ISSN 2278-3091
Volume 9, No.3, May - June 2020
International Journal of Advanced Trends in Computer Science and Engineering
Available Online at http://www.warse.org/IJATCSE/static/pdf/file/ijatcse81932020.pdf
https://doi.org/10.30534/ijatcse/2020/81932020

Generating Animations from Instructional Text

Pooja Yadav[1], Kaivalya Sathe[1], Dr. Manoj Chandak[2]
[1]Shri Ramdeobaba College of Engineering and Management, Nagpur, India, yadavpa_1@rknec.edu
[1]Shri Ramdeobaba College of Engineering and Management, Nagpur, India, satheks@rknec.edu
[2]Shri Ramdeobaba College of Engineering and Management, Nagpur, India, hodcs@rknec.edu

ABSTRACT

Generating animations from text finds application in numerous areas like screenplay writing, instructional videos, public safety, and video user manuals, etc. However, translating natural language text into animations is a challenging task. We develop a text-to-animation system which can handle any simple instructional text. We have created an NLP pipeline to extract action sequences from cooking recipes and map them to appropriate atomic actions present in the system's knowledge base. This paper explores a novel approach of linking Graphics with Natural Language Processing (NLP). Our goal is to obviate the necessity of having a large collection of stored graphics.

Key words: animations, atomic actions, grammar, instructional text, parse tree, verb-noun pairs.

1. INTRODUCTION

Generating animations from instructional text could be useful in many contexts e.g. public safety [2], instructional videos [5], video user manuals, for people who cannot read text but understand visuals, etc.

In this paper, we propose a text-to-animation generation system. Given an instructional input text, the system generates a rough animation of the text. We adopt a modular approach to solve the problem. Specifically, there are two modules in the system: (1) It deals with extracting action sequences from the text, and (2) maps these actions sequences into a 3D world using a novel technique.

The fundamental step in the process of generating animations from text is extracting action sequences from it. Extracting action sequences from text meant for human consumption is however challenging, as it requires understanding the complex contexts of actions. We restrict ourselves to simple sentences as the major focus of this paper is presenting a novel approach for animation generation. This technique obviates the necessity of having a large collection of stored graphics in the database. The system generates a rough animation of the input text which can be refined if needed by using better 3-D models.

The potential applications of our contributions are not restricted to cooking recipes. The techniques we develop can be used in other applications such as game making, public safety, etc.

In the remainder of the paper, we first review previous work related to our approach. Section 3 gives a formal definition of our text-to-animation system and presents it in detail. Section 4 focuses on evaluating the system based on a user study. Section 5 provides our conclusion and future directions.

2. RELATED WORK

A lot of work focused on generating 3D animations and 3D scenes from text has been carried out in the past. WordsEye [1] is one such system that creates 3D scenes from concise texts. It uses over 12,000 3D objects and maps them to create a complex 3D scene from a given textual description.

CarSim [2] is another system that creates animated 3D scenes of car accidents from reports written in Swedish. Our system is similar to it in the sense that, it also creates a structured representation of the input text using an information extraction module and then generates a 3D scene using a visual simulator.

SceneSeer [3] is yet another system that makes 3D scenes from natural language texts. These papers focus primarily on generating 3D scenes, but their ideas could be extended to generate 3D animations too.

CARDINAL [4] is an animation generation system that can generate 2D as well as 3D animations from movie scripts. CARDINAL too divides the task of generating animations into two phases, namely the NLP phase and the subsequent animation generation phase. [5] majorly focuses on simplifying complex scripts into simple texts and then using CARDINAL to generate animations.

Other notable works include [9] where different neural network models are reviewed for text processing and [10] where authors use machine learning methods to deal with textual information.

Existing text-to-animation systems focus on texts that contain both actions and a subject that performs the actions (as in a movie script, the actor). We majorly focus on instructional text. It contains actions and its subject is simply understood. Even though we restrict ourselves to simple instructional sentences, complex sentences can be handled by introducing a text simplification step before information extraction as proposed in [5]. [7] and [8] are other possible alternatives for text simplification.

We extract action sequences from the input text by generating a constituency-based parse tree for the text. Other techniques like deep reinforcement learning are also used for extracting action sequences efficiently [6].

3. TEXT-TO-ANIMATION SYSTEM

A modular approach is adopted for generating animations from cooking recipes instructional text. The system presently only deals with simple instructional sentences. The general overview of our approach is presented in Figure 1. The system is divided into two modules:

- **NLP Module**: Given an input text, this module first performs coreference resolution on the text, parses the resolved text to output a constituency parse tree which is further used to extract information i.e. the actions and their corresponding arguments into pre-defined action representations.
- **Animation Generation Module**: The NLP module passes on to the graphics component all the information necessary to render appropriate graphics. It generates animations based on action representations using a set of grammar rules.

2.1 NLP Module

Coreference Resolution: Input text can contain several entities that have the same referent (e.g. pronouns). To find all the expressions that refer to the same entity, coreference resolution is required. NeuralCoref python package is used which resolves pronouns to the entity they refer to in the text.
Parsing: We are interested in finding the action sequences from the text. The Coreference resolved text is parsed with the Berkeley Neural Parser which generates a constituency-based parse tree. A constituency-based parse tree represents the syntactic structure of a sentence. The root node of a constituency-based parse tree is always denoted as S, which represents the entire sentence. The tress has nodes like VP (Verb Phrase), NP (Noun Phrase), VB (Verb), NN (Singular noun), NNS (Plural noun), DT (Determiners), CC (Conjunctions), TO (To), etc.

A constituency-based parse tree is shown for Sentence 1 "Boil rice and add spices to it" in Figure 1. The tree is seen as a hierarchy of verb phrases (nodes denoted by VP).

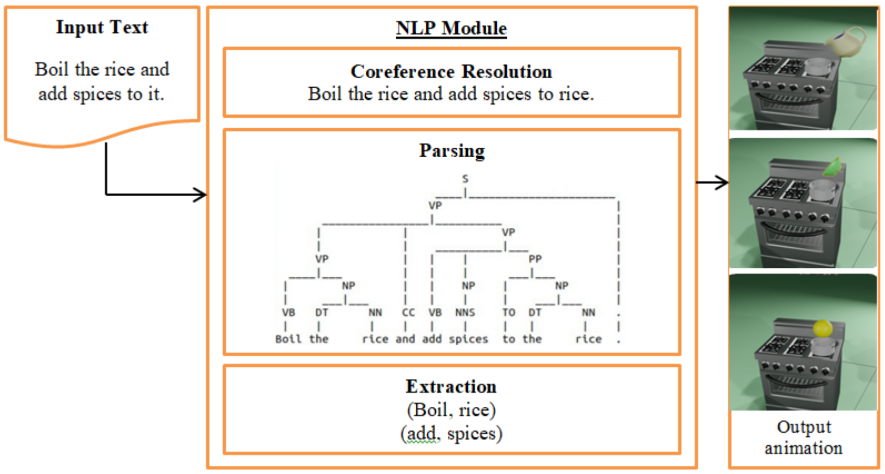
We are interested in those VP nodes whose children are a verb (VB) followed by a noun phrase (NP). A noun phrase is further made up of either a singular noun (NN) or a plural noun (NNS).

Fig. 2 shows a partial parse tree that contains a VB node followed by an NP node having NNS as its child. Fig. 3 contains a VB node followed by an NP node having NN as its child.

In Fig. 2 and Fig. 3, the blue nodes are the nodes of interest as they have a VB child followed by an NP child.

Extraction: In a typical text-to-animation system, an important step is to extract relevant information from the input text (in our case, instructional text). Cooking recipe instructions typically consist of actions (typically verbs) and their arguments (typically objects of the verb).

The extraction algorithm (Extracting Actions Procedure) starts with a constituency-based parse tree of the sentence and recursively processes it to find (action, argument) pairs and appends it to a global list of action sequences. This global list contains the extracted action sequences present in the input text. For example, consider the coreference resolved text: "Wash the tomato and cut tomato. Add tomato and spices to bowl."
The relevant information for this text is as follows:
Step 1: Wash the tomato represented as (wash, tomato).
Step 2: Cut the tomato represented as (cut, tomato).
Step 3: Add tomato to a bowl represented as (add, tomato).
Step 4: Add spices to the bowl represented as (add, spices).
All these 4 action argument pairs get stored in the global list of action sequences maintained in the Extracting Actions Procedure.
These extracted action sequences are passed to the Animation Generation module.



Input Text
Boil the rice and add spices to it.
NLP Module
Coreference Resolution
Boil the rice and add spices to rice.
Parsing
S
VP
VP
VP
PP
NP
NP
NP
VB DT NN CC VB NNS TO DT NN .
Boil the rice and add spices to the rice .
Extraction
(Boil, rice)
(add, spices)
Output animation

Figure 1: System Architecture: Input text is passed through the NLP module to generate

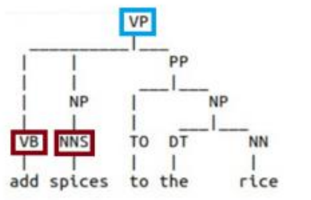


Figure 2: Constituency Tree

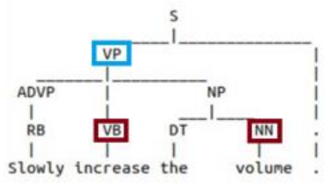


Figure 3: Constituency Tree

2.2 Animation Generation Module

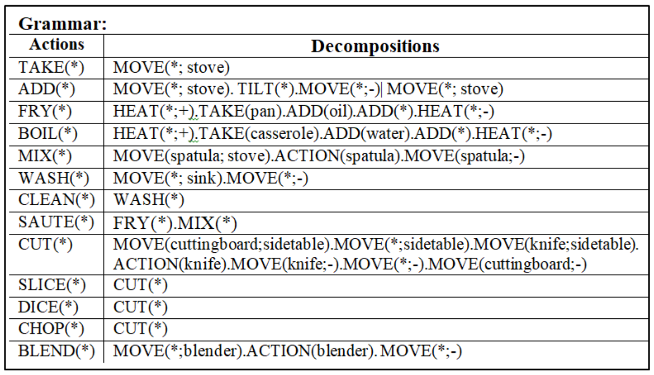
Any action performed in a 3D world can be thought of as being composed of several atomic actions. For example, a cooking recipe instruction (fry, onion) can be decomposed as (1) (heat, stove), (2) (take, pan), (3) (add, oil), and (4) (add, onion). Similarly, (boil, rice) could be decomposed as (1) (heat, stove), (2) (take, casserole), (3) (add, water), (4) (add, rice). In this paper, we restrict ourselves to cooking actions, but the idea of decomposing actions could be generalized. We build on this intuition, a set of grammar production rules for 13 different cooking actions as shown in Figure 4.

Each grammar symbol is an action. Each grammar symbol has a (*object*, *parameter*) structure associated with it. For example, in MOVE (*, stove), '*' is the *object* which represents any object and 'stove' is the *parameter*. The instruction simply means: MOVE the object represented by '*' to the 'stove'.

The left-hand side (LHS) of the grammar represents action and its right-hand side (RHS) represents its corresponding decomposition into atomic actions. The grammar symbols on RHS are concatenated by the concatenation operator (.). Moreover, if multiple productions are possible for a single action in the LHS, they are separated by ('|'). In such a case, the system intelligently chooses the appropriate production based on the (*object*, *parameter*) structure of the action.

The combinations of the atomic actions shown in Table 1, namely MOVE, TILT, ACTION, HEAT can represent complex actions in an elegant way as determined by the grammar. The grammar rules are shown in Figure 4.

```
Algorithm  Extracting Actions Procedure
Input: root node of a tree, tree
Output: list_of_actions
1.  procedure extract_actions(tree)
2.  list_of_actions ← empty list
3.        if tree is a leaf then
4.              return
5.        end if
6.        if children of tree contain VB followed by NP then
7.              let N be the noun in NP
8.              append (VB, N) to list_of_actions
9.              //VB is the action and N is its argument
10.             return
11.       else
12.             call extract_actions for each child of tree
13.       end if
```



Grammar:

Actions	Decompositions
TAKE(*)	MOVE(*; stove)
ADD(*)	MOVE(*; stove). TILT(*).MOVE(*;-)| MOVE(*; stove)
FRY(*)	HEAT(*;+).TAKE(pan).ADD(oil).ADD(*).HEAT(*;-)
BOIL(*)	HEAT(*;+).TAKE(casserole).ADD(water).ADD(*).HEAT(*;-)
MIX(*)	MOVE(spatula; stove).ACTION(spatula).MOVE(spatula;-)
WASH(*)	MOVE(*; sink).MOVE(*;-)
CLEAN(*)	WASH(*)
SAUTE(*)	FRY(*).MIX(*)
CUT(*)	MOVE(cuttingboard;sidetable).MOVE(*;sidetable).MOVE(knife;sidetable). ACTION(knife).MOVE(knife;-).MOVE(*;-).MOVE(cuttingboard;-)
SLICE(*)	CUT(*)
DICE(*)	CUT(*)
CHOP(*)	CUT(*)
BLEND(*)	MOVE(*;blender).ACTION(blender). MOVE(*;-)

Figure 4: Grammar production rules for 13 cooking actions

Notice that in the grammar, we only have a production BOIL(*) and not BOIL(rice). As '*' represents any object, rice replaces '*' in the grammar symbol representation.

The animation generation module uses the action sequences from the NLP module. For every (action, argument) pair, the action gets decomposed into atomic actions according to the grammar. We used the open-source Blender software for creating the animations for atomic actions.

The decomposition for (boil, rice) is represented in the form of a tree in Figure 5. The atomic actions are given by the leaves of the tree, read from left to right.

Blender Implementation: We used Blender software to create animations of atomic actions (Table 1). This section explains how exactly the system is implemented and how some features of blender make it possible.

Blender uses the concept of keyframes. Interpolation generates property values between keyframes. Animation of an object can be achieved by interpolation between these keyframes. For instance, suppose we want to animate the action MOVE of an object, say, casserole. If we specify the coordinates and rotation angle of the casserole at the 10th frame and then at the 20th frame, after which blender uses interpolation to determine the position and rotation angle of the casserole at all the frames between the 10th and the 20th frame.

In our animation generation system, the initial configuration

Table 1: Atomic actions and their descriptions

Grammar Symbol	Description
MOVE(object, parameter)	This action moves the object to the location specified as a parameter. If the parameter is '-', the object is moved to its initial position.
TILT(object, parameter)	This action tilts the object by 45 degrees and back to its initial orientation. This simulates the behavior of pouring any fluidic substance i.e. something that could be contained in a bowl or jug. The parameter for this action is empty.
ACTION(object,parameter)	This action executed the characteristic action associated with the object. A characteristic action of knife is moving it back and forth to simulate cutting. The parameters for this action are empty.
HEAT(object, parameter)	This action brings a flame to the stove when the parameter is '+' and puts it off when the parameter is '-'. The object is ignored in this action.

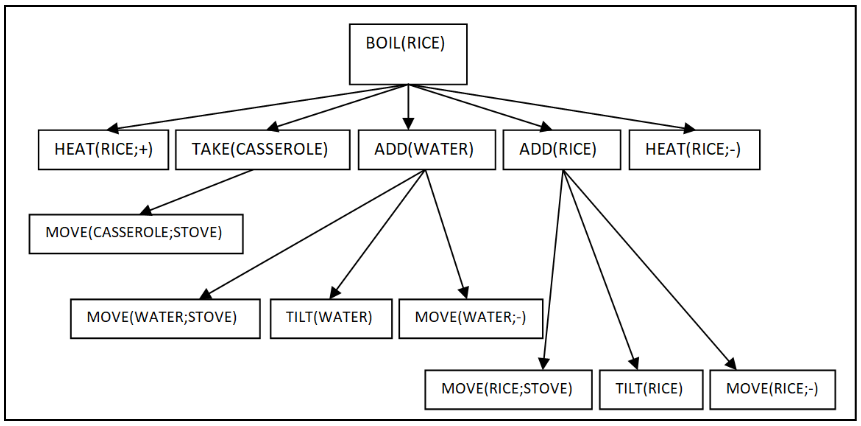


Figure 5: Decomposition tree for (boil, rice)

looks as in Figure 6.

The concept of keyframe is used to create animations for all atomic actions. For instance, MOVE action is implemented as a python function which takes as an input (1) an object from the 3D world, upon which the action has to be performed, and (2) the destination coordinates. The reference to the object is obtained from a dictionary, having the name of objects as keys and the object's initial location coordinates as values. The python function adds 3 keyframes at an interval to generate an animation of the object moving from its initial position to its final destination.

Figure 7 shows different keyframes of MOVE (Casserole; Stove).

Thus, the animation generation module first recursively expands the action object pairs obtained from the NLP module and calls the above-mentioned python function which adds keyframes in the entire animation. The end result is an eye-catching animation of the given input text.

3. EVALUATION

Evaluating a text-to-animation system is a challenging task because of two reasons: (1) there aren't any standard datasets for text-to-animation generation, and (2) no guidelines on how such systems should be evaluated and which evaluation metrics to be used exist.

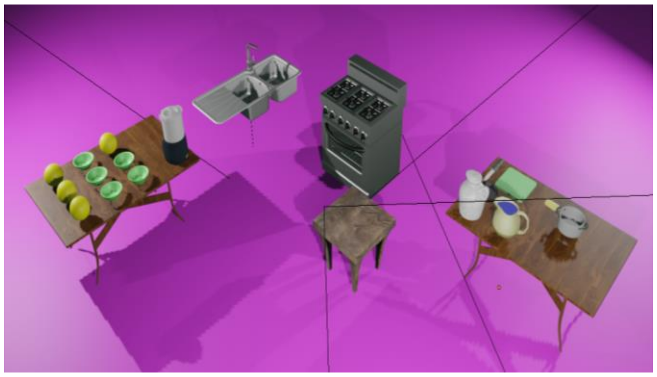


Figure 6: Initial Configuration

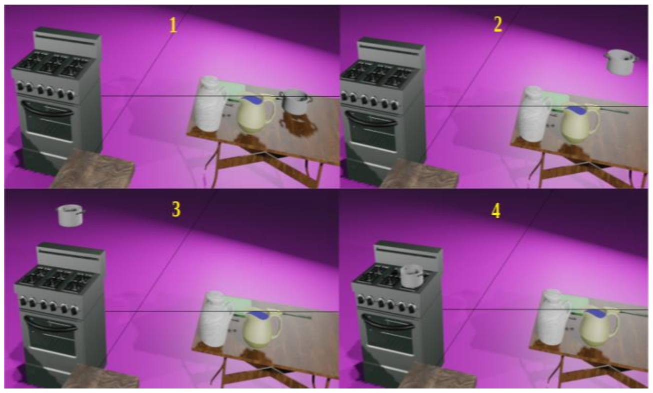


Figure 7: Different keyframes of MOVE (Casserole; Stove)

Nonetheless, it is essential to assess the performance of our system.

We conducted a qualitative evaluation via a user study to evaluate the performance of the system, similar to the study carried out in [5]. The idea was to gauge the performance of the system based on a user's perspective.

We created a set of animations for 20 different cooking instructions sentences. This set was given to 10 users. It took around 30 minutes for a user to complete the study, on average.

The users were asked to evaluate how reasonable were the animations w.r.t the input text and how much of the text information was depicted in the animations, on a five-point *Likert scale*. 76.5% of the user ratings on the overall pre-visualization were 3-point or above (figure 8).

However, it was found that the NLP module can be enhanced in some cases. For instance, in "Give the rice a good mix", "mix" would be identified as a noun instead of a verb. In a few cases where actions not existing in the grammar appear in the input text, the system simply ignores the unknown actions.

Production rules of such unknown actions need to be added to the grammar.

4. CONCLUSION AND FUTURE WORK

In this paper, we propose a novel text-to-animation system. The system decomposes extracted actions into atomic actions based on pre-defined grammar rules. Evaluating such a system is a challenge. Nonetheless, the user study shows a reasonable performance of the system. The proposed system can be improvised in many ways.

Firstly, the current system works only with relatively simple sentences. Consider a sentence "Bring the rice to a boil" which semantically means the same as "Boil the rice". The constituency parse tree generated by the Berkeley neural parser will treat "boil" as a noun. In the future, we would like to use text simplification techniques to first simplify complex sentences into simple sentences and then extract relevant information.

Secondly, for every new possible action, the user has to define a grammar production rule for it. It would be helpful if the system can automatically infer the grammar productions for a new action based on its similarity to some existing action in the grammar. Such similarity can be found out using language models like word2vec, glove, etc.

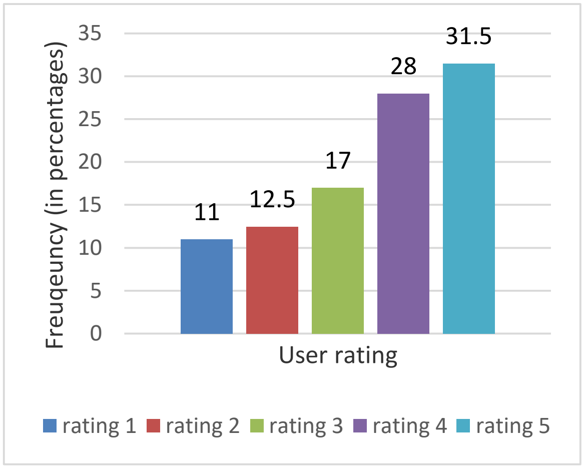


Figure 8: Graph of frequency (in percentages) versus user ratings

ACKNOWLEDGMENT

We would like to thank Mansi Radke for her valuable suggestions. We would also like to express gratitude towards Sweta Jain for her help and support.

REFERENCES

1. Coyne, Bob, and Richard Sproat. **WordsEye: an automatic text-to-scene conversion system.** In *Proceedings of the 28th annual conference on Computer graphics and interactive techniques*, pp. 487-496, August 2001.
2. Johansson, Richard, 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*, pp. 57-64, July 2004.
3. Chang, Angel X., Mihail Eric, Manolis Savva, and Christopher D. Manning. **SceneSeer: 3D scene design with natural language**, *arXiv preprint arXiv:1703.00050*, February 2017.
4. Marti, Marcel, Jodok Vieli, Wojciech Witoń, Rushit Sanghrajka, Daniel Inversini, Diana Wotruba, Isabel Simo, Sasha Schriber, Mubbasir Kapadia, and Markus Gross. **Cardinal: Computer assisted authoring of movie scripts**. In *23rd International Conference on Intelligent User Interfaces*, pp. 509-519, March 2018.
5. Zhang, Yeyao, Eleftheria Tsipidi, Sasha Schriber, Mubbasir Kapadia, Markus Gross, and Ashutosh Modi. **Generating animations from screenplays**. *arXiv preprint arXiv:1904.05440*, April 2019. https://doi.org/10.18653/v1/S19-1032
6. Feng, Wenfeng, Hankz Hankui Zhuo, and Subbarao Kambhampati. **Extracting action sequences from texts based on deep reinforcement learning**. *arXiv preprint arXiv:1803.02632*, March 2018.
7. Ferrés, Daniel, Montserrat Marimon, and Horacio Saggion. **YATS: yet another text simplifier.** In *International Conference on Applications of Natural Language to Information Systems*, pp. 335-342. Springer, Cham, June 2016.
8. Wang, Tong, Ping Chen, John Rochford, and Jipeng Qiang. **Text simplification using neural machine translation.** In *Thirtieth AAAI Conference on Artificial Intelligence*. March 2016.
9. Sheetal S. Pandya, Nilesh B. Kalani. **Review on text sequence processing with use of deep neural network model.** In *International Journal of Advanced Trends in Computer Science and Engineering*. September 2019.
10. Maganti Syamala, N. J. Nalini. **A Deep Analysis on Aspect based Sentiment Text Classification Approaches.** In *International Journal of Advanced Trends in Computer Science and Engineering*. September 2019. https://doi.org/10.30534/ijatcse/2019/01852019