A Language Visualization System

by

Emre Ünal

A Thesis Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computer Science and Engineering

Koç University

March, 2014

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Emre Ünal

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Assoc. Prof. Deniz Yuret

_____

Assoc. Prof. Yücel Yemez

_____

Prof. Aylin Küntay

Date: _____

To my beloved parents and Ceren

ABSTRACT

In this thesis, a novel language visualization system is presented that converts natural language text into 3D scenes. The system is capable of understanding some concrete nouns, visualizable adjectives and spatial prepositions from full natural language sentences and generating 3D static scenes using these sentences. It is a rule based system that uses natural language processing tools, 3D model galleries and language resources during the process. Several techniques are shown that deals with the generality and ambiguity of the language in order to visualize the natural language text. A question answering module is built as well to answer certain types of spatial inference questions after the scene generation process is completed. The system demonstrates a new way of solving spatial inference problems by not only using the language itself but with the extra information provided by the visualization process.

ÖZETÇE

Bu çalışmada doğal dilin 3 boyutlu sahnelere dönüştürülmesini sağlayan yeni bir sistem tasarımı sunulmaktadır. Dilin görselleştirilmesini sağlayan bu sistem, soyut olmayan isimleri, bu isimleri niteleyen görünür etkisi olan sıfatları ve konum belirten sözcükleri doğal dil cümlelerinden anlayıp dinamik olmayan 3 boyutlu sahnelere çevirmekte ve bunları doğal dil işleme araçlarını, 3 boyutlu model galerilerini ve dil ile ilgili bilgi kaynaklarını kullanarak, kurallara dayalı olarak yapmaktadır. Bu sistemin anlatımında aynı zamanda dilin genelliği ve belirsizliğini gidererek onu 3 boyutlu olarak görselleştirebilecek teknikler sunulmaktadır. Bu sahne yaratma işlemi sonrasında bilgisayarın sahne ile ilgili bazı çıkarım sorularını cevaplayabileceği bir arayüz de tanıtılmaktadır. Sistemin, sadece dil işleme teknikleri kullanılarak zor cevaplanabilecek bazı çıkarım sorularına, gerçek dünyayı canlandırarak sadece dilden elde edebileceği bilgilerin yanında 3 boyutlu dünya ile ilgili fazladan edindiği bilgileri kullanarak cevap vermesini sağlayan yöntemler anlatılmaktadır.

ACKNOWLEDGMENTS

First of all, I would like to thank Assoc. Prof. Deniz Yuret who has given me a chance to work with him and accepted to be my advisor. It has been a pleasure for me to work with him for the last few years and I have learned a lot from him about computer science and life. He is one of the greatest people one can meet in life. I thank him again for always being understanding, supportive and encouraging.

I thank Assoc. Prof. Yücel Yemez and Prof. Aylin Küntay as well for participating in my thesis commitee and allocating their valuable time. Their advice and comments are important to me. I also thank Prof. Hakan Ürey who was my undergraduate advisor and who had encouraged me to attend graduate school.

I thank to my closest friends and colleagues Mehmet Ali Yatbaz and Enis Sert for all those great times we had. Despite the distance between us, I still feel as if they are still around and their existence makes me happy. I thank to my friend Yusuf Sahillioğlu as well for all the laughs. I really like his extraordinary personality and unpredictable attitude. I also thank other great friends Tolga Bağcı, Aydın Han, Hüsnü Şensoy and Volkan Cirik who made graduate school much better place for me. I thank to all new and past members of the Artificial Intelligence Laboratory. I specially thank Ozan Arkan Can for working with me on some parts of this work. I also thank Can Saydam who is my friend, colleague and co-founder at Manolin.

I thank Ceren Demirci as well with whom I share life for years. She always makes me feel as the luckiest person on earth and without her understanding, patience and

love this work would not be possible.

Last but not least, I thank to my parents Ekrem Ünal and Nurcan Ünal for their everlasting support, patience and love. I hope that they are proud of me as much as I am proud of them.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Language visualization is a domain that combines natural language processing with computer graphics. In this thesis, we present a novel language visualization system that can generate static 3D scenes from full natural language sentences by understanding some concrete nouns, visualizable adjectives and spatial prepositions. In addition to the text-to-scene conversion engine, our system includes a question answering module which can answer questions about spatial relations of the objects in the scene. In this work, it has been shown that this type of a visualization system can aid solving spatial inference problems since such a system is free of the limitations of just using shallow semantics to answer these types of questions.

One of the primary aims of this research is to develop and discover methods and techniques to visualize natural language by constructing 3D scenes. These methods can have many practical uses. It has been shown in previous studies that these methods are used in applications like car accident simulations [3], storytelling [14] and scene placement tasks [2]. Some other areas such as delivery of news, storyboarding process of games and movies, illustration task of books and magazines can also benefit from an automatized language visualization system.

One of the difficulties in natural language processing is dealing with the uncertainty and ambiguity of the language. Another motivation of this research is to

investigate ways to improve natural language understanding by supporting an understanding system with extra information about the physical world. Our system uses some predefined defaults and estimation when necessary in order to overcome the ambiguities and generality in the language. Our generated 3D scenes serve the purpose of improving understanding by adding geometric properties of 3D models and their spatial relations as extra information. This makes it possible to build a question answering module that can answer spatial relation questions even if the existence of the particular spatial relation is not directly mentioned in the text. This is a scalable system since the scene placement and manipulation procedures are independent of the words themselves and any particular 3D model. Every 3D model is treated equally by the procedures and every computed property used in the scene generation process is dependent on the model itself.

Design of a language visualization system is complex since it has many components like language analyzers, lexical resources, graphics tools and 3D model galleries. In this work, we suggest a language visualization system architecture with a question answering engine where all these components work together in harmony. The details of the system and related studies on the subject are investigated in full detail in their related chapters.

The structure of the thesis is as follows:

Chapter 2 summarizes previous works on language visualization. There are many studies in this domain where natural language is combined with other modalities. The presented works are important historically or as an inspiration for our system. However, language visualization literature is not limited with these studies. A more interested researcher should follow other studies from citations

of our references.

Chapter 3 describes third party components that our system depends on. These components include a 3D environment, lexical resources, 3D model galleries and language analysis tools. A reader should be familiar with these pieces since our system uses them heavily. Therefore an entire chapter is dedicated to the details of these components.

Chapter 4 describes the implementation details of our language visualization system. This chapter solidifies the connection between the separate components that the previous chapter introduces and describes how our work is built on top of them. This chapter shows how language analysis works in our system and how sentences are converted to static scenes. This chapter also covers methods to overcome some problems in natural language processing. The question answering module is covered in this chapter as well.

Chapter 5 concludes the thesis with the summary of contributions and provides the reader with future directions for language visualization. The future work section is organized from short term to long term goals.

Chapter 2

RELATED WORK

In this chapter, we investigate previous studies on language visualization. These multimodal systems include a graphics and a language processing component. The studies in this chapter include rule based text-to-scene and text-to-animation generation systems, question answering systems with virtual agents in virtual environments and learning scene-to-text systems that cover different parts of the language visualization domain.

2.1 SHRDLU

SHRDLU [1] is one of the early systems which uses natural language to interact with a robot living inside a virtual world. This virtual world includes blocks with different shapes, sizes and colors. The user manipulates the robot using natural language and interacts with the virtual world. Blocks can be moved and placed on each other using natural language. SHRDLU interface is shown in Figure 2.1.

SHRDLU has a memory of actions it has taken and these can be queried by the user as well. The user can also query the current state of the world. Normally user picks the objects by describing them with adjectives. However it is also possible to name the objects in the scene. The named object can be referred by its name at a later time. SHRDLU also has a physics engine and it can determine whether two objects can stay on top of each other or not. These properties of SHRDLU enable

it to handle many different cases and scenarios. A sample dialog with the SHRDLU system is shown below.

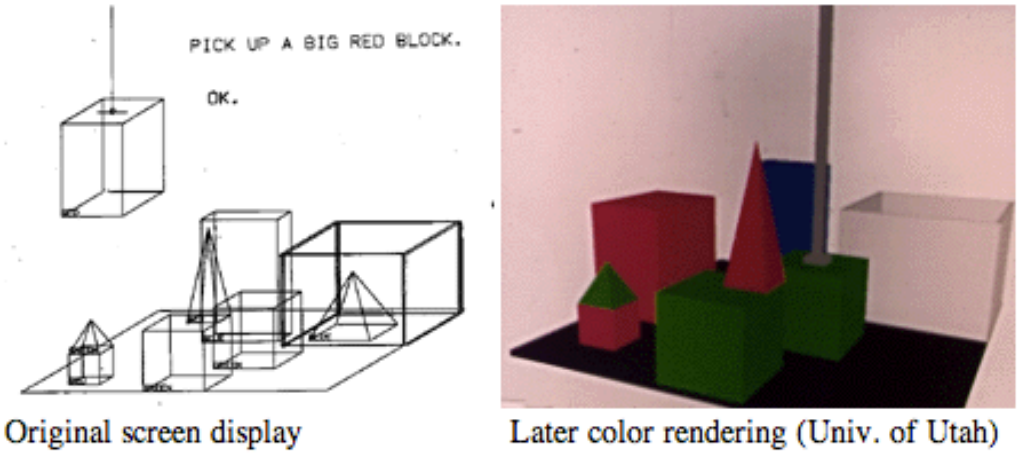

Figure 2.1: A display of SHRDLU blocks world.

```
Person: PICK UP A BIG RED BLOCK.
Computer: OK. (does it)
Person: GRASP THE PYRAMID.
Computer: I DON'T UNDERSTAND WHICH PYRAMID YOU MEAN.
Person: FIND A BLOCK WHICH IS TALLER THAN THE ONE YOU ARE
HOLDING AND PUT IT INTO THE BOX.
Computer: BY "IT", I ASSUME YOU MEAN THE BLOCK WHICH IS
TALLER THAN THE ONE I AM HOLDING.
Computer: OK. (does it)
Person: WHAT DOES THE BOX CONTAIN?
Computer: THE BLUE PYRAMID AND THE BLUE BLOCK.
Person: WHAT IS THE PYRAMID SUPPORTED BY?
Computer: THE BOX.
```

```
Person: HOW MANY BLOCKS ARE NOT IN THE BOX?
Computer: FOUR OF THEM.
Person: IS AT LEAST ONE OF THEM NARROWER THAN THE ONE
WHICH I TOLD YOU TO PICK UP?
Computer: YES, THE RED CUBE.
Person: IS IT SUPPORTED?
Computer: YES, BY THE TABLE.
```

SHRDLU is limited in the sense that all the actions are available and applied to the blocks that already exist in the virtual world. It uses a limited set of vocabulary and objects.

2.2 Put System

The Put system [2] is an object placement system that uses simplified natural language commands. The aim of this system is to make scene construction task easier by using a strict subset of English and direct manipulation. 3D scene construction can be difficult in a 2D projected window. Mouse and keyboard input may not be always sufficient and suitable for such a task. Different views of the 3D scene is needed as well to check if the placement of an object is correct.

Put includes a modeling environment and an application programming interface. It provides a multi window environment for viewing, direct scene manipulations and linguistic input. The input of the system is a file that describes the scene and the output can be a new Put file or an image. The input file includes references to the entities in the scene with names and it provides a link to the geometry file for each entity. The scene is rendered by satisfying spatial constraints independent of the geometry that is provided for each object. After the input file is rendered by

the system, the user can work more on the scene with linguistic input or direct manipulation.

The Put system uses a simple syntax for its natural language commands. The system commands are in the form *Verb Trajector (TR) [spatial relation Landmark (LM)]*. For instance, a user might say:

```
put "book" on "table"
```

Here "book" is the trajector and "table" is the landmark object that "book" is put relative to. "On" in this format is the spatial relation that connects objects "book" and "table". It is possible to chain multiple spatial relations with multiple entities. The user can also provide a specific location for an entity. More complex examples are the following:

```
put "table" on "floor"
put "plant" on "table" under "lamp"
hang "picture" on "wall" at (0 10)
```

The Put system includes a parser for this simplified syntax and it does not deal with the complexities of the natural language processing. 3D geometry information of entities are matched with a name and then they are referred later using that name. The Put system differentiates uses of spatial relations based on the objects. Contact points or any other related properties for placement task is defined for each object. While constructing scenes, it takes into account different senses of the spatial relations. Different uses of "on" or "in" are handled correctly by the system. To ease the computation, PUT uses axis aligned bounding boxes for entities. This also helps to determine front, top and side surfaces and interior of the objects.

Since the Put system focuses only on the scene construction task, its natural language processing capabilities are limited. The system is not able to construct a scene from natural language text for instance from a newspaper article since it requires manual steps to define objects in the scenes and their properties. However their work in defining properties and rules for spatial relations is significant.

2.3 CarSim

CarSim [3] is an automatic text-to-scene conversion system that simulates car accidents from written accident reports in a 3D environment. CarSim architecture has two modules that communicate via a formal description of the accident. The first module does the language analysis and it converts accident reports into a formal description. The second module is the 3D visualizer and it converts the formal description into 3D animated scenes. A generated scene from CarSim system is shown in Figure 2.2 below [4].

The language analysis module is capable of understanding collision verbs and objects that are part of the accident. This module uses WordNet and Link Dependancy Parser and regular expressions to determine the collision verbs and actors in the accident [4]. Three main components are extracted from the written reports by this module [5]. First is the scene that describes static parameters such as weather and road conditions. Second is the road objects such as cars, trucks and trees. The third is the collisions between road objects. After these components are identified by the language analysis module and converted to the formal description, the scene is generated.

The visualization module of CarSim takes the formal description as input and converts it into a scene. This process includes a planner. The planner sets the scene

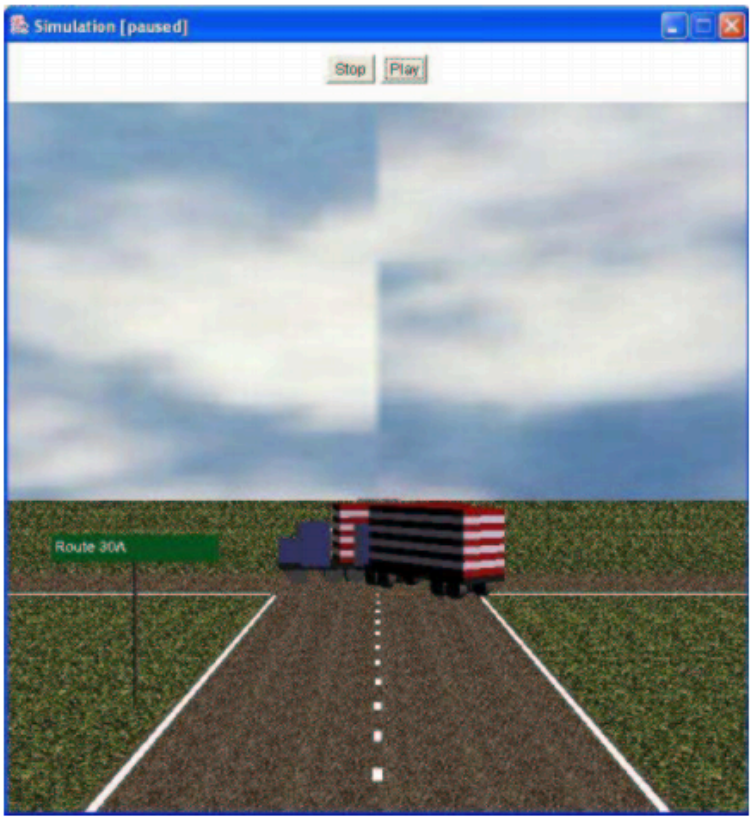

Figure 2.2: A generated scene from CarSim system shows collision of two trucks.

by placing static and dynamic objects and sets a start and end point for vehicles. The planner also determines a path for the vehicles and fixes the path by adding collisions.

CarSim uses a few 3D models and a limited vocabulary since it is very domain specific. However it is an important system that converts real world text into not only static scenes but animations. CarSim generates animated scenes from written text in a restricted domain successfully.

2.4 WordsEye

WordsEye [6, 7] is an automatic text-to-scene conversion system. It converts natural language input into static 3D scenes. WordsEye aims to ease the 3D scene generation

process using a more natural interaction and aid classical scene creation tools. The system can be used online[1] and many example scenes created by the system is available on the web as well.

WordsEye is an ambitious project with a large collection of 3D models and lexical sources. WordsEye database includes over 12000 models, and many 2D images for textures or to decorate scenes. In addition to model mesh data, WordsEye defines extra information for 3D models. These are skeleton data for human and animal models, information of model parts, default sizes, functional properties. Another extra properties defined for 3D models are spatial tags [8]. The spatial tags are very important in the depiction of spatial relations. For spatial relations, certain interaction areas like canopy area, base area, cup area, enclosures, side surfaces, top surfaces, stems and touch points are defined for each model. The manual tagging of 3D models enable the system to cover many different use cases for spatial relations.

WordsEye handles verbs by using a method called posing. The model is turned into a pose that resembles a specific verb. For human and animal models this is achieved by using the skeleton data. Poses are predefined for specific actions and model takes that pose if the specified verb is encountered. The pose of the model changes as needed if the specific action interacts with another model. For instance, a human model can be posed while riding a bicycle and the legs and arms of the model takes appropriate positions.

WordsEye uses a lexical and real world knowledge resource called Scenario-Based Lexical Knowledge Resource (SBLR)[9]. The SBLR contains lexical, semantic, and contextual information. Some parts of SBLR are built semi-automatically from WordNet[10] and FrameNet[11] but the knowledge is also extended by adding data

[1]http://www.wordseye.com/

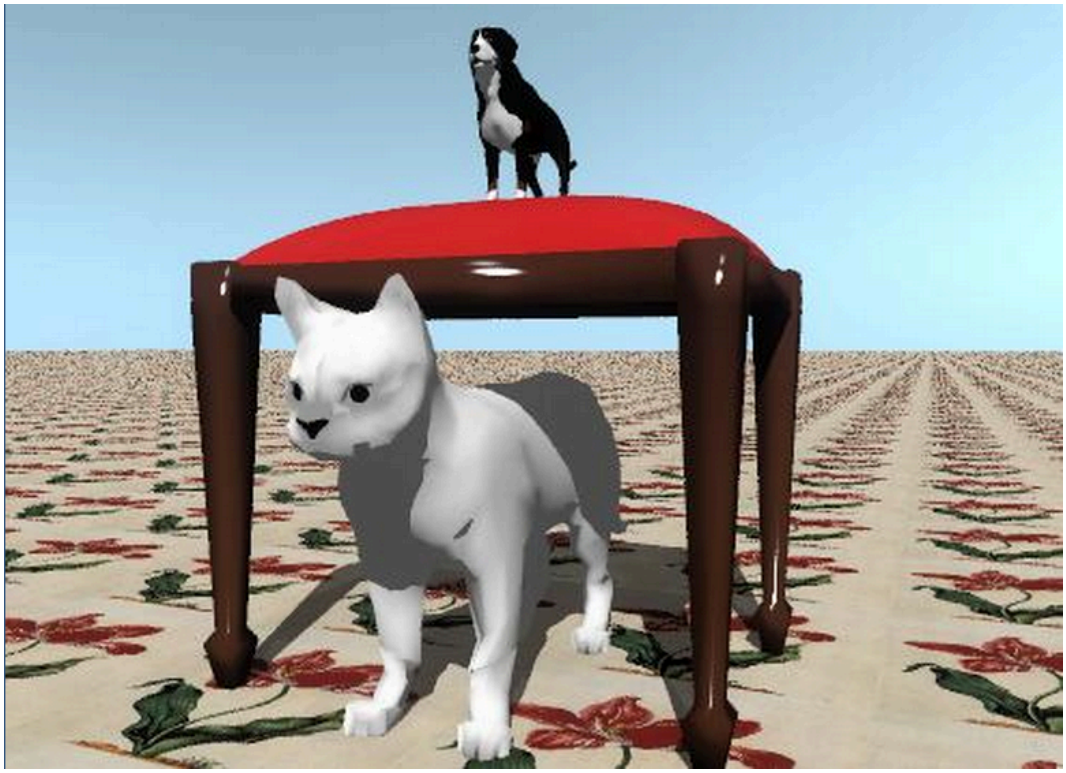

Figure 2.3: A scene from the WordsEye system of "The small dog is on the red stool. The flower texture is on the ground. The texture is 2 feet wide. The large white cat is under the stool."

about 3D models and additional resources like Wikipedia, PropBank[12] and other corpora.

WordsEye is an inspiration for our scene construction work. However, WordsEye focuses on creating visually more detailed scenes where we present a modest scene generation system and instead focus on question answering and reasoning about the scene. As WordsEye, our system is rule based and depends on lexical resources that are mostly built manually. Rule based approaches are good enough for establishing the scene construction framework. However, extending rule based systems manually is a daunting process. As part of the future work, we believe that the connection between natural language and visual scenes can be learned with statistical machine

learning algorithms.

2.5 DESCRIBER

DESCRIBER [13] is a system that generates spoken language to describe objects in a computer generated scene. The system demonstrates a statistical learning based approach for such a task. In this system, randomly generated rectangles (random sizes, positions and colors) are matched with their natural language descriptions. DESCRIBER learns the descriptions of these computer generated images in a grounded learning framework. It has been shown that the machine can successfully describe unseen rectangles.

The aim of DESCRIBER is to develop a system which can learn domain specific rules of language generation with supervised training. This system deals with computer generated images instead of real images since this has many advantages. Computer generated images are noise free and they can be reproduced easily in large amounts. They have well defined digital features such as RGB color and position on screen. A sample scene from DESCRIBER system is shown in Figure 2.4 where rectangles with different visual properties are placed and one of them is targeted with the pointer. The user is expected to describe the target object by speaking in a natural way. At this stage, natural spoken descriptions are obtained from a male speaker for a target object in the scene. Later these are transcribed manually and together fed into the system with their paired visual scenes. Then syntactic and semantic structures are extracted by learning algorithms. Firstly the words are clustered into classes. This step is used to separate words related to size, colors and spatial relations. Then the visual features like color, area, height-to-width ratio and position on screen are associated with word clusters. Then the word order constraints are modeled. After

that stage, the system is able to generate spoken descriptions of novel scenes. These spoken descriptions include syntactically accurate adjective-noun phrases and relative spatial clauses. The generated spoken language is found to be satisfying by human judges. The system is evaluated by measuring the users' ability to select the intended rectangle from the description.

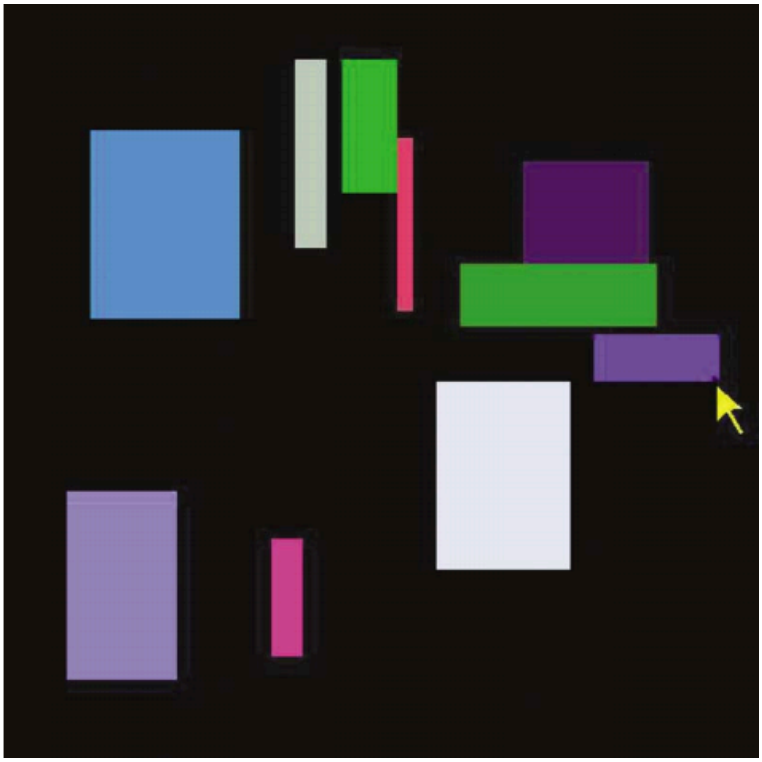

Figure 2.4: A image from visual description task of DESCRIBER system.

The DESCRIBER system is not without constraints. DESCRIBER works with very constrained visual objects (rectangles and squares) that have well defined properties (2D coordinates, RGB colors etc). The system uses task specific simplifications which will cause problems when the system is expanded to a more complex domain.

2.6 CONFUCIUS

CONFUCIUS [14] is a storytelling system that visualizes single English sentences into 3D animations. It combines natural language and computer graphics techniques to generate complex high-level animations, sounds and speech. An example scene from the CONFUCIUS system is shown in Figure 2.5 below.

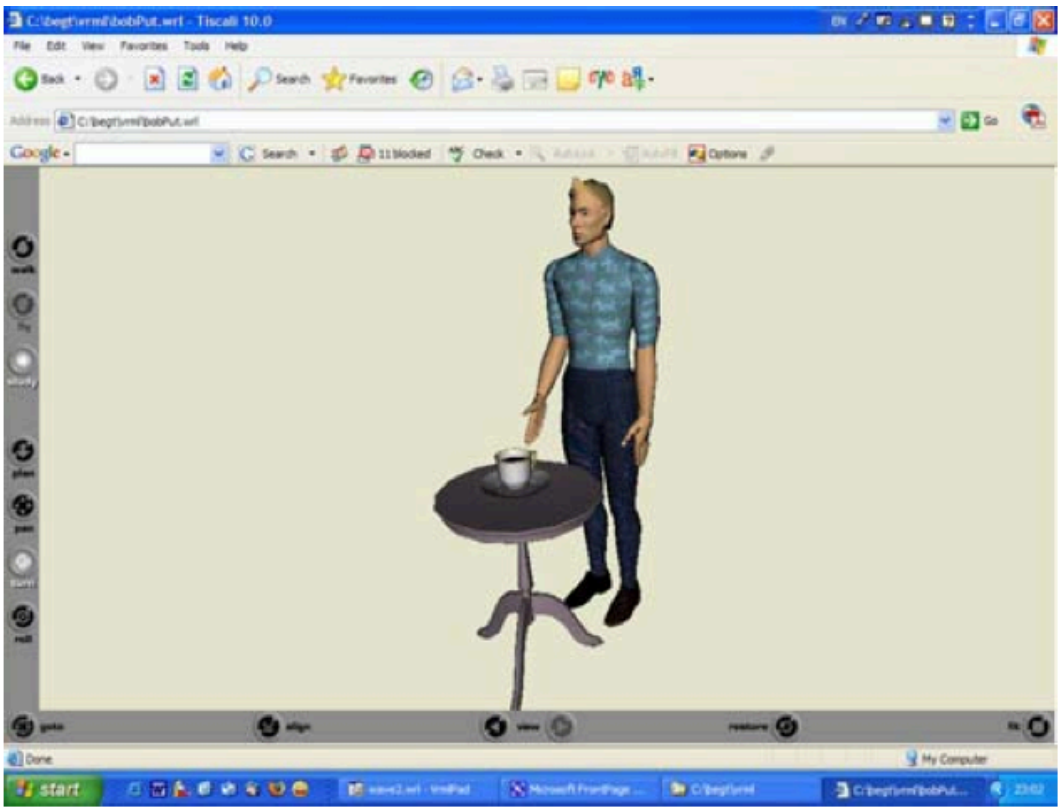

Figure 2.5: A generated animation from the CONFUCIUS system of "John put a cup on the table."

CONFUCIUS is a more complex system in the sense that it creates not only static scenes but high level animations and speech. It is one of the systems that handles action verbs that takes a span of time. The system currently handles a single sentence with an action verb. The sentences are picked from the children stories and used during the development and evaluation of the system. In order to visualize

the natural language input, semantic information is extracted from the sentences. For nouns, CONFUCIUS uses WordNet, a proper noun list and a popular given names list to mark proper nouns, persons and gender. These semantic features are needed for the animation generation module. Semantic features of the verbs are used to determine verb categories. Based on the category, a verb is associated with a predefined animation or if it is a verb of speaking, the words in quotation marks are fed in to the text-to-speech (TTS) engine and lip movement is generated for the speaker model. Semantic features of adjectives correspond to the visually or audio presentable properties. Currently, CONFUCIUS has 40 models and it covers 25 verbs like "walk", "jump" or "push" which are considered as the building blocks of more complex actions.

The overview of the architecture of CONFUCIUS is as follows; the NLP module parses the input sentence and extracts semantic information. Then the media allocator assigns the content to three different media which are animation, character's speech and narration. The animation engine takes semantic representations and visual knowledge to generate 3D animations. The output of the text-to-speech engine and animation engine are combined by the synchronizing module and a VRML[15] file is produced. Finally, narration integration module integrates VRML file with the narrator agent and completes the presentation.

CONFUCIUS uses the H-Anim standard[16] for character modeling and animations [17]. H-Anim provides different levels of detail for different applications. CONFUCIUS uses a moderate level of detail as it is sufficient for a language visualization system. The system is also capable of running two parallel animations for the upper and lower body of human models. As speech is part of the CONFUCIUS system, the system handles lip syncing and facial expressions.

The aim of the CONFUCIUS system is to create a narrator system that visualizes natural language input into animations as accurately as possible by animating human body parts, handling lip synching and facial expressions. Our work shares similar aims as CONFUCIUS on the language visualization side although we only work with the static scenes. As a secondary aim, we investigate the possible uses of a language visualization system that can aid answering certain types of inference problems which cannot be easily answered by only analyzing the language itself.

Chapter 3

COMPONENTS

A language visualization system is composed of many different components that can be categorized under two broad subjects which are 3D graphics and natural language processing. The aim of this work is to use theories and practical solutions from both of these fields and bring them together to propose a method to construct 3D scenes from language. Our system is also capable of answering certain types of spatial inference problems using 3D object geometry and basic language understanding.

The three critical components our system uses are Alice, WordNet and CoreNLP. Alice is an open-source 3D programming environment and it forms the basis of our language visualization system. WordNet is a large electronic lexical database of English and it is used heavily by our system as well. CoreNLP suite from Stanford is a collection of natural language processing tools and they are used for language related tasks in our system. This chapter describes these components which our system is built upon and gives details about how each individual part contributes to it.

3.1 Alice

Alice [18, 19] is a 3D graphics programming environment developed at Carnegie Mellon University. In the beginning, the goal of Alice was to ease 3D behavior programming for the novices, but later the tool is found to be useful to teach problem solving, computational thinking and computer programming to students from college

to mid-school [20, 21, 22, 23].

Alice comes with a 3D scene editor, a tree view for 3D objects and a drag-drop programming interface for scripting [24]. A screenshot is shown in Figure 3.1 below. A typical Alice workflow has two phases, first a scene is created by placing 3D models into the scene and then the user begins scripting for behavior using the drag and drop interface.

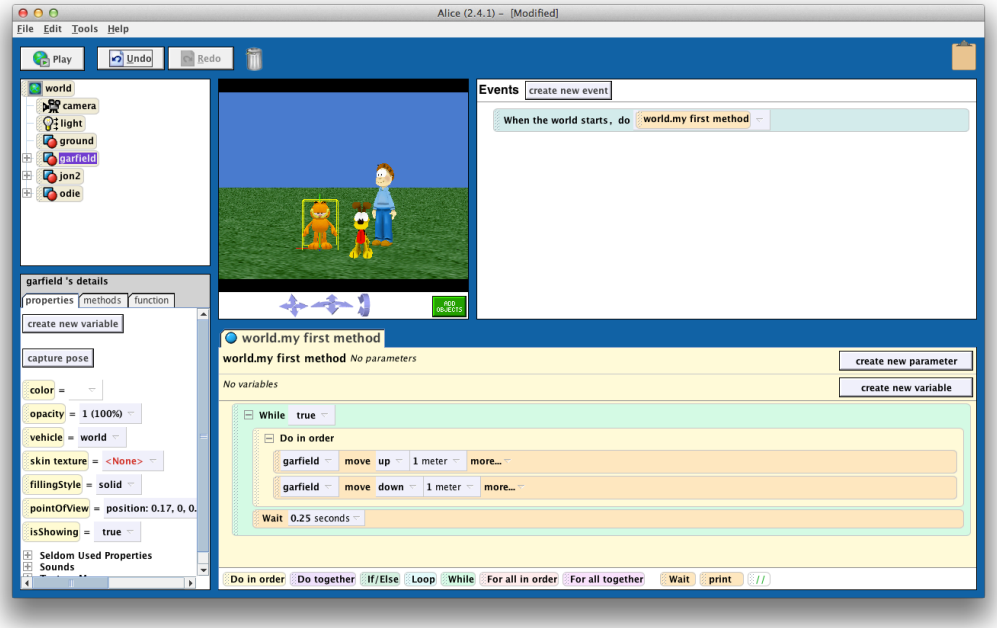

Figure 3.1: A screenshot from Alice v2.4. The toolbar is present at the top to play animations in the scene. The object tree can be seen on the left side. Detail view for model details is shown below it. The scene editor is present in the middle. On the right side, declared events are shown. The drag and drop scripting area is below them.

The design of Alice makes it a great educational tool for problem solving and object oriented programming. There are many contributions of Alice and we should list some of them to give reader a better understanding of why it is an important tool. Alice removes the need to use a global coordinate system, instead it attaches a

coordinate system to every object in the scene. The modification of the objects are done relative to themselves or the objects around them. This enables a more natural interaction where an object can be moved forward or it can be turned to a side without referencing a global coordinate system. The object tree on the left side provides a instant view of the objects in the scene and makes selection easier in a complex scene. The parts of the objects or the child - parent relationship between objects can be viewed on the tree view as well. The drag and drop interface for programming lets novices avoid syntax errors and saves typing time. The object oriented programming concepts can be easily taught with Alice as each 3D model in a scene corresponds to objects in the program and methods are basically actions of those objects. The user can instantly see the effect of their code in the scene, which makes learning and debugging easier.

Alice has an extensive gallery of 3D models consisting of many categories like people, animals, vehicles, furniture etc. In our system, 528 models are used from the Alice gallery. At first we were only interested in using Alice models in our system. Later, we found out that Alice models use a proprietary format and that caused some problems during the implementation. We also realized that many of the functionality built in Alice was needed to be reimplemented in our system. Alice API was already mature with many years of research and had already been tested with many users therefore we decided to build our system on top of Alice.

Alice is an open-source program and that made it possible for us to build our language visualization system upon. Alice source code is available through their website for the version 2.0. However, we could not make the source code work as it is downloaded and we spent a significant amount of time to make it usable. Our version of the code will be released with our contributions, enhancements and fixes

(and bugs) [1]. This will save a significant amount of work for the interested user.

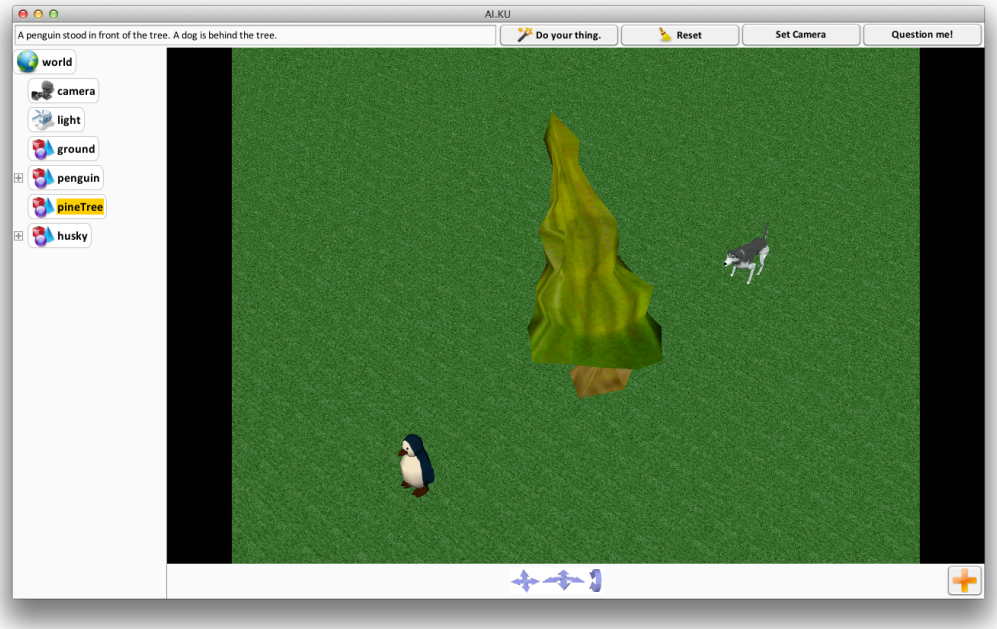

Figure 3.2: A screenshot from our LangVis system.

We have made many changes to Alice code and interface. Our changes to Alice interface is shown in Figure 3.2. We removed drag and drop scripting area, details view and toolbar since they are not needed for our purposes. We kept the object tree view since it gives a quick summary about objects in the scene. Instead we placed a textarea on top which is used to input natural language text. Several buttons are also placed to convert the text to 3D scene, to clean the scene, to set the camera and also to switch to question answering mode.

For our language visualization system, we use model modification and rendering capabilities of Alice so that we can focus more on implementing language related features rather than re-implementing 3D scene manipulation functionalities. Alice has

[1] `http://github.com/ai-ku/langvis`

many built-in functionalities like loading meshes for a 3D model, applying textures, transforming models, basic animations, placing light and camera for a scene. Built-in 3D manipulation capabilities of Alice makes it easier to test quick ideas and debug the problems. This is critical for the implementation of such a system.

3.2 WordNet

WordNet [25] is a large lexical database of English that consists of nouns, verbs, adjectives and adverbs that are grouped into categories called synonym sets (synsets). Each synset represents a distinct concept (word sense) and these synsets are linked to others by semantic relations.

WordNet is a popular lexical resource in natural language processing community and it has been used in other language tools like VerbNet and Ontonotes [26]. The usefulness of WordNet comes from the way it is structured. The main relation between words in WordNet is synonymy. These synonyms denote the same concept and they are grouped together to form synsets. WordNet has 117000 synsets and it also defines semantic relations between these synsets. In addition to that, WordNet includes a definition(gloss) for each synset.

The semantic relations between words and word senses that are incorporated into WordNet are listed below [10].

- ***Synonymy*** is the main relation between words in WordNet since the words in WordNet are grouped into synsets. If two word forms have at least one common word sense then they are synonymous and grouped into the same synset that represents that sense. It is a symmetric relation between words.

  e.g. *board - plank*

- ***Antonymy*** is the opposite semantic relation between word forms. It is also a symmetric relation and it is important in the organization of adjectives and adverbs.

  e.g. *dry - wet, friendly - unfriendly*

- ***Hyponymy*** is the 'is-a' relation between words. This relation is important to organize nouns into hierarchical structure.

  e.g. *crimson - red*

- ***Hypernymy*** is the semantic relation that is inverse of hyponymy.

  e.g. *red - crimson*

- ***Meronymy*** is a complex semantic relation where a word is part-of another word.

  e.g. *window - building*

- ***Holonymy*** is the semantic relation that is inverse of meronymy.

  e.g. *building - window*

- ***Troponymy*** is for verbs what hyponymy is for nouns. However the resulting hierarchies are much shallower.

  e.g. *lisp - talk*

- ***Entailment*** relations between verbs are also present in WordNet.

  e.g. *snore - sleep*

The presence of each relation for each word category is shown in the Table 3.2 below.

Semantic Relation	Syntactic Category
Synonymy	N, V, Aj, Av
Antonymy	Aj, Av
Hyponymy	N
Hypernymy	N
Meronymy	N
Holonymy	N
Troponymy	V
Entailment	V
N: Nouns, V: Verbs, Aj: Adjectives, Av: Adverbs	

Table 3.1: Semantic Relations in WordNet

These semantic relations are used heavily in our system, and in the next chapter we will discuss how they are integrated with the rest of the system.

3.2.1 JWI

JWI [27] is a Java library for interfacing with WordNet written by Mark Alan Finlayson at MIT. We are using WordNet 3.0 in our language visualization system and JWI provides excellent support for all WordNet features needed by our system. JWI also includes a stemmer which we use to stem words before querying the WordNet database. The library is open source and distributed with a MIT license so that it can be used freely for any purpose. The library is used heavily in our system and it saved a tremendous amount of time.

3.3 CoreNLP

CoreNLP [28] suite from Stanford provides many useful natural language analysis tools like parser, named entity recognizer, part of speech tagger and a coreference resolution system for English. These tools make a solid foundation for our language visualization system as they are vital for understanding written text. These tools are used in conjunction with our system to create accurate scenes from a written text input and then answering questions about the scene.

3.3.1 Part-Of-Speech Tagger

One of the critical tools for our language visualization system is the part-of-speech (POS) tagger from CoreNLP. A POS tagger is a program that assigns parts of speech to each word such as noun or verb. POS tags are used in many places in our system. We use them to query the WordNet database or to take certain actions based on the tag when generating the scenes. They are also used by the parser. POS tagger for English in CoreNLP uses PENN Treebank POS tags [29]. A total of 45 different Penn Treebank POS tags are shown in Appendix A.

3.3.2 Parser

Another important tool we use from CoreNLP package is the parser. Stanford parser is a probabilistic natural language parser and it outputs probabilistic context free grammar (PCFG) phrase structure trees and also Stanford dependencies [30, 31]. Stanford Dependencies are grammatical relations between words in a sentence. They are triplets in the form of name of the relation, governor and dependent. These grammatical relations are organized in a hierarchy and the most general grammatical relation is dependent(*dep*). If a more specific relation is identified between a head

and its dependent, then the relations down in the hierarchy are used. The hierarchy of typed dependencies (Stanford dependencies) [32] are shown in Appendix B.

Stanford dependencies are designed to be useful in practical applications. The dependency output is uniform and the relationship is binary. A grammatical relation is present between two words or not. The output of the parser including the parse tree and typed dependencies for a sample sentence is shown below.

"Alice noticed with some surprise that the pebbles were all turning into little cakes as they lay on the floor, and a bright idea came into her head."

```
(ROOT
  (S
    (S
      (NP (NNP Alice))
      (VP (VBD noticed)
        (PP (IN with)
          (NP (DT some) (NN surprise)))
        (SBAR (IN that)
          (S
            (NP (DT the) (NNS pebbles))
            (VP (VBD were) (RB all)
              (VP (VBG turning)
                (PP (IN into)
                  (NP (JJ little) (NNS cakes)))
                (SBAR (IN as)
                  (S
```

```
                    (NP (PRP they))
                    (VP (VBD lay)
                      (PP (IN on)
                        (NP (DT the) (NN floor)))))))))))
    (, ,)
    (CC and)
    (S
      (NP (DT a) (JJ bright) (NN idea))
      (VP (VBD came)
        (PP (IN into)
          (NP (PRP$ her) (NN head)))))
    (. .)))

nsubj(noticed-2, Alice-1)
root(ROOT-0, noticed-2)
det(surprise-5, some-4)
prep_with(noticed-2, surprise-5)
mark(turning-11, that-6)
det(pebbles-8, the-7)
nsubj(turning-11, pebbles-8)
aux(turning-11, were-9)
advmod(turning-11, all-10)
ccomp(noticed-2, turning-11)
amod(cakes-14, little-13)
prep_into(turning-11, cakes-14)
```

```
mark(lay-17, as-15)
nsubj(lay-17, they-16)
advcl(turning-11, lay-17)
det(floor-20, the-19)
prep_on(lay-17, floor-20)
det(idea-25, a-23)
amod(idea-25, bright-24)
nsubj(came-26, idea-25)
conj_and(noticed-2, came-26)
poss(head-29, her-28)
prep_into(came-26, head-29)
```

In our language visualization system, we use typed dependencies instead of the parse tree representation. The Stanford Parser outputs typed dependencies in two different formats, in one all words are present in the output, in the other the parser gives a collapsed representation for the dependencies. For our system, we use the collapsed form which is sufficient for our needs. The details of the implementation will be discussed in the next chapter.

Chapter 4

IMPLEMENTATION

This chapter describes the implementation details of our language visualization system. Starting with the high level workflow of our system, we will go deeper and cover the details of language analysis, converting output of language analysis to intermediate data structures and then translating them to 3D static scenes. Construction of 3D scenes include the steps of finding the accurate 3D models, applying modifications on models and placing them in the scene. These steps will be covered in this chapter as well. After the scene construction section, we will cover the details of the question answering module.

Our system consists of three main components. First one is the language analysis component, the second is the scene construction component and the third is the question answering module. Language Components section will cover the problems in language understanding for a visualization system and how we cope with those. Scene Construction section will talk about converting the language analysis results into scenes by using many different resources, applying modifications and satisfying constraints. Lastly, Question Answering section will describe how the constructed scene can be queried by the user and how our system is capable of solving certain types of spatial inference problems. Each of these parts are covered in full detail in its dedicated section in this chapter but first we will look at the general workflow.

4.1 Workflow

Before going into the details of three main components, it is important to understand the general workflow in our system. The process of converting natural language to 3D scenes in our system is shown in Figure 4.1.

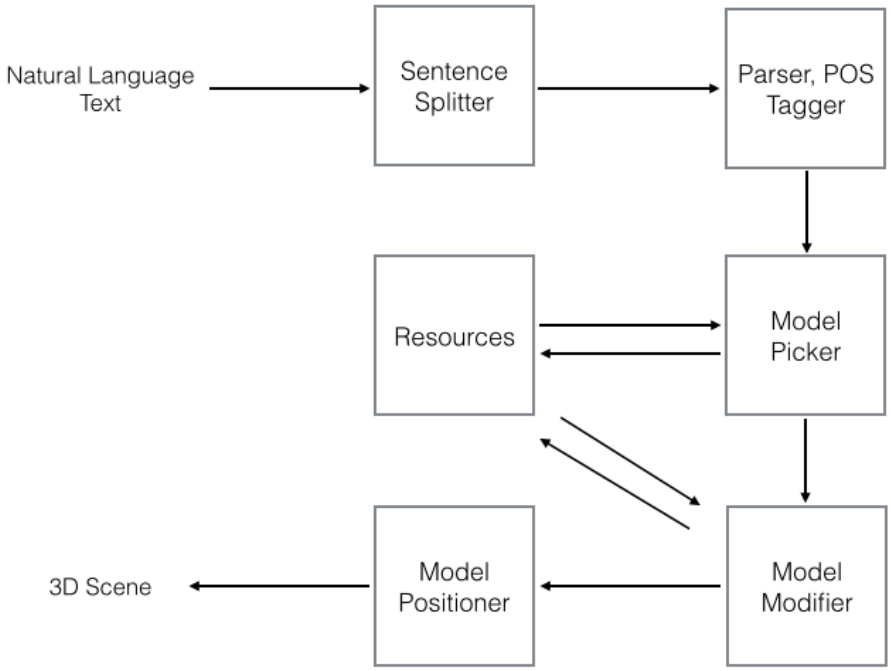

Figure 4.1: Text-To-Scene conversion process

Interaction with our language visualization system starts with the natural language input. This input might be more than a single sentence. After the generate the scene button is pressed, these sentences are fed into a sentence splitter and then processed one at a time by our system. Each sentence is then parsed by the Stanford parser which outputs the dependency structure of the sentence. The parser also uses POS tagger to annotate word types for each token in the sentence. After this stage,

our model picker module tries to come up with an accurate model for the nouns in the sentence. It uses lexical resources like WordNet, our own datasources and 3D model gallery. After an accurate model is picked, if there is an adjective in the sentence that modifies the properties of that model then these modifications are applied by the model modifier module. Lastly the models are positioned in the scene to satisfy any spatial relation constraints if they are present.

This is the general workflow of the system which we will explain more in the following sections. The following section may refer to different parts of the system in a non-sequential order so a general overview of the system is given here.

4.2 Language Components

Our system currently covers three main word types which are nouns, adjectives and prepositions. Dynamic verbs are intentionally left out at this stage as we limited ourselves to only static scenes. Even though they are covered by other systems using posing [6] we believe that this approach is not necessary for our question answering and spatial inference task. Our system only accepts verbs like *to be* and *stand* that contain no actions. The other three word types (nouns, adjectives and prepositions) are sufficient to describe static scenes when combined with these verbs.

The part of speech (POS) tagger and parser from CoreNLP is used to extract POS tags for the given piece of text. The tagger outputs Penn Treebank POS tags [12] for each word. Our system has defined procedures for each POS category that are run in a particular order to generate a scene. The following sections discuss how each part of speech is handled by our system.

4.2.1 Nouns

In our system, nouns correspond to 3D models from the Alice gallery. The system tries to come up with an accurate model for the noun when it is encountered in a sentence. Although this sounds trivial, processing nouns requires many computational steps which may not be obvious at first. The following are some of the problems in representing nouns visually:

- A noun may refer to a non-physical entity.
- A noun may refer to a physical entity but it might be too general.
- A physical entity can be referred to in multiple ways.
- There is not always a one-to-one correspondence with a noun and a 3D model but sometimes a noun phrase can refer to it. The opposite is true as well. A noun can refer to more than one object.
- A noun can be ambiguous. It is impossible to get the word sense by just using the word itself without the context.

For our language visualisation system, we are not concerned with abstract or conceptual nouns. We are only interested in physical entities which can be seen and touched in a physical world that our model gallery covers. These constraints remove many nouns in language from our candidate set because they cannot be visualized. It is easier to agree on a depiction of a physical entity, however this is not the case for abstract concepts like *democracy*. Although there are attempts by other systems[6] to visualize these concepts by using 3D text or some representative images, we decided to eliminate them for the purposes of this research.

For a system that needs to pick accurate models for words, it must have a mapping between these two. However listing all physical entities in language and matching them with 3D models appears to be infeasible. The main reasons for this is that there are many words in language whereas 3D model libraries are limited in size and nouns do not always have one-to-one relationships with them. The relationship can be one-to-many and many-to-one. There is a need for another layer between these two lists of models and words. WordNet helps us to form this middle layer.

Every word in WordNet belongs to a synset (synonym set) and words in a synset have the same sense [25]. We matched our 3D models to the synsets that describe them best. Therefore we formed a relationship between a word and its synset and then by using that synset the system can reach the model for that word. A synset can also be matched by more than one model in a library. For example, there does not need to be only one *cat* model in our object database. This mapping was done manually for the 528 models in the Alice object gallery. As a synset consists of many different words, our system can handle a much larger number of nouns.

A small portion of our Synset-to-3D model map datasource is shown below. Each synset has different attributes which are gloss, hint and id. Gloss describes the synset, it is included as it makes the sense clear for the human reader. Hint attribute is included since it is much more easier for human reader to understand which object the synset refers by just looking at the hint. These two attributes are added because the name of the model may not always be descriptive. Id attribute is the synset identifier in the WordNet database. Our system is only interested in this identifier and uses that internally in many places. Synset identifiers are unique for the word sense. Therefore many models that are referred by the nouns in the same synset should be listed under the same synset identifier. It can be seen below that synset

entry of blender, book and bookcase include only a single model whereas synset entries of town house and bird include more than one.

In our data source, each model is appended as a child to the synset entry it belongs to. Models only have one attribute which is the relative path of the model in the 3D model gallery. These are the 3D models from the Alice gallery. We did not make any changes to Alice models and used them as is in our system.

```
<Synsets>
 ...
 <Synset gloss="an electrically powered mixer with whirling blades that
 mix or chop or liquefy foods" hint="blender" id="SID-02850732-N">
  <Model path="Kitchen/Blender.a2c"/>
 </Synset>
 <Synset gloss="a written work or composition that has been
 published (printed on pages bound together)" hint="book" id="SID-06410904-N">
  <Model path="Objects/Book.a2c"/>
 </Synset>
 <Synset gloss="a piece of furniture with shelves for storing books"
 hint="bookcase" id="SID-02870880-N">
  <Model path="Furniture/Bookcase.a2c"/>
 </Synset>
 <Synset gloss="a house that is one of a row of identical houses situated
 side by side and sharing common walls" hint="town_house" id="SID-04115256-N">
  <Model path="City/Townhouse1.a2c"/>
  <Model path="City/Townhouse2.a2c"/>
  <Model path="City/Townhouse3.a2c"/>
```

```
  <Model path="City/Townhouse4.a2c"/>
  <Model path="City/Townhouse5.a2c"/>
  <Model path="City/Townhouse6.a2c"/>
 </Synset>
 <Synset gloss="warm-blooded egg-laying vertebrates characterized by
 feathers and forelimbs modified as wings" hint="bird" id="SID-01503061-N">
  <Model path="Animals/Bird1.a2c"/>
  <Model path="Animals/Bluebird.a2c"/>
 </Synset>
 ...
</Synsets>
```

For the representation of nouns, we also benefit from the hyponym and hypernym relationships (is-a or kind-of relationship) of WordNet. A noun in WordNet is positioned in a tree structure where the root node is "entity". Every noun is a subclass of that root node which means that every noun is an entity. Nouns become more detailed and specific deeper in the tree structure. This kind of hierarchical relation between words gives useful information like "A cat is an animal" and "A car is a vehicle".

For all nouns we know their parent categories. This knowledge enables us to show a model for a specific type of noun when the user enters a more general name instead. When the user asks for an animal it is perfectly acceptable and reasonable for our system to show a *bird* or a *cat* or a *dog* or any other it finds as a child of the *animal* noun. This requires us to match our 3D models to the synset of the deepest noun we can find in the tree. By doing that we can fetch the model that the user typed specifically or we can again find a suitable model for a more general word.

In our system, we do not disambiguate word senses, but we use the first word sense (or synset) that is associated with that word. The first word sense in the WordNet is the most commonly used sense of the word [25]. In future work, word sense disambiguation can be integrated and this can improve the overall accuracy of natural language understanding.

4.2.2 Adjectives

In grammar, adjectives are the describing words for nouns and noun phrases. They give more information about the object signified. In our system, we use the CoreNLP parser to parse natural language sentences and it gives us the modifier(s) of each noun in the sentence.

One important observation about adjectives in language is that many of them cannot be visualized in a 3D scene. In our language visualization system, every 3D model is associated with nouns and these models have six different properties which are visibility, size, position, orientation, color and transparency. The three of these properties (size, color and transparency) can be modified by adjectives. The adjectives that modify these three properties can be called depictable or visualizable adjectives. We can also refer to them as model modifying adjectives since they have a visual effect on the appearance of the 3D model in a 3D scene.

For instance *jealous* is a type of adjective that does not change one of the three properties listed above and therefore it is not visualizable in our system. Our system knows which adjectives have a visual effect we can control and does the necessary modifications for the model. If it is not depictable, the system just skips that modifier. However, a *jealous girl* model can be present that is separate from the *girl* model. It should be picked and shown even if *jealous* is not a depictable. The sequence is

important here. The system first searches for a corresponding model for the 'adjective + noun' pair. This is done while the system is processing the nouns. If a model for that pair is found, that would be a more accurate result and system would show that. If the adjective is a model-modifying type, the system modifies the model accordingly. If not, it ignores the adjective and just retrieves the corresponding model for that noun.

For the spatial inference task we only focused on depictable adjectives related to size. The color and transparency properties of a model can also be changed manually, but it is not automated yet. In the future versions, that can be implemented as well. The adjectives related to size can modify one or more of the three size related properties which are width, height and depth. As adjectives modify nouns in a language, in our language visualisation system they correspond to procedures that are applied to models. Similar to what we did about nouns before, for these kinds of adjectives, we did a matching between their WordNet synsets and corresponding procedures. These procedures scale the model by using a scalar value that is obtained based on the synset. By forming a map with synsets, the adjectives that have the same word sense are associated with the same procedure. The user sees the same effect on the scene no matter which word she picks. The mapping between adjective synsets and scalar values are shown in Appendix C.

WordNet does not have hypernym or hyponym relations for adjectives since it is not very meaningful for adjectives to have hierarchical relations. However another important information extracted from WordNet is the attribute information of an adjective. For example, for the adjective *tall*, WordNet tells that it modifies the *height* attribute or for *fat* it returns *fatness*. These attributes of the adjectives are matched to the attributes of the 3D models and when the system encounters any

of them it modifies the related attribute accordingly by using a scalar value for the degree of the adjective.

4.2.3 Prepositions

Prepositions are the word types which denote temporal and spatial relations. They have complements which they relate to the context they occur in. For our spatial inference task we limited ourselves to spatial relations. These are the prepositions that denote location and direction. Since our system only handles static verbs, direction related prepositions are not covered. Location related spatial prepositions like *in front of, behind, near, next to, beside, above, below, on, in* are mapped to procedures in our system. The complete list of these procedures are shown in Appendix D. These procedures take two arguments where one of them is a landmark object which the other object is placed relative to. The object is placed by translating it in the corresponding direction denoted by the preposition. The translation amount is determined by the size of the objects. Collisions may occur when two objects are placed based on the same spatial relation. In the future versions, collision prevention can be added as well. The spatial relations are held true for the facing direction of the models. However, at the current stage of our system, the models are not automatically turned around therefore the spatial relations are valid for the position of the camera therefore the view angle of the user as well. For prepositions like *near*, multiple directions are possible and in that case the direction is chosen randomly. First these procedures try to place an object according to the spatial relation while keeping the position of the other object fixed. If these objects already have other spatial constraints on them, then they are placed without breaking the previously applied constraints. This is done by reversing the spatial relation and the landmark object or by moving the

other objects that are part of the previous spatial relations. Prepositions are not included in WordNet, so a mapping from the synsets to procedures is not possible. We directly mapped words to related procedures in this case.

This section summarized how each POS contributes to our system. In the next section, we will discuss scene construction which makes use of these to construct scenes from natural language sentences.

4.3 Scene Construction

4.3.1 Model Placement

After an empty scene is presented, the user can directly enter a piece of text that describes a scene with multiple sentences or she may choose to build the scene incrementally providing one sentence at a time. In either case, after the text entering is complete, the system splits sentences (if there is more than one) using CoreNLP and parses each sentence. For each sentence, nouns are extracted and the system searches WordNet for the synset of the target word. The word types other than nouns are processed at a later stage in the system. For the nouns, the search is done by supplying the word itself and the part-of-speech tag of the word. POS tag comes from the POS tagger in the previous step where the sentence is parsed. If the synset of the target word is found in the WordNet database, then the system searches for a 3D model in our synset-model mapping data source by using the synset identifier.

If there is not a suitable model for that synset, the system switches to the children of the synset using hyponym relations in the hierarchy of WordNet. If there is more than one model for that particular synset, the system picks a model randomly. If there is no exact match, to come up with a similar model for the noun, our system asks user whether to search coordinate terms or not. These are the nodes that share

a hypernym in the WordNet hierarchy. It asks the user because the result may not be always close to what the user has in mind. For instance when the user asks for *cheese* the system cannot find it and comes up with *banana* which are both children of *food*. After these efforts if a suitable match is not found, the system just skips that noun. This is the case for abstract concepts.

4.3.2 Model Modification

When a model is picked succesfully, all of its modifiers (adjectives) are extracted from the parser output. The parser output for a sample sentence is shown below. It is seen that adjectives *giant* and *small* are marked with amod (adjectival modifier) relation in the parser output.

```
A giant dog is behind the small cat.


det(dog-3, a-1)
amod(dog-3, giant-2)
nsubj(is-4, dog-3)
root(ROOT-0, is-4)
det(cat-8, the-6)
amod(cat-8, small-7)
prep_behind(is-4, cat-8)
```

Currently, our system covers only modifications that are related to the size of the model. Based on the modifier, width, height and depth properties of the models are changed. A mapping is done between the synset of the modifier and a scale tuple as shown in the Appendix C that is used by the scaling procedure. If the modifier of the

noun is found in that mapping than the procedure is applied to that model. If it is not found, the system just skips that modifier.

4.3.3 Constraint Resolution

After this stage, our system extracts spatial relations (prepositions) from the sentence. Models are placed into their correct places to satisfy the constraints in the sentences.

The parser output in section 4.3.2 shows that the *cat* has a spatial relation *behind* with the *dog*. In this sentence, *behind* and *nsubj* grammatical relations give us enough information to create a constraint on the scene.

As described in the section 4.2.3 about prepositions, a basic constraint resolution system is used to satisfy the constraints. Constraint resolution is necessary as the number of models increases in the scene. In addition to that, we expect the user not to enter any conflicting constraints. In that case, the system cannot satisfy all of them and produce satisfactory results.

4.3.4 Scene Conversion Result

A fully constructed scene can be seen in Figure 4.2. This is a complex scene as it includes many different models and spatial relations. After the construction is complete, the user may go on with the question answering, which will be described in the next section.

4.4 Question Answering

After the scene construction is completed, the user may switch to question answering mode and ask questions about the current scene. At any point she may choose to further describe the scene and then continue with question answering. A question is

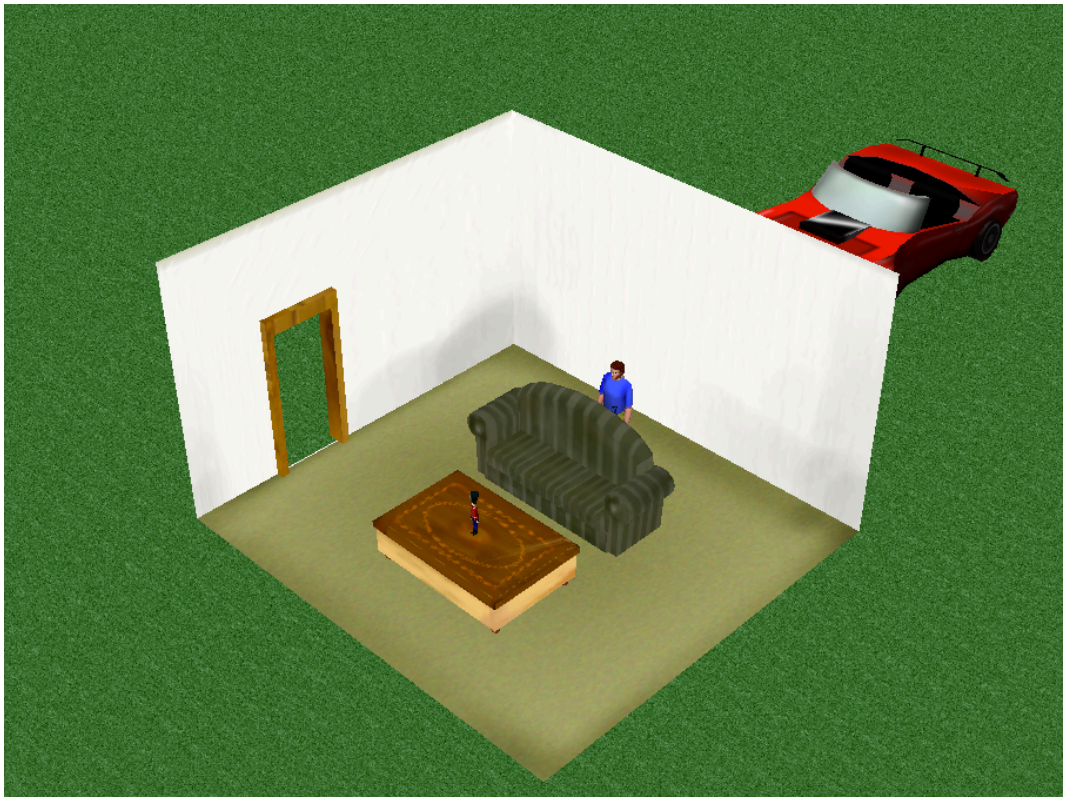

Figure 4.2: There is a room. A sofa is in the room. A table is in front of the sofa. A man is behind the sofa. A toy is on the table. A car is behind the room.

asked in natural language and based on the keywords in the question it is categorized into a specific question type.

The system can handle three types of questions. First of all, the user can directly ask about the position of an object. The system answers this question relative to other objects (landmarks) in the scene. It knows which prepositions correspond to which spatial relations and our rule based sentence builder turns these relations into sentences. It iterates over each spatial constraint in the scene and checks whether the object in question is part of that relation. If it is, then these relations are converted to natural language in the form of *X is in relation to Y*.

Secondly, another type of question a user may ask is a yes/no question. The

user may test if a certain spatial relation is present between two objects. Again the system converts prepositions in the questions to the geometric relations and checks if the target object and landmark object satisfy those criteria.

Questions	Answers
Where is the room?	It is in front of the car.
Where is the sofa?	It is in the room, in front of the man, behind the table.
Is the toy on the table?	Yes.
Is the man in the room?	Yes.
Is the car in the room?	No.
Is the sofa in front of the table?	No
Can the man see the sofa?	Yes.
Can the man see the car?	No.

Table 4.1: Question Answering Examples

Finally, a user can ask whether two objects can see each other or not. For this case, the system first considers orientation, it checks whether the tested object is in front of the other object or not. If it is not in front of the other object then it cannot be seen. If it satisfies this criteria then we draw an imaginary beam from the source object towards the target object. If it is intersected by another object in the scene, the system concludes that two objects cannot see each other. Even in very complex scenes, it is an easy task for our system to come up with the correct answers for these kinds of questions. Some sample questions and answers are shown in Table 4.4 for the scene in Figure 4.2.

Chapter 5

CONCLUSION

In this thesis, we presented a novel natural language to 3D scene conversion system. The system uses an extensive set of 3D models and a rich vocabulary to construct the scene. We have shown several techniques to determine the constraints based on the language and to construct a static 3D scene. In addition to that, a question answering module is provided. This module enables user to query the state of the scene and it is capable of solving spatial inference questions.

Although our language visualization system is in its infancy, it has promising results. It is built as a base for experimentation. All the tools we used is available for free, our code is available with a free-to-use license and all the data is available to anyone who wants to test their ideas on top of the platform. We believe that this kind of openness will invite better ideas and foster innovation on the subject. Our system consists of many different parts therefore any improvement in each individual part will result in a better language visualization system. However, the following future directions have a priority for us.

At its current stage, action related verbs are omitted in our language visualization system. As part of the future work, we plan to integrate actions using dynamic verbs. This can be done by extending our rule based engine with path planning algorithms, a physics engine and animations. It will allow us to create more complex scenes and cover an important part of the language. As these verbs will form animations that last a certain amount of time, time component will be introduced to our system. Question

answering will become much more sophisticated, as it should investigate many scenes and consider the time sequence of events.

Extending a rule based system manually is a very time consuming and an expensive process. We have two ambitious plans for the future direction of this research. The first is to implement a web based version of the system to make it accessible to a huge audience. This will make distribution easier and give us a way to collect data through the system. The second is to build a new version of the language visualization system that learns the connection between the language and the virtual world automatically. The data collected through the web application will be useful during this stage.

Even though the real world and physical interactions are a huge part of the language, they are not all of it. The framework described here is not capable of understanding ideas, feelings and abstract concepts in general. We believe that these issues can be addressed with engines similar to our work here for the physical world that can fill the gap between language and these concepts.

Appendix A

PENN TREEBANK POS TAGS

Penn Treebank Tagset			
CC	Coordinating conjunction	SYM	Symbol
CD	Cardinal number	TO	to
DT	Determiner	UH	Interjection
EX	Existential there	VB	Verb, base form
FW	Foreign word	VBD	Verb, past tense
IN	Prep. or subordinating conj.	VBG	Verb, gerund or present participle
JJ	Adjective	VBN	Verb, past participle
JJR	Adjective, comparative	VBP	Verb, non-3rd person singular present
JJS	Adjective, superlative	VBZ	Verb, 3rd person singular present
LS	List item marker	WDT	Wh-determiner
MD	Modal	WP	Wh-pronoun
NN	Noun, singular or mass	WP$	Possessive wh-pronoun
NNS	Noun, plural	WRB	Wh-adverb
NNP	Proper noun, singular	#	
NNPS	Proper noun, plural	$	
PDT	Predeterminer	”	
POS	Possessive ending	(	
PRP	Personal pronoun	)	
PRP$	Possessive pronoun	,	
RB	Adverb	.	
RBR	Adverb, comparative	:	
RBS	Adverb, superlative	“	
RP	Particle		

Table A.1: Penn Treebank POS Tags

Appendix B

STANFORD TYPED DEPENDENCIES

```
root - root
dep - dependent
  aux - auxiliary
    auxpass - passive auxiliary
    cop - copula
  arg - argument
    agent - agent
    comp - complement
      acomp - adjectival complement
      ccomp - clausal complement with internal subject
      xcomp - clausal complement with external subject
  obj - object
    dobj - direct object
    iobj - indirect object
    pobj - object of preposition
  subj - subject
    nsubj - nominal subject
      nsubjpass - passive nominal subject
    csubj - clausal subject
      csubjpass - passive clausal subject
```

```
cc - coordination

conj - conjunct

expl - expletive (expletive "there")

mod - modifier

  amod - adjectival modifier

  appos - appositional modifier

  advcl - adverbial clause modifier

  det - determiner

  predet - predeterminer

  preconj - preconjunct

  vmod - reduced, non-finite verbal modifier

  mwe - multi-word expression modifier

    mark - marker (word introducing an advcl or ccomp

  advmod - adverbial modifier

    neg - negation modifier

  rcmod - relative clause modifier

  quantmod - quantifier modifier

  nn - noun compound modifier

  npadvmod - noun phrase adverbial modifier

    tmod - temporal modifier

  num - numeric modifier

  number - element of compound number

  prep - prepositional modifier

  poss - possession modifier

  possessive - possessive modifier ('s)
```

```
  prt - phrasal verb particle

parataxis - parataxis

punct - punctuation

ref - referent

sdep - semantic dependent

  xsubj - controlling subject
```

Appendix C

SYNSET - SCALE MAP

Adjective Hint	Synset ID	Width, Height, Depth Scale
astronomical	SID-01383582-A	(3.0, 3.0, 3.0)
giant	SID-01385773-A	(2.5, 2.5, 2.5)
huge	SID-01387319-A	(1.5, 1.5, 1.5)
big	SID-01382086-A	(1.2, 1.2, 1.2)
standard	SID-02295998-A	(1.0, 1.0, 1.0)
small	SID-01391351-A	(0.8, 0.8, 0.8)
tiny	SID-01392249-A	(0.5, 0.5, 0.5)
infinitesimal	SID-01393483-A	(0.25, 0.25, 0.25)
tall	SID-02385102-A	(1.0, 1.1, 1.0)
short	SID-02386612-A	(1.0, 0.9, 1.0)
fat	SID-00986027-A	(1.3, 1.0, 1.3)
thin	SID-00988232-A	(0.8, 1.0, 0.8)

Table C.1: Synset - Scale Map For Size Related Adjectives

Appendix D

SPATIAL RELATIONS

Spatial Relation	Procedure
on	placeOn(modelA, modelB)
in	placeIn(modelA, modelB)
in front of	placeInFrontOf(modelA, modelB)
behind	placeBehind(modelA, modelB)
next to	placeBehind(modelA, modelB)
near	placeNear(modelA, modelB)
above	placeAbove(modelA, modelB)
below	placeBelow(modelA, modelB)

Table D.1: Spatial Relation and Corresponding Procedures

Pseudocodes for placement procedures

```
placeOn(modelA, modelB):
  bottom(modelA) = top(modelB)

placeIn(modelA, modelB):
 center(modelA) = center(modelB)

placeInFrontOf(modelA, modelB)
 center(modelA) = center(modelB)
 orientation(modelA) = orientation(modelB)
 amount = (1.5 * depth(B) + depth(A)) * 0.5
 moveForward(modelA, amount)
```

```
placeBehind(modelA, modelB)
 center(modelA) = center(modelB)
 orientation(modelA) = orientation(modelB)
 amount =  (1.5 * depth(B) + depth(A)) * 0.5
 moveBackward(modelA, amount)

placeNextTo(modelA, modelB)
 center(modelA) = center(modelB)
 orientation(modelA) = orientation(modelB)
 amount = (1.5 * width(B) + width(A)) * 0.5
 moveLeftOrRight(modelA, amount)

placeNear(modelA, modelB)
 center(modelA) = center(modelB)
 orientation(modelA) = orientation(modelB)
 amount = (width(B) + width(A)) * 0.5
 moveLeftOrRight(modelA, amount)

placeAbove(modelA, modelB)
 center(modelA) = center(modelB)
 orientation(modelA) = orientation(modelB)
 amount = (1.5 * height(B) + height(A)) * 0.5
 moveUp(modelA, amount)

placeBelow(modelA, modelB)
 center(modelA) = center(modelB)
 orientation(modelA) = orientation(modelB)
 amount = (1.5 *height(B) + height(A)) * 0.5
```

```
moveDown(modelA, amount)
```

BIBLIOGRAPHY

[1] Terry Winograd, "Procedures as a representation for data in a computer program for understanding natural language," 1971.

[2] Sharon Rose Clay and Jane Wilhelms, "Put: Language-based interactive manipulation of objects," *Computer Graphics and Applications, IEEE*, vol. 16, no. 2, pp. 31–39, 1996.

[3] Sylvain Dupuy, Arjan Egges, Vincent Legendre, and Pierre Nugues, "Generating a 3d simulation of a car accident from a written description in natural language: The carsim system," in *Proceedings of the workshop on Temporal and spatial information processing-Volume 13*. Association for Computational Linguistics, 2001, p. 1.

[4] Ola Åkerberg, Hans Svensson, Bastian Schulz, and Pierre Nugues, "Carsim: an automatic 3d text-to-scene conversion system applied to road accident reports," in *Proceedings of the tenth conference on European chapter of the Association for Computational Linguistics-Volume 2*. Association for Computational Linguistics, 2003, pp. 191–194.

[5] Richard Johansson, David Williams, Anders Berglund, and Pierre Nugues, "Carsim: a system to visualize written road accident reports as animated 3d scenes," in *Proceedings of the 2nd Workshop on Text Meaning and Interpretation*. Association for Computational Linguistics, 2004, pp. 57–64.

[6] Bob Coyne and Richard Sproat, "Wordseye: an automatic text-to-scene conversion system," in *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*. ACM, 2001, pp. 487–496.

[7] Bob Coyne, "Wordseye homepage," `http://www.wordseye.com/`, Jan. 2014.

[8] Bob Coyne, Richard Sproat, and Julia Hirschberg, "Spatial relations in text-to-scene conversion," in *Computational Models of Spatial Language Interpretation, Workshop at Spatial Cognition*, 2010.

[9] Bob Coyne, Owen Rambow, Julia Hirschberg, and Richard Sproat, "Frame semantics in text-to-scene generation," in *Knowledge-Based and Intelligent Information and Engineering Systems*, pp. 375–384. Springer, 2010.

[10] George A Miller, "Wordnet: a lexical database for english," *Communications of the ACM*, vol. 38, no. 11, pp. 39–41, 1995.

[11] Collin F Baker, Charles J Fillmore, and John B Lowe, "The berkeley framenet project," in *Proceedings of the 17th international conference on Computational linguistics-Volume 1*. Association for Computational Linguistics, 1998, pp. 86–90.

[12] Paul Kingsbury and Martha Palmer, "From treebank to propbank.," in *LREC*. Citeseer, 2002.

[13] Deb K Roy, "Learning visually grounded words and syntax for a scene description task," *Computer Speech & Language*, vol. 16, no. 3, pp. 353–385, 2002.

[14] Minhua Ma, *Automatic conversion of natural language to 3D animation*, Ph.D. thesis, University of Ulster, 2006.

[15] W3C, "Vrml virtual reality modeling language," `http://www.w3.org/MarkUp/VRML/`, Jan. 2014.

[16] Humanoid Animation Working Group, "H-anim," `http://h-anim.org/`, Jan. 2014.

[17] Minhua Ma and Paul Mc Kevitt, "Virtual human animation in natural language visualisation," *Artificial Intelligence Review*, vol. 25, no. 1-2, pp. 37–53, 2006.

[18] Matthew J Conway, "Alice: easy-to-learn 3d scripting for novices," 1997.

[19] Carnegie Mellon University, "Alice homepage," `http://www.alice.org`, Jan. 2014.

[20] Matthew Conway, Steve Audia, Tommy Burnette, Dennis Cosgrove, and Kevin Christiansen, "Alice: lessons learned from building a 3d system for novices," in *Proceedings of the SIGCHI conference on Human factors in computing systems*. ACM, 2000, pp. 486–493.

[21] Stephen Cooper, Wanda Dann, and Randy Pausch, "Developing algorithmic thinking with alice," in *The Proceedings of ISECON 2000*, 2000, vol. 17, pp. 506–539.

[22] Stephen Cooper, Wanda Dann, and Randy Pausch, "Alice: a 3-d tool for introductory programming concepts," in *Journal of Computing Sciences in Colleges*. Consortium for Computing Sciences in Colleges, 2000, vol. 15, pp. 107–116.

[23] Thomas Naps, Stephen Cooper, Boris Koldehofe, Charles Leska, Guido Rößling, Wanda Dann, Ari Korhonen, Lauri Malmi, Jarmo Rantakokko, Rockford J Ross,

et al., "Evaluating the educational impact of visualization," in *ACM SIGCSE Bulletin*. ACM, 2003, vol. 35, pp. 124–136.

[24] Stephen Cooper, Wanda Dann, and Randy Pausch, "Teaching objects-first in introductory computer science," in *ACM SIGCSE Bulletin*. ACM, 2003, vol. 35, pp. 191–195.

[25] George A Miller, Richard Beckwith, Christiane Fellbaum, Derek Gross, and Katherine J Miller, "Introduction to wordnet: An on-line lexical database*," *International journal of lexicography*, vol. 3, no. 4, pp. 235–244, 1990.

[26] Collin F Baker and Christiane Fellbaum, "Wordnet and framenet as complementary resources for annotation," in *Proceedings of the Third Linguistic Annotation Workshop*. Association for Computational Linguistics, 2009, pp. 125–129.

[27] Mark Alan Finlayson, "Code for java libraries for accessing the princeton wordnet: Comparison and evaluation," 2013.

[28] Stanford University, "Corenlp homepage," `http://nlp.stanford.edu/software/corenlp.shtml`, Jan. 2014.

[29] Mitchell P Marcus, Mary Ann Marcinkiewicz, and Beatrice Santorini, "Building a large annotated corpus of english: The penn treebank," *Computational linguistics*, vol. 19, no. 2, pp. 313–330, 1993.

[30] Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al., "Generating typed dependency parses from phrase structure parses," in *Proceedings of LREC*, 2006, vol. 6, pp. 449–454.

[31] Marie-Catherine De Marneffe and Christopher D Manning, "The stanford typed dependencies representation," in *Coling 2008: Proceedings of the workshop on Cross-Framework and Cross-Domain Parser Evaluation*. Association for Computational Linguistics, 2008, pp. 1–8.

[32] Marie-Catherine De Marneffe and Christopher D Manning, "Stanford typed dependencies manual," *URL http://nlp. stanford. edu/software/dependencies manual. pdf*, 2008.