries and a $7-$fold rotational-symmetry. *Bottom row*: a chess board which is composed of pieces with very different symmetries but reveals to only have a single $2-$fold symmetry around a vertical axis (Note: in this last model, once tiles have been identified, chess pieces were moved so as to obtain a model with at least one global symmetry).

Model	Plier	Lamp	Chessboard
# polygons	1,940	39,550	24,942
# tiles	3	22	8
Computing moments*	0.9	18.2	15
Finding parameters	0.4	1.2	2.0
Checking candidates	2.3	7.4	7.9
Constructive algo.	0.001	0.05	0.01
Total	3.601	26.85	24.91

Figure 4.16: Computation times in seconds for the different steps of our algorithm, for the models shown in this paper.**Global computation time for moments of order 2 to 8

be merged into a single tile. Then the model is by nature not symmetric anymore, which will also be the output of the constructive algorithm.

4.5.6 Computation cost

Computation times (in seconds) for the models shown in this paper are given in Table 4.16, as well as the complexity of the models. They were measured on a machine equipped with a 2.4 GHz processor with 512 MB of memory. As expected, the cost of the computation of the moment functions and the cost of the verification of the candidates required by the first algorithm occupy the most important part of the total cost, and depend on the model complexity. Conversely, finding the parameters of the symmetries (Section 4.4.2) as well as applying the constructive algorithm only depends on the number of these symmetries.

Regarding accuracy, both algorithms computed the axes of the symmetries with a maximum error of 10^{-4} radians, independently of shape complexity, in our tests.

5

Results And Applications

5.1 Geometry compression and instantiation

Our framework can be used for model compression at two different levels: (1) If a model exhibits symmetries, then it can be compressed by storing only the significant part of the model, and using the symmetries to recreate the full model. (2) if a model contains multiple instances of the same part, then these parts can be instantiated.

Although complex models often do not present symmetries, symmetry-based compression can usually be used on some sub-parts of the model. The ability to express a model by explicitely storing the significant parts only while instancing the rest of the scene is provided by some recent 3D file formats such as X3D (See Table 5.2 on the next page). We thus measure our compression ratios as the size of the X3D files before and after our two compression operations which we detail now:

The scene is first loaded as a raw collection of polygons, before being decomposed into tiles, using the heuristic presented in Section 4.5.5. We then compute symmetries and congruent descriptors for each tile. Computation times shown on Table 5.2 present the average time needed to compute symmetries and congruent descriptors for a single tile. As the process of computing tile properties does not depend on the other tiles, it is an easily parallelizable process. The scene is then first compressed by instancing the tiles. Secondly, when storing each tile, we only store the minimum significant part of its geometry according to its symmetries. This part is extracted using the same algorithm we will present for remeshing a tile according to its symmetries in the next section. Note that compression rates shown on this table are computed using geometry information only, i.e. neither texturing nor material information are taken into account. Compression times shown in Table 5.2 on the following page are the times needed to detect all classes of tile congruency.

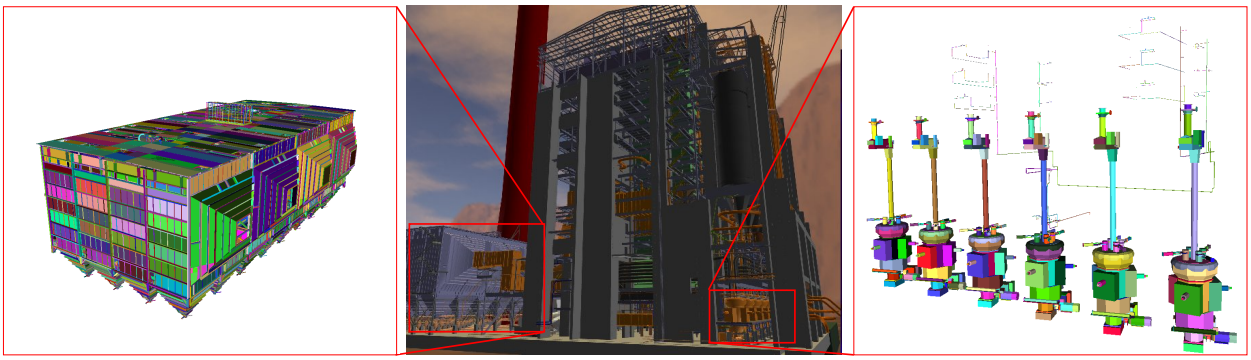


Figure 5.1: Detecting symmetries and similarities between tiles created from a raw list of polygons, allows us to compress geometric models in two ways: (1) by instancing similar tiles and (2) inside each symmetric tile, by instancing the part of the geometry which permits to reconstruct the whole tile. In such a big model as the powerplant (13 millions triangles) we achieve an compression ratio (ratio of geometry file size in X3D format) of $1:4.5$. We show on this figure two sub-parts of the complete model. For each, we show the tiles computed by our heuristic (See Section 4.5.5) as well as the obtained compression ratio. The PowerPlant model is a courtesy of The Walkthru Project.

Model	Room	Plane	Studio	Powerplant
# polygons	$39,557$	$182,224$	$515,977$	$12,748,510$
# tiles	851	480	$5,700$	$155,348$
av. computing tile properties (secs)	1.45	1.3	1.9	1.1
Compression time (secs)	7.2	9	14.6	311
Compression rate	$1:2.7$	$1:8.3$	$1:3.5$	$1:4.5$

Figure 5.2: Examples of compression rates obtained using our symmetry detection method coupled with the congruency descriptor. See text in Section 5.1 for a detailed explanation.

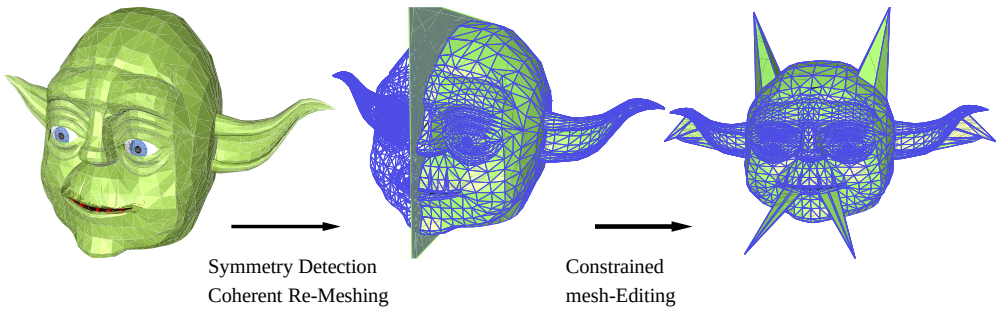


Figure 5.3: Starting from an object in arbitrary orientation, we detect symmetries of the shape (in the figure, a planar symmetry) and use it to remesh the objects with respect to these symmetries. Then, a user can easily edit the mesh and modify it while keeping the symmetries of the initial shape.

5.2 Mesh Editing

It may be interesting, when an object contains symmetries, to remesh the object with respect to these symmetries. In order to do this, we proceed by first extracting the minimum part of the shape that can be reconstructed through each symmetry independently; then we apply the corresponding symmetry to each of them in order to get as many meshes of the shape which are consistent with each symmetry independently. The final step is to compute the union of all these meshes, merging identical vertices and adding new vertices at edge crossings. While not necessarily optimal, the obtained mesh is consistent with all symmetries of the shape.

Since a coherent remeshing allows to establish a correspondence between model vertices, we have developed a proof-of-concept mesh editing system which allows the user to modify a 3D object under the constraints given by the symmetries of the original object. It appears that, under the constraint of too many symmetries, no vertices can be moved independly of the others and the geometry is sometimes bound to scale about its center of gravity. Images collected from this program are displayed in Figure 5.3.

5.3 Isotropic Models

As stated earlier, symmetry detection methods based on eigenvectors of the covariance matrix usually failed for two reasons. The first one is when two eigenvalues are strictly or almost equivalent. In this case, only one axis can be detect in a robust way, which result in a indetermination when looking for axis of symmetry. This is illustrated in Figure 5.4 on the next page.

The second reason that make PCA-based symmetry detection is when the three eigenvalues of the model are strictly or almost equivalent. In this case, no axis can be derived and hence no symmetry can be obtained. In this section, we focus on the latter case by first defining what an isotropic model is and presenting some almost isotropic model.[1]

[1] In our work, this simply mean that the $\mathcal{M}^{2p}$ moment of an isotropic model will be constant for $p = 1$.

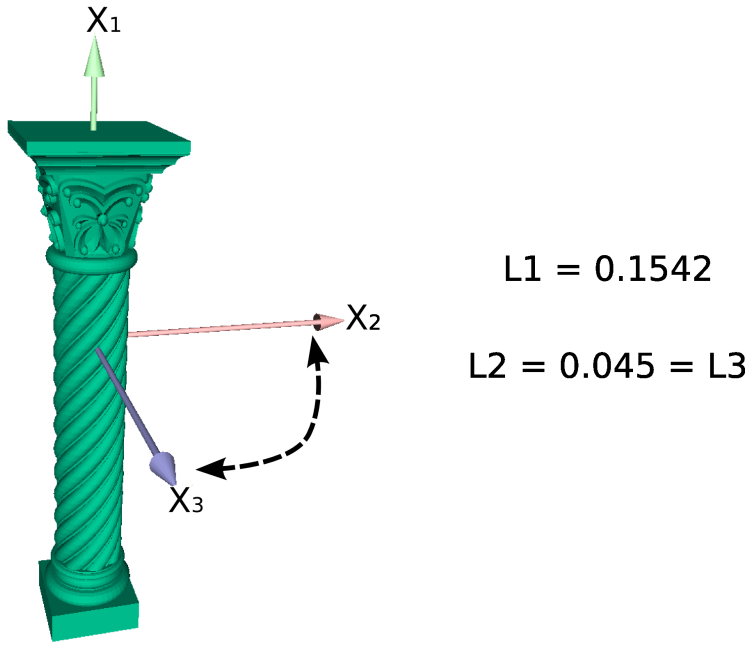


Figure 5.4: Starting from an object in arbitrary orientation, we detect symmetries of the shape (in the figure, a planar symmetry) and use it to remesh the objects with respect to these symmetries. Then, a user can easily edit the mesh and modify it while keeping the symmetries of the initial shape.

Definition 10 *A geometric model is called isotropic if its variance is independent of the direction.*

The most natural form that is known to be isotropic is the cube; from a variance point of view, a cube is equivalent to a sphere. However, it is easily to find model that are also isotropic as those presented on Figure 5.5 on the facing page.

Intuitively, method based on principal axis will have more difficulty to compute robust symmetry axis when the model is almost isotropic. To estimate "the distance" from an object to its isotropic version, we use the method proposed by [Kazhdan et al., 2004a]. In this paper, authors proposed a way of removing anisotropy of a model by rescaling it based on the value of the covariance matrix of the model.

More specifically, it can be proved that, for a point set with covariance matrix C, it suffices to rescale the point set by the matrix $C^{-1/2}$ to obtain an isotropic point set. If $P = p_1, p_2, ..., p_n$ is a point set, its covariance matrix is :

$$C_p = \sum_{i,j=1}^{n} (p_i - p_j).(p_i - p_j)^t$$

When rescaling the point set P by the matrix $C_p^{-1/2}$, we obtain a new point set that can be expressed as $Q = \{C_p^{-1/2}p_1, C_p^{-1/2}p_2, ...C_p^{-1/2}p_n)\}$ and the associated covariance matrix is now :

$$C_q = \sum_{i,j=1}^{n} C_p^{-1/2}(p_i - p_j).(p_i - p_j)^t C_p^{-1/2}$$

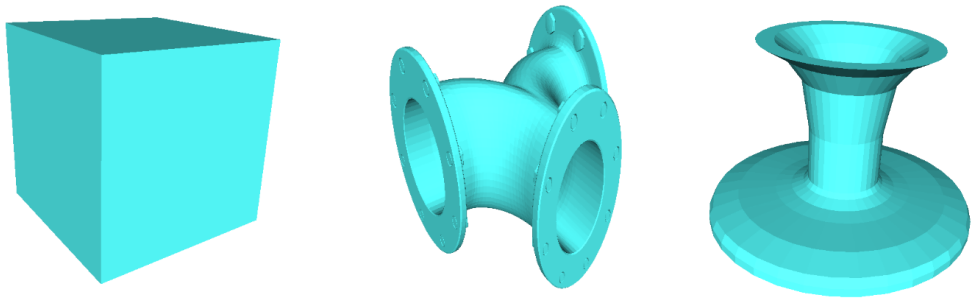


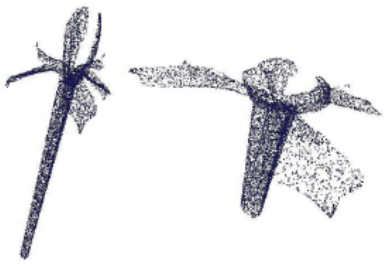
Figure 5.5: A set of isotropic models, i.e. models for which the variance is independent of the direction. The most well-known isotropic object is the cube, but some other object have also this property. In this case, applying a method based on principal axes will failed as no axes can be uniquely defined for testing symmetry.

which transform into :

$$C_q = C_p^{-1/2}(\sum_{i,j=1}^{n}(p_i - p_j).(p_i - p_j)^t)C_p^{-1/2} = 1$$

The point set Q is now isotropic has its variance is independent of the direction.

To apply it on 3D surfaces, authors proposed an iterative version of this method to take into account sampling method (an uniform sampling of the surface, once anisotropically rescaled is not uniform anymore. On right figure, uniform sampling of the iris is shown on the left. After isotropic rescaling, the point set no longer represent uniform sampling of the surface.



6

Summary and Discussion

6.1 Summary

We have presented an algorithm to automatically retrieve symmetries for geometric shapes and models. Our algorithm efficiently and accurately retrieves all symmetries from a given model, independently from its tesselation.

We use a new tool, the *generalized moment* functions, to identify candidates for symmetries. The validity of each candidate is checked against the original shape using a geometric measure. Generalized moments are not computed directly: instead, we compute their spherical harmonic coefficients using an integral expression. Having an analytical expression for the generalized moment functions and their gradients, our algorithm finds potential symmetry axes quickly and with good accuracy.

Compared to recent method that search both local and global symmetries, we restrict our search to global symmetry,i.e, symmetry applied to the center of mass of the objects. For composite shapes assembled from simpler elements, we have presented an extension of this algorithm that works by first identifying the symmetries of each element, then sets of congruent elements. We then use this information to iteratively build the symmetries of the composite shape. This extension is able to handle complex shapes with better accuracy, since it pushes the accuracy issues down to the scale of the tiles.

In the following, we discuss here a number of features of our technique, as well as differences with existing approaches.

6.2 Discussion

Using spherical harmonics

Generalized moments are a central component of our system. As stated before, we do not compute these functions explicitly, but we rather compute their coefficients in a spherical harmonics basis. As for the decomposition itself, any basis could be used. In particular, a well chosen basis of 3D monomials restricted to the unit sphere may also lead to a finite decomposition. Still, using spherical harmonics has many advantages, in particular because we use the same coefficients computed once for different tasks throughout this paper: (1) The expression of moment function as a sum of spherical harmonics provides an accurate detection of the potential axes of symmetries. This detection is made deterministic by finding the zero-directions for the gradient of the moment functions. Such a computation is performed analytically from the 2nd order derivatives of the spherical harmonics, and thus does not introduce further approximation. (2) Computing symmetry parameters for the moment functions is made very easy by working on the spherical harmonic coefficients themselves. Spherical harmonics being orthogonal and easily rotated, finding symmetries on the moment functions translates into simple relationships between the coefficients. (3) The spherical harmonic coefficients provide an effective shape congruency descriptor, which we use to detect which tiles are identical up to an unknown isometric transform.

In summary, the use of spherical harmonics provides us a consistent framework throughout the whole process of our symmetry-finding algorithm.

Non star-shaped objects

Whether the direct algorithm presented in Section 4.4 works for non star-shaped objects is a legitimate question. Our approach never relies on a spherical projection. Indeed, the moment functions, as expressed in equations 4.1 and 4.5 are computed through an integration over the surface itself, possibly covering the same directions multiple times but with different values. Parts of a shape which correspond to a same direction during integration will not contribute the same into the various moment functions because of the varying exponent. By using various orders of moment functions in our symmetry detection process and in the computation of our shape congruency descriptor, we thus capture the geometry of non star-shaped objects as well. Some previous approaches [Kazhdan et al., 2004b] achieved this by decomposing the shape into concentric spherical regions before doing a spherical integration, which can be assimilated to "convoluting" the shape with $0-$degree functions with concentric spherical support; Our technique is similar, but with an other kind of functions expressed into the form of the even moments. In summary, detecting symmetries on non-star-shaped objects has no particular reason to fail, which is illustrated by the result in Figure 4.1.

The second algorithm (for assembled objects) naturally works just as well for non-star-shaped objects, as illustrated by the examples in Figure 4.15.

Avoiding dense sampling

Previous methods that defined a continuous measure of symmetry([Zabrodsky et al., 1995; Kazhdan et al., 2004b]) can theoretically compute both perfect and approximate symmetries.

However, detecting symmetries using such methods involves a sampling step of the directions on the sphere, whose density must be adapted to the desired angular precision for the axis of the symmetry.

The work of Kazhdan et al. [Kazhdan et al., 2004b] leads to impressive results concerning the improvment on the shape matching process. However, relying on this technique to obtain accurate symmetries with high angular precision requires a time-consuming step for the construction of the symmetry descriptors. According to the presented results, the time needed to compute reflective, 2-fold, 3-fold, 4-fold, 5-fold and axial symmetry information for a spherical function of bandwidth $b = 16$ is 0.59 seconds. As stated in the paper [Kazhdan et al., 2004b], the number of samples taken on the sphere is $O(b^2)$ (*i.e* approximately 10^3 sample directions) which is insufficent to reach a high angular precision equivalent to the one obtained with our method: reaching a precision of 10^{-4} radians would approximately require 10^9 sample directions. This would theoretically increase the computation time to approximately $0.59 \times 10^9/10^3 = 5.9\ 10^5$ seconds, making the method inefficient for this task.

In contrast, our method does not rely on a dense sampling of directions to find symmetries, but on the computation of a fixed number of surface integrals which – thanks to the Gauss integration used – provides an extremely accurate approximation of the spherical harmonic coefficients of the moment functions. From there on, no further approximation is introduced in the computation of the directions of the candidate symmetries, which lets us achieve an excellent angular precision at a much lower cost.

Furthermore, the cost of our algorithm does not rely on assumptions about the expected results. The method of [Kazhdan et al., 2004b] indeed computes symmetry descriptors for each kind of searched symmetry. Our method in turn computes all directions of possible symmetries and then checks back on the shape the obtained candidates.

Part III

Hierarchical Instancing of Geometry

7
Instancing and Computer Graphics

In this chapter, we focus on Instancing Information *i.e.* the information of what is instantiated in the scene, possibly at multiple scales. We first demonstrate the utility of such information by introducing methods that can used such information. As presented above, Instancing Informationis often lost or even not present in the input geometric data, depending on the way the data have been modeled or transformed. Our work so focuses on *recovering Instancing Information* and we present, in the second part of this chapter, the previous work related to this task. We then conclude this chapter by presenting an overview of our method developed to compute a hierarchy of instances, starting from unstructured geometry.

7.1 Importance of Instancing Information

Three-dimensional environments, man-made or natural, always exhibit some structure at different scales. This manifests itself by repetition of parts, geometric similarity, and hierarchical relationships. The development of computer graphics and geometric algorithms has amply and repeatedly demonstrated the huge benefits that can be drawn from exploiting this structure, be it for raw performance, convenience of edition and manipulation, or analysis.

In the computer graphics field, instancing is an approved concept for reducing the size of the file, helping save memory, increasing rendering times, and increasing the data transfer rates. In the following, we detailed each of these points.

7.1.1 Compression

3D compression is a branch of data compression aimed at the 3D models and other geometric datasets used in computer graphics, virtual reality, video games, CAD/CAM, and many scientific, engineering, and medical applications. In this section, we only focus of compression using Instancing Information.

With the well-known open formats VRML and X3D, compression with instancing is realized with the DEF/USE/PROTO usage. More precisely, the DEF node is used to named a node that may be referenced later by the USE node. A PROTO definition creates a new node type that can be used anywhere in the rest of the same world file. An example of a small virtual world created with and without instancing in X3D is depicted in Figure 7.1

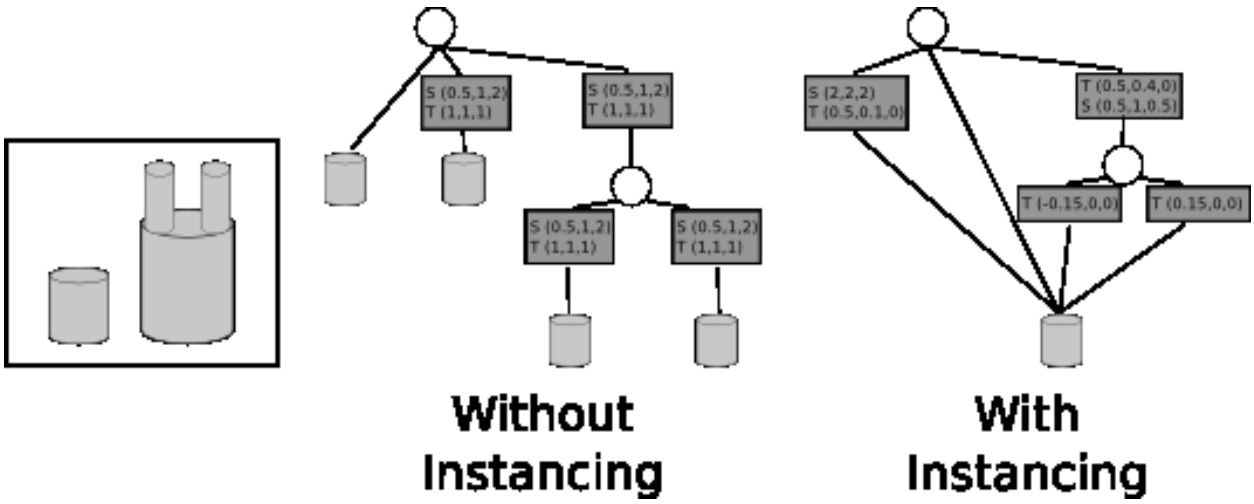


Figure 7.1: An example of a simple 3D scene modeled using instancing capability of X3D.

The compressed scene graph on the right of the above figure can be represented in X3D in the following way :

```
<?xml version="1.0" encoding="UTF-8"?>
<X3D>
  <Scene>
   <Shape  DEF="cylinder">
    <Cylinder height="0.2" radius="0.1"/>
   </Shape>

   <Transform translation="0.5 0.1 0" scale="2 2 2">
    <Shape USE="cylinder"/>
   </Transform>

   <Transform translation="0.5 0.4 0" scale="0.5 1 0.5">
    <Transform translation="-0.15 0 0">
      <Shape USE="cylinder"/>
    </Transform>
    <Transform translation="0.15 0 0">
      <Shape USE="cylinder"/>
    </Transform>
   </Transform>
  </Scene>
</X3D>
```

The advantage of compression is obviously increase if the geometry of the object to instantiate is complex. Moreover, producing such a file naturally supposes to know the instancing relationships between objects.

7.1.2 Rendering

Ray-Tracing

Ray-tracing is a rendering technique that naturally allows instancing. Complex models can be ray traced with little memory since only the geometric representations of original objects must be kept in memory [Snyder and Barr, 1987; Kay and Kajiya, 1986]. Thanks to instancing, all ray intersections are computed against the unique geometrical model common to all instances.

In a formal way, let's O be an object and M a transformation. To determine the intersection q of a ray r with an instance MO, we first compute the intersection p of the inverse transformed ray $M^{-1}r$ and the original object O. The searched point q is then simply Mp. With the gain in storage cost that instancing can bring (which is not a special feature of ray tracing), the main advantage to use instancing is that the untransformed object may have a simpler intersection routine, *e.g.* a sphere versus an ellipsoid.

One of the first used of instancing conjugated with ray-tracing is the work John Amanatides, presented in Figure 7.3 on the following page [Mitchell and Amanatides, 1989].

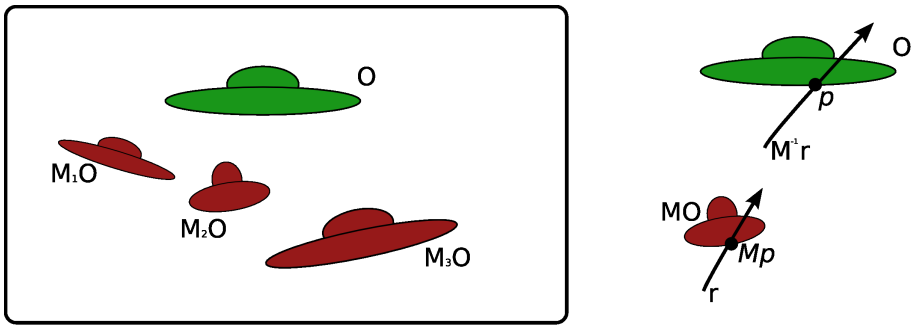


Figure 7.2: left) A simple scene with a single object, instantiated four times. Right) An example of determining intersection between a ray and an object. See the explanation below.

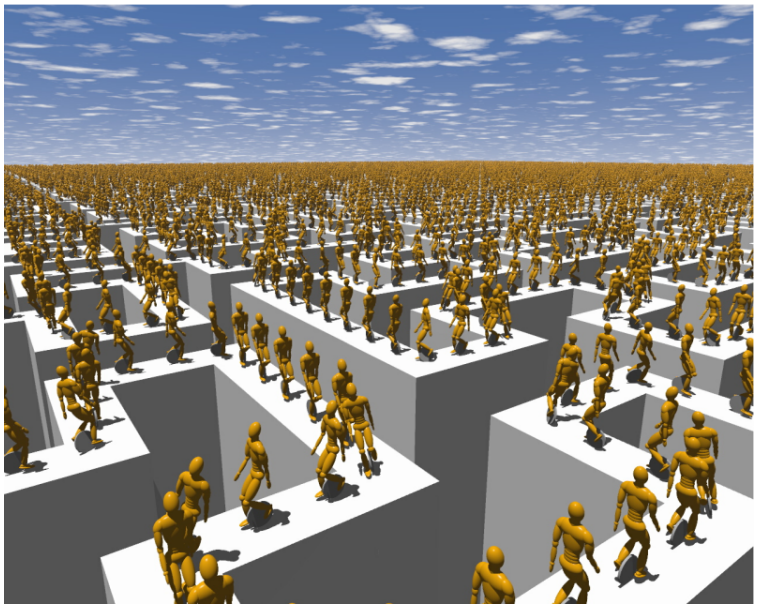


Figure 7.3: Image courtesy of John Amanatides. This image is part of an animation, presented at the Siggraph Electronic Theater, that demonstrates the benefits of instancing for ray-tracing.

Radiosity

Radiosity [Goral et al., 1984] is a global illumination rendering algorithm which has its basis in the theory of thermal radiation, since it relies on computing the amount of light energy transferred between two surfaces.

Computing the radiosity solution has proved to become very costly even for very simple configurations because of its inherent quadratic cost. This has stimulated the development of hierarchical approaches. The idea of hierarchical radiosity is to hierarchically group together surface elements and scene objects into *clusters* as depicted in Figure 7.4.

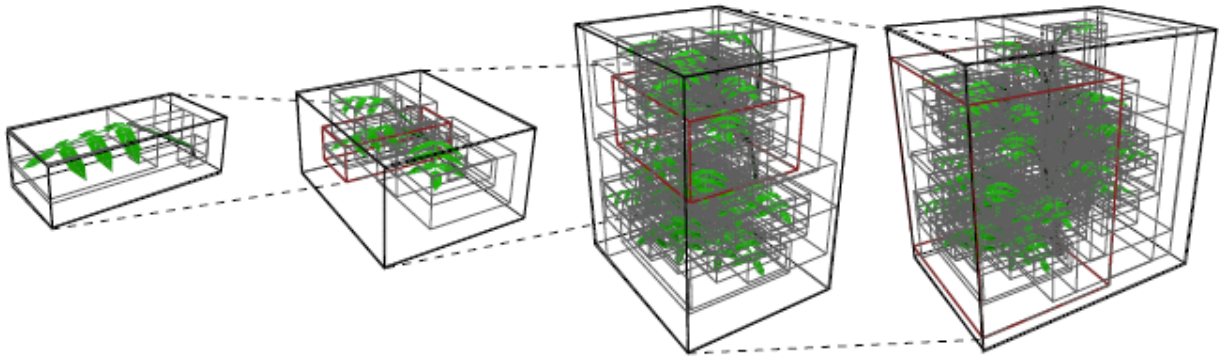


Figure 7.4: Hierarchy of clusters computed on a small plant.

The energy exchanges between all pairs of surface elements in the scene can then be factored (and approximated) by energy exchanges between clusters thus making the economy of a lot of computation [Sillion and Puech, 1994]. Approximations made by hierarchical radiosity algorithms are very sensitive to the quality of the cluster hierarchy as pointed out by [Hasenfratz et al., 1999].

Such methods of computing radiosity are however limited in the size of the scene because of the memory needed to compute the hierarchy. As instancing is an efficient way of reducing memory cost in rendering algorithm such as ray-tracing, this has inspired definition of hierarchical radiosity with instancing [Soler and Sillion, 2000; Soler et al., 2003].

Applying the principle of instancing to radiosity calculations is not strait-forward because the geometry is used in such methods as a support for the distribution of light energy, which in turn depends on the position of the object in the scene. As the radiometric information like reflectance or transmittance functions can somehow be easily shared between instances, the authors introduced two functions, the *generalized BRDF* and the *transmittance function* of an object to efficiently compute radiosity results with hierarchical instancing. A result of their technique is presented in Figure 7.5 on the following page.

Instancing with the GPU

The growing possibilities offered by the GPU have recently made possible *Hardware-based geometry instancing*. This method improves the potential runtime of rendering instanced geometry by explicitly allowing multiple copies of a mesh to be rendered sequentially by specifying the differentiating parameters for each in a separate stream. The different parameters

Figure 7.5: Impact of the hierarchical instancing when rendering a plant model. Self-similarity present in plants model allow them to be represented with few resources thanks to instancing.

are usually either color or transformation matrix. Such a technique is very useful for world clutter such as rocks, bushes, trees, forests, crates and debris (see Figure 7.6). In some genres of games for example, it can also be useful for weapons and armor.

Figure 7.6: Example of using hardware-based geometry instancing. Vegetation objects of the same color are submitted for rendering with only *one* draw call.

We now briefly review the way of using hardware-based instancing for both OpenGL and DirectX API.

Instancing using OpenGL. OpenGL was the first API to support hardware-based instancing through a technique called *pseudo-instancing*. This technique is qualified of *pseudo* as the number of draw calls is equal to the number of instances that need to be rendered. The effi-

ciency of this technique is to pass the per-instance world transform using texture coordinates glMultiTexCoord() instead of calls like glUniform4fcARB(). This exploits the software and hardware advantages of the persistent vertex attributes like texture coordinates.

The limitations of this technique are two-fold. First, it is not "real" instancing as the number of draw calls is equal to the number of instances that need to be rendered. Secondly, this technique only apply to geometry with a small number of per-instance attributes as it is limited to the number of vertex attributes.

Real instancing of geometry, *i.e.* the ability to draw multiple instances of a single draw call is available in OpenGL through the NVX_instanced_arrays. At the time of writing this dissertation, "real" instancing of geometry is however still in its experimental version.

Instancing using DirectX. The DirectX API supports instancing since 9.0c version. As stated above, the main idea is to use multiple streams :

- A primary stream which is a single copy of the model data,
- and a secondary stream that contains per instance data.

Let's take an example to illustrate this principle. We want to render 50 instances of a 100−vertex tree. The stream0 will contains just the one tree model. The stream1 contains model world transform (see Figure 7.7).

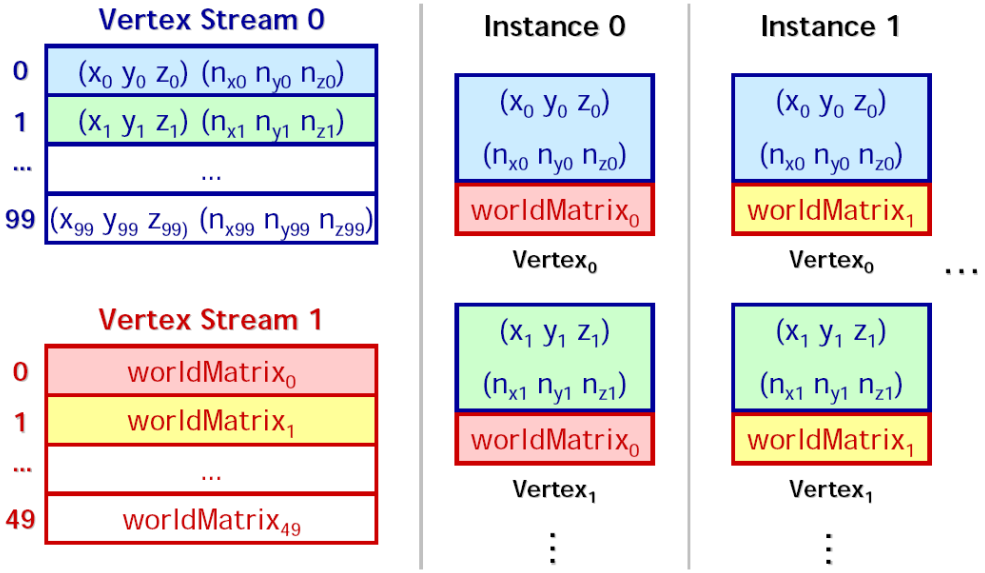


Figure 7.7: Hardware-based instancing with DirectX used two vertex streams. The stream0 contains one geometrical definition of the model and stream1 contains per instance data, in this case world transform. (Image extracted from [Cebenoyan, 2005])

To render the data, the used vertex shader can be the same as usual but must use the matrix from the vertex stream instead of the matrix from the vertex shader constants. We

refer the interested readers to [Carucci, 2005] for implementation details of instancing with directX.

More generally, the performance of hardware-based instancing is dependent on number of parameters such as the CPU frequency and the size of the batch[1] that are sent to the GPU. A detailed explanation of this can be found in [Cebenoyan, 2005].

7.1.3 Scene Editing

Another notable advantage of instancing is the fast update of geometry. A modification of parameters (ex : position) or information (ex : textures) in the generic object is simultaneously reflected in all its instances in the scene. An example of scene editing using Instancing Information is presented in Figure 7.8 on the next page.

7.2 Previous Work

Yet very useful for a broad range of algorithms, the literature about recovering instancing information is not very important. We review below the whole set of methods that focus on computing instancing information.

7.2.1 Instancing and Linear Fractal Modeling

The two most common linear fractal models are the recurrent iterated function system (RIFS) and the $L-$system. Such models are powerful for modeling natural scenes, but are difficult to render efficiently in current computer graphics system. For example, the geometry produced by a $L-$system is not organized efficiently for rendering. This has inspired the work of [Hart, 1992] which proposed a method to convert those two linear fractal models to hierarchical instancing structure.

7.2.2 Approximate Instancing

In most natural scenes, the number of *exact* instantiated objects is quite limited due the strict definition of instancing. This assumption has motivated the development of *Approximate Instancing* which primary goal is to increase the degree of instancing of such scenes.

This way of instancing objects has first been investigated by [Brownbill, 1996] who focus on approximate instancing for plant models. The main idea of approximate instancing is to consider two objects as instances if they resemble each other while being slightly different. In this work, the authors analyzed the trade-off between the size of the geometric model and their perceived visual distortion. Using this technique, dramatic reduction of the size of the plant database can be achieved ranging from $5:1$ to $50:1$ with a negligible visual impact on the generated images.

This technique has been used for generating complete landscape or complete plant ecosystems, such as the methods of [Deussen et al., 1998, 2002] (see Figure 7.9 on page 98).

[1] A batch is a set of polygons

(a) (b)

(c) (d)

Figure 7.8: An example of interactive editing at different level using instancing information collected by our method. a) The *Studio* model which contains many similar objects. b) A close-view of the kitchen. c) In one operation, designer of the scene can interactively modify the aspect of an object (the chair in the example) and automatically apply these changes to each instance found in the scene. d) At an upper level of the hierarchy, designer can now work on the set formed of a table plus two chairs and modify it.

Figure 7.9: A landscape obtained with the technique presented in [Deussen et al., 2002]. The compression rate obtained with instancing is 6.01:1.

More generally, approximate instancing is an effective technique for improve rendering but seems limited to a specific range of data. The main limitation of such methods is the ability to define a "proximity" between objects in order to cluster together objects that resemble each other but are not exactly the same. A promising technique to achieve such goal might be the use of shape descriptors whose primary goal is to translate the problem of computing the distance between two objects into a Euclidean distance computation. To achieve this goal, each object is associated to a vector in a high-dimensional vector space where each component of the vector is usually a geometric or semantic feature of the object.

7.2.3 Automatic Instancing

[Schultz and Schumann, 2001] present a technique to automatically generate instancing information **from hierarchically organized objects**. To our knowledge, this method is the only one in the literature that deal with the automatic generation of hierarchical Instancing Information. However, the basic assumption of their method assume that the objects of the scene are already known and ordered in a tree data structure. As a consequence, their method is not able to construct a hierarchy of instances from a totally unstructured scene, which is a far more difficult problem.

An example of their method is presented on Figure 7.10 on the next page.

The method uses a depth-first discover of the initial graph and progressively discover sibling nodes in the tree. We show on center of Figure 7.10 on the facing page the first discover node, *D* and its sibling discover. By repeating this process, the methods is guaranteed to discover all sibling nodes in the tree. From a data structure point of view, this is equivalent to convert the tree to a Directed Acyclic Graph. Each node labeled *I* will so be a reference to a unique element of its category.

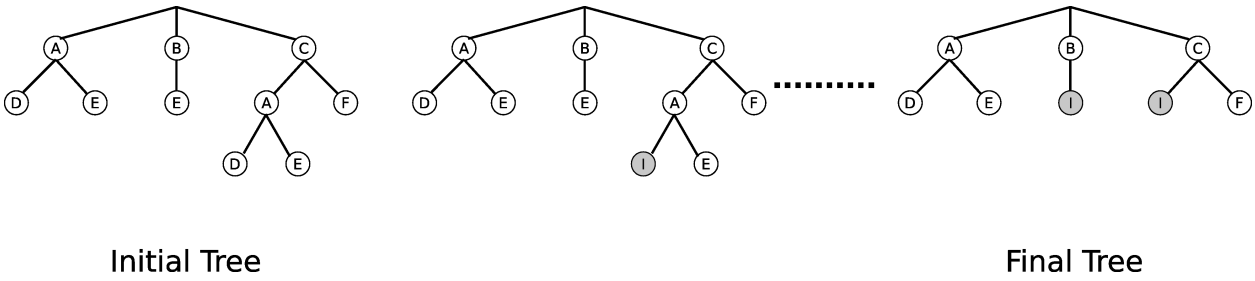


Figure 7.10: Transformation of a tree of objects into a scene graph using method of [Schultz and Schumann, 2001]. Details of each step are given below.

Despite the strong constraints imposed by their methods and the relative simplicity of their approach, the authors report a very good compression ratio, which value reaches 4:1 on their test scenes. Due to its inherent limitations, the scope of this method is however quite limited.

The conclusion that can be made from this state of the art is that, up to our knowledge, no method exist that is able to compute a complete hierarchy of instances from unstructured geometry. We end this chapter with an overview of the work presented in this thesis that allow **efficient recovery of hierarchical instancing information from unstructured data**.

7.3 Overview of our approach

In order to compute a hierarchy of instances, we use a two-step method, starting from the structure at the object level *i.e.* structure of the geometry in tiles, associated with symmetry information. The two steps are represented in Figure 7.11 and the objective of each step is presented below:

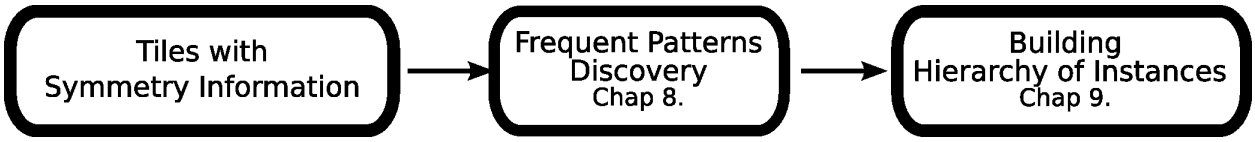


Figure 7.11: Hierarchical Instancing of Geometry: Overview of our approach. Starting from the structure at the object level *i.e.* the decomposition of the geometry in tiles, associated with symmetry information, we use a two-step method to compute a hierarchy of instances of the geometry.

1. The first step, presented in Chapter 8, aims at discovering frequent geometric patterns, *i.e.* objects or set of objects that occur multiple times in a scene. We first identify and explain the complexity of this task and present results that illustrate this complexity on simple test scenes. In a second part, we present an original method that compute frequent geometric patterns based on the detection of local symmetries in the whole scene.

2. The second step, presented in Chapter 9, organizes these patterns into a hierarchy. We identify some criteria that must be respected by the hierarchy in order for it to be efficient, and prove that computing a hierarchy that respects this constraint is an exponential problem *i.e.* a problem where no polynomial algorithm exists to solve this problem. We thus propose an approximation algorithm and present a method that compute a hierarchy of instances optimized for ray-tracing applications.

8

Frequent Geometric Patterns Discovery

In this chapter, we focus on the first step of creating a hierarchy of instances: the discovery of frequent geometric patterns in the scene.

After introducing important generalities about this problem as well as some intuition about its inherent complexity in Section 8.1, we present **two methods** to discover frequent geometric patterns. The first method, presented in Section 8.2 is an *agglomerative approach* that iteratively grows up a pattern by adding an element to it. The second one, presented in Section 8.3 is a *symmetry-based approach* and translates the problem of discovering patterns as the problem of computing local symmetries in the scene.

8.1 Generalities And Notation

As presented in the previous chapter, our input scene is composed of tiles. A pre-process has been used to compute symmetries of each tile (see page 33) and to form the set of congruent tiles using our congruent descriptor (see page 67). We associate to each scene a neighborhood graph that encodes spatial and neighborhood relations between each tile. More specifically, the neighborhood graph $G_N = (V,E)$ is a graph where each vertex is associated to a tile and each edge mean that the geometric distance between the two tile is smaller than a fixed threshold. To compute this distance, we use the technique of sphere-tree proposed by [Quinlan, 1994]. Moreover, we add color information to each vertex of this graph. We affect color to the vertices such that two vertices of the same color are associated to congruent tiles.

An example of a very simple scene and its associated neighborhood graph is presented in Figure 8.1 on the following page.

A set of tiles is said to be *connected* if its neighborhood graph is connected. An *object* is a connected set of tiles. We define the *size of an object* to be the number of tiles that form this object. In the following, an object of size k, i.e. formed of k tiles, will be called a $k-$object. With this definition, a single tile is a $1-$object.

We define *a pattern* as a generic object which is represented in the scene by its *instances*.

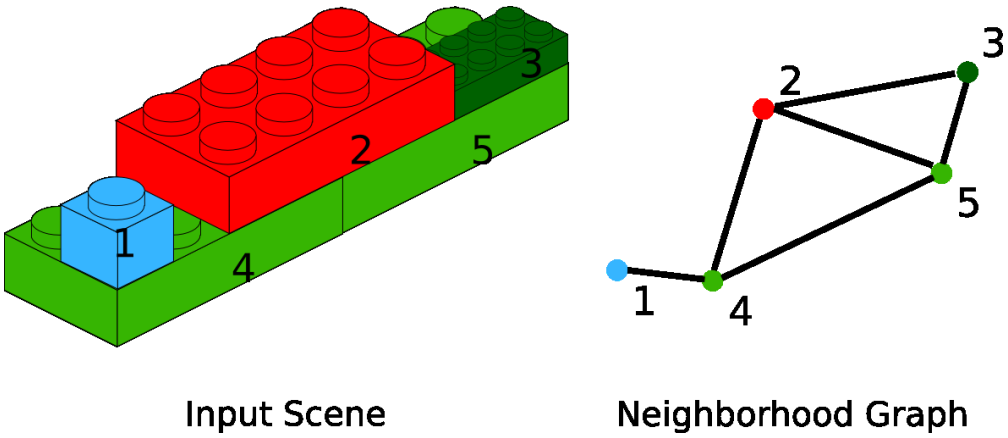


Figure 8.1: We associate to each scene a graph named neighborhood graph that encode spatial proximity between each tile of the scene *i.e.* each node of the graph. Each edge of the graph means that the geometric distance between tiles is less than a given fixed threshold. The nodes associated to congruent tiles have the same color in the neighborhood graph.

Each pattern of the scene has at least one instance, which corresponds to its representation inside the scene.

Definition 11 *Let P be a pattern. The number of instances of a pattern P is noted n_P and all the instances of a pattern form the set I^P:*

$$I^P = \{I_1^P, I_2^P, ..., I_{n_P}^P\}$$

where I_i^P represents the i^{eme} instance of P in the scene.

An instance is a set of tiles. Internally, we represent an instance as a set of integers (the indices of the tiles used to form this instance). In the following, the notation I_i^P will be used to represent both an instance of P and the associated set of integers.

Definition 12 *Let I_1^P and I_2^P be two instances of a pattern P. We say that those two instances are*

- ***identical*** *if they are formed with the same set of tiles, i.e $I_1^P = I_2^P$,*
- ***overlapping*** *if they share at least one tile, i.e $I_1^P \cap I_2^P \neq \emptyset$,*
- ***disjoint*** *if they are not overlapping, i.e $I_1^P \cap I_2^P = \emptyset$.*

A pattern P has n_P instances in the scene, which can possibly overlap. A way of computing the occurency of the instances of a pattern with respect to the non-overlapping constraint is the notion of *frequency*.

Definition 13 *The frequency of a pattern P, denoted* Freq*(P), corresponds to the maximum number of disjoint instances of this pattern in the scene. We have:*

$$\forall P \quad \text{Freq}(P) \leq n_P$$

Computing the frequency of a given pattern is non-trivial as it is equivalent to identifying the Maximum Independent Set in a graph. More details on this frequency counting part will be given in Section 8.2.1.

Based on this notion of frequency, we have:

Definition 14 *A pattern P is* σ*–frequent if* $\text{Freq}(P) \geq \sigma$.

8.1.1 Problem Representation And Complexity

The complete set of frequent patterns can be conceptually organized in the form of a lattice that will be referred to as the *lattice of frequent patterns*. The lattice data structure reflects the fact that a given pattern can be formed in various ways as depicted in Figure 8.2.

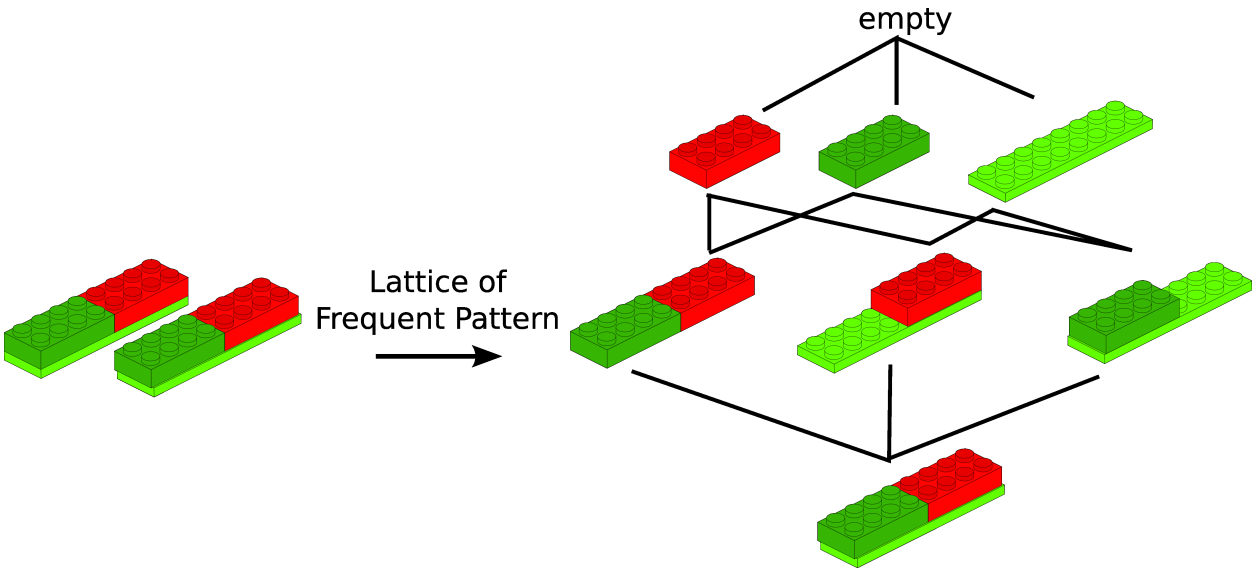


Figure 8.2: The complete set of frequent patterns can be conceptually organized in the form of a lattice that will be referred to as the *lattice of frequent patterns*. The lattice data structure reflect the fact that a given pattern can be formed in various ways. Each pattern at level k is linked to its parents at level $k-1$. Each node at level k has at least two parents and a maximum of $k-1$.

The aim of the frequent pattern discovery step is therefore to discover each node of the lattice and to compute the associated frequency. The output of the frequent pattern discovery step may be quite large, based on the following principle:

Definition 15 *If a pattern is* σ*–frequent, all its sub-patterns are* σ'*–frequent too with* $\sigma' \geq \sigma$.

This principle may seem obvious but it points out the fact that even for scenes of moderate size, where patterns of size 50 are not uncommon, the output of a complete method of retrieving patterns is in theory 2^{50} patterns, *i.e.* the number of subsets of these patterns

As our objects are restricted to be connected set of tiles, the theoretic number of 2^n sub-patterns is never reached. This is illustrated in Figure 8.3 on page 105 which represented

the lattice of frequent patterns for the two plane set represented in the frame at the top left. Each seat contains 16 tiles, which may in theory give 2^{16} frequent sub-patterns. Due to the connectivity constraints, the size of the lattice *i.e.* its number of nodes is only equal to 57.

This large output may still be a problem for very complex scenes which introduces the problem of *compressing the output of the frequent pattern discovery step*. This is the purpose of the next paragraph.

8.1.2 Compressing Frequent Patterns.

We first introduce an important relation between patterns, *the inclusion relation*.

Definition 16 *Let P and $P^{'}$ be two patterns. We say that P is included in $P^{'}$ and note $P \subset P^{'}$ if there exist a mapping between P and a subpart of $P^{'}$.*

Theoretically, computing that a pattern is included in another is a costly operation as a possibly large number of mappings must be tested. If the compression scheme is done as a post-process, a more practical and efficient way of performing this test is to use the instances associated to each pattern. More precisely:

$$P \subset P^{'} \Leftrightarrow \exists i \in [1...n_P]\ I_i^P \subset I_1^{P^{'}}$$

When searching to limit the size of the output of a frequent patterns discovering algorithm, the most natural technique is to discard patterns which are non-*maximal*:

Definition 17 *Let S_σ be the set of σ−frequent patterns. A pattern $P \in S_\sigma$ is said to be **maximal** if:*

$$\nexists P^{'} \in S_\sigma\ P \subset P'$$

The output of a compression strategy that discards every non-maximal patterns will be the leaves of the lattice of frequent patterns as those patterns are the only ones that are not included in any "larger" patterns. In Figure 8.3 on the next page, the only maximal pattern is the leaf of the lattice, *i.e.* the whole seat. This kind of compression scheme lost a lot of information. For example, it removes the information that a plane seat is made of two armrests.

An important aspect in the definition of maximal patterns is that it is frequency-independent, *i.e.* it does not introduce any constraint about the frequency of the patterns. The notion of *closed patterns* can be seen as more restrictive way of compressing patterns, due its dependency on the frequency of patterns.

Definition 18 *Let S_σ be the set of frequent patterns. A pattern $P \in S_\sigma$ is said to be **closed** if:*

$$\nexists P^{'} \in S_\sigma \mid P \subset P^{'} \text{ and } Freq(P) = Freq(P^{'})$$

The closed frequent patterns of the first example is presented in Figure 8.4 on page 106.

On Figure 8.3 on the next page, the number of closed patterns is equal to 2: the armrest and the whole seat, which are represented in pink.

We end this section with two properties related to closed patterns.

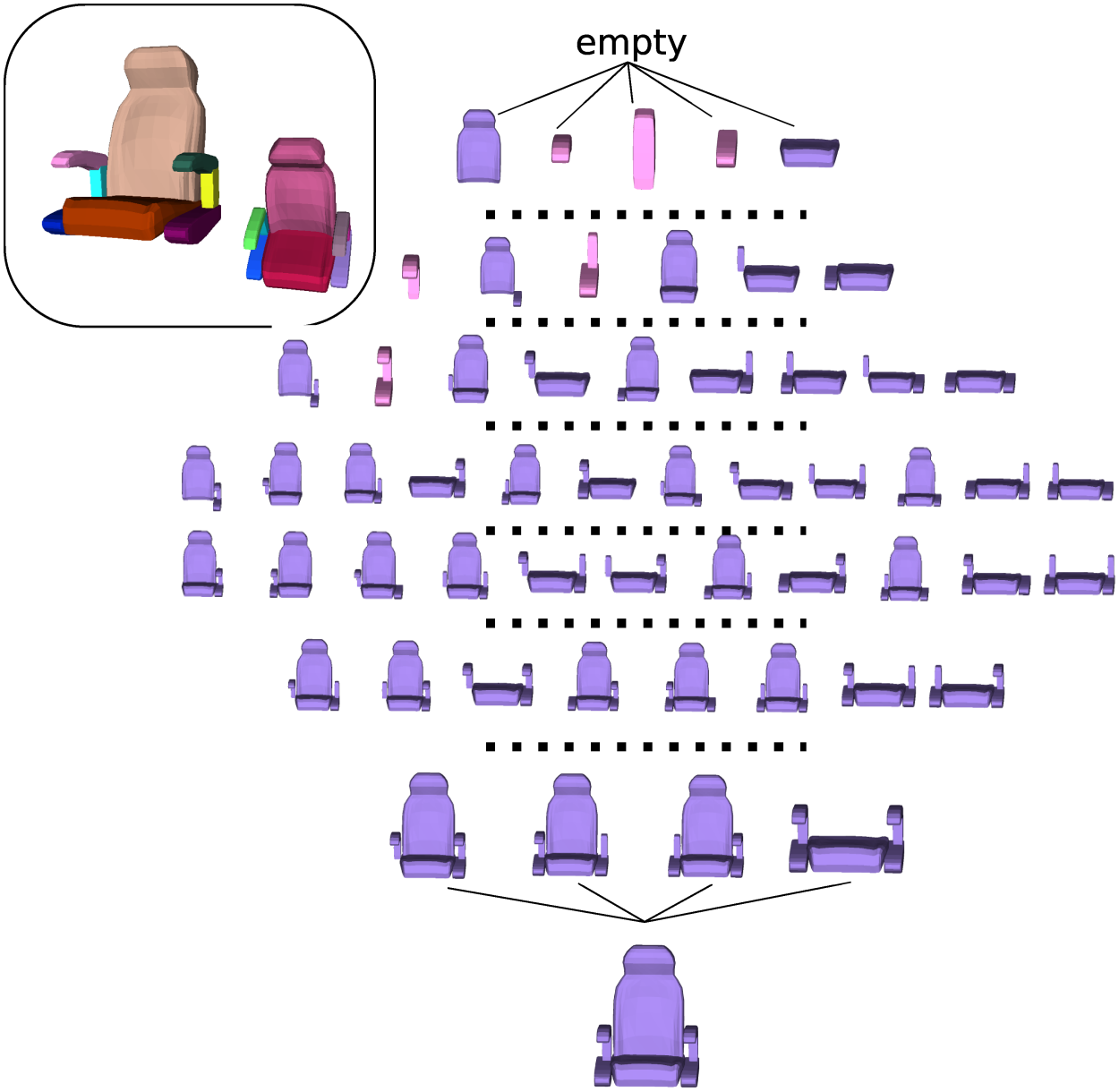


Figure 8.3: Even for a very simple scene, like the one represented in the frame at the top left, the lattice of frequent patterns is quite large and contains 57 nodes. For clarity, edges are not drawn between the levels of the lattice which are separated by dots.

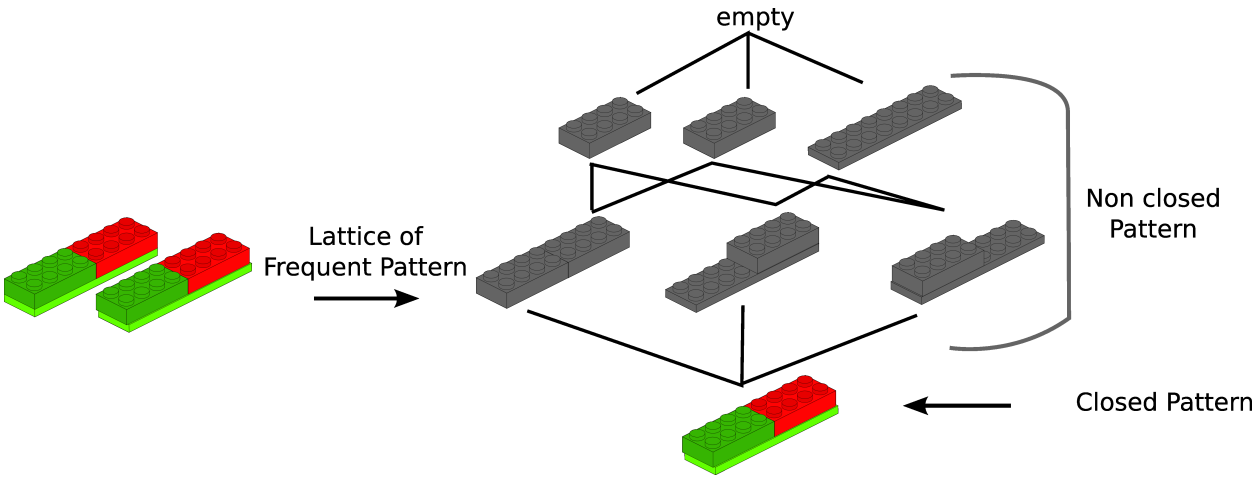


Figure 8.4: When considering the scene on the left, only one of the patterns of the lattice is closed. All the patterns drawn in grey in the lattice, are non-closed as they are included in a bigger pattern with the same frequency.

Property 3 *Non-closed patterns only contain redundant information.*

If a pattern P is non-closed, it means that there exists another pattern Q such that $P \subset Q$ and $\text{Freq}(Q) = \text{Freq}(P)$. It is therefore useless to keep the pattern P as every information it contains (its frequency and its geometry) are completely included in Q.

Property 4 *A maximal pattern is closed. The opposite is not true.*

The proof of this property is immediate as the definition of a closed-pattern is a restriction of the definition of a maximal pattern.

In the following two sections, we present two methods used to compute the frequent geometric patterns, given an input scenes with tiles. We first present in section 8.2 the most intuitive approach to generate frequent patterns which consists at progressively growing a pattern by adding a new tile to it. We rely on the work done by [Kuramochi and Karypis, 2004] to mine frequent graph patterns in a single graph and map it to our problem. In section 8.3, we present a Symmetry-Based approach that compute patterns and instances by computing local symmetry in the scene.

8.2 Agglomerative Approach

In this section, we present the most intuitive technique used to identify the set of frequent patterns: starting from the empty pattern, we progressively increase its size by adding a tile to it.

This method is widely used in the data mining community and we rely on a recent approach proposed by [Kuramochi and Karypis, 2004] to mine frequent graph pattern in a single graph. Although not strictly equivalent to our problem, this parallel will allows us to properly define some important notions and justifies our definition of patterns and instances.

As stated earlier, iterating through each element of the lattice of frequent patterns is a complex task as this lattice contains an exponential number of nodes. To reduce the complexity of this approach, two techniques are used:

- Early pruning of nodes.
- Lattice traversal as a tree.

We detail these two points below, then we present the complete algorithm and present some results in the last section.

8.2.1 Early pruning of nodes

A desirable property when traversing the lattice of frequent patterns is to be able to prune all children of a node if the associated pattern is not frequent as illustrated in Figure 8.5.

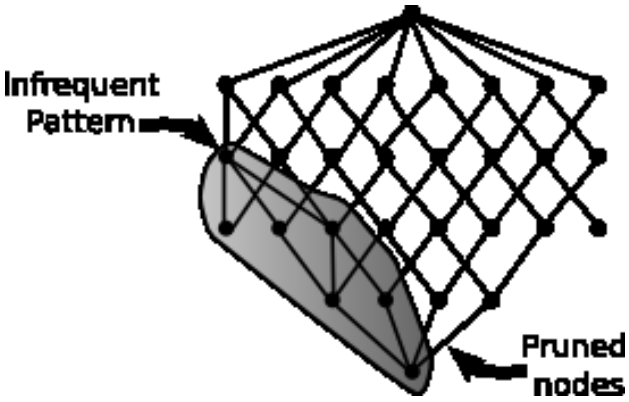


Figure 8.5: One desirable property is to be able to prune the sub-lattice rooted at a node found infrequent, as none of the nodes of this sub-lattices will be declared as frequent.

To be able to prune the lattice of the frequent patterns, the way of computing the frequency of a pattern must respect the *downward closure property*:

Property 5 *downward closure property: The frequency of a pattern decreases as a function of its size.*

This fact justifies the definition of the frequency of a pattern as the maximum number of its disjoint instances. If the frequency of a pattern was defined as the number of its instances, the downward closure property would be violated as presented in Figure 8.6 on the following page.

To determine if a pattern is frequent, it is so necessary to compute its frequency, *i.e.* its maximum number of disjoint instances. This is done by using an *overlap graph*: for a given pattern, this overlap graph is build by considering each instance of the pattern in the scene and associating a node to each of them. Two vertices are then linked by an edge if the two associated instances overlap. This step if critical as obtaining this set turn to be equivalent to find the maximum set of its overlap graph [Kuramochi and Karypis, 2004].

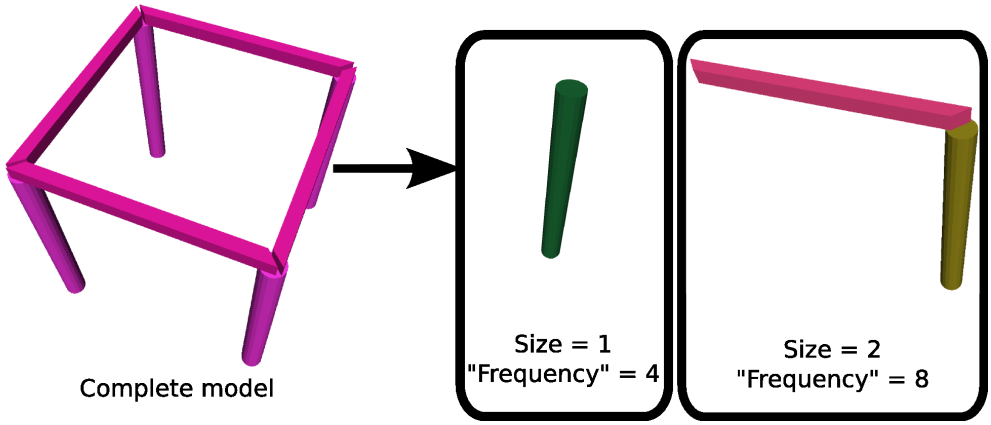


Figure 8.6: If the frequency of a pattern is defined as its number of instances, the downward closure property is violated which is the case in this example. The frequency of the pattern on the right, of size 2, is greater than the frequency of the pattern of size 1.

Maximum Independent Set: We denote an undirected graph by $G = (V,E)$, where V is a set of vertices and E is a set of edges, such that each edge is a set of two vertices in V. A subset of vertices $I \subseteq V$ is called *Independent* if no two vertices in I are connected by an edge in E. An Independent Set I is said to be *Maximal* if its vertices are not a subset of the vertices of a larger (in term of number of vertices) Independent Set, and *Maximum* if there are no larger Independent Set in the graph G. A Maximum Independent Set (MIS) is, of course, Maximal (See example in Figure 8.7). The cardinal of the Maximum Independent Set of a graph G is noted $\Omega(G)$.

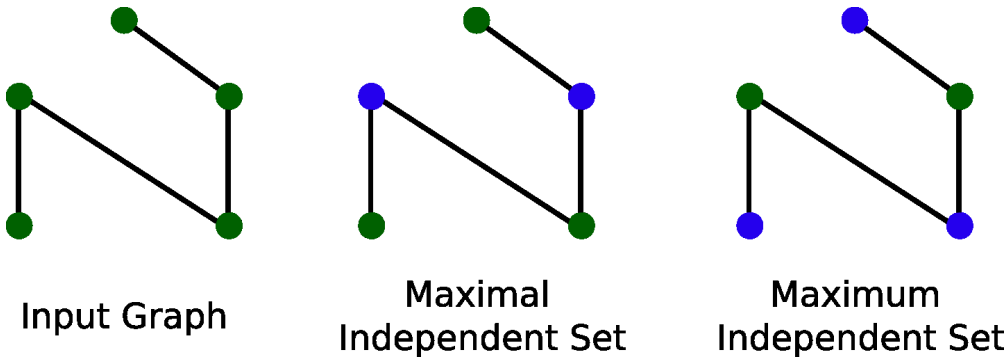


Figure 8.7: Maximal and Maximum Independent Set. The set of vertices of this set are the blue one. Left) Initial Graph. Middle) An Independent Set formed of two vertices. This set is Maximal as no other vertices can be added to form a larger Independent Set. Right) The only Maximum Independent Set of the graph, which is of course also Maximal.

The problem of finding the MIS of a graph was among the first problems proved to be NP-complete [Garey and Johnson, 1979]. It is computationally equivalent to several other central problems for graphs, such as finding maximum clique and finding a minimum vertex cover. This complexity means that, unless $P = NP$, a fact that is widely believed to be false, exact

algorithms are guaranteed to return a solution only in time which increases exponentially with the number of vertices in the graph. However, the importance of the problem and its applicability to a wide-range of domains has attracted a considerable amount of research.

Some exacts implementations exists that compute the MIS of a graph of moderate size. In particular, in our work, for small overlap graph (less than 100 vertices) we use an exact implementation with the program *wclique* [Ostergard, 2002] and an approximation algorithm for large graph. The program *wclique* is an exact **maximum clique** problem solver and can be used to compute the MIS of a graph as the MIS problem on a graph G is equivalent to the maximum clique problem on G's complement graph $\tilde{G}$. The approximation algorithm we used [Kako et al., 2005] to compute the MIS is a greedy approach made of two simple steps:

1. pick the vertex of lowest degree
2. mark it as part of the solution. Remove it from the graph as well as all its neighbors. Go to 1.

8.2.2 Lattice traversal as a tree

The naive approach for transforming a lattice into a tree is simply to check, for each new element, if it has already been discovered by the depth-first search. This technique has high complexity has it must test a large number of patterns for congruency. A more clever way of doing this transformation and to eliminate these redundant computations is to assign to each node at level k (corresponding to a $k-$pattern) a unique parent node a level $k-1$, named the *generating parent*.

With this technique, each node in the lattice has now an unique ancestor hence enabling the traversal of the lattice as a tree. This technique of transforming a lattice into a tree has been previously used in the graph-data mining community [Vanetik et al., 2002; Yan and Han, 2002].The problem we have to solve is so:

Problem Statement: Let P^k be a $k-$pattern and P^{k+1} a $k+1-$pattern formed by extension of P^k. We have to determine if P^k is the generating parent of P^{k+1}.

We use the instances associated to each pattern. Let $S_{P^{k+1}}$ be the set of tiles of all instances of P^{k+1}:

$$S_{P^{k+1}} = \cup\, I_i^{P^{k+1}} \quad i \in [1..n_{P^{k+1}}]$$

We pick, in the set $S_{P^{k+1}}$, the lowest (or the largest) value that does not disconnect the corresponding instance. This last condition is very important as we restrict instances to be a connected set of tiles (see Section 8.1 on page 101). The choice between the lowest or largest value in this set is unimportant but it must be consistent during the whole algorithm.

By removing this value from the corresponding instance, we obtain a new instance I and two cases can occur:

- Either this instance I is an instance of P^k. In this case P^k is the generating parent of P^{k+1}.
- If I is not an instance of P^k, then P^k is not the generating parent of P^{k+1}. We do not need to consider P^{k+1} as it will be considered later by extending another $k-$pattern.

8.2.3 Algorithm Overview

The agglomerative approach can be presented by the two algorithms presented in page 110. The central part of this approach is the *ExtendPattern* method that recursively compute all potential extensions of an input pattern.

Algorithm 1 Frequent Pattern Discovery by depth-first visit of the lattice of Frequent Pattern.

Require: The frequency threshold σ
Ensure: Discover every frequent Pattern
1: $\mathcal{P}^1 \leftarrow$ All frequent $1-$Pattern
2: **for all** $P^1 \in \mathcal{P}^1$ **do**
3: ExtendPattern(P^1, *FrequentPattern*, σ)
4: **end for**
5: return FrequentPattern

Algorithm 2 ExtendPattern

Require: A $k-$Pattern P,
Require: The set of FrequentPattern already found *FrequentPattern*
Require: The frequency threshold σ
Ensure: Extend a given pattern
1: **for all** all instance I^P of the Pattern P **do**
2: Generate all possible $(k+1)-$instances from I^P.
3: **end for**
4: *NewPattern* $\leftarrow$ new patterns generated
5: **for all** Q in *NewPattern* **do**
6: **if** P is the Generating Parent of Q **then**
7: Compute frequency of Q
8: **if** Q.frequency $\geq \sigma$ **then**
9: *FrequentPattern* $\leftarrow$ *FrequentPattern* $\cup Q$
10: ExtendPattern(Q,*FrequentPattern*)
11: **end if**
12: **end if**
13: **end for**

8.2.4 Filtering Frequent Patterns

The output of the algorithm is a set of frequent patterns that occur in the scene with respect to the frequency threshold. We can use a compression scheme by discarding frequent patterns that are not closed. It is interesting to note that, by slightly modifying the algorithm *ExtendPattern*, we can directly output a set of **closed frequent patterns**. More specifically, if one switches the frequency counting part and the parent identification part, non-closed patterns will be discarded. In the pseudo-code of the *ExtendPattern*, this is equivalent to switch line 6 and 8. However, the process of counting the frequency of the pattern is a costly operation

so this method, while restricting the output of the algorithm to closed patterns, results in a significant increase in the runtime of the algorithm. In our experiments, we note that the runtime increases by approximatively 50%.

8.2.5 Results

We test this technique on some sample scenes presented in Figure 8.8.

Scene Name	#Polygons	#Tiles
Plane (part)	13,384	32
Piece	40,410	707
Building	81,451	485
Plane (complete)	182,184	480

Figure 8.8: The scenes used to test the frequent pattern discovery using an agglomerative approach

The results of our tests are presented in Figure 8.9 and an illustration of the frequent patterns found on two models is represented in Figure 8.10 on the next page

Scene Name / Frequency Threshold	# Frequent Patterns	# Closed Patterns	Runtime (secs)
Plane (part)			
8	3	1	0.27
4	30	2	3.53
2	30	2	3.48
Piece			
10	4	4	0.07
5	20	13	0.62
4	990	30	185
2	-	-	> 1h
Building			
20	2	2	0.17
10	16	9	3
5	58	26	48.52
2	-	-	> 1h
Plane (complete)			
30	4	2	5.02
20	42	5	190
10	-	-	> 1h

Figure 8.9: Results obtained for the test scenes presented in Figure 8.8. As expected, the runtime increases as the frequency decreases. Cells with no values means that the algorithm has not finished after more than 1 hour of computation. All the frequencies have been computed using an exact MIS implementation.

The main signification of these results is that the **size of the scene that can be handled**

by this algorithm is limited. More precisely, searching for frequent pattern is a very complex process when the frequency threshold is low. In this case, the lattice of frequent patterns is very large since a large number of patterns can be formed at each step.

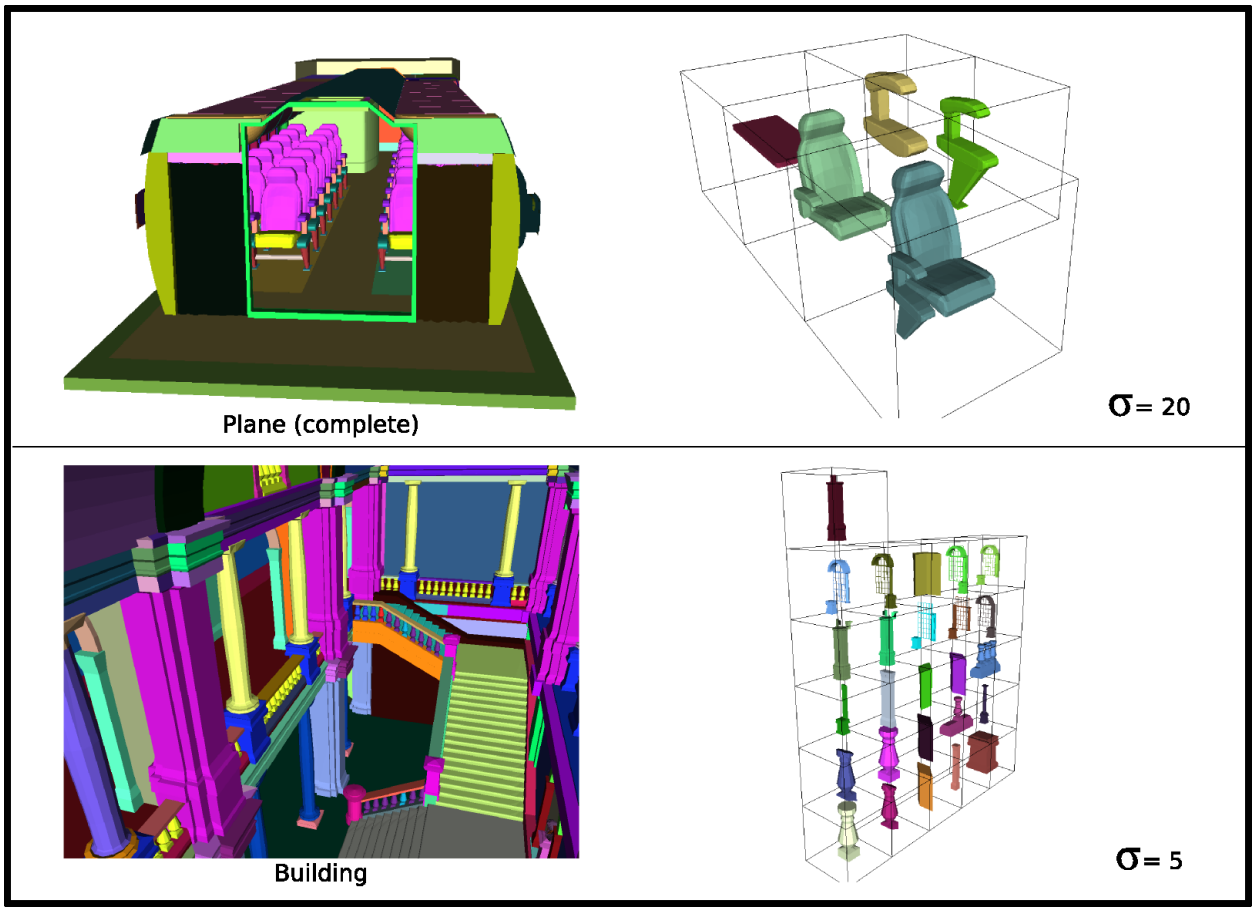


Figure 8.10: Set of frequent closed patterns obtained with the agglomerative approach, for a given σ. All the patterns reported here have a frequency greater or equal to σ. The complexity of the agglomerative approach is reasonable only for high value of frequency threshold. As a consequence, the set of patterns discovered by this method is limited and not representative of all the information contained in the scene.

8.3 Symmetry-Based approach

8.3.1 Basic Statement

As seen in the previous section, searching for frequent patterns based on the traversal of the lattice is a costly process, especially when the frequency threshold is low. We now present a second approach to compute frequent patterns based on the following statement:

Basic assumption: Let I_1^P and I_2^P be two instances of a pattern P. There exists *at least one* euclidean transform T, such that $I_2^P = T(I_1^P)$.

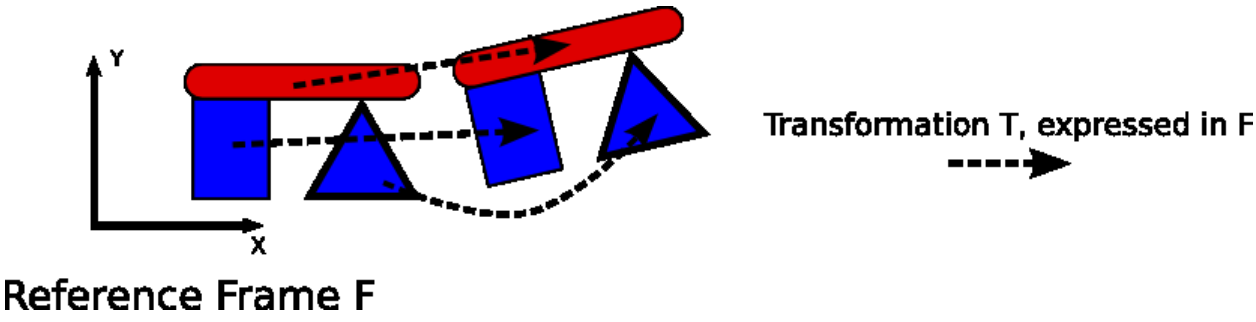


Figure 8.11: There exists at least one euclidean transform T, expressed in F that transform one instance of a pattern into the other.

When considering the input scene as a single entity, the previous statement can be reformulated as follows:

Property 6 *Two instances of a pattern form a* ***local symmetry*** *of the input scene.*

An example of a local symmetry in a complex scene is presented in Figure 8.12 on the following page

This formulation leads to the following statement:

Property 7 *Finding frequent patterns in a scene is equivalent to finding local symmetries, when considering the scene as a single entity.*

8.3.2 The Transformation Space

To compute local symmetries, we consider the transform space generated by translations, rotations and reflections. This topological space, denoted Γ is a non-euclidean manifold formed by the Cartesian product of several spaces. At least two formulations can be used to generate this space, depending on the choice of representation for 3D rigid rotations: quaternions and Euler angles.[1]

[1]We have discarded a third representation, the rotation matrix, because of the difficulty to associate a efficient metric to it (see [Kuffner, 2004] for a detailed discussion).

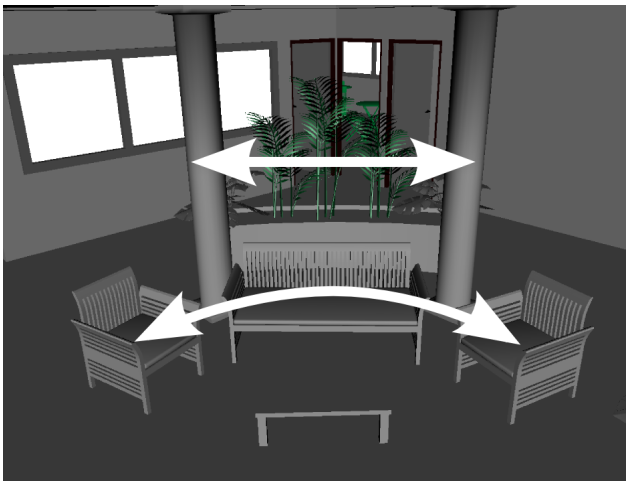


Figure 8.12: Two instances of a pattern form a local symmetry in the scene. This is for example the case for the two chairs and the two pillars in this scene.

- The first possible representation for rotation is a real projective space, denotes $\mathbb{R}P^3$. This space can be represented by three-dimensional sphere embedded in $\mathbb{R}^4$ with antipodal points identified. That is, $\mathbb{R}P^3 = S^3/x \backsim -x$, in which $S^3 = \{x \in \mathbb{R}^4, ||x|| = 1\}$. Each element $x = (x_1, x_2, x_3, x_4) \in \mathbb{R}P^3$ is a unit quaternion $x_1 + x_2\mathrm{i} + x_3\mathrm{j} + x_4\mathrm{k}$, representing a 3D rotation. The metric for two points $x, y \in \mathbb{R}P^3$ is defined as the length of the arc between these two points on the surface of the sphere, that is:

$$\mathrm{dist}_{\mathbb{R}P^3}(x,y) = \min(\mathrm{acos}(x \cdot y), \mathrm{acos}(x \cdot (-y)),$$

  in which $x \cdot y$ denotes the dot product for vectors in $\mathbb{R}^4$.

- The second possible representation for rotation is to used Euler angles. In this case, each rotation is represented as a vector $(x_1, x_2, x_3), x_i \in [-\pi, \pi] / -\pi \backsim \pi$. The topology of this space is $S^1 \times S^1 \times S^1$, where $S^1 = [0,1]/0 \backsim 1$, and the metric for two points of this space $p, q \in S^1$ is defined as:

$$\mathrm{dist}_{S^1}(p,q) = \min(|q-p|, 1-|q-p|)$$

Whatever the choice of the rotation representation is, the space Γ is a non-euclidean manifold formed by the Cartesian product of $(T_1, \mathrm{dist}_{T_1})$ and $(T_2, \mathrm{dist}_{T_2})$ and so the weighted metric for two points q, p in the Cartesian product $T_1 \times T_2$ is defined as:

$$\mathrm{dist}^2_{T_1 \times T_2} = \mu_{T_1} \mathrm{dist}^2_{T_1}(q,p) + \mu_{T_2} \mathrm{dist}^2_{T_2}(q,p)$$

where μ_{T_1} and μ_{T_2} are two weights used to give equal importance for the two spaces T_1 and T_2.

In our work, we represent rotations with quaternions which results in the definition of the space $\Gamma = \mathbb{R}^3 \times \mathbb{R}P^3$. Γ is a $7-$dimensional space where each point in Γ has the form $\mathcal{P} = (\boldsymbol{T}, \boldsymbol{Q})$ where $\boldsymbol{T} = [t_x, t_y, t_z]$ represents the translation part and $\boldsymbol{Q} = [q_w, q_x, q_y, q_z]$ represents a quaternion, *i.e.* the rotation part. To efficiently compute neighborhood in this non-Euclidean space, we use the approach proposed by [Atramentov and LaValle, 2002].

8.3.3 Filling the Transformation space

Our goal is to encode euclidean transforms between tiles of the scene and to seek a set of (almost) equal transformations to be able to find *frequent closed patterns*. As we are interested in finding instances, *i.e.* repetition of geometry, it is useless to compute the euclidean transforms between every tiles of the scene and we restrict ourselves to the euclidean transforms between pairs of congruent tiles (see figure 8.13).

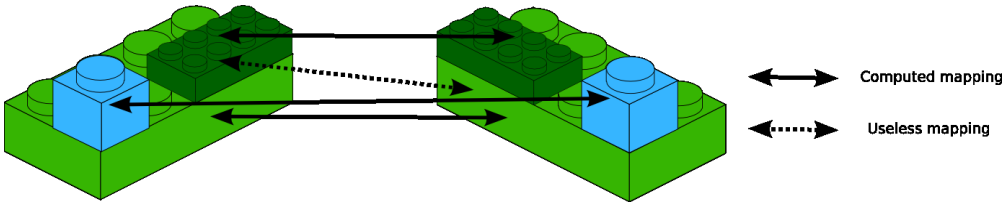


Figure 8.13: When filling our transformation space, we do not need to iterate through every pair of tiles in the scene. This is illustrated on this figure where congruent objects are shown with the same color. The computed mappings are represented as continuous lines. The dashed lines represent useless mappings, that is mappings between two non-congruent tiles.

The first pass of the process is to iterate through the set of pairs of congruent tiles and to encode their relations, *i.e* filling the transformation space Γ.

One important point is that isometries that are computed can be either *direct*, that is rotation and/or translation or *indirect* such as reflection (see page 40). To separate such isometries we divide the set of isometries in two sets, Γ^+ and Γ^- for respectively proper and improper. In practice, we test the nature of the isometry by looking at the determinant of its associated matrix.

A single pair of tiles can lead to more than one euclidean transform, *i.e.* more than one point in Γ, because of the symmetries attached to each tile. We detail in the following two paragraphs the way of filling the transformation space, depending on the nature of the symmetries of each tile: discrete or continuous.

Discrete Symmetries

In the following, we consider two congruent tiles $\mathcal{T}_i$ and $\mathcal{T}_j$ and named $\mathbf{p}_i$, $\mathbf{p}_j$ their respective centers of mass. The method used to generate points in Γ for discrete symmetries is presented in the algorithm 3 on the following page. In line 1 of this algorithm, the method Isometry computes all possible isometries that map $\mathcal{T}_i$ on $\mathcal{T}_j$.

In this method, I is the method that computes an isometry that map frame F_i onto frame F_j. In the same time, this function also compute the *reverse* mapping, i.e. the mapping from $\mathcal{T}_j \rightarrow \mathcal{T}_i$. For the rotation part, the quaternion of the reverse mapping is simply the inverse quaternion of the original mapping. Concerning the translation part, for the original mapping, we have:

$$\mathbf{t}_{ij} = \mathbf{p}_j - s_i \mathbf{p}_i$$

and for the reverse mapping:

$$\tilde{\mathbf{t}}_{ij} = \mathbf{p}_i - s_i^{-1} \mathbf{p}_j$$

Algorithm 3 Generate Points in Γ

Require: Two congruent tiles $\mathcal{T}_i$ and $\mathcal{T}_j$ with discrete symmetries.
Ensure: Add to the space Γ **all** the transformation that map $\mathcal{T}_i \rightarrow \mathcal{T}_j$.

$S_I = \text{Isometry}(\mathcal{T}_i, \mathcal{T}_j)$
for all $s_i \in S_I$ **do**
  $\mathbf{t}_{ij} = \mathbf{p}_j - s_i\mathbf{p}_i$
  $\mathbf{q}_{ij} = \text{quaternion}(s_i)$
  if $\det(s_i) = 1$ **then**
    $\Gamma^+ \leftarrow \Gamma^+ \cup \mathbf{T}_{ij} = (\mathbf{t}_{ij}, \mathbf{q}_{ij})$
    $\tilde{\mathbf{q}}_{ij} = \mathbf{q}_{ij}^{-1}$ {Compute the reverse mapping $\mathcal{T}_j \rightarrow \mathcal{T}_i$}
    $\tilde{\mathbf{t}}_{ij} = -\tilde{\mathbf{q}}_{ij} \cdot \mathbf{t}_{ij}$ {Compute the reverse mapping $\mathcal{T}_j \rightarrow \mathcal{T}_i$}
    $\Gamma^+ \leftarrow \Gamma^+ \cup \tilde{\mathbf{T}}_{ij} = (\tilde{\mathbf{t}}_{ij}, \tilde{\mathbf{q}}_{ij})$ {Compute the reverse mapping $\mathcal{T}_j \rightarrow \mathcal{T}_i$}
  else
    $\Gamma^- \leftarrow \Gamma^- \cup \mathbf{T}_{ij} = (\mathbf{t}_{ij}, \mathbf{q}_{ij})$
    $\tilde{\mathbf{q}}_{ij} = \mathbf{q}_{ij}^{-1}$ {Compute the reverse mapping $\mathcal{T}_j \rightarrow \mathcal{T}_i$}
    $\tilde{\mathbf{t}}_{ij} = \tilde{\mathbf{q}}_{ij} \cdot \mathbf{t}_{ij}$ {Compute the reverse mapping $\mathcal{T}_j \rightarrow \mathcal{T}_i$}
    $\Gamma^- \leftarrow \Gamma^- \cup \tilde{\mathbf{T}}_{ij} = (\tilde{\mathbf{t}}_{ij}, \tilde{\mathbf{q}}_{ij})$ {Compute the reverse mapping $\mathcal{T}_j \rightarrow \mathcal{T}_i$}
  end if
end for

Algorithm 4 Isometry

Require: Two congruent tiles $\mathcal{T}_i$ and $\mathcal{T}_j$ with discrete symmetries.
Ensure: Return all isometries that map $\mathcal{T}_i$ on $\mathcal{T}_j$.

for all all possible frame F_i of $\mathcal{T}_i$ **do**
  for all all possible frame F_j of $\mathcal{T}_j$ **do**
    $S_I \leftarrow S_I \cup I(F_i, F_j)$
  end for
end for
return S_I

which transforms into:

$$\begin{aligned} \tilde{\mathbf{t}}_{ij} &= \mathbf{p}_i - s_i^{-1}\mathbf{p}_j \\ s_i\tilde{\mathbf{t}}_{ij} &= s_i\mathbf{p}_i - \mathbf{p}_j = -\mathbf{t}_{ij} \\ \tilde{\mathbf{t}}_{ij} &= -s_i^{-1}\mathbf{t}_{ij} \end{aligned}$$

If s_i is a direct isometry, this last expression transform into $\tilde{\mathbf{t}}_{ij} = -\tilde{\mathbf{q}}_{ij} \cdot \mathbf{t}_{ij}$ (line 9). If not, then s_i is an indirect isometry which remove the negate sign and so $\tilde{\mathbf{t}}_{ij} = \tilde{\mathbf{q}}_{ij} \cdot \mathbf{t}_{ij}$ (line 14).

Continuous Symmetries

Continuous symmetries are a special case that must be handled with care. In this case, it is not possible to encode the relations between two tiles with a finite number of points in the transformation space. Each tile has an infinite number of frames and the relation between those tiles traces out multiple points in the transformation space (see Figure 8.14).

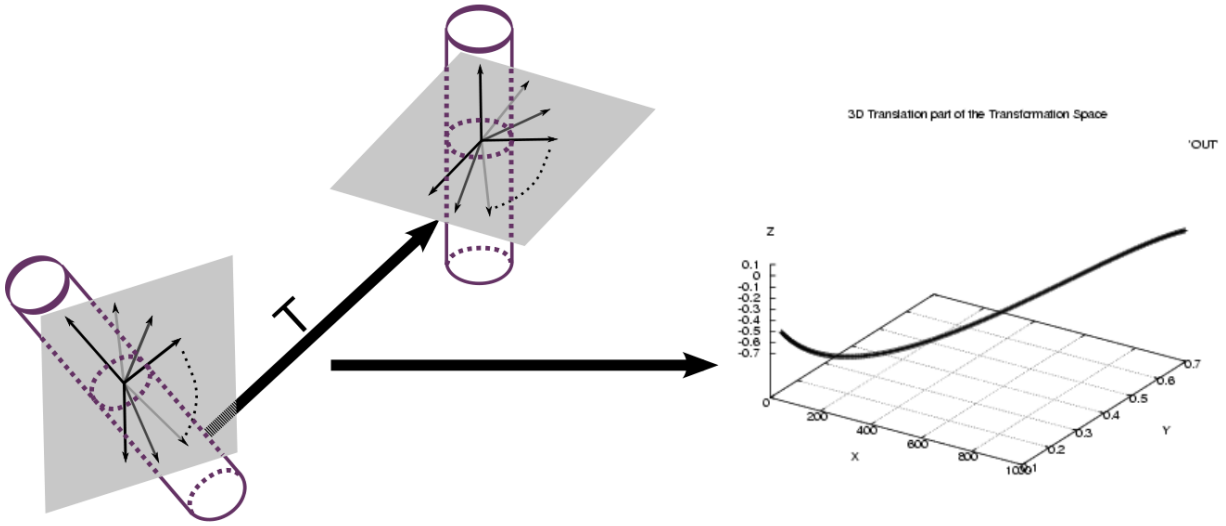


Figure 8.14: In case of revolution surface, the transformation between tiles trace out a curve in transformation space. A representation of a sub-space of Γ corresponding to the translation part is shown on the right. This curve has been built by sampling the set of all potential frames of both tiles and computing the associated transformation.

An equivalent problem is cited in [Mitra et al., 2006] where the problem arises for differential surface patches at umbilic points, *i.e.* those for which a frame cannot be established uniquely. However, the authors do not deal with this problem and discard these points from their sample set. This simple solution is obviously not possible in our case as we do not deal with a sampling set but with a real set of objects. Discarding these cases would be equivalent to erase tiles with continuous symmetries from our 3D scene, which is unrealistic.

We now describe how to deal with tiles with continuous symmetries in two separate ways: we first present a method for efficiently computing the case of revolution surfaces, *i.e.* the

case where there exists a fixed vector in the tile frame. In this case, whe show that the set of transformation is a curve in the transformation space. Then, we present a method to compute the transformation in the special case of sphere surfaces, where no vector is fixed and hence any rotation is allowable.

Tiles with cylindrical symmetries Let $\mathcal{T}_i$ and $\mathcal{T}_j$ be two congruent tiles, and $\mathbf{p}_i$ and $\mathbf{p}_j$ their respective center of mass. If $\mathbf{R}_{ij}$ represent the rotation that rotate local frame of $\mathcal{T}_i$ on the local frame of $\mathcal{T}_j$, we obtain a point in $7-$dimensional transformation space Γ as $\mathbf{T}_{ij} = (\mathbf{R}_{ij}, \mathbf{t}_{ij})$, where $\mathbf{t}_{ij} = \mathbf{p}_j - \mathbf{R}_{ij}\mathbf{p}_i$.

If $\mathcal{T}_i$ (and so $\mathcal{T}_j$) have continuous symmetries, we can attach an infinite number of frames to these tiles (only one axis is determined). In this case the rotational part of the homogeneous transform represents all the rotation that map one vector to another. Given this, we can see that if we want to obtain the equation of the curve, one must be able to analytically found all the rotation that map one vector to another. To resolve it, we rely on quaternion, and more specifically spherical quaternion interpolation, also named slerp:

Given two quaternions, q_0 and q_1 that we want to interpolate, parametrized by a single value t, we have:

$$\text{slerp}(t; q_0, q_1) = \frac{q_0 \sin((1-t)\theta) + q_1 \sin(t\theta)}{\sin(\theta)} \quad t \in [0..1]$$

In order to compute all possible rotations that map one vector v_1 to another v_2, we first compute two basic quaternions that correspond to the shortest and longest possible rotations. The most used definition when we want to map one vector to another, is to define the axis of rotation as the cross-product of the two vectors and the rotation angle as the arc-cosine of the scalar product of the two vectors. More specifically, we will denote this rotation as the shortest one and defined it as for two vectors v_1 and v_2:

$$q_0 = R_{short}(n, \alpha) \Leftrightarrow \begin{cases} n = v_1 \times v_2 \\ \alpha = \text{acos}(v_1 \cdot v_2) \end{cases}$$

This rotation is defined as the shortest one as every other rotation $R' = (n', \alpha')$that map v_1 to v_2 will have $\alpha' > \alpha$. Our quaternion q_0 is so defined as the shortest quaternion, i.e. associated to R_{short}.

We also define to longest quaternion with:

$$q_1 = R_{long}(n, \alpha) \Leftrightarrow \begin{cases} n = \frac{v_1+v_2}{||v_1+v_2||} \\ \alpha = \pi \end{cases}$$

Theorem 9 *By Spherical Linear Interpolation between q_0 and q_1, we obtain all possible rotations between two vectors.*

As the angle between our two quaternions q_0 and q_1 is equal to $\frac{\pi}{2}$ when treating them as unit-length vectors in $4-$dimensional space, we have:

$$\text{slerp}(t; q_0, q_1) = q_0 \sin\left((1-t)\frac{\pi}{2}\right) + q_1 \sin\left(t\frac{\pi}{2}\right)$$

which transforms into:

$$\text{slerp}(t; q_0, q_1) = q_0 \cos\left(\frac{t\pi}{2}\right) + q_1 \sin\left(\frac{t\pi}{2}\right)$$

In the space Γ, the curve $\mathcal{C}_{ij}(t)$ that encode all the mappings from tile $\mathcal{T}_i$ to tile $\mathcal{T}_j$ is defined as:

$$\mathcal{C}_{ij}(t) = (\mathbf{t}_{ij}(t), \mathbf{q}_{ij}(t)) \quad \text{with} \begin{cases} \mathbf{q}_{ij}(t) = q_0 \cos(\frac{t\pi}{2}) + q_1 \sin(\frac{t\pi}{2}) \\ \mathbf{t}_{ij}(t) = \mathbf{p}_j - \mathbf{q}_{ij}(t)\mathbf{p}_i \end{cases}$$

Tiles with spherical symmetries. In case of spherical objects, every frame can be affected to each tile as no axis is fixed. We have to compute the set of transforms associated to such objects in the Γ space.

Let $\mathcal{T}_i$ and $\mathcal{T}_j$ be two congruent tiles, and $\mathbf{p}_i$ and $\mathbf{p}_j$ their respective centers of mass. Let us denote $\mathbf{R}_{ij}$ the rotation that rotate a local frame of $\mathcal{T}_i$ onto one of $\mathcal{T}_j$. As seen previously, we have $\mathbf{t}_{ij} = \mathbf{p}_j - \mathbf{R}_{ij}\mathbf{p}_i$. The set of rotation $\mathbf{R}_{ij}$ is not constrained so we cannot do any assumption about its value.

Geometrically, it is easy to see that the value of the translation lies at the surface of the sphere of center $\mathbf{p}_j$ and of radius $||\mathbf{p}_i||$ in Γ(see Figure 8.15).

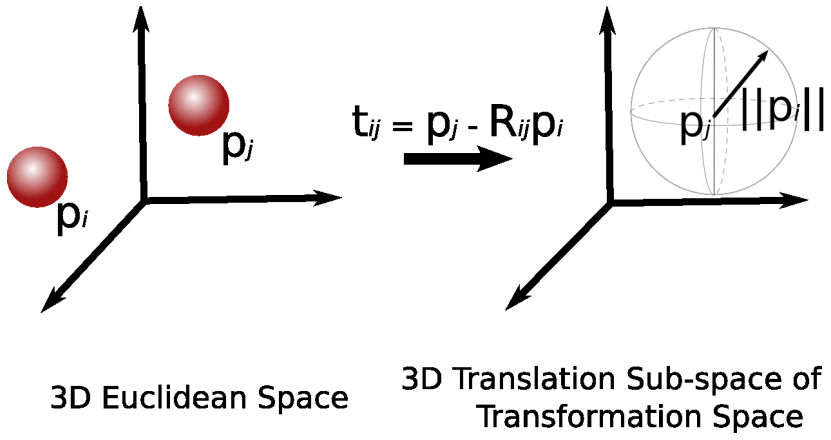


Figure 8.15: Left) Two spheres centered on points $\mathbf{p}_i$ and $\mathbf{p}_j$ in the euclidean space. Right) The translation part of the transformation from $\mathcal{T}_i$ to $\mathcal{T}_j$ describes a sphere in Γ centered in $\mathbf{p}_j$ of norm $||\mathbf{p}_i||$.

8.3.4 Forming patterns

As each point that we have inserted in the transformation space is associated to a mapping between two tiles, the next step is to use this information to compute the set of instances and patterns. This process is done by establishing clusters in the transformation space.

We proceed in two steps:

1. We form clusters of points that come from discrete symmetries,
2. we add information corresponding to continuous symmetries.

We detail these two processes below.

Forming clusters of discrete symmetries

A very popular technique for partitioning and clustering is the K-mean. Yet very efficient and easy to implement, this method need the *a-priori* definition of the number of clusters that we aimed to obtain. Another method of clustering is the *mean-shift clustering* [Comaniciu and Meer, 2002], based on a density estimation technique (see Figure 8.16).

The main idea of the mean-shift clustering is to consider a point cloud in which we want to form clusters as samples from an unknown density function. The mean-shift is an efficient technique to find the modes of this density, *i.e.* the places where its gradient is equal to zero without knowing this density. The modes of the density are the places where most of the samples are and are hence the places where the clusters must be placed. Some technicals aspects of the mean-shift clustering are given in Appendix C.

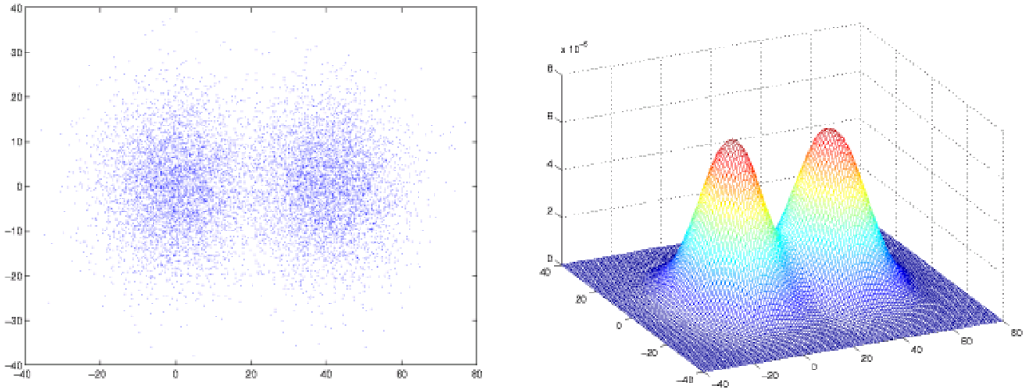


Figure 8.16: The basic idea of mean shift clustering is to consider each point in a point cloud (here in 2D) as a sample from an unknown density function (right). As forming the clusters in this space is equivalent to find the places where points accumulate, this also the places where the gradient of the density is equal to zero. The mean-shift is an elegant way to find these places without knowing the density function or having any assumption about it.

We use mean-shift to form clusters of points in Γ and use the technique of [Atramentov and LaValle, 2002] to compute nearest neighborhood in this space.

Forming clusters of continuous symmetries

For tiles with cylindrical or spherical symmetries, it is easy to test if the transformation belong to a given cluster or not.

Each cluster of discrete symmetries C in Γ^+ or Γ^- is associated with a transformation $T_c = (t_c, q_c)$. For two tiles $\mathcal{T}_i$ and $\mathcal{T}_j$, centered in $\mathbf{p}_i$ and $\mathbf{p}_j$, with continuous symmetries, we consider the mapping $\mathcal{T}_i \rightarrow \mathcal{T}_j$:

- If the tiles have spherical symmetries, testing if these mapping is compatible with the cluster C is done by checking that:

$$\mathbf{p}_j = T_C \mathbf{p}_i$$

- If the tiles have cylindrical symmetries, testing if these mapping is compatible with the cluster C is done by checking that:

$$\begin{cases} \mathbf{p}_j = T_C \mathbf{p}_i \\ \mathbf{v}_j = q_c \mathbf{v}_i \end{cases}$$

  where $\mathbf{v}_i$ and $\mathbf{v}_j$ represent the axis of the cylindrical symmetries respectively associated to $\mathcal{T}_i$ and $\mathcal{T}_j$.

If the transformation space is formed of a large number of points, computing these tests may be time-consuming. We can use the analytic expression of the points corresponding to spherical and cylindrical symmetries to quickly discard part of the transformation space and hence reducing the set of points that need to be tested. The first step consists in computing a hierarchical partitioning, a kd-tree, of the transformation space based on the positions of the clusters. Using this spatial data structure, we are able to prune points in a way dependent of the type of symmetries of the tiles:

Rejection test for spherical symmetries As presented above, the points associated to the translation part of the mapping $\mathcal{T}_i \rightarrow \mathcal{T}_j$ describe a sphere centered in $\mathbf{p}_j$ of norm $||\mathbf{p}_i||$. A simple rejection test consists in computing the bounding box of this sphere and use it to discard points that do not lie inside it. For each of the point that lies inside the bounding box, we test if the point is at the surface of the original sphere.

Rejection test for cylindrical symmetries This time, the points describe a curve whose equation is given above. We use this curve to quickly pruned space bucket and hence reject its associated point. Once again, for each point that lie inside space bucket intersected by the curve, we test if the point lie on the curve.

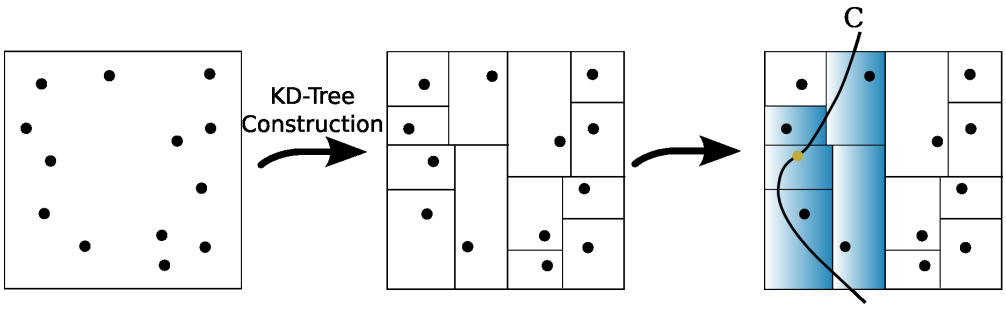


Figure 8.17: Rejection test for Cylindrical Symmetries. We build a kd-tree in the Γ space, using the clusters of discrete symmetries computed with the mean-shift. We then quickly prune space bucket which do not intersect the curve of transformation. The space bucket that are not discard are represented in blue.

Patterns Generation

As each point in the transformation space is associated with a mapping between two tiles, at the end of the clustering process, each cluster is associated with a map which implicitly

defines a local symmetry in the scene. As first remarked by [Mitra et al., 2006], the spatial relation of tiles is lost during the mapping to the transformation space. This implies that we have no insurance that the instances of the local symmetry form a connected set of tiles.

To ensure that our instances are connected set of tiles, we simply compute the connected components inside each cluster. We start from a tile and its image through the transformation of the clusters. Then, we repeat the process for each neighbor tile. This results in forming two instances, *i.e.* set of connected tiles, of the same pattern.

The whole set of Information that are contained in the clusters can be seen as a set X. For a single cluster, the associated information is an equivalence relation, denoted $\sim$.

Property 8 *Let X be the set of objects computed by the clustering step and let $\sim$ the equivalence relation on the set X. Computing the set of patterns and its instances is equivalent to compute the set of all equivalence classes in X.*

For each of the patterns, we compute its frequency and decide to keep it or not depending on the frequency threshold.

Filtering Frequent Closed Patterns

As presented earlier, a pattern $P \in S_P$ is said to be **closed** if:

$$\nexists P^{'} \in S_P \mid P \subset P^{'} \text{ and } Freq(P) = Freq(P^{'})$$

To compute closed patterns, we first compute a simple graph structure, named *the Pattern Graph*. Each vertex of this graph is associated to a pattern of the scene and a directed edge $e = (v_1, v_2)$ links the two vertices if the pattern associated to v_2 is included in the pattern associated to v_1. Thanks to this data structure, it is easy to remove non-closed patterns with the procedure *FilterClosedPatterns* presented below.

For consistency, we add a vertex to this graph, of frequency 1 and that contains a pattern with the whole scene. As a consequence, all other patterns of the scene are included in this pattern. The method *FilterClosedPatterns* is so first apply to this root node.

Algorithm 5 FilterClosedPatterns

Require: A start node, *root*, which is associated to a frequent closed pattern.
  for all children n of *root* **do**
    if Freq(n) = Freq(root) **then**
      n is not associated to a closed patterns. Delete it
    else
      FilterClosedPatterns(n)
    end if
  end for

Once the non-closed patterns are removed, the Pattern Graph represents **inclusion-relation between all frequent closed patterns in the scene**.

8.3.5 Reducing the mappings set

For complex scenes, considering all pairs of congruent tiles may result in a large number of points in the transformation space. For example, in a 3D model of 500,000 polygons which decompose in 9000 tiles, the number of mappings is more than 13 millions[2]. This large number of points may disturb the clustering process but may also be a limit in terms of memory when treating massive 3D scenes. In this section, we describe a way of reducing the number of pair of tiles examined and so reducing the clutter in the transformation space.

Using neighborhood graph. The neighborhood graph G_N presented in Section 8.1 on page 101, encodes the spatial relations between pair of tiles.

The generation of all the mapping between congruent tiles can be thought of as a mapping of the vertices of G_N onto the *compatible vertices* of G_N.

Definition 19 *On the neighborhood graph, two vertices are said to be compatible if the tiles they represent are congruent.*

In the following, the mapping of the vertices of a graph G_1 onto a graph G_2 will be denoted $G_1 \rightsquigarrow G_2$. With this notation, the complete set of mapping between tiles is written $G_N \rightsquigarrow G_N$.

By simply looking at the topology of this graph, we are able to reduce the set of pairs examined by the algorithm. More precisely, let $CC = (CC_1, CC_2, ..CC_n)$ be the n connected components of G_N. The mappings that need to be computed can now be expressed as:

$$CC_i \rightsquigarrow CC_j \qquad i \in [1...n], j \in [i...n]$$

The main idea here is that for two tiles $\mathcal{T}_i$ and $\mathcal{T}_j$, the two mappings $\mathcal{T}_i \rightarrow \mathcal{T}_j$ and $\mathcal{T}_i \leftarrow \mathcal{T}_j$ need to be computed. In the case of two independent connected components, an order can be establish and hence reducing the number of mapping. For example, for two connected components CC_1 and CC_2, we simply need to compute the mappings $CC_1 \rightsquigarrow CC_2$ as the mapping $CC_2 \rightsquigarrow CC_1$ will contained the same information.

We also have to compute the mapping $CC_i \rightsquigarrow CC_i$, in order to discover the potential frequent patterns "inside" the graph CC_i.

This way of computing mapping between tiles is efficient if the neighborhood graph is formed of a consequent number of connected components, as the number of pairs examined decreases linearly with the number of components. However, most of the time the neighborhood graph of a scene is composed of only one connected component. This statement is pretty intuitive since the presence of multiple connected components is physically non plausible as depicted in Figure 8.18 on the following page

We now present a way of dividing the neighborhood graph in multiple components and of extending this process hierarchically, in order to reduce the number of pair of tiles that have to be examined.

[2]The number of mappings is also highly dependent on the number of congruent tiles and on the number of symmetries of each tile

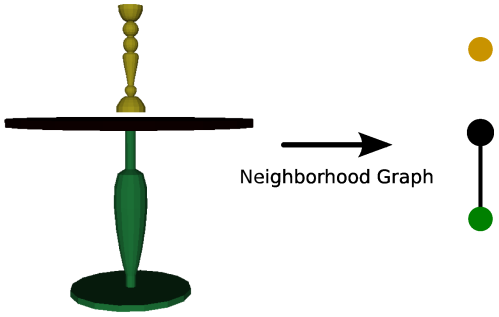


Figure 8.18: The presence of multiple connected components of the neighborhood graph is physically non plausible. In this case, the neighborhood graph has two connected components due to the "floating" vase.

Cutting the neighborhood graph: In order to cut the neighborhood graph, it is necessary to add some information to this graph. We choose to color edges based on the relative position/orientation of the two tiles associated to this edge. Two edges of the neighborhood graph will share the same label if the two objects formed by the two tiles of the edges are congruent (see Figure 8.19).

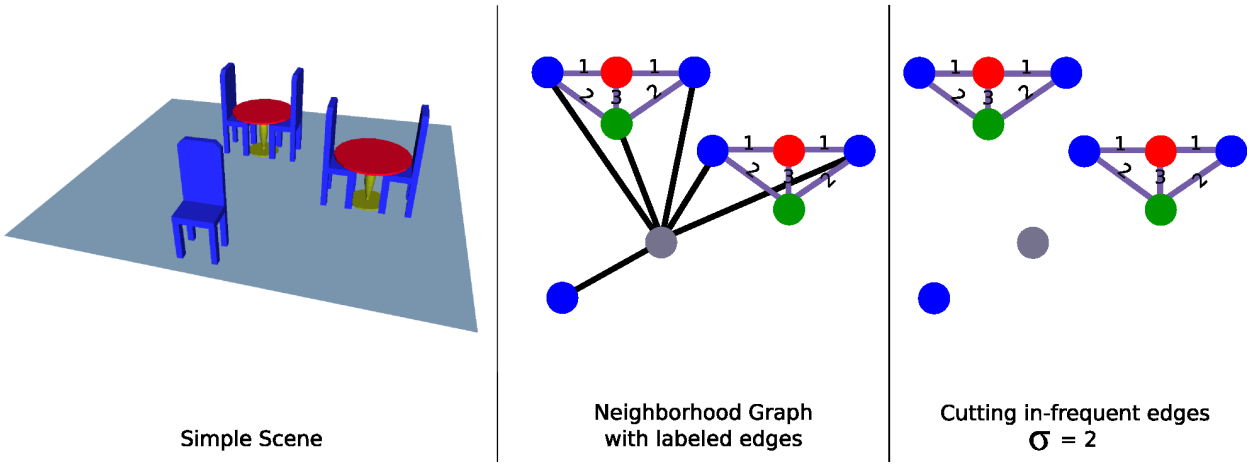


Figure 8.19: Left) A simple scene. Right) Its neighborhood graph. We affect to each edge a label with the property that two edges share the same label if the objects they define are congruent. In this example, we do not show the label of black edges as they appear only once in the scene.

Once this graph is obtained, we can discard edges which label occur less than the frequency threshold σ as they will not be used when forming frequent patterns. More precisely, we can discard an edge e from a graph G if:

$$\text{count}(\text{label}[e], G) < \sigma$$

While it is not theoretically proved that this method will form several connected components, it appears in practice that we obtain a large number of connected components. We

therefore use the mapping presented above to compute the pair of tiles that need to be encoded by our system.

To increase the efficiency of this method, we extend it by repeating this process on each created connected components hence resulting in a hierarchy of graphs. The algorithm is presented below and an illustrating hierarchy of graph is presented in Figure 8.20.

Algorithm 6 GraphDecomposition

Require: A undirected graph G=(V,E) and the frequency threshold σ
Ensure: Erase non-significant edges from G
  Erase all edges e for which $\text{count}(\text{label}[e], G) < \sigma$
  for all connected components CC_i of G **do**
    GraphDecomposition(CC_i, σ)
  end for

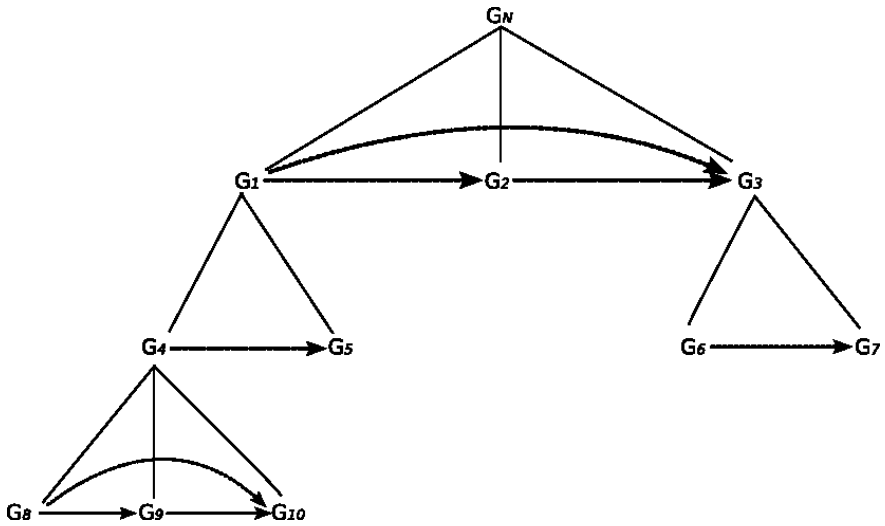


Figure 8.20: Example of a hierarchy obtained by our method for reducing the number of examined mappings.The children of a graph correspond to its connected components after removing of the non-frequent edges. The set of arrowed lines represent the set of mappings that need to be performed.

Results Using this strategy, we are able to greatly reduce the number of pair of tiles examined. These results are presented in Table 8.21 on the next page for some sample scenes. All the results presented are computed with a frequency threshold of two ($\sigma = 2$).

As can be seen, our technique greatly reduces the number of mapping and hence the number of clusters and the number of non-closed patterns. Moreover, computing this limited set of mapping is not time-consuming.

Scene Name	# Basic	# Reduced	Construction (secs)
piece	9,644	883	0.12
building	7,902	2,330	2.82
plane (complete)	25,255	13,115	7.9
LBXStudio	13,491,448	1,380,340	18.2

Figure 8.21: The number of mapping can be greatly reduced, up to a ratio of approximatively 10:1. The overhead time used to build this restricted mapping is not excessive as reported in the table.

8.3.6 Results

All the results presented here are computed with a frequency threshold of two *i.e.* $\sigma = 2$. The method described in the previous section is used to reduce the number of mapping that are proceed by the algorithm. The features of all the test models are presented in Figure 8.22:

Scene Name	# Polygons	# Tiles
Plane (part)	13,384	32
Piece	40,410	707
Building	81,451	485
Plane (complete)	182,184	480
LBXStudio	514,890	8,389
Powerplant	12,748,510	155,348

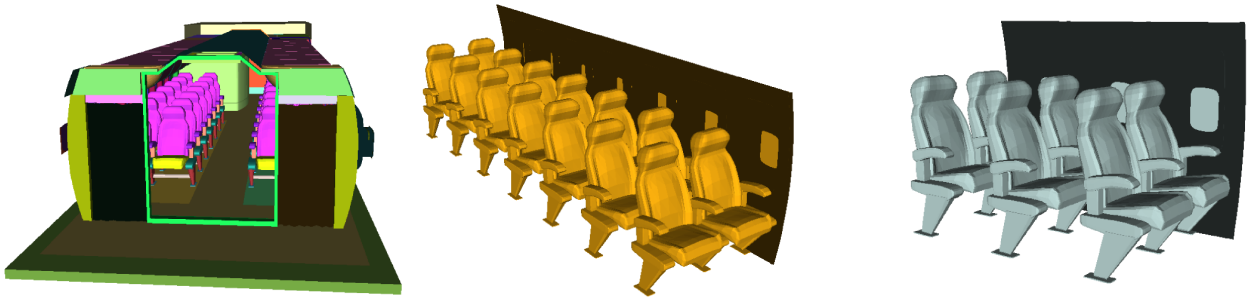
Figure 8.22: The scenes used to test the frequent pattern discovery using the symmetry-based approach

The results of our tests are presented in Figure 8.3.6.

Scene Name	# Frequent Patterns	# Closed Patterns	Runtime (s)
Plane(part)	7	2	0.45
Piece	780	170	12
Building	315	110	6.5
Plane (complete)	173	65	11.8
LBXStudio	1103	395	25mn
Powerplant	134,234	87,100	1h45

Figure 8.23: Results of the patterns discovery obtained with the symmetry-based approach. See the text below for an explanation of the results presented.

Some examples of the frequent closed patterns discovered in the LBXStudio are presented in Figure 8.26 on page 129. While the model is not very complex in terms of polygons and tiles, the time needed to discover frequent patterns may appears important. In our tests, we observe this in the scene with large number of congruent tiles. In this case, even with the enhancement method presented above, the number of mapping is still large which clutter the transformation space and hence increases the running time of the clustering method. In the scene LBXStudio, this complexity is due to the keys of the keyboard which are all detected

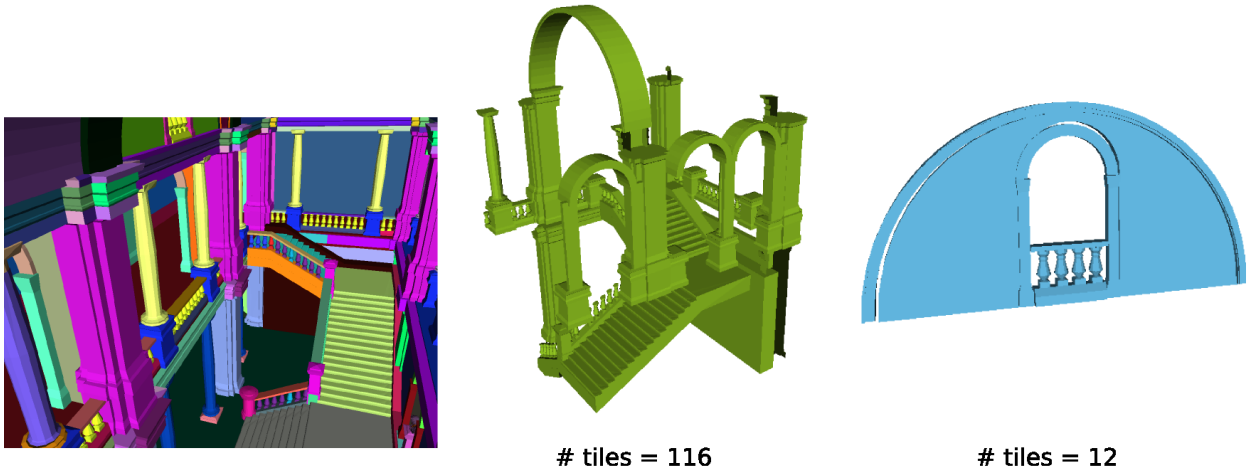


# tiles = 154
Freq = 2

# tiles = 66
Freq = 4

# Tiles	480
Frequency Threshold	2
Runtime (secs)	1.4
# Frequent Patterns	65

Figure 8.24: Frequent Patterns obtained on the Plane Model.



# tiles = 116
Freq = 2

# tiles = 12
Freq = 2

Figure 8.25: Frequent Patterns obtained on the Building Model.

as congruent (see the pattern “keyboard” in the Figure 8.26 on the next page). In order to form the pattern “keyboard”, all mapping between pairs of keys from each keyboard have to be tested which result in a large number of points in the transformation space.

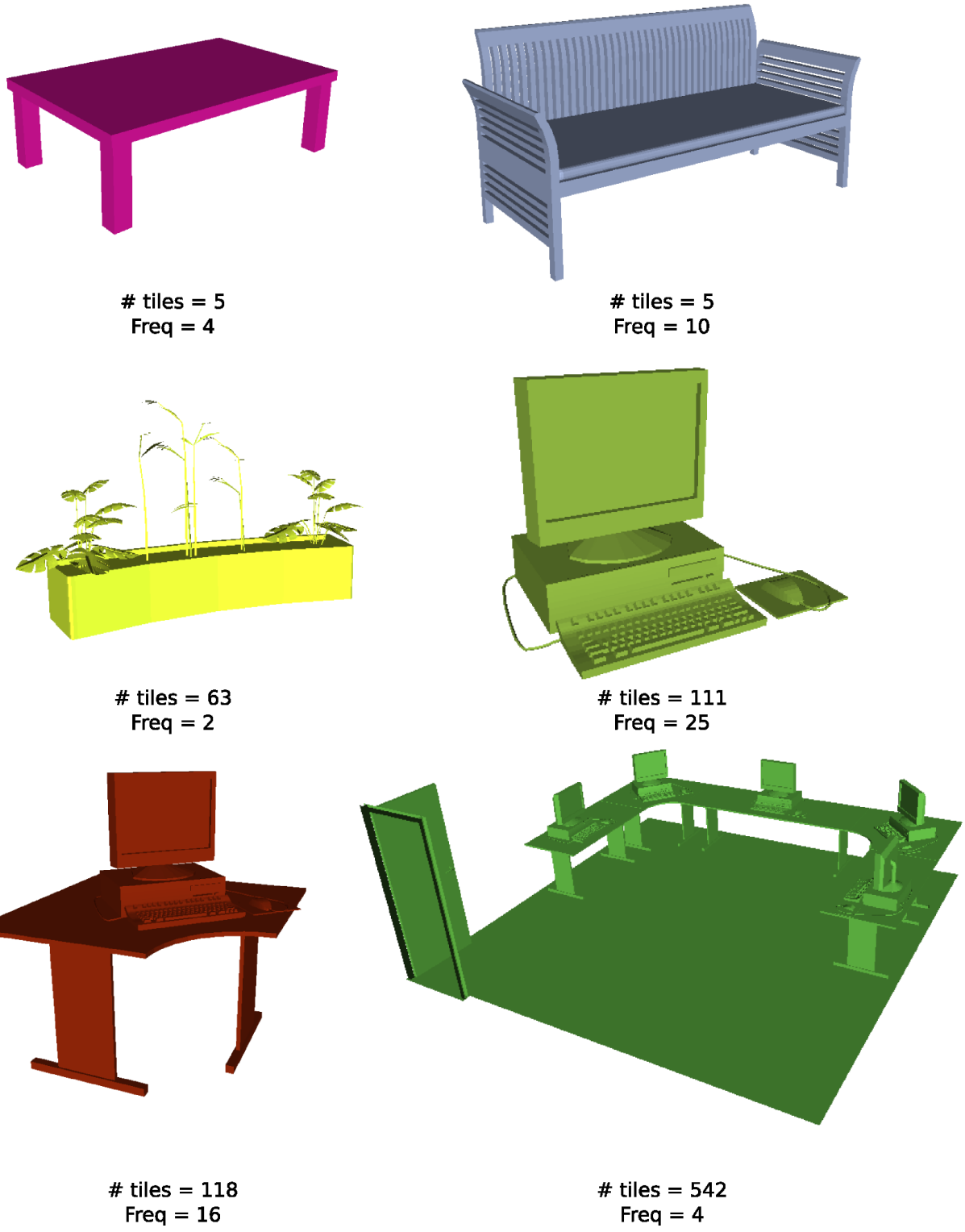


Figure 8.26: Frequent Patterns obtained on the LBX Studio.

9 Building a Hierarchy of Instances

The set of patterns that we have obtained in the previous chapter contains important geometric information and can be used, for example, in mesh editing in order to replicate changes made on a given selected pattern to its siblings. As presented in the related work, lighting simulation can also broadly take benefits of such instancing information. A well-known process to increase efficiency of instancing is to compute a hierarchy of instances and so decompose each instance into a set of sub-instances.

The first part of this chapter focuses on the data structure used to represent instancing information and presents a method to build it from the set of frequent patterns computed in the previous chapter. In the second part, we present in details a specific application scenario *i.e.* a method developed to compute a hierarchy of instances optimized for ray-tracing.

9.1 Representing Hierarchies of Instances

The goal of a hierarchy of instances is to *recursively* represents the sub-parts that formed each object of the scene. As the preferred way of designing a model is hierarchically, by composing objects or parts into more complex objects, a hierarchical data structure now became quite natural to used. In our work, we focus on a data structure named **H**ierarchical **A**ssembly **G**raph (HAG).

A HAG is a **D**irected **A**cyclic **G**raph which is a common way of representing a model designed hierarchically. Each node denotes an object and an arc denotes the sub-part relations between two objects. In general, any geometric or other parameters related to the model can be attached to the nodes or arcs. The most common example is to attach affine transformations to arcs to denote relative placement and scale of part and subparts [Braid, 1978].

An important observation regarding the HAG is that internal nodes do not represent in-

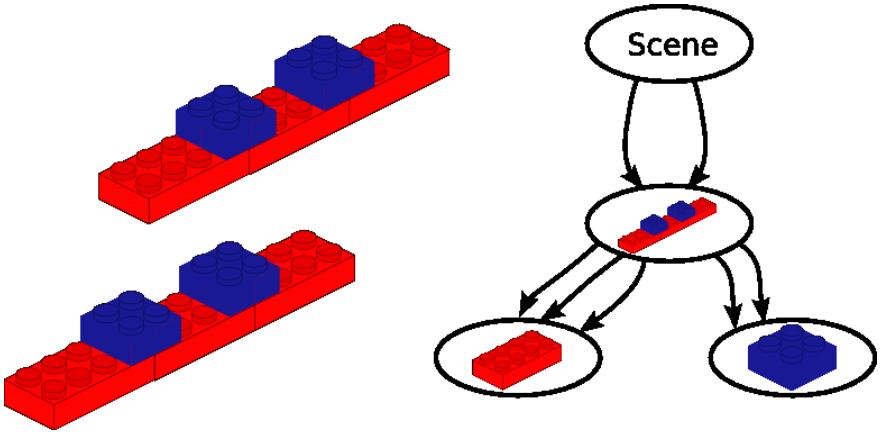


Figure 9.1: Example of a HAG of a very simple scene.

stances of objects in the final model but "generic objects" as represented in Figure 9.1[1]. With this formalism, an instance is defined as follows:

Definition 20 *An instance of an object O in a HAG is a path in that graph starting from the root node and ending in O.*

The HAG data structure has two notable advantages:

- *Space efficiency*: Information common to all instances generated from the same pattern is stored only once.
- *Fast Update*: Modification of parameters or information in the generic object is instantaneously reflected to all its instances in the graph.

Our strategy starts from the Pattern Graph previously defined and the final goal is to turn this graph into a HAG that respect some properties that will be identify later. In a first part, we explain our method to turn the Pattern Graph into a HAG. In a second part, we identify constraints that a hierarchy must respect and present of way of generating a hierarchy that respect such constraints.

9.2 From Frequent Patterns to HAG

To compute the HAG of the scene, we start from the Pattern Graph previously introduced. We briefly re-introduce its meaning in a first part, then present its limiting features and then our method to turn it into a HAG.

9.2.1 HAG Construction

The Pattern Graph is a Directed Acyclic Graph where each vertex is associated to a frequent closed pattern. A directed edge $e(v_1, v_2)$ linked two vertices if the pattern associated to v_2 is

[1] An immediate parallel to our work would translate those "generic objects" into patterns.

included in the pattern associated to v_1.

This simple graph does not have the structure of a HAG as it does not contain geometric transformation between all patterns. To transform the Pattern Graph, we need a *reference instance* for each pattern. All the transformations of the sub-instances will be computed with respect to this reference instance:

Definition 21 *Let P be a pattern and I^P its set of instances. The reference instance of P, I^P_{ref} is defined as the first instance of this set:*

$$I^P_{ref} = I^P_1$$

The algorithm proceeds by examining each edge of the Pattern Graph and computes the appropriate transforms. Let $e = (v_1, v_2)$ be the current examined edge and let P_1 and P_2 the two patterns that are respectively associated to v_1 and v_2. For this edge e, we:

- compute the set S_{P_2} of instances of P_2 that are included in $I^{P_1}_{ref}$,
- compute the appropriate transform between each element of S_{P_2} and $I^{P_1}_{ref}$ and add an edge between v_1 and v_2 that carries this information.

Due to the transitivity nature of the $\subset$ relation between patterns, the obtained HAG may be "dense" *i.e.* have a large number of edges (see Figure 9.2).

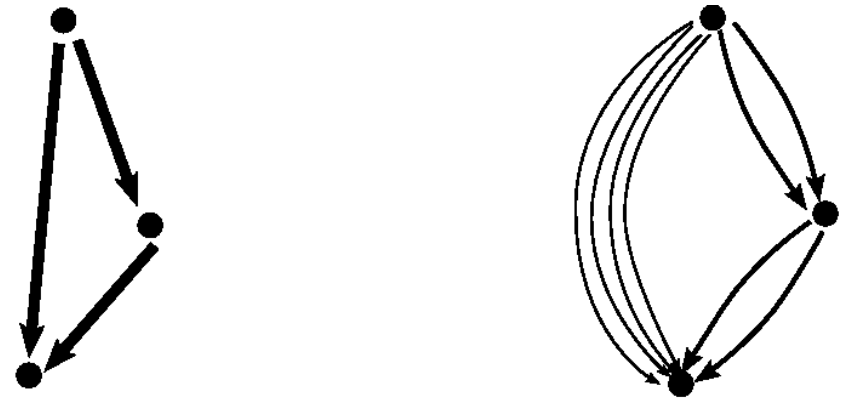


Figure 9.2: Due to the transitivity nature of the $\subset$ relation between patterns, the obtained HAG may be "dense" *i.e.* have a large number of edges.

A natural way of proceeding would be to compute the *transitive reduction* of the graph to eliminate such problems [Aho et al., 1972; La Poutré and van Leeuwen, 1988]:

Definition 22 *The transitive reduction of a directed graph G is the directed graph G' with the smallest number of edges such that for every path between vertices in G, G' has a path between those vertices.*

Such an approach is however not feasible as the transitivity information may sometimes be confounded with "real" instancing information (see Figure 9.3).

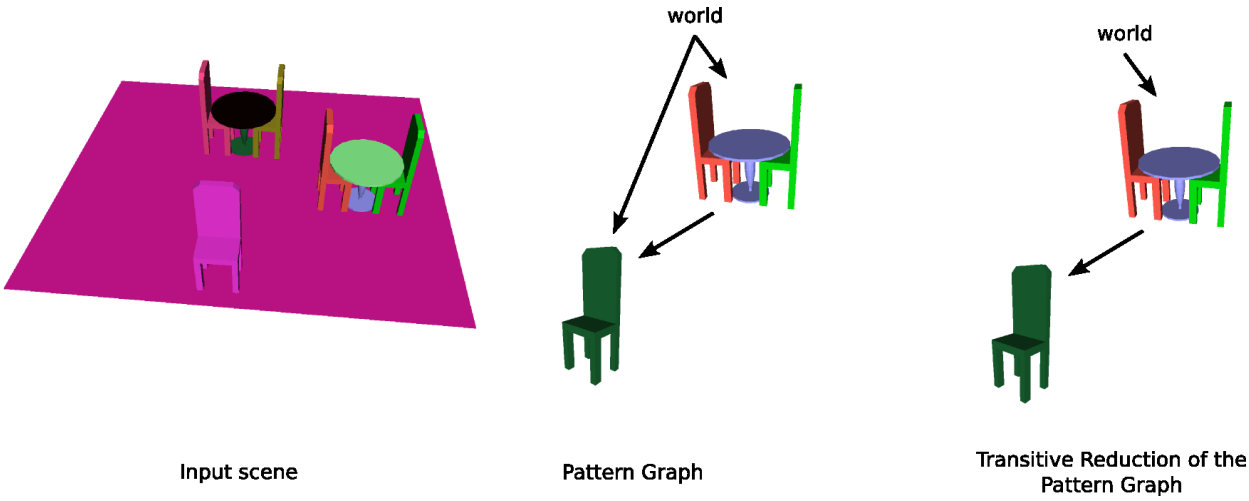


Figure 9.3: Computing the transitive reduction of the Pattern Graph may lead to miss some important information. In the figure, we lost the information that a chair is part of the tables but also part of the whole scene.

The right way of proceeding to compute a HAG without redundant information use this transitive reduction and is a two step process:

1. We first process edges that belong to the transitive reduction of the Pattern Graph.
2. In a second step, we iterate through the edges of the Pattern Graph that have not been treated yet, *i.e.* edges that do not belong to its transitive reduction.

For each edge, we mark the instances of the pattern as used to not use them multiple times (see Figure 9.4 on the next page).

9.2.2 Observations

The set of frequent closed patterns that we compute on the scene contains a large number of elements (see page 126). Computing a HAG by using all the frequent closed patterns results in a complex graph like the one presented in Figure 9.5 for the plane model.

Yet representing the hierarchy of patterns that are present in the scene, we qualify this graph of *unusable* mainly due to the overlapping that appear between instances. This overlapping is for example present in Figure 9.5 where the sub-instances of the pair of seats overlap.

Obtaining and using a Hierarchy of Instances with no overlap between the instances is *essential for three reasons*:

- When usually rendering a model, all children of a node are rendered once. An overlapping among the children will hence result in multiple rendering of the overlapping part.

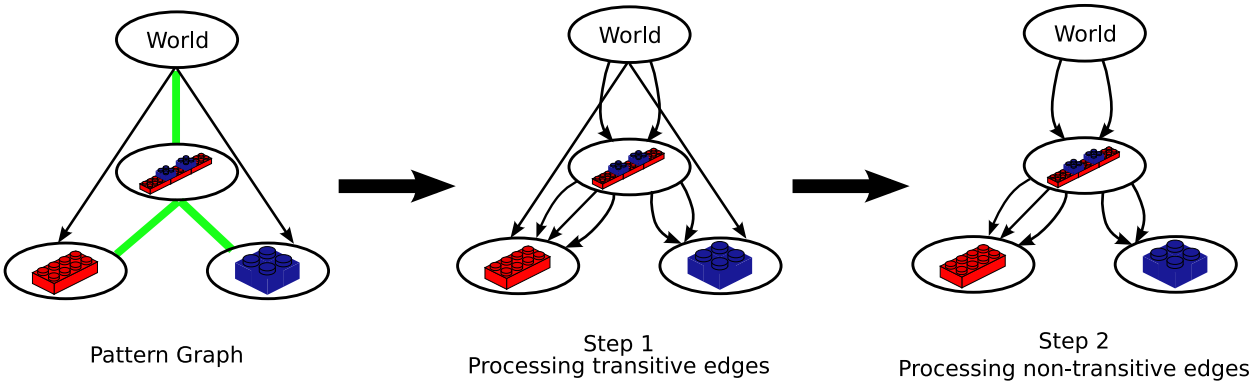


Figure 9.4: The creation of a HAG, starting from the Pattern Graph. Left) The initial Pattern Graph. The transitive edges, *i.e.* edges of the transitive reduction of the Pattern Graph are drawn in green. Middle) The transitive edges are treated simultaneously. Right) The non-transitive edges are treated which give the final HAG. As each red pattern is contained in the biggest blue-and-red patterns, there is no edges between the red pattern and the world pattern.

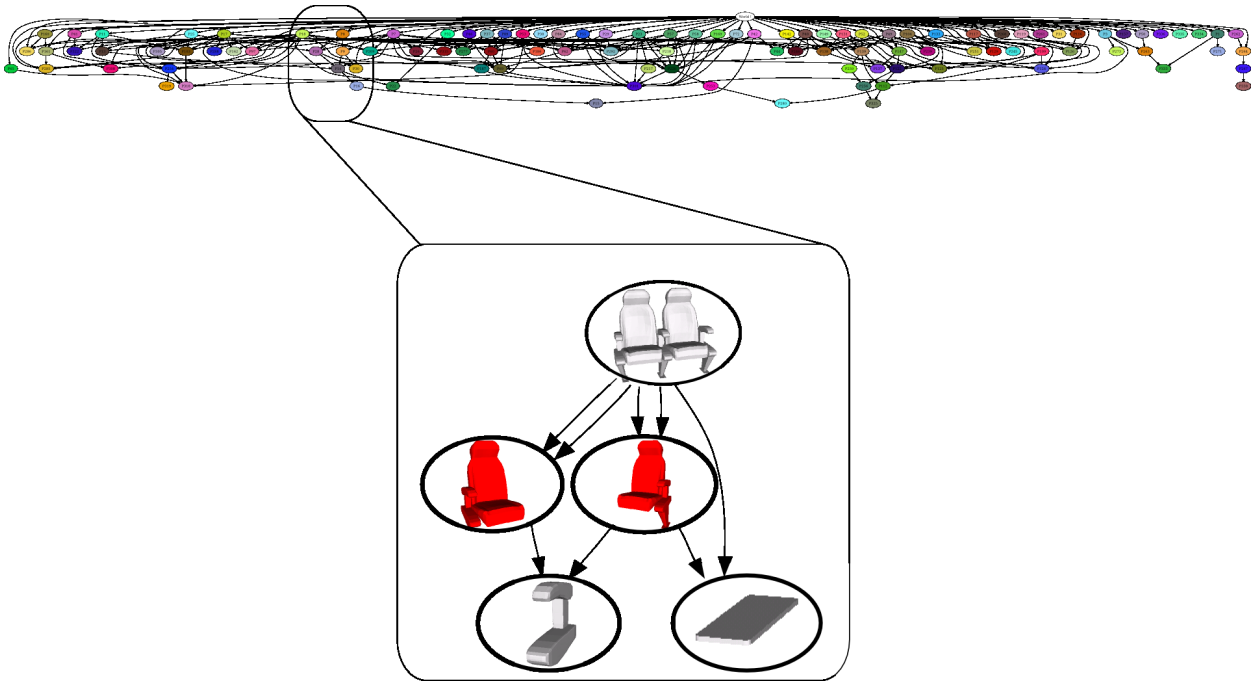


Figure 9.5: Top) the hierarchy of the plane model with overlapping instances. Bottom) A close view of a part of the graph where overlapping occurs. The patterns draw in red overlap.

- This graph is not "usable" for memory reduction *e.g.* reducing the memory cost of ray-tracing, as some parts of the scene are replicated multiple times.
- The overlapping made the editing of the model very difficult.

More formally, we define the non-overlapping property as:

Property 9 *Non-Overlapping Property: for each node in a HAG, its immediate sub-instances form a mutually disjoint set.*

Except in very simple scenes, like those presented in Figure 9.1 on page 132 and Figure 9.6 on the next page, this property of non-overlapping is not respected.

Deriving a hierarchy that respect this non-overlapping property implies that some choices must be made *i.e.* keep some sub-instances and discard some others. We present in the next section our original method developed to obtain a usable hierarchy *i.e.* a hierarchy that respects the non-overlapping property based on the domain of application of the hierarchy.

9.3 Deriving a usable hierarchy of instances

As introduced above, the non-overlapping property implies that if we want that our HAG respect it, some choices must be made, *i.e.* keep some sub-instances and discard some others.

This choice is directly related to the utilization that will be made of this hierarchy. Depending on the application *i.e.* rendering, scene editing,... the hierarchy that we want to obtain is different. For example, in the case of ray-tracing, the hierarchy must compress the scene so that the memory requirements will be lower than without instancing. Conversely, when editing the scene, the optimal solution might be a deep hierarchy of instances or a hierarchy that maximizes the number of different instances.

Whatever the utilization of the hierarchy of instances, the choice of picking a set of instances and discard some other can be formalize as a problem of maximizing the sum of weight associated to each instance with respect to the non-overlapping constraint. The weights that will be affected to instances will of course depend of the utilization of the hierarchy. This problem is known as the *Maximum Weighted Independent Set*.

In a first paragraph, we briefly present the key aspects of the problem of Maximum Weighted Independent Set. In the two last paragraphs, we present a scenario of utilization which computes a hierarchy of instances optimized for ray-tracing.

9.3.1 The Maximum Weighted Independent Set

The Maximum Weighted Independent Set may be seen as an extension of the algorithm of Maximum Independent Set previously used to compute the frequency of a pattern. In the weighted case, each node of a graph is associated to a weight and the goal is to maximize the sum of weights that may picked in this graph providing that two adjacent vertices must not be part of the solution. More formally, if we define a weight vector $w = (w_1, w_2, \dots w_n)$ that represents the weights attached to each vertex of the graph:

Definition 23 *Computing a Maximum Weighted Independent Set of a graph $G = (V,E)$ translates into solving the following optimization problem:*

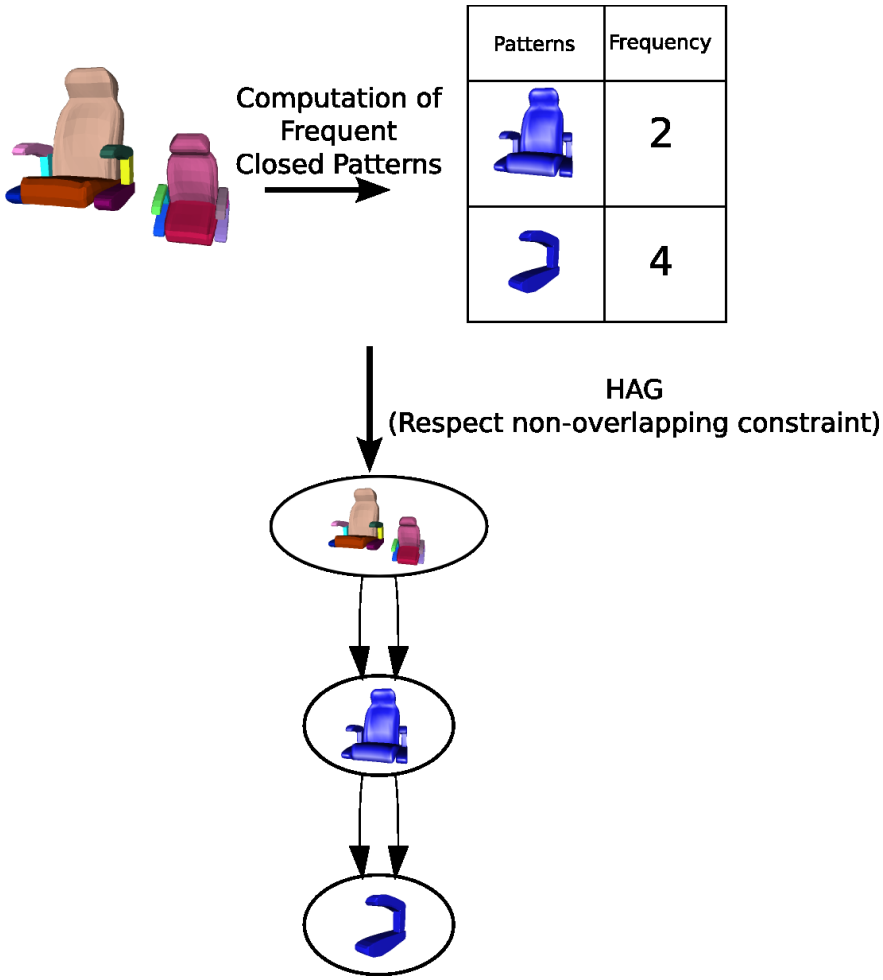


Figure 9.6: A very simple scene where the HAG respect the non-overlapping constraint. In this case, we do not need post-process to obtain the final HAG.

$$\begin{array}{ll} \text{maximize} & \sum w_i x_i, \\ \text{subject to} & x_i^2 - x_i = 0 \quad \text{for all } i \in V, \text{ (i)} \\ & x_i x_j = 0 \quad \text{for all } (i,j) \in E, \text{ (ii)} \end{array}$$

where V is the vertex set of G and $E \subseteq V \times V$ is the edge set of G.

Notation 1 *The weight of the Maximum Weighted Independent Set of a graph G with weight vector w is noted $\Omega(G,w)$.*

As for the unweighted case, computing the Maximum Weighted Independent Set is an $NP-$complete problem. We present below one approximation algorithm to solve it:

Approximation algorithm for $\Omega(G,\mathbf{w})$ For a vertex set X, let $\mathrm{w}(X)$ denote the sum of the weights of the vertices in X. Let $\mathrm{N}_G(v)$ denote the set of vertices adjacent to vertex v in G.

Definition 24 *For a vertex v, we define the weighted degree $d_w(v,G)$ of v in G as follows:*

$$d_w(v,G) = \frac{w(N_G(v))}{w_v}$$

The approximation algorithm to compute $\Omega(G,\mathrm{w})$ is to:

1. Select the vertex of minimum weighted degree
2. delete this vertex and all of its neighbors in G.
3. Apply 1. for the remaining subgraph until its becomes empty.

Readers interested in more details concerning this approximation algorithm can refer to [Kako et al., 2005].

9.3.2 Hierarchy of instances optimized for Ray-Tracing

Ray-tracing is a rendering technique that naturally allows instancing. Complex models can be ray traced with little memory since only the geometric representation of original objects must be kept in memory.

In this case, hierarchical instancing is used to reduce the part of the model that is loaded in memory. If we take our formalism, this is equivalent to load a single instance of a pattern and transformation matrices (or any per-instance attributes) for each other instance.

Picking the hierarchy of instances that is best suited for ray-tracing is equivalent to reduce the storage cost of the scene and hence of each pattern of the scene with respect to the non-overlapping constraint.

Let's take an example on the simple scene presented in Figure 9.7 on the facing page.

We introduce the storage cost of a pattern P as $\mathrm{C}(P)$. We also introduce the number of polygons needed to represent it as $\mathrm{N}_{poly}(P)$. As it may happen that instances of a single pattern may be formed of a different number of polygons due to the tessellation of the scene, we compute the $N_{poly}(P)$ as the average number of polygons of its instances.

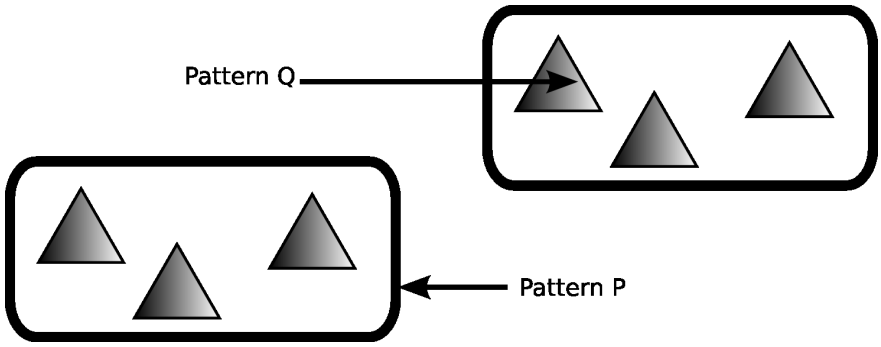


Figure 9.7: A simple test scene where each instance of the pattern P contains three instances of the pattern Q

In Figure 9.7, without hierarchical instancing, the cost of the pattern P is:

$$\mathrm{C}(P) = \mathrm{N}_{poly}(P)$$

If we use the information that each instance of the pattern P is formed of three instances of the pattern Q (the triangle), we now have:

$$\mathrm{C}(P) = \underbrace{\mathrm{N}_{poly}(P) - 3\mathrm{N}_{poly}(Q)}_{\text{number of polygons non instantiated}} + \underbrace{\mathrm{C}(Q) + 3\mathrm{C}_{ref}}_{\text{cost of instancing}}$$

The new storage cost of the pattern P is now the sum of two expressions:

1. The number of polygons non instantiated which is equal to the original number of polygons that formed the pattern P, minus the number of polygons that composes the pattern Q.

2. The cost of instancing *i.e.* the storage cost of a single instance of Q plus the cost of three references to the original pattern. In this term, C_{ref} represents the per-instance data that are used. These data are independent of the instance and represents most of the time transformation matrix, color or texture information.

For any pattern P, we have:

$$\mathrm{C}(P) \leq \mathrm{N}_{poly}(P)$$

with equality if the pattern P has no sub-instances.

Computing the minimum storage cost is equivalent to minimize the cost of every patterns in the hierarchy and is an $NP-$complete problem due to the non-overlapping constraint. We present in the next section our approximation algorithm used to solve this problem.

Approximation Algorithm

In practice, the storage cost of a pattern is computed as the storage cost of its reference instance I^P_{ref}. Using the HAG that we have computed below, we have access to the sub-instances

of I^P_{ref} that we represent as an overlap graph G^P_O. Each sub-instance of I^P_{ref} is associated to a vertex of G^P_O and an edge relies two vertices if the associated instances overlap. We associate to each vertex i of the graph the number of polygons of the associated instance n_i as well as its storage cost c_i.

We use a greedy approach which takes at each step the best potential solution. To take into account the cost of the instantiation, we define for each vertex i a value $\alpha(i)$:

$$\forall i \quad \alpha(i) = n_i - \text{cost}(i)$$

where:

$$\text{cost}(i) = \begin{cases} c_i & \text{if the associated pattern is picked for the first time} \\ \text{C}_{ref} & \text{otherwise} \end{cases}$$

The cost of a vertex is so dependent of the vertex that has been previously picked as part of the solution. To compute an approximate solution to $\text{C}(P)$, we use the following algorithm:

1. $\text{C}(P) \leftarrow \text{N}_{poly}(P)$
2. $i \leftarrow$ vertex of maximum α
3. Remove the vertex i and all its neighbors from the graph and update $\text{C}(P)$: $\text{C}(P) \leftarrow \text{C}(P) - \alpha(i)$
4. Apply 2. for the remaining subgraph until its becomes empty.

After choosing the instances that minimize the weight of all the patterns of the scene, it may happen that some patterns may be *single instantiated*, which means that they are instantiated in the scene only one time. In the HAG, an equivalent definition is:

Definition 25 *A pattern P_u associated to a node u in the HAG is said to be single instantiated if:*

$$\textit{in-degree}(u) = 1$$

At first sight, single-instantiated patterns may be removed from the hierarchies of instances as theses patterns do not participate to the storage cost reduction of the scene (see Figure 9.8 on the next page).

As the method used to compute the optimal cost for a pattern is only an approximation algorithm, it may happen that removing single-instantiated patterns change the cost of the patterns is was linked to as shown in Figure 9.9 on page 142. In this example, the green pattern is instantiated only one time in the hierarchy of instances optimized for ray-tracing. This pattern is so removed and the cost of the grey pattern is recomputed which results in a new representation with a lower cost (which is the optimal representation).

As the process of computing the weight of a pattern is a recursive one (the weight of a pattern is computed from the weights of its children), this has to be done carefully.

We chose a iterative algorithm that iterate through each vertex of the HAG, find a single-instantiated node, remove it from the graph and update the weight of each pattern it was linked to. As the weight of a pattern is computed in a bottom-up fashion in the HAG, we have to discover vertices in the HAG in a bottom-up way. As the HAG is a Directed Acyclic Graph, this is equivalent to discover vertex based on the topological sort of the HAG:

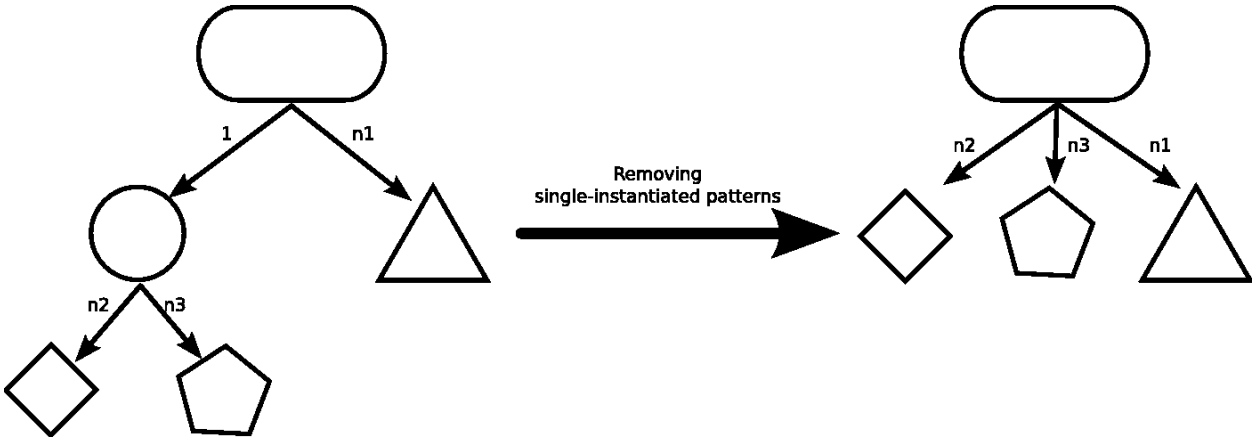


Figure 9.8: Removing single-instantiated patterns. In this figure, the circle pattern is single-instantiated and can be removed from the HAG as it does not bring any interesting instancing information.

Definition 26 *A topological sort of a DAG is a linear ordering of its nodes where x comes before y if there's a directed path from x to y in the DAG.*

Property 10 *Each node comes before all nodes to which it has edges.*

Property 11 *Every DAG has at least one topological sort, and may have many.*

By iterating through vertices based on the **reverse topological order**, we are sure to update the weight of a pattern before its parent nodes.

The complete approximation algorithm is described below:

Algorithm 7 OptimalHierarchy: Approximation algorithm for ray-tracing

Require: A HAG G
Ensure: Compute a hierarchy of instances optimized for ray-tracing
  Sort vertices of G by **reverse topological order**
  $v \leftarrow$ first vertex associated to single-instantiated pattern
  if v do not exist **then**
    return { no single-instantiated pattern left.}
  end if
  Mark parent patterns of v to be updated
  Remove v from G.
  ComputeWeights(G)
  OptimalHierarchy(G)

In practice, the topological order do not need to be compute at each call of the *Optimal-Hierarchy* method.

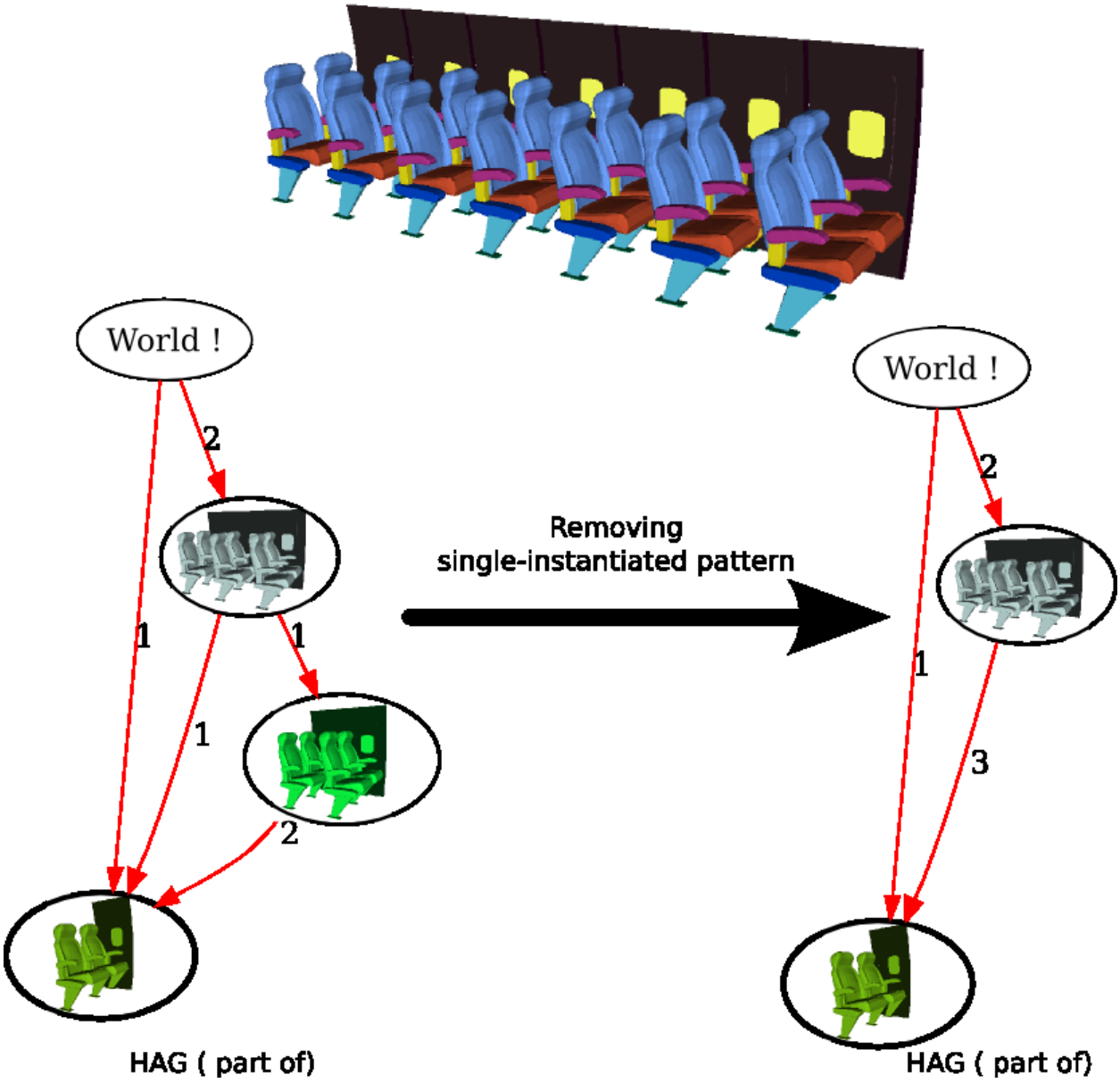


Figure 9.9: Due to the approximation algorithm, removing a single-instantiated pattern may changed the cost of the patterns it was linked to. In this example, the green pattern is single instantiated in the hierarchy of instances optimized for ray-tracing (left). After removing it from the hierarchy (right), we recompute the cost of the grey pattern which is now lower (and is the optimal representation).

Results

A way of quantifying the quality of a hierarchy of instances for ray-tracing is to evaluate the compression rate. These compression rates are obtained by computing the storage cost of the scene with and without instancing and the results are presented in Figure 9.10. Note that compression rates shown on this table are computed using geometry information only *i.e.* neither texturing nor material information are taken into account.

Model	Piece	Building	Plane	LBXStudio	Powerplant
# polygons	40,410	81,451	182,184	514,890	12,748,510
# tiles	707	485	480	8,389	155,348
Compression time (secs)	0.2	0.8	1.4	7	51
Compression rate	1 : 2.7	1 : 8.3	1 : 3.5	1 : 4.5	1:5.2

Figure 9.10: Examples of compression rates obtained using a hierarchy of instances to represent the scene. The *compression time* represents the time needed to compute the hierarchy.

Some examples of hierarchies are presented in Figure 9.12 and Figure 9.11.

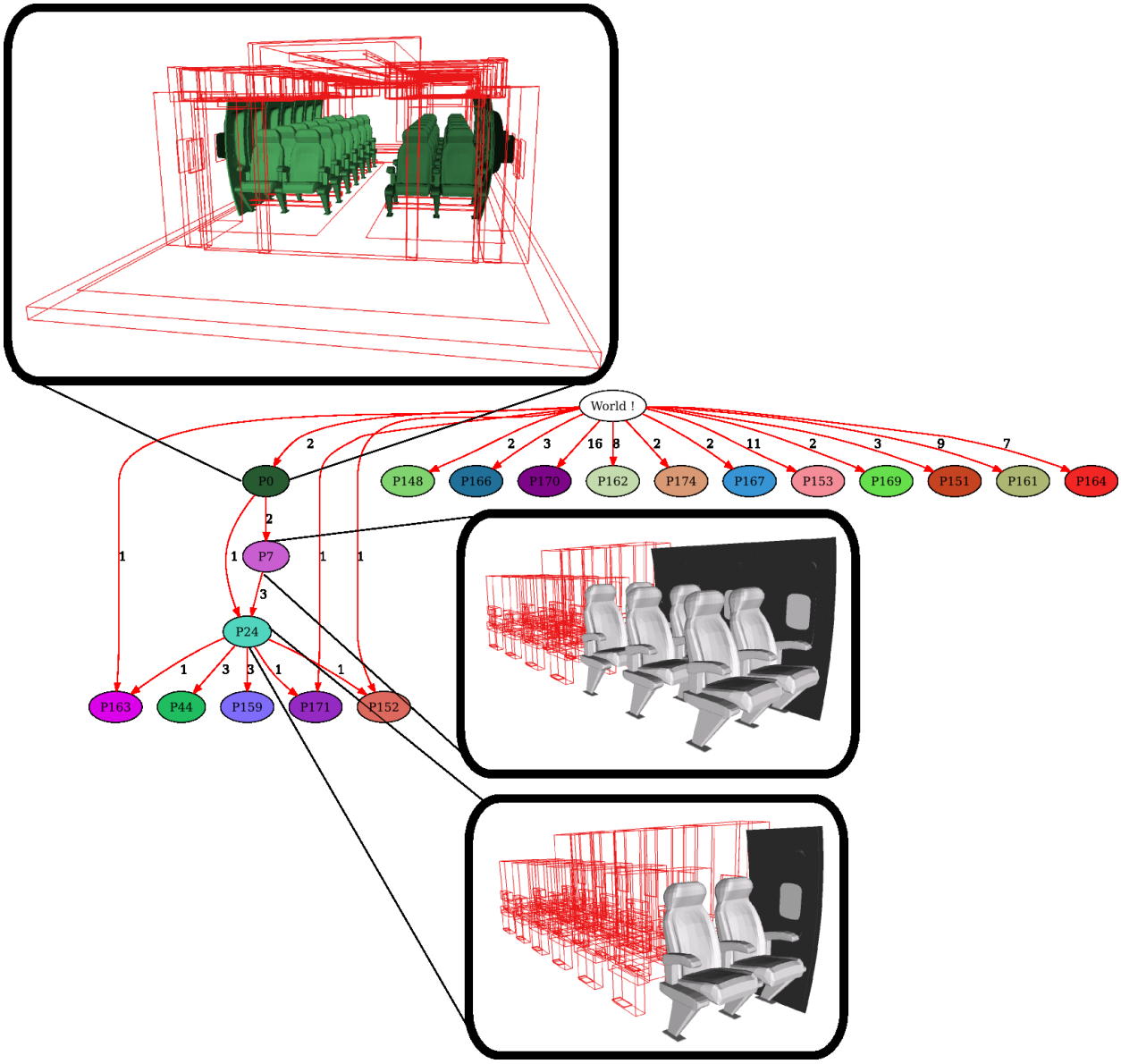


Figure 9.11: Hierarchy of instances for the Plane Model optimized for ray-tracing

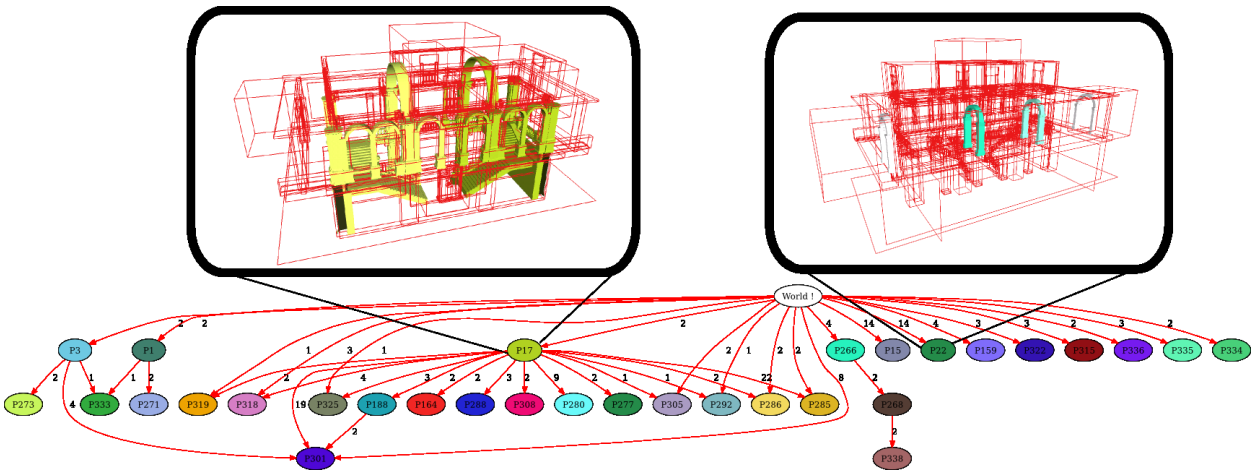


Figure 9.12: Hierarchy of instances for the Building Model optimized for ray-tracing

10 Conclusions

This chapter concludes this thesis with a summary of principal contributions and some thoughts about extensions of this work and future research.

This dissertation presents our work done for structuring geometry based on the information of symmetry and instantiation. In the first part of this conclusion, we sum up the main contributions of this thesis. In a second part, we first present the most promising future work that may be done in both symmetry detection and instantiation and end with some final thoughts about structural information at the semantic level.

Summary of Contributions

Detection of Symmetries in 3D Shapes

We have presented an algorithm to automatically retrieve symmetries for geometric shapes and models. Our algorithm efficiently and accurately retrieves all symmetries from a given model, independently from its tessellation.

We use a new tool, the *generalized moment* functions, to identify candidates for symmetries. The validity of each candidate is checked against the original shape using a geometric measure. Generalized moments are not computed directly: instead, we compute their spherical harmonic coefficients using an integral expression. Having an analytical expression for the generalized moment functions and their gradients, our algorithm finds potential symmetry axes quickly and with good accuracy.

Compared to recent method that search both local and global symmetries, we restrict our search to global symmetry *i.e.* symmetry applied to the center of mass of the objects. For composite shapes assembled from simpler elements, we have presented an extension of this algorithm that works by first identifying the symmetries of each element, then sets of congruent elements. We then use this information to iteratively build the symmetries of the composite shape. This extension is able to handle complex shapes with better accuracy, since

it pushes the accuracy issues down to the scale of the tiles.

The work developed in this part has been published in [Martinet et al., 2006].

Hierarchical Instancing of Geometry

We have presented a two-steps algorithm to compute a hierarchy of instances, starting from the structure of the object level *i.e.* structure of the geometry in tiles, associated with symmetry information. The two steps are summarized below:

Step 1. Frequent pattern discovery: We present, and experiment classical way of discovering frequent patterns in the graph data-mining community, and experiment it for finding frequent geometric patterns. We show that this method cannot be used in practice, especially for low frequency threshold.

In order to compute the set of frequent geometric patterns for every values of the frequency threshold, we then present an original method based on the detection of local symmetries in the scene. We encode the transformation (translation and rotation) between each congruent tile in the scene in a vector space named *the transformation space* and form instances of patterns by forming clusters in this space.

For couple of tiles with continuous symmetries, the set of transformation between them is no longer a single point but a continuous set of points which either form a sphere or a curve in the transformation space. We propose a way of computing an analytical expression of this set, based on quaternion tools, and use it to efficiently form instances which contain tiles with continuous symmetries.

Step 2. Computing hierarchy of patterns: We identify the criteria of non-overlapping that hierarchies must respect in order for it to be usable. This criteria makes the problem of computing hierarchies of instances an NP-complete problem. We propose a way of deriving a hierarchy that respect such constraints and present an application scenario to compute a hierarchy of instances suitable for ray-tracing.

Future Work and Thoughts

Symmetry Detection

The method we propose in this thesis **is the first work published in this area** that compute discrete and continuous symmetries of 3D shapes. In the last two years, a large number of methods have been published in major conferences or journals such as Eurographics or Siggraph. The whole set of papers that compute symmetries are described in the Chapter 3 of this dissertation. The state of the art in symmetry detection is now important and cover every aspect of symmetry: global or local, discrete and/or continuous, approximate and/or exact.

From this statement, it seems that the future work in this area is quite limited. We however think that a promising working avenue concerns the way of automatically make a given model more symmetric. This will potentially increase the efficiency of database retrieval method.

Instancing of Geometry

Once again, the method proposed in this thesis **is the first work that compute a hierarchy of instances from a totally unstructured scene**. We have presented an application scenario based on ray-tracing and we are currently working on a generic system to derive a hierarchy of instances suitable for scene editing.

Deriving a hierarchy of instances suitable for scene-editing is a complicated task as it does not exist a single way of representing instances in a scenes. For example, one might want to obtain the deepest hierarchy, the hierarchy with the largest number or instances, etc... We are currently working on a generic model that associate a parametric weight to each pattern of the HAG and use it to compute an "optimal" hierarchy as done in the application scenario for ray-tracing. The parameters of this weight will control the way of representing each pattern for example by given more importance to deepest representation.

As explained in Chapter 9, the problem of obtaining a hierarchy with no overlap is an $NP-$complete problem. In order to evaluate the quality of our approximation algorithm, the computation of the approximation ratio of our greedy approach for computing hierarchy optimized ray-tracing, presented in Section 9.3.2, is also a part of our future work.

Semantic of 3D geometry

Structuring the scene at the semantic level seems to be the most challenging work to be done in the future. A complete semantic representation of a scene captures the functions, the characteristics and relationships between each object in the scene. We think that a method that aimed to structuring a scene have to cope with the extraction of semantically meaningful features from geometry, scale transitions, geometric and topological consistency. This information is very rich and may be used for example to abstract the data by "understanding" its contents.

In architectural scenes, this semantic information allows to isolate and identify walls, furniture, desktops, etc... as well as the relative placement of objects in the scene e.g the phone is *on the* table. We believe that such knowledge might be helpful for example for a visualization system that want to maintain an interactive frame rate (*e.g.*, a constant ten frames per second) as in [Funkhouser and Séquin, 1993]. Semantic information may be used in such systems to give more importance to some objects based on their semantic and hence derive a order of rendering of objects *i.e.* rendering the table before the phone.

As three-dimensional environments, man-made or natural, always exhibit some structure at different scales, we claim that the work developed in this thesis is also a first step toward the accessibility of semantic information in 3D scenes.

A

Symmetry: Proof of theorems

Appendix: Proofs

of theorem 8 let A be an isometric transform which lets a shape $\mathcal{S}$ globally unchanged. We have:

$$\begin{aligned}
\forall \boldsymbol{\omega} \quad \mathcal{M}^{2p}(A\boldsymbol{\omega}) &= \int_{\boldsymbol{s}\in\mathcal{S}} \|\boldsymbol{s}\times A\boldsymbol{\omega}\|^{2p}\,\mathrm{d}\boldsymbol{s} \\
&= \int_{\boldsymbol{t}\in A^{-1}\mathcal{S}} \|A\boldsymbol{t}\times A\boldsymbol{\omega}\|^{2p}|\det A|\,\mathrm{d}\boldsymbol{t} \\
&= \int_{\boldsymbol{t}\in A^{-1}\mathcal{S}} \|\boldsymbol{t}\times\boldsymbol{\omega}\|^{2p}\,\mathrm{d}\boldsymbol{t} \\
&= \mathcal{M}^{2p}(\boldsymbol{\omega})
\end{aligned}$$

At line 2 we change variables and integrate over the surface transformed by A^{-1}. At line 3, an isometric transform being a unit transform, its determinant is ± 1 and thus vanishes. The cross product is also left unchanged by applying an isometric transform to each of its terms. Line 4: because $A\mathcal{S}=\mathcal{S}$ we also have $\mathcal{S}=A^{-1}S$. The isometric transform A is thus also a symmetry of the $\mathcal{M}^{2p}$ moment functions.

Let A be an isometric transform with axis $\boldsymbol{v}$, and suppose that A is a symmetry of $\mathcal{M}^{2p}$. Let $\boldsymbol{d_v}$ be the direction of steepest descent of function $\mathcal{M}^{2p}$ around direction $\boldsymbol{v}$. Because A is a symmetry of $\mathcal{M}^{2p}$ we have:

$$\boldsymbol{d}_{A\boldsymbol{v}} = A\boldsymbol{d_v} = \boldsymbol{d_v} \tag{A.1}$$

If A is a rotation this is impossible because $\boldsymbol{d_v}\perp\boldsymbol{v}$. Moreover, for all directions ω we have $\mathcal{M}^{2p}(-\omega)=\mathcal{M}^{2p}(\omega)$ and thus:

$$\boldsymbol{d}_{-\boldsymbol{v}} = -\boldsymbol{d_v} \tag{A.2}$$

So, if A is a symmetry, we have $A\boldsymbol{v}=-\boldsymbol{v}$. From Equations A.1 and A.2 it now comes that $\boldsymbol{d_v}=-\boldsymbol{d_v}$, which is impossible.

In both cases, $\mathcal{M}^{2p}$ can not have a direction of steepest descent in direction v. Because $\mathcal{M}^{2p}$ is infinitely derivable, this implies that $\nabla\mathcal{M}^{2p}(v) = 0$

of Property 4 Let $\mathcal{S}$ and $\mathcal{R}$ be two shapes, identical up to an isometric transform. Let J be an isometric transform such that $J\mathcal{S} = \mathcal{R}$. Let T be the translation of vector $-u_{\mathcal{S}}$ with $u_{\mathcal{S}} = g_{\mathcal{S}} - g$, $g_{\mathcal{S}}$ being the center of mass of $\mathcal{S}$, and g the origin of the coordinate system into which J is applied.

- Let $A \in G_S$ be a symmetry of $\mathcal{S}$ such that $Au_{\mathcal{S}} = u_{\mathcal{S}}$. We have $AT\mathcal{S} = T\mathcal{S}$ (the symmetry A operates in the coordinate system centered on $g_{\mathcal{S}}$). Let $K = JT^{-1}AT$. Then

$$\begin{aligned} K\mathcal{S} &= JT^{-1}AT\mathcal{S} \\ &= JT^{-1}T\mathcal{S} \\ &= J\mathcal{S} \\ &= \mathcal{R} \end{aligned} \quad \text{and} \quad \begin{aligned} K\mathbf{0} &= JT^{-1}AT\mathbf{0} \\ &= JT^{-1}A(-u_{\mathcal{S}}) \\ &= JT^{-1}(-u_{\mathcal{S}}) \\ &= J\mathbf{0} = 0 \end{aligned}$$

  By construction K is a rigid transform and conserves distances. It maps the origin onto itself. K is thus an isometric transform. Furthermore, K maps $\mathcal{S}$ to $\mathcal{R}$.

- Let K be an isometric transform such that $K\mathcal{S} = \mathcal{R}$. Let us choose $A = TJ^{-1}KT^{-1}$. This choice leads to $K = JT^{-1}AT$. Moreover:

$$\begin{aligned} AT\mathcal{S} &= TJ^{-1}KT^{-1}T\mathcal{S} \\ &= TJ^{-1}KS \\ &= T\mathcal{S} \end{aligned} \quad \text{and} \quad \begin{aligned} Au_{\mathcal{S}} &= TJ^{-1}KT^{-1}u_{\mathcal{S}} \\ &= TJ^{-1}K2u_{\mathcal{S}} \\ &= T2u_{\mathcal{S}} = u_{\mathcal{S}} \end{aligned}$$

  and

$$\begin{aligned} A\mathbf{0} &= TJ^{-1}KT^{-1}\mathbf{0} \\ &= TJ^{-1}Ku_{\mathcal{S}} \\ &= TJ^{-1}(g_{\mathcal{R}} - g) \\ &= T(-u_{\mathcal{S}}) \\ &= \mathbf{0} \end{aligned}$$

  By construction A is affine and conserves distances. It maps $\mathbf{0}$ onto $\mathbf{0}$. A is thus an isometric transform. A is also a symmetry of $\mathcal{S}$ which verifies $Au_{\mathcal{S}} = u_{\mathcal{S}}$.

- The set of isometries which map $\mathcal{S}$ to $\mathcal{R}$ is therefore the set of functions K of the form $K = JT^{-1}AT$, where $A \in G_{\mathcal{S}}$ is a symmetry of $\mathcal{S}$ such that $A(g - g_{\mathcal{S}}) = (g - g_{\mathcal{S}})$.

of Equation 4.3 We compute the decomposition of function $\theta \longmapsto \sin^{2p}\theta$ into zonal spherical harmonics: we proove that this decomposition is finite, and give the values of the coeficients.

By definition [Hobson, 1931], we have:

$$\begin{aligned} Y_L^0(\theta,\varphi) &= \sqrt{\frac{2L+1}{4\pi}}P_L(\cos\theta) \\ &= \sqrt{\frac{2L+1}{4\pi}}\frac{(-1)^L}{2^L L!}\frac{d^L}{dx^L}\left[(1-x^2)^L\right](\cos\theta) \end{aligned}$$

Where P_k is the Legendre polynomial of order k. Because the set of Legendre polynomials $P_0, P_1, \ldots, P_n$ is a basis for polynomials of order not greater than n, function $\theta \longmapsto \sin^{2p}\theta = (1 -$

$\cos^2\theta)^p$ can be uniquely expressed in terms of $P_L(\cos\theta)$. The decomposition of $\theta \longmapsto \sin^{2p}\theta$ is thus finite and has terms up to Y^0_{2p} at most.

Let's compute them explicitely:

$$\begin{aligned}
\frac{d^L}{dx^L}\left[(1-x^2)^L\right] &= \frac{d^L}{dx^L}\sum_{k=0}^{L}(-1)^{L-k}x^{2L-2k}C_L^k \\
&= (-1)^L\frac{d^L}{dx^L}\sum_{k=0}^{L}(-1)^k x^{2k}C_L^k \\
&= \sum_{L\leq 2k\leq 2L}(-1)^{L+k}C_L^k 2k(2k-1)...(2k-L+1)x^{2k-L} \\
&= \sum_{L\leq 2k\leq 2L}(-1)^{L+k}C_L^k\frac{(2k)!}{(2k-L)!}x^{2k-L}
\end{aligned}$$

So:

$$Y_L^0(\theta,\varphi) = \sqrt{\frac{2L+1}{4\pi}}\sum_{L\leq 2k\leq 2L}\frac{(-1)^k}{2^L L!}C_L^k\frac{(2k)!}{(2k-L)!}\cos^{2k-L}\theta$$

The coeficients of the decomposition we are interested in are thus:

$$\int_{\theta=0}^{\pi}\int_{\varphi=0}^{2\pi}Y_L^0(\theta,\varphi)\sin^{2p}\theta\sin\theta d\theta d\varphi = 2\pi\sqrt{\frac{2L+1}{4\pi}}\sum_{L\leq 2k\leq 2L}\frac{(-1)^k}{2^L L!}C_L^k\frac{(2k)!}{(2k-L)!}I^p_{2k-L} \tag{A.3}$$

where integrals I^p_m are defined by:

$$I^p_m = \int_{\theta=0}^{\pi}\sin^{2p+1}\theta\cos^m\theta d\theta$$

First, $I^p_m = 0$ for all odd m, because the integrand in anti-symetric around $x = \pi/2$. Then, if m is even:

$$\begin{aligned}
I^p_m &= \underbrace{\left[\frac{1}{2p+2}\sin^{2p+2}\theta\cos^{m-1}\theta\right]_0^{\pi}}_{0} + \frac{m-1}{2p+2}\int_0^{\pi}\sin^{2p+3}\theta\cos^{m-2}\theta d\theta \\
&= \frac{m-1}{2p+2}I^{2p+3}_{m-2} \\
&= \frac{(m-1)(m-3)...1}{(2p+2)(2p+4)...(2p+m)}\int_o^{\pi}\sin^{2p+m+1}\theta d\theta
\end{aligned}$$

Let J_q be the integral defined by

$$J_q = \int_0^{\pi}\sin^{2q+1}\theta d\theta$$

We have

$$\begin{aligned}
J_q &= \underbrace{\left[-\cos\theta\sin^{2q}\theta\right]_0^{\pi}}_{0} + 2q\int_0^{\pi}\cos^2\theta\sin^{2q-1}\theta d\theta \\
&= 2qJ_{q-1} - 2qJ_q
\end{aligned}$$

Therefore

$$\begin{aligned} J_q &= \frac{2q}{2q+1} J_{q-1} \\ &= \frac{2q(2q-2)...2}{(2q+1)(2q-1)...3} J_0 \\ &= \frac{2^{2q+1}(q!)^2}{(2q+1)!} \end{aligned}$$

For m even, we can take $m = 2r$ and $q = p+r$; we get:

$$\begin{aligned} I_{2r}^p &= \frac{(2r)!p!}{2^r r! 2^r (p+r)!} \frac{2^{2p+2r+1}(p+r)!^2}{(2p+2r+1)!} \\ &= \frac{(2r)!p!2^{2p+1}(p+r)!}{r!(2p+2r+1)!} \end{aligned} \tag{A.4}$$

From Equation A.5 we deduce that, for L odd,

$$\int\int Y_L^0(\theta,\varphi)\sin^{2p}\theta\sin\theta d\theta d\varphi = 0$$

For L even, we set $L = 2l$. Using $r = k-l$ to match Equation A.6 in Equation A.5, we get:

$$\begin{aligned} S_p^l &= \int\int Y_{2l}^0(\theta,\varphi)\sin^{2p}\theta\sin\theta d\theta d\varphi \\ &= 2\pi\sqrt{\frac{4l+1}{4\pi}} \sum_{2l\le 2k\le 4l} \frac{(-1)^k}{2^{2l}(2l)!} C_{2l}^k \frac{(2k)!}{(2k-2l)!} \frac{(2k-2l)!p!2^{2p+1}(p+k-l)!}{(k-l)!(2p+2k-2l+1)!} \\ &= \frac{\sqrt{(4l+1)\pi}}{2^{2l}(2l)!} \sum_{l\le k\le 2l} (-1)^k C_{2l}^k \frac{(2k)!p!2^{2p+1}(p+k-l)!}{(k-l)!(2p+2k-2l+1)!} \\ &= \frac{\sqrt{(4l+1)\pi}}{2^{2l}} \sum_{l\le k\le 2l} (-1)^k \frac{(2k)!p!2^{2p+1}(p+k-l)!}{k!(2l-k)!(k-l)!(2p+2k-2l+1)!} \end{aligned}$$

In this document we compute the decomposition of function $\theta \longmapsto \sin^{2p}\theta$ into zonal spherical harmonics. We prove that the decomposition is finite, and give the values of the associated coeficients:

$$\sin^{2p}\theta = \sum_{l=0}^{p} S_p^l Y_{2l}^0(\theta,.)$$

with:

$$S_p^l = \frac{\sqrt{(4l+1)\pi}}{2^{2l}} \sum_{k=l}^{2l} (-1)^k \frac{2^{2p+1}p!(2k)!(p+k-l)!}{(2(p+k-l)+1)!(k-l)!k!(2l-k)!}$$

Demonstration

We have

$$Y_L^0(\theta,\varphi) = \sqrt{\frac{2L+1}{4\pi}} P_L(\cos\theta) = \sqrt{\frac{2L+1}{4\pi}} \frac{(-1)^L}{2^L L!} \frac{d^L}{dx^L}\left[(1-x^2)^L\right](\cos\theta)$$

Because the set of Legendre polynomials $P_0, P_1, ..., P_n$ is a basis for polynomials of order not greater than n, function $\theta \longmapsto \sin^{2p}\theta = (1-\cos^2\theta)^p$ can be uniquely expressed in terms of $P_L(\cos\theta)$. The decomposition of $\theta \longmapsto \sin^{2p}\theta$ is thus finite and has terms up to Y^0_{2p} at most.

Let's compute them explicitely:

$$\begin{aligned}
\frac{d^L}{dx^L}\left[(1-x^2)^L\right] &= \frac{d^L}{dx^L}\sum_{k=0}^{L}(-1)^{L-k}x^{2L-2k}C_L^k \\
&= (-1)^L\frac{d^L}{dx^L}\sum_{k=0}^{L}(-1)^k x^{2k}C_L^k \\
&= \sum_{L\le 2k\le 2L}(-1)^{L+k}C_L^k 2k(2k-1)...(2k-L+1)x^{2k-L} \\
&= \sum_{L\le 2k\le 2L}(-1)^{L+k}C_L^k\frac{(2k)!}{(2k-L)!}x^{2k-L}
\end{aligned}$$

So:

$$Y_L^0(\theta,\varphi) = \sqrt{\frac{2L+1}{4\pi}}\sum_{L\le 2k\le 2L}\frac{(-1)^k}{2^L L!}C_L^k\frac{(2k)!}{(2k-L)!}\cos^{2k-L}\theta$$

The coeficients of the decomposition we are interested in are thus:

$$\int_{\theta=0}^{\pi}\int_{\varphi=0}^{2\pi}Y_L^0(\theta,\varphi)\sin^{2p}\theta\sin\theta d\theta d\varphi = 2\pi\sqrt{\frac{2L+1}{4\pi}}\sum_{L\le 2k\le 2L}\frac{(-1)^k}{2^L L!}C_L^k\frac{(2k)!}{(2k-L)!}I^p_{2k-L} \tag{A.5}$$

where integrals I^p_m are defined by:

$$I^p_m = \int_{\theta=0}^{\pi}\sin^{2p+1}\theta\cos^m\theta d\theta$$

First, $I^p_m = 0$ for all odd m, because the integrand in anti-symetric around $x = \pi/2$. Then, if m is even:

$$\begin{aligned}
I^p_m &= \underbrace{\left[\frac{1}{2p+2}\sin^{2p+2}\theta\cos^{m-1}\theta\right]_0^{\pi}}_{0} + \frac{m-1}{2p+2}\int_0^{\pi}\sin^{2p+3}\theta\cos^{m-2}\theta d\theta \\
&= \frac{m-1}{2p+2}I^{2p+3}_{m-2} \\
&= \frac{(m-1)(m-3)...1}{(2p+2)(2p+4)...(2p+m)}\int_o^{\pi}\sin^{2p+m+1}\theta d\theta
\end{aligned}$$

Let J_q be the integral defined by

$$J_q = \int_0^{\pi}\sin^{2q+1}\theta d\theta$$

We have

$$\begin{aligned}
J_q &= \underbrace{\left[-\cos\theta\sin^{2q}\theta\right]_0^{\pi}}_{0} + 2q\int_0^{\pi}\cos^2\theta\sin^{2q-1}\theta d\theta \\
&= 2qJ_{q-1} - 2qJ_q
\end{aligned}$$

Therefore

$$\begin{aligned} J_q &= \frac{2q}{2q+1} J_{q-1} \\ &= \frac{2q(2q-2)...2}{(2q+1)(2q-1)...3} J_0 \\ &= \frac{2^{2q+1}(q!)^2}{(2q+1)!} \end{aligned}$$

For m even, we can take $m = 2r$ and $q = p+r$; we get:

$$\begin{aligned} I_{2r}^p &= \frac{(2r)!p!}{2^r r! 2^r (p+r)!} \frac{2^{2p+2r+1}(p+r)!^2}{(2p+2r+1)!} \\ &= \frac{(2r)!p!2^{2p+1}(p+r)!}{r!(2p+2r+1)!} \end{aligned} \tag{A.6}$$

From Equation A.5 we deduce that, for L odd,

$$\int\int Y_L^0(\theta,\varphi)\sin^{2p}\theta\sin\theta d\theta d\varphi = 0$$

For L even, we set $L = 2l$. Using $r = k-l$ to match Equation A.6 in Equation A.5, we get:

$$\begin{aligned} S_p^l &= \int\int Y_{2l}^0(\theta,\varphi)\sin^{2p}\theta\sin\theta d\theta d\varphi \\ &= 2\pi\sqrt{\frac{4l+1}{4\pi}} \sum_{2l\le 2k\le 4l} \frac{(-1)^k}{2^{2l}(2l)!} C_{2l}^k \frac{(2k)!}{(2k-2l)!} \frac{(2k-2l)!p!2^{2p+1}(p+k-l)!}{(k-l)!(2p+2k-2l+1)!} \\ &= \frac{\sqrt{(4l+1)\pi}}{2^{2l}(2l)!} \sum_{l\le k\le 2l} (-1)^k C_{2l}^k \frac{(2k)!p!2^{2p+1}(p+k-l)!}{(k-l)!(2p+2k-2l+1)!} \\ &= \frac{\sqrt{(4l+1)\pi}}{2^{2l}} \sum_{l\le k\le 2l} (-1)^k \frac{(2k)!p!2^{2p+1}(p+k-l)!}{k!(2l-k)!(k-l)!(2p+2k-2l+1)!} \end{aligned}$$

B

Proof of the Classification Theorem

Proof of the **Classification Theorem**:

Theorem 10 *Every isometry is one of the following: the identity, a translation, a rotation, a reflection, or a glide reflection.*

The proof use the **triangle theorem:**

Theorem 11 *If the two triangles ABC and A'B'C' are congruent, then there exists a unique isometry F such that F(A) = A', F(B) = B', and F(C) = C'.*

Let T be an isometry. Let ABC be a triangle, with A,B and C **noncollinear**, and let A'B'C' be the image of ABC by the transformation T.

To show that T is one of the types identity, translation, rotation, reflection, or glide reflection, we just have to show that there's an isometry F of one of these types such that F also takes the triangle ABC to A'B'C' *i.e.* F(A) = A', F(B) = B', and F(C) = C'. Then, by the uniqueness part of the triangle theorem, T = F.

The proof is organized into four cases:

Case 1: *T fixes A, B, and C i.e. A'= A', B = B', and C = C'. Then T is the identity.*

If I is the identity, then I(A) = A, I(B) = B, and I(C) = C, so T = I by uniqueness.

Case 2: *T fixes exactly two of the points A, B, C. Then T is a reflection.*

Suppose that A = A' and B = B', but C is not C'. Since T is an isometry, AC = AC' and BC = BC'. In other words, C' lies on the circle c1 with center A through the point C, and C' lies on the circle c2 with center B through the point C (see Figure B.1 on the following page).

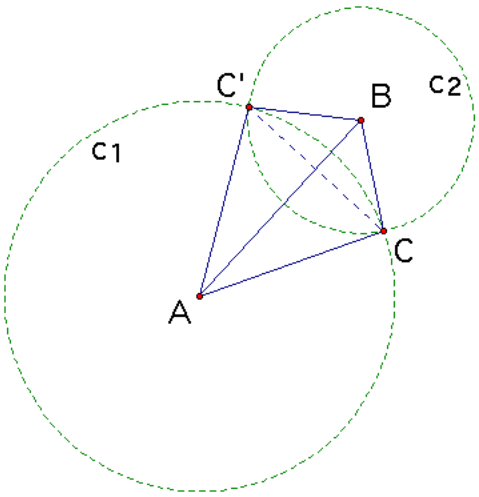


Figure B.1: Case 2: We suppose that T fixes A and B but not C.

The intersection of two different circles must be empty, one point, or two points. The point C lies on both circles c1 and c2. If C were the only point common to c1 and c2, then we would have AC + BC = AB, so C would lie between A and B, and C would be collinear with A and B, contrary to hypothesis.

Therefore the intersection of c1 and c2 is two points, C and C'. Since A is equidistant from C and C', A lies on the perpendicular bisector of CC'. Since B is equidistant from C and C', B also lies on the perpendicular bisector of CC'. Thus AB is the perpendicular bisector of CC'. Thus if F is the reflection with mirror AB, we have F(A) = A, F(B) = B, and F(C) = C'.

Thus T = F by uniqueness.

Case 3: T fixes exactly one of the points A, B, C.

Case 3a: *If T is orientation reversing, then T is a reflection.*

Suppose A = A', but B is not B', and C is not C' Since T is orientation reversing, the proper angle measures of the angles CAB and C'AB' have opposite signs. Since T is an isometry, CA = C'A and BA = B'A (see Figure B.2 on the next page).

Let D be the midpoint of the segment BB'. Since the triangle BAB' is isosceles, AD is the perpendicular bisector of BB', and AD is the bisector of the angle BAB'. Since the angles CAB and B'AC have equal proper angle measures, AD is also the bisector of the angle CAC'. Since the triangle CAC' is isosceles, AD is the perpendicular bisector of CC'. Thus if F is the reflection with mirror AD, we have F(A) = A, F(B) = B', and F(C) = C'. Therefore T = F by uniqueness.

Case 3b: *If T is orientation preserving, then T is a rotation.*

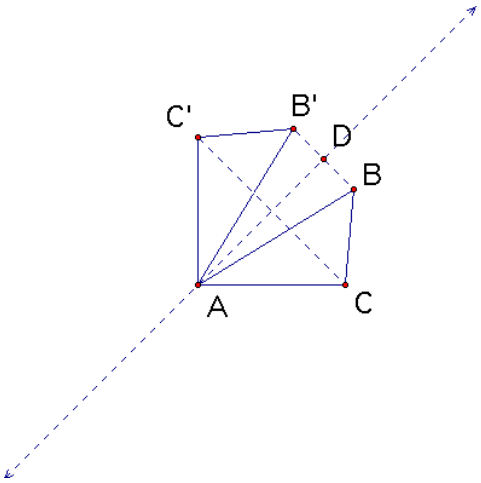


Figure B.2: Case 3a: We suppose that T is orientation reversing and only fixes A.

Again suppose A = A', but B is not B', and C is not C'. Since T is orientation preserving, the proper angle measures of the angles CAB and C'AB' are equal. Since T is an isometry, CA = C'A and BA = B'A. Thus B and B' lie on the same circle c1 with center A, and C and C' lie on the same circle c2 with center A (see Figure B.3)

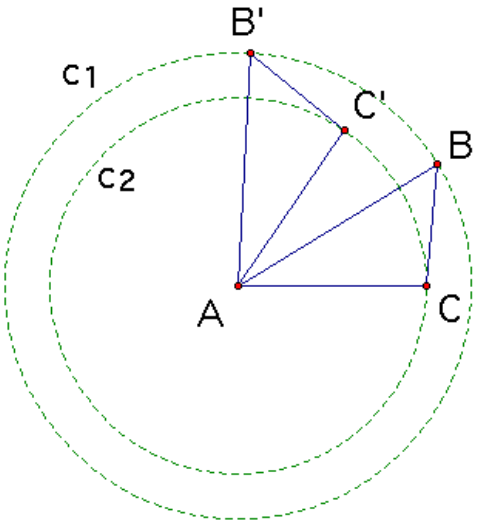


Figure B.3: Case 3b: We suppose that T is orientation preserving and only fixes A.

The angles CAB and C'AB' have equal proper angle measure: $\widehat{CAB} = \widehat{C'AB'}$. Now $\widehat{CAC'} = \widehat{CAB} + \widehat{BAC'}$ and $\widehat{BAB'} = \widehat{BAC'} + \widehat{C'AB'}$. So $\widehat{CAC'} = \widehat{BAB'}$. Thus if R is the rotation with center A and angle CAC', we have R(A) = A, R(B) = B', and R(C) = C'. Therefore T = R by uniqueness.

Case 4: T fixes none of the points A, B, C.

Case 4a: *If T is orientation preserving, then T is a translation or a rotation.*

Since A, B, C are noncollinear, one of the points B, C does not lie on the line AA'. Suppose B does not lie on AA'. Thus the lines AB and A'B' are different.

1. Suppose the lines AB and A'B' are parallel. Then the vectors AB and A'B' are either equal (i.e. ABB'A' is a parallelogram) or opposite (i.e. ABA'B' is a parallelogram). If ABB'A is a parallelogram then AA' is parallel to BB'. Since T is orientation preserving, if ABB'A' is a parallelogram, then so is ACC'A' as the point C cannot lie on both AA' and BB'. We may assume that C does not lie on AA'.

   So if S is the translation with vector AA', we have S(A) = A', S(B) = B', and S(C) = C'. Therefore T = S by uniqueness.

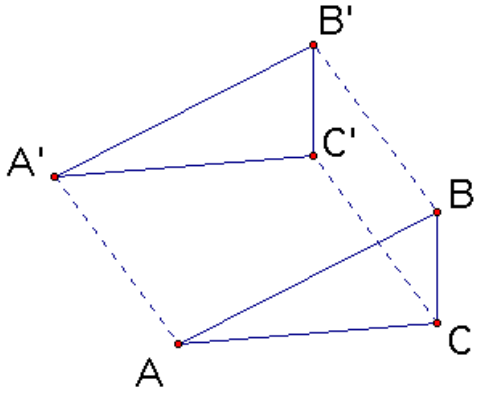


Figure B.4: Classification Theorem: Case 4a where the lines AB and A'B' are parallel

   On the other hand, if ABA'B' is a parallelogram (and C does not lie on BB') then CBC'B' is a parallelogram. In this case, the segments AA' and BB' bisect each other, since they are the diagonals of the parallelogram ABA'B'. For the same reason the segments BB' and CC' bisect each other. Let O be the common midpoint of the three segments AA', BB', CC'. [Special cases: If C lies on BB' but not on AA' then CAC'A' is a parallelogram, and the three segments AA', BB'. CC' still have a common midpoint O. If C lies on both AA' and BB' then C = C' and we let O = C.] If R is the rotation with center O and angle 180 degrees, then R(A) = A', R(B) = B', and R(C) = C', so T = R by uniqueness.

2. Suppose the lines AB and A'B' are not parallel. Let L be the perpendicular bisector of AA', and let M be the perpendicular bisector of BB'. Let O be the intersection of L and M.

   [If L and M are parallel, then AA' and BB' are parallel, so ABB'A' is an isosceles trapezoid, and hence L = M. In this case we let O be the intersection of AB and A'B'.] The (yellow) triangles AOB and A'OB' are congruent, by the side-side-side congruence theorem. Therefore the angles AOB and A'OB' have the same proper angle measure. Thus the angles AOA' and BOB' have the same proper angle measure. Let R be the rota-

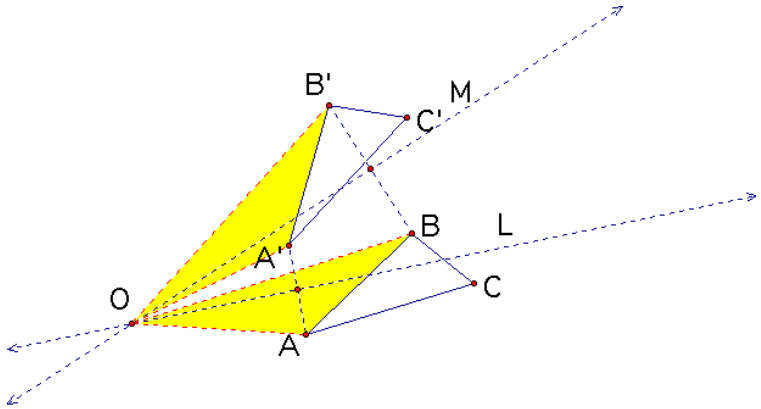


Figure B.5: Classification Theorem: Case 4a where the lines AB and A'B' are not parallel

tion with center O and angle AOA'. Then R(A) = A' and R(B) = B'. Since T and R are orientation preserving, R(C) = C'. Therefore T = R by uniqueness.

Case 4b: *If T is orientation reversing, then T is a reflection or a glide reflection.*

Note that if two angles have corresponding sides parallel, and their proper angle measures are negatives of each other, then both angles must be right angles. Thus, since T is orientation reversing, all three sides of the triangle ABC cannot be parallel to the corresponding sides of triangle A'B'C'. Suppose that AB is not parallel to A'B'.

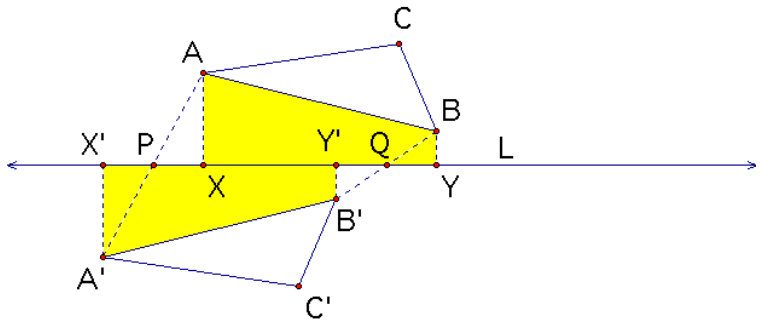


Figure B.6: Classification Theorem: Case 4b

Let P be the midpoint of segment AA' and let Q be the midpoint of segment BB'. Let L be the line PQ. [The points P and Q are not equal, for if they were equal then the triangles APB and A'PB' would be congruent, and AB would be parallel to A'B'.] Let X be the foot of the perpendicular from A to L, let X' be the foot of the perpendicular from A' to L, let Y be the foot of the perpendicular from B to L, and let Y' be the foot of the perpendicular from B' to L. Triangles AXP and A'X'P are congruent, so AX = A'X'. (If X = P then X' = P, and again AX =

AX'.) Similarly, BY = BY'. Since AB = A'B', it follows that the (yellow) trapezoids AXYB and A'X'Y'B' are congruent. Therefore XY = X'Y', so XX' = YY'.

Let G be the glide reflection with mirror L and vector XX'. (If X = X' then G is the reflection with mirror L.) Then G(A) = A' and G(B) = B' since XX' = YY'. Since T and G are orientation reversing, G(C) = C'. Therefore T = G by uniqueness.

This completes the proof of the classification theorem.

C

Mean Shift Clustering

In this appendix, we present the *mean-shift*, a method for seeking the mode of a point set which has been initially proposed in 1975 by [Fukunaga and Hostetler, 1975]. We present here a simplified version of the mean-shift where the metric of the feature space *i.e.* the point cloud is Euclidean, which is not the case in our work (see page 113). We refer once again to [Fukunaga and Hostetler, 1975] for a complete presentation of the mean-shift in non-Euclidean space.

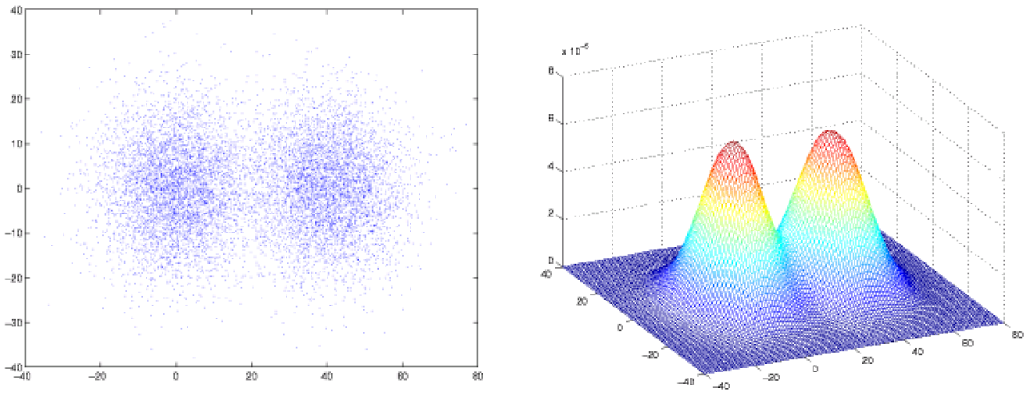


Figure C.1: The basic idea of mean shift clustering is to consider each point in a point cloud (here in 2D) as a sample from an unknown density function (right). As forming the clusters in this space is equivalent to find the places where points accumulate, this also the places where the gradient of the density is equal to zero. The mean-shift is an elegant way to find these places without knowing the density function or having any assumption about it.

We suppose that we have a point clouds in $\mathbb{R}^d$ which we want to estimate the mode (see Figure C.1 for an example in 2D). Each mode of a point cloud correspond to a local maximum of its density function, and a well-known method to reach it consists in following the direction

of the gradient. The mean-shift is an elegant way to find these places without knowing the density function or having any assumption about it.

C.1 Estimating density gradient

With parametric statistics, we have some assumptions about the law followed by the density of the point cloud, which can be assimilated to *a priori* knowledge. Moreover, usual laws are unimodal whereas real data have often multiple modes. This can be treated by composing multiples laws together (such as sum of Gaussian laws) but *a priori* knowledge is still assumed.

The solution to avoid such *a priori* is to use non-parametric statistics. Parzen window allows to estimate the density of a point cloud: if we note $\{x_i\}_{i=1...n}$ the set of n points of the cloud, we can estimate its density in all points x by:

$$\hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^{n} K\Big(\frac{x - x_i}{h}\Big)$$

where K is the *kernel function* and h is the radius of the ball centered on x called the *bandwidth*. A kernel function is a function $K : \mathbb{R}^d \to \mathbb{R}$ that verify:

(1) $\forall u \quad K(u) \geq 0$
(2) $\exists M \; \forall u \quad K(u) \leq M$
(3) $\int_{\mathbb{R}^d} K(u) \mathrm{d}u = 1$
(4) $\lim_{||u|| \to +\infty} ||u||^d K(u) = 0$

The gradient of the density function is estimated by the gradient of the estimated density:

$$\hat{\nabla} f(x) = \nabla \hat{f}(x) = \frac{1}{nh^d} \sum_{i=1}^{n} \nabla K\Big(\frac{x - x_i}{h}\Big) \tag{C.1}$$

The difference between the estimated density function and the real density function can be measured using quadratic error:

$$QE = E\Big(\int_{\mathbb{R}^d} \big(\hat{f}(x) - f(x)\big)^2 dx\Big) = \int_{\mathbb{R}^d} EQM(\hat{f}(x)) dx$$

where E design mathematic esperancy and EQM designed the mean quadratic error. [Scott, 1992] proves that minimizing QE gives the following kernel function K_E:

$$K_E(x) = \begin{cases} \frac{1}{2} C_d^{-1} (d+2)(1 - ||x||^2) & ||x|| < 1 \\ 0 & \text{otherwise} \end{cases}$$

which is called the *Epanechinokov kernel*. We note C_d the volume of the unit sphere in dimension d. Using this kernel in equation C.1, we have:

$$\hat{\nabla} f_E(x) = \frac{n_x}{n(h^d C_d)} \frac{d+2}{h^2} M_h(x)$$

with:

$$M_h(\boldsymbol{x}) = \frac{1}{n_x} \sum_{\boldsymbol{x}_i \in S_h(\boldsymbol{x})} \boldsymbol{x}_i - \boldsymbol{x}$$

where $S_h(\boldsymbol{x})$ is the hyper-sphere of radius h and center $\boldsymbol{x}$ which contains n_x points of the cloud.

The vector $M_h(\boldsymbol{x})$ is called the **mean-shift vector** and allows to define a path that leads to a local maximum of density *i.e.* a mode of the point cloud.

C.2 Algorithm

We suppose that we know the bandwidth h, which is generally computed from that scale of the point cloud. The simple algorithm to find a mode of the point cloud is the following:

1. Initialization : Pick a random point $\boldsymbol{x}$ in the point cloud
2. Compute the mean shift vector $M_h(\boldsymbol{x})$
3. $\boldsymbol{x} \leftarrow \boldsymbol{x} + M_h(\boldsymbol{x})$
   Repeat (2) until convergence

The test of convergence is usually done by testing the norm of the mean-shift vector and the convergence of this algorithm is proved when using the Epanechinokov kernel.

List of Figures

1 Exemple d'information de structure . . . 10
2 Description 3D de la PowerPlant . . . 11
3 Échange de données . . . 12
4 modélisation non-interactive . . . 13
5 Vue d'ensemble . . . 13
6 Exemples de tuiles . . . 14
7 Caractéristiques des symétries . . . 15
8 Extraction des symétries d'une forme simple . . . 16
9 Symétries d'un objet complexe . . . 17
10 Instantiation Hiérarchique: vue d'ensemble . . . 17
11 Hiérarchie d'instances optimisé pour le lancer de rayons . . . 18

1.1 3D Description of the powerplant model . . . 24
1.2 Volume rendering . . . 25
1.3 Laser Scanners . . . 26
1.4 multi-view stereo . . . 27
1.5 Procedural Modeling . . . 27
1.6 Sweep surfaces . . . 28
1.7 $L-$ system to generate plants . . . 29
1.8 Procedural modeling of a city . . . 30
1.9 Thesis Outline . . . 31

2.1 Examples of tiles decomposition . . . 34

3.1 Symmetries of the cube . . . 43
3.2 Geometric Congruency . . . 44
3.3 Symmetry as a pattern-matching problem . . . 46
3.4 Pattern-matching: rotational symmetry . . . 47
3.5 Pattern-matching: reflective symmetry . . . 47
3.6 Symmetry of 2D point set . . . 48
3.7 Symmetry of polyhedra: Graph-based methods . . . 48
3.8 Analytic conditions for symmetry . . . 50
3.9 Global and Local symmetries . . . 51
3.10 2D symmetry of a butterfly . . . 52
3.11 Folding meshes . . . 52
3.12 Symmetry as a continuous feature . . . 54
3.13 Symmetry Descriptor . . . 55
3.14 Planar Reflective Symmetry Transform . . . 56

3.15 Planar Reflective Symmetry Transform: Applications . . . 56

4.1 Extraction of symmetries for a single shape . . . 63
4.2 Symmetries of the cube . . . 63
4.3 Test models for robustness tests . . . 64
4.4 Robustness tests results . . . 65
4.5 Symmetries of scanned models . . . 66
4.6 Scanned models: Computation time . . . 66
4.7 Congruent Descriptor . . . 68
4.8 Test scenes for the congruent descriptor . . . 69
4.9 Efficiency of the congruent descriptor . . . 69
4.10 Efficiency of the congruent descriptor . . . 69
4.11 One-to-one mappings . . . 71
4.12 Step-by-step example . . . 72
4.13 Symmetries of complex objects . . . 73
4.14 Benefits of constructive algorithm . . . 74
4.15 Symmetries inside non coherent geometry . . . 75
4.16 Symmetries: Computation cost . . . 76

5.1 Symmetry-based compression of the powerplant model . . . 78
5.2 Compression rates results . . . 78
5.3 Symmetry-based mesh editing . . . 79
5.4 Isotropic models . . . 80
5.5 Some isotropic models . . . 81

7.1 Instancing capability of X3D . . . 90
7.2 Ray-tracing and Instancing . . . 92
7.3 Benefits of instancing for ray-tracing . . . 92
7.4 Hierarchical Radiosity . . . 93
7.5 Hierarchical Radiosity with instancing . . . 94
7.6 Hardware-based geometry instancing . . . 94
7.7 hardware-based instancing with DirectX . . . 95
7.8 Scene Editing using instancing information . . . 97
7.9 Landscape rendered with approximate instancing . . . 98
7.10 Automatic Instancing of hierarchically organized objects . . . 99
7.11 Hierarchical Instancing of Geometry: Overview of our approach . . . 99

8.1 Neighborhood Graph . . . 102
8.2 Simple lattice of frequent patterns . . . 103
8.3 Complex lattice of frequent patterns . . . 105
8.4 closed patterns . . . 106
8.5 Early pruning of nodes . . . 107
8.6 Violating the downward closure property . . . 108
8.7 Maximum Independent Set . . . 108
8.8 sample scenes features . . . 111
8.9 Results of agglomerative approach . . . 111

8.10 Some patterns obtained with the agglomerative approach . . . 112
8.11 Basic Assumption of symmetry-based approach . . . 113
8.12 Two instances of a pattern form a local symmetry in the scene . . . 114
8.13 Mapping between congruent tiles . . . 115
8.14 Mapping between tiles with cylindrical symmetries . . . 117
8.15 Mapping between tiles with spherical symmetries . . . 119
8.16 Principle of mean shift clustering . . . 120
8.17 Rejection test for Cylindrical Symmetries . . . 121
8.18 The neighborhood graph is connected . . . 124
8.19 Labeling the edges of the neighborhood graph is connected . . . 124
8.20 The graph hierarchy of the enhancement method . . . 125
8.21 Results obtained with the reduced mappings set . . . 126
8.22 sample scenes features . . . 126
8.23 Computation time of the symmetry-based approach . . . 126
8.24 Frequent Patterns obtainedon the Plane Model . . . 127
8.25 Frequent Patterns obtained on the Building Model . . . 127
8.26 Frequent Patterns obtained on the LBX Studio . . . 129

9.1 Hierarchy Assembly Graph . . . 132
9.2 Transitivity nature of the Inclusion relation . . . 133
9.3 Transitive reduction of the Pattern Graph . . . 134
9.4 HAG construction . . . 135
9.5 An example of hierarchy with overlapping instances . . . 135
9.6 A simple HAG with no overlapping . . . 137
9.7 Hierarchy of instances optimized for Ray-Tracing . . . 139
9.8 Removing single-instantiated patterns . . . 141
9.9 Removing a single-instantiated pattern may changed the cost of the patterns it was linked to . . . 142
9.10 Hierarchy of instances for ray-tracing: Results . . . 143
9.11 Hierarchy of instances for the Plane Model optimized for ray-tracing . . . 144
9.12 Hierarchy of instances for the Building Model optimized for ray-tracing . . . 145

B.1 Classification theorem: Case 2 . . . 158
B.2 Classification Theorem: Case 3a . . . 159
B.3 Classification Theorem: Case 3b . . . 159
B.4 Classification Theorem: Case 4a where the lines AB and A'B' are parallel . . . 160
B.5 Classification Theorem: Case 4a where the lines AB and A'B' are not parallel . . . 161
B.6 Classification Theorem: Case 4b . . . 161

C.1 Principle of mean shift clustering . . . 163

List of Algorithms

1 Frequent Pattern Discovery by depth-first visit of the lattice of Frequent Pattern. 110
2 ExtendPattern 110
3 Generate Points in Γ 116
4 Isometry 116
5 FilterClosedPatterns 122
6 GraphDecomposition 125
7 OptimalHierarchy: Approximation algorithm for ray-tracing 141

Bibliography

A. V. Aho, M. R. Garey, and J. D. Ullman. The transitive reduction of a directed graph. *SIAM Journal on Computing*, 1(2):131–137, 1972. http://link.aip.org/link/?SMJ/1/131/1.

Helmut Alt, Kurt Mehlhorn, Hubert Wagener, and Emo Welzl. Congruence, similarity and symmetries of geometric objects. *Discrete Comput. Geom.*, 3(3):237–256, 1988. ISSN 0179-5376.

Remi Arnaud and Mark C. Barnes. *Collada: Sailing the Gulf of 3d Digital Content Creation*. AK Peters, Ltd.; 1 edition (August 30, 2006), 2003.

A. Atramentov and S. M. LaValle. Efficient nearest neighbor searching for motion planning. In *Proceedings IEEE International Conference on Robotics and Automation*, pages 632–637, 2002.

M. J. Attalah. On symmetry detection. *IEEE Transactions on Computers*, 34:663–666, 1985.

I. C. Braid. On storing and changing shape information. In *SIGGRAPH '78*, pages 252–256. ACM Press, 1978. http://doi.acm.org/10.1145/800248.807399.

Peter Brass and Christian Knauer. Testing congruence and symmetry for general 3-dimensional objects. *Comput. Geom. Theory Appl.*, 27(1):3–11, 2004. ISSN 0925-7721. http://dx.doi.org/10.1016/j.comgeo.2003.07.002.

Andrew Brownbill. Reducing the storage required to render L-system based models. Master's thesis, University of Calgary, October 1996.

Francesco Carucci. Inside geometry instancing. In Matt Pharr, editor, *GPU Gems 2*. Addison Wesley, March 2005.

Cem Cebenoyan. Direct3D API issues : Instancing and floating-point specials. Technical report, NVidia, 2005.

Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. *IEEE Trans. Pattern Anal. Mach. Intell.*, 24(5):603–619, 2002. ISSN 0162-8828. http://dx.doi.org/10.1109/34.1000236.

Paul E. Debevec. *Modeling and Rendering Architecture from Photographs*. PhD thesis, University of California at Berkeley, Computer Science Division, Berkeley CA, 1996.

Oliver Deussen, Pat Hanrahan, Bernd Lintermann, Radomir Mech, Matt Pharr, and Przemyslaw Prusinkiewicz. Realistic modeling and rendering of plant ecosystems. In *Proc. of SIGGRAPH '98*, pages 275–286, 1998. ISBN 0-89791-999-8. http://doi.acm.org/10.1145/280814.280898.

Oliver Deussen, Carsten Colditz, Marc Stamminger, and George Drettakis. Interactive visualization of complex plant ecosystems. In *Proceedings of the IEEE Visualization Conference*. IEEE, October 2002. http://www-sop.inria.fr/reves/publications/data/2002/DCSD02.

David S. Ebert, Kenton F. Musgrave, Darwyn Peachey, Ken Perlin, and Steven Worley. *Texturing & Modeling: A Procedural Approach, Third Edition (The Morgan Kaufmann Series in Computer Graphics)*. Morgan Kaufmann, December 2002. http://www.amazon.co.uk/exec/obidos/ASIN/1558608486/citeulike-21.

David Eppstein. Subgraph isomorphism in planar graphs and related problems. In *Proc. 6th Symp. Discrete Algorithms*, pages 632–640. ACM and SIAM, January 1995.

Deborah R. Fowler, Hans Meinhardt, and Przemyslaw Prusinkiewicz. Modeling seashells. In *SIGGRAPH '92*, pages 379–387. ACM Press, 1992. ISBN 0-89791-479-1. http://doi.acm.org/10.1145/133994.134096.

K. Fukunaga and L.D. Hostetler. The estimation of the gradient of a density function, with applications in pattern recognition. In *IEEE Trans. Information Theory*, volume 21, pages 32–40, 1975.

Thomas A. Funkhouser and Carlo H. Séquin. Adaptive display algorithm for interactive frame rates during visualization of complex virtual environments. In *SIGGRAPH '93*, pages 247–254. ACM Press, 1993. ISBN 0-89791-601-8. http://doi.acm.org/10.1145/166117.166149.

Michael R. Garey and David S. Johnson. *Computers and Intractability : A Guide to the Theory of NP-Completeness*. Freeman, 1979.

Michael Goesele, Brian Curless, and Steven M. Seitz. Multi-view stereo revisited. In *CVPR '06*, pages 2402–2409. IEEE Computer Society, 2006. ISBN 0-7695-2597-0. http://dx.doi.org/10.1109/CVPR.2006.199.

Cindy M. Goral, Kenneth E. Torrance, Donald P. Greenberg, and Bennett Battaile. Modeling the interaction of light between diffuse surfaces. In *SIGGRAPH '84*, pages 213–222, 1984. ISBN 0-89791-138-5. http://doi.acm.org/10.1145/800031.808601.

J. C. Hart. The object instancing paradigm for linear fractal modeling. In *Proceedings of Graphics Interface '92*, pages 224–231, 1992. http://citeseer.ist.psu.edu/hart92object.html.

Jean-Marc Hasenfratz, Cyrille Damez, François Sillion, and George Drettakis. A practical analysis of clustering strategies for hierarchical radiosity. *Computer Graphics Forum (Proc. of Eurographics '99)*, 18(3):221–232, September 1999. http://artis.imag.fr/Publications/1999/HDSD99.

P. T. Highnam. Optimal algorithms for finding the symmetries of a planar point set. Technical Report CMU-RI-TR-85-13, Robotics Institute, Carnegie Mellon University, Pittsburgh, PA, August 1985.

E. W. Hobson. *The theory of Spherical and Ellipsoidal harmonics*. Cambridge University Press, Cambridge, UK, 1931.

J. Ivanic and K. Ruedenberg. Rotation matrices for real spherical harmonics, direct determination by recursion. *J. Phys. Chem. A.*, 100:6342–6347, 1996. See also *Additions and corrections* in Vol. 102, No.45, 9099-9100.

Xiao-Yi Jiang and Horst Bunke. Determination of the symmetries of polyhedra and an application to object recognition. In *CG '91: Proceedings of the International Workshop on Computational Geometry - Methods, Algorithms and Applications*, volume 553 of *Lecture Notes in Computer Science*, pages 113–121, London, UK, 1991. Springer. ISBN 3-540-54891-2.

Akihisa Kako, Takao Ono, Tomi Hirata, and Magnus Halldorsson. Approximation algorithms for the weighted independent set probelm. *Lecture notes in computer science (Lect. notes comput. sci.) ISSN 0302-9743*, 2005.

Timothy L. Kay and James T. Kajiya. Ray tracing complex scenes. *Computer Graphics (Proc. of SIGGRAPH '86)*, 20(4):269–278, August 1986. http://doi.acm.org/10.1145/15922.15916.

Michael Kazhdan, Bernard Chazelle, David Dobkin, Thomas Funkhouser, and Szymon Rusinkiewicz. A reflective symmetry descriptor for 3D models. *Algorithmica*, 38(1), October 2003a.

Michael Kazhdan, Thomas Funkhouser, and Szymon Rusinkiewicz. Shape matching and anisotropy. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, August 2004a.

Michael Kazhdan, Thomas A. Funkhouser, and Szymon Rusinkiewicz. Symmetry descriptors and 3D shape matching. In *SGP '04*, July 2004b.

Michael M. Kazhdan, Thomas A. Funkhouser, and Szymon Rusinkiewicz. Rotation invariant spherical harmonic representation of 3D shape descriptors. In *SGP '03*, pages 167–175, June 2003b.

Donald E. Knuth, James H. Morris, Jr., and Vaughan R. Pratt. Fast pattern matching in strings. *SIAM Journal on Computing*, 6(2):323–350, 1977.

James Kuffner. Effective sampling and distance metrics for 3d rigid body path planning. In *Proc. 2004 IEEE Int2019l Conf. on Robotics and Automation (ICRA 2004)*. IEEE, May 2004.

M. Kuramochi and G. Karypis. Finding frequent patterns in a large sparse graph, 2004. citeseer.ifi.unizh.ch/article/kuramochi04finding.html.

J. A. La Poutré and J. van Leeuwen. Maintenance of transitive closure and transitive reduction of graphs. In *Proc. Workshop on Graph-Theoretic Concepts in Computer Science*, pages 106–120. Lecture Notes in Computer Science 314, Springer-Verlag, Berlin, 1988. citeseer.ist.psu.edu/poutre87maintenance.html.

Marc Levoy, Kari Pulli, Brian Curless, Szymon Rusinkiewicz, David Koller, Lucas Pereira, Matt Ginzton, Sean Anderson, James Davis, Jeremy Ginsberg, Jonathan Shade, and Duane Fulk. The digital michelangelo project: 3D scanning of large statues. In *Proceedings of ACM SIGGRAPH 2000*, pages 131–144, July 2000.

Giovanni Marola. On the detection of the axes of symmetry of symmetric and almost symmetric planar images. *IEEE Trans. Pattern Anal. Mach. Intell.*, 11(1):104–108, 1989.

Aurélien Martinet, Cyril Soler, Nicolas Holzschuch, and François X. Sillion. Accurate detection of symmetries in 3d shapes. *ACM Trans. Graph.*, 25(2):439–464, 2006. ISSN 0730-0301. http://doi.acm.org/10.1145/1138450.1138462.

P. Minovic, S. Ishikawa, and K. Kato. Symmetry identification of a 3-D object represented by octree. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 15(5):507–514, 1993. ISSN 0162-8828. http://dx.doi.org/10.1109/34.211472.

D.P Mitchell and J. Amanatides. Megacycles., 1989.

N. J. Mitra, L. Guibas, and M. Pauly. Partial and approximate symmetry detection for 3d geometry. In *ACM Transactions on Graphics*, volume 25, pages 560–568, 2006.

P.R.J Ostergard. A fast algorithm for the maximum clique problem. *Discrete Appl. Math.*, 120(1-3):197–207, 2002. ISSN 0166-218X. http://dx.doi.org/10.1016/S0166-218X(01)00290-6.

Yoav I. H. Parish and Pascal Muller. Procedural modeling of cities. In *SIGGRAPH '01*, pages 301–308. ACM Press, 2001. ISBN 1-58113-374-X. http://doi.acm.org/10.1145/383259.383292.

Joshua Podolak, Philip Shilane, Aleksey Golovinskiy, Szymon Rusinkiewicz, and Thomas Funkhouser. A planar-reflective symmetry transform for 3D shapes. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 25(3), July 2006.

Przemyslaw Prusinkiewicz, Lars Mundermann, Radoslaw Karwowski, and Brendan Lane. The use of positional information in the modeling of plants. In *SIGGRAPH '01*, pages 289–300. ACM Press, 2001. ISBN 1-58113-374-X. http://doi.acm.org/10.1145/383259.383291.

S. Quinlan. Efficient distance computation between non-convex objects. In *IEEE Intern. Conf. on Robotics and Automation*, pages 3324–3329. IEEE, 1994. citeseer.ist.psu.edu/quinlan94efficient.html.

Ravi Ramamoorthi and Pat Hanrahan. A signal-processing framework for reflection. *ACM Transactions on Graphics*, 23(4):1004–1042, 2004. ISSN 0730-0301. http://doi.acm.org/10.1145/1027411.1027416.

R. Schultz and H. Schumann. Automatic instancing of hierarchically organized objects. In *Spring Conference on Computer Graphics*, 2001.

D.W. Scott. *Multivariate Density Estimation. Theory, Practice and Visualization*. 1992.

Steven M. Seitz, Brian Curless, James Diebel, Daniel Scharstein, and Richard Szeliski. A comparison and evaluation of multi-view stereo reconstruction algorithms. In *CVPR '06*, pages 519–528, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2597-0. http://dx.doi.org/10.1109/CVPR.2006.19.

François Sillion and Claude Puech. *Radiosity and Global Illumination*. Morgan Kaufmann Publishers, San Francisco, 1994. URL http://artis.imag.fr/Publications/1994/SP94. ISBN 1-558.

Patricio Simari, Evangelos Kalogerakis, and Karan Singh. Folding meshes: Hierarchical mesh segmentation based on planar symmetry. In *Eurographics Symposium on Geometry Processing*, 2006.

John M. Snyder and Alan H. Barr. Ray tracing complex models containing surface tessellations. *Computer Graphics (Proc. of SIGGRAPH '87)*, 21(4):119–128, July 1987. http://doi.acm.org/10.1145/37401.37417.

Cyril Soler and François Sillion. Hierarchical instantiation for radiosity. In B. Peroche and H. Rushmeier, editors, *Rendering Techniques '00*, pages 173–184. Springer Wien, 2000. http://artis.imag.fr/Publications/2000/SS00.

Cyril Soler, François Sillion, Frédéric Blaise, and Philippe Dereffye. An efficient instantiation algorithm for simulating radiant energy transfer in plant models. *ACM Transactions On Graphics*, 22(2), April 2003. http://artis.imag.fr/Publications/2003/SSBD03.

C. Sun and J. Sherrah. 3D symmetry detection using extended Gaussian image. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 19(2):164–168, February 1997.

Changming Sun. Symmetry detection using gradient information. *Pattern Recogn. Lett.*, 16(9):987–996, 1995. ISSN 0167-8655. http://dx.doi.org/10.1016/0167-8655(95)00049-M.

N. Vanetik, E. Gudes, and S. E. Shimony. Computing frequent graph patterns from semistructured data. In *ICDM '02*, page 458, Washington, DC, USA, 2002. IEEE Computer Society. ISBN 0-7695-1754-4.

Rand Waltzman. *Geometric problem-solving by machine visualization*. PhD thesis, 1989. Director-Azriel Rosenfeld.

Jianning Wang and Manuel M. Oliveira. Improved scene reconstruction from range images. *Comput. Graph. Forum*, 21(3), 2002.

Marta Wilczkowiak. *3D Modelling From Images Using Geometric Constraints*. PhD thesis, Institut National Polytechnique de Grenoble, 2004.

Marta Wilczkowiak, Gilles Trombettoni, Christophe Jermann, Peter Sturm, and Edmond Boyer. Scene modeling based on constraint system decomposition techniques. In *Proceedings of the 9th International Conference on Computer Vision, Nice, France*, volume II, pages 1004–1010. IEEE, IEEE, October 2003. http://perception.inrialpes.fr/Publications/2003/WTJSB03.

J. D. Wolter, T. C. Woo, and R. A. Volz. Optimal algorithms for symmetry detection in two and three dimensions. *The Visual Computer*, 1:37–48, 1985.

X. Yan and J. Han. gspan: Graph-based substructure pattern mining, 2002. citeseer.ist.psu.edu/yan02gspan.html.

Hagit Zabrodsky, Shmuel Peleg, and David Avnir. Symmetry as a continuous feature. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 17(12):1154–1166, 1995. http://citeseer.ist.psu.edu/article/zabrodsky95symmetry.html.