Procrustes: A Declarative Scene Modelling System

A Thesis

Submitted to the Faculty of Graduate Studies and Research

for the Degree of

Master of Science

in Computer Science

University of Regina

by

Spoorthy Seenappa

Regina, Saskatchewan

October, 2012

Copyright 2012: S. Seenappa
Spoorthy Seenappa, candidate for the degree of Master of Science in Computer Science, has presented a thesis titled, *Procrustes: A Declarative Scene Modelling System*, in an oral examination held on September 27, 2012. The following committee members have found the thesis acceptable in form and content, and that the candidate demonstrated satisfactory knowledge of the subject material.

External Examiner: Dr. Raman Paranjape, Electronic Systems Engineering

Supervisor: Dr. Howard J. Hamilton, Department of Computer Science,

Committee Member: Dr. Daryl H. Hepting, Department of Computer Science

Committee Member: Dr. Malek Mouhoub, Department of Computer Science

Chair of Defense: Dr. Nader Mobed, Department of Physics
Abstract

This thesis presents a floor planning and furniture layout system called Procrustes, that helps a user to place rooms and furniture in a virtual house. Scenes are generated by considering the user's partial scene description. The resulting floor plans and furniture arrangements can be used in video games to generate scenes that are not of prime importance to the games.

We developed the Procrustes Declarative Scene Modelling (DSM) system to reduce the amount of user input compared to existing DSM systems. The name Procrustes is extracted from a Greek mythological character who placed people in an iron bed and ensured that their bodies fit by either cutting their limbs or physically stretching them. The concept of forcing the object to fit in a position is the common feature between our system and the Greek myth.

Given a partial scene description, Procrustes extracts hierarchical and spatial relations that constrain the scene. The scene generation process involves floor planning and furniture arrangement. To reduce the amount of input from the user, the sizes of the rooms and furniture objects are set to default values. After all the rooms are placed in the house, the empty spaces scattered throughout the house are reduced and the sizes of the rooms are adjusted correspondingly. To complete the floor plan, a hallway is generated to interconnect the rooms. Then, the furniture objects are placed by considering factors such as purpose, accessibility and visibility.

After a set of scenes has been generated, ten of these scenes are presented to the user who response to each one as positive or negative or no input. Based on these
responses, more scenes are generated with spatial relations similar to the positive scenes but dissimilar to the negative ones. This process is iterated until a fixed number of 20 iterations is reached or the user is satisfied.

The utility of Procrustes is illustrated by generating scenes for several partial scene descriptions. The results are evaluated by comparing the scenes to the descriptions, by checking for wasted space, and by checking the hallway placement.
Acknowledgements

I would like to thank my supervisor, Dr. Howard Hamilton, whose expertise, understanding, and patience added considerably to my research and thesis experience. I would like to thank Dr. Daryl H. Hepting and Dr. Malek Mouhoub for taking time out from their busy schedules to advise me. I would also like to thank the Natural Sciences and Engineering Research Council of Canada (via grants administered by Dr. Hamilton), the Faculty of Graduate Studies and Research and the Department of Computer Science for financial support.
Dedication

I would like to dedicate this work to my father V. Seenappa and mother Bhavani, who loved, supported and encouraged me through my entire life, my sister, Shilpa for making University of Regina possible for me and being around, to my boyfriend, Mark for his love, encouragement and assistance, and to my closest friends Manjeeth and Preethi, who have always been there when I needed them and Allen and Joan for making Regina a home away from home.
# Table of Contents

1. Introduction .................................................................................................................. 1
2. Previous Work ................................................................................................................ 7
   2.1. Classification of Declarative Scene Modellers ....................................................... 7
   2.2. The DSM Design Process ....................................................................................... 8
      2.2.1. Scene Description .............................................................................................. 9
      2.2.2. Scene Generation .............................................................................................. 11
      2.2.3. Insight Phase .................................................................................................. 12
   2.3. Techniques to Perform Scene Generation .............................................................. 12
      2.3.1. ORANOS ........................................................................................................ 13
      2.3.2. MultiForms .................................................................................................... 15
      2.3.3. ARCHiPLAN .................................................................................................. 17
      2.3.4. MultiCAD with Genetic Algorithm ................................................................. 20
      2.3.5. Summary ........................................................................................................ 23
   2.4. Furniture Layout Problems ..................................................................................... 26
      2.4.1. Automatic Optimization of Furniture Arrangement ........................................ 26
      2.4.2. Interactive Furniture Layout Using Interior Design Guidelines .................... 29
   2.5. Differences between DSM and Automatic Furniture Layout Techniques ............ 30
   2.6. Discussion ............................................................................................................... 33
3. Scene Generation Techniques ......................................................................................... 35
   3.1. Scene Generation Input .......................................................................................... 39
      3.1.1. Knowledge Base .............................................................................................. 40
      3.1.2. Description Tree ............................................................................................. 42
   3.2. Relation Extraction ................................................................................................. 42
   3.3. The Box-the-Space Technique .............................................................................. 44
      3.3.1. Update Empty Space ....................................................................................... 46
      3.3.2. Advantages of the-Box-the-Space technique .................................................. 47
   3.4. Processing a Relation ............................................................................................. 48
   3.5. The Space Reduction Algorithm .......................................................................... 51
      3.5.1. The Space Deletion Phase ............................................................................... 53
      3.5.2. Object Expansion ........................................................................................... 56
   3.6. Generation of a Hallway ......................................................................................... 61
   3.7. Placement of Furniture in Rooms ......................................................................... 65
      3.7.1. Common Purpose of Objects ......................................................................... 65
      3.7.2. Visibility Relations ......................................................................................... 67
      3.7.3. Accessibility Relations .................................................................................... 70
   3.8. Similar Scene Generation ....................................................................................... 70
      3.8.1. The Relation Rating Phase ............................................................................. 71
      3.8.2. Selective Relation Merging Phase ................................................................... 73
4. Descriptive Example ....................................................................................................... 78
   4.1. Partial Scene Description ....................................................................................... 79
   4.2. Initial Scene Generation ......................................................................................... 80
4.3. Reduction of Empty Spaces .................................................................82
4.4. Generation of Hallway ........................................................................83
4.5. Placement of Furniture .................................................................83
4.6. Similar Scene Generation ...............................................................88
5. Conclusions and Future Work ............................................................91
References ..........................................................................................95
Appendix A: Procrustes Knowledge Base ..............................................99
Appendix B: Parsing of the Input Description ........................................101
Appendix C: Overview of the Procrustes System ....................................104
List of Tables

Table 2.1: Comparison of various DSM techniques ......................................................... 25
Table 2.2: Differences between DSM and automatic furniture layout techniques ....... 30
Table 3.1: Values for the encountered points for various directions .......................... 59
Table 4.1: Characters representing the room objects ................................................... 80
Table C.1: Procurstes Modules and Files ................................................................. 105
List of Figures

Figure 2.1: Three phases in DSM Design Process .......................................................... 10
Figure 2.2: Calculating contact length (d1) and distance d2 between two objects (O1 and O2) ......................................................................................................................... 19
Figure 2.3: Chromosome representation of a scene with two sub-scenes .................. 21
Figure 2.4: Illustration of crossover operation ............................................................... 22
Figure 2.5: Illustration of mutation operation ................................................................. 23
Figure 2.6: A viewing frustum associated with object A ............................................... 28
Figure 3.1: Input and output for scene generation phase ................................................. 38
Figure 3.2: An example description tree ........................................................................ 43
Figure 3.3: Generation of scenes with various relations ............................................... 45
Figure 3.4: Updating the empty spaces after placing the living room object .............. 47
Figure 3.5: Scenes demonstrating the processing of a relation .................................... 49
Figure 3.6: Effects of space reduction algorithm ........................................................... 52
Figure 3.7: Selection of spaces for deletion with X as reduction axis ......................... 54
Figure 3.8: Illustration of Object Expansion ................................................................. 59
Figure 3.9: Object Expansion with the back side as the starting side ......................... 61
Figure 3.10: Processing the neighbors for generation of hallway .............................. 64
Figure 3.11: Surfaces of a furniture object .................................................................. 67
Figure 3.12: Visible zone of object A .......................................................................... 68
Figure 3.13: Visible parent zone for object A ............................................................... 69
Figure 3.14: The structure of a relation’s score ............................................................. 72
Figure 3.15: Illustration of Selective Relation Merging .................................................. 77
Figure 4.1: Example input scene description ............................................................... 80
Figure 4.2: Example scene demonstrating scene generation ...................................... 81
Figure 4.3: Scenes with reduced empty spaces ............................................................. 82
Figure 4.4: Scene with a hallway .................................................................................. 83
Figure 4.5: Placement of furniture objects ................................................................... 85
Figure 4.6: Visible zone for TV and painting objects .................................................... 86
Figure 4.7: Visible zones for painting intersecting the obstructed zones .................... 87
Figure 4.8: Complete layout with furniture ................................................................. 87
Figure 4.9: Example scenes demonstrating similar scene generation ....................... 89
Figure C.2: Modules in Procrustes .............................................................................. 104
Chapter 1

Introduction

Developing a virtual scene for applications such as animation, video games, furniture layout and house design is a challenging task. In the context of this thesis, a virtual scene or simply a scene, is an entity or environment to be designed. Creating such a scene involves making decisions about the size, position, orientation and other properties of every object in the scene [9]. It also require to make decisions about the relations between the objects. For instance, suppose a user wants to develop a scene with a “coffee table in front of a large sofa”. In this natural language description, the words “large sofa” define a property (“large”) of an object (“sofa”). The words “in front of” define a relation between two objects, which in this case will affect the positions and orientations of the objects. Defining all the properties and relations consistently is not an easy task. Visualising a scene may also require dealing with a graphics engine. Furthermore, generating a scene can also be time consuming due to the number of drafts that may be required in the design process. Software that aids in making decisions about layout properties and handling visualisation can make the design process for a scene more efficient.

In general, many scenes are designed with Computer Aided Design (CAD) tools.
These tools require the geometrical properties of a scene to be defined at the onset of the Design phase. Typically, CAD tools lack decision making capabilities and the ability to deal with natural language description. Due to the lack of decision making abilities in CAD tools, the user needs to have a clear idea of the design he or she plans to generate [16]. It is also expected that the user is aware or can obtain the exact measurements of every object in the scene.

*Declarative Scene Modelling* (DSM), also called *Declarative Modelling* [3] is intended to address the above limitations in utilizing CAD tools for the purpose of design process. DSM is a technique that takes a description of a scene given in natural language and generates scenes matching this description. DSM is intended to help the user during the initial phase of scene design, when many relatively unimportant decisions must be made. The input description specifies the required properties of the scene and the Declarative Scene Modelling software decides upon the remaining optional properties. For simplicity, a DSM software system is referred to as a *modeller*.

A typical approach to DSM has three phases, which are iterated until a satisfactory scene is found [3]. In the *description phase*, the user describes the scene using a natural language. The modeller interprets the high-level description obtained from the user and converts it to facts, i.e. statements in a logical language, such as Prolog [3]. The output of this phase is a *declarative scene description*, typically consisting of facts along with object identifiers for all objects that will be present in the scene. The declarative scene description is passed as input to the generation phase. In the *generation phase*, the variables required to specify a scene are instantiated. Instantiation includes generation of
relations for any objects that are not yet related to other objects. As an output, this phase generates values for all required properties of the objects (such as any size, position and orientation) that are not specified in the description. Where alternate instantiations are possible, multiple scenes are generated.

In the insight phase, the modeller presents the user with or more generated scenes and accepts a user response. If the response is to accept one of the scenes, the modeller is finished. Otherwise, depending on the modeller, the response may be in the form of either additional description or numeric scores for the scenes. If the response is additional description, the user is allowed to specify adjustments to the presented scene in an intuitive manner. Adjustments may be suggestions to add, delete or to change the properties of existing objects. These adjustments, along with the current scenes, are sent back to the description phase, where they are incorporated into the declarative scene description. The resulting description is passed to the generation phase, and so on. On the other hand, if the modeller uses numeric scores, these scores are passed directly to the generation phase to provide information about desirable choices for future scenes. In either case, after accepting the user response, the modeller generate scenes closer to the desired scene. The overall process is repeated until a satisfactory scene is found.

The previously developed modellers have two significant limitations with respect to potential DSM applications in computer games. The first limitation relates to the large number of decisions that must be made. During the DSM process, the user is required to make decisions about all relevant constraints. This makes these modellers appropriate when the user knows the majority of the constraints on the scene to be generated, but is a
significant limitation when the user has not yet decided on many of the constraints and would prefer the system to provide some possibilities. For computer games, many scenes of relatively little importance may be needed and a game designer would prefer not to have to make numerous decisions for each.

The second limitation relates to the methods for receiving a response from the user after generated scenes have been presented. If the modeller requires a response in the form of additional description, the user is required to make more decisions, describe the adjustments relatively precisely, and perform extra text entry. On the other hand to assign numeric scores to several scenes, the user must compare the presented scenes with each other and make quantitative assessments of their differences. Both of these methods require relatively large amounts of input from the user.

In this thesis, we investigate the hypothesis that a DSM system can be designed that performs both floor planning and furniture layout using a consistent form of user interaction. In this form of interaction, the user provides an initial textual description of a scene involving rooms and furniture and then views sample scenes and rates them as positive, negative or neutral. It is further hypothesized that this type of system will require less input than existing DSMs and provide more flexibility than existing furniture layout systems.

In more detail, this thesis investigates an approach to interaction with the user in the insight phase of the DSM process, during which the user provides simple responses to presented scenes. With this approach, at most ten scenes are presented to the user and
every presented scene receives a positive, negative or neutral response from the user. The decisions for the user are relatively simple and no extra text entry is needed. This approach is also beneficial in case where the user may see desirable features in multiple scenes. The basis of this approach is similar scene generation in which every individual scene is represented as a tree, where the nodes represent portions of the scene. By comparing the facts from the positive and negative scenes, the modeller makes an attempt to generates scenes similar to the positive scene(s), but dissimilar to the negative scene(s).

Since the idea of a DSM is to help the user during the initial phase of design, the modeller should expect minimum input description from the user and generate scenes that are consistent with the given description. The goal is to help the user to explore various scenes with little effort.

The remainder of this thesis is organized as follows. In Chapter 2, we discuss some of the existing work on handling constraints and the application of genetic algorithms during the generation phase. We also compare the differences between the scene generation techniques that use solutions to a Constraint Satisfaction Problem (CSP) and the techniques that use a genetic algorithm. At last, we discuss some recent approaches to automatic furniture arrangement and compare these approaches to DSM.

Chapter 3 provides a detailed description of our approach to the scene generation phase. It also describes the algorithms adopted to traverse the concept tree and to place individual objects in the scene. Algorithms to reduce the number and size of empty
spaces and generate hallways are discussed as a part of this chapter. We also discuss the process of similar scene generation and the implementation details of the two stages involved in it.

In Chapter 4, we give a descriptive example showing the effectiveness of the approach introduced in this thesis. Chapter 5 states the conclusions from our research. It also includes discussion of possible enhancements to Procrustes and other suggestions for future research related to DSM.
Chapter 2

Previous Work

As described by Makris et. al [10], the declarative scene modelling is a “technique that allows users to describe the scene to be designed in an intuitive manner, by only giving some expected properties of the scene[,] and letting the modeller find solutions, if any, verifying these properties”. DSM can be implemented in a graphical tool that requires the user to describe a scene in a natural language. The user provides basic relations and properties to the DSM system and it adds the remaining necessary relations and properties. As a result of processing these relations and properties, a scene is generated by satisfying the input description. These scenes are then presented to the user for a response. This process is repeated as user introduces more constraints on the relations or properties of the objects in the scene at every iteration.

2.1. Classification of Declarative Scene Modellers

Based on their purposes, modellers are classified into two groups [13]:

Dedicated modellers: A dedicated modeller is one that is adapted to generate scenes for a particular application in a specific domain. Such a modeller can be efficient if the solution generation engine is well adapted to the properties of the specific domain. On the other hand, each dedicated modeller is limited to use in a specific domain. ARCHiPLAN
and a Declarative Knowledge Framework for Architecture-oriented Building Modeling (DKABM) serve as good examples for a dedicated modeller [9].

**General purpose modellers:** A *general purpose modeller* is one that is applicable to a wide variety of domains. Such a modeller has a solution generation engine that can process diverse domain-specific properties, along with a relatively small set of predefined general properties. A dedicated modeller can be obtained from a general purpose modeller by adding properties corresponding to a specific domain. Although, a general purpose modeller can be used for a variety of purposes, it lacks efficiency due to the generality of its solution generation mechanisms [13]. ORANOS is one of the well known general-purpose modeller.

### 2.2. The DSM Design Process

The *DSM design process* is a method of accepting user input, generating necessary values for the scene that are consistent with this input, and presenting the scene(s) to the user. These operations are performed in three phases, namely the scene description phase, scene generation phase and the insight phase, as shown in Figure 2.1. In order to generate a scene, the modeller has to be aware of the requirements of the user and utilize these requirements to generate an acceptable scene.

All the information required to generate scenes are saved in the knowledge base. *Knowledge base* is a logical database that stores the rules and facts needed to generate a scene. These rules and facts include information such as the name of the possible objects in the scenes, hierarchical relations, spatial relations between the objects [21]. A *hierarchical relation* represents the parent-child relation between two objects. Object A
is said to be the parent of object B, if B is a part of or in or on object A.

Assuming all objects in a scene are related hierarchically in a consistent manner, these relations implicitly define a concept tree. Objects with a common parent object are called sibling objects.

A spatial relation (also referred to as a relation) describes the placement of an object with respect to another. Spatial relations in the knowledge base are developed by considering the properties of objects from real life. Although processing spatial relations between sibling objects is computationally more costly than processing spatial relations between a pair of parent-child objects, they can be handled with adequate efficiency for the sizes of knowledge base utilised in some of the automatic furniture arrangement domain [12,23].

2.2.1. Scene Description

During the scene description phase, the user describes the scene to the modeller using a natural language. The user is required to describe relation between the current object and the object with the common parent or a pair of parent-child objects. The relevant module in the modeller defines the interaction language and offers the user a platform with which he/she supplies a description. This module contains an interpreter and a validator, which are used to process the scene description. The interpreter converts a high-level scene description given in a natural language to a suitable format, such as prolog rules and facts [10]. During this conversions, prepositions such as of, to, it and for are omitted from the input. The validator performs validation of the scene
description in the internal format to check for syntactical errors and inconsistencies. The validated description in the internal format is called the *declarative scene description*.

![Diagram of three phases in DSM Design Process](image)

**Figure 2.1**: Three phases in DSM Design Process

Miaoulisv et al. [13] introduced a widely used technique of *Declarative Modeling by Hierarchical Decomposition (DMHD)* for providing a scene description to the modeller. The DMHD is performed by utilizing the concept tree to interpret the natural language input scene description from the user to a logical form. In general, modellers allows the user to describe the properties of an object in the scene while giving relatively few details on how the object should be placed in the environment. Since the existing environment can contain numerous objects, it is tedious to think about such objects individually and precisely. DMHD uses hierarchical decomposition, to allow the user to briefly describe the overall scene and provide lower level descriptions of the objects. As mentioned by Fisher et. al [4], the user can describe a scene in a top-down manner by first mentioning the major features, such as rooms, and then progressing to minor features. It also increases locality of description because the user will concentrate on a part of a scene at a given time instead of thinking about details from various parts of the
scene. This top-down approach help the modellers categorize the properties of the objects and identify properties and relations that are inherited from the ancestor objects.

2.2.2. Scene Generation

Scene generation involves majority of the computation involved in obtaining the positions for the objects. It involves examining the constraints, properties and relations in the declarative scene description, and then produces one or more solutions, if they can be found. As one can imagine, the problem of object placement has various variables such as the object size, position and orientation. Each of these variables potentially has a wide range of possible values. To reduce these ranges and ensure that the objects appear realistic, constraints are added. Several Constraint Satisfaction Problem (CSP) solving techniques are introduced to ensure that the object variables abide by the constraints and avoid conflicts. The existing modellers are developed with one of the following: a specific procedural approach, a deductive approach, a stochastic approach or a search tree approach [17]. These approaches can result in CSPs that can be solved by applying a constraint solver [8]. In case the modeller cannot generate a scene with the existing constraints, its has to deal with over- and under-constrained scenarios. As mentioned by Sanchez et al. [18], modellers can also use a genetic algorithm to learn from the generated scenes that have been scored, and thus improve the quality of scenes to be generated during the next iteration. The scene generation phase involves identification of relation(s) between the given and other objects [10, 14]. It also ensures that the identified relations do not contradict the existing relations provided by the user in the previous phases. In this phase, these relations are used to generate several sets of values for the positions and
orientations of every object in the scene. These values should result in an acceptable scene [10]. Various techniques for scene generation are briefly described in Section 2.3.

2.2.3. **Insight Phase**

The insight phase retrieves all the values for positions and orientations of the objects in a scene and uses stored 3D models to visualize the objects in the scene [19, 18]. The visualization thus created is presented to the user. Once the generated scenes are presented to the user, a complete iteration of scenes generation is performed. Furthermore, the user is allowed to introduced more constraints to the presented scenes. The user identifies any changes required to the scene and specifies the changes as new constraints, which are passed as an input to the scene description phase which starts a new iteration.

2.3. **Techniques to perform Scene Generation**

In this section, we discuss four techniques adopted by well known modellers used to deal with constraints in the scene generation phase. The first three of these techniques are based on CSPs.

As stated by Tsang [20], a Constraint Satisfaction Problem (CSP) is a 3-tuple \( P = \langle X, D, C \rangle \), where:

- \( X \) is a set of variables.
- \( D \) is a domain of values.
- \( C \) is a set of constraints

Every constraint is in turn a pair \( \langle t, R \rangle \) (usually represented as a matrix), where \( t \) is an
$n$-tuple of variables and $R$ is an $n$-ary relation on $D$. An evaluation of the variables is a function from the set of variables to the domain of values, $v: X \rightarrow D$. An evaluation $v$ satisfies a constraint

$(x_1, \ldots, x_n), R$ if $(v(x_1), \ldots, v(x_n)) \in R$ A solution is an evaluation that satisfies all constraints.

As mentioned by Knuth [7], the two main approaches for solving CSPs are backtracking and forward checking. Knuth[7] also stated that in backtracking, a solution is found for some computational problem by incrementally building candidates to the solutions and abandoning each partial candidate $c$ as soon as it is determined that $c$ cannot possibly be completed to a valid solution [7]. Forward checking is an efficient alternative to backtracking [1]. In Forward Checking, a CSP represented in the form of a graph is combined with a filter algorithm.

2.3.1. ORANOS

Kwaiter et. al [9] introduced the Dynamic, Hierarchical and Numerical CSPs (DHNCSPs), technique which is adopted by ORANOS. ORANOS is a well-known general-purpose modeller. In this section, we discuss the scene generation phase of ORANOS. We also discuss the handling of over-constrained problems in detail and briefly describe the solution to the relevant constraint satisfaction problem.
Numerical Constraint Satisfaction Problems

A Numerical Constraint Satisfaction Problem (NCSP) is a 3-tuple $P = (V, D, C)$, where:

- $V$ is a set of numeric variables $\{V_1, .., V_n\}$.
- $D$ is a set of domains $\{D_1, .., D_n\}$, corresponding to the variables $\{V_1, .., V_n\}$, where $D_i$ is an interval $[a_i, b_i]$ of continuous numerical values.
- $C$ is a set of constraints $\{C_1, .., C_m\}$, where $C_i$ is a constraint defined by any numeric relation involving variables.

A solution to an NCSP is an instantiation of variables $V$ such that each variable $V_i$ is assigned a value selected from the corresponding domain $D_i$ and the selected values satisfy all constraints in $C$.

The NCSP technique has two limitations with respect to over-constrained problems: it is too restrictive and it does not allow constraints to be relaxed [8]. In order to deal with over-constrained problems, it is useful either to know whether a given constraint is mandatory (hard) or optional (soft). It is also beneficial to know importance of these constraints because it allows the system to remove the constraints with lower importance.

Numeric Hierarchies Constraint Satisfaction Problems

A Numeric Hierarchies Constraint Satisfaction Problems (NHCSPs) is a NCSP where every constraint has a unique strength associated with it. These strengths allow the system to prefer one constraint over another. Constraints with relatively low strength can be ignored by the system when confronted with an over-constrained scenario.
Furthermore, constraints can be arranged in an order of decreasing strength to form a constraint hierarchy. The constraint hierarchy is represented by $C$, where $C$ is a multiset of constraints $C_0, C_1, \ldots, C_j$, where $C_j$ is a constraint of strength $j$. While $C_0$ has the highest strength, $C_n$ has lowest strength and $C_{j-1}$ is preferred over $C_j$.

To instantiate variables in ORANOS [9], the Forward Checking approach devised for numeric continuous domains by Bacchus and Grove [1] is used. ORANOS sorts the variables of the scenes to be created in an increasing order of their domain object count. As a variable is instantiated, an interval propagation algorithm is applied to reduce the domain length. During this process, if the domain of a variable is deduced to be inconsistent with some constraints, a fresh value is assigned to the current variable. This approach ensures that all the variables encountered so far are consistent and avoids inconsistency checks while instantiating the next variable.

The approach followed by ORANOS deals successfully with over-constrained problems but it does not consider under-constrained problems.

2.3.2. MultiForms

Plemenos [14] introduced a well known general purpose modeller called MultiForms. Like ORANOS, MultiForms also uses the DMHD technique to process the input from the user. Once the input description from the user is accepted, all the relations between the objects are treated as linear arithmetic constraints.

**Constraint Satisfaction Techniques in MultiForms**

While dealing with constraints, it is important to ensure their consistency. If a
given soft constraint is inconsistent with the hard constraints, it then has to either be manipulated or deleted from the list of constraints. MultiForms adopted the arc consistency algorithm to detect inconsistency of constraints. The arc consistency algorithm is applied to object-constraint graphs to ensure the consistency of the related nodes. In such a group, a node represents an object and an arc (edge) represents the relation between two objects. The arc consistency algorithm is applied to check the consistency of the connecting relation between the objects representing nodes.

The arc consistency algorithm is adopted to implement Constraint Logic Programming with Finite Domain (CLP(FD)). In CLP(FD), every variable is of the form “$X$ in $r$,” where $X$ represents a variable and $r$ represents a range or an indexical term. An indexical term represents the range of a different variable that $X$ is related to. The variable $r$ could be specified as the name of a domain, a pair of $[\text{min}, \text{max}]$ values, or a single value. The combination of all primitive constraints results in restrictions on the ranges for the variables. If more than one constraint applies to the same variable, the ranges are intersected to obtain a new range. When a constraint is enforced on an object, all constraints relevant to that object are also checked for consistency. This results in forward checking. Any new restrictions on the object are propagated until a fixed point is reached, or until no more objects are left. If an inconsistency is encountered, the previous state is obtained by performing backtracking.

**Constraint Satisfaction Techniques with Hierarchy in MultiForms:** During the instantiation of position variables for an object in MultiForms, the order in which the objects are encountered plays a key role. This order depends upon the level of the object
in the hierarchy and the number of constraints associated to it. Dealing with constraints and objects in the solution generation phase are factors that MultiForms considers when deciding on the order of the object to be instantiated [14]. These two factors are handled by a two-step resolution process. Firstly, in the constraint adding step, a new range is obtained for every variables referred to in the new constraint. In addition to the obtaining a new range, a check for inconsistency is made by ensuring that every variable has a non-null range. In the second step, the effect of the current constraint on the other constraints is determined.

To improve the resolution process, MultiForms applies techniques such as fail-first, DMHD, and intelligent back tracking [14]. Like ORANOS, MultiForms ignores the case of under-constrained scenarios where the user provides relatively little input to the DSM.

2.3.3. ARCHiPLAN

ARCHiPLAN [11] is a dedicated modeller for architectural design. ARCHiPLAN is implemented using a topological level and Dynamic Space Ordering (DSO). The topological level allows ARHiPLAN to identify and enforce topological constraints between objects. DSO is a heuristic that attempts to improve the efficiency of object placement. As with the other two modellers we have discussed, ARCHiPLAN uses DMHD to handle input from the user.

**Constraint Representation in ARCHiPLAN:** In ARHiPLAN, a topological constraint is specified as an adjacency constraint. The user is required to specify adjacency relations
while describing the scene so that the system can obtain hierarchical constraints. Some example statements about adjacency relations are:

- The room is adjacent to the bathroom.
- The living room is on the south wall of the building.

ARCHiPLAN also identifies other constraints that are either implicit or explicit. *Implicit constraints* are ones that are assumed to be true in all cases without any statements about them, such as the constraints that no two objects can overlap. The *Explicit constraints* are the ones given in the statements, such those stated above.

**Applying adjacency constraints**

Two parameters, $d_1$ and $d_2$, are obtained as a result of applying an adjacency constraint to a pair of objects. The parameter $d_1$ gives the contact length between the two objects and parameter $d_2$ gives the distance between the closest corners of the two objects. By default, $d_1$ and $d_2$ have a minimum value zero. If two objects are in contact, $d_1$ is greater than 0 and $d_2$ is 0; and otherwise, $d_2$ is greater than zero and $d_1$ is zero. This also implies that the overlapping constraints are satisfied by default.

Applying an adjacency constraint also gives rise to a variable called an *adjacency variable*, which is defined over the domain {East, West, North, South}. Instantiation of such a variable leads to propagation, followed by domain reduction by arc consistency.
**Figure 2.2:** Calculating contact length $d_1$ and distance $d_2$ between two objects $O_1$ and $O_2$

**Space Reduction Constraints**

Kwaiter[8] successfully reduced the search space with the help of some constraints. We briefly discuss those constraints in this section.

The *incoherent space elimination* constraint ensures that an object is either directly adjacent (in contact) to another object or far away from it, enough to allow another object to be placed in between them. This constraint is enforced only if the *total_recovery_constraint* is activated as there is no space lost in the building to be constructed.

A *topological reduction constraint* is applicable when an object is placed near a wall. Consider a case where an attempt is made to place an object at the north side of the north wall. The object cannot be placed at the north side of the north wall as this wall is the north boundary of the scene. Therefore the relation ‘north side’ is deleted from the possible relations for the north wall. Although these types of constraints prevent some small problems involving a small range of variables and introduce modularity, they are not ideal for scenes with more than ten objects [2].
2.3.4. MultiCAD with Genetic Algorithm

Sanchez et al. introduced a DSM system named MultiCAD-II, which is a constraint-based modeller [18]. It generates scenes and tests these scenes for consistency of the provided constraints. The testing ensures that all the constraints are satisfied. The constraints reduce the solution space. The two main drawbacks of MultiCAD–II are the lack of user interaction and the extremely long generation time due to its ability to first generate and then test the scenes.

MultiCAD-GA was developed to overcome the disadvantages of MultiCAD-II by adopting a genetic algorithm to replace the generate and test approach. MultiCAD-GA parses the scene description to identify the objects in the scene that are yet to be instantiated and then runs a Constraint Logic Program (CLP) to generate a population of 20 initial scenes. These scenes are presented to the user, who assigns a fitness score to every object in the scene. The cloning crossover and mutation genetic operations are applied to this population to produce a new population, which will be later scored by the user in similar fashion. This process is repeated until the user is satisfied with one of the presented scenes. In this section, we discuss the genetic operations and omit discussing the generation of the initial scenes.

**Chromosome of MultiCAD-GA**

As mentioned in Section 2.2, every object in a scene is represented by a node in the concept tree. Also recall that an object is a three-dimensional (3D) entity with a position and a size associated with it. Every object (or a sub-scene) in MultiCAD-GA is
represented by an axis-aligned bounding box which is a 3D rectangular shape aligned with the X, Y and Z axes. An *axis-aligned bounding box* (hereafter bounding box) can be represented by two three-dimensional values, the position of bottom-front-left corner and the three-dimensional size (width, height and depth) of the box. It is one of the best known data structures for storing the position and volume of a three-dimensional object. The bounding boxes of the subscenes are stored in binary format. Two examples are shown in Figure 2.3.

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>Δx</th>
<th>Δy</th>
<th>Δz</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>2</td>
<td>1</td>
</tr>
</tbody>
</table>

Chromosome for subscene #1

<table>
<thead>
<tr>
<th>x</th>
<th>y</th>
<th>z</th>
<th>Δx</th>
<th>Δy</th>
<th>Δz</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>

Chromosome for subscene #2

**Figure 2.3:** Chromosome representation of a scene with two sub-scenes

In Figure 2.3, the position and size of the two subscenes is represented by chromosomes. In each chromosome, the first three bits represent the value of the x coordinate. Similarly, the other position and size values are stored in binary format to form a complete chromosome. This approach does not consider the orientation of the objects.

**Genetic Operations**

A genetic algorithm is applied to the scenes that are generated using CLP techniques. The genetic algorithm acts on parent chromosomes in order to generate descendant chromosomes. The MultiCAD-GA iteratively generates new scenes by applying genetic operations. The selection of a parent chromosome is dependent upon the
fitness scores assigned to them by the user. The higher the score, the higher is the probability of a scene being selected.

**Cloning:** The cloning operator simply reproduces a parent with a high fitness score.

**Crossover:** The crossover operator is performed on two parents and results in two child chromosomes. Randomly selected coordinates in parent 1 are exchanged for those at the corresponding coordinates in parent 2, as shown in Figure 2.4. While selecting these coordinates, care is taken to ensure that a single coordinate is selected at most once. The number of bits in a coordinate is represented by \( L \). In the example shown in Figure 2.4 \( L \) is 3. The thick vertical lines in Figure 2.4 signifies the length of \( L \). The selected lengths are 2, 0, 0, 2, 0 and 3 bits long. The selected bits can be any contiguous part of the \( L \) bits in a coordinate.

![Figure 2.4](image.png)

**Mutation:** The mutation operation first randomly selects several co-ordinates from the parent chromosome and alters from 0 to \( L \) contiguous bits starting at the coordinate, as shown in Figure 2.5. In the figure, the selected bits are shaded. The number of bits
selected for mutation can vary randomly from 0 to L.

<table>
<thead>
<tr>
<th>Parent</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>0</th>
<th>0</th>
<th>0</th>
<th>1</th>
<th>1</th>
<th>1</th>
<th>0</th>
<th>1</th>
<th>0</th>
</tr>
</thead>
<tbody>
<tr>
<td>Child</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

**Figure 2.5:** Illustration of mutation operation

The process of generating scenes by applying a genetic algorithm is performed until the user is satisfied with the presented scenes. After the generation of a population of scenes, the user goes through the presented scenes and rates every chromosome (object) on the basis of his or her requirements. These ratings are used to determine the fitness of the chromosomes. Once the fitness is determined, the MultiCAD-GA selects the subset of the evaluated scenes probabilistically, with probabilities based on fitness scores.

### 2.3.5. Summary

In this sub-section, we summarise the major differences between the four techniques for scene generation in the modellers described in this section. Table 2.1 presents the main differences.

**Approach:** The solution generation can either be constructive or iterative; both of these techniques have their advantages and disadvantages. The constructive approach requires less processing time because the modeller can present an incomplete solution to the user and thereafter adopt that existing solution. The iterative approach has to generate a complete scene satisfying all the user descriptions at once. To do so, it must resolve inconsistencies and incompleteness in the input description. While the constructive
approach requires more input from the user, it is more likely to present the scene that best
matches the user's expectations.

**Assured Results:** Two of the four surveyed modellers guarantee that at least one solution
will be produced while the others do not. The ones that assure a result may ignore some
constraints. Their advantage is that they always provide suggestions to the user while the
others may fail to do so.

**Algorithms:** All four discussed DMSs have different approaches to dealing with issues
that arise during the generation of scenes. Section 2.3 briefly discusses the working of the
four DSMs and the various algorithms adopted by them.

**Orientation:** The possible orientations for the objects depend upon the purpose of the
modeller. General purpose modellers are built after considering a wide variety of
domains (fields) and hence the objects are expected to have high flexibility across all the
three axes, as is provided by 24 isothetic orientations (considering all the three axes). The
dedicated modellers are expected to provide the orientation flexibility needed for their
domain. For instance, in the field of architecture, objects are not expected to rotate
around all three axes. In particular, rotation around the Y axis is sufficient.

**User Input:** One goal of DSM system is to reduce the amount of input required from the
user. The user is allowed to provide additional descriptions after the first set of scenes is
presented by stating the exact corrections to be made on the presented scenes.

**Number of solutions produced:** In the insight phase, MultiCAD-GA presents 20 scenes,
while the others present a single scene. The disadvantages of presenting only one scene
per iteration are:
i. The risk of rejecting scenes similar to the desired scene is high.

ii. The modeller does not learn as much about the user's preferences in a single iteration as compared to the many scene approach. Hence this approach may requires more iterations to satisfy the user.

On the other hand, the disadvantages of presenting many scenes are:

i. The user has to evaluate many scenes after each iteration.

ii. The risk of having unfair scores is high since the user may give inconsistent rating to the scenes.

**Table 2.1:** Comparison of various DSM techniques

<table>
<thead>
<tr>
<th></th>
<th>ORANOS</th>
<th>MultiFormes</th>
<th>ARCHiPLAN</th>
<th>MultiCAD-GA</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Approach</strong></td>
<td>constructive</td>
<td>constructive</td>
<td>constructive</td>
<td>iterative</td>
</tr>
<tr>
<td><strong>Assured Results</strong></td>
<td>no</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td><strong>Algorithms</strong></td>
<td>DHN-CSP, Forward Checking</td>
<td>Forward checking + CLP, Intelligent backtracking, AC5-Filtering, Dynamic heuristic</td>
<td>Branch and Bound (optimization) and DSO</td>
<td>CLP, Genetic Algorithm</td>
</tr>
<tr>
<td><strong>Orientation</strong></td>
<td>24 isothetic</td>
<td>1</td>
<td>0° or 90°</td>
<td>Around the Y axis</td>
</tr>
<tr>
<td><strong>User Input</strong></td>
<td>Constraints</td>
<td>Constraints</td>
<td>Constraints</td>
<td>User score</td>
</tr>
<tr>
<td><strong>Number of Solutions Produced</strong></td>
<td>Only 1</td>
<td>at least 1</td>
<td>at least 1</td>
<td>0 to 20</td>
</tr>
</tbody>
</table>

\( n^* \) - \( n \) is a finite set.
2.4. Furniture Layout Problems

Recently developed automatic furniture layout techniques, such as Interactive Furniture Layout [12] and Automatic Optimization of Furniture Arrangement [23], consider various relations between pieces of furniture in a scene and produce an improved arrangement of furniture in a scene. Automatic furniture layout accepts a scene with randomly placed objects. It suggests various furniture arrangements by following some furniture placement guidelines. In this section we briefly discuss two automatic furniture arrangement techniques, namely Interactive Furniture Layout [12] and Automatic Optimization of Furniture Arrangement [23].

2.4.1. Automatic Optimization of Furniture Arrangement

The Automatic Optimization of Furniture Arrangement system performs furniture arrangement by considering the hierarchical and spatial relationships of furniture objects in the scene [23]. To obtain these relationships, it considers a variety of ergonomic factors. This system has adopted a two-stage technique to automate furniture arrangement. In the first stage, it accepts a set of example scenes and extracts spatial relationships from them. In the second stage, it accepts a scene with randomly arranged furniture and produces an improved arrangement. The improved arrangement is obtained by enforcing the relevant relationships, which are extracted during the first stage. The extraction of relations is done only once, but the second stage can be performed repeatedly.
Furniture Relationship Extraction

The extraction of relations is done in two stages. Firstly, obtaining the required object attributes from the example scenes. Secondly, learning prior relationships between objects. The objects in a scene are represented by bounding boxes. In addition to a bounding box, an object has various attributes that contribute to the extraction of relationships from an example scene.

Back Surface: The back surface is the surface of the bounding box that is closest to a wall excluding the top and the bottom surfaces. The three remaining surfaces are called non-back surfaces. The back surface is used as a reference when setting values for attributes such as the relative orientation to the wall and the distance from the wall.

Position and Orientation: The position vector stores the position of the center of the object with respect to the X, Y and Z axes in the given scene. The orientation of the object with respect to the nearest wall is recorded as a single angle of rotation around the Y axis.

Accessible Space: An accessible space for a surface represents the space near an object that has to be vacant for that surface of the object to be accessible. Separate accessible spaces are identified from the objects in the example scene.

Viewing Frustum: The viewing frustum represents the space in front of the front surface of the object. The front surface of an object is the one opposite from its back surface. The viewing frustum is represented by a sequence of rectangles $V_{i1}$, $V_{i2}$, ... with half diagonals $Vd_{i1}$, $Vd_{i2}$, ... which start from the front surface and extend outwards, until the available
space is exhausted or a count of four is reached, as shown in Figure 2.6.

![Figure 2.6: A viewing frustum associated with object A](image)

**Learning Prior Relationships**

Once the required attributes are retrieved from the objects, relationships that provide positive instances are obtained. Some of the relationships that are retrieved are:

**Spatial Relationships:** The distance of an object from its nearest wall as well as its relative orientation to the same wall together represents a spatial relationship.

**Hierarchical Relationships:** If an object \( A \) is a *part of* or *on* or *in* object \( B \), then \( B \) is said to be the *parent* of \( A \) (and \( A \) is the *child* of \( B \)), resulting in a hierarchical relationship [14].

**Pairwise Relationships:** During the process of object placement in real life, we come across many objects that are placed relative to other objects. Such objects are commonly used with each other to fulfill a common purpose. For instance, chairs are often positioned relative to a tables. Each relationship between a chair and a table is a pairwise relationship.
In the second phase of the Automatic Optimization of Furniture Arrangement system, many translation and rotation operations are performed iteratively on the furniture objects. The process stops when a permissible furniture arrangement with a high heuristic score is obtained. Swapping of objects is considered to be a type of translation.

Whenever a door is defined in a scene, the system has to ensure that there is a path from this door to every other door and a path from this door to every other object in the room.

2.4.2. Interactive Furniture Layout Using Interior Design Guidelines

The Interactive Furniture Layout method [12] accepts a scene with randomly placed furniture and suggests various other arrangements. While suggesting arrangements, this system considers the following main features of the furniture objects in the scene:

**Circulation:** *Circulation* ensures that every object in the scene is accessible. This property is evaluated by computing the free configuration space of a person on the ground plane of the room.

**Clearance:** The *clearance* of an object is the space required to comfortably use it. Some surfaces of an object may have to be accessible while others are not. For instance, the front surface of a TV has to be accessible, while the back does not. This space may not have to be of uniform size around all six surfaces of the object. A Minkowski sum is used to find the clearance space for each accessible surface [12].

**Conversation:** The *conversation* property ensures that furniture objects that will be used
by people having conversations should be placed close to each other. These objects should be rotated to directly view each other.

**Visual Balance:** The most widely known principle of visual composition is visual balance. The *visual balance* of an object is directly proportional to the size of the object. The principle is to place the mean of the distribution of visual weight at the center of the composition.

**Alignment:** The *alignment* ensures that the orientation of a furniture object is relative to all objects that it is paired with and to the wall around the room.

**Emphasis:** The *emphasis* property of furniture arrangement ensures that visual activities take place without stressing the eyes. For instance, watching TV while sitting on a sofa should occur without much stress on the eyes of the person sitting on the sofa.

Along with these properties, this system also implements pairwise relationships which work similarly to those described in Section 2.1.

### 2.5. Differences between DSM and Automatic Furniture Layout Techniques

Table 2.2 shows the differences between declarative scene modelling and the two automatic furniture layout techniques discussed in Section 2.4.

**Table 2.2:** Differences between DSM and automatic furniture layout techniques

<table>
<thead>
<tr>
<th>Factor</th>
<th>Declarative Scene Modelling System</th>
<th>Automatic Furniture Layout Techniques</th>
</tr>
</thead>
<tbody>
<tr>
<td>Input</td>
<td>natural language description</td>
<td>randomly generated scene</td>
</tr>
<tr>
<td>Knowledge Base</td>
<td>predefined</td>
<td>predefined;</td>
</tr>
</tbody>
</table>
The similarity and differences are presented with respect to six factors: input, knowledge base, types of constraints, the presence of the hard constraints, optional objects and purposes.

**Input:** A DSM system accepts a description in a natural language, while an automatic furniture layout technique accepts a scene with randomly placed objects. In both cases, the objects will later be organized to form a scene that satisfies constraints required to make an acceptable scene.

**Knowledge Base:** All DSM systems have a stored set of predefined rules in their knowledge bases. Some automatic furniture layout techniques accept an example scene from the user and then retrieve the desirable features from it. With predefined rules, the user is not required to provide an example scene. On the other hand, the modeller will have fewer details available about desirable scenes.

**Constraints:** While generating a scene, the modeller considers hierarchical and user defined spatial; visual constraints permitted after first iteration.

<table>
<thead>
<tr>
<th>Type of constraints</th>
<th>hierarchical; user defined</th>
<th>hierarchical; spatial; visual</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard constraints</td>
<td>yes</td>
<td>permitted after first iteration.</td>
</tr>
<tr>
<td>Optional objects</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Purpose</td>
<td>general purpose; domain specific</td>
<td>domain specific</td>
</tr>
</tbody>
</table>

The similarity and differences are presented with respect to six factors: input, knowledge base, types of constraints, the presence of the hard constraints, optional objects and purposes.
defined input and ignoring other features, such as spatial constraints. The Automatic Furniture Layout systems consider the spatial constraints but not the hierarchical constraints. In most cases, a user will expect the modeller to have knowledge about spatial constraints such as accessibility and visibility. Adding such constraints to the knowledge base of a modeller will not only reduce amount of input from the user, but will also make the modeller appear more intelligent.

**Hard Constraints:** A DSM system allows the user to provide hard constraints as input, while automatic furniture layout techniques do not accept any hard constraints during the first iteration. The advantage of having hard constraints is that the modeller can consider these constraints before generating the scene and then produce scenes by satisfying these constraints. If no hard constraints are available, additional iterations will be required. The disadvantage is that since the user may not have clear idea of the scene to be generated, he or she may provide inconsistent hard constraints.

**Optional Objects:** Since automatic furniture layout techniques accept scenes with randomly placed objects, they are restricted to improving the positions of the objects in the scene without suggesting any objects be added to improve the scene.

**Purpose:** As mentioned in Section 2.1, a domain specific modeller has more knowledge of a particular domain and therefore is more efficient that a general purpose modeller for individual domains [6].

Recalling the discussion of some of the well known DSM systems and automatic furniture arrangement, we notice that the existing DSM systems, such as ARCHiPLAN
and ORANOS accepts the majority of constraints for scene generation from the user as scene descriptions. This approach may lead to over constraint problems, which are addressed by the existing DSM systems. On the other hand, some of the well known automatic furniture arrangement techniques accept either no initial input description or a sample positive scene. In the case where no initial input description is accepted from the user, the number of iterations to generate a furniture arrangement for a desired scene can drastically increase. The automatic furniture arrangement techniques that accept a positive sample scene as initial input expect the user to prepare a complete positive scene but the user may not have a clear idea of all constraints in the scene. To overcome these problems, we have designed a system called Procrustes that accepts a partial scene description and extracts the remaining constraints from a knowledge base. The knowledge base should have information about the domain of the scenes to be generated. In addition to the information stored by the existing modellers, Procrustes stores the purpose and possible relations such as purpose, visibility and accessibility.

2.6. Discussion

Procrustes also attempts to reduce the amount of information a user has to provide for the presented scenes. Recall that the existing DSM and automatic furniture layout systems accept additional constraints or scores to evaluate the presented scenes. Although introducing additional constraints helps the modellers, the user may not always know the exact constraints. If user must assign a numeric score to it, he/she has to be needs to examine them all and qualitatively assess their relative merits to ensure that the scenes and the underlying relations are scored appropriately. To avoid these issues, an additional
phase has been introduced which accepts positive, negative or neutral response from the
user and uses this response to generate scenes that are closer to the desired scenes.
Chapter 3

Scene Generation Techniques

Majority of the existing DSM systems have the ability to create scenes by accepting input that describes the positions of all objects in the scene. Although the user is not expected to provide the spatial coordinates for every object, he or she is expected to describe the position of every object in the form of natural language by relating the current object to other objects. Hence, the majority of the DSMs concentrate on heuristics to generate scenes by handling over-constrained descriptions. The user may not always have a clear idea of the relations describing the positions of every object in the scene.

Procrustes performs scene generation similar to a majority of the existing DSM systems. The main differences are that Procrustes accepts a partial description and reconsiders the predefined sizes for the objects in the scene. It also generates a hallway by considering predefined properties of the object and the empty spaces in the house. As the last step of scene generation, it places objects within the rooms. Procrustes also performs a multilevel scene generation. The type of scene it generates has a house at the top level, rooms and hallways at the second level and furniture at the third level.

Procrustes explores several options available to the user and attempts to generate scenes close to the desired scene over many iterations. To generate such scenes,
Procrustes uses a heuristic method that accepts a partial description of a scene. If the partial scene description includes descriptions of the positions of certain objects, then Procrustes derives the positions of the remaining objects by considering the described objects and the facts from the predefined knowledge base. Procrustes iterates through this approach as it learns from the choices made by the user.

Figure 3.1 demonstrates the complete process of scenes generation. It also describes the flow of information and control over the events. As mentioned in Section 2.3, the knowledge base stores the required information, including the concept tree. The knowledge base for Procrustes is shown in Appendix A. The concept tree is one of the two inputs to the scene description phase, the other being the partial scene description provided by the user. The user description is parsed to identify the objects and relations in it, as explained in Appendix B. Using the relations and objects, description tree is produced by augmenting the concept tree with information extracted from the description. This description tree is passed as an input to the scene generation phase. In this phase, the description tree is traversed by assigning a position to every encountered object to form a scene tree. Assigning a position to an object involves processing the relations associated with the object and extracting required relations. The assigned positions of room objects are re-evaluated to ensure that the space in the generated scene is well utilized. At last the scenes tree are produces as an output from the generation phase.

Figure 3.1 shows the three phases and the flow of control as implemented in Procrustes system. In the implementation, these three phases correspond to three
modules, as shown in Appendix C. Section 3.1 describes the input to the scene generation phase in Procrustes. Section 3.2 explains extraction of the required relations from the knowledge base. Section 3.3 describes the algorithm that updates the empty spaces after the placement of every object. Section 3.4 describes a two-phase method for removing extra empty spaces from a scene. Section 3.5 describes a procedure for generating a hallway. Section 3.6 briefly describes the steps involved in placing furniture within a house.
Figure 3.1: Input and output for scene generation phase
3.1. Scene Generation Input

The generation phase accepts a description tree as input, produced by the
description phase, and the knowledge base. The knowledge base contains information
about the domain of scenes to be generated. The relations in the description tree are
treated as constraints during scene generation. These relations remain constant for all
scenes unless conflicts are encountered. Since the input from the user is a partial scene
description, relations in the description tree are also allowed to be insufficient to generate
scenes. The remaining necessary relations to generate scenes are extracted from the
knowledge base. The knowledge base provides possible relations between objects. These
possible relations are used as constraints during the placement of the objects.

The input description tree is cloned to form a scene tree that is traversed in a
breadth-first manner. During the traversal of a scene tree, relation extraction has to take
place for the objects that do not have relation associated to them. Constraints on objects
at level one include the constraints considered during floor planning, such as the
utilization of space and the placement of the hallway. Constraints on objects at level two
include interior design constraints. By satisfying these constraints, the modeller obtains
3D size, position and orientation values for every object in the scene. The output from the
generation phase is a list of scene trees, each of which has position and orientation values
for every object.

During the instantiation of position and orientation variables [2], care must be
taken to ensure that all objects have appropriate spatial relationships [23, 12] with the
objects to which they are logically related, that no object overlaps any other object and finally that all objects are accessible. An object is accessible when a path exists between it and each of its sibling objects. Since a scene may involve many objects, each with a large range of possible positions and sizes, we use constraints to reduce these ranges. We also restrict the possible values in a range to discrete values at some level of granularity, rather than considering the infinite number of possible real values. Every constraint in the description tree is treated as a hard constraint. Relations in the knowledge base serve as soft constraints during the scene generation phase and also suggest ideal placements of objects.

3.1.1. Knowledge Base

While generating a scene, a modeller should have knowledge about the domain of the scene. As mentioned in Section 2.2, some of the existing well known modellers have hierarchical knowledge of every object that can be incorporated in a scene. Thus, the modeller is aware of the parent object of every object added to a scene. However, with these modellers, it is the responsibility of the user to provide relations that help the modeller place the object with respect to its parent. If an object does not have any relations associated with it, it cannot be placed.

To simplify scene generation, one can imagine various types of information about the relations between objects being stored in the knowledge base. Using such information, the modeller will obtain constraints to restrict the number of positions to place an object. The modeller can also provide ideas to the user that will help him or her
to consider accessibility and visibility relation, which are described shortly.

Recall that the knowledge base also stores the purpose of every object in the scene [6]. The purpose is stored over two levels of details. The least descriptive purpose of an object depends upon its utilization in private and public activities in a house. Storing the purposes of an object also allows the modeller to place objects with similar purposes close to each other in the scene. For instance, a bookcase and a study chair may both have “studying” as their purpose. In the absence of any conflicting constraints, they can be placed together in a “studying” group. The purposes for objects are domain dependent.

Accessibility

The idea of introducing properties such as accessibility in the field of automatic furniture arrangement was suggested by Merrell et al. [12]. An object is accessible if it is reachable from one or more objects. For instance, rooms in a house should have doors that provide access to them. The knowledge base stores the accessible sides of the objects. For instance, any one of the four vertical surfaces of a room can be assumed to be the accessible side, but for a TV, only the front surface should be accessible. Accessibility relations play a major role when the modeller is suggesting better placements to the user.

Some of these accessibility relations are specific while others are generic. The specific relations help the modeller determine a relatively small number of positions for objects. For example, if there is a TV and a sofa in a living room, accessibility relations are used to state that the TV should be placed in front of the sofa or and vice versa. A
generic relation provides a list of relations for the placement of objects involved. One of these relations is selected by satisfying all the other constraints on the objects during the scene generation phase [22]. For example, the dining room is placed close to the kitchen. The \textit{close-to} relation allows the dining room to be placed with a shared wall with the kitchen or on the other side of the hallway from the kitchen.

The accessibility relations suggested by Merrell et al. [12] have details for a single level of objects which are in a room. We extend this idea to accommodate two levels of objects. We apply appropriate domain knowledge to the given hierarchical level. For instance, the rooms in a house should be accessible from each other but may not have to be visible to each other. Such information is saved in the form of Prolog facts, as shown in Appendix B.

\subsection{Description Tree}

A \textit{description tree} is a concept tree augmented with spatial relations obtained from the input scene description. A node in a description tree represents an object in the scene and an edge represents a spatial or hierarchical relation between two objects, as shown in Figure 3.2.
A scene tree is a description tree with augmented relations that were extracted from the user descriptions. Scene trees are built during the scene generation phase by cloning the description tree and adding sufficient relations to generate a scene. During the traversal of a scene tree, when an object is encountered, its child objects are sorted according to the relation associated to it. A child object that is connected to the parent object by spatial relations is given the highest precedence. Among the remaining child objects, the ones with more descriptive spatial relations go first. The remaining child objects which have no relations associated with them, are given the lowest precedence. Extraction of relations from the knowledge base and processing the relations associated with an object in the scene occurs during the traversal of a scene tree. Traversal is performed in a breadth-first fashion. During the traversal, a complete scene tree is built. In the next section we discuss extraction and processing of relations.
3.2. Relation Extraction

Recall that the relations in the description tree might not be sufficient to place all the objects. *Relation extraction* is formed to replace all generic relations by specific relations and to obtain additional relations, as necessary. Relation extraction occurs whenever an object that does not have a relation associated to it is encountered. Relation extraction considers the existing relations associated with other objects and the knowledge about the purpose of the object. For instance, if a kitchen has a relation and the dining room has no relation associated with it and these two objects share a common purpose, a generic relation between the dining room and the kitchen will be present in the knowledge base. As mentioned by Hepting [5], such generic relations can be replaced by specific relation. A maximum of four specific relations (left, right, front and behind), will replace them during the description tree traversal. Processing such a specific relation will result in various positions for the objects implied by the relation. The available empty spaces are considered while replacing the generic relations by specific relations. As a result of which, a maximum of four possible relations can be obtained. In the case where multiple specific relations are obtained, they are used to create multiple scene trees by saving each specific relation in a cloned scene tree.

Each scene tree results in a different scene. Figure 3.3 shows two of the scenes partially generated from the description tree in Figure 3.2. The scenes are a result extracting a relation for the living room and extracting a specific relation for the bedroom. After a relation is extracted, it has to be processed to obtain a position across
the three axes.

![Figure 3.3: Generation of scenes with various relations](image)

### 3.3. Box-the-Space technique

Recall that every relation between a pair of objects has to be processed. Also recall that the existing empty spaces in the scene are considered when processing a relation. To obtain the relevant empty spaces for a relation, we have developed a technique called *Box-the-Space*.

The Box-the-Space technique uses bounding boxes to represent empty space. It uses these bounding boxes to maintain information about the void areas in the scene. Information about the void areas is needed to determine whether objects are accessible and visible to other objects in the same hierarchical level scene. Initially, we assume that the empty space available for the scene is an axis-aligned rectangular parallelepiped called a *bounding box*. It is represented as the root object in the scene tree. In the context of this thesis, it is the area occupied by the house object.
3.3.1. Update Empty Space

After placing an object in the scene, the available spaces for remaining objects have to be updated. Initially the entire root object is considered as one large empty space. As the child objects are placed, the empty spaces within the parent object have to be updated to ensure that the area occupied by the current child object is removed from the list of empty spaces.

Algorithm: updateEmptySpace

Input: direction, objectBB, emptySpaces

Output: emptySpaceInFront

begin
If (direction == “inFront”)
    extendedBB = objectBB.Minimum + new Vector(0,0,1);
    Foreach (space in emptySpaces)
        If (extendedBB.Intersects(space))
            emptySpaceInFront.Add(space);
        end If
    end Foreach
end If
end

The update empty spaces stage removes the volume of empty space that intersects a freshly placed object in the scene. If an empty space $s_i$ intersects with the bounding box of object A, then the intersected volume has to be separated from the empty space $s_i$. This process is called as difference. To separate the intersected volume from $s_i$, it is divided into multiple smaller spaces, excluding the intersected area as shown in Figure 3.4(a). The number of empty spaces that are created depends on the number of intersecting edges of the object. An object can intersect one or more empty spaces.
Consider the scene from Figure 3.4(b), after the placement of the living room object, empty space $s_5$ is divided to form two empty spaces $s_5$ and $s_6$ as shown in Figure 3.4(c).

![Diagram showing empty spaces and objects](image)

(a) Difference of object A empty space $s_i$
(b) Existing scene
(c) Difference of from living room from empty space $s_4$ and $s_5$

**Figure 3.4:** Updating the empty spaces after placing the living room object

In the example scene shown in the Figure 3.4(b), the placement of the living room object has resulted in a different set of empty spaces. This is the result of the Box-the-Space technique which updates the available empty spaces after the placement of every individual object.

### 3.3.2. Advantages of the Box-the-Space technique

As soon as an object is placed, empty spaces are updated. This allows Procrustes to keep track of the current spaces in the scene.

**Avoids collision checking:** Since Procrustes places objects only in available empty spaces, collision checking against previously placed objects is not needed.
Allow accessibility relations: The available empty spaces are provided as input to the accessibility computation which reduces the calculations required to determine the empty space.

3.4. Processing a relation

After the relations are obtained, they have to be processed to produce the 3D positions. Processing relations includes steps such as checking for conflicting relations, available empty spaces and the size of the object. As a result of processing a relation, a position is obtained that satisfies the other relations. The resultant position also depends on the available empty spaces in the scene. The first step in processing a relation is to obtain the relevant spaces for the relation. The space for a relation could be a single empty space or a list of empty spaces that fit the current object. An operation is performed to retrieve the empty spaces for a given direction. This operation involves obtaining immediate empty space at the given direction. The total length and width of these empty spaces should be greater than or equal to the dimension of the object to be placed as shown in Figure 3.5.
Consider the example scene from Figure 3.4(b). If an attempt is made to place the living room object in front of the bedroom object, the front side of bedroom is extended by one unit and $s_5$ as the empty space.

In the next step, we will have to check for the contiguity of the empty spaces for the object to be placed. This step includes obtaining a list of contiguous spaces. A contiguous space element consists of a list of empty spaces which share one of their edges with other empty spaces in the list. After obtaining a list of contiguous spaces, a simple filter operation is performed to obtain the contiguous spaces that have sufficient area to hold the object to be placed. This step is performed by the getContiguousBlocks algorithm. Placing an object either in front of or behind an object requires checking the contiguity of the selected empty spaces parallel to the X axis. Similarly, placing an object
to the left or right of an existing object requires checking for contiguity parallel to the Z axis. This algorithm demonstrates the check for contiguity across the X axis. Figure 3.5(b) shows the result of successfully finding contiguous spaces to place the living room object.

**Algorithm: getContiguousBlocks**

**Input:** emptySpaces  
**Output:** contiguousBlocks  

begin  
contiguousBlocks = [ ]  
emptySpaces.sort() //sort by the X coordinate values  
blockCount = 0  
For i = 0 to emptySpaces.Count - 1  
If (emptySpaces[i].Maximum.X != emptySpaces[i+1].Minimum.X)  
    blockCount = blockCount + 1  
    contiguousBlocks[blockCount].Add(emptySpaces[i])  
end For  
end

During the extraction of empty spaces for the front and back side of an object, the empty spaces are sorted by their minimum X coordinate. If the maximum X coordinate of the current empty space is the same as the minimum for the next empty space, they are said to be contiguous. As long as the empty spaces are contiguous, we keep adding them to a block of empty spaces in the same list. If they are not contiguous, a new list is started. This ensures that the left or the right edge of all the empty spaces are touching the edge of another empty spaces in the list. As a result we will have a list of contiguous empty spaces. This algorithm also ensures that all the added blocks are bigger than the dimension of the object.
The Figure 3.5(a) shows a single empty space, in front of the bedroom object. As the width of the selected space, $s_4$ is equal to the width of living room, the living room can be placed in front of the bedroom. In the Figure 3.5(a), an attempt is made to place the living room to the left side of the bedroom. The two empty spaces, $s_1$ and $s_3$, which intersect the extended left side of bedroom, are obtained as the current empty spaces. Since the current side is left, the empty spaces $s_1$ and $s_3$ are sorted based on their Z coordinates. This will result in a list of empty spaces with $s_1$ as the first element and $s_3$ as the second element. Since the maximum Z coordinate of $s_4$ is not equal to that of $s_3$, two different blocks of contiguous spaces are produced. If the length of neither of the contiguous blocks is greater than or equal to that of living room, it cannot be placed to the left of the bedroom.

3.5. **The Space Reduction Algorithm**

Recall that a room object is placed in the house object based on its relations to its parent or sibling objects. The dimensions of the house are fixed and the whole house is initially considered as the available empty space. After the placement of room objects, the empty spaces in the house make the room objects appear to be scattered in the house, as shown in Figure 3.6(a). The example scenes shows the room objects that have a position associated to them after the relations associated them are processed. Let us assume that the scenes has been generated by processing the following relations:

- room A is to the back-left corner of house.
- room B is to the right of room A.
- room E is in front of room B.
- room C is to the back-right corner of house.
• room D is to the front-left corner of house.
• room G is to the front-right corner of house.
• room F is to the left of room G.

To make a scene appear more realistic, the spaces between the room objects should be in the form of a hallway connecting the room objects. It is assumed that the hallway consists of a series of straight segments connected at right angles. For the example scene shown in Figure 3.6(a), after the empty spaces have been reduced and the hallway has been generated, a desired arrangement is generated as is shown in Figure 3.6(b).

(a) Scattered room objects in a house  (b) Room objects interconnected by a hallway

Figure 3.6: Effects of space reduction algorithm

The space reduction algorithm reduces the wasted space in a house scene by expanding the scattered room objects and connecting them with a hallway. This algorithm is applicable only to the objects in the first level of the scene tree, which are the room
objects.

This algorithm works in two phases, space deletion and object expansion. In the space deletion phase, a list of selected empty spaces is deleted and the remaining empty spaces and objects are relocated to conceal the deletion of the empty spaces. In the object extension phase, the room objects are expanded whenever possible to reduce the area of the empty spaces remaining from the previous phase.

3.5.1. The Space Deletion Phase

The space deletion phase consists of two stages of the same algorithm, one using the X axis as the reduction axis and the other with the Z axis. A scene tree is provided as an input to this phase. The goal of this phase is to obtain strips of empty spaces that extend from one end of the house to the other, as shown in Figure 3.7(a). Let us consider the case where the reduction axis is the X axis, which will result in a house object with a possibly reduced width. The first step is to obtain the empty spaces at the back side of the house object, encountering the empty spaces from the back edge and moving towards the front edge of the house. The empty spaces encountered in this phase match one of the following three cases:

1. Its length is equal to the length of the house object,
2. Its length is less than the length of the house object and it has spaces in front of it, such that the total length of spaces equals the length of the house,
3. Its length is less than the length of the house object and the total length of the space and any spaces in front of it are shorter than the length of the house.
(a) Available spaces for deletion

(b) Generation of space su' from space s and space u

(c) After deletion of space r and space su'

(d) After deletion of spaces using the Z axis as the reduction axis

Figure 3.7: Selection of spaces for deletion with X as reduction axis
In the example scene shown in Figure 3.7(a), spaces r, s and t are selected because they border the back side of the parent house object. Since the length of space r is equal to the length of the house object (first case), it can be deleted from the scene. Since the length of space s is less than the length of the house and repeated extraction of consecutive empty spaces in front of space s results in a sequence of spaces with a total length equal to the length of the house object (second case), space u is obtained as a space in front of space s. A new space su' is created by combining part of space s and all of space u. Its width is equal to the width of space u. This width is less than the width of space s as shown in Figure 3.7(c), its length is the sum of the length of s and u. In the example, the length of su' is equal to that of the house object, so it can be deleted from the scene as shown in Figure 3.7(c). If the third case applies, nothing more is done. For example, space t cannot be extended any further because it does not have any neighboring spaces in front of it. To complete the Space Deletion algorithm, the same algorithm is repeated using the Z axis. The resulting scene will have a possibly further reduced empty space, as shown in Figure 3.7(d).

After reducing the width and length of the house object, the empty spaces still appear to be scattered within the house, as shown in Figure 3.7(d). As described in the next section, an attempt is made to delete any scattered spaces by increasing the length and width of selected room objects.
3.5.2. **Object Expansion**

Recall that the objects in the scene have a predefined dimension to help in the initial process of scene generation. The dimensions of the room objects are now reconsidered to fill the remaining empty spaces in the scene. Since there are multiple possible ways to extend the room object in a scene, multiple intermediate scenes are generated from each given scene [19]. Every scene can result in eight intermediate scenes, each of which are generated with respect to one of the surface sides of the house object. These eight scenes will have left, right, the adjacent sides are considered to be front or behind. Two scenes are created with a single starting side because they select a single adjacent side at a given time. For instance, if the starting side is left, front and behind are the considered as the adjacent sides. After the execution of object expansion algorithm, the intermediate scenes with minimum empty space volume is returned as the result of Object Expansion.

This object expansion phase accepts a scene tree as inputs and results in a scenes tree with possibly expanded room objects. This algorithm consists of the following steps:

1. Repeat steps 1 to 12 with a fresh starting direction $d$.
2. Select the room objects, $O_d$, which have their edge on the starting direction $d$. The variable $l$ represents the number of selected objects.
3. Select the empty spaces objects $S_n$, which have their edge on the starting direction $d$. The variable $n$ represents the number of selected spaces.
4. Find the total width or length that has to be shared between the objects for expansion.
\[ T_v = \sum_{0}^{n} Sd_{vj} \]  

- \( T_v \) is the total dimension that will be shared between the selected objects.
- \( v \) is the selected dimension of the space \( S \) on the axis adjacent to the starting side \( d \). If the starting side is back or front, the width of space \( S \) is selected as \( S_v \). If the starting side is left or right, the \( S_v \) will be the length of the space.
- \( n \) is the number of empty spaces selected to be removed.
- \( j \) is the index of the current space element.

The summation operation adds the non-overlapping dimension of the selected empty spaces

5. Expand the dimension \( x \) for every selected room object, \( O_{dj} \) by \( T_v/l \), where possible.

6. Update existing empty spaces to reflect the expanded dimensions.

7. Calculate the expansion in the current run, \( E_r \) as shown below:

\[ E_r = \sum_{0}^{l} O_{dk} \text{ where } k \rightarrow 0 \text{ to } l \]  

[1] \( l \) is the number of selected objects.

[2] \( O_{dk} \) is the expanded dimension of object \( O \).

[3] \( r \) is the count of object expansion runs on the same set of selected object.

[4] \( k \) represents the index of current object \( O \).

8. If \( E_r \) is less than \( T_v \) OR \( E_{r-1} \) is \( > E_r \), repeat step 4 to 7 with \( T_v = T_v - E_r \).

9. Note the visited objects and select the next set of objects and empty spaces, moving
towards the opposite side of the selected starting side $d$.

10. Repeat the process until all objects have been visited.

11. Repeat Step 1 to 7 with one of the adjacent sides.

12. Add the resultant scene to $H$, a list of scene with expanded object.

13. Select the scene with least empty space from $H$ as a result of object expansion.

Consider a house object with several empty spaces and room objects, as shown in Figure 3.8(a). If we select back side of the house object as the starting side for Object Expansion, this will result in a traversal of objects starting from the back side of house object to the front, while expanding the room objects as they are encountered. Since we have selected the back side as the starting side, space $r$ and $s$ are considered as the selected spaces. $T_s$ of the selected spaces is calculated by adding the non-overlapping widths of the selected spaces $r$ and $s$ as shown below:

$$T_{\text{width}} = r_{\text{width}} + s_{\text{width}}$$

Initially the room objects at the back side are selected, In the example, the room objects A, B and C are selected for expansion. Since objects A, B and C can be extended by $T_{\text{width}} / 3$ they are expanded.
After the expansion of the selected object, *encountered point* is marked to obtain next set of object for expansion. For example, if the starting side is left, then the initially encountered point is the minimum X coordinate of the house object. The encountered point is adjusted as the rooms are processed. Continuing the above example, after a room is processed, the encountered point is increased to the right side of the room. The updated encountered point is selected as shown in Table 3.1.

**Table 3.1:** Values for the encountered points for various directions

<table>
<thead>
<tr>
<th>Starting side</th>
<th>Sort selected object list O based on (in ascending order)</th>
<th>Element to be selected from object list O</th>
</tr>
</thead>
<tbody>
<tr>
<td>left</td>
<td>Maximum.X</td>
<td>first</td>
</tr>
<tr>
<td>right</td>
<td>Minimum.X</td>
<td>last</td>
</tr>
<tr>
<td>back</td>
<td>Maximum.Z</td>
<td>first</td>
</tr>
<tr>
<td>front</td>
<td>Minimum.Z</td>
<td>last</td>
</tr>
</tbody>
</table>
Consider the example scene shown in Figure 3.8, as the length of room object A, B and C are same, the encountered point will be the maximum Z coordinate of one of these room objects. The selected room objects are then added to a list of encountered room objects. As these objects are already expanded to the limit and cannot be expanded anymore, they will be excluded during future expansion on Z axis.

The algorithm then selects the next set of objects and empty spaces, excluding the visited room objects. The next set of objects will be selected by obtaining the objects that intersect the encountered point. If no objects are selected, then the encountered point is updated again. The encountered point is updated with the next corresponding element from the Table 3.1. From our example, the room objects D, E, F and G are expanded, as shown in Figure 3.9(a).

Once the encountered point reaches the limit of the house object, one of the adjacent sides is selected to repeat the process, resulting in a scene with further expanded room objects. This scene will be one of the eight intermediate scenes. In the example, both the right and left adjacent sides will result in the same scene, as shown in Figure 3.9(b).

Once the expansion is done, the room objects are attached to each other with not connecting hallway. In the next section we discuss an algorithm to generate a hallway to connect them.
After the deletion of empty spaces from the scene, the majority of room objects have their edges touching the edges of the other room objects. To ensure that a path exists from every room to every other room, we have designed a simple technique that connects the entrance room object to every other room object by a single hallway. A hallway is a collection of connected paths.

Recall that one of the constraints on the entrance object is that it is always placed on an edge of the house. Procrustes generates the hallway by considering the entrance object as the initial path in the hallway. This path will later be connected to all the room objects in the scene, which results in a house with interconnected rooms. The below algorithm explains the traversal of the generation hallway algorithm.

Algorithm: generateHallway

Input: sceneTree, pathDimension

Output: SceneTree with hallways
The generate hallway algorithm traverses the room objects in the house in a breadth-first order. Initially, the entrance room is the first room to be processed during generation of hallway. The entrance room object is considered as the first entrance path to the house. For the algorithm described above, the room object under processing is represented by the variable current element. The variable current element is the room
object which is on the verge of having paths connecting all its neighboring rooms. If the current room does not have a path, we generate a path to it and mark it as visited. At each of the paths in the hallway, the current room is connected to all of the rooms to the left, front, right and back side of the entrance room. This process is repeated for each of the neighboring room objects in breadth-first order. This process is then repeated until there are no more unprocessed room objects available in the house. The generate hallway algorithm in this section generates the room on the left edge of the current room. We consider the entrance as part of the hallway.

In Figure 3.10, the numbers in parenthesis beneath the room names give the order in which the rooms are connected to the hallways. Consider this algorithm is applied to the example house scene shown in Figure 3.7(a). Initially, the entrance object is added to the connectedRooms queue of objects. Since the entrance room object is considered as a path in the hallway, it is added to the list of paths.

While the list of connectedRooms is not empty, the room objects surrounding the entrance room objects are added to neighbors list. For the example scene shown in Figure 3.9, the living room, bedroom and study rooms are considered as the neighbors of the entrance room object. Since at least one of the edges of each of the living room, bedroom and study objects is connected to a path in the list paths, no change is made. All the neighbors of the entrance room object are added to the connectedRooms. The entrance object is marked as visited to avoid an infinite loop. In the next iteration of the while loop, the first element, which is the living room, is considered as the current element. The neighbors of the living room, the kitchen and dining room as shown in Figure 3.7(b) are
added to the neighbors queue. As the kitchen object is not connected to a path of the living room object, a path is placed from the current path of living room object to the kitchen room object. Since one of the edges of dining room object is connected to the path between the kitchen and living room object, no path is generated. This process is then repeated for the neighbors of the bedroom object, the study object and so forth until the connectedRooms queue is empty. As a result, a house where all rooms are connected to a hallways is generated, as shown in Figure 3.7(c). The rooms at the left edge of the house object are obtained and sorted based on their positions on the Z axis.

Figure 3.10: Processing the neighbors for generation of hallway

After the generation of the hallway, all the resultant scenes go through a scoring procedure that compares the initial dimensions of each room with its current dimensions. The scene with the objects that have the least difference between the current and the initial dimensions are selected as the resultant scene. After the generation of hallway, the
doors are places by considering the paths to a room.

3.7. Placement of Furniture in Rooms

As observed by L.F. Yu et al. [23], considering various real life features of pieces of furniture and their usage may result in useful constraints to generate a realistic scene. Leveraging this observation, we have chosen to consider the purpose, visibility and accessibility features of furniture objects to help generate realistic scenes.

3.7.1. Common Purpose of Objects

As mentioned in Section 3.2, room objects with a common purpose tend to be placed closer to each other than other room objects. Similarly, furniture objects with a common purpose tend to be placed closer to each other than other furniture objects. In addition to placing such objects close to each other, we also ensure that the common purpose between a pair of object can be conveniently fulfilled.

For furniture objects, orientation is the other major feature besides position that has be considered during the placement of objects. By default, surfaces of every furniture object are oriented as shown in Figure 3.11. For furniture without obvious fronts, some default orientation has been arbitrarily selected. To obtain the orientation in the scene of a pair of objects involved in a relation, we define the surfaces associated with the relations. These relations are defined using examples of real world furniture arrangement. An example of a relation involving surfaces is the arrangement of TV and sofa objects in a room. Another example for furniture arrangement in real world is the placement of a chair objects around a dining table object. The accessible surfaces of the TV and sofa
objects are their front surfaces. Similar, the accessible surfaces of a dining table are the four vertical surfaces of the table. Likewise, the accessible surface of a sofa object is its front surface. The relations between such objects are defined as follows:

1) relation(tv, front, view, sofa, front)
2) relation(diningTable, any, around, diningChair, front)

In the first relation shown above, a visibility relation is defined between the TV object and the sofa object. The visible surfaces of the objects are also mentioned. The relation specifies that the front surface of the TV object should be visible to the front surface of the sofa. In the second relation, a placement constraint is given. The front surface of the diningChair object should be placed along any of the vertical surfaces of the diningTable object, which are left, right, front and back. In the next two sections, the properties of visibility and accessibility relations are considered in more details.
3.7.2. Visibility Relations

A visibility relation holds between objects A and B if object A can be seen from object B and object B can be seen from object A. A visibility relation can be either a sibling visibility relation or a parent visibility relation. A sibling visibility relation is a visibility relation between two objects sharing a common parent object. For example, TV and sofa are defined as the sibling object in the concept hierarchy and a visibility relation between them is a sibling visibility relation.

The sibling visibility relation between object A and object B is processed by calculating a visible zone for object A and placing object B in the visible zone. A visible zone for an object A is calculated by placing a series of bounding boxes with increasing unit dimension on each direction as shown in Figure 3.12.
Whenever a visibility relation is defined, the visible surfaces of the objects have to be defined. Figure 3.11 shows the front, bottom, top and right surface of an object. The surfaces of the furniture objects are predefined in the knowledge base. Let us consider the sibling visibility relation between a TV and a sofa: \( \text{relation}(\text{tv, front, view, sofa, front}) \). The front surface of TV should be visible to the front surface of sofa. Based on the assumption that every model for a furniture object faces forward along the positive Z axis, the models are rotated as needed to achieve their orientations.

The visibility relation can also be defined between parent and child objects. The parent visibility relation is a visibility relation that describes the appearance of object A from the majority of the empty space within the parent object. This relation is implemented in a similar fashion to the sibling visibility relation. A set of possible positions are calculated for an object A, the visible parent zone of object A from each of the possible positions are calculated as shown in Figure 3.13. A sequences of empty
spaces are placed against the visible surfaces of object A. This sequence of empty spaces extend from the visible surface of object A to the opposite wall of the parent object. Every empty space in this sequence has its dimensions increased by one unit on all the three axes. If an empty spaces from this sequence intersect with the visible parent zone of an other furniture object, then the difference is the two visible zones are calculated as shown in Figure 3.13. Such a difference operation is implemented by subtracting the volume of visible parent zone of other object. At last, a single position is obtained by finding the position that has the highest visible parent zone volume. For example, Paintings in a living room are placed such that they are visible from majority of the area in the living room.

![Figure 3.13: Visible parent zone for object A](image)

In a parent visibility relation, a visible surface of a furniture object and the side of the parent object it is visible to are defined with the names of these objects. For instance, `relation(mirror, front, view, bedRoom, left)`. The front surface of the mirror object will be facing the left side of the bedroom wall.
3.7.3. Accessibility Relations

As mentioned in Section 3.1, every furniture object in the scene has to be accessible. This is a default relation defined for every furniture object. In order to implement this relation on furniture objects, we utilize two techniques. The first technique is to ensure that every furniture object in the scene has a minimum accessible surface area for each of its accessible surfaces. For every surface of an object, the surface distance is a specified minimum perpendicular distance from the surface to the nearest other furniture object or wall. Enforcing the constraint that no other furniture object is closer than this distance ensures that the object can be comfortably used. The second technique is to ensure that a path exists between every pair of furniture objects in the scene.

3.8. Similar Scene Generation

As mentioned in Chapter 1, the objective of Procrustes is to generate a user-desired scene from a partial scene description. It initially generates ten candidate scenes and then, based on user input, it attempts to iteratively improve these scenes. The user is required to provide a positive, negative or neutral response to each of the presented scenes. A positive response indicates that the scene appears similar to the desired scene and a negative response indicates that relations in the scenes are not similar to those in the desired scene. The user provides a single response to a whole scene rather than responses to individual relations in a scene. Once the user has responded to all scenes, all relations appearing in any of the scenes are rated. Similar scenes are generated by
preferring the relations occurring in positive scenes and eliminating the relations that occur often in the negative scenes.

Similar scene generation works in two phases, the relation rating phase and the selective relation merging phase. After the user has provided his/her response on a scene, the relations in the scene will be rated by Procrustes, considering the user response for the scene. The *relation rating* phase finds all relations in the scenes presented to the user. These relations are rated using a simple heuristic, considering the user response. In the *selective merging* phase, a set of existing relations from the source scene will be replaced by the relations with high heuristic scores, in order to generate the new scenes.

### 3.8.1. The Relation Rating Phase

The relation rating phase works in two steps. In the first step, the scene tree for each of the scenes presented to the user are traversed in breadth first order and every distinct explicit relation between a pair of objects is listed and rated. A pair of relations is said to be equal if the relation name and associated objects are the same. Distinct relations are obtained by ensuring that no two relations in the list are equal to each other. In the second step, these ratings are re-evaluated by identifying implicit occurrences of the same relations and updating their ratings accordingly.

The first step accepts a list of scene trees and produces a list of relation scores. A *relation score* consists of a relation, an integer rating and a list of the scene numbers corresponding to the scenes containing this relation. Figure 3.14 shows an example relation score. In the example relation score shown in Figure 3.14, the first field represent
a relation relation(a, left, b), the second field represents the rating associated to the given relation and the third field represents the scene numbers of the scenes that have the given relation in it. The rating of a relation partly depends upon the number of times it appears in the candidate scenes. For every occurrence of a relation in a positive scene, the rating is increased by one unit. Every occurrence of a relation in a negative scene, the rating is decreased by a unit. If the relation occurs in a scene that has not received any response from the user, no change is made to the rating. This results in a single rating for each distinct relation.

<table>
<thead>
<tr>
<th>Relation</th>
<th>Rating</th>
<th>Scene numbers</th>
</tr>
</thead>
<tbody>
<tr>
<td>relation(a, left, b)</td>
<td>3</td>
<td>1, 3, 5, 8, 12</td>
</tr>
</tbody>
</table>

**Figure 3.14**: The structure of a relation's score

While adding a relation to the list of encountered relation scores, Procrustes considers two cases:

- The current relation is not present in the list of encountered relation scores. In such a case, the following steps are:
  - Add the relation to the encountered relation list.
  - Add the rating.
  - Add the scene number.

- The current relation is already present in the list of encountered relation. In this case, the rating and the list of scene numbers are updated. The rating is incremented or decremented by one. The scene number of the current scene
is added to the list of scene number associated to the relation score.

After the relation score is complete, it is passed on to the next step. In the second step of the relation rating phase, the ratings are reevaluated by looking for the implicit existence of the encountered relations in the scenes. While re-evaluating a relation score, scenes that do not have their scene numbers listed in the current relation score are obtained. For every obtained scene, a simple check is performed to ensure that the relation associated to the current relation score does not hold true between the same pair of objects. During the process of checking for the existence of a relation between a pair of objects, the positions of these two objects are retrieved and a simple check is performed to check the implicit existence of a current relation between this pair of objects. If the relation exists between the selected pair of objects, the rating for the current relation is updated accordingly. This will ensure the consideration of all the relations, although some of those were not explicitly defined in the scene tree. This process is repeated for every scene presented to the user in the previous scene generation run, considering every encountered relation. At the end of relation rating phase, we will have a list of relations scores as an output.

3.8.2. Selective Relation Merging Phase

In the previous relation rating phase, the user response is used to rate the relations. These ratings are used in the selective relation merging phase to generated refined scenes. The selective relation merging phase accepts scene trees and a list of relation scores as input. As output it produces a new set of scene trees that are similar to the scene trees that
had received positive responses from the user.

The average relation score is determined by utilizing the list of relation scores. Firstly, the total rating score is calculated by summing the relation scores of all scenes. Secondly, to obtain the average relation score, the total rating is divided by the number of relation scores.

**Algorithm:** Selective Relation Merging

**Input:** sceneTrees, relationScores

**Output:** similarSceneTrees

begin
    traversedObjects = [ ]
    relationRating = null
    averageRelationScore = calculateAverageScore(relationScores)
    sortedHigherRatedRelations=getRelationWithHigerRating(
        relationScores,averageRelationScore)
    equivalentHigherRatedRelations = []
    For each sceneTree in sceneTrees
        traversedObjects.Add(sceneTree)
        While traversedObjects is not empty
            For each curReln in traversedObjects[0].Relations
                relationRating = getRelationRating(
                    curReln,relationScores)
If(relationRating < averageRelationScore)
    equivalentHigherRatedRelations =
        getEquivalentHigherRatedRelations(
            sortedHigherRatedRelations,
            averageRelationScore)
    If(equivalentHigherRatedRelations.Count > 0)
        curReln = getReplaceableRelation(
            equivalentHigherRatedRelations, curReln)
    end For
traversedObjects.Add( traversedObjects[0].ChildElements)
end While
end For
end

The first step in the selective relation merging phase is to calculate the averageRelationScore. The second step is to obtain a list of all the relations scores with ratings higher than the averageRelationScore. These Relation Scores are sorted based on their ratings and are saved in sortedHigherRatedRelations. In the third step, the sceneTrees are traversed one after the other. During the traversal, an attempt is made to replace every relation with rating less than averageRelationScore.

A relationRating is obtained for every relation associated with an object. In the
next step, a list of relation scores with a relation that could possibly replace the current relation is obtained and stored in equivalentHigherRatedRelations. In the getReplaceableRelation function, a simple constraint check is performed to find whether the current relation can be replaced by the relation associated with the relation scores in equivalentHigherRatedRelations. If a replaceable relation is found, the equivalentHigherRatedRelations function return it and the current relation is replaced by it. This process is repeated for every object in the scene tree. Thus resulting in scene trees with higher rated relations.

For example, let us consider the input scene from Figure 3.15 (a). Assuming that the relation, relation(bathRoom, behind, masterbedRoom) has lower rating than the average relation score, it will be replace by a higher rated relation. If the higher rated relation is the relation(bathRoom, infront, masterbedRoom), an intermediate scene is generated by repositioning the related objects to ensure that the higher rated relation can be processed as shown in Figure 3.15. Finally, the higher rated relation is processed by considering the empty spaces and the position of the related object, as shown in Figure 3.15 (c).
After this phase the similar scenes that were generated are presented to the user to obtain further relation ratings responses. This process is repeated until a fixed number is reached or the user is satisfied with the presented scenes.

Figure 3.15: Illustration of selective relation merging
Chapter 4

Descriptive Example

This chapter presents evidence for the effectiveness of the method introduced in this thesis and implemented in the Procrustes software. We demonstrate the features discussed in this thesis by tracing through the scene generation process. We show how the knowledge base and input description can be used to create a variety of interesting scenes. We begin by presenting a partial scene description in Section 4.1. From this description, a scene is generated, as described in Sections 4.2 to 4.5, which discuss initial scene generation, space reduction, hallway generation and furniture placement, respectively. After the user provides his feedback on the presented scene, then similar scenes are generated.

In this chapter, we also present screenshots of scenes generated during the above mentioned phases. The screenshots in Figures 4.2 to 4.9 show scenes created by Procrustes and rendered by the OGRE graphics rendering engine. These screenshots are taken from the view point of a camera positioned in the 3D world that contains the floor plans. Since the camera is positioned above and behind the house object, the screenshots appear to be bigger at the bottom than the top, with a black background. Most of the black background was clipped from the images using an image editor.
4.1. Partial Scene Description

The input scene description file shown in Figure 4.1 contains a partial scene description, provided by a user. This description specifies constraints on the positions of the kitchen, master bedroom, bedroom and the bathroom objects in a house. The generated scene should have five additional rooms, which are defined in the knowledge base as the child objects of house object. Relation extraction is performed from the knowledge base to obtain relations that help in placing these additional room object. The relevant parts of the knowledge base are shown in Appendix A. Whenever an object without any spatial relation associated to an object is encountered, a simple Prolog function shown below is called.

\[
\text{relation(Any):-}
\]
\[
\text{relation(left);} \\
\text{relation(right);} \\
\text{relation(front);} \\
\text{relation(behind).}
\]

This function returns four possible relations. These extracted relations will be either between a pair of child objects or a pair of objects with a common purpose. These relations are checked for consistency. If they are consistent, possible relations are extracted. If multiple scenes are possible, extracted possible relation are assigned to each of them. If the number of generated scenes is less than the number of extracted possible relations, new scenes are generated for every possible relation. For our example scene, the placement of dining room, living room, store room, study room and the entrance room objects result in multiple scenes due to the extracted relations.
4.2. Initial Scene Generation

Figure 4.2 shows one possible initial scene for the input description. This scene has positions for all the rooms defined in the knowledge base. This scene satisfies the input scene description for the kitchen, master bedroom, bedroom and bathroom objects. The scene also shows one possible arrangement of the remaining rooms.

**Table 4.1:** Characters representing the room objects

<table>
<thead>
<tr>
<th>Characters</th>
<th>Room objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>B</td>
<td>bathroom</td>
</tr>
<tr>
<td>D</td>
<td>dinning room</td>
</tr>
<tr>
<td>E</td>
<td>bedroom</td>
</tr>
<tr>
<td>K</td>
<td>kitchen</td>
</tr>
<tr>
<td>L</td>
<td>living room</td>
</tr>
<tr>
<td>M</td>
<td>master bedroom</td>
</tr>
<tr>
<td>N</td>
<td>entrance</td>
</tr>
<tr>
<td>S</td>
<td>study room</td>
</tr>
<tr>
<td>T</td>
<td>store room</td>
</tr>
</tbody>
</table>

Table 4.1 shows the characters that map to the room objects in the generated scenes. Figure 4.2 shows the an intermediate scene that uses these characters from Table 4.1 to represent the room objects.
As previously mentioned, the purpose of an object is considered before it is placed in a scene. This purpose can be described in the form of a relationship to another object. In the example scene shown in Figure 4.2, the kitchen object is placed at the left front of the house object. left front is a specific relation, which only has one possible interpretation. The dining room object is always placed around the kitchen object as they share a common purpose. The common purpose of kitchen and dining room is dine. It is represented as \text{purpose(kitchen, dine)} and \text{purpose(dining, dine)} in the knowledge base. This common purpose results in a generic \textit{near} relation between the kitchen and the dining room. A generic relation can be replaced by one of several specific relations. Thus, multiple valid scenes may be generated from a single scene description.
4.3. Reduction of Empty Spaces

After the placement of all the room objects, the empty spaces are scattered around the room, as shown in Figure 4.2. As previously discussed, we reduce these empty spaces using a two stage process. In the first stage any empty vertical or horizontal strips extending across the house are deleted. In the second stage, objects are expanded. Figure 4.3(a) shows the scene after the deletion of all empty strips from the example scene shown in Figure 4.2. A subset of the available room objects are selected to be expanded to further reduce the empty spaces in the scene. The results shown in Figure 4.3(b).

![Image](image_url)

(a) Empty strips deleted  
(b) Room objects extended

**Figure 4.3:** Scenes with reduced empty spaces(original in color)

As a result of this phase, we have removed the empty spaces that could have been removed. Hence reducing the area of wasted space in the house.
4.4. Generation of Hallway

One of the requirements for a house object is that there should be a path connecting the rooms with each other. This path is commonly referred to as a hallway. Procrustes ensures that there is a hallway that extends from the entrance room object to all the other rooms in the house, as shown in Figure 4.4. A hallway has been generated for the example scene shown in Figure 4.3(b). Every room has a path connecting it to a common hallway. The white patches on the walls show the doors that allow access to these rooms.

![Figure 4.4: Scene with a hallway (original in color)](image)

4.5. Placement of Furniture

After the layout for the rooms is complete, the furniture objects are placed. During the placement of furniture objects, features such as accessibility and visibility are considered. As mentioned in Section 3.7, accessible and visible surfaces of furniture objects are predefined by rules and facts in the knowledge base. For instance, the front
surface of a stove object should be accessible, as shown in the Figure 4.5(a). The predefined fact stored in the knowledge base to describe the accessible surfaces of the stove is accessibleSurfaces(stove, 1, front). The number 1 in the fact represents the number of accessible surfaces, while the keyword front identifies the surface of the stove that has to be accessible. For the example scene shown in Figure 4.4, although the stove object is placed at the front corner of the kitchen object, its back surface is accessible because it is not against a wall or other object, as shown in Figure 4.5(a). If there was an object a that should have its front and back side accessible, then the facts will be:

- accessibleSurfaces(a, 2, front).
- accessibleSurfaces(a, 2, back).

Similarly, the fact accessibleSurfaces(bed, 3, Surface) says that any three vertical surfaces of the bed object should be accessible. The variable Surface in the rule will be replaced by left, right, front or back. This fact holds true in the scenes shown in Figure 4.5(b) since the left, back and front surfaces are accessible. Procrustes ensure that such features are implemented irrespective of which corner the furniture objects is placed in.

The other feature to be considered is surface distance. The furniture objects should have a predefined area associated with their accessible surface. As shown in Figure 4.5(a), both the stove object and the fridge object have surface distances associated with their accessible surfaces that allows convenient usage of these objects.
Similarly, the cupboard objects (shown in Figure 4.5(b) inside the bedroom object) have a surface distance associated with them that ensures convenient usage of these objects.

(a) Stove object with appropriate rotation  
(b) Cupboard with enough area in front of it and the bed placed against the wall

**Figure 4.5:** Placement of furniture objects (original in color)

As previously discussed in Section 3.6, Procrustes also considers the visibility relations between pairs of objects. The Figure 4.6 also demonstrates the visibility relation between a pair of sibling objects, a TV and a sofa. The TV object was initially placed on the back edge of the living room. The sofa object was later placed by ensuring that the TV object is visible from the sofa object. Visibility relations ensure that objects that are
visibly related should be conveniently visible from each other, without any blocking obstacles. While considering a visibility relation between a parent-child pair of objects, such as a living room and the painting object, Procrustes calculates several possible positions for a painting and finally selects the one with the maximum visible zone, as shown in Figure 4.6. While calculating the visible zone for a painting object, the visible that are visible from the far surfaces of the sofa object and TV object (called the obstructed zone) are removed from the visible zone of the painting. At the end the position with maximum visible zone is selected as the resultant position of painting object. Figure 4.7 shows the obstructed zone of the painting from the left corner of the house.

Figure 4.6: Visible zones for a TV and a painting (original in color)
**Figure 4.7:** Visible zones for painting intersecting the obstructed zones

(original in color)

**Figure 4.8:** Complete layout with furniture
In Figure 4.8, a complete layout with furniture is shown. All constraints specified in the partial scene descriptions given in Figure 4.1 are satisfied in the scene. The wasted empty space between rooms has been eliminated. A single branching hallway connects the entrance hall to every other room. Furniture has been placed on the rooms obeying the purpose, accessibility and visibility constraints. Thus, the Procrustes in producing a valid floor plan.

4.6. Similar Scene Generation

One of the goals for the similar scene generation phase discussed in Section 4.1 is to replace relations with lower scores by relations with higher scores. Let us consider the two scenes shown in Figure 4.8. The user provides a negative response to every scene where the relation between the living room object and the house object is left corner. For example, the scene in Figure 4.8(a) receives a negative response from the user. These negative responses lower the score for the relation and thus reduces the likelihood that it will be present in future scene generation iterations. After two complete iterations, Procrustes has learned that the left corner relation is an undesirable relation for the living room object and none of the generated scenes have this relation. For example, Figure 4.8(b) shows a typical scene from the third iteration, where the living room object is placed at the back right corner of the house object. The scores of the other relations are also affected by the user's negative responses to the leftCorner relation. As a consequence, the positions of other room objects are also changed over the iterations. For example, the position of the study room object has been changes over the two scene generation iterations. In the scene shown in Figure 4.8(a), the study room object in front
corner of the house object but in the scene shown in Figure 4.8(b), it is relocated to the right corner.

Figure 4.9: Effects of user responses during scene generation (original in color)

As mentioned in this Section 4.1 and 4.2, Procrustes has accepted a partial description and generated initial scenes, one of which is shown in Figure 4.2. In the next two stages, an attempt is made to remove the wasted spaces from these initial scenes. This attempt involves deleting the strips of spaces and expanding the objects. Then, a hallway, connecting all the rooms is generated. As shown in Section 4.5, furniture are placed in the rooms to complete an iterations. Then these scenes are presented to the user, the user then provides his response, evaluating the presented scenes. This response leads
to many more iterations, until a fixed number is reached or a user-desired scenes is presented.
Chapter 5

Conclusions and Future Work

In this chapter, we present our conclusions and suggest possible topics for future research related to this thesis. Section 5.1 provides the conclusions and Section 6.2 provides several suggestions for future research.

5.1. Conclusions

We developed the Procrustes system to automate the process of generating floor plans and furniture placement for a house. Our system takes a partial scene description as an input and outputs a 3D plan for a house satisfying the input description. Along with the user description, several other features such as possible placements for the object not described in the input and generation of a hallway, are considered. To place the objects that are not described, one of the predefined spatial relations from the knowledge base is considered. After obtaining a position for every required object, several possible scenes are presented to the user.

The user is then requested to provide either a positive or negative response for the presented scenes. Based on the responses from the user, more scenes are generated with
spatial relations similar to the positive scenes but dissimilar to the negative ones. This process is iterated until a fixed number of iterations is reached or until the user is satisfied.

The main application of this system is to generate house plans for games and simulations in few minutes. The results of the experiments show that Procrustes works as expected.

The original contribution of this thesis are:

(a) Procrustes accepts partial textual scene descriptions while previous systems accepted either no textual description or complete ones;

(b) Procrustes can generate similar scenes based on simple positive and negative ratings of several scenes, while a previous system required numeric ratings; and

(c) Procrustes provides both floor planning and furniture arrangement, while previous systems provided one or the other.

5.1 Future Work

The current system can be improved in several ways. The first improvement would be to generate rooms using multiple bounding boxes. Currently, the room objects are represented using a single bounding box, which results in a scene with room objects that are represented by a axis-aligned rectangular parallelepipeds. To overcome this limitation, a room object could be represented by multiple bounding boxes.

The second improvement would be the increase to number of levels in the concept
The current concept tree has three levels; but it could be increased by including
more objects above or inside the furniture objects. This change would increase the
realism of the scenes.

The third improvement would be to add more information to the knowledge base.
Our current system has considered the purpose, visibility and accessibility of objects. The
knowledge base can be improved to consider more information such as multiple purposes
for single objects. The knowledge base could also be improved to include more objects
and relations to constrain the placement of the freshly introduced objects. For example,
new objects such as a bench in the entrance or a table in the kitchen could be added. New
relations such as accessible surface and purpose of bench are also introduced with the
object.

The fourth improvement would be regarding the similar scene generation phase.
As mentioned in Section 4.1, the current version of similar scene generation is applicable
only for generating similar scenes by changing the spatial relations between the room
objects in a scene. The idea of similar scene generation can be extended to multiple levels
of hierarchy. For example, furniture could be placed in similar but possibly better
locations by rating the relations between furniture objects. After the relations between
furniture objects have been rated, relations with lower ratings could be replaced by
relations with higher ratings.

The fifth improvement would be to allow the user to provide numerical distances
between objects in addition to the spatial relations between them. For example, the distance
between the TV and couch could be specified as: **TV is 5 meters away from the couch.** We could also extend the list of allowed relations by implementing ones such as **near** and **far**. These descriptions could help reduce the size of the search space examined by Procrustes.

The sixth improvement would be to create multiple scenes by performing object expansion in several ways instead of only one. For example, with the approach described in this Section 3.5, if space is available to expand three room objects, all three are expanded as evenly as possible. Additional possibilities could be created by dividing the space unevenly, perhaps by expanding one room object as far as possible and the others by as little as possible. Similarly, expanding the width of a room object could result in scenes that appear different from those created by expanding the length. Instead of producing a single scene as a result of object expansion, by discarding the remaining ones, several scenes could be produced and possibly presented to the user. Overall this approach would allow the user to explore more options.
References


Appendix A: Procrustes Knowledge Base

This appendix gives the Prolog knowledge base used by the Procrustes software.

// Location of accessible surfaces of the sofa object:
// the number of accessible vertical surfaces is 3.
accessibleSurfaces(sofa, 3, front).
accessibleSurfaces(sofa, 3, left).
accessibleSurfaces(sofa, 3, right).

// Size of the empty space beside an accessible surface:
// reserves an empty space up to 1 meter wide against
// the left surface of the sofa object. Here the 1
// represents the distance that has to be empty and left
// identifies the surface.
distanceSurfaces(sofa, left, 1).

// Visibility relation: any surface of the bedroom can be
// viewed from the front surface of the mirror. Here X
// represents any surface.
relation(mirror, front, view, bedRoom, X).

// Hierarchical relations
relation(kitchen, in, house).
relation(diningRoom, in, house).
relation(studyRoom, in, house).
relation(entrance, in, house).
relation(storeRoom, in, house).
relation(bedRoom, in, house).
relation(livingRoom, in, house).
relation(sofa, in, livingRoom).
relation(masterBedroom, in, house).
relation(bathRoom, in, house).
relation(closet, in, masterBedroom).

// Predefined relations
relation(left).
relation(behind).
relation(right).
relation(front).
relation(back).
relation(rightFront).
relation(rightBack).
relation(rightBehind).
relation(frontRight).
relation(backRight).
relation(behindRight).
relation(leftFront).
relation(leftBack).
relation(leftBehind).
relation(frontLeft).
relation(backLeft).
relation(behindLeft).

// Predefined objects
object(fridge).
object(toilet).
object(tv).
object(chair).
object(matress).
object(house).
object(bedRoom).
object(masterBedroom).
object(kitchen).
object(livingRoom).
object(diningRoom).
object(bathRoom).
object(studyRoom).
object(sofa).
object(tv).
object(tvStand).
object(chair).
object(storeRoom).
object(entrance).
object(fridge).
object(stove).
object(microwave).
object(sink).
object(kitchenTable).
object(diningTable).
object(table).
object(mirror).
object(bed).
object(bedSidetable1).
object(bedSidetable2).
object(window).
Appendix B: Parsing of the Input Description

This appendix explains how Procrustes parses an input description. The user provides the input description in structural natural language using the exact names for the objects and relations listed in the knowledge base. An input description from the user should be of form:

otherObject1, otherObject2,..., otherObjectN RelationName curObject

Each word from the description is scanned and categorised as an object name, a relation name or an unknown word. To categorise the words in the input description, two simple queries are sent to the knowledge base:

• object(currentWord).
• relation(currentWord).

Every relation has three strings, which give the names of:

• current object
• relation
• other object

The algorithm below parses a description from the user.

**Algorithm:** Parsing an input description

**Input:** inputDescription

**Output:** List of relations

```
begin
    relations[] = null;
    otherObjsCollected = false;
```
relationFound = false;
curObjectFound = false;
Foreach curWord in inputDescription
    If (curWord is not an object name and
        curWord is not a relation name)
        ignore unknown word in input
    Else if (curWord is an object name and
        !relationFound)
        curRelation = null;
curRelation.OtherObj = curWord;
relations.Add(curRelation);
otherObjsCollected = true;
Else if (curWord is a relation name and
        otherObjsCollected and !relationFound)
    Foreach relation in relations
        relation.Relation = curWord;
    end Foreach
    relationFound = true;
Else if (curWord is an object name and
        relationFound and !curObjectFound)
    Foreach relation in relations
        relation.CurObj = curWord;
    end Foreach
    curObjectFound = true;
Else
    ignore erroneous input
end If
end Foreach
If (!otherObjsCollected or !relationFound or !curObjectFound)
    report error
end if
end

An input description is processed word by word. Considering that the format of the input description allows for multiple names of other objects, every object name until a relation name is encountered is treated the same: a new instance of a relation is created and added to the list of `relations` and its `otherObj` field is set to the current input word. Upon encountering a relation name, every relation in the `relations` list is updated to add the name of the relation. Similarly, upon encountering an object name after the relation name, every relation in the `relations` list is updated to save the name of the current object. At the end of processing all the words in the given input description, if any of the relation name, other object name or current object name was not encountered, an error is reported to the user.
Appendix C: Overview of the Procrustes System

As mentioned in Section 2.1, Procrustes has three phases, the scene description phase, the scene generation phase and the similar scenes generation phase. In the implementation, each of these phases corresponds to an individual module, as shown in Figure A.2.

Figure C.2: Modules in Procrustes

Table A.1 shows the name of the modules and files, the language of implementation, and the number of lines of code in the modules of Procrustes.
<table>
<thead>
<tr>
<th>Module name</th>
<th>File name</th>
<th>Purpose of File</th>
<th>Count of lines</th>
<th>Language</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scene description</td>
<td>Conflicts.pl</td>
<td>Performs simple conflict detection in the input description</td>
<td>26</td>
<td>C# and Prolog</td>
</tr>
<tr>
<td></td>
<td>GeneralMethods.cs</td>
<td>General methods required for whole project, such as float comparison.</td>
<td>208</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Parse.cs</td>
<td>Parses the input scene description</td>
<td>620</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Relations.cs</td>
<td>Represents a relation with two object and one relation names</td>
<td>56</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Trees.cs</td>
<td>Generates a description tree</td>
<td>358</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td></td>
<td>1268</td>
<td></td>
</tr>
<tr>
<td>Knowledge base</td>
<td>Objects.pl</td>
<td>Saves the list of allowed objects</td>
<td>19</td>
<td>Prolog</td>
</tr>
<tr>
<td></td>
<td>Relations.pl</td>
<td>Saves the list of allowed relations</td>
<td>26</td>
<td></td>
</tr>
<tr>
<td></td>
<td>SpatialKnowledge.pl</td>
<td>Saves the information regarding the purpose, accessible surfaces and distances from the surfaces of an object</td>
<td>251</td>
<td></td>
</tr>
<tr>
<td></td>
<td>Total</td>
<td></td>
<td>296</td>
<td></td>
</tr>
</tbody>
</table>
### Table C.1 (cont’d): Procrustes Modules and Files

<table>
<thead>
<tr>
<th>Scene generation</th>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Accessibility.cs</td>
<td>Access functions related to the accessibility relation</td>
<td>167 C#</td>
</tr>
<tr>
<td>CameraSetup.cs</td>
<td>Handles values related to the 3D camera and views</td>
<td>282</td>
</tr>
<tr>
<td>DisplayObjects.cs</td>
<td>Uses OGRE library to display scenes and receive user response</td>
<td>643</td>
</tr>
<tr>
<td>EmptyBB.cs</td>
<td>Handles empty spaces in scenes</td>
<td>1368</td>
</tr>
<tr>
<td>FurnitureHandler.cs</td>
<td>Queries the knowledge base for the relations relevant to furniture placement</td>
<td>789</td>
</tr>
<tr>
<td>GenerateHallway.cs</td>
<td>Handles generation of hallway</td>
<td>756</td>
</tr>
<tr>
<td>GeneralMethods.cs</td>
<td>General methods such comparison of bounding boxes</td>
<td>289</td>
</tr>
<tr>
<td>SceneObject.cs</td>
<td>Saves the values associated with an object in a scene</td>
<td>128</td>
</tr>
<tr>
<td>SpaceReduction.cs</td>
<td>Handles space deletion and object expansion</td>
<td>653</td>
</tr>
<tr>
<td>SpecificRelation-Handler.cs</td>
<td>Obtains a position for an object, given the name of the relation</td>
<td>710</td>
</tr>
<tr>
<td>TreeTraversal.cs</td>
<td>Tree data structure and relevant functionality to generate a scene tree</td>
<td>210</td>
</tr>
<tr>
<td>VisibilityHandler.cs</td>
<td>Functions related to the accessibility relation</td>
<td>488</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>6483 (20% unused)</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Similar scenes generation</th>
<th>File Name</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SelectiveMerger.cs</td>
<td>Replaces the relations with ratings below threshold rating value</td>
<td>216 C#</td>
</tr>
<tr>
<td>RelationRate-Handler.cs</td>
<td>List all the relations from all scenes and rates each relation</td>
<td>120</td>
</tr>
<tr>
<td><strong>Total</strong></td>
<td></td>
<td>336</td>
</tr>
</tbody>
</table>