Using SCT Generator and Unity in Automatic Generation of 3D Scenes and Applications

Anton Kvesić, Danijel Radošević, Tihomir Orehovački
University of Zagreb, Faculty of Organization and Informatics
Pavlinska 2, 42000 Varaždin, Croatia
{anton.kvesic, danijel.radosevic, tihomir.orehovacki}@foi.hr

Abstract. *This paper deals with the automatic generation of 3D scene and related source code using the SCT based generator. Compared to current ways of 3D modeling, the proposed approach is based on textual specification and code generation. Application built this way and its 3D scenes can still be edited in a 3D modeling environment such as Unity. On the other hand, by exporting the specification from such modeling environment, novel 3D scenes and applications can be generated. This feature enables higher level of code generation flexibility thus avoiding restrictions that are common in software development. An example of the generated application is presented and discussed.*

Keywords. 3D scene, SCT generator, Unity

1 Introduction

Modeling of 3D scenes and corresponding applications is significantly facilitated with the use of software tools like Unity [17]. These tools enable 3D modeling with physical model included as well as programming in standard programming languages, like C#. In addition, some tools enable programming of program editor and application itself. On the other hand, 3D modeling is still faced with significant problems that are outcome of the 3D scene complexity.

The employment of code generators enables another, textual form of 3D scene representation that can be, as demonstrated in SCT based generator [1], adapted to the form of program specification. Separation of textual specification parts is much easier than extraction of the appropriate model parts using 3D editor which represents model elements graphically. In that respect, 3D scene could be generated from separated specification part and edited in a visual 3D editor that can export updated specification. This means that 3D scene can be edited as a textual representation as well as by using the 3D editor. Apart from the set forth, it is possible to extract just a part of textual representation, edit appropriate part of 3D scene in an editor, and update the textual specification.

Considering the general case, there is an issue of integrating generators into the software development systems because generated code should not be modified outside the generator. This paper demonstrates one possible way of solving that shortcoming in the domain of 3D modelling.

The remainder of the paper is structured as follows. Brief literature review is provided in section 2. Foundations of SCT generator model are offered in section 3. Section 4 describes the process of 3D scene generation. The example is presented in section 5. Concluding remarks are given in the last section.

2 Background to the research

The automatic generation of 3D models has found its application in various aspects of human endeavor including architecture, medicine, and military. This section offers an overview of current relevant advances in the field.

Sugihara and Kikata [16] introduced an integrated system that automatically generates 3D urban models (e.g. multi-layer buildings, residential areas, city plans, etc.) from building polygons such as ground plans or top views. By employing the proposed polygon expression together with the partitioning scheme, their system is able to generate two hundred 3D building models in less than thirty minutes.

With an objective to enhance the effectiveness of tracking systems and thereby prevent collateral damages in military operations, Witte et al. [18] proposed a concept of generating accurate 3D models of a target. The generation of 3D object models is based on the interpretation and combination of near-term laser range data and infrared images collected by reconnaissance carried out in advance.

Lim et al. [7] developed a novel method meant for the automatic generation of 3D models from a set of the training shapes in form of binary images. In comparison to prior approaches, this method works independently of the object shape, geometry, or topology.

In order to improve reliability, efficiency, and accuracy in diagnosing coronary artery diseases, Lee et

al. [6] created a method for 3D modelling of particular vessels. The method is comprised of three steps (the image acquisition, matching of the adaptive control points and the vessel warping) during which 3D models are automatically generated from angiograms that illustrate six different angles of the vessel bifurcation.

As a result of the comprehensive literature review, Azri et al. [1] identified four different approaches for automatic generation of 3D indoor models. Semantics dependent generation approach is based on the analysis of the text, interview records, and video files that contain semantic information about a building. Information fusion approach integrates semantic information (e.g. names, attributes, or states of building elements) gathered from diverse sources (e.g. CAD files, digitalized blueprints, ID tags, etc.) with geometric information (e.g. height of a building, dimensions of floors, etc.). Based on users' requirements, the third approach enables transformation of building representations in different models. In the final approach, the automatic generation of indoor models is enabled with the use of novel techniques for tracking people's motions such as gesture recognition sensors, computer vision, accelerometers in the mobile phones, mobile augmented reality, etc.

Dachselt and Rukzio [3] proposed an Extensible 3D (X3D) based declarative framework called Behavior3D meant for modeling behaviors of 3D objects. The aforementioned declarative architecture is comprised of behavior nodes which represent particular functionalities within a 3D scene, XML Schema grammar that describes interface of behavior nodes at the development level, node repertoire which denotes automatically generated integration of all available behavior nodes, and behavior node collections that refer to the groups of semantically and functionally related nodes.

Klöckner et al. [5] introduced a scripting-based technique meant for Graphics Processing Units (GPUs) run-time code generation (RTCG) that address several issues related to programming the GPUs, including the automated selection of the best code variant in terms of the predefined metric such as execution speed as well as the high-performance abstractions and cost-benefit flexibility in generating the needed number of code variants. In order to facilitate the implementation of the GPU RTCG, the authors developed two open-source toolkits. While PyOpenCL connects Python programming language with the Open Computing Language (OpenCL), PyCUDA connect Python with NVIDIA's Compute Unified Device Architecture (CUDA) parallel computing platform. Together they represent computing architecture with considerable performance and productivity in GPU RTCG.

Outcomes of the literature review indicate that generation of 3D models increasingly attracts academic attention. However, existing research addressing the generation of 3D scene is rather scarce. The set forth motivated us to initiate a research into the design of a generator that would enable automatic generation of 3D scene objects from specification. Features of the generator model which represent a backbone of our work are described in the following section.

3 SCT generator model

SCT generator model was proposed in [11] which is based on previously introduced Scripting generator model [8]. The name SCT model comes from its building elements: Specification, Configuration and Templates. The main advantage of SCT in relation to Scripting model is in separation of Configuration from generator code. This enables specification of the complete generation process using easy configuration syntax, rather than programming of Configuration manually. SCT is based on dynamic frames [11], named as SCT frames (presented in Figure 1), unlike some other frames based generator models, such as XVCL [4] and Basset's frames [2].

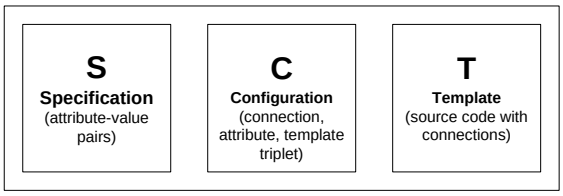

Figure 1. SCT frame [10]

The aforementioned indicates that user has to define only top-level frames, while the others are dynamically allocated during the generation process. The outcome of the generation process is SCT tree illustrated in Figure 2.

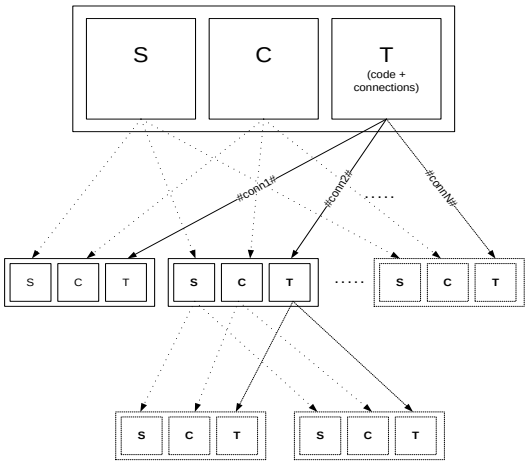

Figure 2: The generation tree [10]

Focusing on the development of the full program code, not just its backbone, the SCT model enables generation and execution of programming code on demand [9]. Furthermore, SCT is suitable for producing applications that consist of different types of

code (e.g. web applications that are comprised of snippets of code written in diverse languages such as HTML, XML, JavaScript, etc.). All parts of such heterogeneous applications can be produced from the same Specification thus achieving the high level of reusability. Up to now, the SCT model has been employed for the development of Autogenerator [14], generation of student assignments [15], determining error messages that occur at the level of generator [13], and design of a framework for building generators [12].

4 Generation of 3D scene

Regardless of which software is used, few steps should be made and experiments carried out in order to achieve the complete symbiosis between 3D modeling and generation of the 3D scene. First of all, software functionalities should be exhaustively explored in order to determine which of them are most appropriate for the generation process. The set forth does not specifically includes advanced 3D modeling capabilities but support for certain programming language and corresponding editor which represent working environment. Another important step is the employment of properties for the purpose of defining the models in a 3D space.

After software abilities are identified and 3D models are defined, connection between generator and 3D software should be established. This includes creation of common language that would be able to represent 3D models and support communication. More specifically, 3D software should be able to produce an understandable form (e.g. textual) of 3D models and present their properties to the generator. Likewise, generator could then communicate with 3D software and, for example, through the common language, change some of the 3D object's properties, create a new object, remove some of the existing objects, etc.

Last part refers to the selection of the appropriate 3D software. Considering the objective of our research and features of available software for 3D modeling, for the purpose of generating 3D scene we have chosen game development software Unity 3D [17]. The main advantages of Unity is its feature meant for the programmatic allocation of artifacts which are part of the 3D scene and support for building the 3D scene in an executable file. Apart from the set forth, Unity's editor can be programmatically managed. This feature is especially interesting in the development of generators because it enables modifications of generated 3D scene in the editor. As depicted in Figure 3, the process of generating 3D scene consists of following operations:

- Building and updating the SCT generator [12] in order to produce the necessary application and editor code for dynamic allocation of 3D objects and scene. Both application code and editor code are being generated.
- Loading the generated 3D scene into the 3D editor. Editing the 3D scene includes adding/deleting 3D artifacts as well as changing their features (position, rotation, scale, texture, etc.). The scene can be exported in a form of Specification for the purpose of the generator.
- Building an executable application. After Specification is completed, the generator produces the program code. The application can be compiled by means of the standalone compiler and without using the interface of the 3D editor.

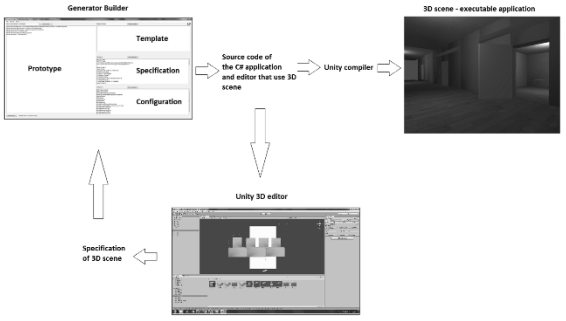

Figure 3. 3D scene editing and generation process

There are several features important for the development of SCT generator aimed for building of 3D scene based applications:

- Generator is meant for producing 3D scene from previously prepared 3D artefacts.
- The behavior of 3D objects and scene, as well as other application features, is defined in program Templates which can be applied to different scenes.
- The Specification consists of attributes that specify main objects and with them associated child objects. It is possible to extract only selected main/child objects for the purpose of editing in 3D editor or producing the executable application. This feature can be used for decreasing the complexity of 3D model while editing.

5 Unity 3D

Previously mentioned tool and software chosen for creation and building the 3D scenes – Unity, is most commonly described as a cross-platform game development software or game engine. It has built-in support for three programming languages – C#, JavaScript and Boo. Regarding 3D scenes and application view, Unity offers Scene view and Game view, which can be described as editor (Scene view) and application (Game view). The working environment of the Unity 3D editor is presented in Figure 4.

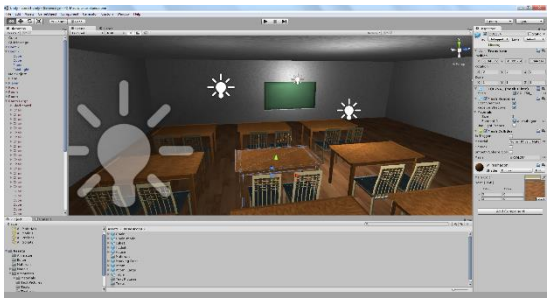

Figure 4. Unity 3D editor's interface

One of the most interesting features regarding Unity and its use with our generator is a programmable built-in editor. Programming code execution in Unity is possible not only during application runtime, but also during scenes editing inside Unity's projects. This allows us scene generation during application runtime, i.e. its start and during scene editing, which can be very useful because we do not have to start our application every time we want to make changes to our scene and update it from Specification. Likewise, it is possible to update Specification in real-time and save every changes we made during scene's editing. Lastly, there is another Unity's feature that allows adding custom menu items which are mapped to specific static methods. As illustrated in Figure 5, these methods can then be executed with a single click on the custom defined menu item.

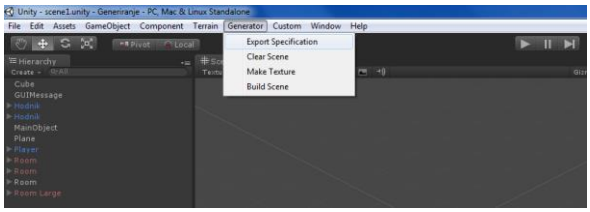

Figure 5. Custom menu items in Unity

6 An Example

An example[1] of 3D scene generator that will be presented in this paper includes previously mentioned approaches related to Unity and generator.

First of all, basic 3D scene consists of various 3D models which are represented in the 3D space through their properties, like position, rotation, size, etc. Our example shows the 3D scene in which 3D objects are generated according to predefined properties, which are forwarded to Unity through Specification. On the other hand, Unity can also export chosen 3D object's properties, i.e. export Specification, which creates connection and circle between generator and Unity.

Every 3D object in the scene is represented by corresponding part in Specification, but it is worth mentioning that most of the objects share basic properties (position, rotation, size, texture, etc.), which are considered as most important and fundamental for the purpose of representation and manipulation of 3D objects.

Figure 6 illustrates the interaction between generator and Unity, where Specification plays most important part. 3D objects that we want to include in the 3D scene are given in Specification and then forwarded to Unity using code generation. These 3D objects are previously embedded into Unity in order to be used in a scene Specification.

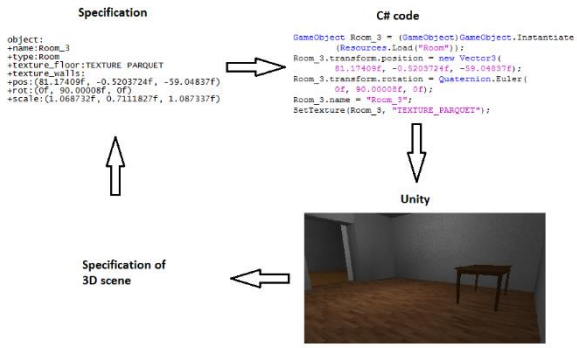

Figure 6. Communication between generator and Unity

Unity then applies generated C# code which includes methods like the one presented in the Figure 6. In this specific example, that method is Instantiate and it is used for the creation of new objects in a 3D scene. The result of the process presented in the Figure 6 are 3D objects populated through the scene which are determined by their position, rotation, size, texture and any other property included in the Specification. This is a simple example and demonstration of possibilities and ways of using Unity for generation and creation of 3D scenes.

Interaction between generator and Unity goes both ways, i.e. it is possible to generate Specification from given 3D scene which can then be used for generation of similar or even drastically changed scenes. This can also be achieved using Unity and tags that are one of its features. Tags can be used to represent and identify one or multiple game objects inside Unity. They are more often used for identifying multiple objects so they can easily be accessed and manipulated. Using tags in the specific example gives many possibilities. For example, it is possible to assign the same tag to the multiple objects that are included in Specification and simply ignore other objects and parts of the scene. Unity can then export Specification which consists only of chosen objects and can be used to recreate, change and manipulate given 3D scene in different ways and approaches.

[1] An example is available at http://gpml.foi.hr/Generator_3D/

Another important and interesting feature is real-time generation of 3D objects. For example, one floor of a building or a room inside a hallway can be generated and populated in a given time and thus provide dynamic and changeable environment. Likewise, it is possible to remove one part of the 3D scene given by Specification and consider saving the resources or temporarily reducing the load. Generation of the 3D scene does not necessarily have to imply building and creation of whole environment, but also manipulating and managing its parts.

Example described in this part of the paper also includes movement and interactions with the environment, but these functionalities are not covered by Specification itself. They can be, however, manipulated and conditioned through Specification and that is something worth investigating and studying in the future.

7 Conclusion

The objective of this paper was twofold: to introduce a new approach in building the 3D scenes based on the SCT generator and to demonstrate how to overcome limitations of integrating source code generators in the development of software systems. Drawing on the SCT model, the 3D scene can be modified by means of both 3D editor and textual Specification for the generator. Furthermore, the employment of the SCT generator reduces the complexity of the 3D scene during its modification in the 3D editor.

The example of generating the 3D scene clearly illustrated an interplay between the SCT based generator and Unity 3D editor. The textual Specification enables building and managing the 3D scene. The set forth includes representation of Unity's editor as well as executable application that deals with the 3D scene. Some other elements of the application, like movements and interactions with the environment are included in code Templates, but not in the Specification itself.

Our future research efforts will be focused on the development of a method for building the 3D scenes together with associated interactions, scenarios and objects, by means of source code generators.

References

[1] Azri, S.; Isikdag, U.; Rahman, A. A. Automatic Generation of 3D Indoor Models: Current State of the Art and New Approaches. In *International Workshop on Geoinformation Advances*, Johor, Malaysia, 2012, http://www.academia.edu/2604376/Automatic_Generation_of_3D_Indoor_Models_Current_State_of_the_Art_and_New_Approaches, downloaded April 4th 2014.

[2] Bassett, P.G.: Framing software reuse - lessons from real world. Prentice Hall, Upper Saddle River, NJ, USA, 1997.

[3] Dachselt, R., Rukzio, E.: Behavior3D: an XML-based framework for 3D graphics behavior. In *Proceedings of the 8th international conference on 3D Web technology*, pages 101-112, St. Malo, France, 2003.

[4] Jarzabek, S., Bassett, P., Zhang, H., Zhang, W.: XVCL: XML-based variant configuration language. In *Proceedings of the 25th International Conference on Software Engineering*, pages 810-811, Los Alamitos, CA, USA, 2003.

[5] Klöckner, A., Pinto, N., Lee, Y., Catanzaro, B., Ivanov, P., Fasih, A.: PyCUDA and PyOpenCL: A scripting-based approach to GPU run-time code generation. *Parallel Computing*, 38(3):157–174, 2012.

[6] Lee, N-Y.; Lee, J-J.; Kim, G-Y.; Choi, H-I. Automatic 3D Model Generation based on a Matching of Adaptive Control Points. In *You, K. (Ed.) Adaptive Control*. InTech, Shanghai, China, 2009, http://www.intechopen.com/books/adaptive_control/automatic_3d_model_generation_based_on_a_matching_of_adaptive_control_points__, downloaded April 4th 2014.

[7] Lim, S-J.; Udupa, J. K.; Souza, A.; Jeong, Y-Y.; Ho, Y-S.; Torigian, D. A. A New, General Method of 3D Model Generation for Active Shape Image Segmentation. In *SPIE Proceedings 6144, Medical Imaging 2006: Image Processing*, http://dx.doi.org/10.1117/12.653751, downloaded April 4th 2014.

[8] Magdalenić, I., Radošević, D., Skočir, Z.: Dynamic Generation of Web Services for Data Retrieval Using Ontology. *Informatika*, 20(3): 397-416, 2009.

[9] Magdalenić, I., Radošević, D., Orehovački, T.: Autogenerator: Generation and Execution of Programming Code on Demand. *Expert Systems with Applications*, 40(8): 2845-2857, 2013.

[10] Radošević, D., Magdalenić, I.: Python Implementation of Source Code Generator Based on Dynamic Frames. In *Proceedings of the 34th International Convention on Information and Communication Technology, Electronics and Microelectronics*, pages 369-374, Opatija, Croatia, 2011.

[11] Radošević, D., Magdalenić, I.: Source Code Generator Based on Dynamic Frames. *Journal of Information and Organizational Sciences*, 35(2): 73–91, 2011.

[12] Radošević, D., Magdalenić, I., Orehovački, T.: Building process of SCT generators. In Proceedings of the 36th International Convention on Information and Communication Technology, Electronics and Microelectronics, pages 1037-1042, Opatija, Croatia, 2013.

[13] Radošević, D., Magdalenić, I., Orehovački, T.: Error Messaging in Generative Programming. In *Proceedings of the 22nd Central European Conference on Information and Intelligent Systems*, pages 181-186, Varaždin, Croatia, 2011.

[14] Radošević, D., Orehovački, T., Magdalenić, I.: Towards Software Autogeneration. In *Proceedings of the 35th International Convention on Information and Communication Technology, Electronics and Microelectronics*, pages 1076-1081, Opatija, Croatia, 2012.

[15] Radošević, D., Orehovački, T., Stapić, Z.: Automatic On-line Generation of Student's Exercises in Teaching Programming. In *Proceedings of the 21st Central European Conference on Information and Intelligent Systems*, pages 87-93, Varaždin, Croatia, 2010.

[16] Sugihara, K.; Kikata, J. Automatic Generation of 3D Building Models from Complicated Building Polygons. *Journal of Computing in Civil Engineering*, 27(5):476-488, 2013.

[17] Unity3D, Unity - Game Engine, http://unity3d.com/, downloaded: May 5th 2014.

[18] Witte, C.; Armbruster, W.; Jäger, K. Automatic generation of 3D models from real multisensor data. In *Proceedings of the 11th International Conference on Information Fusion*, pages 1823-1828, Cologne, Germany, 2008.