Automated Simulation Creation from Military Operations Documents

John Balint, Jan M. Allbeck, Michael R. Hieb
George Mason University
Fairfax, VA
{jbalint2, jallbeck, mhieb}@gmu.edu

ABSTRACT

The creation of virtual reality simulations for training or analysis is an arduous process requiring specialized knowledge. Graphical models and even animated, articulated figures can now be obtained from websites or hired artists. Even after these assets are obtained, putting scenes together and authoring character behaviors can be a lengthy process. Furthermore, ensuring that character behaviors will be successfully performed in a virtual environment is often a trial-and-error process. Automating the creation of these behaviors and facilitating their modification by Subject Matter Experts (SMEs) – as opposed to technicians – will shorten the time required and reduce costs.

This paper presents a framework (VerbsEye) for using descriptive texts, such as military operations documents to semi automate simulation creation. While previous research, such as the WordsEye system, have created static scenes from natural language inputs, our framework further automates the process and includes the generation of agent behavior scripts from the text. Specifically, we present a text-to-scene system that generates 1) scene scripts and 2) agent behavior scripts for virtual environments. The spatial information required for the scenes is obtained both explicitly through prepositions found in the input text and implicitly from the described agent behaviors. Motion data used to depict agent behaviors is exploited to provide additional spatial constraints and assure the behaviors will be possible.

Automated scene creation is challenging and unlikely to result in perfection. The VerbsEye framework is evaluated in terms of how well sample military operations documents can be used to generate scenes and behaviors. The specific metrics used are the percentage of scenes and behaviors in the sample operation documents successfully processed. Our framework shows how additional automation can be used to enable SMEs and technicians to better and more quickly create training tools and environments.

ABOUT THE AUTHORS

Dr. Jan M. Allbeck, Ph.D. (http://cs.gmu.edu/~jallbeck) is an Associate Professor in the Department of Computer Science at George Mason University. As the faculty advisor for the Applied Computer Science degree in Game Design, Dr. Allbeck is responsible for advising, recruiting, and curriculum development. She is also the director of the Games and Intelligent Animation Laboratory at Mason. She received her Ph.D. in Computer Science from the University of Pennsylvania in 2009. Dr. Allbeck's background is in computer graphics, but is currently focusing on multidisciplinary research utilizing animation, artificial intelligence, and psychology for the simulation of virtual humans. Her Ph.D. dissertation focused on the creation and simulation of functional crowds.

John Balint is currently pursuing his Ph.D. in computer science at George Mason University. He is studying games for training and simulation, as well as intelligent animation, and has published four papers on virtual humans. John Balint received a B.S. in physics from Roanoke College in 2011. While at Roanoke College, he was an undergraduate research assistant, working on using virtual environments as a teaching tool.

Michael R. Hieb, Ph.D. is a Research Associate Professor at the George Mason University's C4I Center. He received his Ph.D. from George Mason University in 1996. Dr. Hieb is the C4I Center's Lead for its Modeling and Simulation (M&S) Technology Focus Area and has worked extensively in the area of C2 technologies to support

Decision Making. He has served as the Technical Director for the Army's Simulation to Mission Command Overarching IPT (SIMCI OIPT) and is a regular contributor to both I/ITSEC and the Simulation Interoperability Workshops and has over 120 publications.

Automated Simulation Creation from Military Operations Documents

John Balint, Jan M. Allbeck, Michael R. Hieb
George Mason University
Fairfax, VA
{jbalint2, jallbeck, mhieb}@gmu.edu

INTRODUCTION

Virtual worlds have many military applications, including training, analysis, and after action review, but their creation is a heavy burden that requires specialized knowledge. Terrain and object models must be constructed. Animated characters must be designed and created such that they can interact with the virtual environment. Then scripts or control mechanisms for the characters must be programmed. The development cycle for these simulations can be very lengthy and Subject Matter Experts (SMEs) are often marginalized during much of the process. Our aim is to automate simulation creation and 1) make it more accessible to SMEs by using Natural Language (NL) for authoring and 2) reduce the amount of time artists need to create scenes.

Military strategy and operations documents describe locations and events that arise during combat. While these could be manually, painstakingly recreated by specialized simulation authors, technology has progressed to the point that the process can now be automated. Consider this partial description on the "Battle of Aachen" (Global Security Manual Chapter 4):

> *... one platoon of Company "F," with a light machine gun section, would stage the initial diversionary attack. It would be supported by two tanks and two tank destroyers, who were instructed to shoot at all or any suspected targets. Observation posts had been manned on a slag pile to support the advance with 81-mm mortar fire...*

This small snippet contains objects, characters, and their relation to one-another. From the description a simulation author could appropriately place the objects necessary to simulate the operation. It also contains a temporal change in one of the unit's location. While existing text-to-scene frameworks could be used to setup the static virtual operation, they cannot include these dynamic elements. More subtly, the setup of the objects in the virtual scene also needs to take into account the actions that will be performed with them. For example, characters need to be able to reach objects that they will need to interact with.

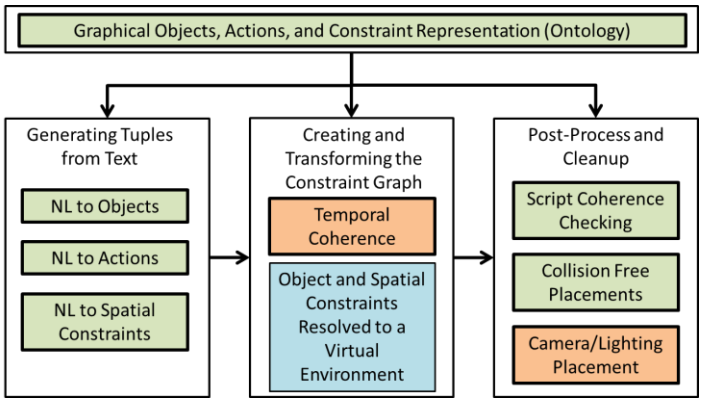

Figure 1. A diagram of components used to design a text-to-scenario system. The components depicted by green boxes have been implemented in previous systems. The blue components are addressed in this paper. The orange boxes represent important components that are future work.

In fact, there are many components necessary to convert the above excerpt into an accurate virtual scenario, as can be seen in Figure 1. While previous work on text-to-scene generation (Coyne, Rambow, Hirschberg, & Sproat, 2010) (Ma, 2006) (Hanser, Mc Kevitt, Lunney, & Condell, 2010) (Liu & Leung, 2003) examined the aspects depicted as the green boxes of the diagram, there is a large amount of, often ambiguous, information present in text that these systems do not take into account. Mainly, most other systems have not explored creating virtual simulations that include animations and respect relative relationships between several objects spread out over several sentences. One important piece of information that forms relationships between objects comes from the actions used in the text. In the above excerpt, the verb *shoot* provides not only actions for the actors to perform, but also the relative location of objects that some of the actors

use during their performance (e.g. *the targets*). It is natural for a physical human to use the relative locations of objects linked to verbs when imagining a scene, but the relationship of an action such as *shoot* can differ greatly between actors and the objects being used when constructing a virtual scene. It would be foolish to use the same relative placement for a soldier shooting at a target and a tank shooting at the same target. This becomes especially apparent in actions that require stylized animations, whose movements and areas of object interaction can vary greatly depending on the artist. If this information is not accounted for then the placement of objects and the actions used can be mismatched in the scene, which removes the benefit of automatically creating and placing objects in a scene. For example, if a character is to kick a ball during a scene, then the ball needs to be placed such that the result of the kicking motion is contact with the ball without incidental collision with other objects in the scene.

To co-create scenes and action scripts from descriptive text such as military doctrine, we create a new framework that we call *VerbsEye*[1]. The input is a descriptive text. The outputs are two scripts – one generating the visual environment (scenes) and one generating actions for agents. The framework provides:

- A text-to-scene system that generates scene graphs and agent scripts
- An ontology containing the information needed to link text to graphical models
- A method for converting relative transformation data into absolute object placements

RELATED WORK

Scene Generation

Due to the cost of manually creating large amounts of content, procedural scene generation has received a fair amount of attention in the past few years. A good survey of the topic can be found in (Hendrikx, Meijer, Van Der Velden, & Iosup, 2013). Static scenes have been generated from a variety of formats including natural language (Coyne & Sproat, WordsEye: An Automatic Text-to-scene Conversion System, 2001) (Hanser, Mc Kevitt, Lunney, & Condell, 2010) (Liu & Leung, 2003) (Chang, Savva, & Manning, 2014) and sketches (Xu, Chen, Fu, Sun, & Hu, 2013). Our work is most closely related to (Ma, 2006) (Kadir, Hashim, Wirza, & Mustapha, 2011), in which a single sentence is used to create a scene with (Ma, 2006) also creating an animation from the scene. However, our framework looks at constructing a more complex scene using several sentences based off of both the interactions between agents and their environment as well as the spatial data. Our system is also closely related to (Coyne, Rambow, Hirschberg, & Sproat, 2010) that provides prepositional frame semantics for the WordsEye system. Whereas the WordsEye system creates a static scene from one perspective, our system is designed for use in games and other free-formed simulations, which apply to multiple perspectives and dynamic scenes.

A core component of our framework is determining object placement within a scene. Hence the effort has been influenced by work on automated furniture placement. There have been several methods that create pleasing static scenes based on given heuristics, such as interior design guidelines (Merrell, Schkufza, Li, Agrawala, & Koltun, 2011) and spatial relationships (Yu, et al., 2011) (Chang, Savva, & Manning, 2014). The PUTS system (Clay & Wilhelms, 1996) also places new objects in an already developed scene using a single tuple command (i.e. *puts*). (Hodhod, Huet, & Riedl, 2014) uses a commonsense knowledge base and crowd-sourcing to learn relationships between objects in a type of scene, such as a dining room. As our framework uses text to determine spatial constraints, we only need to refer to these methods as a post-processing step to remove collisions. Furthermore, our system does not attempt to infer objects from exemplar scenes, like (Yu, et al., 2011) (Chang, Savva, & Manning, 2014), but instead requires a scenario author to specify the objects they desire.

Set Design and Spatial Relations for Actions

Our work, by parsing out transformations and actions, lends itself greatly to theatre productions. (Giannachi, Gilles, Kaye, & Swapp, 2009) created a CAVE system to train actors in theatre performances, evaluating the system with both trained and untrained actors. SceneMaker (Hanser, Mc Kevitt, Lunney, & Condell, 2010) is a text-to-scene generator specifically designed for plays that creates a corpora for emotional states in plays. (Talbot & Youngblood, 2013) also examined spatial action cues using natural language by parsing the sentences into different feature sets

[1] An homage to the WordsEye scene generation system (Coyne & Sproat, WordsEye: An Automatic Text-to-scene Conversion System, 2001).

and learning actions from them. (Porteous, Cavazza, & Charles, 2010) used planning to create timing cues for spatial cues in *The Merchant of Venice*. The latter system performs natural language processing to feed a classical planner for spatial actions for agents as does (Liu & Leung, 2003). Our method assumes input text that contains explicit actions and determines spatial relations for agents based on the cued animations similar to (Oshita, 2010). Our method extends (Oshita, 2010) by determining areas of interaction from motion data, instead of simply assigning that information to the animation.

OVERVIEW

In the following sections, we will step through our methodology for translating text into animated scenes. First we outline our perspective on actions and how they impact the spatial dynamics of a scene. Then we describe how the text is parsed and the scene is encoded by a set of logic tuples. Next we present our representation of spatial constraints, followed by a method for solving those constraints and generating the global positions and orientations of the objects and agents.

ACTION DESCRIPTIONS

A key component to transforming text into animated scenes requires understanding of how actions impact a scenario. If an agent is given an action to perform in the text, and there is no indication that the action fails in the text, then the scene must be suitably set up for the agent to complete that action with the animations available to it. For example if a character is meant to *climb* on a tank, then the agent must be close enough to *reach* the tank. This information can be inherent to an action and automatically chains to additional objects using (Li, Li, Xue, & Zhao, 2010). Inherent understandings of preconditions of actions do not properly specify an action that must always take place, but instead are conditional actions that appear only for certain agent configurations, which may or may not be desired when constructing a scene. Therefore, we currently assume that all actions are explicit, so a *walk and climb* action in the text will appear as two verbs, and encountering a *climb* action regardless will have the system automatically place the object in a *climb* interaction zone.

Actions that an agent performs can have varying impacts on the environment, such as placing an agent in a certain pose or changing the state of objects. We have found that it is important during the design phase to determine the effect that an action should have on its environment. Specifically, we categorize actions into three types: *gestural*, *event*, and *postural* actions. *Gestural* actions have no effect on the environment and can be thought of as filler actions. *Event* actions change objects in the environment and require the object to be in a certain configuration based on the agent's configuration. Finally, *postural* actions change the rotation and configuration of an agent, which can be either the dimensions of the agent's bounding volume, for actions such as *sit*, or the agent's position, for actions such as *walk*.

As *postural* actions change the state of an agent, it is important to carry this state transition forward for object placement throughout the course of the scenario. To account for this temporal change, we create marks that represent an agent's transformation at a given time. These marks behave similarly to a stage mark and are queried for where an agent is to move for a postural action. Each *postural* action in the script corresponds to a new mark for the agent, with spatial constraints being temporally aware of this postural change.

TEXT TO TUPLES

Our system eases the authorial burden of creating virtual environments by parsing natural language documents and generating plausible scenario. A large hurdle to this is to have a system determine the important information in a document, and separate that out from both the information that is not applicable to the scene and the information that cannot be graphically displayed. To overcome these issues, we implement a tuple generation system that extracts important information from natural language text.

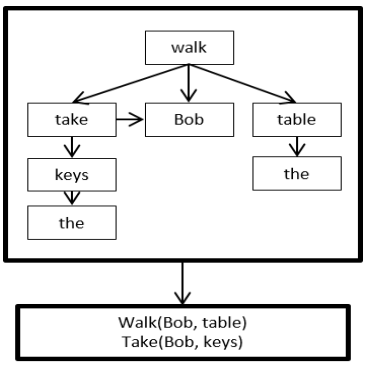

Figure 2. The top box contains the Stanford dependency tree for "Bob walks to the table and takes the keys." The lower box contains the tuples automatically produced by our framework through the application of a set of chaining rules.

Similar to other text-to-scene generation systems, we process the non-dialogue text using a dependency structure. Dialogue text is not parsed as we assume it is meant to be spoken, and therefore can be processed separately. This structure consists of a Directed Acyclic Graph (DAG), where objects are linked by their relationships in the tree. Specially, text is parsed and translated into the Stanford Dependency structure (de Marneffe, et al., 2014) (See Figure 2). Our framework then applies a set of chaining rules to collapse unneeded dependencies into a set of concise tuples, as also seen in Figure 2. In the example, the root of the tree is *walk*, which is connected to several words, some of which do not properly convey necessary scenario information, such as the connection between *walk* and *take*. Our chaining rules remove these connections and distill the dependency tree into the most vital set of relationships.

While some of these connections can be used as-is, many must be modified and chained together to create the concrete tuple connections necessary for further processing. We have found that there are two cases that require chaining: either a replacement of one variable by another or the creation of a new predicate using shared variable information. These new predicates may then be used by the chaining system to create a condensed understanding of the sentence. Therefore, our chaining system is a series of rules that are applied to each sentence and are examined until there are no new predicates added or removed. For example, in Figure 2, the relationship between *take* and *Bob* and the relationship between *take* and *keys* are combined into a single *take* tuple, which can be easily understood as both an action and spatial tuple.

We generate three types of tuples important to the display of a simulation: descriptive, action, and spatial tuples. Descriptive tuples convey properties of objects, and are generally derived from adjectives connected to the objects. Action tuples describe events executed by the actors in our scenario, which are either virtual personnel or vehicles. These tuples are generated from multiple parts of the sentence, requiring at least a subject, with the ability to condense several tuples related to the manner, duration, or objects participants into a single action. Action tuples are also used in conjunction with spatial tuples specifying the relative configuration of objects. The generated series of tuples are ultimately processed into a graphical scene and agent scripts.

The conversion from the dependency graph to tuples is a brittle process that is highly dependent on the style of writing. Text written in one tense, such as the passive tense, may not be able to be parsed if the created rules are designed for text written in the active tense. Authors of the rules need to recognize subtle style differences, but the writing of the rules themselves is not difficult.

REPRESENTATION OF SPATIAL CONSTRAINTS

Once tuple information has been extracted from a given document, our system must translate that information into a graphical representation. This depends heavily on the 3-D models and animations a simulation author has prepared and desires to use for a given text – that are stored in the VerbsEye ontologies. The competing demands of generality and specificity for varying scenes must be considered, as well as the overall accuracy of the scene to a given text.

We link tuple objects and actions to 3-D models and animations using two ontologies, one for the objects and one for the actions. The creation of these ontologies can also be automated (Pelkey & Allbeck, 2014) (Balint & Allbeck, 2015). Ontologies allow our method to connect related information together and maintain a parent/child hierarchy. By maintaining a child/parent hierarchy, more detailed descriptions can be used for objects without requiring an author to supply an object for each description. For example, if a document describes an “Abram’s Tank” and the system only has a tank model as an object, then the simulation can traverse the parent/child relationships from more specific objects to more general objects until it finds an object that is associated with a general tank model and display the general model. We chose to traverse the ontology in this manner to ensure that artifacts generated by

having too specific an object were not encountered. If it is imperative to the training parameters that a specific graphical model is used, ontologies are extendable to add this information.

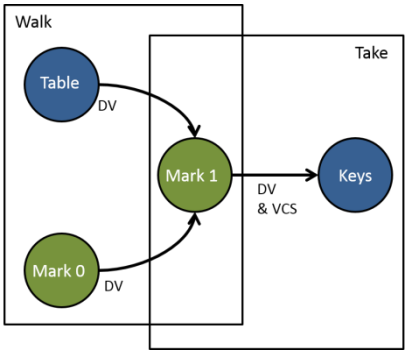

Figure 3. The generated transformation graph for the sentence "Bob walks to the table and takes the keys." Note the creation of a new stage mark based on the postural action "Walk". The relationship between the table object and this mark is a displacement vector as is the relationship to the previous mark. The forced displacement between marks ensures that a walk cycle can be performed.

Our system is also capable of commands for the characters to perform during a simulation. Action description tuples extracted from the system connect the *agent* performing a given *action* to *objects* the agent performs the action on (if any) and a relative timing system for the action. Agents can represent individual characters or entire units. To do this, our system must have an understanding of the action to be performed, including the allowable object participants and game state information that the virtual simulation must be in so the action can be performed. We create action templates, known as *Uninstantiated PARs* in the Parameterized Action Representation (Bindiganavale, et al., 2000) which contain all of the needed information to go from text to an animation. For example, a *take cover* action requires an object that can protect the agent and requires that agent to be close to the object. Actions can be simple or multi-step, complex actions. Furthermore, they can be manually composed or take advantage of task planners. While work is being done to automate the creation of these representations, at this time, a SME is responsible for the PAR representation. Information in PARs can be used to compensate for details not specified in the input text. Our system then combines tuple information by assuming that all actions of the same type in our action ontology are the same, and creating a new action if the tuple violates a constraint such as the action having two subjects or attempting to perform an action on two different objects. Next the system generates a script that contains all action information for a given scenario. We ensure that a simulation author can correct any mistakes in the script before using it in a virtual environment by generating the script as a human readable document.

In addition to resolving the objects to display and the commands to run in a simulation, spatial constraints for all objects in a scene must be determined. As natural language is inherently ambiguous, it should not be expected that all spatial constraints are accurately described by a given text. It is more likely that chains of descriptions would be provided, leading to a graph structure of transformations, such as the example provided in Figure 3. The graph provides the cornerstone for how objects in a space are placed. Nodes in our graph define objects with their associated Axis-Aligned Bounding Boxes (AABBs). Edges represent spatial constraints.

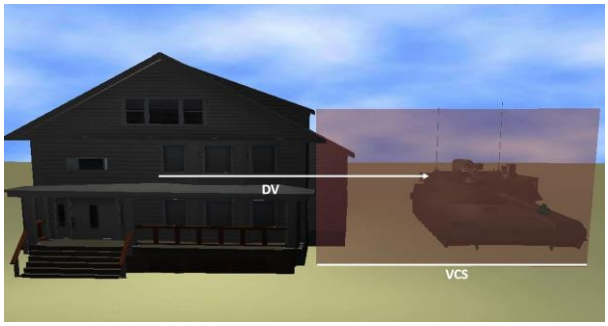

Figure 4. A scene generated from the sentence: *Armored vehicles can be positioned next to a building allowing soldiers to use the vehicle as a platform to enter a room or gain access to a roof.* For each defined spatial constraint, the system generates a Volume of Candidate Space (VCS) and Displacement Vector (DV).

Spatial constraints in our framework emanate from two sources, prepositional phrases such as *on top of* and *near* and inherent constraints between action participants, such as the constraint generated from "*Bob walks to the table"* in Figure 3. Both of these constraints do not directly explain the relationship between two objects in an absolute manner, but instead provide relative descriptions of placements. We define both of these spatial constraints similarly, as a Displacement Vector (*DV)*, and a Volume of Candidate Space (*VCS*) relative to the object model. *DV* defines the offset of the center of a *VCS* from the center of a model's AABB (See Figure 4). *VCS*s correspond to subjective volumes associated with spatial constraints. For example, Figure 4 shows the *VCS* for *next to* the building model. In addition to position, we also define a relative rotation that can take place between two objects.

A *VCS* indicates the space in which the center point of a model's AABB can exist to adhere to the spatial constraint being represented (e.g. *in front of*). In addition to the common VCSs of *point*, *cube*, *cylinder* and *sphere*, we define a *path* and *pose* transformation specifically to model changes to the environment caused by actions. A *path* VCS is comprised of a *point* VCS and an associated final rotation while a *pose* VCS is considered a change in volume and is used to capture the dimensional change of the bounding volumes of a person. Due to the variability of object sizes, we consider both *VCS* and *DV* to be in the coordinate system of the parent object in the constraint graph. This causes each transformation to be based on the size of the parent object, allowing us to generate and store both *VCS* and *DV* as general parameters, increasing the reusability of each.

The method we use to generate spatial constraints depends on if the constraint is based off an available animation for a human agent. If there is an action for a human agent, we can track movement based on user selected joints. For example, if a virtual soldier has a crouch animation, we map the change in the agent's scale as a *pose* transformation. The details of this technique are outside the scope of the paper. For spatial constraints that are not based on a humanoid agent's animation, we generate spatial constraints as a transformation and scale change between two objects in a manner similar to (Kadir, Hashim, Wirza, & Mustapha, 2011).

CREATING AND TRANSFORMING THE CONSTRAINT GRAPH

Once spatial data is generated, it can be used to layout and animate a scene from text. Recall that the input text is processed and parsed into a set of tuples. These tuples are then used to create a spatial constraint graph. Nodes in the graph correspond to objects or models. Edges are generated from constraints, with multiple edges signifying multiple constraints between the positions of the Axis Aligned Bounding Boxes (AABB) of objects. However, to determine the final global positions of each object a solution to the constraint graph must be found.

The first step in determining a solution to the generated constraint graph is to transform the graph into one or more Directed Acyclic Graphs (DAGs). We do this by performing a topological sort on the graph, which creates an ordering of objects that the transformations can propagate through. However, this structure may be over or under constrained depending on the input text. An over-constrained graph will either contain cycles or multiple, possibly conflicting constraints from one object to another (such as an object facing towards and away from its parent object at the same time). An under constrained graph will create a forest from the topological sort and can result in colliding objects that must be resolved before display.

Since a topological sort cannot be performed on a graph with cycles, the system must first create a graph without cycles. To do so, we perform a topological sort on the graph and, if a cycle is discovered, invert all transformations on one item in the cycle. An example of this can be seen in Figure 5. As all the spatial transformations in the system are linear, we can invert the transformations and edges and continue to keep an equivalent system. Note that marks are not inverted. Orientation constraints are inverted by taking the transpose of the rotation matrix and translations are simply negated. For cycles in a graph, choosing the best connection to invert can be a difficult problem. We use a least connection heuristic to determine which node in the cycle needs to be switched with ties broken in a first in order fashion.

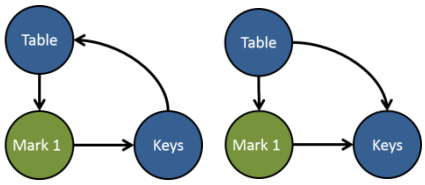

Figure 5. Two constraint graphs using the same items and depicting the same configuration. (Left) *Bob is near the table. The table is under the keys. Bob takes the keys.* (Right) *Bob is near the table and takes the keys on the table.* While both of these are valid sentences, the first one creates a loop.

At this stage, we have a topological sort of the relative spatial constraints of objects, which are described through candidate volumes (*VCSs*) that are expressed on the connections between objects. However, a virtual environment requires absolute positions and orientations for each object. We call our method for solving the constraint graph, a *Global Intersection of Volume of Candidate Spaces* (GIVCS) method. The method is able to determine the position and orientation of the AABB for each object in world coordinates, while minimizing collisions between objects.

Global Intersection of VCS

The GIVCS method performs a global minimization in order to determine the positions and orientations of all objects in two steps. By using local constraint information and a knowledge of the objects being constrained, it can determine offsets of all nodes from the origin of the virtual environment. This method first determines the rotation of each object in global coordinates. This is so that *DV* may be correctly oriented when solving for the displacement of the *VCS*. This system is solved through a least squares optimization on the difference between rotational constraints and only requires the relative rotation that is attached to the *VCS* of each constraint.

Once rotations have been defined for each object, we can determine *DV* in world coordinates as the displacement from a parent node to the center of a *VCS*, which determines a relative position for spatially dependent object as depicted in Figure 4. In the figure, the position of the tank model should lie somewhere within the shown *VCS*. As the *VCS* is connected to the house, a change in the house's position would lead to a change in the constraint for the tank. An inequality is defined for each link in the graph with multiple links to the parent object giving multiple inequalities for the node. Using this representation allows us to treat scene design as a non-linear optimization problem, where the system tries to minimize the overlap of each object's bounding box.

Minimizing over each node's bounding box provides several advantages when constructing a scene from relative information. In most text, there is not a strict constraint between most objects. If it is assumed that all objects start at the origin and are transformed, this optimization is free to move the objects far away from each other, so there is no chance of collision artifacts in the scene. Also, if the *GIVCS* method returns a result, it provides the global positions and orientations of all objects in a scene. If a solution is not found, the *GIVCS* method is able to determine that the constraints placed on the simulation are not feasible and a simulation author must make adjustments to the text in order to find a solution.

Clean-up and Post Processing

Our post-processing step is a physics step using the method proposed by (Xu, Stewart, & Fiume, 2002). It is done to determine the final positions of the objects. Using a physics processing or relaxation step removes any floating objects essentially by turning on gravity. All objects are settled onto a surface. As (Xu, Stewart, & Fiume, 2002) did not consider transformation constraints in their system, at this stage, we assume all objects are unique and do not attempt to keep constraints between objects perfectly. Since this processing step simply settles the objects and does not drastically change their placement, we assume this to generally be a valid assumption. Environments with many angled surfaces would be an exception. The end result of the VerbsEye process are two scripts – that can be imported into any visualization or game engine that can parse a plan text script or that a parser can be written for. We have written our own visualization software using the open source graphics engine OGRE and added an import option to the Unity game engine development environment.

ANALYSIS OF RESULTS

Our system can be divided into two pieces: a system to generate objects, actions, and constraints from natural language and a system to create a 3-D simulation from the generated information. Therefore, the utility of our system is based on its ability to accomplish both of these tasks. We examined four military documents, two training manuals (Shelters, 2002) (Department of the Army, 1992), and two operational documents (Department of the Army, 2002) (Department of the Army, 2001), extracting a total of nine text snippets from which to examine our system. The input texts and a sample of a result simulation can be seen at: https://cs.gmu.edu/~gaia/Creation/samples.html

We first test the ability for our system to generate tuples from natural language. Recall that our system uses a tuple generation system from dependency graphs generated by the Stanford Parser. The generation system uses chained rules to connect pieces of the graph. For our tests, we used eighteen rules for our nine text snippets. Eight of these rules differ only by one connection, such as looking for a "*subject"* or "*direct object"* and "*spatial property"* to chain together. It then connects objects, actions, and spatial information from the ontology, removing items that do not have a connection. We therefore determine the ratio of the number of tuples our system determines a simulation can use compared to the total number generated by our system, and also the ratio of correctly found tuples compared

to the total expected as determined by a simulation author. The results of these two experiments can be seen in Figure 6.

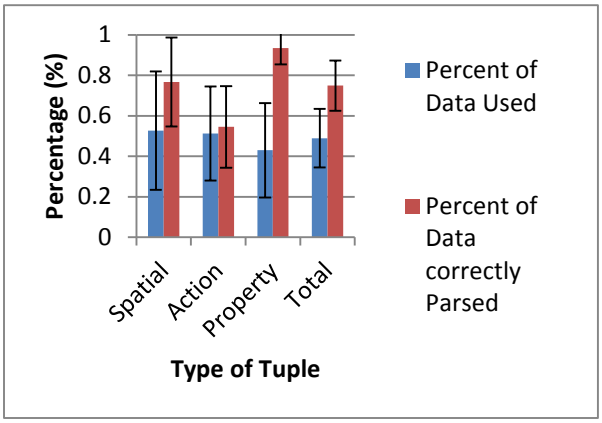

Figure 6. The percentage average versus the type of tuple examined for each of the three types of tuples as well as considering all tuples together. The error bars represent the 90% confidence interval for each of the averages.

As can be seen from Figure 6, there is a large amount of variability between the documents. However, on average, 80% of the tuples that are expected from a given document are found by our system. In general, the percentage of tuples that are useful to the system is around half of the tuples generated. An example of a tuple that is generated and not used is the property tuple "*Correctly(Pass, Window)"* While this is certainly an important property of the text, it has no bearing on how the scene should be set up or on how the animation should be displayed. By comparing the tuples with the ontologies to control the tuples processed by the system, the system has a better understanding of what is important from the text.

When examining the percent of correct tuples found, it can be seen that there is a large confidence interval for where the average between all text snippets are. A subsequent two factor ANOVA analysis on the percent correct show that there is statistical significance between the percent correct in each of the documents with a *p* value of *p=0.005*. This proves that the ability of the system to interpret a piece of text is highly dependent on the rules of the tuple generation system. If the simulation author knows what kind of text the system should receive, they can prepare specific rules to ensure good parsing. The two factor ANOVA did not find statistical significance between the types of tuples with *p=0.06* signifying that a piece of text is either correctly parsed or it is not.

Using the results of Figure 6, we tested VerbsEye's ability to design a scene. We chose five of the examples from our previous test that generated over 75% of the expected tuples. We then perform a timed experiment on how quickly acceptable scenes could be generated. The first author on this paper, who is well versed in scene design in Unity, read through the five examples and created plausible scenes using Unity's default tools. We then generated a candidate scene for each example using presented our method. These candidate scenes were cleaned up in Unity. Timing information for using our method and the post clean up portion are shown as separate times in Figure 7.

In Figure 7, the average time to complete a static scene is much less using our method than creating the scene in Unity. Even though each of the examples requires some changes in order to make the scene presentable, the difference between the total time for our method and creating a scene using ubiquitous scene design tools is statistically significant, which shows that this is a promising method to reducing the burden of creating animations from text documents.

CONCLUSIONS AND FUTURE WORK

We have presented an innovative method of text-to-scene generation that uses large sections of descriptive text to create a constrained scene with performing virtual characters. Our VerbsEye framework generates both object placements and agent performance scripts. Natural language descriptions are automatically translated into Stanford dependency trees and through chaining rules converted to tuples representing the relationships between objects and agents in the scene. These tuples are then used to generate spatial constraint graphs. Geometric information represented

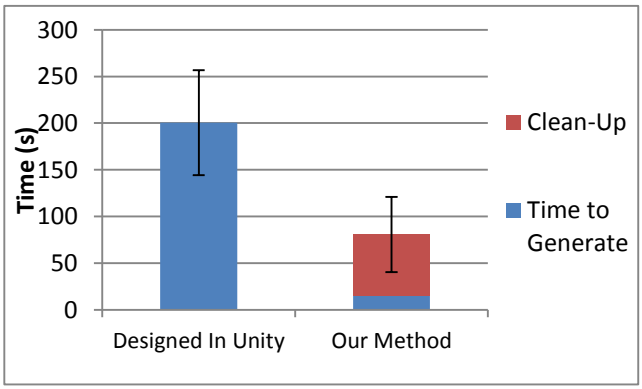

Figure 7. The average time to create an example scene compared to the method. A single factor ANOVA analysis on the two methods show a difference in the total average time between the two methods, with a *p* value of 0.003.

in the graphs includes spatial relationships between objects as described in the text, but also spatial information between the agents and objects as calculated from motion data. The latter component ensures that actions can be performed successfully and cleanly. Throughout the process, the sometimes complex, interdependencies of the various spatial relationships are represented and transformed to obtain coherent, plausible environments that can be further manipulated by artists or SMEs.

The work presented in this paper is on-going with several limitations and components still needing to be addressed (See Figure 1). One crucial limitation of our framework is that it requires the entire script to be parsed before generating a constraint graph, as opposed to being able to create a scene sentence by sentence. This forces a scenario author to write out or obtain an entire scene before the system can be executed. However, if the amount of time to parse and resolve a constraint graph can be reduced, it is conceivable that our technique can achieve interactive rates, allowing a scene designer to see and manipulate the text together to obtain the desired simulation.

Furthermore, our automated method is not perfect. Errors that occur during parsing of the natural language and generation of the tuples can be manually corrected in the resulting text files. Tools to aid naïve users with these corrections still need to be developed and tested. Certainly, users can also simply edit the input natural language to be more specific to achieve their desired results. As noted in the previous section, objects that are not placed in the most desired locations can be moved using traditional graphics tools. As objects tend to be close to their desired locations and not all objects need to be moved, our method still greatly reduces the burden of creating virtual scenarios. Also note that DVs and VCSs are specified per spatial relation, with the ability to specify constraints per object as well. Users can change their values once and have the change propagate to every instance of the object and relation in the environment, resulting in greatly reduced manual effort when many instances of the object appear in the simulation(s).

Another area of future work involves dialogue not currently processed directly in our framework. Gestures often accompany dialogue and may refer to objects or agents in the environment, potentially impacting the setup of a scene. Automatic generation of gestures from speech is ongoing work in several research groups. As a part of the extension to our framework, we plan to examine the work of (Marsella, et al., 2013) and others.

Our method can track spatial changes for stylized motions. However, when general identifiers are used to describe different styles of actions, chains of stylized actions cannot be properly created (Temporal Coherence in Figure 1). While specific action names can be used, such as *right fall* and *right kick* to create spatial coherence, future work will need to address tracking and aligning stylized changes for groups of actions in a scene. The VerbsEye framework is a working foundation on which to build additional functionality. There many opportunities ahead.

ACKNOWLEDGEMENTS

This work was partially supported by a grant from the U.S. Army Night Vision and Electronic Sensor Directorate (W15P7T-06-D-E402) and software license from Autodesk and Unity. We are also grateful for the use of ICT's SmartBody software for human character animation. Finally, we would like to acknowledge Shib Duman for the creation of 3D models used in our examples and John Mooney and Jessica Randall for their contributions in refining the presentation of this research**.**

REFERENCES

Balint, T., & Allbeck, J. M. (2015). Automated Generation of Plausible Agent Object Interactions. *Proceedings of Intelligent Virtual Agents* (p. To Appear). Springer.

Bindiganavale, R., Schuler, W., Allbeck, J. M., Badler, N. I., Joshi, A. K., & Palmer, M. (2000). Dynamically Altering Agent Behaviors Using Natural Language Instructions. *Autonomous Agents*, (pp. 293-300).

Chang, A. X., Savva, M., & Manning, C. D. (2014). Learning Spatial Knowledge for Text to 3D Scene Generation. *Proceddings of the 2014 Conference on Empirical Methods in Natural Language Processing.*

Clay, S., & Wilhelms, J. (1996, March). Put: language-based interactive manipulation of objects. *Computer Graphics and Applications, IEEE, 16*(2), 31-39.

Coyne, B., & Sproat, R. (2001). WordsEye: An Automatic Text-to-scene Conversion System. *Proceedings of the 28th Annual Conference on Computer Graphics and Interactive Techniques* (pp. 487-496). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/383259.383316

Coyne, B., Rambow, O., Hirschberg, J., & Sproat, R. (2010). Frame Semantics in Text-to-Scene Generation. *Knowledge-Based and Intelligent Information and Engineering Systems. 6279*, pp. 375-384. Springer Berlin Heidelberg.

de Marneffe, M.-C., Dozat, T., Silveira, N., Haverinen, K., Ginter, F., Nivre, J., & Manning, C. D. (2014). Universal Stanford dependencies: A cross-linguistic typology. *Proceedings of 9th International Conference on Language Resources and Evaluation*, 8.

Department of the Army. (1992). Doctrine. In D. o. Army, *Infantry Rifle Platoon and Squad* (p. 23). Washington DC.

Department of the Army. (2001). Encirclement Operations. In D. o. Army, *Tactics* (pp. D0 -- D17). Washington DC.

Department of the Army. (2002). *Combined Arms Operations in Urban Terrain.* Washington DC.

Giannachi, G., Gilles, M., Kaye, N., & Swapp, D. (2009). Mediating Performance through Virtual Agents. *Intelligent Virtual Agents*, 439-445.

Hanser, E., Mc Kevitt, P., Lunney, T., & Condell, J. (2010). SceneMaker: Intelligent Multimodal Visualisation of Natural Language Scripts. In L. Coyle, & J. Freyne (Eds.), *Artificial Intelligence and Cognitive Science* (Vol. 6206, pp. 144-153). Springer Berlin Heidelberg.

Hendrikx, M., Meijer, S., Van Der Velden, J., & Iosup, A. (2013, February). Procedural Content Generation for Games: A Survey. *ACM Trans. Multimedia Comput. Commun. Appl., 9*(1), 1:1--1:22. Retrieved from http://doi.acm.org/10.1145/2422956.2422957

Hodhod, R., Huet, M., & Riedl, M. (2014). Toward Generating 3D Games with the Help of Commonsense Knowledge and the Crowd.

Kadir, R., Hashim, A., Wirza, R., & Mustapha, A. (2011). 3D Visualization of Simple Natural Language Statement Using Semantic Description. In H. Badioze Zaman, P. Robinson, M. Petrou, P. Olivier, T. Shih, S. Velastin, & I. Nystram (Eds.), *Visual Informatics: Sustaining Research and Innovations* (Vol. 7066, pp. 36-44). Springer Berlin Heidelberg.

Li, H.-J., Li, Z., Xue, X.-P., & Zhao, T.-J. (2010, July). Event entailment chains extraction for Text-to-Scene conversion. *Machine Learning and Cybernetics (ICMLC), 2010 International Conference on, 3*, pp. 1284-1287.

Liu, Z.-Q., & Leung, K.-M. (2003, Nov). Script Visualization (ScriptVis): a smart system that makes writing fun. *Machine Learning and Cybernetics, 2003 International Conference on, 5*, pp. 2990-2995 Vol.5.

Ma, M. (2006). *Automatic Conversion of Natural Language to 3D Animation.* Ph.D. dissertation, University of Ulster.

Marsella, S., Xu, Y., Lhommet, M., Feng, A., Scherer, S., & Shapiro, A. (2013). Virtual Character Performance from Speech. *Proceedings of the 12th ACM SIGGRAPH/Eurographics Symposium on Computer Animation* (pp. 25-35). New York, NY, USA: ACM. Retrieved from http://doi.acm.org/10.1145/2485895.2485900

Merrell, P., Schkufza, E., Li, Z., Agrawala, M., & Koltun, V. (2011, July). Interactive Furniture Layout Using Interior Design Guidelines. *ACM Trans. Graph., 30*(4), 87:1--87:10. Retrieved from http://doi.acm.org/10.1145/2010324.1964982

Oshita, M. (2010). Generating animation from natural language texts and semantic analysis for motion search and scheduling. *The Visual Computer, 26*(5), 339-352.

Pelkey, C., & Allbeck, J. M. (2014). Populating Virtual Semantic Environments. *Computer Animation and Virtual Worlds Journal, 24*(3-4), 405-412.

Porteous, J., Cavazza, M., & Charles, F. (2010, Dec). Applying Planning to Interactive Storytelling: Narrative Control Using State Constraints. *ACM Trans. Intell. Syst. Technol., 1*(2), 10:1--10:21.

Shelters. (2002). In D. o. Army, *U.S. Army Field Manual 3-05.70: Survival.* Fort Bragg.

Talbot, C., & Youngblood, G. M. (2013). Lack of Spatial Indicators in Hamlet. 154-159.

Xu, K., Chen, K., Fu, H., Sun, W.-L., & Hu, S.-M. (2013). Sketch2Scene: Sketch-based Co-retrieval and Co-placement of 3D Models. *ACM Trans. Graph., 32*(4), 123:1--123:15. Retrieved from http://doi.acm.org/10.1145/2461912.2461968

Xu, K., Stewart, J., & Fiume, E. (2002). Constraint-based Automatic Placement for Scene Composition. *Proceedings of the Graphics Interface Conference*.

Yu, L.-F., Yeung, S.-K., Tang, C.-K., Terzopoulos, D., Chan, T. F., & Osher, S. J. (2011). Make It Home: Automatic Optimization of Furniture Arrangement. *ACM Trans. Graph., 30*(4), 86:1--86:12. Retrieved from http://doi.acm.org/10.1145/2010324.1964981